Durability and Crash Recovery for Distributed In-Memory Storage

Ryan Stutsman

Stanford University

Introduction

- **RAMCloud:** large-scale datacenter storage in DRAM
  - **Scale:** up to 10,000 servers, 1+ PB capacity
  - **Low latency:** 5 µs remote access, 1M ops/s/server
  - 1000× faster than disk-based storage systems
- **Impact:** more data-intensive applications

- **Key problem:** durability and availability
  - Cost of DRAM rules out traditional replicated approaches
- **My work: DRAM as reliable as replicated disk storage without performance or cost penalties**

Contributions

DRAM as reliable as replicated disk storage without performance or cost penalties

Scalable Crash Recovery

- Availability through fast recovery rather than redundancy
- Leverages scale to restore an entire server's DRAM in 1 to 2 s

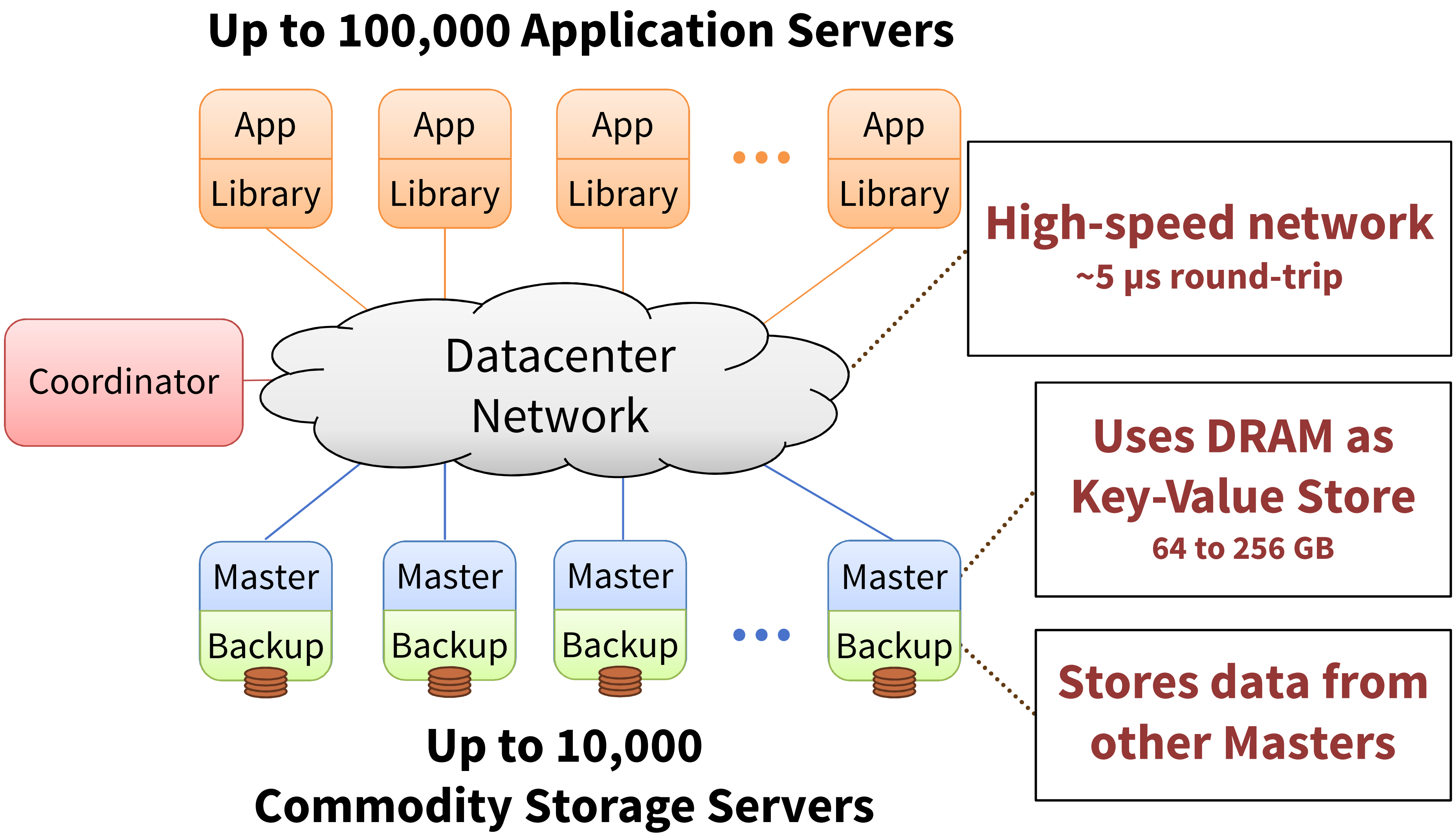
Fault-tolerant Decentralized Log Structure

- Provides fast writes by eliminating disk accesses
- Retains consistency and restores durability after failures
- Scales by avoiding centralization

Outline

- Introduction
- Contributions
- RAMCloud Overview
- Fast Crash Recovery
  - Evaluation
- Distributed Fault-Tolerant Log
- Surviving Massive Failures
- Conclusion

RAMCloud Architecture



Outline

- Introduction
- Contributions
- RAMCloud Overview
- Fast Crash Recovery
    - Evaluation
- Distributed Fault-Tolerant Log
- Surviving Massive Failures
- Conclusion

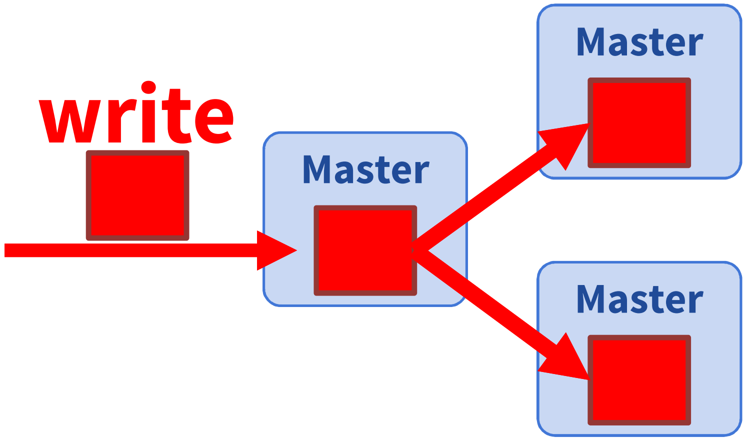
Availability Challenges

- Server failures frequent
- DRAM is **volatile** and **expensive**

Goals

- As **durable as disk-based storage** systems
- Minimize impact on performance
- Minimum **cost**, **energy**

Strawman: Replicate in DRAM?



