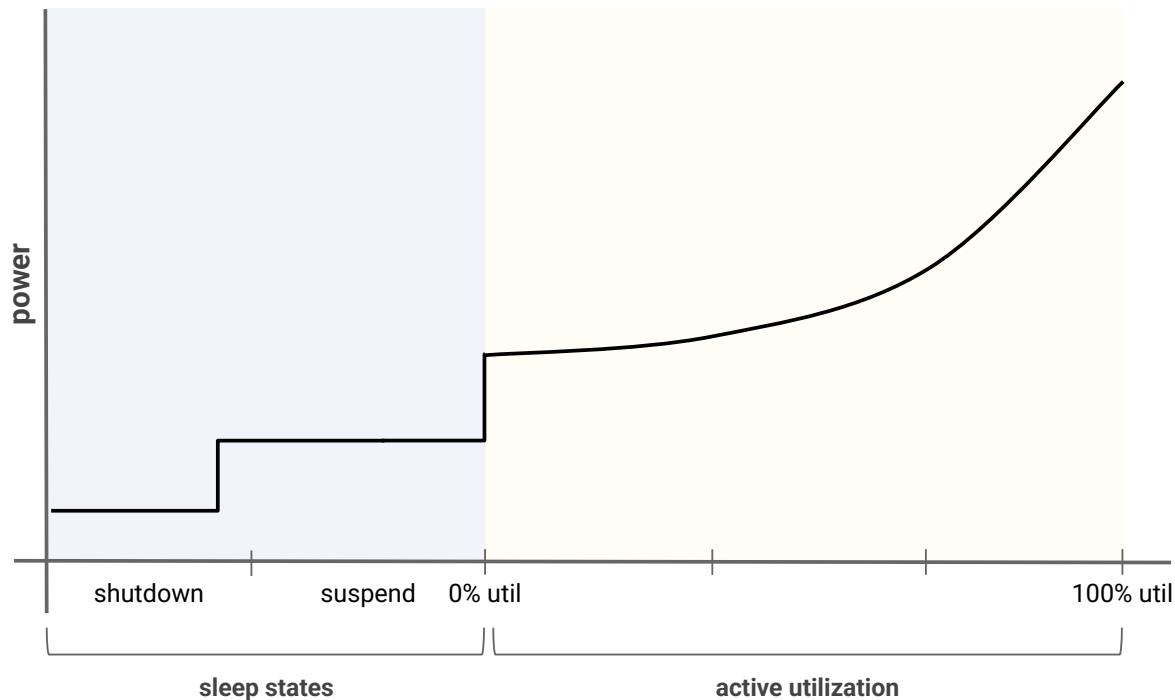


# Amortizing CPU Wakeup Costs

Samuel Wu <[wusamuel@google.com](mailto:wusamuel@google.com)>



# Background: Device power

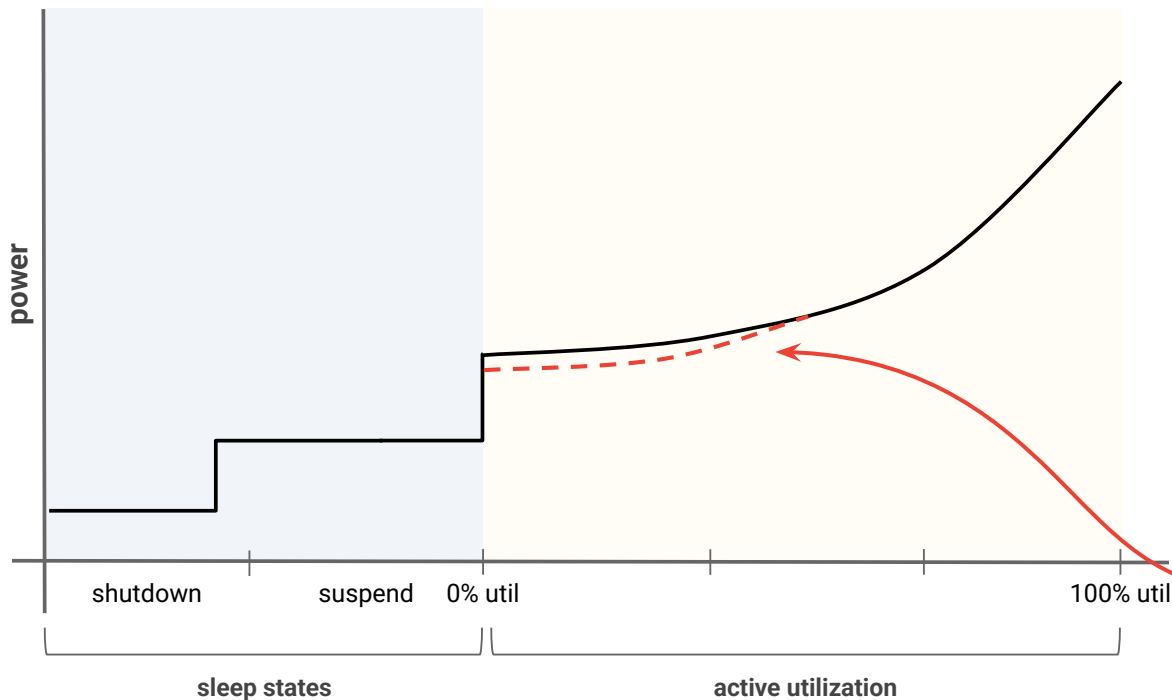


## Device power

- Power during sleep states is like a staircase
- During active use, power ramps up polynomially

*\*Chart not to scale*

# Background: Lowering power at low-util



Targeting around the **quiescent state**, when the device is at low CPU util but not suspended.

## Examples

- screen-off audio playback
- Off-peak server traffic
- Just a generally idle device!

