

RAMCloud: Scalable High-Performance Storage Entirely in DRAM

New research project at *Stanford*

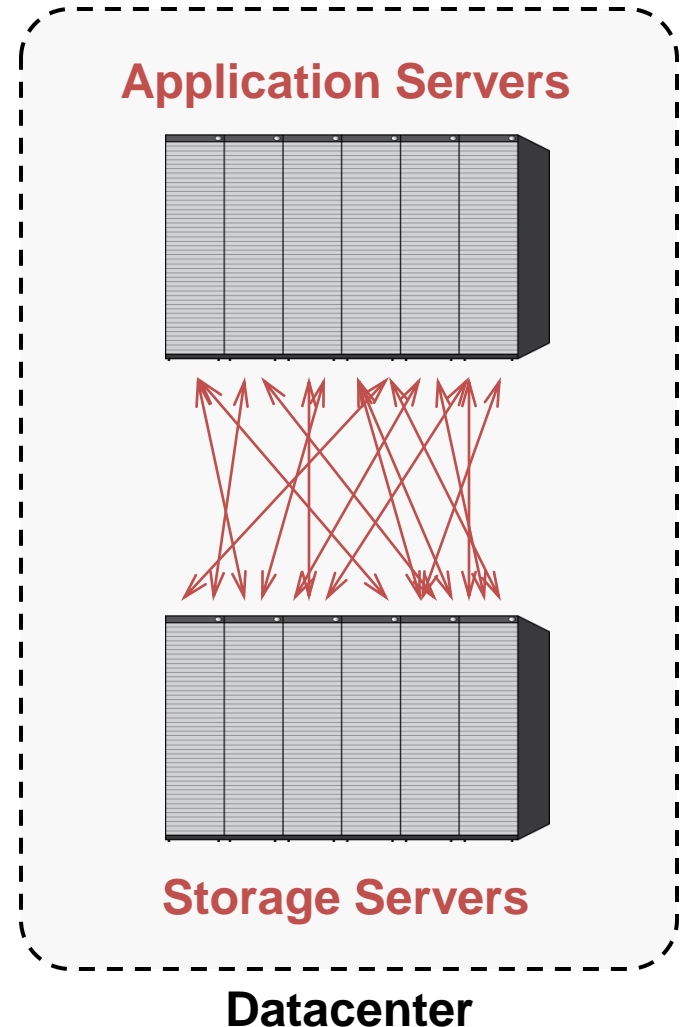
John Ousterhout, Christos Kozyrakis, David Mazières, Aravind Narayanan, Diego Ongaro, Mendel Rosenblum, Stephen Rumble, and Ryan Stutsman

Outline

- **Overview of RAMCloud**
- **Motivation**
- **Research Challenges**
- **Basic cluster structure and data model**

RAMCloud Overview

- Storage for datacenters
- 1000-10000 commodity servers
- 64 GB DRAM/server
- All data always in RAM
- Durable and Available
- Performance goals:
 - High throughput:
1M ops/sec/server
 - Low-latency access:
5-10 μ s RPC

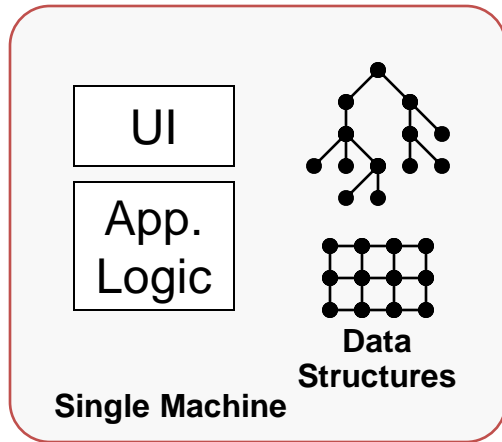


Example Configurations

	Today	5-10 years
# Servers	1000	1000
GB/server	64GB	1024GB
Total capacity	64TB	1PB
Total server cost	\$4M	\$4M
\$/GB	\$60	\$4

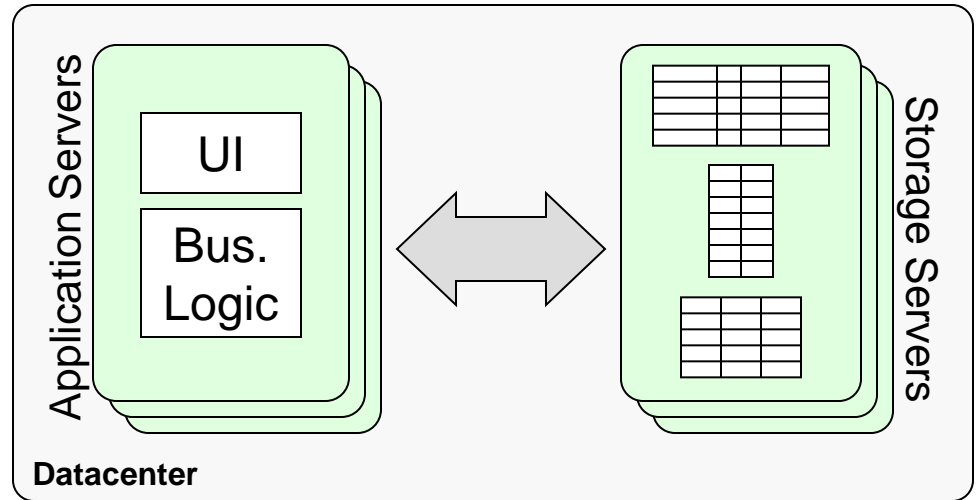
RAMCloud Motivation: Latency

Traditional Application



$\ll 1\mu s$ Latency

Web Application



0.5 – 10 ms Latency

- **Large-Scale apps struggle with high latency**
 - Facebook: can only make 100-150 internal requests per page

