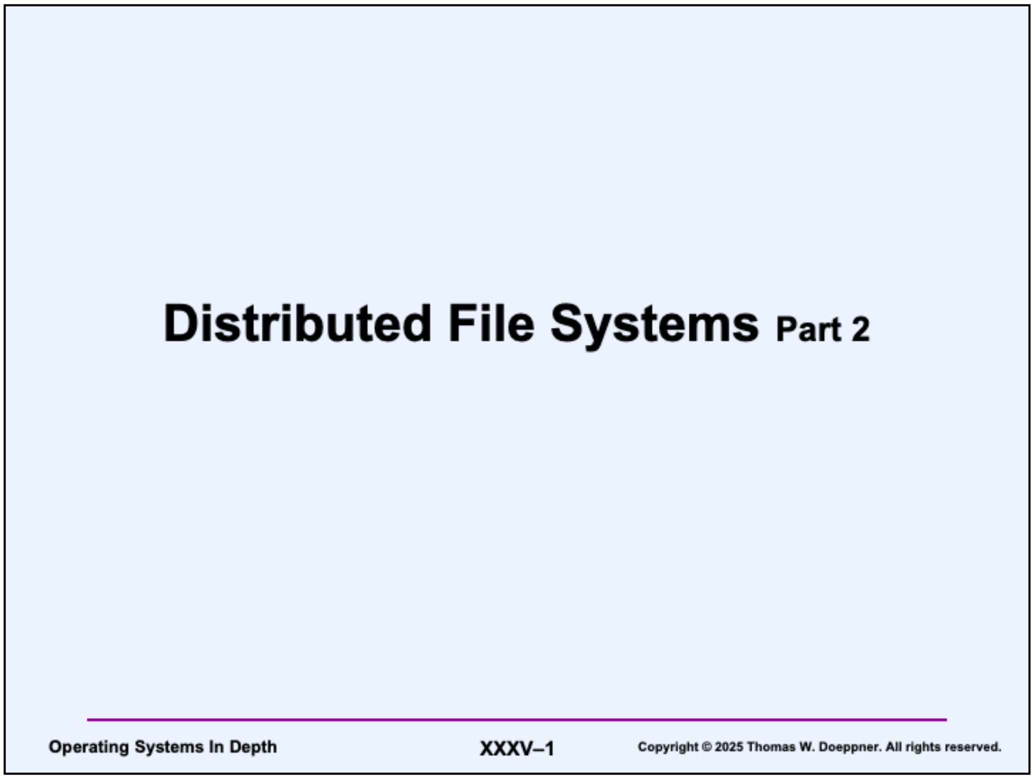
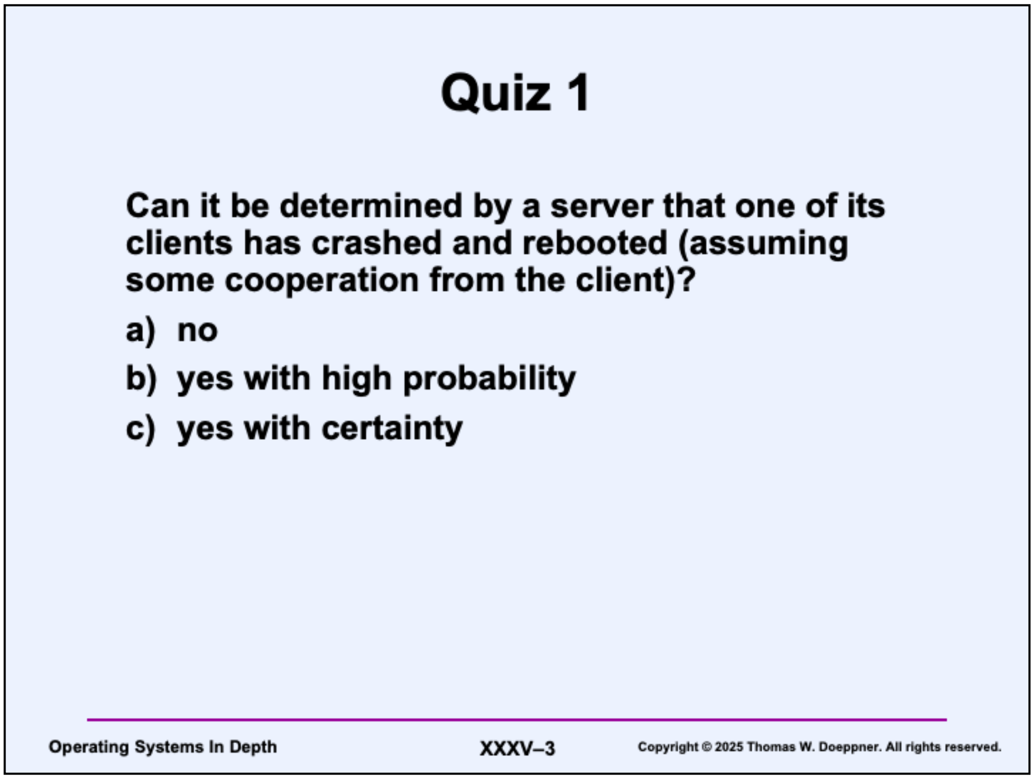


Distributed File Systems Part 2

File Locking

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



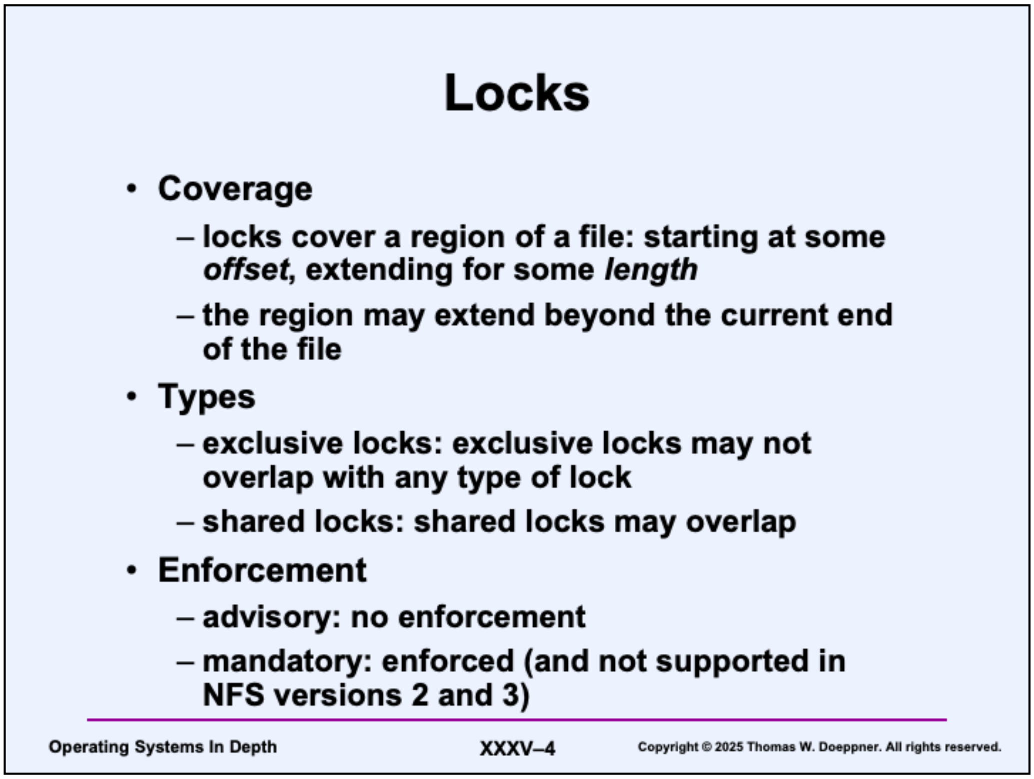
Quiz 1

Can it be determined by a server that one of its clients has crashed and rebooted (assuming some cooperation from the client)?

a) no

b) yes with high probability

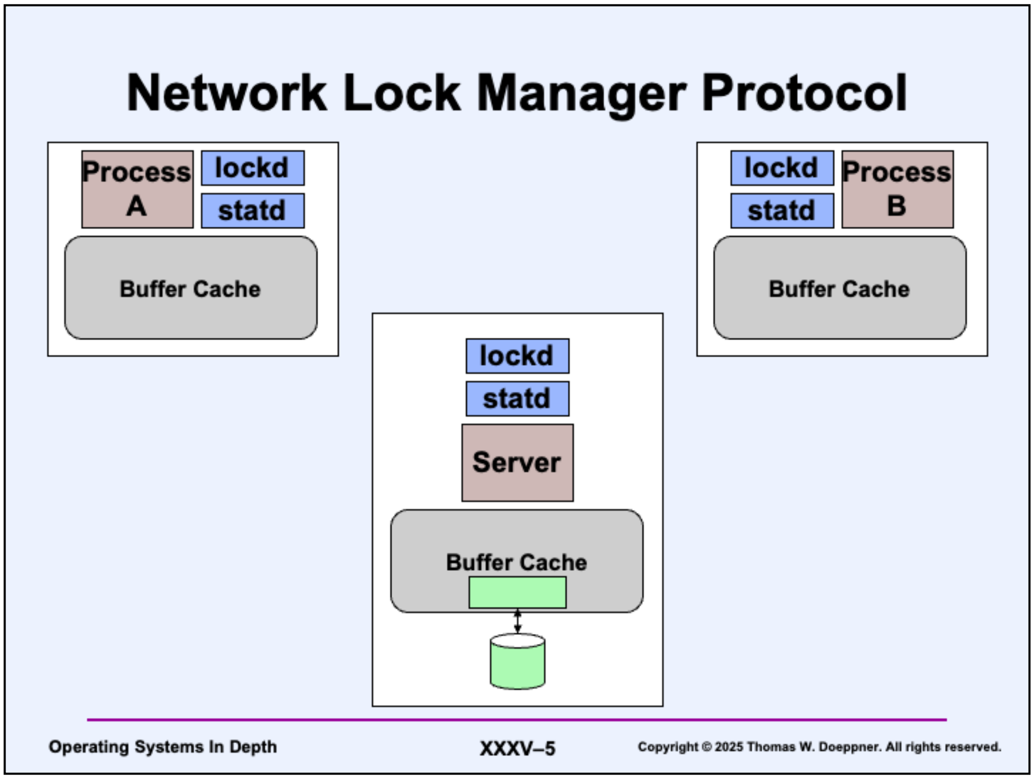
c) yes with certainty



Locks

- Coverage
  - locks cover a region of a file: starting at some *offset*, extending for some *length*
  - the region may extend beyond the current end of the file
- Types
  - exclusive locks: exclusive locks may not overlap with any type of lock
  - shared locks: shared locks may overlap
- Enforcement
  - advisory: no enforcement
  - mandatory: enforced (and not supported in NFS versions 2 and 3)

Exclusive locks are better known as write locks; shared locks are better known as read locks.



