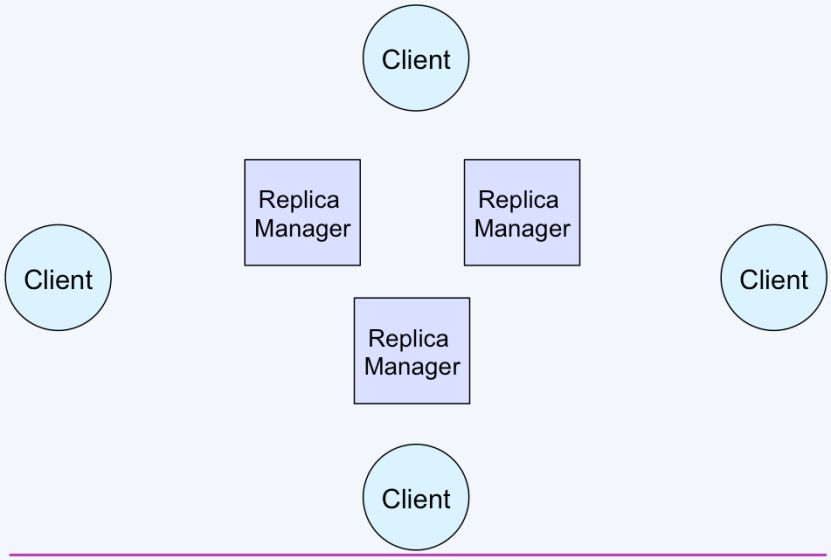
Distributed File Systems (Part 1)

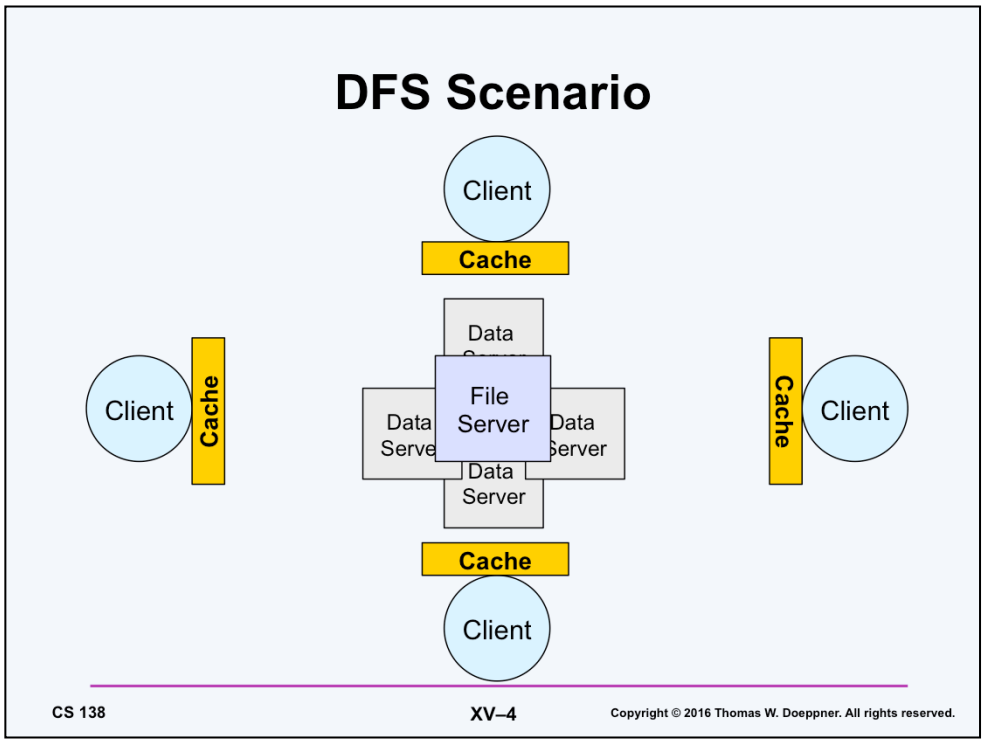
Outline

- Basic concepts
- NFS version 2
- CIFS
- DCE DFS
- NFS version 4
- CS Department: NFS + CIFS + GPFS

