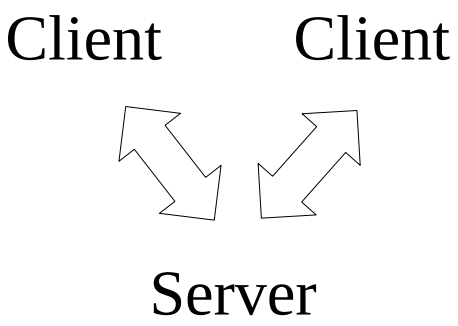
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

HY590.45
Modern Topics in Scalable Storage Systems

Kostas Magoutis
magoutis@csd.uoc.gr
http://www.csd.uoc.gr/~hy590-45

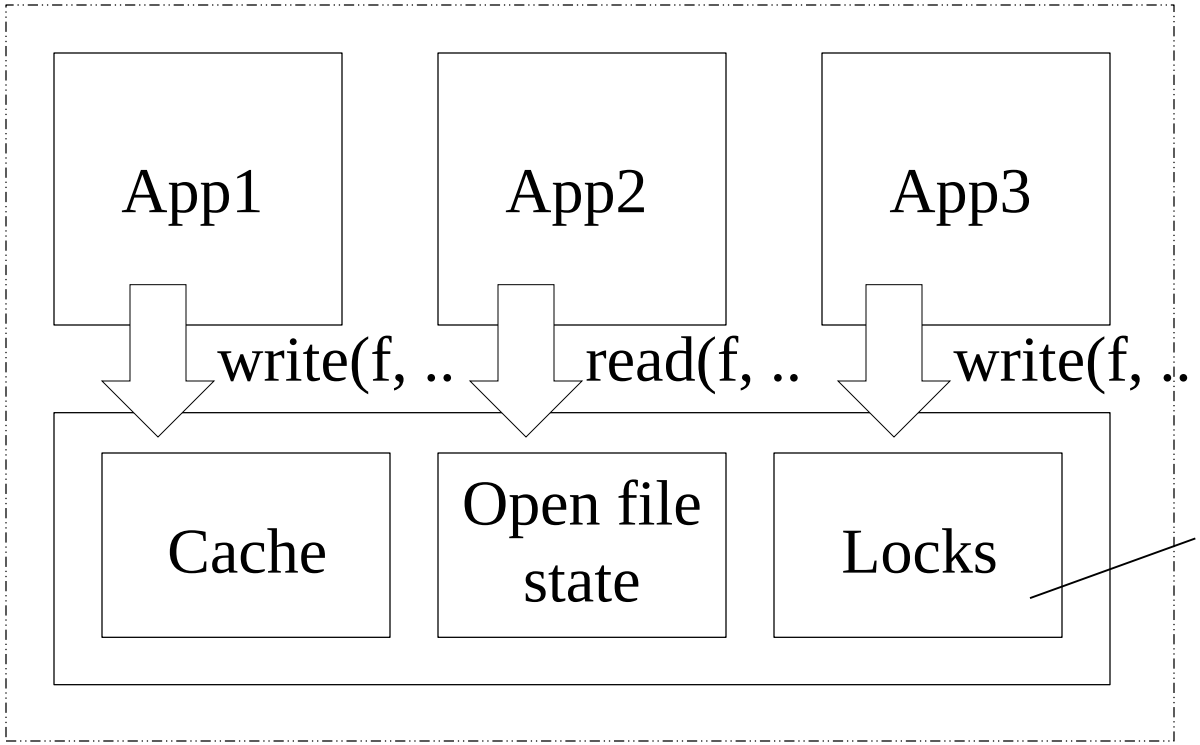
Distributed file sharing

- Benefits
  - Ability to access files from many locations
    - E.g., home directories
  - Consolidate storage management



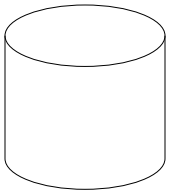
- Makes it possible to share files
  - Often concurrent readers or single writer
  - Less often, concurrent writers
    - Exclusive access to non-overlapping parts of file
    - Several data producers, concurrent append to shared file
    - Infrequent in engineering/office type workloads

File sharing in a single system



- File lock
- Range lock
  - Byte range
- Type of lock (op)
  - R/W

Single system



Data + metadata

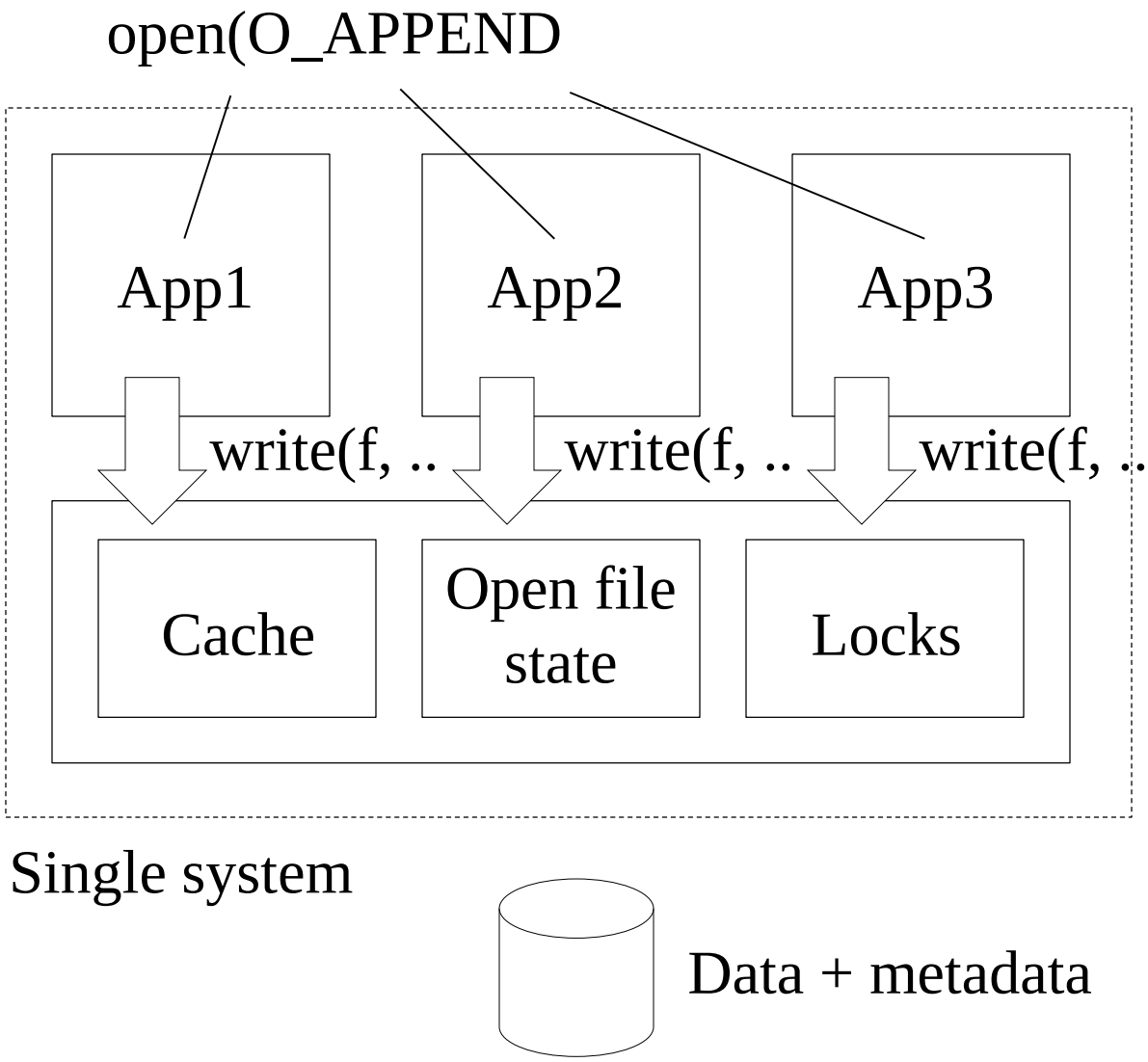
Crash recovery?

