Memory and Object Management in a Distributed RAM-based Storage System

Thesis Proposal

Steve Rumble

April 23rd, 2012

RAMCloud Introduction

- **General-purpose datacenter storage system**
- **All data in DRAM at all times**
- **Pushing two boundaries:**
  - Low Latency: 5 – 10μs roundtrip (small reads)
  - Large Scale: To 10,000 servers, ~1PB total memory
- **Goal:**
  - Enable novel applications with 100 – 1,000x increase in sequential storage ops/sec
- **Problem:**
  - How to store data while getting high performance, high memory utilization, and durability?

Thesis

- **Structuring memory as a log and using parallel and two-level cleaning enables high-performance memory allocation without sacrificing utilization or durability.**

Contributions

- **Log-structured memory**
  - High performance in memory with durability on disk
- **Parallel cleaning**
  - Fast memory allocation, overheads off critical path
- **Two-level cleaning**
  - Optimizing the utilization/write-cost trade-off
- **Tombstones**
  - Delete consistency in the face of recoveries
- **Tablet Migration**
  - Rebalancing and cluster-wide data management

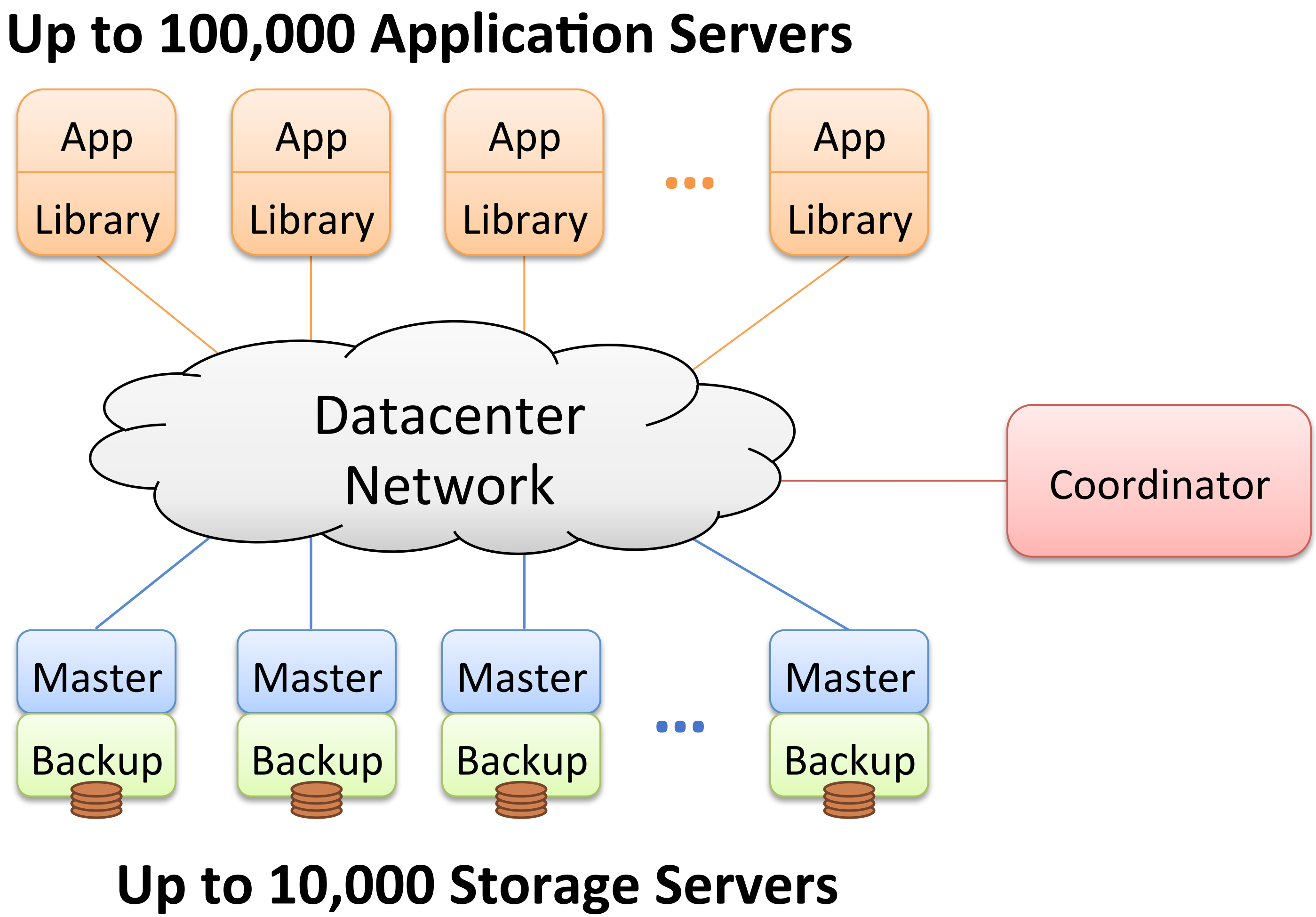
Outline

- **RAMCloud Background**
- **Contributions**
  - Log-structured memory
  - Parallel Cleaning
  - Two-level Cleaning
  - Tombstones
  - Tablet Migration
- **Conclusion**
  - Status
  - Future Work
  - Summary

