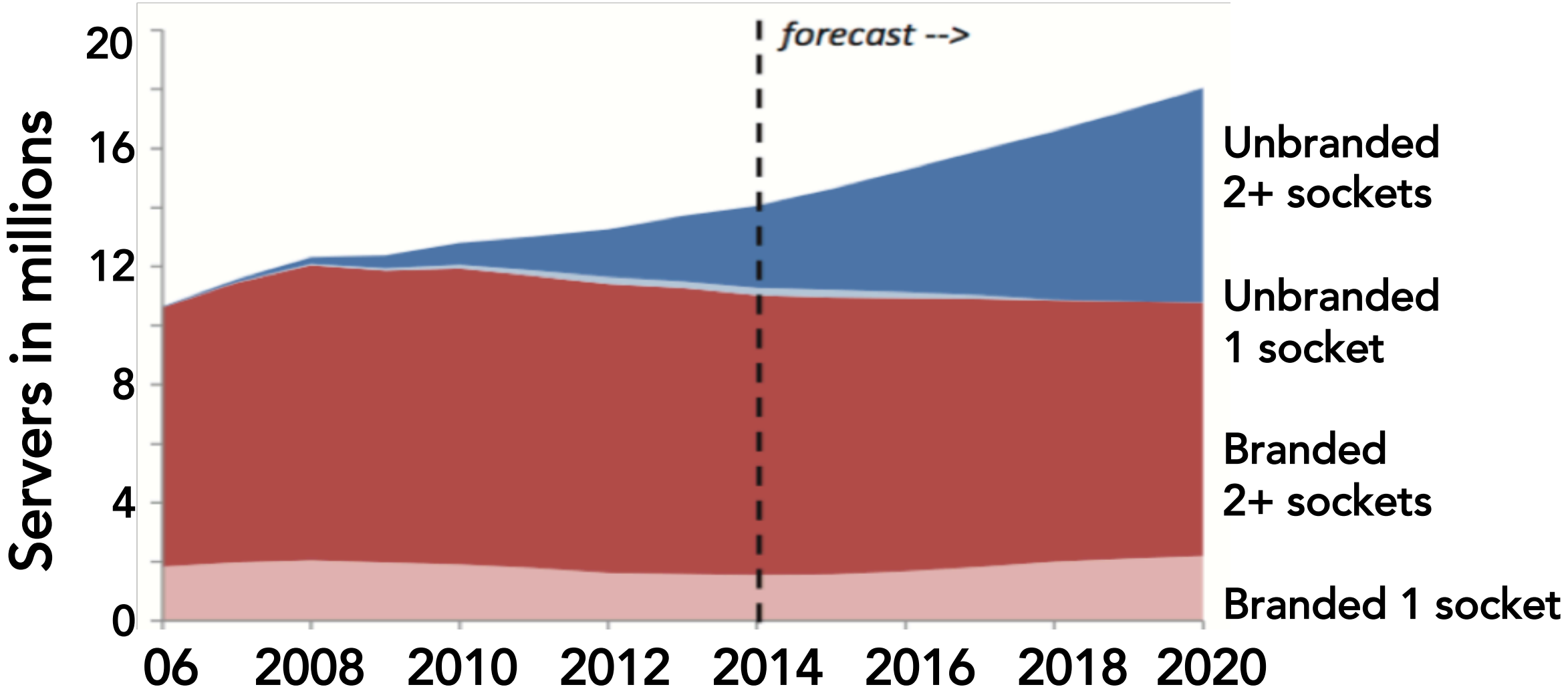
Tail Latency: Beyond Queuing Theory

Kathryn S McKinley
Xi Yang, Stephen M Blackburn,
Sameh Elnikety, Yuxiong He, Ricardo Bianchini

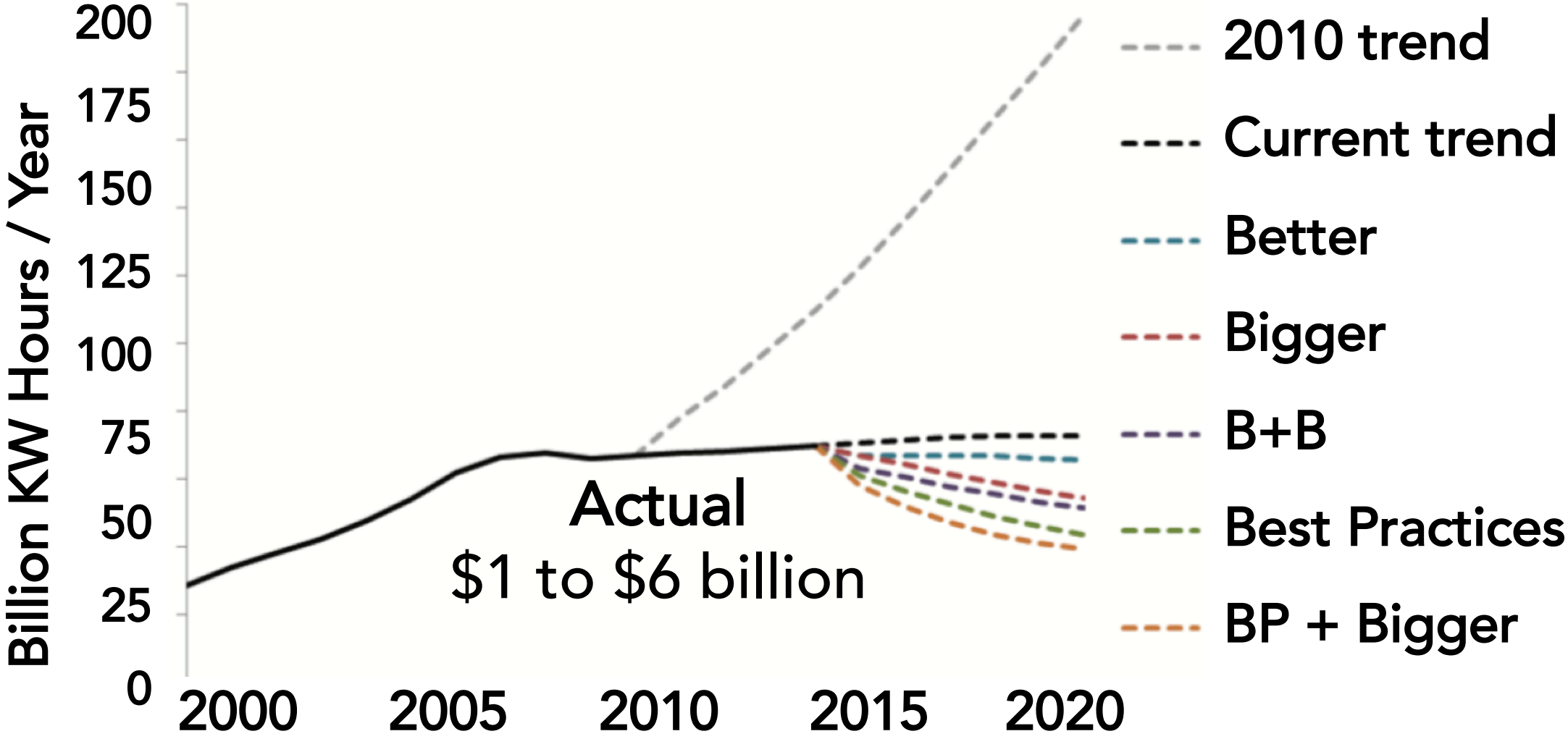
Microsoft's Quincy datacenter

Servers in US datacenters



*Shehabi et al., United States Data Center Energy Usage Report, Lawrence Berkeley, 2016.

Electricity in US datacenters



*Shehabi et al., United States Data Center Energy Usage Report, Lawrence Berkeley, 2016.

Datacenter economics quick facts*

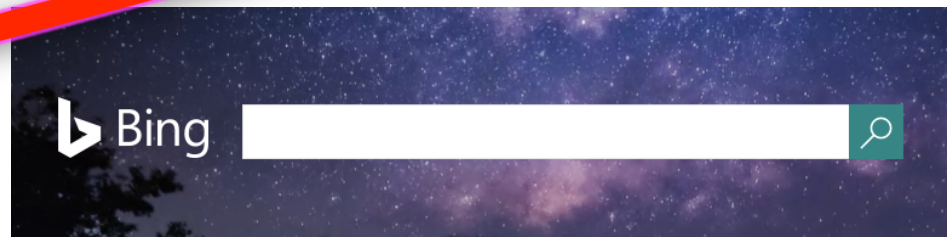
~ $500,000	Cost of one datacenter
~3,000,000	US datacenters in 2016
~ $1.5 trillion	US Capital investment to date
~ $3,000,000,000	KW dollars / year
~ $30,000,000	Savings from 1% less work
Lots more	by not building a datacenter

*Shehabi et al., United States Data Center Energy Usage Report, Lawrence Berkeley, 2016.

Improve efficiency!

Tail Latency Matters

Google Australia



400 millisecond delay decreased searches/user by 0.59%. [Jack Brutlag, Google]

Two second slowdown reduced revenue/user by 4.3%. [Eric Schurman, Bing]

