Distributed Operating Systems

Distributed File Systems

Topics

- Client-Server Architecture
- Cluster-Based Distributed File Systems
- Single and Compound Communications
- Remote Procedure Calls in NFS
- Enhancement to RPC (RPC2)
- Naming in NFS
- Semantics of File Sharing
- File Locking
- Client-Side Caching
- Security in NFS

Motivation

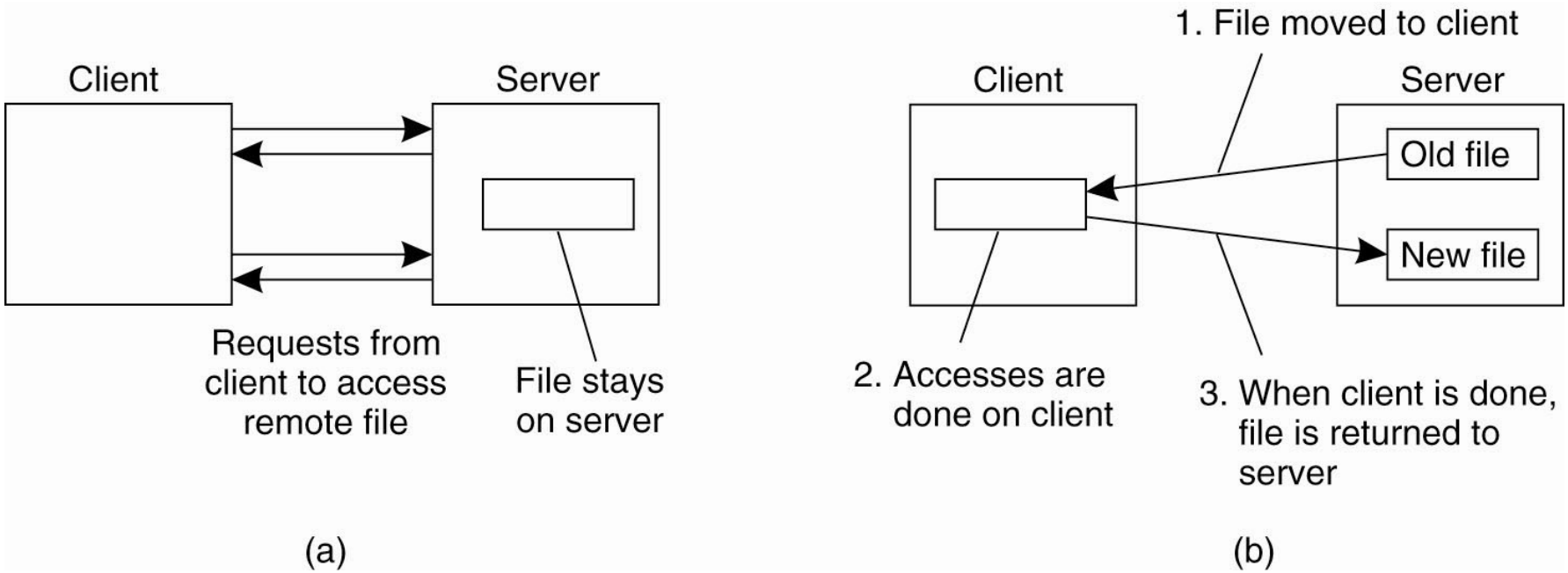
- Sharing data is fundamental to distributed systems
- Distributed file systems as a method of sharing data, form the basis for many distributed applications.
- Distributed file systems allow multiple processes to **share** data over long periods of time in a **secure** and **reliable** way.
- Most distributed file systems are built following a traditional client-server architecture, but fully decentralized solutions exist as well.

Client-Server Architecture (1)

- Many distributed files systems are organized using client-server architectures.

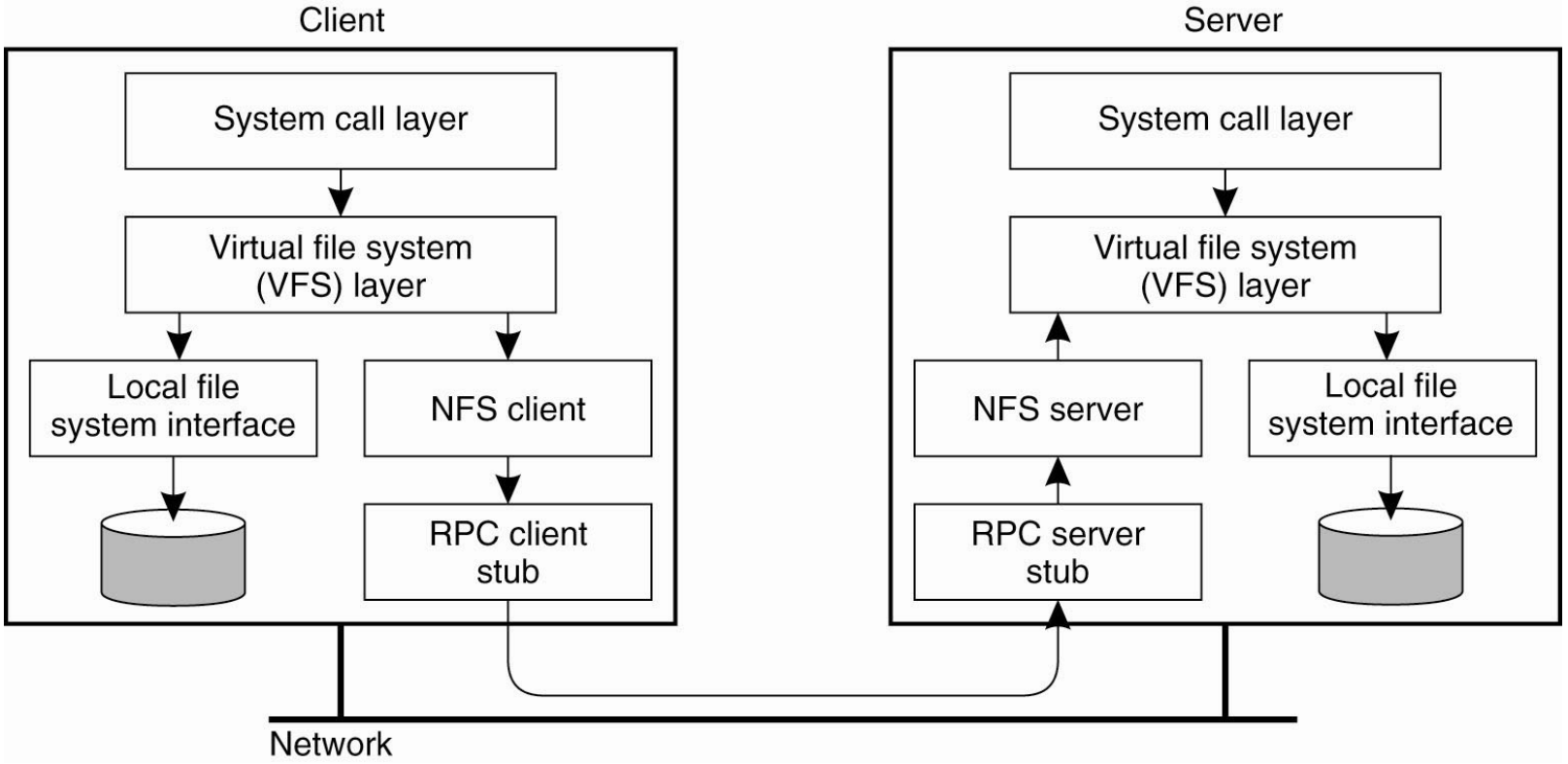
- The most widely used model is Sun Microsystem's Network File System (NFS) deployed for UNIX-based systems.

Client-Server Architectures (2)



(a) The remote access model.
(b) The upload/download model.

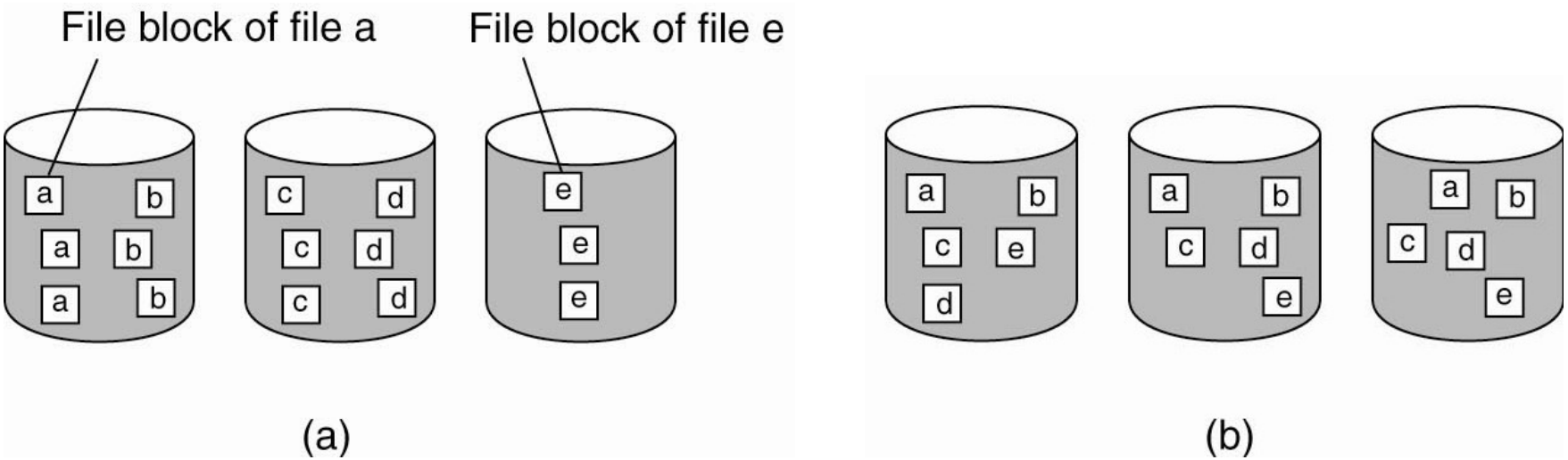
Client-Server Architectures (3)



Cluster-Based Distributed File Systems (1)

- The client-server architecture is often enhanced for server clusters.
- The file systems are adjusted for parallel applications.
- File-striping techniques are used to distribute files across multiple servers.
- Such an organization works well only if the application is organized in such a way that parallel data access makes sense (for example, a dense matrix).
- For general-purpose applications file striping may not be an effective tool.

