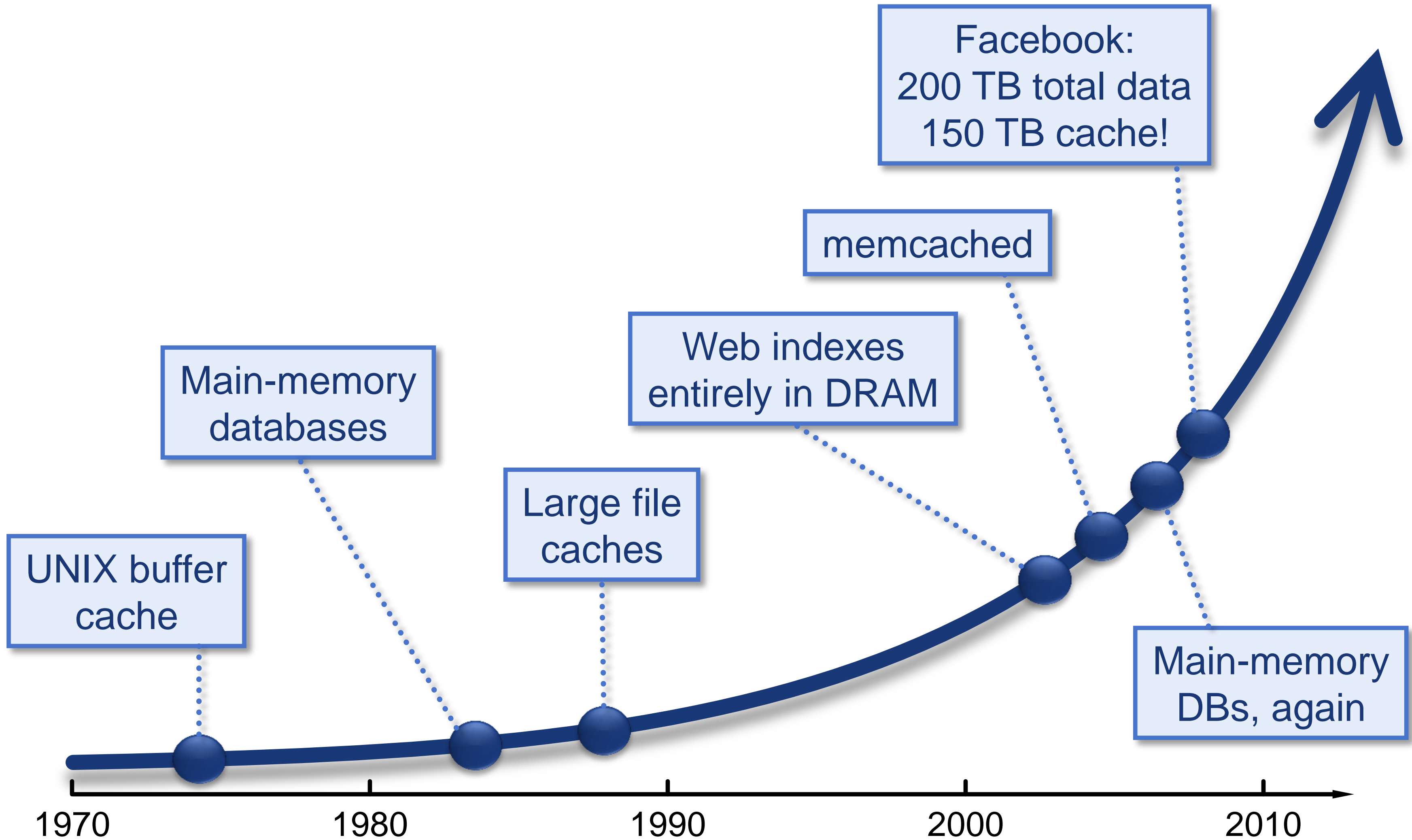
A Compilation of RAMCloud Slides (through Oct. 2012)

John Ousterhout

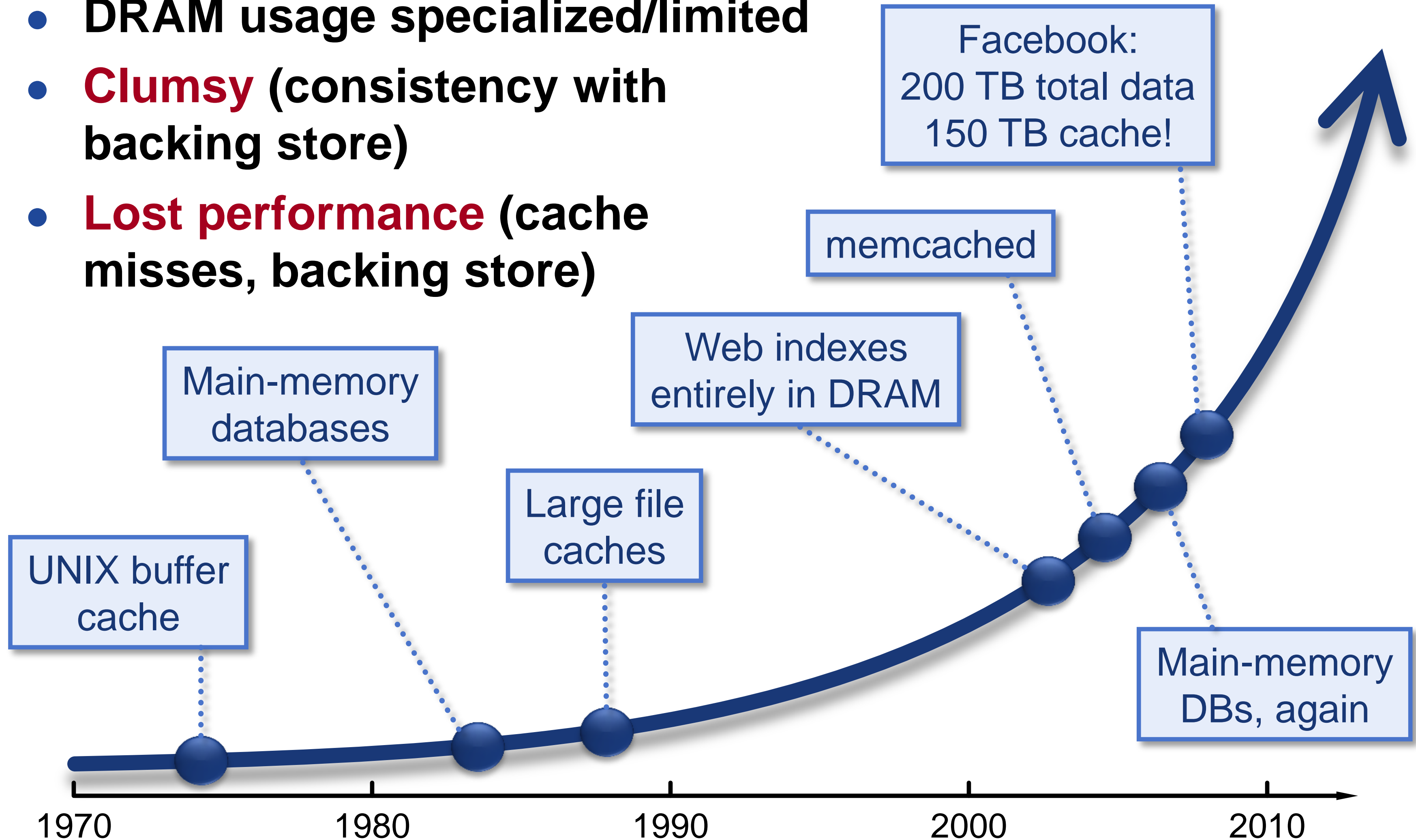
Stanford University

DRAM in Storage Systems



DRAM in Storage Systems

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



RAMCloud

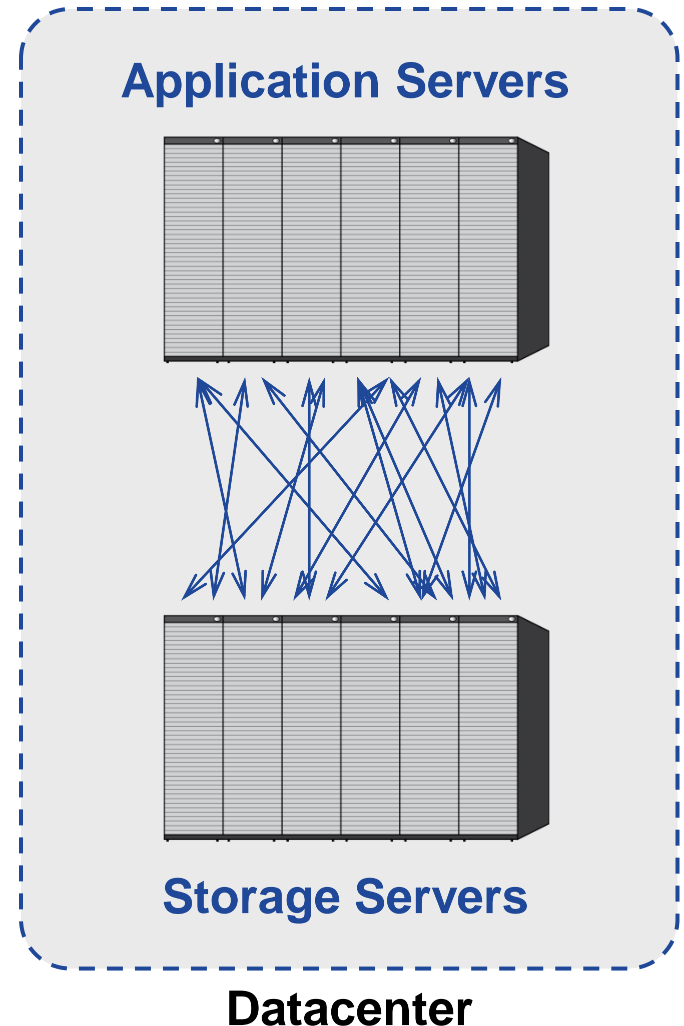
Harness full performance potential of large-scale DRAM storage:

- **General-purpose storage system**
- **All data always in DRAM (no cache misses)**
- **Durable and available**
- **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



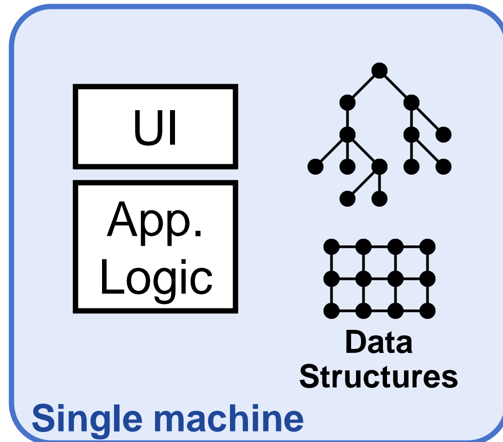
Example Configurations

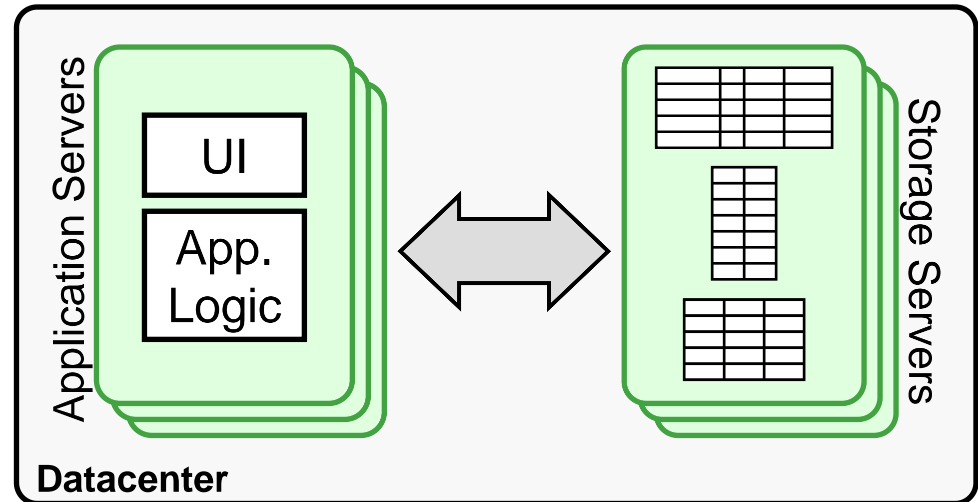
	2010	2015-2020
# servers	2000	4000
GB/server	24GB	256GB
Total capacity	48TB	1PB
Total server cost	$3.1M	$6M
$/GB	$65	$6

For $100-200K today:

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

Why Does Latency Matter?