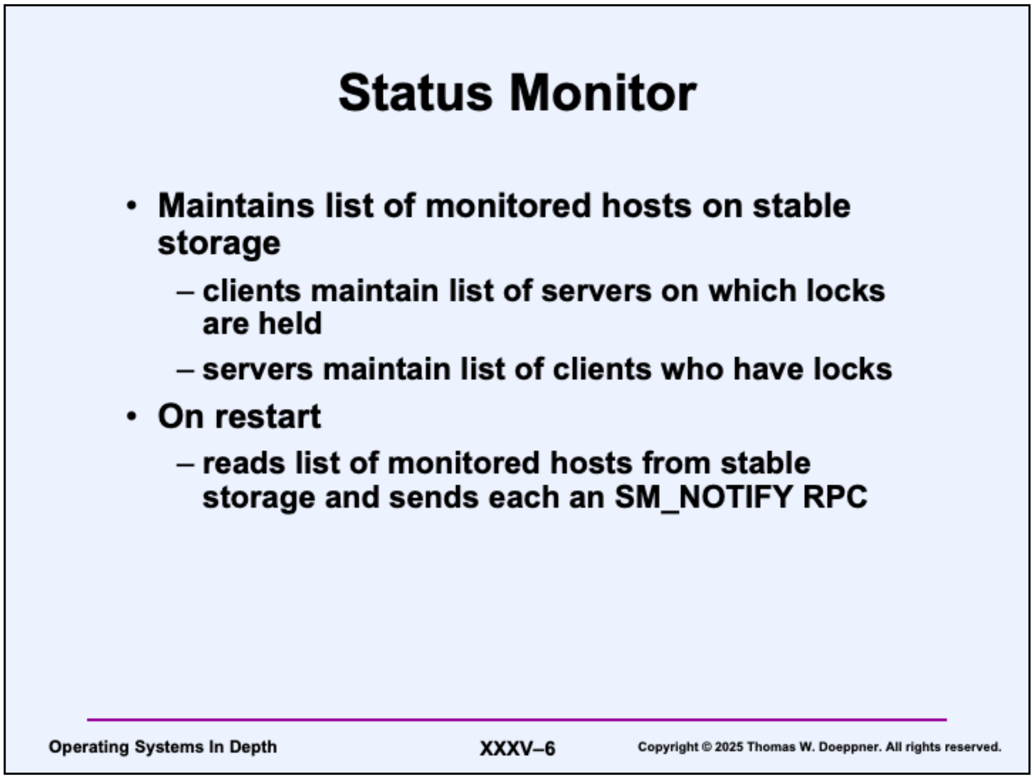
Network Lock Manager Protocol

Operating Systems In Depth XXXV–5 Copyright © 2025 Thomas W. Doeppner. All rights reserved.

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.

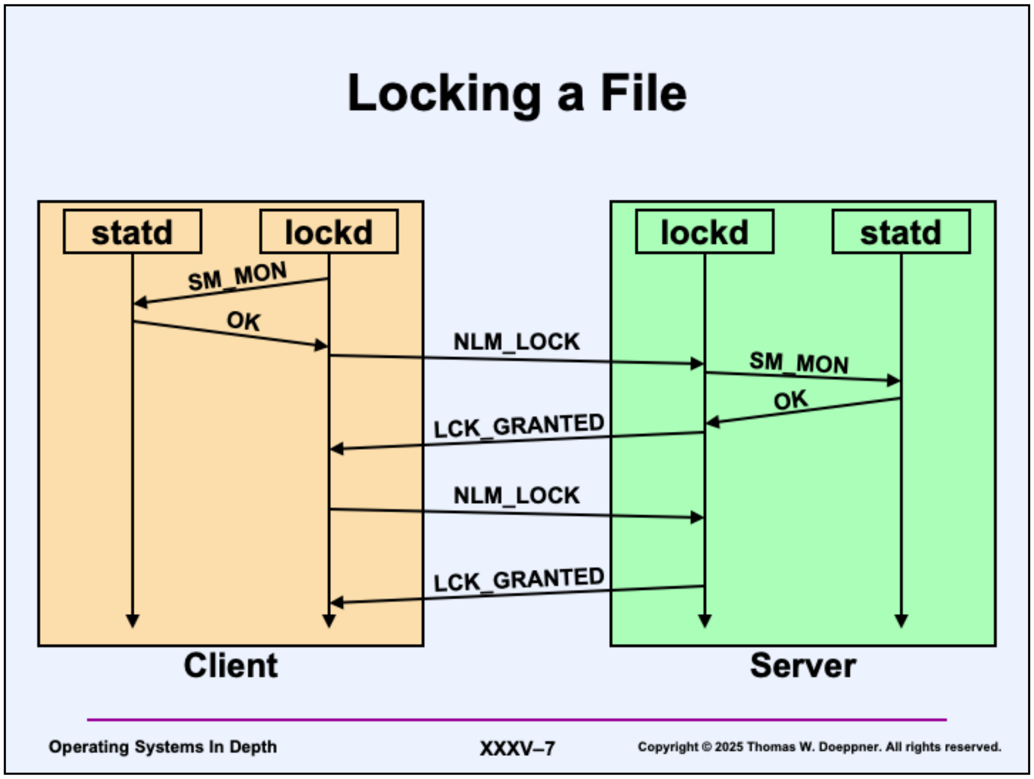
If a server crashes, it loses all knowledge of which files are locked and by whom. So, when it comes back up, it must ask the clients to tell it which files they had locked. Upon receipt of this information, it restores its original state and resumes normal operation.



Status Monitor

- Maintains list of monitored hosts on stable storage
  - clients maintain list of servers on which locks are held
  - servers maintain list of clients who have locks
- On restart
  - reads list of monitored hosts from stable storage and sends each an SM_NOTIFY RPC

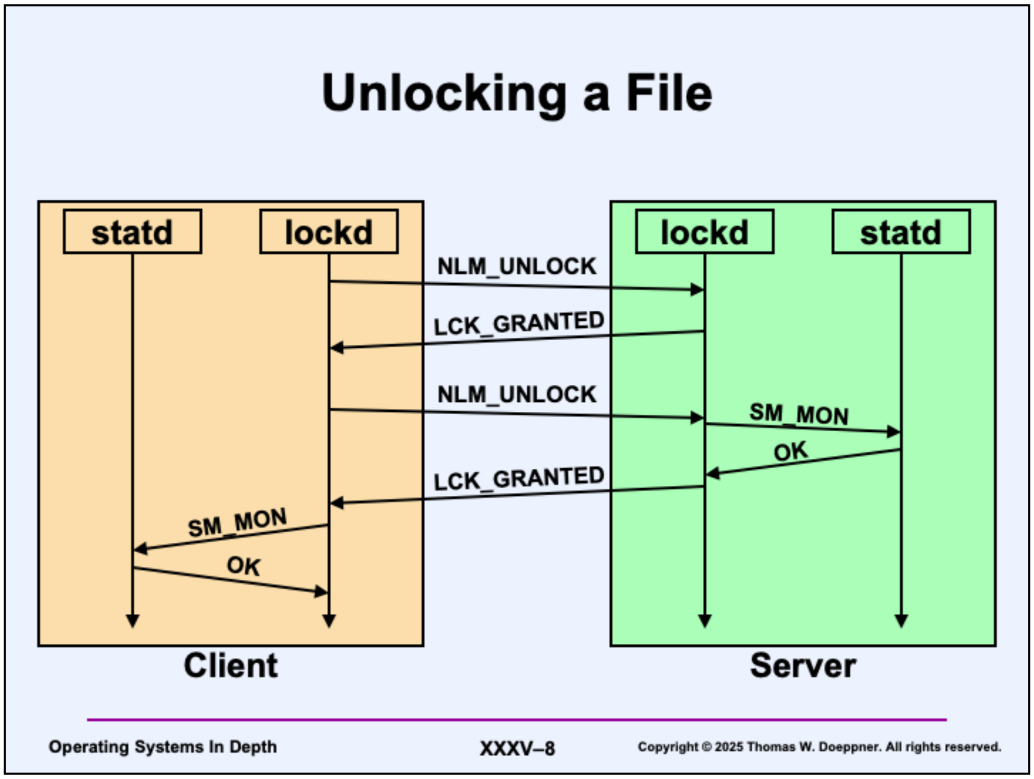
Stable storage is storage that hold data persistently and does so despite crashes and power failures. It’s debatable whether disk storage qualifies, but it’s what is normally used.



Locking a File

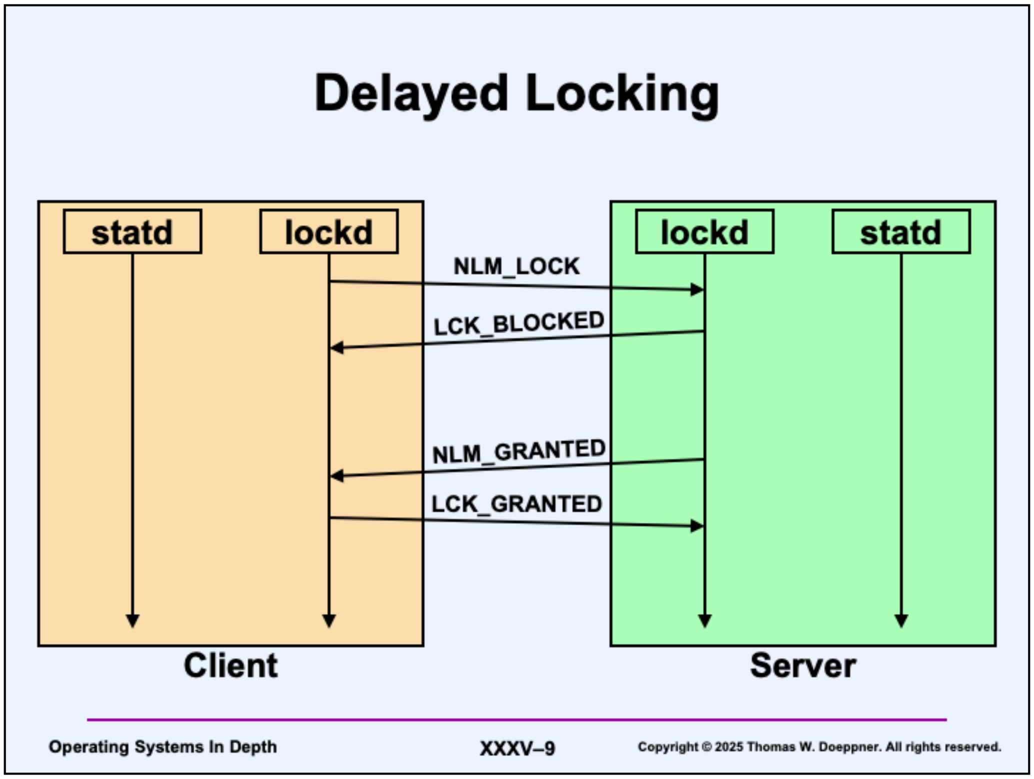
Here a client takes its first lock on the server. The status monitors on both sides are notified. The lock manager provides a callback procedure to the status monitor: the status monitor will call this routine if the lock must be reclaimed due to a server crash or if the lock must be released due to a client crash.

