RAMCloud and the Low-Latency Datacenter

John Ousterhout
Stanford Platform Laboratory

Introduction

- **Most important driver for innovation in computer systems:**
  Rise of the datacenter
- **Phase 1: large scale**
- **Phase 2: low latency**
- **RAMCloud: new class of storage for low-latency datacenters**
- **Potential impact: enable new class of applications**

How Many Datacenters?

- **Capitalize IT like other infrastructure (power, water, highways, telecom)?**
  - $1-10K per person?
  - 0.5-5 datacenter servers/person?

	U.S.	World
Servers	**0.15-1.5B**	**3.5-35B**
Datacenters	**1500-15,000**	**35,000-350,000**

(assumes 100,000 servers/datacenter)

- **Computing in 10 years:**
  - Most non-mobile computing (i.e. Intel processors) in datacenters
  - Devices provide user interfaces

Evolution of Datacenters

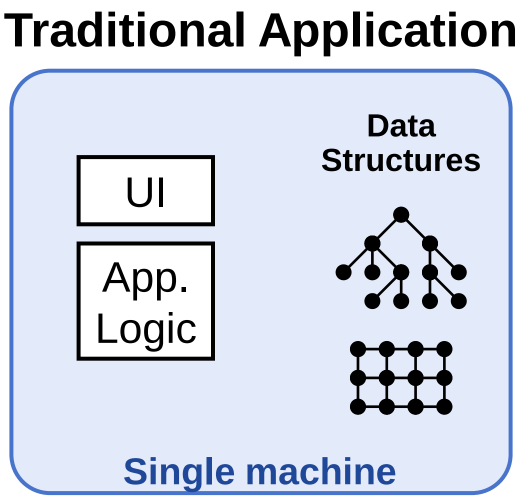
- **Phase 1: manage scale**
  - 10,000-100,000 servers within 50m radius
  - 1 PB DRAM
  - 100 PB disk storage
  - Challenge: how can one application harness thousands of servers?
    - Answer: MapReduce, etc.
- **But, communication latency high:**
  - 300-500µs round-trip times
  - Must process data sequentially to hide latency
    (e.g. MapReduce)
  - Interactive applications limited in functionality

Evolution of Datacenters

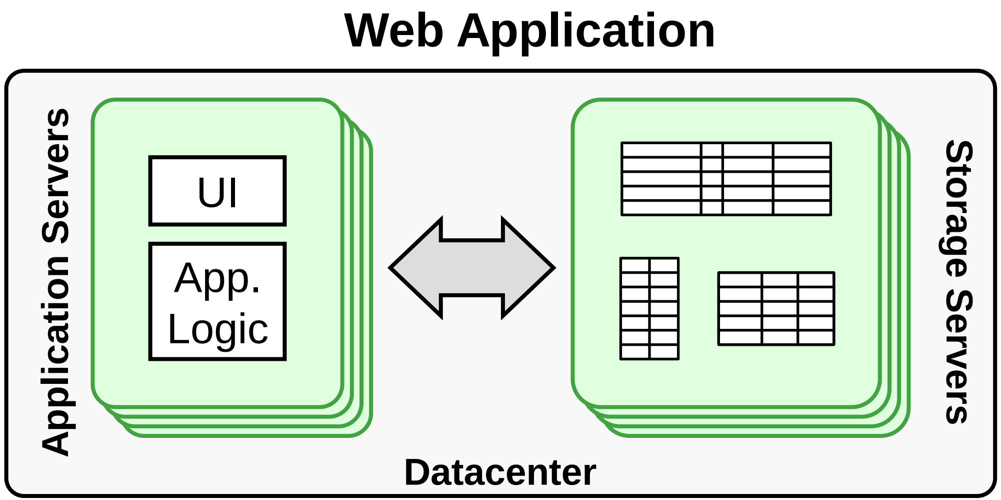
- **Phase 1: manage scale**
  - 10,000-100,000 servers within 50m radius
  - 1 PB DRAM
  - 100 PB disk storage
  - Challenge: how can one application harness thousands of servers?
    - Answer: MapReduce, etc.
- **But, communication latency high:**
  - 300-500µs round-trip times
  - Must process data sequentially to hide latency
    (e.g. MapReduce)
  - Interactive applications limited in functionality

- **Phase 2: low latency**
  - Speed-of-light limit: 1µs
  - Round-trip time achievable today: 5-10µs
  - Practical limit (5-10 years): 2-3µs
- **Why does low latency matter?**
- **How does low latency affect system architecture?**

Why Does Latency Matter?



