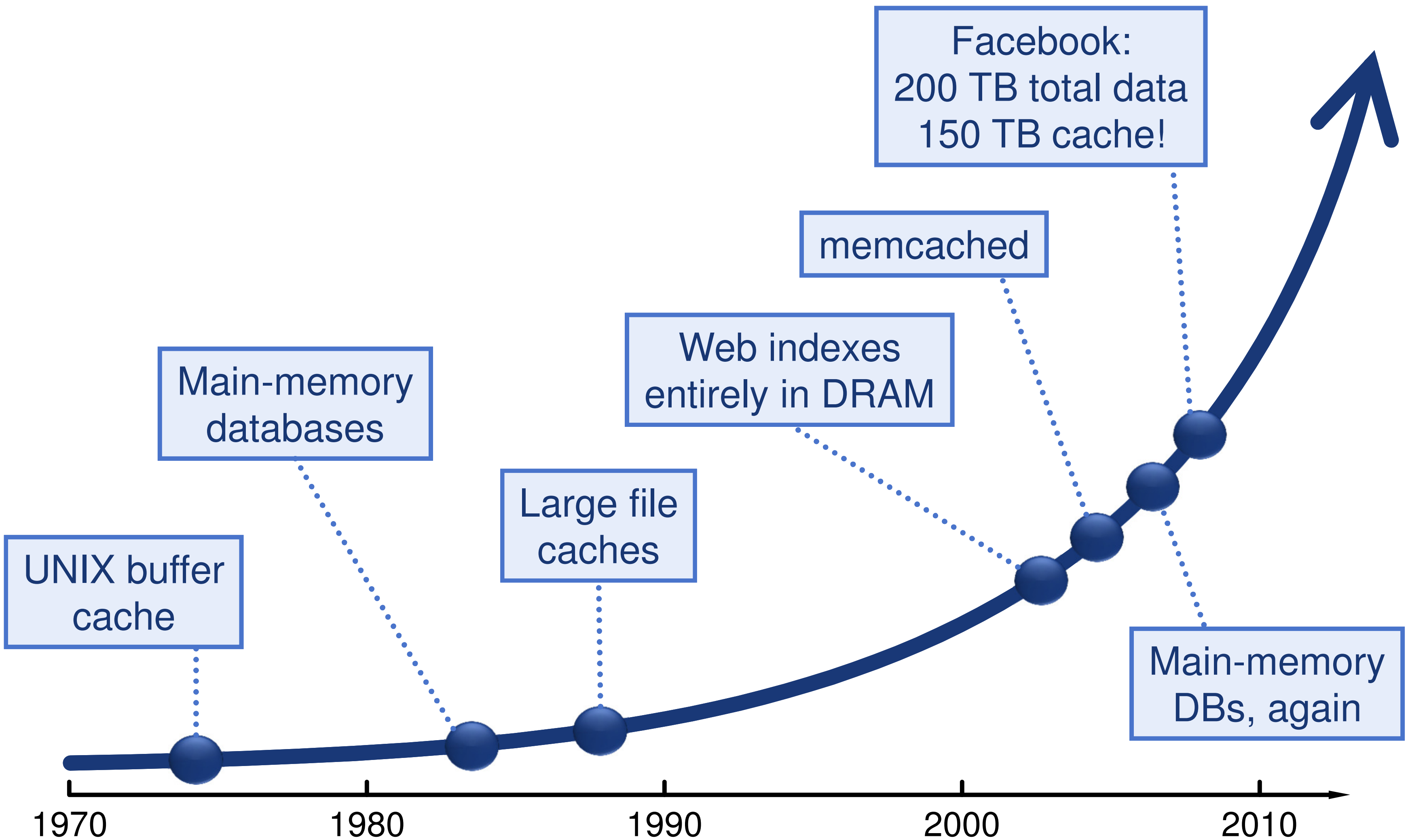
RAMCloud: Scalable High-Performance Storage Entirely in DRAM

