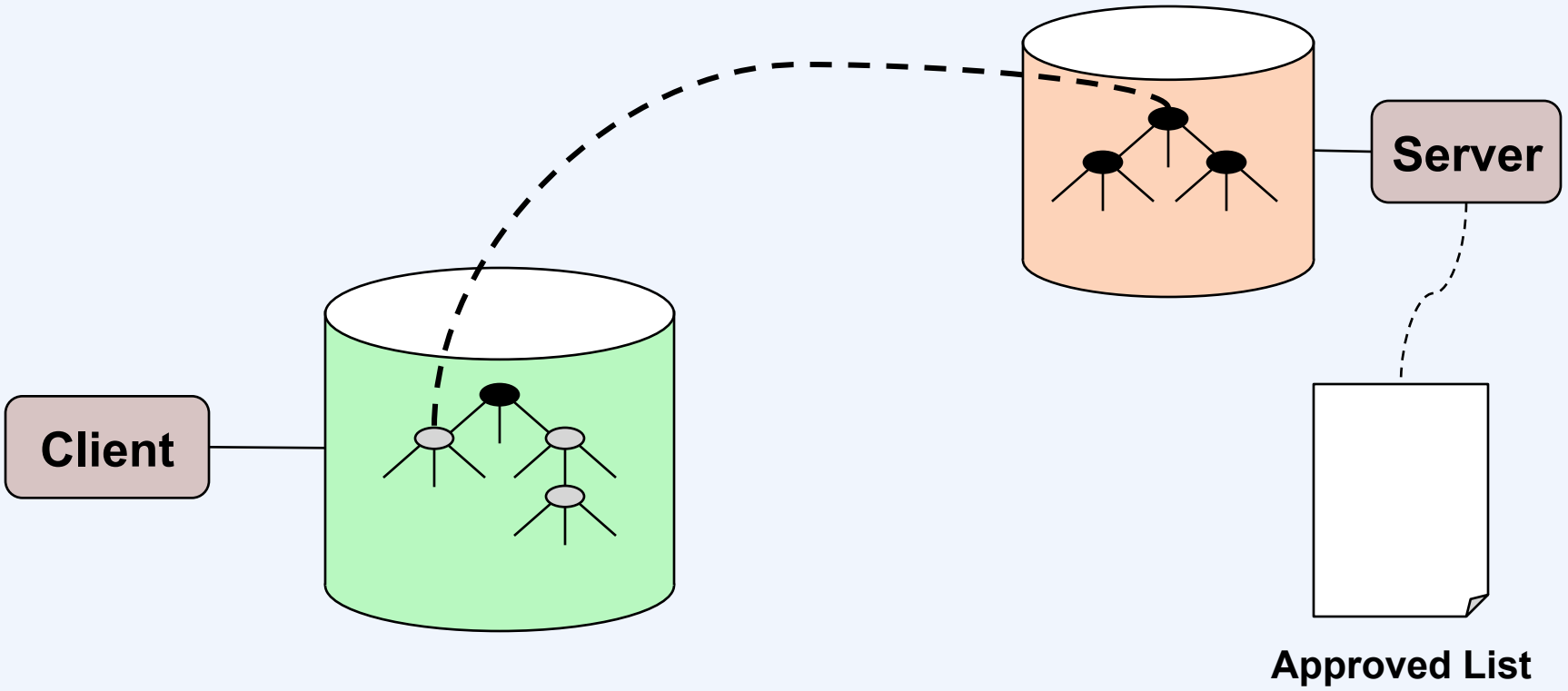
Distributed File Systems (Part 2)

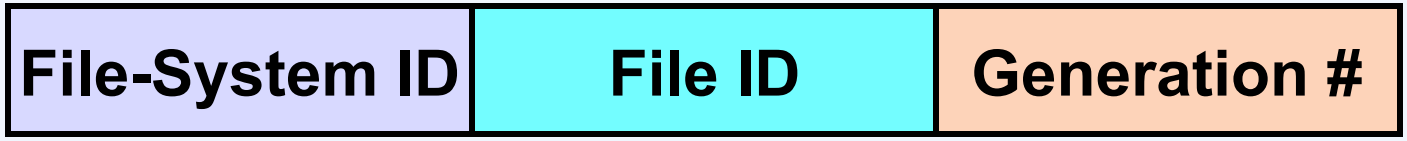
NFS Mount Protocol



File Handles

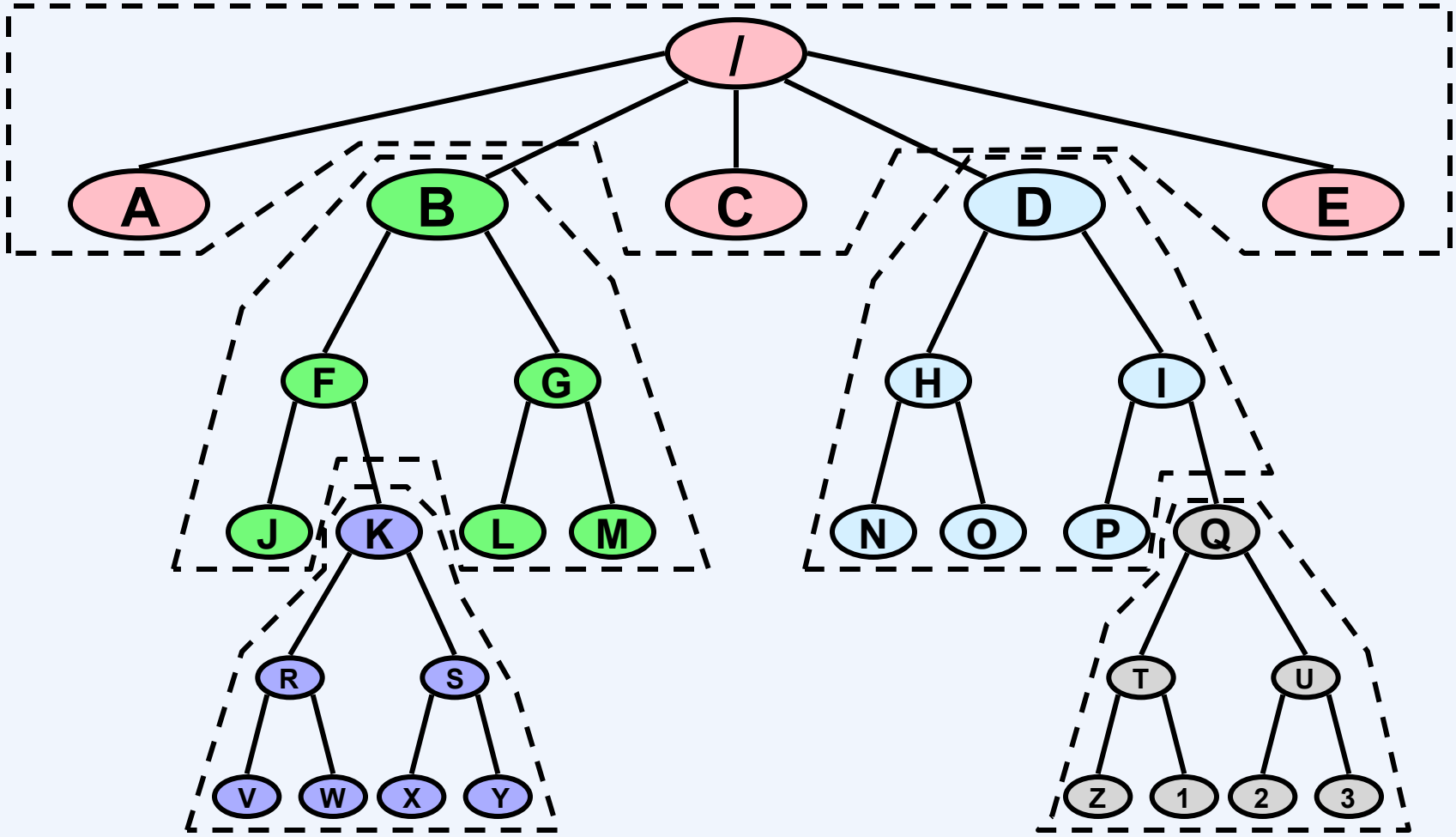
- Servers provide opaque *file handles* to clients to refer to files
  - contents mean nothing to clients
  - identify files on server
- Clients contact server via mount protocol to obtain file handles of roots of exported file systems

File Handle Contents

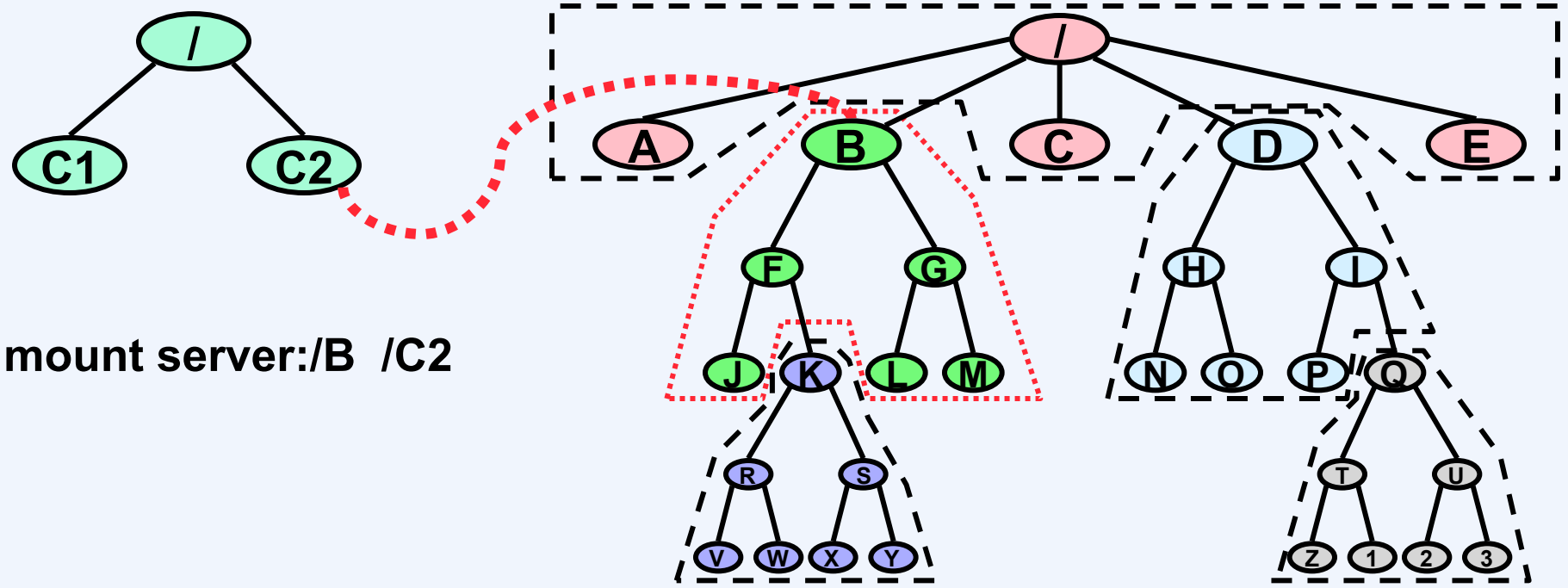


- File-System ID
  - which server file system
- File ID
  - which file within file system
- Generation #
  - guards against inode reuse

