Kernel Planet - Latest News

- Brendan Gregg: On "AI Brendans" or "Virtual Brendans" (2025/11/27 13:00)
  There are now multiple AI performance engineering agents that use or are trained on my work. Some are helper agents that interpret flame graphs or eBPF metrics, sometimes privately called AI Brendan; others have trained on my work to create a Virtual Brendan that claims it can tune everything just like the real thing. It sounds like my brain has been uploaded to the cloud by someone else who is now selling it (yikes!). I've been told it's even "easy" to do this thanks to all my publications available to train on: >90 talks, >250 blog posts, >600 open source tools, and >3000 book pages. Are people allowed to sell you, virtually? And am I the first individual engineer to be AI'd? (There is a 30-year-old precedent for this, which I'll get to later.) This is an emerging subject, with lots of different people, objectives, and money involved. Note that this is a personal post about my opinions, not an official post by my employer, so I won't be discussing internal details about any particular project. I'm also not here to recommend you buy any in particular. Summary There are two types: AI agents. I've sometimes heard them called an AI Brendan because it does Brendan-like things, like systems performance recommendations and interpretation of flame graphs and eBPF metrics. There are already several of these and this idea in general should be useful. Virtual Brendan can refer to something not just built on my work, but trained on my publications to create a virtual me. These would only automate about 15% of what I do as a performance engineer, and will go out of date if I'm not training it to follow industry changes. Pricing is hard, in-house is easier. With a typical pricing model of $20 per instance per month, customers may just use such an agent on one instance and then copy-and-paste any tuning changes to their entire fleet. There's no practical way to keep tuning changes secret, either. These projects are easier as internal in-house tools. Some claim a lot but do little. There's no Brendan Gregg benchmark or empirical measurement of my capability, so a company could claim to be selling a virtual Brendan that is nothing more than a dashboard with a few eBPF-based line charts and a flame graph. The few times I've given suggestions to projects, my ideas have been considered too hard or a low priority. Which leads me to believe that some aren't trying to make a good product -- they're in it to make a quick buck. There's already been one expensive product failure, but I'm not concluding that the idea is bad and the industry will give up. Other projects already exist. I’m not currently involved with any of these products. We need AI to help save the planet from AI. Performance engineering gets harder every year as systems become more complex. With the rising cost of AI datacenters, we need better performance engineering more than ever. We need AI agents that claim a lot and do a lot. I wish the best of luck to those projects that agree with this mantra. Earlier uses of AI Before I get into the AI/Virtual Brendans, yes, we've been using AI to help performance engineering for years. Developers have been using coding agents that can help write performant code. And as a performance engineer, I'm already using ChatGPT to save time on resarch tasks, like finding release notes and recent developments for a given technology. I once used ChatGPT to find and old patch sent to lkml, just based on a broad description, which would otherwise take hours of trial-and-error searches. I keep finding more ways that ChatGPT/AI is useful to me in my work. AI Agents (AI Brendans) A common approach is to take a CPU flame graph and have AI do pattern matching to find performance issues. Some of these agents will apply fixes as well. It's like a modern take on the practice of "recent performance issue checklists," just letting AI do the pattern matching instead of the field engineer. I've recently worked on a Fast by Friday methodology: where we engineer systems so that performance can be root-cause analyzed in 5 days or less. Having an AI agent look over flame graphs, metrics, and other data sources to match

previously seen issues will save time and help make Fast by Friday possible. For some companies with few or no performance engineers, I'd expect matching previously seen issues should find roughly 10-50% performance gains. I've heard some flame graph agents privately referred to as an "AI Brendan" (or similar variation on my name) and I guess I should be glad that I'm getting some kind of credit for my work. Calling a systems performance agent "Brendan" makes more sense than other random names like Siri or Alexa, so why not. I've also suspected this day would come ever since I began my performance career (more on this later). Challenges: Hard to quantify and sell. What the product will actually do is unknown: maybe it'll improve performance by 10%, 30%, or 0%. Consider how different this is from other products where you need a thing, it does the thing, you pay for it, the end. Here you need a thing, it might do the thing but no one can promise it, but please pay us money and find out. It's a challenge. Free trials can help, but you're still asking for engineering time to test something without a clear return. This challenge is also present for building in-house tools: it's likewise hard to quantify the ROI. The analysis pricing model is hard. If this is supposed to be a commercial product (and not just an in-house tool) customers may only pay for one server/instance a month and use that to analyze and solve issues that they then fix on the entire fleet. In a way, you're competing with an established pricing model in this space: performance consultants (I used to be one) where you pay a single expert to show up, do analysis, suggest fixes, and leave. Sometimes that takes a day, sometimes a week, sometimes longer. But the fixes can then be used on the entire fleet forever, no subscription. The tuning pricing model is harder. If the agent also applies tuning, can't the customer copy the changes everywhere? At least one AI auto-tuner initially explored solving this by keeping the tuning changes secret so you didn't know what to copy-and-paste, forcing you to keep running and paying for it. A few years ago there was a presentation about one of these products with this pricing model, to a room of performance engineers from different companies (people I know), and straight after the talk the engineers discussed how quickly they could uncover the changes. I mean, the idea that a company is going to make some changes to your production systems (including at the superuser level) without telling you what they are changing is a bit batty anyway, and telling engineers you're doing this is just a fun challenge, a technical game of hide and seek. Personally I'd checksum the entire filesystem before and after (there are tools that do this), I'd trace syscalls and use other kernel debugging facilities, I'd run every tool that dumped tunable and config settings and diff it to a normal system, and that's just what comes to mind immediately. Or maybe I'd just run their agent through a debugger (if their T&Cs let me). There are so many ways. It'd have to be an actual rootkit to stand half a chance, and while that might hide things from file system and other syscalls, the weird kernel debuggers I use would take serious effort to disguise. It may get blamed for outages. Commercial agents that do secret tuning will violate change control. Can you imagine what happens during the next company-wide outage? "Did anyone change anything recently?" "We actually don't know, we run a AI performance tuning agent that changes things in secret" "Uh, WTF, that's banned immediately." Now, the agent may not be responsible for the outage at all, but we tend to blame the thing we can't see. Shouldn't those fixes be upstreamed? Let's say an agent discovers a Java setting that improves performance significantly, and the customer's engineers figure this out (see previous points). I see different scenarios where eventually someone will say "we should file a JVM ticket and have this fixed upstream." Maybe someone changes jobs and remembers the tunable but doesn't want to pay for the agent, or maybe they feel it's good for the Java community, or maybe it only works on some hardware (like Intel) and that hardware vendor finds out and wants it upstreamed as a competitive edge. Over time these changes may get upstreamed and the agent's effectiveness decays. The effort to build. (As is obvious) there's challenging work to build orchestration, the UI, logging, debugging, security, documentation, and support for different targets (runtimes, clouds). That support will need frequent updates. For customers: AI-outsourcing your performance thinking may leave you vulnerable. If a company spends less on performance engineers as it's considered AI'd, it will reduce the company's effective "performance IQ." I've already seen

an outcome: large companies that spend tens of millions on low-featured performance monitoring products, because they don't have in-house expertise to build something cheaper and better. This problem could become a positive feedback loop where fewer staff enter performance engineering as a profession, so the industry's "performance IQ" also decreases. So it's easier to see this working as an in-house tool or an open source collaboration, one where it doesn't need to keep the changes secret and it can give fixes back to other upstream projects. Virtual Brendans Now onto the sci-fi-like topic of a virtual me, just like the real thing. Challenges: My publications are an incomplete snapshot, so you can only make a partial virtual Brendan (at some tasks) that gets out of date quickly. I think this is obvious to an engineer but not obvious to everyone. Incomplete: I've published many blog posts (and talks) about some performance topics (observability, profiling, tracing, eBPF), less on others (tunining, benchmarking), and nearly nothing on some (distributed tracing). This is because blogging is a spare time hobby and I cover what I'm interested in and working on, but I can't cover it all, so this body of published knowledge is incomplete. It's also not as deep as human knowledge: I'm summarizing best practices but in my head is every performance issue I've debugged for the past 20+ years. Books are different because in Systems Performance I try to summarize everything so that the reader can become a good performance engineer. You still can't make a virtual Brendan from this book because the title isn't "The Complete Guide to Brendan Gregg.". I know that might be obvious, but when I hear about Virtual Brendan projects discussed by non-engineers like it really is a virtual me I feel I need to state it cleary. Maybe you can make a good performance engineering agent, but consider this: my drafts get so big (approaching 2000 pages) that my publisher complains about needing special book binding or needing to split it into volumes, so I end up cutting roughly half out of my books (an arduous process) and these missing pages are not training AI. Granted, they are the least useful half, which is why I deleted them, but it helps explain something wrong with all of this: The core of your product is to scrape publications designed for human attention spans -- you're not engineering the best possible product, you're just looking to make a quick buck from someone else's pre-existing content. That's what annoys me the most: not doing the best job we could. (There's also the legality of training on copyrighted books and selling the result, but I'm not an expert on this topic so I'll just note it as another challenge.) Out of date: Everything I publish is advice at a point in time, and while some content is durable (methodologies) other content ages fast (tuning advice). Tunables are less of a problem as I avoid sharing them in the first place, as people will copy-n-paste them in environments where they don't make sense (so tunables is more of an "incomplete" problem). The out-of-date problem is getting worse because I've published less since I joined Intel. One reason is I've been mentally focused on an internal strategy project. But there is another, newer reason: I've found it hard to get motivated. I now have this feeling that blogging means I'm giving up my weekends, unpaid, to train my AI replacement. So far these AI agents only automate a small part of my job. The analysis, reporting, and tuning of previously seen issues. It's useful, but to think those activities alone are an AI version of me is misleading. In my prior post I listed 10 things a performance engineer did (A-J), and analysis & tuning is only 2 out of 10 activities. And it's only really doing half of analysis (next point), so 1.5/10 is 15%. Half my analysis work is never-seen-before issues. In part because seen-before issues are often solved before they reach me. A typical performance engineer will have a smaller but still significant portion of these issues. That's still plenty of issues where there's nothing online about it to train from, which isn't the strength of the current AI agents. ”Virtual Brendan" may just be a name. In some cases, referring to me is just shorthand for saying it's a systems-performance-flame-graphs-ebpf project. The challenge here is that some people (business people) may think it really is a virtual me, but it's really more like the AI Brendan agent described earlier. I don't know everything. I try to learn it all but performance is a vast topic, and I'm usually at large companies where there are other teams who are better than I am at certain areas. When I worked at Netflix they had a team to handle distributed tracing, so I didn't have to go deep on the topic myself, even though it's important. So a Virtual Brendan is useful for a lot of

things but not everything. Some Historical Background The first such effort that I'm aware of was "Virtual Adrian" in 1994. Adrian Cockcroft, a performance engineering leader, had a software tool called Virtual Adrian that was described as: "Running this script is like having Adrian actually watching over your machine for you, whining about anything that doesn't look well tuned." (Sun Performance and Tuning 2nd Ed, 1998, page 498). It both analyzed and applied tuning, but it wasn't AI, it was rule-based. I think it was the first such agent based on a real individual. That book was also the start of my own performance career: I read it and Solaris Internals to see if I could handle and enjoy the topic; I didn't just enjoy it, I fell in love with performance engineering. So I've long known about virtual Adrian, and long suspected that one day there might be a virtual Brendan. There have been other rule-based auto tuners since then, although not named after an individual. Red Hat maintains one called TuneD: a "Daemon for monitoring and adaptive tuning of system devices." Oracle has a newer one called bpftune (by Alan Maguire) based on eBPF. (Perhaps it should be called "Virtual Alan"?) Machine learning was introduced by 2010. At the time, I met with mathematicians who were applying machine learning to all the system metrics to identify performance issues. As mathematicians, they were not experts in systems performance and they assumed that system metrics were trustworthy and complete. I explained that their product actually had a "garbage in garbage out" problem – some metrics were unreliable, and there were many blind spots, which I have been helping fix with my tools. My advice was to fix the system metrics first, then do ML, but it never happened. AI-based auto-tuning companies arrived by 2020: Granulate in 2018 and Akamas in 2019. Granulate were pioneers in this space, with a product that could automatically tune software using AI with no code changes required. In 2022 Intel acquired Granulate, a company of 120 staff, reportedly for USD$650M, to boost cloud and datacenter performance. As shared at Intel Vision, Granulate fit into an optimization strategy where it would help application performance, accomplishing for example "approximately 30% CPU reduction on Ruby and Java." Sounds great! As Intel's press release described, Granulate was expected to lean on Intel's 19,000 software engineers to help it expand its capabilities. The years that followed were tough for Intel in general. Granulate was renamed "Intel Tiber App-Level Optimization." By 2025 the entire project was reportedly for sale but, apparently finding no takers, the project was simply shut down. An Intel press release stated: "As part of Intel's transformation process, we continue to actively review each part of our product portfolio to ensure alignment with our strategic goals and core business. After extensive consideration, we have made the difficult decision to discontinue the Intel Tiber App-Level Optimization product line." As it happened, Granulate had been acquired just before I joined Intel. I was told that their product was entirely based on my work, using flame graphs for code profiling and my publications for tuning. There was also a lot of infrastructure code for safe orchestration of tuning changes, which is not an easy problem. Flame graphs were the central interface: the first time I saw the product they wanted to highlight their dynamic flame graphs thinking I hadn't seen them before, but I recognized them as d3-flame-graphs that Martin Spier and I created at Netflix. It was a bit dizzying to think that my work had just been "AI'd" and sold for $650M, but I wasn't in a position to complain since it was now a project of my employer. But it was also exciting, in a sci-fi kind of way, to think that an AI Brendan could help tune the world, sharing all the solutions I'd previously published so I didn't have to repeat them for the umpteenth time. It would give me more time to focus on new stuff. The most difficult experience I had wasn't with the people building the tool: they were happy I joined Intel (I heard they gave the CTO a standing ovation when he announced it) and they were happy to have any ongoing help I could provide. I also recognized that automating my prior tuning for everyone would be good for the planet. The difficulty was with others on the periphery (business people) who were not directly involved and didn't have performance expertise, but were gung ho on the idea of an AI performance engineering agent. Specifically, a Virtual Brendan that could be sold to everyone. I (human Brendan and performance expert) had no role or say in these ideas, as there was this sense of: "now we've copied your brain we don't need you anymore, get out of our way so we can sell it." This

was the only time I had concerns about the impact of AI on my career. It wasn't the risk of being replaced by a better AI, it was being replaced by a worse one that people think is better, and with a marketing budget to make everyone else think it's better. Human me wouldn't stand a chance. 2025 and beyond: As an example of an in-house agent, Uber has one called PerfInsights that analyzes code profiles to find optimizations. And I learned about another agent, Linnix: AI-Powered Observability, while writing this post. Final Thoughts There are far more computers in the world than performance engineers to tune them, leaving most running untuned and wasting resources. In future there will be AI performance agents that can be run on everything, helping to save the planet by reducing energy usage. Some will be described as an AI Brendan or a Virtual Brendan (some already have been) but that doesn't mean they are necessarily trained on all my work or had any direct help from me creating it. (Nor did they abduct me and feed me into a steampunk machine that uploaded my brain to the cloud.) Virtual Brendans only try to automate about 15% of what I do. Intel and the AI auto-tuning startup it acquired for $650M (based on my work) were pioneers in this space, but after Intel invested more time and resources into the project it was shut down. That doesn't mean the idea was bad -- Intel's public statement about the shutdown only mentions a core business review -- and this happened while Intel has been struggling in general (as has been widely reported). Commercial AI auto-tuners have extra challenges: customers may only pay for one server/instance then copy-n-paste the tuning changes everywhere. Similar to the established pricing model of hiring a performance consultant. For 3rd-party code, someone at some point will have the bright idea to upstream any change an AI auto-tuner suggestss, so a commercial offering will keep losing whatever tuning advantages it develops. In-house tools don't have these same concerns, and perhaps that's the real future of AI tuning agents: an in-house or non-commercial open source collaboration.

- Dave Airlie (blogspot): fedora 43: bad mesa update oopsie (2025/11/24 01:42)
  F43 picked up the two patches I created to fix a bunch of deadlocks on laptops reported in my previous blog posting. Turns out Vulkan layers have a subtle thing I missed, and I removed a line from the device select layer that would only matter if you have another layer, which happens under steam.The fedora update process caught this, but it still got published which was a mistake, need to probably give changes like this more karma thresholds.I've released a new update https://bodhi.fedoraproject.org/updates/FEDORA-2025-2f4ba7cd17 that hopefully fixes this. I'll keep an eye on the karma.
- Brendan Gregg: Intel is listening, don't waste your shot (2025/11/21 13:00)
  Intel's new CEO, Lip-Bu Tan, has made listening to customers a top priority, saying at Intel Vision earlier this year: "Please be brutally honest with us. This is what I expect of you this week, and I believe harsh feedback is most valuable." I'd been in regular meetings with Intel for several years before I joined, and I had been giving them technical direction on various projects, including at times some brutal feedback. When I finally interviewed for a role at Intel I was told something unexpected: that I had already accomplished so much within Intel that I qualified to be an Intel Fellow candidate. I then had to pass several extra interviews to actually become a Fellow (and was told I may only be the third person in Intel's history to be hired as a Fellow) but what stuck with me was that I had already accomplished so much at a company I'd never worked for. If you are in regular meetings with a hardware vendor as a customer (or potential customer) you can accomplish a lot by providing firm and tough feedback, particularly with Intel today. This is easier said than done, however. Now that I've seen it from the other side I realize I could have accomplished more, and you can too. I regret the meetings where I wasn't really able to have my feedback land as the staff weren't really getting it, so I eventually gave up. After the meeting I'd crack jokes with my colleagues about how the product would likely fail. (Come on, at least I tried to tell them!) Here's what I wish I had done in any hardware vendor meeting: Prep before meetings: study the agenda items and look up

attendees on LinkedIn and note what they do, how many staff they say they manage, etc. Be aware of intellectual property risks: Don't accept meetings covered by some agreement that involves doing a transfer of intellectual property rights for your feedback (I wrote a post on this); ask your legal team for help. Make sure feedback is documented in the meeting minutes (e.g., a shared Google doc) and that it isn't watered down. Be firm about what you know and don't know: it's just as important to assert when you haven't formed an opinion yet on some new topic. Stick to technical criticisms that are constructive (uncompetitive, impractical, poor quality, poor performance, difficult to use, of limited use/useless) instead of trash talk (sucks, dumb, rubbish). Check minutes include who was present and the date. Ask how many staff are on projects if they say they don't have the resources to address your feedback (they may not answer if this is considered sensitive) and share industry expectations, for example: “This should only take one engineer one month, and your LinkedIn says you have over 100 staff.” Decline freeloading: If staff ask to be taught technical topics they should already know (likely because they just started a new role), decline, as I'm the customer and not a free training resource. Ask "did you Google it?" a lot: Sometimes staff join customer meetings to elevate their own status within the company, and ask questions they could have easily answered with Google or ChatGPT. Ask for staff/project bans: If particular staff or projects are consistently wasting your time, tell the meeting host (usually the sales rep) to take them off the agenda for at least a year, and don't join (or quit) meetings if they show up. Play bad cop, often no one else will. Review attendees. From time to time, consider: Am I meeting all the right people? Review the minutes. E.g., if you're meeting Intel and have been talking about a silicon change, have any actual silicon engineers joined the call? Avoid peer pressure: You may meet with the entire product team who are adamant that they are building something great, and you alone need to tell them it's garbage (using better words). Many times in my life I've been the only person to speak up and say uncomfortable things in meetings, yet I'm not the only person present who could. Ask for status updates: Be prepared that even if everyone appears grateful and appreciative of your feedback, you may realize six months later that nothing was done with it. Ask for updates and review the prior meeting minutes to see what you asked for and when. Speak to ELT/CEO: Once a year or so, ask to speak to someone on the executive leadership team (ELT; the leaders on the website) or the CEO. Share brutal feedback, and email them a copy of the meeting minutes showing the timeline of what you have shared and with whom. This may be the only way your feedback ever gets addressed, in particular for major changes. Ask to hear what they have been told about you and be prepared to refute details: your brutal feedback may have been watered down. I'm now in meetings from the other side where we'd really appreciate brutal feedback, but some customers aren't comfortable doing this, even when prompted. It isn't easy to tell someone their project is doomed, or that their reasons for not doing something are BS. It isn't easy dealing with peer pressure and a room of warm and friendly staff begging you say something, anything nice about their terrible product for fear of losing their jobs -- and realizing you must be brutal to their faces otherwise you're not helping the vendor or your own company. And it's extra effort to check meeting minutes and to push for meetings with the ELT or the CEO. Giving brutal feedback takes brutal effort.

- Linux Plumbers Conference: Slides templates available (2025/11/20 22:32)
  Dear speakers, You can find the LPC 2025 slides templates in different formats in the following link: https://drive.google.com/drive/folders/1oGQz6MXtq7fjRJS0Q7Q_oBI91g38VFOC They were created by our designer, Zohar Nir-Amitin. Zohar has been working with LPC since 2015, and has created all our wonderful t-shirts, badges and signage designs.
- Brendan Gregg: Third Stage Engineering (2025/11/16 13:00)
  The real performance of any computer hardware in production is the result of the hardware, software, and tuning; the investment and sequence of these efforts can be pictured as a three-stage rocket: I recently presented this embarrassingly simple diagram to Intel's executive leadership,

and at the time realized the value of sharing it publicly. The Internet is awash with comparisons about Intel (and other vendors') product performance based on hardware performance alone, but the performance of software and then tuning can make a huge difference for your particular workload. You need all three stages to reach the highest, and most competitive, performance. It's obvious why this is important for HW vendors to understand internally – they, like the Internet, can get overly focused on HW alone. But customers need to understand it as well. If a benchmark is comparing TensorFlow performance between HW vendors, was the Intel hardware tested using the Intel Extension for TensorFlow Software, and was it then tuned? The most accurate and realistic evaluation for HW involves selecting the best software and then tuning it, and doing this for all HW options. I spend a lot of time on the final stage, tuning – what I call third-stage engineering. It's composed of roughly four parts: People, training, tools, and capabilities. You need staff, you need them trained to understand performance methodologies and SW and HW internals, they need tools to analyze the system (both observational and experimental), and finally they need capabilities to tune (tunable parameters, settings, config, code changes, etc.). I see too many HW evaluations that are trying to understand customer performance but are considering HW alone, which is like only testing the first stage of a rocket. This doesn't help vendors or customers. I hope that's what my simple diagram makes obvious: We need all three stages to reach the highest altitude.

