CS G623: Advanced Operating Systems

Lecture 19

BITS Pilani

Pilani Campus

Amit Dua

Sept 17, 2018

Topics

innovate achieve lead

File Systems

Distributed File Systems

Definition:

- Implement a single file system that can be shared by all autonomous computers with heterogeneous FSs (OSs) in a distributed system

Goals:

- Network transparency
- High availability
- Performance

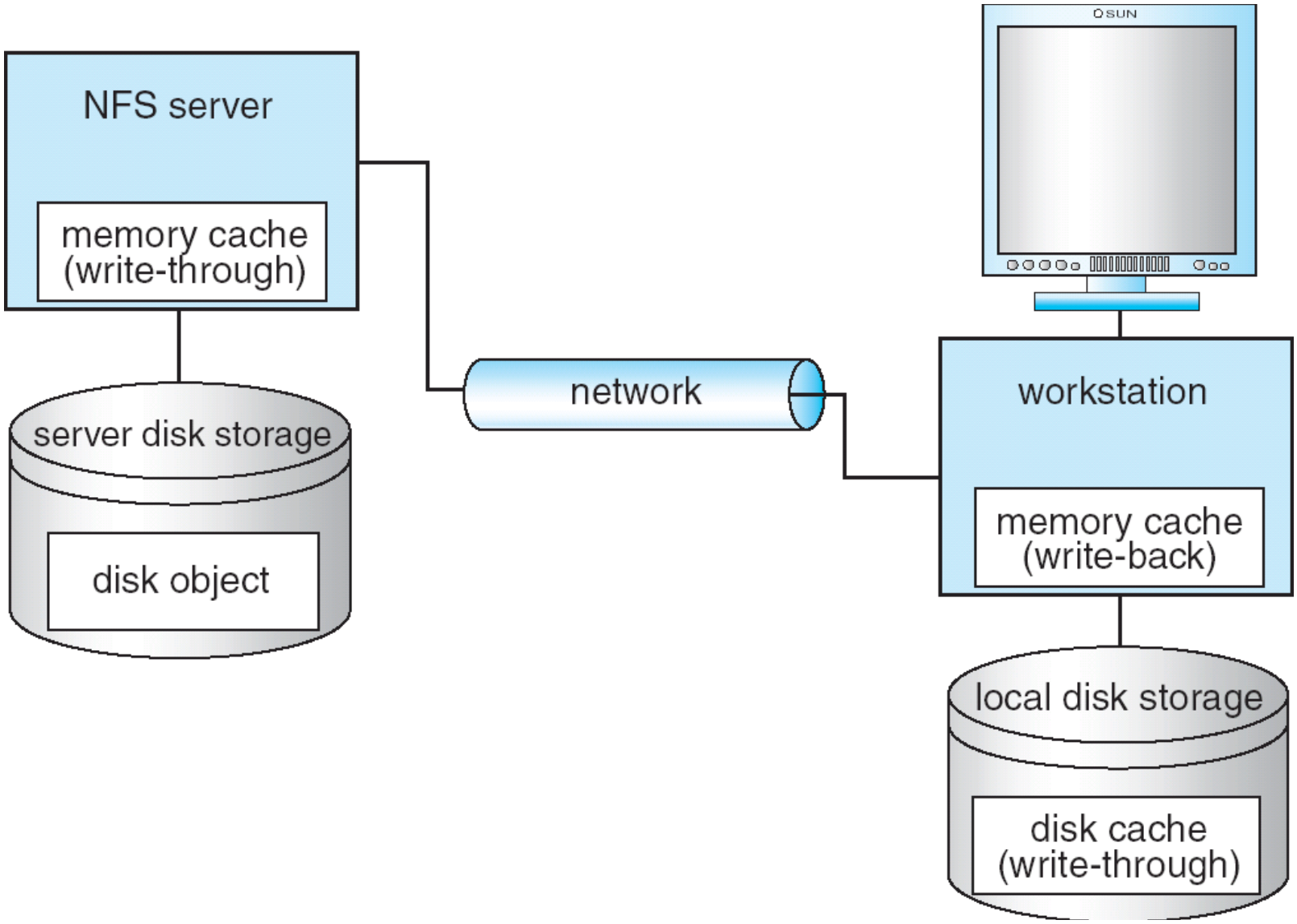
Architectural options:

- Fully distributed: files distributed to all sites
  - Issues: performance, implementation complexity
  - Ivy (CHORD DHT based P2P file system)

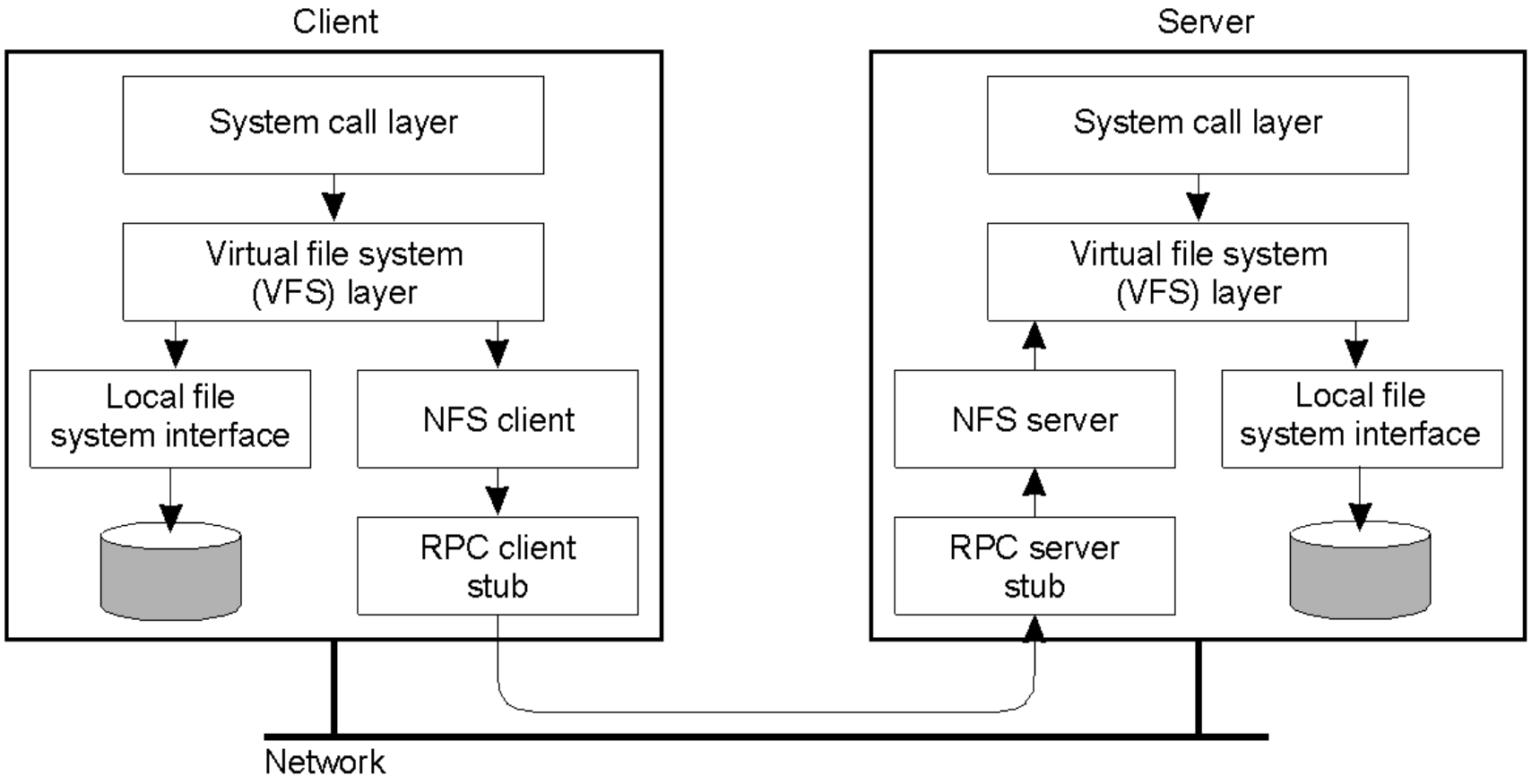
- Client-server Model:
  - Traditional (NFS), Cluster based (GFS)

Client-Server Architecture

Sun NFS

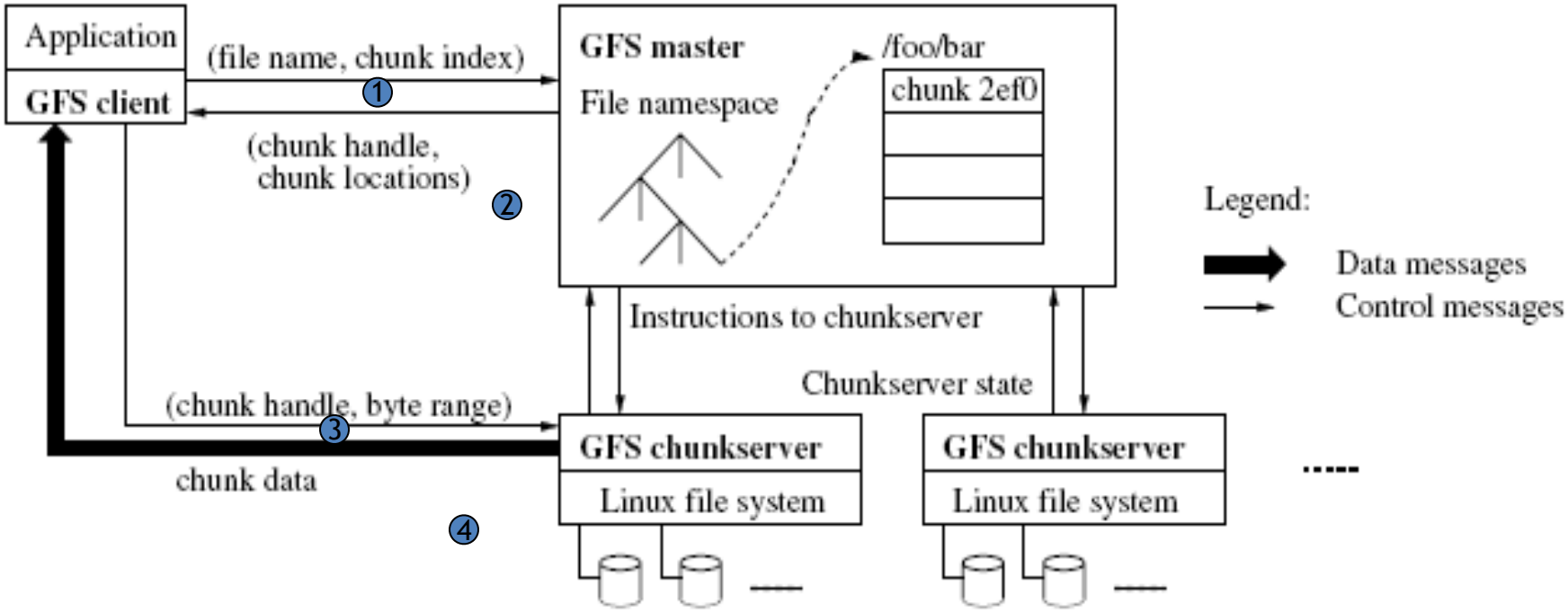

NFS server
memory cache
(write-through)
server disk storage
disk object
network
SUN
workstation
memory cache
(write-back)
local disk storage
disk cache
(write-through)

Continued…

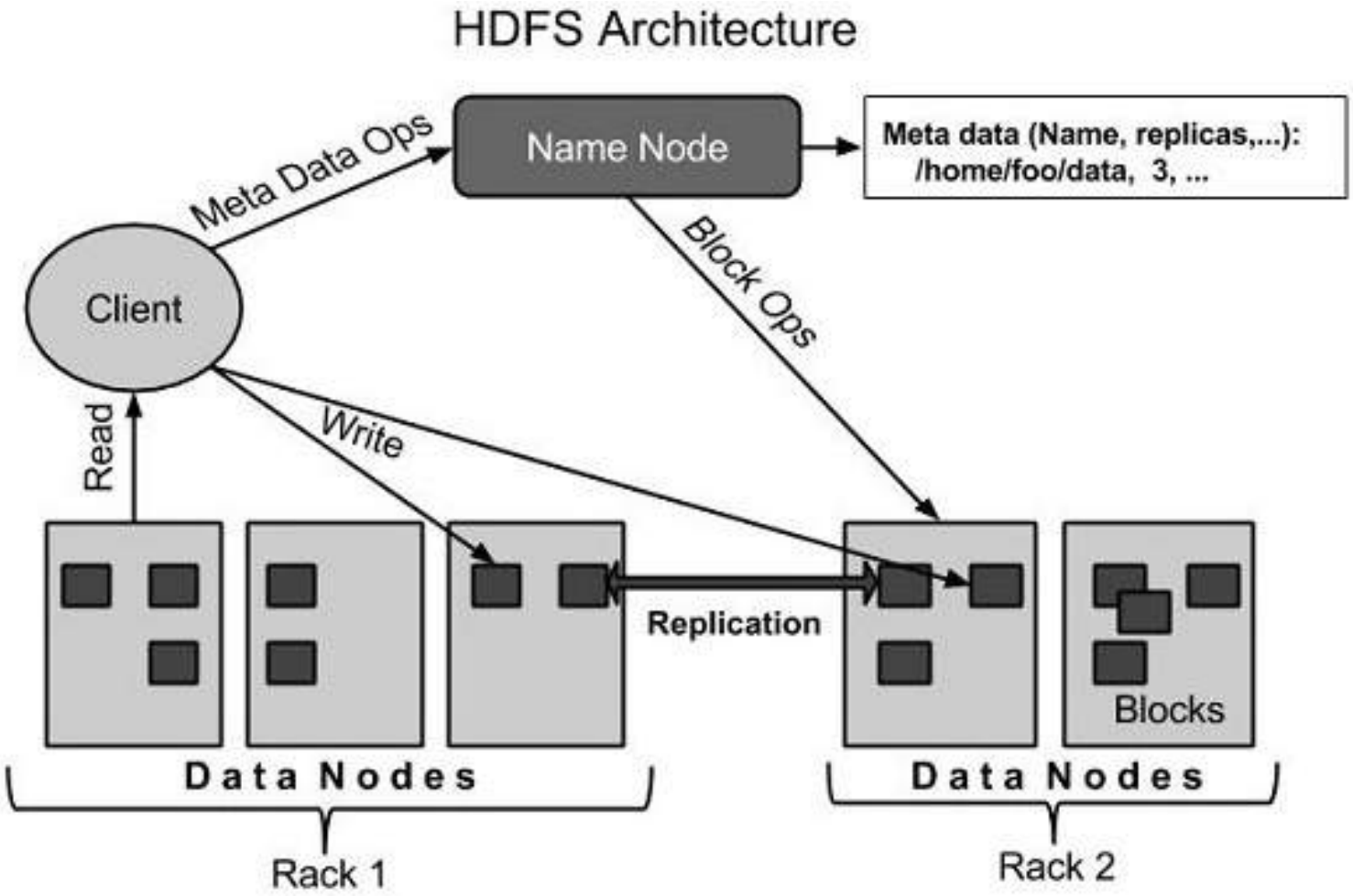


More later…

Google File System (GFS)



Hadoop distributed file system



Source: https://www.tutorialspoint.com/hadoop/hadoop_hdfs_overview.htm

Ivy P2P (Symmetric)

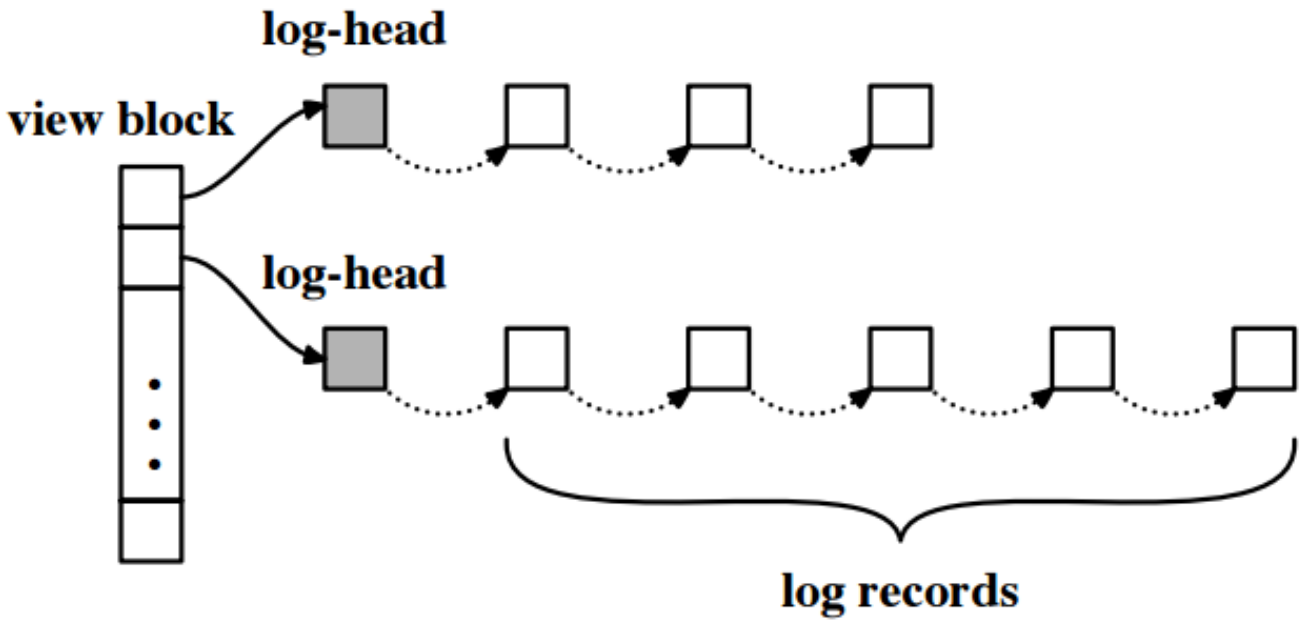
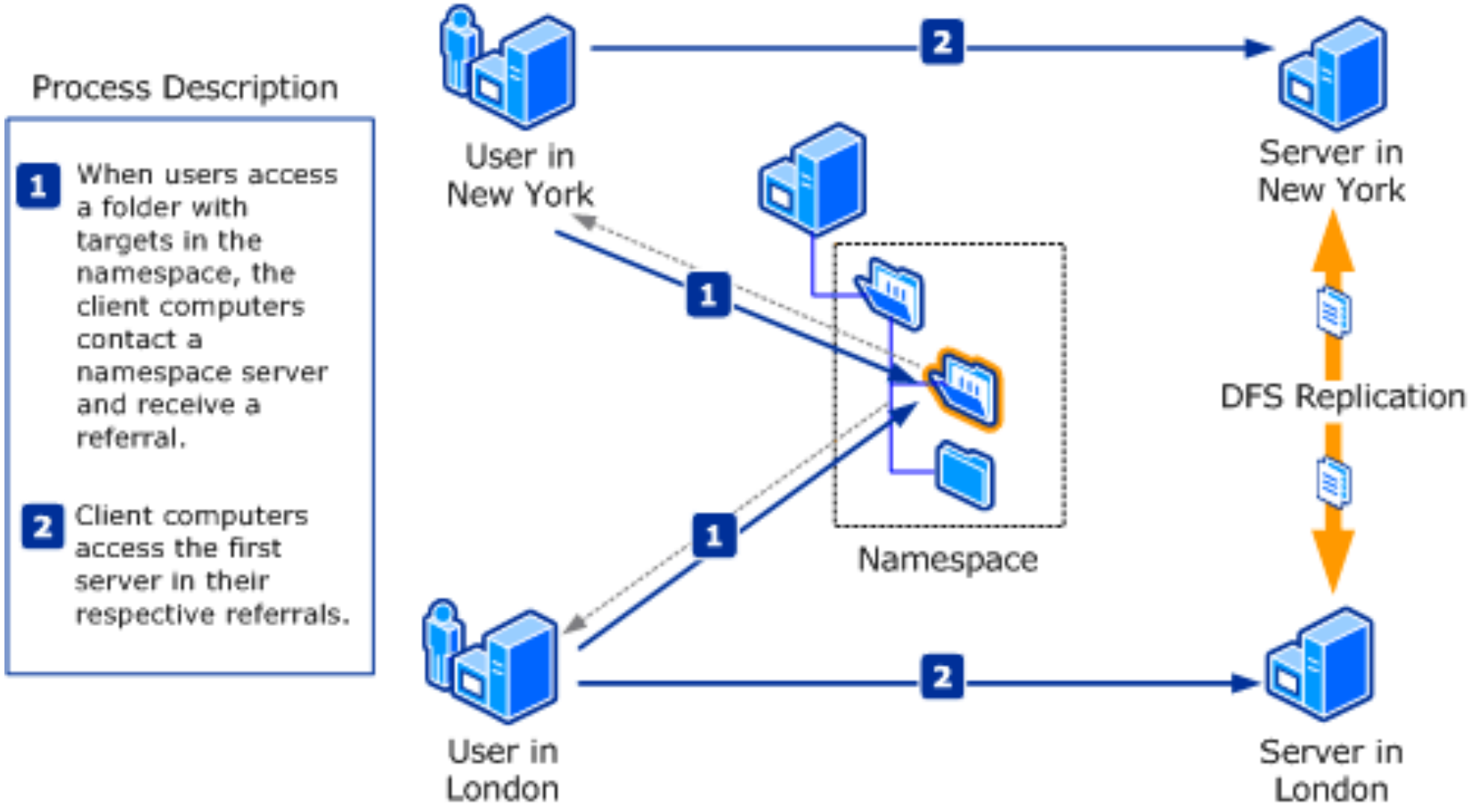


Figure 1: Example Ivy view and logs. White boxes are DHash content-hash blocks; gray boxes are public-key blocks.