John Ousterhout

Stanford University

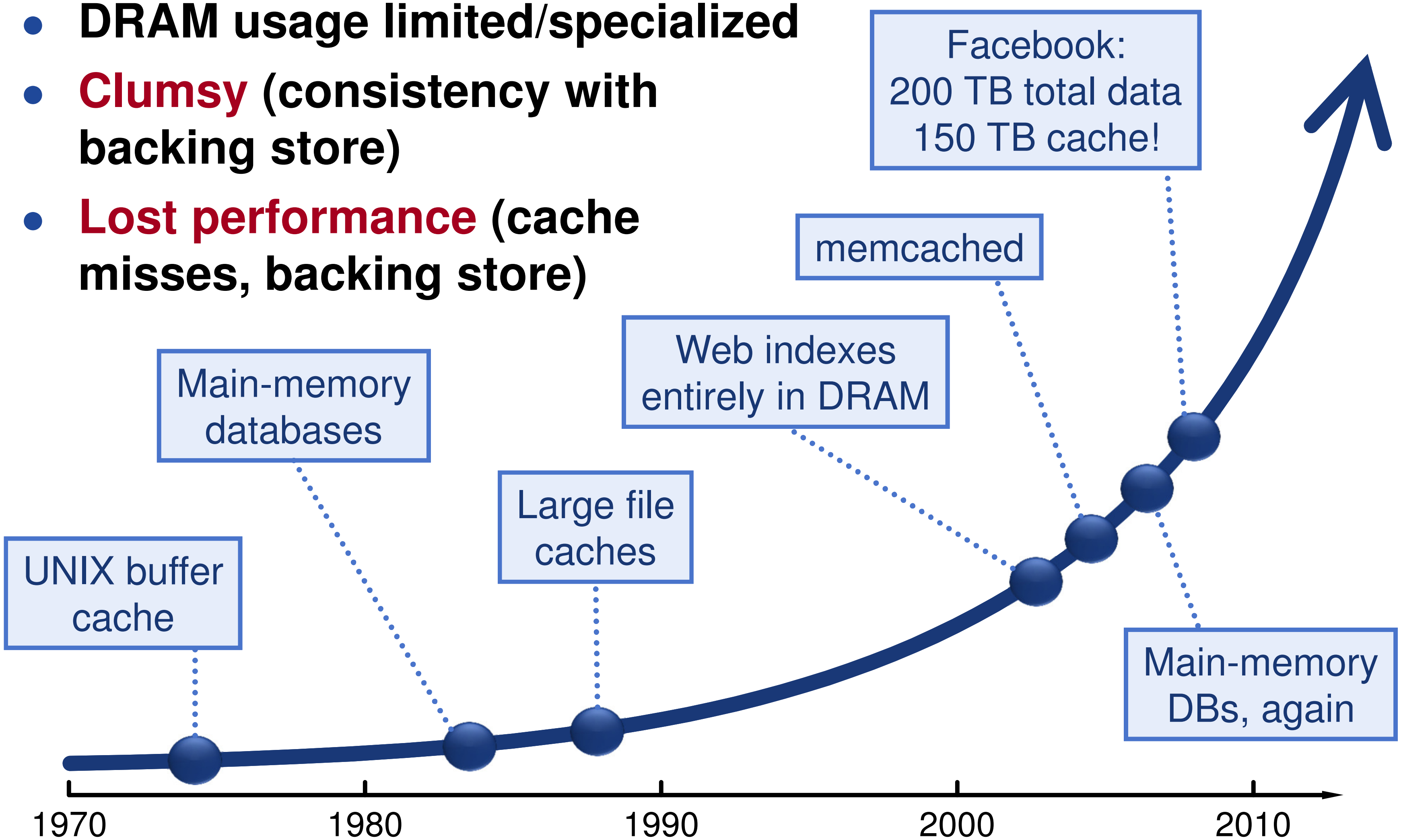
(with Nandu Jayakumar, Diego Ongaro, Mendel Rosenblum, Stephen Rumble, and Ryan Stutsman)

DRAM in Storage Systems



DRAM in Storage Systems

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



RAMCloud

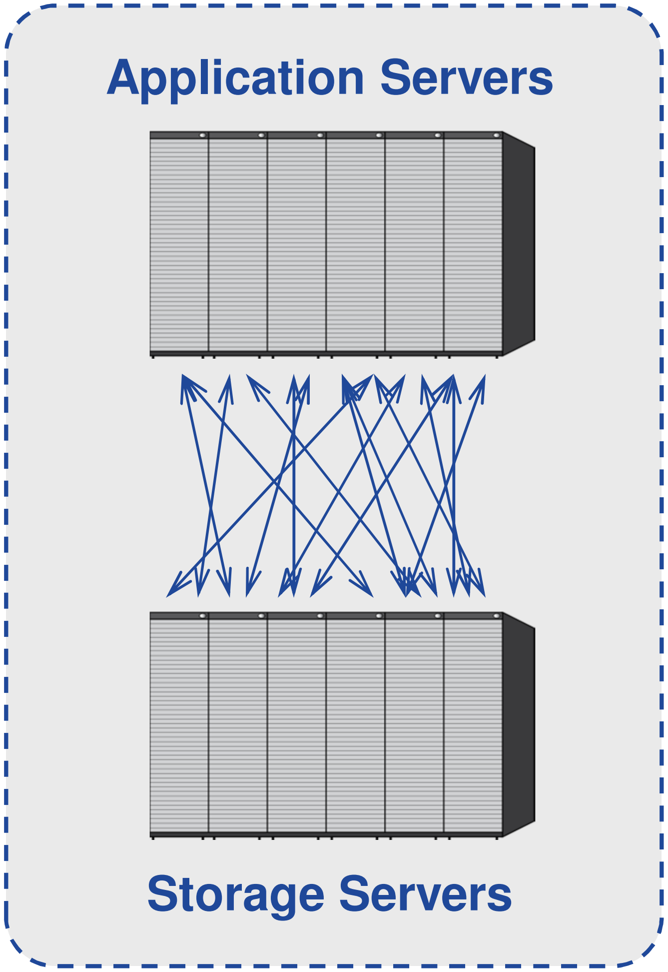
Harness full performance potential of large-scale DRAM storage:

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

Potential impact: enable new class of applications

RAMCloud Overview

- **Storage for datacenters**
- **1000-10000 commodity servers**
- **32-64 GB DRAM/server**
- **All data always in RAM**
- **Durable and available**
- **Performance goals:**
  - High throughput: 1M ops/sec/server
  - Low-latency access: 5-10µs RPC



Datacenter

Example Configurations

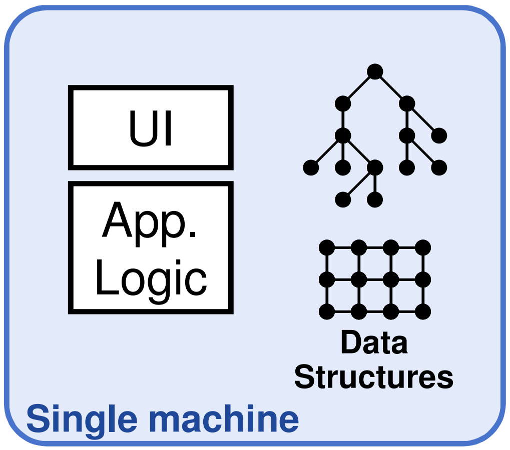
	Today	5-10 years
# servers	2000	4000
GB/server	24GB	256GB
Total capacity	48TB	1PB
Total server cost	$3.1M	$6M
$/GB	$65	$6

For $100-200K today:

