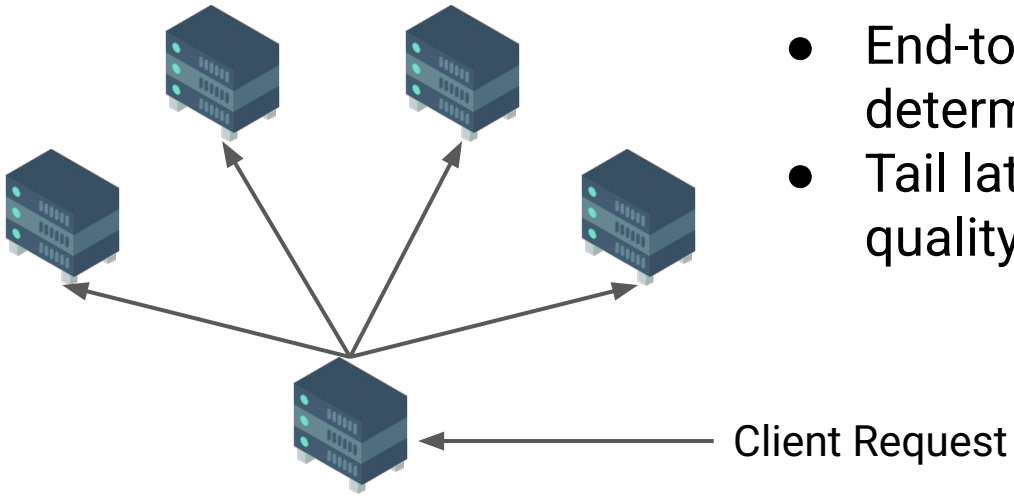
Caladan: Mitigating Interference at Microsecond Timescales

東京大学 品川研究室
呉 国駿

Background

Minimizing tail latency is critical



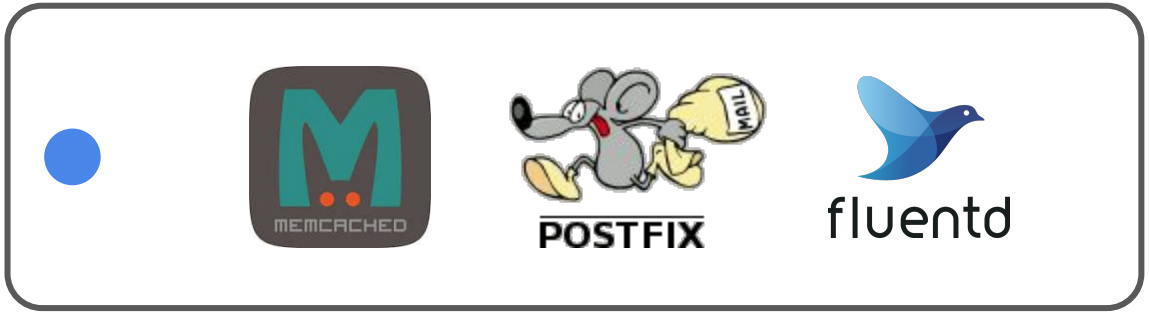
Distributed System

- End-to-end response time is determined by tail latency
- Tail latency strongly affects the quality of service

Background

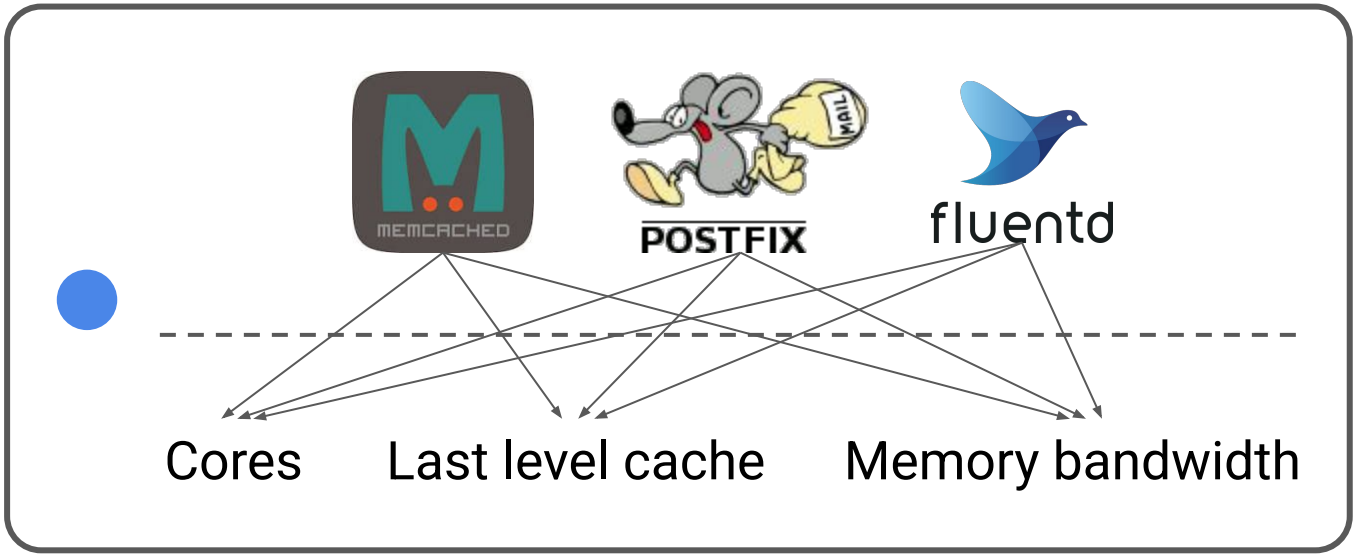
Balance latency with datacenter efficiency