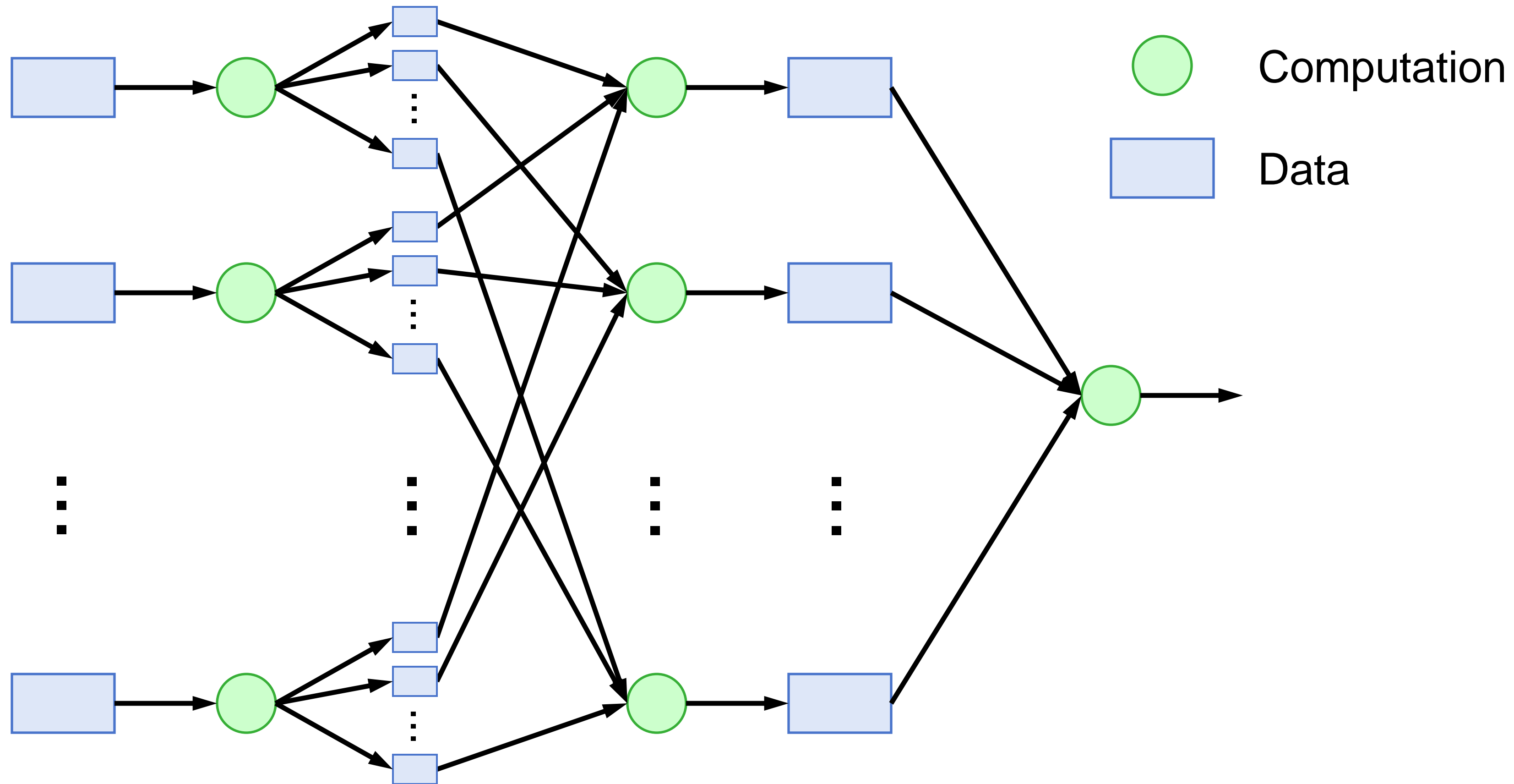


<< 1µs latency

0.5-10ms latency

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

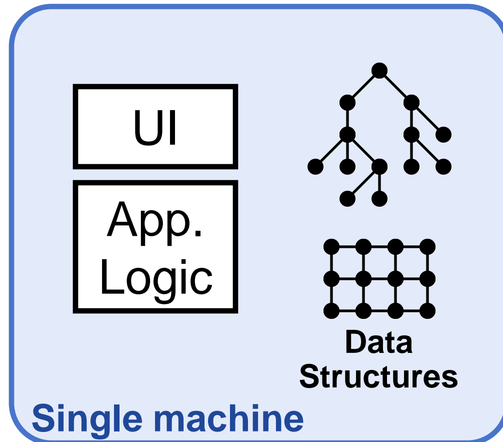
MapReduce



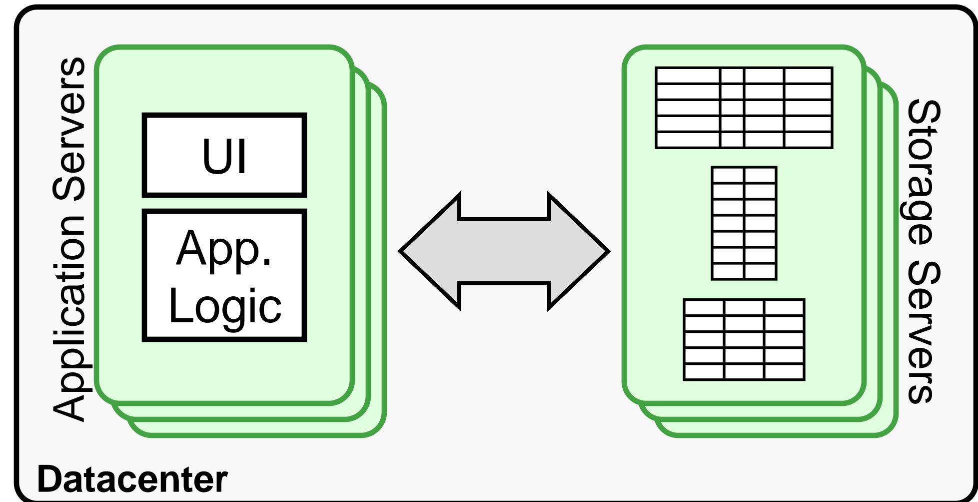
- ✓ **Sequential data access → high data access rate**
- ✗ **Not all applications fit this model**
- ✗ **Offline**

Goal: Scale and Latency

Traditional Application



Web Application

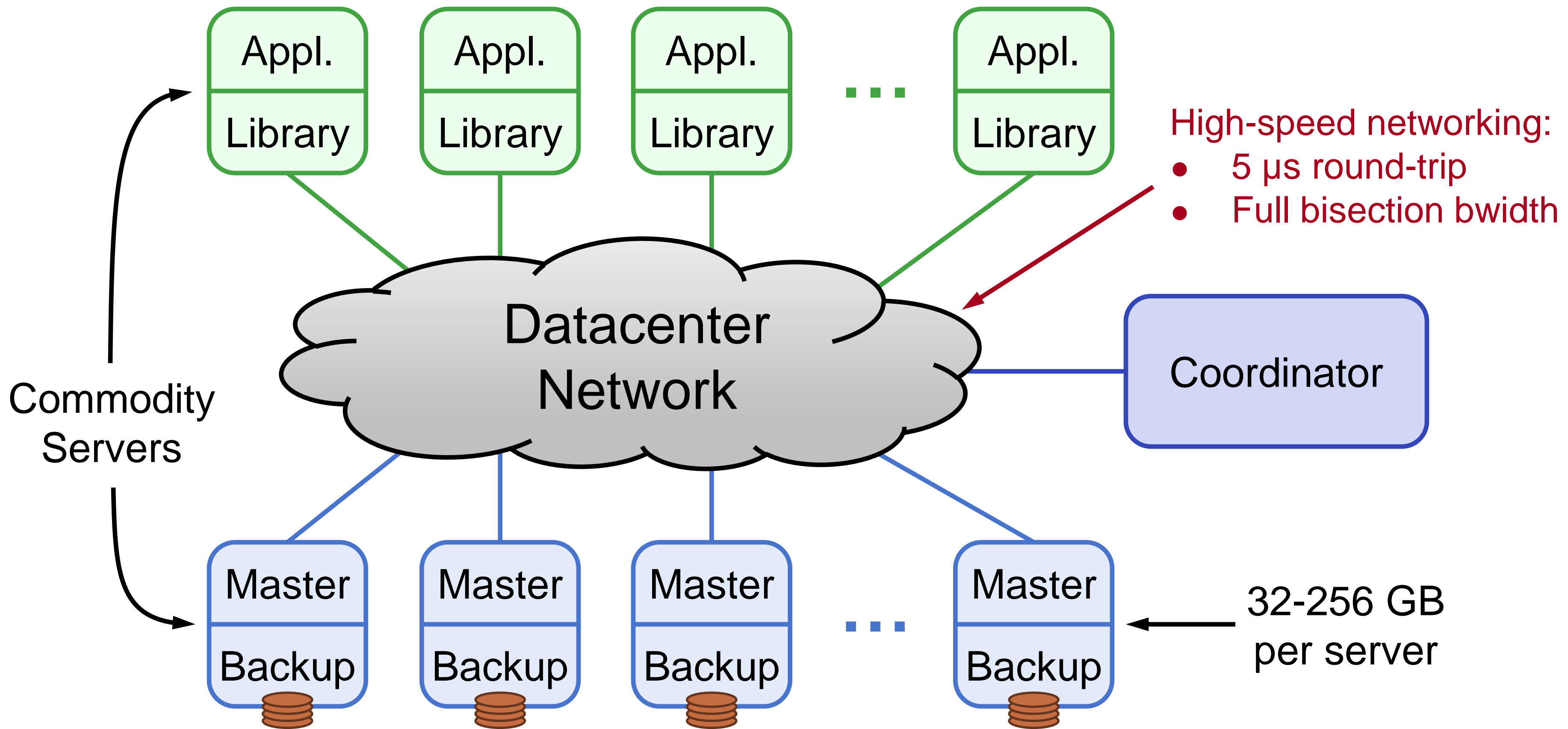


<< 1µs latency

~~0.5-10ms~~ latency
5-10µs

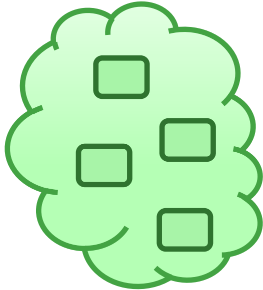
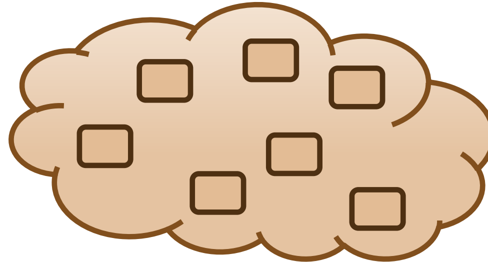
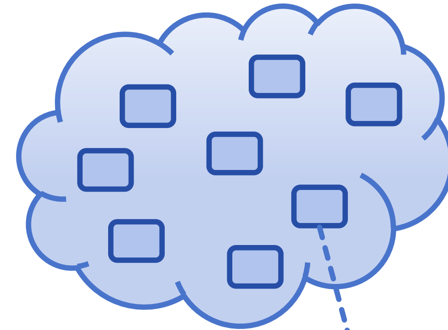
- **Enable new class of applications:**
  - Crowd-level collaboration
  - Large-scale graph algorithms
  - Real-time information-intensive applications

RAMCloud Architecture



Data Model

Tables

Object

Key (≤ 64KB)
Version (64b)
Blob (≤ 1MB)

```
read(tableId, key)
    => blob, version

write(tableId, key, blob)
    => version

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

delete(tableId, key)
```

(Only overwrite if version matches)

Richer model in the future:

- Indexes?
- Transactions?
- Graphs?

Research Issues

- Durability and availability
- Fast communication (RPC)
- Data model
- Concurrency, consistency, transactions
- Data distribution, scaling
- Multi-tenancy
- Client-server functional distribution
- Node architecture

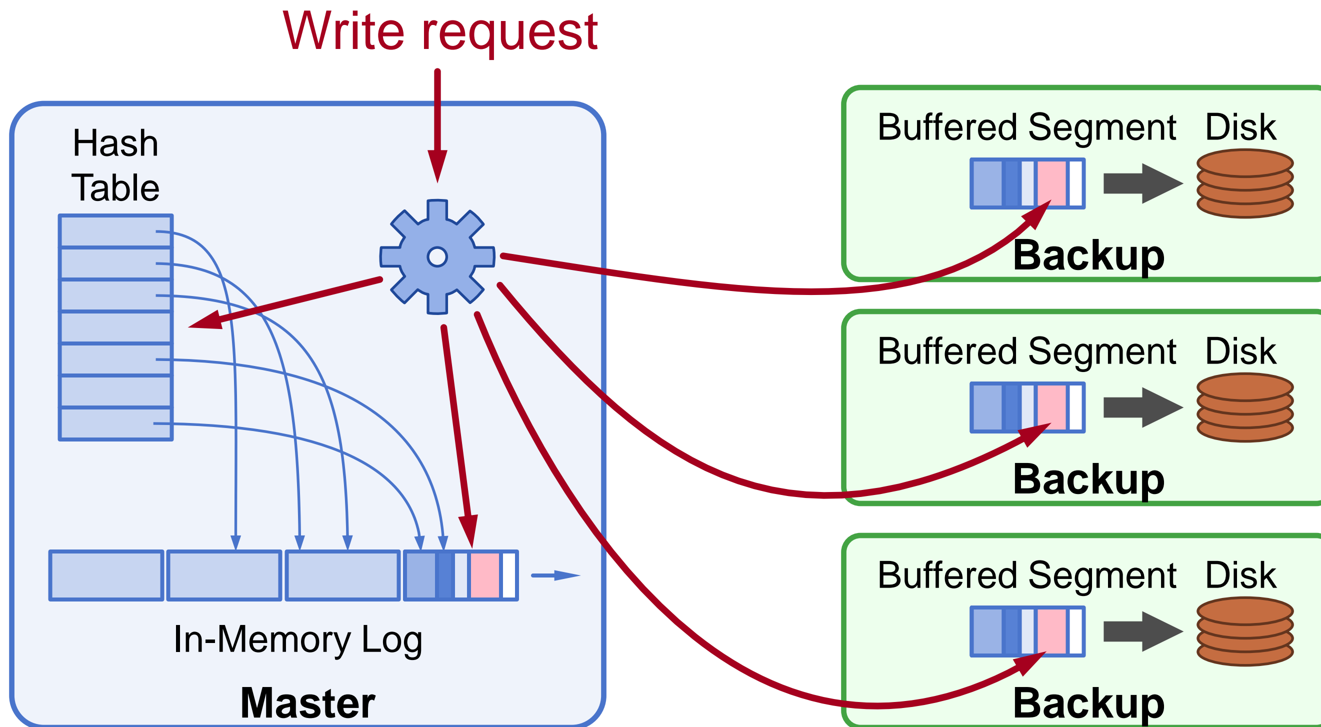
Research Areas

- **Data models for low latency and large scale**
- **Storage systems: replication, logging to make DRAM-based storage durable**
- **Performance:**
  - Serve requests in < 10 cache misses
  - Recover from crashes in 1-2 seconds
- **Networking: new protocols for low latency, datacenters**
- **Large-scale systems:**
  - Coordinate 1000's of machines
  - Automatic reconfiguration

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
- **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



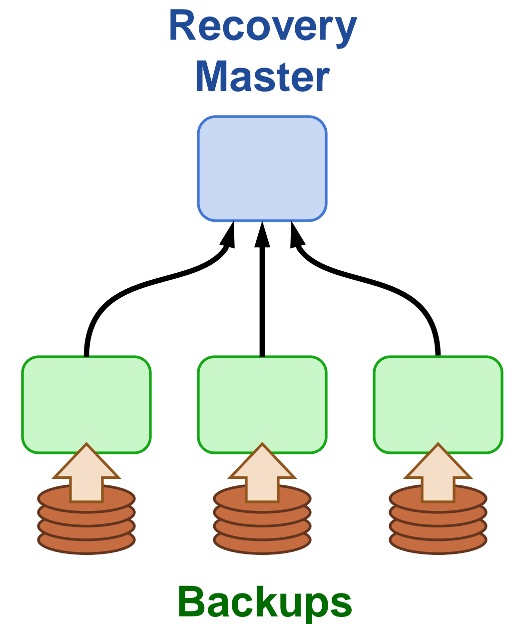
- **No disk I/O during write requests**
- **Log-structured: backup disk and master's memory**
- **Log cleaning ~ generational garbage collection**

Crash Recovery

- **Power failures: backups must guarantee durability of buffered data:**
  - Per-server battery backups?
  - DIMMs with built-in flash backup?
  - 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**

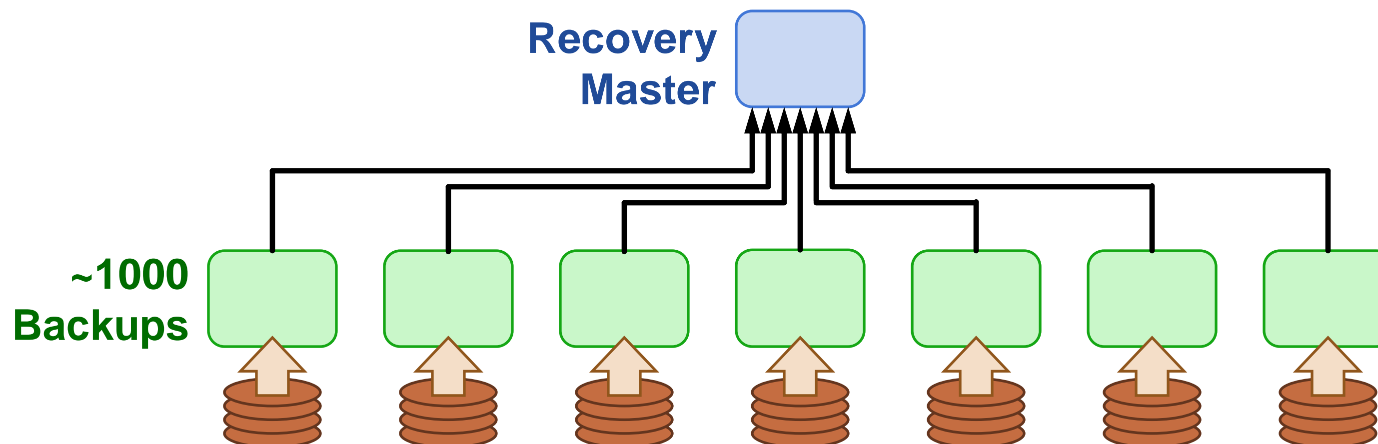
Recovery, First Try

- **Master chooses backups statically**
  - Each backup mirrors 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

