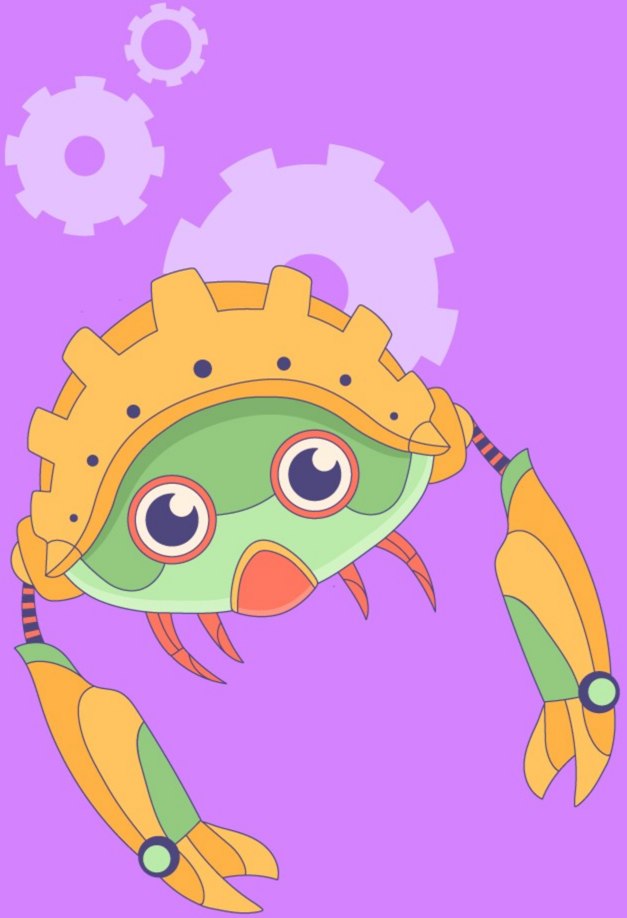
ANDREA RIGHI

Principal System Software Engineer @ NVIDIA

Crafting a Linux kernel scheduler in Rust



Scheduling

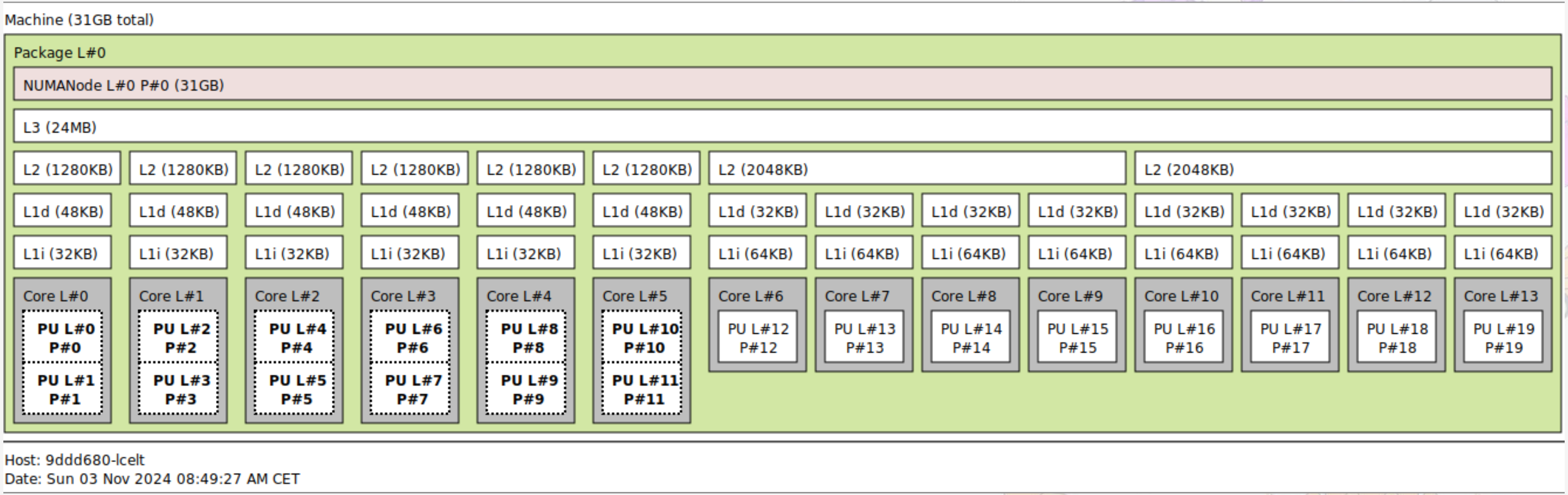
What is a scheduler?

