Sistemas de Archivos Distribuidos

1

Material Complementario!

Sistemas Operativos y Distribuidos

Mg. Javier Echaiz
D.C.I.C. – U.N.S.
http://cs.uns.edu.ar/~jechaiz
je@cs.uns.edu.ar

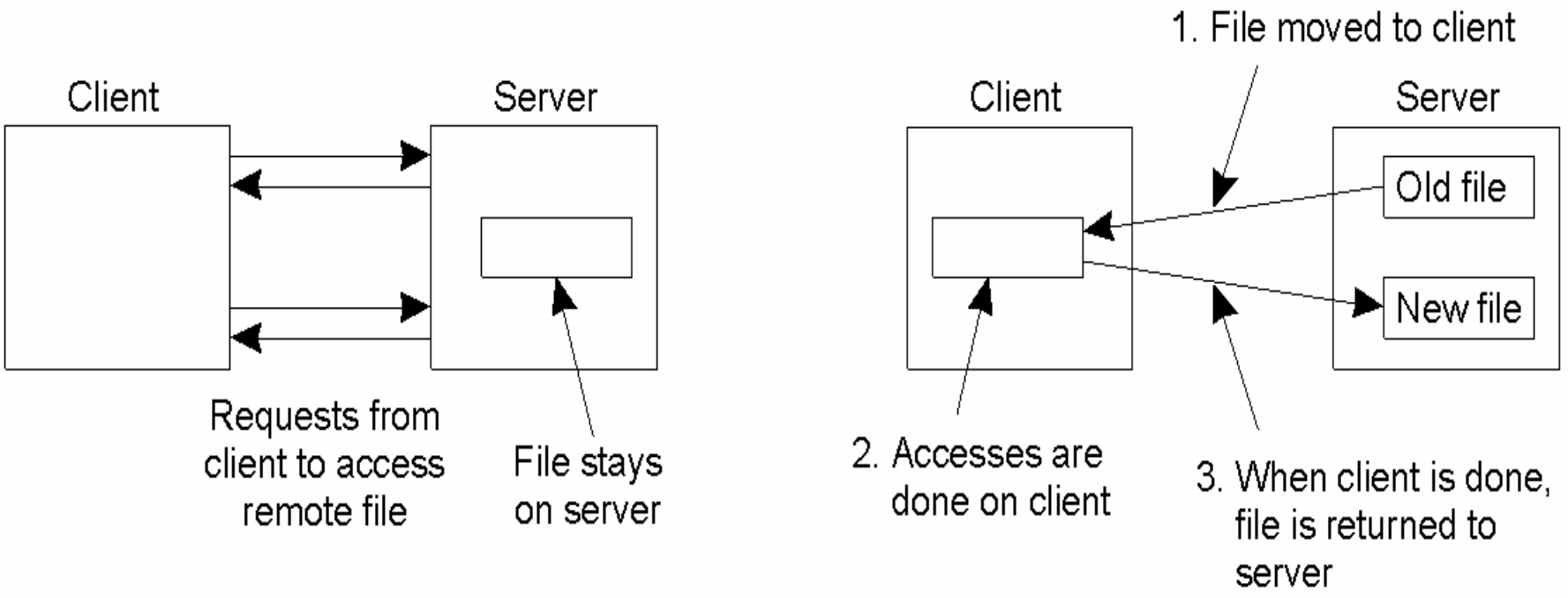
Distributed File Systems

Slides by Manolis Marazakis
maraz@csd.uoc.gr

Desirable properties

- Network transparency
- Location transparency & independence
- Fault tolerance
- Scalability
- File mobility
- User mobility

Remote file access methods

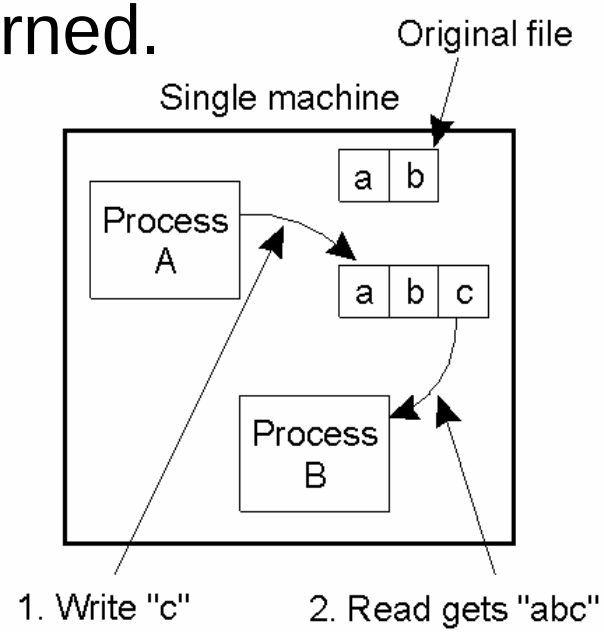


a) The remote access model.
b) The upload/download (“session”) model

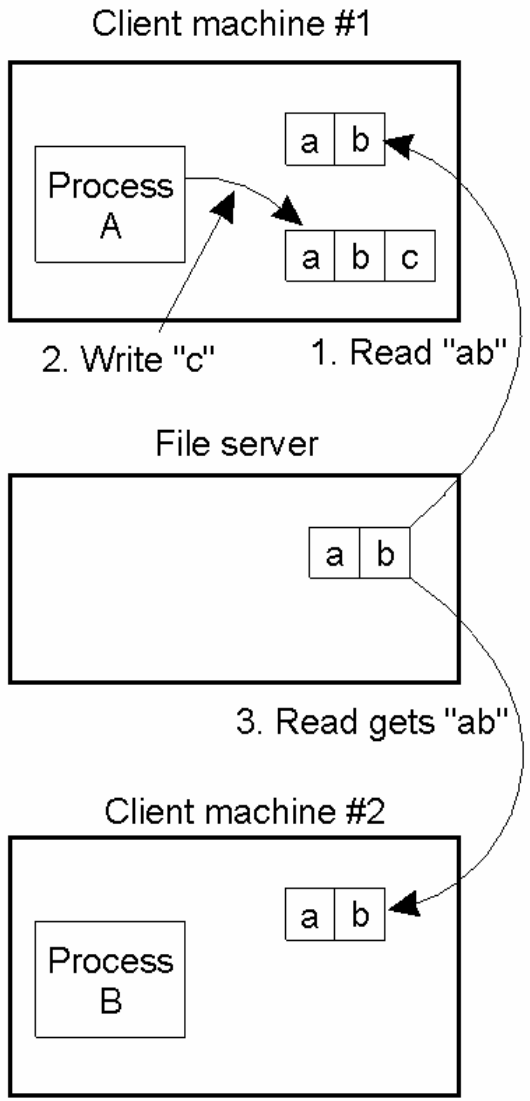
Semantics of File Sharing (I)

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 (II)

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 DFS.

NFS v2 (SUN, 1985)

- NFS protocol
  - Stateless : self-contained requests
- RPC + XDR
- NFS server + client
- Mounting protocol
  - Obtain “initial” file handle
- Network Lock manager: lockd
- Network Status Manager: statd
- Daemon processes: nfsd, mountd, biod

Stateless NFS

- Server does not maintain state per client
  - Client requests must be self-contained
    - Include (absolute) file offset with each read/write
- Client crash:
  - Server does not need to know or care
- Server crash:
  - Client keeps sending request until it is satisfied
- Server must commit data & metadata to stable storage before responding
- Server cannot on its own perform locking
  - Separate locking service

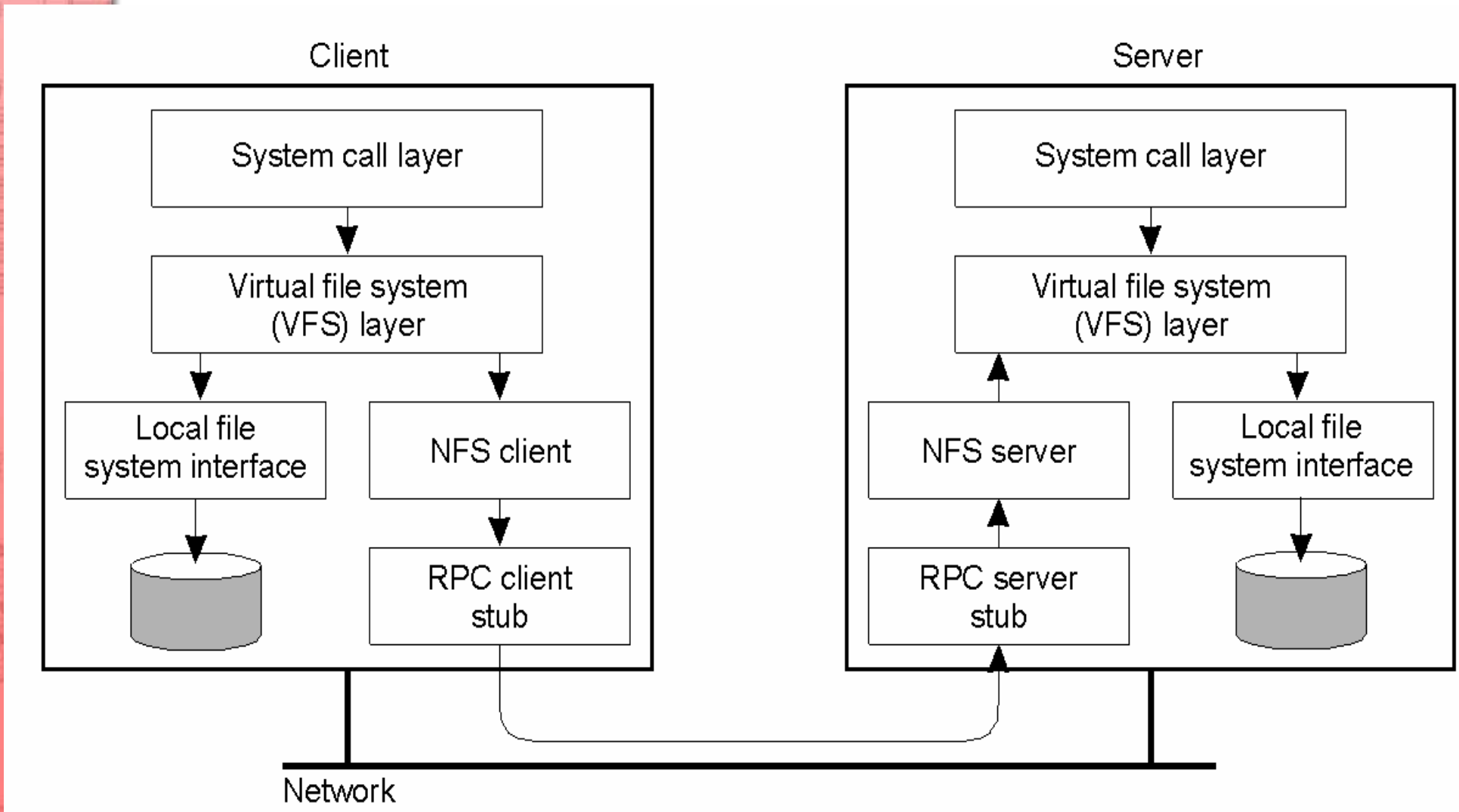
NFS v3 (1995)

- RFC-1813
- ASYN_WRITE: asynchronous write
  - Allow server-side caching
- COMMIT:
  - Flush server's write buffers
- READPLUSDIR:
  - Obtain directory's file names, handles & attributes
- 64 bits for file size/offset
  - NFS v2 allows only 32 bits

NFS v4 (IETF, 1998-2000)

- Based on SUN's WebNFS
- **Stateful protocol:**
  - Open/close requests
  - Integrates mount & locking protocols
  - Lease-based locking
  - COMPOUND request: group of multiple operations

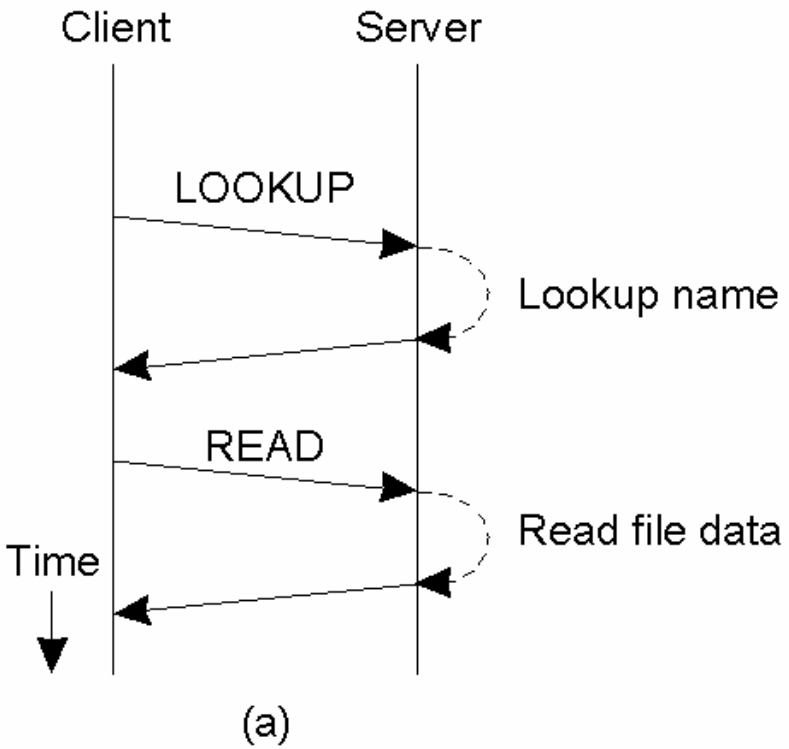
NFS Architecture

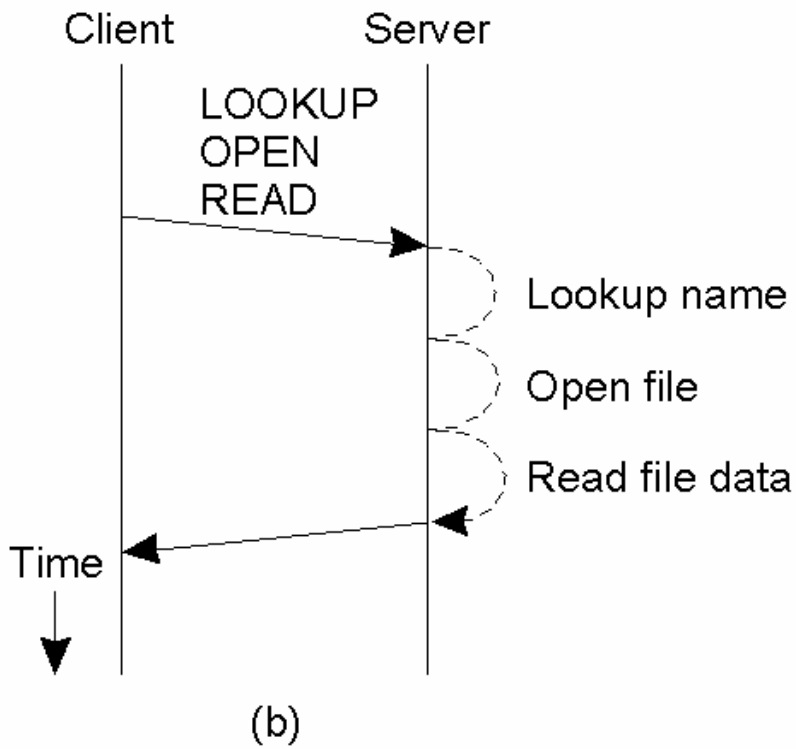


File System Operations

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



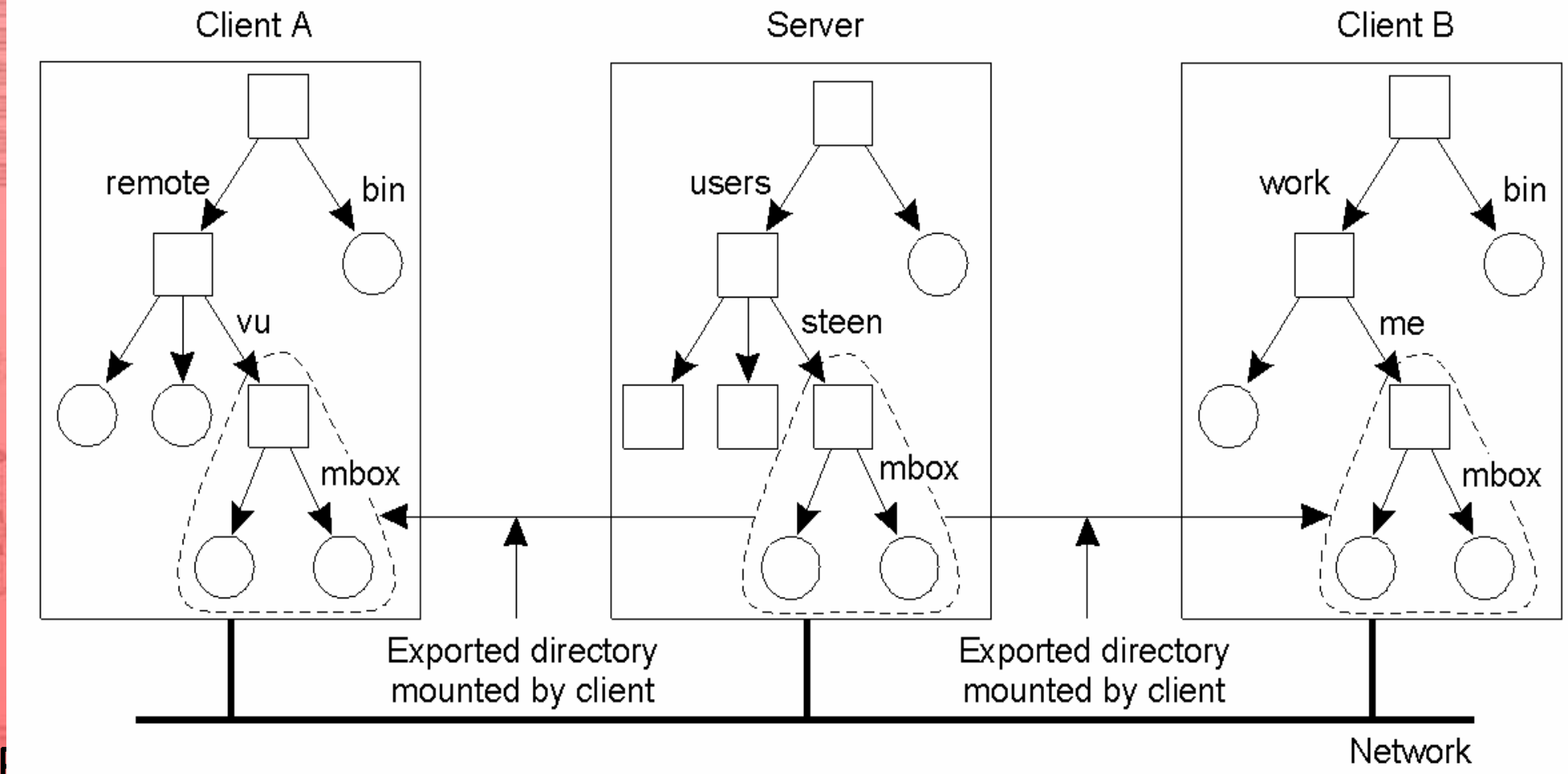


a) Reading data from a file in NFS v.3.

b) Reading data using a compound procedure in v.4.

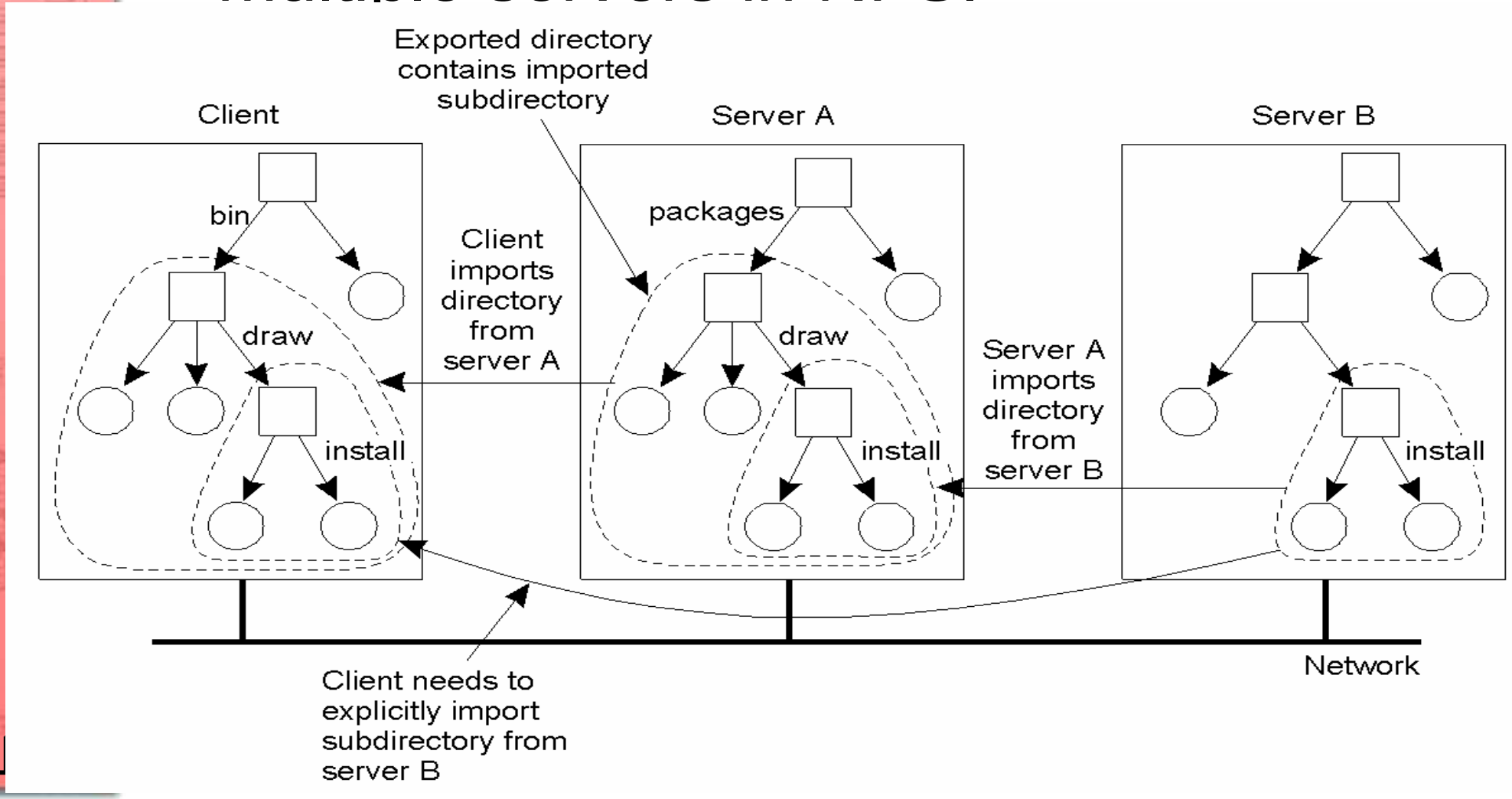
Naming (I)

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

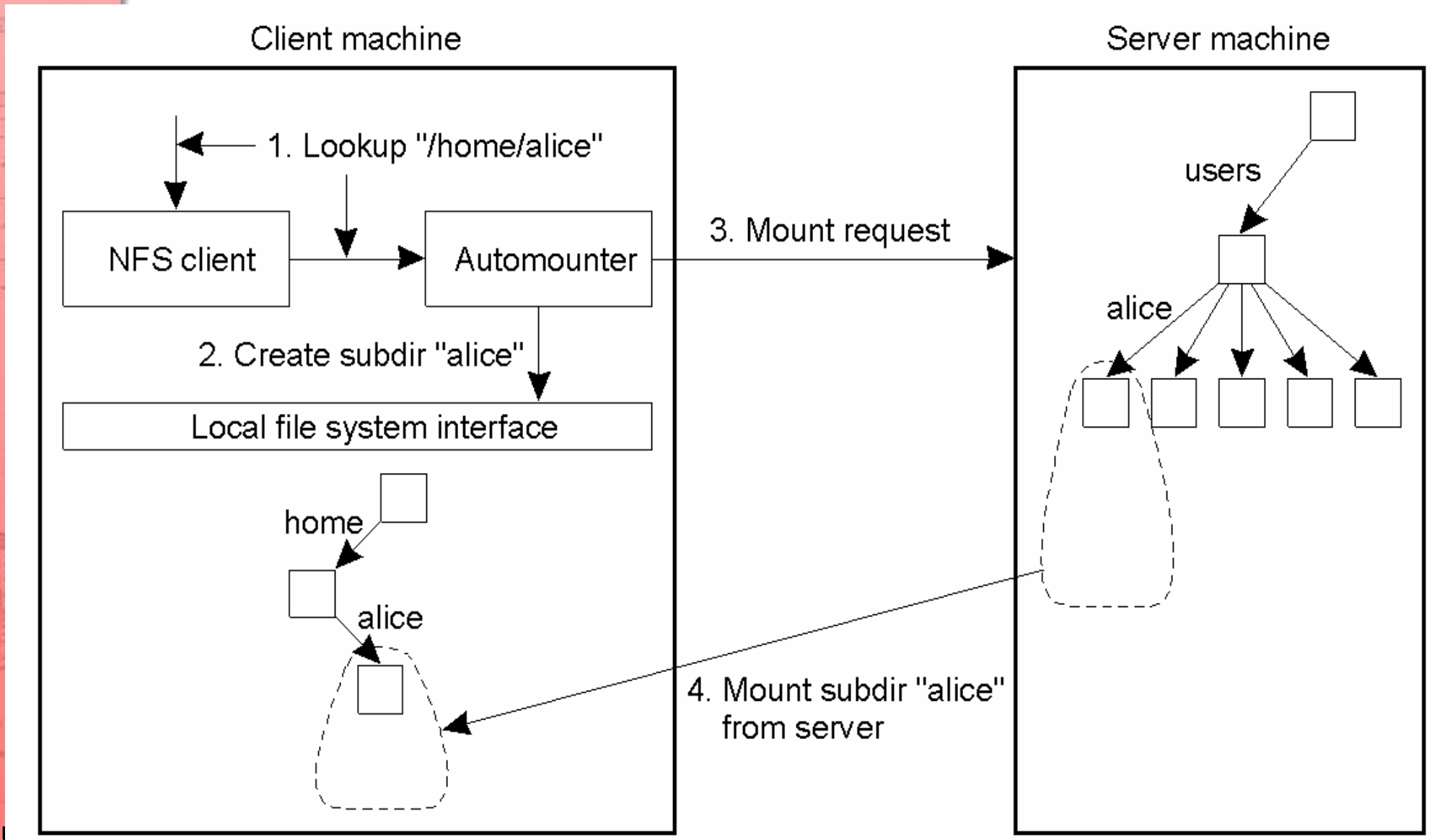


Naming (II)

- Mounting nested directories from multiple servers in NFS.



Automounting



File Attributes (I)

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.

File Attributes (II)

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.

File Locking in NFS (I)

Operation	Description
Lock	Creates 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 version 4 operations related to file locking.

