UNIVERSITY OF CAMBRIDGE

P51(bis): High Performance Networked-Systems

Prof. Andrew W. Moore

Lecture 4

With great appreciation to Dick Sites for sharing wisdom, patience, and teaching materials.

A **huge** thank you to Eben Upton, Raspberry Pi Foundation, and PiHut people for enabling this incarnation of the module at incredibly short notice.

Disclaimer

- Material is a snapshot of evolving (not established) wisdom
- Material is incomplete
  - many details on how and why datacenter networks operate aren’t public

Datacenter networks

10's to 100's of thousands of hosts, often closely coupled, in close proximity:

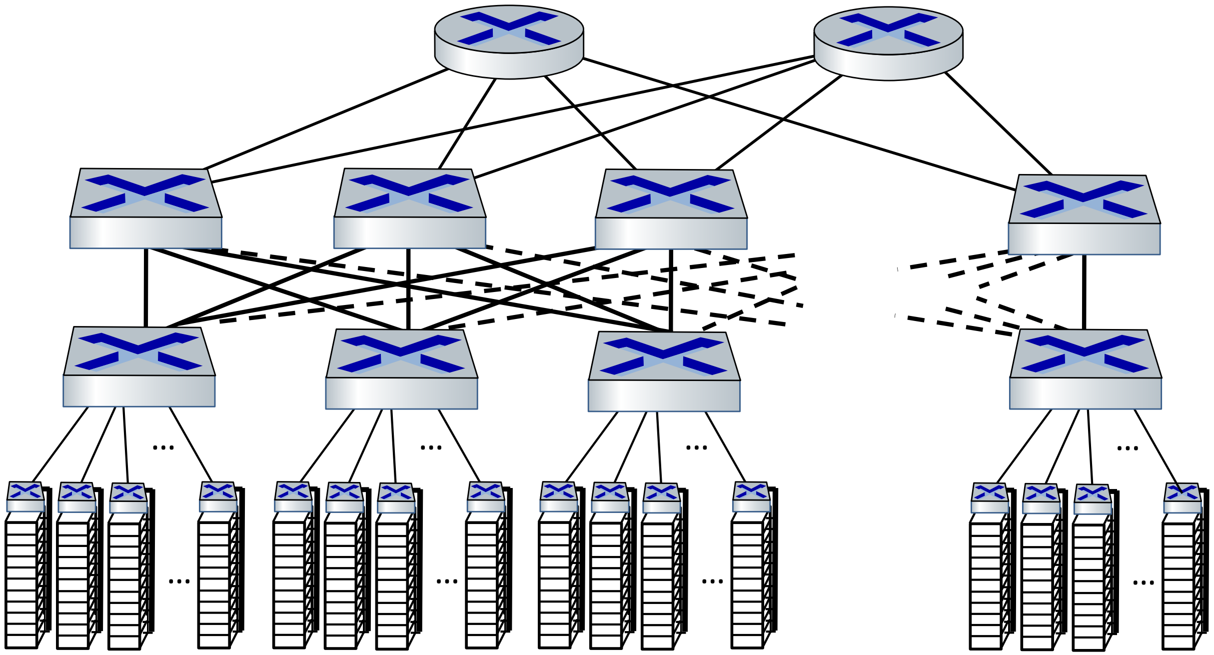
- e-business (e.g. Amazon)
- content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
- search engines, data mining (e.g., Google)

challenges:

- multiple applications, each serving massive numbers of clients
- reliability
- managing/balancing load, avoiding processing, networking, data bottlenecks

Inside a 40-ft Microsoft container, Chicago data center

Datacenter networks: network elements



Border routers

- connections outside datacenter

Tier-1 switches

- connecting to ~16 T-2s below

Tier-2 switches

- connecting to ~16 TORs below

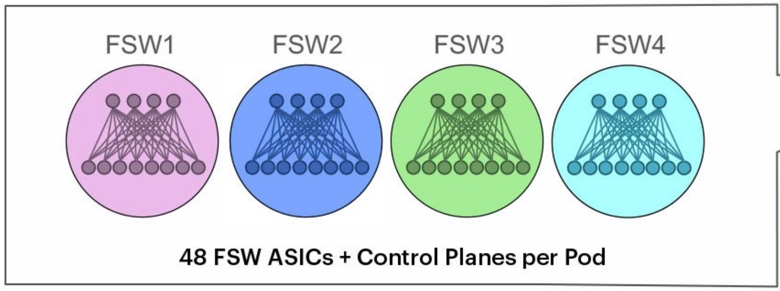
Top of Rack (TOR) switch

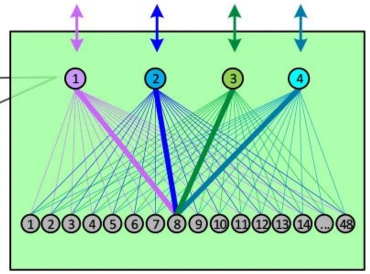
- one per rack
- 40-100Gbps Ethernet to blades

Server racks

- 20- 40 server blades: hosts

➔ from 4 x 128p multi-chip 400G fabric switches

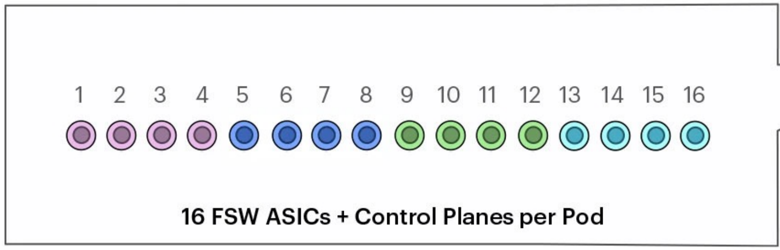


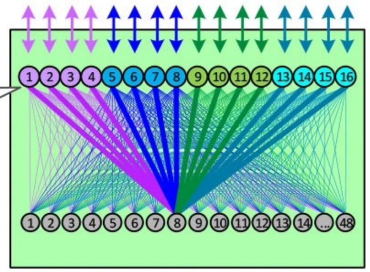


4 x 400G = 1.6T
uplink per rack

Sample Server Pod

➔ to 16 x 128p **single-chip 100G** fabric switches

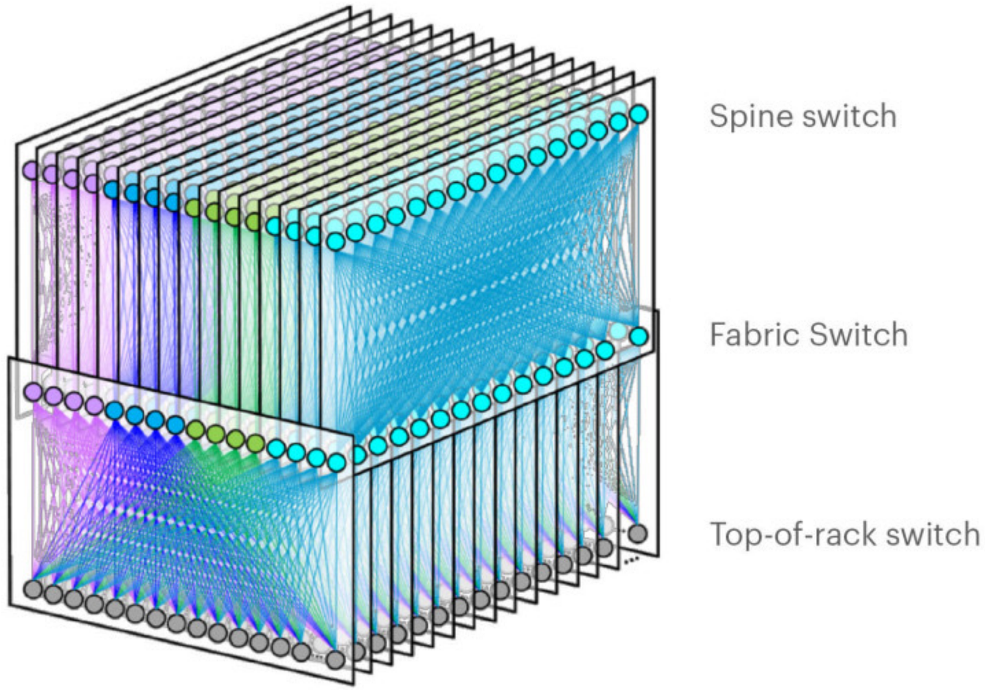




16 x 100G = 1.6T
uplink per rack

Datacenter networks: network elements

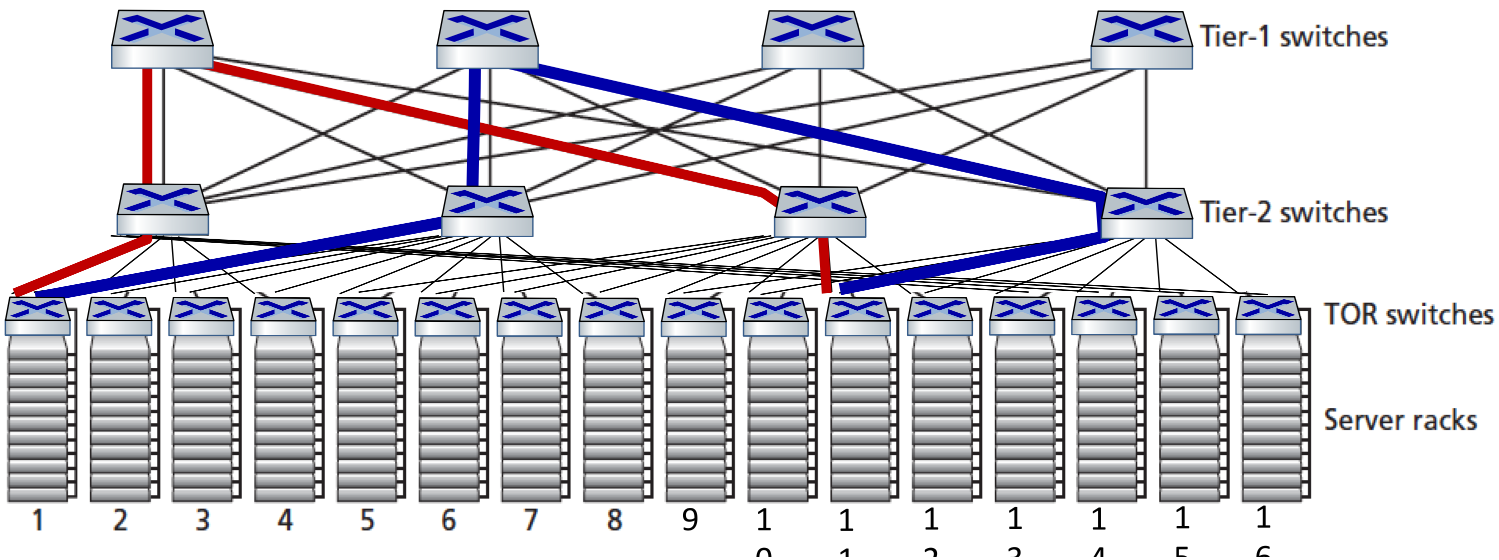
Facebook F16 data center network topology:



https://engineering.fb.com/data-center-engineering/f16-minipack/ (posted 3/2019)

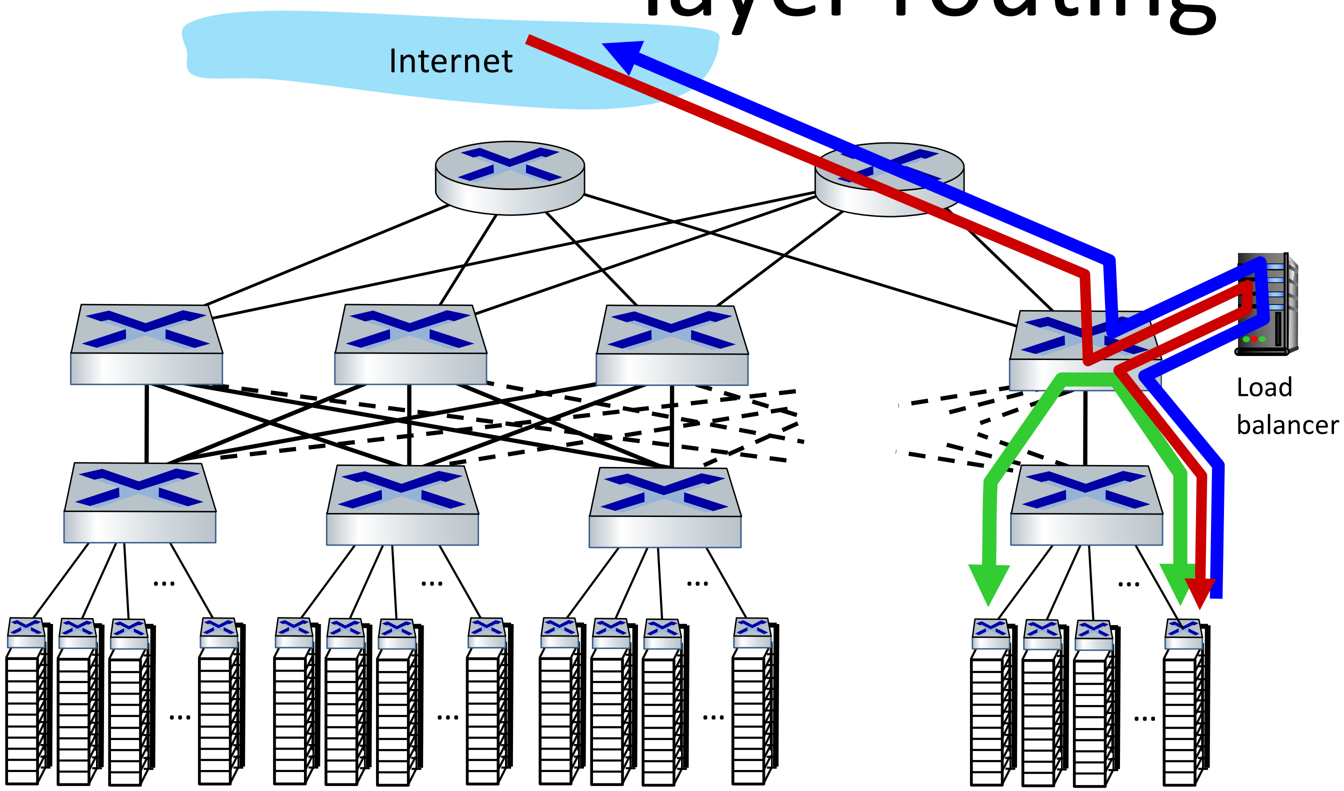
Datacenter networks: multipath

- rich interconnection among switches, racks:
  - increased throughput between racks (multiple routing paths possible)
  - increased reliability via redundancy



two disjoint paths highlighted between racks 1 and 11

Datacenter networks: application-layer routing



load balancer: application-layer routing

- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)

Observations on DC architecture

- Regular, well-defined arrangement
- Hierarchical structure with rack/aggr/core layers
- Mostly homogenous within a layer
- Supports communication between servers and between servers and the external world

Contrast: ad-hoc structure, heterogeneity of WANs

What’s different?

SCALE!

How big exactly?

- 1Million servers [Microsoft]
  - less than google, more than amazon

- > $1B to build one site [Facebook]

- >$20M/month/site operational costs [Microsoft '09]

But only O(10-100) sites

What’s new?

