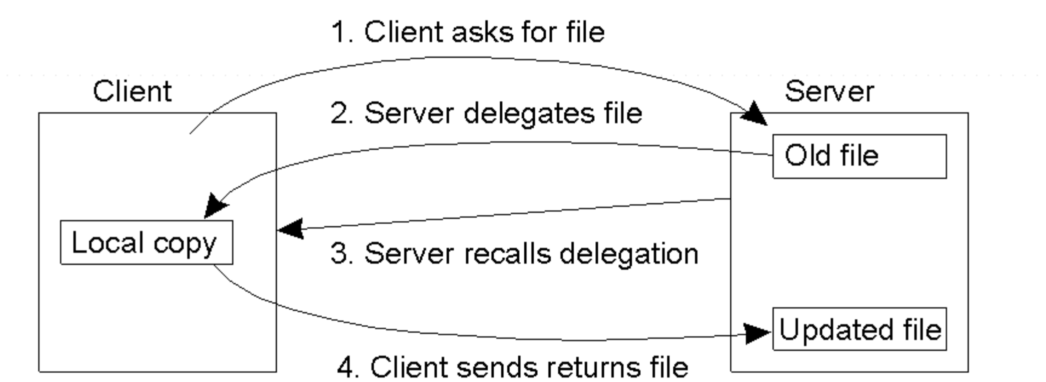
Today: Coda, xFS

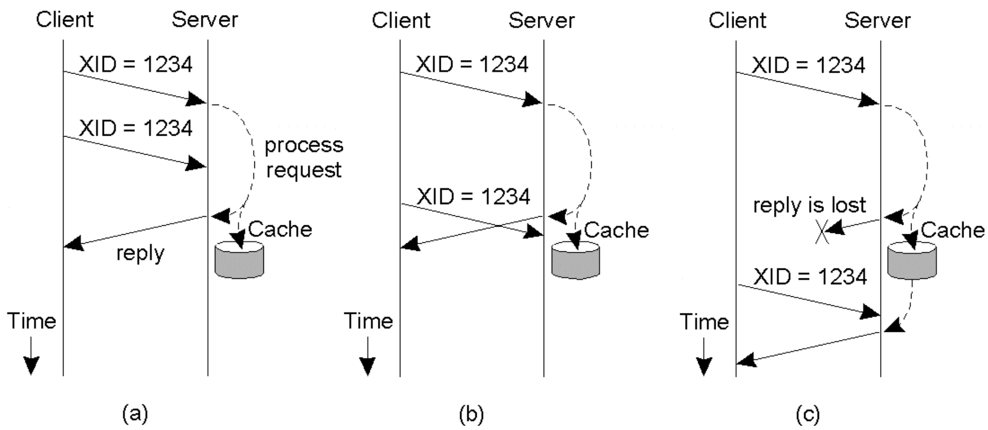
- Case study: NFS (continued)
- Case Study: Coda File System
- Brief overview of other recent file systems
  - xFS
  - Log structured file systems

Client Caching: Delegation



- NFS V4 supports open delegation
  - Server delegates local open and close requests to the NFS client
  - Uses a callback mechanism to recall file delegation.

RPC Failures



- Three situations for handling retransmissions: use a duplicate request cache

a) The request is still in progress

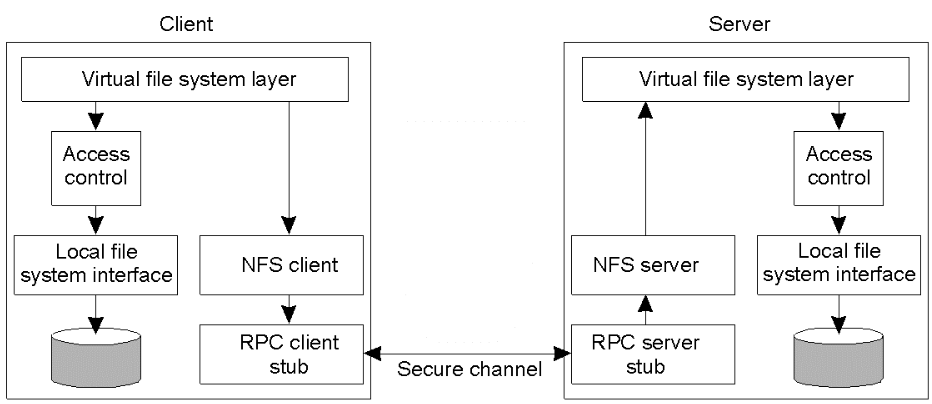
b) The reply has just been returned

c) The reply has been some time ago, but was lost.

