

Today: Distributed File Systems

- Overview of stand-alone (UNIX) file systems
- Issues in distributed file systems
- Next two classes: case studies of distributed file systems
 - NFS
 - Coda
 - xFS
 - Log-structured file systems (time permitting)
 - HDFS; object storage systems



File System Basics

- File: named collection of logically related data
 - Unix file: an uninterpreted sequence of bytes
- File system:
 - Provides a logical view of data and storage functions
 - User-friendly interface
 - Provides facility to create, modify, organize, and delete files
 - Provides sharing among users in a controlled manner
 - Provides protection



Unix File System Review

- User file: linear array of bytes. No records, no file types
- Directory: special file not directly writable by user
- File structure: directed acyclic graph [directories may not be shared, files may be shared (*why?*)]
- Directory entry for each file
 - File name
 - inode number
 - Major device number
 - Minor device number
- All inodes are stored at a special location on disk [super block]
 - Inodes store file attributes and a multi-level index that has a list of disk block locations for the file



Inode Structure

- Fields
 - Mode
 - Owner_ID, group_id
 - Dir_file
 - Protection bits
 - Last access time, last write time, last inode time
 - Size, no of blocks
 - Ref_cnt
 - Address[0], ... address[14]
 - Multi-level index: 12 direct blocks, one single, double, and triple indirect blocks

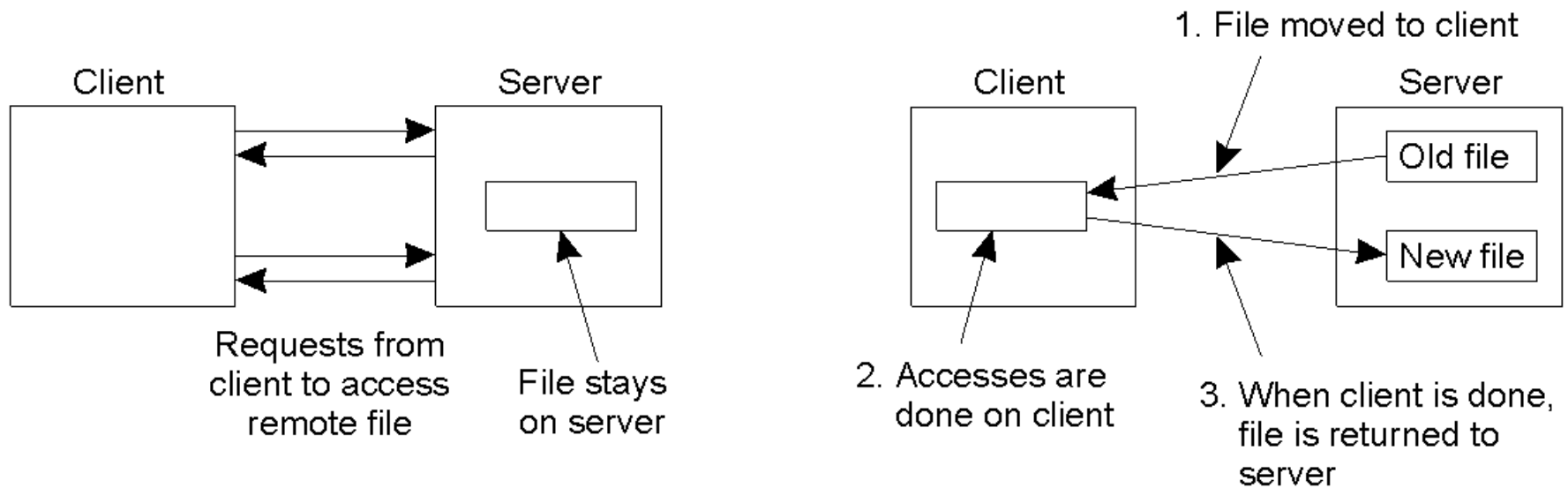


Distributed File Systems

- *File service*: specification of what the file system offers
 - Client primitives, application programming interface (API)
- *File server*: process that implements file service
 - Can have several servers on one machine (UNIX, DOS,...)
- Components of interest
 - File service
 - Directory service



File Service



- Remote access model
 - Work done at the server
- Stateful server (e.g., databases)
- Consistent sharing (+)
- Server may be a bottleneck (-)
- Need for communication (-)
- Upload/download mode
 - Work done at the client
- Stateless server
- Simple functionality (+)
- Moves files/blocks, need storage (-)



