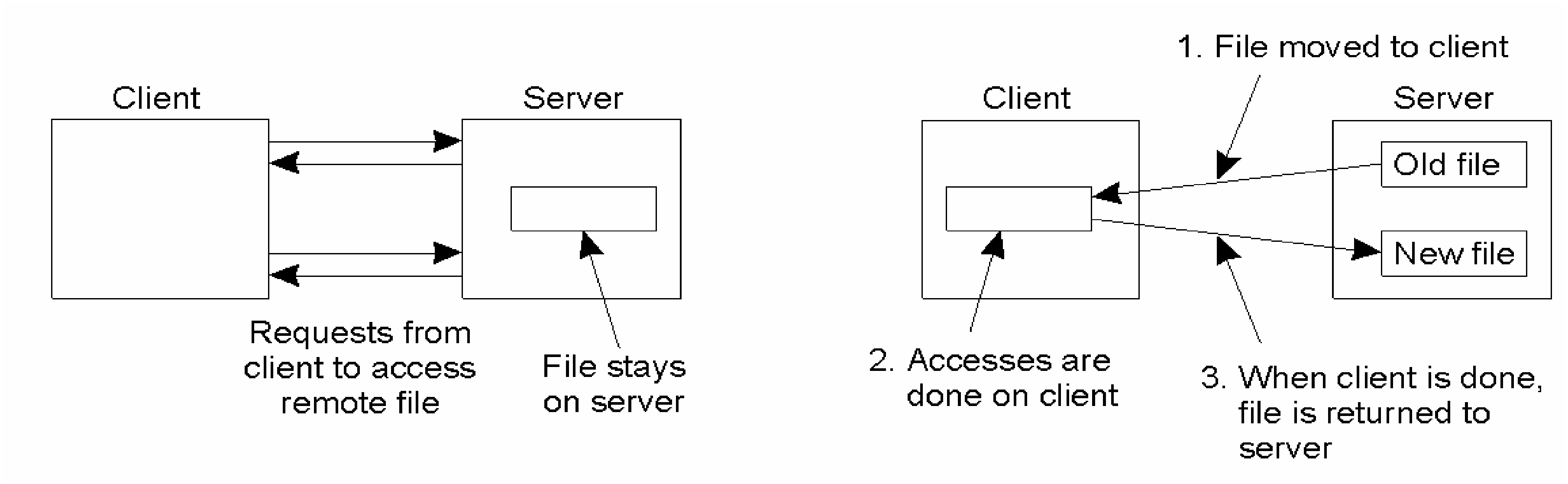
Distributed Systems

9. Distributed File Systems and Overlay Networks (CDN and P2P)

Importance of DFS

- Oldest and most important kind of Dist. Sys.
- NFS (Network File System, SUN)
  - Set of protocols providing a unified view to the FS, no matter how the local implementation works
  - Oldest DFS with largest installation base
- Coda (Carnegie Mellon University)
  - Successor of AFS (Andrew FS), main goal: scalability
  - Works even if some (mobile) clients are disconnected
  - Transactional-like session semantics
- xFS (UC at Berkely, NOW project)
  - Has no servers, clients implement the DFS
  - Not to confuse with XFS (Silicon Graphics)

NFS

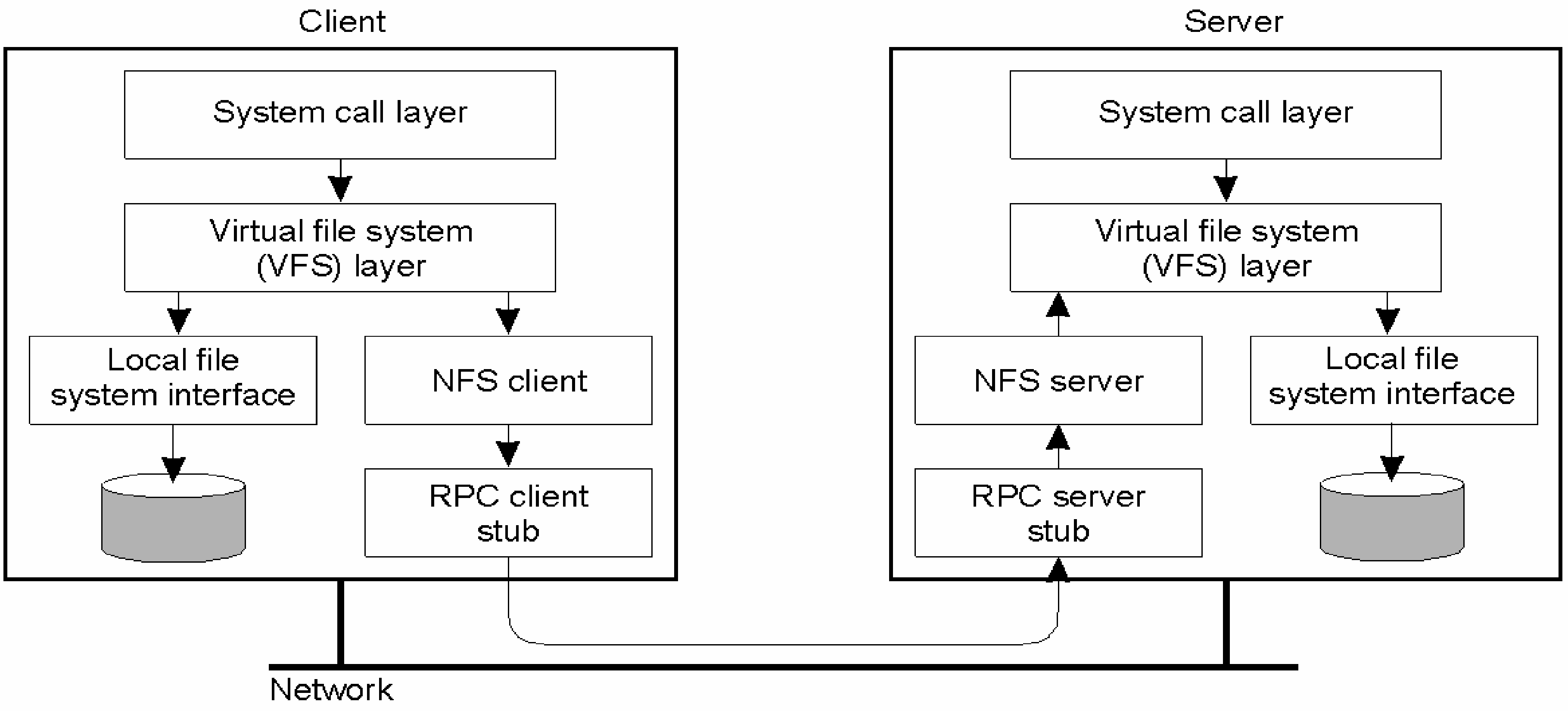

Client
Server
Requests from client to access remote file
File stays on server
1. File moved to client
Client
Server
Old file
New file
2. Accesses are done on client
3. When client is done, file is returned to server

The remote access model

The upload/download model

- The upload/download model copies files into the local FS (e.g. FTP)
- The remote access model (NFS) leaves the files on the server
  - Can be combined with *caching*

NFS Architecture


Client
Server
System call layer
Virtual file system (VFS) layer
Local file system interface
NFS client
RPC client stub
System call layer
Virtual file system (VFS) layer
NFS server
Local file system interface
RPC server stub
Network

