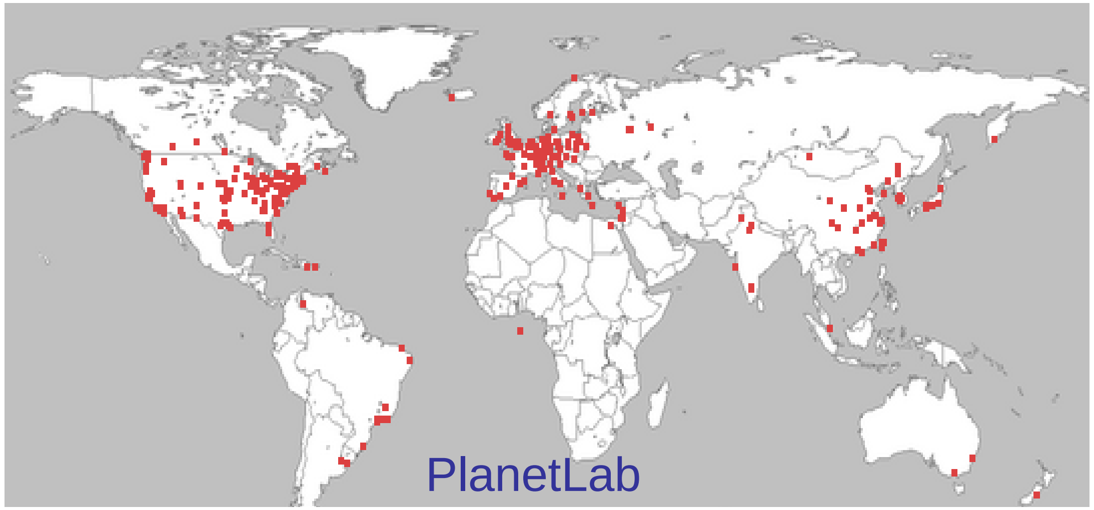
Flexible, Wide-Area Storage for Distributed Systems Using Semantic Cues

Jeremy Stribling

Thesis Defense, August 6, 2009

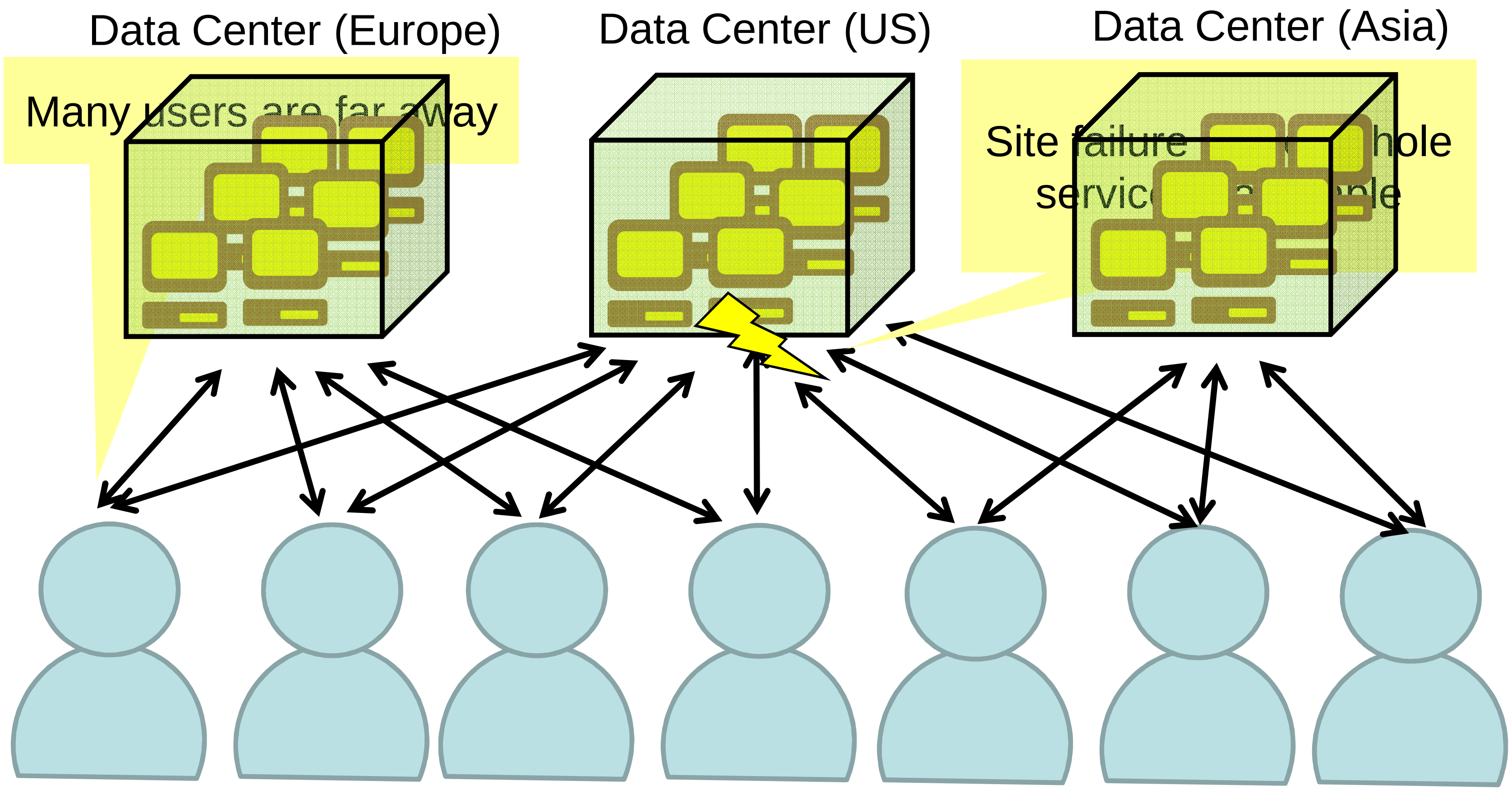
Including material previously published in:
Flexible, Wide-Area Storage for Distributed Systems With WheelFS
Jeremy Stribling, Yair Sovran, Irene Zhang, Xavid Pretzer,
Jinyang Li, M. Frans Kaashoek, and Robert Morris
NSDI, April 2009.

Storing Data All Over the World



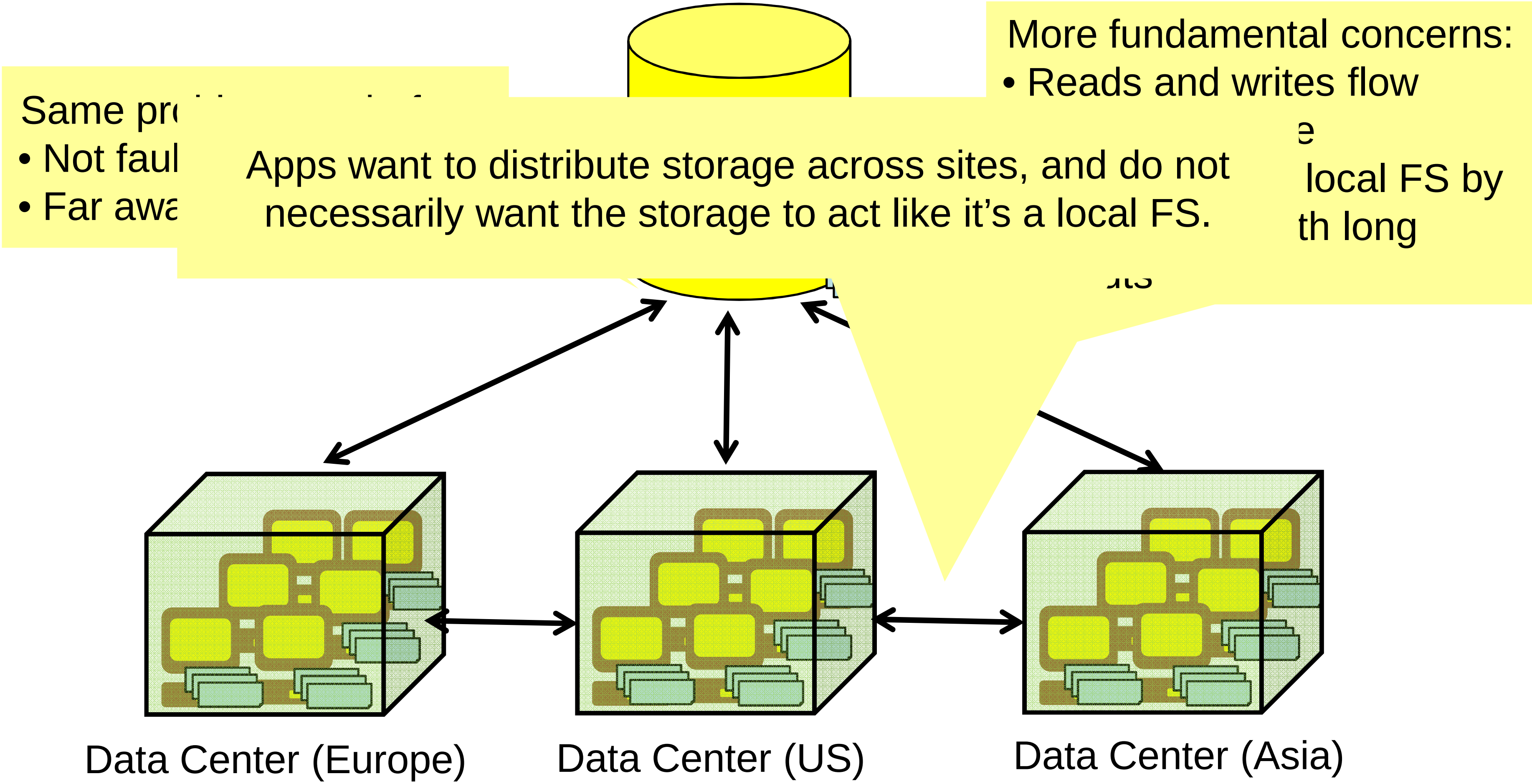
- Apps store data on widely-spread resources
  - Testbeds, Grids, data centers, etc.
  - Yet there’s no universal storage layer
- WheelFS: a file system for wide-area apps