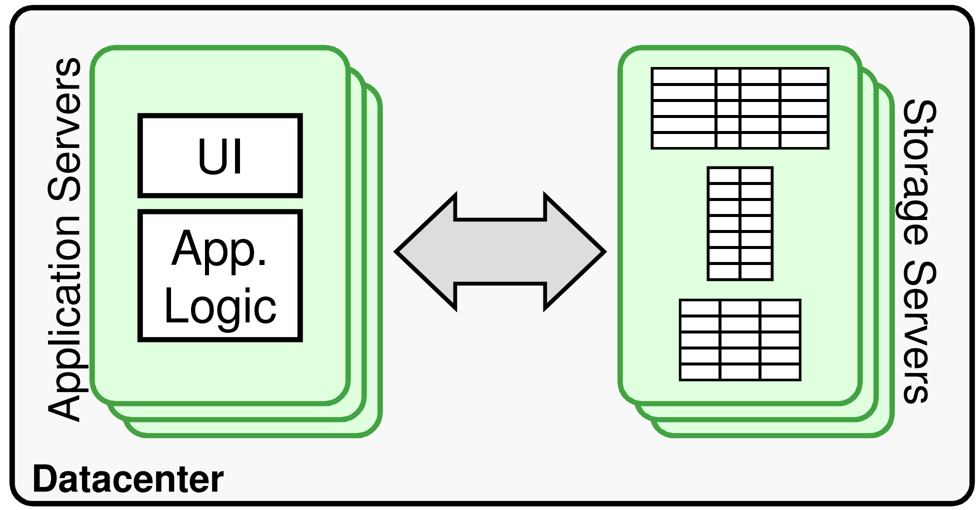
- One year of Amazon customer orders
- One year of United flight reservations

Why Does Latency Matter?

