Cool ideas from RAMCloud

Diego Ongaro
Stanford University

Joint work with Asaf Cidon, Ankita Kejriwal, John Ousterhout, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, et al.

DEVIEW 2013

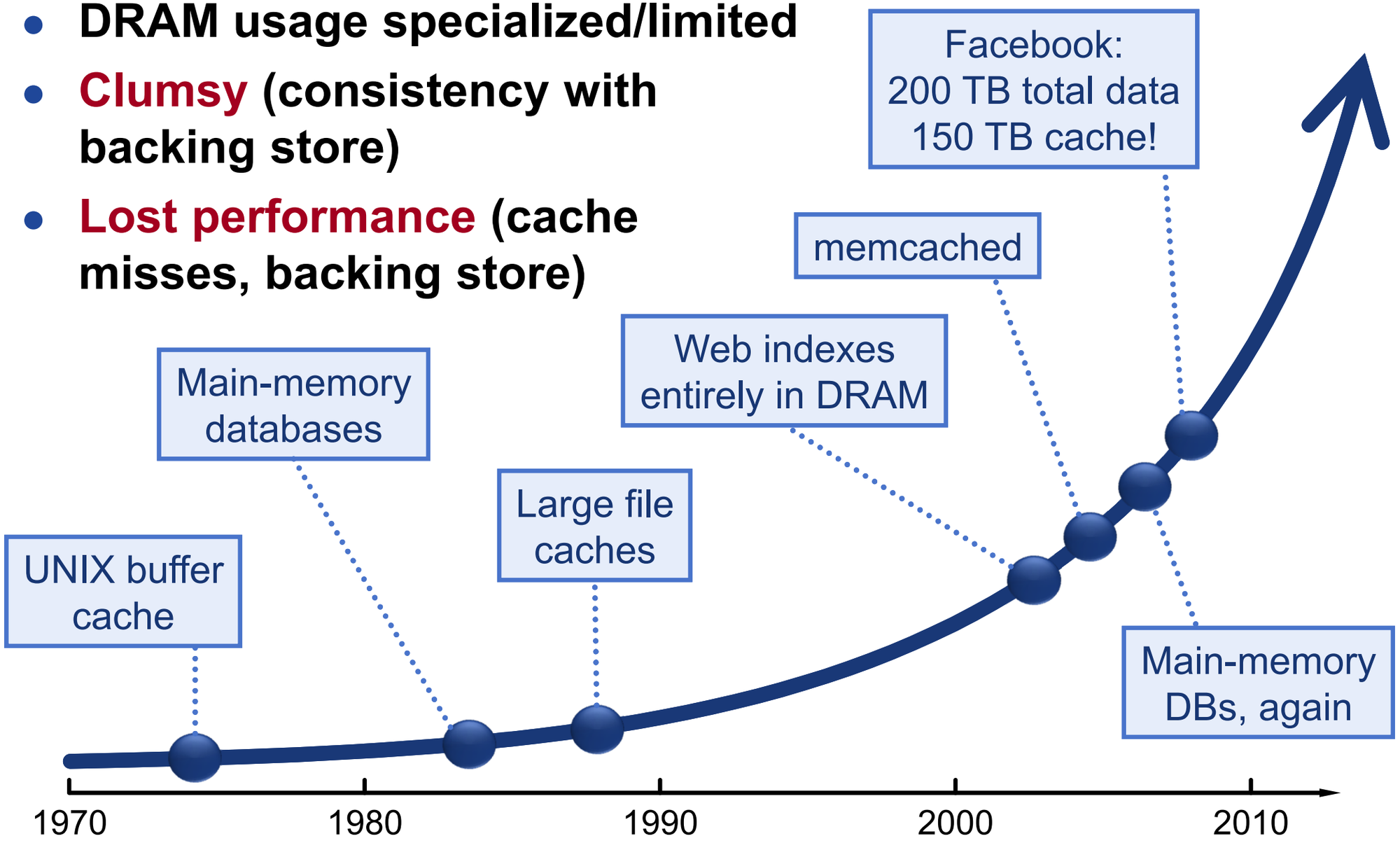
Introduction

- **RAMCloud: research project at Stanford University**
  - Led by Professor John Ousterhout, started in 2009
- **Building a real system (open source, C++)**
  - Looking for more users!

1. **RAMCloud basics**
2. **Fast crash recovery**
   - Recover failed servers in 1-2 seconds
3. **New consensus algorithm called Raft**
   - Used to store RAMCloud's cluster configuration
   - Designed for understandability

DRAM in Storage Systems

- **DRAM usage specialized/limited**
- **Clumsy (consistency with backing store)**
- **Lost performance (cache misses, backing store)**