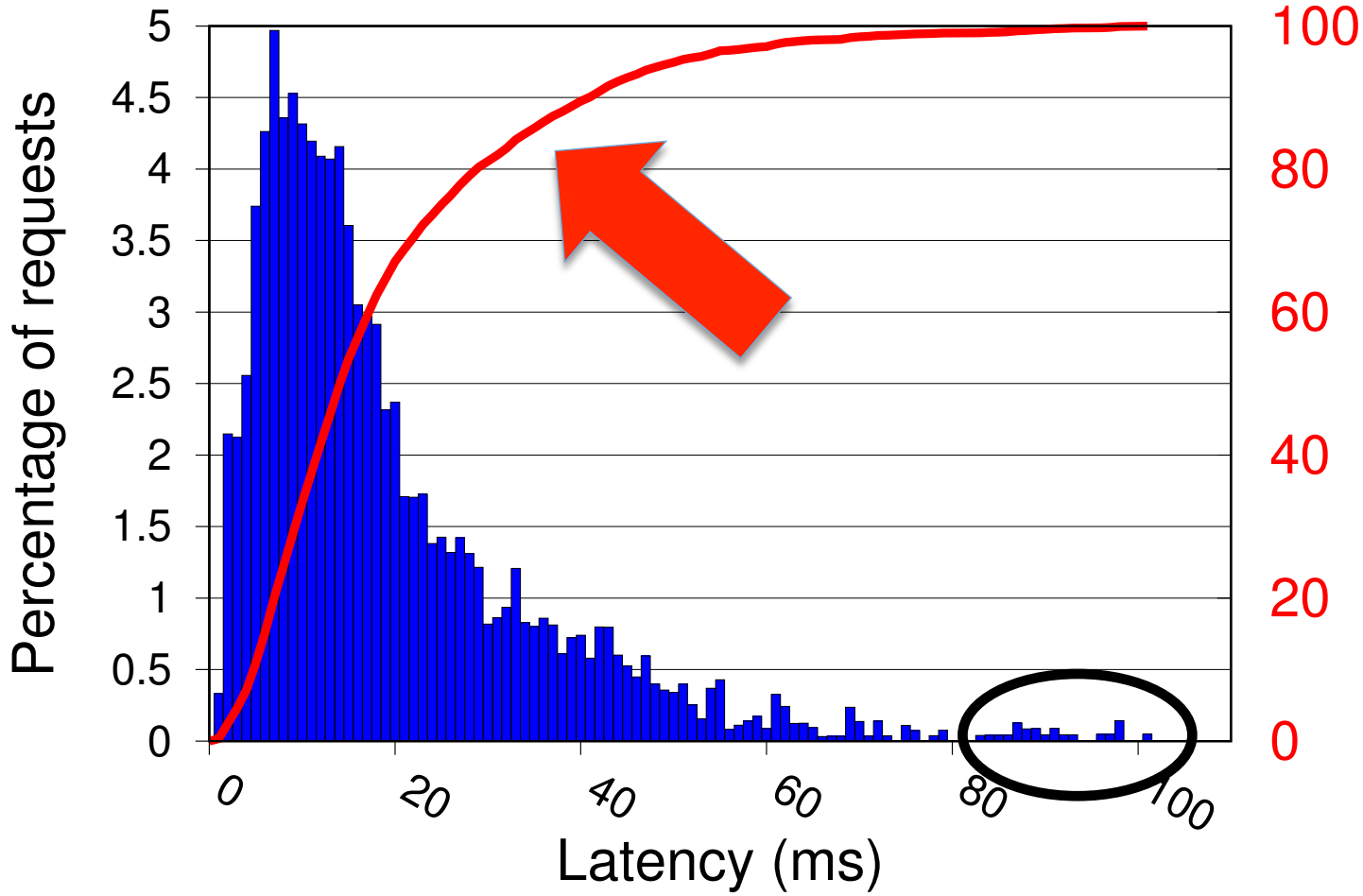
Server architecture

client

aggregator

workers

Characteristics of interactive services

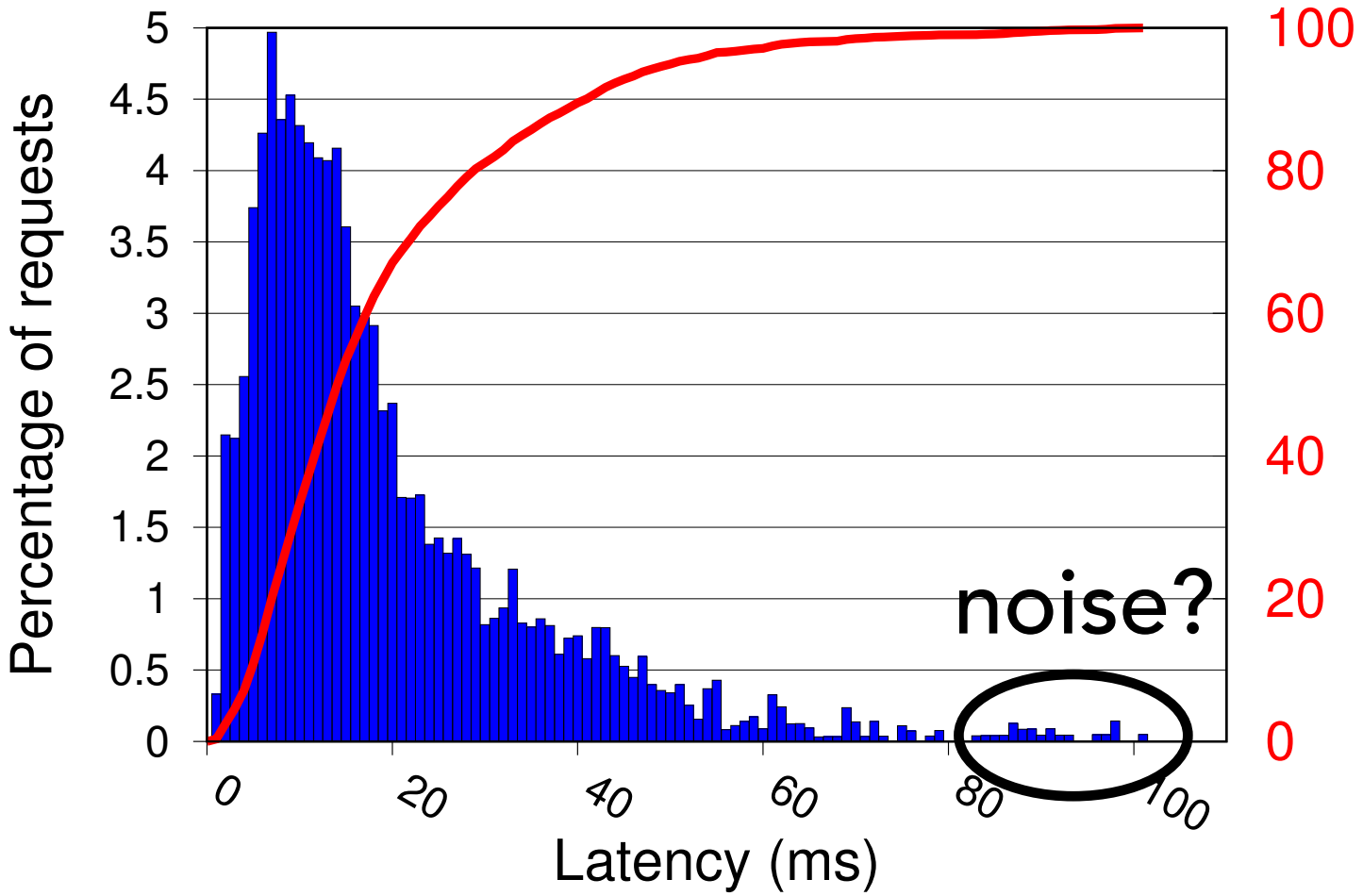


LC

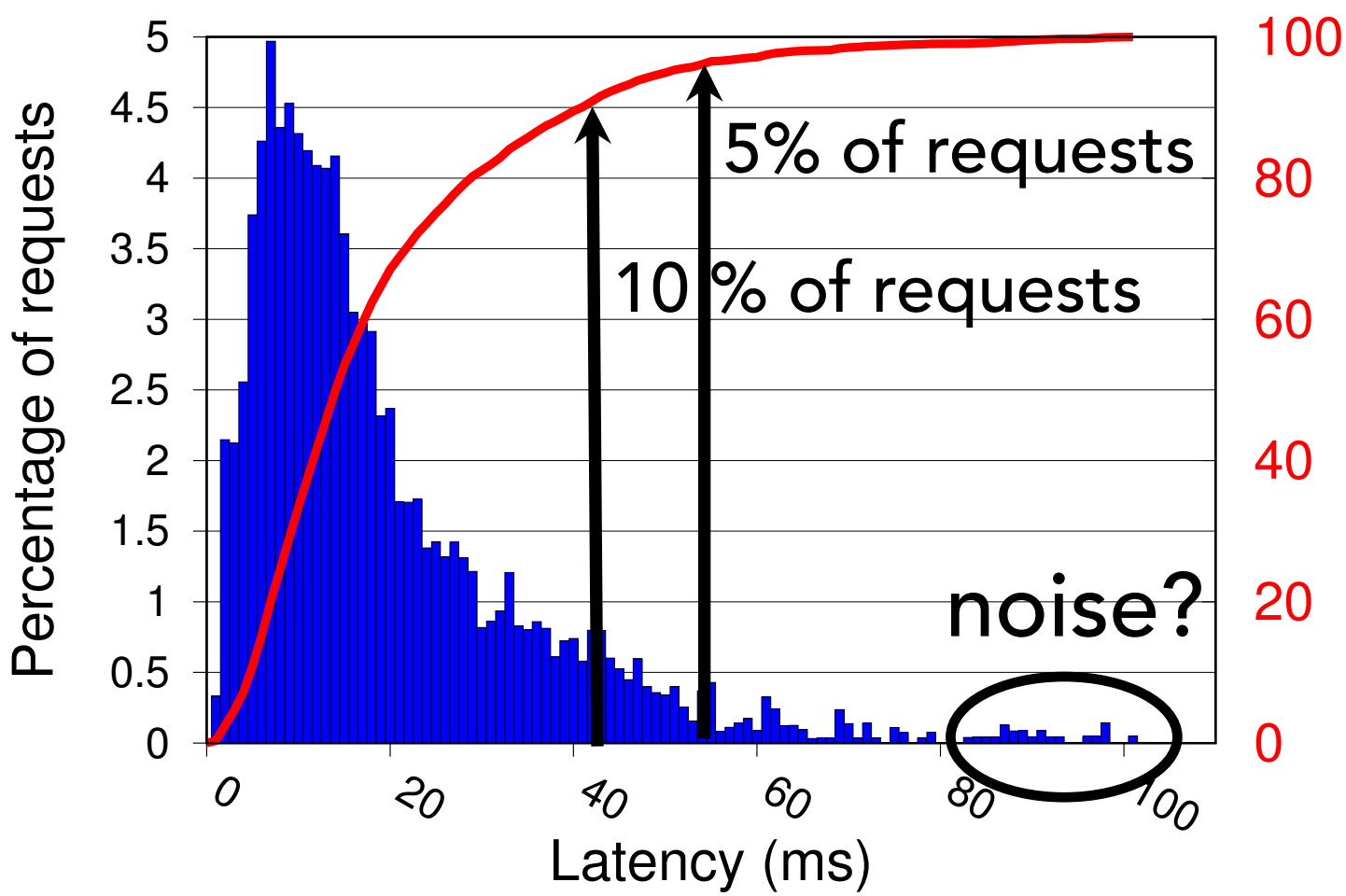
Lucene

- Bursty, diurnal
- **CDF** changes slowly
- Slowest server dictates tail
- Orders of magnitude diff average & 99+ %tile

Client side observations



Client side observations

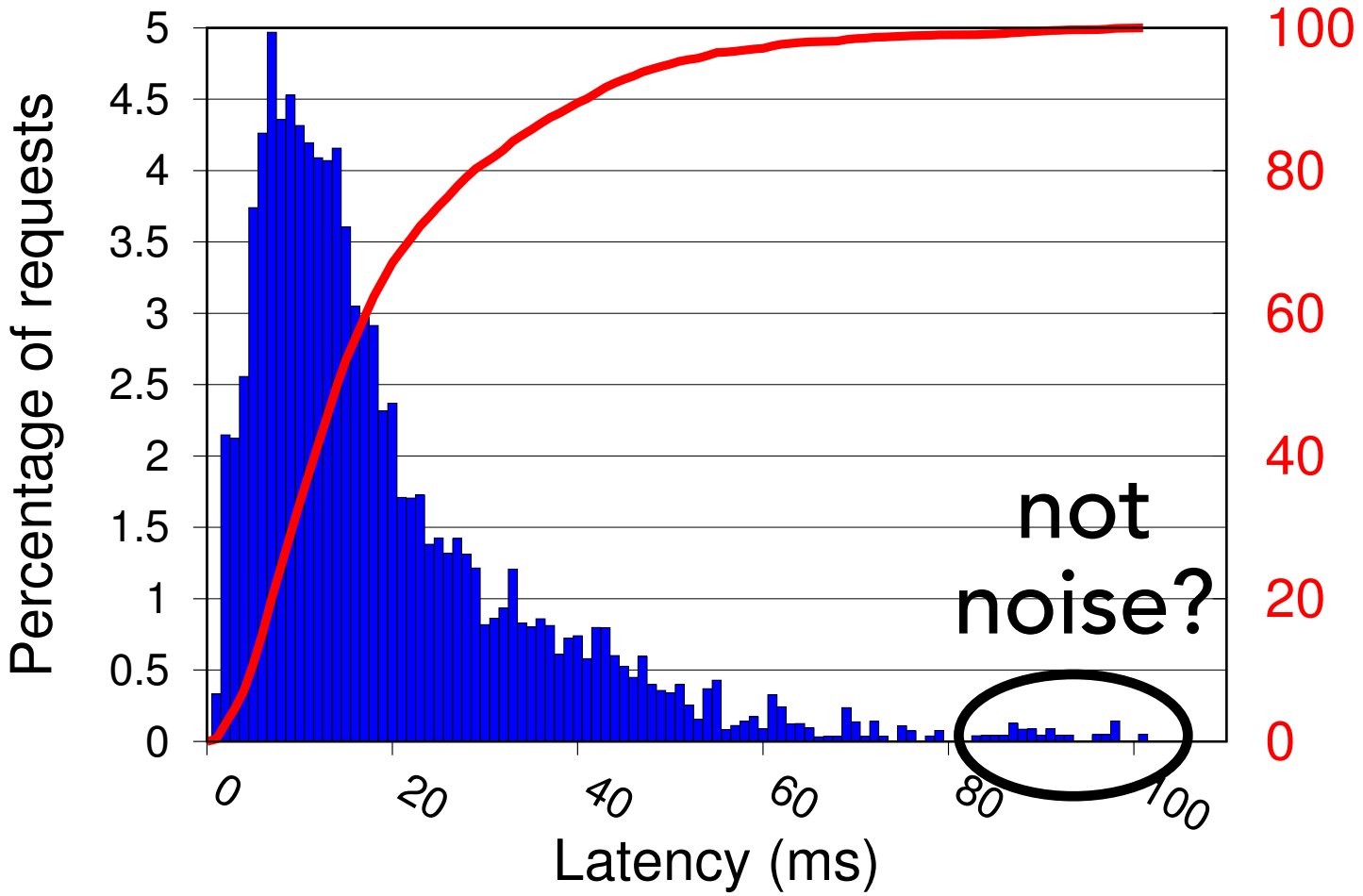


Solution to noise

Replication

- All requests?
- CFD shows cost & potential

Client side observations



Roadmap

What's in the tail?

Continuous profiling to diagnose the tail

Real problems

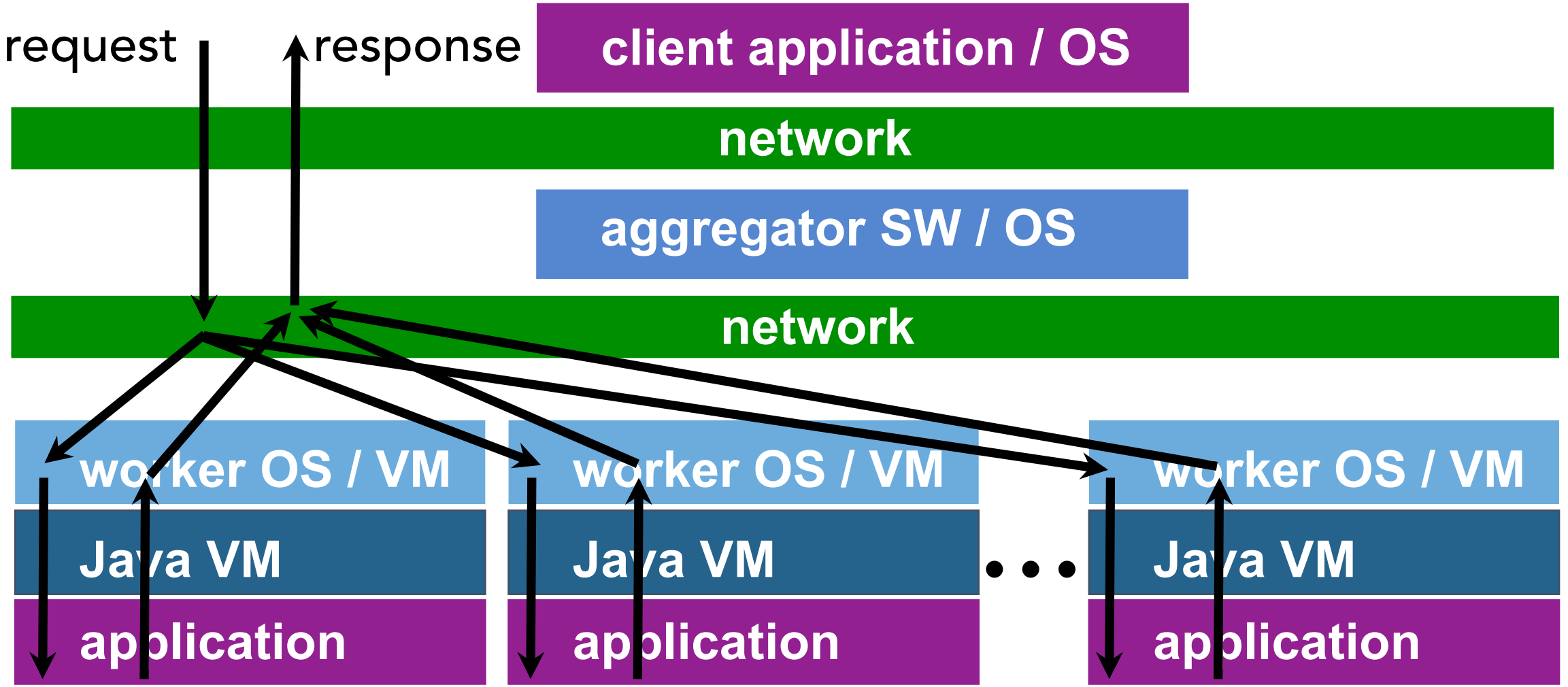
- Noise: replication
- Work: parallelism
- Other opportunities

Still poor utilization due to bursty diurnal workload

- Colocation for utilization without impacting tail latency

Opportunities in hardware/software codesign

Simplified life of a request



Prior state of the art

Dick Site's talk: https://www.youtube.com/watch?v=QBu2Ae8-8LM

Dick Sites & team

Hand instrument system

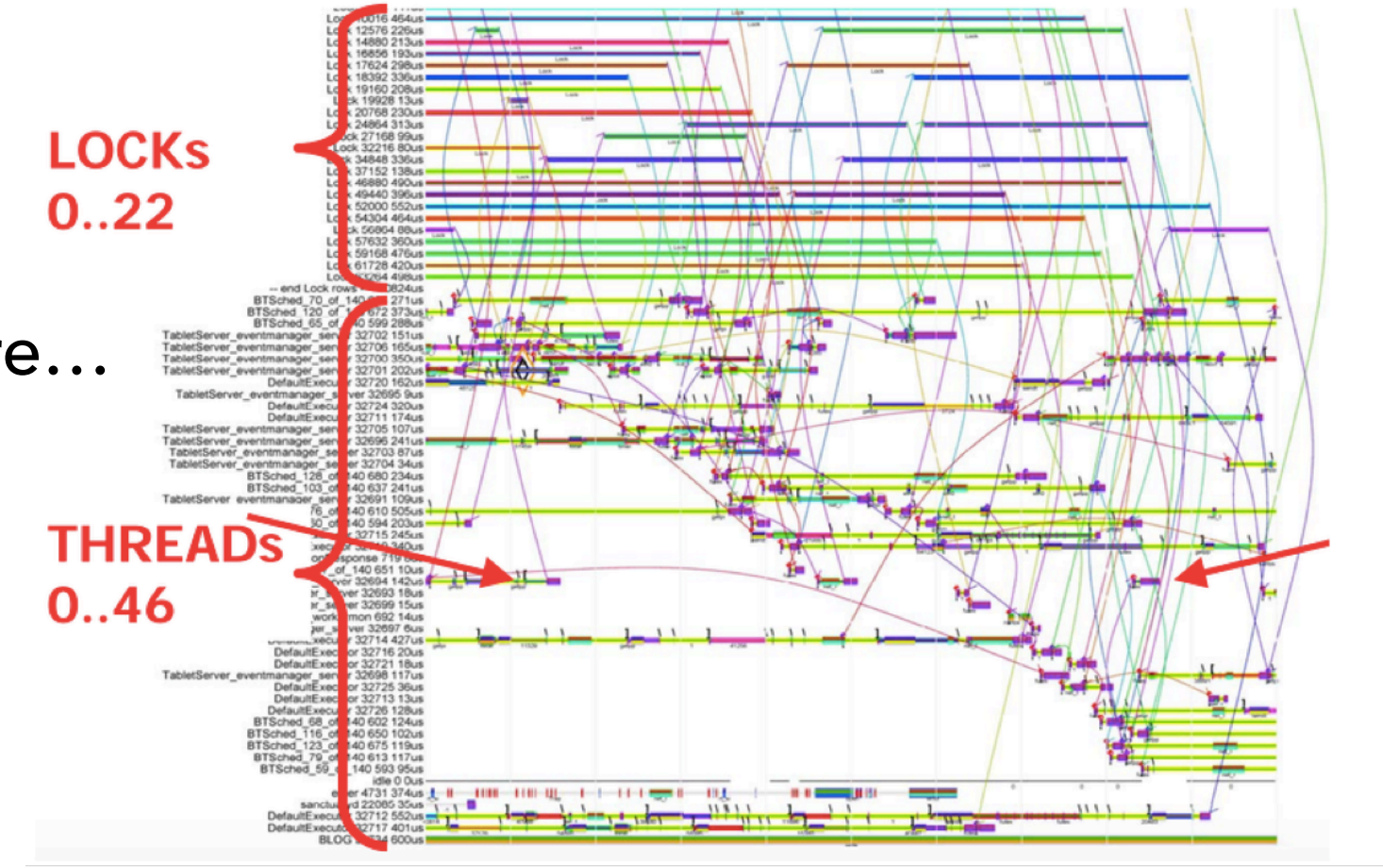
1% on-line budget

sample – but tails are rare…

Off-line schematics

Have insight

Improve the system



Dick Sites & team

Hand instrument system

1% on-line budget

sample – but tails are rare…

Off-line schematics

Have insight

Improve the system

Dick Sites & team

~~Hand~~ instrument system

1% on-line budget

~~sample~~ – but tails are rare…

Off-line schematics

Have insight

Improve the system

Automated instrumentation

1% on-line budget

continuous on-line profiling

Off-line schematics

Have insight

Improve the system

+ On-line optimization

Automated cycle-level on-line profiling

[ISCA'15 (Top Picks HM), ATC'16]

