Peer to Peer III

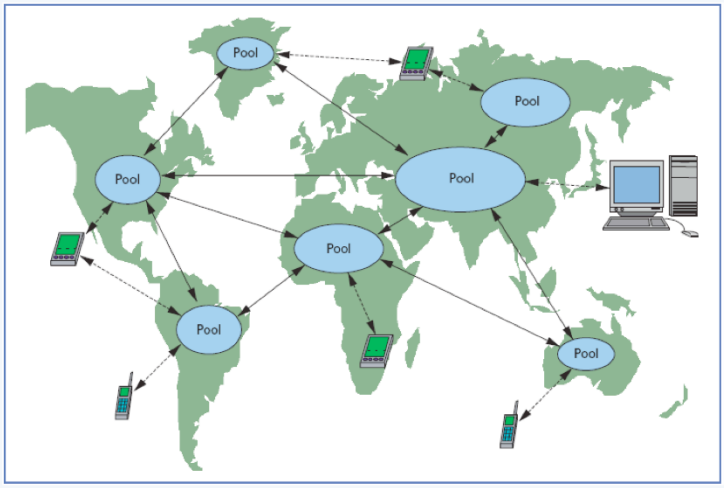
Building a Global Storage Service

Today

- How to build a global data storage utility
  - Pooling resources from millions of devices that
    - Are not trusted
    - Come and go
    - Are independent (no centralized management)
- Target
  - 10^{10} users x 10^4 files of 10^4 bytes each = 10^{18} bytes

What do we need if we want secure, durable, consistent, and efficient access to mutable files?

OceanStore



The slide is taken from “Maintenance-Free Global Data Storage,” by S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, published in IEEE Internet Computing, September-October 2001. Other OceanStore papers can be found at http://oceanstore.cs.berkeley.edu/publications/index.html. It is also covered in the textbook starting on page 422.

Goals

- Data lasts forever
- Data is tamper-proof and private
  - subject to user-specified authorization
- Access is generally quick

Issues

- What is the unit of replication
  - File or block?
- Finding a nearby copy
- Updating all copies
  - consistently
- Access control
- Integrity
- Fault tolerance

OceanStore

- Infrastructure
  - lots of administrative domains
  - servers trusted in aggregate, but not individually
    - arbitrary passive failures (crashes)
    - arbitrary active failures
      - really smart and malicious people and computers out there …
      - all communication is subject to eavesdropping and disruption

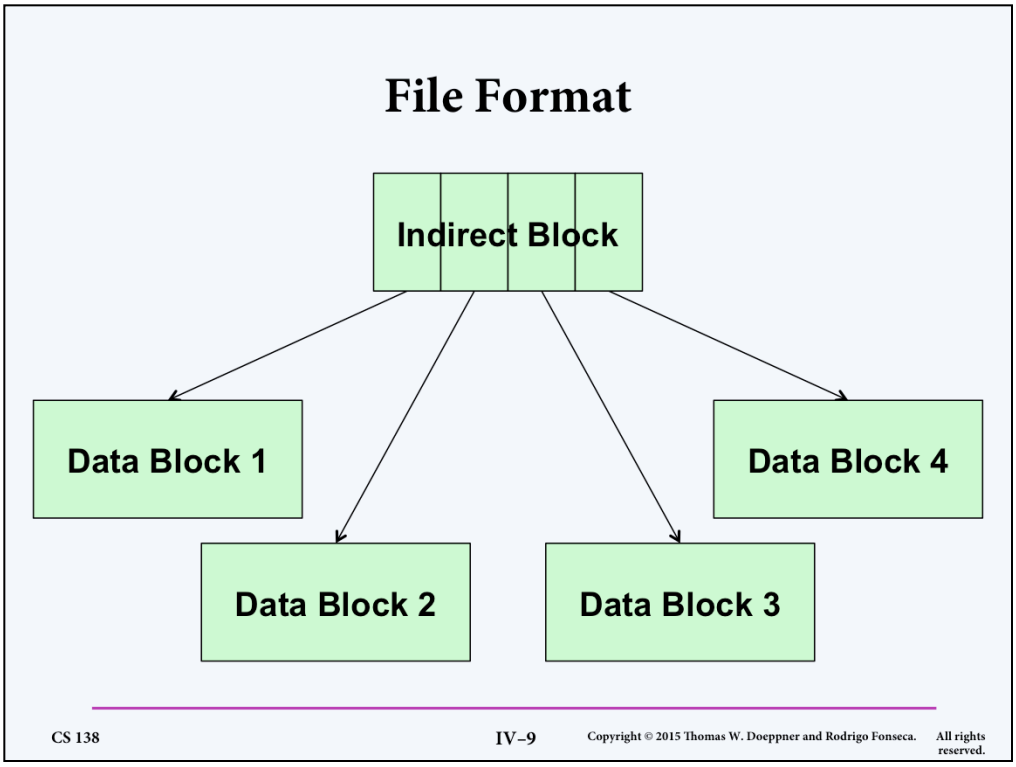
OceanStore

- Designed and developed at Berkeley, early 2000's
- 5 Key Techniques
  - End-to-end encryption, *self-certifying* data
  - Tapestry self-organizing routing infrastructure
  - Erasure coding for durability (m-of-n)
  - Byzantine update commitment
  - Dynamic replica management
- Great use of a DOLR (Tapestry)
- *Teaser* for many other techniques we will see throughout the semester

