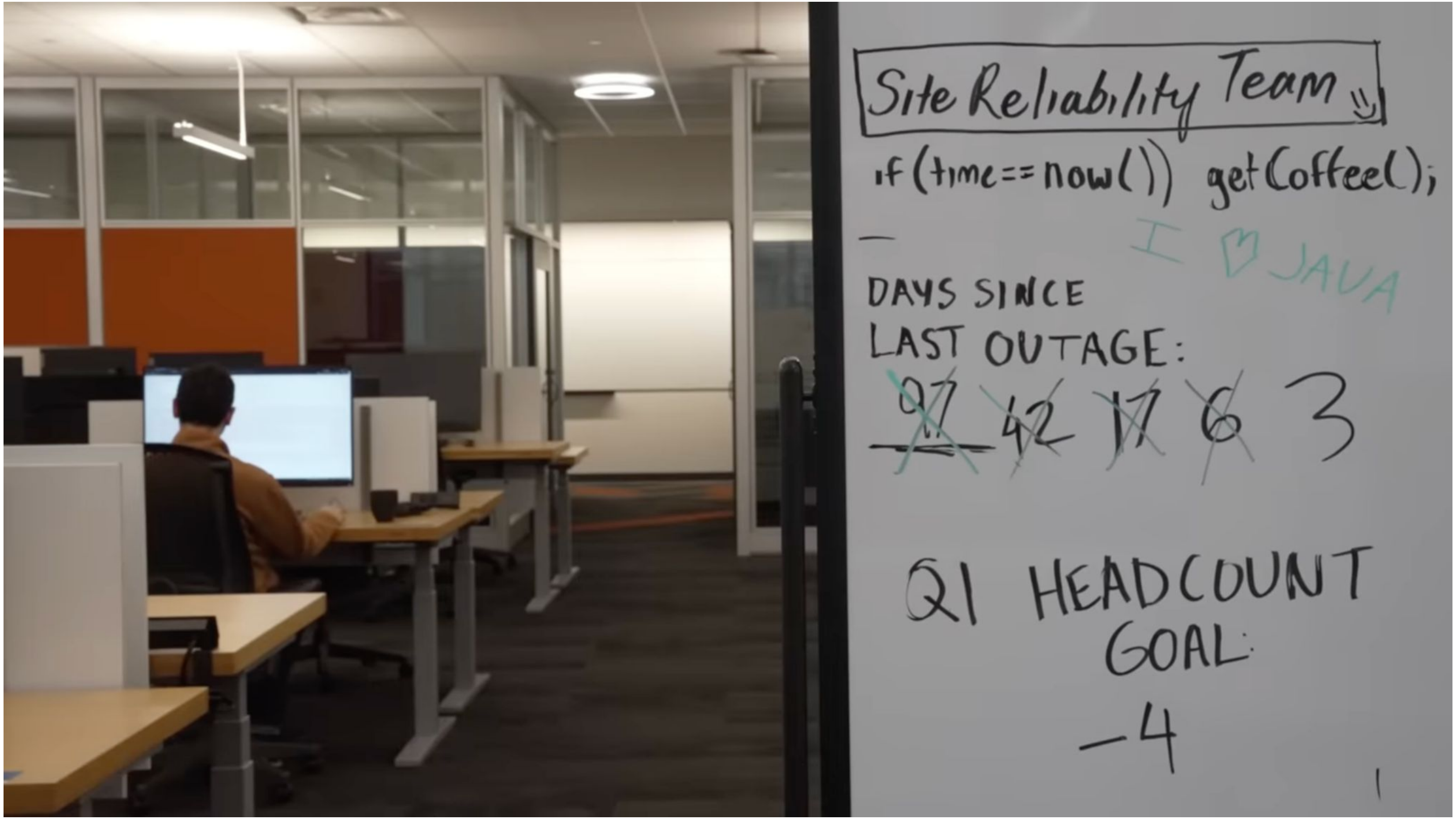
Better latency with fewer servers

avoiding RAM noisy neighbors

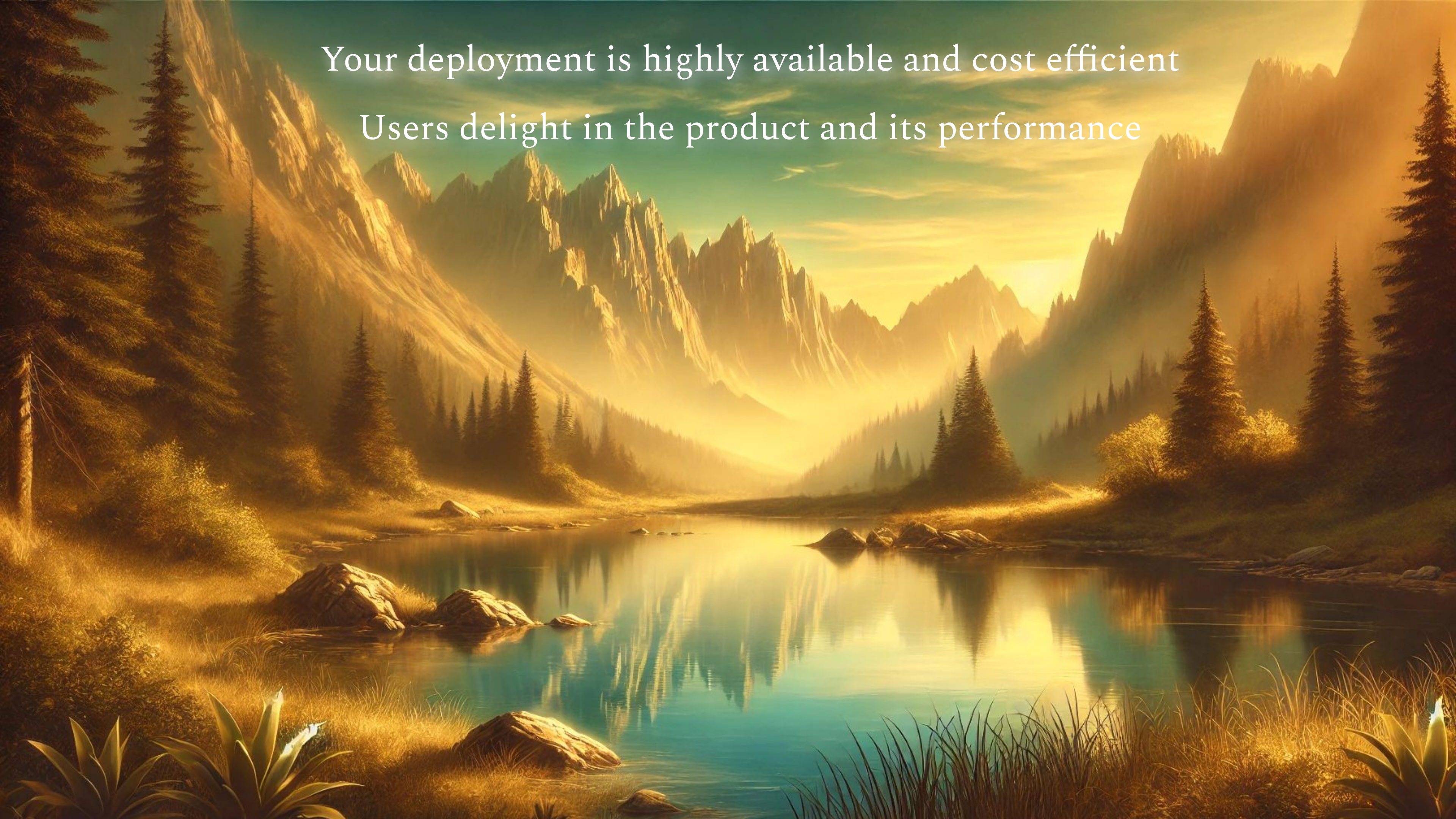
SCALE



Positive Affirmations for Site Reliability Engineers by KRAZAM (link)

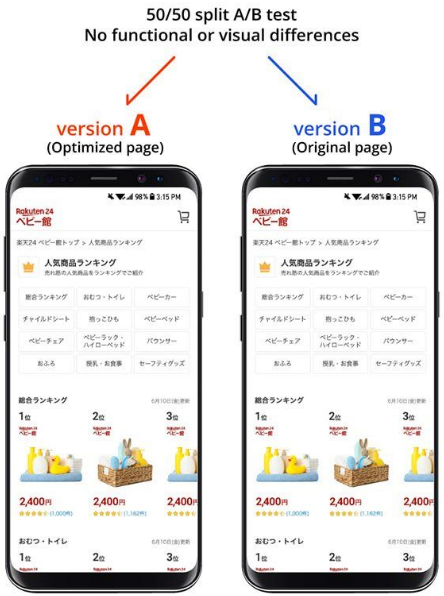
Why SRE?

Positive Affirmations for Site Reliability Engineers by KRAZAM (link)

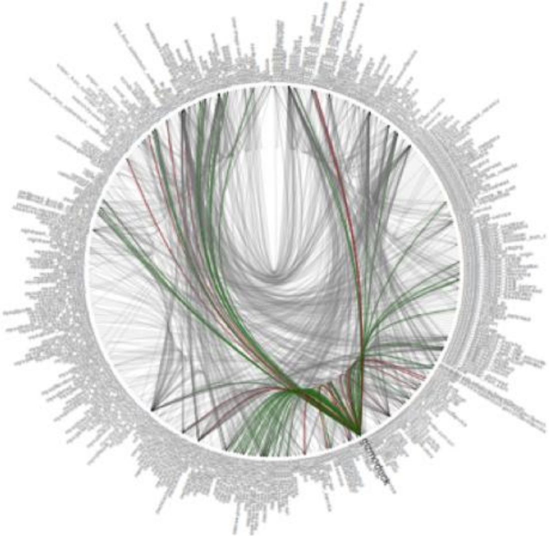

Your deployment is highly available and cost efficient
Users delight in the product and its performance

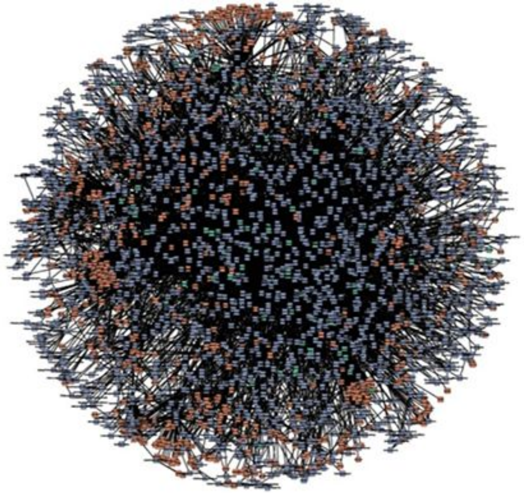
Performance matters

Company	Year	Impact	Source
Amazon	2		source
Google	2		source
Akamai	2		source
Ebay	2		source
Vodafone	2		source
Yelp	2		source
Adobe Experience Cloud	2	ce	source
Pfizer	2		source
Rakuten 24	2022	-0.4s latency → +53% revenue per visitor, +33% conversion, +15% average order value	source