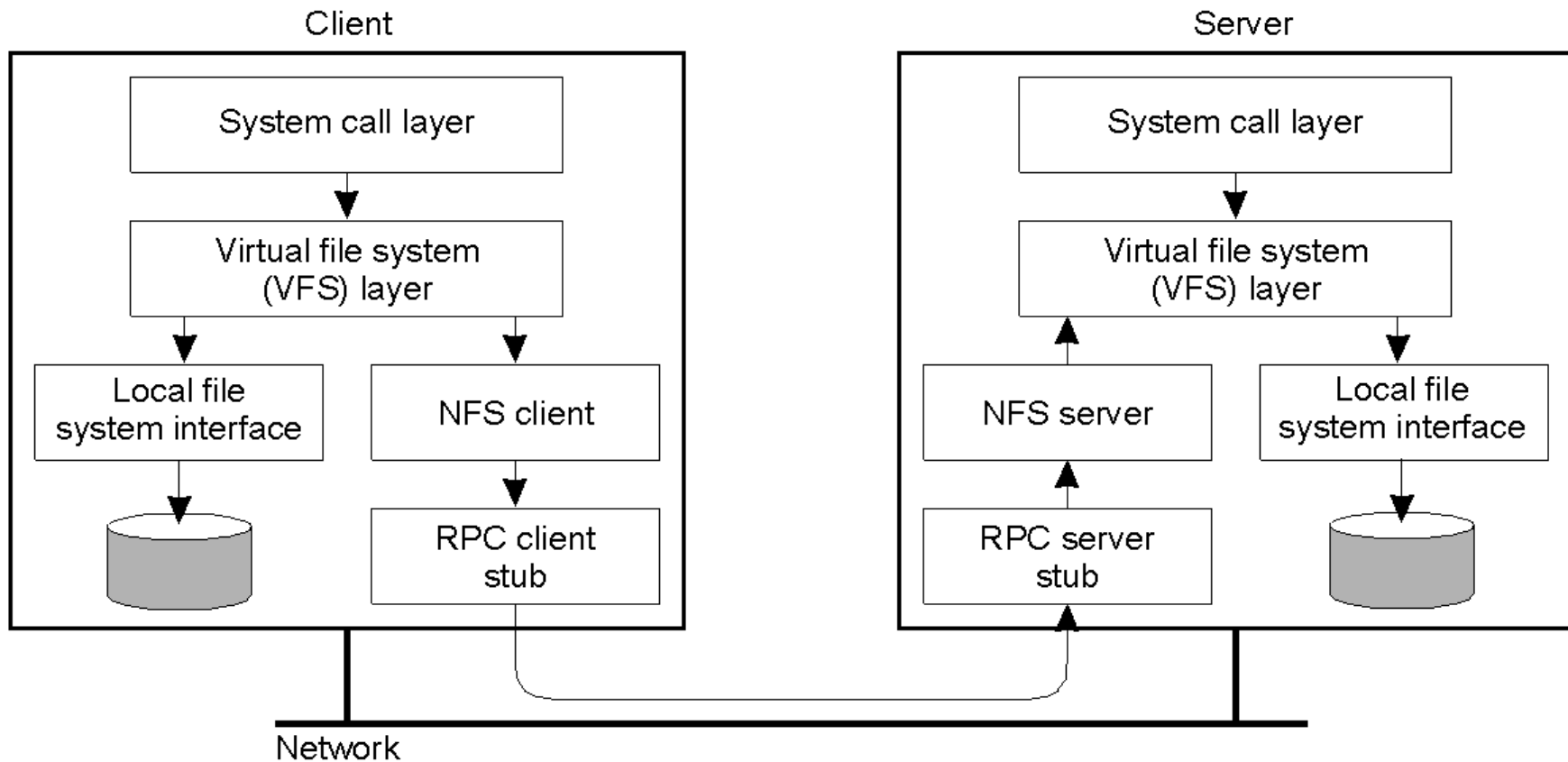
System Structure: Server Type

- Stateless server
 - No information is kept at server between client requests
 - All information needed to service a requests must be provided by the client with each request (*what info?*)
 - More tolerant to server crashes
- Stateful server
 - Server maintains information about client accesses
 - Shorted request messages
 - Better performance
 - Idempotency easier
 - Consistency is easier to achieve



NFS Architecture

- Sun's Network File System (NFS) – widely used distributed file system
- Uses the virtual file system layer to handle local and remote files

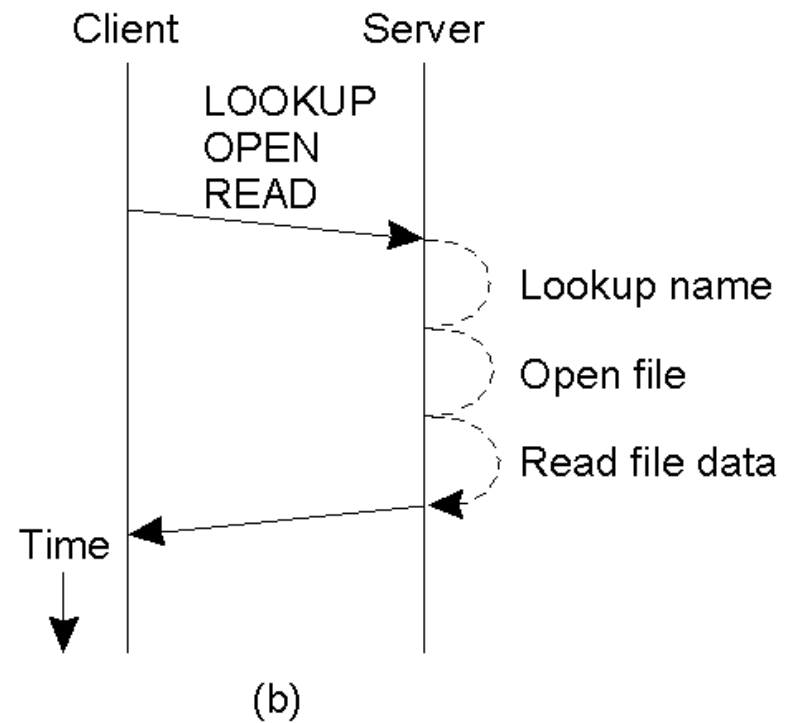
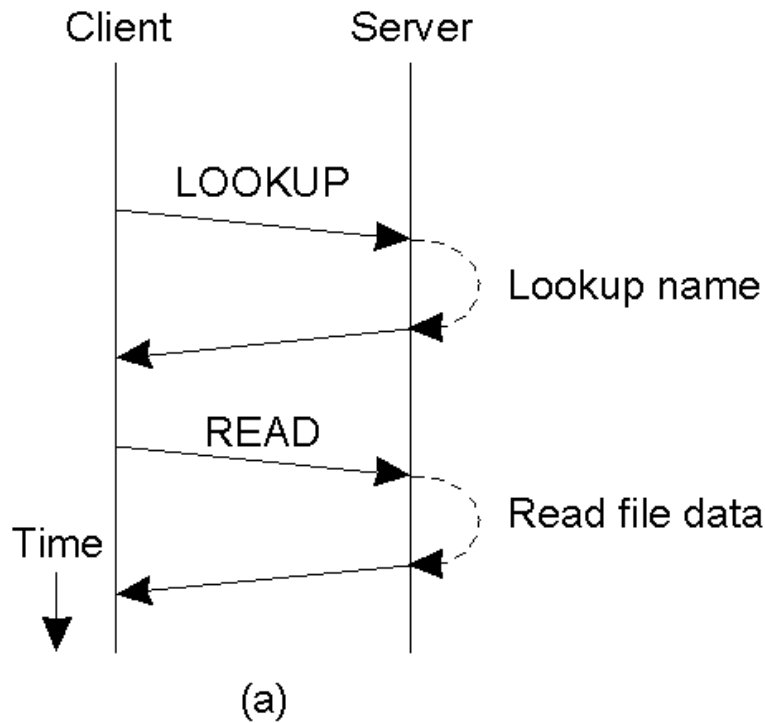


