CS 152 Computer Architecture and Engineering

Lecture 20: Datacenters

Dr. George Michelogiannakis
EECS, University of California at Berkeley
CRD, Lawrence Berkeley National Laboratory

http://inst.eecs.berkeley.edu/~cs152

Thanks to Christina Delimitrou, Christos Kozyrakis

Administrivia

- PS 5 due NOW
- Quiz 5 on Wednesday next week
- Please show up on Monday April 21st (last lecture)
  - Neuromorphic, quantum
  - Parting thoughts that have nothing to do with architecture
  - Class evaluation

What Is A Datacenter?

- The compute infrastructure for internet-scale services & cloud computing
  - x10k of servers, x100k hard disks
  - Examples: Google, Facebook, Microsoft, Amazon (+ Amazon Web Services), Twitter, Yahoo, …
- Both consumer and enterprise services
  - Windows Live, Gmail, Hotmail, Dropbox, bing, Google, Adcenter, GoogleApps, Web apps, Exchange online, salesforce.com, Azure, …

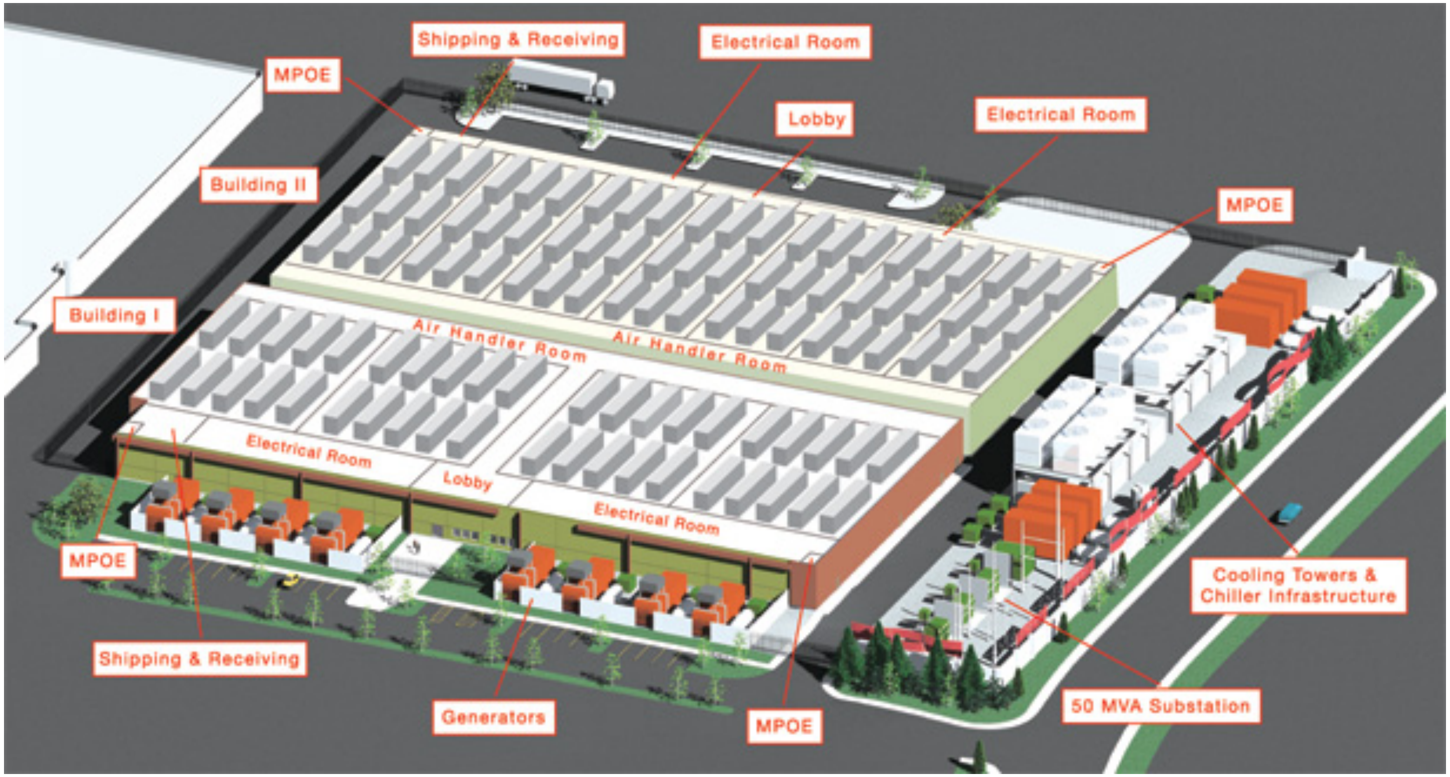
Other Definitions

- Centralized repository for the storage, management, and dissemination of data and information, pertaining to a particular business or service

- Datacenters involve large quantities of data and their processing

- Largely made up of commodity components

Components



- Apart from computers & network switches, you need:
  – Power infrastructure: voltage converters and regulators, generators and UPSs, …
  – Cooling infrastructure: A/C, cooling towers, heat exchangers, air impellers,…
- Everything is co-designed!

Example: MS Quincy Datacenter

- 470k sq feet (10 football fields)
- Next to a hydro-electric generation plant
  - At up to 40 MegaWatts, $0.02/kWh is better than $0.15/kWh ☺
  - That's equal to the power consumption of 30,000 homes

Example: MS Chicago Datacenter

[K. Vaid, Microsoft Global Foundation Services, 2010]

Google's Datacenter Locations

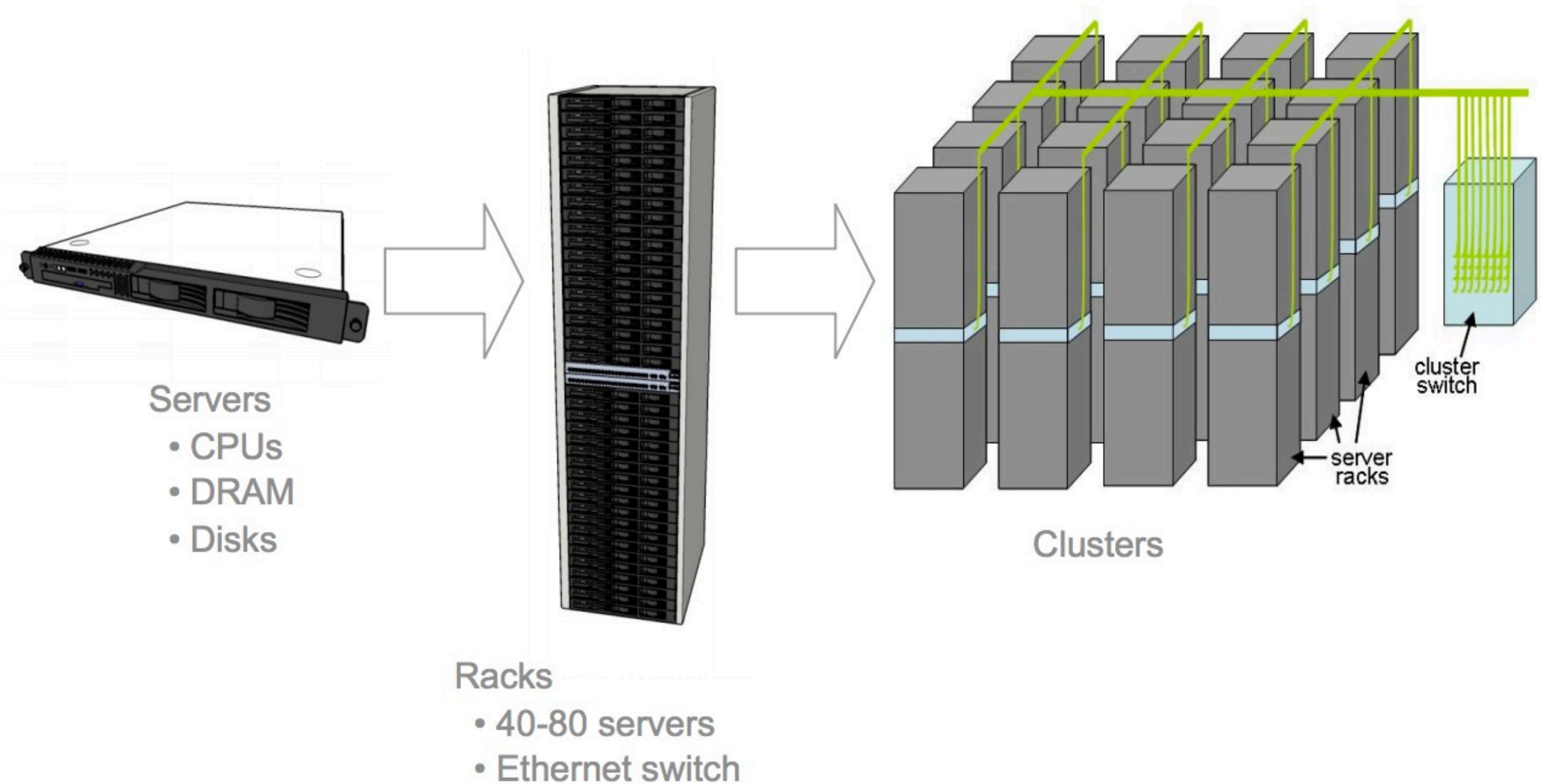
Applications That Use Datacenters

- Storage
  - Large and small files (e.g., phone contacts)
- Search engines
- Compute time rental & web hosting
  - Amazon EC2 – virtual server hosting
- Cloud gaming
  - File or video streaming

Why Is Cloud Gaming Possible?