- Dave Airlie (blogspot): a tale of vulkan/nouveau/nvk/zink/mutter + deadlocks (2025/11/10 03:16)

  I had a bug appear in my email recently which led me down a rabbit hole, and I'm going to share it for future people wondering why we can't have nice things.Bug:1. Get an intel/nvidia (newer than Turing) laptop.2. Log in to GNOME on Fedora 42/43 3. Hotplug a HDMI port that is connected to the NVIDIA GPU.4. Desktop stops working.My initial reproduction got me a hung mutter process with a nice backtrace which pointed at the Vulkan Mesa device selection layer, trying to talk to the wayland compositor to ask it what the default device is. The problem was the process was the wayland compositor, and how was this ever supposed to work. The Vulkan device selection was called because zink called EnumeratePhysicalDevices, and zink was being loaded because we recently switched to it as the OpenGL driver for newer NVIDIA GPUs.I looked into zink and the device select layer code, and low and behold someone has hacked around this badly already, and probably wrongly and I've no idea what the code does, because I think there is at least one logic bug in it. Nice things can't be had because hacks were done instead of just solving the problem. The hacks in place ensured under certain circumstances involving zink/xwayland that the device select code to probe the window system was disabled, due to deadlocks seen. I'd no idea if more hacks were going to help, so I decided to step back and try and work out better.The first question I had is why WAYLAND_DISPLAY is set inside the compositor process, it is, and if it wasn't I would never hit this. It's pretty likely on the initial compositor start this env var isn't set, so the problem only becomes apparent when the compositor gets a hotplugged GPU output, and goes to load the OpenGL driver, zink, which enumerates and hits device select with env var set and deadlocks.I wasn't going to figure out a way around WAYLAND_DISPLAY being set at this point, so I leave the above question as an exercise for mutter devs.How do I fix it?Attempt 1:At the point where zink is loading in mesa for this case, we have the file descriptor of the GPU device that we want to load a driver for. We don't actually need to enumerate all the physical devices, we could just find the ones for that fd. There is no API for this in Vulkan. I wrote an initial proof of concept instance extensions call VK_MESA_enumerate_devices_fd. I wrote initial loader code to play with it, and wrote zink code to use it. Because this is a new instance API, device-select will also ignore it. However this ran into a big problem in the Vulkan loader. The loader is designed around some internals that PhysicalDevices will enumerate in similiar ways, and it has to trampoline PhysicalDevice handles to underlying driver pointers so that if an app enumerates once, and enumerates again later, the PhysicalDevice handles remain consistent for the first user. There is a lot of code, and I've no idea how hotplug GPUs might fail in such situations. I couldn't find a decent path forward without

knowing a lot more about the Vulkan loader. I believe this is the proper solution, as we know the fd, we should be able to get things without doing a full enumeration then picking the answer using the fd info. I've asked Vulkan WG to take a look at this, but I still need to fix the bug.Attempt 2:Maybe I can just turn off device selection, like the current hacks do, but in a better manner. Enter VK_EXT_layer_settings. This extensions allows layers to expose a layer setting in the instance creation. I can have the device select layer expose a setting which says don't touch this instance. Then in the zink code where we have a file descriptor being passed in and create an instance, we set the layer setting to avoid device selection. This seems to work but it has some caveats, I need to consider, but I think should be fine.zink uses a single VkInstance for it's device screen. This is shared between all pipe_screens. Now I think this is fine inside a compositor, since we shouldn't ever be loading zink via the non-fd path, and I hope for most use cases it will work fine, better than the current hacks and better than some other ideas we threw around. The code for this is in [1].What else might be affected:If you have a vulkan compositor, it might be worth setting the layer setting if the mesa device select layer is loaded, esp if you set the DISPLAY/WAYLAND_DISPLAY and do any sort of hotplug later. You might be safe if you EnumeratePhysicalDevices early enough, the reason it's a big problem in mutter is it doesn't use Vulkan, it uses OpenGL and we only enumerate Vulkan physical devices at runtime through zink, never at startup.AMD and NVIDIA I think have proprietary device selection layers, these might also deadlock in similiar ways, I think we've seen some wierd deadlocks in NVIDIA driver enumerations as well that might be a similiar problem. [1] https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/38252

- Linux Plumbers Conference: Japan Visas need a longer processing time (2025/10/29 13:13)
  If you hold a passport from a visa exempt country, this doesn’t apply to you: https://www.mofa.go.jp/j_info/visit/visa/short/novisa.html But if you don’t have a passport from that list, you do need a visa. Unfortunately, the change of government in Japan has made the process for getting a visa more taxing on the body supplying the invitation letter (in our case, the Linux Foundation). For this reason, the LF is insisting that anyone who needs a visa letter have their application in to the LF dashboard by 17 November at the latest: https://openprofile.dev/myevents?applyfor=visa-letter If you have any queries or problems with the process, please contact visaletters@linuxfoundation.org
- Matthew Garrett: Where are we on X Chat security? (2025/10/21 16:07)
  AWS had an outage today and Signal was unavailable for some users for a while. This has confused some people, including Elon Musk, who are concerned that having a dependency on AWS means that Signal could somehow be compromised by anyone with sufficient influence over AWS (it can't). Which means we're back to the richest man in the world recommending his own "X Chat", saying The messages are fully encrypted with no advertising hooks or strange “AWS dependencies” such that I can’t read your messages even if someone put a gun to my head.Elon is either uninformed about his own product, lying, or both.As I wrote back in June, X Chat genuinely end-to-end encrypted, but ownership of the keys is complicated. The encryption key is stored using the Juicebox protocol, sharded between multiple backends. Two of these are asserted to be HSM backed - a discussion of the commissioning ceremony was recently posted here. I have not watched the almost 7 hours of video to verify that this was performed correctly, and I also haven't been able to verify that the public keys included in the post were the keys generated during the ceremony, although that may be down to me just not finding the appropriate point in the video (sorry, Twitter's video hosting doesn't appear to have any skip feature and would frequently just sit spinning if I tried to seek to far and I should probably just download them and figure it out but I'm not doing that now). With enough effort it would probably also have been possible to fake the entire thing - I have no reason to believe that this has happened, but it's not externally verifiable.But let's assume these published public keys are legitimately the ones used in the HSM

Juicebox realms[1] and that everything was done correctly. Does that prevent Elon from obtaining your key and decrypting your messages? No.On startup, the X Chat client makes an API call called GetPublicKeysResult, and the public keys of the realms are returned. Right now when I make that call I get the public keys listed above, so there's at least some indication that I'm going to be communicating with actual HSMs. But what if that API call returned different keys? Could Elon stick a proxy in front of the HSMs and grab a cleartext portion of the key shards? Yes, he absolutely could, and then he'd be able to decrypt your messages.(I will accept that there is a plausible argument that Elon is telling the truth in that even if you held a gun to his head he's not smart enough to be able to do this himself, but that'd be true even if there were no security whatsoever, so it still says nothing about the security of his product)The solution to this is remote attestation - a process where the device you're speaking to proves its identity to you. In theory the endpoint could attest that it's an HSM running this specific code, and we could look at the Juicebox repo and verify that it's that code and hasn't been tampered with, and then we'd know that our communication channel was secure. Elon hasn't done that, despite it being table stakes for this sort of thing (Signal uses remote attestation to verify the enclave code used for private contact discovery, for instance, which ensures that the client will refuse to hand over any data until it's verified the identity and state of the enclave). There's no excuse whatsoever to build a new end-to-end encrypted messenger which relies on a network service for security without providing a trustworthy mechanism to verify you're speaking to the real service.We know how to do this properly. We have done for years. Launching without it is unforgivable.[1] There are three Juicebox realms overall, one of which doesn't appear to use HSMs, but you need at least two in order to obtain the key so at least part of the key will always be held in HSMs comments

- Pete Zaitcev: Time flies (2025/10/20 04:34)
  A guy who sits next to me is in his 70s, and he said: "I started out on a teletype." But I didn't. Not only I never lived in a world without computers, but when I started out, CRT displays were already a thing. Guys who worked on vacuum tube computers are in their 90s now.
- Pete Zaitcev: git submodule woe (2025/10/16 01:57)
  Problem:A submodule is stuck in a commit, like so:$ git show..... shows a stuck submodule--- a/badsub+++ b/badsub@@ -1 +1 @@-Subproject commit 4ba912892c1b8c213c6c2e78b3bf257635dc534e+Subproject commit 4b813c322ebe236cddc6b3acd70a31994efd7a56Solution:Focus on the commit, not submodule. Submodules work as designed, it's the commit that needs to be fixed (with `git commit --amend`, obviously):$ cd badsub$ git checkout 4ba912892c1b8c213c6c2e78b3bf257635dc534e$ cd ..$ git add badsub$ git commit --amendNowhere as bad as copying a file while preserving history. Still, not obvious if one focuses on `git submodule`.
- Pete Zaitcev: podman versus dbus (2025/10/16 01:51)
  Problem:`podman container ls` warns:WARN[0000] The cgroupv2 manager is set to systemd but there is no systemd user session availableSolution:$ sudo apt install dbus-user-session; systemctl --user start dbus
- Greg Kroah-Hartman: The only benchmark that matters is... (2025/10/01 00:00)
  ...the one that emulates your real workload. And for me (and probably many of you reading this), that would be “build a kernel as fast as possible.” And for that, I recommend the simple kcbench. I kcbench mentioned it a few years ago, when writing about a new workstation that Level One Techs set up for me, and I’ve been using that as my primary workstation ever since (just over 5 years!).
- Matthew Garrett: Investigating a forged PDF (2025/09/24 22:46)
  I had to rent a house for a couple of months recently, which is long enough in California that it pushes you into proper tenant protection law. As

landlords tend to do, they failed to return my security deposit within the 21 days required by law, having already failed to provide the required notification that I was entitled to an inspection before moving out. Cue some tedious argumentation with the letting agency, and eventually me threatening to take them to small claims court.This post is not about that.Now, under Californian law, the onus is on the landlord to hold and return the security deposit - the agency has no role in this. The only reason I was talking to them is that my lease didn't mention the name or address of the landlord (another legal violation, but the outcome is just that you get to serve the landlord via the agency). So it was a bit surprising when I received an email from the owner of the agency informing me that they did not hold the deposit and so were not liable - I already knew this.The odd bit about this, though, is that they sent me another copy of the contract, asserting that it made it clear that the landlord held the deposit. I read it, and instead found a clause reading SECURITY: The security deposit will secure the performance of Tenant’s obligations. IER may, but will not be obligated to, apply all portions of said deposit on account of Tenant’s obligations. Any balance remaining upon termination will be returned to Tenant. Tenant will not have the right to apply the security deposit in payment of the last month’s rent. Security deposit held at IER Trust Account., where IER is International Executive Rentals, the agency in question. Why send me a contract that says you hold the money while you're telling me you don't? And then I read further down and found this:Ok, fair enough, there's an addendum that says the landlord has it (I've removed the landlord's name, it's present in the original).Except. I had no recollection of that addendum. I went back to the copy of the contract I had and discovered:Huh! But obviously I could just have edited that to remove it (there's no obvious reason for me to, but whatever), and then it'd be my word against theirs. However, I'd been sent the document via RightSignature, an online document signing platform, and they'd added a certification page that looked like this:Interestingly, the certificate page was identical in both documents, including the checksums, despite the content being different. So, how do I show which one is legitimate? You'd think given this certificate page this would be trivial, but RightSignature provides no documented mechanism whatsoever for anyone to verify any of the fields in the certificate, which is annoying but let's see what we can do anyway.First up, let's look at the PDF metadata. pdftk has a dump_data command that dumps the metadata in the document, including the creation date and the modification date. My file had both set to identical timestamps in June, both listed in UTC, corresponding to the time I'd signed the document. The file containing the addendum? The same creation time, but a modification time of this Monday, shortly before it was sent to me. This time, the modification timestamp was in Pacific Daylight Time, the timezone currently observed in California. In addition, the data included two ID fields, ID0 and ID1. In my document both were identical, in the one with the addendum ID0 matched mine but ID1 was different.These ID tags are intended to be some form of representation (such as a hash) of the document. ID0 is set when the document is created and should not be modified afterwards - ID1 initially identical to ID0, but changes when the document is modified. This is intended to allow tooling to identify whether two documents are modified versions of the same document. The identical ID0 indicated that the document with the addendum was originally identical to mine, and the different ID1 that it had been modified.Well, ok, that seems like a pretty strong demonstration. I had the "I have a very particular set of skills" conversation with the agency and pointed these facts out, that they were an extremely strong indication that my copy was authentic and their one wasn't, and they responded that the document was "re-sealed" every time it was downloaded from RightSignature and that would explain the modifications. This doesn't seem plausible, but it's an argument. Let's go further.My next move was pdfalyzer, which allows you to pull a PDF apart into its component pieces. This revealed that the documents were identical, other than page 3, the one with the addendum. This page included tags entitled "touchUp_TextEdit", evidence that the page had been modified using Acrobat. But in itself, that doesn't prove anything - obviously it had been edited at some point to insert the landlord's name, it doesn't prove whether it happened before or after the signing.But in the process of editing,

Acrobat appeared to have renamed all the font references on that page into a different format. Every other page had a consistent naming scheme for the fonts, and they matched the scheme in the page 3 I had. Again, that doesn't tell us whether the renaming happened before or after the signing. Or does it?You see, when I completed my signing, RightSignature inserted my name into the document, and did so using a font that wasn't otherwise present in the document (Courier, in this case). That font was named identically throughout the document, except on page 3, where it was named in the same manner as every other font that Acrobat had renamed. Given the font wasn't present in the document until after I'd signed it, this is proof that the page was edited after signing.But eh this is all very convoluted. Surely there's an easier way? Thankfully yes, although I hate it. RightSignature had sent me a link to view my signed copy of the document. When I went there it presented it to me as the original PDF with my signature overlaid on top. Hitting F12 gave me the network tab, and I could see a reference to a base.pdf. Downloading that gave me the original PDF, pre-signature. Running sha256sum on it gave me an identical hash to the "Original checksum" field. Needless to say, it did not contain the addendum.Why do this? The only explanation I can come up with (and I am obviously guessing here, I may be incorrect!) is that International Executive Rentals realised that they'd sent me a contract which could mean that they were liable for the return of my deposit, even though they'd already given it to my landlord, and after realising this added the addendum, sent it to me, and assumed that I just wouldn't notice (or that, if I did, I wouldn't be able to prove anything). In the process they went from an extremely unlikely possibility of having civil liability for a few thousand dollars (even if they were holding the deposit it's still the landlord's legal duty to return it, as far as I can tell) to doing something that looks extremely like forgery.There's a hilarious followup. After this happened, the agency offered to do a screenshare with me showing them logging into RightSignature and showing the signed file with the addendum, and then proceeded to do so. One minor problem - the "Send for signature" button was still there, just below a field saying "Uploaded: 09/22/25". I asked them to search for my name, and it popped up two hits - one marked draft, one marked completed. The one marked completed? Didn't contain the addendum. comments

- Linux Plumbers Conference: In Person Registration is sold out (2025/09/19 13:22)
  Apparently there was quite a bit more demand than we anticipated. We are running a waitlist which you can get on by filling in this form: https://forms.gle/tYjjbyn66q5SQMLPA The venue is smaller this year but we do have a block of reserved passes for MC content so we'll allocate places to the waitlist after it's decided how many of them get used. Note that in order to be fair to everyone, if you sign up for the waitlist you'll have 7 days to register otherwise your pass will go to the next person.
- Linux Plumbers Conference: Registration for LPC 2025 is now open! (2025/09/15 20:16)
  We're happy to announce that registration for LPC 2025 is now open. To register please go to our attend page. To try to prevent the instant sellout, we are keeping our cancellation policy of no refunds, only transfers of registrations. You will find more details during the registration process. LPC 2025 follows the Linux Foundation's health & safety policy. As usual we expect to sell our rather quickly so don't delay your registration for too long!
- Dave Airlie (blogspot): radv takes over from AMDVLK (2025/09/15 19:08)
  AMD have announced the end of the AMDVLK open driver in favour of focusing on radv for Linux use cases.When Bas and I started radv in 2016, AMD were promising their own Linux vulkan driver, which arrived in Dec 2017. At this point radv was already shipping in most Linux distros. AMD strategy of having AMDVLK was developed via over the wall open source releases from internal closed development was always going to be a second place option at that point.When Valve came on board and brought dedicated developer power to radv, and the aco compiler matured, there really was no point putting effort into using AMDVLK which was hard to package and impossible to contribute to meaningfully for external

developers.radv is probably my proudest contribution to the Linux ecosystem, finally disproving years of idiots saying an open source driver could never compete with a vendor provided driver, now it is the vendor provided driver.I think we will miss the open source PAL repo as a reference source and I hope AMD engineers can bridge that gap, but it's often hard to find workarounds you don't know exist to ask about them. I'm also hoping AMD will add more staffing beyond the current levels especially around hardware enablement and workarounds.Now onwards to NVK victory :-)[1] https://github.com/GPUOpen-Drivers/AMDVLK/discussions/416

- Linux Plumbers Conference: The Call for Proposals is nearing its end! (2025/09/08 16:14)
  The CfPs for the Linux Plumbers events are coming to an end. If you still want to submit, please get your submission in by the deadline. The deadlines are: Refereed track: September 10th (that’s this Wednesday) Kernel Summit track: September 10th (See Ted Tso’s email about how to submit). eBPF track: September 29th Networking track: October 17th Each of the Microconferences has their own last day to submit. Those are listed in the Accepted Microconferences tab on the website. All submissions may be added in the Call for Proposals tab. Click the Submit new abstract button at the bottom of that page, and make sure you select the proper Track.
- Matthew Garrett: Locally hosting an internet-connected server (2025/09/06 15:20)
  I'm lucky enough to have a weird niche ISP available to me, so I'm paying $35 a month for around 600MBit symmetric data. Unfortunately they don't offer static IP addresses to residential customers, and nor do they allow multiple IP addresses per connection, and I'm the sort of person who'd like to run a bunch of stuff myself, so I've been looking for ways to manage this.What I've ended up doing is renting a cheap VPS from a vendor that lets me add multiple IP addresses for minimal extra cost. The precise nature of the VPS isn't relevant - you just want a machine (it doesn't need much CPU, RAM, or storage) that has multiple world routeable IPv4 addresses associated with it and has no port blocks on incoming traffic. Ideally it's geographically local and peers with your ISP in order to reduce additional latency, but that's a nice to have rather than a requirement.By setting that up you now have multiple real-world IP addresses that people can get to. How do we get them to the machine in your house you want to be accessible? First we need a connection between that machine and your VPS, and the easiest approach here is Wireguard. We only need a point-to-point link, nothing routable, and none of the IP addresses involved need to have anything to do with any of the rest of your network. So, on your local machine you want something like:[Interface]PrivateKey = privkeyhereListenPort = 51820Address = localaddr/32[Peer]Endpoint = VPS:51820 PublicKey = pubkeyhere AllowedIPs = VPS/0And on your VPS, something like:[Interface]Address = vpswgaddr/32SaveConfig = trueListenPort = 51820PrivateKey = privkeyhere[Peer]PublicKey = pubkeyhereAllowedIPs = localaddr/32The addresses here are (other than the VPS address) arbitrary - but they do need to be consistent, otherwise Wireguard is going to be unhappy and your packets will not have a fun time. Bring that interface up with wg-quick and make sure the devices can ping each other. Hurrah! That's the easy bit.Now you want packets from the outside world to get to your internal machine. Let's say the external IP address you're going to use for that machine is 321.985.520.309 and the wireguard address of your local system is 867.420.696.005. On the VPS, you're going to want to do:iptables -t nat -A PREROUTING -p tcp -d 321.985.520.309 -j DNAT --to-destination 867.420.696.005iptables -t nat -A PREROUTING -p udp -d 321.985.520.309 -j DNAT --to-destination 867.420.696.005Now, all incoming packets for 321.985.520.309 will be rewritten to head towards 867.420.696.005 instead (make sure you've set net.ipv4.ip_forward to 1 via sysctl!). Victory! Or is it? Well, no.What we're doing here is rewriting the destination address of the packets so instead of heading to an address associated with the VPS, they're now going to head to your internal system over the Wireguard link. Which is then going to ignore them, because the AllowedIPs statement in the config only allows packets coming from your VPS, and these packets still have their original source IP. We could rewrite the source IP to match the VPS IP, but then you'd have no

idea where any of these packets were coming from, and that sucks. Let's do something better. On the local machine, in the peer, let's update AllowedIps to 0.0.0.0/0 to permit packets form any source to appear over our Wireguard link. But if we bring the interface up now, it'll try to route all traffic over the Wireguard link, which isn't what we want. So we'll add table = off to the interface stanza of the config to disable that, and now we can bring the interface up without breaking everything but still allowing packets to reach us. However, we do still need to tell the kernel how to reach the remote VPN endpoint, which we can do with ip route add vpswgaddr dev wg0. Add this to the interface stanza as:PostUp = ip route add vpswgaddr dev wg0PreDown = ip route del vpswgaddr dev wg0That's half the battle. The problem is that they're going to show up there with the source address still set to the original source IP, and your internal system is (because Linux) going to notice it has the ability to just send replies to the outside world via your ISP rather than via Wireguard and nothing is going to work. Thanks, Linux. Thinux.But there's a way to solve this - policy routing. Linux allows you to have multiple separate routing tables, and define policy that controls which routing table will be used for a given packet. First, let's define a new table reference. On the local machine, edit /etc/iproute2/rt_tables and add a new entry that's something like:1 wireguardwhere "1" is just a standin for a number not otherwise used there. Now edit your wireguard config and replace table=off with table=wireguard - Wireguard will now update the wireguard routing table rather than the global one. Now all we need to do is to tell the kernel to push packets into the appropriate routing table - we can do that with ip rule add from localaddr lookup wireguard, which tells the kernel to take any packet coming from our Wireguard address and push it via the Wireguard routing table. Add that to your Wireguard interface config as:PostUp = ip rule add from localaddr lookup wireguardPreDown = ip rule del from localaddr lookup wireguardand now your local system is effectively on the internet.You can do this for multiple systems - just configure additional Wireguard interfaces on the VPS and make sure they're all listening on different ports. If your local IP changes then your local machines will end up reconnecting to the VPS, but to the outside world their accessible IP address will remain the same. It's like having a real IP without the pain of convincing your ISP to give it to you. comments

- Matthew Garrett: Cordoomceps - replacing an Amiga's brain with Doom (2025/08/05 03:43)
  There's a lovely device called a pistorm, an adapter board that glues a Raspberry Pi GPIO bus to a Motorola 68000 bus. The intended use case is that you plug it into a 68000 device and then run an emulator that reads instructions from hardware (ROM or RAM) and emulates them. You're still limited by the ~7MHz bus that the hardware is running at, but you can run the instructions as fast as you want.These days you're supposed to run a custom built OS on the Pi that just does 68000 emulation, but initially it ran Linux on the Pi and a userland 68000 emulator process. And, well, that got me thinking. The emulator takes 68000 instructions, emulates them, and then talks to the hardware to implement the effects of those instructions. What if we, well, just don't? What if we just run all of our code in Linux on an ARM core and then talk to the Amiga hardware?We're going to ignore x86 here, because it's weird - but most hardware that wants software to be able to communicate with it maps itself into the same address space that RAM is in. You can write to a byte of RAM, or you can write to a piece of hardware that's effectively pretending to be RAM[1]. The Amiga wasn't unusual in this respect in the 80s, and to talk to the graphics hardware you speak to a special address range that gets sent to that hardware instead of to RAM. The CPU knows nothing about this. It just indicates it wants to write to an address, and then sends the data.So, if we are the CPU, we can just indicate that we want to write to an address, and provide the data. And those addresses can correspond to the hardware. So, we can write to the RAM that belongs to the Amiga, and we can write to the hardware that isn't RAM but pretends to be. And that means we can run whatever we want on the Pi and then access Amiga hardware.And, obviously, the thing we want to run is Doom, because that's what everyone runs in fucked up hardware situations.Doom was Amiga kryptonite. Its entire graphical model was based on memory directly representing the contents of your display, and being able to modify that by just moving pixels around. This

worked because at the time VGA displays supported having a memory layout where each pixel on your screen was represented by a byte in memory containing an 8 bit value that corresponded to a lookup table containing the RGB value for that pixel.The Amiga was, well, not good at this. Back in the 80s, when the Amiga hardware was developed, memory was expensive. Dedicating that much RAM to the video hardware was unthinkable - the Amiga 1000 initially shipped with only 256K of RAM, and you could fill all of that with a sufficiently colourful picture. So instead of having the idea of each pixel being associated with a specific area of memory, the Amiga used bitmaps. A bitmap is an area of memory that represents the screen, but only represents one bit of the colour depth. If you have a black and white display, you only need one bitmap. If you want to display four colours, you need two. More colours, more bitmaps. And each bitmap is stored in an independent area of RAM. You never use more memory than you need to display the number of colours you want to.But that means that each bitplane contains packed information - every byte of data in a bitplane contains the bit value for 8 different pixels, because each bitplane contains one bit of information per pixel. To update one pixel on screen, you need to read from every bitmap, update one bit, and write it back, and that's a lot of additional memory accesses. Doom, but on the Amiga, was slow not just because the CPU was slow, but because there was a lot of manipulation of data to turn it into the format the Amiga wanted and then push that over a fairly slow memory bus to have it displayed.The CDTV was an aesthetically pleasing piece of hardware that absolutely sucked. It was an Amiga 500 in a hi-fi box with a caddy-loading CD drive, and it ran software that was just awful. There's no path to remediation here. No compelling apps were ever released. It's a terrible device. I love it. I bought one in 1996 because a local computer store had one and I pointed out that the company selling it had gone bankrupt some years earlier and literally nobody in my farming town was ever going to have any interest in buying a CD player that made a whirring noise when you turned it on because it had a fan and eventually they just sold it to me for not much money, and ever since then I wanted to have a CD player that ran Linux and well spoiler 30 years later I'm nearly there. That CDTV is going to be our test subject. We're going to try to get Doom running on it without executing any 68000 instructions.We're facing two main problems here. The first is that all Amigas have a firmware ROM called Kickstart that runs at powerup. No matter how little you care about using any OS functionality, you can't start running your code until Kickstart has run. This means even documentation describing bare metal Amiga programming assumes that the hardware is already in the state that Kickstart left it in. This will become important later. The second is that we're going to need to actually write the code to use the Amiga hardware.First, let's talk about Amiga graphics. We've already covered bitmaps, but for anyone used to modern hardware that's not the weirdest thing about what we're dealing with here. The CDTV's chipset supports a maximum of 64 colours in a mode called "Extra Half-Brite", or EHB, where you have 32 colours arbitrarily chosen from a palette and then 32 more colours that are identical but with half the intensity. For 64 colours we need 6 bitplanes, each of which can be located arbitrarily in the region of RAM accessible to the chipset ("chip RAM", distinguished from "fast ram" that's only accessible to the CPU). We tell the chipset where our bitplanes are and it displays them. Or, well, it does for a frame - after that the registers that pointed at our bitplanes no longer do, because when the hardware was DMAing through the bitplanes to display them it was incrementing those registers to point at the next address to DMA from. Which means that every frame we need to set those registers back.Making sure you have code that's called every frame just to make your graphics work sounds intensely irritating, so Commodore gave us a way to avoid doing that. The chipset includes a coprocessor called "copper". Copper doesn't have a large set of features - in fact, it only has three. The first is that it can program chipset registers. The second is that it can wait for a specific point in screen scanout. The third (which we don't care about here) is that it can optionally skip an instruction if a certain point in screen scanout has already been reached. We can write a program (a "copper list") for the copper that tells it to program the chipset registers with the locations of our bitplanes and then wait until the end of the frame, at which point it