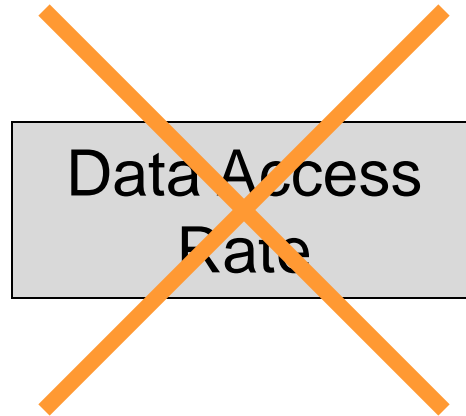
Forms of Scalability

Data Access
Rate

Computation
Power

Storage
Capacity

Forms of Scalability

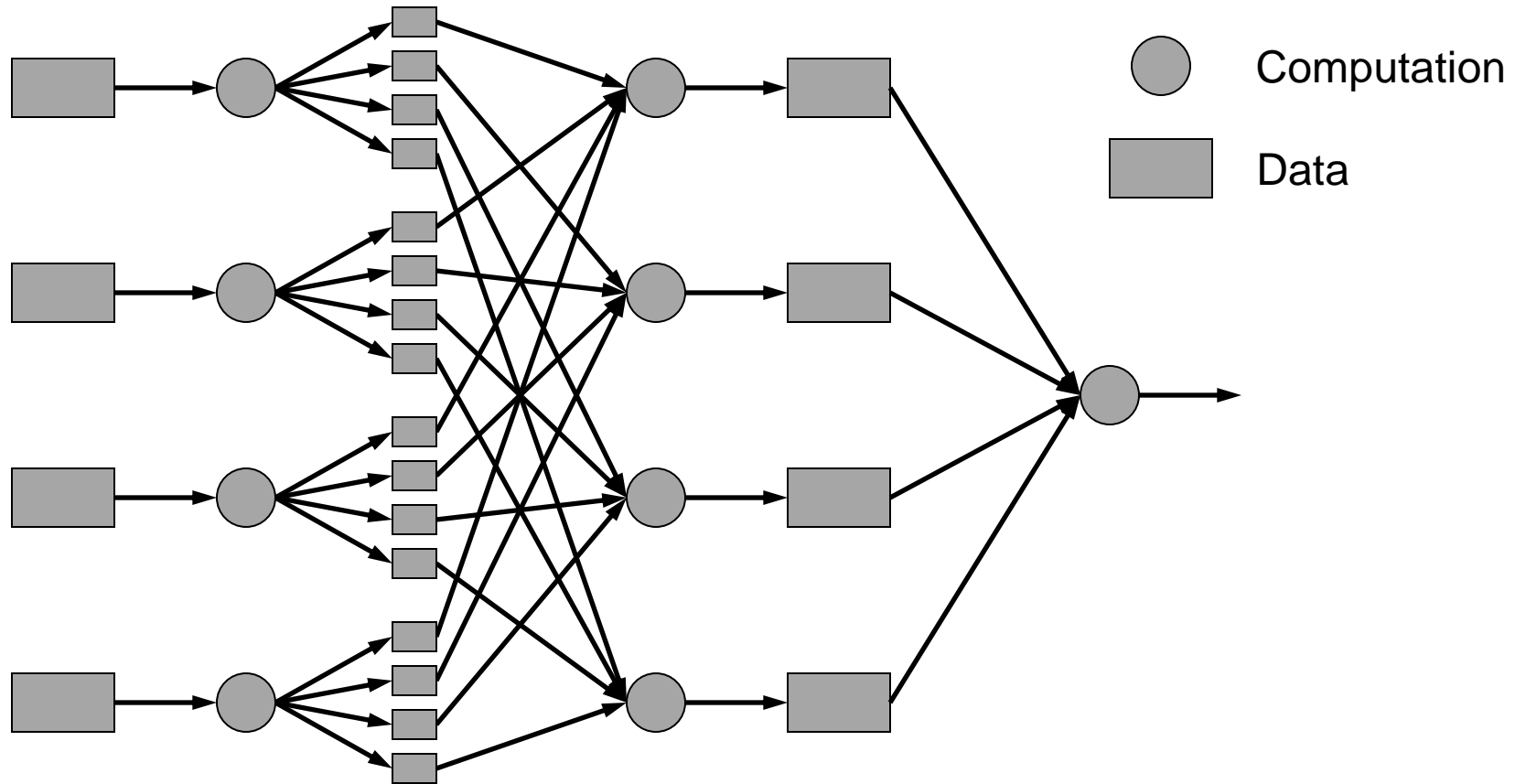


**Not provided by
today's application
infrastructure**

Computation
Power

Storage
Capacity

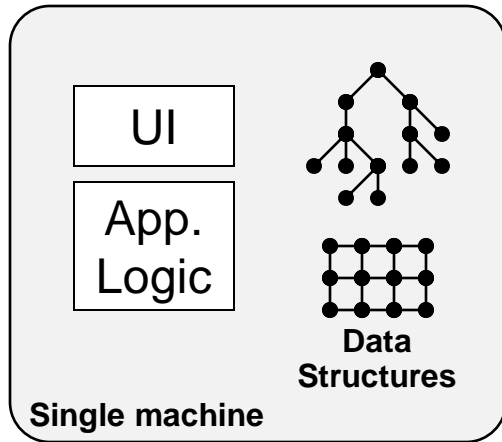
MapReduce



- **Sequential data access → high data access rate**
- **Not all applications fit this model**

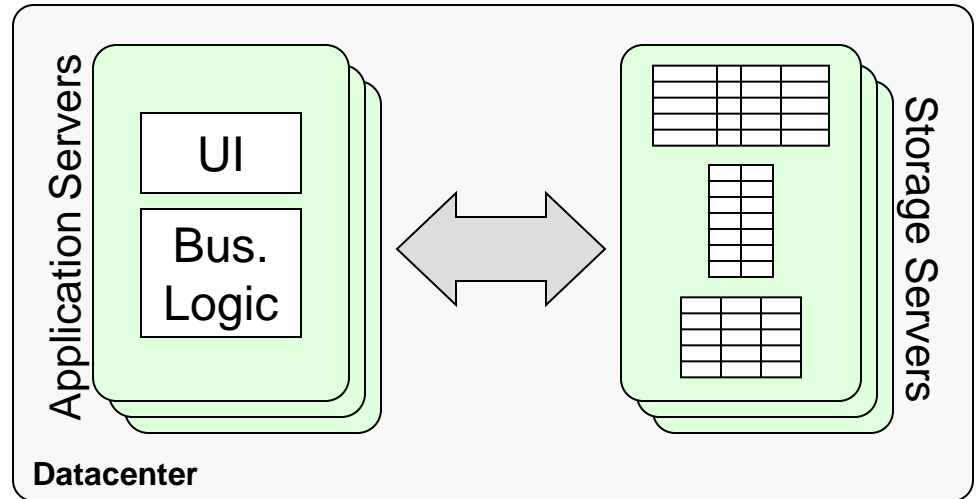
RAMCloud Motivation: Latency

Traditional Application



≪ 1 μ s Latency

Web Application



0.5 – 10 ms Latency
5 – 10 μ s

- RAMCloud goal: **large scale** and **low latency**
- Enable a new breed of information-intensive applications

RAMCloud Motivation: Scalability

- **Relational databases don't scale**
- **Every large-scale Web application has problems:**
 - Facebook: 4000 MySQL instances + 2000 memcached servers
- **New forms of storage starting to appear:**
 - Bigtable
 - Dynamo
 - PNUTS
 - Sinfonia
 - H-store
 - memcached

RAMCloud Motivation: Technology

Disk access rate not keeping up with capacity:

	Mid-1980's	2009	Change
Disk capacity	30 MB	500 GB	16667x
Max. transfer Rate	2 MB/s	100 MB/s	50x
Latency (seek & rotate)	20 ms	10 ms	2x
Capacity/bandwidth (large blocks)	15 s	5000 s	333x
Capacity/bandwidth (1KB blocks)	600 s	58 days	8333x
Jim Gray's rule	5 min	30 hrs	360x

- Disks must become **more archival**
- More **information** must move to **memory**

Why Not a Caching Approach?

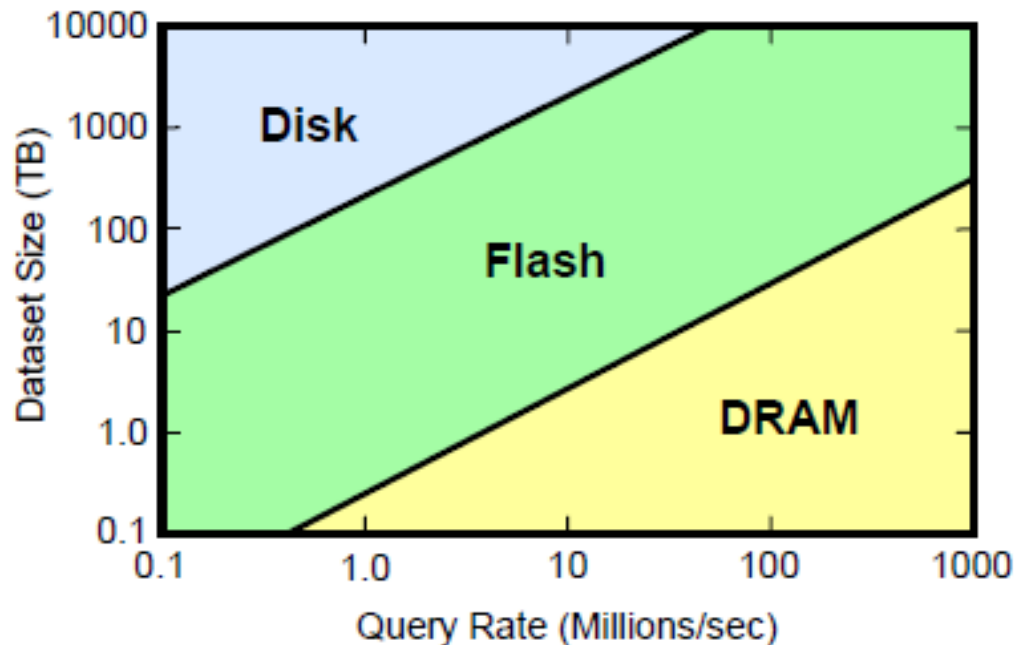
- **Lost Performance:**
 - 1% misses → 10x performance degradation
- **Won't save much money:**
 - Already have to keep information in memory
 - Example: Facebook caches ~75% of data size
- **Changes disk management issues:**
 - Optimize for Reads, **vs.** Writes & Recovery

Why not Flash Memory?