Caveats

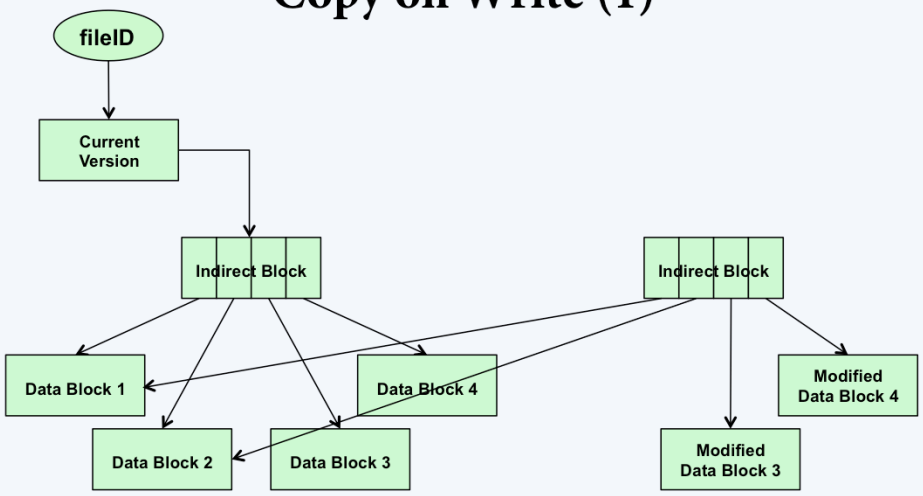
- It's far more complex than PuddleStore!
- This presentation is based on a number of OceanStore papers
  - not everything is totally clear

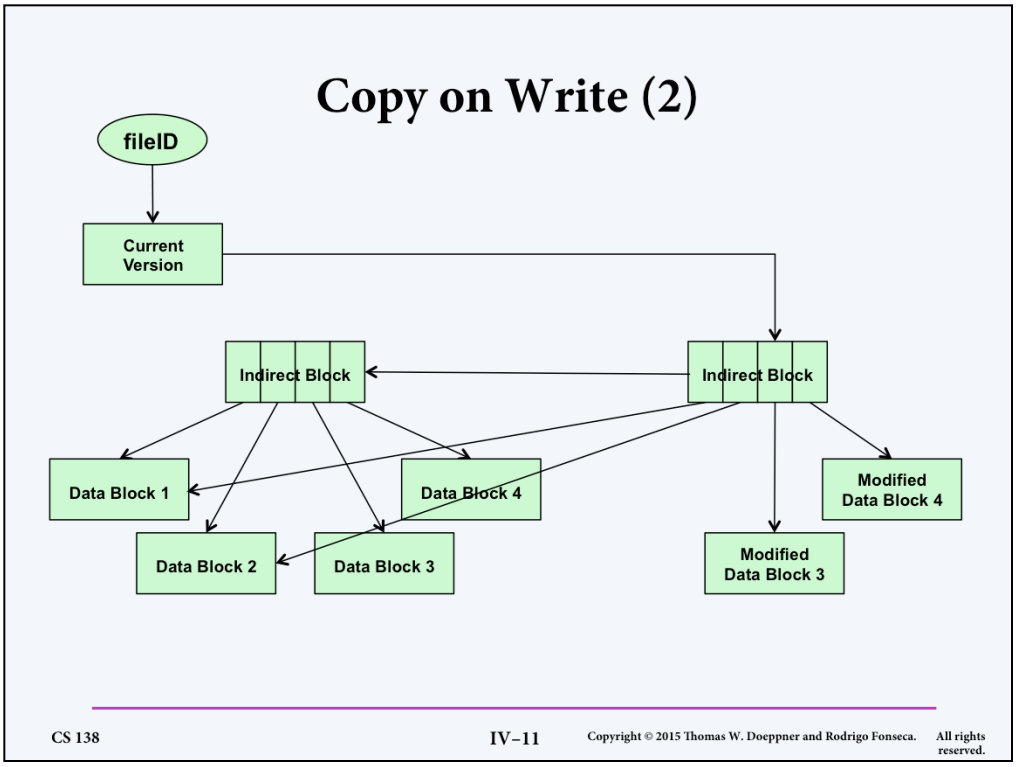
Pond: the OceanStore Prototype, Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Appears in Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03), March 2003



This and the next two slides are from the first lecture. They illustrate (roughly) the logical layout of files in OceanStore. The actual file layout is a form of B-tree, but such details aren't important here.

Copy on Write (1)





