Data Durability

Steve Rumble
Stanford University

RAMCloud Design Review   April 1, 2010

Introduction

- **Want durability with single copy in RAM**
  - Use disks for replicated backup copies (cheap, non-volatile)
- **Natural to use a logging approach**
  - Exploit high sequential I/O bandwidth
  - Avoid high access latencies
- **Scatter backups across cluster**
  - Fast recovery
  - High burst write bandwidth
- ***Next talk will discuss recovery from durable storage***

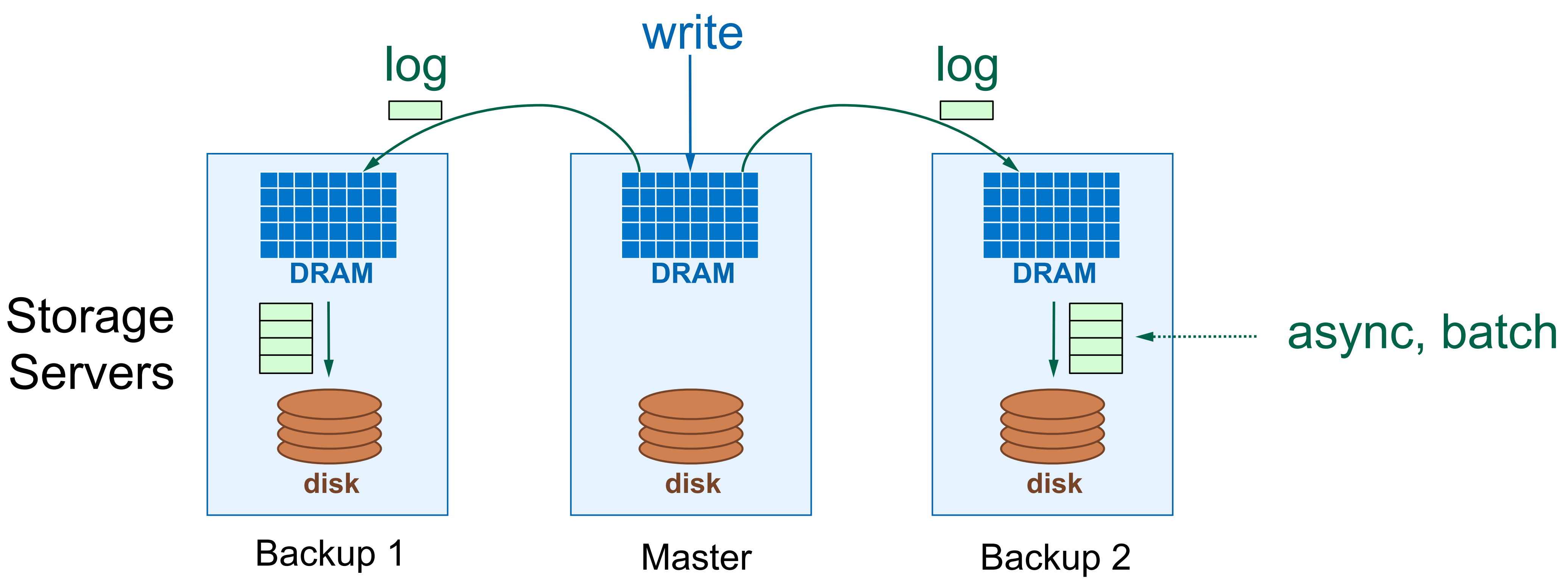
Data Durability

- **We need durability**
  - Servers will fail
  - The power will go out
  - Failures will be frequent
    - System always in recovery?