- **Many candidate technologies besides DRAM**
 - Flash (NAND, NOR)
 - PC RAM
 - ...
- **DRAM enables lowest latency:**
 - 5-10x faster than flash
- **Most RAMCloud techniques will apply to other technologies**
- **Ultimately, choose storage technology based on cost, performance, energy, **not volatility****

Disk/Flash/DRAM

- Given requirements in terms of **dataset size** and **operations/sec**:
 - **High query rates & smaller dataset sizes** DRAM is cheapest
 - **Low query rates & large datasets** disk is cheapest
 - However; Flash is cheapest in the middle ground.



Is RAMCloud Capacity Sufficient?

- **Facebook: 260 TB of (non-image) data today**
- **Amazon:**

Revenues/year:	\$16B
Orders/year:	400M? (\$40/order?)
Bytes/order:	1000-10000?
Order data/year:	0.4-4.0 TB?
RAMCloud cost:	\$24-240K?
- **United Airlines:**

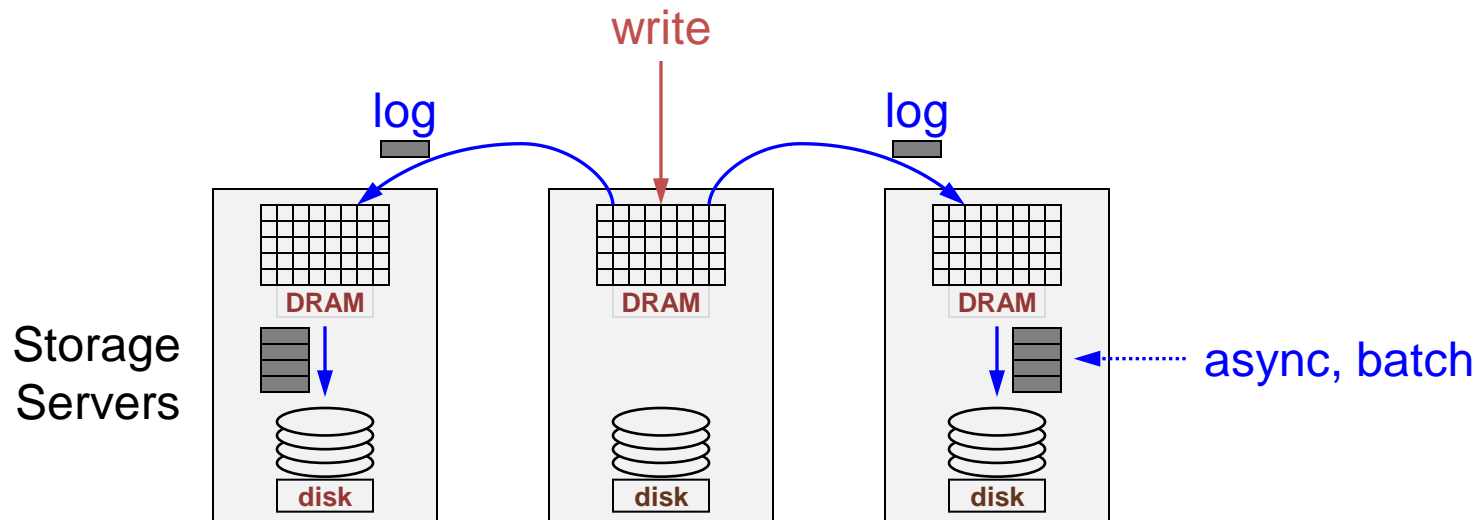
Total flights/day:	4000? (30,000 for all airlines in U.S.)
Passenger flights/year:	200M?
Bytes/passenger-flight:	1000-10000?
Order data/year:	0.2-2.0 TB?
RAMCloud cost:	\$13-130K?
- **Ready today for all online data; media soon : googleTV**

RAMCloud Research Issues

- **Data durability/availability**
- **Fast RPCs**
- **Data model, concurrency/consistency model**
- **Data distribution, scaling**
- **Automated management**
- **Multi-tenancy**
- **Client-server functional distribution**
- **Node architecture**

Data Durability/Availability

- Data must be durable when write RPC returns
- Unattractive approaches:
 - Replicate in other memories (too expensive)
 - Synchronous disk write (100-1000x too slow)
- One possibility: buffered logging

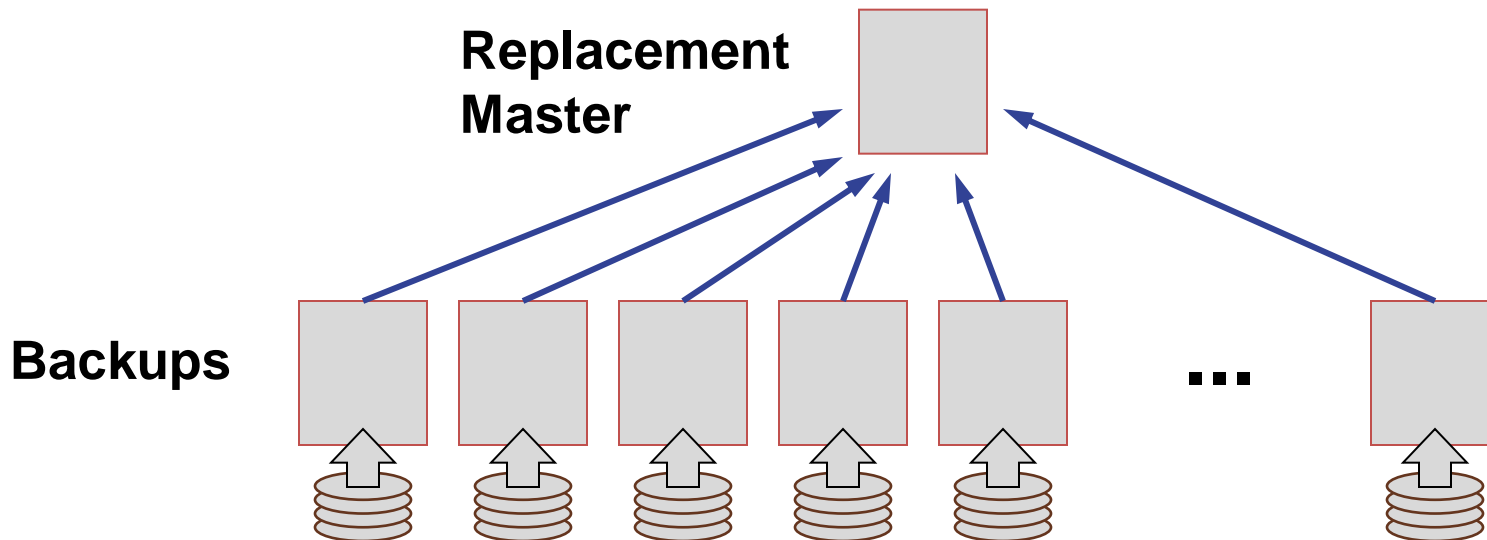


Buffered Logging, cont'd

- **Potential problem: crash recovery**
 - If **master crashes**, data unavailable until read from disks on backups
 - Read 64 GB from one disk? **10 minutes**
 - **Our goal: recover in 1-2 seconds**
- **Solution: take advantage of system scale**
 - Shard backup data across many servers
 - Recover in parallel
 - Master Recovery
 - 2-Phase
 - Partitioned
 - Failures
 - Backups
 - Rack/Switch
 - Power
 - Datacenter

Recovery, First Try

- Scatter log segments randomly across all servers
- After crash, all backups read disks in parallel (64 GB/1000 backups @ 100 MB/sec = 0.6 sec)
- Collect all backup data on replacement master (64 GB/10GB/sec ~ **60 sec: too slow!**)

