Staggeringly Large File Systems

Presented by Haoyan Geng

Large-scale File Systems

How Large?

- Google's file system in 2009 (Jeff Dean, LADIS '09)
  - 200+ clusters
  - Thousands of machines per cluster
  - Pools of thousands of clients
  - More than 4 Petabytes of data
  - 40 GB/s I/O load

- Amazon's S3 in 2010 (Werner Vogels' blog)
  - Over 100 billion objects
  - Over 120,000 storage operations per second

“Large” can be different

- Range of the network
  - Wide/Local area
- Organizing model
  - Centralized/P2P/Layered ...
- Environment
  - Trusted/Untrusted infrastructure
- Abundance of resources
  - Bandwidth, storage space, ...
- The goals:
  - Availability, reliability, scalability, ...

“Large” can be different

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

Pond: the OceanStore Prototype*

Sean Rhea, Patrick Eaton, Dennis Geels,
Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz
University of California, Berkeley
{*srhea,eaton,geels,hweather,ravenben,kubitron*}*@cs.berkeley.edu*

“Large” can be different

	GFS	OceanStore
Infrastructure	Datacenter	Wide-area
Organizing Model	Centralized	Fully distributed
Target Users	Google	Anyone
Environment	Trusted	Untrusted
Availability	High	High
Reliability	High	High
Recovery	Self-maintaining	Self-maintaining

Google File System

The Authors

- Sanjay Ghemawat
  - Google Fellow, worked on GFS, MapReduce, BigTable, ...
  - PhD from MIT (Barbara Liskov)
- Howard Gobioff
  - PhD from CMU (Garth Gibson)
- Shun-Tak Leung
  - PhD from UW

Motivation

- A file system for Google's infrastructure
  - Clusters of commodity PCs
  - Component failures are common
    - Program bugs (System/Apps)
    - Storage failures (Memory/Disk)
    - Network failure
    - Power outage
    - Other human errors ...

Motivation

- A file system for Google's infrastructure
  - Clusters of commodity PCs
  - Component failures are common
  - Trusted environment
    - Malicious users? Not a big problem ...

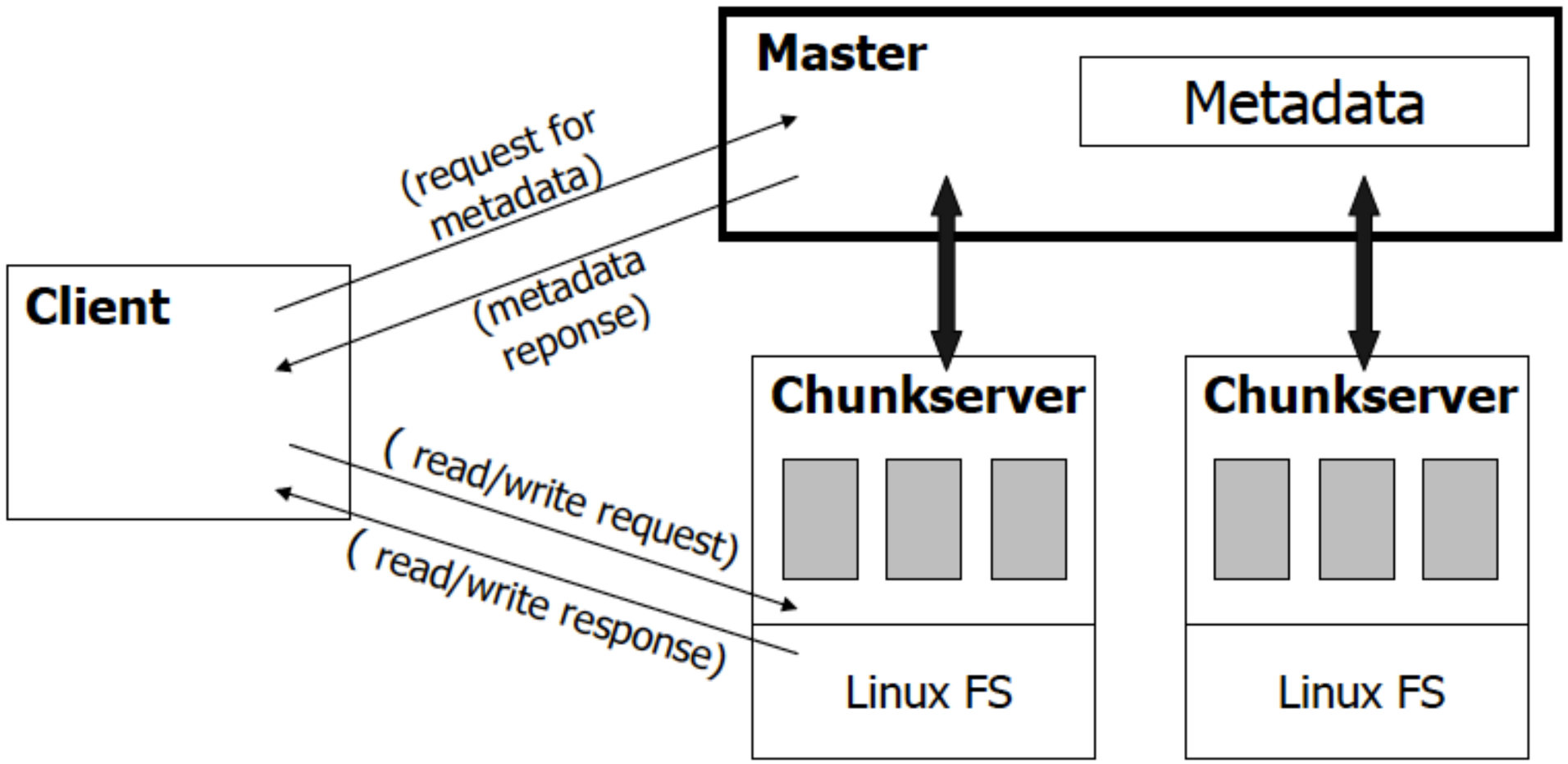
Motivation

- A file system for Google's infrastructure
  - Clusters of commodity PCs
  - Component failures are common
  - Trusted environment
  - Files are huge
  - Most mutations are append

Design Choices

- Fault tolerant
  - ... but no need for BFT
- Fast failure recovery
- Relaxed consistency guarantee
- Selective optimization (block size, append, ...)
  - "Small files must be supported, but we need not optimize for them."
  - "Small writes at arbitrary positions in a file are supported but do not have to be efficient."
- Co-design with applications

Architecture



Architecture

- Interface
- Chunkserver
- Master
- Mutation

Interface

- Standard operations
  - Create, Delete, Open, Close, Read, Write
- Record append
- Snapshot

Chunkserver

- Runs on top of Linux file system
- Basic unit: Chunks
  - 64 MB each
  - A file in GFS consists of several chunks
- Each chunk is a Linux file
  - Eliminates fragmentation problems
- Replicated for reliability (3 by default)
- The primary: serializes mutations