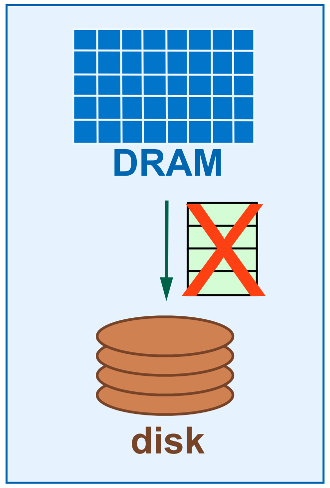
- **We need to replicate main memory contents**
  - RAM is not feasible
    - Highest performance, but:
      - Assumes we can keep all RAM powered at all times
      - Too expensive: increases cost per bit by replication factor
  - Local disks are not feasible
    - Require synchronous writes (latency much too high!)
    - Too slow to recover (single spindle)
    - What if the box dies?

Buffered Logging

- **Each server maintains a log of updates to its objects**
  - We call the owner server a "*master*"
- **Masters' log updates are sent to *R* backups**
  - RPCs return when backups have updates buffered in RAM
  - Backups batch and write to disk asynchronously
  - Assume for now each master always uses same *R* backups
- **Each master is also a backup for other servers**

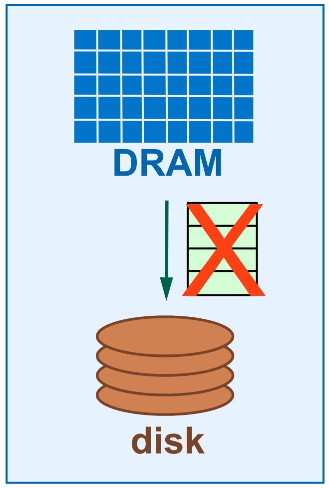


Backup Buffer Volatility



- **Problem: Backup buffers are in volatile DRAM**
  - Vulnerable until disk flush
- **Solution 1: Synchronous disk writes**
  - 2,000 - 8,000 microsecond latency!
  - No write batching => very low bandwidth (< 1% of sequential I/O)
- **Solution 2: Synchronous flash SSD writes**
  - ~50 - 100 microsecond latency, still a big non-sequential I/O penalty
- **Solution 3: Reduce consistency guarantees**
  - “Sorry, we lost your data. Deal with it.”
- **Solution 4: Battery Backups**
  - Batteries provide enough power to flush buffers

Backup Buffer Volatility



- **Problem: Backup buffers are in volatile DRAM**
  - Vulnerable until disk flush
- **Solution 1: Synchronous disk writes**
  - 2,000 - 8,000 microsecond latency!
  - No write batching => very low bandwidth (< 1% of sequential I/O)
- **Solution 2: Synchronous flash SSD writes**
  - ~50 - 100 microsecond latency, still a big non-sequential I/O penalty
- **Solution 3: Reduce consistency guarantees**
  - "Sorry, we lost your data. Deal with it."
- **Solution 4: Battery Backups**
  - Batteries provide enough power to flush buffers