Outline

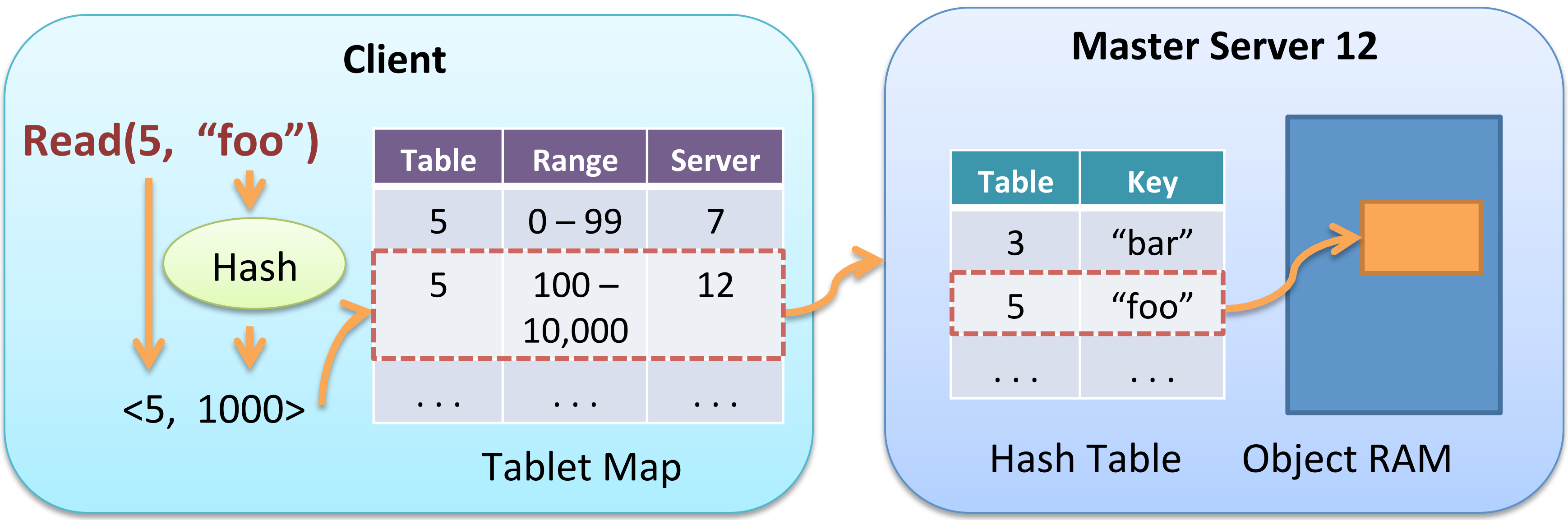
- **RAMCloud Background**
- **Contributions**
  - Log-structured memory
  - Parallel Cleaning
  - Two-level Cleaning
  - Tombstones
  - Tablet Migration
- **Conclusion**
  - Status
  - Future Work
  - Summary

RAMCloud Architecture



Distributed Key-Value Store

- **Data model: key-value**
  - Keys scoped into tables
  - Tables may span multiple servers (“tablets”)
  - Addressing: <table=5, key=“foo”>