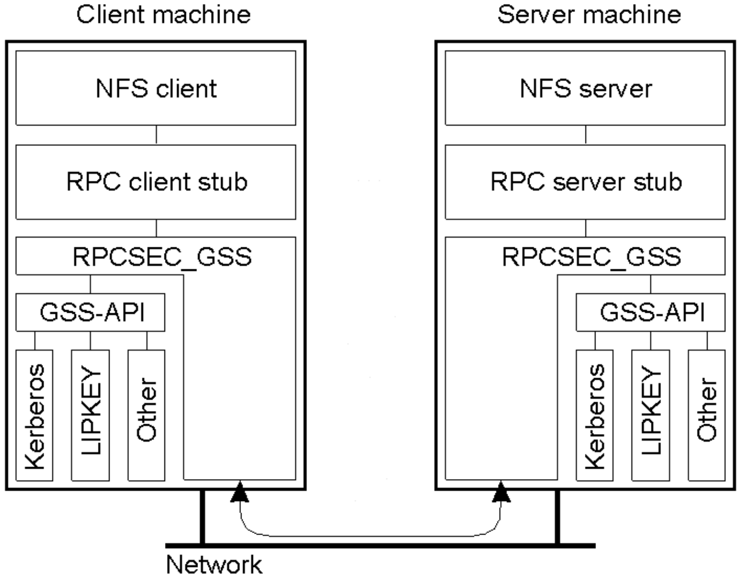
Use a duplicate-request cache: transaction Ids on RPCs, results cached

Security

- The NFS security architecture.
  - Simplest case: user ID, group ID authentication only



Secure RPCs

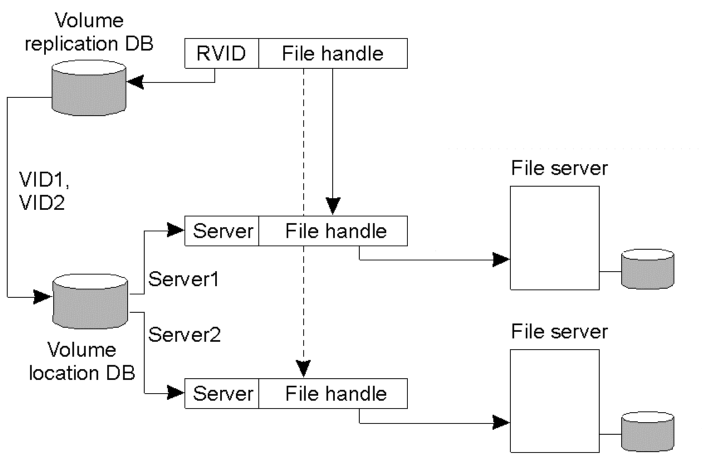


- Secure RPC in NFS version 4.

Replica Servers

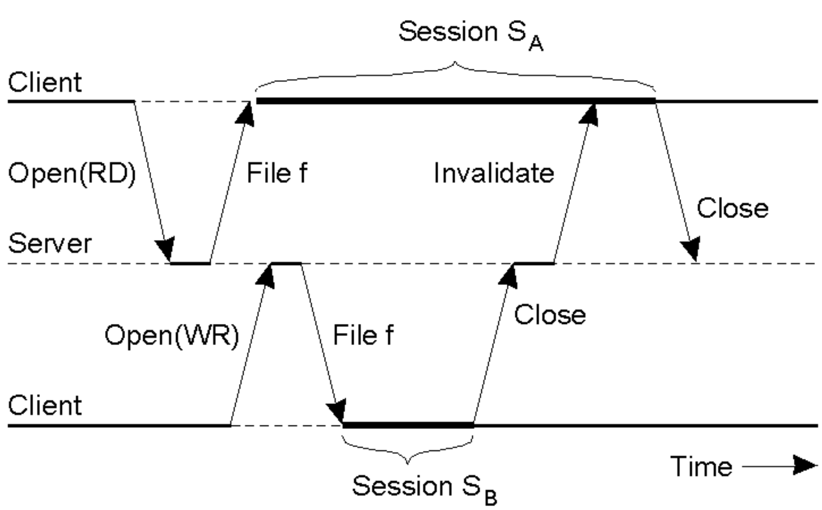
- NFS ver 4 supports replications
- Entire file systems must be replicated
- FS_LOCATION attribute for each file
- Replicated servers: implementation specific

CODA: File Identifiers



- Each file in Coda belongs to exactly one volume
  - Volume may be replicated across several servers
  - Multiple logical (replicated) volumes map to the same physical volume
  - 96 bit file identifier = 32 bit RVID + 64 bit file handle