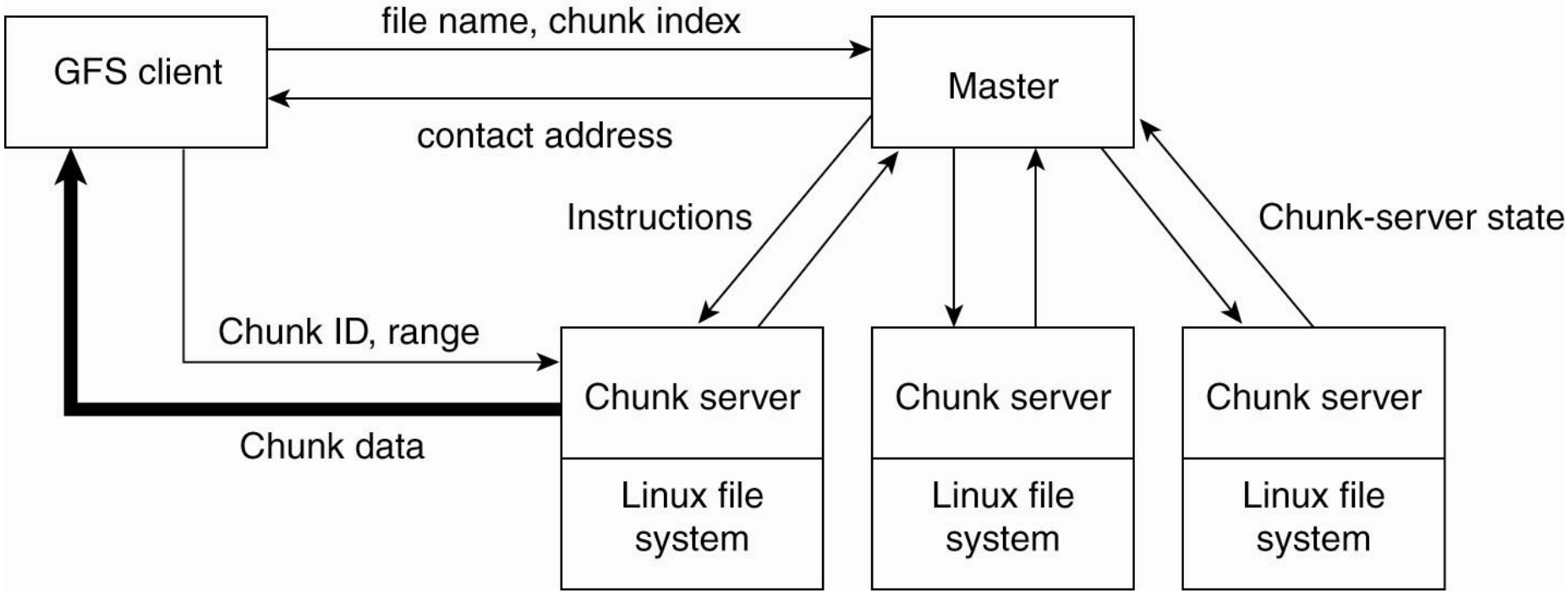
Cluster-Based Distributed File Systems (2)



The difference between

(a) distributing whole files across several servers
(b) striping files for parallel access.

Cluster-Based Distributed File Systems (3)

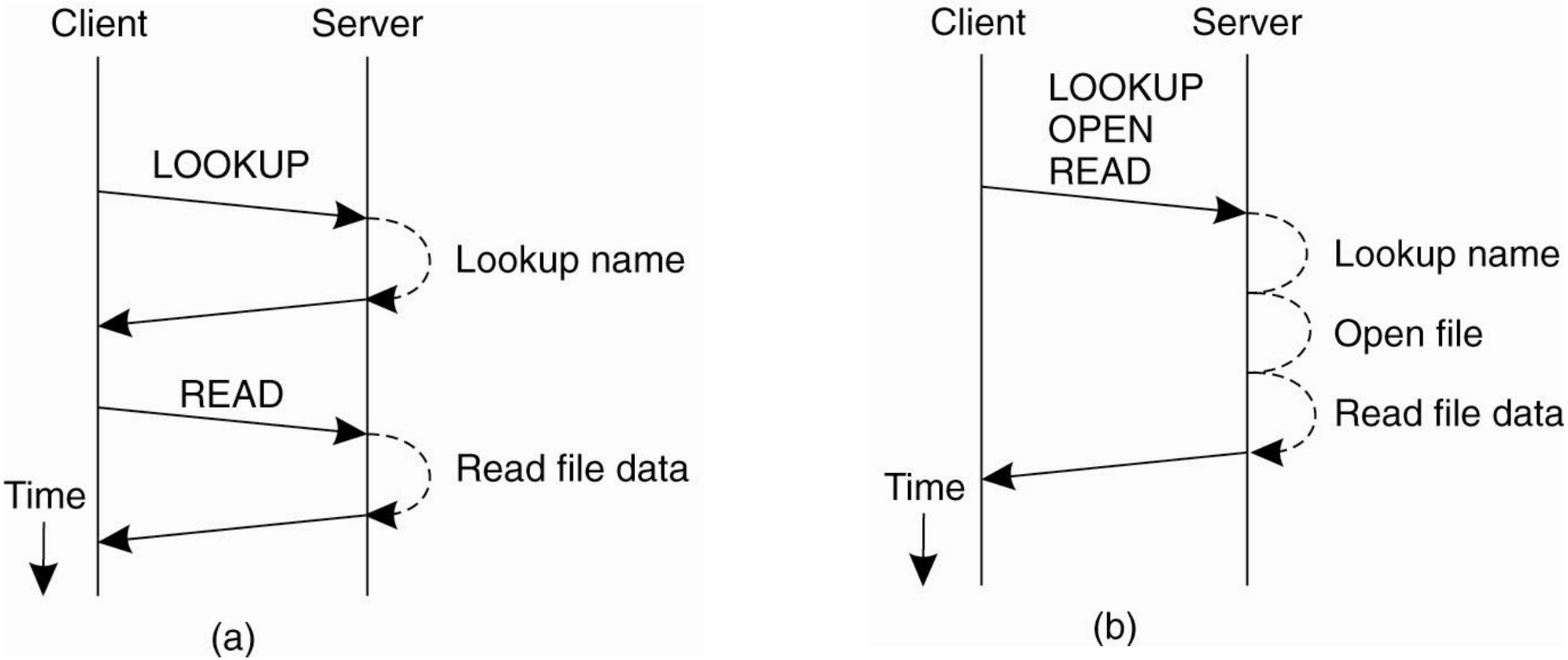


The organization of a Google cluster of servers.

Single and Compound Communications

- Every NFS operation can be implemented as a single remote procedure call to a file server.
- The operations can be grouped together in a compound remote procedure call.
- Compound procedure calls are simply handled in the order as requested.
- If there are concurrent operations from other clients, then no measures are taken to avoid conflicts.
- If an operation fails for whatever reason, then no further operations in the compound procedure are executed, and the results found so far are returned to the client.

