Log-Structured Memory for DRAM-Based Storage

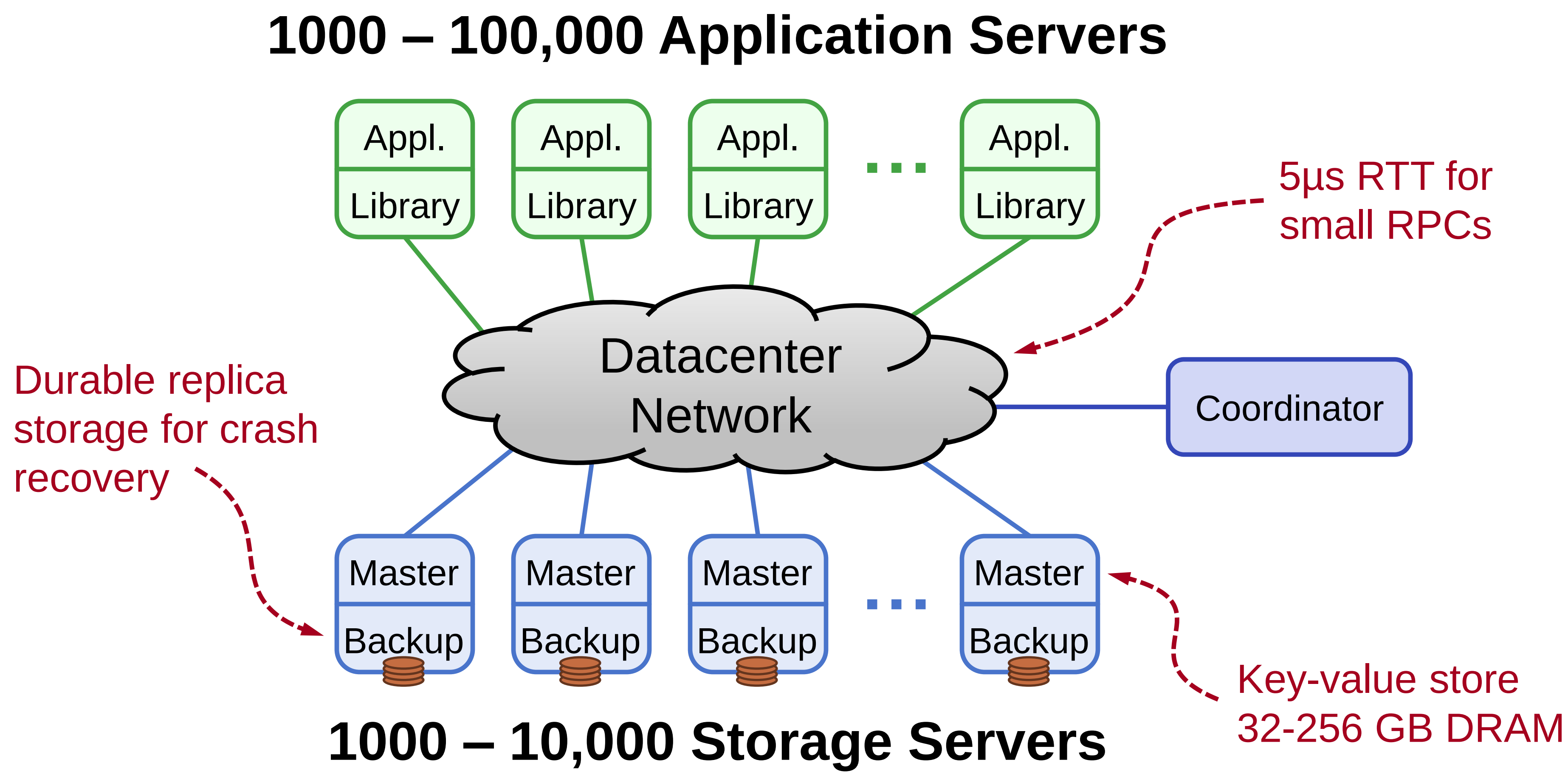
Stephen Rumble, Ankita Kejriwal, and John Ousterhout

Stanford University

Introduction

- **Traditional memory allocators can't provide all of**
  - Fast allocation/deallocation
  - Handle changing workloads
  - Efficient use of memory
- **RAMCloud: log-structured allocator**
  - Incremental copying garbage collector
  - Two-level approach to cleaning (separate policies for disk and DRAM)
  - Concurrent cleaning (no pauses)
- **Results:**
  - High performance even at 80-90% memory utilization
  - Handles changing workloads
  - Makes sense for any DRAM-based storage system

