What's new in the Linux kernel? A year in review

Thomas Petazzoni, Bootlin

Capitole du Libre 2025

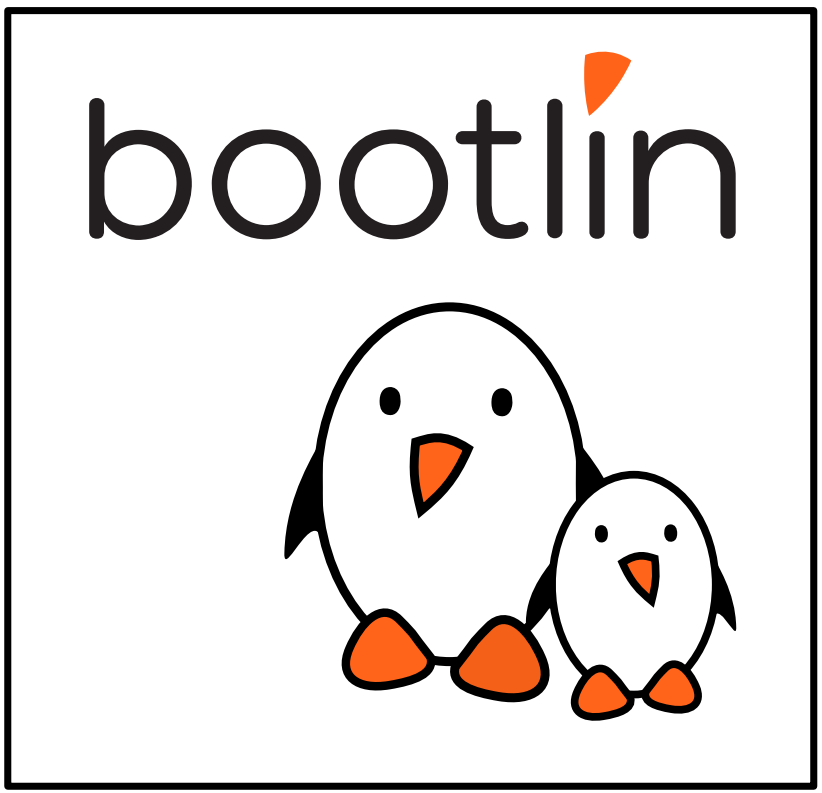
whoami

- Thomas Petazzoni
- CEO of *Bootlin*
  - Engineering company specialized in **Embedded Linux** and **Zephyr**
  - 27 people, mostly Toulouse and Lyon
  - Engineering services
  - Training services
  - Very strong open-source focus
  - We are **hiring**, including **interns**
  - Visit our booth in the main hall
- Former contributor to the *Linux kernel*: 900+ patches
- Co-maintainer and contributor to *Buildroot*
- Co-founder of *Toulibre* and *Capitole du Libre*

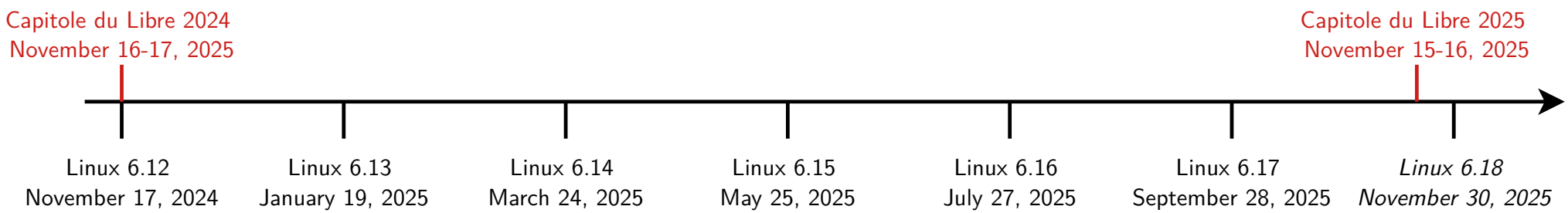
Some advertising: Embedded Toulouse Meetup

- Monthly-ish physical meetup
- Talks about all things embedded
- Some recent talks
  - SELinux
  - Graphics acceleration support in Linux
  - Yocto/Mender
  - Linux in Space
  - Zephyr
- Talks followed by pizza/beer party
- Organized by Bootlin and Smile
- In Toulouse, at La Cantine, just accross ENSEEIHT
- Next meetup: December 9 @ 7 PM
- Details/registration: linkedin.com/company/embedded-meetup-toulouse