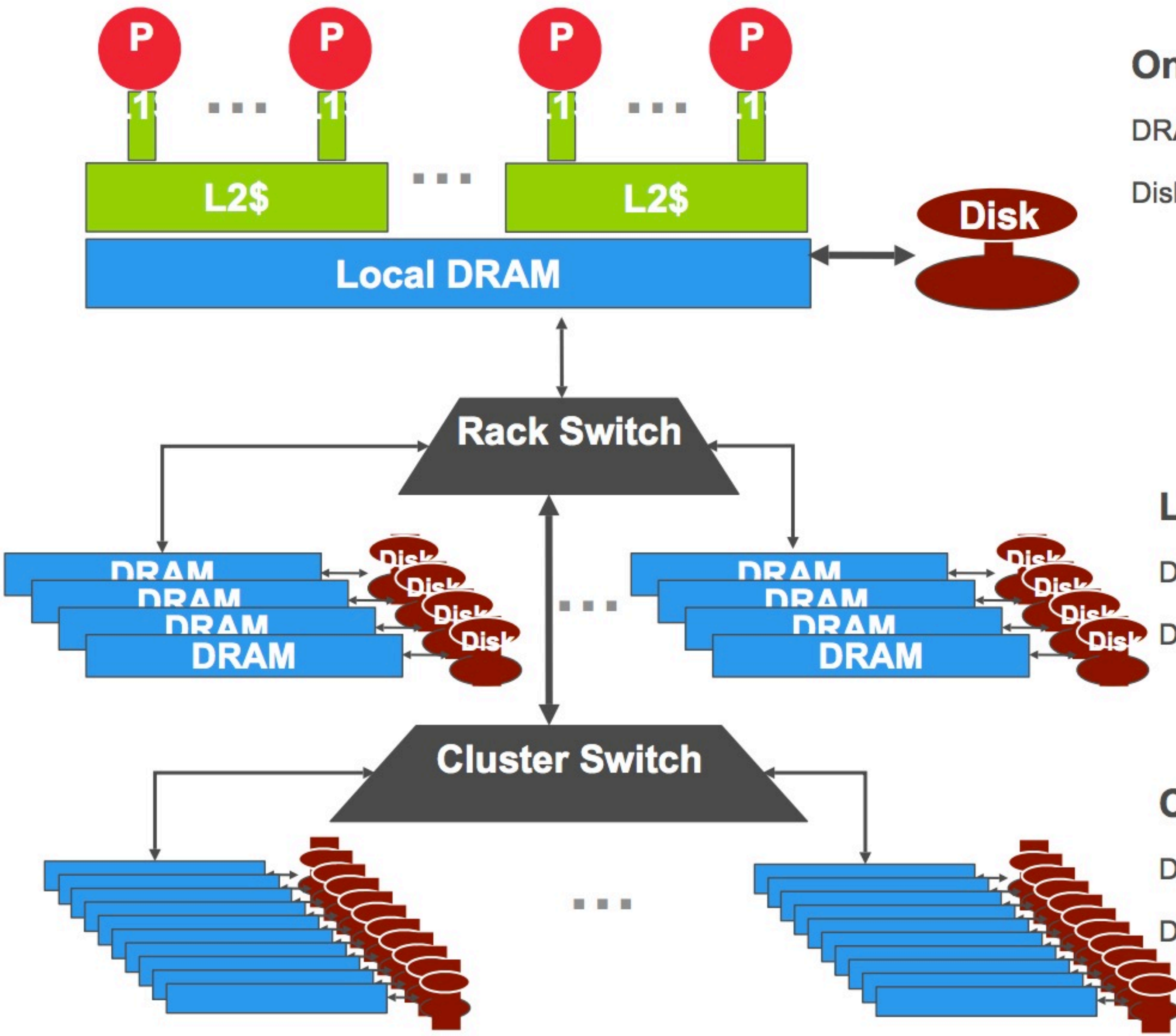
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

The Inside


Servers
• CPUs
• DRAM
• Disks
Racks
• 40-80 servers
• Ethernet switch
Clusters
cluster switch
server racks

Example: FB Datacenter Racks

Storage Hierarchy



One server

DRAM: 16GB, 100ns, 20GB/s

Disk: 2TB, 10ms, 200MB/s

Local rack (80 servers)

DRAM: 1TB, 300us, 100MB/s

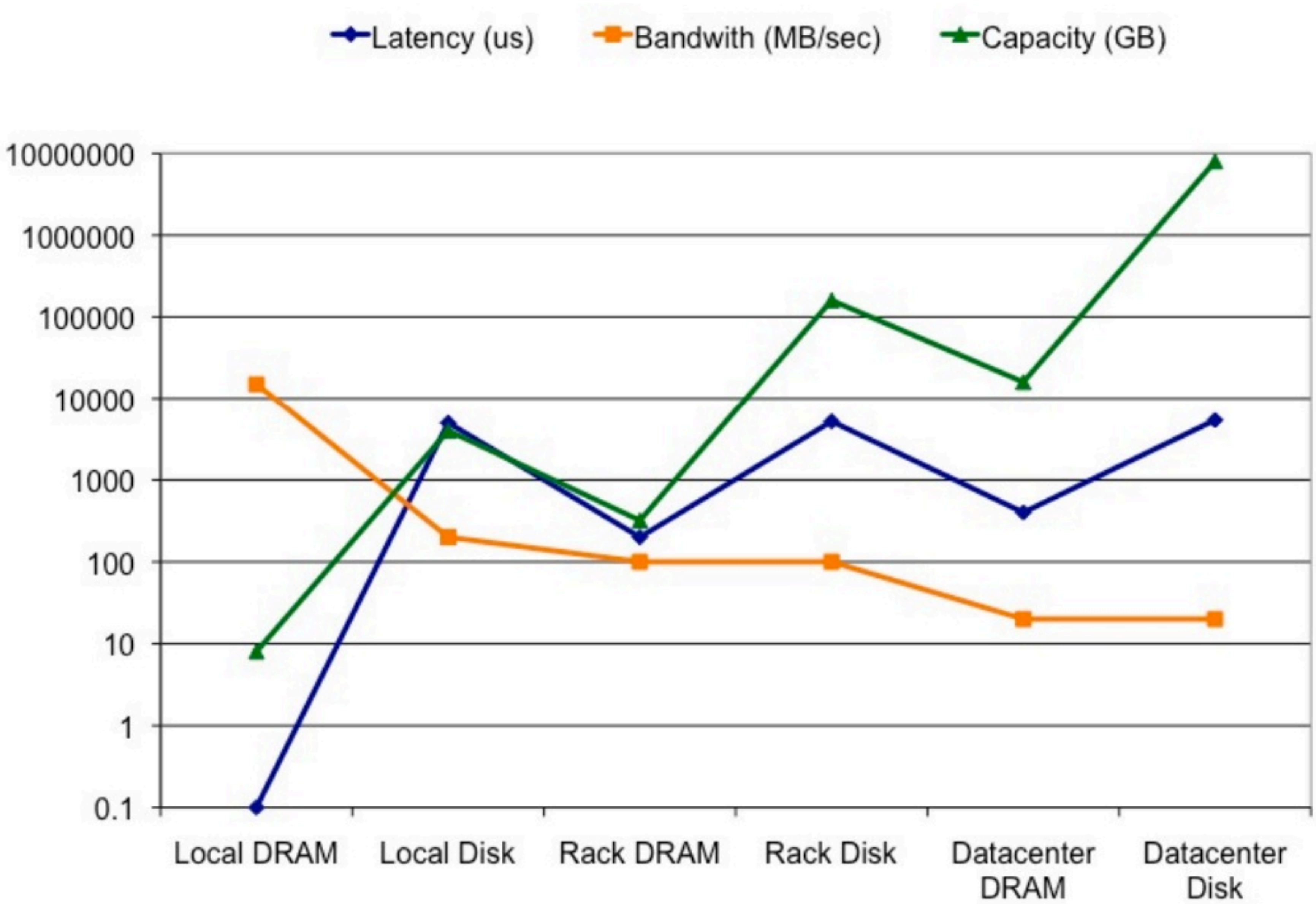
Disk: 160TB, 11ms, 100MB/s

Cluster (30+ racks)