Insight Hardware & software generate signals

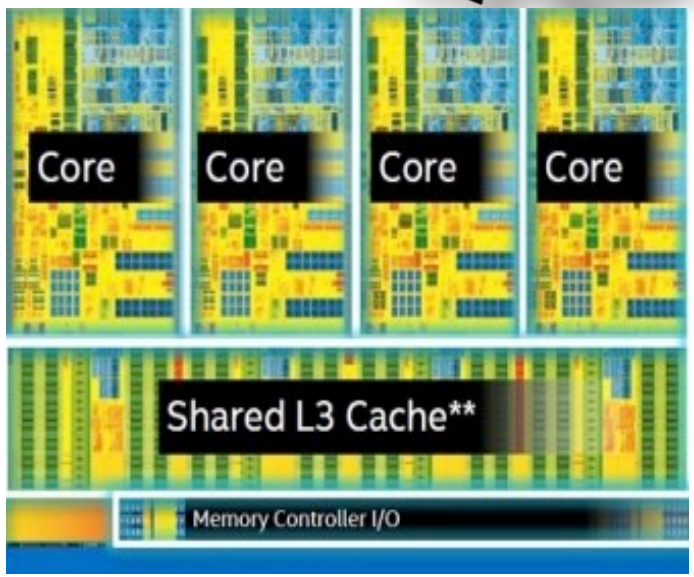
	hardware signals performance counters	software signals memory locations
counters	✓	✓
tags	✓	✓

SHIM Design

ISCA'15 (Top Picks HM), ATC'16

Observe global state from other core

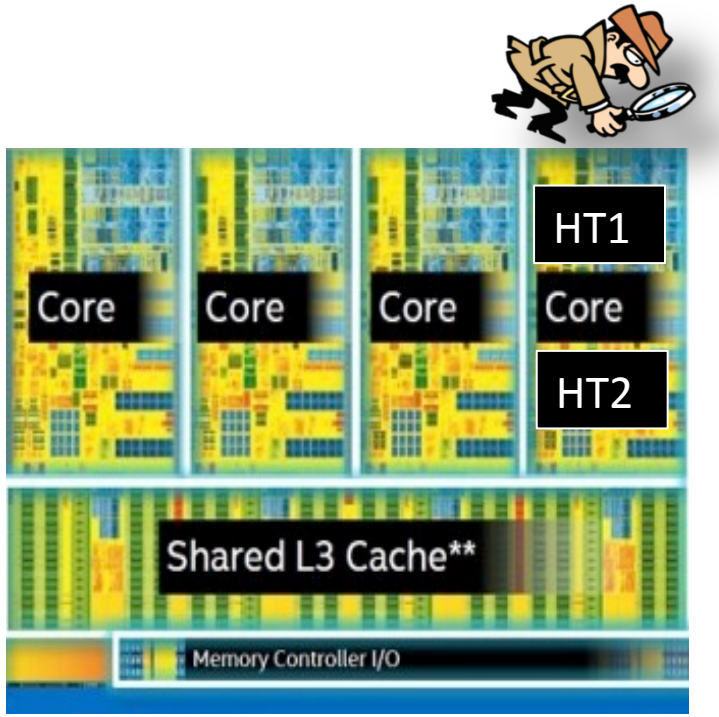
```
while (true):
    for counter in LLC misses, cycles:
        buf[i++] = readCounter(counter)
```

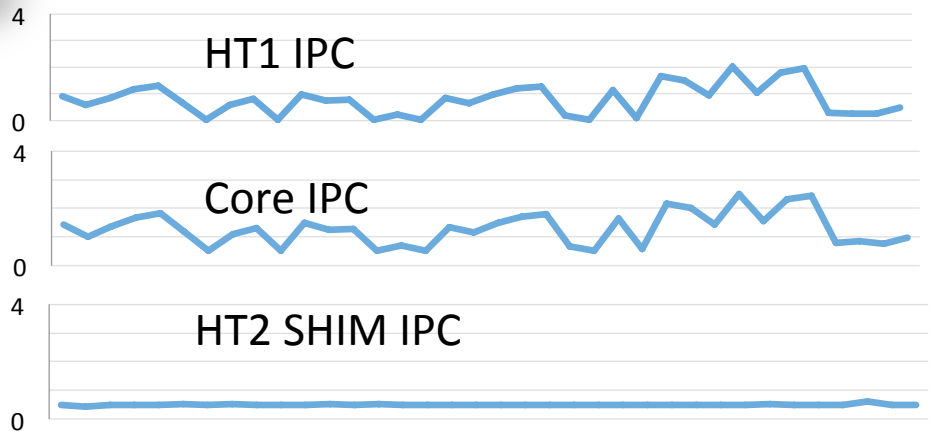


LLC misses per cycle

Observe local state with SMT hardware

```
while (true):
    for counter in HT2 SHIM, Core, Cycles:
        buf[i++] = readCounter(counter);
```

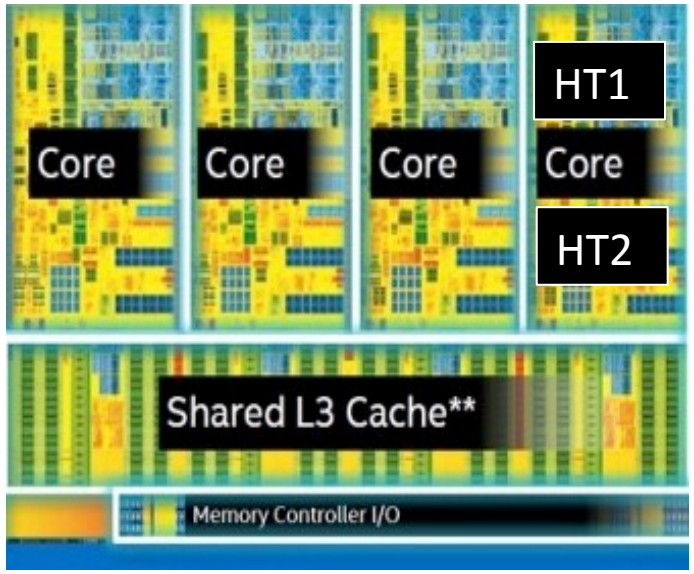


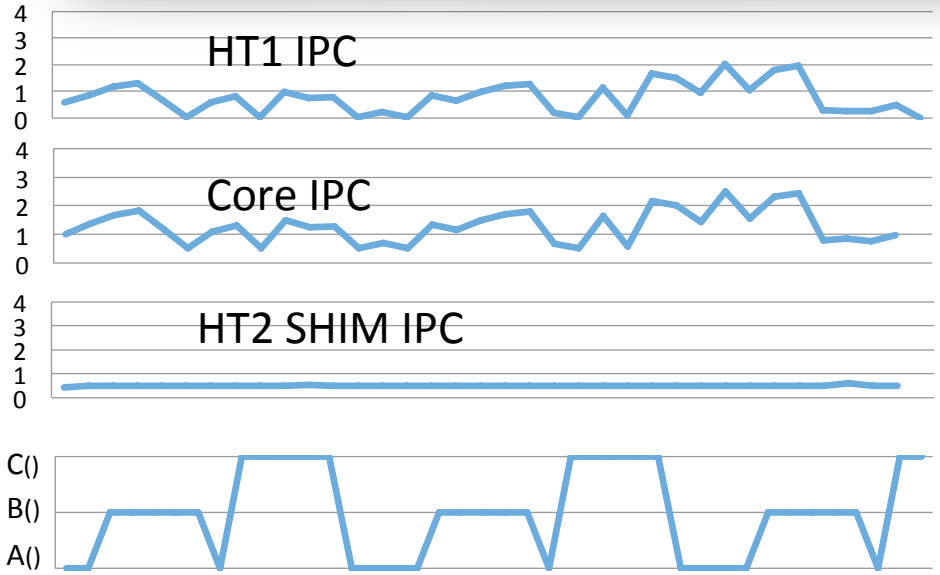


HT1 IPC = Core IPC – HT2 SHIM IPC

Correlate hardware & software events

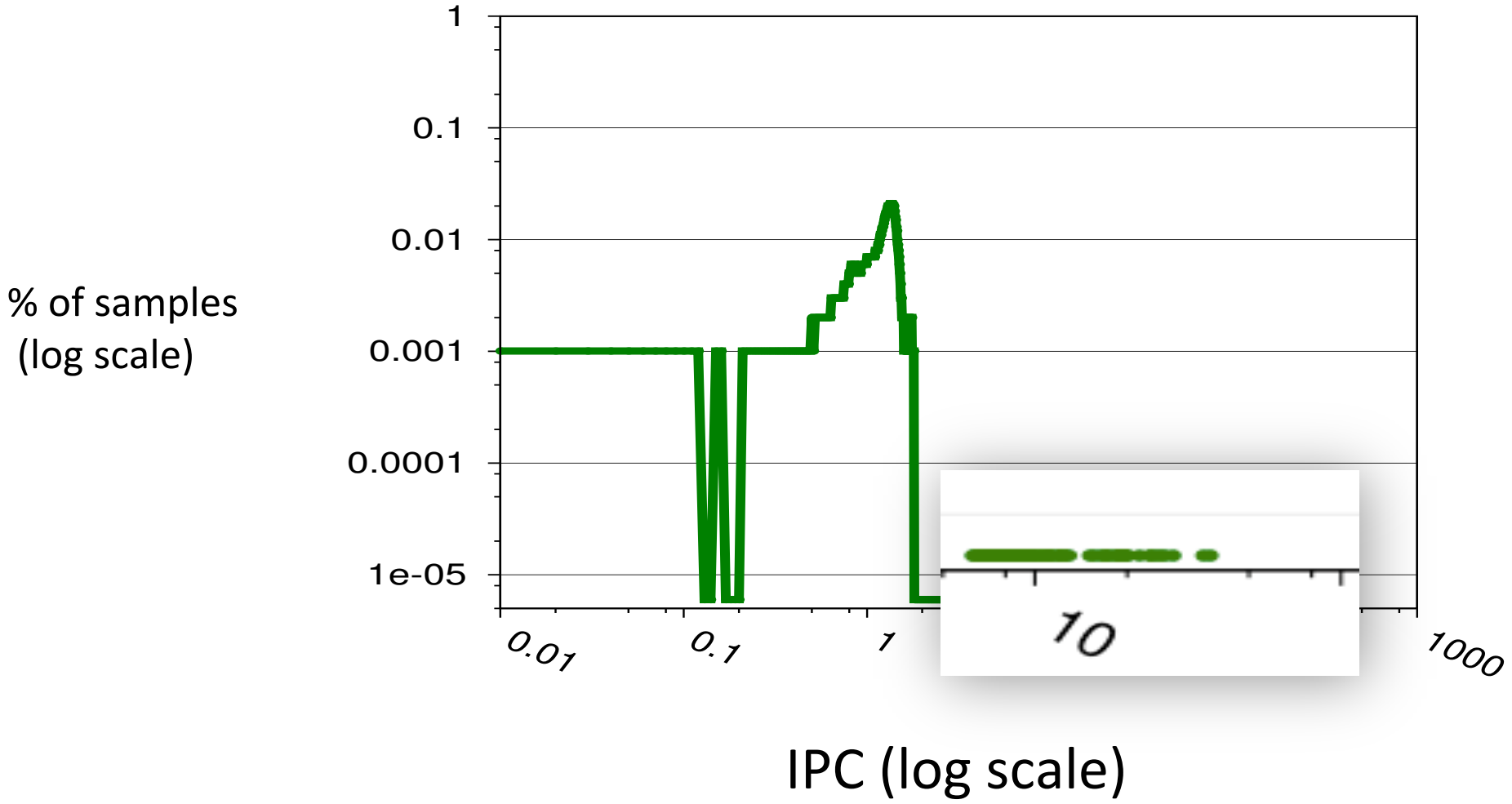
```
while (true):
    for counter in HT2 SHIM, Core, cycles:
        buf[i++] = readCounter(counter);
    tid = thread on HT1
    buf[i++] = tid.method;
```





Fidelity

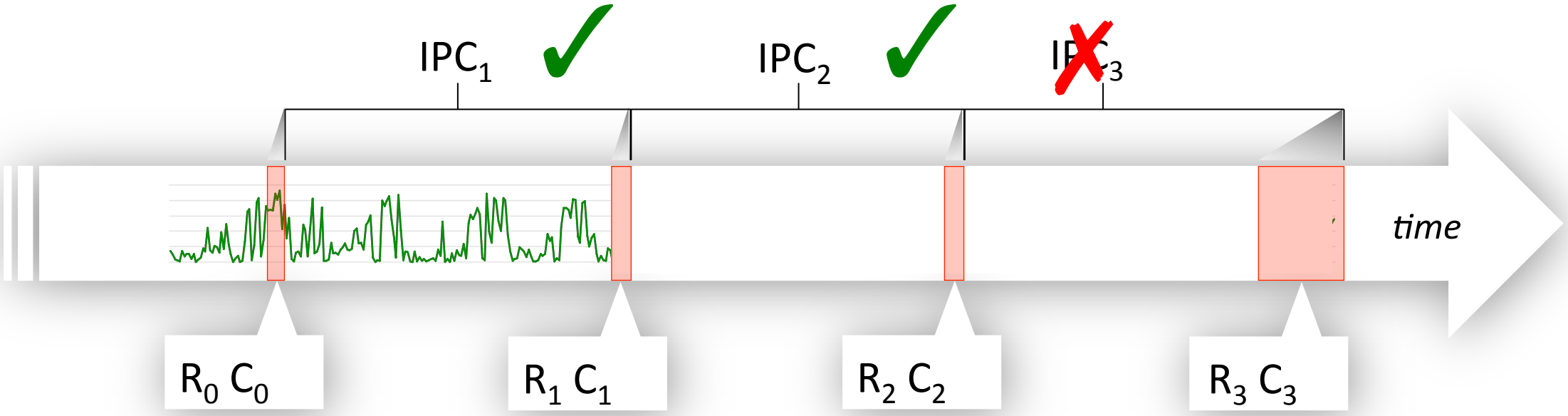
Raw samples


% of samples
(log scale)
1
0.1
0.01
0.001
0.0001
1e-05
0.01
0.1
1
10
1000
IPC (log scale)

Problem: samples are not atomic

Counters C: cycles R: retired instructions

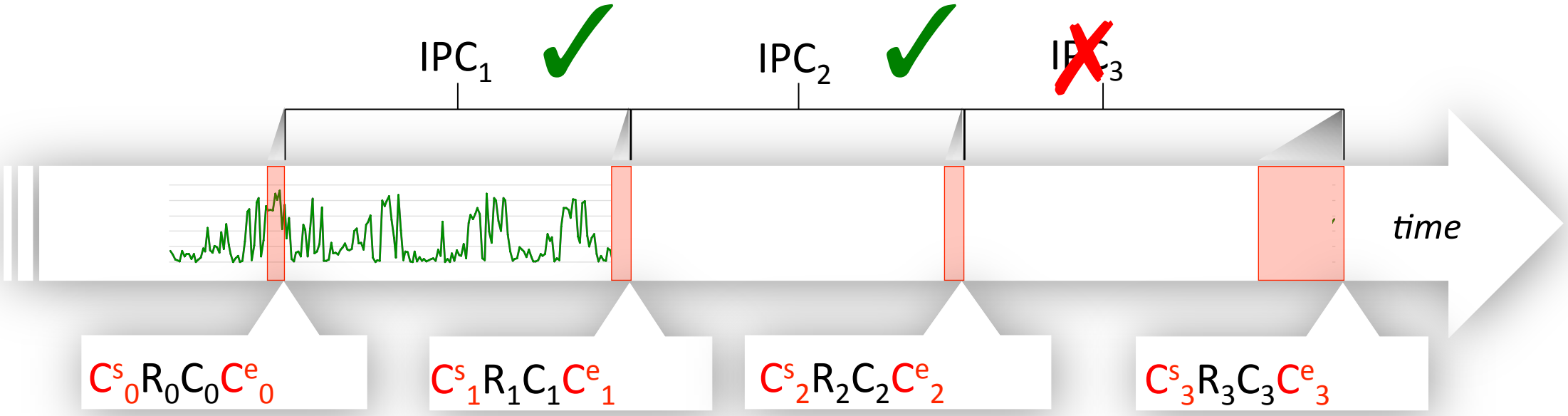
$IPC = (R_t - R_{t-1}) / (C_t - C_{t-1})$