Server File Systems

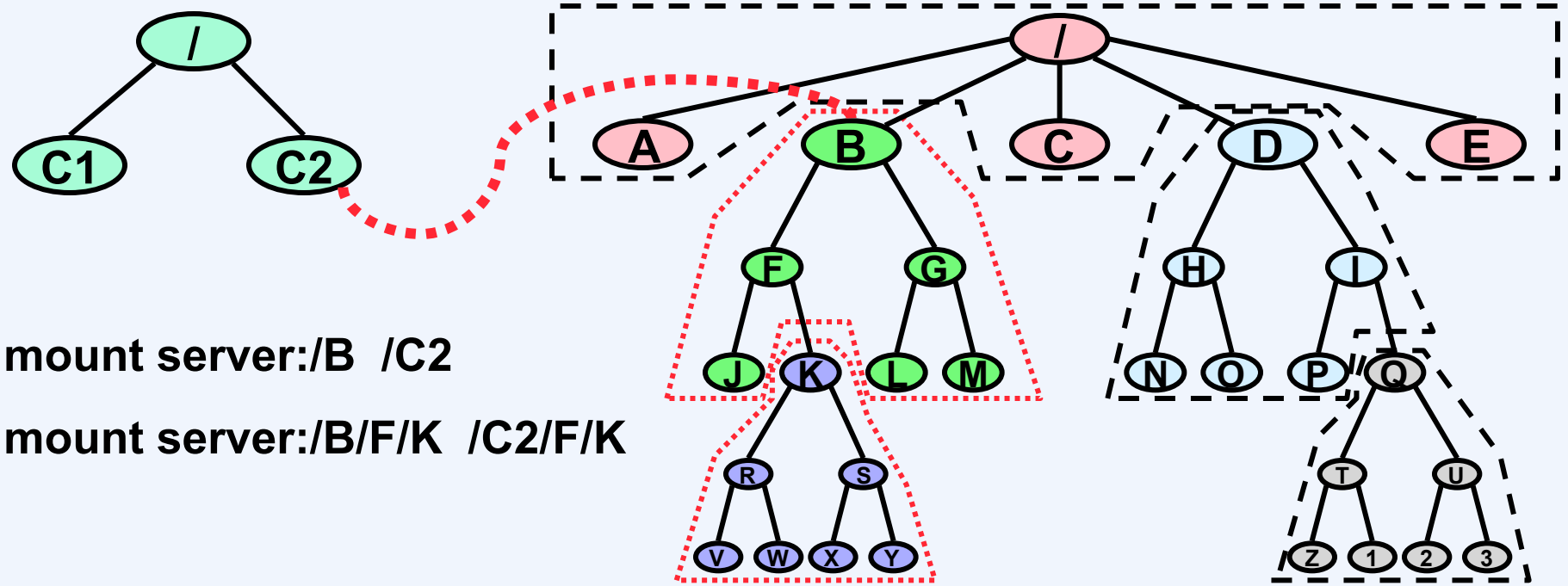


Client vs. Server Mount Points (1)



mount server:/B /C2

Client vs. Server Mount Points (2)



mount server:/B /C2

mount server:/B/F/K /C2/F/K

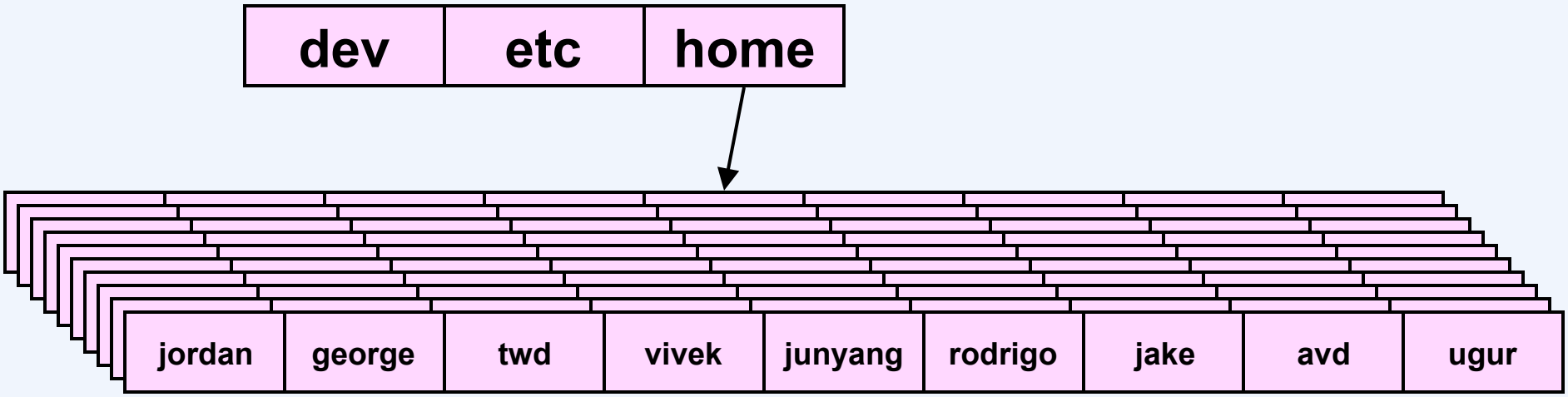
Local vs. Global Namespace

- Local namespace
  - each host configures its own file-system namespace
  - NFS clients each mount the appropriate remote file systems
- Global namespace
  - all hosts share the same namespace
  - not done in early NFS

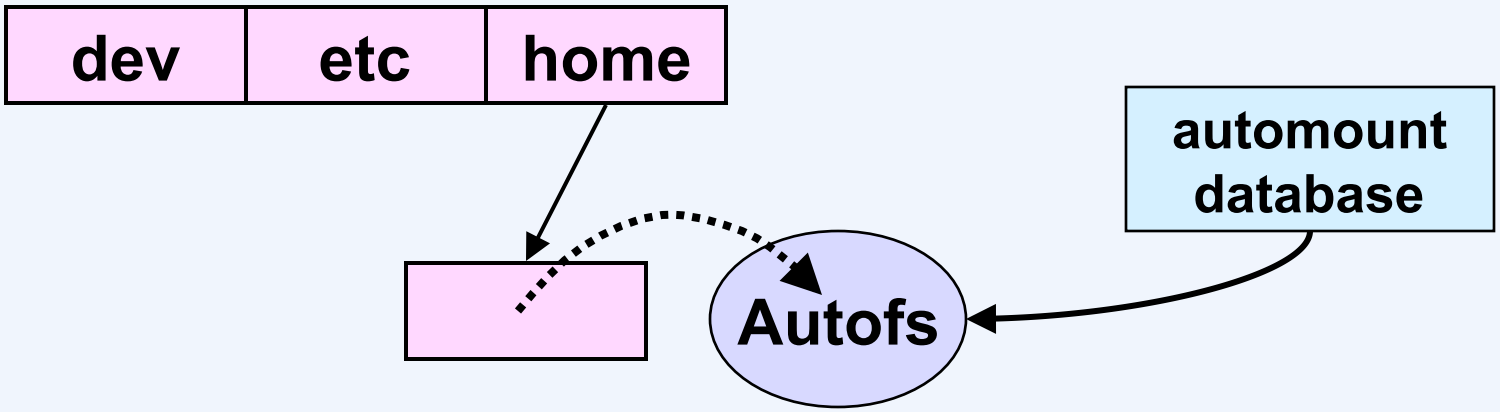
Mount Protocol Problems

- Local namespaces don't work
- Achieve global name space by having each client mount everything consistently
  - giving each client a table listing all possible mounts is administratively difficult
  - performing all possible mounts is time consuming
    - mounting is a "heavyweight" operation

Rather than this …



… this



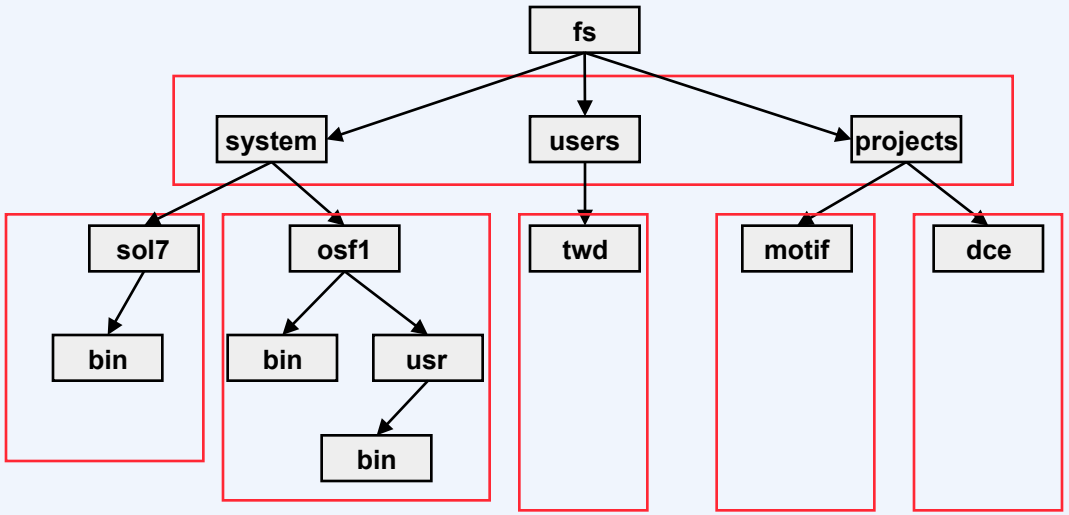
Automounting: 2000

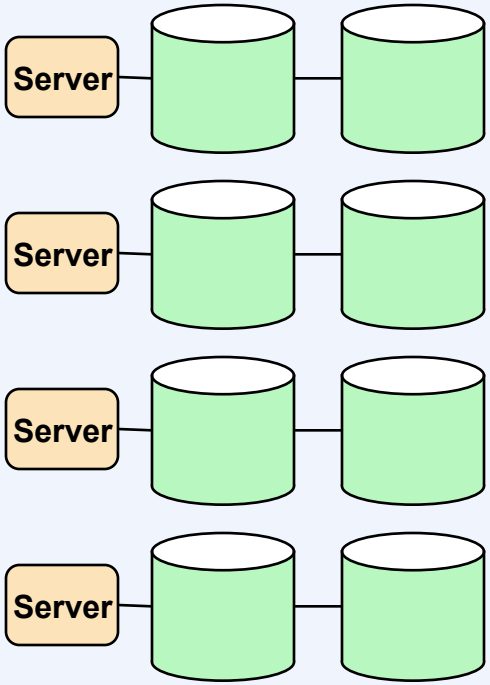
- Maintain description of global namespace in global database: NIS
- Do mounts only when needed
- Automount times out after period of unuse