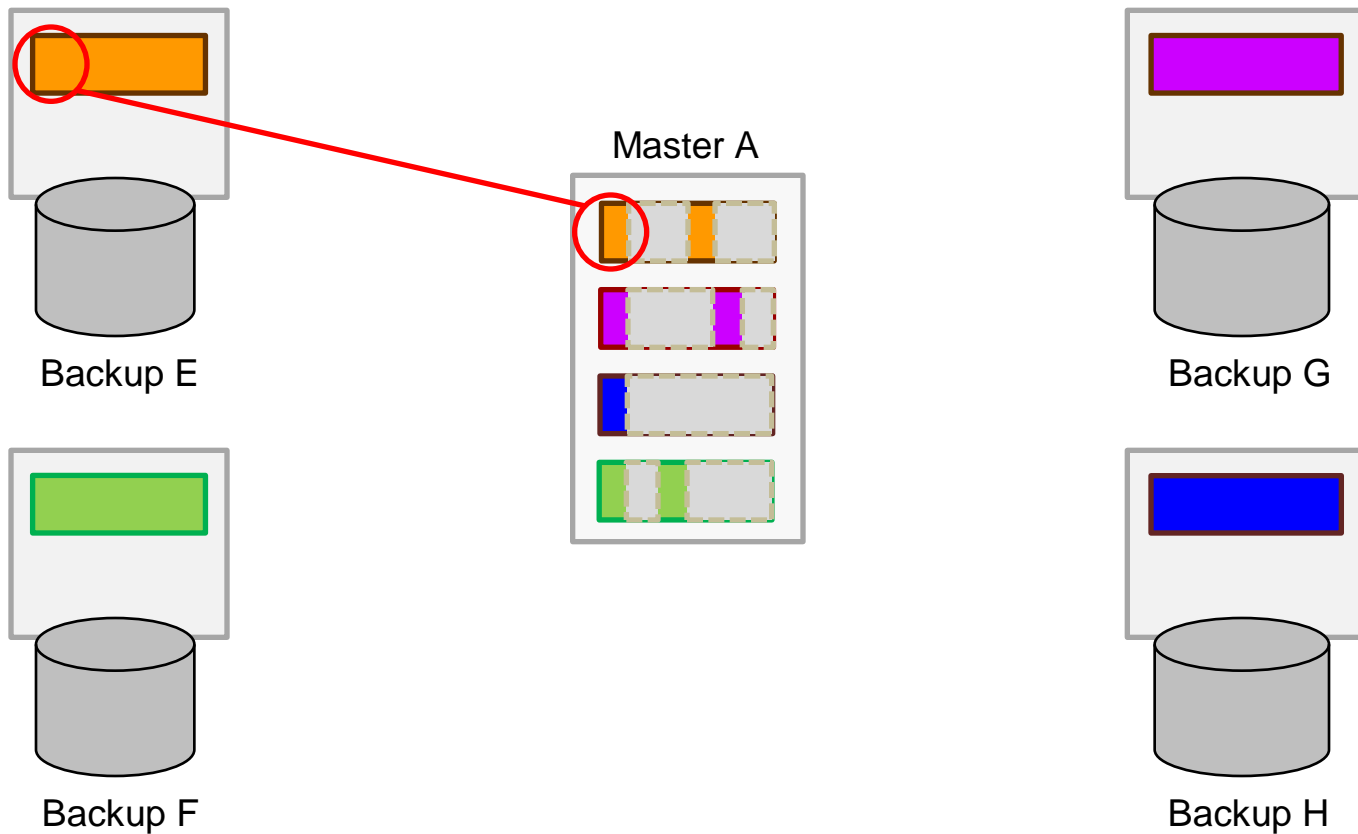


Recovery, Second Try

- **Phase #1:**
 - Read all segments into memories of backups
 - Send only location info to replacement master
 - Elapsed time depends on # objects
- **Phase #2:**
 - **System resumes operation:**
 - Replacement master fetches objects on demand from backups (2x performance penalty for reads)
 - Writes handled on replacement master
 - Meanwhile, transfer segments from backups to replacement master
 - Elapsed time: ~1 minute
- **Performance returns to normal after Phase #2**

2-Phase Recovery

- Idea: **Data already in memory** on backups, just need to know **“where?”**



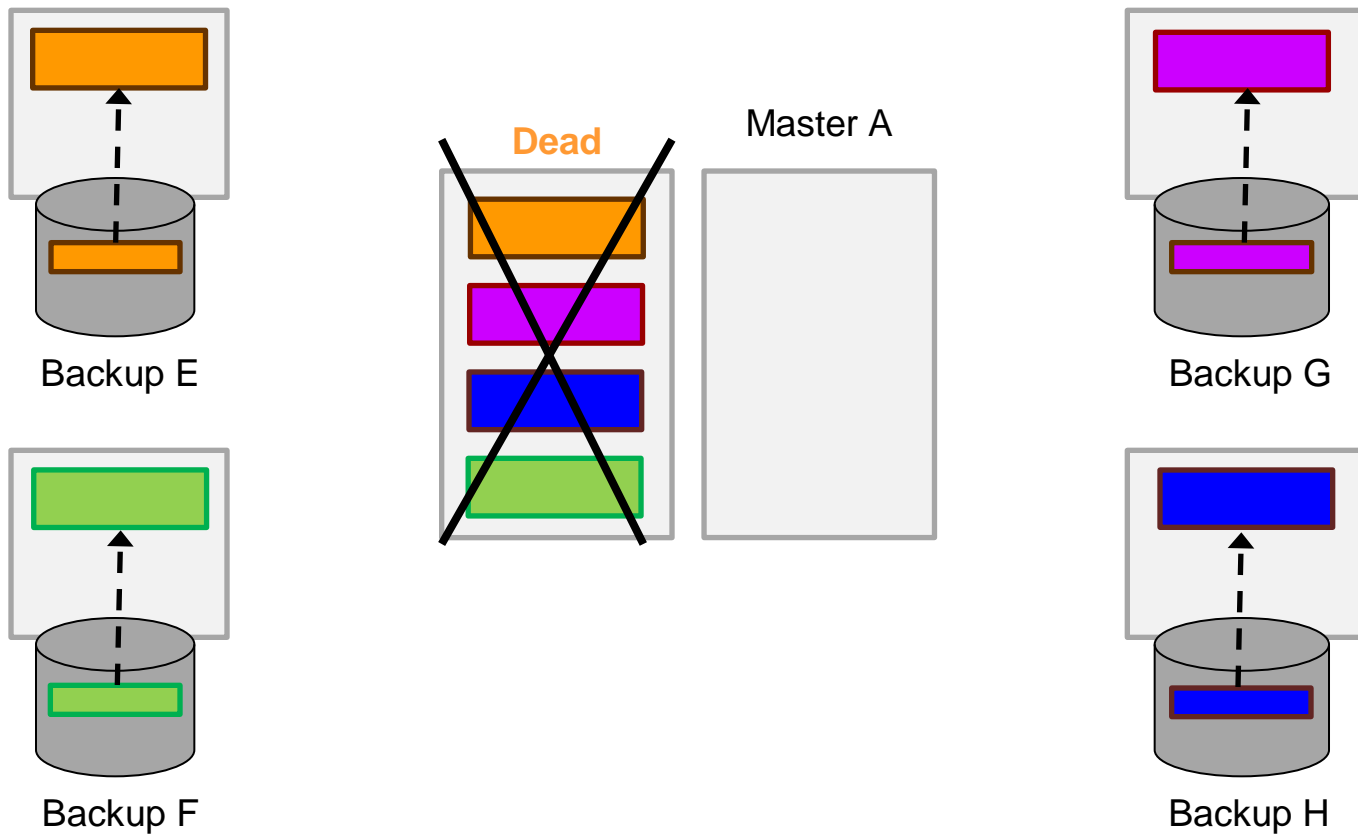
2-Phase Recovery

- Phase #1: Recover location info (< 1s)