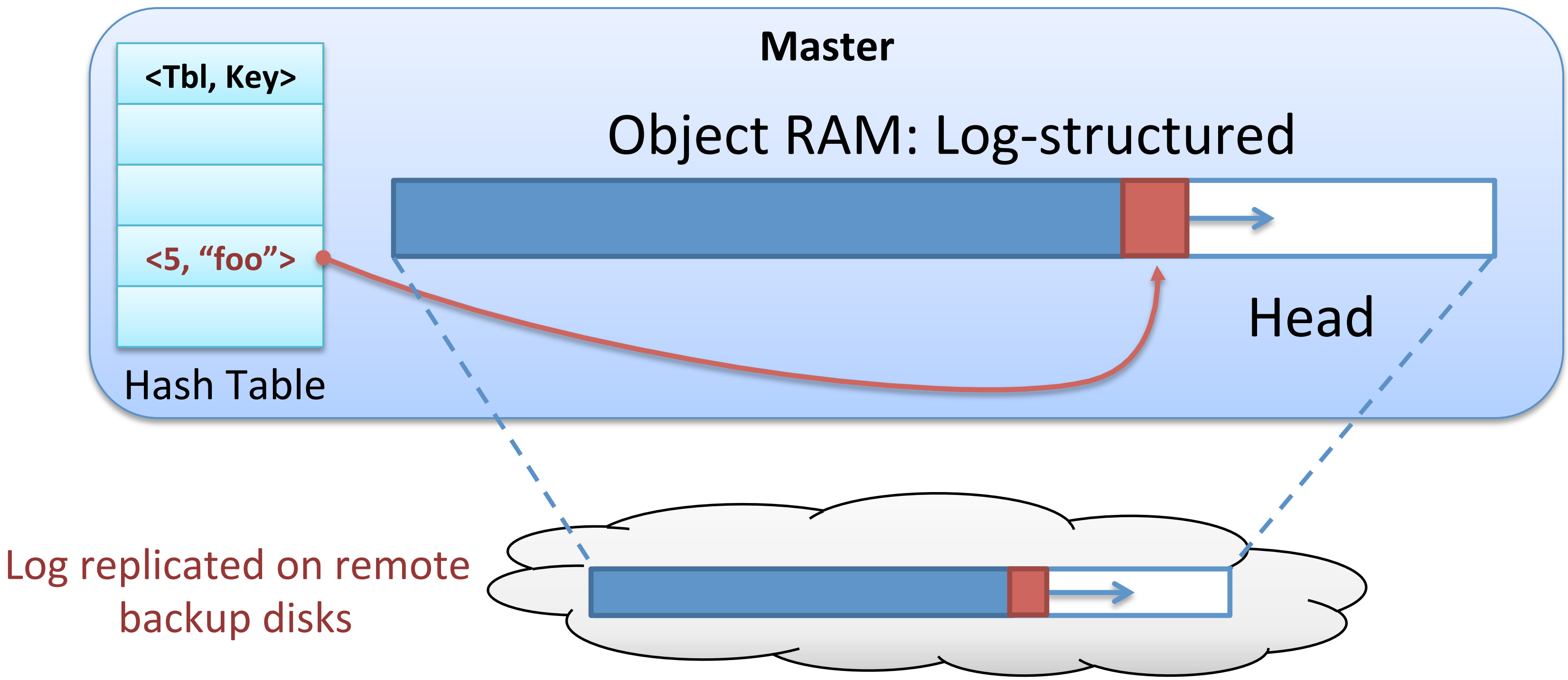


Outline

- RAMCloud Background
- **Contributions**
  - Log-structured memory
  - Parallel Cleaning
  - Two-level Cleaning
  - Tombstones
  - Tablet Migration
- Conclusion
  - Status
  - Future Work
  - Summary

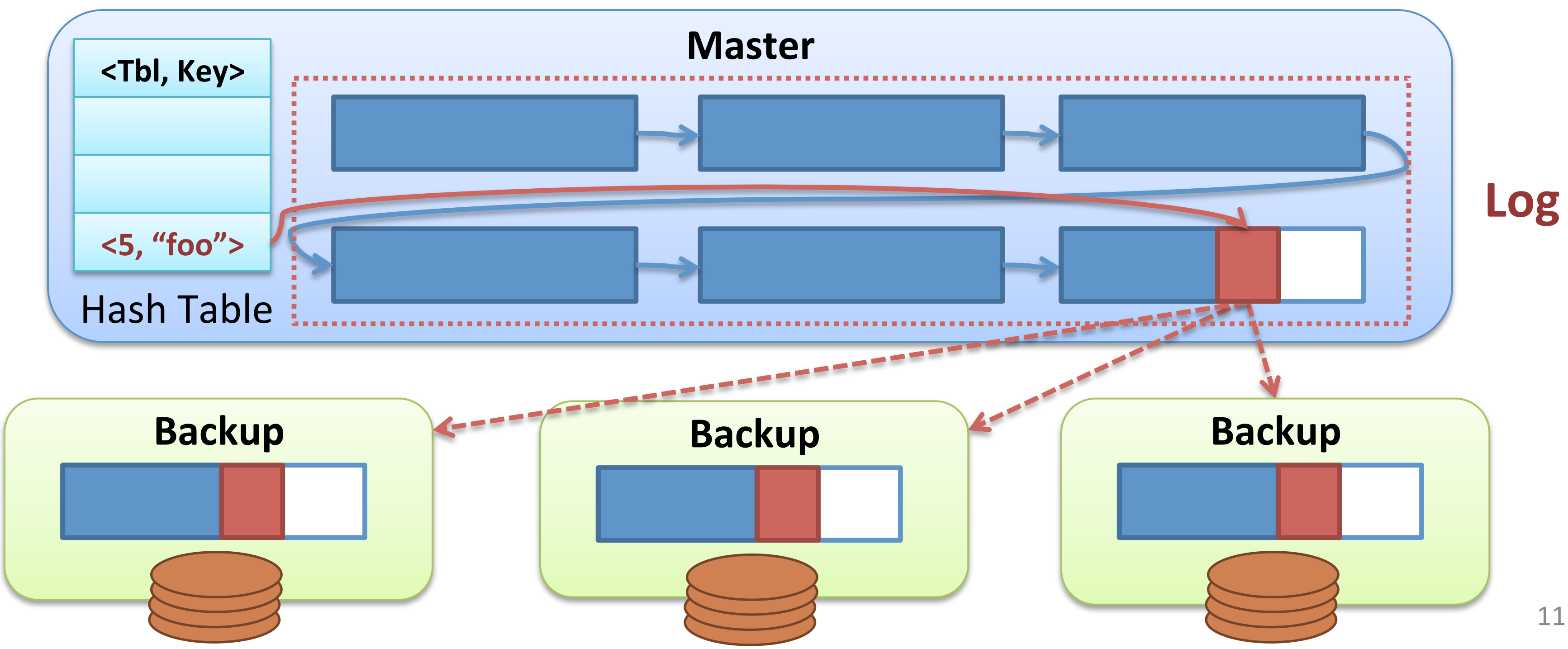
Log-structured Memory

- **Log-structure: high disk write bandwidth on backups**
  - Sequential I/O amortizes seek & rotational latency
  - Append only: Objects written to end of log (the *head*)
  - Fast allocation: Increment pointer



Benefits of Segments

- **Log divided into fixed-sized segments**
  - More efficient garbage collection (cleaning)
  - High write bandwidth (striped across backups)
  - High read bandwidth for recovery

