Durability and Crash Recovery for Distributed In-Memory Storage

Ryan Stutsman,

Asaf Cidon, Ankita Kejriwal,

Ali Mashtizadeh, Aravind Narayanan,

Diego Ongaro, Stephen M. Rumble,

John Ousterhout, and Mendel Rosenblum

RAMCloud Overview

- **RAMCloud: General purpose storage in RAM**
  - Low latency: **5 μs** remote access
  - Large scale: 10,000 nodes, 100 TB to 1 PB
- **Enable applications which access data intensively**
  - Access 100 to 1000x times more data
- **Simplify development of large scale applications**
  - Eliminate issues that kill developer productivity
    - Partitioning, parallelism/pipelining, locality, consistency
- **Key Problem: RAM’s lack of durability**

Thesis

Large-scale RAM-based storage must be durable and available for applications to gain its full benefits, and it can be made so cost-effectively using commodity hardware.

- **Solution**
  - Fast (1 - 2s) crash recovery for availability
  - Restoration of durability after crashes
  - Survival of massive failures (with loss of availability)

Durability Requirements

- **Minimal impact on normal case performance**
- **Remain available during typical failures**
- **Remain durable during massive/correlated failures**
- **No undetected data loss**
- **Low cost, low energy**
    - Use commodity datacenter hardware available in a few years

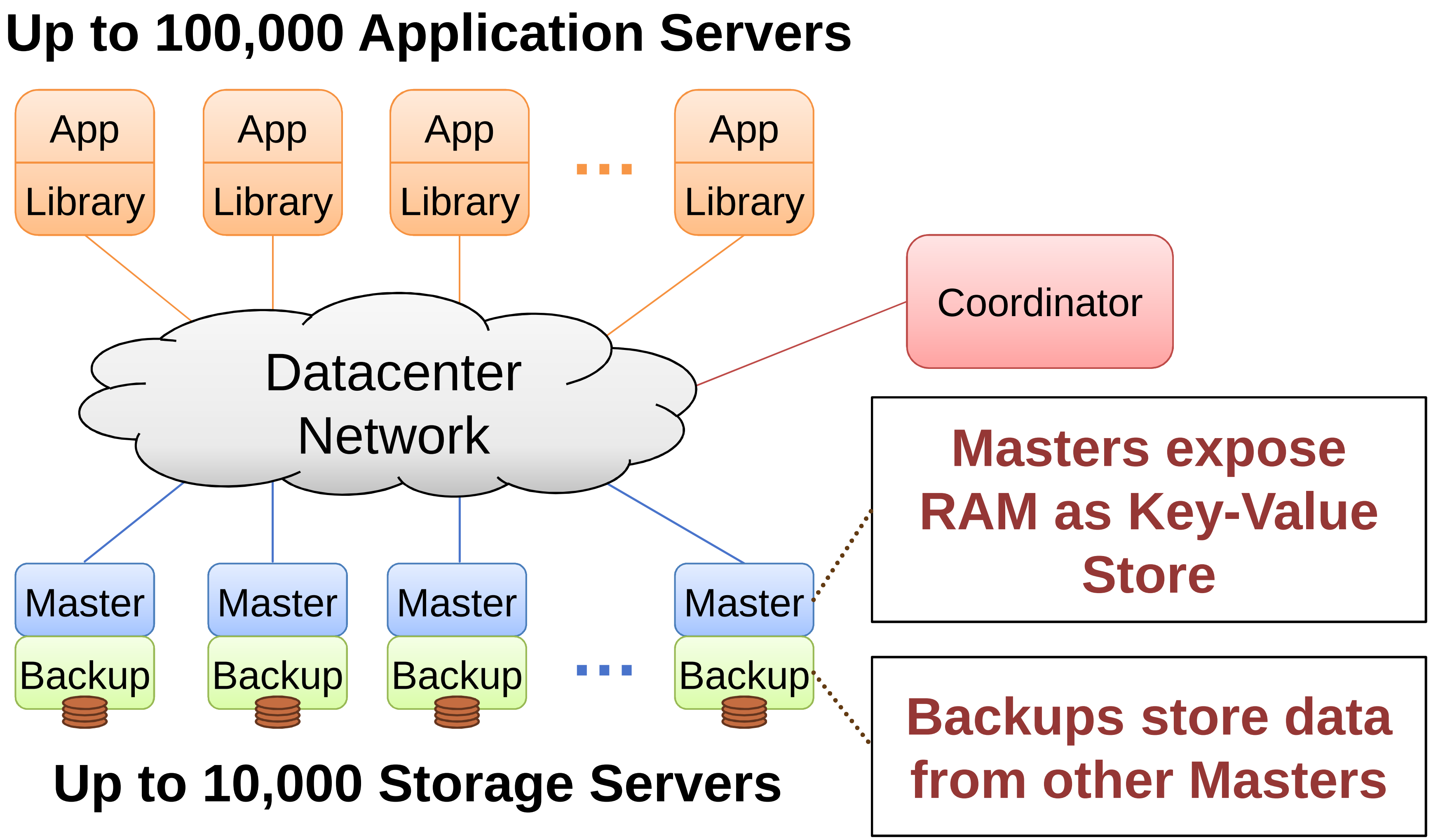
The performance of RAM-based storage

with durability of traditional datacenter storage

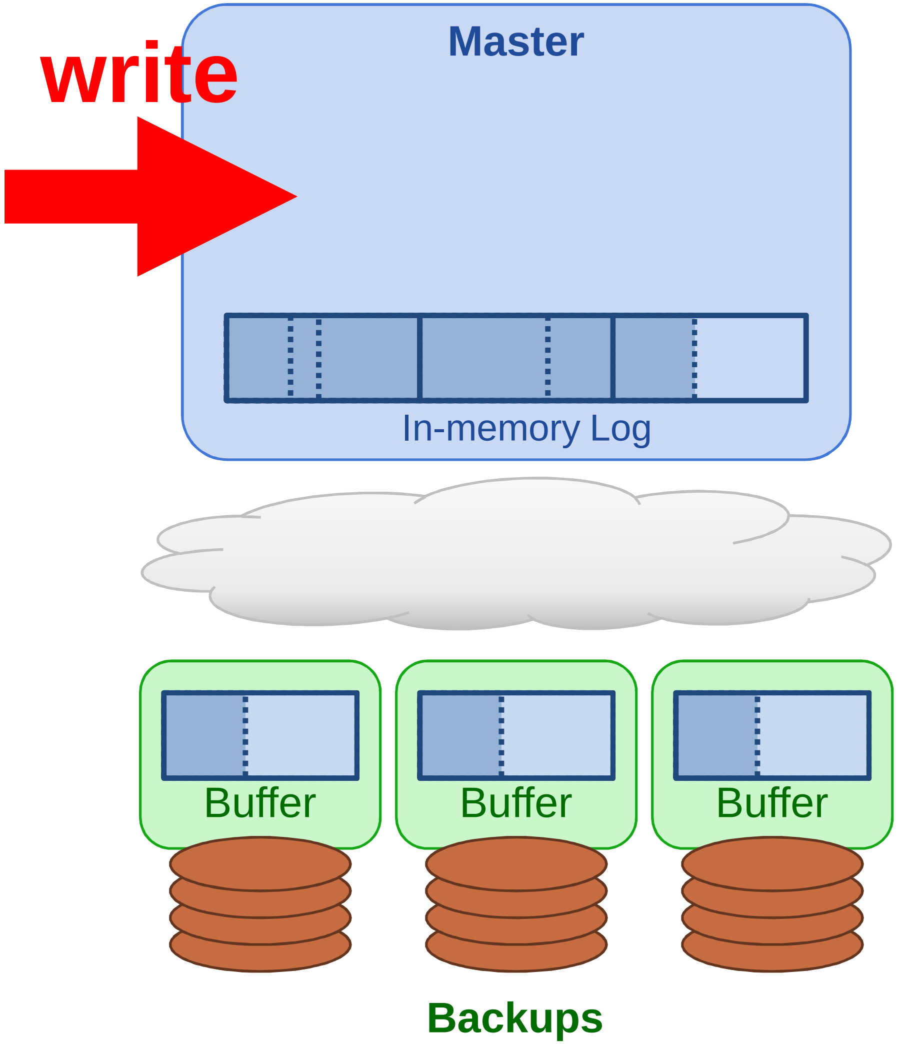
Outline

- **Master Recovery**
- **Backup Recovery**
- **Massive Failures/Cluster Restart**
- **Distributed Log**
  - Key structure underlying RAMCloud

