# Log-Structured Memory for DRAM-Based Storage

**Stephen Rumble and John Ousterhout**

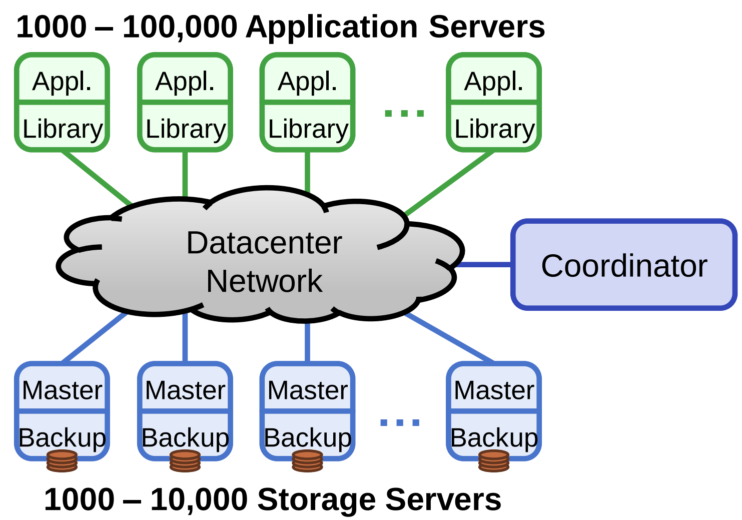
**Stanford University**

# Introduction

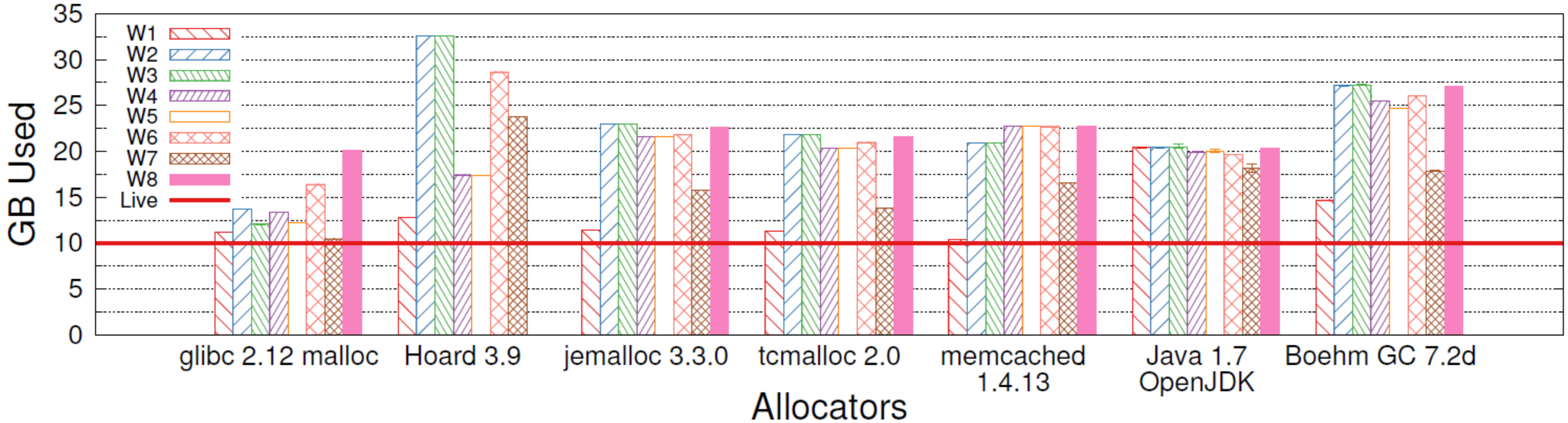
- **Storage allocation: the impossible dream**
  - Fast allocation/deallocation
  - Efficient use of memory
  - Handle changing workloads
- **RAMCloud: log-structured allocator**
  - Incremental garbage collector for both disk and DRAM
  - Two-level approach to cleaning (separate policies for disk and DRAM)
  - Concurrent cleaning
- **Results:**
  - High performance even at 80-90% memory utilization
  - No pauses
  - Handles changing workloads

# RAMCloud Overview

- **Datacenter storage system**
- **Key-value store**
- **All data in DRAM at all times**
  - Disk/flash for backup only
- **Large scale:**
  - 1000-10000 servers
  - 100TB - 10PB capacity
- **Low latency:**
  - 5µs remote access

