

## Kernel Planet

- [James Bottomley: Adding Two Factor Authentication to Android \(LineageOS\)](#) (2026/01/22 17:17)

I really like the idea of using biometrics to add extra security, but have always hated the idea that simply touching the fingerprint sensor would unlock your entire phone, so in my version of LineageOS the touch to unlock feature is disabled but I still use second factor biometrics for the security of various apps. Effectively the android unlock policy is Fingerprint OR PIN/Pattern/Password and I simply want that OR to become an AND. The problem The idea of using two factor authentication (2FA) was pioneered by GrapheneOS but since I like the smallness of the Pixel 3 that's not available to me (plus it only seems to work with pin and fingerprint and my preferred unlock is pattern). However, since I build my own LineageOS anyway (so I can sign and secure boot it) I thought I'd look into adding the feature ... porting from GrapheneOS should be easy, right? In fact, when looking in the GrapheneOS code for frameworks/base, there are about nine commits adding the feature: a7a19bf8fb98 add second factor to fingerprint unlock5dd0e04f82cd add second factor UI9cc17fd97296 add second factor to FingerprintServicec92a23473f3f add second factor to LockPattern classesc504b05c933a add second factor to TrustManagerService0aa7b9ec8408 add second factor to AdaptiveAuthService62bbdf359687 add second factor to LockSettingsStateListener7429cc13f971 add second factor to LockSettingsService6e2d499a37a2 add second factor to DevicePolicyManagerService And a diffstat of over 3,000 lines ... which seems a bit much for changing an OR to an AND. Of course, the reason it's so huge is because they didn't change the OR, they implemented an entirely new bouncer (bouncer being the android term in the code for authorisation gateway) that did pin and fingerprint in addition to the other three bouncers doing pattern, pin and password. So not only would I have to port 3,000 lines of code, but if I want a bouncer doing fingerprint and pattern, I'd have to write it. I mean colour me lazy but that seems way too much work for such an apparently simple change. Creating a new 2FA unlock So is it actually easy? The rest of this post documents my quest to find out. Android code itself isn't always easy to read: being Java it's object oriented, but the curse of object orientation is that immediately after you've written the code, you realise you got the object model wrong and it needs to be refactored ... then you realise the same thing after the first refactor and so on until you either go insane or give up. Even worse when many people write the code they all end up with slightly different views of what the object model should be. The result is what you see in Android today: model inconsistency and redundancy which get in the way when you try to understand the code flow simply by reading it. One stroke of luck was that there is actually only a single method all of the unlock types other than fingerprint go through KeyguardSecurityContainerController.showNextSecurityScreenOrFinish() with fingerprint unlocking going via a listener to the KeyguardUpdateMonitorCallback.onBiometricAuthenticated(). And, thanks to already disabling fingerprint only unlock, I know that if I simply stop triggering the latter event, it's enough to disable fingerprint only unlock and all remaining authentication goes through the former callback. So to implement the required AND function, I just have to do this and check that a fingerprint authentication is also present in showNext.. (handily signalled by KeyguardUpdateMonitor.userUnlockedWithBiometric()). The latter being set fairly late in the sequence that does the onBiometricAuthenticated() callback (so I have to cut it off after this to prevent fingerprint only unlock). As part of the Android redundancy, there's already a check for fingerprint unlock as its own segment of a big if/else statement in the showNext.. code; it's probably a vestige from a different fingerprint unlock mechanism but I disabled it when the user enables 2FA just in case. There's also an insanely complex set of listeners for updating the messages on the lockscreen to guide the user through unlocking, which I decided not to change (if you enable 2FA, you need to know how to use it). Finally, I diverted the code that would call the onBiometricAuthenticated() and instead routed it to onBiometricDetected()

which triggers the LockScreen bouncer to pop up, so now you wake your phone, touch the fingerprint to the back, when authenticated, it pops up the bouncer and you enter your pin/pattern/password ... neat (and simple)! Well, not so fast. While the code above works perfectly if the lockscreen is listening for fingerprints, there are two cases where it doesn't: if the phone is in lockdown or on first boot (because the Android way of not allowing fingerprint only authentication for those cases is not to listen for it). At this stage, my test phone is actually unusable because I can never supply the required fingerprint for 2FA unlocking. Fortunately a rooted adb can update the 2FA in the secure settings service: simply run sqlite3 on /data/system/locksettings.db and flip user\_2fa from 1 to 0. The fingerprint listener is started in KeyguardUpdateMonitor, but it has a fairly huge set of conditions in updateFingerprintListeningState() which is also overloaded by doing detection as well as authentication. In the end it's not as difficult as it looks: shouldListenForFingerprint needs to be true and runDetect needs to be false. However, even then it doesn't actually work (although debugging confirms it's trying to start the fingerprint listening service); after a lot more debugging it turns out that the biometric server process, which runs fingerprint detection and authentication, also has a redundant check for whether the phone is encrypted or in lockdown and refuses to start if it is, which also now needs to return false for 2FA and bingo, it works in all circumstances. Conclusion The final diffstat for all of this is 5 files changed, 55 insertions(+), 3 deletions(-) So I'd say that is way simpler than the GrapheneOS one. All that remains is to add a switch for the setting (under the fingerprint settings) in packages/apps/Settings and it's done. If you're brave enough to try this for yourself you can go to my github account and get both the frameworks and settings commits (if you don't want fingerprint unlock disable when 2FA isn't selected, you'll have to remove the head commit in frameworks). I suppose I should also add I've up-ported all of my other security stuff and am now on Android-15 (LineageOS-22.2).