NFS Operations

| Operation | v3 | v4 | Description |
|-----------|-----|-----|---|
| Create | Yes | No | Create a regular file |
| Create | No | Yes | Create a nonregular file |
| Link | Yes | Yes | Create a hard link to a file |
| Symlink | Yes | No | Create a symbolic link to a file |
| Mkdir | Yes | No | Create a subdirectory in a given directory |
| Mknod | Yes | No | Create a special file |
| Rename | Yes | Yes | Change the name of a file |
| Rmdir | Yes | No | Remove an empty subdirectory from a directory |
| Open | No | Yes | Open a file |
| Close | No | Yes | Close a file |
| Lookup | Yes | Yes | Look up a file by means of a file name |
| Readdir | Yes | Yes | Read the entries in a directory |
| Readlink | Yes | Yes | Read the path name stored in a symbolic link |
| Getattr | Yes | Yes | Read the attribute values for a file |
| Setattr | Yes | Yes | Set one or more attribute values for a file |
| Read | Yes | Yes | Read the data contained in a file |
| Write | Yes | Yes | Write data to a file |



Communication



- a) Reading data from a file in NFS version 3.
- b) Reading data using a compound procedure in version 4.

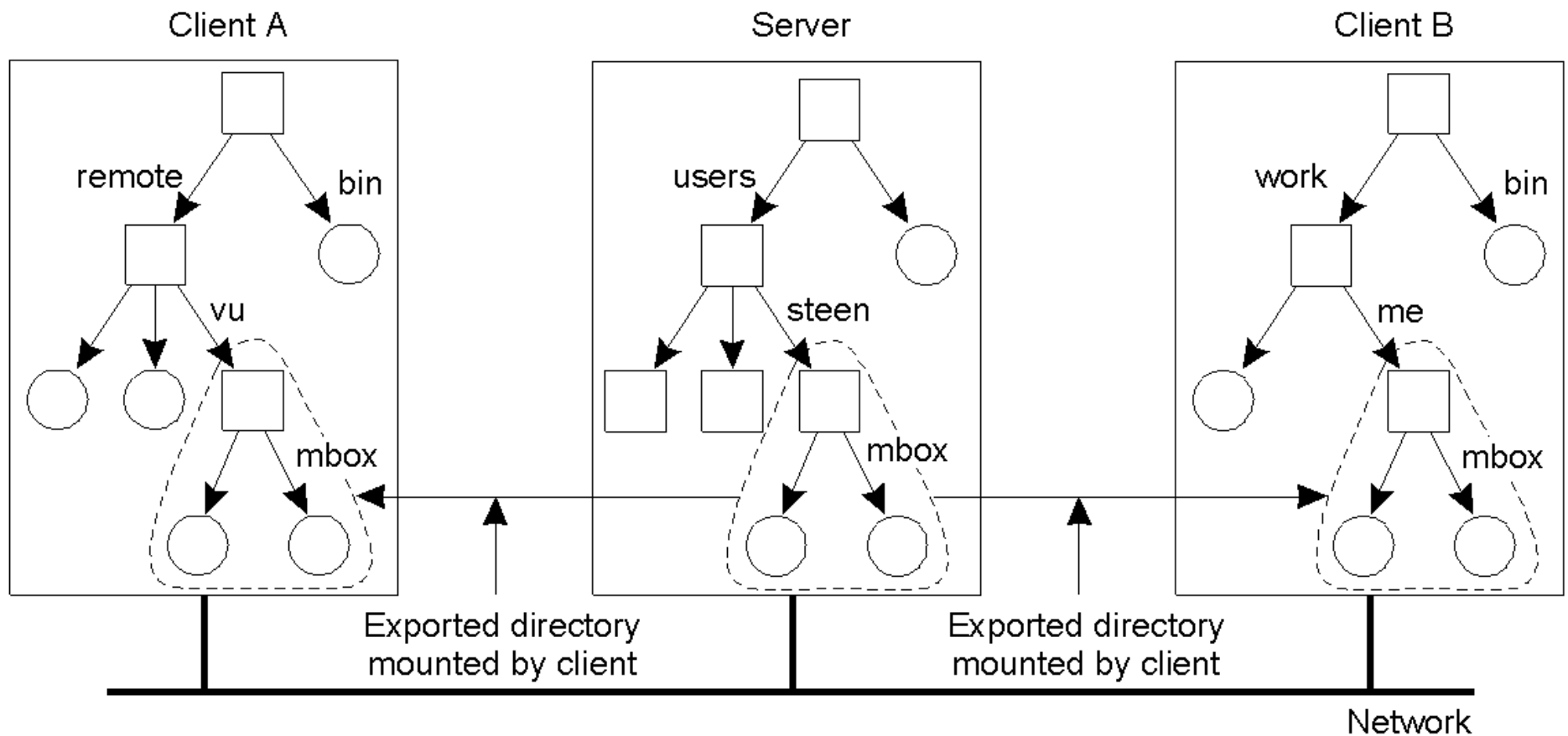
Both versions use Open Network Computing (ONC) RPCs

- One RPC per operation (NFS v3); multiple operations supported in v4.