will repeat the process. Now our bitplane pointers are always valid at the start of a frame.Ok! We know how to display stuff. Now we just need to deal with not having 256 colours, and the whole "Doom expects pixels" thing. For the first of these, I stole code from ADoom, the only Amiga doom port I could easily find source for. This looks at the 256 colour palette loaded by Doom and calculates the closest approximation it can within the constraints of EHB. ADoom also includes a bunch of CPU-specific assembly optimisation for converting the "chunky" Doom graphic buffer into the "planar" Amiga bitplanes, none of which I used because (a) it's all for 68000 series CPUs and we're running on ARM, and (b) I have a quad core CPU running at 1.4GHz and I'm going to be pushing all the graphics over a 7.14MHz bus, the graphics mode conversion is not going to be the bottleneck here. Instead I just wrote a series of nested for loops that iterate through each pixel and update each bitplane and called it a day. The set of bitplanes I'm operating on here is allocated on the Linux side so I can read and write to them without being restricted by the speed of the Amiga bus (remember, each byte in each bitplane is going to be updated 8 times per frame, because it holds bits associated with 8 pixels), and then copied over to the Amiga's RAM once the frame is complete.And, kind of astonishingly, this works! Once I'd figured out where I was going wrong with RGB ordering and which order the bitplanes go in, I had a recognisable copy of Doom running. Unfortunately there were weird graphical glitches - sometimes blocks would be entirely the wrong colour. It took me a while to figure out what was going on and then I felt stupid. Recording the screen and watching in slow motion revealed that the glitches often showed parts of two frames displaying at once. The Amiga hardware is taking responsibility for scanning out the frames, and the code on the Linux side isn't synchronised with it at all. That means I could update the bitplanes while the Amiga was scanning them out, resulting in a mashup of planes from two different Doom frames being used as one Amiga frame. One approach to avoid this would be to tie the Doom event loop to the Amiga, blocking my writes until the end of scanout. The other is to use double-buffering - have two sets of bitplanes, one being displayed and the other being written to. This consumes more RAM but since I'm not using the Amiga RAM for anything else that's not a problem. With this approach I have two copper lists, one for each set of bitplanes, and switch between them on each frame. This improved things a lot but not entirely, and there's still glitches when the palette is being updated (because there's only one set of colour registers), something Doom does rather a lot, so I'm going to need to implement proper synchronisation.Except. This was only working if I ran a 68K emulator first in order to run Kickstart. If I tried accessing the hardware without doing that, things were in a weird state. I could update the colour registers, but accessing RAM didn't work - I could read stuff out, but anything I wrote vanished. Some more digging cleared that up. When you turn on a CPU it needs to start executing code from somewhere. On modern x86 systems it starts from a hardcoded address of 0xFFFFFFF0, which was traditionally a long way any RAM. The 68000 family instead reads its start address from address 0x00000004, which overlaps with where the Amiga chip RAM is. We can't write anything to RAM until we're executing code, and we can't execute code until we tell the CPU where the code is, which seems like a problem. This is solved on the Amiga by powering up in a state where the Kickstart ROM is "overlayed" onto address 0. The CPU reads the start address from the ROM, which causes it to jump into the ROM and start executing code there. Early on, the code tells the hardware to stop overlaying the ROM onto the low addresses, and now the RAM is available. This is poorly documented because it's not something you need to care if you execute Kickstart which every actual Amiga does and I'm only in this position because I've made poor life choices, but ok that explained things. To turn off the overlay you write to a register in one of the Complex Interface Adaptor (CIA) chips, and things start working like you'd expect.Except, they don't. Writing to that register did nothing for me. I assumed that there was some other register I needed to write to first, and went to the extent of tracing every register access that occurred when running the emulator and replaying those in my code. Nope, still broken. What I finally discovered is that you need to pulse the reset line on the board before some of the hardware starts working - powering it up doesn't put you in a well defined state, but resetting it does.So, I now

have a slightly graphically glitchy copy of Doom running without any sound, displaying on an Amiga whose brain has been replaced with a parasitic Linux. Further updates will likely make things even worse. Code is, of course, available.[1] This is why we had trouble with late era 32 bit systems and 4GB of RAM - a bunch of your hardware wanted to be in the same address space and so you couldn't put RAM there so you ended up with less than 4GB of RAM comments

- Brendan Gregg: When to Hire a Computer Performance Engineering Team (2025) part 1 of 2 (2025/08/03 14:00)
  As a leader in computer performance I've been asked by companies about how (and why) to form a performance engineering team, and as this is broadly useful I'll share my advice here. Large tech companies in the US hire performance engineers (under that or other titles) to ensure that infrastructure costs and service latency don't grow too high, and that their service is reliable under peak load. A new performance team can likely find enough optimizations to halve infrastructure spend in their first couple of years, even for companies that have been using commercial performance or observability tools. Performance engineers do much more than those tools, working with development teams and vendors to build, test, debug, tune, and adopt new performance solutions, and to find deep optimizations that those tools can miss. I previously worked on the performance engineering team for Netflix, a large tech consumer running on hundreds of thousands of AWS instances. I'm now doing similar work at Intel (a large tech vendor) for Intel and their customers. As a leader in this space I've also interacted with other performance teams and staff doing performance work at many companies. In this post I'll explain what these teams do and when you should consider forming one. In part 2 I'll provide sample job descriptions, specialties, advice, pitfalls, comments on AI, and what to do if you can't hire a performance team. It's easy for hardware vendors like Intel to justify hiring performance engineers, as the number one factor in sales is beating a competitor's performance. However, my focus in this post is on non-vendor tech-heavy companies who hire these staff to drive down costs and latency outliers (e.g., banks, telecomms, defence, AI, tech-based companies, and anyone else who is spending more than $1M/year on back-end compute and AI). What is the ROI of performance engineering? The main ROIs are infrastructure cost savings, latency reductions, improved scalability and reliability, and faster engineering. The cost savings alone can justify a performance team and can help calculate its size, and I'll explore that in depth, but the other ROIs are worth considering and may be more important to a company depending on its stage of growth. Infrastructure Cost Savings and Margin Improvements An appropriately-sized performance team should be targeting 5-10% cost savings per year through tuning and product adoptions. (I'll explain appropriate sizes in the When To Hire section.) For many large companies a 5% result would be considered "good" and a 10% would be "great." Achieving this in practice can mean finding large wins (15-80%) on parts of the infrastructure, which become 5-10% overall. Wins are cumulative, so a team hitting 5% savings each year will multiply to become 28% after their 5th year (like compound interest). Even a modest 2% per year will become significant over time. While these compounded numbers can become large, a team needs to continue finding new cost savings each year to justify long-term retention, and should always be focused on the next 5-10%. Companies may invest in this work for more than just the cost savings: It can be about developing a competitive advantage in their area by providing a better cost/performance ratio, especially for companies with similar tech-based services that pass costs on to customers. For sites that haven't employed performance engineers before, there can be enough low-hanging fruit that the team can halve infrastructure costs in their first couple of years (50%). It all depends on the number of staff, their level of expertise, how much perf work other staff are already doing (senior developers, SREs), how much custom code is running, and how complex and volatile the stack is. It would great if we could publicly share specific results, which would look something like this: "This year we helped reduce our company's infrastructure spend by 5%, from $60M/year to $57M/year, however, since our user base also grew by 3%, we actually reduced cost-per-user by 8%, saving $5M/year in this and all future years." However, these numbers are

usually considered financially sensitive as they can reveal company growth, financial health, confidential infrastructure discounts, etc. As a performance engineer I can talk publicly about percent wins on a back-end service, but I usually can't map it to dollar signs. That doesn't help other companies to understand the value of performance engineering. It's not so much of a problem in Silicon Valley, since staff change companies all the time and word spreads about the latest practices in tech. But in far away countries performance engineering doesn't really exist yet, even though there are companies with sufficiently large infrastructure spend. Continuing the above example, a typical 8% win could be composed of: 2%: direct optimizations. We looked across all the infrastructure and found on average 5% wins on 40% of the infrastructure (on the rest we found nothing, this year). 3%: developer/SRE enablement. We developed and adopted custom observability tools and helped developers use them, who in turn found some 15% wins across 20% of our infrastructure. 3%: vendor adoptions. We supported a major adoption project that replaced 10% of our infrastructure for a 30% win. With developer/SRE enablement and vendor adoptions, the performance team isn't finding the wins directly but is enabling other teams and vendors to do so. For example, when I worked at Netflix we built and maintained the flame graph "self-service" application, which developers used daily to find wins, and we worked on multiple product adoptions every year. This all needs to be considered as part of the performance team’s ROI. Latency Reductions Reducing the response time or latency of a service is a large part of performance engineering. This involves analyzing average latency, 99th percentile latency, and outlier latency; ensuring latency SLA/SLOs are met; and ensuring acceptable latency during perturbations or peak usage. Many of the cost optimizations described earlier will also reduce average latency, but latency variance or outliers can remain. For example, a once-every-5-minute system task may have negligible cost and CPU footprint, but it may briefly perturb the application and cause latency outliers. These are debugged differently, often using monitoring, logs, distributed tracing, system-level tracing, packet logs, and custom ad-hoc tools. Sometimes the high latency is caused by the request type itself (the system is fine, but the end-user has requested a slow thing) or is an expected consequence of load (queueing theory, tail latency). Other times it can be from complex interactions across multiple layers of the software stack or from interactions across multiple network endpoints. As a related aside: One performance anti-pattern is when a company, to debug one performance problem, installs a monitoring tool that periodically does work and causes application latency outliers. Now the company has two problems. Tip: try turning off all monitoring agents and see if the problem goes away. While latency is the main consideration to improve end user experience, others include throughput and parallelism. Improved Scalability and Reliability Systems under load can respond with exponential latency or a cascading failure, causing disruptions or a service outage. Performance engineers can test resource scalability with custom load generators and benchmarks, and use analysis tools to study all parts of the system to find and solve bottlenecks. A performance engineer will not just measure scalability limits, but should also explain what the limiting factors are and how to address them to scale further. A stable and performant service will also earn trust in your company, and can help you grow customers more quickly. It may be a requirement for satisfying enterprise SLA/SLOs. I'll share a scalability story from my time at Sun Microsystems (a vendor). My goal was to achieve the number one throughput in the industry for a storage system, which would require exceeding 1M IOPS. The expected bottleneck was the rotational disks. I developed my own load generators and analysis tools and concluded that the real bottleneck was, surprisingly, the CPU interconnect. The interconnect was AMD HyperTransport 1, so AMD sent me a new systemboard with HT3 and faster CPUs. I installed it and…performance was identical. I was upset with myself for getting it wrong, until I discovered that AMD had sent me a HT1 board by mistake. They then sent me a real HT3 board and the performance increased by up to 75%! The CPU interconnect (when present) is just one of many components that companies typically don't check, and commercial observability tools don't check either. Faster Engineering Performance engineers can take care of components outside of a developer's code base so the developers can stay focused,

and also provide them with a greater performance budget so that they can adopt expensive features earlier. For some early stage companies this ROI may be their most important (and is sometimes called engineering velocity). In detail: Eliminate outside performance distractions: A development team can be coding at one hundred miles an hour in their codebase, but then run into a performance problem in a library, kernel, hypervisor, or hardware component, and need to slow down to learn some new thing and its analysis tools. Or, it could be some new performance technology that someone saw on hackernews and the development team wonder if they should stop to evaluate it. These performance activities can be offloaded to the perf team so the developers stay focused on their code and at full speed. This is the same as how OS, deployment, and reliability issues can be offloaded to SRE and DevOps teams. Bypass expensive project failures: Good performance engineers know the internals of the software and hardware stack (including the Linux kernel). Many developers don't and most of the time they don't need to. But this can lead to developers proposing ideas based on incorrect assumptions. (This actually happens a lot.) Having a one hour meeting with a performance engineering team can save months of engineering effort: A developer talks about their ideas A and B, and the perf team says "A won't work and this is why, but B sounds good and we can help." Months saved in one hour. At one large tech company many years ago I analyzed an engineering proposal to adopt a new event-based coding framework on the belief it would improve performance by ten fold. My analysis showed the real gain would have been less than 10%. The project was abandoned. This saved having all the engineers rewrite all the company's code, something we were expecting would take a year. This is one of the biggest performance wins of my career, not measured as a percentage but rather as engineering hours saved. Develop accelerator tools: As mentioned earlier, performance teams can develop custom observability tools that developers use, finding issues sooner and accelerating development. For example, I've been publishing here about AI flame graphs, which involved kernel and hardware internals to build. Faster feature adoption: Reducing costs and latency can enable developers to do more with a limited infrastructure and latency budget, offering more capabilities for their service or allowing more processing to be crammed in per request. Some companies have cost limits for adding features, so optimization work can mean a feature is now approved. All this work can mean a company can outpace their competitors with feature adoption, while maintaining a reliable, low-latency, cost/performance competitive service. What do performance engineers do? For non-vendor tech companies, in summary: A. Test, debug, and tune new software and hardware products to find performance improvements, and drives company-wide adoption. Examples: New cloud instance types, language runtimes, JVM versions, JVM subsystems (new GC algorithms or compilers: Graal vs c2), system libraries (glibc vs tcmalloc etc.), kernels (Linux vs BSD) and versions, compilers (gcc, llvm, icc), processor features (AVX, QAT, etc.), hardware accelerators, and so on. It can take months to debug, fix, and patch everything so the latest thing delivers its performance claim. B. Develop in-house performance solutions, such as custom analysis tools, that other teams use to find performance wins. Examples: Custom monitoring using Prometheus and Grafana, one-click flame graphs, and analysis tools using eBPF: All of this is open-source based, but someone has to get it working locally, integrate them with existing local tools, teach other teams how to use them, and maintain them. C. Does deep-dive analysis to identify and reduce workload bottleneck(s) and latency outliers. Examples: Using code profilers (CPU flame graphs), distributed tracers (OpenTelemetry and products), application logs, system counters (Linux: sysstat), system tracers (Linux: eBPF, Ftrace, perf), static and dynamic instrumentation (Linux: kprobes, uprobes), debuggers (gdb, etc.), hardware counters (Linux: perf), and on rare occasions hardware instruction tracing. A lot of hands-on live debugging over an SSH session, following methodologies to efficiently find the root-cause(s), which can require the development of custom tools (mini load generators, observability tools, etc.). D. Optimize software and hardware via tunable parameters and configuration choices. Examples: System tunables (Linux: sysctls), network tunables (socket options, qdiscs), device tunables, runtime tunables (Java -XX:*), library settings, environment variables,

etc. As with (C), the team needs SSH access to do this and likely superuser privileges. E. Work with development teams (internal and external) to catch non-scalable solutions early in development, and to suggest or test later performance improvements. Examples: Identifying communication layer will flood network links when horizontally scaled; A developer has a good optimization idea but can't get it to work and needs some help; There's a performance-related pull request on some software the company uses but the request is two years old and needs someone to fix code conflicts, test it, and advocate for merging it. F. Develop proof-of-concept demonstrations of new performance technologies. Examples: Linux eBPF and io_uring can provide significant performance improvements when developed into hot-path kernel-based accelerators, but someone needs to at least build a POC to show it would work for the company. These are typically too esoteric for developers to try on their own. G. Develop performance improvements directly for internal and external code. Examples: Performance engineers get a lot done by asking the right people, but sometimes no one has the time to code that Linux/runtime/database performance fix the company needs, so a perf engineer takes it on. We aren't as quick as full-time developers since we are hopping between different languages all the time, and as a new code base committer will typically come under extra (and time-consuming) scrutiny. H. Capacity planning activities: purchase guidance, choosing metrics to monitor, and bottleneck forecasting. Examples: Modeling and performance characterization for hardware purchases, resource utilization monitoring to forecast capacity issues (nowadays often done by developers and SREs using monitoring tools); propose the best metrics to be watched in those monitoring tools for alert generation and auto-scaling rules; work with business side of the company to help define practical SLA/SLOs. I. Perform knowledge sharing to uplift engineering. Examples: Performance education to help developers produce more efficient software; act as a conduit to share performance learnings between teams (that may otherwise be siloed) to avoid rework and rediscovery. J. Provide in-house expertise to guide purchasing performance solutions. Examples: Providing in-house expertise for performance topics like observability, telemetry, and eBPF can help the company choose better commercial products by evaluating their capabilities and overhead costs, and can recognize which are just Prometheus and Grafana, or my open source eBPF tools, in a suit. Without expertise you're vulnerable to being ripped off, or may adopt a tool that increases infrastructure costs more than the gains it provides (I've seen some that have overhead exceeding 10%). To elaborate on (A), the testing of new products: Other staff will try a technology by configuring it based on the README, run a load test, and then share the result with management. Some companies hire dedicated staff for this called "performance testers." Performance engineers get more out of the same technology by running analyzers during the test to understand its limiter ("active benchmarking"), and will tune the technology to get an extra 5%, 50%, or more performance. They may also discover that the limiter is an unintended target (e.g., accidentally testing a caching layer instead). Any performance test should be accompanied by an explanation of the limiting factor, since no explanation will reveal the test wasn't analyzed and the result may be bogus. You can simply ask "why is the result not double?". As an aside: "CPU bound" isn't an explanation. Do you mean (a) clockspeed, (b) thread pool size, (c) core count, (d) memory bus (which kernels misleadingly include in %CPU counters), or something else (like power, thermal, CPU subsystem bottleneck)? Each of those leads to a different actionable item for the company (E.g.: (a) faster processors; (b) more threads; (c) more cores; (d) faster memory, bigger caches, less NUMA, or software techniques like zero copy). That's just the generic stuff. The code behind any CPU bound workload will also be analyzed to look for inefficiencies, and sometimes their instructions as well. Day to day, a performance engineer can spend a lot of time fixing broken builds and configuring workloads, because you're the first person testing new patches and bleeding-edge software versions. What I've described here is for companies that consume tech. For vendors that sell it, performance engineering includes design modeling, analysis of prototype and in-development software and hardware, competitive benchmarking, non-regression testing of new product releases, and pre- and post-sales performance analysis support. (I explain this in more

detail in Systems Performance 2nd edition, chapter 1.) When to Hire a Performance Team and How Many Most companies I've encountered are already doing some kind of performance work scattered across projects and individuals, but they don't yet have a central performance engineering team looking deeply at everything. This leaves their attention spotty, ok in some areas, poor to absent in others. A central performance team looks at everything and prioritizes work based on the potential ROI. Here are a few rough rules to determine when you should start forming a company-wide performance engineering team and how to size it (see caveats at the end): (A) One engineer at $1M/year infrastructure spend, then one per $10M to $20M/year That first engineer finds some of the low-hanging fruit, and should be cost effective as your company grows past $1M/year. I'd then consider another performance engineer for every $10M to $20M, and maintain a 3:1 junior:senior ratio. The values you use depend on your performance engineer's skill and the complexity of your environment, and how aggressively you wish to improve performance. At a $20M spend, 5% yearly wins means $1M in savings per staff member (minus their cost); whereas for a $10M spend you'd need to hit 10% wins yearly for $1M in savings. Consider that as your spend keeps growing you will keep adding more staff, which makes their job harder as there is less low-hanging fruit to find. However, your site will also be growing in scale and complexity, and developing new performance issues for the growing team to solve. Also, smaller percent wins become more impactful at large scale, so I expect such a growing perf team to remain cost effective. (To a point: the largest teams I've seen stop at around 150 staff.) (B) Staff spend should equal or exceed observability monitoring spend If you're spending $1M/year on an observability product, you can spend $1M/year on a performance engineering team: e.g., 3 to 4 good staff. If you're only spending $50k/year on an observability product, you can't hire a performance engineer at that price, but you can bring in a consultant or pay for performance training and conference attendance. As I'd expect staff to halve infrastructure costs over time, just the savings on monitoring alone (which typically scale with instance/server count) will pay for the new staff. Because these new engineers are actively reducing infrastructure spend, the total savings are much greater. (C) When latency or reliability is prohibitive to growth I’ve heard some small companies and startups say they spend more money on coffee than they do back-end compute, and don't want to waste limited developer time on negligible cost reductions. However, when a wave of new customers arrive they may hit scalability issues and start losing customers because latency is too high or reliability is too inconsistent. That's usually a good time for small companies to start investing in performance engineering. Caveats for A-C You likely already have some perf engineers under different titles, such as senior developers or SREs who are focusing on perf work, leaving less work for the central performance team. Account for these focused staff when you are determining how many performance engineers to hire. There will be a backpressure effect: If a new team halves your infrastructure, do you then halve the team? Up to you. But first think about what a perf engineer does: They have to work on anything, any operating system, language, or hardware the company needs. So you have these extra staff that can go deep on anything. Just some personal examples: There was an 18 month stretch when Netflix needed more core SREs on rotation, so myself and other perf engineers were temporarily assigned to do SRE work; Netflix would also have me do kernel crash dump analysis and application core dump analysis, since I was already using debuggers at that level. You ideally have more performance engineers than technologies so staff members can specialize. E.g.: If your stack is AWS, Intel, Linux kernel, Ubuntu OS, containers, lambda, gRPC, Java, golang, Python, Cassandra, and TensorFlow, that's 12 performance engineers. Plus at least one for locally developed code. Want to go multi-cloud? Add another engineer for each CSP. Remember that performance wins are cumulative for future years. A novice team could struggle to get up to speed with both performance engineering and your complex environment, and could also discover that you already had senior staff find all the low-hanging fruit. They might therefore only deliver 2% cost savings in each of their first three years. But that still combines to be 6% going forward. Companies and Global Staff Here are some example articles about performance engineering work at

non-vendor companies: Netflix: Cloud Performance Root Cause Analysis [me, 2018] Meta: Strobelight: A profiling service built on open source technology [Jordan Rome, 2025] Pinterest: Debugging the One-in-a-Million Failure: Migrating Pinterest's Search Infrastructure to Kubernetes [Samson Hu, et al. 2025] LinkedIn: Who moved my 99th percentile latency? [Richard Hsu, Cuong Tran, 2015] eBay: Speed by a Thousand Cuts [Senthil Padmanabhan, 2020] Twitter: #ExpandTheEdge: Making Twitter Faster [Todd Segal, Anthony Roberts, 2019] Salesforce: How Salesforce makes performance engineering work at Enterprise Scale [Archana Sethuraman] Uber: PerfInsights: Detecting Performance Optimization Opportunities in Go Code using Generative AI [Lavanya Verma, Ryan Hang, Sung Whang, Joseph Wang] Twitch: Go's march to low-latency GC [Rhys Hiltner, 2016] Airbnb: Creating Airbnb's Page Performance Score [Andrew Scheuermann, 2021] Stripe: Using ML to detect and respond to performance degradations in slices of Stripe payments [Lakshmi Narayan, Lakshmi Narayan, 2025] DoorDash: How DoorDash Standardized and Improved Microservices Caching [Lev Neiman, Jason Fan, 2023] Roblox: How We Redesigned a Highly Scaled Data Store to Lower 99th Percentile Latency 100x [Andrew Korytko, 2020] Capital One: Driving cloud value through cost optimization & efficiency [Jerzy Grzywinski, Brent Segner, 2024] I would like to add Bank of America, Wells Fargo, JPMorgan Chase, and CitiGroup to this list since they have many staff with the title "performance engineer" (as you can find on LinkedIn) but it's hard to find public articles about their work. I'd also like a canonical list of central performance engineering teams, but such org chart data can also be hard to find online, and staff don't always call themselves "performance engineers." Other keywords to look out for are: insights, monitoring, and observability; some are just called "support engineers". Note that there is also a lot of performance engineering done at hardware, software, and cloud vendors (Intel, AMD, NVIDIA, Apple, Microsoft, Google, Amazon, Red Hat, etc.) not listed here, as well as at performance solution companies. In this post I just wanted to focus on non-vendor companies. Global Staff I've never seen concrete data on how many people are employed worldwide in performance engineering. Here are my guesses: Staff identifying as performance engineers at non-vendor companies: <1,000. Staff identifying as performance engineers at SW/HW vendors: >10,000. Staff who have become focused on performance engineering work under other titles (developer, SRE, support): >100,000. Staff who occasionally do performance engineering work: I'd guess most developers. It's possible LinkedIn can provide better estimates if you have enterprise access. Conclusion There are many reasons to hire a performance engineering team, such as infrastructure cost savings, latency reductions, improved scalability and reliability, and faster engineering. Cost savings alone can justify hiring a team, because a team should be targeting 5-10% cost reductions every year, which over the years adds up to become significantly larger: 28%-61% savings after 5 years. In this post I explained what performance engineers do and provided some suggested rules on hiring: A) One engineer at >$1M infrastructure spend, then another for every $10-20M. B) Performance staff spend should equal or exceed observability monitoring spend. Note that you likely already have some senior developers or SREs who are focusing on perf work, reducing the number of new performance engineers you need. I've met people who would like to work as performance engineers but their employer has no such roles (other than performance testing: not the same thing) despite spending millions per year on infrastructure. I hope this post helps companies understand the value of performance engineering and understand when and how many staff to hire. Hiring good performance engineers isn't easy as it's a specialized area with a limited talent pool. In part 2 I'll discuss how to hire or train a performance engineering team and provide sample job descriptions and tips, and what to do if you can't hire a performance team. Thanks Thanks for the feedback and suggestions: Vadim Filanovsky (OpenAI), Jason Koch (Netflix), Ambud Sharma (Pinterest), Harshad Sane (Netflix), Ed Hunter, Deirdre Straughan.

