東京 2025

# LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

# Scheduler Composability

Emil Tsalapatis

Meta

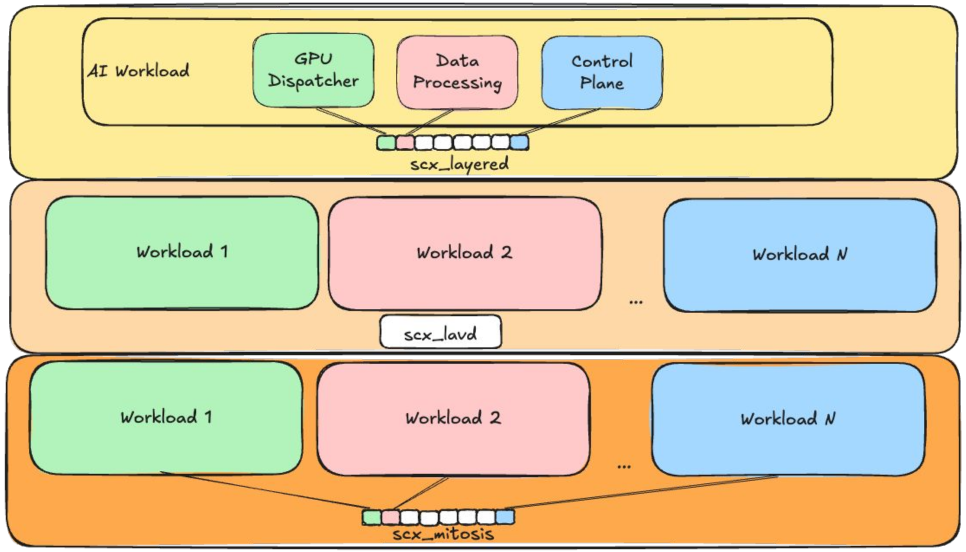
東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

## Theme & Intended Audience

- Managing BPF complexity in sched-ext
  - audience: scheduler writers
- Maintaining a large BPF codebase
  - useful for sched-ext like work
- [**Discussion**] Future Directions

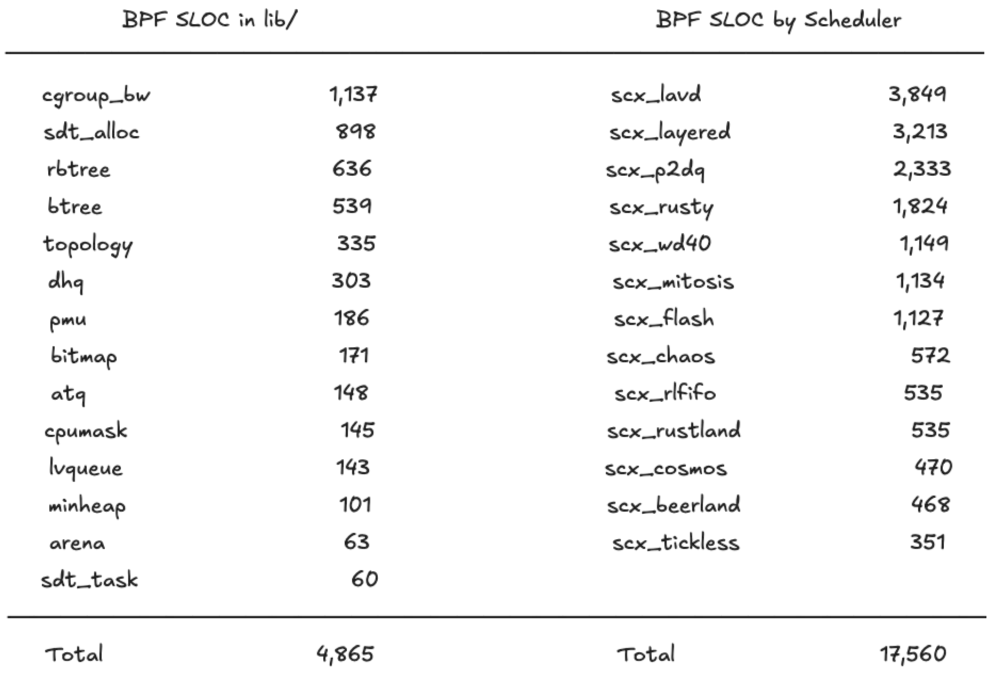
# Scheduler Niches

- scx_layered: Single-workload
    - example: upcoming AI talk
- scx_lavd: Generic default scheduler
- scx_mitosis: Orchestrator integration
- Many, many research deployments
    - performance, reliability, debugging