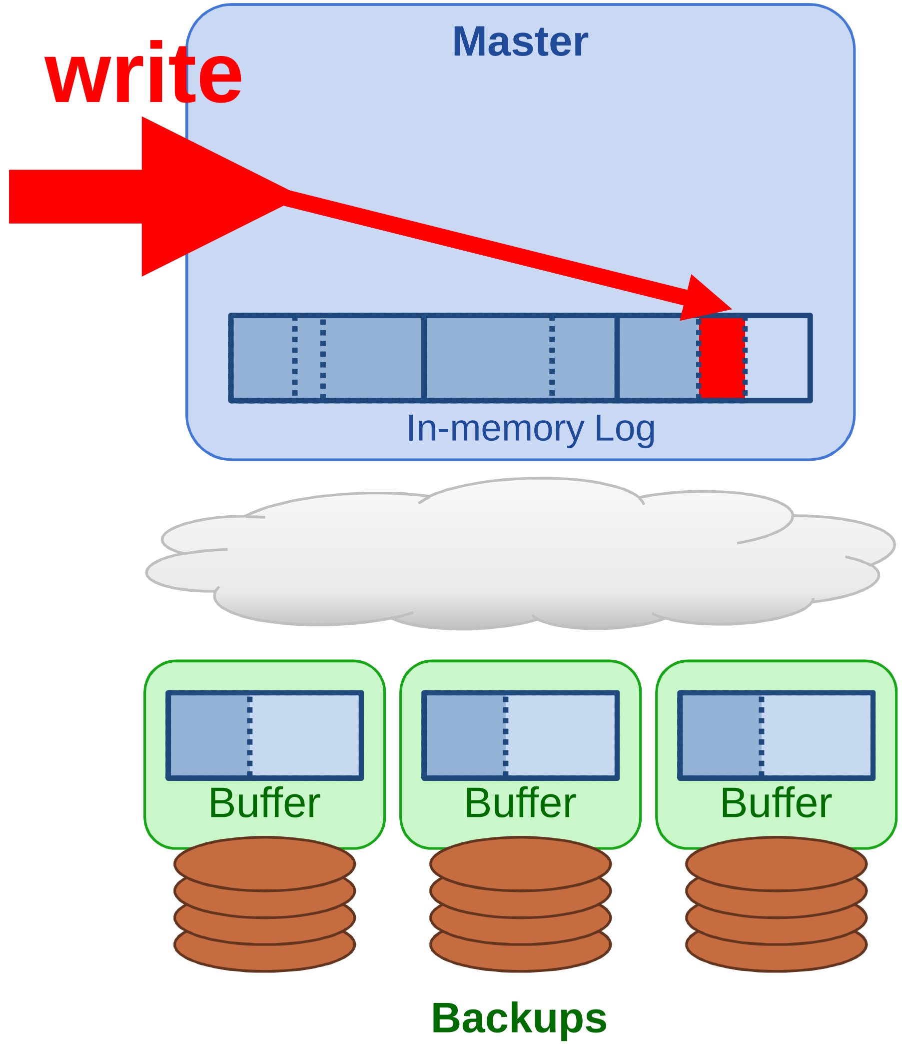
RAMCloud Architecture



Normal Operation

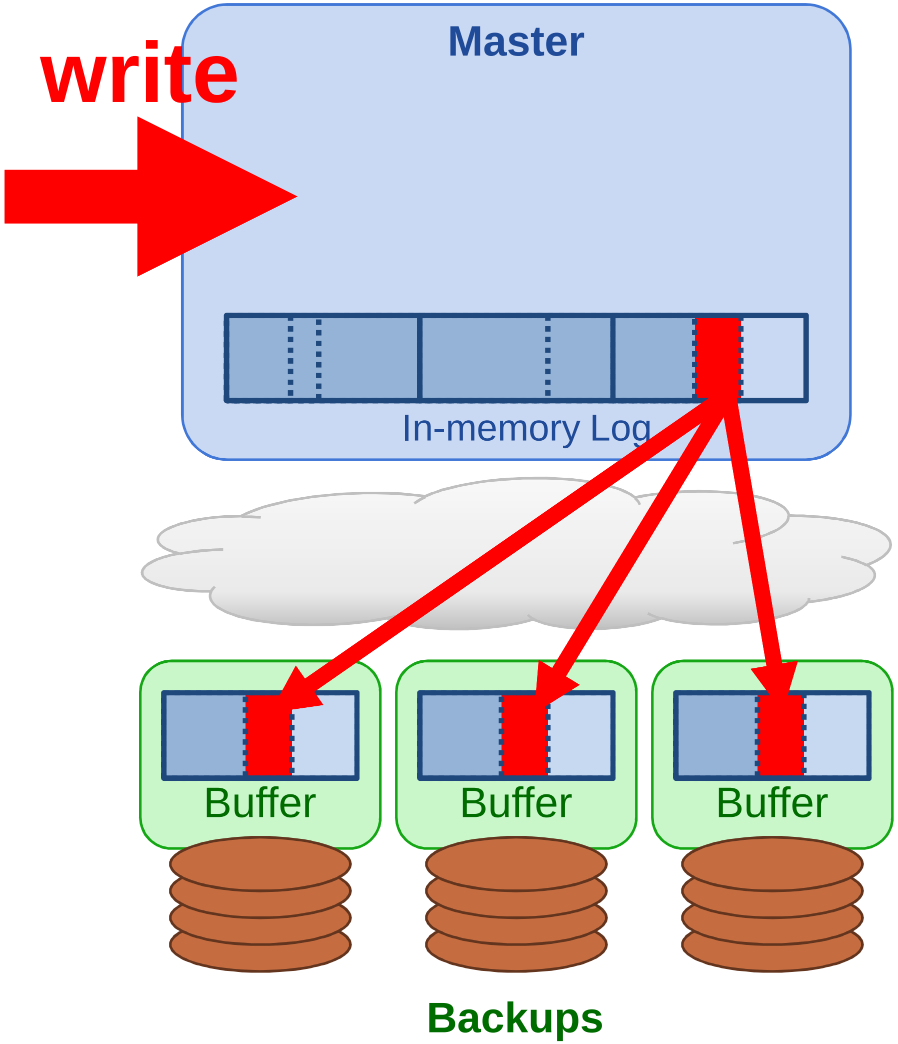


Normal Operation



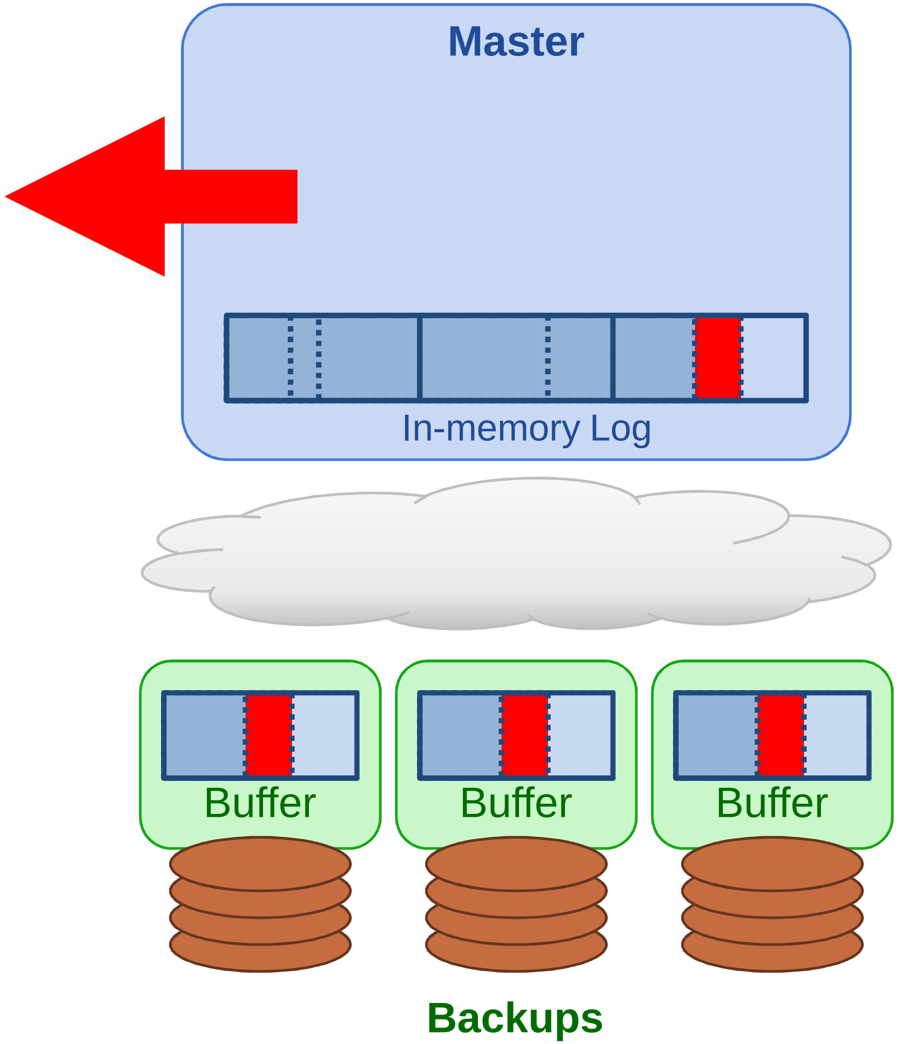
Normal Operation

- **Backups buffer update**
  - No synchronous disk write
  - Must flush on power loss