If another lock is obtained on a file on the same server, there’s no need to further inform the status monitors.



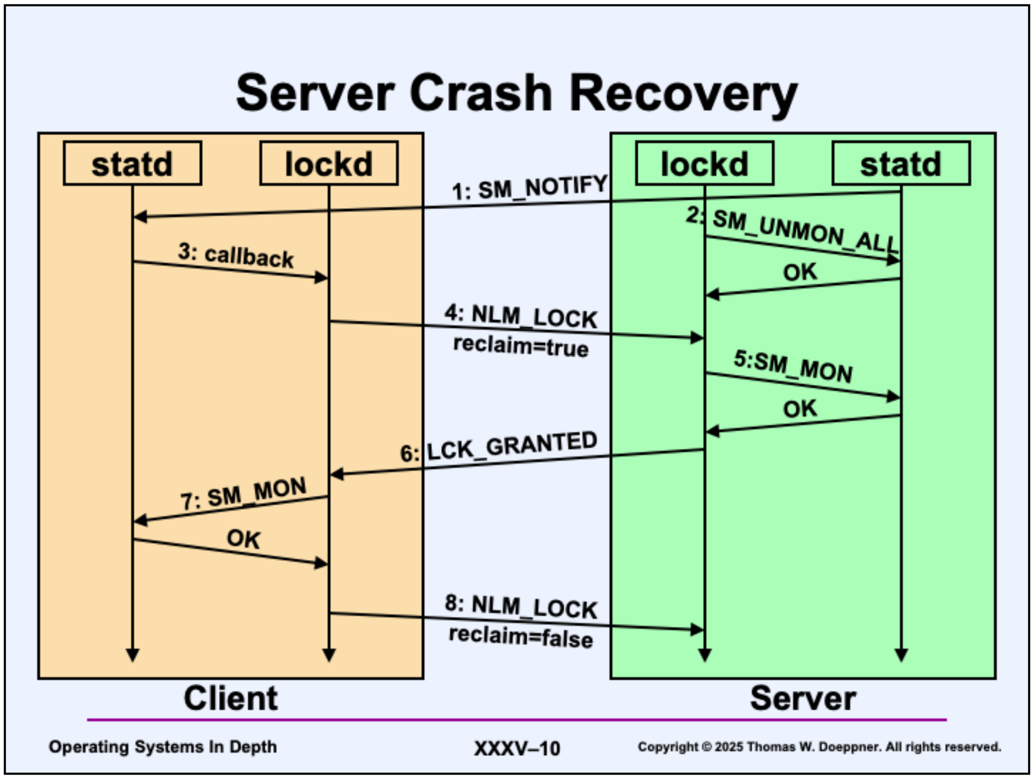
Unlocking a File

Now the client unlocks its lock. When the last lock is unlocked, the status monitor is notified to remove the respective host from the monitored list.



Delayed Locking

Here the client requests the lock, but it’s not immediately granted; the server returns a LCK_BLOCKED response, meaning that the client must wait. When the server can finally grant the lock, the server places an RPC call (a **callback**) to the client, as shown in the slide.



Server Crash Recovery

Operating Systems In Depth XXXV–10

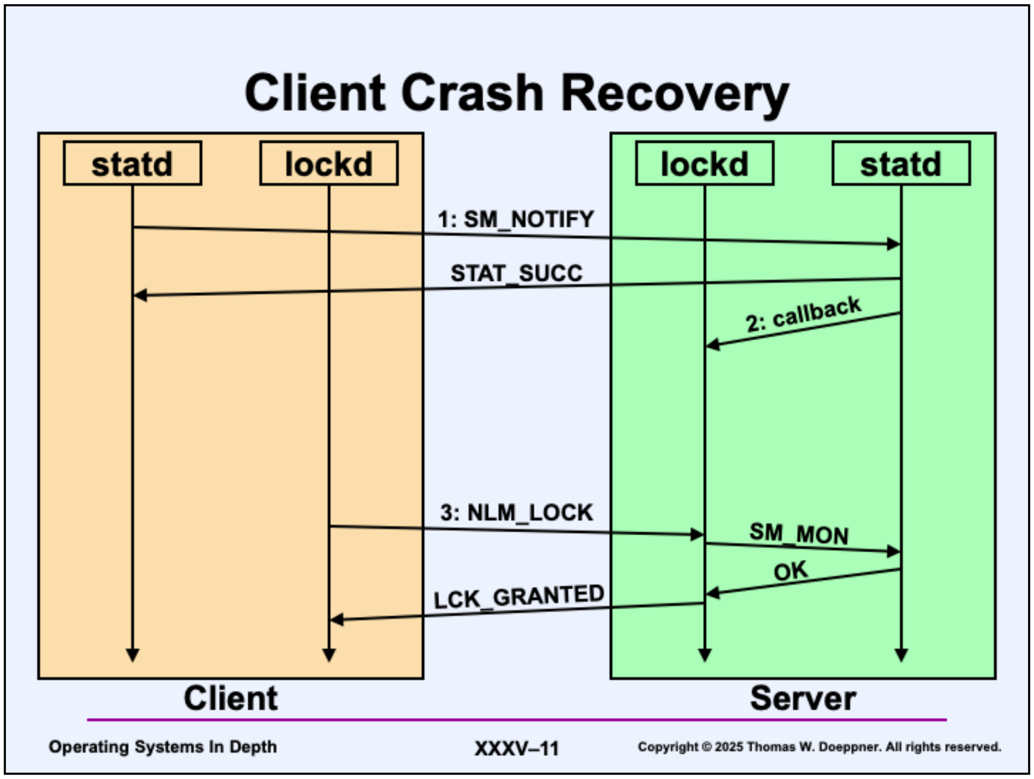
When the server comes up after a crash, its status monitor reads its list of monitored hosts from stable storage and sends each one of them an SM_NOTIFY RPC. The server's lock manager then sends the local status monitor an SM_UNMON_ALL RPC to get it to empty out the list of monitored hosts. The lock manager enters a "grace period" in which it accepts lock requests from only those clients who set the reclaim flags in their requests.

On the client side, once the status monitor receives the SM_NOTIFY, it sends places a call to the callback procedure provided to it by its local lock manager, which tells the lock manager to reclaim its locks on the server. The lock manager does this by sending an NLM_LOCK requests with the *reclaim* flag to true, for each lock it had on the server.

On receipt of the lock request, the server lock manager retakes the lock and places a call to the server's status monitor's SM_MON routine to re-add the client to the list of monitored hosts. Once this call returns, the server responds to the lock request.

When the client receives the successful response, it re-notifies its status monitor of the need to monitor the server.

Once the grace period is over (45 seconds is typical), the server's lock daemon goes back to normal process of lock requests.

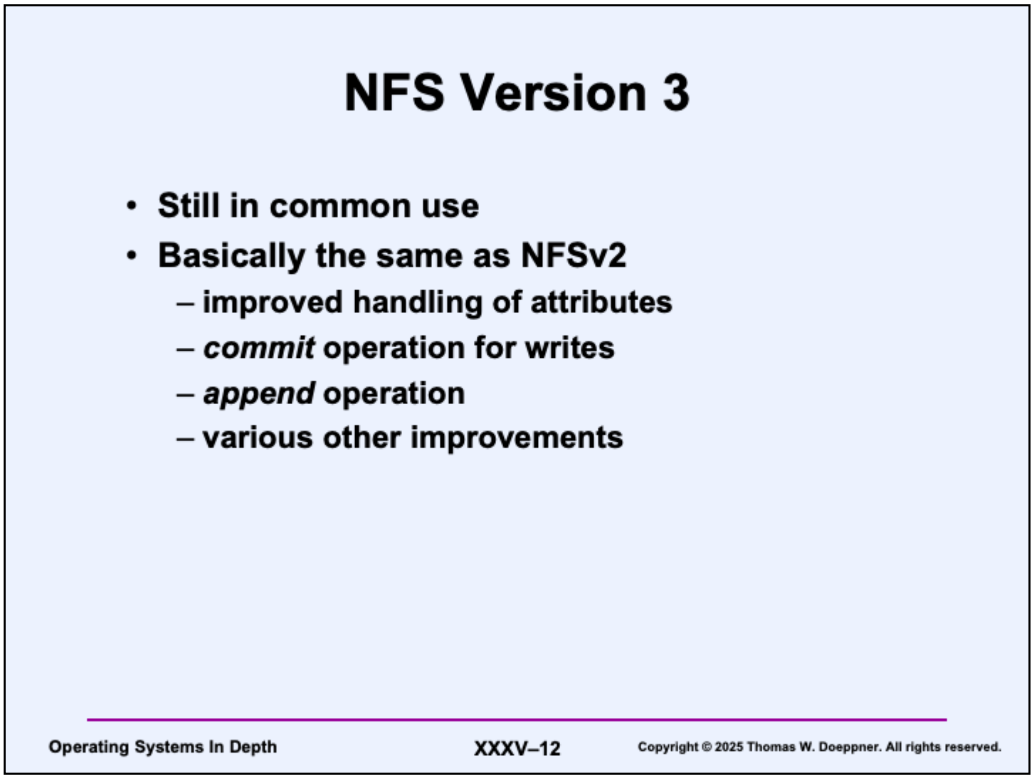


Client Crash Recovery

When the client comes up after a crash, its status monitor reads the list of monitored hosts from stable storage and sends each an SM_NOTIFY RPC. If the server does not respond, then if the server is down, there's no need to contact it further, the lock no longer exists. But if the server is merely unreachable at the moment (perhaps due to a network partition), then the status monitor should continue to try to contact it.

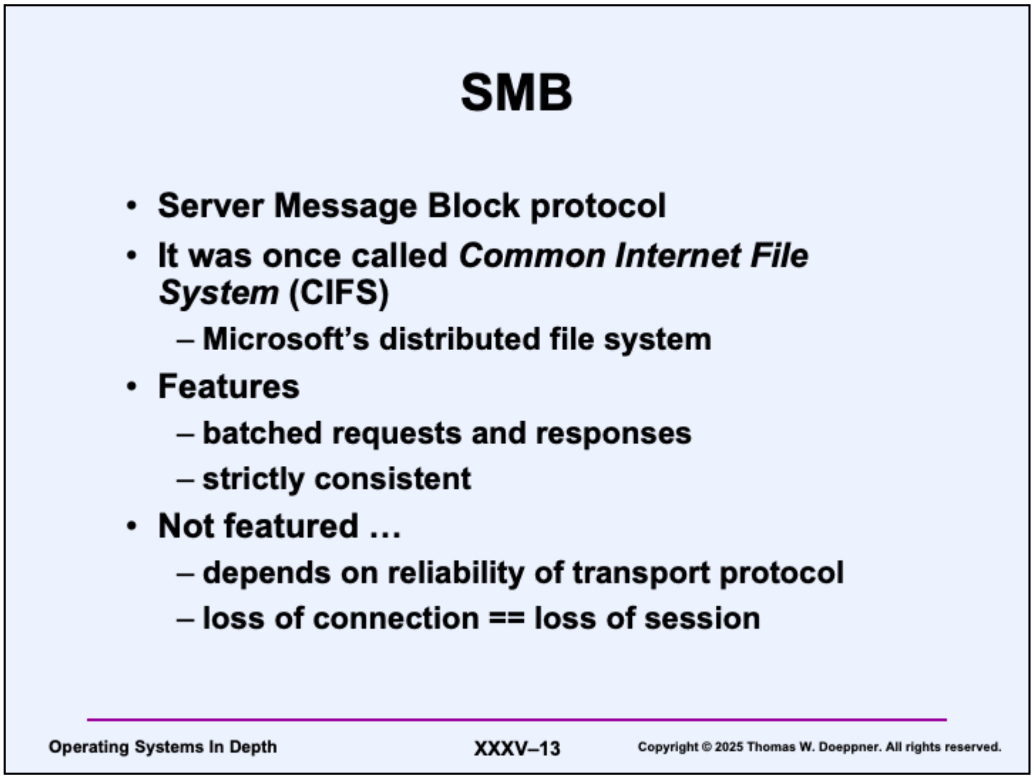
On receipt of the SM_NOTIFY, the server's status monitor calls the callback procedure provided by the lock manager to notify it to release all locks held by the client.