- Scale
- Service model
  - user-facing, revenue generating services
  - multi-tenancy
  - jargon: SaaS, PaaS, DaaS, IaaS, …

Implications

- Scale
  - need scalable solutions
  - improving efficiency, lowering cost is critical

  → *`scale out' solutions w/ commodity technologies*

- Service model
  - performance means $$
  - *virtualization* for isolation and portability

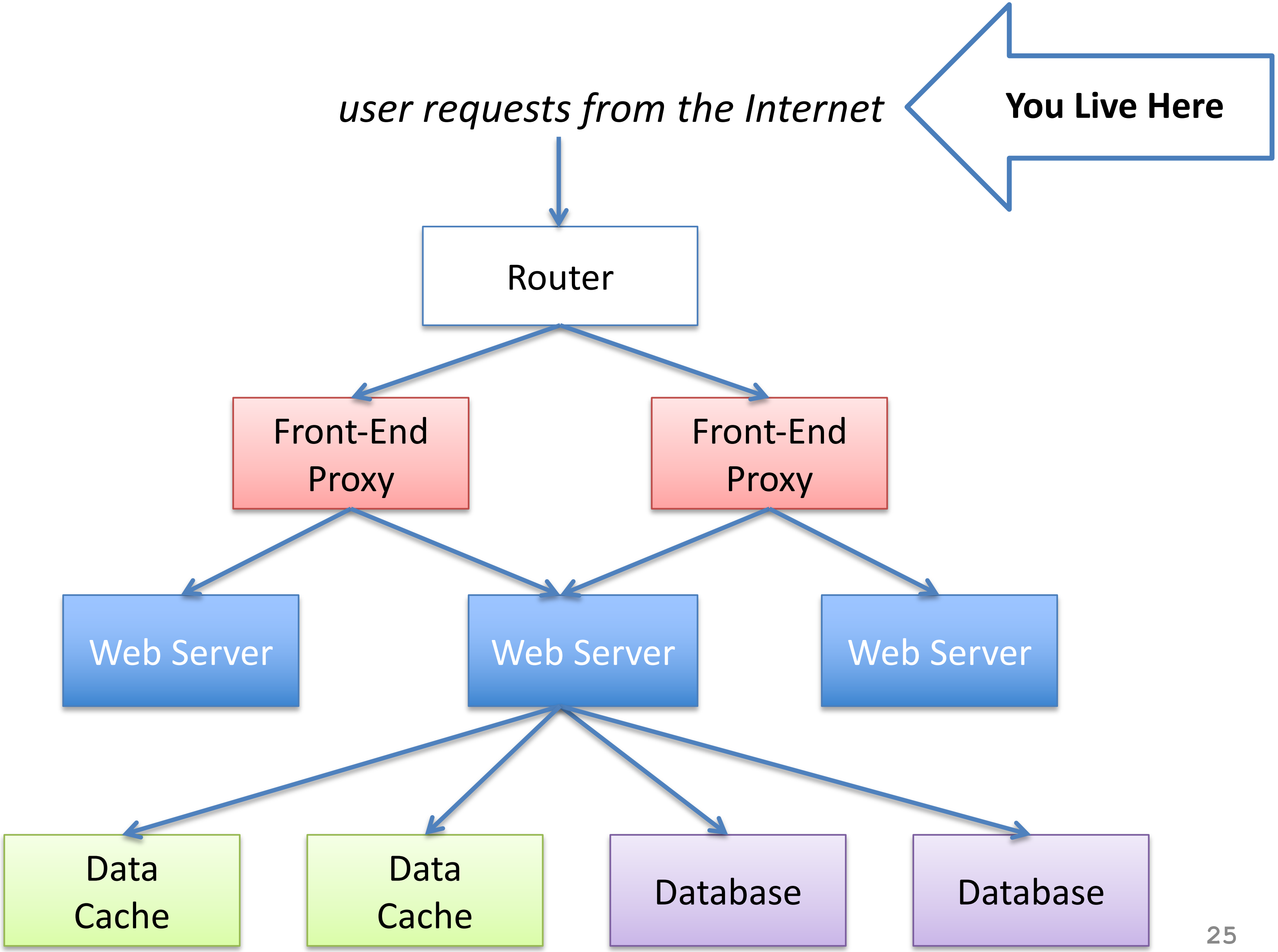
Multi-Tier Applications

- Applications decomposed into tasks
  - Many separate components
  - Running in parallel on different machines

Componentization leads to different types of network traffic

- "North-South traffic"
  - Traffic between external clients and the datacenter
  - Handled by front-end (web) servers, mid-tier application servers, and back-end databases
  - Traffic patterns fairly stable, though diurnal variations

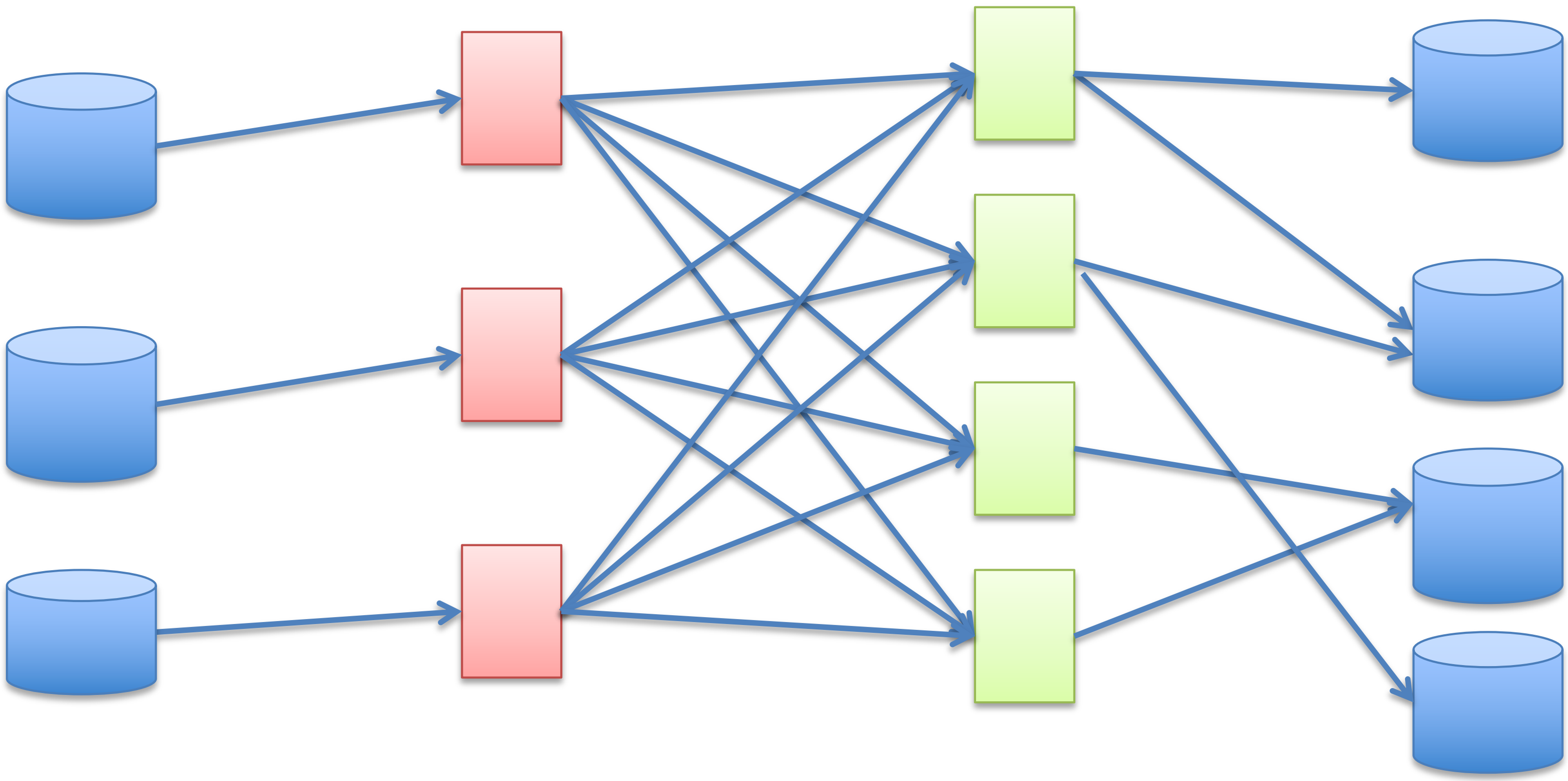
North-South Traffic



Componentization leads to different types of network traffic

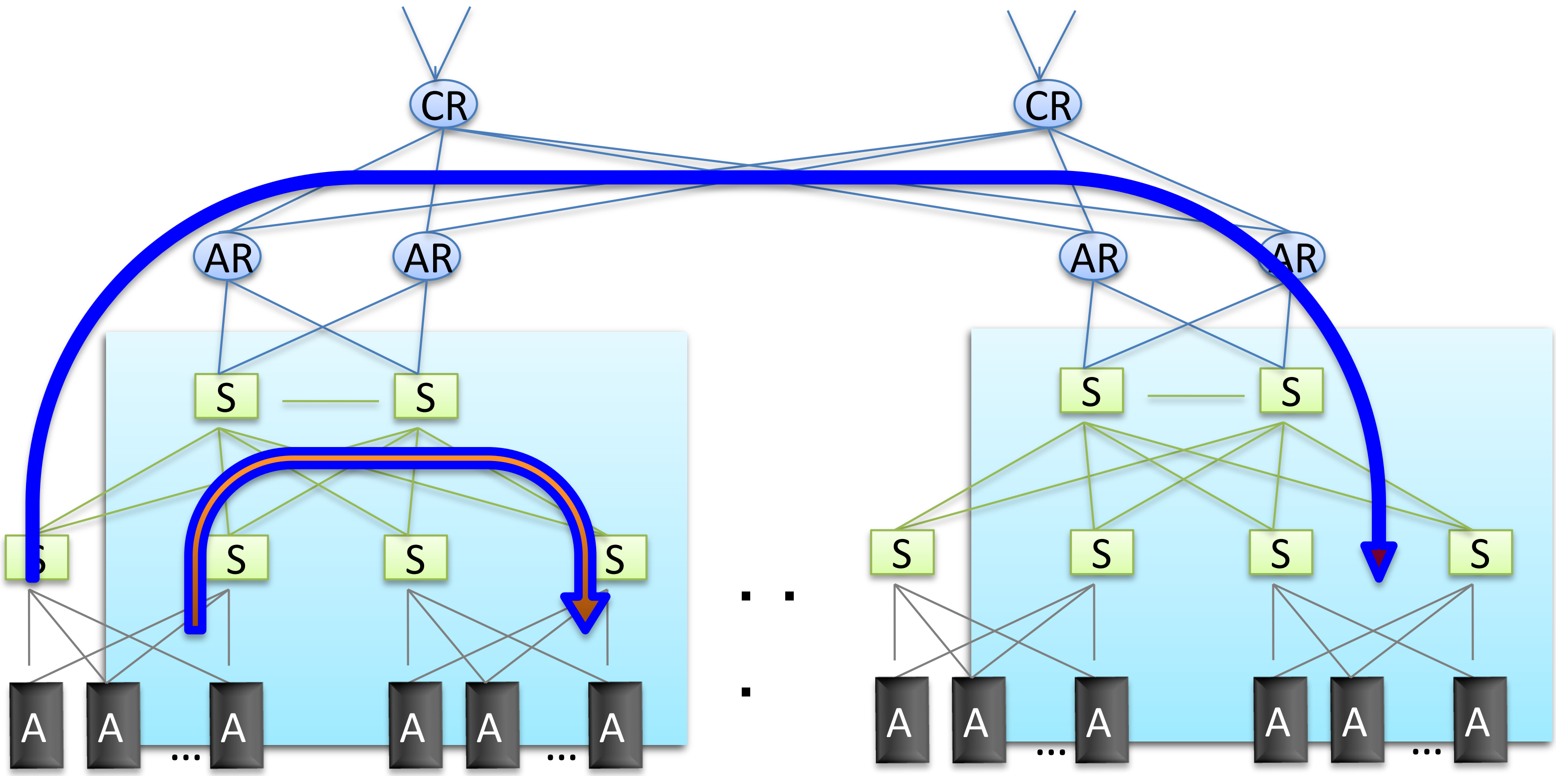
- "North-South traffic"
  - Traffic between external clients and the datacenter
  - Handled by front-end (web) servers, mid-tier application servers, and back-end databases
  - Traffic patterns fairly stable, though diurnal variations
- "East-West traffic"
  - Traffic between machines in the datacenter
  - Comm *within* "big data" computations (e.g. Map Reduce)
  - Traffic may shift on small timescales (e.g., minutes)

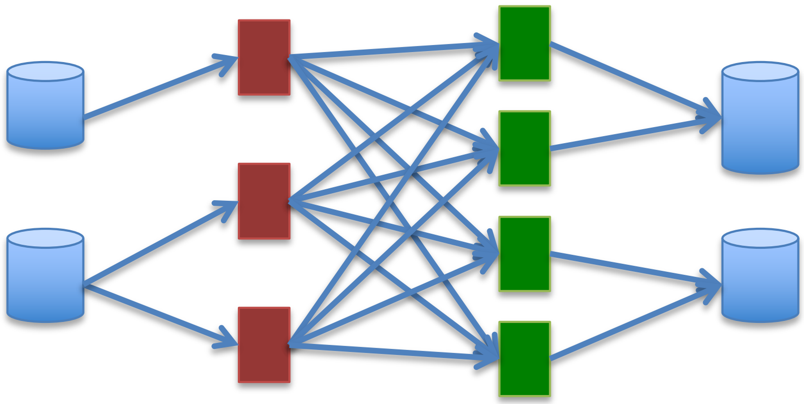
East-West Traffic

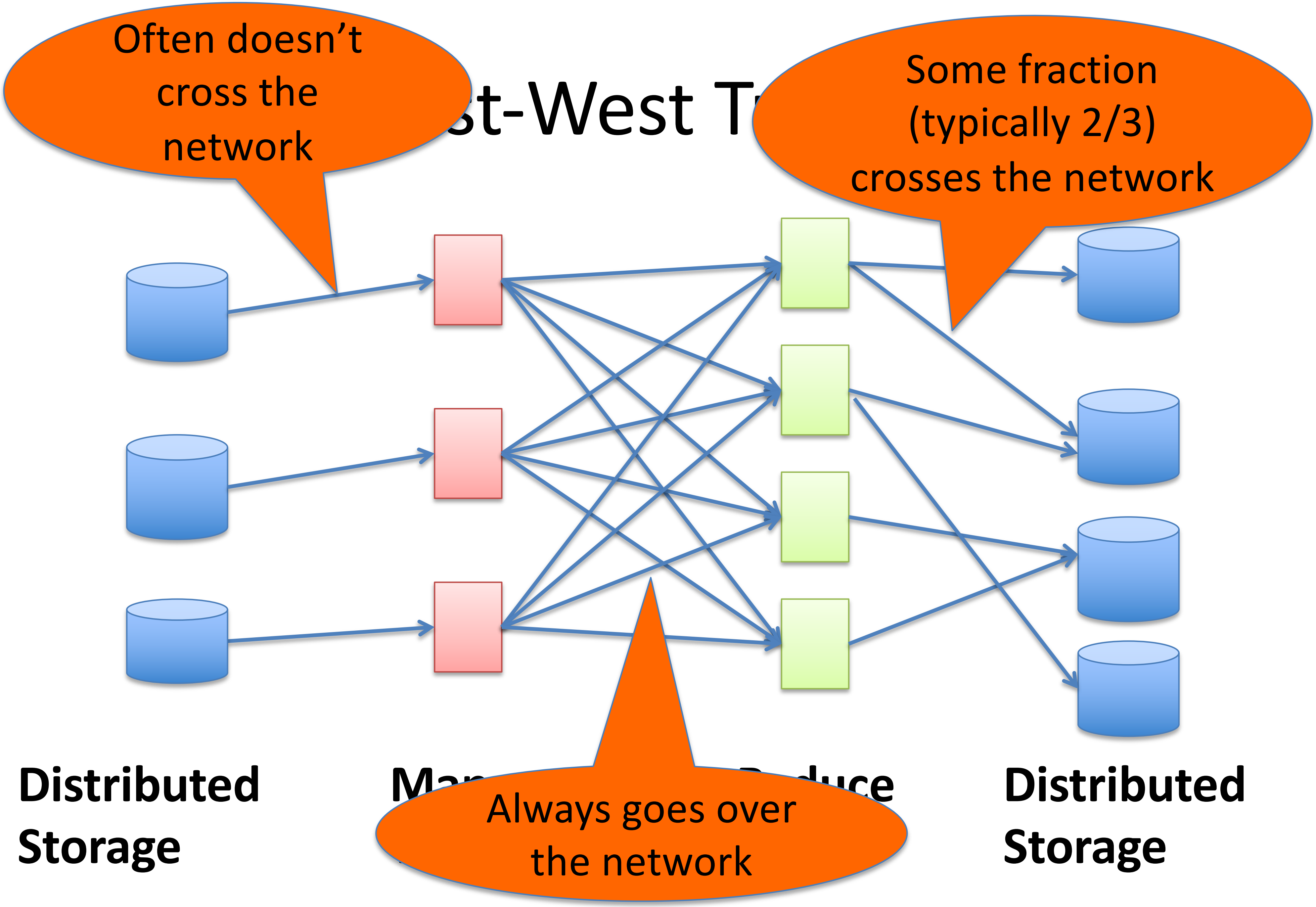


Distributed Storage | **Map Tasks** | **Reduce Tasks** | **Distributed Storage**

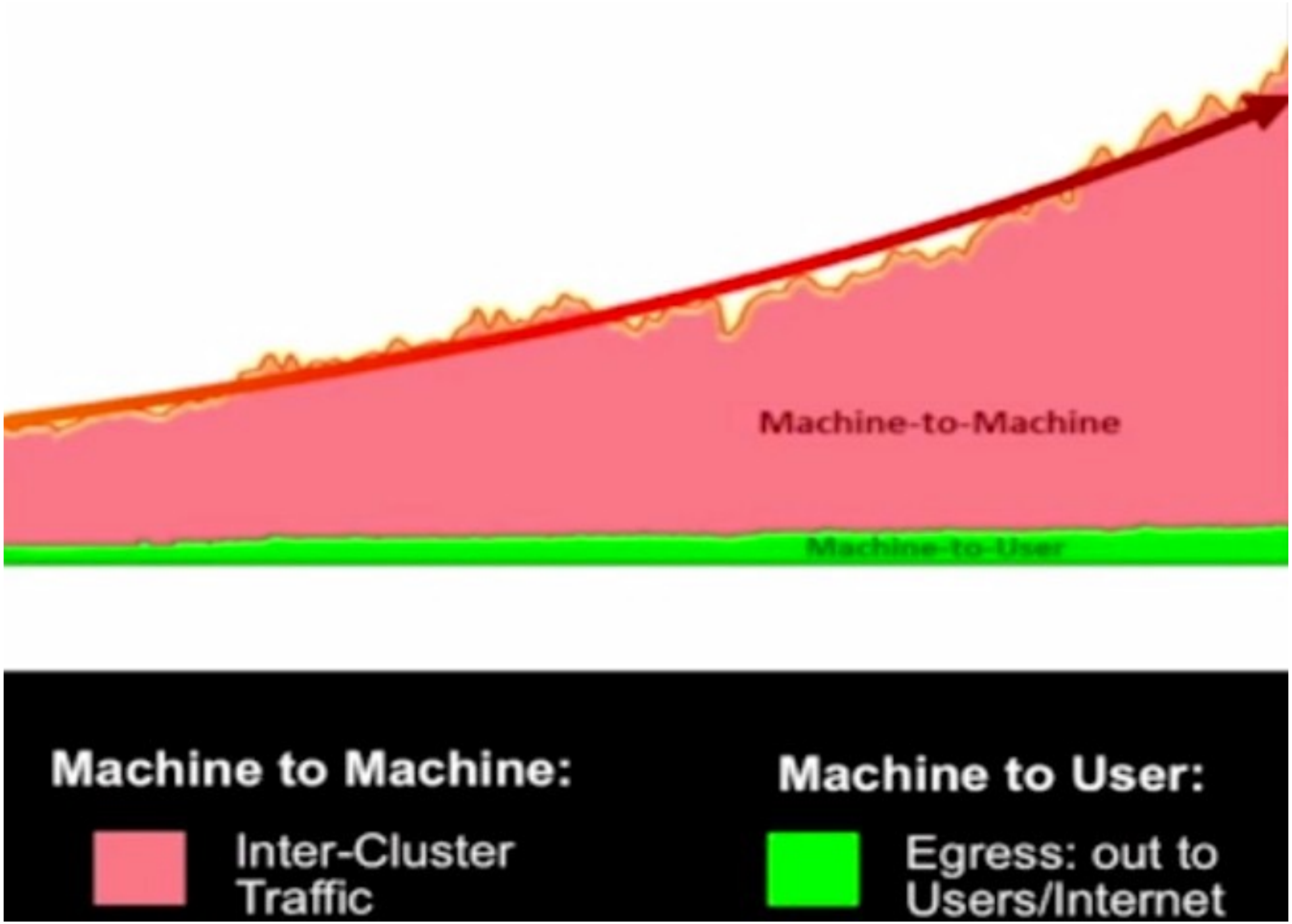
East-West Traffic






Often doesn't cross the network
Some fraction (typically 2/3) crosses the network
Distributed Storage
Always goes over the network
Distributed Storage

East-West vs North-South



What's different about DC networks?