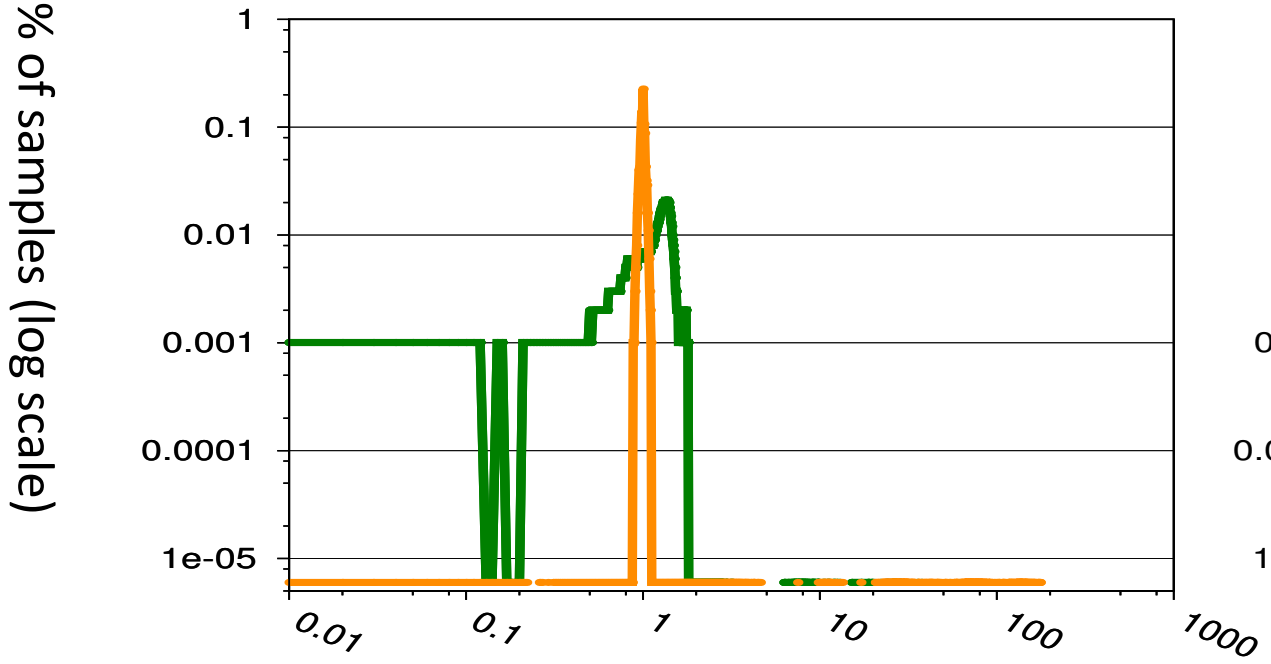
Solution: use clock as ground truth

$CPC = (C^e_t - C^e_{t-1}) / (C^s_t - C^s_{t-1})$ *this should be 1!*

$CPC_1 = 1.0 +/- 1\%$ $CPC_2 = 1.0 +/- 1\%$ $CPC_3 != 1.0 +/- 1\%$

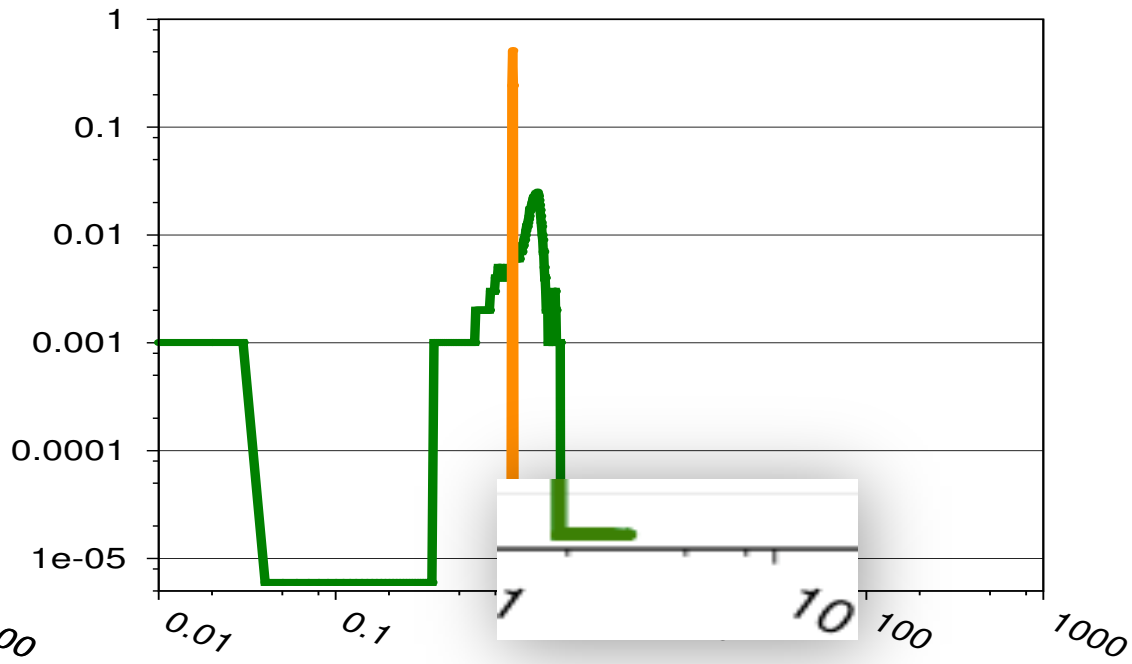


Filtering Lusearch IPC samples



---- raw IPC

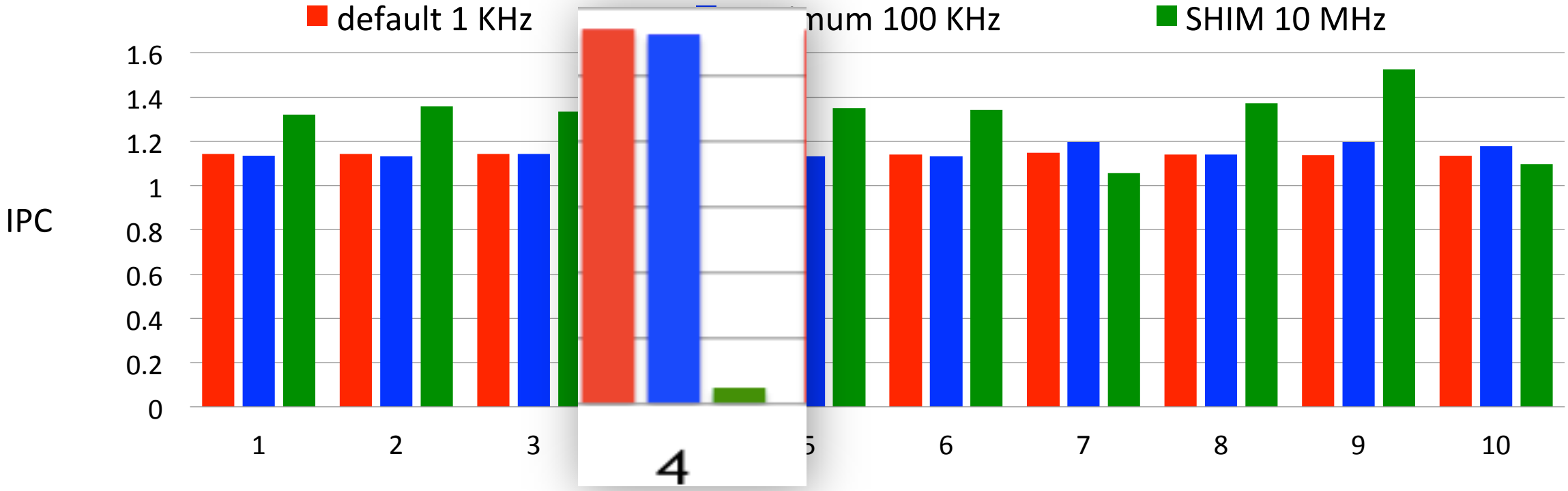
---- raw CPC



---- filtered IPC

---- filtered CPC in [0.99,1.01]

IPC of individual methods in Lucene

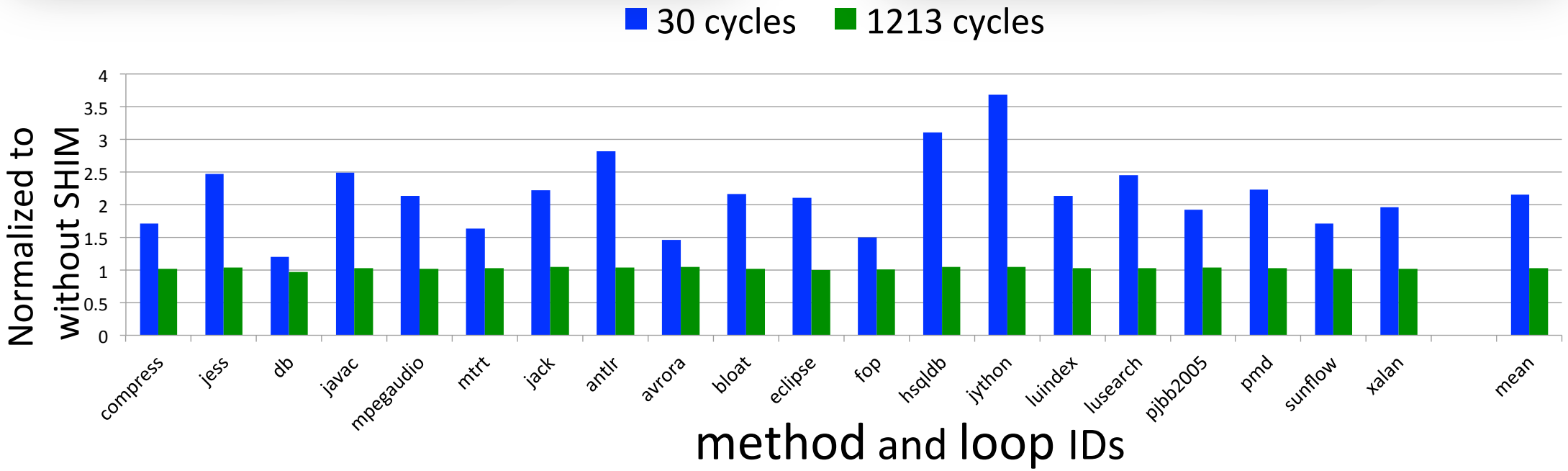


top 10 methods (74% total execution time)

Overheads from other core

113MHz: 3+ orders of magnitude over interrupt ‘maximum’

3MHz: 1+ order of magnitude over interrupt ‘maximum’



