CS 795: Distributed Systems & Cloud Computing
Fall 2018

Lec 1: Clouds & Data consistency

Yue Cheng

Announcement

- Paper presentation Doodle sign-up is out. Please complete by signing-up at least 2 papers by this Friday
- After that, presentation schedule will be sorted out quickly

What is Cloud Computing?

What is Cloud Computing?

- Computing as a utility
  - Outsourced to a third party or internal organization

- Providers do more, tenants do less

Types of cloud services

- Infrastructure as a Service (**IaaS**): **VMs, storage**

Types of cloud services

- Infrastructure as a Service (**IaaS**): **VMs, storage**
- Platform as a Service (**PaaS**): **Web, MapReduce**

Types of cloud services

- Infrastructure as a Service (**IaaS**): **VMs, storage**
- Platform as a Service (**PaaS**): **Web, MapReduce**
- Software as a Service (**SaaS**): **Email, Messenger**

New cloud computing paradigm

- Function as a Service (**FaaS**)
  - AWS Lambda
  - Google Cloud Functions
  - Microsoft Azure Functions
- Runs functions in a Linux container on events
- Tenants focus on function logics without needing to worry about backend server maintenance, auto-scaling, etc.

Public vs. private clouds

- Public clouds
  - Shared across arbitrary orgs/customers
- Private clouds
  - Internal/private to one organization

Cloud economics: Tenants

- Pay-as-you-go (usage-based) pricing
  - Most services charged per minute, per byte, etc.
  - No minimum or up-front fee

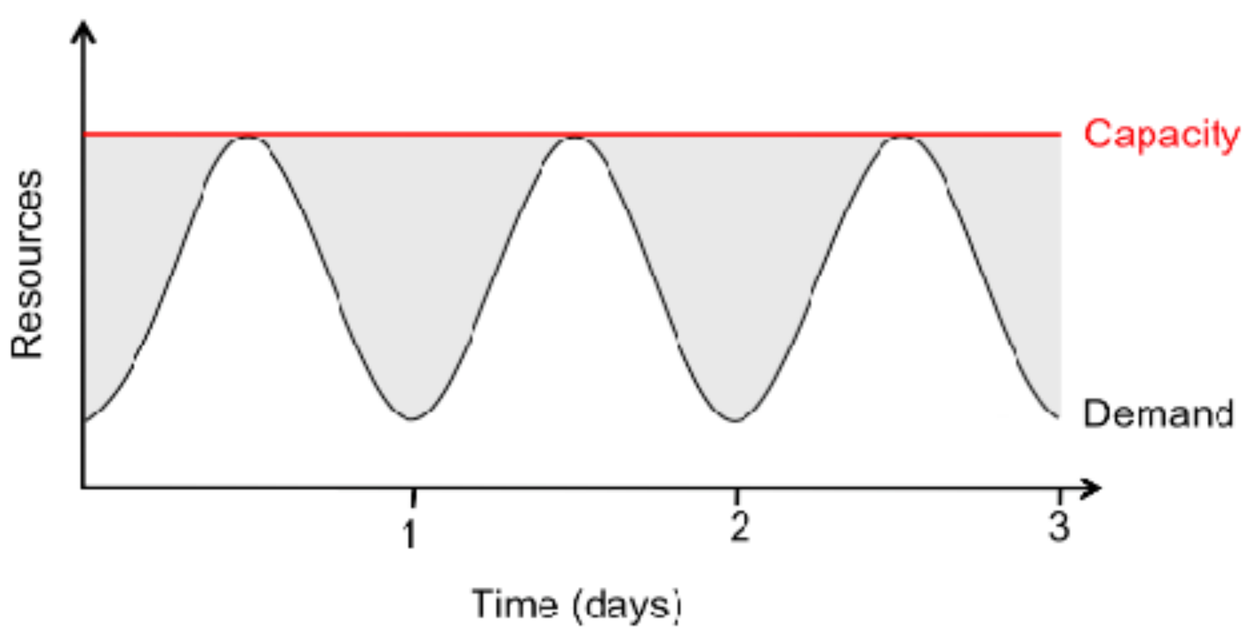
Cloud economics: Tenants

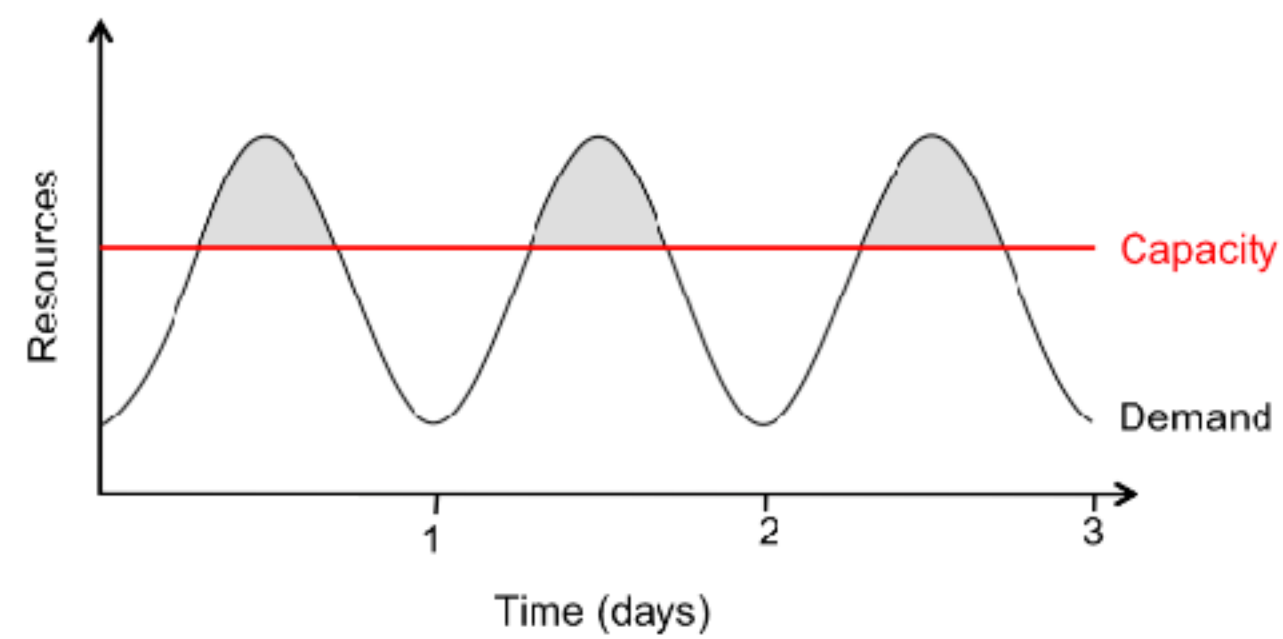
- Pay-as-you-go (usage-based) pricing
  - Most services charged per minute, per byte, etc.
  - No minimum or up-front fee

- Problem: How to perform strategic planning?

Cloud economics: Tenants

- Pay-as-you-go (usage-based) pricing
  - Most services charged per minute, per byte, etc.
  - No minimum or up-front fee
- Problem: How to perform strategic planning?





Cloud economics: Tenants

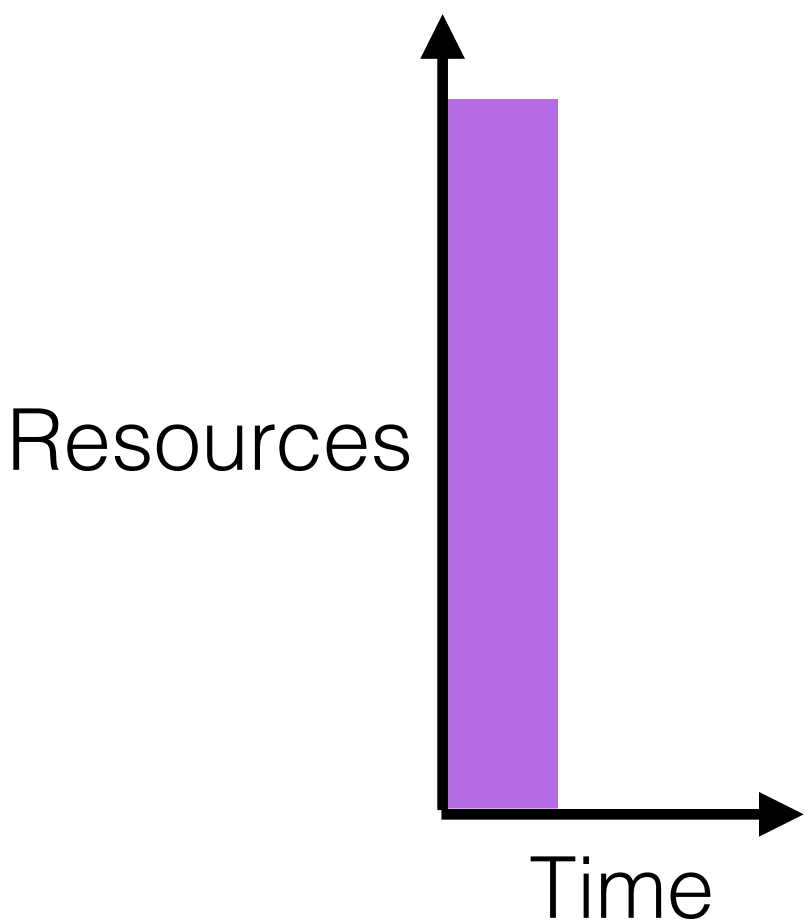
- Pay-as-you-go (usage-based) pricing
  - Most services charged per minute, per byte, etc.
  - No minimum or up-front fee

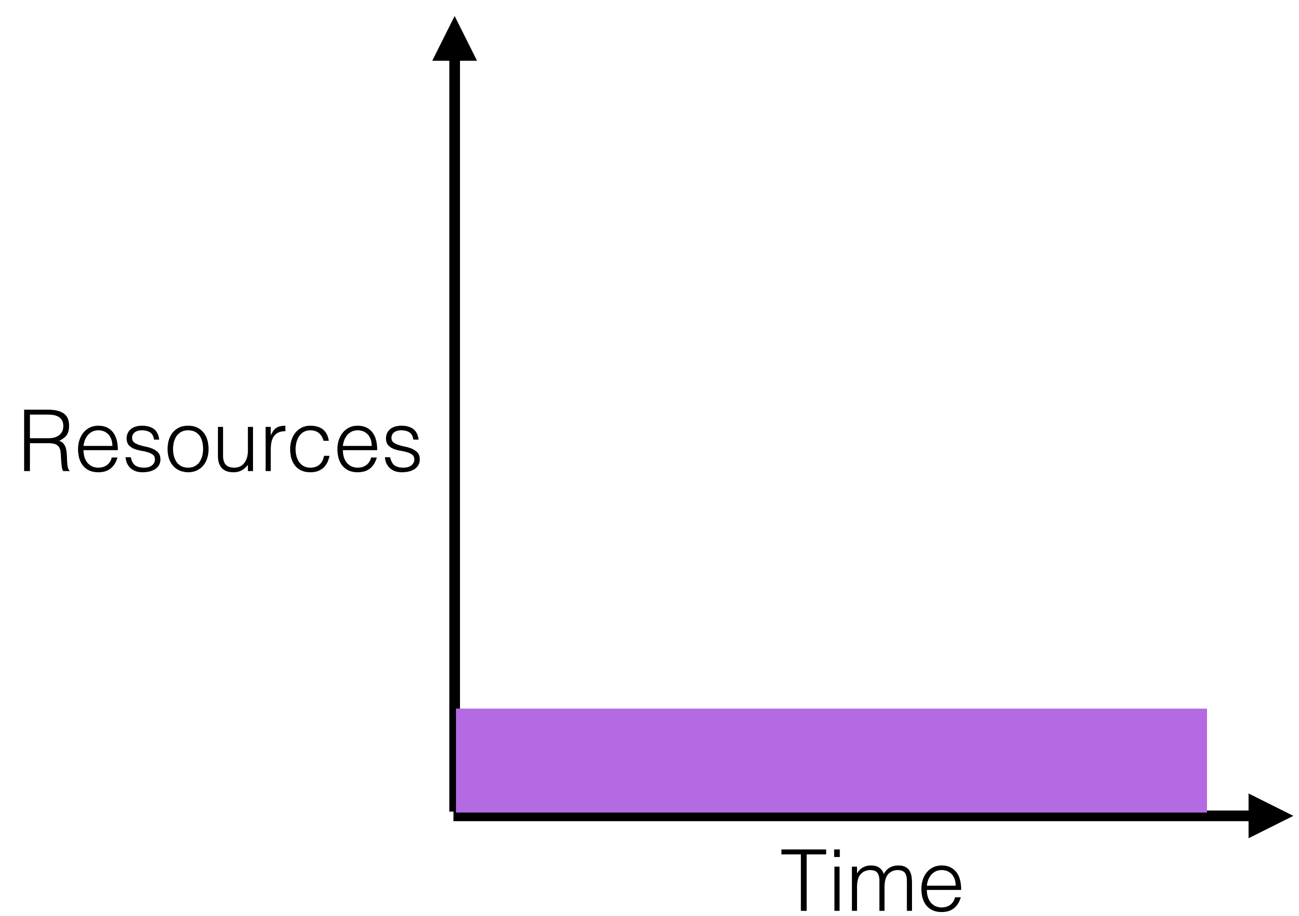
Caveat: Not essentially Pay-as-You-Go!

- Why?

Cloud economics: Tenants

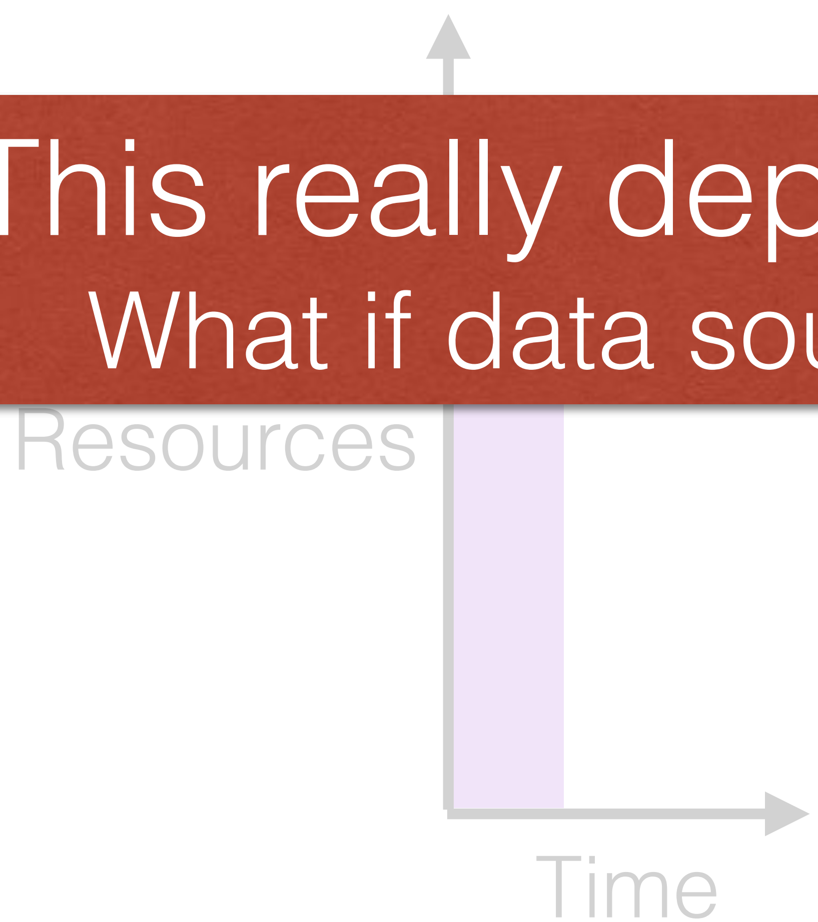
- Elasticity
  - Using 1000 servers for 1 hour costs the same as 1 server for 1000 hours
  - Same price to get a result faster

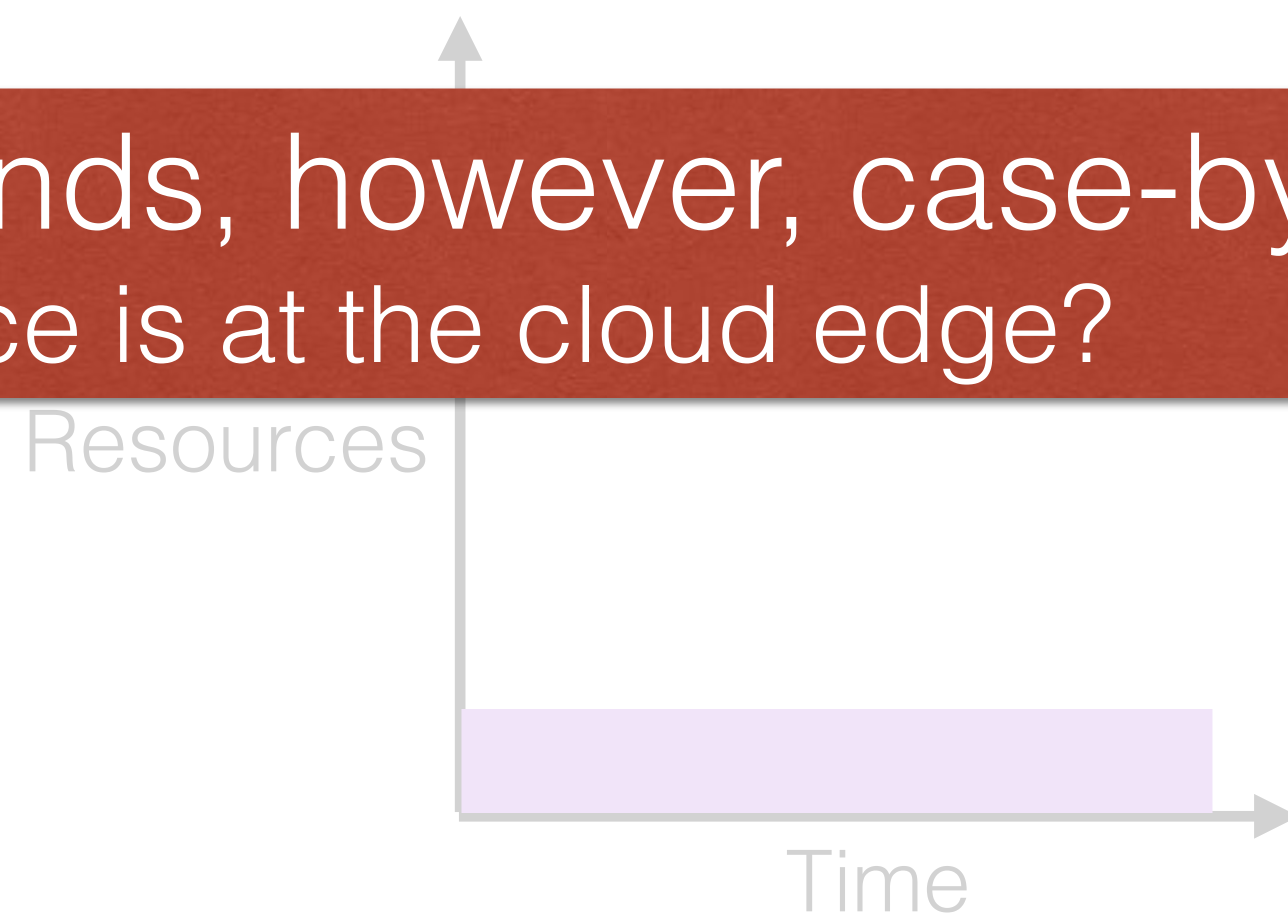




Cloud economics: Tenants

- Elasticity
  - Using 1000 servers for 1 hour costs the same as 1 server for 1000 hours
  - Same price to get a result faster





This really depends, however, case-by-case

- What if data source is at the cloud edge?