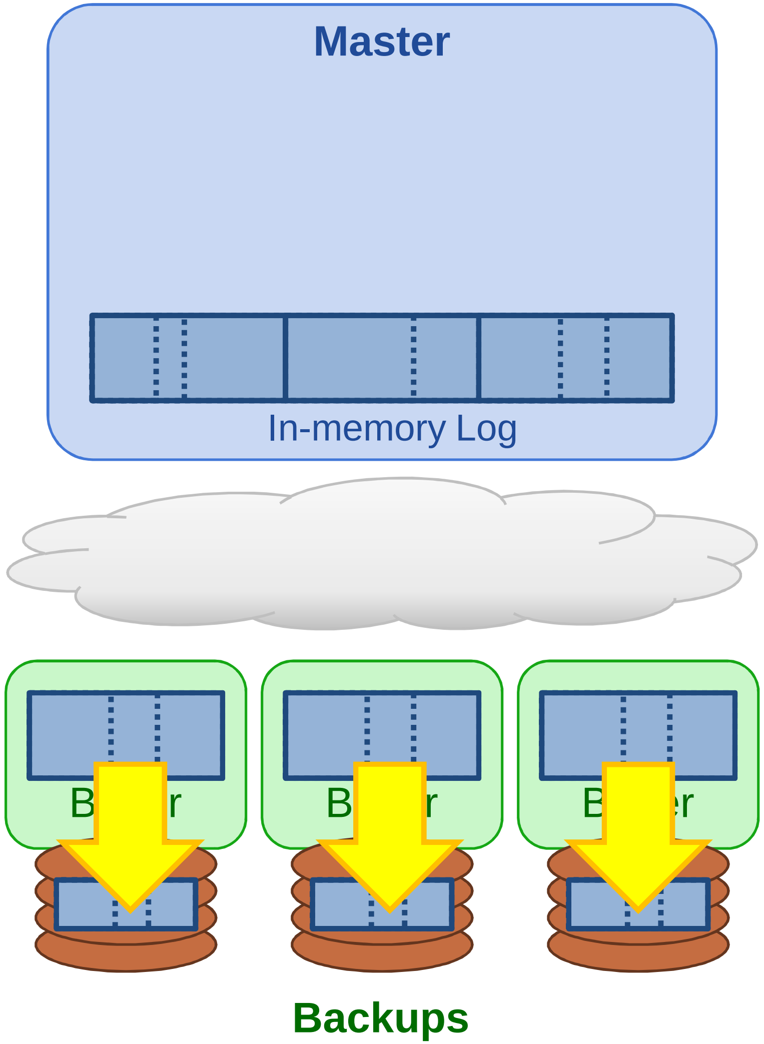
Normal Operation

- **Backups buffer update**
  - No synchronous disk write
  - Must flush on power loss



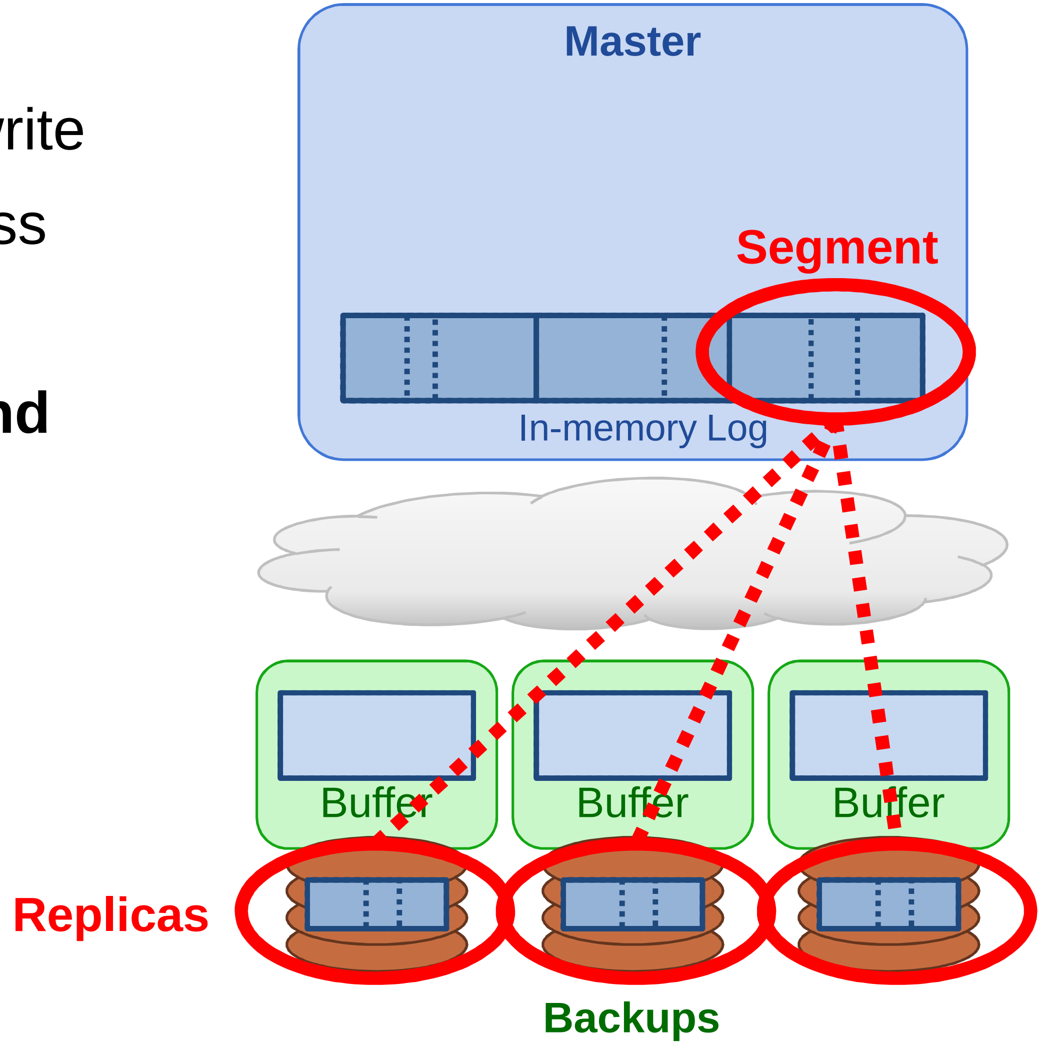
Normal Operation

- **Backups buffer update**
  - No synchronous disk write
  - Must flush on power loss
- **Bulk writes in background**



