

Stickiness

Keeping tasks cache-warm —
in work-conserving LAVD scheduler



slides · gavinguo.cc/lavd_sticky_lpc2026.html



Speaker: **Gavin Guo** · Igalia
gavinguo@igalia.com



7 October 2026

Work conservation picks the idle core

- **Work-conserving** schedulers prefer running tasks on idle CPUs immediately, ensuring no processing capacity is wasted while work is waiting.
- In lavd's `select_cpu`, when a task wakes, the scheduler does its best to find an **idle** core.
- That ranks idle cores above cache-warm ones: more migrations, tasks turned **cache-cold**, L1/L2 caches and TLBs **refilled** again and again.

Some workloads pay for every migration

- The work-conserving mechanism works well in many scenarios. But there are **cache-sensitive workloads**, such as edge routers using routing tables as in-memory KV stores.
- A migration to a cold CPU has a **global** impact: the refill evicts other cache lines and pulls new ones in, making cache contention worse and pushing out the tail latency.

Two kinds of stickiness

Intra-domain

Stay on the previous CPU, or take an idle one in the same domain?

- scx_lavd: Modulate CPU Stickiness using Task Warmth by David Dai, Meta
- [PATCH v2 0/3] Introduce SIS_CACHE to choose previous CPU during task wakeup by Chen Yu, Intel
- scx_lavd: Add L2-sticky redistribution in cpdom DSQ dispatch by Gavin Guo, Igalia

Inter-domain

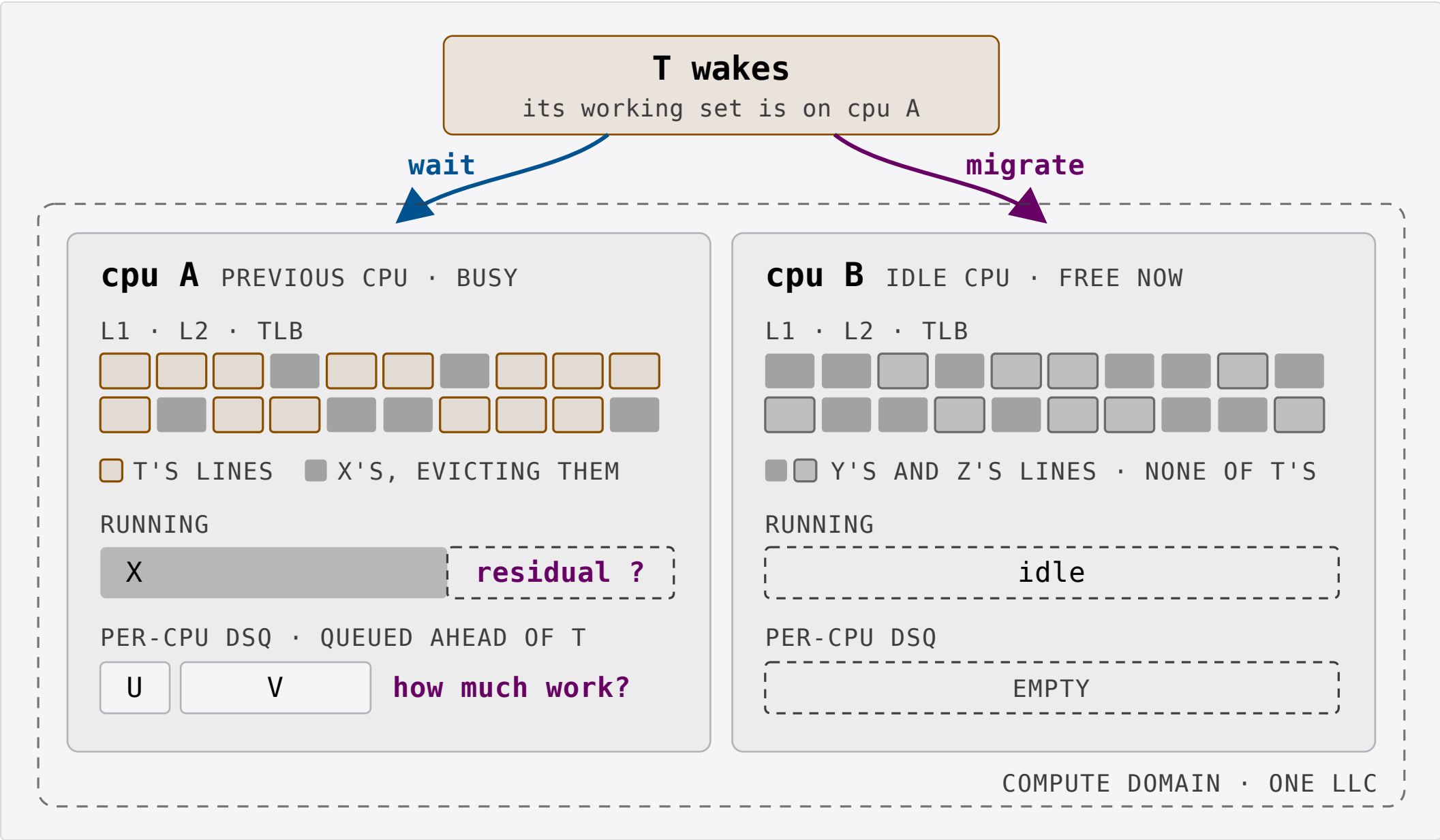
Stay in this LLC, or wake in a less loaded one?