**Can the power floor be lowered?**

# Background: CPU idle transitions

## Ftrace

1.0042: cpu\_idle: state=4294967295 cpu\_id=7

1.0052: sched\_switch: prev\_comm=swapper ... next\_comm=task

1.0067: sched\_switch: prev\_comm=task ... next\_comm=swapper

1.0070: cpu\_idle: state=1 cpu\_id=7

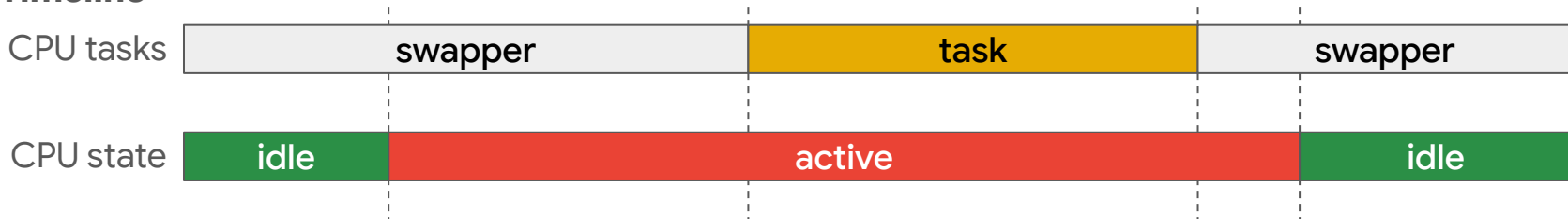
(idle exit)

(switch to task)

(switch to swapper)

(idle entry)

## Timeline



# Background: CPU idle transition overhead

## Ftrace

1.0042: cpu\_idle: state=4294967295 cpu\_id=7

1.0052: sched\_switch: prev\_comm=swapper ... next\_comm=task

1.0067: sched\_switch: prev\_comm=task ... next\_comm=swapper

1.0070: cpu\_idle: state=1 cpu\_id=7

(idle exit)