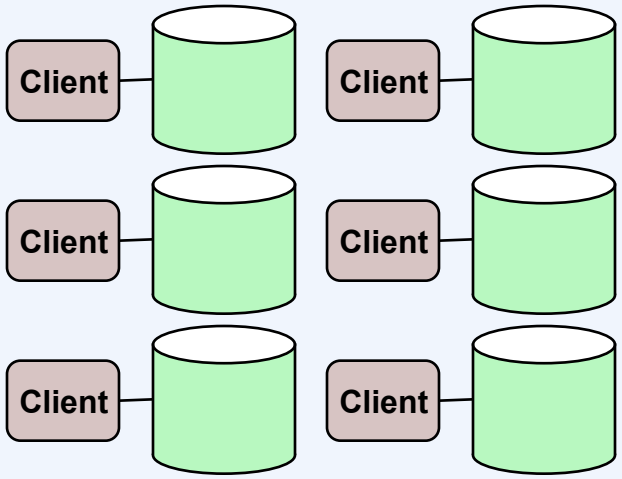
Automounting: 2016

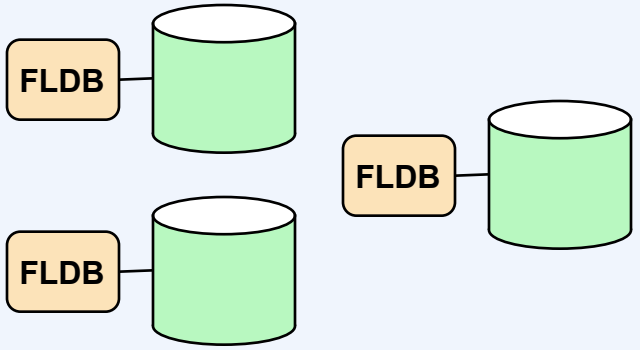
- Global namespace maintained in LDAP database
  - lightweight directory access protocol
    - vendor neutral
  - everything mounted at boottime
    - fewer, but larger, file systems
  - no timeout