RAMCloud Overview

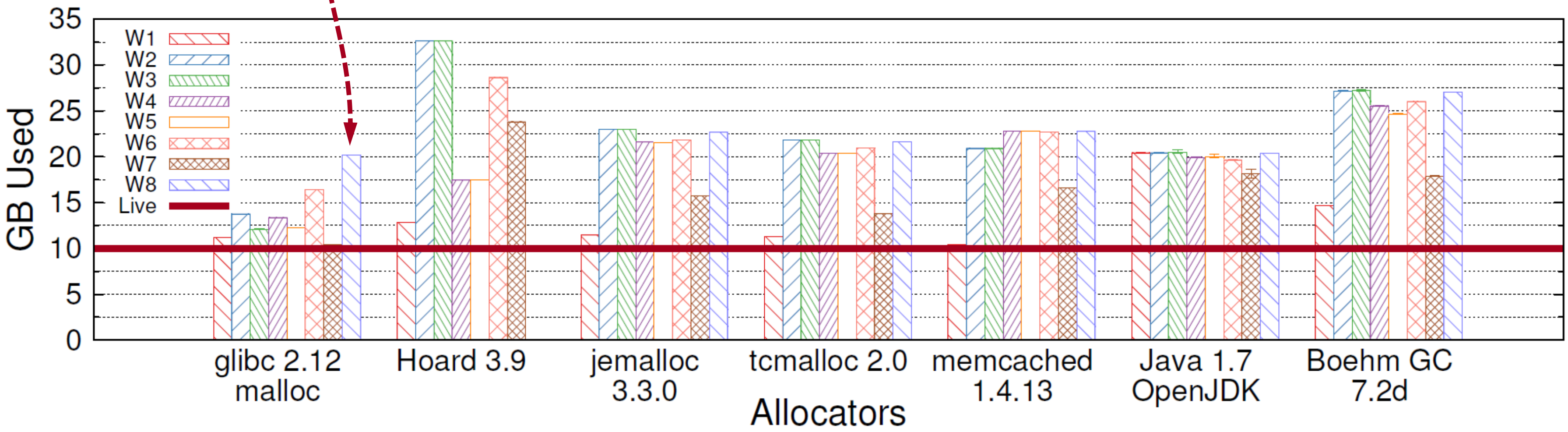


All data in DRAM at all times

Workload Sensitivities

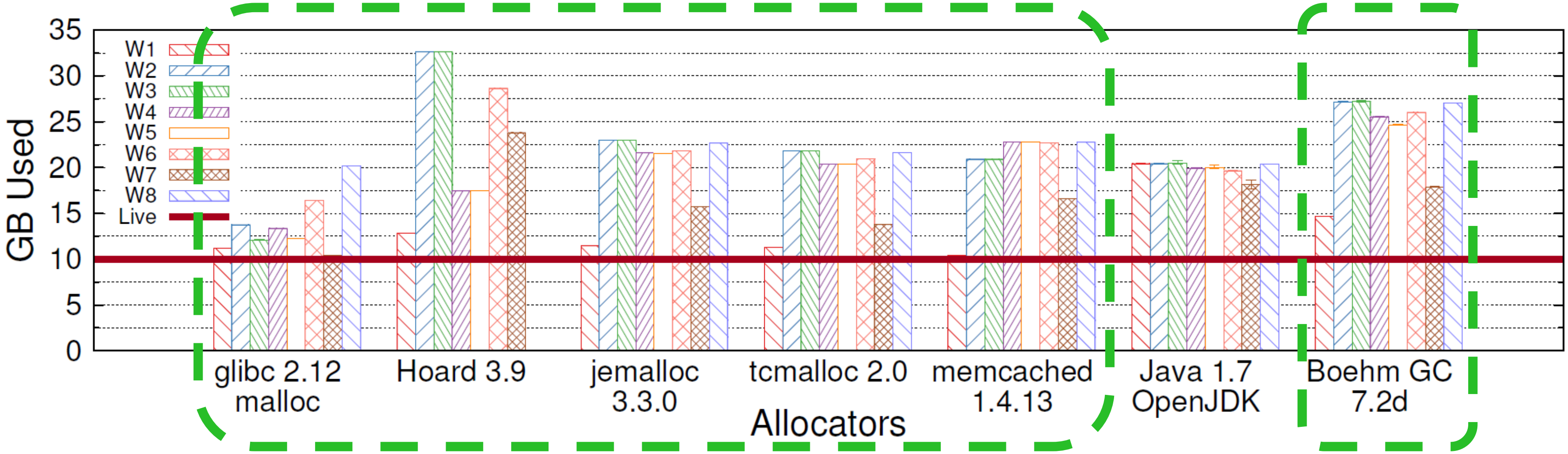
glibc malloc: 20 GB memory to hold 10 GB data under workload W8:

- Allocate many 50-150B objects
- Then delete 90%, write new 5-15KB objects

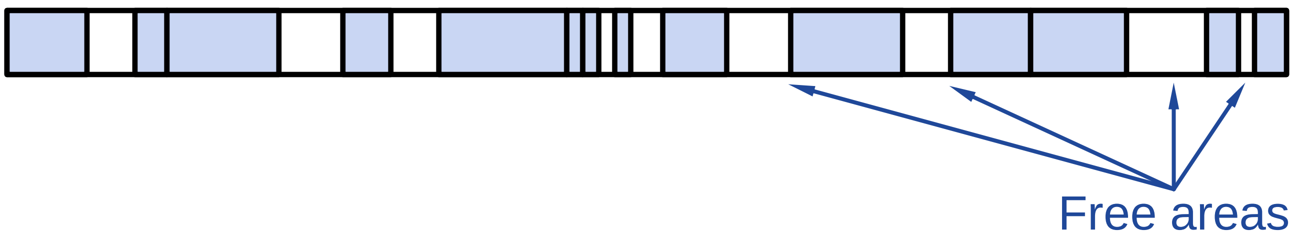


- **7 memory allocators, 8 workloads**
  - Total live data constant (10 GB)
  - But workload changes (except W1)
- **All allocators waste at least 50% of memory in some situations**

