# Kernel Planet - Latest News

- Matthew Garrett: Getting Google Cast on my Soundbar (2026/10/30 18:03)
  I have a Samsung Q800A soundbar. It ships with built-in support for the Apple Airplay protocol, but I'm not really an Apple kind of guy so that's never been particularly helpful for me. So, instead, I have a Chromecast Audio plugged into it via TOSLINK, and that works fine - except for the fact that there's no way to explicitly pick TOSLINK as the audio input. Instead, there's a "Digital In" option that refers to both the optical in and eARC input from a TV. Not inherently a problem, given I'm unlikely to be watching TV while listening to music, but my TV is a Samsung Frame, a device that's intended to display art whenever someone is standing nearby, and which infuriatingly generates an eARC signal even if it's only in art mode and isn't going to make any noise. This switches the soundbar away from optical and I get sad. I could plug a video cast device into the HDMI port on the soundbar and cast audio to that, but then the art would get replaced with album covers whenever I'm playing music, and again that's not quite the ambience I want. Still, none of this is that important. Until I learned that the Australian version of the soundbar does have Google Cast support. The firmware image was downloadable from Samsung support, so I grabbed that, but unfortunately found that it was encrypted. However, if I downloaded the firmware image for the US model, the encrypted blob was identical - there's no difference in the underlying firmware between the two of them. So, how come one has support, and the other doesn't? I'd need an unencrypted copy of the firmware to have a real chance of figuring this out, and the easiest way to get one was to just run an ebay watch for a broken model of the same series. Once I had one in my hands I used hot air to remove the eMMC storage from the device, cleaned up the chip, and then stuck it in one of these adapters to read it on a local system. Samsung uses a weird in-house filesystem called VDFS4 on some of their devices and this was one of them, so I needed to cobble together a kernel that was compatible with their source dump for this device (not all VDFS drivers seem to be able to read all VDFS filesystems), but eventually I had it all extracted, and then I got distracted by other stuff, so let's fast forward to this week. Samsung makes use of a modified version of the Android Debug Bridge called sdb, which is included in the Tizen SDK. The firmware showed that this device had sdbd on it, and it would be executed if a network device called "huart0" was plugged in. Samsung does, thankfully, provide the source code for their GPLed components, so I could dig through that to discover that drivers/net/usb/usbnet.c had a bunch of references to huart (which apparently stands for "HyperUART"). If CONFIG_SAMSUNG_USBNET_HYPERUART_IFNAME is set, and if the ethernet device is using either the asix or ax88179_178a drivers, and if "/opt/etc/HyperUART_IP_Addr"1 isn't just "0", it sets the interface name to "huart0" and in turn triggers sdbd startup. So now I just needed to figure out how to set /opt/etc/HyperUART_IP_Addr to something else. Some digging revealed a binary called wap-factory that referenced the file, which then led to /usr/share/mobile-mgr/jsons/com.samsung.audio.waapp-verification-test.json which mentioned a "SetUartOnOff" call, which brought me here2, which led me into a bit of a rabbit hole about the Samsung WAM protocol. Long story short, you connect to either port 55001 or 56001 and make two requests over the same connection: 1 2 3 4 5 6 7 8 9 GET /connect HTTP/1.1 Host: 192.168.87.126:56001 User-Agent: SamsungMultiroom/1.0 Connection: interactive GET /UIC?cmd=%3Cname%3ESetUartOnOff%3C/name%3E%3Cp%20type%3D%22str%22%20name%3D%22uartOnOff%22%20val%3D%22on%22/%3E HTTP/1.1 Host: 192.168.87.126:56001 User-Agent: SamsungMultiroom/1.0 Connection: interactive and then plug in an asix chipset ethernet adapter into the USB port (which on mine is micro-B, so I needed an OTG adapter), and suddenly it's sitting there on 192.168.250.250 and will

speak sdb to you. sdb root on switches it into root mode, and sdb shell now gives you a root shell. I was in, and now I could investigate what was happening. First up, there was a process called “com.samsung.audio.google-cast-control-app” running, which seemed like a good start. Unfortunately it wasn’t doing anything. After some tracing I found that it was making a call to Samsung at startup time, and getting back a response telling it that Cast support wasn’t available. The request (which was helpfully dumped to stdout when I ran this by hand) included the country code, in my case “US”. Repeating the request by hand with curl and changing the code to “AU” told me it was supported, so now I needed to figure out how to get the soundbar to believe it was Australian. Some digging showed it was reading this from a config variable called db/comss/countrycode, and running 1 buxton2ctl set-string system db/comss/countrycode AU got me past that gate. Now, however, it was complaining that it couldn’t provision a key. /usr/apps/com.samsung.audio.google-cast-control-app/shared/res included a bunch of encrypted blobs and an app called teeStoreKey.elf, and tracing com.samsung.audio.google-cast-control-app showed it was trying to call that, but it seemed like it was failing with a permissions issue. Running com.samsung.audio.google-cast-control-ap directly as root triggered key provisioning successfully, and a few seconds later the rest of the Cast API came up and I could talk to port 8443 with standard Cast API calls. Unfortunately it still wasn’t working as a Cast device, though - it wasn’t advertising itself via mDNS and so nothing could see it. Querying the /setup/eureka_info endpoint gave me a clue - the ’tos_accepted’ field was false. Normally this would be set as part of initial device setup, but the Samsung app wasn’t doing that because it didn’t think the device had Cast support. It turns out that POSTing some json with {“tos_accepted”: true} to setup/set_eureka_info made it work, and suddenly there was an additional Cast device available to me. (Update: I realised I didn’t mention anything about why it was doing key provisioning in the first place! The Cast protocol doesn’t enforce cryptographic identity as such, but Google’s reference client implementation which is what almost everything uses verifies that the device is presenting an identity that chains back to Google. Unsurprisingly they don’t want the private key material to be easily extractable and used to produce clone devices, so it’s all supposed to be encrypted and only accessible to some sort of trusted execution environment, which is why keys here need to be loaded into Trustzone and never be visible in the clear to the main OS) Mission accomplished. Unfortunately the device vanished from my Smartthings home aferwards, presumably because there’s a mismatch between its reported country code and my account country. To get around that I’d somehow need to get com.samsung.audio.google-cast-control-app to either change the query without changing the global device country, or to ignore the response from Samsung (it does this every 24 hours, ugh), and that’s made more annoying by the fact that this thing will only execute signed binaries. Although, weirdly, one of the signed binaries shipped is a full Python interpreter, so if I cared enough I could probably cobble together enough Python to execute arbitrary binaries. I currently don’t. More modern soundbars don’t implement the listener on port 55001 or 56001. I don’t know how you get sdb working on them. Yes, it reads a file directly from in-kernel ↩ Good god I hate reading this kind of slop ↩

- Linux Plumbers Conference: The sched_ext Microconference (2026/10/28 14:57)
  The sched_ext Microconference was very active this year at Linux Plumbers in Prague. It featured six talks, preceded by an opening discussion on the future direction of sched_ext. (Written by Andrea Righi) Three main topics dominated the discussions Priority inversion and how to handle it in sched_ext: lock holder and lock waiter preemption, time slice extension, and making proxy execution work together with BPF schedulers. Paravirtualized scheduling: letting the host and guest schedulers cooperate across the VM boundary, with sched_ext as the policy engine on the host. Device and topology-aware scheduling: using hardware feedback (Intel HFI), device/accelerator locality and cache topology to drive task placement. Intro: where sched_ext is going The opening discussion outlined three directions for sched_ext development: Move more into BPF. With BPF arenas, queuing and idle CPU management can be implemented entirely in BPF, and several schedulers are already converging on this

model. Once they all move, a significant amount of in-kernel sched_ext code can be removed. Composability. Long discussed, now a good time to tackle it: reusable BPF/user-space building blocks (load balancer from one scheduler, idle CPU selection from another, etc.) that can be combined into new schedulers. Generate BPF code from Rust. Most schedulers today are Rust in user space plus C for the BPF part. The trend, aligned with the wider BPF community, is to generate the BPF code from Rust as well, so a single language and shared abstractions can be used to access scheduling data structures from both sides. Prototypes exist but this is not there yet. Priority inversion Making proxy execution compatible with sched_ext (Andrea Righi and John Stultz) Proxy execution lets a mutex-blocked task lend its scheduling context to the lock owner. The work presented makes this compatible with sched_ext, with support enabled explicitly by each BPF scheduler. Dispatching a blocked task triggers the core proxy machinery, keeping the BPF scheduler in control. The compatibility changes and follow-up improvements are queued for Linux v7.4. Distributions will be able to enable both `CONFIG_SCHED_PROXY_EXEC` and `CONFIG_SCHED_CLASS_EXT` in the same kernel, removing their previous mutual exclusion. The follow-up changes will also make proxy migrations transparent to BPF, without exposing intermediate bookkeeping through extra enqueue callbacks. Discussion points: User-space locks: futexes are not covered since `FUTEX_WAIT` does not identify the owner. A new futex op that carries ownership is being explored; passing user-space owner hints (e.g., via BPF maps or futex flags) is inherently racy. A “directed yield” (timed proxy to a specific task) was mentioned as future work. rwsems / semaphores: writer boosting is supported via a patch; boosting multiple readers is harder and has overhead. Plain semaphores have no owner and cannot be proxied. Observability: there are no upstream tracepoints for proxy execution yet, which makes traces confusing (the owner runs under the donor’s context). Something similar to the existing PI tracepoints was requested. Taming latency spikes caused by lock holder and lock waiter preemption (Changwoo Min) On the Steam Frame VR headset, camera processing must finish within 33ms but needs only about 7–8ms of CPU time. Under heavy gaming load, preempting a lock holder or delaying a woken lock waiter can stretch the pipeline enough to miss that deadline. LAVD addresses both cases with time slice extension and boosting of lock waiters. The mitigations reduced scheduling delays above 2ms across four games, with no missed camera frames observed during an hour of gameplay in the tested titles. A common interface for lock ownership and waiting information, covering kernel and user-space locks, would help replace scheduler-specific heuristics. Discussion points: Abuse of the extension is bounded by fairness: a task that constantly extends its slice is deprioritized. Why not just rely on proxy execution? Proxy is reactive (owner scheduled out, donor scheduled in, owner scheduled in again), with extra overhead. The consensus was that the two approaches are complementary: slice extension is like a spinlock (cheap, good for short critical sections), proxy execution is like a mutex (heavier, but handles long ones). Paravirtualized scheduling Bridging the VM Boundary: Scheduling Passthrough via pvsched and sched_ext (Josh Don and Vineeth Remanan Pillai) A host scheduler normally sees opaque vCPU threads, without knowing whether they are running urgent guest work or background jobs. pvsched provides a bidirectional channel: guests communicate priorities and preemption hints, while hosts report which vCPUs they can run concurrently. The proposed v3 design moves negotiation into the VMM through a virtual PCI device, keeping policy outside KVM. On the host, sched_ext can implement bounded vCPU boosting and gang scheduling. Guests can concentrate work on the vCPUs the host can actually run, reducing the cost of double scheduling. The next revision is still to be posted upstream. Discussion points: BPF arenas as an alternative to a dedicated device: a host scheduler could allocate an arena, hand out per-guest chunks and get zero-copy communication with no kernel changes, needing only a way to pin arena memory. Someone from the audience pointed out that allocating memory on the host and mapping it into the guest on the fly is slow (memslot creation, guest notification), which is why guest-initiated allocation was chosen. However, this cost can be avoided: the host BPF scheduler can create the arena once when it is started and hand out per-guest chunks from the

preallocated arena, mapping each chunk when the guest is set up rather than on the fly, which keeps the fast path cheap. It was also noted that the device model keeps the handshake arch-agnostic and evolvable without kernel changes. There is related work from others in this area; the agreement was that the various efforts should converge on a single kernel module, possibly alongside a BPF-arena-based path. Standardization: post the kernel module first, then the QEMU device, then other VMMs. The preferred-CPU framework (steal governor) is related, but reactive, while pvsched allows an explicit policy. Device and topology-aware scheduling Automatic placement of accelerator workloads (Balbir Singh and Lee Trager) CPU and memory placement can have a large effect on accelerator performance, but manual pinning requires applications or administrators to know the topology and device usage in advance. The proposed alternative combines hints about which device a task is using, sched_ext task placement, and memory-access telemetry to guide page placement. PMU counters were also suggested as a source of remote-access and bandwidth telemetry to inform those decisions. Two interfaces are needed: a generic process-to-device association and BPF control over NUMA placement preferences. A series enabling NUMA balancing under sched_ext had recently been posted, together with a kfunc for requesting a preferred memory node. Discussion points: Exposing PMU telemetry to non-root applications raises side-channel concerns; IPC is fairly safe, cache/TLB misses are a gray area. An allowlist of audited events was suggested. Is a BPF map the right interface, or is a common user-space library needed so all frameworks can provide hints? Migrating network-heavy tasks across nodes may interact badly with per-CPU queues and TCP flows. NUMA balancing needs improvements, including asynchronous migration and charging migration work to the right task. Enabling it under sched_ext reuses the existing mechanism, so that work can proceed in parallel with those fixes. Telemetry-driven `move_pages()` can serve as an interim solution. HFI plug-in for sched_ext (Srinivas Pandruvada) Intel’s Hardware Feedback Interface supplies per-CPU performance and efficiency hints that reflect the platform’s changing power budget. Integrating these hints into LAVD improved Shadow of the Tomb Raider performance by about three frames per second on a hybrid Intel handheld at a 10W power limit. The prototype took less than a month. A Rust user-space thread receives HFI updates over netlink and passes them to BPF through a map; LAVD then adjusts CPU selection and effective capacity. Discussion points: Why is HFI not used by the in-kernel scheduler? Older generations were too noisy, and the firmware was tuned mostly on Windows workloads; the last two generations treat Linux on equal footing, so it is worth revisiting. Qualcomm’s WALT has a similar CPU isolation feature (thermal and low-utilization cases), implemented downstream via Android vendor hooks. The preferred-CPU framework (steal governor) could be a natural in-kernel hook for HFI exclusion. The thermal pressure signal filter currently only allows slowing down the reaction time, and patches to make it configurable in both directions may be revisited given how fast firmware can now report changes. The HFI code is structured as a library and was proposed for inclusion in the scx library so other schedulers can reuse it. Stickiness: keeping tasks cache-warm in the work-conserving LAVD scheduler (Gavin Guo) Keeping a task on a cache-warm CPU can improve performance, but waiting for that CPU competes with the benefit of running immediately elsewhere. The session compared cache-warmth modelling, CPU reservation, and look-ahead dispatch from a shared queue. Results depended on the workload and approach. Warmth-based placement improved P99 latency for short waits, while look-ahead dispatch showed no noticeable memcached latency improvement under light or moderate load and added overhead under heavier load. Discussion points: Staying on a busy CPU is a short-term gain, while an idle CPU may offer long-term uninterrupted execution, so the right answer is very workload- and topology-dependent (shared vs non-shared L2/L3, snoop efficiency). Suggestions included using PMU counters (as scx_cosmos already does) or Intel RDT/CMT for cache occupancy (limited by the number of monitoring IDs), tracking only latency-critical tasks, and broader workloads such as stress-ng and video encoders.

