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

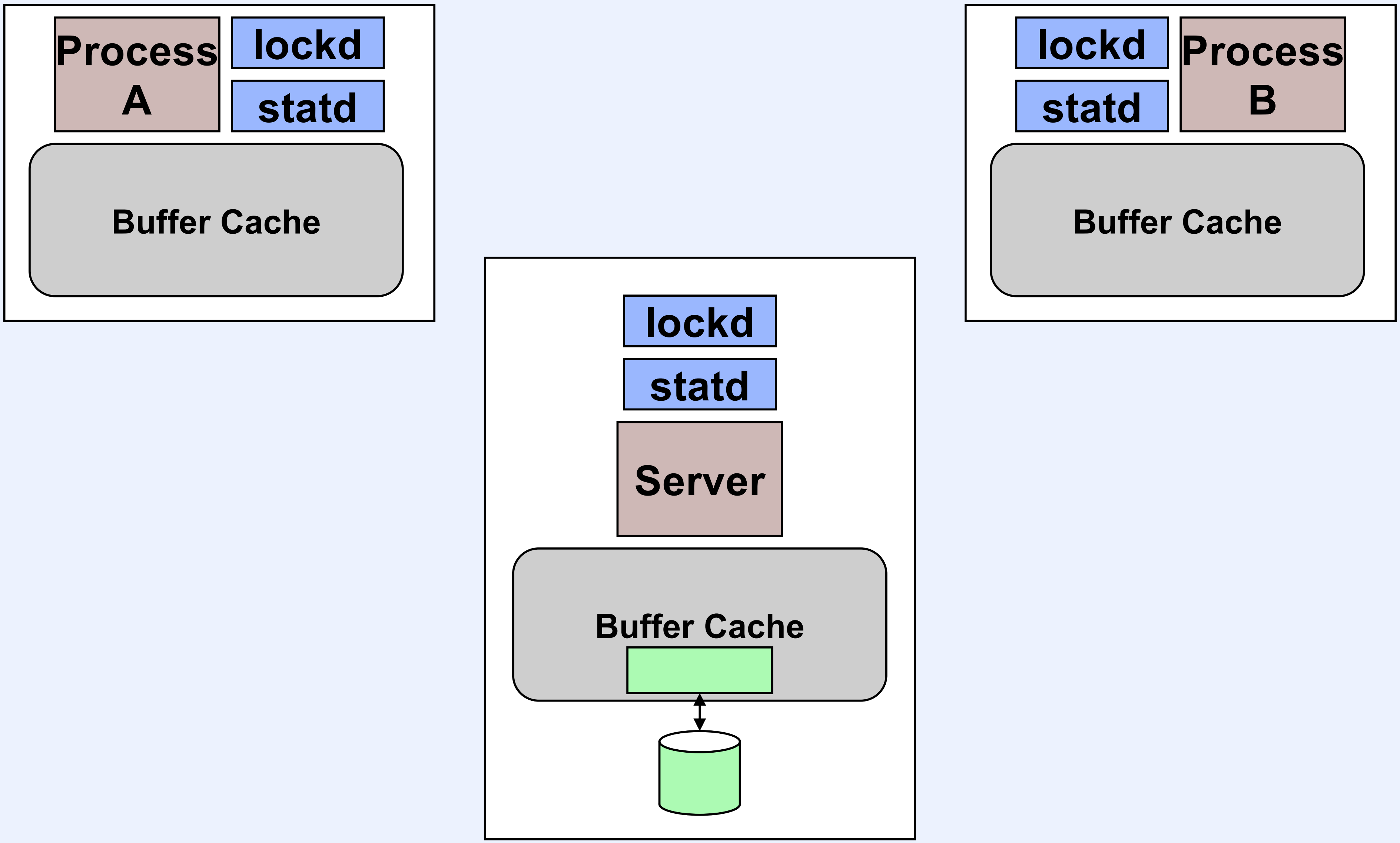
b) yes with high probability

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)

