



**Federated, Available, and Reliable
Storage for an Incompletely
Trusted Environment**

Atul Adya, Bill Bolosky, Miguel Castro,
Gerald Cermak, Ronnie Chaiken,
John Douceur, Jon Howell, Jay Lorch,
Marvin Theimer, Roger Wattenhoffer

Microsoft
Research

Project Goal

Build a scalable serverless file system

Security against malicious attacks is a necessity

- Byzantine protocols for untrusted infrastructure
- Leases, caching, batching to reduce overhead
- Separate replication for file and directory info

Talk Outline

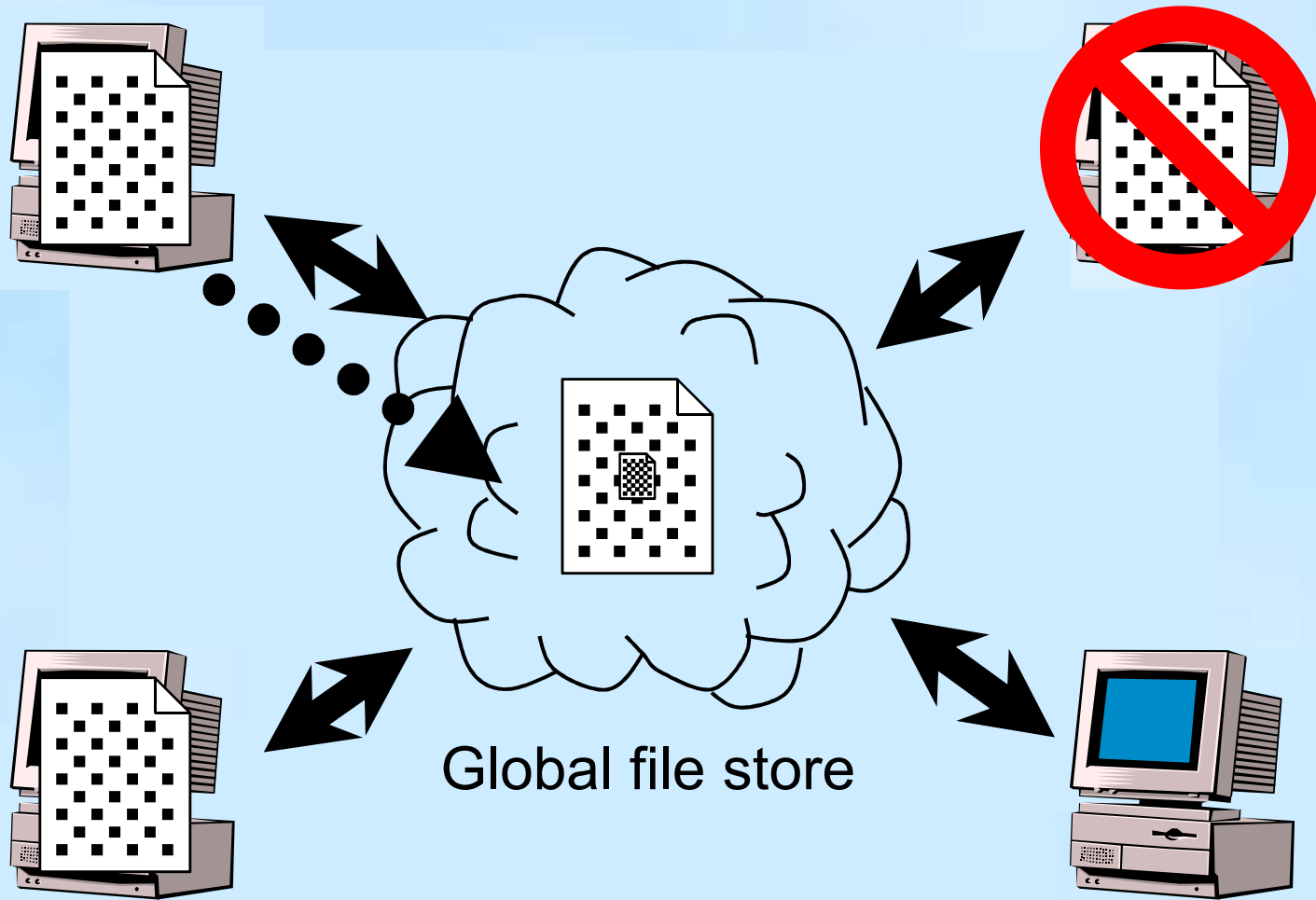
- Overview
- Architecture
- Implementation
- Related Work
- Conclusions

Why Serverless?