DRAM: 30TB, 500us, 10MB/s

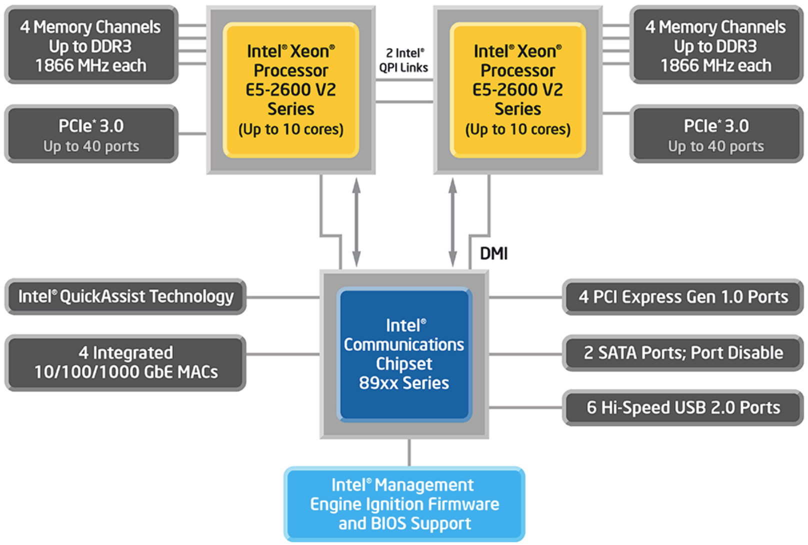
Disk: 4.80PB, 12ms, 10MB/s

Storage Hierarchy

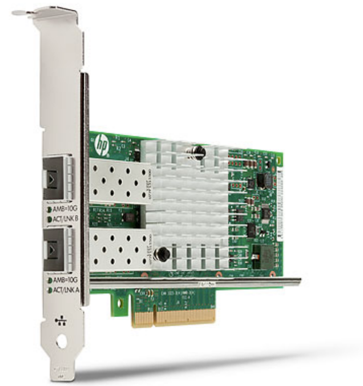


Commodity Hardware

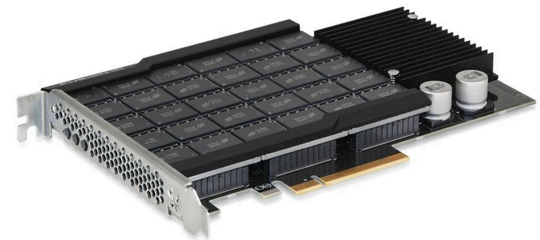
2-socket server



10GbE NIC



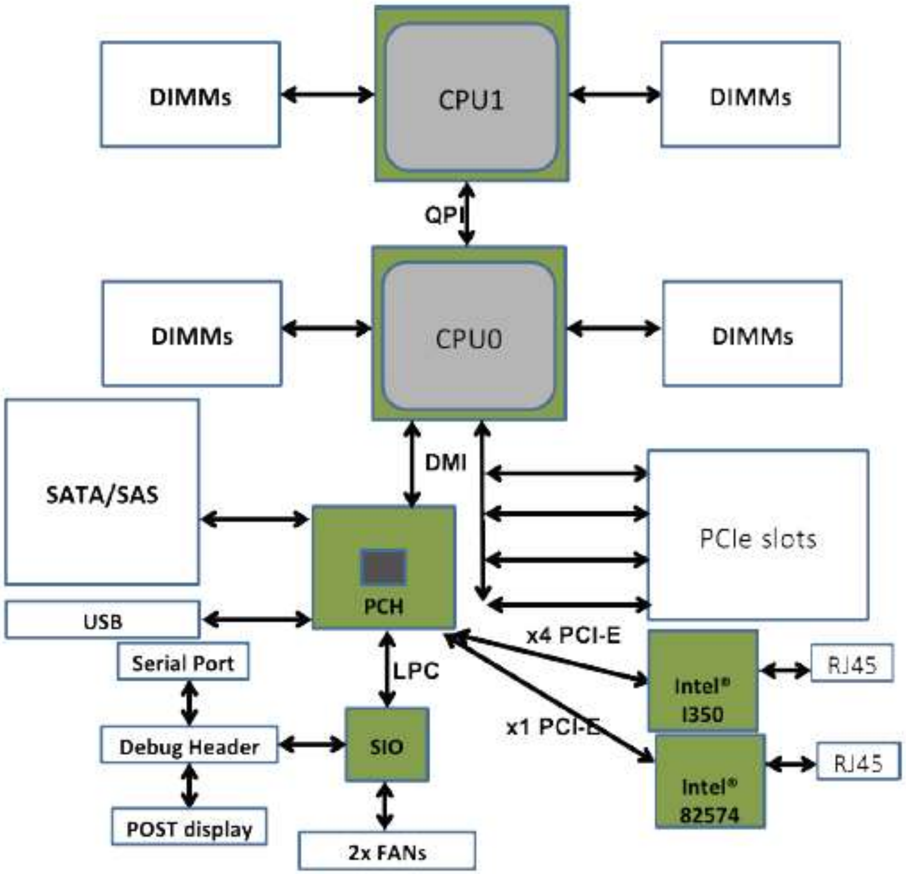
Flash Storage



JBOD disk array

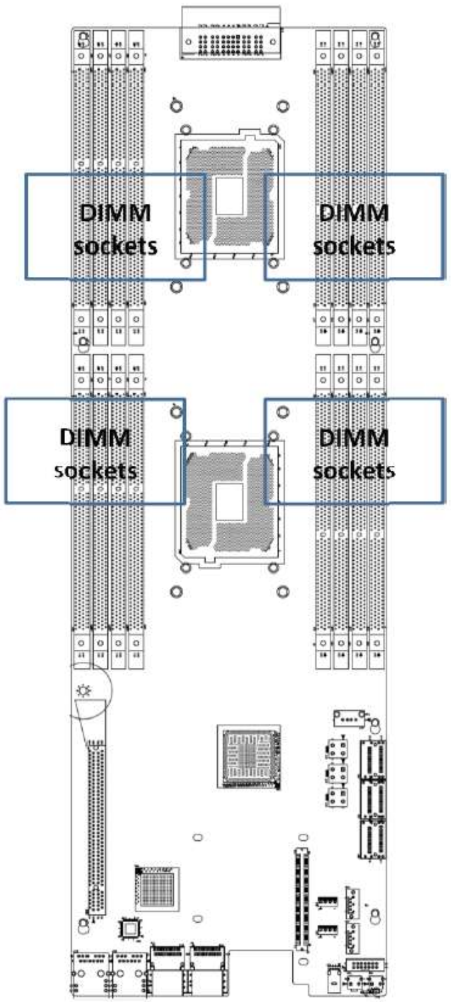
Low-latency 10GbE Switch

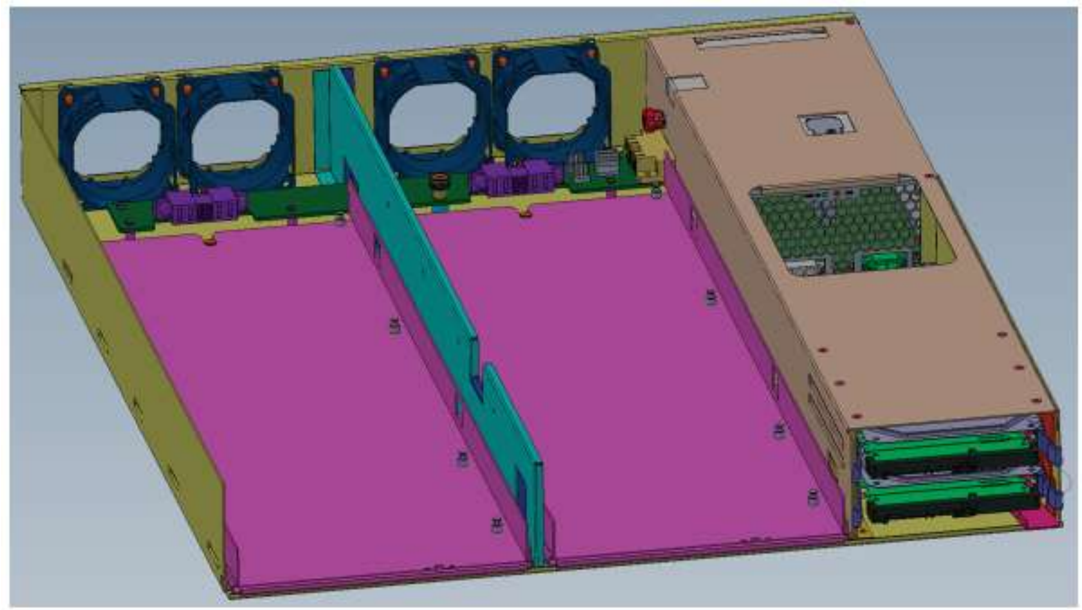
Basic Unit: 2-socket Server



- 1-2 multi-core chips
- 8-16 DRAM DIMMS
- 1-2 ethernet ports
  - 10Gbps or higher
- Storage
  - Internal SATA/SAS disks (2-6)
  - External storage expansion
- Configuration/size vary
  - Depending on tier role
  - 1U - 2U (1U = 1.67 inches)

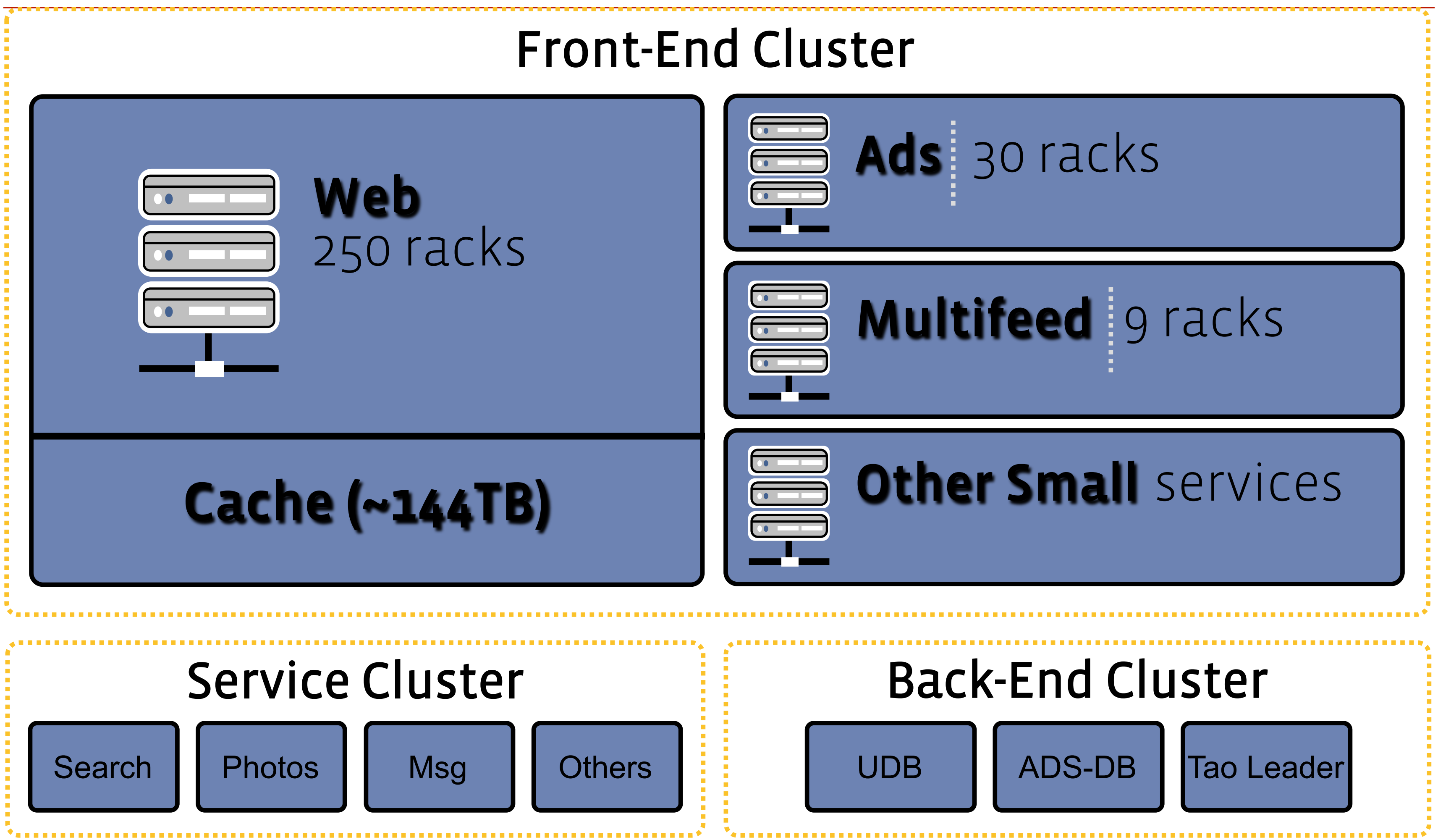
Example: FB 2-socket Server



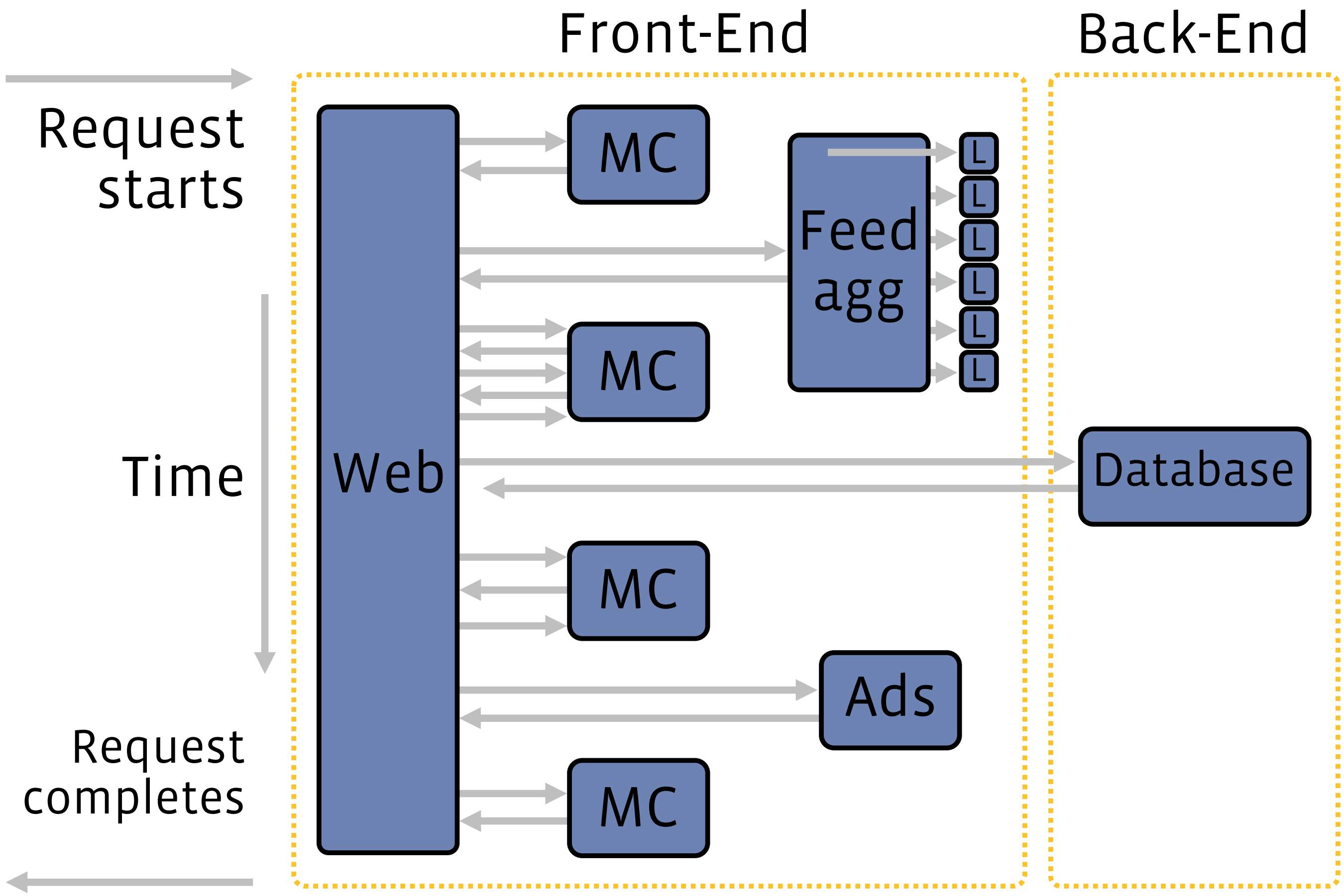


- Characteristics
  - Upgradable CPUs & memory, boot on LAN, external PCIe link, feature reduced
  - Similar design for AMD servers (why?)