Pack several tasks together on the same machine



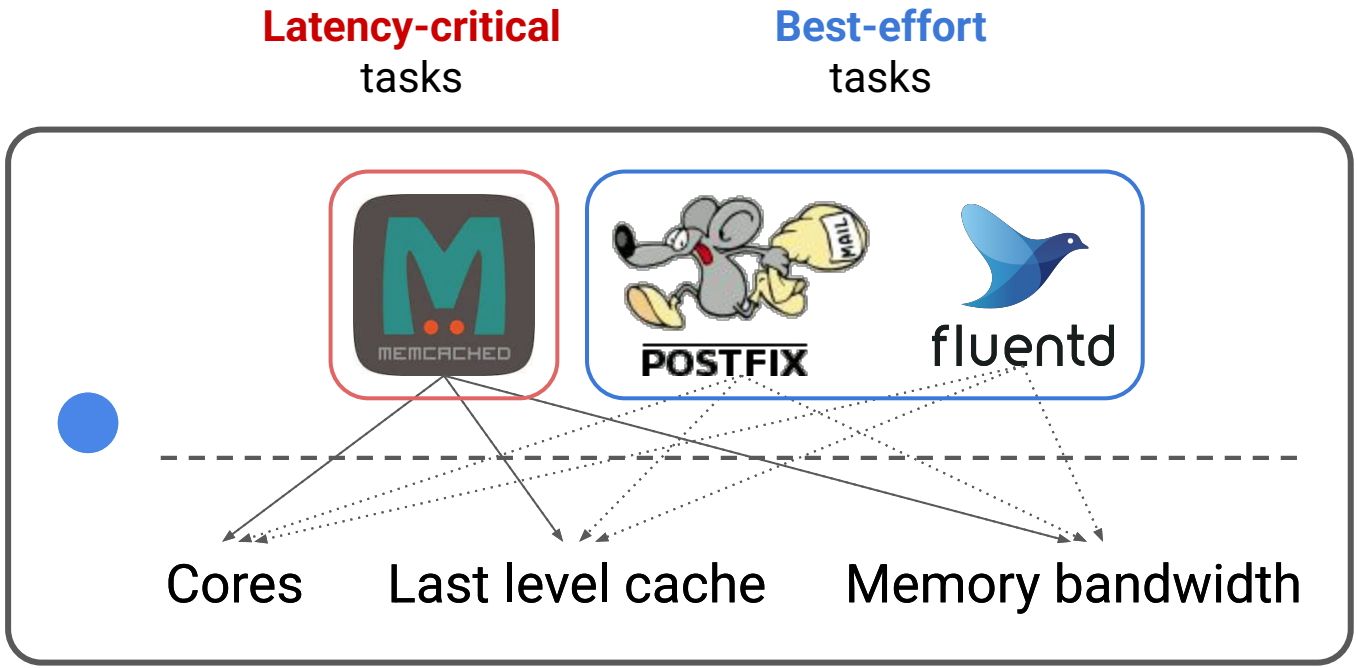
Background

Tasks compete over shared CPU resources



Background

Prioritize latency-critical tasks when interference is detected

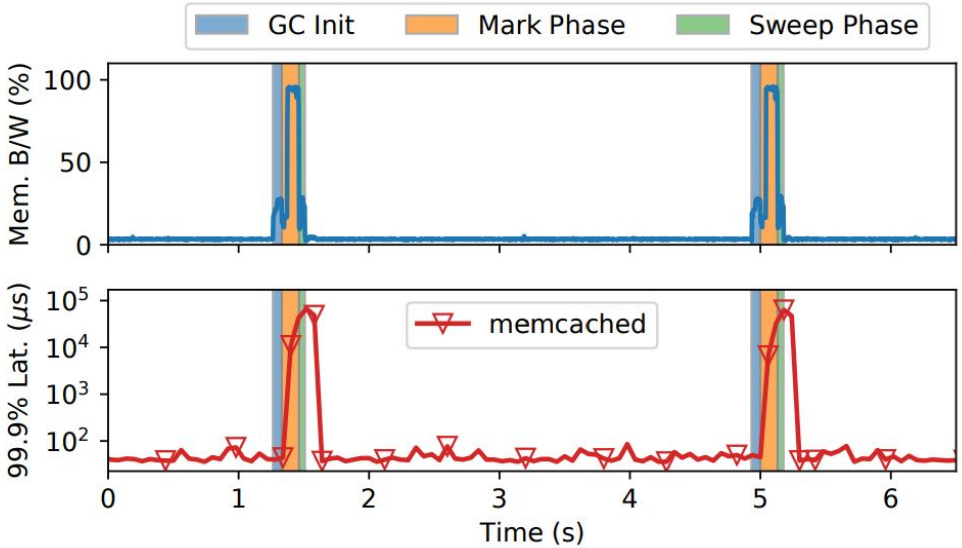


Background

- Avoid performance regression in LC tasks
- When interference is detected,
  - throttle BE tasks
  - assign additional resources to LC tasks

Challenge

Abrupt changes in resource usage abruptly increase interference



Periodic GC increases memory bandwidth usage, causing memcached (LC task) to reach a **1000×** higher tail latency than normal.