Wide-Area Applications

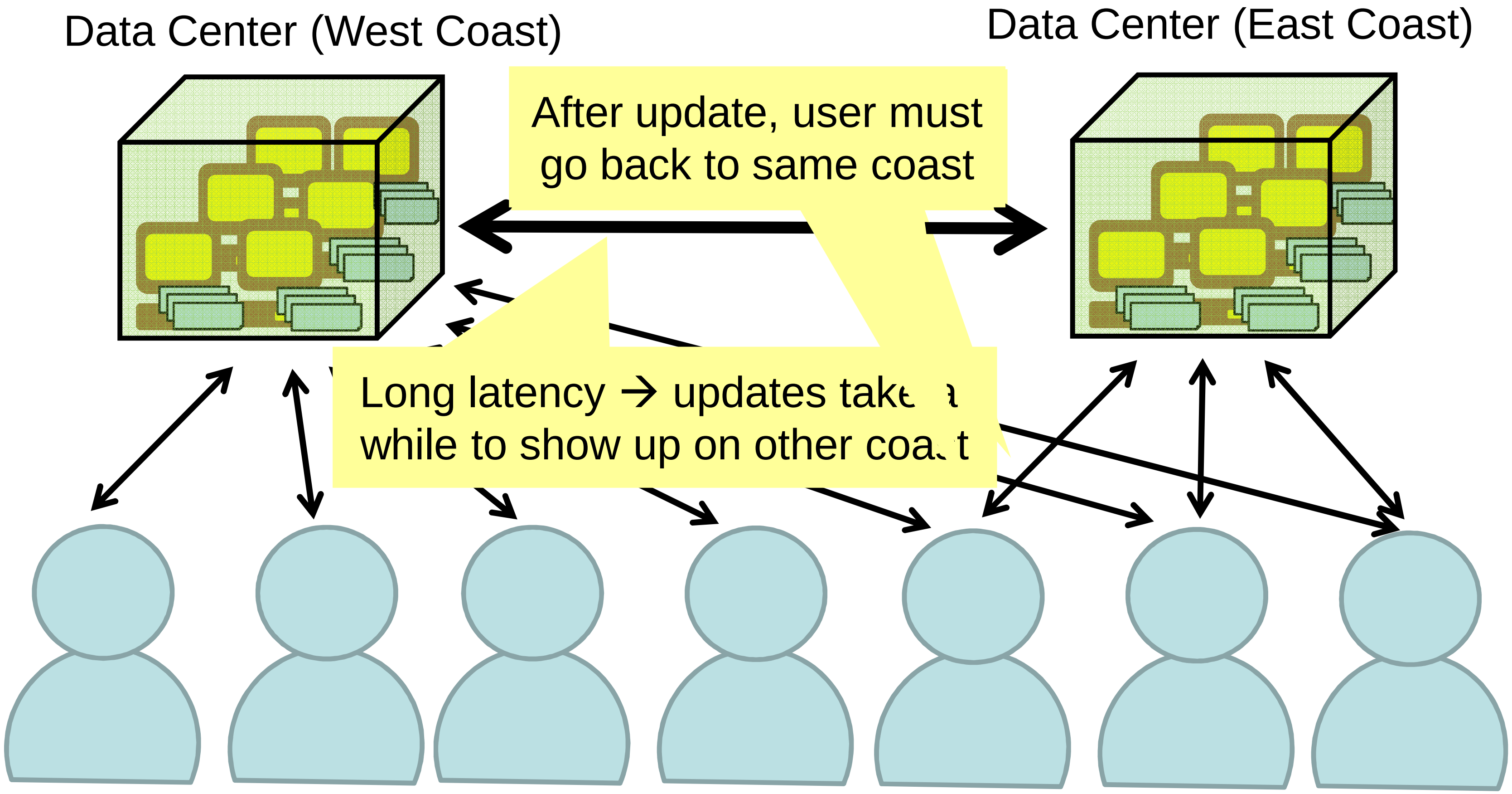


Data, not just app logic, needs to be shared across sites

Current Network FSes Don't Solve the Problem

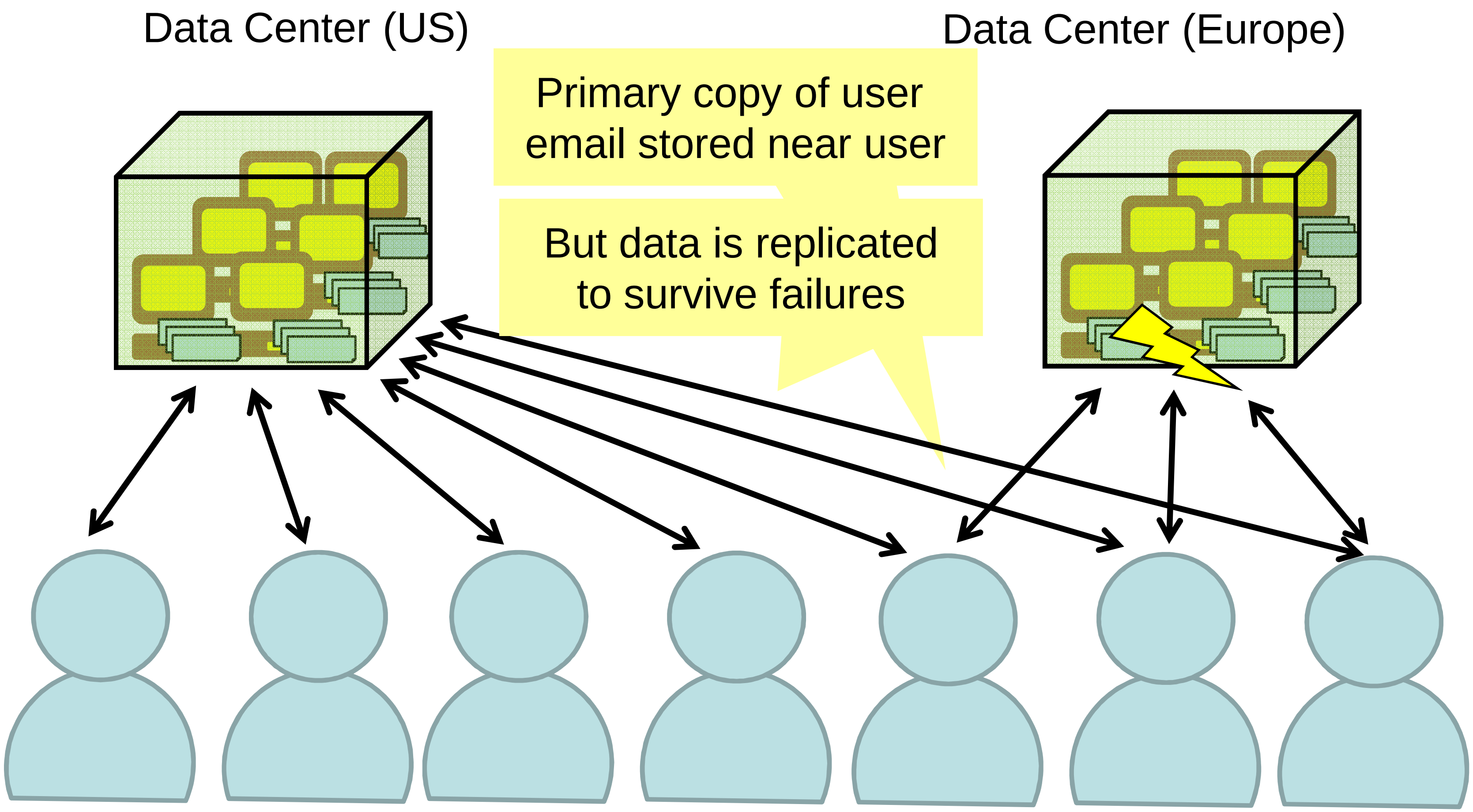


Wide-Area Storage Example: Facebook



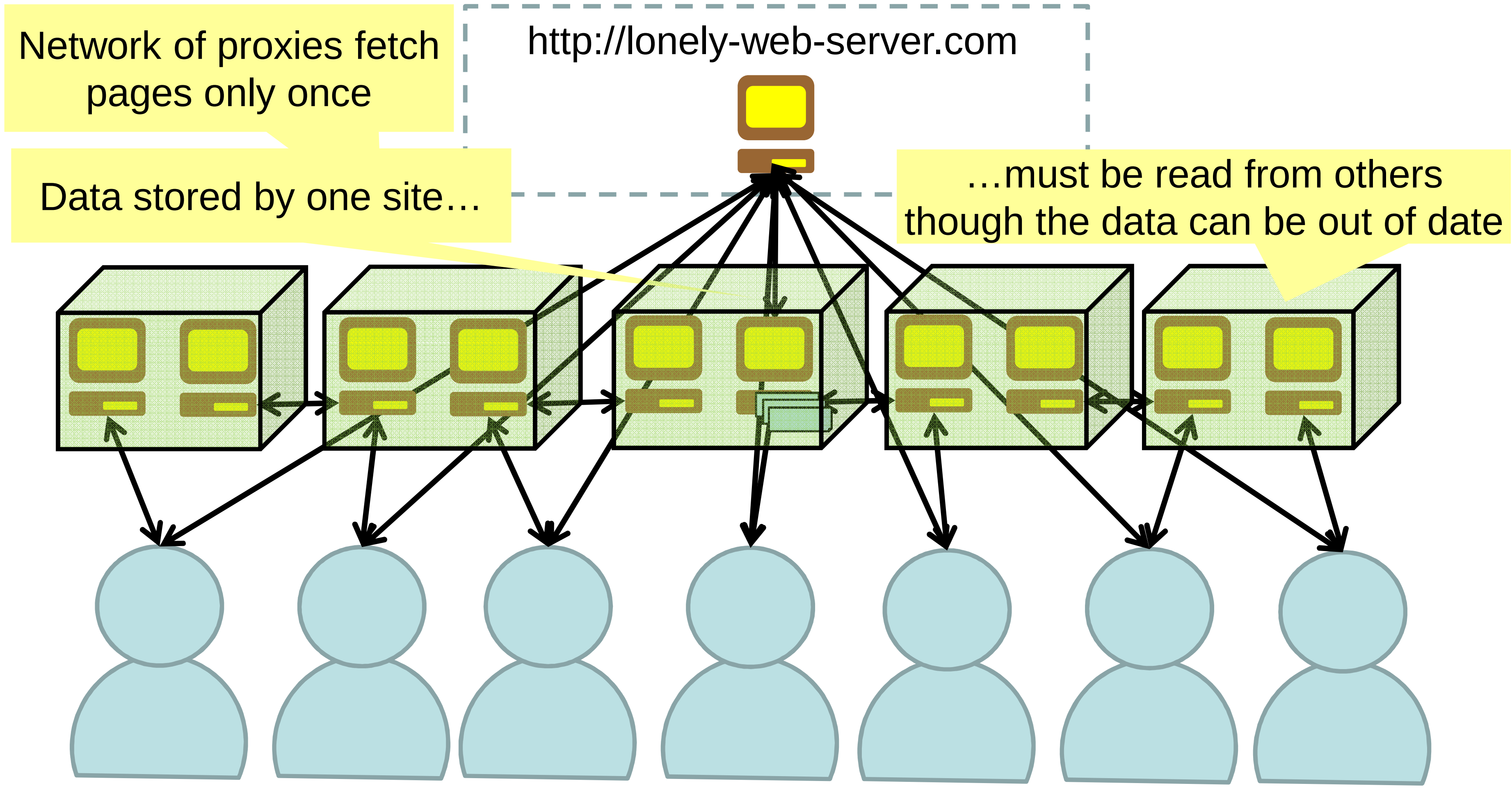
Storage requirement: control over *consistency*