- Matthew Garrett: Secure boot certificate rollover is real but probably won't hurt you (2025/07/31 16:53)
  LWN wrote an article which opens with the assertion "Linux users who have Secure Boot enabled on their systems knowingly or unknowingly rely

on a key from Microsoft that is set to expire in September". This is, depending on interpretation, either misleading or just plain wrong, but also there's not a good source of truth here, so.First, how does secure boot signing work? Every system that supports UEFI secure boot ships with a set of trusted certificates in a database called "db". Any binary signed with a chain of certificates that chains to a root in db is trusted, unless either the binary (via hash) or an intermediate certificate is added to "dbx", a separate database of things whose trust has been revoked[1]. But, in general, the firmware doesn't care about the intermediate or the number of intermediates or whatever - as long as there's a valid chain back to a certificate that's in db, it's going to be happy.That's the conceptual version. What about the real world one? Most x86 systems that implement UEFI secure boot have at least two root certificates in db - one called "Microsoft Windows Production PCA 2011", and one called "Microsoft Corporation UEFI CA 2011". The former is the root of a chain used to sign the Windows bootloader, and the latter is the root used to sign, well, everything else.What is "everything else"? For people in the Linux ecosystem, the most obvious thing is the Shim bootloader that's used to bridge between the Microsoft root of trust and a given Linux distribution's root of trust[2]. But that's not the only third party code executed in the UEFI environment. Graphics cards, network cards, RAID and iSCSI cards and so on all tend to have their own unique initialisation process, and need board-specific drivers. Even if you added support for everything on the market to your system firmware, a system built last year wouldn't know how to drive a graphics card released this year. Cards need to provide their own drivers, and these drivers are stored in flash on the card so they can be updated. But since UEFI doesn't have any sandboxing environment, those drivers could do pretty much anything they wanted to. Someone could compromise the UEFI secure boot chain by just plugging in a card with a malicious driver on it, and have that hotpatch the bootloader and introduce a backdoor into your kernel.This is avoided by enforcing secure boot for these drivers as well. Every plug-in card that carries its own driver has it signed by Microsoft, and up until now that's been a certificate chain going back to the same "Microsoft Corporation UEFI CA 2011" certificate used in signing Shim. This is important for reasons we'll get to.The "Microsoft Windows Production PCA 2011" certificate expires in October 2026, and the "Microsoft Corporation UEFI CA 2011" one in June 2026. These dates are not that far in the future! Most of you have probably at some point tried to visit a website and got an error message telling you that the site's certificate had expired and that it's no longer trusted, and so it's natural to assume that the outcome of time's arrow marching past those expiry dates would be that systems will stop booting. Thankfully, that's not what's going to happen.First up: if you grab a copy of the Shim currently shipped in Fedora and extract the certificates from it, you'll learn it's not directly signed with the "Microsoft Corporation UEFI CA 2011" certificate. Instead, it's signed with a "Microsoft Windows UEFI Driver Publisher" certificate that chains to the "Microsoft Corporation UEFI CA 2011" certificate. That's not unusual, intermediates are commonly used and rotated. But if we look more closely at that certificate, we learn that it was issued in 2023 and expired in 2024. Older versions of Shim were signed with older intermediates. A very large number of Linux systems are already booting certificates that have expired, and yet things keep working. Why?Let's talk about time. In the ways we care about in this discussion, time is a social construct rather than a meaningful reality. There's no way for a computer to observe the state of the universe and know what time it is - it needs to be told. It has no idea whether that time is accurate or an elaborate fiction, and so it can't with any degree of certainty declare that a certificate is valid from an external frame of reference. The failure modes of getting this wrong are also extremely bad! If a system has a GPU that relies on an option ROM, and if you stop trusting the option ROM because either its certificate has genuinely expired or because your clock is wrong, you can't display any graphical output[3] and the user can't fix the clock and, well, crap.The upshot is that nobody actually enforces these expiry dates - here's the reference code that disables it. In a year's time we'll have gone past the expiration date for "Microsoft Windows UEFI Driver Publisher" and everything will still be working, and a few months later "Microsoft Windows Production PCA 2011" will also expire and

systems will keep booting Windows despite being signed with a now-expired certificate. This isn't a Y2K scenario where everything keeps working because people have done a huge amount of work - it's a situation where everything keeps working even if nobody does any work.So, uh, what's the story here? Why is there any engineering effort going on at all? What's all this talk of new certificates? Why are there sensationalist pieces about how Linux is going to stop working on old computers or new computers or maybe all computers?Microsoft will shortly start signing things with a new certificate that chains to a new root, and most systems don't trust that new root. System vendors are supplying updates[4] to their systems to add the new root to the set of trusted keys, and Microsoft has supplied a fallback that can be applied to all systems even without vendor support[5]. If something is signed purely with the new certificate then it won't boot on something that only trusts the old certificate (which shouldn't be a realistic scenario due to the above), but if something is signed purely with the old certificate then it won't boot on something that only trusts the new certificate.How meaningful a risk is this? We don't have an explicit statement from Microsoft as yet as to what's going to happen here, but we expect that there'll be at least a period of time where Microsoft signs binaries with both the old and the new certificate, and in that case those objects should work just fine on both old and new computers. The problem arises if Microsoft stops signing things with the old certificate, at which point new releases will stop booting on systems that don't trust the new key (which, again, shouldn't happen). But even if that does turn out to be a problem, nothing is going to force Linux distributions to stop using existing Shims signed with the old certificate, and having a Shim signed with an old certificate does nothing to stop distributions signing new versions of grub and kernels. In an ideal world we have no reason to ever update Shim[6] and so we just keep on shipping one signed with two certs.If there's a point in the future where Microsoft only signs with the new key, and if we were to somehow end up in a world where systems only trust the old key and not the new key[7], then those systems wouldn't boot with new graphics cards, wouldn't be able to run new versions of Windows, wouldn't be able to run any Linux distros that ship with a Shim signed only with the new certificate. That would be bad, but we have a mechanism to avoid it. On the other hand, systems that only trust the new certificate and not the old one would refuse to boot older Linux, wouldn't support old graphics cards, and also wouldn't boot old versions of Windows. Nobody wants that, and for the foreseeable future we're going to see new systems continue trusting the old certificate and old systems have updates that add the new certificate, and everything will just continue working exactly as it does now.Conclusion: Outside some corner cases, the worst case is you might need to boot an old Linux to update your trusted keys to be able to install a new Linux, and no computer currently running Linux will break in any way whatsoever.[1] (there's also a separate revocation mechanism called SBAT which I wrote about here, but it's not relevant in this scenario)[2] Microsoft won't sign GPLed code for reasons I think are unreasonable, so having them sign grub was a non-starter, but also the point of Shim was to allow distributions to have something that doesn't change often and be able to sign their own bootloaders and kernels and so on without having to have Microsoft involved, which means grub and the kernel can be updated without having to ask Microsoft to sign anything and updates can be pushed without any additional delays[3] It's been a long time since graphics cards booted directly into a state that provided any well-defined programming interface. Even back in 90s, cards didn't present VGA-compatible registers until card-specific code had been executed (hence DEC Alphas having an x86 emulator in their firmware to run the driver on the card). No driver? No video output.[4] There's a UEFI-defined mechanism for updating the keys that doesn't require a full firmware update, and it'll work on all devices that use the same keys rather than being per-device[5] Using the generic update without a vendor-specific update means it wouldn't be possible to issue further updates for the next key rollover, or any additional revocation updates, but I'm hoping to be retired by then and I hope all these computers will also be retired by then[6] I said this in 2012 and it turned out to be wrong then so it's probably wrong now sorry, but at least SBAT means we can revoke vulnerable grubs without having to revoke Shim[7] Which shouldn't

happen! There's an update to add the new key that should work on all PCs, but there's always the chance of firmware bugs comments

- Linux Plumbers Conference: All Microconferences have been Accepted! (2025/07/25 20:12)
  Good news! All Microconferences have been accepted and are now accepting submissions. The accepted Microconferences are: Android Build Systems Confidential Computing Containers and checkpoint/restore Device and Specific Purpose Memory Devicetree Embedded & Internet of Things Gaming on Linux Kernel Memory Management Kernel Testing & Dependability Linux System Monitoring and Observability Live Update Power and Thermal management RISC-V Rust Safe Systems with Linux sched_ext: The BPF extensible scheduler class Scheduler and Real-Time System Boot and Security Toolchains VFIO/IOMMU/PCI x86 You can start submitting topics to these Microconferences. Remember to read the Blog on what makes the ideal Microconference topic before submitting. After that, submit your topic and make sure that you select the appropriate track that you are submitting for (they are all listed under LPC Microconference Proposals and end with MC).
- Dave Airlie (blogspot): ramalama/mesa : benchmarks on my hardware and open source vs proprietary (2025/07/24 22:19)
  One of my pet peeves around running local LLMs and inferencing is the sheer mountain of shit^W^W^W complexity of compute stacks needed to run any of this stuff in an mostly optimal way on a piece of hardware.CUDA, ROCm, and Intel oneAPI all to my mind scream over-engineering on a massive scale at least for a single task like inferencing. The combination of closed source, over the wall open source, and open source that is insurmountable for anyone to support or fix outside the vendor, screams that there has to be a simpler way. Combine that with the pytorch ecosystem and insanity of deploying python and I get a bit unstuck.What can be done about it?llama.cpp to me seems like the best answer to the problem at present, (a rust version would be a personal preference, but can't have everything). I like how ramalama wraps llama.cpp to provide a sane container interface, but I'd like to eventually get to the point where container complexity for a GPU compute stack isn't really needed except for exceptional cases.On the compute stack side, Vulkan exposes most features of GPU hardware in a possibly suboptimal way, but with extensions all can be forgiven. Jeff Bolz from NVIDIA's talk at Vulkanised 2025 started to give me hope that maybe the dream was possible.The main issue I have is Jeff is writing driver code for the NVIDIA proprietary vulkan driver which reduces complexity but doesn't solve my open source problem.Enter NVK, the open source driver for NVIDIA GPUs. Karol Herbst and myself are taking a look at closing the feature gap with the proprietary one. For mesa 25.2 the initial support for VK_KHR_cooperative_matrix was landed, along with some optimisations, but there is a bunch of work to get VK_NV_cooperative_matrix2 and a truckload of compiler optimisations to catch up with NVIDIA.But since mesa 25.2 was coming soon I wanted to try and get some baseline figures out.I benchmarked on two systems (because my AMD 7900XT wouldn't fit in the case). Both Ryzen CPUs. The first I used system I put in an RTX5080 then a RTX6000 Ada and then the Intel A770. The second I used for the RX7900XT. The Intel SYCL stack failed to launch unfortunately inside ramalama and I hacked llama.cpp to use the A770 MMA accelerators. ramalama bench hf://unsloth/Qwen3-8B-GGUF:UD-Q4_K_XL I picked this model at random, and I've no idea if it was a good idea. Some analysis:The token generation workload is a lot less matmul heavy than prompt processing, it also does a lot more synchronising. Jeff has stated CUDA wins here mostly due to CUDA graphs and most of the work needed is operation fusion on the llama.cpp side. Prompt processing is a lot more matmul heavy, extensions like NV_coopmat2 will help with that (NVIDIA vulkan already uses it in the above), but there may be further work to help close the CUDA gap. On AMD radv (open source) Vulkan is already better at TG than ROCm, but behind in prompt processing. Again coopmat2 like extensions should help close the gap there.NVK is starting from a fair way behind, we just pushed support for the most basic coopmat extension and we know there is a long way to go, but I think most of it is achievable as we move forward and I hope to update with new scores on a semi regular basis. We also know we can definitely close the gap on the NVIDIA proprietary Vulkan driver if we apply

enough elbow grease and register allocation :-)I think it might also be worth putting some effort into radv coopmat2 support, I think if radv could overtake ROCm for both of these it would remove a large piece of complexity from the basic users stack.As for Intel I've no real idea, I hope to get their SYCL implementation up and running, and maybe I should try and get my hands on a B580 card as a better baseline. When I had SYCL running once before I kinda remember it being 2-4x the vulkan driver, but there's been development on both sides. (The graphs were generated by Gemini.)

- Pete Zaitcev: Floating Point (2025/07/22 17:28)
  I'm unemployed right now and I go to job interviews once in a while. One time, the company was doing another AI thing, having to do with verifying that training computations were doing something useful, and not just "dumping a stream of floating point numbers".Until now I didn't think of it, but apparently AI is all in FP. And it reminded me how I worked in a CPU design place, where they had a group focused on FP. Those guys were doing FP since the days of transistor. They migrated their designs, generation by generation, through TTL, ECL, Bi-CMOS, CMOS. When I heard from them last, they were tinkering with "deep sub-micron".One remarkable part about their thing was that because they started out in transistors, their FPU didn't have any microcode. It was all in hardware. Even divisions! Just a bunch of counters that sequenced whatever necessary.For a long time during the reign of x86, the group was somewhat de-prioritized, because many microprocessors at the time treated FP performance as an afterthought. A number of desktop CPUs shipped with no hardware FP at all. But look how the tables have turned. I honestly hope that it was not too late and AI has become a boon for the successors of my past colleagues.
- Pete Zaitcev: AI writing (2025/07/12 03:23)
  On the topic of AI writing code, I've read a Sci-Fi story some 30 years ago, probably from the 1950s or 1960s.At the future Earth, fiction writers write using machines. The quality of writing is associated with the sophistication of writer's machine. Publishers reject stories written on a lower end machine. The hero of the story is a struggling writer, who has to make do with a cheap unit. As his machine writes poorly, he's paid little, so he cannot save up for an upgrade. He hatches a plan to sneak into the house of a successful writer, and use the better machine to write a break-out story. The oddly prescient punch-line is, he discovers that the successful writer's machine was non-functional. His better wrote his stories manually in secret.I wonder if such scenario may even be possible, programming-wise, if you work in a company that does not micro-manage velocity, or work as a consultant, so that your sausage factory remains behind a curtain.
- Harald Welte: Security Issues regarding GSMA eSIMs / eUICCs + Javacard (2025/07/08 22:00)
  The independent security researcher Adam Gowdiak has published an extensive report on flaws he found in some eUICCs (the chips used to store eSIM profiles within the GSMA eSIM architecture). While the specific demonstrable exploit was in a product of one specific CardOS Vendor (Kigen, formerly part of ARM), the fundamental underlying issue is actually an architectural one. The Oracle Javacard [memory] safety architecture relies on a so-called bytecode verifier which is a program that you run after compiling an application, but before executing the code on the Javacard. The specifications allow for both on-card and off-card verification. However, the computational complexity of this verifier is generally assumed to exceed the resources available inside many microcontrollers used to implement java cards. Such microcontrollers often are ARM SC000 (Cortex-M0 based) or SC300 (Cortex-M3 based) based, with only tens of kilobytes of RAM and hundreds of kilobytes of flash. Javacard was originally developed for use cases within the banking/payment industry. In that industry, the card-issuing bank is the sole entity that has the keys to load java applets onto a card. That entity is of course interested in the security of the card, and will hence always run an off-card bytecode verifier. In a world of physical SIM/USIM cards issued by a single mobile operator, the situation is the same: The card-issuing MNO/MVNO is the only entity

with key materials to install additional java applets on the card. This fundamental problem became already apparent by earlier findings by Adam Gowdiak in 2019, but at least in terms of public responses by Oracle and Gemalto back then, they mostly did hand-waving and/or made lame excuses. However, when the industry represented in GSMA standardized the eSIM architecture, this changed. Suddenly we have various eSIM profiles of various different operators, each holding key material to install Java applets on the shared card. In such an environment, it is no longer safe to assume that every MNO/MVNO can be trusted to be non-adverserial and hence trusted to run that off-card bytecode verifier before loading applets onto the card. If the Javacard runtime on the existing card/chip itself cannot autonomously perform those verification tasks, I don't see how the problem can ever be solved short of completely removing/disabling Javacard support in such eUICCs. Luckily it is an optional feature and not a mandatory requirement for an eUICC to be approved/accredited. Sadly many MNOs/MVNOS however will mandate Javacard support in their eSIM profiles and hence refuse to install into an eUICC without it :(In my opinion, the solution to the problem can only be to either make the GSMA require full on-card bytecode verification on all eUICCs, or to remove Javacard support from the eUICC. We have to keep in mind that there are hundreds if not thousands of MVNOs around the planet, and all of them are subject to whatever local jurisdiction they operate in, and also subject to whatever government pressure (e.g from intelligence agencies). In hindsight, anyone familiar with the 2019 work by Gowdiak and an understanding of the fundamental change to multiple stakeholders in an eUICC (compared to classic SIM/USIM) should have arrived at the conclusion that there is a serious problem that needs addressing. I think the 2019 work had not been publicized and covered significantly enough to make sure that everyone in the industry was made aware of the problems. And that in turn is mostly a result of Oracle + Gemalto downplaying the 2019 findings back in the day, rather than raising awareness within all relevant groups and bodies of the industry. Mitigation via TS.48 key diversification The specific attack presented was using a GSMA TS.48 test profile to install the malicious java bytecode; those TS.48 profiles are standardized profiles used by the industry for cellular testing; until the attack they contained well-known static OTA key material. The mitigation to randomize/diversity those keys in TS.48v7 closes that particular vector, but the attack itself is not dependent on test profiles. Any MNO/NVNO (or rather, anyone with access to a commercial service of a SM-DP+ accredited by GSMA) obviously has the ability to load java applets into the eSIM profile that they create, using keys that they themselves specify. What IMHO ought to be done Oracle should get off their we only provide a reference implementation and vendors should invent their own prporietary verification mechanisms horse. This is just covering their own ass and not helping any of their downstream users/implementers. The reference implementation should show how proper verification can be done in the most resource-constrained environment of cards (it's JavaCard, after all!), and any advances of the verifier should happen once at Oracle, and then used by all the implementers (CardOS vendors). Anyone who really cares about security of a standardized platform (like Javacard) should never leave key aspects of it up to each and every implementer, but rather should solve the problem once, publicly, with validation and testing tools, independent 3rd party penetration testing and then ensure that every implementer uses that proven implementation. GSMA should have security requirements (and mandatory penetration tests) specifically regarding the JVM/JRE of each card that gets SAS-UP accredited. GSMA should require that Javacard support should be disabled on all existing eUICCs that cannot legitimately claim/demonstrate that they are performing full bytecode verification entirely on-card. GSMA should refuse any future SAS-UP accreditation to any product that requires off-card bytecode verification The entire industry should find a way to think beyond Javacard, or in fact any technology whose security requires verification of the executable program that is too complex to perform on-card on the targeted microcontrollers.

- James Morris: Where Else to Find Me (2025/07/04 19:59)
  I’m not blogging much these days, and more likely posting on these accounts: Fediverse (“Mastodon”): https://social.kernel.org/jmorris Bluesky:

https://bsky.app/profile/jamesmorris.bsky.social If you'd like to follow updates for the Linux Security Summit (LSS), see here: https://social.kernel.org/LinuxSecSummit For topics which are specifically $work related, see my LinkedIn: https://www.linkedin.com/in/jamesmorris/

- Dave Airlie (blogspot): nvk: blackwell support (2025/07/01 10:20)
  Blog posts are like buses sometimes...I've spent time over the last month enabling Blackwell support on NVK, the Mesa vulkan driver for NVIDIA GPUs. Faith from Collabora, the NVK maintainer has cleaned up and merged all the major pieces of this work and landed them into mesa this week. Mesa 25.2 should ship with a functioning NVK on blackwell. The code currently in mesa main passes all tests in the Vulkan CTS.Quick summary of the major fun points:Ben @ NVIDIA had done the initial kernel bringup in to r570 firmware in the nouveau driver. I worked with Ben on solidifying that work and ironing out a bunch of memory leaks and regressions that snuck in.Once the kernel was stable, there were a number of differences between Ada and Blackwell that needed to be resolved. Thanks to Faith, Mel and Mohamed for their help, and NVIDIA for providing headers and other info.I did most of the work on a GB203 laptop and a desktop 5080.1. Instruction encoding: a bunch of instructions changed how they were encoded. Mel helped sort out most of those early on.2. Compute/QMD: the QMD which is used to launch compute shaders, has a new encoding. NVIDIA released the official QMD headers which made this easier in the end.3. Texture headers: texture headers were encoded different from Hopper on, so we had to use new NVIDIA headers to encode those properly4. Depth/Stencil: NVIDIA added support for separate d/s planes and this also has some knock on effects on surface layouts. 5. Surface layout changes. NVIDIA attaches a memory kind to memory allocations, due to changes in Blackwell, they now use a generic kind for all allocations. You now longer know the internal bpp dependent layout of the surfaces. This means changes to the dma-copy engine to provide that info. This means we have some modifier changes to cook with NVIDIA over the next few weeks at least for 8/16 bpp surfaces. Mohamed helped get this work and host image copy support done.6. One thing we haven't merged is bound texture support. Currently blackwell is using bindless textures which might be a little slower. Due to changes in the texture instruction encoding, you have to load texture handles to intermediate uniform registers before using them as bound handles. This causes a lot of fun with flow control and when you can spill uniform registers. I've written a few efforts at using bound textures, so we understand how to use them, just have some compiler issues to maybe get it across the line.7. Proper instruction scheduling isn't landed yet. I have a spreadsheet with all the figures, and I started typing, so will try and get that into an MR before I take some holidays.
- Dave Airlie (blogspot): radv: VK_KHR_video_encode_av1 support (2025/07/01 09:27)
  I should have mentioned this here a week ago. The Vulkan AV1 encode extension has been out for a while, and I'd done the initial work on enabling it with radv on AMD GPUs. I then left it in a branch, which Benjamin from AMD picked up and fixed a bunch of bugs, and then we both got distracted. I realised when doing VP9 that it hasn't landed, so did a bit of cleanup. Then David from AMD picked it up and carried it over the last mile and it got merged last week.So radv on supported hw now supports all vulkan decode/encode formats currently available.
- Matthew Garrett: Why is there no consistent single signon API flow? (2025/06/24 06:03)
  Single signon is a pretty vital part of modern enterprise security. You have users who need access to a bewildering array of services, and you want to be able to avoid the fallout of one of those services being compromised and your users having to change their passwords everywhere (because they're clearly going to be using the same password everywhere), or you want to be able to enforce some reasonable MFA policy without needing to configure it in 300 different places, or you want to be able to disable all user access in one place when someone leaves the company, or, well, all of the above. There's any number of providers for this, ranging from it being integrated with a more general app service

platform (eg, Microsoft or Google) or a third party vendor (Okta, Ping, any number of bizarre companies). And, in general, they'll offer a straightforward mechanism to either issue OIDC tokens or manage SAML login flows, requiring users present whatever set of authentication mechanisms you've configured.This is largely optimised for web authentication, which doesn't seem like a huge deal - if I'm logging into Workday then being bounced to another site for auth seems entirely reasonable. The problem is when you're trying to gate access to a non-web app, at which point consistency in login flow is usually achieved by spawning a browser and somehow managing submitting the result back to the remote server. And this makes some degree of sense - browsers are where webauthn token support tends to live, and it also ensures the user always has the same experience.But it works poorly for CLI-based setups. There's basically two options - you can use the device code authorisation flow, where you perform authentication on what is nominally a separate machine to the one requesting it (but in this case is actually the same) and as a result end up with a straightforward mechanism to have your users socially engineered into giving Johnny Badman a valid auth token despite webauthn nominally being unphisable (as described years ago), or you reduce that risk somewhat by spawning a local server and POSTing the token back to it - which works locally but doesn't work well if you're dealing with trying to auth on a remote device. The user experience for both scenarios sucks, and it reduces a bunch of the worthwhile security properties that modern MFA supposedly gives us.There's a third approach, which is in some ways the obviously good approach and in other ways is obviously a screaming nightmare. All the browser is doing is sending a bunch of requests to a remote service and handling the response locally. Why don't we just do the same? Okta, for instance, has an API for auth. We just need to submit the username and password to that and see what answer comes back. This is great until you enable any kind of MFA, at which point the additional authz step is something that's only supported via the browser. And basically everyone else is the same.Of course, when we say "That's only supported via the browser", the browser is still just running some code of some form and we can figure out what it's doing and do the same. Which is how you end up scraping constants out of Javascript embedded in the API response in order to submit that data back in the appropriate way. This is all possible but it's incredibly annoying and fragile - the contract with the identity provider is that a browser is pointed at a URL, not that any of the internal implementation remains consistent.I've done this. I've implemented code to scrape an identity provider's auth responses to extract the webauthn challenges and feed those to a local security token without using a browser. I've also written support for forwarding those challenges over the SSH agent protocol to make this work with remote systems that aren't running a GUI. This week I'm working on doing the same again, because every identity provider does all of this differently.There's no fundamental reason all of this needs to be custom. It could be a straightforward "POST username and password, receive list of UUIDs describing MFA mechanisms, define how those MFA mechanisms work". That even gives space for custom auth factors (I'm looking at you, Okta Fastpass). But instead I'm left scraping JSON blobs out of Javascript and hoping nobody renames a field, even though I only care about extremely standard MFA mechanisms that shouldn't differ across different identity providers.Someone, please, write a spec for this. Please don't make it be me.
comments