Cloud economics: Providers

- Economies of scale
  - Purchasing, powering, managing machines at scale gives lower per-unit costs than that of costumers

Technology	Cost in Medium DC	Cost in Large DC	Ratio
Network	$95 per Mbit/sec/month	$13 per Mbit/sec/month	7.1
Storage	$2.2 per GB/month	$0.4 per GB/month	5.7
Admin.	~140 Servers/admin	>1000 Servers/admin.	7.1

Cloud economics: Providers

- Economies of scale
    - Purchasing, powering, managing machines at scale gives lower per-unit costs than that of costumers

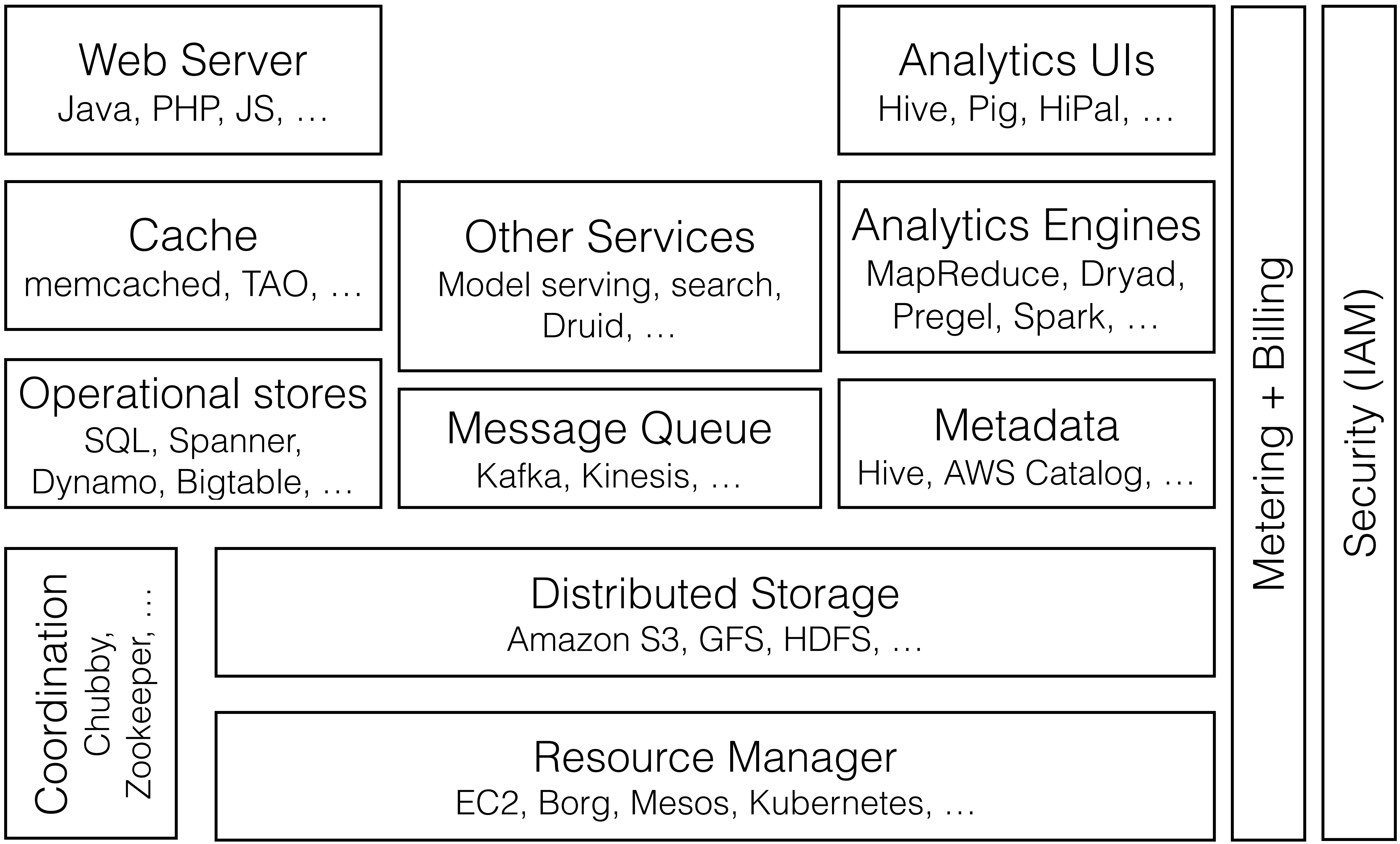
Technology	Cost in Medium DC	Cost in Large DC	Ratio
Network	$95 per Mbit/sec/month	$13 per Mbit/sec/month	7.1
Storage	$2.2 per GB/month	$0.4 per GB/month	5.7
Admin.	~140 Servers/admin	>1000 Servers/admin.	7.1

- Leveraging existing investments: Many AWS technologies were initially developed for Amazon's internal operations

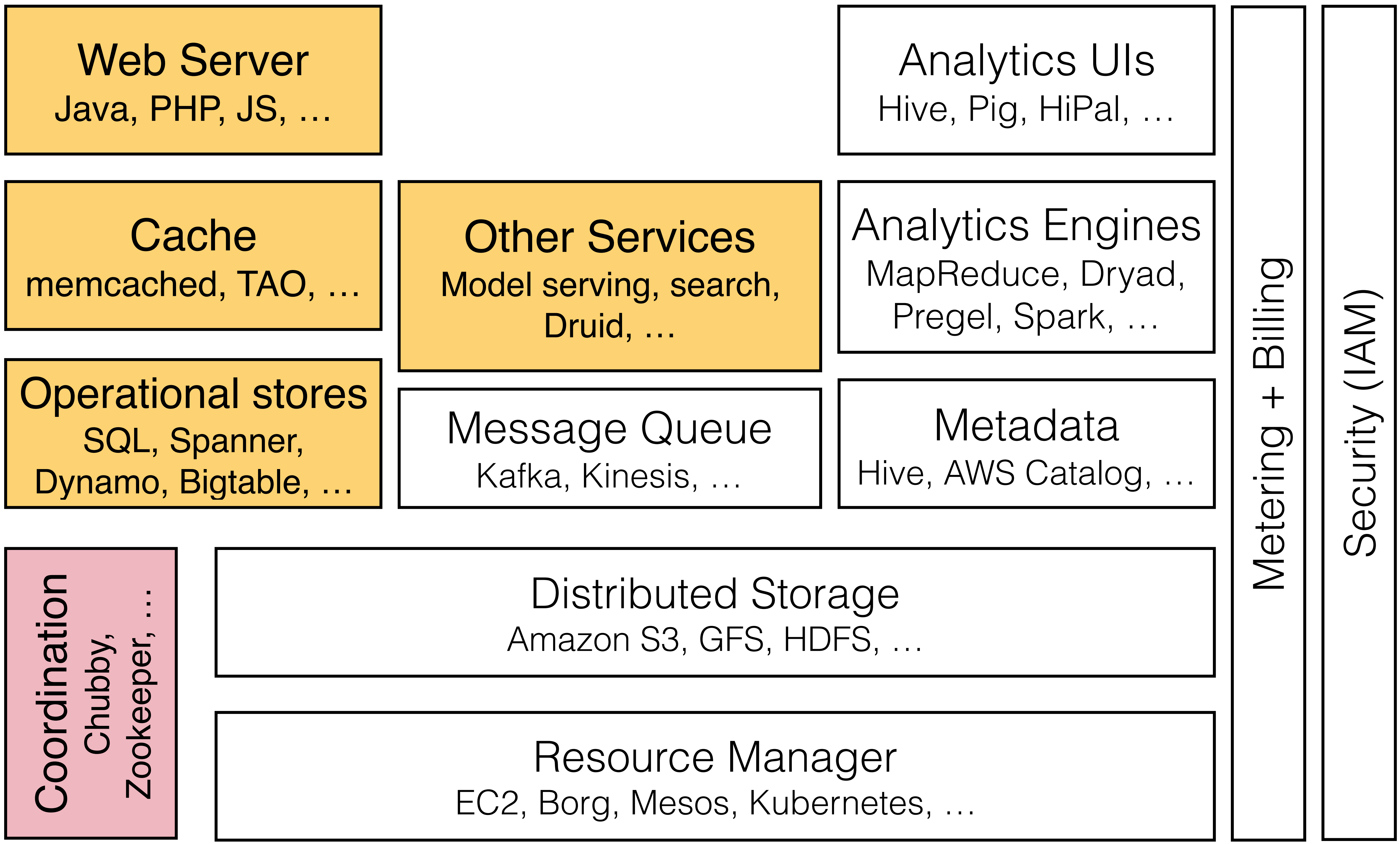
Common cloud applications

- Web/mobile applications
- Data analytics (MapReduce, SQL, ML, etc.)
- Stream processing
- Parallel/HPC batch computation

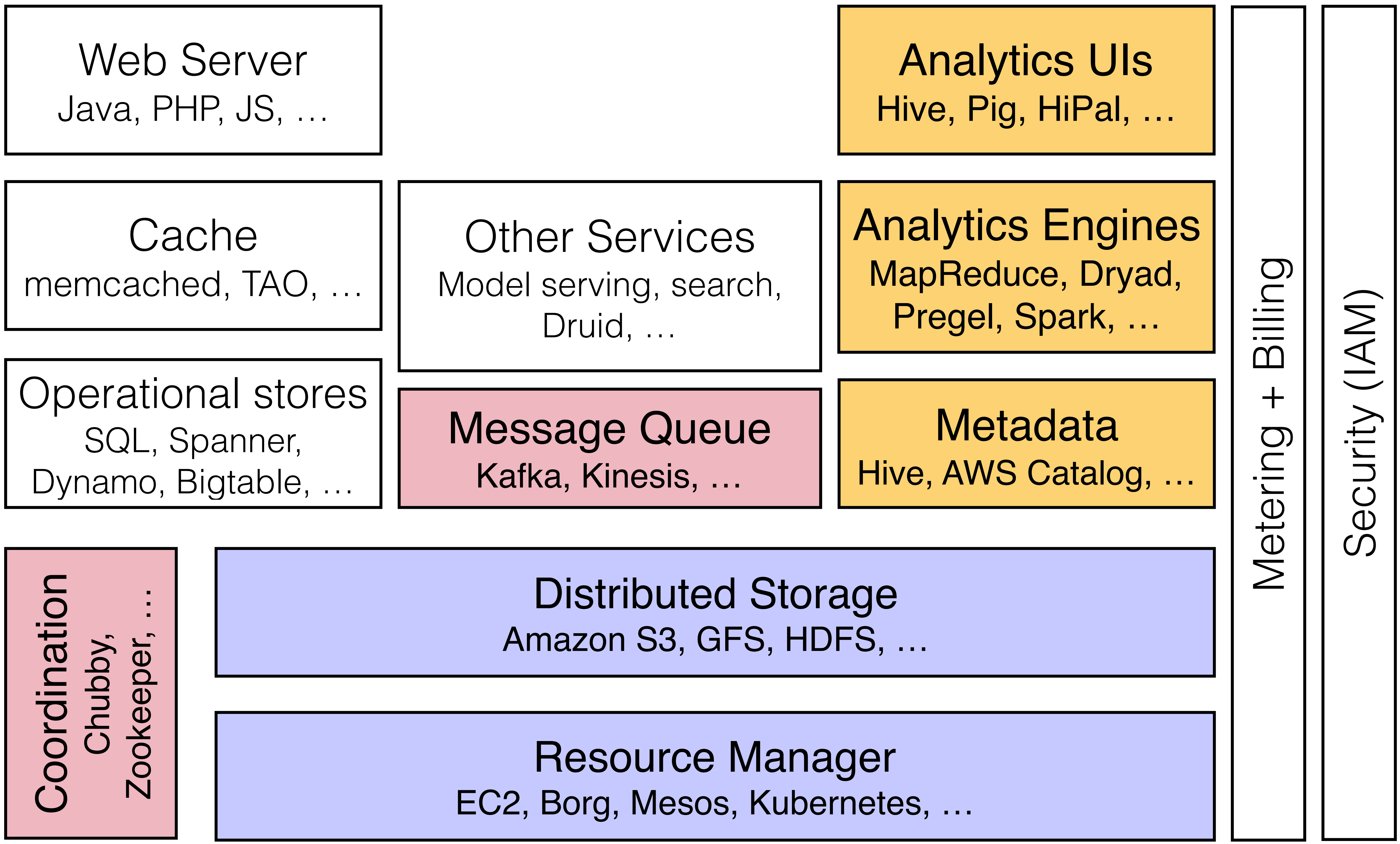
Cloud software stack



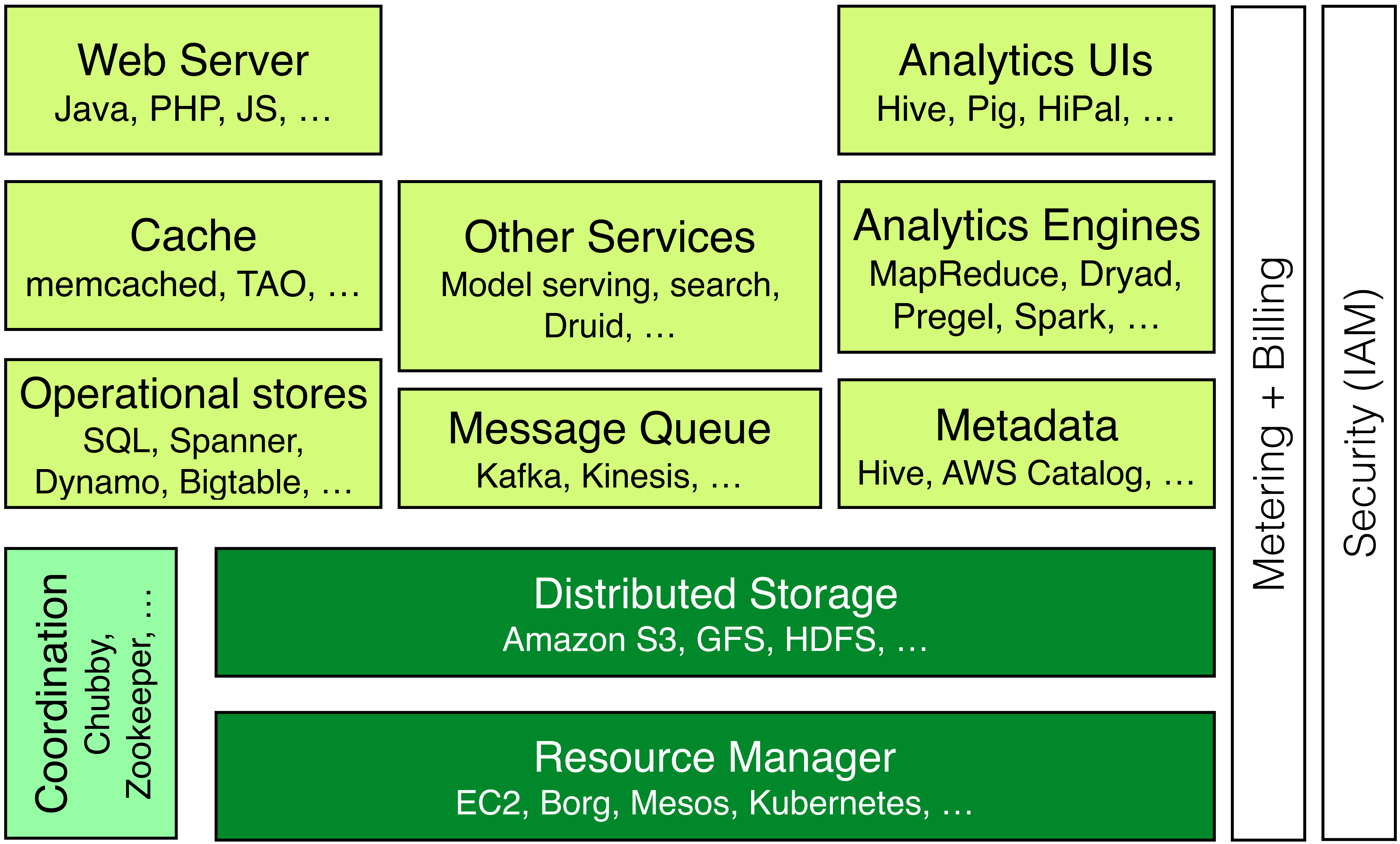
Example: Web applications



Example: Analytics warehouse



Components offered as PaaS



Cloud economics case study: Pricing games for hybrid cloud object stores

amazon web services™ | S3

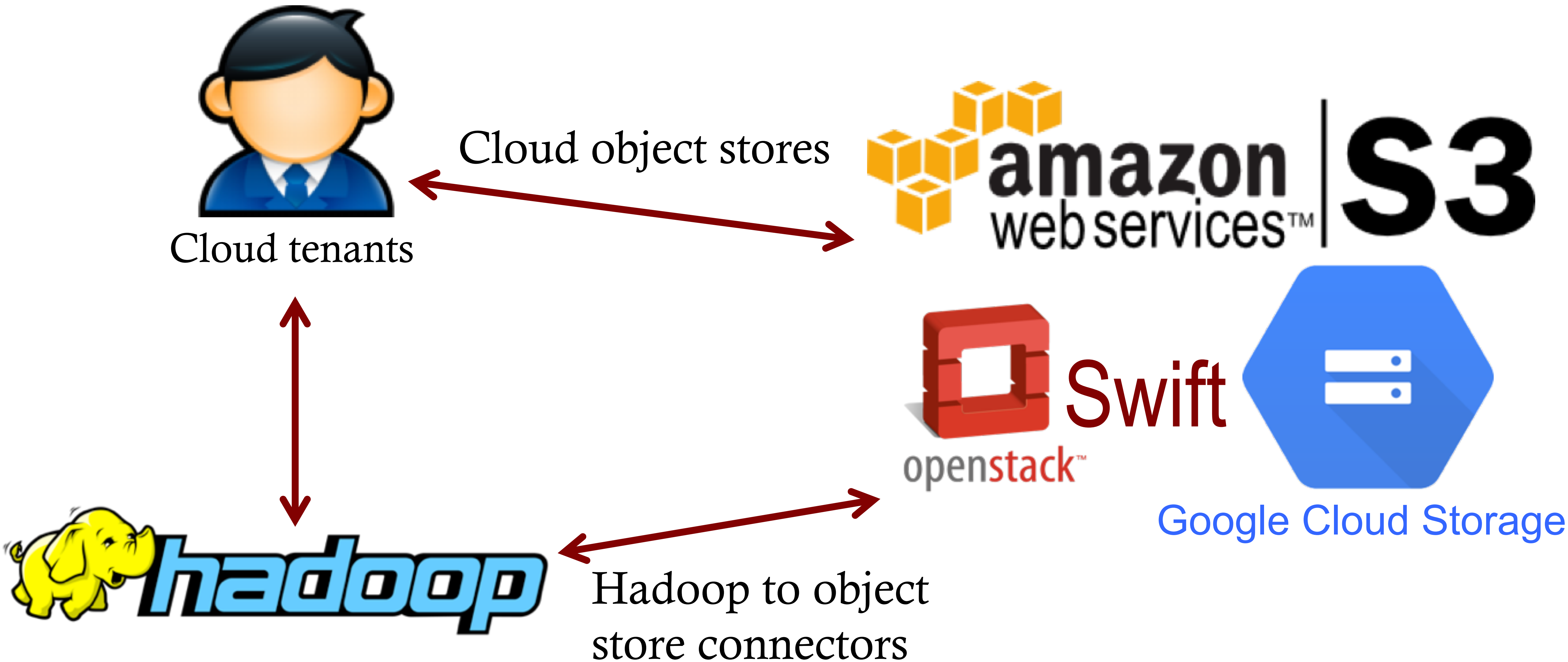
Swift

openstack™

Google Cloud Storage

*: Pricing games for hybrid object stores in the cloud: Provider vs. tenant [USENIX HotCloud'15]

Data analytics over object stores



Cloud object stores heavily rely on spinning HDDs

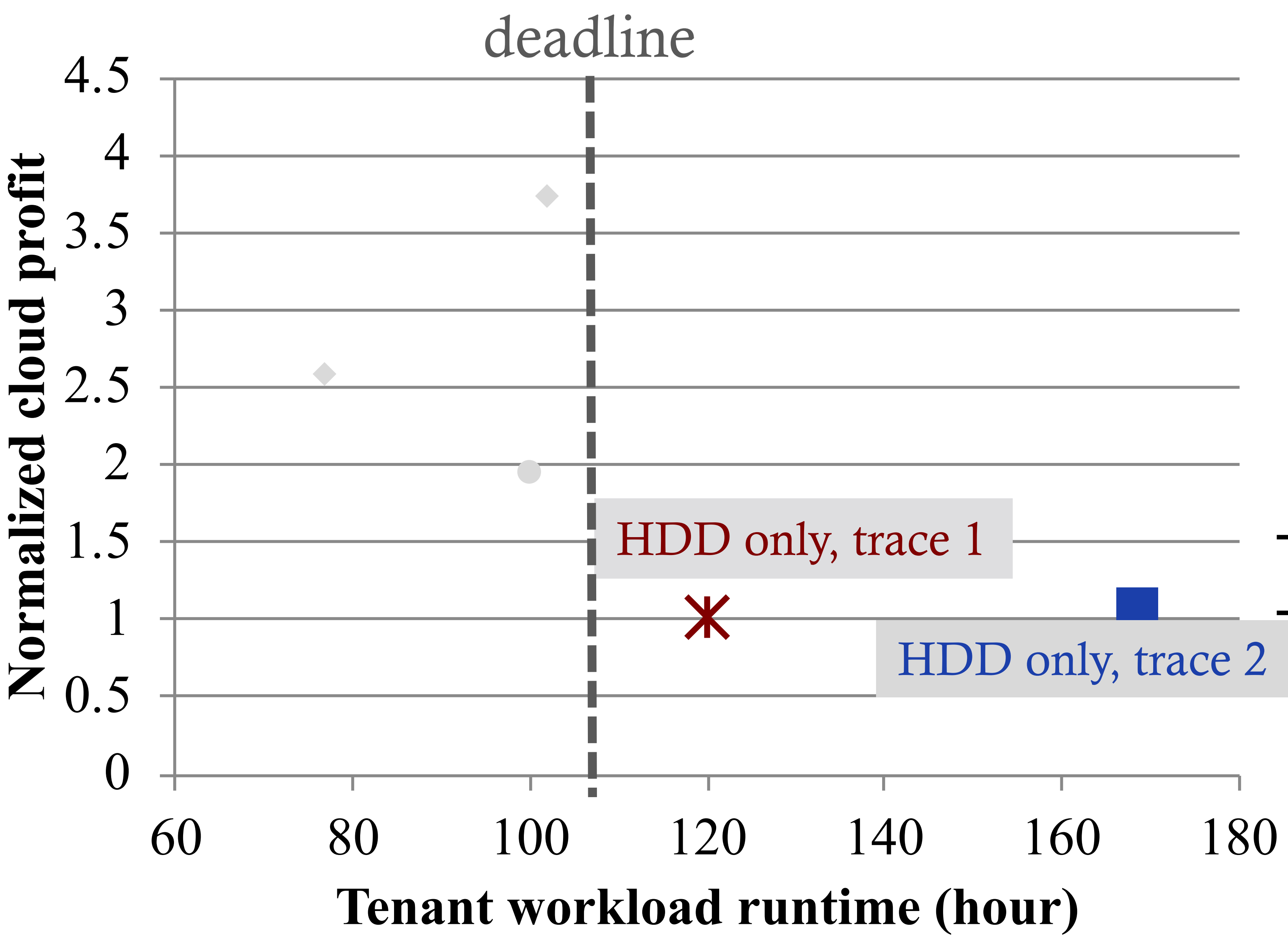
amazon web services™ | S3

Swift

openstack™

Google Cloud Storage

HDD-based object store??