File-access APIs and semantics

Concurrent append, implicit serialization

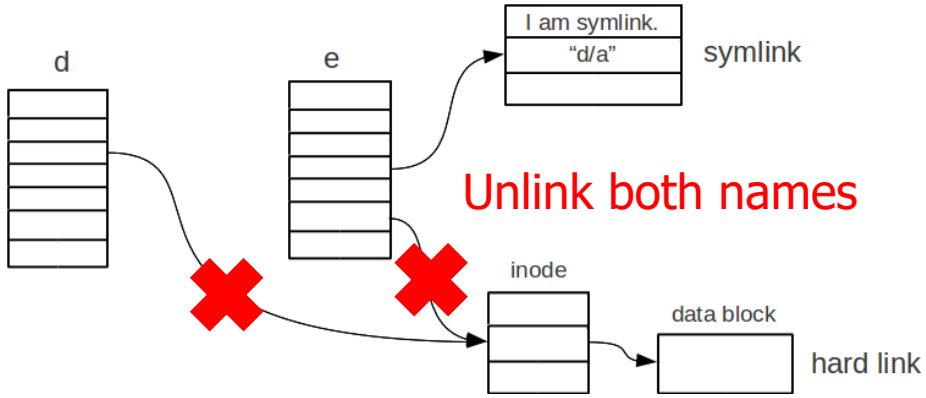
P1	P2
fd=open(f, O_APPEND	fd=open(f, O_APPEND
write(fd, …	write(fd, …
write(fd, …	write(fd, …
write(fd, …	write(fd, …
write(fd, …	write(fd, …
…	…
close(fd, …	close(fd, …

Concurrent append in a single system



File-access APIs and semantics

- Hard links: multiple names can be linked to an inode

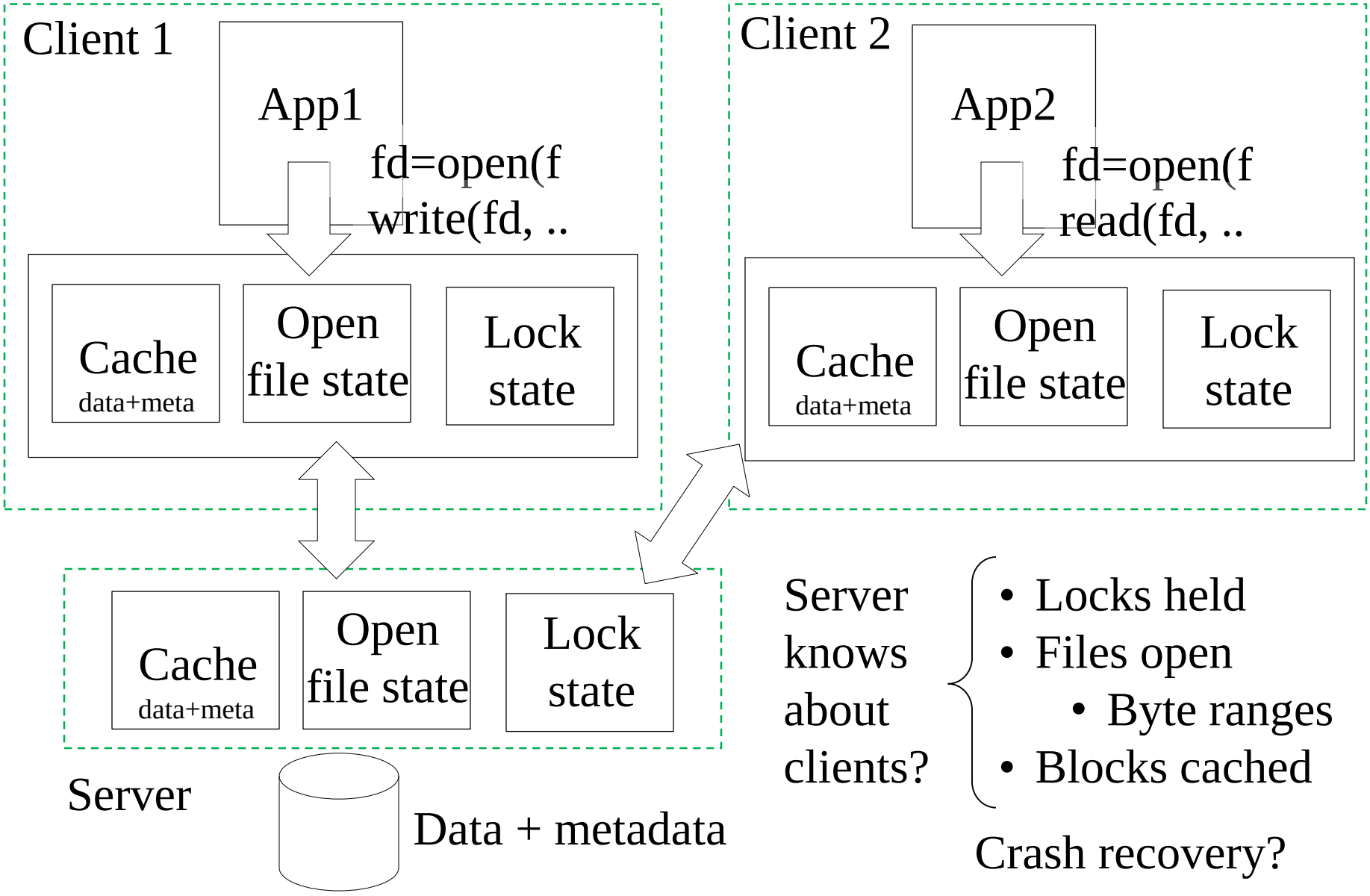


- In Unix, when files are unlinked they are not removed unless all open references to them are closed