- [Pete Zaitcev: The fall of LJ](#) (2026/01/04 20:32)

Great, I am unable to comment at BG. Theoretically, I have a spare place at Meenuvia, but that platform is also in decline. The owner, Pixy, has no time even to fix the slug problem that cropped up a few months ago (how do you regress a platform that was stable for 20 years, I don't know). Most likely, I'll give up on blogging entirely, and move to Twitter or Fediverse.

- [Matthew Garrett: What is a PC compatible?](#) (2026/01/04 03:11)

Wikipedia says "An IBM PC compatible is any personal computer that is hardware- and software-compatible with the IBM Personal Computer (IBM PC) and its subsequent models". But what does this actually mean? The obvious literal interpretation is for a device to be PC compatible, all software originally written for the IBM 5150 must run on it. Is this a reasonable definition? Is it one that any modern hardware can meet? Before we dig into that, let's go back to the early days of the x86 industry. IBM had launched the PC built almost entirely around off-the-shelf Intel components, and shipped full schematics in the IBM PC Technical Reference Manual. Anyone could buy the same parts from Intel and build a compatible board. They'd still need an operating system, but Microsoft was happy to sell MS-DOS to anyone who'd turn up with money. The only thing stopping people from cloning the entire board was the BIOS, the component that sat between the raw hardware and much of the software running on it. The concept of a BIOS originated in CP/M, an operating system originally written in the 70s for systems based on the Intel 8080. At that point in time there was no meaningful standardisation - systems might use the same CPU but otherwise have entirely different hardware, and any software that made assumptions about the underlying hardware wouldn't run elsewhere. CP/M's BIOS was effectively an abstraction layer, a set of code that could be modified to suit the specific underlying hardware without needing to modify the rest of the OS. As long as applications only called BIOS functions, they didn't need to care about the underlying hardware and would run on all systems that had a working CP/M port. By 1979, boards based on the 8086, Intel's successor to the 8080, were hitting the market. The 8086 wasn't machine code compatible

with the 8080, but 8080 assembly code could be assembled to 8086 instructions to simplify porting old code. Despite this, the 8086 version of CP/M was taking some time to appear, and a company called Seattle Computer Products started producing a new OS closely modelled on CP/M and using the same BIOS abstraction layer concept. When IBM started looking for an OS for their upcoming 8088 (an 8086 with an 8-bit data bus rather than a 16-bit one) based PC, a complicated chain of events resulted in Microsoft paying a one-off fee to Seattle Computer Products, porting their OS to IBM's hardware, and the rest is history. But one key part of this was that despite what was now MS-DOS existing only to support IBM's hardware, the BIOS abstraction remained, and the BIOS was owned by the hardware vendor - in this case, IBM. One key difference, though, was that while CP/M systems typically included the BIOS on boot media, IBM integrated it into ROM. This meant that MS-DOS floppies didn't include all the code needed to run on a PC - you needed IBM's BIOS. To begin with this wasn't obviously a problem in the US market since, in a way that seems extremely odd from where we are now in history, it wasn't clear that machine code was actually copyrightable. In 1982 *Williams v. Artic* determined that it could be even if fixed in ROM - this ended up having broader industry impact in *Apple v. Franklin* and it became clear that clone machines making use of the original vendor's ROM code wasn't going to fly. Anyone wanting to make hardware compatible with the PC was going to have to find another way. And here's where things diverge somewhat. Compaq famously performed clean-room reverse engineering of the IBM BIOS to produce a functionally equivalent implementation without violating copyright. Other vendors, well, were less fastidious - they came up with BIOS implementations that either implemented a subset of IBM's functionality, or didn't implement all the same behavioural quirks, and compatibility was restricted. In this era several vendors shipped customised versions of MS-DOS that supported different hardware (which you'd think wouldn't be necessary given that's what the BIOS was for, but still), and the set of PC software that would run on their hardware varied wildly. This was the era where vendors even shipped systems based on the Intel 80186, an improved 8086 that was both faster than the 8086 at the same clock speed and was also available at higher clock speeds. Clone vendors saw an opportunity to ship hardware that outperformed the PC, and some of them went for it. You'd think that IBM would have immediately jumped on this as well, but no - the 80186 integrated many components that were separate chips on 8086 (and 8088) based platforms, but crucially didn't maintain compatibility. As long as everything went via the BIOS this shouldn't have mattered, but there were many cases where going via the BIOS introduced performance overhead or simply didn't offer the functionality that people wanted, and since this was the era of single-user operating systems with no memory protection, there was nothing stopping developers from just hitting the hardware directly to get what they wanted. Changing the underlying hardware would break them. And that's what happened. IBM was the biggest player, so people targeted IBM's platform. When BIOS interfaces weren't sufficient they hit the hardware directly - and even if they weren't doing that, they'd end up depending on behavioural quirks of IBM's BIOS implementation. The market for DOS-compatible but not PC-compatible mostly vanished, although there were notable exceptions - in Japan the PC-98 platform achieved significant success, largely as a result of the Japanese market being pretty distinct from the rest of the world at that point in time, but also because it actually handled Japanese at a point where the PC platform was basically restricted to ASCII or minor variants thereof. So, things remained fairly stable for some time. Underlying hardware changed - the 80286 introduced the ability to access more than a megabyte of address space and would promptly have broken a bunch of things except IBM came up with an utterly terrifying hack that bit me back in 2009, and which ended up sufficiently codified into Intel design that it was one mechanism for breaking the original XBox security. The first 286 PC even introduced a new keyboard controller that supported better keyboards but which remained backwards compatible with the original PC to avoid breaking software. Even when IBM launched the PS/2, the first significant rearchitecture of the PC platform with a brand new expansion bus and associated patents to prevent people cloning it without paying off IBM, they made sure that all the hardware was backwards