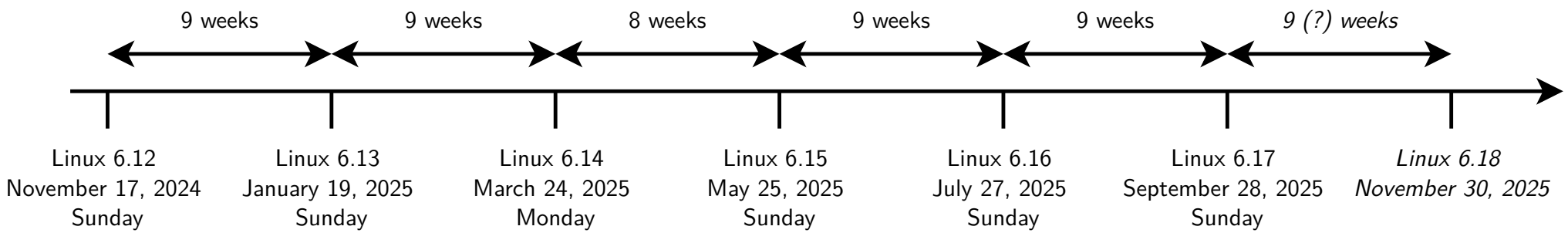
Kernel releases



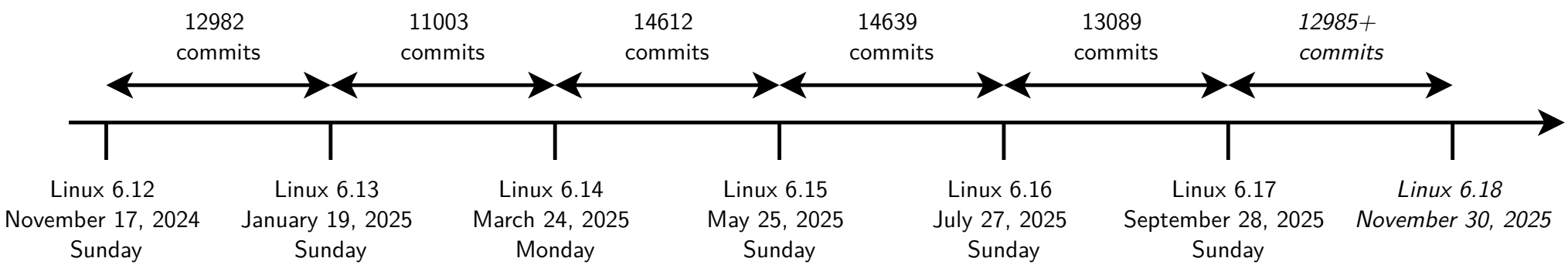
From CdL 2024 to CdL 2025



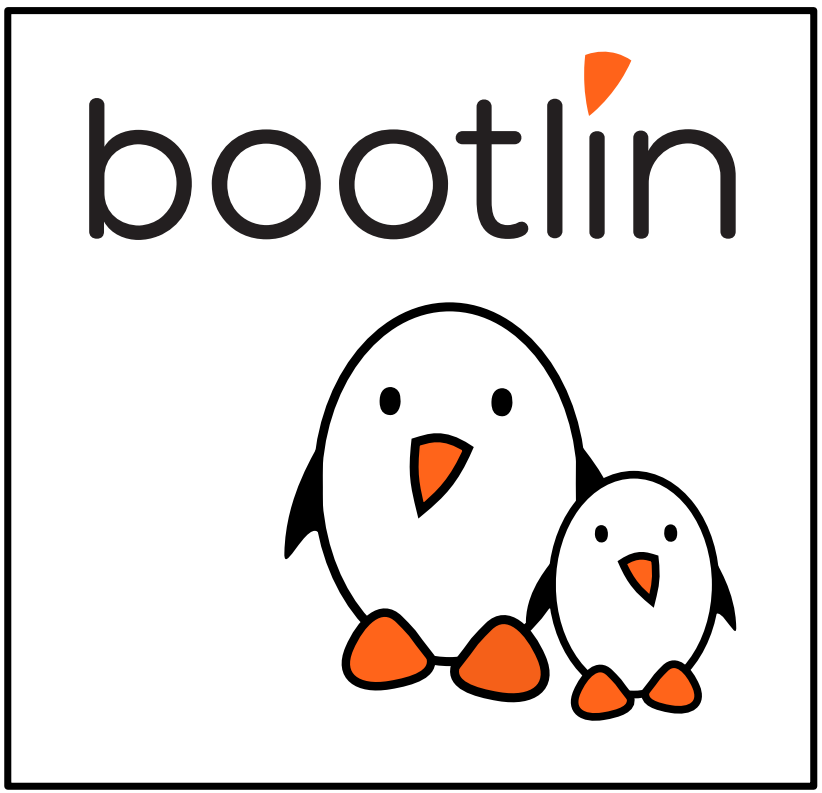
Regular cadence



Lots of changes



Selection of important topics



Disclaimer

- A lot happens in the Linux kernel in each release
- Highlighting “important” topics is obviously highly **subjective**
  - So it is my personal selection of topics I found relevant
- Impossible to cover all topics
- A very significant fraction of kernel changes are related to hardware support
  - Chosen to not cover them in this selection
  - Hundreds/thousands of new drivers

Real-time and PREEMPT_RT

Real-time and PREEMPT_RT

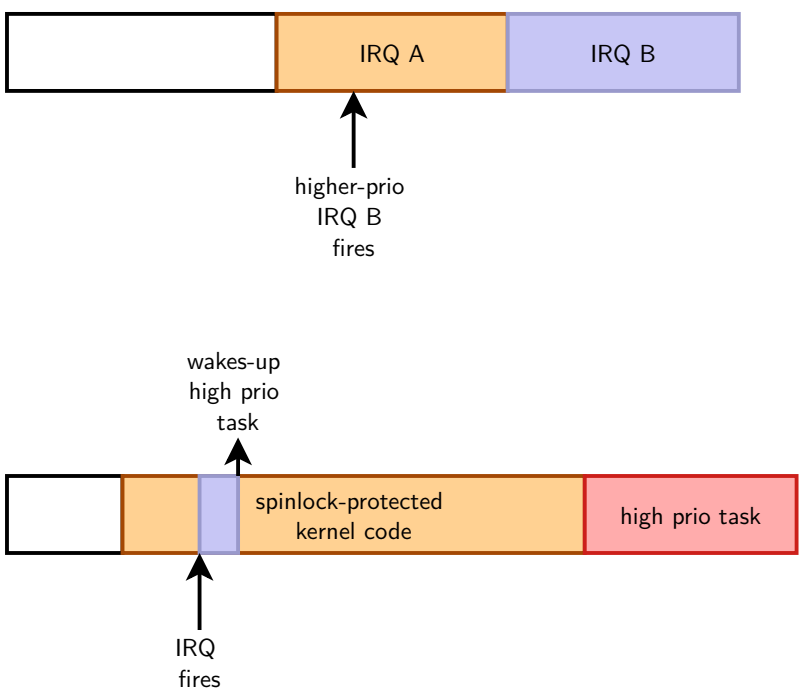
- Linux 6.12 saw the merge of patch finally enabling **PREEMPT_RT**
- Conclusion of 20 years of work to make Linux real-time capable
- The **PREEMPT_RT** patch has been used in the industry for 10-15 years, it is now fully in mainline