Log-Structured Backups

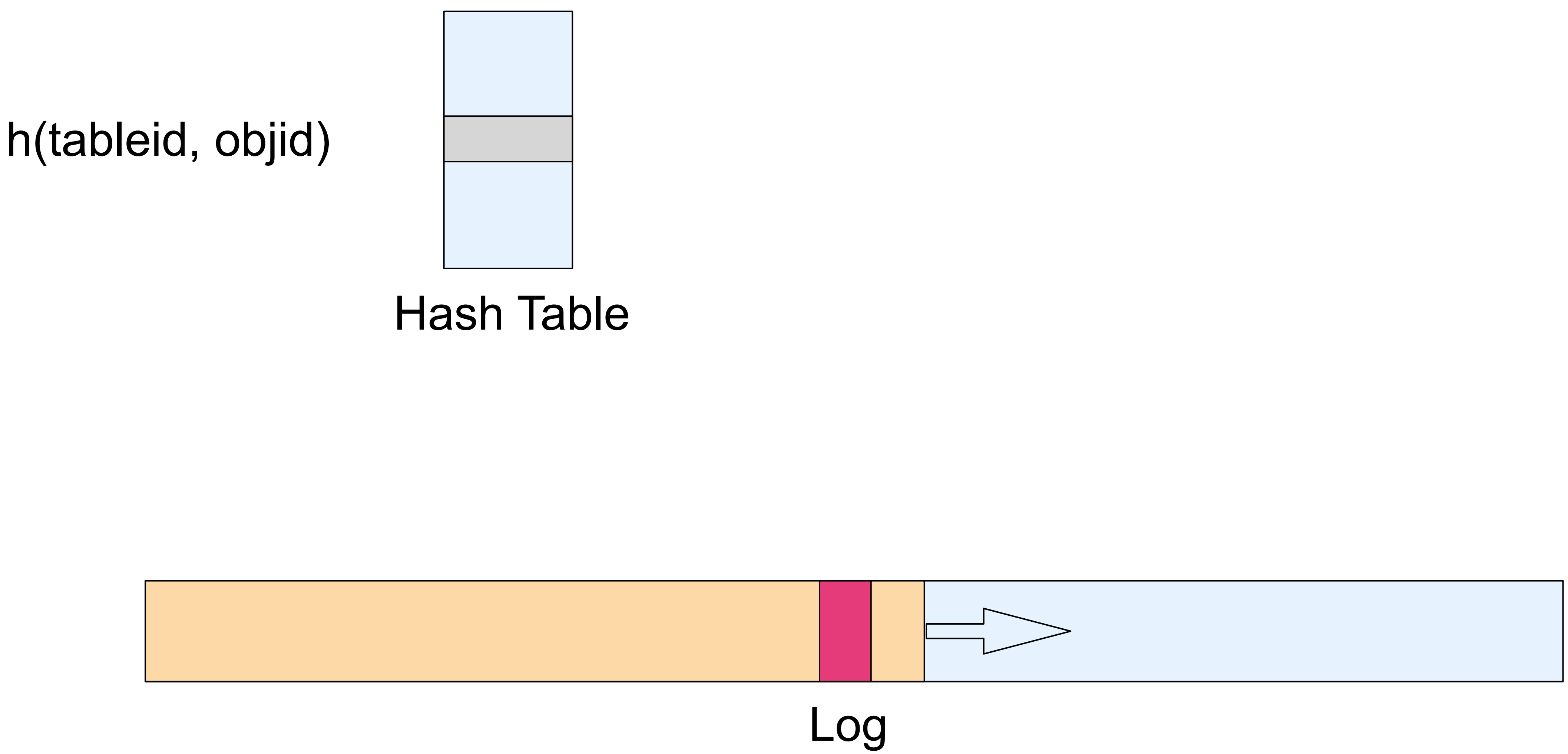
- **Problem: Need fast write rates, but have disks**
  - RAMCloud is about performance, after all
- **Solution: Log-structure on disks**
  - Exploits sequential I/O
  - But we need to do cleaning
- **What about log cleaning overheads?**
  - All data is in RAM, so no need to re-read for cleaning
    - 50% of the overhead immediately out the window
  - Don't need to use disks efficiently
    - Worry about RAM utilisation
    - Assume backups have capacity to spare.

Log-Structured Memory

- **Problem: Server must keep track of the Log**
  - I.e. we need to do cleaning
- **Solution: Make server memory log-structured**
  - Memory layout matches disk layout
  - Simplicity Benefit: Unify disk-based storage and memory allocation
    - Clean memory and disk simultaneously
- **Hey, wait! This couples disk and memory utilisation!**
  - Means more aggressive cleaning to avoid wasted memory
  - No reason this cannot be decoupled in the future (we expect to)
  - Primarily an initial design simplification