<< 1µs latency

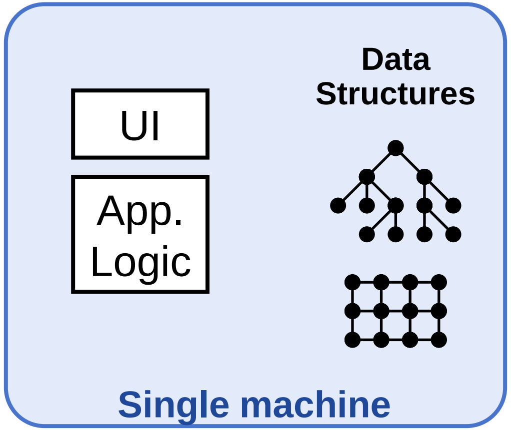


0.5-10ms latency

- **Large-scale apps struggle with high latency**
  - Random access data rate has not scaled!
  - Facebook: limited to 100-150 internal requests per page

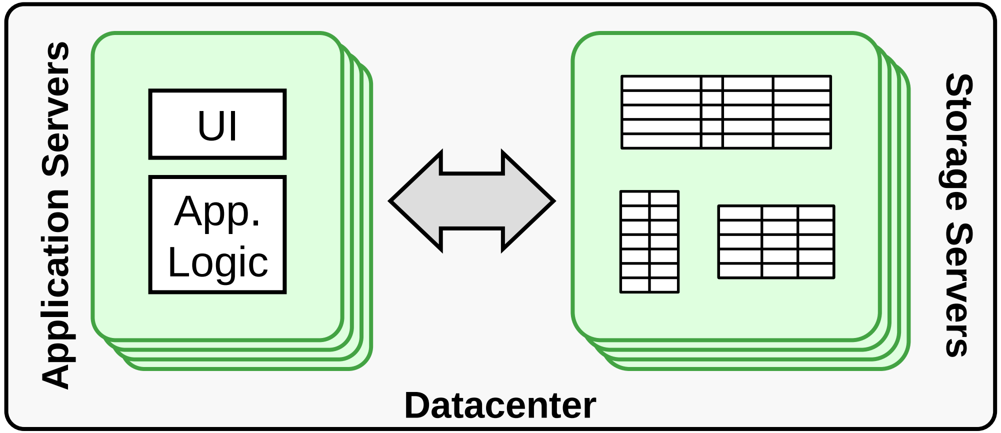
Goal: Scale and Latency