The client may now resume normal operation.



NFS Version 3

- Still in common use
- Basically the same as NFSv2
  - improved handling of attributes
  - *commit* operation for writes
  - *append* operation
  - various other improvements



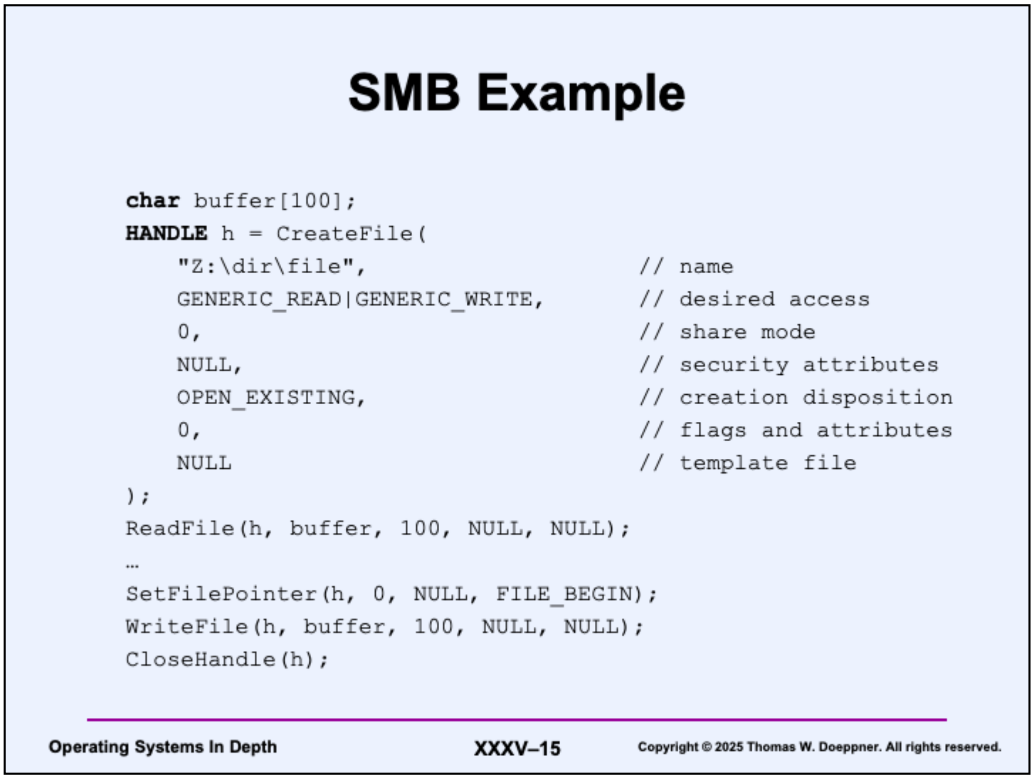
SMB

- Server Message Block protocol
- It was once called *Common Internet File System* (CIFS)
  - 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

SMB is also used by Apple on OS X.

History

- Originally a simple means for sharing files
  - – developed by IBM
  - – ran on top of NetBIOS
- Microsoft took over
  - – renamed CIFS in late 1990s
    - later rerenamed SMB
  - – uses SMB as RPC-like communication protocol
    - runs on NetBIOS
    - usually layered on TCP
    - often no NetBIOS, just TCP

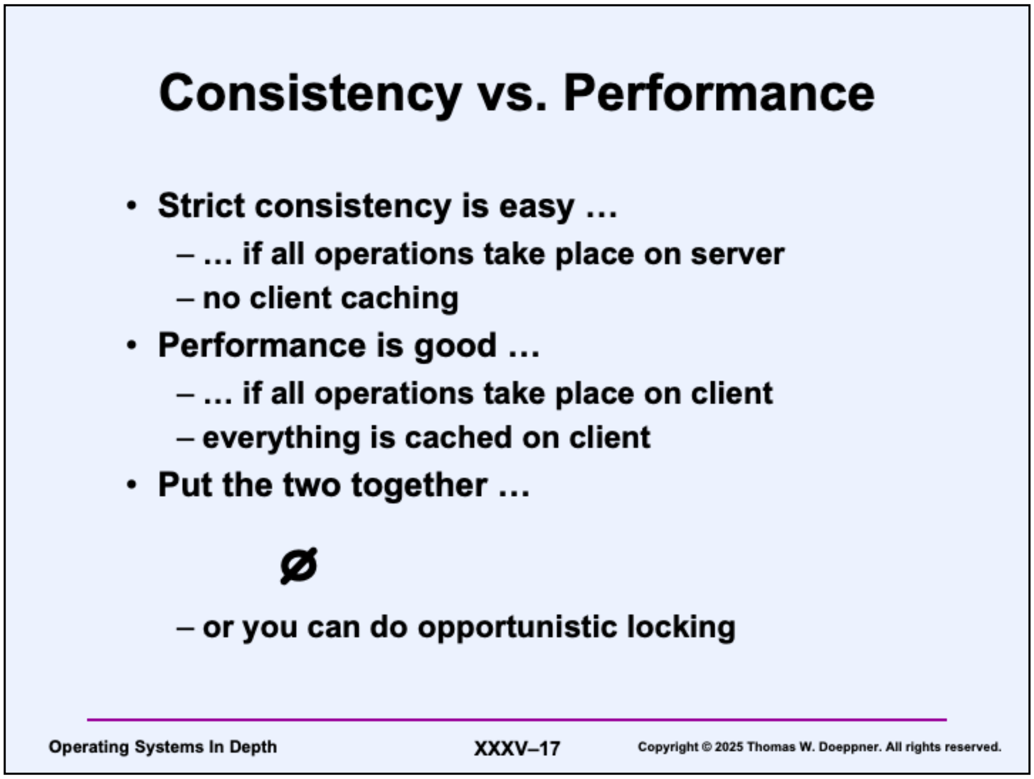


SMB Example

```
char buffer[100];
HANDLE h = CreateFile(
    "Z:\dir\file",                    // name
    GENERIC_READ|GENERIC_WRITE,       // desired access
    0,                                // share mode
    NULL,                             // security attributes
    OPEN_EXISTING,                    // creation disposition
    0,                                // flags and attributes
    NULL                              // template file
);
ReadFile(h, buffer, 100, NULL, NULL);
…
SetFilePointer(h, 0, NULL, FILE_BEGIN);
WriteFile(h, buffer, 100, NULL, NULL);
CloseHandle(h);
```

Share Mode

- When opening a file
  - specify intended use of file (desired access)
    - read, write, or both
  - specify restrictions on how others may use the file (controlled sharing)
    - read, write, both, or none
    - known as *share reservations*