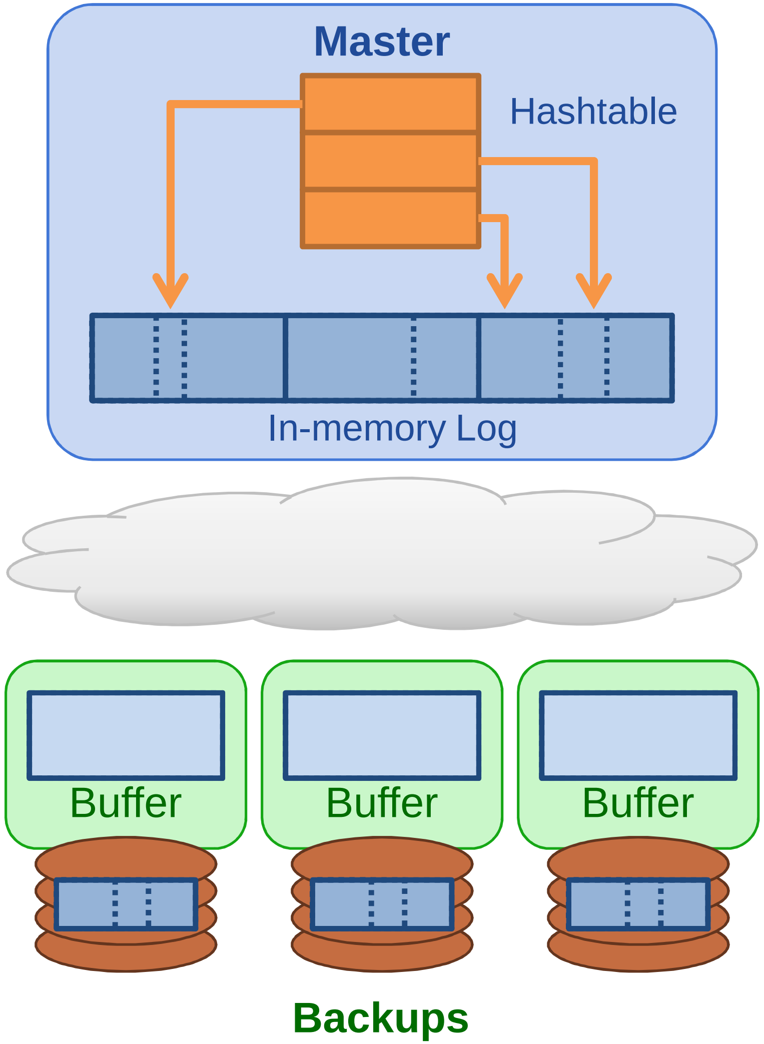
Normal Operation

- **Backups buffer update**
  - No synchronous disk write
  - Must flush on power loss
- **Bulk writes in background**
- **Pervasive log structure**
  - Even RAM is a log
  - Log cleaner



Normal Operation

- **Backups buffer update**
  - No synchronous disk write
  - Must flush on power loss
- **Bulk writes in background**
- **Pervasive log structure**
  - Even RAM is a log
  - Log cleaner
- **Hashtable, key → location**



Normal Operation Summary

✓ **Cost-effective**

- 1 copy in RAM
- Backup copies on disk/flash: durability ~ free!

✓ **Fast**

- RAM access times for all accesses
- Avoids synchronous disk writes

✗ **Non-volatile buffers**

✗ **Unavailability on crash**

Fast Crash Recovery

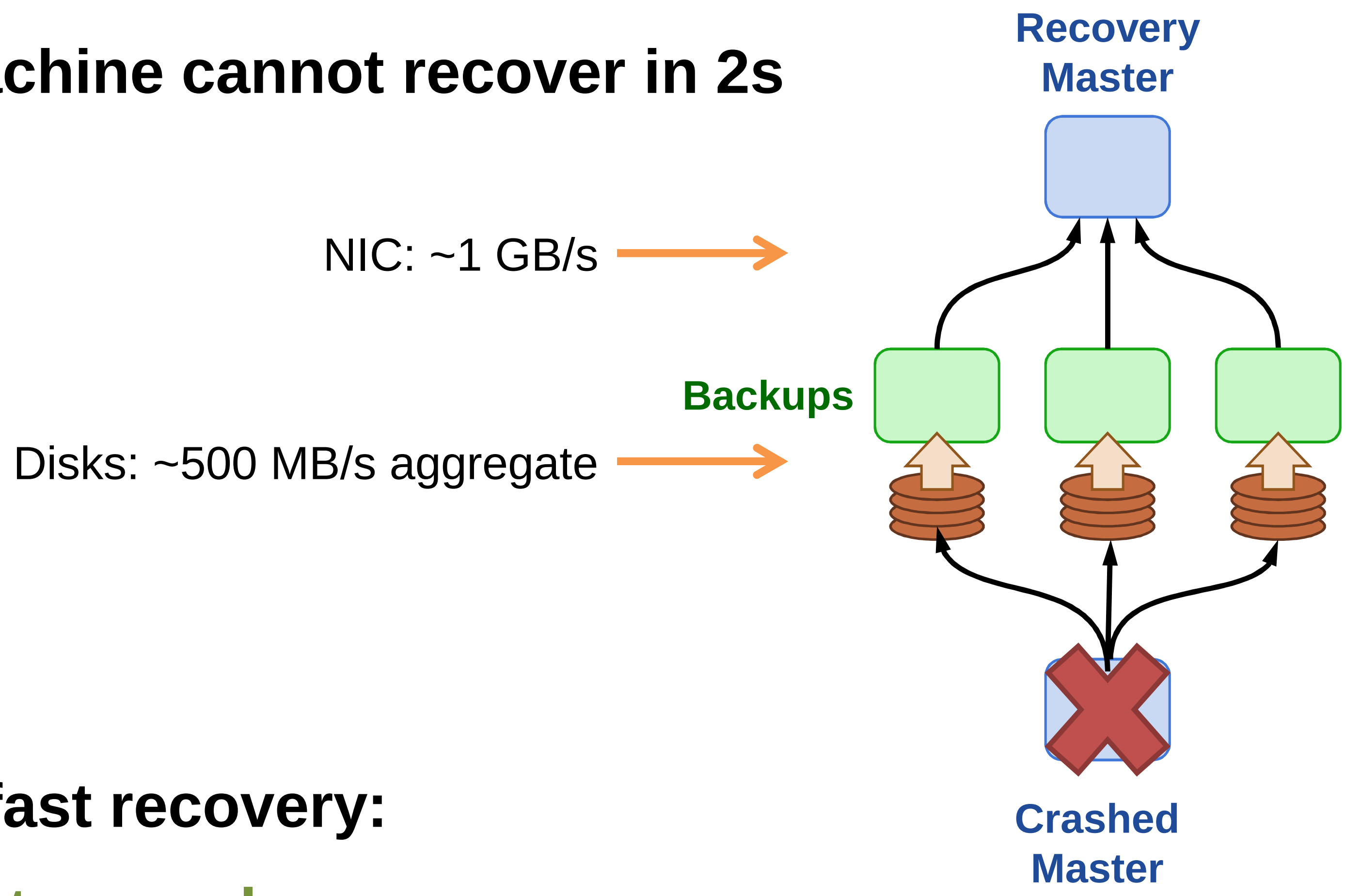
- **What is left when a Master crashes?**
  - Log data stored on disk on backups