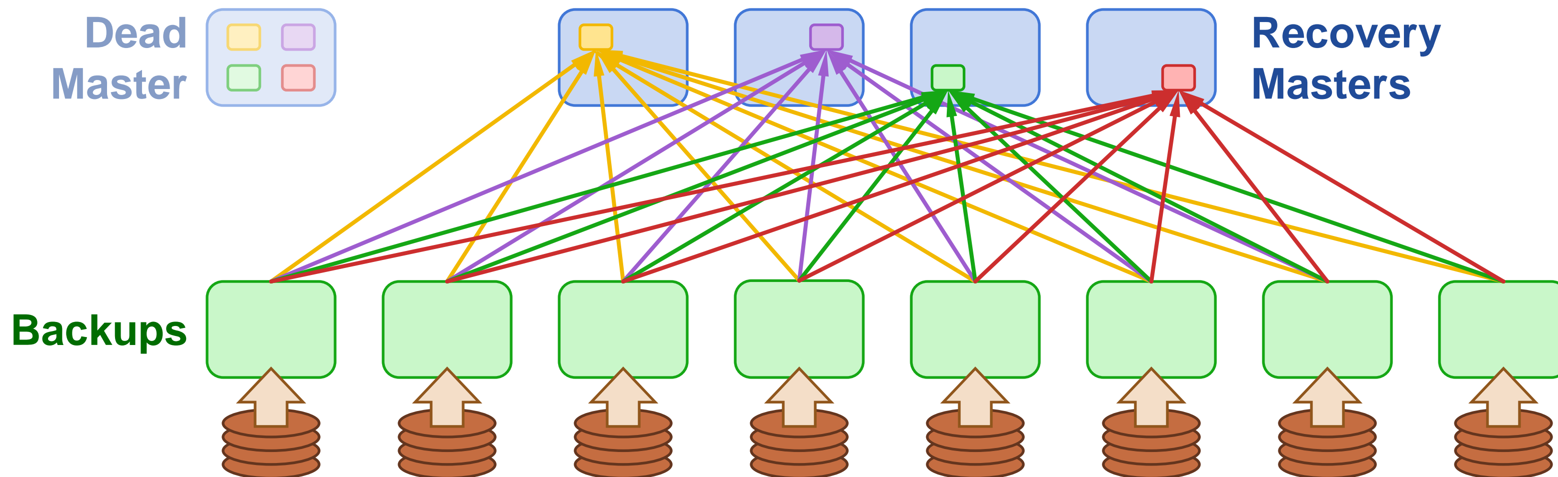


Scattered Logs, cont'd

- **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

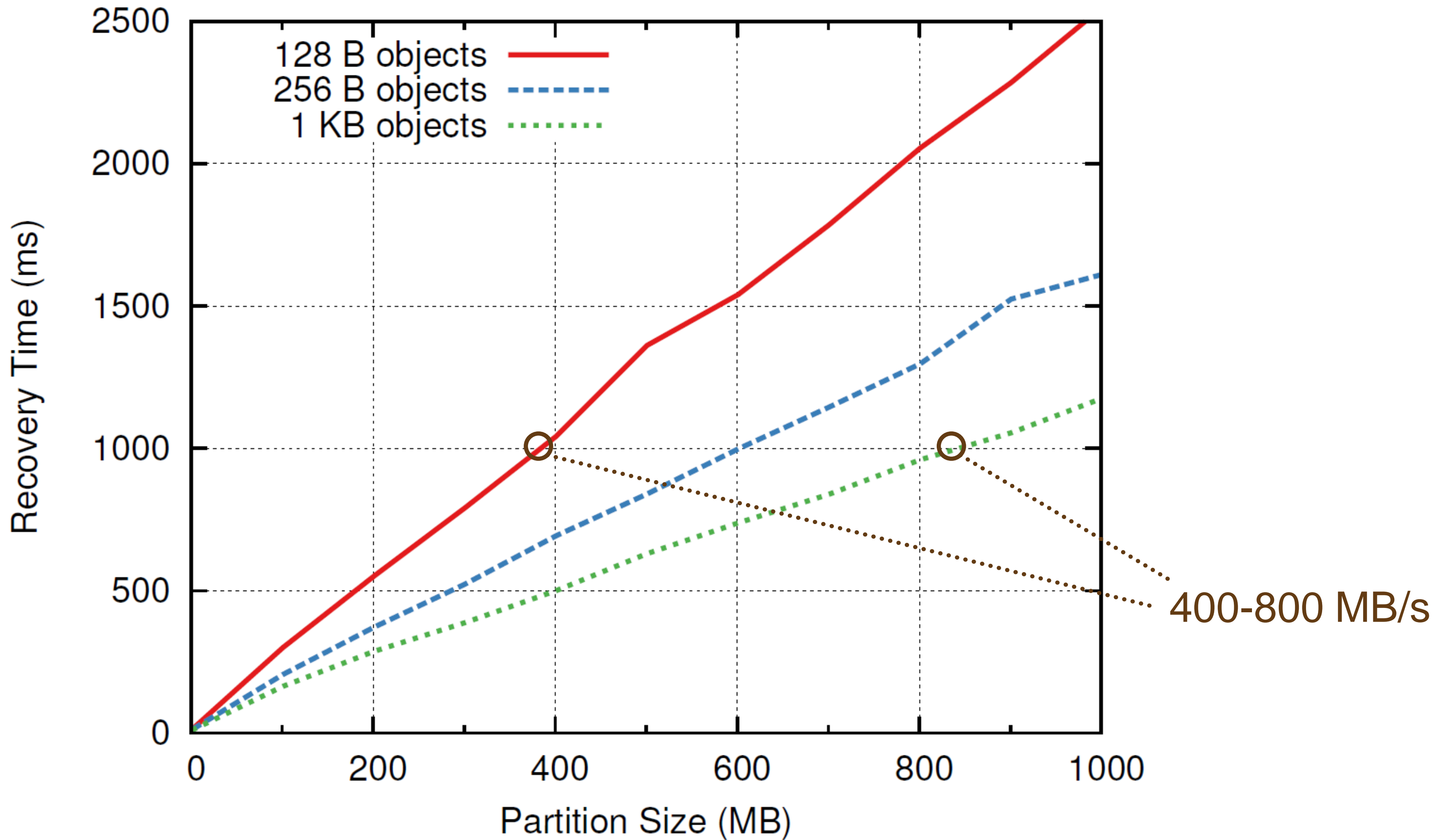
- **Divide each master's data into partitions**
  - Recover each partition on a separate recovery master
  - Partitions based on tables & key ranges, *not log segment*
  - Each backup divides its log data among recovery masters



Project Status

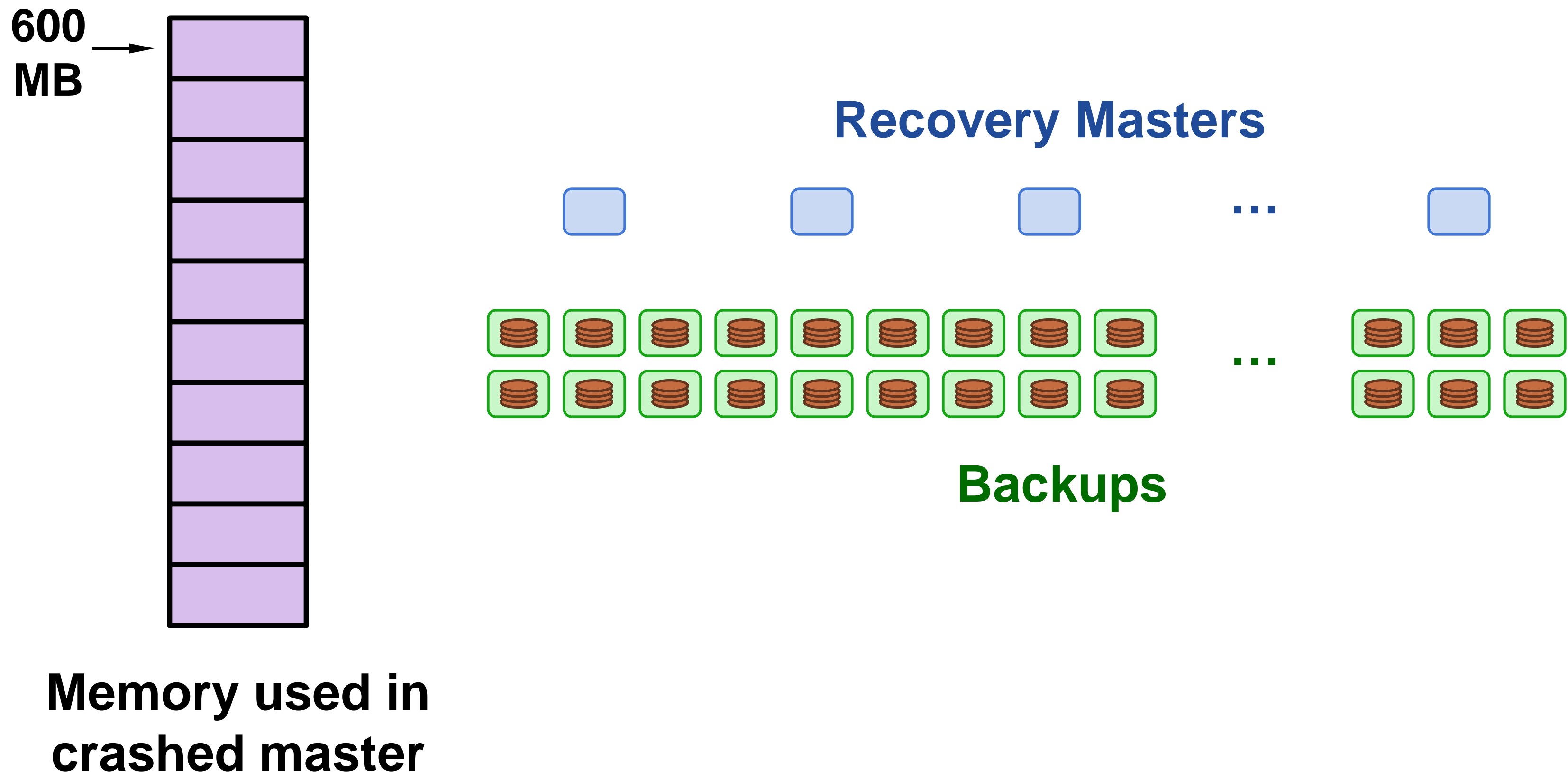
- **Goal: build production-quality implementation**
- **Started coding Spring 2010**
- **Major pieces working:**
  - RPC subsystem (supports multiple networking technologies)
  - Basic operations, log cleaning
  - Fast recovery
  - Prototype cluster coordinator
- **Nearing 1.0-level release**
- **Performance (80-node cluster):**
  - Read small object: 5.3μs
  - Throughput: 850K small reads/second/server

Single Recovery Master



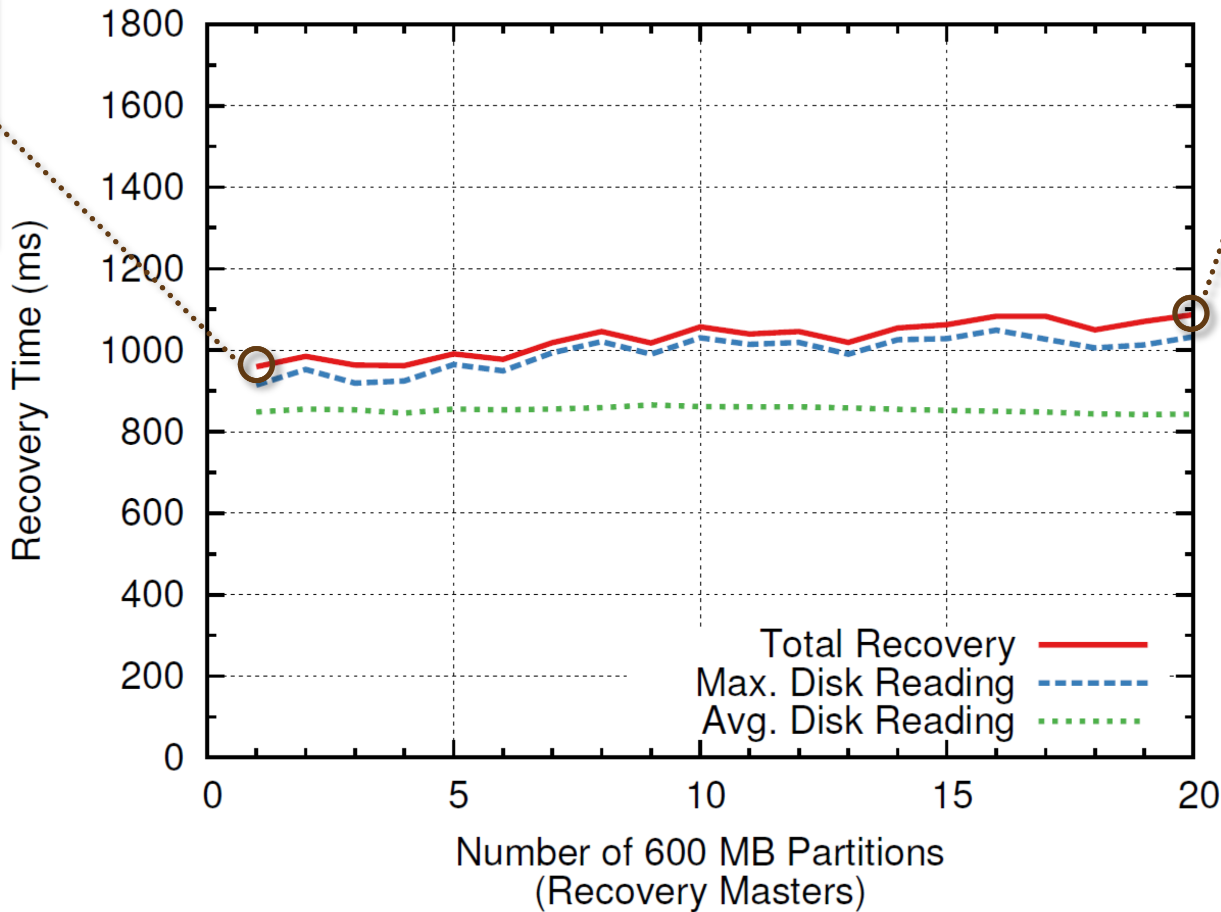
Evaluating Scalability

Recovery time should stay constant as system scales



Recovery Scalability

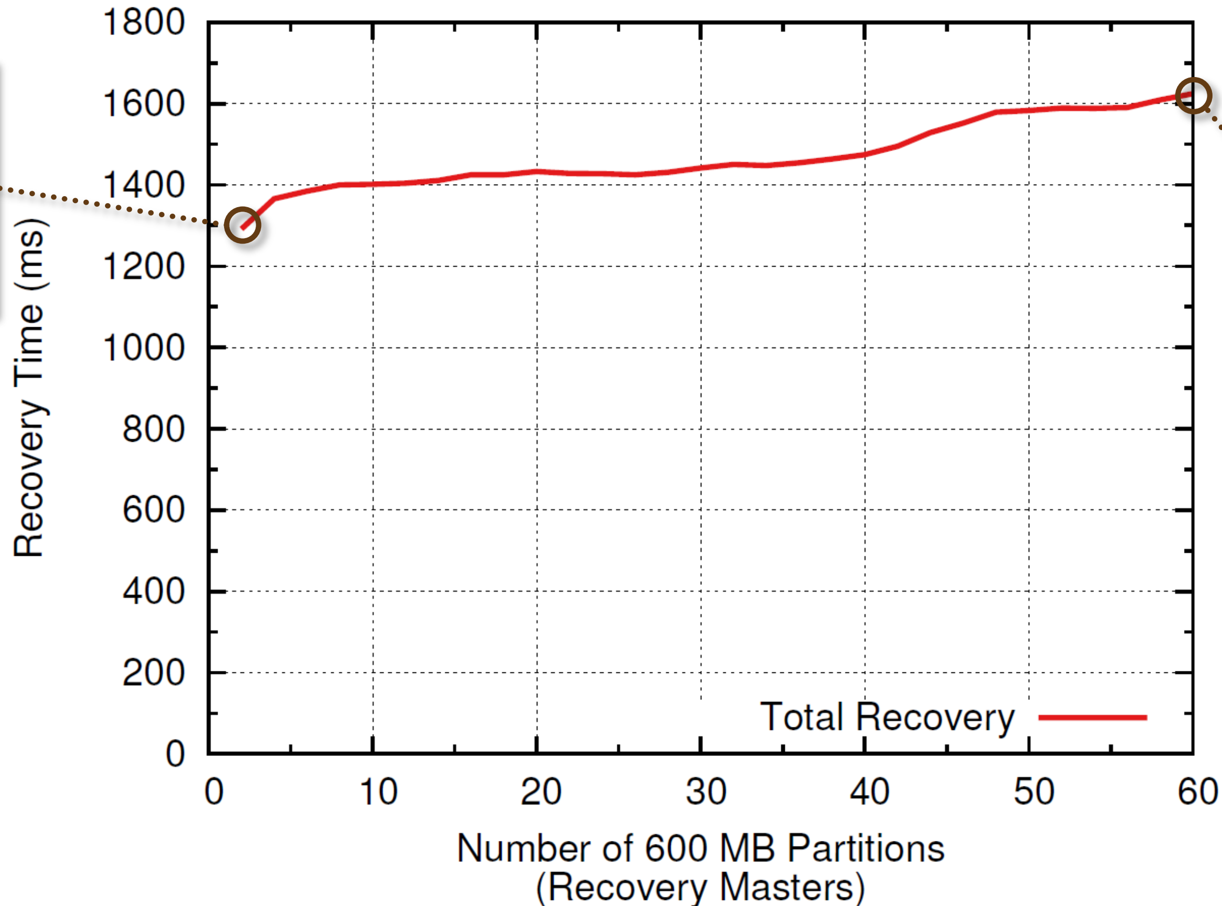
1 master
6 backups
6 disks
600 MB



20 masters
120 backups
120 disks
11.7 GB

Scalability (Flash)