compatible. For decades, PC compatibility meant not only supporting the officially supported interfaces, it meant supporting the underlying hardware. This is what made it possible to ship install media that was expected to work on any PC, even if you'd need some additional media for hardware-specific drivers. It's something that still distinguishes the PC market from the ARM desktop market. But it's not as true as it used to be, and it's interesting to think about whether it ever was as true as people thought. Let's take an extreme case. If I buy a modern laptop, can I run 1981-era DOS on it? The answer is clearly no. First, modern systems largely don't implement the legacy BIOS. The entire abstraction layer that DOS relies on isn't there, having been replaced with UEFI. When UEFI first appeared it generally shipped with a Compatibility Services Module, a layer that would translate BIOS interrupts into UEFI calls, allowing vendors to ship hardware with more modern firmware and drivers without having to duplicate them to support older operating systems<sup>1</sup>. Is this system PC compatible? By the strictest of definitions, no. Ok. But the hardware is broadly the same, right? There's projects like CSMWrap that allow a CSM to be implemented on top of stock UEFI, so everything that hits BIOS should work just fine. And well yes, assuming they implement the BIOS interfaces fully, anything using the BIOS interfaces will be happy. But what about stuff that doesn't? Old software is going to expect that my Sound Blaster is going to be on a limited set of IRQs and is going to assume that it's going to be able to install its own interrupt handler and ACK those on the interrupt controller itself and that's really not going to work when you have a PCI card that's been mapped onto some APIC vector, and also if your keyboard is attached via USB or SPI then reading it via the CSM will work (because it's calling into UEFI to get the actual data) but trying to read the keyboard controller directly won't<sup>2</sup>, so you're still actually relying on the firmware to do the right thing but it's not, because the average person who wants to run DOS on a modern computer owns three fursuits and some knee length socks and while you are important and vital and I love you all you're not enough to actually convince a transglobal megacorp to flip the bit in the chipset that makes all this old stuff work. But imagine you are, or imagine you're the sort of person who (like me) thinks writing their own firmware for their weird Chinese Thinkpad knockoff motherboard is a good and sensible use of their time - can you make this work fully? Haha no of course not. Yes, you can probably make sure that the PCI Sound Blaster that's plugged into a Thunderbolt dock has interrupt routing to something that is absolutely no longer an 8259 but is pretending to be so you can just handle IRQ 5 yourself, and you can probably still even write some SMM code that will make your keyboard work, but what about the corner cases? What if you're trying to run something built with IBM Pascal 1.0? There's a risk that it'll assume that trying to access an address just over 1MB will give it the data stored just above 0, and now it'll break. It'd work fine on an actual PC, and it won't work here, so are we PC compatible? That's a very interesting abstract question and I'm going to entirely ignore it. Let's talk about PC graphics<sup>3</sup>. The original PC shipped with two different optional graphics cards - the Monochrome Display Adapter and the Color Graphics Adapter. If you wanted to run games you were doing it on CGA, because MDA had no mechanism to address individual pixels so you could only render full characters. So, even on the original PC, there was software that would run on some hardware but not on other hardware. Things got worse from there. CGA was, to put it mildly, shit. Even IBM knew this - in 1984 they launched the PCjr, intended to make the PC platform more attractive to home users. As well as maybe the worst keyboard ever to be associated with the IBM brand, IBM added some new video modes that allowed displaying more than 4 colours on screen at once<sup>4</sup>, and software that depended on that wouldn't display correctly on an original PC. Of course, because the PCjr was a complete commercial failure, it wouldn't display correctly on any future PCs either. This is going to become a theme. There's never been a properly specified PC graphics platform. BIOS support for advanced graphics modes<sup>5</sup> ended up specified by VESA rather than IBM, and even then getting good performance involved hitting hardware directly. It wasn't until Microsoft specced DirectX that anything was broadly usable even if you limited yourself to Microsoft platforms, and this was an OS-level API rather than a hardware one. If you stick to BIOS interfaces then CGA-era code will