Challenge

Abrupt changes in resource usage are common

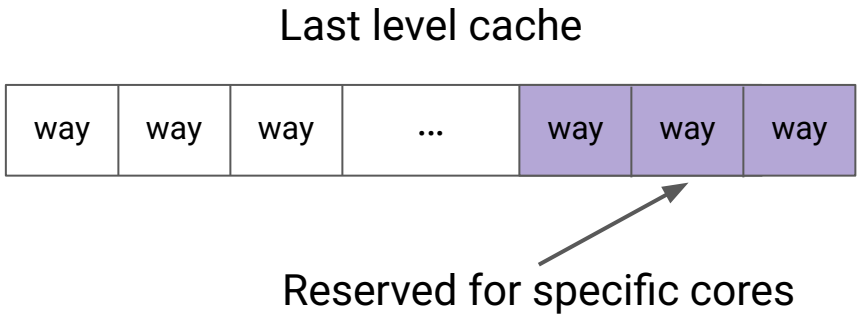
- Application load can be bursty at microsecond-scales
  - Network traffic in Google's datacenters
  - Thread wakeups in Microsoft's Bing service
- Many applications also exhibit phased resource usage
  - Garbage collection
  - Compression, compilation

Observation: To prevent the latency from increasing by 50 µs, the system has to mitigate interference within **100 µs**.

Existing solutions

Resource partitioning technology

- Intel’s Cache Allocation Technology (CAT)
- Intel’s Memory Bandwidth Allocation (MBA)
- Dedicating cores
- Disable hyperthreading



Existing solutions

Systems based on resource partitioning fail to react quickly

- Statically assigning enough resources prevents efficient CPU utilization
- Making dynamic adjustments to partitions takes many seconds to converge to the right configuration

System	Decision Interval	Typical Convergence	Requires CAT	Supports HT
Heracles [38]	2–15 s	30 s	✓	✗
Parties [12]	500 ms	10–20 s	✓	✗
Caladan	10–20 μs	20 μs	✗	✓

Table 1: A comparison of Caladan to state-of-the-art systems that use partitioning to manage interference. Caladan can converge to the right resource configuration 500,000× faster.

Much longer than 100µs

Proposal

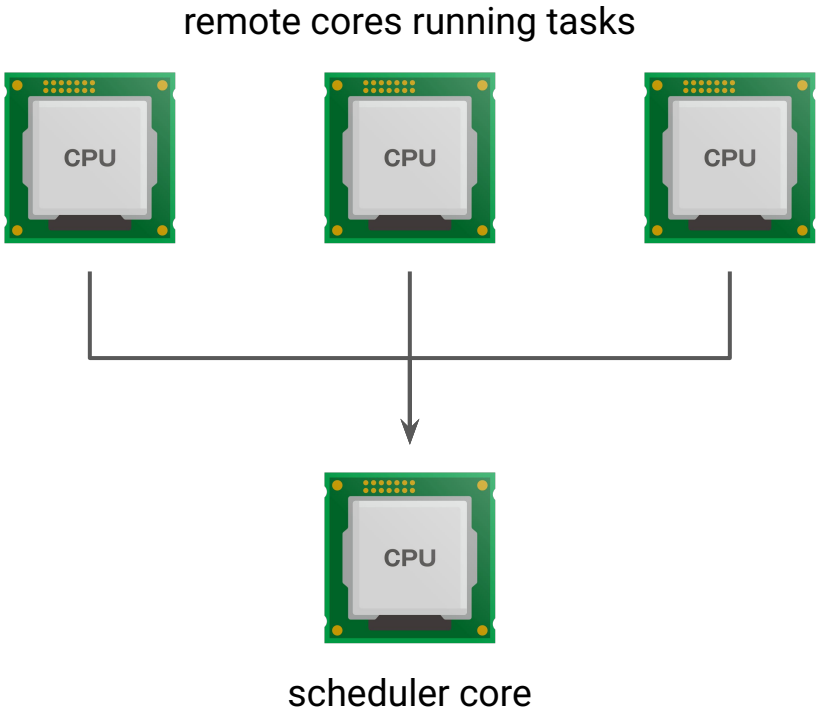
Manage interference exclusively by adjusting core allocations

Resource partitioning is not necessary. What really needed are:

- **Fast interference detection**
  - Identify the task and contented resource within microsecond timescales
- **Scalability regarding core count and task count**
  - Keep the overhead from Caladan low