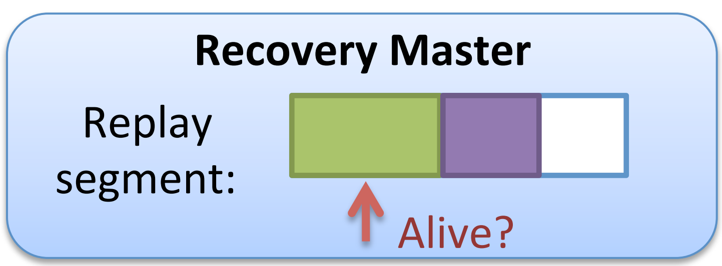


Object Deletion

- **Problems with deleting & updating objects:**
  1. Fragmentation: Reclaiming dead space for new writes

❑ Solution: *Cleaning*

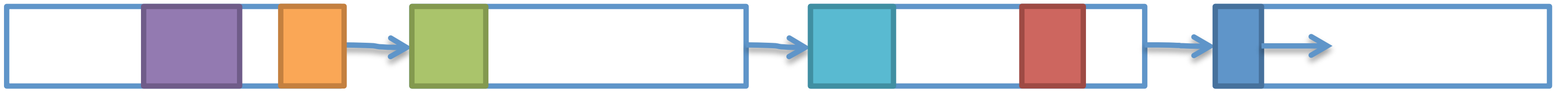
  2. Consistency: Skipping dead objects during log replay



❑ Solution: *Tombstones*

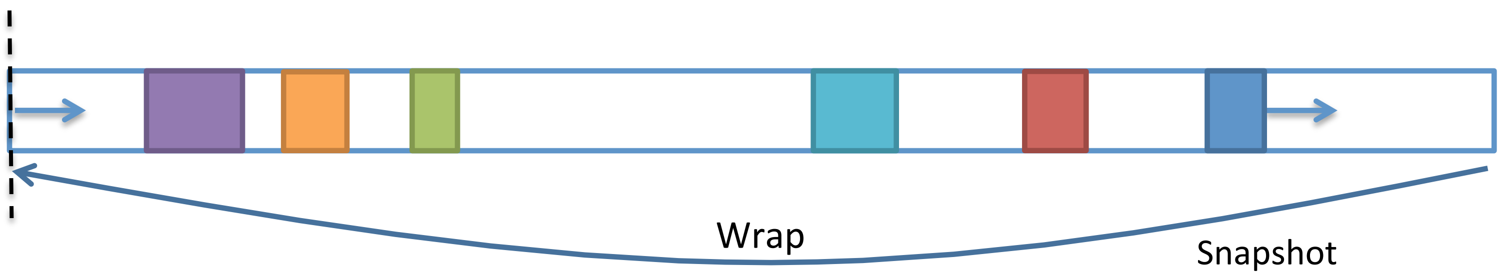
Log Cleaning

- **Problem: Deletes & updates create fragmentation**
- **Cleaning used to reclaim this space**
- **Procedure:**
  - – Select segments (LFS cost-benefit)
  - – Write live data to head of log
    - Hash table updated to point to new location
  - – Free cleaned segments



Why Cleaning?

- **Alternative: Snapshotting**
  - Mark current head position
  - Write live contents to head
  - Reclaim old space - log begins at snapshot position