Ivy: A Read/Write Peer-to-Peer File System

Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen
{athicha, rtm, thomer, benjie}@lcs.mit.edu
MIT Laboratory for Computer Science
200 Technology Square, Cambridge, MA 02139.

DFS in Win2003 R2



Distributed File Systems Services

(1) Name Server: Provides mapping (name resolution) the names supplied by clients into objects (files and directories)

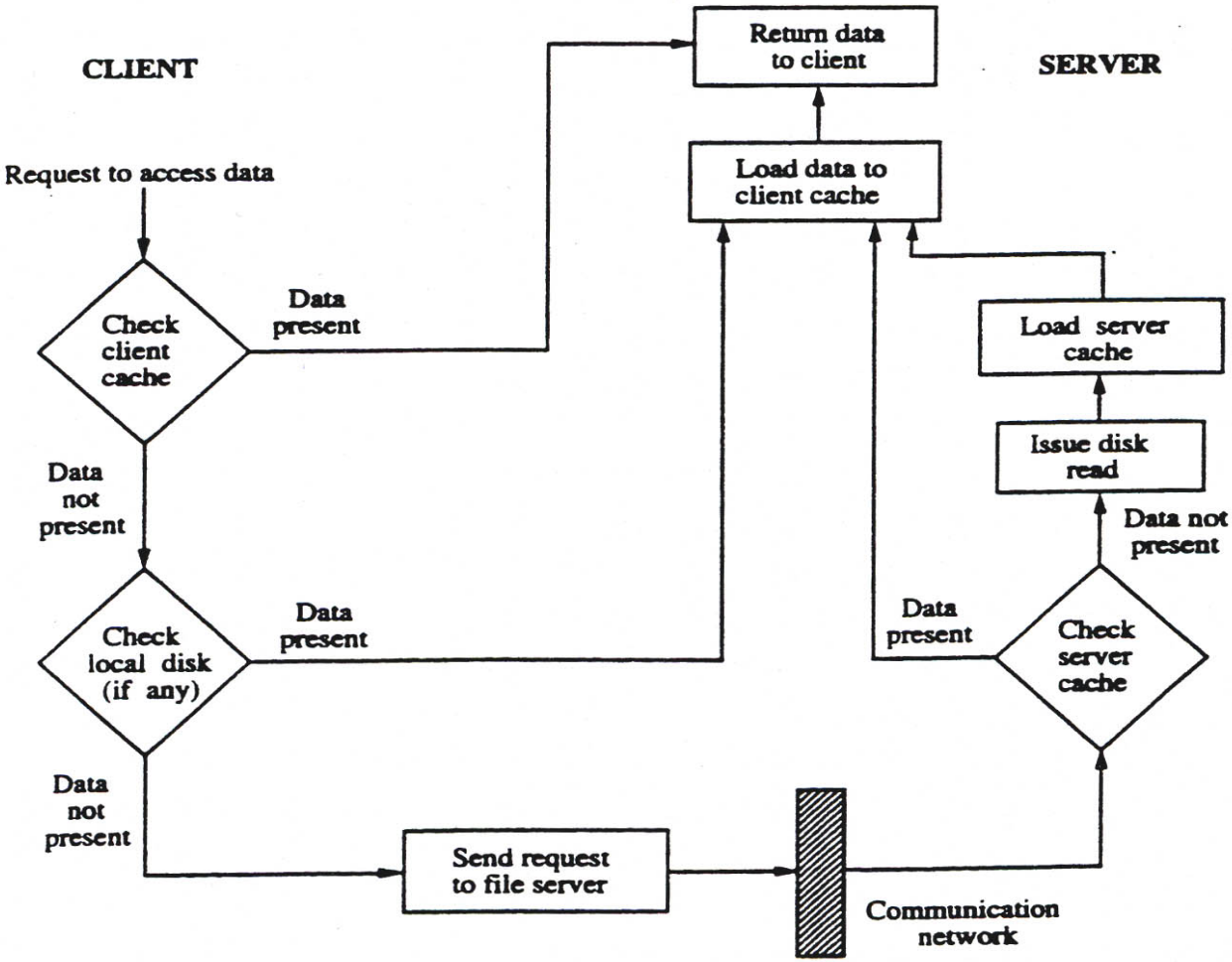
- Takes place when process attempts to access file or directory the first time.

(2) Cache manager: Improves performance through file caching

- Caching at the client - When client references file at server:
  - Copy of data brought from server to client machine
  - Subsequent accesses done locally at the client
- Caching at the server:
  - File saved in memory to reduce subsequent access time

* Issue: different cached copies can become inconsistent. Cache managers (at server and clients) have to provide coordination.

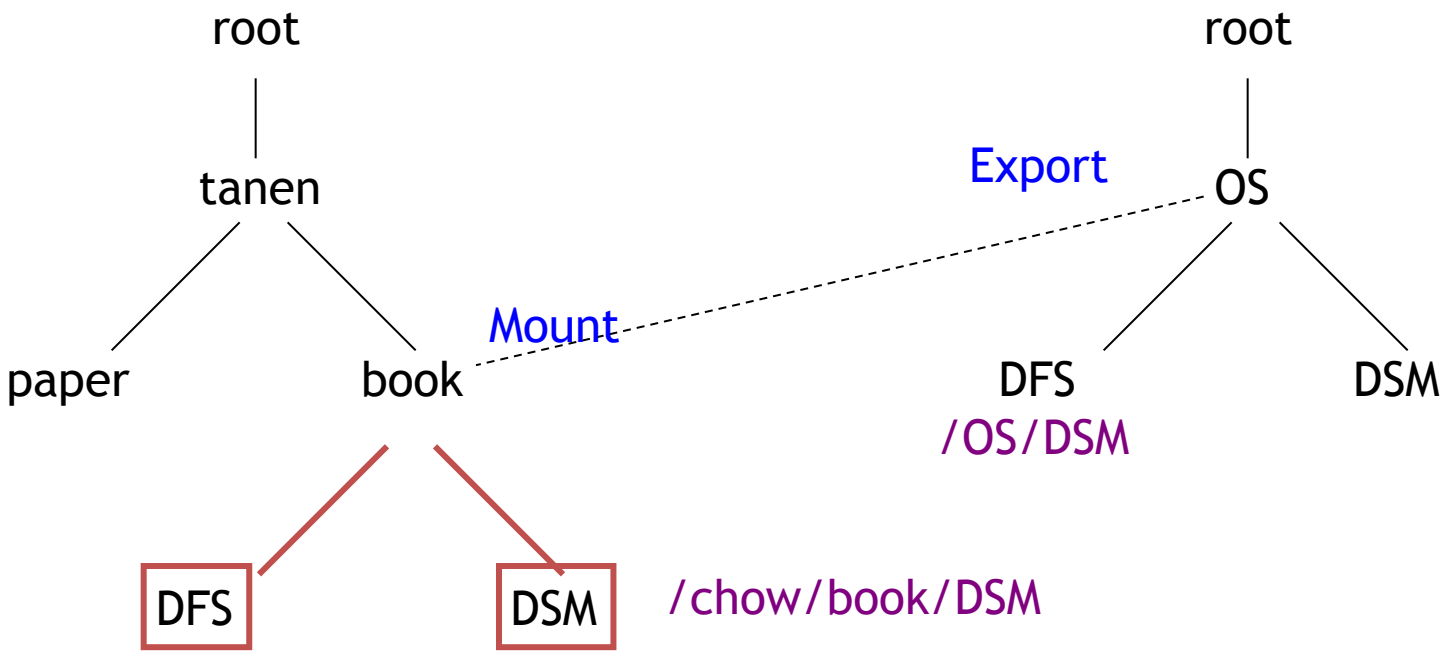
Typical Data Access in a Client/Server Architecture



Mechanisms used in distributed file systems

(1) Mounting

- The mount mechanism binds together several filename spaces (collection of files and directories) into a single hierarchically structured name space (Example: UNIX and its derivatives)



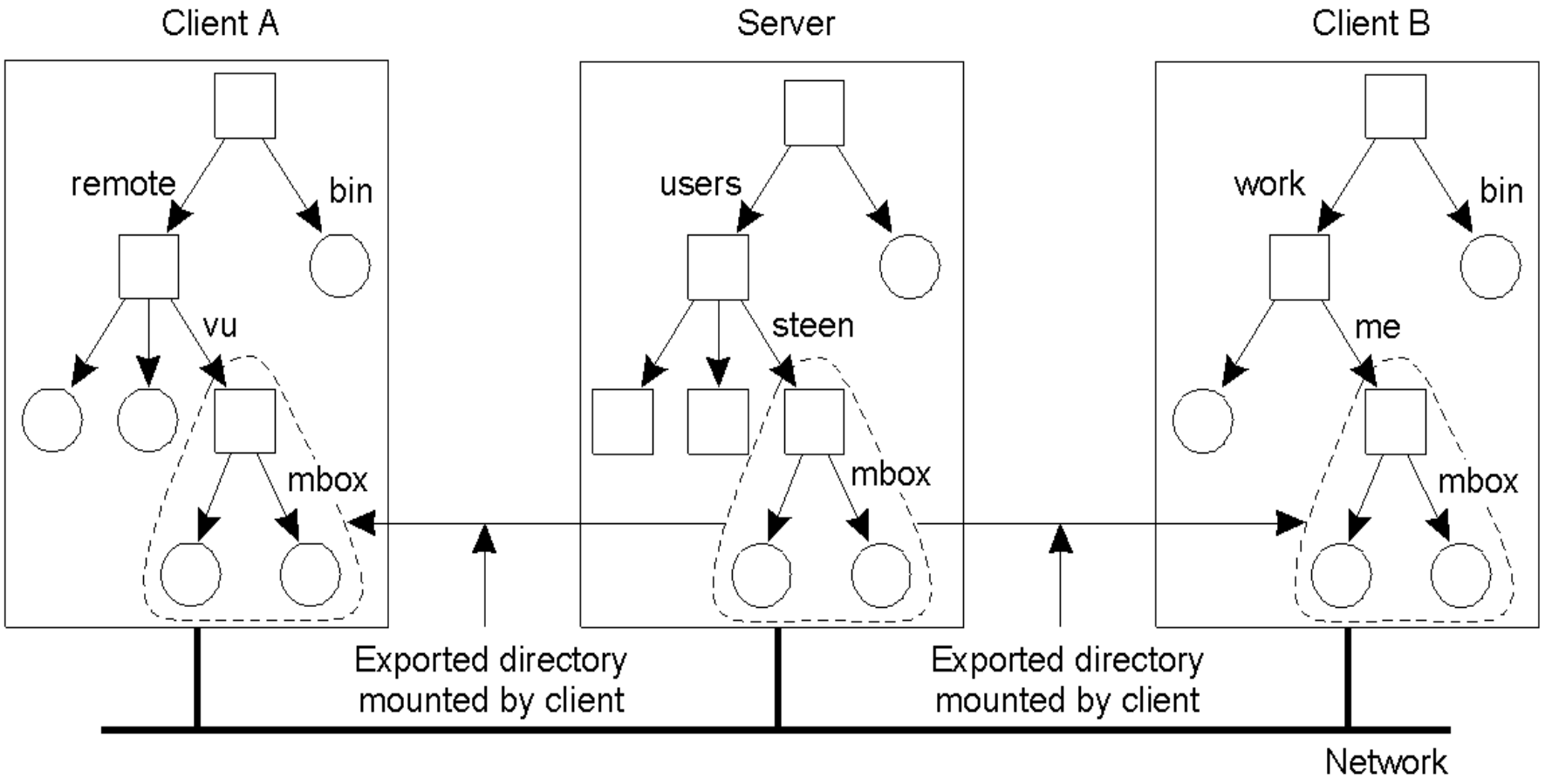
bash$ cat /etc/fstab

```
# device-spec   mount-point     fs-type      options                                          dump pass
LABEL=/         /               ext4         defaults                                            1 1
/dev/sda6       none            swap         defaults                                            0 0
none            /dev/pts        devpts       gid=5,mode=620                                      0 0
none            /proc           proc         defaults                                            0 0
none            /dev/shm        tmpfs        defaults                                            0 0

# Removable media
/dev/cdrom      /mnt/cdrom      udf,iso9660  noauto,owner,ro                                     0 0

# NTFS Windows 7 partition
/dev/sda1       /mnt/Windows    ntfs-3g      quiet,defaults,locale=en_US.utf8,umask=0,noexec     0 0

# Partition shared by Windows and Linux
/dev/sda7       /mnt/shared     vfat         umask=000                                           0 0

# mounting tmpfs
tmpfs           /mnt/tmpfschk   tmpfs        size=100m                                           0 0

# mounting cifs
//pingu/ashare  /store/pingu    cifs         credentials=/root/smbpass.txt                       0 0

# mounting NFS
pingu:/store    /store          nfs          rw                                                  0 0
```

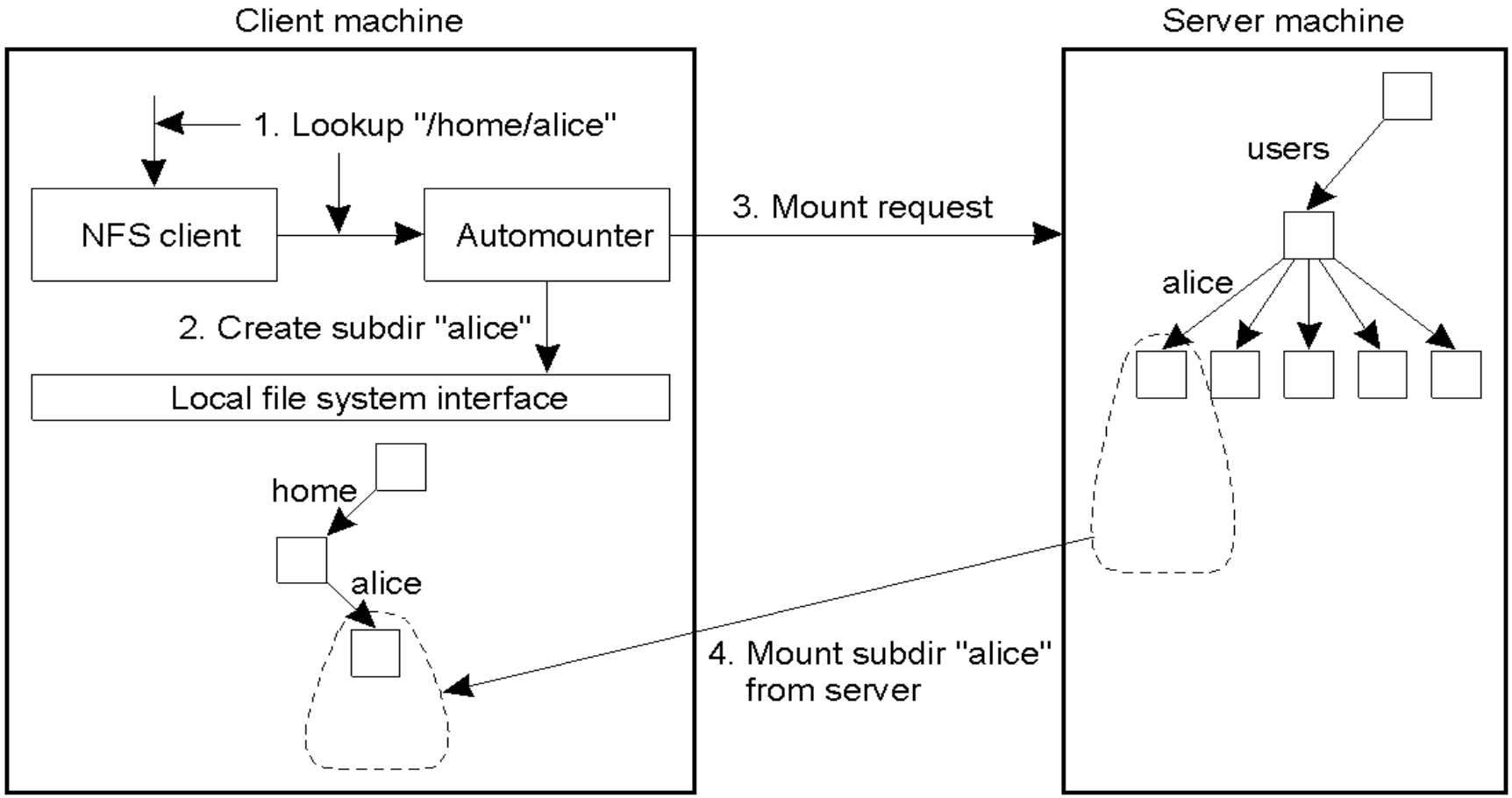
Mounting types

- Explicit mounting: clients make explicit mounting system calls whenever one is desired.
- Boot mounting: a set of file servers is prescribed and all mountings are performed at the client's boot time.
- Auto-mounting: mounting of the servers is implicitly done on demand when a file is first opened by a client.

Location Transparency



A Simple Automounter for NFS



Continued…

(2) Caching

- Improves file system performance by exploiting the locality of reference
- When client references a remote file, the file is cached in the main memory of the server (server cache) and at the client (client cache)
- When multiple clients modify shared (cached) data, cache consistency becomes a problem
- It is very difficult to implement a solution that guarantees consistency

(3) Hints

- Treat the cached data as hints, i.e. cached data may not be completely accurate
- Can be used by applications that can discover that the cached data is invalid and can recover

  - Example:
    - After the name of a file is mapped to an address, that address is stored as a hint in the cache.
    - What will happen if the location of the file is changed?