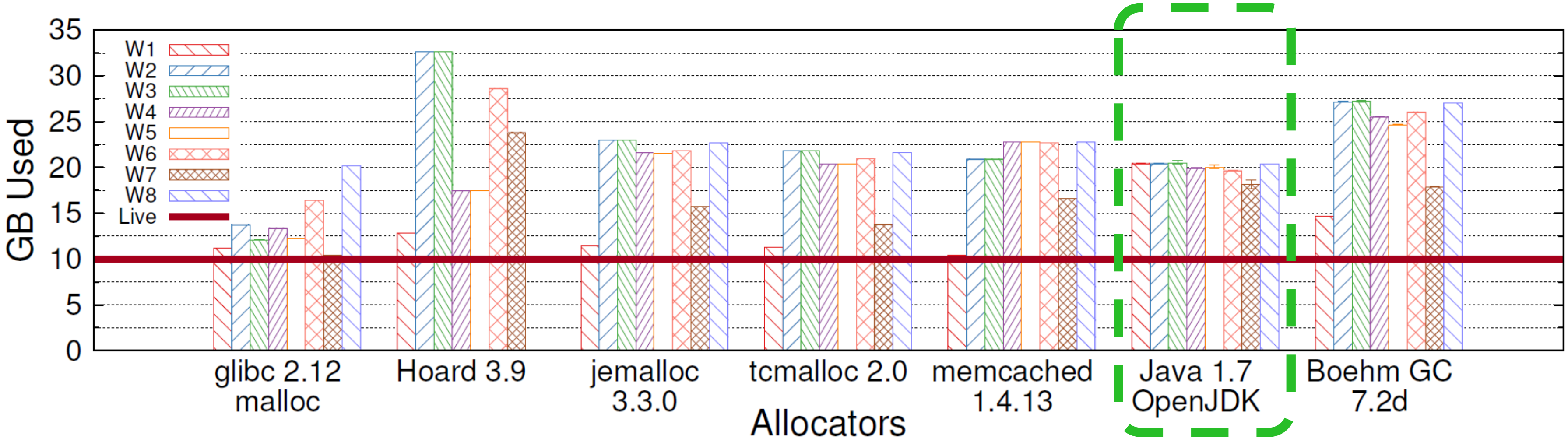
Non-Copying Allocators



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

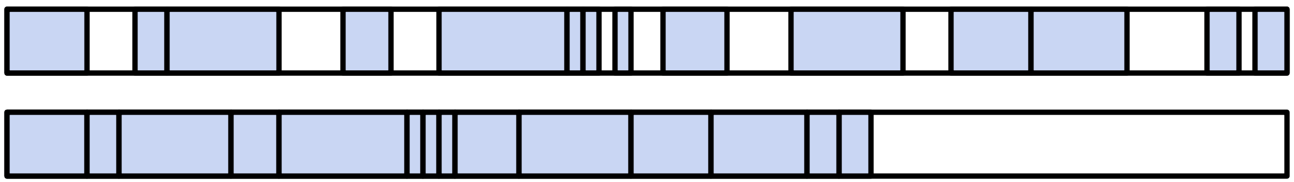


Copying Garbage Collectors



Before collection:

After collection:

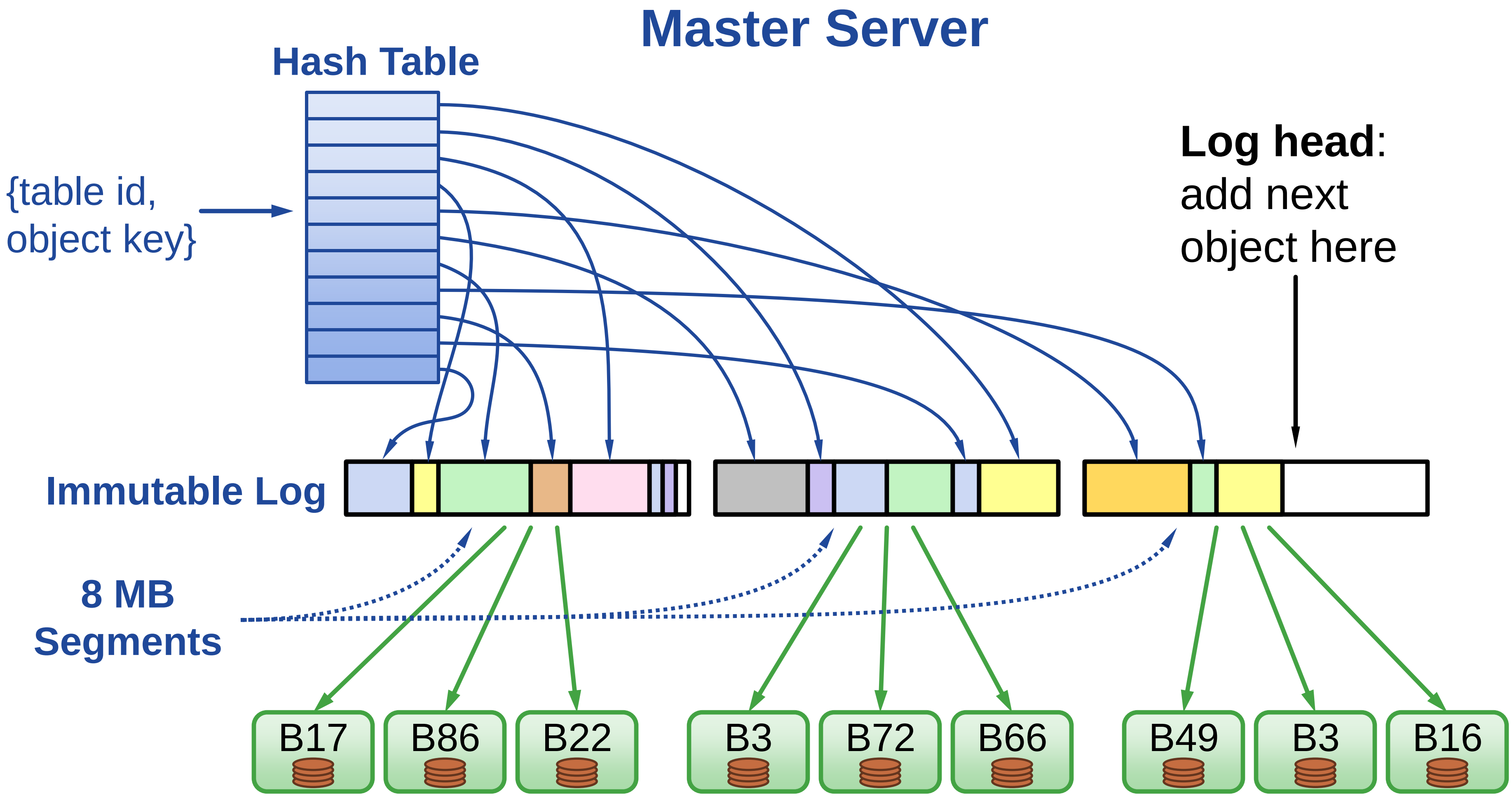


- **Must scan all memory to update pointers**
  - Expensive, scales poorly
  - Wait for lots of free space before running GC
- **State of the art: 3-5x overallocation of memory**
- **Long pauses: 3+ seconds for full GC**

Allocator for RAMCloud

- **Requirements:**
  - Must use copying approach
  - Must collect free space incrementally
- **Storage system advantage: pointers restricted**
  - Pointers stored in index structures
  - Easy to locate pointers for a given memory block
  - Enables incremental copying
- **Can achieve overall goals:**
  - Fast allocation/deallocation
  - Insensitive to workload changes
  - 80-90% memory utilization

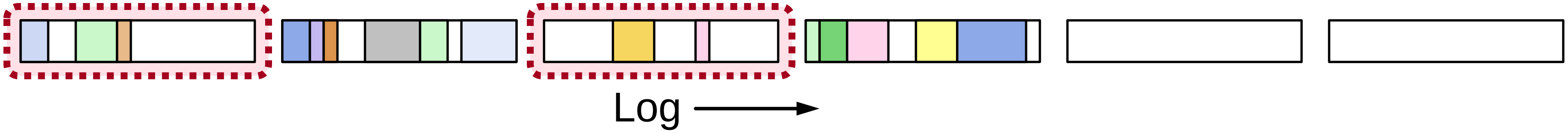
Log-Structured Storage



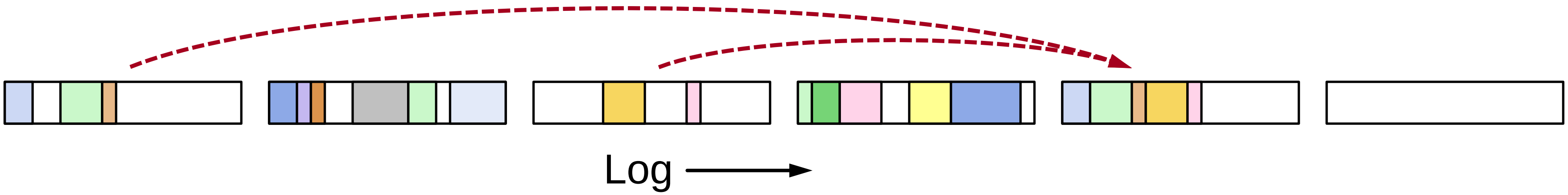
Each segment replicated on disks of 3 backup servers