Real-time and PREEMPT_RT

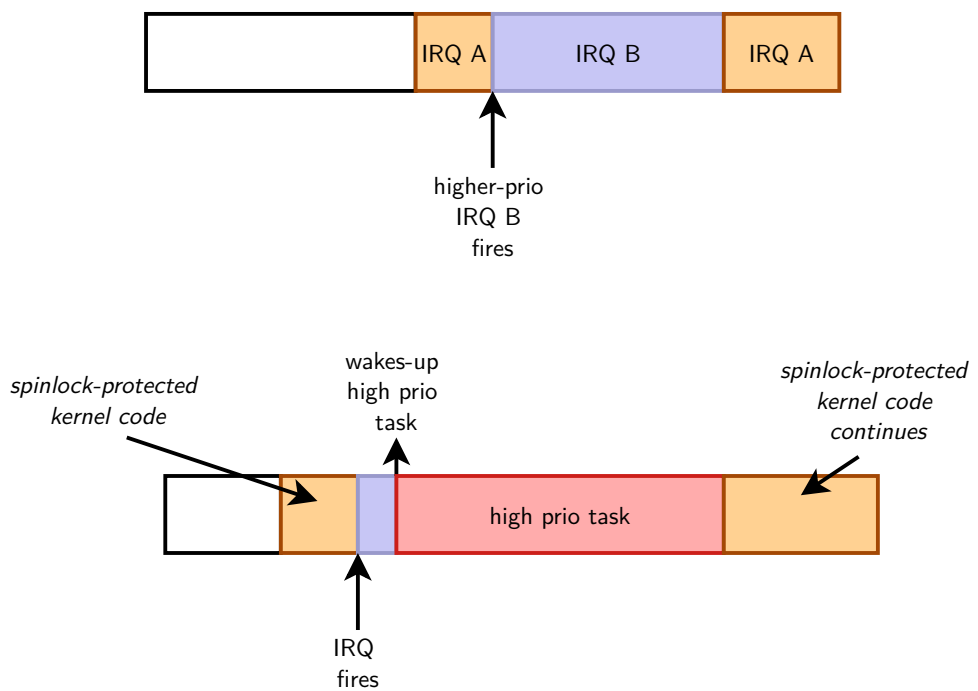
- Real-time is all about **preemption**
  - Being able to interrupt what the system is doing at **any** time to perform some more important processing
  - Before Linux 2.6, only user-space code could be preempted, kernel code was not preemptible
  - Linux 2.6, released in 2003, made the kernel partially preemptible
  - The **PREEMPT_RT** work consisted in making all parts of the kernel preemptible, to the exception of a small number of well-defined sections
  - Guaranteed **worst-case latencies**
  - Last remaining work was around **printk**
- Reminder: real-time != performance

PREEMPT_RT: how it works on one slide

PREEMPT_FULL



PREEMPT_RT



Lazy preemption: some context on preemption

- The kernel offers **four** preemption modes:
  - PREEMPT_NONE
  - PREEMPT_VOLUNTARY
  - PREEMPT_FULL
  - PREEMPT_RT
- Preemption is good for real-time, but less good for overall system throughput: more preemption means more context switches, and less throughput
- PREEMPT_VOLUNTARY is the default in most Linux distributions
- PREEMPT_VOLUNTARY works with explicit preemption points sprinkled throughout the kernel code, **cond_resched()**
- Annoying to sprinkle such calls everywhere, and some might be missing, delaying preemption too much

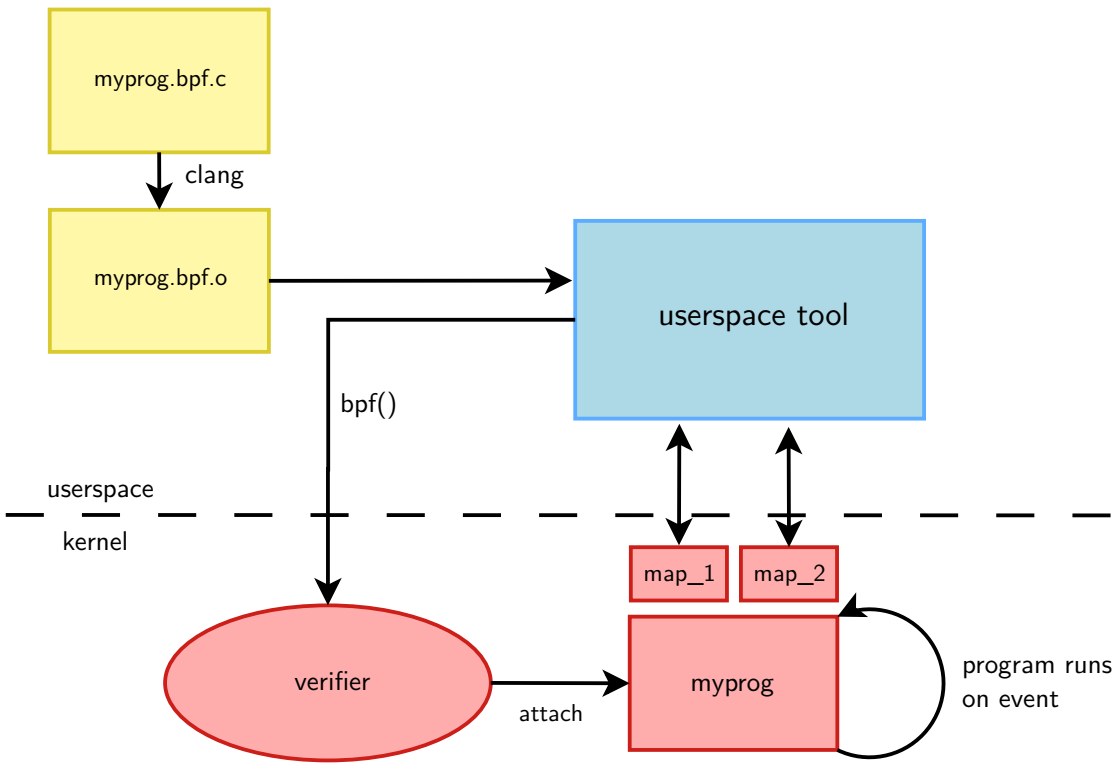
Lazy preemption: the idea

- With full preemption, when a higher prio task is woken up, a flag is set indicating the **need for preemption** is set and checked
  - On the scheduler timer tick
  - On return to user-space from a syscall
  - On return from interrupt
- The idea of **lazy preemption** is to have a **separate** flag that for non real-time tasks ask for preemption but not necessarily right away
  - This lazy flag is not observed at preemption points
  - Lazy flag gets promoted to a normal preemption on scheduler tick
- Consequence
  - For real-time tasks behaves like **PREEMPT_FULL**
  - For non real-time tasks, it will delay preemption to favor throughput over preemption

eBPF

- "Extended BPF", evolution from **B**erkeley **P**acket **F**ilter
- A **virtual machine** inside the kernel, allowing to run user-provided programs directly inside the kernel
  - without having to reboot or modify the kernel
  - safely: cannot make the kernel hang or crash
  - almost anywhere in the kernel
- Initially developed for networking use-cases
  - Allows to inject custom networking logic executed directly within the kernel network drivers
- But being used now for many other things: system monitoring, debugging, profiling, security
- Strong adoption to do all sorts of "custom" stuff with the kernel

eBPF: principle



eBPF: from 6.12 to 6.18 (1)