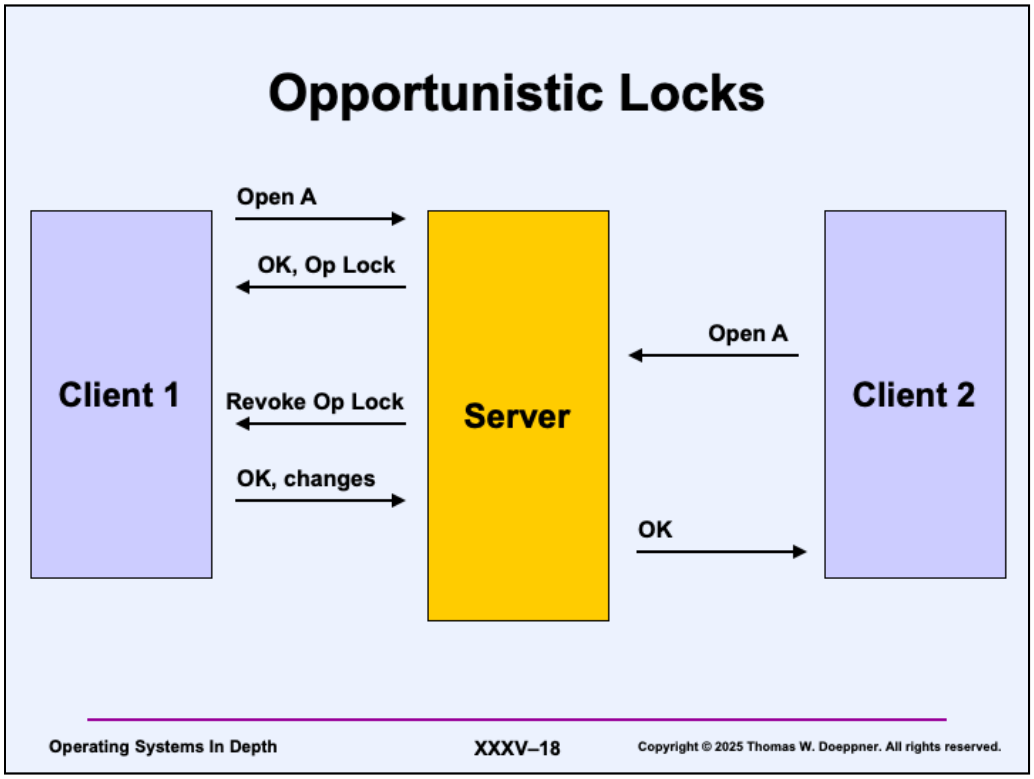


Consistency vs. Performance

- 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

Client 1

Open A

OK, Op Lock

Revoke Op Lock

OK, changes

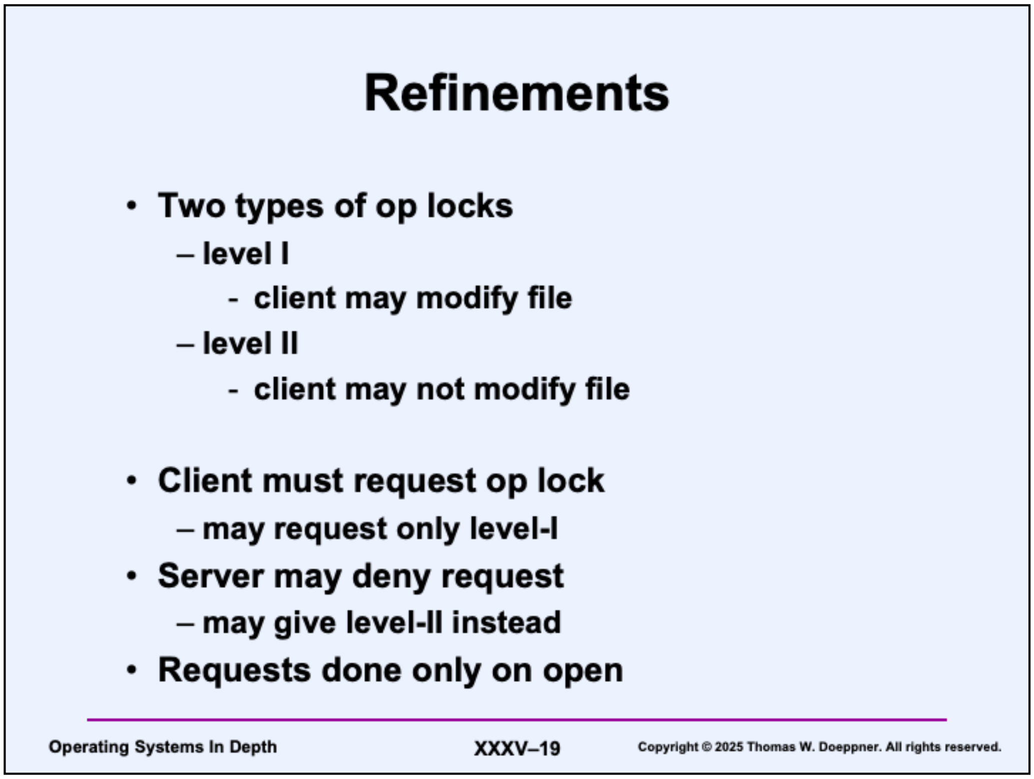
Server

Open A

OK

Client 2

Opportunistic locks are used by SMB.



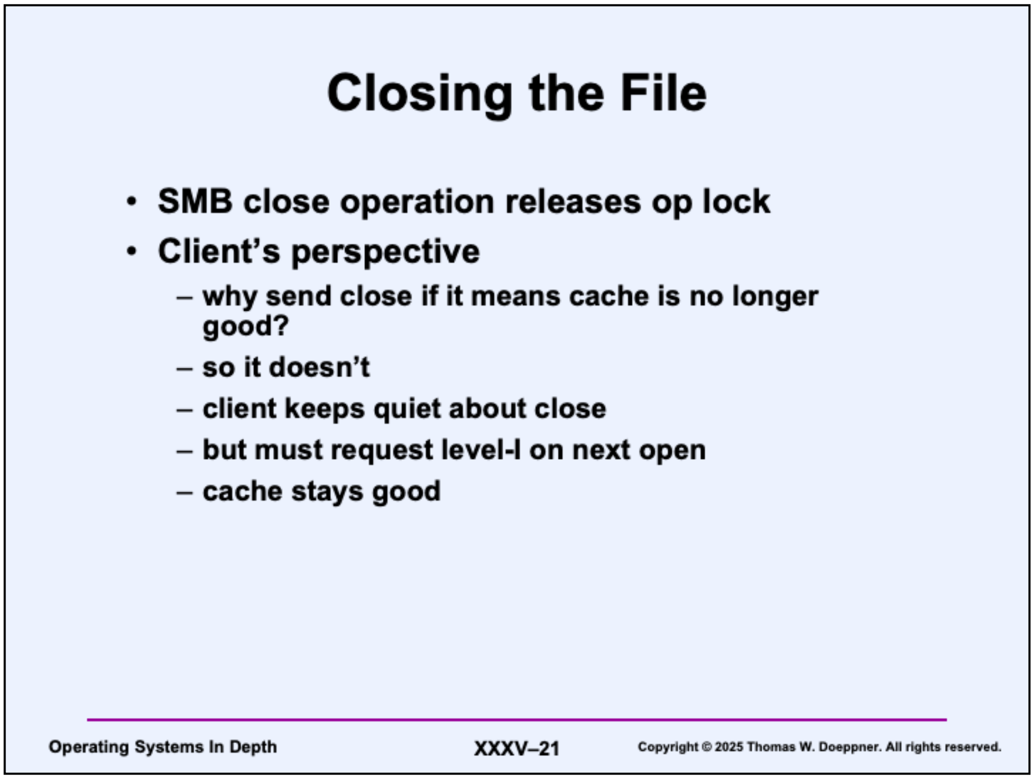
Refinements

- Two types of op locks
  - level I
    - client may modify file
  - level II
    - client may not modify file

- Client must request op lock
  - may request only level-I
- Server may deny request
  - may give level-II instead
- Requests done only on open

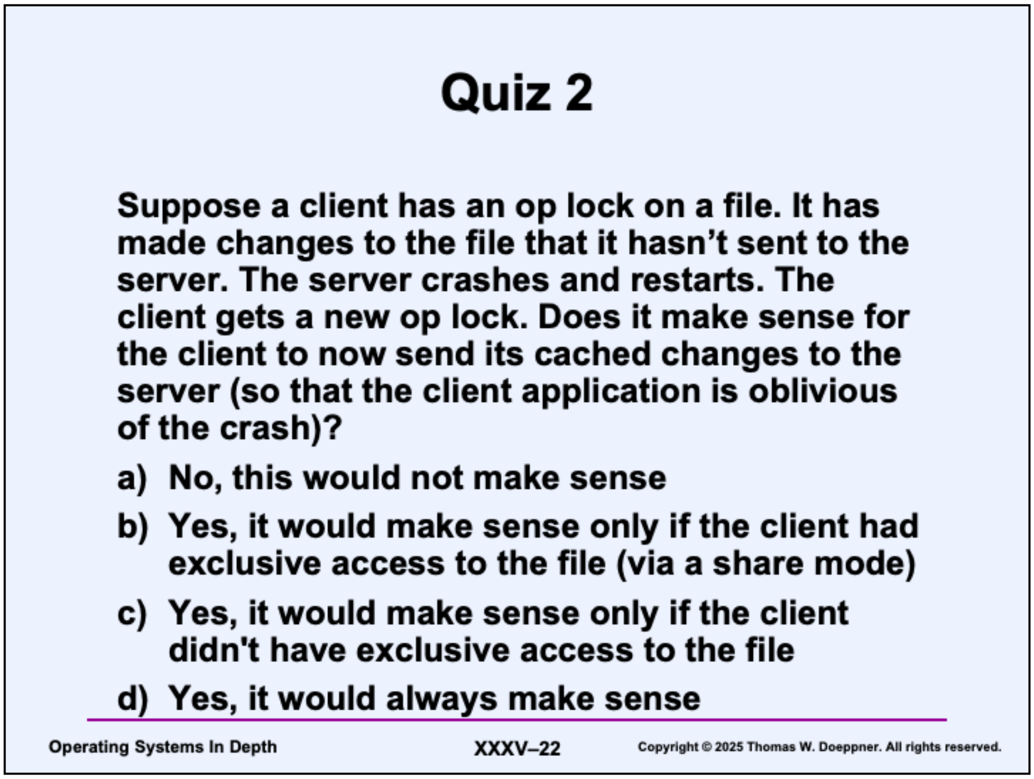
Read-Only Access

- Why request a level-I op lock if file is to be open read-only?
  - – typical Windows application always opens read-write
  - – may never get around to writing …



Closing the File

- SMB close operation releases op lock
- Client's perspective
  - why send close if it means cache is no longer good?
  - so it doesn't
  - client keeps quiet about close
  - but must request level-I on next open
  - cache stays good



Quiz 2

Suppose a client has an op lock on a file. It has made changes to the file that it hasn’t sent to the server. The server crashes and restarts. The client gets a new op lock. Does it make sense for the client to now send its cached changes to the server (so that the client application is oblivious of the crash)?

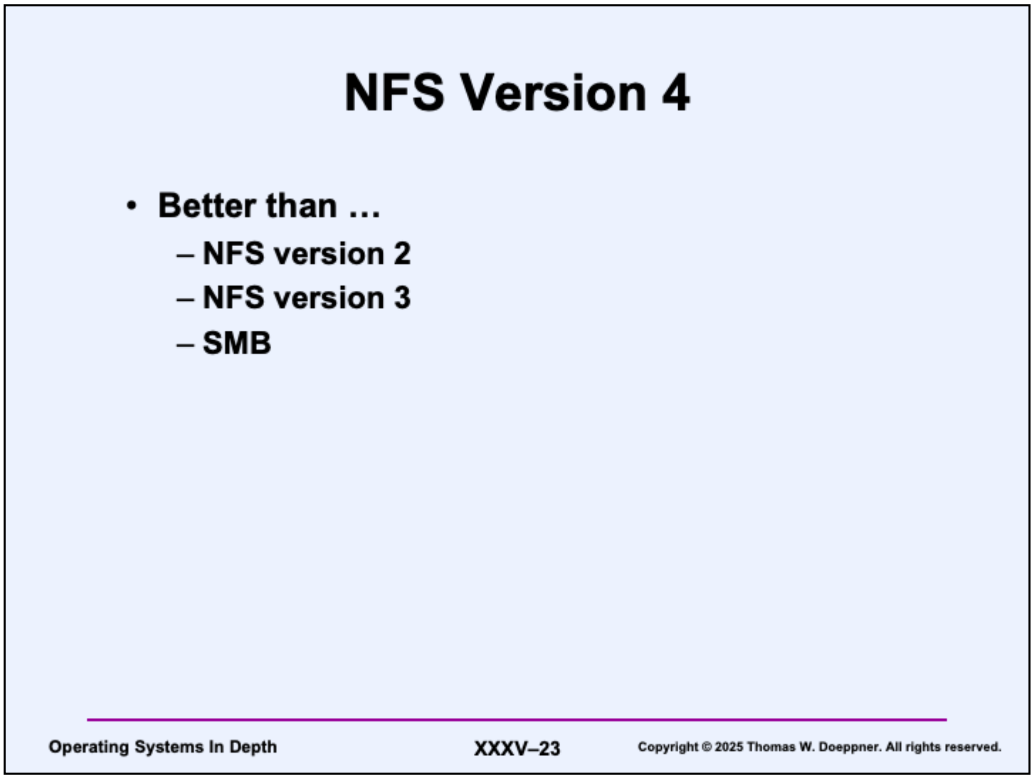
a) No, this would not make sense

b) Yes, it would make sense only if the client had exclusive access to the file (via a share mode)

c) Yes, it would make sense only if the client didn't have exclusive access to the file