Caladan's Design — Toward Fast Interference Detection

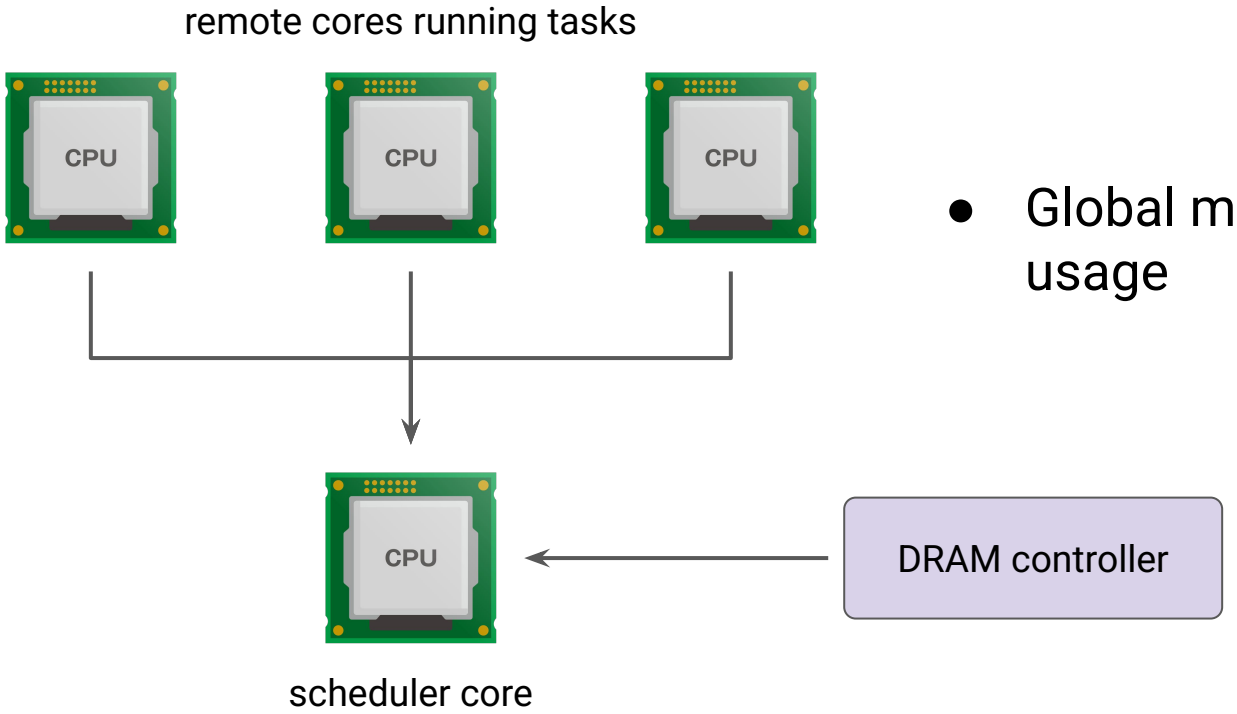
A dedicated scheduler core polls *control signals* every 10µs



- Queueing delays
- Request processing times
- LLC miss rates
- Idle notices

Caladan's Design — Toward Fast Interference Detection

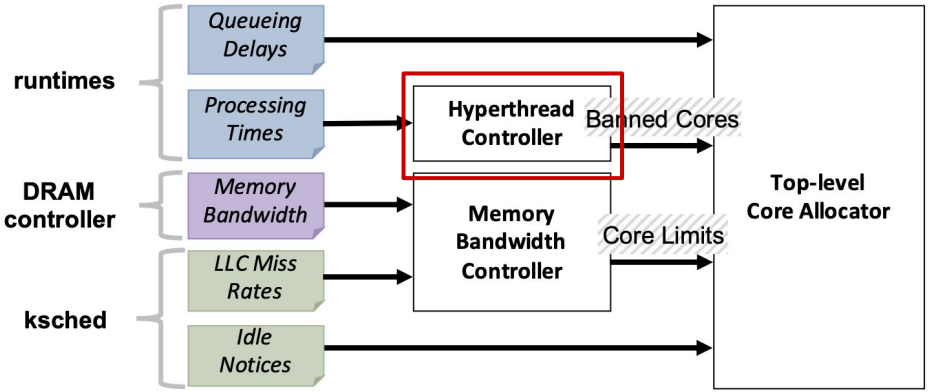
A dedicated scheduler core polls *control signals* every 10µs