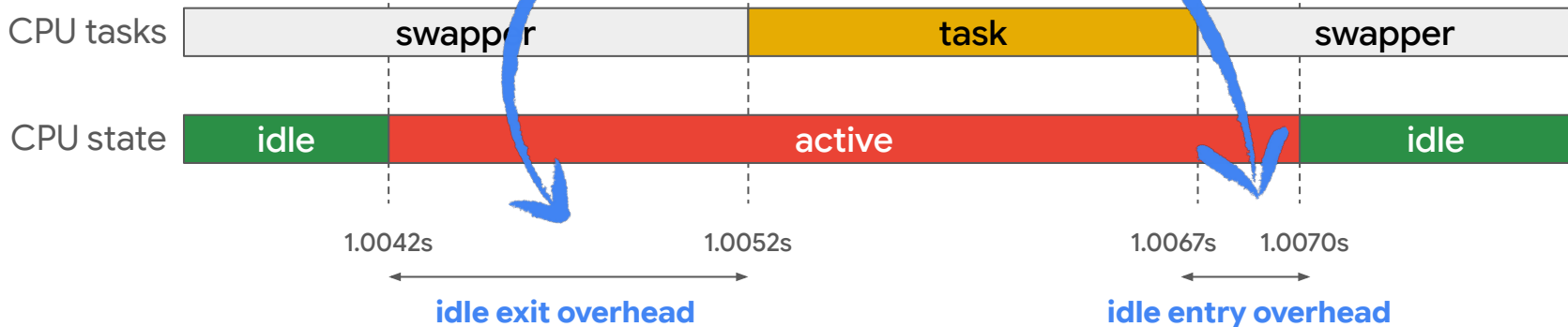
(switch to task)

(switch to swapper)

(idle entry)

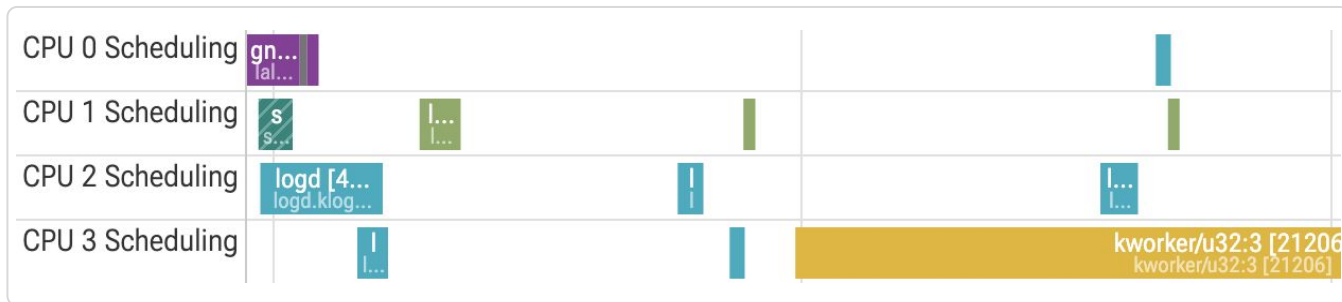
## Overhead!