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

- **Recover fast: 64 GB in 1-2 seconds**

Recovery Bottlenecks

- **One machine cannot recover in 2s**

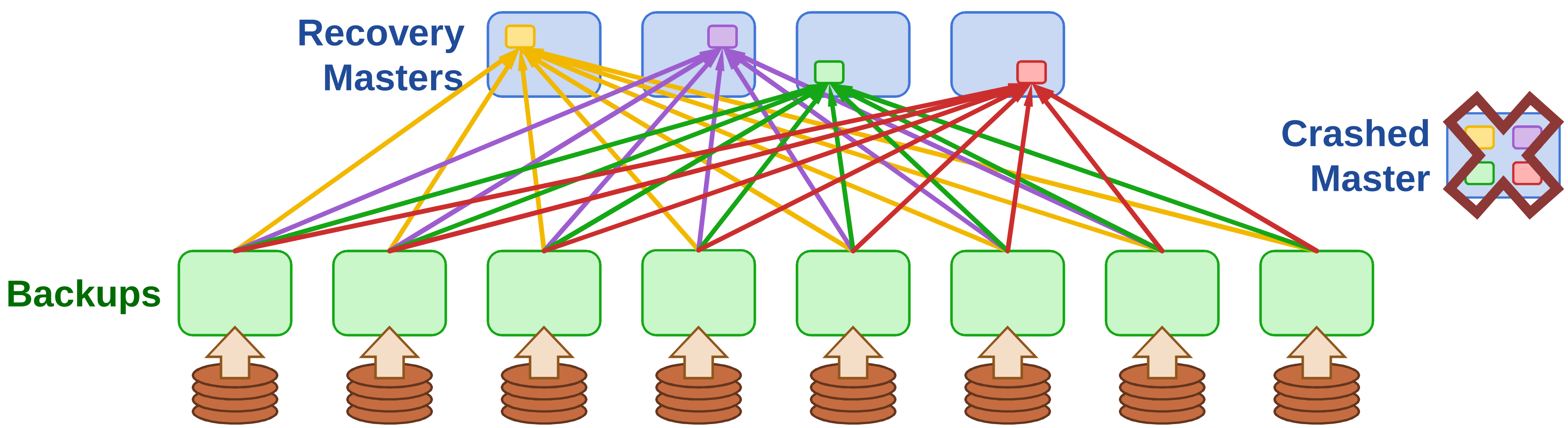


- **Key to fast recovery:**

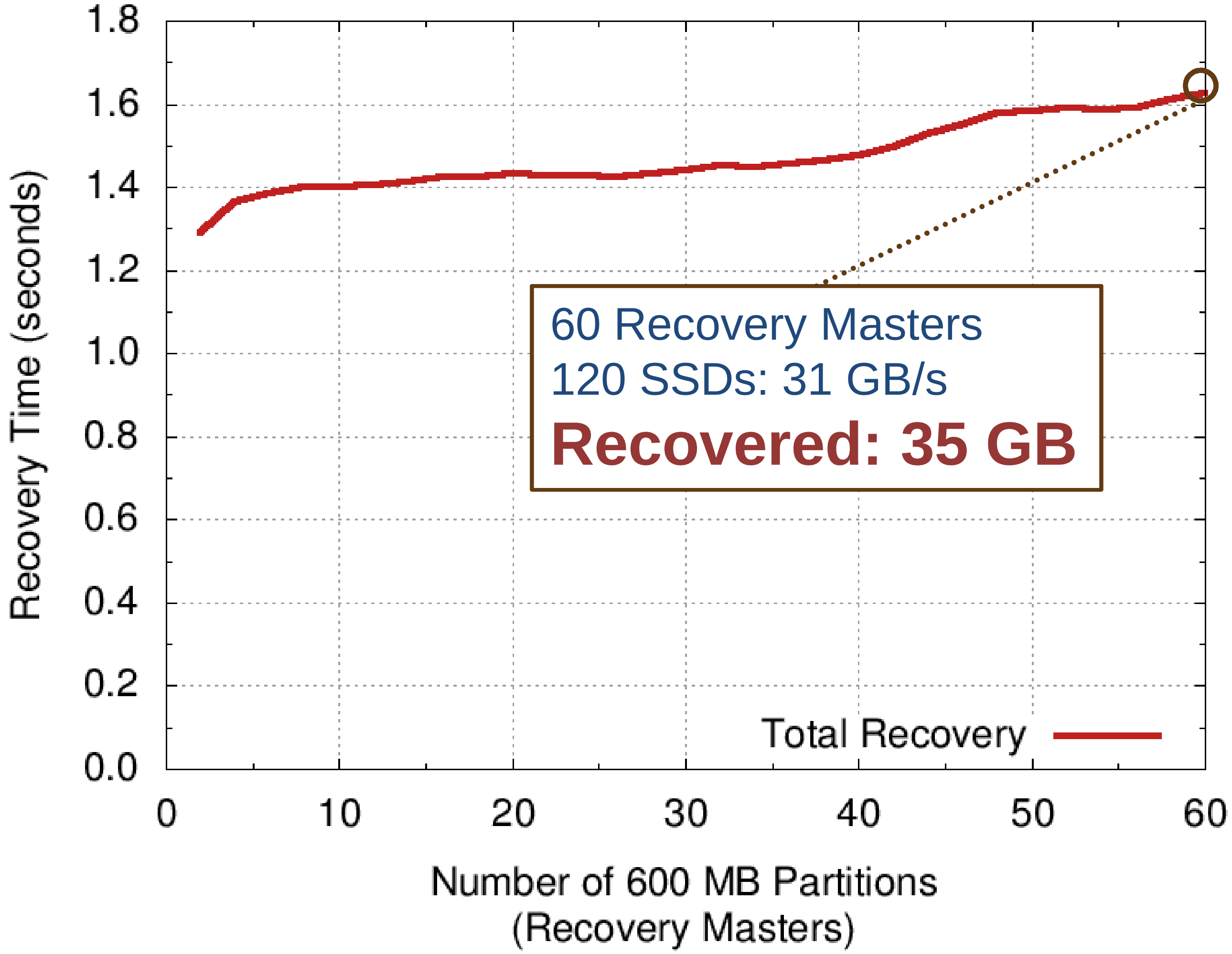
  Use system scale

Distributed Recovery

- **Scatter/read segments across all backups**
  - Solves disk bandwidth bottleneck
- **Recover to many masters**
  - Solves NIC and memory bandwidth bottlenecks



Results

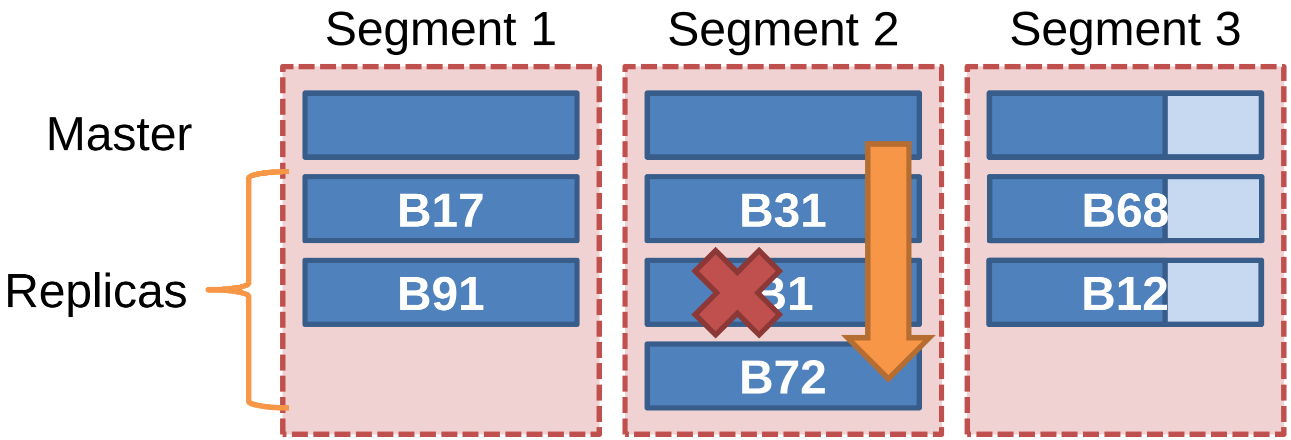


2x270 MB/s SSDs, 60 machines

Key Issues and Solutions

- **Maximizing concurrency**
  - Data parallelism
  - Heavily pipelined
    - Out-of-order replay
- **Balancing work evenly**
  - Balancing partitions
    - Divide crashed master up at recovery time
  - Segment scattering
    - Random, but bias to balance disk read time
- **Avoiding centralization**
  - Finding segment replicas
  - Detecting incomplete logs

Backup Recovery



- **Failures result in loss of segment replicas**
- **Recreate replicas to maintain desired redundancy**
  - Simple: Master uses same procedure as normal operation
  - Efficient: Use master's in-memory copy
  - Concurrent: Work is divided up among all the hosts

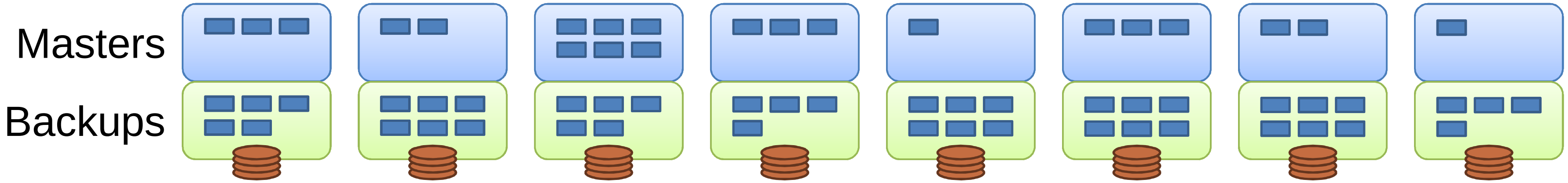
Expected Backup Recovery Time

- **Backup could have 192 GB of data**
  - 1000 64 GB masters triplicating segments
- **Aggregate network bandwidth: 1000 GB/s**
- **Aggregate disk bandwidth: 200 GB/s**
- **~200 ms if backups have buffer space**
- **~1,000 ms if not**

Massive Failures/Cluster Restart

- **Unavailability ok during massive/complete outages**
  - Remain durable
- **Simplicity is the goal**
  - Massive failures are expected to be rare
  - Shortest restart time is a non-goal (within reason)
- **Reuse master/backup recovery**
  - Must make them work correctly under massive failure
- **Treat massive failure as a series of single failures**