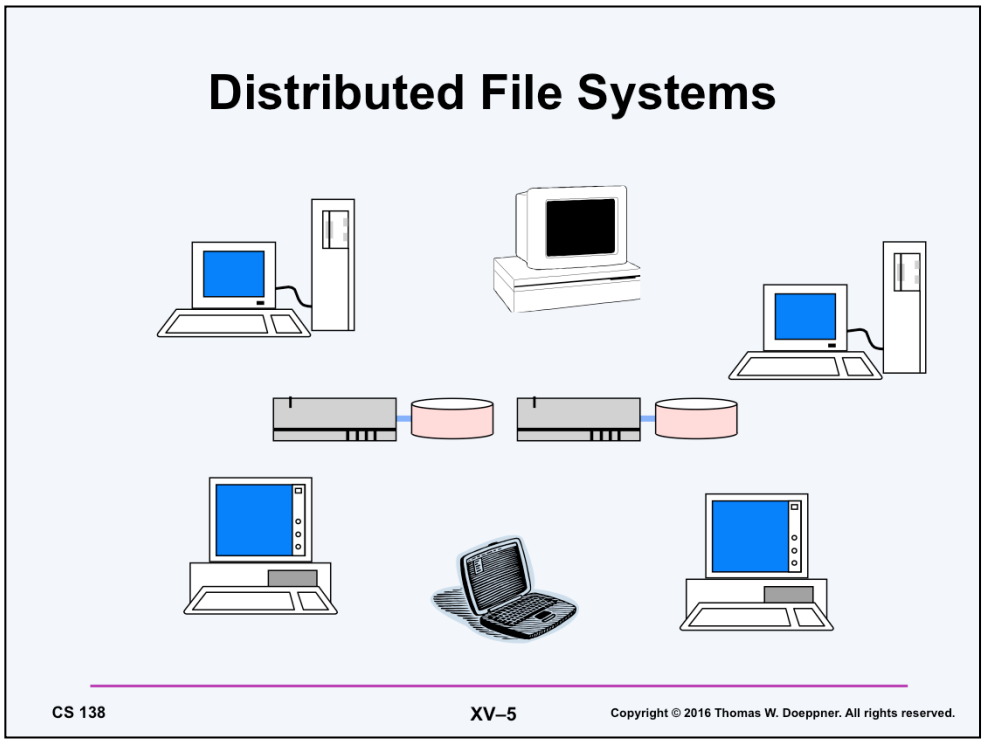
This material is partially covered in Chapter 12 of Coulouris, Dollimore, Kindberg, and Blair.

Previous Scenario





Note that in the typical distributed file system, servers are not replicated. However, the data might be. In fact, one server, the “file server”, might handle meta data, and data might be handled on other servers.

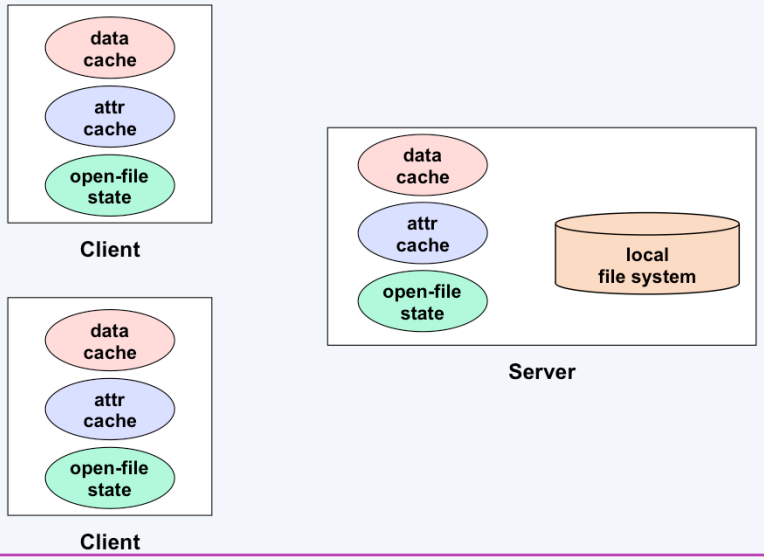


File systems are certainly important parts of general-purpose computer systems. They are responsible for the storage of data (organized as files) and for providing a means for applications to store, access, and modify data. Local file systems handle these chores on individual computers; distributed file systems handle these chores on collections of computers. In the typical design, distributed file systems provide a means for getting at the facilities of local file systems. What is usually desired of a distributed file system is that it be access transparent: programs access files on remote computers as if the files were stored locally. This rules out approaches based on explicit file transfer, such as the Internet's FTP (file transfer protocol) and Unix's rcp (remote copy).