File Locking in NFS (II)

Current file denial state

Request access	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Succeed
WRITE	Succeed	Succeed	Fail	Succeed
BOTH	Succeed	Succeed	Succeed	Fail

(a)

Requested file denial state

Current access state	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Succeed
WRITE	Succeed	Succeed	Fail	Succeed
BOTH	Succeed	Succeed	Succeed	Fail

(b)

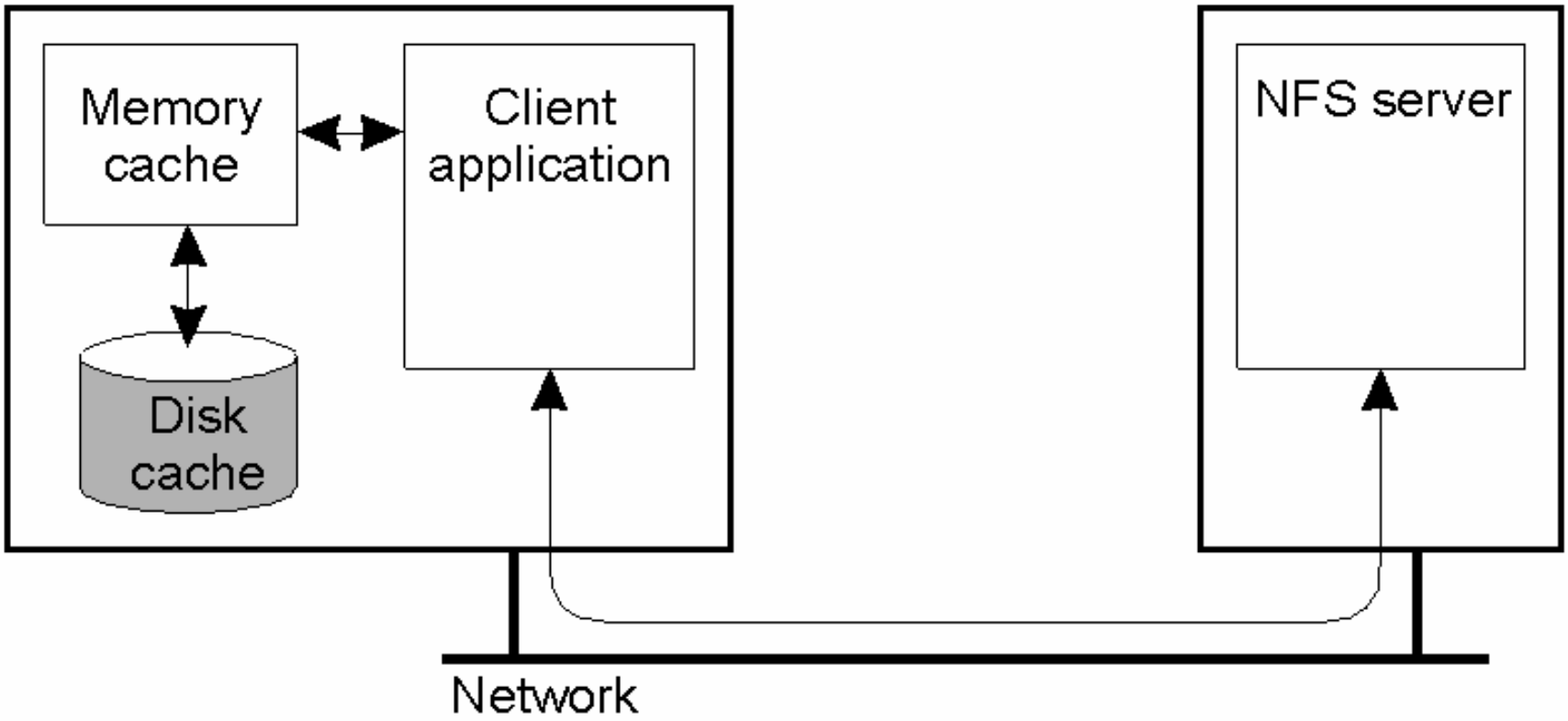
- The result of an *open* operation with share reservations in NFS v.4.

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.

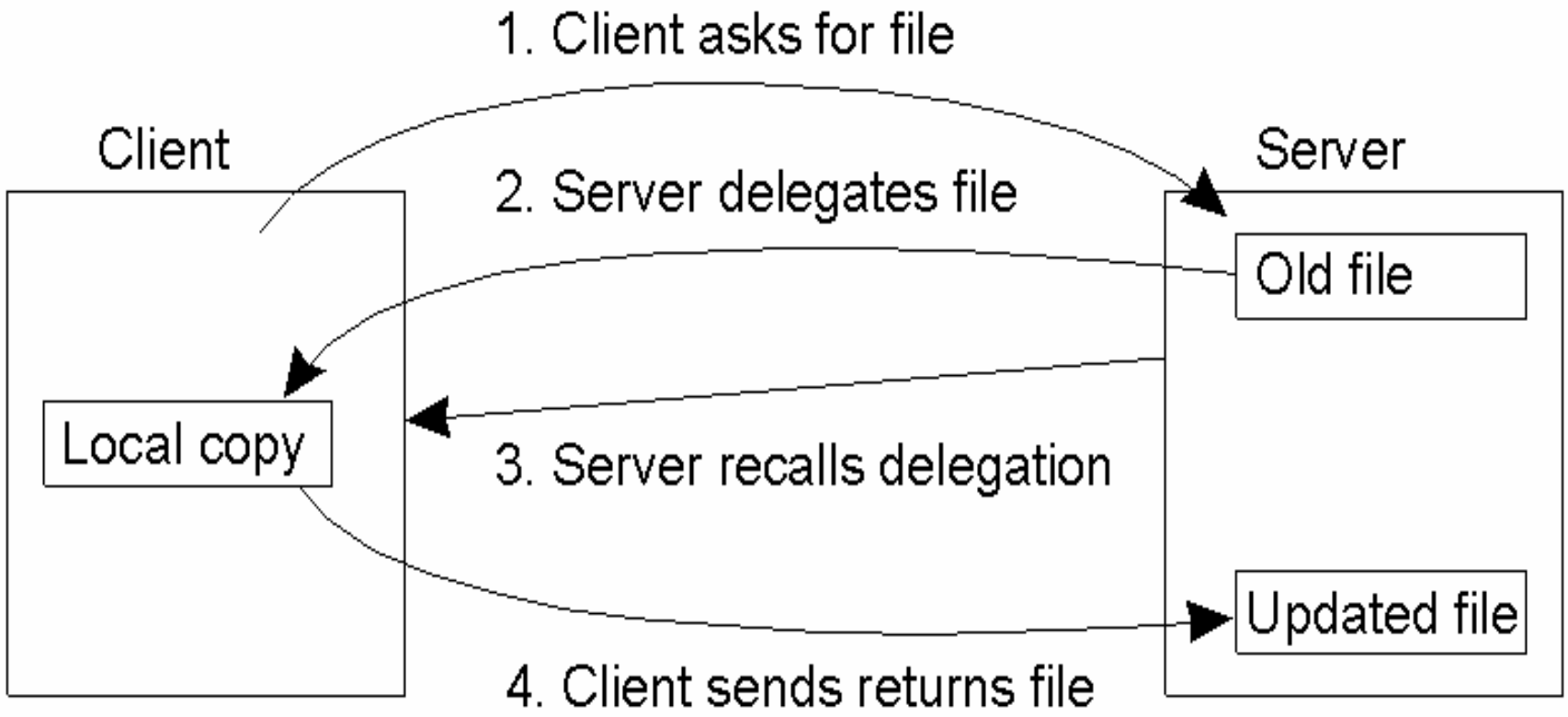
Client Caching (I)

- Client-side caching in NFS.

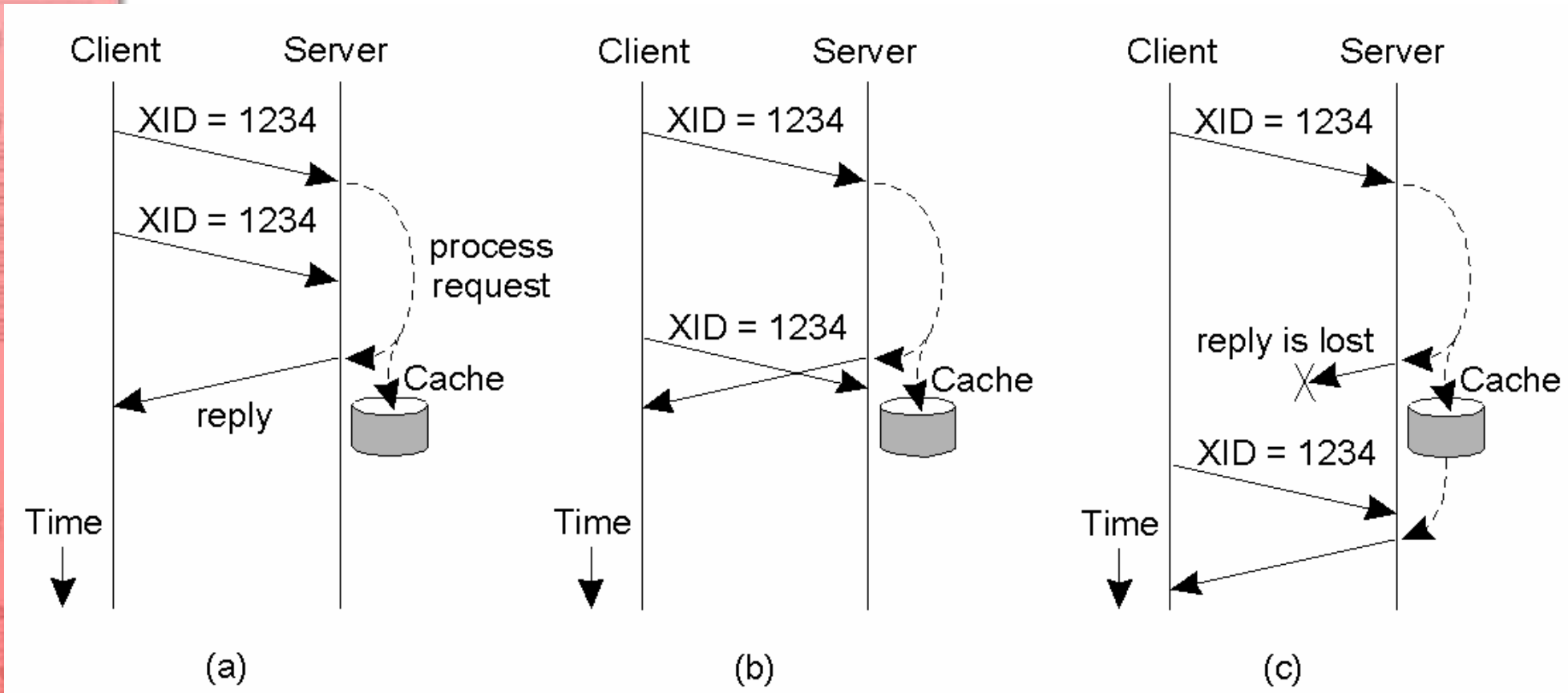


Client Caching (II)

- Using the NFS version 4 callback mechanism to recall file delegation.



RPC Failures



- Three situations for handling retransmissions.

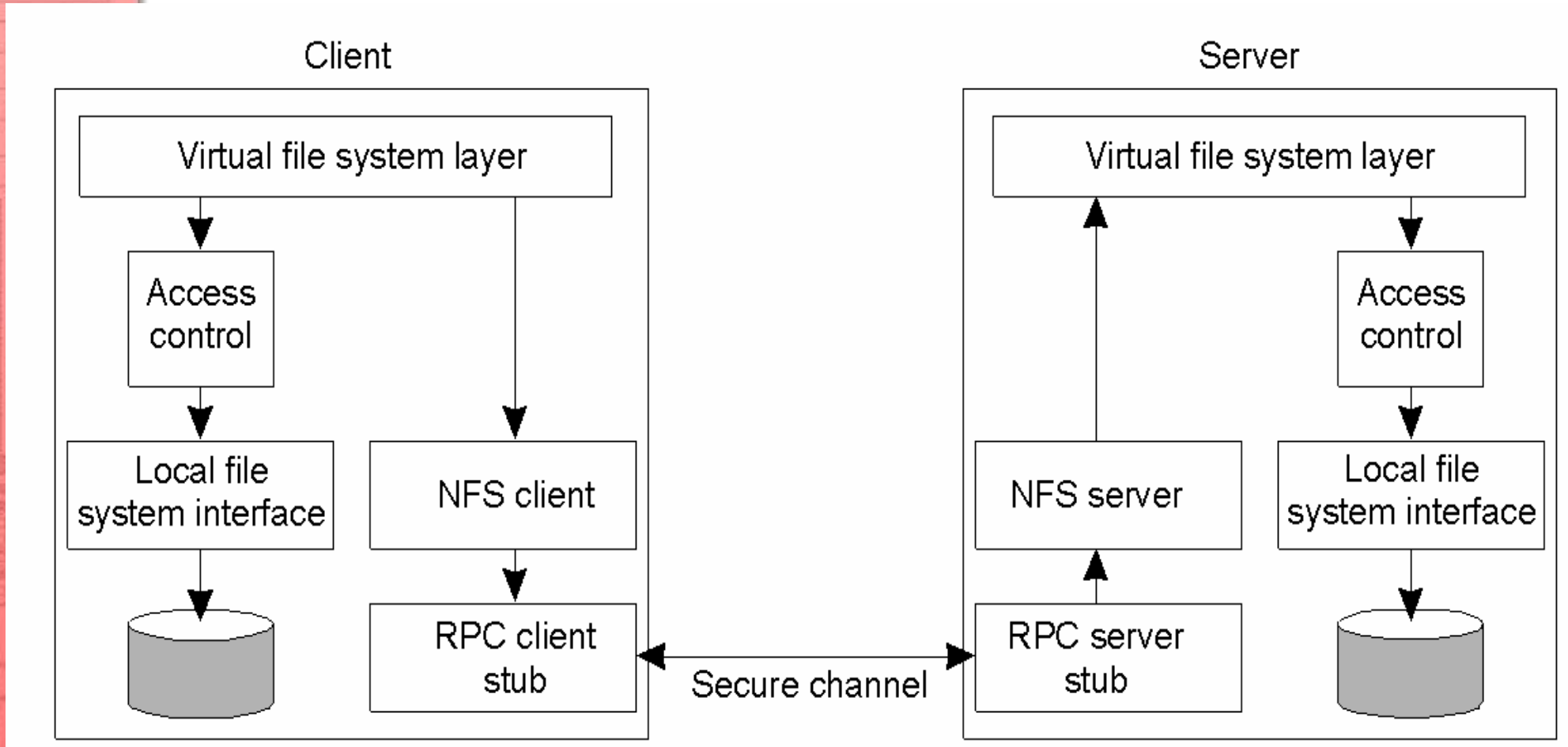
a) The request is still in progress

b) The reply has just been returned

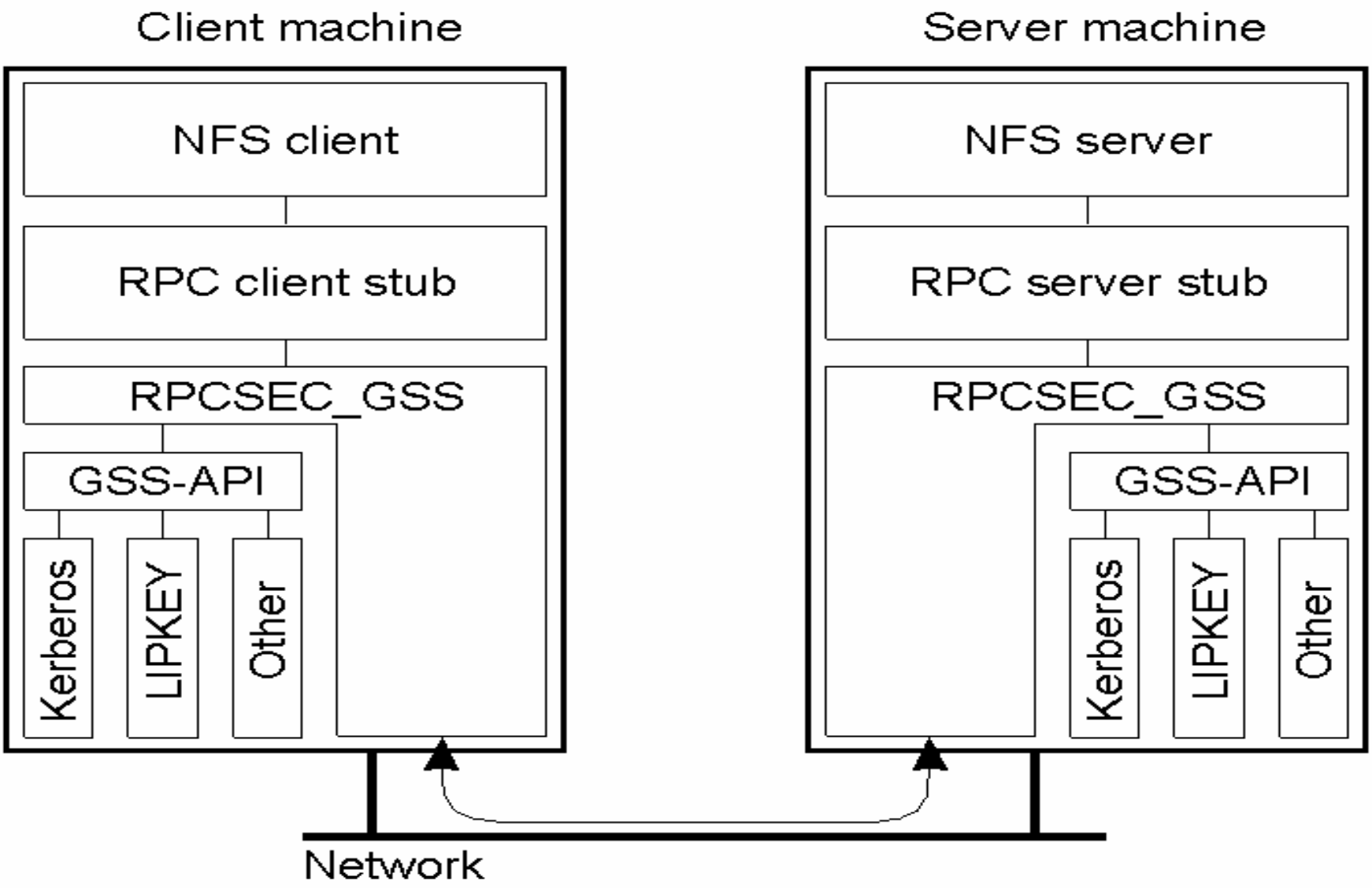
c) The reply has been sent some time ago, but was lost.

Security

- The NFS security architecture.



Secure RPCs (in NFS v.4)



Access Control

Operation	Description
Read_data	Permission to read the data contained in a file
Write_data	Permission to to modify a file's data
Append_data	Permission to to append data to a file
Execute	Permission to to execute a file
List_directory	Permission to to list the contents of a directory
Add_file	Permission to to add a new file t5o a directory
Add_subdirectory	Permission to to create a subdirectory to a directory
Delete	Permission to to delete a file
Delete_child	Permission to to delete a file or directory within a directory
Read_acl	Permission to to read the ACL
Write_acl	Permission to to write the ACL
Read_attributes	The ability to read the other basic attributes of a file
Write_attributes	Permission to to change the other basic attributes of a file
Read_named_attrs	Permission to to read the named attributes of a file
Write_named_attrs	Permission to to write the named attributes of a file
Write_owner	Permission to to change the owner
Synchronize	Permission to to access a file locally at the server with synchronous reads and writes

File access patterns

- File size distribution is skewed
  - Toward small sizes
- Reads are much more frequent than writes
- Random access is rare
- Once opened, files are usually read in their entirety
- Files are more frequently overwritten than selectively updated
- Many files are used by only one user
- Most shared files are used by only one user at a time
- When shared, there is usually only one writer per file
- High locality in file references

Andrew File System (AFS)

- Developed at CMU (~1985), commercial product by Transarc (part of OSF/DCE)
- Segment network into clusters, with one file server per cluster
- Dynamic reconfigurations to balance load
- Stateful protocol + aggressive caching
  - Servers participate in client cache management
- Entire files are cached
- Session semantics
  - Weaker than UNIX semantics
  - See new data on next open()
  - Immediate updates of metadata

Interception and Caching

- fd = open(pathname)
  - Files in local file system are opened as normal.
  - Cached files in shared file system are opened locally.
  - Other shared files are copied to cache.
- All read/write directed to cached copy.
- close(fd)
  - Local or cached copy is closed.
  - If shared file is modified it is copied back to server.

Vice File Service

- Flat file service with volumes
- 96 bit file identifier

volume number	file handle	uniquifier

- Volume
  - Group of related files (e.g. one user's files)
  - Used for location and management
  - Server is custodian of volume
- Volume location database replicated at each server

Cache Consistency - Call Back

- Callback promise sent to Venus with opened file
- Server invalidates callback promise (RPC) when file updated
- Venus confirms call back on opening cached file
- Must validate promises after “down” period
- Introduces client state held at server

Vice Interface

Fetch(fid) -> attr, data	**Returns the attributes (status) and, optionally, the contents of file identified by the *fid* and records a callback promise on it.**
Store(fid, attr, data)	**Updates the attributes and (optionally) the contents of a specified file.**
Create() -> fid	**Creates a new file and records a callback promise on it.**
Remove(fid)	**Deletes the specified file.**
SetLock(fid, mode)	**Sets a lock on the specified file or directory. The mode of the lock may be shared or exclusive. Locks that are not removed expire after 30 minutes.**
ReleaseLock(fid)	**Unlocks the specified file or directory.**
RemoveCallback(fid)	**Informs server that a Venus process has flushed a file from its cache.**
BreakCallback(fid)	**This call is made by a Vice server to a Venus process. It cancels the callback promise on the relevant file.**

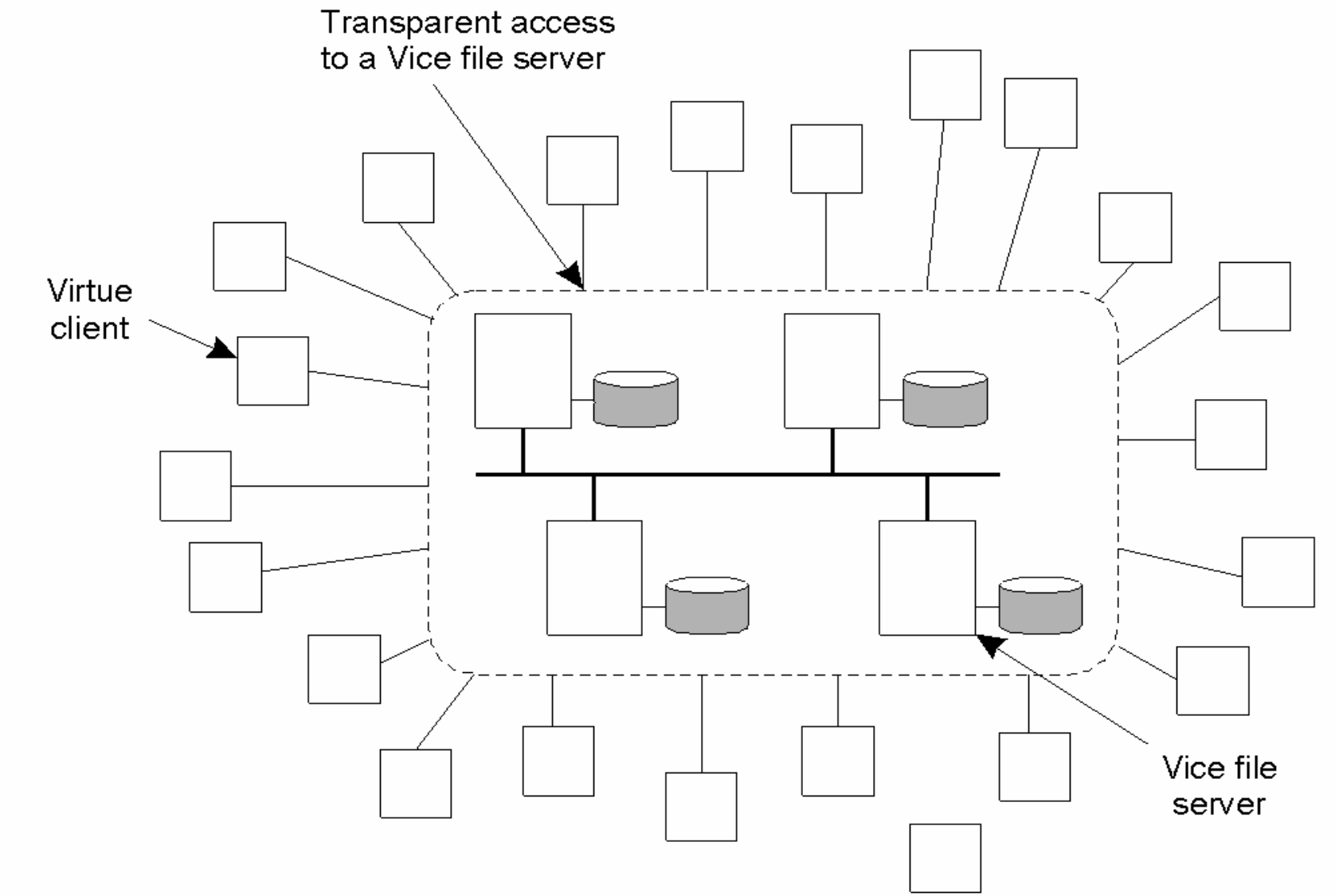
AFS properties

- Advantages
  - Only contact server on open/close
  - Usually single user at one workstation
  - Most files are read in entirety
  - Cache copies valid for long periods
  - Disk based cache survives reboot
  - Simplified caching scheme
- Disadvantages
  - Files larger than cache can’t be opened
  - Workstations require disk
  - Not appropriate for database support

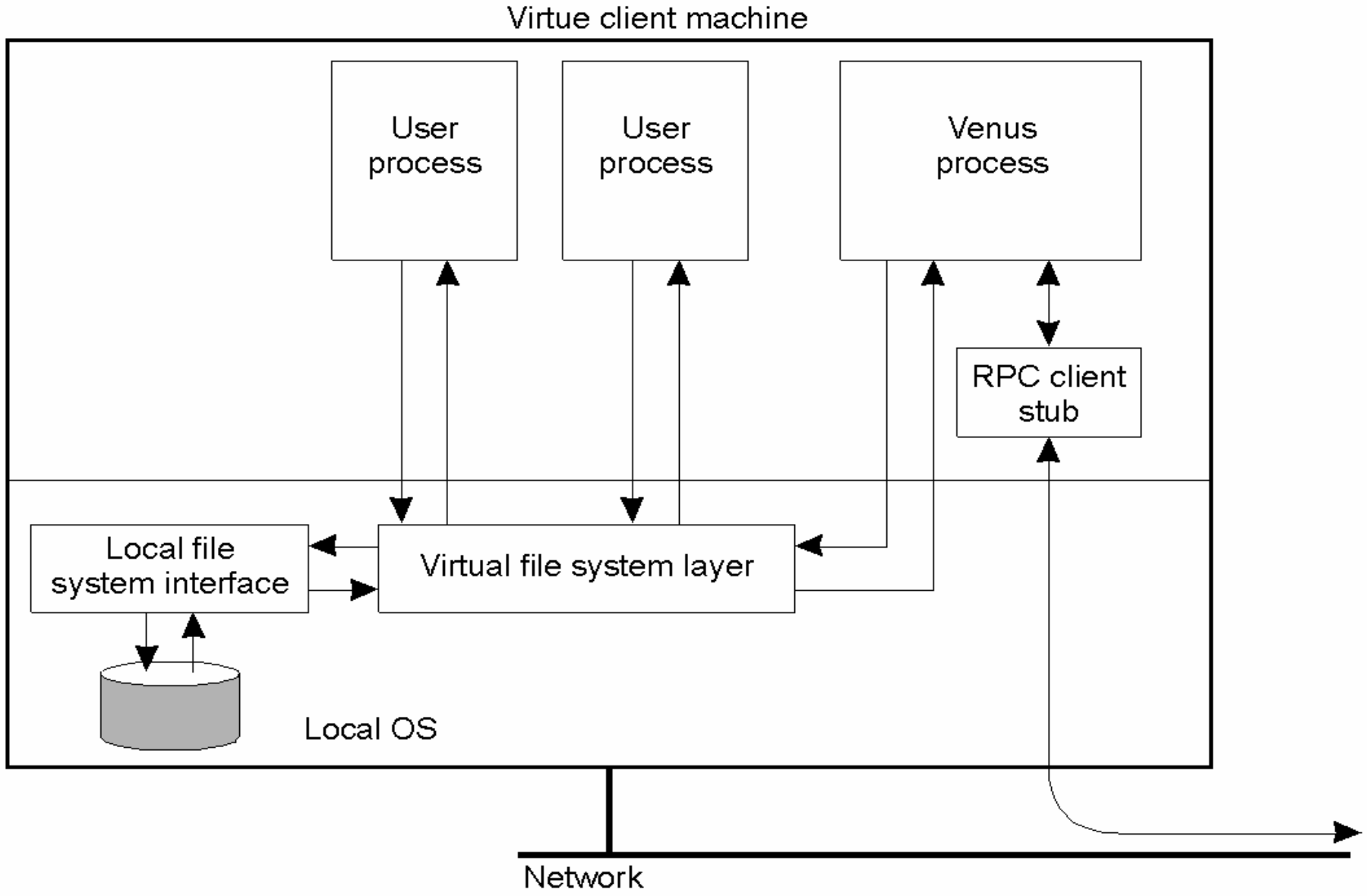
The Coda File System

Type of user	Description
Owner	The owner of a file
Group	The group of users associated with a file
Everyone	Any user of a 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 a batch job
Anonymous	Anyone accessing the file without authentication
Authenticated	Any authenticated user of a process
Service	Any system-defined service process

Overview of Coda (I)