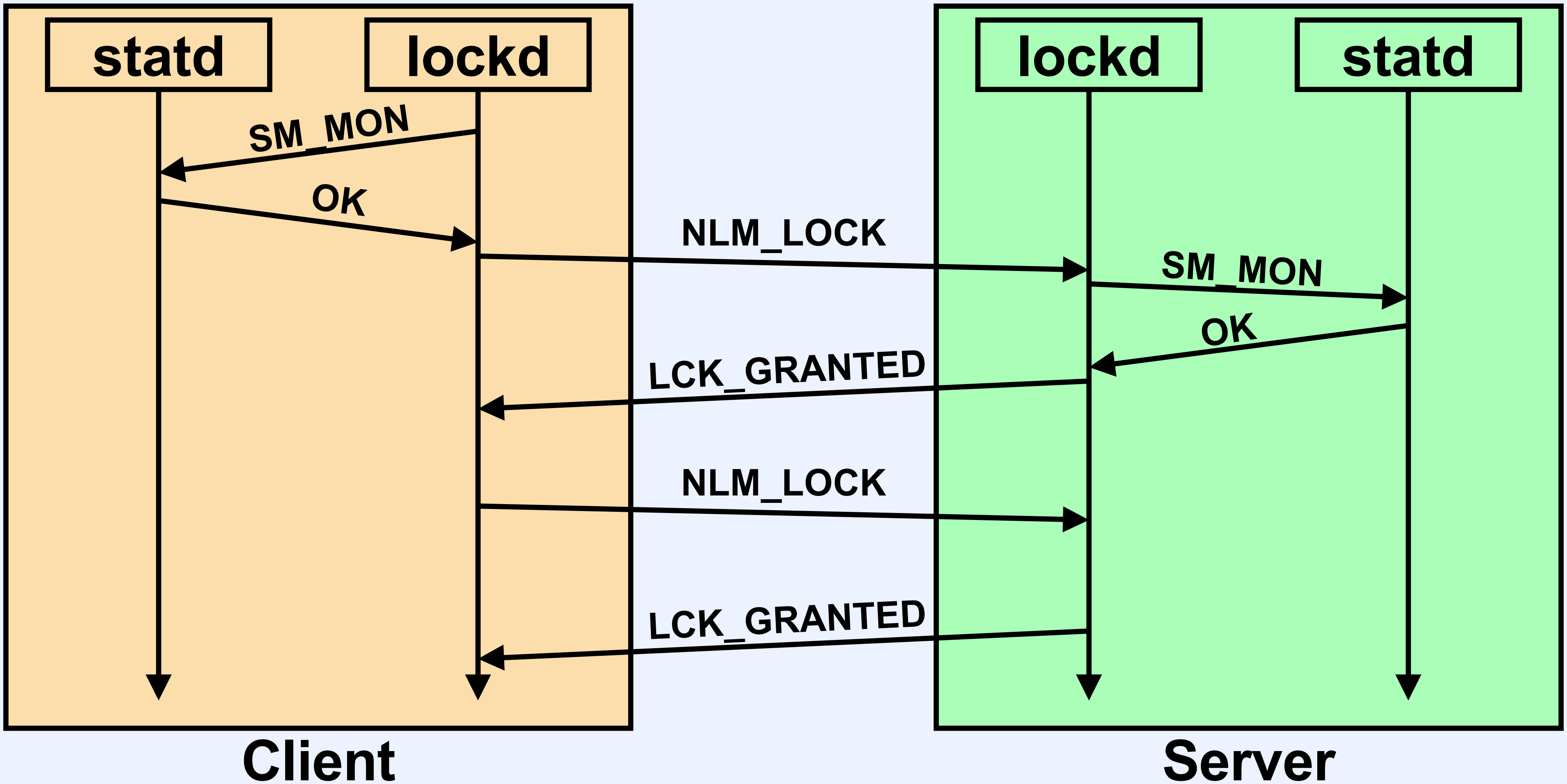
Network Lock Manager Protocol



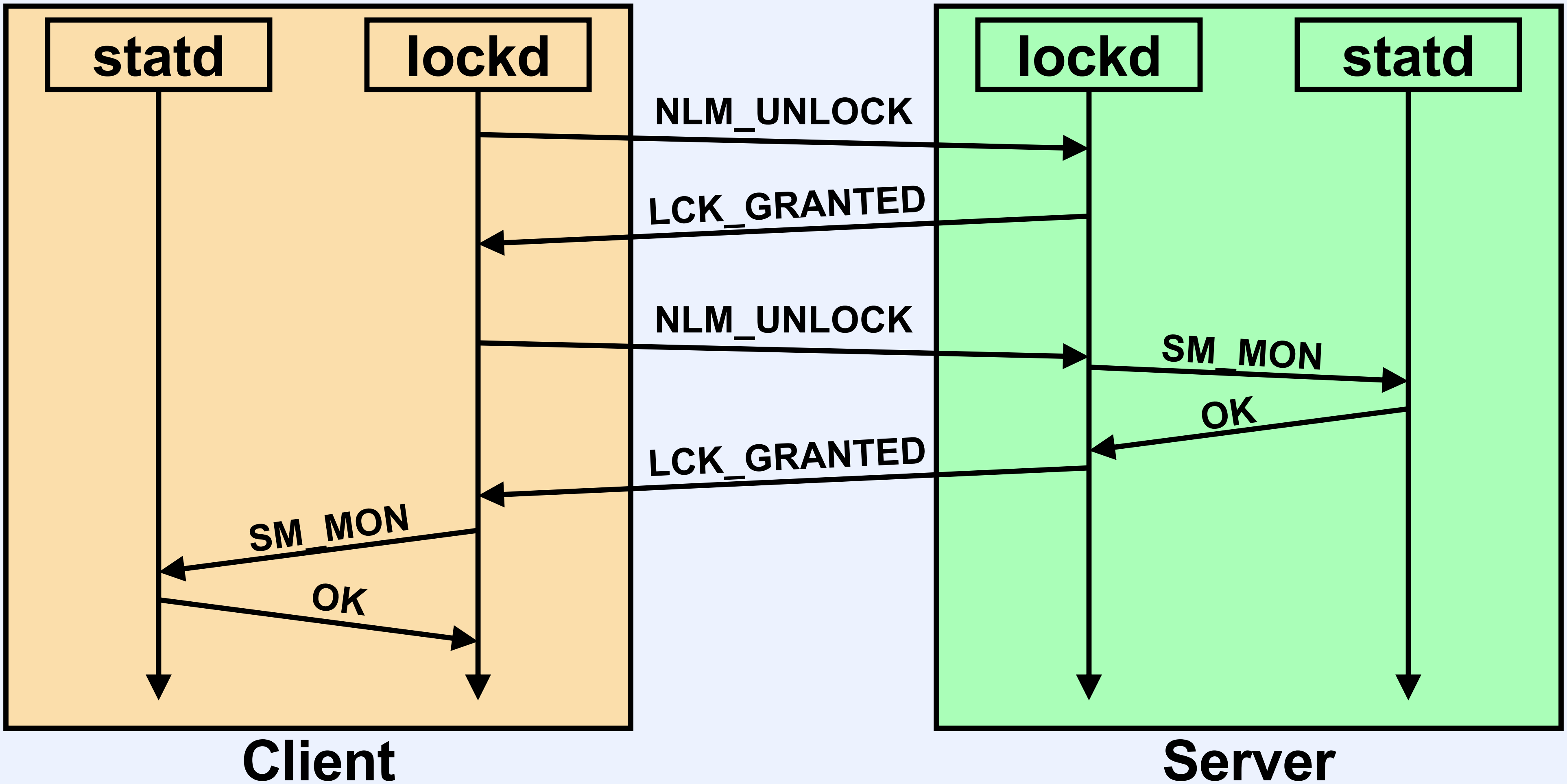
Status Monitor

- 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

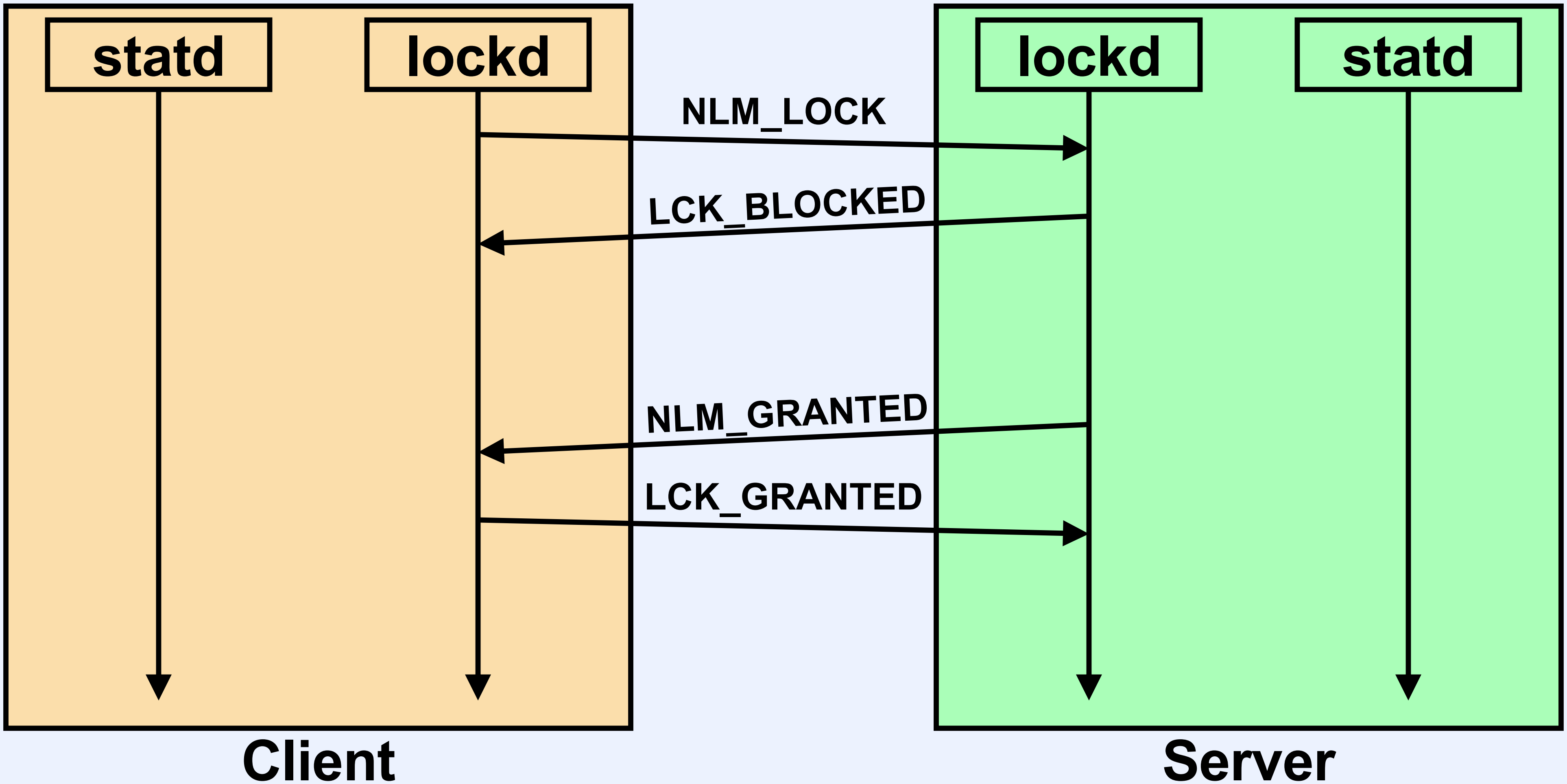
Locking a File