DCE's DFS

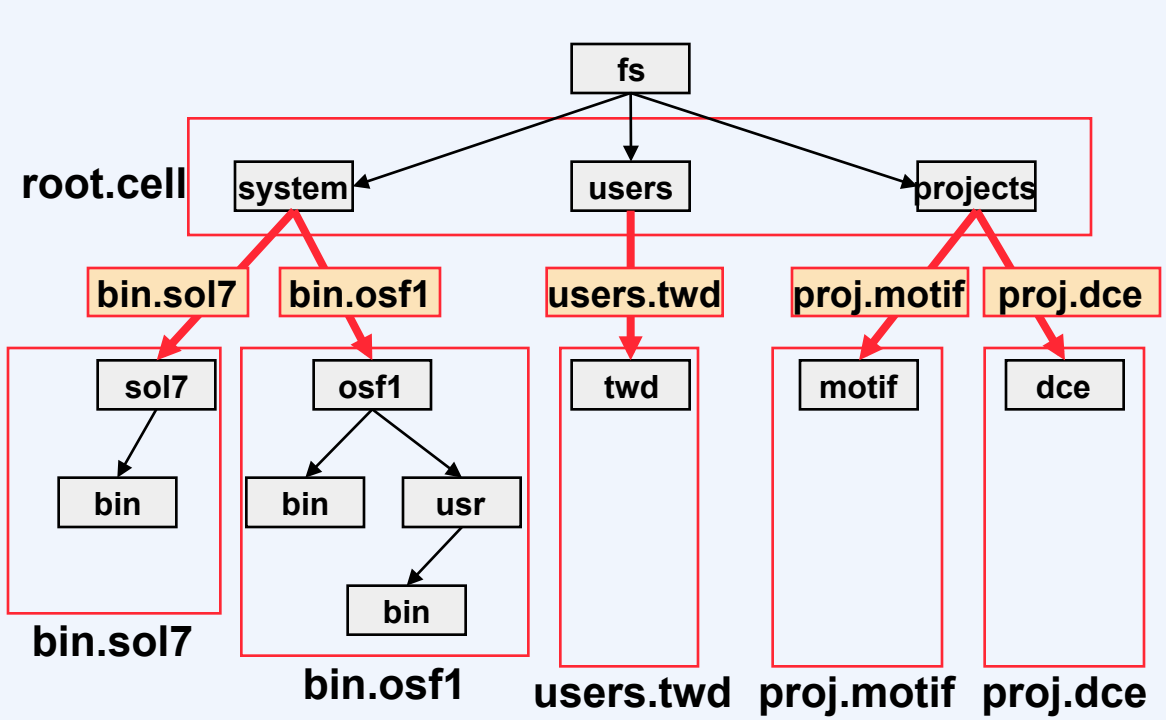


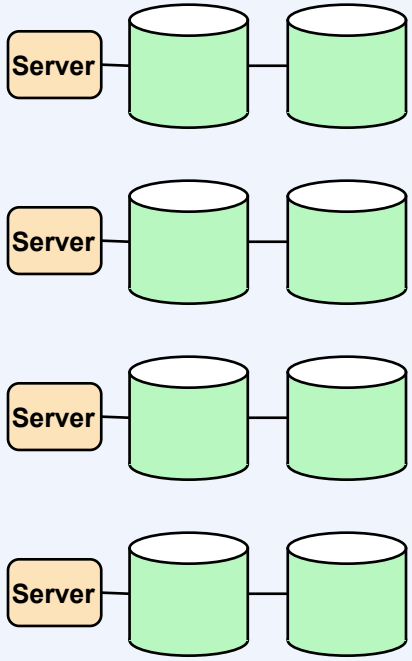


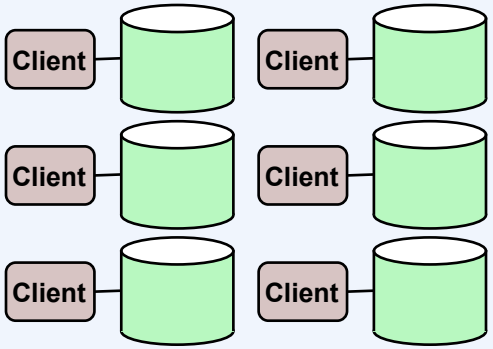


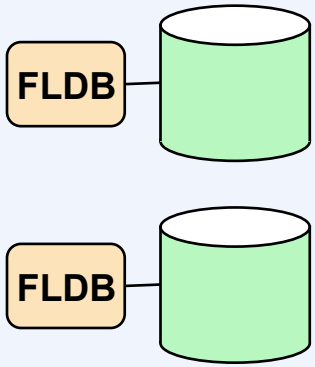


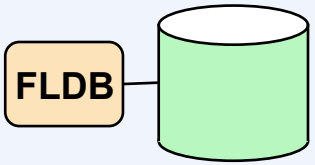
DFS Mount Points



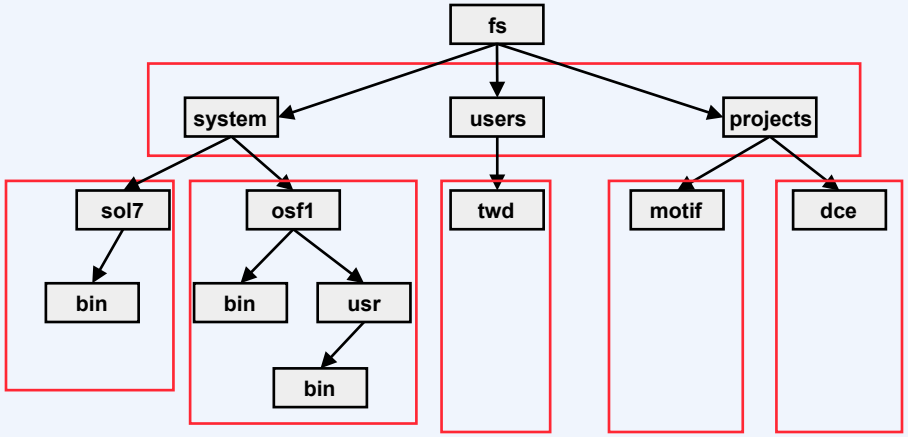


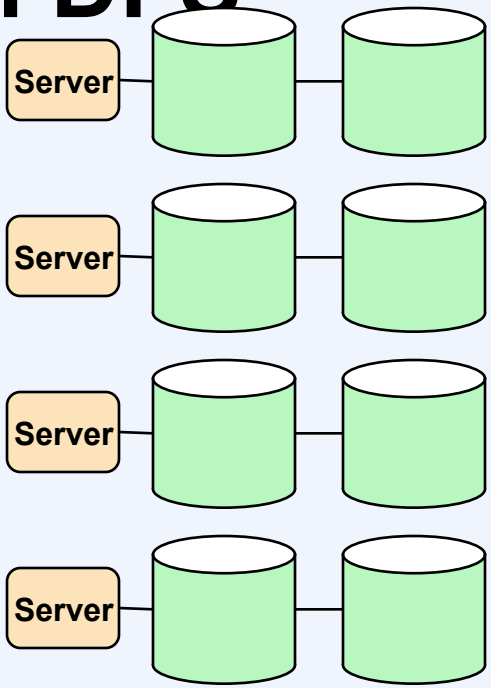


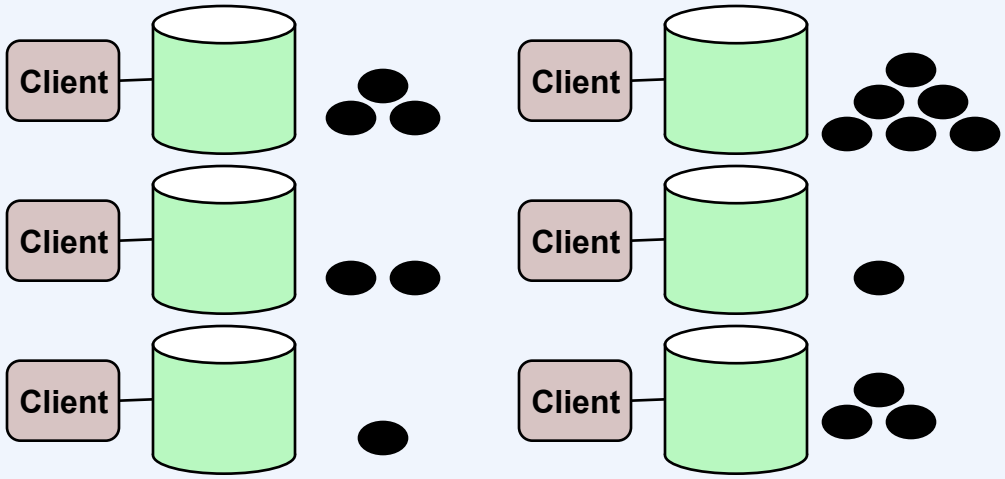


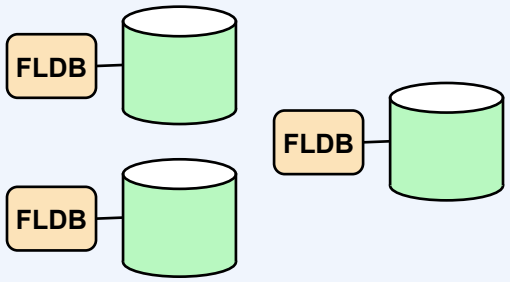


Strict Consistency in DFS

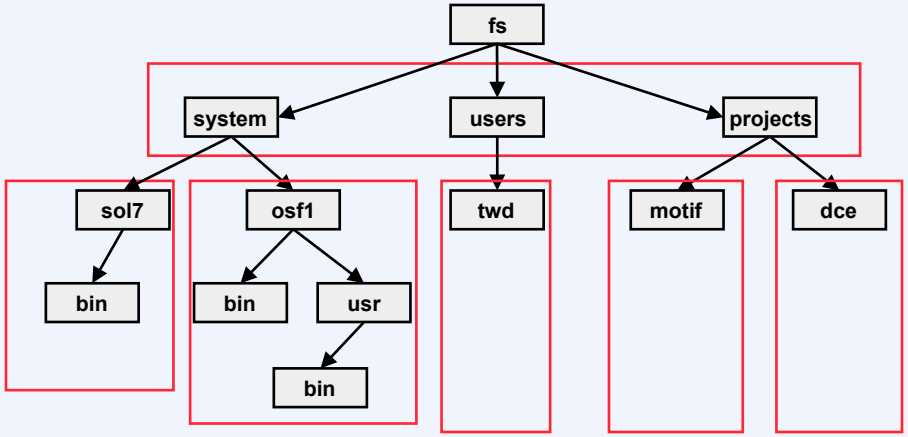


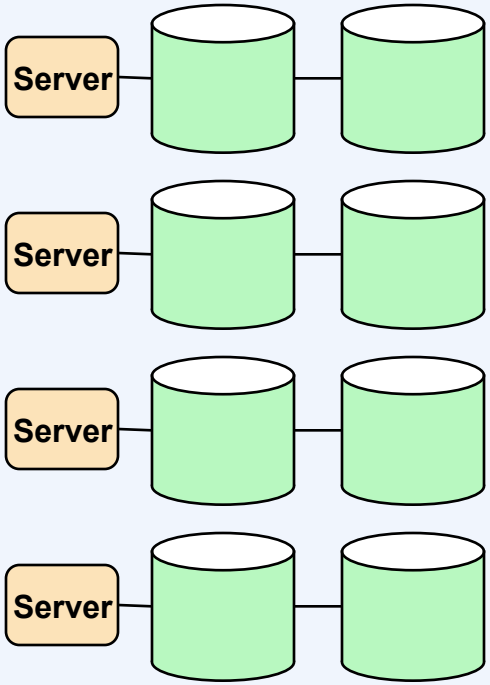


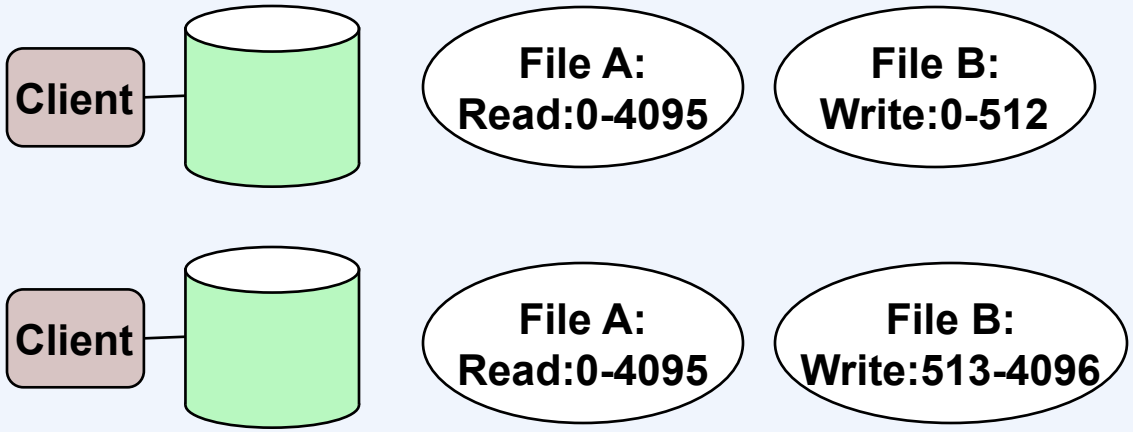


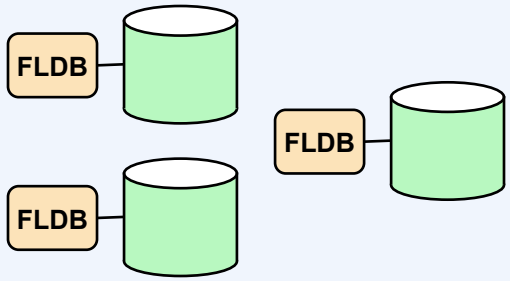


DFS Tokens (1)

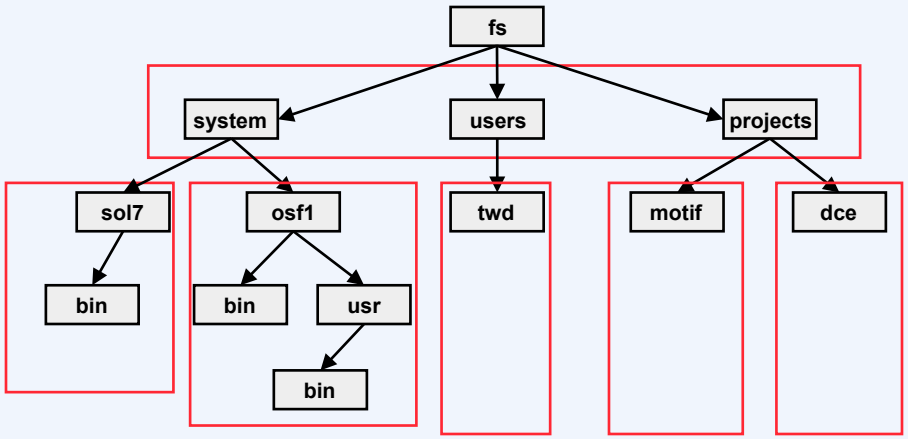


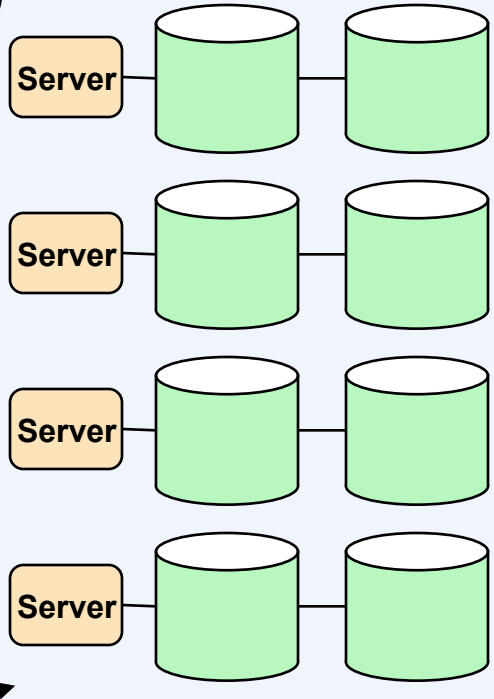


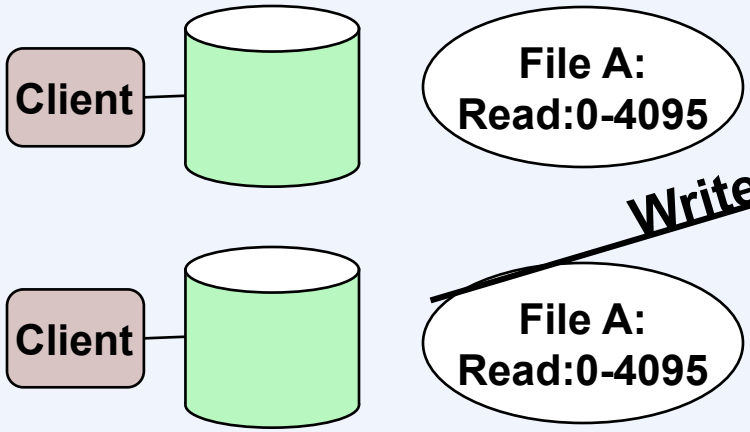


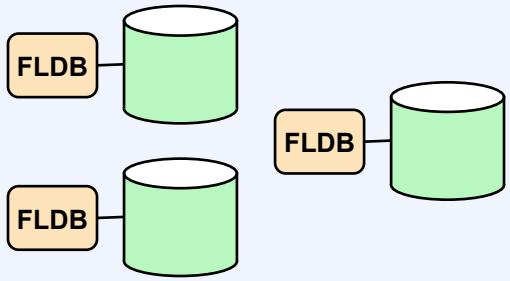


DFS Tokens (2)