- queue load
- latency time

This talk focuses on the first. If time permits, we also go through the second — inter-LLC domain migration.

Warm but taken, or cold but free?

T wakes. Its cache is on cpu A, which is busy running X. Cpu B is idle and holds nothing of T. Work conservation picks B every time.



THE DECISION

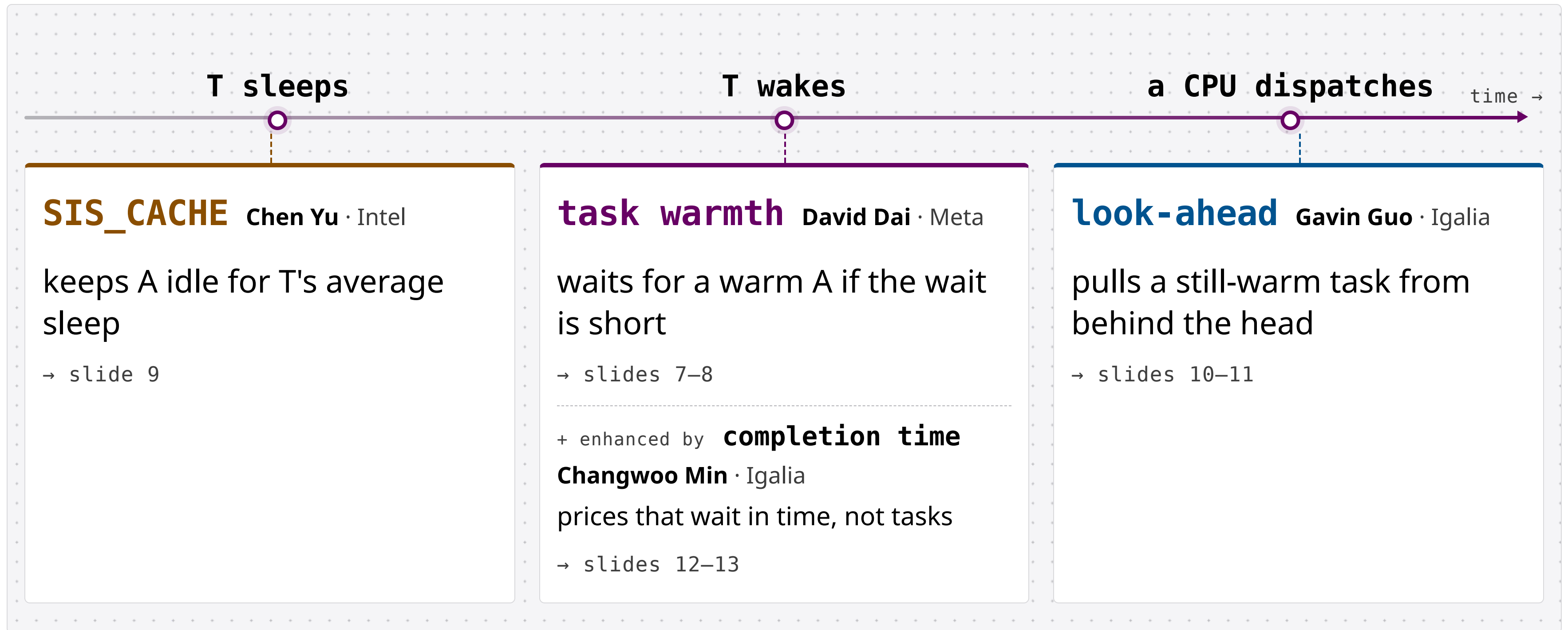
wait for A if
wait(A) < refill(B)