- Matthew Garrett: My a11y journey (2025/06/20 08:48)
  23 years ago I was in a bad place. I'd quit my first attempt at a PhD for various reasons that were, with hindsight, bad, and I was suddenly entirely aimless. I lucked into picking up a sysadmin role back at TCM where I'd spent a summer a year before, but that's not really what I wanted in my life. And then Hanna mentioned that her PhD supervisor was looking for someone familiar with Linux to work on making Dasher, one of the group's research projects, more usable on Linux. I jumped.The timing was fortuitous. Sun were pumping money and developer effort into accessibility support, and the Inference Group had just received a grant from the Gatsy Foundation that involved working with the ACE Centre to

provide additional accessibility support. And I was suddenly hacking on code that was largely ignored by most developers, supporting use cases that were irrelevant to most developers. Being in a relatively green field space sounds refreshing, until you realise that you're catering to actual humans who are potentially going to rely on your software to be able to communicate. That's somewhat focusing.This was, uh, something of an on the job learning experience. I had to catch up with a lot of new technologies very quickly, but that wasn't the hard bit - what was difficult was realising I had to cater to people who were dealing with use cases that I had no experience of whatsoever. Dasher was extended to allow text entry into applications without needing to cut and paste. We added support for introspection of the current applications UI so menus could be exposed via the Dasher interface, allowing people to fly through menu hierarchies and pop open file dialogs. Text-to-speech was incorporated so people could rapidly enter sentences and have them spoke out loud.But what sticks with me isn't the tech, or even the opportunities it gave me to meet other people working on the Linux desktop and forge friendships that still exist. It was the cases where I had the opportunity to work with people who could use Dasher as a tool to increase their ability to communicate with the outside world, whose lives were transformed for the better because of what we'd produced. Watching someone use your code and realising that you could write a three line patch that had a significant impact on the speed they could talk to other people is an incomparable experience. It's been decades and in many ways that was the most impact I've ever had as a developer.I left after a year to work on fruitflies and get my PhD, and my career since then hasn't involved a lot of accessibility work. But it's stuck with me - every improvement in that space is something that has a direct impact on the quality of life of more people than you expect, but is also something that goes almost unrecognised. The people working on accessibility are heroes. They're making all the technology everyone else produces available to people who would otherwise be blocked from it. They deserve recognition, and they deserve a lot more support than they have.But when we deal with technology, we deal with transitions. A lot of the Linux accessibility support depended on X11 behaviour that is now widely regarded as a set of misfeatures. It's not actually good to be able to inject arbitrary input into an arbitrary window, and it's not good to be able to arbitrarily scrape out its contents. X11 never had a model to permit this for accessibility tooling while blocking it for other code. Wayland does, but suffers from the surrounding infrastructure not being well developed yet. We're seeing that happen now, though - Gnome has been performing a great deal of work in this respect, and KDE is picking that up as well. There isn't a full correspondence between X11-based Linux accessibility support and Wayland, but for many users the Wayland accessibility infrastructure is already better than with X11.That's going to continue improving, and it'll improve faster with broader support. We've somehow ended up with the bizarre politicisation of Wayland as being some sort of woke thing while X11 represents the Roman Empire or some such bullshit, but the reality is that there is no story for improving accessibility support under X11 and sticking to X11 is going to end up reducing the accessibility of a platform.When you read anything about Linux accessibility, ask yourself whether you're reading something written by either a user of the accessibility features, or a developer of them. If they're neither, ask yourself why they actually care and what they're doing to make the future better. comments

- Dave Airlie (blogspot): radv: vulkan VP9 video decode (2025/06/09 19:42)
  The Vulkan WG has released VK_KHR_video_decode_vp9. I did initial work on a Mesa extensions for this a good while back, and I've updated the radv code with help from AMD and Igalia to the final specification.There is an open MR[1] for radv to add support for vp9 decoding on navi10+ with the latest firmware images in linux-firmware. It is currently passing all VK-GL-CTS tests for VP9 decode.Adding this decode extension is a big milestone for me as I think it now covers all the reasons I originally got involved in Vulkan Video as signed off, there is still lots to do and I'll stay involved, but it's been great to see the contributions from others and how there is a bit of Vulkan Video community upstream in Mesa. [1]

https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/35398

- Matthew Garrett: Twitter's new encrypted DMs aren't better than the old ones (2025/06/05 13:44)
  (Edit: Twitter could improve this significantly with very few changes - I wrote about that here. It's unclear why they'd launch without doing that, since it entirely defeats the point of using HSMs)When Twitter[1] launched encrypted DMs a couple of years ago, it was the worst kind of end-to-end encrypted - technically e2ee, but in a way that made it relatively easy for Twitter to inject new encryption keys and get everyone's messages anyway. It was also lacking a whole bunch of features such as "sending pictures", so the entire thing was largely a waste of time. But a couple of days ago, Elon announced the arrival of "XChat", a new encrypted message platform built on Rust with (Bitcoin style) encryption, whole new architecture. Maybe this time they've got it right?tl;dr - no. Use Signal. Twitter can probably obtain your private keys, and admit that they can MITM you and have full access to your metadata.The new approach is pretty similar to the old one in that it's based on pretty straightforward and well tested cryptographic primitives, but merely using good cryptography doesn't mean you end up with a good solution. This time they've pivoted away from using the underlying cryptographic primitives directly and into higher level abstractions, which is probably a good thing. They're using Libsodium's boxes for message encryption, which is, well, fine? It doesn't offer forward secrecy (if someone's private key is leaked then all existing messages can be decrypted) so it's a long way from the state of the art for a messaging client (Signal's had forward secrecy for over a decade!), but it's not inherently broken or anything. It is, however, written in C, not Rust[2].That's about the extent of the good news. Twitter's old implementation involved clients generating keypairs and pushing the public key to Twitter. Each client (a physical device or a browser instance) had its own private key, and messages were simply encrypted to every public key associated with an account. This meant that new devices couldn't decrypt old messages, and also meant there was a maximum number of supported devices and terrible scaling issues and it was pretty bad. The new approach generates a keypair and then stores the private key using the Juicebox protocol. Other devices can then retrieve the private key. Doesn't this mean Twitter has the private key? Well, no. There's a PIN involved, and the PIN is used to generate an encryption key. The stored copy of the private key is encrypted with that key, so if you don't know the PIN you can't decrypt the key. So we brute force the PIN, right? Juicebox actually protects against that - before the backend will hand over the encrypted key, you have to prove knowledge of the PIN to it (this is done in a clever way that doesn't directly reveal the PIN to the backend). If you ask for the key too many times while providing the wrong PIN, access is locked down.But this is true only if the Juicebox backend is trustworthy. If the backend is controlled by someone untrustworthy[3] then they're going to be able to obtain the encrypted key material (even if it's in an HSM, they can simply watch what comes out of the HSM when the user authenticates if there's no validation of the HSM's keys). And now all they need is the PIN. Turning the PIN into an encryption key is done using the Argon2id key derivation function, using 32 iterations and a memory cost of 16MB (the Juicebox white paper says 16KB, but (a) that's laughably small and (b) the code says 16 * 1024 in an argument that takes kilobytes), which makes it computationally and moderately memory expensive to generate the encryption key used to decrypt the private key. How expensive? Well, on my (not very fast) laptop, that takes less than 0.2 seconds. How many attempts to I need to crack the PIN? Twitter's chosen to fix that to 4 digits, so a maximum of 10,000. You aren't going to need many machines running in parallel to bring this down to a very small amount of time, at which point private keys can, to a first approximation, be extracted at will.Juicebox attempts to defend against this by supporting sharding your key over multiple backends, and only requiring a subset of those to recover the original. I can't find any evidence that Twitter's does seem to be making use of this,Twitter uses three backends and requires data from at least two, but all the backends used are under x.com so are presumably under Twitter's direct control. Trusting the keystore without needing to trust whoever's hosting it requires a trustworthy

communications mechanism between the client and the keystore. If the device you're talking to can prove that it's an HSM that implements the attempt limiting protocol and has no other mechanism to export the data, this can be made to work. Signal makes use of something along these lines using Intel SGX for contact list and settings storage and recovery, and Google and Apple also have documentation about how they handle this in ways that make it difficult for them to obtain backed up key material. Twitter has no documentation of this, and as far as I can tell does nothing to prove that the backend is in any way trustworthy. (Edit to add: The Juicebox API does support authenticated communication between the client and the HSM, but that relies on you having some way to prove that the public key you're presented with corresponds to a private key that only exists in the HSM. Twitter gives you the public key whenever you communicate with them, so even if they've implemented this properly you can't prove they haven't made up a new key and MITMed you the next time you retrieve your key)On the plus side, Juicebox is written in Rust, so Elon's not 100% wrong. Just mostly wrong.But ok, at least you've got viable end-to-end encryption even if someone can put in some (not all that much, really) effort to obtain your private key and render it all pointless? Actually no, since you're still relying on the Twitter server to give you the public key of the other party and there's no out of band mechanism to do that or verify the authenticity of that public key at present. Twitter can simply give you a public key where they control the private key, decrypt the message, and then reencrypt it with the intended recipient's key and pass it on. The support page makes it clear that this is a known shortcoming and that it'll be fixed at some point, but they said that about the original encrypted DM support and it never was, so that's probably dependent on whether Elon gets distracted by something else again. And the server knows who and when you're messaging even if they haven't bothered to break your private key, so there's a lot of metadata leakage.Signal doesn't have these shortcomings. Use Signal.[1] I'll respect their name change once Elon respects his daughter[2] There are implementations written in Rust, but Twitter's using the C one with these JNI bindings [3] Or someone nominally trustworthy but who's been compelled to act against your interests - even if Elon were absolutely committed to protecting all his users, his overarching goals for Twitter require him to have legal presence in multiple jurisdictions that are not necessarily above placing employees in physical danger if there's a perception that they could obtain someone's encryption keys comments

- Matthew Garrett: How Twitter could (somewhat) fix their encrypted DMs (2025/06/05 13:18)
  As I wrote in my last post, Twitter's new encrypted DM infrastructure is pretty awful. But the amount of work required to make it somewhat better isn't large.When Juicebox is used with HSMs, it supports encrypting the communication between the client and the backend. This is handled by generating a unique keypair for each HSM. The public key is provided to the client, while the private key remains within the HSM. Even if you can see the traffic sent to the HSM, it's encrypted using the Noise protocol and so the user's encrypted secret data can't be retrieved.But this is only useful if you know that the public key corresponds to a private key in the HSM! Right now there's no way to know this, but there's worse - the client doesn't have the public key built into it, it's supplied as a response to an API request made to Twitter's servers. Even if the current keys are associated with the HSMs, Twitter could swap them out with ones that aren't, terminate the encrypted connection at their endpoint, and then fake your query to the HSM and get the encrypted data that way. Worse, this could be done for specific targeted users, without any indication to the user that this has happened, making it almost impossible to detect in general.This is at least partially fixable. Twitter could prove to a third party that their Juicebox keys were generated in an HSM, and the key material could be moved into clients. This makes attacking individual users more difficult (the backdoor code would need to be shipped in the public client), but can't easily help with the website version[1] even if a framework exists to analyse the clients and verify that the correct public keys are in use.It's still worse than Signal. Use Signal.[1] Since they could still just serve backdoored Javascript to specific users. This is, unfortunately, kind of an inherent problem when it comes to web-based

clients - we don't have good frameworks to detect whether the site itself is malicious. comments

- Brendan Gregg: 3 Years of Extremely Remote Work (2025/05/21 14:00)
  In the last 3 years I've attended 77 meetings that began between 1am and 6am, roughly once every two weeks, followed by my usual 7am start, Monday to Saturday. I'm working remotely from Australia for a US firm (Intel) who does not have a local office here. I'm not complaining. I work weird hours, but I don't think I work too many. I'm writing this post because there are some misconceptions and assumptions about the lives of remote workers, and I thought I'd share my own details as an anecdote, along with some tips for others in a similar situation (US job from Asia). Most early meetings were 1 hour, but some were longer, for a total of 102 hours awake. As a histogram: I've never been a morning person, but I can manage a 7am start. What about a 2am meeting? Also doable. They sound ok in isolation, but consider that every 2-3am is followed by a 7am start, which means a 6:30am alarm after going back to sleep at 3:30am, if you're lucky. It's not easy working this timezone difference, and I'd guess others have it even worse than I do (more meetings). Like other remote staff I work with, I have a dedicated office at home where I can close the door and work uninterrupted for hours (it's changed quite a bit since it was filmed in the eBPF documentary): That's a Samson Meteor mic on a desktop suspension boom, and I have another suspension boom clamped onto a bookcase holding a Logitech BRIO camera (when I had it sitting on my monitor it'd shake as I typed). It's important to have the best sound and video when that's how you're interacting with people all day. Miscellaneous notes and tips about remote work: Count out-of-hours meetings. Sometimes people hesitate to book me at 2am, when there's no other time that would work, and it takes a few emails to convince them that it's ok. I've found it saves time to log these meetings, count them, and share stats: "I've had 76 meetings between 1am and 6am, if you need to add another, that's ok, it'll be my 77th." Never complain about the hours. There may be someone in management who doesn't support remote work, who could use such complains to argue against it. Sometimes, when I'm searching to find the right words to say at 4-something-am, I've confessed that I'm a bit tired, but I really try to never mention it. Staying motivated. I found keeping a daily log of what I accomplished works best (I've done this for over a decade). If one day my entry looks soft, I try to do more on the next. I summarize each week for my manager. People assume it's a problem when it isn't. Brendan didn't accept the meeting, oh, it's because it's 2am for him and he'll be asleep. Wrong! It's because I have a clash with another 2am meeting. I try to communicate this with the meeting host, but there can be others in the meeting that don't get the message, and they never consider this possibility. If I were back in California I'd still miss the meeting, but people would assume it's due to a clash. Here, they assume I'm asleep and not making the effort, when in fact I am. It means people are assuming that remote work is a problem when it isn't. Some meetings get cancelled or silently recorded. The 77 count I mentioned earlier doesn't include the several meetings that were cancelled at the last minute, but I woke up for anyway, or those that weren't supposed to be recorded but I woke up to find they were. If you have remote staff, please try to cancel meetings before they go to bed, and be clear on which meetings are recorded for later viewing. Upset stomaches. One early meeting every two weeks doesn't sound too bad. The worst problem is that it can leave me with an upset stomach that can last for days. I'm still working fine, it's just uncomfortable. I don't know if other people suffer this or why it happens. Maybe it's just the extra coffee. Fewer sick days. I don't normally take many sick days, so I can't say my data is very significant. FWIW: In my last in-person job I averaged 1.5 sick days/year (9 in 6 years), and now it's 0.33 (1 in 3 years). Northen/Southern hemisphere times get weird. Daylight savings moves in different directions and on different dates, so from Sydney depending on the time of year I have 3, 4, or 5 hours of overlap with 9-5pm on the US west coast. To make it easy for people I setup my calendar with my work hours highlighted. Saturday? Saturday morning is Friday afternoon in the US. For a while I tried working Tuesday to Saturday, but it's better for family time if I finish early on Saturday (at US 5pm) and work Monday to make up the difference. Phone alarms Career limiting I've

experienced it to be career limiting, and I've heard the saying "out of sight, out of mind." Opportunities can be given to local workers despite the remote worker being more qualified, or even number one in the world in that area. The remote worker is then expected to teach the local worker in weekly or monthly meetings. It's not enough time and there's a power imbalance: the local worker is in charge and doesn't have to listen. This reduces company competitiveness. It's also fixable: try giving remote workers the chance they are best qualified to do. Compared to office work. It depends on the job and meeting load. In my last years as an in-person office worker I was on a team where we typically worked solo on different projects, with headphones on, and primarily communicated over chatrooms with few in-person meetings. The only regular time we had human interaction was lunch. Remote work has not been a big difference to that kind of job: similar work, similar chatrooms. (The thing I miss the most about the office is playing cricket with other staff after work.) It wouldn't make a big difference to my current job either, as the people I work with are scattered. The one thing it would solve is "out of sight" problem, but that's not the only way to solve it. Remote work success stories. Linux development is a great example of engineers around the world working together. Note that Linux engineers do try to meet at least once a year at one of the Linux conferences, something remote office workers can do as well at company get-togethers. My books are another example: they are out-of-hours projects where I worked with the reviewers online, some of whom I still haven't met. And the world's first AI Flame Graphs is the work of a fully-remote team. I was in a video meeting recently with US staff where I mentioned the "77 meetings between 1 and 6am" statistic, and I could see the look of shock in their faces. Did they assume I was just working 9-5 and not making an effort to accommodate other timezones? I know some companies are discussing ending remote-work: are the deciders also making such assumptions about the lives of remote workers? It may not do much, but I can at least share my own details here as an example anecdote. Update: Some comments about this post have pointed out that these hours can be unhealthy and shouldn't be encouraged, and I don't disagree. I hope to have fewer early meetings in the years ahead, myself. This post is just to show that remote workers can be more accomodating than people may assume, and I hope that's taken into consideration when changing remote work policies. When some people hear I'm working from Australia, they may think of beaches and surfboards and sunny vacations, but my reality is a full time job across weird hours, lots of coffee, and being overlooked for opportunities. That said, every job has its pros and cons, and I'm still grateful that some companies make fully remote work an option.

- Linux Plumbers Conference: Submission time for Linux Plumbers 2025 (2025/05/14 20:44)
  Submissions for the Refereed Track and Microconferences are now open. Linux Plumbers will be held this year in Tokyo from December 11th – 13th (Note, the 13th is on a Saturday). The Refereed presentations are 45 minutes in length and should focus on a specific aspect of the “plumbing” in a Linux ecosystem. Examples of Linux plumbing include core kernel subsystems, init systems, core libraries, toolchains, windowing systems, management tools, device support, media creation/playback, testing, and so on. The best presentations are not about finished work, but rather problem statements, proposals, or proof-of-concept solutions that require face-to-face discussions and debate. The Microconferences are 3 and a half hours of technical discussion, broken up into 15 to 30 minute subtopics. The only presentations allowed are those that are needed to bring the audience up to speed and should not last more than half the allotted time for the subtopic. To submit a Microconference, provide a topic, some examples of subtopics to be discussed and a list of key people that should be present to have meaningful discussions. For Microconferences that have been to Linux Plumbers in the past, they should provide a list of accomplishments that were a direct result of the discussions from their previous sessions (with links to patches and such). Presentations and Microconference subtopic leads should ideally be physically present at the conference. Remote presentations may be available but are strongly discouraged. The Refereed submissions end at 11:59PM UTC on Wednesday, September 10, 2025. The Microconference submissions end at 11:59PM UTC on Sunday, June 29, 2025. Go ahead

and submit your Refereed track presentation or Microconference topic. We are looking forward to seeing the great content that is submitted that makes Linux Plumbers the best technical conference there is.

- Brendan Gregg: Doom GPU Flame Graphs (2025/04/30 14:00)
  AI Flame Graphs are now open source and include Intel Battlemage GPU support, which means it can also generate full-stack GPU flame graphs for providing new insights into gaming performance, especially when coupled with FlameScope (an older open source project of mine). Here's an example of GZDoom, and I'll start with flame scopes for both CPU and GPU utilization, with details annotated: (Here are the raw CPU and GPU versions.) FlameScope shows a subsecond-offset heatmap of profile samples, where each column is one second (in this example, made up of 50 x 20ms blocks) and the color depth represents the number of samples, revealing variance and perturbation that you can select to generate a flame graph just for that time range. Update: the row size can be ajusted (it is limited by the sample rate captured in the profile), e.g., you could generate 60 rows to match 60fps games. Putting these CPU and GPU flame scopes side by side has enabled your eyes to do pattern matching to solve what would otherwise be a time-consuming task of performance correlation. The gaps in the GPU flame scope on the right – where the GPU was not doing much work – match the heavier periods of CPU work on the left. CPU Analysis FlameScope lets us click on the interesting periods. By selecting one of the CPU shader compilation stripes we get the flame graph just for that range: This is brilliant, and we can see exactly why the CPUs were busy for about 180 ms (the vertical length of the red stripe): it's doing compilation of GPU shaders and some NIR preprocessing (optimizations to the NIR intermediate representation that Mesa uses internally). If you are new to flame graphs, you look for the widest towers and optimize them first. Here is the interactive SVG. CPU flame graphs and CPU flame scope aren't new (from 2011 and 2018, both open source). What is new is full-stack GPU flame graphs and GPU flame scope. GPU Analysis Interesting details can also be selected in the GPU FlameScope for generating GPU flame graphs. This example selects the "room 3" range, which is a room in the Doom map that contains hundreds of enemies. The green frames are the actual instructions running on the GPU, aqua shows the source for these functions, and red (C) and yellow (C++) show the CPU code paths that initiated the GPU programs. The gray "-" frames just help highlight the boundary between CPU and GPU code. (This is similar to what I described in the AI flame graphs post, which included extra frames for kernel code.) The x-axis is proportional to cost, so you look for the widest things and find ways to reduce them. I've included the interactive SVG version of this flame graph so you can mouse-over elements and click to zoom. (PNG version.) The GPU flame graph is split between stalls coming from rendering walls (41.4%), postprocessing effects (35.7%), stenciling (17.2%), and sprites (4.95%). The CPU stacks are further differentiated by the individual shaders that are causing stalls, along with the reasons for those stalls. GZDoom We picked GZDoom to try since it's an open source version of a well known game that runs on Linux (our profiler does not support Windows yet). Intel Battlemage makes light work of GZDoom, however, and since the GPU profile is stall-based we weren't getting many samples. We could have switched to a more modern and GPU-demanding game, but didn't have any great open source ideas, so I figured we'd just make GZDoom more demanding. We built GPU demanding maps for GZDoom (I can't believe I have found a work-related reason to be using Slade), and also set some Battlemage tunables to limit resources, magnifying the utilization of remaining resources. Our GZDoom test map has three rooms: room 1 is empty, room 2 is filled with torches, and room 3 is open with a large skybox and filled with enemies, including spawnpoints for Sergeants. This gave us a few different workloads to examine by walking between the rooms. Using iaprof: Intel's open source accelerator profiler The AI Flame Graph project is pioneering work, and has needed various changes to graphics compilers, libraries, and kernel drivers, not just the code but also how they are built. Since Intel has its own public cloud (the Intel® Tiber™ AI Cloud) we can fix the software stack in advance so that for customers it "just works." Check the available releases. It currently supports the Intel

Max Series GPU. If you aren't on the Intel cloud, or you wish to try this with Intel Battlemage, then it can require a lot of work to get the system ready to be profiled. Requirements include: A Linux system with superuser (root) access, so that eBPF and Intel eustalls can be used. A newer Linux kernel with the latest Intel GPU drivers. For Intel Battlemage this means Linux 6.15+ with the Xe driver; For the Intel Max Series GPU it's Linux 5.15 with the i915 driver. The Linux kernel built with Intel driver-specific eustall and eudebug interfaces (see the github docs for details). Some of these modifications are upstreamed in the latest versions of Linux and others are currently in progress. (These interfaces are made available by default on the Intel® Tiber™ AI Cloud.) All system libraries or programs that are being profiled need to include frame pointers so that the full stacks are visible, including Intel's oneAPI and graphics libraries. For this example, GZDoom itself needed to be compiled with frame pointers and also all libraries used by GZDoom (glibc, etc.). This is getting easier in the lastest versions of Fedora and Ubuntu (e.g., Ubuntu 24.04 LTS) which are shipping system libraries with frame pointers by default. But I'd expect there will be applications and dependencies that don't have frame pointers yet, and need recompilation. If your flame graph has areas that are very short, one or two frames deep, this is why. If you are new to custom kernel builds and library tinkering, then getting this all working may feel like Nightmare! difficulty. Over time things will improve and gradually get easier: check the github docs. Intel can also develop a much easier version of this tool as part of a broader product offering and get it working on more than just Linux and Battlemage (either watch this space or, if you have an Intel rep, ask them to make it a priority). Once you have it all working, you can run the iaprof command to profile the GPU. E.g.: git clone --recursive https://github.com/intel/iaprof cd iaprof make deps make sudo iaprof record > profile.txt cat profile.txt | iaprof flame > flame.svg iaprof is modeled on the Linux perf command. (Maybe one day it'll become included in perf directly.) Thanks to Gabriel Muñoz for getting the work done to get this open sourced. FAQ and Future Work From the launch of AI flame graphs last year, I can guess what FAQ #1 will be: “What about NVIDIA?”. They do have flame graphs in Nsight Graphics for GPU workloads, although their flame graphs are currently shallow as it is GPU code only, and onerous to use as I believe it requires an interposer; on the plus side they have click-to-source. The new GPU profiling method we've been developing allows for easy, everything, anytime profiling, like you expect from CPU profilers. Future work will include github releases, more hardware support, and overhead reduction. We're the first to use eustalls in this way, and we need to add more optimization to reach our target of <5% overhead, especially with the i915 driver. Conclusion We've open sourced AI flame graphs and tested it on new hardware, Intel Battlemage, and a non-AI workload: GZDoom (gaming). It's great to see a view of both CPU and GPU resources down to millisecond resolution, where we can see visual patterns in the flame scope heat maps that can be selected to produce flame graphs to show the code. We applied these new tools to GZDoom and explained GPU pauses by selecting the corresponding CPU burst and reading the flame graph, as well as GPU code use for arbitrary time windows. While we have open sourced this, getting it all running requires Intel hardware and Linux kernel and library tinkering – which can be a lot of work. (Actually playing Doom on Nightmare! difficulty may be easier.) This will get better over time. We look forward to seeing if anyone can fight their way through this work in the meantime and what new performance issues they can solve. Authors: Brendan Gregg, Ben Olson, Brandon Kammerdiener, Gabriel Muñoz.