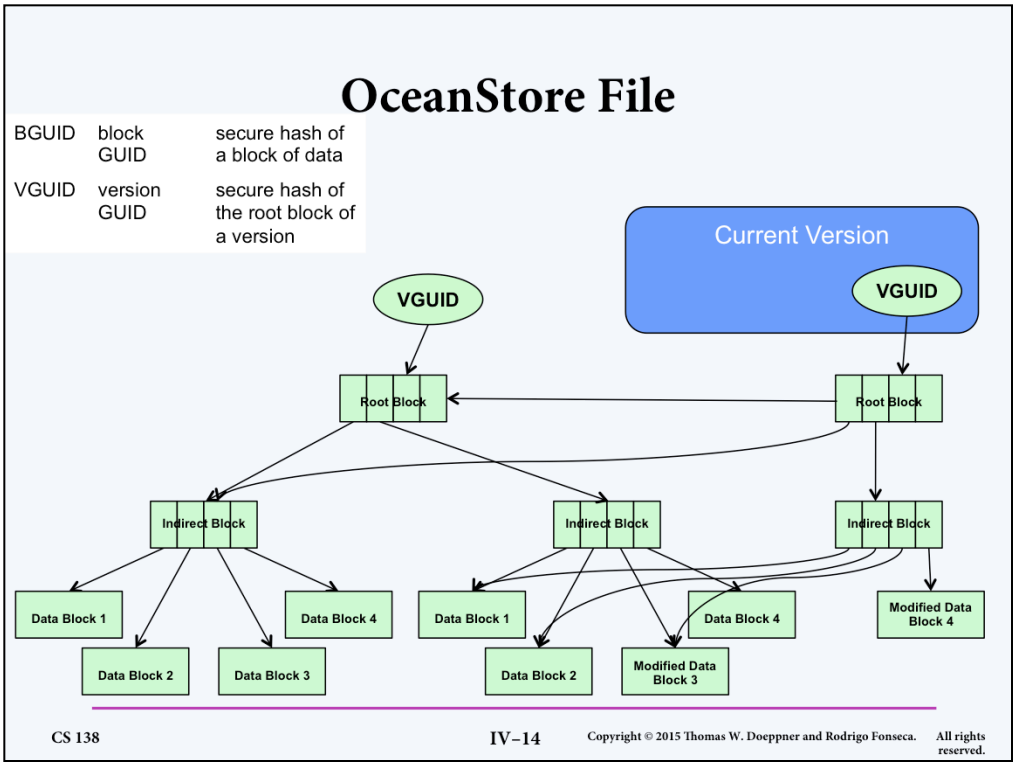
A file is a sequence of versions. One modifies a file by creating a new version. Blocks not modified in creating a new version are shared with the old version. In principle, versions last forever.

Questions

- Where do the blocks go?
- How are they referenced?
- What if we can't trust the computers that hold them?
- What if there are communication failures?

The Answer

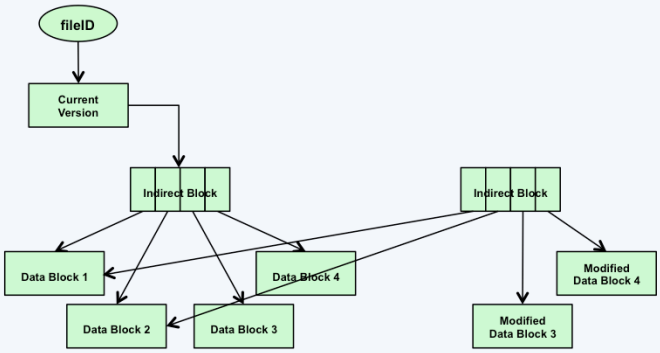
- Tapestry
  - blocks are identified by *secure hashes of their contents*
  - contents may be encrypted if necessary
  - multiple copies may be published for redundancy
- Self-certifying blocks
  - What happens if the the block changes?



GUID = globally unique identifier. Each link is a GUID. The intent is that GUIDs are published in Tapestry.

How do you find a file

- Need to go from file name to a record
- Actually just the pointer to the latest version
  - Only mutable record!
  - Id is a hash of (file name, owner id)



Replication

- **Example with 2 replicas**
  - **N = total number of servers = 10^6**
  - **M = number of servers expected to be down = 10^5 (10%)**
- **What is the probability that at least one copy of a replicated block is available?**

$$\sum_{i=0}^{1} \frac{\binom{M}{i}\binom{N-M}{2-i}}{\binom{N}{2}}$$

- **= 0.99**

Note that there are two copies of each block.
i is the number of unavailable copies
This is, for all possible numbers of unavailable copies that still allow reading,
the product of (# ways to arrange unavailable copies in unavailable servers) x (# of ways to arrange available copies in available servers) / # of ways to arrange all copies on all servers

For this particular case, the formula would be simpler if you consider the complement case: what is the probability that NO blocks are available? However, we use this formula as it is the same as the one in the next slide.

This and the next two slides are based on material from "Erasure Coding vs. Replication: A Quantitative Comparison," by H. Weatherspoon and J. Kubiatowicz, published in *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, March 2002. A copy may be found at http://oceanstore.cs.berkeley.edu/publications/papers/pdf/erasure_iptps.pdf.

Erasure Codes

- Divide an object into *m* equal-size fragments and code them as *n* equal-size fragments, *n>m*
  - *m/n = rate of encoding = r*
  - *1/r* = multiplicative storage increase
- Original object can be reconstructed from any m out of n fragments
  - handles *n-m* "erasures"

Benefits of Erasure Codes

- N = total number of servers = 10^6
- M = number of servers expected to be down = 10^5
- n = 32, m = 16
- What is the probability that at least m fragments of an erasure-coded block is available?

$$\sum_{i=0}^{n-m} \frac{\binom{M}{i}\binom{N-M}{n-i}}{\binom{N}{n}}$$

- = 0.999999998

In Practice

- Original block is stored in Tapestry by its BGUID
- Erasure-coded fragments stored by id *f*(BGUID, *k*), where k is the block number
- Can be replicated at will
- These are called *archival copies*