Traditional Application



<< 1µs latency

Web Application

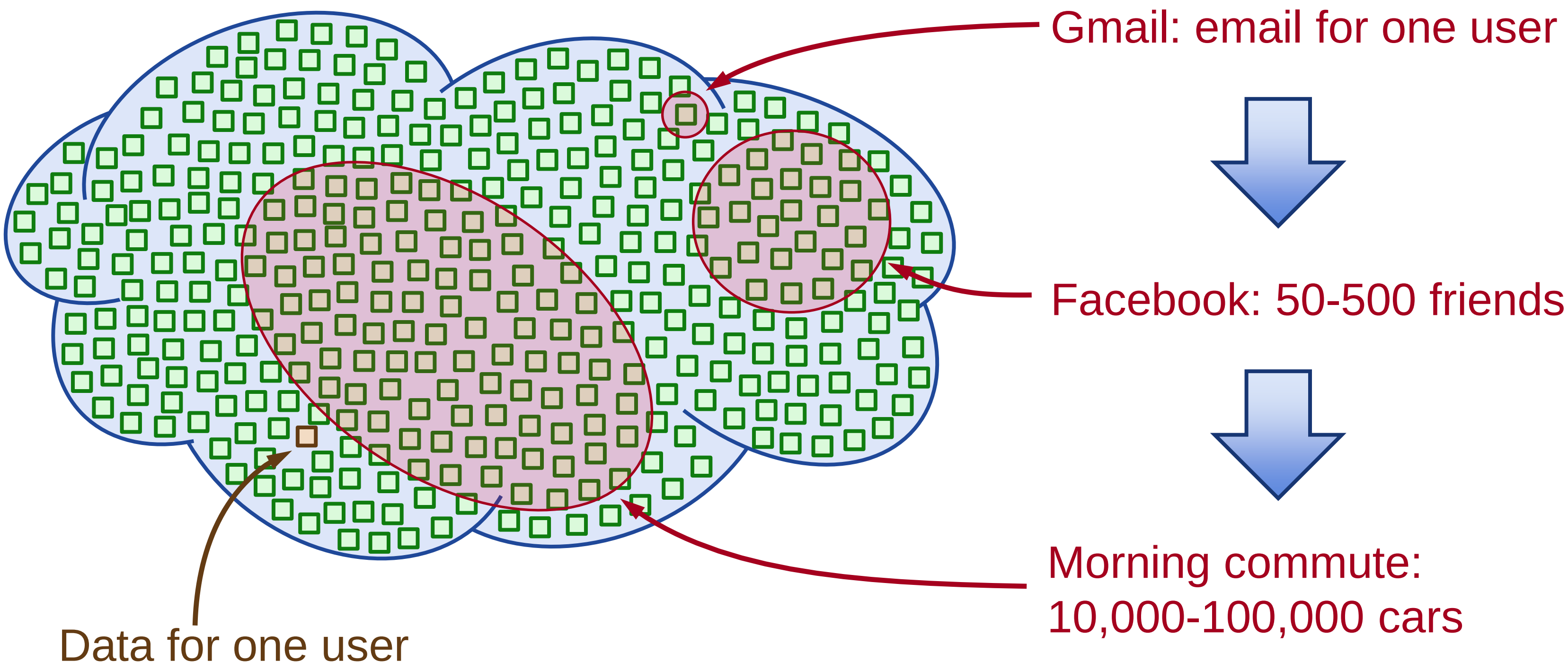


~~0.5-10ms~~ latency
5-10µs

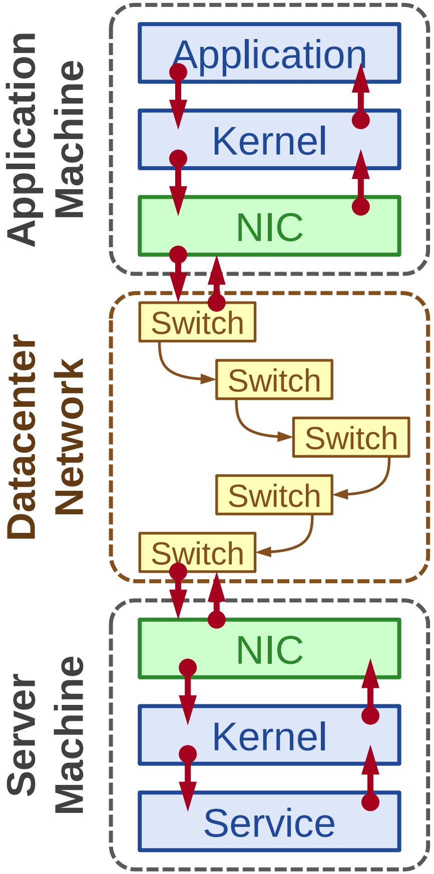
- **Enable new class of applications:**
  - Large-scale graph algorithms (machine learning?)
  - Collaboration at scale?