d) Yes, it would always make sense

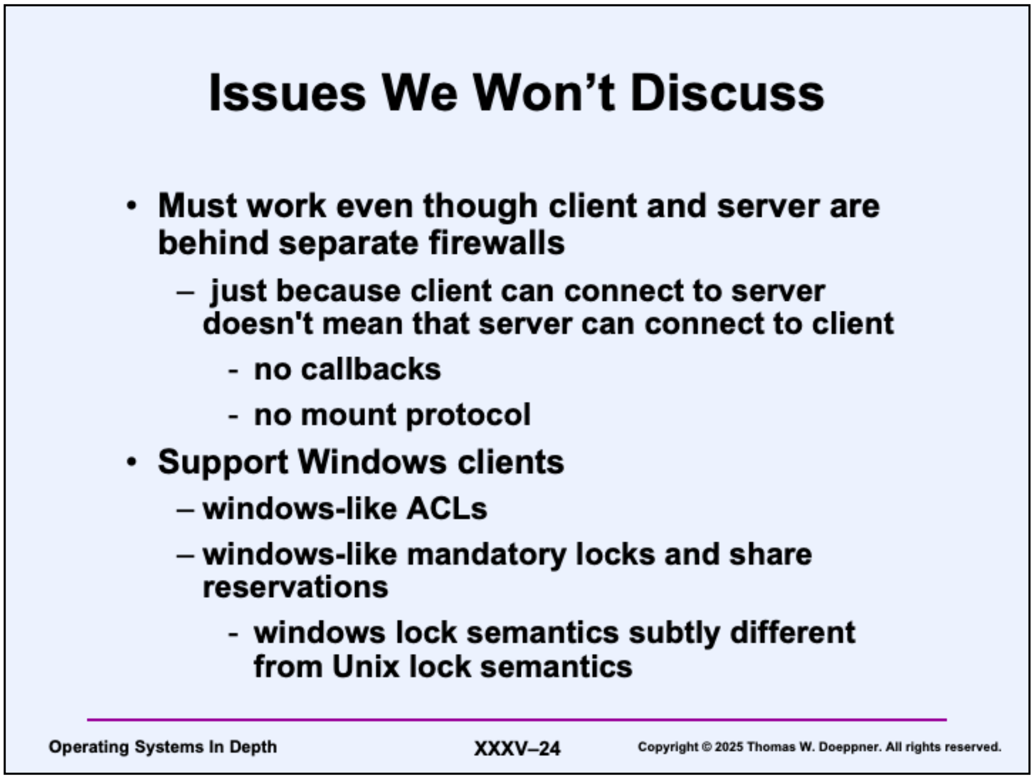
By "make sense", we mean that it would make sense to modify the protocol so this is done automatically.



NFS Version 4

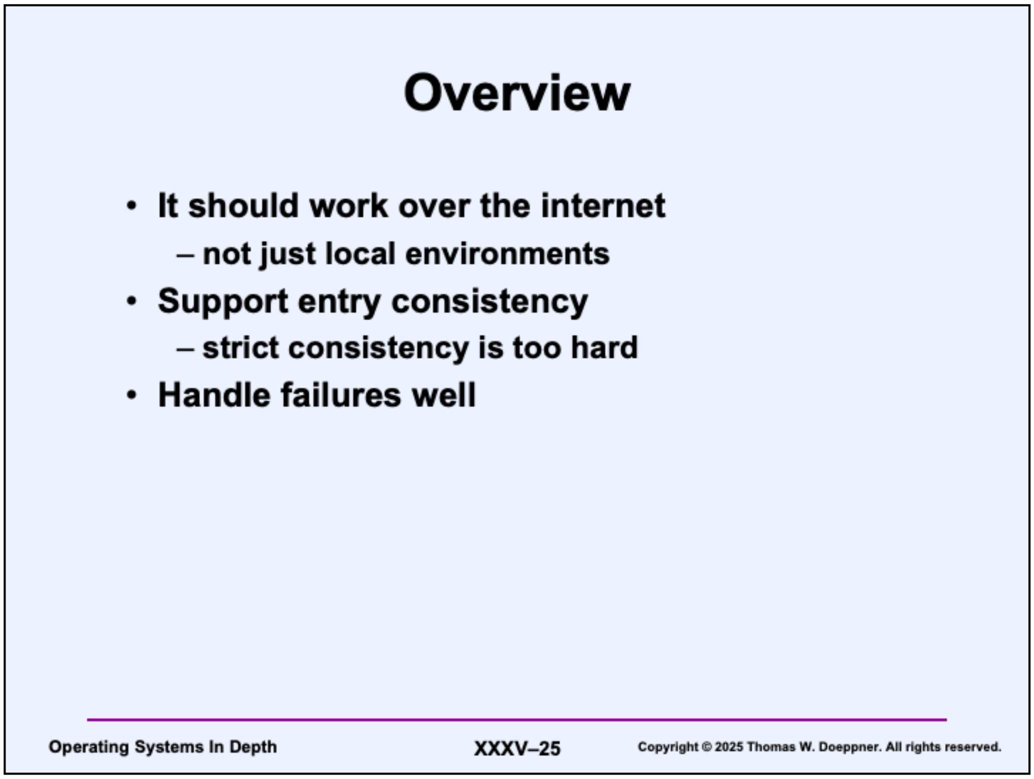
- Better than …
  - NFS version 2
  - NFS version 3
  - SMB

NFSv4 is used at Brown, along with NFSv3 and SMB.



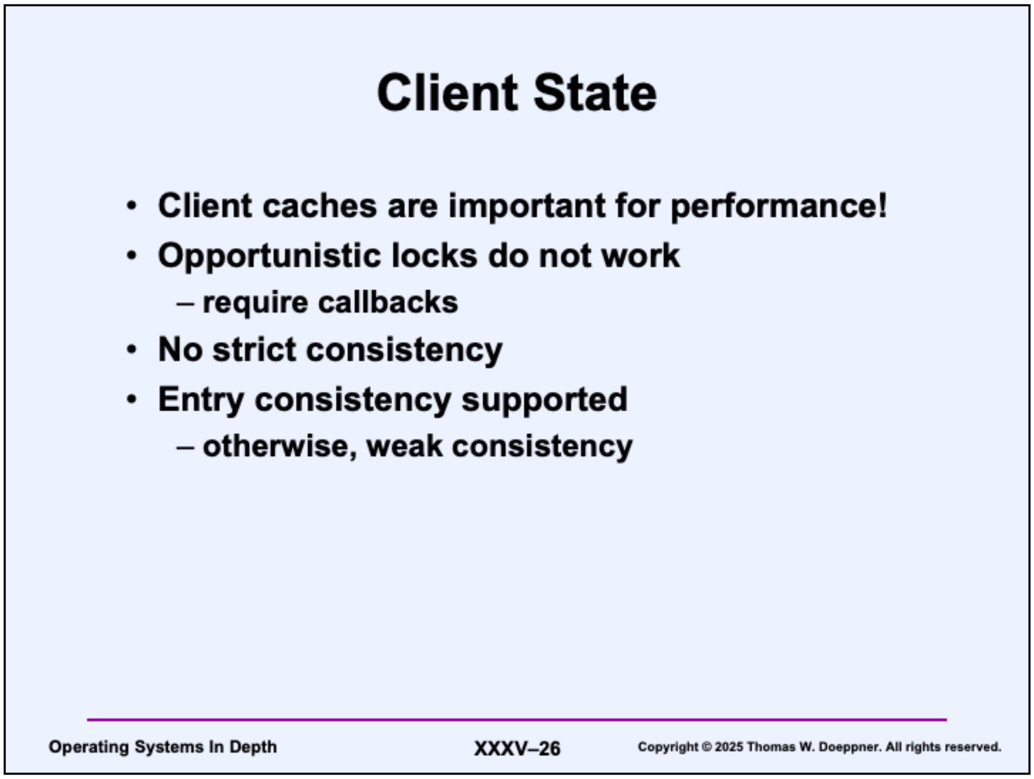
Issues We Won’t Discuss

- Must work even though client and server are behind separate firewalls
  - just because client can connect to server doesn't mean that server can connect to client
    - no callbacks
    - no mount protocol
- Support Windows clients
  - windows-like ACLs
  - windows-like mandatory locks and share reservations
    - windows lock semantics subtly different from Unix lock semantics



Overview

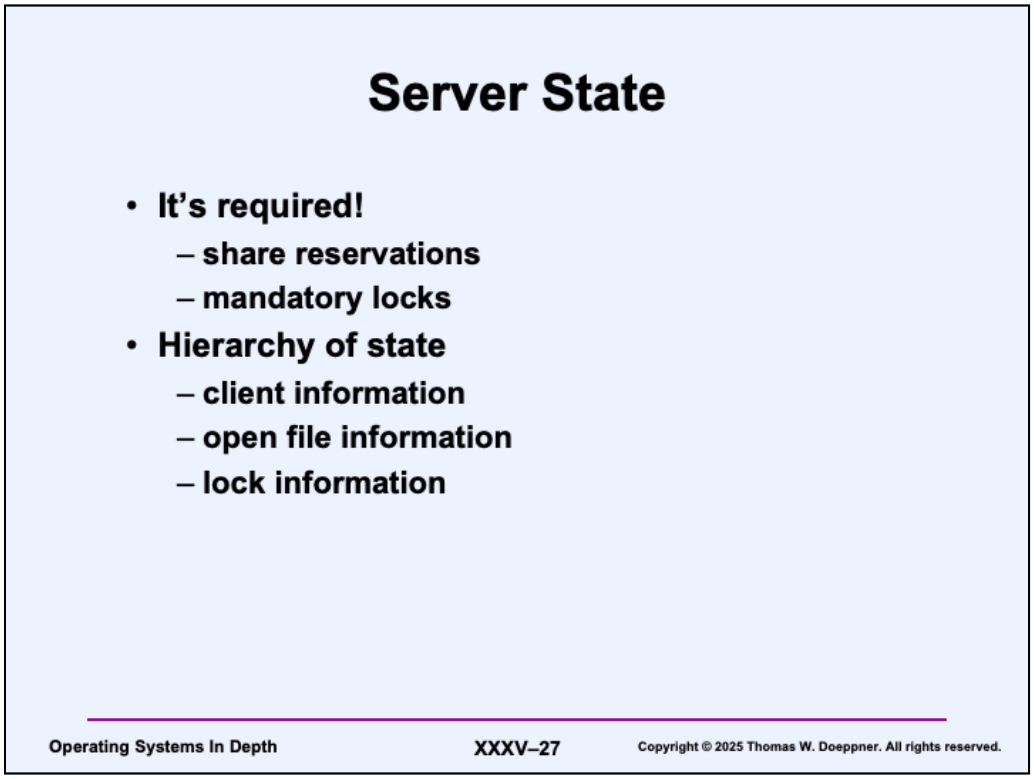
- It should work over the internet
  - not just local environments
- Support entry consistency
  - strict consistency is too hard
- Handle failures well