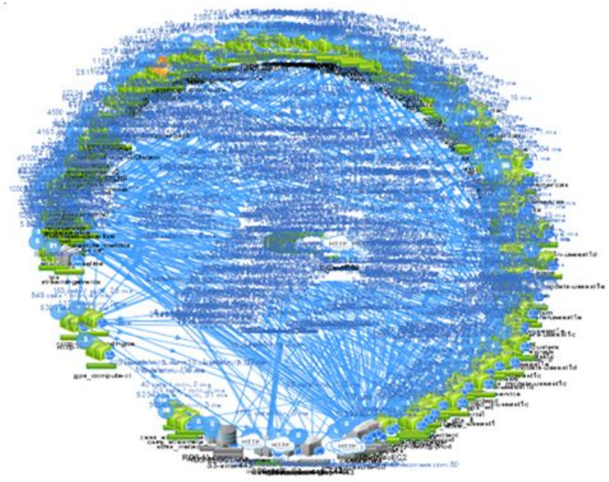


Users want more functionality



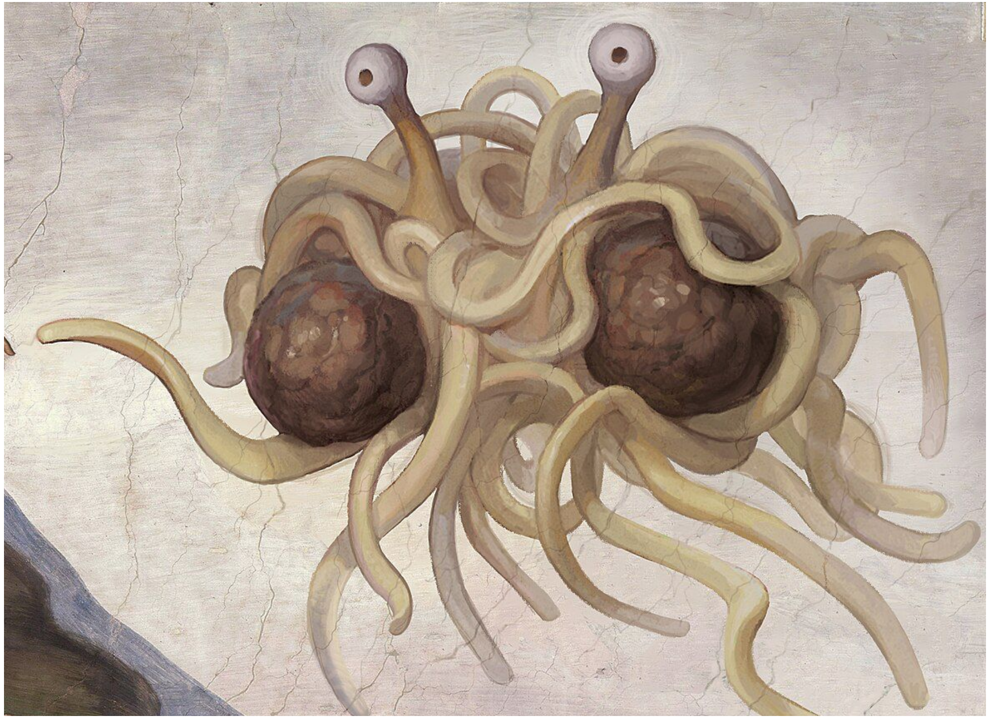


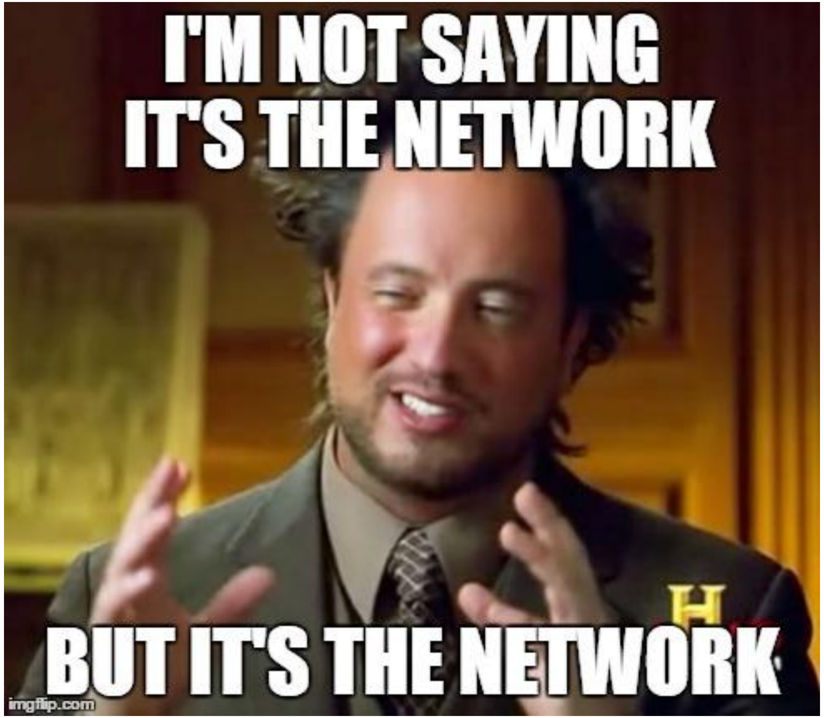
amazon.com



NETFLIX

Easy to blame "complexity"





Blame resource congestion!

Run 20%-50% more transactions

Reduce tail latency by 20-80%

2011	2013	2015	2016	2018	2019	2020	2021	2023
BubbleUp	iBench	Heracles	Dirigent	PerfIso	PARTIES	Caladan	Libra	Themis
	CPI^2		CAT at Scale			Alita		

HUAWEI

Talk Outline

The Problem	Why Solve	Available Tooling	Mitigation Systems	Current Efforts

Hi, I’m Jonathan Perry

yonch@yonch.com
@yonchco

- Ph.D.: *Network* Noisy Neighbor mitigation
- CEO @ Flowmill: Network observability
- CEO @ Unvariance

Massachusetts Institute of Technology
FLOWMILL
splunk>
OpenTelemetry

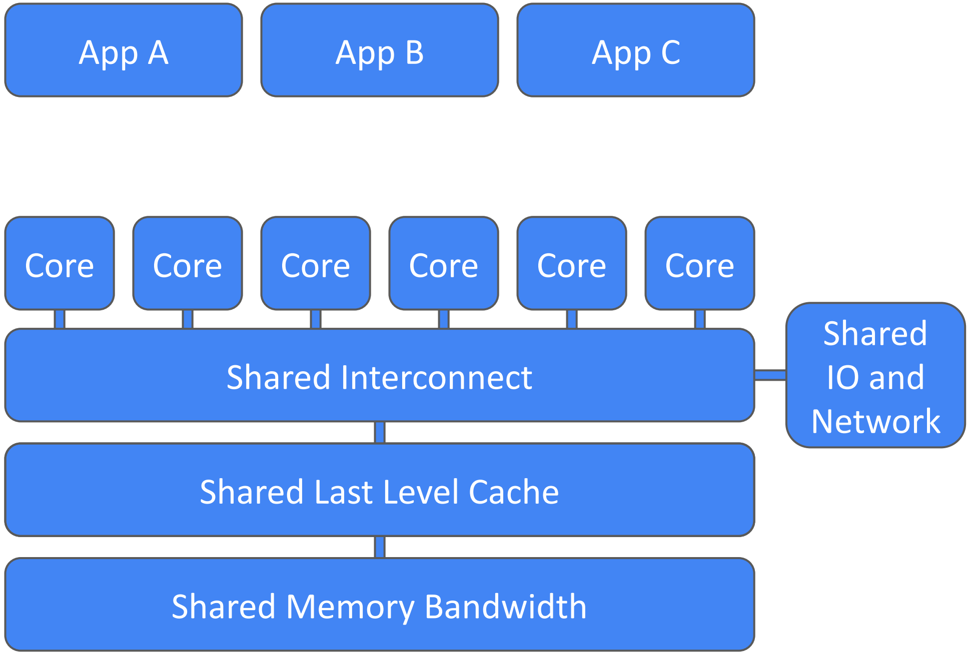
The Problem

- What is memory noisy neighbor
- How does this affect my Pods?
- Do I have it in my cluster?

Noisy neighbor: tragedy of the commons

Apps access physical resources

Shared resources are constrained



Noisy neighbor: tragedy of the commons

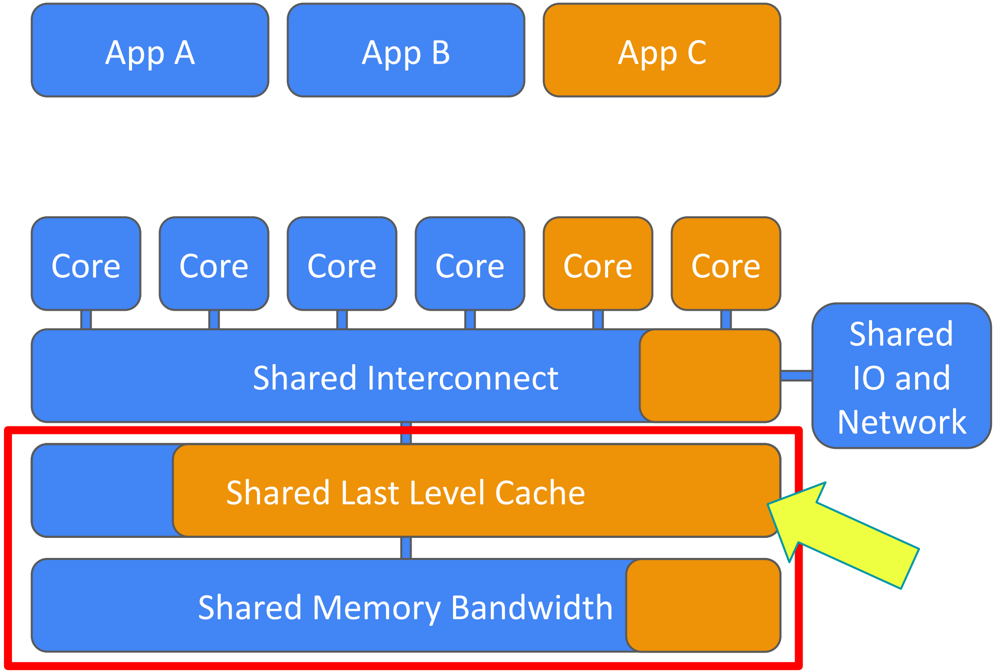
Apps access physical resources

Shared resources are constrained

One App can use more than its fair share, degrading others

This talk:

- Cache
- Memory bandwidth

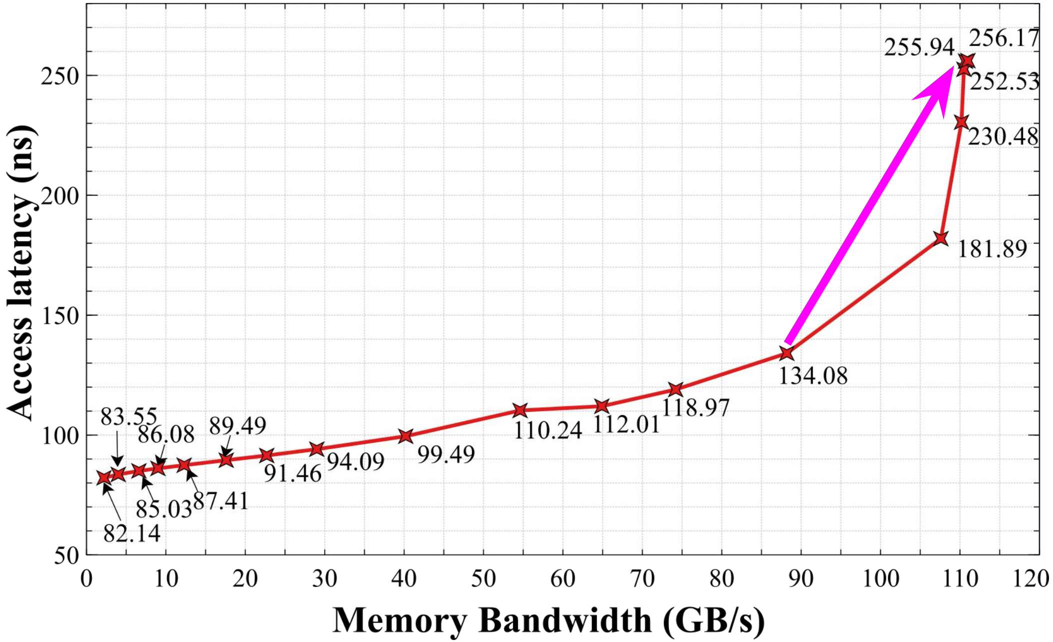


Latency increases with memory bandwidth

Change memory bandwidth,
Measure latency

Knee-point around 80%

80% → 100% bandwidth
latency doubles!



W. Tang et al, "Themis: Fair Memory Subsystem Resource Sharing with Differentiated QoS in Public Clouds," ICPP '22. doi: [10.1145/3545008.3545064](https://doi.org/10.1145/3545008.3545064).

Does memory latency cause slowdown?

CPUs try to hide memory latency:

- Prefetchers
- Reorder buffer (ROB)
- Caches

Are they effective?