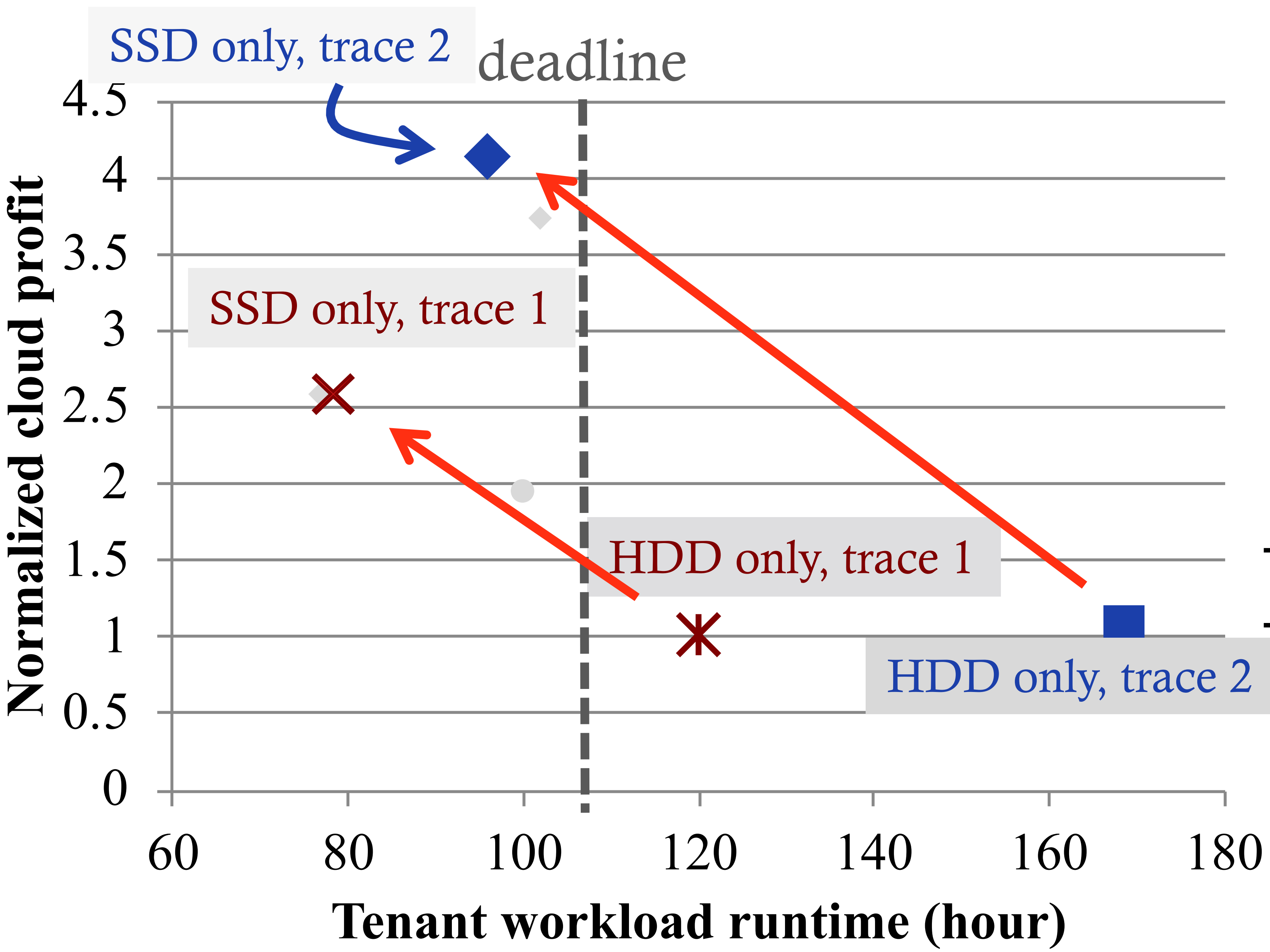
HDD pricing in EC2:
$0.0011/GB/day

Trace 1: 12TB input + 5TB output

Trace 2: 18TB input + 8TB output

Tenants not able to meet deadline
Provider gets low profit

SSD-based object store??

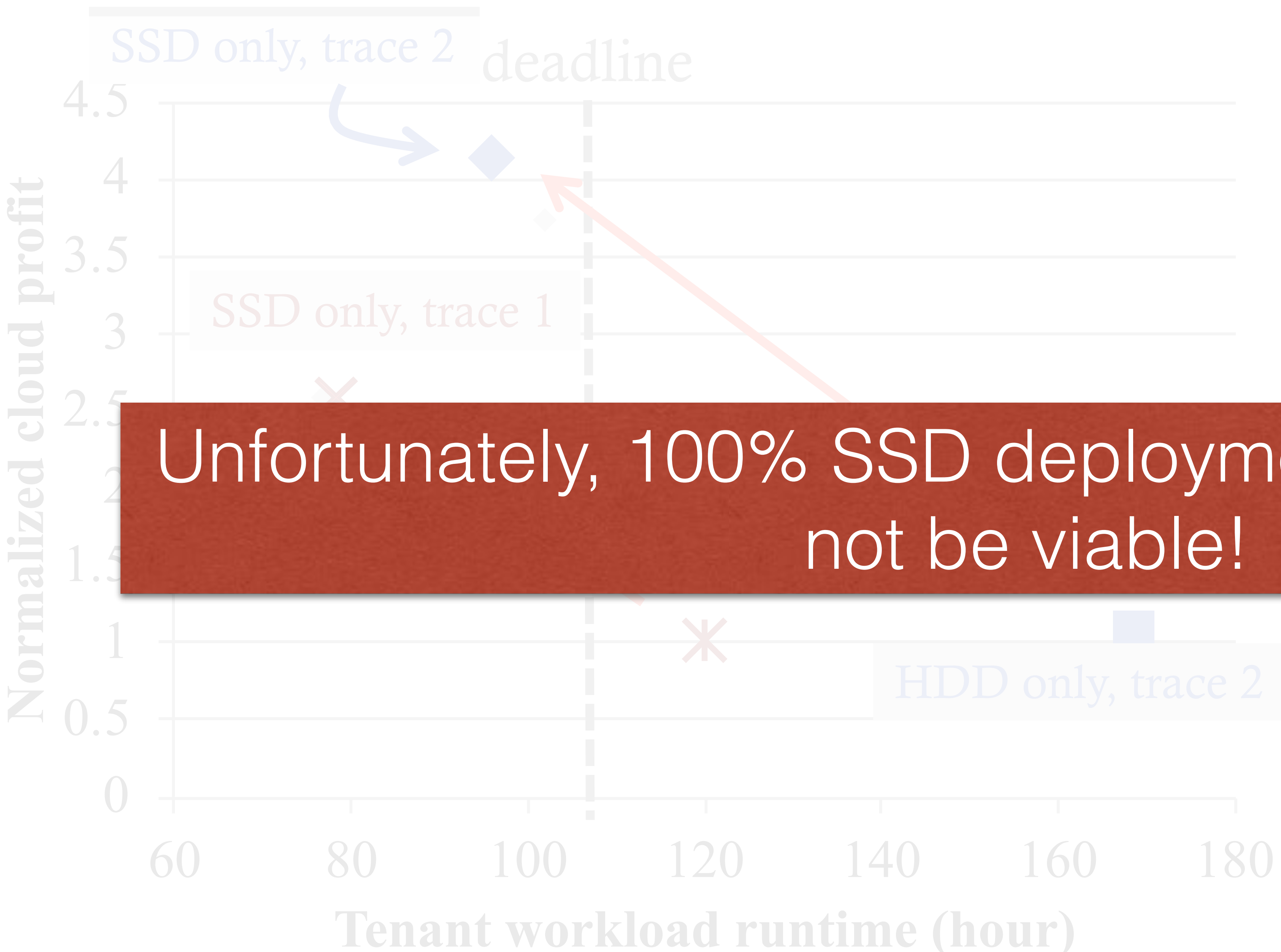


HDD pricing in EC2:
$0.0011/GB/day
SSD pricing in EC2:
$0.0044/GB/day

Trace 1: 12TB input + 5TB output
Trace 2: 18TB input + 8TB output

SSD-based object store helps meet workload deadline for tenants while increasing profit for provider

SSD-based object store??



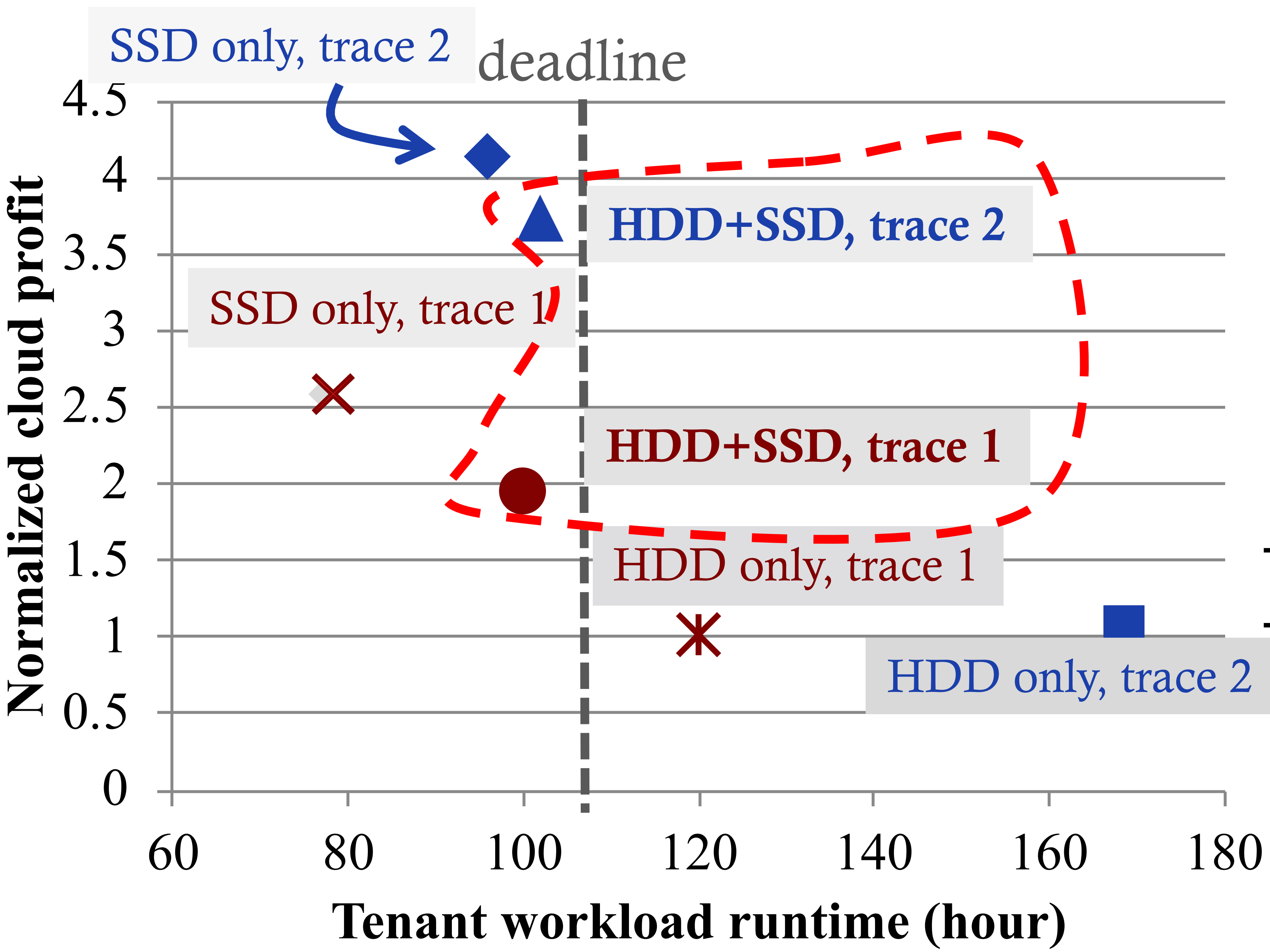
HDD pricing in EC2:
$0.0011/GB/day
SSD pricing in EC2:
day

utput
Trace 2: 18B input + 8TB output

Unfortunately, 100% SSD deployment across DC may not be viable!

SSD-based object store helps meet workload deadline for tenants while increasing profit for provider

A hybrid object store??

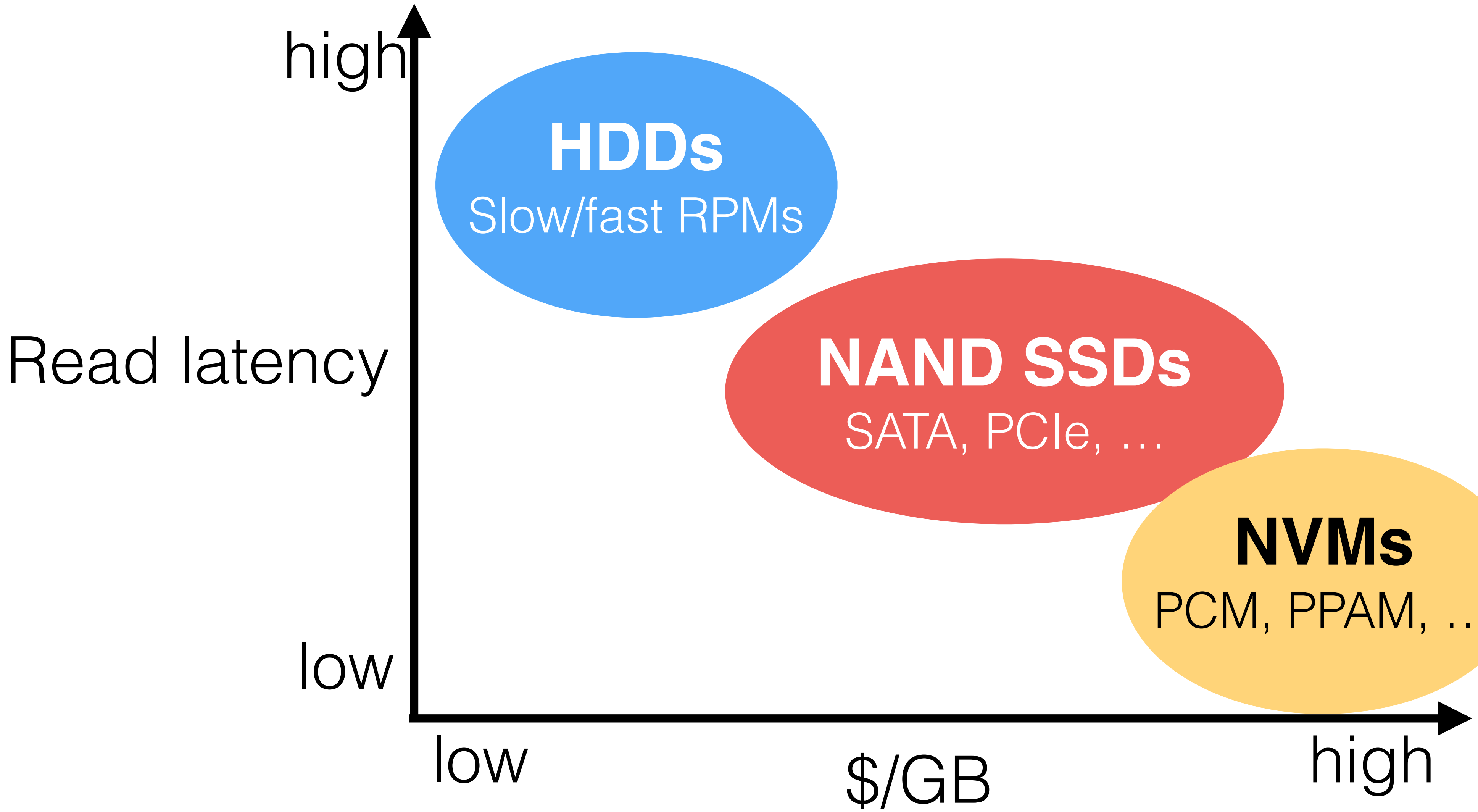


HDD pricing in EC2:
$0.0011/GB/day
SSD pricing in EC2:
$0.0044/GB/day

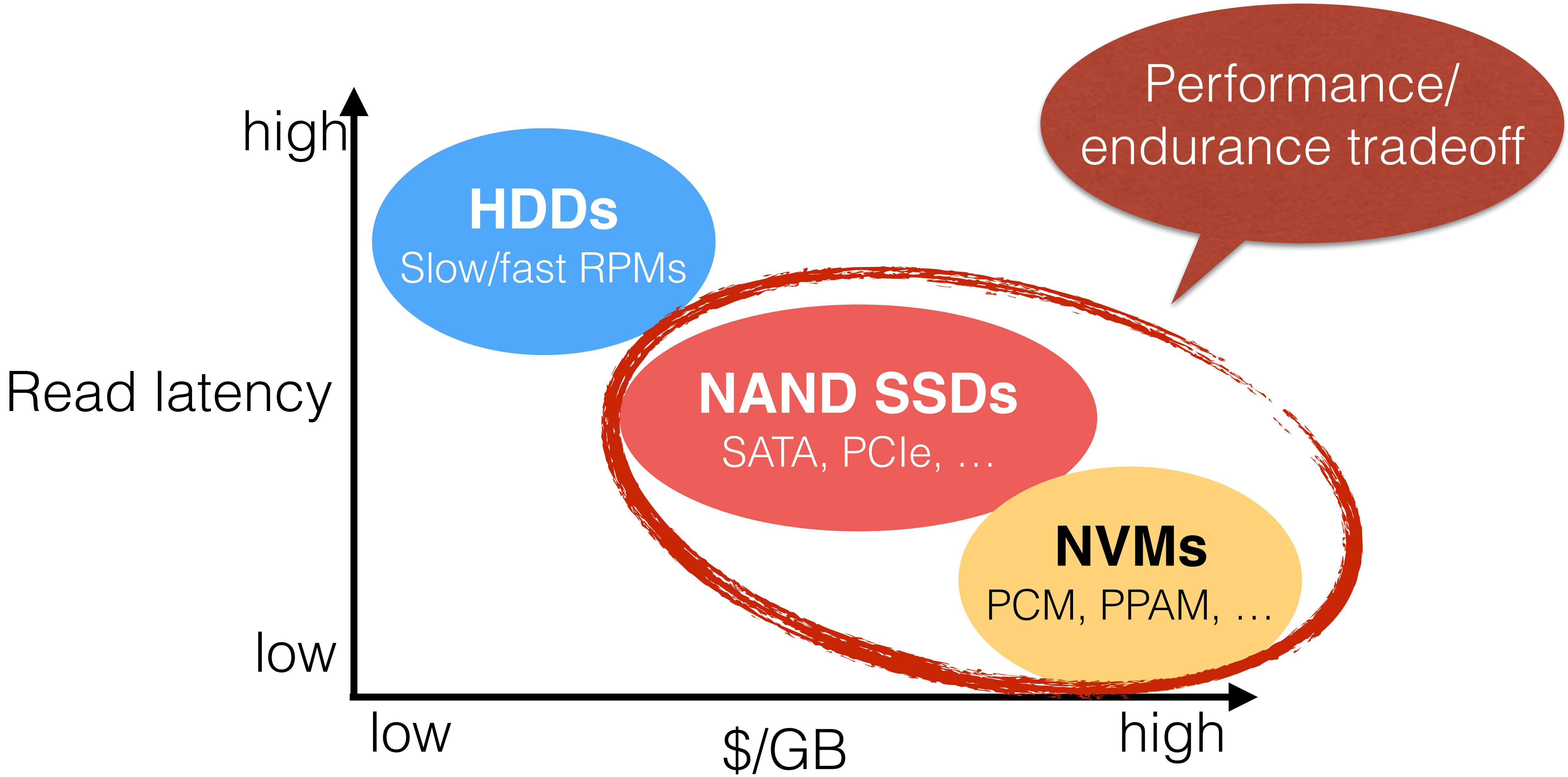
Trace 1: 12TB input + 5TB output
Trace 2: 18TB input + 8TB output

Tenants are happy since workloads meet deadline
Provider is happy with comparatively high profit