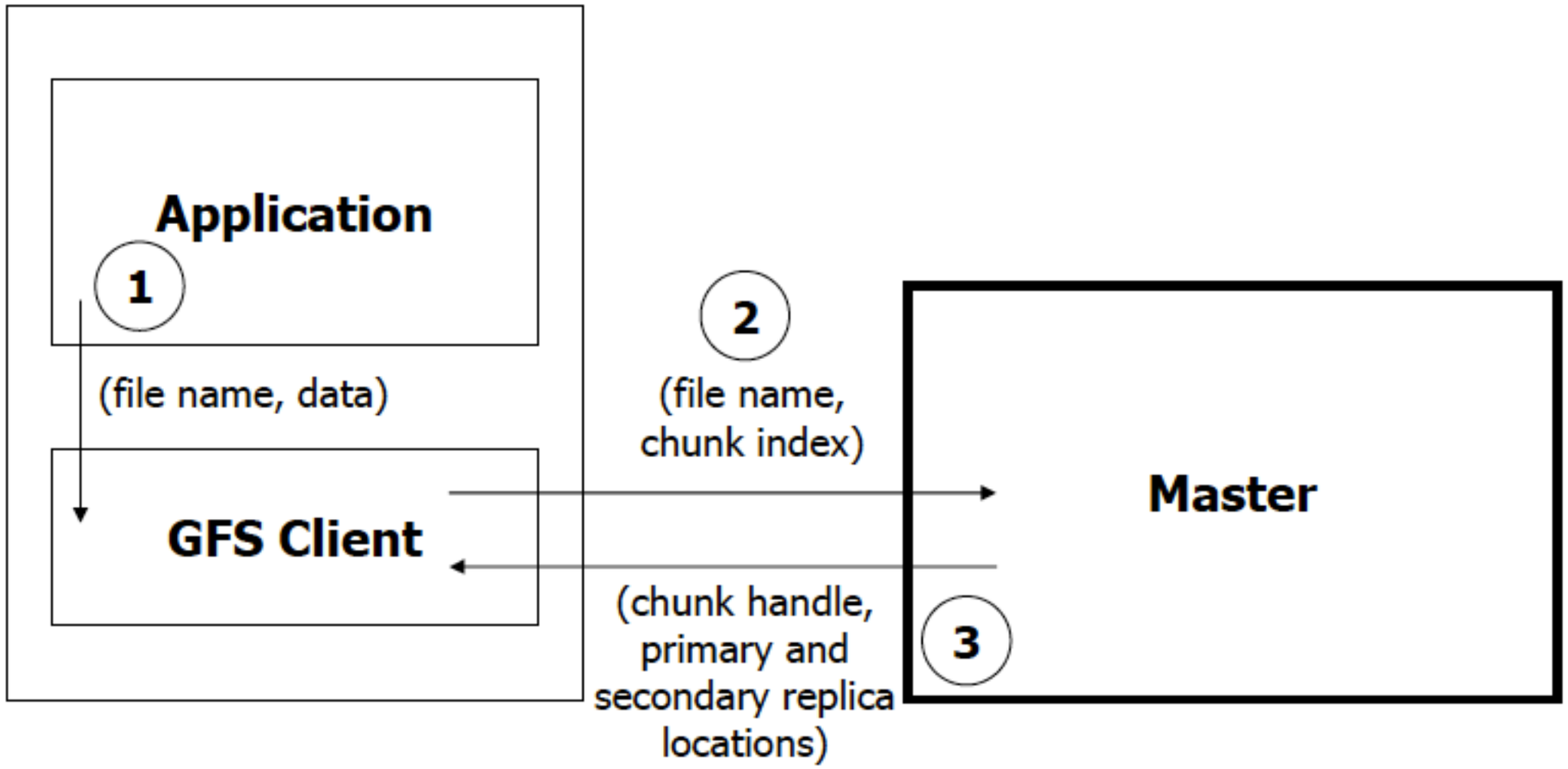
Master

- Controls system-wide activities
  - Leases, replica placement, garbage collection, ...
- Stores all metadata in memory, 64 bytes / chunk
  - File/chunk namespaces
  - File-to-chunk mapping
  - Replica locations
- The good part: simple, fast
- Not so good?
  - Throughput
  - Single point of failure

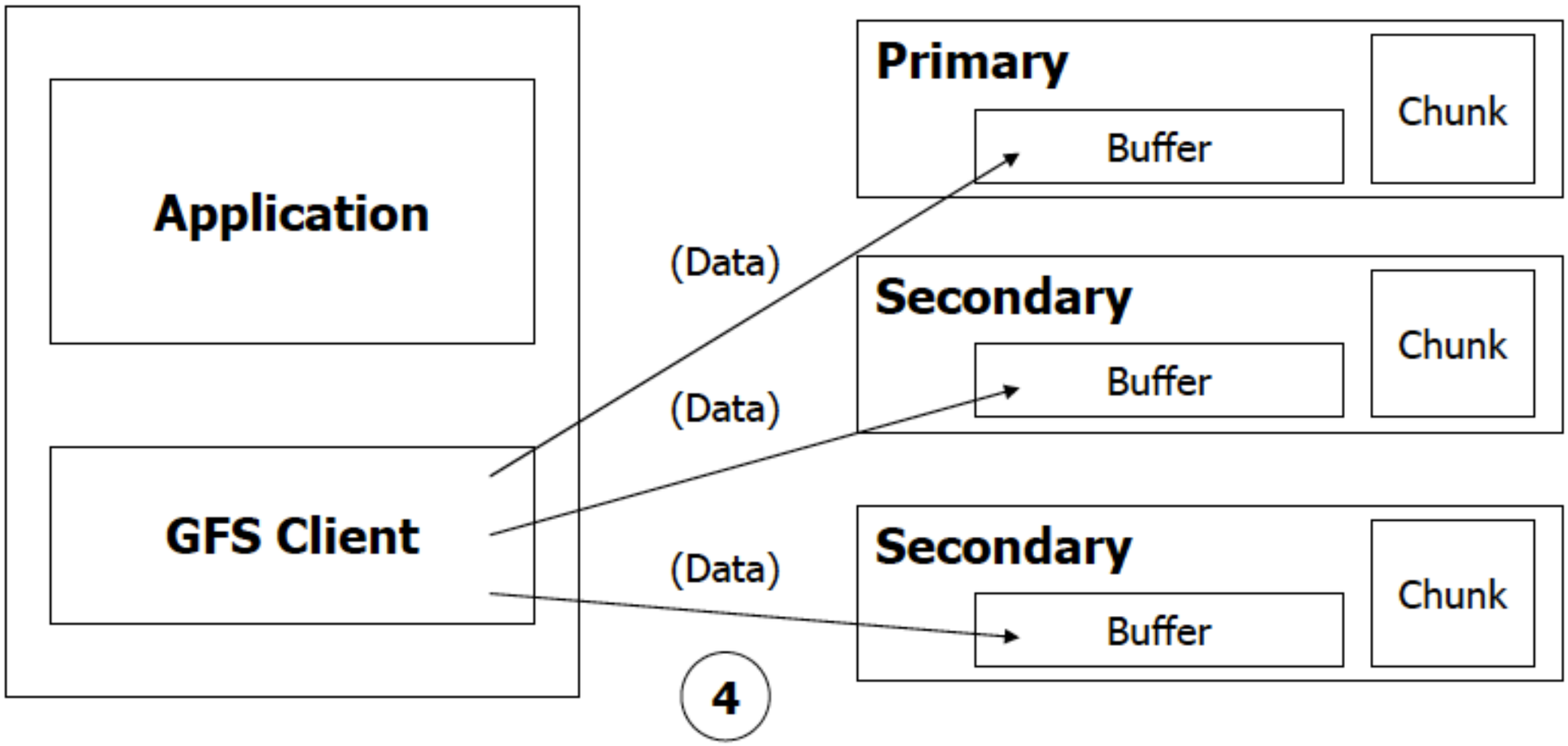
More on Master

- Optimizations
  - Throughput
    - Minimize operations at master
    - Fast recovery using logs & checkpoints
  - Single point of failure
    - Replicates itself
    - “Shadow” masters

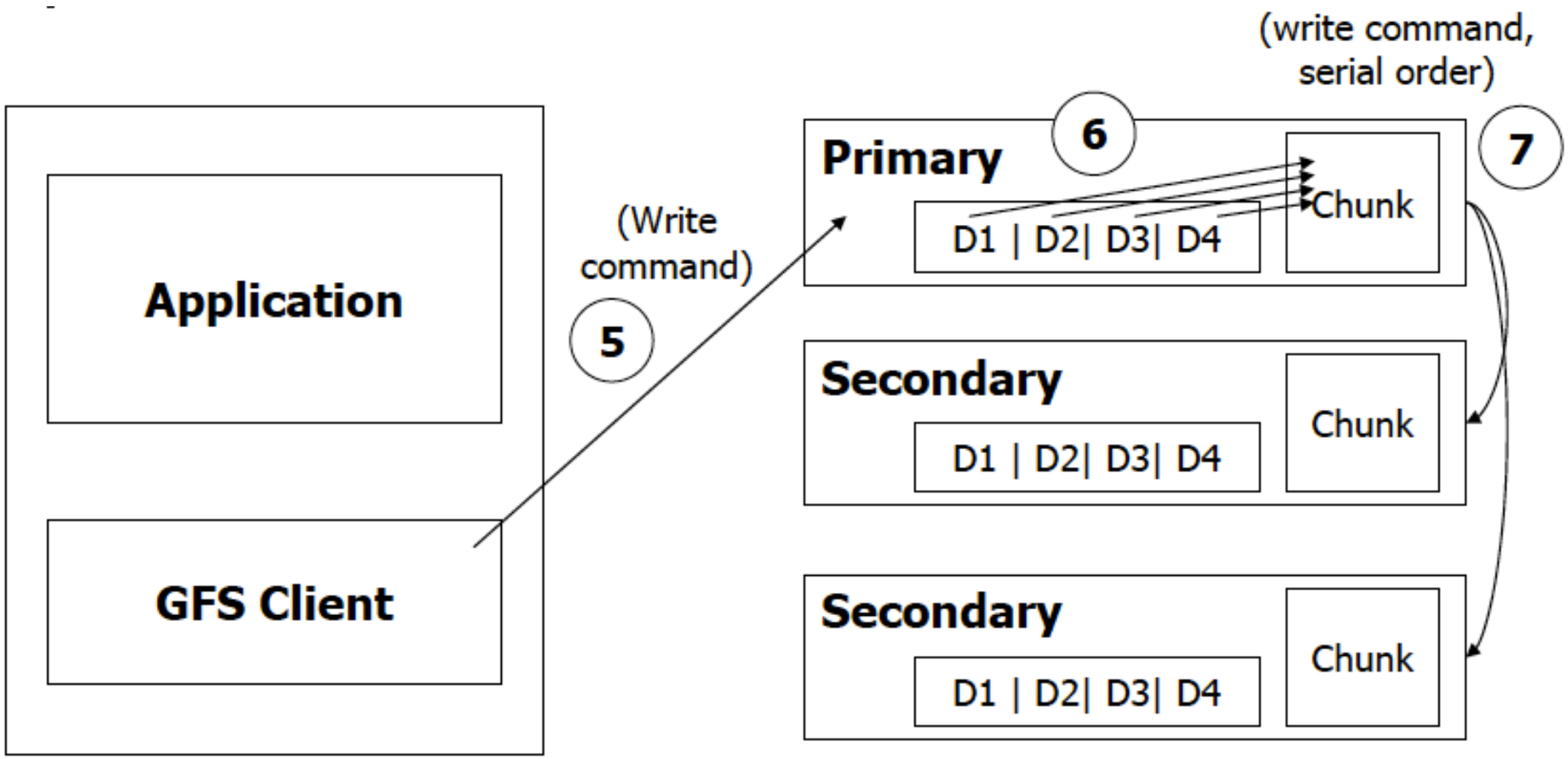
Mutation


Application
1
(file name, data)
GFS Client
2
(file name,
chunk index)
(chunk handle,
primary and
secondary replica
locations)
3
Master