work fine on graphics hardware produced up until the 20-teens, but if you were trying to hit CGA hardware registers directly then you're going to have a bad time. This isn't even a new thing - even if we restrict ourselves to the authentic IBM PC range (and ignore the PCjr), by the time we get to the Enhanced Graphics Adapter we're not entirely CGA compatible. Is an IBM PC/AT with EGA PC compatible? You'd likely say "yes", but there's software written for the original PC that won't work there. And, well, let's go even more basic. The original PC had a well defined CPU frequency and a well defined CPU that would take a well defined number of cycles to execute any given instruction. People could write software that depended on that. When CPUs got faster, some software broke. This resulted in systems with a Turbo Button - a button that would drop the clock rate to something approximating the original PC so stuff would stop breaking. It's fine, we'd later end up with Windows crashing on fast machines because hardware details will absolutely bleed through. So, what's a PC compatible? No modern PC will run the DOS that the original PC ran. If you try hard enough you can get it into a state where it'll run most old software, as long as it doesn't have assumptions about memory segmentation or your CPU or want to talk to your GPU directly. And even then it'll potentially be unusable or crash because time is hard. The truth is that there's no way we can technically describe a PC Compatible now - or, honestly, ever. If you sent a modern PC back to 1981 the media would be amazed and also point out that it didn't run Flight Simulator. "PC Compatible" is a socially defined construct, just like "Woman". We can get hung up on the details or we can just chill. Windows 7 is entirely happy to boot on UEFI systems except that it relies on being able to use a BIOS call to set the video mode during boot, which has resulted in things like UEFISeven to make that work on modern systems that don't provide BIOS compatibility ↵ Back in the 90s and early 2000s operating systems didn't necessarily have native drivers for USB input devices, so there was hardware support for trapping OS accesses to the keyboard controller and redirecting that into System Management Mode where some software that was invisible to the OS would speak to the USB controller and then fake a response anyway that's how I made a laptop that could boot unmodified MacOS X ↵ (my name will not be Wolfwings Shadowflight) ↵ Yes yes ok 8088 MPH demonstrates that if you really want to you can do better than that on CGA ↵ and by advanced we're still talking about the 90s, don't get excited ↵

- [Greg Kroah-Hartman: Linux kernel security work](#) (2026/01/02 00:00)

Lots of the CVE world seems to focus on "security bugs" but I've found that it is not all that well known exactly how the Linux kernel security process works. I gave a talk about this back in 2023 and at other conferences since then, attempting to explain how it works, but I also thought it would be good to explain this all in writing as it is required to know this when trying to understand how the Linux kernel CNA issues CVEs.

- [Linux Plumbers Conference: Video recordings are available](#) (2025/12/16 05:35)

We are glad to announce that video recordings of the talks are available on our YouTube channel. You can use the complete conference playlist or look for "video" links in each contribution in the schedule

- [Greg Kroah-Hartman: Tracking kernel commits across branches](#) (2025/12/15 00:00)

With all of the different Linux kernel stable releases happening (at least 1 stable branch and multiple longterm branches are active at any one point in time), keeping track of what commits are already applied to what branch, and what branch specific fixes should be applied to, can quickly get to be a very complex task if you attempt to do this manually. So I've created some tools to help make my life easier when doing the stable kernel maintenance work, which ended up making the work of tracking CVEs much simpler to manage in an automated way.

- [James Morris: Ultraviolet Linux Talk at Linux Plumbers Conf 2025](#) (2025/12/14 03:38)

I presented an overview of the Ultraviolet Linux (UV) project at Linux Plumbers Conference (LPC) 2025. UV is a proposed architecture and reference implementation for generalized code integrity in Linux. The goal of the presentation was to seek early feedback from the community

and to invite collaboration — it's at an early stage of development currently. A copy of the slides may be found [here \(pdf\)](#).