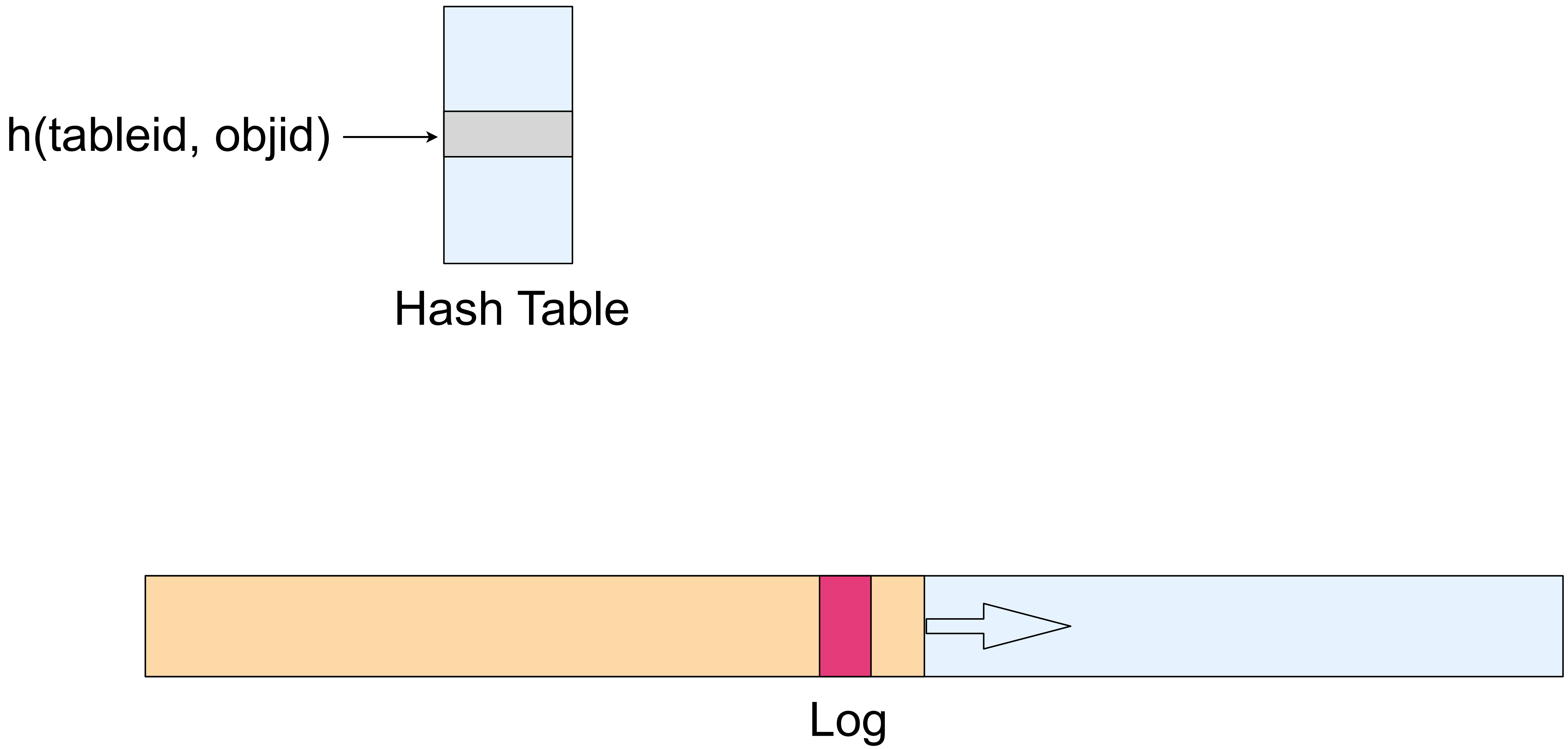
Locating Objects in the Log

- **How do we find objects in the main-memory Log?**
  - Hash table lookup
  - Two cache misses from (tableId, objectId) to object
    - Extremely fast, despite complex memory management scheme