- The VFS layer hides the differences and locations
  - A client request is automatically forwarded to the proper FS
  - The local files systems may be different (MSDOS, Windows, Unix …)

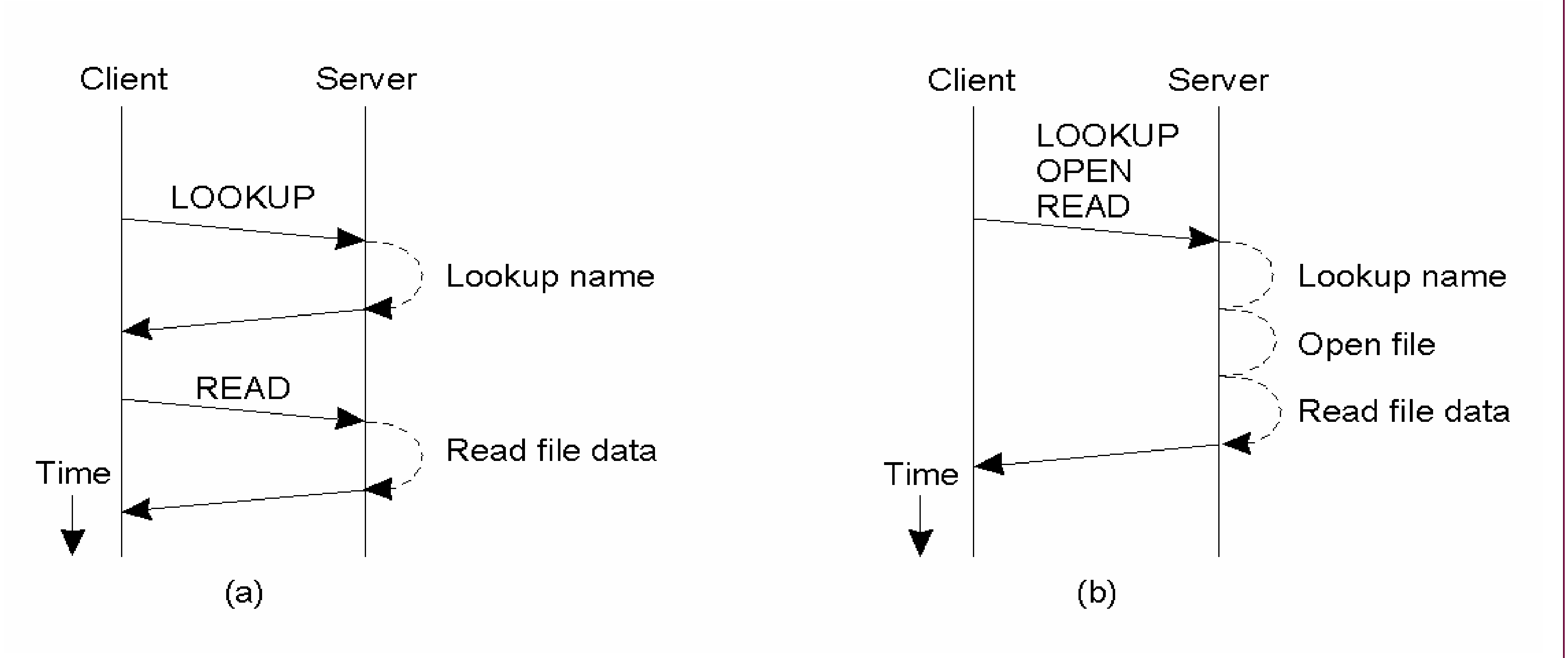
Unix-like File System Model

Operation	v3	v4	Description
Create	Yes	No	Create a regular file
Create	No	Yes	Create a nonregular file (incl. symbolic links, directories, special files)
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 (device files, sockets, named pipes)
Rename	Yes	Yes	Change the name of a file
Remove	Yes	Yes	Decrement hard link count, remove file, if cnt=0 (also directories in v4)
Open	No	Yes	Open a file (or create a new file) with certain attributes (e.g. write)
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 (n bytes at given offset)
Write	Yes	Yes	Write data to a file (n bytes at given offset), maybe also stable storage

States

- NFS used originally (up to v3) stateless servers
  - ➢ Makes fault tolerance easy
  - ➢ Usual file system semantics is sometimes hard
  - ➢ Locking and concurrency control are hard
- In NFS v4 (a few) states were introduced
  - ➢ Explicit open necessary
  - ➢ Locking can be implemented easily
  - ➢ Completion of operations can be signaled to the client
  - ➢ Server may call back the client
  - ➢ Enables effective cache consistency protocols
    - ▪ E.g. a lease based protocol cannot be implemented over a stateless server