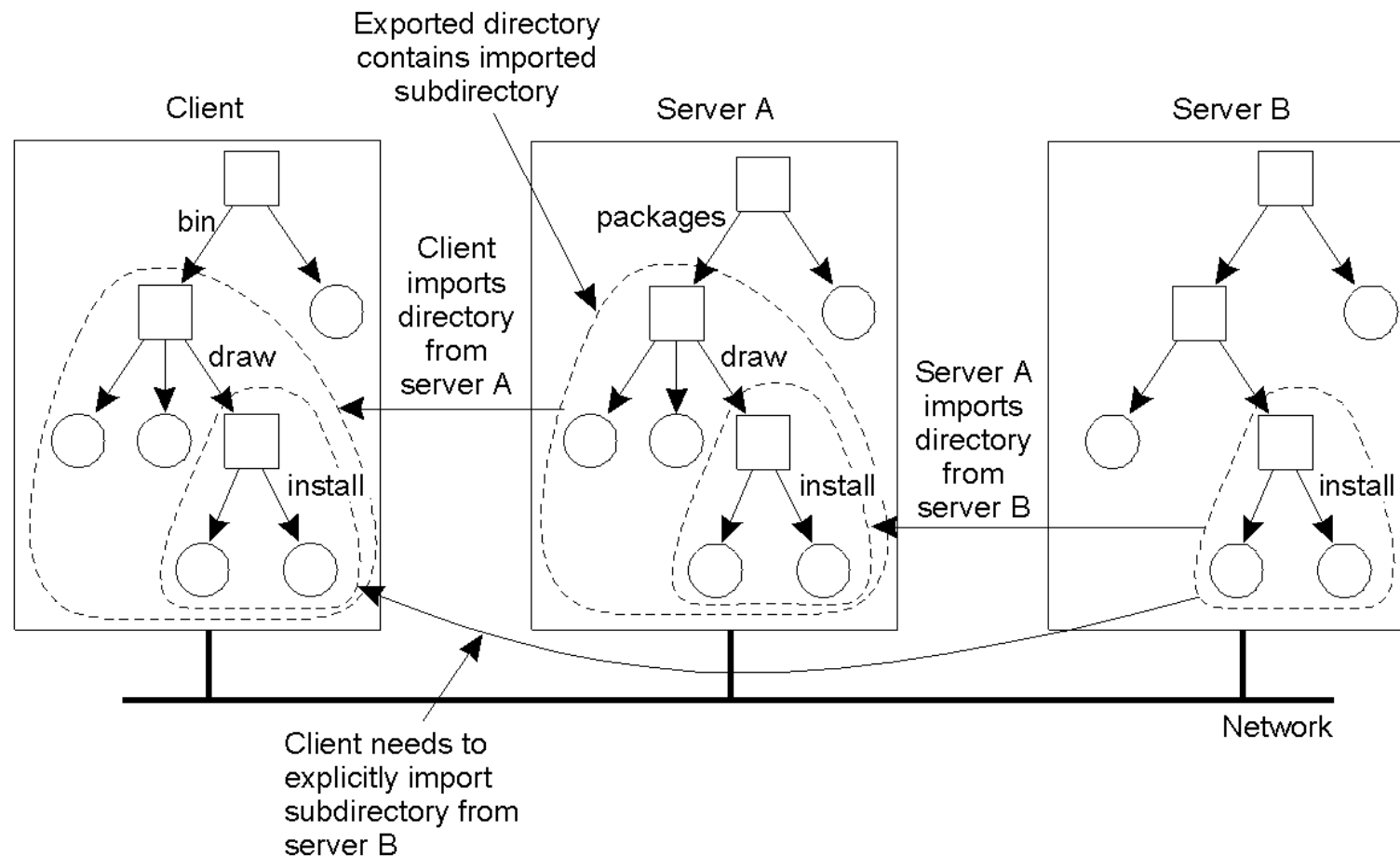
Naming: Mount Protocol

- NFS uses the mount protocol to access remote files
 - Mount protocol establishes a local name for remote files
 - Users access remote files using local names; OS takes care of the mapping

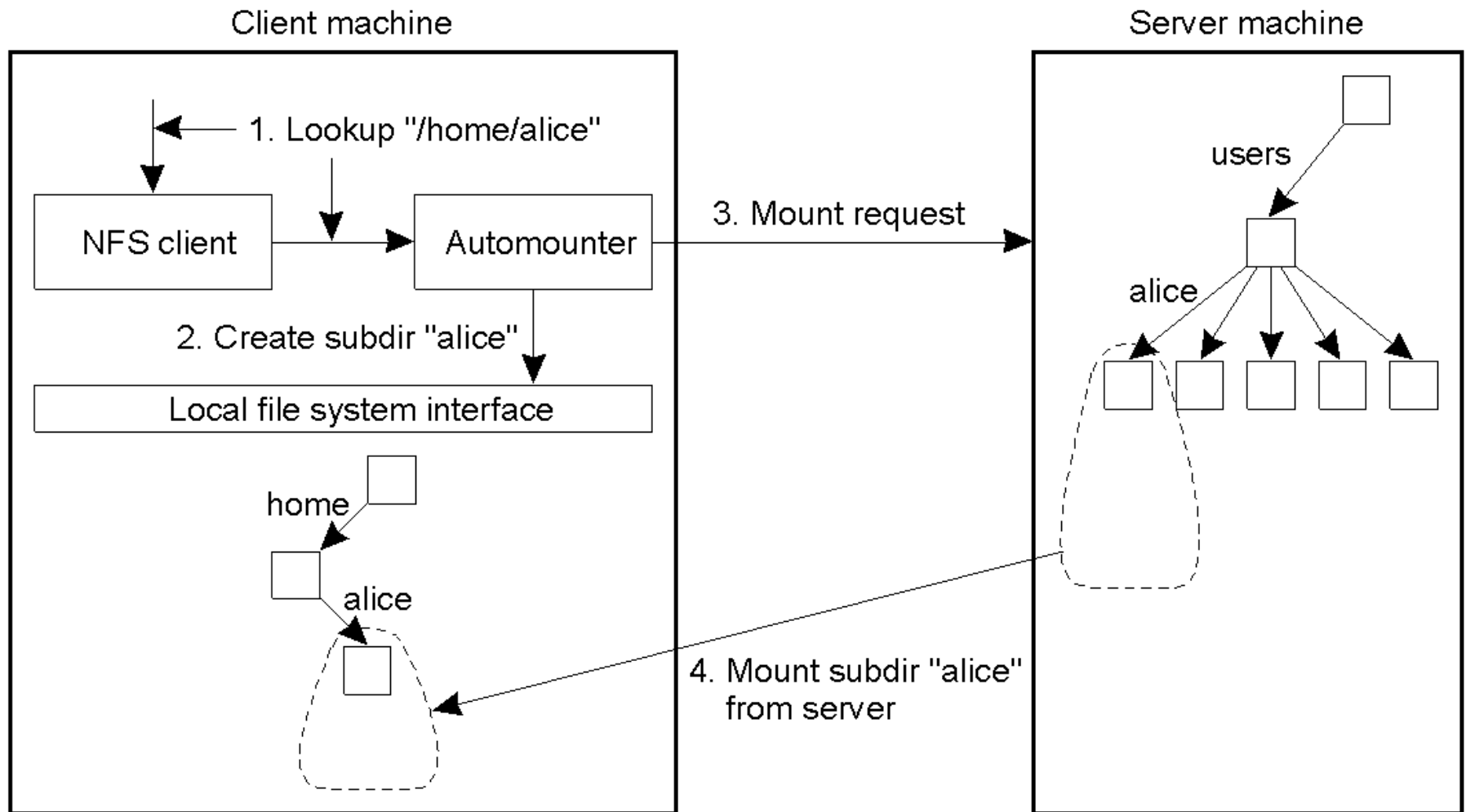


Naming: Crossing Mount Points

- Mounting nested directories from multiple servers
- NFS v3 does not support transitive exports (for security reasons)
 - NFS v4 allows clients to detect crossing of mount points, supports recursive lookups