Sharing Files in Coda



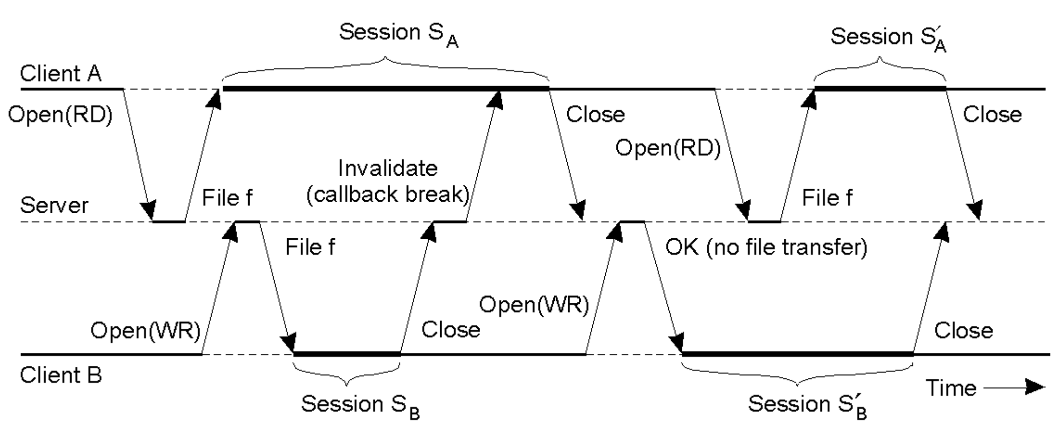
- Transactional behavior for sharing files: similar to share reservations in NFS
  - File open: transfer entire file to client machine [similar to delegation]
  - Uses session semantics: each session is like a transaction
    - Updates are sent back to the server only when the file is closed

Transactional Semantics

File-associated data	Read?	Modified?
File identifier	Yes	No
Access rights	Yes	No
Last modification time	Yes	Yes
File length	Yes	Yes
File contents	Yes	Yes

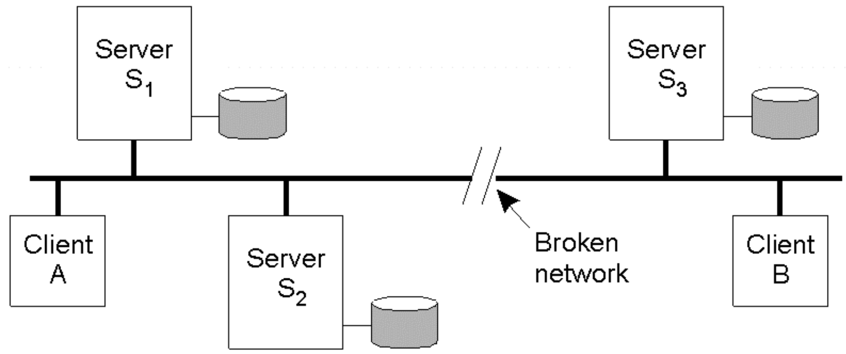
- Network partition: part of network isolated from rest
  - Allow conflicting operations on replicas across file partitions
  - Reconcile upon reconnection
  - Transactional semantics => operations must be serializable
    - Ensure that operations were serializable *after thay have executed*
  - Conflict => force manual reconciliation

Client Caching



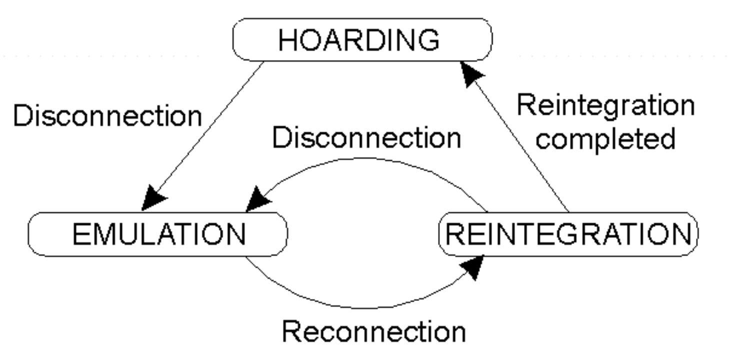
- Cache consistency maintained using callbacks
  - Server tracks all clients that have a copy of the file [provide *callback promise*]
  - Upon modification: send invalidate to clients