More options in a hybrid setup



More options in a hybrid setup



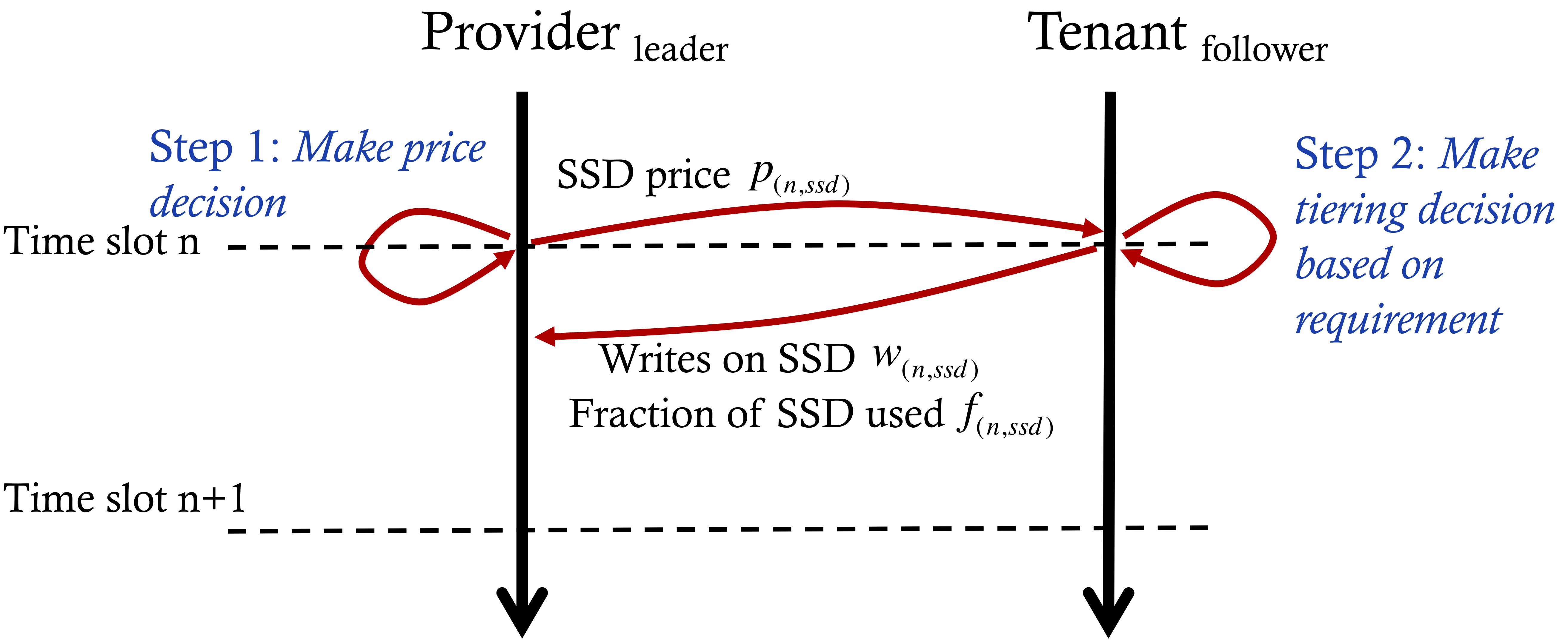
A dynamic pricing model

- Two objectives
  - Objective 1: to balance the increasing profit and SSD wearout rate
  - Objective 2: to provide incentivizing mechanism to tenants

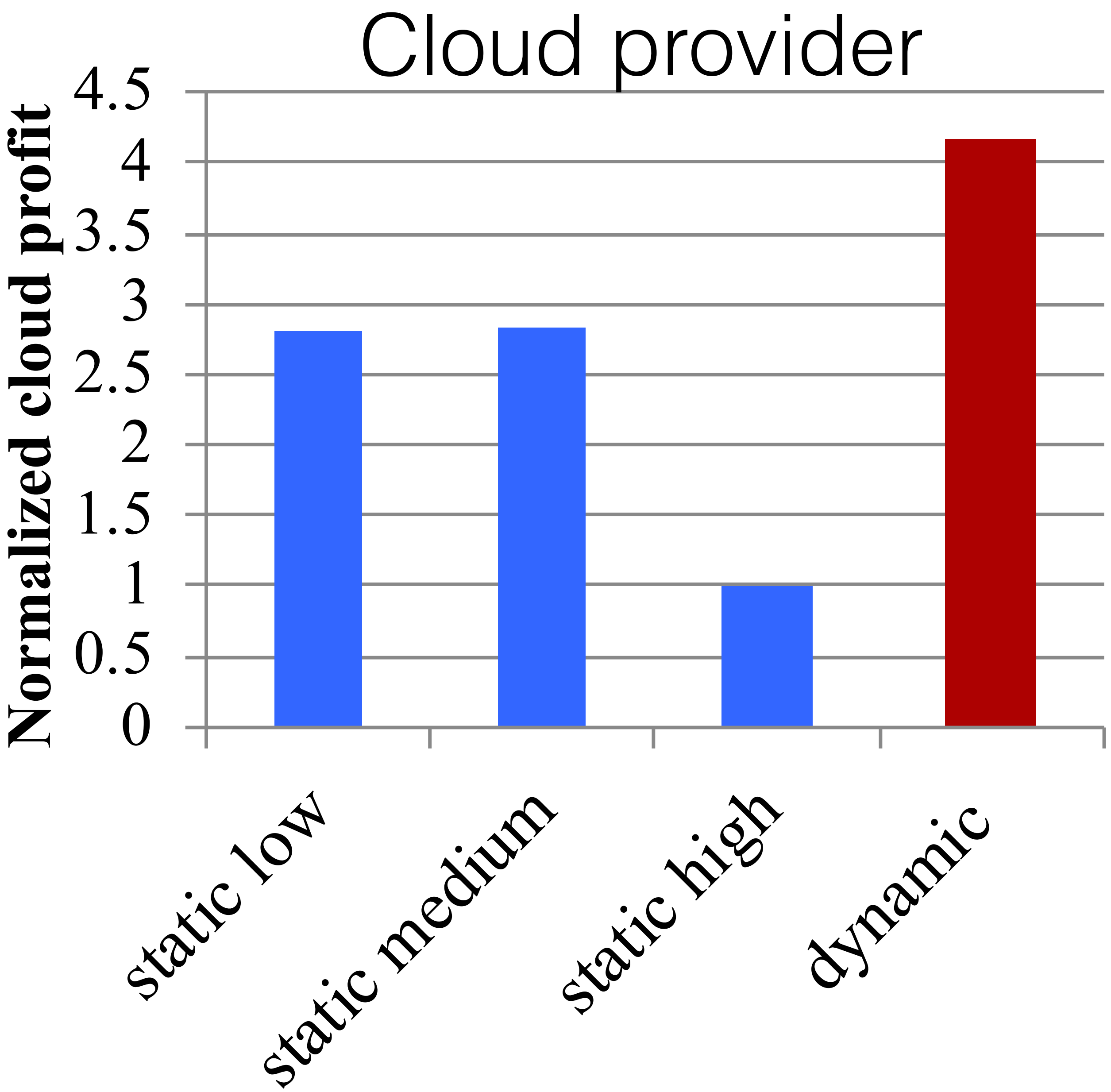
A dynamic pricing model

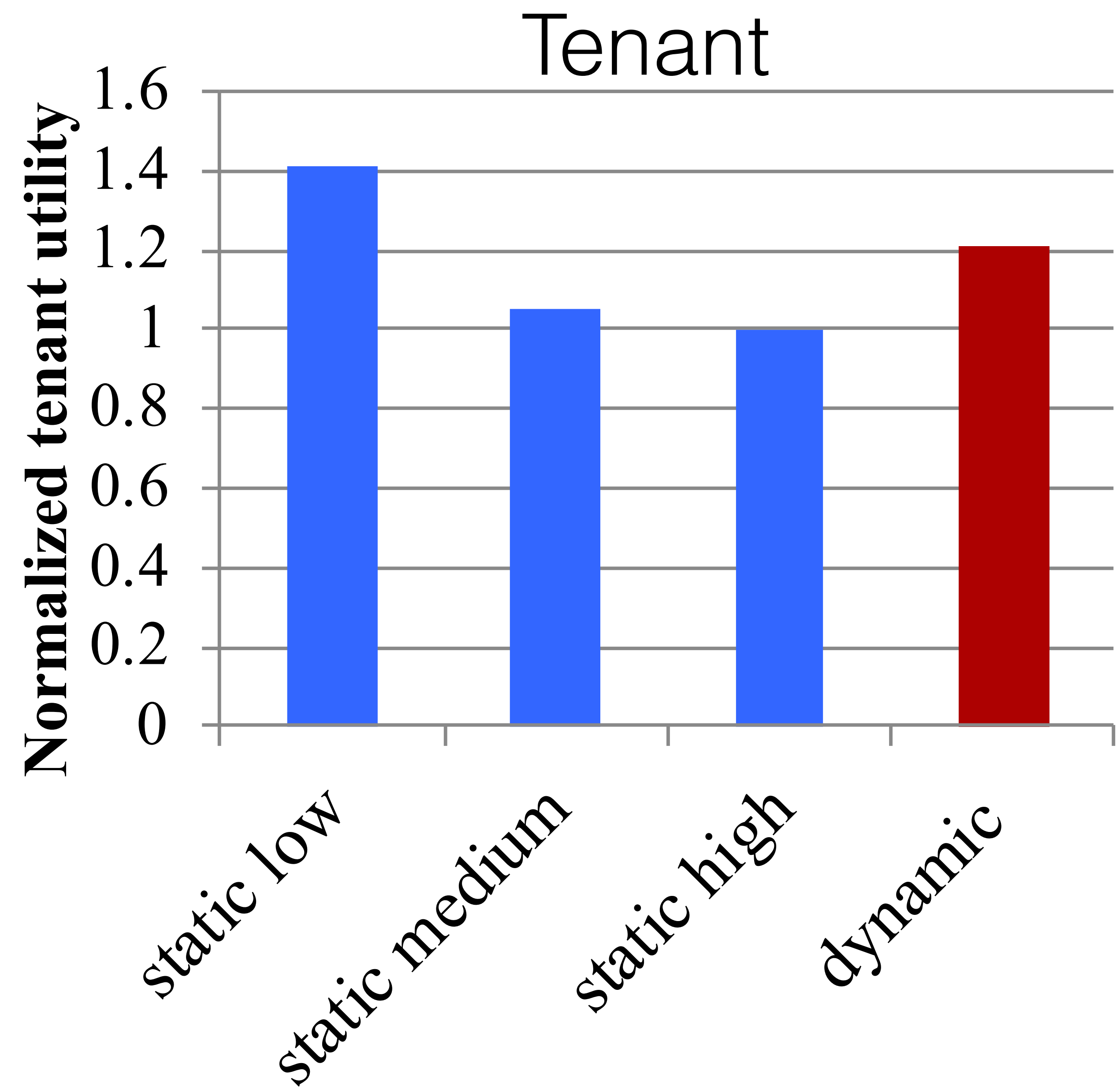
- Dynamic pricing engages both provider and tenants in a **pricing game**
  - Objectives of provider and tenants are essentially **conflicting**!

The leader/follower game



Impact of different SSD pricing



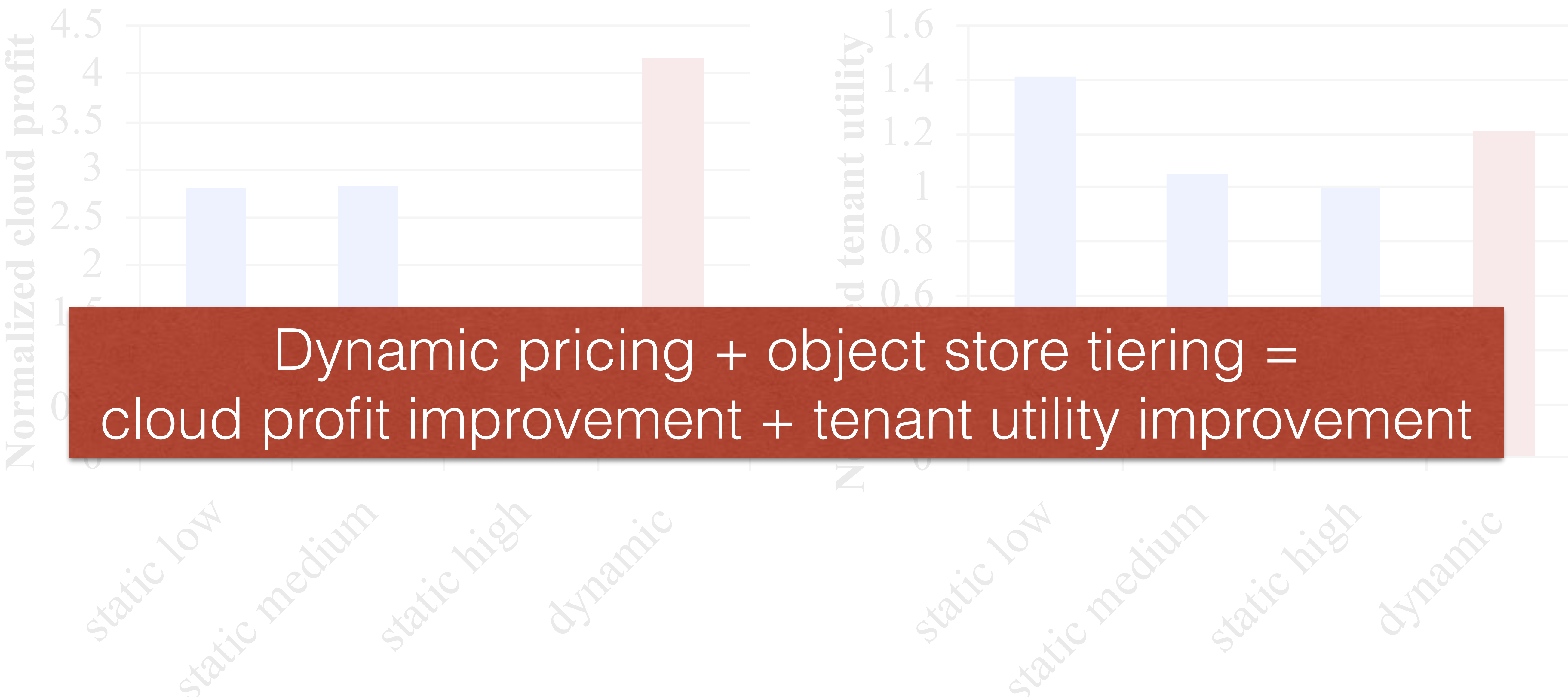


Static low:
$0.0035/GB/day

Static high:
$0.0121/GB/day

Static medium:
$0.0082/GB/day

Case study summary



Dynamic pricing + object store tiering =
cloud profit improvement + tenant utility improvement

Static low: $0.0035/GB/day

Static high: $0.0121/GB/day

Static medium: $0.0082/GB/day

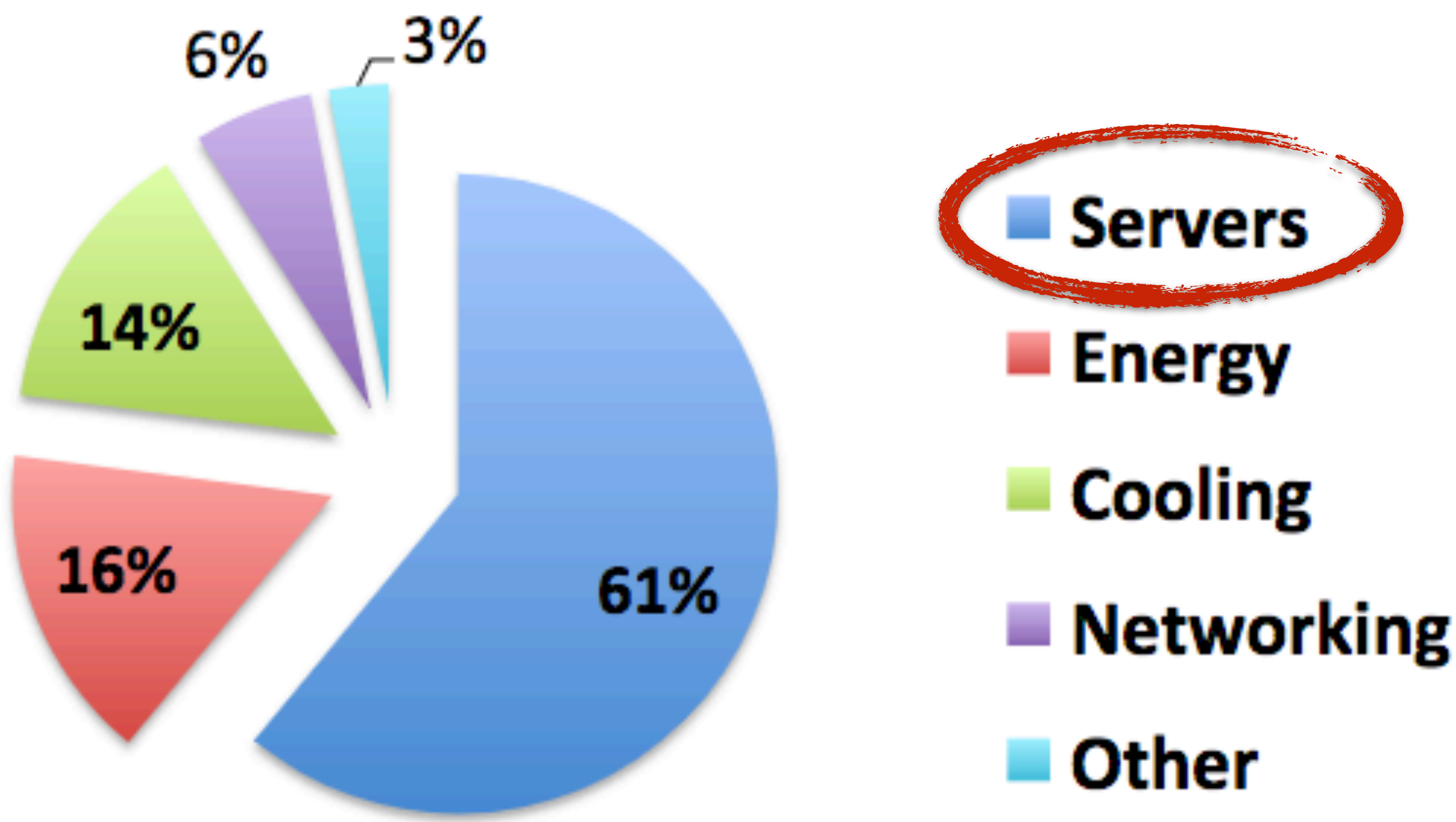
Dynamic pricing in AWS EC2

Instance type	Cost	Spin-up	Termination
Reserved	High upfront, Low per hour	Deterministic	Non-preemptible
On-demand	No upfront, High per hour	May fail	Non-preemptible
Spot	No upfront, Low per hour	May fail	Preemptible

Total cost of ownership (TCO)

- TCO = capital (CapEx) + operational (OpEx) expenses
- Provider's perspective
  - CapEx: building, generators, cooling, compute/storage/network hardware (including spares, amortized over 3—15 years)
  - OpEx: electricity (5—7c/KWh), repairs, people, WAN, insurance, …
- Tenant's perspective
  - CapEx: cost of long term leases on hardware and services
  - OpEx: pay per use cost on hardware and services, people

Provider's TCO breakdown



[Source: James Hamilton]

Hardware dominates TCO, make it cheap
Must utilize it as efficiently as possible

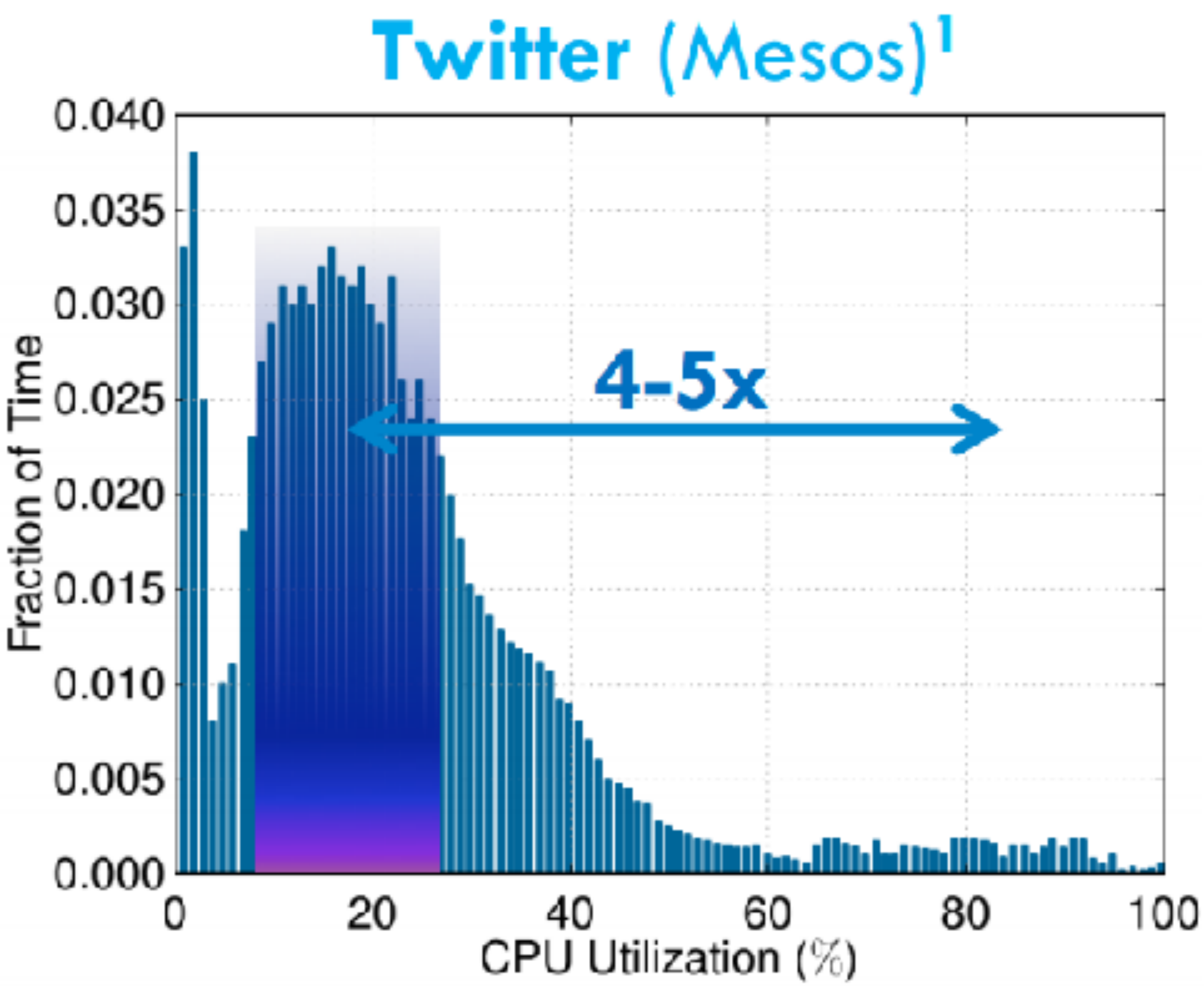
The sorry state of server utilization and the impending post-hypervisor era