Application Mapping (FB Example)



Servers Used for a FB Request

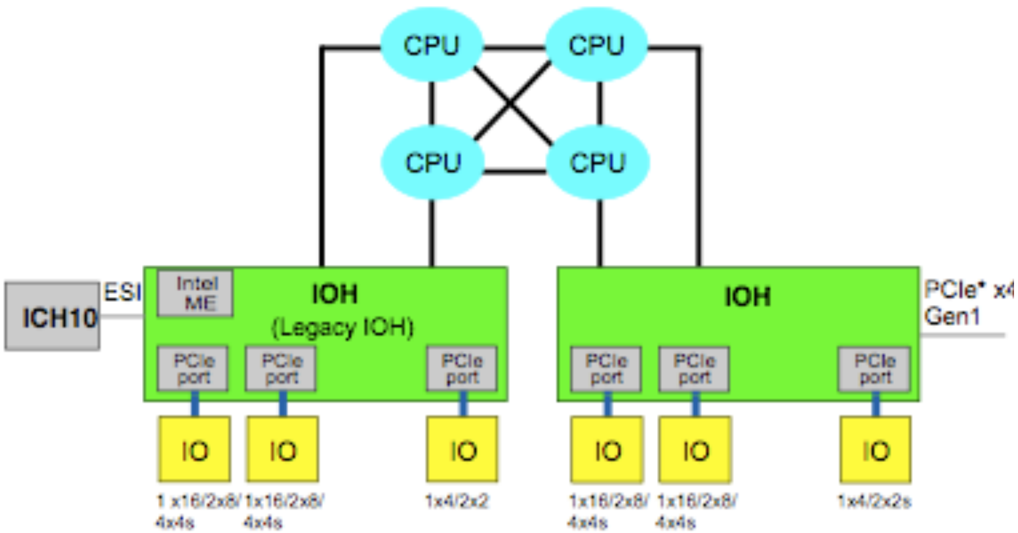


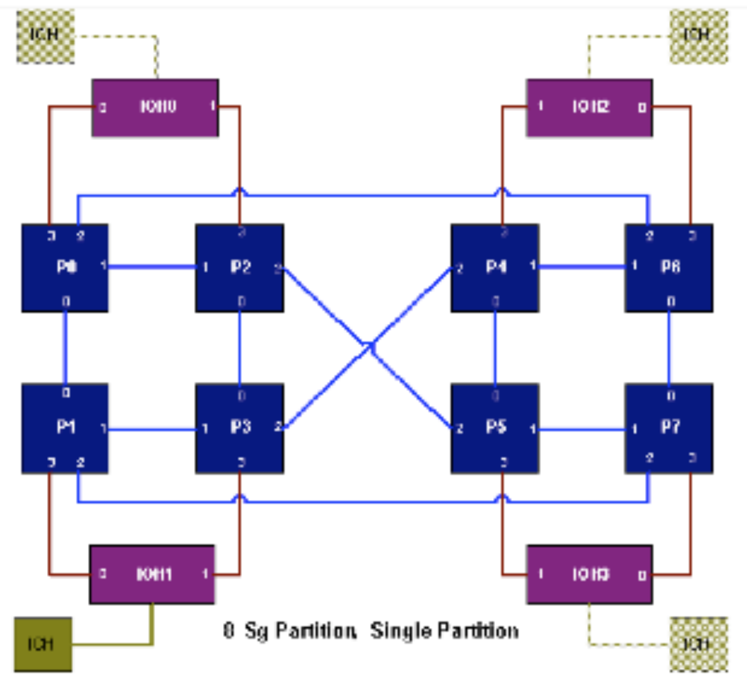
What Server Should We Use in a Datacenter?

- Many options
  - 1-socket server
  - 2-socket server
  - 4-socket server
  - …
  - 64-socket server
  - …

- What are the issues to consider?

2 vs 4 vs 8 Sockets per Server





- What is great about 2 vs 1 socket?
- Why not 4 or 8 sockets then?

Performance Scaling of Internet Scale Applications

[IEEE Micro'11]

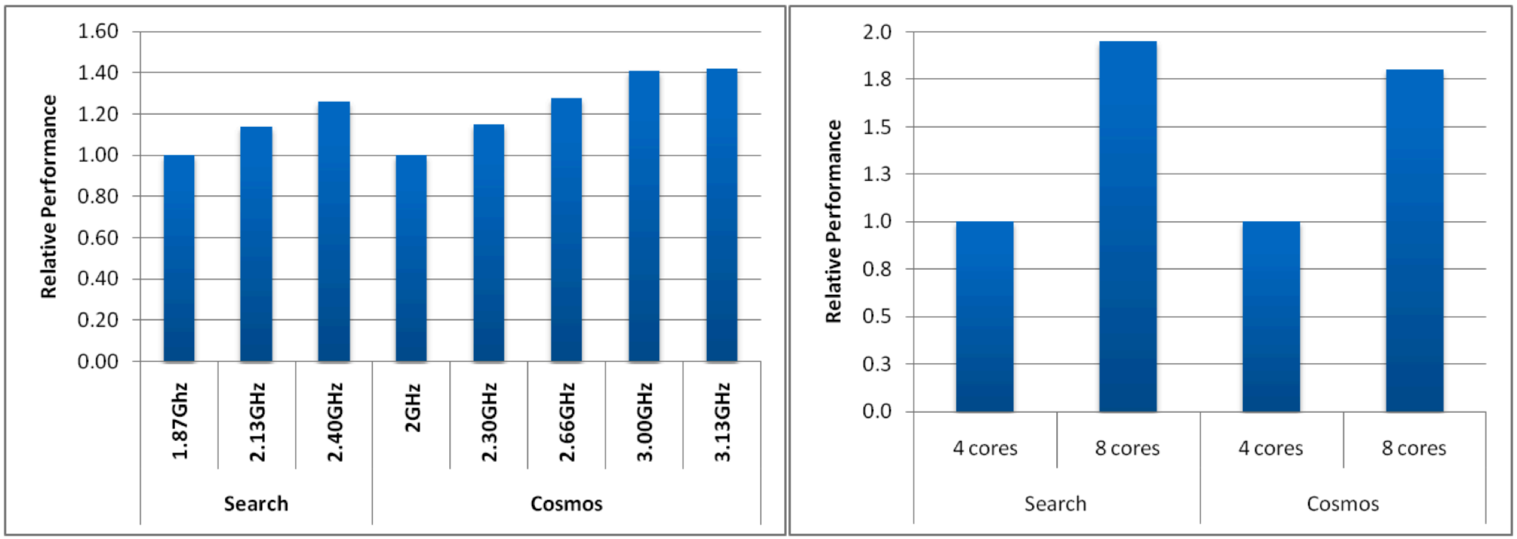


Figure 0. Performance scaling as a function of processor frequency and number of cores for Bing and Cosmos.

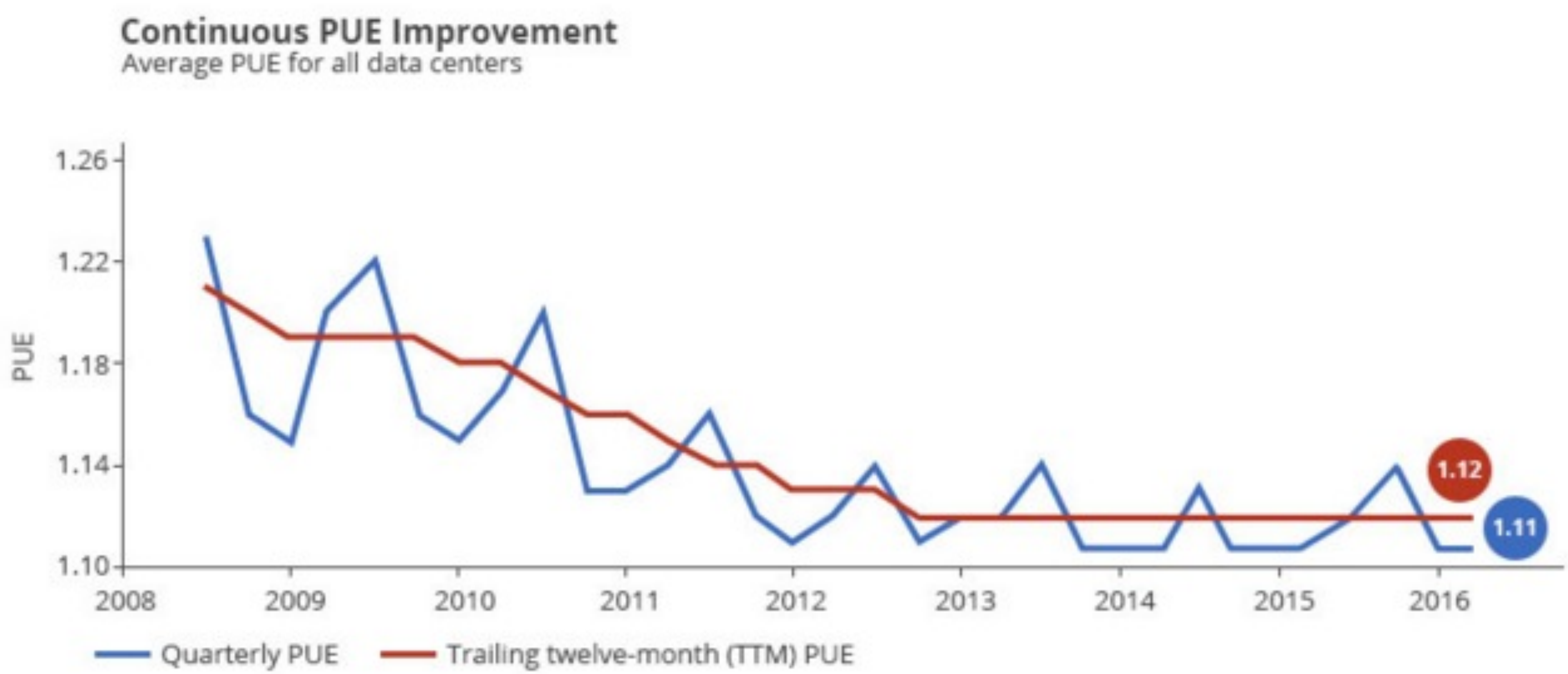
- Scaling analysis for Search & MapReduce at Microsoft
- Any observations?