## Timeline

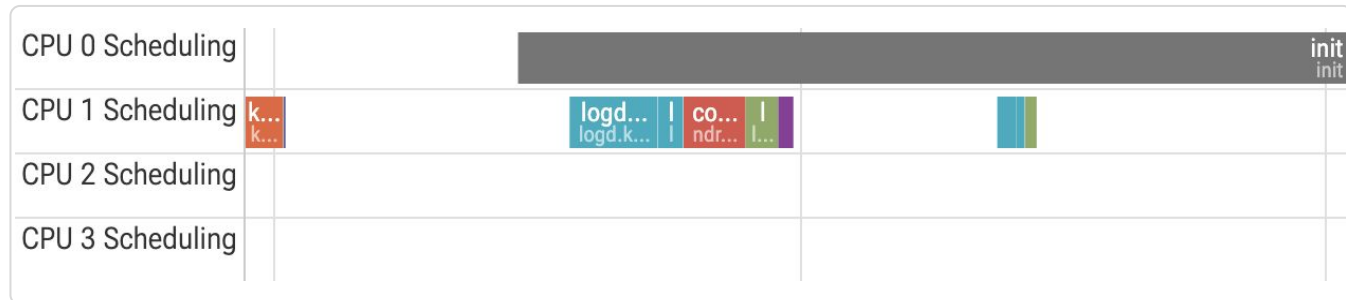


\*Illustrative timeline

# Proposal: Minimize idle transitions with a “free ride”?



Every task here pays the CPU idle exit/entry overhead



Batching minimizes idle exits, amortizing overhead

Subsystems have learned to batch—  
except for the subsystem that decides  
when CPUs wake up

There already is a precedent for batching and  
deferrals in Linux:

- RCU\_LAZY
- timer slack
- dirty expiry writebacks

**Does batching and  
reducing CPU idle  
exits result in less CPU  
power?**

“ It could be preferable from an energy standpoint to pack small tasks on a CPU in order to let other CPUs go in deeper idle states, but that could also hurt our chances to go cluster idle, and we have no ways to tell with the current Energy Model

– sched/[fair.c](#)

”

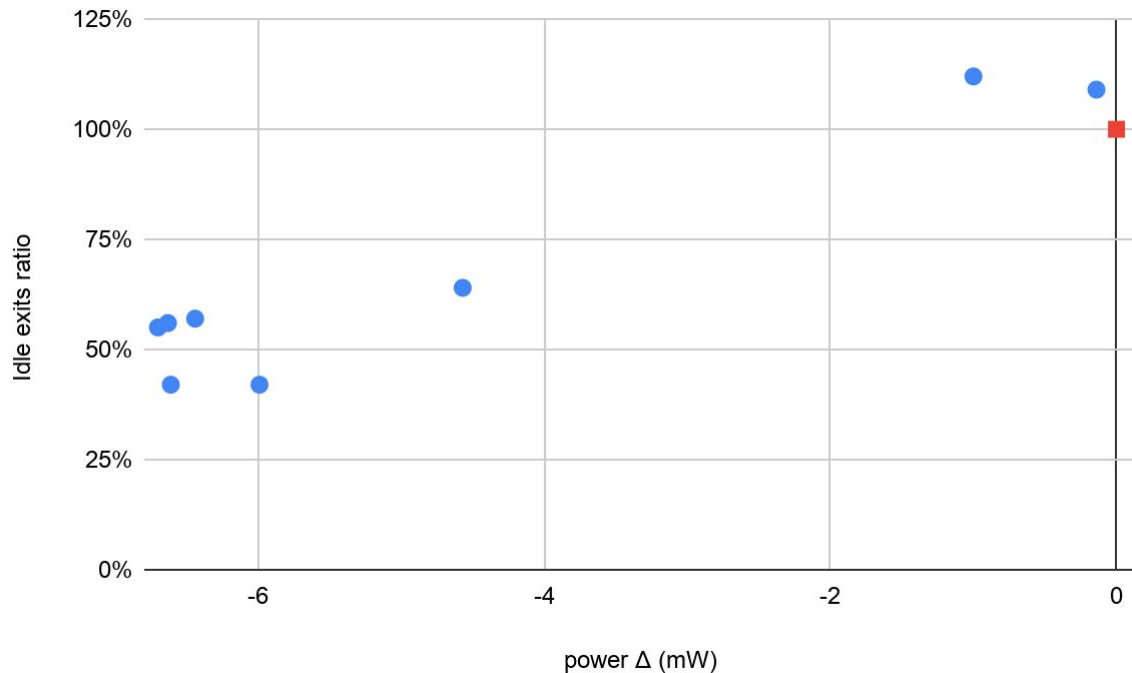
Task scheduling doesn't account for \_\_\_\_\_ because it doesn't have the information to do so

- CPU idle power
- cluster idle power
- idle transition costs

Is it worth adding? If so, how can it be added?