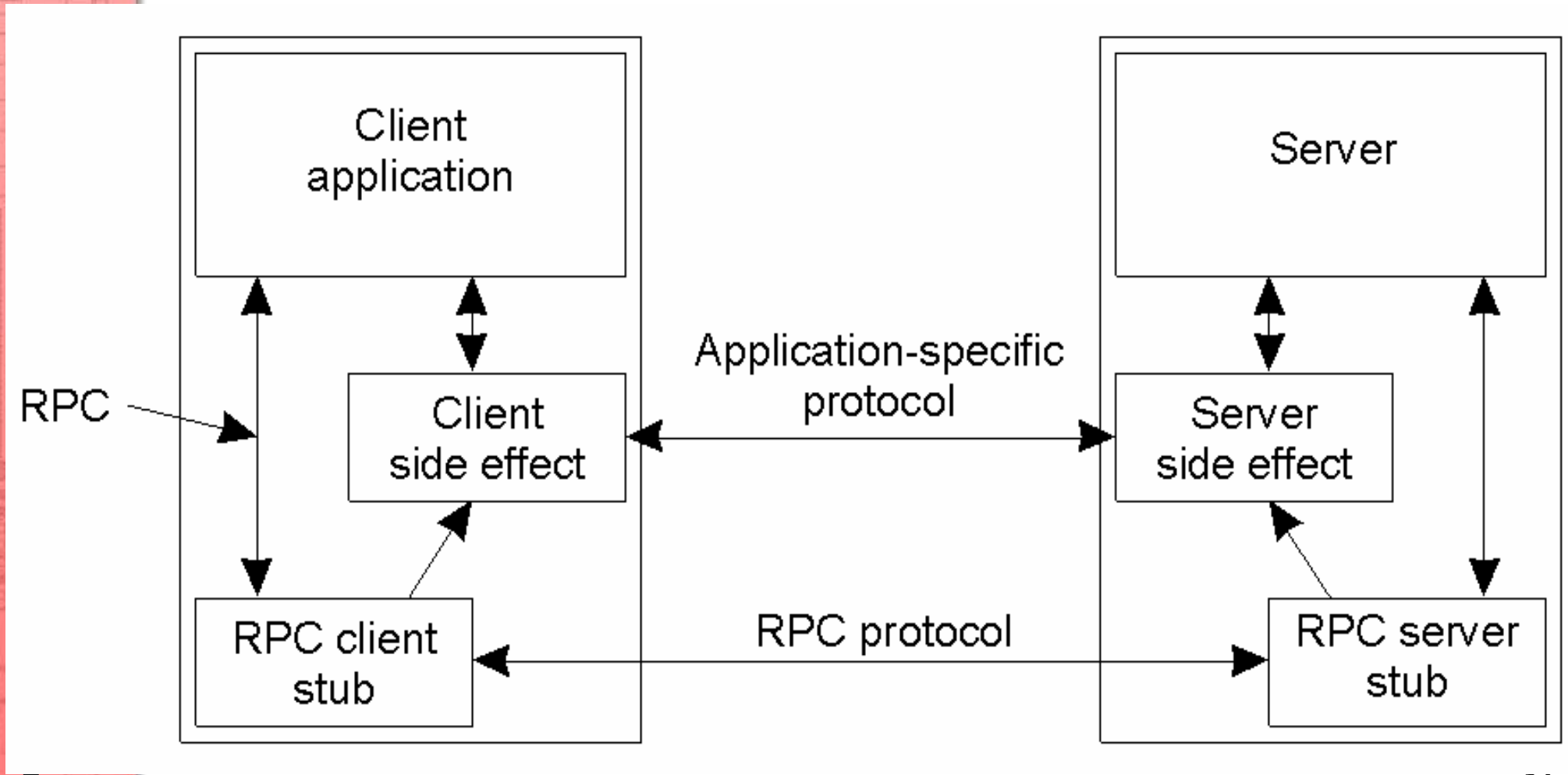


Overview of Coda (II)

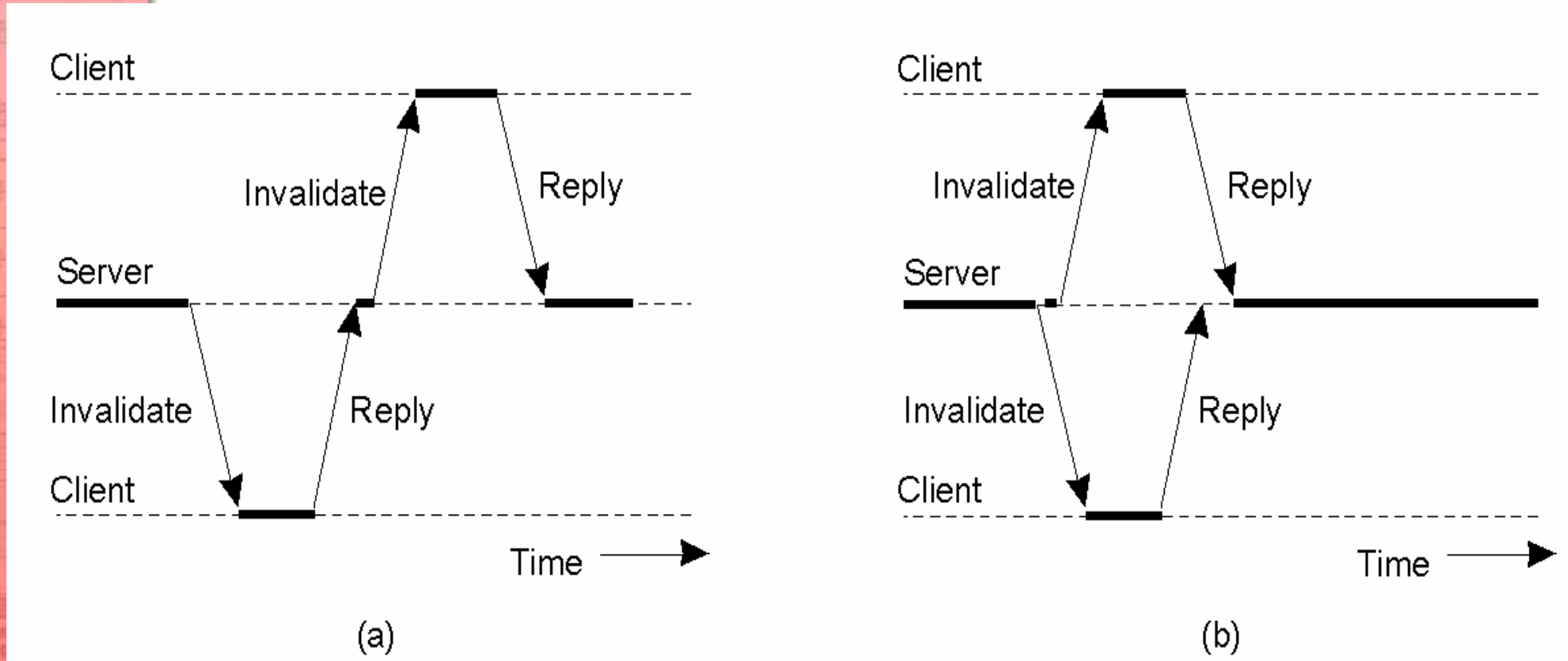


Communication (I)

- Side effects in Coda's RPC2 system.



Communication (II)

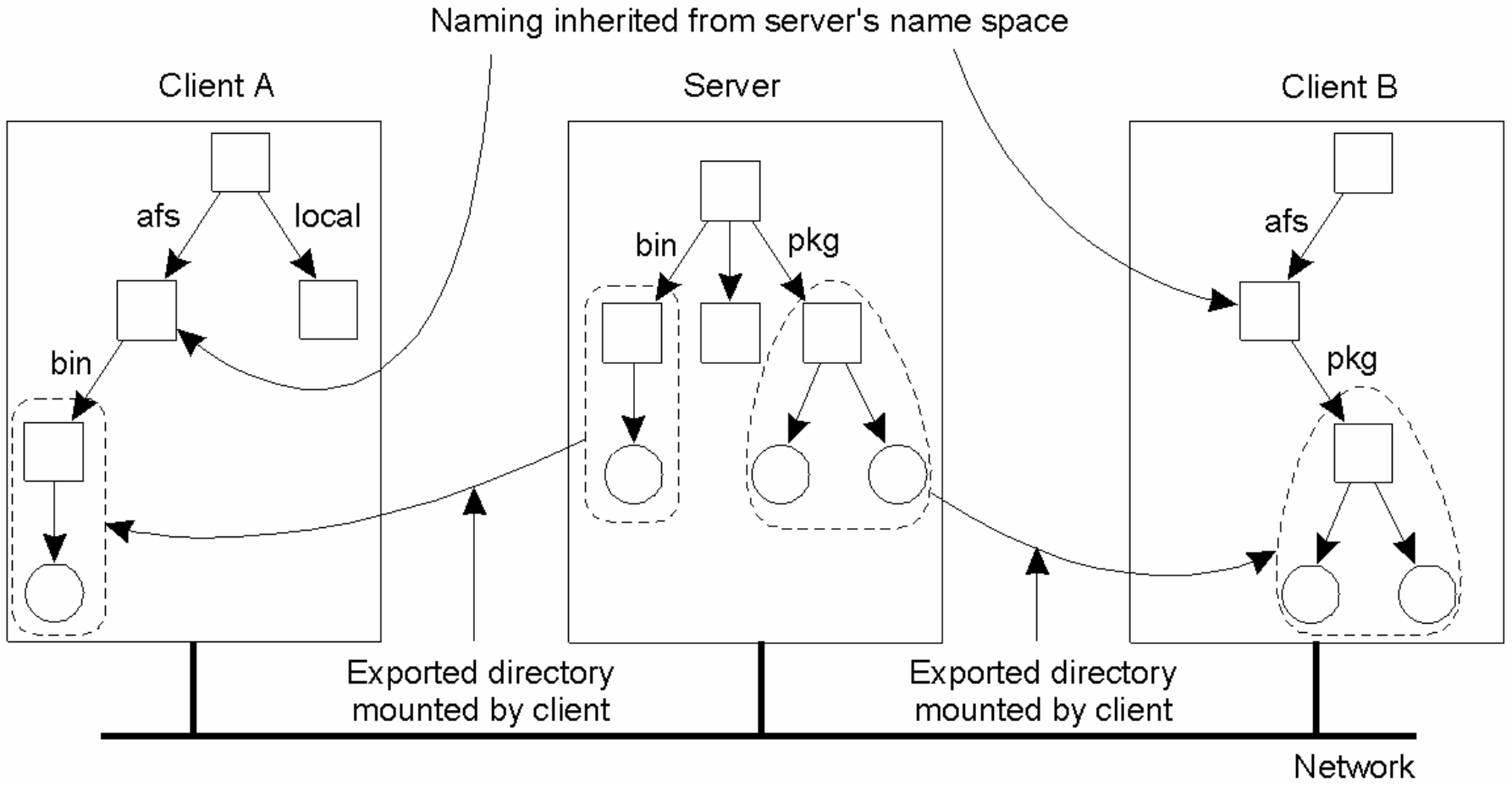


a) Sending an invalidation message one at a time.

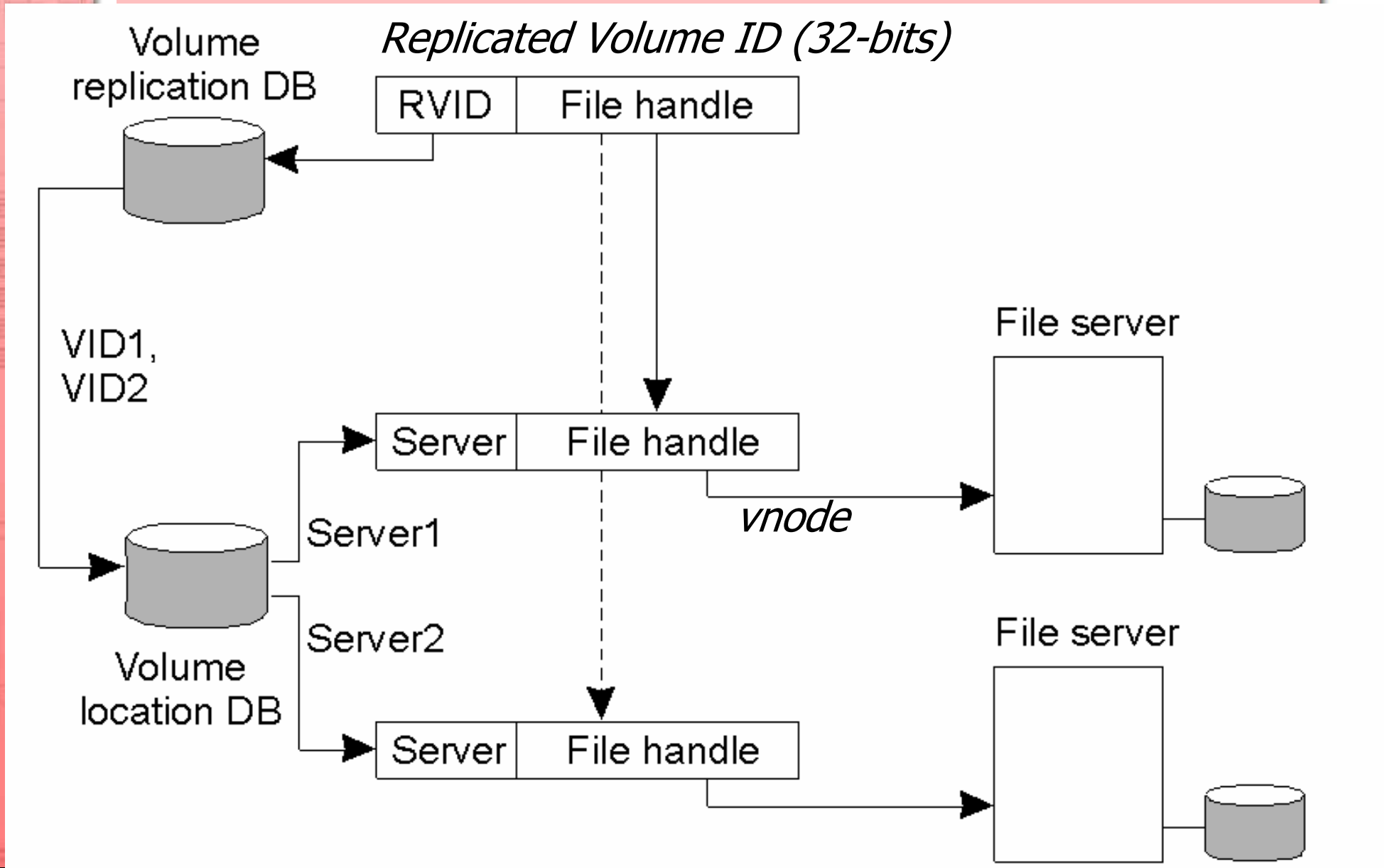
b) Sending invalidation messages in parallel.

Naming

- Clients in Coda have access to a single shared name space.

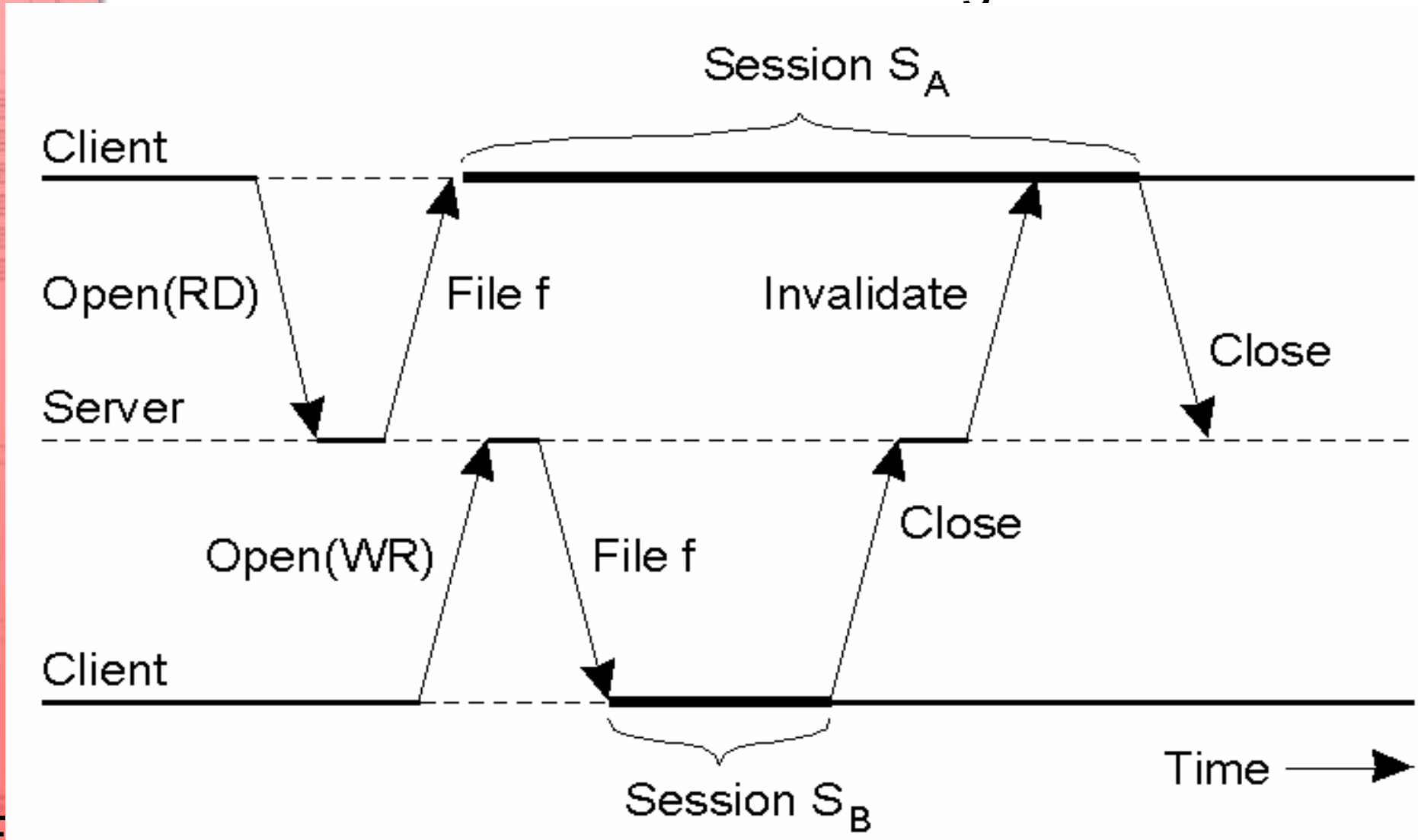


File Identifiers



Sharing Files in Coda

- Transactional behavior in sharing files in Coda.

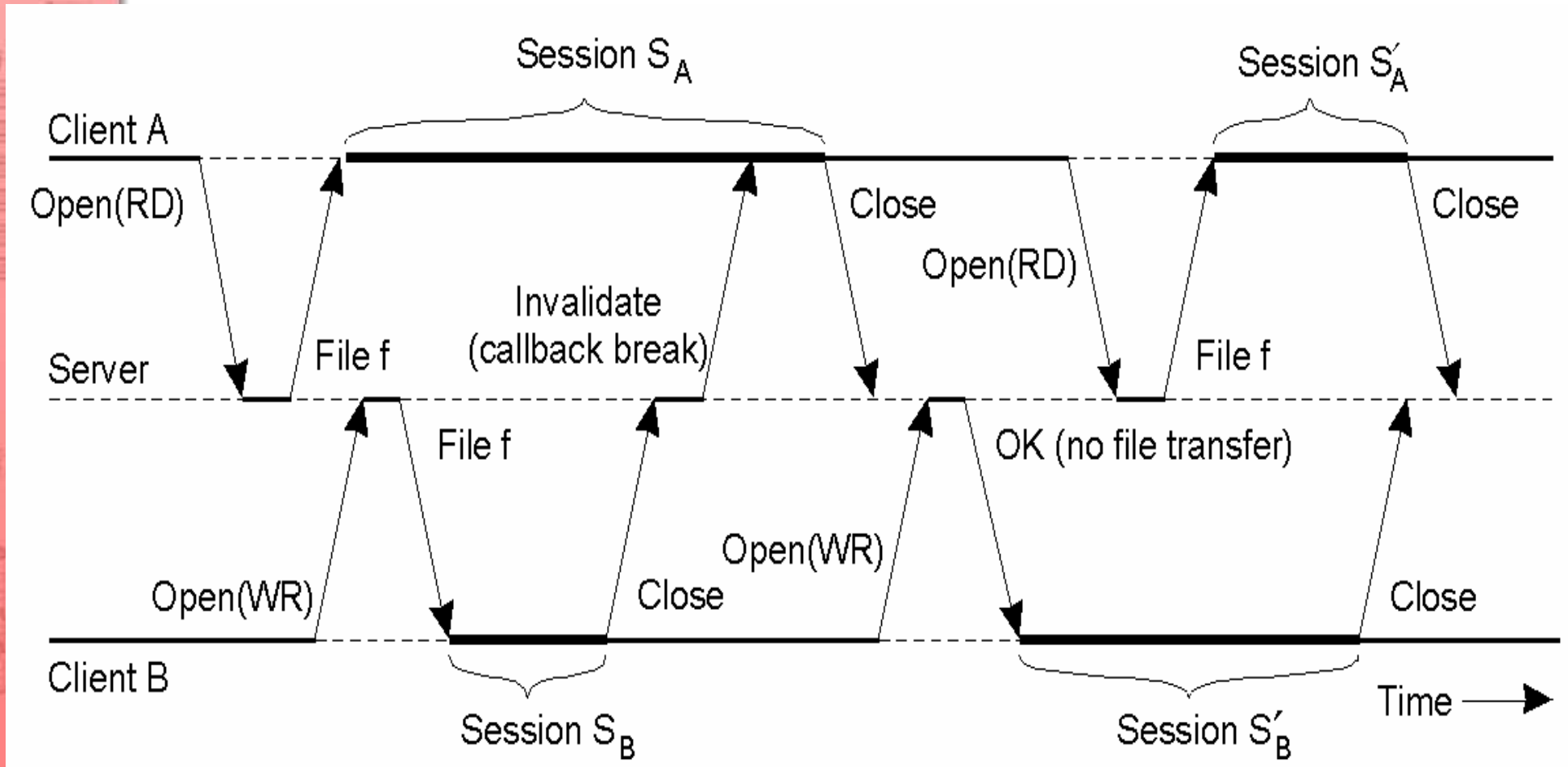


Transactional Semantics

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

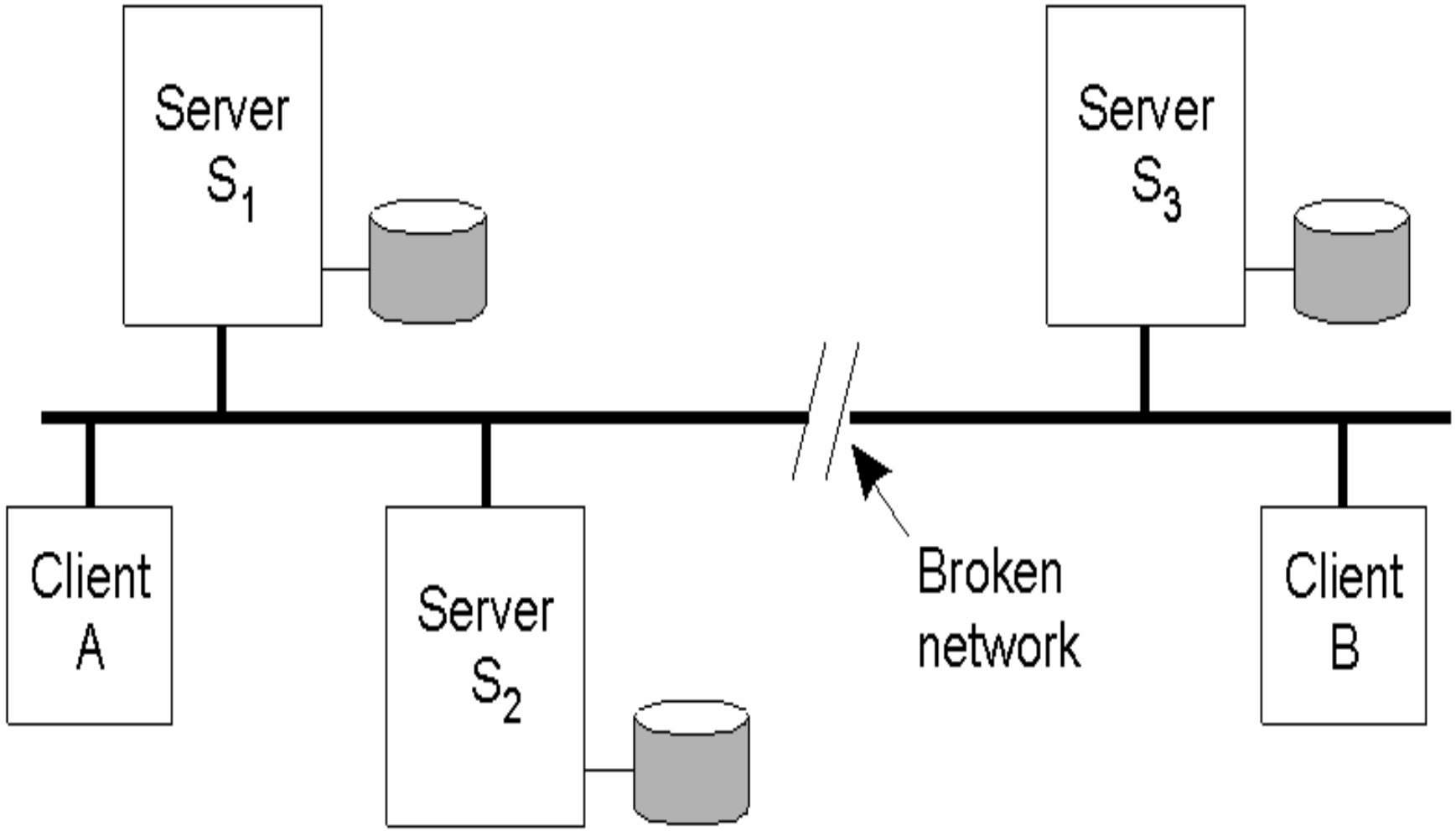
- Metadata read & modified for a *store* session type in Coda.