We need to **measure slowdown**

Popular metric:

Cycles Per Instruction – CPI

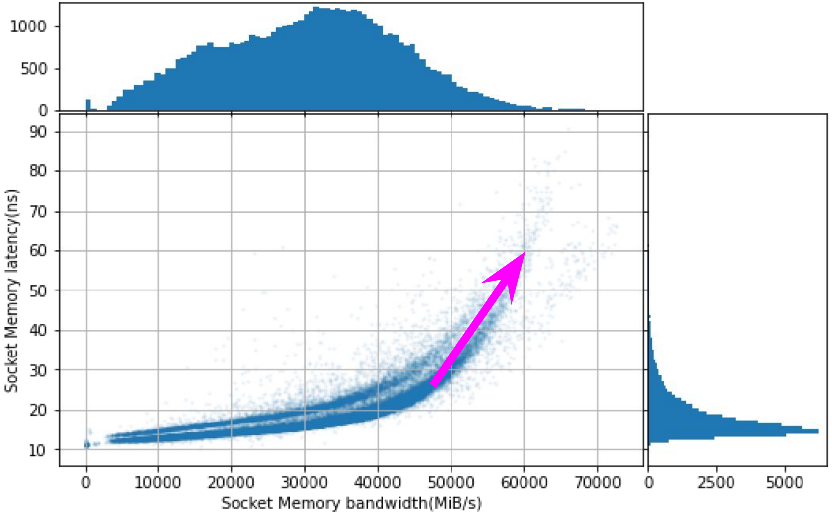
CPU waits for memory → Stall cycles

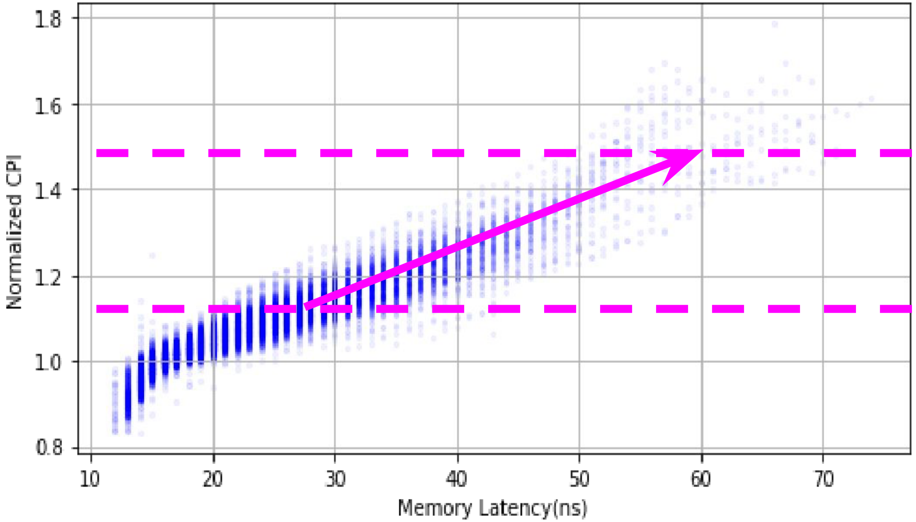
Many stalls → High CPI

Few stalls → Low CPI

80% bandwidth cap → 25% more compute-efficient

Alibaba Cloud production trace: 8k+ hosts, 1M+ containers, 24 hours

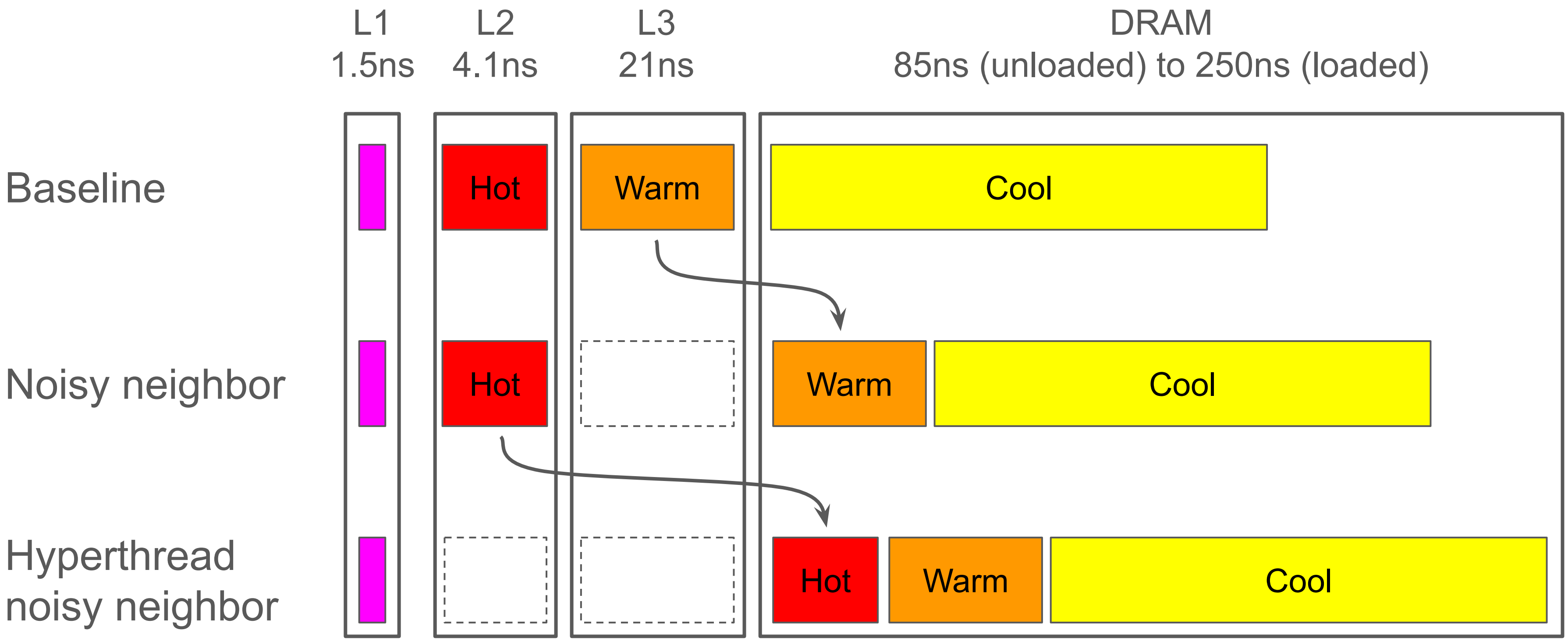




Memory bandwidth only – has cache mitigation

K. Wang *et al.*, "Characterizing Job Microarchitectural Profiles at Scale: Dataset and Analysis,", *ICPP'*22. doi: 10.1145/3545008.3545026.

Cache contention degrades access times



Cache sizes, Ice Lake SP cache benchmarks, DRAM Latency on Xeon Gold 6278

Tail latency can explode with noisy neighbors!

websearch	10%	20%	30%	40%	50%	60%	70%	80%	90%
baseline(approx)	52%	57%	61%	60%	63%	62%	66%	77%	88%
LLC(small)	103%	96%	102%	96%	104%	100%	100%	103%	103%
LLC(med)	106%	99%	111%	103%	116%	[illegible]	110%	125%	111%
LLC(big)	>300%	>300%	>300%	>300%	>300%	[illegible]	[illegible]	264%	123%
DRAM	>300%	>300%	>300%	>300%	>300%	>300%	>300%	270%	122%

> 4.75x

ml_cluster	[illegible]	20%	30%	40%	50%	60%	70%	80%	90%
baseline(approx)	[illegible]	58%	57%	59%	56%	58%	58%	60%	68%
LLC(small)	88%	84%	110%	93%	216%	106%	105%	206%	202%
LLC(med)	88%	91%	115%	104%	>300%	212%	220%	212%	205%
LLC(big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	250%	214%
DRAM	>300%	>300%	>300%	>300%	>300%	>300%	>300%	287%	223%

> 5.35x

memkeyval	10%	20%	30%	40%	50%	60%	70%	80%	90%
baseline(approx)	20%	20%	21%	21%	22%	29%	35%	42%	34%
LLC(small)	88%	91%	101%	91%	101%	138%	140%	150%	78%
LLC(med)	148%	107%	119%	108%	138%	230%	181%	162%	100%
LLC(big)	>300%	>300%	>300%	>300%	>300%	>300%	280%	222%	79%
DRAM	>300%	>300%	>300%	>300%	>300%	>300%	>300%	234%	103%

> 13.6x

Google, 2015

3 production services

get % of SLO target
(99th / 95th percentile)

w/ synthetic noise generators

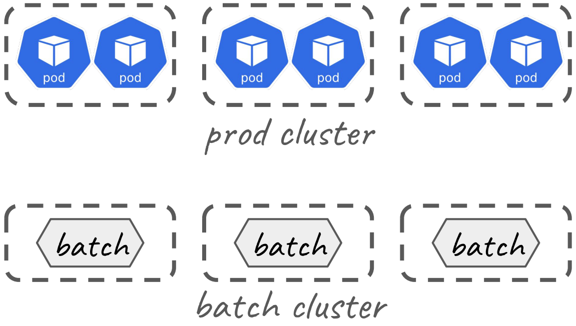
D. Lo et al. “Heracles: improving resource efficiency at scale,”, ISCA ’15. doi: 10.1145/2749469.2749475.

Survey: Node size

Please raise hand if:

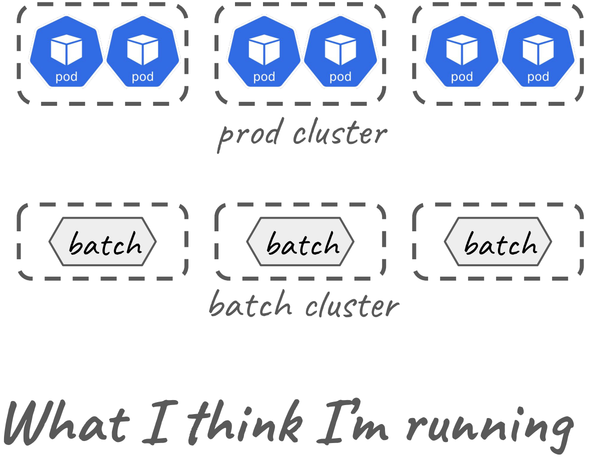
- Know what VMs or bare-metal are used in prod
- Never use fraction of physical CPU

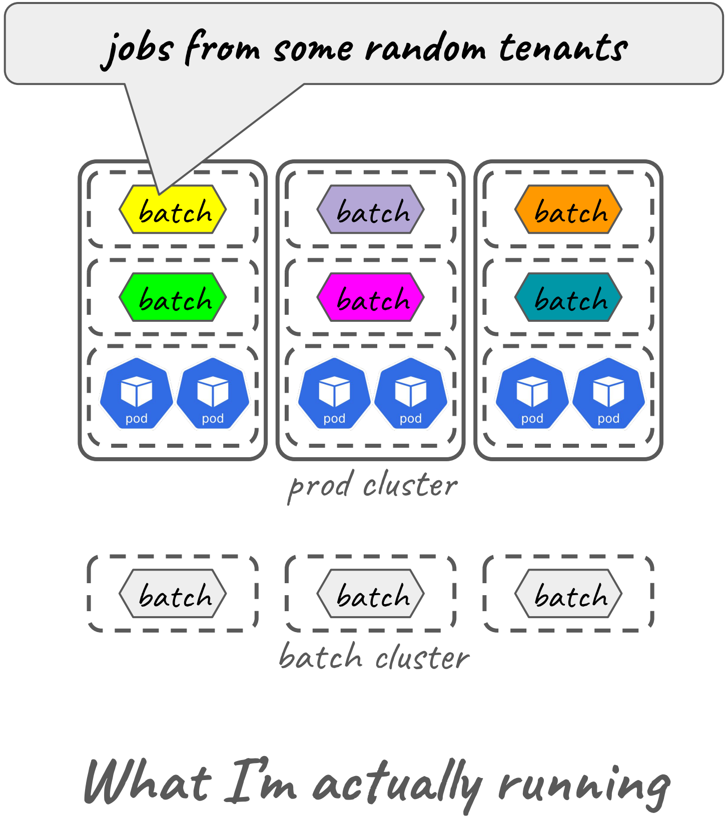
Separating batch clusters



What I think I'm running

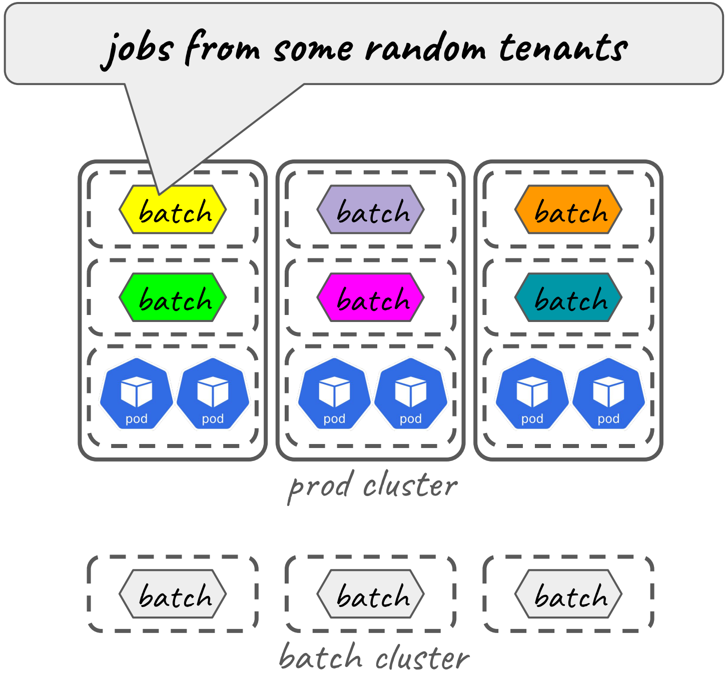
Separating batch clusters





Separating batch clusters

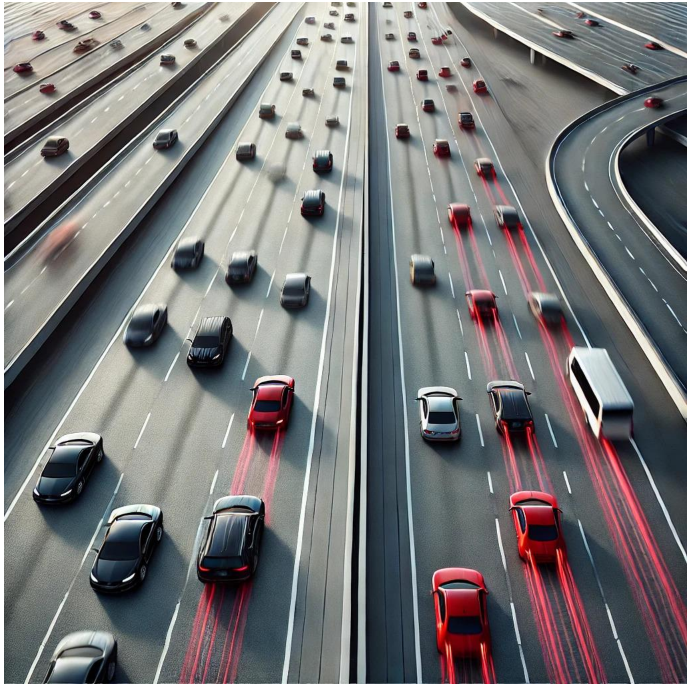
Does your provider protect your VMs from other VMs on the same machine?