# Scheduling policies vs EEVDF

idle exits vs power delta



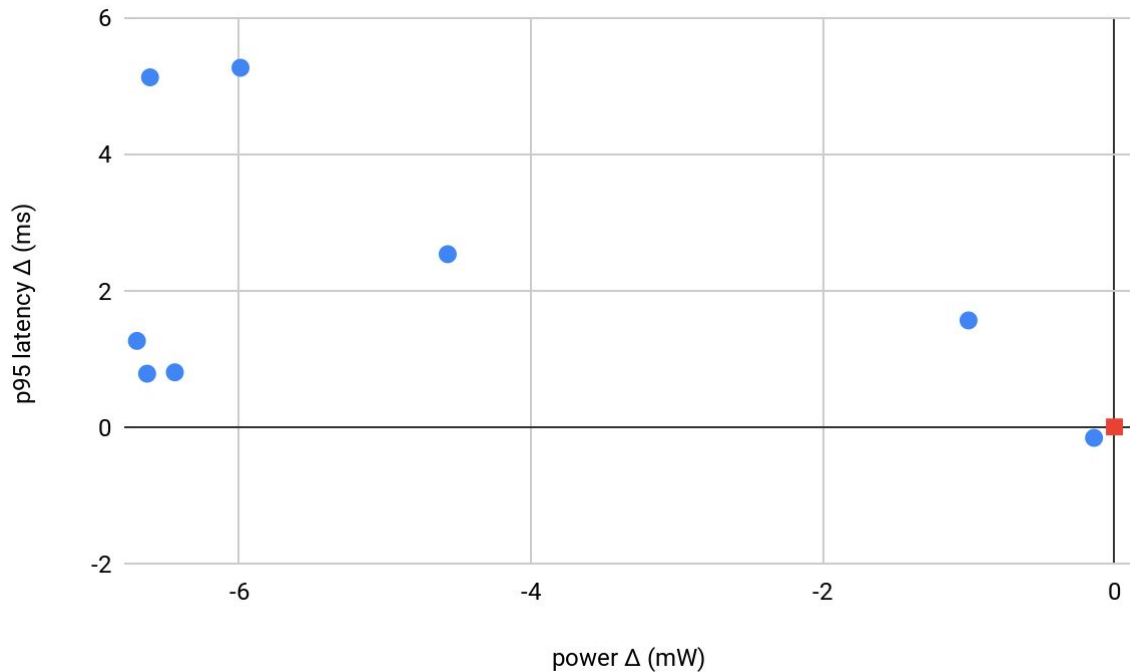
Each point represents a different scheduling policy's CPU power and CPU idle exits at low-util state.

Numbers are relative to the 7.1 EEVDF scheduler.