- Linux 6.12
  - **sched_ext** (more later)
- Linux 6.13
  - new `kmem_cache iterator`, allow to get state of the slab allocator.
  - `bpf_send_signal_task()` kfunc to send a signal to another process
  - new BPF map feature that allows threads to share some memory with a BPF scheduler
  - BPF programs can now have a private stack
- Linux 6.14
  - new BPF kfuncs: `bpf_local_irq_save()` and `bpf_local_irq_restore()`
- Linux 6.15
  - new set of BPF callbacks to obtain **timestamps** from various places in the networking stack
  - improved verification of programs with **loops**, new load-acquire and store-release instructions, ability to change extended attributes on files within BPF programs
  - new **resilient queued spinlock** locking primitive

eBPF: from 6.12 to 6.18 (2)

- Linux 6.16
  - BPF can be used to implement **traffic-control queueing disciplines**
- Linux 6.17
  - set of standard string operations for BPF programs
  - standard output and error streams that can be used to communicate back to user space
- Linux 6.18
  - now supports signing BPF programs
  - ten new **kfuncs** added

eBBF: workshop at CdL

- Want to **learn more** about eBPF and build a real-life example of eBPF program in the networking space?
- Workshop: *Master eBPF and XDP: create a high-performance networking ad blocker*
  - This **Sunday** at 10:30 AM
  - Presented by **Bootlin** engineers Alexis Lothoré and Maxime Chevallier

sched_ext: user-definable schedulers (1)

- The kernel has “multiple” schedulers
  - Real-time scheduler for `SCHED_FIFO` and `SCHED_RR` tasks
  - Deadline scheduler for `SCHED_DEADLINE` tasks
  - *Completely Fair Scheduler* for `SCHED_NORMAL` tasks
- The whole scheduler code/logic is “hardcoded” into the kernel
- Desire to have the possibility of writing “custom” scheduler, to support specific workloads, or simply to experiment
- New `SCHED_EXT` scheduling class
  - Sits between the *idle* class and the *completely fair scheduling* class

sched_ext: user-definable schedulers (2)

- Possibility to load an eBPF program that will handle the scheduling of tasks in the `SCHED_EXT` class
  - Only one scheduler sched_ext scheduler for the whole system
  - Will manage **dispatch queues** containing tasks waiting for the CPU
  - Must implement a set of callbacks to react to scheduling related events
- Has been fairly controversial
  - Strong opposition from some scheduler developers/maintainers
  - But ended up being merged by Linus Torvalds

EEVDF scheduler

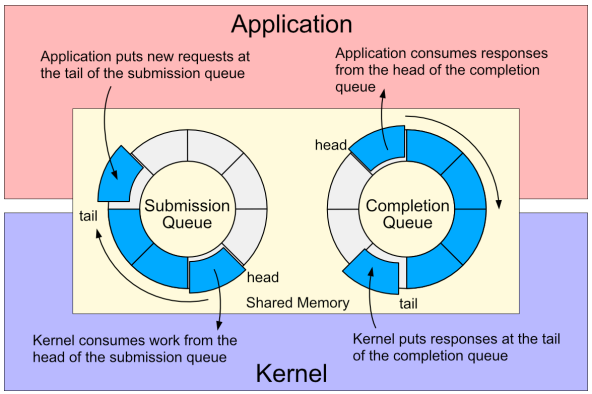
- Since 2.6.23, the scheduler for `SCHED_NORMAL` tasks was CFS, *Completely Fair Scheduler*
- Exhibited some issues for latency-sensitive tasks
  - Attempts to fix it with *latency nice* patches, but a better solution was identified
- EEVDF: *Earliest Eligible Virtual Deadline First*
  - Research paper from 1995 (!)
- EEVDF scheduled introduced in Linux 6.6, completed in Linux 6.12

EEVDF scheduler: main idea

- Divide the available CPU time equally between all the runnable tasks in the system (assuming they have the same priority)
  - 4 tasks at same prio should each get 25% of the CPU time
- Virtual run-time: time representing the allocated share of the CPU time
- Actual run-time: time during which the task really ran
- Difference between the two: the **lag**
- Task eligible if the lag is zero or greater
- Virtual deadline computed: time remaining in the time slice + time it became eligible
  - Task with the earliest virtual deadline is chosen
  - Time slice specified through `sched_attr::sched_runtime`

io_uring

- In high-performance applications, system calls are expensive
  - One system call required for each I/O operation
  - Each system call requires a round trip with the kernel
- **io_uring** is a general asynchronous I/O interface
- Principle
  - Two ring buffers memory-mapped between user-space and kernel-space
  - One **submission queue** (SQ) where user-space submits operations to be done by the kernel
  - One **completion queue** (CQ) where kernel-space returns results to user-space
  - Each command identified by an **opcode**