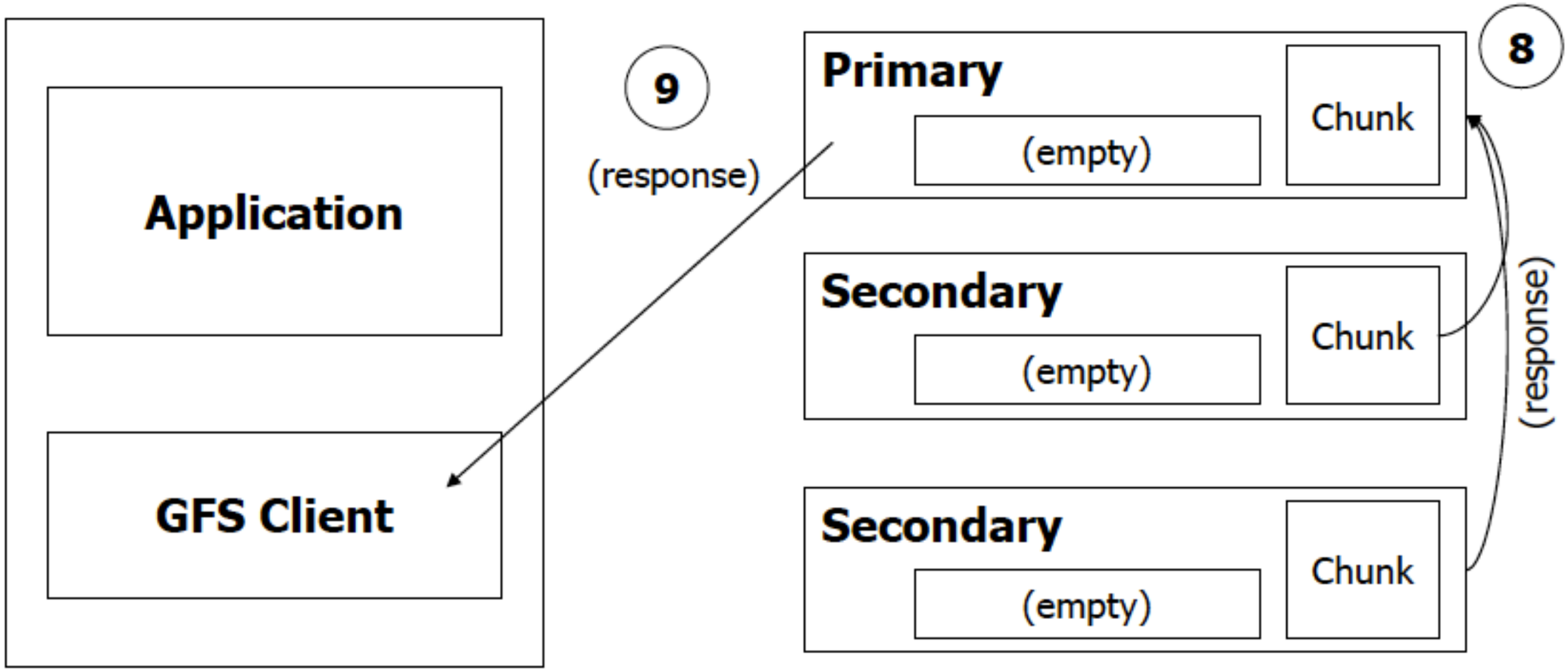
Mutation


Application
GFS Client
(Data)
(Data)
(Data)
Primary
Buffer
Chunk
Secondary
Buffer
Chunk
Secondary
Buffer
Chunk
4

Mutation


(write command, serial order)
Application
GFS Client
(Write command)
5
6
7
Primary
D1 | D2| D3| D4
Chunk
Secondary
D1 | D2| D3| D4
Chunk
Secondary
D1 | D2| D3| D4
Chunk

Mutation


Application
GFS Client
9
(response)
Primary
(empty)
Chunk
8
Secondary
(empty)
Chunk
Secondary
(empty)
Chunk
(response)

A Few Issues

- Consistency Guarantee
- Data Integrity
- Replica Placement
- File Deletion

Consistency Guarantee

- Consistent: all clients will always see the same data
- Defined: consistent and mutated in entirety

- Semantics of (concurrent) Write/Append
  - Write: Consistent but undefined
  - Append: Defined, interspersed with inconsistent - “Atomically at least once”

Data Integrity

- Checksum
  - 32-bit for each 64-KB block
  - Independent verification, not chunk-wise comparison
    - Recall the semantics of Append

Replica Placement

- Below-average utilization
  - Load balancing
- Distributes “recent” creations
  - Avoid hot spot
- Places replicas across racks
  - Reliability
  - Bandwidth utilization

File Deletion

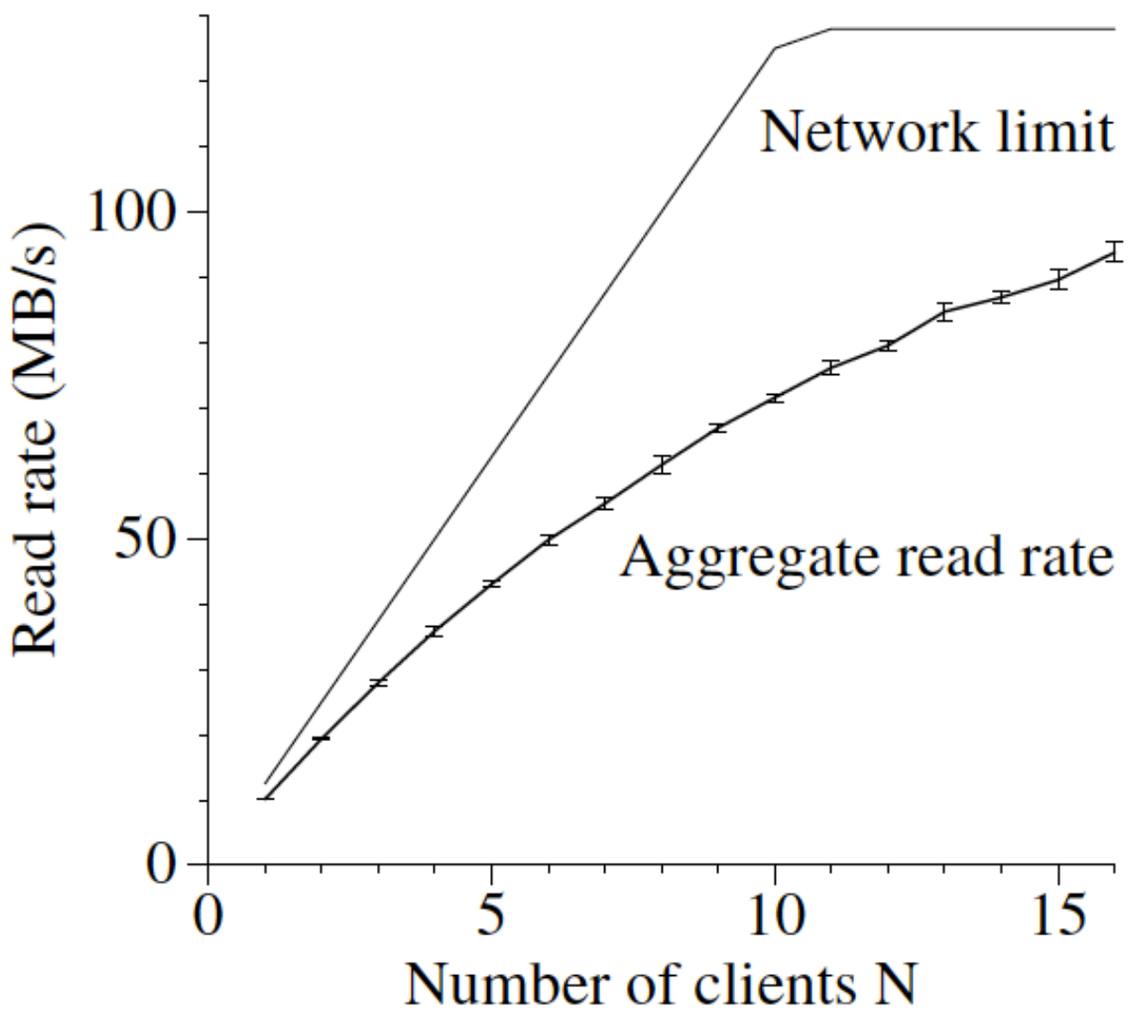
- Rename to hidden file upon deletion
- Permanently remove after 3 days
- Garbage collection: scan periodically to remove orphan chunks