Automounting



- Automounting: mount on demand



File Attributes (1)

| Attribute | Description |
|-----------|---|
| TYPE | The type of the file (regular, directory, symbolic link) |
| SIZE | The length of the file in bytes |
| CHANGE | Indicator for a client to see if and/or when the file has changed |
| FSID | Server-unique identifier of the file's file system |

- Some general mandatory file attributes in NFS.
 - NFS modeled based on Unix-like file systems
 - Implementing NFS on other file systems (Windows) difficult
 - NFS v4 enhances compatibility by using mandatory and recommended attributes



File Attributes (2)

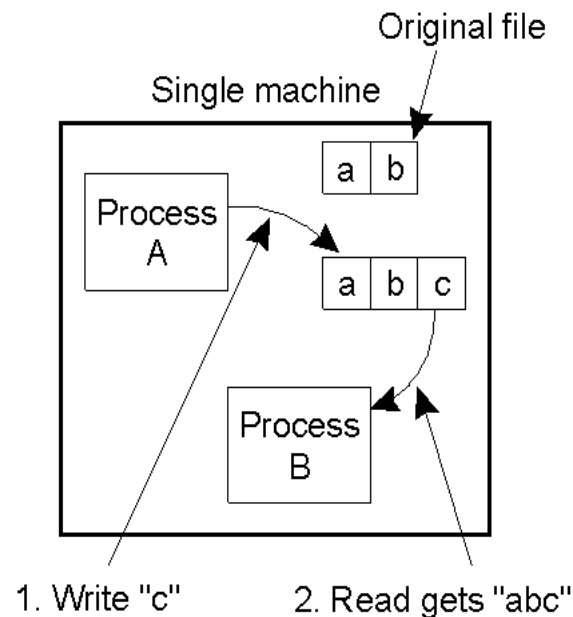
| Attribute | Description |
|--------------|--|
| ACL | an access control list associated with the file |
| FILEHANDLE | The server-provided file handle of this file |
| FILEID | A file-system unique identifier for this file |
| FS_LOCATIONS | Locations in the network where this file system may be found |
| OWNER | The character-string name of the file's owner |
| TIME_ACCESS | Time when the file data were last accessed |
| TIME_MODIFY | Time when the file data were last modified |
| TIME_CREATE | Time when the file was created |

- Some general recommended file attributes.

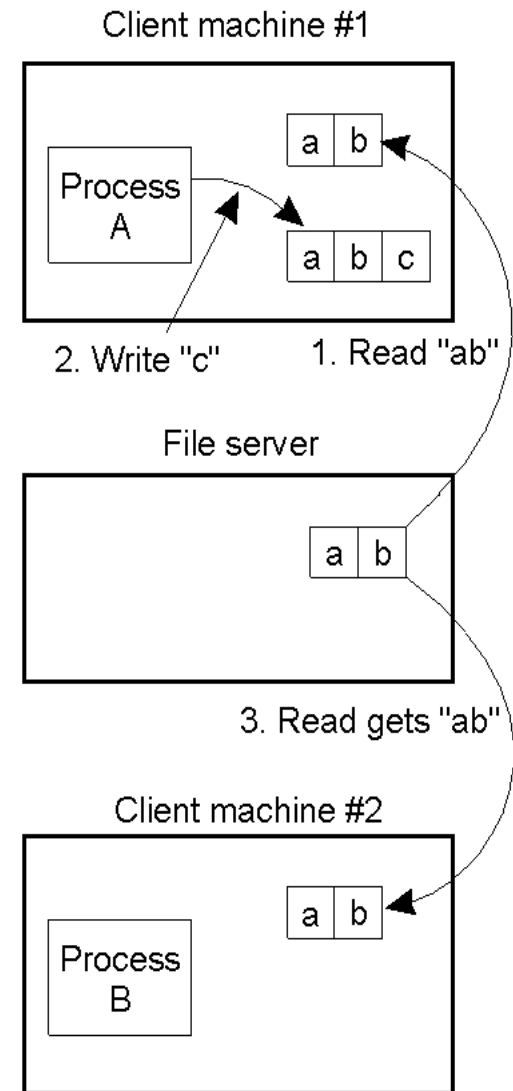


Semantics of File Sharing

- a) On a single processor, when a *read* follows a *write*, the value returned by the *read* is the value just written.
- b) In a distributed system with caching, obsolete values may be returned.



(a)



(b)



Semantics of File Sharing

| Method | Comment |
|-------------------|--|
| UNIX semantics | Every operation on a file is instantly visible to all processes |
| Session semantics | No changes are visible to other processes until the file is closed |
| Immutable files | No updates are possible; simplifies sharing and replication |
| Transaction | All changes occur atomically |

- Four ways of dealing with the shared files in a distributed system.
 - NFS implements session semantics
 - Can use remote/access model for providing UNIX semantics (expensive)
 - Most implementations use local caches for performance and provide session semantics



File Locking in NFS

| Operation | Description |
|-----------|--|
| Lock | Creates a lock for a range of bytes (non-blocking_ |
| Lockt | Test whether a conflicting lock has been granted |
| Locku | Remove a lock from a range of bytes |
| Renew | Renew the lease on a specified lock |

- NFS supports file locking
 - Applications can use locks to ensure consistency
 - Locking was not part of NFS until version 3
 - NFS v4 supports locking as part of the protocol (see above table)