Communication

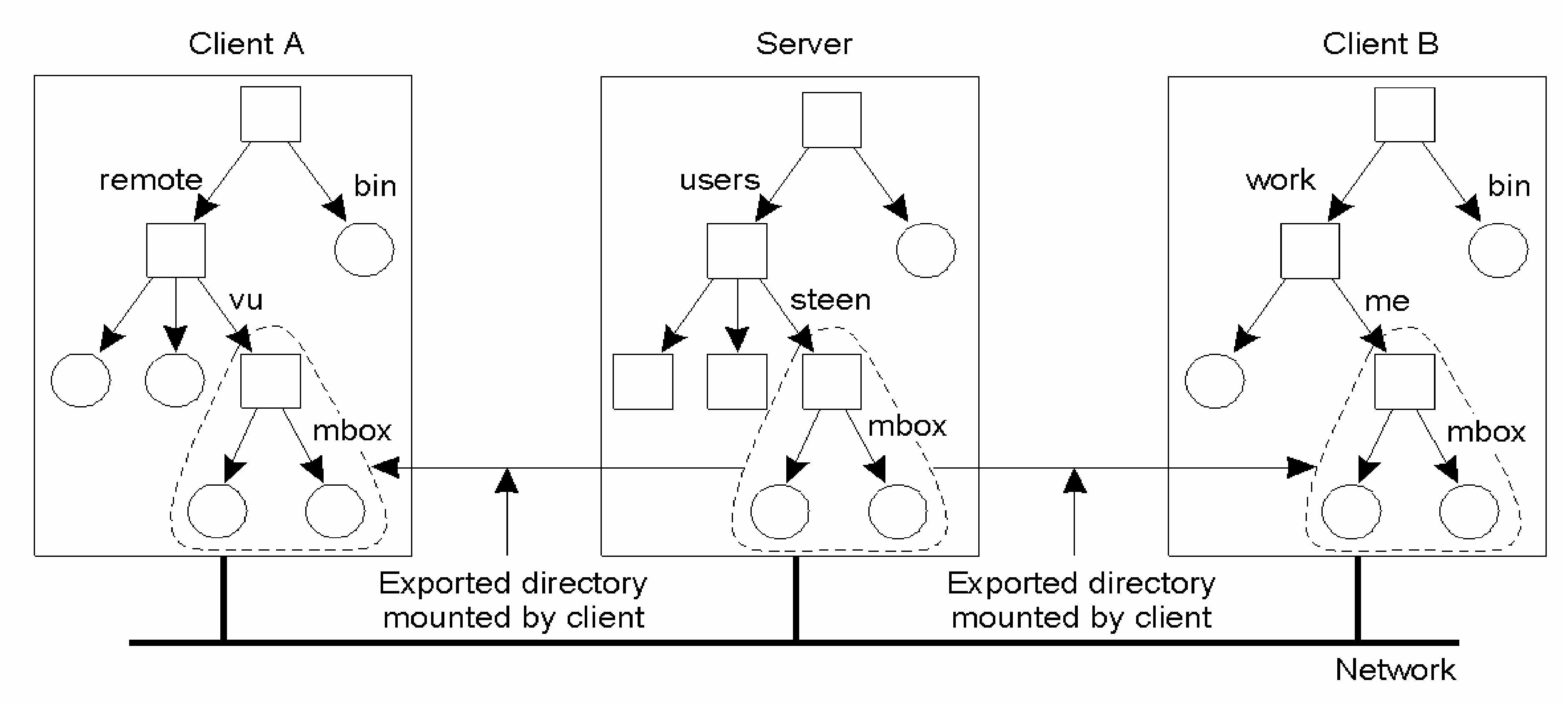


a) In NFS v3 each operation requites an own RPC

b) In v4 compound procedures can group several operations

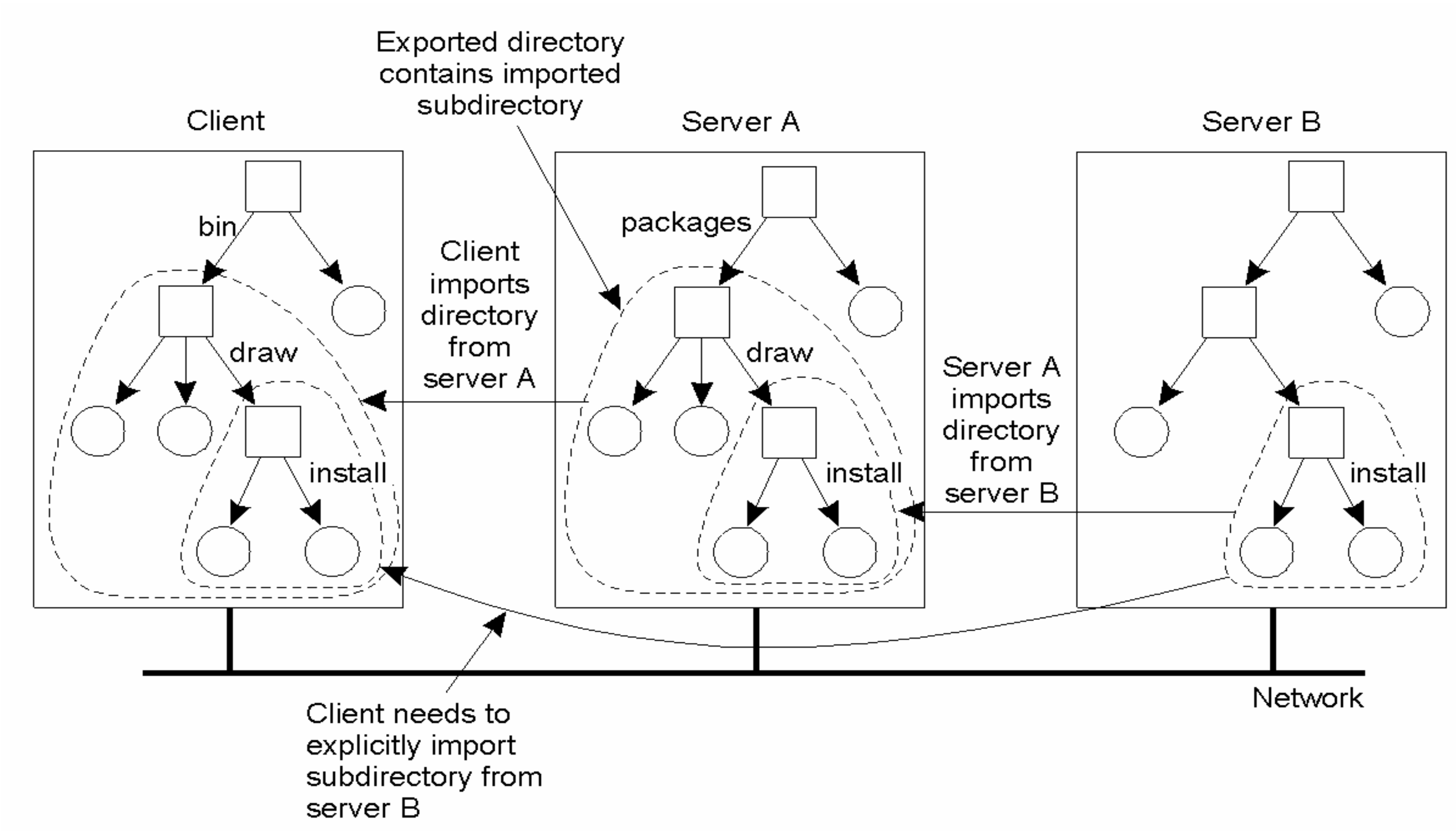
- ➢ Concurrency is not handled, on failure just abort: this is *not* a transaction

Naming (1)


Client A
Server
Client B
remote
bin
vu
mbox
users
steen
mbox
work
bin
me
mbox
Exported directory mounted by client
Exported directory mounted by client
Network

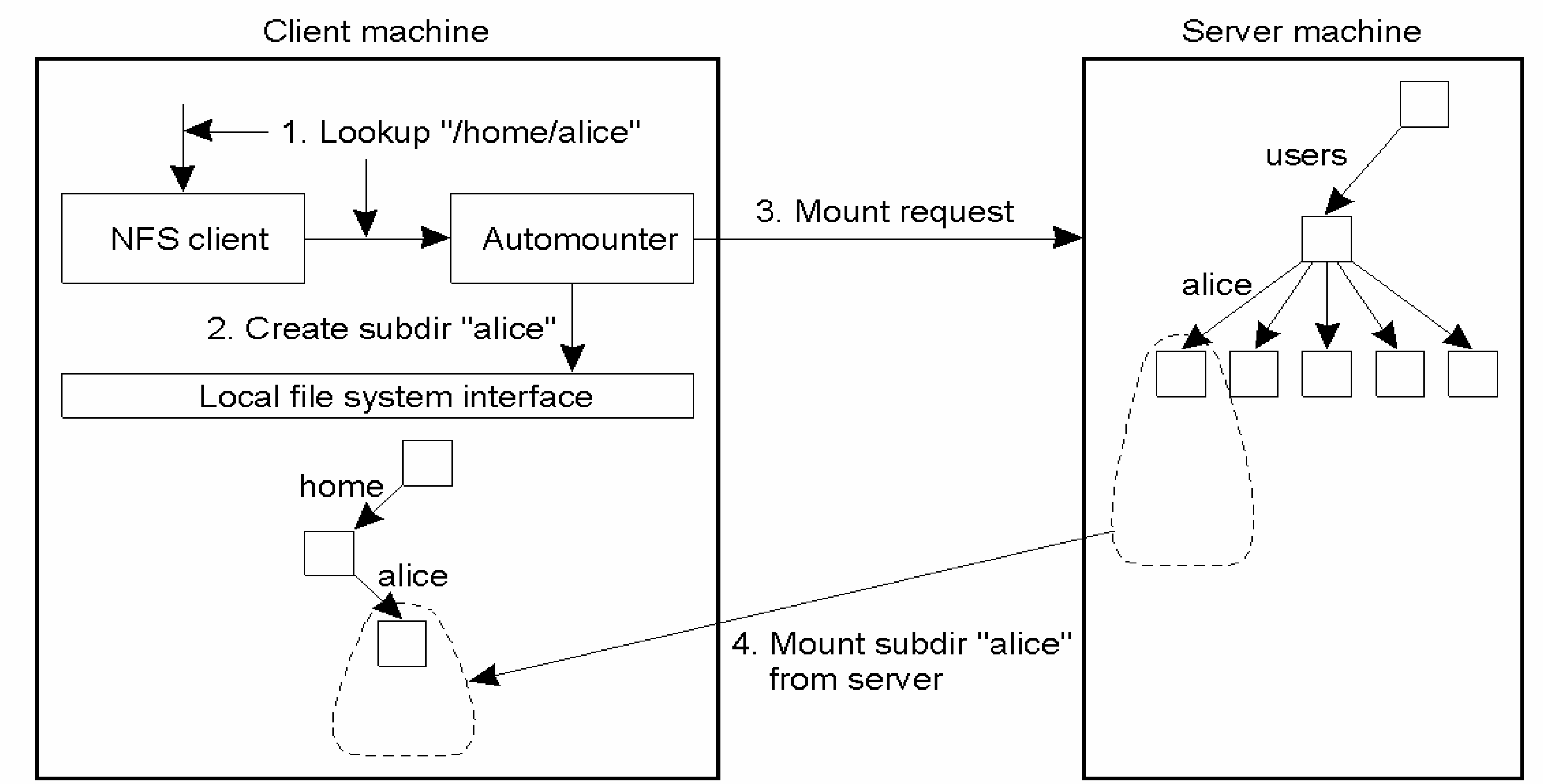
- *A* uses path “/remote/vu/mbox”, *B* “/work/me/mbox “for the same
- This makes sharing difficult – many different “names” for the same file
  - Well-defined, “standardized” naming conventions can help (e.g. /usr/bin)