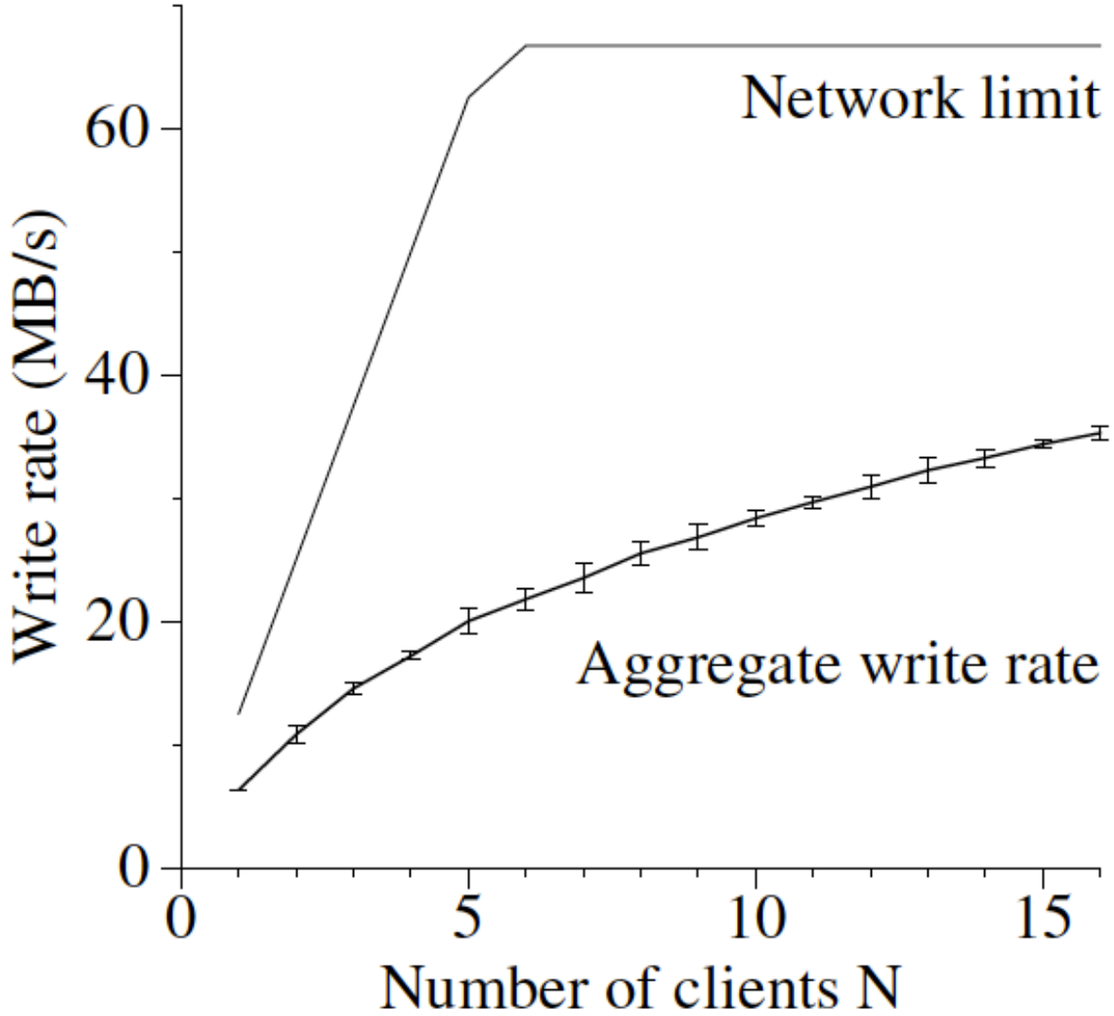
Performance

- Micro-benchmark
- Google's working environment

Performance

Micro-benchmark

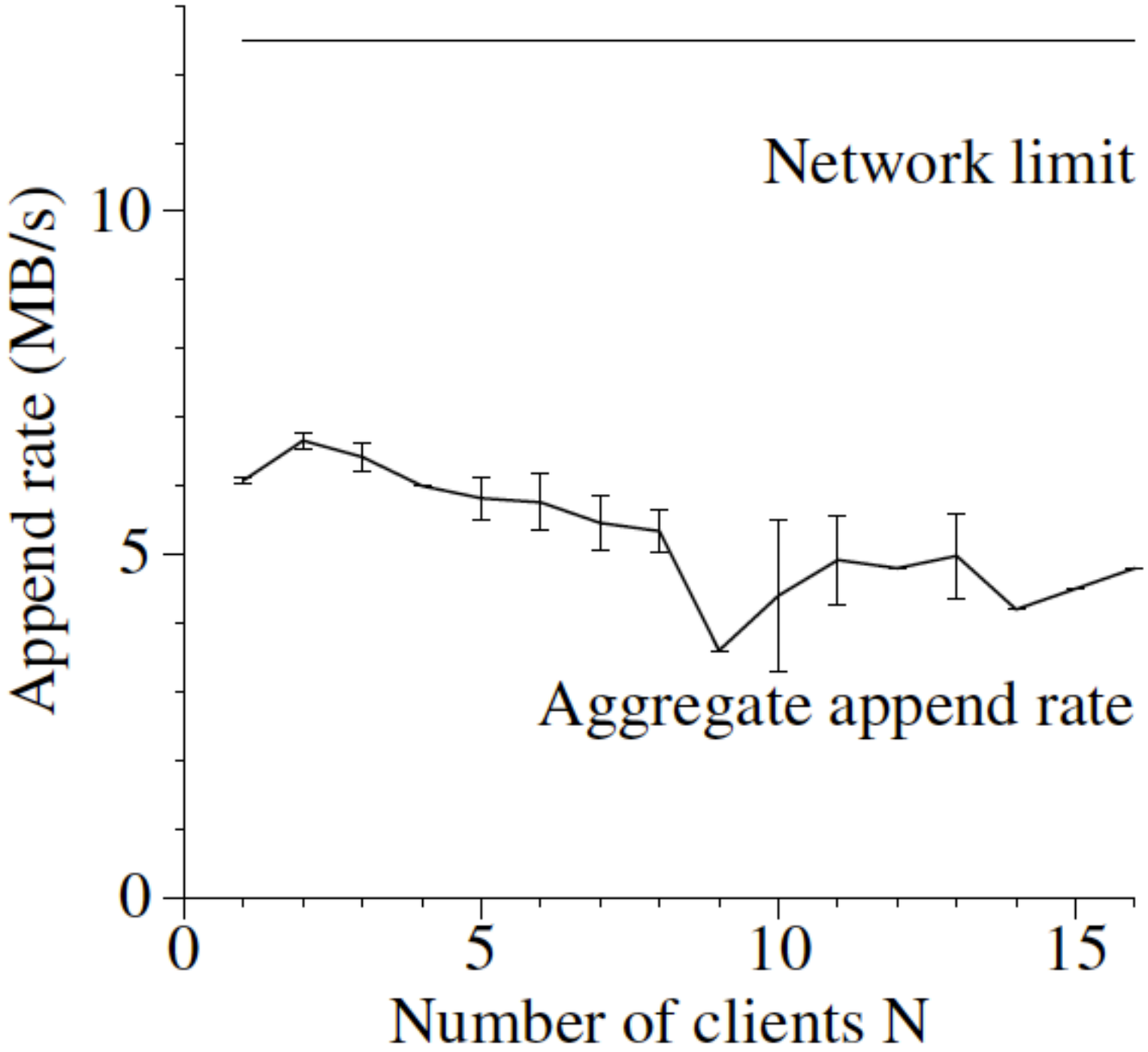




- 16 Chunkservers, up to 16 clients
- Links: 100 Mbps host-switch, 1 Gbps inter-switch

Performance

Micro-benchmark



Performance

Clusters

Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

- 300~500 operations per second on master
- 300~500 MB/s read, 100 MB/s write

Discussion

- A big engineering success
  - Anything new?
- Highly specialized design
- Comparison with others?

- Questions?