wait(A) X's residual, plus the work queued ahead of T

refill(B) what T still has on A, fetched again — plus the per-CPU userspace state it leaves behind

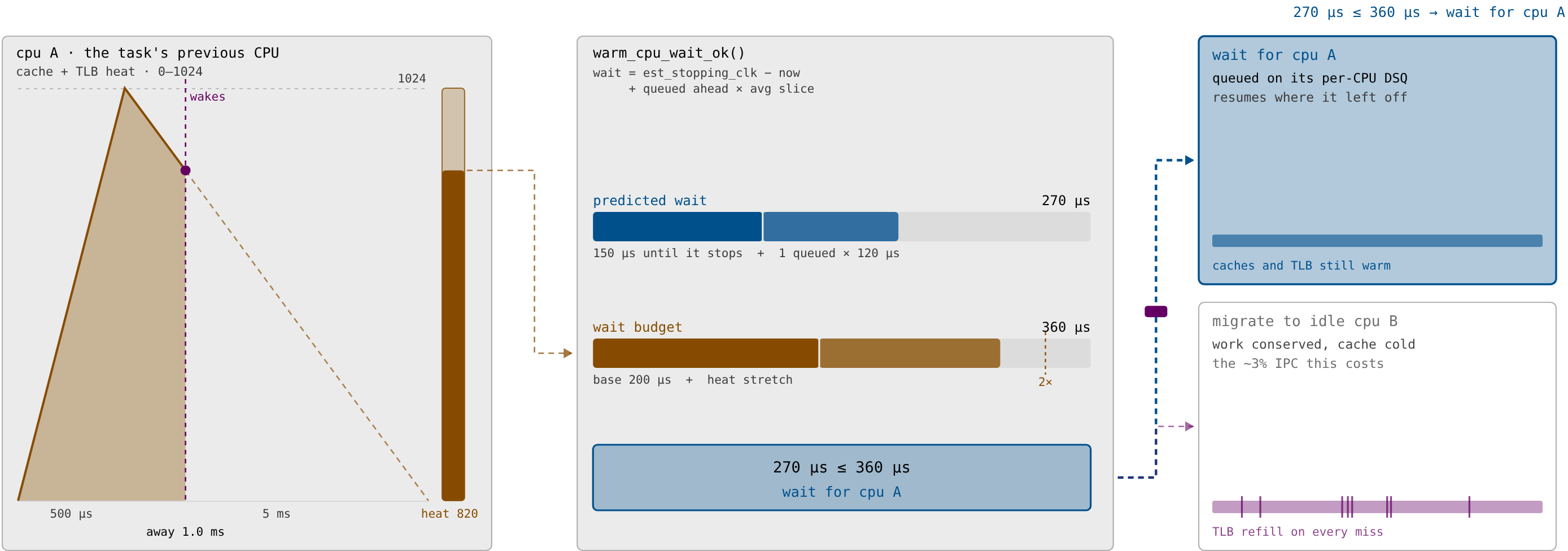
Neither side is observable. Both are predictions — and while T waits, X keeps evicting its lines.

Three existing approaches to stickiness



Should a task wait for its old CPU?

Stay if the previous CPU is **idle** — or if the predicted wait, time until it stops + nr_queued × avg slice, fits a budget the task's heat stretches up to 2×. Otherwise migrate, and **refill**.