The Down Side …

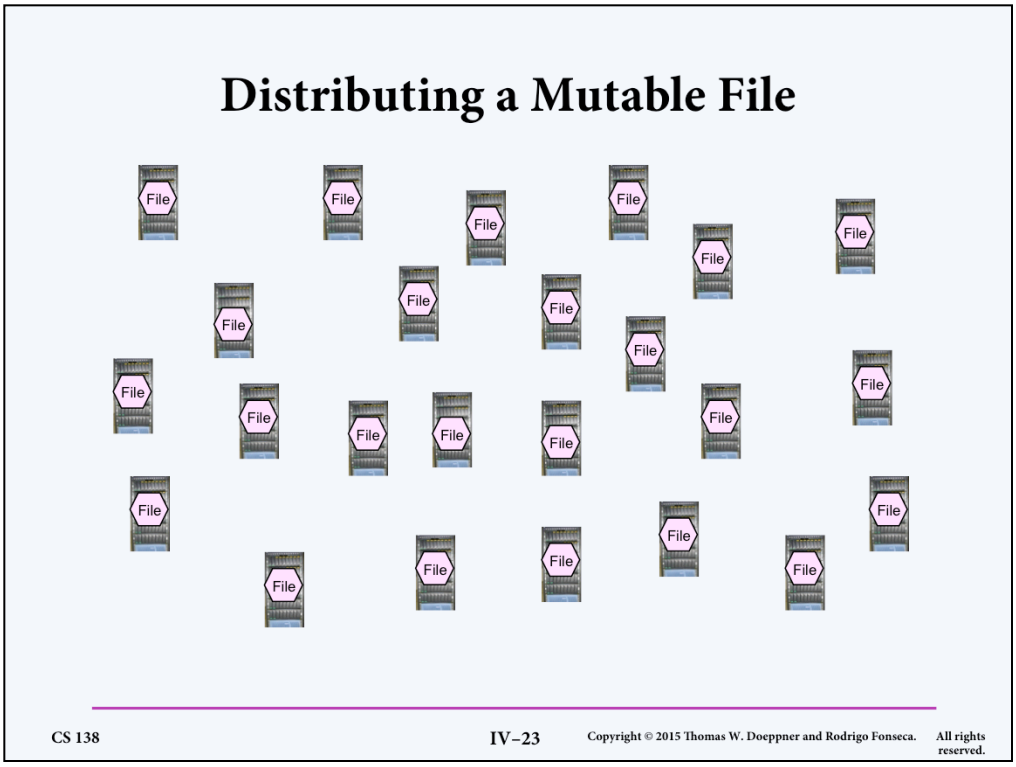
- Fetching and assembling the fragments is expensive
  - how can it be made cheaper (on average)?

Caching

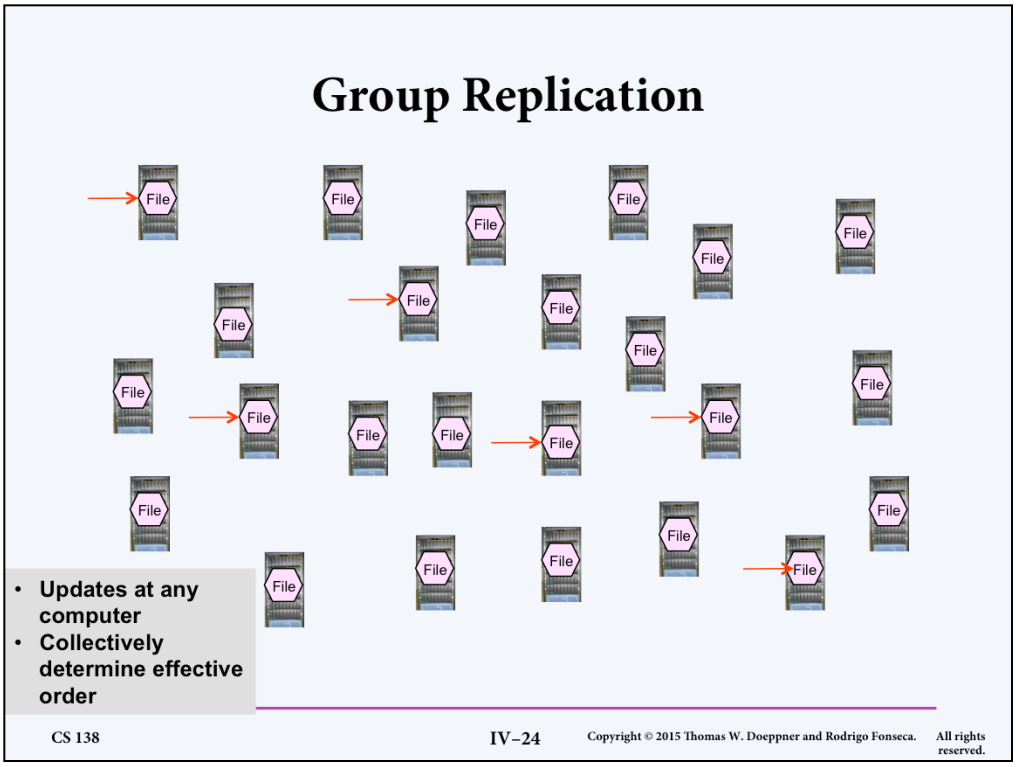
- If you've reconstructed a block
  - publish it in tapestry
  - available for others until you delete it

Where Are We?