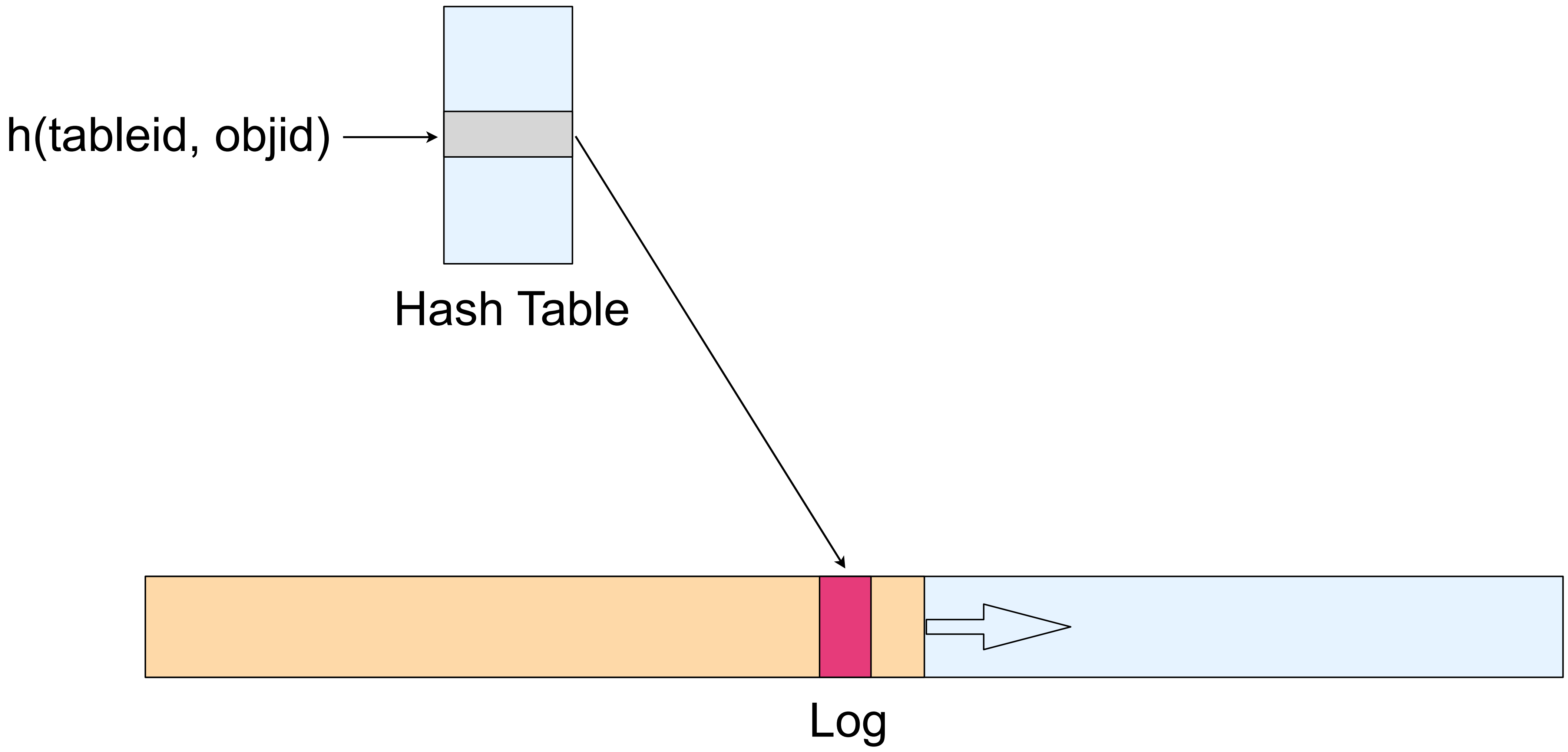
Locating Objects in the Log

- **How do we find objects in the main-memory Log?**
  - Hash table lookup
  - Two cache misses from (tableId, objectId) to object
    - Extremely fast, despite complex memory management scheme



Locating Objects in the Log

- **How do we find objects in the main-memory Log?**
  - Hash table lookup
  - Two cache misses from (tableId, objectId) to object
    - Extremely fast, despite complex memory management scheme