Remote Procedure Calls in NFS



(a) Reading data from a file in NFS using single procedure

(b) Reading data using a compound procedure

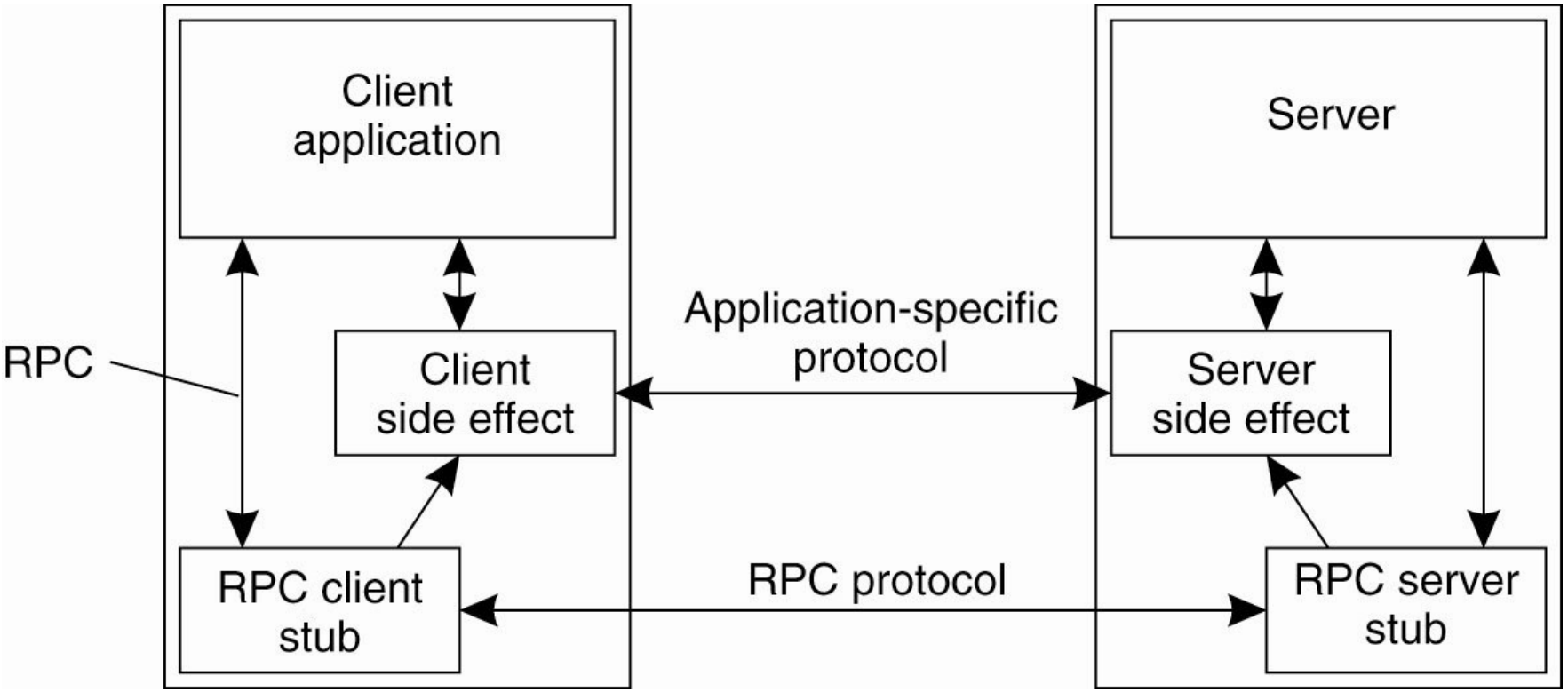
Enhancement to RPC (RPC2)

- RPC2 offers reliable RPCs on top of the (unreliable) UDP protocol.
- Each time a remote procedure is called, the RPC2 client code starts a new thread that sends an invocation request to the server and subsequently blocks until it receives an answer.
- As request processing may take an arbitrary time to complete, the server regularly sends back messages to the client to let it know it is still working on the request.
- If the server dies, sooner or later this thread will notice that the messages have ceased and report back failure to the calling application.

Side Effect in RPC2

- RPC2 supports side effects which is a mechanism by which the client and server can communicate using an application-specific protocol.
- For example, a client opening a file at a video server.
- Video data transfer from the server to the client should be guaranteed to be within a minimum and maximum end-to-end delay.
- RPC2 allows the client and the server to set up a separate connection for transferring the video data to the client on time as a side effect of an RPC call to the server.

The RPC2 Subsystem

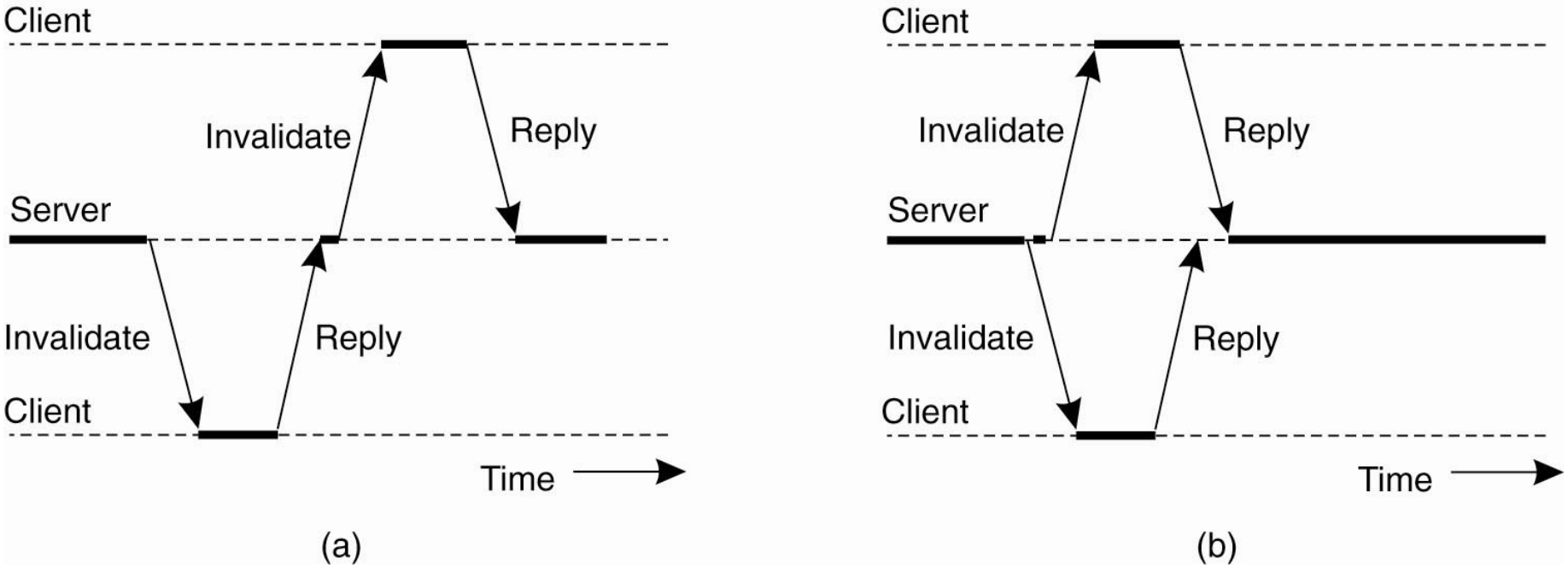


Side effects in Coda's RPC2 system.

Support for Multicasting in RPC2

- An important design issue in Coda is that servers keep track of which clients have a local copy of a file.
- When a file is modified, a server invalidates local copies by notifying the appropriate clients through an RPC.
- Invalidate messages can be multicast.

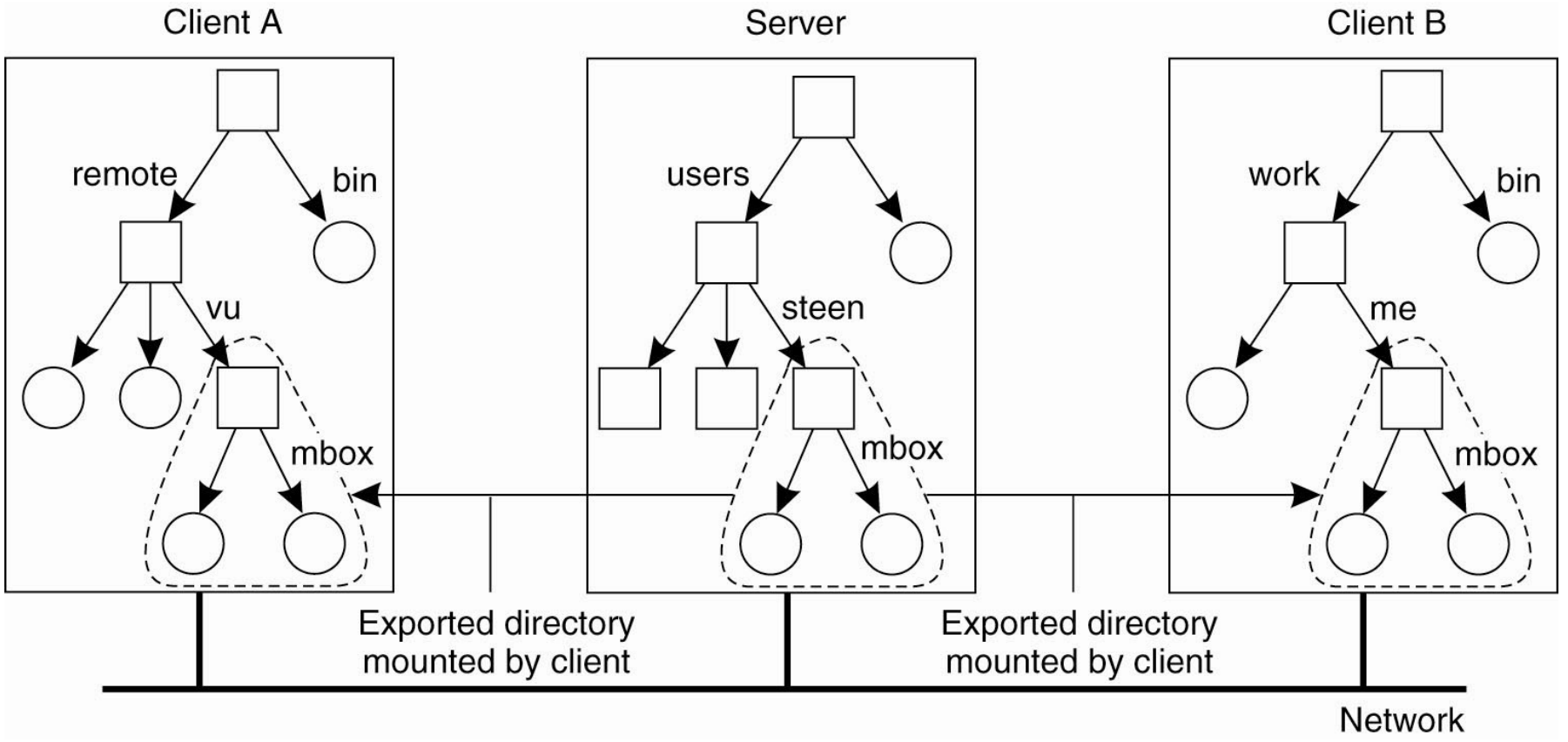
The RPC2 Subsystem



(a) Sending an invalidation message one at a time.

