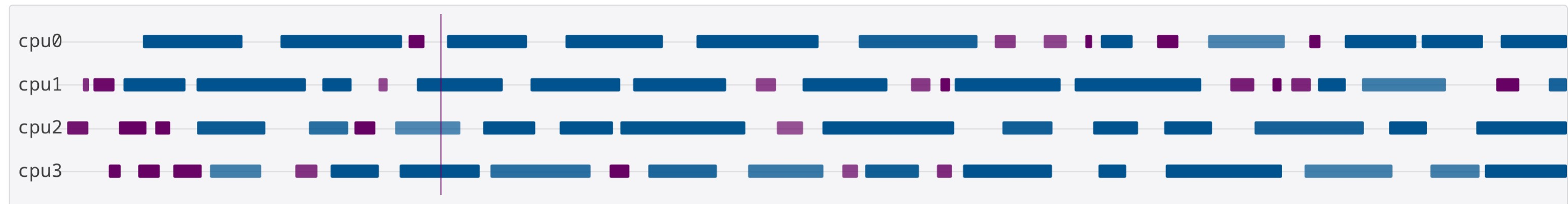


LHP & LWP

Lock **H**older **P**reemption · Lock **W**aiter **P**reemption

Taming latency spikes —
caused by lock holder and lock waiter preemption



Speaker: **Changwoo Min** · Igalia

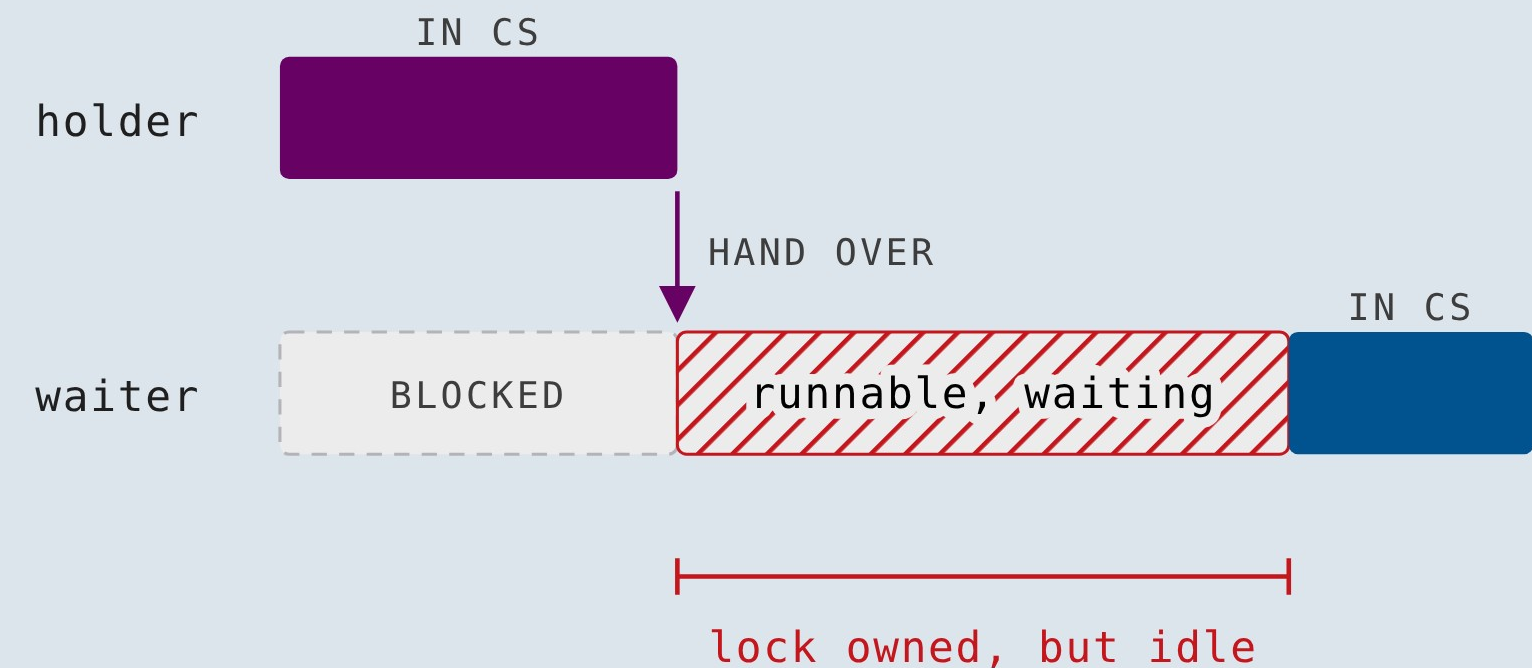
Linux Plumbers Conference 2026 · sched_ext microconference

Two ways a lock stretches a critical section

LHP · holder preempted



LWP · waiter not scheduled



LHP — the task **holding** a lock is preempted; everyone waiting on it waits for the preemption too.