2 flash drives (250MB/s) per partition:



Conclusion

- **Achieved low latency (at small scale)**
- **Not yet at large scale (but scalability encouraging)**
- **Fast recovery:**
  - < 2 seconds for memory sizes up to 35GB
  - 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

RPC Transport Architecture

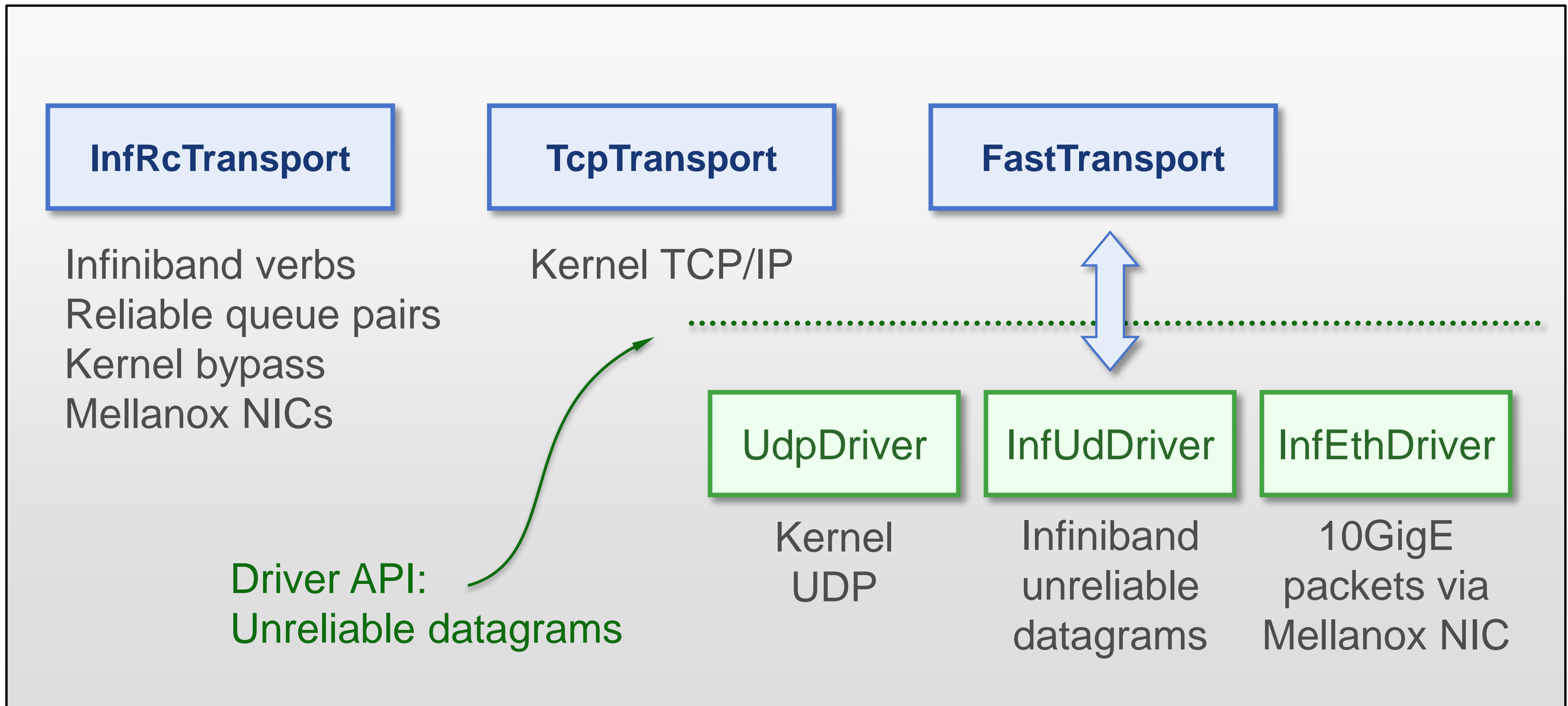
Transport API:
Reliable
request/response

Clients

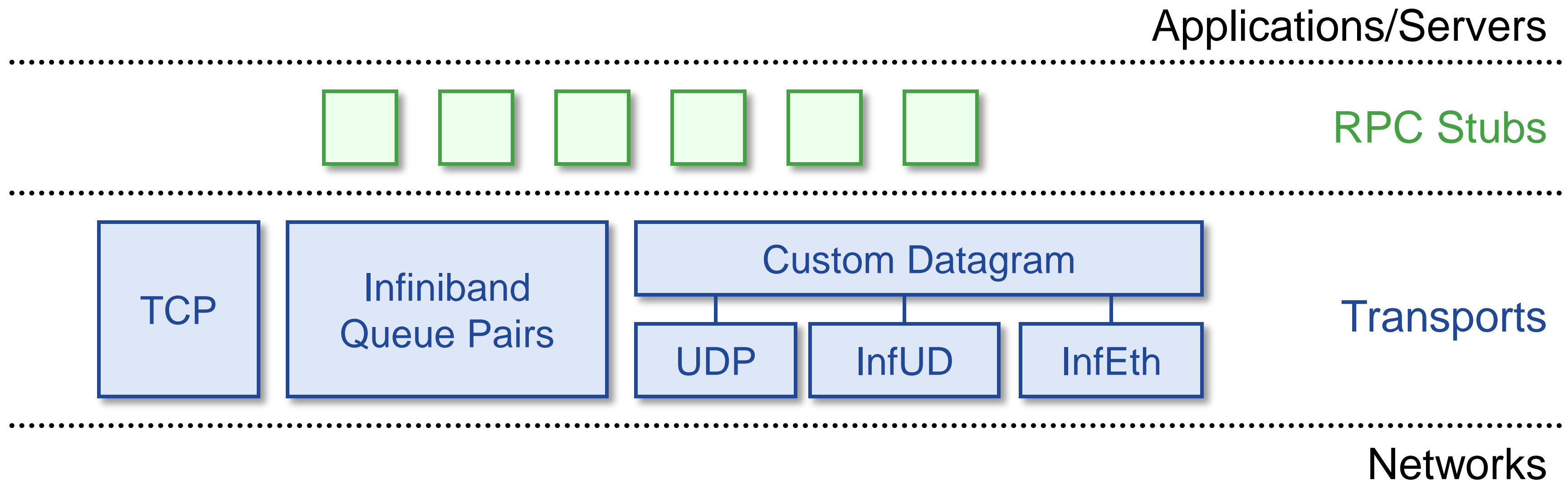
```
getSession(serviceLocator)
clientSend(reqBuf, respBuf)
wait()
```

Servers

```
handleRpc(reqBuf, respBuf)
```

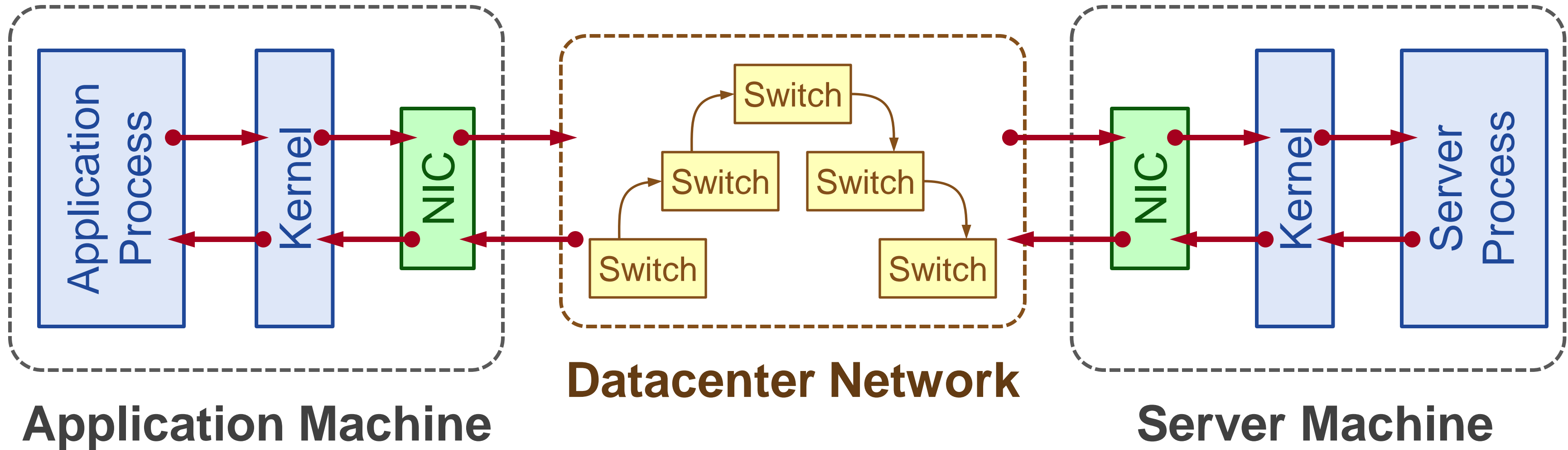


RAMCloud RPC



- **Transport layer enables experimentation with different networking protocols/technologies**
- **Basic Infiniband performance (one switch):**
  - 100-byte reads: 5.3 μs
  - 100-byte writes (3x replication): 16.1 μs
  - Read throughput (100 bytes, 1 server): 800 Kops/sec

Datacenter Latency Today



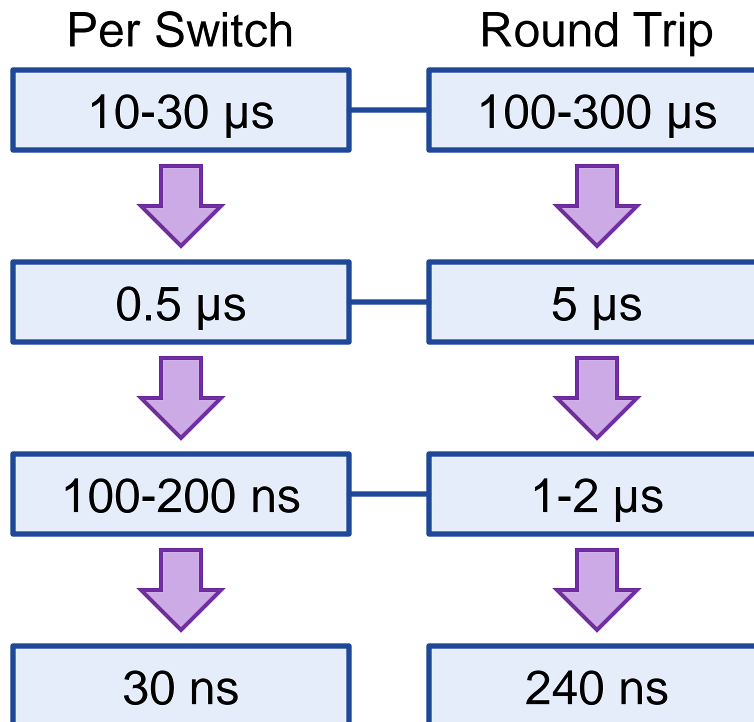
Component	Delay	Round-trip
Network switch	10-30µs	100-300µs
OS protocol stack	15µs	60µs
Network interface controller (NIC)	2.5-32µs	10-128µs
Propagation delay	0.2-0.4µs	0.4-0.8µs

RAMCloud goal: 5-10µs **Typical today: 200-400µs**

Faster RPC: Switches

Overall: must focus on latency, not bandwidth

- **Step 1: faster switching fabrics**



Typical 1 Gb/sec Ethernet switches

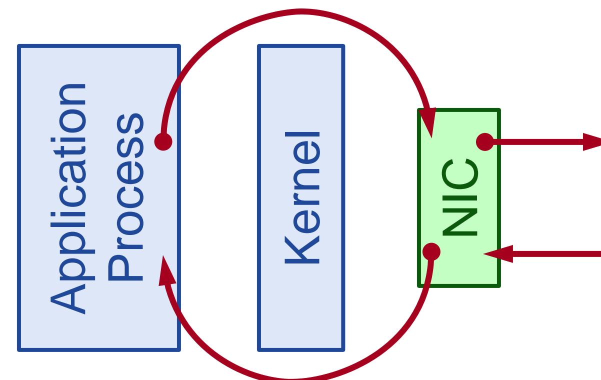
Newer 10 Gb/sec switches (e.g. Arista 7124)

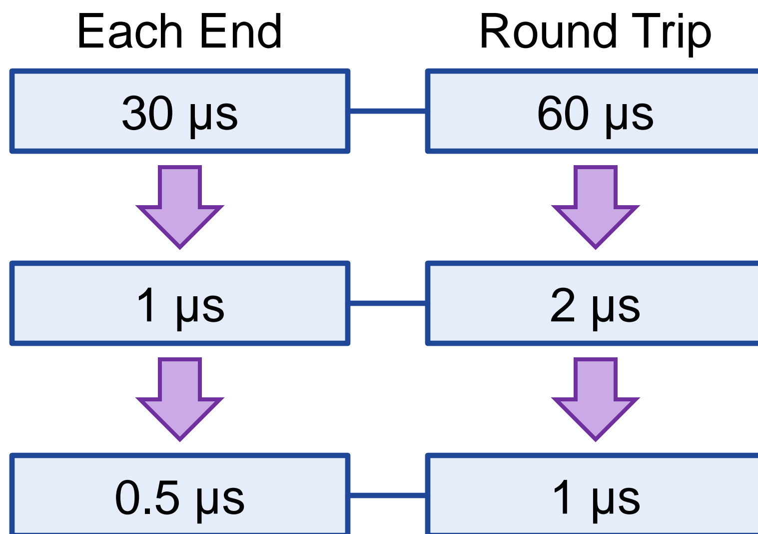
Infiniband switches

Radical new switching fabrics (Dally)

Faster RPC: Software

- **Step 2: new software architecture:**
  - Packets cannot pass through OS
    - Direct user-level access to NIC
  - Polling instead of interrupts
  - New network protocol





Typical today

With kernel bypass

With new NIC architecture

Faster RPC: NICs

- **Traditional NICs focus on throughput, not latency**
  - E.g. defer interrupts for 32 µs to enable coalescing
- **CPU-NIC interactions are expensive:**
  - Data must pass through memory
  - High-latency interconnects (Northbridge, PCIe, etc.)
  - Interrupts
- **Best-case today:**
  - 0.75 µs per NIC traversal
  - 3 µs round-trip delay
- **Example: Mellanox Infiniband NICs (kernel bypass)**

Total Round-Trip Delay

Component	Today	3-5 Years	10 Years
Switching fabric	100-300µs	5µs	0.2µs
Software	60µs	2µs	2µs
NIC	8-128µs	3µs	3µs
Propagation delay	1µs	1µs	1µs
Total	200-400µs	11µs	6.2µs

- **In 10 years, 2/3 of round-trip delay due to NIC!!**
  - 3µs directly from NIC
  - 1µs indirectly from software (communication, cache misses)

New NIC Architecture?