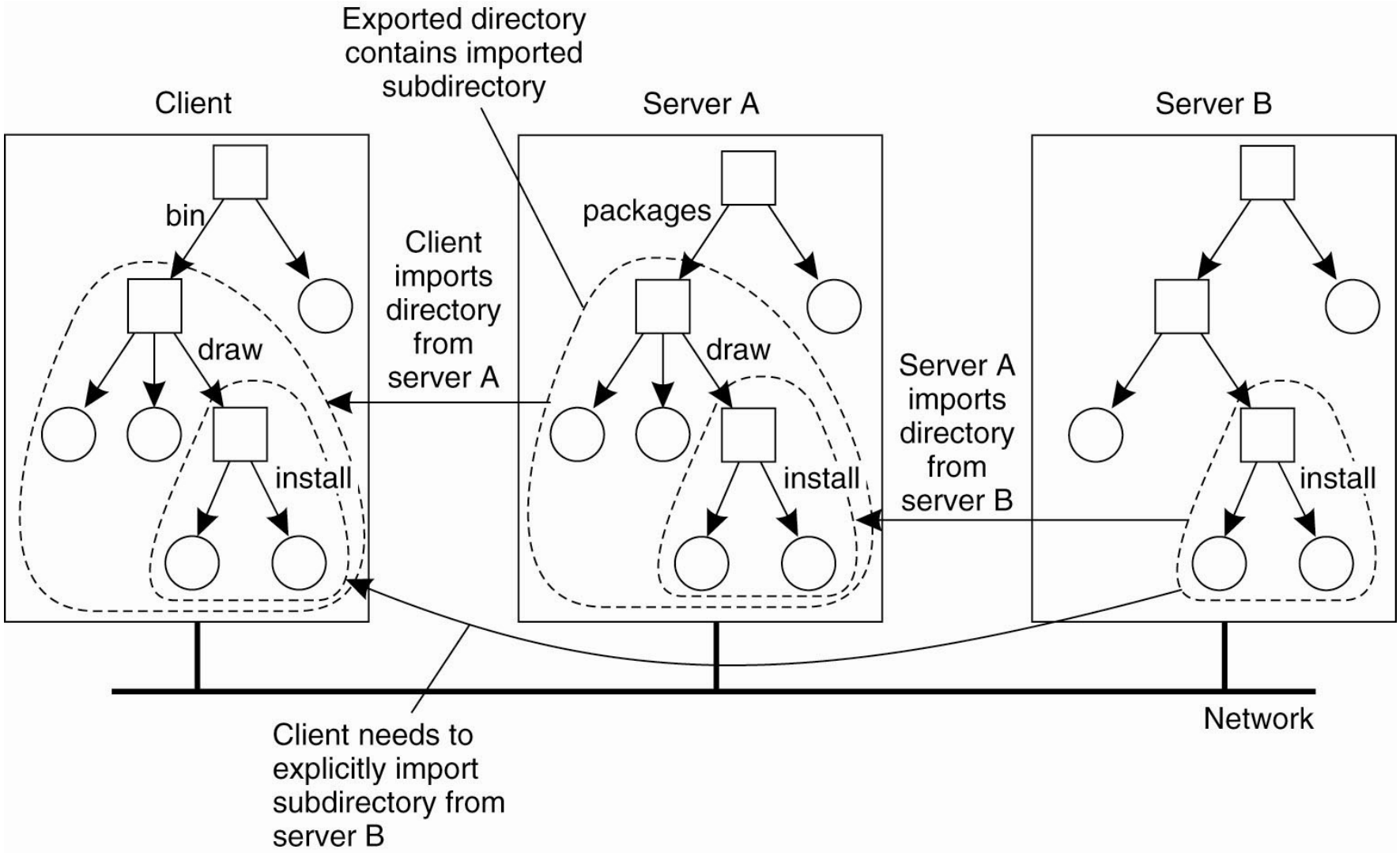
(b) Sending invalidation messages in parallel.

Naming in NFS (1)



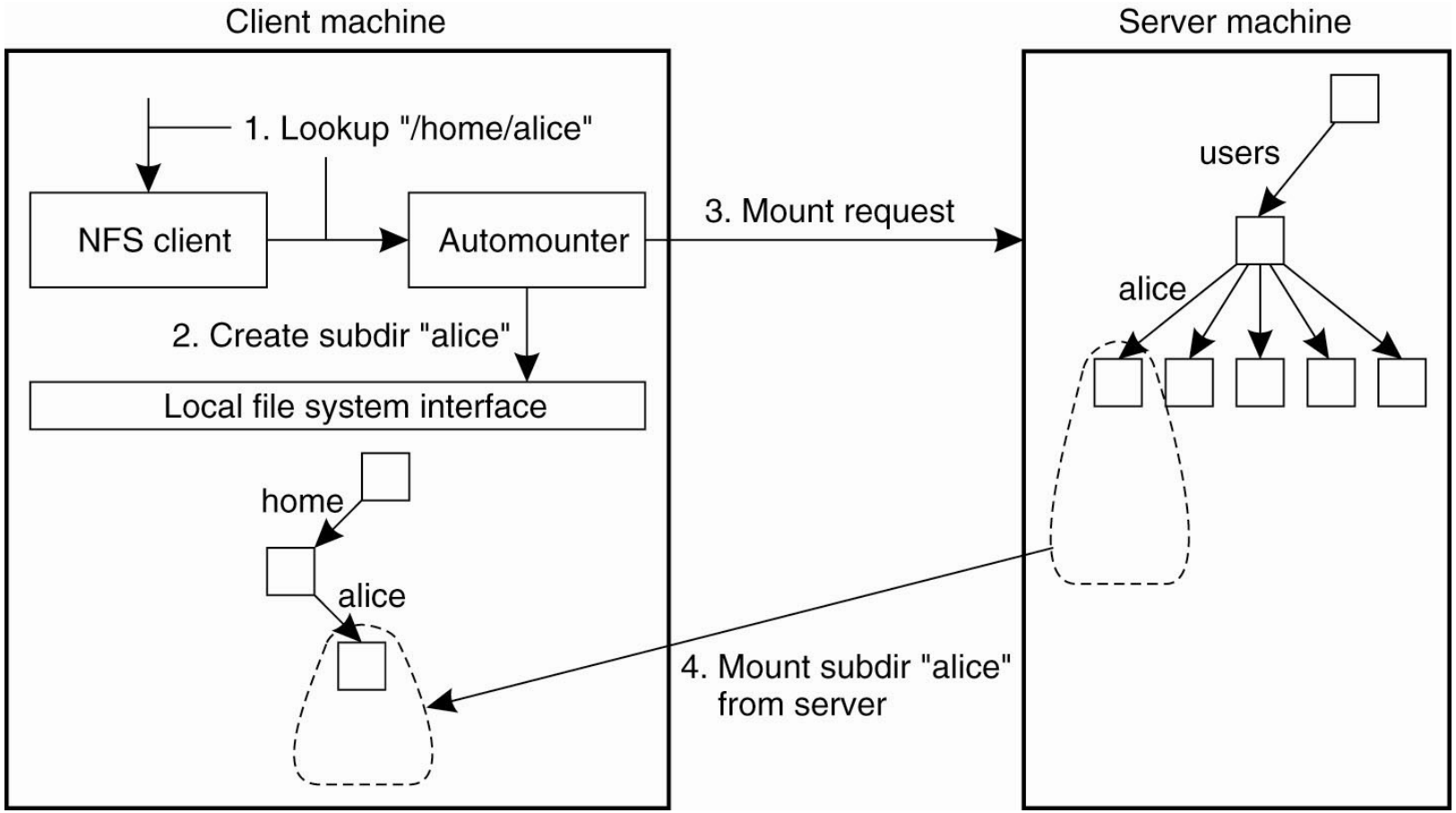
Mounting (part of) a remote file system in NFS.