Picture from https://nick-black.com/dankwiki/index.php/File:Uring_0.png

io_uring: opcodes

IORING_OP_NOP, IORING_OP_READV, IORING_OP_WRITEV, IORING_OP_FSYNC, IORING_OP_READ_FIXED,
IORING_OP_WRITE_FIXED, IORING_OP_POLL_ADD, IORING_OP_POLL_REMOVE, IORING_OP_SYNC_FILE_RANGE,
IORING_OP_SENDMSG, IORING_OP_RECVMSG, IORING_OP_TIMEOUT, IORING_OP_TIMEOUT_REMOVE, IORING_OP_ACCEPT,
IORING_OP_ASYNC_CANCEL, IORING_OP_LINK_TIMEOUT, IORING_OP_CONNECT, IORING_OP_FALLOCATE,
IORING_OP_OPENAT, IORING_OP_CLOSE, IORING_OP_FILES_UPDATE, IORING_OP_STATX, IORING_OP_READ,
IORING_OP_WRITE, IORING_OP_FADVISE, IORING_OP_MADVISE, IORING_OP_SEND, IORING_OP_RECV,
IORING_OP_OPENAT2, IORING_OP_EPOLL_CTL, IORING_OP_SPLICE, IORING_OP_PROVIDE_BUFFERS,
IORING_OP_REMOVE_BUFFERS, IORING_OP_TEE, IORING_OP_SHUTDOWN, IORING_OP_RENAMEAT, IORING_OP_UNLINKAT,
IORING_OP_MKDIRAT, IORING_OP_SYMLINKAT, IORING_OP_LINKAT, IORING_OP_MSG_RING, IORING_OP_FSETXATTR,
IORING_OP_SETXATTR, IORING_OP_FGETXATTR, IORING_OP_GETXATTR, IORING_OP_SOCKET, IORING_OP_URING_CMD,
IORING_OP_SEND_ZC, IORING_OP_SENDMSG_ZC, IORING_OP_READ_MULTISHOT, IORING_OP_WAITID,
IORING_OP_FUTEX_WAIT, IORING_OP_FUTEX_WAKE, IORING_OP_FUTEX_WAITV, IORING_OP_FIXED_FD_INSTALL,
IORING_OP_FTRUNCATE, IORING_OP_BIND, IORING_OP_LISTEN, IORING_OP_RECV_ZC, IORING_OP_EPOLL_WAIT,
IORING_OP_READV_FIXED, IORING_OP_WRITEV_FIXED, IORING_OP_PIPE, IORING_OP_LAST

io_uring: from 6.12 to 6.18

- Linux 6.12: io_uring operations can now have absolute timeouts
- Linux 6.13: new command `IORING_REGISTER_RESIZE_RINGS`
- Linux 6.15: initial support for zero-copy data reception
- Linux 6.15: possible to read epoll events
- Linux 6.15: security hook for io_uring, allowing Linux Security Modules to have more control over io_uring
- Linux 6.16: io_uring can now be used to create pipes

Rust in Linux

- The Linux kernel is implemented in C, with a small bit of assembly in specific places

Rust in Linux

- The Linux kernel is implemented in C, with a small bit of assembly in specific places
- If you've not been living under a rock for the past few years, you're probably aware that **Rust** is a thing
  - And a thing considered for the Linux kernel as well

Rust in Linux

- The Linux kernel is implemented in C, with a small bit of assembly in specific places
- If you've not been living under a rock for the past few years, you're probably aware that **Rust** is a thing
  - And a thing considered for the Linux kernel as well
- Source of **numerous debates** in the Linux kernel community
  - Some strong advocates
  - Some "opponents", or at least skeptical people
  - A big challenge is the **skills** of existing maintainers and developers
  - Linus Torvalds generally supportive of Rust in the Linux kernel

Rust in Linux

- The Linux kernel is implemented in C, with a small bit of assembly in specific places
- If you've not been living under a rock for the past few years, you're probably aware that **Rust** is a thing
  - And a thing considered for the Linux kernel as well
- Source of **numerous debates** in the Linux kernel community
  - Some strong advocates
  - Some "opponents", or at least skeptical people
  - A big challenge is the **skills** of existing maintainers and developers
  - Linus Torvalds generally supportive of Rust in the Linux kernel
- **Clear growth** of the amount of Rust code
  - Linux 6.12: 76 files in Rust
  - Linux 6.18-rc1: 242 files in Rust

Rust in Linux

- The Linux kernel is implemented in C, with a small bit of assembly in specific places
- If you've not been living under a rock for the past few years, you're probably aware that **Rust** is a thing
  - And a thing considered for the Linux kernel as well
- Source of **numerous debates** in the Linux kernel community
  - Some strong advocates
  - Some "opponents", or at least skeptical people
  - A big challenge is the **skills** of existing maintainers and developers
  - Linus Torvalds generally supportive of Rust in the Linux kernel
- **Clear growth** of the amount of Rust code
  - Linux 6.12: 76 files in Rust
  - Linux 6.18-rc1: 242 files in Rust
  - Still not much compared to the 62249 files in C

Rust in Linux: 6.12 to 6.18 (1)

- Doubly-linked **list** and red-back **trees** (Linux 6.12)
- Integration with **security features**: KASAN, Spectre mitigations, kernel control-flow integrity (Linux 6.12)
- Abstractions for some **VFS** data structures and interfaces (Linux 6.13)
- **Trace events** (Linux 6.13)
- **Misc** devices (Linux 6.13)
- New **memory allocator** for Rust (Linux 6.13)
- Bindings for **PCI** buses, **platform** drivers, **Open Firmware**, and more (Linux 6.14)
  - *"We are almost at the 'write a real driver in rust' stage now, depending on what you want to do", Greg KH*

Rust in Linux: 6.12 to 6.18 (2)

- Abstractions for **high-resolution timers** and **coherent DMA** mapping (Linux 6.15)
- Bindings for **configfs** (Linux 6.16)
- **Power management** Rust abstractions: CPU frequency, operating performance points (OPPs), etc. (Linux 6.16)
- Rust abstractions for **core memory-management** operations (Linux 6.16)
- Rust abstractions for the **regulator** subsystem, **firmware** properties, **I/O resources**, and **I/O memory** (Linux 6.17)
- Rust bindings for Linux-kernel-memory-model-compliant **atomic operations**, maple **trees**, **DebugFS** files, manipulating **bitmaps**, many **APIs for drivers** (Linux 6.18)

Rust: actual drivers

- **Nova Core GPU** driver
  - driver for the NVidia GPUs based on GSP, Turing and later GPUs
  - `drivers/gpu/nova-core/`
- **Tyr GPU** driver
  - driver for ARM Mali CSF-based GPUs
  - `drivers/gpu/drm/tyr`
- Rust implementation of **Android Binder**
  - IPC mechanism used in Android
  - `drivers/android/binder/`
- QT2025 **Ethernet PHY** driver
  - `drivers/net/phy/qt2025.rs`
- Asix **Ethernet PHY** driver
  - `drivers/net/phy/ax88796b_rust.rs`