**There is a correlation between idle exits and power saved**

# Scheduling policies vs EEVDF

latency vs power delta



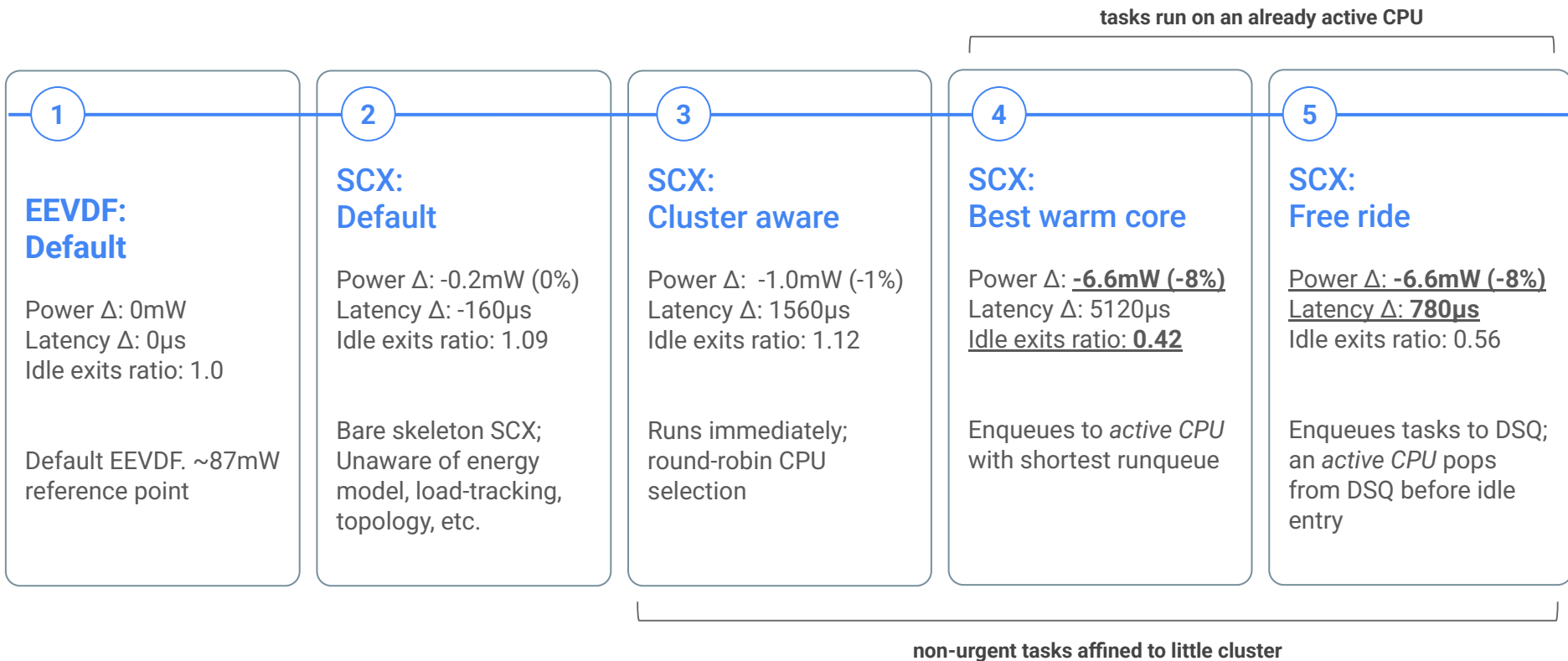
Each point represents a different scheduling policy's CPU power and tasks' p95 latency at low-util state.

Numbers are relative to the 7.1 EEVDF scheduler.

Can the performance requirements be **relaxed** during device **low-util state**?

There is a scheduling heuristic that minimizes power with acceptable latency.

# A few noteworthy results



Latency is not the price of batching;  
it's the price of committing at  
wakeup time

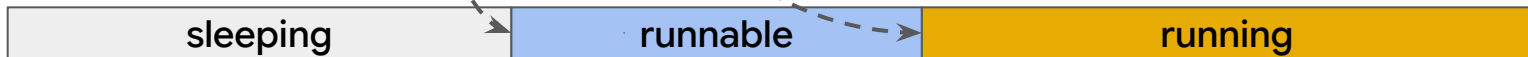
Energy savings comes from running on an **already active** CPU. But when to assign a task to a CPU?

Assign a CPU at task wakeup

- p95 latency: +5120 $\mu$ s

Assign a CPU at task switch-in

- p95 latency: +780 $\mu$ s



# Summary

01

## Background and motivation

- Scope: during low device utilization
- Reducing idle exits == reduces power
- Batching tasks reduces idle exits

02

## Proposal

- Other subsystems use batching and deferrals
- Getting a “free ride” by leveraging an already active CPU saves significant power

03

## Tradeoffs

- Batching can increase latency
- Deferring committing to a CPU protects us from latency

# Open questions:

Energy aware scheduling

Batching and deferral

Classifying latency-tolerance

Should the EM expose an idle-exit cost term?

Should cluster packing vs CPU packing be a heuristic (e.g. util is low, CPUs are at FMin)?