Servers are ...

- Reliant on system operators to
 - Perform maintenance functions
 - Not read/modify users' files
- Expensive (special hardware)
 - High-performance I/O, RAID disk, special rooms
- Centralized points of failure
- High-value targets for attacks

Farsite Solution



Farsite vs. central file server

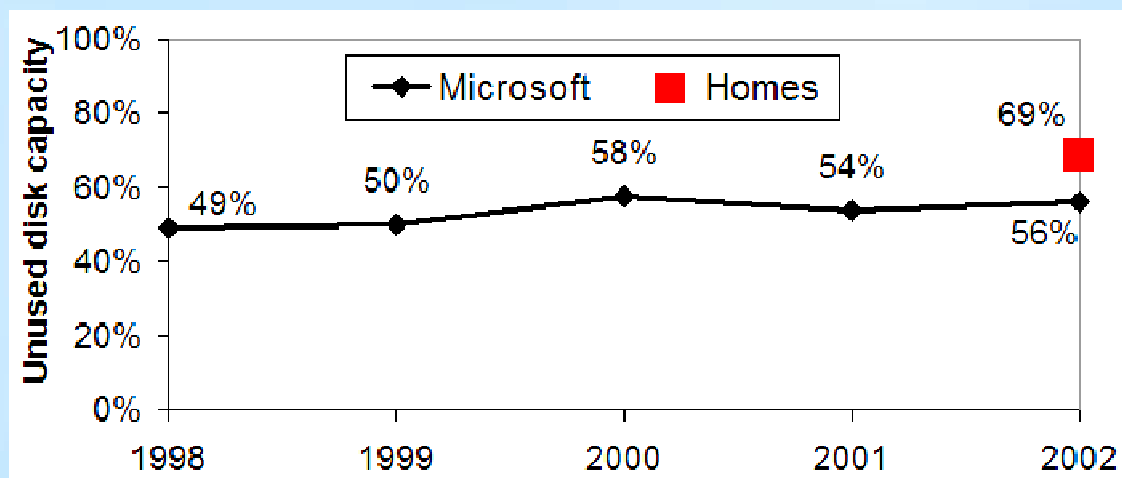
- Same functionality
 - Shared namespace
 - File and directory write-sharing
 - Strong consistency
- Better privacy: Uses cryptography
 - Less vulnerability to system operators
- Higher availability: Uses Byzantine fault-tolerance
 - More resilient to malicious attacks
- Cheaper
 - Implemented using client desktop machines
 - Reduced administration

Target Environment

- Large university or company
 - Fast network
 - Ignore different link bandwidth and latency
- Rough scale:
 - 10^5 total machines
 - 10^{10} total files
 - 10^{15} total bytes
- Parts of network may be compromised
- Small fraction of machines may be attacked

Enabling Trends

- Availability: enough disk space for replicas
 - Low disk costs
 - Unused disk capacity
 - Duplicate files: ~50% space savings [Sigmetrics00]
- Privacy: fast crypto
 - Symmetric encryption: 225 MB/sec
 - Disk sequential I/O bandwidth: 30 – 40 MB/sec
 - Digital signature: 4 msec