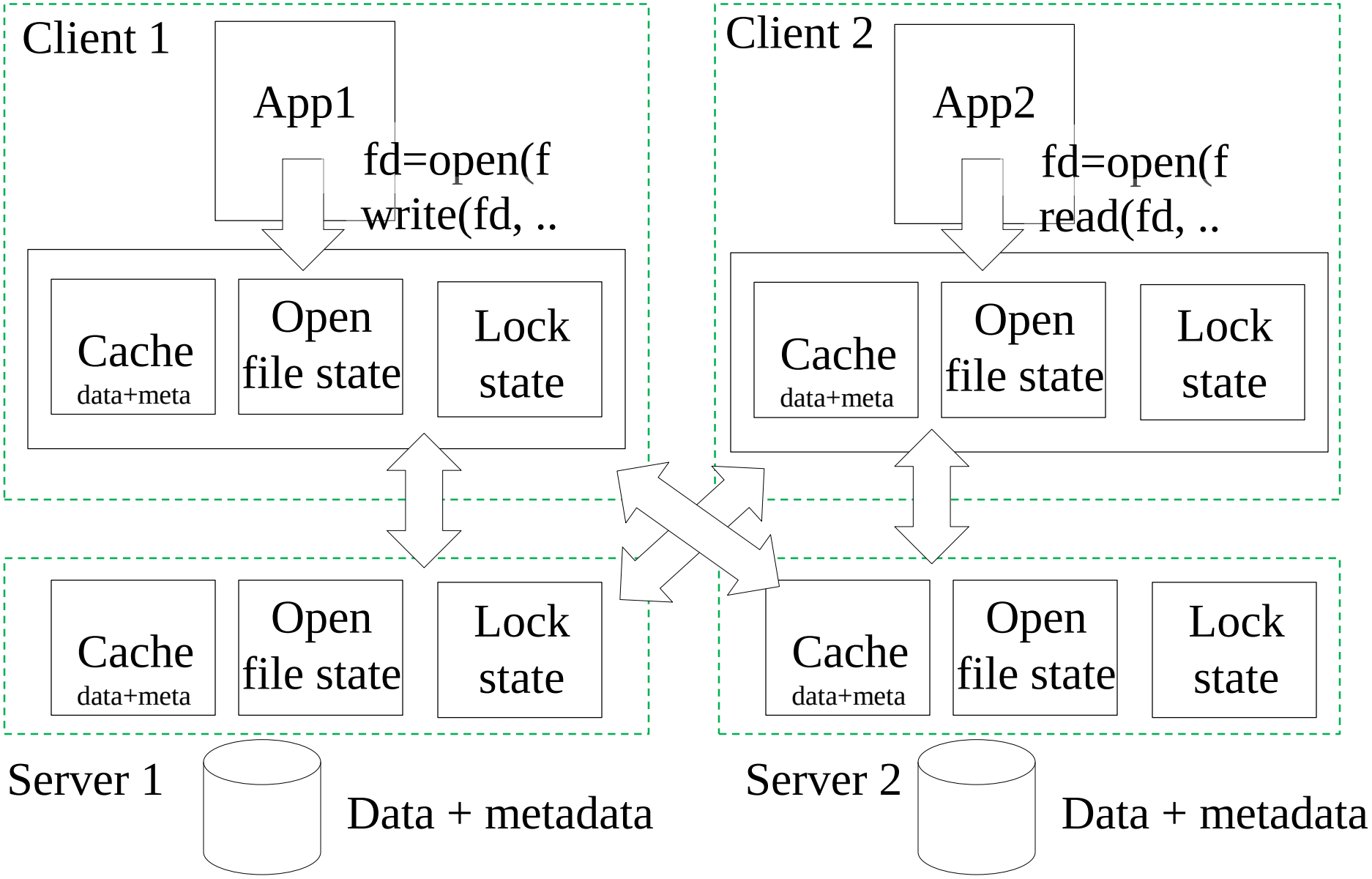
- File system semantics imply state

Image © https://medium.com/@meghamohan/hard-link-and-symbolic-link-3cad74e5b5dc

Extend to a distributed setting



Extend to a distributed setting

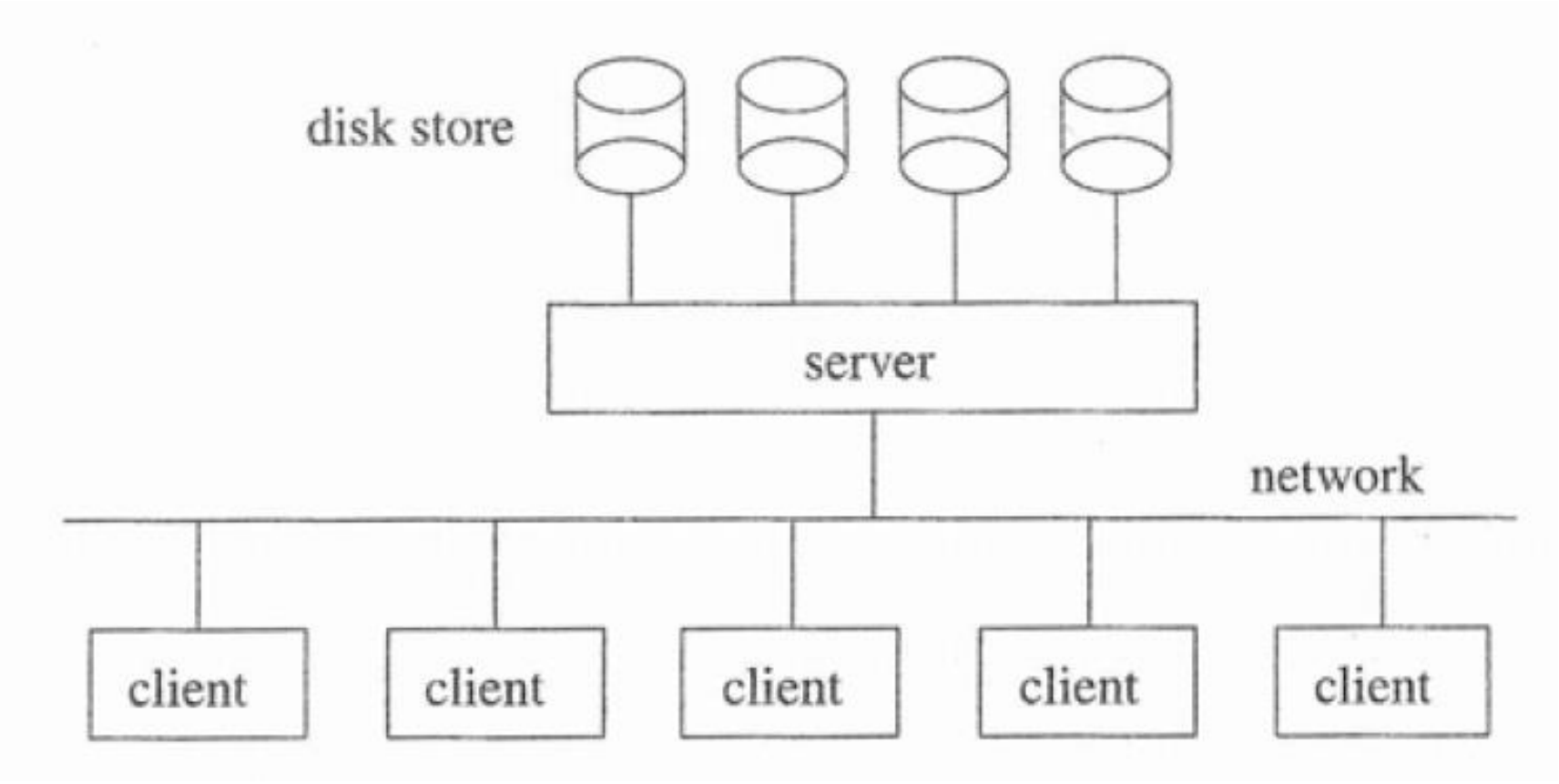


Network File System (NFS)

- History
  - UNIX United
  - SUN Network Disk
  - RFS
  - Andrew File System (AFS)

- Overview of NFS
  - Stateless
  - Aims to offer UNIX semantics
  - Transport independent
  - UNIX security and access control
  - Client caching and consistency