Must integrate NIC tightly into CPU cores:

- **Bits pass directly between L1 cache and the network**
- **Direct access from user space**
- **Will require architectural changes:**
  - New instructions for network communication
  - Some packet buffering in hardware
  - Hardware tables to give OS control
    - Analogous to page tables
    - Take ideas from OpenFlow?
  - Ideally: CPU architecture designed in conjunction with switching fabric
    - E.g. minimize buffering

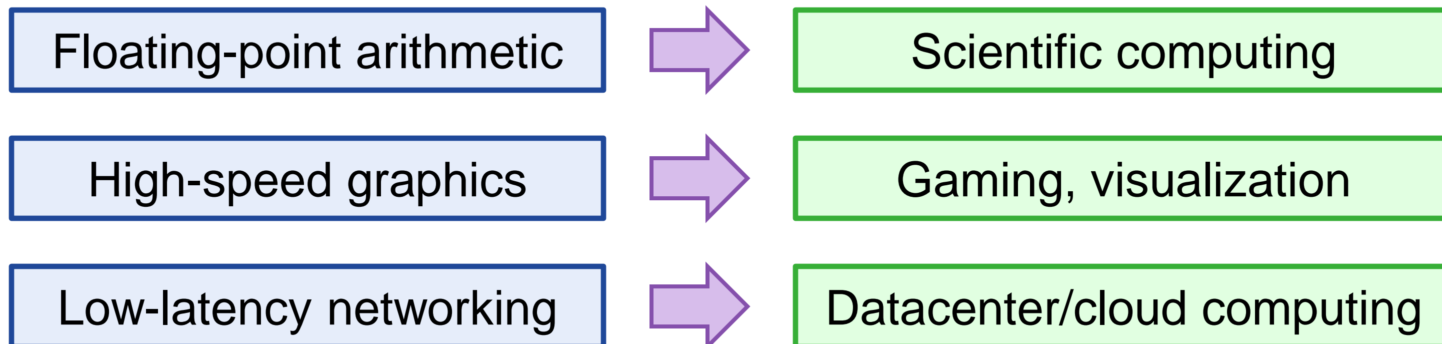
Round-Trip Delay, Revisited

Component	Today	3-5 Years	10 Years
Switching fabric	100-300µs	5µs	0.2µs
Software	60µs	2µs	1µs
NIC	8-128µs	3µs	0.2µs
Propagation delay	1µs	1µs	1µs
Total	200-400µs	11µs	2.4µs

- **Biggest remaining hurdles:**
  - Software
  - Speed of light

Other Arguments

- **Integrated functionality drives applications & markets:**



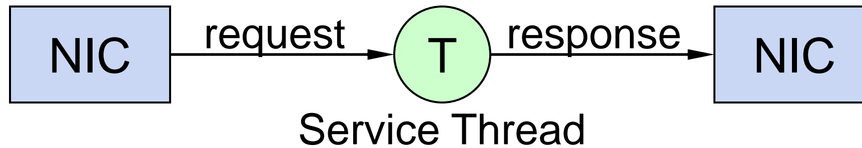
- **As # cores increases, on-chip networks will become essential: solve the off-chip problem at the same time!**

Using Cores

- **Goal: service RPC request in 1µs:**
  - < 10 L2 cache misses!
  - Cross-chip synchronization cost: ~1 L2 cache miss
- **Using multiple cores/threads:**
  - Synchronization overhead will increase latency
  - Concurrency may improve throughput
- **Questions:**
  - What is the best way to use multiple cores?
  - Does using multiple cores help performance?
  - Can a single core saturate the network?

Baseline

- **1 core, 1 thread:**

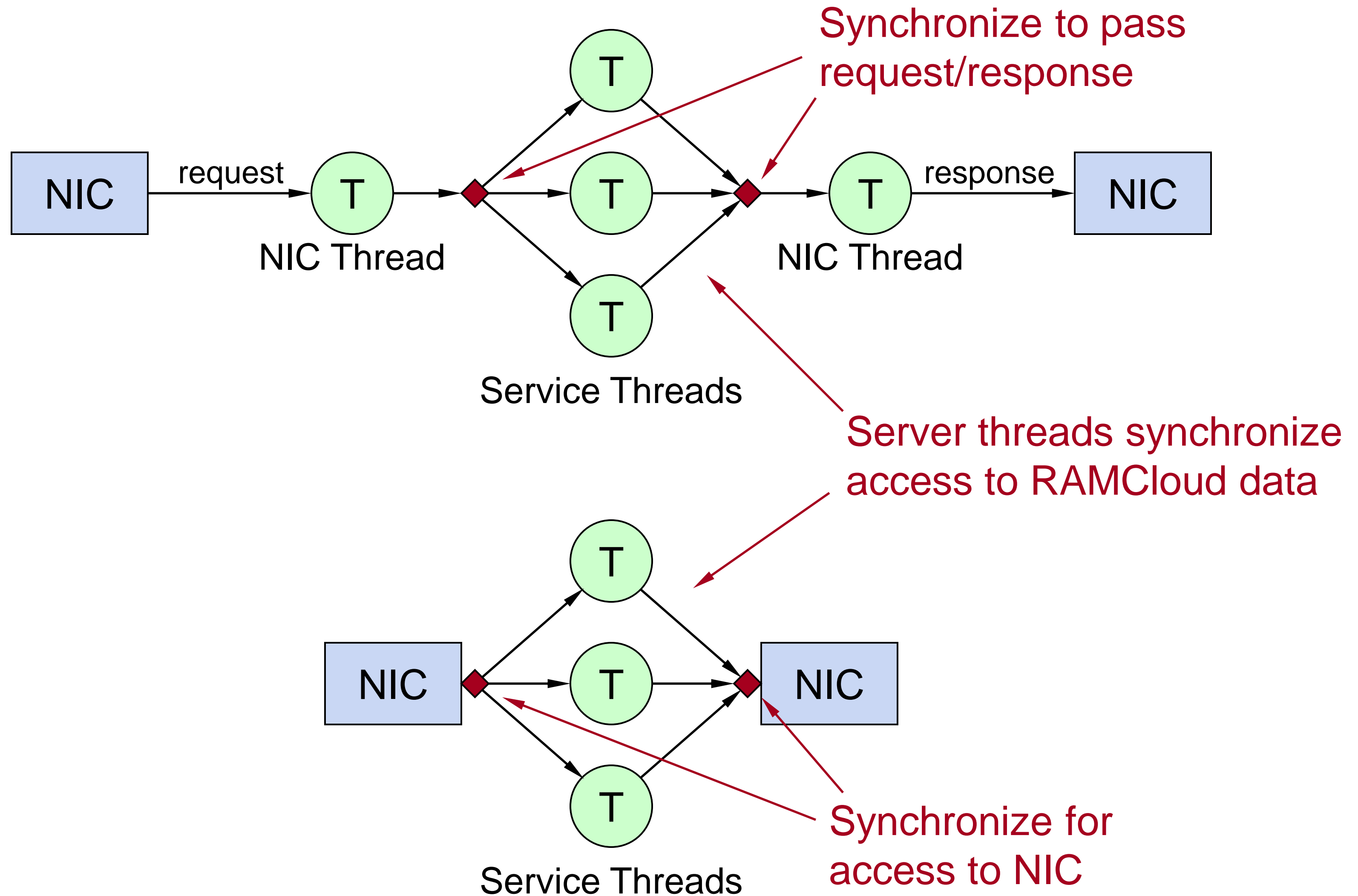


- **Poll NIC, process request, send response**

✓ **No synchronization overhead**

✗ **No concurrency**

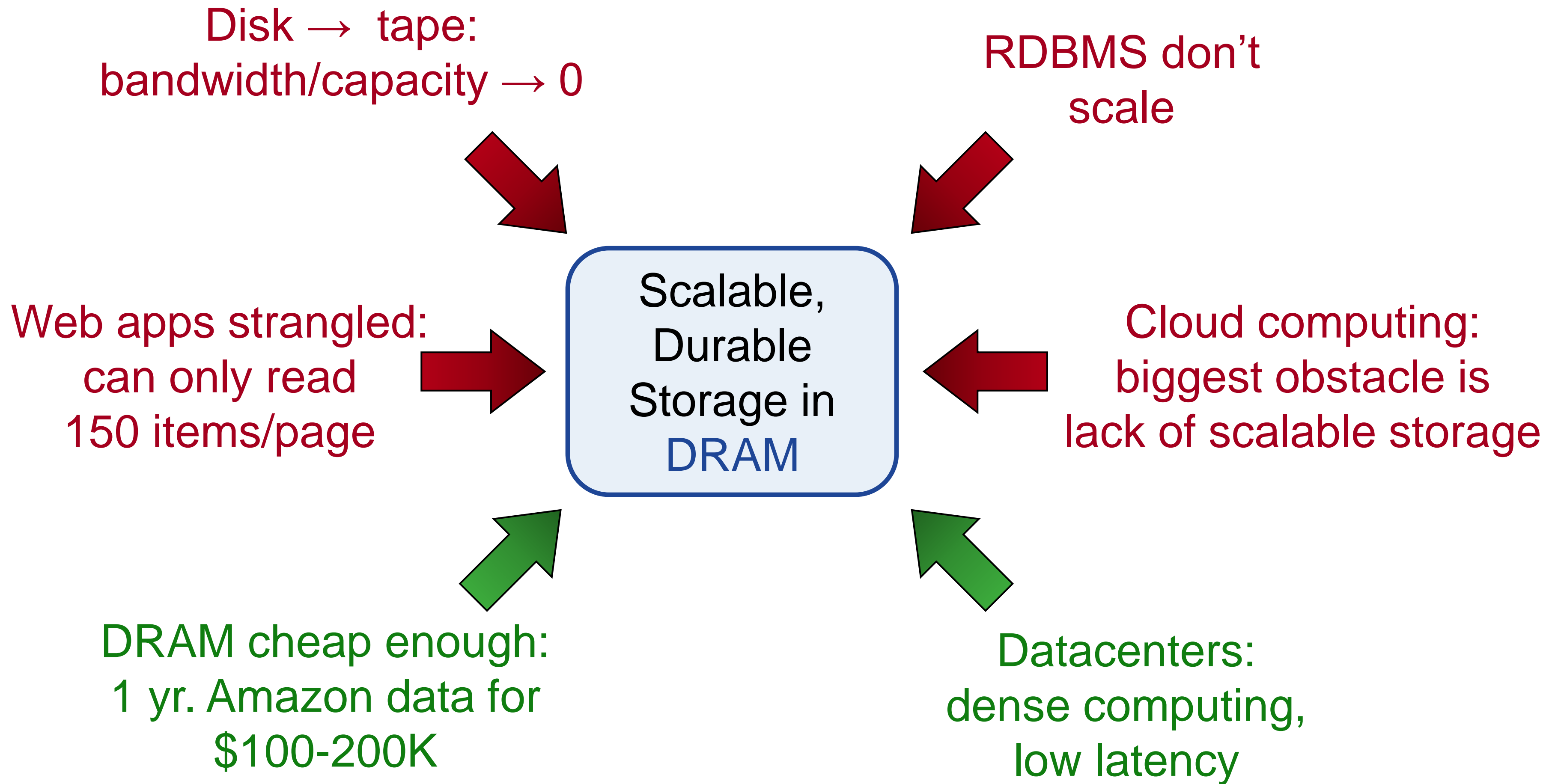
Multi-Thread Choices



Other Thoughts

- **If all threads/cores in a single package:**
  - No external memory references for synchronization
  - Does this make synchronization significantly faster?
- **Why not integrated NIC?**
  - Transfer incoming packets directly to L2 cache (*which* L2 cache?....)
  - Eliminate cache misses to read packets
- **Plan:**
  - Implement a couple of different approaches
  - Compare latency and bandwidth

Winds of Change



Research Areas

- **Data models for low latency and large scale**
- **Storage systems: replication, logging to make DRAM-based storage durable**
- **Performance:**
  - Serve requests in < 10 cache misses
  - Recover from crashes in 1-2 seconds
- **Networking: new protocols for low latency, datacenters**
- **Large-scale systems:**
  - Coordinate 1000's of machines
  - Automatic reconfiguration

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!

Why not Flash Memory?

- **DRAM enables lowest latency today:**
  - 5-10x faster than flash
- **Many candidate technologies besides DRAM**
  - Flash (NAND, NOR)
  - PC RAM
  - …
- **Most RAMCloud techniques will apply to other technologies**

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

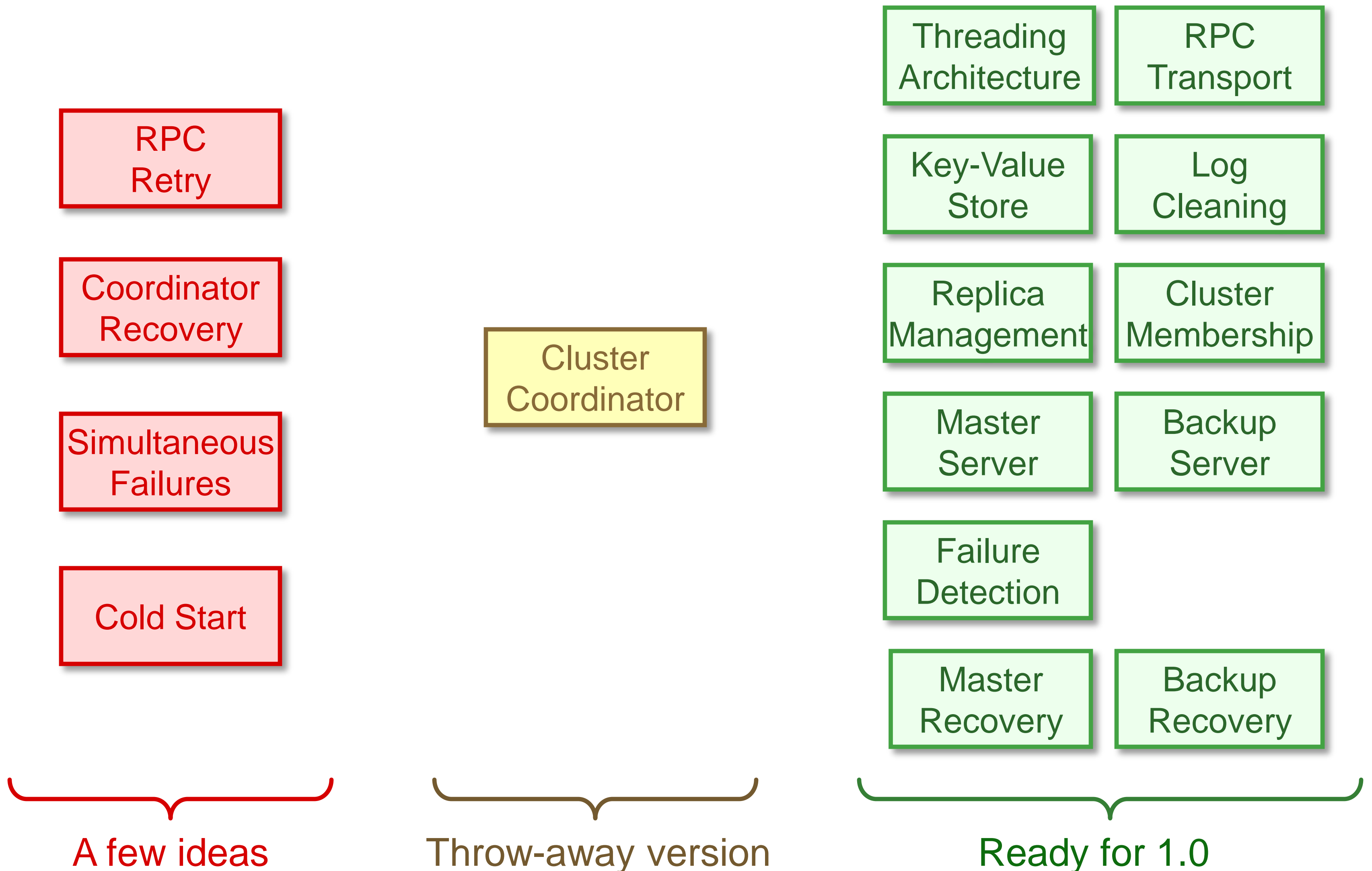
RAMCloud Motivation: Technology

Disk access rate not keeping up with capacity:

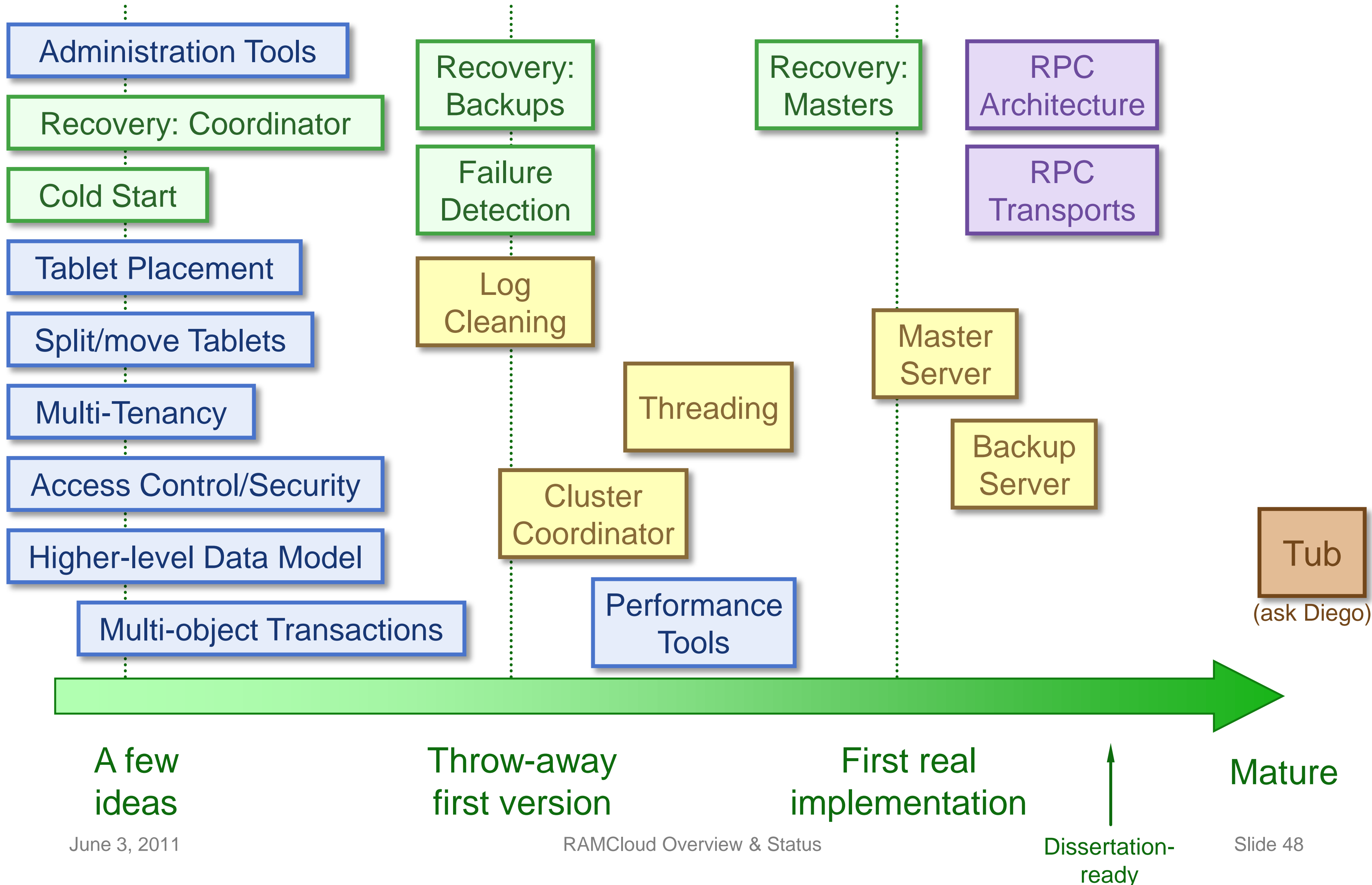
	Mid-1980's	2009	Change
Disk capacity	30 MB	500 GB	16667x
Max. transfer rate	2 MB/s	100 MB/s	50x
Latency (seek & rotate)	20 ms	10 ms	2x
Capacity/bandwidth (large blocks)	15 s	5000 s	333x
Capacity/bandwidth (1KB blocks)	600 s	58 days	8333x
Jim Gray's rule	5 min	30 hours	360x

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

Implementation Status



Implementation Status



RAMCloud Code Size

Code	36,900 lines
Unit tests	16,500 lines
Total	53,400 lines

Selected Performance Metrics

- **Latency for 100-byte reads (1 switch):**

InfRc	4.9 μs
TCP (1GigE)	92 μs
TCP (Infiniband)	47 μs
Fast + UDP (1GigE)	91 μs
Fast + UDP (Infiniband)	44 μs
Fast + InfUd	4.9 μs

- **Server throughput (InfRc, 100-byte reads, one core):**

  1.05×10^6 requests/sec

- **Recovery time (6.6GB data, 11 recovery masters, 66 backups)**

  1.15 sec

Lessons/Conclusions (so far)

- **Fast RPC is within reach**
- **NIC is biggest long-term bottleneck: must integrate with CPU**
- **Can recover fast enough that replication isn't needed for availability**
- **Randomized approaches are key to scalable distributed decision-making**

The Datacenter Opportunity

- **Exciting combination of features for research:**
  - Concentrated compute power (~100,000 machines)
  - Large amounts of storage:
    - 1 Pbyte DRAM
    - 100 Pbytes disk
  - Potential for fast communication:
    - Low latency (speed-of-light delay < 1μs)
    - High bandwidth
  - Homogeneous
- **Controlled environment enables experimentation:**
  - E.g. new network protocols
- **Huge Petri dish for innovation over next decade**

How Many Datacenters?

- **Suppose we capitalize IT at the same level as other infrastructure (power, water, highways, telecom):**
  - $1-10K per person?
  - 1-10 datacenter servers/person?

	U.S.	World
Servers	**0.3-3B**	**7-70B**
Datacenters	**3000-30,000**	**70,000-700,000**

(assumes 100,000 servers/datacenter)

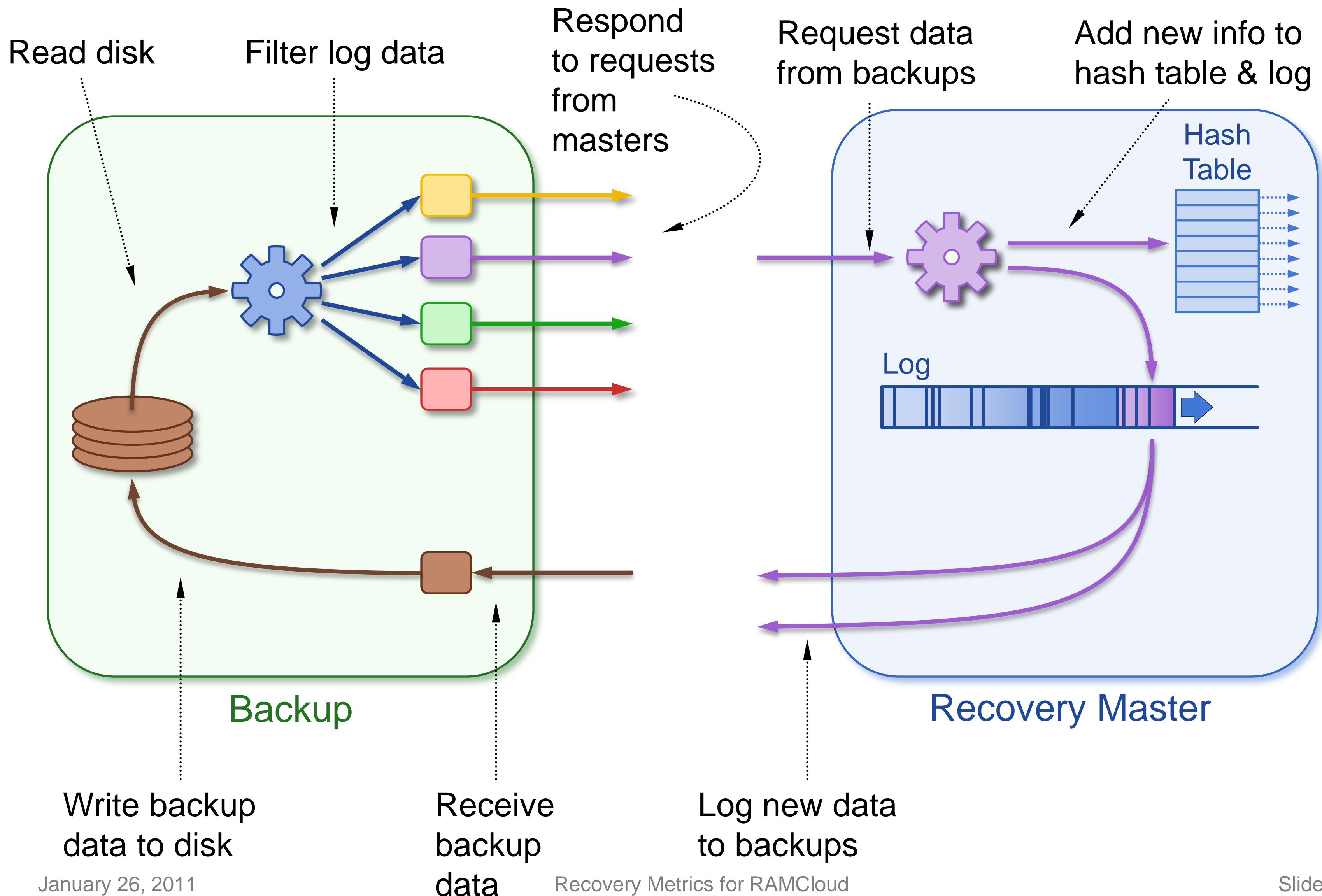
- **Computing in 10 years:**
  - Devices provide user interfaces
  - Most general-purpose computing (i.e. Intel processors) will be in datacenters

Logging/Recovery Basics

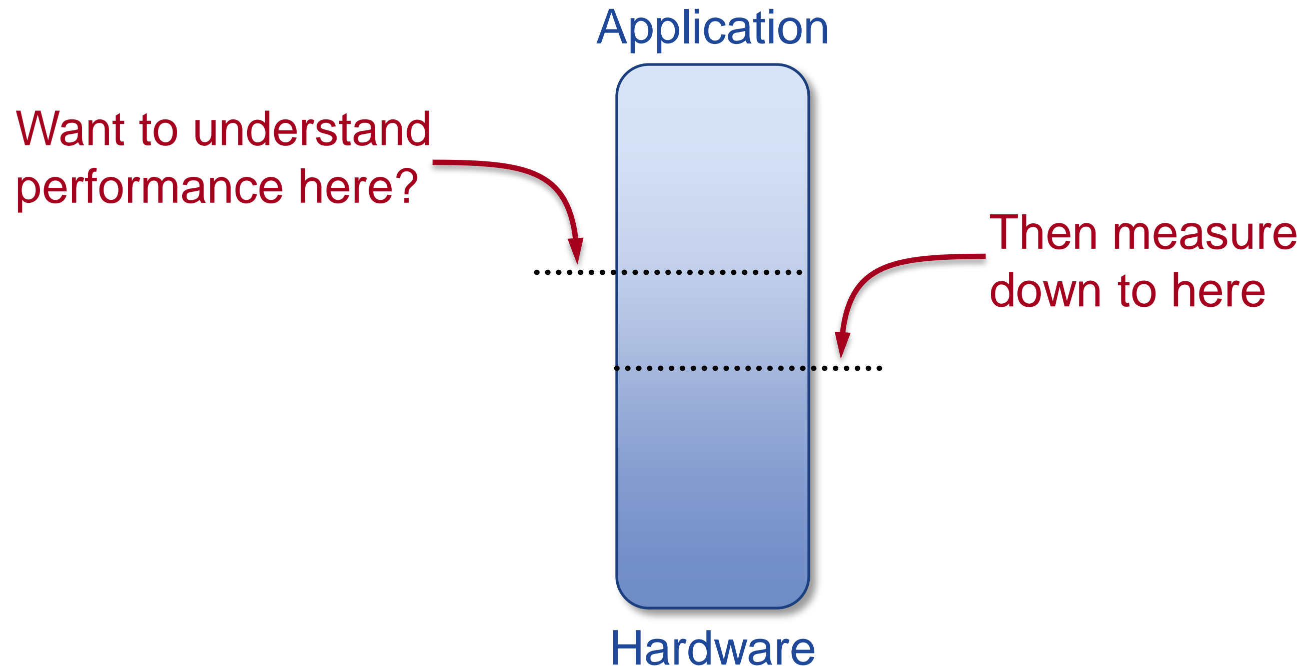
- **All data/changes appended to a log:**

- **One log for each master (kept in DRAM)**
- **Log data is replicated on disk on 2+ backups**
- **During recovery:**
  - Read data from disks on backups
  - Replay log to recreate data on master
- **Recovery must be fast: 1-2 seconds!**
  - Only one copy of data in DRAM
  - Data unavailable until recovery completes