Overheads from write invalidations

Understanding Tail Latency

SHIM signals

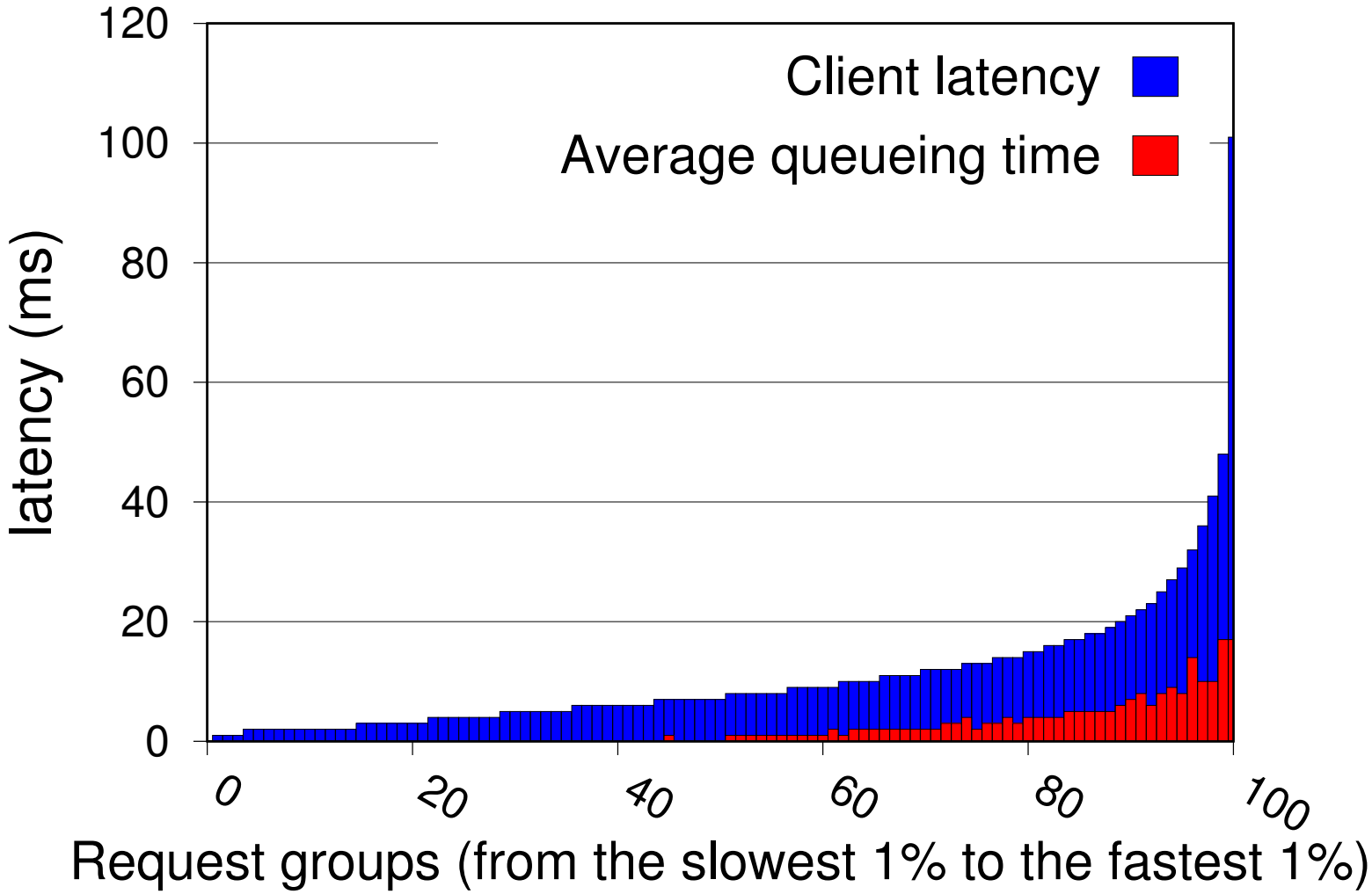
Requests

- thread ids
- request id (software configured)
- time stamps, PC

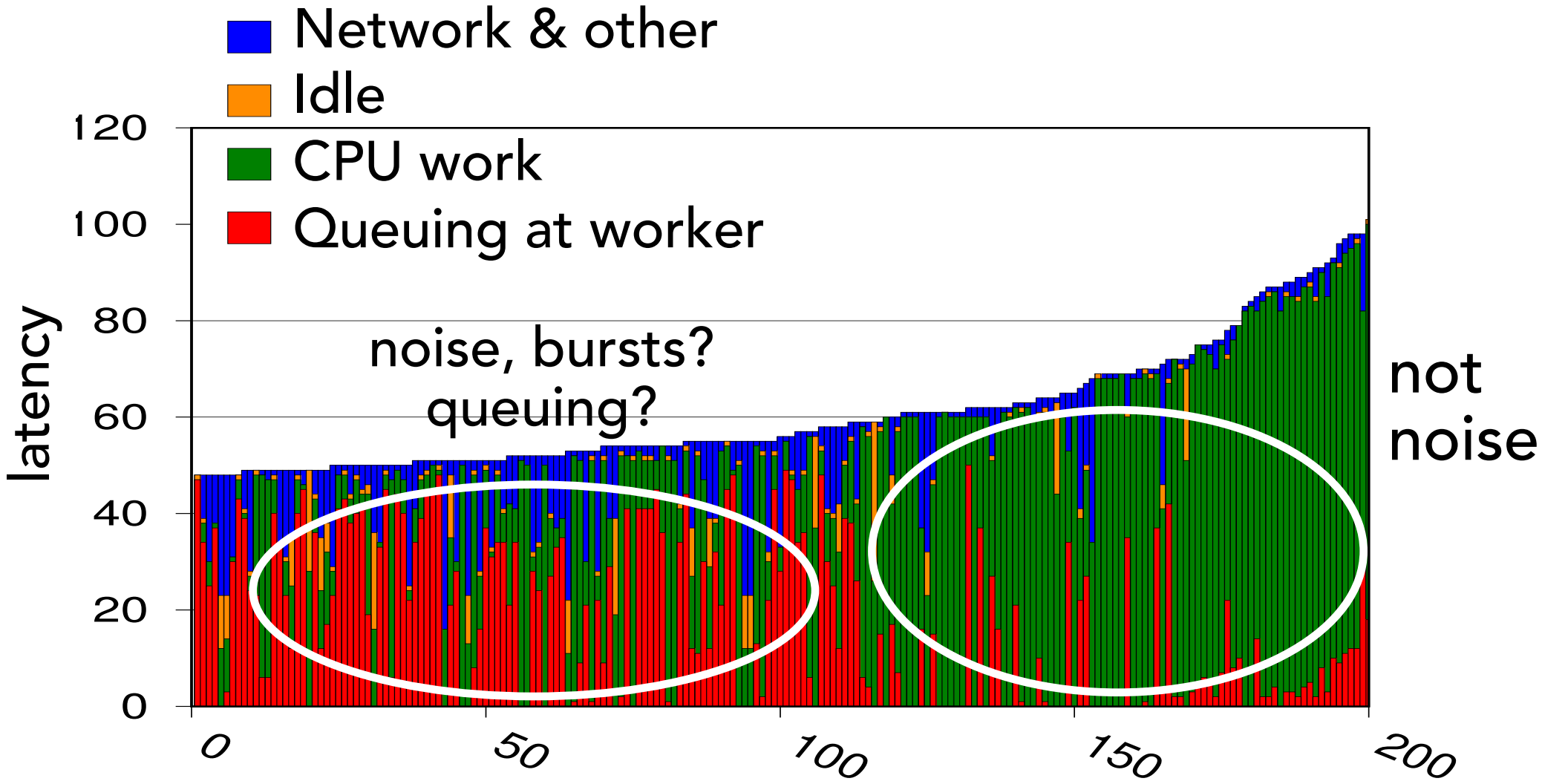
System threads

- thread ids
- time stamp, PC

All requests



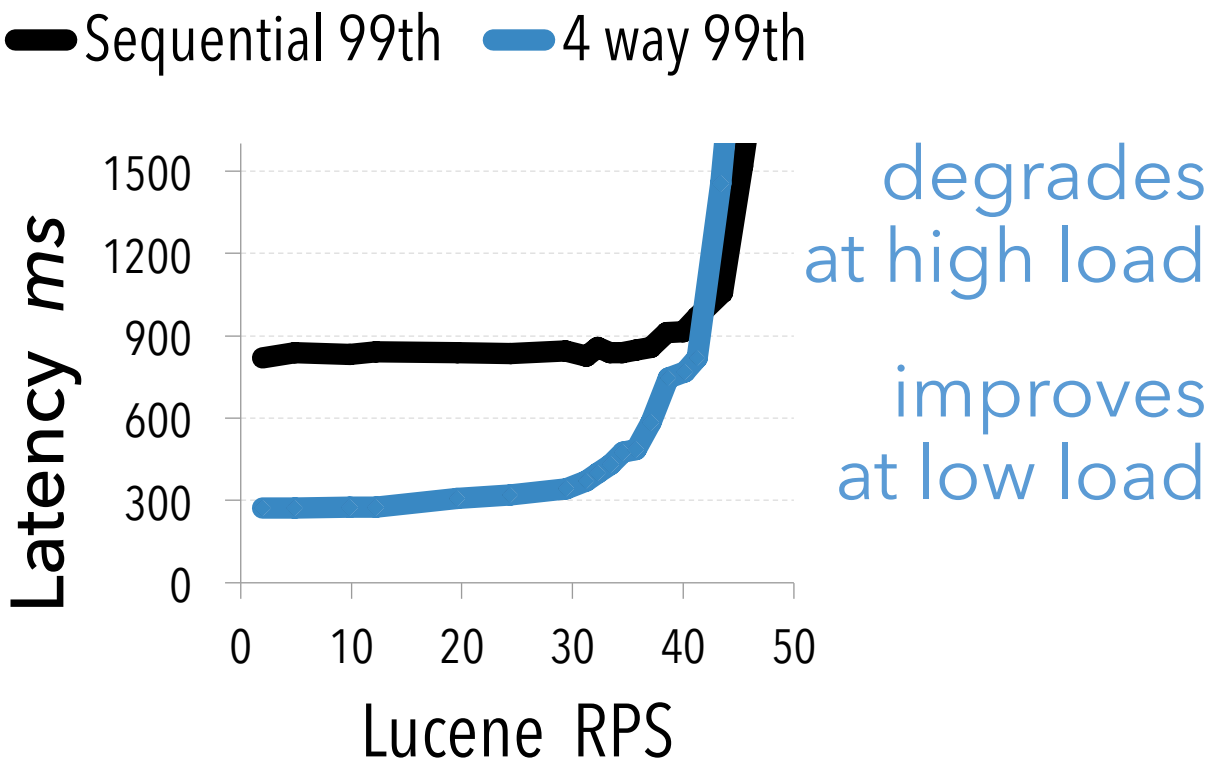
Longest 200 requests



Parallelism

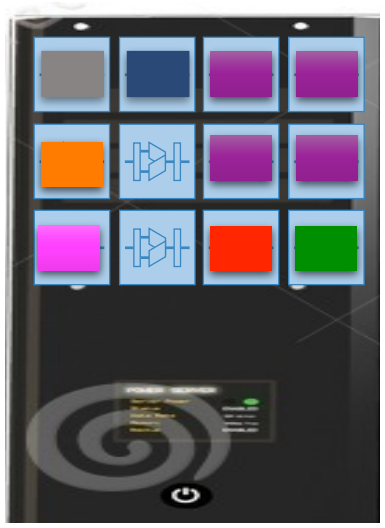
Parallelism historically for **throughput**

Idea Parallelism for **tail latency**



Tail of Many Dynamic Parallelism [ASPLOS'15]

Parallelism historically for **throughput**

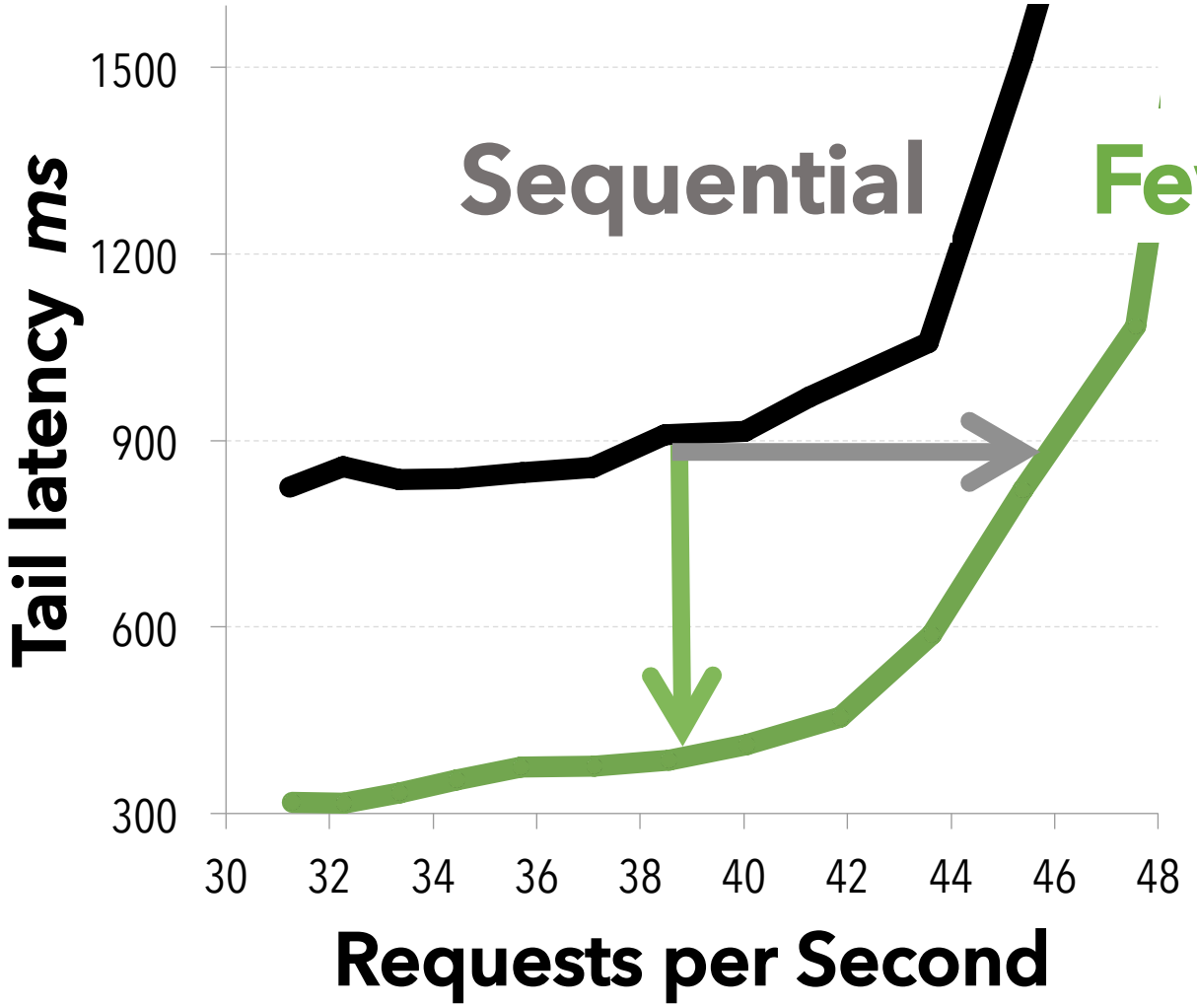


Idea Parallelism for **tail latency**

Insight Long requests reveal themselves

Approach Incrementally add parallelism to long requests – the tail – based on request progress & load

Evaluation 2x8 64 bit 2.3 GHz Xeon, 64 GB



buy fewer servers

reduce tail latency

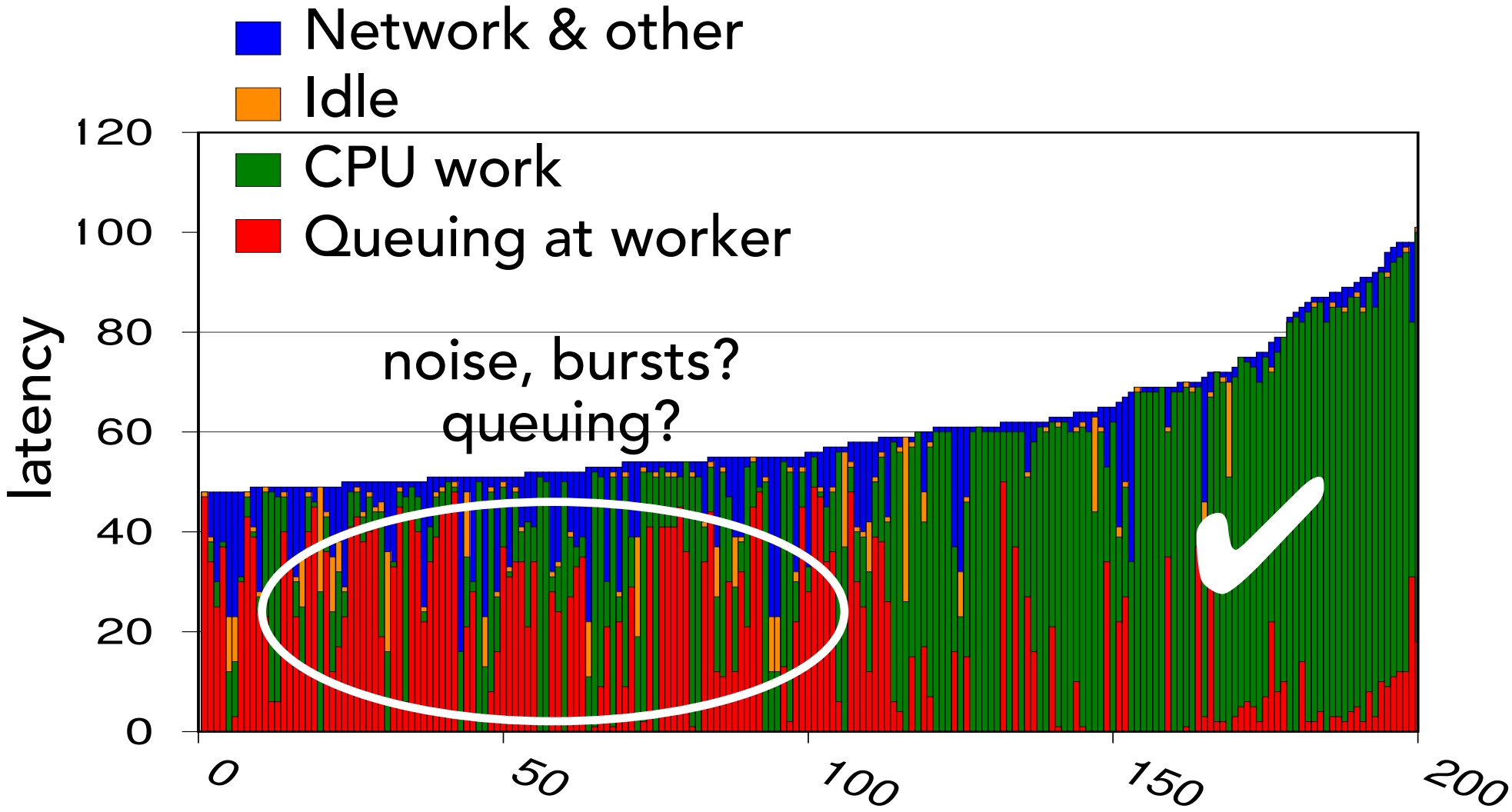
Queuing theory

Optimizing average latency maximizes throughput

But not the tail!

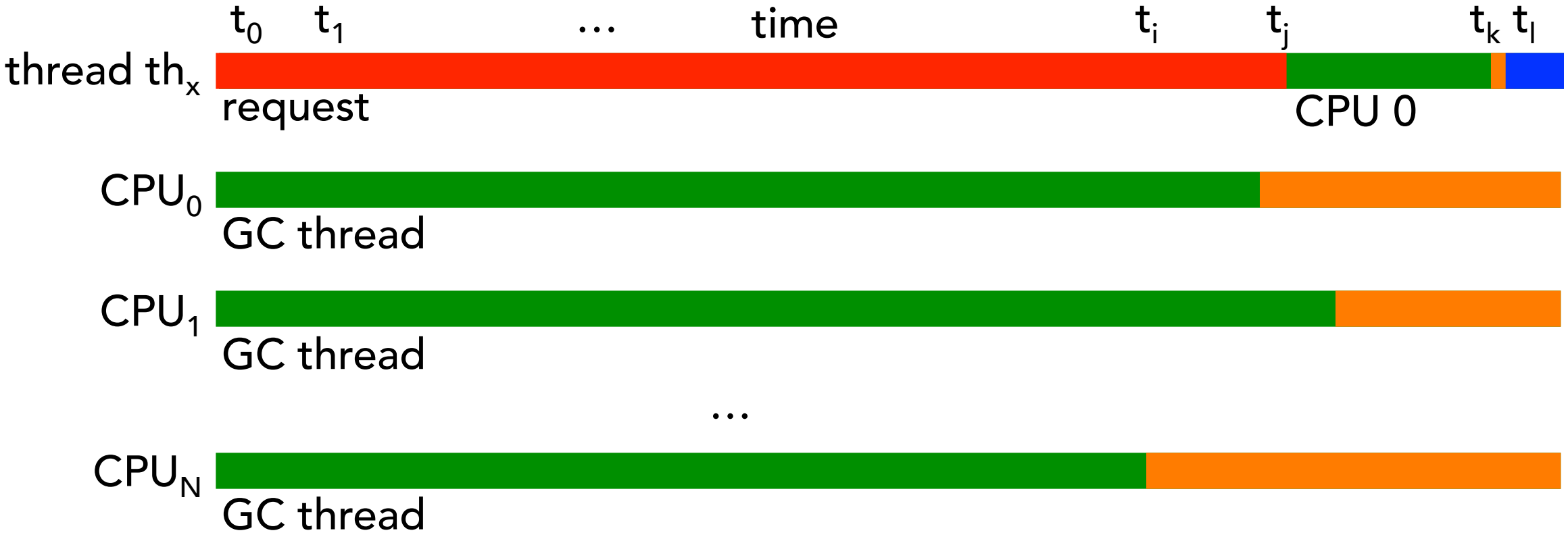
Shortening the tail reduces queuing latency