What I'm actually running

Engineers spend years optimizing user experience

Bare-metal: no cross-tenant noisy neighbor



A few 100-core machines

>

Lots of 8-core VMs

* Need enough servers to handle server failures ("~10 servers")

Does my cluster have noisy neighbors?

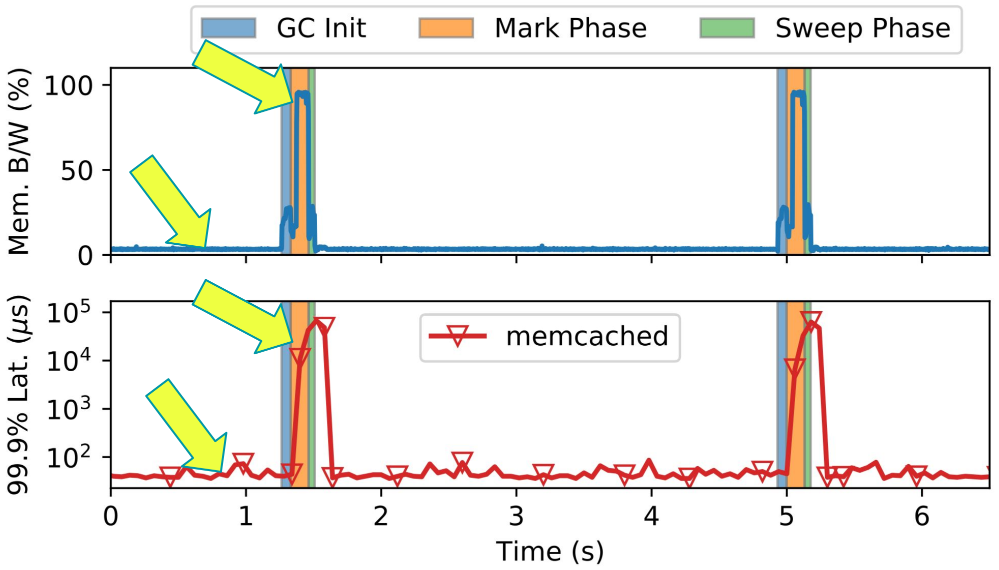
Run:

- Memcached
- garbage-collected workload

Mark Phase is memory-intensive, causes significant slowdown!

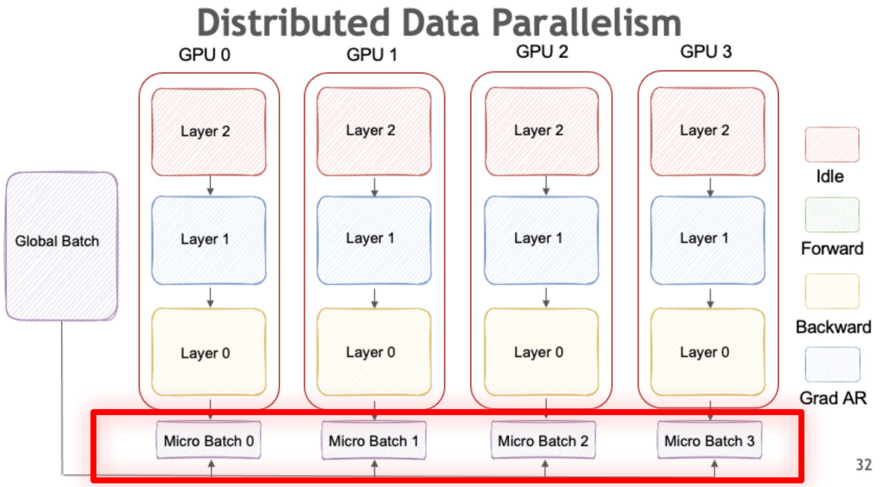
Not only "big-data" is noisy

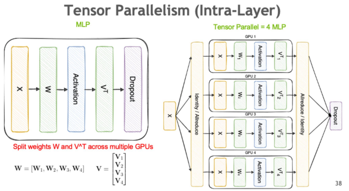
Also: security scanning, video streaming, transcoding…

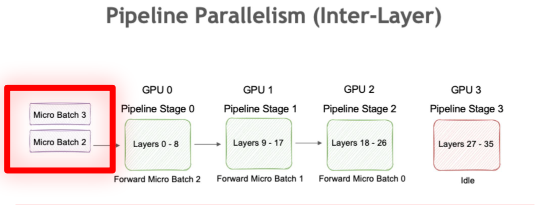


J. Fried et al., "Caladan: Mitigating Interference at Microsecond Timescales,". OSDI'20. [on usenix.org]

Implications for AI Workloads







CPU at **training**:

- Read data
- Pre-process
- Transfer micro-batches → GPU

CPU at **inference**:

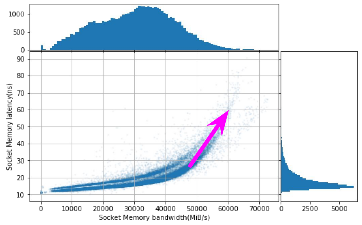
- Input → GPU
- Output ← GPU
- Tokenization, batching
- User communication

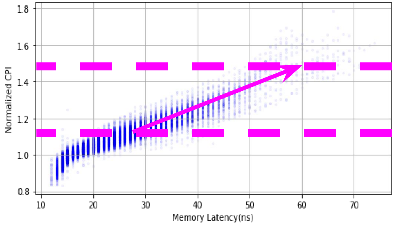
These compete for memory bandwidth

Looking for use-cases

Recap: The Problem

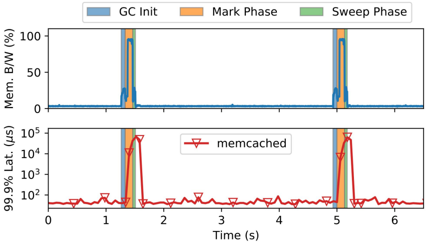
High memory contention
→ High memory latency
→ High CPI (low efficiency)





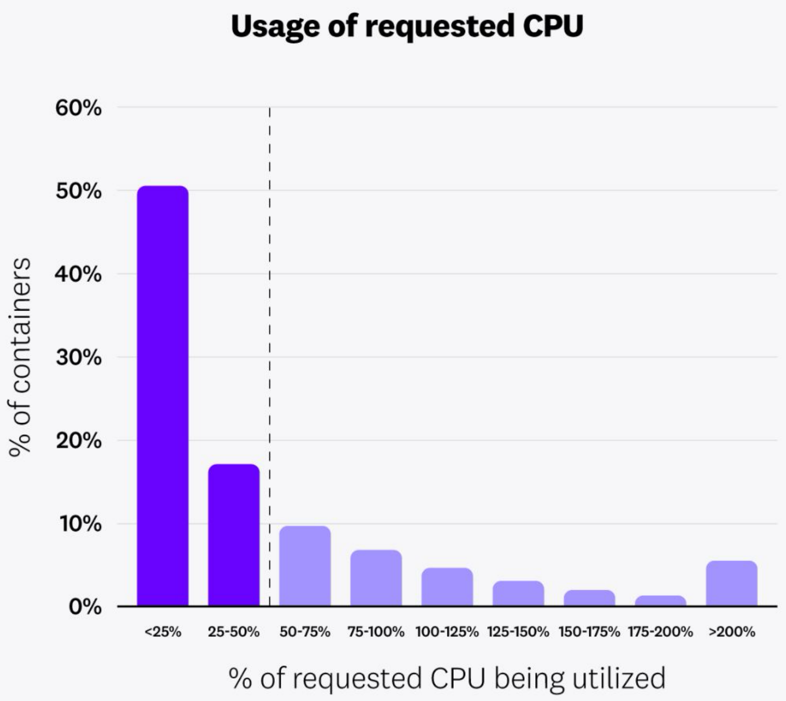
Tail latency with noisy neighbor increases 4-13x

Many workloads can be noisy (e.g., garbage collection)



Why Solve: Benefits

- State of cluster utilization
- How tail latency affects utilization
- Leveraging reduced tail latency



>65% OF CONTAINERS USE LESS THAN HALF OF REQUESTED CPU

Source: Datadog

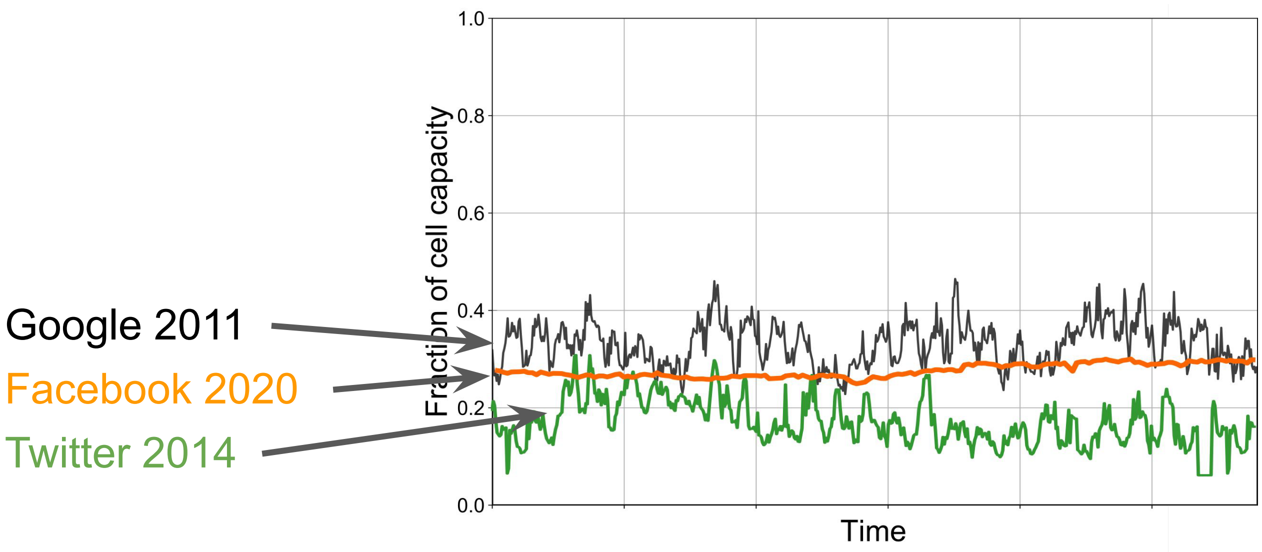
source

Survey: cluster utilization

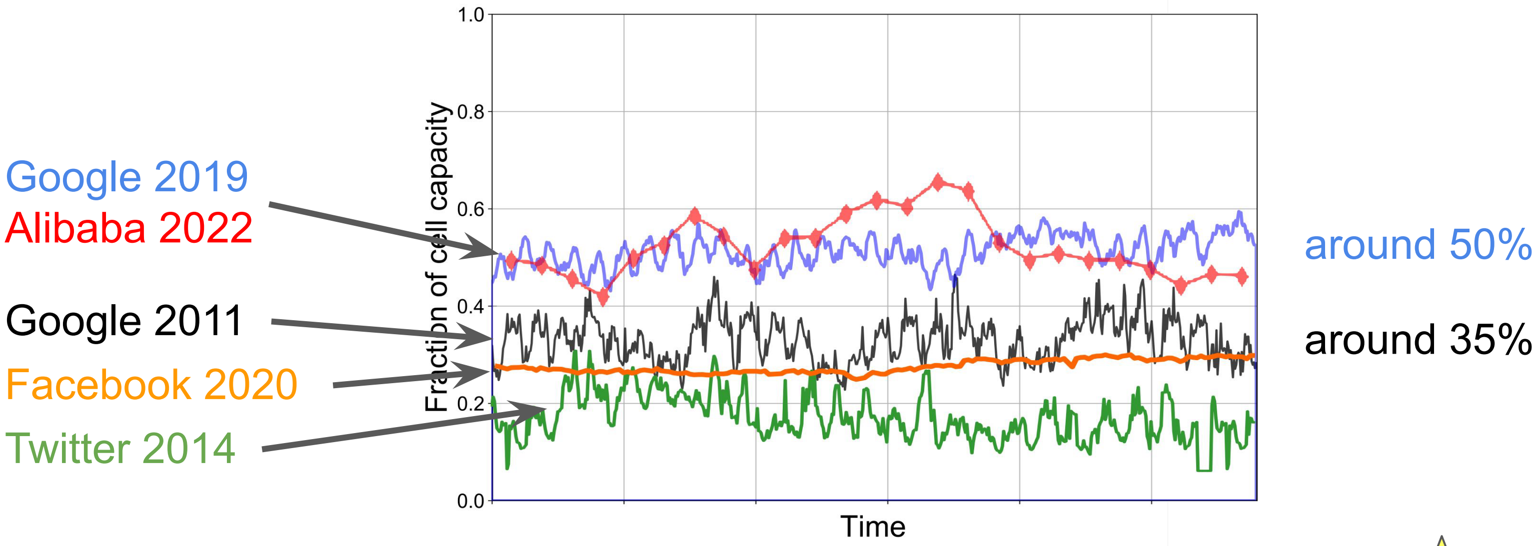
Raise hand if:

- Know prod cluster avg. CPU utilization
- Above 20%?
- Above 30%?
- Above 40%?
- Above 50%?