Log Cleaning

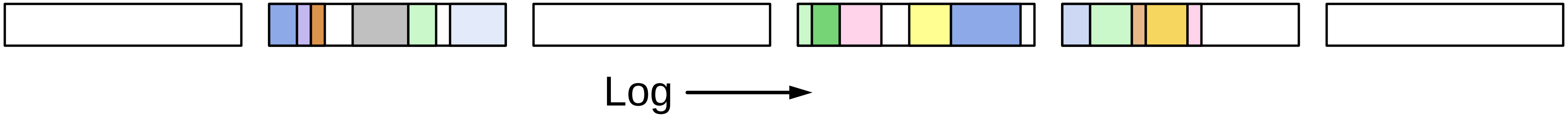
1. **Pick segments with lots of free space:**



2. **Copy live objects (survivors):**



3. **Free cleaned segments (and backup replicas)**



Cleaning is incremental

Cleaning Cost

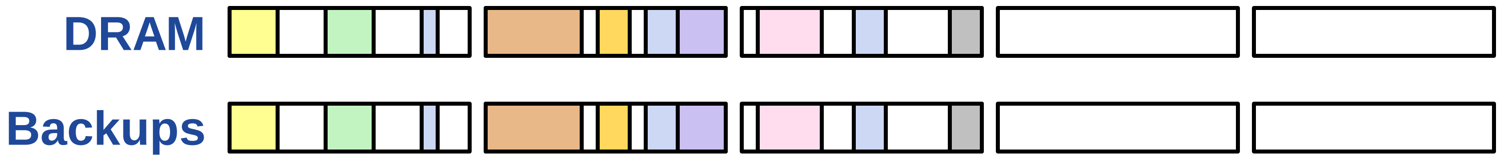
	U: fraction of live bytes in cleaned segments		
	0.5	0.9	0.99
Bytes copied by cleaner (U)	0.5	0.9	0.99
Bytes freed (1-U)	0.5	0.1	0.01
Bytes copied/byte freed (U/(1-U))	1.0	9.0	99.0

Conflicting Needs:

	Capacity	Bandwidth
Memory	expensive	cheap
Disk	cheap	expensive

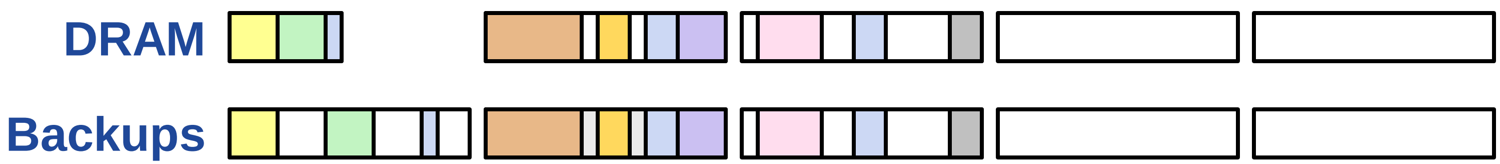
Need different policies for cleaning disk and memory

Two-Level Cleaning



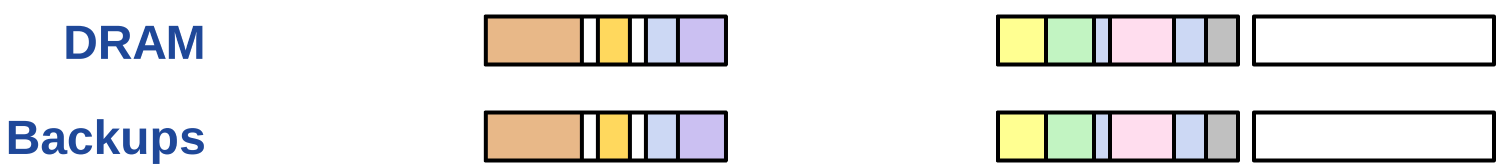
Compaction:

- Clean single segment in memory
- No change to replicas on backups



Combined Cleaning:

- Clean multiple segments
- Free old segments (disk & memory)