Performance Metrics

- Completion time (e.g., how fast)
  - Of a certain operations
- Availability
- Power/energy
- Total cost of ownership (TCO)

Power Usage Effectiveness

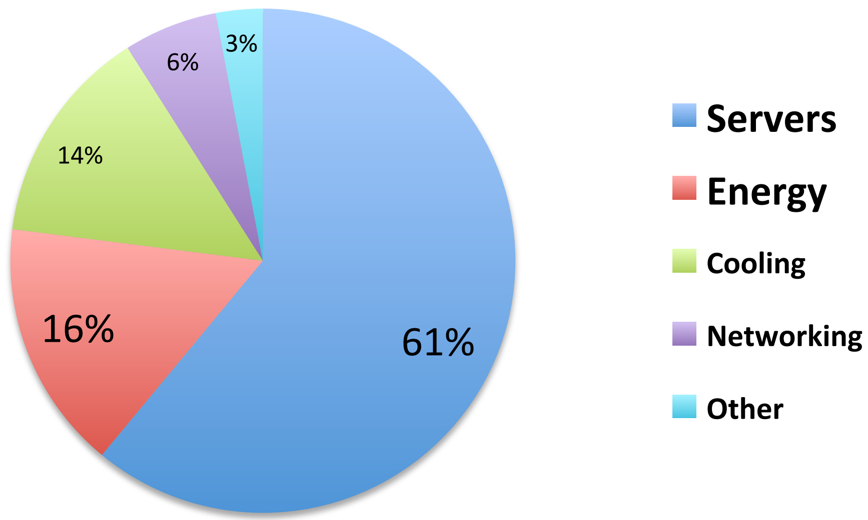
- PUE = Total datacenter power / IT equipment power



Total Cost of Ownership (TCO)

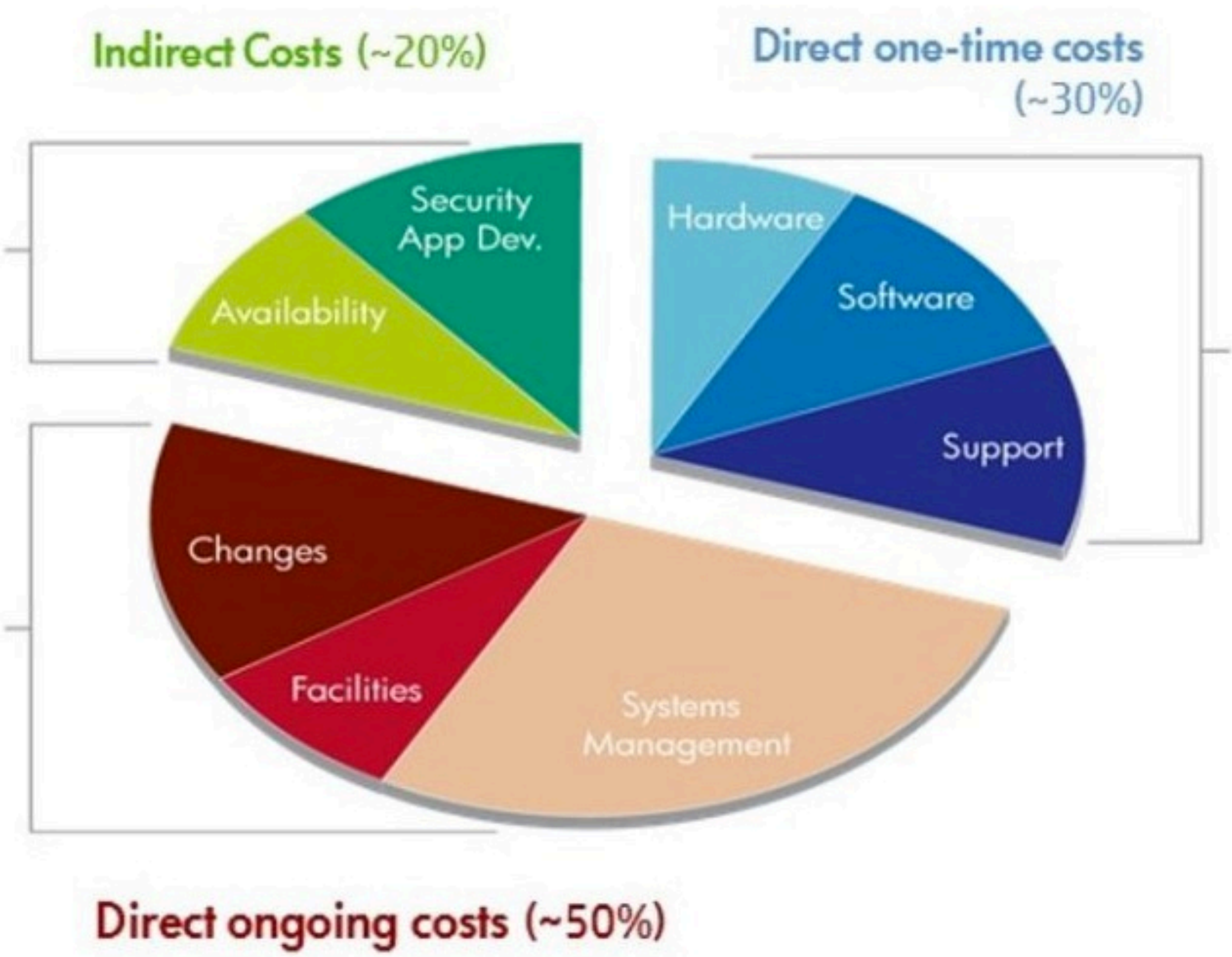
- Capital expenses
  - Land, building, generators, air conditioning, computing equipment
- Operating expenses
  - Electricity repairs
- Cost of unavailability

TCO Breakdown



- Observations
  - >50% of cost in buying the hardware
  - ~30% costs related to power
  - Networking ~10% of overall costs (including cost for servers)

TCO Breakdown (2)



Cost Analysis

- Cost model powerful tool for design tradeoffs
  - Evaluate "what-if" scenarios

- E.g., can we reduce power cost with different disk?

- A 1TB disk uses 10W of power, costs \$90. An alternate disk consumes only 5W, but costs \$150. If you were the data center architect, what would you do?

Answer

- A 1TB disk uses 10W of power, costs $90. An alternate disk consumes only 5W, but costs $150. If you were the data center architect, what would you do?

- @ $2/Watt – even if we saved the entire 10W of power for disk, we would save $20 per year. We are paying $60 more for the disk – probably not worth it.
  - What is this analysis missing?

Reliability & Availability

- Common goal for services: 99.99% availability
  - 1 hour of down-time per year
- But with thousands of nodes, things will crash
  - Example: with 10K servers rated at 30 years of MTBF, you should expect to have 1 failure per day

Reliability Challenges

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)

~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)

~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)

~1 network rewiring (rolling ~5% of machines down over 2-day span)

~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)

~5 racks go wonky (40-80 machines see 50% packetloss)

~8 network maintenances (4 might cause ~30-minute random connectivity losses)

~12 router reloads (takes out DNS and external vips for a couple minutes)

~3 router failures (have to immediately pull traffic for an hour)

~dozens of minor 30-second blips for dns

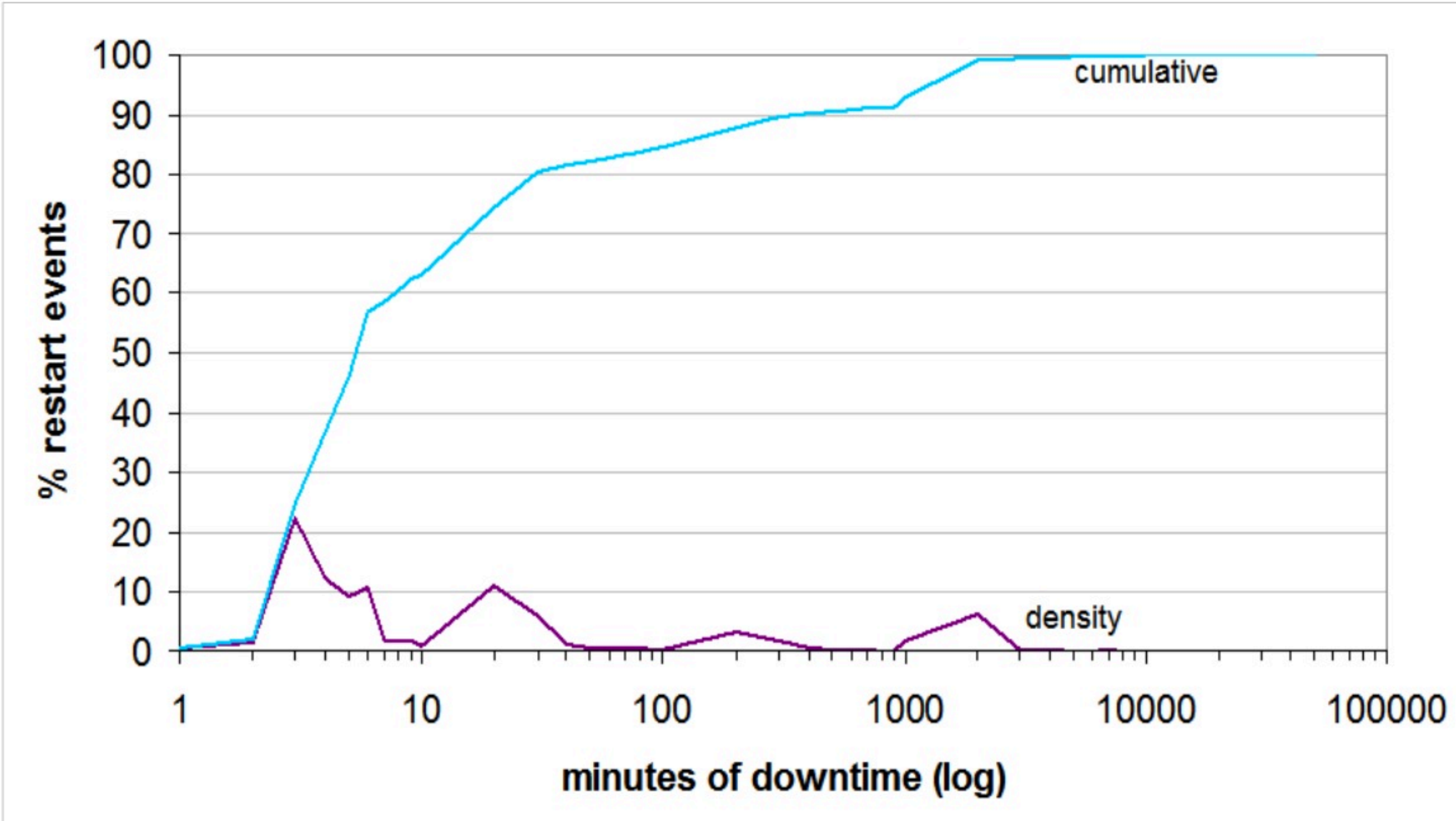
~1000 individual machine failures

~thousands of hard drive failures

slow disks, bad memory, misconfigured machines, flaky machines, etc.