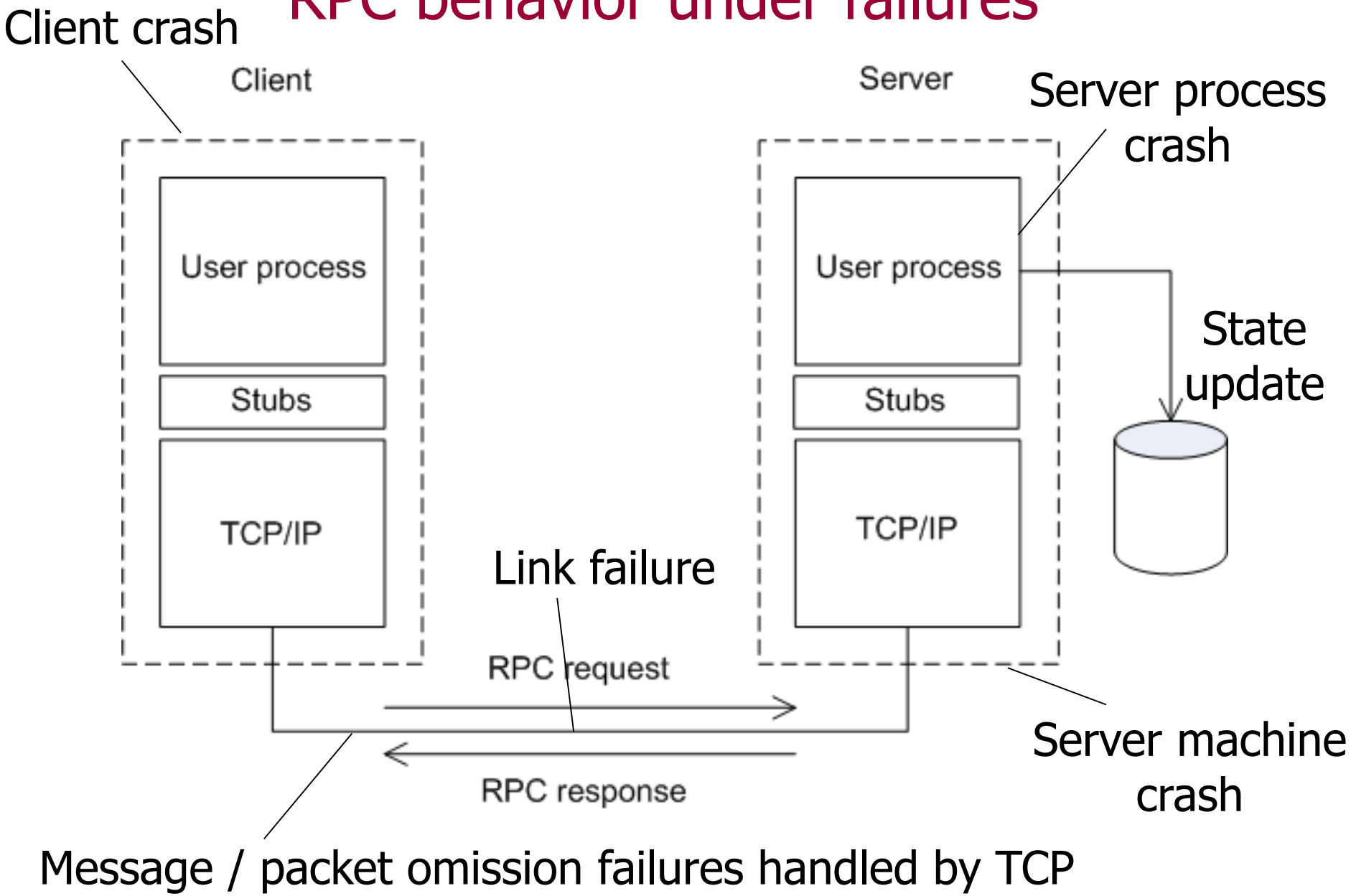
NFS division between clients and server



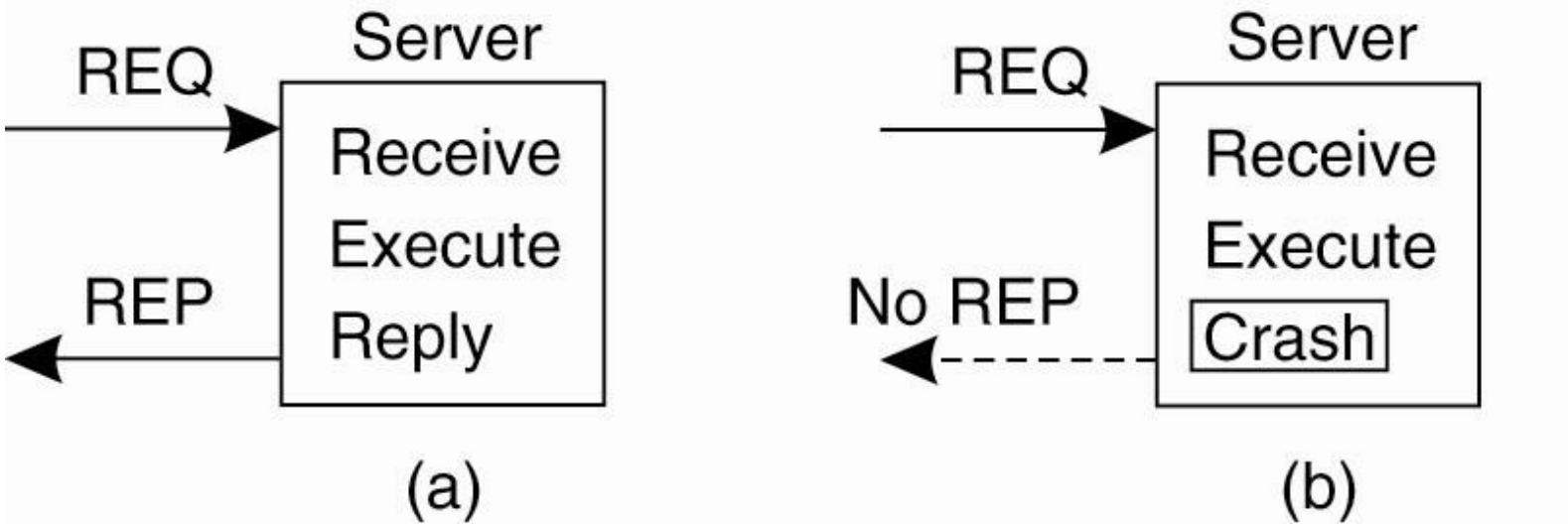
NFS structure and operation

- Based on Remote Procedure Calls (RPCs)
  - Handle problems that may occur due to crashes
- Files identified by NFS handle
  - Comprises inode id, file system id, generation number
- VFS/Vnode layer

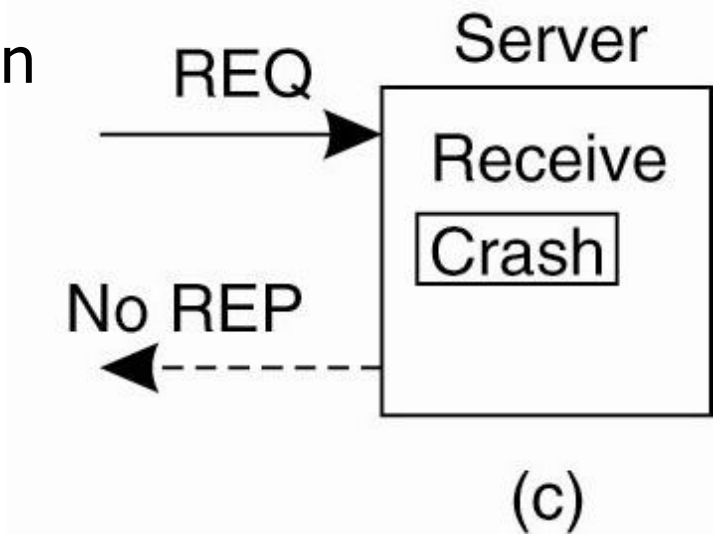
RPC behavior under failures



Server crashes



A server in client-server communication
(a) The normal case
(b) Crash after execution
(c) Crash before execution



RPC semantics

- At-least-once
  - Retry after an exception/timeout until successful
  - Good choice with idempotent operations (e.g., reads)
  - How about non-idempotent operations (e.g., writes)?

- At-most-once
  - Do not retry an operation or try to avoid duplicates

NFS Version 2 RPC requests

RPC request	Action	Idempotent
GETATTR	get file attributes	yes
SETATTR	set file attributes	yes
LOOKUP	look up file name	yes
READLINK	read from symbolic link	yes
READ	read from file	yes
WRITE	write to file	yes
CREATE	create file	yes
REMOVE	remove file	no
RENAME	rename file	no
LINK	create link to file	no
SYMLINK	create symbolic link	yes
MKDIR	create directory	no
RMDIR	remove directory	no
READDIR	read from directory	yes
STATFS	get filesystem attributes	yes

Stable-store requirement (NFS v2)

- All procedures in NFS v2 are synchronous
- When a procedure returns to the client, it assumes that the operation has completed and any data associated with the request is now on stable storage
- A WRITE may cause the server to update data blocks, indirect blocks, and attribute information (size, modify times)
- When the WRITE returns to the client, it can assume that the write is safe, even in case of a server crash