Naming (2) – nested mounting


Exported directory contains imported subdirectory
Client
Server A
Server B
bin
packages
draw
draw
install
install
install
Client imports directory from server A
Server A imports directory from server B
Network
Client needs to explicitly import subdirectory from server B

- In v3, name resolution is iterative, i.e. the client has to handle nesting
- In v4, recursive path resolution and lookup is supported

Automounting



- Directories can be mounted on demand, via *automounting*
  - On each access to e.g. "home", a lookup is forwarded to the *automounter* process
  - The automounter creates dir. "alice" locally and mounts the remote dir. "alice" to it

File Attributes (1) – Mandatory attributes

Attribute	**Description**
TYPE	The type of the file (regular, directory, symbolic link)
SIZE	The length of the file in bytes
CHANGE	If and/or when the file has changed
FSID	Server-unique identifier of the file's file system
. . .	

- In v3 the set of attributes was fixed
- In v4 two sets of attributes
  - ➢ Mandatory (12 altogether) and recommended (43 altogether)

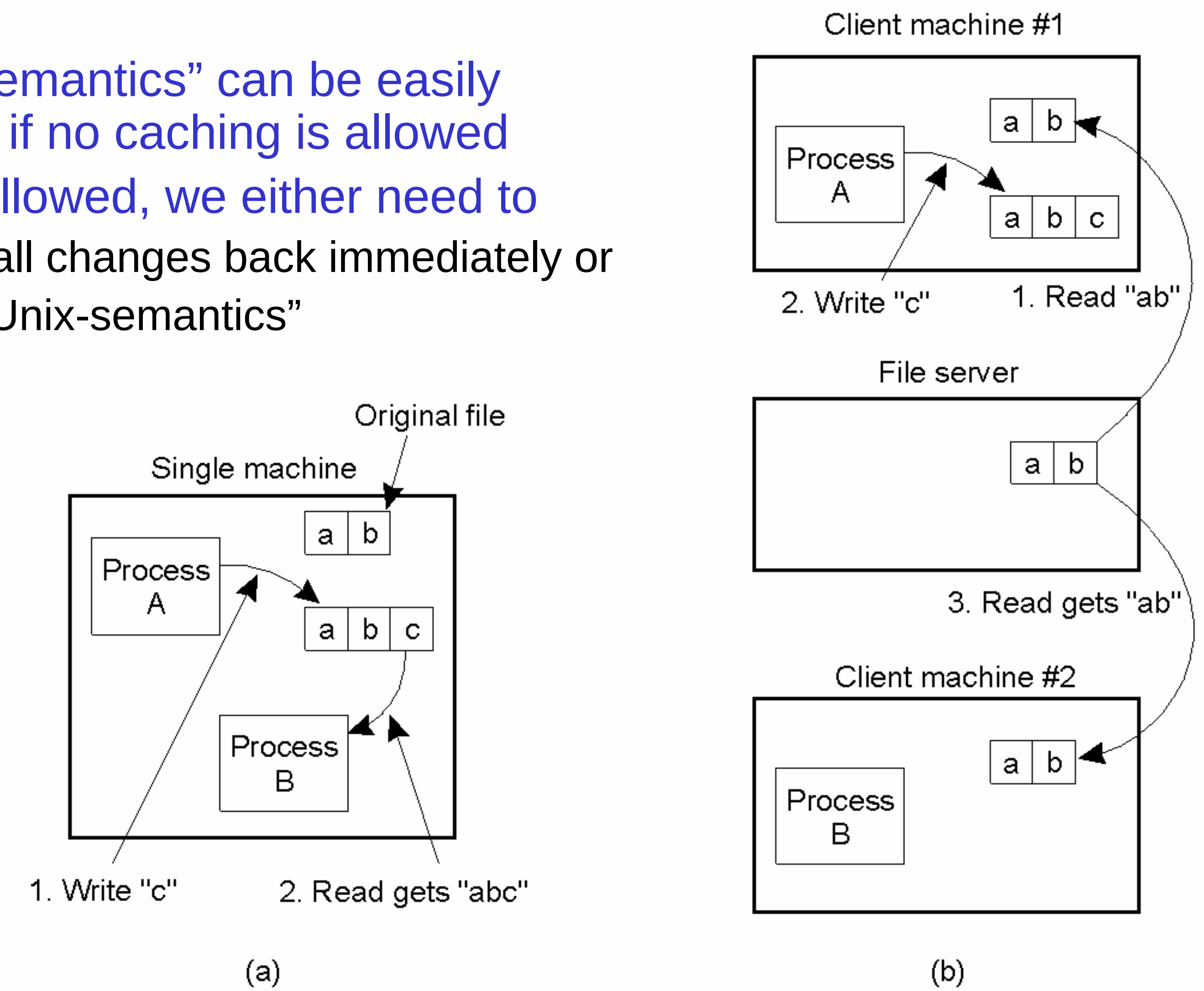
File Attributes (2) – Recommended attributes

Attribute	**Description**
ACL	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	Network-locations 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
. . .	