Characteristics

- Huge scale:
  - ~20,000 switches/routers
  - *contrast: AT&T ~500 routers*

What’s different about DC networks?

Characteristics

- Huge scale:
- Limited geographic scope:
  - High bandwidth: 10/40/100G
  - *Contrast: Cable/aDSL/WiFi*
  - Very low RTT: 10s of microseconds
  - *Contrast: 100s of milliseconds in the WAN*

What's different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
  - Can deviate from standards, invent your own, *etc.*
  - "Green field" deployment is still feasible

What's different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
  - can change (say) addressing, congestion control, *etc.*
  - can add mechanisms for security/policy/etc. at the endpoints (typically in the hypervisor)

What's different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
- Control over the *placement* of traffic source/sink
  - e.g., map-reduce scheduler chooses where tasks run
  - alters traffic pattern (what traffic crosses which links)

What's different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
- Control over the *placement* of traffic source/sink
- Regular/planned topologies (e.g., trees/fat-trees)
  - Contrast: ad-hoc WAN topologies (dictated by real-world geography and facilities)

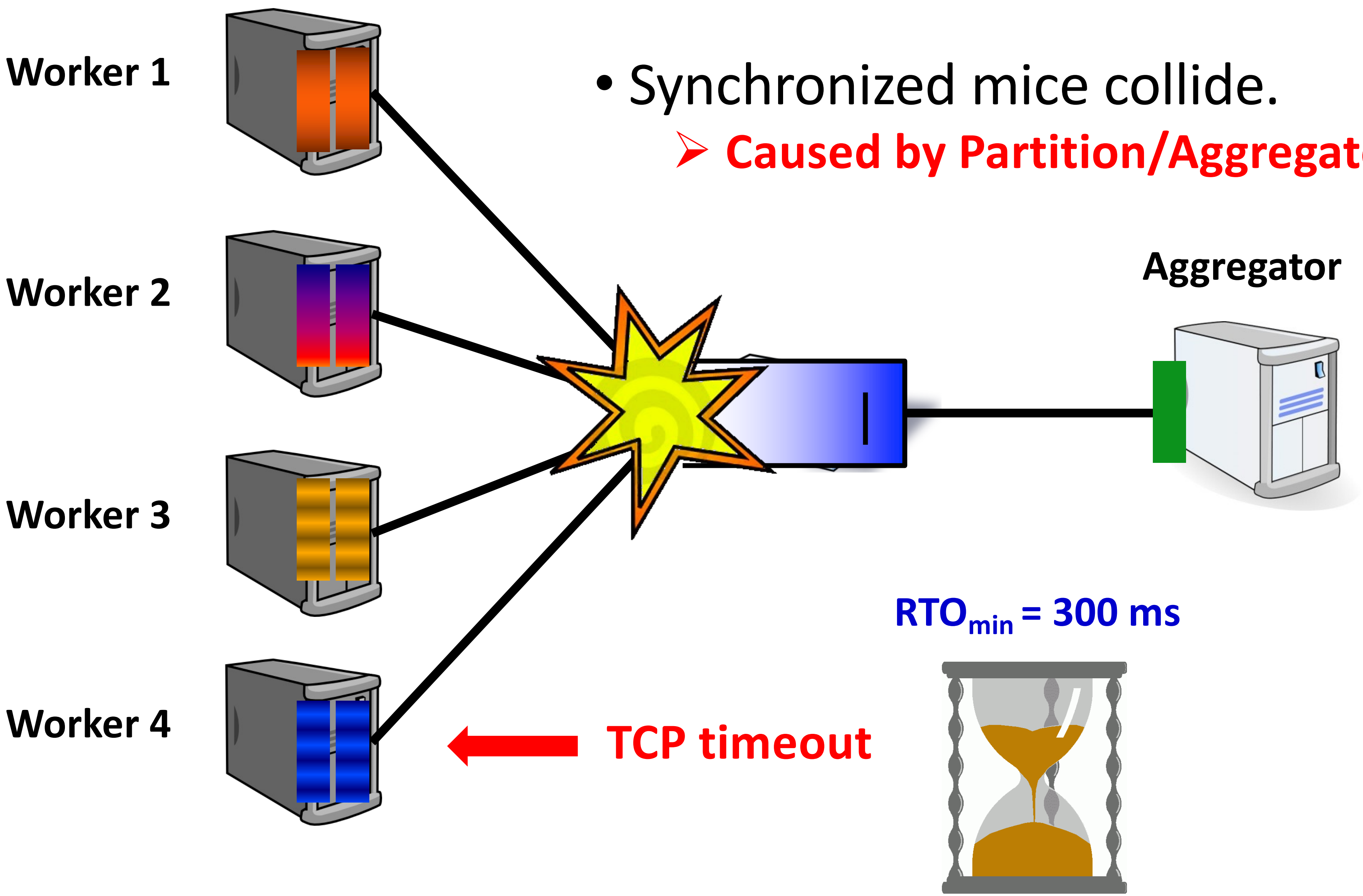
What’s different about DC networks?

<u>Characteristics</u>

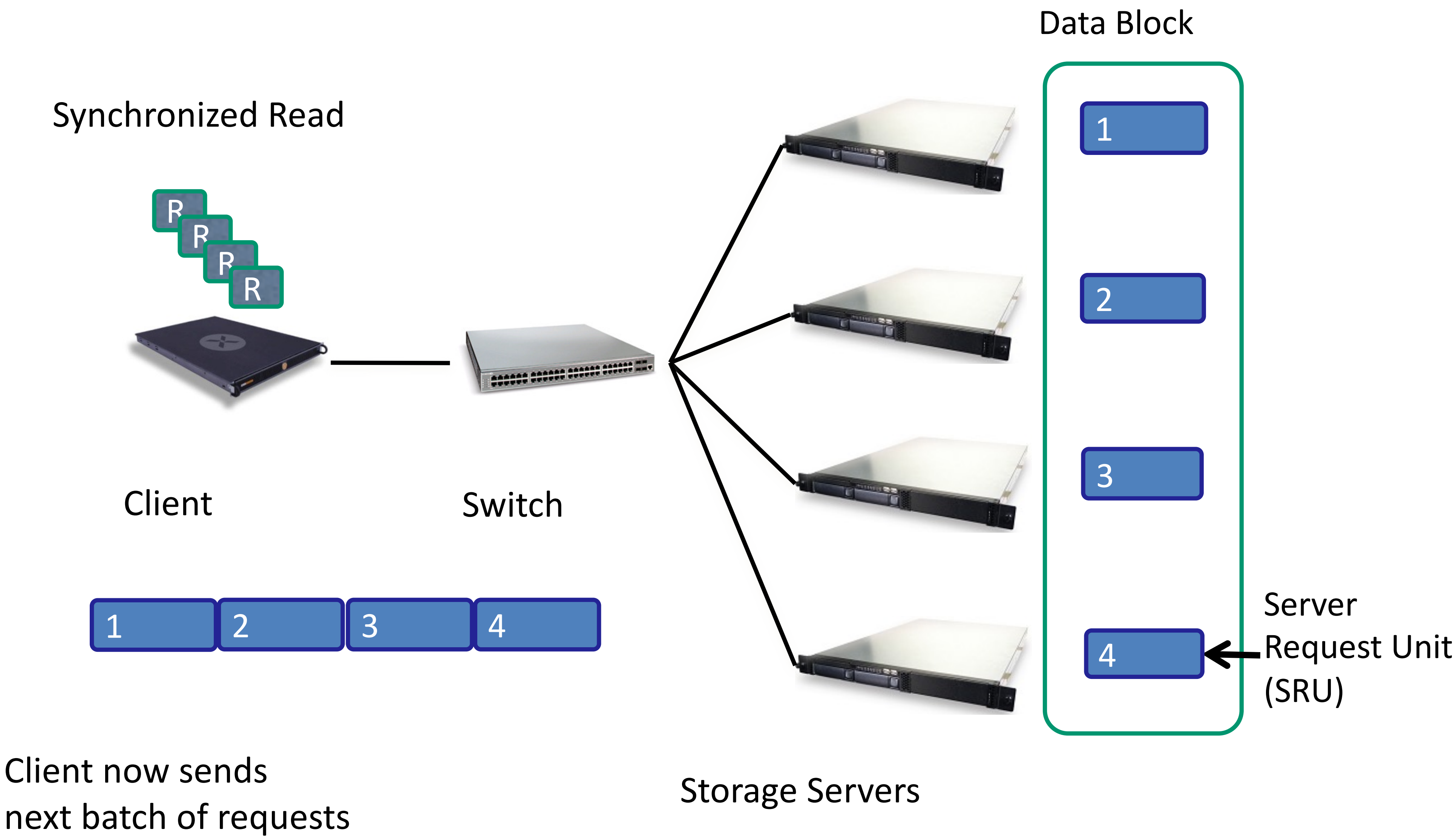
- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
- Control over the *placement* of traffic source/sink
- Regular/planned topologies (e.g., trees/fat-trees)
- Limited heterogeneity
  - link speeds, technologies, latencies, …

An example problem at scale - INCAST

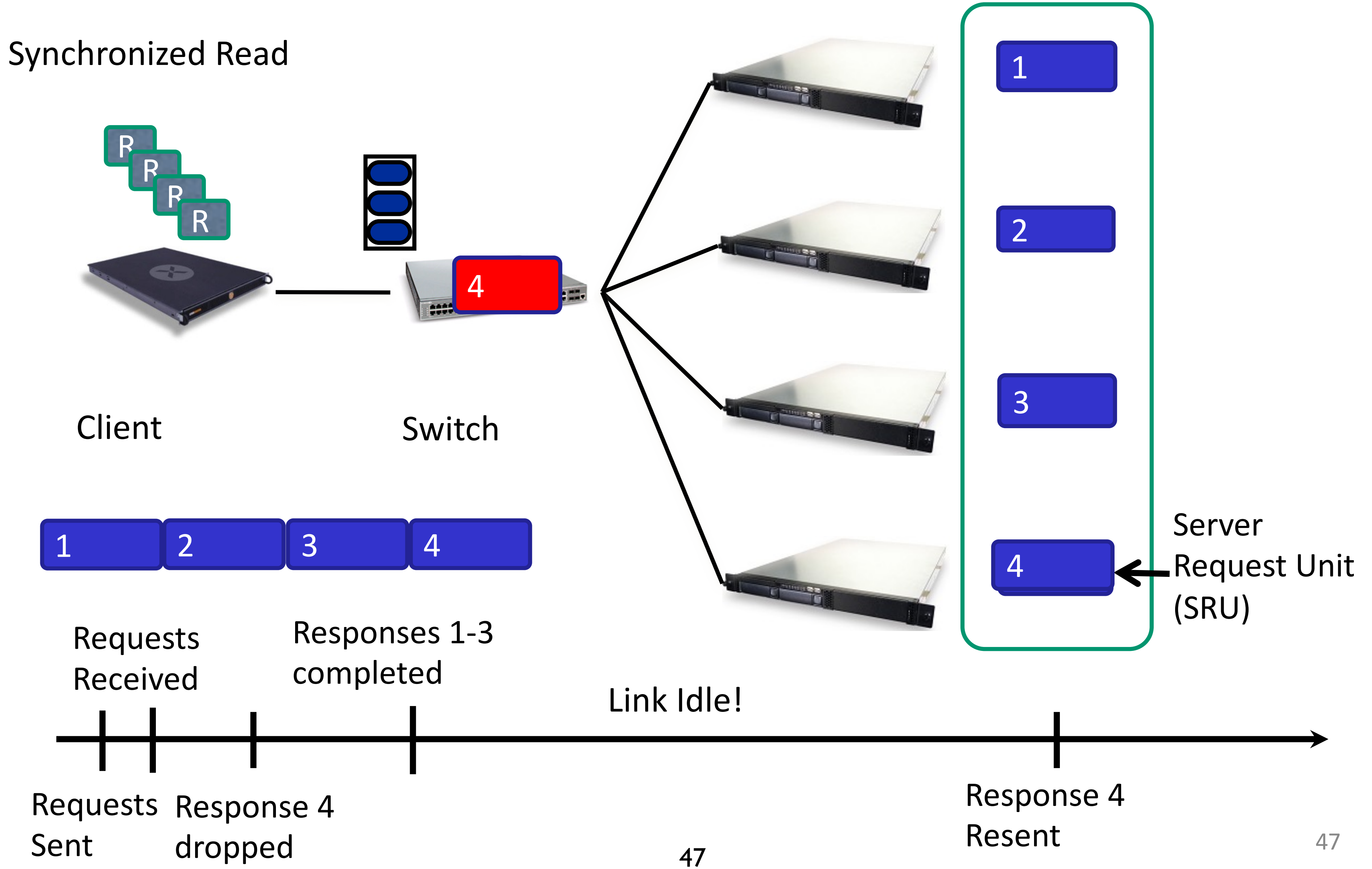
- Synchronized mice collide.
  - **Caused by Partition/Aggregate.**