Wide-Area Storage Example: Gmail



Storage requirements: control over *placement* and *durability*

Wide-Area Storage Example: CoralCDN



Storage requirements: distributed serving of popular files and control over *consistency*

Apps Handle Wide-Area Differently

- Facebook wants consistency for some data (Customized MySQL/Memcached)
- Google stores email near consumer (Gmail's storage layer)
- CoralCDN prefers low delay to strong consistency (Coral Sloppy DHT)

→ Each app builds its own storage layer

Opportunity: General-Purpose Wide-Area Storage

- Apps need control of wide-area tradeoffs
  - Availability vs. consistency
  - Fast writes vs. durable writes
  - Few writes vs. many reads
- Need a common, familiar API: File system
  - Easy to program, reuse existing apps
- No existing DFS allows such control

Solution: Semantic Cues

- Small set of app-specified controls
- Correspond to wide-area challenges:
  - **EventualConsistency**: relax consistency
  - **RepLevel**=*N:* control number of replicas
  - **Site=***site*: control data placement
- Allow apps to specify on per-file basis
  - */fs/.EventualConsistency/file*

Contribution: WheelFS

- Wide-area file system
- Apps embed cues directly in pathnames
- Many apps can reuse existing software
- Multi-platform prototype w/ several apps

File Systems 101

- Basic FS operations:
  - Name resolution: hierarchical name → flat id (*i.e.*, an *inumber*)

```
open("/dir1/file1", …) → id: 1235
```

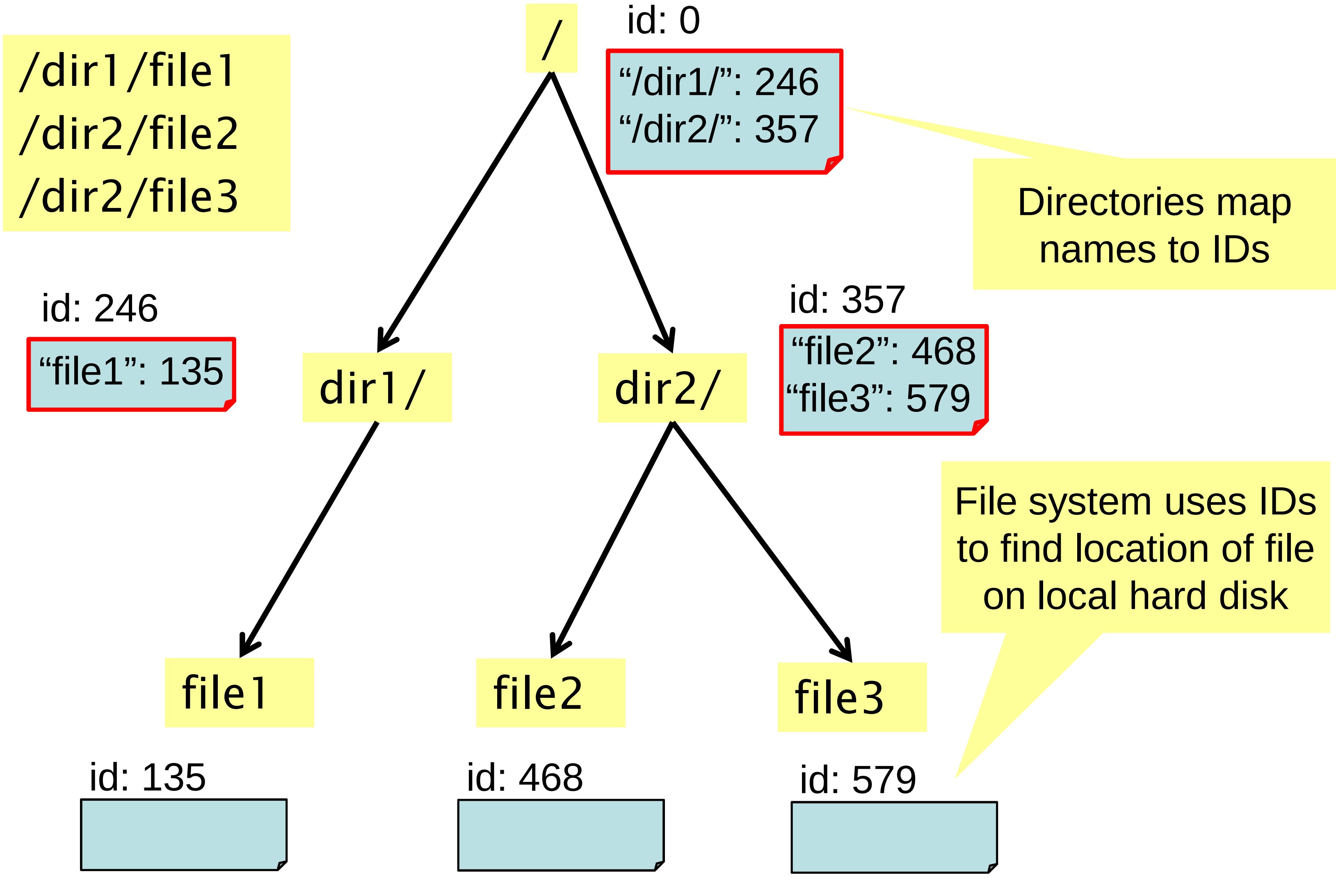
  - Data operations: read/write file data

```
read(1235, …)
write(1235, …)
```

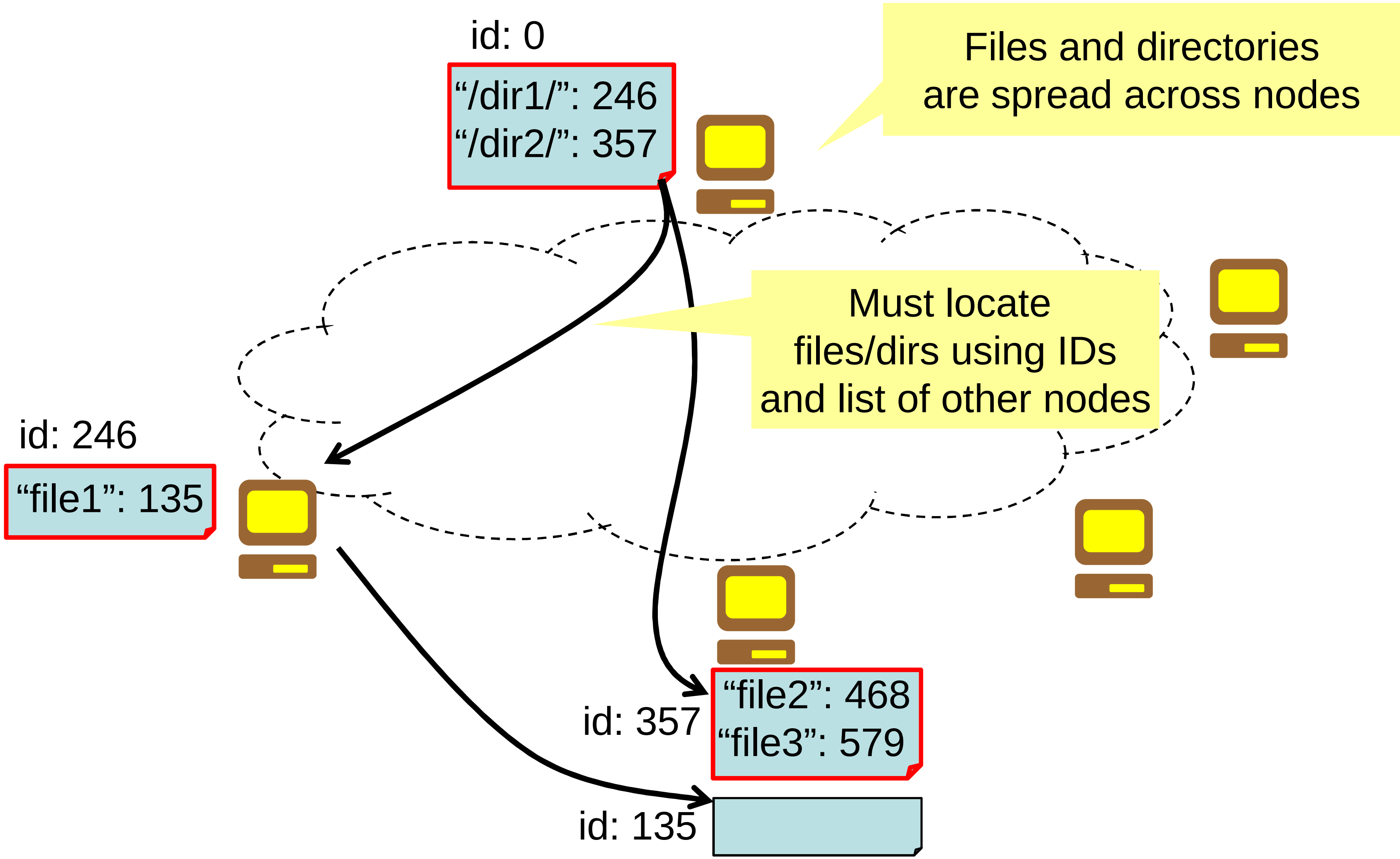
  - Namespace operations: add/remove files or dirs

```
mkdir("/dir2", …)
```