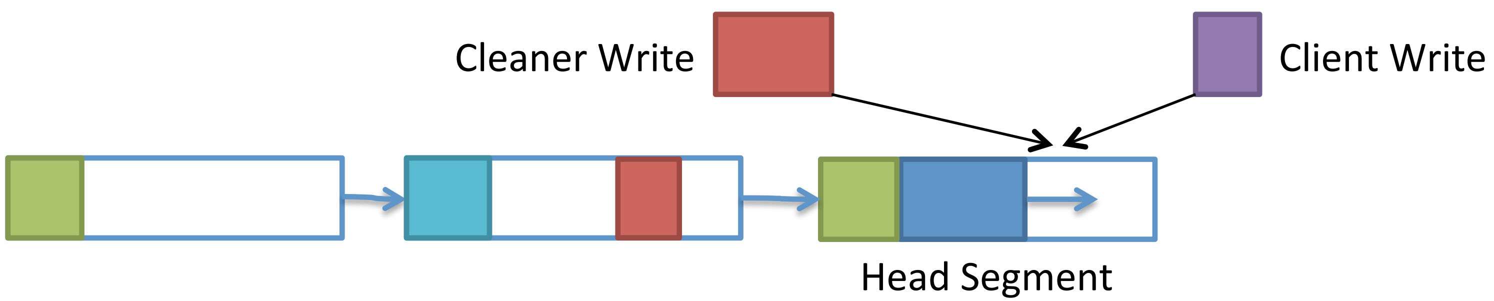


X **Problem: Expensive**
  - Always copies entire contents of log

✓ **Cleaning can skip segments with low fragmentation**

Minimizing Write Latency

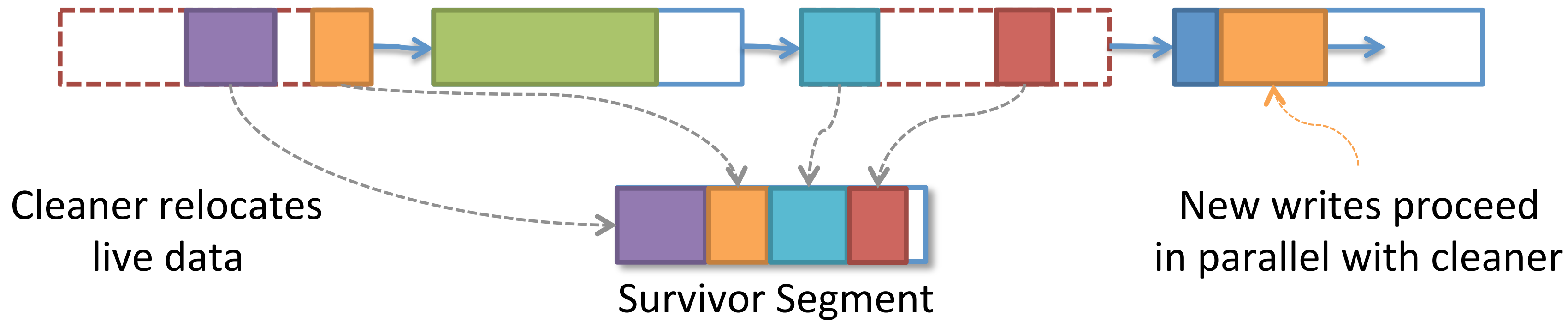
- **Problem: Cleaning contends with regular writes**
  - Recall our low latency goal
  - In steady state must constantly clean
  - But interference from cleaning threatens write latency



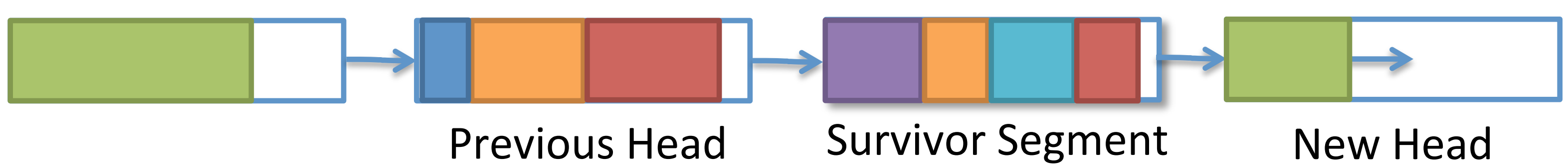
- **Solutions:**
  - Use the cores: Run cleaner in parallel
  - Minimize contention: Don't clean to head of log

Parallel Cleaning

- **Cleaner thread writes to segments outside of log**



- **Cleaned and survivor segments atomically swapped out of / into log when next head allocated**
  - Each log head enumerates all segments in the log

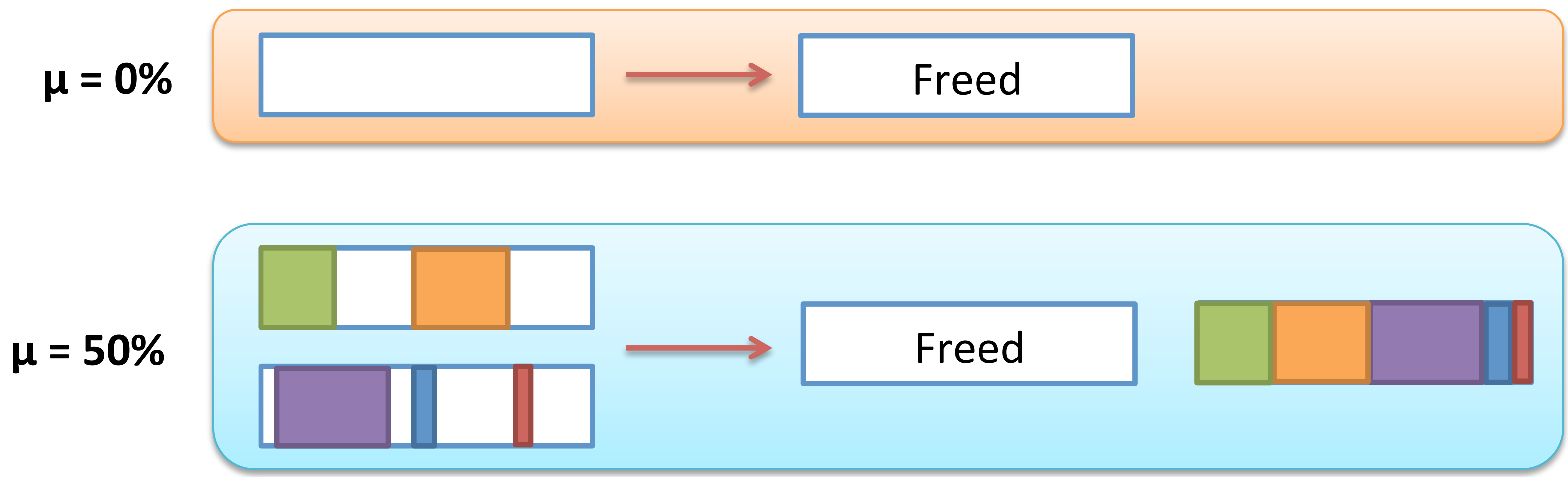


Parallelism Isn't Sufficient

- **Parallelism can hide some performance impact**
- **However, cleaning still contends for**
  - Network, disk, and memory bandwidth
  - Opportunities for contention in other parts of system
- **Questions:**
  - How expensive do we think cleaning will be?
  - If not cheap enough, how might we do better?