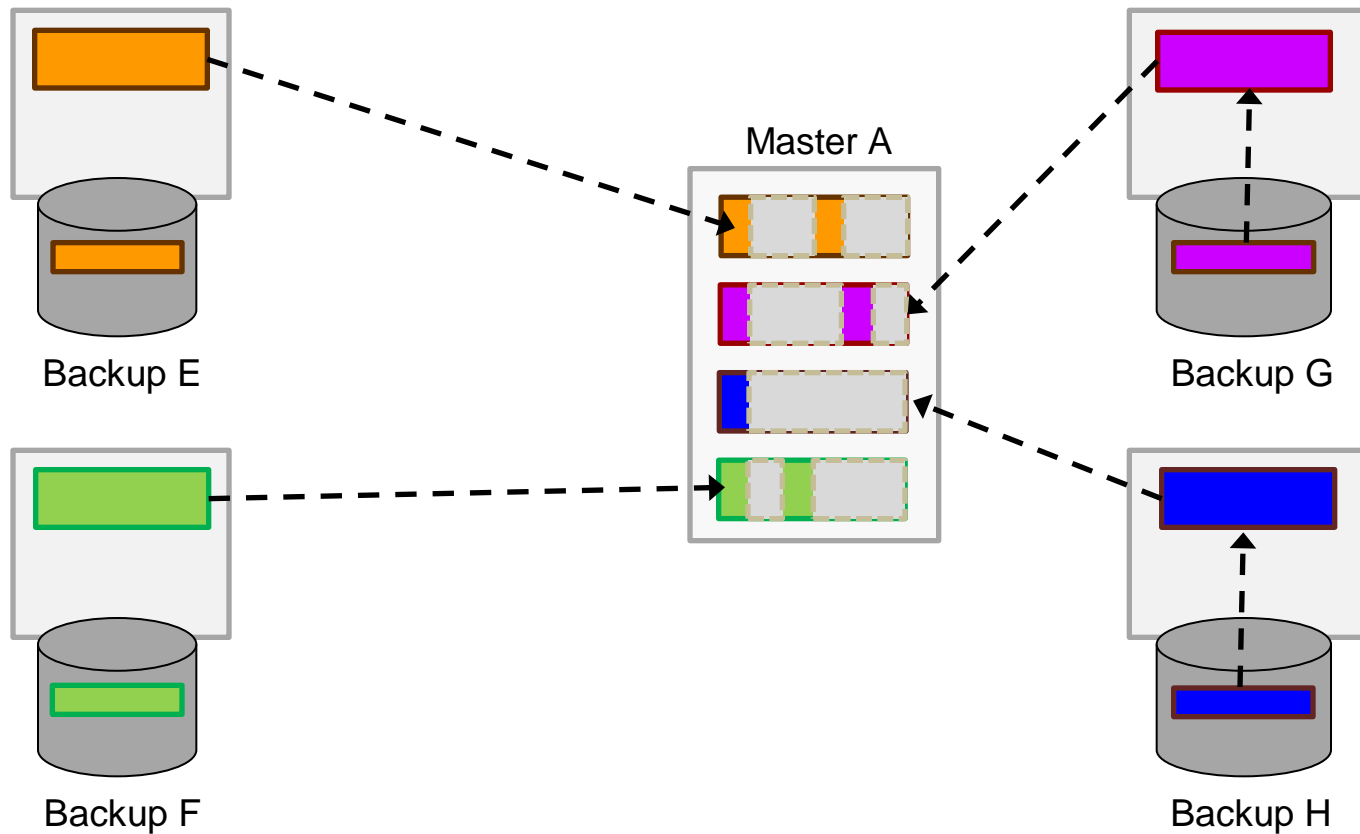
2-Phase Recovery

- **Phase #1: Recover location info (< 1s)**
 - Read all data into memories of backups



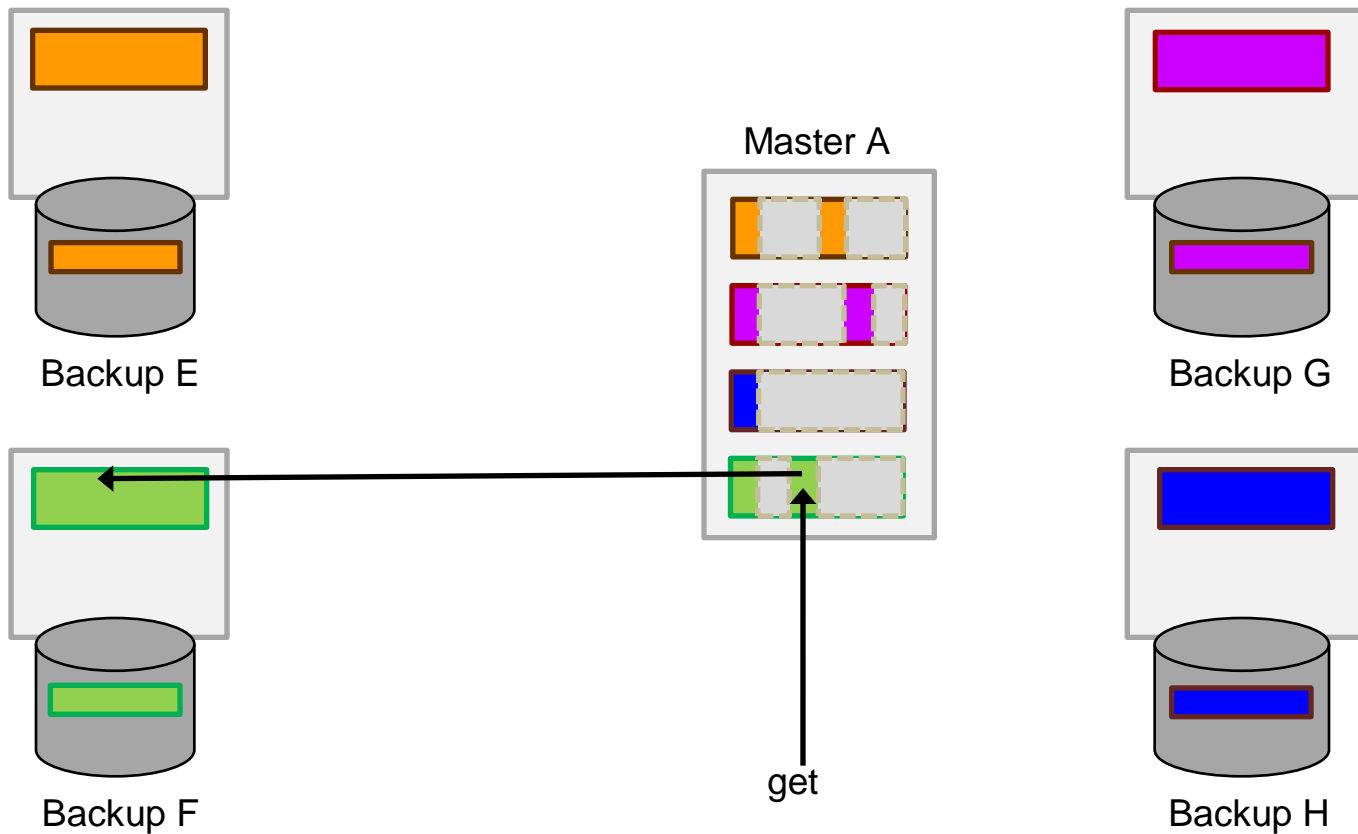
2-Phase Recovery

- **Phase #1: Recover location info (< 1s)**
 - Read all data into memories of backups
 - Send **only location info** to replacement master