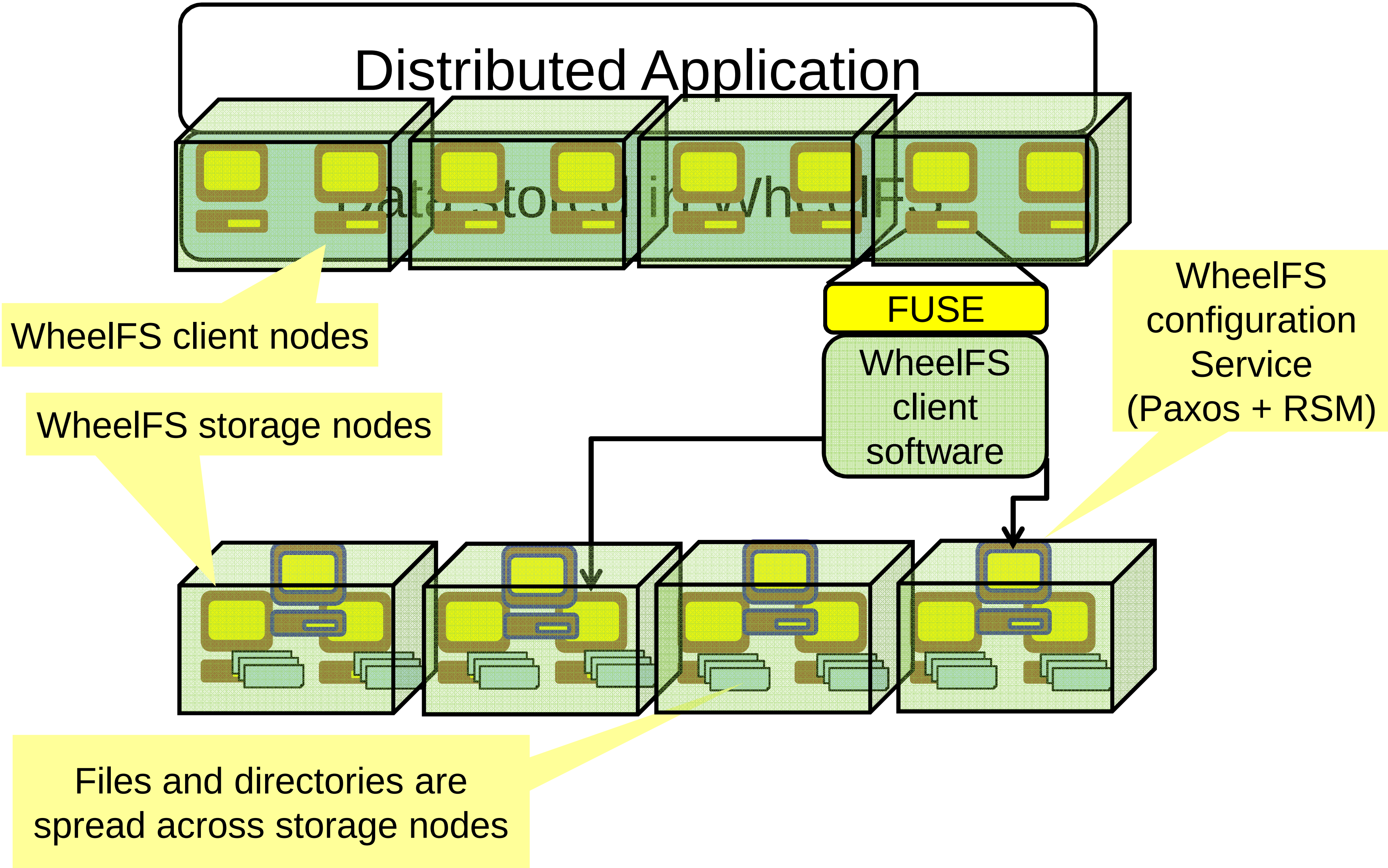
File Systems 101



Distributing a FS across nodes



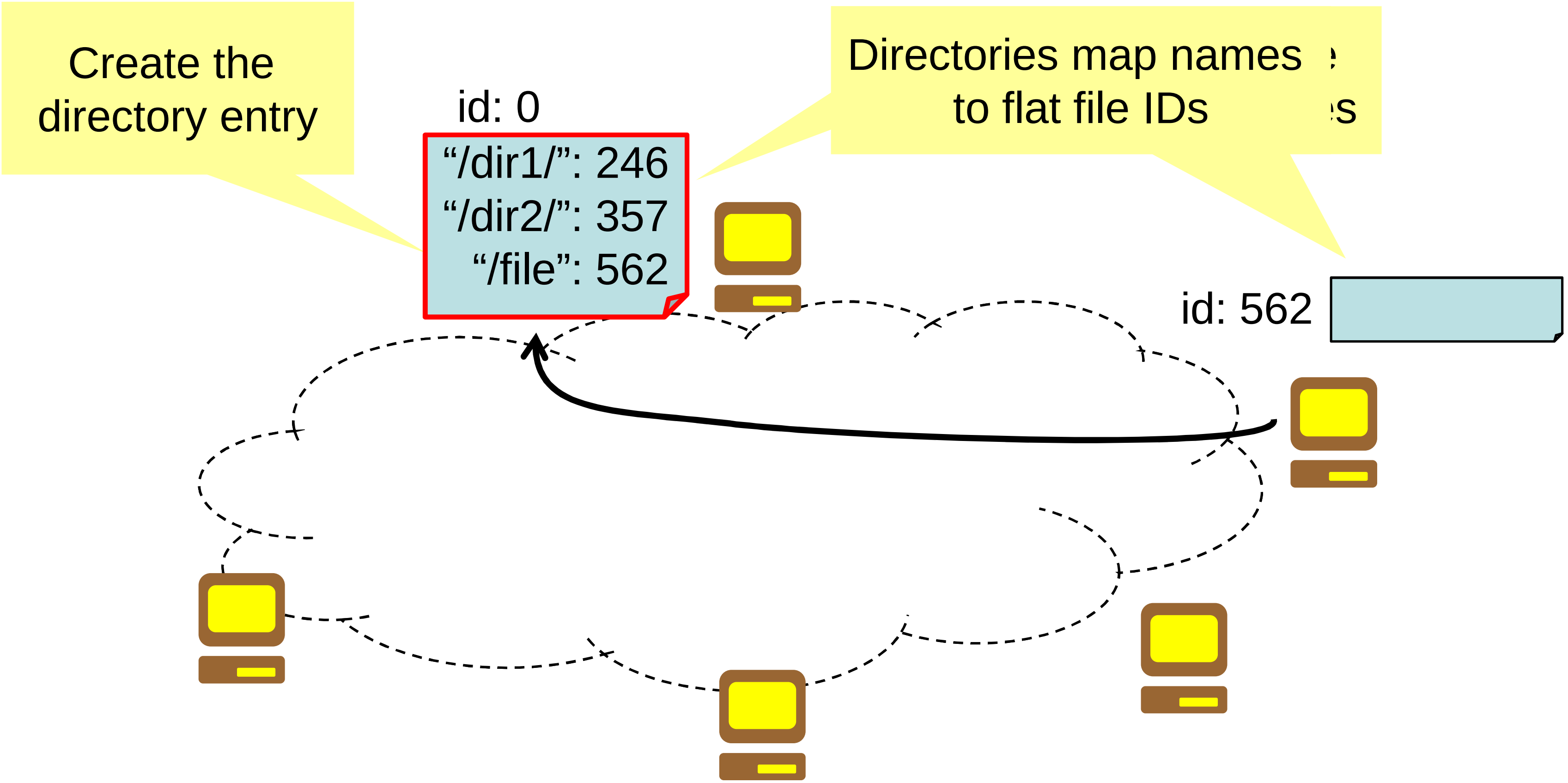
WheelFS Design Overview



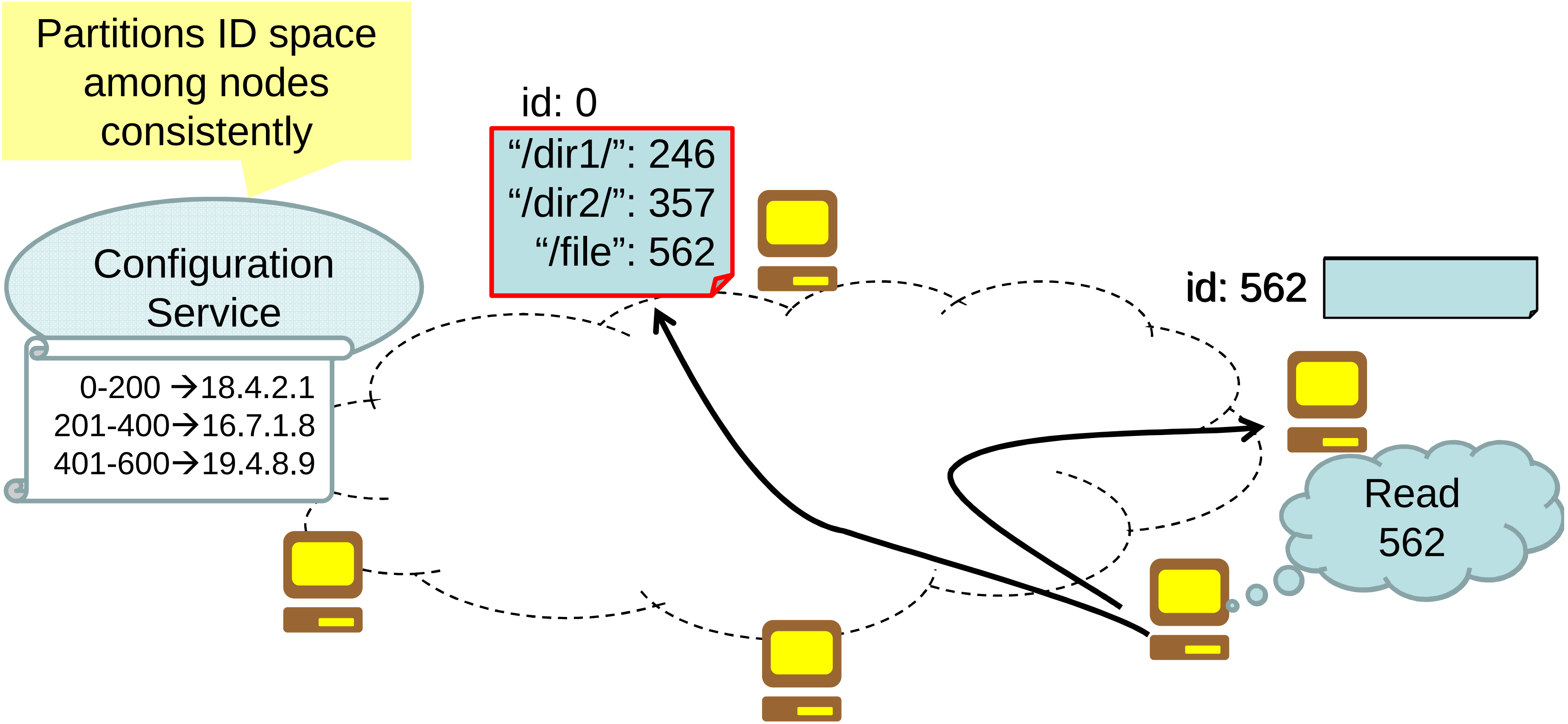
WheelFS Default Operation

- Files have a primary and two replicas
  - A file's primary is the closest storage node
- Clients can cache files
  - Lease-based invalidation protocol
- Strict close-to-open consistency
  - All operations serialized through the primary

WheelFS Design: Creation

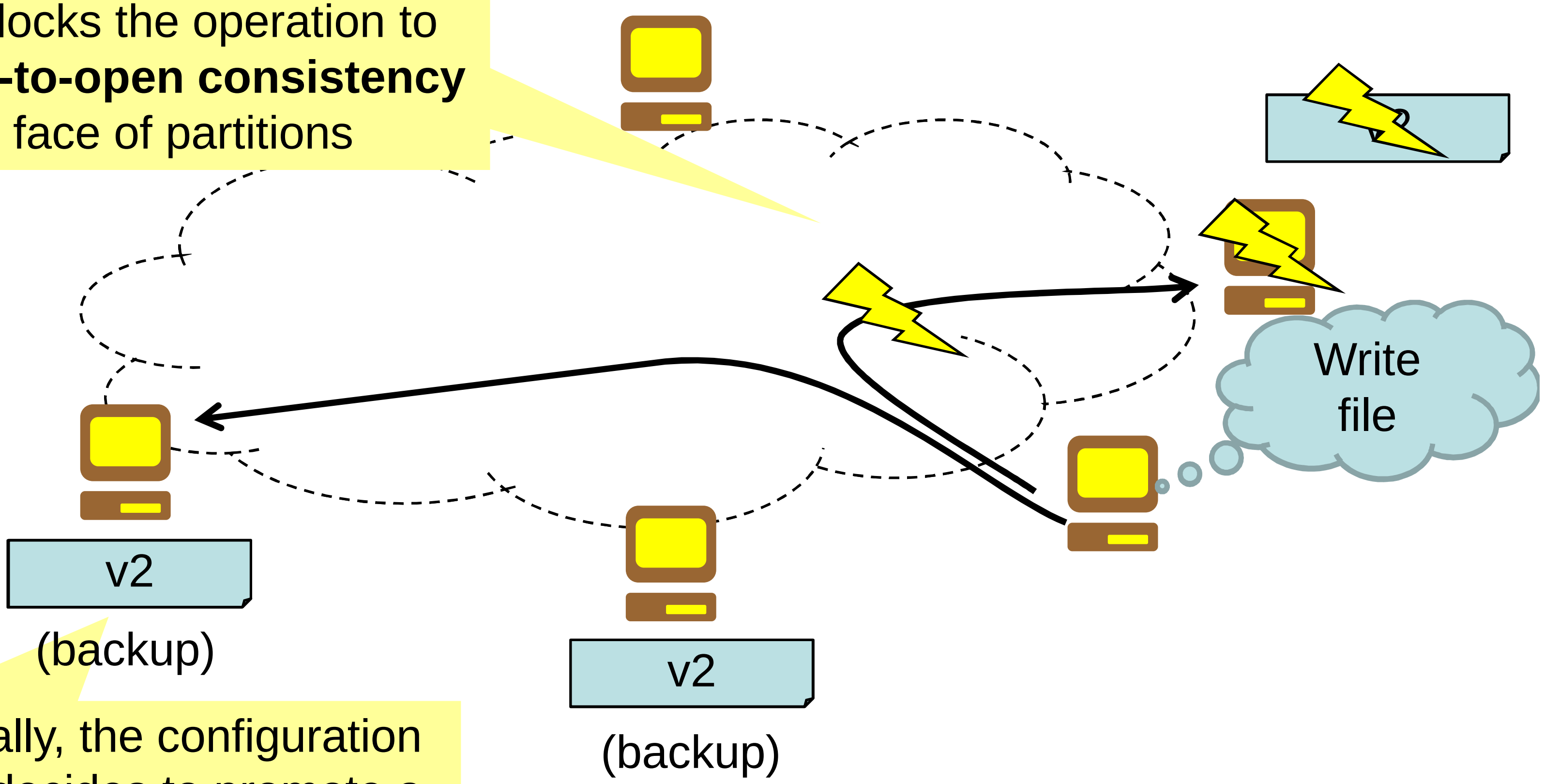


WheelFS Design: Open



Enforcing Close-to-Open Consistency

By default, failing to reach the primary blocks the operation to offer **close-to-open consistency** in the face of partitions



Eventually, the configuration service decides to promote a backup to be primary

Wide-Area Challenges

- Transient failures are common
  - Availability vs. consistency
- High latency
  - Fast writes vs. durable writes
- Low wide-area bandwidth
  - Few writes vs. many reads

Only applications can make these tradeoffs

Semantic Cues Gives Apps Control