- 3× system cost, energy
- Still have to handle power failures
- Standard datacenter battery backups limited to 45 s

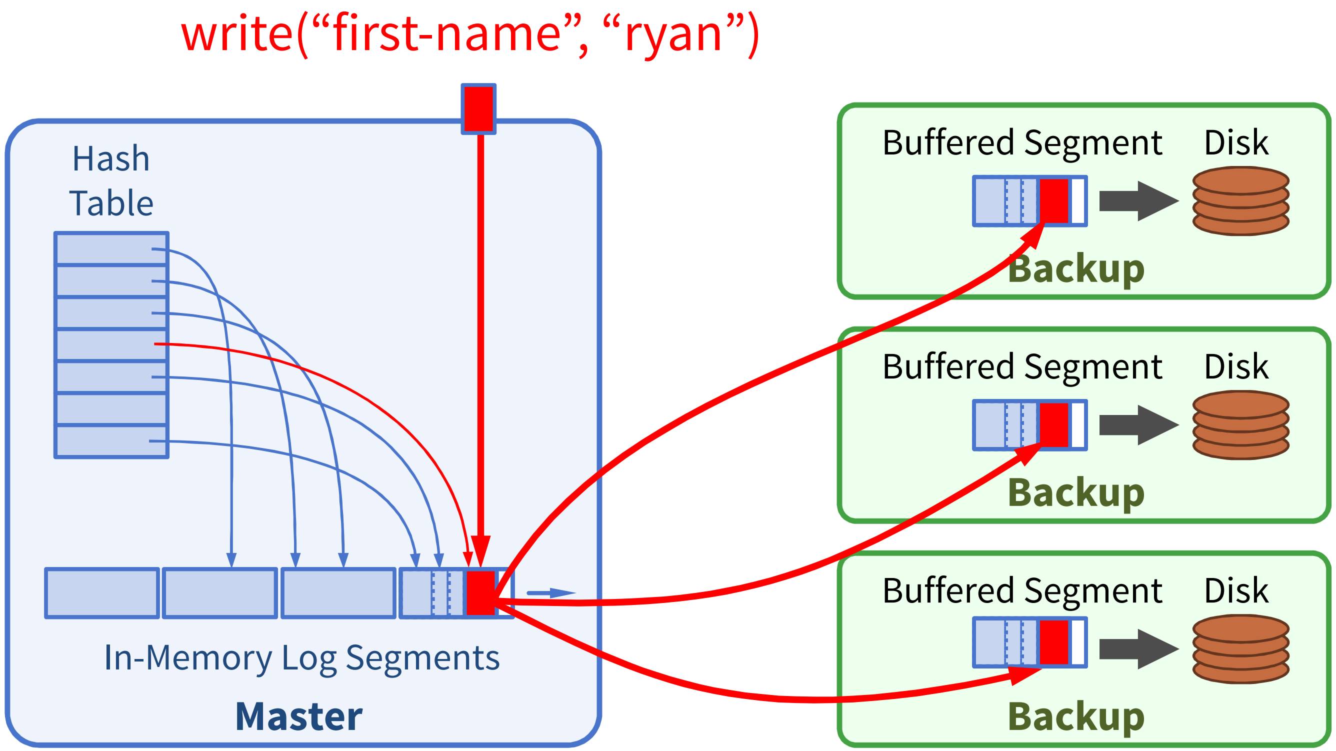
RAMCloud's Approach

- 1 copy in DRAM
- Backup on disk/flash: cheap compared to DRAM

- **Problem:** synchronous disk writes too slow
  - **Fault-tolerant decentralized log structure**

- **Problem:** data is unavailable on crash
  - **Fast crash recovery in 1 to 2 s**
  - Fast enough that applications won't notice

Fast Writes: Buffered Logging



- No disk I/O during write requests
- Efficient 8 MB bulk writes
- Limits buffered data; flush on power loss in < 250 ms

Restoring Availability: Crash Recovery

- What is left when a master crashes?
  - Log data stored on disk on backups

- What must be done to restart servicing requests?
  - Replay log data into DRAM
  - Reconstruct the hashtable

Recovery Bottlenecks

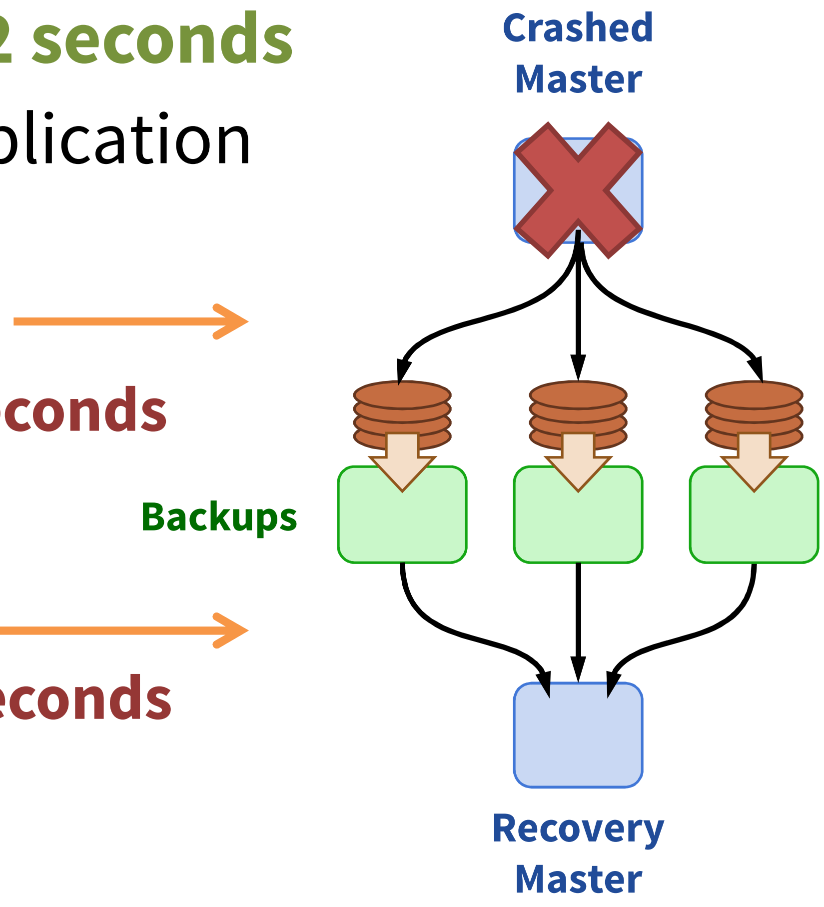
- Goal: recover 64 GB in **1-2 seconds**
- Can't do it with simple replication

Disks: ~100 MB/s each

64 GB / 300 MB/s ≈ **210 seconds**

NIC: ~1 GB/s

64 GB / 1 GB/s ≈ **60 seconds**



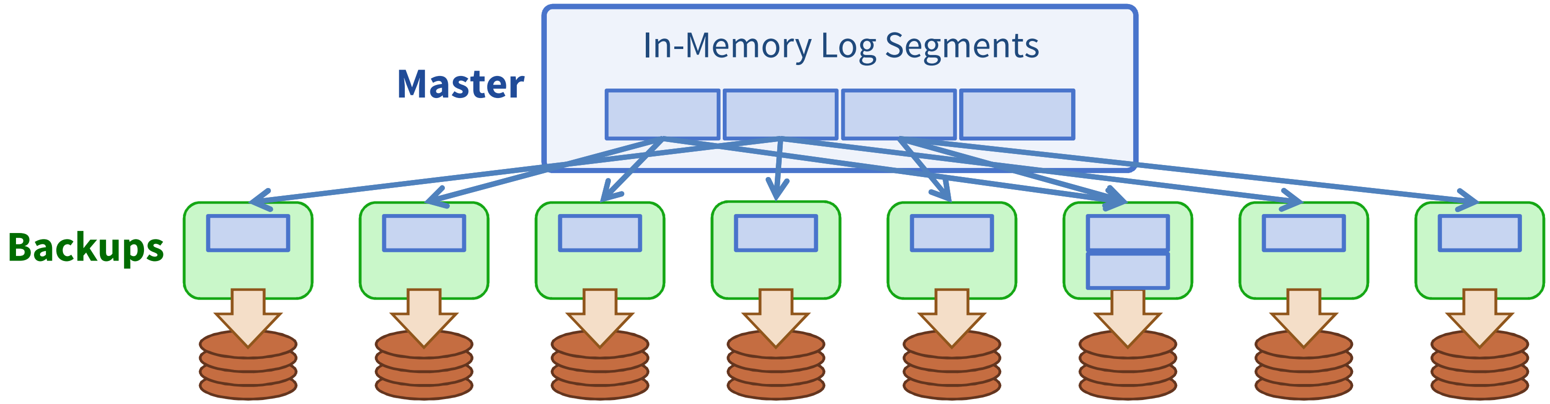
- Key to fast recovery:

  use system scale

Scatter Segments

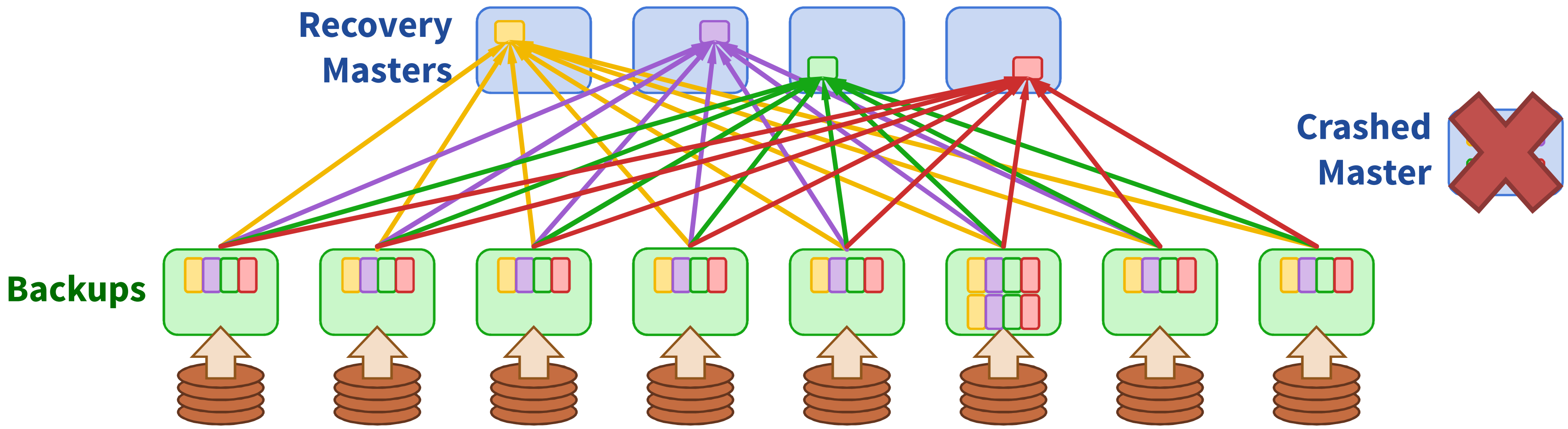
- Each log divided into 8MB **segments**
- Master chooses different backups for each segment
- Segments scattered randomly across all servers

- During recovery:
  - All backups read from disk in parallel
  - 64 GB / (1000 backups * 100 MB/s/backup) = **0.6 seconds**

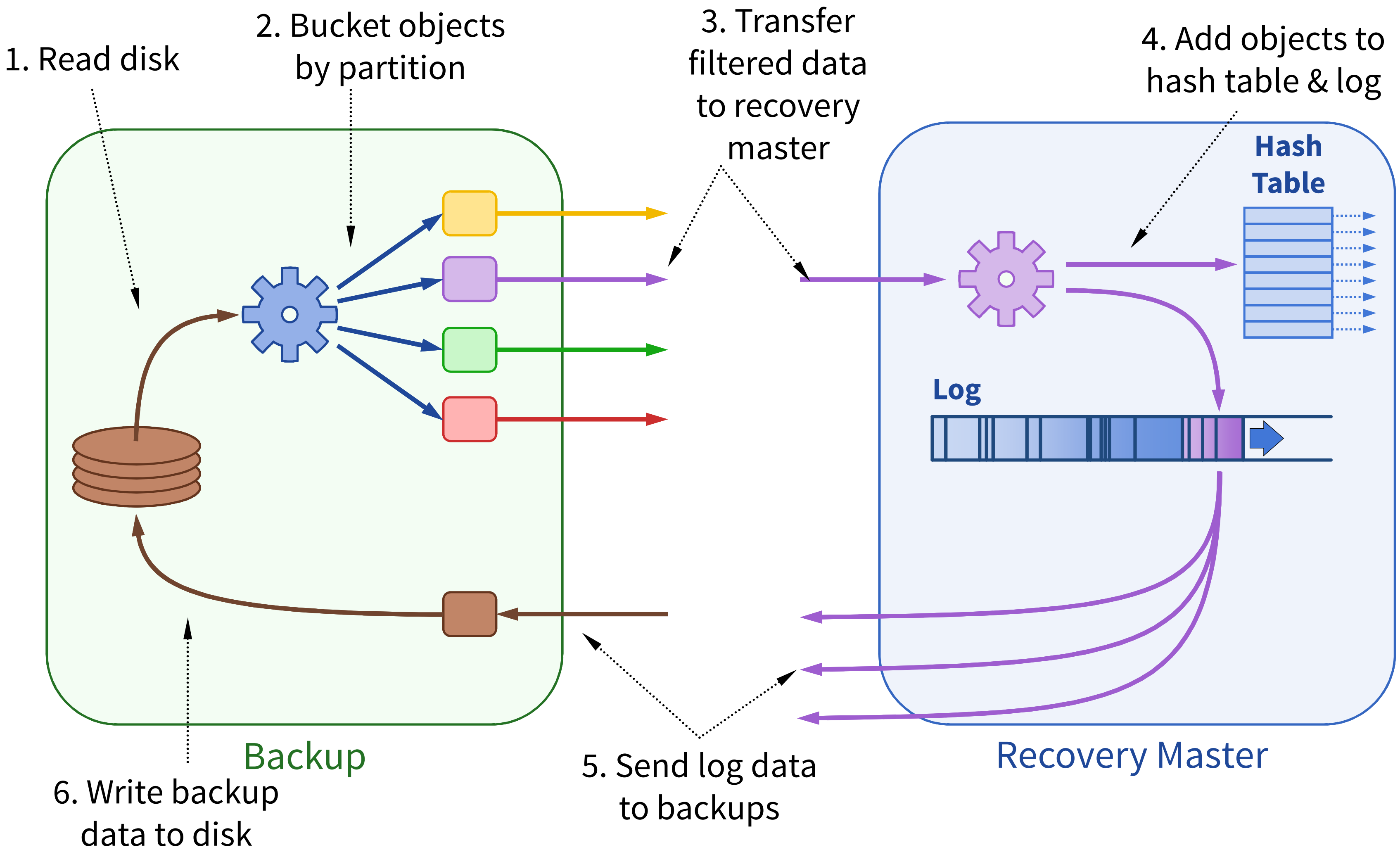


Partitioned Recovery

- Divide each master's data into **partitions**
  - Each partition groups a fraction of crashed key space
  - 64 GB / (100 masters * 1 GB/s/master) = **0.6 seconds**
  - Recover each partition on separate Recovery Master