Cleaning Efficiency

- **Efficiency depends on utilization of selected segments**
  - The lower the utilization, the cheaper it is



- **To get one segment's worth of space back, clean:**
  - 1 segment at 0%, 2 segments at 50%, 4 at 75%, …
  - In general, clean $\frac{1}{1-\mu}$ and write $\frac{\mu}{1-\mu}$ segments

Write Cost

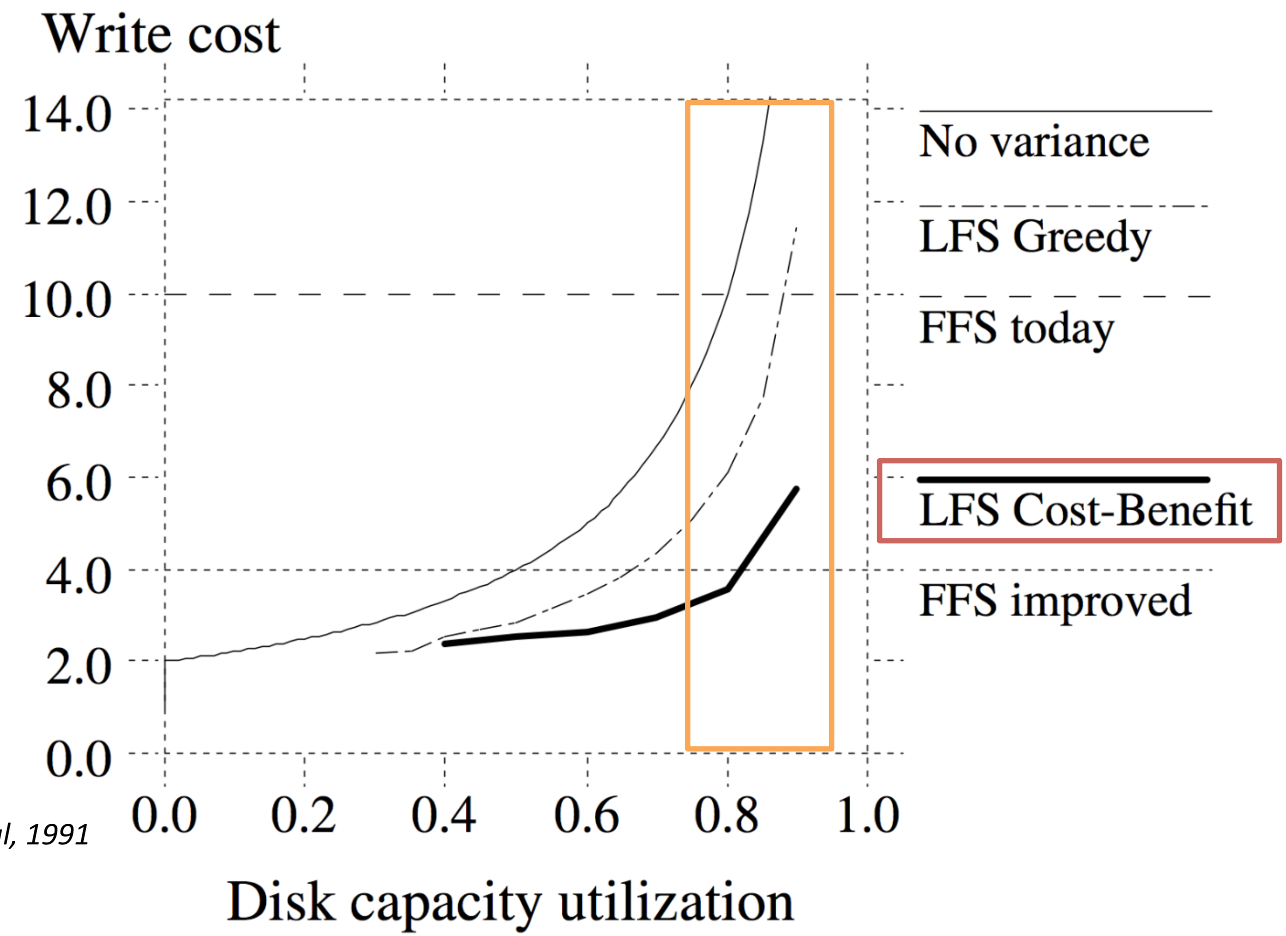
- **“Write cost”: Avg number of times each byte is copied**
  – Depends on utilization of segments cleaned

$$writeCost = \frac{1}{1 - \mu}$$

  – 1.0 is optimal
    • Cleaning always encounters empty segments
  – Same as “write amplification” in SSDs
- **LFS showed how to optimize cleaning for write cost**
  – Cost-benefit selection, hot/cold segregation

LFS Approach is Too Expensive

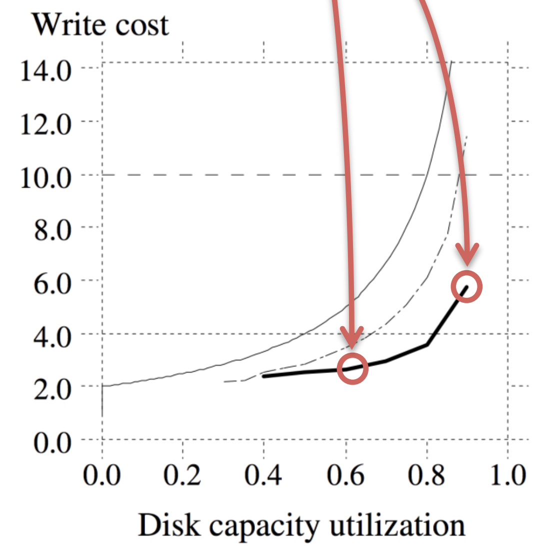
- **Conjecture:**
  DRAM expense compels running at higher utilization