`find_energy_efficient_cpu()` spreads inside a cluster because the Energy Model only knows of active OPP, not idle OPP or C-state exit energy

# Open questions:

Energy aware scheduling

Batching and deferral

Classifying latency-tolerance

Per-cluster runqueue?

Task push-pull (like RT threads)?

`select_task_rq()` commits a task to a CPU at task wakeup.

From a power perspective, our best result comes from deferring that decision.

# Open questions:

Energy aware scheduling

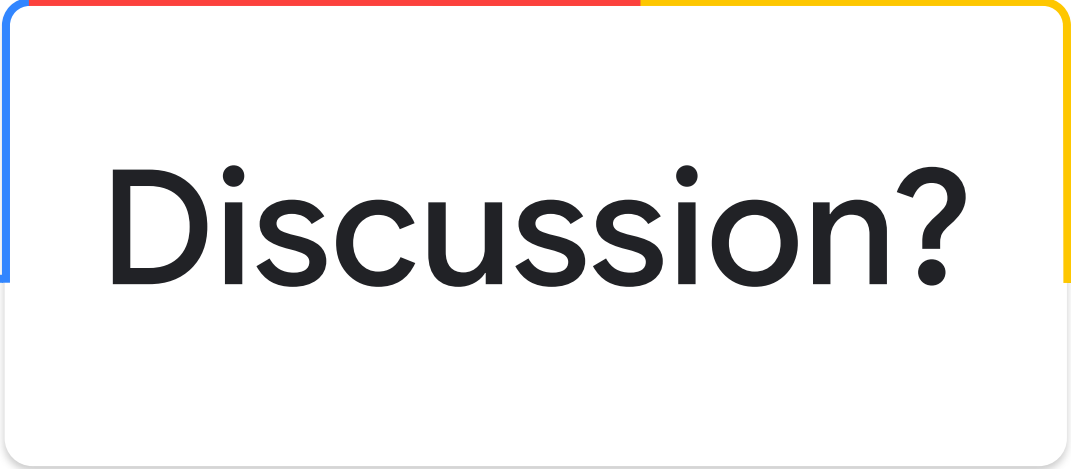
Batching and deferral

Classifying latency-tolerance

Are there QoS hints or other heuristics to classify latency-tolerant tasks?

What signals help classify that the system is currently latency-tolerant?

1. There exists a subset of tasks that are latency tolerant
2. Certain situations/use-cases are intrinsically latency tolerant (e.g. device idle)



**Discussion?**

| Aggregated raw data (Pixel 9, 7.1 kernel, pinned to FMin, n=90) |                  |                                 |                        |                    |             |              |                  |                    |                  |
|-----------------------------------------------------------------|------------------|---------------------------------|------------------------|--------------------|-------------|--------------|------------------|--------------------|------------------|
|                                                                 | Cluster affinity | Target CPU                      | Batching               | Urgency filter     | power (mW)  | power Δ (mW) | p95 latency (μs) | p95 latency Δ (μs) | idle exits ratio |
| eevdf                                                           | All              | EAS capacity-aware              | Immediate              | Native             | 87.00 ±0.45 | —            | 890              | —                  | 100%             |
| m0                                                              | All              | scx_bpf_select_cpu_dfl()        | Immediate              | N/A (all imm.)     | 86.81 ±0.39 | -0.14        | 720              | -160               | 109%             |
| m1                                                              | Little           | DSQ pull                        | Deferred               | Yes (nice <0)      | 80.37 ±0.44 | -6.63        | 1660             | +780               | 56%              |
| m2                                                              | Little           | Pack onto CPU0                  | Immediate              | Yes (nice <0)      | 82.43 ±0.38 | -4.57        | 3410             | +2530              | 64%              |
| m3                                                              | Little           | DSQ pull                        | Deferred               | Yes (prev-cluster) | 80.56 ±0.43 | -6.44        | 1690             | +800               | 57%              |
| m4                                                              | Little           | Round-robin                     | Immediate              | Yes (nice <0)      | 86.01 ±0.46 | -1.00        | 2440             | +1560              | 112%             |
| m7                                                              | Little           | Search awake                    | Hybrid (Imm. if awake) | Yes (nice <0)      | 81.01 ±0.40 | -5.99        | 6140             | +5260              | 42%              |
| m8                                                              | Little           | Search awake + shallowest queue | Hybrid (Imm. if awake) | Yes (nice <0)      | 80.44 ±0.40 | -6.61        | 6000             | +5120              | 42%              |
| m9                                                              | Little           | DSQ pull                        | Deferred               | No (all deferred)  | 80.30 ±0.42 | -6.70        | 2150             | +1260              | 55%              |