Longest 200 requests



Correlate bad requests with system state

Use time stamps to post-process traces



Recap & what's next

SHIM continuous profiling to diagnose the tail

- Noise: replication
- Work: parallelism
- Scalability bottlenecks

Continuous monitoring suggests dynamic optimizations

but… still poor utilization due to bursty diurnal workload

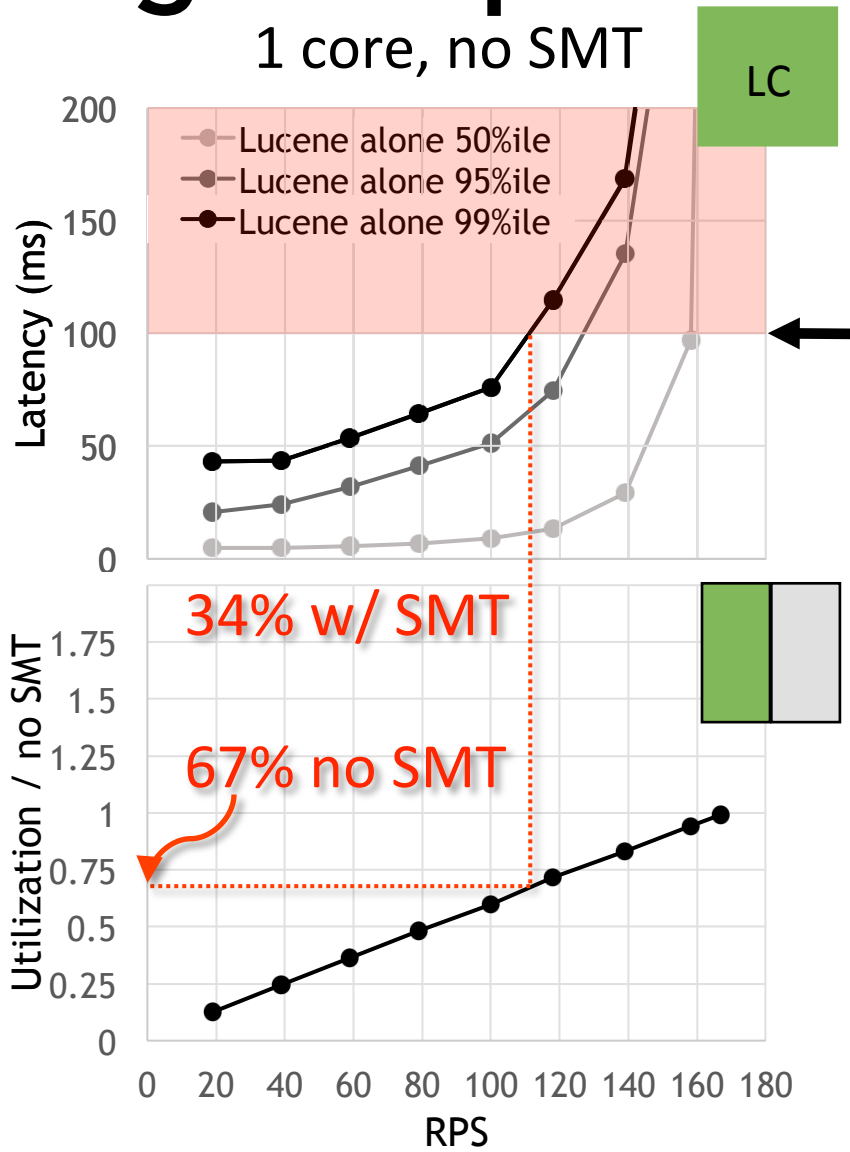
- Colocation

Looking forward

Queuing theory

Over provision for maximum burst, otherwise queuing delay degrades average and tail latency

High Responsiveness–Low Utilization



Service Level Objective
100ms SLO

LC

Lucene

Intel Xeon-D 1540 Broadwell

"Such WSCs tend to have relatively low average utilization, spending most of its time in the
10 - 50% CPU utilization range."

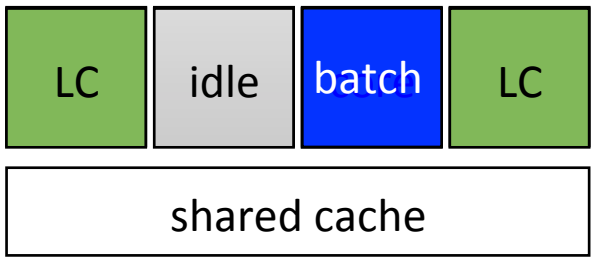
Luiz André Barroso, Urs Hölzle
"The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines"

Soak up Slack with Batch?

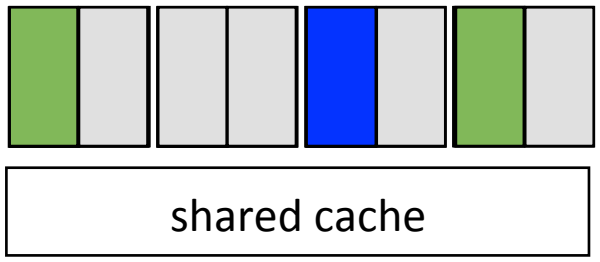
Goal No tail latency impact [TOCS'16, EuroSys'14]

requires idle cores in part because

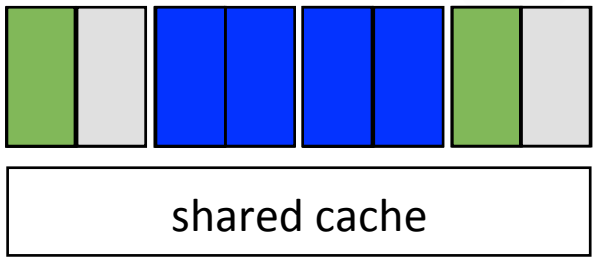
OS descheduling is slow



Co-running on different cores
SMT turned off



Co-running on different cores
SMT turned off



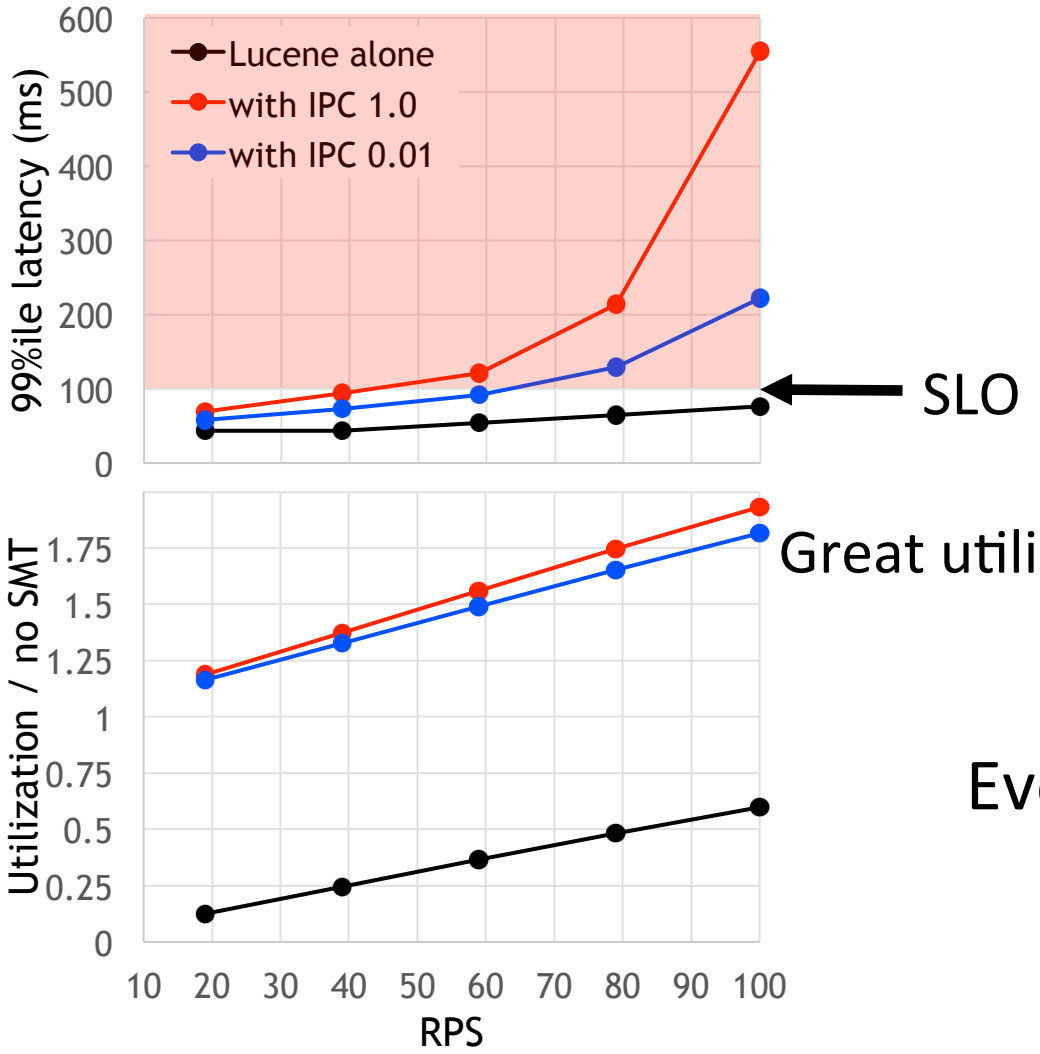
Co-running on same core
in SMT lanes

LC

batch

SMT Co-Runner

1 core, 2 SMT lanes

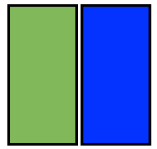


SLO

Great utilization!

```
while(1);
```

IPC 1.0

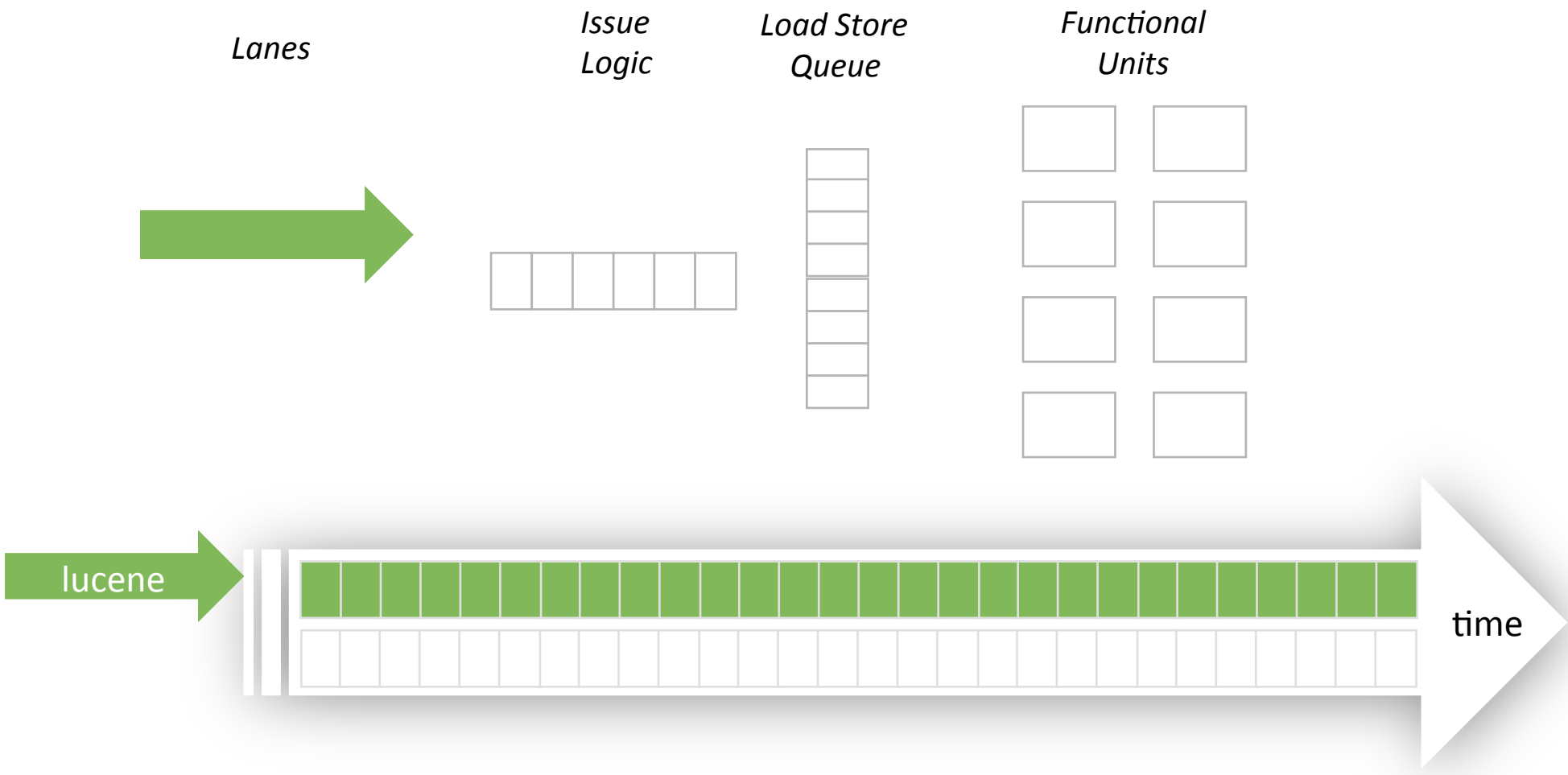


```
while(1) {
  movnti();
  mfence();
}
```

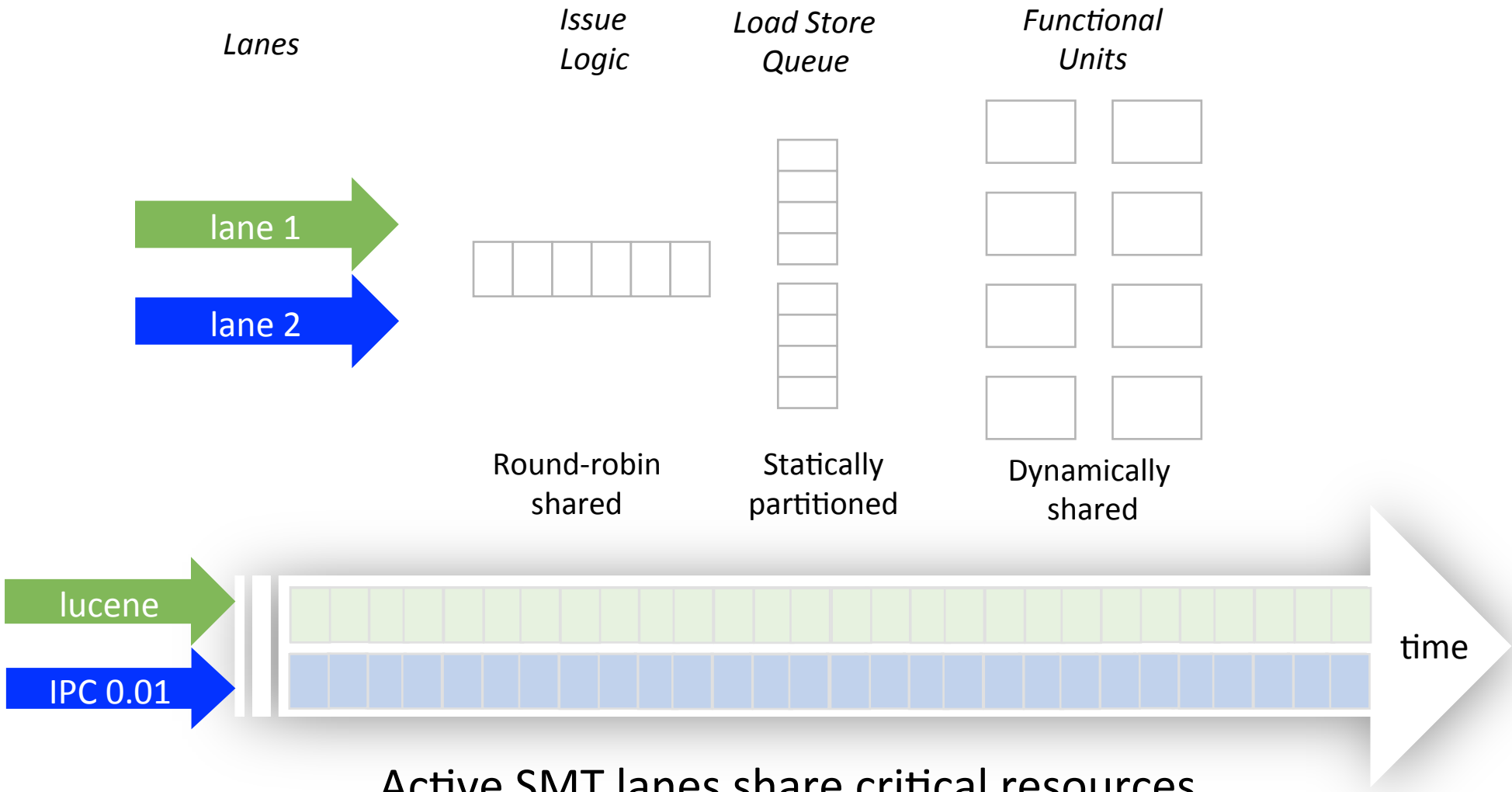
IPC 0.01

Even IPC 0.01 violates SLO at low load!