- Attributes are sent as an array of pairs:
  (attribute: string, value: sequence-of-bytes)
- The application is free to interpret them

Semantics of File Sharing (1)

- Usual “Unix semantics” can be easily implemented, if no caching is allowed
- If caching is allowed, we either need to
  - ➢ Propagate all changes back immediately or
  - ➢ Relax the “Unix-semantics”



Semantics of File Sharing (2)

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

- NFS implements session semantics (as many other DFS)
  - ➢ In case of concurrent accesses, the most recently closed value wins
  - ➢ Concurrency can be restricted by locking
- Immutable files: Files cannot be changed, but directories can
  - ➢ On any change a new file must be created
  - ➢ The new file may replace the old file (under the same name)
  - ➢ Concurrent replacement is still a problem – similar as by session semantics

File Locking in NFS via lock operations

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

- In a stateless server locking is complicated – extra daemon process
- In v4 states make locking simpler
  - ➢ Different locks for read and write (e.g. for many readers 1 writer)
  - ➢ Locks are granted for a required byte range for a specific time (lease)
  - ➢ Locks are removed, if the lease expires
  - ➢ Failure of server or client remains a problem

File Locking in NFS via sharing (for Windows)

Current file denial state

Request access

	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Fail
WRITE	Succeed	Succeed	Fail	Fail
BOTH	Succeed	Fail	Fail	Fail

Requested file denial state

Current access state

(by an other)

	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Fail
WRITE	Succeed	Succeed	Fail	Fail
BOTH	Succeed	Fail	Fail	Fail

- At opening the client may specify
  - ➢ Type of access (read, write, both)
  - ➢ Type of denial against others (none, read, write, both)

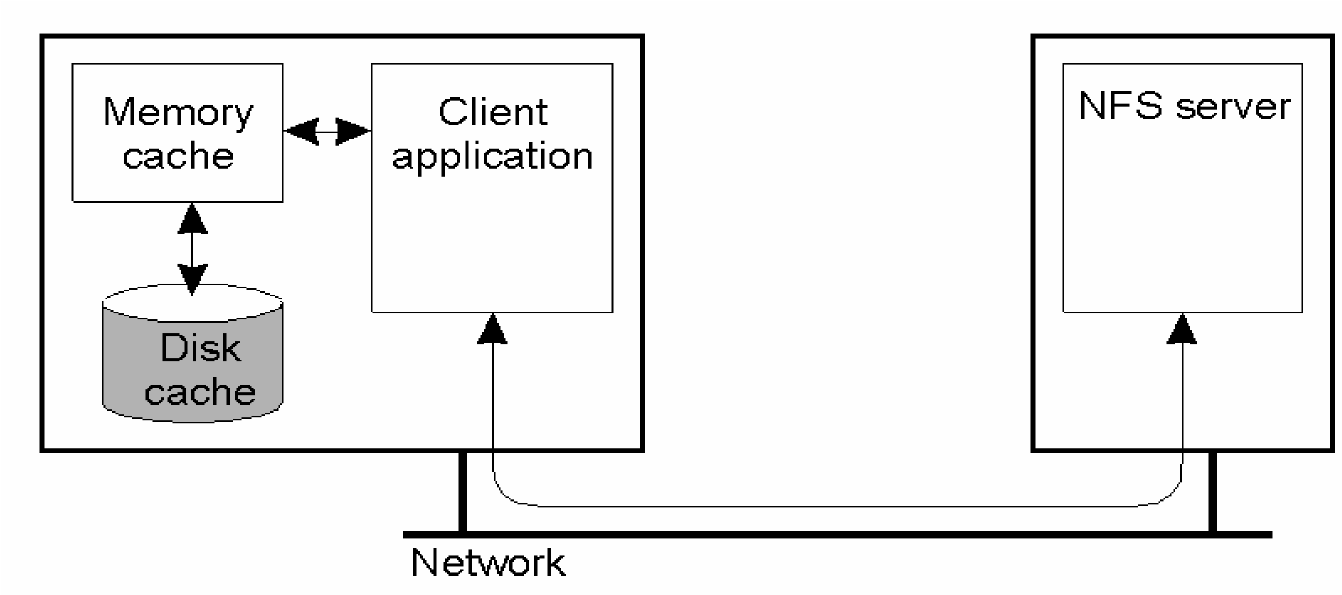
Client Caching (1)

- In v3 caching is implementation dependent
- In v4 on-demand and delegated caching

1. On-demand caching

- The blocks, read by the client, are cached
- Updates must be propagated back, at the end of the session
- On opening a file, data in cache maybe invalidated (revalidation)

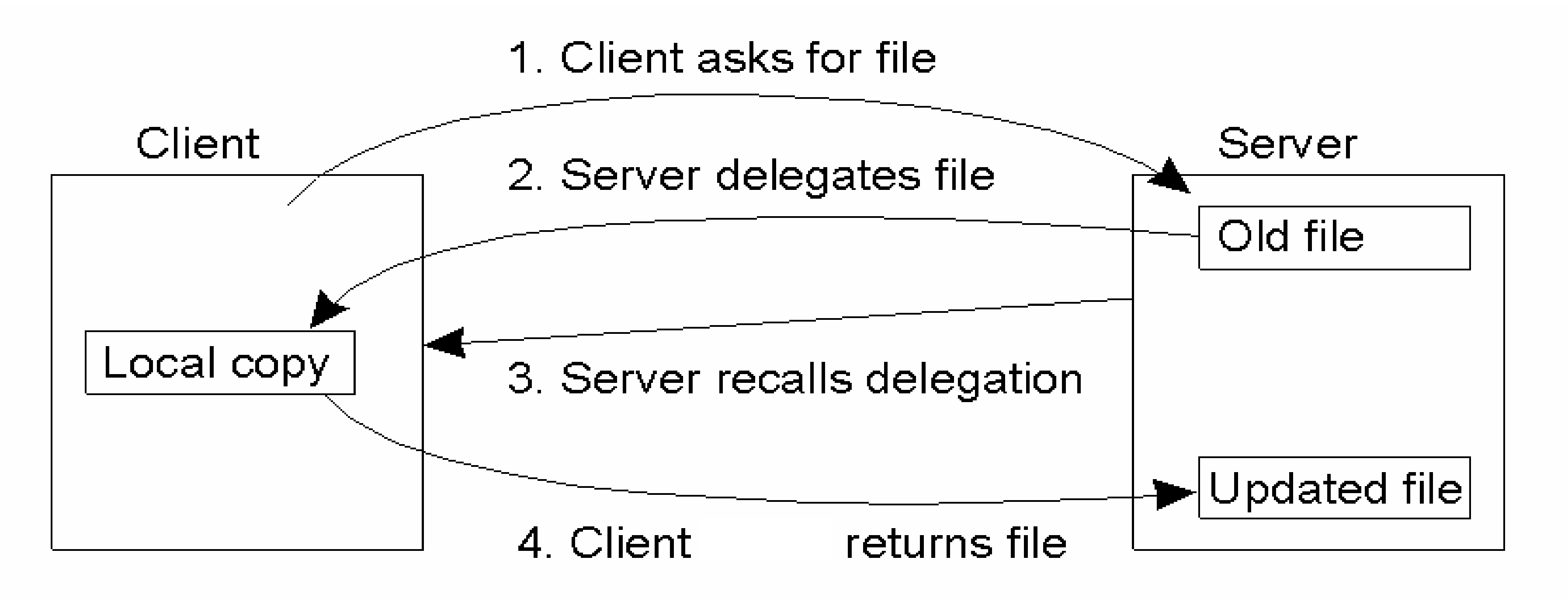
Data, attributes and handles may be cached