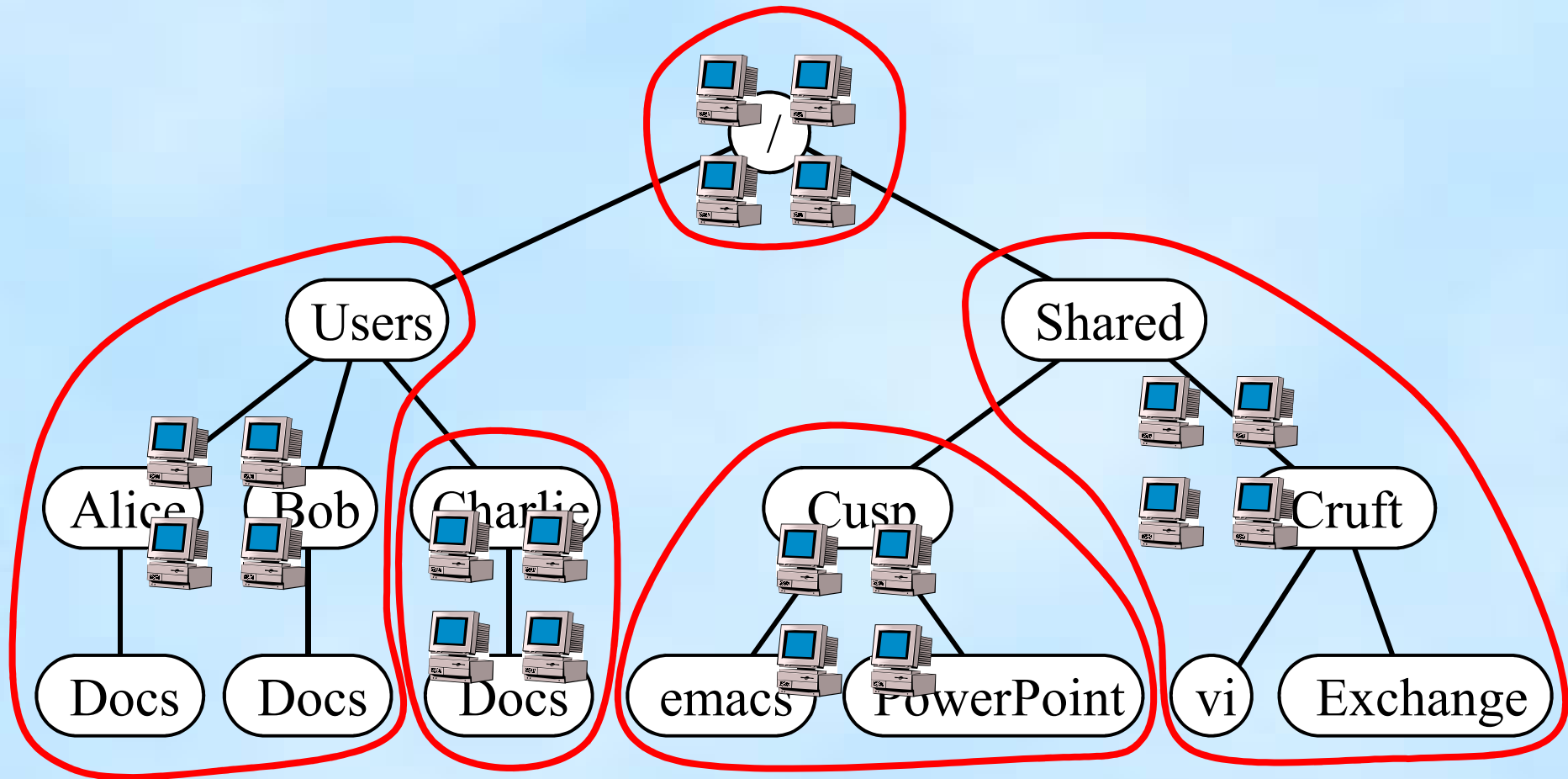
Project Non-Goals

- Efficient large-scale write sharing
- Database semantics
- High-performance parallel I/O
- Disconnected operation with offline conflicts

Talk Outline

- Overview
- **Architecture**
 - Space efficiency
 - Reducing expensive operations
- Implementation
- Related Work
- Conclusions