- Kernel component that determines:
  - **Where** each task needs to run
  - **When** each task needs to run
  - **How long** each task needs to run

Why does scheduling matter?

- Performance
  - Workload
  - Topology
- Security
  - Isolation
- Energy Efficiency
  - EAS

CPU topology can be complex...



[From `lstopo` on a Dell Precision 5480 equipped with 13th Gen Intel(R) Core(TM) i7-13800H CPUs]

Scheduling in Linux

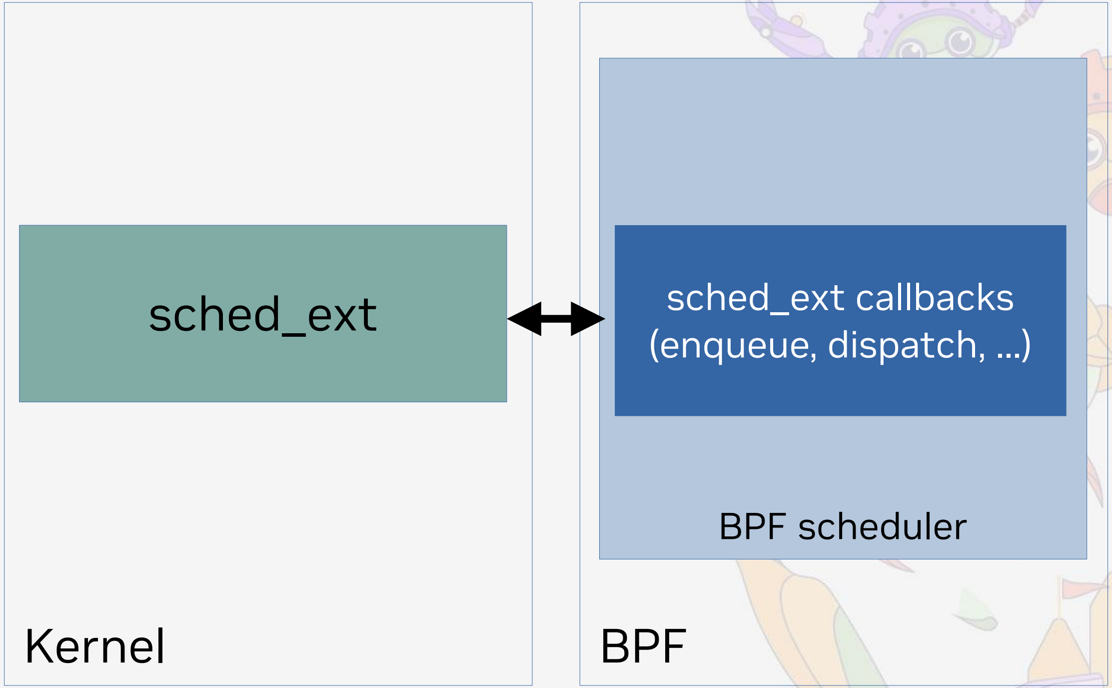
- One scheduler to “rule them all”
  - CFS < v6.6
  - EEVDF >= v6.6
- Really difficult to conduct experiments
- Really difficult to upstream changes

sched_ext

- Implement custom CPU schedulers as loadable BPF programs
- BPF guarantees safety (no kernel panic, memory bugs, ...)
- Watchdog prevents deadlock and starvation
- Available in Linux v6.12