- Matthew Garrett: Failing upwards: the Twitter encrypted DM failure (2025/03/18 23:58)
  Almost two years ago, Twitter launched encrypted direct messages. I wrote about their technical implementation at the time, and to the best of my knowledge nothing has changed. The short story is that the actual encryption primitives used are entirely normal and fine - messages are encrypted using AES, and the AES keys are exchanged via NIST P-256 elliptic curve asymmetric keys. The asymmetric keys are each associated with a specific device or browser owned by a user, so when you send a message to someone you encrypt the AES key with all of their asymmetric

keys and then each device or browser can decrypt the message again. As long as the keys are managed appropriately, this is infeasible to break.But how do you know what a user's keys are? I also wrote about this last year - key distribution is a hard problem. In the Twitter DM case, you ask Twitter's server, and if Twitter wants to intercept your messages they replace your key. The documentation for the feature basically admits this - if people with guns showed up there, they could very much compromise the protection in such a way that all future messages you sent were readable. It's also impossible to prove that they're not already doing this without every user verifying that the public keys Twitter hands out to other users correspond to the private keys they hold, something that Twitter provides no mechanism to do.This isn't the only weakness in the implementation. Twitter may not be able read the messages, but every encrypted DM is sent through exactly the same infrastructure as the unencrypted ones, so Twitter can see the time a message was sent, who it was sent to, and roughly how big it was. And because pictures and other attachments in Twitter DMs aren't sent in-line but are instead replaced with links, the implementation would encrypt the links but not the attachments - this is "solved" by simply blocking attachments in encrypted DMs. There's no forward secrecy - if a key is compromised it allows access to not only all new messages created with that key, but also all previous messages. If you log out of Twitter the keys are still stored by the browser, so if you can potentially be extracted and used to decrypt your communications. And there's no group chat support at all, which is more a functional restriction than a conceptual one.To be fair, these are hard problems to solve! Signal solves all of them, but Signal is the product of a large number of highly skilled experts in cryptography, and even so it's taken years to achieve all of this. When Elon announced the launch of encrypted DMs he indicated that new features would be developed quickly - he's since publicly mentioned the feature a grand total of once, in which he mentioned further feature development that just didn't happen. None of the limitations mentioned in the documentation have been addressed in the 22 months since the feature was launched.Why? Well, it turns out that the feature was developed by a total of two engineers, neither of whom is still employed at Twitter. The tech lead for the feature was Christopher Stanley, who was actually a SpaceX employee at the time. Since then he's ended up at DOGE, where he apparently set off alarms when attempting to install Starlink, and who today is apparently being appointed to the board of Fannie Mae, a government-backed mortgage company.Anyway. Use Signal. comments

- Andi Kleen: Quitting an Intel x86 hypervisor (2025/03/18 21:34)
  This is an esoteric topic that might be of interest to people implementing Intel hypervisors. It assumes you know the basics of the Intel virtualization architecture, see Hypervisor from scratch for a tutorial. The actual full VT architecture is described in Volume 3 of the Intel SDMLet’s say we write an x86 hypervisor that starts in the UEFI environment and virtualizes the initialization phase of an OS. But the hypervisor wants to eventually quit itself to not cause extra overhead during OS run time.The way the hypervisor works is that it runs in its own memory and with its own page tables which are switched atomically on every VM exit by the VT-x implementation. This way it is isolated from the main OS.At some vm exit with the hypervisor running in its own context it decides that it is not needed anymore and wants to quit. To disable VT support the VMXOFF instruction can be used. But what we really need is an atomic VMXOFF + switch to the original OS page tables plus a jump, and all that without using any registers which need to be already restored to the original state of the OS. One trick is to use the MOV to CR3 instruction that reloads the page table as a jump. As soon as the page table is reloaded the CPU will fetch the next instruction with the translations from the freshly loaded page table, so we can transfer execution to the guest context. However to do that the MOV CR3 needs to be located just before the page offset of the target instruction. This can be done by copying a trampoline to the right page offset (potentially overlapping into the previous page). The trampoline is located in a special transfer page table mapping that places writable code pages overlapping the target mapping.But there are some complications. The hypervisor also needs to load the segmentation state (like GDT/LDT/IDT) of the guest. In theory

they could just be loaded by mapping these guest pages into the transfer mapping and loading them before the transfer. But what happens if the GDT/LDT/IDT is on the same page as the target address? This is common in real OS' assembler startup code which is implemented in a small assembler file without any page separation between code and data. One option would be to copy them to the transfer page too and load it there, or the hypervisor first copies them to a temporary buffer and loads it from there. In the second option the base addresses of these structures will be incorrect, but in practice you can often rely on them getting reloaded eventually anyways.Another problem is the register state of the target. MOV to CR3 needs a register as the source of the reload, and it needs to be the last instruction of the trampoline. So it is impossible to restore the register it uses. But remember the hypervisor is doing this as the result of a VM exit. If we chose an exit for a condition that already clobbers a register we can use the same register for the reload and the next instruction executed in the original target (and which caused the exit originally) will just overwrite it again. A very convenient instruction for this is CPUID. It is executed multiple times in OS startup and overwrites multiple registers. In fact VMX always intercepts CPUID so it has to handle these exits in any case. So the trick to quit an hypervisor is to wait for the next CPUID exit and then use one of the registers clobbered by CPUID for the final CR3 reload. This will have inconsistent register state for one instruction in the target, but unless the original OS is currently running a debugger it will never notice. In principle any exit as a result of an instruction that clobbers a register can be used for this.There is another potential complication if the target address of the OS conflicts with where the hypervisor is running before entering the transfer mapping. The transfer mapping needs to map the original code so that it can be jumped to. This could be solved with a third auxiliary mapping that is used before jumping to the transfer trampoline. In practice it doesn't seem to be a problem because x86 OS typically run in a 1:1 mapping for startup, and that cannot conflict with the 1:1 mapping used by UEFI programs as our hypervisor. Happy hypervisor hacking!

- Lucas De Marchi: Lazy libkmod compression library loading (2025/03/03 18:00)
  One new feature in kmod 34 is related to lazily loading the decompression libraries. In other words, dlopen them when/if they are needed. Although it's desired for a tool like modinfo to be able to inspect a .ko.xz, .ko.zst or .ko.gz module, other daemons linking to libkmod would benefit from never loading them. This is the case for systemd-udevd that will load the kernel modules during boot when they are requested by the kernel. Testing on Archlinux, it goes from this: $ cat /proc/$(pidof systemd-udevd)/maps | grep -e libz -e liblzma 7f27a9ae8000-7f27a9aeb000 r--p 00000000 00:19 15656 /usr/lib/liblzma.so.5.6.4 7f27a9aeb000-7f27a9b0e000 r-xp 00003000 00:19 15656 /usr/lib/liblzma.so.5.6.4 7f27a9b0e000-7f27a9b19000 r--p 00026000 00:19 15656 /usr/lib/liblzma.so.5.6.4 7f27a9b19000-7f27a9b1a000 r--p 00031000 00:19 15656 /usr/lib/liblzma.so.5.6.4 7f27a9b1a000-7f27a9b1b000 rw-p 00032000 00:19 15656 /usr/lib/liblzma.so.5.6.4 7f27a9b1b000-7f27a9b27000 r--p 00000000 00:19 15985 /usr/lib/libzstd.so.1.5.7 7f27a9b27000-7f27a9bed000 r-xp 0000c000 00:19 15985 /usr/lib/libzstd.so.1.5.7 7f27a9bed000-7f27a9bfe000 r--p 000d2000 00:19 15985 /usr/lib/libzstd.so.1.5.7 7f27a9bfe000-7f27a9bff000 r--p 000e3000 00:19 15985 /usr/lib/libzstd.so.1.5.7 7f27a9bff000-7f27a9c00000 rw-p 000e4000 00:19 15985 /usr/lib/libzstd.so.1.5.7 7f27aa892000-7f27aa895000 r--p 00000000 00:19 7852 /usr/lib/libz.so.1.3.1 7f27aa895000-7f27aa8a3000 r-xp 00003000 00:19 7852 /usr/lib/libz.so.1.3.1 7f27aa8a3000-7f27aa8a9000 r--p 00011000 00:19 7852 /usr/lib/libz.so.1.3.1 7f27aa8a9000-7f27aa8aa000 r--p 00017000 00:19 7852 /usr/lib/libz.so.1.3.1 7f27aa8aa000-7f27aa8ab000 rw-p 00018000 00:19 7852 /usr/lib/libz.so.1.3.1 $ to this: $ cat /proc/$(pidof systemd-udevd)/maps | grep -e libz -e liblzma $ … even if all modules in Archlinux are zstd-compressed. systemd itself started doing this a few releases ago and published https://systemd.io/ELF_PACKAGE_METADATA/. That spec is also used for this new support in kmod to annotate what libraries are possibly dlopen'ed. However although it prevented libkmod from being loaded in other binaries, it didn't prevent all these decompression

libraries from being mapped in systemd-udevd since it uses libkmod. One might wonder why not even libzstd.so is mapped. That’s because when loading the modules and the kernel supports decompressing that format, libkmod just skips any decompression: it opens the file and passes the descriptor to the Linux kernel via finit_module. /sys/module/compression shows what algorithm the Linux kernel can use for module decompression. In kmod 34 all that is needed is to setup the build with -Ddlopen=all. More fine-grained options are also supported, allowing to specify individual libraries to dlopen. It may go away in a future release if distros just choose an all-or-nothing support.

- Pete Zaitcev: Looking for a BSSID (2025/01/11 01:42)
  I'm looking for a name for a new WiFi area.The current one is called "Tokyo-Jupiter". It turns out hard to top, it meets all the requirements. It's a geographic area. It's weeb, but from old enough times: not Naruto Shippuuden, Attack On Titan, or Kimetsu no Yaiba. Classy and unique enough."Konoha" is too new, too washed-up, and too short."Kodena" and "Yokosuka" add a patriotic American tint nicely, but also too short."Minas-Tirith" is a place and outstanding in its reference, but not weeb."Big-Sight" is an opposite of the above: too much. I'm a weeb, not otaku.Any ideas are appreciated.UPDATE 2025-01-11: The provisional candidate is "Nishi-Teppelin". Don't google it, it's not canon. I remain open to better ideas.UPDATE 2025-02-20: Ended with "Ostrov-Krym" after all.
- Matthew Garrett: The GPU, not the TPM, is the root of hardware DRM (2025/01/02 01:14)
  As part of their "Defective by Design" anti-DRM campaign, the FSF recently made the following claim:Today, most of the major streaming media platforms utilize the TPM to decrypt media streams, forcefully placing the decryption out of the user's control (from here).This is part of an overall argument that Microsoft's insistence that only hardware with a TPM can run Windows 11 is with the goal of aiding streaming companies in their attempt to ensure media can only be played in tightly constrained environments.I'm going to be honest here and say that I don't know what Microsoft's actual motivation for requiring a TPM in Windows 11 is. I've been talking about TPM stuff for a long time. My job involves writing a lot of TPM code. I think having a TPM enables a number of worthwhile security features. Given the choice, I'd certainly pick a computer with a TPM. But in terms of whether it's of sufficient value to lock out Windows 11 on hardware with no TPM that would otherwise be able to run it? I'm not sure that's a worthwhile tradeoff.What I can say is that the FSF's claim is just 100% wrong, and since this seems to be the sole basis of their overall claim about Microsoft's strategy here, the argument is pretty significantly undermined. I'm not aware of any streaming media platforms making use of TPMs in any way whatsoever. There is hardware DRM that the media companies use to restrict users, but it's not in the TPM - it's in the GPU.Let's back up for a moment. There's multiple different DRM implementations, but the big three are Widevine (owned by Google, used on Android, Chromebooks, and some other embedded devices), Fairplay (Apple implementation, used for Mac and iOS), and Playready (Microsoft's implementation, used in Windows and some other hardware streaming devices and TVs). These generally implement several levels of functionality, depending on the capabilities of the device they're running on - this will range from all the DRM functionality being implemented in software up to the hardware path that will be discussed shortly. Streaming providers can choose what level of functionality and quality to provide based on the level implemented on the client device, and it's common for 4K and HDR content to be tied to hardware DRM. In any scenario, they stream encrypted content to the client and the DRM stack decrypts it before the compressed data can be decoded and played.The "problem" with software DRM implementations is that the decrypted material is going to exist somewhere the OS can get at it at some point, making it possible for users to simply grab the decrypted stream, somewhat defeating the entire point. Vendors try to make this difficult by obfuscating their code as much as possible (and in some cases putting some of it in-kernel), but pretty much all software DRM is at least somewhat broken and copies of any new streaming media end up being available via Bittorrent pretty quickly after release. This is why higher quality media tends

to be restricted to clients that implement hardware-based DRM.The implementation of hardware-based DRM varies. On devices in the ARM world this is usually handled by performing the cryptography in a Trusted Execution Environment, or TEE. A TEE is an area where code can be executed without the OS having any insight into it at all, with ARM's TrustZone being an example of this. By putting the DRM code in TrustZone, the cryptography can be performed in RAM that the OS has no access to, making the scraping described earlier impossible. x86 has no well-specified TEE (Intel's SGX is an example, but is no longer implemented in consumer parts), so instead this tends to be handed off to the GPU. The exact details of this implementation are somewhat opaque - of the previously mentioned DRM implementations, only Playready does hardware DRM on x86, and I haven't found any public documentation of what drivers need to expose for this to work.In any case, as part of the DRM handshake between the client and the streaming platform, encryption keys are negotiated with the key material being stored in the GPU or the TEE, inaccessible from the OS. Once decrypted, the material is decoded (again either on the GPU or in the TEE - even in implementations that use the TEE for the cryptography, the actual media decoding may happen on the GPU) and displayed. One key point is that the decoded video material is still stored in RAM that the OS has no access to, and the GPU composites it onto the outbound video stream (which is why if you take a screenshot of a browser playing a stream using hardware-based DRM you'll just see a black window - as far as the OS can see, there is only a black window there).Now, TPMs are sometimes referred to as a TEE, and in a way they are. However, they're fixed function - you can't run arbitrary code on the TPM, you only have whatever functionality it provides. But TPMs do have the ability to decrypt data using keys that are tied to the TPM, so isn't this sufficient? Well, no. First, the TPM can't communicate with the GPU. The OS could push encrypted material to it, and it would get plaintext material back. But the entire point of this exercise was to avoid the decrypted version of the stream from ever being visible to the OS, so this would be pointless. And rather more fundamentally, TPMs are slow. I don't think there's a TPM on the market that could decrypt a 1080p stream in realtime, let alone a 4K one.The FSF's focus on TPMs here is not only technically wrong, it's indicative of a failure to understand what's actually happening in the industry. While the FSF has been focusing on TPMs, GPU vendors have quietly deployed all of this technology without the FSF complaining at all. Microsoft has enthusiastically participated in making hardware DRM on Windows possible, and user freedoms have suffered as a result, but Playready hardware-based DRM works just fine on hardware that doesn't have a TPM and will continue to do so. comments

- Matthew Garrett: When should we require that firmware be free? (2024/12/12 15:57)
  The distinction between hardware and software has historically been relatively easy to understand - hardware is the physical object that software runs on. This is made more complicated by the existence of programmable logic like FPGAs, but by and large things tend to fall into fairly neat categories if we're drawing that distinction.Conversations usually become more complicated when we introduce firmware, but should they? According to Wikipedia, Firmware is software that provides low-level control of computing device hardware, and basically anything that's generally described as firmware certainly fits into the "software" side of the above hardware/software binary. From a software freedom perspective, this seems like something where the obvious answer to "Should this be free" is "yes", but it's worth thinking about why the answer is yes - the goal of free software isn't freedom for freedom's sake, but because the freedoms embodied in the Free Software Definition (and by proxy the DFSG) are grounded in real world practicalities.How do these line up for firmware? Firmware can fit into two main classes - it can be something that's responsible for initialisation of the hardware (such as, historically, BIOS, which is involved in initialisation and boot and then largely irrelevant for runtime[1]) or it can be something that makes the hardware work at runtime (wifi card firmware being an obvious example). The role of free software in the latter case feels fairly intuitive, since the interface and functionality the hardware offers to the operating system

is frequently largely defined by the firmware running on it. Your wifi chipset is, these days, largely a software defined radio, and what you can do with it is determined by what the firmware it's running allows you to do. Sometimes those restrictions may be required by law, but other times they're simply because the people writing the firmware aren't interested in supporting a feature - they may see no reason to allow raw radio packets to be provided to the OS, for instance. We also shouldn't ignore the fact that sufficiently complicated firmware exposed to untrusted input (as is the case in most wifi scenarios) may contain exploitable vulnerabilities allowing attackers to gain arbitrary code execution on the wifi chipset - and potentially use that as a way to gain control of the host OS (see this writeup for an example). Vendors being in a unique position to update that firmware means users may never receive security updates, leaving them with a choice between discarding hardware that otherwise works perfectly or leaving themselves vulnerable to known security issues.But even the cases where firmware does nothing other than initialise the hardware cause problems. A lot of hardware has functionality controlled by registers that can be locked during the boot process. Vendor firmware may choose to disable (or, rather, never to enable) functionality that may be beneficial to a user, and then lock out the ability to reconfigure the hardware later. Without any ability to modify that firmware, the user lacks the freedom to choose what functionality their hardware makes available to them. Again, the ability to inspect this firmware and modify it has a distinct benefit to the user.So, from a practical perspective, I think there's a strong argument that users would benefit from most (if not all) firmware being free software, and I don't think that's an especially controversial argument. So I think this is less of a philosophical discussion, and more of a strategic one - is spending time focused on ensuring firmware is free worthwhile, and if so what's an appropriate way of achieving this?I think there's two consistent ways to view this. One is to view free firmware as desirable but not necessary. This approach basically argues that code that's running on hardware that isn't the main CPU would benefit from being free, in the same way that code running on a remote network service would benefit from being free, but that this is much less important than ensuring that all the code running in the context of the OS on the primary CPU is free. The maximalist position is not to compromise at all - all software on a system, whether it's running at boot or during runtime, and whether it's running on the primary CPU or any other component on the board, should be free.Personally, I lean towards the former and think there's a reasonably coherent argument here. I think users would benefit from the ability to modify the code running on hardware that their OS talks to, in the same way that I think users would benefit from the ability to modify the code running on hardware the other side of a network link that their browser talks to. I also think that there's enough that remains to be done in terms of what's running on the host CPU that it's not worth having that fight yet. But I think the latter is absolutely intellectually consistent, and while I don't agree with it from a pragmatic perspective I think things would undeniably be better if we lived in that world.This feels like a thing you'd expect the Free Software Foundation to have opinions on, and it does! There are two primarily relevant things - the Respects your Freedoms campaign focused on ensuring that certified hardware meets certain requirements (including around firmware), and the Free System Distribution Guidelines, which define a baseline for an OS to be considered free by the FSF (including requirements around firmware).RYF requires that all software on a piece of hardware be free other than under one specific set of circumstances. If software runs on (a) a secondary processor and (b) within which software installation is not intended after the user obtains the product, then the software does not need to be free. (b) effectively means that the firmware has to be in ROM, since any runtime interface that allows the firmware to be loaded or updated is intended to allow software installation after the user obtains the product.The Free System Distribution Guidelines require that all non-free firmware be removed from the OS before it can be considered free. The recommended mechanism to achieve this is via linux-libre, a project that produces tooling to remove anything that looks plausibly like a non-free firmware blob from the Linux source code, along with any incitement to the user to load firmware - including even removing suggestions to update CPU microcode in order to mitigate

CPU vulnerabilities.For hardware that requires non-free firmware to be loaded at runtime in order to work, linux-libre doesn't do anything to work around this - the hardware will simply not work. In this respect, linux-libre reduces the amount of non-free firmware running on a system in the same way that removing the hardware would. This presumably encourages users to purchase RYF compliant hardware.But does that actually improve things? RYF doesn't require that a piece of hardware have no non-free firmware, it simply requires that any non-free firmware be hidden from the user. CPU microcode is an instructive example here. At the time of writing, every laptop listed here has an Intel CPU. Every Intel CPU has microcode in ROM, typically an early revision that is known to have many bugs. The expectation is that this microcode is updated in the field by either the firmware or the OS at boot time - the updated version is loaded into RAM on the CPU, and vanishes if power is cut. The combination of RYF and linux-libre doesn't reduce the amount of non-free code running inside the CPU, it just means that the user (a) is more likely to hit since-fixed bugs (including security ones!), and (b) has less guidance on how to avoid them.As long as RYF permits hardware that makes use of non-free firmware I think it hurts more than it helps. In many cases users aren't guided away from non-free firmware - instead it's hidden away from them, leaving them less aware that their freedom is constrained. Linux-libre goes further, refusing to even inform the user that the non-free firmware that their hardware depends on can be upgraded to improve their security.Out of sight shouldn't mean out of mind. If non-free firmware is a threat to user freedom then allowing it to exist in ROM doesn't do anything to solve that problem. And if it isn't a threat to user freedom, then what's the point of requiring linux-libre for a Linux distribution to be considered free by the FSF? We seem to have ended up in the worst case scenario, where nothing is being done to actually replace any of the non-free firmware running on people's systems and where users may even end up with a reduced awareness that the non-free firmware even exists.[1] Yes yes SMM comments

- Matthew Garrett: Android privacy improvements break key attestation (2024/12/12 12:16)
  Sometimes you want to restrict access to something to a specific set of devices - for instance, you might want your corporate VPN to only be reachable from devices owned by your company. You can't really trust a device that self attests to its identity, for instance by reporting its MAC address or serial number, for a couple of reasons: These aren't fixed - MAC addresses are trivially reprogrammable, and serial numbers are typically stored in reprogrammable flash at their most protected A malicious device could simply lie about themIf we want a high degree of confidence that the device we're talking to really is the device it claims to be, we need something that's much harder to spoof. For devices with a TPM this is the TPM itself. Every TPM has an Endorsement Key (EK) that's associated with a certificate that chains back to the TPM manufacturer. By verifying that certificate path and having the TPM prove that it's in posession of the private half of the EK, we know that we're communicating with a genuine TPM[1].Android has a broadly equivalent thing called ID Attestation. Android devices can generate a signed attestation that they have certain characteristics and identifiers, and this can be chained back to the manufacturer. Obviously providing signed proof of the device identifier is kind of problematic from a privacy perspective, so the short version[2] is that only apps installed using a corporate account rather than a normal user account are able to do this.But that's still not ideal - the device identifiers involved included the IMEI and serial number of the device, and those could potentially be used to correlate devices across privacy boundaries since they're static[3] identifiers that are the same both inside a corporate work profile and in the normal user profile, and also remains static if you move between different employers and use the same phone[4]. So, since Android 12, ID Attestation includes an "Enterprise Specific ID" or ESID. The ESID is based on a hash of device-specific data plus the enterprise that the corporate work profile is associated with. If a device is enrolled with the same enterprise then this ID will remain static, if it's enrolled with a different enterprise it'll change, and it just doesn't exist outside the work profile at all. The other device identifiers are no longer exposed.But device ID verification isn't enough to solve the underlying problem here. When we receive a device ID attestation we know

that someone at the far end has posession of a device with that ID, but we don't know that that device is where the packets are originating. If our VPN simply has an API that asks for an attestation from a trusted device before routing packets, we could pass that on to said trusted device and then simply forward the attestation to the VPN server[5]. We need some way to prove that the the device trying to authenticate is actually that device.The answer to this is key provenance attestation. If we can prove that an encryption key was generated on a trusted device, and that the private half of that key is stored in hardware and can't be exported, then using that key to establish a connection proves that we're actually communicating with a trusted device. TPMs are able to do this using the attestation keys generated in the Credential Activation process, giving us proof that a specific keypair was generated on a TPM that we've previously established is trusted.Android again has an equivalent called Key Attestation. This doesn't quite work the same way as the TPM process - rather than being tied back to the same unique cryptographic identity, Android key attestation chains back through a separate cryptographic certificate chain but contains a statement about the device identity - including the IMEI and serial number. By comparing those to the values in the device ID attestation we know that the key is associated with a trusted device and we can now establish trust in that key."But Matthew", those of you who've been paying close attention may be saying, "Didn't Android 12 remove the IMEI and serial number from the device ID attestation?" And, well, congratulations, you were apparently paying more attention than Google. The key attestation no longer contains enough information to tie back to the device ID attestation, making it impossible to prove that a hardware-backed key is associated with a specific device ID attestation and its enterprise enrollment.I don't think this was any sort of deliberate breakage, and it's probably more an example of shipping the org chart - my understanding is that device ID attestation and key attestation are implemented by different parts of the Android organisation and the impact of the ESID change (something that appears to be a legitimate improvement in privacy!) on key attestation was probably just not realised. But it's still a pain.[1] Those of you paying attention may realise that what we're doing here is proving the identity of the TPM, not the identity of device it's associated with. Typically the TPM identity won't vary over the lifetime of the device, so having a one-time binding of those two identities (such as when a device is initially being provisioned) is sufficient. There's actually a spec for distributing Platform Certificates that allows device manufacturers to bind these together during manufacturing, but I last worked on those a few years back and don't know what the current state of the art there is[2] Android has a bewildering array of different profile mechanisms, some of which are apparently deprecated, and I can never remember how any of this works, so you're not getting the long version[3] Nominally, anyway. Cough.[4] I wholeheartedly encourage people not to put work accounts on their personal phones, but I am a filthy hypocrite here[5] Obviously if we have the ability to ask for attestation from a trusted device, we have access to a trusted device. Why not simply use the trusted device? The answer there may be that we've compromised one and want to do as little as possible on it in order to reduce the probability of triggering any sort of endpoint detection agent, or it may be because we want to run on a device with different security properties than those enforced on the trusted device. comments

- Pete Zaitcev: virtio_pci: do not wait forvever at a reset (2024/10/30 17:58)
  We all know how it's possible for a guest VM to access various host functions by accessing a PCI device, right? When KVM traps an access to this fake PCI, QEMU emulates the device, which allows packets sent, console updated, or whatever. This is called "virtio". NVIDIA took it a step further: they have a real PCI device that emuilates QEMU. No joke. And, they have a firmware bug! The following patch works around it: diff --git a/drivers/virtio/virtio_pci_modern.c b/drivers/virtio/virtio_pci_modern.c index 9193c30d640a..6bbb34f9b088 100644 --- a/drivers/virtio/virtio_pci_modern.c +++ b/drivers/virtio/virtio_pci_modern.c @@ -438,6 +438,7 @@ static void vp_reset(struct virtio_device *vdev) { struct virtio_pci_device *vp_dev = to_vp_device(vdev); struct virtio_pci_modern_device *mdev = &vp_dev->mdev; + int i; /* 0 status

means a reset. */ vp_modern_set_status(mdev, 0); @@ -446,8 +447,16 @@ static void vp_reset(struct virtio_device *vdev) * This will flush out the status write, and flush in device writes, * including MSI-X interrupts, if any. */ - while (vp_modern_get_status(mdev)) + i = 0; + while (vp_modern_get_status(mdev)) { + if (++i >= 10000) { + printk(KERN_INFO + "virtio reset ignoring status 0x%02x\n", + vp_modern_get_status(mdev)); + break; + } msleep(1); + } vp_modern_avq_cleanup(vdev); I'm not dumping on NVIDIA here at all, I think it's awesome for this devious hardware to exist. And bugs are just a way of life.