Scattering Writes

- **Problem: Need fast recovery (1-2 seconds)**
  - If R = 2, at least 5 minutes to pull 64GB from disk!
    - R * 100MB/s I/O bandwidth insufficient for quick recovery
  - Want no noticeable availability lapses on failure
  - Writing log updates to same R backups not good enough
- **Solution: Scatter log across many spindles**
  - Don't fix the R backup hosts for each master
  - Fill buffers on R backups, then move on
  - < 1 second to read 64GB from 1,000 disks (0.65s @ 100MB/s/disk)
- **For each buffer filled, choose a new set of *R***
  - Find additional backups with idle bandwidth
  - Can accommodate more writes immediately
  - Example mechanism:
    - Cache potential backup lists (obtained from cluster coordinator)
    - Choose 2R of them as potentials and query to find the best R

Expected Sustained Write Rate

- **So, what write rate can we sustain?**
  - Only 2.5% of the read rate!
    - About 25,000 1KB objects/server/second
    - Why?

Raw Disk Bandwidth	**100 MB/s**
1KB Objects	**100,000 objs/sec**
Replica Overhead (*R* = 2)	**50,000 objs/sec**
100% Cleaning Overhead	**25,000 objs/sec**

- **10GigE: 1M 1KB objects/second => 2.5% of read rate**
  - 1,000,000/25,000, or 40:1 read/write ratio!

Boosting Write Rates

- **But 2.5% is conservative**

- **Improvements:**

Compression	**1.5 - 3x**
Additional Disks	**2 - 4x**
Total	**3 - 12x**
Write Percentage	**7.5% - 30%**

- **Need modest capacity devices with high bandwidth**
  - Prefer cheap bandwidth over cheap capacity
  - Latency less important
  - What about flash? We'll get to that later...

High Burst Bandwidth

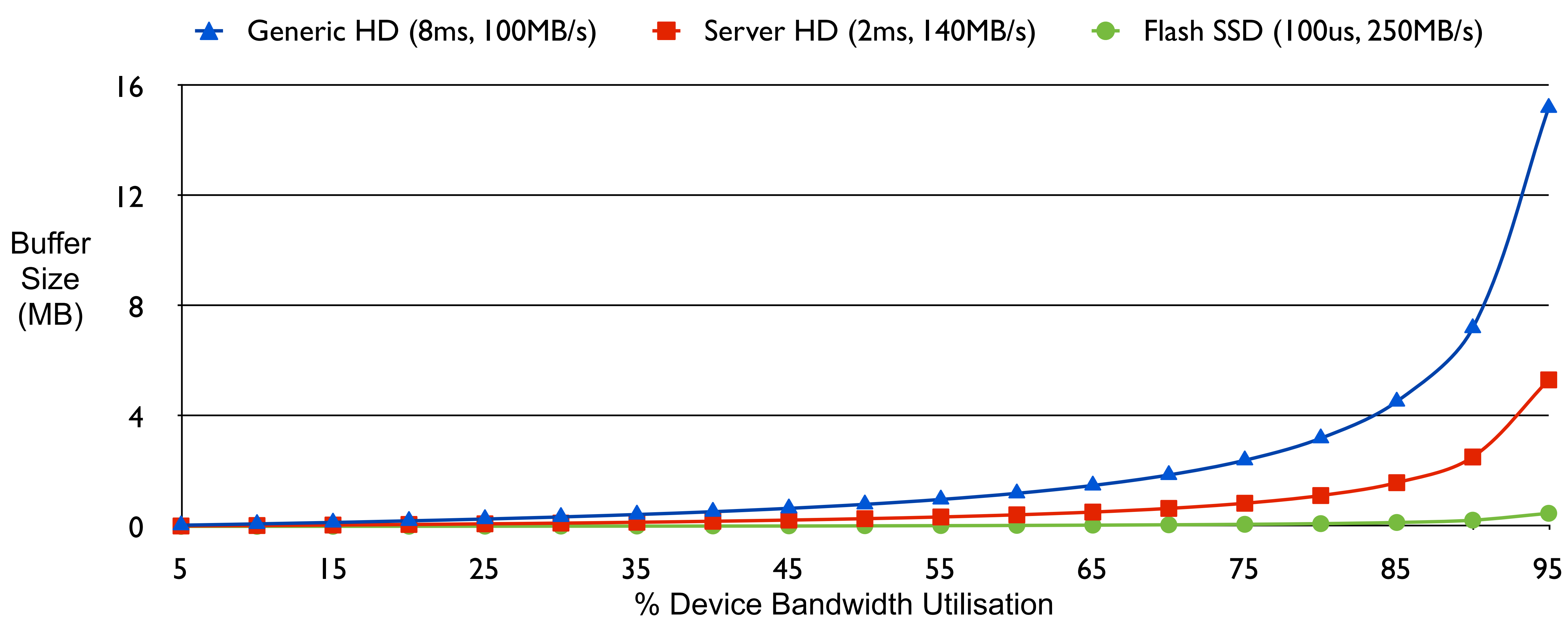
- **7.5-30% is a worst-case number**
  - Only if many masters are saturated with writes
- **Scattering writes permits large write bursts**
  - Servers make use of idle disk bandwidth throughout cluster
    - Full bisection bandwidth assumption
  - Statistical multiplexing of cluster aggregate I/O
- **Network interface becomes the bottleneck**
  - 4-15 low write-load backup servers for each write-saturated master
    - 10GigE permits about 1,000,000 1KB objects/second
      - About 10 * *R* disks' worth of sequential I/O bandwidth (100MB/s/disk)
      - 20-30 servers for reasonable values of *R*
      - Divide by 2-6x after compression and additional disks/server