- Global memory bandwidth usage

Caladan's Design — Toward Fast Interference Detection

How the gathered *control signals* are used



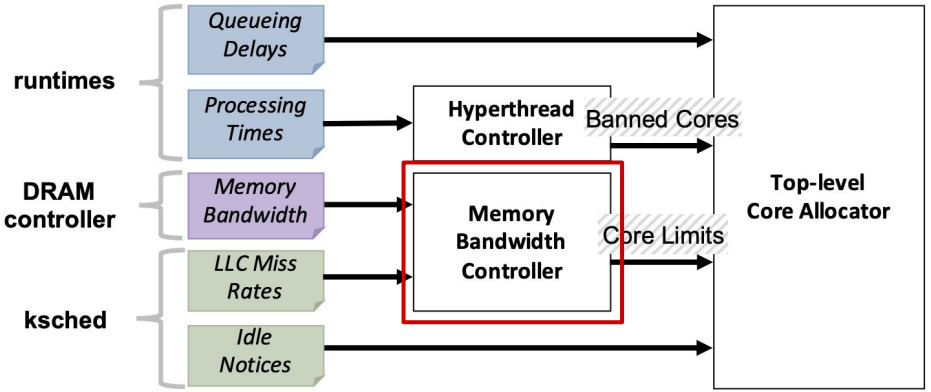
Hyperthread Controller

An LC task exceeds the processing time threshold

The controller bans the use of the sibling hyperthread

Caladan's Design — Toward Fast Interference Detection

How the gathered *control signals* are used



Memory Bandwidth Controller

Global memory bandwidth crosses a saturation threshold

↓

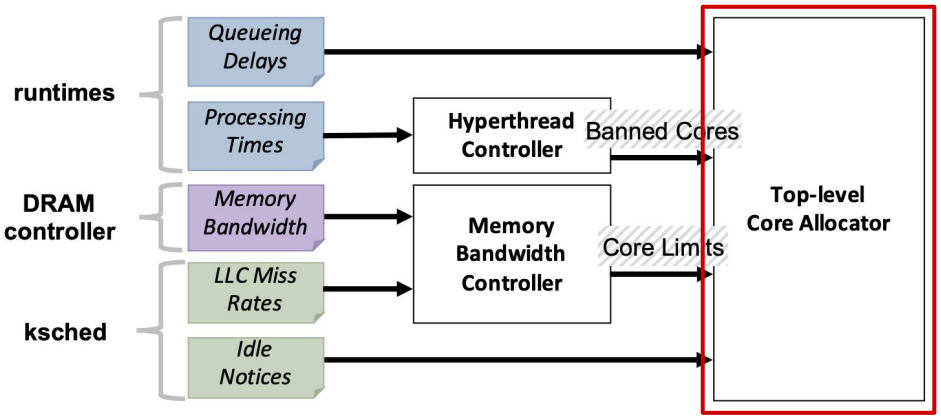
The controller attributes the increased usage to a specific task by relying on LCC miss rates

↓

The controller revokes one core from the worst offending task

Caladan's Design — Toward Fast Interference Detection

How the gathered *control signals* are used



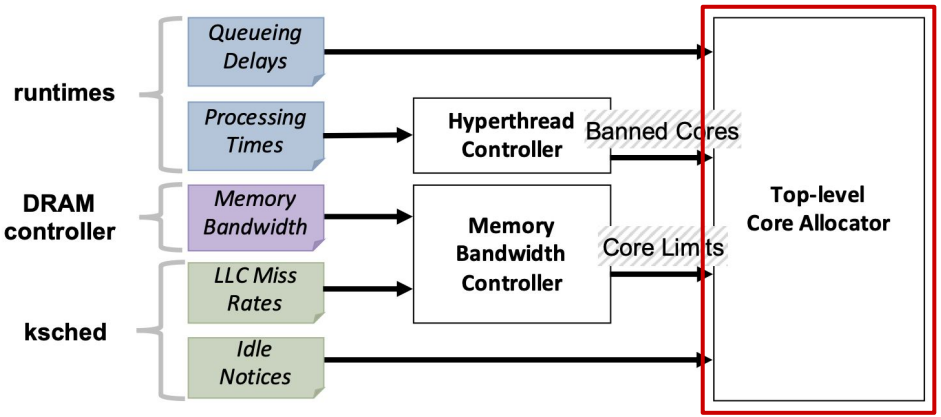
Top-level Core Allocator

An LC task experiences queueing delays above the threshold

Add cores to the task in a way that satisfies the controller constraints (A preemption may occur)

Caladan's Design — Toward Fast Interference Detection

How the gathered *control signals* are used



Top-level Core Allocator

When a task yields a core voluntarily, immediately tries to grant the core to another task

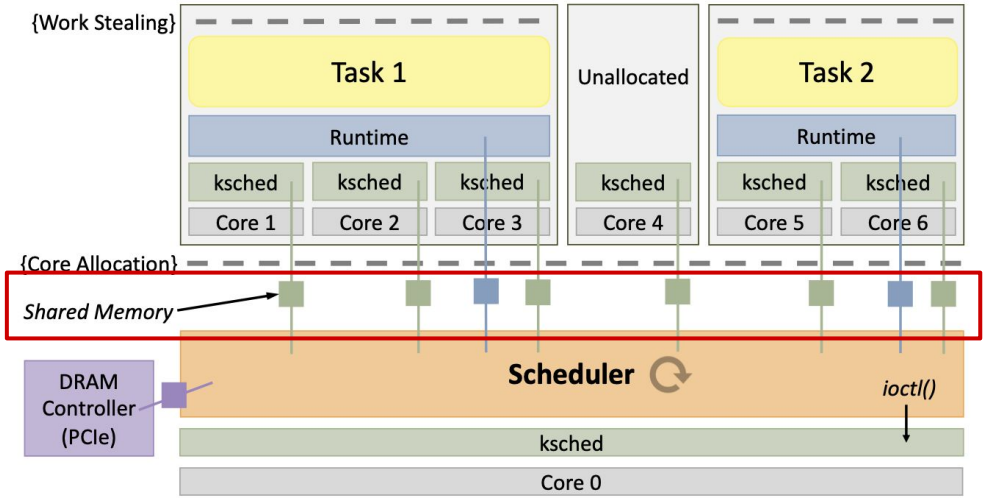
Caladan's Design — Toward Scalability

KSCHED Kernel Module accelerates scheduling and signal gathering

- Linux Kernel system call interface has limitations in performance
- With KSCHED, the scheduler core can effectively
  - Preempt an existing task and wake up a new task
  - Idle cores
  - read control signals