Naming in NFS (2)



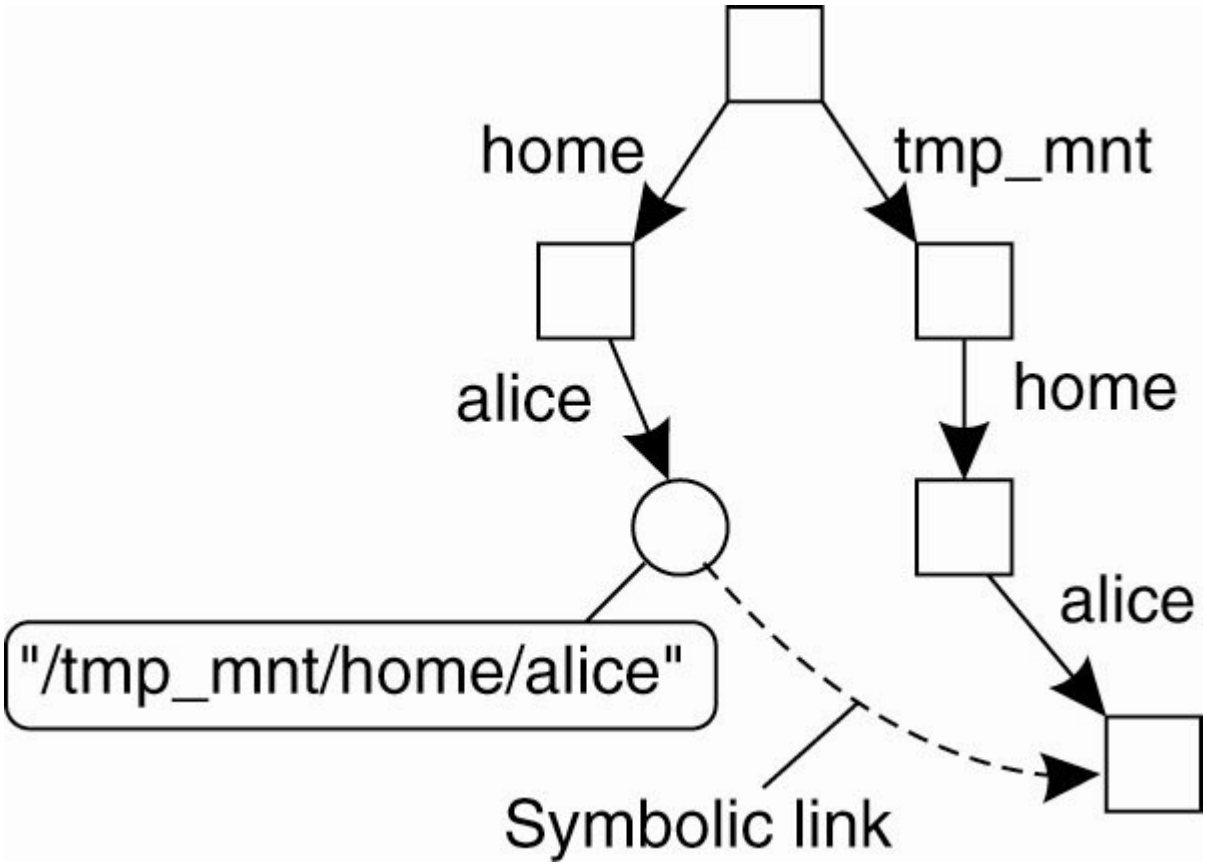
Mounting nested directories from multiple servers in NFS.

Automounting (1)



A simple automounter for NFS.

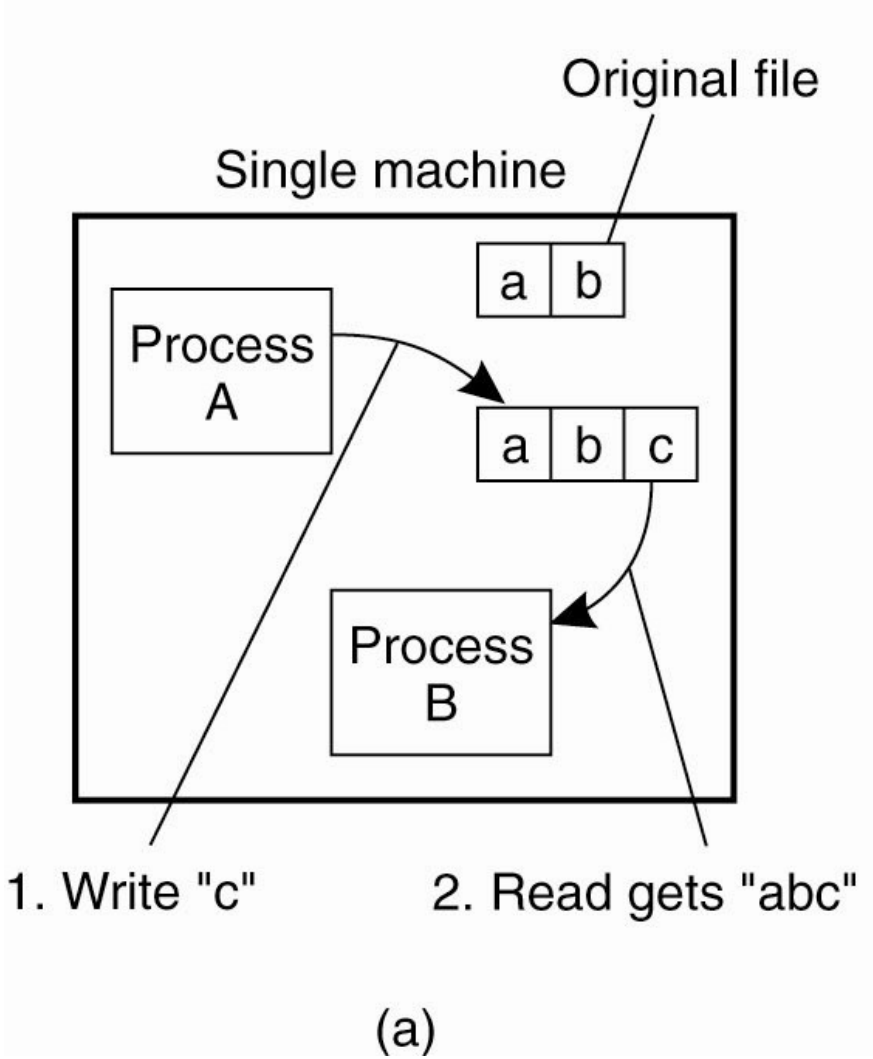
Automounting (2)



Using symbolic links with automounting.

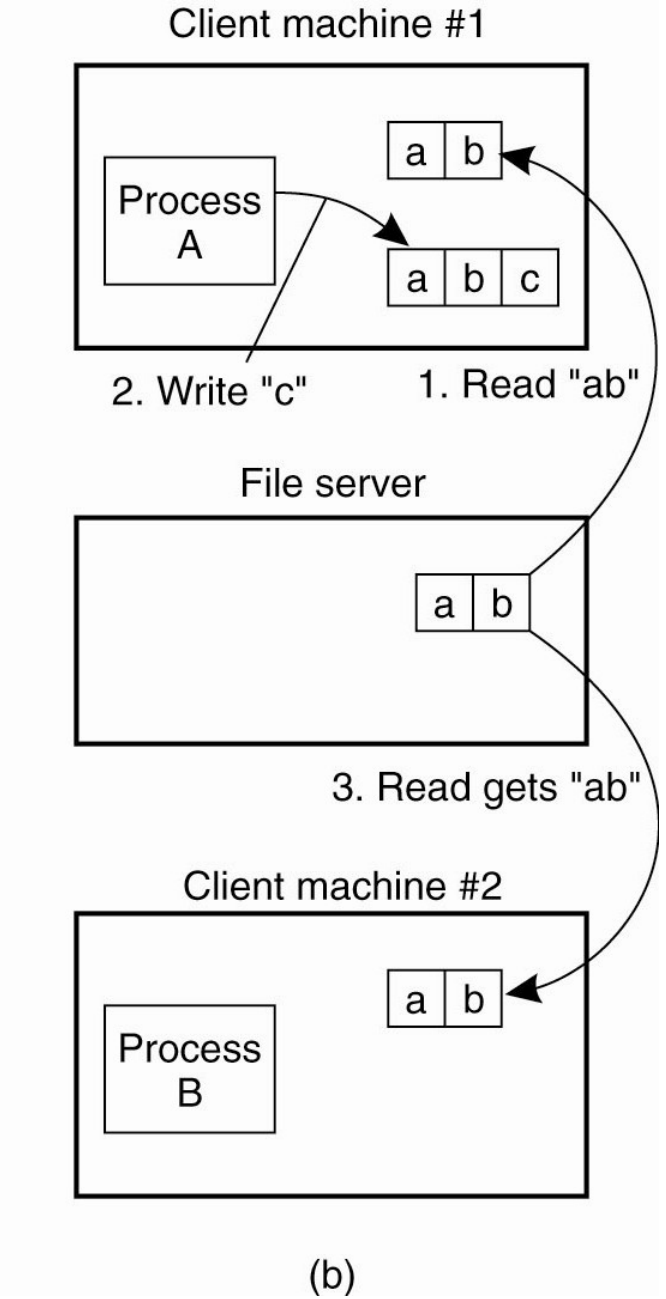
Semantics of File Sharing (1)

On a single processor, when a read follows a write, the value returned by the read is the value just written.



Semantics of File Sharing (2)

In a distributed system with caching, obsolete values may be returned.



(b)

Semantics of File Sharing (3)

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
Transactions	All changes occur atomically

Four ways of dealing with the shared files in a distributed system.