Massive Failures/Cluster Restart



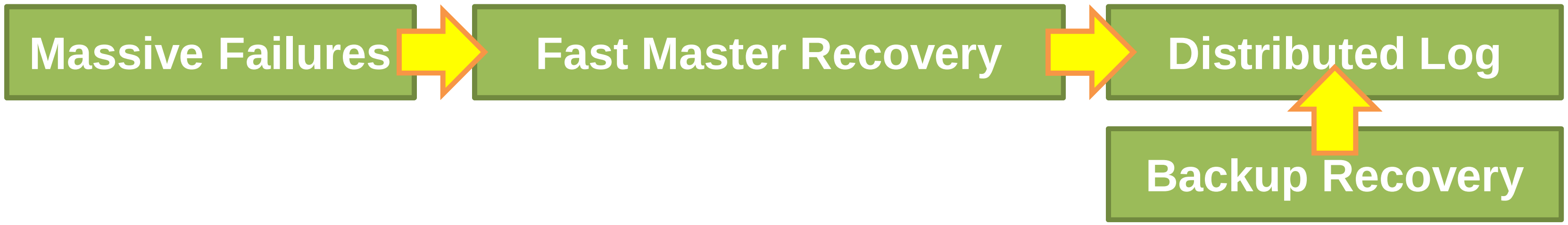
- **Must readmit replicas to cluster after mass failure**
  - All master data is lost
  - Only replicas on backups’ disks persist
- **Start recovery when complete log becomes available**
  - Determine if recovery can proceed as backups rejoin

Key Issues

- **Replica garbage collection**
  - Dilemma when a replica rejoins the cluster
    - It may be needed for a recovery
    - It may be redundant; master has created new replicas

Fault-tolerant Distributed Log

- **Distributed log**
  - Fast durability in normal case
  - Restores availability quickly during crashes
  - Clear strategy for maintaining durability



- **We've talked about how the log operates on a good day**

Potential Problems

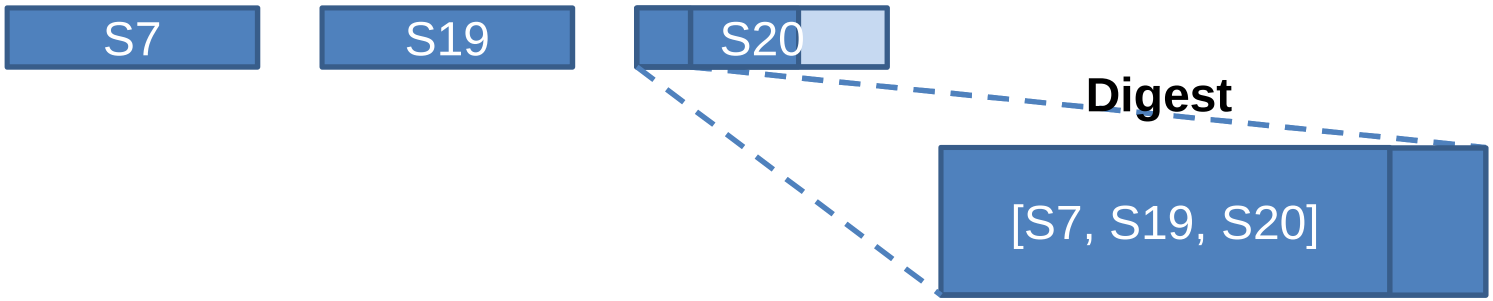
- **Detecting incomplete logs**
  - Has a complete log been found for recovery?
- **Replica consistency**
  - How are the inconsistencies due to failures handled?

Finding Replicas

- **Log segments created/destroyed rapidly across cluster**
  - 100,000s per second
- **Centralized log information too expensive**
  - No centralized list of segments that comprise a log
  - No centralized map of replica locations
- **Broadcast on Master crash**
  - Coordinator contacts each backup explicitly
  - Have to contact all backups anyway
    - Master’s segments are scattered across all of them
  - Collects a list of all available replicas

Detecting Incomplete Logs

- **Problem: Ensure complete log found during 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



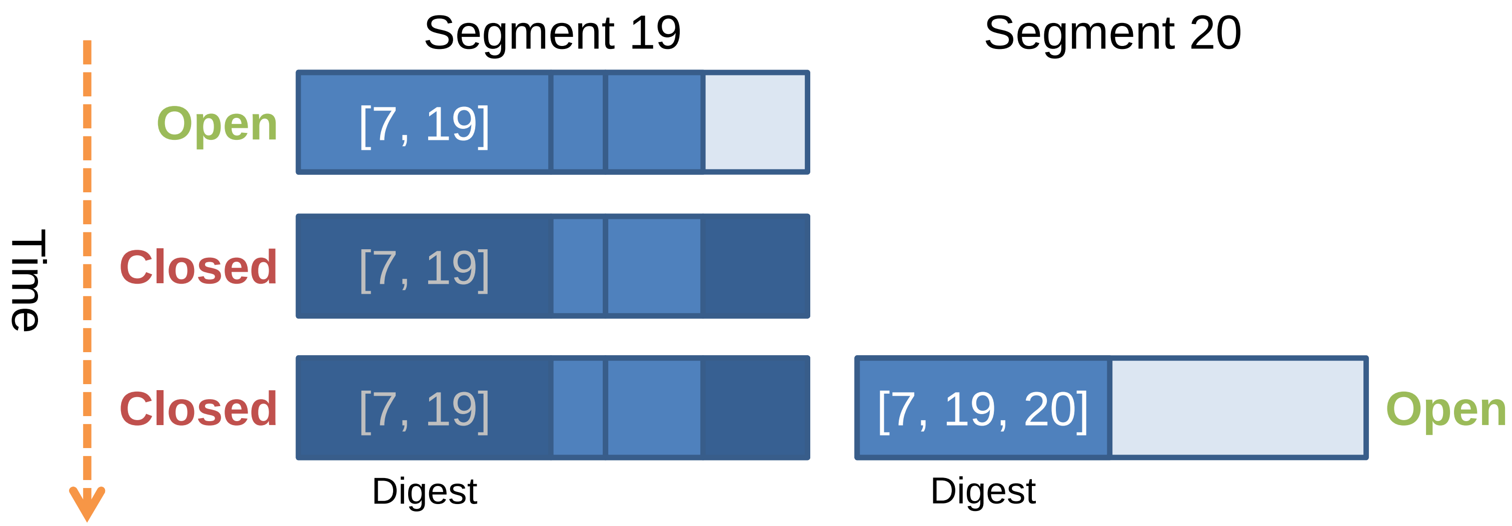
- **Reduces problem to finding the up-to-date log digest**

Choosing the Right Digest

- **Solution: Make most recent segment of log identifiable (the “head”)**
  - Segments are allocated in an “open” status
  - Mark “closed” when head segment fills up
  - Only use digests from head segment on recovery
- **Challenge: Safe transition between log heads**