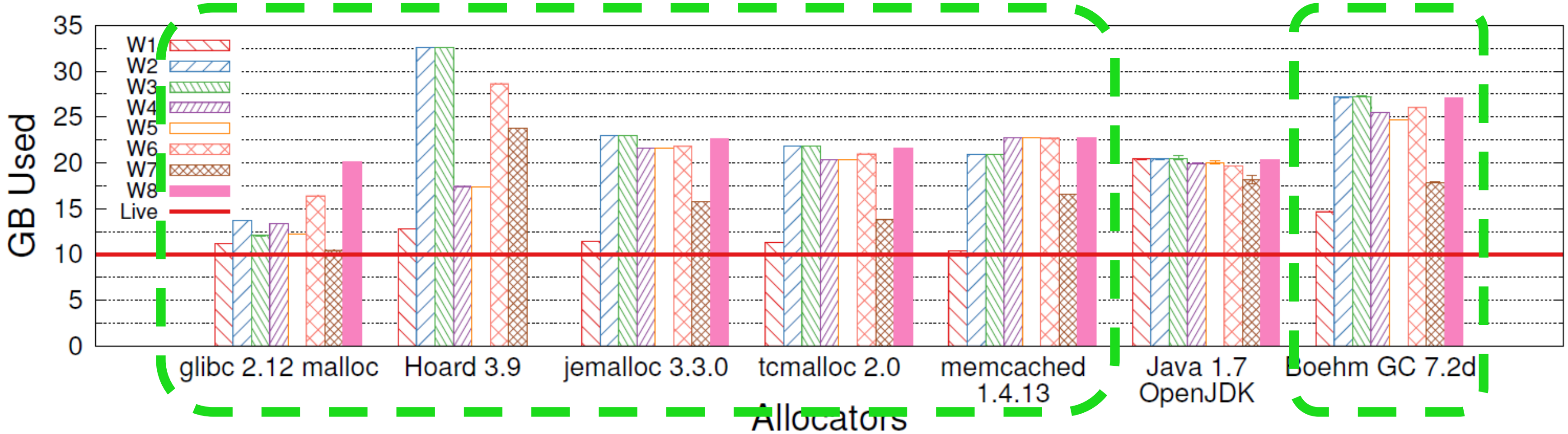


# Workload Sensitivities




- **Allocators waste memory if workloads change:**
  - E.g., W2 (simulates schema change):
    - Allocate 100B objects
    - Gradually overwrite with 130B objects
- **All existing allocators waste at least 50% of memory under some conditions**

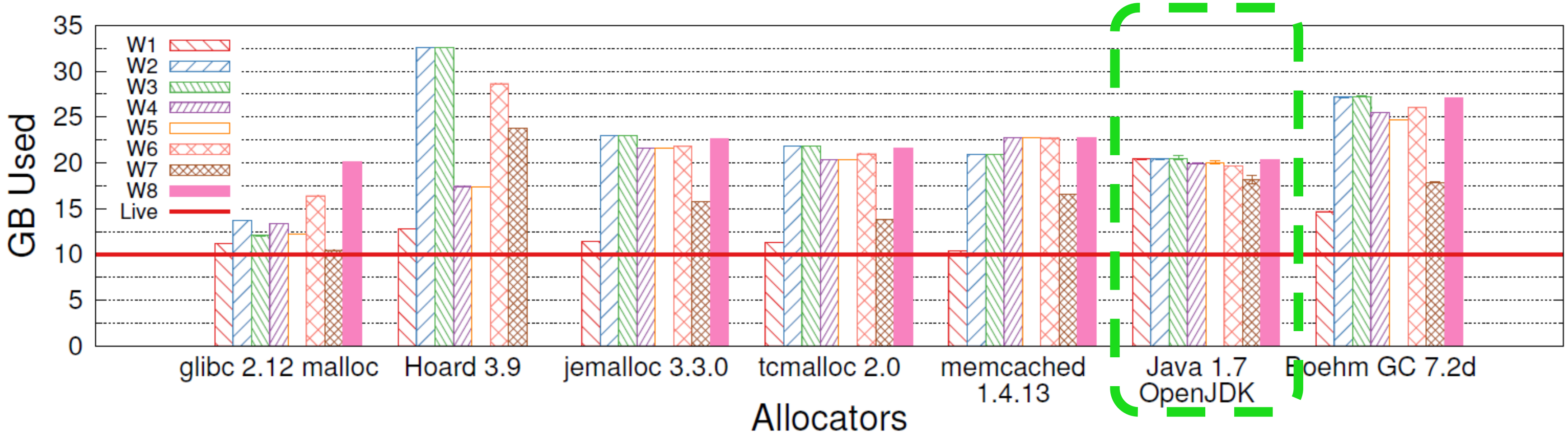
# Non-Copying Allocators



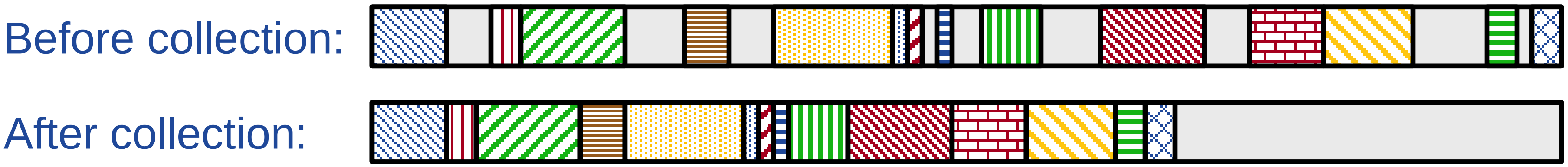

- **Blocks cannot be moved once allocated**
- **Result: fragmentation**

Current memory layout:

# Copying Allocators




- **Garbage collector moves objects, coalesces free space**

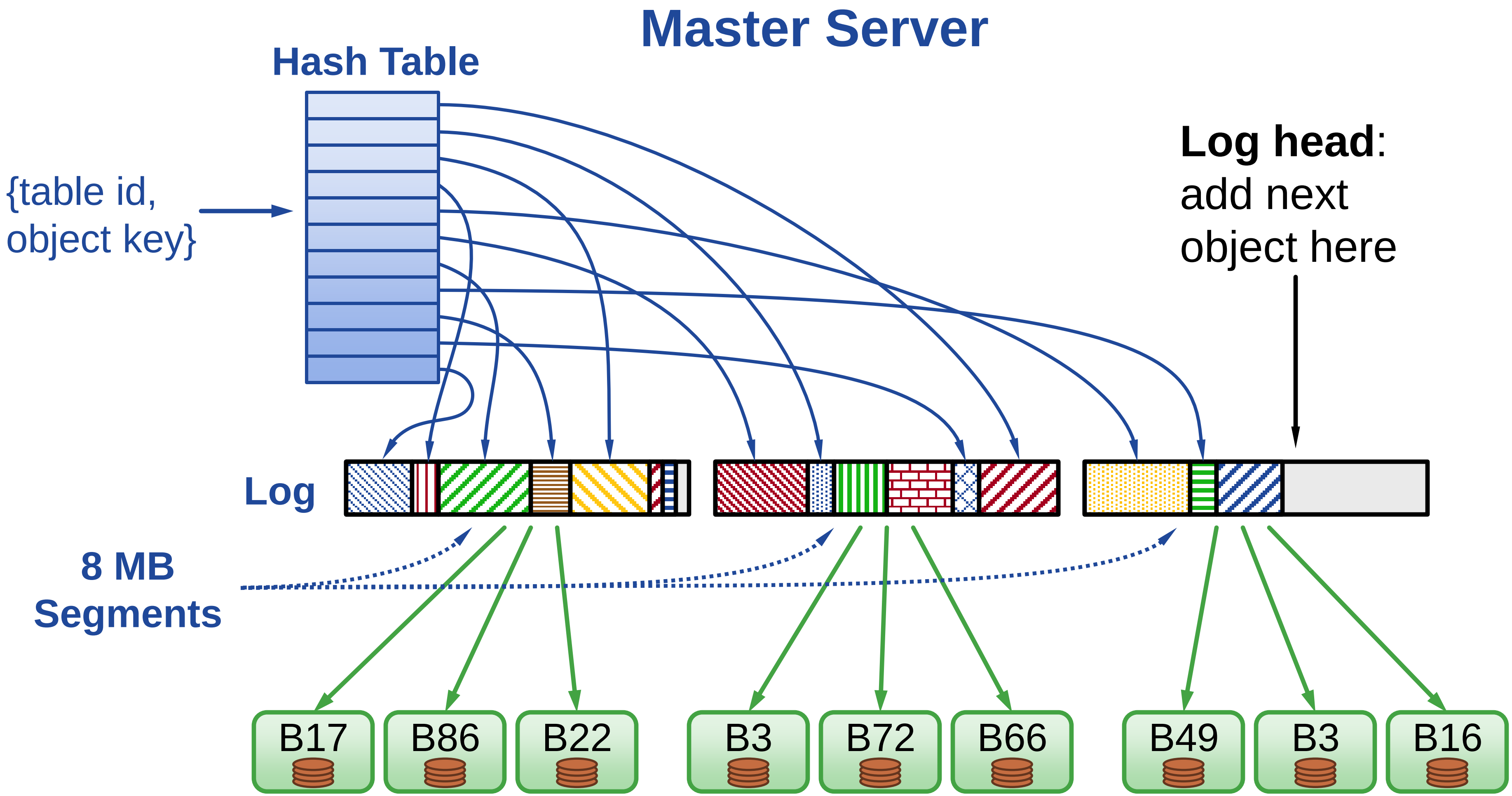



- **Expensive, scales poorly:**
  - Must scan all memory to find and update pointers
  - Only collect when there is lots of free space
- **State of the art: 3-5x overallocation of memory**
- **Long pauses: 3+ seconds for full GC**

# Storage System Goals

- **Use memory efficiently even with workload changes (80-90% utilization)**
- **Must use a copying approach**
- **Must be able to collect free space incrementally:**
  - Pick areas with most free space
  - Avoid long pauses
- **Key advantage: restricted use of pointers**
  - Pointers stored in index structures
  - Easy to locate pointers for a given memory block