- [Greg Kroah-Hartman: Linux kernel version numbers](#) (2025/12/09 00:00)

Despite having a stable release model and cadence since December 2003, Linux kernel version numbers seem to baffle and confuse those that run across them, causing numerous groups to mistakenly make versioning statements that are flat out false. So let's go into how this all works in detail.

- [Greg Kroah-Hartman: Linux CVEs, more than you ever wanted to know](#) (2025/12/08 00:00)

It's been almost 2 full years since Linux became a CNA (Certificate Numbering Authority) which meant that we (i.e. the kernel.org community) are now responsible for issuing all CVEs for the Linux kernel. During this time, we've become one of the largest creators of CVEs by quantity, going from nothing to number 3 in 2024 to number 1 in 2025. Naturally, this has caused some questions about how we are both doing all of this work, and how people can keep track of it.

- [Brendan Gregg: Leaving Intel](#) (2025/12/04 13:00)

InnovatiON 2022 AI Flame Graphs GPU Flame Scope Harshad Sane SREcon APAC Cloud strategy Last day I've resigned from Intel and accepted a new opportunity. If you are an Intel employee, you might have seen my fairly long email that summarized what I did in my 3.5 years. Much of this is public: AI flame graphs and released them as open source GPU subsecond-offset heatmap Worked with Linux distros to enable stack walking Was interviewed by the WSJ about eBPF for security monitoring Provided leadership on the eBPF Technical Steering Committee (BSC) Co-chaired USENIX SREcon APAC 2023 Gave 6 conference keynotes It's still early days for AI flame graphs. Right now when I browse CPU performance case studies on the Internet, I'll often see a CPU flame graph as part of the analysis. We're a long way from that kind of adoption for GPUs (and it doesn't help that our open source version is Intel only), but I think as GPU code becomes more complex, with more layers, the need for AI flame graphs will keep increasing. I also supported cloud computing, participating in 110 customer meetings, and created a company-wide strategy to win back the cloud with 33 specific recommendations, in collaboration with others across 6 organizations. It is some of my best work and features a visual map of interactions between all 19 relevant teams, described by Intel long-timers as the first time they have ever seen such a cross-company map. (This strategy, summarized in a slide deck, is internal only.) I always wish I did more, in any job, but I'm glad to have contributed this much especially given the context: I overlapped with Intel's toughest 3 years in history, and I had a hiring freeze for my first 15 months. My fond memories from Intel include meeting Linus at an Intel event who said "everyone is using flame graphs these days" (Finnish accent), meeting Pat Gelsinger who knew about my work and introduced me to everyone at an exec all hands, surfing lessons at an Intel Australia and HP offsite (mp4), and meeting Harshad Sane (Intel cloud support engineer) who helped me when I was at Netflix and now has joined Netflix himself -- we've swapped ends of the meeting table. I also enjoyed meeting Intel's hardware fellows and senior fellows who were happy to help me understand processor internals. (Unrelated to Intel, but if you're a Who fan like me, I recently met some other people as well!) My next few years at Intel would have focused on execution of those 33 recommendations, which Intel can continue to do in my absence. Most of my recommendations aren't easy, however, and require accepting change, ELT/CEO approval, and multiple quarters of investment. I won't be there to push them, but other employees can (my CloudTeams strategy is in the inbox of various ELT, and in a shared folder with all my presentations, code, and weekly status reports). This work will hopefully live on and keep making Intel stronger. Good luck.

- [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. These virtual Brendans sound like my brain has been uploaded to the cloud by someone 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: 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. On some occasions when 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 rushing to conclude 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 research tasks, like finding URLs for release notes and recent developments for a given technology. I once used ChatGPT to find an 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 long as end users understand it means a Brendan-like agent and not a full virtual Brendan. 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. So over time agent finds fewer wins as what it does find gets fixed in the target software. 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 good. 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." I learned about Granulate in my first days at Intel. I was told their product was entirely based on my work, using flame graphs for code profiling and my publications for tuning, and that as part of my new job I had to support it. It was also a complex project, as there was also a lot of infrastructure code for safe orchestration of tuning changes, which is not an easy problem. Flame graphs were the key interface: the first time I saw them demo their product they wanted to highlight their dynamic version of 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). 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 Perflnsights 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 my job (see my prior post for "What do performance engineers 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 suggests, 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](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 similar 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 extension 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](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](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](mailto:visaletters@linuxfoundation.org)

- [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

4b813c322ebe236cddc6b3acd70a31994efd7a56 Solution: 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 --amend Nowhere 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 available] Solution: \$ 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!).

- [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 out 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.

- [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.

- [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 breakout 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-adversarial 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/diversify 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 prporiety 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 no 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.<sup>6</sup> 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.<sup>7</sup> 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 now supports all vulkan decode/encode formats currently available.