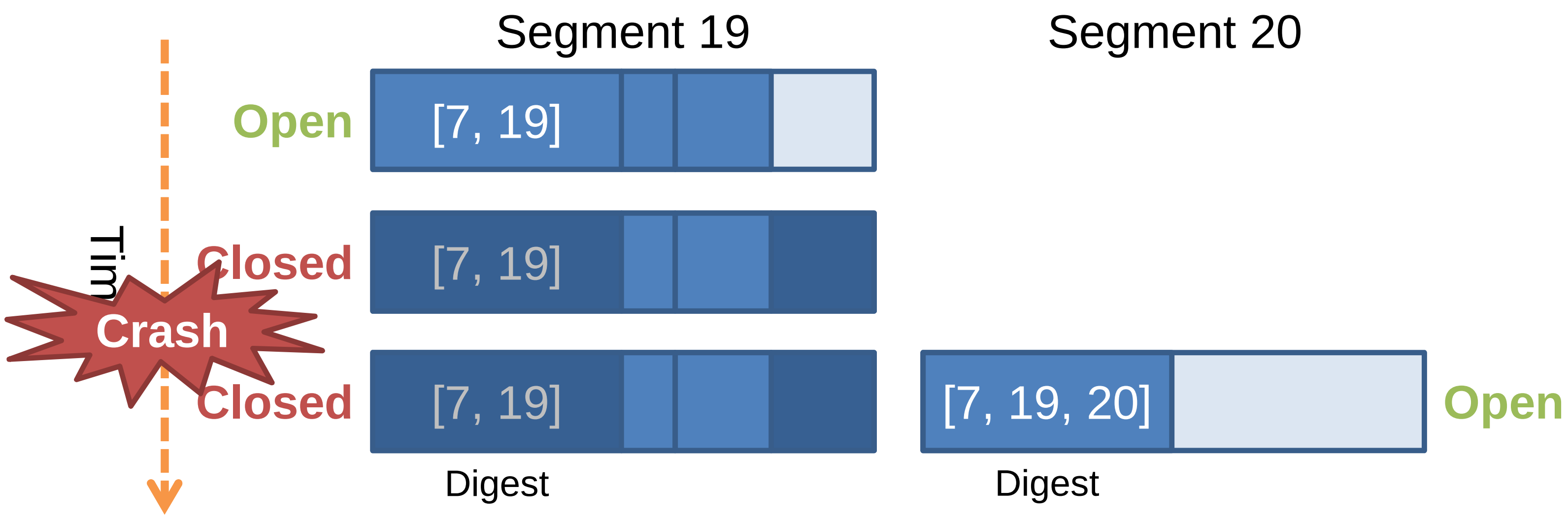
Preventing Lost Digests

- **Problem: Crash during head transition can leave no log head**



Preventing Lost Digests

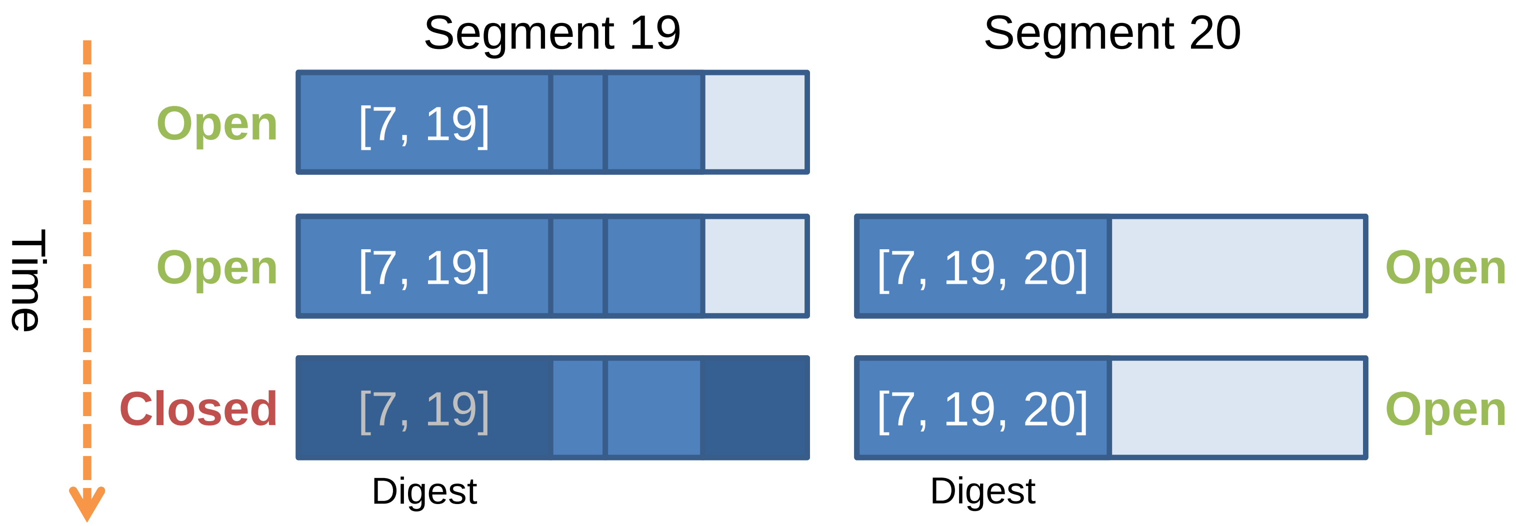
- **Problem: Crash during head transition can leave no log head**



- **Segment 19 no longer head, segment 20 hasn't been created**

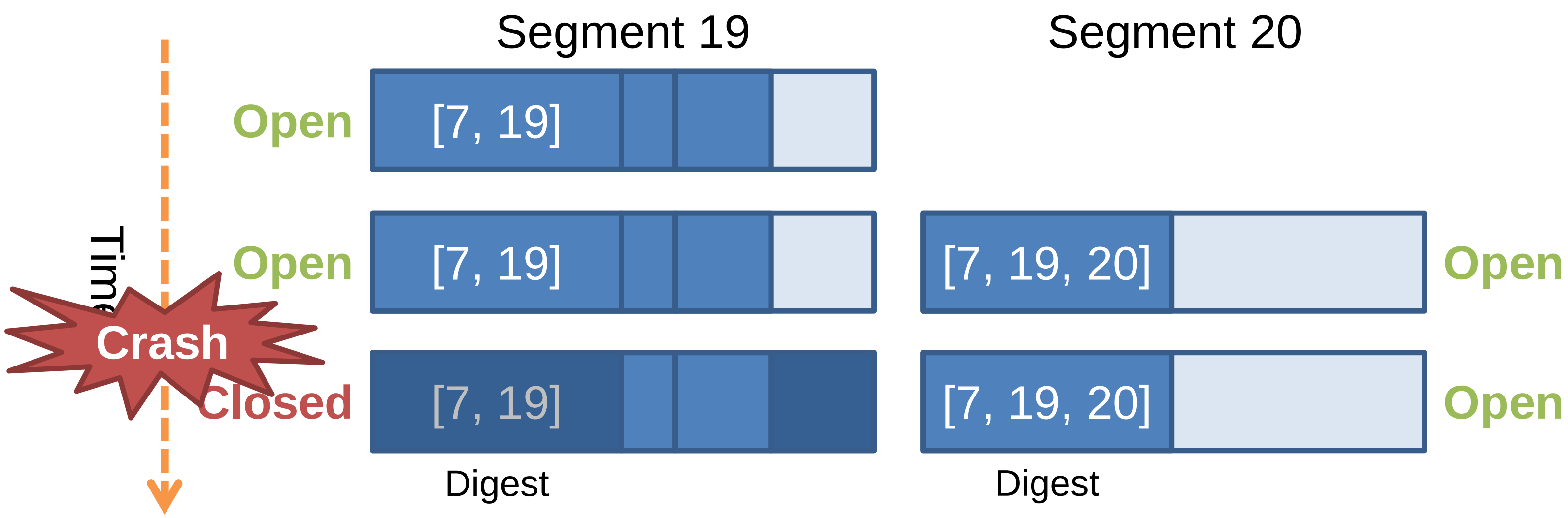
Preventing Lost Digests

- **Solution: Open new log head before closing old one**



Inconsistent Digests

- **Problem: Crash during transition can leave two log digests**

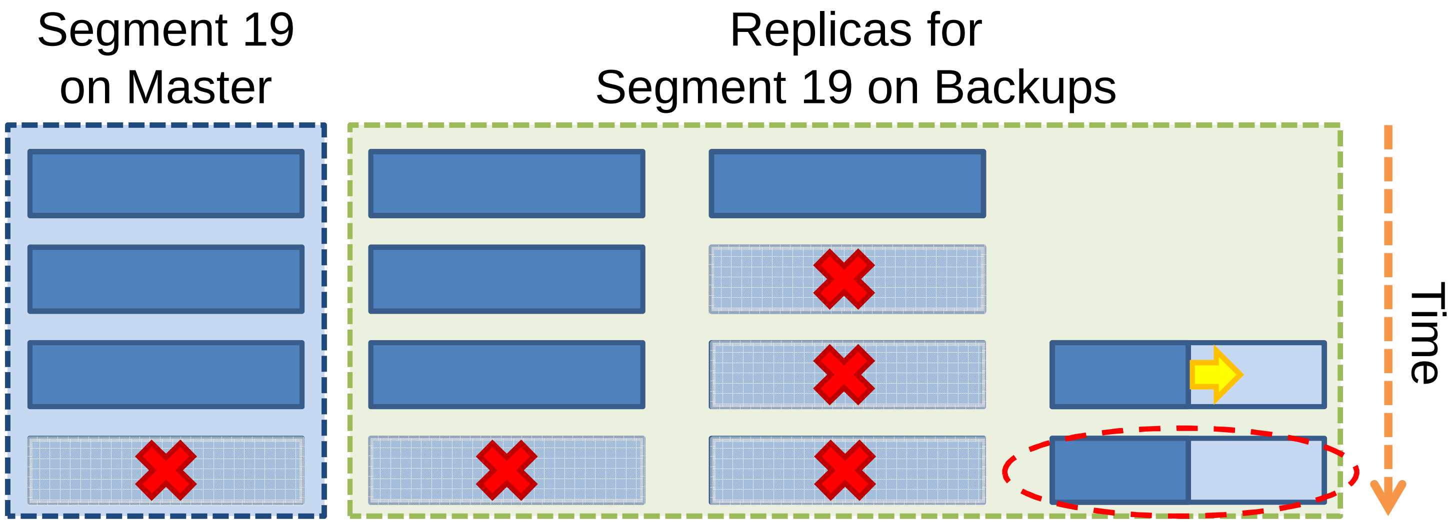


- **Solution: Don’t put data in new head until the old head is closed**
  - If new log head has no data either is safe to use