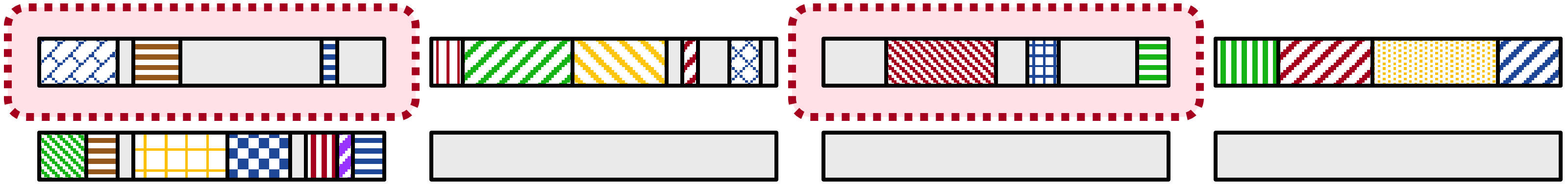
# Log-Structured Storage



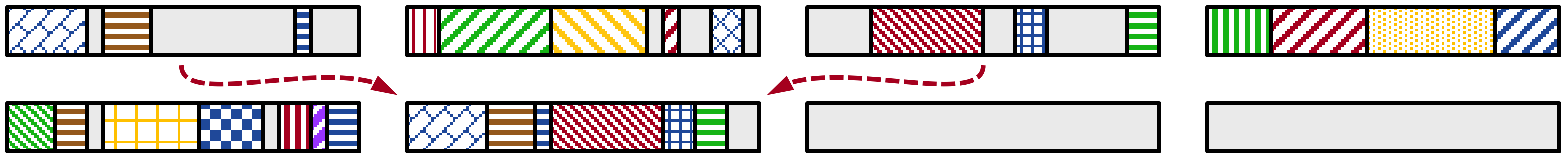

**Each segment replicated on disks of 3 backup servers**

# Log Cleaning

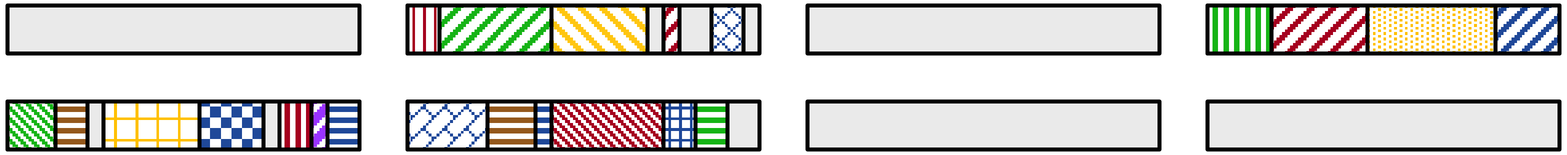
- Pick segments with lots of free space



- Copy live objects (survivors) into new segment(s)



- Free cleaned segments (use for new objects)



**Cleaning is incremental**

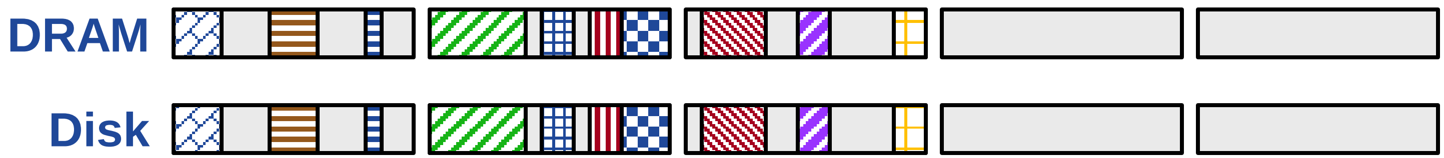
# Cleaning Cost

- **Cleaning cost increases with memory utilization**
  - U: fraction of bytes still live in cleaned segments

| | | | | | |
|---|---|---|---|---|---|
| **Bytes copied by cleaner** | **U** | **0.5** | **0.8** | **0.9** | **0.99** |
| **Bytes freed** | **1-U** | **0.5** | **0.2** | **0.1** | **0.01** |
| **Bytes copied/byte freed** | **U/(1-U)** | **1.0** | **4.0** | **9.0** | **99.0** |

- **Conflict between disk and memory**
  - Initial RAMCloud implementation: clean disk and memory together
  - Better to run disk at low utilization to reduce cleaning costs
  - But, this would mean low utilization of DRAM too

# Two-Level Cleaning

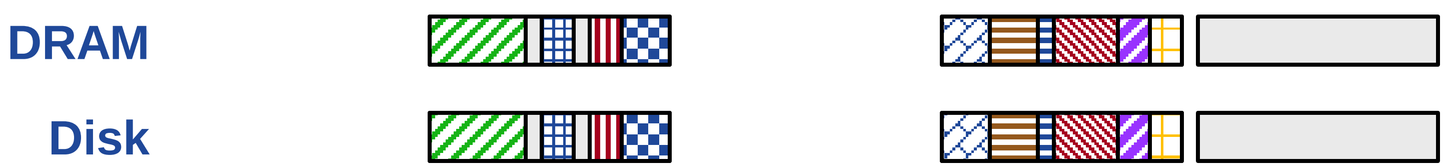



**Compaction:**

- Clean single segment in memory
- Free unused memory space
- No change to disk log