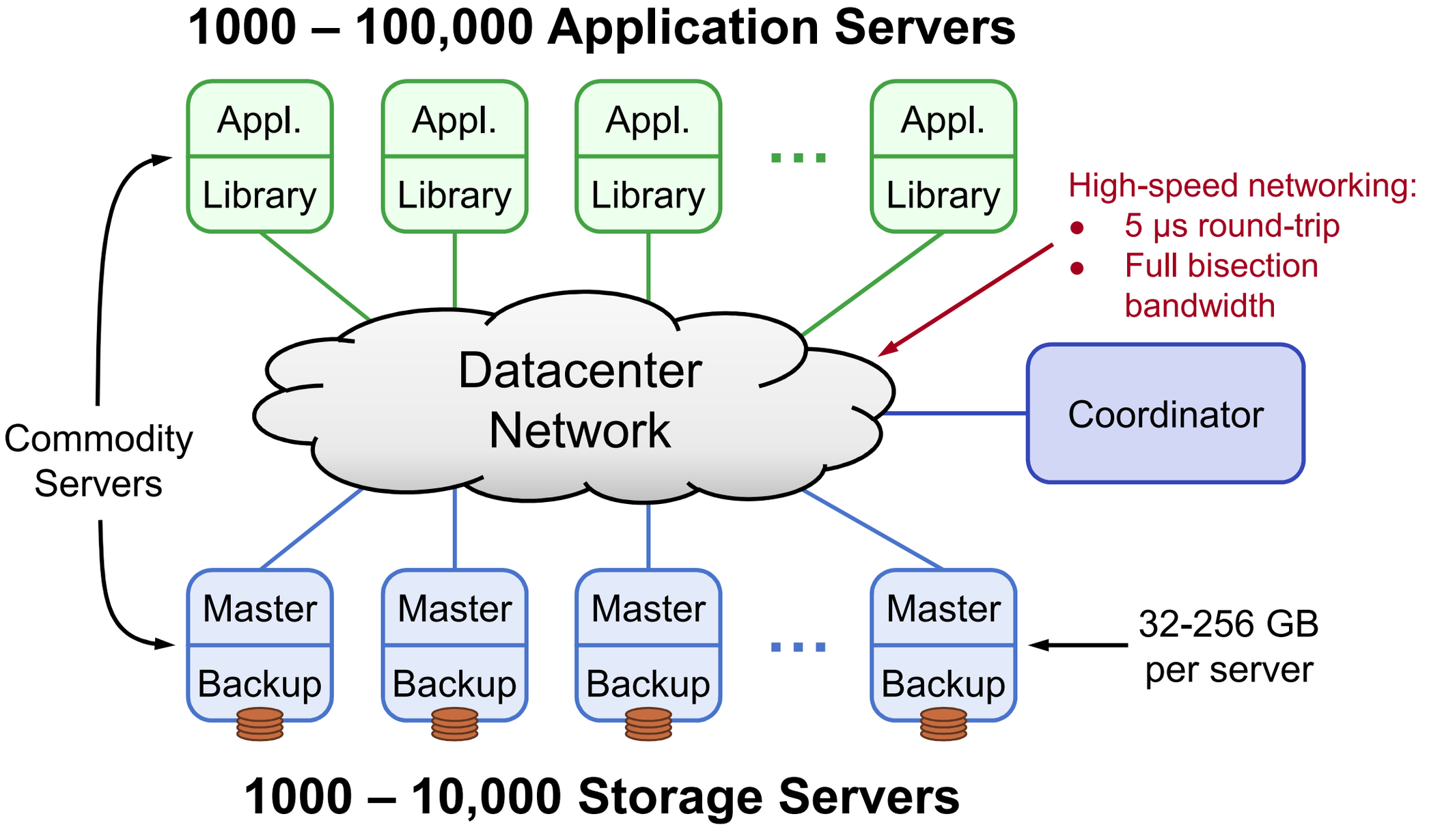
RAMCloud Overview

Harness full performance potential of large-scale DRAM storage:

- **General-purpose storage system**
- **All data always in DRAM (no cache misses)**
- **Durable and available**
- **Scale: 1000+ servers, 100+ TB**
- **Low latency: 5-10µs remote access**

Potential impact: enable new class of applications

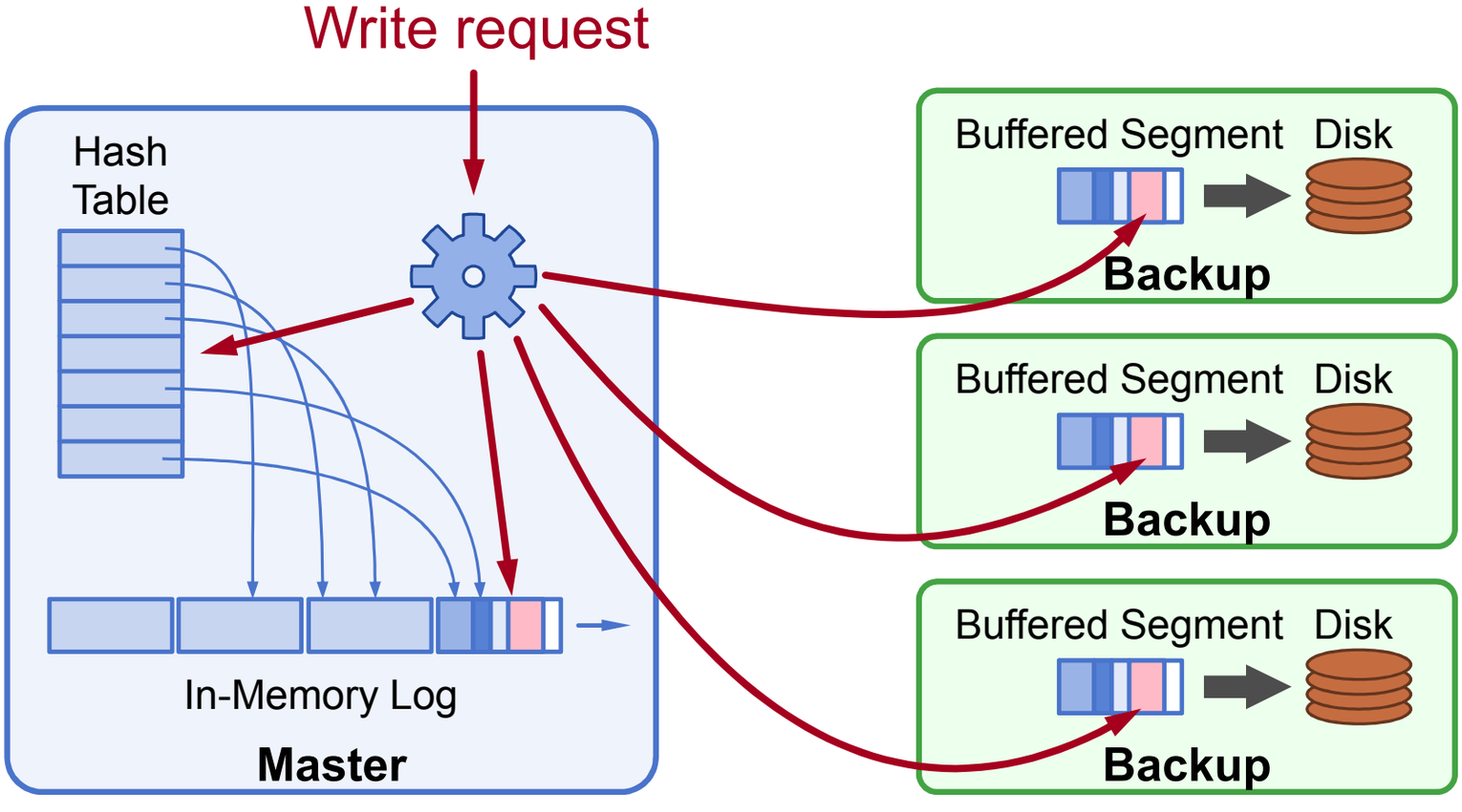
RAMCloud Architecture



Durability and Availability

- **Challenge: making volatile memory durable**
- **Keep replicas in DRAM of other servers?**
  - 3x system cost, energy
  - Still have to handle power failures
- **RAMCloud approach:**
  - 1 copy in DRAM
  - Backup copies on disk/flash: durability ~ free!