Parallelism in Recovery

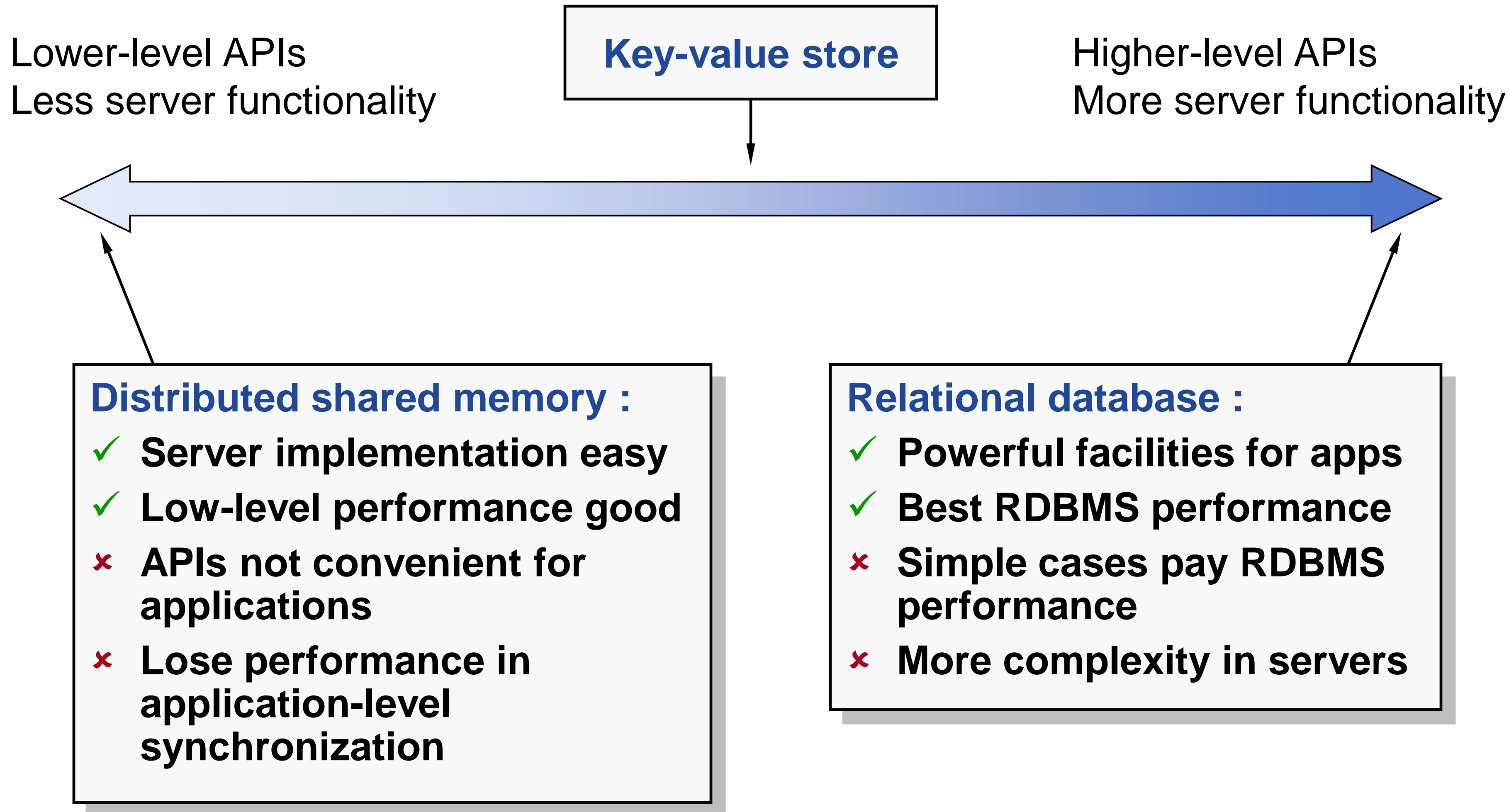


Measure Deeper



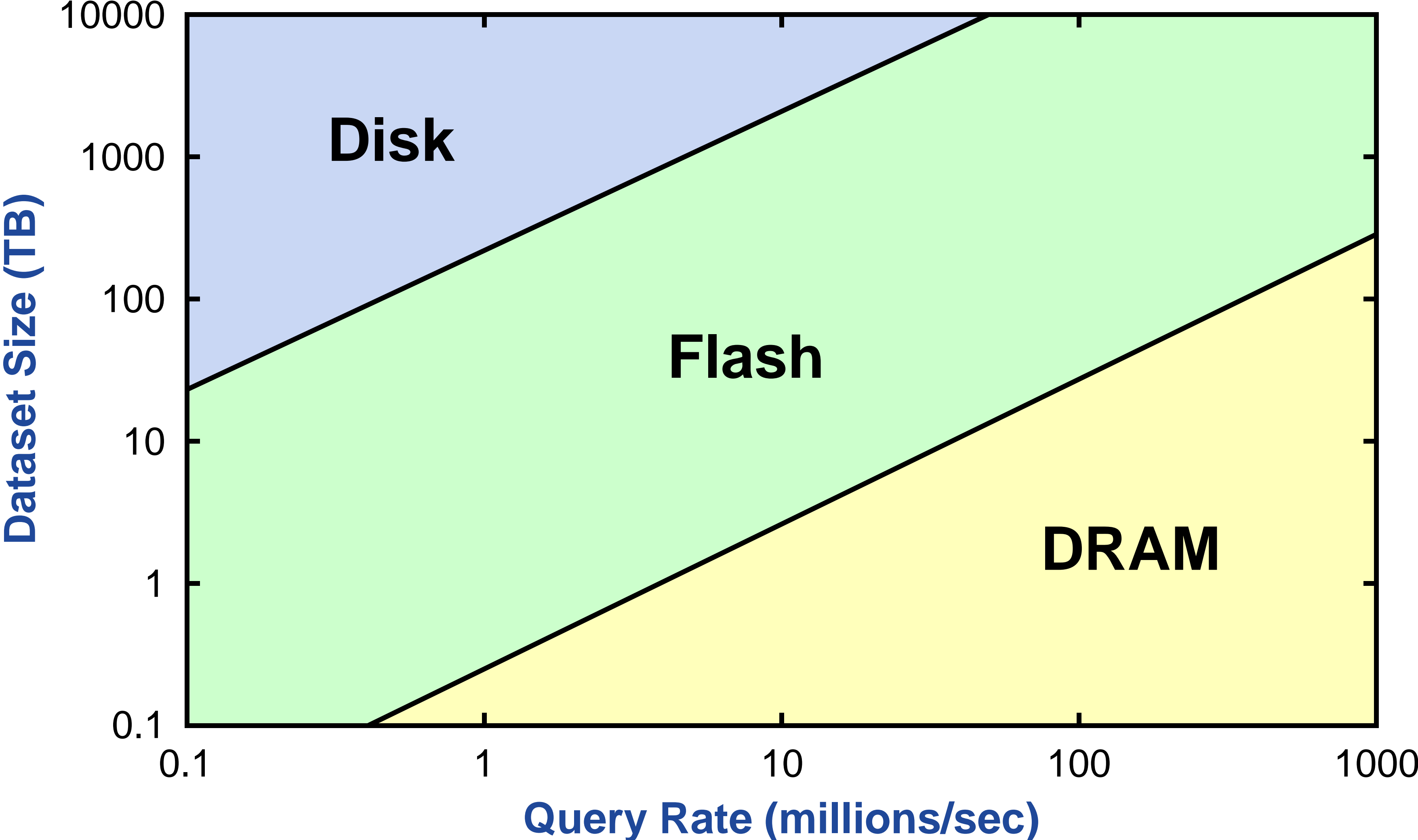
- **Performance measurements often wrong/counterintuitive**
- **Measure components of performance**
- **Understand *why* performance is what it is**

Data Model Rationale



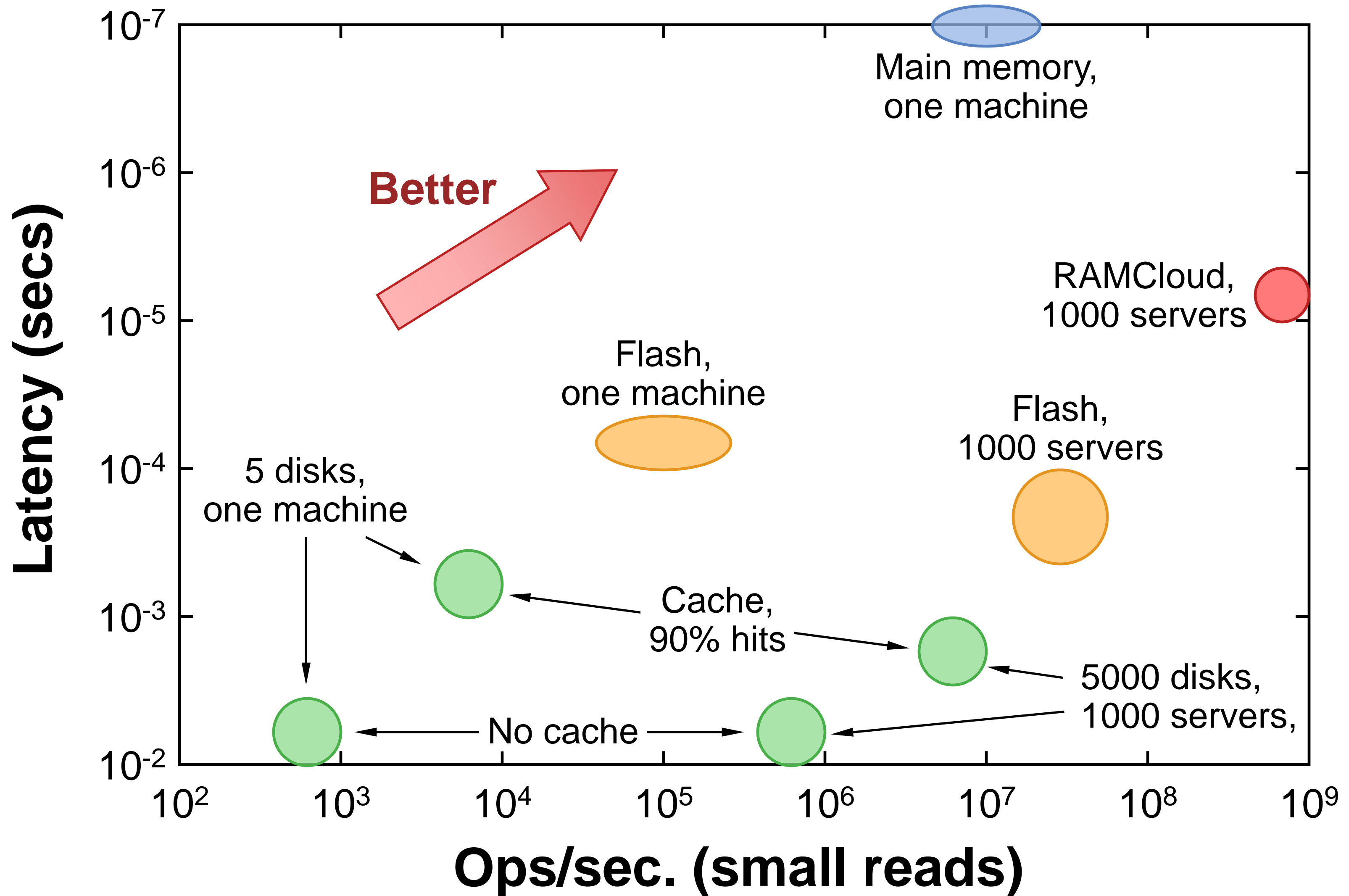
How to get best application-level performance?

Lowest TCO

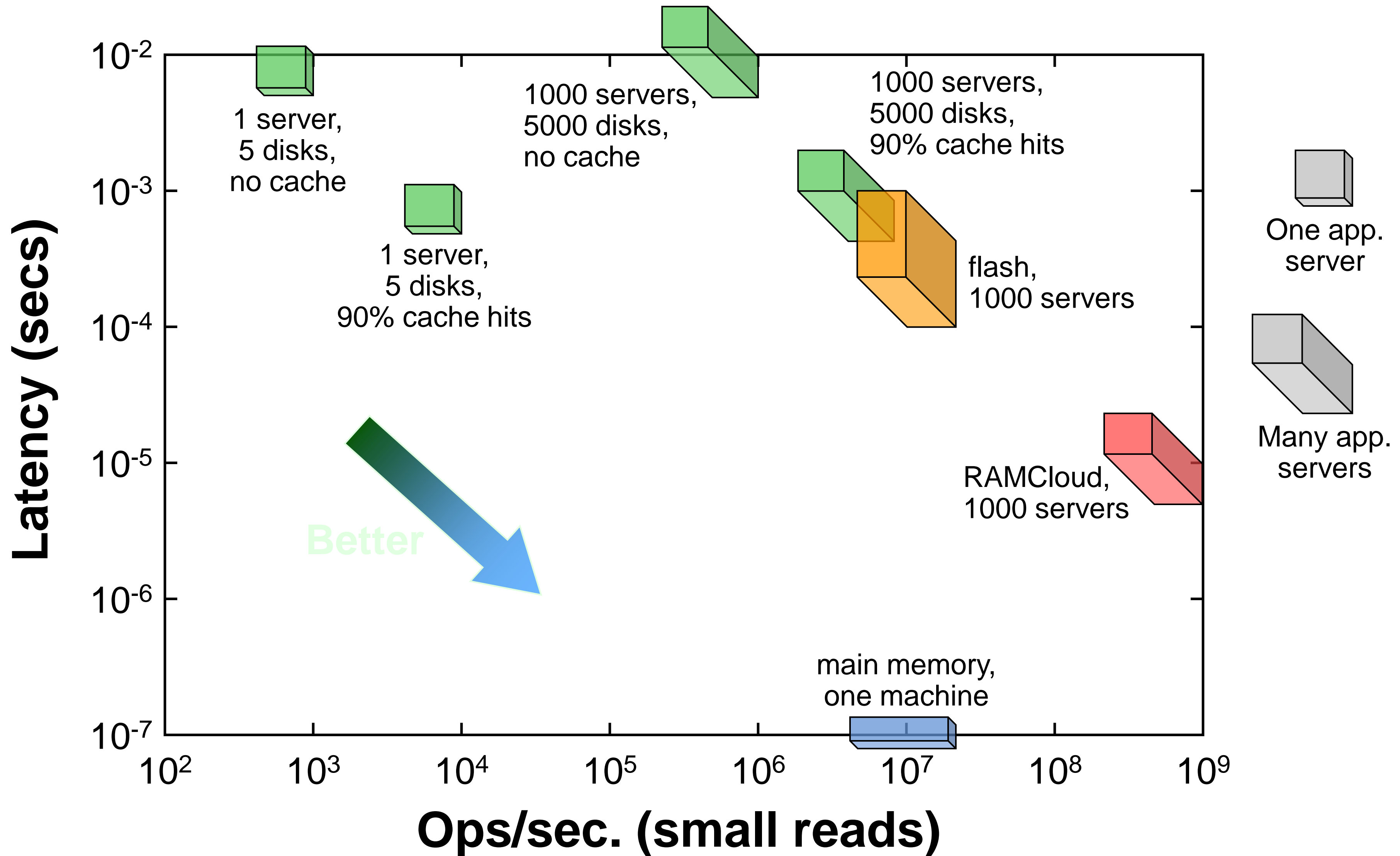


from "Andersen et al., "FAWN: A Fast Array of Wimpy Nodes", Proc. 22nd Symposium on Operating System Principles, 2009, pp. 1-14.