**Combined Cleaning:**

- Clean multiple segments
- Write new survivor segments (disk & memory)
- Free old segments (disk & memory)

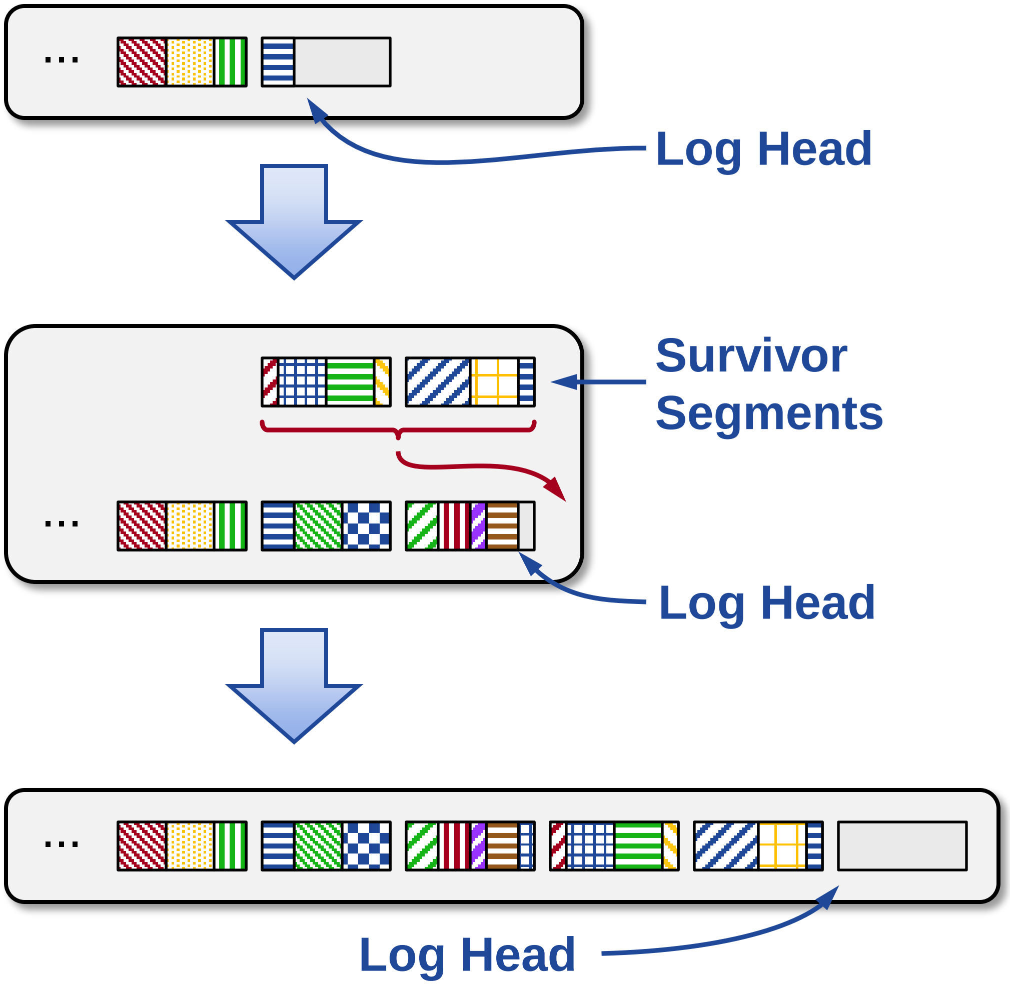


# Two-Level Cleaning, cont'd

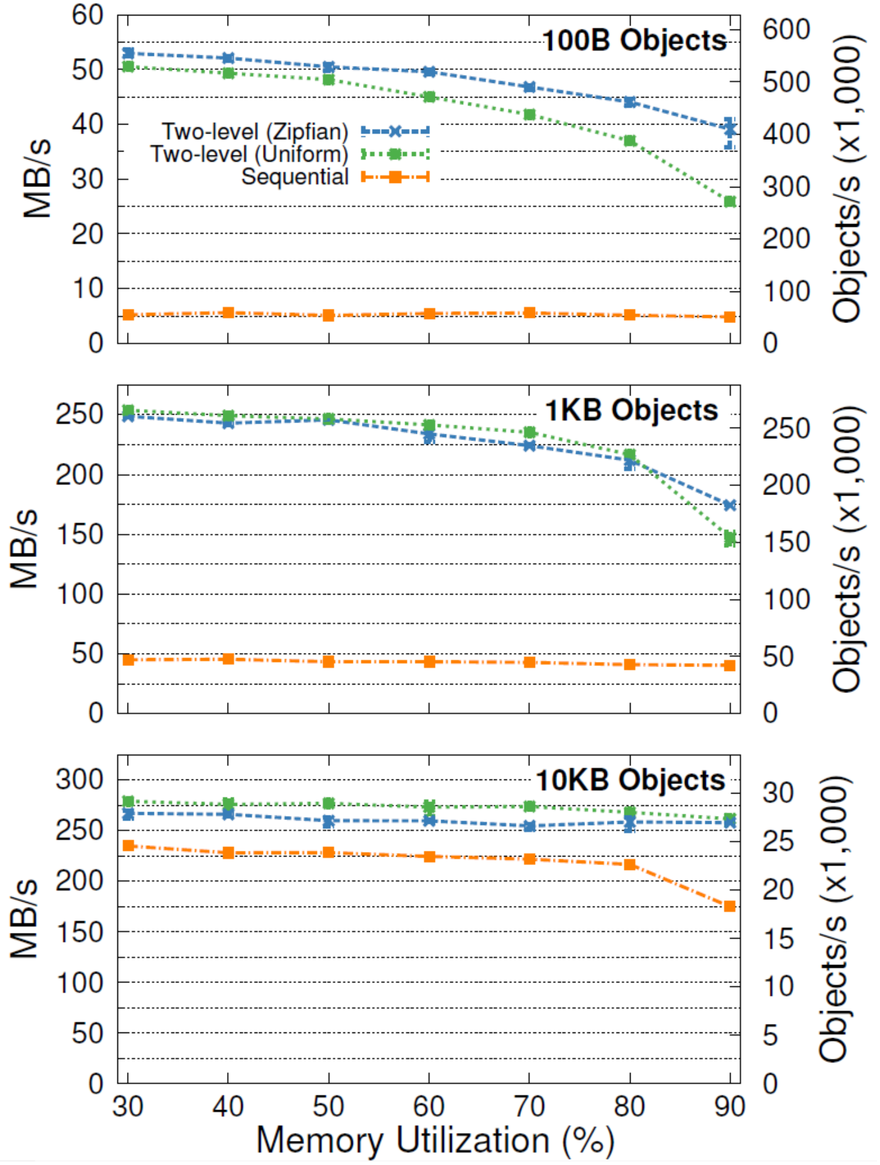
- **Best of both worlds:**
  - Optimize utilization of memory
    (can afford high bandwidth cost for compaction)
  - Optimize disk bandwidth
    (can afford extra disk space to reduce cleaning cost)

# Parallel Cleaning

- **Cleaner runs concurrently with normal requests**
- **Log is immutable (no updates in place)**
- **Survivor data written to "side log"**
  - No competition for log head
- **Synchronization points:**
  - Updates to hash table (cleaner moves object while being read/written)
  - Adding survivor segments to log
  - Freeing cleaned segments (wait for active requests to complete)



