ANDREA RIGHI

Principal System Software Engineer @ NVIDIA

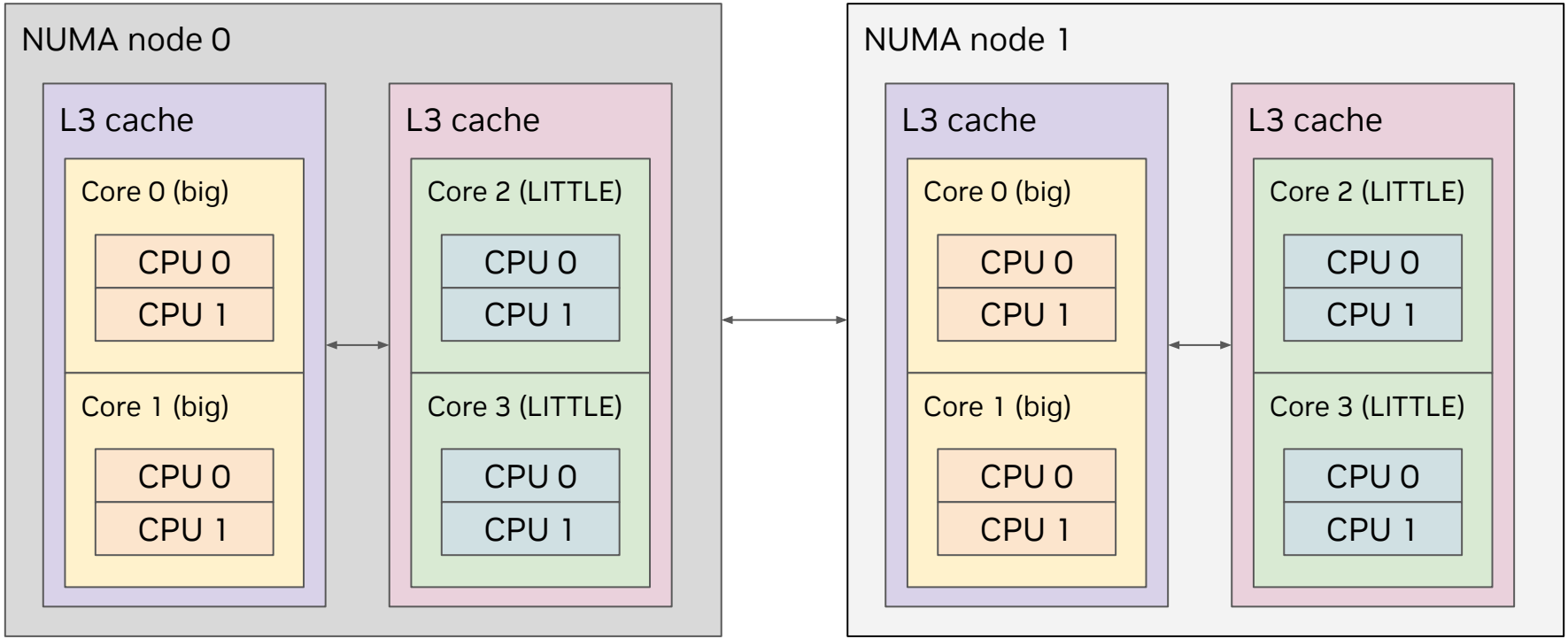
Exploring AI-powered Kernel Schedulers with Rust

Why are we still talking about schedulers?

"The arguments that 'servers' have a different profile than 'desktop' is pure and utter garbage, and is perpetuated by people who don't know what they are talking about."