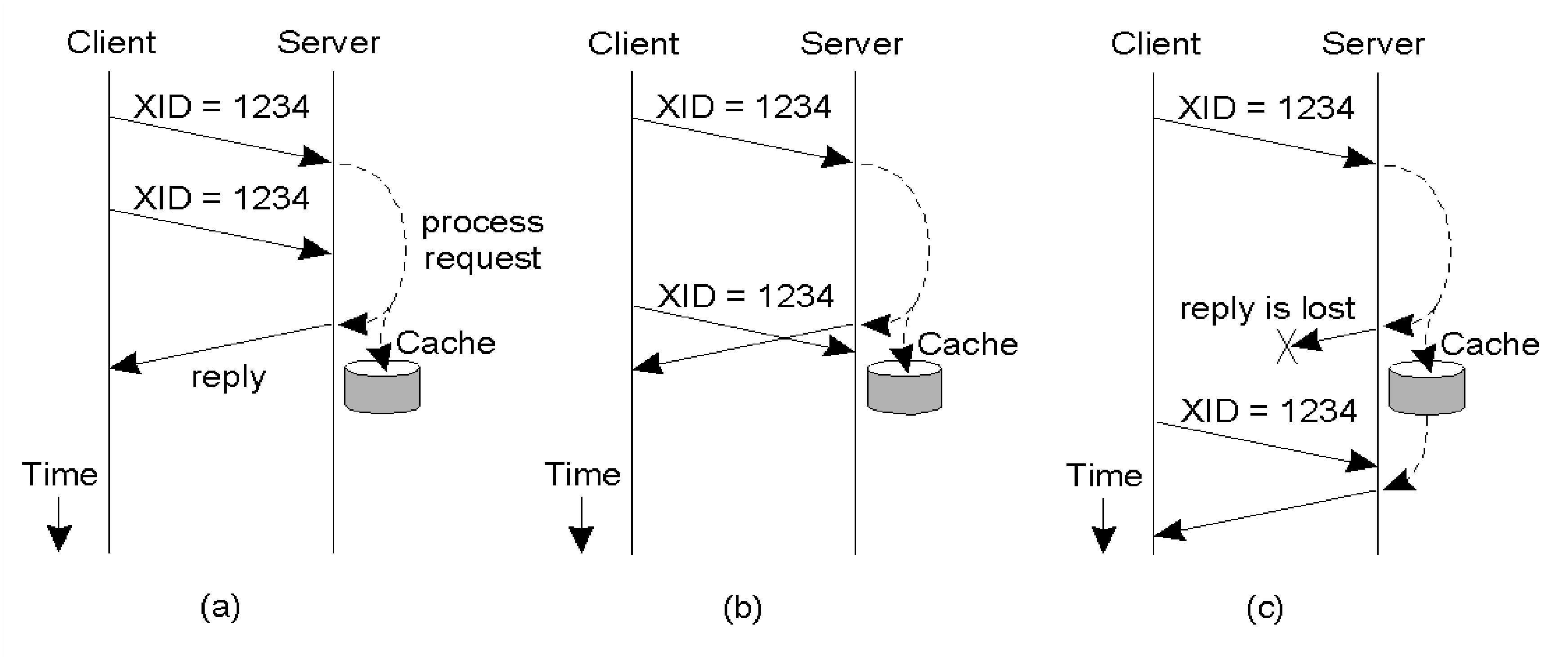
Client Caching (2)

2. Delegated caching

- The server may delegate – and later recall – some rights
- Open delegation: the client may handle *open* and *close* for others
  - E.g. file locking on the same node can be handled in this case locally
  - Remote locks must be refused



NFS Reliability (1)



- NFS does not specify the exact RPC semantics
  - ➢ Implementation over TCP or UDP are possibke (duplicates may occur)
  - ➢ Client requests get a unique “transaction identifier” (XID)
  - ➢ Servers store client requests and own replies for a while in a *duplicate-request cache*

a) The request is still in progress – ignore the same request

b) The reply has just been returned – ignore the same request

c) The reply has been sent some time ago, but was lost – resend the answer

NFS Reliability (2)

- File locking and failures
- With stateful servers fault tolerance is harder
- Client – holding a lock – crashes
  - ➢ The grants expire after a certain time (lease)
  - ➢ Server removes a lock, unless client renews the lease
  - ➢ The lack of global time makes expiration-handling hard
  - ➢ False removal, if renew does not reach the server
- Server – having granted locks – crashes
  - ➢ After recovering, the server accepts *lock reclaims* – and rejects all other lock requests – during a *grace period*
  - ➢ With this, it can recreate the original locking state

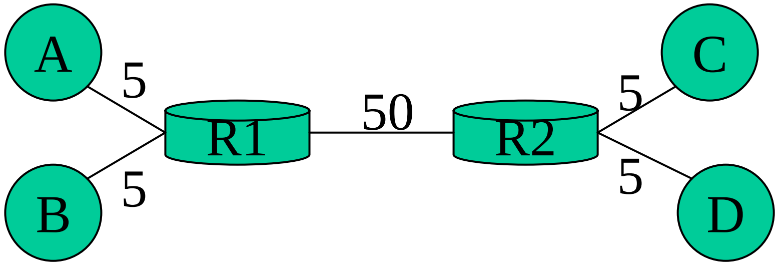
NFS Reliability (3)

- Open delegation and failures
- Client – holding open delegation – crashes
  - Full recovery will be impossible, unless the client saves all changes to stable storage
  - Client is partially responsible for the recovery
- Server – having granted open delegation – crashes
  - Similar procedure as with lock recovery
  - Clients may reclaim their open delegations
  - The server forces the clients to flush back all modifications and thus recalls the delegation
  - Clients may apply again for delegation

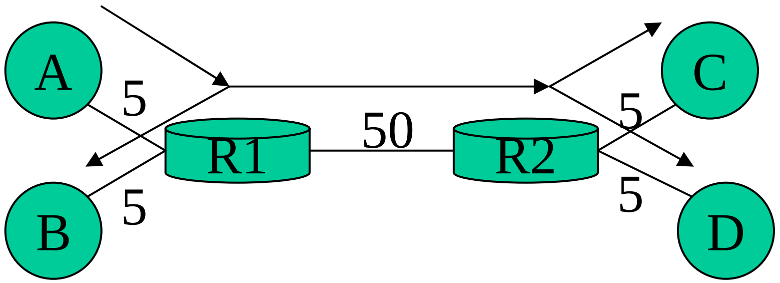
Overlay Networks (1)

- An overlay is a logical network on top of the physical network
- Routing Overlays
  - ➢ The simplest kind of overlay
  - ➢ Virtual Private Networks (VPN), supported by the routers
  - ➢ If no router support – we can apply end system routing
- End System Multicast
  - ➢ Naive multicasting wastes resources
  - ➢ End system multicast: compromise without router support
- General solution: multiple level of overlay networks
  - ➢ Web caches
  - ➢ Content Distribution Networks
  - ➢ Peer-to-Peer file sharing

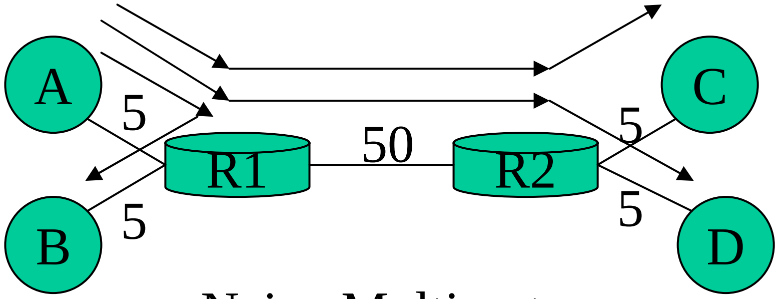
Overlay Networks (2)