Buffered Logging



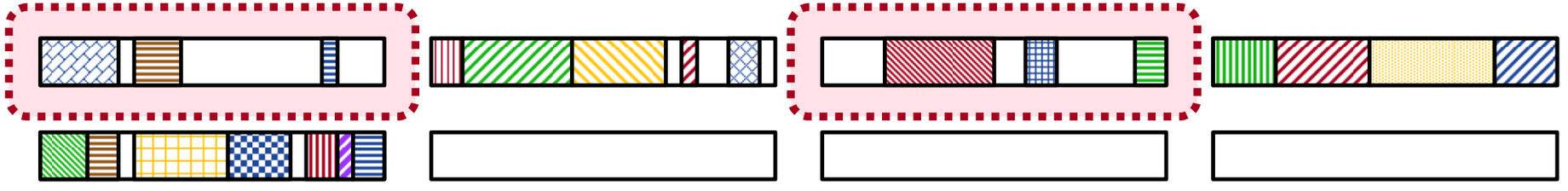
- **All data/changes appended to the log**
- **No disk I/O during write requests**
  - Backups need ~64 MB NVRAM (or battery) for power failures

Log-Structured Memory

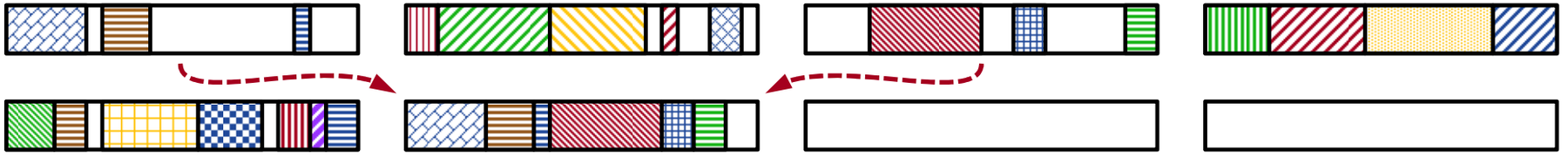
- **Log-structured: backup disk and master's memory**
  - Similar to Log-Structured File Systems (Rosenblum 1992)
- **Cleaning mechanism reclaims free space (next slide)**
- **Use memory efficiently (80-90% utilization)**
  - Good performance independent of workload + changes
- **Use disk bandwidth efficiently (8 MB writes)**
- **Manage both disk and DRAM with same mechanism**

Log Cleaning

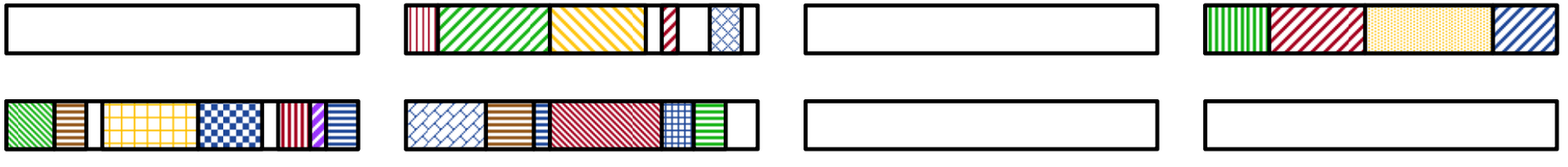
1. **Find** log segments with lots of free space



2. **Copy** live objects into new segment(s)



3. **Free** cleaned segments (reuse for new objects)



Cleaning is incremental and concurrent – no pauses!

Outline

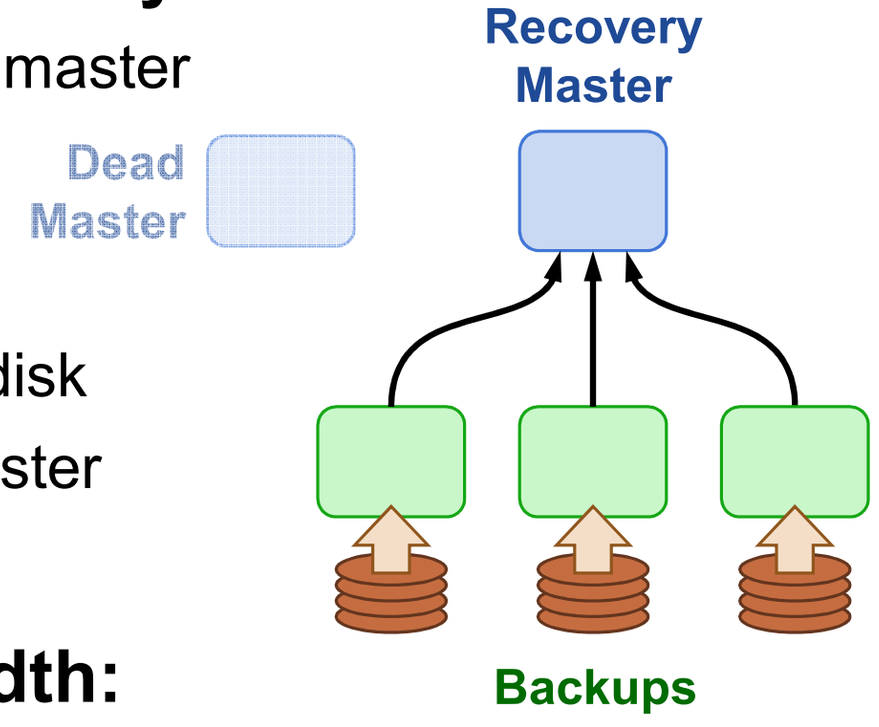
1. ~~RAMCloud basics~~
2. Fast crash recovery
3. New consensus algorithm called Raft

Crash Recovery Introduction

- **Applications depend on 5µs latency: must recover crashed servers quickly!**
- **Traditional approaches don't work**
  - Can't afford latency of paging in from disk
  - Replication in DRAM too expensive, doesn't solve power loss
- **Crash recovery:**
  - Need data loaded from backup disks back into DRAM
  - Must replay log to reconstruct hash table
  - Meanwhile, data is unavailable
  - Solution: fast crash recovery (1-2 seconds)
  - If fast enough, failures will not be noticed
- **Key to fast recovery: use system scale**