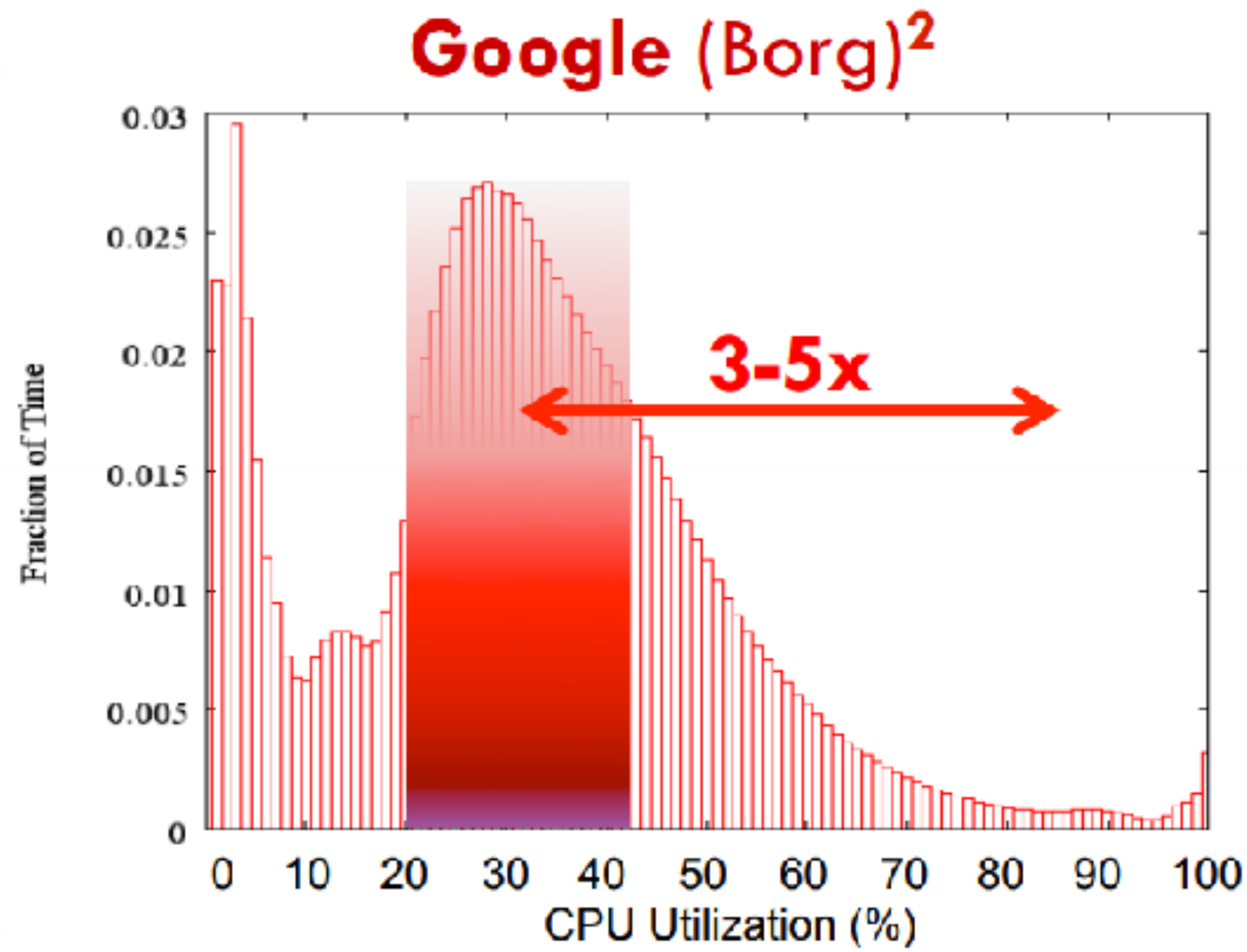
Alex Benik, Battery Ventures Nov 30, 2013 - 10:30 AM CDT

> *Me: Do you track server and CPU utilization?*
>
> ***Wall Street IT Guru**: Yes*
>
> *Me: So it's a metric you report on with other infrastructure KPIs?*
>
> ***Wall Street IT Guru**: No way, we don't put it in reports. If people knew how low it really is, we'd all get fired.*

- A McKinsey study in 2008 pegging data-center utilization at roughly 6 percent.
- A Gartner report from 2012 putting industry wide utilization rate at 12 percent.
- An Accenture paper sampling a small number on Amazon EC2 machines finding 7percent utilization over the course of a week.
- The charts and quote below from Google, which show three-month average utilization rates for 20,000 server clusters. The typical cluster on the left spent most of its time running between 20-40 percent of capacity, and the highest utilization cluster on the right reaches such heights only because it's doing batch work.

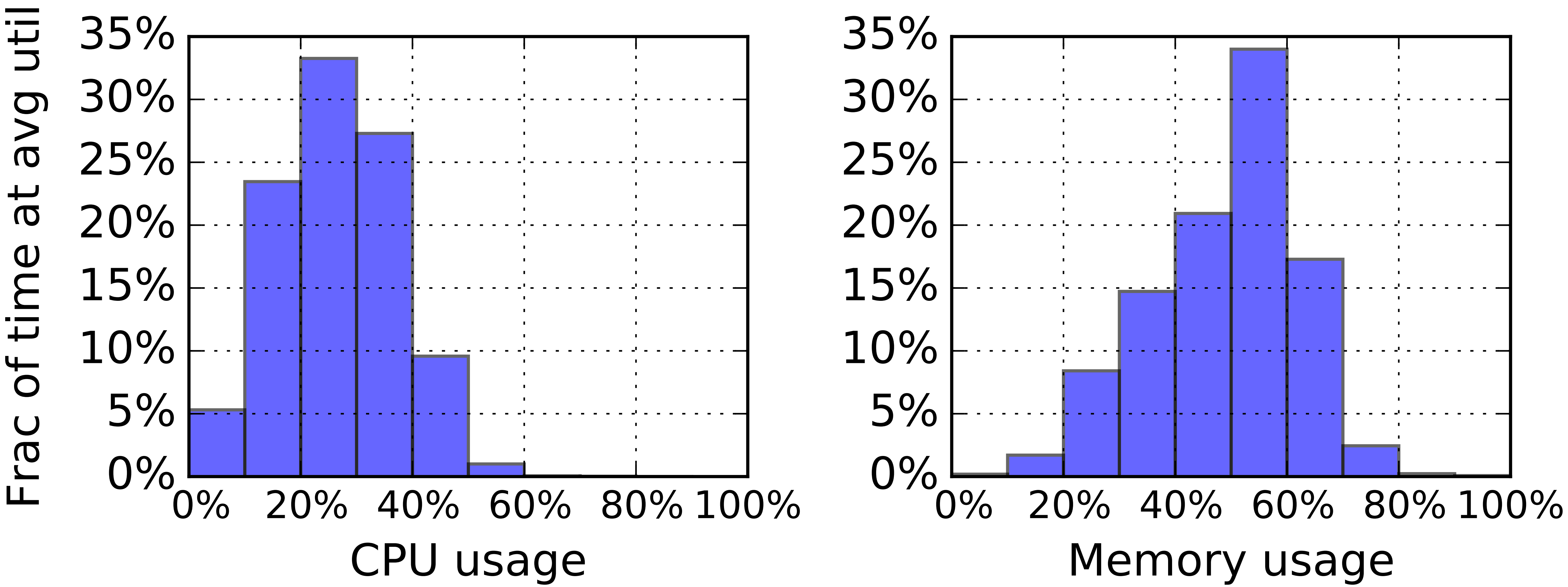
Datacenter underutilization has been a notoriously persistent problem





1: C. Delimitrou et al., Quasar: Resource-Efficient and QoS-Aware Cluster Management [ACM ASPLOS'14]
2: L. A. Barroso, U. Holzle: The Datacenter as a Computer, 2013

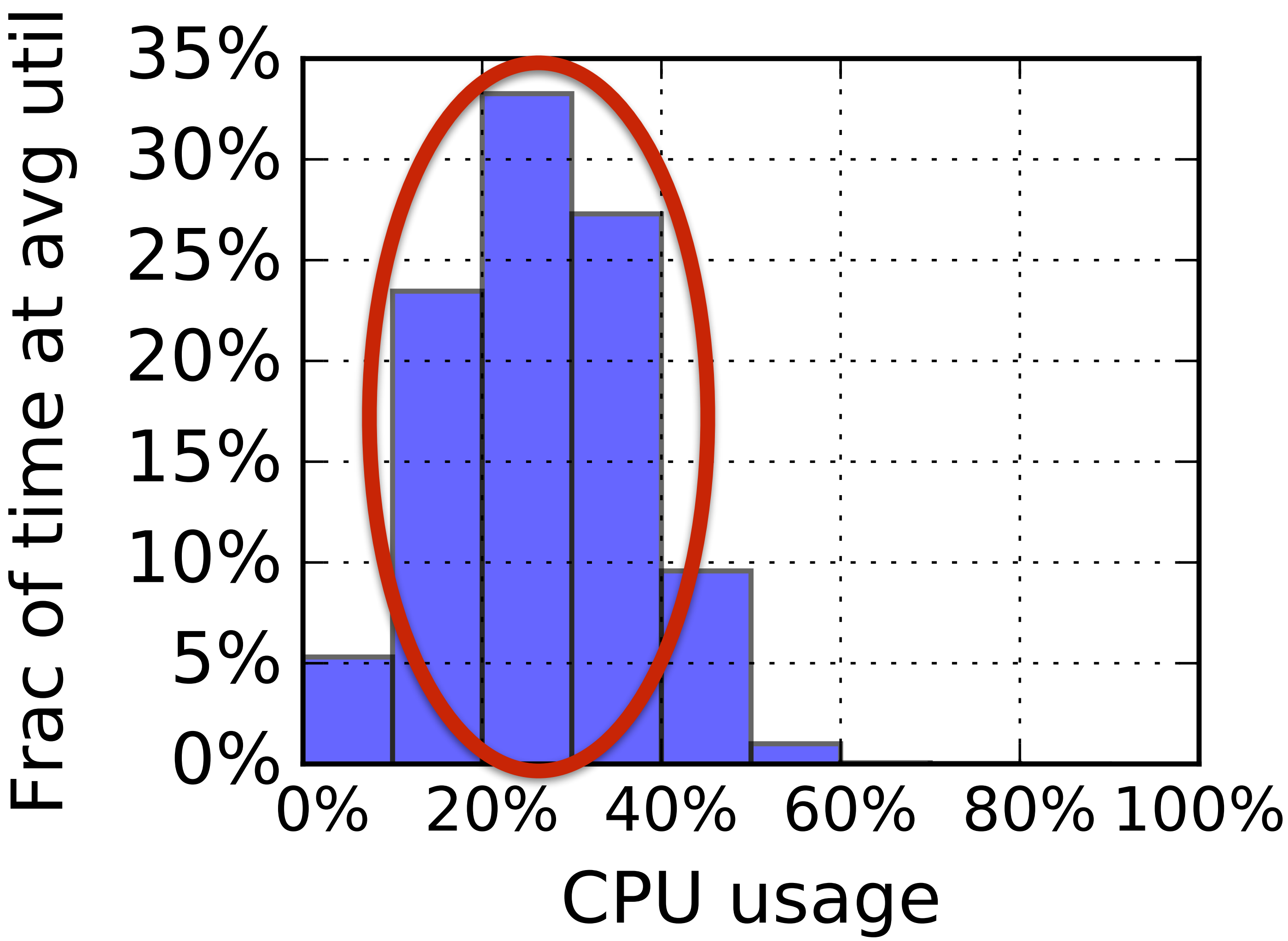
Alibaba datacenter utilization



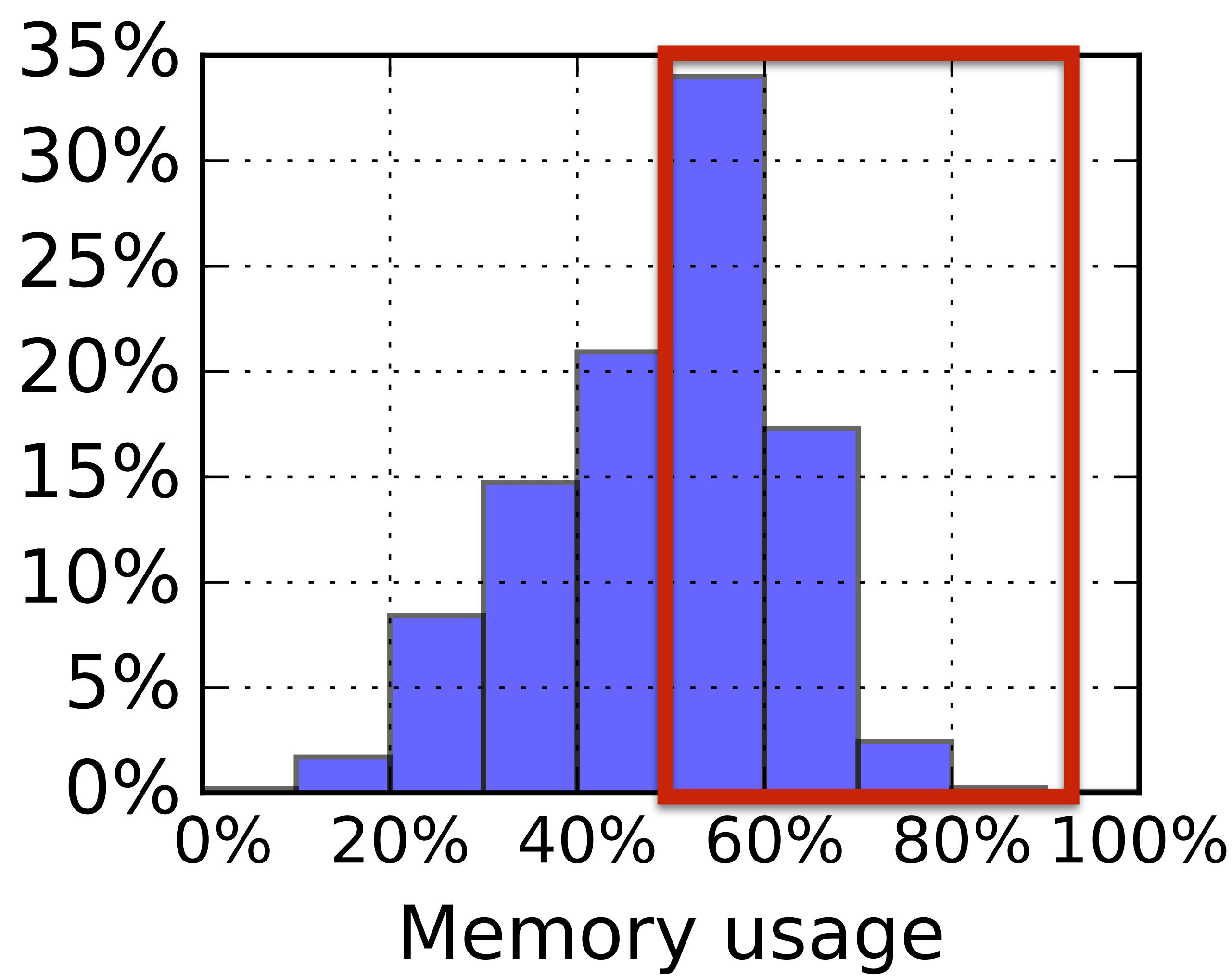
*: Characterizing Co-located Datacenter Workloads: An Alibaba Case Study [ACM APSys'18]

Alibaba datacenter utilization

> 80% time running b/w
10-30% CPU usage

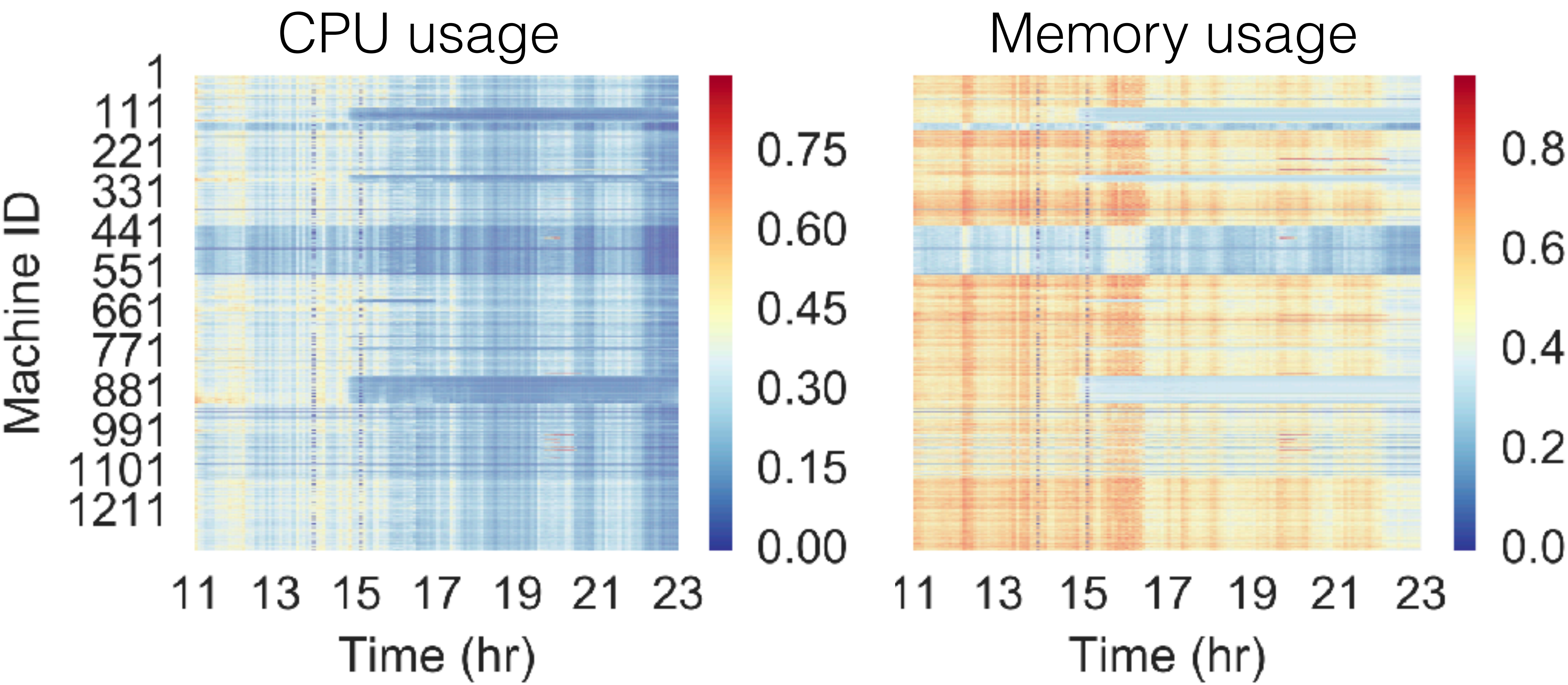


> 50% memory usage
for over 55% time



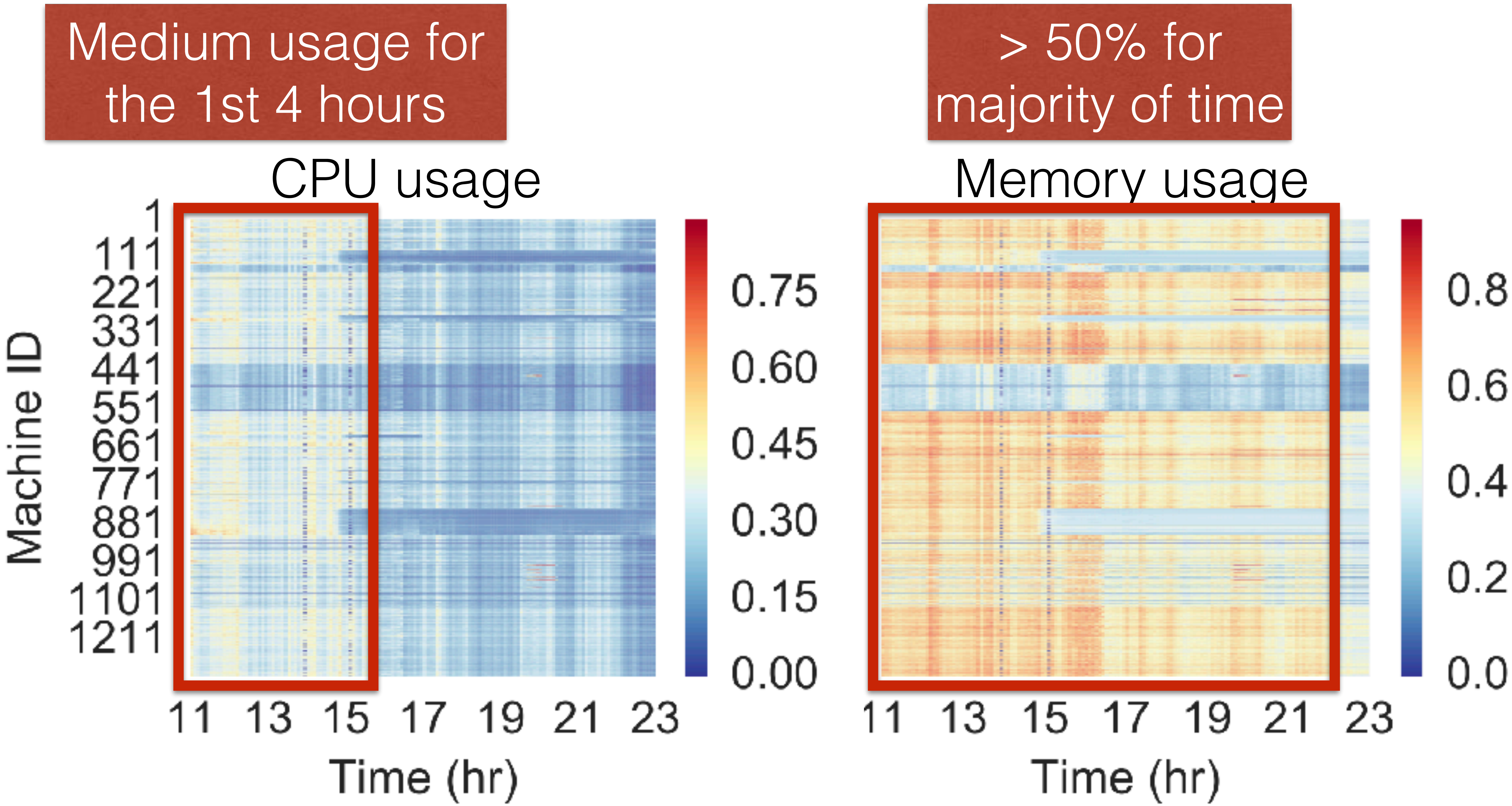
*: Characterizing Co-located Datacenter Workloads: An Alibaba Case Study [ACM APSys'18]