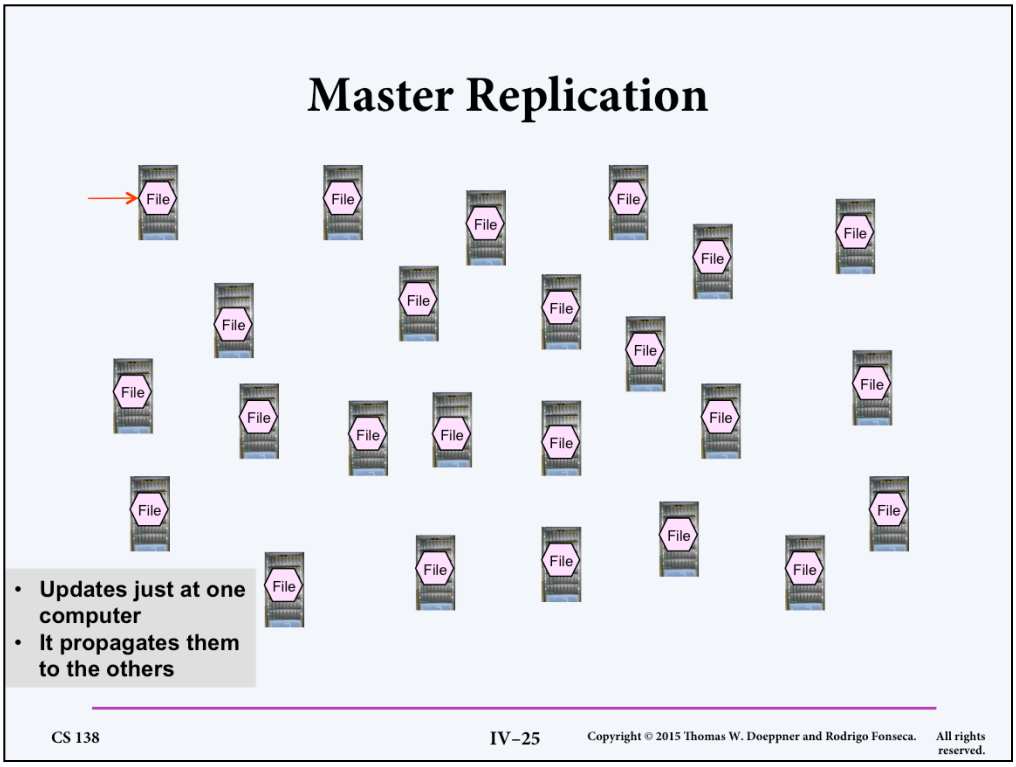
- *h*(file name, owner id) -> AGUID -> VGUID -> blocks…
- Blocks are widely replicated using erasure coding
  - fragments form the archival copy of the file
- Caching takes place as file is accessed



Here we pause to look at options for replicating a mutable file



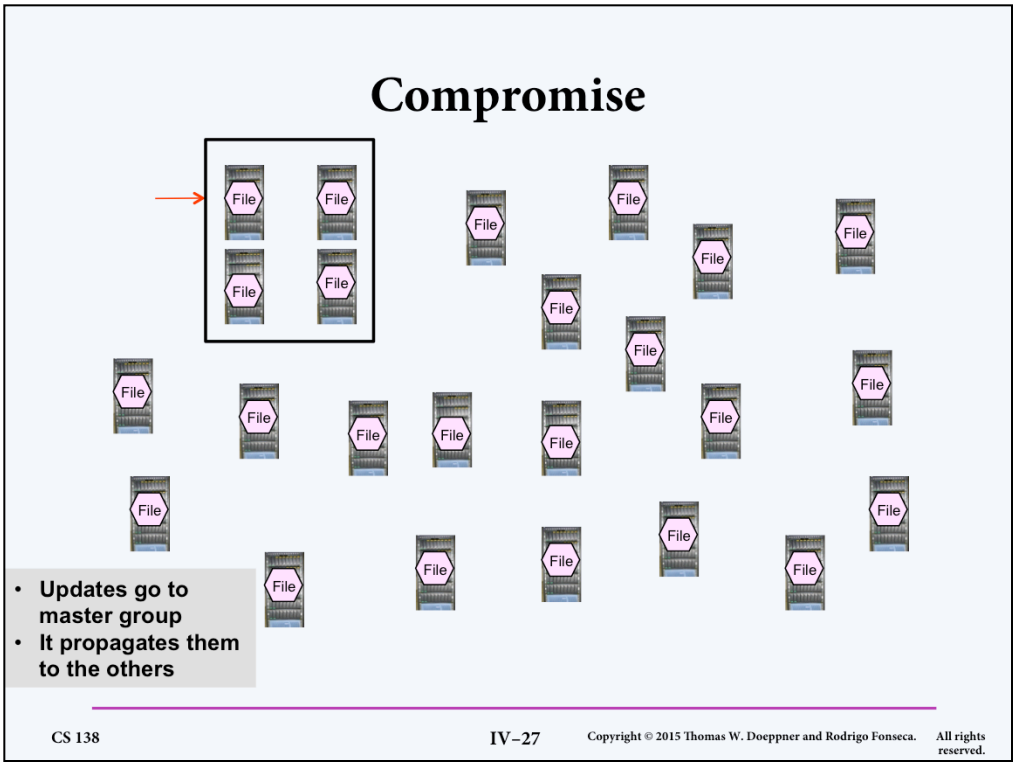
With group replication, updates may be sent to any member of the group (where the group here is all computers). This is convenient, except that getting all computers to agree on the order of updates could be a problem. We will study approaches for doing this, but they don't scale to hundreds of thousands of computers.



With master replication, updates are sent just to one computer, which then propagates them to all the others, determining the effective order. However, now the system has a bottleneck.