Parallelism in Recovery



Recovery is both data parallel and tightly pipelined

Issues

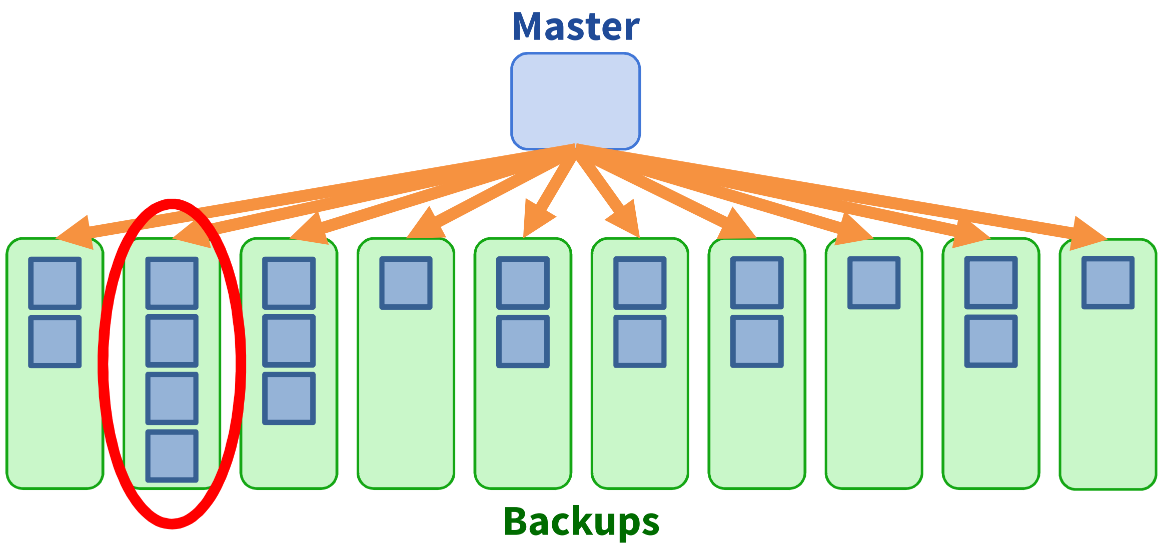
- Eliminating stragglers
  - **Randomized segment scattering**
  - Balancing partitions
- Finding log segments
  - Verifying log integrity
- Maximizing concurrency
  - Out-of-order log replay
- Fast failure detection
- Failures during recovery
- Consistency/Zombie masters

Randomized Segment Scattering

- **Randomizing** replica locations has many benefits
  - Avoids centralized allocation: 100k+ segments/second
  - Avoids hotspots; spreads request load
  - Balances total replicas on each disk
  - Spreads replicas from each master across all disks

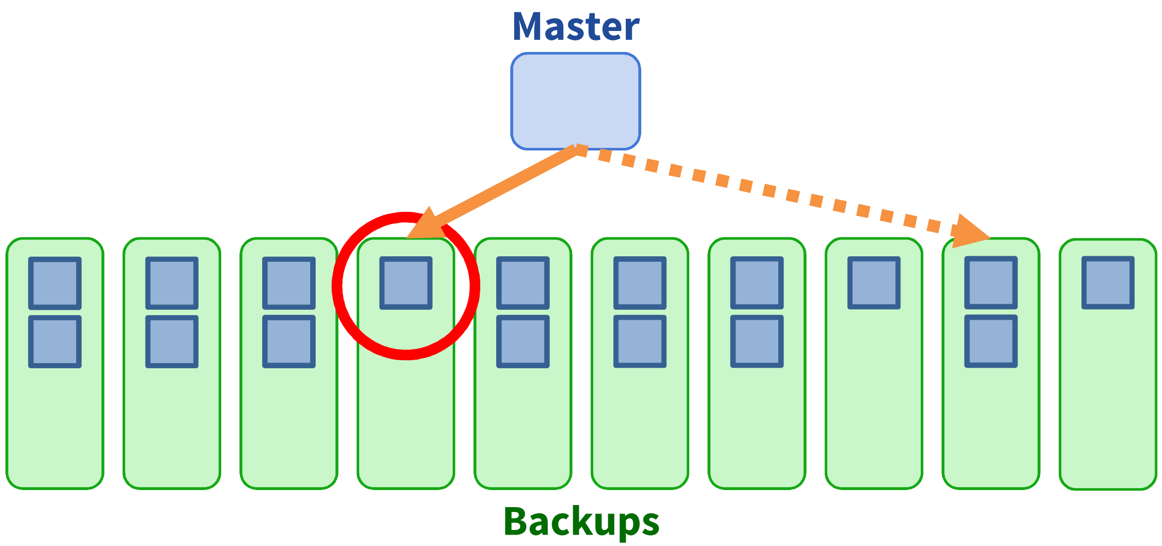
Uniform Randomization Doesn't Work

- **Problem:** balancing time reading across disks
  - Recovery time determined by the slowest disk
- Random distribution has large variance
  - Frequently 2 to 3× more replicas on one backup than avg



Balancing Disk Read Times

- **Solution:** add small amount of choice
  - Analyzed in [Mitzenmacher 1996]
  - Choose candidate backups randomly
  - Select candidate that minimizes expected disk read time
  - Must take into account disk performance; **fan vibration**



Issues

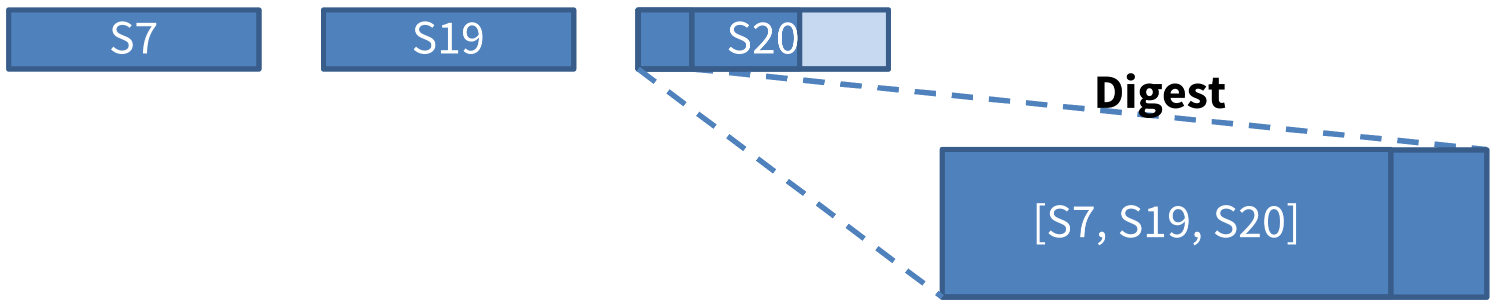
- Eliminating stragglers
  - Randomized segment scattering
  - Balancing partitions
- **Finding log segments**
  - **Verifying log integrity**
- Maximizing concurrency
  - Out-of-order log replay
- Fast failure detection
- Failures during recovery
- Consistency/Zombie masters

Finding Log Segment Replicas

- **Problem:** centralized segment catalog prohibitive
  - No centralized map of replica locations
  - No centralized list of which segments comprise a log
- **Solution:** ask all backups with broadcast
  - On master crash coordinator contacts each backup
  - Have to contact all backups anyway: scattered segments
  - Collects a list of all available replicas