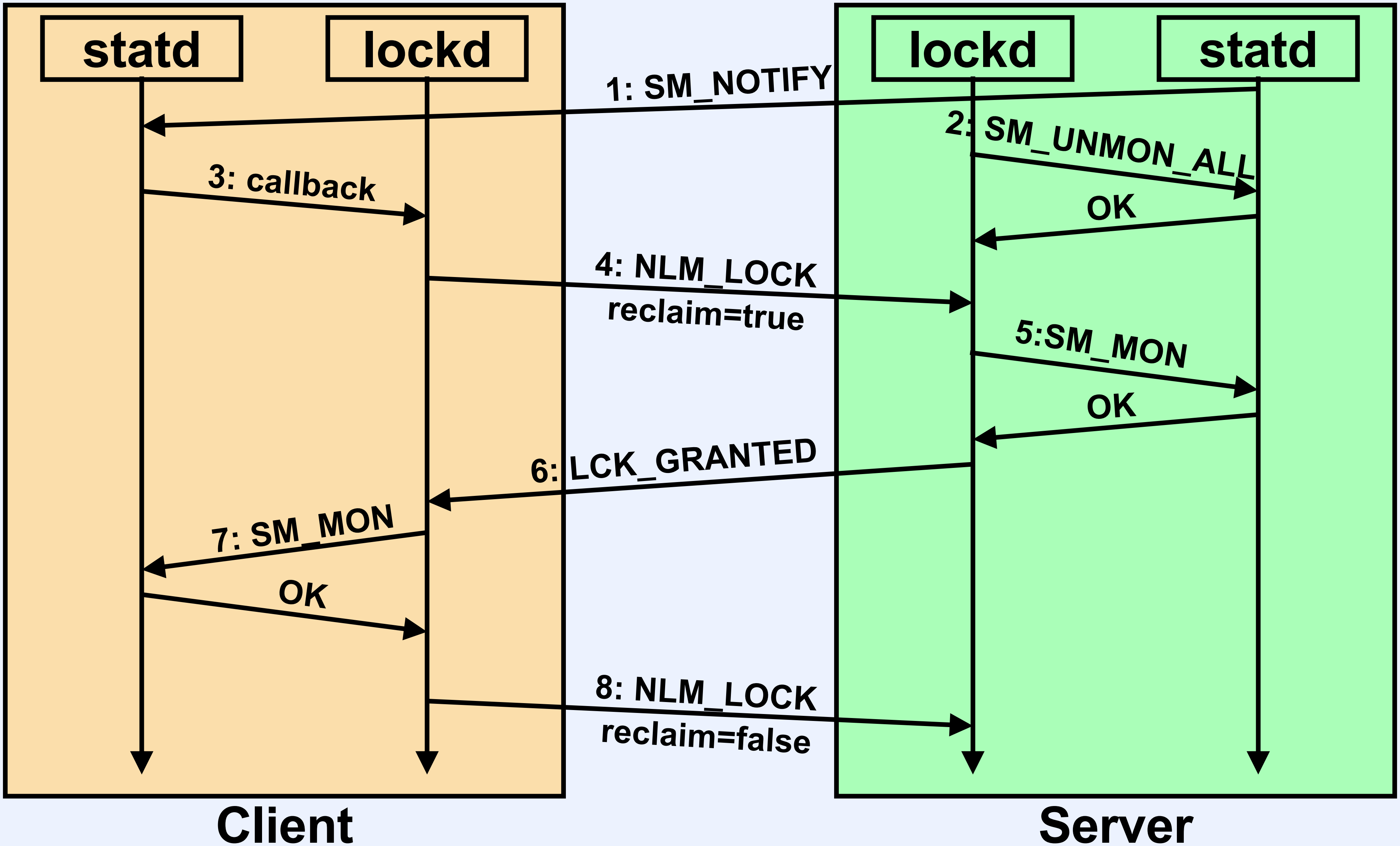
Unlocking a File



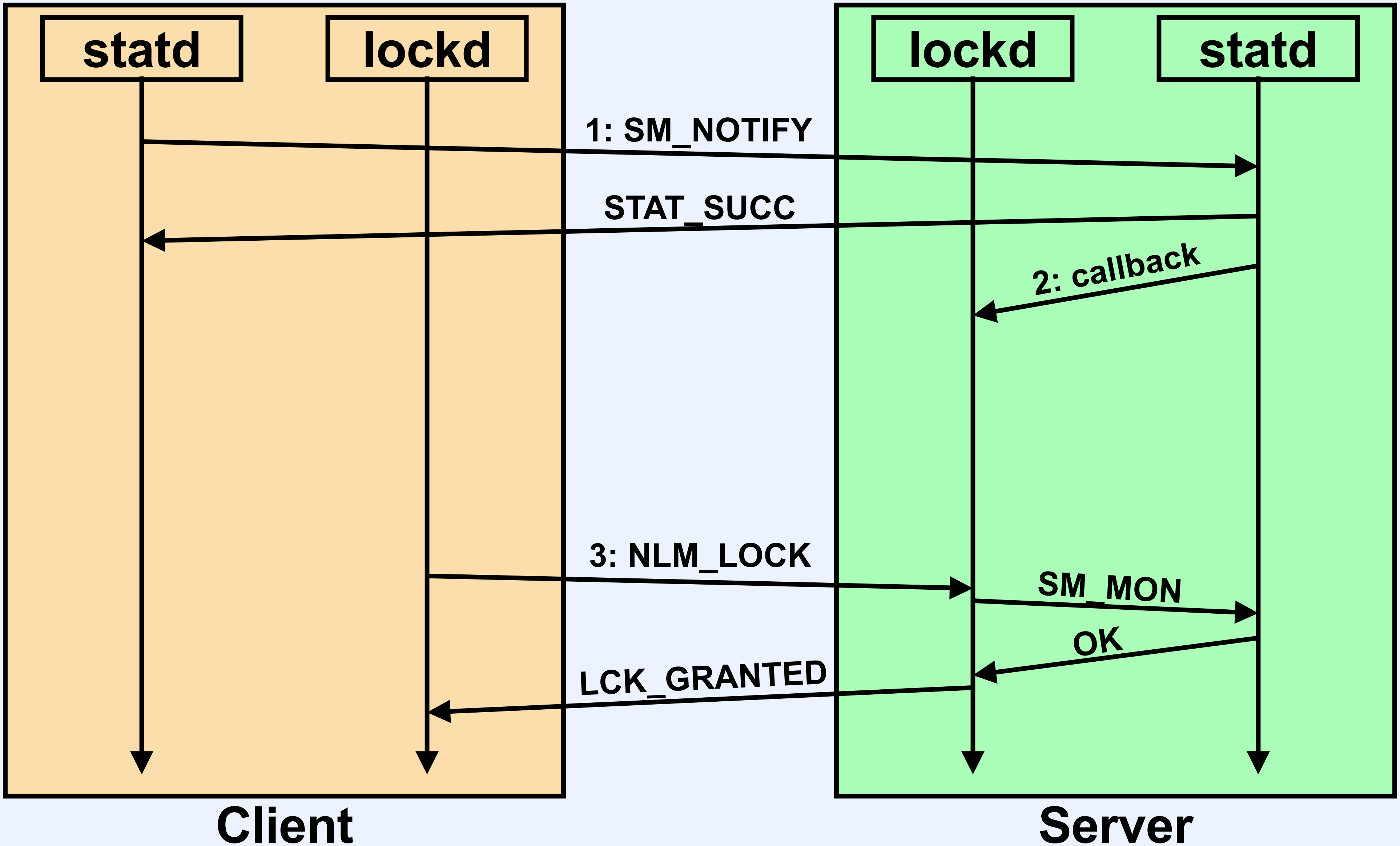
Delayed Locking



Server Crash Recovery



Client Crash Recovery



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