Issues

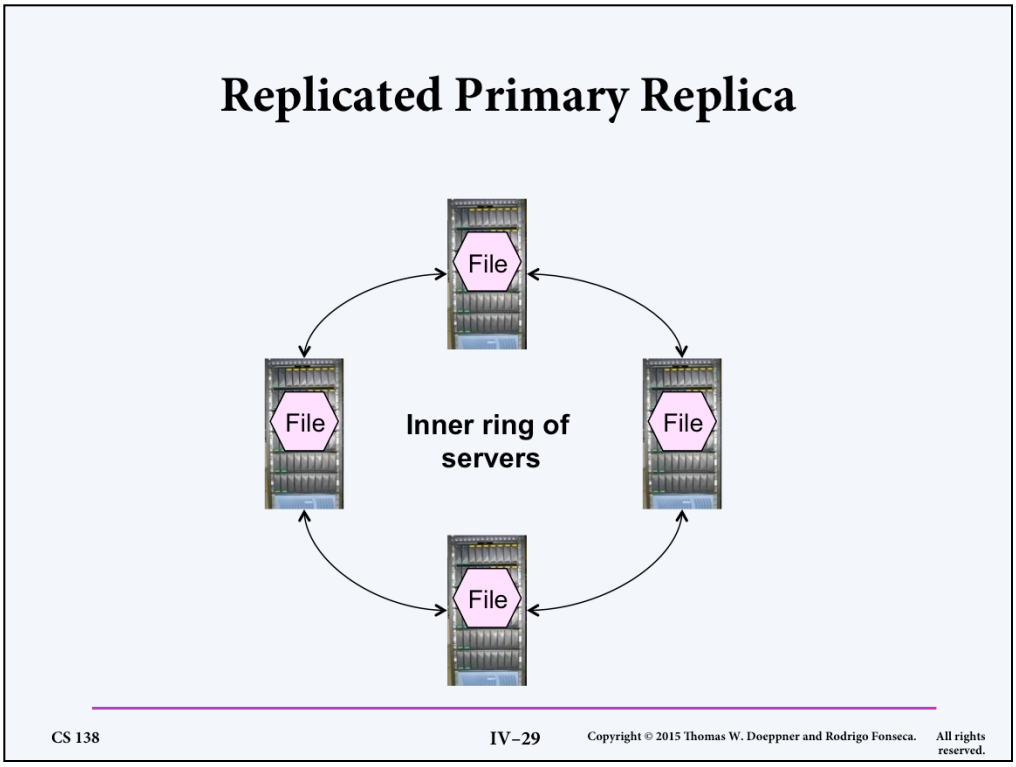
- Should we trust the server holding the primary copy?
  - should it be special?
    - No!



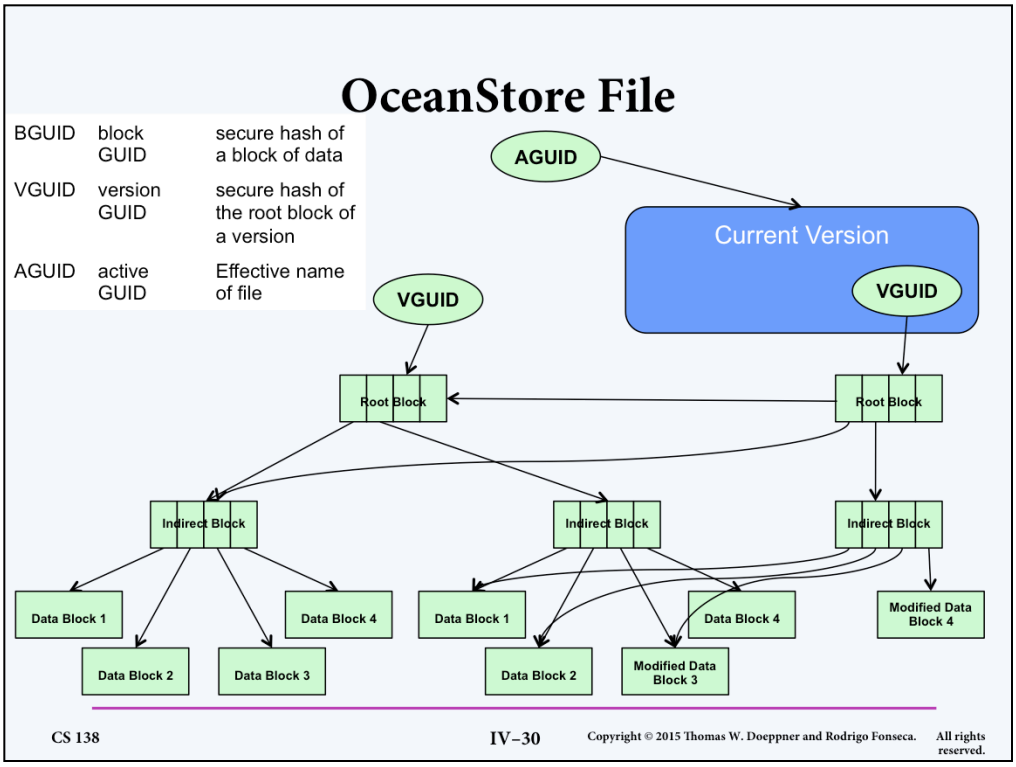
With this approach, the order of updates is determined by the master group, which then propagates them to the others.