The Incast Workload

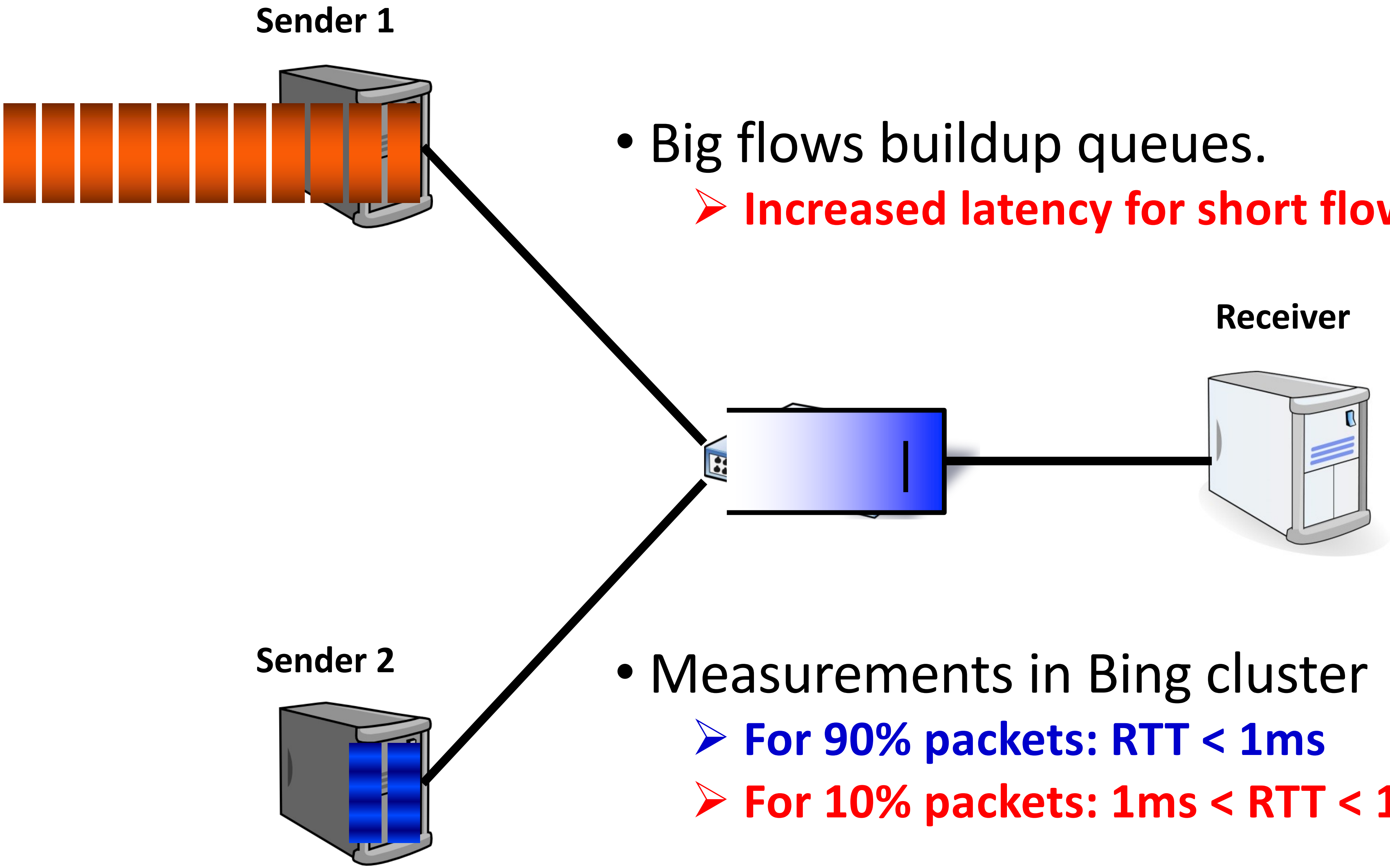


Incast Workload Overfills Buffers



47

Queue Buildup



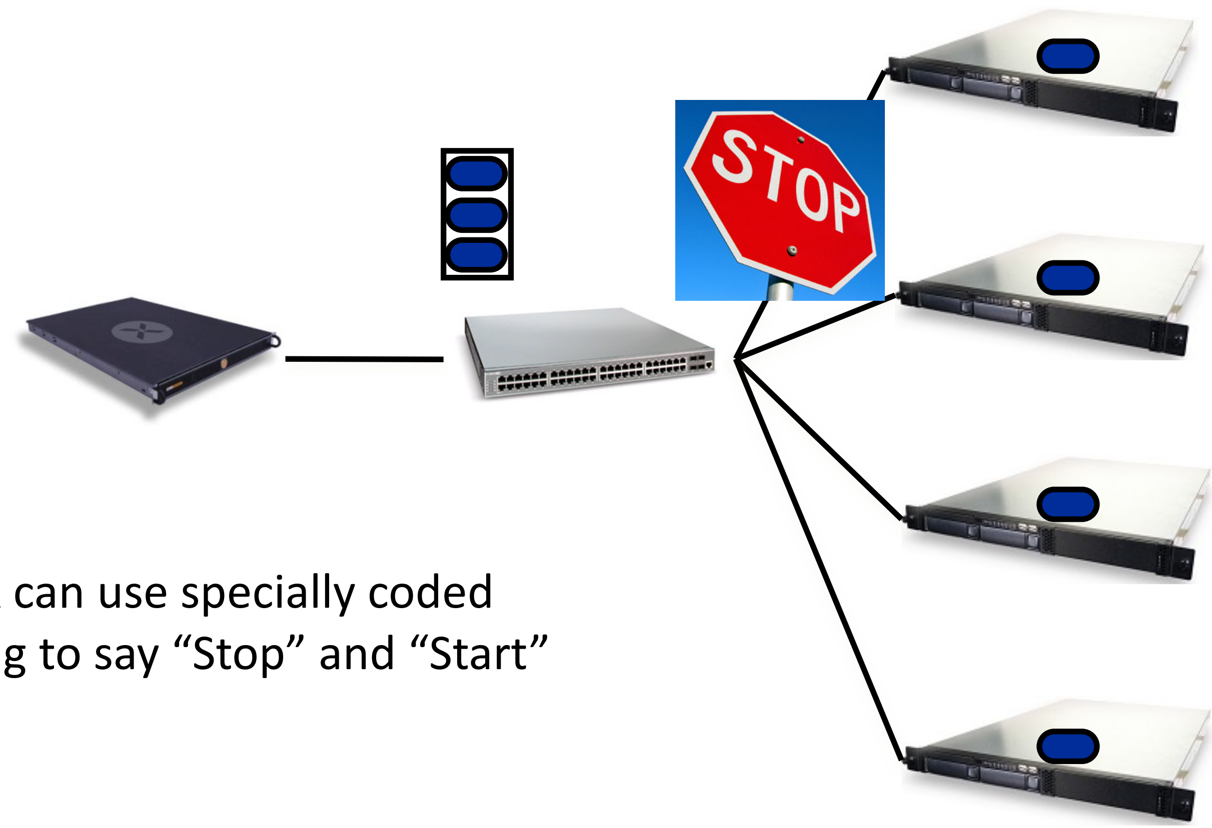
- Big flows buildup queues.
  - ➢ **Increased latency for short flows.**

- Measurements in Bing cluster
  - ➢ **For 90% packets: RTT < 1ms**
  - ➢ **For 10% packets: 1ms < RTT < 15ms**

Link-Layer Flow Control

Common between switches but this is flow-control to the end host too…

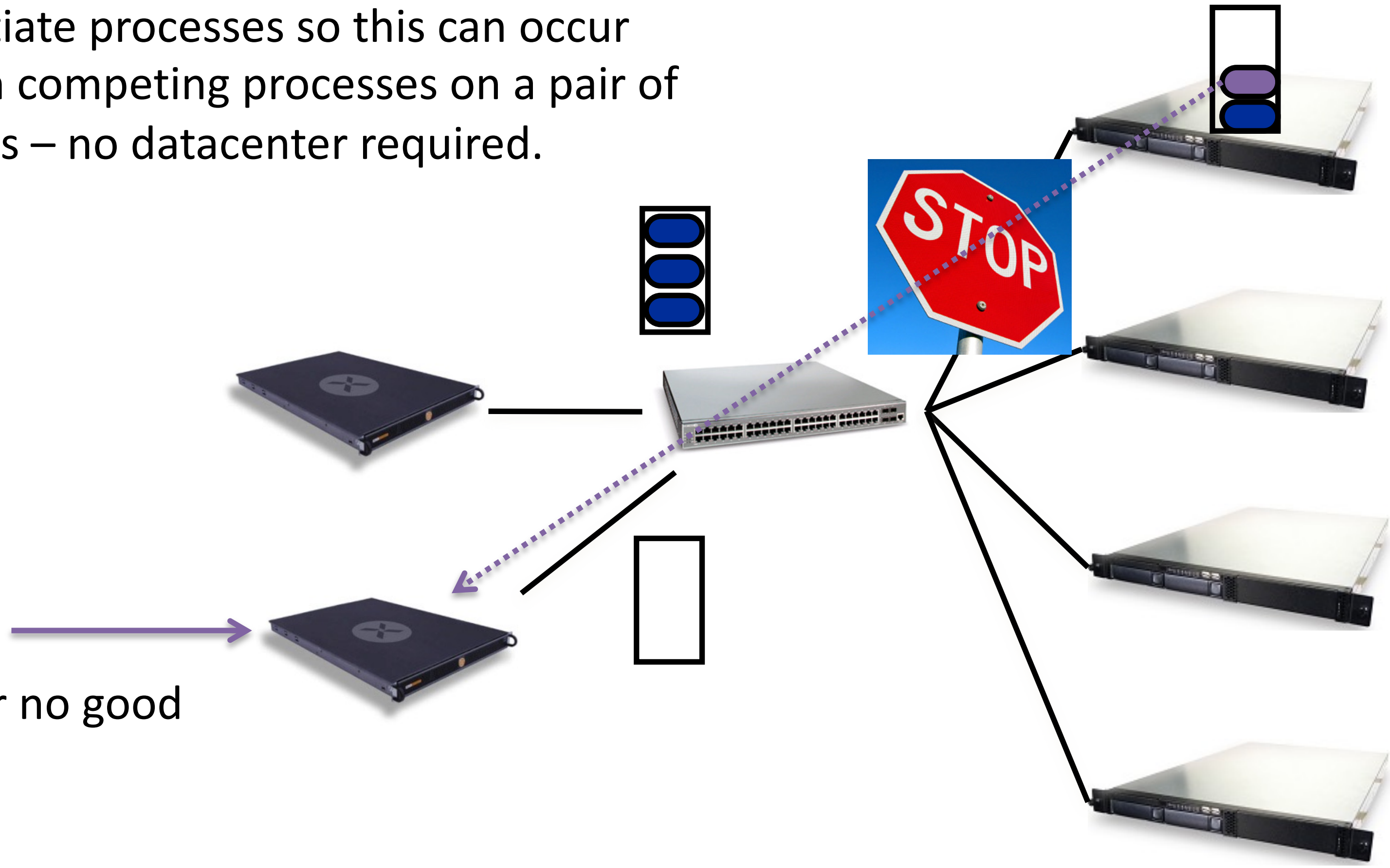
- Another idea to reduce incast is to employ Link-Layer Flow Control…..



Recall: the Data-Link can use specially coded symbols in the coding to say "Stop" and "Start"

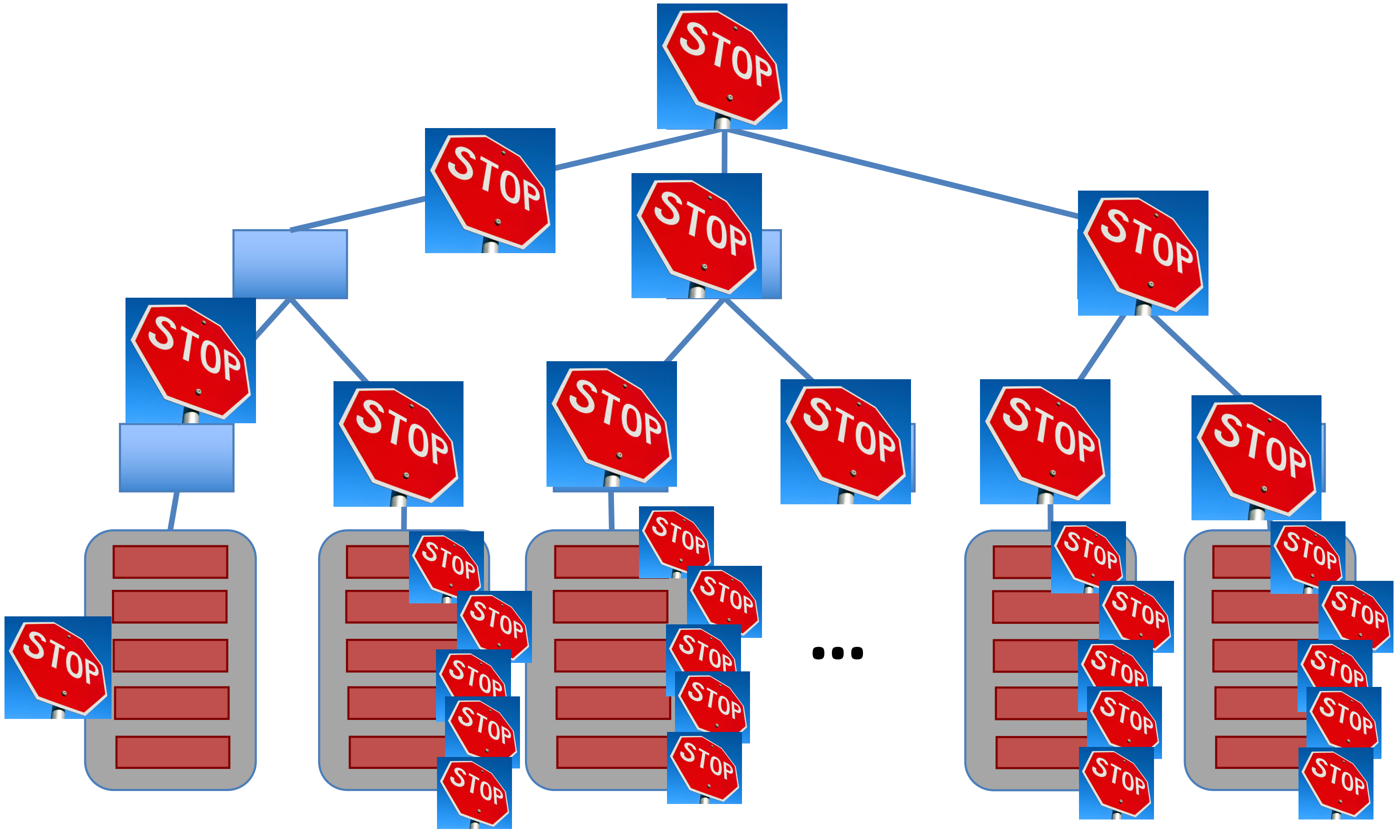
Link Layer Flow Control – The Dark side Head of Line Blocking….

Such HOL blocking does not even differentiate processes so this can occur between competing processes on a pair of machines – no datacenter required.



Waiting for no good reason….

Link Layer Flow Control
But its worse that you imagine....



Double down on trouble....

Did I mention this is Link-Layer!

That means no (IP) control traffic, no routing messages....

a whole system waiting for one machine

Incast is very unpleasant.

Reducing the impact of Head of Line (blocking) in Link Layer Flow Control can be done through priority queues and *overtaking*....

Enough with Datacenter networks, what about CPU?

Datacenter Servers are Different

- Server programs start and run for months
  - Tens of programs per server
- Real-time user-facing transactions, not batch
  - Minimize latency, not CPU idle time
- Big fanout: one transaction to 1000+ servers
  - only 1-10ms CPU time per server
- 10K+ computers, 100K+ disks, 1000k+ network connections

Datacenter Servers are Different

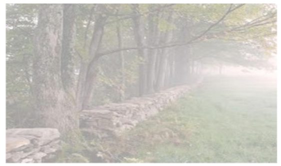
① Move data: big and small

② Real-time transactions: 1000s per second

③ Isolation between programs

④ Measurement underpinnings





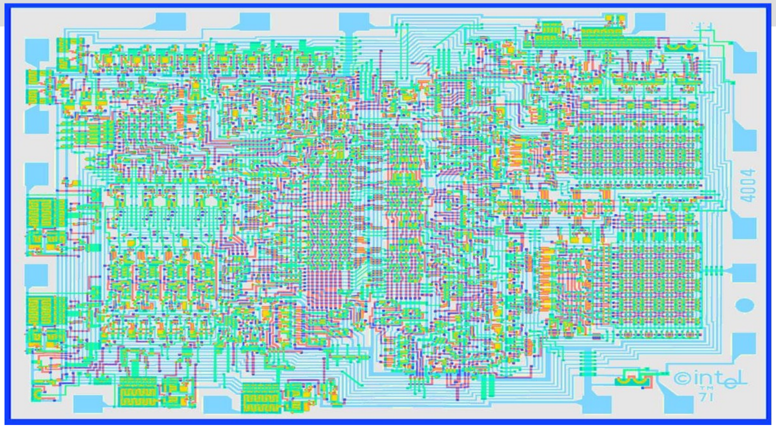

① Move data: big and small

Move data: big and small

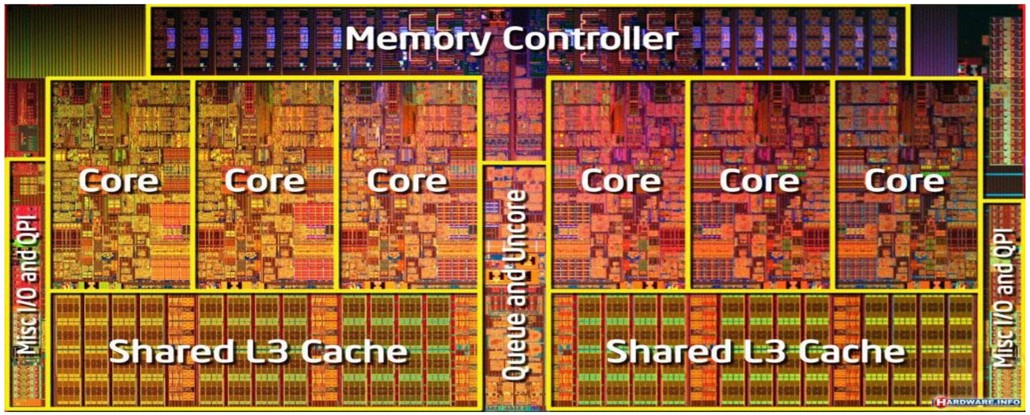
- Move *lots* of data
  - Disk to/from RAM
  - Network to/from RAM
  - SSD to/from RAM
  - Within RAM
- Bulk data
- *Short* data: variable-length items
- Compress, encrypt, checksum, hash, sort