- Pete Zaitcev: LinkedIn Asked You To Train Their AI (2024/10/30 17:17)
  They pushed the "You're one of a few experts invited to answer" notifications for a long time - maybe a year, I don't remember. When I had enough and started to capture them with the intent of mockery, they stopped. So sad. Here's what I got:"You're facing pushback from vendors on cloud integration. How can you convince them to collaborate?""You're focused on cutting costs in cloud computing. How do you ensure security protocols aren't compromised?""You're overseeing a code review process. How do you ensure feedback boosts developer morale?"What a dystopia. LinkedIn is owned by Microsoft, so I'm not suprised someone in a giant corporation thought this sort of nonsense was a good idea. But still, the future is stupid, and all that.P.S. The notification inserts were non-persistent — inserted on the fly. That was just fraud w.r.t. the idea of notification ticker.P.P.S. Does anyone else think that this sort of thing would cause self-selection? They made their AI trained by the most vain and also least bright members of their user population. I'm not an expert in any of these fields.UPDATE 2024-10-31: Spoke too soon! They hit me with the notificantion insert: "Here's how you can craft a personalized learning plan for advancing in Cloud Computing." That is not even a formed question. Getting lazy, are we?UPDATE 2024-11-02: "You're facing budget disputes over cloud solutions. How can you align IT and non-technical teams effectively?" They are not stopping.Meanwhile, how about another perspective: I saw an update that Hubbert Smith contributed an answer to: "You're facing a ransomware attack crisis. How do you convey the severity to a non-technical executive?" Instead of answering what LinkedIn AI asked, he answered a question of how to deal with ransomware ("Ransomware is fixable with snapshots of sensitive data."). Unless he is an AI himself, he may be thinking that he's dealing with a LinkedIn equivalent of Quora.I'm trying to ask him what happened.
- Brendan Gregg: AI Flame Graphs (2024/10/28 13:00)
  Imagine halving the resource costs of AI and what that could mean for the planet and the industry -- based on extreme estimates such savings could reduce the total US power usage by over 10% by 20301. At Intel we've been creating a new analyzer tool to help reduce AI costs called AI Flame Graphs: a visualization that shows an AI accelerator or GPU hardware profile along with the full software stack, based on my CPU flame graphs. Our first version is available to customers in the Intel Tiber AI Cloud as a preview for the Intel Data Center GPU Max Series (previously called Ponte Vecchio). Here is an example: Simple example: SYCL matrix multiply microbenchmark (Click for interactive SVG.) The green frames are the actual instructions running on the AI or GPU accelerator, aqua shows the source code for these functions, and red (C), yellow (C++), and orange (kernel) show the CPU code paths that initiated these AI/GPU programs. The gray "-" frames just help highlight the boundary between CPU and AI/GPU code. The x-axis is proportional to cost, so you look for the widest things and find ways to reduce them. Layers This flame graph shows a simple program for SYCL (a high-level C++ language for accelerators) that tests three implementations of matrix multiply, running them with the same input workload. The flame graph is dominated by the slowest implementation, multiply_basic(), which doesn't use any optimizations and consumes at 72% of stall samples and is shown as the widest tower. On the right are two thin towers for multiply_local_access() at 21% which replaces the accessor with a local variable, and multiply_local_access_and_tiling() at 6% which also adds matrix tiling. The towers are getting smaller as optimizations are added. This flame graph profiler is a prototype based on Intel EU stall profiling

for hardware profiling and eBPF for software instrumentation. It's designed to be easy and low-overhead, just like a CPU profiler. You should be able to generate a flame graph of an existing AI workload whenever you want, without having to restart anything or launch additional code via an interposer. Instruction-offset Profiling This is not the first project to build an AI profiler or even something called an AI Flame Graph, however, others I've seen focus on tracing CPU stacks and timing accelerator execution, but don't profile the instruction offsets running on the accelerator; or do profile them but via expensive binary instrumentation. I wanted to build AI flame graphs that work like CPU flame graphs: Easy to use, negligible cost, production safe, and shows everything. A daily tool for developers, with most of the visualization in the language of the developer: source code functions. This has been an internal AI project at Intel for the past year. Intel was already investing in this space, building the EU stall profiler capability for the Intel Data Center GPU Max Series that provides an approximation of HW instruction sampling. I was lucky to have Dr. Matthew (Ben) Olson, an Intel AI engineer who has also worked on eBPF performance tooling (processwatch) as well as memory management research, join my team and do most of the development work. His background has helped us power through difficulties that seemed insurmountable. We've also recently been joined by Dr. Brandon Kammerdiener (coincidentally another graduate of the University of Tennessee, like Ben), who also has eBPF and memory internals experience, and has been helping us take on harder and harder workloads. And Gabriel Muñoz just joined today to help with releases. Now that our small team has shown that this is possible, we'll be joined by other teams at Intel to develop this further. We could have built a harder-to-use and higher-overhead version months ago using Intel GTPin but for widespread adoption it needs minimal overhead and ease of use so that developers don't hesitate to use this daily and to add it to deployment pipelines. What's a Flame Graph? A flame graph is a visualization I invented in 2011 for showing sampled code stack traces. It has become the standard for CPU profiling and analysis, helping developers quickly find performance improvements and eliminate regressions. A CPU flame graph shows the "big picture" of running software, with x-axis proportional to CPU cost. The example picture on the right summarizes how easy it can be to go from compute costs to responsible code paths. Prior to flame graphs, it could take hours to understand a complex profile by reading through hundreds of pages of output. Now it takes seconds: all you have to do is look for the widest rectangles. Flame graphs have had worldwide adoption. They have been the basis for five startups so far, have been adopted in over thirty performance analysis products, and have had over eighty implementations. My first implementation of flame graphs took a few hours on a Wednesday night after work. The real effort has been in the decade since, where I worked with different profilers, runtimes, libraries, kernels, compilers, and hypervisors to get flame graphs working properly in different environments, including fixing stack walking and symbolization. Earlier this year I posted about the final missing piece: Helping distros enable frame pointers so that profiling works across standard system libraries. Similar work is necessary for AI workloads: fixing stacks and symbols and getting profiling to work for different hardware, kernel drivers, user-mode drivers, frameworks, runtimes, languages, and models. A lot more work, too, as AI analysis has less maturity than CPU analysis. Searching Samples If you are new to flame graphs, it's worth mentioning the built-in search capability. In the earlier example, most of the stall samples are caused by sbid: software scoreboard dependency. As that may be a unique search term, you can run search (Ctrl-F, or click "Search") on "sbid" and it will highlight it in magenta: Search also shows the total number of stack samples that contained sbid in the bottom right: 78.4%. You can search for any term in the flame graph: accelerator instructions, source paths, function names, etc., to quickly calculate the percentage of stacks where it is present (excluding vertical overlap) helping you prioritise performance work. Note that the samples are EU stall-based, which means theoretical performance wins can take the percentages down to zero. This is different to timer-based samples as are typically used in CPU profiling. Stalls mean you better focus on the pain, the parts of the code that aren't making forward progress, but you aren't seeing resource usage by unstalled instructions. I'd like to

supuport timer-based samples in the future as well, so we can have both views. Who will use this? At a recent golang conference, I asked the audience of 200+ to raise their hands if they were using CPU flame graphs. Almost every hand went up. I know of companies where flame graphs are a daily tool that developers use to understand and tune their code, reducing compute costs. This will become a daily tool for AI developers. My employer will use this as well for evaluation analysis, to find areas to tune to beat competitors, as well as to better understand workload performance to aid design. Why is AI profiling hard? Consider CPU instruction profiling: This is easy when the program and symbol table are both in the file system and in a standardized file format (such as ELF) as is the case with native compiled code (C). CPU profiling gets hard for JIT-complied code, like Java, as instructions and symbols are dynamically generated and placed in main memory (the process heap) without following a universal standard. For such JITted code we use runtime-specific methods and agents to retrieve snapshots of the heap information, which is different for each runtime. AI workloads also have different runtimes (and frameworks, languages, user-mode drivers, compilers, etc.) any of which can require special tinkering to get their CPU stacks and symbols to work. These CPU stacks are shown as the red, orange, and yellow frames in the AI Flame Graph. Some AI workloads are easy to get these frames working, some (like PyTorch) are a lot more work. But the real challenge is instruction profiling of actual GPU and AI accelerator programs -- shown as the aqua and green frames -- and correctly associating them with the CPU stacks beneath them. Not only may these GPU and AI programs not exist in the file system, but they may not even exist in main memory! Even for running programs. Once execution begins, they may be deallocated from main memory and only exist in special accelerator memory, beyond the direct reach of OS profilers and debuggers. Or within reach, but only through a prohibitively high-overhead HW-specific debugger interface. There's also no /proc representation for these programs either (I've been proposing building an equivalent) so there's no direct way to even tell what is running and what isn't, and all the other /proc details. Forget instruction profiling, even ps(1) and all the other process tools do not work. It's been a mind-bending experience, revealing what gets taken for granted because it has existed in CPU land for decades: A process table. Process tools. Standard file formats. Programs that exist in the file system. Programs running from main memory. Debuggers. Profiliers. Core dumping. Disassembling. Single stepping. Static and dynamic instrumentation. Etc. For GPUs and AI, this is all far less mature. It can make the work exciting at times, when you think something is impossible and then find or devise a way. Fortunately we have a head start as some things do exist. Depending on the runtime and kernel driver, there are debug interfaces where you can list running accelerator programs and other statistics, as used by tools like intel_gpu_top(1). You can kill -9 a GPU workload using intel_gpu_abrt(1). Some interfaces can even generate basic ELF files for the running accelerator programs that you can try to load in a debugger like gdb(1). And there is support for GPU/AI program disassembly, if you can get your hands on the binary. It feels to me like GPU/AI debugging, OS style, is about two years old. Better than zero, but still early on, and lots more ahead of us. A decade, at least. What do AI developers think of this? We've shown AI Flame Graphs to other AI developers at Intel and a common reaction is to be a bit puzzled, wondering what to do with it. AI developers think about their bit of code, but with AI Flame Graphs they can now see the entire stack for the first time, including the HW, and many layers they don't usually think about or don't know about. It basically looks like a pile of gibberish with their code only a small part of the flame graph. CPU Flame Graph Implementations This reaction is similar to people's first experiences with CPU flame graphs, which show parts of the system that developers and engineers typically don't work on, such as runtime internals, system libraries, and kernel internals. Flame graphs are great at highlighting the dozen or so functions that matter the most, so it becomes a problem of learning what those functions do across a few different code bases, which are typically open source. Understanding a dozen such functions can take a few hours or even a few days -- but if this leads to a 10% or 2x cost win, it is time well spent. And the next time the user looks at a flame graph, they start saying "I've seen that function before"

and so on. You can get to the point where understanding the bulk of a CPU flame graph takes less than a minute: look for the widest tower, click to zoom, read the frames, done. I'm encouraged by the success of CPU flame graphs, with over 80 implementations and countless real world case studies. Sometimes I'm browsing a performance issue I care about on github and hit page down and there's a CPU flame graph. They are everywhere. I expect AI developers will also be able to understand AI Flame Graphs in less than a minute, but to start with people will be spending a day or more browsing code bases they didn't know were involved. Publishing case studies of found wins will also help people learn how to interpret them, and also help explain the value. What about PyTorch? Another common reaction we've had is that AI developers are using PyTorch, and initially we didn't support it as it meant walking Python stacks, which isn't trivial. But prior work has been done there (to support CPU profiling) and after a lot of tinkering we now have the first PyTorch AI Flame Graph: PyTorch frames in pink (Click for interactive SVG.) The PyTorch functions are at the bottom and are colored pink. This example runs oneDNN kernels that are JIT-generated, and don't have a source path so that layer just reads "jit". Getting all other the layers included was a real pain to get going, but an important milestone. We think if we can do PyTorch we can do anything. In this flame graph, we show PyTorch running the Llama 2 7B model using the Intel Extensions for PyTorch (IPEX). This flame graph shows the origin of the GPU kernel execution all the way back to the Python source code shown in pink. Most samples are from a stack leading up to a gemm_kernel (matrix multiply) shown in aqua, which like the previous example has many stalls due to software scoreboarding. There are two instructions (0xa30 and 0xa90) that combined are 27% of the entire profile. I expect someone will ask: Can't we just click on instructions and have it bring up a dissassembly view with full source? Yes, that should be possible, but I can't answer how we're going to provide this yet. Another expected question I can't yet answer: Since there are now multiple products providing AI auto-tuning of CPU workloads using CPU flame graphs (including Intel Granulate) can't we have AI auto-tuning of AI workloads using AI Flame Graphs? First Release: Sometimes hard and with moderate overhead Getting AI Flame Graphs to work with some workloads is easy, but others are currently hard and cost moderate overhead. It's similar to CPU profiling, where some workloads and languages are easy to profile, whereas others need various things fixed. Some AI workloads use many software dependencies that need various tweaks and recompilation (e.g., enabling frame pointers so that stack walking works) making setup time consuming. PyTorch is especially difficult and can take over a week of OS work to be ready for AI Flame Graphs. We will work on getting these tweaks changed upstream in their respective repositories, something involving teams inside and outside of Intel, and is a process I'd expect to take at least a year. During that time AI workloads will gradually become easier to flame graph, and with lower-overhead as well. I'm reminded of eBPF in the early days: You had to patch and recompile the kernel and LLVM and Clang, which could take multiple days if you hit errors. Since then all the eBPF dependency patches have been merged, and default settings changed, so that eBPF "just works." We'll get there with AI Flame Graphs too, but right now it's still those early days. The changes necessary for AI Flame Graphs are really about improving debugging in general, and are a requirement for Fast by Friday: A vision where we can root-cause analyze anything in five days or less. Availability AI Flame Graphs will first become available on the Intel Tiber AI Cloud as a preview feature for the Intel Data Center GPU Max Series. If you are currently deployed there you can ask through the Intel service channel for early access. As for if or when it will support other hardware types, be in other Intel products, be officially launched, be open source, etc., these involve various other teams at Intel and they need to make their own announcements before I can discuss them here. Conclusions Finding performance improvements for AI data centers of just fractions of a percent can add up to planetary savings in electricity, water, and money. If AI flame graphs have the success that CPU flame graphs have had, I'd expect finding improvements of over 10% will be common, and 50% and higher will eventually be found*. But it won't be easy in these early days as there are still many software components to tweak and recompile, and software layers to learn about that are

revealed in the AI flame graph. In the years ahead I imagine others will build their own AI flame graphs that look the same as this one, and there may even be startups selling them, but if they use more difficult-to-use and higher-overhead technologies I fear they could turn companies off the idea of AI flame graphs altogether and prevent them from finding sorely needed wins. This is too important to do badly. AI flame graphs should be easy to use, cost negligible overhead, be production safe, and show everything. Intel has proven it's possible. Disclaimer * This is a personal blog post that makes personal predictions but not guarantees of possible performance improvements. Feel free to take any claim with a grain of salt, and feel free to wait for an official publication and public launch by Intel on this technology. 1 Based on halving the Arm CEO Rene Haas' estimate of 20-25% quoted in Taking a closer look at AI's supposed energy apocalypse by Kyle Orland of ArsTechnica. Updates (Jan 2025): I wasn't going to talk about other specific profilers, but since FAQ #1 is "what about NVIDIA?": They do have flame graphs in Nsight Graphics for GPU workloads, so I'd guess they aren't far from supporting AI workloads as well. Their version looks shallow as it's only the GPU code, so it is missing the CPU context necessary to understand the full picture, and seems based on interposing (launching the app from Nsights) which is the old method. The new method we've created allows for easy, everything, anytime profiling, like you expect from CPU profilers. Theirs does have click-to-source integration, which is quite handy, and I do like the default orientation and colors, thank you! Thanks Thanks to everyone at Intel who have helped us make this happen. Markus Flierl has driven this project and made it a top priority, and Greg Lavender has expressed his support. Special thanks to Michael Cole, Matthew Roper, Luis Strano, Rodrigo Vivi, Joonas Lahtinen, Stanley Gambarin, Timothy Bauer, Brandon Yates, Maria Kraynyuk, Denis Samoylov, Krzysztof Raszkowski, Sanchit Jain, Po-Yu Chen, Felix Degrood, Piotr Rozenfeld, Andi Kleen, and all of the other coworkers that helped clear things up for us, and thanks in advance for everyone else who will be helping us in the months ahead. My final thanks is to the companies and developers who do the actual hands-on work with flame graphs, collecting them, examining them, finding performance wins, and applying them.You are helping save the planet.

- Harald Welte: On Linux MAINTAINERS file removal of Russian developers (2024/10/23 22:00)
  I sincerely regret to see Linux kernel patches like this one removing Russian developers from the MAINTAINERS file. To me, it is a sign or maybe even a symbol of how far the Linux kernel developer community I remember from ~ 20 years ago has changed, and how much it has alienated itself from what I remember back in the day. In my opinion this commit is wrong at so many different levels: it is intransparent. Initially it gave no explanation whatsoever (other than some compliance hand-waving). There was some follow-up paraphrasing one paragraph of presumed legal advice that was given presumably by Linux Foundation to Linus. That's not a thorough legal analysis at all. It doesn't even say to whom it was given, and who (the individual developers? Linux Foundation? Distributors?) is presumed to be subject to the unspecified regulations in which specific jurisdiction it discriminates developers based on their presumed [Russian] nationality based on their name, e-mail address domain name or employer. A later post in the thread has clarified that it's about an U.S. embargo list against certain Russian individuals / companies. It is news to me that the MAINTAINERS file was usually containing Companies or that the Linux kernel development is Companies engaging with each other. I was under the naive assumption that it's individual developers who work together, and their employers do not really matter. Contributions are judged by their merit, and not by the author or their employer / affiliation. In the super unlikely case that indeed those individual developers removed from the MAINTAINERS file would be personally listed in the embargo list: Then yes, of course, I agree, they'd have to be removed. But then the commit log should of course point to [the version] of that list and explicitly mention that they were personally listed there. And no, I am of course not a friend of the Russian government at all. They are committing war crimes, no doubt about it. But since when has the collaboration of individual developers in an open source project been something related to actions completely unrelated to those individuals? Should I as a

German developer be excluded due to the track record of Germany having started two world wars killing millions? Should Americans be excluded due to a very extensive track record of violating international law? Should we exclude Palestinians? Israelis? Syrians? Iranians? [In case it's not obvious: Those are rhetorical questions, my position is of course no to all of them]. I just think there's nothing more wrong than discriminating against people just because of their passport, their employer or their place of residence. Maybe it's my German upbringing/socialization, but we've had multiple times in our history where the concept of **Sippenhaft** (kin liability) existed. In those dark ages of history you could be prosecuted for crimes committed by other family members. Now of course removal from the MAINTAINERS file or any other exclusion from the Linux kernel development process is of course not in any way comparable to prosecution like imprisonment or execution. However, the principle seems the same: An individual is punished for mere association with some others who happen to be committing crimes. Now if there really was a compelling legal argument for this (I doubt it, but let's assume for a second there is): In that case I'd expect a broad discussion against it; a reluctance to comply with it; a search for a way to circumvent said legal requirement; a petition or political movement against that requirement. Even if there was absolutely no way around performing such a "removal of names": At the very least I'd expect some civil disobedience by at least then introducing a statement into the file that one would have hoped to still be listing those individuals as co-maintainers but one was forced by [regulation, court order, ...] to remove them. But the least I would expect is for senior Kernel developers to simply do apply the patch with a one-sentence commit log message and thereby disrespect the work of said [presumed] Russian developers. All that does is to alienate individuals of the developer community. Not just those who are subject to said treatment today, but any others who see this sad example how Linux developers treat each other and feel discouraged from becoming or remaining active in a community with such behaviour. It literally hurts me personally to see this happening. It's like a kick in the gut. I used to be proud about having had an involvement with the Linux kernel community in a previous life. This doesn't feel like the community I remember being part of.

- Harald Welte: Back to Taiwan the first time after 5 years (2024/10/22 22:00)
  Some of the readers of this blog know that I have a very special relationship with Taiwan. As a teenager, it was the magical far-away country that built most of the PC components in all my PCs since my first 286-16 I got in 1989. Around 2006-2008 I had the very unexpected opportunity to work in Taiwan for some time (mainly for Openmoko, later some consulting for VIA). During that time I have always felt most welcome in and fascinated by the small island nation who managed to turn themselves into a high-tech development and manufacturing site for ever more complex electronics. And who managed to evolve from decades of military dictatorship and turn into a true democracy - all the while being discriminated by pretty much all of the countries around the world, as everybody wanted to benefit from cheap manufacturing in mainland China and hence expel democratic Taiwan from the united nations in favour of communist mainland Chine. I have the deepest admiration for Taiwan to manage all of their economic success and progress in terms of democracy and freedom despite the political situation across the Taiwan strait, and despite everything that comes along with it. May they continue to have the chance of continuing their path. Setting economy, society and politics behind: On a more personal level I've enjoyed their culinary marvels from excellent dumplings around every street corner to niu rou mien (beef noodle soup) to ma la huo guo (spicy hot pot). Plus then the natural beauty, particularly of the rural mountainous regions once you leave the densely populated areas around the coast line and the plains of the north west. While working in Taiwan in 2006/2007 I decided to buy a motorbike. Using that bike I've first made humble day trips and later (once I was no longer busy with stressful work at Openmoko) multiple week-long road trips around the island, riding on virtually any passable road you can find. My typical routing algorithm is "take the smallest possible road from A to B". So even after concluding my work in Taiwan, I returned again and again for holidays, each one with more road trips. For some

time, Taiwan had literally become my second home. I had my favorite restaurants, shops, as well as some places around the rural parts of the Island I cam back to several times. I even managed to take up some mandarin classes, something I never had the time for while doing [more than] full time work. To my big regret, it's still very humble beginner level; I guess had I not co-started a company (sysmocom) in Berlin in 2011, I'd have spent more time for a more serious story. In any case, I have nothing but the fondest memory of Taiwan. My frequent visits cam to a forcible halt with the COVID-19 pandemic, Taiwan was in full isolation in 2020/21, and even irrespective of government regulations, I've been very cautious about travel and contact. Plus of course, there's always the bad conscience of frequent intercontinental air travel. Originally I was planning to finally go on an extended Taiwan holiday in Summer 2024, but then the island was hit by a relatively serious earthquake in April, affecting particularly many of the remote mountain regions that are of main interest to me. There are some roads that I'd have wanted to ride ever since 2008, but which had been closed every successive year when I went there, due to years of reconstructions after [mostly landslides following] earthquakes and typhoons. So I decided to postpone it for another year to 2025. However, in an unexpected change of faith, the opportunity arose to give the opening Keyonte at the 2024 Open Compliance Summit in Japan, and along with that the opportunity to do a stop-over in Taiwan. It will just be a few days of Taipei this time (no motorbike trips), but I'm very much looking forward to being back in the city I probably know second or third-best on the planet (after Berlin, my home for 23 years, as well as Nuernberg, my place of birth). Let's see what is still the same and what has changed during the past 5 years!