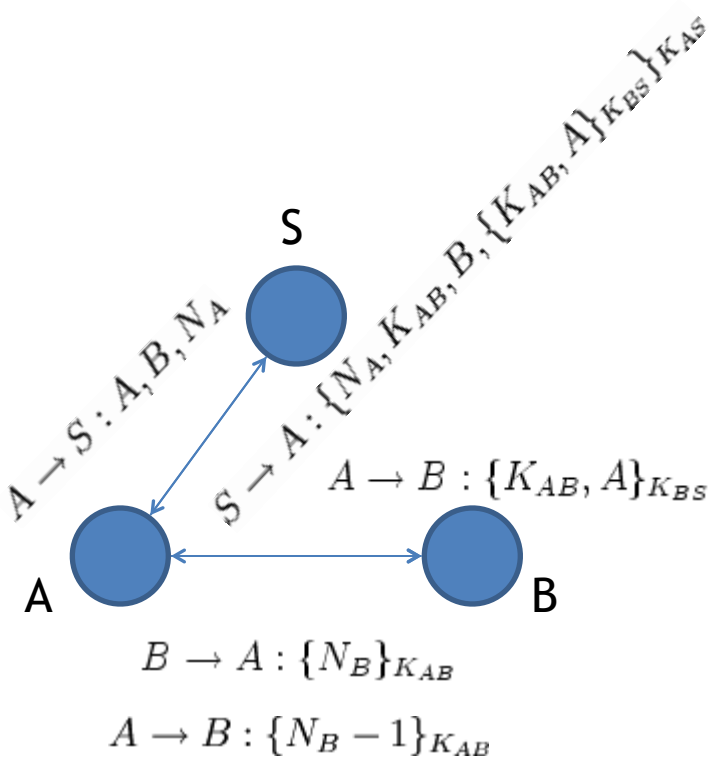
Continued…

(4) Bulk data transfer

- Observations:
  - Overhead introduced by protocols does not depend on the amount of data transferred in one transaction
  - Most files are accessed in their entirety
- Common practice: when client requests one block of data, multiple consecutive blocks are transferred

(5) Encryption

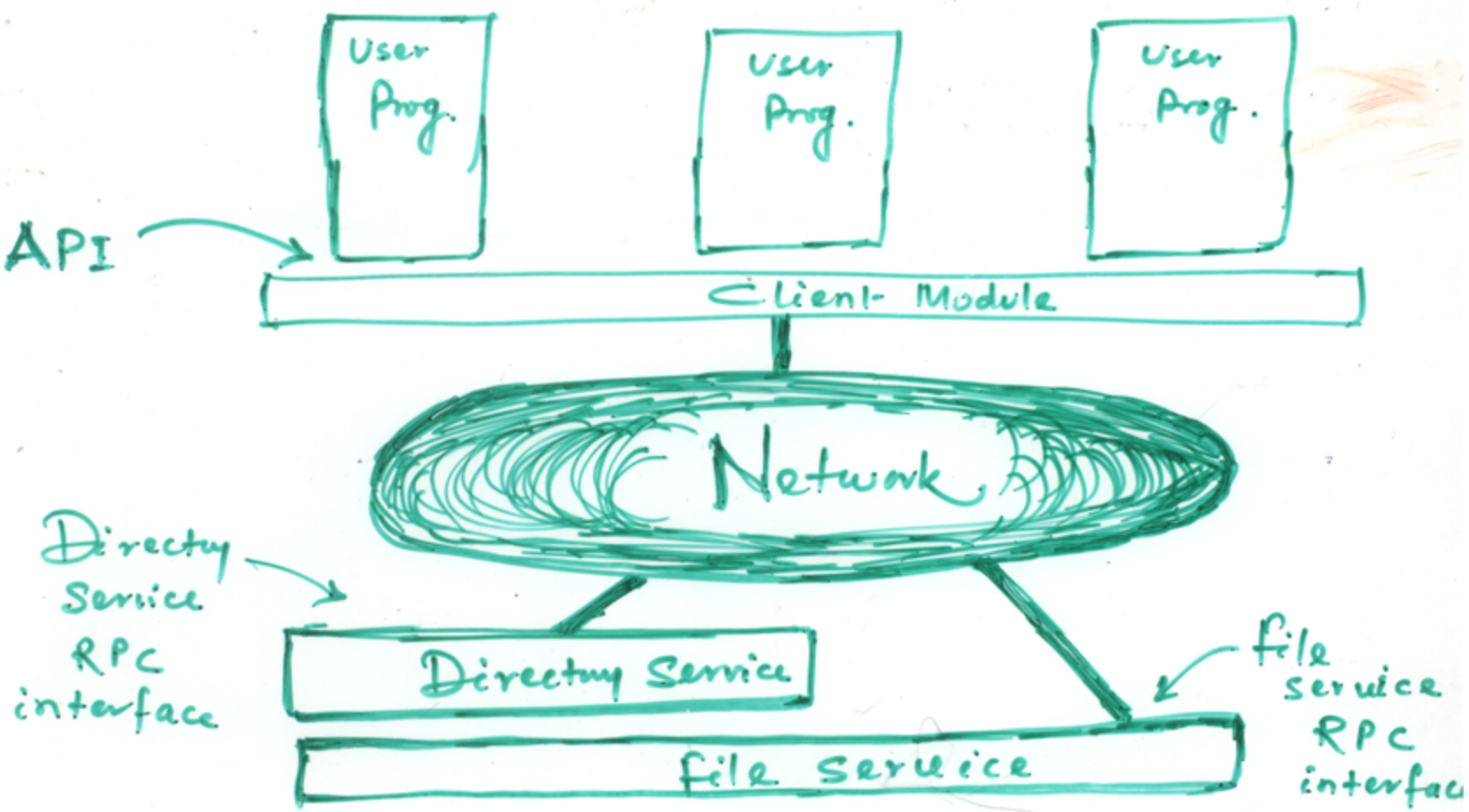
- Encryption is needed to provide security in distributed systems
- Entities that need to communicate send request to authentication server
- Authentication server provides key for conversation



Needham-schroeder

Design Issues

1. Components of a DFS (File service, directory service)



Distributed systems

- Job of file system is to store programs and data
- Make them available when needed
- Two important components

1. File service and
2. file server

File service

- Specification of what the file system offers to its clients
- To the clients: defines what service they can count on
- Nothing to say how it is implemented
- Specifies the interface to the clients

File server

- Process that runs on some machine and helps implement the file service
- Clients should not know

1. how many file servers
2. Location of file servers
3. And functions of each file server

- When clients calls a specific file service, required work is done somehow and results are returned
- Clients should not know that file service is distributed
- It should look same as single-processor file system

Distributed file system design

- File system has two distinct components
- File service and
- Directory service
- File service: Operations on individual files including reading, writing, and appending.
- Directory service: creating and managing directories. adding and deleting files from directories.

File service interface

- What is a file?
- Un-interpreted sequence of bytes.
- Meaning and structure of the information depends on application programs
- OS not interested
- E.g. UNIX and WINDOWS
- File can be structured as sequence of records
- Each record number or some other field specifies different record.
- B-tree or hash table to locate records quickly
- E.g. mainframes

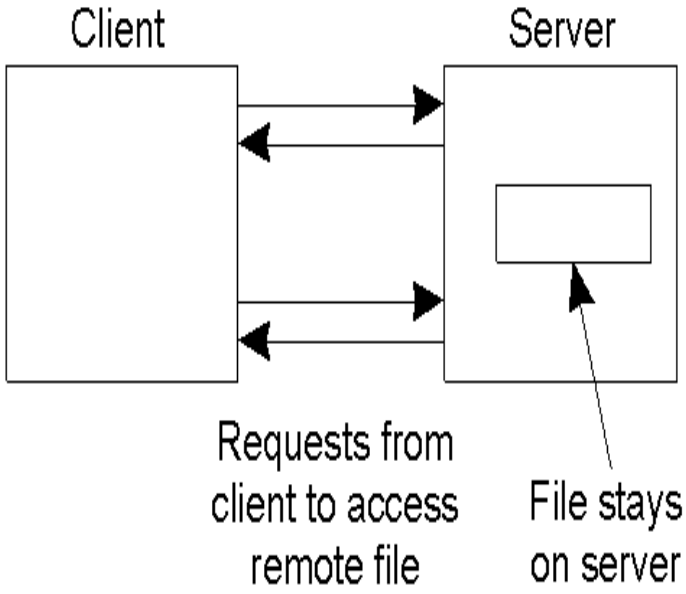
- Attributes of files
- Information not part of files but about the files
- Owner, size, creation date, access permissions
- User-defined attributes possible in addition to standard ones existing.

Immutable files

- Files can not be modified after creation
- Distributed systems have two operations possible
- CREATE
- READ
- Makes job of file system easier to support file caching and replication
- Eliminates all problems associated with having to update all copies of a file whenever it changes.

Design Issues continued…

- File Service

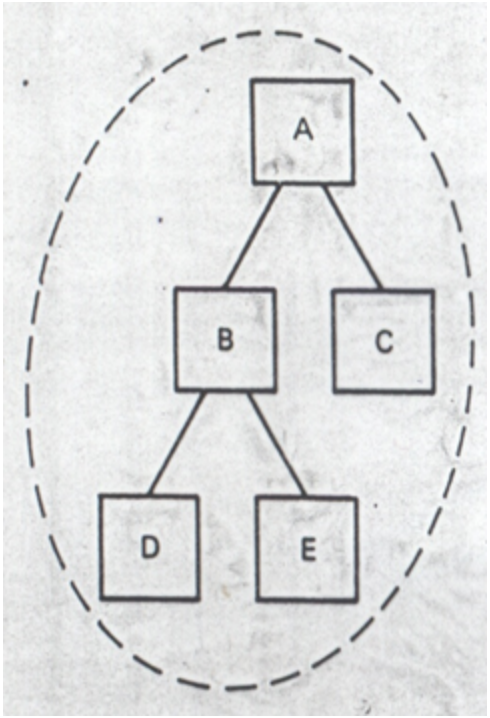


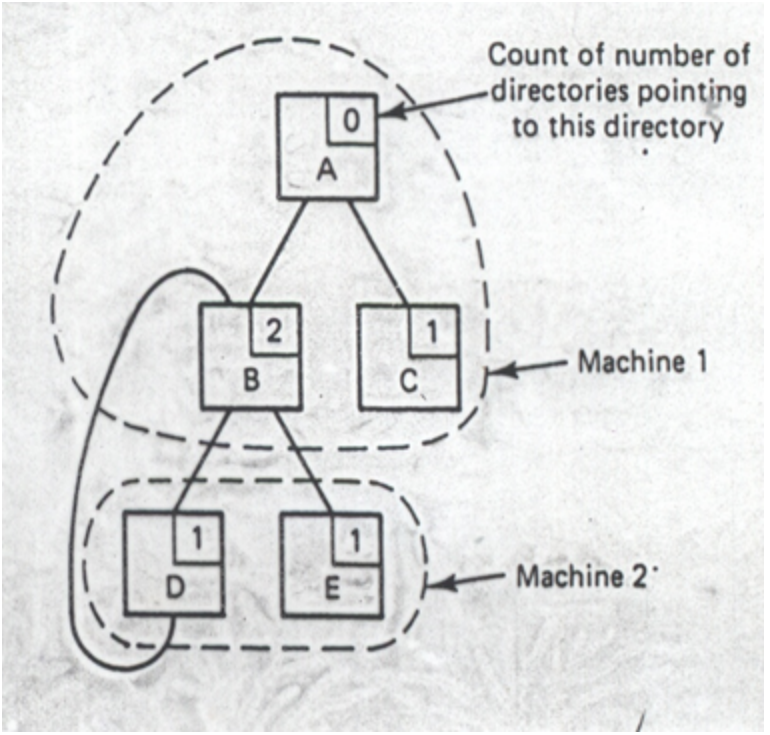
Remote Access

- File service can either of the form of
- upload/download or
- remote access model
- Upload/download:-
- Read file and write file
- Store file in a memory or local storage
- Fetch the files when needed and use them locally
- No complicated file service interface to be mastered
- File transfer is highly efficient
- Enough storage must be available
- Wasteful when fraction of file is needed

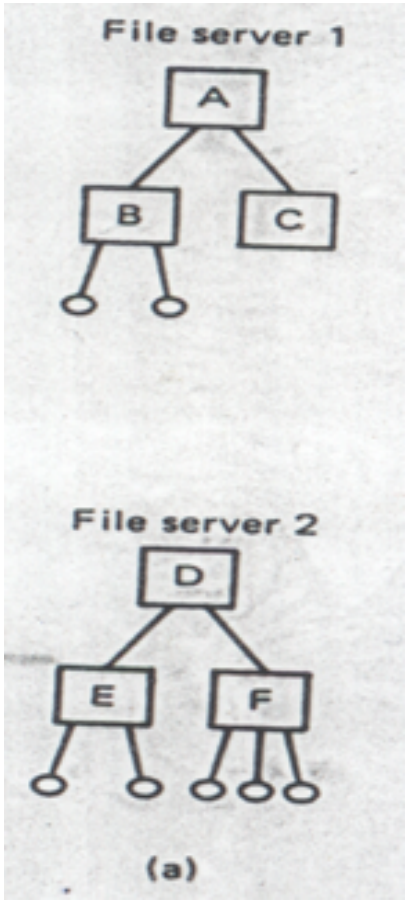
Continued…

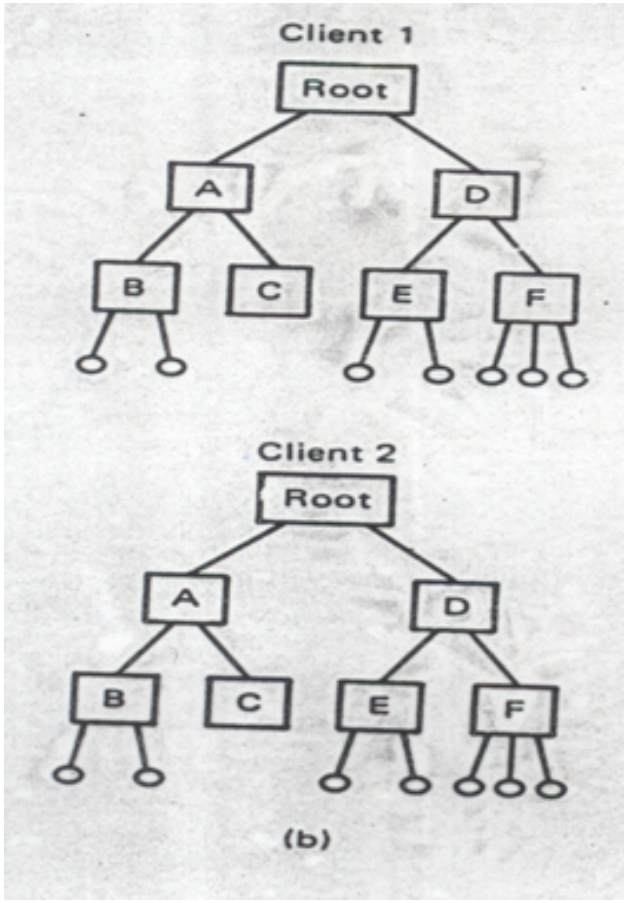
- Directory Service

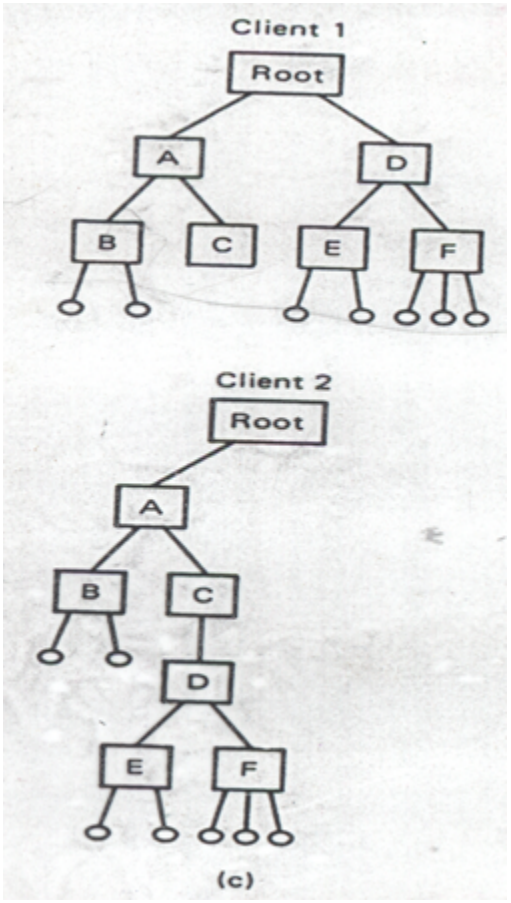




Continued…


File server 1
A
B
C
File server 2
D
E
F
(a)


Client 1
Root
A
D
B
C
E
F
Client 2
Root
A
D
B
C
E
F
(b)


Client 1
Root
A
D
B
C
E
F
Client 2
Root
A
B
C
D
E
F
(c)

Directory server interface

- Operations for creating and deleting directories
- Naming and renaming files
- Moving from one directory to another
- Identify files either by having an extension or
- Through specific attribute dedicated for it.
- Hierarchical file system
- Unreachable or orphan directories
- Should all the machines/processes have same view of directory hierarchy.
- Is there a global root directory (one entry for each server)?

Design Issues Continued…

2. Naming and name resolution
   - Machine + path
   - Mounting (Sun NFS)
   - Single global directory (Sprite DFS)
     - (Mounting the same way: Difficult to enforce this restriction. Can work only among cooperating systems (or system administrators!)
   - Contexts (geographical, organizational, etc.)

~john may be mapped to /home/students/john in client 1 and /usr/students/john in client 2

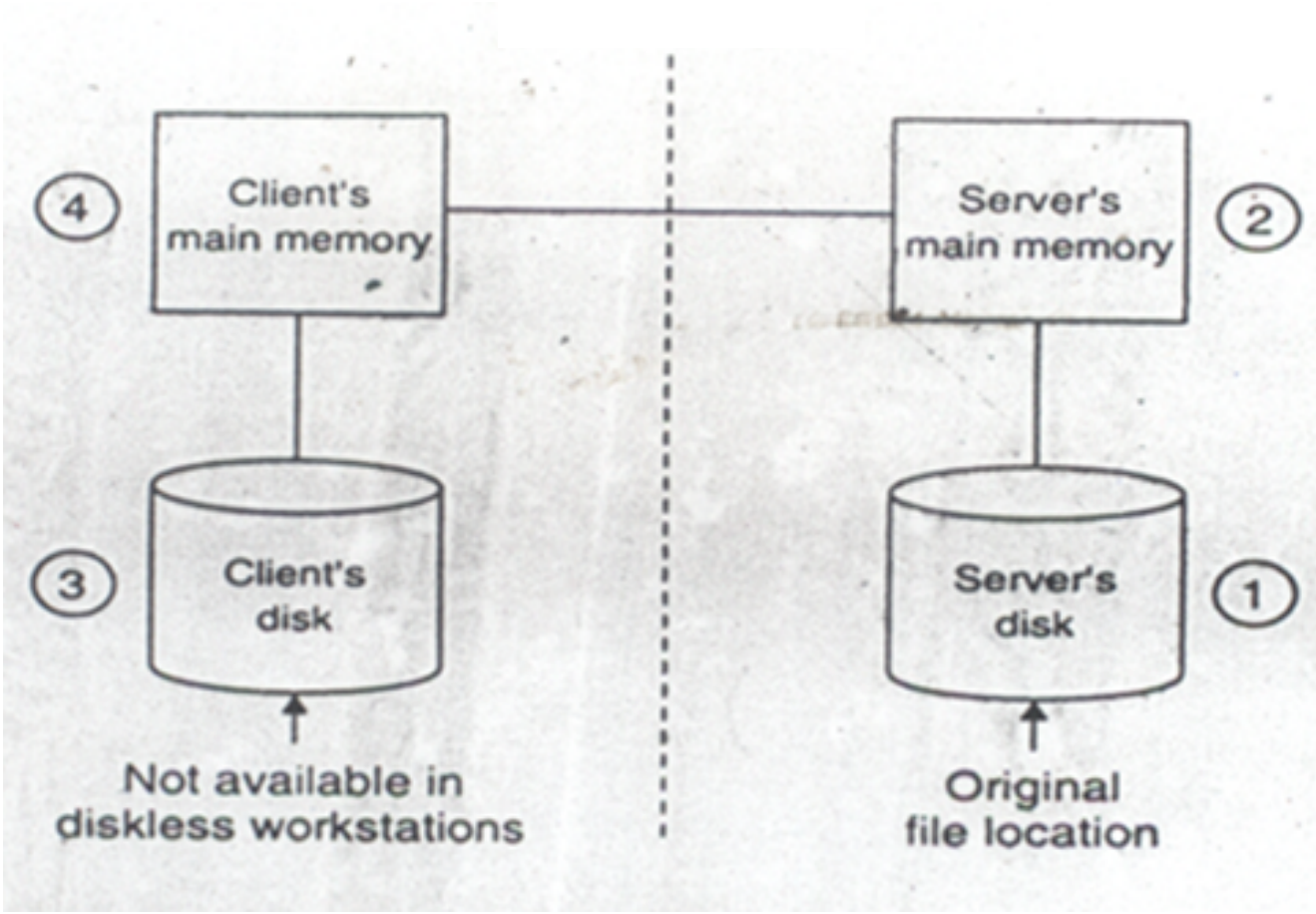
- Name servers

Design Issue Continued…

3. Caches on Disk or Main memory

VM mgt becomes more complex

VM mgt becomes easy



Design Issues Continued…

4. Writing Policies

- When should a modified cache content be transferred to the server?
- *Write-through* policy:
  - Immediate writing at server when cache content is modified.
  - Advantage: reliability.
  - Disadvantage: Several writes for each small change.
- *Delayed writing* policy:
  - Write at the server, after a delay.
  - Advantage: frequent changes do not increase network traffic.
  - Disadvantage: less reliable, susceptible to client crashes.
- *Write on close*.

Design Issues Continued…

5. Cache Consistency

- Server initiated:
  - Server informs cache managers when data in client caches is stale. Client cache managers invalidate stale data or retrieve new data.
- Client initiated:
  - Cache managers at the clients validate data with server before returning it to clients.
- *Concurrent-write sharing* policy:
  - ✓ Multiple clients open the file, at least one client is writing.
  - ✓ File server asks other clients to purge/remove the cached data for the file, to maintain consistency.

Continued…

- *Sequential-write sharing* policy: a client opens a file that was recently closed after writing.
  - This client may have outdated cache blocks of the file (since the other client might have modified the file contents).
    - Use time stamps for both cache and files. Compare the time stamps to know the freshness of blocks.
  - The other client (which was writing previously) may still have modified data in its cache that has not yet been updated on server. (e.g.,) due to delayed writing.
    - Server can force the previous client to flush its cache whenever a new client opens the file.

Design Issues Continued…

6. DFS Replication

- Issue: what is the level of availability of files in a distributed file system?