DFS Components

- Data state
  - file contents
- Attribute state
  - size, access-control info, modification time, etc.
- Open-file state
  - which files are in use (open)
  - lock state

Possible Locations



In Practice …

- Data state
  - NFS
    - weakly consistent
    - less weak if program uses locks
  - CIFS and DFS
    - strictly consistent
- Lock state
  - must be strictly consistent

Thursday morning, November 17th
At 7:00 a.m.

Maytag, the department's central file server, will be taken down to kick off a filesystem consistency check.

Linux machines will hang.

All Windows users should log off.

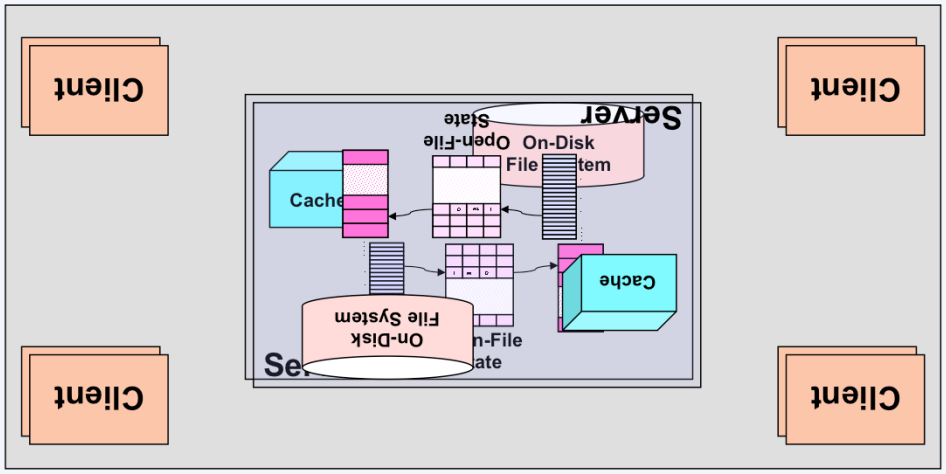
Normal operation will resume by 8:30 a.m. if all goes well.

All windows users should log off before this time.

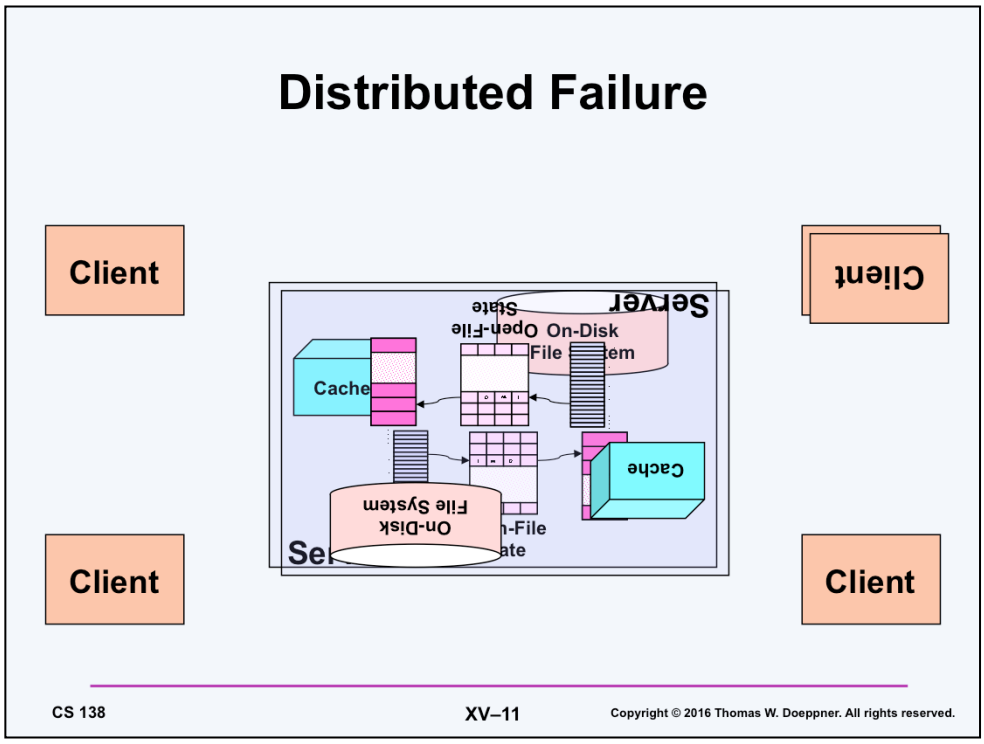
Questions/concerns to problem@cs.brown.edu