- Harald Welte: Oral history transcripts: Pioneers of Taiwans Chip + PC industry (2024/10/22 22:00)
  During the preparation of my current brief visit to Taiwan, I've more or less by coincidence stumbled on several transcripts of oral history interviews with pioneers of the Taiwanese Chip and PC industry (click on the individual transcripts in the Related Records section at the bottom). They have been recorded, transcribed and translated in 2011 by the Computer History Museum under funding from the National Science Council, Taiwan, R.O.C.. As some of you know, I've been spending a lot of time in recent years researching (and practically exploring + re-implementing) historical telecommunications with my retronetworking project. Retrocomputing itself is not my main focus. I usually feel there's more than enough people operating, repairing, documenting at least many older computers, as well as keeping archives of related software and continuing to spread knowledge on how they operated. Nevertheless, it is a very interesting topic - I just decided that with my limited spare time I want to focus on retro-communications which is under-explored and under-represented. What's equally important than keeping the old technology alive, is keeping the knowledge around its creation alive. How did it happen that certain technologies were created and became successful or not? How where they key people behind it? etc. Given my personal history with Taiwan during the last 18 years, it's actually surprising I haven't yet given thought on how or where the history of the Taiwanese IT industry is documented or kept alive. So far I didn't know of any computer museums that would focus especially on the Taiwanese developments. It didn't even occur to me to even check if there are any. During my work in Taiwan I've had the chance to briefly meet a few senior people at FIC (large mainboard maker that made many PC mainboards I personally used) and both at VIA (chipset + CPU maker). But I didn't ever have a chance to talk about the history. In any case, I now found those transcripts of interviews. And what a trove of interesting first-hand information they are! If you have an interest in computer history, and want to understand how it came about that Taiwan became such a major player in either the PC industry or in the semiconductor design + manufacturing, then I believe those transcripts are a "must read". Now they've made me interested to learn more. I have little hope of many books being published on that subject, particularly in a Language I can read (i.e. English, not mandarin Chinese). But I shall research that subject. I'd also be interested to hear about any other information, like collections of historical artifacts, archives, libraries, etc. So in the unlikely case anybody reading this has

some pointers on information about the history of the Taiwanese Chip and Computer history, please by all means do reach out and share!. Once I have sufficiently prepared myself in reading whatever I can find in terms of written materials, I might be tempted to try to reach out and see if I can find some first-hand witnesses who'd want to share their stories on a future trip to Taiwan...

- Paul E. Mc Kenney: Parallel Programming: Cooperation (2024/10/10 17:16)
  First, let me paraphrase something from my LiveJournal profile: These posts are my own, and in particular do not necessarily reflect my employer's positions, strategies, or opinions. With that said, some say that the current geopolitical outlook is grim. And far be it from me to minimize the present-day geopolitical problems, nor am I at all interested in comparing them to their counterparts in the "good old days". But neither do I wish to obsess on these problems. I will instead call attention to a few instances of global cooperation, current and past. Last month, NASA's oldest active astronaut traveled to Kazakhstan's Baikonur Cosmodrome, entered a Soyuz capsule atop a Roscosmos rocket and flew to the International Space Station. For me, this is especially inspiring: If he can do that at age 69, I should certainly be able to continue doing my much less demanding job for many years to come. Some decades ago, during the Cold War, I purchased an English translation of Gradshteyn's and Ryzhik's classic "Table of Integrals, Series, and Products". Although computer-algebra systems have largely replaced this book, I have used it within the past few years and I used it heavily in the 1980s and early 1990s. Thus, along with many others, I am indebted to the longstanding Russian tradition of excellence in mathematics. So just this past month, I was happy to receive hard copies of "Параллельное программирование – так ли это сложно?", which is a Russian translation of "Is Parallel Programming Hard, And, If So, What Can You Do About It?" I would like to think that this might be a down payment on my aforementioned debt. Many other countries have also made many excellent contributions to mathematics, science, and technology. For example, the smartphone that I used hails from South Korea. And earlier this year, SeongJae (SJ) Park completed a Korean translation of the Second Edition of "Is Parallel Programming Hard, And, If So, What Can You Do About It?" Returning to rocketry, China started working with rockets in the 1200s, if not earlier, and has made a great deal of more recent progress in a wide variety of fields. And rumor has it that a Chinese translation of the Second Edition will be appearing shortly. So if you tried reading this book, but the English got in the way, you now have two other options and hopefully soon a third! But what if you want a fourth option? Then you, too, can do a translation! Just send me a translated chapter and I will add it to the list in the book's FAQ.txt file.
- Pete Zaitcev: Adventures in proprietary software, Solidworks edition (2024/10/06 16:39)
  Because FreeCAD was such a disaster for me, I started looking at crazy solutions, like exporting STEP from OpenSCAD. I even stooped to looking at proprietary alternatives. First on the runway was SolidWorks. If it's good for Mark Serbu, surely it's good for me, right? The first thing I found, you cannot tap your card and download. You have to contact a partner representative — never a good sign. The representative quoted me for untold thousands. I'm not going to post the amount, I'm sure they vary it every time, like small shop owners who vary prices according to the race of the shopper. In addition, they spam like you would not believe. First you have to unsubscribe from the partner, next from community.3ds.com, next from draftsight.3ds.com, and so on. Eventually, you'll get absolutely random spam, you try to unsubscribe, and they just continue and spam. Fortunately, I used a one-time address, and I killed it. Phew.
- Dave Airlie (blogspot): zinking the video (2024/10/04 01:00)
  A few years ago Mike and I discussed adding video support to zink, so that we could provide vaapi on top of vulkan video implementations.This of course got onto a long TODO list and we nerdsniped each other into moving it along, this past couple of weeks we finally dragged it over the line.This MR adds initial support for zink video decode on top of Vulkan Video. It provides vaapi support. Currently it only support H264 decode,

but I've implemented AV1 decode and I've played around a bit with H264 encode. I think adding H265 decode shouldn't be too horrible.I've tested this mainly on radv, and a bit on anv (but there are some problems I should dig into).

- Linux Plumbers Conference: That’s a wrap! Thanks everyone for Linux Plumbers 2024 (2024/09/23 09:07)
  Thank you to everyone who attended Linux Plumbers 2024 both in person and virtually! This year we were able to accommodate huge demand for in-person participation and we were glad to see more than 700 people in the Austria Center. As in previous years after the pandemic we also had a virtual component with more than 200 participants. We had a lot of great content in Refereed Track, Kernel Summit, eBPF and Networking Summits and Toolchains Track and a lot of productive discussions in 24 microconferences. There also were 25 Birds-of-a-Feather sessions, many of them were added during the event to continue a discussion that started in a microconference or in the Hallway Track. There are recordings of live streams and we hope to have recordings of all the sessions soon. Finally, I want to thank all those that were involved in making Linux Plumbers the best technical conference there is. This would not have happened without the hard work from the planning committee (Alice Ferrazzi, André Almeida, Christian Brauner, David Woodhouse, James Bottomley, Kate Stewart, Lorenzo Pieralisi, Shuah Khan, Song Liu, Steve Rostedt, Tim Bird), the runners of the Networking and BPF Summit tracks, the Toolchain track, Kernel Summit, and those that put together the very productive microconferences. I would also like to thank all those that presented as well as those who attended both in-person and virtually. I want to thank our sponsors for their continued support, without them Linux Plumbers Conference would not be possible. And a very special thanks to the Linux Foundation and their staff who did really great job behind the scenes and on-site to make this conference run smoothly. Their work is greatly appreciated by the LPC planning committee. Sincerely, Mike Rapoport Linux Plumbers 2024 Conference chair
- Linux Plumbers Conference: Playback of Presenter and BBB Training is available (2024/09/13 14:37)
  We recorded a playback of the 10:00 session which you can watch: https://bbb1.lpc.events/playback/presentation/2.3/62e3456da3c0598910e28d204ee24b669d714c04-1725975646004 To get a feel for how the BBB platform works. In addition, your credentials are the email address you registered with in cvent and the confirmation number of the registration it sent you back. You can use those to log in here: https://meet.lpc.events And practice in a Hackroom (after logging in select Hackrooms from the leftnav and then pick a Hackroom which is empty).
- Dave Airlie (blogspot): On Rust, Linux, developers, maintainers (2024/08/30 01:52)
  There's been a couple of mentions of Rust4Linux in the past week or two, one from Linus on the speed of engagement and one about Wedson departing the project due to non-technical concerns. This got me thinking about project phases and developer types.Archetypes:I will regret making an analogy, in an area I have no experience in, but let's give it a go with a road building analogy. Let's sort developers into 3 rough categories. Let's preface by saying not all developers fit in a single category throughout their careers, and some developers can do different roles on different projects, or on the same project simultaneously.1. Wayfinders/MapmakersI want to go build a hotel somewhere but there exists no map or path. I need to travel through a bunch of mountains, valleys, rivers, weather, animals, friendly humans, antagonistic humans and some unknowns. I don't care deeply about them, I want to make a path to where I want to go. I hit a roadblock, I don't focus on it, I get around it by any means necessary and move onto the next one. I document the route by leaving maps, signs. I build a hotel at the end. 2. Road buildersI see the hotel and path someone has marked out. I foresee that larger volumes will want to traverse this path and build more hotels. The roadblocks the initial finder worked around, I have to engage with. I engage with each roadblock differently. I build a bridge, dig a tunnel, blow up some stuff, work with with/against humans, whatever is necessary to get a road built to the place the wayfinder built the hotel. I work on each roadblock until

I can open the road to traffic. I can open it in stages, but it needs a completed road.3. Road maintainersI've got a road, I may have built the road initially. I may no longer build new roads. I've no real interest in hotels. I deal with intersections with other roads controlled by other people, I interact with builders who want to add new intersections for new roads, and remove old intersections for old roads. I fill in the holes, improve safety standards, handle the odd wayfinder wandering across my 8 lanes.Interactions:Wayfinders and maintainers is the most difficult interaction. Wayfinders like to move freely and quickly, maintainers have other priorities that slow them down. I believe there needs to be road builders engaged between the wayfinders and maintainers.Road builders have to be willing to expend the extra time to resolving roadblocks in the best way possible for all parties. The time it takes to resolve a single roadblock may be greater than the time expended on the whole wayfinding expedition, and this frustrates wayfinders. The builder has to understand what the maintainers concerns are and where they come from, and why the wayfinder made certain decisions. They work via education and trust building to get them aligned to move past the block. They then move down the road and repeat this process until the road is open. How this is done might change depending on the type of maintainers.Maintainer types:Maintainers can fall into a few different groups on a per-new road basis, and how do road builders deal with existing road maintainers depends on where they are for this particular intersection:1. Positive and engaged Aligned with the goal of the road, want to help out, design intersections, help build more roads and more intersections. Will often have helped wayfinders out.2. Positive with real concernsAgrees with the road's direction, might not like some of the intersections, willing to be educated and give feedback on newer intersection designs. Moves to group 1 or trusts that others are willing to maintain intersections on their road.3. Negative with real concernsDon't agree fully with road's direction or choice of building material. Might have some resistance to changing intersections, but may believe in a bigger picture so won't actively block. Hopefully can move to 1 or 2 with education and trust building. 4. Negative and unwillingDon't agree with the goal, don't want the intersection built, won't trust anyone else to care about their road enough. Education and trust building is a lot more work here, and often it's best to leave these intersections until later, where they may be swayed by other maintainers having built their intersections. It might be possible to build a reduced intersection. but if they are a major enough roadblock in a very busy road, then a higher authority might need to be brought in.5. Don't care/DisengagedDoesn't care where your road goes and won't talk about intersections. This category often just need to be told that someone else will care about it and they will step out of the way. If they are active blocks or refuse interaction then again a higher authority needs to be brought in.Where are we now?I think the r4l project has a had lot of excellent wayfinding done, has a lot of wayfinding in progress and probably has a bunch of future wayfinding to do. There are some nice hotels built. However now we need to build the roads to them so others can build hotels. To the higher authority, the road building process can look slow. They may expect cars to be driving on the road already, and they see roadblocks from a different perspective. A roadblock might look smaller to them, but have a lot of fine details, or a large roadblock might be worked through quickly once it's engaged with. For the wayfinders the process of interacting with maintainers is frustrating and slow, and they don't enjoy it as much as wayfinding, and because they still only care about the hotel at the end, when a maintainer gets into the details of their particular intersection they don't want to do anything but go stay in their hotel. The road will get built, it will get traffic on it. There will be tunnels where we should have intersections, there will be bridges that need to be built from both sides, but I do think it will get built.I think my request from this is that contributors should try and identify the archetype they currently resonate with and find the next group over to interact with.For wayfinders, it's fine to just keep wayfinding, just don't be surprised when the road building takes longer, or the road that gets built isn't what you envisaged.For road builder, just keep building, find new techniques for bridging gaps and blowing stuff up when appropriate. Figure out when to use higher authorities. Take the high road, and focus on the big picture.For maintainers, try and keep up

with modern road building, don't say 20 year old roads are the pinnacle of innovation. Be willing to install the rumble strips, widen the lanes, add crash guardrails, and truck safety offramps. Understand that wayfinders show you opportunities for longer term success and that road builders are going to keep building the road, and the result is better if you engage positively with them.

- Linux Plumbers Conference: Welcome to the Android Micro-conference! (2024/08/23 17:34)
  Every year the Android Micro-conference brings the upstream Linux community and the Android systems developers together at the Linux Plumbers Conference. They discuss how they can effectively engage the existing issues and collaborate on upcoming changes to the Android platform and their upstream dependencies. This year Android MC is scheduled to start at 10am on Friday, 20th Sep at Hall L1 (Austria Center). Attending Android MC gives you a chance to contribute to the broader discussion on Android platform ecosystem and Linux kernel development. You can share your own experiences, offer feedback, and help shape the future direction of these technologies. Discussion topics for this year include: Android kernel support and long term AOSP maintainership to support device longevity The pursuit of AOSP developer community Android Generic bootloader efforts Supporting generic restricted dmabuf heap memcg v2 updates in Android Bring-up devices with 16kb page_size support ublk based zero copy I/O use case in Android Leveraging large folios (mTHP) on Android phones Android MC will be followed by a Android BoF session, which will be a audience directed discussion. It can be a follow-up of the discussions from any of the Android MC topics or a free-form discussion on Android related topics.
- Matthew Garrett: What the fuck is an SBAT and why does everyone suddenly care (2024/08/22 08:52)
  Short version: Secure Boot Advanced Targeting and if that's enough for you you can skip the rest you're welcome.Long version: When UEFI Secure Boot was specified, everyone involved was, well, a touch naive. The basic security model of Secure Boot is that all the code that ends up running in a kernel-level privileged environment should be validated before execution - the firmware verifies the bootloader, the bootloader verifies the kernel, the kernel verifies any additional runtime loaded kernel code, and now we have a trusted environment to impose any other security policy we want. Obviously people might screw up, but the spec included a way to revoke any signed components that turned out not to be trustworthy: simply add the hash of the untrustworthy code to a variable, and then refuse to load anything with that hash even if it's signed with a trusted key.Unfortunately, as it turns out, scale. Every Linux distribution that works in the Secure Boot ecosystem generates their own bootloader binaries, and each of them has a different hash. If there's a vulnerability identified in the source code for said bootloader, there's a large number of different binaries that need to be revoked. And, well, the storage available to store the variable containing all these hashes is limited. There's simply not enough space to add a new set of hashes every time it turns out that grub (a bootloader initially written for a simpler time when there was no boot security and which has several separate image parsers and also a font parser and look you know where this is going) has another mechanism for a hostile actor to cause it to execute arbitrary code, so another solution was needed.And that solution is SBAT. The general concept behind SBAT is pretty straightforward. Every important component in the boot chain declares a security generation that's incorporated into the signed binary. When a vulnerability is identified and fixed, that generation is incremented. An update can then be pushed that defines a minimum generation - boot components will look at the next item in the chain, compare its name and generation number to the ones stored in a firmware variable, and decide whether or not to execute it based on that. Instead of having to revoke a large number of individual hashes, it becomes possible to push one update that simply says "Any version of grub with a security generation below this number is considered untrustworthy".So why is this suddenly relevant? SBAT was developed collaboratively between the Linux community and Microsoft, and Microsoft chose to push a Windows update that told systems not to trust versions of grub with a security generation below a certain level.

This was because those versions of grub had genuine security vulnerabilities that would allow an attacker to compromise the Windows secure boot chain, and we've seen real world examples of malware wanting to do that (Black Lotus did so using a vulnerability in the Windows bootloader, but a vulnerability in grub would be just as viable for this). Viewed purely from a security perspective, this was a legitimate thing to want to do.(An aside: the "Something has gone seriously wrong" message that's associated with people having a bad time as a result of this update? That's a message from shim, not any Microsoft code. Shim pays attention to SBAT updates in order to avoid violating the security assumptions made by other bootloaders on the system, so even though it was Microsoft that pushed the SBAT update, it's the Linux bootloader that refuses to run old versions of grub as a result. This is absolutely working as intended)The problem we've ended up in is that several Linux distributions had not shipped versions of grub with a newer security generation, and so those versions of grub are assumed to be insecure (it's worth noting that grub is signed by individual distributions, not Microsoft, so there's no externally introduced lag here). Microsoft's stated intention was that Windows Update would only apply the SBAT update to systems that were Windows-only, and any dual-boot setups would instead be left vulnerable to attack until the installed distro updated its grub and shipped an SBAT update itself. Unfortunately, as is now obvious, that didn't work as intended and at least some dual-boot setups applied the update and that distribution's Shim refused to boot that distribution's grub.What's the summary? Microsoft (understandably) didn't want it to be possible to attack Windows by using a vulnerable version of grub that could be tricked into executing arbitrary code and then introduce a bootkit into the Windows kernel during boot. Microsoft did this by pushing a Windows Update that updated the SBAT variable to indicate that known-vulnerable versions of grub shouldn't be allowed to boot on those systems. The distribution-provided Shim first-stage bootloader read this variable, read the SBAT section from the installed copy of grub, realised these conflicted, and refused to boot grub with the "Something has gone seriously wrong" message. This update was not supposed to apply to dual-boot systems, but did anyway. Basically:1) Microsoft applied an update to systems where that update shouldn't have been applied2) Some Linux distros failed to update their grub code and SBAT security generation when exploitable security vulnerabilities were identified in grubThe outcome is that some people can't boot their systems. I think there's plenty of blame here. Microsoft should have done more testing to ensure that dual-boot setups could be identified accurately. But also distributions shipping signed bootloaders should make sure that they're updating those and updating the security generation to match, because otherwise they're shipping a vector that can be used to attack other operating systems and that's kind of a violation of the social contract around all of this.It's unfortunate that the victims here are largely end users faced with a system that suddenly refuses to boot the OS they want to boot. That should never happen. I don't think asking arbitrary end users whether they want secure boot updates is likely to result in good outcomes, and while I vaguely tend towards UEFI Secure Boot not being something that benefits most end users it's also a thing you really don't want to discover you want after the fact so I have sympathy for it being default on, so I do sympathise with Microsoft's choices here, other than the failed attempt to avoid the update on dual boot systems.Anyway. I was extremely involved in the implementation of this for Linux back in 2012 and wrote the first prototype of Shim (which is now a massively better bootloader maintained by a wider set of people and that I haven't touched in years), so if you want to blame an individual please do feel free to blame me. This is something that shouldn't have happened, and unless you're either Microsoft or a Linux distribution it's not your fault. I'm sorry. comments

- Matthew Garrett: Client-side filtering of private data is a bad idea (2024/08/19 19:17)
  (The issues described in this post have been fixed, I have not exhaustively researched whether any other issues exist)Feeld is a dating app aimed largely at alternative relationship communities (think "classier Fetlife" for the most part), so unsurprisingly it's fairly popular in San Francisco. Their website makes the claim:Can people see what or who I'm looking for?No. You're the only person who can see which genders or sexualities

you're looking for. Your curiosity and privacy are always protected.which is based on you being able to restrict searches to people of specific genders, sexualities, or relationship situations. This sort of claim is one of those things that just sits in the back of my head worrying me, so I checked it out.First step was to grab a copy of the Android APK (there are multiple sites that scrape them from the Play Store) and run it through apk-mitm - Android apps by default don't trust any additional certificates in the device certificate store, and also frequently implement certificate pinning. apk-mitm pulls apart the apk, looks for known http libraries, disables pinning, and sets the appropriate manifest options for the app to trust additional certificates. Then I set up mitmproxy, installed the cert on a test phone, and installed the app. Now I was ready to start.What became immediately clear was that the app was using graphql to query. What was a little more surprising is that it appears to have been implemented such that there's no server state - when browsing profiles, the client requests a batch of profiles along with a list of profiles that the client has already seen. This has the advantage that the server doesn't need to keep track of a session, but also means that queries just keep getting larger and larger the more you swipe. I'm not a web developer, I have absolutely no idea what the tradeoffs are here, so I point this out as a point of interest rather than anything else.Anyway. For people unfamiliar with graphql, it's basically a way to query a database and define the set of fields you want returned. Let's take the example of requesting a user's profile. You'd provide the profile ID in question, and request their bio, age, rough distance, status, photos, and other bits of data that the client should show. So far so good. But what happens if we request other data?graphql supports introspection to request a copy of the database schema, but this feature is optional and was disabled in this case. Could I find this data anywhere else? Pulling apart the apk revealed that it's a React Native app, so effectively a framework for allowing writing of native apps in Javascript. Sometimes you'll be lucky and find the actual Javascript source there, but these days it's more common to find Hermes blobs. Fortunately hermes-dec exists and does a decent job of recovering something that approximates the original input, and from this I was able to find various lists of database fields.So, remember that original FAQ statement, that your desires would never be shown to anyone else? One of the fields mentioned in the app was "lookingFor", a field that wasn't present in the default profile query. What happens if we perform the incredibly complicated hack of exporting a profile query as a curl statement, add "lookingFor" into the set of requested fields, and run it?Oops.So, point 1 is that you can't simply protect data by having your client not ask for it - private data must never be released. But there was a whole separate class of issue that was an even more obvious issue.Looking more closely at the profile data returned, I noticed that there were fields there that weren't being displayed in the UI. Those included things like "ageRange", the range of ages that the profile owner was interested in, and also whether the profile owner had already "liked" or "disliked" your profile (which means a bunch of the profiles you see may already have turned you down, but the app simply didn't show that). This isn't ideal, but what was more concerning was that profiles that were flagged as hidden were still being sent to the app and then just not displayed to the user. Another example of this is that the app supports associating your profile with profiles belonging to partners - if one of those profiles was then hidden, the app would stop showing the partnership, but was still providing the profile ID in the query response and querying that ID would still show the hidden profile contents.Reporting this was inconvenient. There was no security contact listed on the website or in the app. I ended up finding Feeld's head of trust and safety on Linkedin, paying for a month of Linkedin Pro, and messaging them that way. I was then directed towards a HackerOne program with a link to terms and conditions that 404ed, and it took a while to convince them I was uninterested in signing up to a program without explicit terms and conditions. Finally I was just asked to email security@, and successfully got in touch. I heard nothing back, but after prompting was told that the issues were fixed - I then looked some more, found another example of the same sort of issue, and eventually that was fixed as well. I've now been informed that work has been done to ensure that this entire class of issue has been dealt with, but I haven't done any significant amount of work to ensure that that's

the case.You can't trust clients. You can't give them information and assume they'll never show it to anyone. You can't put private data in a database with no additional acls and just rely on nobody ever asking for it. You also can't find a single instance of this sort of issue and fix it without verifying that there aren't other examples of the same class. I'm glad that Feeld engaged with me earnestly and fixed these issues, and I really do hope that this has altered their development model such that it's not something that comes up again in future.(Edit to add: as far as I can tell, pictures tagged as "private" which are only supposed to be visible if there's a match were appropriately protected, and while there is a "location" field that contains latitude and longitude this appears to only return 0 rather than leaking precise location. I also saw no evidence that email addresses, real names, or any billing data was leaked in any way) comments

- Linux Plumbers Conference: Registration is now reopened (2024/08/16 10:02)
  It’s better late than never. This year there was a huge demand to attend Linux Plumbers Conference in person and at last we were able to add more places and reopen the registration.