Large-Scale Collaboration

"Region of Consciousness"



Getting to Low Latency



Network delay (per switch):

2010	Today	Limit
10-30µs	0.5µs	30ns

Eliminate buffering

Kernel overhead (per transit):

2010	Today
10-15µs	0.0µs

Kernel bypass (access NIC directly from app)

NIC overhead (per transit):

2010	Today	Limit
2-32µs	0.75µs	50ns

Radical new NIC-CPU integration

Achievable Round-Trip Latency

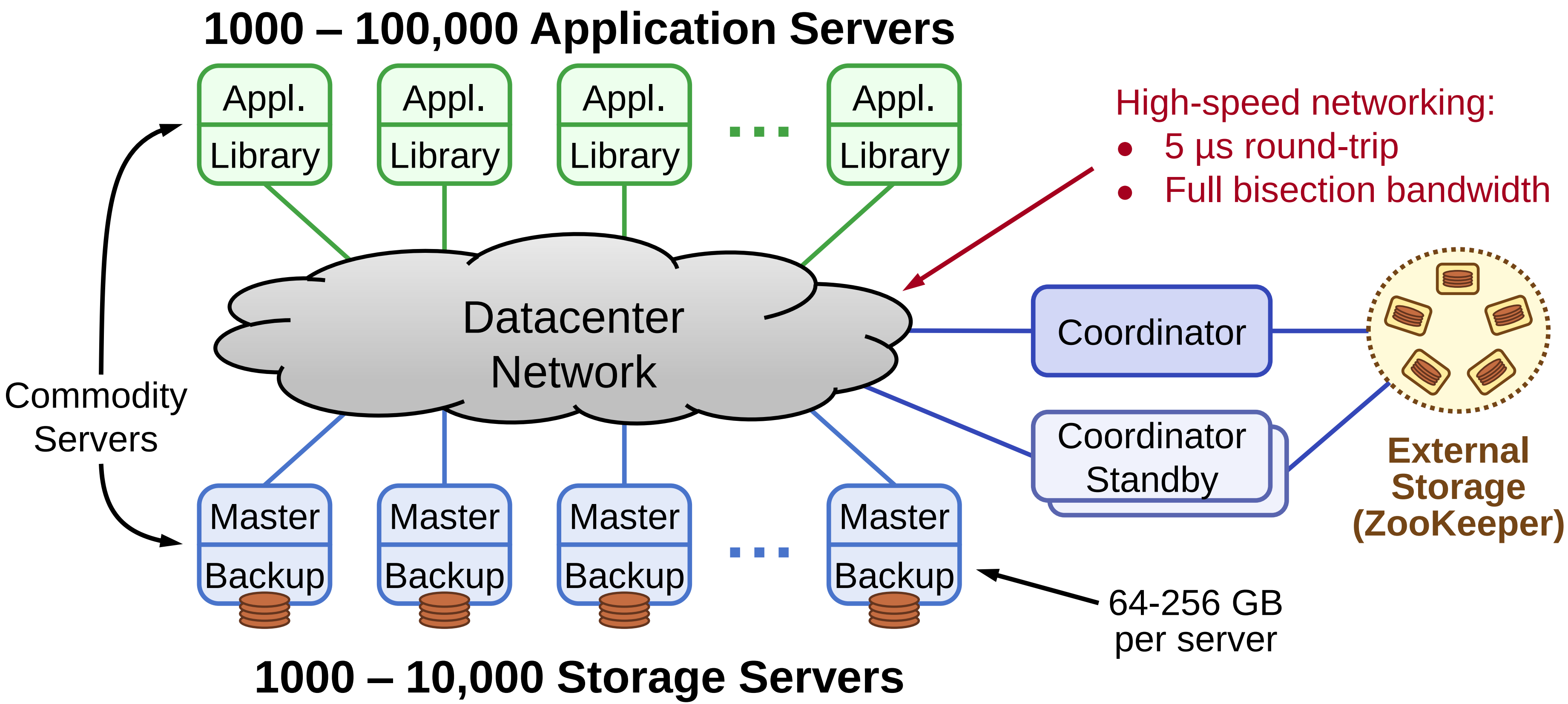
Component	2010	Possible Today	5-10 Years
Switching fabric	100-300µs	5µs	0.2µs
Software	50µs	2µs	1µs
NIC	8-128µs	3µs	0.2µs
Propagation delay	1µs	1µs	1µs
Total	200-400µs	11µs	2.4µs

RAMCloud

Storage system for low-latency datacenters:

- **General-purpose**
- **All data always in DRAM (not a cache)**
- **Durable and available**
- **Scale: 1000+ servers, 100+ TB**
- **Low latency: 5-10µs remote access**

RAMCloud Architecture



Example Configurations

	2010	2015-2020
# servers	1000	2000
GB/server	648 GB	512 GB
Total capacity	64 TB	1 PB
Total server cost	$4M	$7M
$/GB	$60	$7

For $100K today:

- One year of Amazon customer orders (10 TB?)
- One year of United flight reservations (10 TB?)