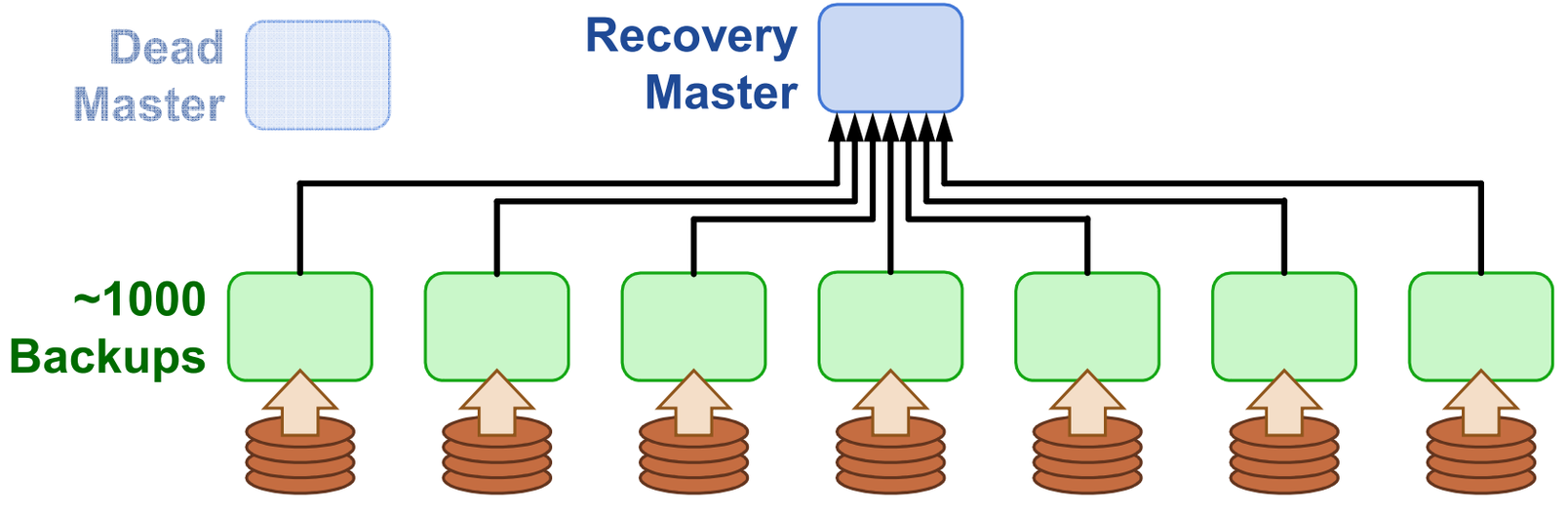
Recovery, First Try

- **Master chooses backups statically**
  - Each backup mirrors entire log for master
- **Crash recovery:**
  - Choose recovery master
  - Backups read log segments from disk
  - Transfer segments to recovery master
  - Recovery master replays log
- **First bottleneck: disk bandwidth:**
  - 64 GB / 3 backups / 100 MB/sec/disk
    ≈ 210 seconds
- **Solution: more disks (and backups)**

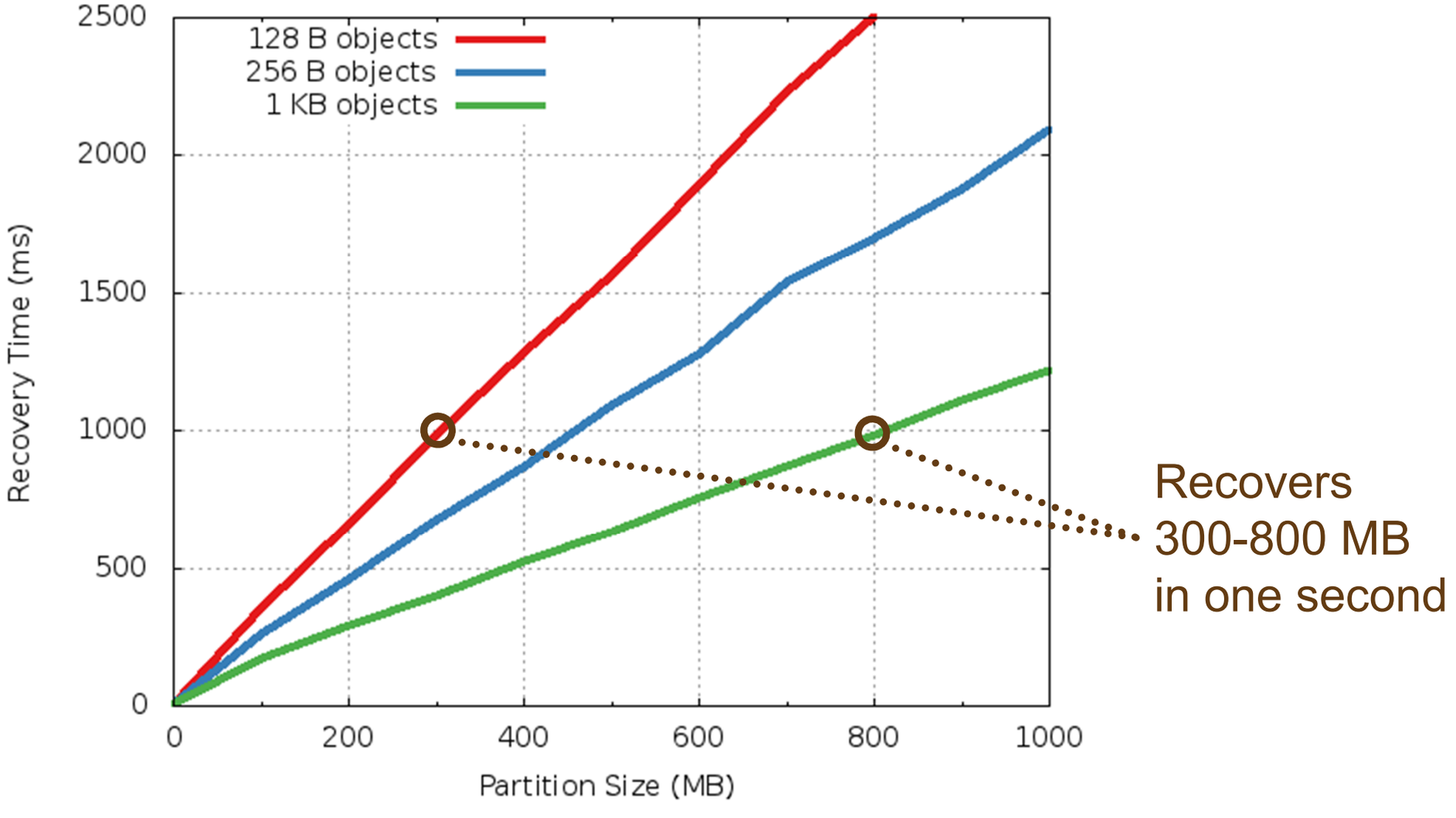


Recovery, Second Try

- **Scatter logs:**
  - Each log divided into 8MB segments
  - Master chooses different backups for each segment (randomly)
  - Segments scattered across all servers in the cluster
- **Crash recovery:**
  - All backups read from disk in parallel (100-500 MB each)
  - Transmit data over network to recovery master