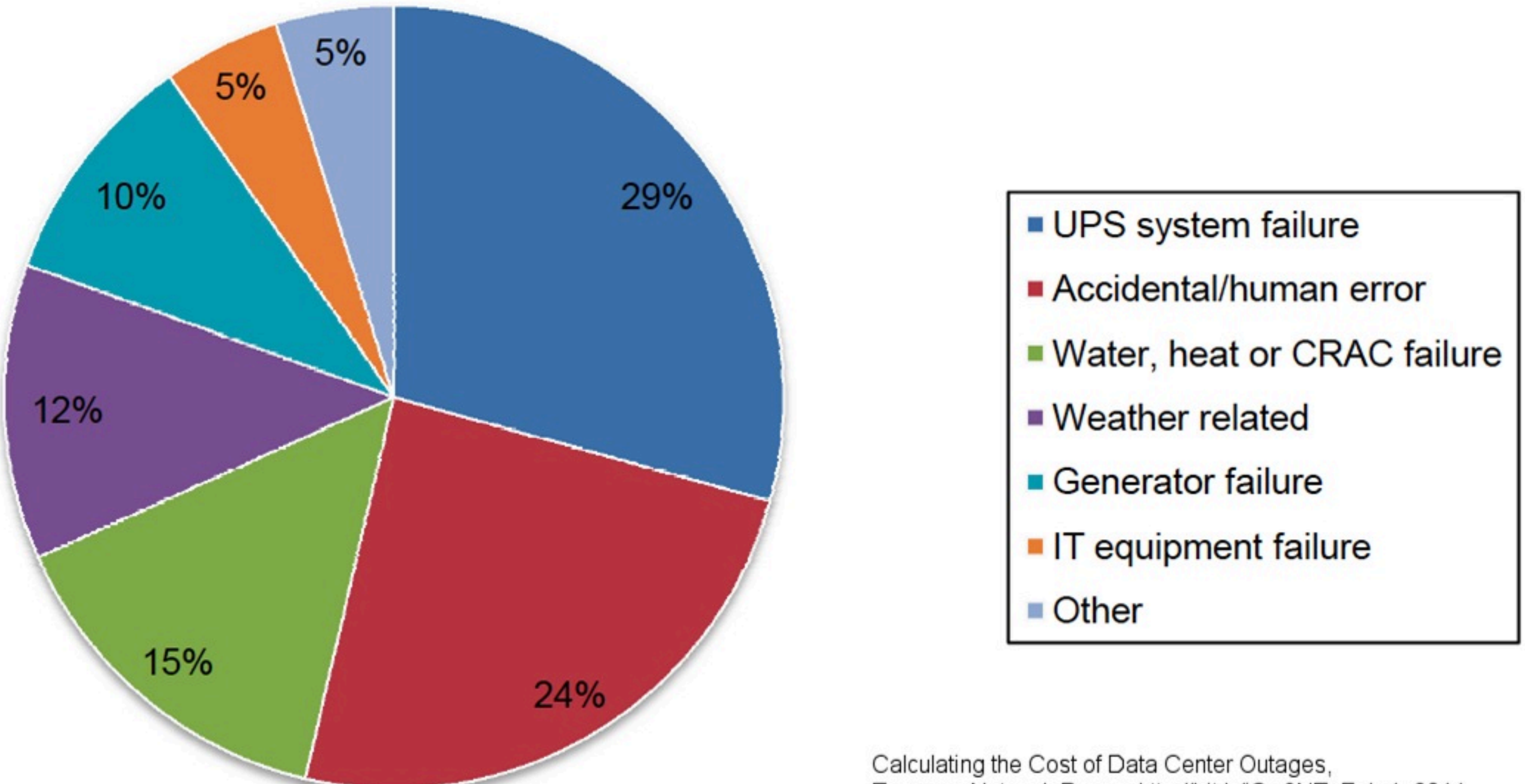
Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

Downtime Density



Sources of Outages

Computed from 41 benchmarked data centers



Calculating the Cost of Data Center Outages,
Emerson Network Power, http://bit.ly/jCw9NE, Feb 1, 2011

Robustness to Failures

- Failover to other replicas/datacenters
- Bad backend detection
  - Stop using for live requests until behavior gets better
- More aggressive load balancing when imbalance is more severe
- Make your apps do something reasonable even if not all is right
  - Better to give users limited functionality than an error page

Consistency

- Multiple data centers implies dealing with consistency issues
  - Disconnected/partitioned operation relatively common, e.g., datacenter down for maintenance
  - Insisting on strong consistency likely undesirable
  - "We have your data but can't show it to you because one of the replicas is unavailable"
  - Most products with mutable state gravitating towards "eventual consistency" model
  - A bit harder to think about, but better from an availability standpoint

Performance/Availability Techniques in DCs

Technique	Performance	Availability
Replication	✔	✔
Partitioning (sharding)	✔	✔
Load-balancing	✔	
Watchdog timers		✔
Integrity checks		✔
App-specific compression	✔	
Eventual consistency	✔	✔

Characteristics of Internet-scale Services

- Huge datasets, user sets, …

- High request level parallelism
  – Without much read-write sharing

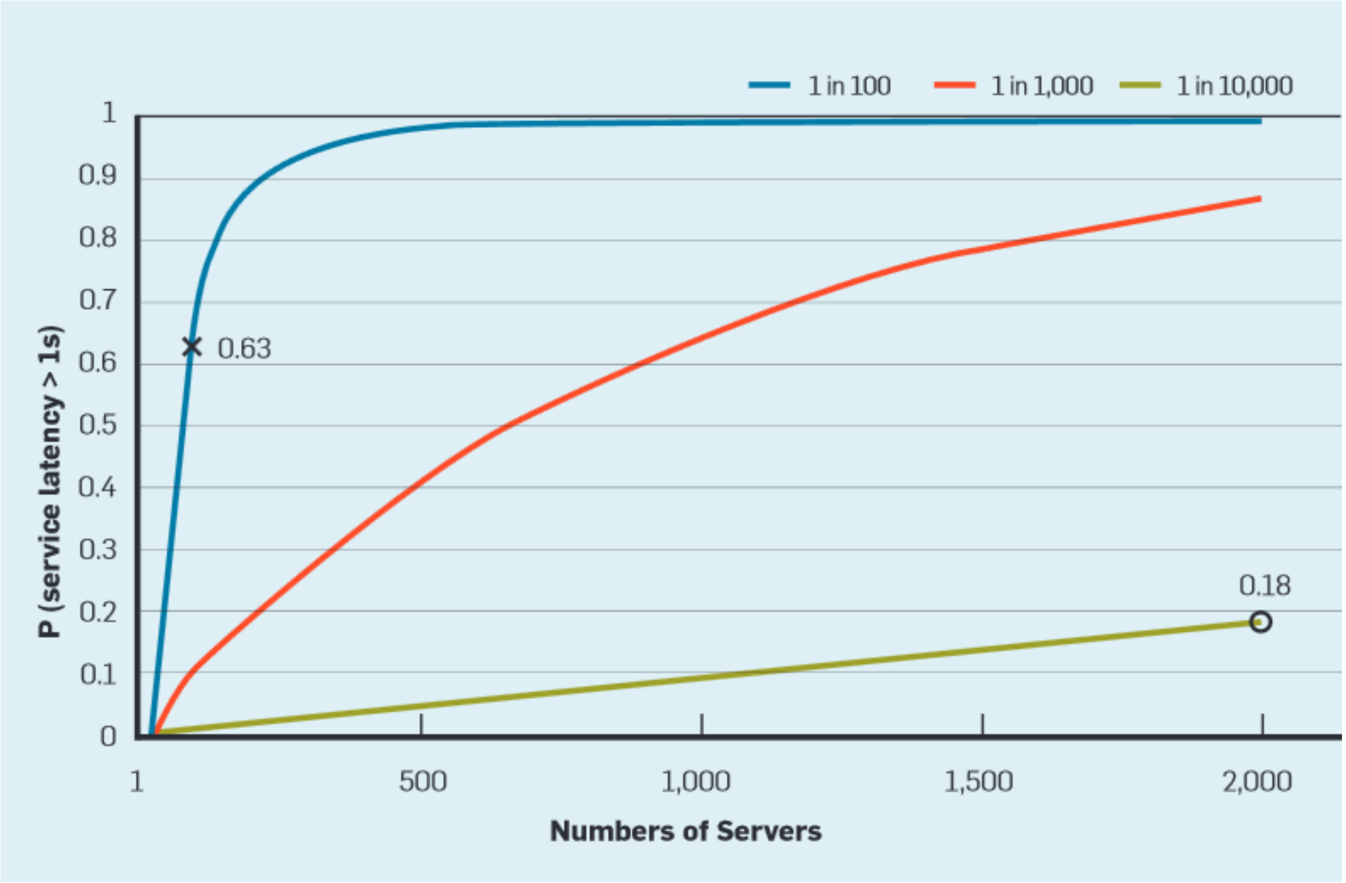
- High workload churn
  – New releases of code on a weekly basis

- Require fault-free operation

Performance Metrics

- Throughput
  - User requests per second (RPS)
  - Scale-out address this (more servers)
- Quality of Service (QoS)
  - Latency of individual requests (90th, 95th, or 95th percentile)
  - Scale-out does not necessarily help
- Interesting notes
  - The distribution matters, not just the averages
  - Optimizing throughput often hurts latency
    - And optimizing latency often hurts power consumption
  - At the end, it is RPS/$ within some QoS constraints