Traditional Application



Web Application

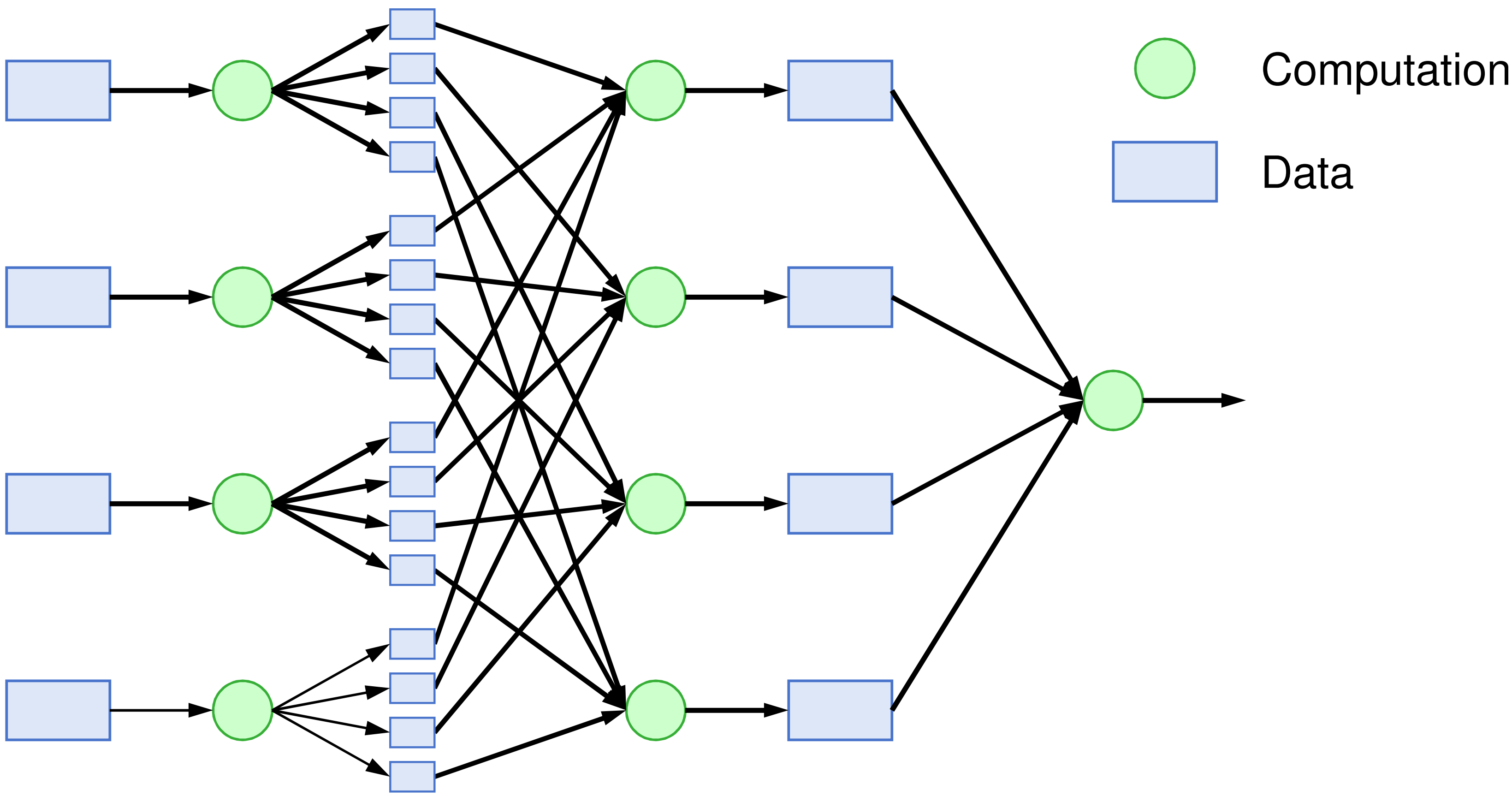


<< 1µs latency

0.5-10ms latency

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

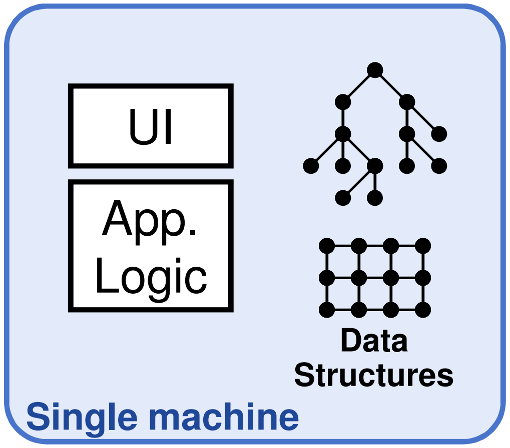
MapReduce



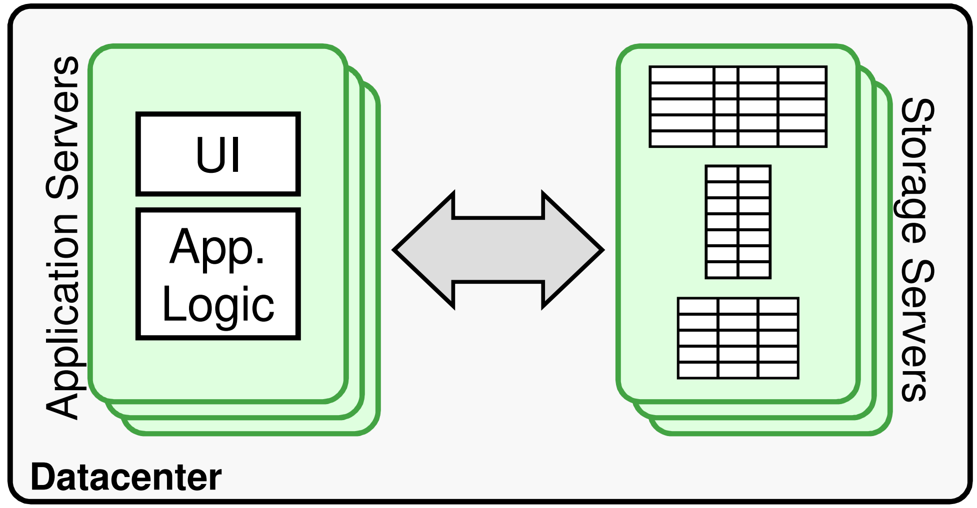
- ✓ **Sequential data access → high data access rate**
- ✗ **Not all applications fit this model**
- ✗ **Offline**