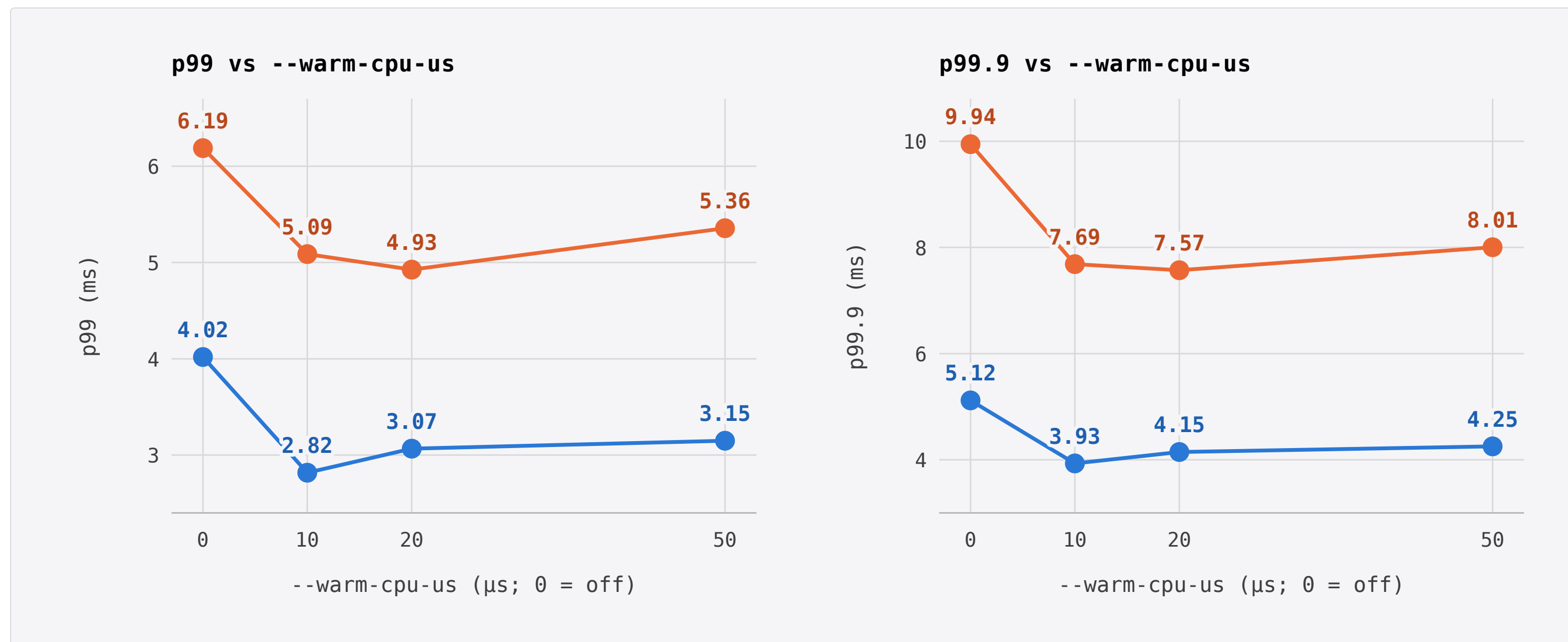


scx_lavd: Modulate CPU Stickiness using Task Warmth – David Dai, Meta, July 2026 · scx #3714

Observation: the --warm-cpu-us curve

Wake-time warmth stick. 0 = lavd base. The win plateaus at 10–20 μ s; 50 μ s decays it.