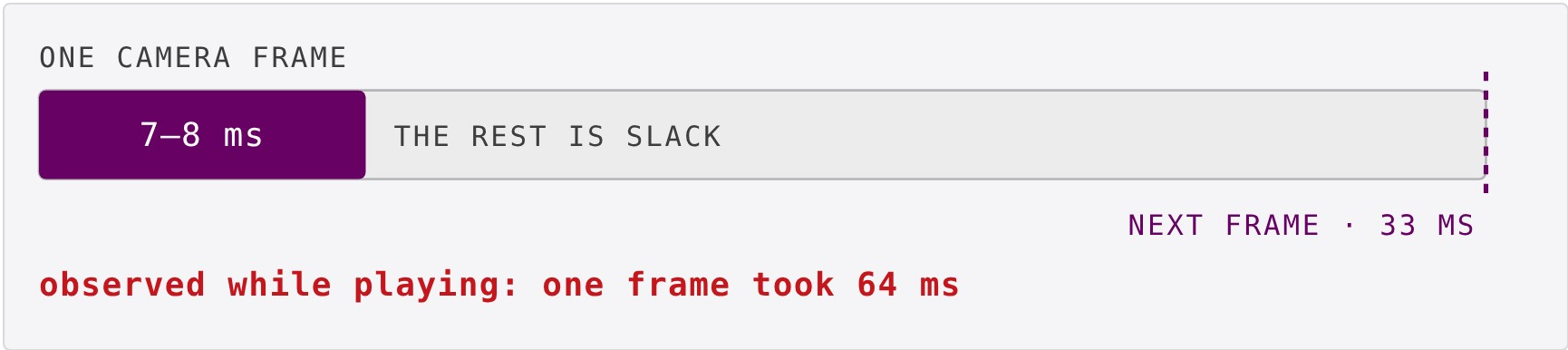
LWP — the lock is handed to a woken **waiter** that is runnable but not running. Every waiter behind it pays the gap again.

Head tracking: one deadline per camera frame

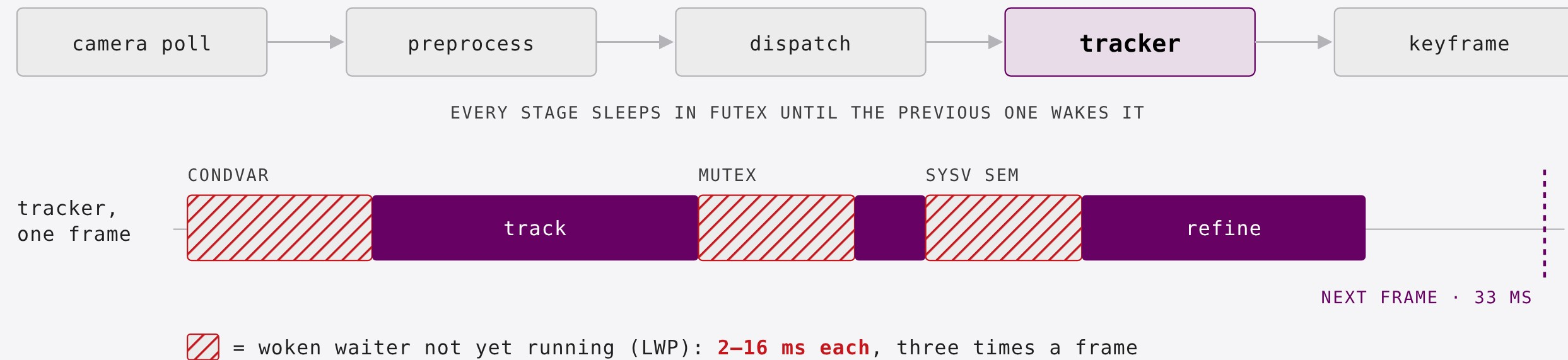


STEAM FRAME · THE HEADSET TRACKS ITSELF FROM ITS OWN CAMERAS
[source: clan.akamai.steamstatic.com/images/clan/45479024/6de9f794...b63.webm](https://clan.akamai.steamstatic.com/images/clan/45479024/6de9f794...b63.webm)