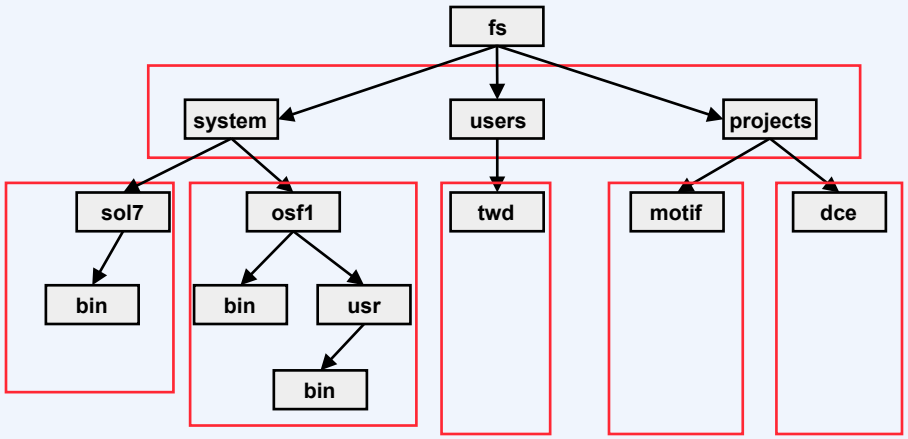


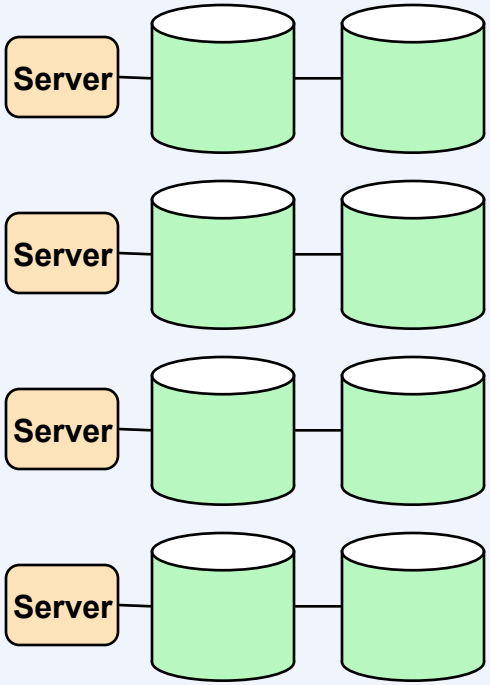


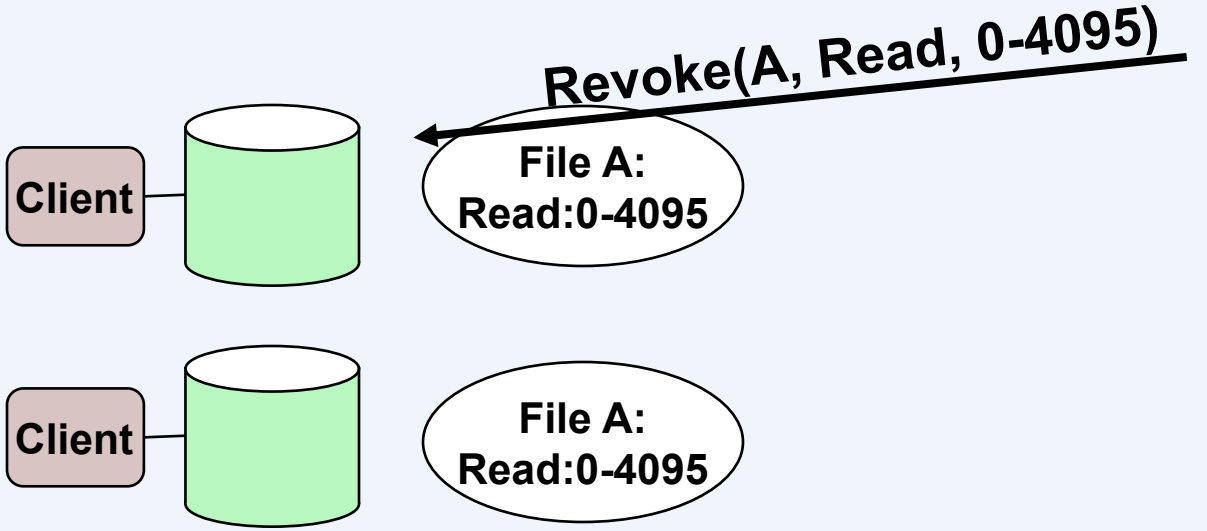


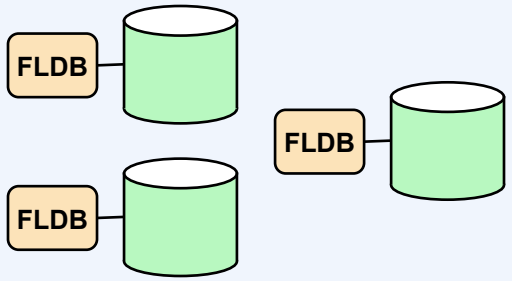


DFS Tokens (3)

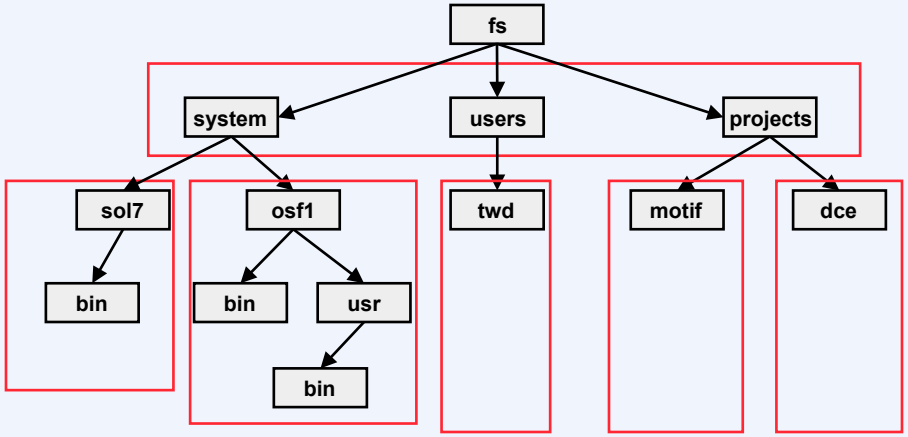


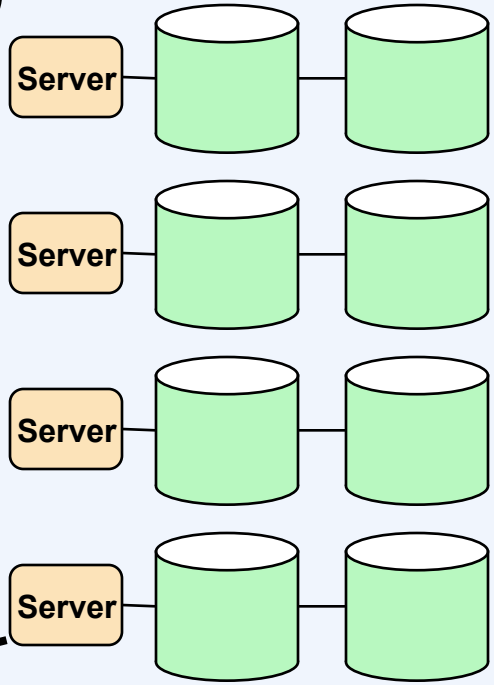


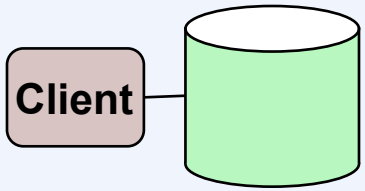


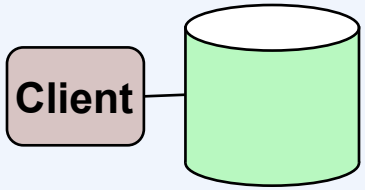


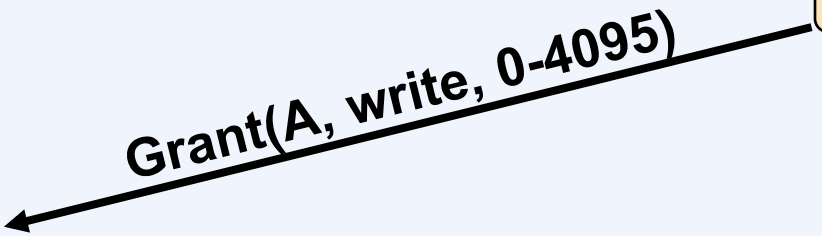
DFS Tokens (4)