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***

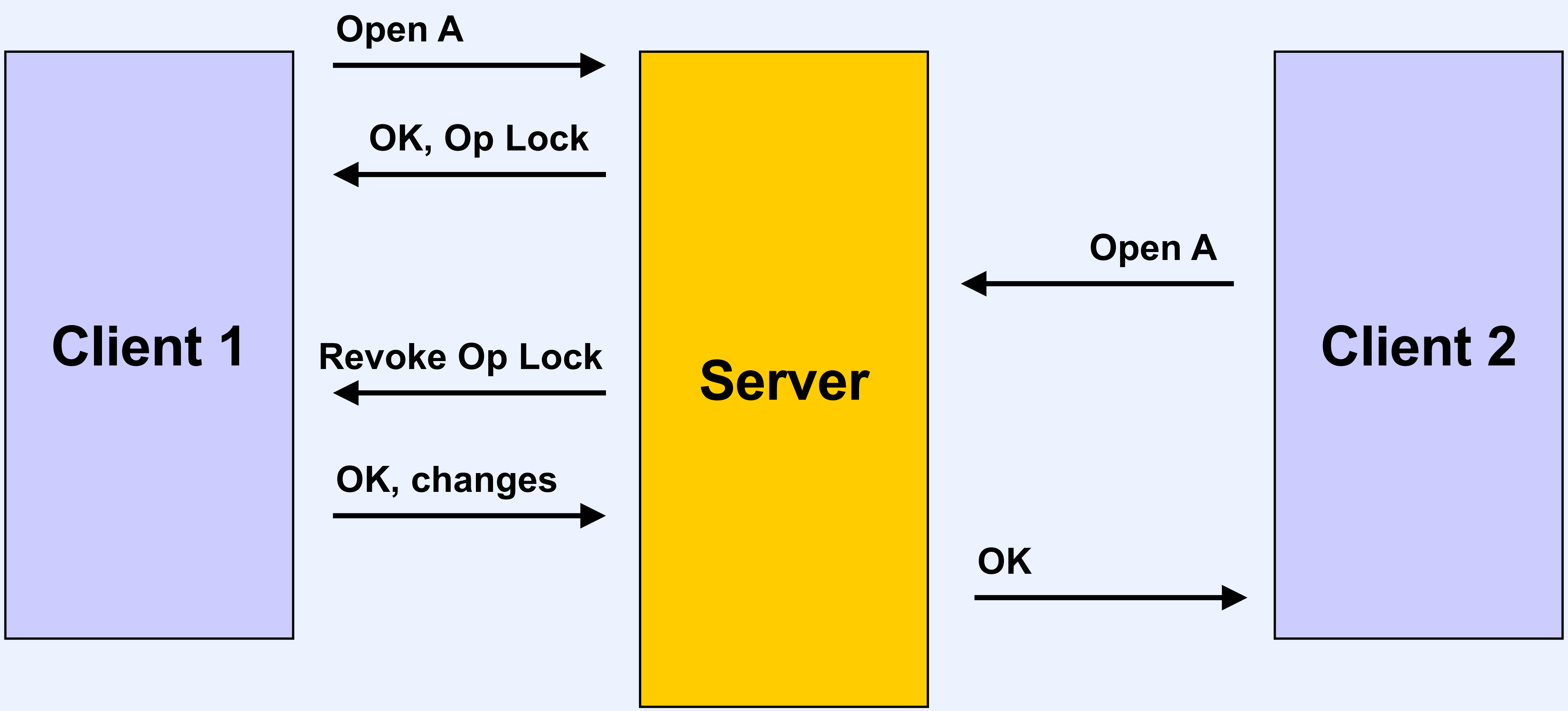
Consistency vs. Performance

- 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 …

  ø

  - or you can do opportunistic locking

Opportunistic Locks