File Locking: Share Reservations

| Request access | Current file denial state | | | | |
|----------------|---------------------------|---------|---------|---------|------|
| | | NONE | READ | WRITE | BOTH |
| | READ | Succeed | Fail | Succeed | Fail |
| | WRITE | Succeed | Succeed | Fail | Fail |
| | BOTH | Succeed | Fail | Fail | Fail |

(a)

| Current access state | Requested file denial state | | | | |
|----------------------|-----------------------------|---------|---------|---------|------|
| | | NONE | READ | WRITE | BOTH |
| | READ | Succeed | Fail | Succeed | Fail |
| | WRITE | Succeed | Succeed | Fail | Fail |
| | BOTH | Succeed | Fail | Fail | Fail |

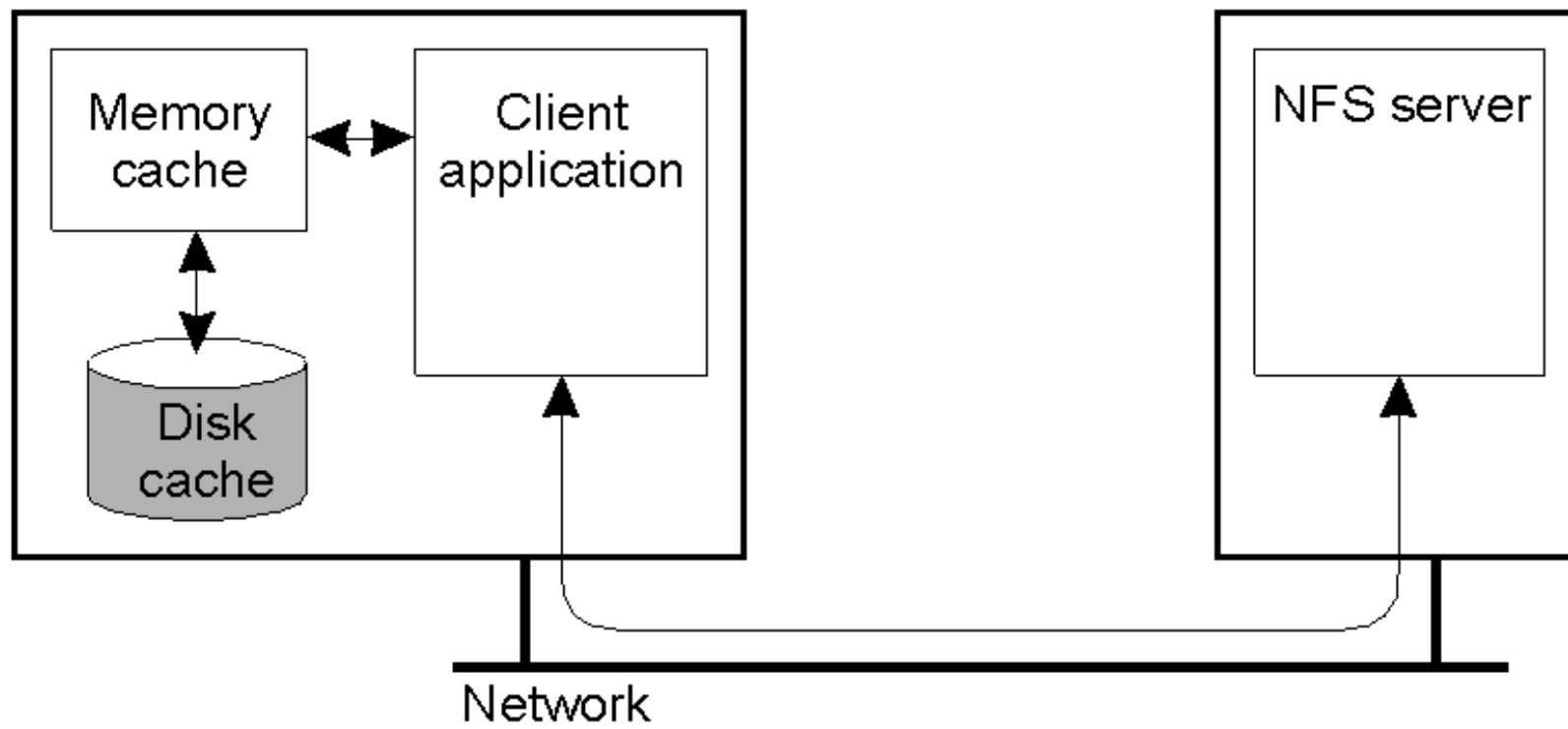
(b)

- The result of an *open* operation with share reservations in NFS.
- a) When the client requests shared access given the current denial state.
- b) When the client requests a denial state given the current file access state.