Detecting Incomplete Logs

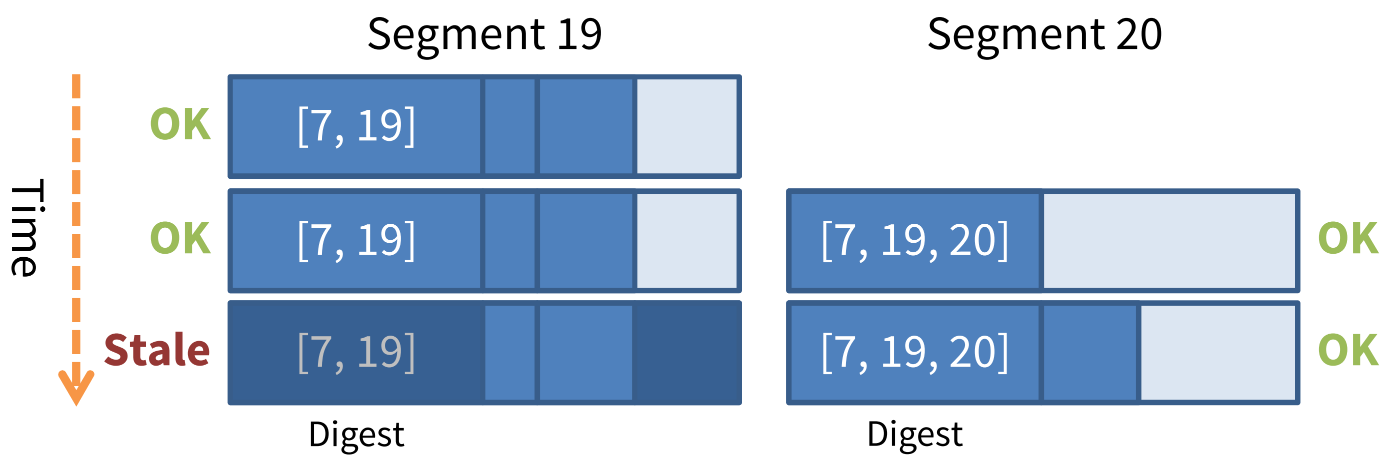
- **Problem:** ensure **complete log** found on recovery
  - What if all replicas for some segment are missing?
- **Solution:** make log self-describing
  - Add a “log digest” to each replica when it is allocated
  - Lists all segments of the log



- Reduces problem to finding up-to-date log digest

Choosing an Up-to-date Digest

- **Solution:** mark the most recent log digest
  - Whenever a new digest is created it is marked
  - Clear mark when last modification is made to segment
- **Challenge:** ensure safe transition between digests
  - Mark new digest before clearing mark on the old one
  - Two may be marked; only add data after just one marked



Issues

- Eliminating stragglers
  - Randomized segment scattering
  - Balancing partitions
- Finding log segments
  - Verifying log integrity
- Maximizing concurrency
  - Out-of-order log replay
- Fast failure detection
- Failures during recovery
- Consistency/Zombie masters

Outline

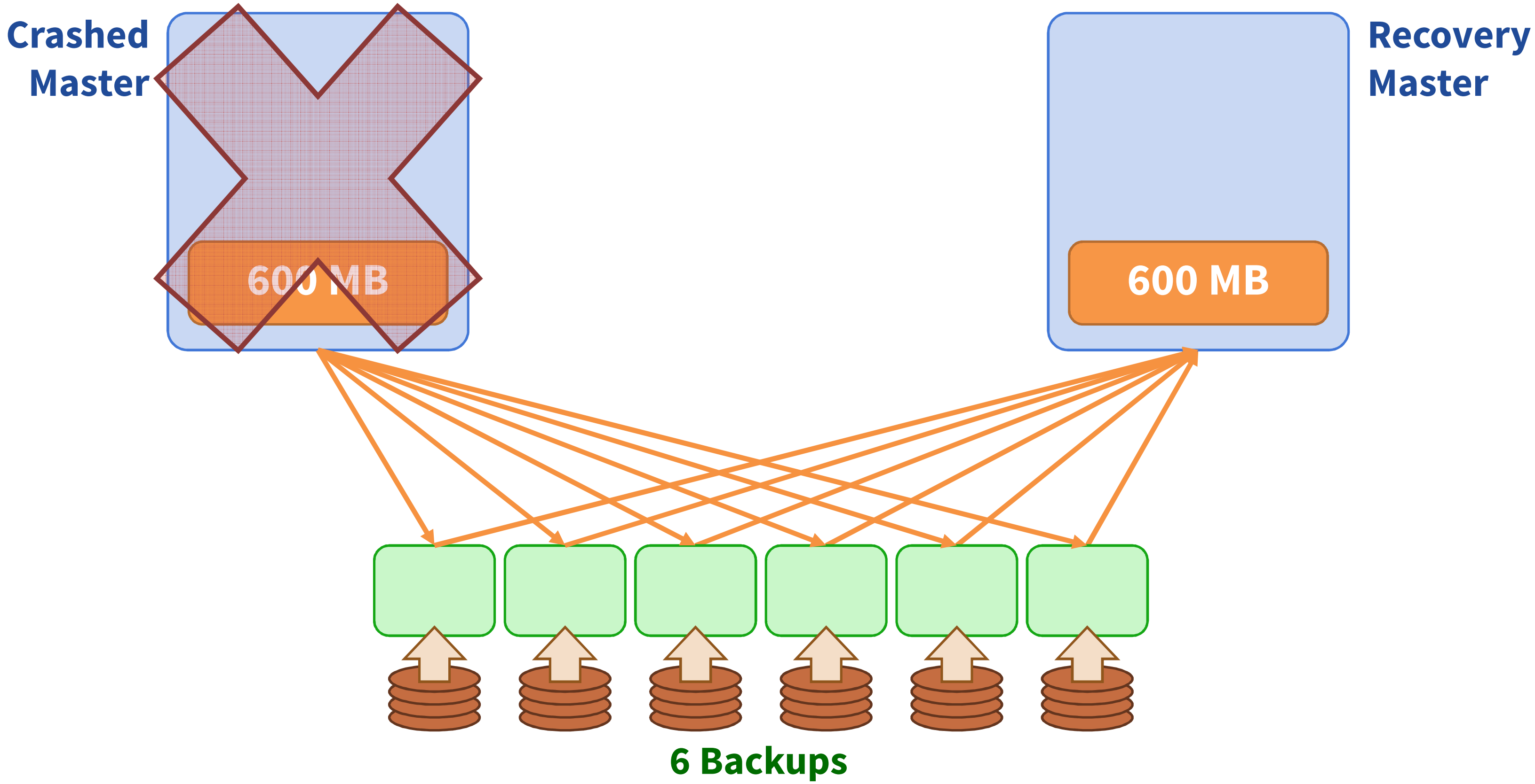
- Introduction
- Contributions
- RAMCloud Overview
- Fast Crash Recovery
  - Evaluation
- Distributed Fault-Tolerant Log
- Surviving Massive Failures
- Conclusion