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

NFS Version 4

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

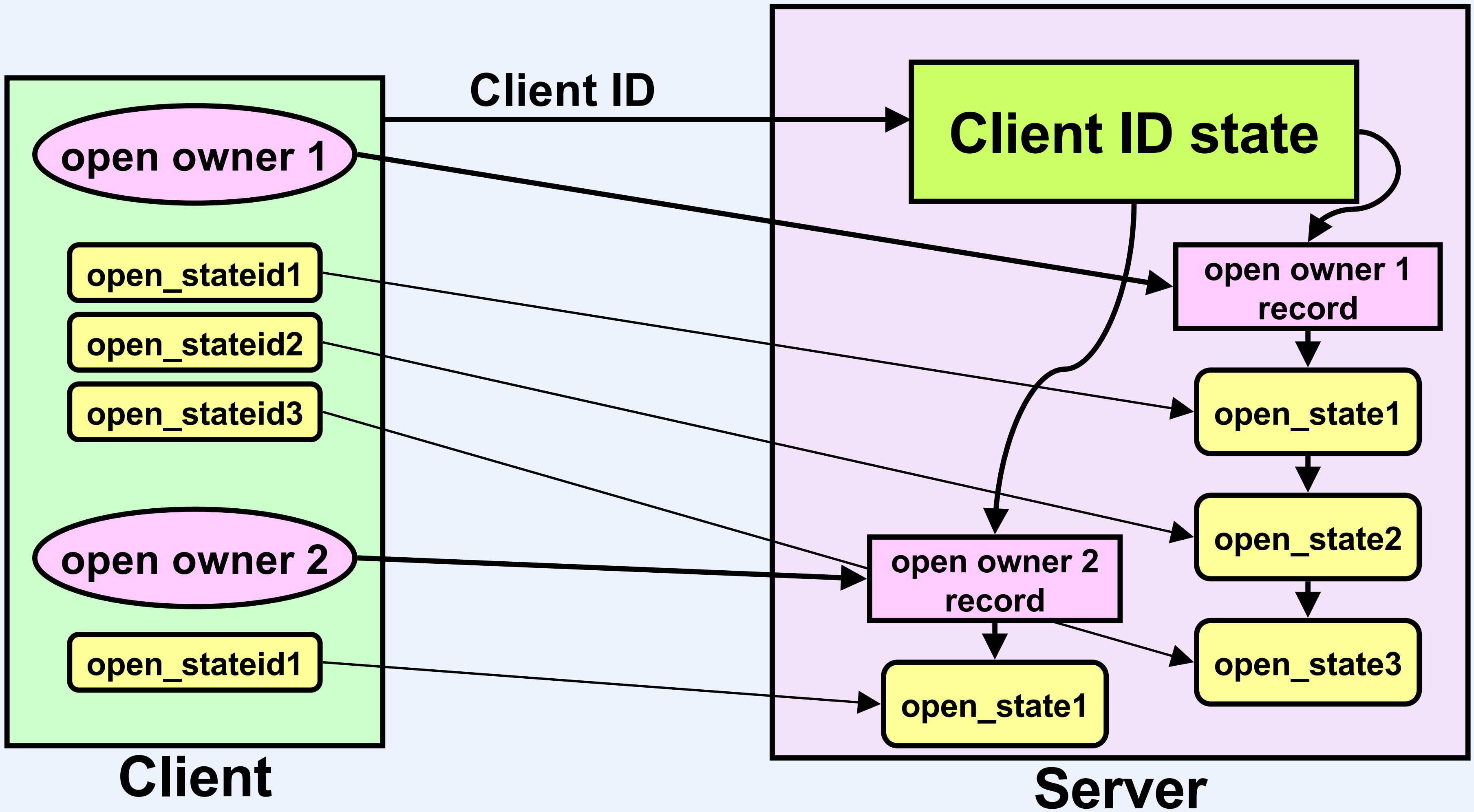
Client State

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

Server State