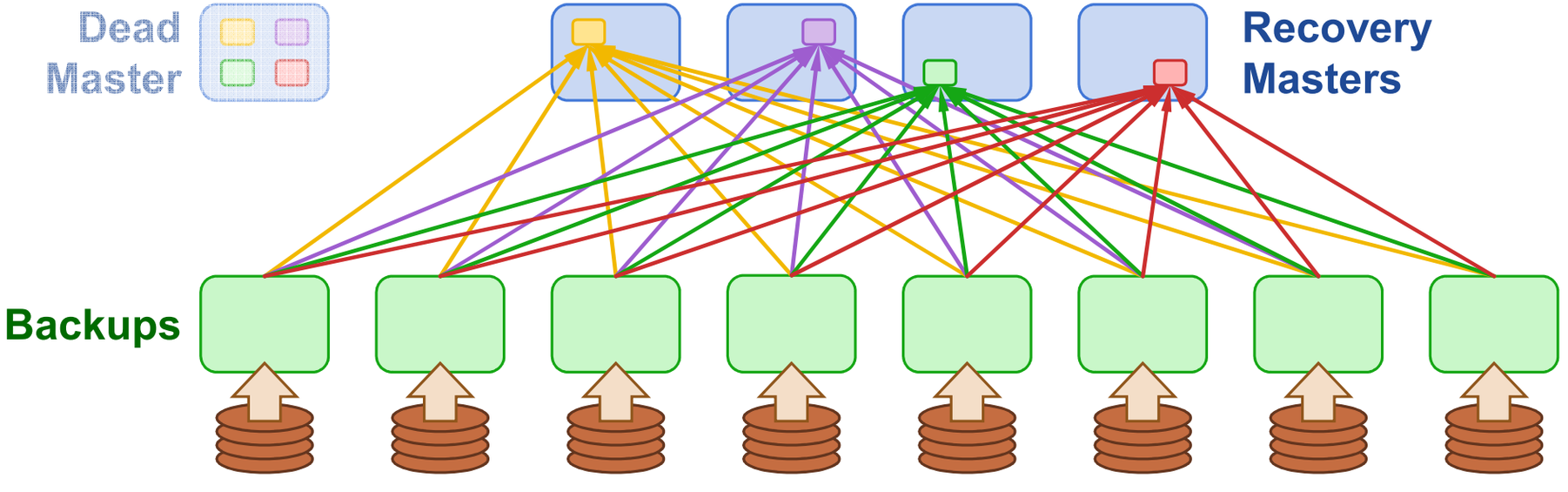
Single Recovery Master



- **New bottlenecks: recovery master's NIC and CPU**

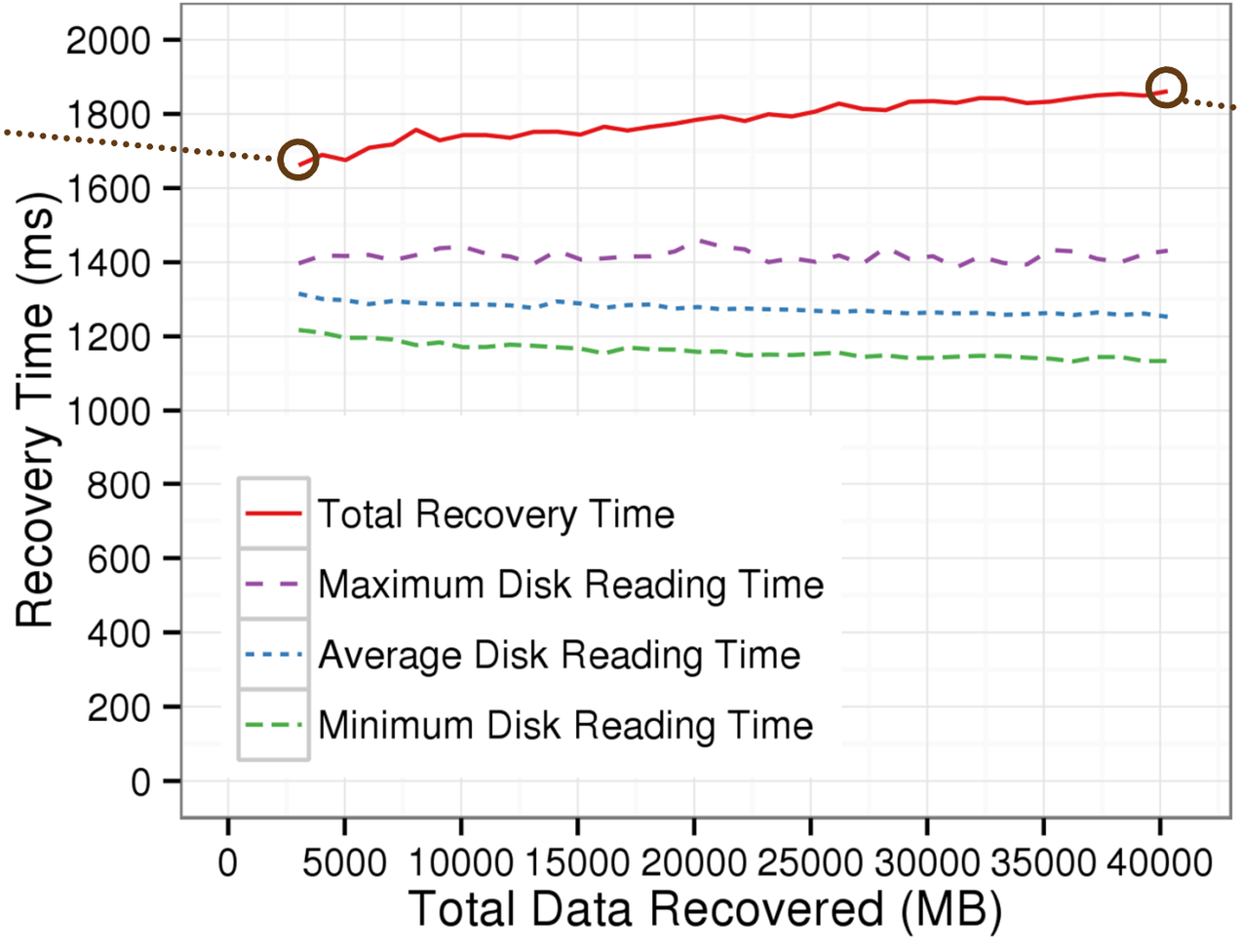
Recovery, Third Try

- Divide dead master's data into partitions
  - Recover each partition on a separate recovery master
  - Partitions based on key ranges, *not log segment*
  - Each backup divides its log data among recovery masters
  - Highly parallel and pipelined