- Replication issues:
  - How to keep replicas consistent
  - Unit of replication
    - File (requiring extra name resolutions to locate the replicas)
    - Group of files

    a) Volume: group of all files of a user or group of all files in a server
      - » Advantage: ease of implementation
      - » Disadvantage: wasteful, user may need only a subset of a volume

    b) Primary pack vs. pack
      - » Primary pack : all files of a user
      - » Pack: subset of primary pack. Can receive a different degree of replication for each pack

Replica management

- Two-phase commit protocols can be used to update all replicas.
- Other schemes:
  - Current Synchronization Site (CSS)
  - Weighted votes (Voting schemes):
    - A certain number of votes *r* or *w* is to be obtained before reading or writing. {r+w > v, and w > (v/2)}

DFS Design Issues Continued…

7. Scalability
   - Client-Server design
   - Client caching
   - Server initiated cache invalidation
     - Do not bother about read-only files
     - Clients serving as servers for few clients
   - Structure of server

DFS Design Issues Continued…

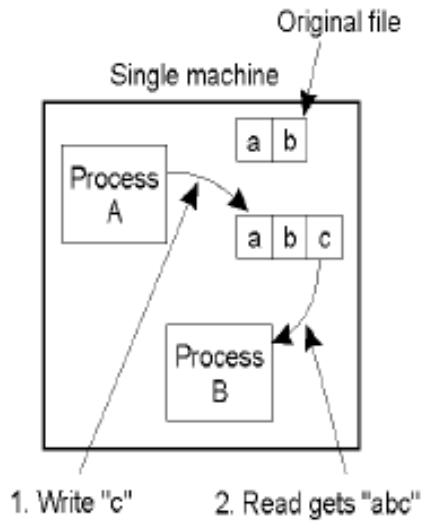
8. File Sharing Semantics

a) On a single processor, when a *read* follows a *write*, the value returned by the *read* is what?

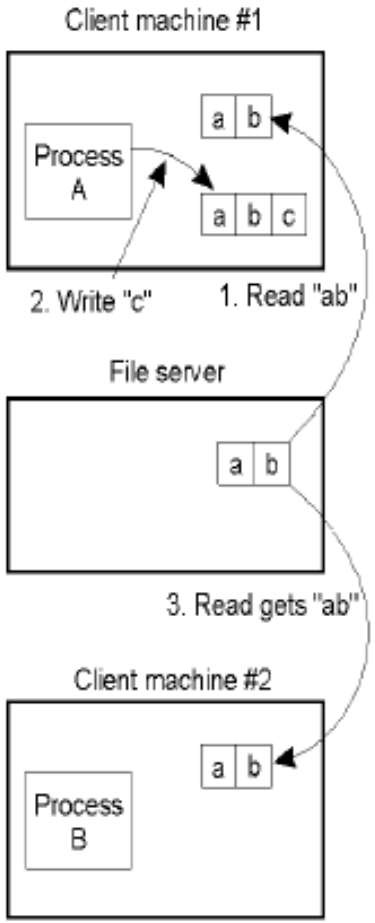
b) In a distributed system with caching, obsolete values may be returned.

Why is it needed?

To solve coherency and concurrency control problems.



(a)



(b)

Semantics of file sharing

- UNIX semantics: every operation on a file is instantly visible on all processes
- Single server, performance poor
- Clients store local copies
- Obsolete files in one of the cached copy.
- Propagate all changes to cached files back to server immediately
- inefficient

- Session semantics: no change are visible to other processes until the file is closed
- If two files changing simultaneously
- Store one copy, ignore other
- Pointer to one file not possible in session semantics

- Immutable files: no updates possible, simplifies sharing and replication.
- CREATE and READ possible
- Replace x with new file
- Two process try to replace x at same time?

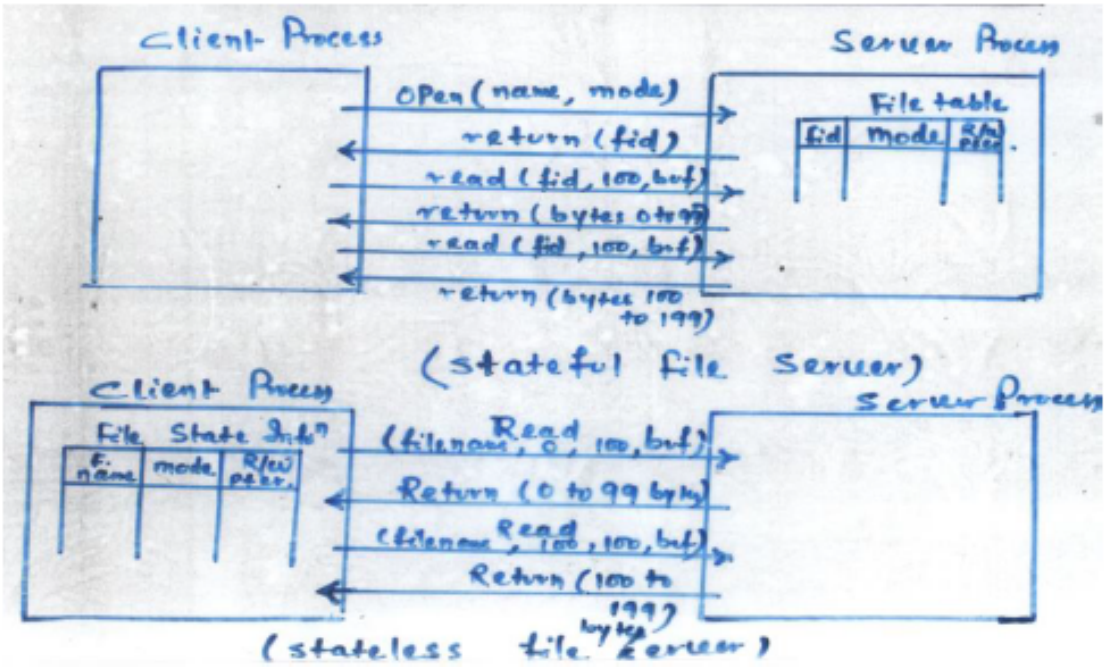
- Transactions : all changes have the all-or-nothing property
- START TRANSACTION
- END TRANSACTION
- Final result same as if all simultaneous transactions were all run in some sequential order
- E.g. Rs. 100 added to 50 by two processes at same time

Semantics of File Sharing (Cont.)

Unix Semantics	Every operation on a file is instantly visible to all processes.
Transaction Semantics (Consistency)	All changes have the all-or-nothing property. Update the server at the end of a transaction.
Immutable Files	No updates are possible; simplify sharing and replication.
Session Semantics	No changes are visible to other processes until the file is closed. Update the server at the end of a session.

DFS Design Issues Continued…

9. Stateful and Stateless File Servers



- State information – may be kept in servers in a stateful file server
  - Opened files and their clients
  - File descriptors and file handles
  - Current file position pointers
  - Mounting information
  - Lock status
  - Session keys
  - Cache or buffer

Case Studies

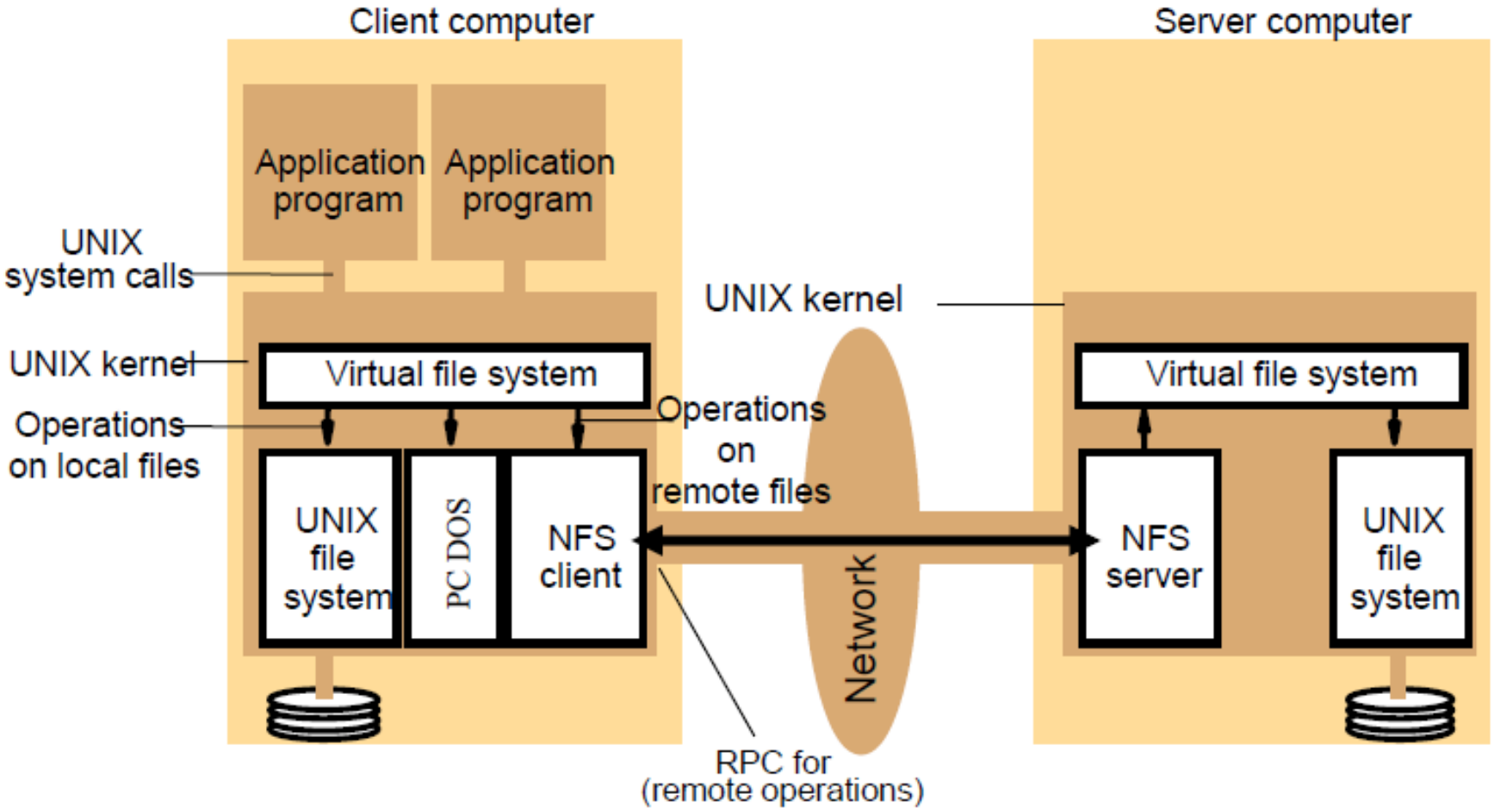
1. Sun NFS
2. Sprite File system
3. Hadoop distributed file system
4. CODA
5. X-kernel logical file system

Case study: Network File System(NFS)

- developed by Sun MicroSystems
- early 1980s

- dominant for Unix environments
- Currently de facto standard for LANs
- Versions
  - NFS-3
  - NFS-4, IETF standard

NFS architecture



NFS RPC Calls

- NFS / RPC using XDR / TCP/IP

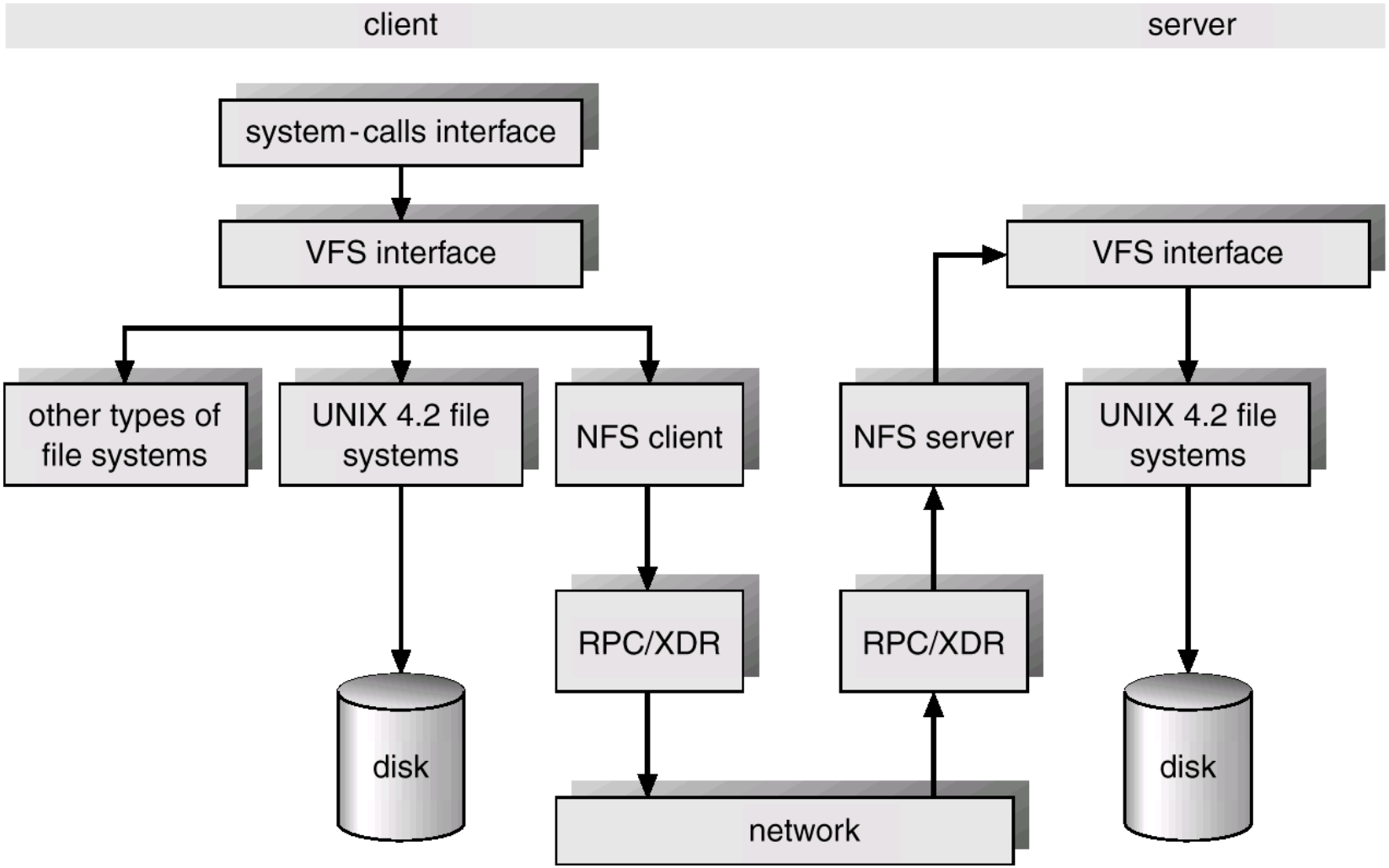
Procedure	Input arguments	Results
lookup	dirfh, name	status, fhandle, fattr
read	fhandle, offset, count	status, fattr, data
create	dirfh, name, fattr	status, fhandle, fattr
write	fhandle, offset, count, data	status, fattr

- fhandle: 32-byte opaque data (64-byte in v3)
  - What information is available in file handle?

NFS Operations

- V2:
  - NULL, GETATTR, SETATTR
  - LOOKUP, READLINK, READ
  - CREATE, WRITE, REMOVE, RENAME
  - LINK, SYMLINK
  - READIR, MKDIR, RMDIR
  - STATFS
- V3: add
  - *READDIRPLUS*, *COMMIT*
  - FSSTAT, FSINFO, PATHCONF

NFS Architecture



Source: AoS, Silberschatz, Galvin, and Gagne ©1999

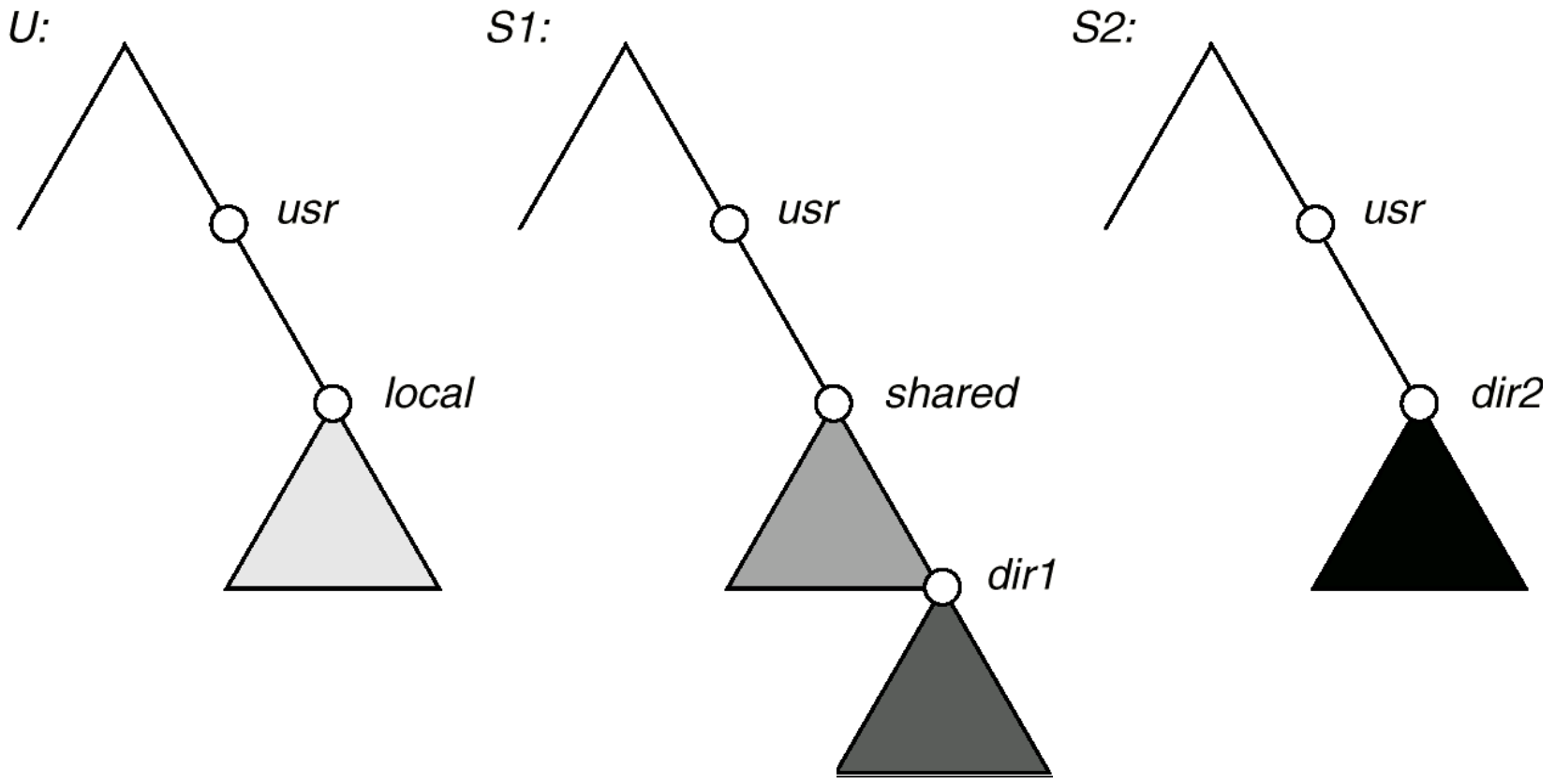
NFS components

- File system interface called VFS
- Vnode for every object
- Network wide unique
- Mount table to distinguish between local and remote file systems
- Any node in FS can be a mount point

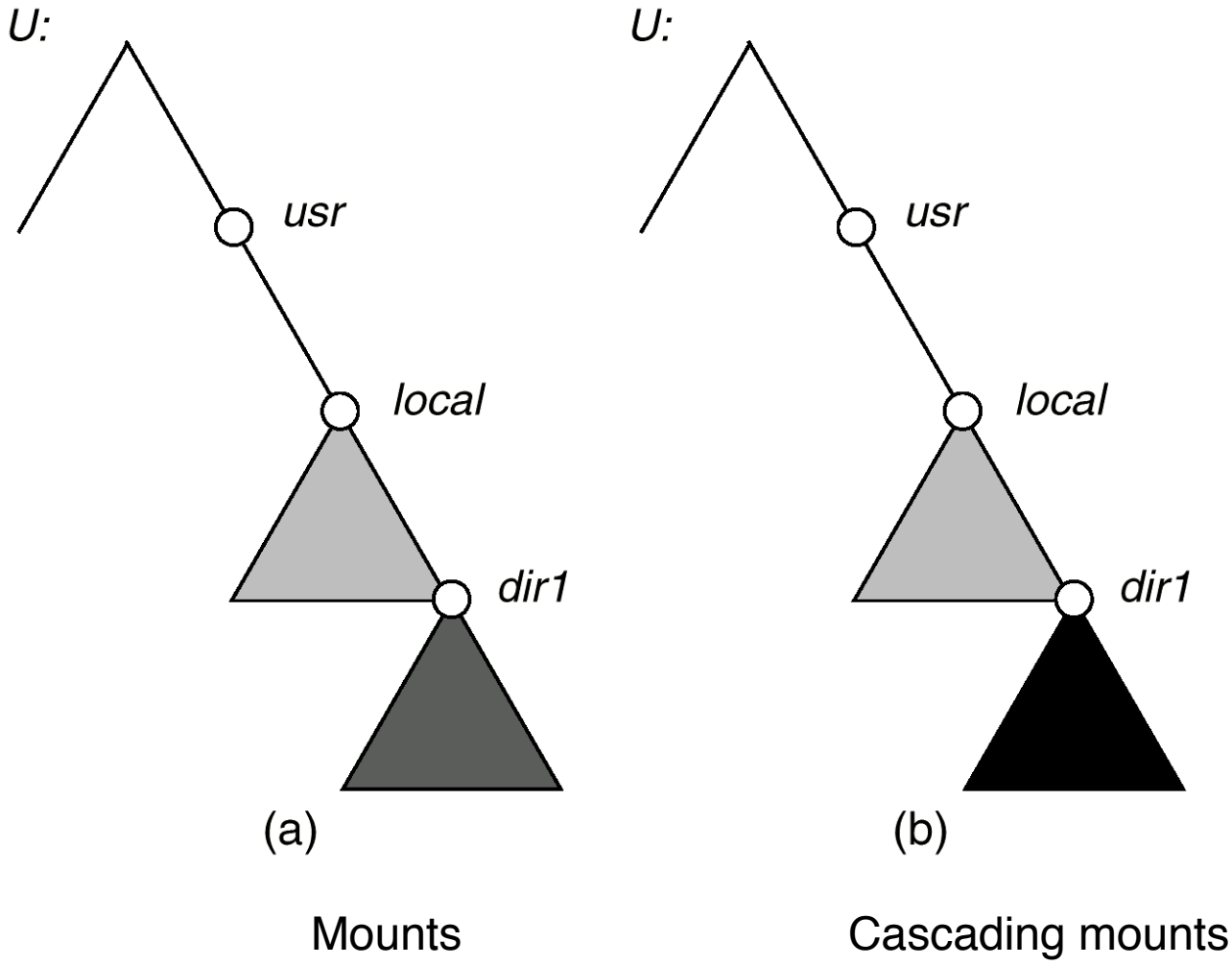
NFS: Naming & resolution

- Each client can configure its file system independent of others.
- Name resolution example:
  - Look up for *a/b/c. a* corresponds to vnode1 (let).
  - Look up on vnode1/b returns vnode2 that might say the object is on server X.
  - Look up on vnode2/c is sent to X. X returns a *file handle*.
  - File handle: a file-system identifier, and an inode number to identify the mounted directory within the exported file system.
- What type of process is this name resolution?
- Name space information is not maintained at each server as the servers in NFS are *stateless*.