Experimental Setup

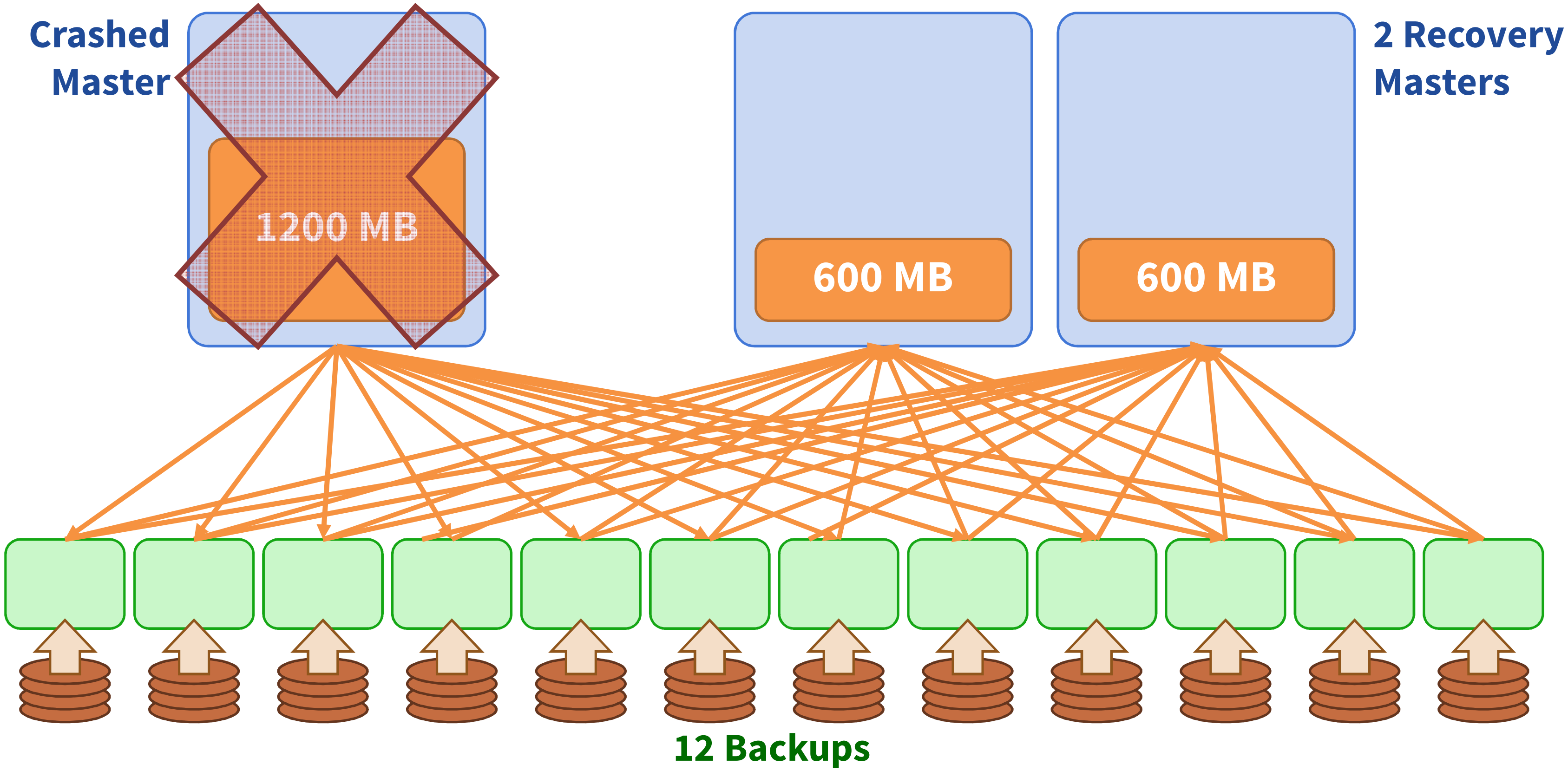
Cluster Configuration
60 Machines
2 Disks per Machine (100 MB/s/disk)
Mellanox Infiniband HCAs (25 Gbps, PCI Express limited)
5 Mellanox Infiniband Switches Two layer topology Nearly full bisection bandwidth

- Approximation for datacenter networks in 3-5 years
- 5.2 μs round trip from 100 B read operations

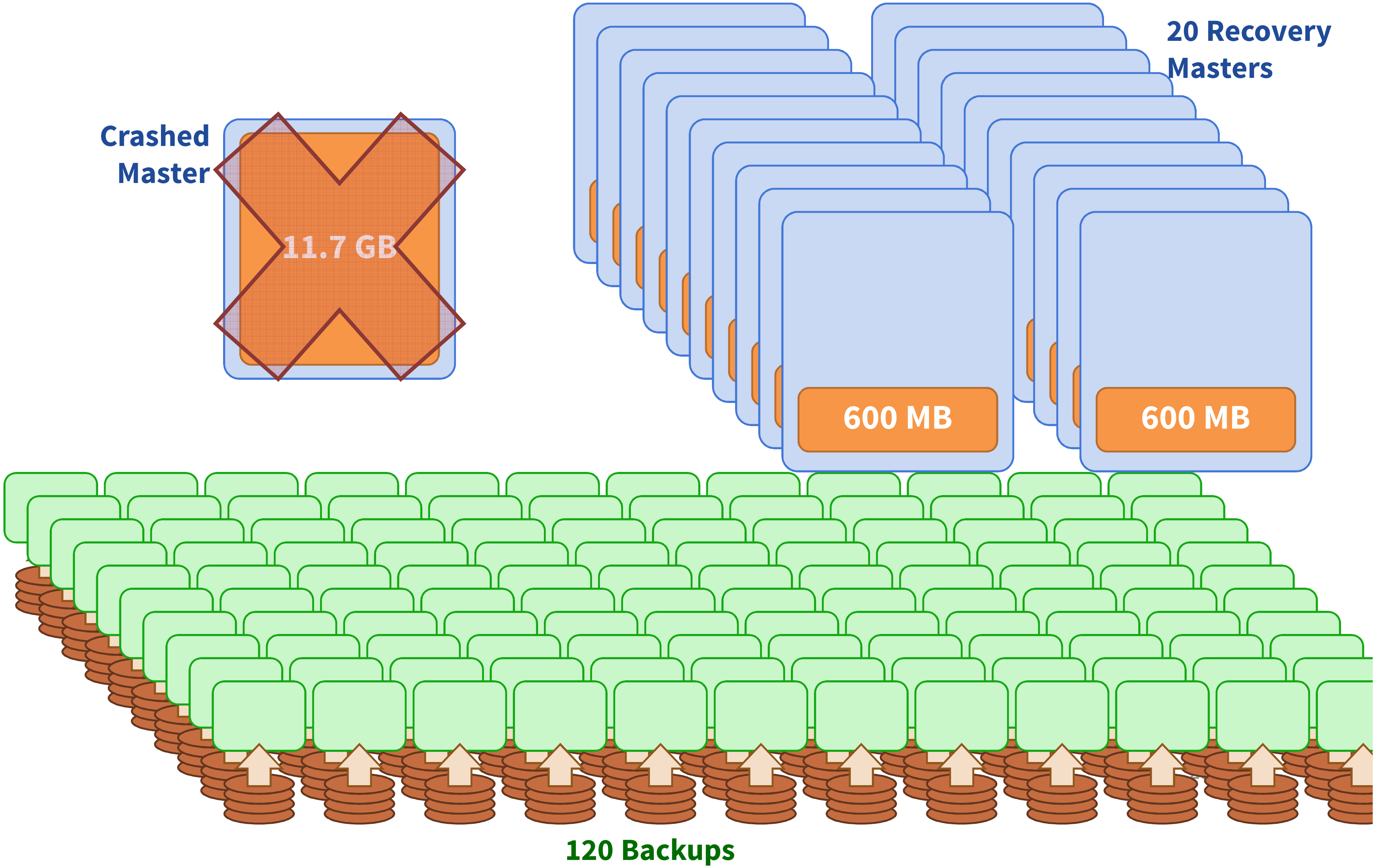
How well does recovery scale?