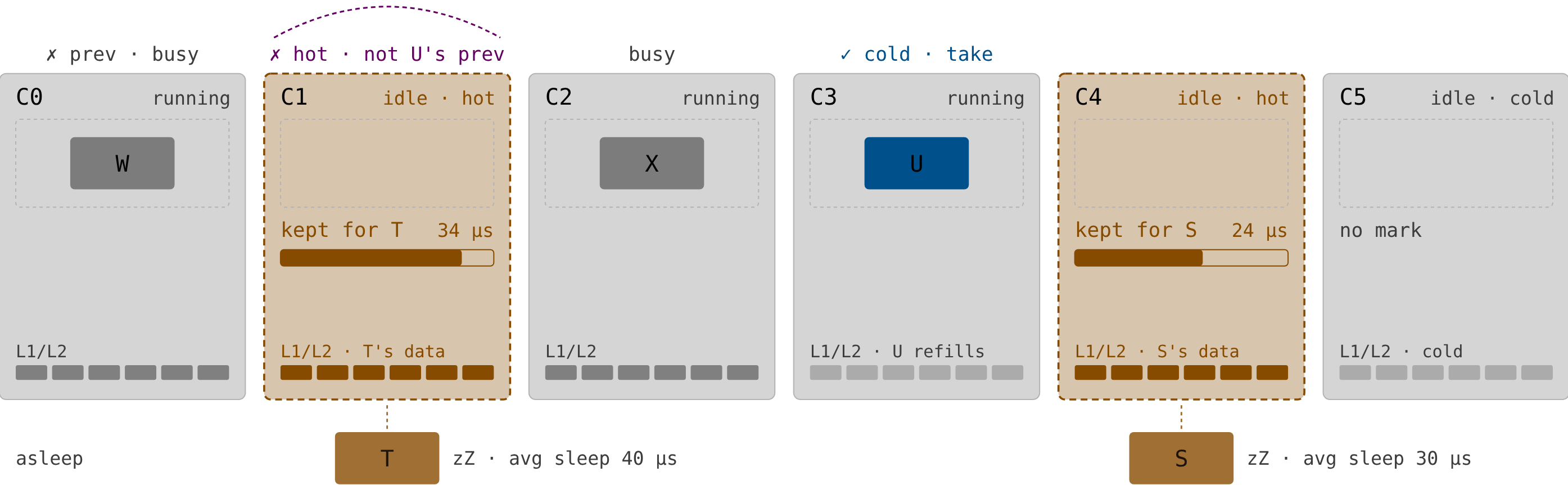
— light load (~25%) — moderate (~55%)



Keep the warm CPU idle for one short sleep

When a short sleeper leaves its CPU idle, the CPU stays **cache-hot** for the task's average sleep. Another wakee's idle search **skips** it and takes a cold CPU; the sleeper finds its previous CPU idle and takes it at once. A timer, not a lock — no owner is stored.

t = 6 μs · U wakes · its previous CPU C0 is busy → search the LLC for an idle CPU



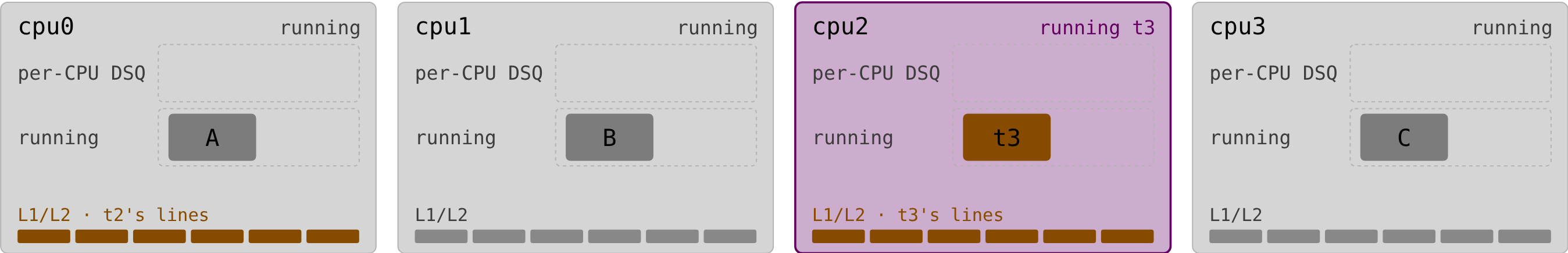
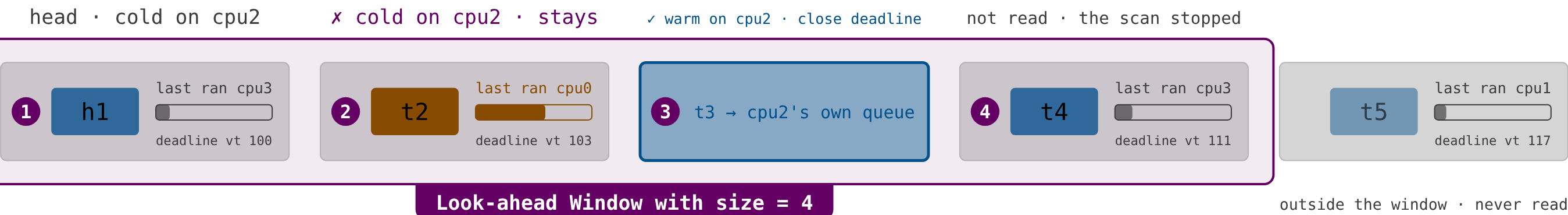
[PATCH v2 0/3] Introduce SIS_CACHE to choose previous CPU during task wakeup – Chen Yu, Intel, Nov 2023 · not merged