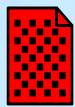
Farsite from 10,000 feet



Traditional Byzantine Approach [CL99]



Client

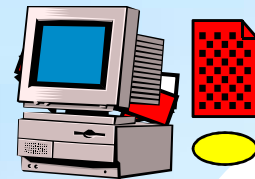
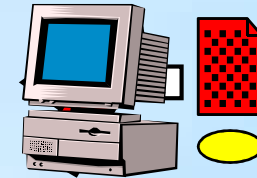
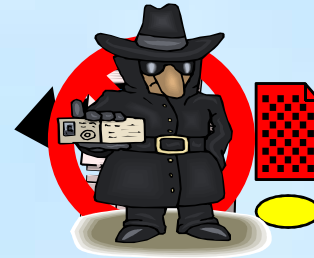


File

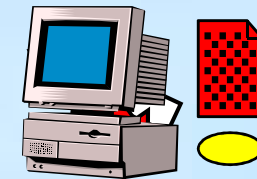
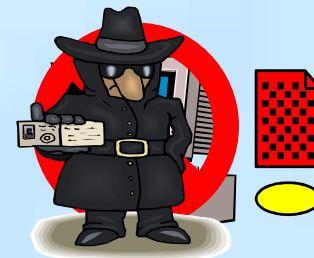
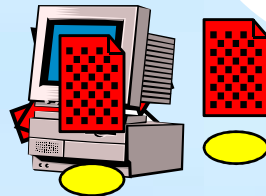
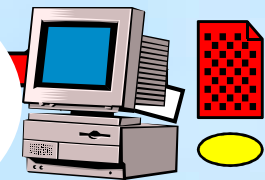


Meta-Data

**$3f + 1$ file copies
to handle f failures**

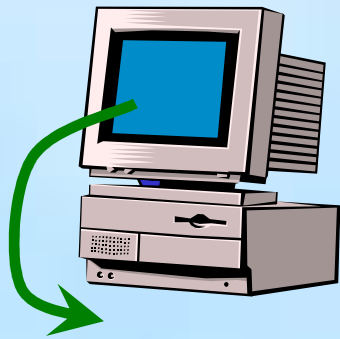


Byzantine fault-tolerant protocol

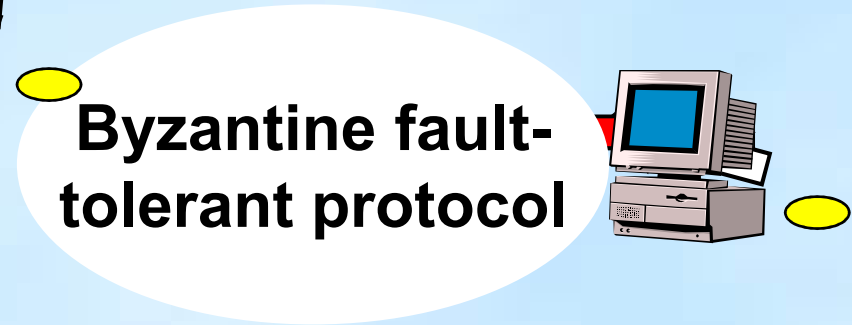
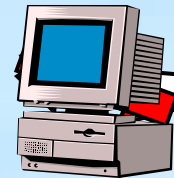
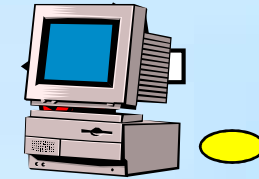
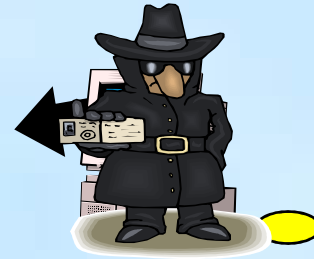