sched_ext

BPF scheduler



sched_ext pros / cons

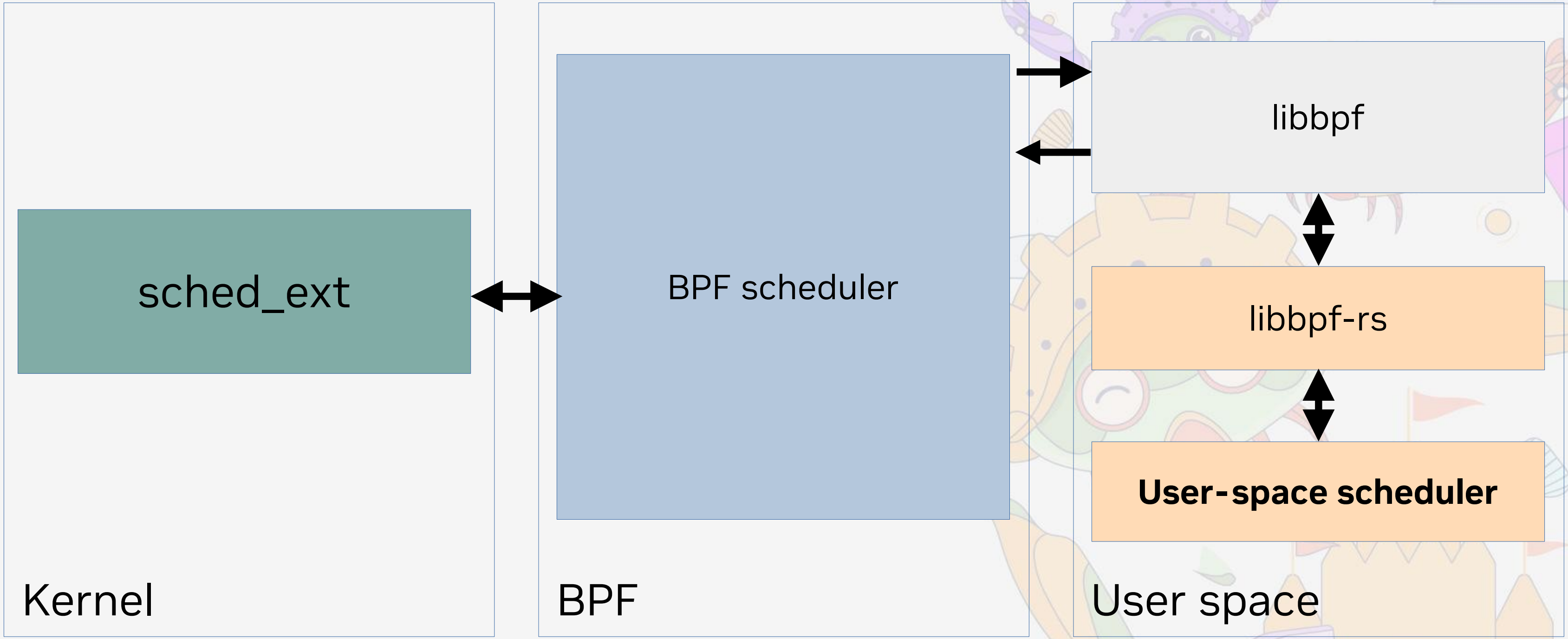
- Pros
  - Ease of experimentation
  - Fast edit/compile/test iteration
  - Safety
- Cons
  - Limited programming model
  - BPF verifier complexity
  - Kernel restrictions (no user-space libs, no floating point, etc.)

User-space scheduling

Idea

- Use BPF + sched_ext to channel scheduling events to user space
- A scheduler becomes a regular user-space process
- Offload complexity to user space
- Access to user-space languages
  - Use **Rust!**

User-space scheduler: design



scx_rustland

- EDF-based scheduler
  - Deadline is evaluated as a function of the task's vruntime and the average amount of **voluntary context switches**
- Tasks use a **variable time slice**
  - Time slice inversely proportional to the amount of tasks waiting to be scheduled

- Is it working?

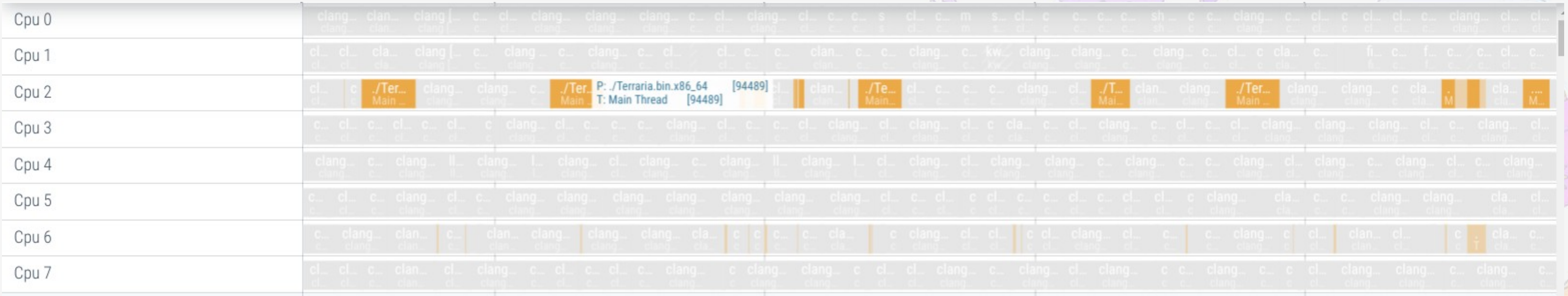
Demo: playing Terraria while building the kernel