- [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](https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/35398)

- [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 stomachs. 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). Northern/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 adjusted (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.

- [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/](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.

- [Pete Zaitcev: virtio\\_pci: do not wait forever 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 emulates 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 surprised 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 notification 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 Hubert 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 2030<sup>1</sup>. 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 support 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-compiled 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. Profilers. 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 disassembly 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 Degroot, 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 retronetwoking 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 draftsgt.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 lefnav 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/Mapmakers** I 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 builders** I 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 maintainers** I'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 concerns** Agrees 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 concerns** Don'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 unwilling** Don'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/Disengaged** Doesn'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 had a 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.

- [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.

- [Lucas De Marchi: Linux module dependencies](#) (2024/08/13 17:00)

With the imminent release of kmod 33, I thought it'd be good to have a post about the different types of module dependencies that we have in the Linux kernel and kmod. The new version adds another type, weak dependency, and as the name implies, is the weakest of all. But let's revisit what are the other types first. Hard (symbol) dependency This is the first dependency that every appeared in kmod (and module-init-tools). A

hard (or as some call, “symbol”) dependency occurs when your module calls or uses an exported symbol of another module. The most common way is by calling a function that is exported in another module. Example: the xe.ko calls a function ttm\_bo\_pin() that is provided and exported by another module, ttm.ko. Looking to the source: \$ nm build/drivers/gpu/drm/ttm/ttm.ko | grep -e "\bttm\_bo\_pin\b" 00000000000000bc0 T ttm\_bo\_pin \$ nm build64/drivers/gpu/drm/xe/xe.ko | grep -e "\bttm\_bo\_pin\b" U ttm\_bo\_pin It is not possible to insert the xe.ko module before ttm.ko and if you try via insmod (that doesn’t handle dependencies) it will fail with the kernel complaining that the ttm\_bo\_pin symbol is undefined. The manual invocations to nm illustrates what the depmod tool does: it opens the modules and reads the ELF headers. Then it takes note of all the symbols required and provided by each module, creating a graph of symbol dependencies. Ultimately that leads to module dependencies: xe → ttm. This is recorded in the modules.dep file and its sibling modules.dep.bin. The former is human-readable and the latter is used by libkmod, but they contain the same information: all dependencies for all the modules. Also note that each line reflects indirect dependencies: module A calls symbol from B and B calls symbol from C will lead to A depending on both B and C. Real world example: \$ cat /lib/modules/\$(uname -r)/modules.dep | grep kernel/drivers/gpu/drm/xe/xe.ko.zst: kernel/drivers/gpu/drm/xe/xe.ko.zst: kernel/drivers/gpu/drm/drm\_gpuvm.ko.zst kernel/drivers/gpu/drm/drm\_exec.ko.zst kernel/drivers/gpu/drm/scheduler/gpu-sched.ko.zst kernel/drivers/gpu/drm/drm\_buddy.ko.zst kernel/drivers/i2c/algos/i2c-algo-bit.ko.zst kernel/drivers/gpu/drm/drm\_suballoc\_helper.ko.zst kernel/drivers/gpu/drm/drm\_ttm\_helper.ko.zst kernel/drivers/gpu/drm/ttm.ko.zst kernel/drivers/gpu/drm/display/drm\_display\_helper.ko.zst kernel/drivers/media/cec/core/cec.ko.zst kernel/drivers/acpi/video.ko.zst kernel/drivers/platform/x86/wmi.ko.zst So the xe.ko (with .zst extension since it’s compressed) directly or indirectly depends on drm\_gpuvm.ko, drm\_exec.ko, gpu-sched.ko, drm\_buddy.ko, i2c-algo-bit.ko, drm\_suballoc\_helper.ko, drm\_ttm\_helper.ko, ttm.ko, drm\_display\_helper.ko, cec.ko, video.ko, wmi.ko. Same information, but using the kmod tools rather than looking at the raw index: \$ modinfo -F depends xe

drm\_display\_helper,ttm,drm\_gpuvm,drm\_suballoc\_helper,video,drm\_buddy,drm\_exec,drm\_ttm\_helper,gpu-sched,cec,i2c-algo-bit