Data Model: Key-Value Store

- **Table Operations**

  createTable(*name*) → *id*
  getTableId(*name*) → *id*
  dropTable(*name*)

- **Basic Operations**

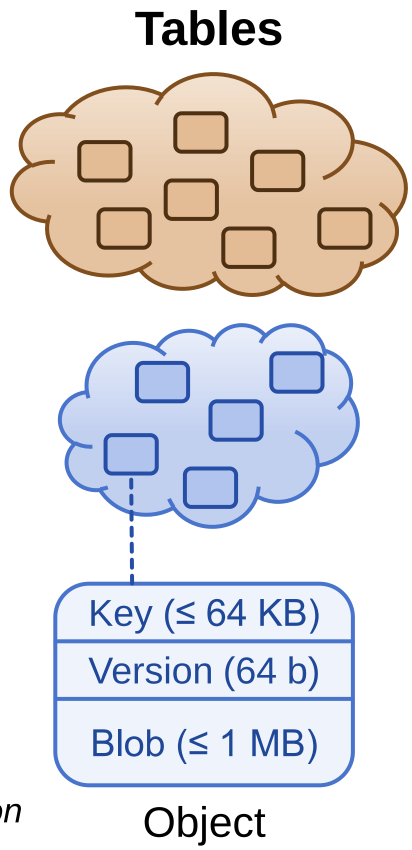
  read(*tableId*, *key*) → *value*, *version*
  write(*tableId*, *key*, *value*) → *version*
  delete(*tableId*, *key*)

- **Bulk Operations**

  multiRead([*tableId*, *key*]*) → [*value*, *version*]*
  multiWrite([*tableId*, *key*, *value*]*) → [*version*]*
  multiDelete([*tableId*, *key*]*)
  enumerateTable(*tableId*) → [*key*, *value*, *version*]*

- **Atomic Operations**

  increment(*tableId*, *key*, *amount*) → *value*, *version*
  conditionalWrite(*tableId*, *key*, *value*, *version*) → *version*



Recent additions:

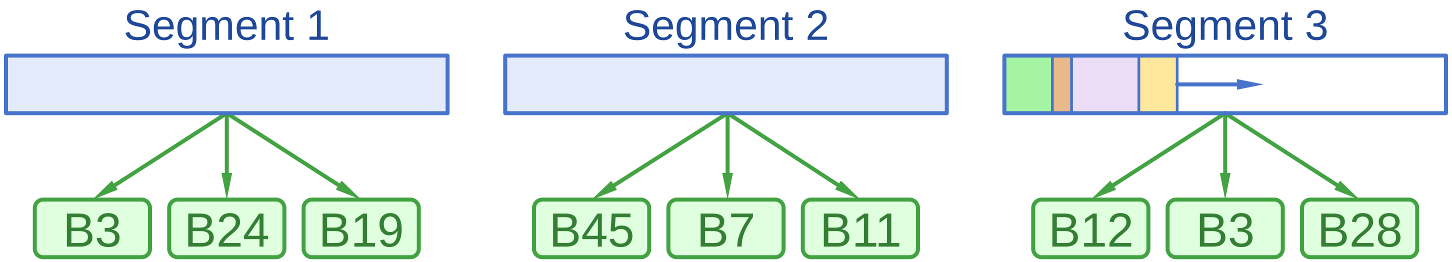
- **Secondary indexes**
- **Multi-object transactions**

RAMCloud Performance

- **Using Infiniband networking (24 Gb/s, kernel bypass)**
  - Other networking also supported, but slower
- **Reads:**
  - **100B objects: 4.7µs**
  - 10KB objects: 10µs
  - Single-server throughput (100B objects): 900 Kops/sec.
  - Small-object multi-reads: 2M objects/sec.
- **Durable writes:**
  - **100B objects: 13.5µs**
  - 10KB objects: 35µs
  - Small-object multi-writes: 400-500K objects/sec.