(Note that the November 17 in question was in 2005.)

Failures in a Local File System



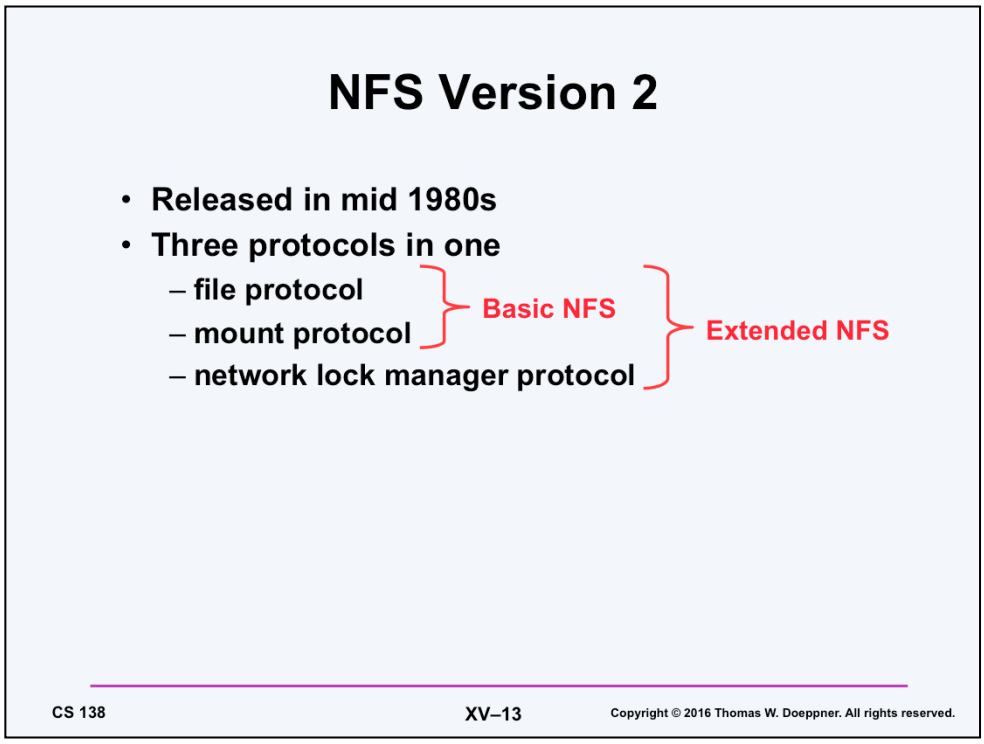
What we're accustomed to with local file systems is that, in the event of a crash, everything goes down. This is simple to deal with.



In a distributed system, if the server crashes, there is no inherent reason for clients to crash as well. Assuming there was no damage done to the on-disk file system, client processes might experience a delay while the server is down, but should be able to continue execution once the server comes back up, as if nothing had happened. The crash of a client computer is bad news for the processes running on that computer, but should have no adverse effect on the server or on other client computers. We'd like the effect to be as if the client processes on the crashed computer had suddenly closed all their files and terminated.