Soft dependency There are situations in the kernel where it’s not possible or desired to use a symbol directly from another module - they may interact by registering in a subsystem, scanning a bus etc.. In this case depmod doesn’t have enough information from the ELF file about that. Yet the user would have a more complete support if both modules were available - it may even cause failures visible to the end user and not only “partial support for features”. The softdep implementation contains 2 parts: pre and post dependencies. The post dependencies are not very much used in practice: they instruct kmod to load another module after loading the target one. They can come from a configuration file like e.g. /etc/modprobe.d/foo.conf or from the kernel itself by embedding that info in the module. From the kernel source this is achieved by using the macro MODULE\_SOFTDEP(). Example: lib/libcrc32c.c:MODULE\_SOFTDEP("pre: crc32c"); When libkmod is loading a module it will first load, in order: hard dependencies soft pre dependencies target module soft post dependencies Historically softdeps were also a way to (mostly) get rid of install rules, in which the configuration instructs libkmod to execute something instead of loading the module - people would add an install rule to execute something and then call modprobe again with --ignore-install to fake a dependency. That could easily lead to a runtime loop which is avoided with softdep since kmod can (and does) check for loops. Weak dependency After explaining the other types of dependencies, back to the new addition in kmod 33. These are very similar to pre softdep: they come either from a configuration file or embedded in the module and they express a dependency that wouldn’t cause the target module to fail to load, but that may cause the initialization to export less features or fail while initializing. There is one important difference: weak dependencies don’t cause libkmod to actually load the module. Rather the dependency information may be used by tools like dracut and other tools responsible for assembling an initrd to make the module available since it may or

may not be used. Why are they called “weak”? This was a borrowed terminology from “weak symbols”: a weak symbol is there, waiting to be used, but it may or may not be, with the final decision happening in the final link stage. With weak dependencies, hopefully some of the pre softdep embedded in the kernel may be replaced: if the target module is already doing a `request_module()` or in some way getting the other module to be loaded, it doesn’t need a softdep that would serialize the module load order and possibly load more modules than required.

- [Pete Zaitcev: Fedora Panic Canceled \(2024/07/30 19:06\)](#)

The other day I was watching a video by Rich Jones about Fedora on RISC-V. In it, he mentions off-hand that CentOS Stream 10 inherits from Fedora 40. I don't know how that happened anymore, but previously someone made me think that 1. there will be no more numbered releases of CentOS, which is why it is called "CentOS Stream" now, and 2. CentOS Stream is now the upstream of RHEL, replacing Fedora. I really was concerned for the future of Fedora, that was superfluous in that arrangement, you know! But apparently, Fedora is still the trunk upstream, and CentOS Stream is only named like that. Nothing changes except CentOS is no longer a clone of RHEL, but instead RHEL is a clone of CentOS. What was all the panic for? I made a VM at Kamatera a few months ago, and they didn't even have Fedora among images. I ended using Rocky.

- [Brendan Gregg: No More Blue Fridays \(2024/07/21 14:00\)](#)

In the future, computers will not crash due to bad software updates, even those updates that involve kernel code. In the future, these updates will push eBPF code. Friday July 19th provided an unprecedented example of the inherent dangers of kernel programming, and has been called the largest outage in the history of information technology. Windows computers around the world encountered blue-screens-of-death and boot loops, causing outages for hospitals, airlines, banks, grocery stores, media broadcasters, and more. This was caused by a config update by a security company for their widely used product that included a kernel driver on Windows systems. The update caused the kernel driver to try to read invalid memory, an error type that will crash the kernel. For Linux systems, the company behind this outage was already in the process of adopting eBPF, which is immune to such crashes. Once Microsoft's eBPF support for Windows becomes production-ready, Windows security software can be ported to eBPF as well. These security agents will then be safe and unable to cause a Windows kernel crash. eBPF (no longer an acronym) is a secure kernel execution environment, similar to the secure JavaScript runtime built into web browsers. If you're using Linux, you likely already have eBPF available on your systems whether you know it or not, as it was included in the kernel several years ago. eBPF programs cannot crash the entire system because they are safety-checked by a software verifier and are effectively run in a sandbox. If the verifier finds any unsafe code, the program is rejected and not executed. The verifier is rigorous -- the Linux implementation has over 20,000 lines of code -- with contributions from industry (e.g., Meta, Isovalent, Google) and academia (e.g., Rutgers University, University of Washington). The safety this provides is a key benefit of eBPF, along with heightened security and lower resource usage. Some eBPF-based security startups (e.g., Oligo, Uptycs) have made their own statements about the recent outage, and the advantages of migrating to eBPF. Larger tech companies are also adopting eBPF for security. As an example, Cisco acquired the eBPF-startup Isovalent and has announced a new eBPF security product: Cisco Hypershield, a fabric for security enforcement and monitoring. Google and Meta already rely on eBPF to detect and stop bad actors in their fleet, thanks to eBPF's speed, deep visibility, and safety guarantees. Beyond security, eBPF is also used for networking and observability. The worst thing an eBPF program can do is to merely consume more resources than is desirable, such as CPU cycles and memory. eBPF cannot prevent developers writing poor code -- wasteful code -- but it will prevent serious issues that cause a system to crash. That said, as a new technology eBPF has had some bugs in its management code, including a Linux kernel panic discovered by the same security company in the news today. This doesn't mean that eBPF has solved nothing, substituting a vendor's bug for its own. Fixing these bugs in eBPF means fixing these bugs for all