- Every camera frame, the tracker works out **where your head is**; the renderer draws the next image from that pose.
- Cameras run at 30 Hz: a **33 ms** budget for **7–8 ms** of CPU work.
- Miss it and the **camera frame is dropped**: the pose is predicted from older data, and the error shows up in the headset as judder.



A pipeline pays at every hand-over



- Each stage **sleeps in a futex** until the previous stage wakes it.
- Within one frame the tracker is the woken waiter **three times**: a condition variable, a mutex, then a SysV semaphore.
- Each time it was runnable but waited **2–16 ms** before running — LWP. Three in a row blow the 33 ms budget.

The tracker was also **preempted mid-step**, about 1 ms each time — LHP.

LAVD before the fix. EEVDF does worse: see the results (§13).

What LHP and LWP cost an application

	WHAT STRETCHES	GAMES & VR	SERVERS
LHP	the critical section, by the holder's preemption – every waiter waits it out	a frame misses its deadline: stutter, judder	lock convoys: throughput drops, P99 spikes
LWP	the hand-over: the lock is owned but idle until the woken waiter runs – once per hand-over	pipeline stages add up: dropped frames (2–16 ms × 3 per frame)	request pipelines and thread pools: tail latency

Neither shows up in average CPU use. Both show up in the tail.

Three key ideas to mitigate LHP and LWP

LHP.EXTEND

“I hold a lock. Give me a little more time to finish.”

Don't preempt the holder yet.
Opportunistic: act before it hurts.

LHP.DONATE

“I'm blocked on your lock and you aren't running. Run on my slice.”

The waiter's priority drives the holder. *Reactive:* the wait reveals the holder.

LWP.HURRY

“I'm next. When the lock is handed to me, run me now.”

Make the hand-over cheap.

EACH IDEA, IN TWO LANDS	Kernel locks	User-space locks
	mutex, rwsem, semaphore — the kernel can know the holder	futex, SysV semaphore — the kernel sees only the slow path

Time slice extension (TSE)

```
lock(m);  
rseq->slice_ctrl.request = 1; /* holding m */  
barrier();  
critical_section();  
barrier();  
rseq->slice_ctrl.request = 0;  
unlock(m);  
if (rseq->slice_ctrl.granted)  
    rseq_slice_yield();
```

highlighted = added for TSE

7.0

LHP.EXTEND

USER

OPT-IN

What

If an interrupt would preempt a thread with `request` set, the kernel returns to it instead and grants **5 μ s** (up to 50 μ s). An hrtimer enforces it; the thread hands the CPU back with `rseq_slice_yield()`.

Why it's good

Exact — the thread declares it. Covers **every** holder, pure spinlocks included. Tightly bounded, below any scheduling class.

Limits

Targets only very short user-space critical sections. Any syscall ends the grant. Requires library-level changes (glibc does not support it yet).

Proxy execution



```

rq->donor = D · policy, slice
rq->curr = O · the code that runs
  
```

6.17+

LHP.DONATE

KERNEL

What

A mutex waiter stays on the runqueue. When it's picked, the kernel walks `blocked_on → owner` and runs the owner on the waiter's scheduling context; a remote owner pulls the donor over.

Why it's good

Priority inheritance for every class, fair included. The owner is known exactly. No application change.

Limits

`struct mutex` only. A sleeping owner can't be boosted (pending). **EXPERT**, no RT. `sched_ext` only from 7.4, opt-in. User futexes: an RFC.

not yet in mainline · lore: [sched_ext proxy execution v14 \(7.4\)](#) · [sleeping-owner handling v32](#) · [FUTEX_PING RFC](#)

LAVD: TSE's idea, for contended futexes

	TSE	LAVD Futex LHP
holder found by	the app declares it	inferred: <code>futex_wait()</code> returned 0
covers	any marked section	contended futexes
user-space change	required	none
extension	5–50 μs	up to one tick
ends at	syscall or timer	<code>futex_wake()</code> or one slice

How it protects the holder

- never chosen as a preemption victim
- one extra dispatch when its slice runs out
- a one-shot latency-criticality boost next time

Misses fast-path holders and pure spinlocks; condvars and idle pools are false positives.