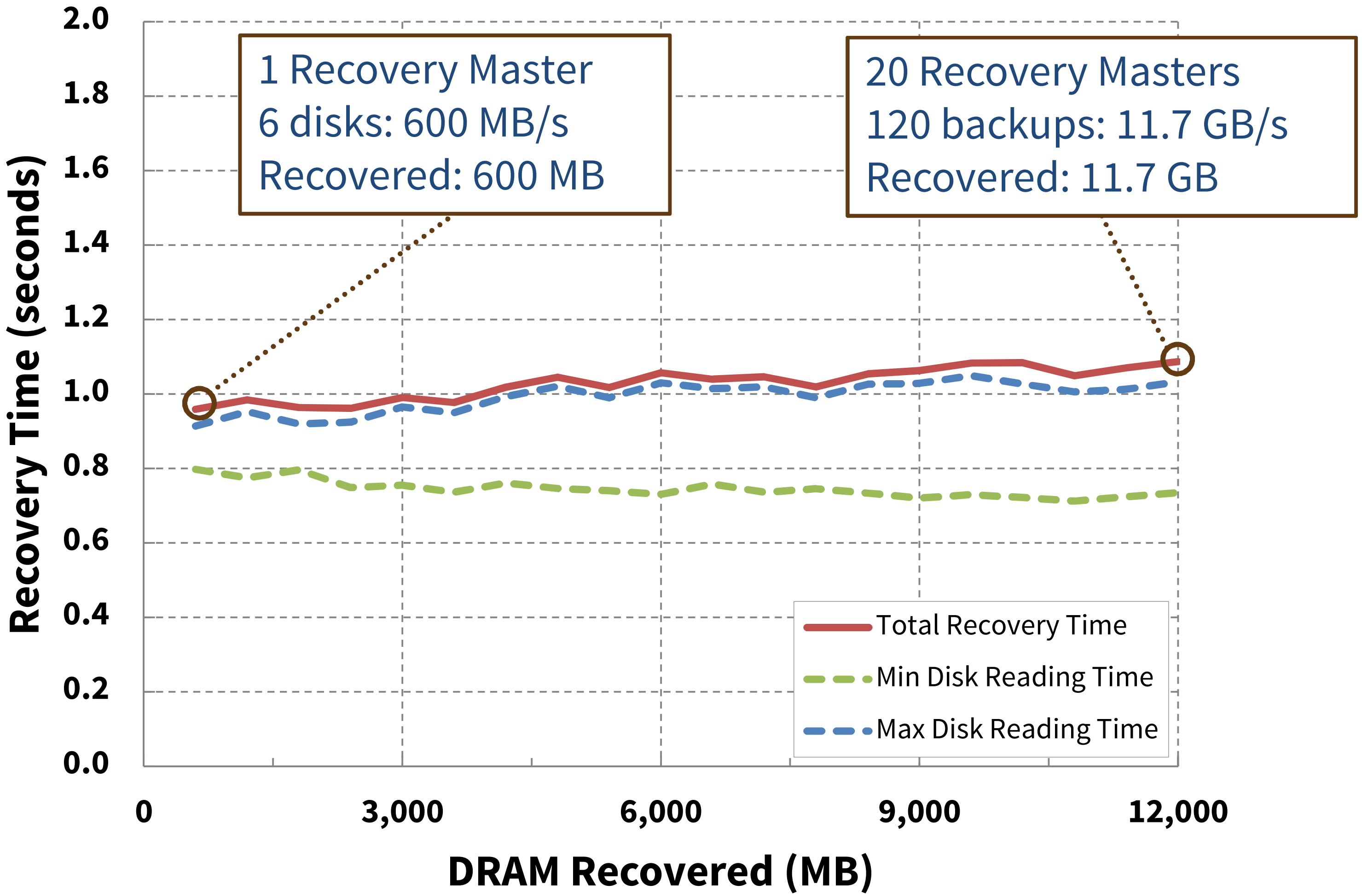
How well does recovery scale?



How well does recovery scale?

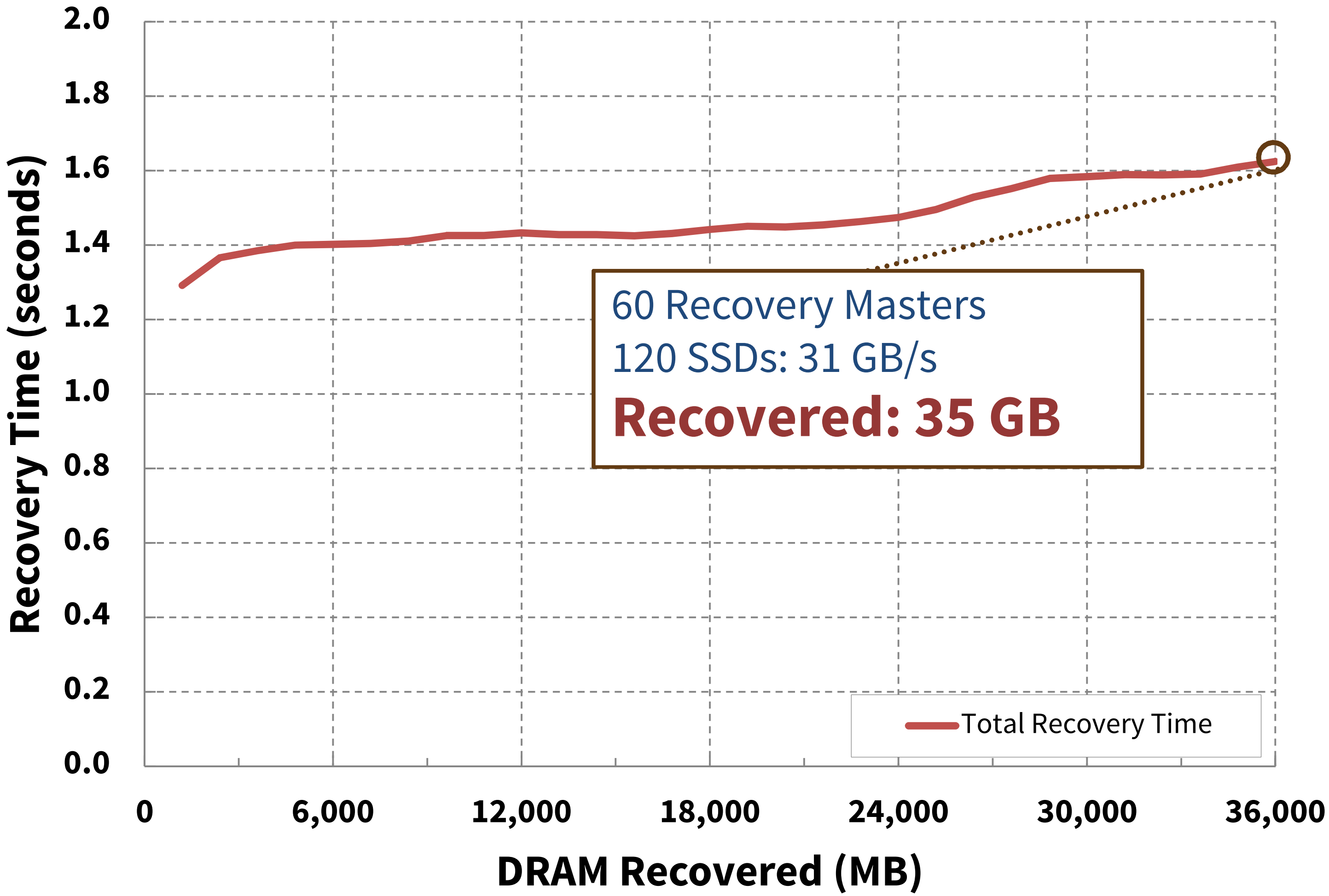


How well does recovery scale?