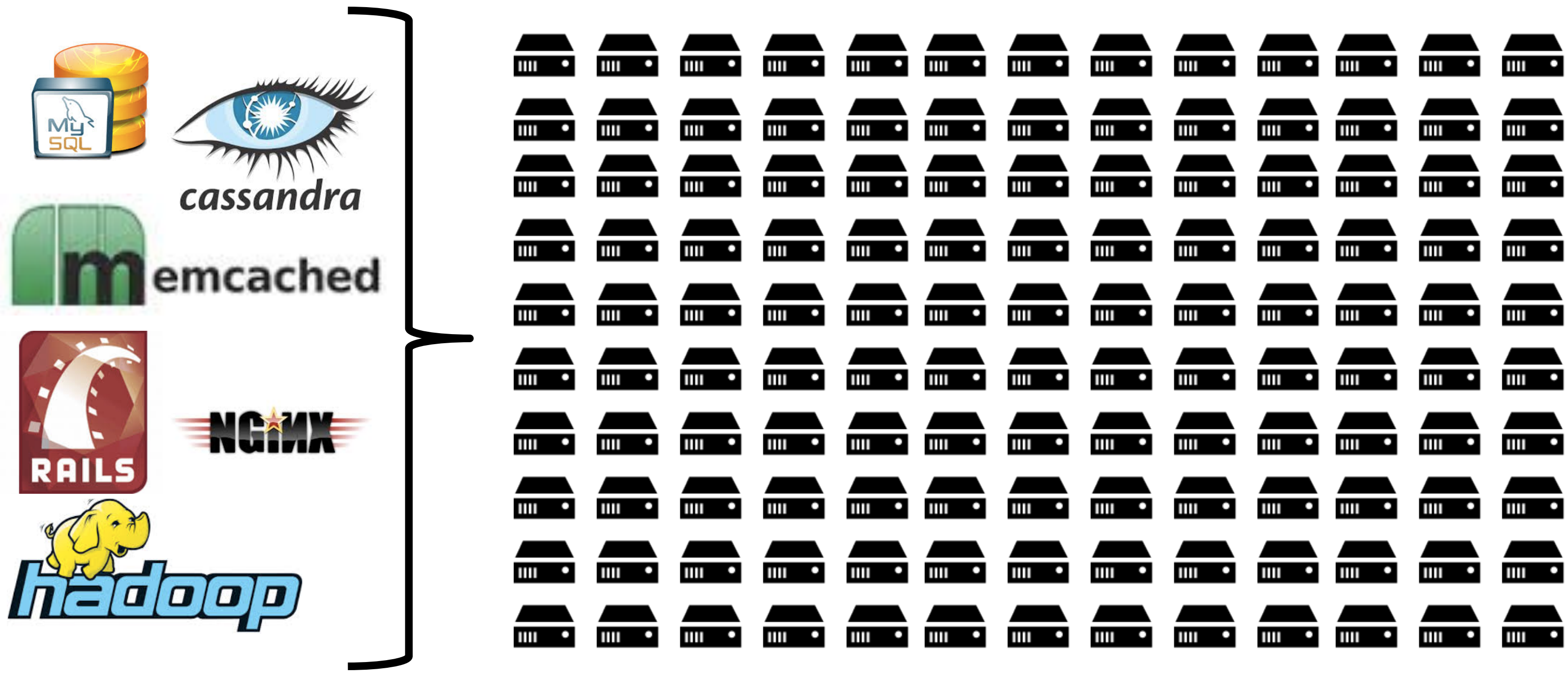
Tail At Scale



- Larger clusters → more prone to high tail latency

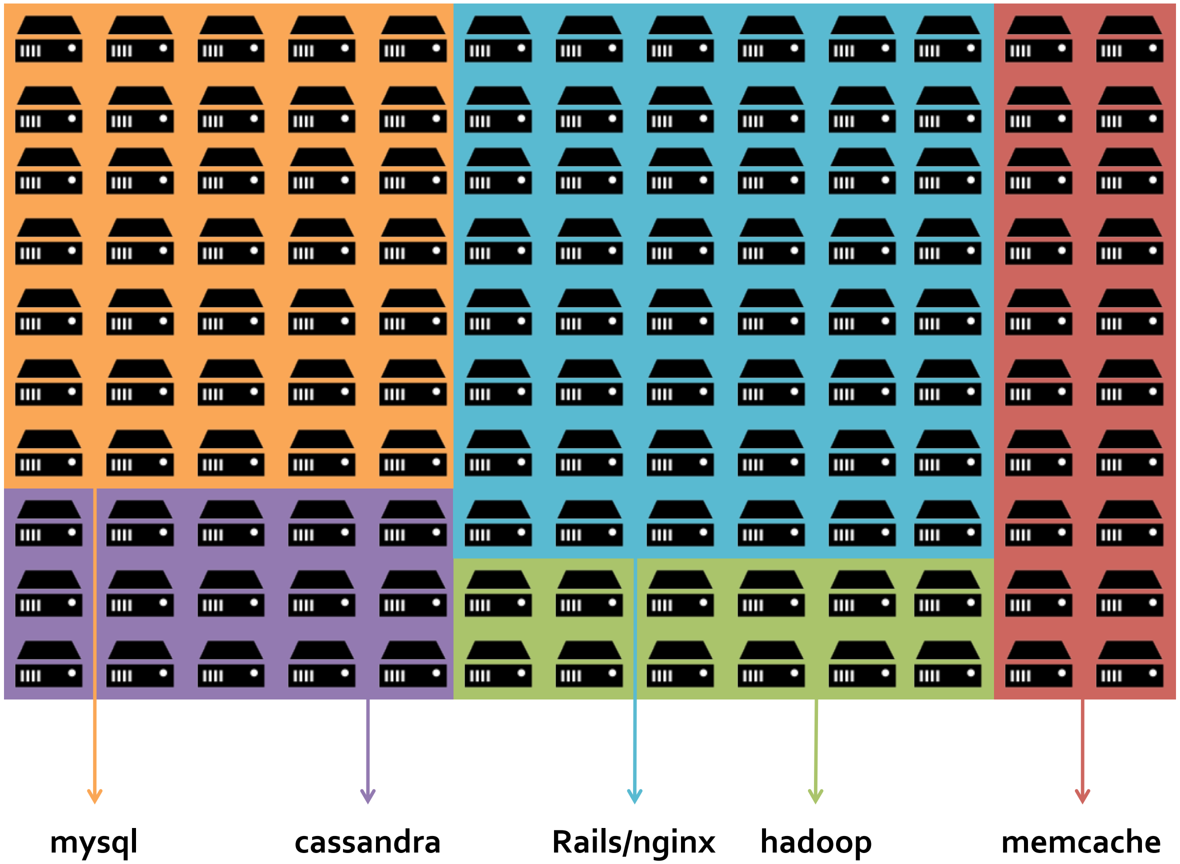
[1]The Tail at Scale. Jeffrey Dean, Luiz André Barroso. CACM, Vol. 56 No. 2, Pages 74-80, 2013

Resource Assignment Options



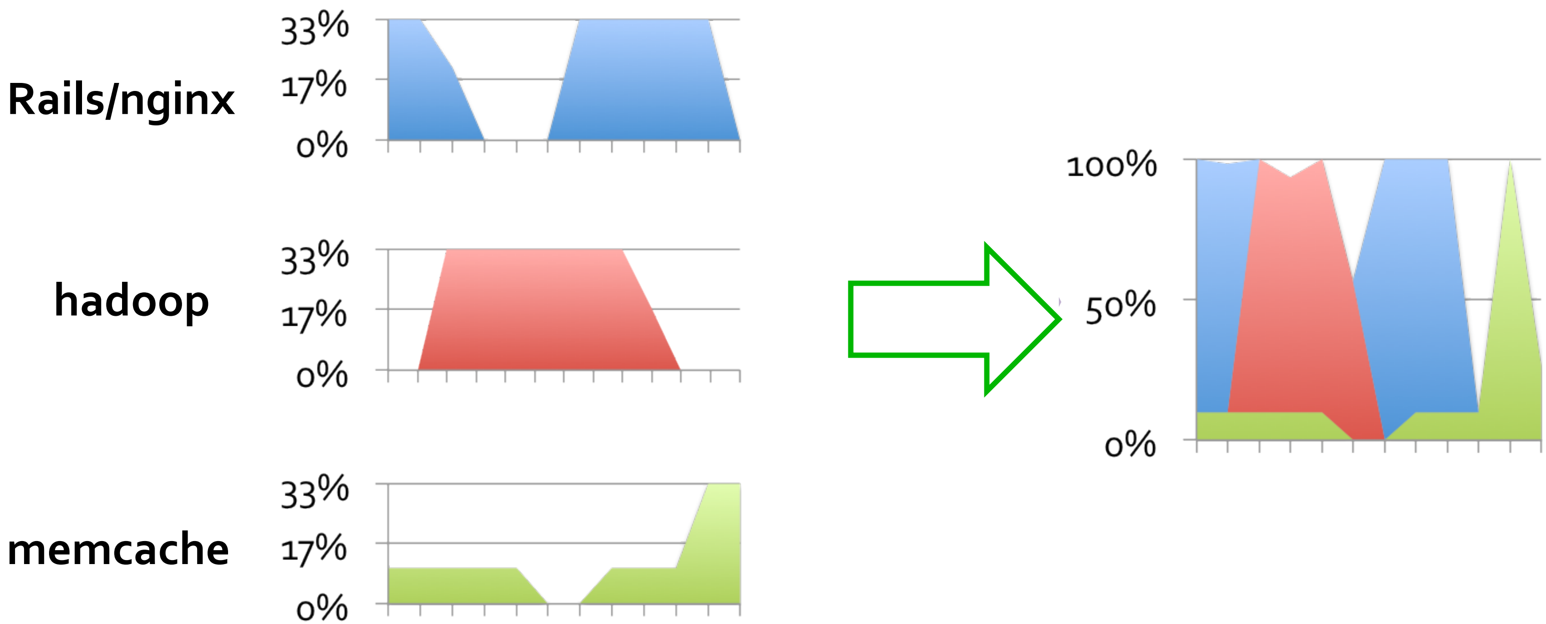
- How do we assign resources to apps?
- Two major options: private vs shared assignment

Private Resource Assignment



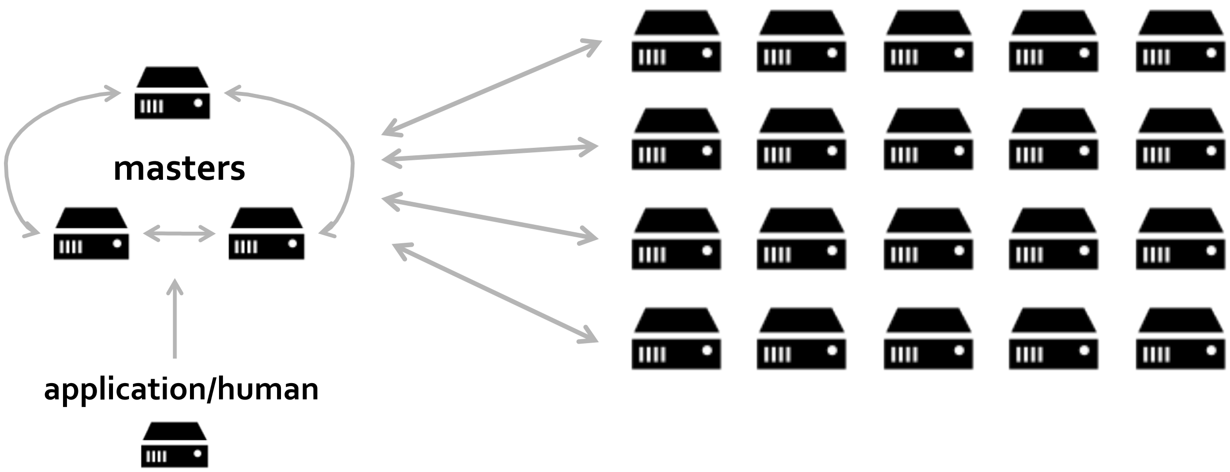
- Each app receives a private, static set of resources
- Also known as static partitioning

Shared Resource Assignment



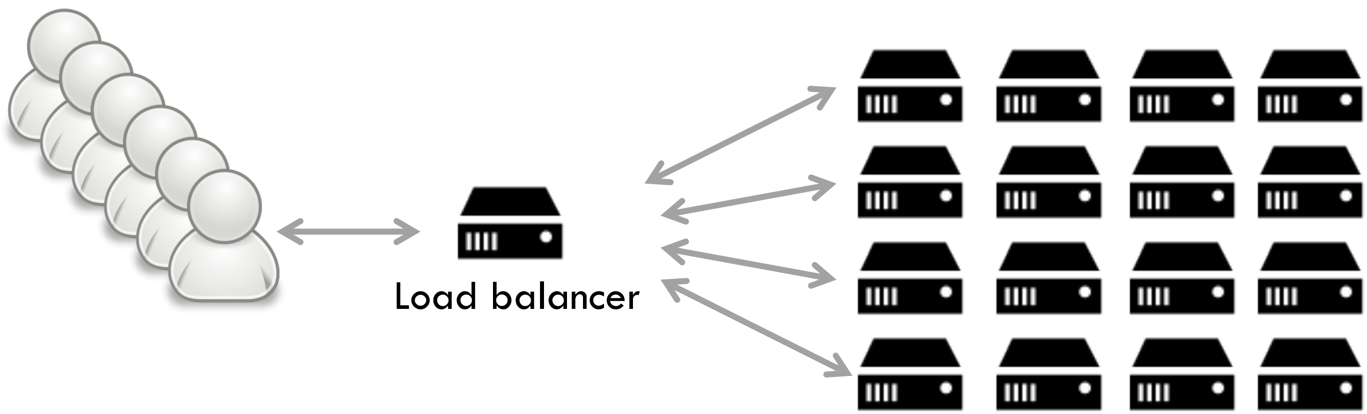
- Shared resources: flexibility → high utilization
  - Common case: user-facing services + analytics on same servers
  - Also helps with failures, maintenance, and provisioning