Semantics of File Sharing (4)

- In a distributed system, UNIX semantics can be achieved if there is only one file server and clients do not cache files.
- All reads and writes go directly to the file server, which processes them strictly sequentially.
- In practice, however, the performance of a distributed system in which all file requests must go to a single server is poor.
- This problem is often solved by allowing clients to maintain local copies of heavily-used files in their private (local) caches.
- One solution of cache inconsistency is to propagate all changes to cached files back to the server immediately.

File Locking (1)

Operation	Description
Lock	Create a lock for a range of bytes
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 operations related to file locking.

Share Reservation

- Share reservation is used to implement NFS for Windows-based systems.
- When a client opens a file, it specifies the type of access it requires (namely READ, WRITE, or BOTH), and which type of access the server should deny other clients (NONE, READ, WRITE, or BOTH).
- If the server cannot meet the client‘s requirements, the open operation will fail for that client.

File Locking (2)

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)

The result of an open operation with share reservations in NFS.

When the client requests shared access given the current denial state.

File Locking (3)

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

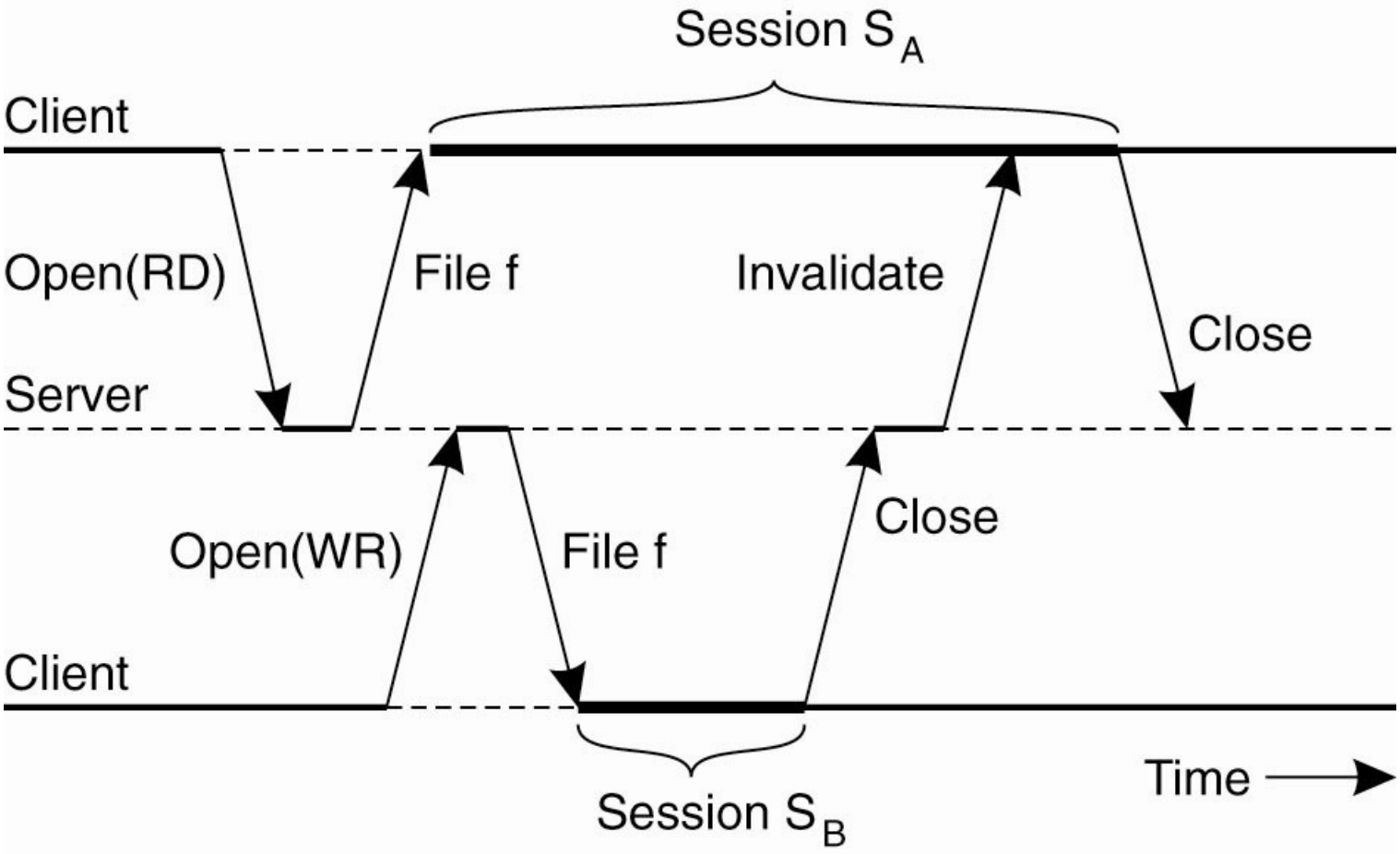
(b)

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

Sharing Files in Coda (1)

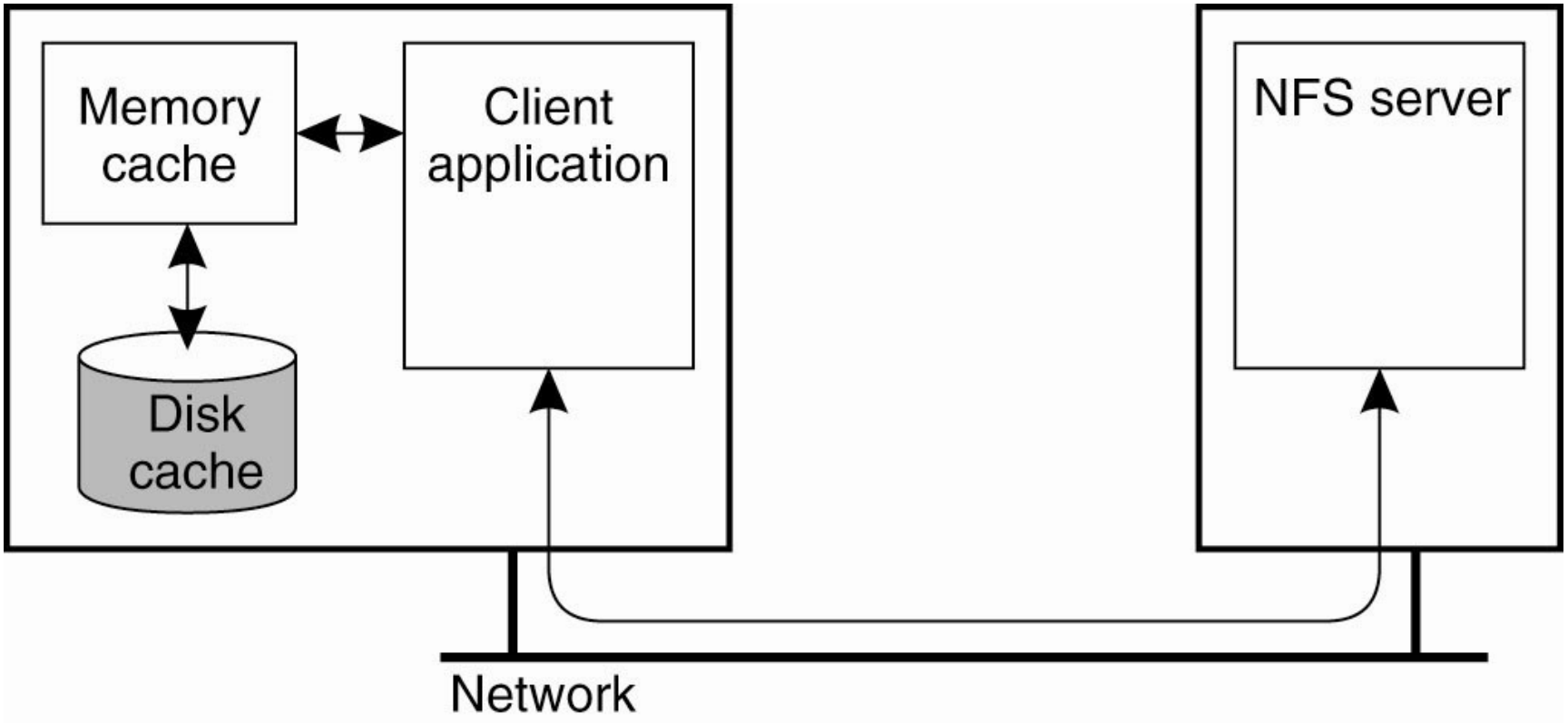
- When a client successfully opens a file *f*, an entire copy of *f* is transferred to the client's machine. The server records that the client has a copy of *f*.
- If a client has opened file f for writing, any request to open *f* will fail. (server has recorded that *f* might have been modified).
- But if the client has opened *f* for reading, an attempt to get a copy from the server for reading would succeed. An attempt to open for writing would succeed as well.
- The second client may read from outdated copy because a session is treated as a transaction in Coda.