NFS Mount protocol: Three Independent File Systems



Mounting in NFS



Mounts

Cascading mounts

NFS: Caching

- NFS Client Cache:
  - File blocks: cached on demand.
    - Employs read ahead. Large block sizes (8 Kbytes) for data transfer to improve the sequential read performance.
    - Entire files cached, if they are small. Timestamps of files are also cached.
    - Cached blocks are valid for certain period after which validation is needed from server. Validation done by comparing time stamps of file at server.
    - Delayed writing policy is used. Modified files are flushed after closing to handle seq write sharing.
  - File name to vnode translations: directory name lookup cache holds the vnodes for remote directory names.
    - Cache updated when lookup fails (cache acts as *hints*).
  - Attributes of files & directories: File attributes are discarded after 3 seconds and directory attributes after 30 seconds.

NFS: Stateless Server

- NFS servers are stateless to help crash recovery.
- Stateless: no record of past requests
- Client requests contain all the needed information. No response, client simply re-sends the request. Idempotent requests.
- After a crash, a stateless server simply restarts. No need to:
  - Restore previous transaction records.
  - Update clients or negotiate with clients on file status.
- Disadvantages:
  - Client message sizes are larger.
  - Server cache management difficult since server has no idea on which files have been opened/closed.

Caching with Server Control

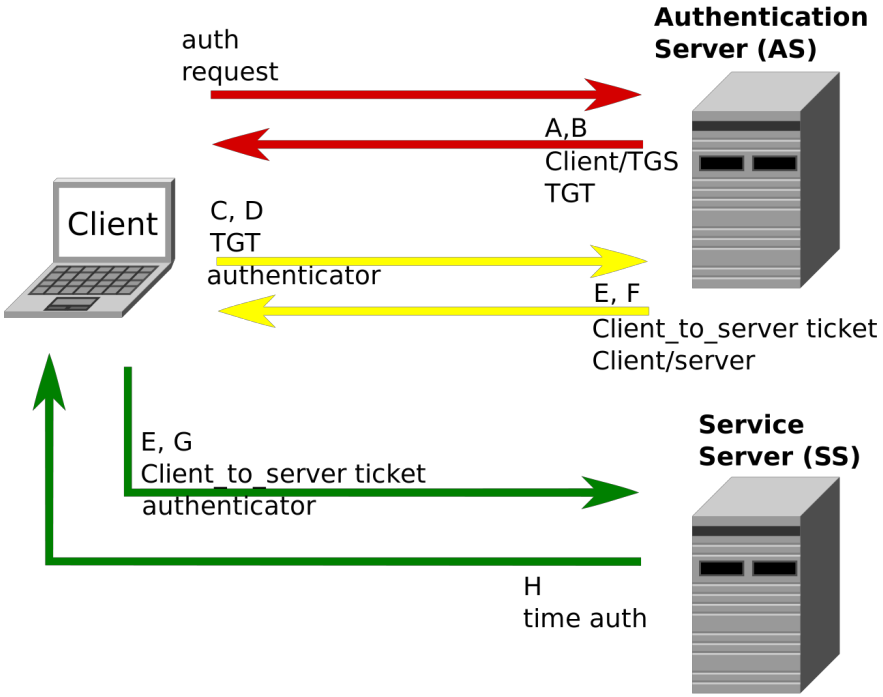
- In caching with server control
  - All clients on a single machine may read and write the same cached data if they have access rights
  - data remaining in the cache after a file closes doesn't need to be removed, although changes must be sent to server.
- If a new client on the same machine opens a file after it has been closed, the client cache manager usually must validate local cached data with the server
  - If the data is stale, replace it.

Caching With Open Delegation

- Allows a client machine to handle some local `open` and `close` operations from other clients on the same machine.
  - Normally the server decides if a client can open a file
- Delegation can improve performance by limiting contact with the server
- The client machine gets a copy of the entire file, not just certain blocks.

NFS access control and authentication

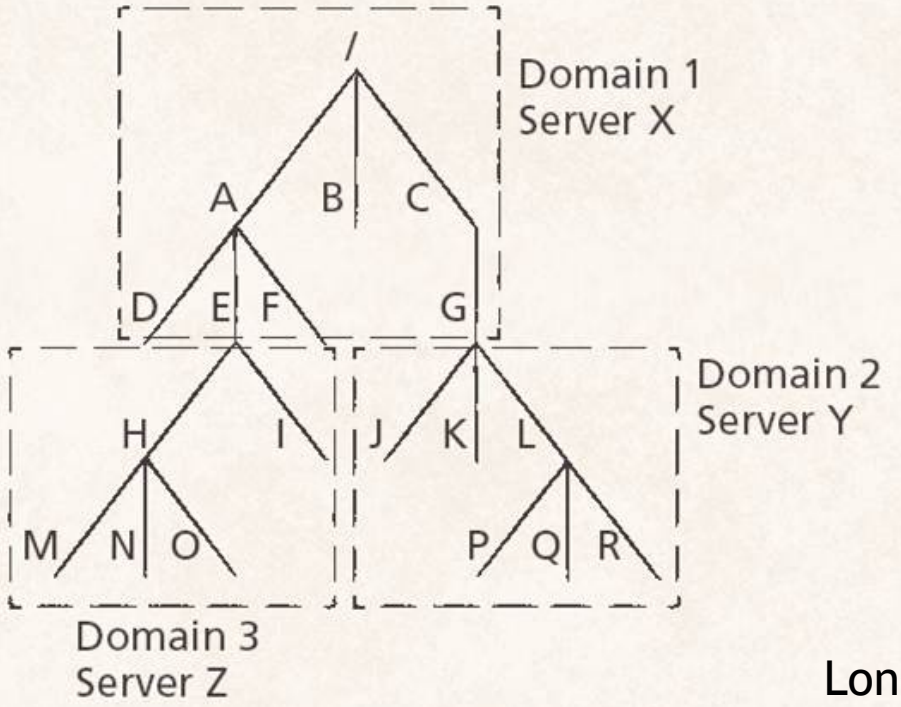
- Every client request is accompanied by the userID and groupID
  - They are inserted by the RPC system.
- Kerberos has been integrated with NFS to provide a stronger and more comprehensive security solution.



Sprite File System

- Part of Sprite distributed OS
  - Developed at UC Berkeley, mid 1990s
- Sprite file system characteristics
  - Emulates a UNIX file system
  - Every client has the exact same view of the hierarchy
  - Use large physical memory of workstations

Sprite File System domains



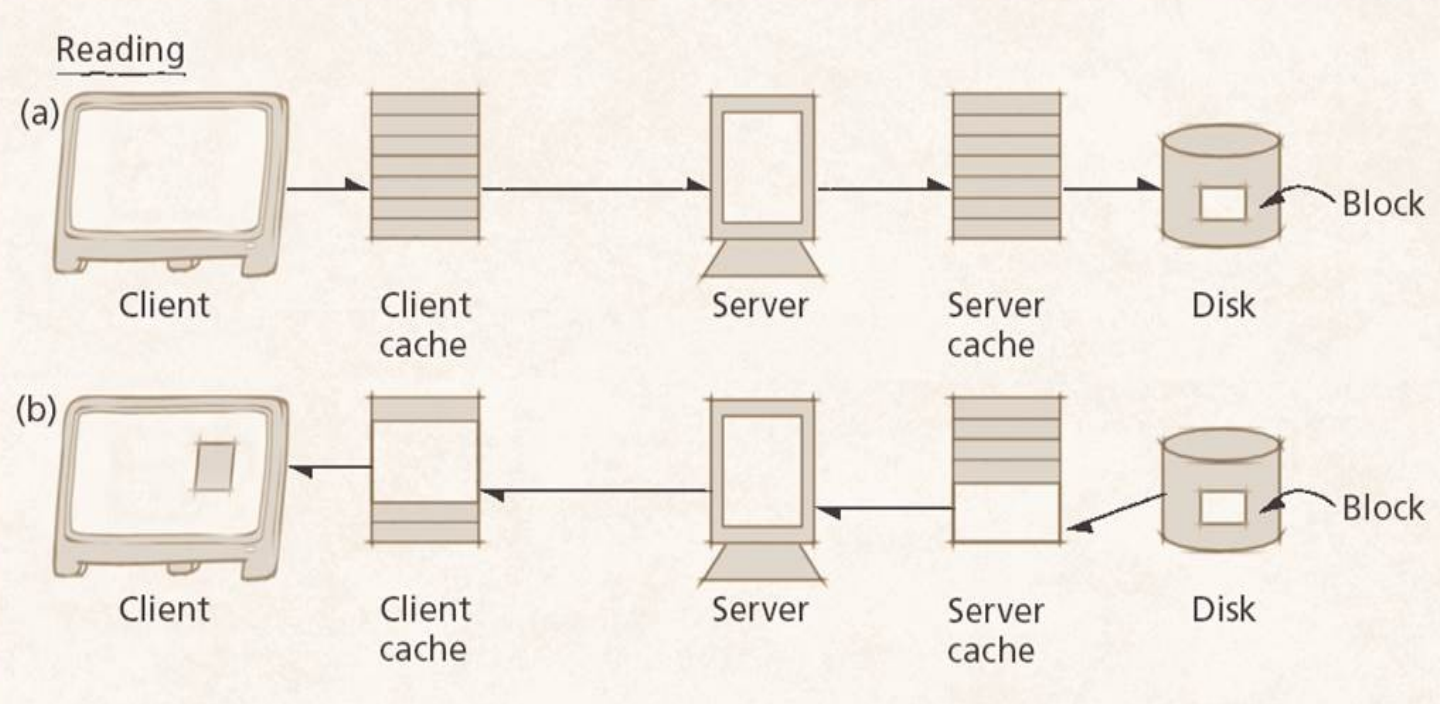
Prefix Table

Pathname	Server	Token
/	X	1
/A/E/	Z	3
/C/G/	Y	2

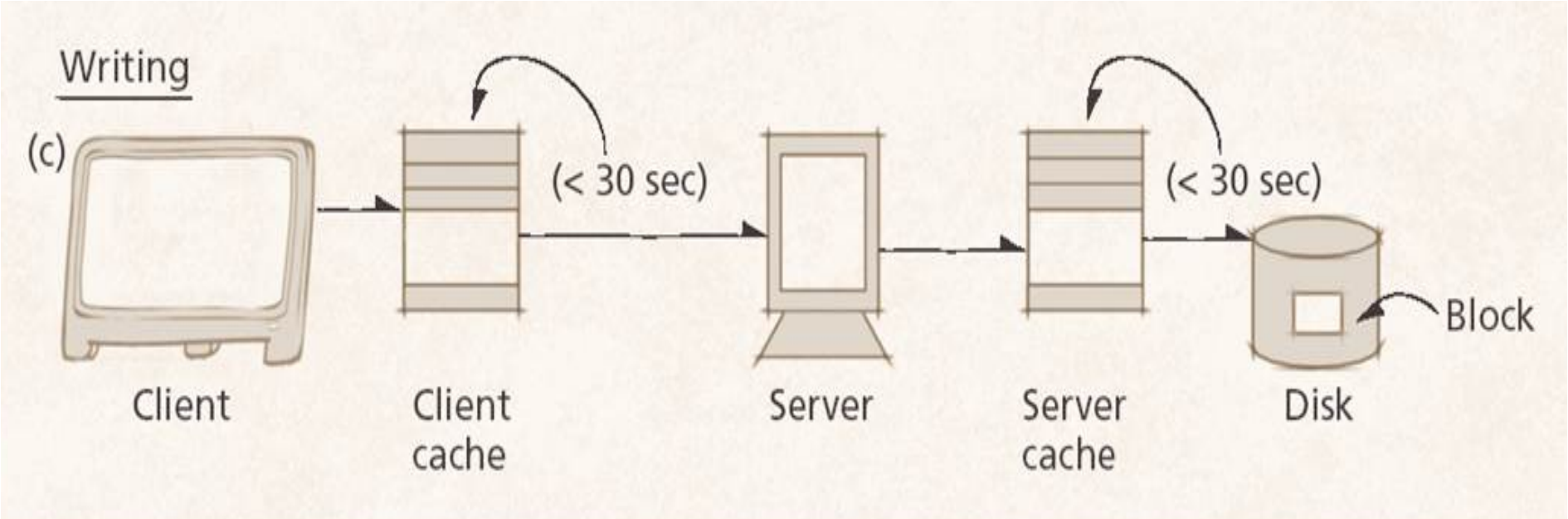
per client

hints
Longest prefix match
Store the token and server address in process state

Sprite read scenario



Sprite write scenario



Delayed writing policy
Daemon process checks every 5 sec for unmodified block for last 30 seconds
Least recently used policy for cache replacement

Sprite caching protocol

Server initiated approach cache consistency

- sequential write sharing problem
  - upon opening file for write, client checks file version with server.
  - Server keeps track of the last writer. Flush data
- concurrent write sharing
  - if 2 clients open file for write, caching is turned disabled. Write back modified blocks. Informs cache to invalidate.

A client's cache dynamically adapts to the changing demands on the machine's virtual memory systems & the file system.

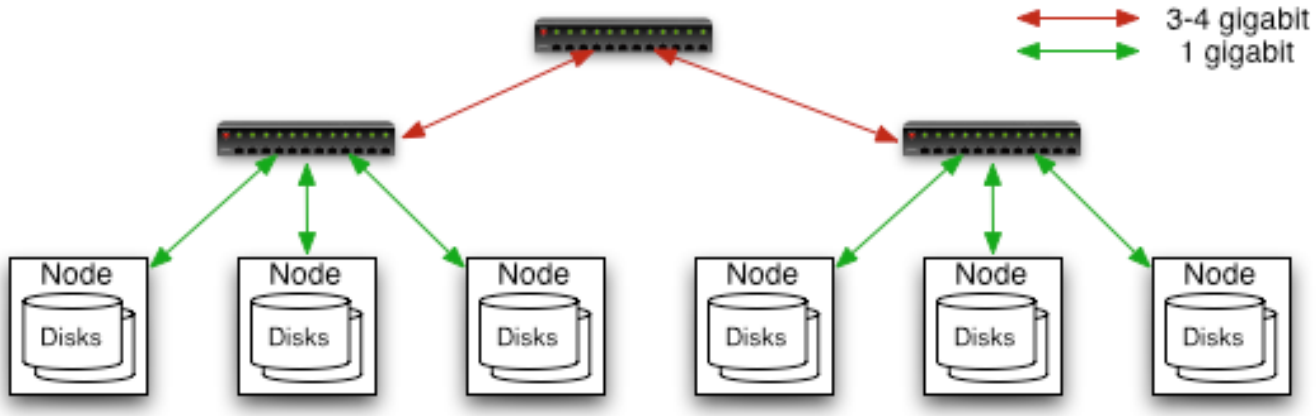
Virtual memory

- Double caching
- Executable code in cache given infinite age
- Cache has 24 MB of physical memory.
- Virtual addressing using #token and offset in block. Used in logical fachion.
- Backing files simplifies the process migration

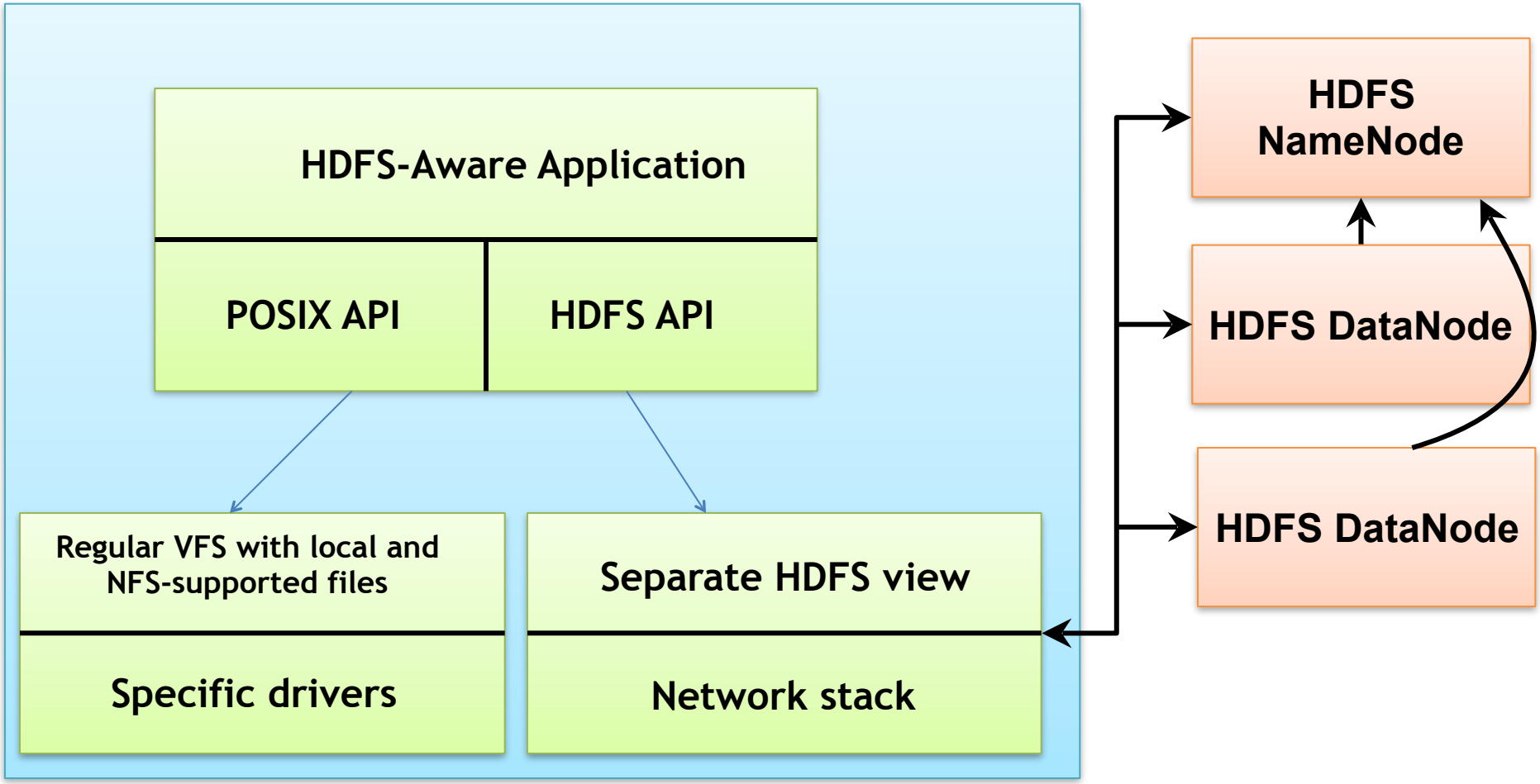
HDFS Design

- **Files stored as blocks**
  - Default 64MB
- **Reliability through replication**
  - replicated across 3+ DataNodes
- **Single NameNode coordinates access, metadata**
  - Centralized management
- **No data caching**
  - Little benefit due to large data sets, streaming reads