eBPF vendors, and more quickly improving the security of everyone. There are other ways to reduce risks during software deployment that can be employed as well: canary testing, staged rollouts, and "resilience engineering" in general. What's important about the eBPF method is that it is a software solution that will be available in both Linux and Windows kernels by default, and has already been adopted for this use case. If your company is paying for commercial software that includes kernel drivers or kernel modules, you can make eBPF a requirement. It's possible for Linux today, and Windows soon. While some vendors have already proactively adopted eBPF (thank you), others might need a little encouragement from their paying customers. Please help raise awareness, and together we can make such global outages a lesson of the past. Authors: Brendan Gregg, Intel; Daniel Borkmann, Isovalent; Joe Stringer, Isovalent; KP Singh, Google.

- [Linux Plumbers Conference: System Boot and Security Microconference CFP \(2024/07/12 19:38\)](#)

The System Boot and Security Microconference has been a critical platform for enthusiasts and professionals working on firmware, bootloaders, system boot, and security. This year, the conference focuses on the challenges of upstreaming boot process improvements to the Linux kernel. Cryptography, an ever-evolving field, poses unique demands on secure elements and TPMs as newer algorithms are introduced and older ones are deprecated. Additionally, new hardware architectures with DRTM capabilities, such as ARM's D-RTM specification and the increased use of fTPMs in innovative applications, add to the complexity of the task. This is the fifth time the conference has been held in the last six years. Trusted Platform Modules (TPMs) for encrypting disks have become widespread across various distributions. This highlights the vital role that TPMs play in ensuring platform security. As the field of confidential computing continues to grow, virtual machine firmware must evolve to meet end-users' demands, and Linux would have to leverage exposed capabilities to provide relevant security properties. Mechanisms like UEFI Secure Boot that were once limited to OEMs now empower end-users. The System Boot and Security Microconference aims to address these challenges collaboratively and transparently. We welcome talks on the following technologies that can help achieve this goal. TPMs, HSMs, secure elements Roots of Trust: SRTM and DRTM Intel TXT, SGX, TDX AMD SKINIT, SEV ARM DRTM Growing Attestation ecosystem IMA TrenchBoot, tboot TianoCore EDK II (UEFI), SeaBIOS, coreboot, U-Boot, LinuxBoot, hostboot Measured Boot, Verified Boot, UEFI Secure Boot, UEFI Secure Boot Advanced Targeting (SBAT) shim boot loaders: GRUB2, systemd-boot/sd-boot, network boot, PXE, iPXE UKI u-root OpenBMC, u-bmc legal, organizational, and other similar issues relevant to people interested in system boot and security. If you want to participate in this microconference and have ideas to share, please use the Call for Proposals (CFP) process. Your submissions should focus on new advancements, innovations, and solutions related to firmware, bootloader, and operating system development. It's essential to explain clearly what will be discussed, why, and what outcomes you expect from the discussion. Edit: The submission deadline has been updated to July 14th!

- [Linux Plumbers Conference: Sched-Ext: The BPF extensible scheduler class Microconference CFP \(2024/07/12 02:38\)](#)

sched\_ext is a Linux kernel feature which enables implementing host-wide, safe kernel thread schedulers in BPF, and dynamically loading them at runtime. sched\_ext enables safe and rapid iterations of scheduler implementations, thus radically widening the scope of scheduling strategies that can be experimented with and deployed, even in massive and complex production environments. sched\_ext was first sent to the upstream list as an RFC patch set back in November 2022. Since then, the project has evolved a great deal, both technically, as well as in the significant growth of the community of sched\_ext users and contributors. This MC is the space for the community to discuss the developments of sched\_ext, its impact on the community, and to outline future strategies aimed at integrating this feature into the Linux kernel and mainstream Linux distributions. Ideas of topics to be discussed include (but are not limited to): Challenges and plans to facilitate the upstream merge of sched\_ext User-space scheduling (offload part / all of the scheduling from kernel to user-space) Scheduling for gaming and latency-sensitive workloads

Scheduling & cpufreq integration Distro support While we anticipate having a schedule with existing talk proposals at the MC, we invite you to submit proposals for any topic(s) you'd like to discuss. Time permitting, we are happy to readjust the schedule for additional topics that are of relevance to the sched\_ext community. Submissions are made via LPC submission system, selecting the track Sched-Ext: The BPF extensible scheduler class. We will consider the submissions until July 12th.

From:

<https://wiki.tromjaro.alexio.tf/> - **TROMjaro wiki**

Permanent link:

<https://wiki.tromjaro.alexio.tf/doku.php?id=news:planet:kernel&rev=1583616224>

Last update: **2021/10/30 11:38**