- Apps want to control consistency, data placement ...
- How? Embed cues in path names

/wfs/cache/a/b/foo

/wfs/cache/a/b/foo

*/wfs/cache/a/b/**EventualConsistency**/foo*

→ Flexible and minimal interface change

Semantic Cue Details

- Cues can apply to directory subtrees

*/wfs/cache/**.EventualConsistency**/a/b/foo*

Cues apply recursively over an entire subtree of files

- Multiple cues can be in effect at once

*/wfs/cache/**.EventualConsistency/.RepLevel=2**/a/b/foo*

Both cues apply to the entire subtree

- Assume developer applies cues sensibly

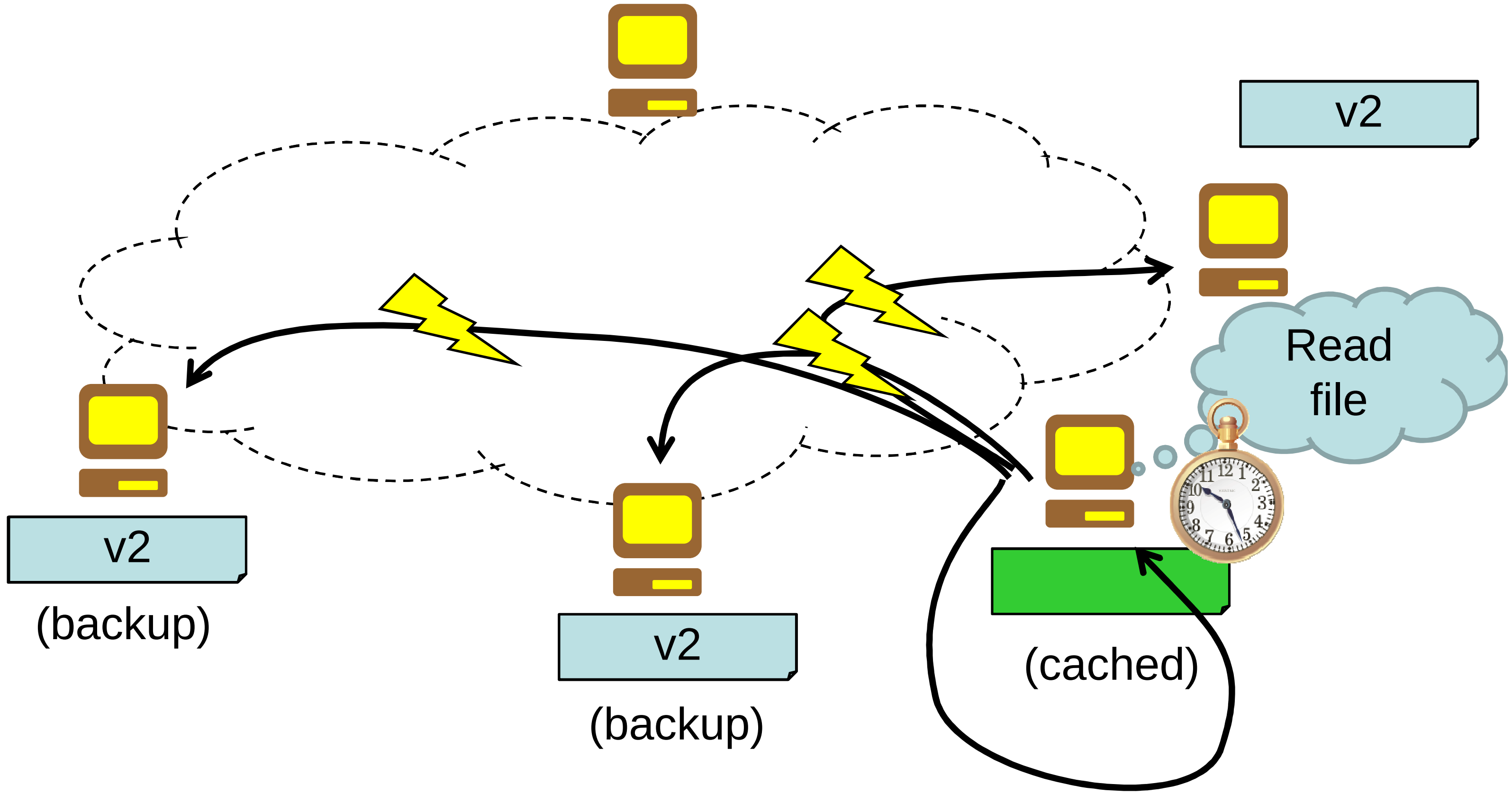
A Few WheelFS Cues

	Name	Purpose
Durability	**RepLevel=** *(permanent)*	How many replicas of this file should be maintained
Large reads	**HotSpot** *(transient)*	This file will be read simultaneously by many nodes, so use p2p caching
Hint about data placement	**Site=** *(permanent)*	Hint for which group of nodes should store a file
Consistency	**Eventual-Consistency** *(trans/perm)*	Control whether reads must see fresh data, and whether writes must be serialized