Hyperscaler CPU was low…



Then.. breakthrough?



Vertical autoscaler

Google's Autopilot (EuroSys'20, not GKE)
StormForge, PerfectScale, FairWinds, …

Better packing → Less wasted resources

Handle noisy neighbor

- Reduce cycles per request
- Improve tail latency

This talk

→ Improved SLA at higher utilization

Tuning of Scaling Behavior

Adjusting scaling:

- Manual, (`spec.replicas`)
- HPA (Horizontal Pod Autoscaler)
  - CPU Utilization
  - Requests per second
  - Memory Utilization

How do we set thresholds?

So we meet SLOs

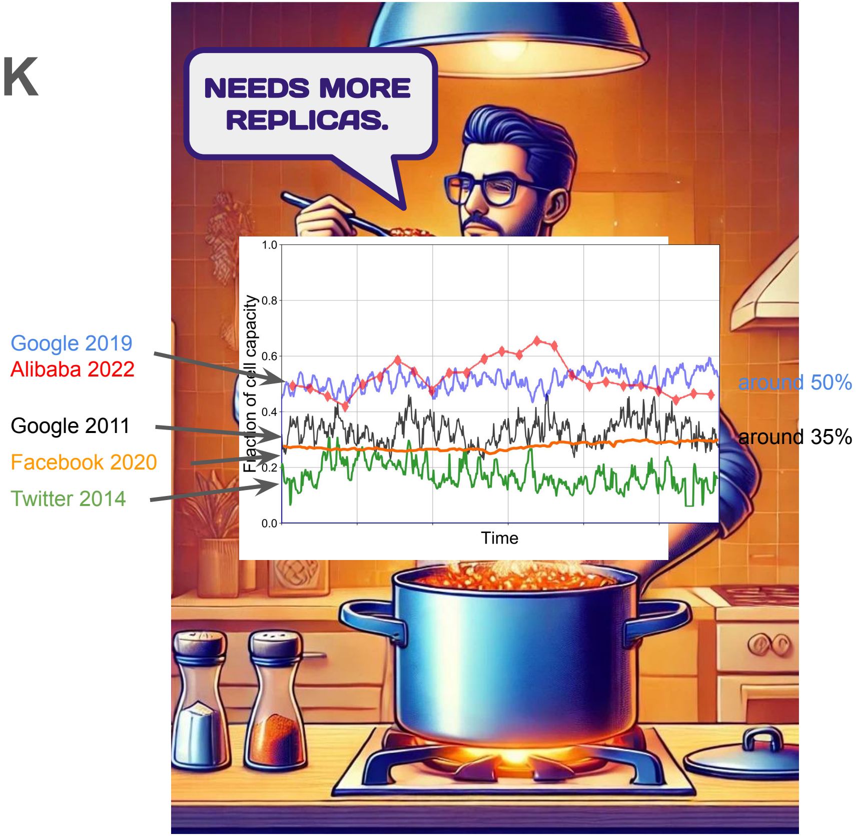
Improve SLOs → Higher utilization OK

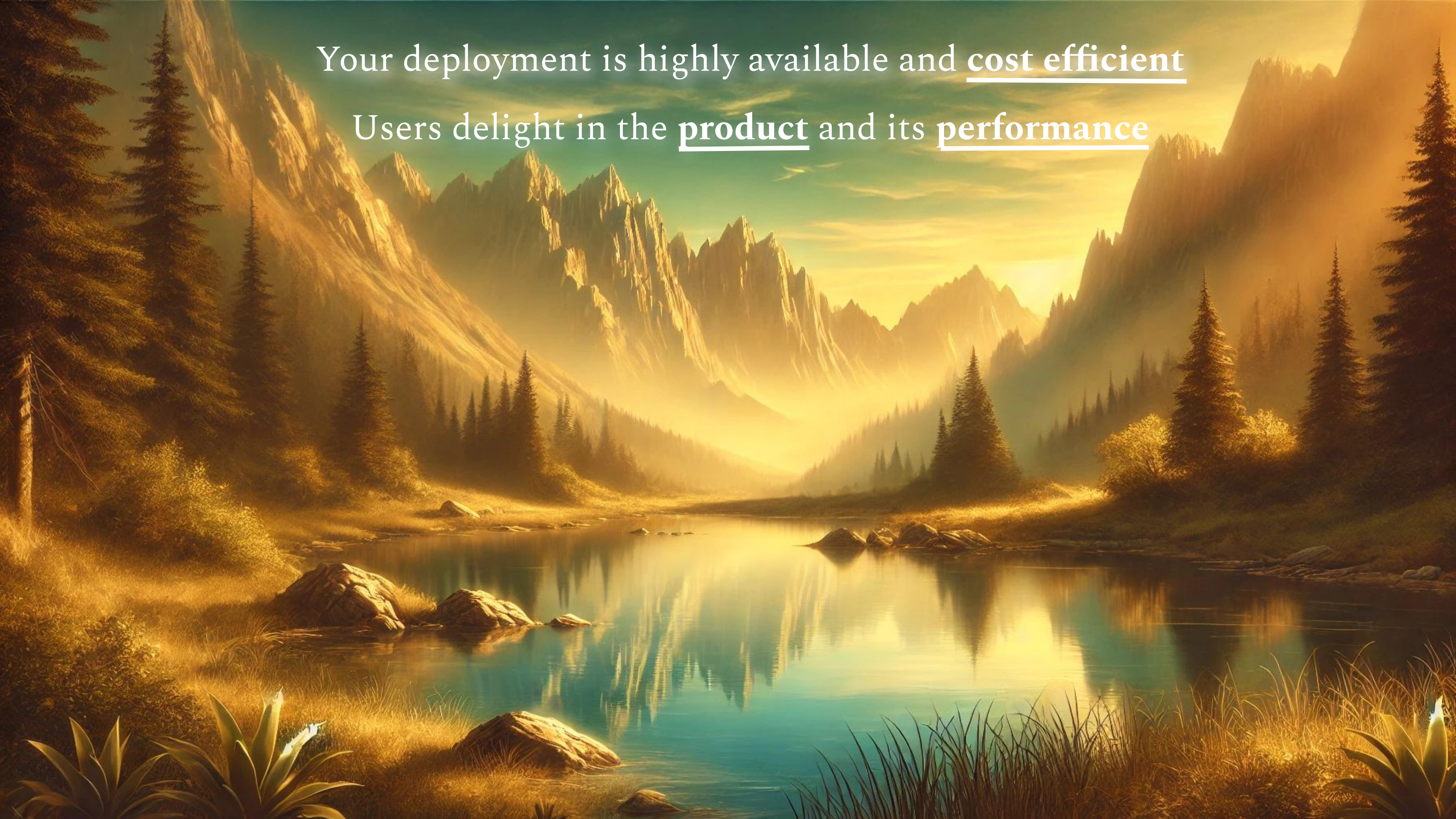
[Kubernetes.io HPA walkthrough](), [Kubecost](), [OpenShift HPA docs]()

Tuning of Scaling Behavior (cont'd)

Improve SLOs → Higher utilization OK

- Reduce cost
  - same functionality
  - fewer replicas (at higher utilization)
- More features / business outcomes
  - added functionality
  - same replicas (at higher utilization)
- Improve SLO
  - Same functionality
  - Same replicas (same utilization)

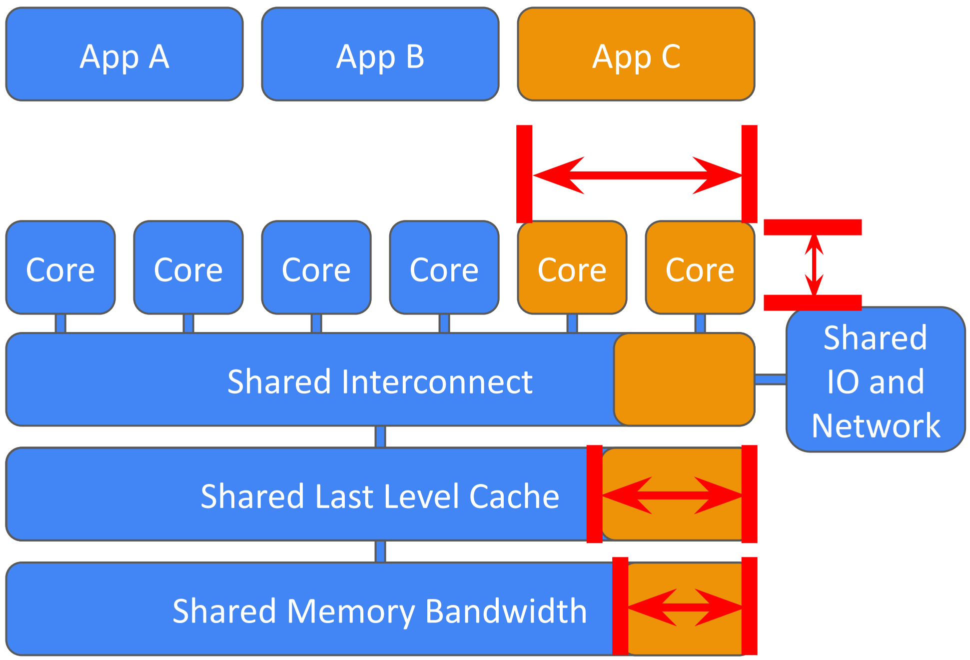



Your deployment is highly available and **cost efficient**
Users delight in the **product** and its **performance**

Available Tooling

- Hardware and software mitigation
- Hardware interfaces, limitations
- Support in Linux

Memory system mitigation knobs



Reduce demand (#cycles):

- Core pinning
- Frequency scaling

Direct control:

- Cache allocation
- Memory bandwidth limits

Direct Control: Intel RDT's Interface

- CPU performs allocation using identifiers
  - For measurement, **RMID**s (Resource Measurement ID)
  - For enforcement, **CLOS**s (Classes of Service)
- On every context switch, OS sets thread's (CLOS, RMID)
- Can read RMID's memory bandwidth, cache occupancy
- On EC2's m7i.metal-24xl:
  - 448 RMIDs
  - 15 CLOSs

  Limited
- Supports GPU-DRAM bandwidth
  - **Intel RDT for Non-CPU Agents**

Containers (not) to the rescue!