Sharing Files in Coda (2)

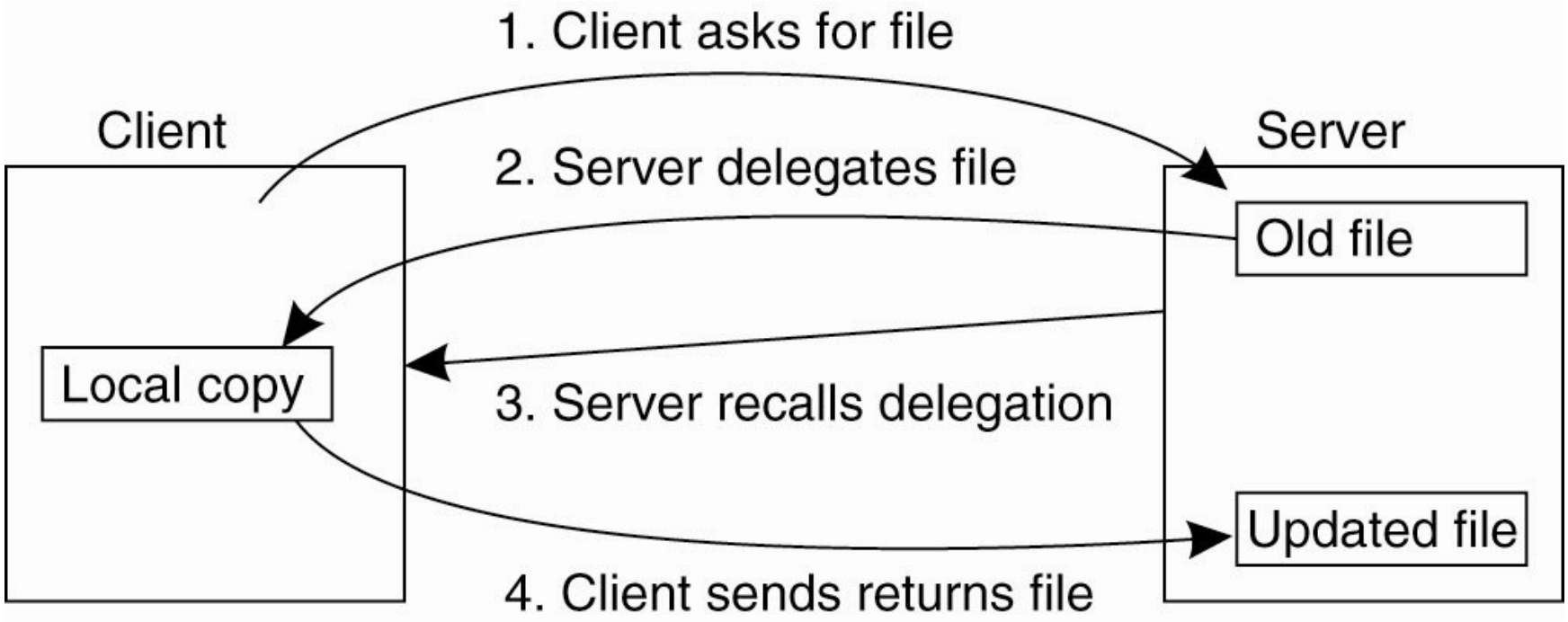


The transactional behavior in sharing files in Coda.

Client-Side Caching (1)



Client-Side Caching (2)

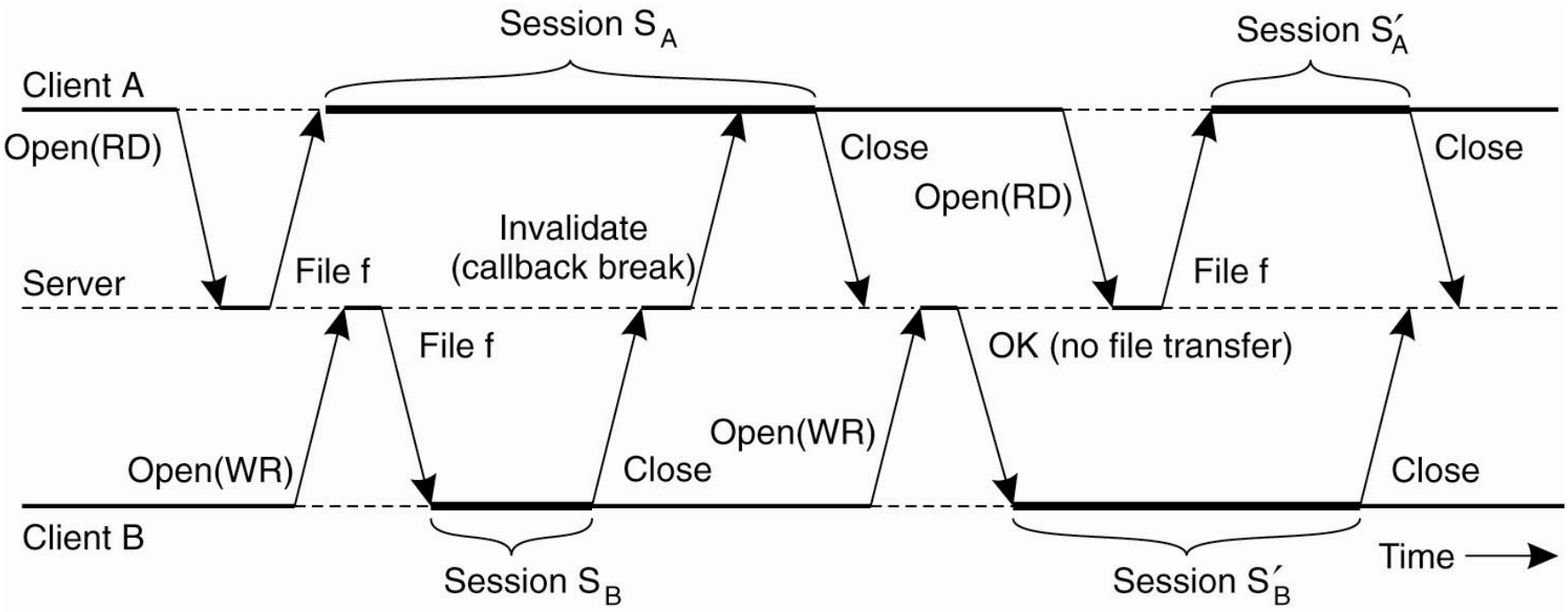


Using the NFS callback mechanism to recall file delegation.

Client-Side Caching in Coda (1)

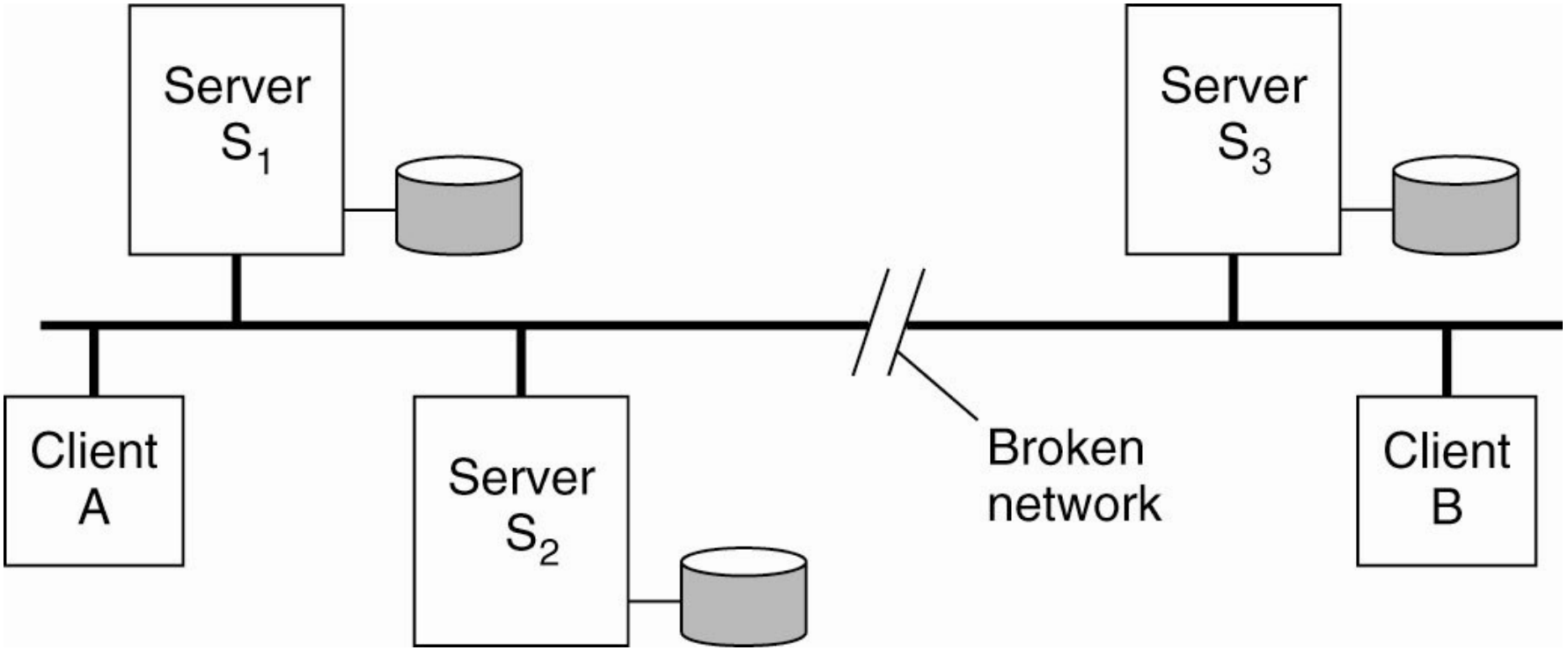
- Clients in Coda always cache entire files.
- Cache coherence in Coda is maintained by means of callbacks.
- For each file, the server from which a client had fetched the file keeps track of which clients have a copy of that file cached locally.
- When a client updates its local copy of the file for the first time, it notifies the server, which, in turn, sends an invalidation message to the other clients.
- Such an invalidation message is called a callback break
- The server will then discard the callback list it held for the client it just sent an invalidation.