In Practice …

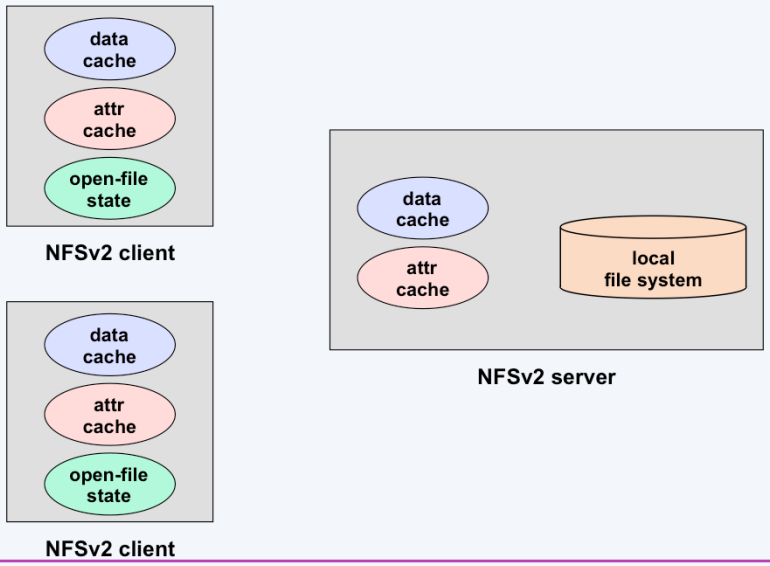
- NFS version 2
  - relaxed approach to consistency
  - handles failures well
- CIFS
  - strictly consistent
  - intolerant of failures
- DCE DFS
  - strictly consistent
  - sort of tolerant of failures
- NFS version 4
  - either relaxed or strictly-consistent
  - handles failures very well



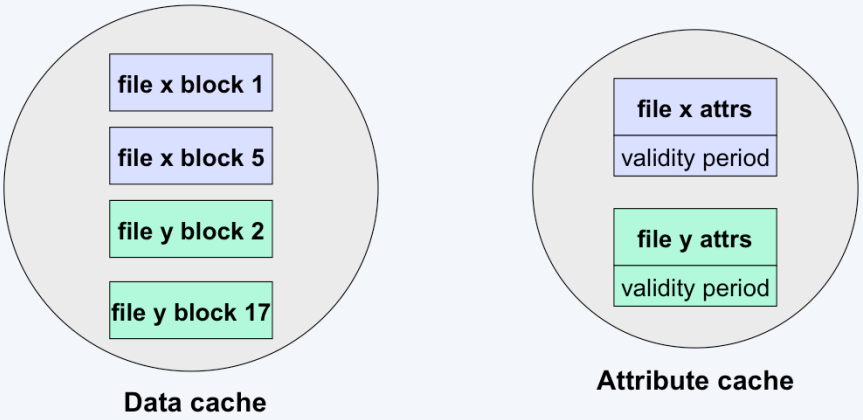
NFS consists of three component protocols: a *mount protocol* for making collections of files stored on servers available to client nodes, the *NFS file protocol* for accessing and manipulating files and directories, and a *network lock manager* (NLM) for locking files over the network and recovering lock state after failures.

An NFS server gives its clients access to one or more of its local file systems, providing them with opaque identifiers called *file handles* to refer to files and directories. Clients use a separate protocol, the *mount protocol*, to get a file handle for the root of a server file system, then use the *NFS protocol* to follow paths within the file system and to access its files, placing simple remote procedure calls to read them and write them. A third protocol, the *network lock manager protocol* (NLM), was added later and may be used, if desired, to synchronize access to files. All communication between client and server is with ONC RPC.