- Linux Plumbers Conference: LPC 2026 – Kernel Testing MC (2026/10/20 19:06)

The Kernel Testing & Dependability Micro-Conference was back again at Plumbers this year, in Prague and with a wide variety of topics. In addition to the well-established frameworks such as KUnit and kselftest, some new frontiers are being explored with the spotlight on LLM-assisted tools. Developers are also getting special treats with new kci-dev features and expert insights into local build testing at scale. Primary author: Guillaume Tucker kci-dev What Changed, What Works, and What Kernel Developers Still Need – by Arisu Tachibana (virtual) The kci-dev command line tool enables developers to interact with KernelCI. Here's a summary of the latest changes: Core implementation moved to the kcidev Python module as an API library Standalone patch file testing enabled on top of a Git checkout Submission of results to KCIDB enabled with artifacts uploaded to storage Experimental MCP added on top of the API The tool also benefits from these new plugins: kci-patchwork to apply patch series from Patchwork and submit test job kci_release_review to compare revisions and create reports Plugins can help extend functionality so they should be a good way of driving future kci-dev development. The MCP may also enable new use cases. Please join the KernelCI mailing list and Discord channel as per the contact page if you wish to take part and discuss things further. Build Identity Kernel Builds Deserve an Identity: Portable Kernel Artifacts and Traceable Test Results – by Cong Wang The raw metadata provided by uname, Git, IDs from various CI systems etc. does not provide any standard or consistent way to identify a build. As such, bug reports can't directly refer to a single reproducible binary. Cong Wang's proposal is the Kernel Bundle Image (KBI) project. It draws a parallel with how OCI solved this for container images. The idea is essentially to create OCI images for kernel builds. However, it still has a low level of adoption and isn't used by current automated testing systems. How may KBI gain more traction? Where should the line be drawn about what to include in the kernel build ID? There hasn't been a clear consensus on this yet, whether it should include device trees, firmware blobs and other artifacts. Yocto / OE also has a way to reproduce entire OS builds, including the kernel. This may be used as a guideline or potentially combined with the KBI approach. It's also worth noting that TuxMake is able to reproduce binary-identical builds for a given set of inputs (container image and build parameters), which does not constitute a build ID per se but could still be potentially very helpful for this purpose. A related initiative is KernelSbom since a bill of materials could define the input to a KBI image: source with metadata, main build artifacts, additional contents (firmware, BPF). KUnit Configuration Rethinking KUnit Configuration – by David Gow The session started with a poll: roughly half of the room use KUnit regularly and 3 people answered yes to the last question about running KUnit with multiple configurations. This tends to show that the tool is well established for standard usage but few developers use it in an expert way. The main issue covered by this session is about limitations with the .kunitconfig file and qemu_config Python module, especially the lack of output format options and the fact that each config covers only a single KUnit run. Proposal: Determine a single standard configuration mechanism to go beyond limitations while being easy to share between developers. Passing metadata through to the results would also be especially useful for CI systems which can deal with more details and trace the results back to their origin. Should this be solved with a new config system or by extending an existing one? Option 1: extend .kunitconfig (currently a plain Kconfig fragment) Option 2: native Python module since kunit.py is already in Python Any other options? David's preferred option is to extend .kunitconfig. For multiple runs bundled together, it's important to be able to parse the results as if they were individual ones e.g. objects embedded within a bigger blob. One potential solution is to produce results in separate files (KTAP). Multiple runs are useful for CI integration with a single entry point. It also removes the overhead of setting up many short-lived individual jobs. This doesn't however appear to be too useful for manual runs. It can help a bit with git bisect run although multiple KUnit commands may be bundled within a single script. It's also worth noting that having isolated kernel runs would prevent a single crash from breaking the whole series. The overall consensus is that multiple runs in a single command or config aren't probably that useful except in particular cases e.g. CI hooks. Build Testing Optimizing local kernel build

testing – by Arnd Bergmann As a sequel from Arnd's talk at Kernel Recipes, this session is to discuss using a manual, local setup to achieve wide build testing coverage. The focus is on linux-next, with -Werror enabled to try and catch all new build issues. The best approach when something breaks is to send a fix if possible as it is a lot more effective than just reporting bugs. By doing this for many years, Arnd has fixed thousands of bugs upstream (3000+) and touched every subsystem in the kernel. The most common problems are Kconfig dependency bugs since its logic is complicated and deceptive, especially with tristate options. The new kconfirm tool aims at solving this issue via static analysis. Arnd keeps a small config fragment to maximise the effectiveness of build testing: # maximize search space, disable options not worth testing CONFIG_COMPILE_TEST=y # avoid long build times # PROFILE_ALL_BRANCHES is not set # reduce compile-time dependencies CONFIG_STANDALONE=y CONFIG_PREVENT_FIRMWARE_BUILD=y Should orchestration of parallel builds be integrated upstream? This sounds useful although the discussion didn't expand on how to approach this topic in practice. The general makefile and arm64-specific testrandconfig64 files attached have been kindly provided by Arnd as a reference based on his local setup. Please feel free to reach out to him via email should you wish to discuss this further. Should local build regression reports be sent to KCIDB? With a proposed fix in a patch? Several people would welcome this and a follow-up discussion is intended to take place with the KernelCI team. kcov and kselftest Kselftest coverage with kcov – by Shuah Khan Thousands of tests are available in-tree via kselftest and KUnit, but what is their coverage? Shuah has made an experimental tool to run kselftest with kcov and start looking into the issue. Coverage data is not ready to be published yet so the discussion was more about the potential use cases. You can already expect to see a more in-depth follow-up analysis on this topic next year and other general kselftest sessions at every future LPC. How is code coverage useful? When working with unknown customer workloads, comparing their coverage with well-known tests gives an idea of how much the tests are covering these workloads. One issue with enabling kcov at build time is that it's not suitable for production kernels. However, ftrace is usually enabled so it could be used in combination with kcov with different granularity (function-level only coverage with ftrace). Another dimension to the problem is about evaluating other tools e.g. LLVM/Clang coverage. See also the LLM fuzzing session on a related topic. KFuzzTest KFuzzTest: One year on – by Ethan Graham The aim of KFuzzTest is to combine the benefits of fuzzing and unit testing. A new, experimental API is now available in the latest version of the patch series to make it easier to write tests and facilitate adoption. The next core feature which is currently under development is stateful fuzzing. Initial trials found a known bug which had already been fixed but this showed the tool's potential. Can fuzzing target specific code? This is not trivial. In some cases, hardware mocks may be used to enable coverage on specific drivers. Generally speaking, KFuzzTest's aim is to narrow down fuzzing to isolated functions like a unit test would typically do. The effectiveness of this method is still being debated, perhaps the sweet spot lies in between this and syzkaller's broader approach. How about stateful fuzzing? Stateful fuzzing in userspace exists and enables setting extra boundaries. Kernel fuzzing is typically run from outside the kernel, kind of guessing how to invoke the syscalls. Having the kernel asking a userspace fuzzer has led to successful experiments in the past. So overall, stateful fuzzing is feasible. How about KUnit overlap? David Gow is in favour of adding functionality that lets userspace pass data into KUnit tests. This could also facilitate making reproducers. However, the stateful approach is not a good fit for KUnit with long strings of operations as unit tests are meant to be run in isolation. LLM Fuzzing LLM-Assisted Fuzzing with Syzkaller – by Chuyang Wang & Aleksandr Nogikh The issue when fuzzing with high-level descriptions is that they don't go deep enough to reach good code coverage. Hand-written seed programs partially solve this but the approach doesn't scale. The proposed solution here is to rely on LLM-generated ones instead. While attempting to reach areas never covered by syzbot, an overall 15% increased coverage was reached over a few months using this approach. This uncovered a new issue with the LLM trying to evade syzkaller's sandboxing environment. The next step is going to integrate with syzbot. Is AI-driven cheaper?

Probably but not measured in terms of actual costs. One seed program could then expand coverage. How about fuzzing drivers? This is tricky because testing drivers often requires access to real hardware. USB is an exception as physical devices are not strictly necessary. This has enabled the team to significantly increase fuzzing coverage and report many bugs over a short period of time. Still useful to upstream hand-written seed programs? Yes, these are complementary approaches. KernelCI in the Cloud Testing the kernel where the fleet runs: KernelCI in the cloud – by Norbert Manthey & Stanislav Uschakow Going beyond the historical use cases of KernelCI which originated on embedded platforms, this session is about testing kernels deployed in the Cloud. The system covers x86_64 and Graviton (arm64) VMs in an AWS lab. Additionally, nested KVM guests on particular VM types are also supported. It provides the typical scalability and variety of platforms one would typically expect to find in this environment. Developers may also run the tests themselves using their own AWS account. Current upstream kernel coverage includes mainline, linux-next and LTS. Runtime coverage is limited to baseline boot testing and basic checks. Performance testing is next on the roadmap. How is A/B testing done in the AWS Cloud? The approach is to run both A and B kernels with the same VM definition to reduce dependencies on the environment (hardware of physical host, virtualization stack, firmware, …). Keeping the kernel as the single moving part enforces it to be the only root cause for any difference found in test results. The code for this A/B testing is made available as a GitHub repository under the KernelCI project umbrella. How does A/B testing on AWS work with live migrations? Tests are not relying on a file system for now to keep resource dependencies low. Hypervisor updates can cause one VM to stop but tests are run multiple times and in a distributed way so the impact is mitigated. Are you running KUnit? KernelCI is running KUnit in other environments, not in AWS. It’s unclear whether doing this would increase existing coverage. Which kernel trees have the priority? Testing linux-next and mainline kernels should be the priority. Covering LTS is also useful to verify that fixes get backported successfully as they would typically be applied to the AWS forks too although those are outside the scope of KernelCI. Are any more details of the test runs available? In order to give maintainers access to more details about test runs, the AWS lab setup needs to expand its integration with KernelCI in general. Additional features have been requested during LPC such as being able to automatically bisect issues and sending regressions as direct replies to patches via emails threads. Those reports should also be sent to the kernel regressions list. Testing BoF Note – The speaker has apologised for not being able to host the last session of the schedule due to personal reasons. It was replaced with an impromptu Testing BoF instead. One particular point was discussed: How to address kselftest failing to build and run? Tests may fail to build due to missing dependencies. The kselftest_deps.sh tool can be used to discover those. Integrating it into the kselftest Makefile is being considered in order to report them directly to the user. Test build failures don’t cause the overall kselftest build to fail. This is good, although reporting this in a better way might be helpful. Build and run regressions need better reporting, they could be made more visible using a tag in email subjects. Wrapping Up One lesson learnt is that AI-assisted development tends to be more about data analysis while the initial scope of the Testing MC is rather to focus on low-level tools. These may be seen as optimised, primitive functions producing raw data that LLMs can then leverage and potentially use to orchestrate test runs. Having a session about LLM-assisted fuzzing was a good fit though as it showcased how AI can make use of existing testing components in novel ways. Last but not least, many thanks to all the speakers for reviewing this blog post, the conference committee and all the attendees for making Plumbers such a great experience. It was a pleasure to be a co-organiser of the Testing MC again this year. See you at LPC 2027

- Linux Plumbers Conference: LPC 2026 Video recordings are available (2026/10/19 15:39)
  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

- Dave Airlie (blogspot): jetlag diversions - llvmpipe 512-wide vector support (2026/10/16 09:32)
  I was travelling last week, and got back to Mike rebasing the zink-video branch and using LLM powerups to fix the final bits of it. We had been locked in a death stare for a few years, and he flinched first. But it got me thinking about another stalled branch I started working on ages ago.Adding support for avx512 to llvmpipe. The basics have been there, but running LP_NATIVE_VECTOR_WIDTH=512 piglit exploded in a bunch of fragment shading paths, mostly blending and depth handling. This should work even without actual avx512, we just make LLVM generate worse code. I'd gotten it most of the way and then fell over some 1d texture and RGB32 type failures and let it slide when more urgent work came up.Well this week I had tiredness, and inspired by Mike I decided to dust off my old branch and let Opencode + GPT Terra Pro 5.6 have a go at closing the final gaps. It started finding a bunch of problem, and I chewed through a lot of my monthly quota, and it was generating a lot of good work, but costing a lot of tokens and taking a long time. It closed the gap on a bunch of the 512 wide work over a day or two.I then decided to switch to Astra, and it was so much better/quicker/cheaper for this job. It started to find other problems in llvmpipe that I hadn't seen, it found some bugs in the 128-wide handling for 1D framebuffer fetch while looking at the code for 512-wide framebuffer fetch. It identified the current test gap and wrote a test filling it. It found a 128/256 difference on a misc piglit fog test and it identified the conversion path that was failing. I had it review the patch series I wrote and Terra had fixed, and it found a bunch of cleanups and cases where the 4/8/16 lane code be made common.I fixed a niche bug in geometry shader indirect loads (without the LLM) and an egl fd = -1 problem valgrind popped up.All the niche fixes are pushed, and the 512-wide support is in an MR, it's still all my code, I've just cleaned it up a lot.I also don't have any avx512 hardware, so I can't answer is it faster until I either rent some AWS time or find a machine in RH, but if anyone else wants to test gears at least it should be possible now.
- Matthew Garrett: Epic's EAC2 v2 seems fairly easily bypassable (2026/10/15 20:23)
  Epic just announced the upcoming version 2 of their Linux implementation of Easy Anti-Cheat. It’s based on using the Linux eBPF security module to monitor what’s happening, and then a bunch of extra infrastructure to make sure this can’t be bypassed. The core part of this is requiring secure boot be enabled (ensuring that modified kernels can’t be used), using measured boot to verify that secure boot is enabled and which keys are used (to prevent you from just signing your own modified kernel), and then doing remote attestation, a mechanism where the TPM generates a signed copy of the boot measurements and passes that off to a server which verifies it came from a real TPM, verifies that you have secure boot enabled and are using the right keys, and presumably then allows you into the game. This is all fine as long as you can prove that the measurements provided are real, and that’s actually a lot harder than you might think. First, we can simply make up our own quote signed with our own key and say that it came from a TPM. This is avoided by every TPM having an Endorsement Key (or EK) and an associated certificate, with that certificate being signed by the TPM vendor. If you have a list of all the legitimate TPM vendors and their CA keys, you can verify that the measurements came from a real TPM. Easy enough. So we need to get the measurements from a real TPM, and the one on our motherboard contains the measurements that were generated by the firmware and bootloader, and we can’t reset those at runtime without resetting the TPM, which… oh huh there’s actually nothing stopping us from doing that at runtime, as long as we have access to the hardware. Many desktop boards have the TPM on a small plugin module with a standard pinout that includes a reset pin. Simply short that to ground briefly and the TPM is back in its initial state, and userland can write whatever measurements it wants to it. Boot a modified kernel which lies to the eBPF code? As long as you replay the expected log values from a boot with secure boot on and a trusted kernel, the TPM will still happily provide a legitimate set of measurements that’ll satisfy the checker. What if the TPM is less accessible, either because it’s mounted directly to the motherboard or it’s inside

the CPU or chipset? That's going to be trickier. But here's the thing. The quote needs to come from a TPM, but not necessarily the TPM on your motherboard. TPMs these days are usually connected via the Serial Peripheral Interface, usually referred to as SPI. Microchip make a USB to SPI bridge, and there's a Linux driver. Wire one of those up with an appropriate header and you can plug in an easily available TPM module intended for plugging into a motherboard, and now we tell the kernel to ignore the motherboard TPM and make /dev/tpmrm0 point at the USB one and program it with a set of trusted measurements and once again everything is going to look just fine (yes obviously the eBPF code can look for this, but we're running a modified kernel so we can just lie to the eBPF code). Remote attestation on PCs just doesn't work that well in environments where you can't associate a specific TPM with a specific system, which is basically everywhere except enterprises. But maybe that doesn't matter and cheaters wouldn't be willing to spend the $20 or so it'd take to produce one of these things? That would seem somewhat surprising, but it's not impossible. And obviously any server-side cheat detection that triggers could then block that specific TPM, meaning they'd need to go and get another TPM, and maybe that ends up being enough to discourage people from bothering. From a practical level, this may be sufficient. But at a technical level? It's not great.