Lots of memory

- 4004: no memory



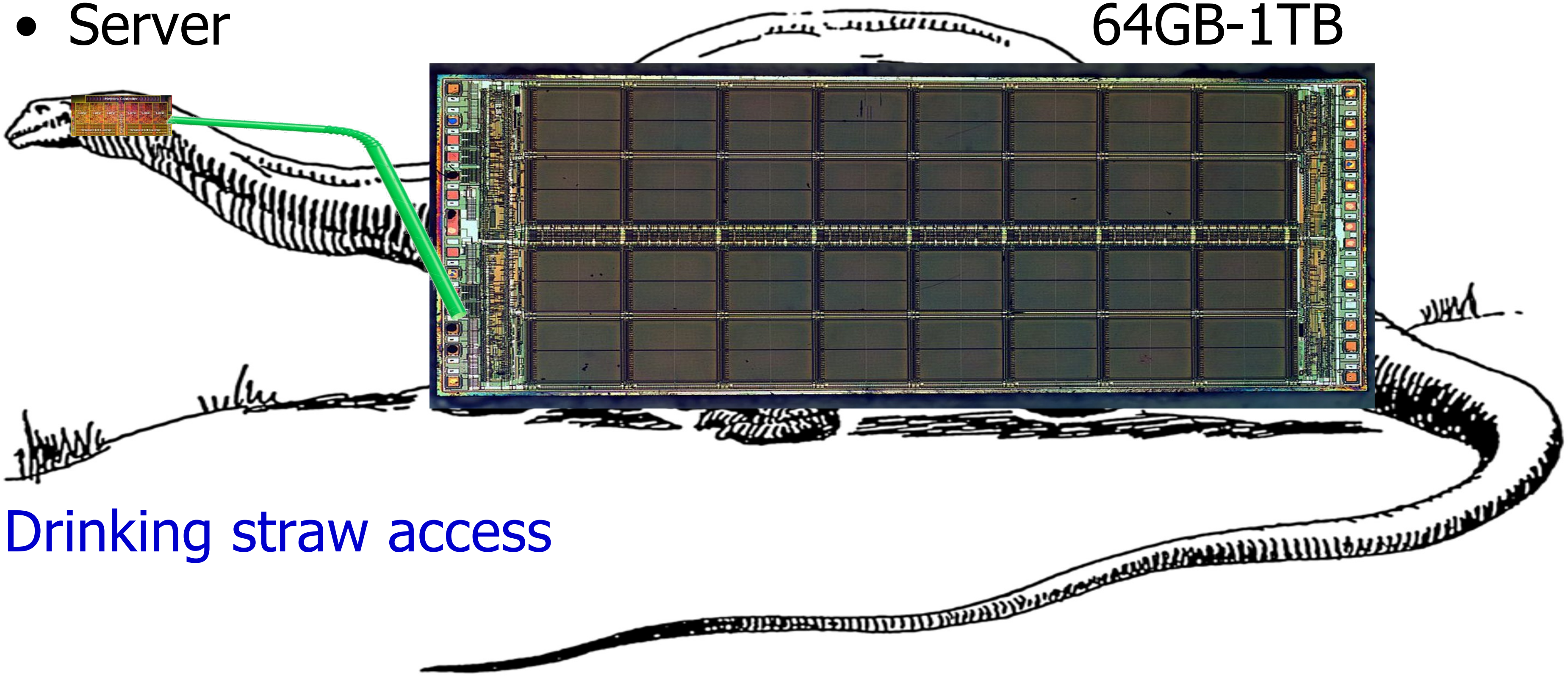
- i7: 12MB L3 cache



Little brain, **LOTS** of memory

- Server

64GB-1TB

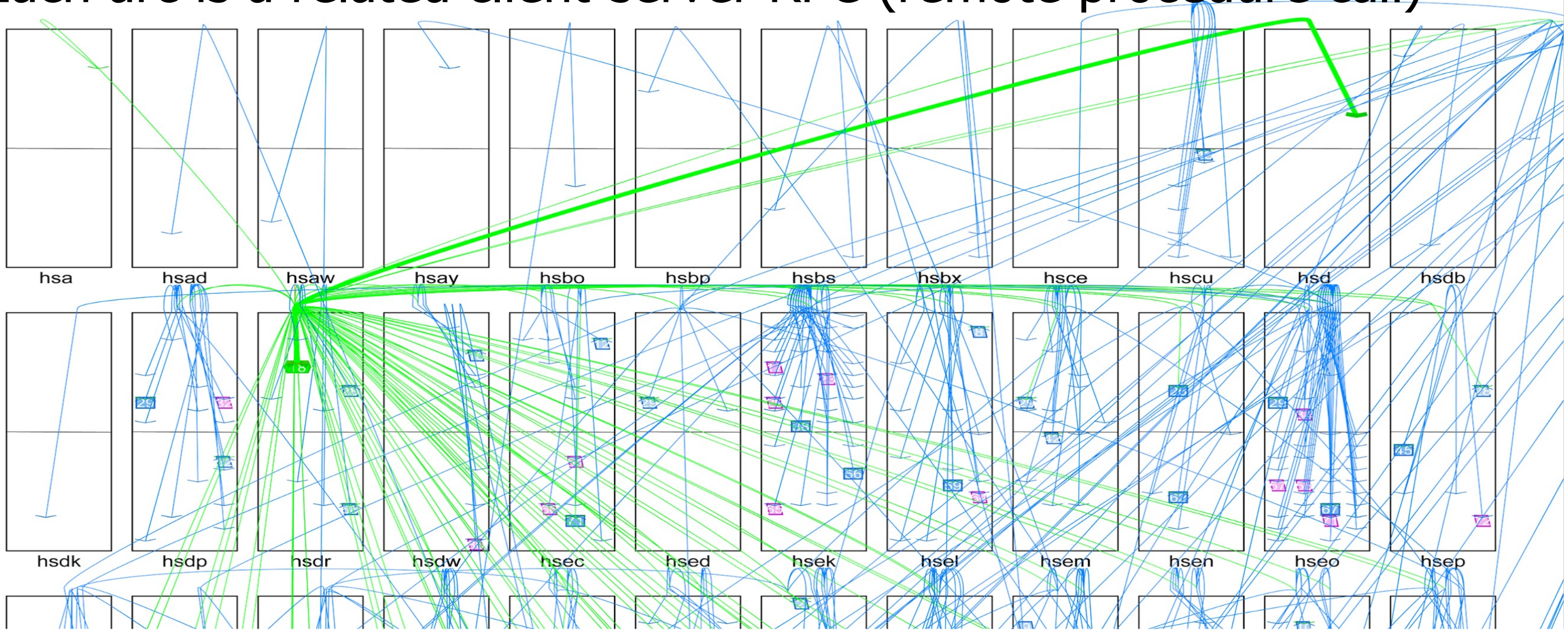


Drinking straw access

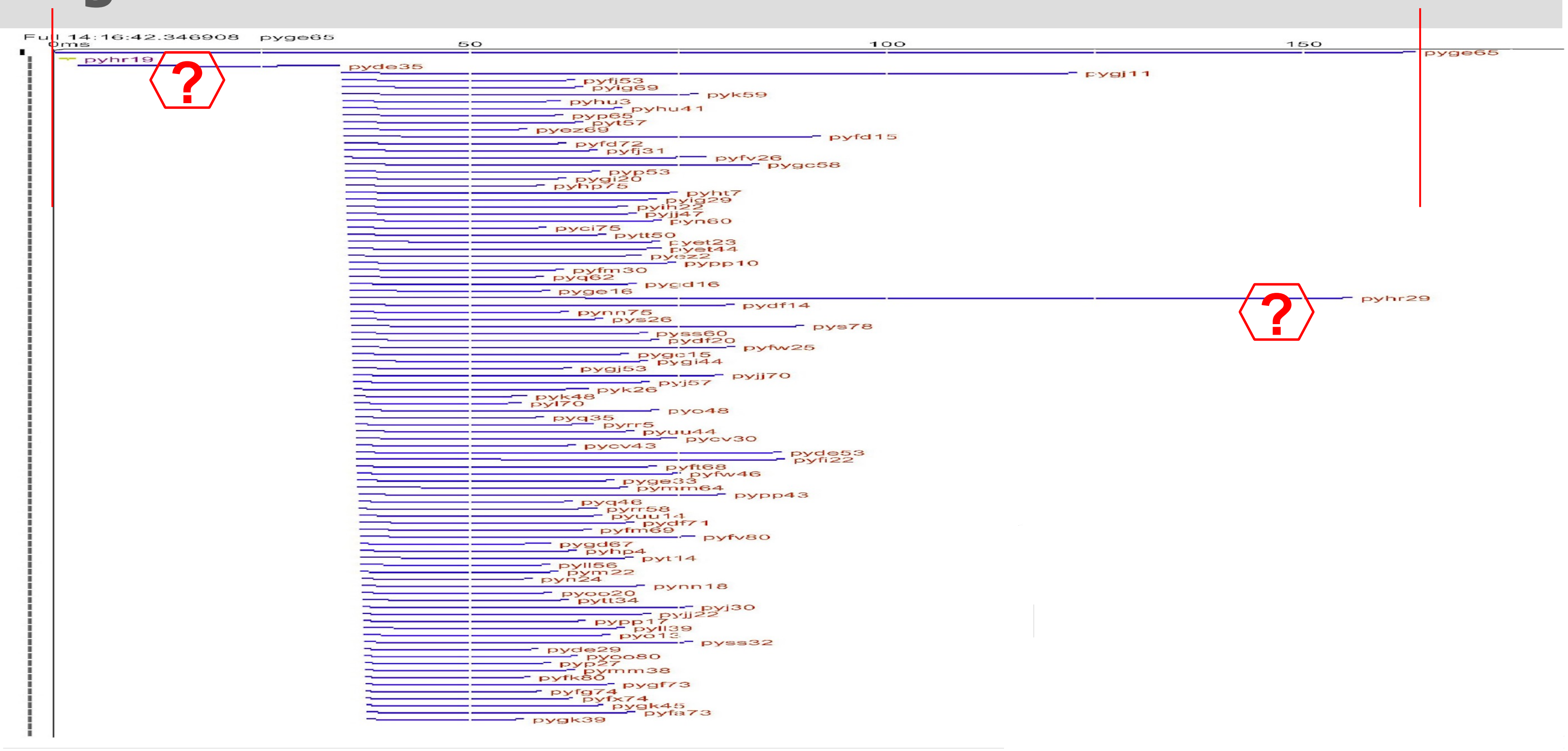
② Real-time transactions: 1000s per second

A **Single** Transaction Across ~40 Racks of ~60 Servers

- Each arc is a related client-server RPC (remote procedure call)