Byzantine servers

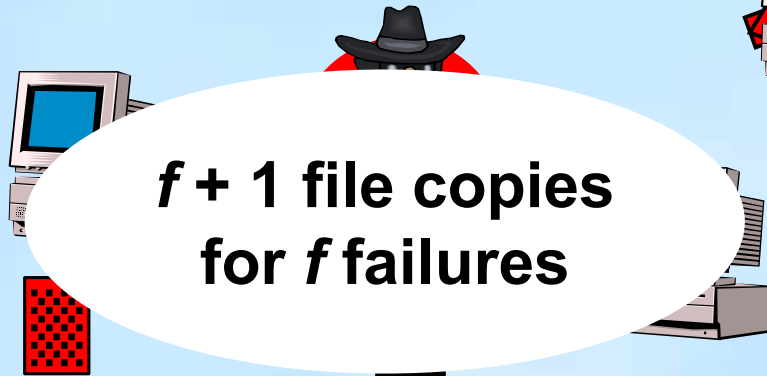
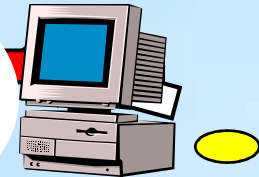
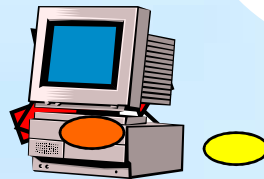
Farsite: BFT only for meta-data



Client

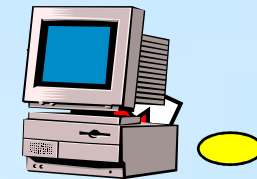


Byzantine fault-tolerant protocol



$f + 1$ file copies
for f failures

File hosts



Directory group



Data integrity rooted in BFT

- Directories/meta-data maintained via BFT
 - Perform trusted computation on behalf of clients
 - $3f + 1$ replicas for tolerating f faults
 - Consume less than 1% of space
- Files replicated via simple replication
 - $f + 1$ replicas to tolerate f faults
 - File content hash in meta-data allows verification
 - Average file size: 100 KB

Talk Outline

- Overview
- Architecture
 - Space efficiency
 - Reducing expensive operations
- Implementation
- Related Work
- Conclusions

Reducing Critical Path Delays

Cost of Byzantine operations:

- Less than 8 msec for 10 replicas
- 200 messages for 10 replicas

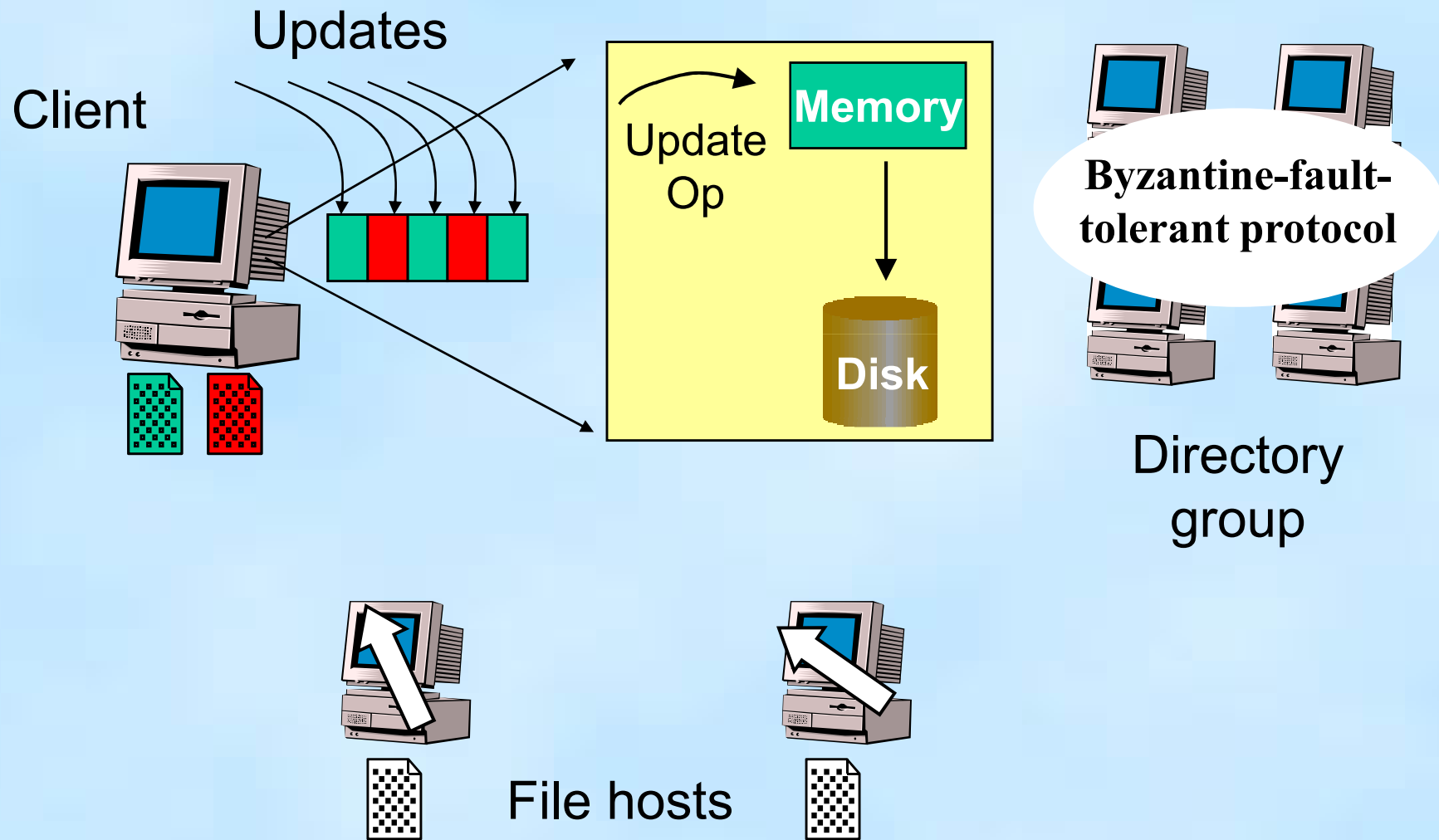
Reducing Byzantine operations on critical path:

- Localizing operations: leases
- Optimizing writes: batching and lazy updates

Localizing Operations via Leases

- *Content leases*: File data consistency
- *Name leases*: Namespace consistency
- *Mode leases*: Windows file-sharing semantics
- *Access leases*: Windows deletion semantics

