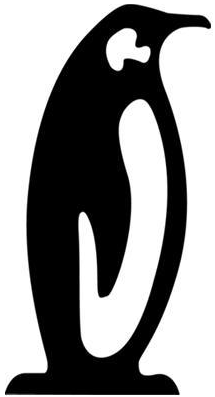


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

From Fragmentation to Integration: Enhancing sched_ext BPF Scheduler Interoperability with Linux

Daniel Hodges

東京 2025 LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

sched_ext User Questions

What workloads is <scheduler> good for?

Is <scheduler> is cache aware?

How does <scheduler> compare to eevdf/cfs?

What is the best scheduler for my hardware?

What is the best scheduler for gaming?

Are there answers?

scx_layered - per workload scheduling policy

general purpose vs specialized (scx_tickless)

What does the user care about (throughput vs latency)?

Do they know what they care about?

What is the system topology?

Is Fragmentation A Problem?

Knowledge silos:

- BPF scheduling ≠ kernel scheduling

Duplicated effort:

- Each scheduler reimplements topology, idle selection, etc.

Missed opportunities:

- Kernel improvements don't automatically benefit BPF schedulers

Innovation barrier:

- Hard to contribute ideas across the boundary

Fragmentation

BPF Scheduler Developers:

- Expertise in specific scheduler (scx_lavd, scx_layered, etc.)
- Custom topology representations (~1k LOC in scx_utils)
- Scheduler-specific patterns and idioms

Kernel Scheduler Developers:

- Deep knowledge of sched_domain, load balancing, topology
- Understanding of decades of optimization and edge cases
- Limited visibility into BPF scheduler innovations

The Cost

New developers learn scheduler-specific quirks, not scheduling fundamentals

Improvements in scx_layered don't help scx_rusty (and vice versa)

BPF scheduler expertise doesn't translate to kernel scheduler contribution

Each new scheduler starts from scratch -> good for simple topology

Goal

Bridge these worlds without stifling innovation

Encourage novel scheduling policies and experiments

But build on shared foundations and common vocabulary

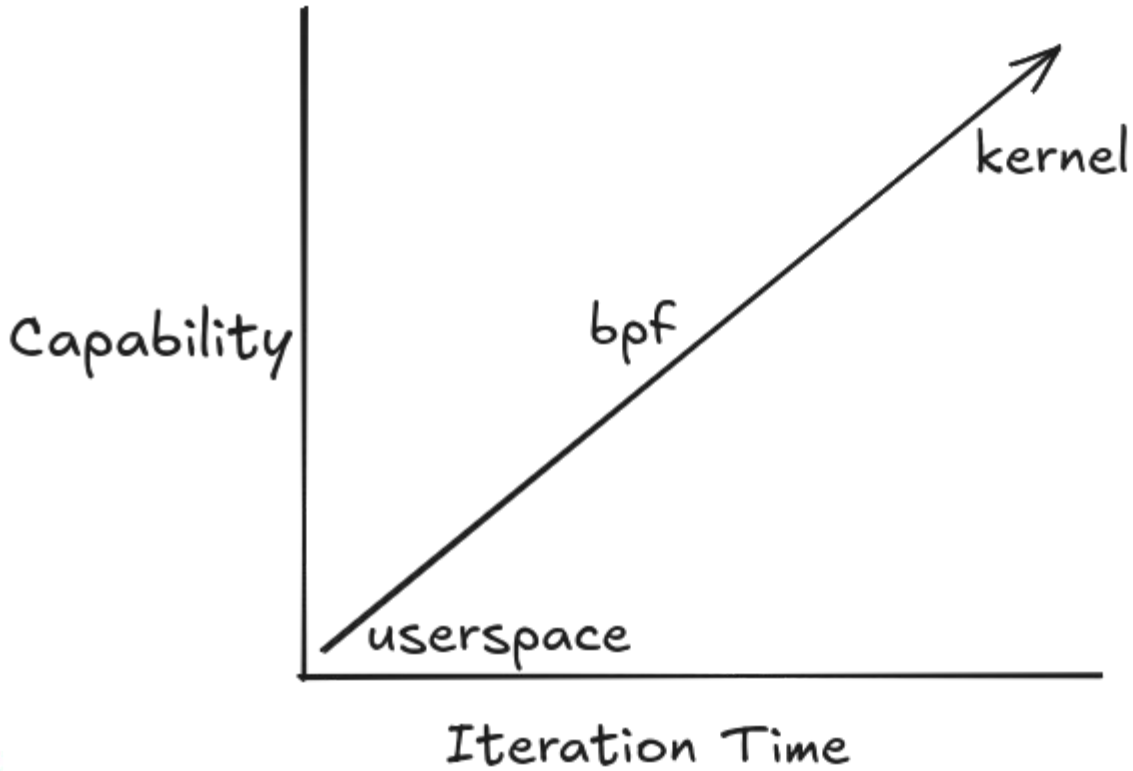
Make BPF scheduler development a path to kernel scheduler understanding

Example: Topology

Problem: Hybrid BPF/userspace schedulers need topology in multiple places

BPF scheduler developers may never see kernel/sched/topology.c

Migration cost/load balancing equally important to scheduling policies in complex topologies


kernel
Capability
bpf
userspace
Iteration Time

Userspace Topology

```
pub struct Topology {
    pub nodes: BTreeMap<usize, Node>,
    pub span: Cpumask,
    pub smt_enabled: bool,
    // Skip indices for fast access
    pub all_llcs: BTreeMap<usize, Llc>,
    pub all_cores: BTreeMap<usize, Core>,
    pub all_cpus: BTreeMap<usize, Cpu>,
}
```

Can we make it better?

```
// Kernel per-CPU variables exposed via __ksym
extern int sd_llc_id __ksym __weak;        // LLC domain ID
extern int sd_llc_size __ksym __weak;      // CPUs in LLC
extern struct sched_domain* sd_llc __ksym;  // LLC domain ptr
extern struct cpufreq_policy* cpufreq_cpu_data __ksym;

// Helper macros generate accessors
DEFINE_PER_CPU_VAL_FUNC(cpu_llc_id, int, sd_llc_id)
DEFINE_PER_CPU_VAL_FUNC(cpu_llc_size, int, sd_llc_size)
DEFINE_THIS_CPU_VAL_FUNC(cpu_llc_id)  // Current CPU
```

Data structure reuse

```
// scx_beerland/src/bpf/main.bpf.c:248
static bool is_cpu_same_llc(s32 this_cpu, s32 that_cpu) {
   return cpu_llc_id(this_cpu) == cpu_llc_id(that_cpu);
}
```

Benefit:

Direct access to kernel topology cache, no userspace required

Bridge to kernel:

Using these accessors exposes developers to kernel topology concepts

What can we do?

Composability/BPF Arenas

- Emil (next talk)

More kfuncs/default implementations

- Idle CPU selection

Scheduler Governors- Steven Rostedt

- Seems like a good long term approach

AI as a Knowledge Bridge

Developer:

"How does the kernel maintain LLC topology?"

AI:

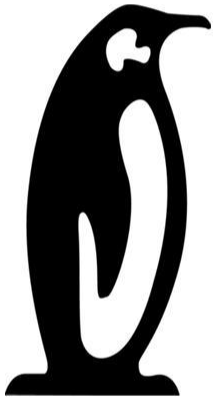
Shows kernel/sched/topology.c, sd_llc per-CPU cache

Shows update_top_cache_domain() and when it's called

Links to sched_domain structure definition

Result:

Hours of exploration condensed to minutes



東京 2025
LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025