Cues designed to match wide-area challenges

Eventual Consistency: Reads

- Read latest version of the file you can find quickly
- In a given time limit (**.MaxTime=**)



Eventual Consistency: Writes

- Write to primary *or* any backup of the file

Reconciling divergent replicas:

<u>Directories</u>	<u>Files</u>
• Merge replicas into single directory by taking union of entries → Tradeoff: May lose some unlinks	• Choose one of the recent replicas to win →Tradeoff: May lose some writes

(No application involvement)

v3

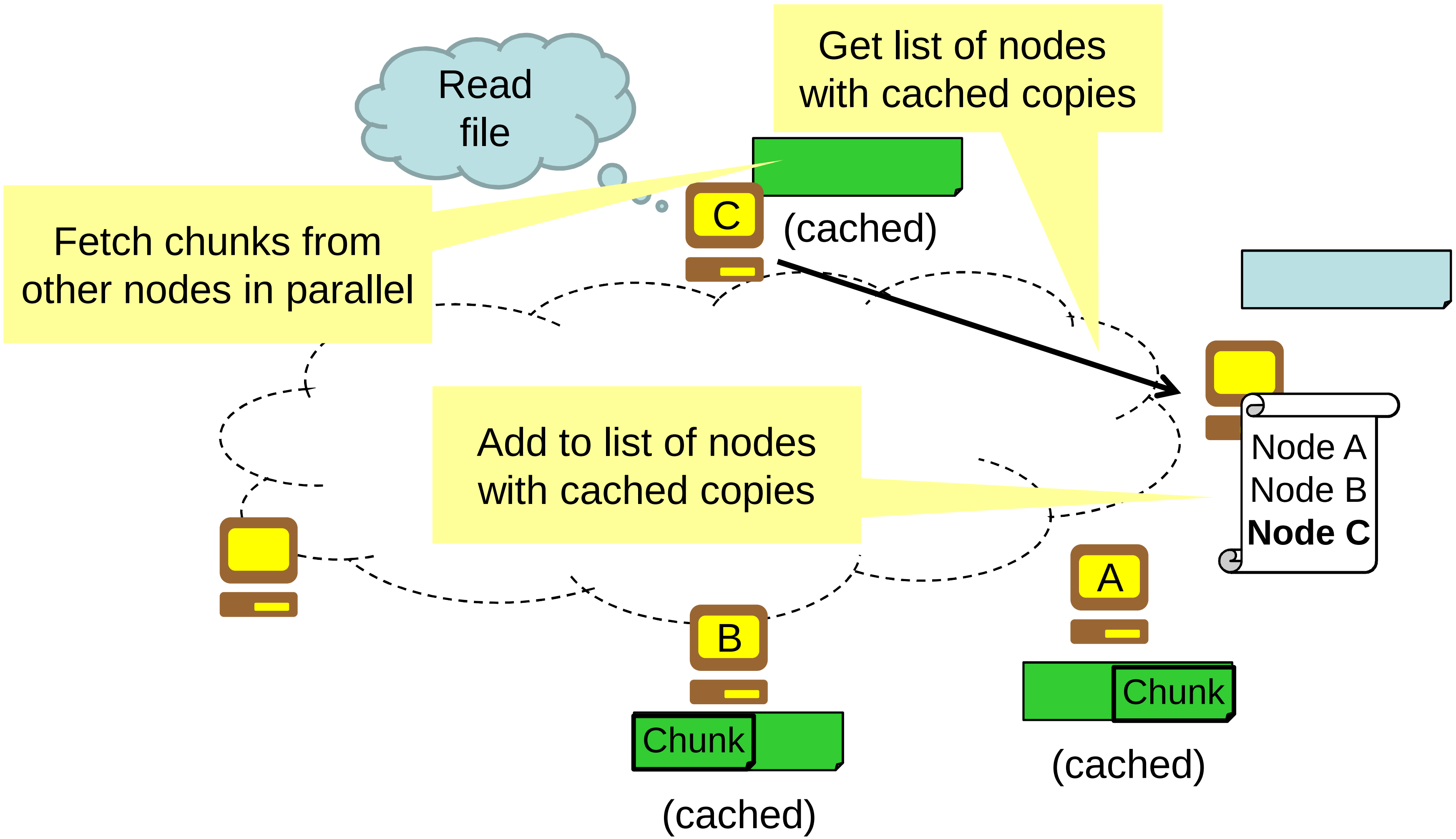
v3

(backup)

will merge divergent replicas

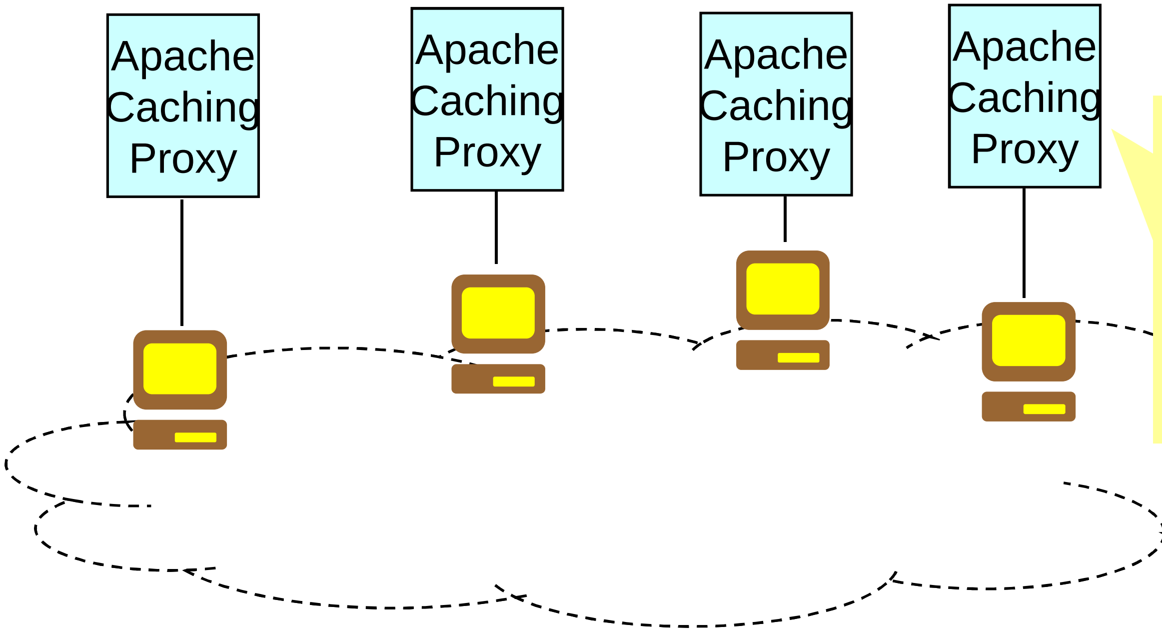
Create new version at backup

HotSpot: Client-to-Client Reads



Use Vivaldi network coordinates to find nearby copies

Example Use of Cues: Cooperative Web Cache (CWC)



Blocks under failure with default strong consistency

```
If $url exists in cache dir
   read $url from WheelFS
else
   get page from web server
   store page in WheelFS
```

One line change in Apache config file: /wfs/cache/$URL

Example Use of Cues: CWC

- Apache proxy handles potentially stale files well
  - The freshness of cached web pages can be determined from saved HTTP headers

Cache dir: */wfs/cache/*.**EventualConsistency**/**.MaxTime=200**/**.HotSpot**

Read a cached file even when the corresponding primary cannot be contacted

Write the file data to any backup when the corresponding primary cannot be contacted

Reads only block for 200 ms; after that, fall back to origin web server

Tells WheelFS to read data from the nearest client cache it can find

WheelFS Implementation

- Runs on Linux, MacOS, and FreeBSD
- User-level file system using FUSE
- 25K lines of C++
- Unix ACL support
- Vivaldi network coordinates

Applications Evaluation

App	Cues used	Lines of code/configuration written or changed
Cooperative Web Cache	**.EventualConsistency, .MaxTime, .HotSpot**	1
All-Pairs-Pings	**.EventualConsistency, .MaxTime, .HotSpot, .WholeFile**	13
Distributed Mail	**.EventualConsistency, .Site, .RepLevel, .RepSites, .KeepTogether**	4
File distribution	**.WholeFile, .HotSpot**	N/A
Distributed make	**.EventualConsistency** (for objects)**, .Strict** (for source)**, .MaxTime**	10