- Matthew Garrett: Linux Lockdown and Hibernation (again) (2026/10/13 18:14)
  (This is an updated version of an older post - the introduction remains the same, the technical details are different) Linux draws a distinction between code running in kernel (kernel space) and applications running in userland (user space). This is enforced at the hardware level - in x86-speak1, kernel space code runs in ring 0 and user space code runs in ring 32. If you're running in ring 3 and you attempt to touch memory that's only accessible in ring 0, the hardware will raise a fault. No matter how privileged your ring 3 code, you don't get to touch ring 0. Kind of. In theory. Traditionally this wasn't well enforced. At the most basic level, since root can load kernel modules, you could just build a kernel module that performed any kernel modifications you wanted and then have root load it. Technically user space code wasn't modifying kernel space code, but the difference was pretty semantic rather than useful. But it got worse - root could also map memory ranges belonging to PCI devices3, and if the device could perform DMA you could just ask the device to overwrite bits of the kernel4. Or root could modify special CPU registers ("Model Specific Registers", or MSRs) that alter CPU behaviour via the /dev/msr interface, and compromise the kernel boundary that way. It turns out that there were a number of ways root was effectively equivalent to ring 0, and the boundary was more about reliability (ie, a process running as root that ends up misbehaving should still only be able to crash itself rather than taking down the kernel with it) than security. After all, if you were root you could just replace the on-disk kernel with a backdoored one and reboot. Going deeper, you could replace the bootloader with one that automatically injected backdoors into a legitimate kernel image. We didn't have any way to prevent this sort of thing, so attempting to harden the root/kernel boundary wasn't especially interesting. In 2012 Microsoft started requiring vendors ship systems with UEFI Secure Boot, a firmware feature that allowed5 systems to refuse to boot anything without an appropriate signature. This not only enabled the creation of a system that drew a strong boundary between root and kernel, it arguably required one - what's the point of restricting what the firmware will stick in ring 0 if root can just throw more code in there afterwards? What ended up as the Lockdown Linux Security Module provides the tooling for this, blocking userspace interfaces that can be used to modify the kernel and enforcing that any modules have a trusted signature. But that comes at something of a cost. Most of the features that Lockdown blocks are fairly niche, so the direct impact of having it enabled is small. Except that it also blocks hibernation6, and it turns out some people were using that. The obvious question is "what does hibernation have to do with keeping root out of kernel space", and the answer is a little convoluted and is tied into how Linux implements hibernation. Basically, Linux saves system state into the swap partition and modifies the header to indicate that there's a hibernation image there instead of swap. On the

next boot, the kernel sees the header indicating that it's a hibernation image, copies the contents of the swap partition back into RAM, and then jumps back into the old kernel code. What ensures that the hibernation image was actually written out by the kernel? Absolutely nothing, which means a motivated attacker with root access could turn off swap, write a hibernation image to the swap partition themselves, and then reboot. The kernel would happily resume into the attacker's image, giving the attacker control over what gets copied back into kernel space. This is annoying, because normally when we think about attacks on swap we mitigate it by requiring an encrypted swap partition. But in this case, our attacker is root, and so already has access to the plaintext version of the swap partition. Disk encryption doesn't save us here. We need some way to verify that the hibernation image was written out by the kernel, not by root. And thankfully we have some tools for that. Trusted Platform Modules (TPMs) are cryptographic coprocessors7 capable of doing things like generating encryption keys and then encrypting things with them. You can ask a TPM to encrypt something with a key that's tied to that specific TPM - the OS has no access to the decryption key, and nor does any other TPM. So we can have the kernel generate an encryption key, encrypt part of the hibernation image with it, and then have the TPM encrypt it. We store the encrypted copy of the key in the hibernation image as well. On resume, the kernel reads the encrypted copy of the key, passes it to the TPM, gets the decrypted copy back and is able to verify the hibernation image. That's great! Except root can do exactly the same thing. This tells us the hibernation image was generated on this machine, but doesn't tell us that it was done by the kernel. We need some way to be able to differentiate between keys that were generated in kernel and ones that were generated in userland. TPMs have the concept of "localities" (effectively privilege levels) that would be perfect for this. Userland is only able to access locality 0, so the kernel could simply use locality 1 to encrypt the key. Unfortunately, localities are typically only available on systems that implement a Dynamic Root of Trust Measurement like Intel's TXT, and most consumer systems have no support for this. So, we need another approach. TPMs have a set of Platform Configuration Registers (PCRs), intended for keeping a record of system state. The OS isn't able to modify the PCRs directly. Instead, the OS provides a cryptographic hash of some material to the TPM. The TPM takes the existing PCR value, appends the new hash to that, and then stores the hash of the combination in the PCR - a process called "extension". This means that the new value of the TPM depends not only on the value of the new data, it depends on the previous value of the PCR - and, in turn, that previous value depended on its previous value, and so on. The only way to get to a specific PCR value is to either (a) break the hash algorithm, or (b) perform exactly the same sequence of writes. On system reset the PCRs go back to a known value, and the entire process starts again. Some PCRs are different. PCR 23, for example, can be reset back to its original value without resetting the system. We can make use of that. The first thing we need to do is to prevent userland from being able to reset or extend PCR 23 itself. All TPM accesses go through the kernel, so this is a simple matter of parsing the write before it's sent to the TPM and returning an error if it's a sensitive command that would touch PCR 23. We now know that any change in PCR 23's state will be restricted to the kernel. When we encrypt material with the TPM, we can ask it to record the PCR state. This is given back to us as metadata accompanying the encrypted secret. Along with the metadata is an additional signature created by the TPM, which can be used to prove that the metadata is both legitimate and associated with this specific encrypted data. In our case, that means we know what the value of PCR 23 was when we encrypted the key. That means that if we simply extend PCR 23 with a known value in-kernel before encrypting our key, we can look at the value of PCR 23 in the metadata. If it matches, the key was encrypted by the kernel - userland can create its own key, but it has no way to extend PCR 23 to the appropriate value first. We now know that the key was generated by the kernel. Unfortunately it turns out that some people actually make use of PCR 23, and restricting it to the kernel makes them sad. There are other resettable PCRs, but only in non-0 localities, so (a) we can't reset them on most systems and (b) we'll just find someone who uses them anyway and run into the same problem. We need a different

approach. Thankfully TPMs have a small chunk of non-volatile RAM that can be used for storing things, and by “storing” I mean they can be used in a range of different ways. One of these is effectively as additional PCRs. These values can be configured such that they work the same way as PCRs, with the only way to modify them being to append a new hash to their existing value and update the new value to the hash of that merged value. We can allocate one of these for in-kernel use and reject any access from userspace, and also there are officially allocated ranges (so people shouldn’t be using random values) and the namespace is large enough anyway that w eshouldn’t collide with anyone, so we can carve one of those out and use it for this purpose. Unfortunately, though, these NV indexes aren’t included in the creation data we get when we create a key, so we need another approach. TPMs have the concept of sessions, a series of commands executed in sequence. A flag allows these sessions to be “audited”, with the TPM keeping a running track of the hash of the commands executed and the results returned. At the end of the session you can ask for a signed copy of this hash, and if you’ve kept a log of all the commands then you can verify that they match that signed hash and know that this log represents a real communication with the TPM. If we read the NV index and generate the key inside the same session, we can verify that the key we’re looking at was created while the NV index had a specific value, and if only the kernel can set that value then we know that the key was generated by the kernel. Or, well, can we? Only new kernels would impose the restriction on modifying that NV index. An attacker could boot an old kernel, generate a key there in userland, sign a hibernation image with it, and then reboot into a new kernel. The new kernel would see a valid signature with a valid NV index associated with it and have no way to know it was generated from userland rather than the kernel. We need to be able to tell the difference between kernels that implement this feature and ones that don’t, and we need to do that without being able to go back in time to change old kernel behaviour. We can extend a PCR to indicate we implement the new functionality, but so can an attacker running an old kernel. The normal approach to determining whether an event occurred in a specific phase of the boot process is to “cap” the PCRs - extend them with a known value that indicates a transition between stages of the boot process. Any events that occur before the cap event must have occurred in the previous stage of boot, and since the final PCR value depends on the order of measurements and not just the contents of those measurements, if a PCR is capped before userland runs, userland can’t fake the same PCR value afterwards. If Linux capped the PCRs before userland started running, we’d be able to place a measurement there before the cap occurred and then prove that that extension occurred before userland had the opportunity to interfere. We could simply place a statement that the kernel supported the NV index restrictions there, and we’d be fine. Unfortunately Linux doesn’t currently do this, and adding support for doing so doesn’t fix the problem - if an attacker boots a kernel that doesn’t cap a PCR, they can just cap it themselves from userland. So, we’re faced with the same problem: booting an older kernel allows the system to be placed in an identical state to the current kernel, and a fake hibernation image can be written out. Solving this requires a PCR that was being modified after kernel code was running, but before userland was started, even with existing kernels. Thankfully, there is one! PCR 5 is defined as containing measurements related to boot management configuration and data. One of the measurements it contains is the result of the UEFI ExitBootServices() call. ExitBootServices() is called at the transition from the UEFI boot environment to the running OS, and the kernel contains code that executes before it. So, if we measure an assertion regarding whether or not we support restricted access to NV indexes into PCR 5 before we call ExitBootServices(), this will prevent userspace from spoofing us (because userspace will only be able to extend PCR 5 after the firmware extended PCR 5 in response to ExitBootServices() being called). Obviously this depends on the firmware actually performing the PCR 5 extension when ExitBootServices() is called, but if firmware’s out of spec then I don’t think there’s any real expectation of it being secure enough for any of this to buy you anything anyway. That’s what this patchset does. Review feedback so far seems positive, although James Bottomley suggested a design improvement. My approach of using an NV index as

a PCR and generating a signing key made sense as a consequence of my original approach involving PCR 23, which had been to use the key creation data and not use audit sessions at all. But the goal isn't to use a key - it's to prove that hibernation occurred in-kernel, and while signing is one way of doing that, there are others. His suggestion is to simply record the hash of the hibernation image into the NV index and then read that back as part of the audit session. On resume we verify the audit session log is legitimate, read the hibernation image, and compare the hash of the hibernation image against the hash in the log. If it matches, and if the PCR 5 value matches, we know the audit session happened in kernel and that the hibernation image hasn't been modified. It still seems like we're missing something, though. If we're not storing a key anywhere, how do we verify the audit session? To explain this we're going to need to go into some details about how TPM keys work in the first place. TPMs can produce a special kind of key called an Attestation Key, or AK. Attestation keys are used to sign attestation statements from the TPM, such as PCR quotes or audit logs. We need a separate key type for this in order to prove that the signature was generated as part of an attestation - normal TPM signing keys are fundamentally designed to sign arbitrary material, and so can be used by userland to sign something that looks identical to an attestation record. By being able to distinguish the types of key the TPM uses, we can prove that a signature was generated using an AK, and so know that it's a real TPM attestation (because nothing other than the TPM can sign stuff with an AK). But even beyond this distinction, TPMs have different key types. Ordinary keys are generated using an RNG and will be unique. Primary keys are generated using a seed that is used as an input into a key derivation function. For a given TPM, using the same seed to generate the same type of primary key will always generate the same primary key8. We can have the kernel deterministically generate an AK during suspend in order to sign the audit log, and then generate the same key again on resume to validate it. This means we don't need any other form of trusted storage for the AK - the TPM already provides that functionality. This is all rather complicated but I've been thinking about this for some time now and I haven't come up with a cleaner solution. The only real open issue is the degree to which machines actually perform the PCR 5 extension. If it turns out there's a bunch that don't we'll need to come up with some alternative approach to identifying whether the kernel supports this functionality, but the design is such that that should be invisible to userspace so adding alternatives later shouldn't be an issue. Filtering accesses to the NV index is also not super clean, and depends on there not being alternative mechanisms to get at the TPM by sending raw commands. Again, this is intended to be opaque to userland and can be improved if we come up with something better later on (such as a secure way to pass a secret to the kernel on boot that can be used as an auth value for acess to the NV index). Anyway. Time to go and fix the next thing. Other architectures use different terminology here, such as "supervisor" and "user" mode, but it's broadly equivalent ↩ In theory rings 1 and 2 would allow you to run drivers with privileges somewhere between full kernel access and userland applications, but in reality we just don't talk about them in polite company ↩ This is how graphics worked in Linux before kernel modesetting turned up. XFree86 would just map your GPU's registers into userland and poke them directly. This was not a huge win for stability ↩ IOMMUs can help you here, by restricting the memory PCI devices can DMA to or from. The kernel then gets to allocate ranges for device buffers and configure the IOMMU such that the device can't DMA to anything else. Except that region of memory may still contain sensitive material such as function pointers, and attacks like this can still cause you problems as a result. ↩ This describes why I'm using "allowed" rather than "required" here ↩ Saving the system state to disk and powering down the platform entirely - significantly slower than suspending the system while keeping state in RAM, but also resilient against the system losing power. ↩ With some handwaving around "coprocessor". TPMs can't be part of the OS or the system firmware, but they don't technically need to be an independent component. Intel have a TPM implementation that runs on the Management Engine, a separate processor built into the motherboard chipset. AMD have one that runs on the Platform Security Processor, a small ARM core built into their CPU. Various ARM implementations run a TPM in

Trustzone, a special CPU mode that (in theory) is able to access resources that are entirely blocked off from anything running in the OS, kernel or otherwise. ↩ Unless the TPM is cleared, in which case it'll generate a new internal seed and now the keys will be different to before the clear ↩

- Linux Plumbers Conference: Free Live Streaming and Matrix Chat available (2026/10/06 06:27)
  To watch any of our live streams on youtube, go to the schedule overview and hover over the paperclip to see the 'Live Stream' link. Live streams are only available for A/V assisted tracks (so not non-A/V BoFs). If you want a two way conversation, you need to have a matrix account, join the Linux Plumbers Space and find the room corresponding to the track you're watching.
- Dave Airlie (blogspot): nouveau on nvidia spark GB10 - it's alive! (2026/09/09 19:56)
  After much back and forth and hoops jumping, I can finally reveal nouveau/nvk running on a NVIDIA Spark box. This is running on a version of nouveau[1] that hasa) ported to the 610 NVIDIA firmware b) a bunch of display rework from Mohamc) a bunch of 0 VRAM and L2 cache handling fixesd) spark boot support e) spark display support and NVK[2] with patches to handle gb10 depth/stencil differences and 0 VRAM support. I'm not sure how best to upstream it all, it's 100 patches and a new firmware which might mean it's a wait for nova type situation, but I just wanted to see it work. [1] https://gitlab.freedesktop.org/nouvelles/kernel/-/commits/nouveau-610-wip-spark[2] https://gitlab.freedesktop.org/airlied/mesa/-/commits/nvk-spark-wip
- Matthew Garrett: SystemIO conflicts are not firmware bugs (2026/09/09 18:15)
  I'm looking at something entirely unrelated, but tripped over some search results that made me realise that a lot of people still think getting errors like ACPI Warning: SystemIO range 0x0000000000001828-0x000000000000182F conflicts with OpRegion 0x0000000000001800-0x000000000000187F indicate a firmware bug. This is generally untrue. We need to dive a little into what ACPI is to clarify why. The Advanced Configuration and Power Interface1 specification defines a whole bunch of stuff, but what's interesting to us here is the hardware abstraction it performs. While PCs are nominally a well-defined platform that's really not true at the hardware level once you get beyond a certain level of complexity. When you suspend a system you want to power down the hardware in the correct order, for instance, and knowing what that order is requires you to know details about the specific motherboard design. The approach taken in the embedded world is to just bake that knowledge into the OS in some form, which is how we end up with Devicetree. ACPI takes an alternative approach - rather than provide that information as data that has to be consumed by OS drivers, it distributes it as code. The ACPI Source Language, or ASL, is a simple language that gets compiled into a bytecode that's then interpreted by the OS at runtime. One of the features of this language is the ability to define "Operation Regions", effectively structure definitions that describe access to underlying hardware. Let's imagine a simple device with two exposed registers. The first is an index register - it describes which internal register we want to access. The second is a data register, where reading it gives us the value of the internal register whose address is currently in the index register, and writing to it modifies that register. An example operation region declaration would look something like 1 2 3 4 5 6 OperationRegion(OPR1, SystemIO, 0x400, 0x2) Field(OPR1, ByteAcc, NoLock, Preserve) { INDX, 8 DATA, 8 } This defines an operation region called "OPR1" at IO port 0x400, 2 bytes long. Inside it are two 8-bit fields, INDX and DATA. These are to be accessed one at a time, do not need the ACPI interpreter to take a global lock when accessing them, and if a subset of the register is modified then the other values should be preserved (irrelevant in this case since the fields are only a byte wide). Now any references to INDX or DATA in this scope will trigger accesses to those registers. So, a method to read the value of register 0x03 would look something like: 1 2 3 4 Method (RD03) { INDX = 0x3 Return (DATA) } ie, set INDX to 3, and then read the value of DATA and return it. But! What if another ACPI method is running at the same time? Let's say we have one that writes to register 0x05: 1 2 3 4 Method (WR05, 1) { INDX = 0x05