Rust: example code

```
struct PhyQT2025;

#[vtable]
impl Driver for PhyQT2025 {
    const NAME: &'static CStr = c_str!("QT2025 10Gpbs SFP+");
    const PHY_DEVICE_ID: phy::DeviceId = phy::DeviceId::new_with_exact_mask(0x0043a400);

    fn probe(dev: &mut phy::Device) -> Result<()> {
        // Check the hardware revision code.
        // Only 0xb3 works with this driver and firmware.
        let hw_rev = dev.read(C45::new(Mmd::PMAPMD, 0xd001))?;
        if (hw_rev >> 8) != 0xb3 {
            return Err(code::ENODEV);
        }

        // `MICRO_RESETN`: hold the micro-controller in reset while configuring.
        dev.write(C45::new(Mmd::PMAPMD, 0xc300), 0x0000)?;
        [...]
    }

    fn read_status(dev: &mut phy::Device) -> Result<u16> {
        dev.genphy_read_status::<C45>()
    }
}
```

AI in Linux kernel development

- AI is obviously all over the place
- Numerous discussions in the kernel community around usage of AI for kernel development
- Already used for
  - Detecting which commits are fixes and need to be backported to the stable LTS branches
  - In some kernel subsystems, for automated code reviews posted on the mailing list
- Some proposals in flight for
  - coding agent configuration files
  - tagging commits generated with the help of AI using `Co-developed-by: Claude claude-opus-4-20250514` tags
  - not merged yet, no official policy
- Will surely be a hot topic of discussion later this year at the Linux Plumbers Conference

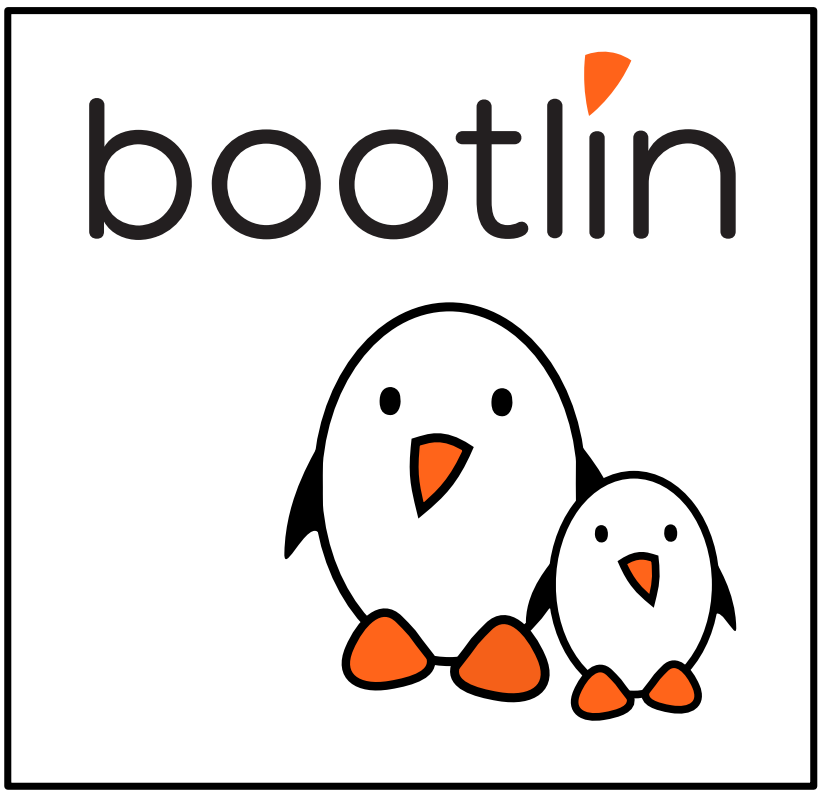
Hardware support: example of Linux 6.18

- **GPIO and pin control**: Nuvoton NCT6694 GPIO controllers, Maxim MAX7360 GPIO and pin controllers, AAeon UP board FPGA pin controllers, NVIDIA Tegra186 pin controllers, Renesas RZ/T2H and RZ/N2H controllers, Broadcom STB pin controllers, Qualcomm Glymur pin controllers, and Qualcomm SDM660 LPASS LPI pin controllers.
- **Graphics**: Solomon SSD2825 MIPI bridges, Waveshare DSI2DPI bridges, Radxa Ra620 bridges, Realtek RTD2171 bridges, MT8189 Chromebook panels, BOE NV140WUM-N64 panels, SHP LQ134Z1 panels for Dell XPS 9345s, Olimex LCD-OLinuXino-5CTS panels, Samsung AMS561RA01 panels, Hydis HV101HD1 panels, Bestar BSD1218-A101KL68 LCD panels, Ampire AMP19201200B5TZQW-T03 panels, EDT ETML0700Z8DHA panels, Mali G710, G510, G310, Gx15, Gx20, and Gx25 GPUs, Rockchip NPU neural processors, Rockchip RK3576 SoCs, Mayqueen Pixpaper eInk displays, T-HEAD TH1520 GPUs, and Arm Mali CSF-based GPUs (driver written in Rust), Qualcomm QCS8300 eDP PHYs.
- **Hardware monitoring**: Kontron SMARC-sAM67 SoCs, GPD device sensors, Monolithic Power Systems MP29502, MP2869, MP29608, MP5998, MP29612, and MP29816 PWM controllers, various ASUS GAMING WIFI motherboard sensors, Texas Instruments INA700 and INA780 power monitors, NXP P3T1750 temperature sensors, Analog Devices sq24905c hotswap controllers, Renesas RAA228244 and RAA228246 PWM controllers, and Sensirion SHT20 and SHT25 humidity and temperature sensors.
- **Industrial I/O**: Tegra 256 GPIO controllers, Loongson-2K0300 GPIO controllers, Amlogic AL113L2 SPI controllers, Atmel SAMA7D65 SPI controllers, and Atmel SAM9x7 SPI controllers, Vishay VEML6046X00 high accuracy RGBIR color sensors, Intel Bay Trail/Cherry Trail Dollar Cove TI PMIC ADC devices, Infineon TLV493D 3D magnetometers, Analog Devices ADE9000 multiphase power monitors, Marvell 88PM886 PMIC analog-digital converters, and Rohm BD79112 analog-digital converters.
- **Input**: Maxim MAX7360 key switch controllers, Awinic AW86927 haptic chips, Hynitron CST816x series controllers, and Himax HX852x(ES) touchscreen controllers.
- **Interrupt controllers**: Aspeed AST2700 SCU interrupt controllers.
- **Media**: OmniVision OV6211 and OG0VE1B sensors.
- **Memory**: AMD 0x1a EDAC memory, AMD VersalNET memory controller, and EDAC on ADM Cortex A72 cores.
- **Networking**: Huawei 3rd gen NICs, SpacemiT K1 Ethernet MACs, and Qualcomm packet-process engines, Skyworks Si3474 power-sourcing equipment, NXP NETC V4 timer-based PTP clocks, TI PRU ICSSM Ethernet ports, and Allwinner sun55i GMAC200 Ethernet controllers, Pensando Ethernet RDMA devices and MediaTek PCIe 5G HP DRMR-H01 networking cards.