Performance Questions

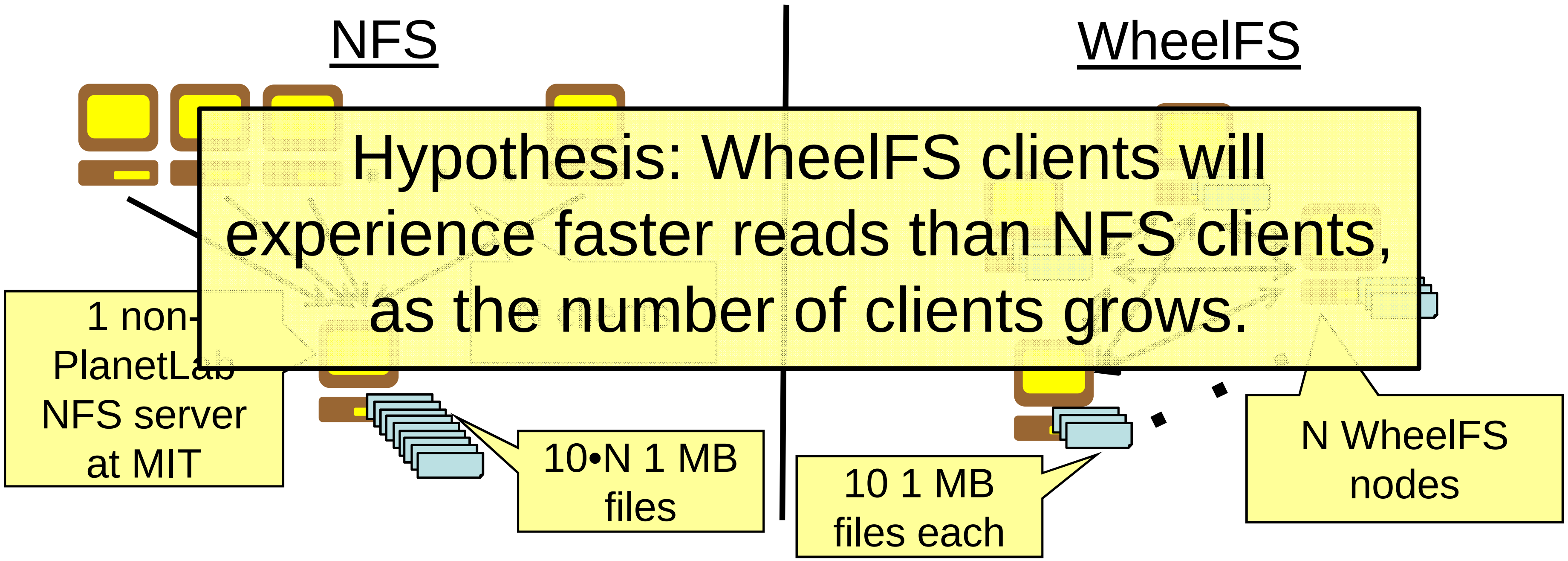
WheelFS is a wide-area file system that:

- spreads the data load across many nodes
- aims to support many wide-area apps
- give apps control over wide-area tradeoffs using cues

1. Does WheelFS distribute app storage load more effectively than a single-server DFS?
2. Can WheelFS apps achieve performance comparable to apps w/ specialized storage?
3. Do semantic cues improve application performance?

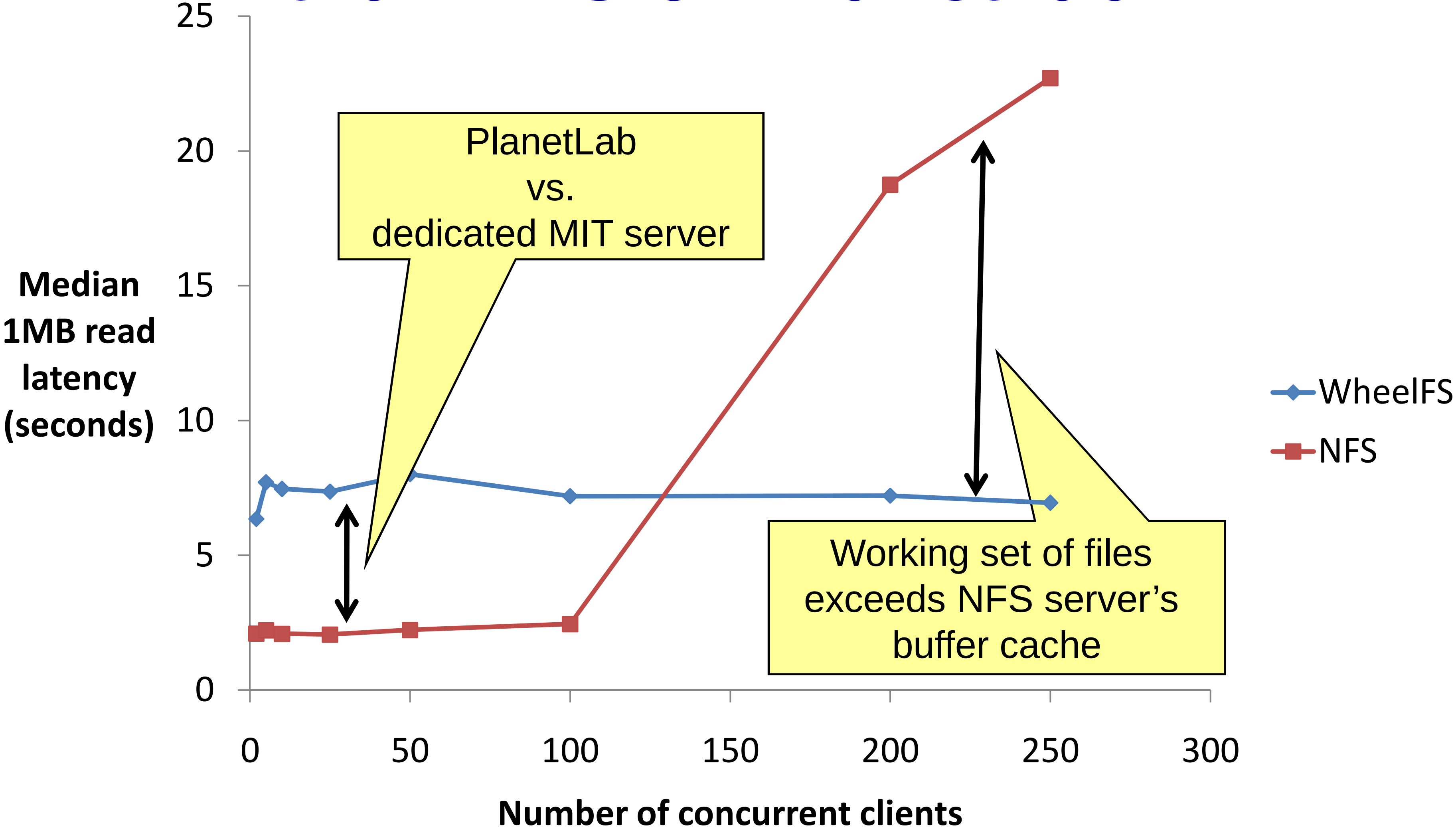
Storage Load Distribution Evaluation

- Up to 250 PlanetLab nodes



- Each client reads 10 files at random

WheelFS Spreads Load More Evenly than NFS on PlanetLab

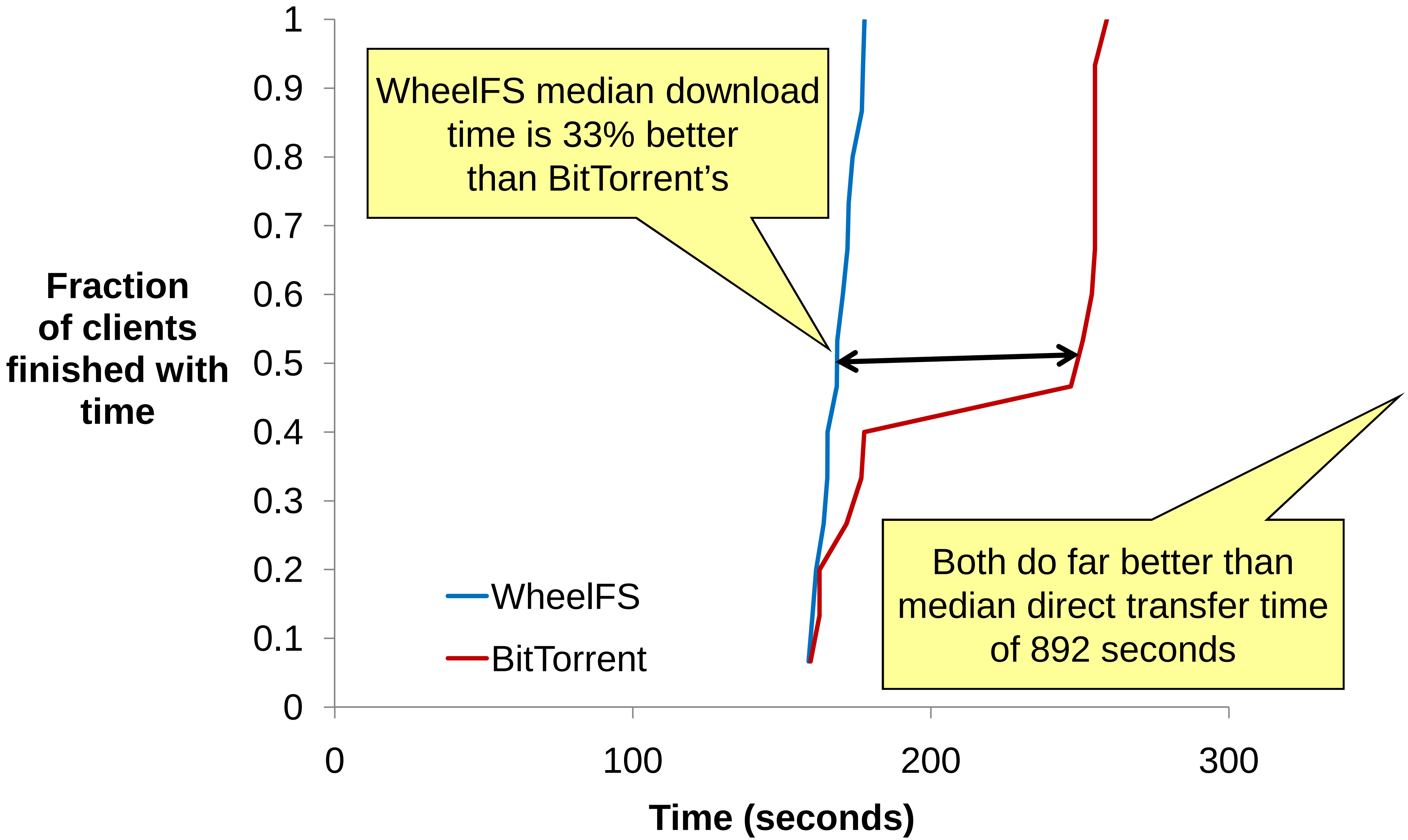


File Distribution Evaluation

- 15 nodes at 5 wide-area sites on Emulab
- All nodes download 50 MB at the same time
- Direct transfer time for one node is 73 secs
- Use .HotSpot cue
- Compare against BitTorrent

Hypothesis: WheelFS will achieve performance comparable to BitTorrent's, which uses a specialized data layer.