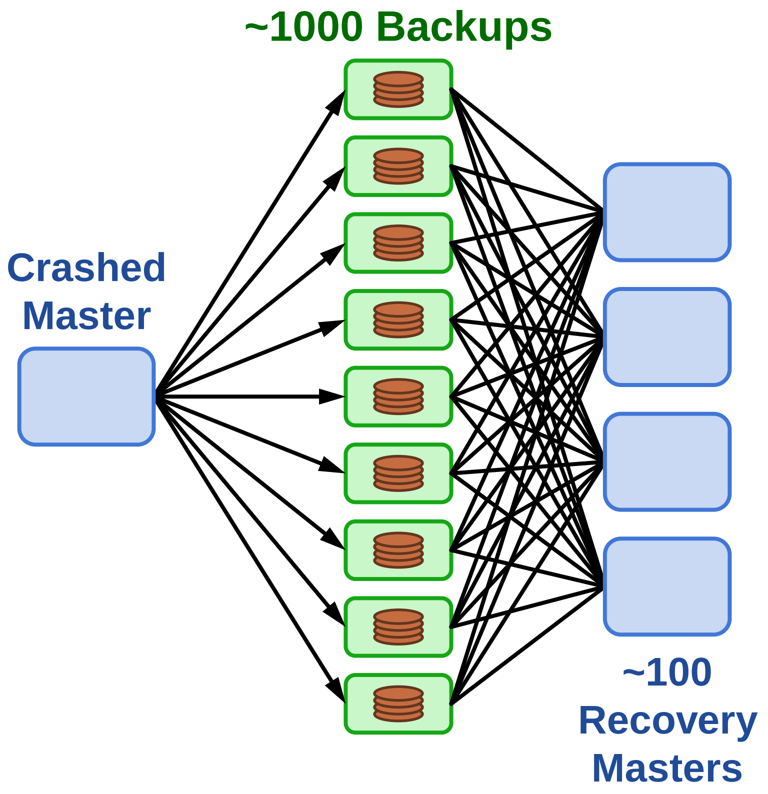
Data Durability

- **Objects (eventually) backed up on disk or flash**
- **Logging approach:**
  - Each master stores its objects in an append-only log
  - Log divided into segments
  - Segments replicated on multiple backups
  - Segment replicas scattered across entire cluster
- **For efficiency, updates buffered on backups**
  - Assume nonvolatile buffers (flushed during power failures)