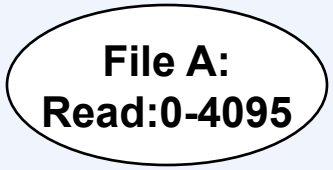


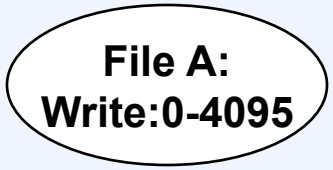


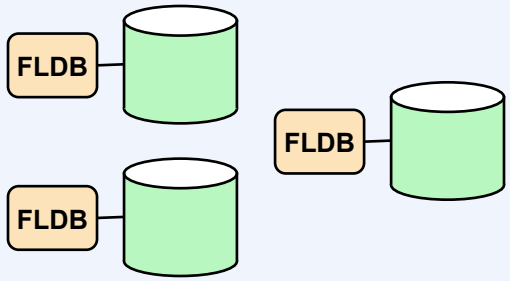




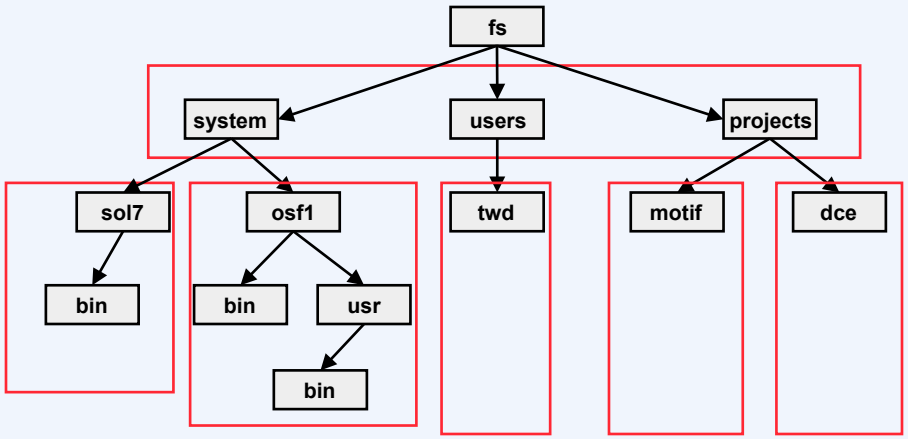


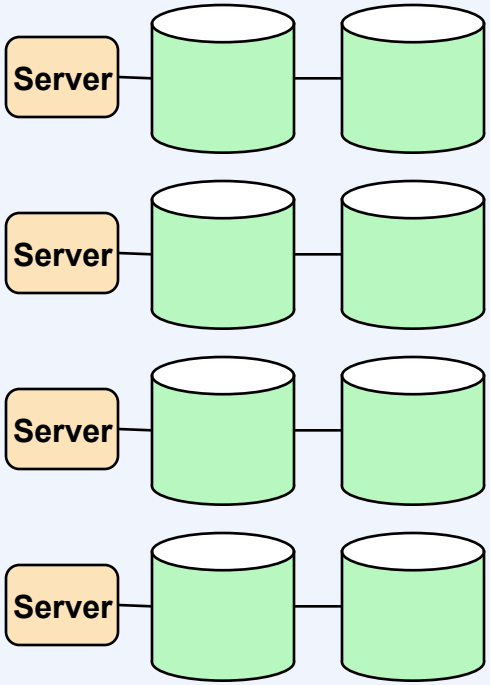


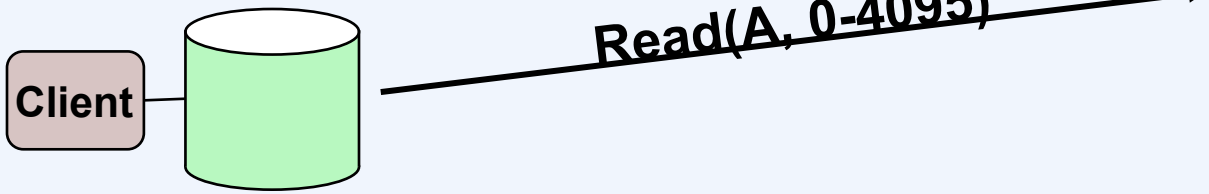


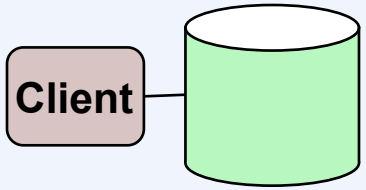


DFS Tokens (5)



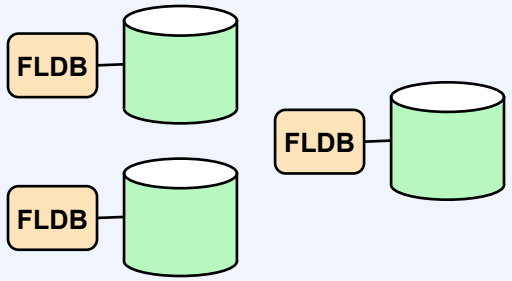




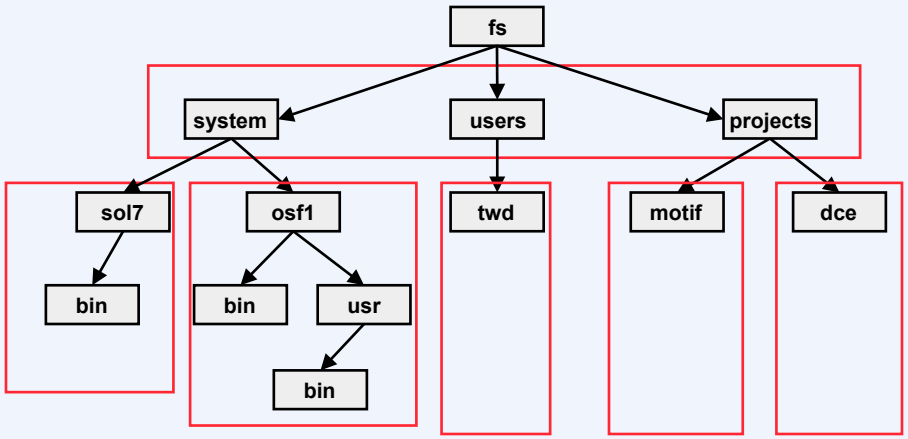


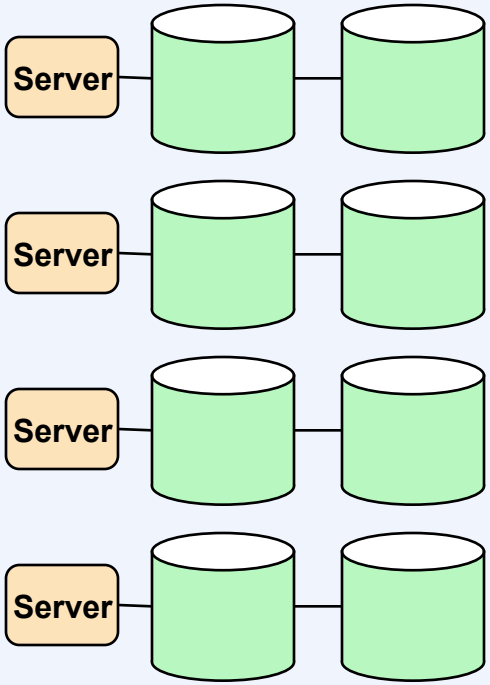
File A:
Read:0-4095

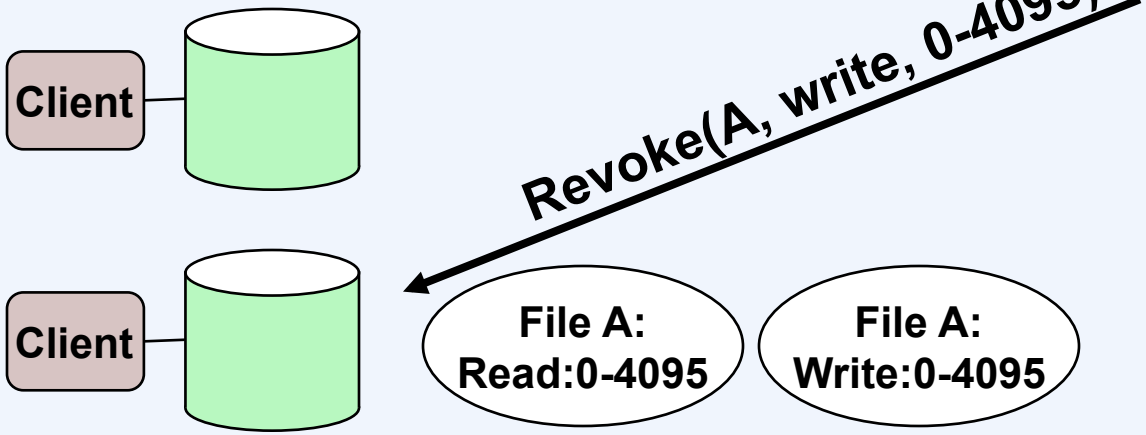
File A:
Write:0-4095

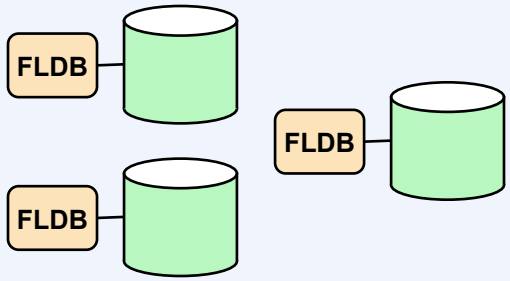


DFS Tokens (6)