Different focus:

- cycles
- memory capacity (size)

Cgroup v2	Responsibility
CPU	Regulates distribution of CPU cycles
Memory	Controls distribution and accounting of memory usage
IO	Manages distribution of IO resources
PID	Limits the number of processes in a cgroup
Cpuset	Assigns specific CPU(s) and memory nodes
Device	Controls access to device files
RDMA	Regulates distribution and accounting of RDMA resources
HugeTLB	Limits the HugeTLB usage per control group
Perf_event	Allows perf events to be filtered by cgroup path
Misc	Provides resource limiting for scalar resources not covered by other controllers

Reading 1 2

Linux supports allocation

~~cgroup~~ resctrl allocates:

- Memory bandwidth
- Cache space

Separate subsystem:

- Maintainers wanted self-tuning like cpuset
- Limited number of CLOS, RMIDs
- API through /sys/fs/resctrl

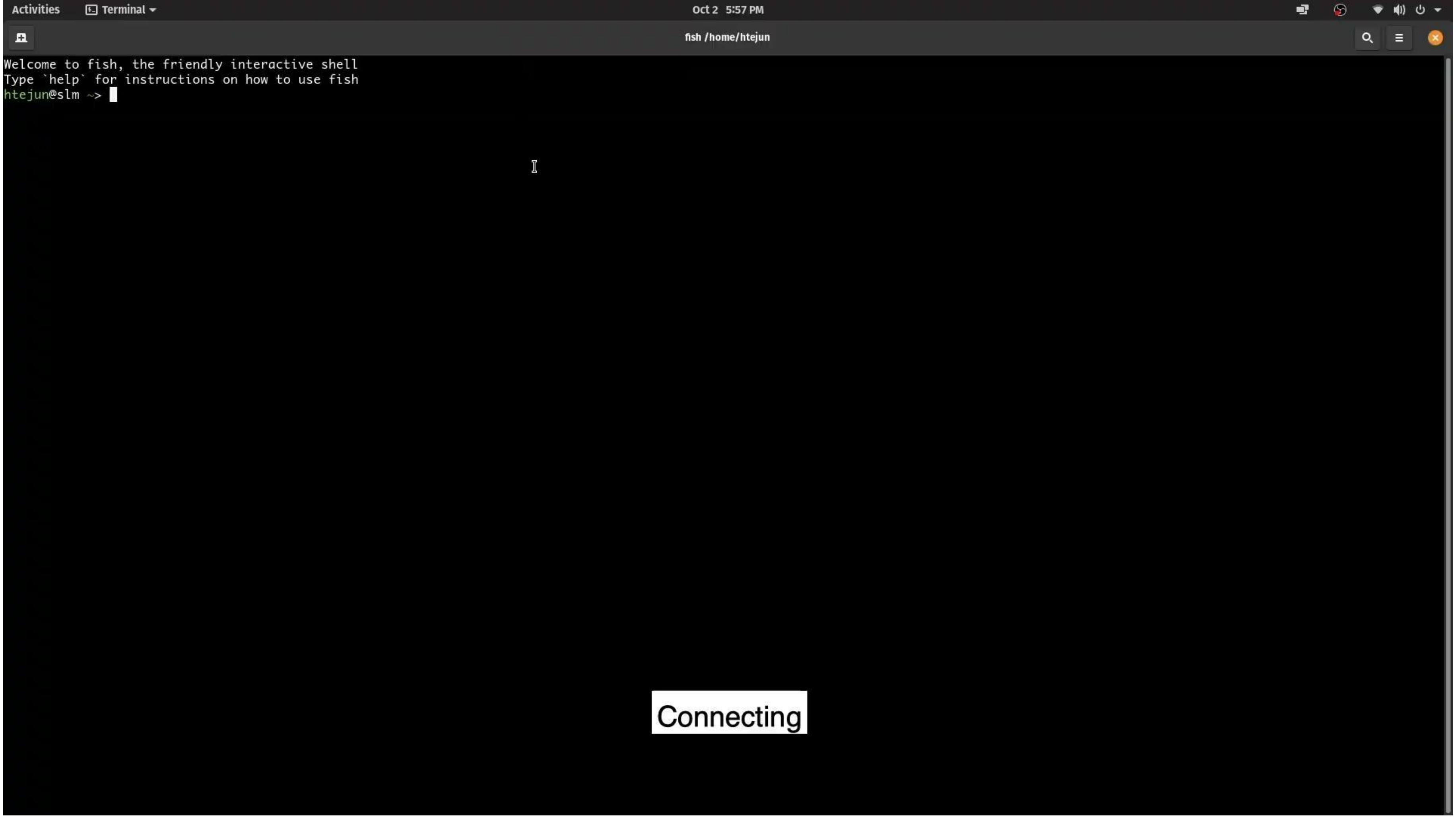
Demo:

- Facebook Resource Control Demo
  by Tejun Heo and collaborators

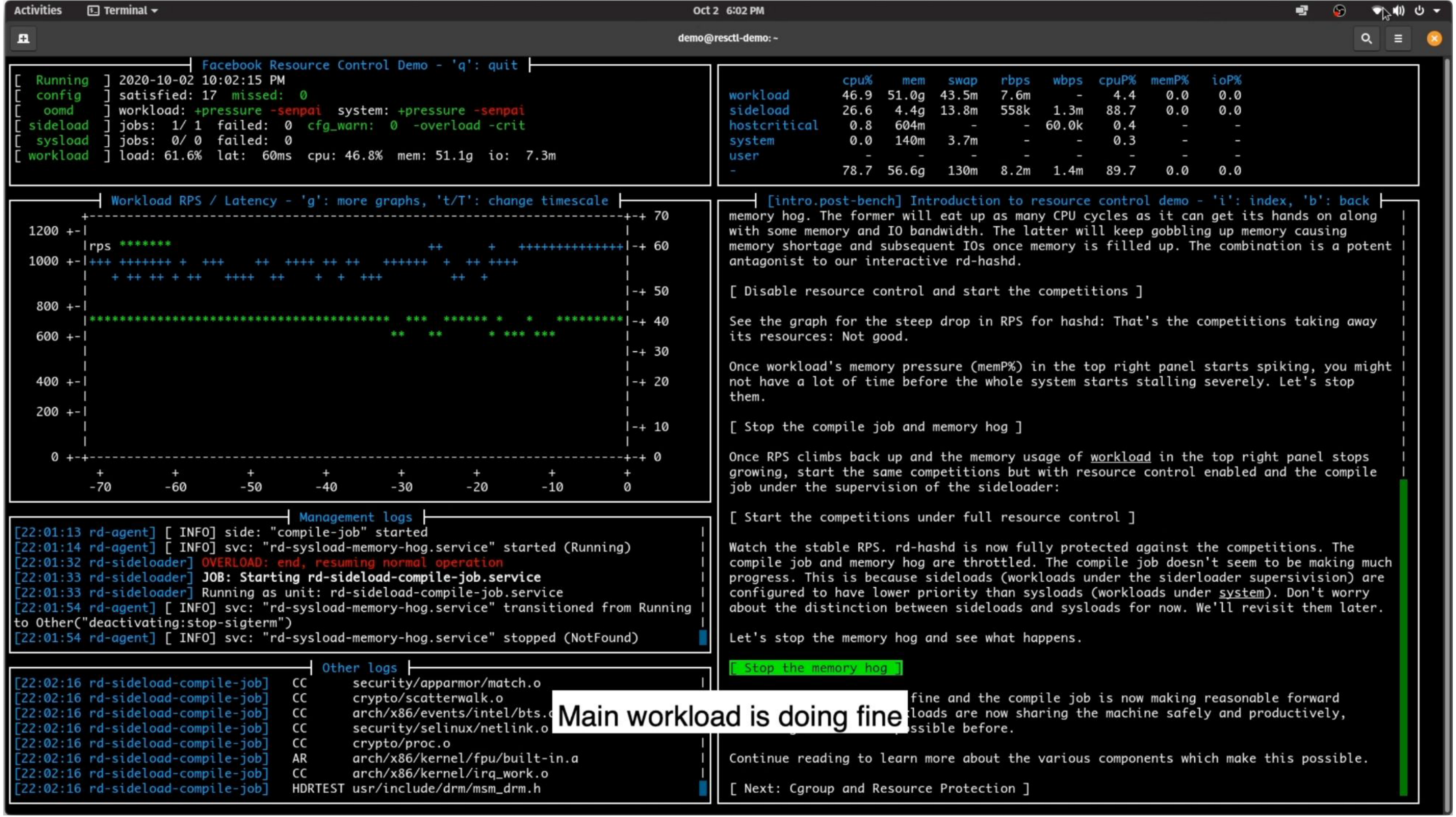
Linux resctrl CPU support

	Kernel
Intel RDT Intel Resource Director Technology	v4.10, v4.12 (2016)
AMD QoS AMD Platform Quality of Service	V5.0 (2018)
ARM MPAM Memory System Resource Partitioning and Monitoring	WIP, x86→generic Review needed

AMD, ARM, Intel “Linux kernel interface” discussion, lkml discussion



Credit: Meta; Tejun Heo and collaborators. [resctl-demo on GitHub](#).



Credit: Meta; Tejun Heo and collaborators. resctl-demo on GitHub.

Hypervisors support allocation

- VMware in Telco Cloud Automation

  → Telco + Finance?

- Static

Intel RDT support in hypervisors

	Cache partitioning	Memory bandwidth
Xen	✔	✔
KVM	✔	
VMware	✔	Monitor (vSphere)
Hyper-V		Monitor PMU
ACRN	✔	✔

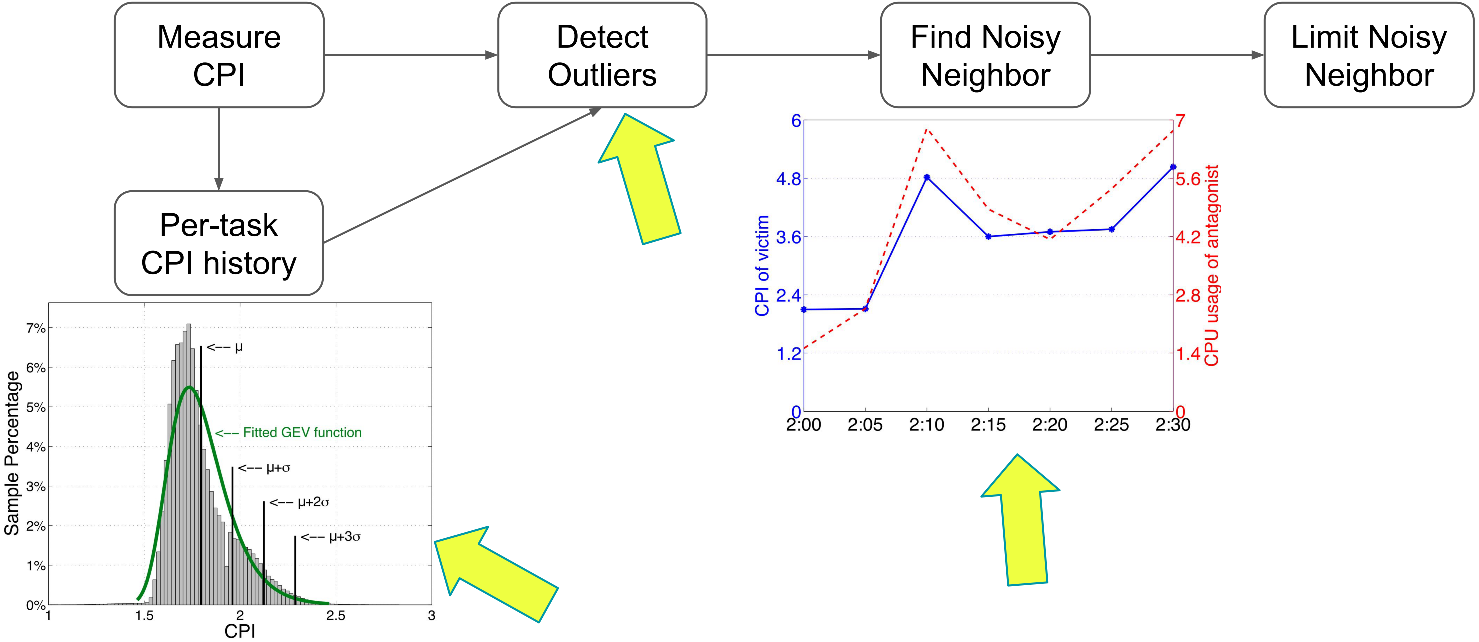
Telco Cloud Automation, KVM libvirt NUMA tuning, Xen memory bandwidth, Xen LLC, ACRN, Hyper-V PMU

Mitigation Systems

- Type 1: cycles per instruction (CPI)
- Type 2: latency control
- Type 3: usage control

Type 1: Cycles Per Instruction (CPI)

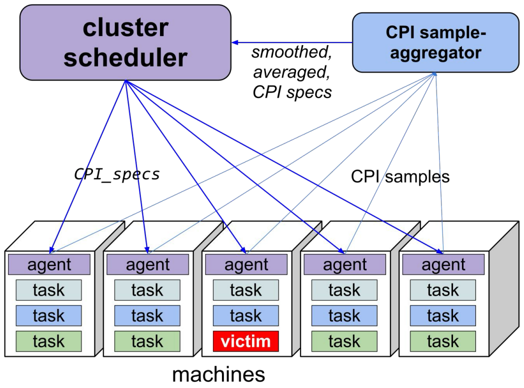
Uses: High interference → high CPI



X. Zhang et al., "CPI [2]: CPU performance isolation for shared compute clusters," in *EuroSys 2013*. doi: 10.1145/2465351.2465388.