Look ahead for warm tasks in the domain queue

When a CPU drains the shared domain queue, it looks ahead a few tasks past the head and tries to find one still **warm on this CPU**, sequentially, in deadline-increasing order; when a warm task is found, it **pulls** it to the per-CPU queue and keeps every other task in the shared queue **for any CPU**.

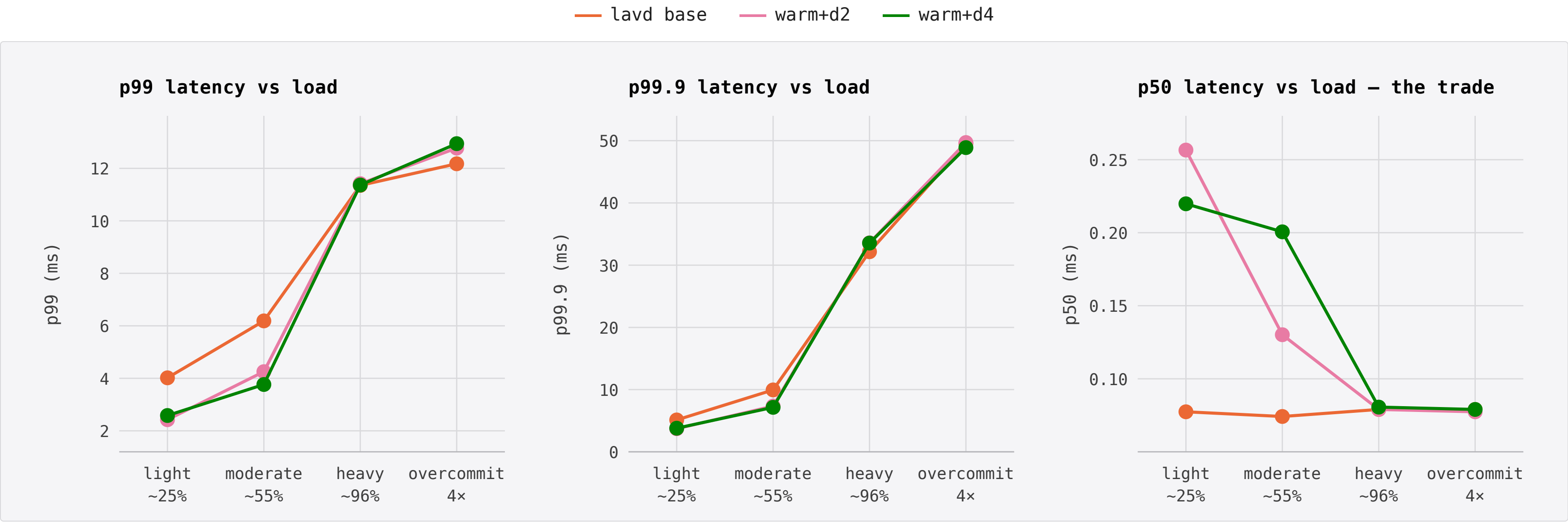
cpu2 consumes its per-CPU DSQ at once → t3 runs on its warm cache · the scan stops

Domain DSQ



scx_lavd: Add second-pass warm-task pull at domain DSQ dispatch – Gavin Guo, Igalia, Aug 2026 · not upstream

Latency vs load – all configs



Counting the queue is not timing it

Changwoo Min's completion-time estimate (#3806) prices the wait in time, not tasks: the running task's remaining time plus each queued task's own service time, over the core's capacity. For task warmth, T now waits for its warm CPU behind short work and moves on when long work is really ahead.