Client Caching



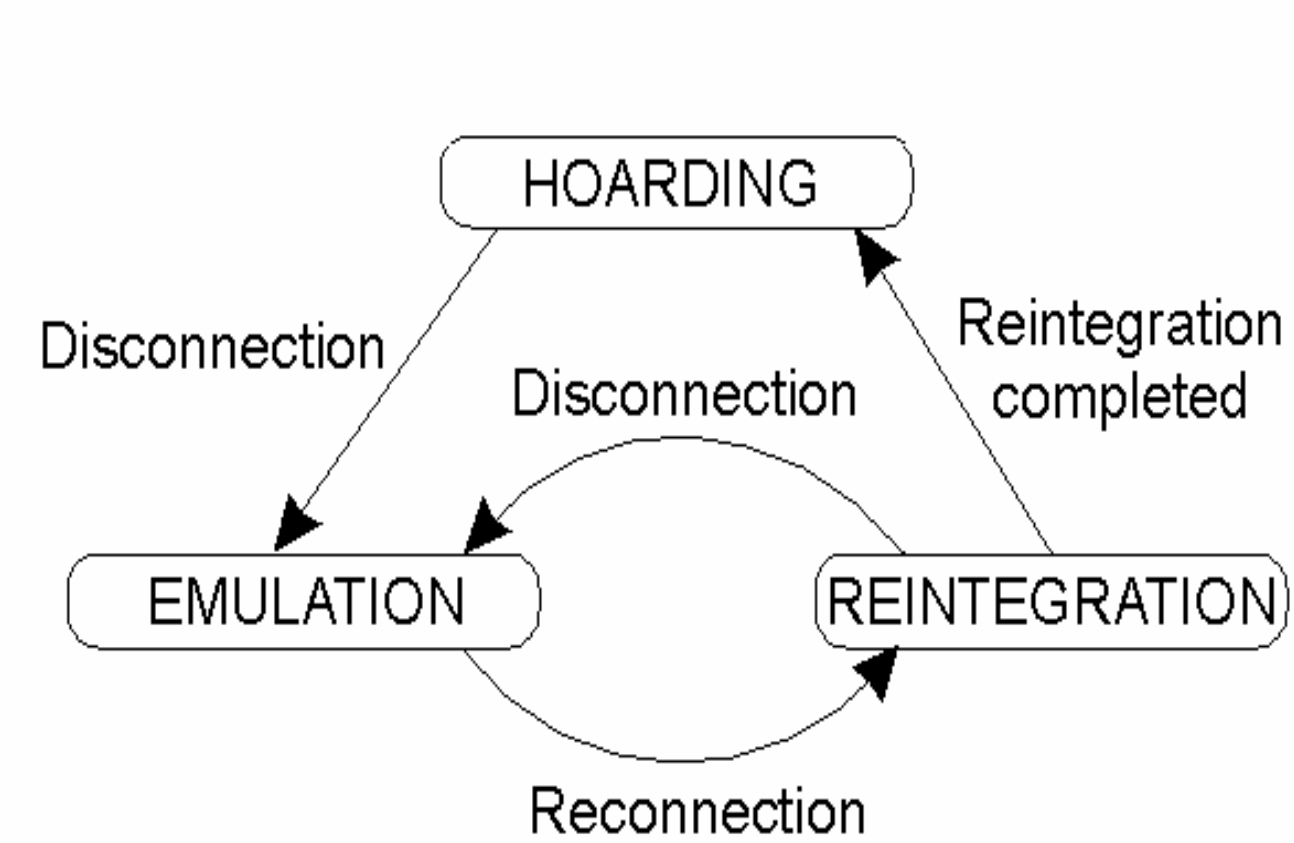
Server Replication

Version # per file



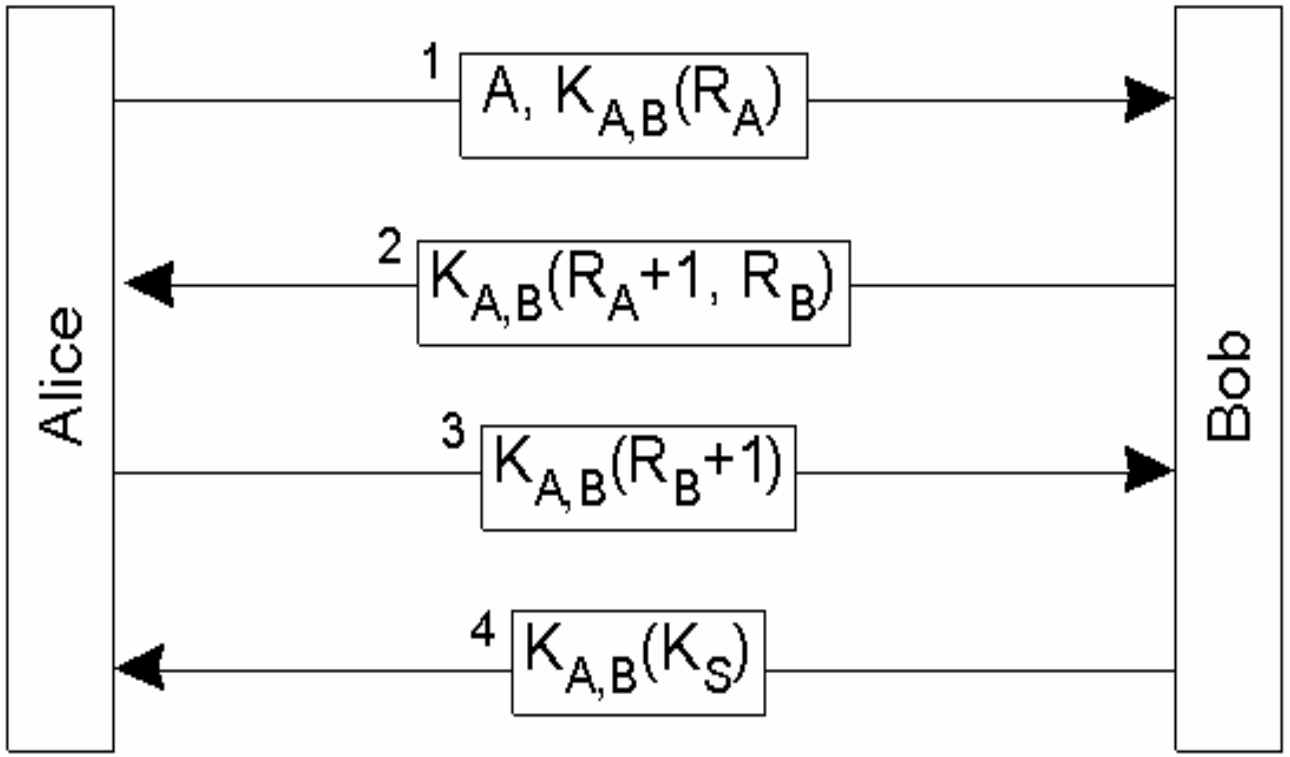
Disconnected Operation

- The state-transition diagram of a Coda client wrt a volume.

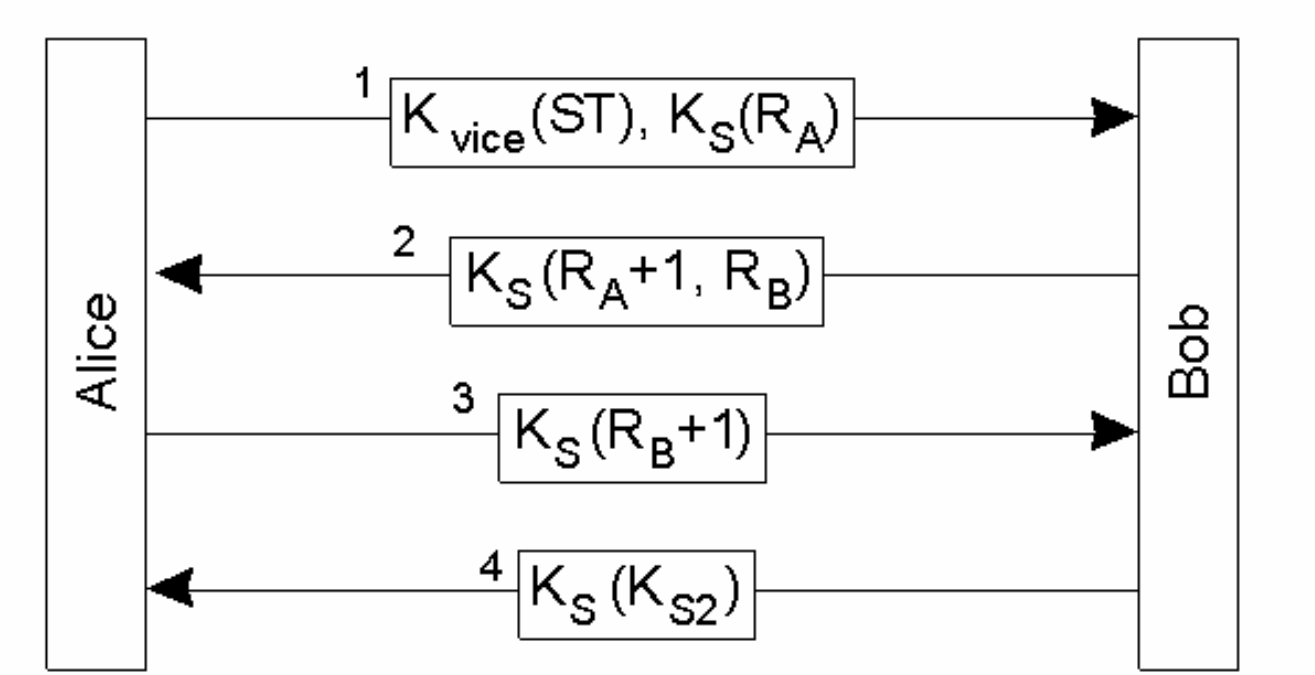


Secure Channels (I)

- Mutual authentication in RPC2.



Secure Channels (II)



- Setting up a secure channel between a (Venus) client & a (Vice) server in Coda.

Access Control

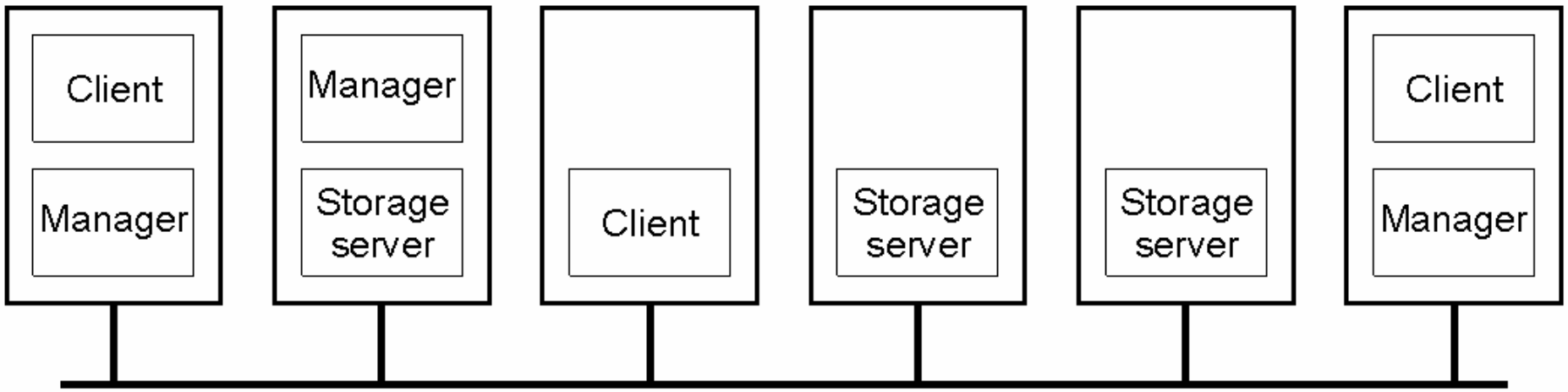
Operation	Description
Read	Read any file in the directory
Write	Modify any file in the directory
Lookup	Look up the status of any file
Insert	Add a new file to the directory
Delete	Delete an existing file
Administer	Modify the ACL of the directory

- Classification of file & directory operations recognized by Coda wrt access control.

xFS: A “Serverless” File System

- Distribute file server processing across a set of available hosts, at the granularity of individual files
  - Separate file management & file storage/access
  - Dynamically assign files to hosts
- Software RAID storage system
  - Striping file data across disks
  - Log-structured organization
- Manager Map
  - Replicated at all hosts

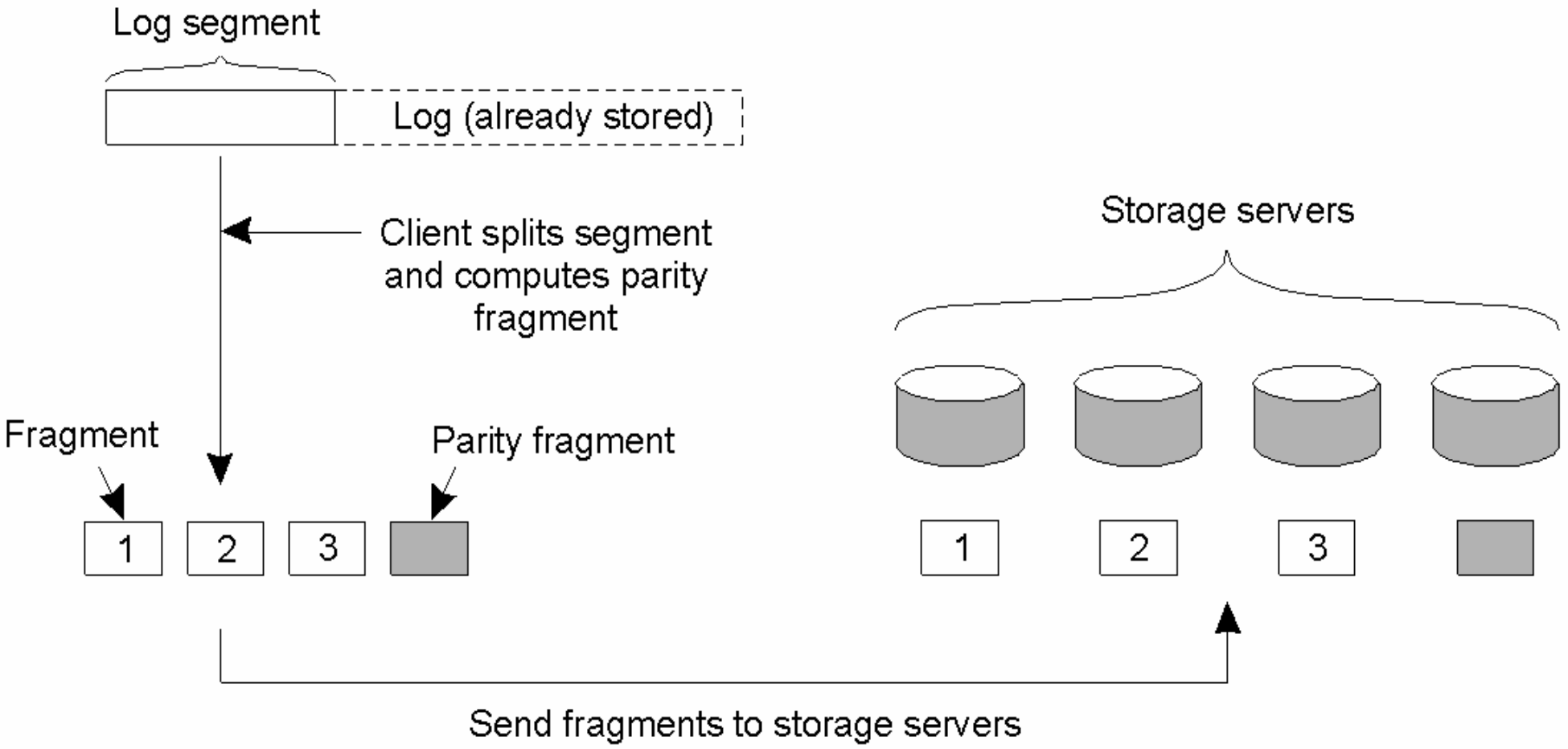
Overview of xFS



- A typical distribution of xFS processes across multiple machines.

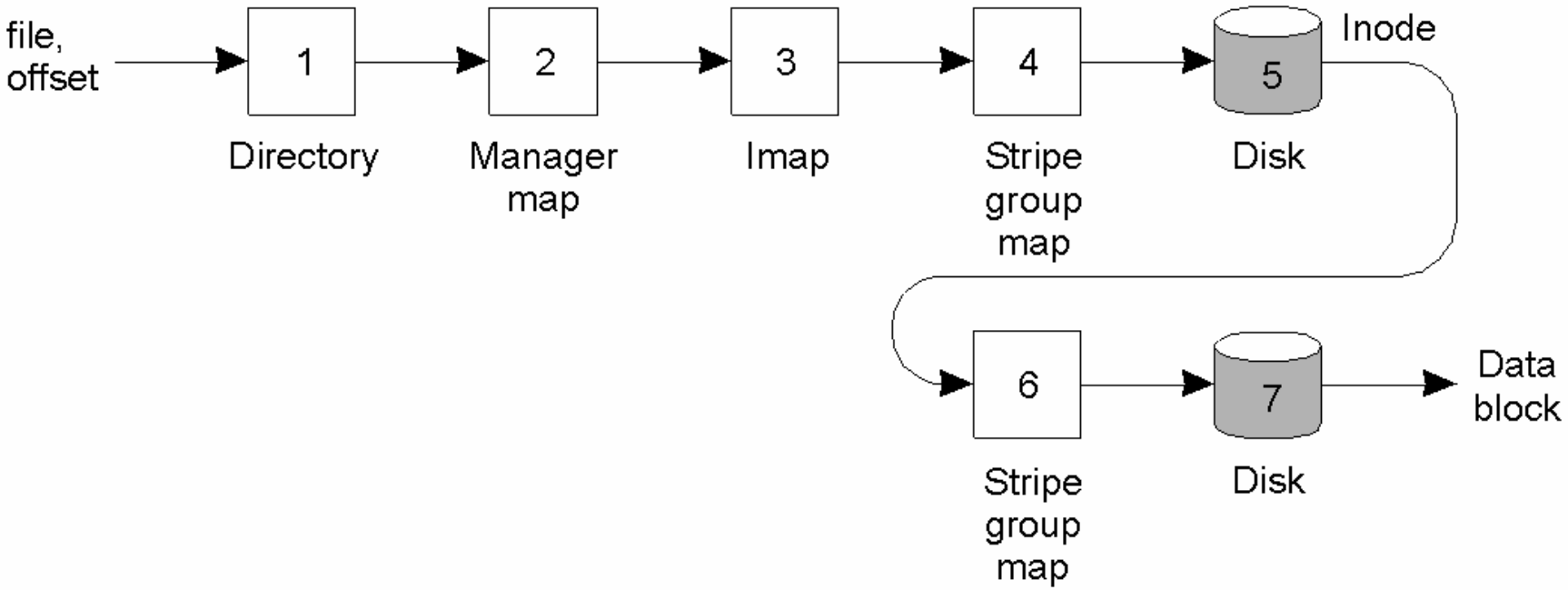
Processes (I)

- The principle of log-based striping in xFS.



Processes (II)

- Reading a block of data in xFS.



Naming

- Main data structures used in xFS.

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

DAFS: Direct Access File System

- Allow clusters of application servers to share data without the overhead of a general-purpose OS
- "Local" file sharing
  - Usually within a data center
  - Small number of file servers
    - Access over a separate high-performance interconnect
    - Table of "partners" & authenticated "clients"
  - Intense sharing of individual files
    - High-performance file & record locking
    - Lock caching & on-demand transfer
- Based on Virtual Interface (VI) transport

Key drivers for networked storage

- Exponential growth of storage requirements
- … AND rate of accumulation
  - Mail.com -> 27 TB in 45 days (end of 2000)
- Advances (mainly in B/W) in interconnects
- Externalization of storage onto the network
- Shortage of IT staff & ever increasing costs of ownership/management

RAID Technology

- Redundant Arrays of Inexpensive Disks
- ➢ A RAID controller acts as an intelligent SCSI I/O port
- ➢ Benefits include ECC memory, battery backup and other fault tolerant features not available in non-intelligent SCSI I/O port controllers

RAID Levels:

RAID-0 → striping

RAID-1 → mirroring

RAID-3 → striping + dedicated parity disk

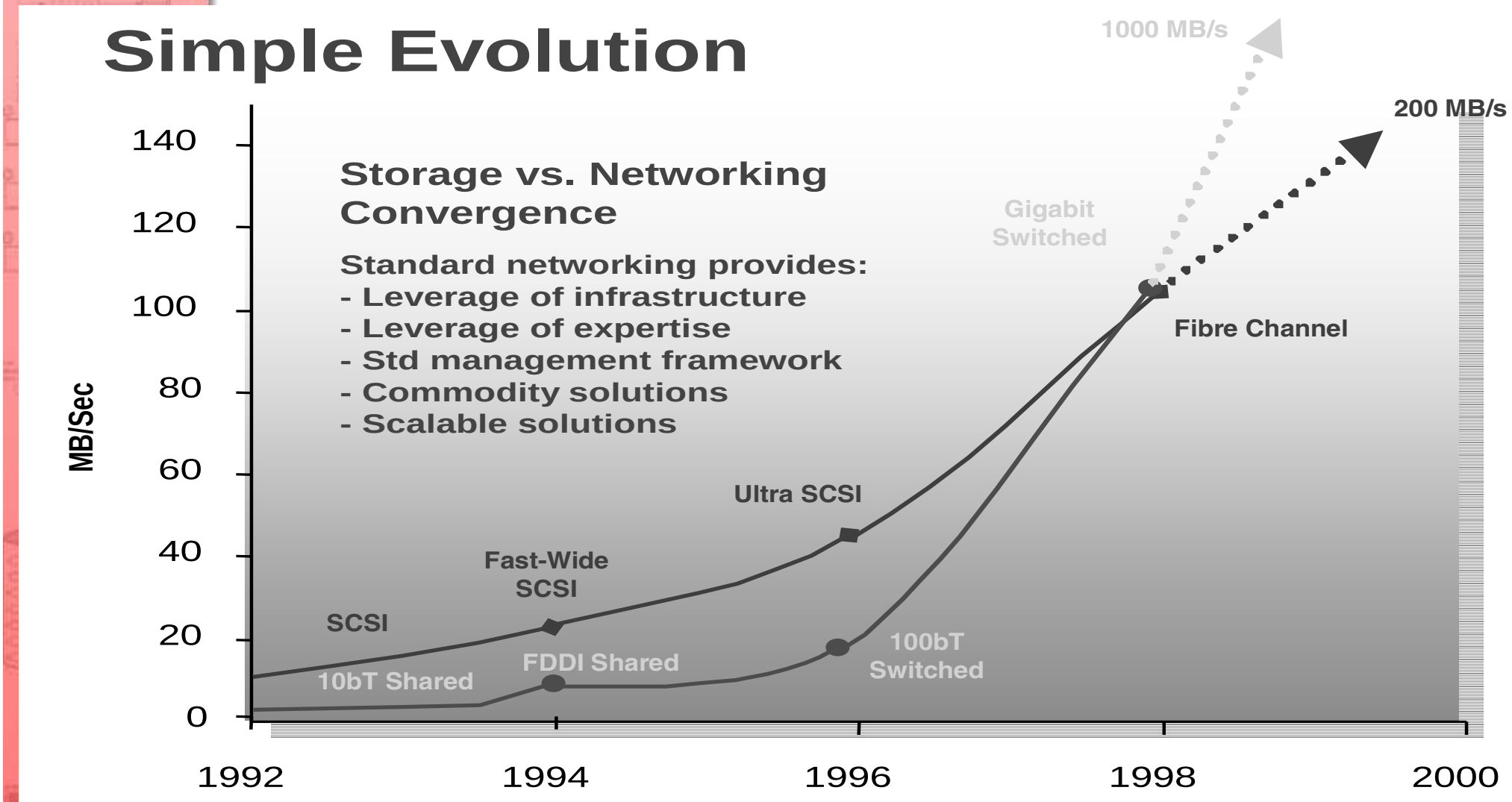
RAID-5 → striping + rotational parity

RAID-10.30.50 (multi-layer configurations)

Storage Virtualization

- Separate storage system implementation from host's view of storage
  - Make interconnect & data location invisible to hosts
  - Allow substantial changes within the storage system to be invisible to applications & the host environment
    - Allow data location to change without consequences to hosts
- Standards for managing virtualized storage
- Existing products:
  - Logical data managers
  - Network/Enterprise management systems
  - Storage network systems
  - Device managers
    - Switch, Storage

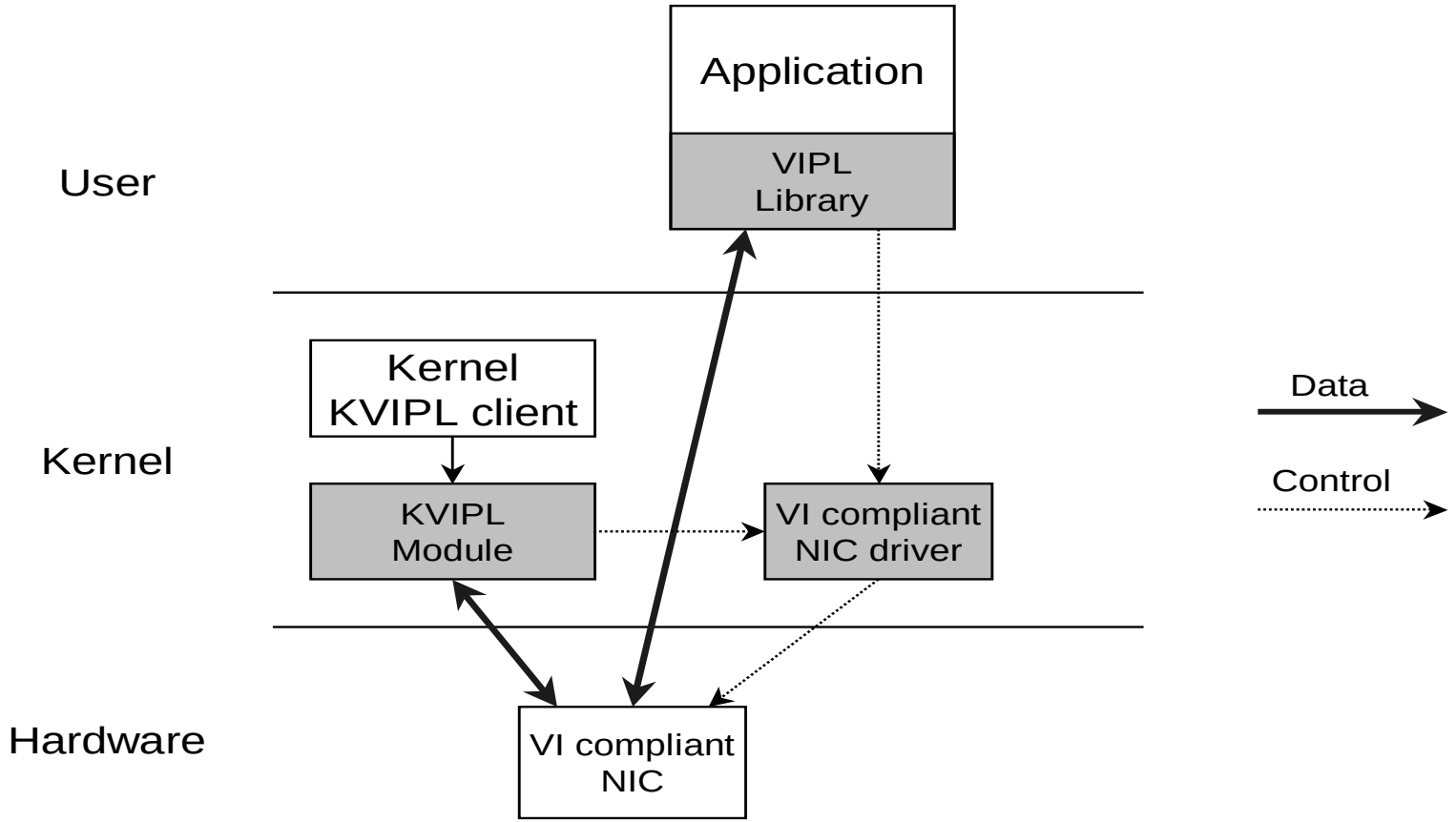
Storage vs Interconnect Evolution (source: B. Pawlowski, June 2001)



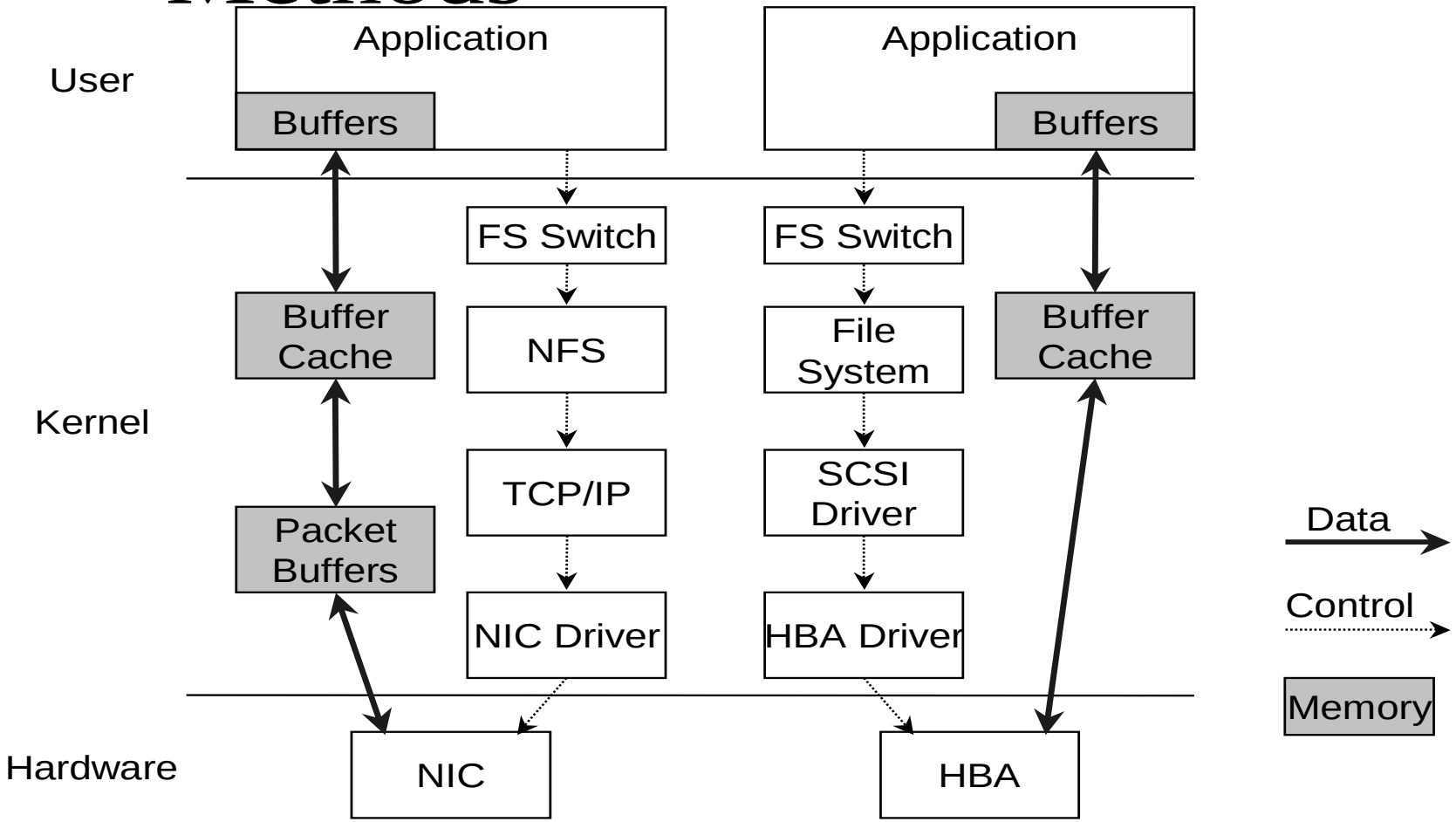
Virtual Interface (VI)

- Initiative by Intel, Compaq, Microsoft
  - "networked blocks of shared memory"
  - Standard for cluster interconnection
    - Independent of underlying networking technology
- Direct memory-to-memory transfers
- Direct application access
  - Queues of transfer operations
  - Directly write data to receiver's address space
    - Without OS involvement
- Optimized for high-bandwidth, low-latency interconnect, not for general WAN !