Alibaba datacenter util heatmap



*: Characterizing Co-located Datacenter Workloads: An Alibaba Case Study [ACM APSys'18]

Alibaba datacenter util heatmap

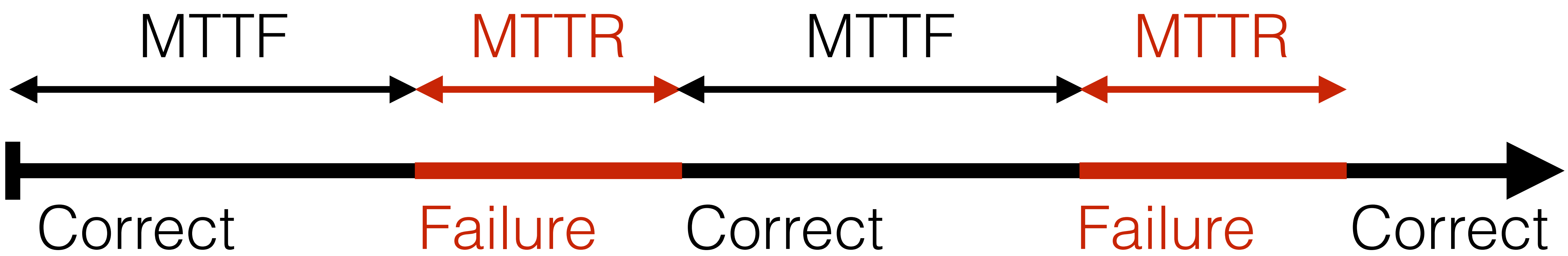


*: Characterizing Co-located Datacenter Workloads: An Alibaba Case Study [ACM APSys'18]

Datacenter reliability

- Failure in time (FIT)
  - Failures per billion hours of operation = 10^9 /MTTF
- Mean time to failure (MTTF)
  - Time to produce first incorrect output
- Mean time to repair (MTTR)
  - Time to detect and repair a failure

Service availability



Steady state availability = MTTF / (MTTF+MTTR)

* Picture credit: Gunawi et al., Why does the cloud stop computing? [ACM SoCC'16]

Yearly datacenter flakiness

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

~5 racks go wonky (40–80 machines see 50% packet loss)

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

~1000 individual machine failures (2—4% failure rate, machines crash at least twice)

~thousands of hard drive failures (1—5% of all disks will die)

Key availability techniques

Technique	Performance	Availability
Replication	√	√
Partitioning (sharding)	√	√
Load balancing	√	
Watchdog timers		√
Integrity checks		√
Eventual consistency	√	√

Make apps do something reasonable when not all is right

Better to give users limited functionality than an error page

Example: AWS S3 outage (Feb 2017)

Data Centre ▸ Cloud

AWS's S3 outage was so bad Amazon couldn't get into its own dashboard to warn the world

Websites, apps, security cams, IoT gear knackered

By Shaun Nichols in San Francisco 1 Mar 2017 at 03:00 122 SHARE ▼

By Brandon Butler, Senior Editor, Network World | FEB 28, 2017 10:59 AM PT

Cloud Chronicles is written by Network World Writer Brandon Butler, who tracks the cloud computing industry.

Amazon's S3 cloud storage service isn't working

Error rates in Simple Storage Service are bringing down sites across the web

The CAP theorem

- In distributed systems, choose **2** out of **3**
  - **Consistency**: Every read returns data from most recent writes
  - **Availability**: Every request executes & receives a (non-error) response
  - **Partition-tolerance**: The system continues to function when network partitions occur (messages dropped or delayed)

Why is Consistency important?

- Consistency = All nodes see the same data at any time, or reads return latest written value by any client
- Take multi-client bank account for example
  - You want the updates done from one client to be visible to other clients
- When thousands of customers are looking to book a flight, all updates from any client (e.g., book a flight) should be accessible by other clients
  - Rather than mistakenly overbooking the same seat three times (eventual consistency)

Why is Availability important?

- Availability = Reads/writes complete reliably and quickly
- Measurements have shown that a 500 ms increase in latency for operations at amazon.com or at google.com can cause a 20% drop in revenue
- At Amazon, each added millisecond (ms) of latency implies a $6M yearly loss
- SLAs (Service-Level Agreement) written by providers predominantly deal with:
  - Availability percentages (X nines): 99.99% “four nines”
  - Latencies faced by clients

SLA Percentages

*: https://cloudiqtech.com/aws-sla-summary/

Availability %	Downtime/Month	Downtime/Week	Downtime/Day
90% ("one nine")	72 hours	16.8 hours	2.4 hours
95%	36 hours	8.4 hours	1.2 hours
97%	21.6 hours	5.04 hours	43.2 minutes
98%	14.4 hours	3.36 hours	28.8 minutes
99% ("two nines")	7.20 hours	1.68 hours	14.4 minutes
99.5%	3.60 hours	50.4 minutes	7.2 minutes
99.8%	86.23 minutes	20.16 minutes	2.88 minutes
99.9% ("three nines")	43.8 minutes	10.1 minutes	1.44 minutes
99.95%	21.56 minutes		43.2 seconds
99.99% ("four nines")	4.38 minutes	1.01 minutes	8.66 seconds

Useful Links Cloud Provider Service Availability https://cloudharmony.com/status

Why is Partition-tolerance important?

- Partitions can happen across datacenters when the Internet gets disconnected
  - Internet router outages
  - Under-sea cables cut
  - DNS not working
- Partitions can also occur within a datacenter
  - E.g., a rack switch outage
- Still desire system to continue functioning normally under this scenario

The CAP theorem fallout

- Since partition-tolerance is essential in today's cloud computing systems, CAP theorem implies that a system has to choose between consistency and availability

- Cassandra: Eventual (weak) consistency, availability, and partition-tolerance
- Traditional RDBMSs (relational database management systems): Strong consistency over availability under a network partitioning

Replicated distributed storage

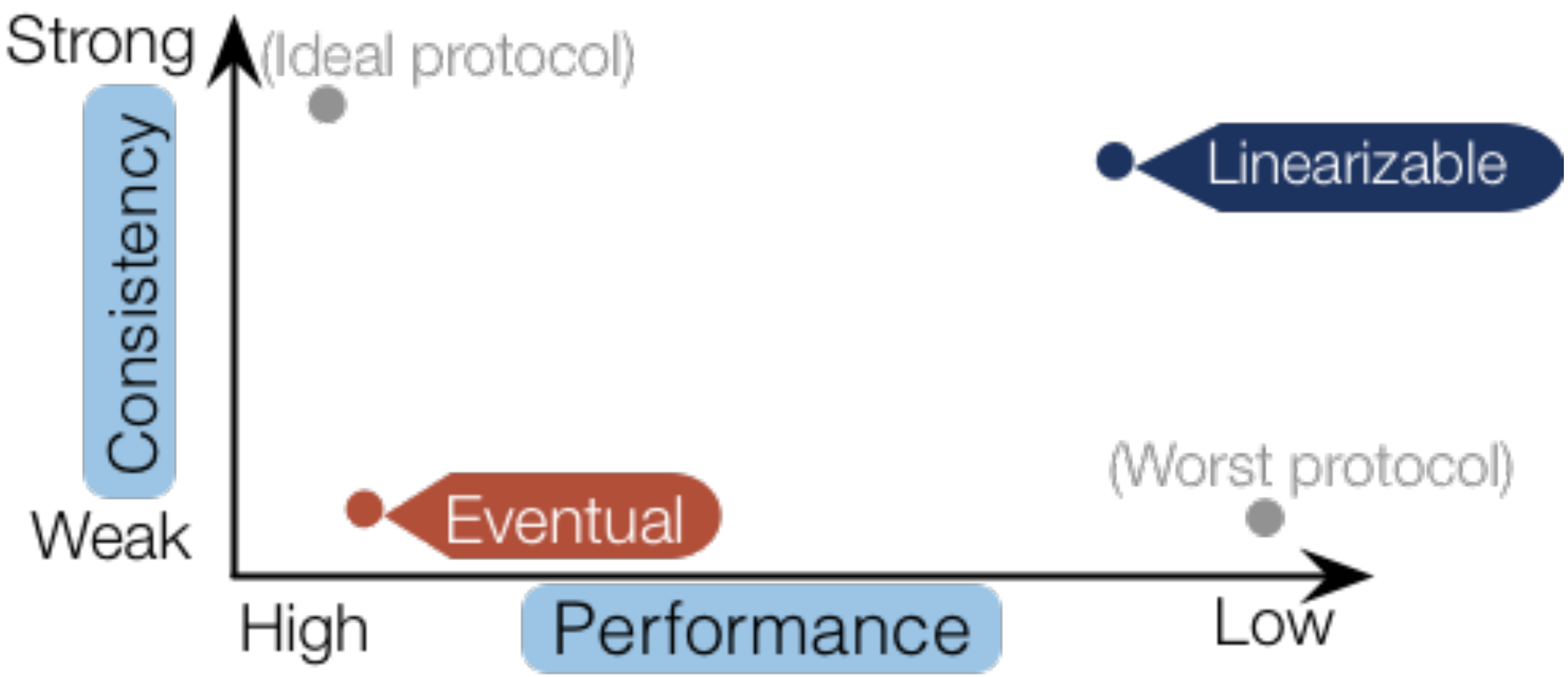
- A distributed storage system that partitions the whole namespace into shards
  - Each shard is replicated N times for fault tolerance and performance
- How close does the distributed storage system emulate a single machine in terms of read and write semantics?

Key-value stores

- RDBMS provide **ACID** guarantee of a transaction
  - **A**tomicity
  - ***Consistency***
  - **I**solation
  - **D**urability

- Key-value stores like Cassandra provide **BASE**
  - **B**asically **A**vailable **S**oft-state **E**ventual Consistency
  - Prefers Availability over Consistency

Consistency models

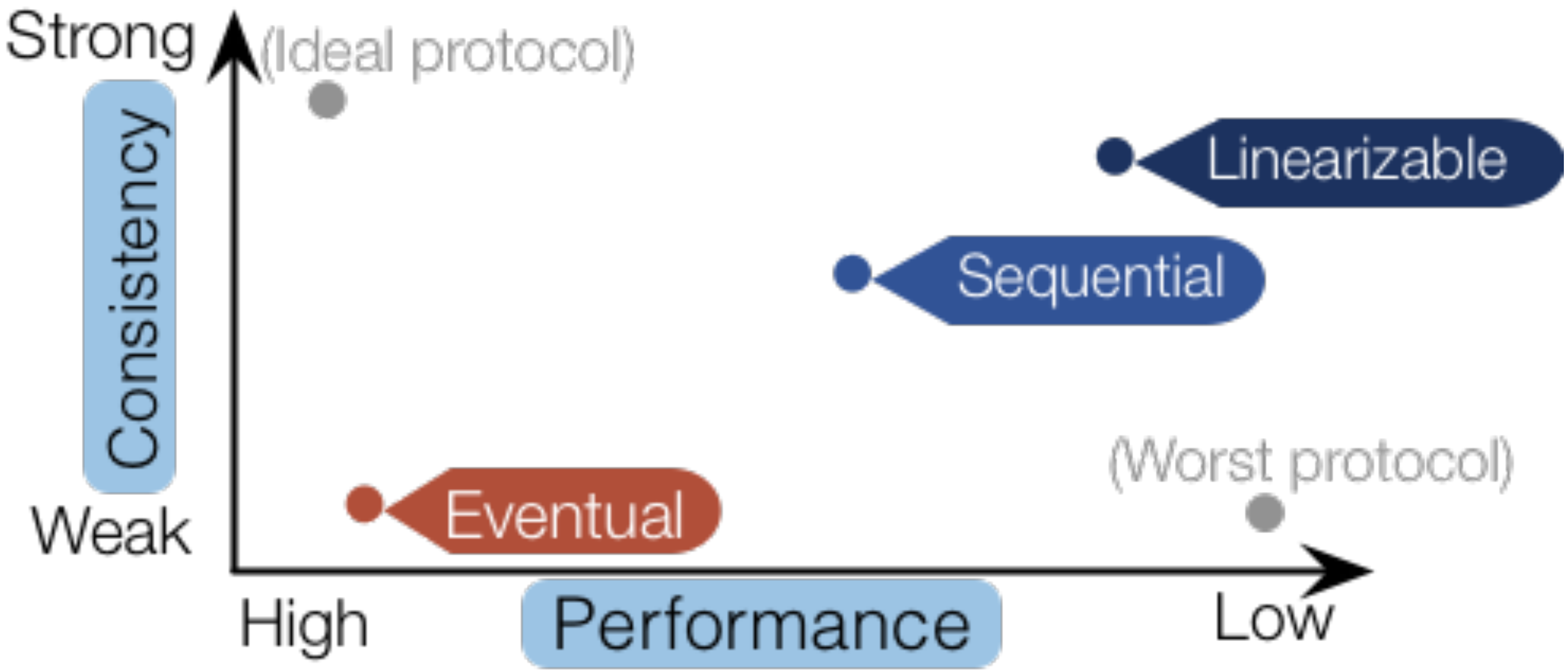


*: Incremental consistency guarantees for replicated objects [USENIX OSDI'16]

(Strong) Consistency

- **Strong** consistency (client-perceived)
  - **Linearizability**: Each update (***successful***) by a client is visible (or available) instantaneously to all other clients
- **Sequential** consistency [Lamport]
  - *"… the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program"*
  - After the fact, find a "reasonable" ordering of the operations (can re-order operations) that obeys sanity (consistency) at all clients, and across clients
  - **Sequantial = Linearizability - real-time ordering**

Consistency models

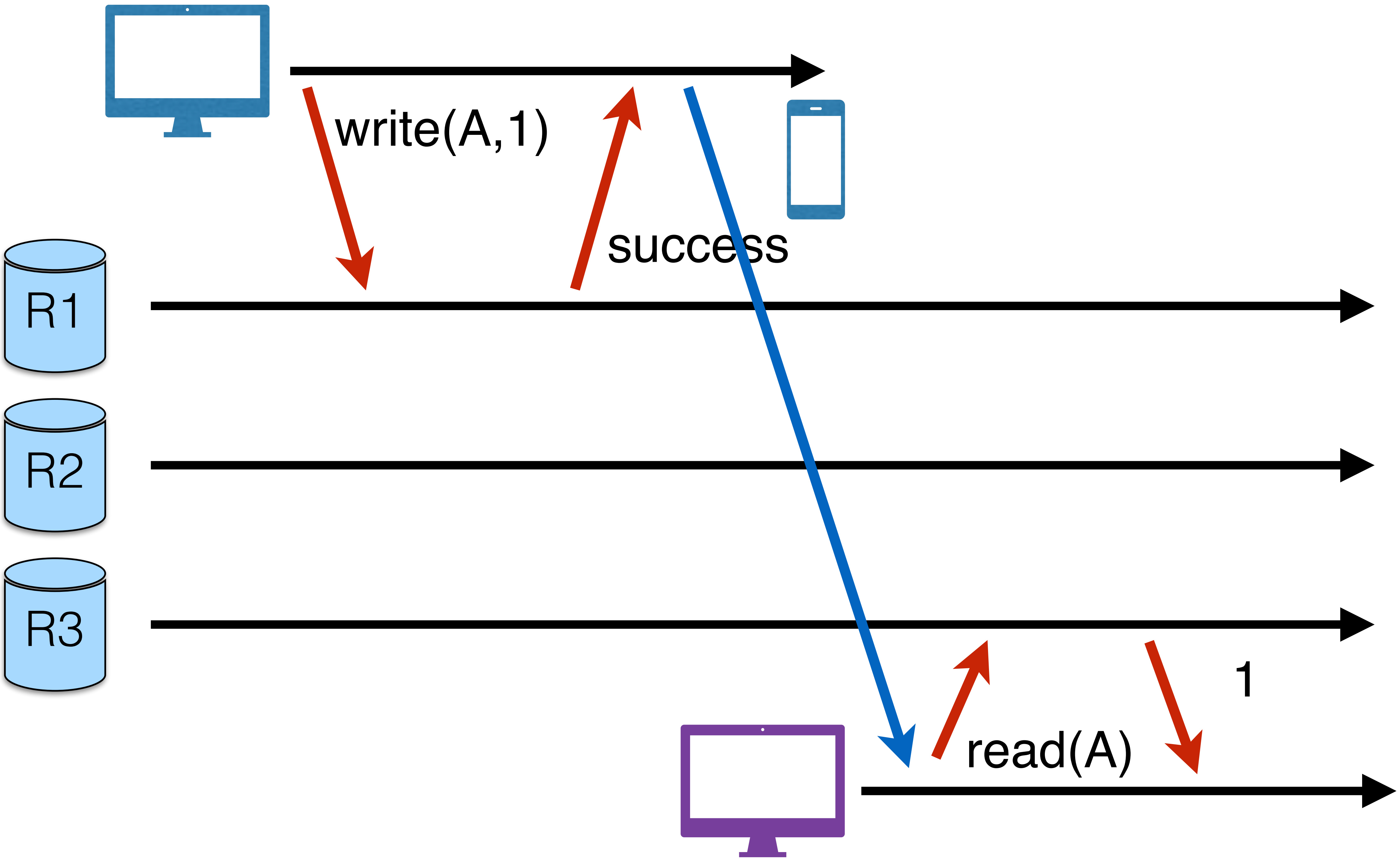


*: Incremental consistency guarantees for replicated objects [USENIX OSDI'16]

Telephone intuition

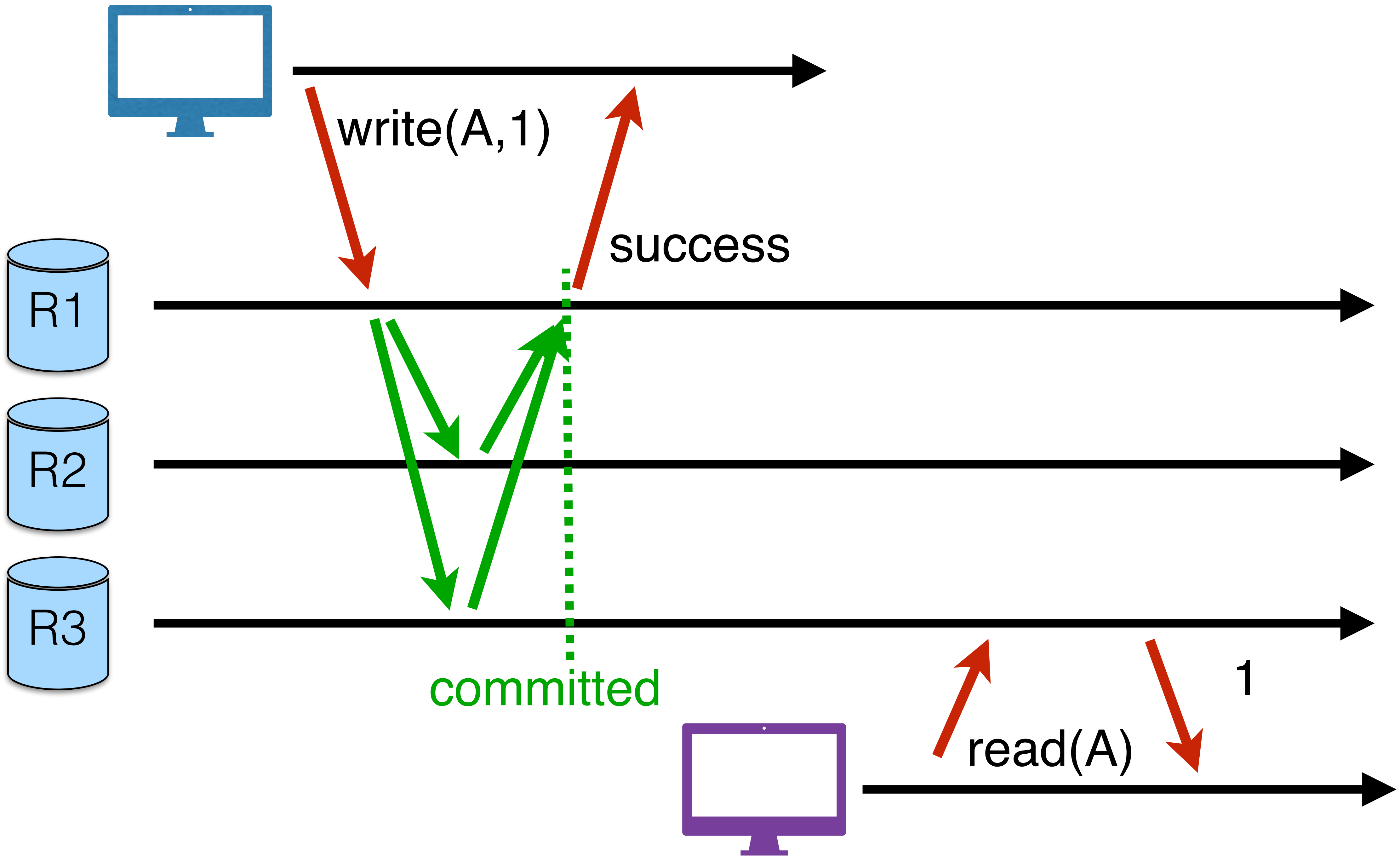
1. Alice updates Facebook post
2. Alice calls Bob on phone: “Hey check my Facebook post!”
3. Bob reads Alice’s wall, sees her post

Strong consistency?