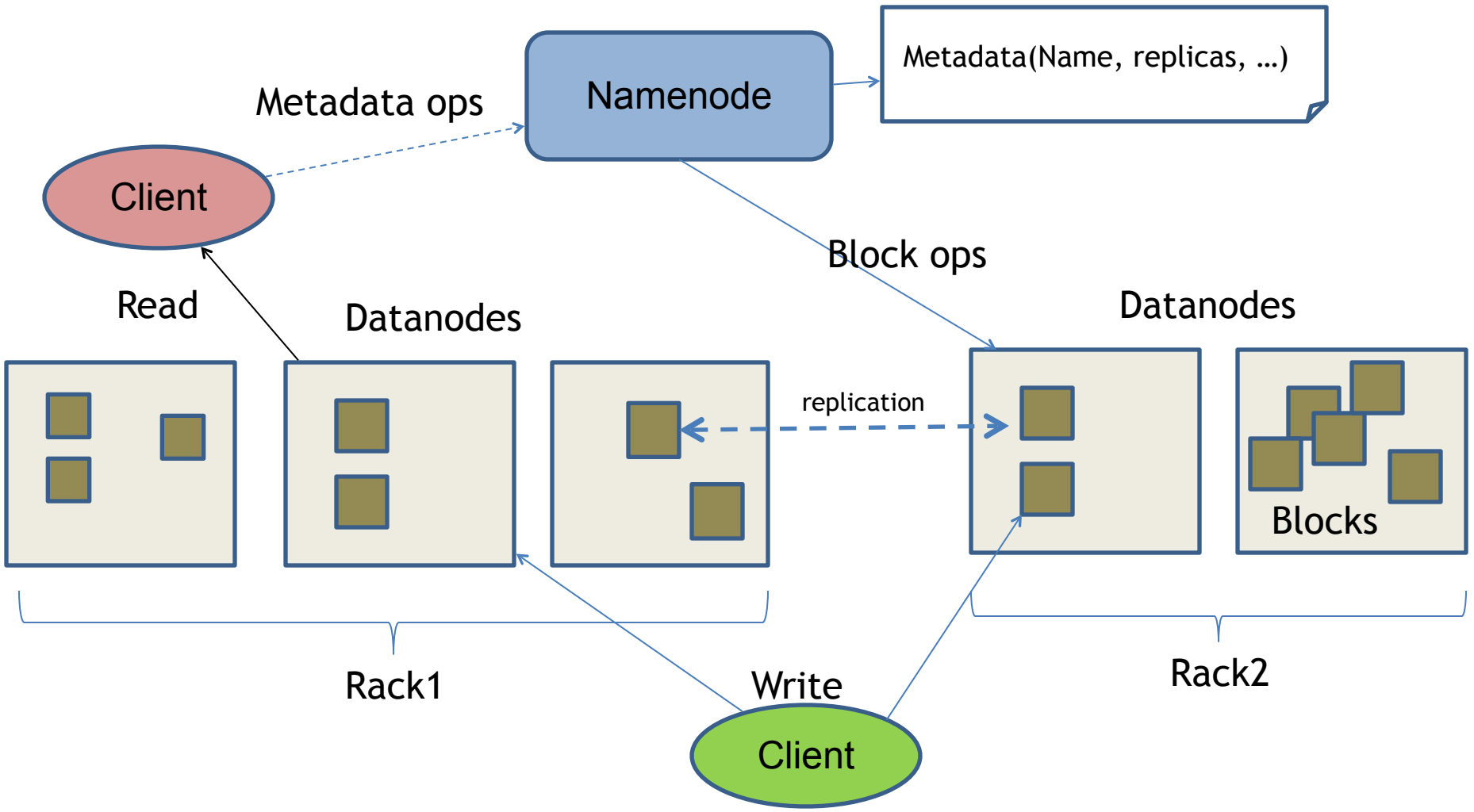
Commodity Hardware



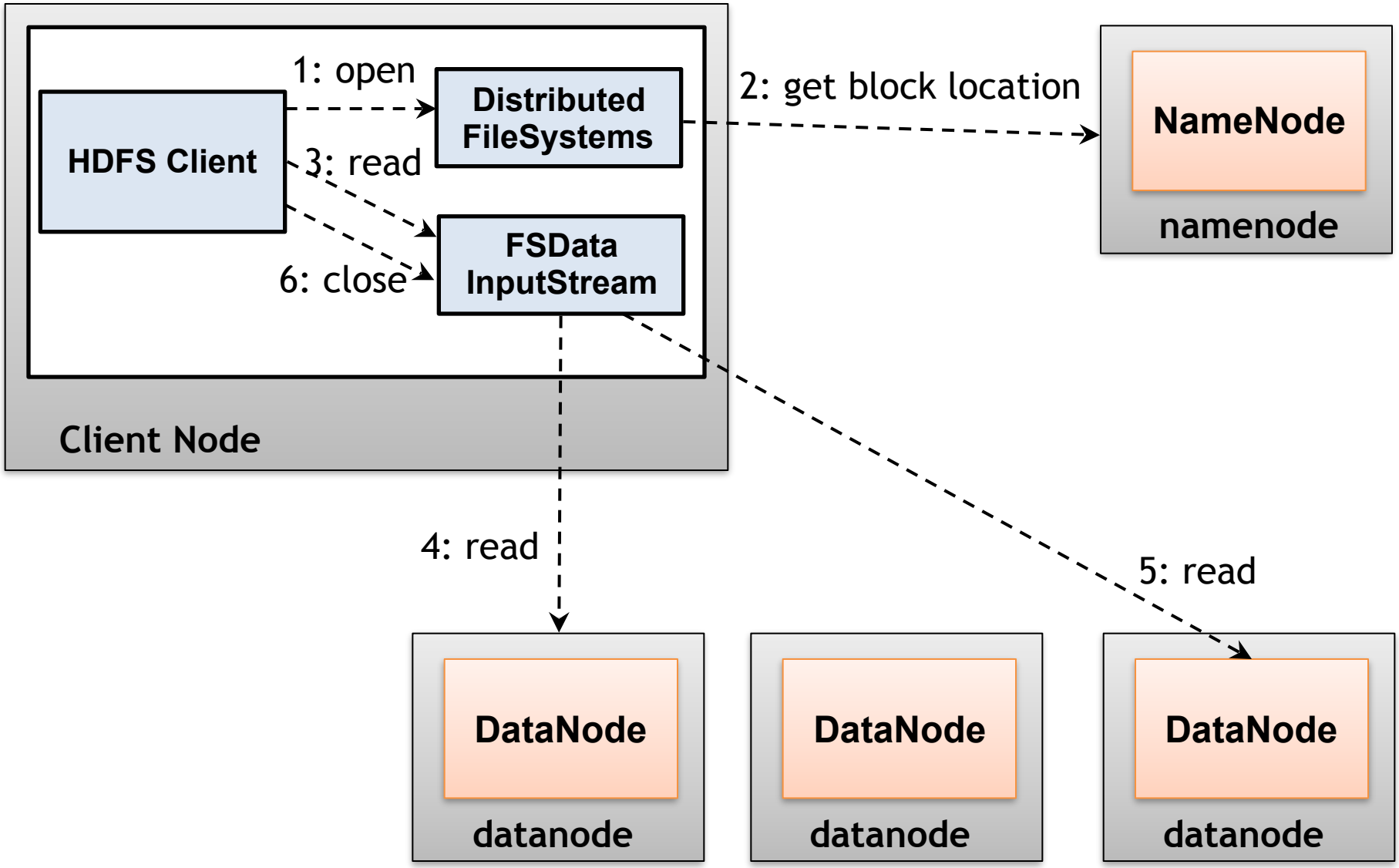
HDFS Architecture



HDFS Architecture

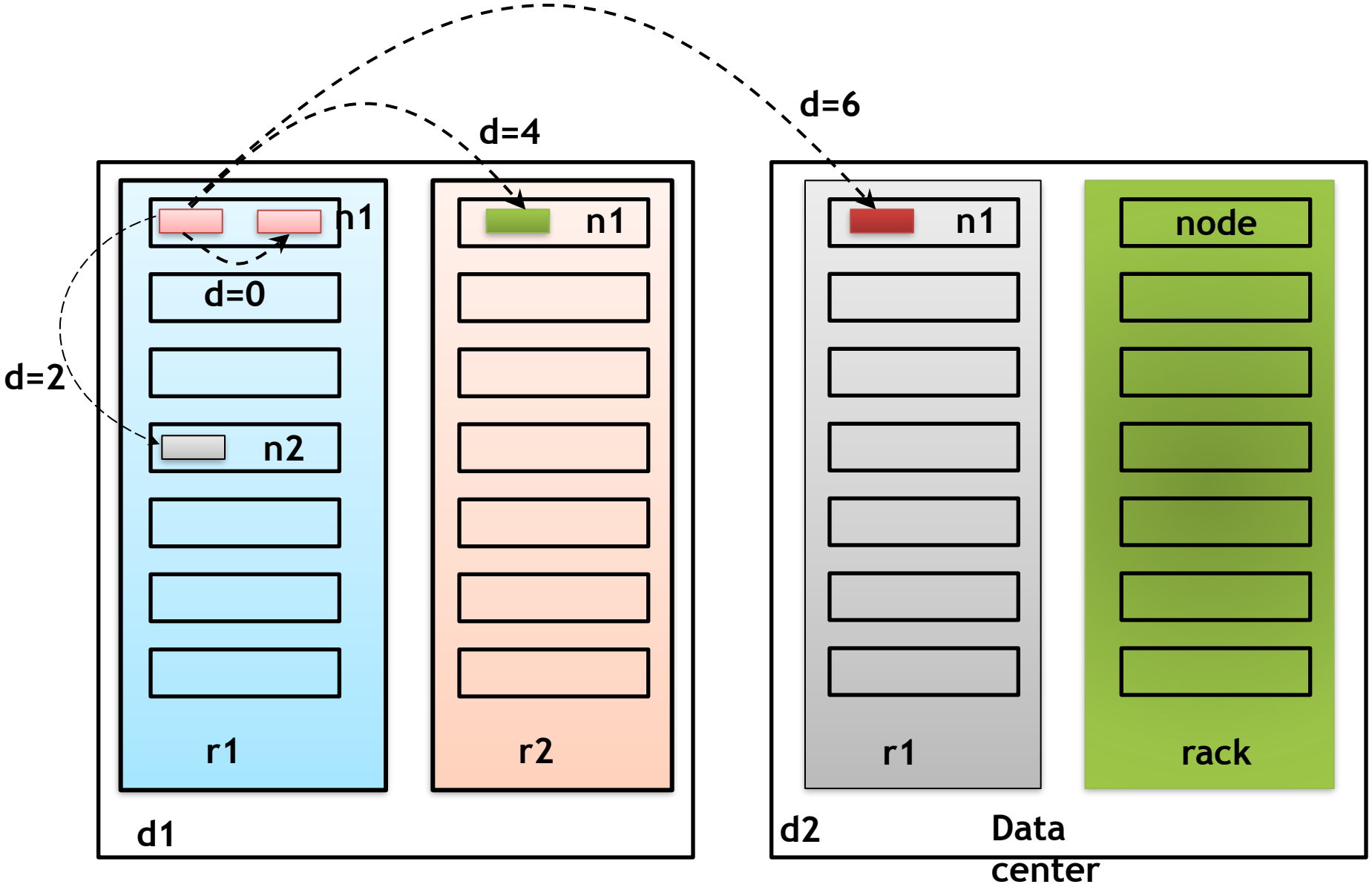


HDFS File Read

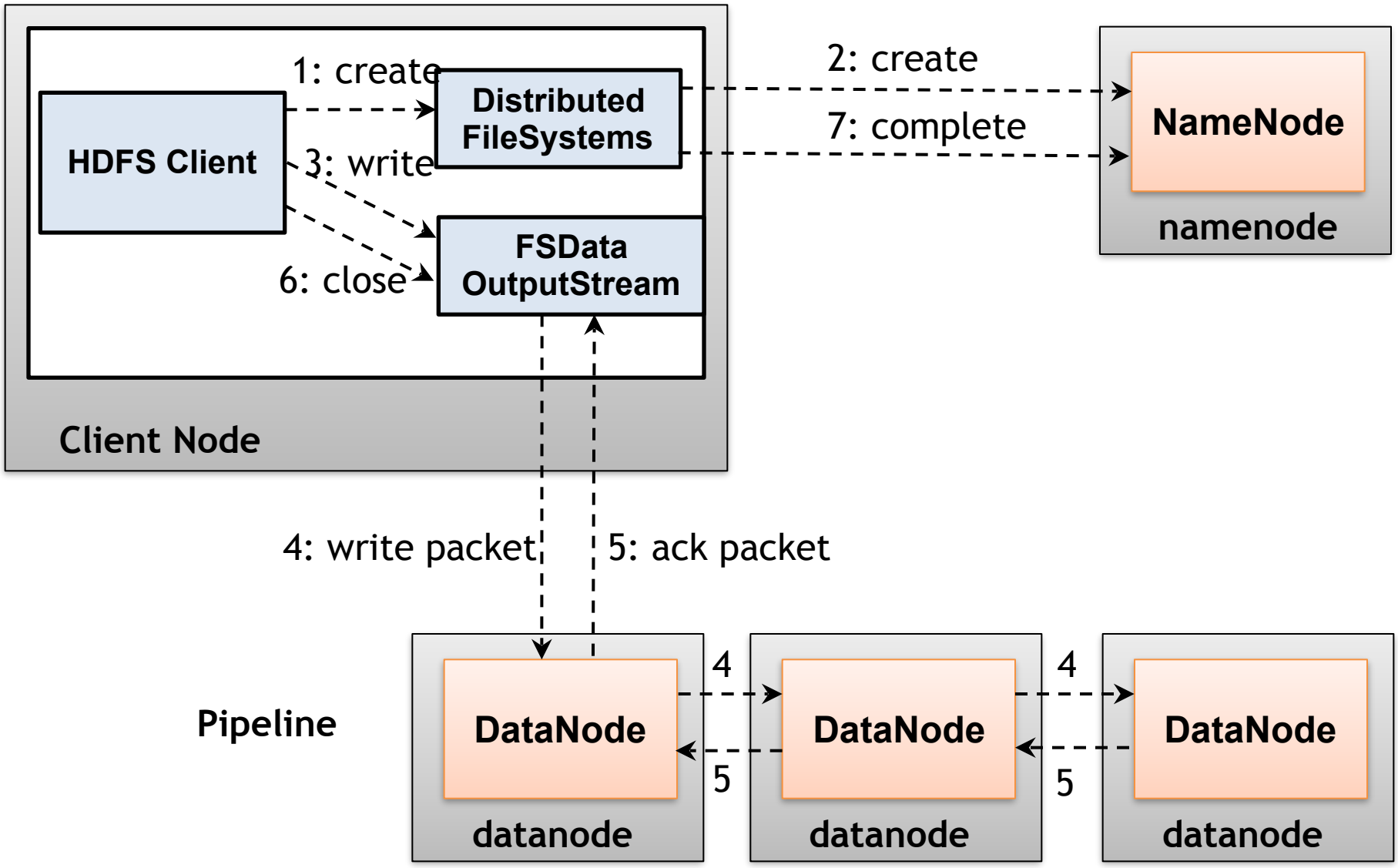


Hadoop Clusters

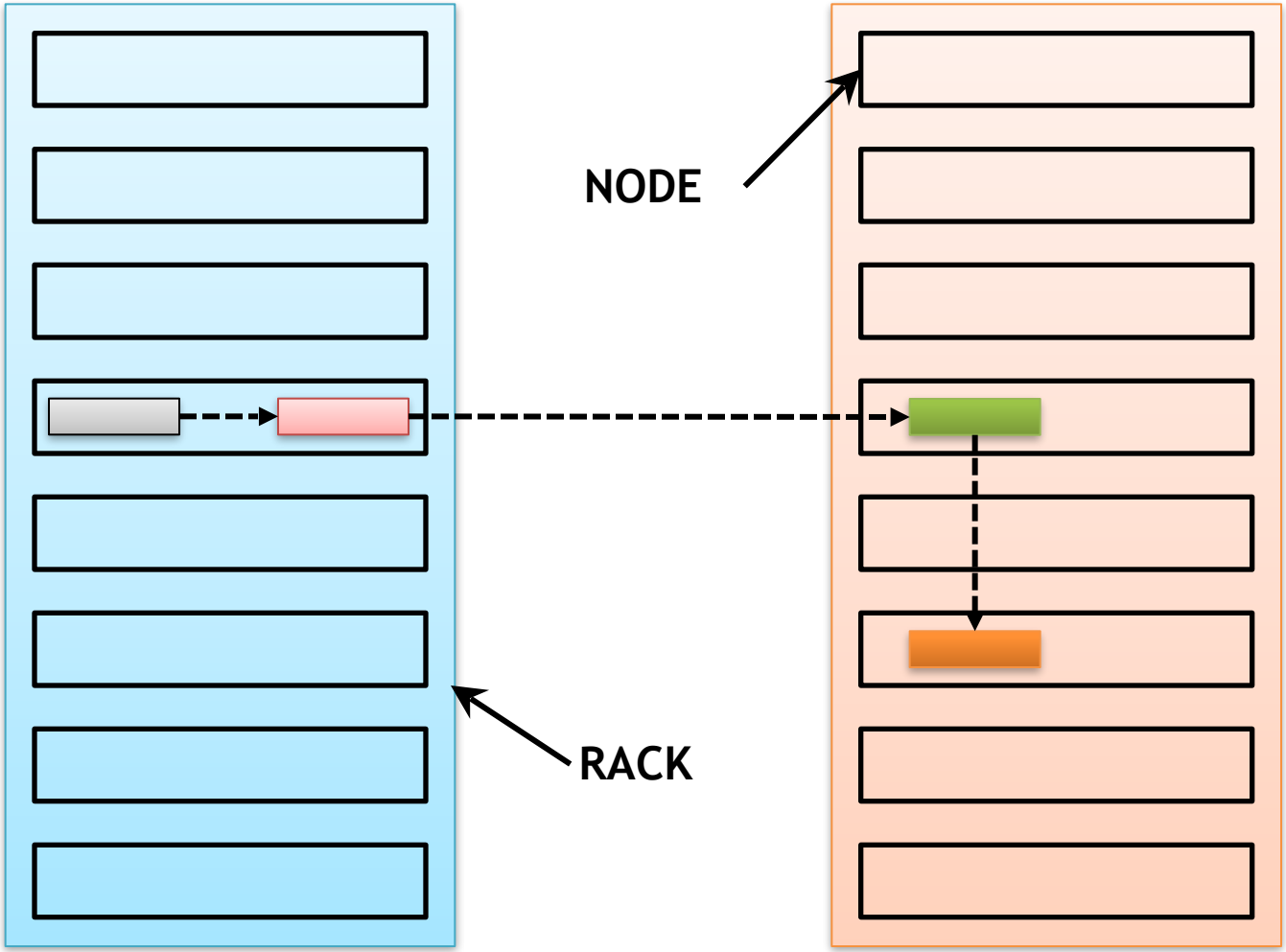
Rack Awareness



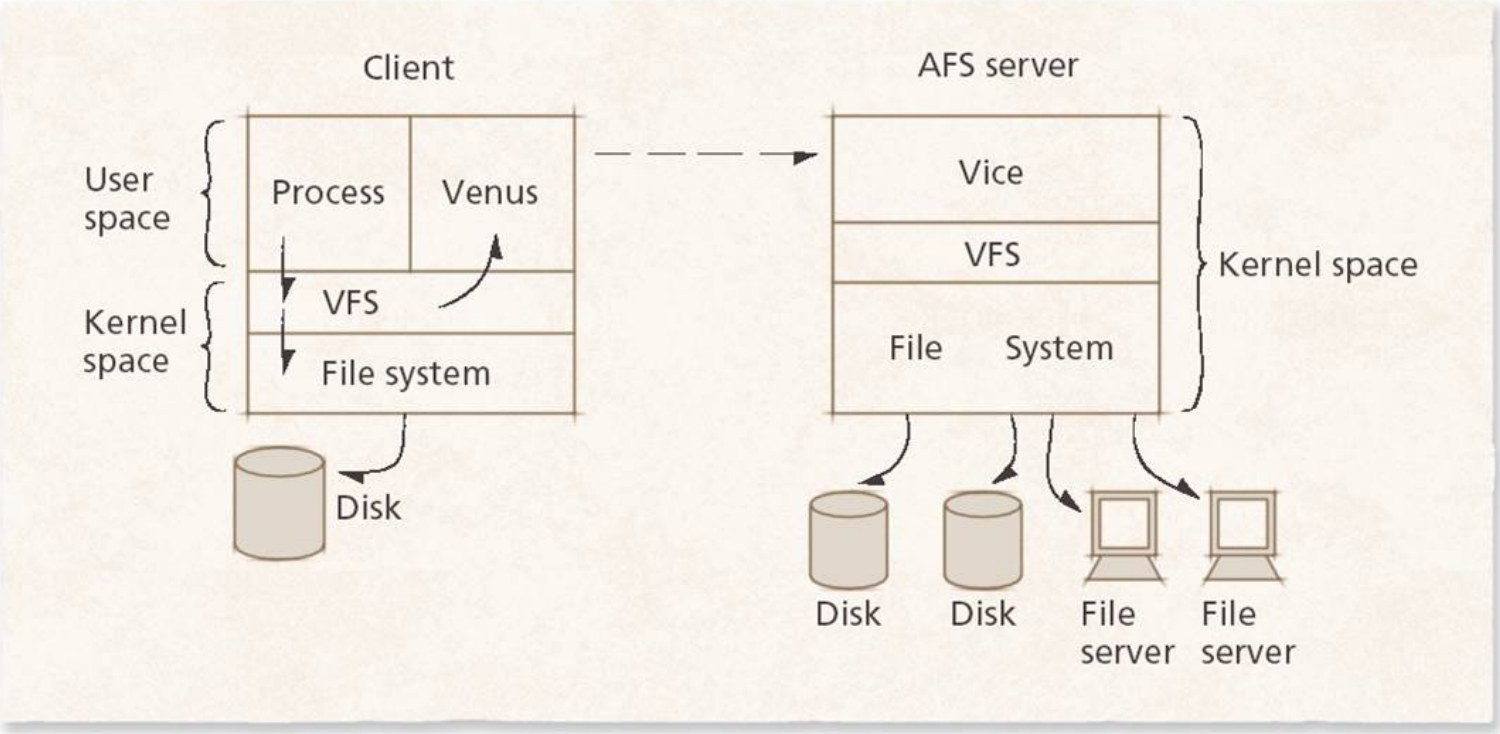
HDFS Write


HDFS Client
1: create
Distributed FileSystems
3: write
6: close
FSData OutputStream
Client Node
2: create
7: complete
NameNode
namenode
4: write packet
5: ack packet
Pipeline
DataNode
datanode
4
5
DataNode
datanode
4
5
DataNode
datanode