EEVDF

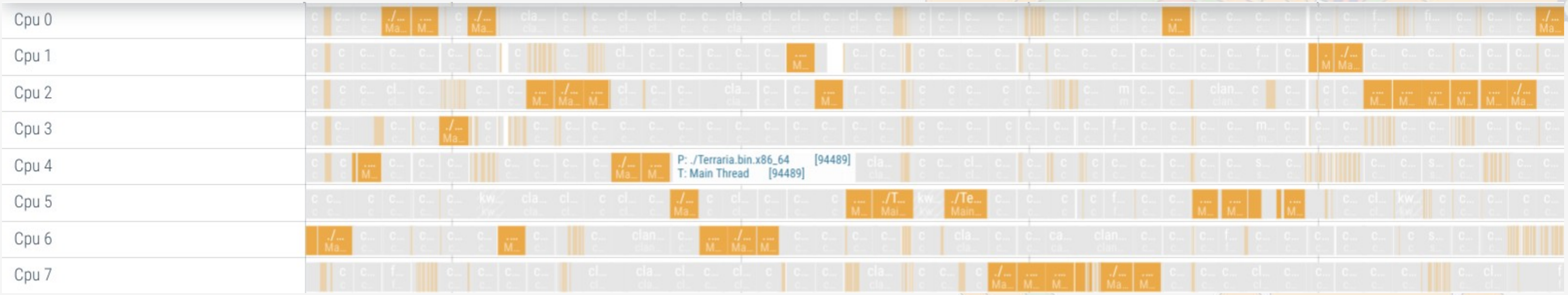
scx_rustland

EEVDF vs scx_rustland - https://perfetto.dev

EEVDF

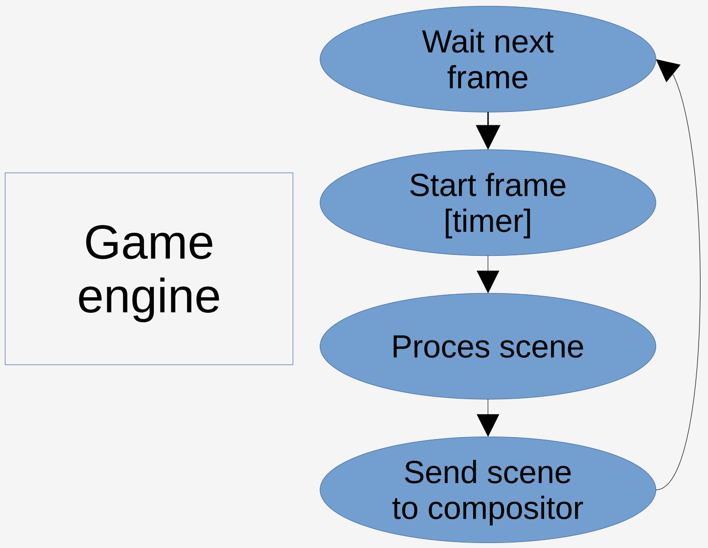


scx_rustland



Why is it better?

- Interactive workloads are typically cyclic (pipeline)
- Tasks release the CPU voluntarily