Distribution of Components

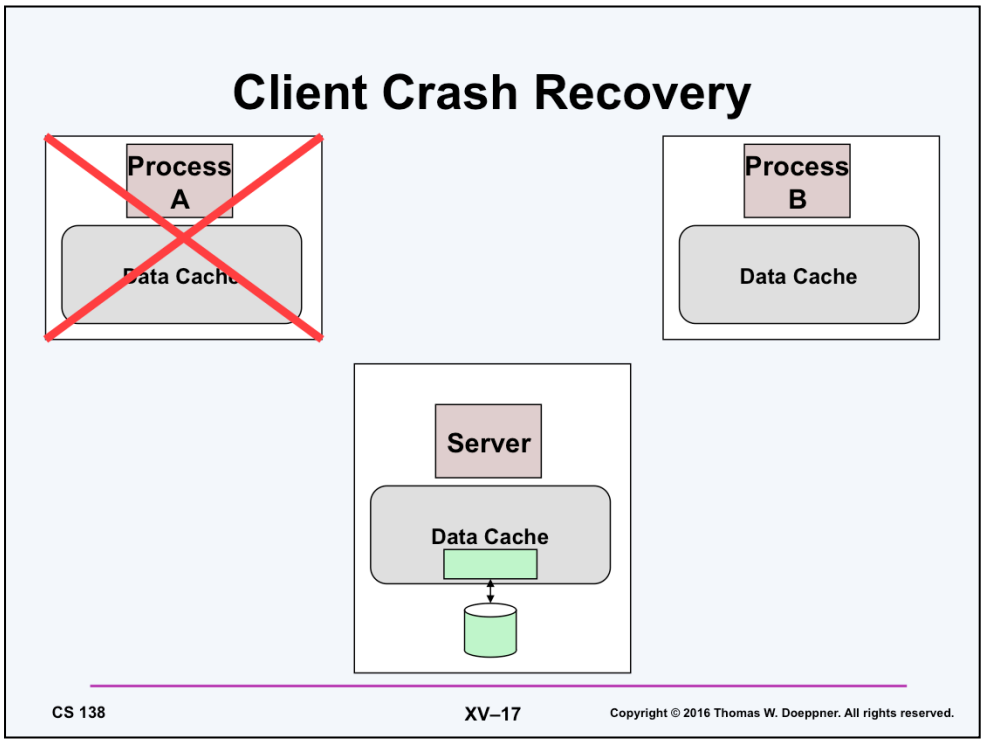


Consistency in Basic NFSv2

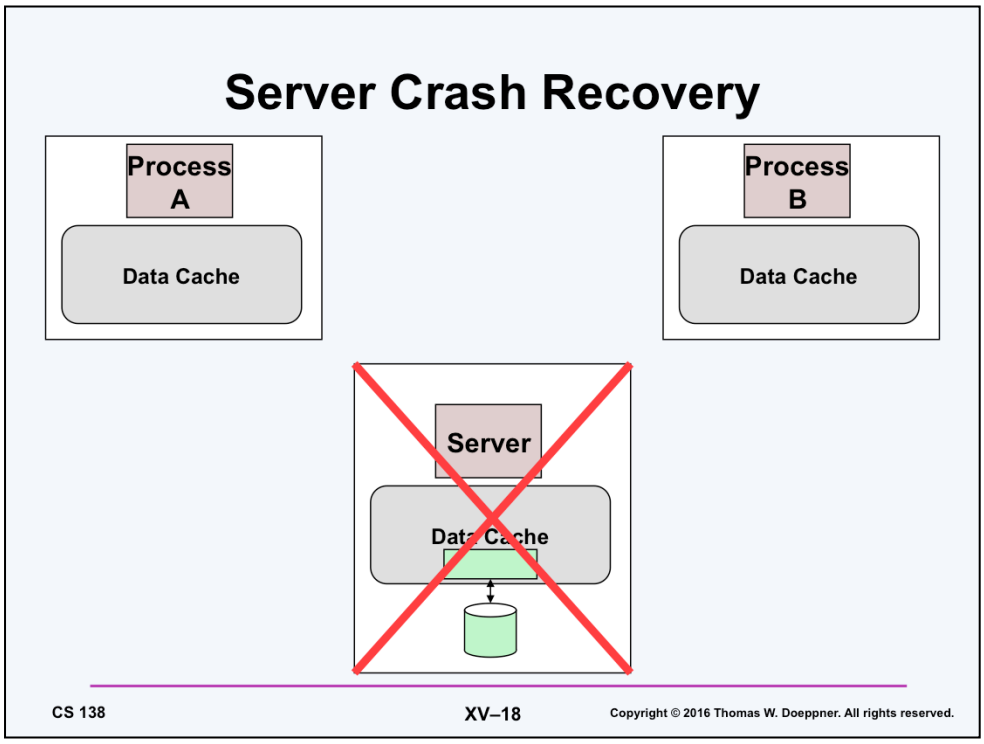


More …

- All write RPC requests must be handled synchronously on the server
- Close-to-Open consistency
  - client writes back all changes on close
  - flushes cached file info on open