Scalability

6 masters
12 SSDs
3 GB



80 masters
160 SSDs
40 GB

- **Nearly flat (needs to reach 120-200 masters)**
- **Expect faster: out of memory bandwidth (old servers)**

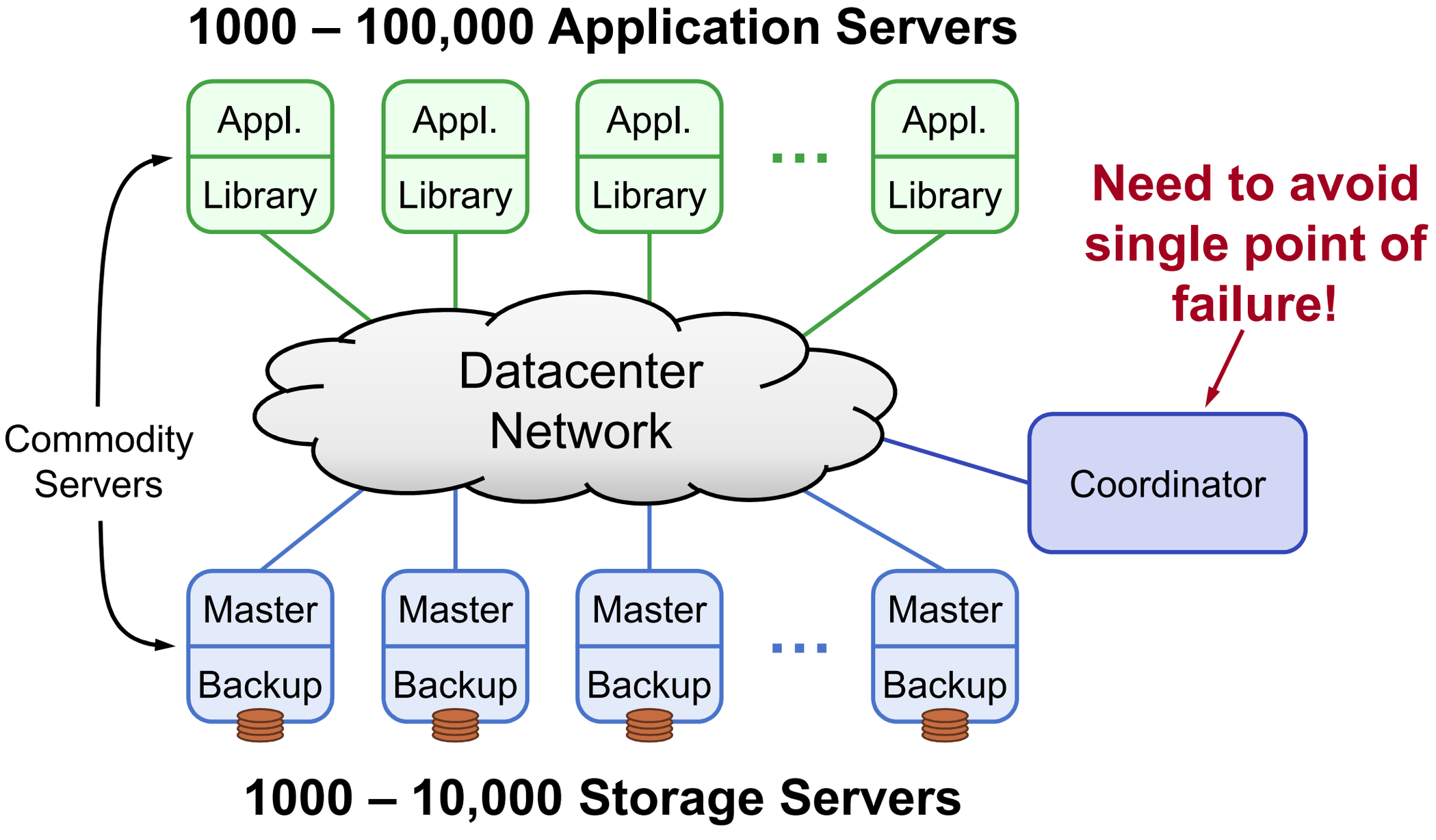
Crash Recovery Conclusion

- **Fast: under 2 seconds for memory sizes up to 40GB**
- **Nearly "high availability"**
- **Much cheaper than DRAM replication (and it handles power outages)**
- **Leverages scale:**
  - Uses every disk, every NIC, every CPU, …
  - Recover larger memory sizes with larger clusters
- **Leverages low latency:**
  - Fast failure detection and coordination