Simultaneous Multithreading OFF

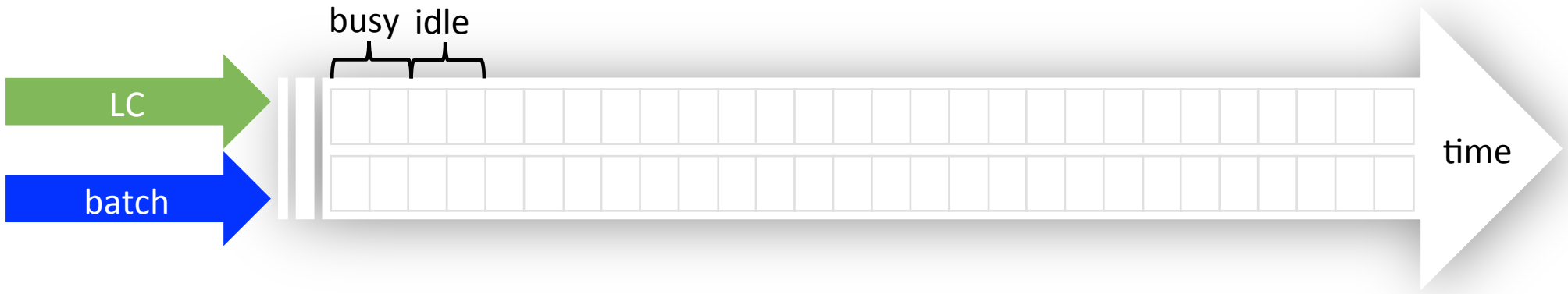


Simultaneous Multithreading ON



Active SMT lanes share critical resources

Principled Borrowing

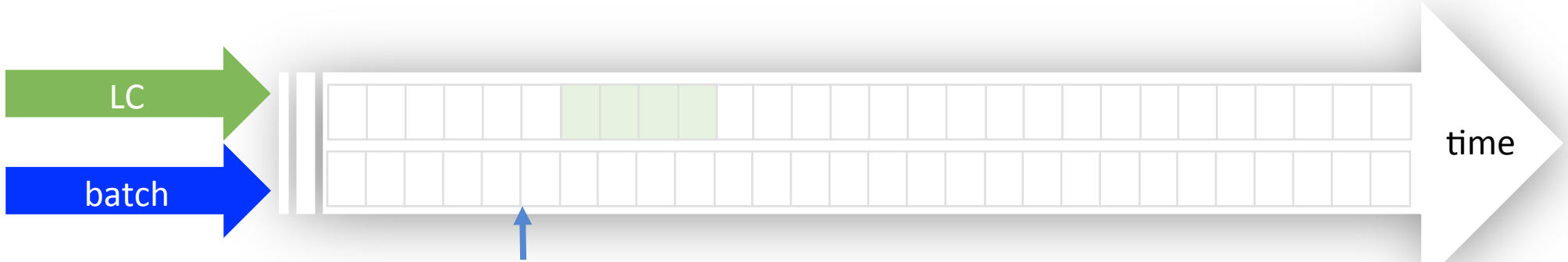


Batch borrows hardware when LC is idle

Batch releases hardware when LC is busy

Can we implement principled borrowing on current hardware?

Hardware is Ready – Software is Not



Batch lane calls “`mwait`”

Thread sleeps, releasing hardware to OS (~2K cycles)

OS schedules batch lane with any ready job

OS supports thread sleeping, but not hardware sleeping

release SMT hardware to other lane

nanonap()

Thread invoking nanonap releases SMT hardware without releasing SMT context

OS can interrupt & wakeup thread

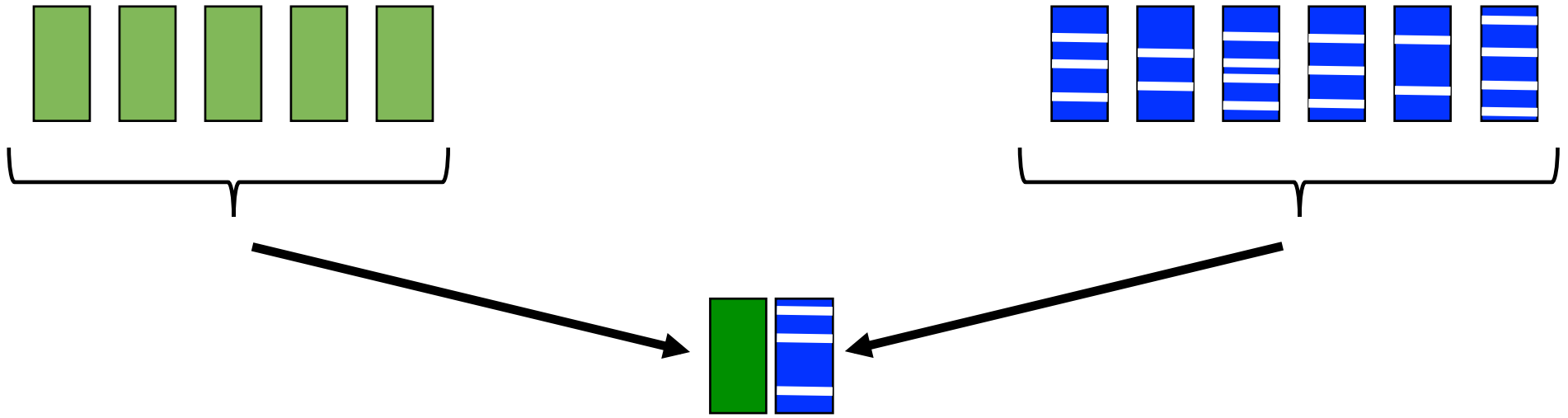
OS cannot schedule hardware context

```
per_cpu_variable: nap_flag;
void nanonap() {
    enter_kernel();
    disable_preemption();
    my_nap_flag = this_cpu_flag(nap_flag);
    monitor(my_nap_flag);
    mwait();
    enable_preemption();
    leave_kernel();
}
```

Elfen Scheduler

No change to latency-critical threads

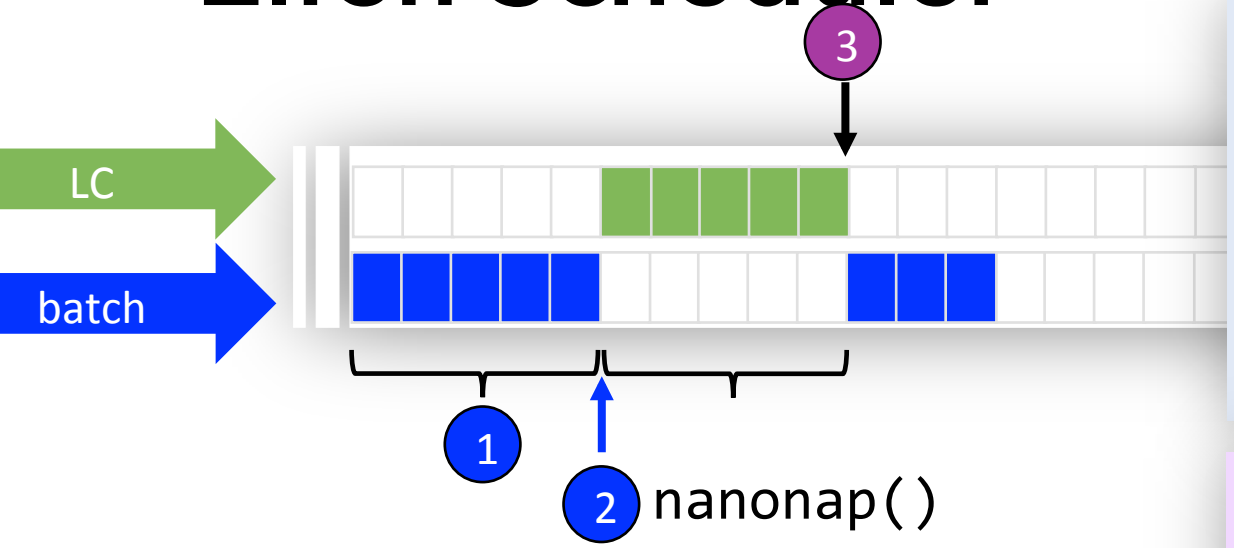
Instrument batch workloads to detect LC threads & nap



Bind latency-critical threads to LC lane

Bind batch threads to batch lane

Elfen Scheduler



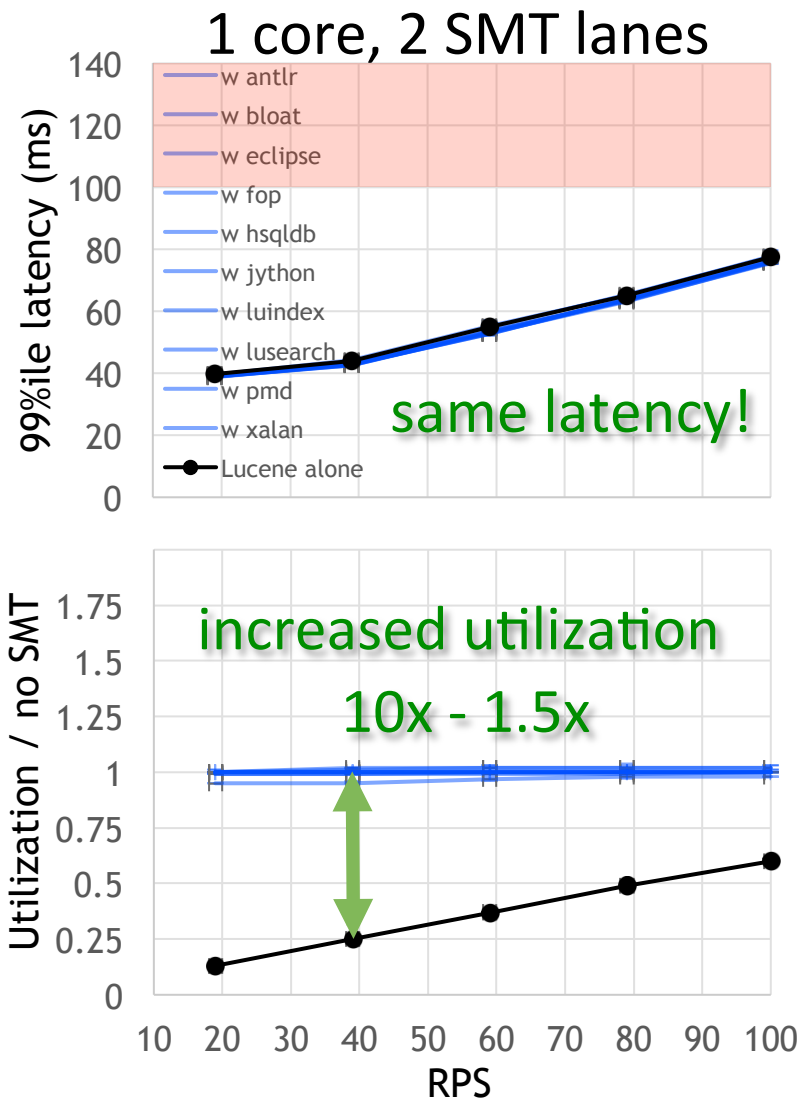
1. Batch thread borrows resources, continuously checks LC lane status
2. LC starts, batch calls `nanonap()` to release SMT hardware resources
3. OS touches `nap_flag` to wake up batch thread

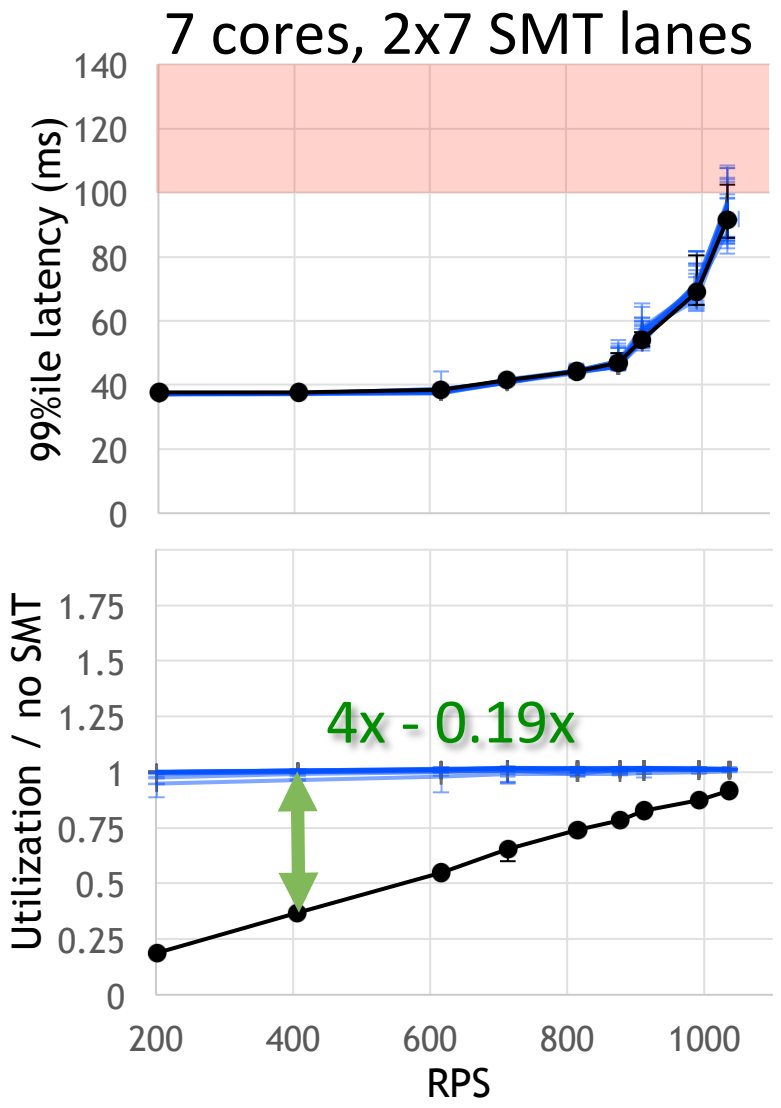
```
/* fast path check injected into method body */
    check:
1   if (!request_lane_idle)
        slow_path();

2   slow_path() {
        nanonap(); }
```

```
/*     maps lane IDs to the running task      */
    exposed SHIM signal: cpu_task_map
    task_switch(task T) {
       cpu_task_map[thiscpu] = T;
    }
    idle_task() {
3     // wake up any waiting batch thread
      update_nap_flag_of_partner_lane();
      ......
}
```