Caladan's Design — Toward Scalability

Use shared memory for communication between the scheduler and remote cores

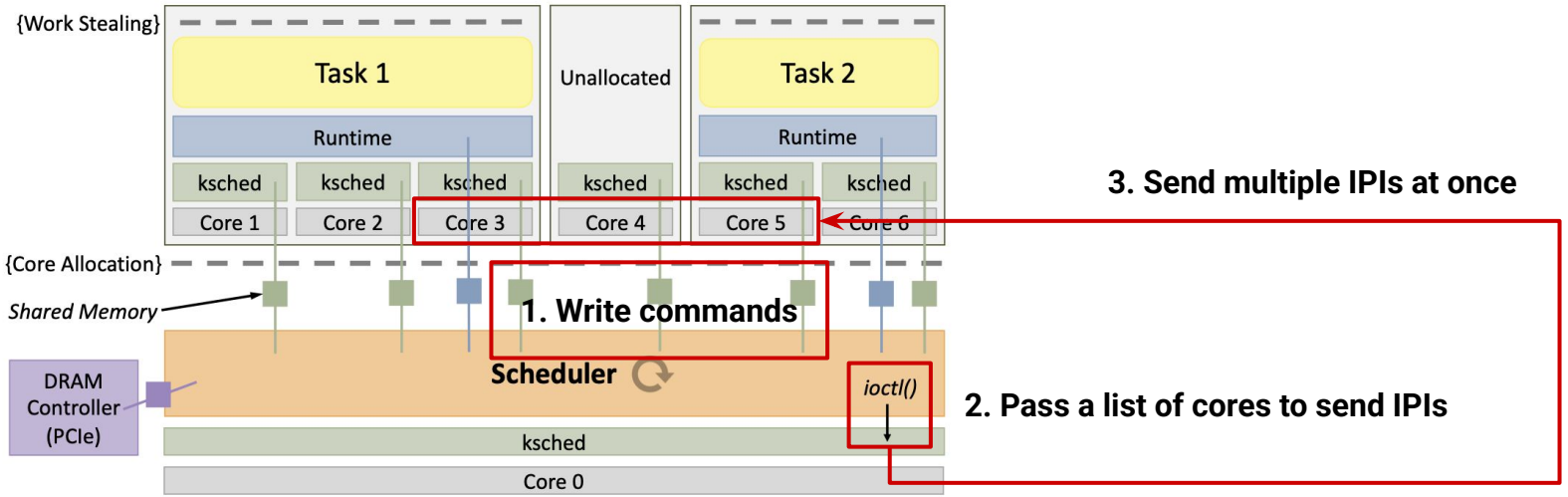


- **KSCHED** in remote cores writes
  - LCC miss rates
  - Idle notices
- **Runtime** writes
  - Queueing delays
  - Processing times
- **Scheduler** writes commands for the remote KSCHED to execute, including
  - Wake up a new task
  - Idle the core

Caladan's Design — Toward Scalability

KSCHED optimizations

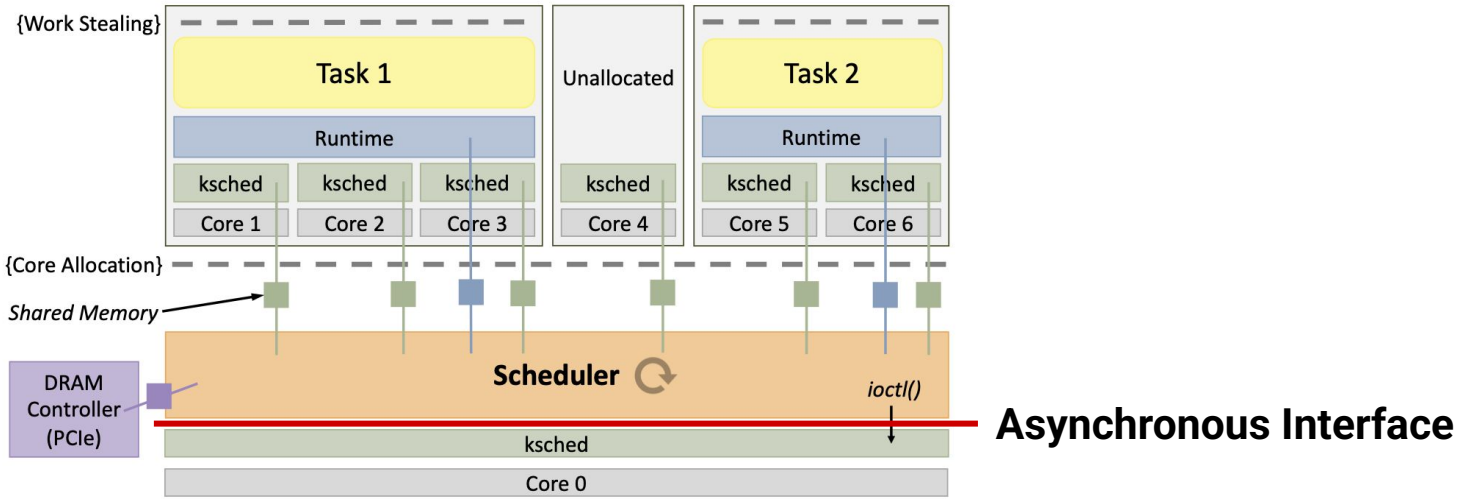
- Leveraging the ability of the interrupt controller to send multiple Inter-processor Interrupts (IPIs) at once
  - Sending IPIs are the most expensive operations