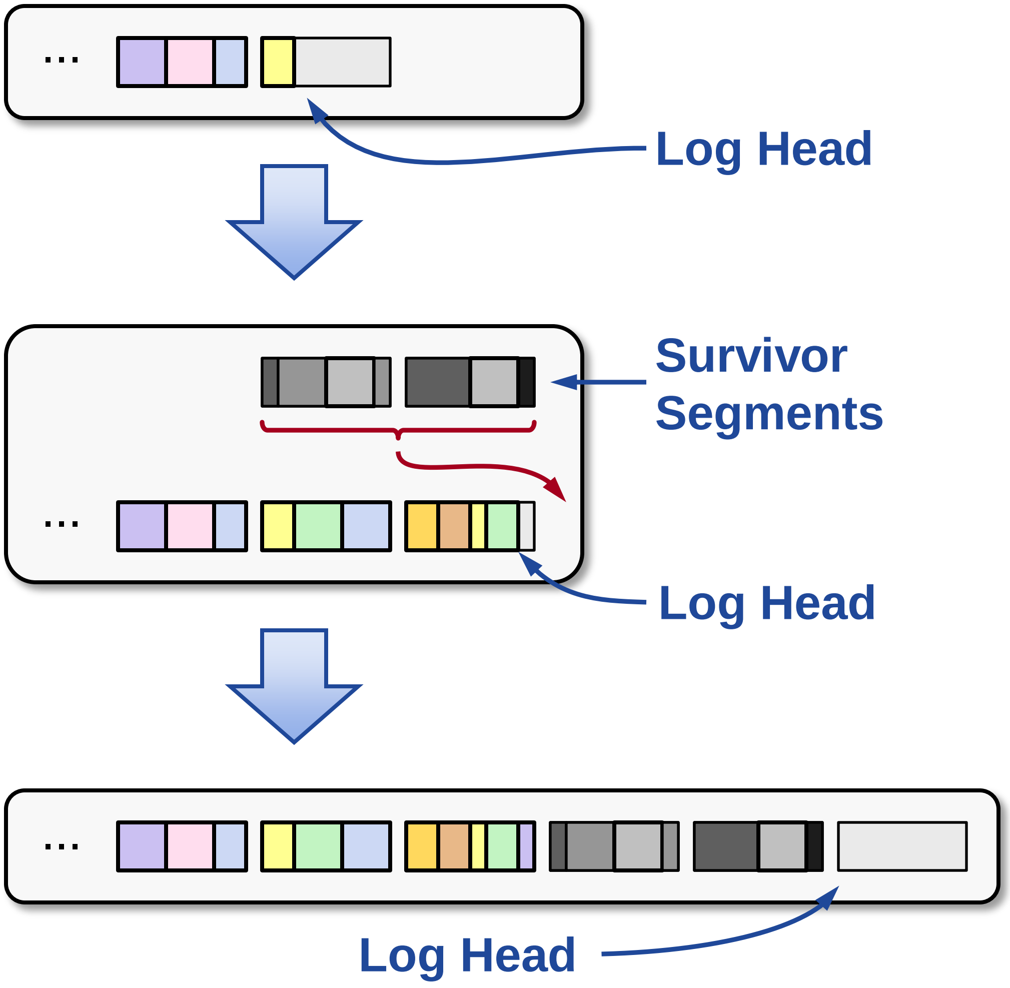


Two-Level Cleaning, cont'd

- **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)

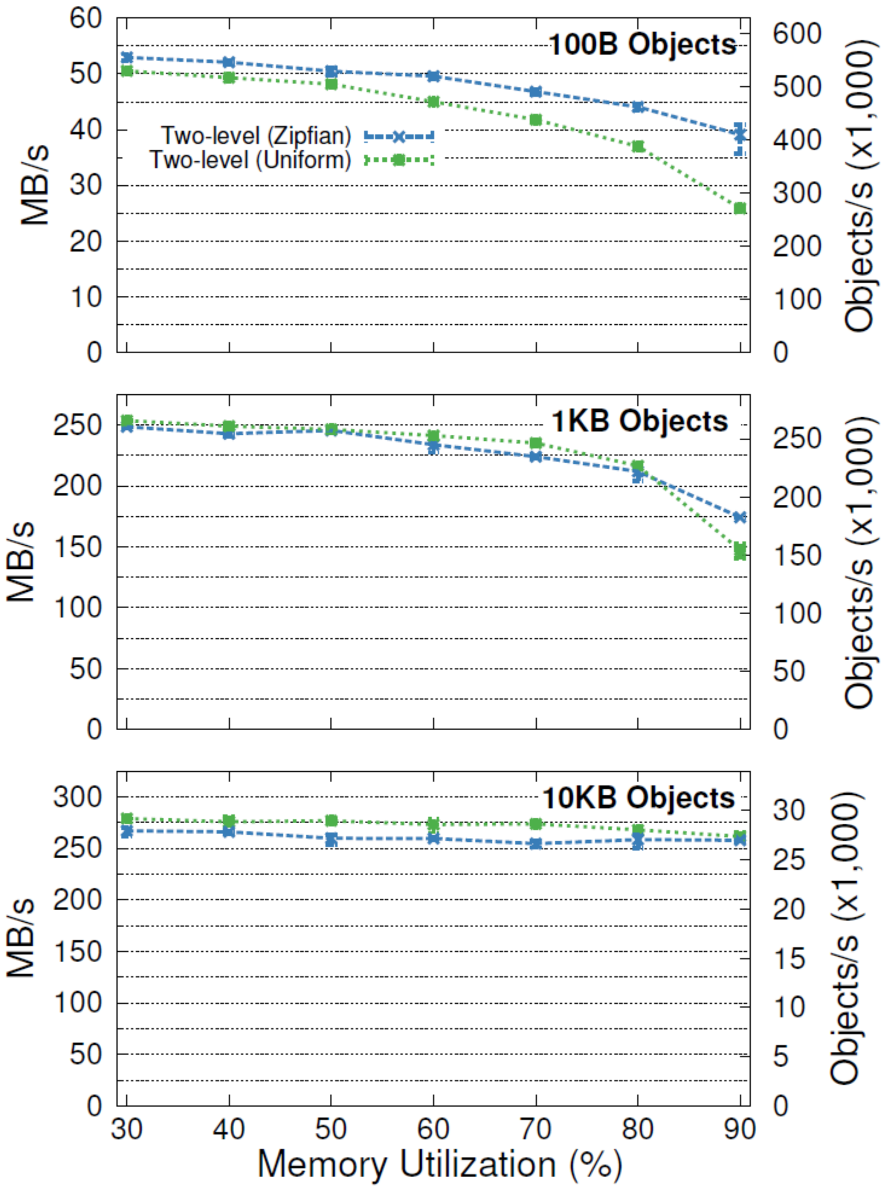
Parallel Cleaning

- **Survivor data written to "side log"**
  - No competition for log head
  - Different backups for replicas
- **Synchronization points:**
  - Updates to hash table
  - Adding survivor segments to log
  - Freeing cleaned segments