Replica Placement



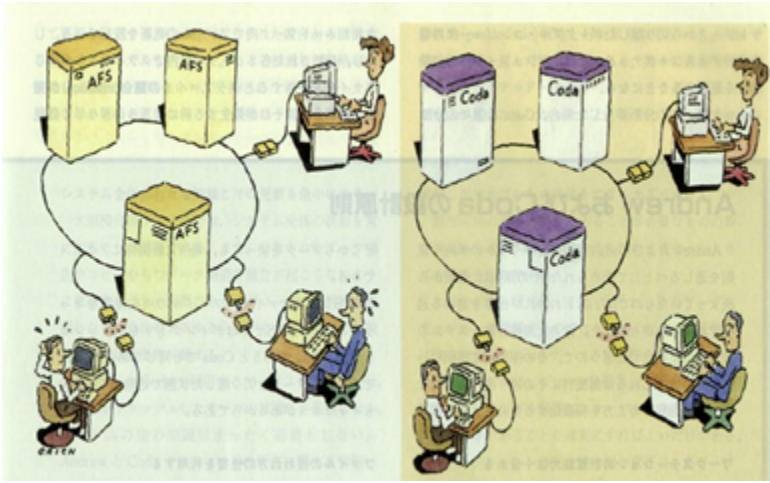
Andrew File System (AFS)



- Limited form of replication posed scaling problems
- Non-availability of services when servers and network components fail
- No catering for mobile use of portable computers

CODA File System

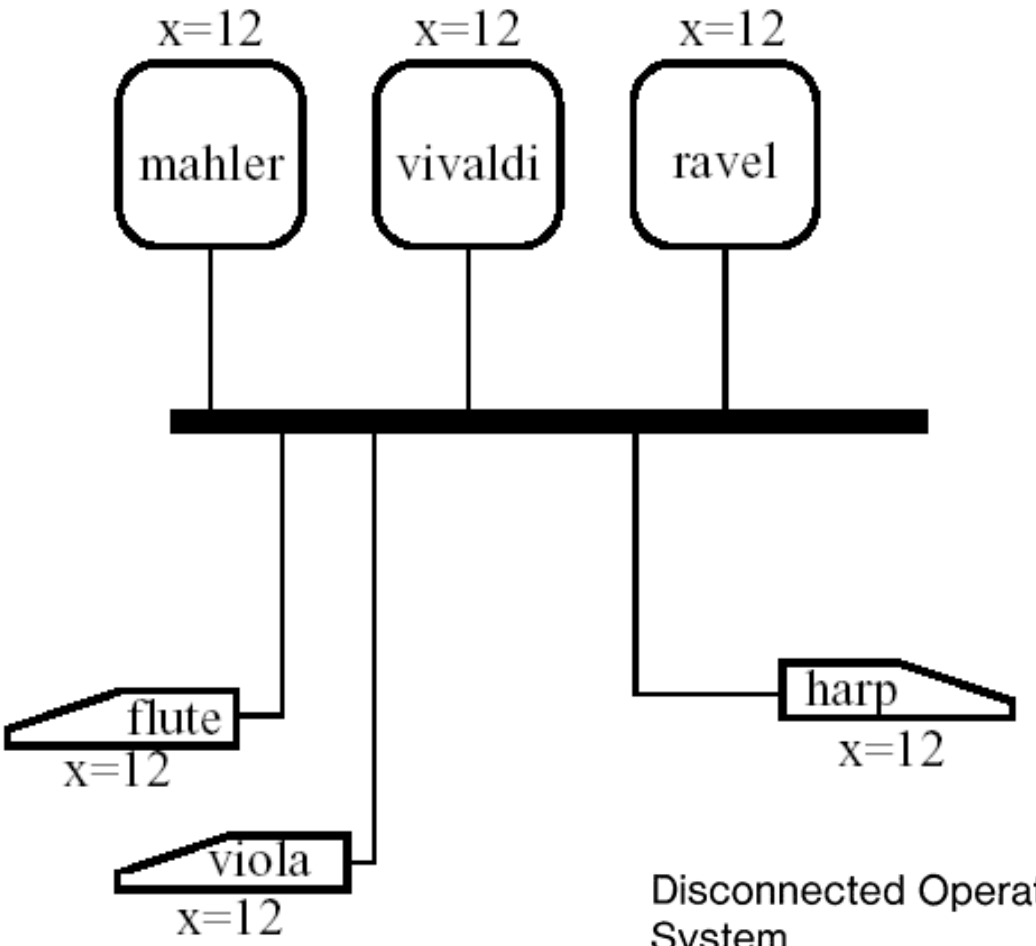
- disconnected operation for mobile computing
- high performance through client side persistent caching
- server replication
- security model for authentication, encryption and access control
- network bandwidth adaptation
- well defined semantics of sharing, even in the presence of network failures



CMU

more than 10, 000 workstations

An Example



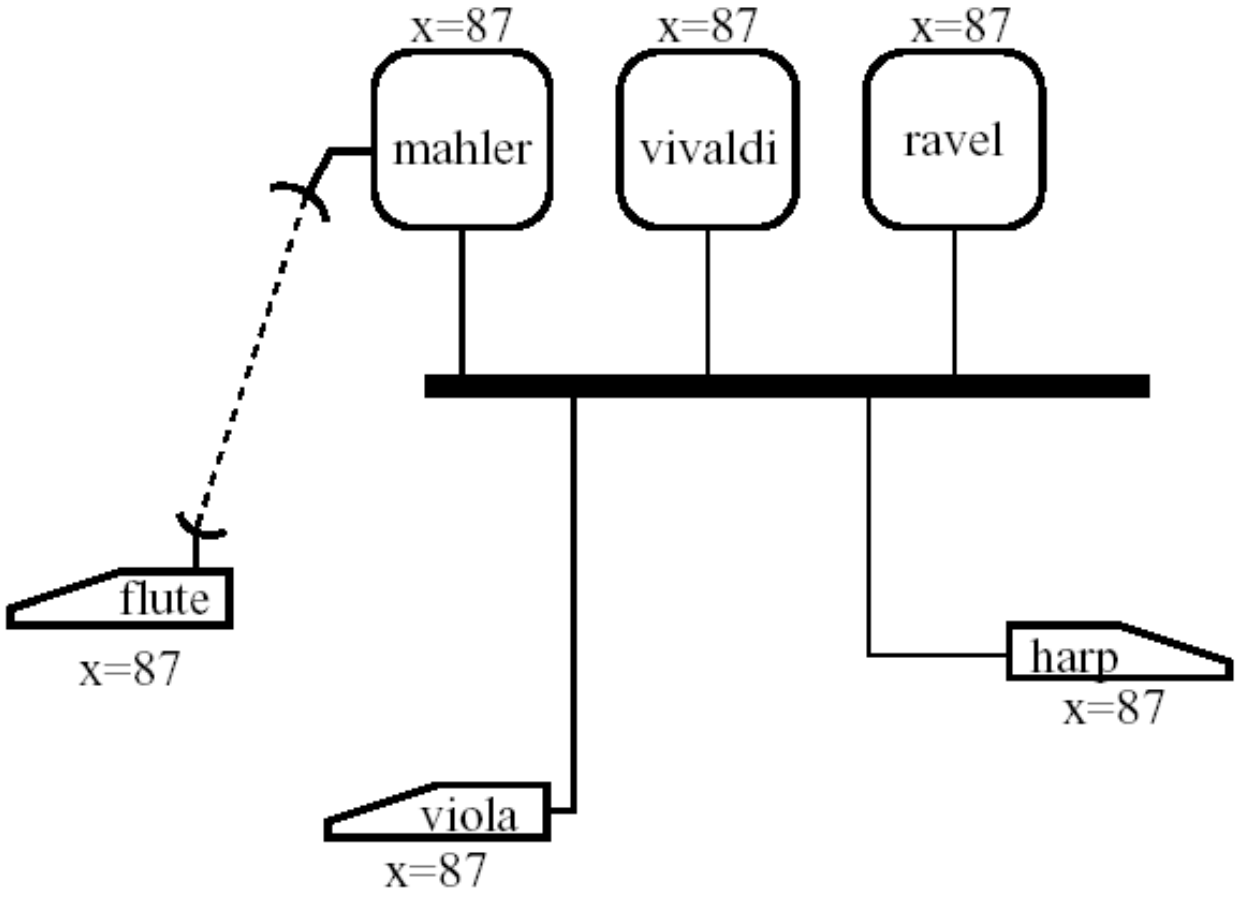
Disconnected Operation in the Coda File System

(a)

JAMES J. KISTLER and M. SATYANARAYANAN
Carnegie Mellon University

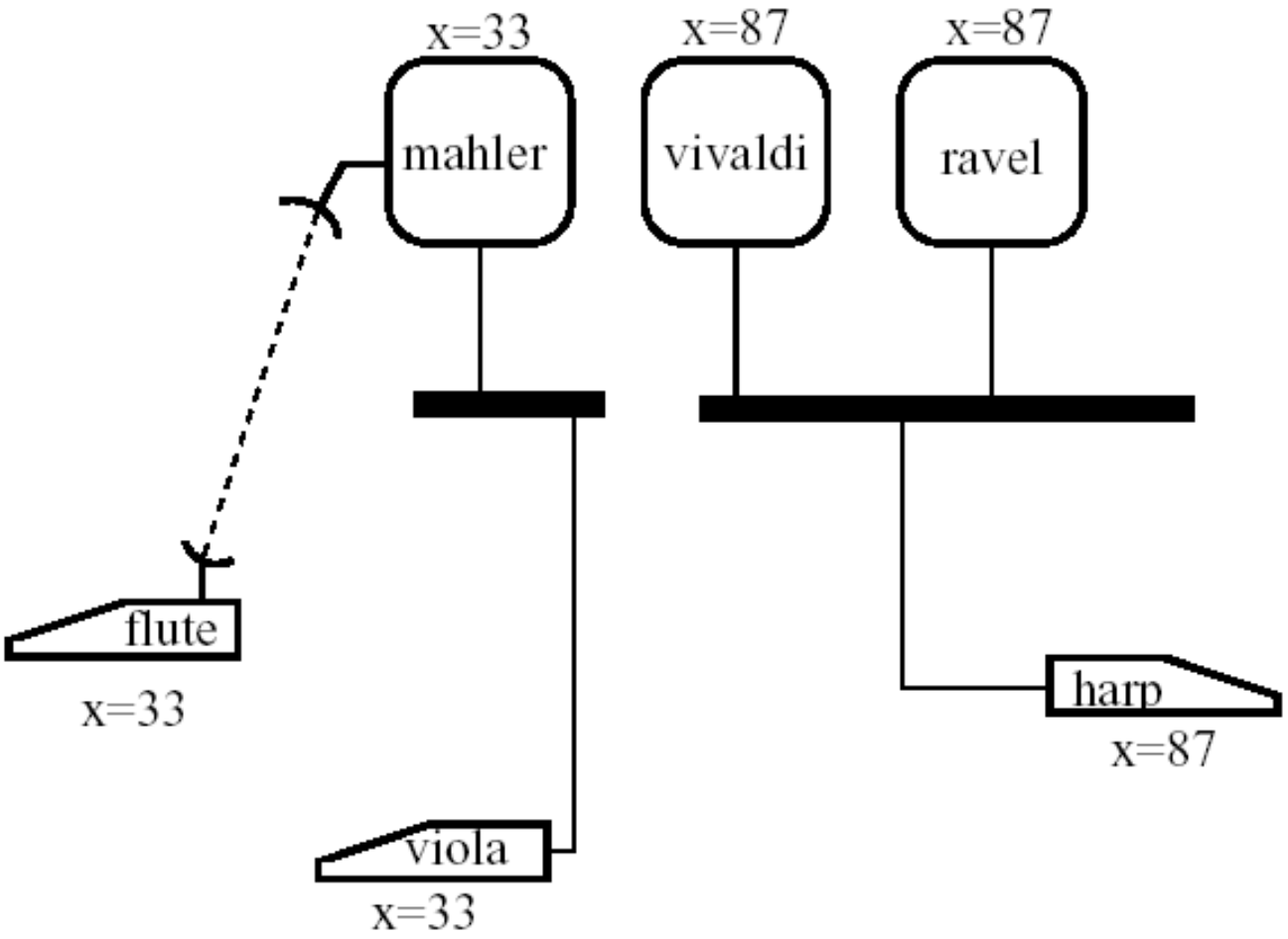
ACM Transactions on Computer Systems, Vol. 10, No. 1, February 1992, Pages 3–25.

An Example


x=87
x=87
x=87
mahler
vivaldi
ravel
flute
x=87
harp
x=87
viola
x=87

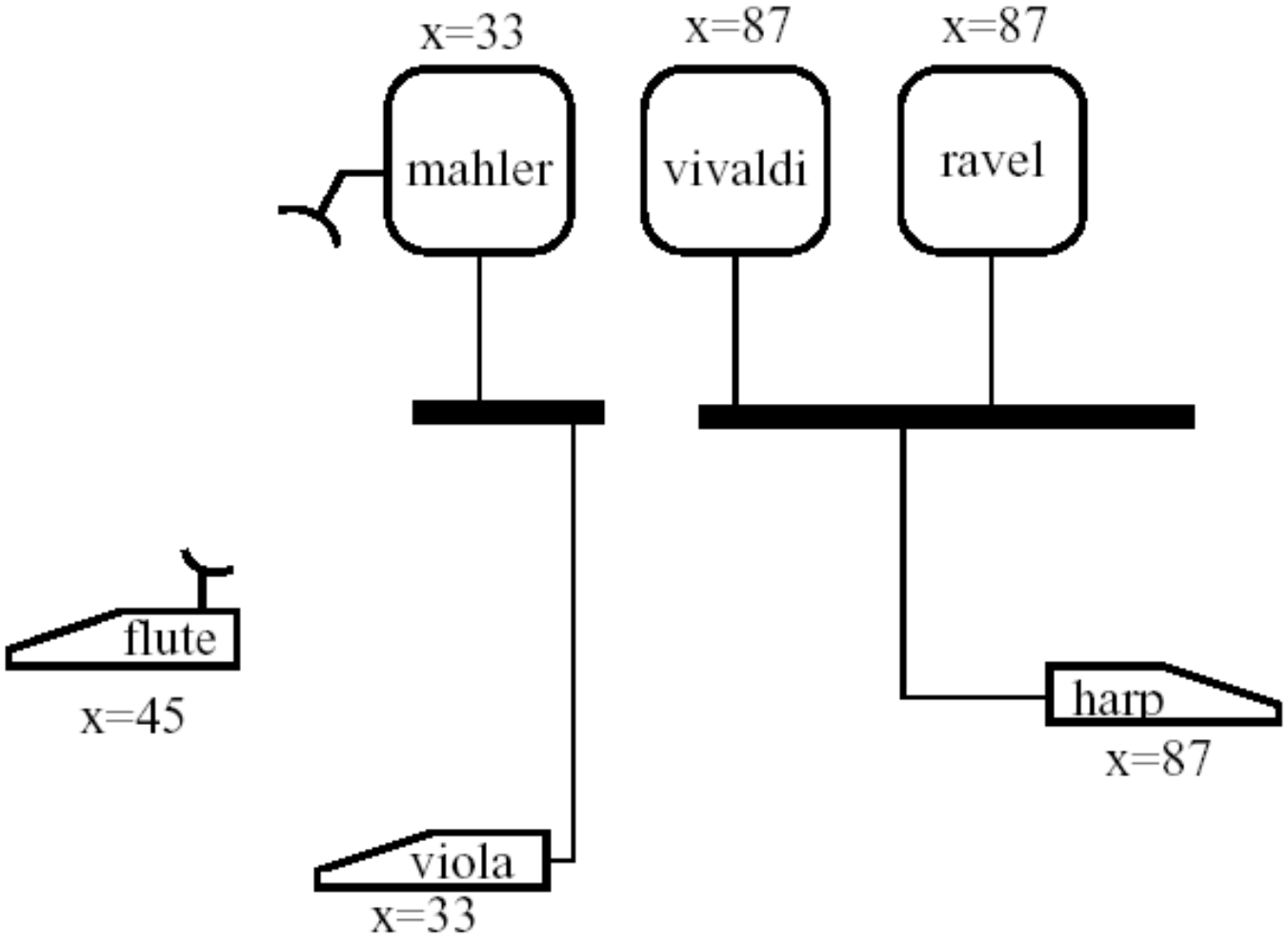
(b)

An Example


x=33
x=87
x=87
mahler
vivaldi
ravel
flute
x=33
harp
x=87
viola
x=33

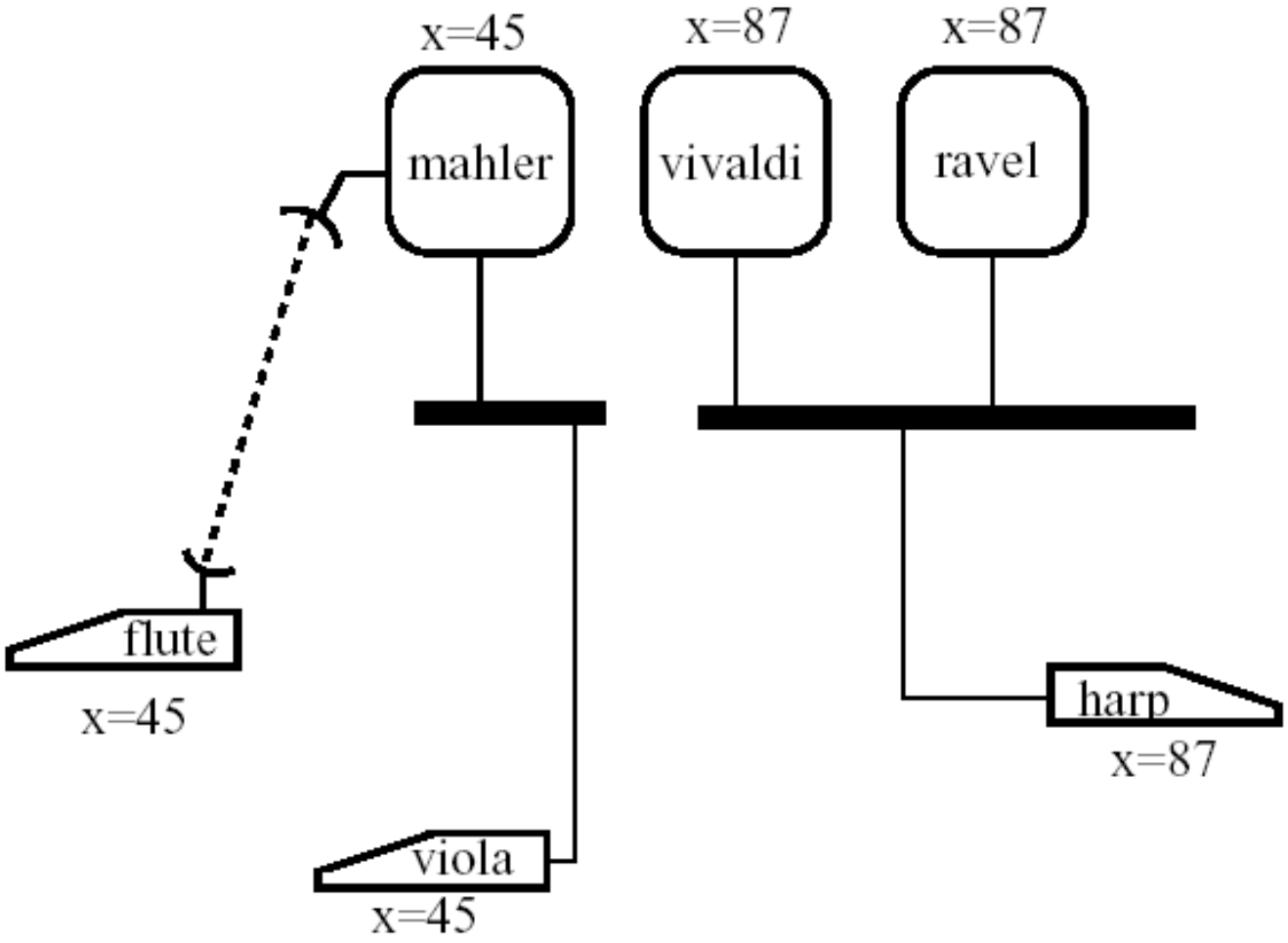
(c)