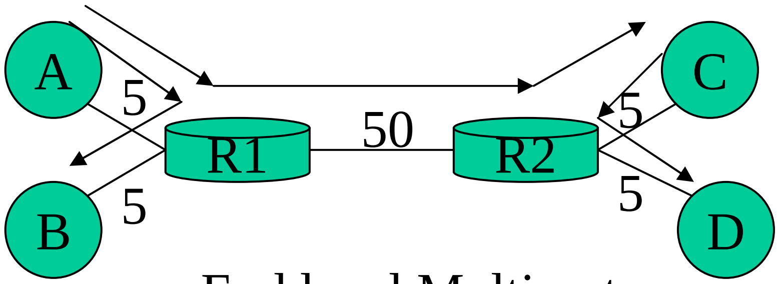
Physical topology



Router level Multicast



Naive Multicast



End level Multicast

Content Distribution Networks (1)

- The paying clients are the *content providers*, not the ISPs
  - A content provider (e.g. Yahoo) pays a CDN provider (e.g. Akamai) to get its contents to the user
- A CDN company
  - Provides many CDN servers throughout the Internet
    - Typically at an *Internet hosting center*, (e.g. Worldcom)
    - Runs often on servers of a server vendor (e.g. Inktomi)
  - Replicates the content at the CDN servers
    - When the content provider changes the content the CDN redistributes the fresh content to the CDN servers
  - Provides mechanisms that the clients get the content from the best located CDN server – relying on DNS *redirection*

Content Distribution Networks (2)

- Steps of content distribution
  1. Provider determines which objects (e.g. all its gif files) should be distributed via a CDN company (e.g. cdn.com)
  2. Provider prefixes the gif files with http://www.cdn.com
  3. The browser of the client sends the request (e.g. www.content.com/sports/goal.gif) to the original server.
  4. At parsing the HTML file it finds the link: http://www.cdn.com/ www.content.com/sports/goal.gif
  5. The browser makes a DNS lookup for www.cdn.com
  6. The DNS server sends the request to the CDN’s authoritative server, which extracts the clients IP address
  7. Using an internal network map the DNS server finds the best located CDN server (e.g. geographically the closest)
  8. The DNS in the client receives a DNS reply with the IP address of the best CDN server
  9. The browser obtains goal.gif from this
  10. For subsequent requests for www.cdn.com the same CDN server is used (IP address is in the DNS cache)

Peer-to-Peer File Sharing (1)

- Instead of installing CDN servers use the clients’ computer
  - E.g. Napster, Gnutella, Freenet, KaZaA/FastTrack, Bittorrent ...
- Usage model
  - Client Alice has installed a peer-to-peer software
  - Alice wishes to get the MP3 file for „Hey Jude“
  - She contacts the internet and asks for this file
  - She gets a list of clients storing „Hey Jude“
    - Additional information, such as bandwidth may be included
  - Alice selects Bob’s computer and loads the file
    - If Bob disconnects in between, she (her software) may select an alternative supplier
  - From now Alice is also a supplier of „Hey Jude“
- For the file transfer usually HTTP is used
  - The clients are both Web clients and transient Web servers

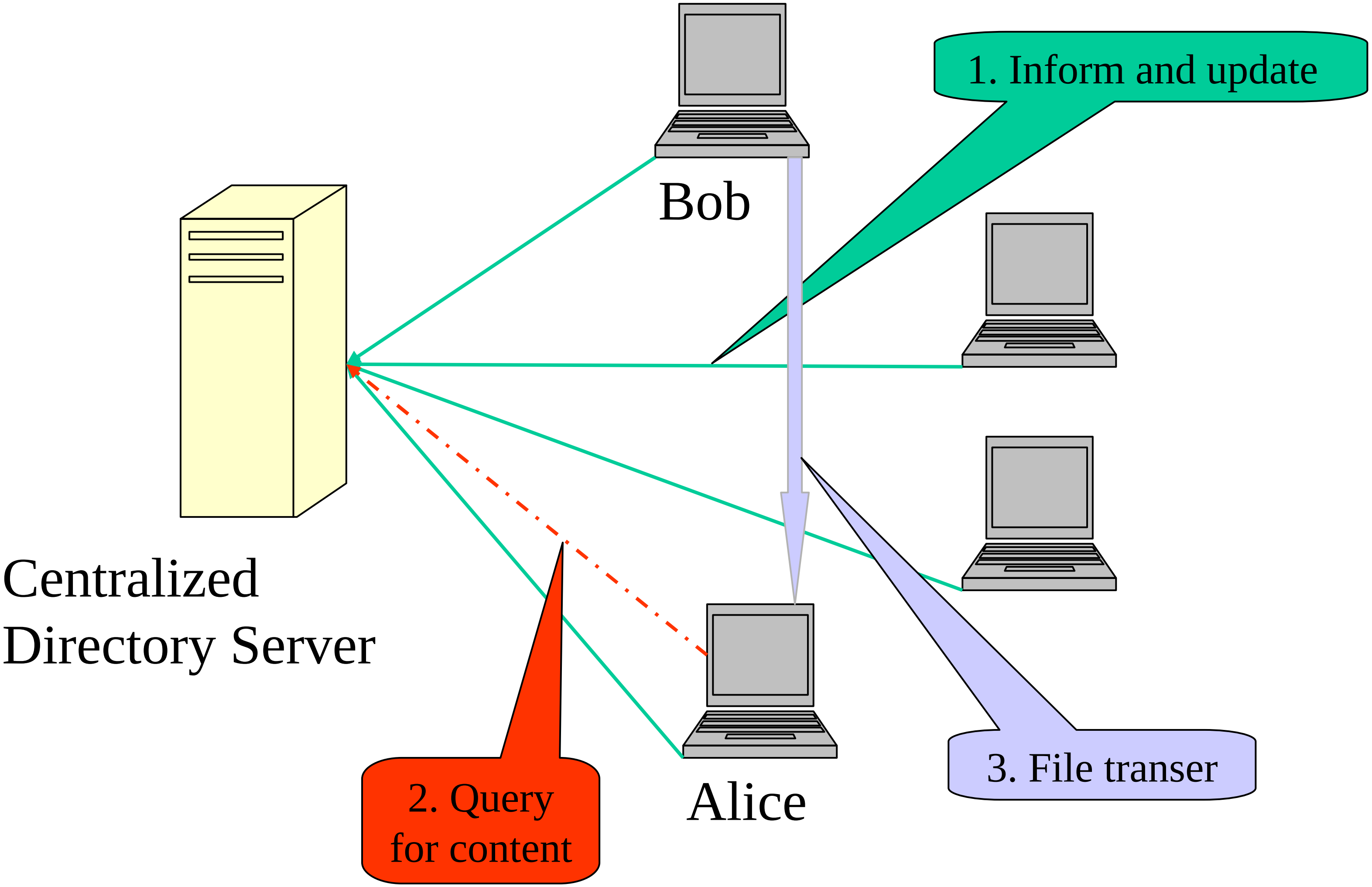
Peer-to-Peer File Sharing (2)

- Advantages
  - Unlimited resources, thousands of clients may participate
  - Scales well – at least principally
- Difficulties
  - Clients may spontaneously disconnect – bad for a server
  - How can the clients be found, who have the desired object?
  - Possible solutions
    - Centralized Directory
    - Decentralized Directory
    - Query Flooding

P2P with Centralized Directory (1)

- Simple to implement
  - Napster used this technique
- The P2P file sharing service uses a large server (farm)
- The P2P software at each client sends its P2P directory information to the server and keeps this info up-to-date
- Disconnections must de noticed at the server
  - Send probe messages to the clients or
  - Maintain a TCP connection with each peer
- Main drawbacks
  - Single point of failure
  - Performance bottleneck at the server
  - Copyright infringement
    - Easy to shut down – advantage or disadvantage?