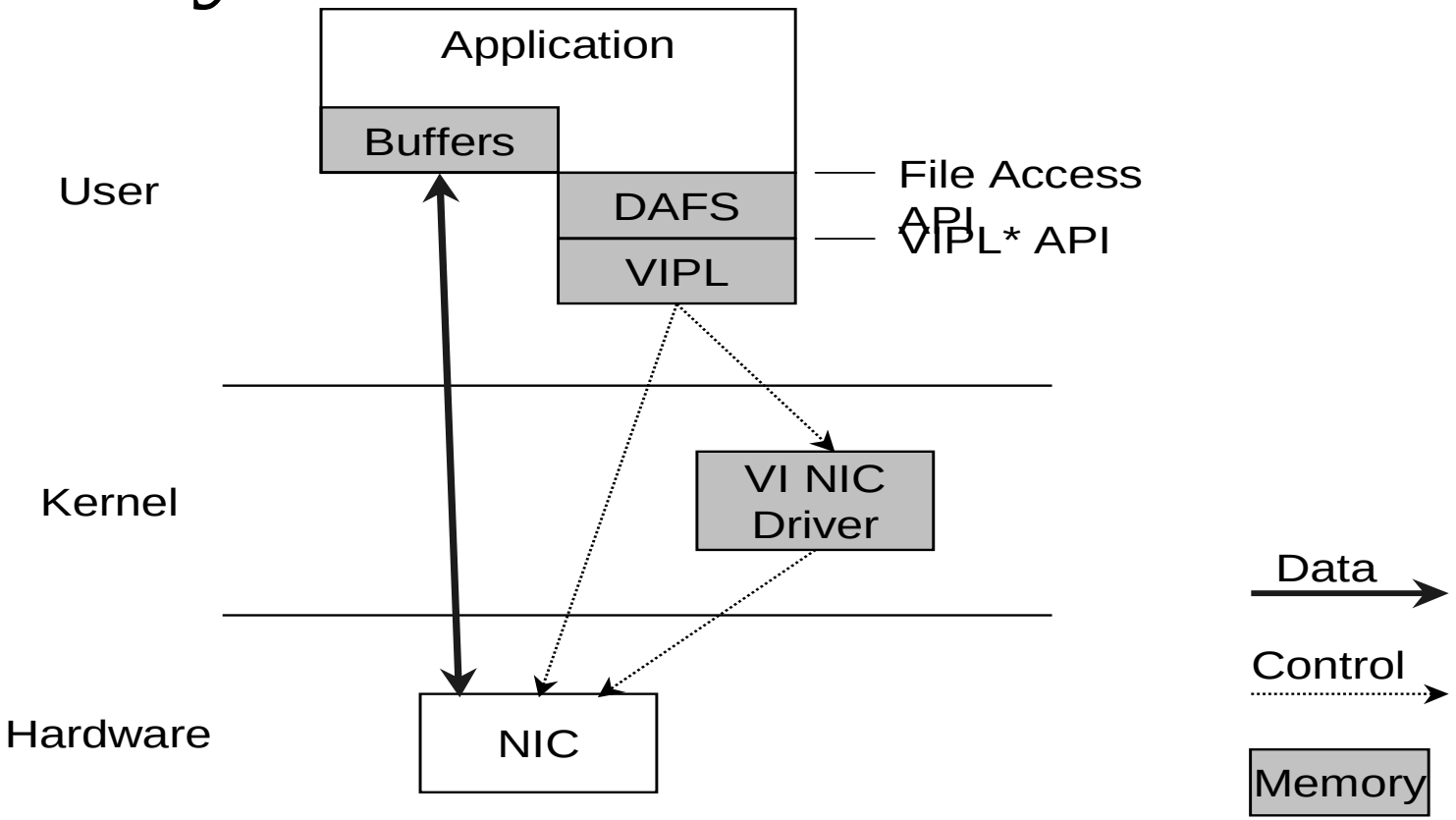
VI Architecture



Conventional File Access Methods



Direct Access File System



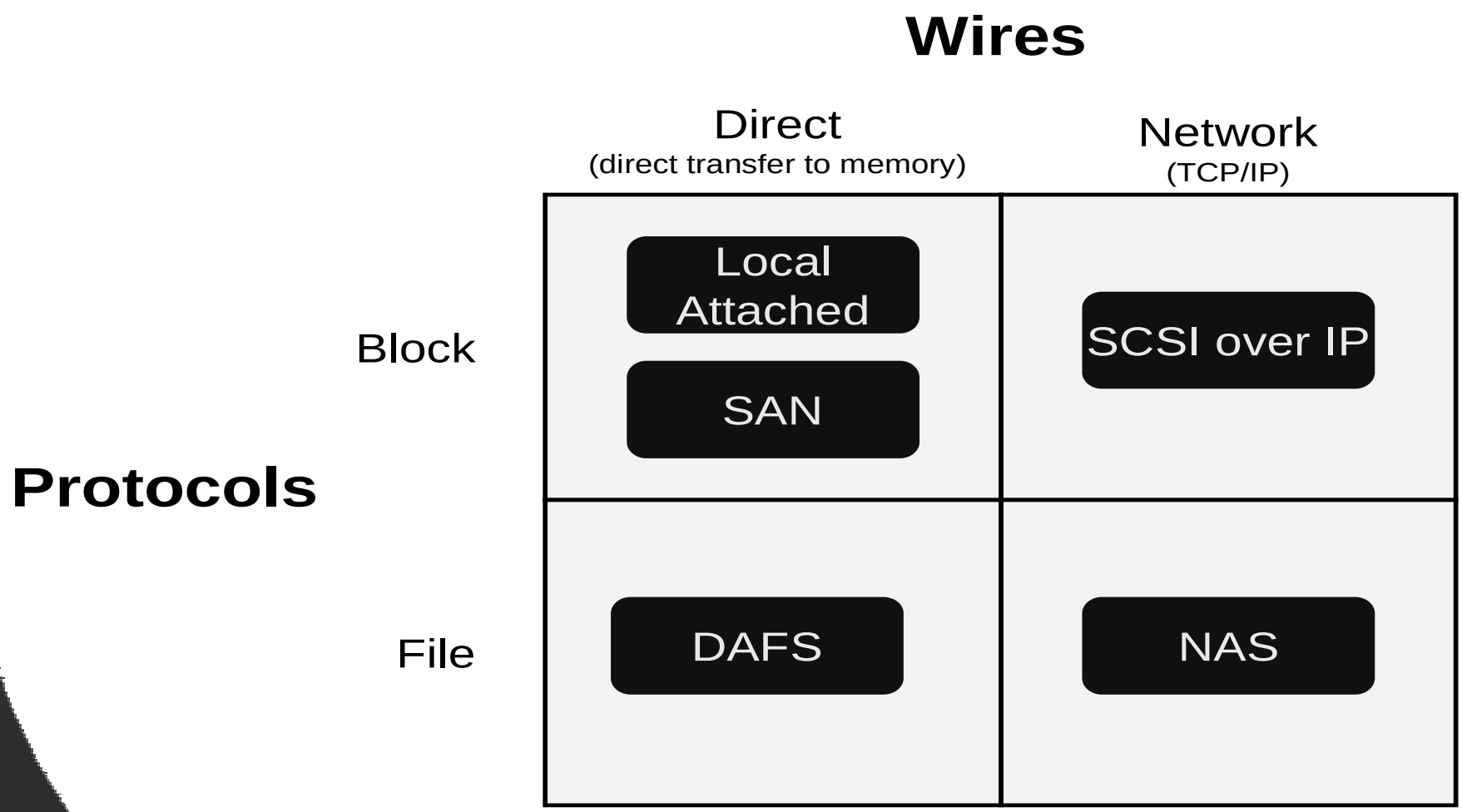
* VI Provider Layer specification maintained by the VI Developers Forum

DAFS advantages

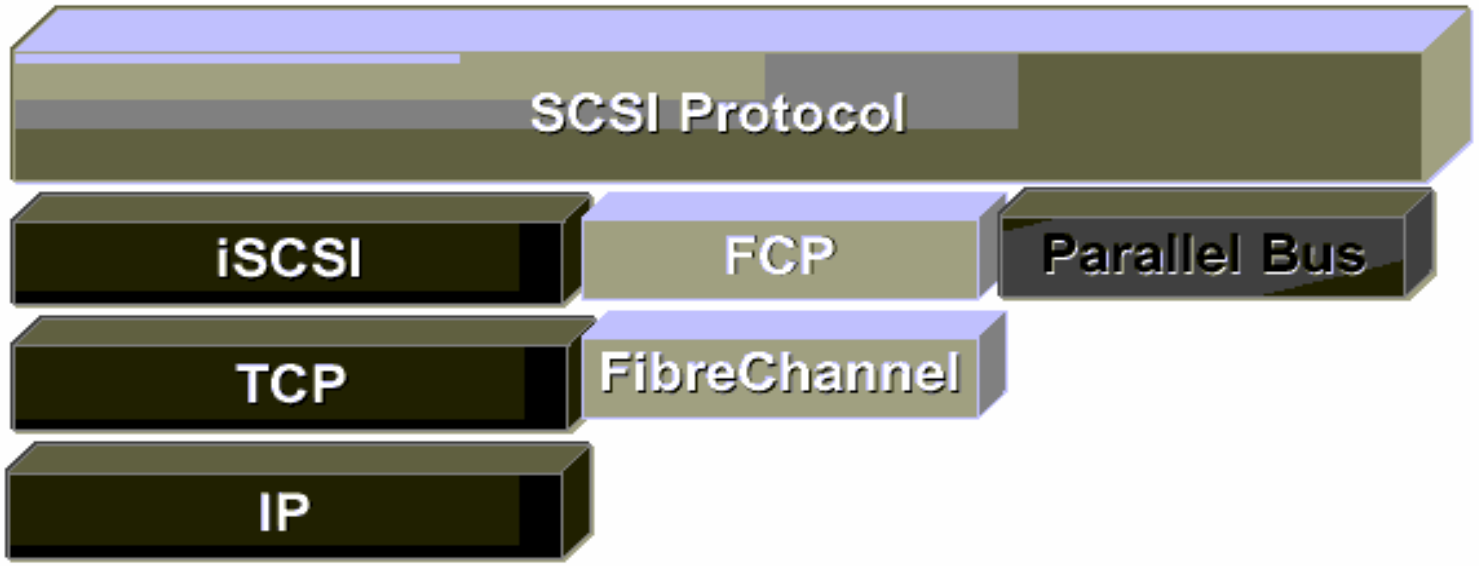
- No fragmentation, reassembly & realignment of data copies
- No user/kernel boundary crossing
- No user/kernel data copies
  - "remote DMA" directly to pre-registered buffers in the receiving application's address space

ATTENTION: DAFS is not for WAN !

DAFS vs. SAN

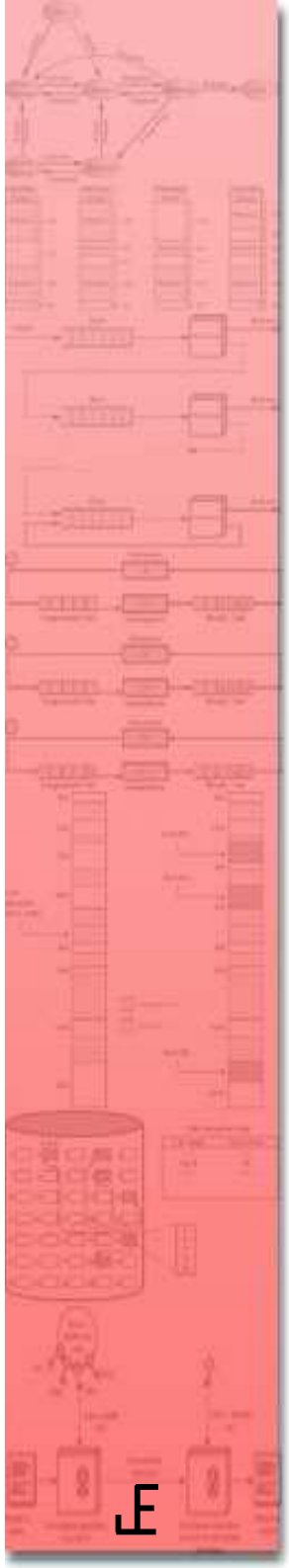

Wires
Direct
(direct transfer to memory)
Network
(TCP/IP)
Protocols
Block
File
Local Attached
SAN
SCSI over IP
DAFS
NAS

What Is iSCSI?



- TCP/IP Transport for SCSI command sets
- Mature Internet draft, SNIA working group
- Block access with no application modifications

NAS
Network Attached Storage

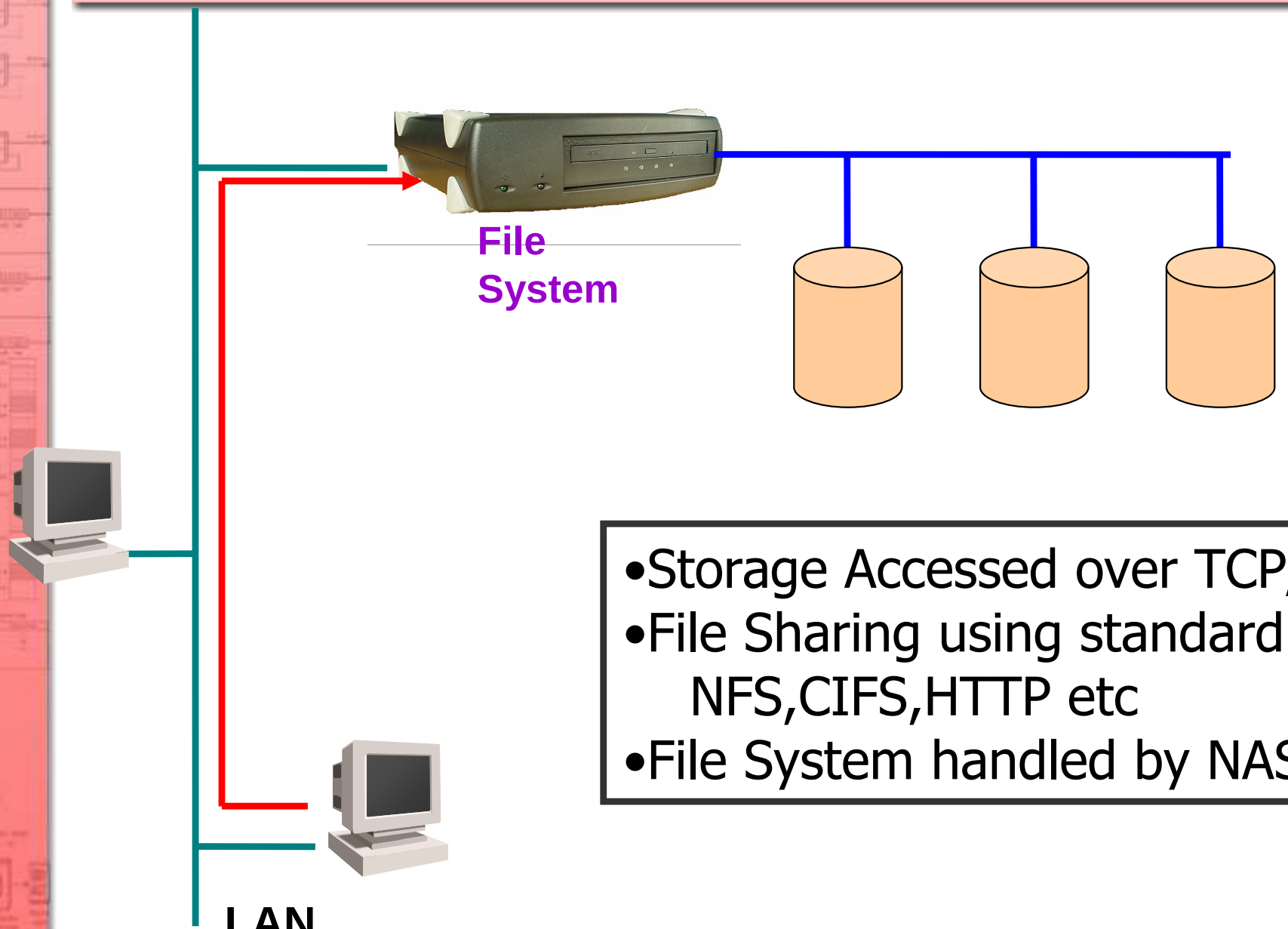


- Connects IDE or SCSI hard disks to Ethernet networks
- Designed for file sharing and data storage in local area networks
  - Simple to install, easy to use and highly reliable
  - No PC required, no cumbersome set-up or administration
  - Virtually maintenance free
  - Flexible & scaleable, low maintenance overheads
  - Complements existing file servers

NAS Workgroup Applications

- Local file sharing in remote and small offices
- Cross-platform file sharing
- Extended personal storage
- Project and workgroup storage
- Backup to low-cost disk
- Program and data distribution
- Portable storage

NAS setup

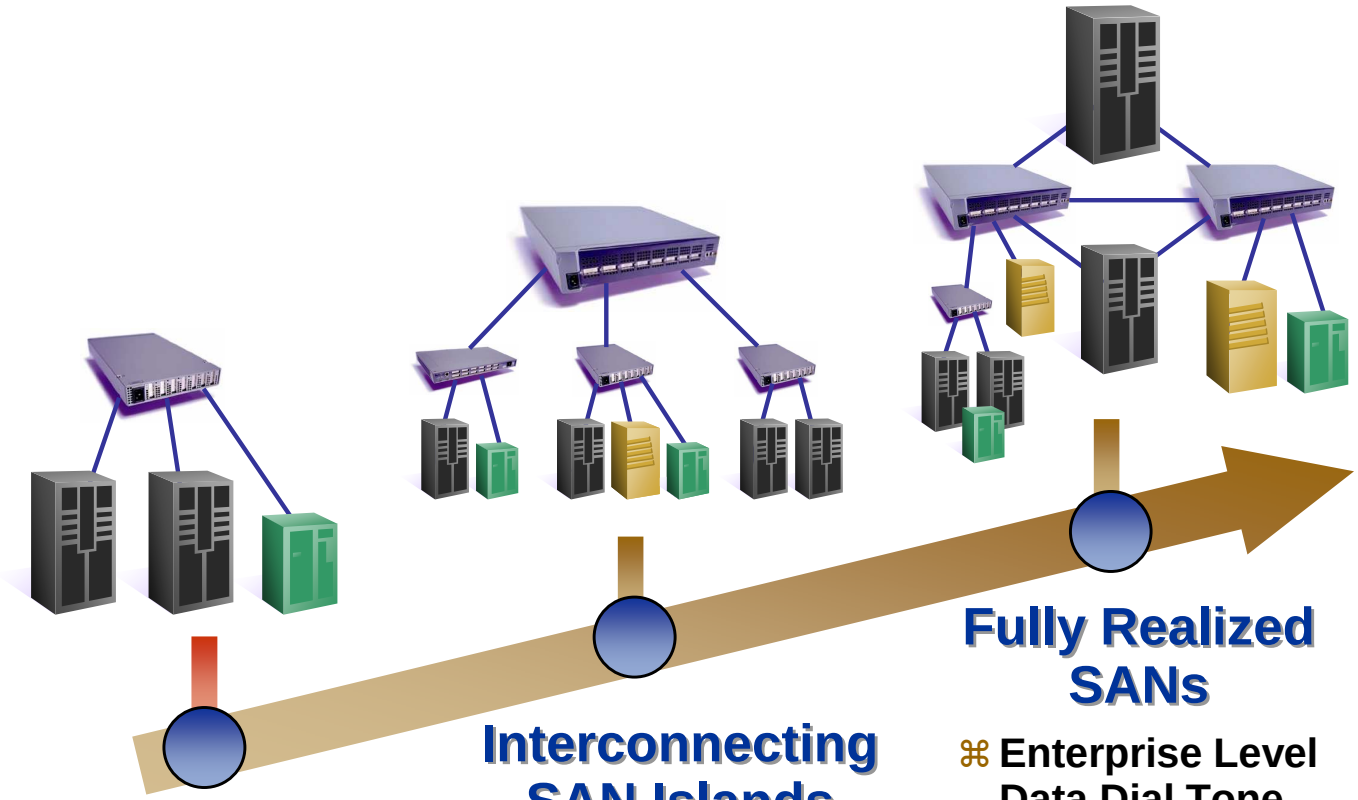


- Storage Accessed over TCP/IP
- File Sharing using standard NFS,CIFS,HTTP etc
- File System handled by NAS unit

SANs – Storage Area Networks

- Geometric increase in the demand for storage capacity
  - Requirement for high-performance storage
  - Ubiquitous storage
- High cost of administering directly-attached storage
- Server consolidation, HA server clusters
- Inability to backup data on LANs

Evolution of Fibre Channel SANs



SAN Islands

- Clustering
- LAN Free Backup
- Simple Applications

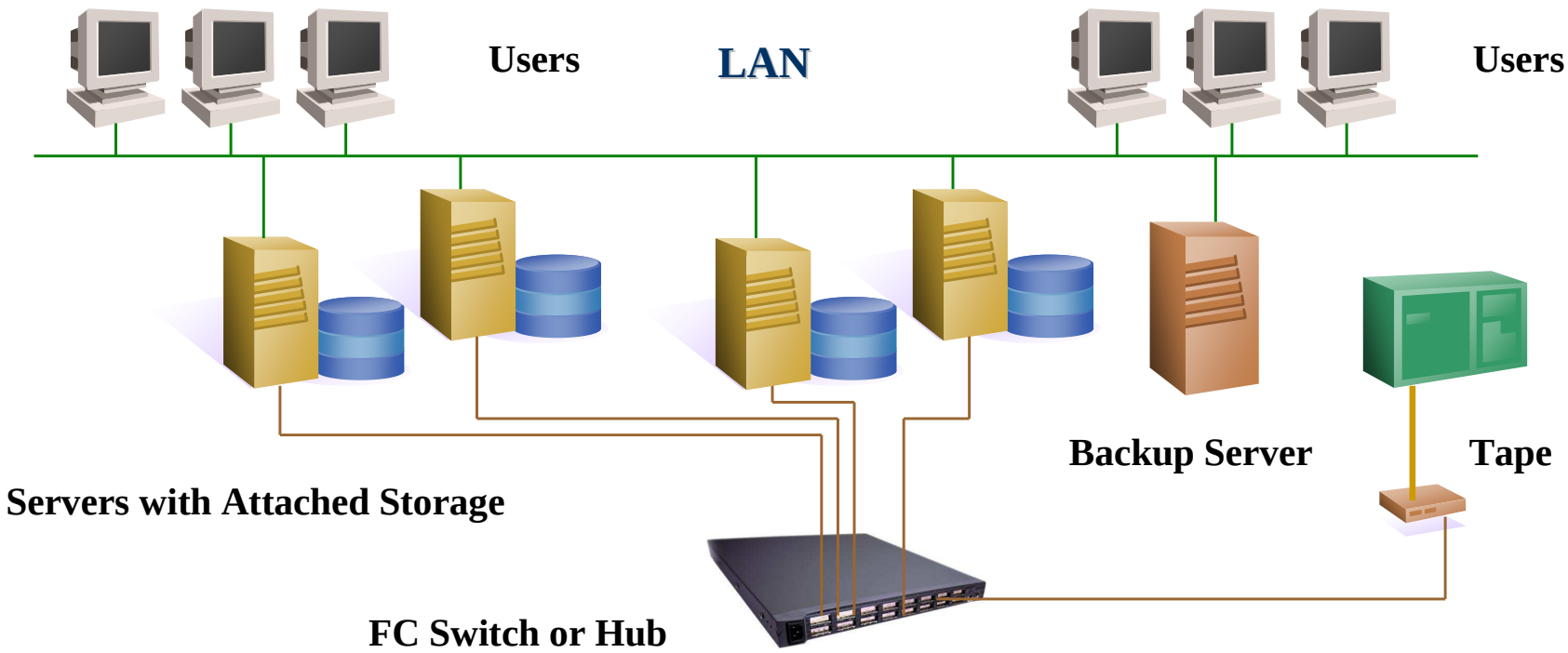
Interconnecting SAN Islands

- Incremental Scaling of SAN Applications

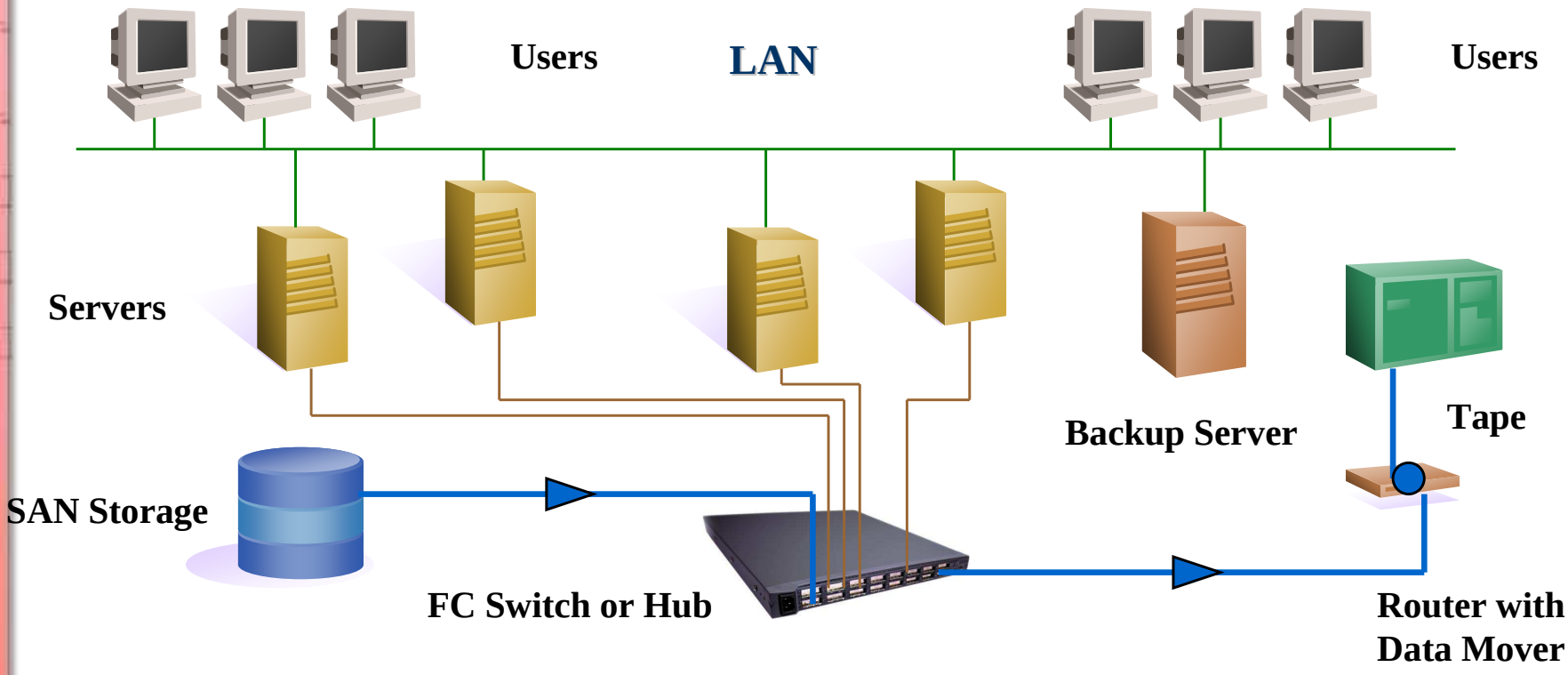
Fully Realized SANs

- Enterprise Level Data Dial Tone
- Open Systems Storage
- Complex Applications

SAN example: LAN-free backup

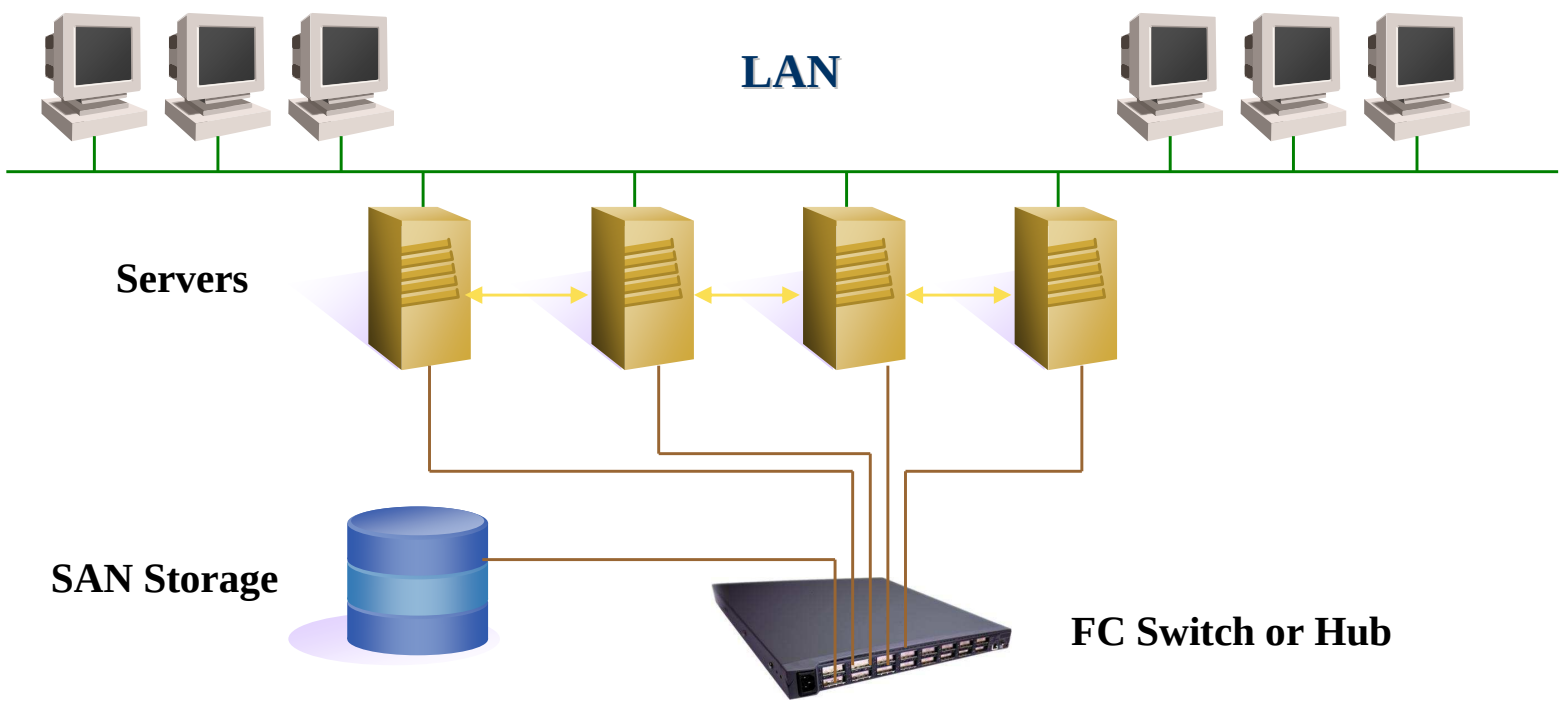


SAN example: server-free backup



- Snapshot of File Structure Synchronizes Data Image
- SAN-based Data Mover Solicits Data Blocks from Storage