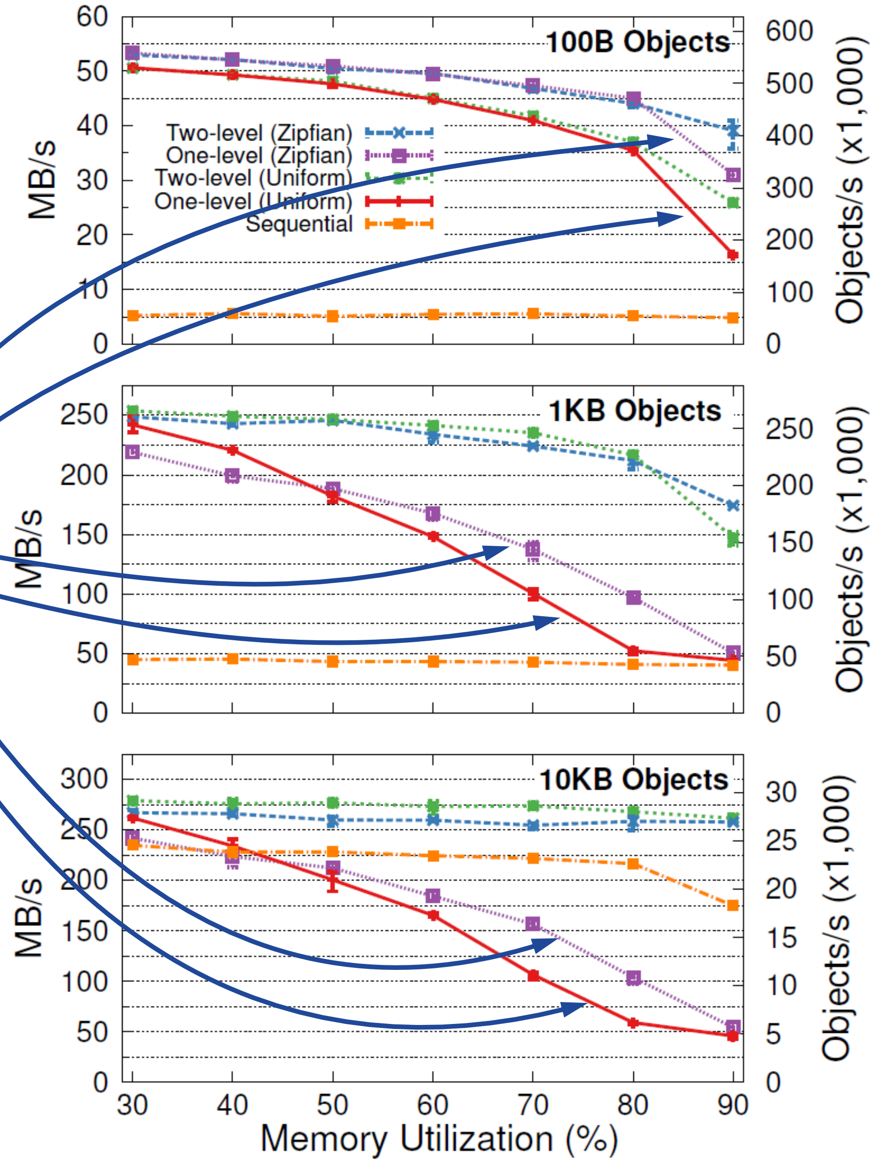
# Client Write Throughput



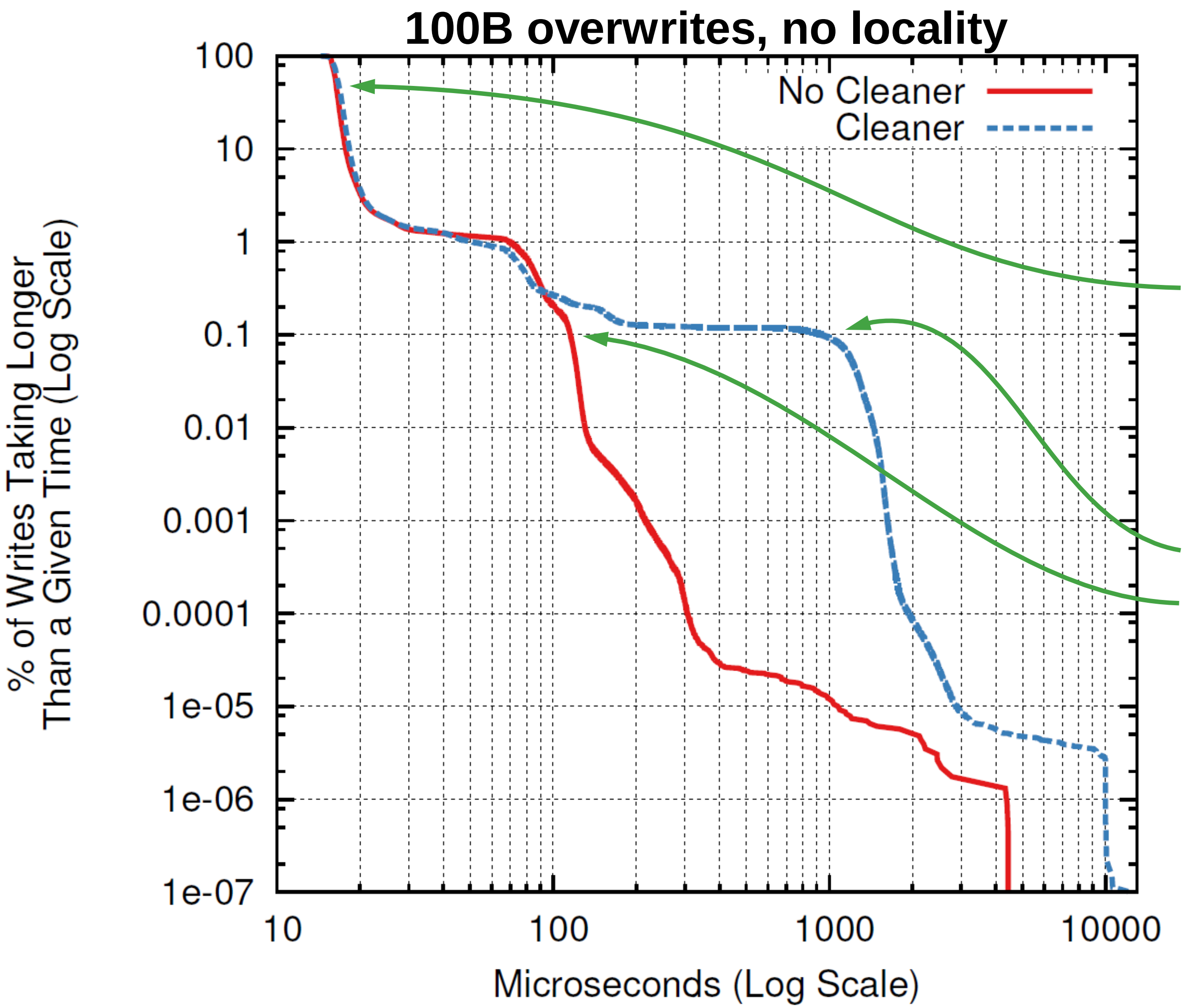

| Memory Utilization | Performance Degradation |
|---|---|
| 80% | 17-27% |
| 90% | 26-49% |
| 80% | 14-15% |
| 90% | 30-42% |
| 80% | 3-4% |
| 90% | 3-6% |