Hardware support: example of Linux 6.18

- **Miscellaneous**: Loongson Security Engine, RNG, and TPMs, TI TPS6594 power buttons, TI BQ257xx charger ICs, Lumissil Microsystems IS31FL3236A LED drivers, QNAP MCU status LEDs, Loongson-2K BMC IPMI devices, Nuvoton NCT6694 I2C adapters, Nuvoton NCT6694 socket CANfd controllers, Nuvoton NCT6694 watchdog timers, Nuvoton NCT6694 hardware monitors, Maxim MAX7360 pulse-width modulators, Maxim MAX7360 rotary encoders, Amlogic A4 SPI flash controllers, Intel Bay / Cherry Trail Dollar Cove TI batteries, Analog Devices I3C controllers, Aspeed AST2700 reset controllers, and Qualcomm trusted execution environment controllers, ECB and CBC modes of Texas Instruments DTHE V2 hardware AES engines, Xilinx Versal random-number generators, ThinkPad T14s Gen6 Snapdragon embedded controllers, Acer A1-840 tablets, Loongson-2K100 and 2K0500 NAND controllers, FudanMicro FM25S01A NAND controllers, Realtek RTl93xx switch SoC ECC controllers, STMicroelectronics M2LRxx I2C devices, ADLink PCI-7250, LPCI-7250, and LPCIe-7250 PCI/PCIe boards, Airoha AN8855 switch electronic fuses, SpacemiT K1 PDMA controllers, Sophgo SG2042 SoC PCIe controllers, STMicroelectronics STM32MP25 PCIe controllers, and MediaTek MT8196 SoC GPUEB IPI mailboxes.
- **Power**: Maxim MAX77838 power controllers, NXP PF0900 and PF5300 power controllers, Richtek RT5133 power controllers, SpacemiT P1 power controllers, Amlogic S6/S7/S7D power-domain controllers, IMX i.MX91 power-domain controllers, PXA1908 power-domain controllers, AN7583 SoCs, ipq5424 frequency controllers, MT8196 frequency controllers, and AM62D2 frequency controllers.
- **Regulator**: NXP PF0900/PF0901/PF09XX regulators, NXP PF5300/PF5301/PF5302 regulators, Richtek RT5133 PMIC regulators, Maxim 77838 regulators, and SpacemiT P1 regulators.
- **SoCs**: ESWIN EIC7700 SOCs and Renesas RZ/T2H, RZ/N2H, RZ/T2H, and RZ/N2H SoCs.
- **Sound**: Qualcomm PM4125 codecs, TI PCM1754 digital-to-analog converters, TI TAS2783A audio amplifiers, Realtek RT1321, Shanghai FourSemi FS2104/5S auioo amplifiers, Tascam US-144mkII USB sound devices, and Presonus S1824 USB sound devices, DualSense Playstation controller audio jacks.
- **Thermal**: Renesas RZ/G3S and Renesas RZ/G3E SoCs.
- **Clock**: Allwinner A523/T527 MCU clock control units, Qualcomm ipq5424 APSS clock controllers, Qualcomm Glymur SoC TCSR, global, and display clock controllers, Qualcomm MSM8937 SoC clocks, STMicroelectronics STM32MP21 clocks, MediaTek MT8195 vencsys clocks, MediaTek MT8196 vencsys, vdecsys, ufssys, pextpsys, I2C, mcu, mdpsys, mfg, and disp0 clock controllers, and SpacemiT P1 real-time clocks.
- **USB**: Intel USBIO USB IO-expanders, Renesas RZ/G3E SoC USB host controllers, Maxim MAX14526 MUIC devices, Qualcomm PMIV0104 eUSB2 repeaters, and Sophgo CV18XX/SG200X USB PHYs.

Bootlin contributions



Bootlin contributions: Linux 6.12 to Linux 6.18

- Bootlin contributes to every single release
  - As contributors: patches authored by us
  - And maintainers: patches reviewed/ merged by Bootlin maintainers
- Bootlin in the top 25-30 companies contributing to Linux

Release	Commits	Merges
6.12	118	60
6.13	96	77
6.14	125	60
6.15	145	104
6.16	89	117
6.17	99	94
6.18	71	101

Bootlin contributions: Power over Ethernet support

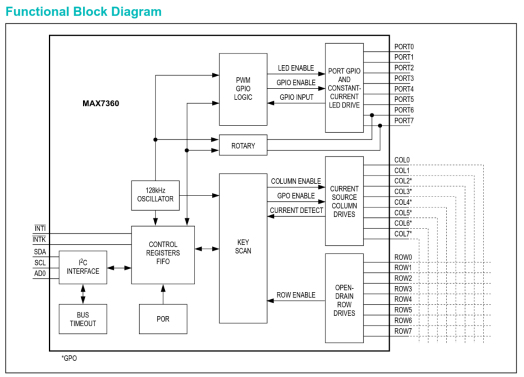
- Standard that allows to provide power to devices through Ethernet cable
- No Linux kernel support was available
  - Vendors were all doing their own custom stuff
- Bootlin was funded by the DENT project from the Linux Foundation to develop and upstream a new Linux kernel subsystem
  - Support Power Sourcing Equipment, like network switches, that **provide** power
  - Initial support landed in Linux 6.10
  - Improved power capabilities in 6.11
  - Port priority and power budget in 6.17
  - Additional improvements in 6.19
  - Work done by Bootlin engineer Köry Maincent