Goal: Scale and Latency

Traditional Application



Web Application

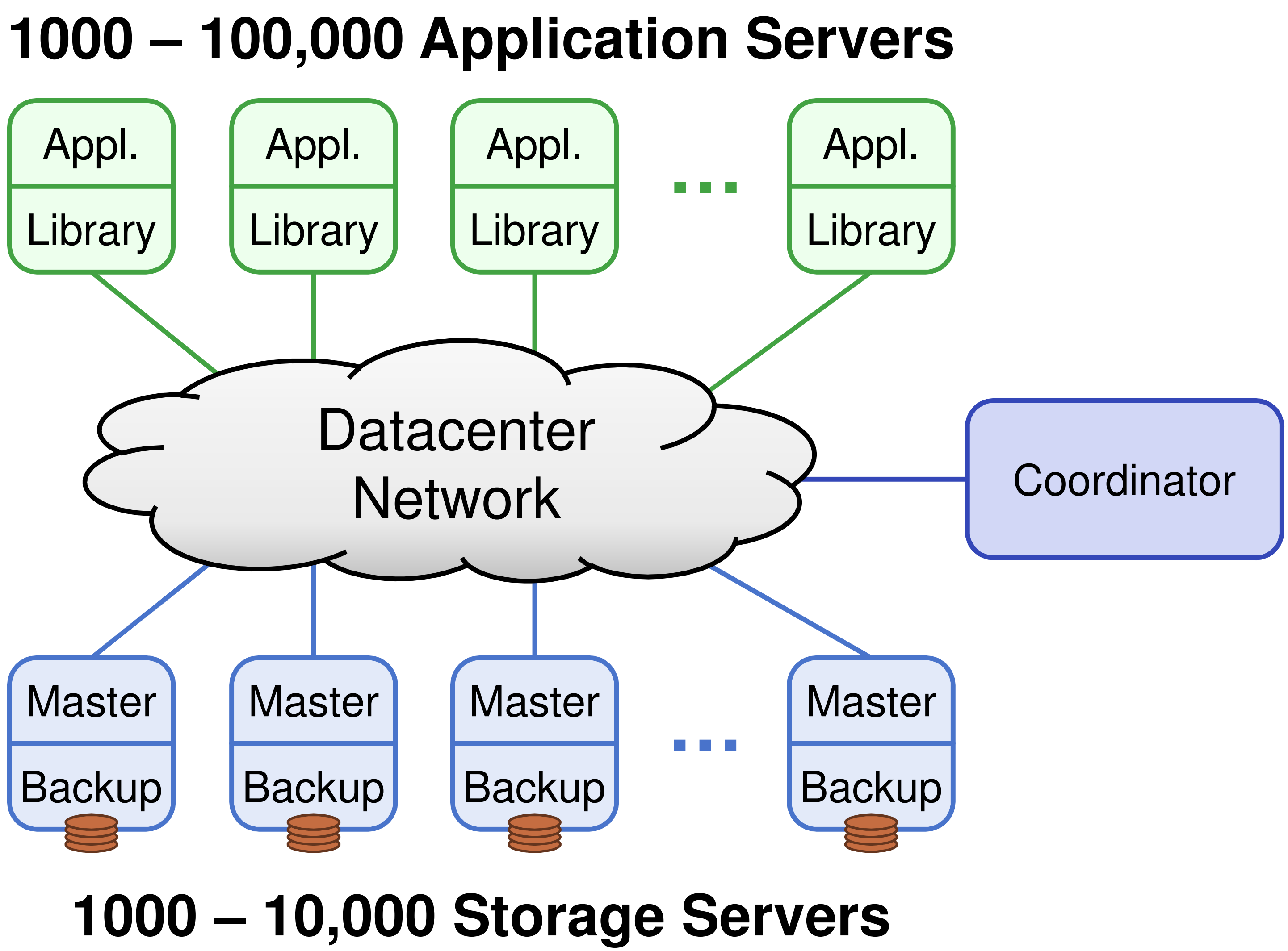


<< 1µs latency

~~0.5-10ms~~ latency
5-10µs

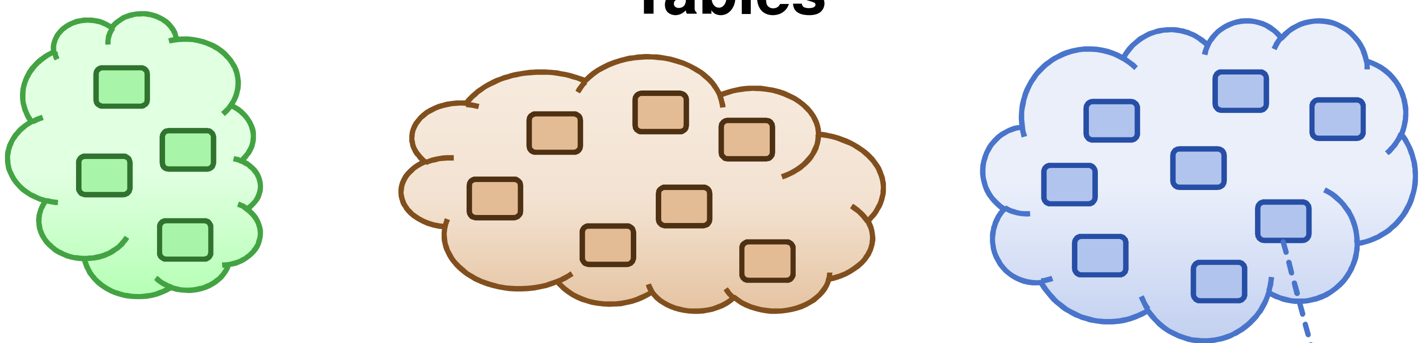
- **Enable new class of applications:**
  - Crowd-level collaboration
  - Large-scale graph algorithms
  - Real-time information-intensive applications

RAMCloud Architecture



Data Model

Tables



Object

Identifier (64b)
Version (64b)
Blob (≤1MB)

```
create(tableId, blob)
    => objectId, version

read(tableId, objectId)
    => blob, version

write(tableId, objectId, blob)
    => version

cwrite(tableId, objectId, blob, version)
    => version

delete(tableId, objectId)
```

(Only overwrite if version matches)

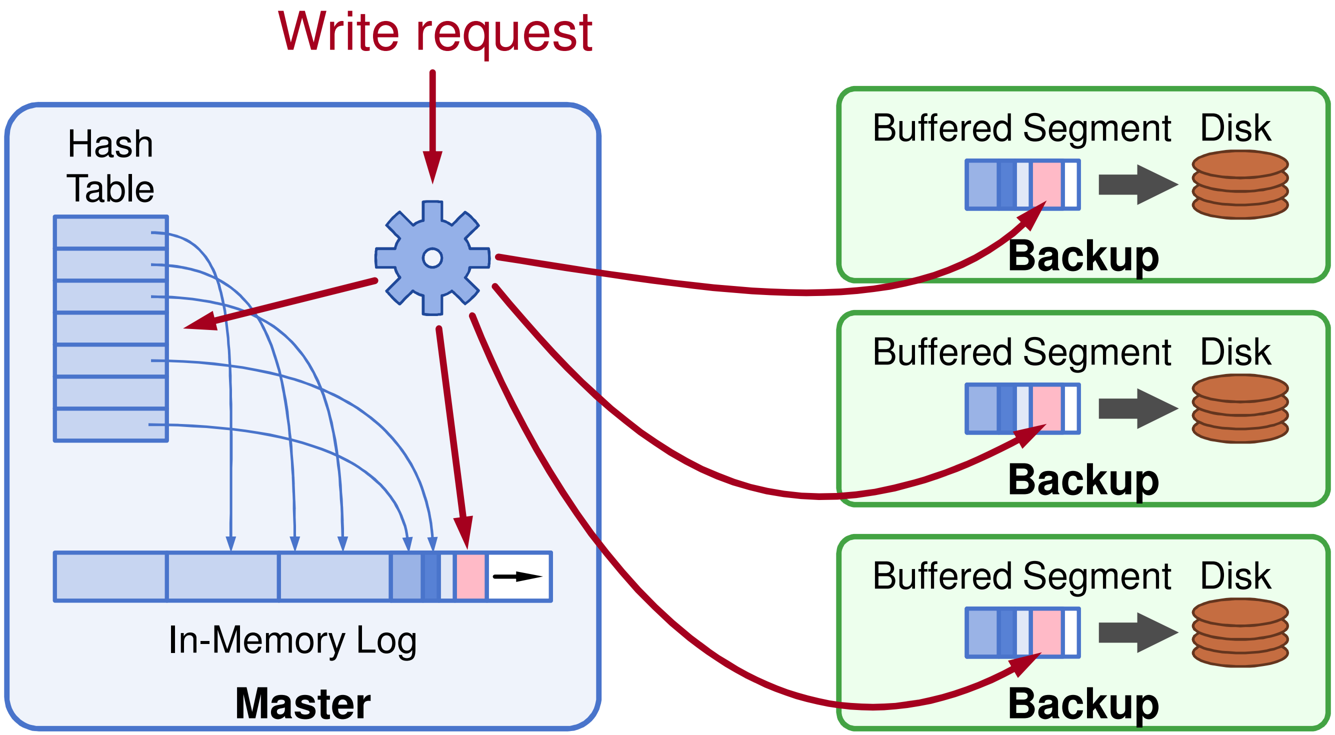
Richer model in the future:

- Indexes?
- Transactions?
- Graphs?