TODAY

5699c7f08582

```
wait = (est_stopping - now)
      + nr_queued × slice_wall
```

per-CPU DSQ only · a count times an average

▼ Changwoo's enhancement: priced in time, not tasks

COMPLETION TIME ENHANCEMENT

80eabdc60198

```
wait = residual(running)
      + queued service ÷ capacity
```

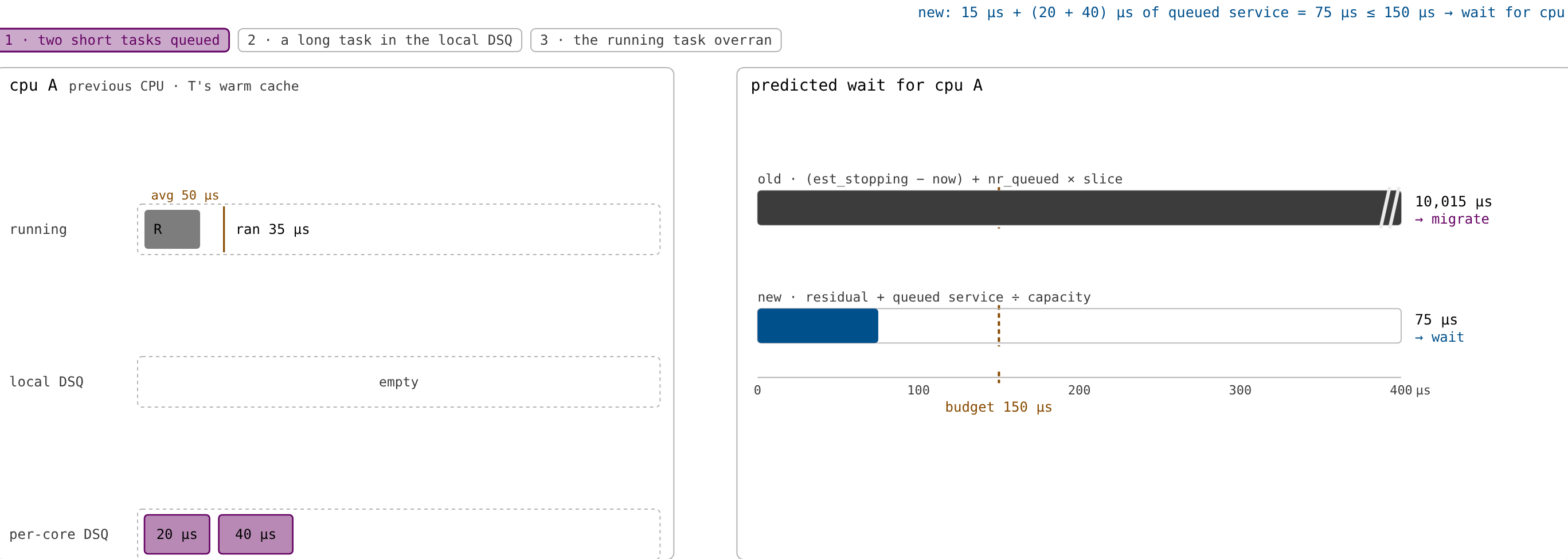
local + per-CPU DSQ · each task's own service time

QUEUED AHEAD OF T		nr_queued × slice	QUEUED SERVICE
20 μs	20 μs	10 ms	40 μs
4 ms	4 ms	10 ms	8 ms

slice_wall = 5 ms, the default slice max, below saturation

The count is wrong in both directions

Completion time prices each queued task at its own **service time**, reads the local DSQ, and charges a running task that has **outrun** its average for the overrun.



Completion time: Changwoo Min, PR #3806 (open) · stickiness change 80eabdc60198, not yet split out

Questions for the room

■ Warmth estimation

- How long does L1/L2/TLB state really survive on a busy CPU?
- Could the kernel expose a residency signal — PMU counters, cache occupancy?
- Intel RDT and Arm MPAM count cache occupancy — per task?
- Cheap enough to read at scheduling-decision frequency?

■ Wait or migrate?

- What should a scheduler maintain to predict the wait honestly?
- What is a migration worth, in time, for *this* task?

■ Scheduler ↔ userspace

- Allocator per-CPU caches: the real victims of migration
- Should they hint their locality needs to the scheduler?
- Or should the scheduler expose warmth and migration state to them?

Questions?

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



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