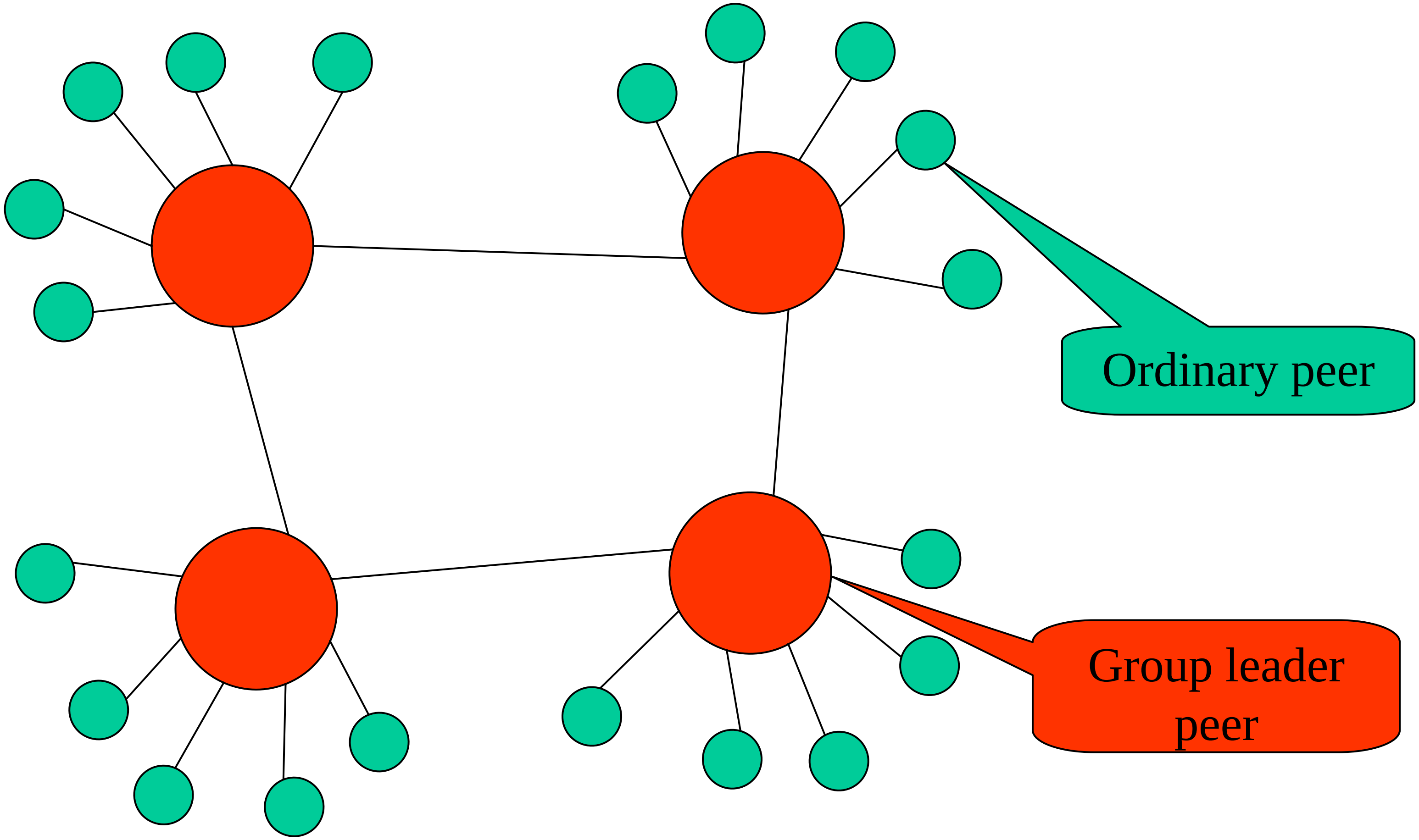
P2P with Centralized Directory (2)



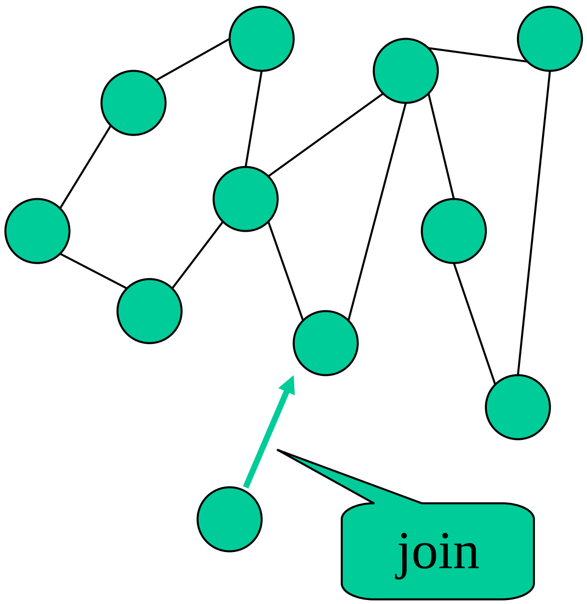
P2P with Decentralized Directory (1)

- Implementation based on hierarchical overly networks
  - ➢ KaZaA/FastTrack uses this approach
- A number of peers are designated as group leaders
- A new P2P client is assigned to a group leader
- The client gets the IP address of the leader and informs it about its own P2P contents
- The leader maintains a database of contents of its group

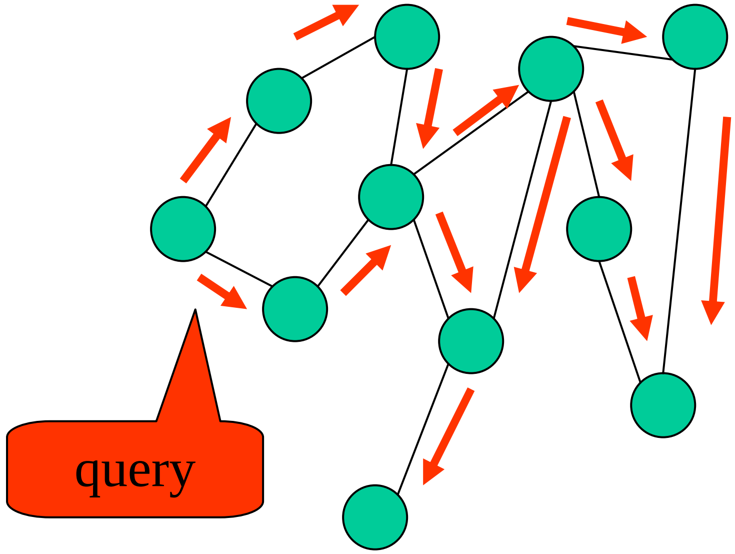
P2P with Decentralized Directory (2)



P2P with Flooding



join



query

Peer joining the overlay network

Query flooding in the network