DATA = Arg1 } What happens if RD03 executes while we're part-way through WR05? INDX might get reset to 0x03, and now WR05 will modify register 0x03 instead of 0x05. Oh no! But we can avoid this - we declare a mutex (Mutex (MUTX, 0x00)), and update our methods to be something like: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 Method (RD03) { Acquire (MUTX, 0xFFFF) INDX = 0x3 Local0 = DATA Release (MUTX) Return (Local0) } Method (WR05, 1) { Acquire (MUTX, 0xFFFF) INDX = 0x05 DATA = Arg1 Release (MUTX) } Each method takes a lock (waiting up to 0xffff milliseconds and then erroring out if it doesn't), and performs the access. There's now no chance of a race. Phew! Now suppose someone writes a Linux driver for this piece of hardware. It accesses the hardware directly, with no knowledge of ACPI. What stops the driver from racing against one of the ACPI access methods? Nothing at all. Oh no! Again! This isn't hypothetical, by the way - here's a relatively harmless example, but back in the day we did trip over cases where temperature monitoring chips would be accessed by the firmware and Linux simultaneously and as a result you might end up thinking you're reading a temperature when you're actually reading a status flag, resulting in an impossibly high temperature and an immediate thermal shutdown. In this case, the kernel saves you from this (potentially hardware damaging) outcome by printing a message like ACPI Warning: SystemIO range 0x0000000000000400-0x0000000000000401 conflicts with OpRegion 0x0000000000000400-0x0000000000000401 (OPR1), telling you that the kernel has detected that a driver is attempting to allocate IO ports 0x400-0x401, but that there's an ACPI operation region called OPR1 that is claiming the same addresses. The kernel isn't in a position to know what type of access the firmware might perform in that region, so assumes that it might be dangerous and blocks the driver from loading. But all is not lost! The kernel also prints some helpful advice, ACPI: If an ACPI driver is available for this device, you should use it instead of the native driver. And ACPI tables will often actually have a definition that looks like this: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 Device (HDW1) { Name (_HID, "VEND0001") OperationRegion(OPR1, SystemIO, 0x400, 0x2) Field(OPR1, ByteAcc, NoLock, Preserve) { INDX, 8 DATA, 8 } Mutex (MUTX, 0) Method (RD03) { Acquire (MUTX, 0xFFFF) INDX = 0x3 Local0 = DATA Release (MUTX) Return (Local0) } Method (WR05, 1) { Acquire (MUTX, 0xFFFF) INDX = 0x05 DATA = Arg1 Release (MUTX) } } which defines an ACPI device and associated methods. The _HID field defines the device type, and a Linux driver can be written that will be automatically loaded if a device with type VEND0001 is seen. That driver can then call ACPI methods associated with the device and access the resources in a way that matches the firmware's expectations. (Interested in writing such a driver? I wrote a guide back in 2009) The firmware did absolutely nothing wrong here2, but trying to load the native driver will generate an error and the internet will tell you that PC firmware developers are incompetent3 and you should pass a kernel argument that overrides this behaviour and it never did them any harm, and it probably won't do you any harm either but it might and you might never know why your system occasionally wedges or catches fire. The ACPI spec used to live at acpi.info, but sadly that seems to have vanished some time after UEFI took over stewardship of the spec ↩ You might argue that the firmware should simply not do anything at runtime because it is not the firmware's job to do that, and I do understand that and you can certainly boot with acpi=off if you want to and no ACPI code will be executed at runtime. Let me know how that goes. ↩ I'm not going to present an opinion on that here, merely say that this provides no supporting evidence for that assertion ↩

- Matthew Garrett: Hooking an old magicJack adapter to modern Asterisk (2026/08/26 04:04)
  I'm on a VPN setup with several friends that, obviously, includes a VoIP network. I also have an old magicJack adapter and a deep and abiding need to use hardware in ways I should not. There was obvious synergy here. Plugging in the magicJack gives a USB vendor id of 0x06e6, which belonged to a company called TigerJet who made a range of chips for hooking up phones to computers, either via USB or PCI. Some more digging suggested that it was a 580 part, and someone had conveniently uploaded some reference code and datasheets, so figuring out how to talk to

the chip wasn’t terribly difficult. Once configured it simply sends HID events whenever a user hits a phone key or changes the hook state, and otherwise exposes a USB audio device that can be spoken to using the stock kernel driver. It also has the ability to generate dial tone and assert ring signal, giving a full traditional phone experience. So you’d think this would be a super easy project, but I’d made things harder for myself by deciding I wanted to tie directly into Asterisk rather than just smashing an existing SIP stack onto the device. Asterisk uses channels to talk to devices, and channels end up as compiled C code that Asterisk can load dynamically. I didn’t want to have to deal with the pain of compiling stuff and matching ABIs and everything so writing a new channel from scratch was unappealing. Fortunately, the websocket channel is available in recent versions of Asterisk and provides a convenient way to get audio in and out, but that still leaves the job of handling incoming and outgoing calls. That’s handled with the Asterisk Rest Interface, which can initiate a call or respond to an incoming one and bridge various channels together to produce a bidirectional audio stream. There’s a convenient async Python library that handles the low level protocol. Code for all this is here1, and works for my use case, but I should really abstract out the asterisk side and the magicJack side to make it easier to adapt to other devices. That’s a job for later, though. For now, you get this: Your browser doesn't support HTML5 video. Here is a link to the video instead. This has also been an excuse for me to figure out how to make Tangled work, which I’ll write about at some later point. But self-hosted git repo with a convenient collaboration plane! ↩

- Linux Plumbers Conference: Linux Plumbers Conference 2026 is now SOLD OUT! (2026/08/15 14:25)
  It has become a trend for us now. Linux Plumbers Conference 2026 that is to be held in Prague on October 5th through to the 7th is now sold out. But do not fret, you may still be able to attend, but to do so, you will need to add yourself to the wait-list. Please note that the wait-list is a priority ordered list but adding yourself early still does help. The highest priority is for people with accepted content, followed by people who have submitted content but it was not accepted. Then we look at the rest. Each of these priority levels are processed in a first-come/first-served fashion. Thus, the earlier you add yourself, the higher you are in your priority level. We also take the “latest date you can be registered” seriously. If a spot opens up after the date you put in there and you are the next in line, you will be skipped. Good luck everyone who wants to attend but still did not buy a ticket. We hope the wait-list will be empty by the time the conference begins.
- Linux Plumbers Conference: Microconference Topic CfP is coming to an End (2026/08/05 19:52)
  This Friday (August 7th) is the last day to submit a topic to any of the Microconferences. If you have something interesting to discuss at a Microconference, do not hesitate, submit now! It’s also a good idea to reread The Ideal Microconference Topic Session. Notifications for accepted and rejected topics will happen on or before August 21st.
- Linux Plumbers Conference: Linux Plumbers Conference Registration is now Open (2026/07/24 16:05)
  We’re pleased to announce Linux Plumbers Conference registration is now open. Please see our attend page for details.
- Matthew Garrett: Securing agentic identity (2026/07/03 00:38)
  As is the case for many people working in the security industry, the last few months of my life have been focused on dealing with people wanting to use LLMs everywhere. From an enterprise security perspective that’s not an inherent problem - what’s more of a problem is that people want those agents to have access to resources like their calendar and email and so on, and now we have somewhat non-deterministic agents that seem very enthusiastic to achieve what you asked whether that’s a good idea or not, and we’re combining this with credentials that give them access to sensitive data, and leaving those credentials on disk where they can be committed into git repos or exfiltrated to some other service to make use of them on the agent’s behalf or well just any other number of things, at which point your CEO’s email is suddenly readable by

everyone and you're having a bad day. As I mentioned in my last post, pretty much every strong mechanism for keeping credentials in place is just not supported in the wider world. We can imagine a universe where agents use hardware (or at least hypervisor) backed certificates to obtain credentials and any that end up leaking are worthless as a result. But, sadly, that's not an option for most people using existing identity providers. The state of the art is that you use the device code flow and a human authenticates and the token ends up back inside the agent environment and then it proceeds to do whatever it wants with it and you just hope that you wake up the next morning without an awful infoleak occurring. (An aside: I do not like the device code flow as used in enterprise environments, and I never will. The identity provider doesn't have a real opportuity to inspect the security posture of the system asking for the token, and as a result some identity providers will restrict tokens that are issued in this way. The common alternative of doing stuff using a more standard flow and having a redirect URI pointing at localhost works fine for local systems and is a pain for remote ones, even if you can commit crimes with SSH forwarding. I'm going to suggest something that I think is better, and you are free to disagree) I'm not in a position to get every identity provider and service provider to change their security posture, so I'm somewhat stuck in terms of the tokens they're willing to issue me - largely either JWTs or opaque access tokens, with no support for any mechanism of binding that token to an instance. The token that's going to have to be provided to the remote service is something I have little influence over. But that doesn't mean I can't influence the token that lands inside the agent's environment. I can issue a placeholder token to the agent, and force it to communicate via a proxy that swaps out the placeholder for the real thing. The worst the agent can do is exfiltrate the placeholder token, and as long as malicious actors don't have access to that proxy, it doesn't matter - nobody else can do anything with the placeholder. This isn't a terribly novel insight, and it seems like almost everybody has reinvented this on their own. But a lot of these implementations involve you somehow obtaining the real token in advance and then pasting that into something that generates a placeholder that you provide to your agent environment somehow, and it's all a bit clunky and awkward, and it also means that you need to deal with something that keeps track of the mapping between placeholders and real tokens and oh no we've just invented a secret store, and if you want this to work at scale and reliably you're just invented a high availability distributed secret store, and a lot of people who've read that are now shaking their heads and reaching for gin. Can we simplify this, and improve security at the same time? I think we can! Remember when I said "as long as malicious actors don't have access to that proxy, it doesn't matter"? What if they do? What if they compromise one machine inside your environment and are then able to email a bunch of employees and convince their agents to send more tokens back to them and then delete the email before a human reads it? Now you have someone inside the wall with access to those tokens, and presumably with access to the proxy, and now they can be anyone whose agent was gullible enough to think sending them a token was a good idea. This isn't good! So, I thought for a while, and I came up with a new idea. We can have a broker service that obtains credentials for us. We can run that centrally, away from the agents. A client in an agentic environment can request a token, and that can result in a URL being generated and the user being directed to open a URL in a browser and authenticate. When the user authenticates, the authentication flow redirects the confirmation back via the broker, and the broker obtains the real auth token. The obvious thing to do now would be to return the auth token to the client in the agentic environment, but we don't do that. Instead, we mint a new JWT, and add a new claim - one that contains an encrypted copy of the token. In the process we can copy over all the original claims, because those aren't secret - and now even if the client inspects the token to figure out what access it has, it'll get a correct answer. We sign the new token with our own signing key, and pass that back to the client. The client now has a legitimate JWT that is utterly useless, because the signature isn't trusted by anyone other than us. How does it use it? It makes an API request via a proxy, including the new token in the Authorization: header. The proxy verifies the signature on the token, and then decrypts the original token and swaps out the

fake token for the real one. The remote API sees what it expects, and everyone is happy. There's never a real token in the agentic environment, but also we don't need to store anyting anywhere. The only state is the encryption keys, and those can be injected into the environment at startup. You need to scale? Just start more of these processes. You need to support multiple availability zones? Just start more of these processes in different places. No persistent data is ever held in the broker or the proxy. You don't need to care about distributed databases or secret stores. This felt wonderfully elegant and I felt smug about coming up with a better idea, and then I went to a bar earlier this week and sat down to read RFC 8705 and the guy next to me saw that over my shoulder and asked what I was reading and I explained why I was interested and we talked about agentic identity and then he mentioned that fly.io had something that sounded very similar and I read that and gosh yes it is very similar, so damn you fly.io for stealing my ideas 3 years before I even had them. Anyway. Now I need to do better. Remember that there's still a risk around anyone who has access to the proxy having access to the encrypted keys? We can remove that risk as well. It's not uncommon for agentic environments to have an identity issued via something like SPIFFE, at which point they have a client certificate. You can probably guess where I'm going with this. If we require that an agent present a client cert to the broker when requesting a token, we can embed a representation of that client cert into the token we mint. The proxy can then require mTLS for the client connection, and can verify that the presented certificate matches the one represented in the token. If it does then whoever's using the token has access to the private key associated with the environment it was issued to. If we then ensure that the private keys backing these certificates are either hardware or hypervisor backed, and as such tied to a specific instance, we now have a high degree of confidence that the token can only be used in its intended environment. Even if our identity provider doesn't support RFC 8705, we can. This is fairly straightforward where you're using a platform where your identity provider is also the environment that's consuming your tokens, and more annoying for third parties. The broker potentially needs some amount of third party vendor knowledge to make that work for everyone. This is even more the case where login isn't via your identity provider (thanks, github), but none of this is insurmountable - just annoying. And where vendors issue opaque tokens rather than JWTs, this still isn't a problem; we can just mint a new JWT that includes the opaque token as an encrypted claim, and include the same certificate binding. The opaque token ends up being the thing that's presented to the third party, but only after we've verified the mTLS binding. In an ideal world none of this would be necessary - someone would spin up a new agentic environment, a user would prove their identity, and a certificate embodying that identity would be issued to the environment with a private key that can't be exfiltrated. That certificate would be sufficient to obtain new certificates associated with the same private key, and we could still bind that into mTLS identity. This would be much simpler, but browsers don't support it, so it's not likely to happen any time soon. Anyway. Even if we can't have the best thing, we can do better than we are at the moment, and also it would be lovely if we could standardise on this rather than have everyone build their own thing. The end.

- Matthew Garrett: Preventing token theft (2026/07/02 02:23)
  When you log into a service you're given an authentication token. Each further request to the site includes that token, allowing the server to figure out who you are and ensuring that you have access to your data. Depending on site policy, this token may either be stored in memory (and so vanish if you restart your browser) or disk. The token is the proof of your identity. As far as the site is concerned, anyone with your token is you. These tokens may be traditional browser cookies, but they may also be stored in either site local storage or (if you're not using a browser) in some other storage location. In recent years we've seen infostealer malware (like LummaC2) gain the ability to exfiltrate user tokens, allowing attackers to gain access to the user's data without needing to retain access to the user's machine. This attack is viable even if the site has strong MFA requirements, so passkeys don't help. Encrypting the tokens on disk doesn't prevent the malware from scraping them out of the browser's

RAM or obtaining whatever key is used to encrypt them. This feels like a pretty hard problem to solve. But that hasn't stopped people from trying! Dirk Balfanz wrote an IETF draft describing a mechanism for using self-signed certificates for TLS authentication. This uses the mutual authentication feature of the TLS protocol that requires both sides prove their identity to each other. In regular TLS, the remote site presents a signed certificate that tells you who it is. When performing mutual authentication, you then present a certificate to the remote site telling it who you are. These client certificates are largely unused outside enterprise environments because they're a huge pain to deploy. It's not so much that this has sharp edges, it's that it's entirely made of sharp edges. Managing certificate deployment to your devices is hard. Browsers get confused if the certificates change under them. You have one certificate and it lives forever, so sites you present it to can track your identity. Users are prompted to choose a certificate to authenticate with, and if they pick the wrong one everything breaks and is hard to recover. I've deployed this and I did not have a good time. But Balfanz's idea was simple. Rather than require certificates to be deployed, browsers would simply generate a certificate on the fly. The goal wasn't to prove the device or user's identity in any global way - but it would associate a TLS session with a specific certificate. You could then, for example, include a hash of the certificate in the cookie, and if someone tried to use that cookie without presenting that certificate then the cookie could be rejected. If the browser used a hardware-backed private key for the certificate then it would be impossible for an attacker to steal it. Sure, you could still steal cookies, but you wouldn't be able to use them. This was written almost 15 years ago, and seems simple, elegant, and functional. It didn't happen. Part of the reason for that is that, well, it wasn't quite so simple. One problem was privacy related. Cookies are only sent after the TLS session is established, so anyone monitoring the network doesn't know anything about the user identity. A naive implementation of this approach would have meant the client certificate being sent before session establishment, and now user identity can be tracked (no longer an issue if this was implemented on top of TLS 1.3, but this was a log time ago). This was avoided by reordering the client handshake, but that meant having to modify the TLS specification and implementations would have to be updated to support this. Another was that figuring out the granularity of the certificates was difficult. You'd want to use different certificates for every site to avoid them effectively becoming tracking cookies, but you need to provide the certificate before cookies are set, and you don't know what origin the site is going to set in its cookies. If you generate a certificate for a.example.com and a different one for b.example.com, and a.example.com sets a cookie for *.example.com and includes the certificate you used for a.example.com, that cookie isn't going to work on b.example.com and things are broken. This meant supporting it wasn't as straightforward as it seemed - you'd need to ensure that your cookie scope was compatible with the certificate scope. You could probably make this work well enough by aligning it with the Public Suffix List, but there was still some risk of expectations not being aligned. And, perhaps most importantly, TLS session resumption (replaced by pre-shared keys in TLS 1.3) somewhat defeats the purpose of the exercise - clients store state that allows them to re-establish a TLS connection without performing certificate exchange (this reduces overhead if a connection gets interrupted or you switch to a new network or anything along those lines), and anyone in a position to steal cookies could steal that state as well. The followup attempt was channel IDs. This simplified the implementation somewhat - rather than certificates, a raw public key would be sent, along with proof of possession of the private key in the form of a signature over a portion of the TLS handshake. This was required even in the event of session resumption, which avoided having to worry about theft of session secrets. The timing of the exchange was after the encrypted session had been established, so user identity couldn't be leaked that way either. Cookies could then be bound to this identifier. Unfortunately it didn't really deal with the problem of scoping keys in a way that would match cookie requirements, and the spec suggests that the right way of handling this is to scope keys to TLDs, which would enable user tracking across sites (Chrome's implementation apparently restricted it to eTLD+1, which would match the third party cookie policy and avoid the tracking risk).