Inner Ring

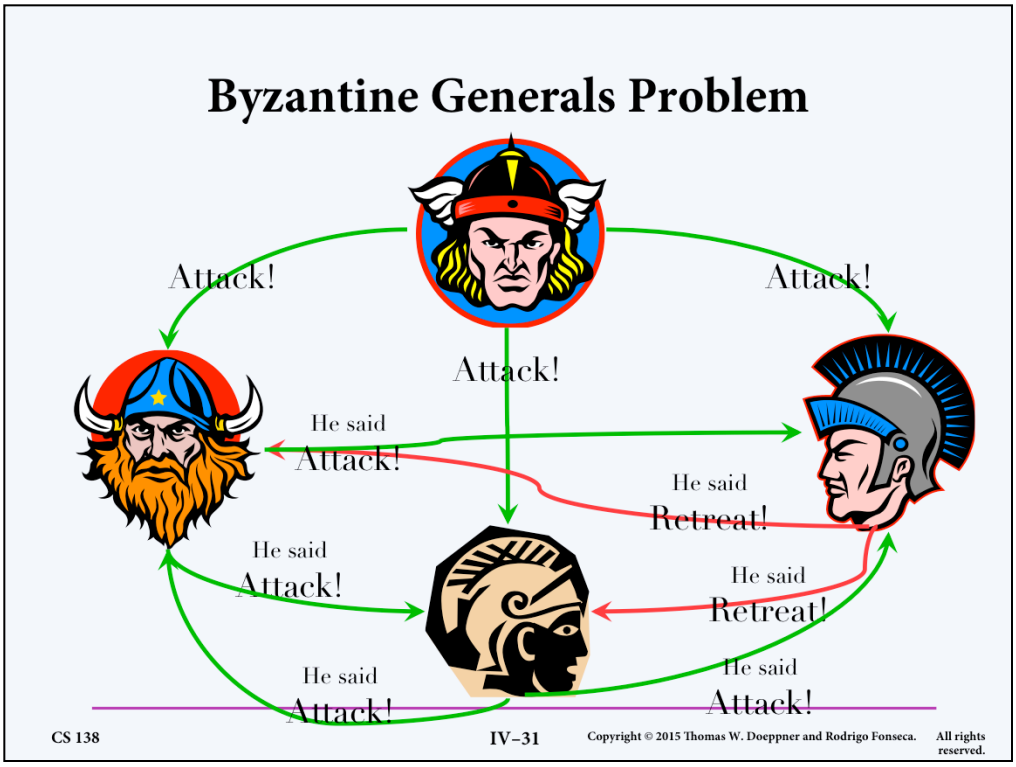
- A collection of servers are responsible for the primary copy
  - called the *inner ring*
- Collectively make updates
  - together they hold the *primary replica*
- File is identified by its *active GUID (AGUID)*
  - secure hash of application-specific name and owner's public key
- Each inner-ring server publishes the AGUID in Tapestry
  - each holds a copy of the current VGUID



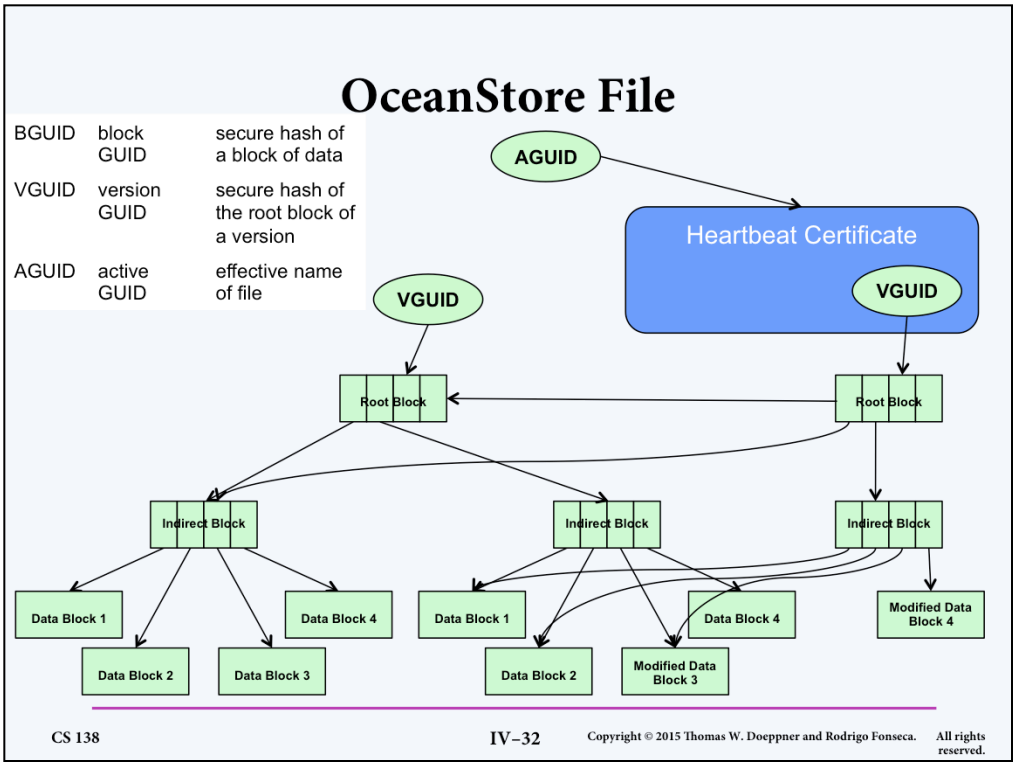
The primary replica's function is too important to be trusted to a single server. Thus all its actions must be agreed upon by a set of servers known as the *inner ring*.



Copies of the current-version information are found via the file's AGUID, which serves to identify the file.