Latency vs. Ops/sec



Latency vs. Ops/sec



What We Have Learned From RAMCloud

John Ousterhout

Stanford University

(with Asaf Cidon, Ankita Kejriwal, Diego Ongaro, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang)

Introduction

A collection of broad conclusions we have reached during the RAMCloud project:

- Randomization plays a fundamental role in large-scale systems
- Need new paradigms for distributed, concurrent, fault-tolerant software
- Exciting opportunities in low-latency datacenter networking
- Layering conflicts with latency
- Don’t count on locality
- Scale can be your friend

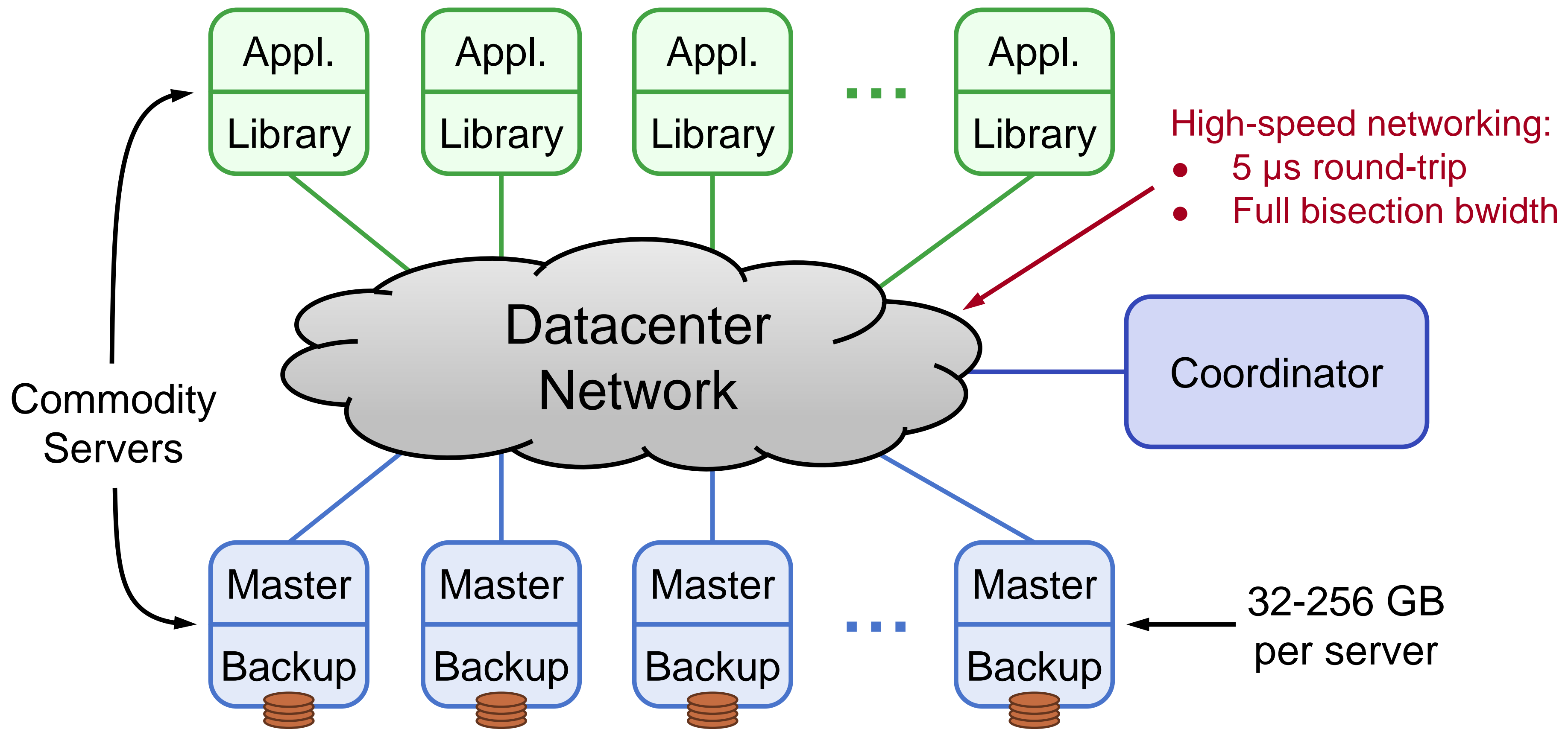
RAMCloud Overview

Harness full performance potential of large-scale DRAM storage:

- **General-purpose key-value 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

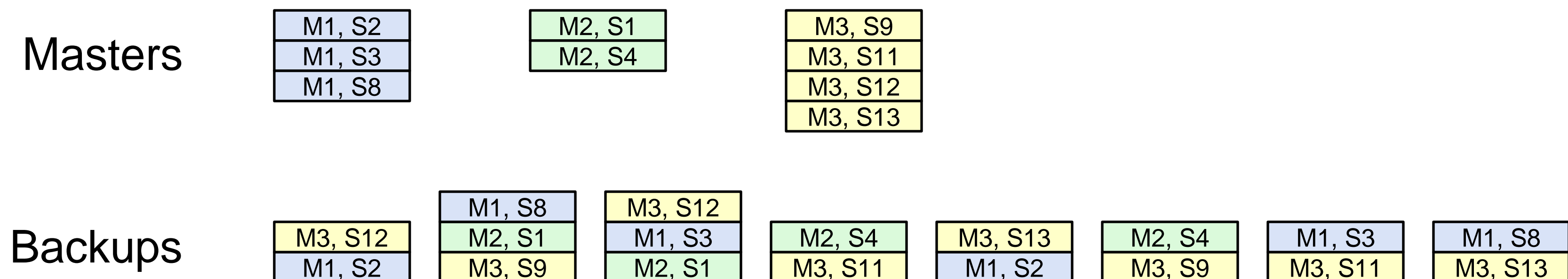
RAMCloud Architecture



Randomization

Randomization plays a fundamental role in large-scale systems

- **Enables decentralized decision-making**
- **Example: load balancing of segment replicas. Goals:**
  - Each master decides where to replicate its own segments: no central authority
  - Distribute each master's replicas uniformly across cluster
  - Uniform usage of secondary storage on backups

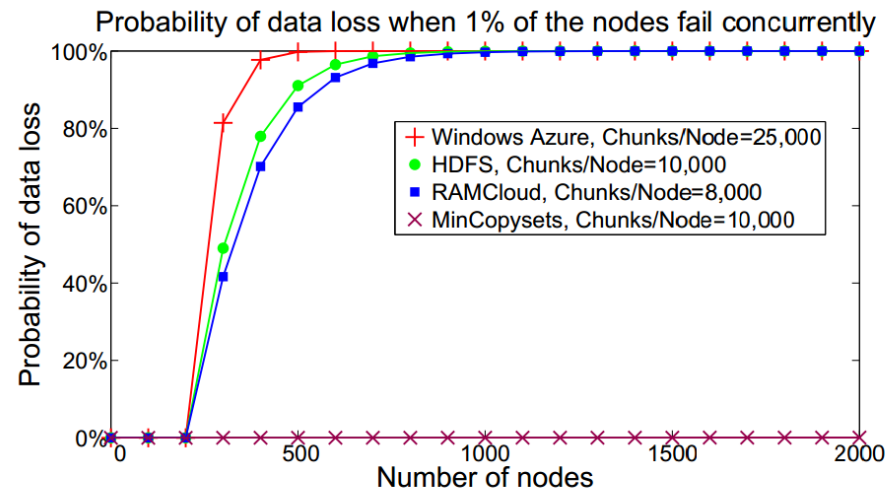


Randomization, cont'd

- **Choose backup for each replica at random?**
  - Uneven distribution: worst-case = 3-5x average
- **Use Mitzenmacher's approach:**
  - Probe several randomly selected backups
  - Choose most attractive
  - Result: almost uniform distribution

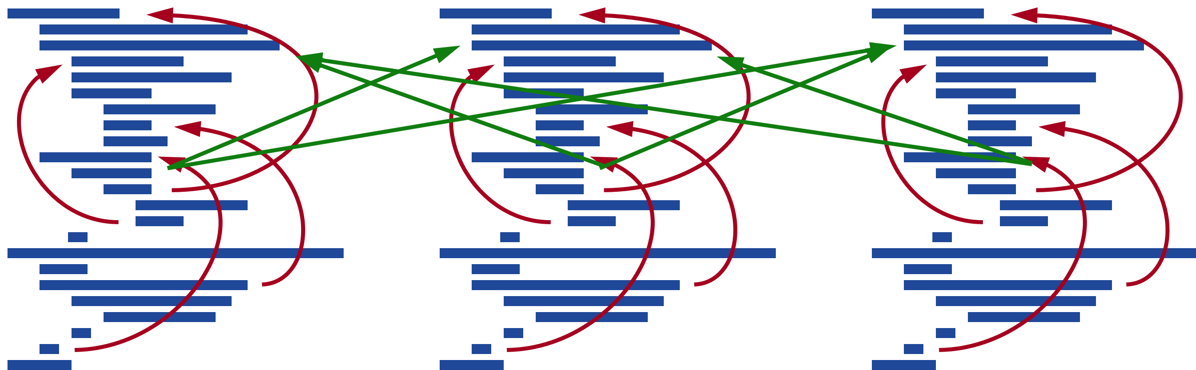
Sometimes Randomization is Bad!

- **Select 3 backups for segment at random?**
- **Problem:**
  - In large-scale system, any 3 machine failures results in data loss
  - After power outage, ~1% of servers don't restart
  - Every power outage loses data!
- **Solution: derandomize backup selection**
  - Pick first backup at random (for load balancing)
  - Other backups deterministic (replication groups)
  - Result: data safe for hundreds of years
  - (but, lose more data in each loss)



DCFT Code is Hard

- **RAMCloud often requires code that is distributed, concurrent, and fault tolerant:**
  - Replicate segment to 3 backups
  - Coordinate 100 masters working together to recover failed server
  - Concurrently read segments from ~1000 backups, replay log entries, re-replicate to other backups
- **Traditional imperative programming doesn't work**

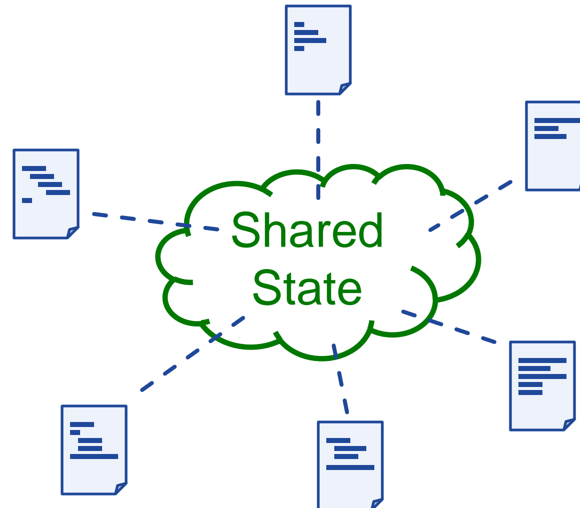


Must "go back" after failures

- **Result: spaghetti code, brittle, buggy**

DCFT Code: Need New Pattern

- **Experimenting with new approach: more like a state machine**



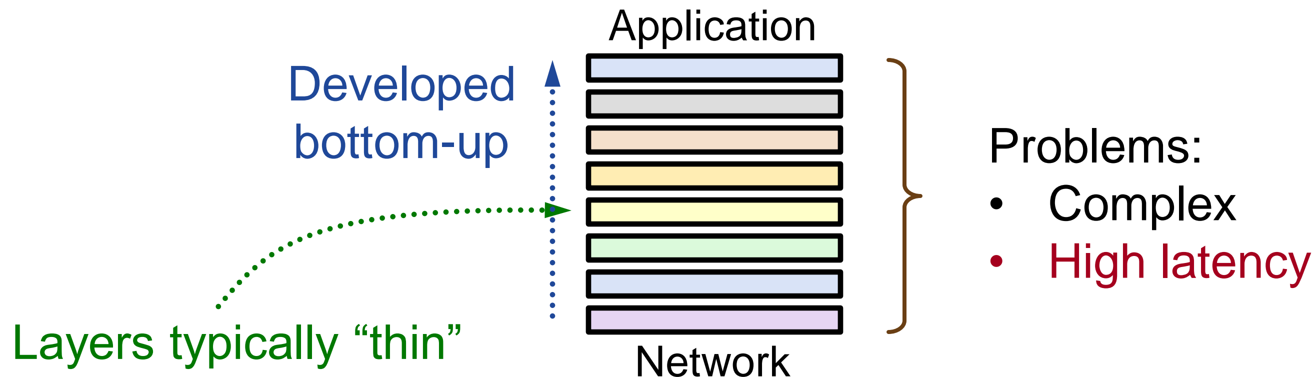
- **Code divided into smaller units**
  - Each unit handles one invariant or transition
  - Event driven (sort of)
  - Serialized access to shared state
- **These ideas are still evolving**

Low-Latency Networking

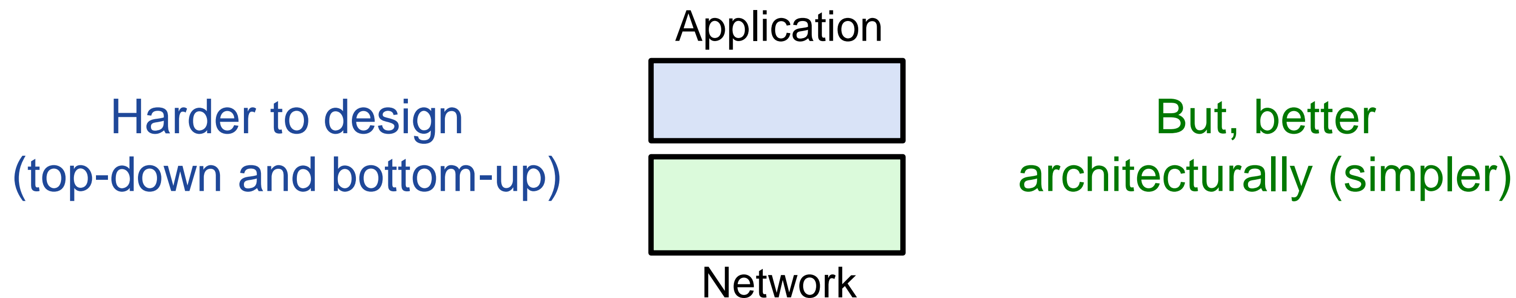
- **Datacenter evolution, phase #1: scale**
- **Datacenter evolution, phase #2: latency**
  - Typical round-trip in 2010: 300µs
  - Feasible today: 5-10µs
  - Ultimate limit: < 2µs
- **No fundamental technological obstacles, but need new architectures:**
  - Must bypass OS kernel
  - New integration of NIC into CPU
  - New datacenter network architectures (no buffers!)
  - New network/RPC protocols: user-level, scale, latency (1M clients/server?)

Layering Conflicts With Latency

Most obvious way to build software: lots of layers

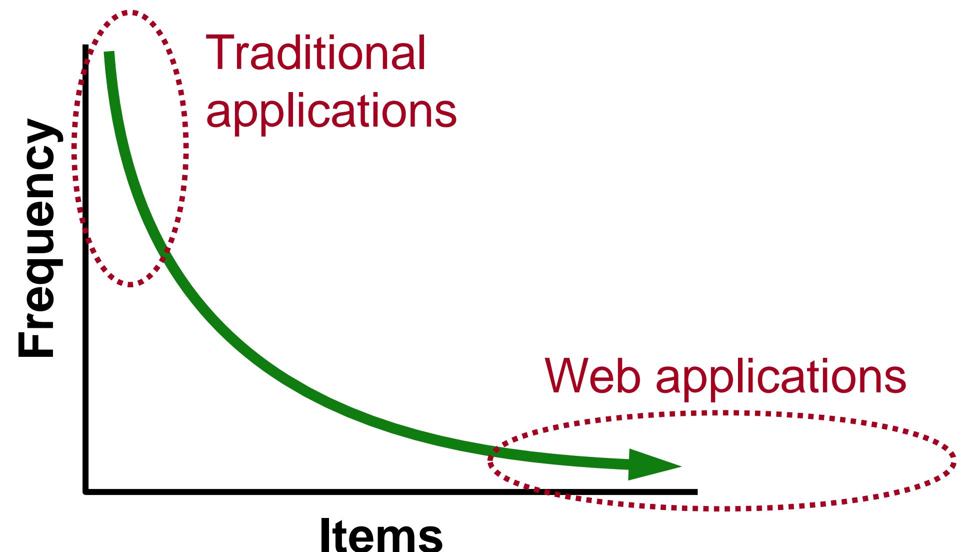


For low latency, must rearchitect with fewer layers



Don’t Count On Locality

- **Greatest drivers for software and hardware systems over last 30 years:**
  - Moore’s Law
  - Locality (caching, de-dup, rack organization, etc. etc.)
- **Large-scale Web applications have huge datasets but less locality**
  - Long tail
  - Highly interconnected (social graphs)



Make Scale Your Friend

- **Large-scale systems create many problems:**
  - Manual management doesn't work
  - Reliability is much harder to achieve
  - "Rare" corner cases happen frequently
- **However, scale can be friend as well as enemy:**
  - RAMCloud fast crash recovery
    - Use 1000's of servers to recover failed masters quickly
    - Since crash recovery is fast, "promote" all errors to server crashes
  - Windows error reporting (Microsoft)
    - Automated bug reporting
    - Statistics identify most important bugs
    - Correlations identify buggy device drivers

Conclusion

Build big => learn big

- **My pet peeve: too much “summer project research”**
  - 2-3 month projects
  - Motivated by conference paper deadlines
  - Superficial, not much deep learning
- **Trying to build a large system that really works is hard, but intellectually rewarding:**
  - Exposes interesting side issues
  - Important problems identify themselves (recurrences)
  - Deeper evaluation (real use cases)
  - Shared goal creates teamwork, intellectual exchange
  - Overall, deep learning