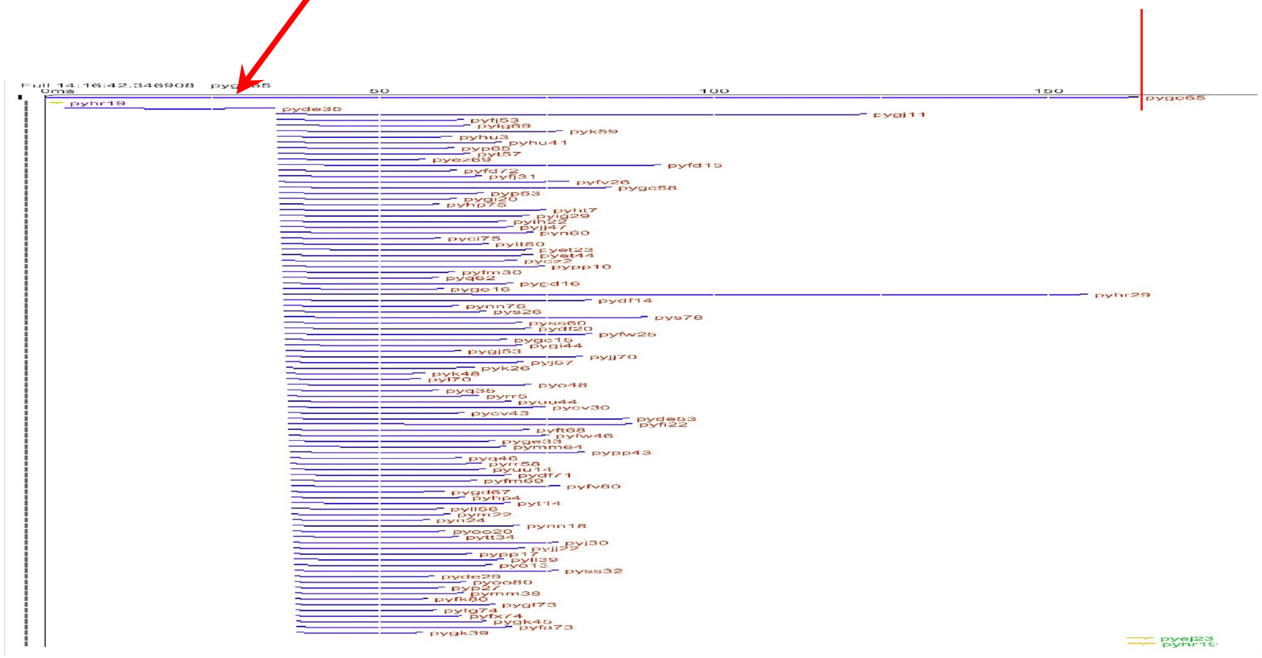


A **Single** Transaction RPC Tree: Client & 93 Servers



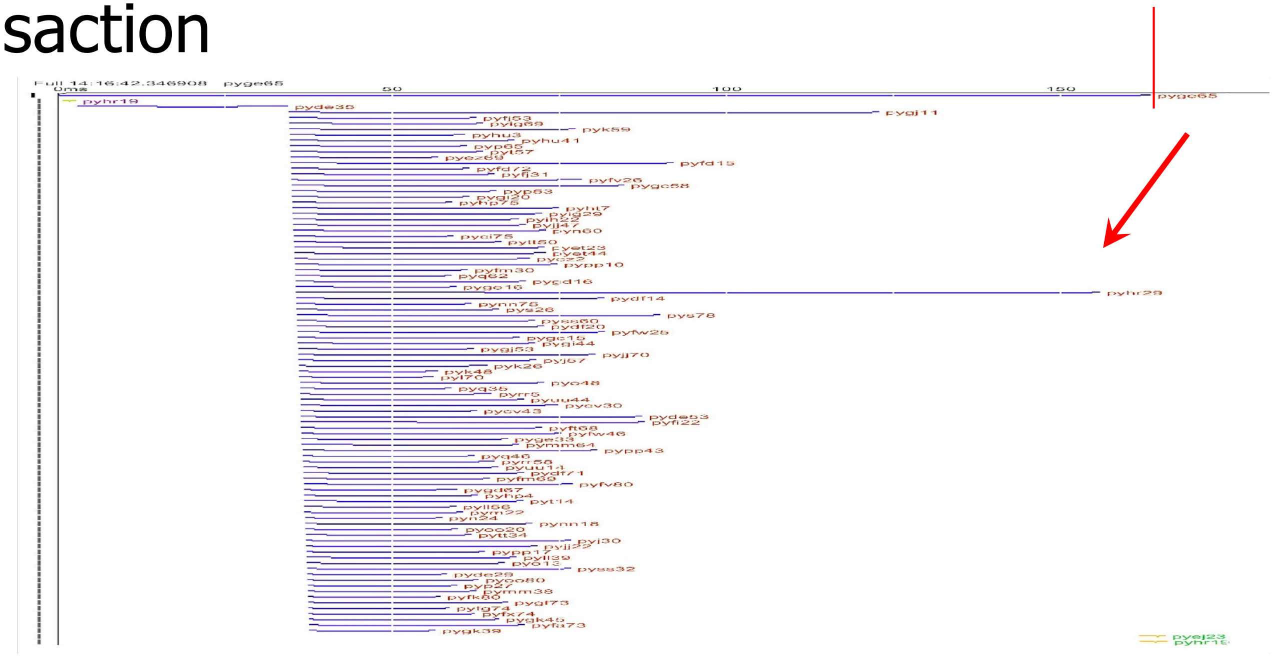
Single Transaction Tail Latency

- Canary transaction



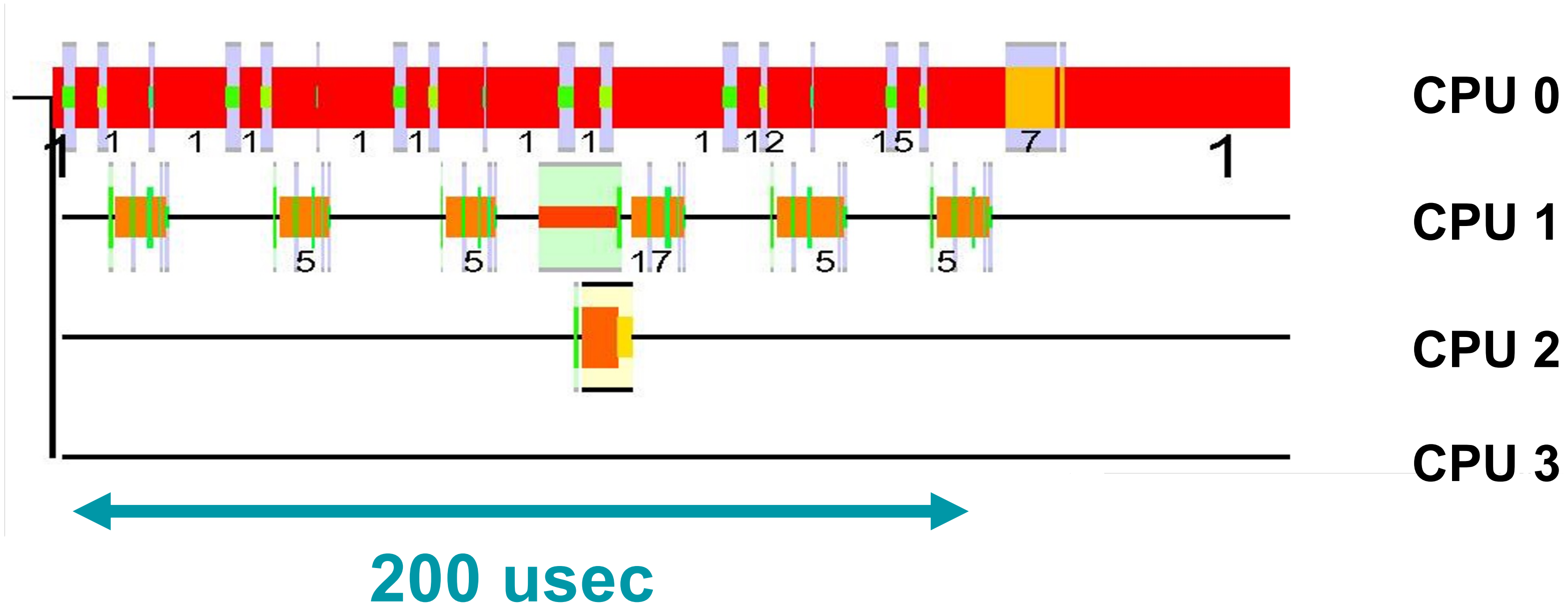
Single Transaction Tail Latency

- One slow response out of 93 parallel RPCs slows the *entire* transaction

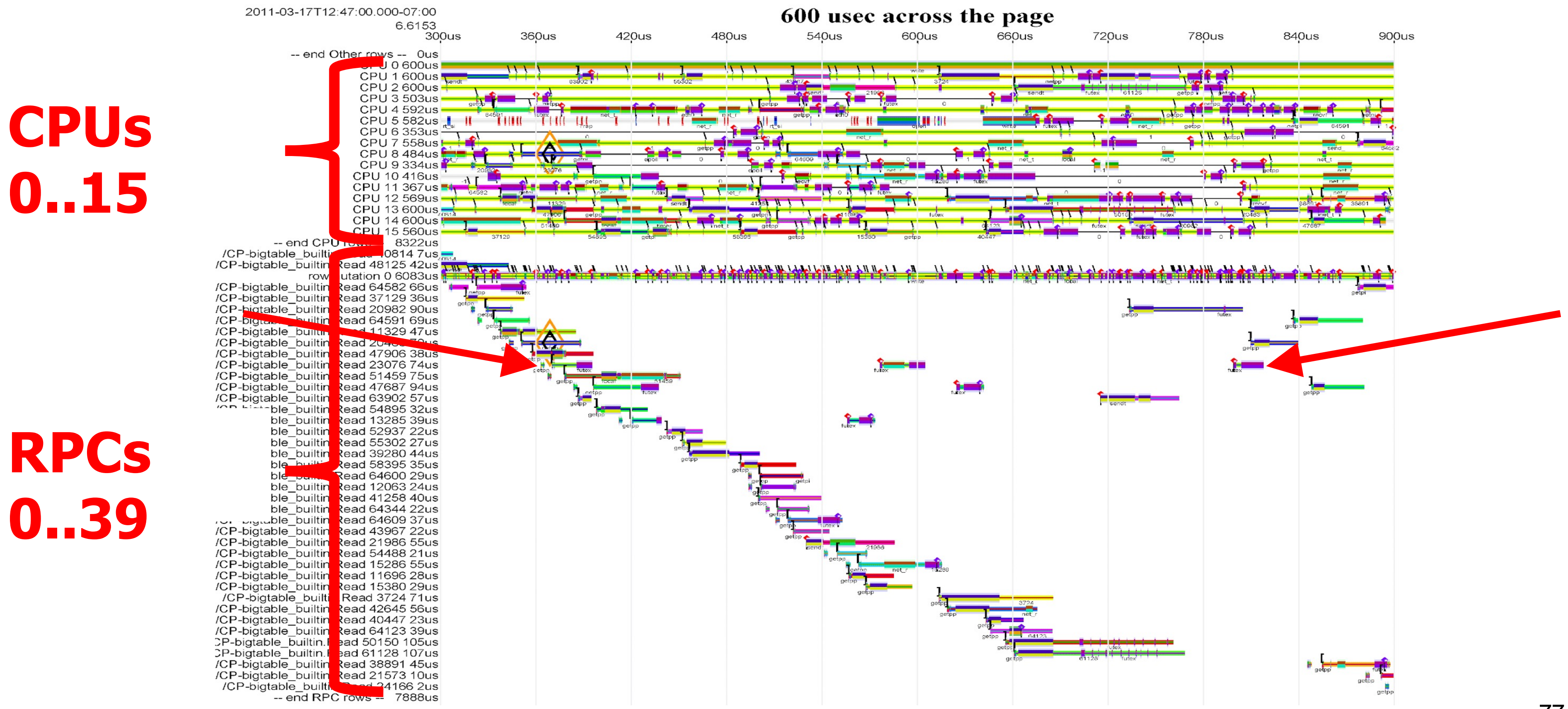


One Server, Four CPUs: User/kernel transitions

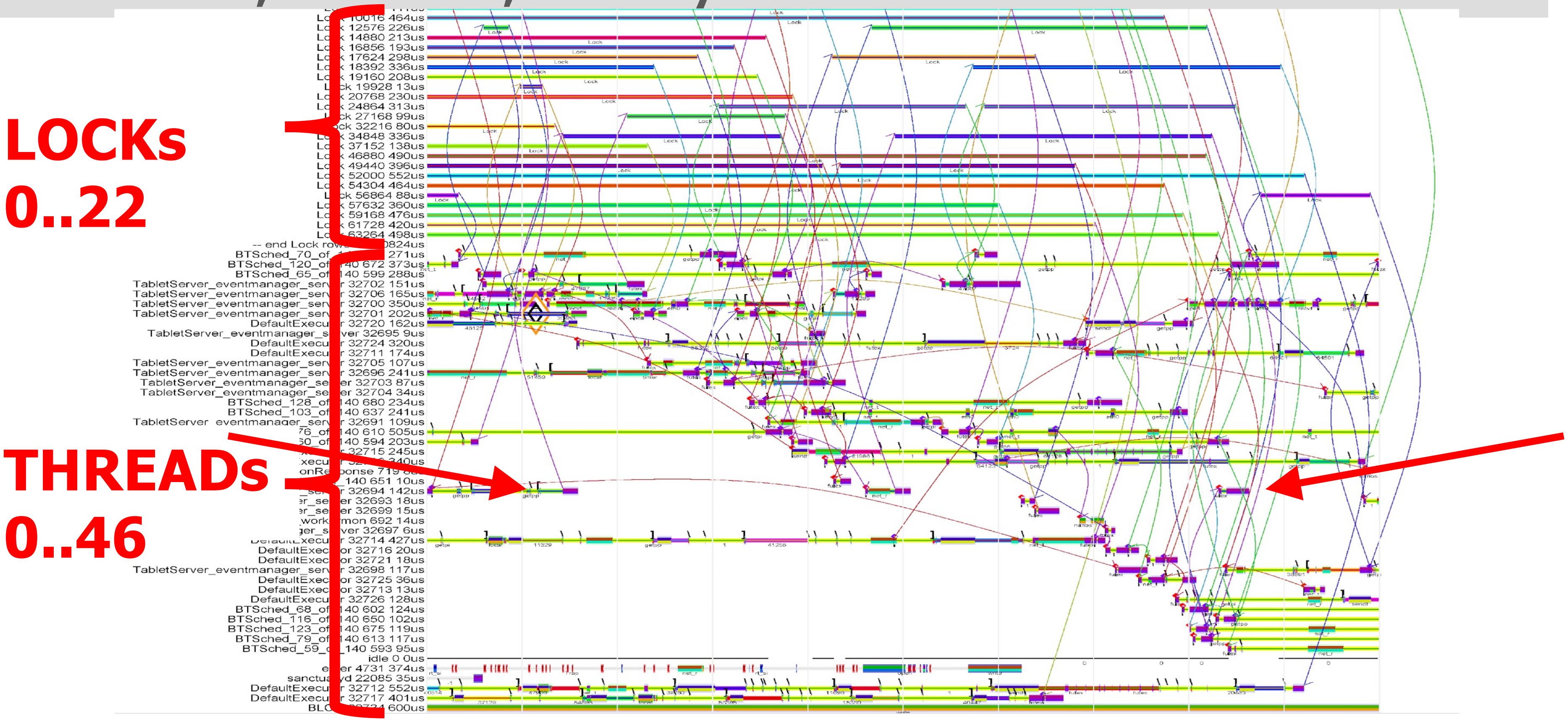
every CPU every nanosecond (Ktrace)



16 CPUs, 600us, Many RPCs



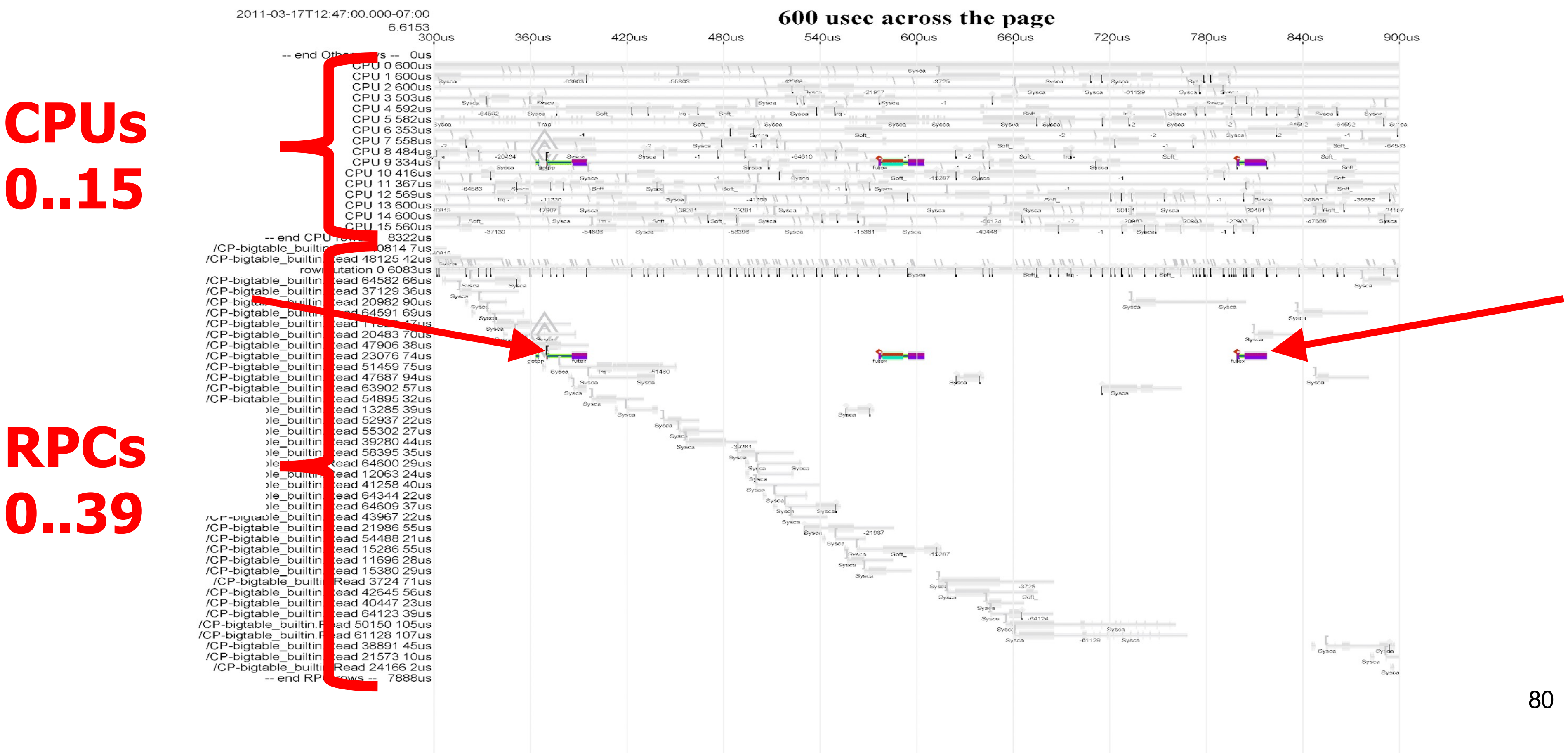
16 CPUs, 600us, Many RPCs



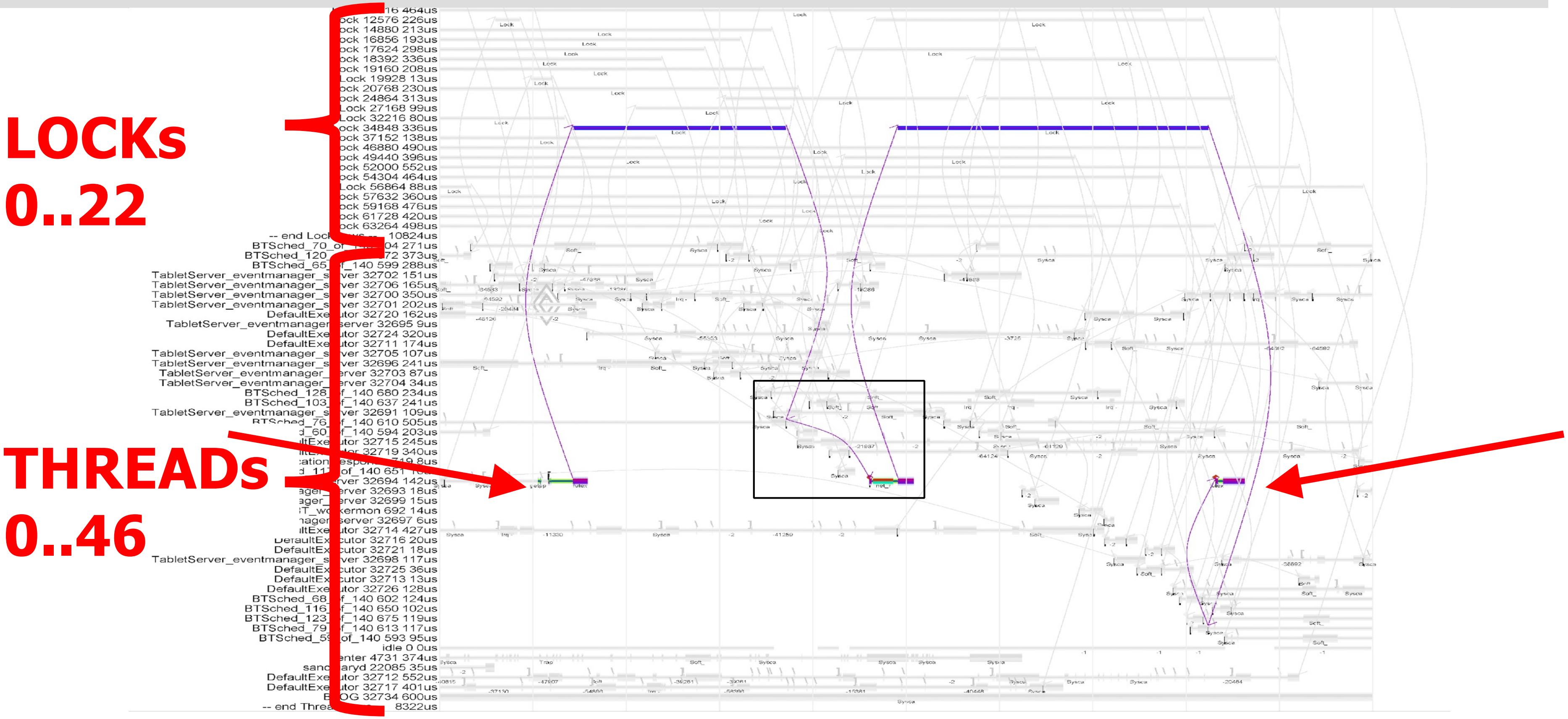
That is A LOT going on at once

- Let’s look at just *one* long-tail RPC in context

16 CPUs, 600us, one RPC



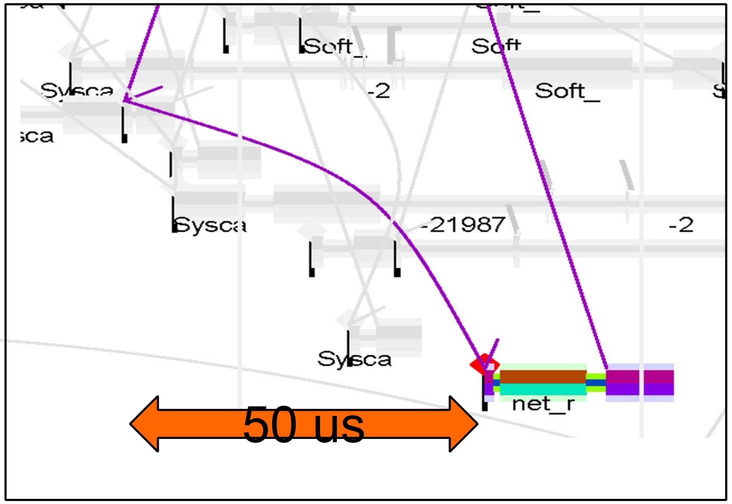
16 CPUs, 600us, one RPC



Wakeup Detail

Thread 19 frees lock, sends wakeup to waiting thread 25

Thread 25 actually runs



50us wakeup delay ??