Replica Consistency

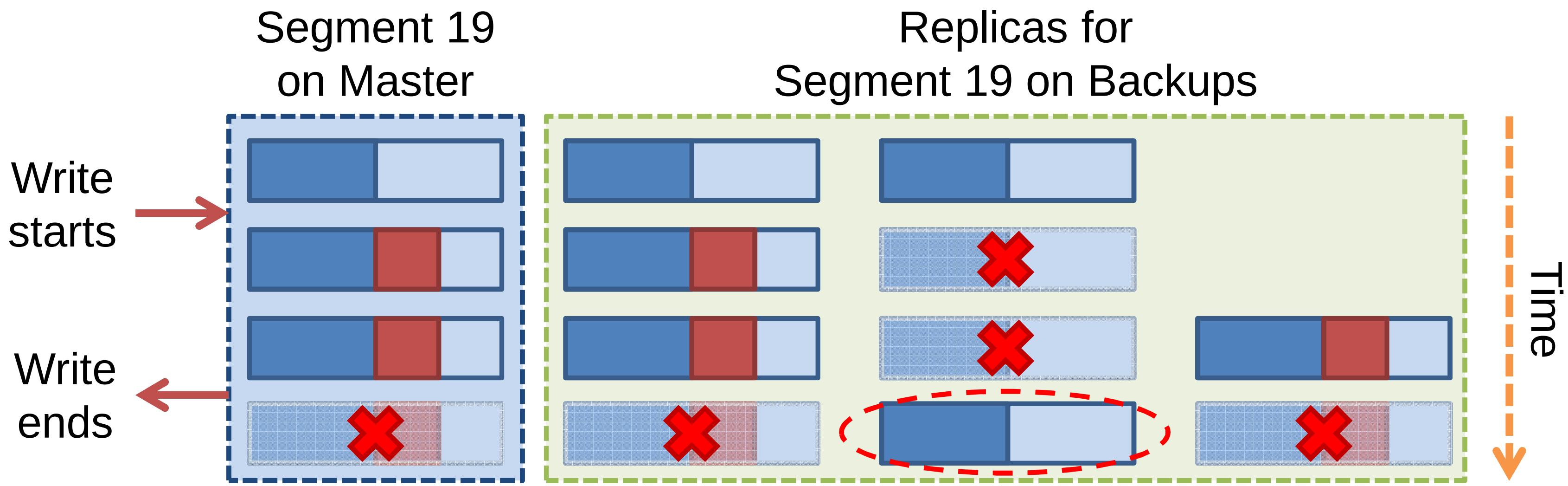
- **Replicas can become inconsistent**
  - Failures during (re-)creation of replicas
- **Must recover even with one only replica available for each segment**
- **Can’t prevent inconsistencies**
  - Make them detectable
  - Prevent recovery from using them
- **Open segments present special challenges**

Failures on Closed Segments



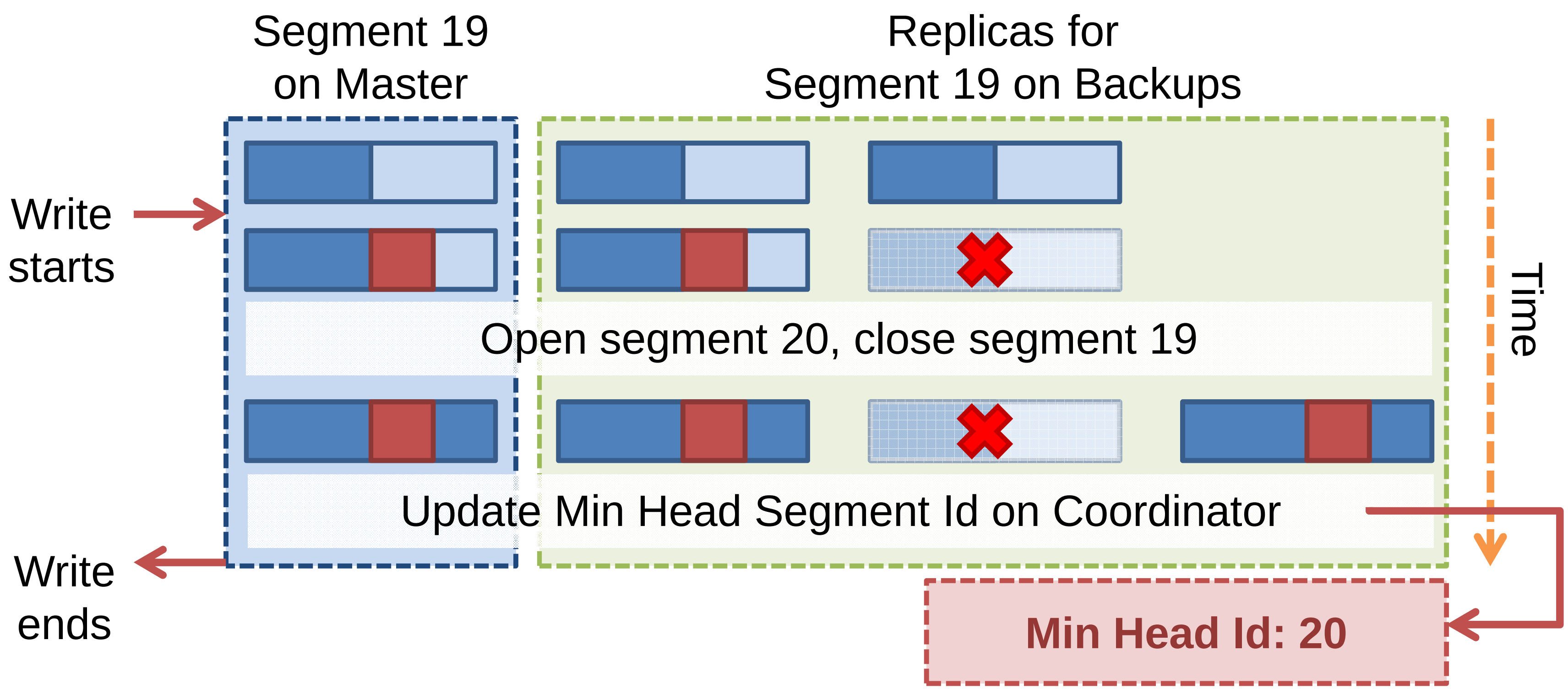
- **Problem: Master may recover from a partial replica**
- **Solution: Recreate replicas atomically**
- **Backup considers atomic replica invalid until closed**
  - Will not be persisted or used in recovery

Failures on Open Segments



- **Closed segment solution doesn't work**
- **Problem: Segment in Master's RAM is changing**
  - Lost replica may be useful for recovery one moment
  - A moment later it must not be used during recovery
- **Rare: Expect 3 Masters affected every 30 mins or longer**

Failures on Open Segments



- **Solution**
  - Make valid replicas distinguishable (close them)
  - Notify coordinator to prevent use of open replicas

Fault-tolerant Distributed Log

- **Detecting incomplete logs**
  - Broadcast to find replicas
  - Log digest provides catalog of segments
  - Head of log ensures up-to-date digest is found
- **Replica consistency**
  - Atomic replica recreation for most segments
  - Centralized minimum head id invalidates lost replicas
- **No centralization during normal operation**
  - Even during transitions between segments

What’s done

- **Normal operation** [Fall 2009]
  - Low latency **5 µs** remote access [Fall 2010]
- **Master recovery** [SOSP 2011]
  - 1 - 2 s restoration of availability after failures
- **Backup recovery** [Feb 2012]
  - Durability safely maintained via re-replication

What's left

- **Massive Failures/Cluster Restart** [Spring 2012]
  - Replica garbage collection from backups [Apr 2012]
  - Master recovery on coordinator [May-June 2012]
    - Recoveries which can wait for lost replicas
- **Experiment with longer-lived RAMCloud instances** [Summer 2012]
  - Work out kinks
  - Measure/understand complex interactions
    - e.g. Multiple recoveries, impact of recovery on normal operation, storage leaks, etc.
- **Explore strategies for surviving permanent backup failures**

Summary

- **RAM's volatility makes it difficult to use for storage**
  - Applications expect large-scale storage to be durable
  - RAMCloud's durability & availability solve this
- **RAMCloud provides general purpose storage in RAM**
  - Low latency with large scale
- **Enables new applications which access more data**
- **Simplifies development of large scale applications**