Durability and Availability

- **Goals:**
  - No impact on performance
  - Minimum cost, energy
- **Keep replicas in DRAM of other servers?**
  - 3x system cost, energy
  - Still have to handle power failures
  - Replicas unnecessary for performance
- **RAMCloud approach:**
  - 1 copy in DRAM
  - Backup copies on disk/flash: durability ~ free!
- **Issues to resolve:**
  - Synchronous disk I/O's during writes??
  - Data unavailable after crashes??

Buffered Logging



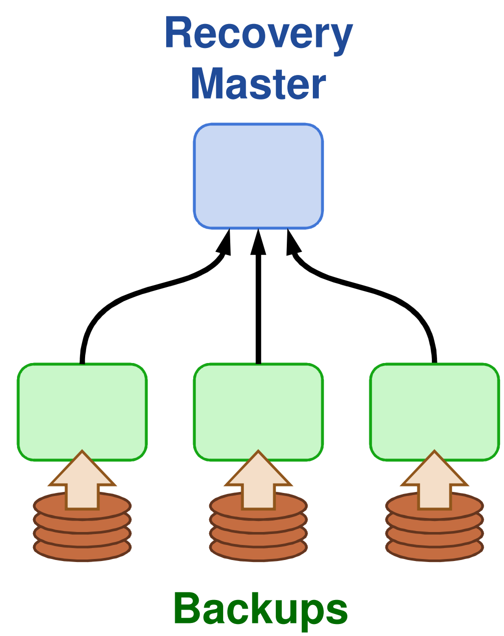
- **No disk I/O during write requests**
- **Master's memory also log-structured**
- **Log cleaning ~ generational garbage collection**

Crash Recovery

- **Power failures: backups must guarantee durability of buffered data:**
  - DIMMs with built-in flash backup?
  - Per-server battery backups?
  - Caches on enterprise disk controllers?
- **Server crashes:**
  - Must replay log to reconstruct data
  - 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**

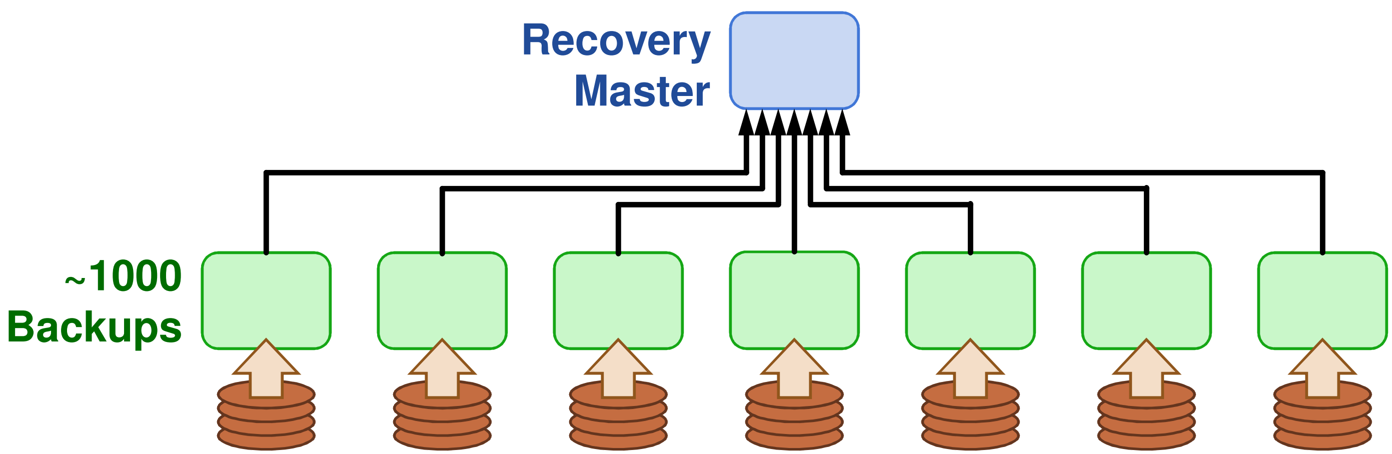
Recovery, First Try

- **Master chooses backups statically**
  - Each backup stores entire log for master
- **Crash recovery:**
  - Choose recovery master
  - Backups read log info from disk
  - Transfer logs 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
  - Transmit data over network to recovery master