CPU Scheduling, 2 Designs

- Re-dispatch on any idle CPU core
  - But if idle CPU core is in deep sleep, can take 75-100us to wake up
- Wait to re-dispatch on previous CPU core, to get cache hits
  - Saves nothing if could use same L1 cache
  - Saves ~10us if could use same L2 cache
  - Saves ~100us if could use same L3 cache
  - Expensive if cross-socket cache refills
  - Don't wait too long…

Real-time transactions: 1000s per second

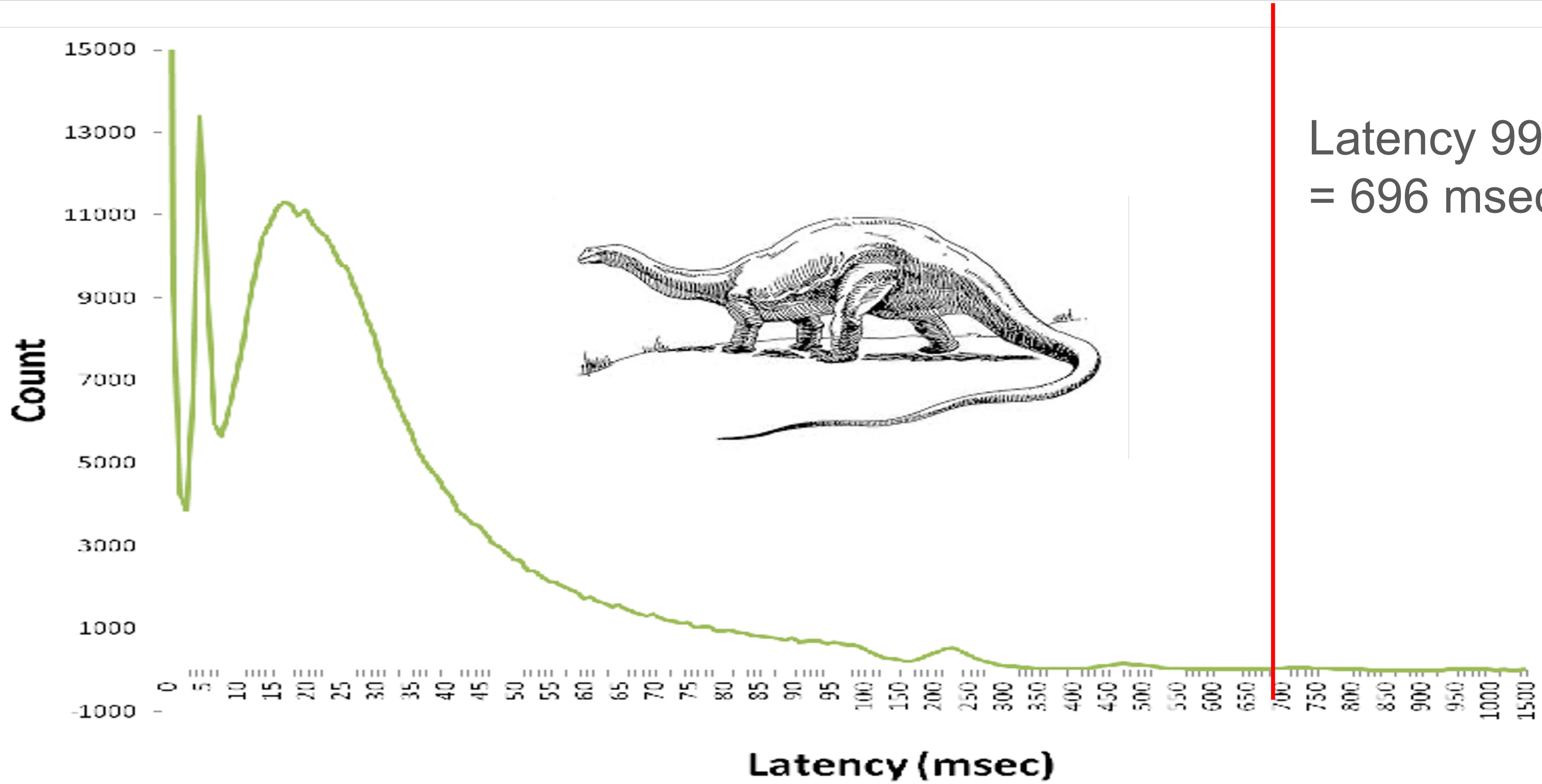
- Not your father's SPEC benchmarks
- To understand delays, need to track simultaneous transactions across servers, CPU cores, threads, queues, locks

Modern challenges in CPU design

- A single transaction can touch thousands of servers in parallel
- The slowest parallel path dominates
- Tail latency is the enemy
  - Must control lock-holding times
  - Must control scheduler delays
  - Must control interference via shared resources

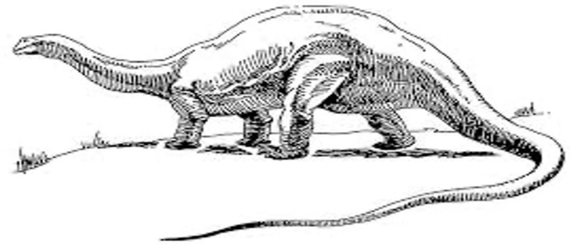
③ Isolation between programs

Histogram: Disk Server Latency; Long Tail



Latency 99th %ile
= 696 msec

Non-repeatable Tail Latency Comes from Unknown Interference



Isolation of programs reduces tail latency.

Reduced tail latency = higher utilization.

Higher utilization = $$$.

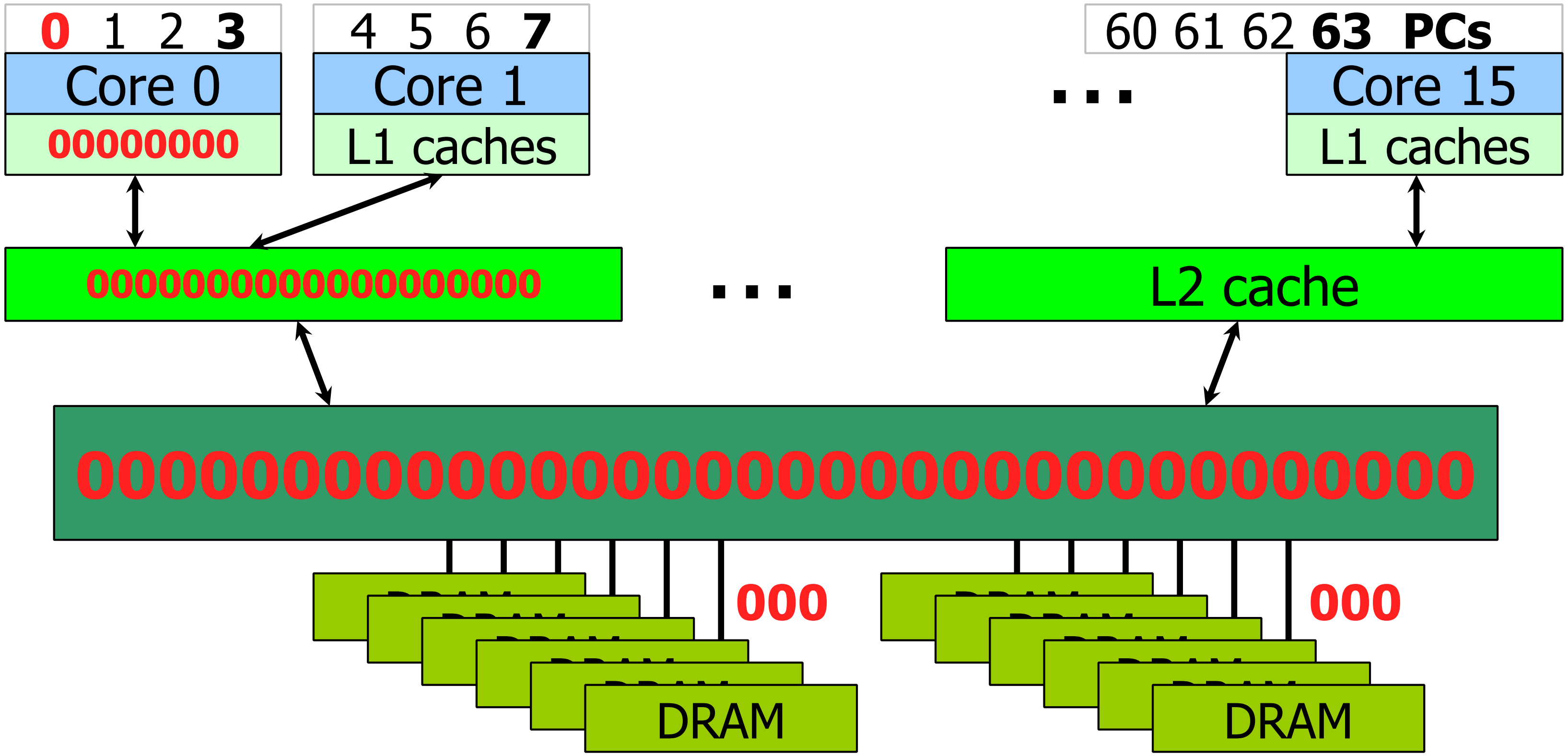
Many Sources of Interference

- Most interference comes from software
- But a bit from the hardware underpinnings

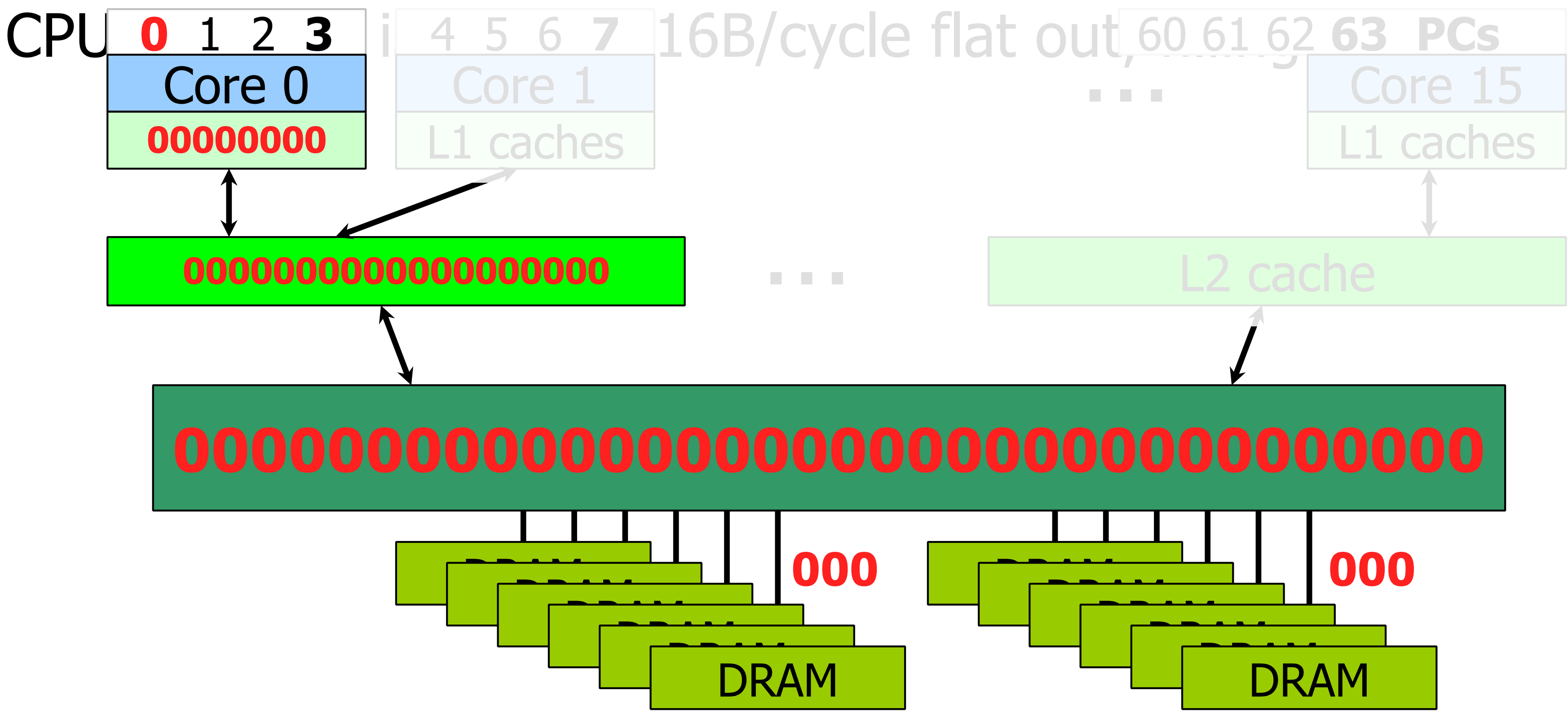
- In a shared apartment building, most interference comes from jerky neighbors
- But thin walls and bad kitchen venting can be the hardware underpinnings

Isolation issue: Cache Interference

CPU thread 0 is moving 16B/cycle flat out, filling caches

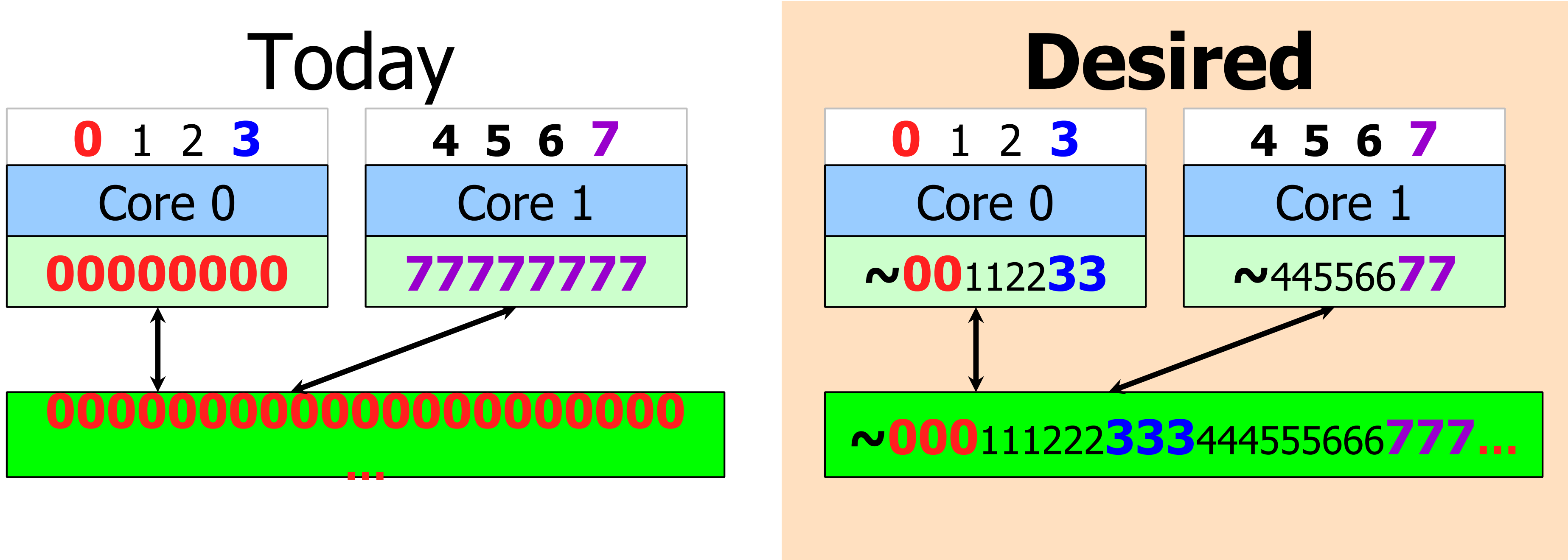


Isolation issue: Cache Interference



Isolation issue: Cache Interference

- CPU thread 0 is moving 16B/cycle flat out



Isolation between programs

- Good fences make good neighbors
- We need better hardware support for program isolation in shared memory systems

Modern challenges in CPU design

- Isolating programs from each other on a shared server is hard
- As an industry, we do it poorly
  - Shared CPU scheduling
  - Shared caches
  - Shared network links
  - Shared disks
- More hardware support needed
- More innovation needed

④ Measurement underpinnings

Differences from desktop software

Tens of thousands of user-facing transactions per second

Distributed computation across thousands of servers

The important metric is response time, i.e. latency

Excessive tail latency is the most important performance problem

As an industry, we have poor tools for observing and understanding tail latency

Tail latency

Some transactions, total latency for each one

We seek to understand why the one is slow

Easy case

A particular transaction is slow every time it is run

It is straightforward to run the transaction repeatedly offline on a few load-test servers

Existing profiling tools, disk byte counts, network byte counts, etc. will reveal where the time goes

"Interesting" case

A particular transaction is usually fast, but occasionally quite slow

It is only slow under live load during the busiest hour of the day

Running it again it runs fast

Until the *reason* for slowness is found, we cannot reproduce the problem offline