Caladan's Design — Toward Scalability

KSCHED optimizations

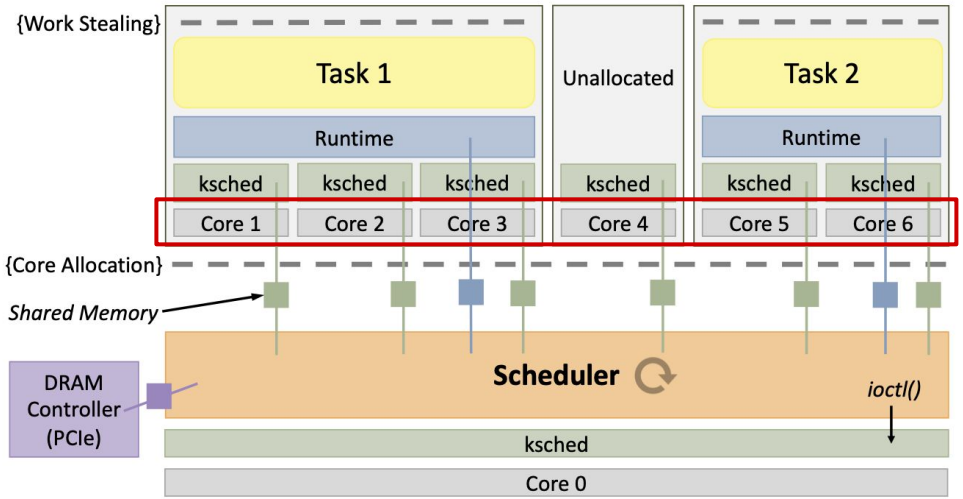
- KSCHED's commands are issued asynchronously
  - The scheduler can perform other work while waiting for the commands to complete



Caladan's Design — Toward Scalability

KSCHED optimizations

- Offload scheduling work to remote cores, such as
  - Sending signals to tasks
  - Affinitizing tasks to cores



Remote cores also does scheduling work

Caladan's Design

- Address the challenge of **fast interference detection** by
  - Carefully selecting control signals
  - Dedicating a core to monitor these signals and take action to mitigate interference

- Address the challenge of **scalability** by
  - Introducing a Linux Kernel module named KSCHED

Evaluation

Experiment 1. Colocating Memcached with swaptions-GC

(LC task) (BE task)

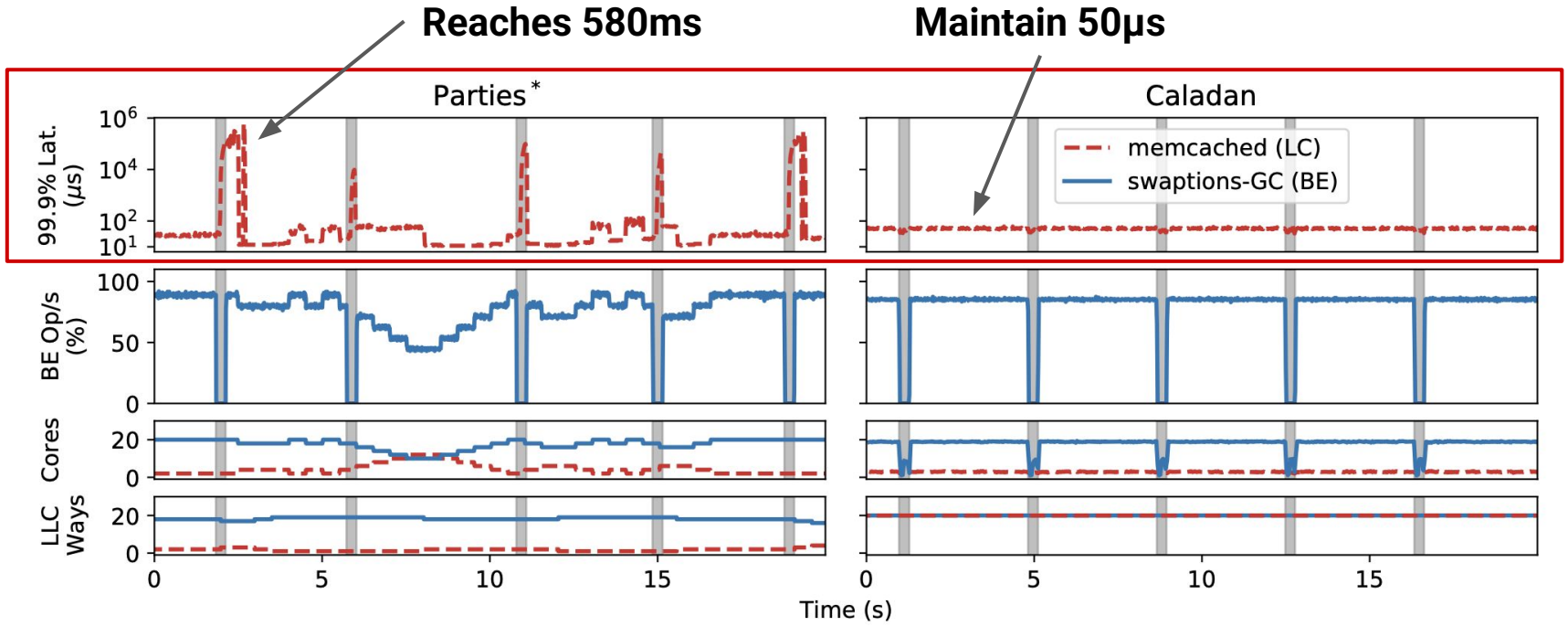
Baseline: state-of-the-art partitioning-based system, **Parties** [ASPLOS '19]