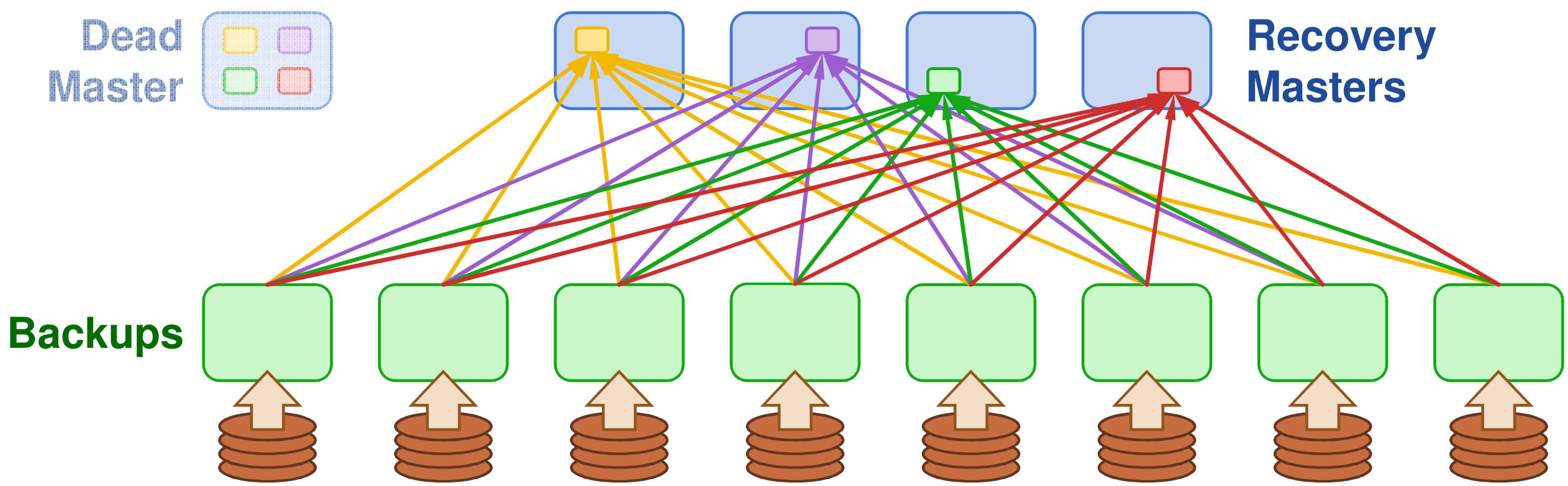


Scattered Logs, cont'd

- **Disk no longer a bottleneck:**
  - 64 GB / 8 MB/segment / 1000 backups ≈ 8 segments/backup
  - 100ms/segment to read from disk
  - 0.8 second to read all segments in parallel
- **Second bottleneck: NIC on recovery master**
  - 64 GB / 10 Gbits/second ≈ 60 seconds
  - Recovery master CPU is also a bottleneck
- **Solution: more recovery masters**
  - Spread work over 100 recovery masters
  - 64 GB / 10 Gbits/second / 100 masters ≈ 0.6 second

Recovery, Third Try

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



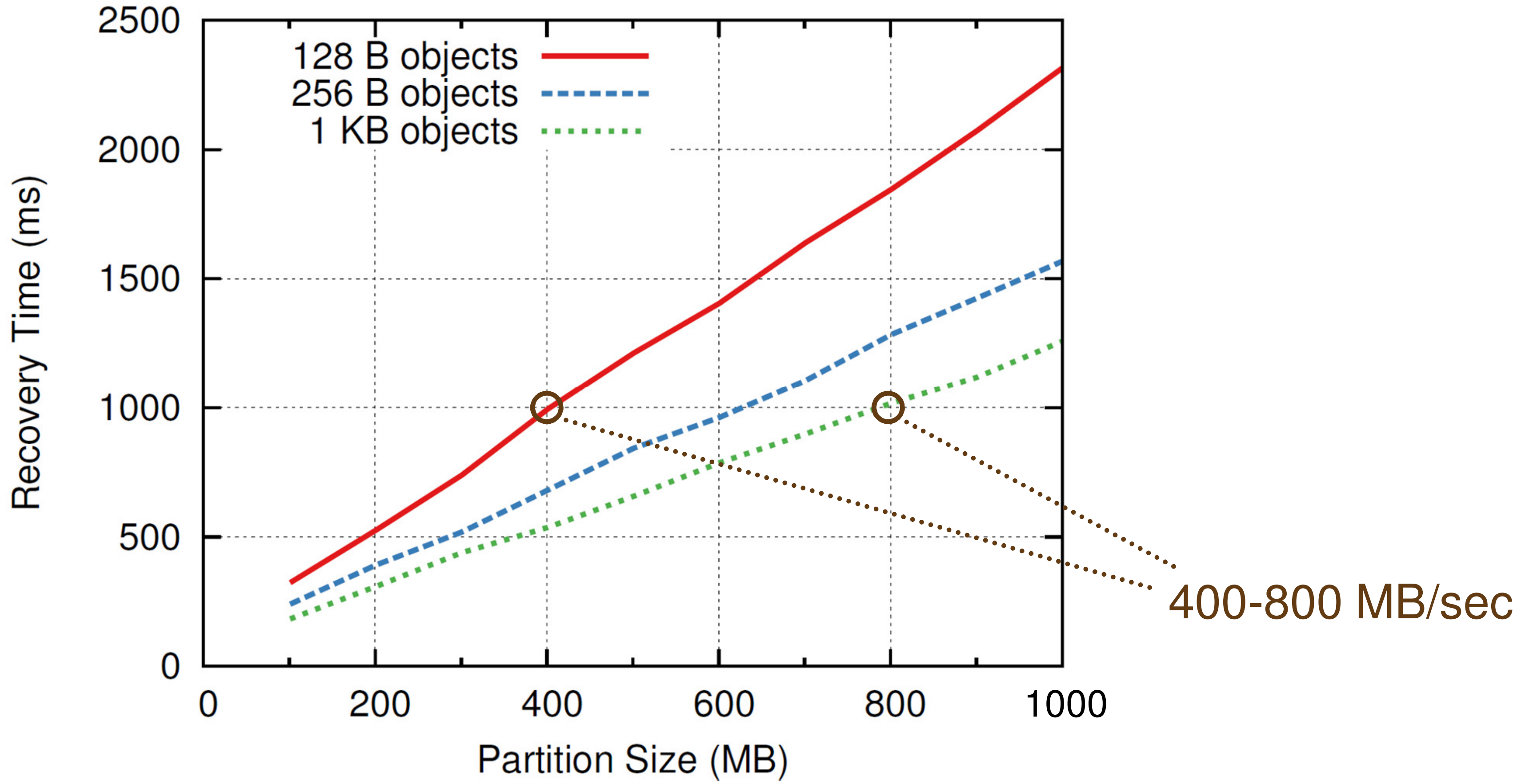
Other Research Issues

- **Fast communication (RPC)**
  - New datacenter network protocol?
- **Data model**
- **Concurrency, consistency, transactions**
- **Data distribution, scaling**
- **Multi-tenancy**
- **Client-server functional distribution**
- **Node architecture**