I'll say when to stop!

Type 1: Cycles Per Instruction



“We have rolled out CPI² to all of Google’s shared compute clusters.” – paper authors, 2013 @Google

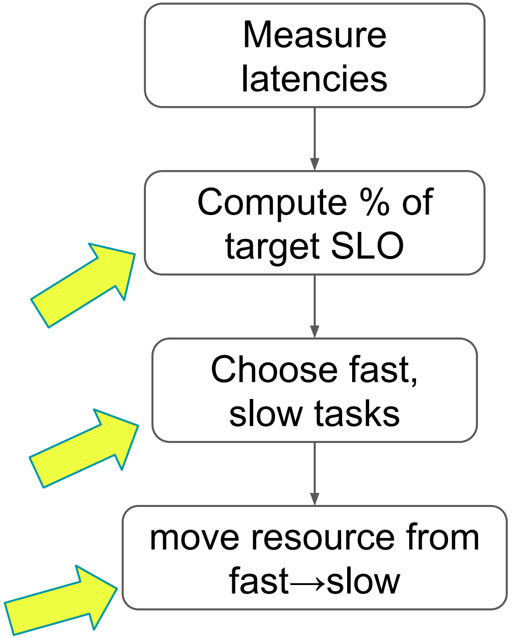
	Type 1: CPI
Measurement	Cycles, Instructions
Averaging	High
Cluster components	Aggregator
Pros	● Simple to measure
Cons	● Complex deployment ● Averaging → Slow reaction

X. Zhang et al., “CPI ²: CPU performance isolation for shared compute clusters,” in *EuroSys 2013*. doi: 10.1145/2465351.2465388.

Type 2: Latency Control

Use *application* latency

Example algorithm:



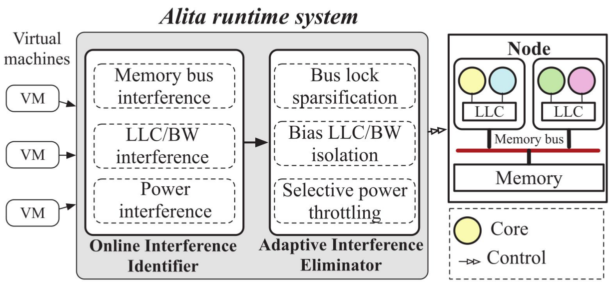
	Type 2: Latency
Measurement	App latency
Averaging	Medium
Cluster components	Node only
Pros	• No profiling → Node-local • Control what you care about
Cons	• High developer effort • Noisy signal → Averaging

S. Chen et al. “PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services,” in ASPLOS ’19. doi: 10.1145/3297858.3304005.

Type 3: Usage Control

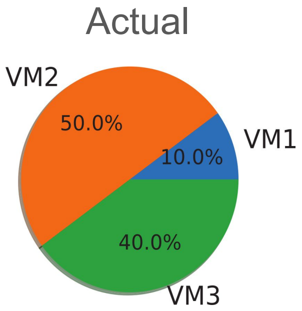
How:

- Measure per-app resource usage
- Find unfair allocation
- Limit offender

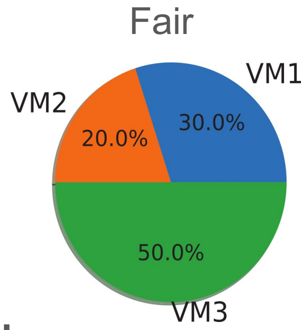


Cache example:

If similarity (



,



) < 0.8,

then limit cache polluters to their fair share

Q. Chen *et al.*, “Alita: Comprehensive Performance Isolation through Bias Resource Management for Public Clouds,” in *SC20:* link.

Do we really want a fair allocation?

Type 3: Usage Control (cont'd)

	Type 3: Usage
Measurement	CPU counters
Averaging	Low
Cluster components	Node only
Pros	• Node-local • Measure directly → Fast reaction → Easy to reason
Cons	• Do we really want fair allocation?

Deployed in production, > 2 years:

- 30k nodes (24 to 48 cores each)
- 250k VMs

- authors, 2020 @Alibaba Cloud

Q. Chen *et al.*, "Alita: Comprehensive Performance Isolation through Bias Resource Management for Public Clouds," in *SC20:* link.

Summary: Mitigation Systems

	Type 1: CPI	Type 2: Latency	Type 3: Usage
Measurement	Cycles, Instructions	App latency	CPU counters
Averaging	High	Medium	Low
Cluster components	Aggregator	Node only	Node only
Pros	• Simple to measure	• No profiling → Node-local • Control what you care about	• Node-local • Measure directly → Fast, Easy to reason
Cons	• Complex deployment • Averaging → Slow reaction	• High developer effort • Noisy signal → Averaging	• Do we really want fair allocation?

- The three “categories” of published systems
- Many good ideas → general purpose
- Usage control (#3) is promising

Recap

- Cache and bandwidth crunch can increase tail latency 4x - 13x
- Reducing tail latency: cost⬇️, functionality⬆️, response times⬇️
- CPUs support monitoring and allocation, Linux has resctrl
- Systems that explicitly monitor and allocate: simple, fast reaction

Current Efforts

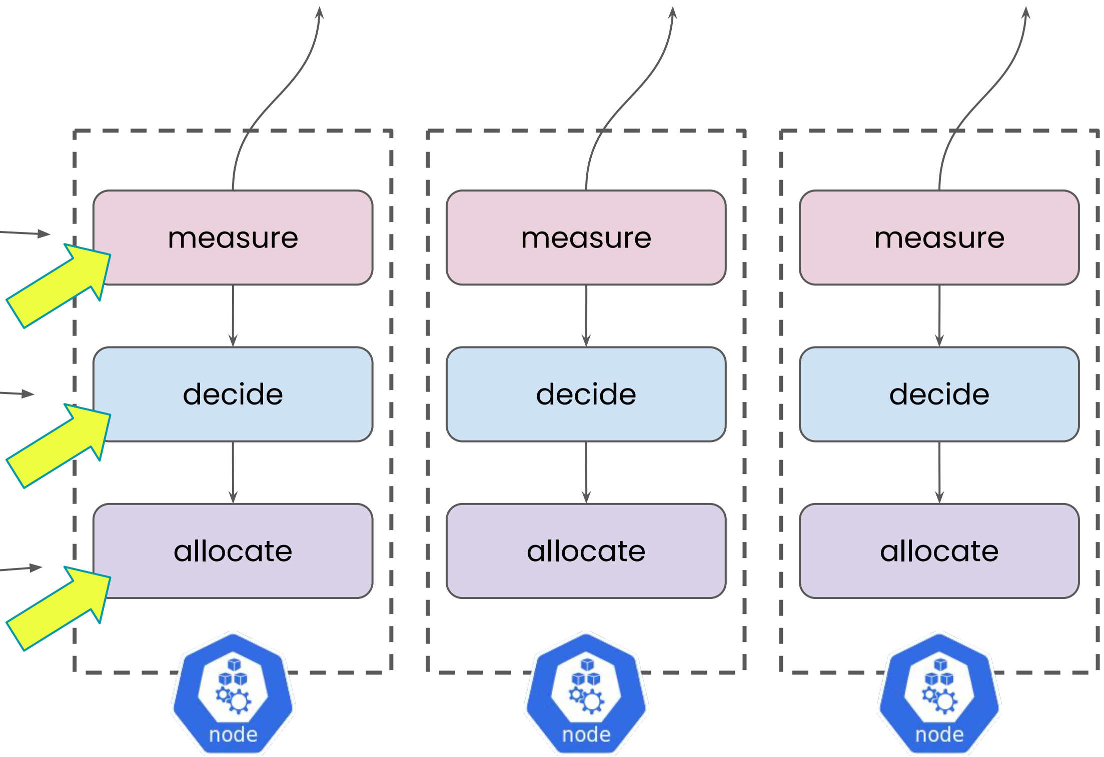
- Kubernetes Deployment
- Major Challenge
- Collector architecture
- Goals, Open Questions
- Community Next Steps

Kubernetes Deployment

Measure resource usage
(memory bandwidth, cache)

Find applications
exceeding their fair share

Adjust allocations using
Intel RDT / core pinning



Traditional Observability: Too Slow!

Many collectors measure at 5 second intervals or more

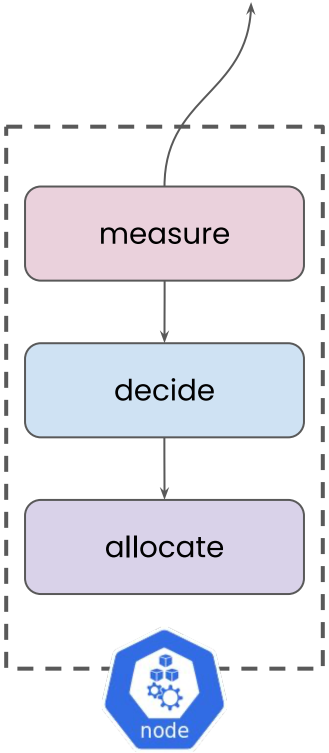
Garbage collection:

- Differs by language, application, heap size
- Minor GC: “1-10 milliseconds, every 0.1-10 seconds”
- Major GC: “10-100 milliseconds, every 10-100 seconds”

Heavy user transaction:

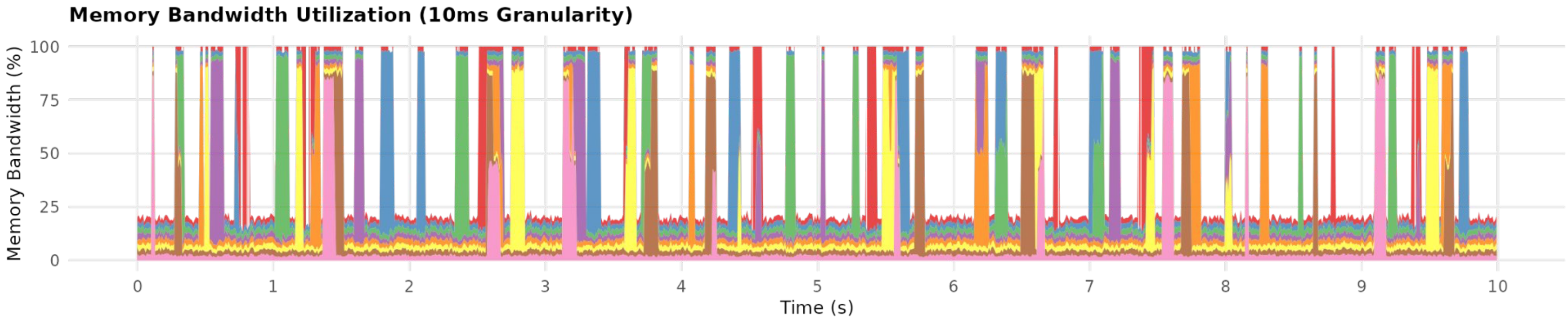
- “500 milliseconds every 10s of seconds”

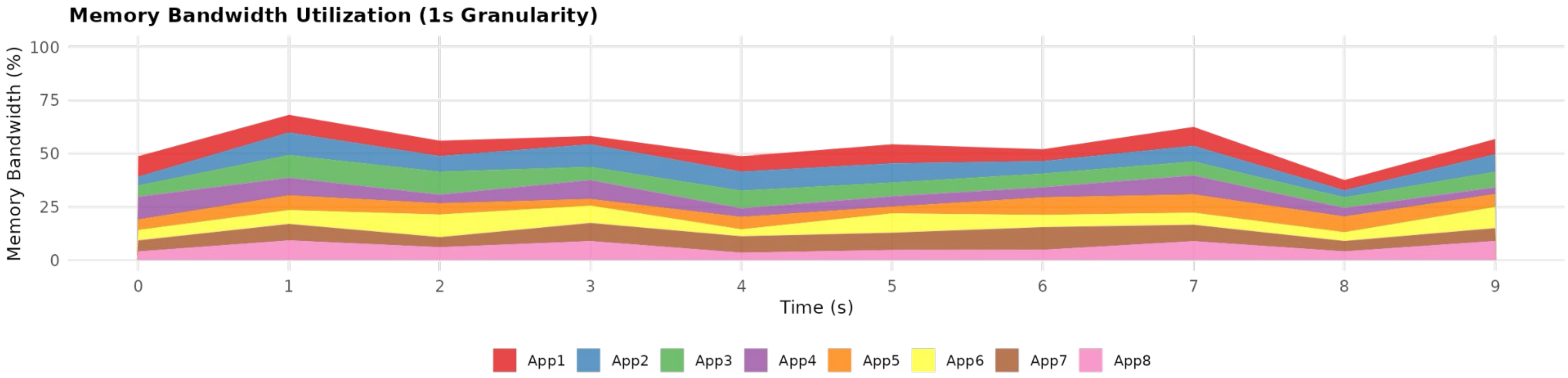
5 seconds is too slow!