Chrome added support for this, but it was removed in early 2018. The discussion of some of the pain points in that message is interesting, explicitly calling out problems with connection coalescing across domains and the incompatibility with zero-RTT TLS1.3. The overall consensus at the time seems to be that trying to solve this entirely at the TLS layer has too many rough edges, and a different approach should be taken. And so almost 7 years after the initial draft for origin bound certificates, we come to token binding. This ended up being a rather more complex endeavour, covering 3 different RFCs describing how it impacts TLS, how to incorporate it into HTTP, and how to manage all the various parties involved in the process. The short version is that it's pretty similar to channel ID, except that there's also a documented mechanism for allowing tokens to be bound to one party and consumed by another, avoiding any need for widely scoped keys. Token binding effectively solved all the issues in the original proposal, but at the cost of somewhat more complexity. The RFC was finalised in October 2018. Chrome removed its (incomplete, draft) support for token binding in November 2018. Edge carried support until late 2024. Despite getting all the way through the RFC process, it's functionally dead. The process up until this point had been largely initiated by Google, with Microsoft contributing significantly to the token binding standards. The work had been focused on identifying a generic solution to the problem rather than tying it to any specific authentication flow. The next step was in a different direction - rather than trying to fix this for the entire internet, how about we try to fix it for OAuth? RFC 8705 is titled "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens". This is basically the 2011 approach, but (a) with an explicit definition of how the certificate should be incorporated into issued auth cookies, and (b) with a proviso that well uh if you're going to use tokens issued by your IdP to authenticate to someone else then well you're going to need to use the same cert for both. This is probably fine for the company-owned-laptop case where you're actually fine with multiple sites being able to tie identities together (that's kind of the point here!), and also works for "I am using an app and not a browser", but doesn't work for more generic scenarios. It also doesn't seem to take the session resumption case into account at all? Support for RFC8705 seems poor, as far as I can tell of the big players only Auth0 implements it. In theory it works fine with self-signed client certs but in reality that's going to be almost as difficult to support across multiple platforms as just issuing proper client certs in the first place, so deployment is going to be kind of a pain. But the good news is it doesn't rely on any TLS extensions or custom browser behaviour, so at the client side it works fine with any browser. Which brings us on to RFC 9449, "Demonstrating Proof of Possession". This goes even further than RFC8705 in terms of reducing the burden of deployment - it works fine with existing browsers, and it doesn't even require any certs. The client generates a keypair and provides the pubkey when requesting the cookie. The cookie contains the pubkey. Every request to the service now provides the cookie with the pubkey and also provides a signature over the URI and HTTP method. If the signature matches the pubkey in the token then clearly the signature came from the machine the token was issued to, and everything is good. This does come with some downsides, though. The first is that it uses browser interfaces to generate the keys (typically crypto.subtle.generatekey()) and as far as I can tell there are no browsers that guarantee that that key is going to be generated in hardware even if it's marked non-exportable, so anyone able to steal the cookies can also steal the keys. The second is that the signature only covers the URI and HTTP method, and not the message content or any other headers, so anyone able to exfiltrate a valid signature can replay it against the same URI with different message content. The recommended way to handle this is to reject any signatures that weren't generated within the last few seconds, which is a wonderful additional way to allow clock skew to give you a Bad Day. And the third is that every single request has to be separately signed, which is not intrinsically a problem because computers are fast and have multiple cores, but if you're trying to solve the first problem by sticking the key in a TPM then you're dealing with something that's slow and single threaded and that's maybe acceptable if you're using client certificates (because there's going to be one signature per session and you can use the same session for multiple requests) but

probably not if you're dealing with a user opening a browser that restores previous tabs and each of those is a webapp that fires off 100 requests in parallel. In case it wasn't clear, I don't like DPoP. It doesn't feel like it actually solves the underlying problem that we see in the real world (malware running in a context where if it can grab the tokens it can grab the keys), it adds a massive amount of overhead, and it has baked in replay vulnerabilities. I don't know why it exists and I'm incredibly suspicious of vendors telling me that it fixes my problems, because if they're telling me that then I'm going to end up assuming that they either don't understand my problems or they don't understand their technology, and neither of those is good. Still. Then we get to the thing that prompted me to write this - Chrome's announcement that they had launched device-bound session credentials. This is interesting because it's a Chrome feature that's explicitly intended to counter on-device malware, which was one of the things that was out of scope in 2018 when token binding was being removed. Since this is entire web level it doesn't have to be an RFC, and so is instead defined by W3C. I'm going to handwave all the complexity and say that it's basically a way to register a public key when a cookie is issued, and then prove possession of the private key when it's time to renew the cookie. By making the cookies shortlived and having support for rotating them in the background, user impact is basically zero and while it's still possible for an attacker to exfiltrate and use a cookie they'll only be able to do so for a short window before it needs to be refreshed - something the attacker can't do, since they don't have the private key. This avoids the DPoP overhead because you only need to do signing once per cookie per cookie lifetime, and not on every single request. I don't like this due to the window where exfiltrated tokens can be used, but it feels like a strict improvement over the status quo. An extension called device-bound session credentials for enterprise allows pre-enrollment of device keys, so even though the actual runtime DBCE flow doesn't involve certificates, certificates can be used for device registration in enterprise environments and you can make sure that auth cookies only go to trusted devices. Unfortunately this is Chrome-only, and so we're going to need to wait for it to be backported to all the random app frameworks for it to have widespread support on mobile or for almost everyone's desktop app that's actually three websites in an Electron wrapper. Mozilla's current position is that they're not in favour of it, so I guess we'll see where Safari lands in terms of broad uptake. The last thing on my list is another client cert/OAuth binding, this one still in draft state at the time of writing. This one is aimed primarily at the use of agent-driven tooling, where you have something running in the background using a whole bunch of tools that are each acting on your behalf. Authenticating to all of them separately isn't a fun time, but giving broadly scoped access tokens to a non-deterministic agent and trusting that it'll never post them somewhere public also isn't a fun time. The key distinction between it and RFC8705 is that it's aimed at connections rather than sessions, which avoids the worries about session resumption. This is done with TLS Exporters, which in TLS 1.3 should be unique to the connection even over session resumption (TLS 1.2 may reuse some of the same key material for exporters over session resumption, so it's recommended to enforce 1.3 for this). By providing a new signature alongside the cookie on every new connection, the client proves that it still has access to the private key. This is a very new spec and I haven't had much time to work through it yet, but my naive understanding is that unlike RFC8705 this would require some additional client support to be able to regenerate the client signature on every TLS reconnection. This doesn't avoid all the problems that RFC8705 has, including how to scope certificates. For the agentic use case that probably doesn't matter - all these tools are acting on behalf of the same user, it's fine if all the sites involved know they're the same user. But it doesn't solve the general purpose user use case, and right now DBSC seems like the best we have there. But. Part of me still wonders whether Dirk Balfanz's approach was the right one. Yes, there's risk associated with TLS session resumption, but in the worst case you could just switch that off for high risk setups. The cookie scope argument is real, and also in cases where it could violate privacy the site owner could already choose to broaden their cookie scope and violate your privacy, and in cases where it breaks things you could just not make use of it. The other problems are largely fixed by TLS

1.3, and then we're just left with "Browsers handle client certificates badly" to which my answer is "Yes, and we should fix that anyway". Despite having a pretty good answer to this solution over a decade ago, the closest we have to actual deployment is something that offers strictly worse security guarantees. And tokens keep getting stolen, and compromises keep occurring, and for the most part people shrug and get on with things.

- Dave Airlie (blogspot): Appearing on the Software Engineering Radio Podcast (2026/06/04 00:05)
  Software Engineering Radio is a podcast for people in IT/development with over 700 episodes across many topics over 20 years. They haven't touched on the Linux kernel much. I was invited on as part of my role at Red Hat as a Distinguished Engineer, but the podcast is really an insight into kernel maintenance, in graphics and beyond, touching on the scope and scale of the project.It was my first time to record something that wasn't just me talking at a conference/meetup, and it was all very professional, with sound checks and brainstorming before hand. The content is at a pretty broad and introductory level. We talked about kernel development processes, maintenance processes, and we touch on rust in the kernel a bit. It's mostly about the sheer size and scale of the project and how Linus releases things, how trees get to Linus and how the GPU work is done. Hopefully you enjoy listening to it![1] https://se-radio.net/2026/06/se-radio-723-dave-airlie-on-linux-kernel-maintenance/
- Linux Plumbers Conference: All Microconferences have now been accepted! (2026/06/01 20:25)
  Hello Linux Plumbers community! All Microconferences have now been accepted. Please go ahead and take a look at them and if you find something you would like to discuss, please submit a contribution. To do so, go to the Call for Proposals page, select the Submit new abstract button, add your discussion topic and submit. Make sure to select the proper Microconference in the Track field. It is also a good idea to read the blog about The Ideal Microconference Topic Submission to understand what is expected of a microconference topic.
- Linux Plumbers Conference: Changes to Registration Availability for 2026 (2026/04/06 16:13)
  As most of you are painfully aware, Linux Plumbers Conference registrations can run out very fast (yes, we got lots of complaints last year). This year, we're taking a couple of steps to alleviate the issue. Firstly, we're expanding the venue size in Prague to match the number of attendees we got in Vienna (800) which will hopefully mean we have more than enough places to keep registration open all the way up to the beginning of the conference. Secondly, we're going to have an pre-registration period starting two weeks before general registration opens for anyone who submits content. The way this will work is that if you submit anything via indico before general registration opens, you'll receive a voucher and instructions to participate (this applies to every track and MC submission regardless of the accept/reject or pending state). The cost will be the same as general registration (US$600) but you'll be under no obligation to take up the voucher, which will expire when general registration opens. We're aligning the acceptance/rejection notices of the tracks we directly control (Refereed and Kernel Summit) to be complete around the time we open pre-registration. However, for other tracks and MC submissions that aren't aligned, if you take up an early registration voucher but are subsequently offered a free pass, we'll refund it (although if your company pays, we'd appreciate not having to since cvent charges us). As a reminder of free pass distribution: every accepted Track Speaker (Refereed, Kernel Summit, Net, BPF and Toolchain) gets a free pass. However, Microconferences operate differently and accepted Microconference discussion leads may not receive a free pass (Microconferences have two free passes each and can distribute them arbitrarily to encourage key attendees). The anticipated date for the opening of pre-registration is Friday 10 July 2026 (with the first day of general registration being 24 July), but remember this may change due to logistical problems with the cvent website (which we don't control).
- Dave Airlie (blogspot): drm subsystem contributor numbers (2026/04/01 20:59)

I'm doing a podcast recording this week, so I wanted to run some numbers so I could have some facts rather than feels. It turns out my feels were off by a factor of 3 or so.If asked, I've always said the contributor count to the drm subsystem is probably in the 100 or so developers per release cycle.Did the simplest:git log --format='%aN' v6.14..v6.15 drivers/gpu/drm/ include/uapi/drm/ include/drm/ | sort -u | wc -lIterated over a few kernel releasesv6.15 326v6.16 322v6.17 300v6.18 334v6.19 332v7.0-rc6 346 The number for the complete kernel in those scenarios are ~2000 usually, which means drm subsystem has around 15-16% of the kernel contributors.I'm a bit spun out, that's quite a lot of people. I think I'll blame Sima for it. This also explains why I'm a bit out of touch with the process problems other maintainers have, and when I say stuff like a lot of workflows don't scale, this is what I mean.

- Matthew Garrett: Self hosting as much of my online presence as practical (2026/04/01 02:35)
  Because I am bad at giving up on things, I've been running my own email server for over 20 years. Some of that time it's been a PC at the end of a DSL line, some of that time it's been a Mac Mini in a data centre, and some of that time it's been a hosted VM. Last year I decided to bring it in house, and since then I've been gradually consolidating as much of the rest of my online presence as possible on it. I mentioned this on Mastodon and a couple of people asked for more details, so here we are. First: my ISP doesn't guarantee a static IPv4 unless I'm on a business plan and that seems like it'd cost a bunch more, so I'm doing what I described here: running a Wireguard link between a box that sits in a cupboard in my living room and the smallest OVH instance I can, with an additional IP address allocated to the VM and NATted over the VPN link. The practical outcome of this is that my home IP address is irrelevant and can change as much as it wants - my DNS points at the OVH IP, and traffic to that all ends up hitting my server. The server itself is pretty uninteresting. It's a refurbished HP EliteDesk which idles at 10W or so, along 2TB of NVMe and 32GB of RAM that I found under a pile of laptops in my office. We're not talking rackmount Xeon levels of performance, but it's entirely adequate for everything I'm doing here. So. Let's talk about the services I'm hosting. Web This one's trivial. I'm not really hosting much of a website right now, but what there is is served via Apache with a Let's Encrypt certificate. Nothing interesting at all here, other than the proxying that's going to be relevant later. Email Inbound email is easy enough. I'm running Postfix with a pretty stock configuration, and my MX records point at me. The same Let's Encrypt certificate is there for TLS delivery. I'm using Dovecot as an IMAP server (again with the same cert). You can find plenty of guides on setting this up. Outbound email? That's harder. I'm on a residential IP address, so if I send email directly nobody's going to deliver it. Going via my OVH address isn't going to be a lot better. I have a Google Workspace, so in the end I just made use of Google's SMTP relay service. There's various commerical alternatives available, I just chose this one because it didn't cost me anything more than I'm already paying. Blog My blog is largely static content generated by Hugo. Comments are Remark42 running in a Docker container. If you don't want to handle even that level of dynamic content you can use a third party comment provider like Disqus. Mastodon I'm deploying Mastodon pretty much along the lines of the upstream compose file. Apache is proxying /api/v1/streaming to the websocket provided by the streaming container and / to the actual Mastodon service. The only thing I tripped over for a while was the need to set the "X-Forwarded-Proto" header since otherwise you get stuck in a redirect loop of Mastodon receiving a request over http (because TLS termination is being done by the Apache proxy) and redirecting to https, except that's where we just came from. Mastodon is easily the heaviest part of all of this, using around 5GB of RAM and 60GB of disk for an instance with 3 users. This is more a point of principle than an especially good idea. Bluesky I'm arguably cheating here. Bluesky's federation model is quite different to Mastodon - while running a Mastodon service implies running the webview and other infrastructure associated with it, Bluesky has split that into multiple parts. User data is stored on Personal Data Servers, then aggregated from those by Relays, and then displayed on Appviews. Third parties can run any of these, but a user's actual posts are stored on a PDS. There

are various reasons to run the others, for instance to implement alternative moderation policies, but if all you want is to ensure that you have control over your data, running a PDS is sufficient. I followed these instructions, other than using Apache as the frontend proxy rather than nginx, and it's all been working fine since then. In terms of ensuring that my data remains under my control, it's sufficient. Backups I'm using borgmatic, backing up to a local Synology NAS and also to my parents' home (where I have another HP EliteDesk set up with an equivalent OVH IPv4 fronting setup). At some point I'll check that I'm actually able to restore them. Conclusion Most of what I post is now stored on a system that's happily living under a TV, but is available to the rest of the world just as visibly as if I used a hosted provider. Is this necessary? No. Does it improve my life? In no practical way. Does it generate additional complexity? Absolutely. Should you do it? Oh good heavens no. But you can, and once it's working it largely just keeps working, and there's a certain sense of comfort in knowing that my online presence is carefully contained in a small box making a gentle whirring noise.