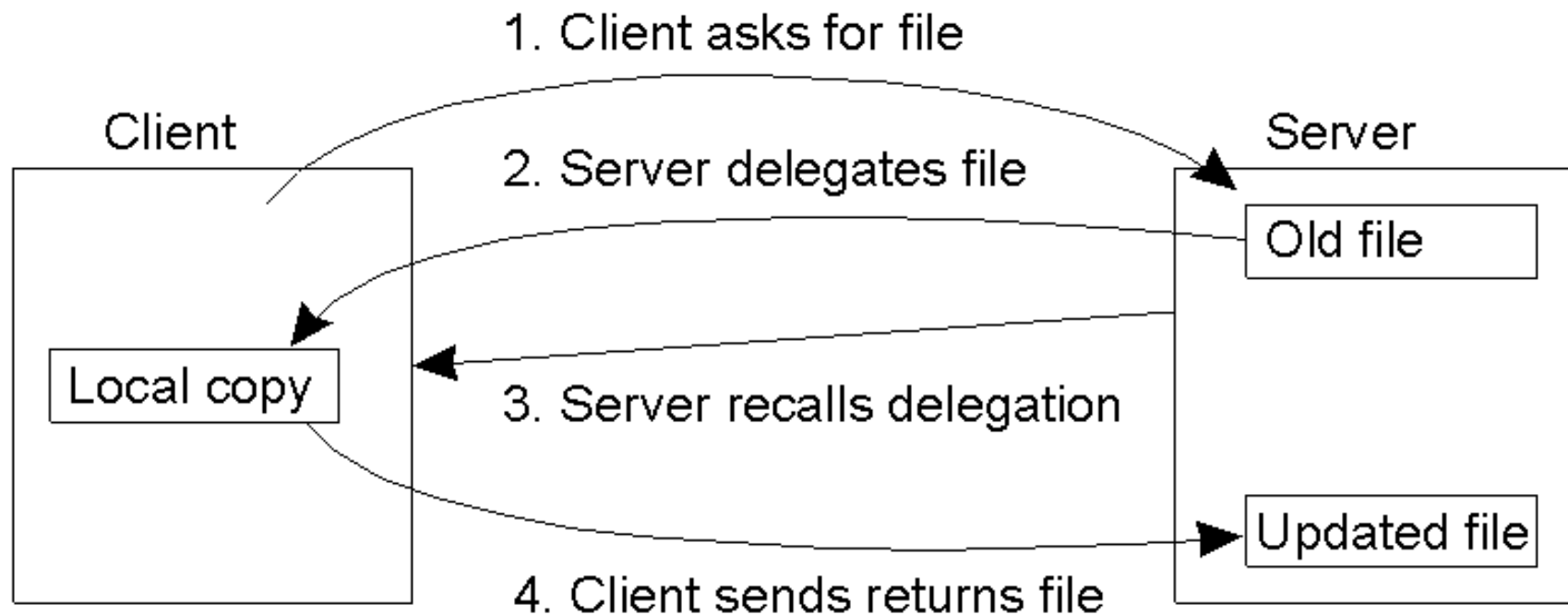
Client Caching

- Client-side caching is left to the implementation (NFS does not prohibit it)
 - Different implementation use different caching policies
 - Sun: allow cache data to be stale for up to 30 seconds