The servers in a file's inner ring must agree on all updates to the file, and hence must agree on its current-version information. Getting agreement among a group of servers, some of which may be (maliciously) faulty, is known as the Byzantine Generals Problem. We take this up in a later lecture. However, we will see that it is solvable if more than two-thirds of the participants do what they're supposed to do.



The current-version information agreed upon by the inner ring is called the *heartbeat* and is represented as a security certificate, signed by at least one more than two-thirds of the inner-ring servers. As we will discuss in a later lecture, clients, upon getting a copy of such a certificate, can verify that it really was signed by such super majority. In addition to the latest VGUID, the signed heartbeat contains the AGUID, a timestamp, and a version sequence number.

Almost There …

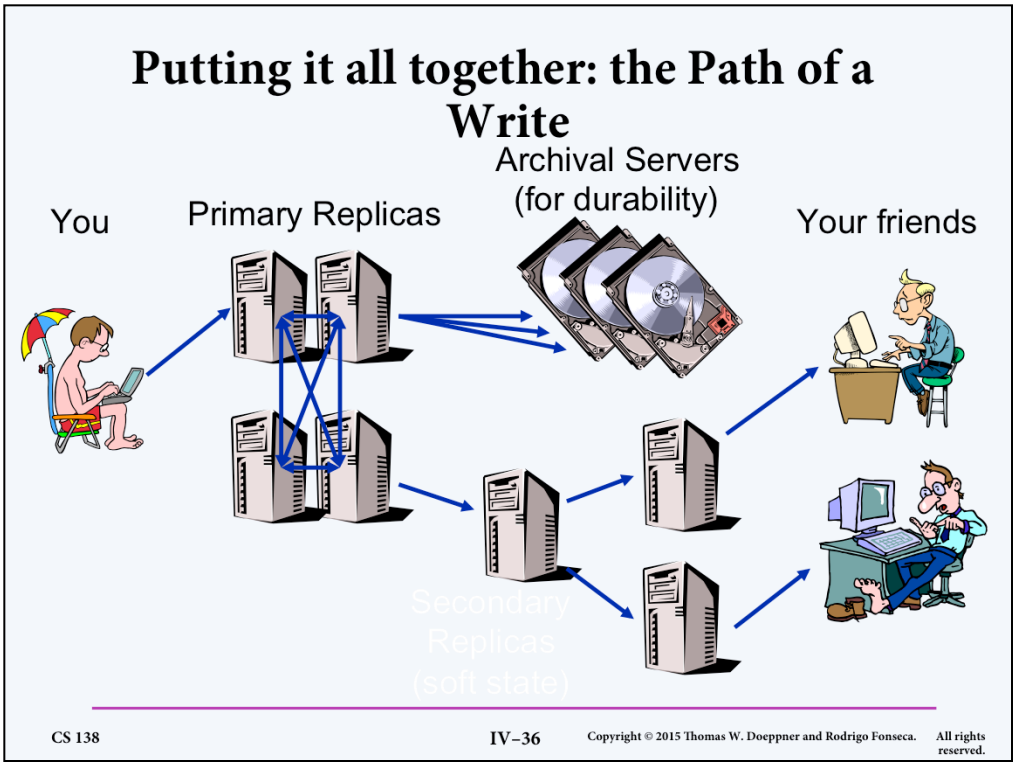
- What about:
  - getting notified about file updates?
  - finding the AGUID in the first place?

Secondary Replicas

- May be stored on any server
- Hold copies of heartbeat
  - new copies pushed to them by inner-ring servers
    - new secondary replicas find and link to existing ones via Tapestry
    - forms tree of replicas

Where Are We?

- File update
  - clients send update requests to primary replica
  - inner-ring machines come to agreement on update order
    - commit changes to local copies
    - propagate heartbeats to secondary replicas
    - fragment new blocks and add to archival servers using erasure code



This slide taken from the original presentation of “Pond: the OceanStore Prototype”, Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Appears in Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03), March 2003,

Where Are We?

- File read
  - given a BGUID, block is found via Tapestry
    - first look up BGUID
    - if not found compute fragment GUIDs and look them up
      - verify results
      - combine into block
  - to find latest version of file, contact primary replica (or, perhaps, secondary replica)

What Else?

- Not described here
  - all file blocks are encrypted by clients
  - directories
  - access control
- Not implemented in Pond
  - automatic repair
    - periodic verification of data
    - periodic copying of data to new disks
  - introspection
    - adaptive system management
- Not clear
  - management of inner rings

Performance

Phase	LAN Linux NFS	LAN Pond	WAN Linux NFS	WAN Pond	Predominant operations in benchmark
1	0.0	1.9	0.9	2.8	Read and write
2	0.3	11.0	9.4	16.8	Read and write
3	1.1	1.8	8.3	1.8	Read [status only]
4	0.5	1.5	6.9	1.5	Read
5	2.6	21.0	21.5	32.0	Read and write
Total	4.5	37.2	47.0	54.9	

Performance vs NFS
Writes dominated by cryptographic operations (group signing), erasure coding
This is a killer in the local area
Latency in the wide area hides most of this overhead
In conclusion, it is feasible!

This slide is taken from the textbook (Coulouris, Dollimore, and Kindberg, 4th edition) and shows the performance of Pond vs. NFS. The text of the figure, also taken from the textbook, is: “The figures show times in seconds to run different phases of the Andrew benchmark. It has five phases: (1) creates subdirectories recursively; (2) copies a source tree; (3) examines the status of all the files in the tree without examining their data; (4) examines every byte of data in all the files; and (5) compiles and links the files.”