- Linux Plumbers Conference: Submission time for Linux Plumbers 2026 (2026/03/23 20:51)
  Submissions for the Refereed Track, Kernel Summit, BoF and Microconferences are now open. Linux Plumbers will be held this year in Prague, Czechia from October 5-7th. The Refereed presentations are 45 minutes in length (Talk+Q&A) 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 Kernel Summit track is divided into sessions of 45 minutes. Sessions may be focused on the discussion of specific Linux kernel topics which can be better resolved in person than over e-mail. The program committee will also consider "information sharing" topics if they are clearly of interest to the wider development community (i.e., advanced training in topics that would be useful to kernel developers). In addition to submitting proposals to the Linux Plumber website, please also send a copy of the proposal as an e-mail to the ksummit@lists.linux.dev mailing list with a subject prefix of [TECH TOPIC]. 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). Presenters and Microconference subtopic leads should be physically present at the conference. Remote presentations will be allowed strictly on an emergency basis. The Microconference Track submissions end at 11:59PM UTC on Thursday, April 23, 2026, submissions for subtopics within a Microconference will open shortly after the Microconference has been fully accepted. The Refereed Track and Kernel Summit submissions end at 11:59PM UTC on Sunday, June 28, 2026. Please submit your Refereed Track, Kernel Summit, BoF 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.
- Matthew Garrett: SSH certificates and git signing (2026/03/21 19:38)
  When you're looking at source code it can be helpful to have some evidence indicating who wrote it. Author tags give a surface level indication, but it turns out you can just lie and if someone isn't paying attention when merging stuff there's certainly a risk that a commit could be merged with an author field that doesn't represent reality. Account compromise can make this even worse - a PR being opened by a compromised user is going to be hard to distinguish from the authentic user. In a world where supply chain security is an increasing concern, it's easy to understand

why people would want more evidence that code was actually written by the person it's attributed to. git has support for cryptographically signing commits and tags. Because git is about choice even if Linux isn't, you can do this signing with OpenPGP keys, X.509 certificates, or SSH keys. You're probably going to be unsurprised about my feelings around OpenPGP and the web of trust, and X.509 certificates are an absolute nightmare. That leaves SSH keys, but bare cryptographic keys aren't terribly helpful in isolation - you need some way to make a determination about which keys you trust. If you're using someting like GitHub you can extract that information from the set of keys associated with a user account1, but that means that a compromised GitHub account is now also a way to alter the set of trusted keys and also when was the last time you audited your keys and how certain are you that every trusted key there is still 100% under your control? Surely there's a better way. SSH Certificates And, thankfully, there is. OpenSSH supports certificates, an SSH public key that's been signed by some trusted party and so now you can assert that it's trustworthy in some form. SSH Certificates also contain metadata in the form of Principals, a list of identities that the trusted party included in the certificate. These might simply be usernames, but they might also provide information about group membership. There's also, unsurprisingly, native support in SSH for forwarding them (using the agent forwarding protocol), so you can keep your keys on your local system, ssh into your actual dev system, and have access to them without any additional complexity. And, wonderfully, you can use them in git! Let's find out how. Local config There's two main parameters you need to set. First, 1 git config set gpg.format ssh because unfortunately for historical reasons all the git signing config is under the gpg namespace even if you're not using OpenPGP. Yes, this makes me sad. But you're also going to need something else. Either user.signingkey needs to be set to the path of your certificate, or you need to set gpg.ssh.defaultKeyCommand to a command that will talk to an SSH agent and find the certificate for you (this can be helpful if it's stored on a smartcard or something rather than on disk). Thankfully for you, I've written one. It will talk to an SSH agent (either whatever's pointed at by the SSH_AUTH_SOCK environment variable or with the -agent argument), find a certificate signed with the key provided with the -ca argument, and then pass that back to git. Now you can simply pass -S to git commit and various other commands, and you'll have a signature. Validating signatures This is a bit more annoying. Using native git tooling ends up calling out to ssh-keygen2, which validates signatures against a file in a format that looks somewhat like authorized-keys. This lets you add something like: 1 * cert-authority ssh-rsa AAAA… which will match all principals (the wildcard) and succeed if the signature is made with a certificate that's signed by the key following cert-authority. I recommend you don't read the code that does this in git because I made that mistake myself, but it does work. Unfortunately it doesn't provide a lot of granularity around things like "Does the certificate need to be valid at this specific time" and "Should the user only be able to modify specific files" and that kind of thing, but also if you're using GitHub or GitLab you wouldn't need to do this at all because they'll just do this magically and put a "verified" tag against anything with a valid signature, right? Haha. No. Unfortunately while both GitHub and GitLab support using SSH certificates for authentication (so a user can't push to a repo unless they have a certificate signed by the configured CA), there's currently no way to say "Trust all commits with an SSH certificate signed by this CA". I am unclear on why. So, I wrote my own. It takes a range of commits, and verifies that each one is signed with either a certificate signed by the key in CA_PUB_KEY or (optionally) an OpenPGP key provided in ALLOWED_PGP_KEYS. Why OpenPGP? Because even if you sign all of your own commits with an SSH certificate, anyone using the API or web interface will end up with their commits signed by an OpenPGP key, and if you want to have those commits validate you'll need to handle that. In any case, this should be easy enough to integrate into whatever CI pipeline you have. This is currently very much a proof of concept and I wouldn't recommend deploying it anywhere, but I am interested in merging support for additional policy around things like expiry dates or group membership. Doing it in hardware Of course, certificates don't buy you any additional security if an attacker is able to steal your private key

material - they can steal the certificate at the same time. This can be avoided on almost all modern hardware by storing the private key in a separate cryptographic coprocessor - a Trusted Platform Module on PCs, or the Secure Enclave on Macs. If you're on a Mac then Secretive has been around for some time, but things are a little harder on Windows and Linux - there's various things you can do with PKCS#11 but you'll hate yourself even more than you'll hate me for suggesting it in the first place, and there's ssh-tpm-agent except it's Linux only and quite tied to Linux. So, obviously, I wrote my own. This makes use of the go-attestation library my team at Google wrote, and is able to generate TPM-backed keys and export them over the SSH agent protocol. It's also able to proxy requests back to an existing agent, so you can just have it take care of your TPM-backed keys and continue using your existing agent for everything else. In theory it should also work on Windows3 but this is all in preparation for a talk I only found out I was giving about two weeks beforehand, so I haven't actually had time to test anything other than that it builds. And, delightfully, because the agent protocol doesn't care about where the keys are actually stored, this still works just fine with forwarding - you can ssh into a remote system and sign something using a private key that's stored in your local TPM or Secure Enclave. Remote use can be as transparent as local use. Wait, attestation? Ah yes you may be wondering why I'm using go-attestation and why the term "attestation" is in my agent's name. It's because when I'm generating the key I'm also generating all the artifacts required to prove that the key was generated on a particular TPM. I haven't actually implemented the other end of that yet, but if implemented this would allow you to verify that a key was generated in hardware before you issue it with an SSH certificate - and in an age of agentic bots accidentally exfiltrating whatever they find on disk, that gives you a lot more confidence that a commit was signed on hardware you own. Conclusion Using SSH certificates for git commit signing is great - the tooling is a bit rough but otherwise they're basically better than every other alternative, and also if you already have infrastructure for issuing SSH certificates then you can just reuse it4 and everyone wins. Did you know you can just download people's SSH pubkeys from github from https://github.com/<username>.keys? Now you do ↩ Yes it is somewhat confusing that the keygen command does things other than generate keys ↩ This is more difficult than it sounds ↩ And if you don't, by implementing this you now have infrastructure for issuing SSH certificates and can use that for SSH authentication as well. ↩

- Matthew Garrett: To update blobs or not to update blobs (2026/03/03 03:09)
  A lot of hardware runs non-free software. Sometimes that non-free software is in ROM. Sometimes it's in flash. Sometimes it's not stored on the device at all, it's pushed into it at runtime by another piece of hardware or by the operating system. We typically refer to this software as "firmware" to differentiate it from the software run on the CPU after the OS has started1, but a lot of it (and, these days, probably most of it) is software written in C or some other systems programming language and targeting Arm or RISC-V or maybe MIPS and even sometimes x862. There's no real distinction between it and any other bit of software you run, except it's generally not run within the context of the OS3. Anyway. It's code. I'm going to simplify things here and stop using the words "software" or "firmware" and just say "code" instead, because that way we don't need to worry about semantics. A fundamental problem for free software enthusiasts is that almost all of the code we're talking about here is non-free. In some cases, it's cryptographically signed in a way that makes it difficult or impossible to replace it with free code. In some cases it's even encrypted, such that even examining the code is impossible. But because it's code, sometimes the vendor responsible for it will provide updates, and now you get to choose whether or not to apply those updates. I'm now going to present some things to consider. These are not in any particular order and are not intended to form any sort of argument in themselves, but are representative of the opinions you will get from various people and I would like you to read these, think about them, and come to your own set of opinions before I tell you what my opinion is. THINGS TO CONSIDER Does this blob do what it claims to do? Does it suddenly introduce functionality you don't want? Does it introduce security

flaws? Does it introduce deliberate backdoors? Does it make your life better or worse? You're almost certainly being provided with a blob of compiled code, with no source code available. You can't just diff the source files, satisfy yourself that they're fine, and then install them. To be fair, even though you (as someone reading this) are probably more capable of doing that than the average human, you're likely not doing that even if you are capable because you're also likely installing kernel upgrades that contain vast quantities of code beyond your ability to understand4. We don't rely on our personal ability, we rely on the ability of those around us to do that validation, and we rely on an existing (possibly transitive) trust relationship with those involved. You don't know the people who created this blob, you likely don't know people who do know the people who created this blob, these people probably don't have an online presence that gives you more insight. Why should you trust them? If it's in ROM and it turns out to be hostile then nobody can fix it ever The people creating these blobs largely work for the same company that built the hardware in the first place. When they built that hardware they could have backdoored it in any number of ways. And if the hardware has a built-in copy of the code it runs, why do you trust that that copy isn't backdoored? Maybe it isn't and updates would introduce a backdoor, but in that case if you buy new hardware that runs new code aren't you putting yourself at the same risk? Designing hardware where you're able to provide updated code and nobody else can is just a dick move5. We shouldn't encourage vendors who do that. Humans are bad at writing code, and code running on ancilliary hardware is no exception. It contains bugs. These bugs are sometimes very bad. This paper describes a set of vulnerabilities identified in code running on SSDs that made it possible to bypass encryption secrets. The SSD vendors released updates that fixed these issues. If the code couldn't be replaced then anyone relying on those security features would need to replace the hardware. Even if blobs are signed and can't easily be replaced, the ones that aren't encrypted can still be examined. The SSD vulnerabilities above were identifiable because researchers were able to reverse engineer the updates. It can be more annoying to audit binary code than source code, but it's still possible. Vulnerabilities in code running on other hardware can still compromise the OS. If someone can compromise the code running on your wifi card then if you don't have a strong IOMMU setup they're going to be able to overwrite your running OS. Replacing one non-free blob with another non-free blob increases the total number of non-free blobs involved in the whole system, but doesn't increase the number that are actually executing at any point in time. Ok we're done with the things to consider. Please spend a few seconds thinking about what the tradeoffs are here and what your feelings are. Proceed when ready. I trust my CPU vendor. I don't trust my CPU vendor because I want to, I trust my CPU vendor because I have no choice. I don't think it's likely that my CPU vendor has designed a CPU that identifies when I'm generating cryptographic keys and biases the RNG output so my keys are significantly weaker than they look, but it's not literally impossible. I generate keys on it anyway, because what choice do I have? At some point I will buy a new laptop because Electron will no longer fit in 32GB of RAM and I will have to make the same affirmation of trust, because the alternative is that I just don't have a computer. And in any case, I will be communicating with other people who generated their keys on CPUs I have no control over, and I will also be relying on them to be trustworthy. If I refuse to trust my CPU then I don't get to computer, and if I don't get to computer then I will be sad. I suspect I'm not alone here. Why would I install a code update on my CPU when my CPU's job is to run my code in the first place? Because it turns out that CPUs are complicated and messy and they have their own bugs, and those bugs may be functional (for example, some performance counter functionality was broken on Sandybridge at release, and was then fixed with a microcode blob update) and if you update it your hardware works better. Or it might be that you're running a CPU with speculative execution bugs and there's a microcode update that provides a mitigation for that even if your CPU is slower when you enable it, but at least now you can run virtual machines without code in those virtual machines being able to reach outside the hypervisor boundary and extract secrets from other contexts. When it's put that way, why would I not install the update? And the straightforward

answer is that theoretically it could include new code that doesn't act in my interests, either deliberately or not. And, yes, this is theoretically possible. Of course, if you don't trust your CPU vendor, why are you buying CPUs from them, but well maybe they've been corrupted (in which case don't buy any new CPUs from them either) or maybe they've just introduced a new vulnerability by accident, and also you're in a position to determine whether the alleged security improvements matter to you at all. Do you care about speculative execution attacks if all software running on your system is trustworthy? Probably not! Do you need to update a blob that fixes something you don't care about and which might introduce some sort of vulnerability? Seems like no! But there's a difference between a recommendation for a fully informed device owner who has a full understanding of threats, and a recommendation for an average user who just wants their computer to work and to not be ransomwared. A code update on a wifi card may introduce a backdoor, or it may fix the ability for someone to compromise your machine with a hostile access point. Most people are just not going to be in a position to figure out which is more likely, and there's no single answer that's correct for everyone. What we do know is that where vulnerabilities in this sort of code have been discovered, updates have tended to fix them - but nobody has flagged such an update as a real-world vector for system compromise. My personal opinion? You should make your own mind up, but also you shouldn't impose that choice on others, because your threat model is not necessarily their threat model. Code updates are a reasonable default, but they shouldn't be unilaterally imposed, and nor should they be blocked outright. And the best way to shift the balance of power away from vendors who insist on distributing non-free blobs is to demonstrate the benefits gained from them being free - a vendor who ships free code on their system enables their customers to improve their code and enable new functionality and make their hardware more attractive. It's impossible to say with absolute certainty that your security will be improved by installing code blobs. It's also impossible to say with absolute certainty that it won't. So far evidence tends to support the idea that most updates that claim to fix security issues do, and there's not a lot of evidence to support the idea that updates add new backdoors. Overall I'd say that providing the updates is likely the right default for most users - and that that should never be strongly enforced, because people should be allowed to define their own security model, and whatever set of threats I'm worried about, someone else may have a good reason to focus on different ones. Code that runs on the CPU before the OS is still usually described as firmware - UEFI is firmware even though it's executing on the CPU, which should give a strong indication that the difference between "firmware" and "software" is largely arbitrary ↩ And, obviously 8051 ↩ Because UEFI makes everything more complicated, UEFI makes this more complicated. Triggering a UEFI runtime service involves your OS jumping into firmware code at runtime, in the same context as the OS kernel. Sometimes this will trigger a jump into System Management Mode, but other times it won't, and it's just your kernel executing code that got dumped into RAM when your system booted. ↩ I don't understand most of the diff between one kernel version and the next, and I don't have time to read all of it either. ↩ There's a bunch of reasons to do this, the most reasonable of which is probably not wanting customers to replace the code and break their hardware and deal with the support overhead of that, but not being able to replace code running on hardware I own is always going to be an affront to me. ↩

- Pete Zaitcev: Meanwhile in space (2026/02/19 02:55)
  Jonny Dyer: ODCs (Orbital Data Centers — zaitcev) will happen. The incentives are aligned from too many directions for them not to. But if you're still debating whether datacenters in space "make sense," you've missed the point. The real story is a technology revolution hiding in plain sight. Access to space has transformed over the last decade. Space and terrestrial infrastructure are converging into a single global system. ... Whether any particular infrastructure bet succeeds in the near term matters far less than the fact that the underlying transformation is structural and self-reinforcing.