- Adjusts cores and last level cache partitions
- 500ms decision intervals, 10-20 seconds convergence

Evaluation

Memcached's latency at 99.9th percentile

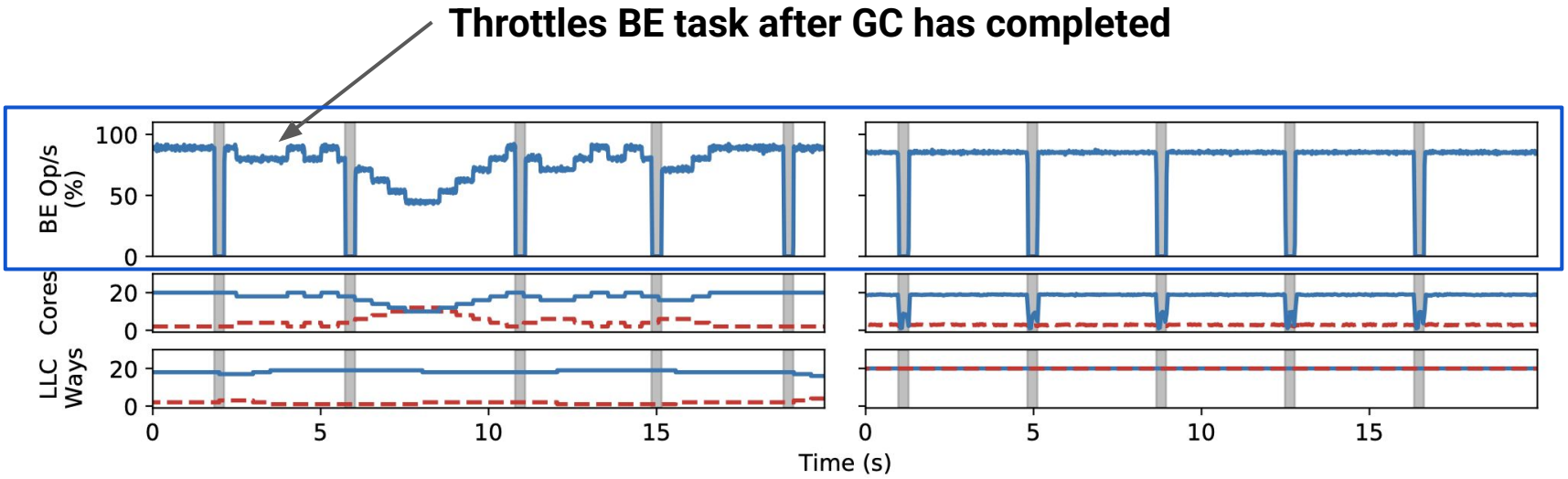
↓ Lower is better



Evaluation

BE task's throughput

↑ Higher is better



Evaluation

Experiment 2. Colocating many tasks (3 LC tasks and 2 BE tasks)

- Caladan was able to maintain low tail latency for all 3 LC tasks under varying load and interference
- Core reallocations occur up to 230,000 times per second

Discussion regarding compatibility

- Applications need to use Caladan’s runtime
  - Partial compatibility layer can help

- LC tasks are recommended to expose concurrency by spawning either a thread per connection or a thread per request
  - Applications that multiplex TCP connections per thread require changes to their architecture

Summary

- Present an interference-aware CPU scheduler, called Caladan
- Caladan consists of
  - A dedicated scheduler core that polls control signals and make a scheduling decision
  - A Linux Kernel module named KSCHED, which helps achieve better scalability
  - A custom runtime that exposes task concurrency and load
- Caladan manages interference exclusively by core reallocations
- Caladan can mitigate interference at microsecond timescales

References

Joshua Fried et al. Caladan: Mitigating Interference at Microsecond Timescales. In Proc. 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20).