Rosenblum et al, 1991

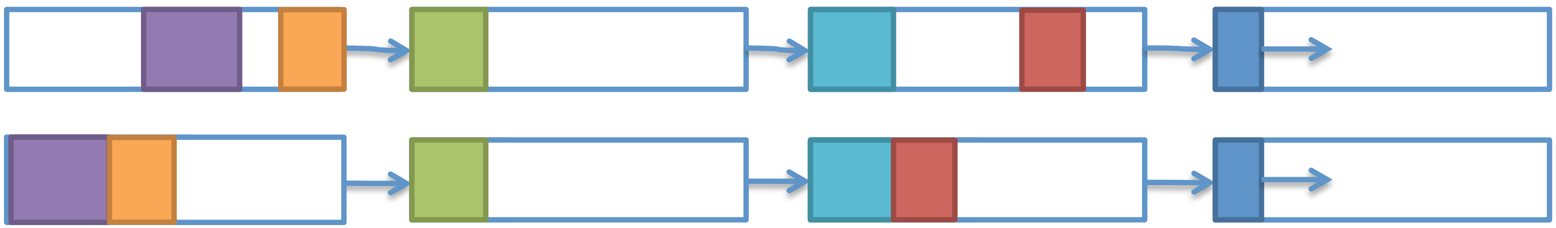
Utilization/Efficiency Dilemma

- **Problem: Disk and memory layouts coupled**
  - Cleaning in memory requires cleaning on disk
- **Forces an unpleasant choice**
  - Low memory utilization & cheap cleaning, or
  - High memory utilization & expensive cleaning
- **Can we get the best of both worlds?**
  - High utilization of precious memory
  - Low cleaning overhead
- **Idea: What if we decouple disk and memory?**



Two-level Cleaning

- **Compact segments in memory without going to disk**
  – Copy live data to front, use MMU to free and reuse tail



- **More dead objects on disk segments than in RAM**
  ⇒ Lower disk utilization
  ⇒ Lower write cost (cheaper to clean)
- **Result:**
  – Optimizes memory utilization
    - Use copious RAM bandwidth to aggressively reclaim space
  – Optimizes disk write cost
    - 2x data on disk, 50% max disk util., 2.0 max LFS write cost

Two-level Cleaning Ramifications

- **More space used on backups:**

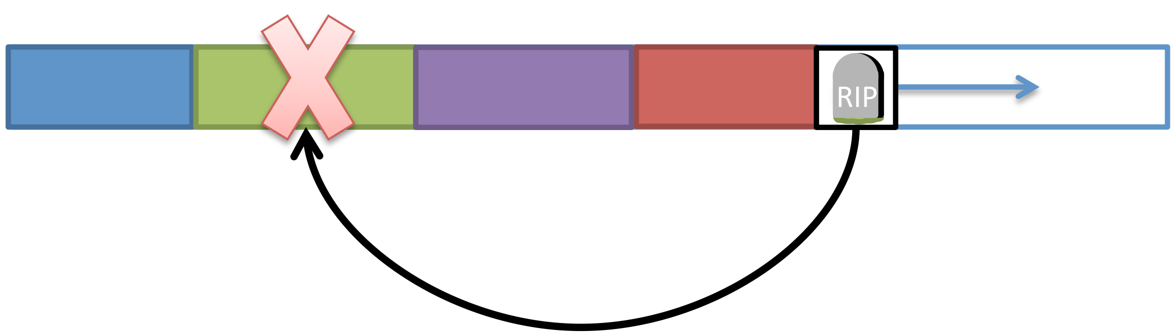
$$spaceNeeded = replicationFactor * memoryPerMaster * X,$$

(Where $X \geq 1$ is the disk expansion factor due to two-level cleaning)

  - More data to read during recovery
    - More resources needed to recover in same time
    - Or slower recovery times
  - More expense in disk hardware
    - Cost per GB in hard drives probably too low to be an issue
- **Must sometimes clean segments on disk to before cleaning segments in memory**
  - Dependent log entries: not freed until another entry is purged from log

Tombstones: Telling When an Object was Deleted

- **Problem: Must skip deleted objects during recovery**
  - Objects could otherwise resurrect after failure
- **Master's hash table dictates which objects are alive**
  - But the hash table is not made durable
  - Unlike filesystems, no persistent indexing structure
- **Solution: Tombstones**
  - Metadata appended to log whenever object is deleted or overwritten



Tombstone Issues

- **Two main issues:**
  - Use space to free space
  - Garbage collection is tricky
- **But we have not found a reasonable alternative**
  - Tombstones are a thorn in our side
  - To keep data from being replayed must either:
    - Destroy it (overwrite)
    - Have some data structure that precludes it (e.g. index)
  - Cannot afford synchronous overwrites
  - No indexing in RAMCloud