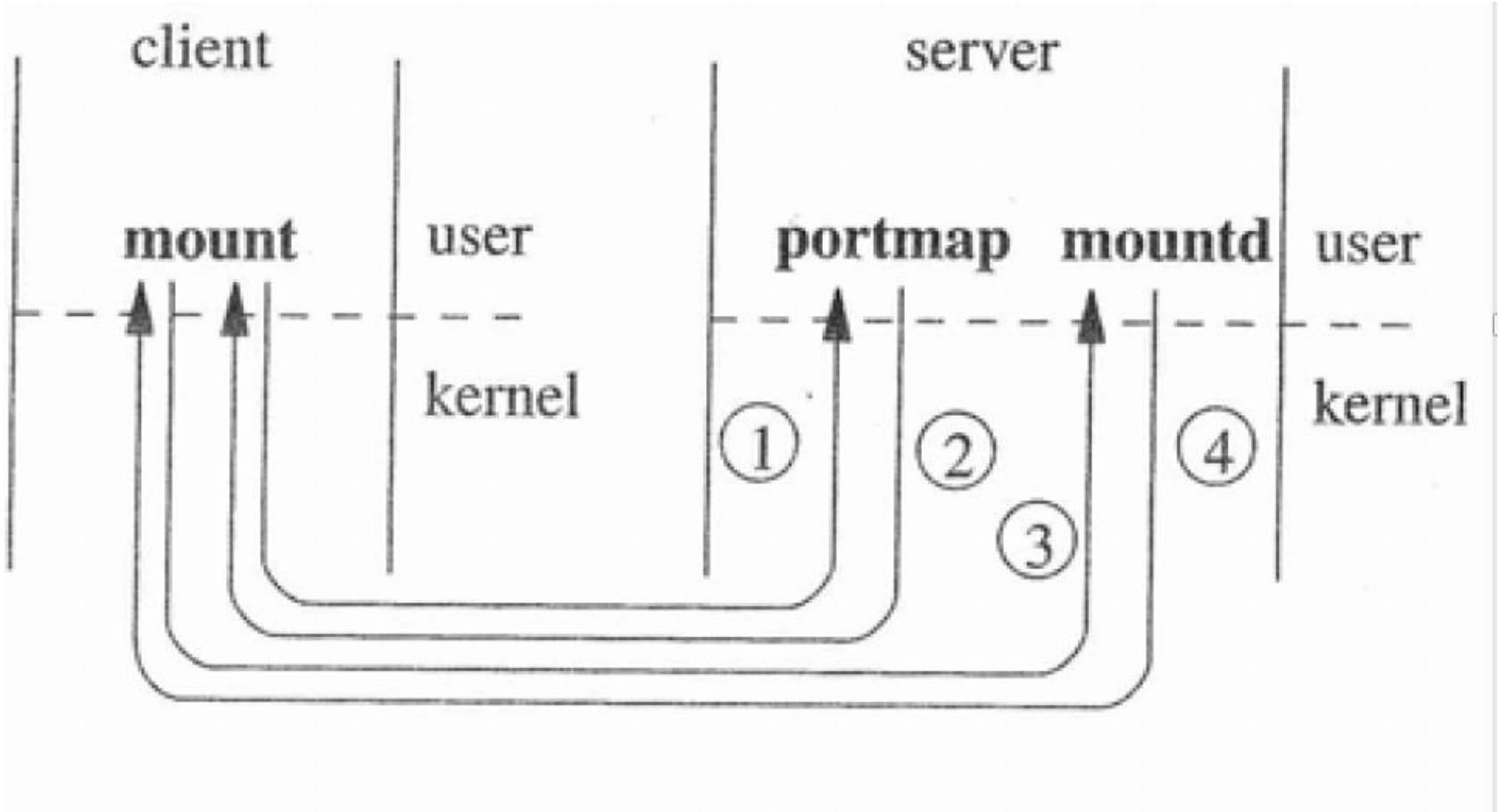
Statelessness

- Server has no information about clients, open files
  - Not entirely true in practice (e.g., retransmission cache)
- Pros
  - Recovery
- Cons
  - Local file system semantics imply state
  - Performance (due to stable store requirement)
    - Addressed by NFS specs following v2

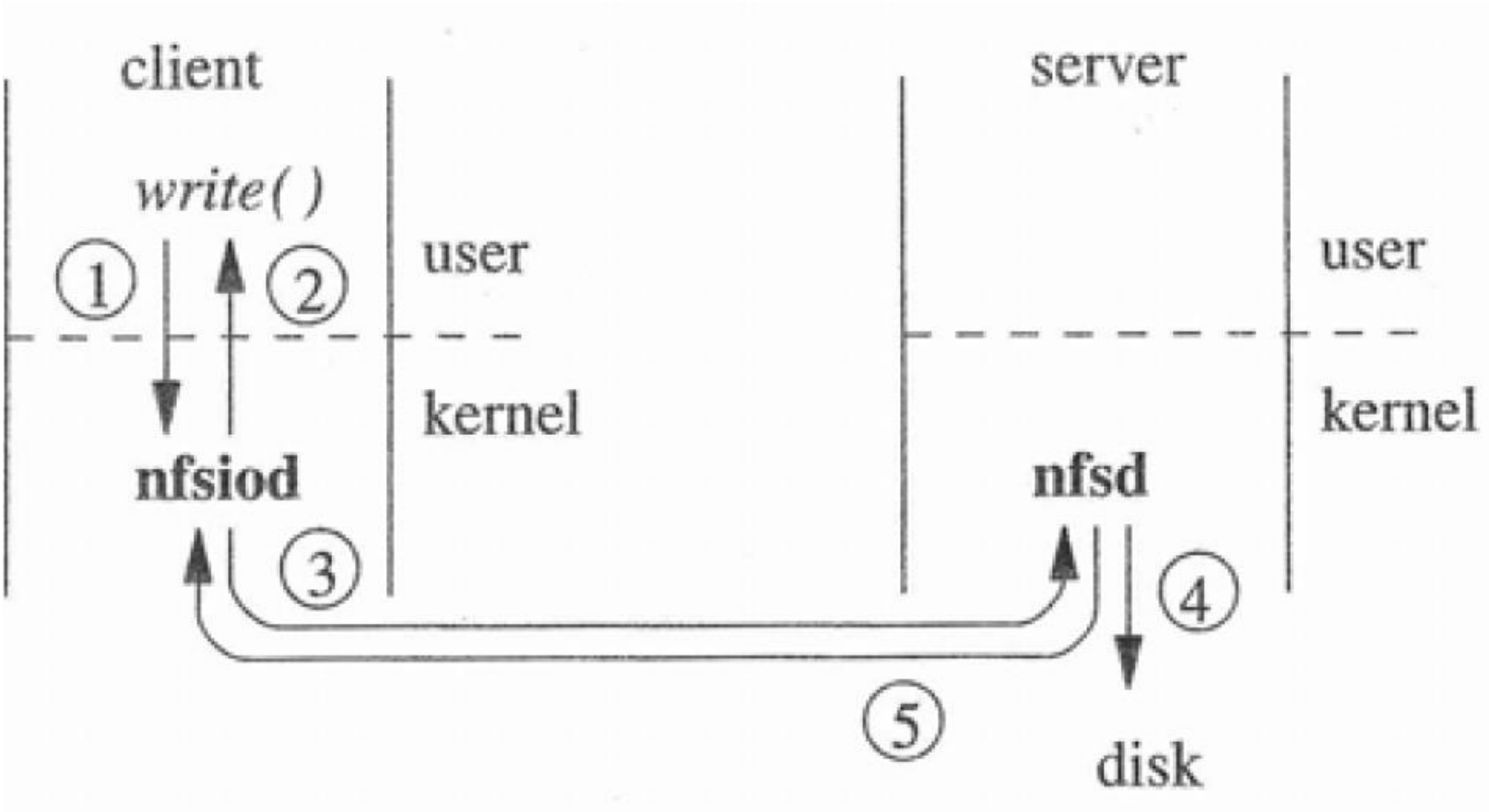
NFS Implementation

- Daemons
- Mounting a file system
- Performing I/O

Daemon interaction for mounting



Daemon interaction for I/O



Transport issues

- UDP implementation
  - RPC must fit in datagram (early versions restricted to 8KB)
  - Timeout, retransmission handled by RPC layer
  - Difficult to estimate RTT
  - RPC request broken down into IP fragments/Ethernet MTUs
- Or, run over TCP