- Pete Zaitcev: The end of MinIO (2026/02/19 02:46)
  Someone wrote about the collapse of MinIO (as an open-source project): The CNCF badge isn’t a safety net. MinIO was a CNCF-associated project. That association didn’t prevent any of this. The CNCF doesn’t control the licensing or business decisions of associated projects. If your risk model assumes that CNCF membership means long-term stability, MinIO is your counterexample. Swift is not mentioned among the possible alternative by the author.
- Greg Kroah-Hartman: Linux CVE assignment process (2026/02/16 00:00)
  As described previously, the Linux kernel security team does not identify or mark or announce any sort of security fixes that are made to the Linux kernel tree. So how, if the Linux kernel were to become a CVE Numbering Authority (CNA) and responsible for issuing CVEs, would the identification of security fixes happen in a way that can be done by a volunteer staff? This post goes into the process of how kernel fixes are currently automatically assigned to CVEs, and also the other “out of band” ways a CVE can be issued for the Linux kernel project.
- Dave Airlie (blogspot): drm subsystem AI patch review (2026/02/13 06:56)
  This topic came up at kernel maintainers summit and some other groups have been playing around with it, particularly the BPF folks, and Chris Mason's work on kernel review prompts[1] for regressions. Red Hat have asked engineers to investigate some workflow enhancements with AI tooling, so I decided to let the vibecoding off the leash.My main goal:- Provide AI led patch review for drm patches- Don't pollute the mailing list with them at least initially.This led me to wanting to use lei/b4 tools, and public-inbox. If I could push the patches with message-ids and the review reply to a public-inbox I could just publish that and point people at it, and they could consume it using lei into their favorite mbox or browse it on the web.I got claude to run with this idea, and it produced a project [2] that I've been refining for a couple of days.I started with trying to use Chris' prompts, but screwed that up a bit due to sandboxing, but then I started iterating on using them and diverged.The prompts are very directed at regression testing and single patch review, the patches get applied one-by-one to the tree, and the top patch gets the exhaustive regression testing. I realised I probably can't afford this, but it's also not exactly what I want.I wanted a review of the overall series, but also a deeper per-patch review. I didn't really want to have to apply them to a tree, as drm patches are often difficult to figure out the base tree for them. I did want to give claude access to a drm-next tree so it could try apply patches, and if it worked it might increase the review, but if not it would fallback to just using the tree as a reference.Some holes claude fell into, claude when run in batch mode has limits on turns it can take (opening patch files and opening kernel files for reference etc), giving it a large context can sometimes not leave it enough space to finish reviews on large patch series. It tried to inline patches into the prompt before I pointed out that would be bad, it tried to use the review instructions and open a lot of drm files, which ran out of turns. In the end I asked it to summarise the review prompts with some drm specific bits, and produce a working prompt. I'm sure there is plenty of tuning left to do with it.Anyways I'm having my local claude run the poll loop every so often and processing new patches from the list. The results end up in the public-inbox[3], thanks to Benjamin Tissoires for setting up the git to public-inbox webhook.I'd like for patch submitters to use this for some initial feedback, but it's also something that you should feel free to ignore, but I think if we find regressions in the reviews and they've been ignored, then I'll started suggesting it stronger. I don't expect reviewers to review it unless they want to. It was also suggested that perhaps I could fold in review replies as they happen into another review, and this might have some value, but I haven't written it yet. If on the initial review of a patch there is replies it will parse them, but won't do it later. [1] https://github.com/masoncl/review-prompts[2] https://gitlab.freedesktop.org/airlied/patch-reviewer[3] https://lore.gitlab.freedesktop.org/drm-ai-reviews/

- Brendan Gregg: Why I joined OpenAI (2026/02/06 13:00)
  The staggering and fast-growing cost of AI datacenters is a call for performance engineering like no other in history; it's not just about saving costs – it's about saving the planet. I have joined OpenAI to work on this challenge directly, with an initial focus on ChatGPT performance. The scale is extreme and the growth is mind-boggling. As a leader in datacenter performance, I've realized that performance engineering as we know it may not be enough – I'm thinking of new engineering methods so that we can find bigger optimizations than we have before, and find them faster. It's the opportunity of a lifetime and, unlike in mature environments of scale, it feels as if there are no obstacles – no areas considered too difficult to change. Do anything, do it at scale, and do it today. Why OpenAI exactly? I had talked to industry experts and friends who recommended several companies, especially OpenAI. However, I was still a bit cynical about AI adoption. Like everyone, I was being bombarded with ads by various companies to use AI, but I wondered: was anyone actually using it? Everyday people with everyday uses? One day during a busy period of interviewing, I realized I needed a haircut (as it happened, it was the day before I was due to speak with Sam Altman). Mia the hairstylist got to work, and casually asked what I do for a living. "I'm an Intel fellow, I work on datacenter performance." Silence. Maybe she didn't know what datacenters were or who Intel was. I followed up: "I'm interviewing for a new job to work on AI datacenters." Mia lit up: "Oh, I use ChatGPT all the time!" While she was cutting my hair – which takes a while – she told me about her many uses of ChatGPT. (I, of course, was a captive audience.) She described uses I hadn't thought of, and I realized how ChatGPT was becoming an essential tool for everyone. Just one example: She was worried about a friend who was travelling in a far-away city, with little timezone overlap when they could chat, but she could talk to ChatGPT anytime about what the city was like and what tourist activities her friend might be doing, which helped her feel connected. She liked the memory feature too, saying it was like talking to a person who was living there. I had previously chatted to other random people about AI, including a realtor, a tax accountant, and a part-time beekeeper. All told me enthusiastically about their uses of ChatGPT; the beekeeper, for example, uses it to help with small business paperwork. My wife was already a big user, and I was using it more and more, e.g. to sanity-check quotes from tradespeople. Now my hairstylist, who recognized ChatGPT as a brand more readily than she did Intel, was praising the technology and teaching me about it. I stood on the street after my haircut and let sink in how big this was, how this technology has become an essential aide for so many, how I could lead performance efforts and help save the planet. Joining OpenAI might be the biggest opportunity of my lifetime. It's nice to work on something big that many people recognize and appreciate. I felt this when working at Netflix, and I'd been missing that human connection when I changed jobs. But there are other factors to consider beyond a well-known product: what's my role, who am I doing it with, and what is the compensation? I ended up having 26 interviews and meetings (of course I kept a log) with various AI tech giants, so I learned a lot about the engineering work they are doing and the engineers who do it. The work itself reminds me of Netflix cloud engineering: huge scale, cloud computing challenges, fast-paced code changes, and freedom for engineers to make an impact. Lots of very interesting engineering problems across the stack. It's not just GPUs, it's everything. The engineers I met were impressive: the AI giants have been very selective, to the point that I wasn't totally sure I'd pass the interviews myself. Of the companies I talked to, OpenAI had the largest number of talented engineers I already knew, including former Netflix colleagues such as Vadim who was encouraging me to join. At Netflix, Vadim would bring me performance issues and watch over my shoulder as I debugged and fixed them. It's a big plus to have someone at a company who knows you well, knows the work, and thinks you'll be good at the work. Some people may be excited by what it means for OpenAI to hire me, a well known figure in computer performance, and of course I'd like to do great things. But to be fair on my fellow staff, there are many performance engineers already at OpenAI, including veterans I know from the industry, and they have been busy finding important wins. I'm not

the first, I'm just the latest. Building Orac AI was also an early dream of mine. As a child I was a fan of British SciFi, including Blake's 7 (1978-1981) which featured a sarcastic, opinionated supercomputer named Orac. Characters could talk to Orac and ask it to do research tasks. Orac could communicate with all other computers in the universe, delegate work to them, and control them (this was very futuristic in 1978, pre-Internet as we know it). Orac was considered the most valuable thing in the Blake's 7 universe, and by the time I was a university engineering student I wanted to build Orac. So I started developing my own natural language processing software. I didn't get very far, though: main memory at the time wasn't large enough to store an entire dictionary plus metadata. I visited a PC vendor with my requirements and they laughed, telling me to buy a mainframe instead. I realized I needed it to distinguish hot versus cold data and leave cold data on disk, and maybe I should be using a database… and that was about where I left that project. Last year I started using ChatGPT, and wondered if it knew about Blake's 7 and Orac. So I asked: ChatGPT's response nails the character. I added it to Settings->Personalization->Custom Instructions, and now it always answers as Orac. I love it. (There's also surprising news for Blake's 7 fans: A reboot was just announced!) What's next for me I am now a Member of Technical Staff for OpenAI, working remotely from Sydney, Australia, and reporting to Justin Becker. The team I've joined is ChatGPT performance engineering, and I'll be working with the other performance engineering teams at the company. One of my first projects is a multi-org strategy for improving performance and reducing costs. There's so many interesting things to work on, things I have done before and things I haven't. I'm already using Codex for more than just coding. Will I be doing more eBPF, Ftrace, PMCs? I'm starting with OpenAI's needs and seeing where that takes me; but given those technologies are proven for finding datacenter performance wins, it seems likely -- I can lead the way. (And if everything I've described here sounds interesting to you, OpenAI is hiring.) I was at Linux Plumber's Conference in Toyko in December, just after I announced leaving Intel, and dozens of people wanted to know where I was going next and why. I thought I'd write this blog post to answer everyone at once. I also need to finish part 2 of hiring a performance engineering team (it was already drafted before I joined OpenAI). I haven't forgotten. It took months to wrap up my prior job and start at OpenAI, so I was due for another haircut. I thought it'd be neat to ask Mia about ChatGPT now that I work on it, then realized it had been months and she could have changed her mind. I asked nervously: "Still using ChatGPT?". Mia responded confidently: "twenty-four seven!" I checked with Mia, she was thrilled to be mentioned in my post. This is also a personal post: no one asked me to write this.

- Dave Airlie (blogspot): nouveau: a tale of two bugs (2026/02/04 21:04)
  Just to keep up some blogging content, I'll do where did I spend/waste time last couple of weeks.I was working on two nouveau kernel bugs in parallel (in between whatever else I was doing).Bug 1: Lyude, 2 or 3 weeks ago identified the RTX6000 Ada GPU wasn't resuming from suspend. I plugged in my one and indeed it wasn't. Turned out since we moved to 570 firmware, this has been broken. We started digging down various holes on what changed, sent NVIDIA debug traces to decode for us. NVIDIA identified that suspend was actually failing but the result wasn't getting propogated up. At least the opengpu driver was working properly.I started writing patches for all the various differences between nouveau and opengpu in terms of what we send to the firmware, but none of them were making a difference.I took a tangent, and decided to try and drop the latest 570.207 firmware into place instead of 570.144. NVIDIA have made attempts to keep the firmware in one stream more ABI stable. 570.207 failed to suspend, but for a different reason.It turns out GSP RPC messages have two levels of sequence numbering, one on the command queue, and one on the RPC. We weren't filling in the RPC one, and somewhere in the later 570's someone found a reason to care. Now it turned out whenever we boot on 570 firmware we get a bunch of async msgs from GSP, with the word ASSERT in them with no additional info. Looks like at least some of those messages were due to our missing sequence numbers and fixing that stopped those.And then? still didn't

suspend/resume. Dug into memory allocations, framebuffer suspend/resume allocations. Until Milos on discord said you did confirm the INTERNAL_FBSR_INIT packet is the same, and indeed it wasn't. There is a flag bEnteringGCOff, which you set if you are entering into graphics off suspend state, however for normal suspend/resume instead of runtime suspend/resume, we shouldn't tell the firmware we are going to gcoff for some reason. Fixing that fixed suspend/resume.While I was head down on fixing this, the bug trickled up into a few other places and I had complaints from a laptop vendor and RH internal QA all lined up when I found the fix. The fix is now in drm-misc-fixes.Bug 2: A while ago Mary, a nouveau developer, enabled larger pages support in the kernel/mesa for nouveau/nvk. This enables a number of cool things like compression and gives good speedups for games. However Mel, another nvk developer reported random page faults running Vulkan CTS with large pages enabled. Mary produced a workaround which would have violated some locking rules, but showed that there was some race in the page table reference counting. NVIDIA GPUs post pascal, have a concept of a dual page table. At the 64k level you can have two tables, one with 64K entries, and one with 4K entries, and the addresses of both are put in the page directory. The hardware then uses the state of entries in the 64k pages to decide what to do with the 4k entries. nouveau creates these 4k/64k tables dynamically and reference counts them. However the nouveau code was written pre VMBIND, and fully expected the operation ordering to be reference/map/unmap/unreference, and we would always do a complete cycle on 4k before moving to 64k and vice versa. However VMBIND means we delay unrefs to a safe place, which might be after refs happen. Fun things like  ref 4k, map 4k, unmap 4k, ref 64k, map 64k, unref 4k, unmap 64k, unref 64k can happen, and the code just wasn't ready to handle those. Unref on 4k would sometimes overwrite the entry in the 64k table to invalid, even when it was valid. This took a lot of thought and 5 or 6 iterations on ideas before we stopped seeing fails. In the end the main things were to reference count the 4k/64k ref/unref separately, but also the last thing to do a map operation owned the 64k entry, which should conform to how userspace uses this interface.The fixes for this are now in drm-misc-next-fixes.Thanks to everyone who helped, Lyude/Milos on the suspend/resume, Mary/Mel on the page tables.

- 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 systems1. 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't2, 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 graphics3. 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 once4, 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 modes5 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: LPC 2025 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 fleme 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 resarch tasks, like finding URLs for release notes and recent developments for a given technology. I once used ChatGPT to find and old patch sent to lkml, just based on a broad description, which would otherwise take hours of trial-and-error searches. I keep finding more ways that ChatGPT/AI is useful to me in my work. AI Agents (AI Brendans) A common approach is to take a CPU flame graph and have AI do pattern matching to find performance issues. Some of these agents will apply fixes as well. It's like a modern take on the practice of "recent performance issue checklists," just letting AI do the pattern matching instead of the field engineer. I've recently worked on a Fast by Friday methodology: where we engineer systems so that performance can be root-cause analyzed in 5 days or less. Having an AI agent look over flame graphs, metrics, and other data sources to match previously seen issues will save time and help make Fast by Friday possible. For some companies with few or no performance engineers, I'd expect matching previously seen issues should find roughly 10-50% performance gains. I've heard some flame graph agents privately referred to as an "AI Brendan" (or similar variation on my name) and I guess I should be glad that I'm getting some kind of credit for my work. Calling a systems performance agent "Brendan" makes more sense than other random names like Siri or Alexa, so 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 the 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 PerfInsights that analyzes code profiles to find optimizations. And I learned about another agent, Linnix: AI-Powered Observability, while writing this post. Final Thoughts There are far more computers in the world than performance engineers to tune them, leaving most running untuned and wasting resources. In future there will be AI performance agents that can be run on everything, helping to save the planet by reducing energy usage. Some will be described as an AI Brendan or a Virtual Brendan (some already have been) but that doesn't mean they are necessarily trained on all my work or had any direct help from me creating it. (Nor did they abduct me and feed me into a steampunk machine that uploaded my brain to the cloud.) Virtual Brendans only try to automate about 15% of 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 suggestss, so a commercial offering will keep losing whatever tuning advantages it develops. In-house tools don't have these same concerns, and perhaps that's the real future of AI tuning agents: an in-house or non-commercial open source collaboration.