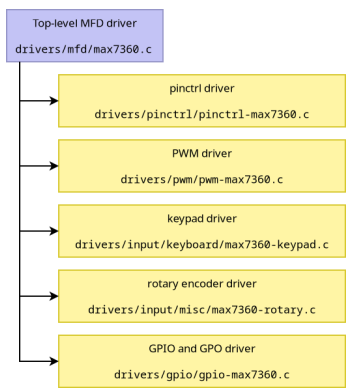
Bootlin contributions: Power over Ethernet support

- Drivers for
  - Texas Instruments TPS23811, by Bootlin
  - Microchip PD69200, by Bootlin
  - Skyworks Si3474, by Piotr Kubik
  - Linear Technology LTC4266, by Kyle Swenson, *not merged yet*
- User-space API provided through *netlink*
- Usable through the standard *ethtool* program
  - Turn power on/off
  - Configure port priority
  - etc.

Bootlin contributions: Maxim MAX7360 driver

- Maxim MAX7360 is a multi-function chip used in some HW designs: matrix keypad, GPIOs/GPOs, rotary encoder, PWMs
- Developed a complete set of Linux kernel drivers to support all features
  - A Multi-Function Driver (MFD) to describe the whole chip
  - A **pinctrl** driver to configure pin functionality
  - A **PWM** driver
  - A **keypad** driver
  - A **rotary** encoder driver
  - A **GPIO/GPO** driver
- Required many iterations, will be part of Linux 6.18
- Work done by Bootlin engineer Mathieu Dubois-Briand

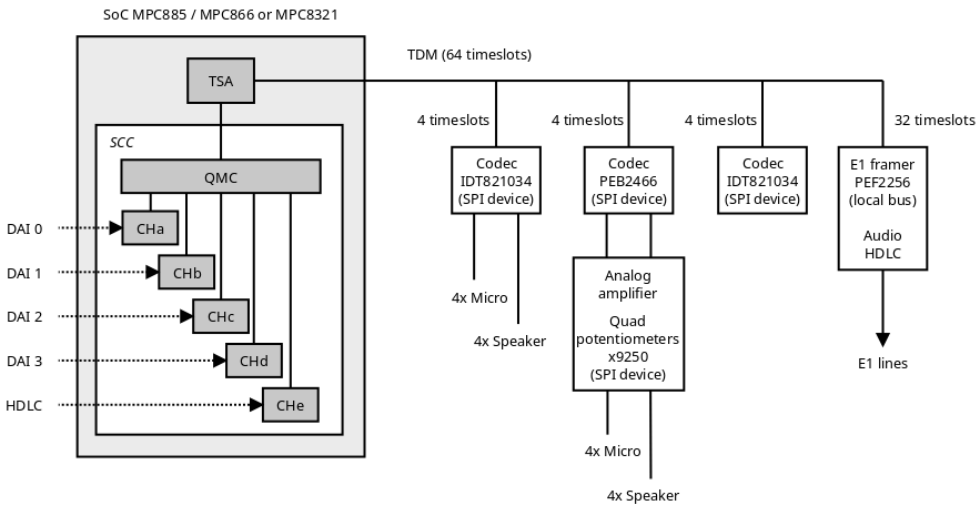




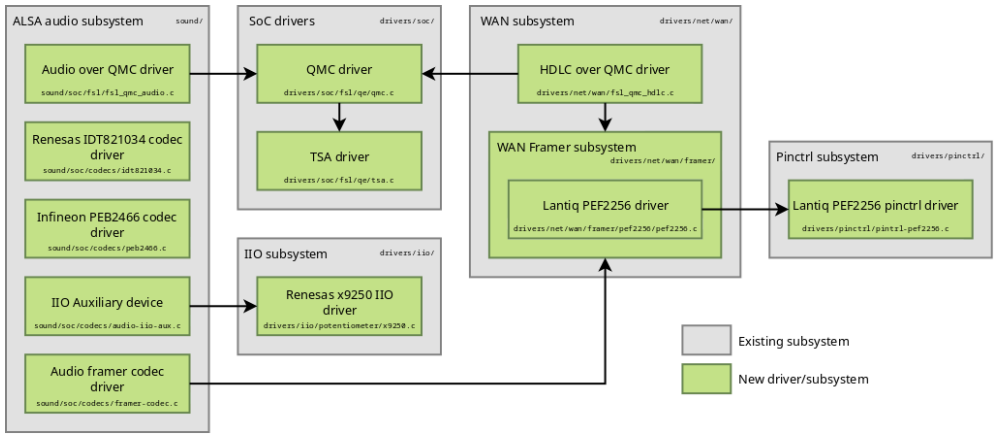
Bootlin contributions: modernizing an old BSP

- French customer with a product in the field of audio communication for Air Traffic Control
- Product in production for 15+ years
- Running on a quite old 32-bit PowerPC platform with 32 MB of RAM (!)
- Need to regularly update the Linux kernel to benefit from security updates
- Large amount of **out-of-tree** code, making updates difficult
- Our mission: upstream the complex audio support
  - Not just sending patches
  - Rewriting everything to use proper kernel subsystems
  - Or even create new ones
- Work done by Bootlin engineer Hervé Codina

Bootlin contributions: modernizing an old BSP



Hardware description



Software support

Bootlin contributions: modernizing an old BSP

- 1 new subsystem: WAN framer subsystem
- 11 new drivers
- 123 patches
- ≈12000 lines of code added
- Worked spanned from Linux 6.6 to Linux 6.12
  - With some fixes afterwards

Conclusion

- A **lot** continues to happen in the kernel world
  - With fascinating things
  - And complicated things
- Keeping tracking of everything is impossible
- But there are some good resources:
  - **LWN.net** clearly being the best
  - KernelNewbies.org is also useful

Questions? Suggestions? Comments?

Thomas Petazzoni, Bootlin