- Linus Torvalds, Oct 2007

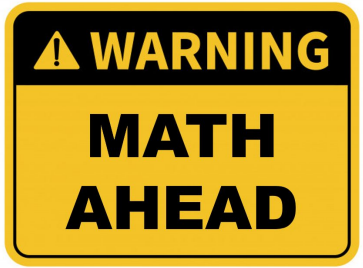
Topology complexity



Workload complexity

- Workload diversity has exploded
  - A decade ago workloads were simple and homogeneous
  - Multi-tenant environments with conflicting requirements (latency vs throughput)
  - Cloud computing, virtualization, AI have introduced new layers of abstraction
  - Power consumption has become a critical bottleneck, especially for large data

Earliest Virtual Deadline First

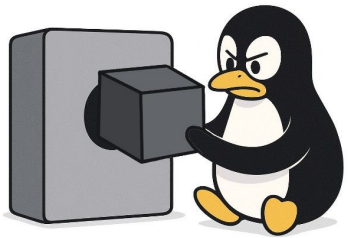


- Time slice
  - How much CPU time a task is going to use
- Virtual deadline
  - [Total runtime + requested time slice] ⇐ All scaled by the task's weight

$$D_T(t) = [R_T(t) + \Delta_T(t)] \times \frac{w_{base}}{w_i}$$

Kernel-space machine learning challenges

- Kernel limitations
  - Language barrier
  - No user-space libraries
  - No floating point
  - Stack size is limited
  - Slow experimentation
  - Bugs are problematic



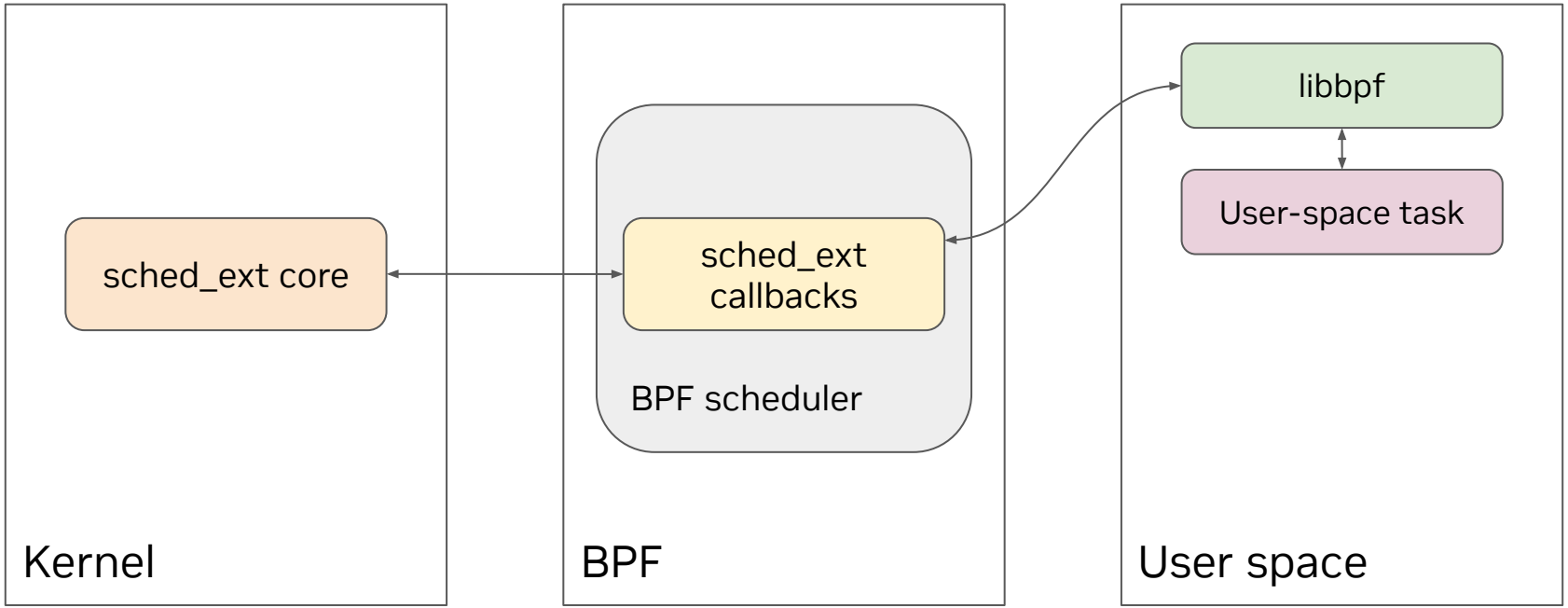
sched_ext: extensible scheduler class

- Technology in the Linux kernel that allows to implement scheduling policies as BPF programs (GPLv2)
- Available since Linux v6.12
- Key features
  - Bespoke scheduling policies
  - Rapid experimentation
  - Safety (can’t crash the kernel)