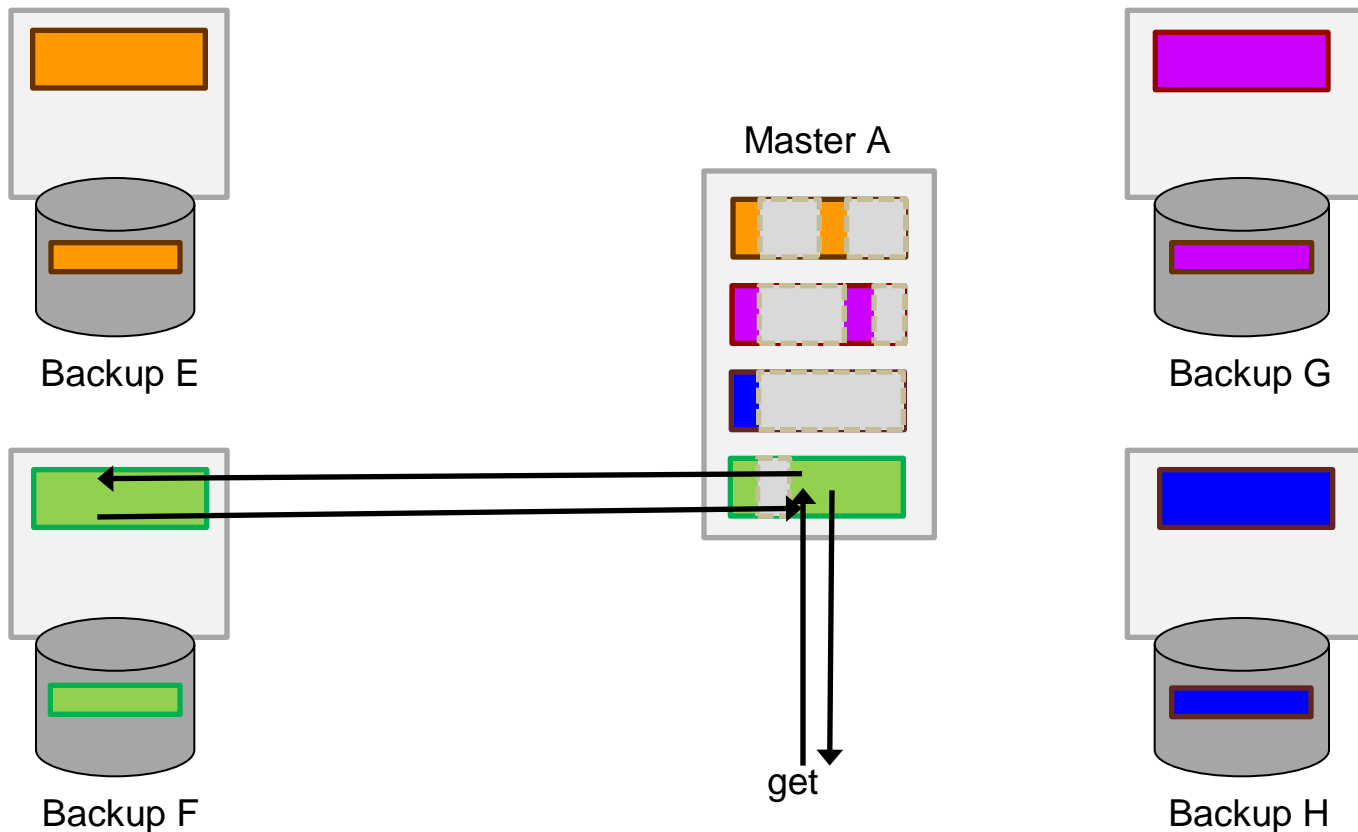
2-Phase Recovery

- Phase #2: Proxy & Recover Full Data (~60s)
 - System resumes operation:
 - Fetch on demand from backups
 - 1 extra round trip on first read of an object
 - Writes are full speed



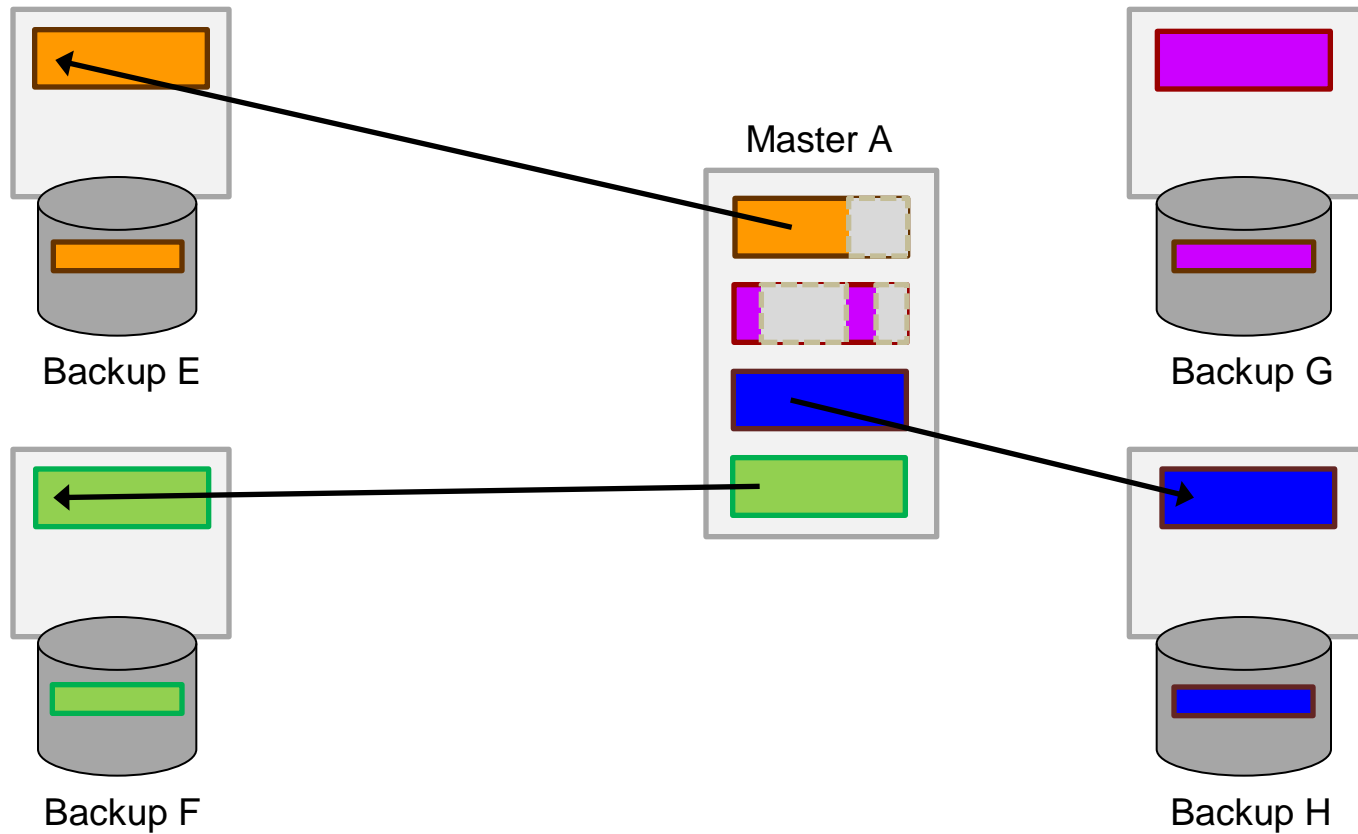
2-Phase Recovery

- Phase #2: Proxy & Recover Full Data (~60s)
 - System resumes operation:
 - Fetch on demand from backups
 - 1 extra round trip on first read of an object
 - Writes are full speed