Client State

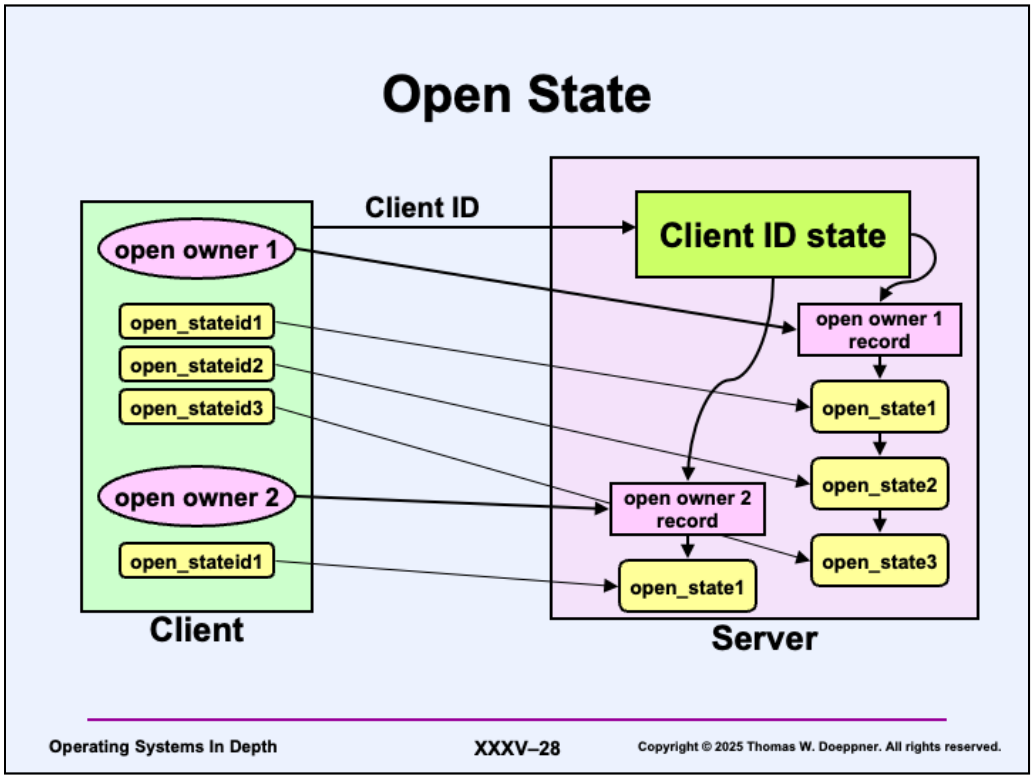
- Client caches are important for performance!
- Opportunistic locks do not work
  – require callbacks
- No strict consistency
- Entry consistency supported
  – otherwise, weak consistency

Note that entry consistency wasn't available in NFSv3 because mandatory locking was not supported.



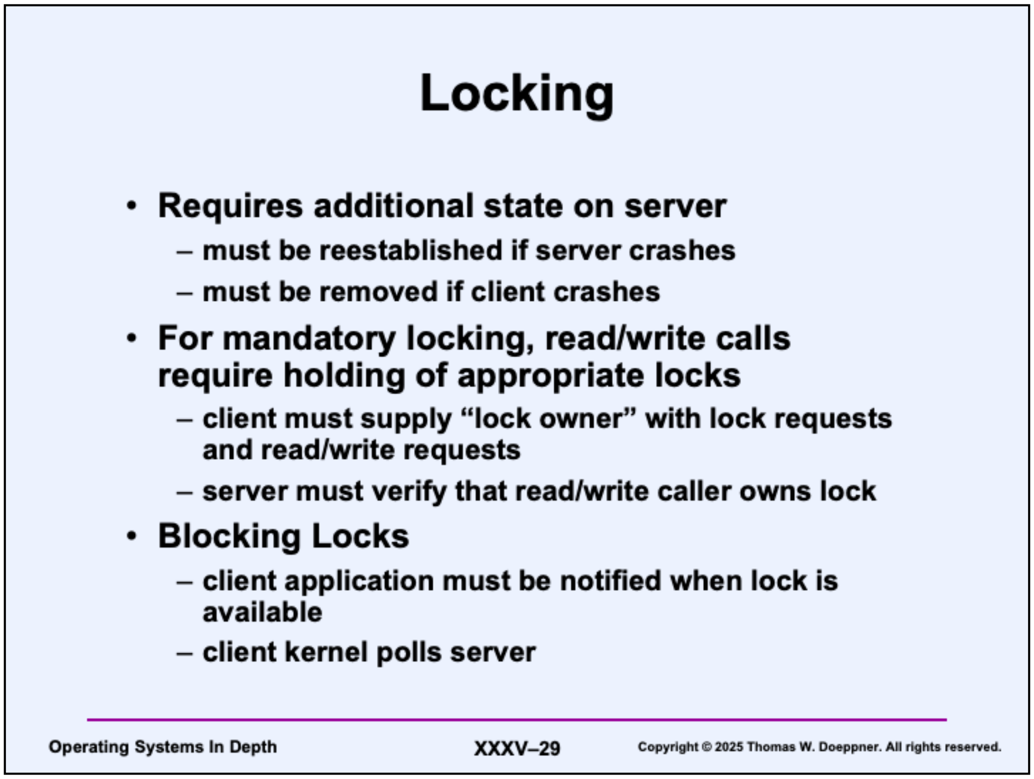
Server State

- It’s required!
  - share reservations
  - mandatory locks
- Hierarchy of state
  - client information
  - open file information
  - lock information



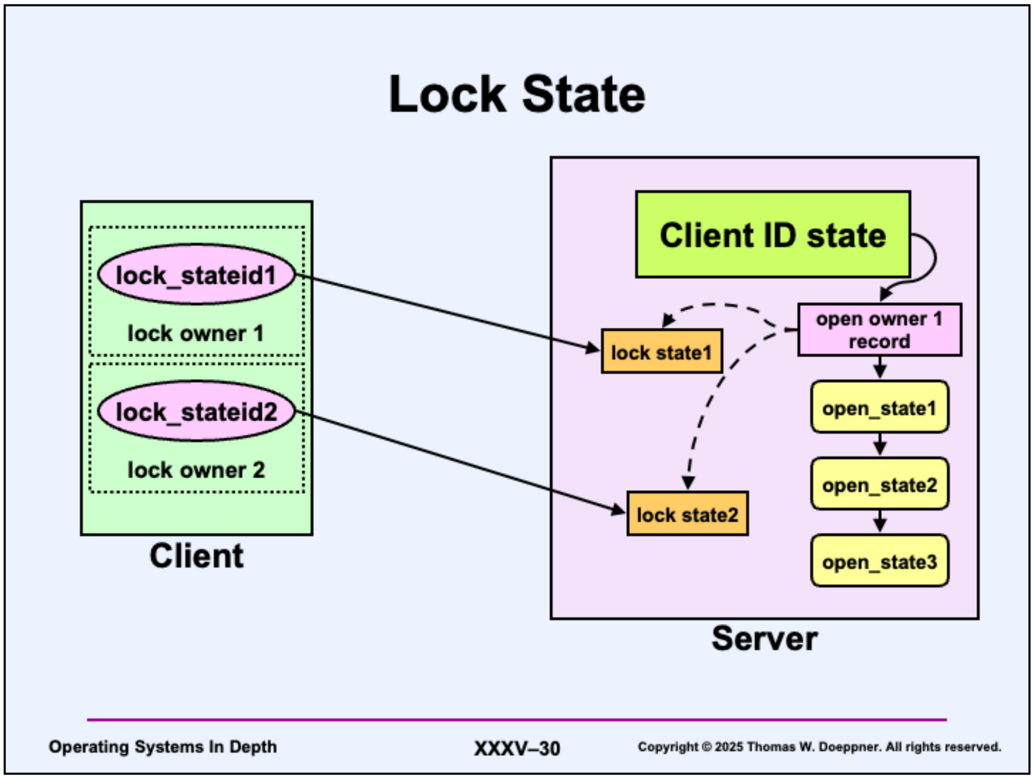
Open State

Note that there's a hierarchy of state information. Thus if the client crashes, everything related to that client can be nullified. If an open owner terminates, then just what's related is nullified. Note that an "open owner" represents all processes sharing the opening of a file.



Locking

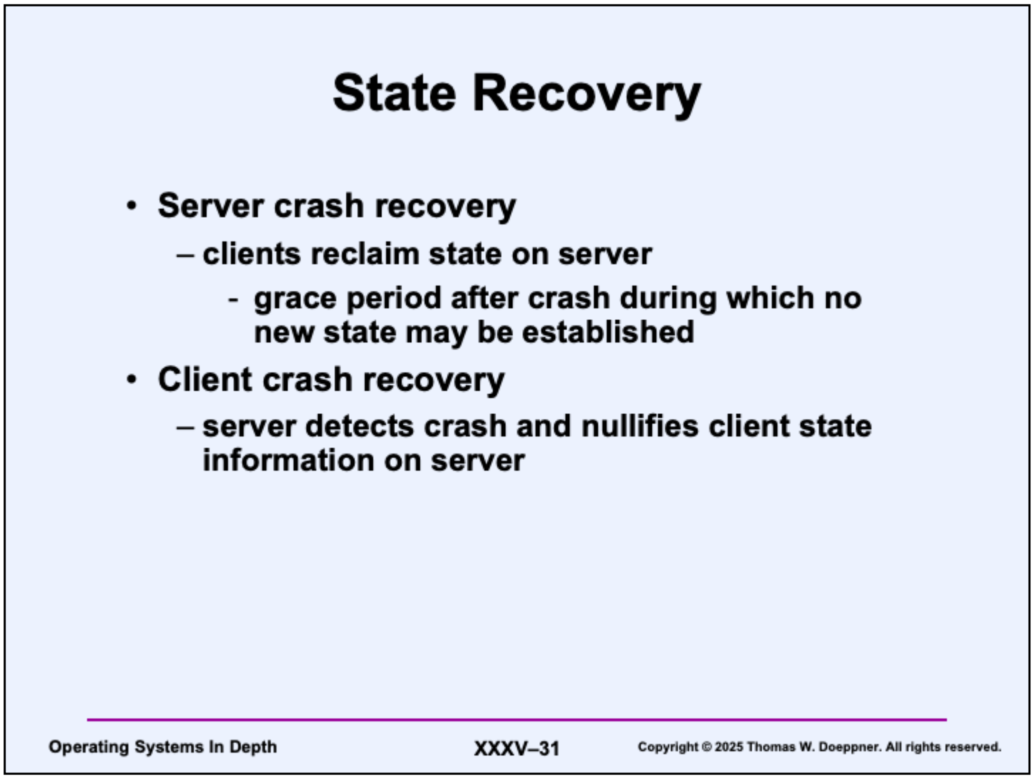
- Requires additional state on server
  - must be reestablished if server crashes
  - must be removed if client crashes
- For mandatory locking, read/write calls require holding of appropriate locks
  - client must supply “lock owner” with lock requests and read/write requests
  - server must verify that read/write caller owns lock
- Blocking Locks
  - client application must be notified when lock is available
  - client kernel polls server