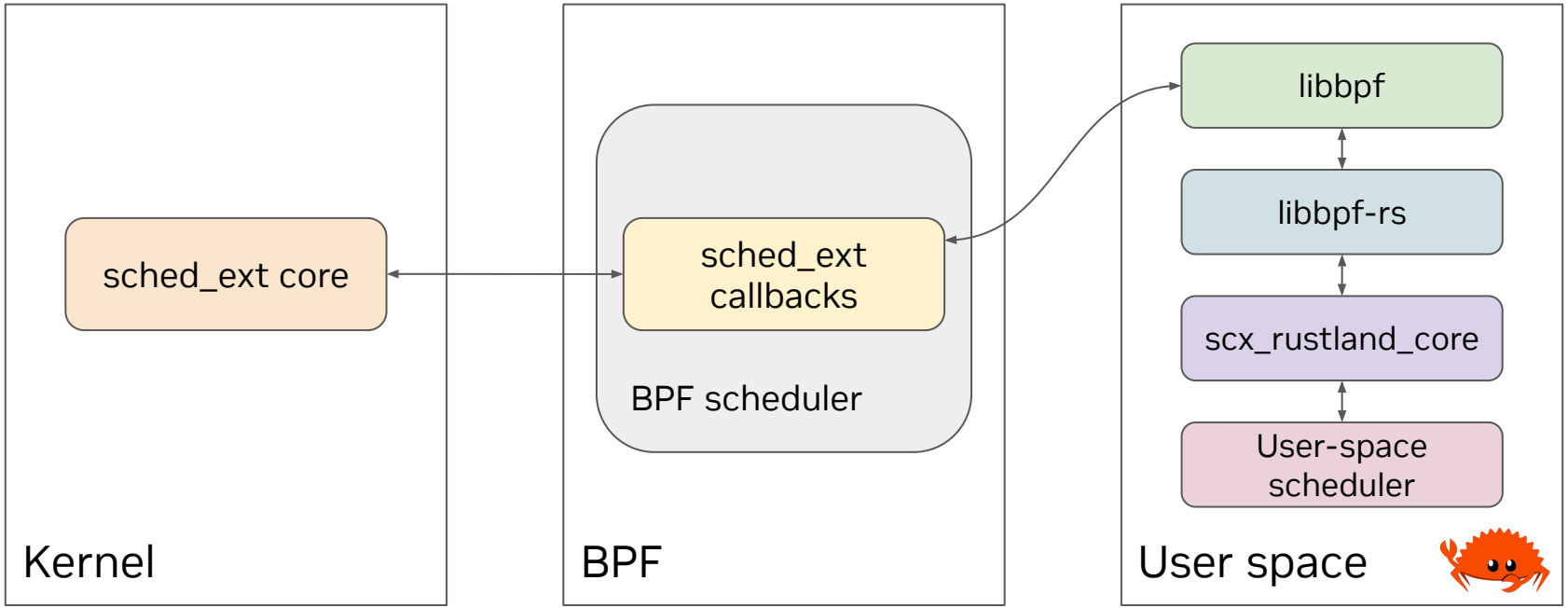
What is BPF?