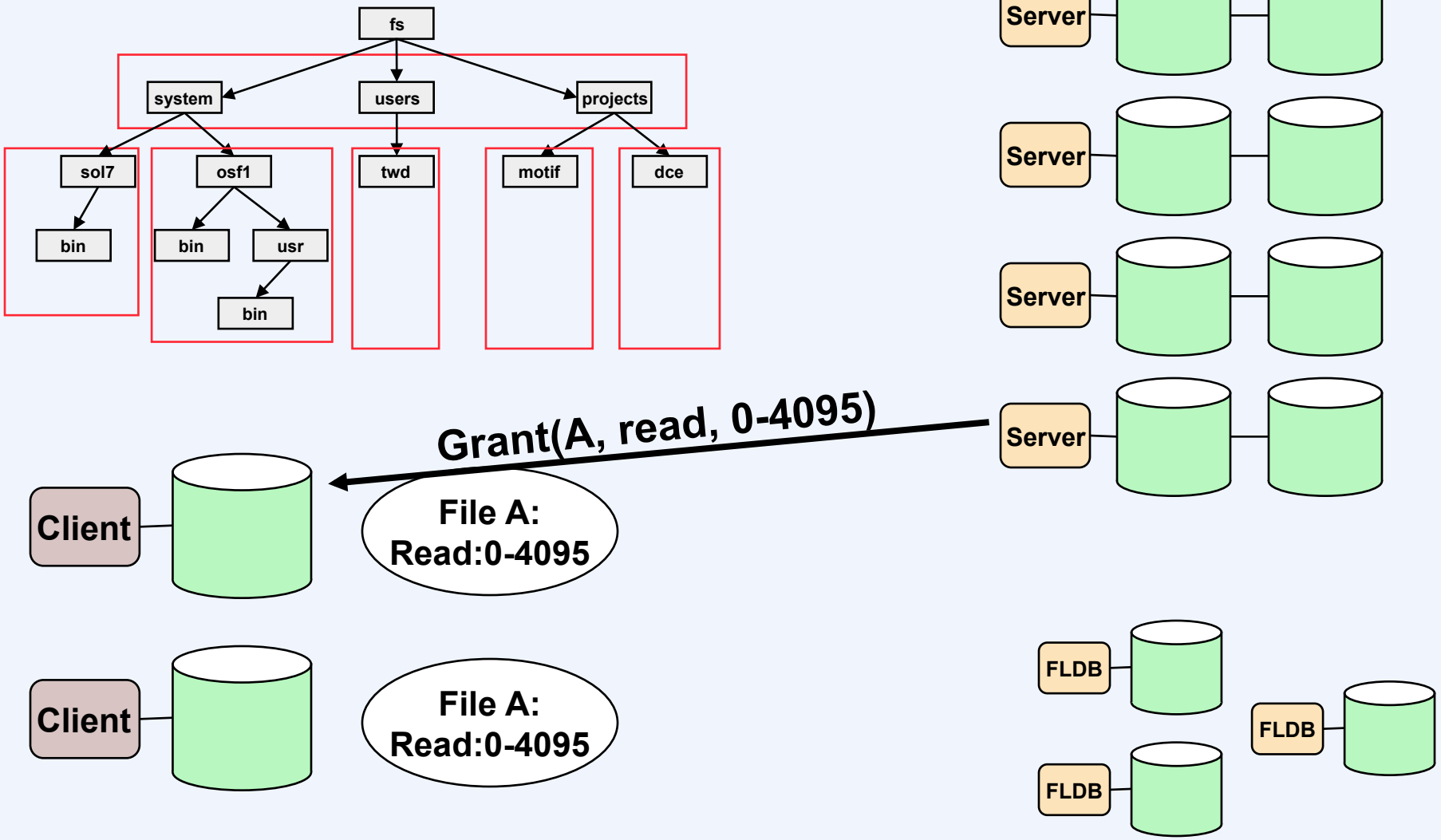




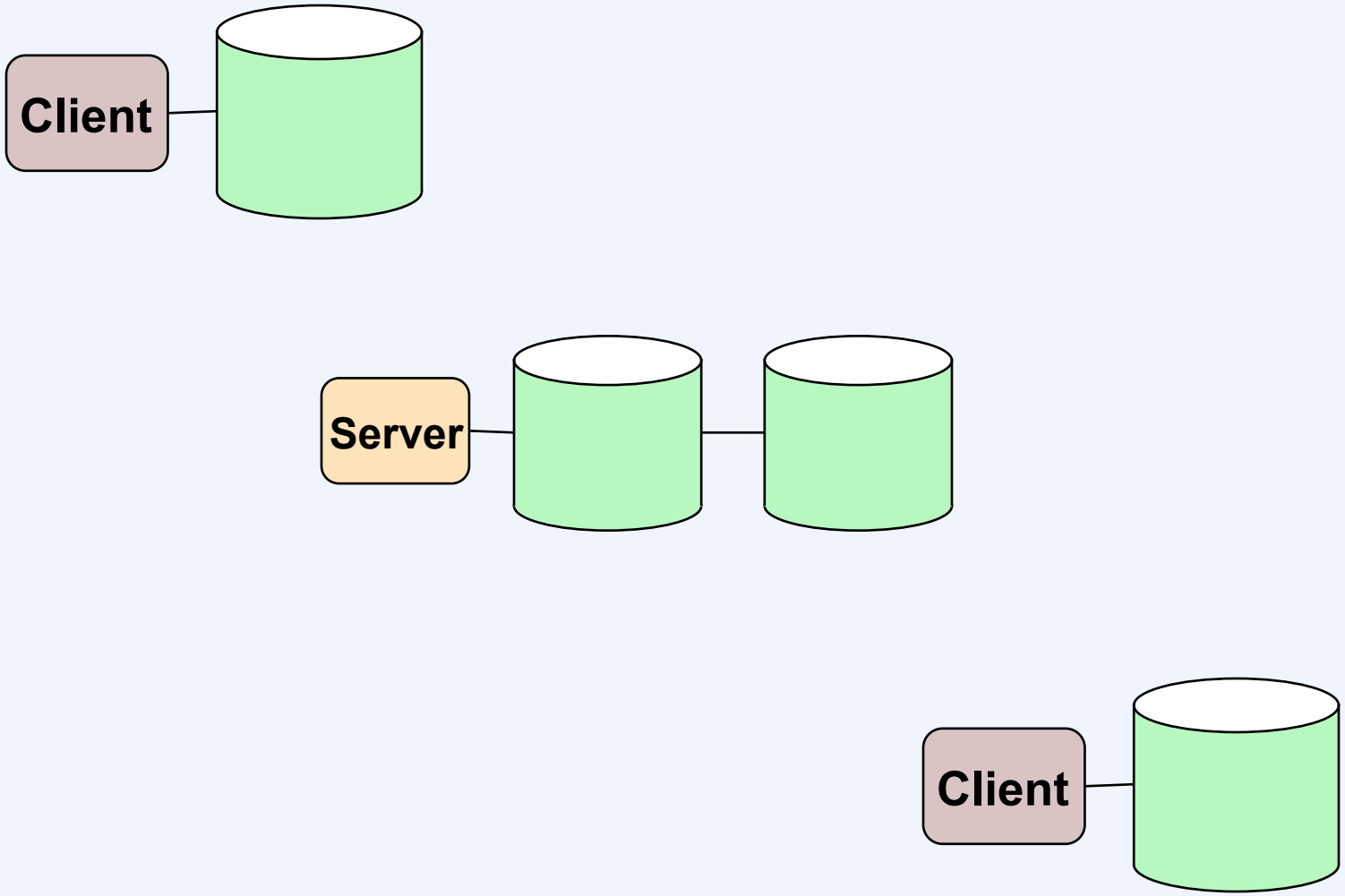




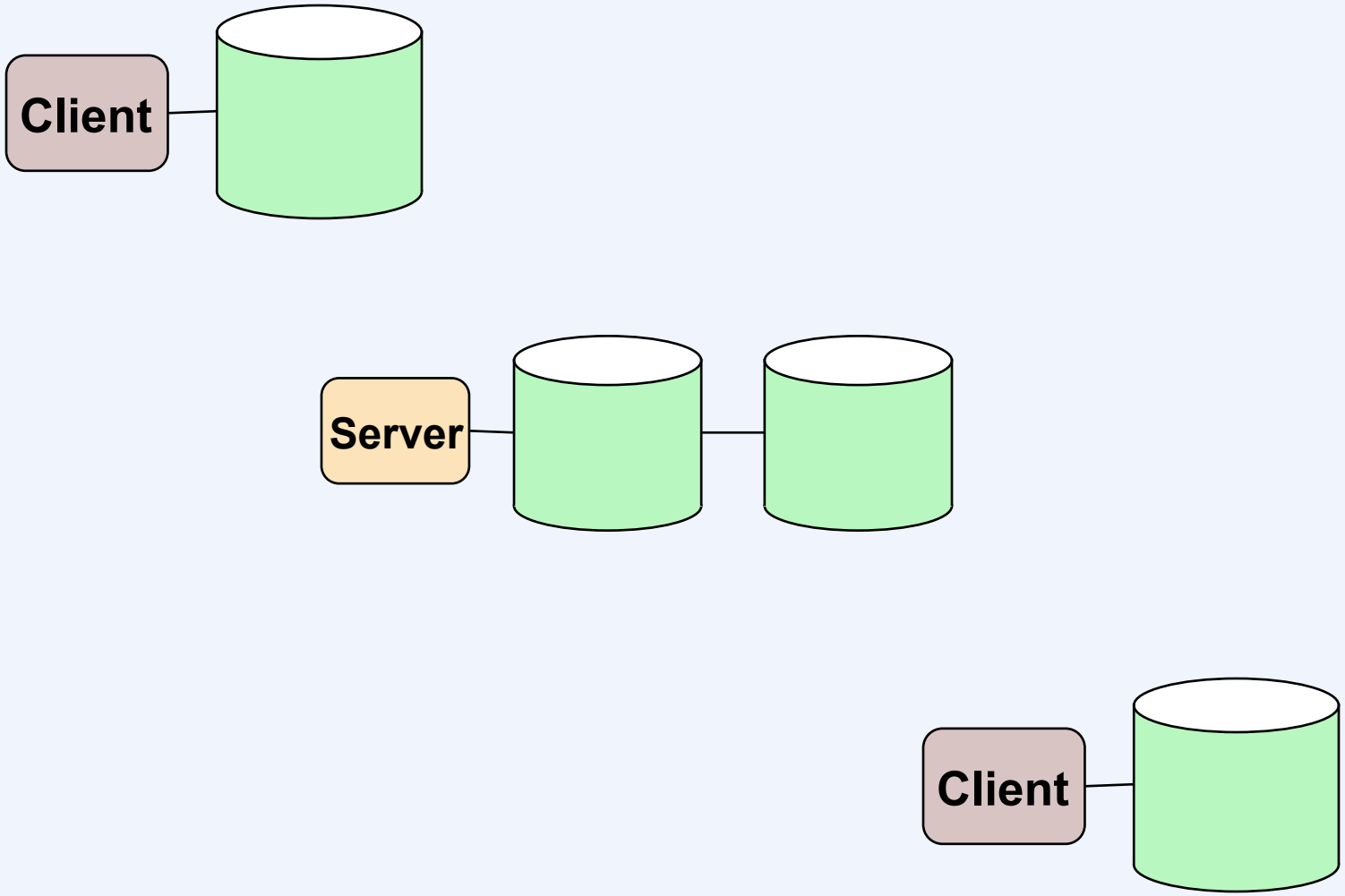
DFS Tokens (7)



DFS Crash Recovery



DFS Crash Recovery
(notes continued)



DFS Recovery Problems

- Client application must participate!
  - must recognize that operation returns "timed-out" error
  - must retry
- Due to semantics of tokens, it isn't feasible to provide NFS-style hard mount

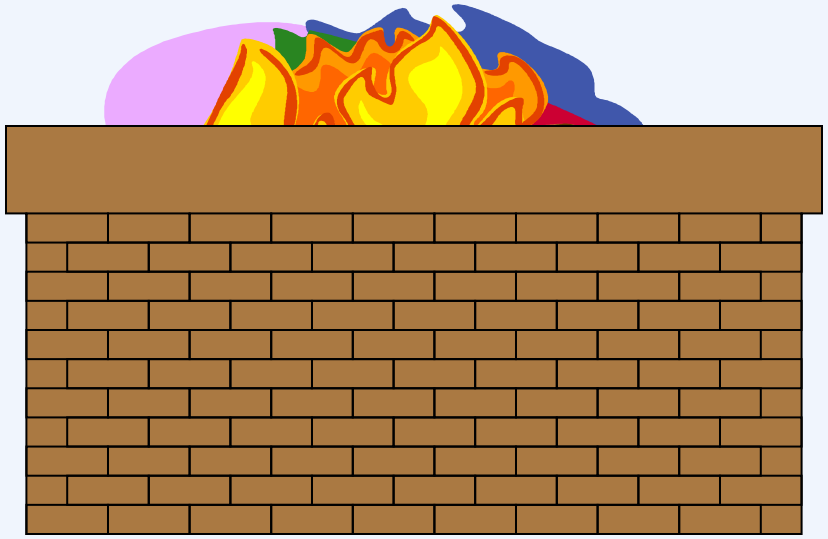
NFS Version 4

- Better than …
  - NFS version 2
  - NFS version 3
  - CIFS
  - DFS
  - (why aren't we running it?)