The map, and its holes

	KERNEL LOCKS	USER LOCKS
LHP · opportunistic	A only lazy preemption – not lock-aware	TSE (opt-in) · LAVD futex LHP (contended)
LHP · reactive	B proxy execution, mutex only – sched_ext needs a donor policy	PI futex · FUTEX_PING (RFC)
LWP	C nothing	D nothing

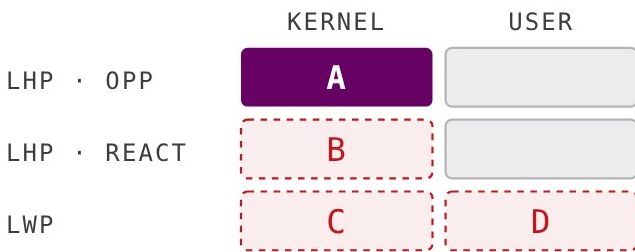
EXISTS

PARTIAL

MISSING

not yet in mainline · lore: [sched_ext proxy execution v14 \(7.4\)](#) · [FUTEX_PING RFC](#)

Protect holders of hand-picked locks



```
/* kernel: a lock opts in at init (RFC) */
mutex_init(&cgroup_mutex);
mutex_track_holder(&cgroup_mutex);
/* acquire: current->nr_tracked_locks++
   release: current->nr_tracked_locks-- */

/* scheduler: one more chance per run */
void dispatch(cpu, prev)
  if (!prev->granted &&
      prev->nr_tracked_locks > 0) {
    prev->granted = true;
    extend(prev); return; } /* ≤ 1 tick */
```

Holder tracking

Each task counts the opted-in mutexes and rwsems (readers too) it holds; BPF reads `nr_tracked_locks`.

Curated locks

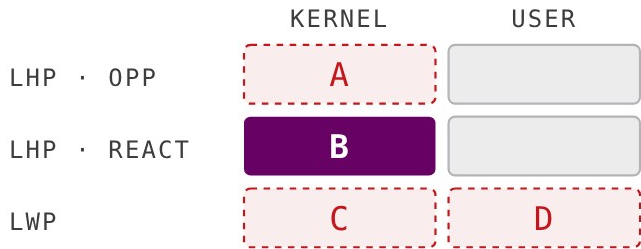
Only hand-picked locks opt in — today the cgroupfs locks. We take that set as the critical ones; waiters of every other lock are marked at `contention_begin` (§12).

Limitations

- **Untracked locks:** no holder signal, only their waiters.
- **Contended or not:** every holder gets one bounded extension.

not yet in mainline · lore: [opt-in tracking of sleeping lock holders](#)

Let the donor drive the owner



```
/* SCX_OPS_ENQ_BLOCKED (7.4) */
void enqueue(p, enq_flags)
{
    if (enq_flags & SCX_ENQ_BLOCKED) {
        o = owner(p->blocked_on);
        cpu = runnable(o) && allowed(p, cpu_of(o))
            ? cpu_of(o) : prev_cpu(p);
        boost_lat_cri(p);
        insert(p, LOCAL_ON | cpu, PREEMPT);
    }
}
```

What the core does

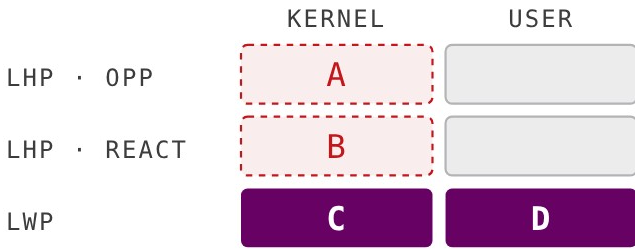
A blocked donor reaches `ops.enqueue()` with `SCX_ENQ_BLOCKED`. When picked, the core runs the owner on the donor's slice; a remote owner migrates the donor and enqueues it again.