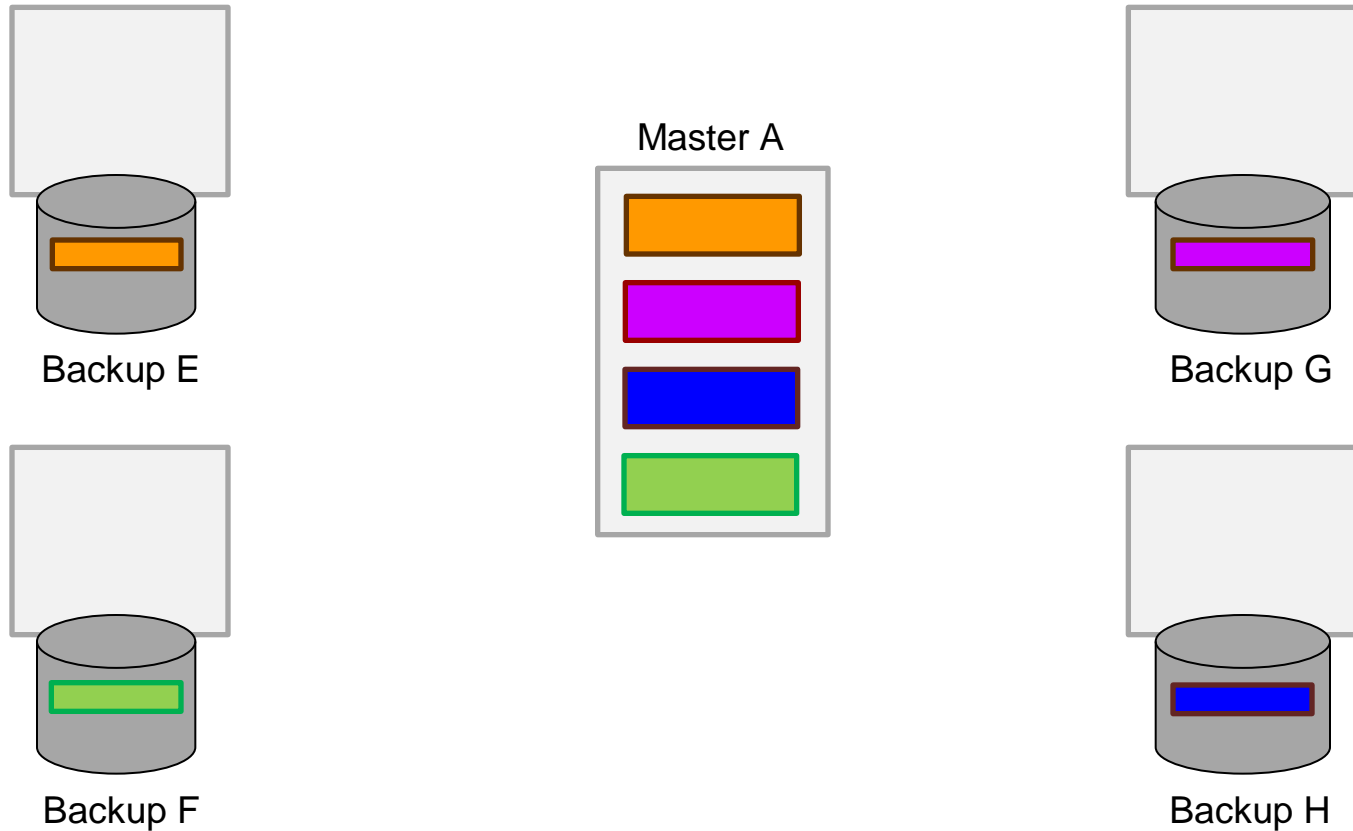
2-Phase Recovery

- **Phase #2: Proxy & Recover Full Data (~60s)**
 - Transfer data from backups in between servicing requests



2-Phase Recovery

- Performance **normal** after **Phase #2** completes

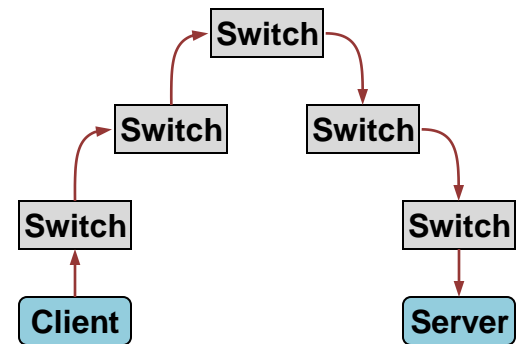


Low-Latency RPCs

Achieving 5-10 μ s will impact every layer of the system:

- **Must reduce network latency:**

- Typical today: 150-300 μ s (10-30 μ s/switch, 5 switches each way)
- Arista: 0.9 μ s/switch: 9 μ s roundtrip
- Need cut-through routing, congestion mgmt



- **Tailor OS on server side:**

- Dedicated cores
- No interrupts?
- No virtual memory?

Low-Latency RPCs, cont'd

- **Network protocol stack**
 - TCP too slow (especially with packet loss)
 - Must avoid copies
- **Client side: need efficient path through VM**
 - User-level access to network interface?
- **Preliminary experiments:**
 - 10-15 μ s roundtrip
 - Direct connection: no switches
 - Biggest current problem: NICs (3-5 μ s each way)

Consistency: Thoughts

- **Atomic puts give index updates atomicity**
- **Low-latency gives simplified consistency**
 - Can afford to have a single writer per object
 - Provides us with atomic put primitive for free

Consistency

- **Problem: Index/Object inconsistency on puts**
 - Object and index may reside on different hosts
 - Apps can get objects that aren't in the index yet
 - Apps may see index entries for objects not in table yet
- **Avoid commit protocol**
- **Idea: Index entries “commit” on object put**
 - Write index entries
 - **Then** write object to table
 - Index entries considered invalid until object written
- **Turns atomic puts into atomic index updates**

Low Latency: Stronger Consistency?

- **Might be able to help with scaling challenges**
 - Example: Durability, update multiple in-memory replicas
- **Cost of consistency rises with transaction overlap:**
 - $O \sim R * D$
 - O** = # overlapping transactions
 - R** = arrival rate of new transactions
 - D** = duration of each transaction
- **R increases with system scale**
 - Eventually, scale makes consistency unaffordable
- **But, D decreases with lower latency**
 - **Stronger consistency affordable at larger scale**
 - Is this phenomenon strong enough to matter?

Data Model

- Defines the **nature** of the **basic objects** stored
- Support for **aggregation**: how are basic objects organized into higher-level structures?
- Mechanisms for **naming** and **indexing**: when retrieving or modifying basic objects, how are the objects named? In the simplest case, such as a key-value store

Data Model Basics

- **Workspace:**
 - All data for one or more apps
 - Unit of access control
- **Table:**
 - Related collection of objects
- **Object:**
 - Variable-length up to 1MB
 - **Contents opaque to servers**
- **Id:**
 - 64 bits, unique within table
 - Chosen explicitly by client or implicitly by server (0,1,2,...)
- **Version:**
 - 64 bits
 - Guaranteed increasing, even across deletes