- The safe way to run kernel code
- VM that lives in the kernel
  - Inject safe sandboxes code into the kernel
  - Attach to kernel functions or events
  - In-kernel JIT compiler
  - Access in-kernel data structures directly without the risk of crashing, hanging, or breaking the kernel

How does it work?



scx_rustland_core: user-space scheduler (in Rust)

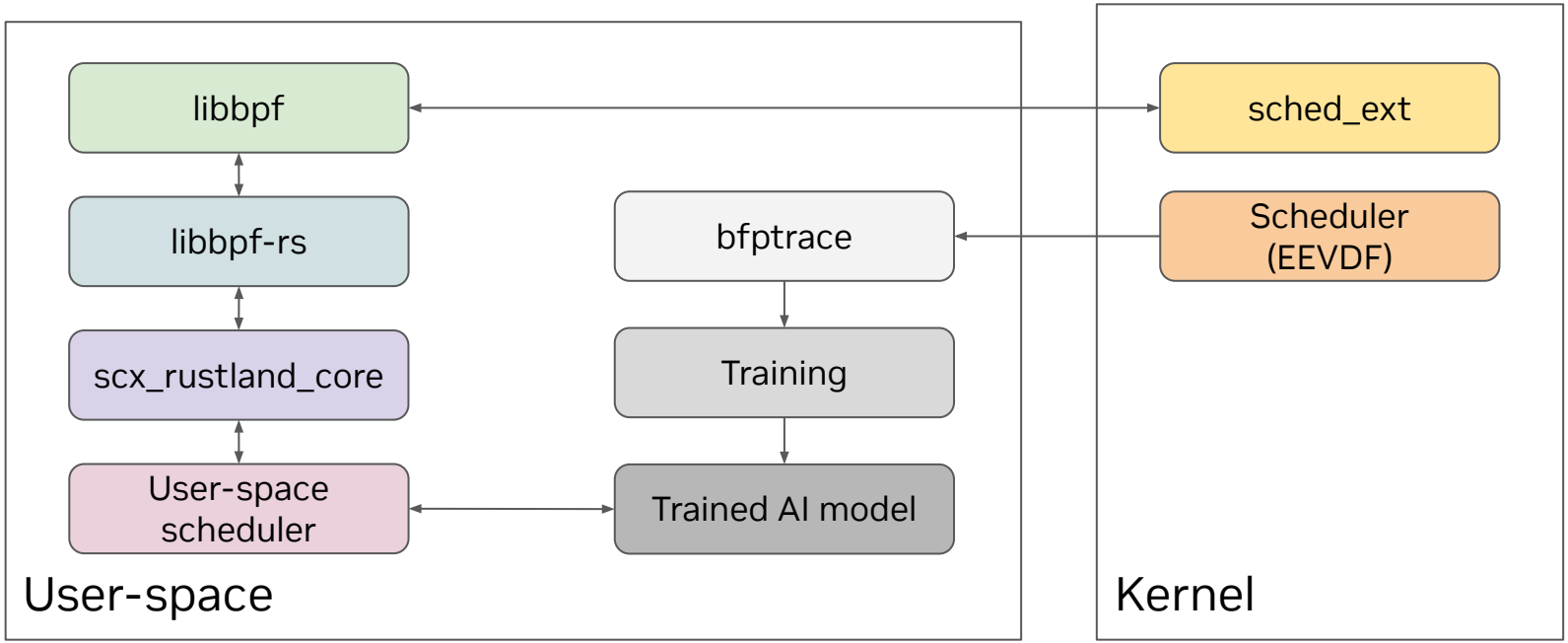


scx_rustland_core: FIFO scheduler

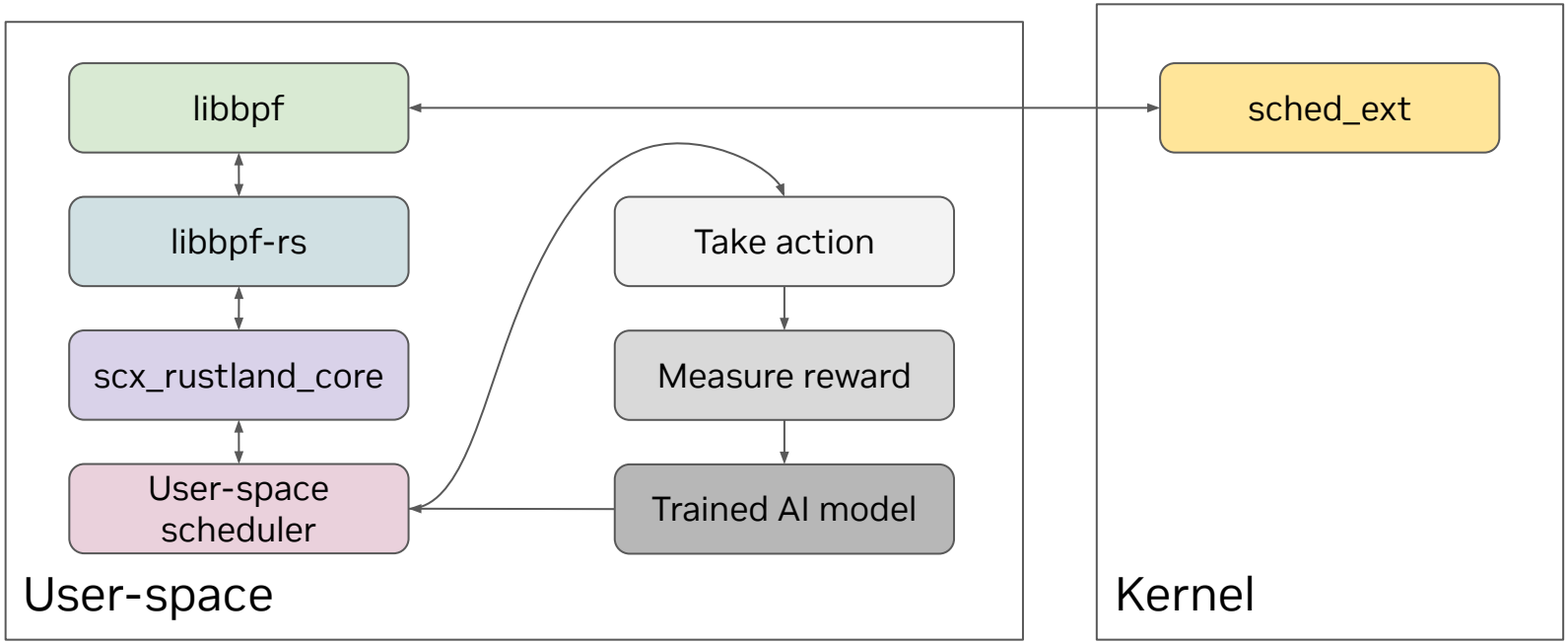
```
fn schedule(&mut self) {
    let nr_waiting = *self.bpf.nr_queued_mut();

    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);
}
```

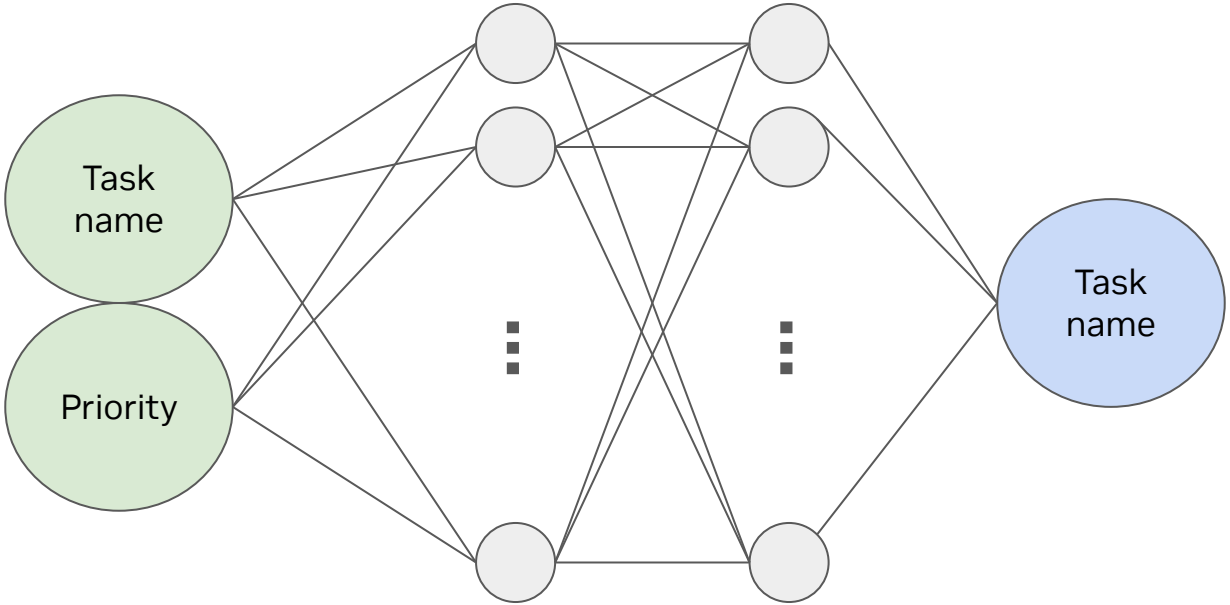
AI-driven scheduler: supervised learning