- eevdf (Stock Baseline): Native Linux EEVDF scheduler with Android Energy Aware Scheduling (EAS) enabled (sched\_ext unloaded).
- m0 (scx Default Baseline): Thin sched\_ext pass-through that calls scx\_bpf\_select\_cpu\_dfl() and immediately dispatches/kicks across all CPUs (0–7).
- m1 (Lazy Batching + Urgency Filter): Defers normal (nice >= 0) wakeups into a shared LAZY\_DSQ without sending an IPI—drained only by Little cores (0–3) when already awake or on a tick—while urgent (nice < 0 / pinned) tasks bypass LAZY\_DSQ and kick immediately.
- m2 (Immediate Little Packing): Immediately dispatches and kicks normal wakeups onto cpu0 (SCX\_DSQ\_LOCAL\_ON | 0, packing onto a single Little core with no LAZY\_DSQ deferral), isolating spatial packing without temporal batching.
- m3 (Cluster-Local + Lazy Little Batching): Same Little-cluster LAZY\_DSQ deferral as m1 for background work, but constrains Mid/Big (cpu 4–7) wakeups to stay within their previous cluster rather than spilling across clusters.
- m4 (Immediate Little Round-Robin): Immediately dispatches and kicks normal wakeups across Little cores (0–3) in round-robin order (SCX\_DSQ\_LOCAL\_ON | (rr & 3), no LAZY\_DSQ), isolating Little-cluster confinement without packing or batching.
- m7 (Active Awake-Little Push + Lazy Fallback): Actively scans Little cores (0–3) at wakeup and immediately queues onto the first non-idle (!idle) Little core's local runqueue (SCX\_DSQ\_LOCAL\_ON), falling back to LAZY\_DSQ only when all four Little cores are idle.
- m8 (Active Warm-Little Push + Lazy Fallback): Uses cpu\_idle tracepoint tracking to immediately queue onto an active (!idle) or shallow-WFI (C0) Little core (SCX\_DSQ\_LOCAL\_ON), falling back to LAZY\_DSQ only when all four Little cores are in deep idle (C1+).
- m9 (Unfiltered Lazy Batching): Identical to m1's LAZY\_DSQ deferral and Little-cluster drain, except the urgency filter (nice < 0) is turned off, forcing all movable tasks (including high-priority audio/UI/binder threads) into LAZY\_DSQ.

| Aggregated raw data (Pixel 6, 7.1 kernel, pinned to FMin, n=70) |                  |                    |           |                |             |               |                  |                    |                  |
|-----------------------------------------------------------------|------------------|--------------------|-----------|----------------|-------------|---------------|------------------|--------------------|------------------|
|                                                                 | Cluster affinity | Target CPU         | Batching  | Urgency filter | power (mW)  | power Δ (mW)  | p95 latency (μs) | p95 latency Δ (μs) | idle exits ratio |
| eevdf                                                           | All              | EAS capacity-aware | Immediate | Native         | 15.24 ±0.26 | —             | 1202             | —                  | 100%             |
| m1                                                              | Little           | DSQ pull           | Deferred  | Yes (nice <0)  | 14.02 ±0.23 | -1.23 (-8.1%) | 3202             | +2000              | 86%              |