Pond: the OceanStore Prototype

The Authors

- Sean Rhea - PhD student → Meraki
- Patrick Eaton - PhD student → EMC
- Dennis Geels - PhD student → Google
- Hakim Weatherspoon - PhD student → ... here!
- Ben Zhao - PhD student → UCSB
- John Kubiatowicz - The professor
  - PhD from MIT (Anant Agarwal)

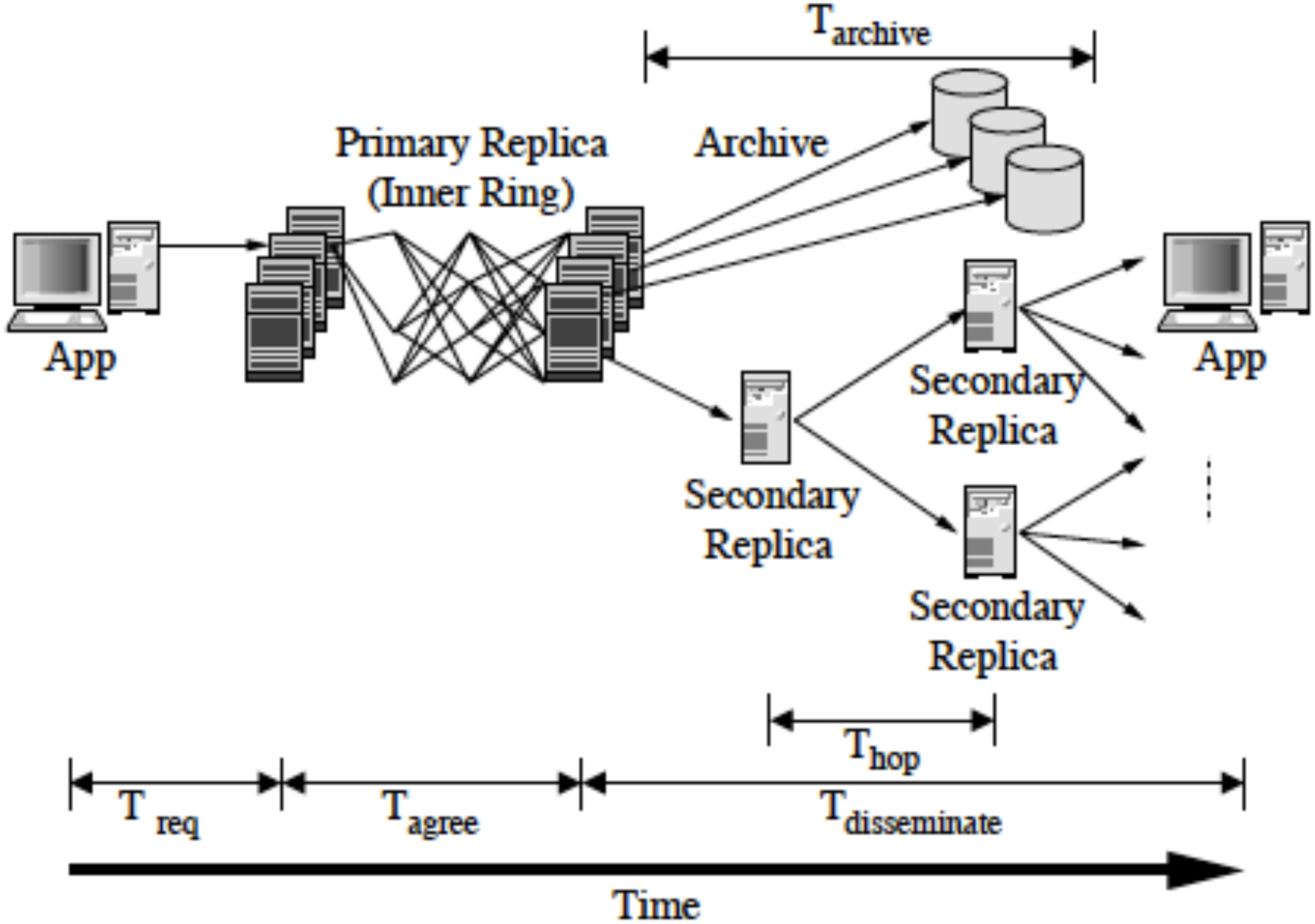
OceanStore's Vision

- Universal availability
- Transparency
- High durability
- Decentralized storage
- Self-maintainable
- Strong consistency
- Strong security
- Untrusted infrastructure

"As a rough estimate, we imagine providing service to roughly 10^{10} users, each with at least 10,000 files. OceanStore must therefore support over 10^{14} files."

-- The OceanStore paper (ASPLOS '00)

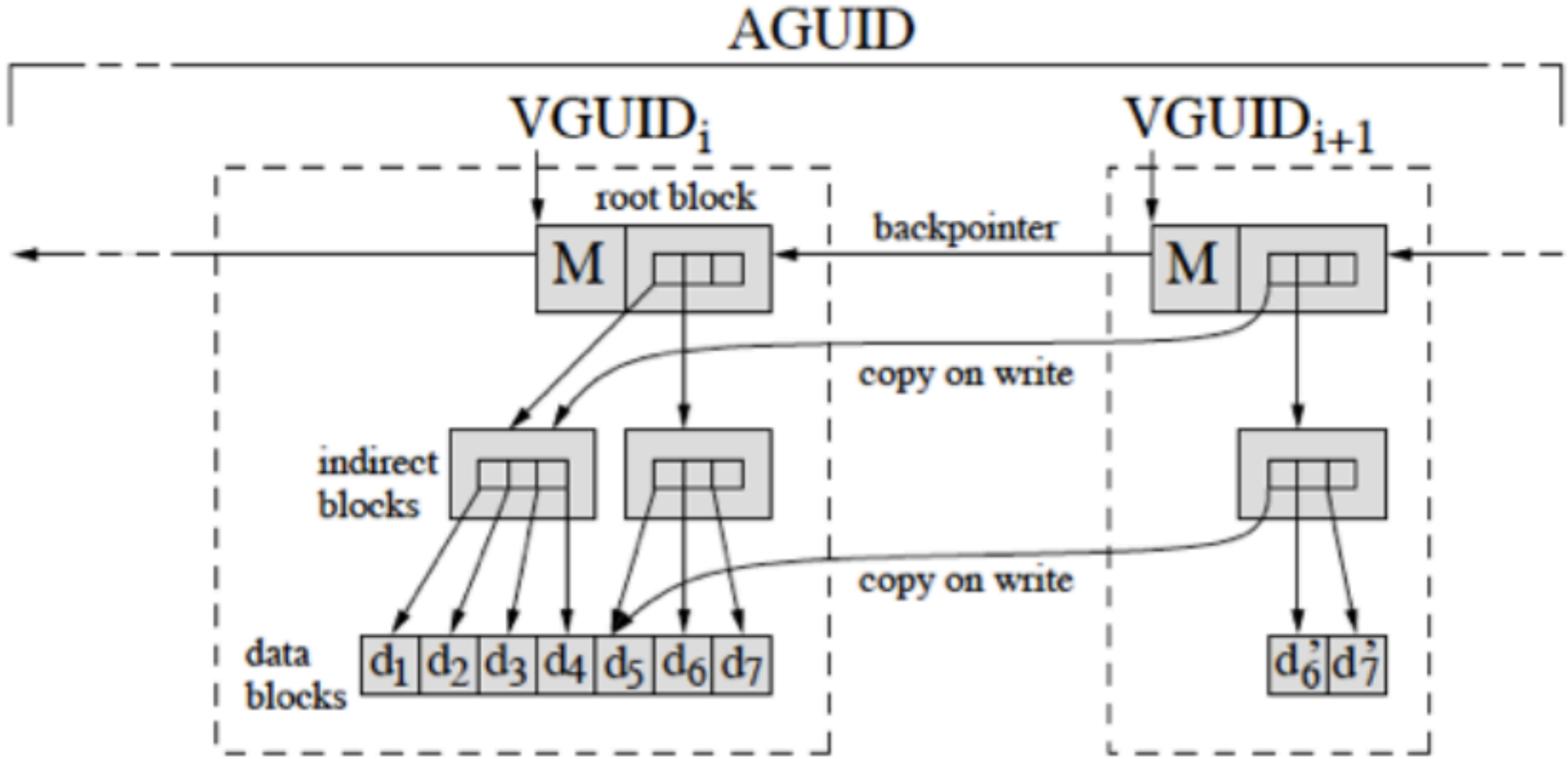
System Overview



Inside the System

- Data Object: "File" in Pond
- Tapestry: Distributed storage
- Primary Replica: Inner-ring
- Archival Storage: Erasure codes
- Secondary Replica: Caching

Data Object



- B-Tree of read-only blocks
- Every version is kept forever
- Copy on write

Tapestry

- DHTs & Storage systems
  - Chord - Cooperative File System (MIT)
  - Pastry - PAST (Microsoft & Rice)
  - Tapestry - OceanStore (Berkeley)
- Distributed Object Location and Routing
- Uses GUID to locate objects
- Self-organizing
- Tapestry's locality: Locates the closest replica

Primary Replica

- Serializes & applies changes
- A set of servers: inner ring
  - Assigned by the responsible party
- Byzantine-fault-tolerant protocol
  - Tolerates at most f faulty nodes with 3f+1 nodes
  - MAC within inner ring, public-key outside
  - Proactive threshold signatures
- How to durably store the order of updates?
  - Antiquity: Log-based method (Eurosys '07)