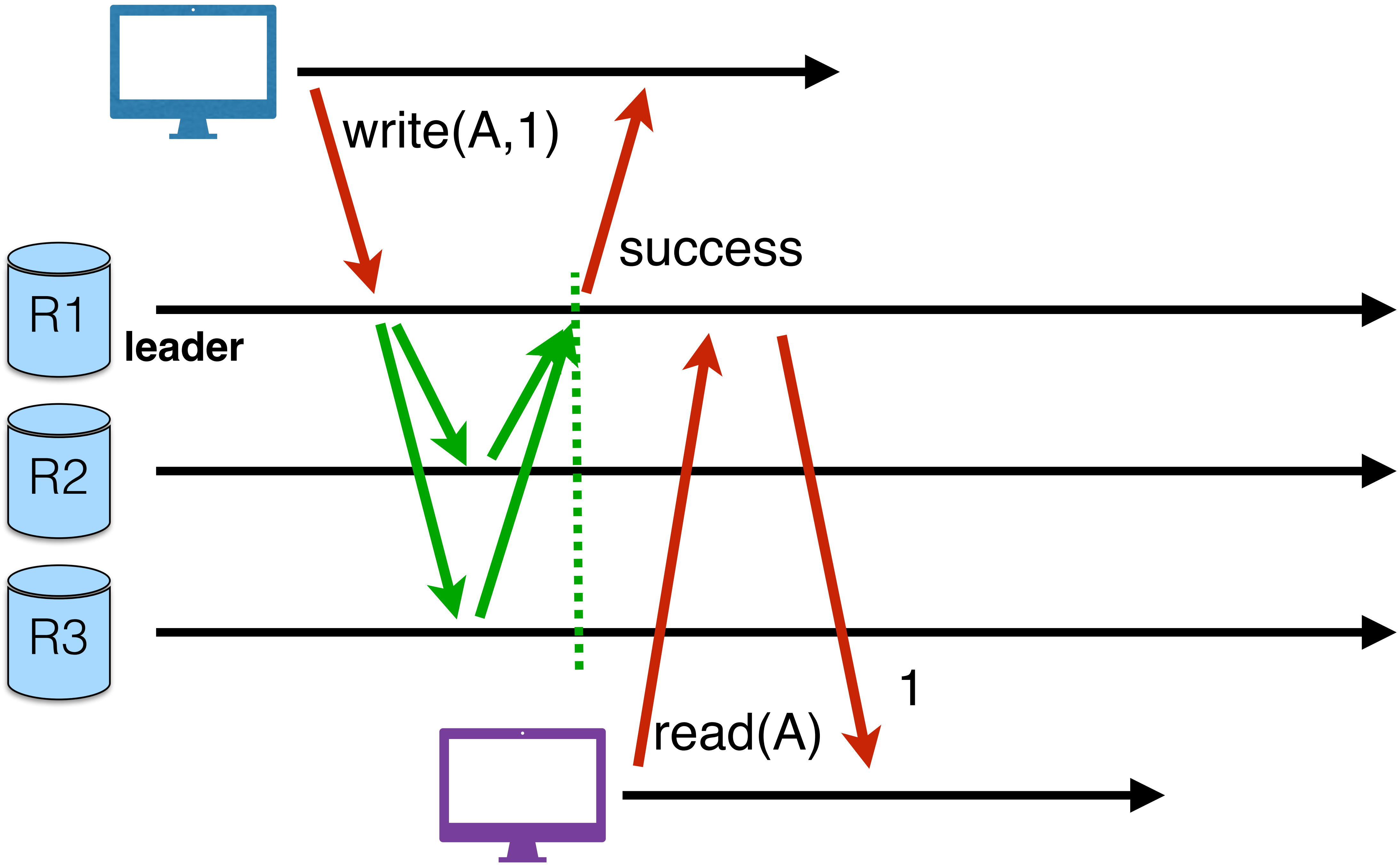
Phone call: Ensures ***happens-before*** relationship, even though “out-of-band” communication

Strong consistency? This is buggy!



Isn't sufficient to return value of third replica: It does not know precisely when op is **globally** committed

Strong consistency



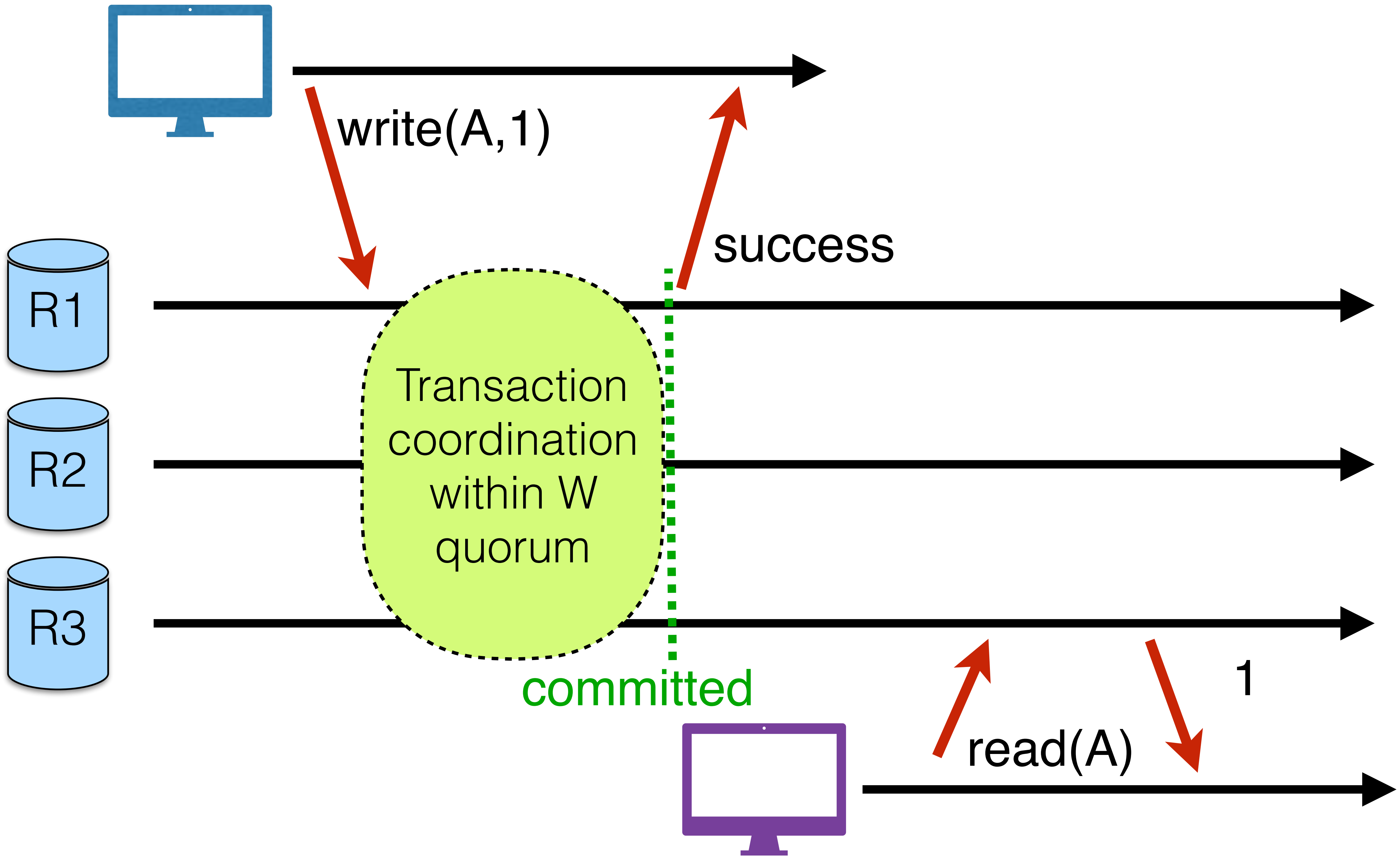
Instead, need to actually order all operations via:
(1) leader, (2) consensus

Server-side consistency

- How system understands consistency internally under the hood
  - **Quorums**

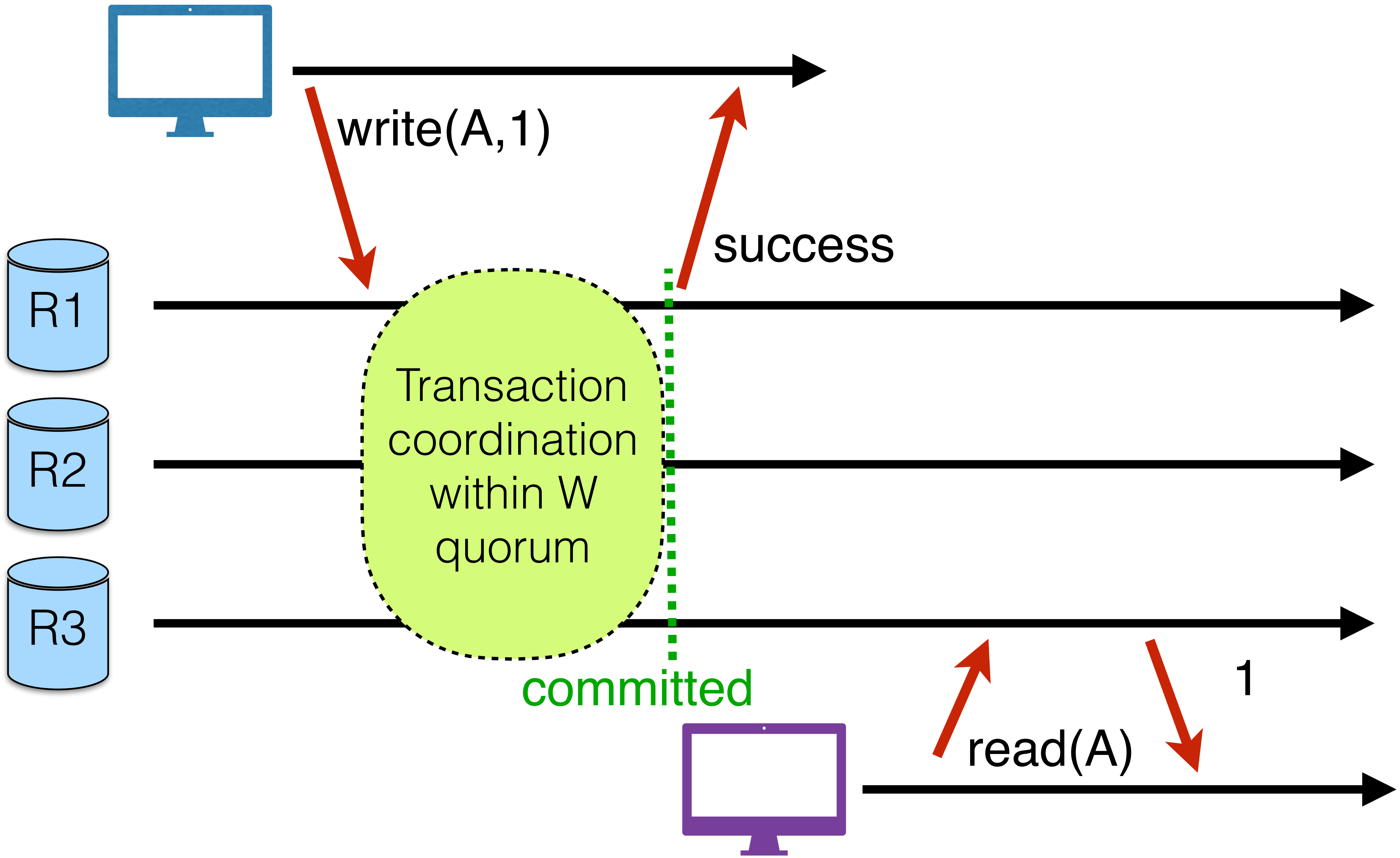
- **N**: The number of nodes that store a replica of data
- **W**: The number of replicas that need to acknowledge the receipt of the update before the update completes
- **R**: The number of replicas that are contacted when a data object is accessed through a read operation

Strong consistency with Quorum



N: 3 replicas of a data
W: 3 replicas ack before write commits
R: 1 replica needs to be contacted for a read

Strong consistency with Quorum



Understanding strong consistency: No matter on which replica the read op is performed, the outcome will be the same as it was read by any replica

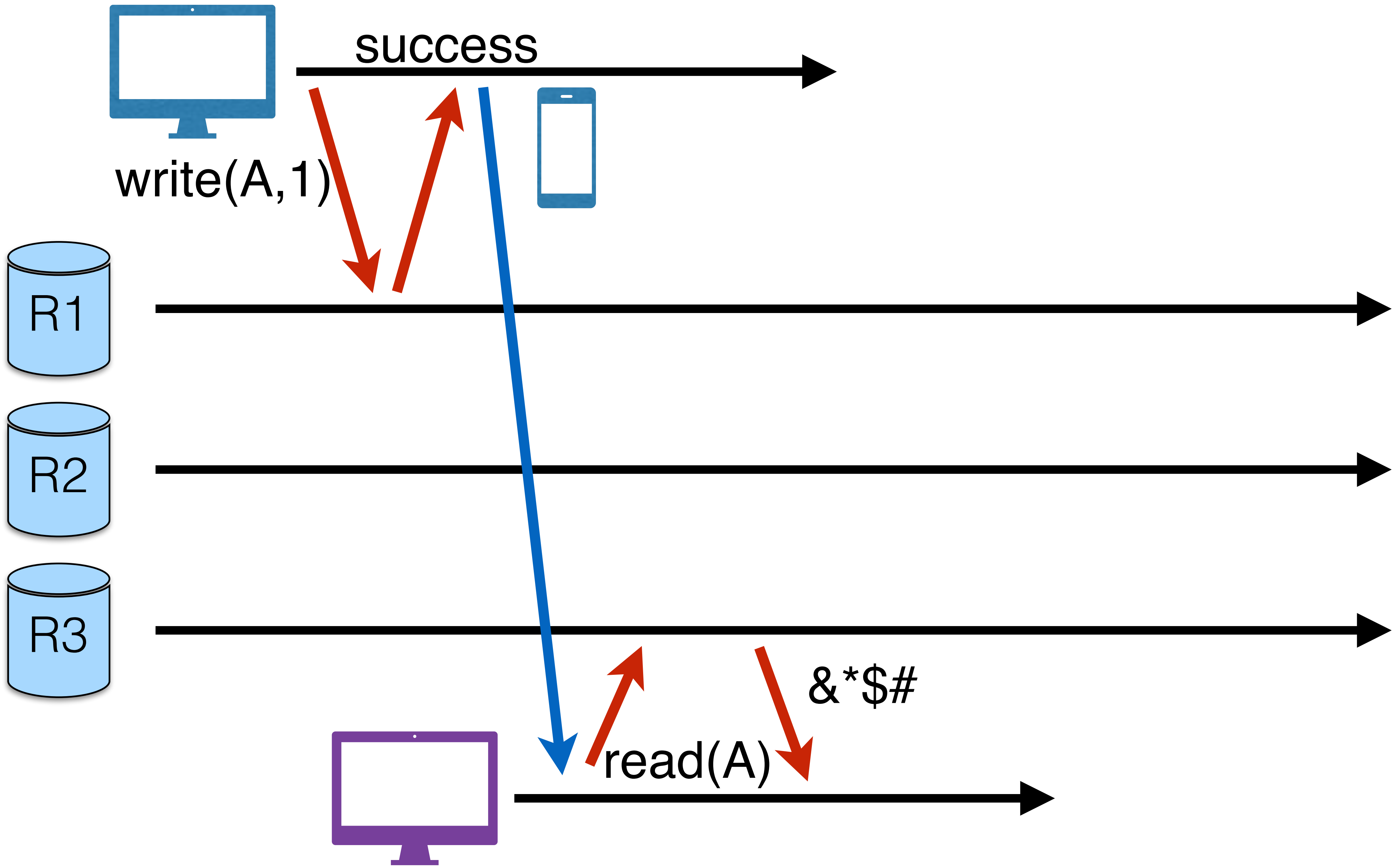
Eventual consistency

- Eventual consistency (client-perceived)
  - The storage system guarantees that if no new updates are made to the object, eventually (after the inconsistency window closes) all accesses will return the last updated value
- If writes continue, then system always tries to keep converging
  - May still return stale values to clients (e.g., if many back-to-back writes)
- But works well when there are a few periods of low writes
  - System converges quickly (eventually)

Telephone intuition

1. Alice updates Facebook post
2. Alice calls Bob on phone: “Hey check my Facebook post!”
3. Bob reads Alice’s wall, but does not see her post
4. Bob refreshes Alice’s wall, and <u>*eventually*</u> sees her post