Our policy

Treat a donor as urgent, and place it on the owner's CPU **up front** — skip the migration round trip.

not yet in mainline · lore: [sched_ext_proxy_execution v14 \(7.4\)](#)

Hurry the next owner



```
/* tag at entry: the wakeup enqueue must see it */
SEC("fentry/__futex_wait")
    set_flag(current, LOCK_WAITER);

SEC("tp_btf/contention_begin")
    if (!(flags & LCB_F_SPIN)) /* will sleep */
        set_flag(current, LOCK_WAITER);

void enqueue(p, enq_flags)
    if (has_flag(p, LOCK_WAITER)) {
        boost_lat_cri(p); /* earlier deadline */
    }
    /* ... place p as usual ... */
    clear_flag(p, LOCK_WAITER); /* one-shot */
```

LOCK	WHERE THE WAITER IS TAGGED	
futex D	__futex_wait , futex_wait_multiple , futex_lock_pi	
SysV sem D	__do_semtimedop , sem_op < 0	
kernel C	contention_begin , not LCB_F_SPIN	
mutex C	SCX_ENQ_BLOCKED (7.4)	

D is implemented in LAVD; C is proposed. A waiter must block, so it is always visible.

Limitations

Any wakeup of a tagged waiter is boosted — timeouts and signals too (next: tag from the waker side). The fast path stays invisible: no waiter, nothing to boost.

not yet in mainline · lore: [sched_ext_proxy_execution v14 \(7.4\)](#)

Before and after

Before · LAVD without LWP

2–16 ms

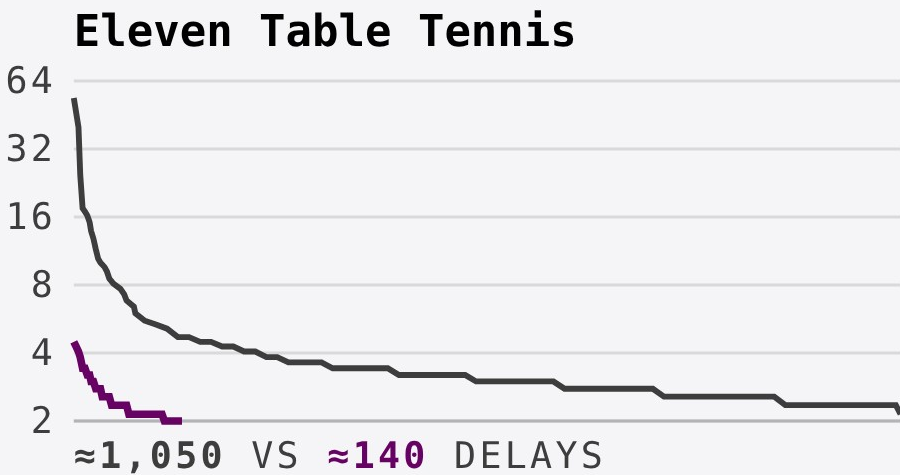
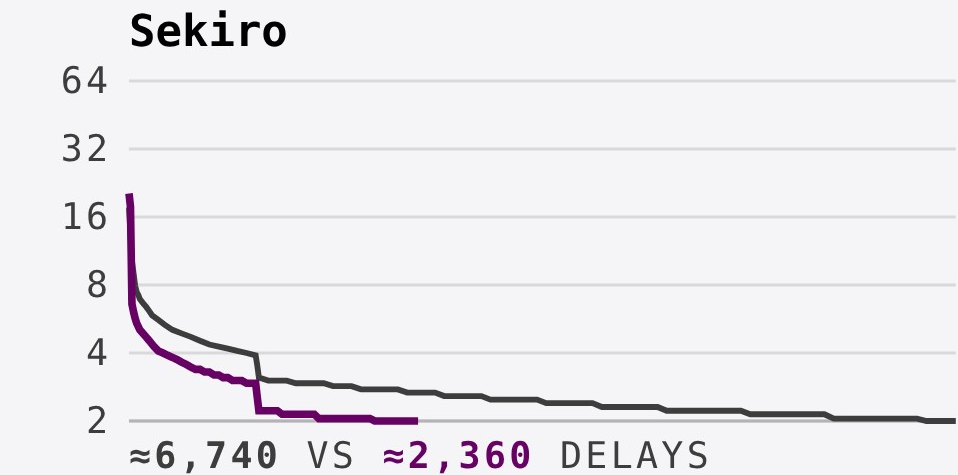
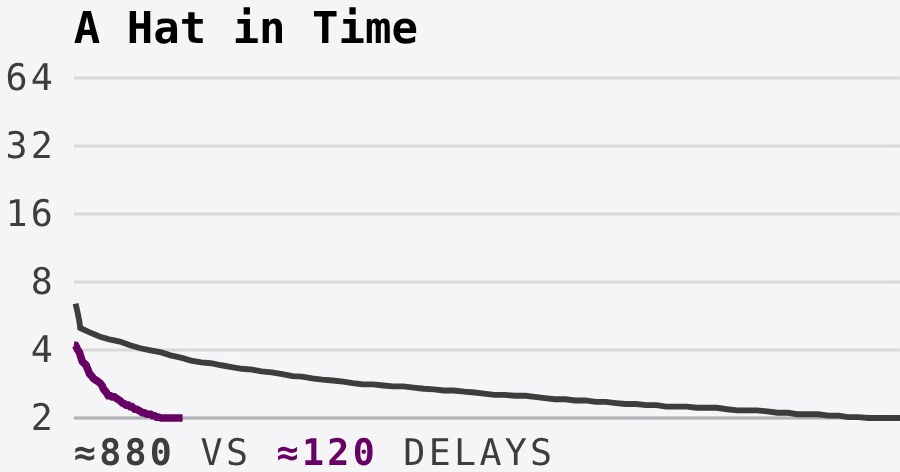
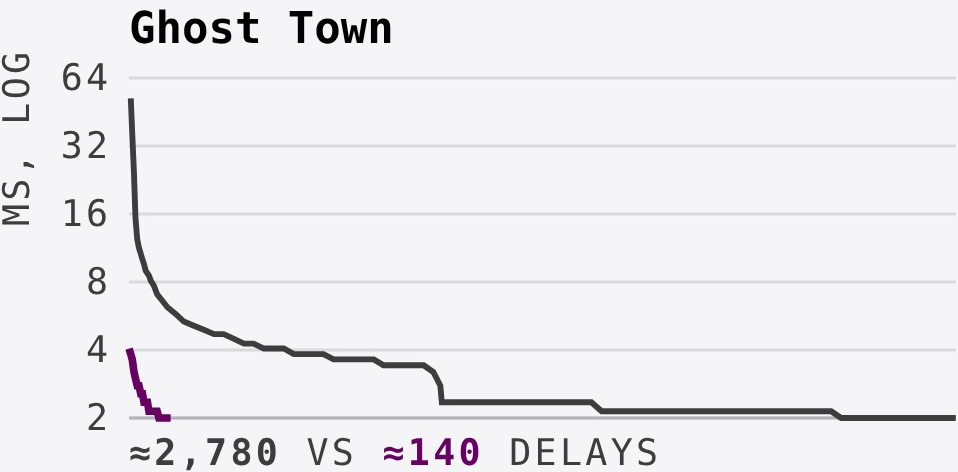
PER LOCK HAND-OVER