# Problem: Development Complexity

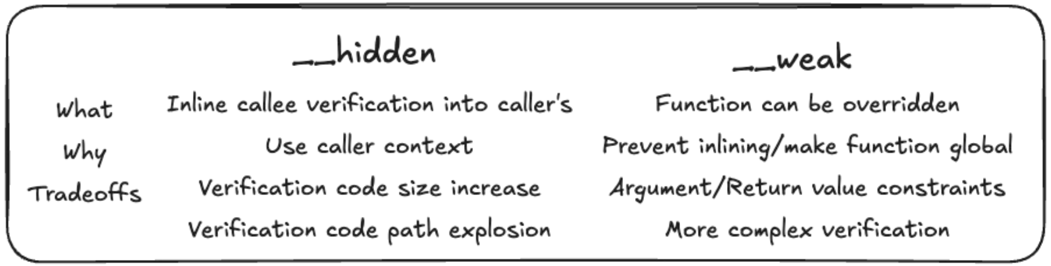
- Mature schedulers are huge
  - up to 4KSLOC of BPF
- Conflicting scheduler abstractions
  - cannot readily combine schedulers
  - e.g., scx_layered + scx_lavd
- BPF C work-intensive



| BPF SLOC in lib/ | | BPF SLOC by Scheduler | |
|---|---|---|---|
| cgroup_bw | 1,137 | scx_lavd | 3,849 |
| sdt_alloc | 898 | scx_layered | 3,213 |
| rbtree | 636 | scx_p2dq | 2,333 |
| btree | 539 | scx_rusty | 1,824 |
| topology | 335 | scx_wd40 | 1,149 |
| dhq | 303 | scx_mitosis | 1,134 |
| pmu | 186 | scx_flash | 1,127 |
| bitmap | 171 | scx_chaos | 572 |
| atq | 148 | scx_rlfifo | 535 |
| cpumask | 145 | scx_rustland | 535 |
| lvqueue | 143 | scx_cosmos | 470 |
| minheap | 101 | scx_beerland | 468 |
| arena | 63 | scx_tickless | 351 |
| sdt_task | 60 | | |
| Total | 4,865 | Total | 17,560 |

# Development in BPF

- Limited programming model
  - no pointer chasing, map-based allocations
- Example: Verifier constraints
  - verification constraints→__hidden
  - instruction count limit→__weak
  - mutually exclusive, require care
- Language toolchain-level problems



| | __hidden | __weak |
|---|---|---|
| What | Inline callee verification into caller's | Function can be overridden |
| Why | Use caller context | Prevent inlining/make function global |
| Tradeoffs | Verification code size increase | Argument/Return value constraints |
| | Verification code path explosion | More complex verification |

## Goal: Composability

- Fancy name for code reuse
- End goal: Stop worrying about BPF
  - focus on scheduling problems
- Remove friction with the tooling
- Code directly in scheduling abstractions

# Composability with BPF Arenas

- Write arbitrary* C code, run in BPF
    - no bounds, type checking
    - allows recursive data structures
    - trees, priority queues, hash tables, strings
- This talk: Details elided
    - focus on how we can use them
- **Opinion:** New abstractions should be directly in BPF
    - As few new kfuncs as possible

* Terms and conditions apply

# What Arenas Can and Can’t Do

- Can implement most things in arenas
  - idle CPU/cpumask selection
  - non-local DSQs
- Arenas can’t hold privileged data
  - pointers are falsifiable, so no `__kptr` fields
  - can’t pass arena pointers to the kernel
- Can’t take privileged locks/write to the kernel

# sched_ext & BPF Arenas: Status

- Used in p2dq and lavd
  - interest for the rest
- Core abstractions: ATQs/Topology/CPU masks
  - lib/ folder
- KASAN library w/ memory allocators
  - LLVM support upstreamed
  - working tree: [https://github.com/etsal/aasan](https://github.com/etsal/aasan)
- Plan: ops.dequeue() fix (compromises ATQs)

## [Topic] Composable Abstractions

- Discuss and converge
- We *already* have common abstractions
  - the sched_ext struct_ops (select_cpu/enqueue/dispatch)
- What do we need for:
  - Capacity Model
  - Power Management
  - Load Balancing

## [Topic] Hierarchical sched_ext abstractions

- Hierarchical scheduling almost here
  - multiple schedulers at once
- How do schedulers communicate?
  - kernel vs pure BPF based
- **Opinion**: IPC should be exclusively BPF-based
  - kernel mechanisms only for enforcement
- Arenas facilitate building those mechanisms

## [Topic] Default sched_ext code in BPF

- Write BPF defaults, preload at boot time
- Reason: Share abstractions with sched_ext repo
- Provide efficient, overridable defaults
- Candidate: Idle CPU placement
  - although Rust-based implementation interesting
- Similarly, BPF-based bypass logic
  - e.g., current one missing load balancing

東京 2025

# LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025