Client-Side Caching in Coda (2)



The use of local copies when opening a session in Coda.

Server Replication in Coda



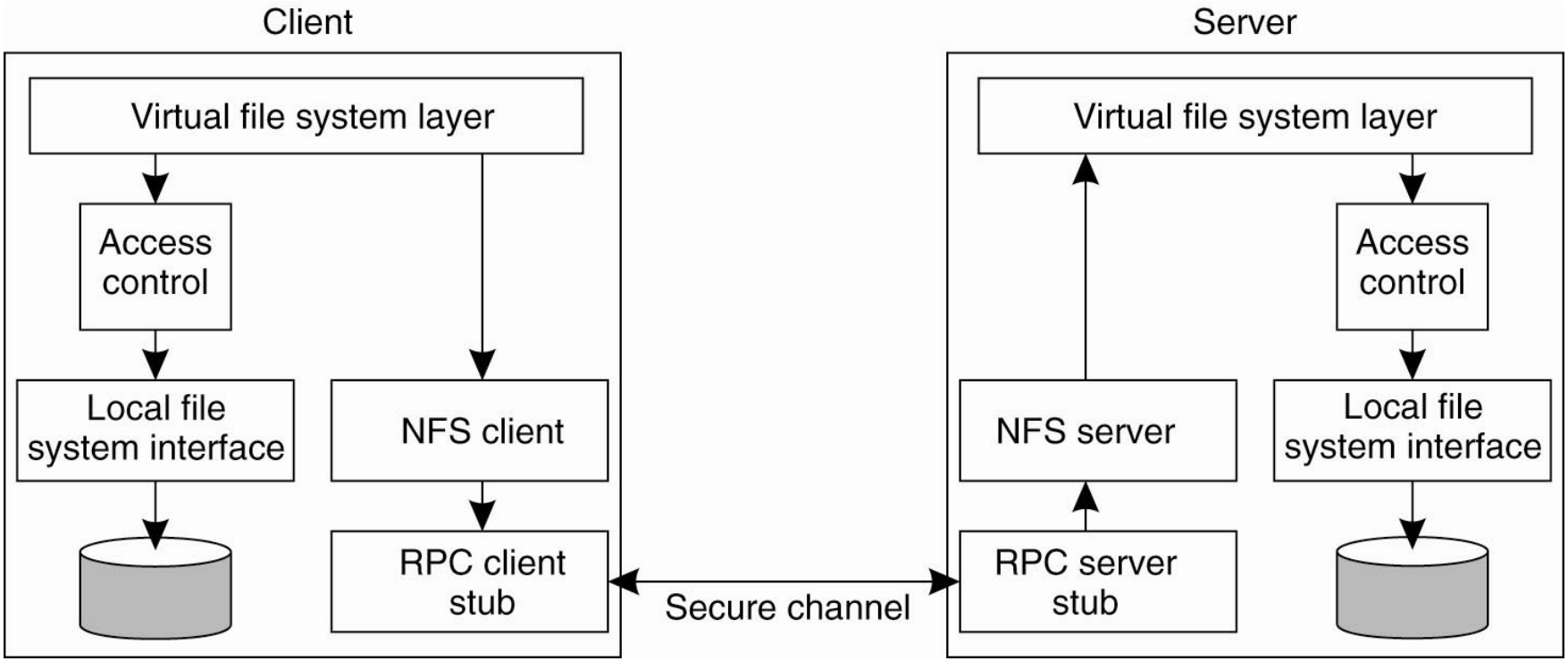
Two clients with a different Accessible Volume Storage Group (AVSG) for the same replicated file.

Coda uses a variant of Read-One, Write-All (ROWA) for consistency

Security in NFS

- The basic idea behind NFS is that a remote file system should be presented to clients as if it were a local file system.
- Security in NFS mainly focuses on the communication between a client and a server.
- In addition to secure RPCs, it is necessary to control file accesses.
- A file server is responsible for verifying the access rights of its clients

Security in NFS

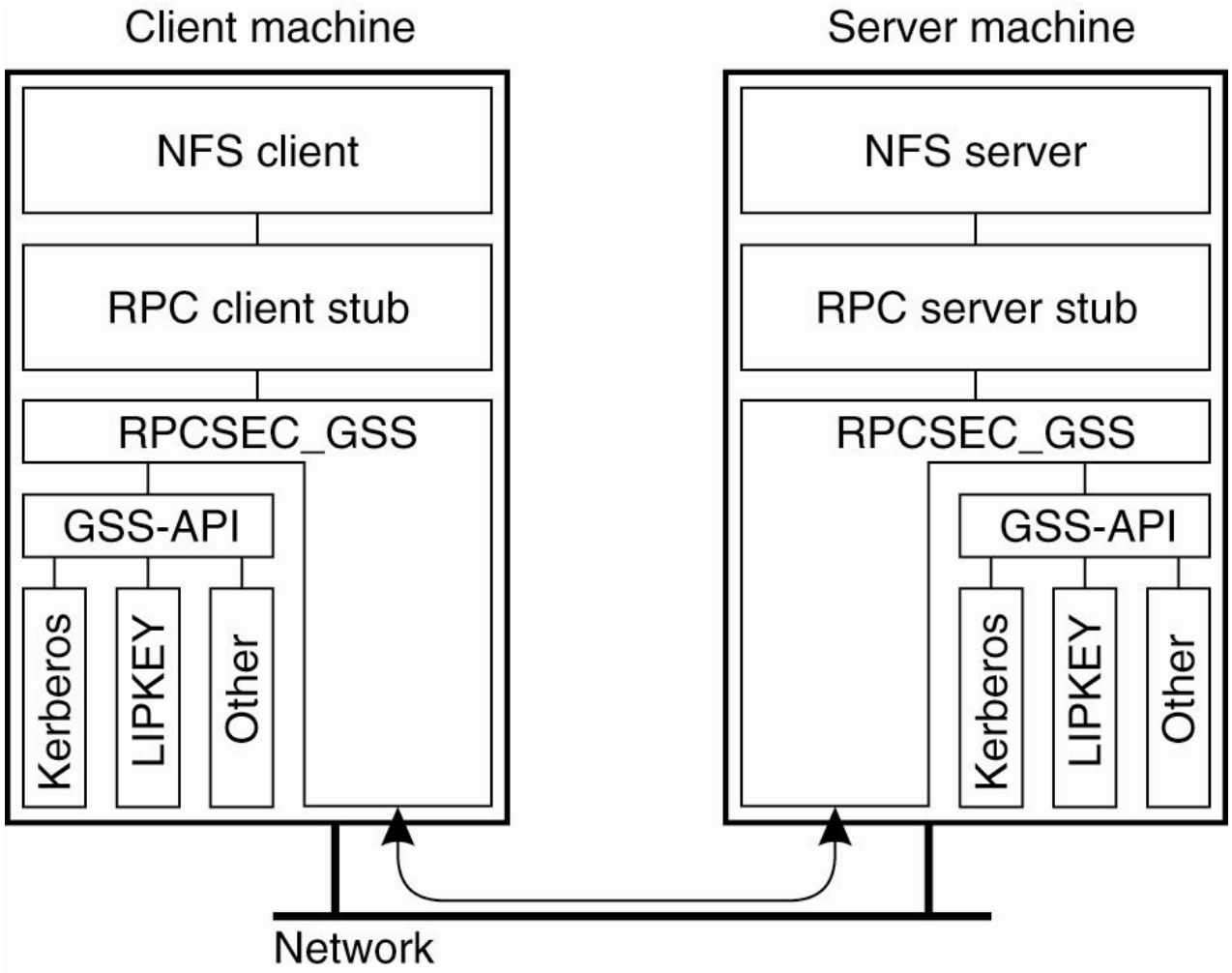


The NFS security architecture.

Secure RPCs (1)

- Security is enhanced by the support for RPCSEC_GSS.
- RPCSEC_GSS is a general security framework that can support a myriad of security mechanism for setting up secure channels Not only provides different authentication systems, but also supports message integrity
- Supports public-key systems that allows clients to be authenticated using a password while servers can be authenticated using a public key.
- The important aspect of secure RPC in NFS is that the designers have chosen not to provide their own security mechanisms, but only to provide a standard way for handling security.

Secure RPCs (2)



Access Control

Type of user	Description
Owner	The owner of a file
Group	The group of users associated with a file
Everyone	Any user or process
Interactive	Any process accessing the file from an interactive terminal
Network	Any process accessing the file via the network
Dialup	Any process accessing the file through a dialup connection to the server
Batch	Any process accessing the file as part of batch job
Anonymous	Anyone accessing the file without authentication
Authenticated	Any authenticated user or process
Service	Any system-defined service process

The various kinds of users and processes distinguished by NFS with respect to access control.

Questions?