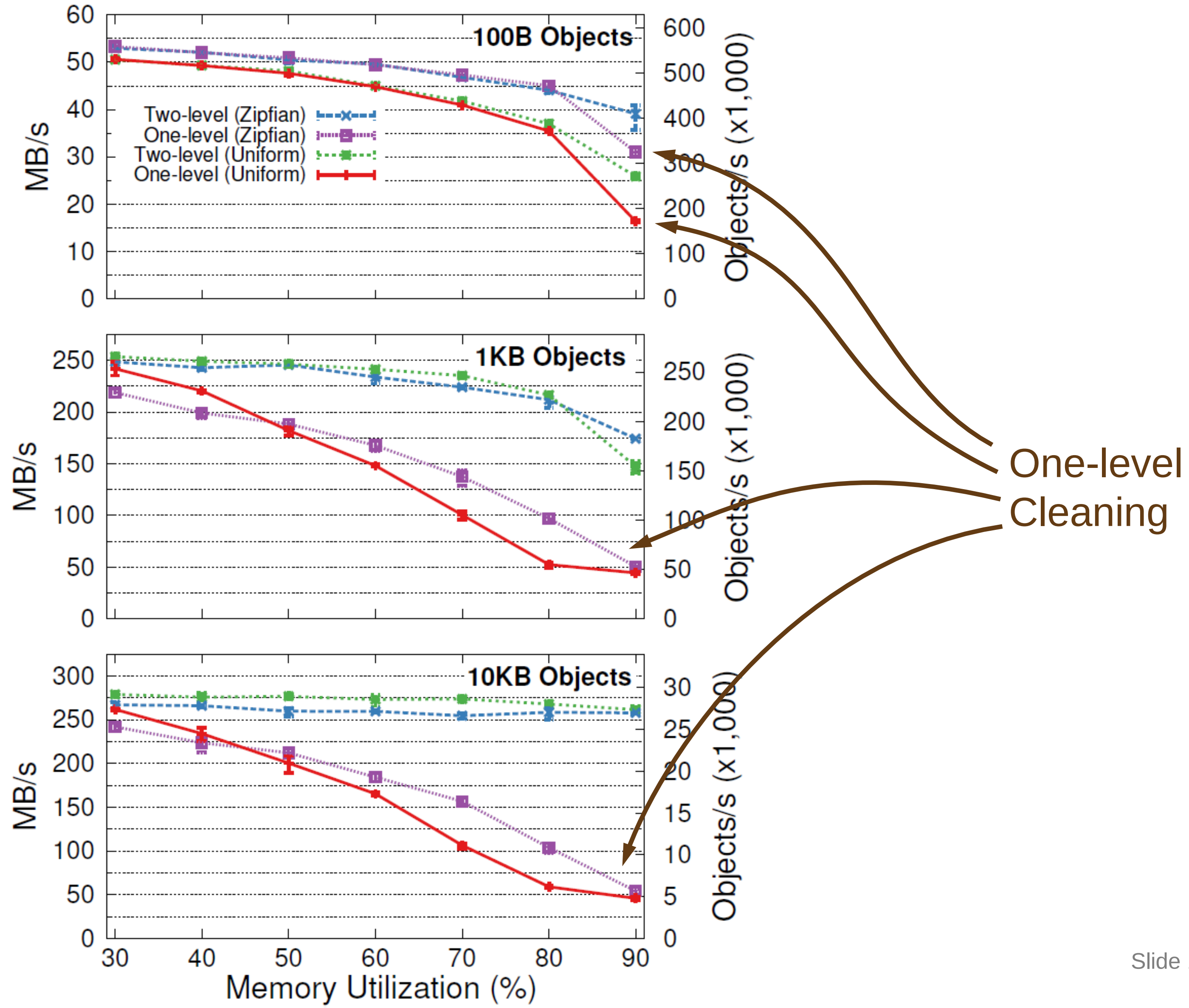
Throughput vs. Memory Utilization

1 master,
3 backups,
1 client,
concurrent
multi-writes



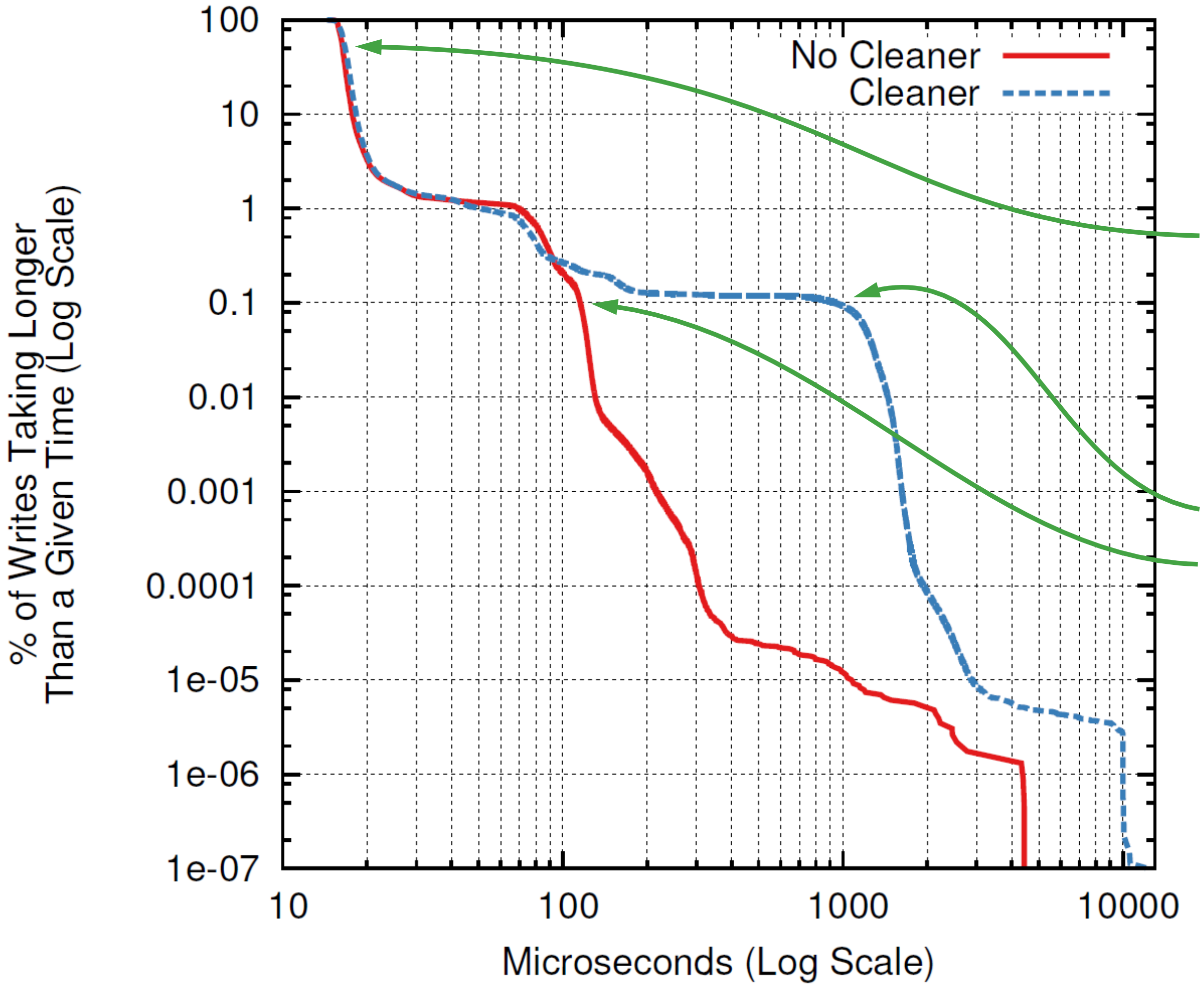
Memory Utilization	Performance Degradation
80%	17-27%
90%	26-49%
80%	14-15%
90%	30-42%
80%	3-4%
90%	3-6%

1-Level vs. 2-Level Cleaning



Cleaner's Impact on Latency

1 client, sequential 100B overwrites, no locality, 90% utilization



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

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

Additional Material in Paper

- **Tombstones: log entries to mark deleted objects**
  - Mixed blessing: impact cleaner performance
- **Preventing memory deadlock**
  - Need space to free space
- **Fixed segment selection defect in LFS**
- **Modified memcached to use log-structured memory:**
  - 15-30% better memory utilization
  - 3% higher throughput
  - Negligible cleaning cost (5% CPU utilization)
- **YCSB benchmarks vs. HyperDex and Redis:**
  - RAMCloud better except vs. Redis under write-heavy workloads with slow RPC.

Related Work

- **Storage allocators and garbage collectors**
  - Performance limited by lack of control over pointers
  - Some slab allocators almost log-like (slab <=> segment)
- **Log-structured file systems**
  - All info in DRAM in RAMCloud (faster, more efficient cleaning)
- **Other large-scale storage systems**
  - Increasing use of DRAM:
    Bigtable/LevelDB, Redis, memcached, H-Store, ...
  - Log-structured mechanisms for distributed replication
  - Tombstone-like objects for deletion
  - Most use traditional memory allocators

Conclusion

- **Logging approach is an efficient way to allocate memory (if pointers are restricted)**
  - Allows 80-90% memory utilization
  - Good performance (no pauses)
  - Tolerates workload changes
- **Works particularly well in RAMCloud**
  - Manage both disk and DRAM with same mechanism
- **Also makes sense for other DRAM-based storage systems**

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**