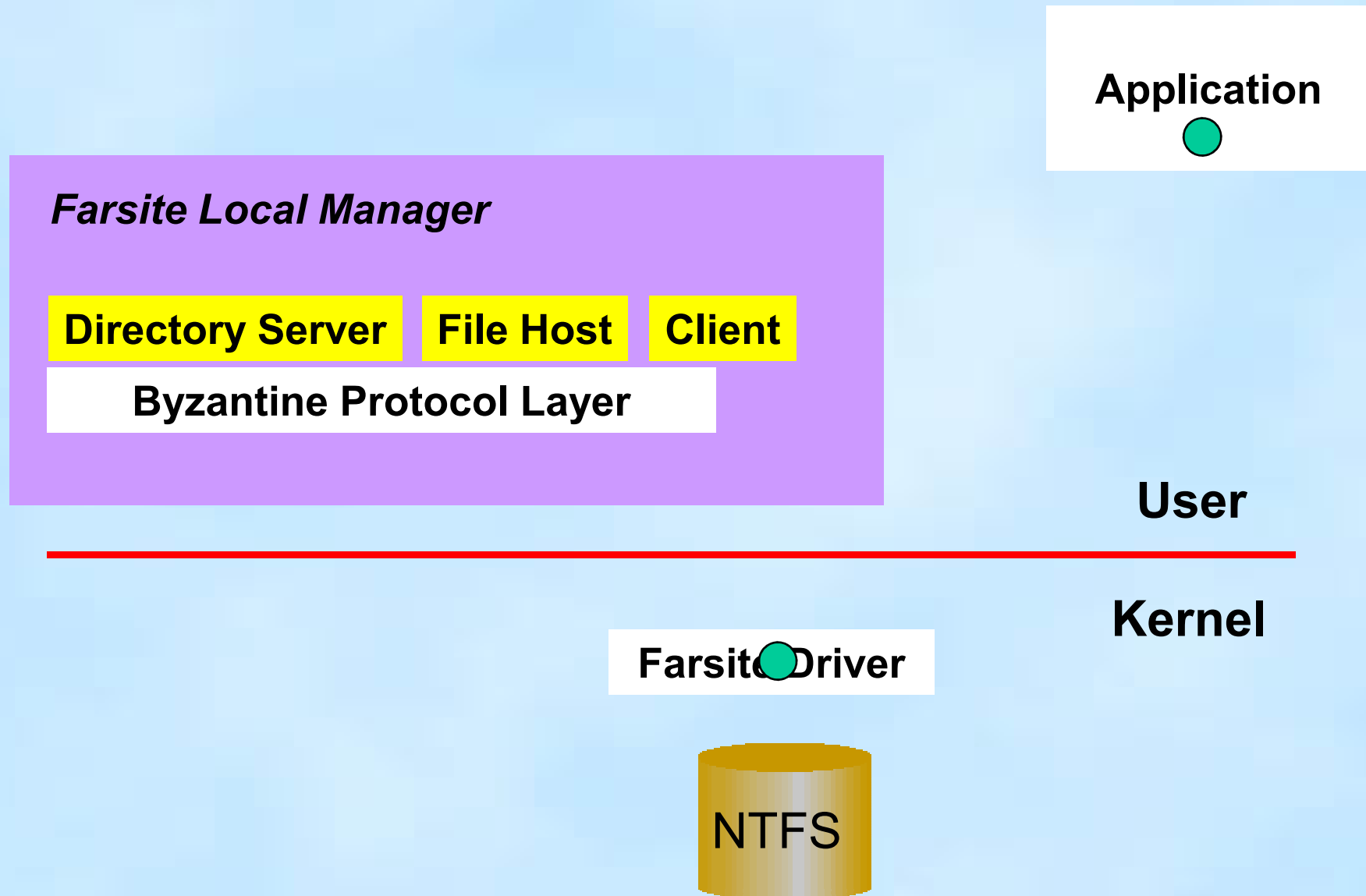
Optimizing Updates: Batching



Talk Outline

- Overview
- Architecture
- **Implementation**
- Related Work
- Conclusions

Software Structure



Performance

- Method
 - 1-hour representative trace from 3-week traces
 - Played at real time (~ 450,000 operations)
 - Measured aggregate file operations latency
- Results (Untuned system)
 - 20% faster than central file system (CIFS)
 - 6x slower than NTFS

Performance Analysis

- Fine tuning needed to compete with local FS
 - Half of performance penalty on slow *stat* path
- Untuned system faster than central server
 - Due to client disk caching and leases
- Exploit low average file system load
 - Push batched updates in background

Future Work

- Implementation work
 - Directory group delegation
 - Metadata checkpointing and recovery
 - Distributed duplicate-content detection
- Design work
 - Monitoring machine availability for file replication
 - Quota management for fair space allocation

Related Work

- Distributed File Systems
 - NFS, AFS, xFS, Frangipani, BFS, SFS, SUNDR:
require *trusted* servers/clients or *unscalable*
- Peer-to-peer *immutable* storage systems
 - CFS, PAST, Eternity Service, Freenet, Pasis, Intermemory
- Peer-to-peer mutable storage systems
 - Oceanstore, Pangaea, Ivy
- Separation of directory and file info
 - Napster

Conclusions

If you build a scalable serverless file system

⇒ *security* is a necessary pre-requisite for scalability

Farsite distributed file system:

- Operates securely on untrusted client machines
- Replicates file and directory information separately
- Leases, caching, batching reduce overhead
- Untuned system performs acceptably