SAN example: server cluster



Heartbeat (IP over Ethernet) Monitors Status of Servers

Failure of Component or Application Triggers Fail over Routine

References

- SUN Microsystems Inc, "NFS: Network File System Protocol Specification", RFC-1094, 1989.
- M. Satyanarayanan, H.J. Howard, D.N. Nichols, R.N. Sidebotham, A. Spector, and D.C Steere, "Coda: A highly-available file system for a distributed workstation environment", IEEE Trans. Computers, vol. 39, no. 4, pp. 447-459, 1990.
- T.E. Anderson, M.D. Dahlin, J.M. Neefe, D. Patterson, D.S. Roselli, and R.Y.Wang, "Serverless file systems", ACM Trans. Computer Systems, vol. 14, no. 1, pp.41-79, 1996.
- M. Satyanarayanan, "Distributed File Systems", in: "Distributed Systems", 2nd Edition, edited by S. Mullender, ACM Press, 1993.
- S. Kleinman and J. Katcher, "An Introduction to the Direct Access File System (v. 0.6)", Network Appliance Inc, June 2001.

CS 194: Distributed Systems
Distributed File Systems

Scott Shenker and Ion Stoica
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720-1776

Main Goal

- Provide a client **transparent** access to a file system stored at a **remote** server

- Why would you want to store files remotely?

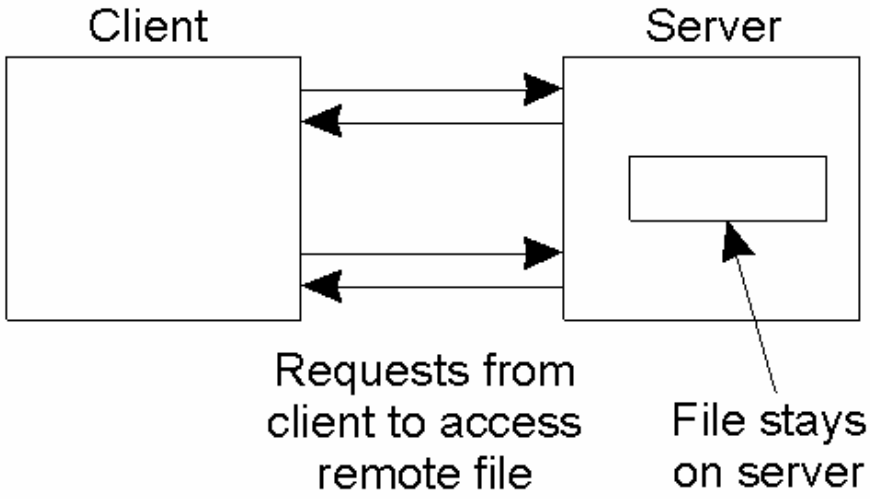
NFS

- A specification for a distributed file system (by Sun, 1984)
- Implemented on various OS's
- De facto standard in the UNIX community
- Latest version is 4 (2000)
- Client-server file system

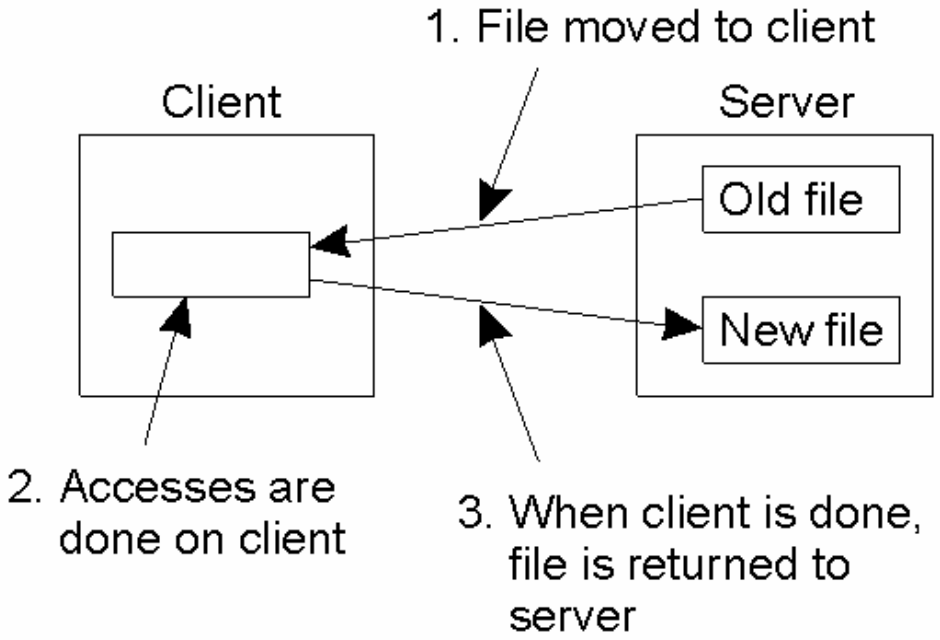
Access Model

- Two access models :
  - Remote access
  - Upload/Download

Remote access

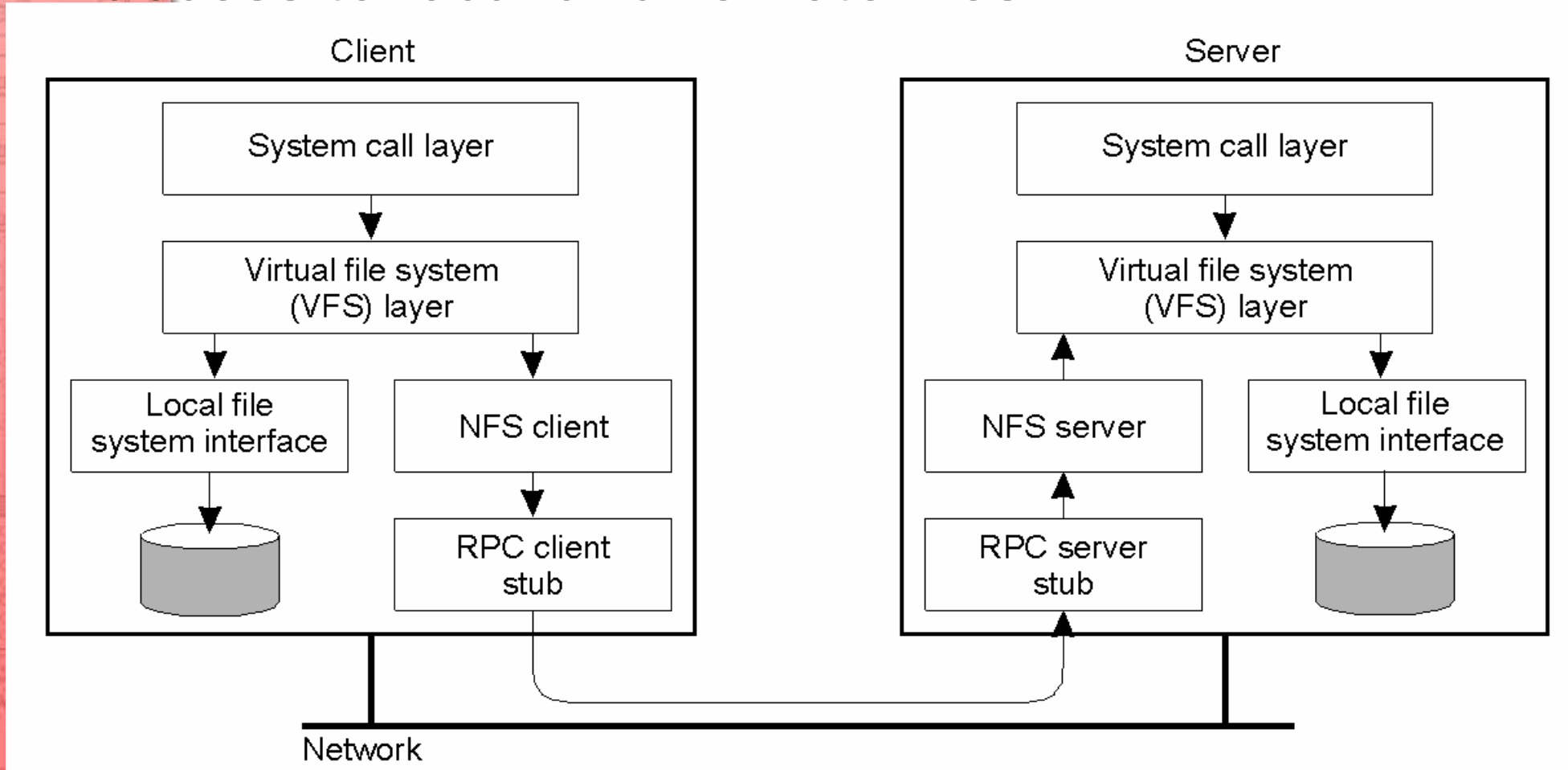


Upload/Download



NFS Architecture

- Virtual File System (VFS) provide a uniform access to local and remote files



File System Model

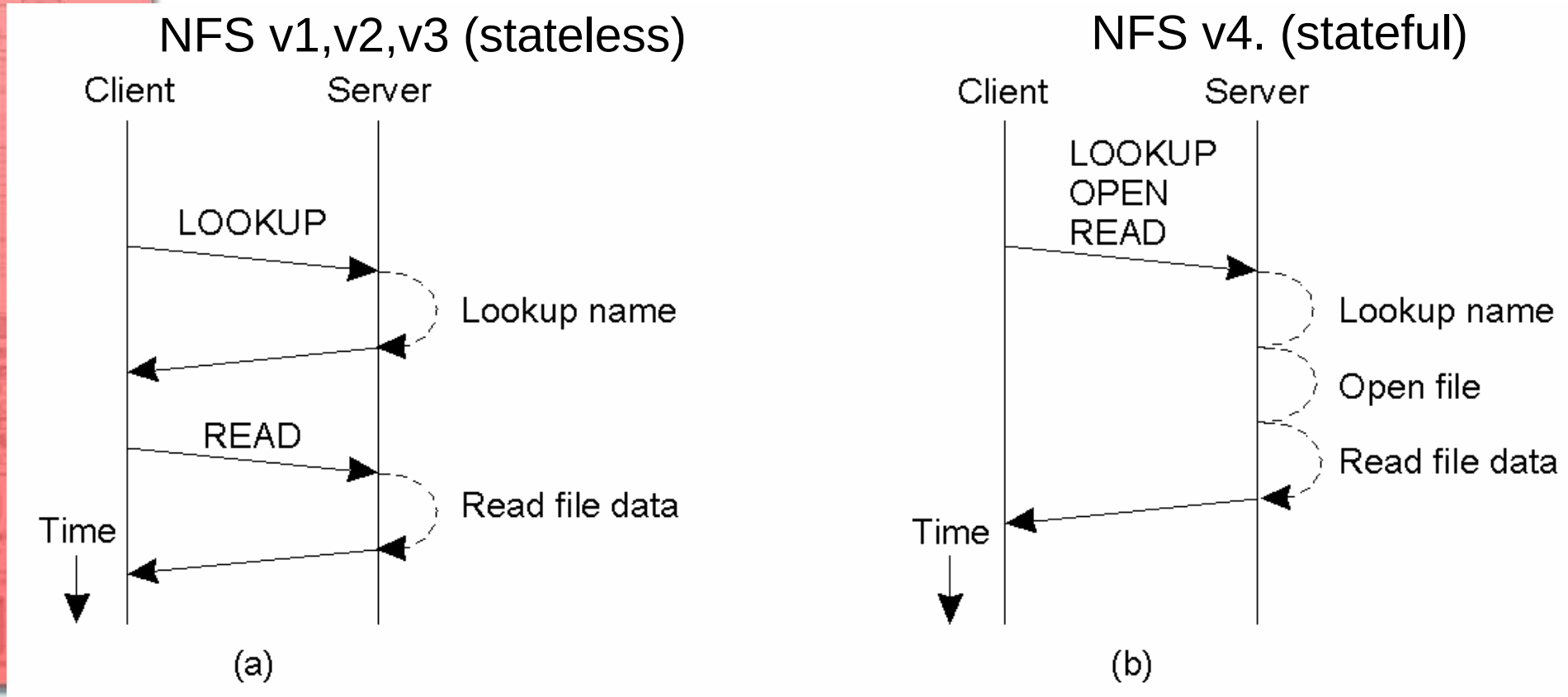
- Similar to UNIX: files are treated as uninterpreted sequences of bytes
- Each files has a name, but usually referred by a **file handle**
  - Client use a name service to get file handle
- Files organized into a naming graphs
  - Nodes → directories or files
- First three versions were **stateless**; version 4 is **stateful**

Stateful vs. Stateless

- Stateless model: each call contains complete information to execute operation.
- Stateful model: server maintain context (info) shared by consecutive operations.
- **Discussion:** compare stateless and stateful design.

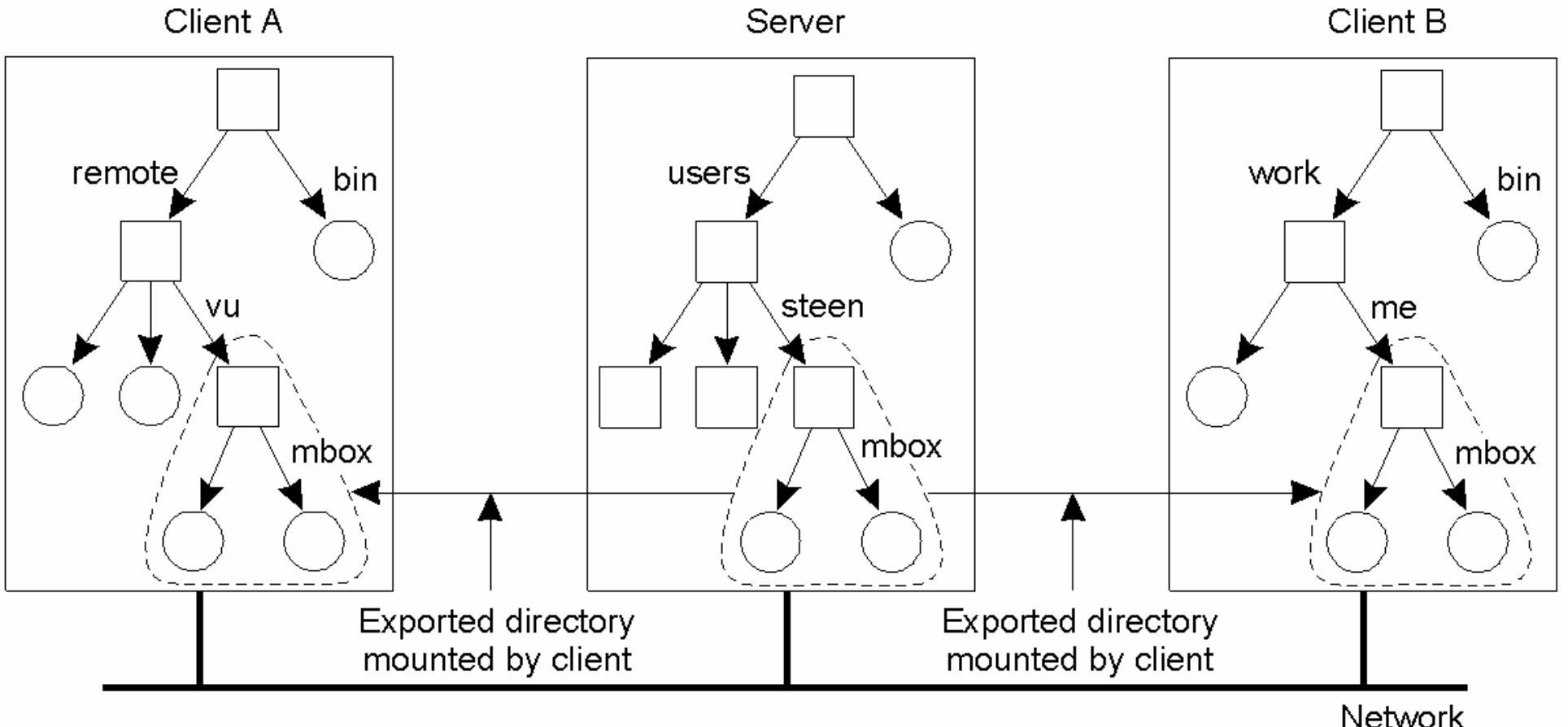
Communication

- RPC based
- One operation per RPC (NFS v. 1,2,3)
- Multiple operations per RPC (NFS v. 4)

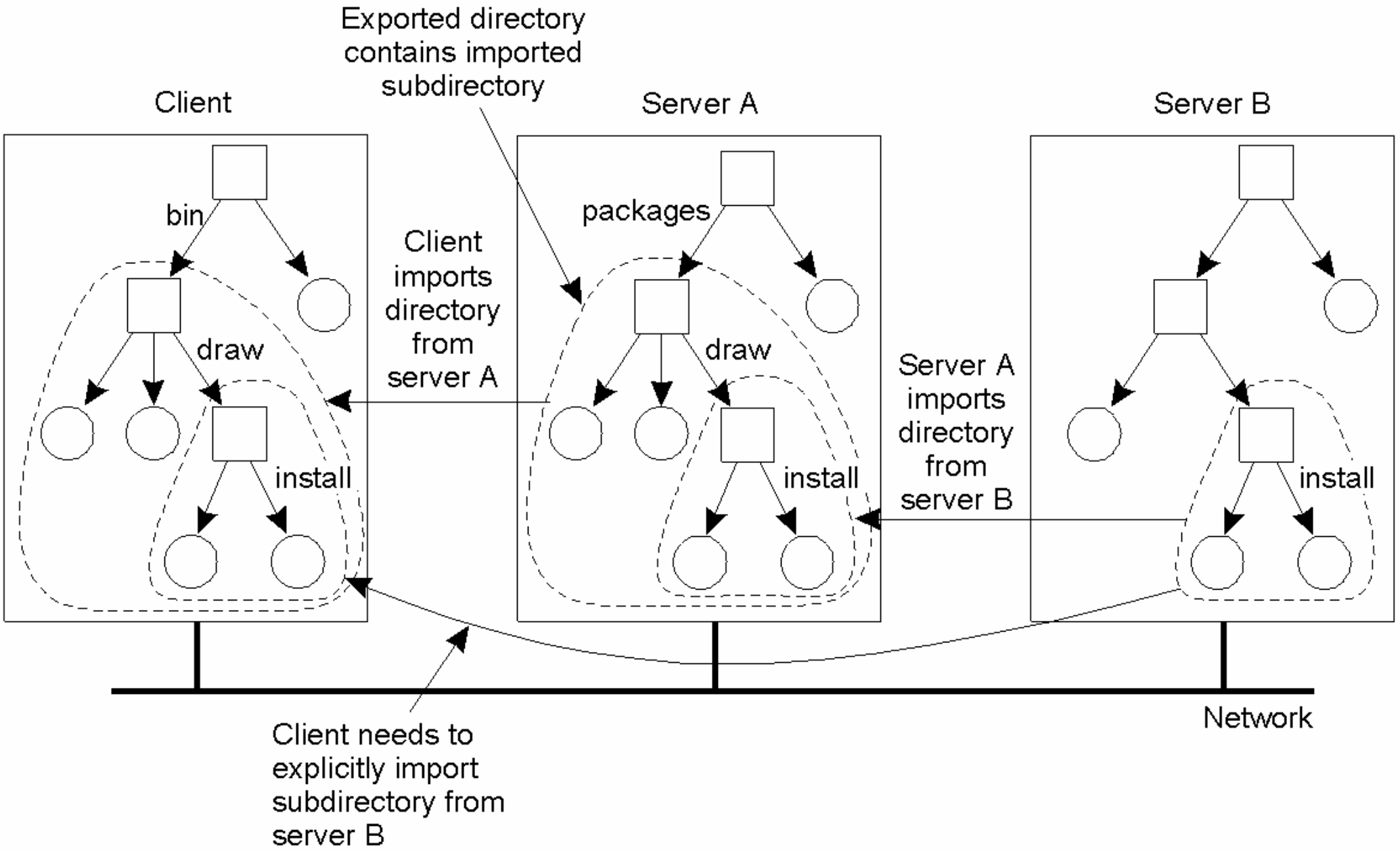


Naming

- Allow a client to mount a remote file system into its own local file system.
- Pathnames are **not** globally unique; what's the implication?



Example: Mounting Nested Directories

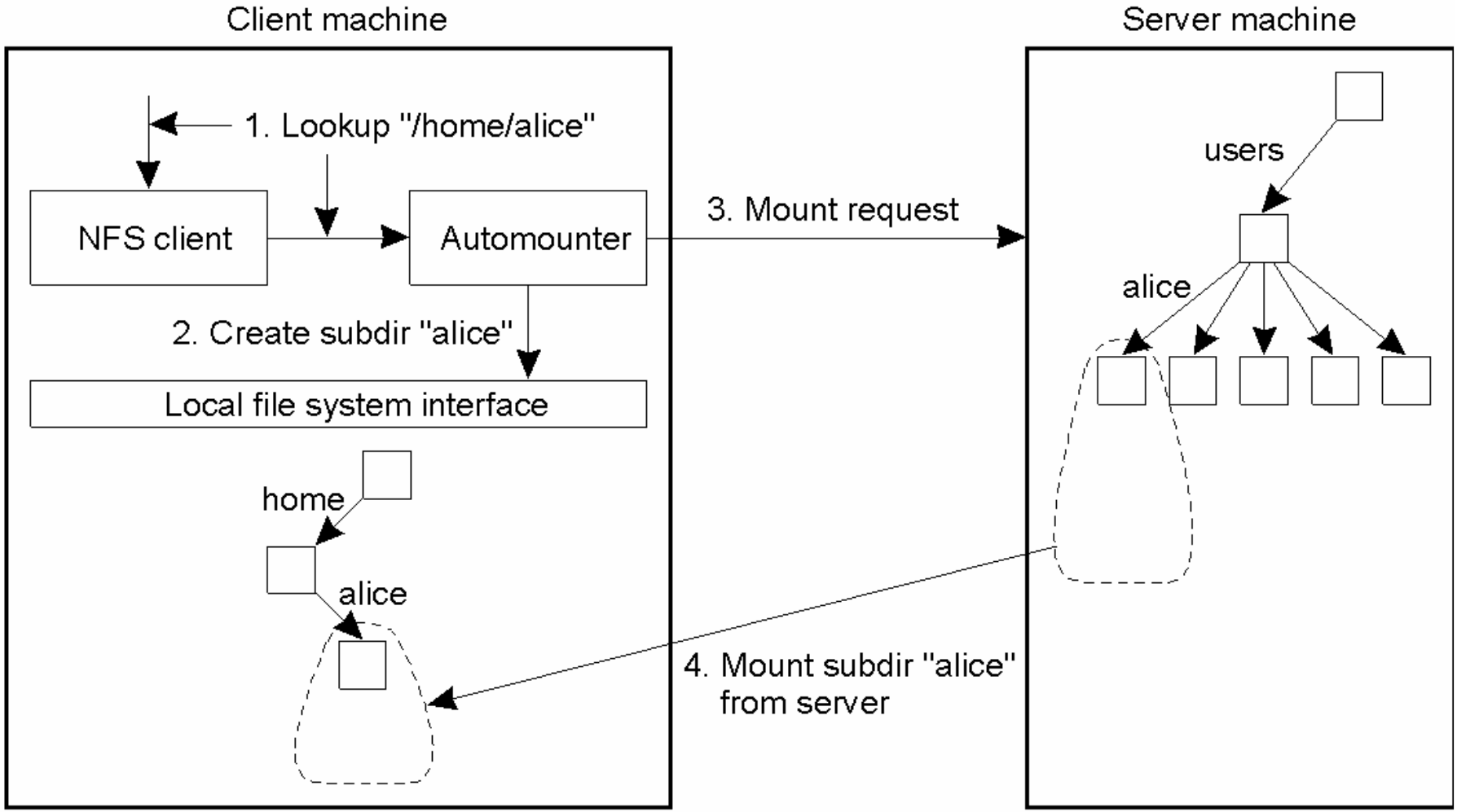


File Handles

- File handle: created by server hosting the file.
- Unique with respect to all file systems exported by servers.
- Persistent: doesn’t change during file’s lifetime.
- Length: 32b in v2, 64b in v3, and 64b in v4

Automounting

- Mount file system **transparently** when client accesses it.