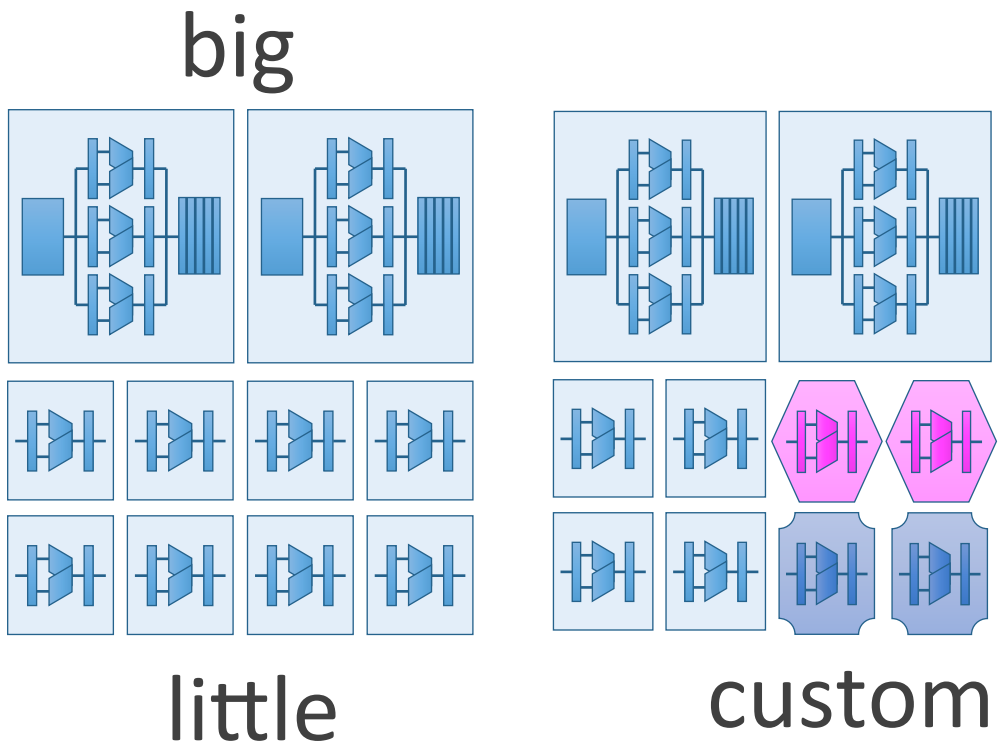
Results: Borrow Idle



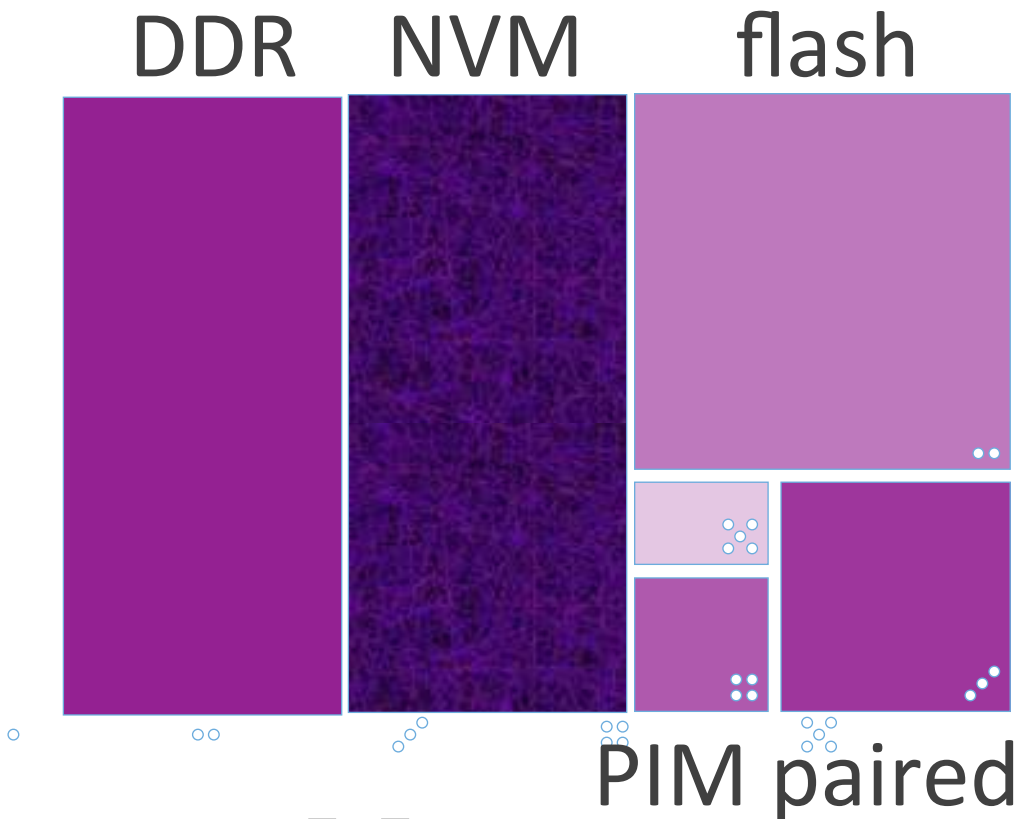


Exciting times

Hardware heterogeneity – opportunity & challenge

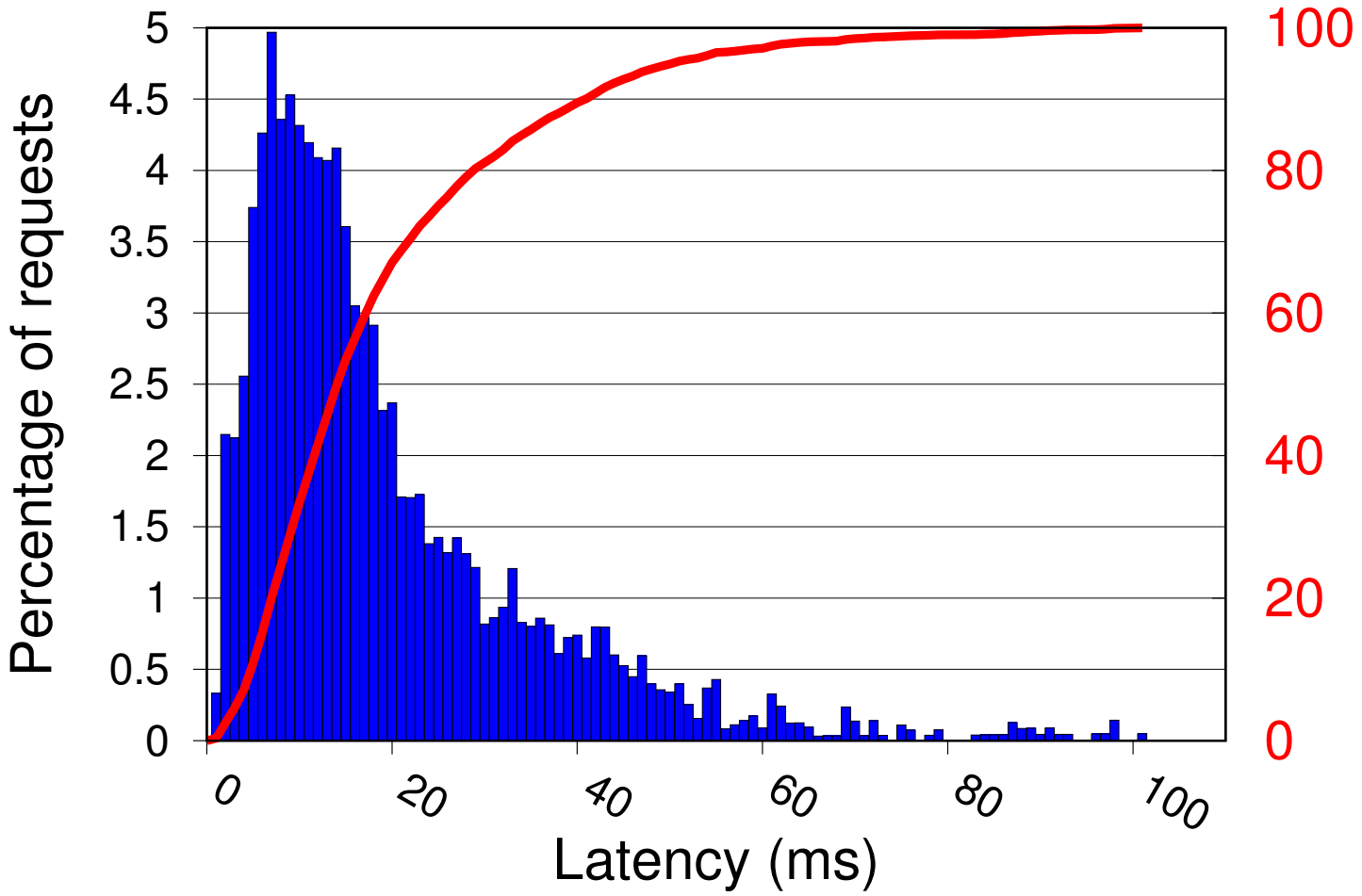


Processors



Memory

Heterogeneous workload!



Requirements pull for heterogeneity!

[DISC'14, ICAC'13, submission]

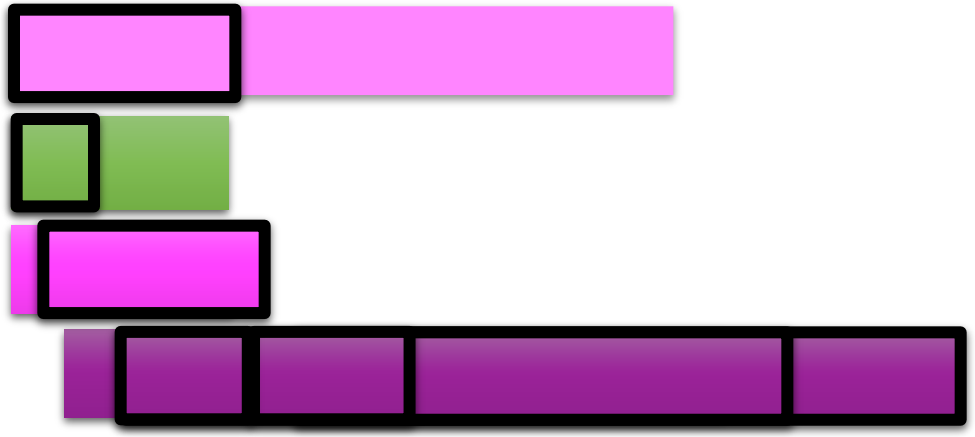
Heterogeneous hardware dominates homogeneous hardware for throughput, performance, and energy with a fixed power budget & variable request demand

Slow-to-Fast sacrifice average a bit to reduce energy & tail latency

Thank you

Extras

Online self scheduling



|requests|	$Interval_0 = 0$	$Interval_{1,2} = 50, 100$
≤ 2	@ 0 parallelism = 3	
3	@ 0 parallelism = 1	@ 50, parallelism = 3
4 - 6	@ 50 parallelism = 1	@ 100, parallelism = 3
≥ 7	@ exit parallelism = 1	@ 100, parallelism = 3

Software & hardware

Lucene open source enterprise search Wikipedia English

- 10 GB index of 33 million pages
- 10k queries from Lucene nightly tests

Bing web search with one Index Serving Node (ISN)

- 160 GB web index in SSD, 17 GB cache
- 30k Bing user queries

Hardware 2x8 64 bit 2.3 GHz Xeon, 64 GB Windows

- 15 request servers, 1 core issues requests
- Target parallelism = 24 threads

Policies

Sequential

N way single degree of parallelism for each request

Adaptive Select parallelism degree when request starts using system load [EUROSYS'13]

Request Clairvoyant parallelizes long requests by perfect prediction of tail

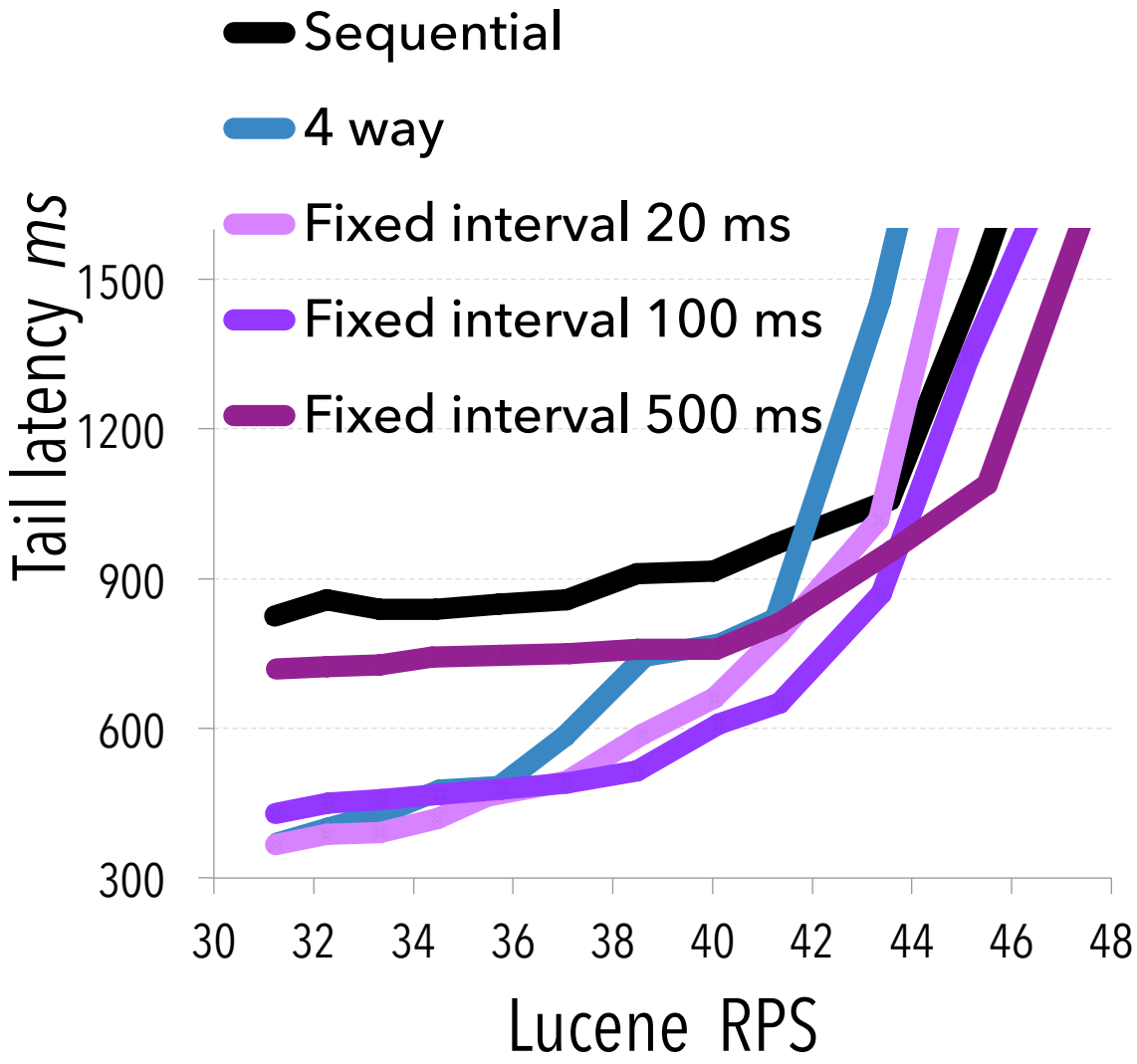
FM Few to Many incrementally add parallelism

Fixed interval

Add thread every X *ms*

Long intervals good at high load

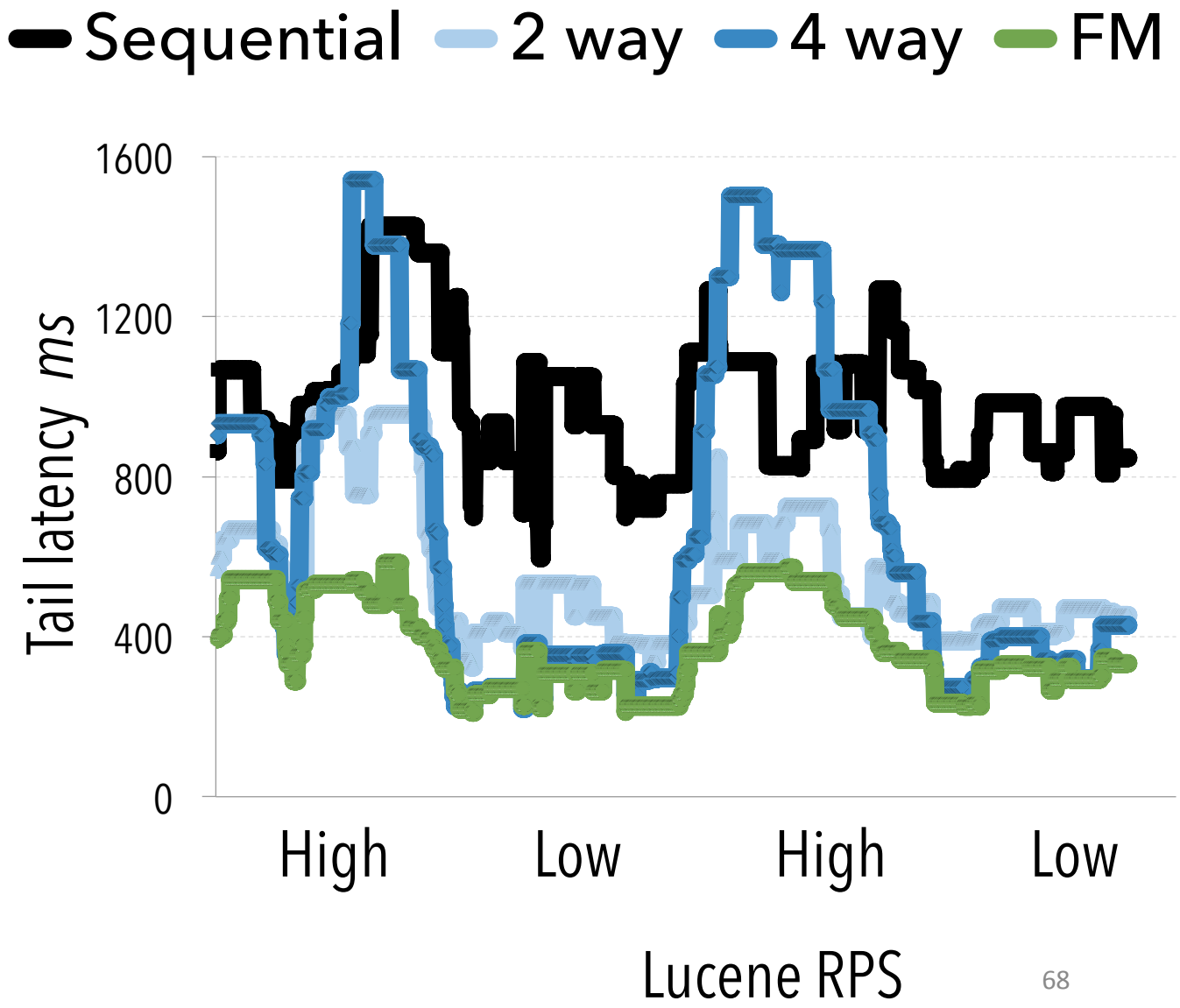
Short intervals good at low load



Load variation

Alternate between high & low load

FM adapts to bursts with low variance



Fewer servers: Total Cost of ownership