Eventual consistency

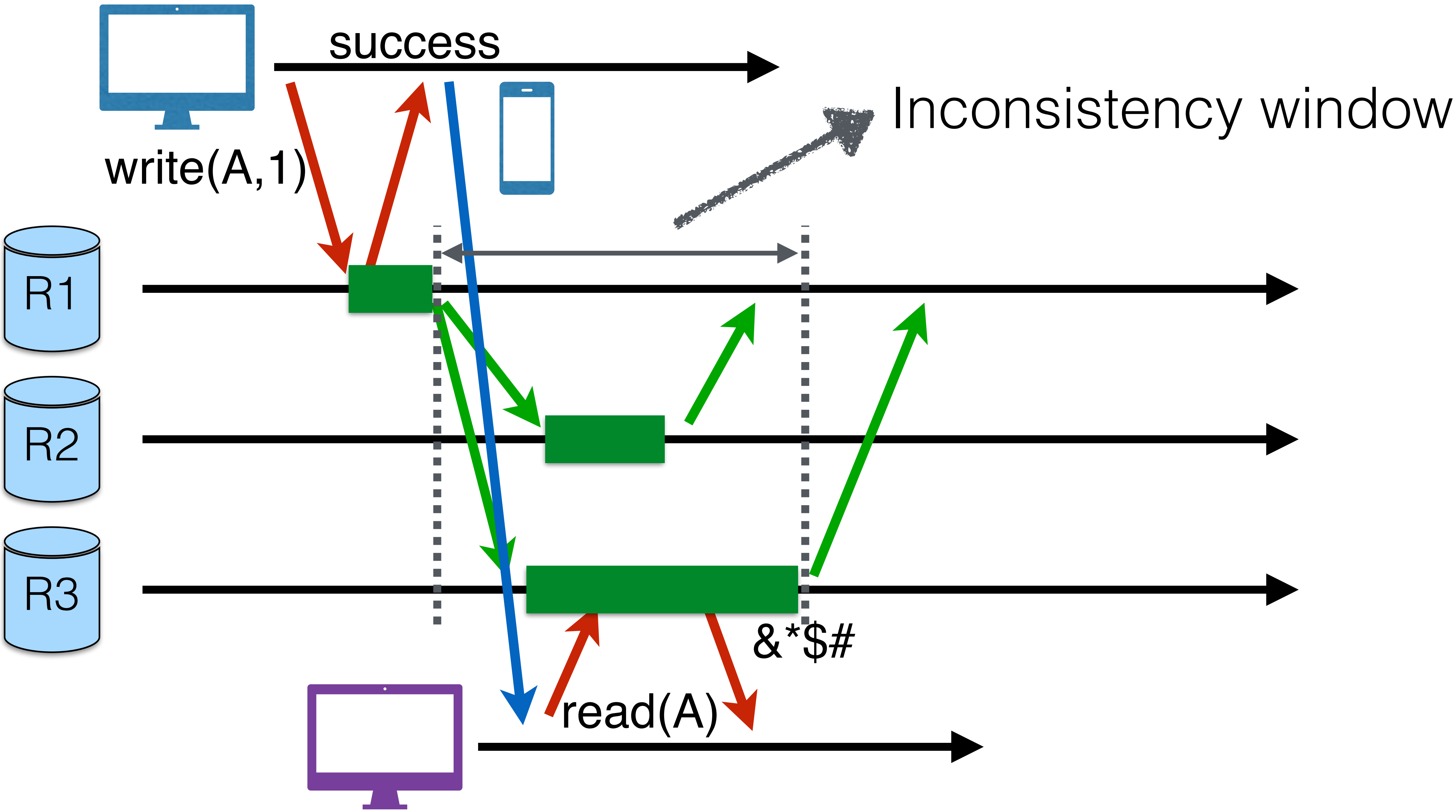


N: 3 replicas of a data

W: 1 replica ack before write commits

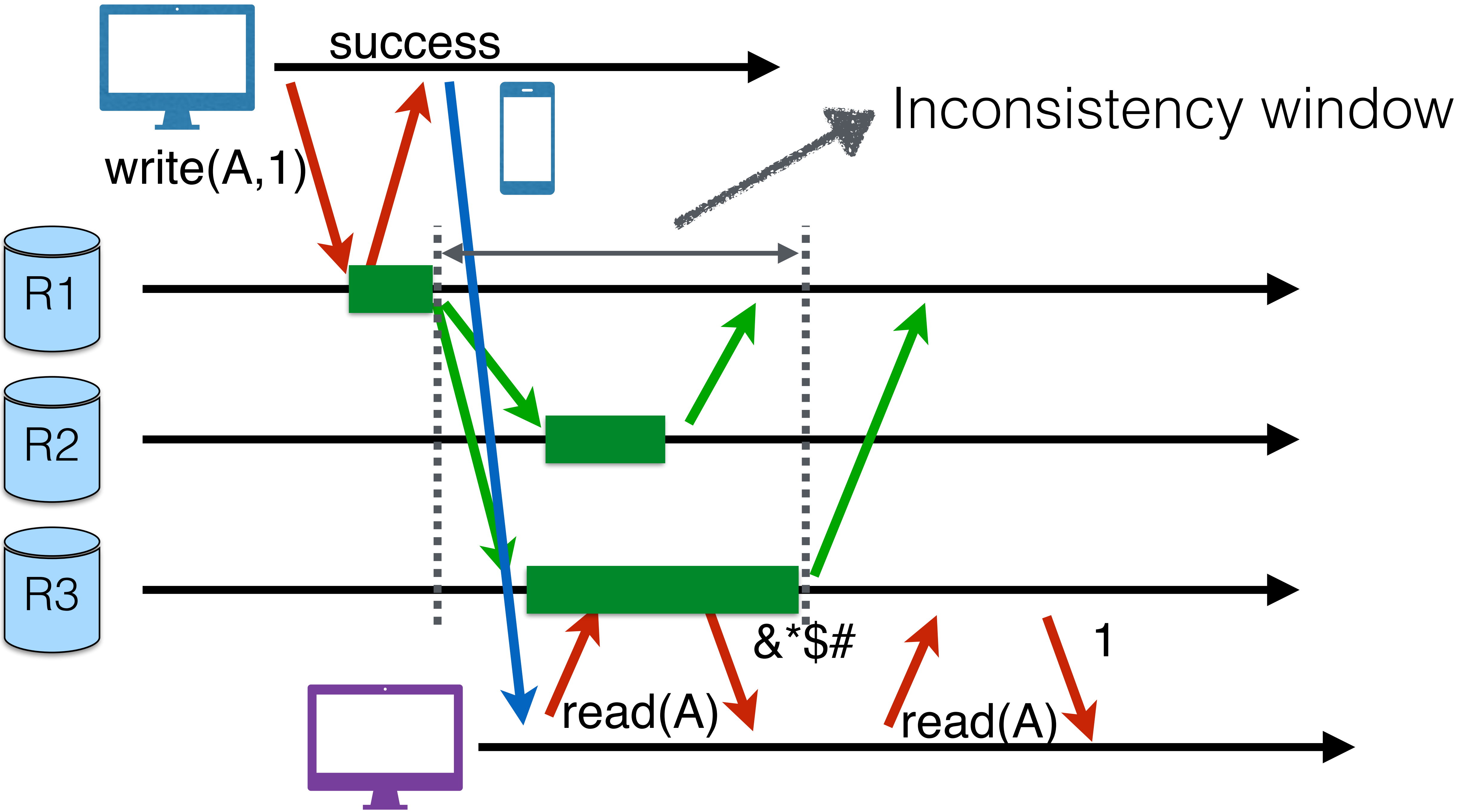
R: 1 replica needs to be contacted for a read

Eventual consistency



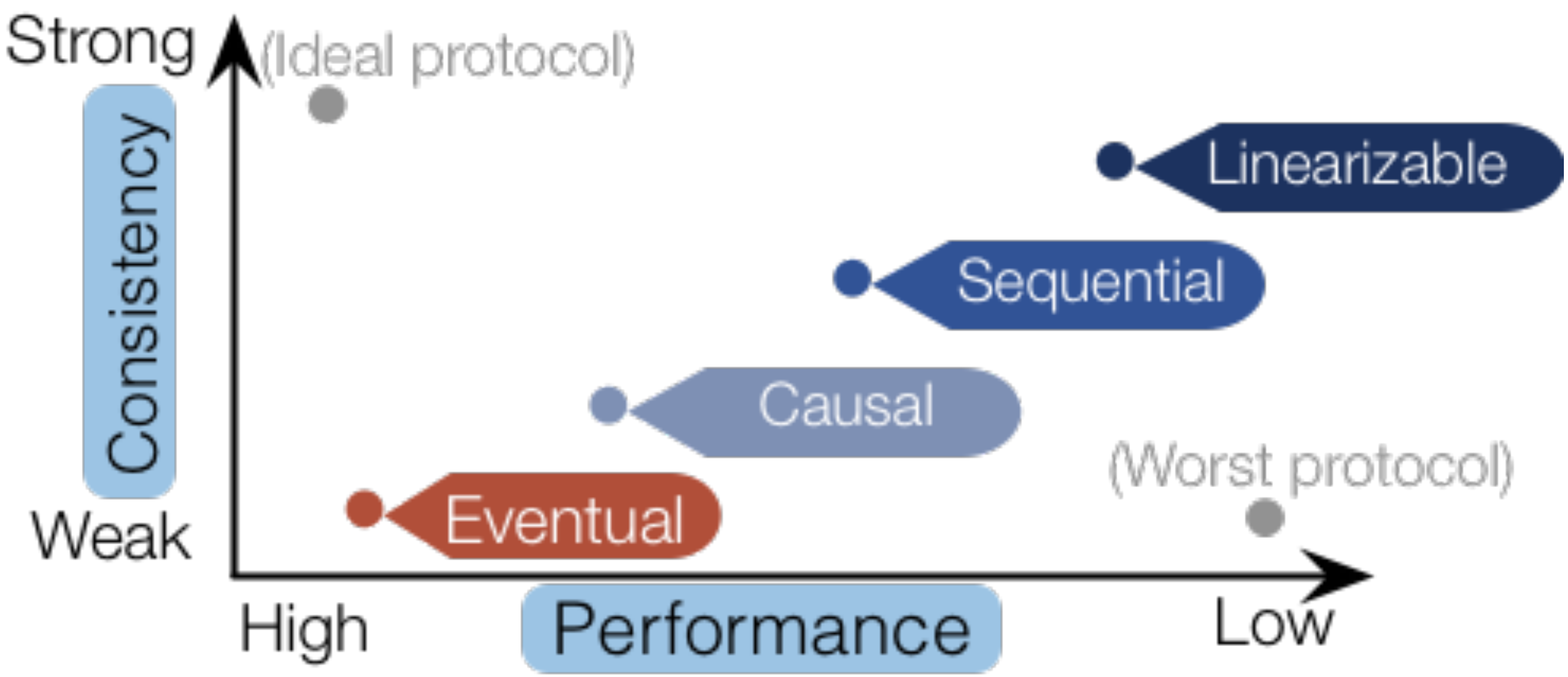
Bob sees stale data within the inconsistency window

Eventual consistency



Bob sees updated data eventually

Consistency models

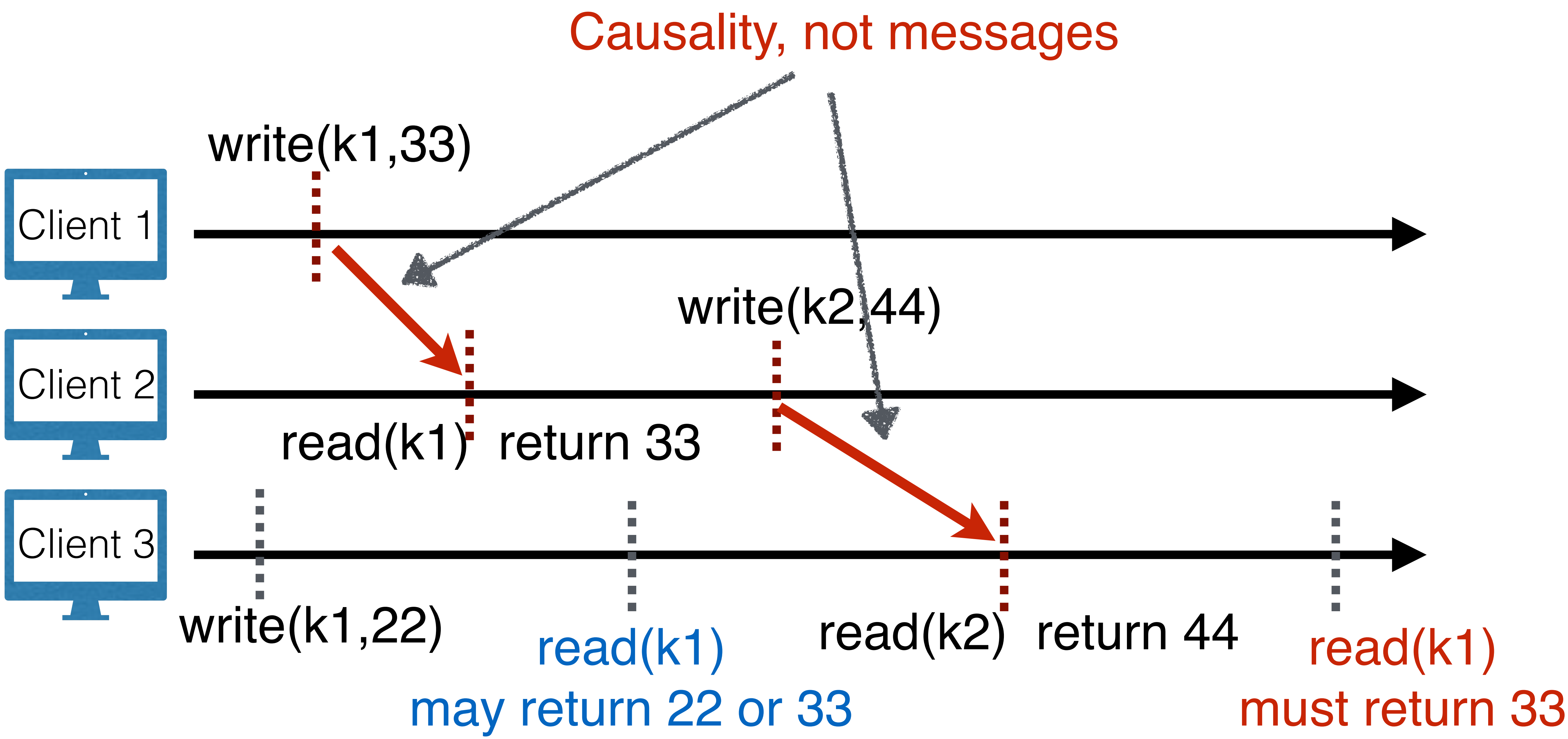


*: Incremental consistency guarantees for replicated objects [USENIX OSDI'16]

Causal consistency

- Reads must respect **partial** order based on information flow
- If **Client 1** has communicated to **Client 2** that it has updated a data item, a subsequent access of **Client 2** will return the updated value and a write is guaranteed to supersede the earlier write
- Access by **Client 3** that has no causal relationship to **Client 1** is subject to the normal eventual consistency rules

Causal consistency example



Announcement

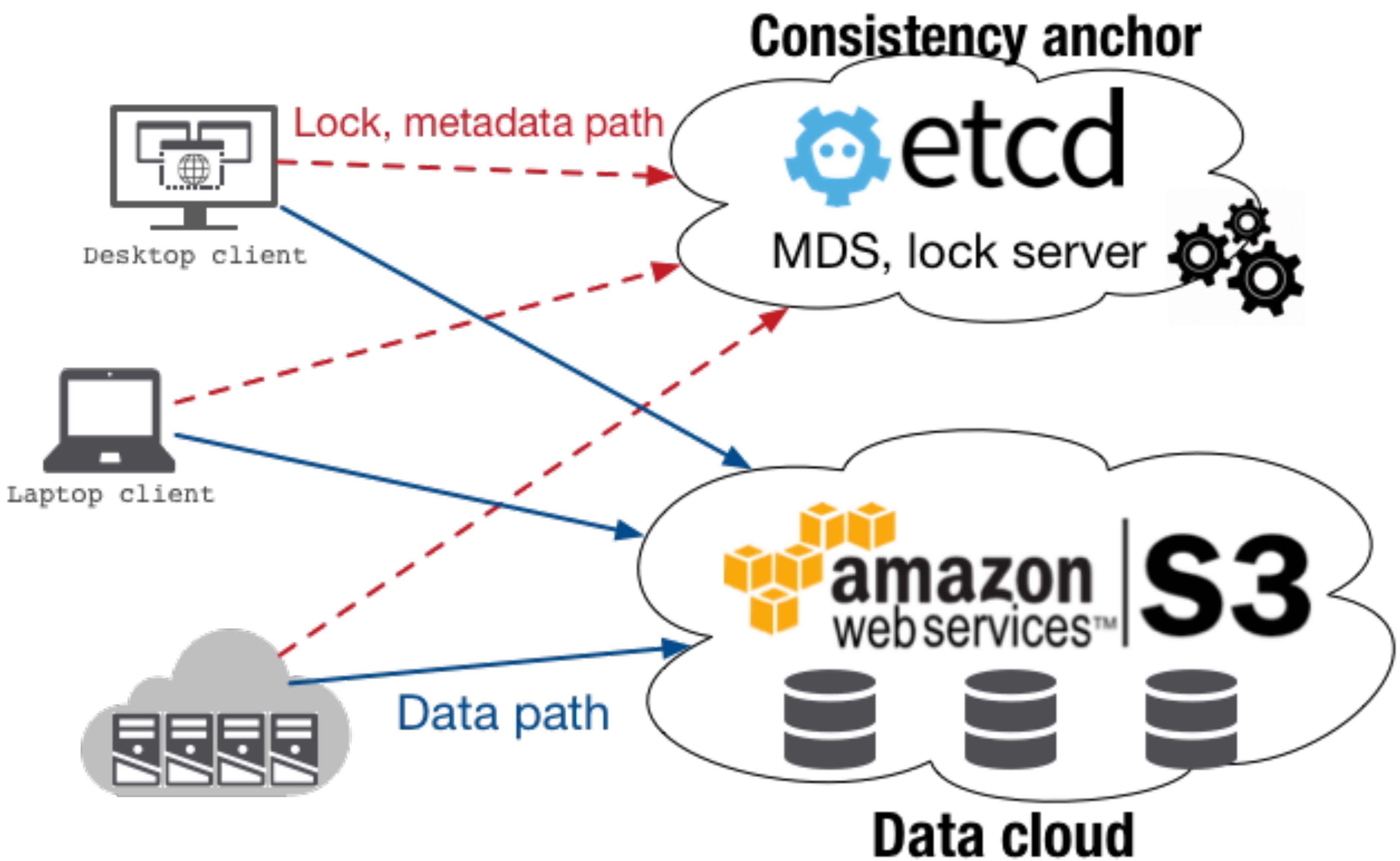
- Homework assignment #1 is out
  - Due by end of Sep 14
- Paper presentations will start from Week 4

- Next class
  - Replicated storage system implementation
  - Distributed consensus
  - Read the papers on website for next class (Sep 12)

Homework assignment 1

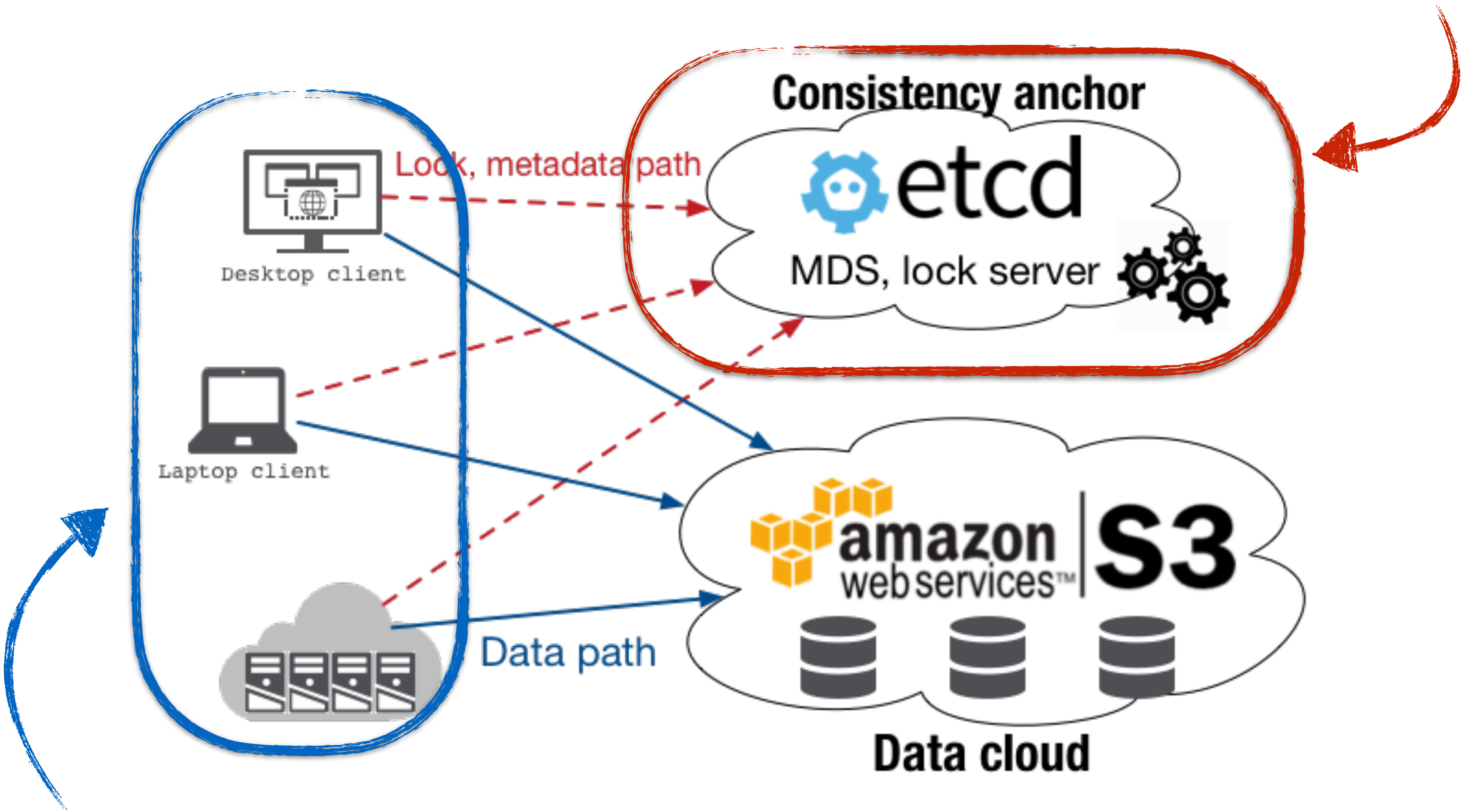
- Build a consistent cloud object store service atop eventually consistent S3
  - Leverage a consistent anchor (etcd)
  - python-etcd provides distributed locking APIs and key-value APIs using the backend Raft-enabled strongly consistent etcd key-value store
- Core idea: leverage S3's versioning support
  - Enable versioning at the bucket level
  - Puts store the latest version ID at etcd (metadata server)
  - Gets fetch the latest version ID and then read from S3, ignores object with stale version ID, until the version ID matches with what has been read from etcd
  - Get/Put/CreateBucket have to be protected using locks (etcd as a lock server)

Homework assignment 1



Homework assignment 1

Deploy etcd microservice



Implement Get/Put/CreateBucket APIs