- Dave Airlie (blogspot): fedora 43: bad mesa update oopsie (2025/11/24 01:42)
  F43 picked up the two patches I created to fix a bunch of deadlocks on laptops reported in my previous blog posting. Turns out Vulkan layers have a subtle thing I missed, and I removed a line from the device select layer that would only matter if you have another layer, which happens under steam.The fedora update process caught this, but it still got published which was a mistake, need to probably give changes like this more karma thresholds.I've released a new update https://bodhi.fedoraproject.org/updates/FEDORA-2025-2f4ba7cd17 that hopefully fixes this. I'll keep an eye on the karma.
- Brendan Gregg: Intel is listening, don't waste your shot (2025/11/21 13:00)

Intel's new CEO, Lip-Bu Tan, has made listening to customers a top priority, saying at Intel Vision earlier this year: "Please be brutally honest with us. This is what I expect of you this week, and I believe harsh feedback is most valuable." I'd been in regular meetings with Intel for several years before I joined, and I had been giving them technical direction on various projects, including at times some brutal feedback. When I finally interviewed for a role at Intel I was told something unexpected: that I had already accomplished so much within Intel that I qualified to be an Intel Fellow candidate. I then had to pass several extra interviews to actually become a Fellow (and was told I may only be the third person in Intel's history to be hired as a Fellow) but what stuck with me was that I had already accomplished so much at a company I'd never worked for. If you are in regular meetings with a hardware vendor as a customer (or potential customer) you can accomplish a lot by providing firm and tough feedback, particularly with Intel today. This is easier said than done, however. Now that I've seen it from the other side I realize I could have accomplished more, and you can too. I regret the meetings where I wasn't really able to have my feedback land as the staff weren't really getting it, so I eventually gave up. After the meeting I'd crack jokes with my colleagues about how the product would likely fail. (Come on, at least I tried to tell them!) Here's what I wish I had done in any hardware vendor meeting: Prep before meetings: study the agenda items and look up attendees on LinkedIn and note what they do, how many staff they say they manage, etc. Be aware of intellectual property risks: Don't accept meetings covered by some agreement that involves doing a transfer of intellectual property rights for your feedback (I wrote a post on this); ask your legal team for help. Make sure feedback is documented in the meeting minutes (e.g., a shared Google doc) and that it isn't watered down. Be firm about what you know and don't know: it's just as important to assert when you haven't formed an opinion yet on some new topic. Stick to technical criticisms that are constructive (uncompetitive, impractical, poor quality, poor performance, difficult to use, of limited use/useless) instead of trash talk (sucks, dumb, rubbish). Check minutes include who was present and the date. Ask how many staff are on projects if they say they don't have the resources to address your feedback (they may not answer if this is considered sensitive) and share industry expectations, for example: “This should only take one engineer one month, and your LinkedIn says you have over 100 staff.” Decline freeloading: If staff ask to be taught technical topics they should already know (likely because they just started a new role), decline, as I'm the customer and not a free training resource. Ask "did you Google it?" a lot: Sometimes staff join customer meetings to elevate their own status within the company, and ask questions they could have easily answered with Google or ChatGPT. Ask for staff/project bans: If particular staff or projects are consistently wasting your time, tell the meeting host (usually the sales rep) to take them off the agenda for at least a year, and don't join (or quit) meetings if they show up. Play bad cop, often no one else will. Review attendees. From time to time, consider: Am I meeting all the right people? Review the minutes. E.g., if you're meeting Intel and have been talking about a silicon change, have any actual silicon engineers joined the call? Avoid peer pressure: You may meet with the entire product team who are adamant that they are building something great, and you alone need to tell them it's garbage (using better words). Many times in my life I've been the only person to speak up and say uncomfortable things in meetings, yet I'm not the only person present who could. Ask for status updates: Be prepared that even if everyone appears grateful and appreciative of your feedback, you may realize six months later that nothing was done with it. Ask for updates and review the prior meeting minutes to see what you asked for and when. Speak to ELT/CEO: Once a year or so, ask to speak to someone on the executive leadership team (ELT; the leaders on the website) or the CEO. Share brutal feedback, and email them a copy of the meeting minutes showing the timeline of what you have shared and with whom. This may be the only way your feedback ever gets addressed, in particular for major changes. Ask to hear what they have been told about you and be prepared to refute details: your brutal feedback may have been watered down. I'm now in meetings from the other side where we'd really appreciate brutal feedback, but some customers aren't comfortable doing this, even when prompted. It isn't easy to tell someone

their project is doomed, or that their reasons for not doing something are BS. It isn't easy dealing with peer pressure and a room of warm and friendly staff begging you say something, anything nice about their terrible product for fear of losing their jobs -- and realizing you must be brutal to their faces otherwise you're not helping the vendor or your own company. And it's extra effort to check meeting minutes and to push for meetings with the ELT or the CEO. Giving brutal feedback takes brutal effort.

- Linux Plumbers Conference: Slides templates available (2025/11/20 22:32)
  Dear speakers, You can find the LPC 2025 slides templates in different formats in the following link: https://drive.google.com/drive/folders/1oGQz6MXtq7fjRJS0Q7Q_oBI91g38VFOC They were created by our designer, Zohar Nir-Amitin. Zohar has been working with LPC since 2015, and has created all our wonderful t-shirts, badges and signage designs.
- Brendan Gregg: Third Stage Engineering (2025/11/16 13:00)
  The real performance of any computer hardware in production is the result of the hardware, software, and tuning; the investment and sequence of these efforts can be pictured as a three-stage rocket: I recently presented this embarrassingly simple diagram to Intel's executive leadership, and at the time realized the value of sharing it publicly. The Internet is awash with comparisons about Intel (and other vendors') product performance based on hardware performance alone, but the performance of software and then tuning can make a huge difference for your particular workload. You need all three stages to reach the highest, and most competitive, performance. It's obvious why this is important for HW vendors to understand internally – they, like the Internet, can get overly focused on HW alone. But customers need to understand it as well. If a benchmark is comparing TensorFlow performance between HW vendors, was the Intel hardware tested using the Intel Extension for TensorFlow Software, and was it then tuned? The most accurate and realistic evaluation for HW involves selecting the best software and then tuning it, and doing this for all HW options. I spend a lot of time on the final stage, tuning – what I call third-stage engineering. It's composed of roughly four parts: People, training, tools, and capabilities. You need staff, you need them trained to understand performance methodologies and SW and HW internals, they need tools to analyze the system (both observational and experimental), and finally they need capabilities to tune (tunable parameters, settings, config, code changes, etc.). I see too many HW evaluations that are trying to understand customer performance but are considering HW alone, which is like only testing the first stage of a rocket. This doesn't help vendors or customers. I hope that's what my simple diagram makes obvious: We need all three stages to reach the highest altitude.
- Dave Airlie (blogspot): a tale of vulkan/nouveau/nvk/zink/mutter + deadlocks (2025/11/10 03:16)
  I had a bug appear in my email recently which led me down a rabbit hole, and I'm going to share it for future people wondering why we can't have nice things.Bug:1. Get an intel/nvidia (newer than Turing) laptop.2. Log in to GNOME on Fedora 42/43 3. Hotplug a HDMI port that is connected to the NVIDIA GPU.4. Desktop stops working.My initial reproduction got me a hung mutter process with a nice backtrace which pointed at the Vulkan Mesa device selection layer, trying to talk to the wayland compositor to ask it what the default device is. The problem was the process was the wayland compositor, and how was this ever supposed to work. The Vulkan device selection was called because zink called EnumeratePhysicalDevices, and zink was being loaded because we recently switched to it as the OpenGL driver for newer NVIDIA GPUs.I looked into zink and the device select layer code, and low and behold someone has hacked around this badly already, and probably wrongly and I've no idea what the code does, because I think there is at least one logic bug in it. Nice things can't be had because hacks were done instead of just solving the problem. The hacks in place ensured under certain circumstances involving zink/xwayland that the device select code to probe the window system was disabled, due to deadlocks seen. I'd no idea if more hacks were going to help, so I decided to step back and try and work out

better.The first question I had is why WAYLAND_DISPLAY is set inside the compositor process, it is, and if it wasn't I would never hit this. It's pretty likely on the initial compositor start this env var isn't set, so the problem only becomes apparent when the compositor gets a hotplugged GPU output, and goes to load the OpenGL driver, zink, which enumerates and hits device select with env var set and deadlocks.I wasn't going to figure out a way around WAYLAND_DISPLAY being set at this point, so I leave the above question as an exercise for mutter devs.How do I fix it?Attempt 1:At the point where zink is loading in mesa for this case, we have the file descriptor of the GPU device that we want to load a driver for. We don't actually need to enumerate all the physical devices, we could just find the ones for that fd. There is no API for this in Vulkan. I wrote an initial proof of concept instance extensions call VK_MESA_enumerate_devices_fd. I wrote initial loader code to play with it, and wrote zink code to use it. Because this is a new instance API, device-select will also ignore it. However this ran into a big problem in the Vulkan loader. The loader is designed around some internals that PhysicalDevices will enumerate in similiar ways, and it has to trampoline PhysicalDevice handles to underlying driver pointers so that if an app enumerates once, and enumerates again later, the PhysicalDevice handles remain consistent for the first user. There is a lot of code, and I've no idea how hotplug GPUs might fail in such situations. I couldn't find a decent path forward without knowing a lot more about the Vulkan loader. I believe this is the proper solution, as we know the fd, we should be able to get things without doing a full enumeration then picking the answer using the fd info. I've asked Vulkan WG to take a look at this, but I still need to fix the bug.Attempt 2:Maybe I can just turn off device selection, like the current hacks do, but in a better manner. Enter VK_EXT_layer_settings. This extensions allows layers to expose a layer setting in the instance creation. I can have the device select layer expose a setting which says don't touch this instance. Then in the zink code where we have a file descriptor being passed in and create an instance, we set the layer setting to avoid device selection. This seems to work but it has some caveats, I need to consider, but I think should be fine.zink uses a single VkInstance for it's device screen. This is shared between all pipe_screens. Now I think this is fine inside a compositor, since we shouldn't ever be loading zink via the non-fd path, and I hope for most use cases it will work fine, better than the current hacks and better than some other ideas we threw around. The code for this is in [1].What else might be affected:If you have a vulkan compositor, it might be worth setting the layer setting if the mesa device select layer is loaded, esp if you set the DISPLAY/WAYLAND_DISPLAY and do any sort of hotplug later. You might be safe if you EnumeratePhysicalDevices early enough, the reason it's a big problem in mutter is it doesn't use Vulkan, it uses OpenGL and we only enumerate Vulkan physical devices at runtime through zink, never at startup.AMD and NVIDIA I think have proprietary device selection layers, these might also deadlock in similiar ways, I think we've seen some wierd deadlocks in NVIDIA driver enumerations as well that might be a similiar problem.  [1] https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/38252

- Linux Plumbers Conference: Japan Visas need a longer processing time (2025/10/29 13:13)
  If you hold a passport from a visa exempt country, this doesn’t apply to you: https://www.mofa.go.jp/j_info/visit/visa/short/novisa.html But if you don’t have a passport from that list, you do need a visa.  Unfortunately, the change of government in Japan has made the process for getting a visa more taxing on the body supplying the invitation letter (in our case, the Linux Foundation).  For this reason, the LF is insisting that anyone who needs a visa letter have their application in to the LF dashboard by 17 November at the latest: https://openprofile.dev/myevents?applyfor=visa-letter If you have any queries or problems with the process, please contact visaletters@linuxfoundation.org
- Pete Zaitcev: Time flies (2025/10/20 04:34)
  A guy who sits next to me is in his 70s, and he said: "I started out on a teletype." But I didn't. Not only I never lived in a world without computers,

but when I started out, CRT displays were already a thing. Guys who worked on vacuum tube computers are in their 90s now.

- Pete Zaitcev: git submodule woe (2025/10/16 01:57)
  Problem:A submodule is stuck in a commit, like so:$ git show..... shows a stuck submodule--- a/badsub+++ b/badsub@@ -1 +1 @@-Subproject commit 4ba912892c1b8c213c6c2e78b3bf257635dc534e+Subproject commit 4b813c322ebe236cddc6b3acd70a31994efd7a56Solution:Focus on the commit, not submodule. Submodules work as designed, it's the commit that needs to be fixed (with `git commit --amend`, obviously):$ cd badsub$ git checkout 4ba912892c1b8c213c6c2e78b3bf257635dc534e$ cd ..$ git add badsub$ git commit --amendNowhere as bad as copying a file while preserving history. Still, not obvious if one focuses on `git submodule`.
- Pete Zaitcev: podman versus dbus (2025/10/16 01:51)
  Problem:`podman container ls` warns:WARN[0000] The cgroupv2 manager is set to systemd but there is no systemd user session availableSolution:$ sudo apt install dbus-user-session; systemctl --user start dbus
- Greg Kroah-Hartman: The only benchmark that matters is... (2025/10/01 00:00)
  ...the one that emulates your real workload. And for me (and probably many of you reading this), that would be “build a kernel as fast as possible.” And for that, I recommend the simple kcbench. I kcbench mentioned it a few years ago, when writing about a new workstation that Level One Techs set up for me, and I’ve been using that as my primary workstation ever since (just over 5 years!).
- Linux Plumbers Conference: In Person Registration is sold out (2025/09/19 13:22)
  Apparently there was quite a bit more demand than we anticipated. We are running a waitlist which you can get on by filling in this form: https://forms.gle/tYjjbyn66q5SQMLPA The venue is smaller this year but we do have a block of reserved passes for MC content so we’ll allocate places to the waitlist after it’s decided how many of them get used.  Note that in order to be fair to everyone, if you sign up for the waitlist you’ll have 7 days to register otherwise your pass will go to the next person.
- Linux Plumbers Conference: Registration for LPC 2025 is now open! (2025/09/15 20:16)
  We’re happy to announce that registration for LPC 2025 is now open. To register please go to our attend page. To try to prevent the instant sellout, we are keeping our cancellation policy of no refunds, only transfers of registrations. You will find more details during the registration process. LPC 2025 follows the Linux Foundation’s health & safety policy. As usual we expect to sell our rather quickly so don’t delay your registration for too long!
- Dave Airlie (blogspot): radv takes over from AMDVLK (2025/09/15 19:08)
  AMD have announced the end of the AMDVLK open driver in favour of focusing on radv for Linux use cases.When Bas and I started radv in 2016, AMD were promising their own Linux vulkan driver, which arrived in Dec 2017. At this point radv was already shipping in most Linux distros. AMD strategy of having AMDVLK was developed via over the wall open source releases from internal closed development was always going to be a second place option at that point.When Valve came on board and brought dedicated developer power to radv, and the aco compiler matured, there really was no point putting effort into using AMDVLK which was hard to package and impossible to contribute to meaningfully for external developers.radv is probably my proudest contribution to the Linux ecosystem, finally disproving years of idiots saying an open source driver could never compete with a vendor provided driver, now it is the vendor provided driver.I think we will miss the open source PAL repo as a reference source and I hope AMD engineers can bridge that gap, but it's often hard to find workarounds you don't know exist to ask about them.

I'm also hoping AMD will add more staffing beyond the current levels especially around hardware enablement and workarounds.Now onwards to NVK victory :-)[1] https://github.com/GPUOpen-Drivers/AMDVLK/discussions/416

- Linux Plumbers Conference: The Call for Proposals is nearing its end! (2025/09/08 16:14)
  The CfPs for the Linux Plumbers events are coming to an end. If you still want to submit, please get your submission in by the deadline. The deadlines are: Refereed track: September 10th (that’s this Wednesday) Kernel Summit track: September 10th (See Ted Tso’s email about how to submit). eBPF track: September 29th Networking track: October 17th Each of the Microconferences has their own last day to submit. Those are listed in the Accepted Microconferences tab on the website. All submissions may be added in the Call for Proposals tab. Click the Submit new abstract button at the bottom of that page, and make sure you select the proper Track.
- 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 break-out story. The oddly prescient punch-line is, he discovers that the successful writer's machine was non-functional. His better wrote his stories manually in secret.I wonder if such scenario may even be possible, programming-wise, if you work in a company that does not micro-manage velocity, or work as a consultant, so that your sausage factory remains behind a curtain.

From:
https://wiki.tromjaro.alexio.tf/ - **TROMjaro wiki**

Permanent link:
**https://wiki.tromjaro.alexio.tf/doku.php?id=news:planet:kernel&rev=1583651764**

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