Since NFS servers are stateless, the crash and restart of a client has no effect on them. Thus client crash recovery involves no actions on NFS's part.

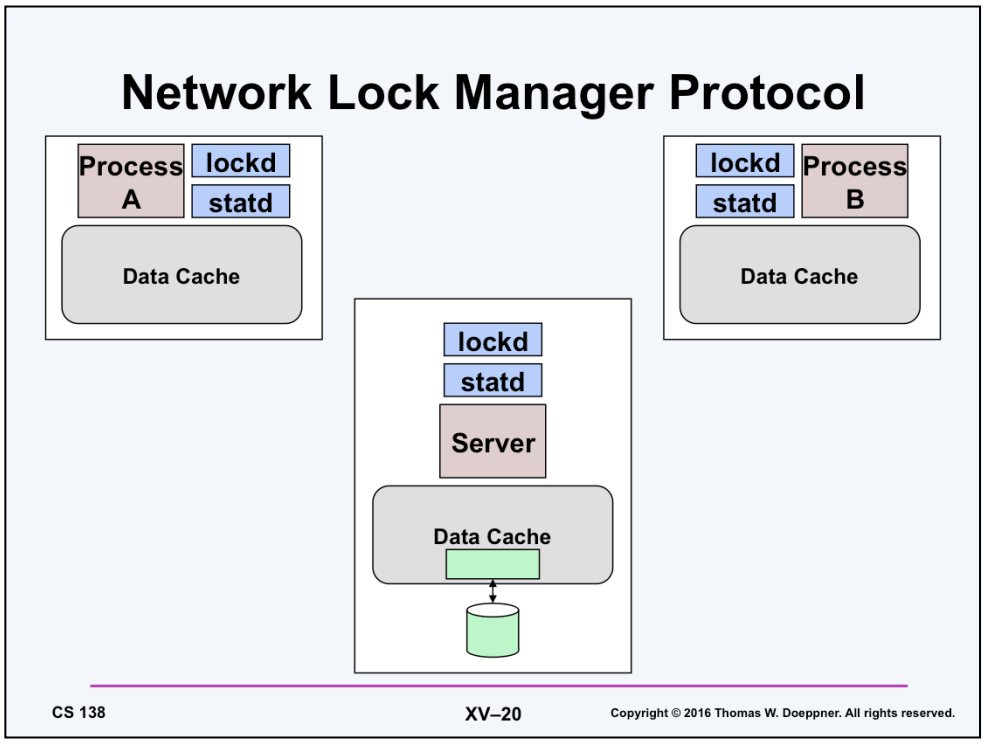


Recovery from a server crash is easy for the server — it merely resumes processing requests. The client is generally delighted when the server recovers; its concern is what to do while the server is down.

Client machines may choose to "mount" NFS file systems in one of two ways: soft mounts and hard mounts. With a soft mount, if the server crashes, then attempts to access the down file system fail — a "timed-out" error code is returned. This is often not terribly useful, since most applications are not equipped to deal with such problems. In the other approach (the hard mount) if the server crashes, then clients repeatedly re-try operations until the server recovers. This can try the patience of users, but it doesn't break applications.

File Locking

- State is required on the server!
  - recovery must take place in the event of client and server crashes



The stateless approach clearly doesn't work if one is concerned about locking files. For this, NFS employs a separate lock protocol. Each participating machine runs two special processes: a lock manager, typically known as the *lock daemon,* and a status monitor, typically known as the *status daemon*. The lock daemons manage the locking and unlocking of files; the status daemons help cope with crashes. When an application on a client machine attempts to lock a remote file, it contacts the local lock manager which places an RPC to the lock manager on the server, which locks it there.