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

Open State

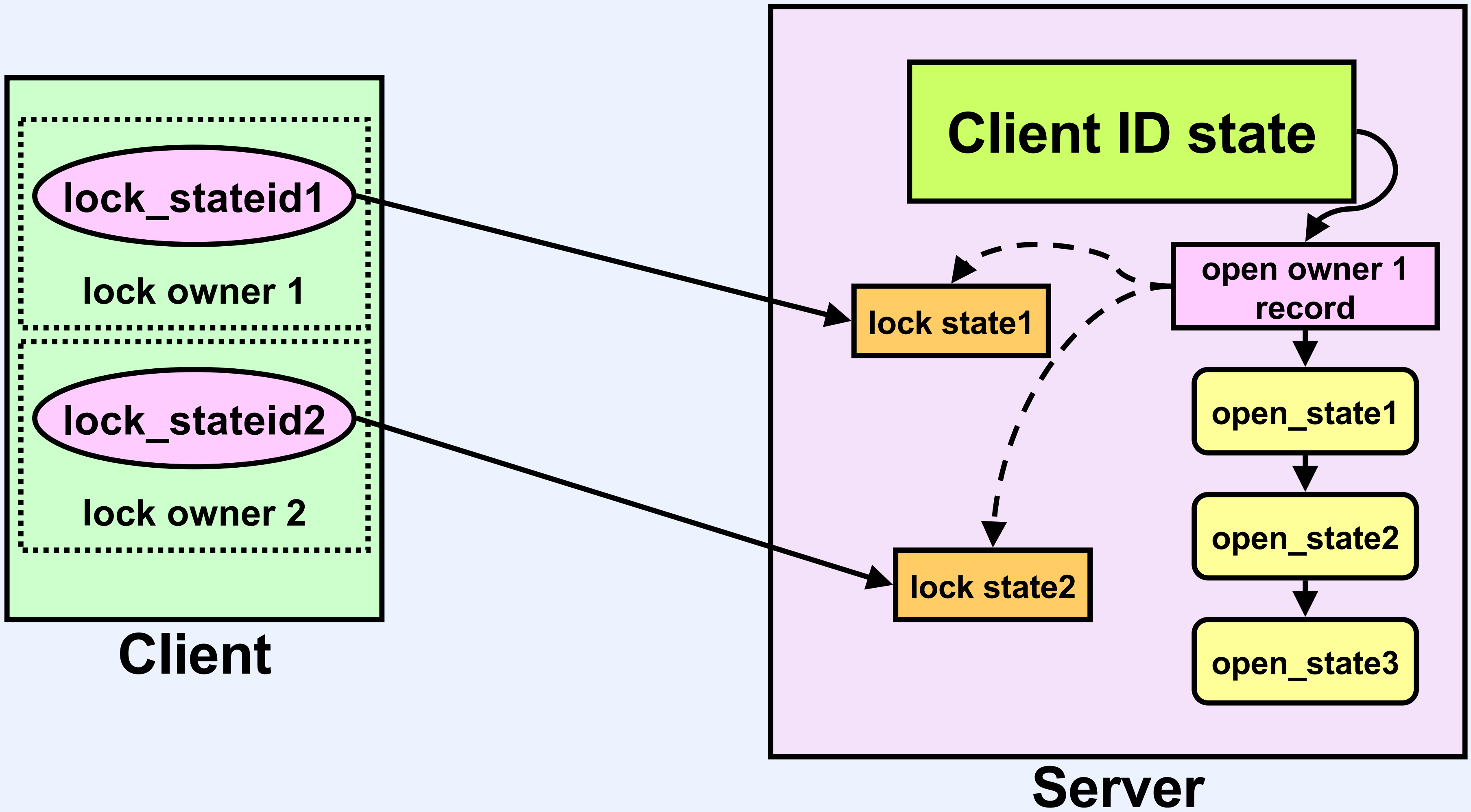


Client ID
open owner 1
open_stateid1
open_stateid2
open_stateid3
open owner 2
open_stateid1
Client
Client ID state
open owner 1 record
open_state1
open_state2
open_state3
open owner 2 record
open_state1
Server

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



lock_stateid1
lock owner 1
lock_stateid2
lock owner 2
Client
Client ID state
open owner 1 record
lock state1
lock state2
open_state1
open_state2
open_state3
Server

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

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

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

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