Lock State

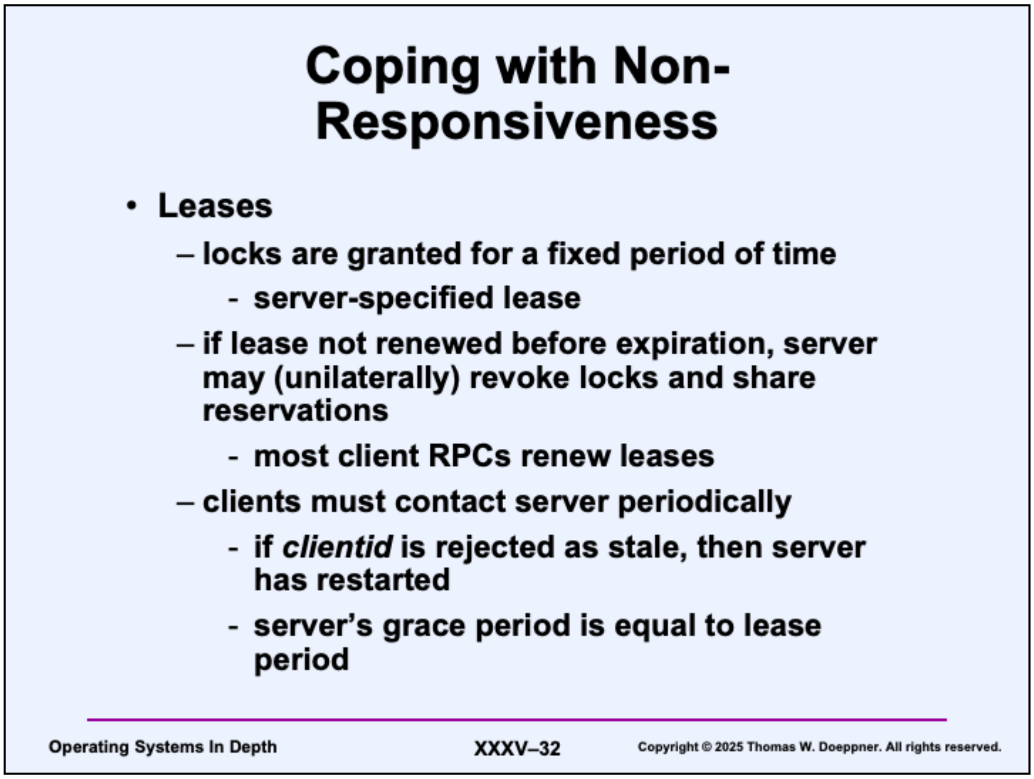
The client determines who the lock owner is, based on Unix or Windows semantics (or perhaps a different OS's).

Unix semantics are that the lock is associated with the process. Thus if the process has multiple file descriptors referring to the same file, if the file is locked via one file descriptor, all file descriptors have the lock. But, if the file is unlocked via one file descriptor, it's unlocked for them all. (Some consider this to be a design bug.) Windows lock semantics is that the lock is strictly associated with the file handle (its term for file descriptor).



State Recovery

- Server crash recovery
  - clients reclaim state on server
    - grace period after crash during which no new state may be established
- Client crash recovery
  - server detects crash and nullifies client state information on server

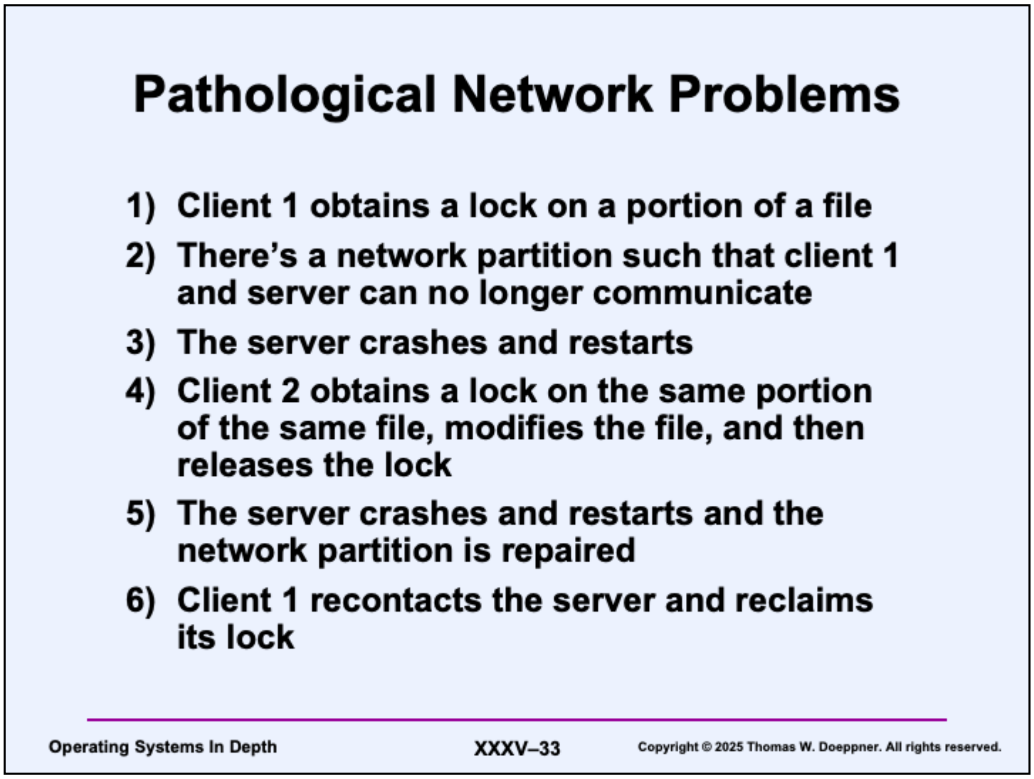


Coping with Non-Responsiveness

- **Leases**
  - – locks are granted for a fixed period of time
    - server-specified lease
  - – if lease not renewed before expiration, server may (unilaterally) revoke locks and share reservations
    - most client RPCs renew leases
  - – clients must contact server periodically
    - if *clientid* is rejected as stale, then server has restarted
    - server’s grace period is equal to lease period

Leases are used to cope with lock-holding clients that either crash and take a long time to recover or suffer network prolonged network outages (or both).

Note that, for Unix clients, it’s rare to have lock state. Thus for most applications, NFSv4 behaves like NFSv3.



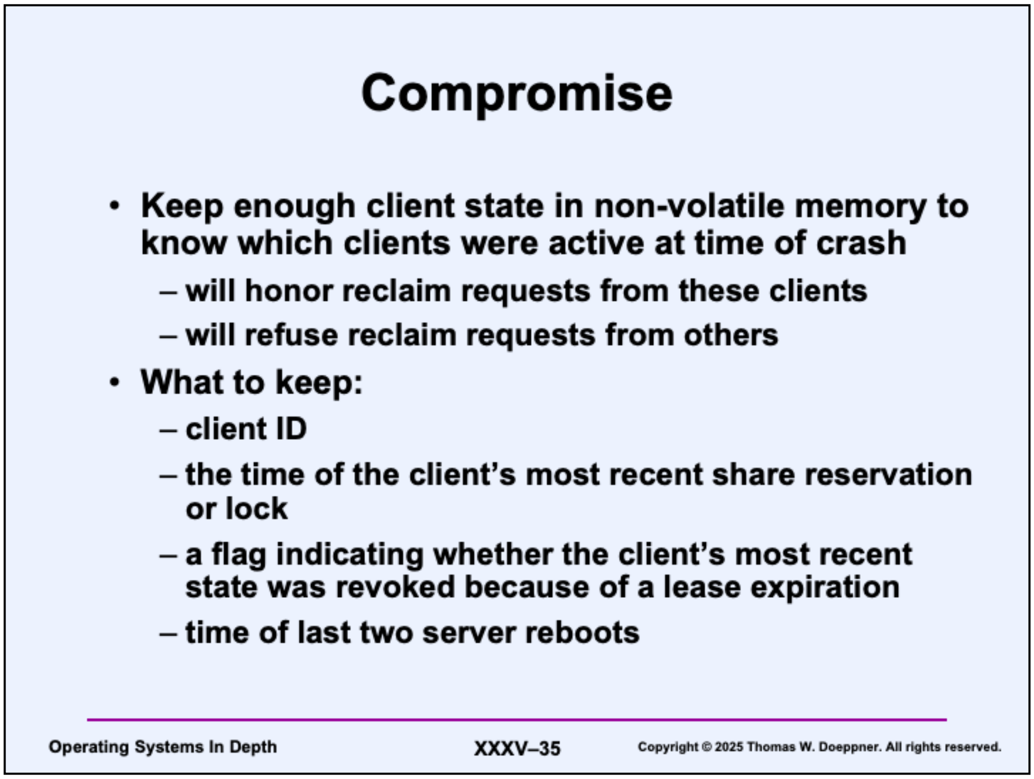
Pathological Network Problems

1) Client 1 obtains a lock on a portion of a file
2) There's a network partition such that client 1 and server can no longer communicate
3) The server crashes and restarts
4) Client 2 obtains a lock on the same portion of the same file, modifies the file, and then releases the lock
5) The server crashes and restarts and the network partition is repaired
6) Client 1 recontacts the server and reclaims its lock

As far as client 1 is concerned, the server crashed at step 2 (since the client couldn't contact it) and didn't restart until step 5. If the server, at step 6, has no information about its lock state prior to the crash, it cannot recognize that client 1 should not be allowed to reclaim its lock in step 6.

Coping …

- Possibilities
  1) server keeps all client state in non-volatile storage
  2) server keeps all client state in volatile storage and refuses all reclaim requests (effectively emulating SMB)
  3) something in between …



Compromise

- Keep enough client state in non-volatile memory to know which clients were active at time of crash
  - will honor reclaim requests from these clients
  - will refuse reclaim requests from others
- What to keep:
  - client ID
  - the time of the client's most recent share reservation or lock
  - a flag indicating whether the client's most recent state was revoked because of a lease expiration
  - time of last two server reboots

If the client was active (had lock state on the server) at the time of the crash, then it must have acquired a share reservation or lock before the most recent reboot, but after the prior reboot. However, if this share reservation of lock was revoked before the the most recent server reboot, then it must not have been active at the time of the most recent server crash.

Note that, rather than record the time of each share reservation or lock, it's sufficient to record the first time it occurs after a reboot or lease expiration: we simply want to be able to figure out, in the event it issues a reclaim request, whether it was active at the time of the most recent server crash.

Quiz 3

Our system employs the compromise of the previous slide. Consider the scenario previously discussed:

1) Client 1 obtains a lock on a portion of a file
2) There’s a network partition such that client 1 and server can no longer communicate
3) The server crashes and restarts
4) Client 2 obtains a lock on the same portion of the same file, modifies the file, and then releases the lock
5) The server crashes and restarts and the network partition is repaired
6) Client 1 recontacts the server and reclaims its lock

a) In step 4, client 2 is refused the lock

b) In step 6, client 1 is refused the reclaim request

c) The complete scenario can still occur

Quiz 4

Again, assume our NSFv4 system uses the compromise approach. A client obtains a lock on a file. The server crashes (the client does not). The server comes back up.

a) the client will be able to reclaim its lock, just as it could in NFSv3

b) the client will be able to reclaim its lock only if the server wasn't down too long

c) the client won't be able to reclaim its lock