Why we need frequent measurements

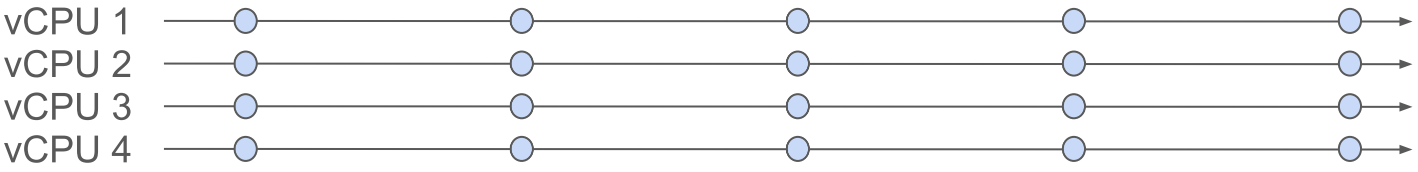
Simulation: 8 applications, each noisy for 10-100 ms every second



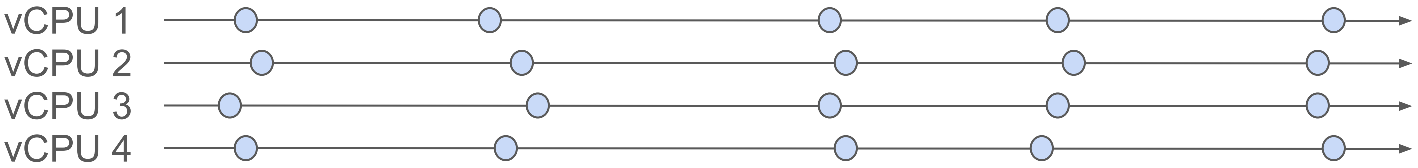


Measuring at 1 millisecond intervals

Ideally:



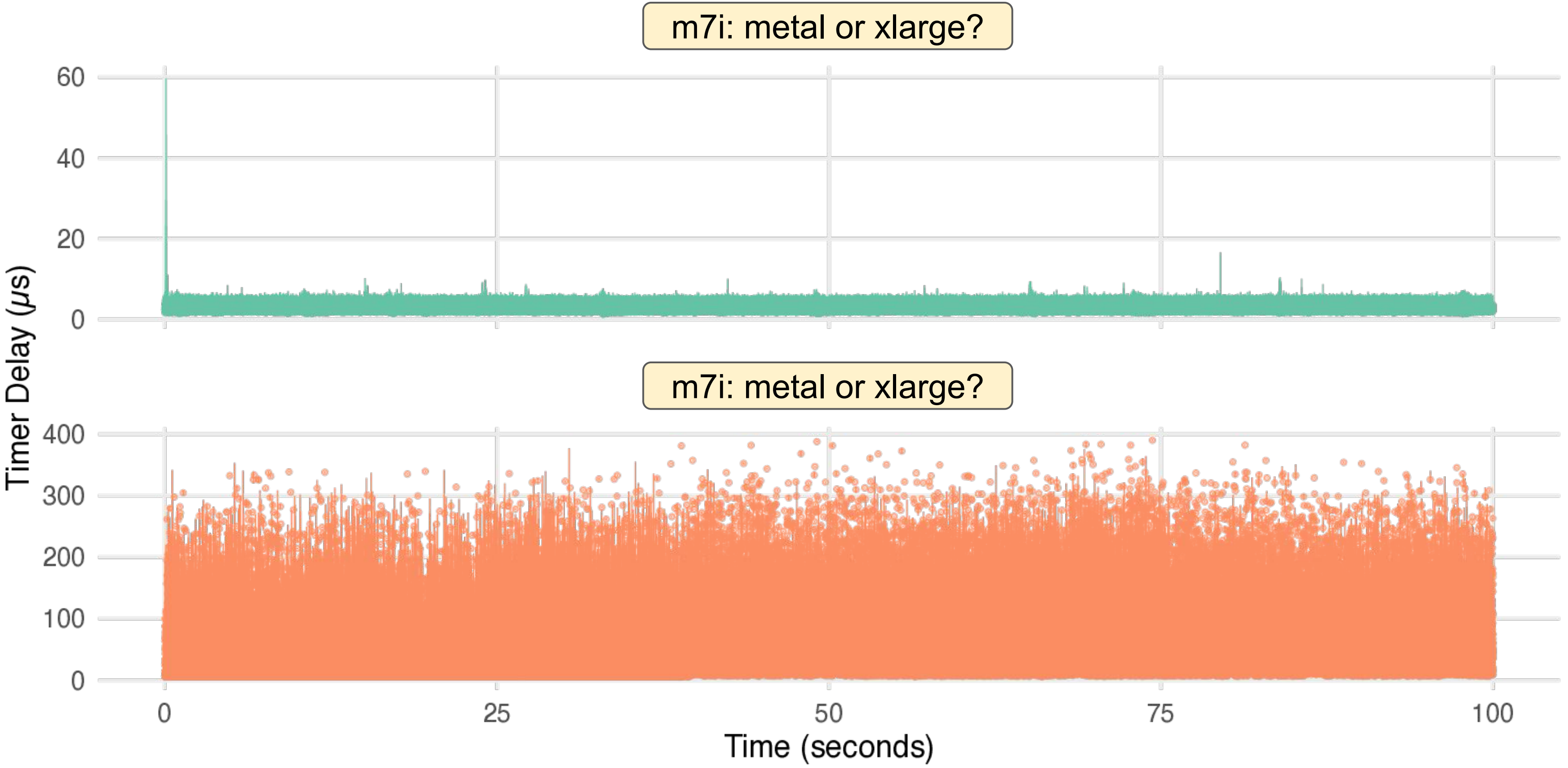
In practice:



Q: How much jitter are we going to have?

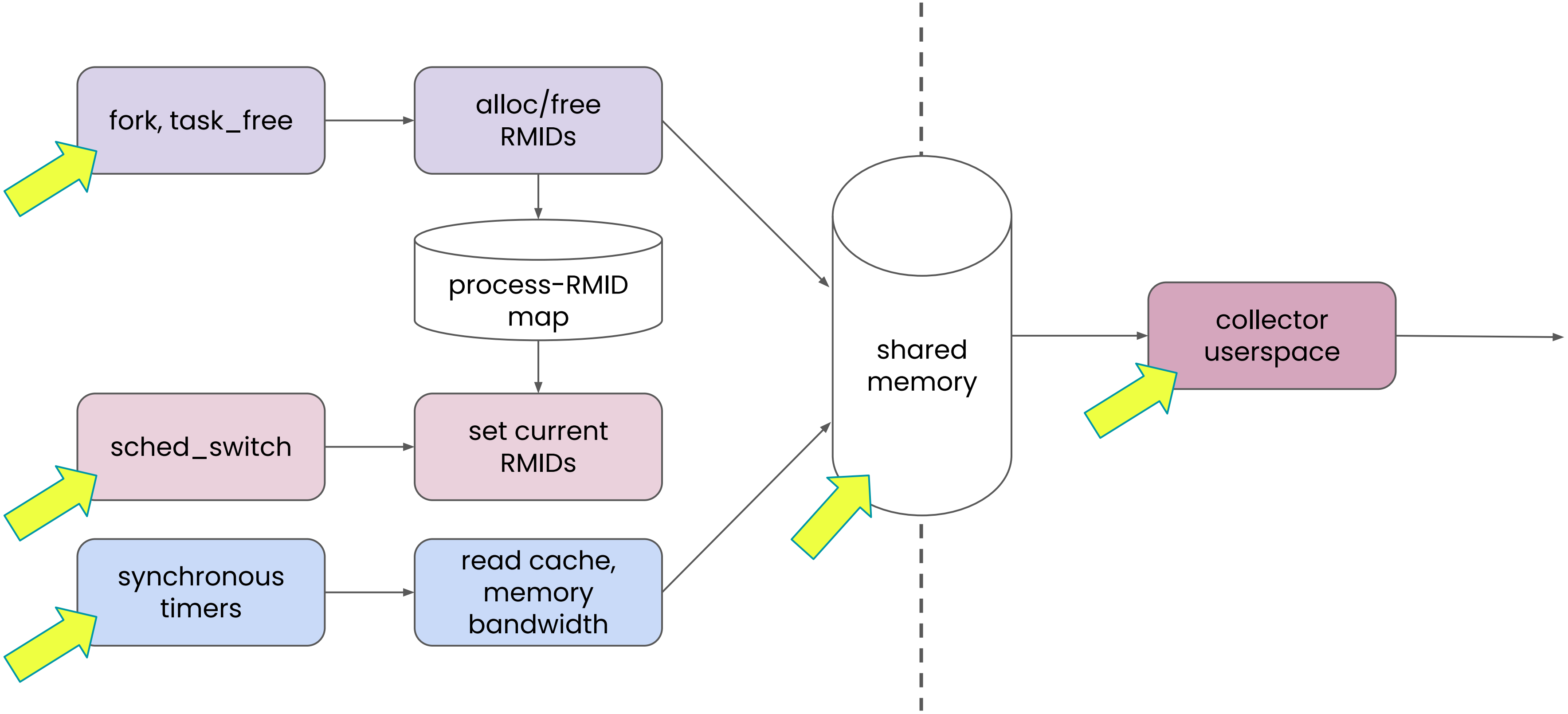
Sync Timer Benchmark Results

Points show mean delay, vertical lines show min–max range

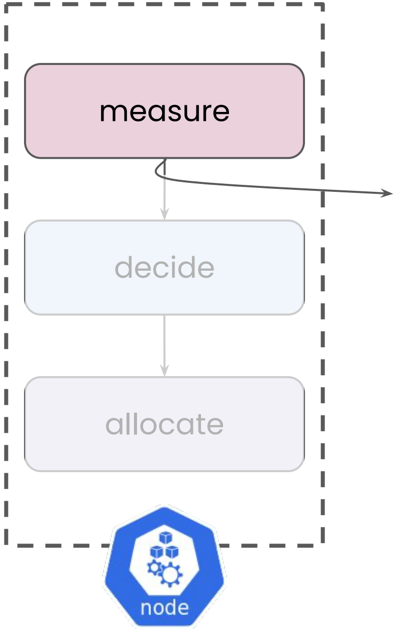


*** idle machines**

Collector architecture



Data helps develop detection algorithms



Collect per-RMID, per-millisecond:

- Cache utilization
- Memory bandwidth
- Cycles per Instruction (CPI)
- Cache misses

Goal: develop detection algorithms

How:

- Calculate detector values on data
- Validate detection with CPI

Synthetic workloads ok; **looking for production data**

Open Questions

- How to detect contention
  (bandwidth saturation)
- Is 1 millisecond frequent enough?
  (100 microseconds)
- Missing critical measurements?
  (CPU frequency)
- What if resctrl is unavailable?
  (use perf counters)

Security and Privacy

- Schema contains only profiling data:
  - Process names, PIDs
  - Container names (soon)
  - Cycles, instructions, LLC misses
  - Memory bandwidth, cache utilization
- No PII captured
- Runs locally, produces parquet files
  - No external communication

Overhead

- Designing for:
  - < 0.1% in-line overhead
  - < 1% userspace

- Even if more, might still make sense!

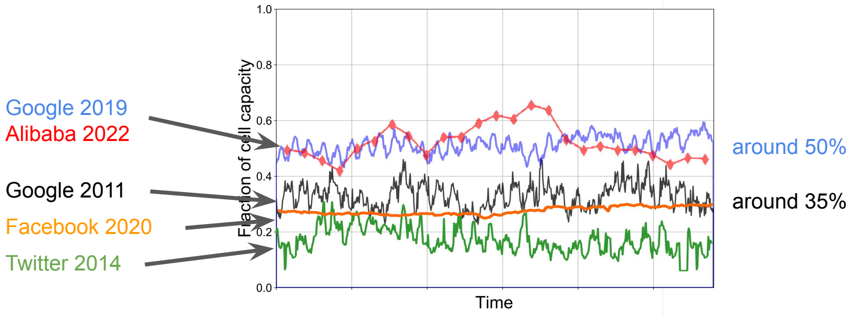
average	p95
40 ms	250 ms

average	p95
42 ms	75 ms

Community Next Steps

unvariance/collector

Ian Off
Tarun Verghis
Darshan Dedhia
Nimrod Liberman



Call for:

- Contributors
    - Collector
    - Kubernetes benchmarks
- Data contributions
    - Deploy in test/staging
    - Advance detector development

Contact:

CNCF Slack: @Jonathan Perry
yonch@yonch.com

Set up time: yonch.com/collector