Outline

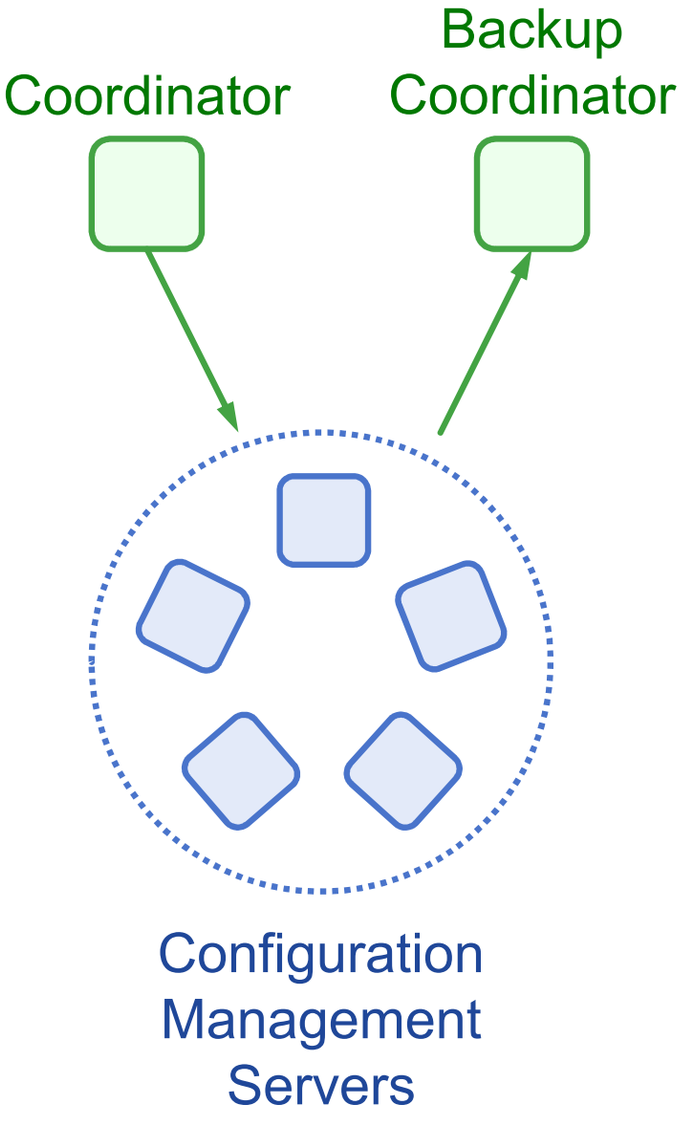
1. ~~RAMCloud basics~~
2. ~~Fast crash recovery~~
3. New consensus algorithm called Raft

RAMCloud Architecture



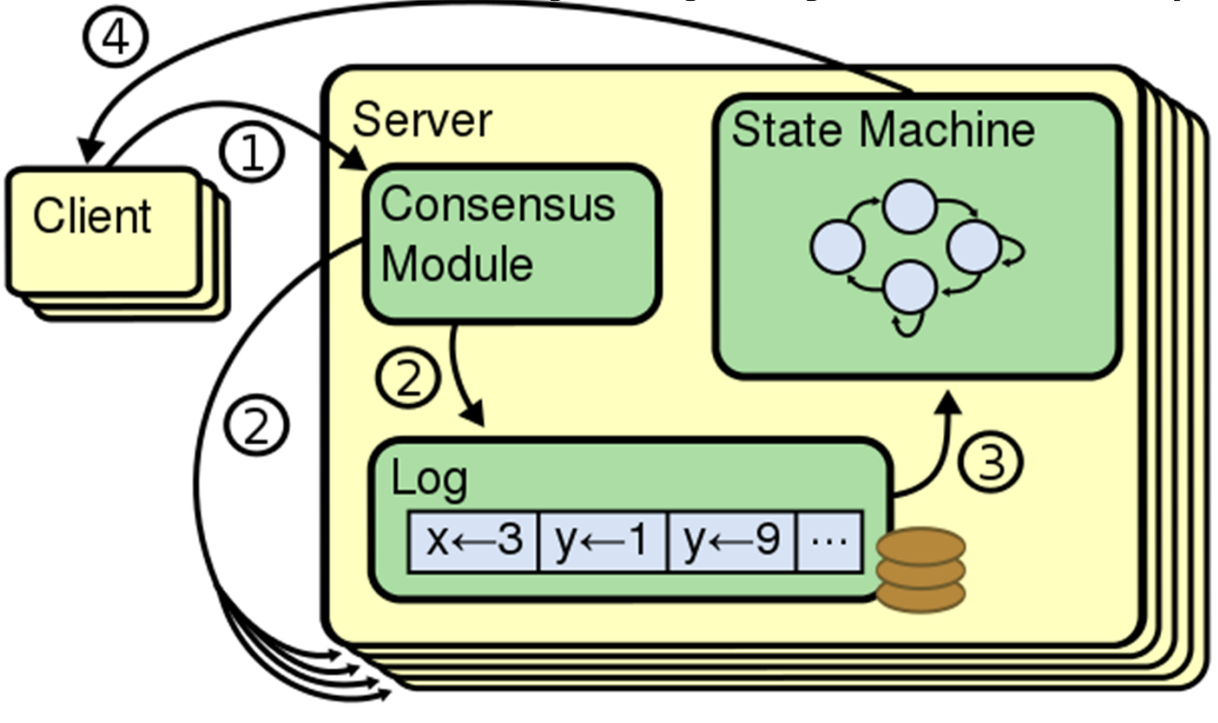
Configuration Management

- **Keep single coordinator server**
- **Rely on external system to store top-level configuration:**
  - Cluster membership
  - Data placement
- **Typically consensus-based**
- **Examples:**
  - Chubby (Google)
  - ZooKeeper (Yahoo/Apache)
- **Unhappy with ZooKeeper, so decided to build our own**



Replicated State Machines

- Approach to make any (det.) service fault-tolerant
- Each server's state machine processes the same sequence of requests from its log
- Consensus algo.: order requests into replicated log
- Small number of servers (typically 5)
  - Service available with any majority of servers (3)



New Consensus Algorithm: Raft

- **Initial plan: use Paxos**
- **Paxos is "industry standard" consensus algorithm, but:**
  - Very hard to understand
  - Not a good starting point for real implementations
- **Our new algorithm: Raft**
  - Primary design goal: understandability
  - Also must be practical and complete