WheelFS HotSpot Cue Gets Files Faster than BitTorrent

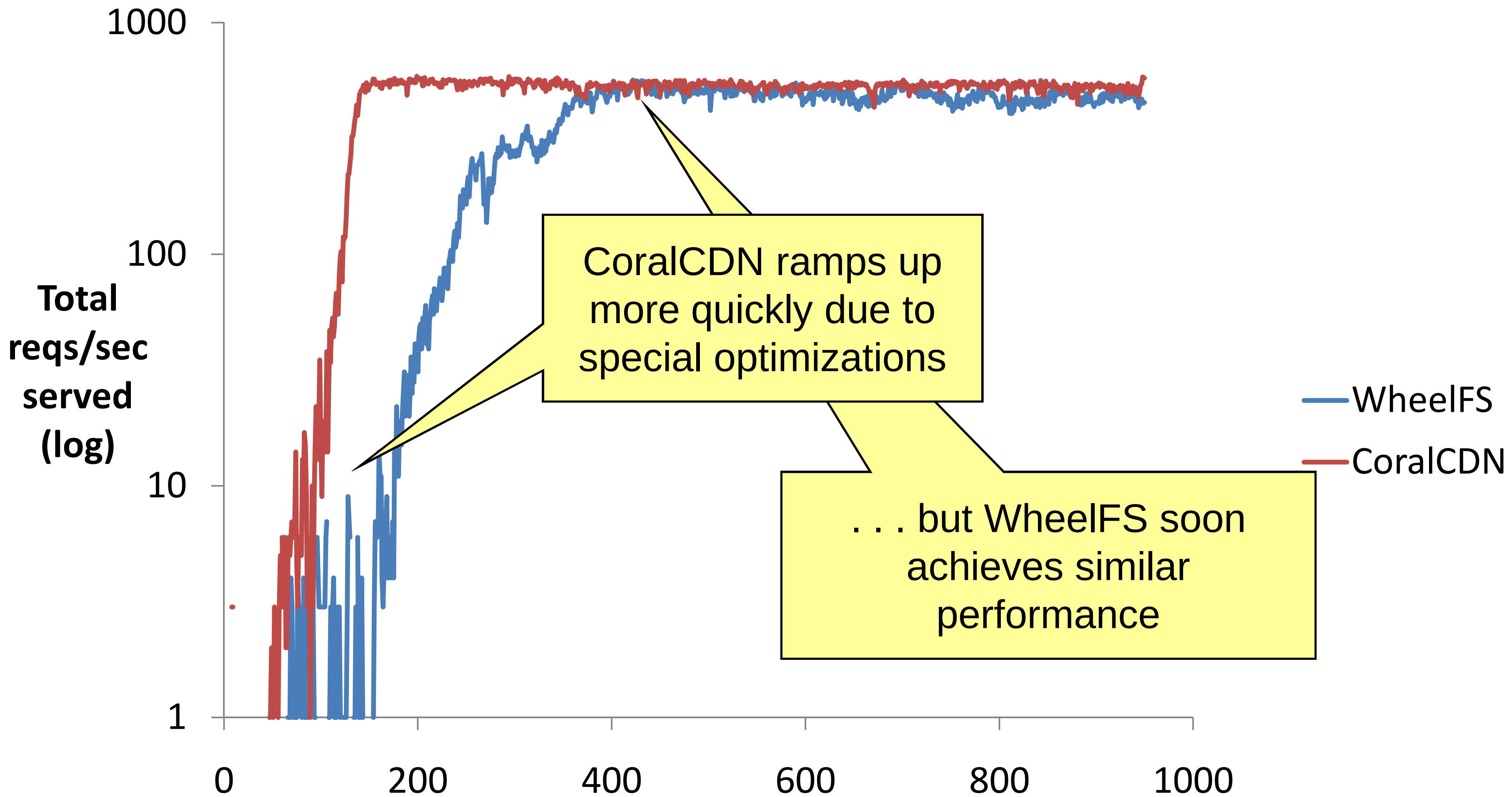


CWC Evaluation

- 40 PlanetLab nodes as Web proxies
- 40 PlanetLab nodes as clients
- Web server
  - 400 Kbps link
  - 100 unique 41 KB pages
- Each client downloads random pages
  - (Same workload as in CoralCDN paper)
- CoralCDN vs. WheelFS + Apache

Hypothesis: WheelFS will achieve performance comparable to CoralCDN's, which uses a specialized data layer.

WheelFS Achieves Same Rate As CoralCDN



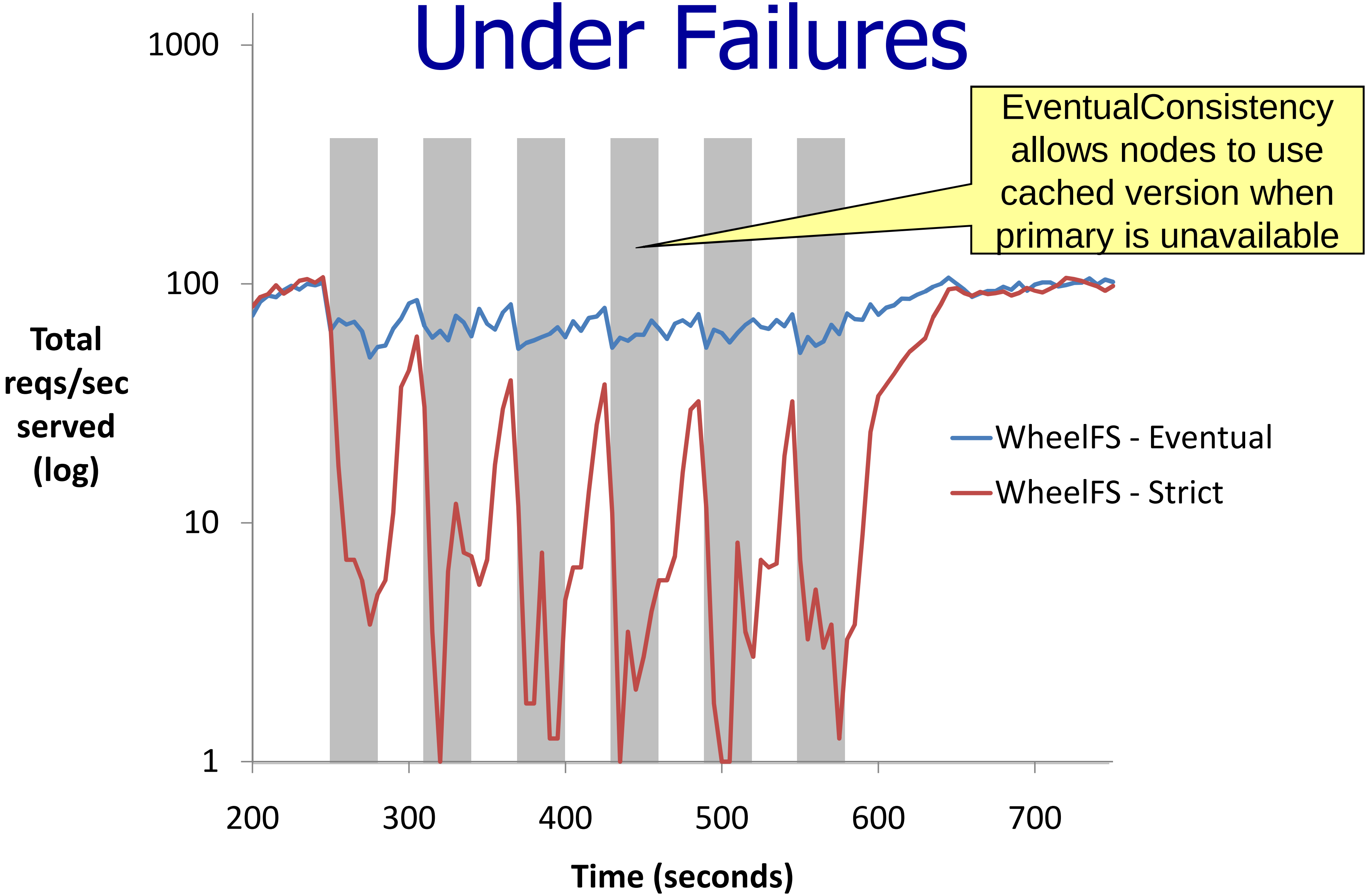
Total reqs/unique page: > 32,000
Origin reqs/unique page: 1.5 (CoralCDN) 2.6 (WheelFS)

CWC Failure Evaluation

- 15 proxies at 5 wide-area sites on Emulab
- 1 client per site
- Each minute, one site offline for 30 secs
  - Data primaries at site unavailable
- Eventual vs. strict consistency

Hypothesis: WheelFS using eventual consistency will achieve better performance during failures than WheelFS with strict consistency.

EC Improves Performance Under Failures



WheelFS Status

- Source available online

  `http://pdos.csail.mit.edu/wheelfs`

- Public PlanetLab deployment
  - PlanetLab users can mount shared storage
  - Usable by apps or for binary/configuration distribution

Related File Systems

- Single-server FS: NFS, AFS, SFS
- Cluster FS: Farsite, GFS, xFS, Ceph
- Wide-area FS: Shark, CFS, JetFile
- Grid: LegionFS, GridFTP, IBP, Rooter

- WheelFS gives applications control over wide-area tradeoffs

Storage Systems with Configurable Consistency

- PNUTS [VLDB '08]
  - Yahoo!'s distributed, wide-area database
- PADS [NSDI '09]
  - Flexible toolkit for creating new storage layers

- WheelFS offers broad range of controls in the context of a single file system

Conclusion

- Storage must let apps control data behavior
- Small set of *semantic cues* to allow control
  - **Placement, Durability**, **Large reads** and **Consistency**
- WheelFS:
  - Wide-area file system with semantic cues
  - Allows quick prototyping of distributed apps

`http://pdos.csail.mit.edu/wheelfs`