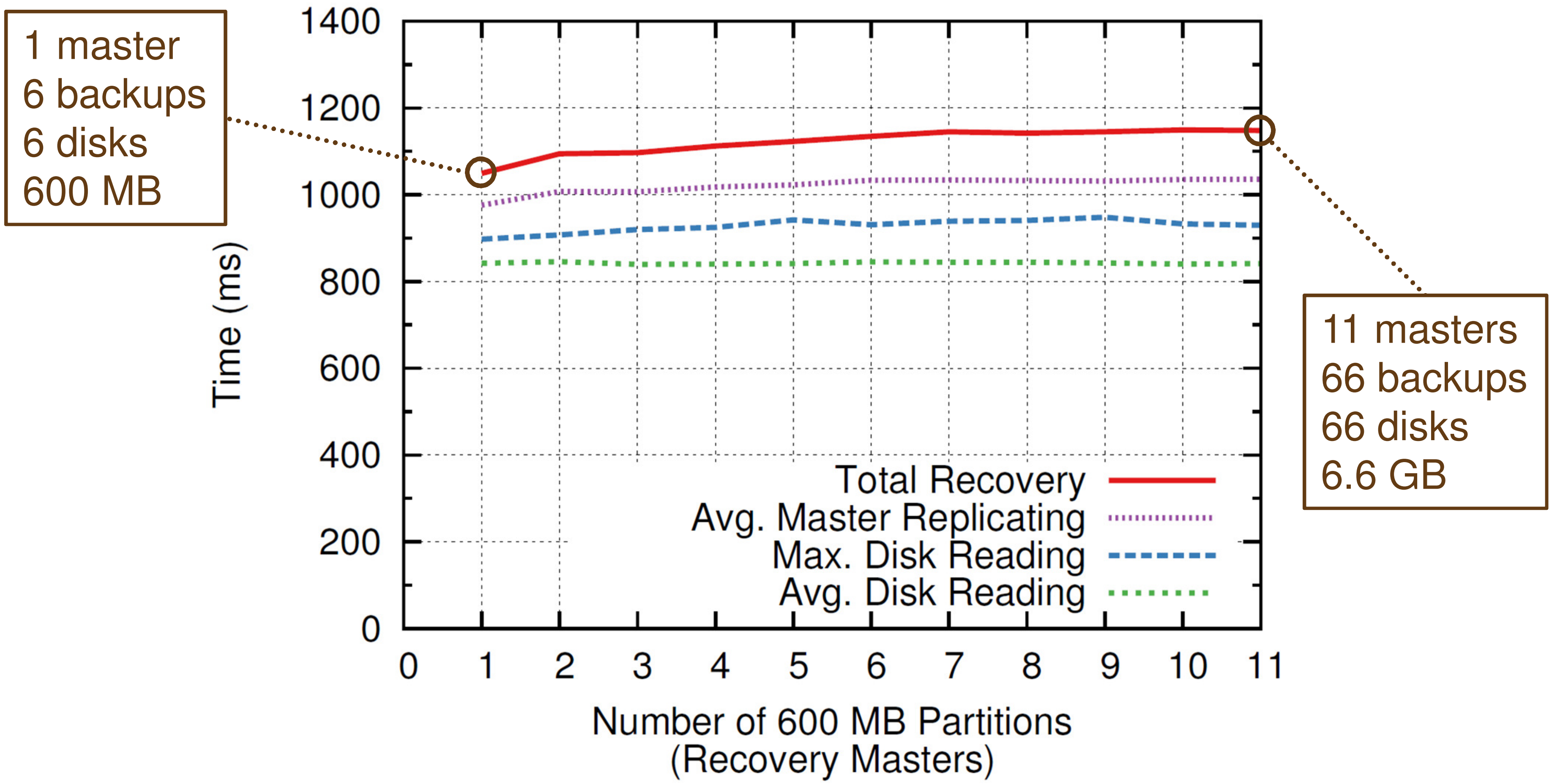
Project Status

- **Goal: build production-quality implementation**
- **Started coding Spring 2010**
- **Major pieces coming together:**
  - RPC subsystem
    - Supports many different transport layers
    - Using Mellanox Infiniband for high performance
  - Basic data model
  - Simple cluster coordinator
  - Fast recovery
- **Performance (40-node cluster):**
  - Read small object: 5µs
  - Throughput: > 1M small reads/second/server

Single Recovery Master



Recovery Scalability



Conclusion

- **Achieved low latency (at small scale)**
- **Not yet at large scale (but scalability encouraging)**
- **Fast recovery:**
  - 1 second for memory sizes < 10GB
  - Scalability looks good
  - Durable and available DRAM storage for the cost of volatile cache
- **Many interesting problems left**
- **Goals:**
  - Harness full performance potential of DRAM-based storage
  - Enable new applications: intensive manipulation of large-scale data

Why not a Caching Approach?

- **Lost performance:**
  - 1% misses → 10x performance degradation
- **Won't save much money:**
  - Already have to keep information in memory
  - Example: Facebook caches ~75% of data size
- **Availability gaps after crashes:**
  - System performance intolerable until cache refills
  - Facebook example: 2.5 hours to refill caches!

Data Model Rationale

Key-value store

Lower-level APIs
Less server functionality

Higher-level APIs
More server functionality

Distributed shared memory :

- ✓ Server implementation easy
- ✓ Low-level performance good
- ✗ APIs not convenient for applications
- ✗ Lose performance in application-level synchronization

Relational database :

- ✓ Powerful facilities for apps
- ✓ Best RDBMS performance
- ✗ Simple cases pay RDBMS performance
- ✗ More complexity in servers

How to get best application-level performance?

RAMCloud Motivation: Technology

Disk access rate not keeping up with capacity:

	Mid-1980's	2009	Change
Disk capacity	30 MB	500 GB	16667x
Max. transfer rate	2 MB/s	100 MB/s	50x
Latency (seek & rotate)	20 ms	10 ms	2x
Capacity/bandwidth (large blocks)	15 s	5000 s	333x
Capacity/bandwidth (1KB blocks)	600 s	58 days	8333x

- **Disks must become more archival**
- **More information must move to memory**