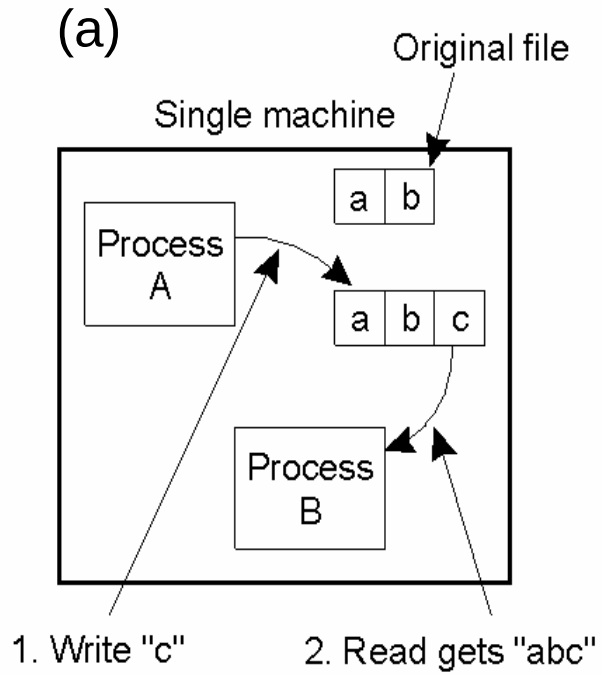
Mandatory File Attributes

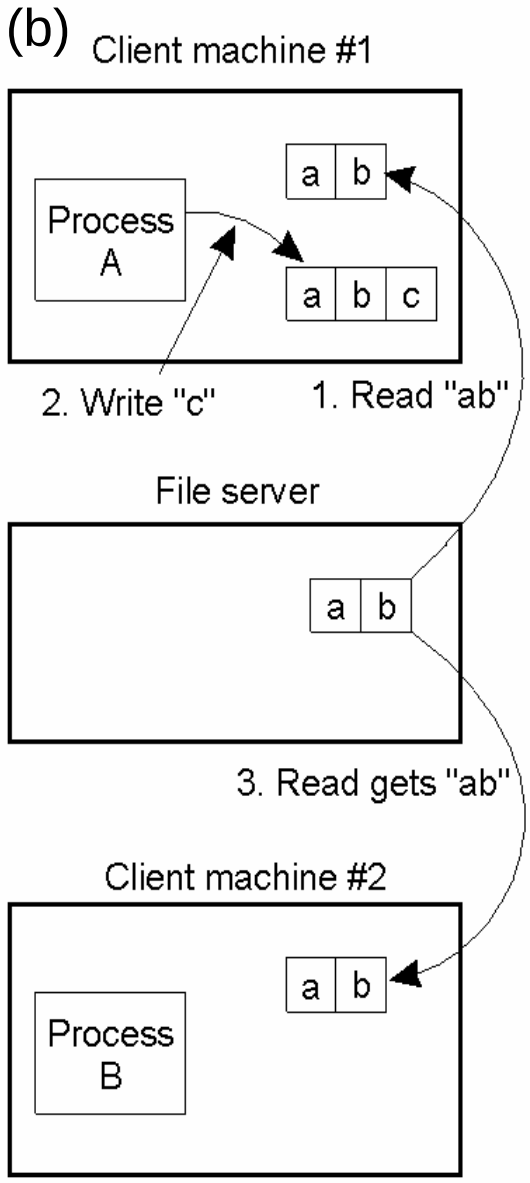
Attribute	Description
TYPE	The type of the file (regular, directory, symbolic link)
SIZE	The length of the file in bytes
CHANGE	Indicator for a client to see if and/or when the file has changed
FSID	Server-unique identifier of the file's file system

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.





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

- NFS implements session semantics

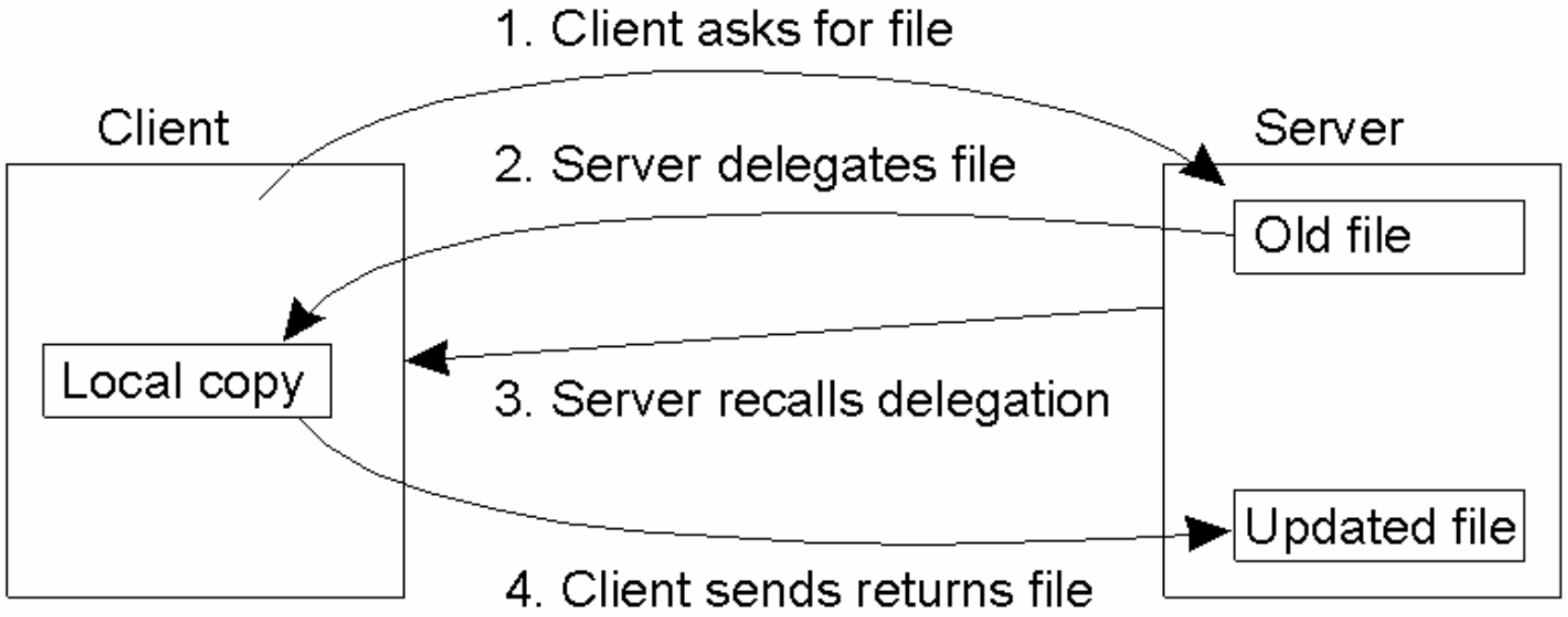
File Locking in NFS

- NFS v1-v3: use a separate (stateful) lock manager
- NFS v4: integrated in the file system

Operation	Description
Lock	Creates 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 leas on a specified lock

Client Caching (repaso)

- NFS v1-v3: mainly left outside the protocol (see book)
- NFS v4: use file delegation



Fault Tolerance

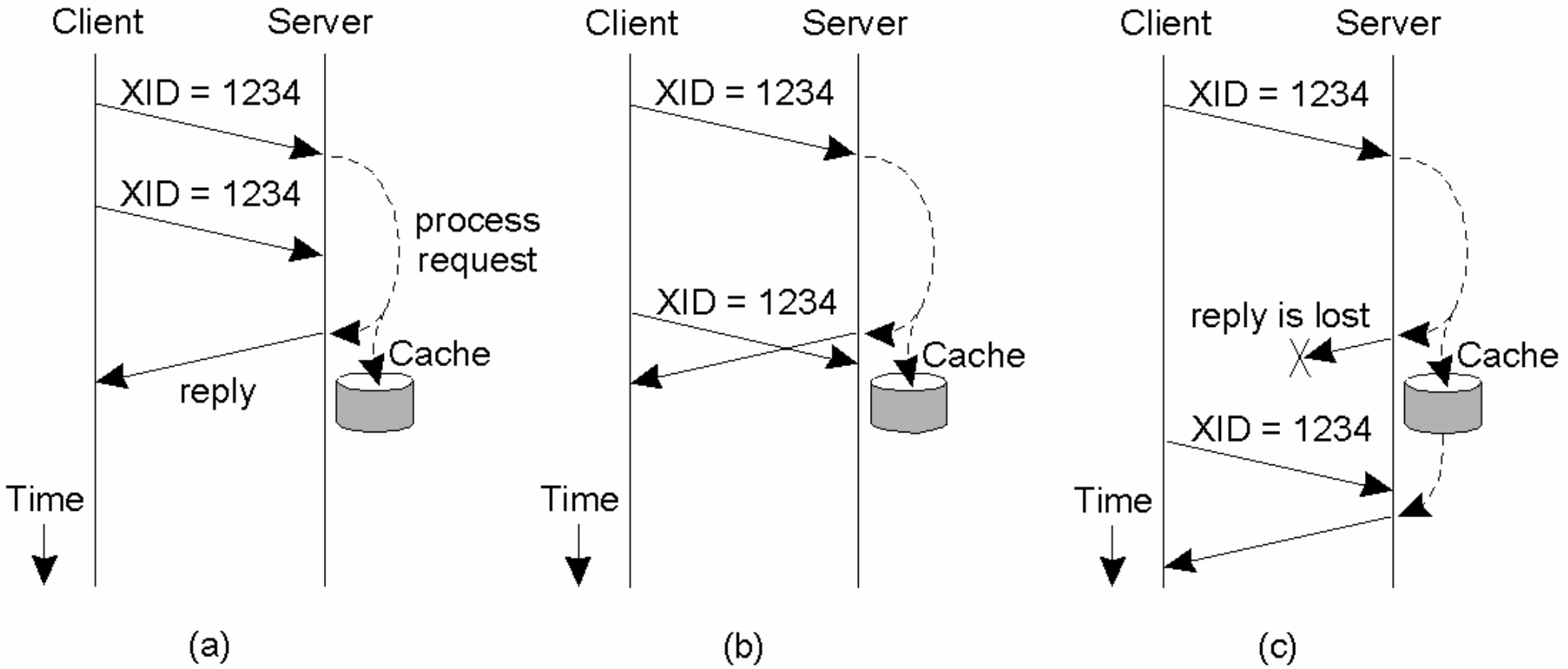
- Need to maintain state consistent in v4, e.g.,
  - Locking
  - Delegation
- Challenge: eliminate duplicate operations in case of failure
- Solution: use transaction identifiers (XID)

Handling Retransmissions

a) Request still in progress

b) Reply has just been returned

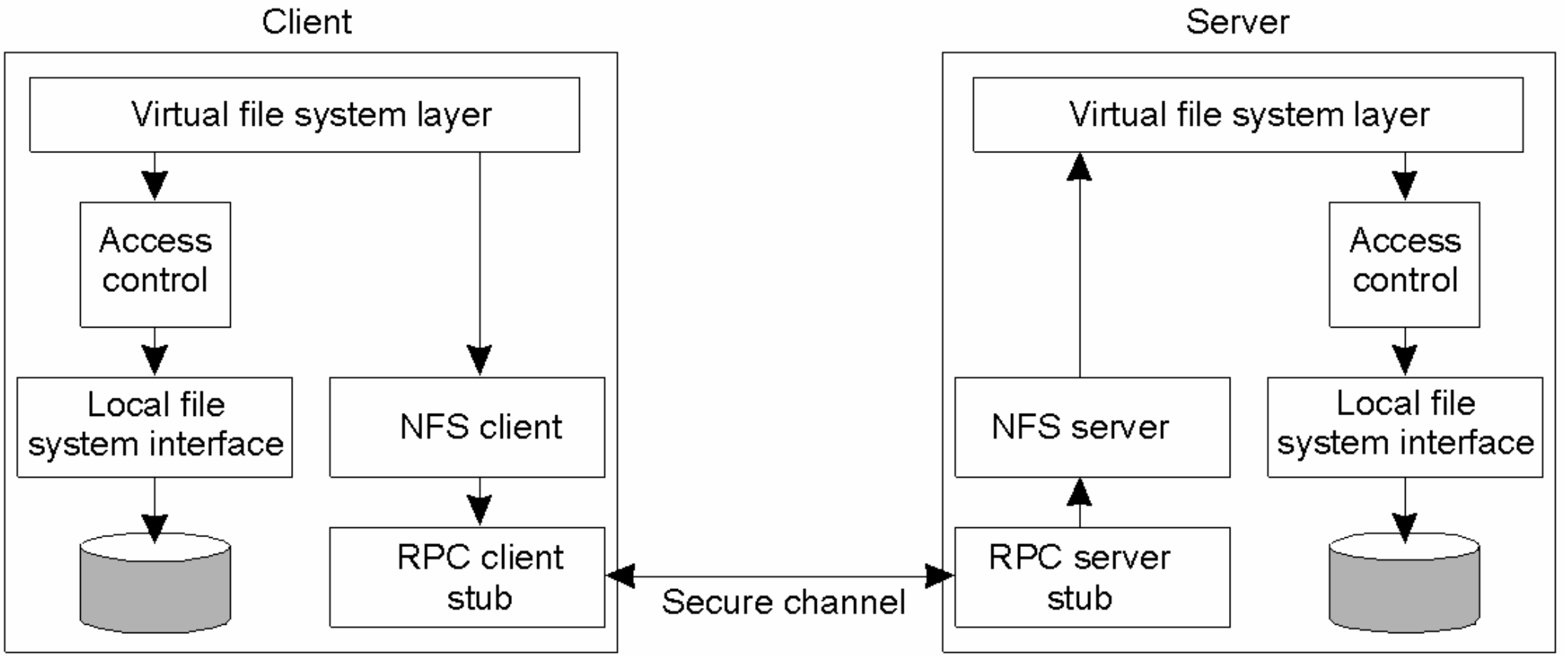
c) Reply has been some time ago, but was lost



Security

- Secure RPC: three methods
  - System authentication: trust the user has passed a proper login procedure
  - Diffie-Hellman key exchange (but not very secure—uses only 192b Keys)
  - Kerberos
- File access control
  - Use access control list (ACL)

Security Architecture



The Coda File System

- Developed at CMU
- Based on Andrew File System (AFS), another distributed system developed at CMU
  - Community wide system

AFS Goals

- **Scalability:** system should grow without major problems
- **Fault-Tolerance:** system should remain usable in the presence of server failures, communication failures and voluntary disconnections
- **Unix Emulation**
- **Design philosophy: Scalability and Accessibility more important than consistency**

Coda Goals

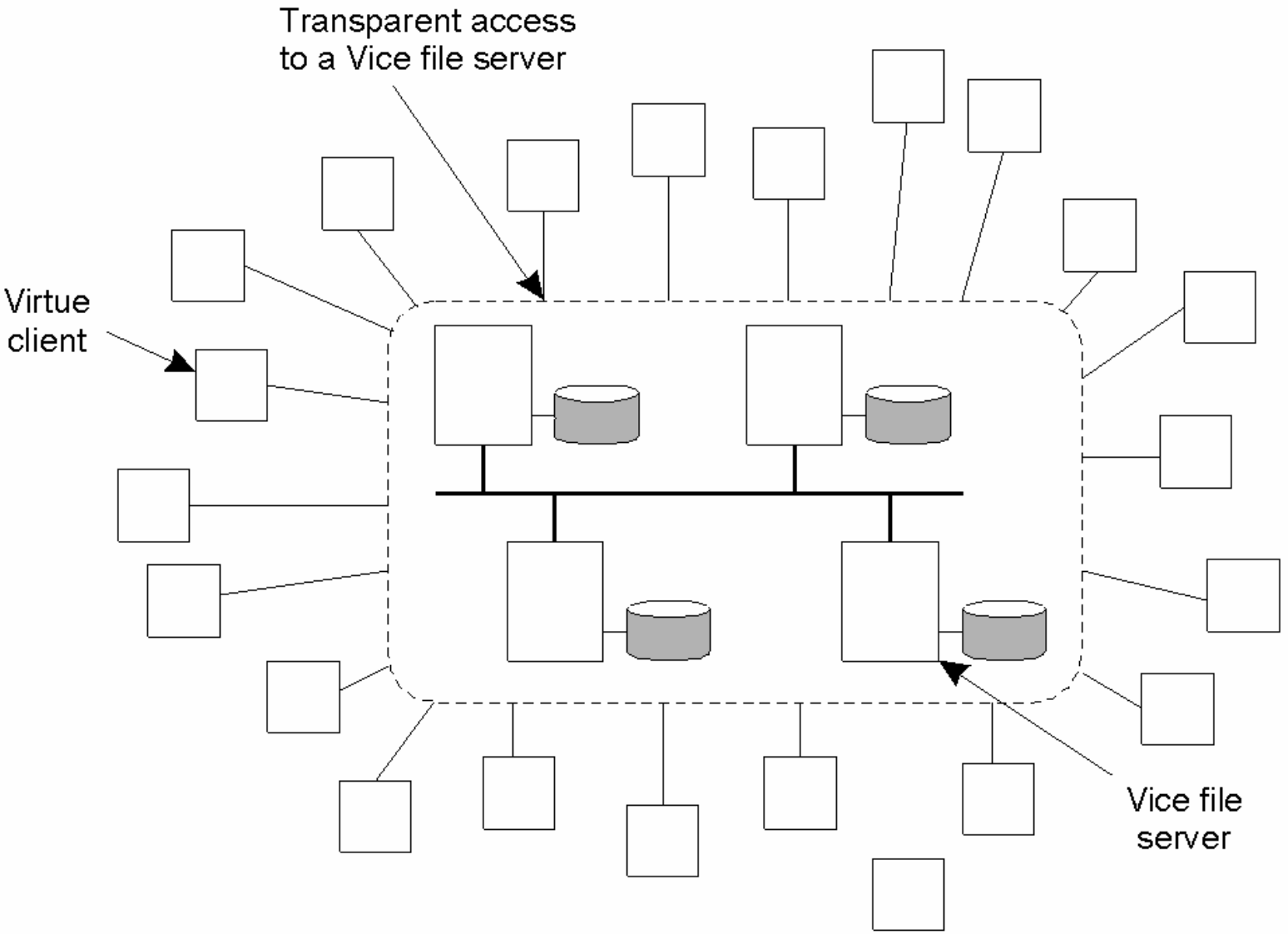
- AFS goals, plus
- **Disconnected mode** for **portable computers**

System Model

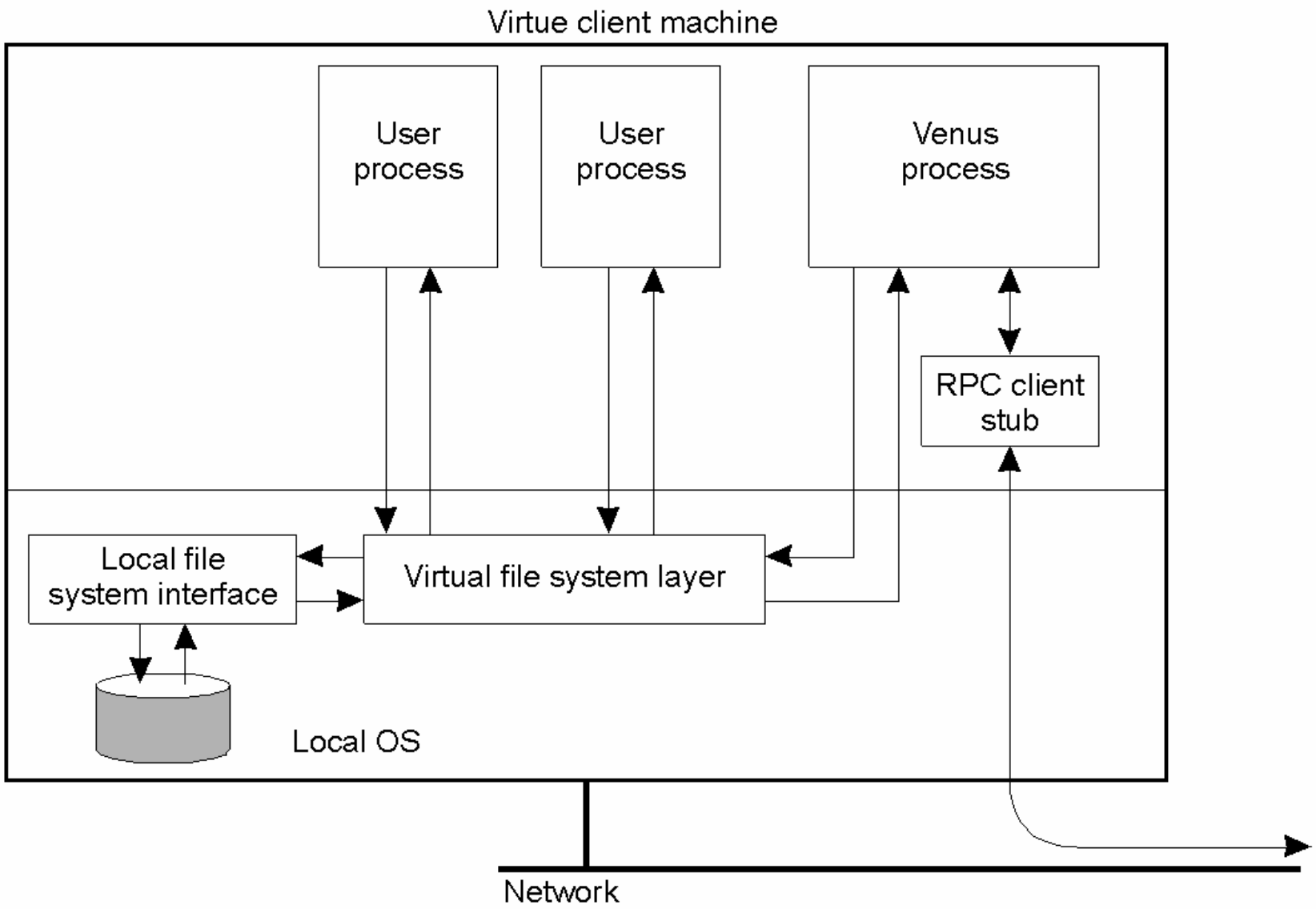
- Client workstations are personal computers owned by their users
  - **Fully autonomous**
  - **Cannot be trusted**

- Coda allows laptops that operate in **disconnected mode**

Overall Organization of AFS & Coda



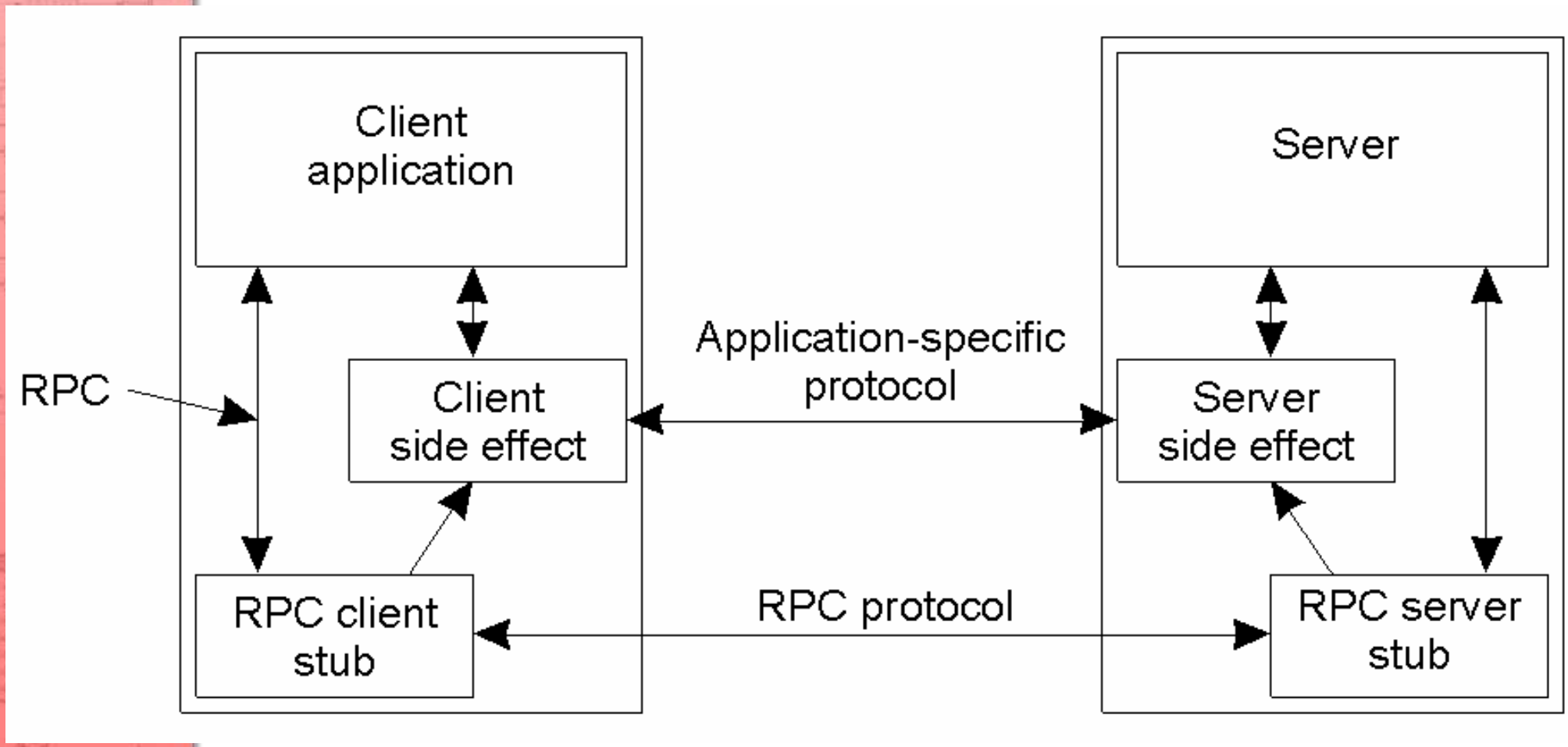
Internal Organization of Virtue



Communication

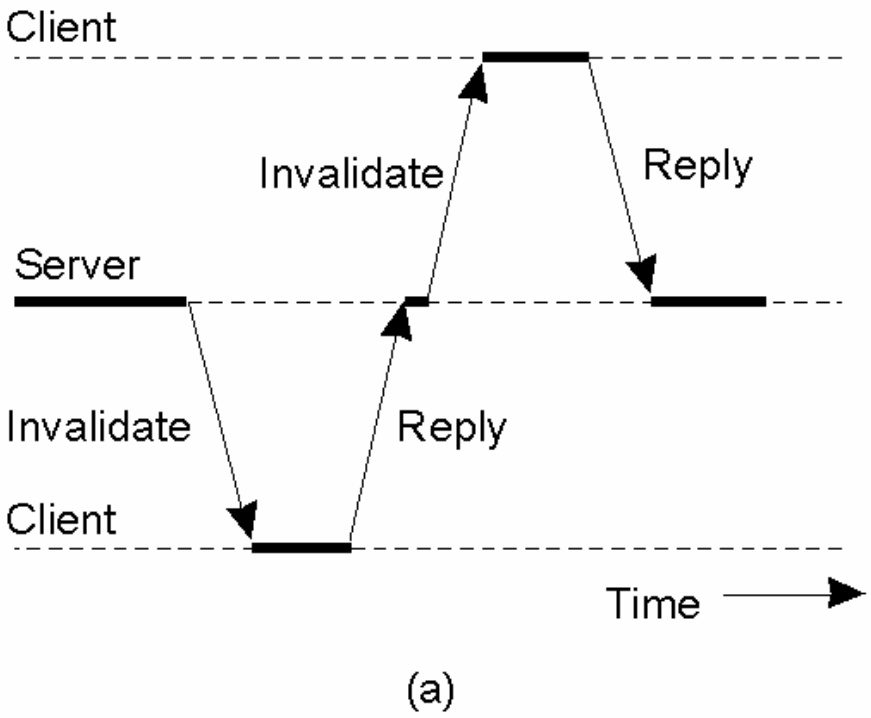
- Based on RPC2: provides reliable transmission on top of UDP
- RPC2 supports side-effects, i.e., user defined protocols
- RPC2 provides support for multicast
  - Transparent for the client