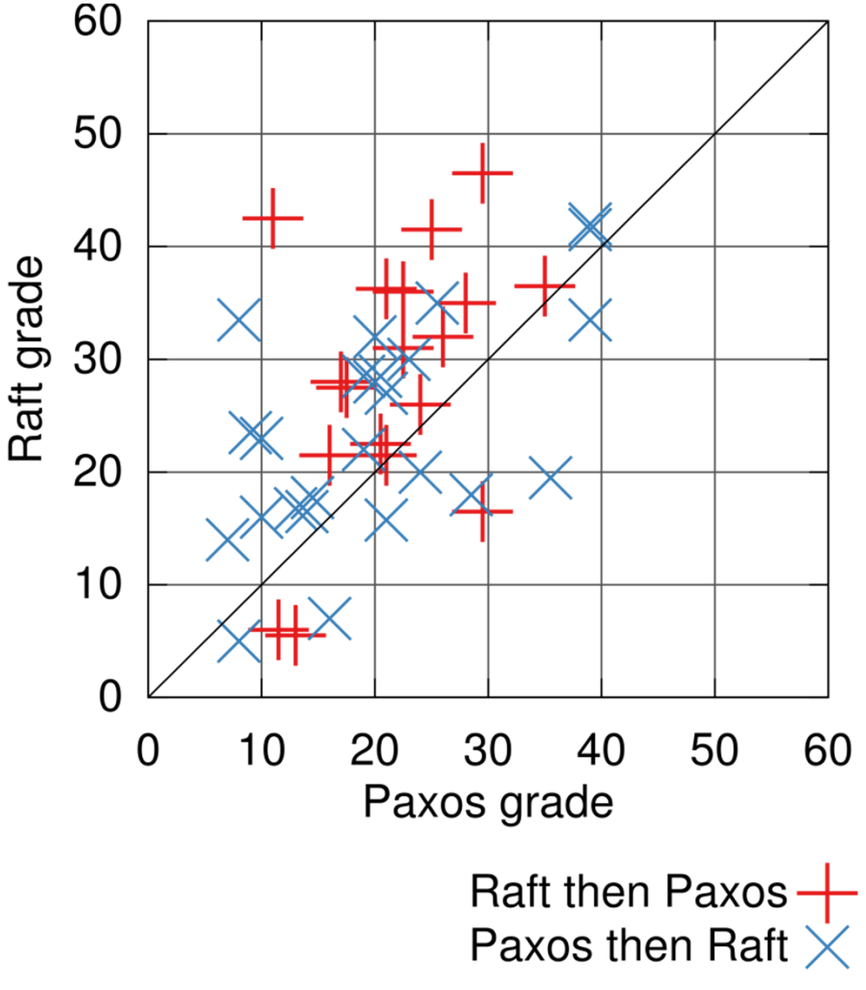
Raft Overview

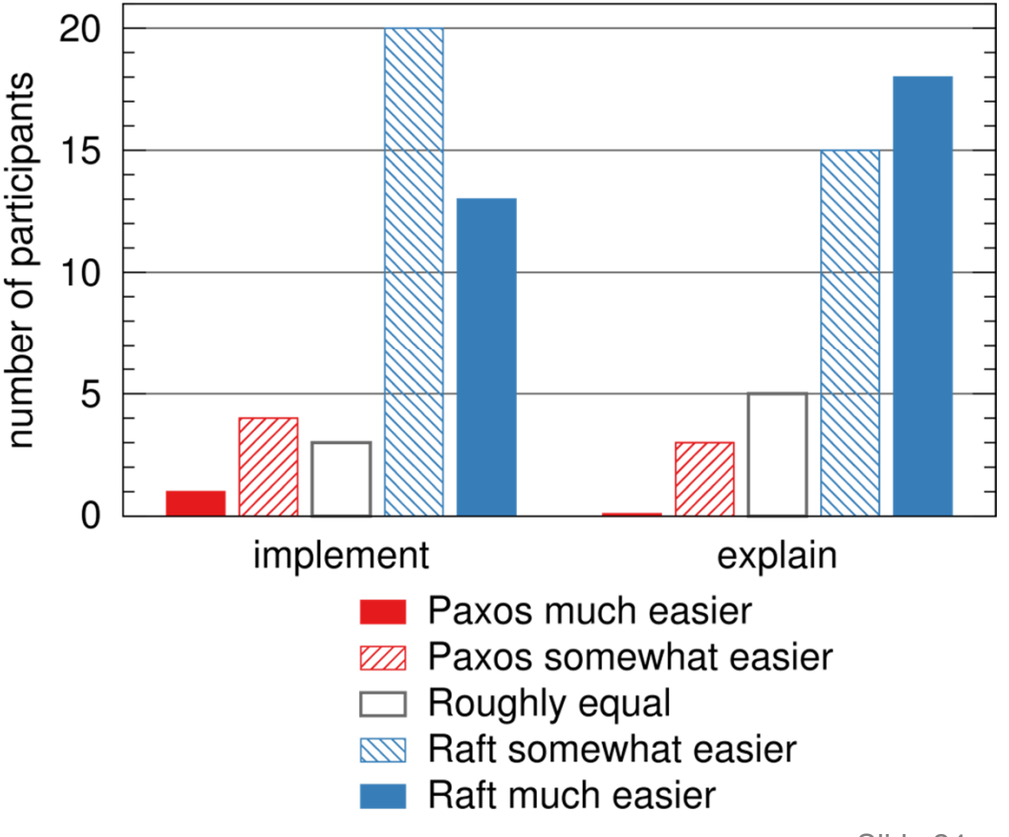
Designed from the start to manage a replicated log:

1. **Elect a leader:**
   - A single server at a time creates new entries in the replicated log
   - Elect a new leader when the leader fails
2. **Replication:**
   - The leader makes other servers' logs match its own, and
   - Notifies other servers when it's safe to apply operations
3. **Safety:**
   - Defines when operations are safe to apply
   - Ensures that only servers with up-to-date logs can be elected leader

User Study

- Taught students both Paxos and Raft
- One-hour videos, one-hour quizzes, short survey
- Raft significantly easier for students to understand





Raft Conclusion

- **Raft consensus algorithm**
  - Equivalent to Paxos, but designed to be easy to understand
  - Decomposes the problem well
  - Built to manage a replicated log, uses strong leader
  - Watch our videos!
- **LogCabin: configuration service built on Raft**
  - Open source, C++
  - Similar to ZooKeeper, not as many features yet
- **At least 25 other implementations on GitHub**
  - go-raft (Go) quite popular, used by etcd in CoreOS

Overall Conclusion

- **RAMCloud goals:**
  - Harness full performance potential of DRAM-based storage
  - Enable new applications: intensive manipulation of big data
- **Achieved low latency (at small scale)**
  - Not yet at large scale (but scalability encouraging)
- **Fast crash recovery:**
  - Recovered 40GB in < 2 seconds; more with larger clusters
  - Durable + available DRAM storage for the cost of volatile cache
- **Raft consensus algorithm:**
  - Making consensus easier to learn and implement

Questions?

- **Speaker: Diego Ongaro**
  - @ongardie on Twitter
  - ongaro@cs.stanford.edu
- **Much more at http://ramcloud.stanford.edu**
  - **Papers:** Case for RAMCloud, Fast Crash Recovery, Log Structured Memory, Raft, etc.
  - Check out the RAMCloud and LogCabin **code**
  - Ask **questions** on ramcloud-dev, raft-dev mailing lists
  - Watch Raft and Paxos **videos** to learn consensus!
- Joint work with Asaf Cidon, Ankita Kejriwal, John Ousterhout, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, et al.