64 ms

WORST TRACKER FRAME

After: no large tracker spikes in an hour of play.

SCHEDULING DELAYS > 2 MS IN 60 S, RANKED · EEVDF VS LAVD WITH LWP



Open questions

1. What does the scheduler need to know?

	KERNEL LOCKS	USER LOCKS
holds	<code>task->nr_locks_held</code>	<code>rseq->nr_locks_held</code>
waits	<code>task->b_waiting</code>	<code>rseq->b_waiting</code>
extended	<code>task->b_granted</code>	<code>rseq->b_granted</code>

2. How do we collect it?

KERNEL LOCKS	
Change the sleeping-lock code	all locks · kernel change
Holder tracking (RFC)	curated locks only
Existing tracepoints: <code>contention_begin/end</code> , <code>contended_release</code>	no kernel change · imprecise accounting
USER LOCKS	
libc shares holder/waiter state through rseq	precise · libc change
Trace lock-related syscalls	misses the fast path

Questions?

LHP protects the task holding the lock. LWP hurries the task it's handed to — and that one was missing.

PROJECT `github.com/sched-ext/scx`
SCHEDULER `scheds/rust/scx_lavd`
SPEAKER Changwoo Min · Igalia



github.com/sched-ext/scx

JOIN US

Igalia is a **worker-owned cooperative** — remote-friendly and consensus-run. **We are hiring.**

igalia.com/jobs



Linux Plumbers Conference 2026 · sched_ext MC