Side Effects in Coda's RPC

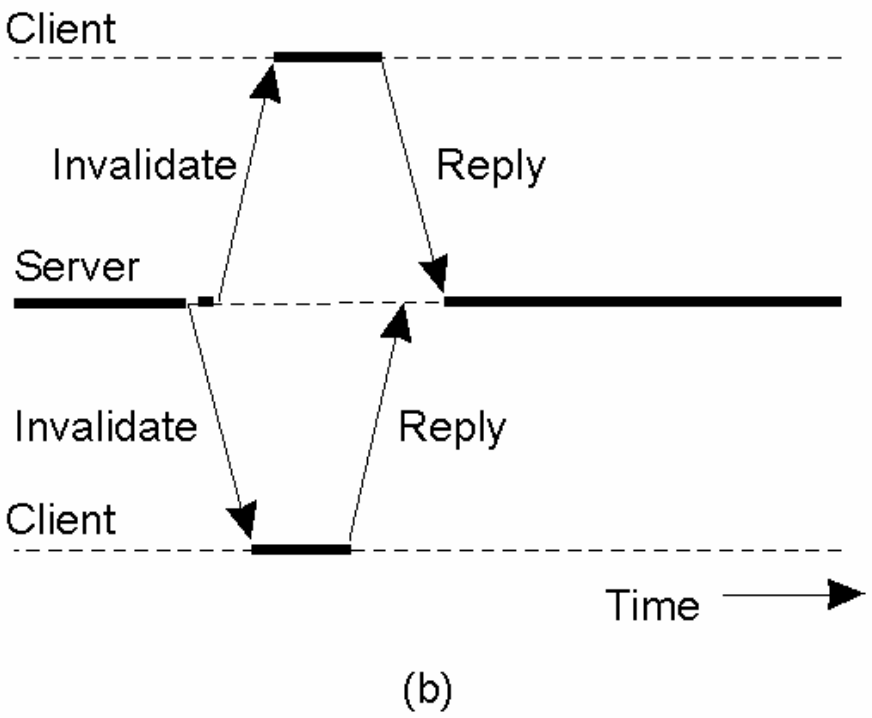


Support for Multicast in RCP2

- Example: send invalidation
  a) One at a time
  b) Multicast



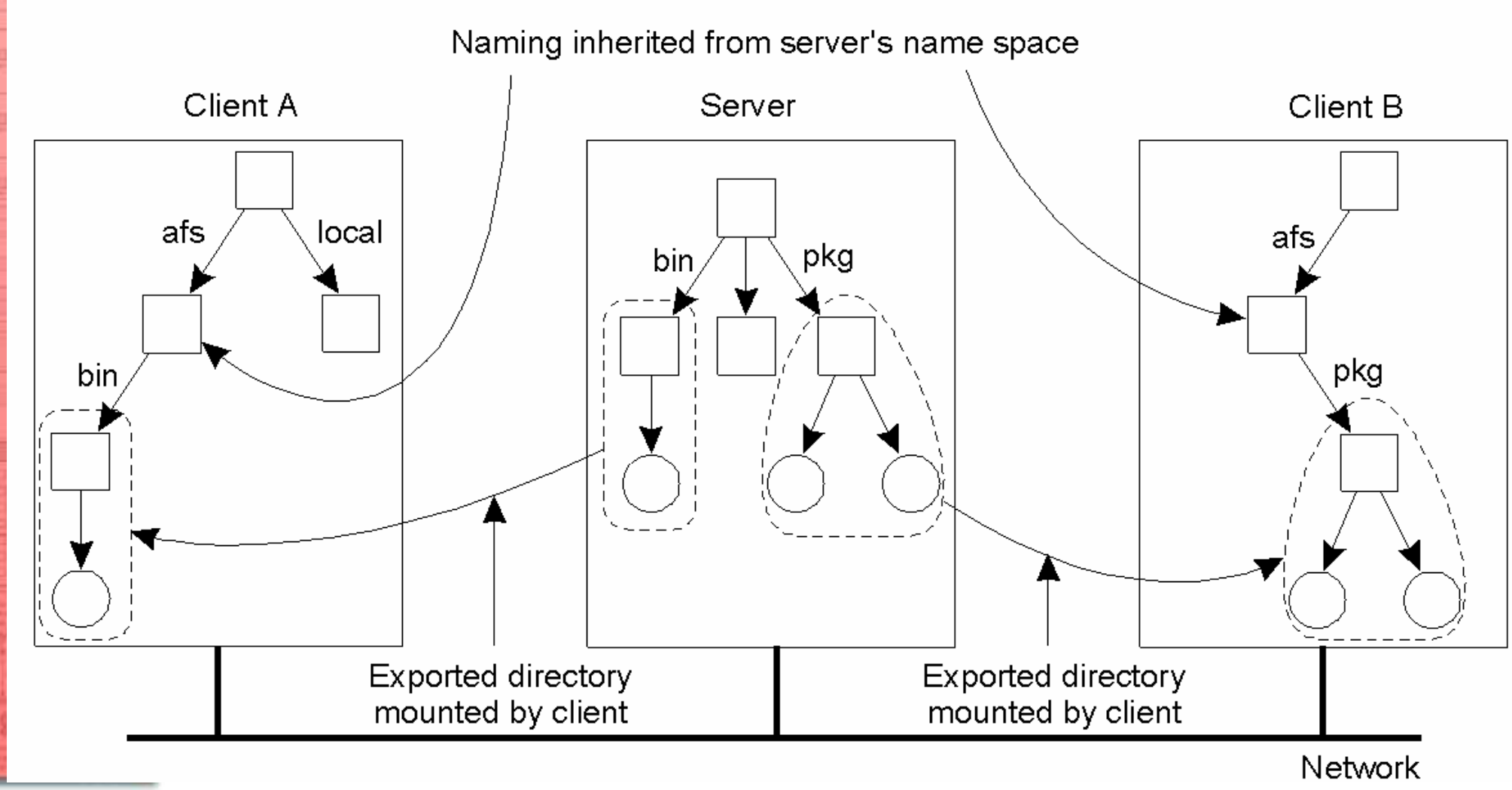
(a)



(b)

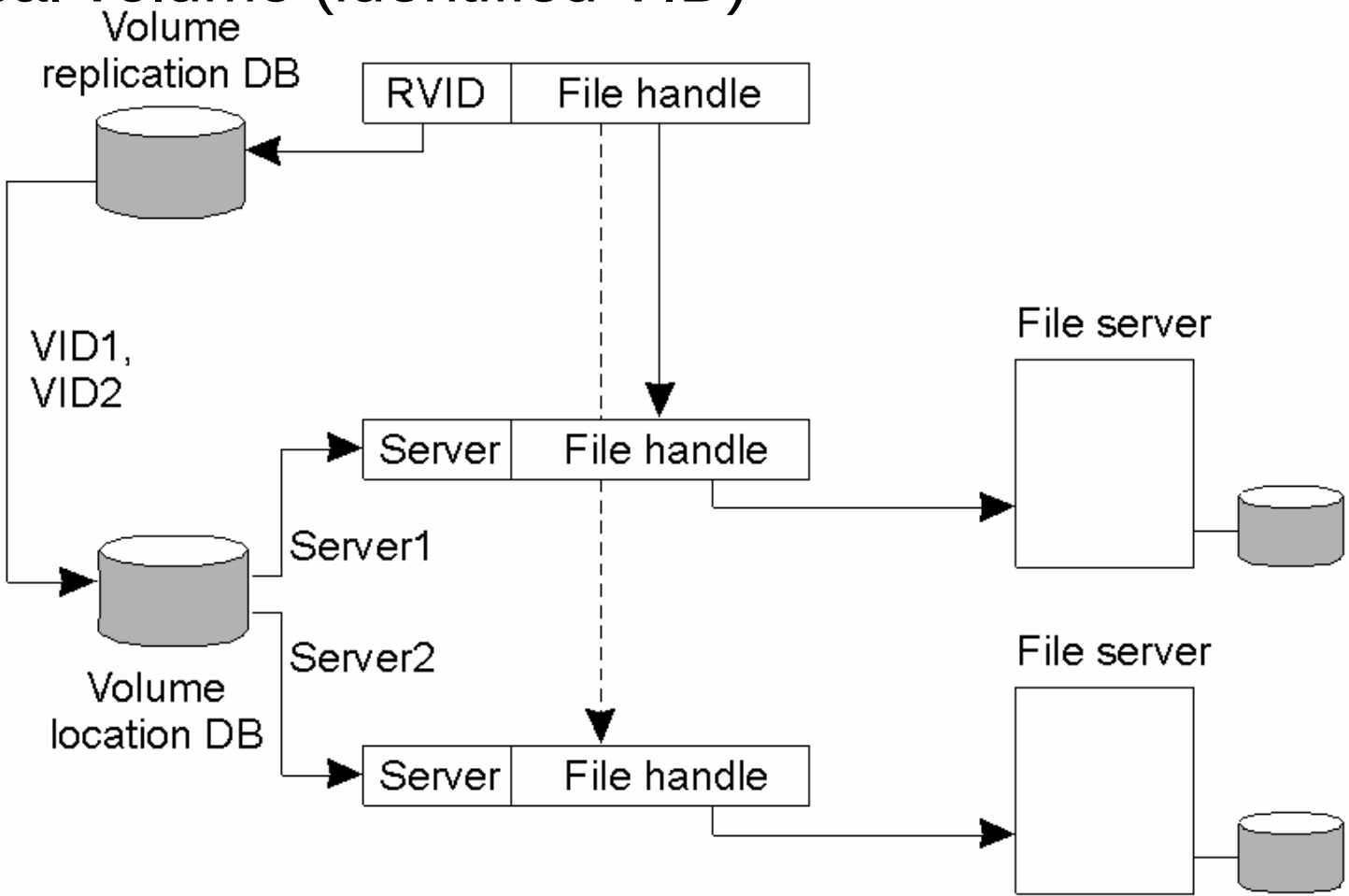
Naming

- Single shared naming space (vs. client-based in NFS)



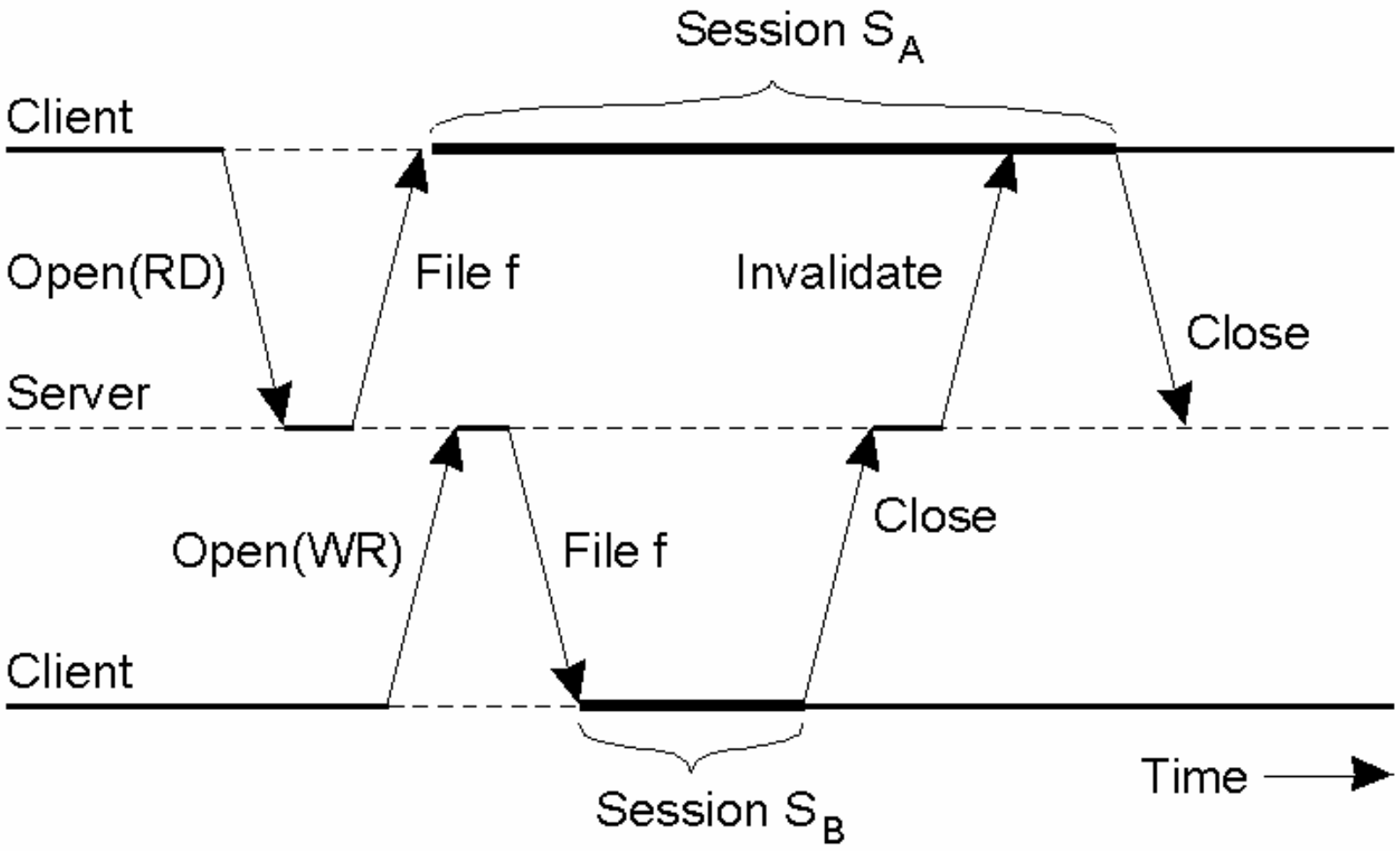
File Identifiers

- Globally unique (vs server unique in NFS)
- Coda distinguishes between physical and logical volumes
- Logical volume (identified by RVID) a possible replicated physical volume (identified VID)



Sharing Files in Coda

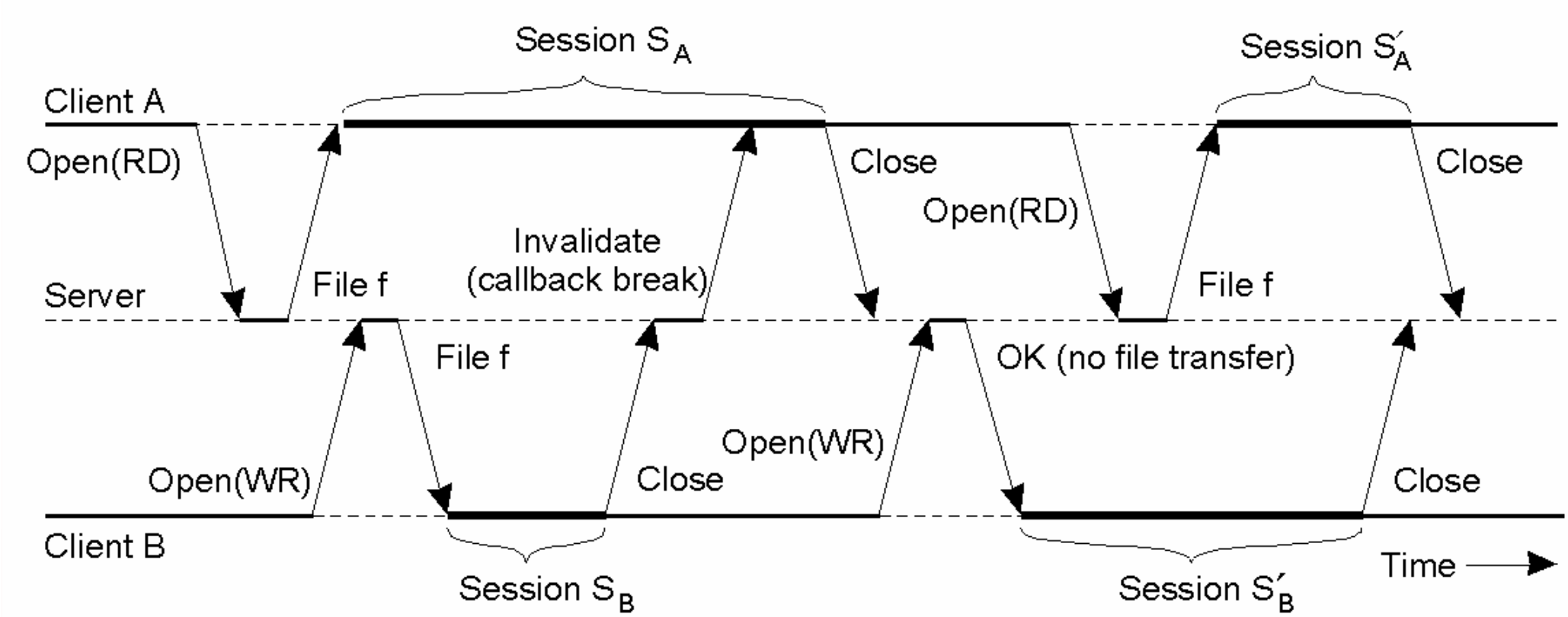
- Transaction semantics
  - Session is treated like a transaction



Caching and Replication

- Caching:
  - Achieve scalability
  - Increases fault tolerance
- Challenge: how to maintain data consistency?
- Solution: use callbacks to notify clients when a file changes
  - If a client modifies a copy, server sends a **callback break** to all clients maintaining copies of same file

Example: Client Caching

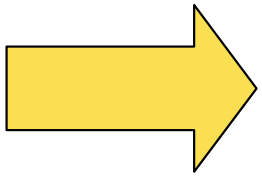


Server Replication

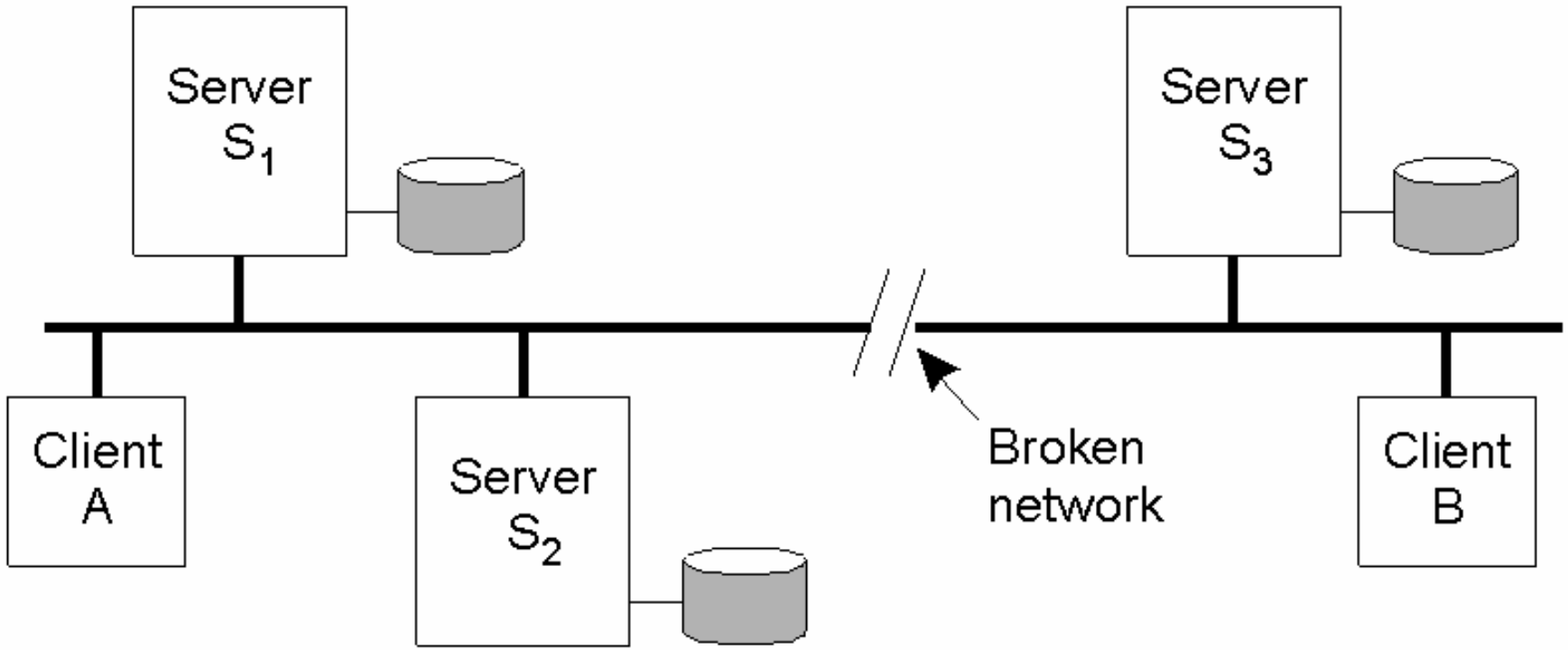
- Unit of replication: volume
- Volume Storage Group (VSG): set of servers that have a copy of a volume
- Accessible Volume Storage Group (AVSG): set of servers in VSG that the client can contact
- Use vector versioning
  - One entry for each server in VSG
  - When file updated, corresponding version in AVSG is updated

Example: Handling Network Partition (1)

- Versioning vector when partition happens: [1,1,1]
- Client A updates file → versioning vector in its partition: [2,2,1]
- Client B updates file → versioning vector in its partition: [1,1,2]
- Partition repaired → compare versioning vectors: **conflict**!

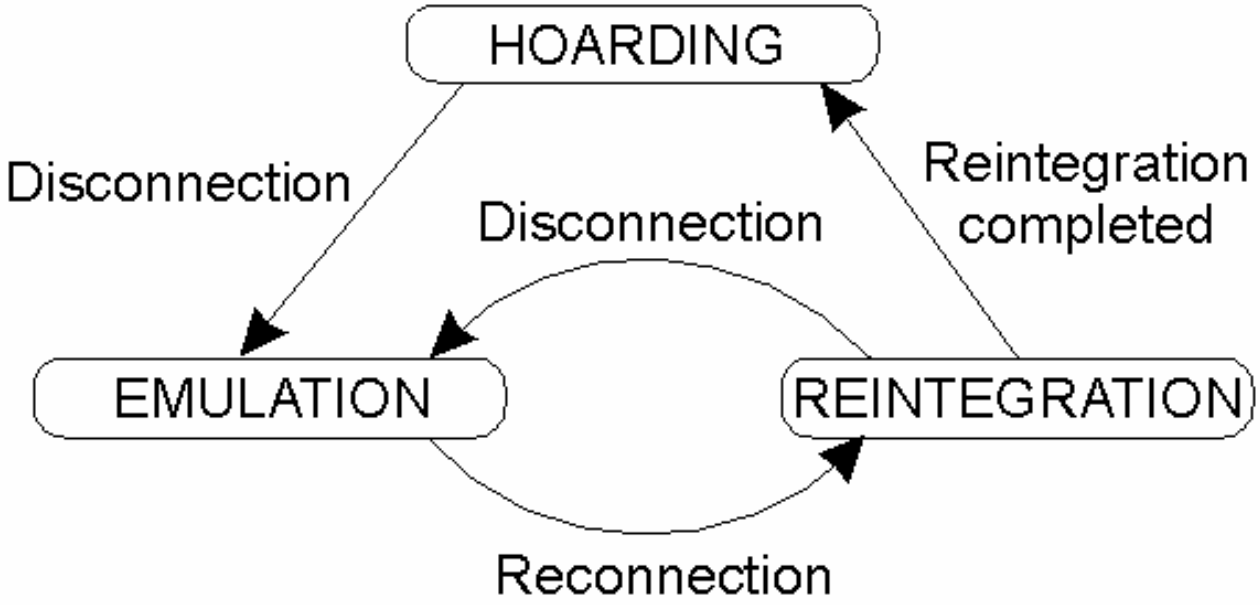


Example: Handling Network Partition (2)



Disconnected Operation

- HOARDING: File cache in advance with all files that will be accessed when disconnected
  - Best effort
- EMULATION: when disconnected, behavior of server emulated at client
- REINTEGRATION: transfer updates to server; resolves conflicts

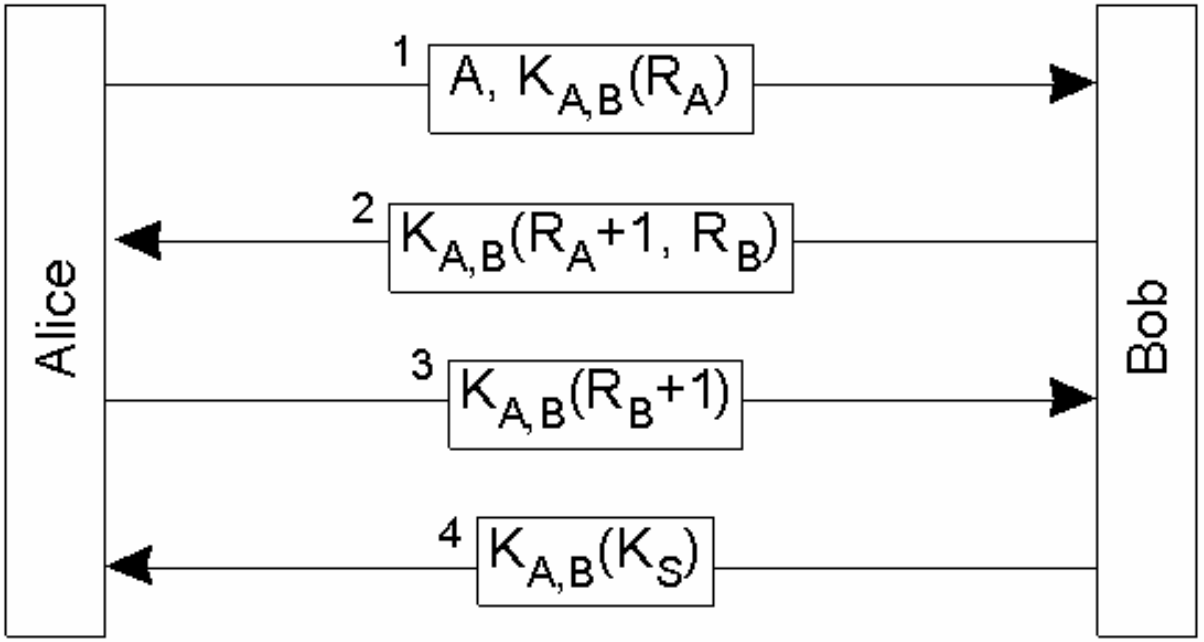


Security

- Set-up a secure channel between client and server
  - Use secure RPC
- System-level authentication

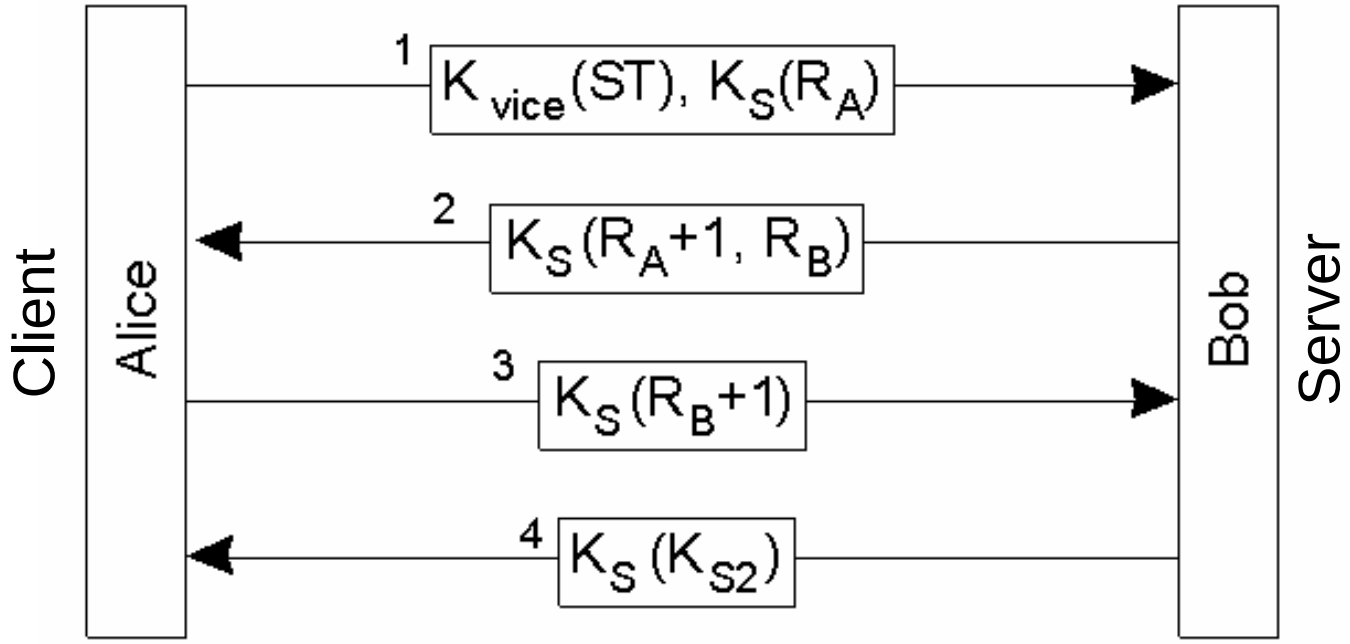
Mutual Authentication in RPC2

- Based on Needham-Schroeder protocol



Establishing a Secure Channel

- Upon authentication AS (authentication server) returns:
  - Clear token: CT = [Alice, TID, K_S, T_{start}, T_{end}]
  - Secret token: ST = $K_{vice}([CT]^*_{Kvice})$
  - K_S: secret key obtained by client during login procedure
  - K_{vice}: secret key shared by vice servers
- Token is similar to the ticket in Kerberos



NFS vs Coda

Issue	NFS	Coda
Design goals	Access transparency	High availability
Access model	Remote	Up/Download
Communication	RPC	RPC
Server groups	No	Yes
Mount granularity	Directory	File system
Name space	Per client	Global
Sharing sem.	Session	Transactional
Cache consist.	write-back	write-back
Fault tolerance	Reliable comm.	Replication and caching
Recovery	Client-based	Reintegration
Secure channels	Existing mechanisms	Needham-Schroeder