Archival Storage

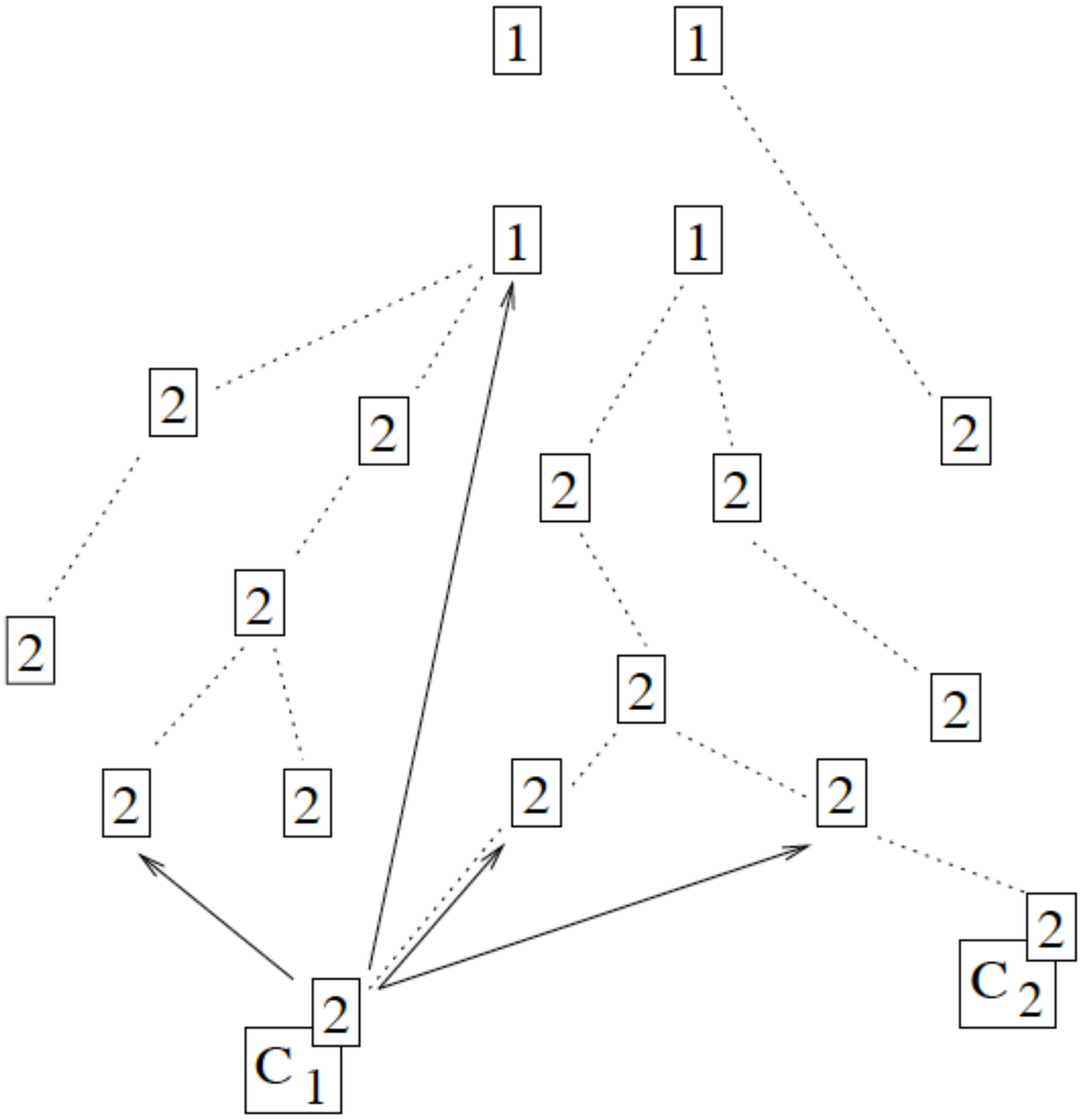
- Erasure code
  - Divide a block into m fragments
  - Encode into n (n > m) fragments
  - Recover from any m fragment
  - Fragments are distributed uniformly and deterministically

  (n = 32, m = 16 in Pond)
- Distributes data across machines
- Efficiency: Erasure code vs. Replication

Secondary Replica (Caching)

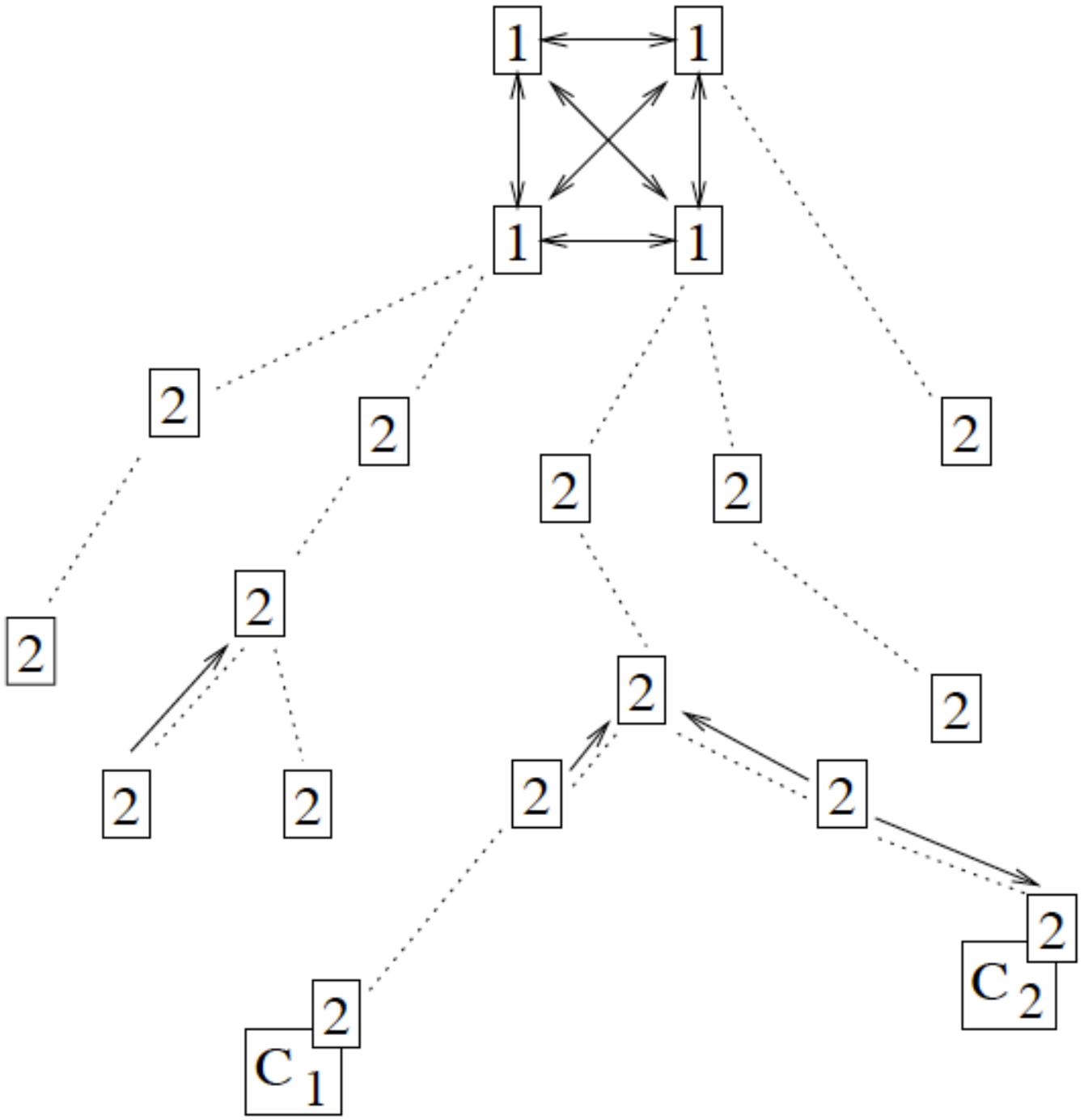
- Reconstruction from fragments is expensive
- An alternative: whole-block caching
- Greedy local caching: LRU in Pond
- Contact the primary for the latest version