Server Replication



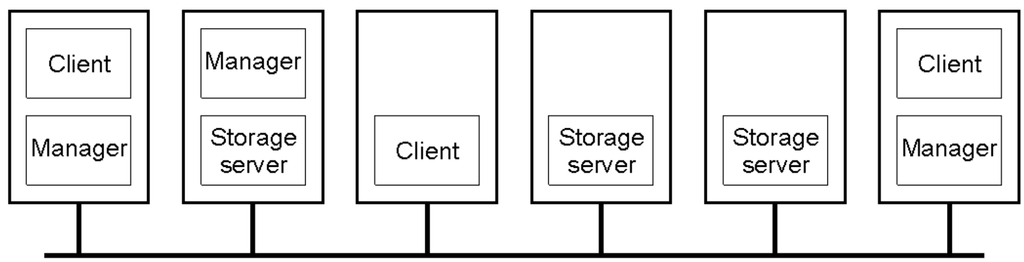
- Use replicated writes: read-once write-all
  - Writes are sent to all AVSG (all accessible replicas)
- How to handle network partitions?
  - Use optimistic strategy for replication
  - Detect conflicts using a Coda version vector
  - Example: [2,2,1] and [1,1,2] is a conflict => manual reconciliation

Disconnected Operation



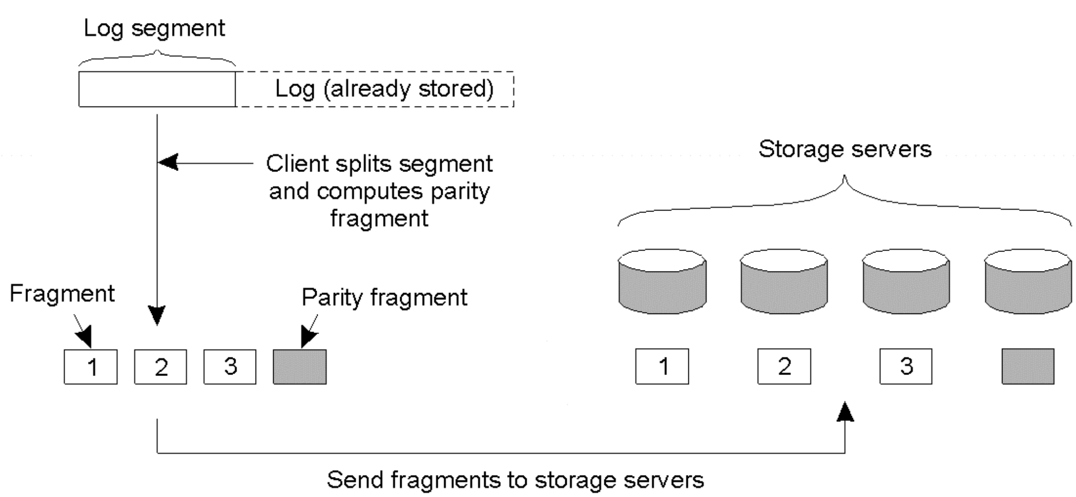
- The state-transition diagram of a Coda client with respect to a volume.
- Use hoarding to provide file access during disconnection
  - Prefetch all files that may be accessed and cache (hoard) locally
  - If AVSG=0, go to emulation mode and reintegrate upon reconnection

Overview of xFS.



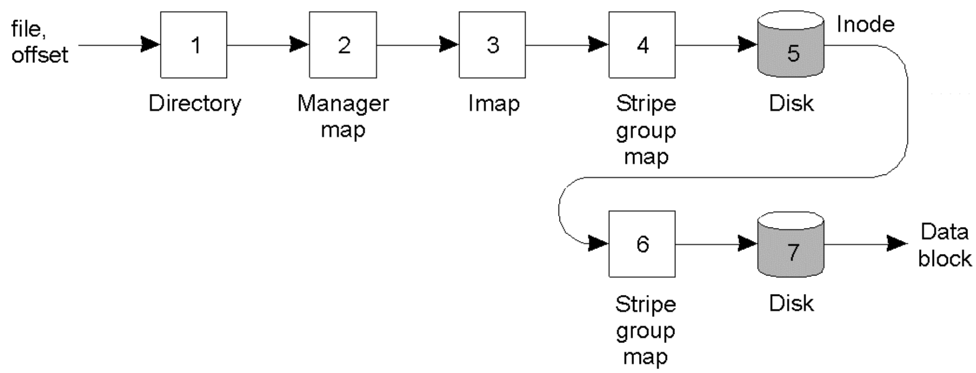
- Key Idea: fully distributed file system [*serverless* file system]
- xFS: x in "xFS" => no server
- Designed for high-speed LAN environments

Processes in xFS



- The principle of log-based striping in xFS
  – Combines striping and logging

Reading a File Block



- Reading a block of data in xFS.

xFS Naming

Data structure	Description
Manager map	Maps file ID to manager
Imap	Maps file ID to log address of file's inode
Inode	Maps block number (i.e., offset) to log address of block
File identifier	Reference used to index into manager map
File directory	Maps a file name to a file identifier
Log addresses	Triplet of stripe group, ID, segment ID, and segment offset
Stripe group map	Maps stripe group ID to list of storage servers

- Main data structures used in xFS.