Techniques for improving performance: Client caching and write buffering

- Delayed writes are allowed but cause problems
  - Other clients may see old versions of data
- Solution: close-to-open consistency
  - Clients flush on close(), so other clients will see the latest version on a later open()
- A write to server won't be reflected on clients' caches
  - No updates or invalidations (due to stateless server)
- Clients occasionally validate their caches
  - Send a GETATTR every 3 seconds and check the file's modification time and see if it has been updated

Existing solutions

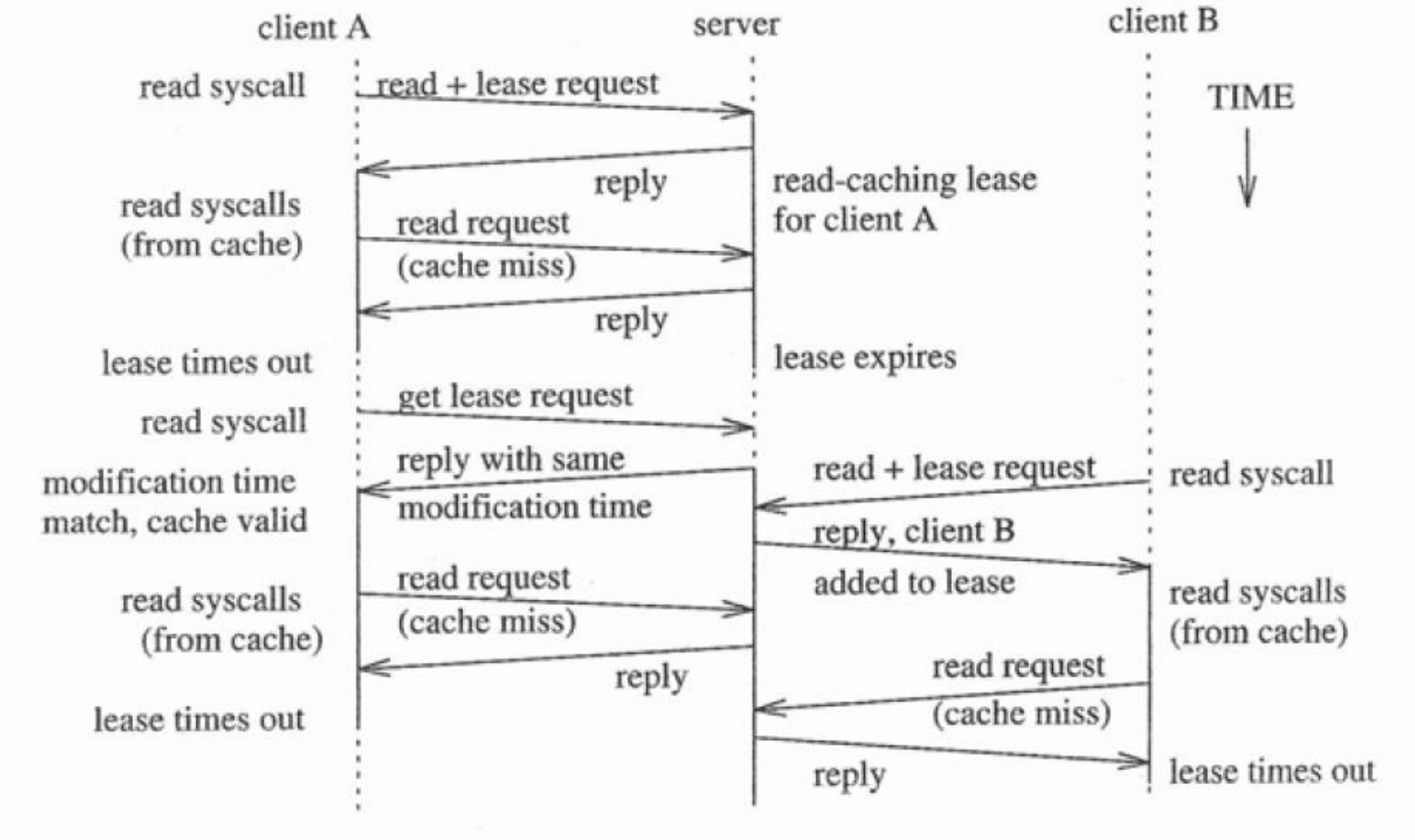
- NFS version 3
  - Leverage asynchronous writes
  - Writes back at 30’ intervals, flushes (commit) on close
  - Read consistency by checking with server on read (every 3”)
- Sprite [Nelson88]
  - File locks
  - Disable caching when detecting concurrent writes
- AFS [Howard88]
  - File locks
  - Callback-based invalidations