- Generalize user-space scheduling

scx_rustland_core framework

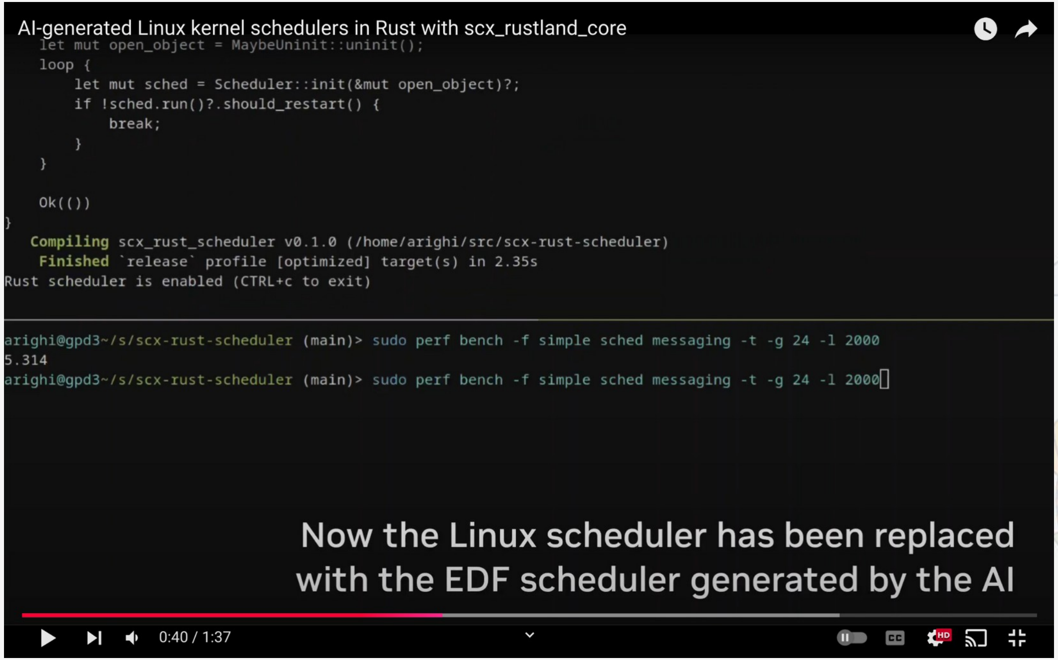
- Abstract scx_rustland backend
- Define generic scheduling API
- Provide a Rust crate (scx_rustland_core)
- Allow to implement Linux schedulers easily as regular Rust projects

FIFO scheduler (using the scx_rustland_core crate)

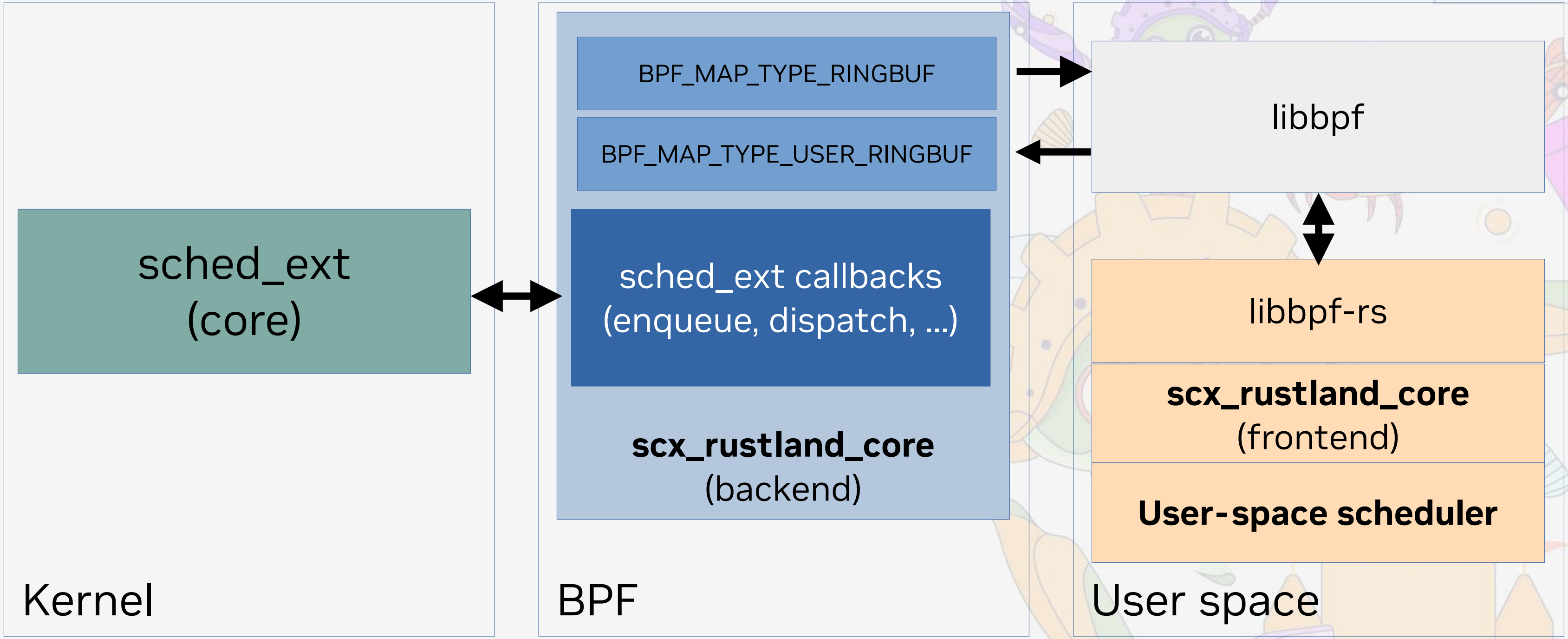
```
fn schedule(&mut self) {
    let nr_waiting = *self.bpf.nr_queued_mut();

    while let Ok(Some(task)) = self.bpf.dequeue_task() {
        let mut dispatched_task = DispatchedTask::new(&task);
        let cpu = self.bpf.select_cpu(task.pid, task.cpu, 0);
        dispatched_task.cpu = if cpu >= 0 { cpu } else { RL_CPU_ANY };
        dispatched_task.slice_ns = SLICE_NS / (nr_waiting + 1);
        self.bpf.dispatch_task(&dispatched_task).unwrap();
    }
    self.bpf.notify_complete(0);
}
```