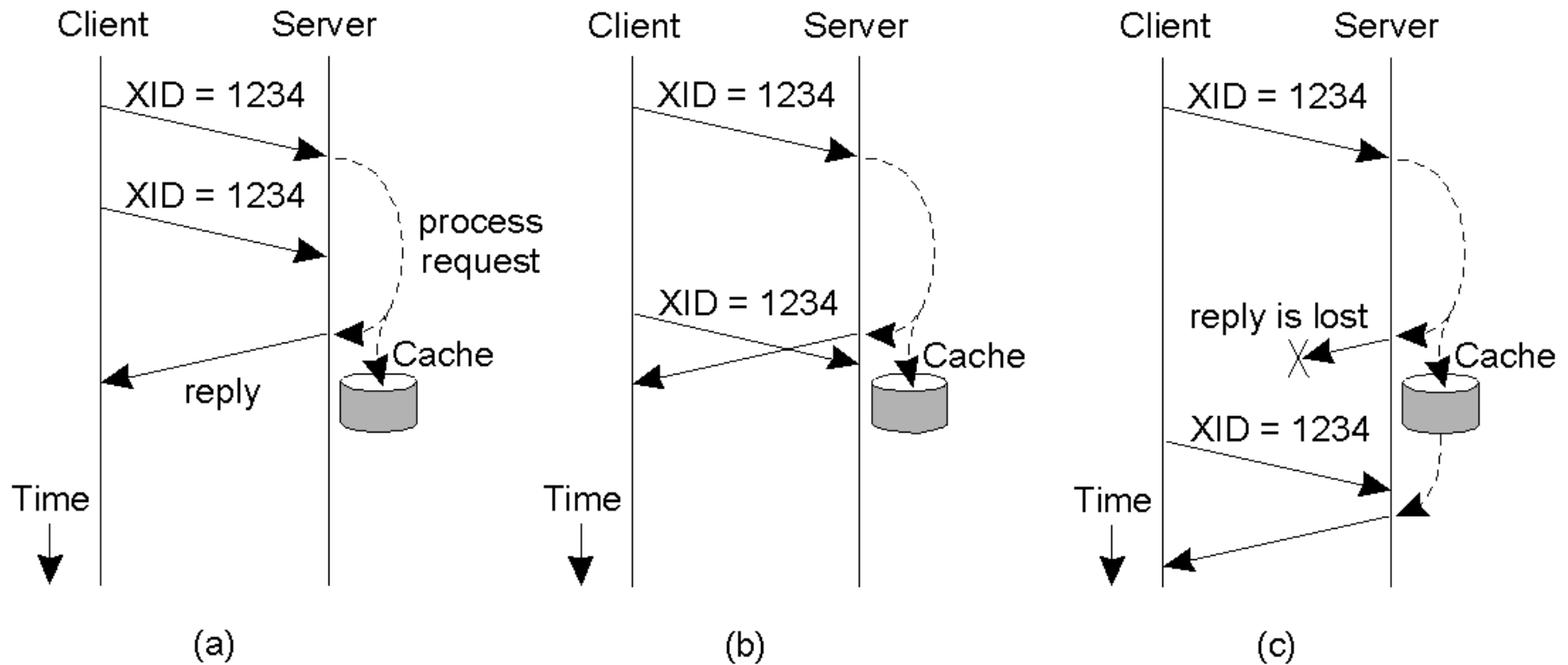


Client Caching: Delegation

- 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



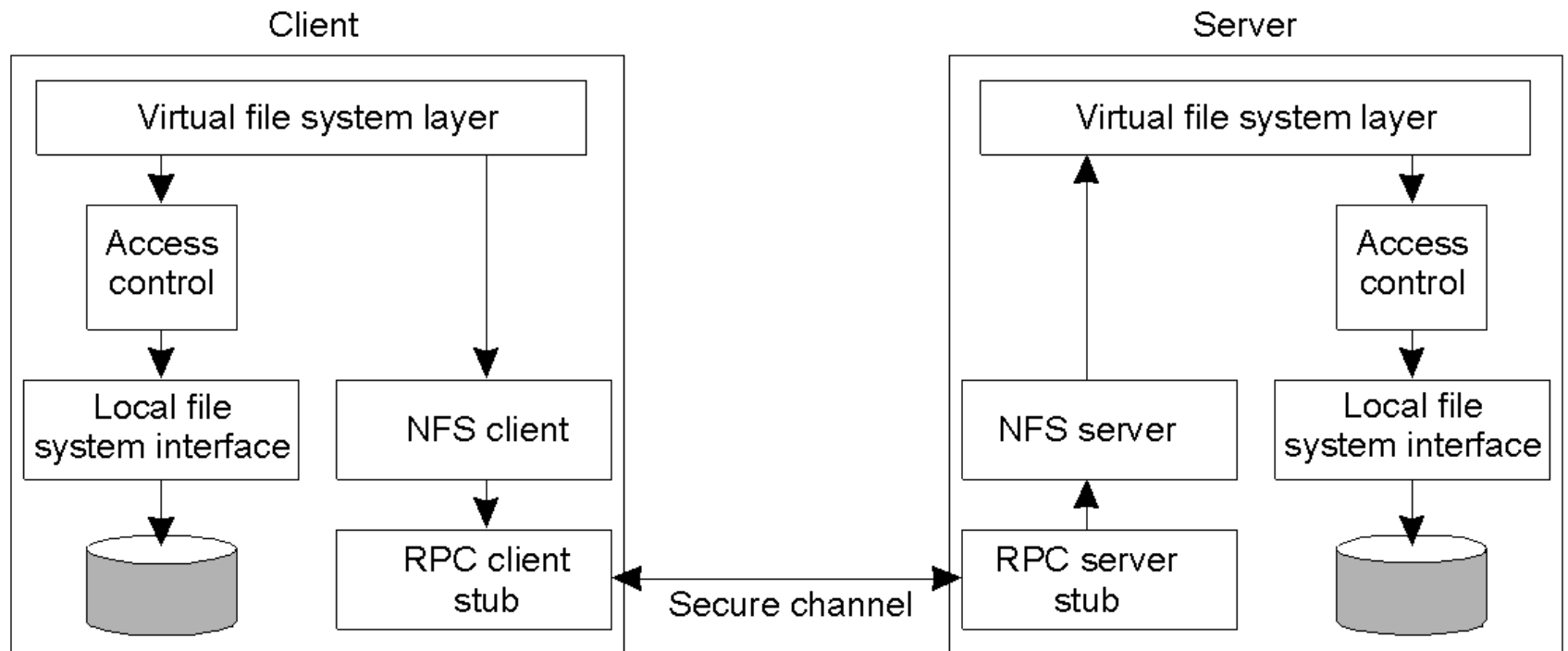
- Three situations for handling retransmissions: use a duplicate request cache
- a) The request is still in progress
- b) The reply has just been returned
- c) The reply has been some time ago, but was lost.

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

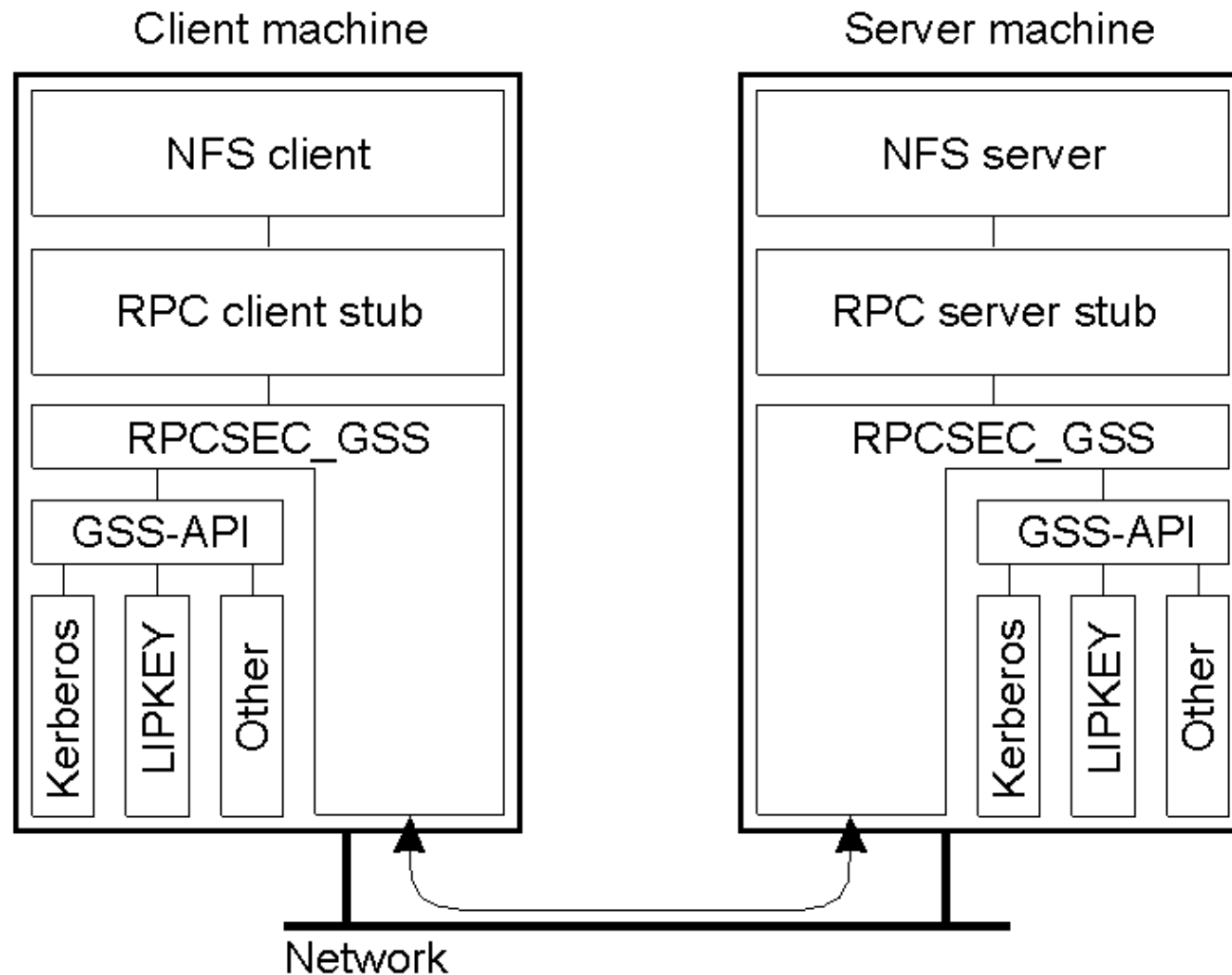


Security

- 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