Leases: Fault-tolerant locks

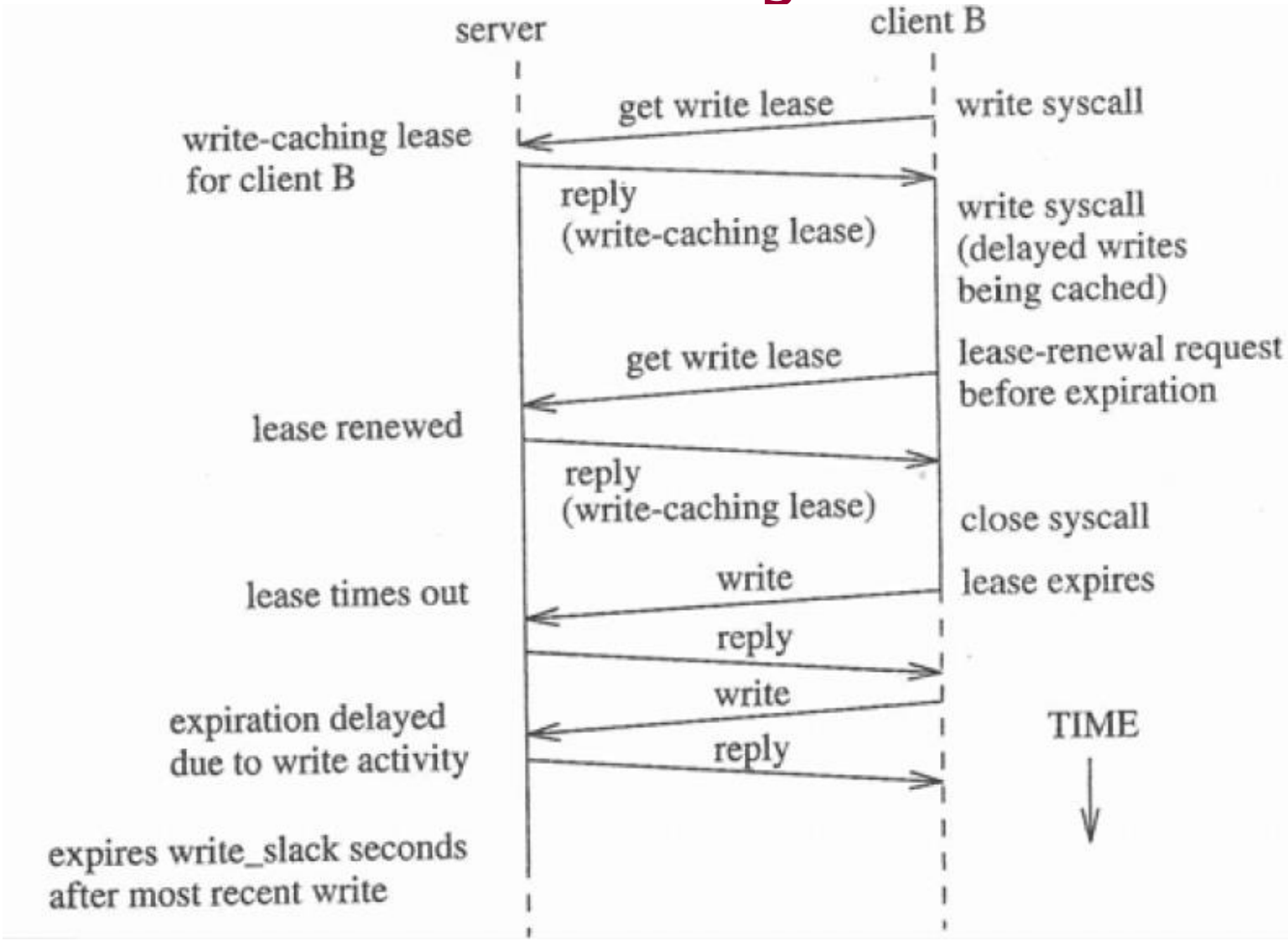
- Pros
  - Quick recovery without requiring hard state
  - Can tolerate partitions
  - Require moderate amount of soft state

- Important constants
  - `maximum_lease_term` (normally ~30")
  - `clock_skew`
  - `write_slack`

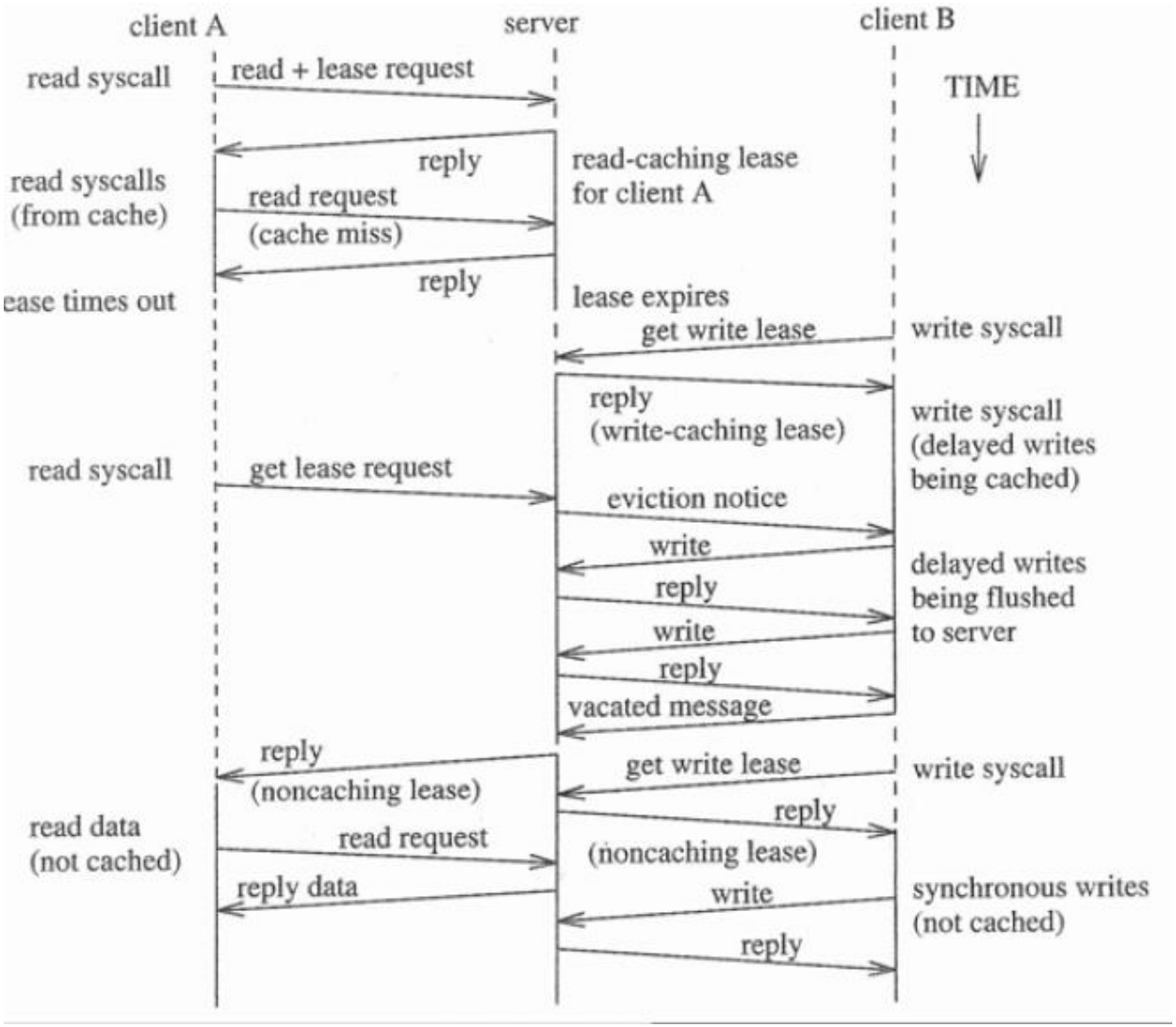
Read-caching leases



Write-caching lease



Write-sharing leases



Recovery

- `maximum_lease_term` seconds after the server stops issuing leases, there should be no leases left
- After rebooting, server accepts writes for up to `write_slack` seconds after final lease expired
- Server that do not know when they crashed estimate final-lease expiration time by adding up
  - `boot_time`
  - `maximum_lease_term`
  - `write_slack`
  - `clock_skew`