Handling Updates



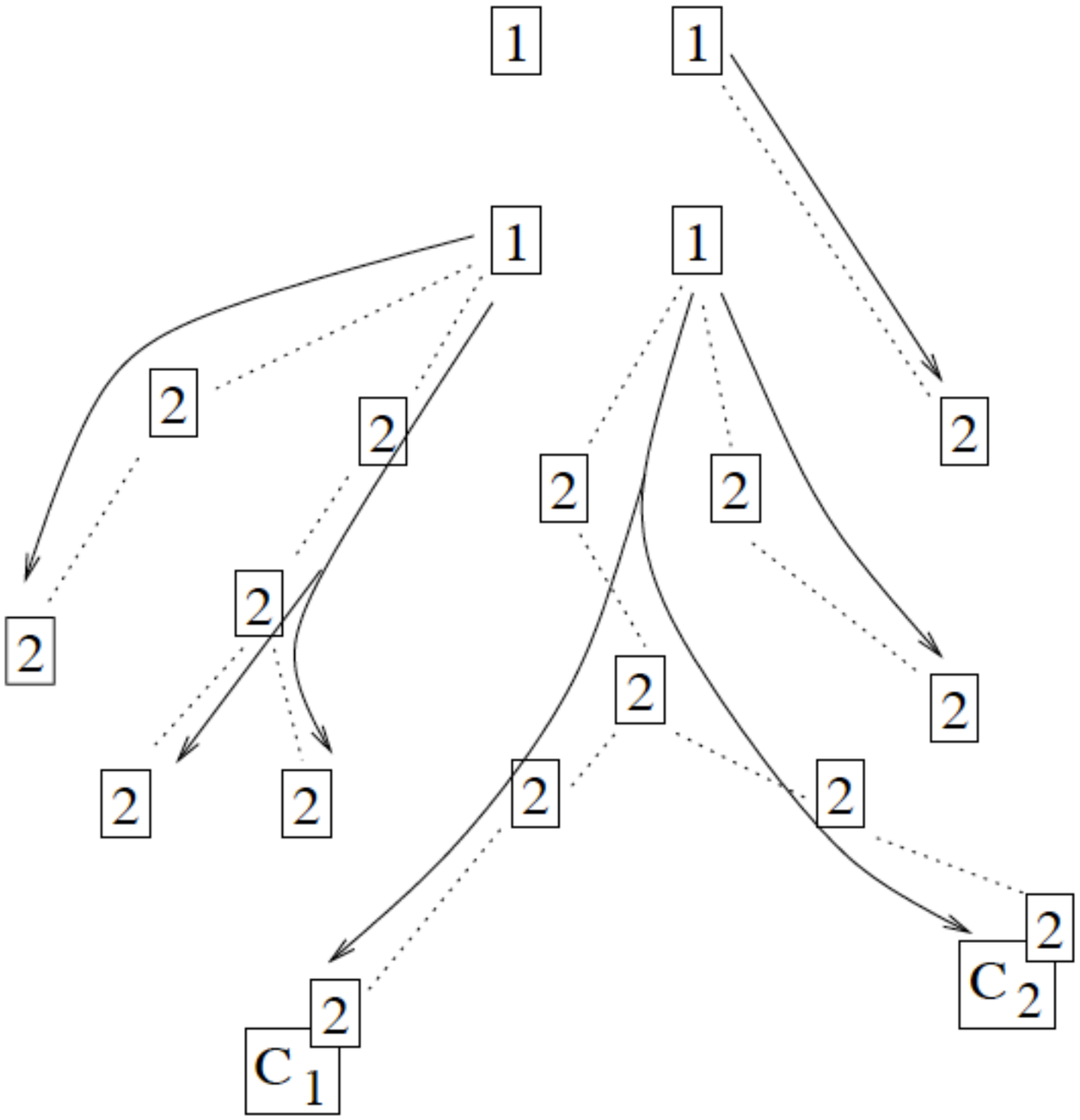
- Sends update to
- Primary tier
- Random replicas

Handling Updates



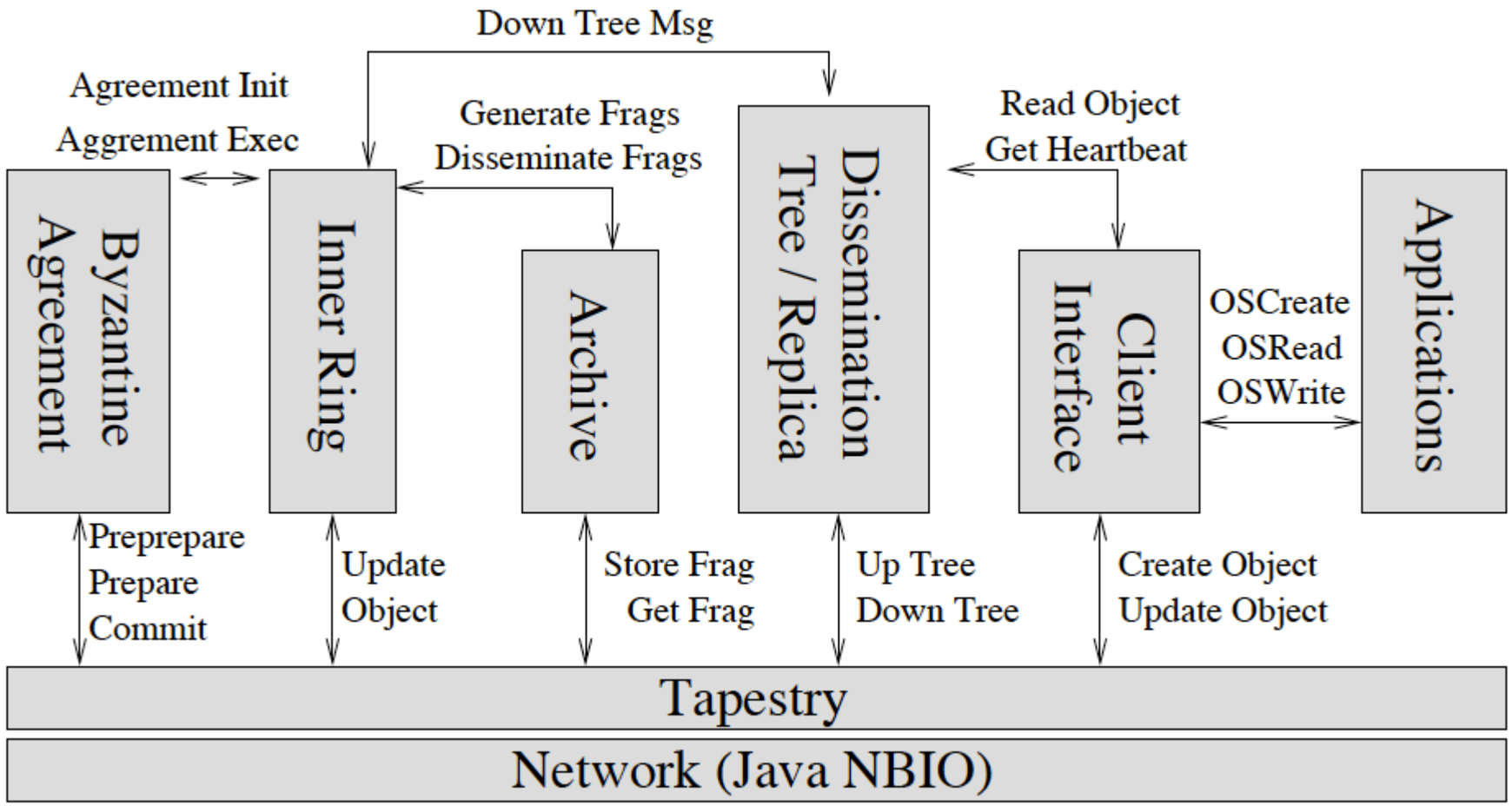
- Primary:
  - Byzantine agreement
- Secondary:
  - Epidemically propagation (Gossiping)

Handling Updates



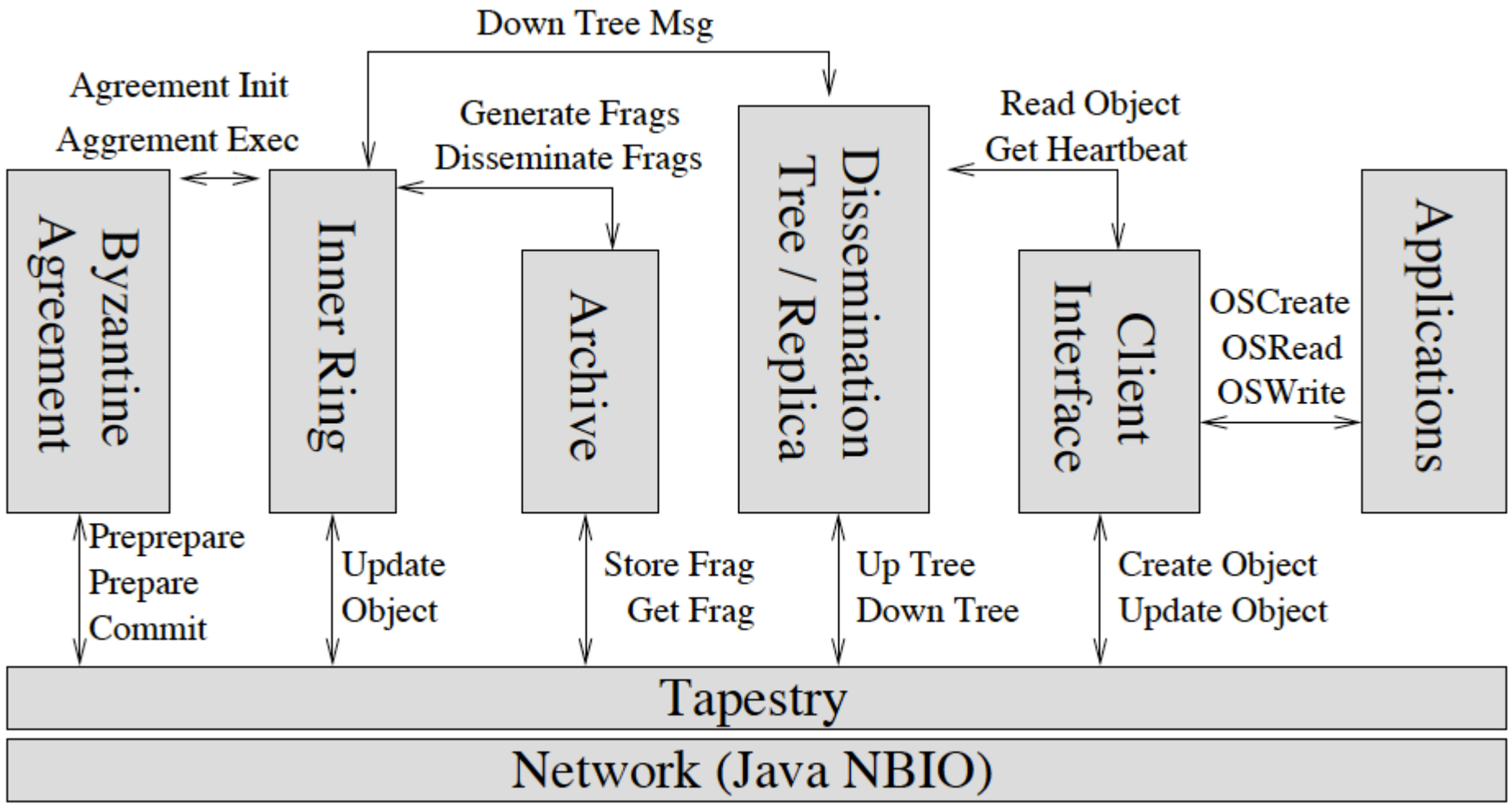
- Multicast from the primary

Implementation



- Based on SEDA
  - Implemented in Java
  - Event driven, each subsystem as a stage