Workspace

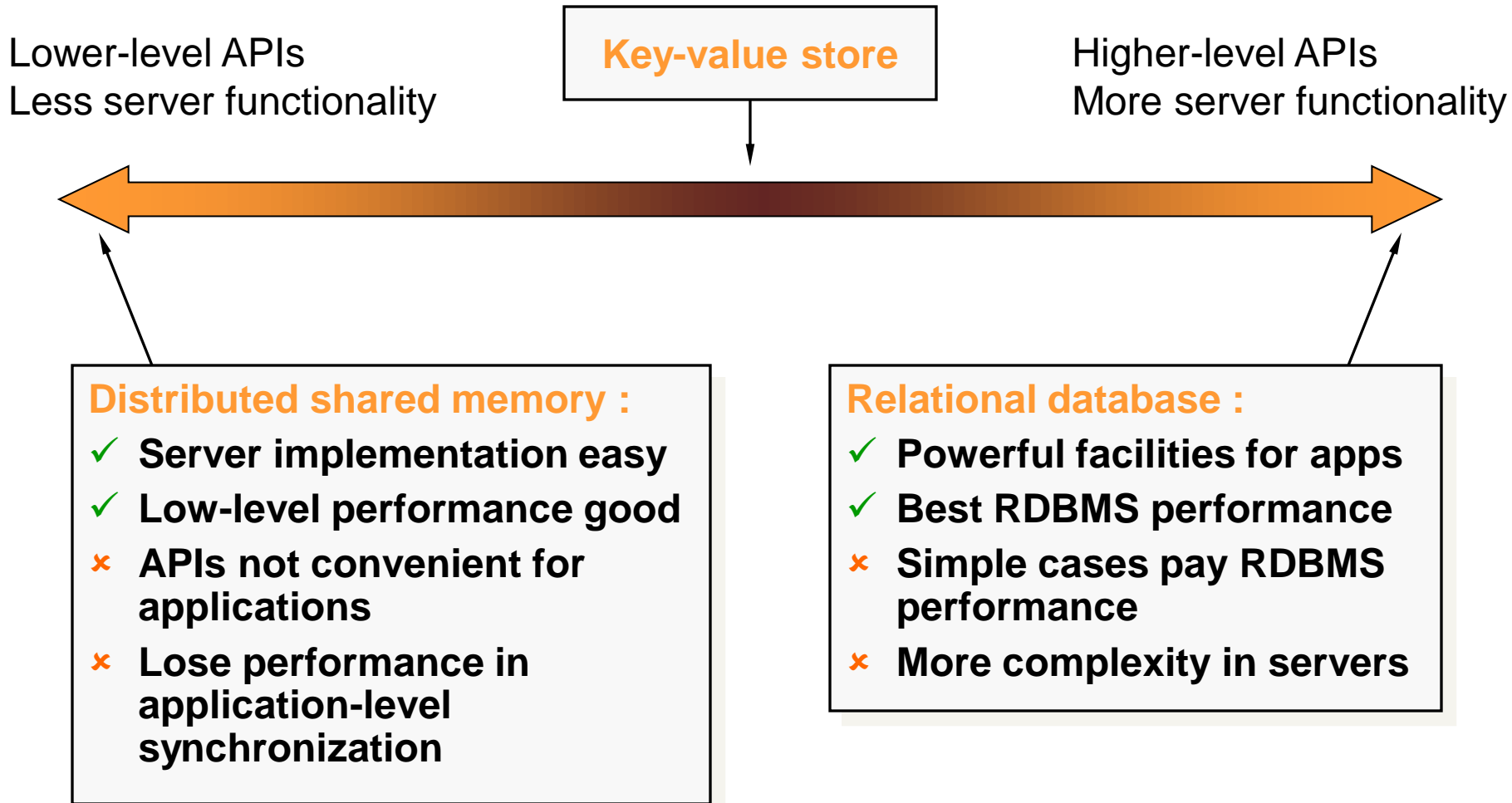
Table

id	object	vers
id	object	vers
id	object	vers
id	object	vers
id	object	vers

Table

id	object	vers
id	object	vers
id	object	vers
id	object	vers

Data Model Rationale



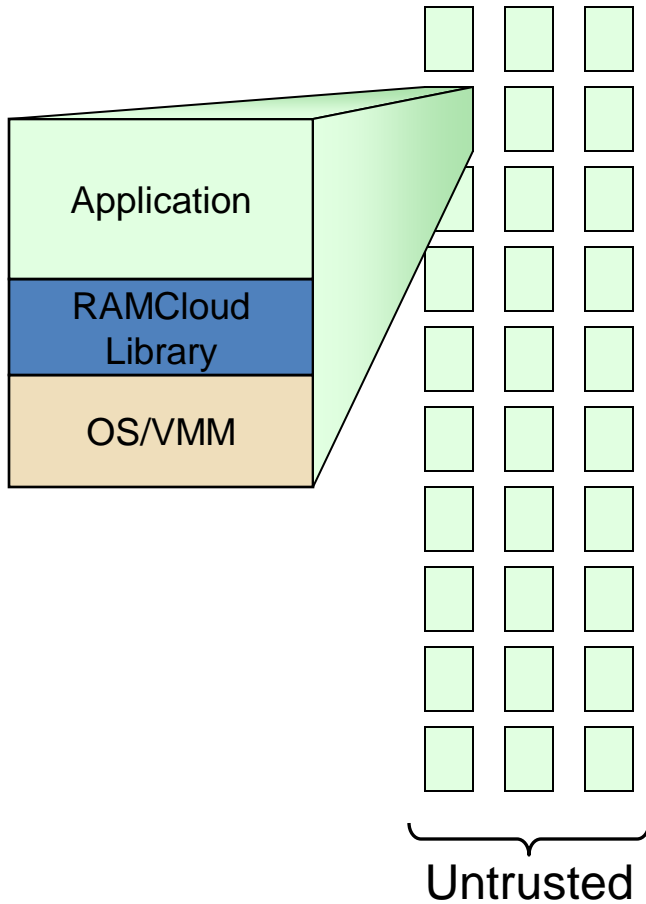
How to get best **application-level** performance?

Other Design Goals

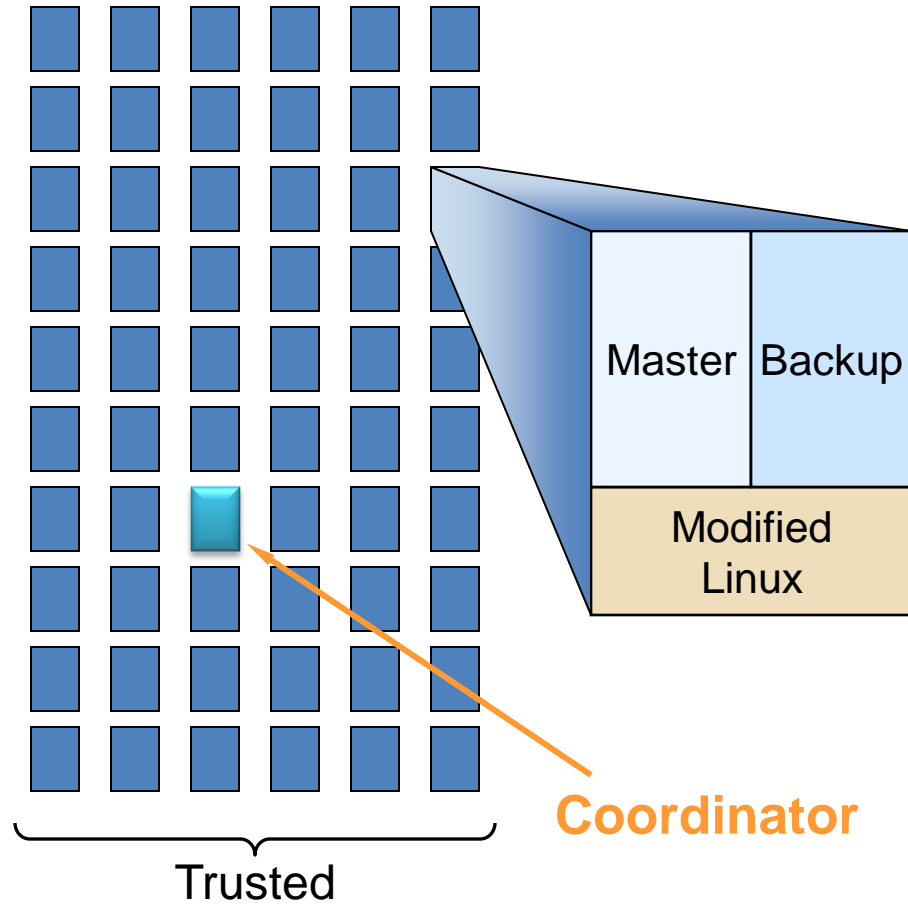
- **Data distributed automatically by RAMCloud:**
 - Tables can be split across multiple servers
 - Indexes can be split across multiple servers
 - Distribution transparent to applications
- **Multi-tenancy for cloud computing:**
 - Support multiple (potentially hostile) applications
 - Cost proportional to application size
- **Explicit search keys both flexible and efficient**
- **Split indexes on search key for fast lookup**
- **Atomic puts simplify atomic indexes**
- **Scale drives index recovery for availability**

RAMCloud Cluster Structure

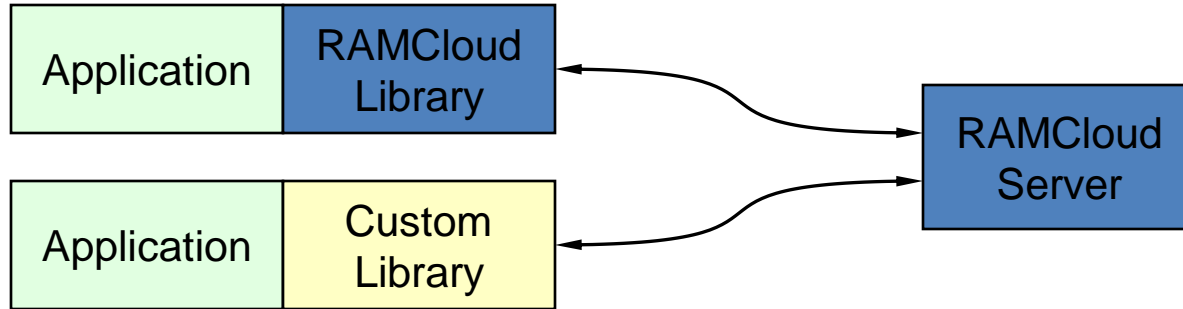
**Clients
(App Servers)**



Servers



Client Library vs. Server



- **Move functionality to library?**
 - Flexibility: enable different implementations
 - Throughput: offload servers
 - May improve performance (e.g., aggregation)
- **Concentrate functionality in servers?**
 - May improve performance (e.g., faster synchronization)
 - Can't depend on proper client behavior:
 - Security/access control
 - Consistency/crash recovery

RAMCloud Disadvantages

- **High cost per bit and high energy usage per bit.**
 - For both of these metrics RAMCloud storage will be 50-100x worse than a pure disk-based system and 5-10x worse than a storage system based on flash memory
- **Require more floor space in a datacenter than a system based on disk or flash memory.**
 - if an application needs to store a large amount of data inexpensively and has a relatively low access rate, RAMCloud is not the best solution.
- Cross-datacenter replication makes it harder for RAMClouds to achieve stronger consistency

Interesting Facets

- **Use each system property to improve the others**
- **High server throughput:**
 - No replication for performance, only durability?
 - Simplifies consistency issues
- **Low-latency RPC:**
 - Cheap to reflect writes to backup servers
 - Stronger consistency?
- **1000's of servers:**
 - Sharded backups for fast recovery

Conclusion

- Interesting combination of **scale** and **latency**
- Enable more powerful uses of information at scale:
 - 1000-10000 clients
 - 100TB - 1PB
 - 5-10 μ s latency

Thank You!