Recovery well pipelined; all disks active > 75% of the time

Flash Allows Higher Scalability



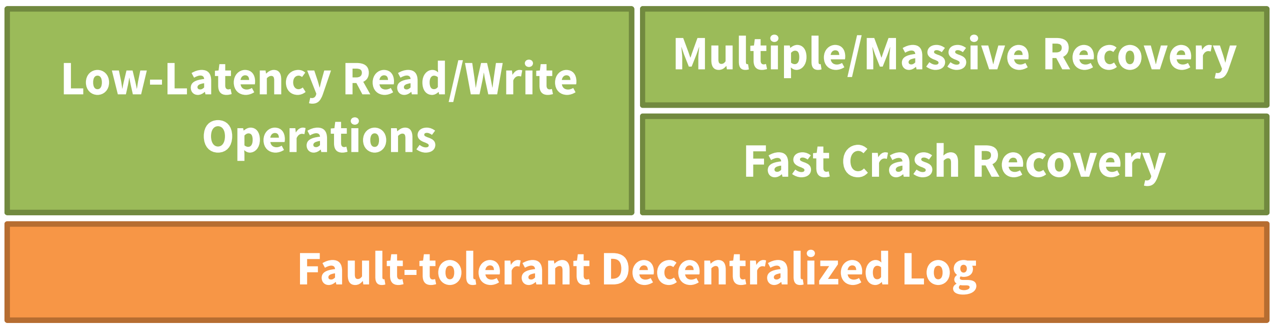
2x270 MB/s SSDs per recovery master
(vs. 6x100 MB/s disks per recovery master)

Outline

- Introduction
- Contributions
- RAMCloud Overview
- Fast Crash Recovery
    - Evaluation
- Fault-Tolerant Decentralized Log
- Surviving Massive Failures
- Conclusion

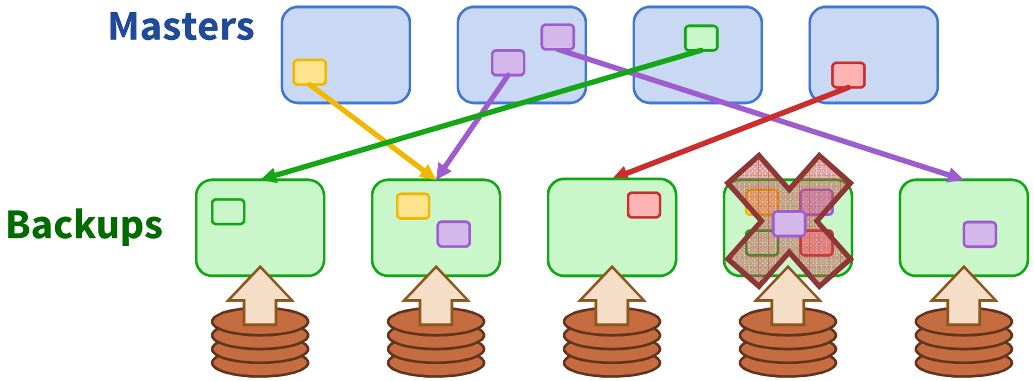
Fault-tolerant Decentralized Log

- **Fast and cheap durability in normal case**
  - Eliminates synchronous disk writes
- **High read bandwidth for fast crash recovery**
- **Scalable; avoids centralization**
  - Even on segment transitions



- But, how do **failures** impact logs?

Distributed Replica Recreation



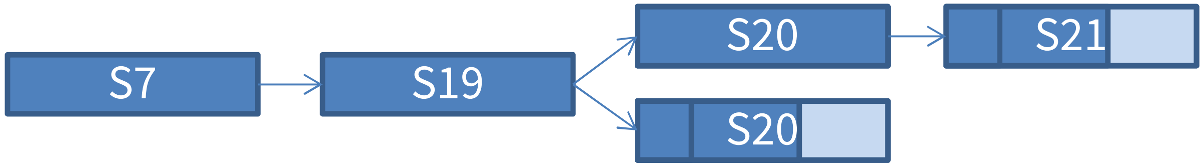
- **Problem:** failures accumulate into loss of segments
- **Solution:** recreate replicas to restore redundancy
  - Simple: master uses same approach as normal operation
  - Efficient: use master's in-memory copy
  - Concurrent: work is divided up among all the hosts
- Faster than master recovery, but lower priority

Restarting After Widespread Failure

- **Problem:** widespread correlated failures may lose all replicas of some segments
  - Power outages, loss of core switch, coordinated segfaults
  - Rare events, a few times a year; usually temporary
  - Unavailability ok until problem subsides
- **Solution:** wait for servers to rejoin with replicas
  - Perform recovery when all segments available for server

Inconsistent Replicas

- **Problem:** some found replicas may be inconsistent
  - Master can't wait for backup to rejoin to add data to log
  - Master can't clear log digest mark on lost head replicas



  - Cannot solve with log alone and remain available
- **(Sketch of) Solution:** must centralize some state
  - Prevent inconsistency of most replicas: atomic recreation
  - Minimize centralized state for inconsistent head replicas
  - Update log head version on coordinator on inconsistency
  - Rare, expect 6 updates/hr, constant 16 bytes of state/log

Outline

- Introduction
- Contributions
- RAMCloud Overview
- Fast Crash Recovery
  - Evaluation
- Fault-Tolerant Decentralized Log
- Surviving Massive Failures
- Conclusion

Coordinating Restart/Massive Recovery

- Simplicity is goal on loss of many/all servers
  - Massive failures are expected to be rare
- Reuse master recovery
- Treat massive failure as a series of single failures
- Recoveries fail until a complete log is available
  - Retry recoveries round-robin

Outline

- Introduction
- Contributions
- RAMCloud Overview
- Fast Crash Recovery
    - Evaluation
- Fault-Tolerant Decentralized Log
- Surviving Massive Failures
- Conclusion

Key Design Principles

- **Make scale your friend**
  - Failures are frequent at scale
  - But, scale provides resources for solving problems; crash recovery
- **Cannot design for the “common” case**
  - “Rare” corner cases happen frequently
  - Code must always be ready to adapt to failures
- **Pervasive randomization**
  - Decentralized decision-making, avoids pathologies; use carefully
- **Collapse error handling**
  - Fast recovery simplifies the system
  - When in doubt, crash
  - DRAM corruption and more

Related Work

Large-scale DRAM storage

- Large-scale memcached [NSDI 13]
  - Apps must deal with backing store and consistency
  - Reduced performance from misses, cold caches

Fast recovery for availability

- Fast Crash Recovery in Distributed File Systems [Baker 94]
  - Reconstruct server's view of client cache state
  - Restart quickly enough for continuous availability

Related Work

Fast updates by buffering in non-volatile memory

- POSTGRES [VLDB 87]

Log-structured storage

- Log-structured Filesystem (LFS) [SOSP 91]
  - Filesystem interface on a single disk
  - RAMCloud keeps log in-memory and on disk
- GFS [SOSP 03]
  - Fault-tolerant log via replicated chunks throughout cluster
  - Buffers writes in buffer cache
  - Centralized metadata server to allocate chunks
  - Supports Bigtable [OSDI 06], primarily disk-based DB

Conclusions

- **RAMCloud:** large-scale datacenter storage in DRAM
  - **Scale:** Up to 10,000 servers, 1+ PB capacity
  - **Low latency:** 5 μs remote access, 1M ops/s/server
  - 1000× faster than disk-based storage systems
- **Impact:** more data-intensive applications

- **Durability:** fault-tolerant decentralized log
- **Availability:** scalable fast crash recovery
- **Result:** **DRAM as reliable as replicated disk storage without performance or cost penalties**

Contributions

DRAM as reliable as replicated disk storage without performance or cost penalties

Scalable Crash Recovery

- Availability through fast recovery rather than redundancy
- Leverages scale to restore an entire server's DRAM in 1 to 2 s

Fault-tolerant Decentralized Log Structure

- Provides fast writes by eliminating disk accesses
- Retains consistency and restores durability after failures
- Scales by avoiding centralization

Thanks

- Christine, Mendel, Christos
- Mikhail Atallah, David, John
- Nickolai, Daniel, Jad, Mike Walfish
- Steve, Diego, Ankita, Satoshi
- Mom, Dad


BOSTON


Hello Kitty
TINY LOVE

Contributions

DRAM as reliable as replicated disk storage without performance or cost penalties

Scalable Crash Recovery

- Availability through fast recovery rather than redundancy
- Leverages scale to restore an entire server's DRAM in 1 to 2 s

Fault-tolerant Decentralized Log Structure

- Provides fast writes by eliminating disk accesses
- Retains consistency and restores durability after failures
- Scales by avoiding centralization