AI-generated schedulers?



scx_rustland_core: design



Caveats

- User-space scheduler can’t be blocked
  - Page faults are bad
  - Override the Rust allocator via GlobalAlloc
  - Work on a pre-allocated mlock()ed memory arena

Conclusion

Key takeaways

- scx_rustland is not a better scheduler in general
- Rust itself doesn’t make the scheduling better
- Ease of experimentation is the key
  - Fast edit/compile/test cycle
  - Integration with user-space components (Rust crates)

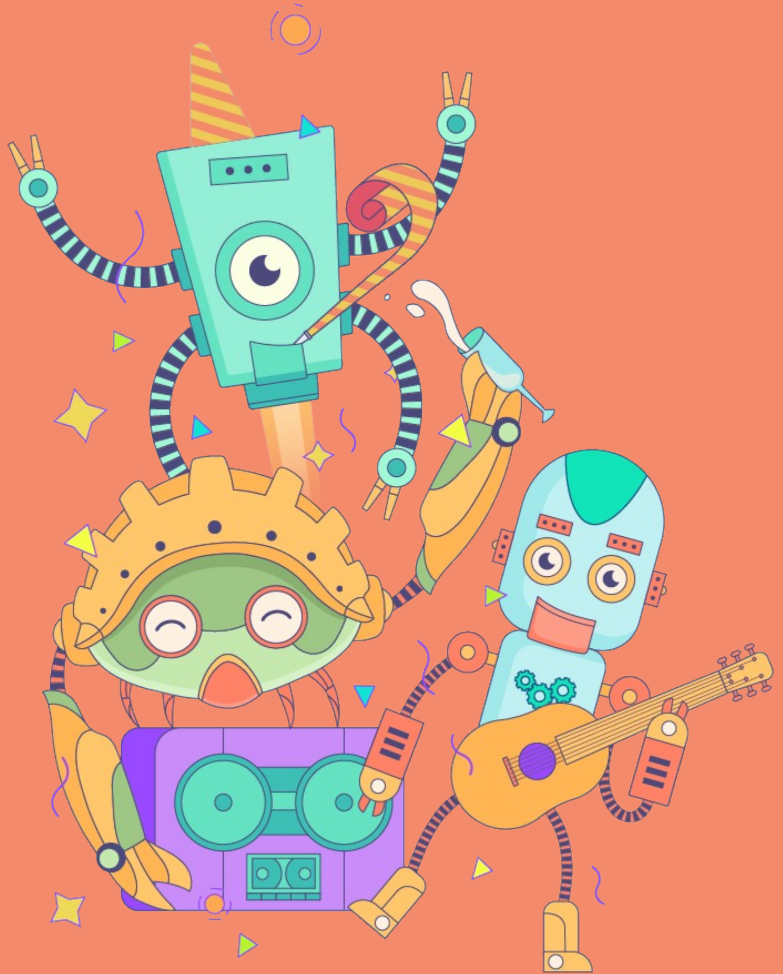
Future idea

- What if we provide a similar technology for other kernel subsystems?
  - Drivers
  - Filesystems
  - …
- Implement more kernel subsystems in Rust (without adding Rust into the kernel)

References

- Main scx project
  - https://github.com/sched_ext/scx
- Rust scheduler template
  - https://github.com/arighi/scx_rust_scheduler
- LWN.net – sched_ext at LPC
  - https://lwn.net/Articles/991205

Questions?



ANDREA RIGHI

arighi@nvidia.com