LFS Comparison

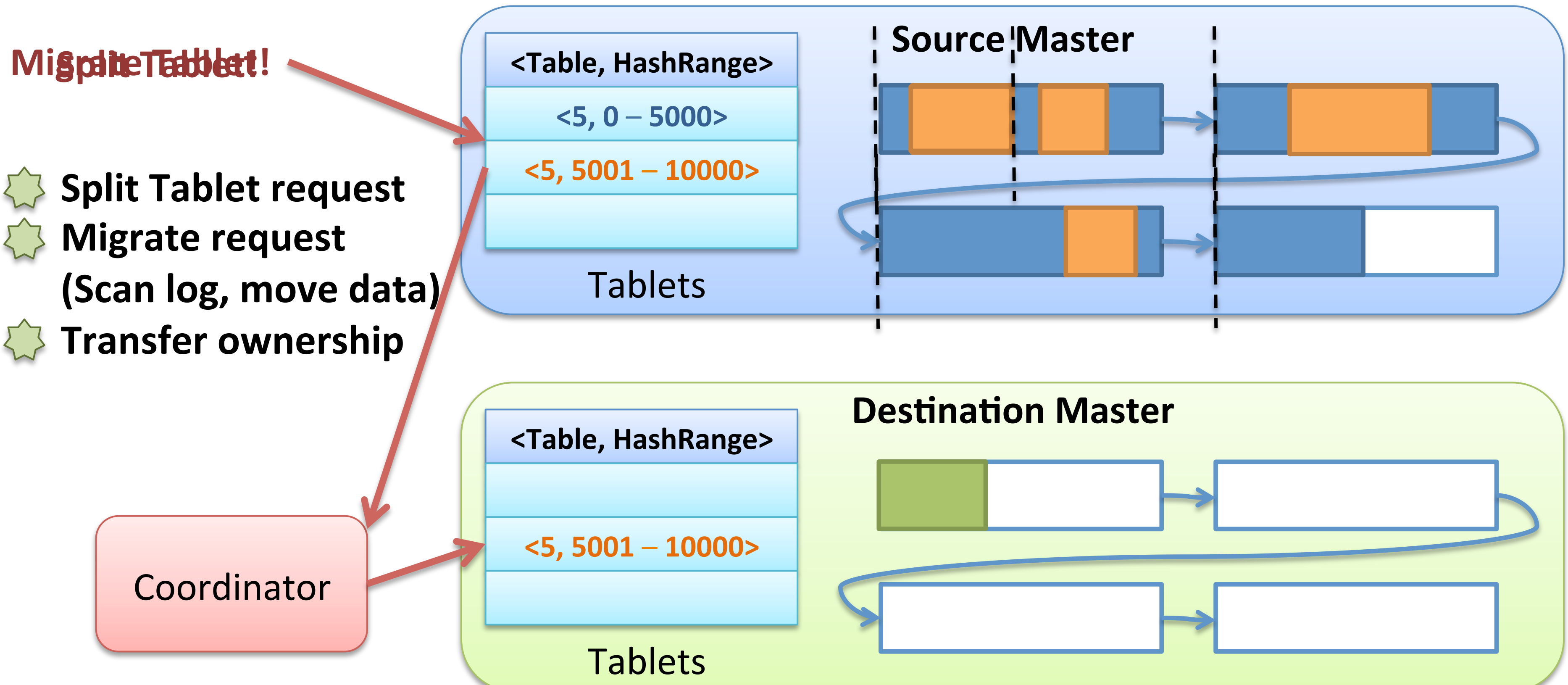
- **Similarities**
  - General structure, nomenclature (log, segments, cleaning)
  - Cost-benefit for segment selection
- **Differences**
  - Log is memory-based
    - Distributed for durability
    - Two-level cleaning
  - No disk read on cleaning (lower write cost)
  - No fixed block size
    - Reordering can create fragmentation
  - Per-object ages for cost-benefit calculation
    - Rather than per-segment
  - Filesystem vs. key-value store
    - Need to support very small objects (~100 bytes) efficiently
    - No tombstones in LFS, but more and different metadata

Cluster Memory Management

- **Managing memory across servers**
- **Need policies:**
  - When to move data between machines
    - Server memory utilization too high
    - Server request load too high (hot data)
- **Need mechanisms:**
  - How to move data between machines
    - Efficiently
    - Failure-tolerantly
    - Consistently

Mechanism: Tablet Migration

- **Tablets as basis for data movement**
- **Log-structure makes migration easy**



- **Tricky issues**
  - Migrating tablets away and back, merging adjacent tablets

Outline

- **RAMCloud Background**
- **Contributions**
  - – Log-structured memory
  - – Parallel Cleaning
  - – Two-level Cleaning
  - – Tombstones
  - – Tablet Migration
- ➢ **Conclusion**
  - Status
  - Future Work
  - Summary

Current Status

- **“First draft” log and cleaner since 2010/2011**
  - On-disk cleaning only
  - Parallel cleaning with cost-benefit selection
  - Very little performance measurement

- **Two-level prototype cleaner off of main branch**

- **Prototype tablet migration mechanism partially implemented**
  - Full tables can be migrated, no splitting/joining, no failure tolerance

Timeline for Future Work

- **Goal:** Graduation in 12 – 18 months
- **2012**
  - Integrate revised two-level cleaner
  - Measure performance, iterate on design, write up
  - Complete tablet migration mechanism
  - Explore cluster-wide data management policies
    - When to migrate, what to tablet move, where to move it to, etc.
    - Interaction with cleaning
- **2013**
  - Wrap-up, dissertation writing

Summary: Thesis Contributions

Managing memory for high performance, high utilization, and durability via:

- **Log-structured memory**
- **Parallel cleaning**
- **Two-level cleaning**
- **Tombstones**
- **Tablet Migration**