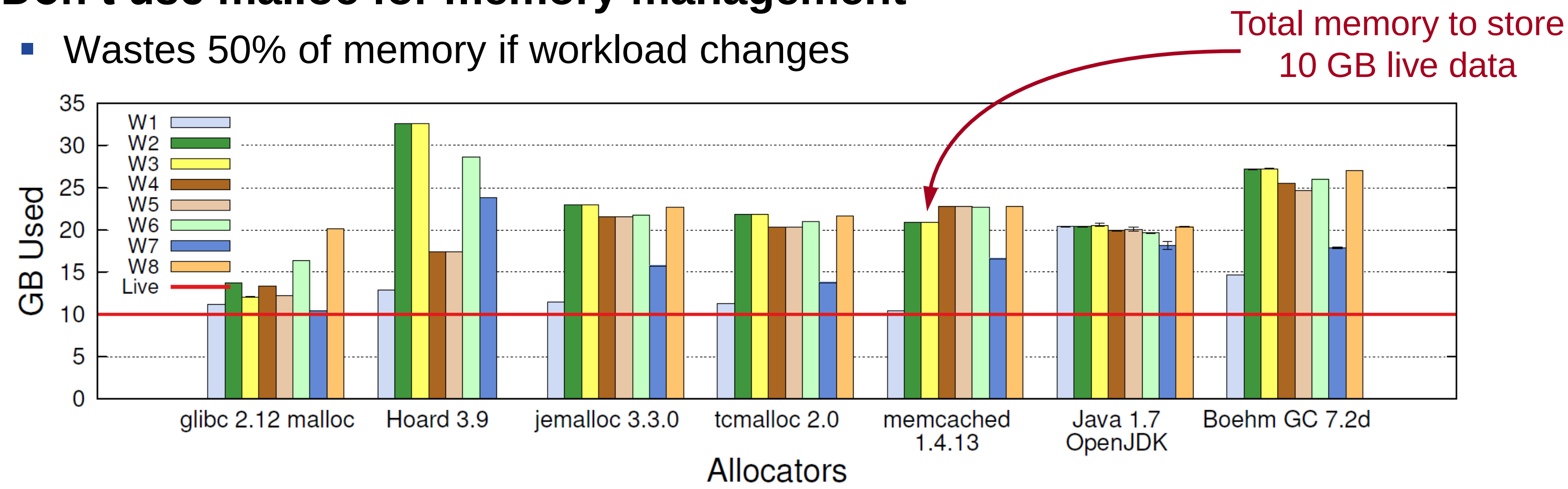
1-2 Second Crash Recovery

- **Each master scatters segment replicas across entire cluster**
- **On crash:**
  - Coordinator partitions dead master's tablets
  - Partitions assigned to different recovery masters
  - Backups read disks in parallel
  - Shuffle log data from backups to recovery masters
  - Recovery masters replay log entries, incorporate objects into their logs
- **Fast recovery:**
  - 300 MB/s per recovery master
  - Recover 40 GB in 1.8 seconds (80 nodes, 160 SSDs)



Log-Structured Memory

- **Don't use malloc for memory management**
  - Wastes 50% of memory if workload changes



- **Instead, structure memory as a log**
  - Allocate by appending
  - Log cleaning to reclaim free space
- **Can run efficiently at 80-90% memory utilization**

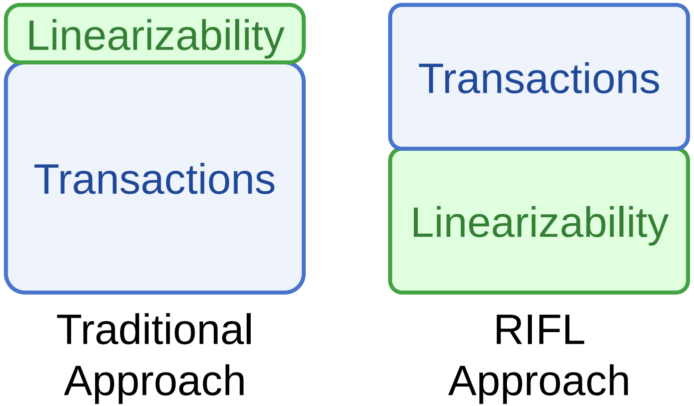
Other Work Related to RAMCloud

Raft consensus algorithm [USENIX 2014]

- **Paxos theoretically interesting, but poor basis for implementation:**
  - Hard to understand
  - Underspecified
  - Poor performance
- **Raft: new formulation of consensus**
  - Designed for understandability
  - Complete
  - Efficient
- **User study shows: Raft easier to understand than Paxos**

RIFL: Reusable Infrastructure for Linearizability [SOSP 2015]

- **New system layer: implements exactly-once semantics**
- **Used to implement transactions in RAMCloud**



Threats to Latency

- **Layering**
  - Great for software structuring
  - Bad for latency
  - E.g. RAMCloud threading structure costs 200-300ns/RPC
  - Virtualization is potential problem
- **Buffering**
  - Network buffers are the enemy of latency
  - TCP will fill them, no matter how large
  - Facebook measured 10's of ms RPC delay because of buffering
  - Need new networking architectures with no buffers
- **Substitute switching bandwidth for buffers**

Conclusion

- **Datacenter revolution only half over:**
  - Scale is here
  - Low latency is coming
- **Next steps:**
  - New networking architectures
  - New storage systems
- **Ultimate result:**
  - Exciting new applications
- **What could you do with:**
  - 1 million cores, accessing
  - 1 PB data, with
  - 5 µs access time??