AI-driven scheduler: reinforcement learning



Time slice prediction neural network

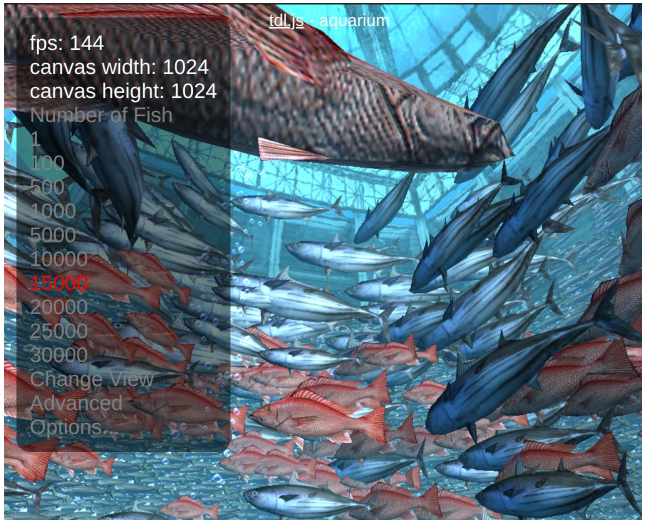


Implementation

- Trace EEVDF
  - sched:sched_switch()
    - Task name
    - Used time slice
- Neural network (multi-layer perceptron)
  - Input
    - Task name (hashed)
    - Task priority
  - Model
    - 4-layer feedforward neural network (ReLU activations, 32 neurons per layer)
    - Loss: Mean Squared Error (MSE)
    - Optimizer: Adam (learning rate = 1e-4)
  - Output
    - Predicted time slice

Demo

- AI model trained to predict the optimal task time slice
- Plug the predictor into scx_rustland
- Measure WebGL performance (fps)
  - WebGL samples - Aquarium



Conclusion

- Humans can design better schedulers than AI
- User-space schedulers present some challenges (non-blocking hot paths)
- It could lead to innovation
  - It’s a new approach to scheduling
  - Can unleash creativity
  - We can find new ways to solve problems once thought impossible to solve

What’s next

- Explore reinforced learning
- Improve supervised learning (more metrics via tracing and profiling)
- Use AI for scheduler components in the cold paths
- Use AI to configure existent sched_ext schedulers

References

- AI-driven scx_rustland: https://github.com/sched-ext/scx/tree/rustland-ai
- Towards Agentic OS: An LLM Agent Framework for Linux Schedulers - Yusheng Zheng, Yanpeng Hu, Wei Zhang, Andi Quinn
- (https://arxiv.org/html/2509.01245v1)
- Improved load balancing with machine learning - Jim Huang (https://lwn.net/Articles/1027096)
- Earliest Eligible Virtual Deadline First : A Flexible
- and Accurate Mechanism for Proportional Share
- Resource Allocation https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=805acf7726282721504c8f00575d91ebfd750564

Questions?

arighi@nvidia.com