Existing tools are unable to reveal where the time goes

"Interesting" case -- why it matters

Until the *reason* for slowness is found, we cannot reproduce the problem offline

At 10,000 transactions/second and no call tree, the 99th percentile slow cases happen 100 times per second

But, for software with 100:1 fanout transaction call trees, *almost everything runs at the 99th percentile slow rate*

Existing tools are unable to reveal where the time goes

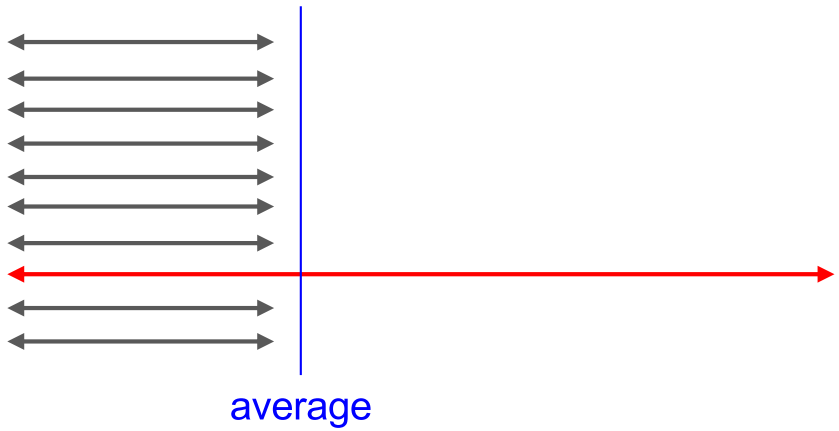
Tail latency

Some transactions, total latency

We seek to understand why the one is slow

Tail latency

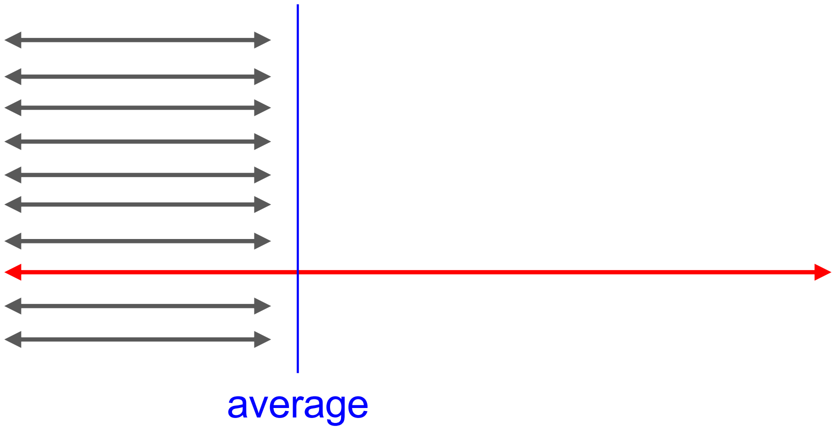
Possible tool: average latency



Tells us **nothing** about the slow one

Tail latency

Possible tool: average latency

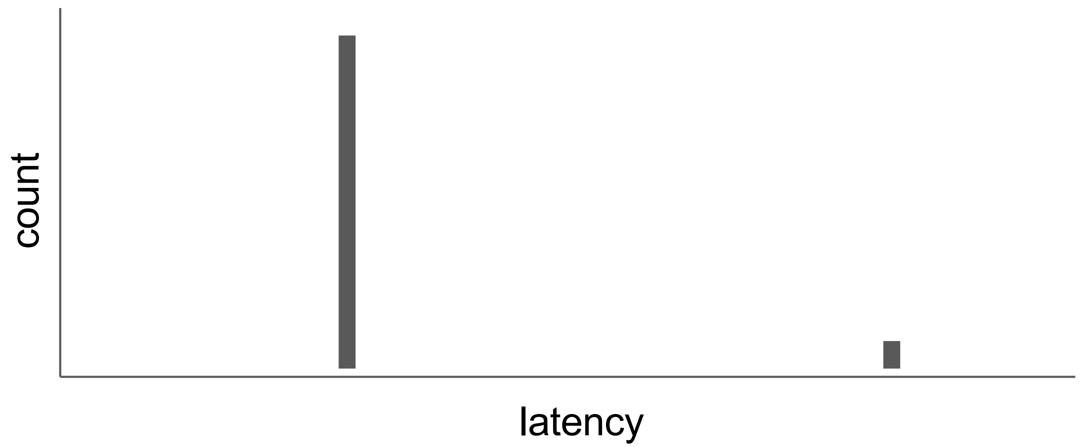


Tells us **nothing** about the slow one

Average is the wrong tool for understanding variance

Tail latency

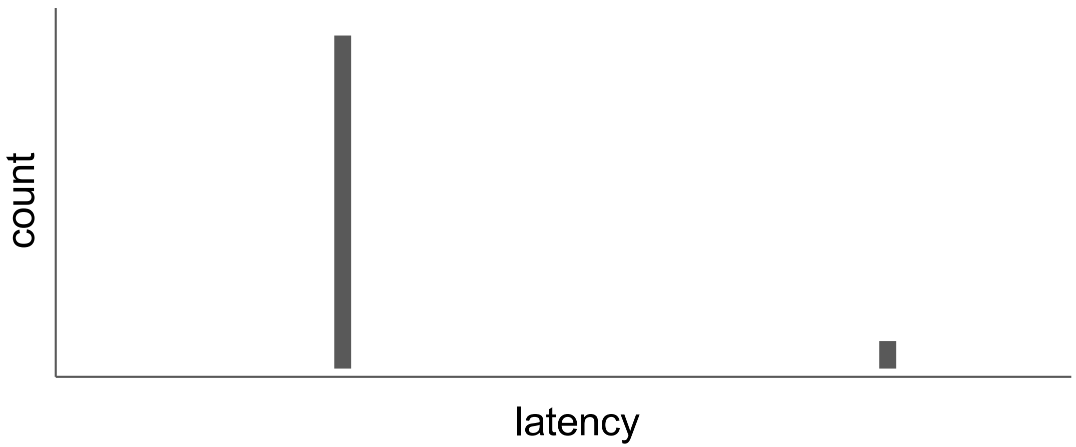
Possible tool: latency histogram



Tells us **what** we have but not **why**

Tail latency

Possible tool: latency histogram



Tells us **what** we have but not **why**

Histogram is the wrong tool for understanding variance

Tail latency

Possible tool: Profile of CPU time per source function

gather_inputs()	
process_item()	
process_more()	
calculate()	
produce_output()	

But where is the slow transaction?

Tail latency

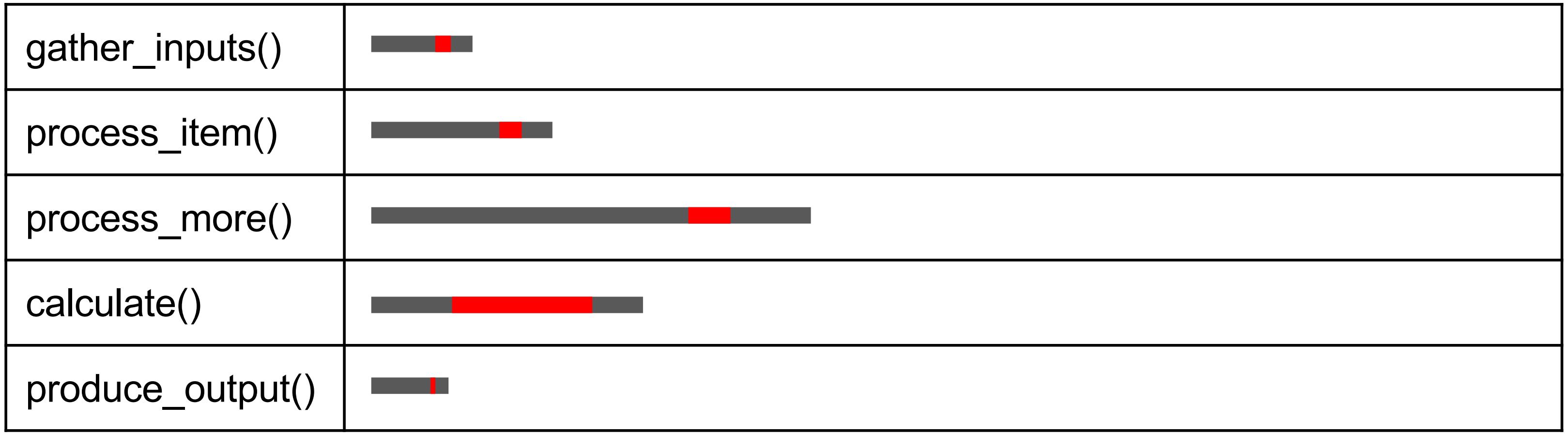
Possible tool: Profile of CPU time per source function

But where is the slow transaction?

Profiling is the wrong tool for understanding variance

Profiling: the wrong tool for understanding variance

Possible tool: Profile of CPU time per source function



It merges together many normal transactions with few slow ones, hiding the 1% signal in 99% noise

Tail latency

Possible tool: Profile of CPU time per source function

gather_inputs()	
process_item()	
process_more()	
calculate()	
produce_output()	

But where is the slow transaction?

CPU Profiling is the wrong tool for a second reason ...

CPU Profiling: the wrong tool for a second reason

Some transactions, total time

The delay may not be using CPU time at all; it may be *waiting* for something. CPU profiling is *blind* to non-CPU wait time

Tail latency

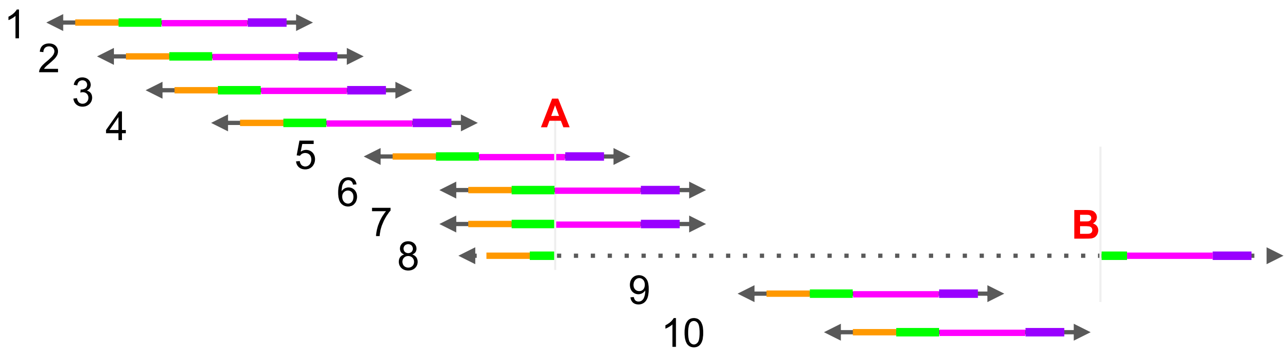
Possible tool: **trace** events for each transaction

Now we can see **what** is different about the slow one.

But we don't know **why** it is different

Tail latency

Possible tool: trace events for each transaction, lined up in time

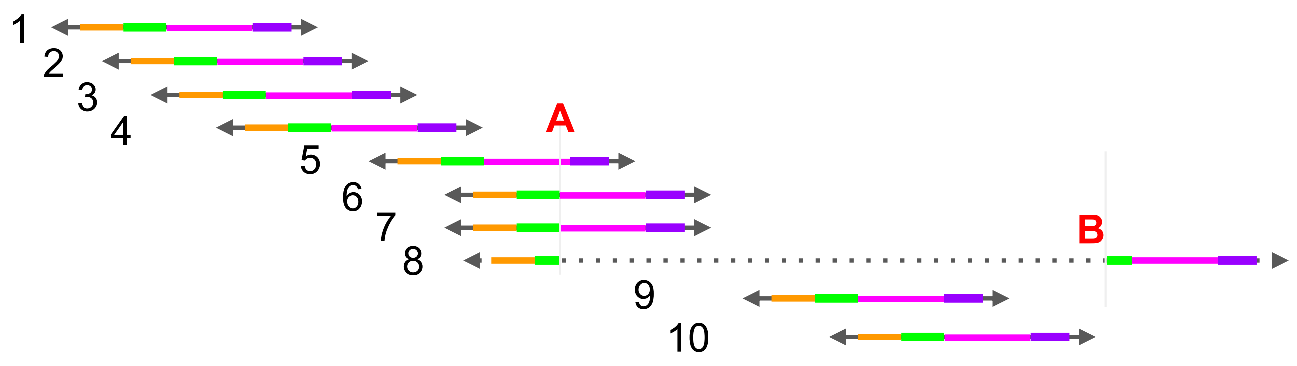


Now we can see **what** is different about the slow one.

And the time alignment of 6's end-of-green (A) with 8's green blocking until B.

Tail latency

Possible tool: trace events for each transaction, lined up in time



More detailed events at A or B will reveal the reason for starting or stopping blocking. (Possibly a race condition or a contended software lock.)

Tail latency, summary

Average is the wrong tool for understanding variance

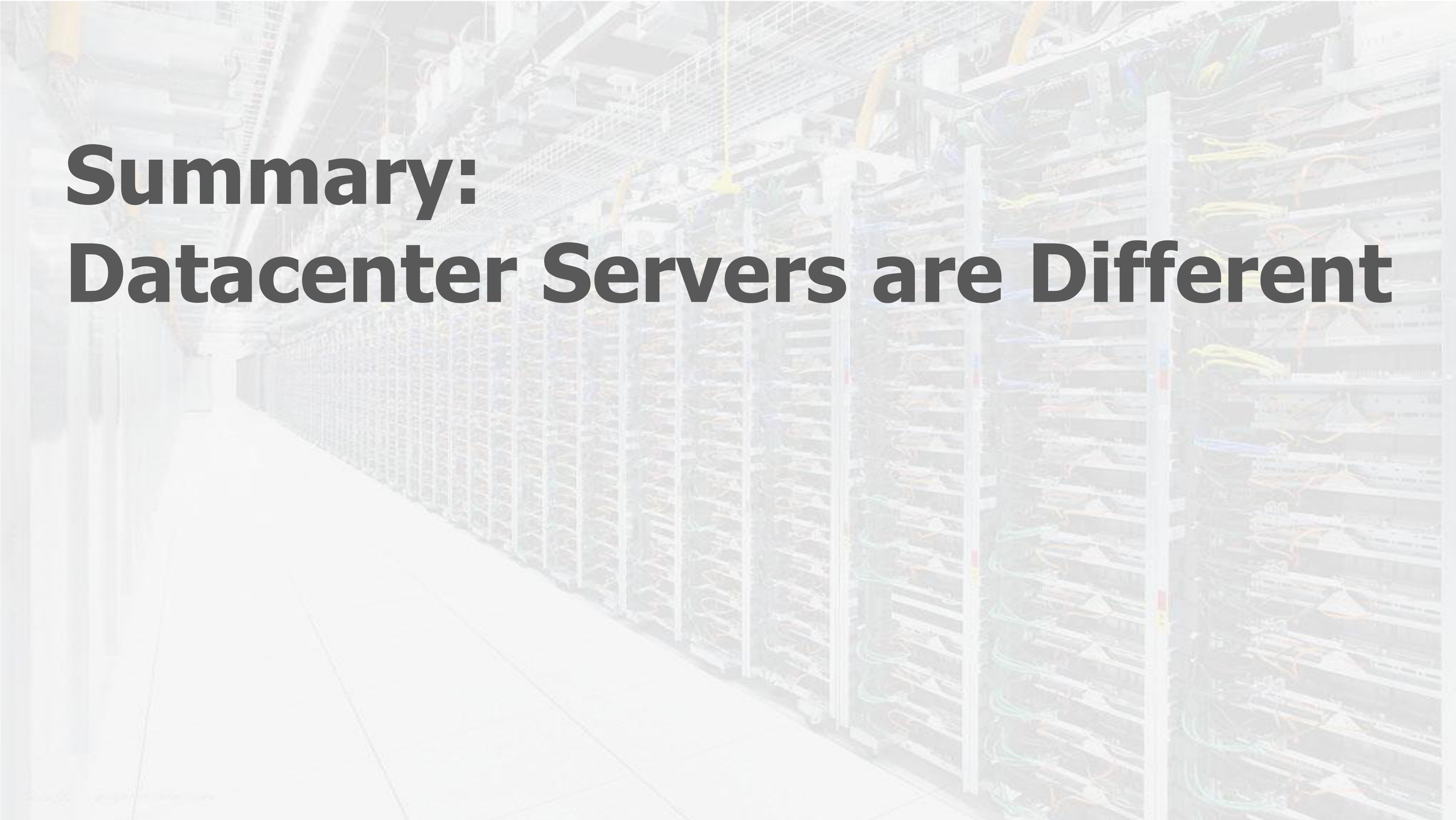
Histogram is the wrong tool for understanding variance

Profile is the wrong tool for understanding variance

Event tracing is a good tool for understanding variance

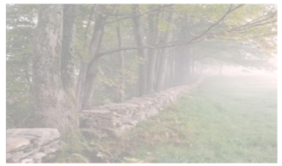
Event tracing lined up in time is a great tool for understanding variance -- it can show causes of delay directly

Summary:
Datacenter Servers are Different



Datacenter Servers are Different






① Move data: big and small

② Real-time transactions: 1000s per second

③ Isolation between programs

④ Measurement underpinnings

Thank You, Questions?