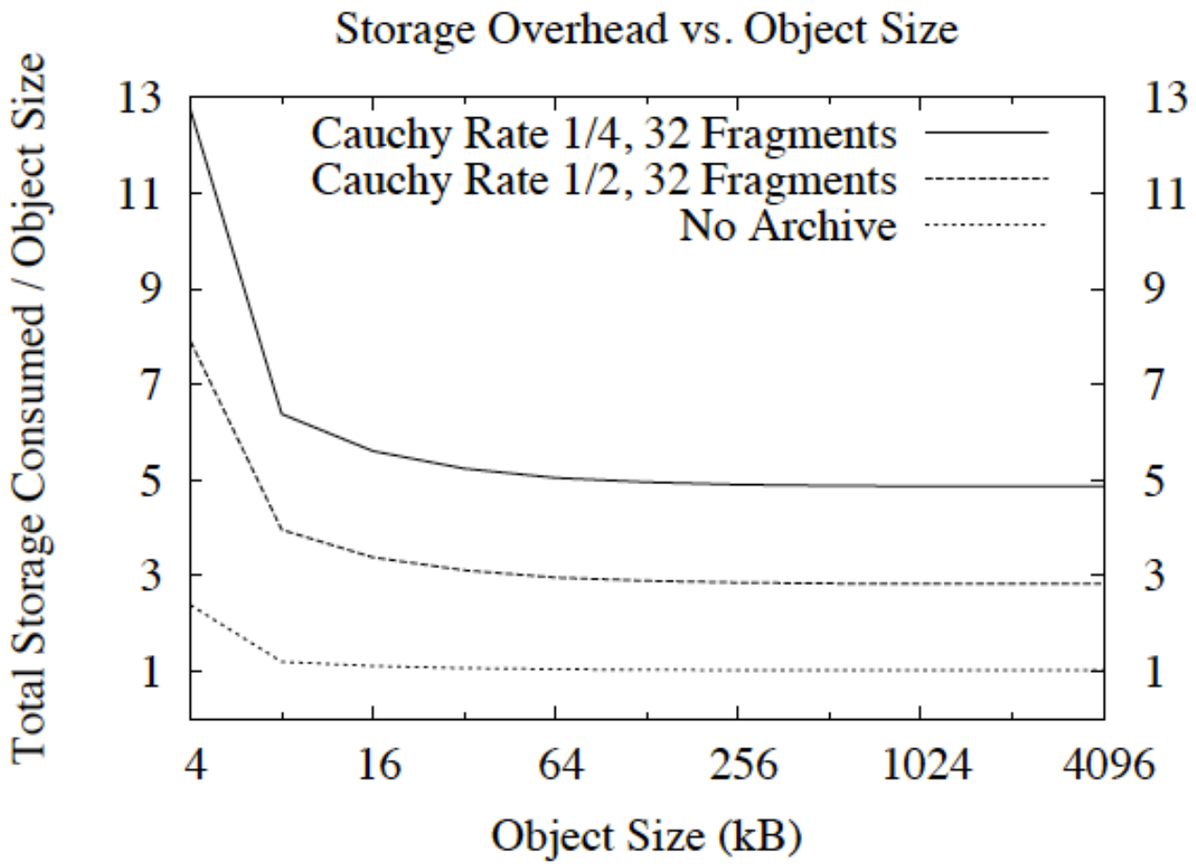
Implementation

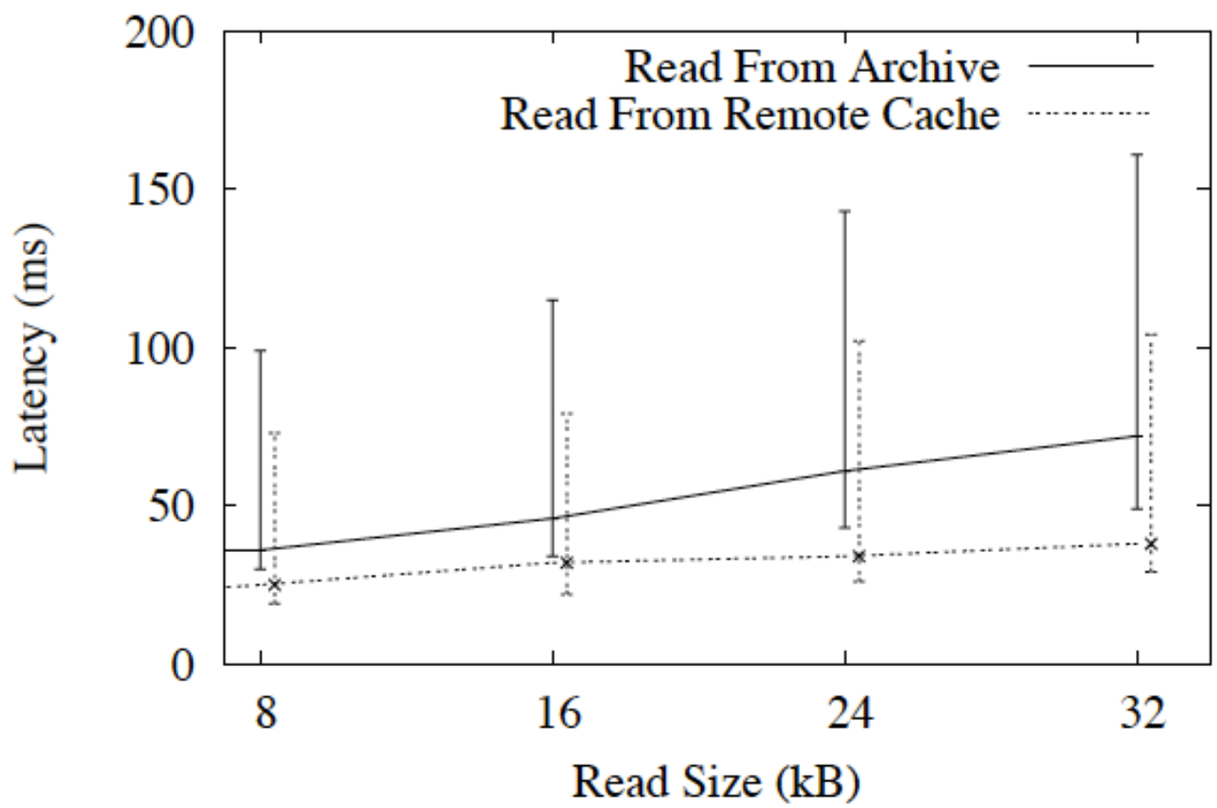


"The current code base of Pond contains approximately 50,000 semicolons and is the work of five core graduate student developers and as many undergraduate interns."

Performance

Archival Storage





Performance

Latency Breakdown

Phase	Time (ms)	
	4 kB Update	2 MB Update
Check Validity	0.3	0.4
Serialize	6.1	26.6
Update	1.5	113.0
Archive	4.5	566.9
Sign Result	77.8	75.8

Performance

Andrew Benchmark

- Five phases:
  - Create target directory (Write)
  - Copy files (Write)
  - Examine file status (Read)
  - Examine file data (Read)
  - Compile & link the files (Read+Write)

Performance

Andrew Benchmark

	LAN			WAN		
	Linux	OceanStore		Linux	OceanStore	
Phase	NFS	512	1024	NFS	512	1024
I	0.0	1.9	4.3	0.9	2.8	6.6
II	0.3	11.0	24.0	9.4	16.8	40.4
III	1.1	1.8	1.9	8.3	1.8	1.9
IV	0.5	1.5	1.6	6.9	1.5	1.5
V	2.6	21.0	42.2	21.5	32.0	70.0
Total	4.5	37.2	73.9	47.0	54.9	120.3

Times are in seconds

Discussion

- OceanStore: a great vision
- Pond: a working subset of the vision
- Cool functionalities, but some are expensive
- Performance upon failures?

- Questions?