NFSv4: Why?

- Problems with NFSv3
  - firewalls
  - coordination
  - pathological network problems
  - security
  - high performance
  - no support for Windows clients

Firewall Issues

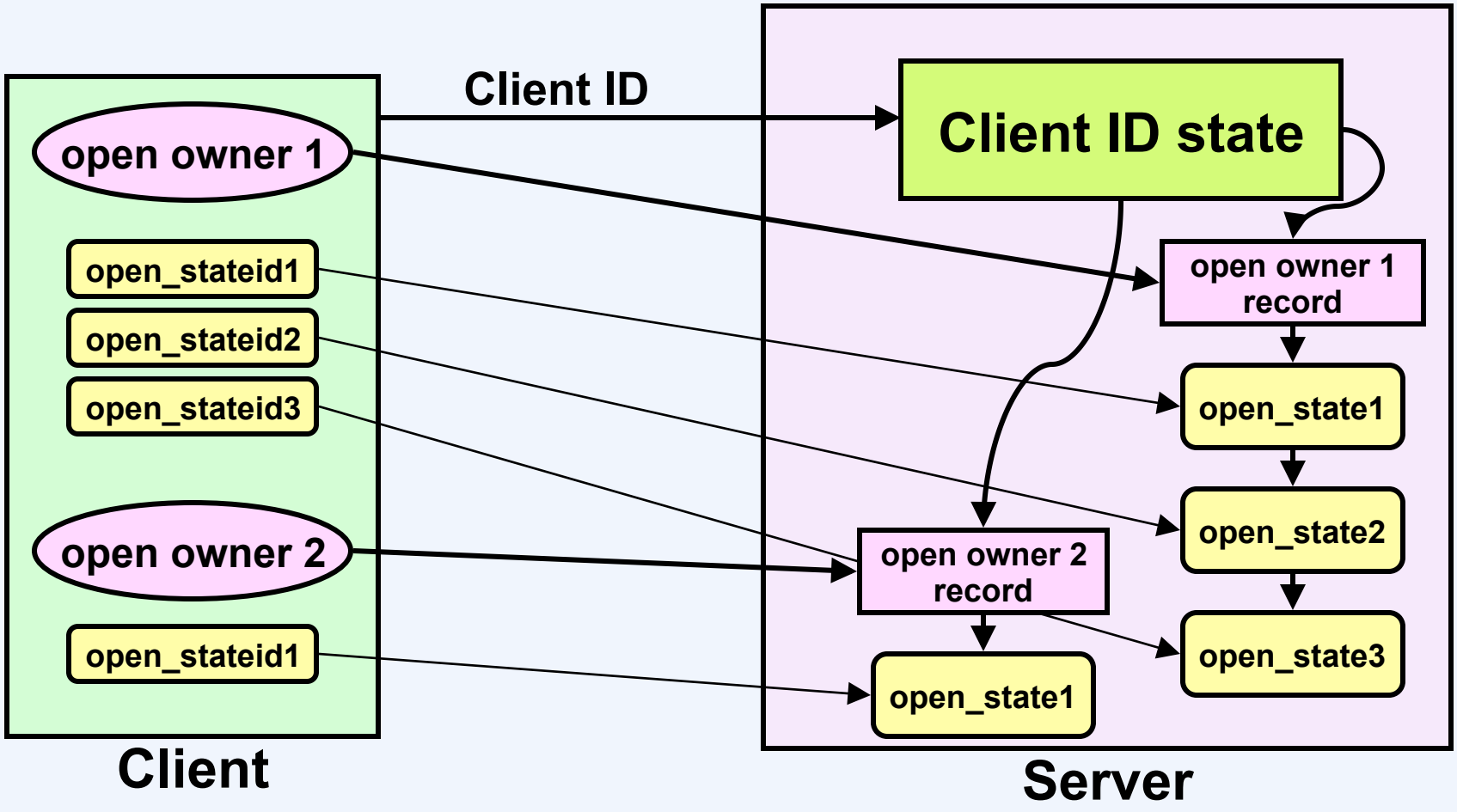


- Port numbers
  - NFS (v2 and v3) protocol uses 2049
  - mount, network lock manager, and network status manager protocols use dynamic ports
    - firewalls often allow access only to certain standard ports
    - it may be impossible to access dynamic ports from outside a firewall
- Client and Server behind (different) firewalls

Server State

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

Open State



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

Implementing Locks Right …

- Handle both Unix and Windows
- Get the semantics right
  - both advisory and mandatory
  - who owns a lock?
- Handle failures sanely
- Make it efficient
  - client-side caching where possible
- Make it doable
  - if lock not currently available
    - server might not be able to send callback
      - client polls server

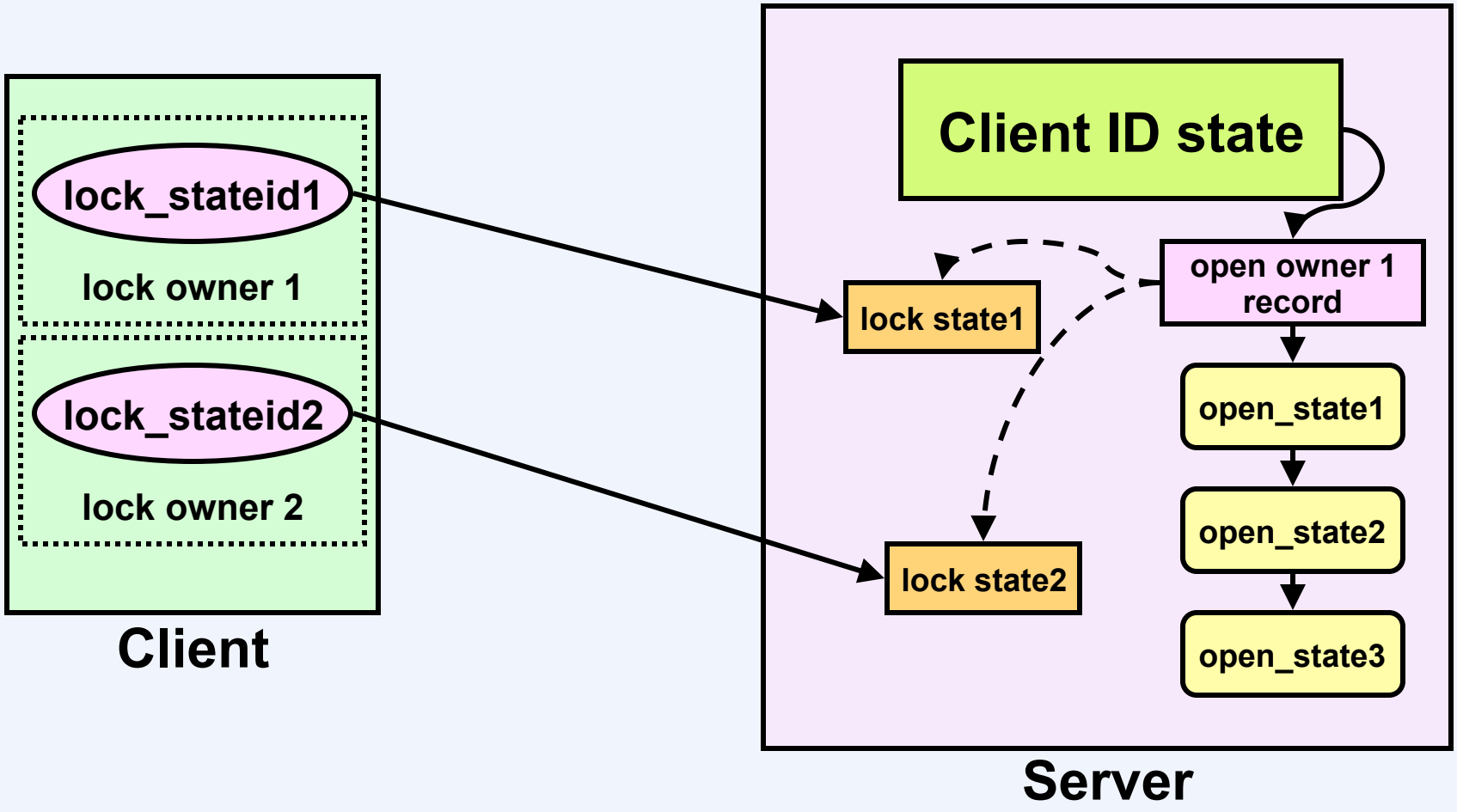
Mandatory Locks (1)

- Just like advisory locks, but you can't ignore them
  - require state on server
  - state is recovered after a server crash
- Nothing more to say …
  - (wrong …)

Mandatory Locks (2)

- Unix semantics
  - locks belong to process
    - not shared among processes
  - locks apply to *file*
    - doesn't matter which file descriptor is used
  - locks disappear on close of any file descriptor for file
  - to turn on mandatory locks:
    - turn on set-gid bit
    - turn off group-execute permission bit
- Windows semantics
  - locks belong to process
    - not shared among processes
  - locks apply to file descriptor (Windows file handle)
    - matters which one is used
  - locks disappear only when appropriate file handle is closed
  - all locks are mandatory

Lock State



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 CIFS)
  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 first acquisition of a share reservation or lock after a server reboot or client lease expiration
  - a flag indicating whether the client's most recent state was revoked because of a lease expiration

Additional Issues

- Authentication
  - poorly supported in NFSv3
  - extensible (and well supported) in NFSv4
- Authorization
  - Windows clients require ACLs
  - NFSv4 supports Windows-like ACLs
- Parallel I/O
  - pNFS