If a client crashes, the server's status daemon unlocks all of the files that the client had locked. The only difficulty here is determining whether the client has crashed — perhaps it is merely slow. The approach used in NFS is that the client, when it comes back to life, announces to the server that it has been down. Of course, human intervention may be required if the client is down for a long period of time.

If a server crashes, it loses all knowledge of which files are locked and by whom. So, when it comes back up, it must ask the clients to tell it which files they had locked. Upon receipt of this information, it restores its original state and resumes normal operation.

NFS Version 3

- In use at Brown and in most of the rest of the world
- Basically the same as NFSv2
  - improved handling of attributes
  - *commit* operation for writes
  - various other things

CIFS

- Common Internet File System
  - Microsoft's distributed file system
- Features
  - batched requests and responses
  - strictly consistent
- Not featured …
  - depends on reliability of transport protocol
  - loss of connection == loss of session

History

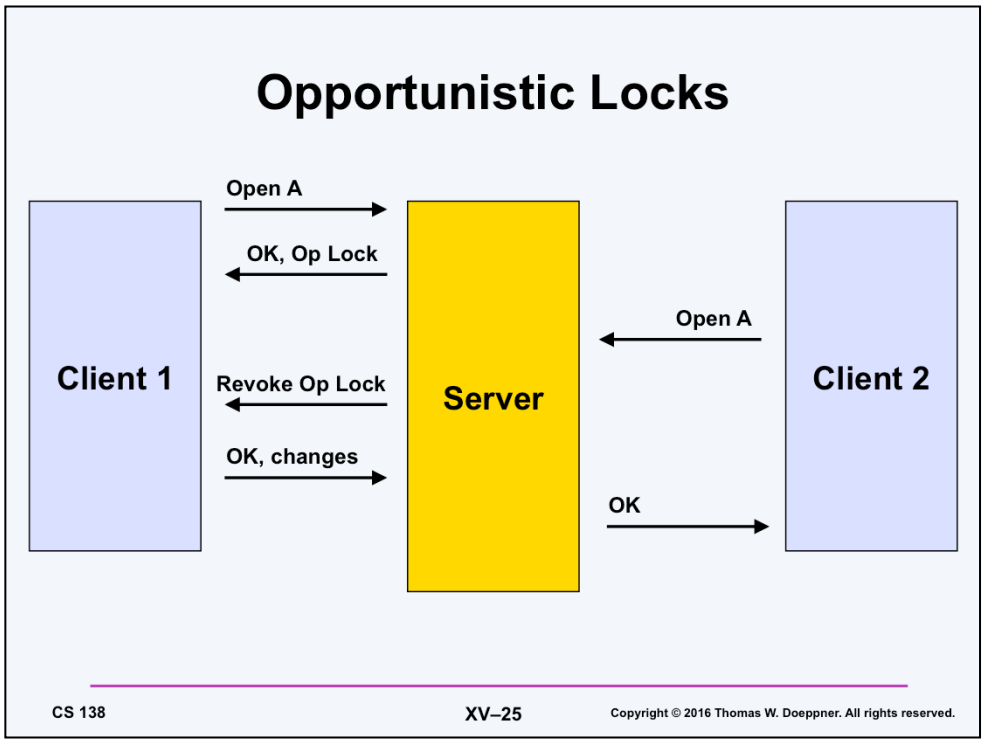
- Originally a simple means for sharing files
  - developed by IBM and called server message block protocol (SMB)
  - ran on top of NetBIOS
- Microsoft took over
  - renamed CIFS in late 1990s
  - uses SMB as RPC-like communication protocol
    - runs on NetBIOS
    - usually layered on TCP
    - sometimes no NetBIOS, just TCP

Consistency vs. Performance

- Strict consistency is easy …
  - … if all operations take place on server
  - no client caching
- Performance is good …
  - … if all operations take place on client
  - everything is cached on client
- Put the two together …

  ⌀

  - or you can do opportunistic locking



Opportunistic locks are used by CIFS.