How Big are the Buffers?

- **Amortising overhead means sufficient buffering**
  - But how big is "sufficient"? 7.2MB
  - Example: Want 90% utilisation
    - HDD Parameters:
      - 100MB/s sequential I/O
      - 8ms access time (seek + rotational latency)
    - 90% bandwidth => 72ms of data transfer for every op (8ms overhead)
      - 72ms at 100MB/s => 7.2MB transferred per op
    - Generalised:
      - $\text{Buffering\_Needed} = \text{Latency} \times \dfrac{\text{Utilisation}}{1 - \text{Utilisation}} \times \text{Disk\_Bandwidth}$

How Big are the Buffers? (cont'd)

- **It doesn't take much...**
  - If 8MB/buffer and *R* = 5:
    - 2*R* * 8MB = 80MB total per server (0.12% of 64GB!)
  - And it only gets better! 90% utilisation means:
    - 2.5MB buffers for server disks (2ms, 140MB/s)
    - 200KB buffers for flash SSDs (100usec, 250MB/s)



What About Flash?

- **Latency too high for primary store, but for backup?**
- **Flash is currently modest-sized and high bandwidth**
  - ~50% read/write ratio achievable with SSDs.
    - 2-3x HDD bandwidth (Recall 7.5-30% ratio for HDDs)
    - Without multicast to backups, 50% is the best we can hope for if R = 2
- **However**
  - SSDs are still very expensive
  - Insufficient write/erase cycles - 10 month wear out!
    - Common figures: 100k cycles for SLC, 10k for MLC
    - 10 months to reach 100k cycles at peak I/O (64GB flash at 250MB/s)
  - Performance expectations are complex (FTL)
- **Our techniques should work well with flash**
  - Locality is still important, so buffered approach fits
  - And may obviate complex FTLs

Conclusion & Discussion

- **Conclusion**
  - Durability with one copy in RAM
  - Logging approach for disk I/O utilisation
  - Backups scattered across cluster for recovery and bursty load

- **Possible Discussion Topics**
  - What read/write ratios should we expect?
  - How reasonable is the per-server battery backup assumption?