# 1-Level vs. 2-Level Cleaning

One-Level Cleaning



# Cleaner's Impact on Latency



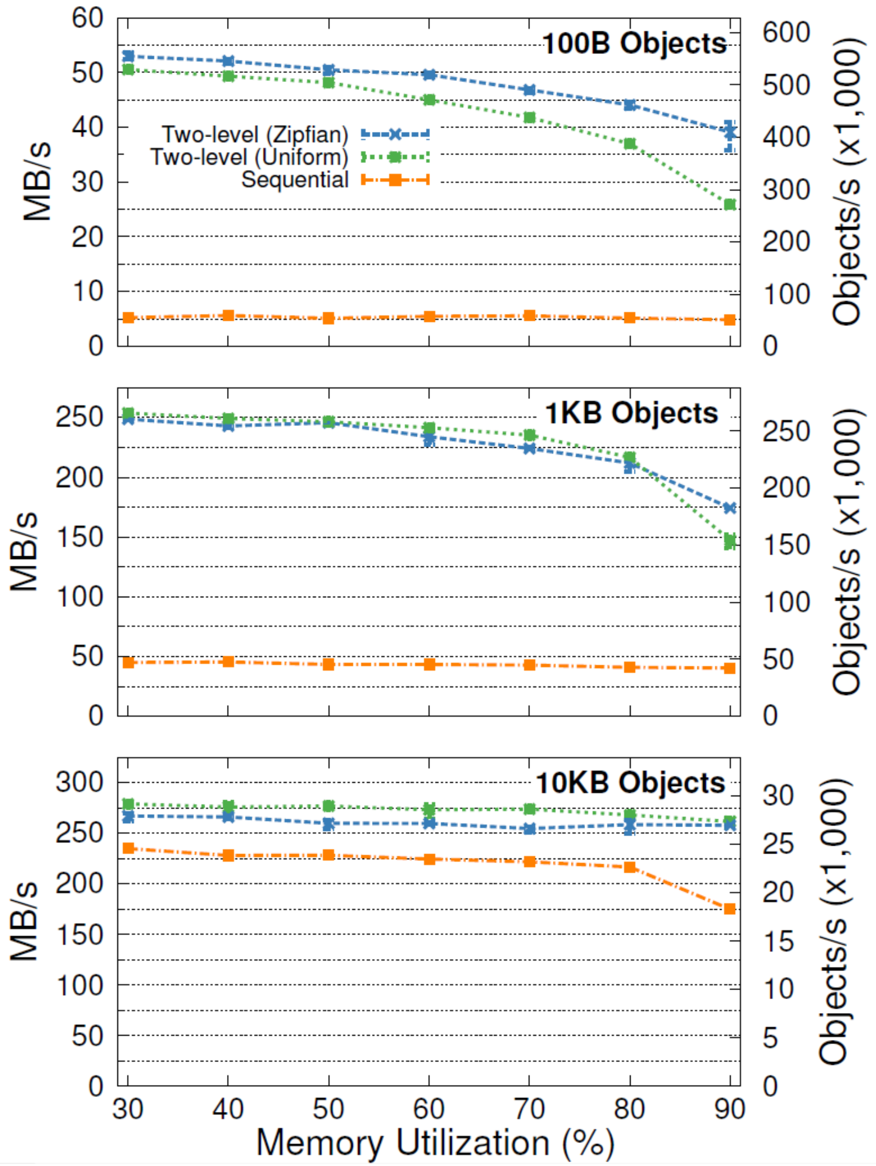

Median:
- With cleaning: 16.70µs
- No cleaner: 16.35µs

99.9%:
- With cleaning: 900µs
- No cleaner: 115µs

# Conclusion

- **Logging approach to storage allocation works well if pointers are restricted**
  - Allows 80-90% memory utilization
  - Good performance independent of workload
  - Supports concurrent cleaning: no pauses
- **Works particularly well in RAMCloud**
  - Manage both disk and DRAM with same mechanism
- **Also makes sense for other DRAM-based storage systems (see paper for details)**

# Client Throughput




80%: 17-27% degradation
90%: 26-49% degradation

80%: 14-15% degradation
90%: 30-42% degradation

80%: 3-4% degradation
90%: 3-6% degradation

# Tombstones

- **Server crash? Replay log on other servers to reconstruct lost data**
- **Tombstones identify deleted objects:**
  - Written into log when object deleted or overwritten
  - Info in tombstone:
    - Table id
    - Object key
    - Version of dead object
    - Id of segment where object stored
- **When can tombstones be deleted?**
  - After segment containing object has been cleaned (and replicas deleted on backups)
- **Note: tombstones are a mixed blessing**