Shared Cluster Management



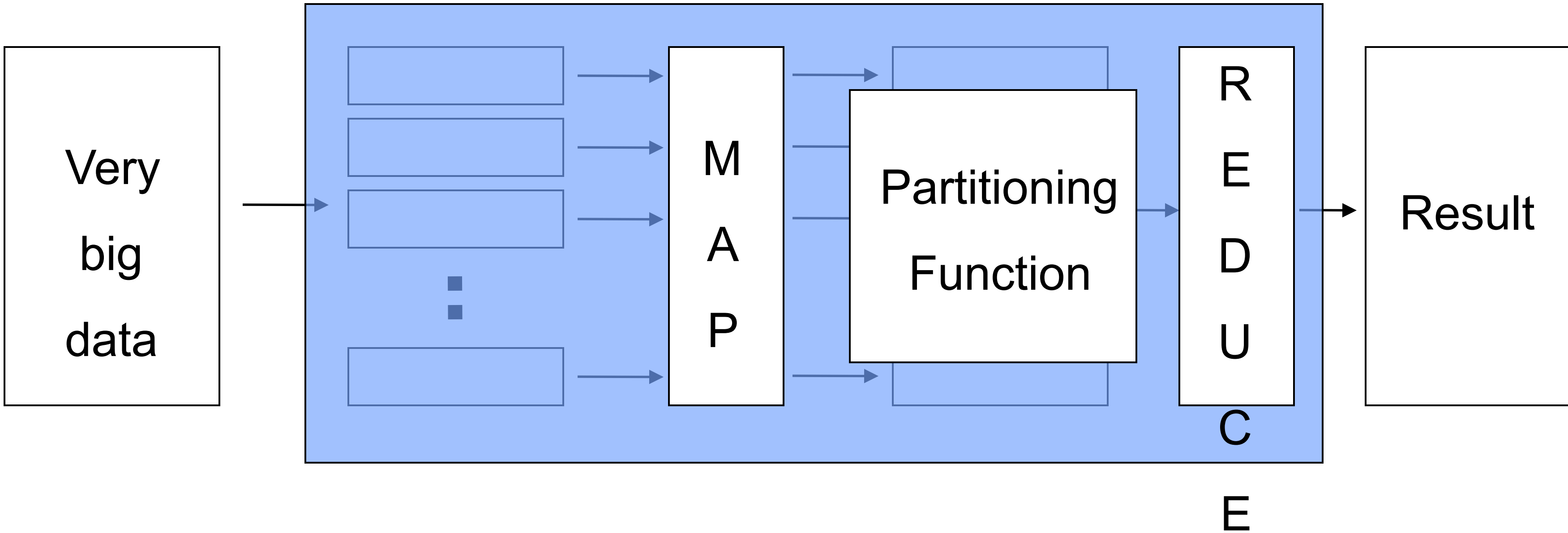
- The manager schedules apps on shared resources
  - Apps request resource reservations (cores, DRAM, …)
  - Manager allocates and assigns specific resources
    - Considering performance, utilization, fault tolerance, priorities, …
    - Potentially, multiple apps on each server
  - Multiple manager architectures (see Borg paper for example)

Autoscaling



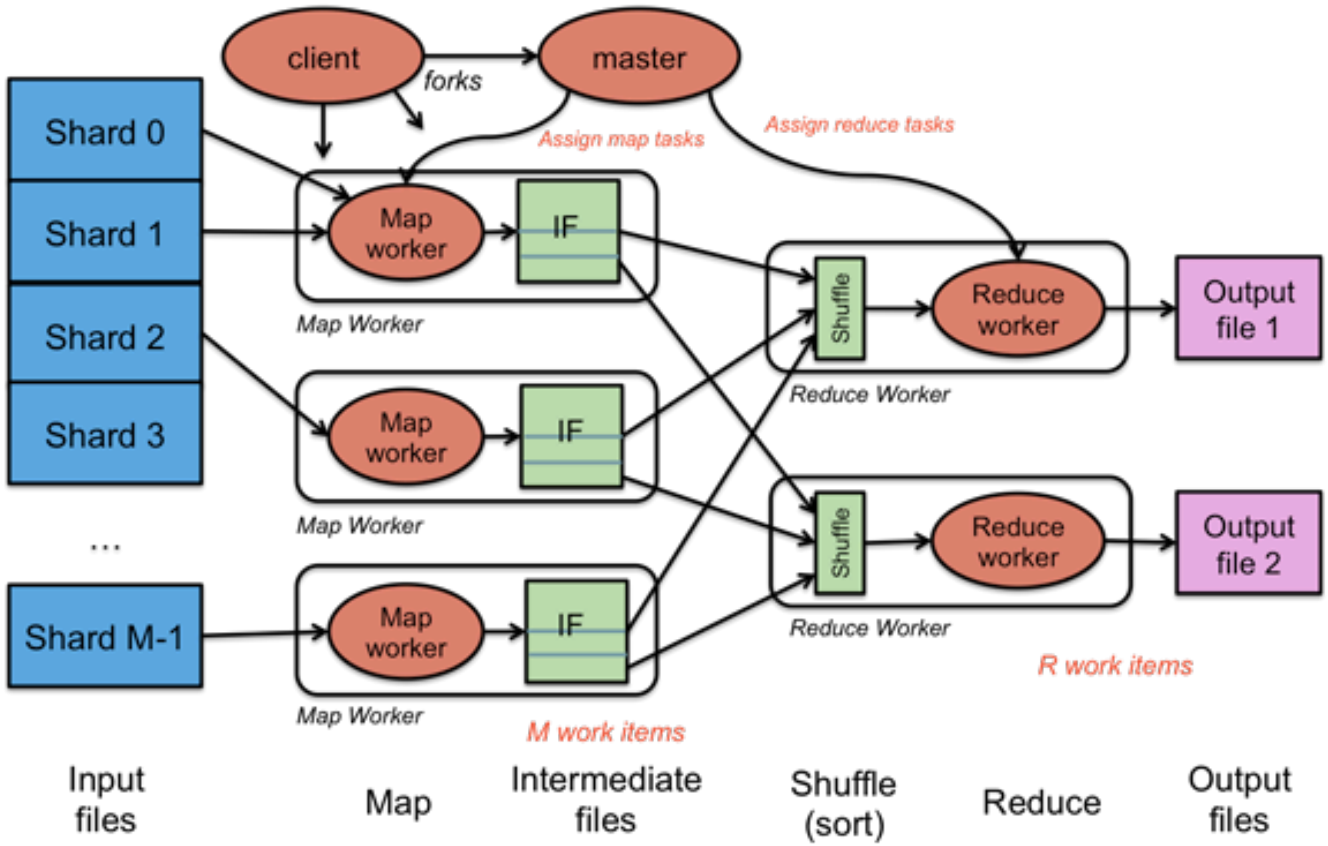
- Monitor app performance or server load
  - [Chase'01, AWS AutoScale, Lim'10, Shen'11, Gandhi'12, ...]
- Adjust resources given to app
  - Add or remove to meet performance goal
  - Feedback-based control loop

Map+Reduce



- Map:
  - Accepts *input* key/value pair
  - Emits *intermediate* key/value pair
- Reduce :
  - Accepts *intermediate* key/value* pair
  - Emits *output* key/value pair

Analytics Example: MapReduce

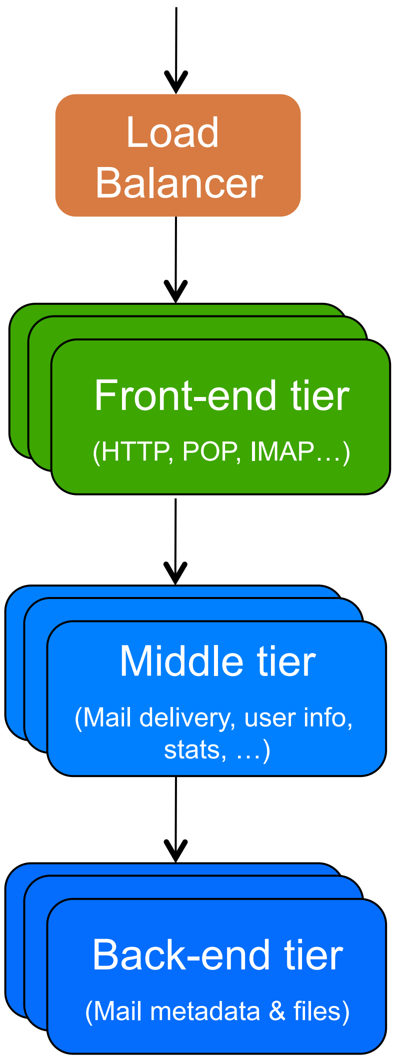


[Figure credit: Paul Krzyzanowski]

- Single-tier architecture
  - Distributed FS, worker servers, coordinator
  - Disk based or in-memory
- Metric: throughput

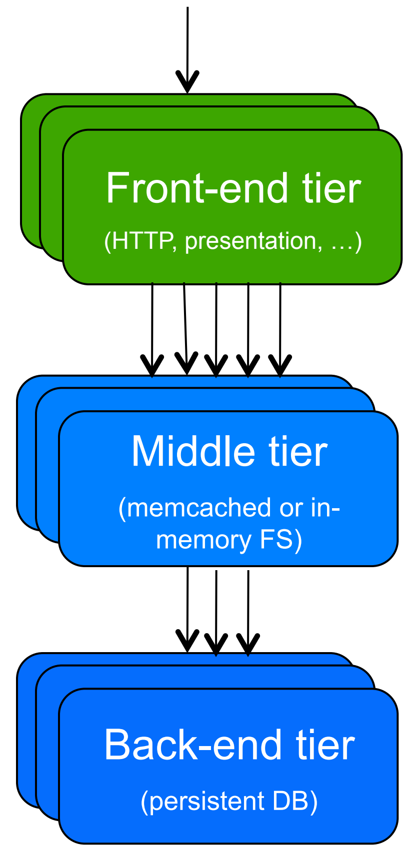
Example 3-tier App: WebMail

- May include thousands of machines, PetaBytes of data, and billions of users
- 1st tier: protocol processing
  - Typically stateless
  - Use a load balancer
- 2nd tier: application logic
  - Often caches state from 3rd tier
- 3rd tier: data storage
  - Heavily stateful
  - Often includes bulk of machines



Example: Social Networking

- 3 tier system
  - Web server, fast user data storage, persistent storage
  - 2rd tier: latency critical, large number of servers



- User data storage
  - Using memcached for distributed caching
  - 10s of Tbytes in memory (Facebook 150TB)
  - Sharded and replicated across many servers
  - Read/write (unlike search), bulk is read-dominated
- From in-memory caching to in-memory FS
  - RAMcloud @Stanford, Sinfonia @HP, …

Acknowledgements

- Christos Kozyrakis, Christina Delimitrou
  - Based on EE282 @ Stanford
- “Designs, Lessons, and Advice from Building Large Distributed Systems” by Jeff Dean, Google Fellow
- “A thousand chickens or two oxen? Part 1: TCO” by Isabel Martin
- “UPS as a service could be an economic breakthrough” by Mark Monroe
- “Take a close look at MapReduce” by Xuanhua Shi

Reducing Tail Latency

- Reduce queuing (reduce head of line blocking)
- Separate different types of requests
- Coordinate background activities
- Hedged requests to replicas
- Tied requests to replicas
- Micro-sharding & selective replication
- Latency induced probation , canary requests

- See The Tail @ Scale paper for details