An Example


x=33
x=87
x=87
mahler
vivaldi
ravel
flute
x=45
harp
x=87
viola
x=33

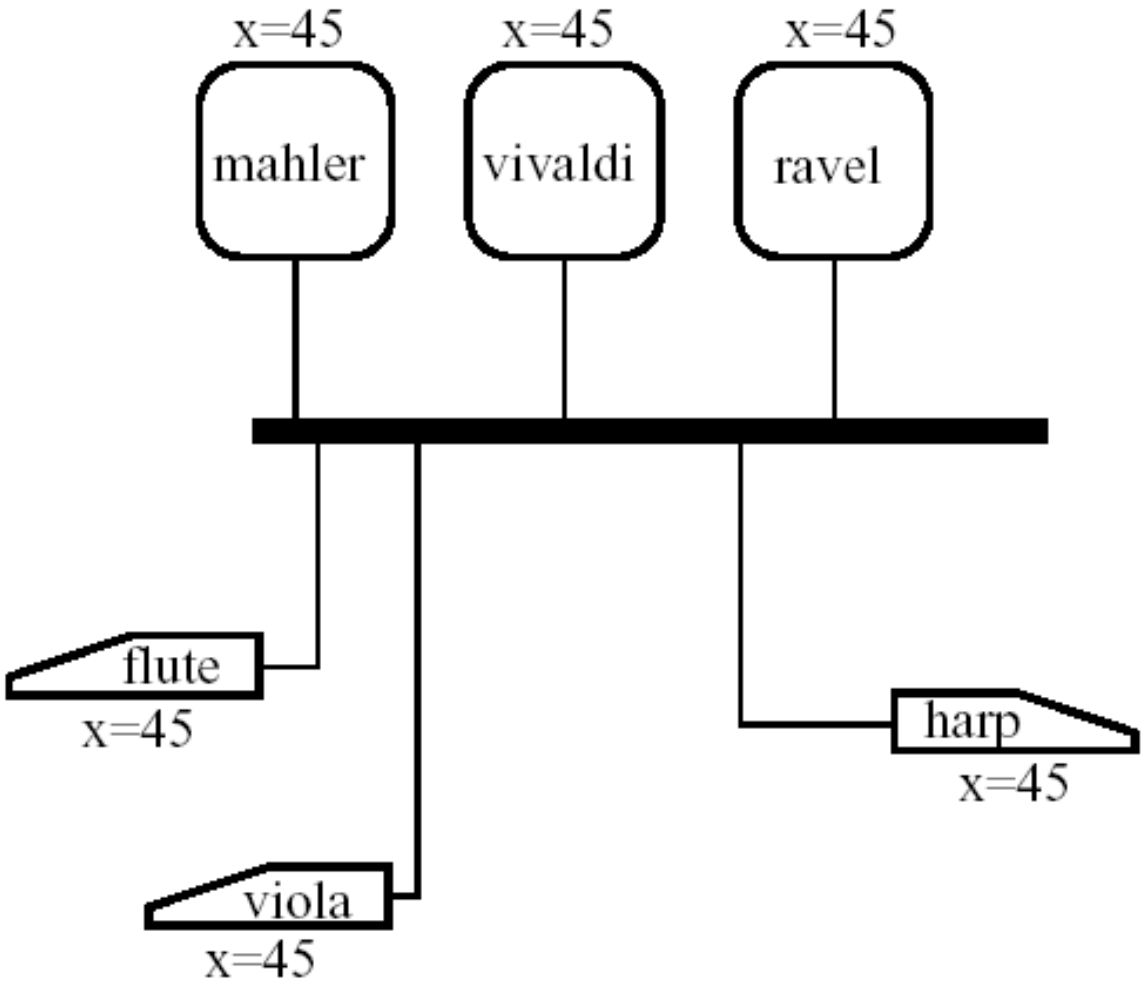
(d)

An Example


x=45
x=87
x=87
mahler
vivaldi
ravel
flute
x=45
harp
x=87
viola
x=45

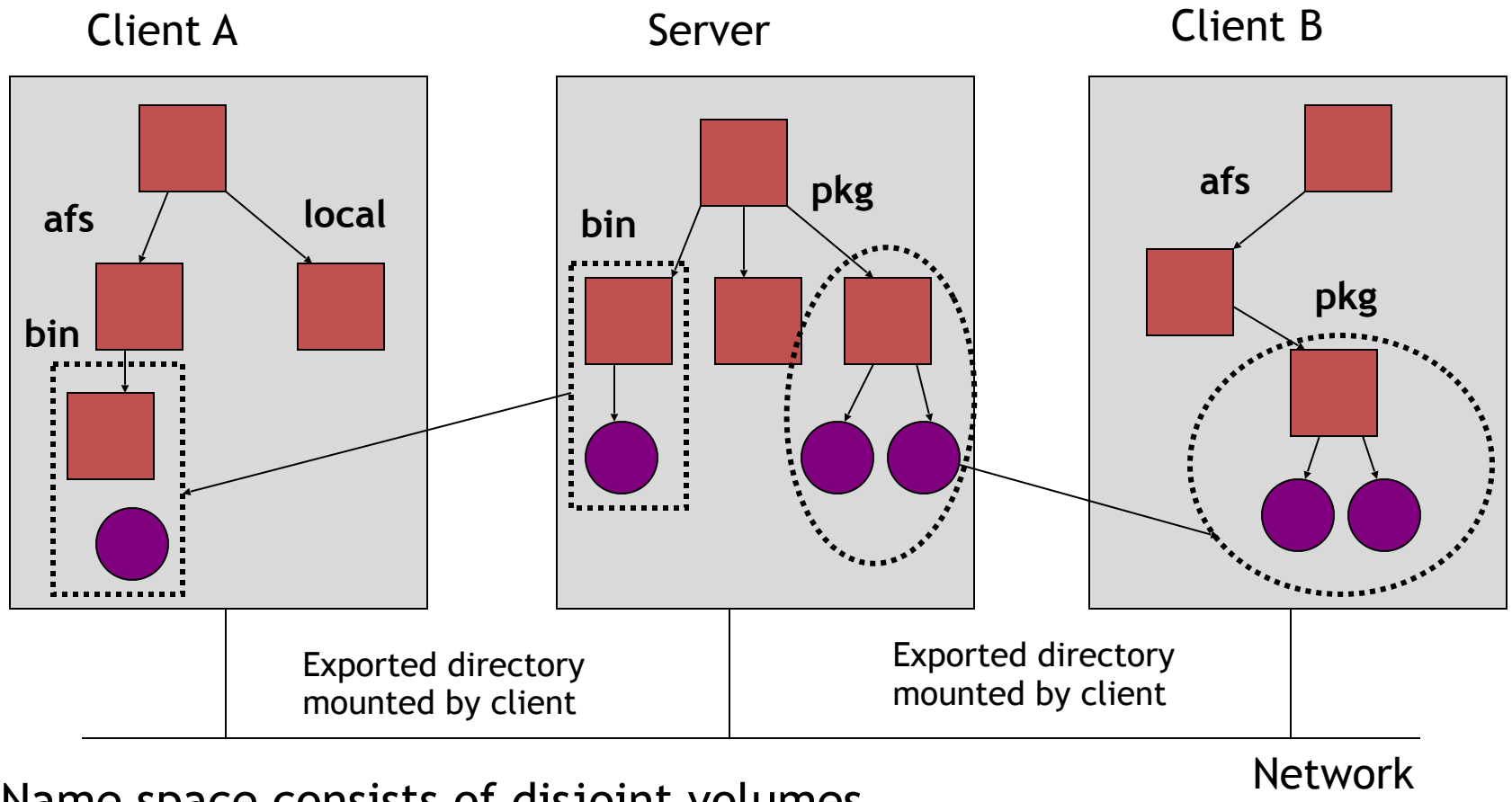
(e)

An Example


x=45
x=45
x=45
mahler
vivaldi
ravel
flute
x=45
harp
x=45
viola
x=45

(f)

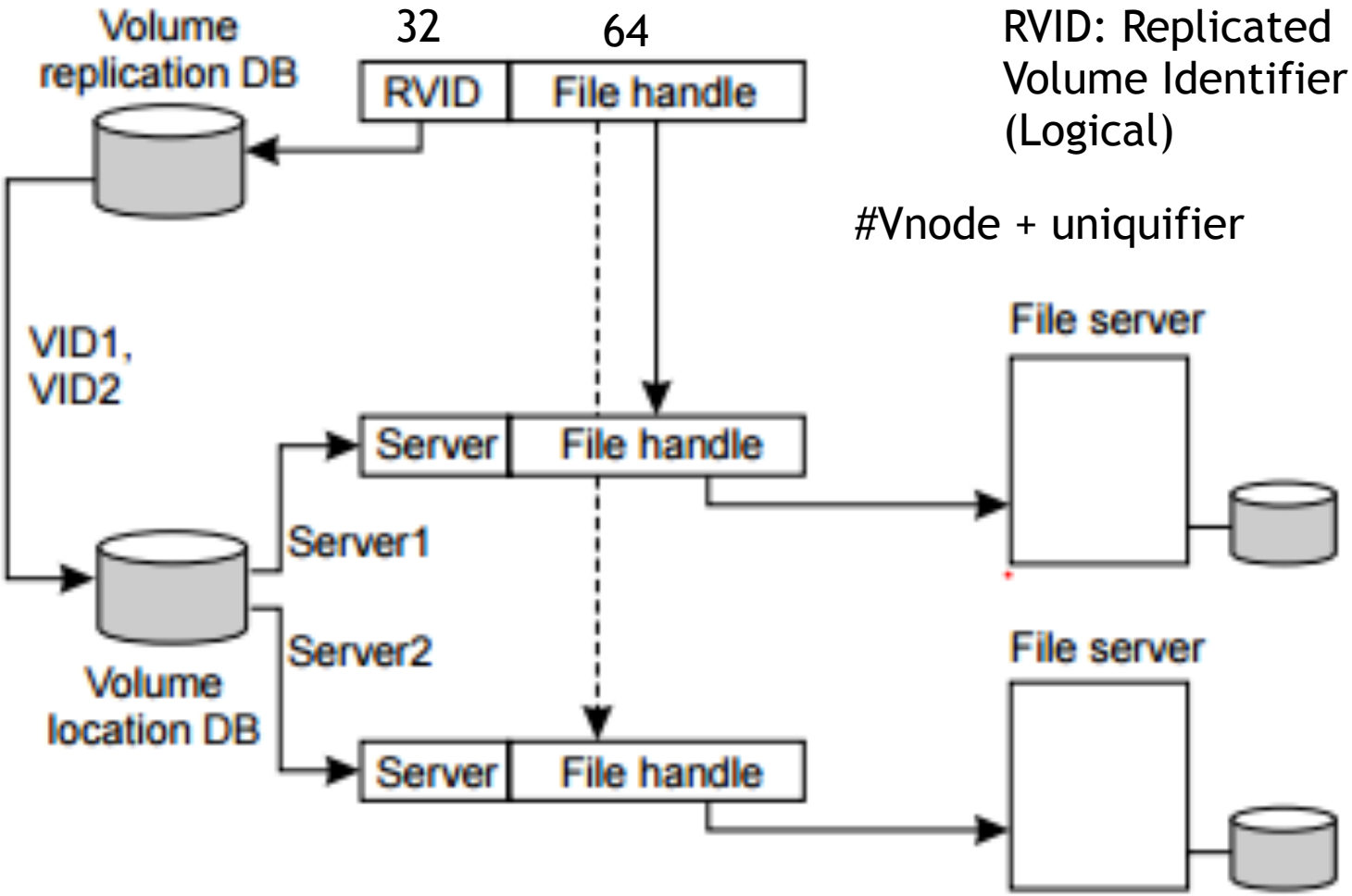
Naming and Location in Coda



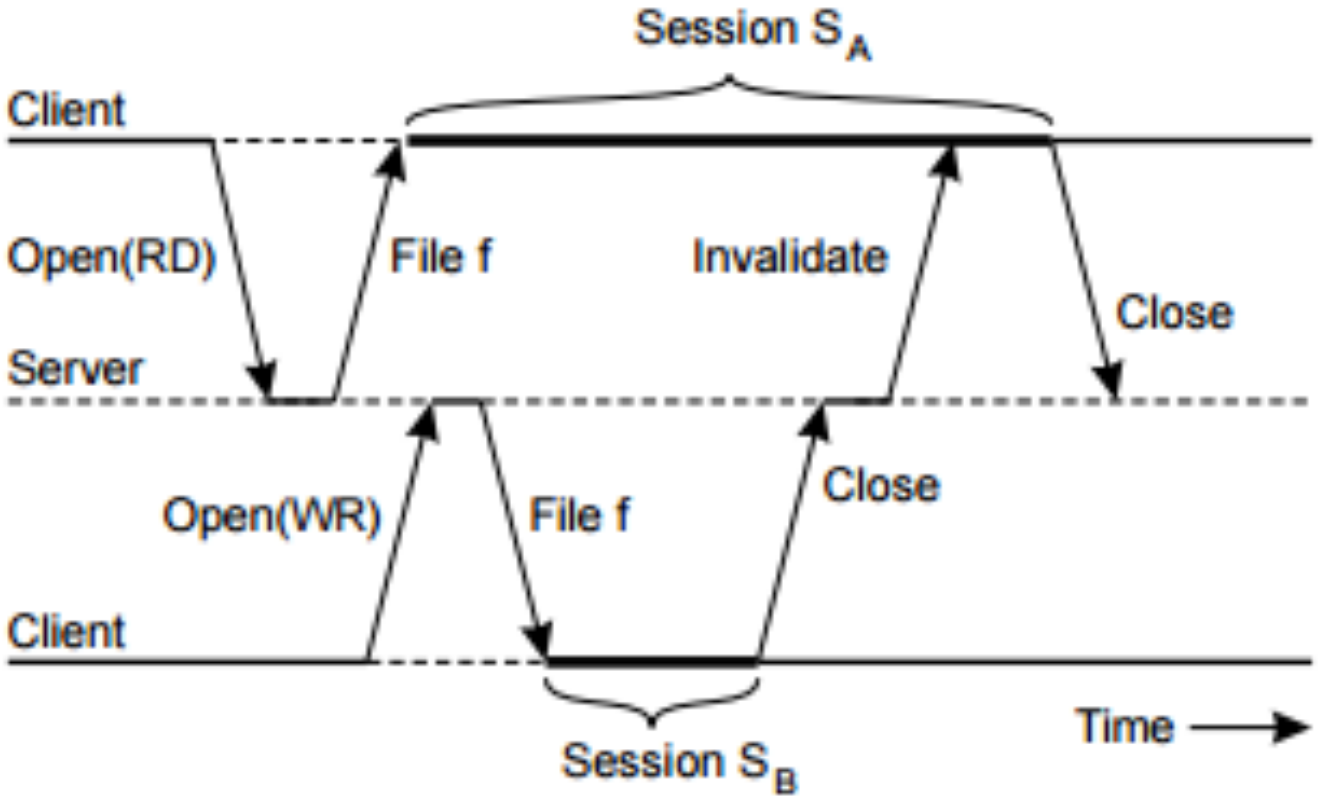
Name space consists of disjoint volumes.

Volumes: logical pieces of the file system, and are replicated physically across multiple file servers.

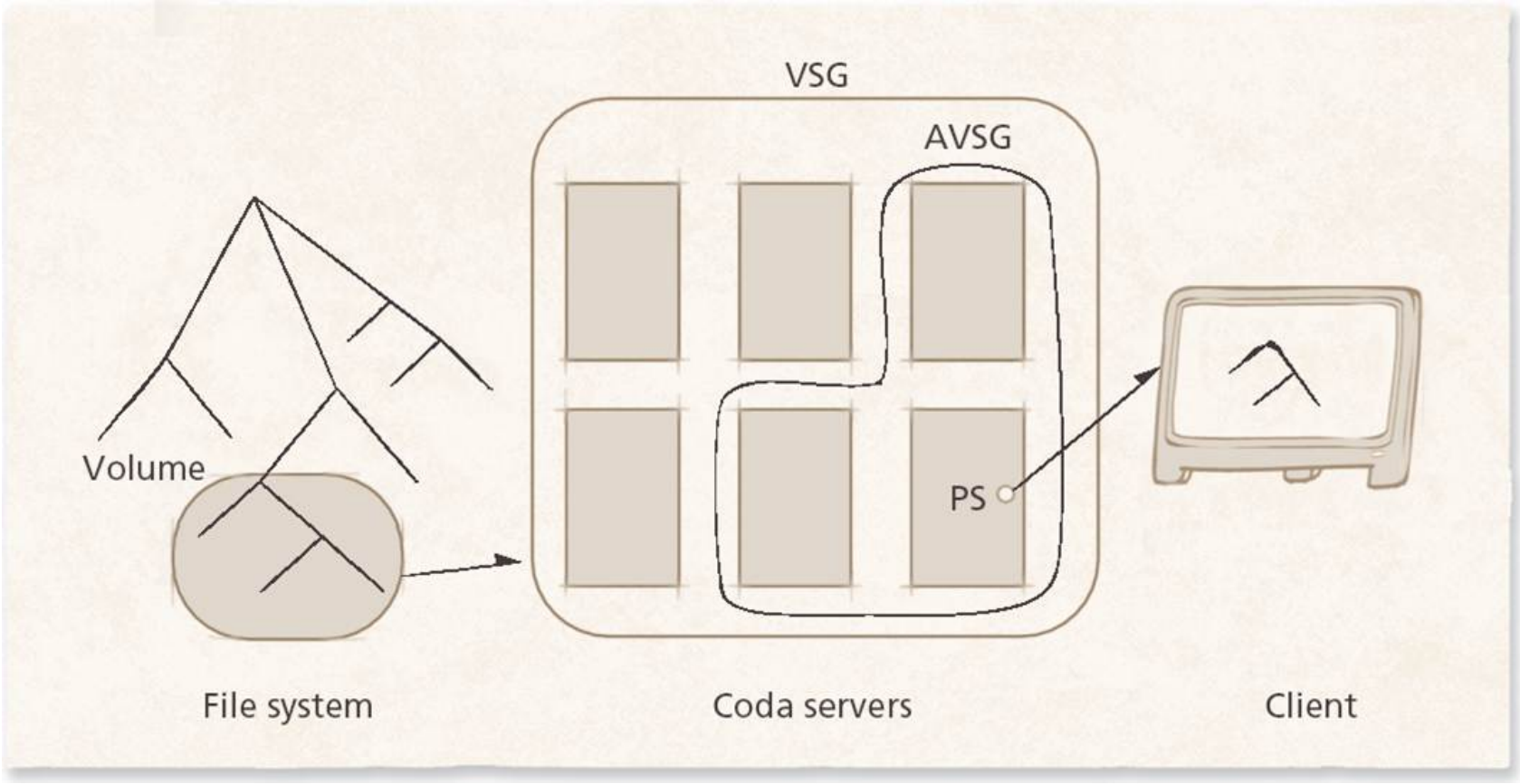
File Identifiers (96-bits)



File sharing (Synchronization) in Coda



Coda Volume Structure



Venus (client cache manager) keeps track of accessible volume storage group

- Enlargement and Shrinking of AVSGs are possible

Coda read/write scenario

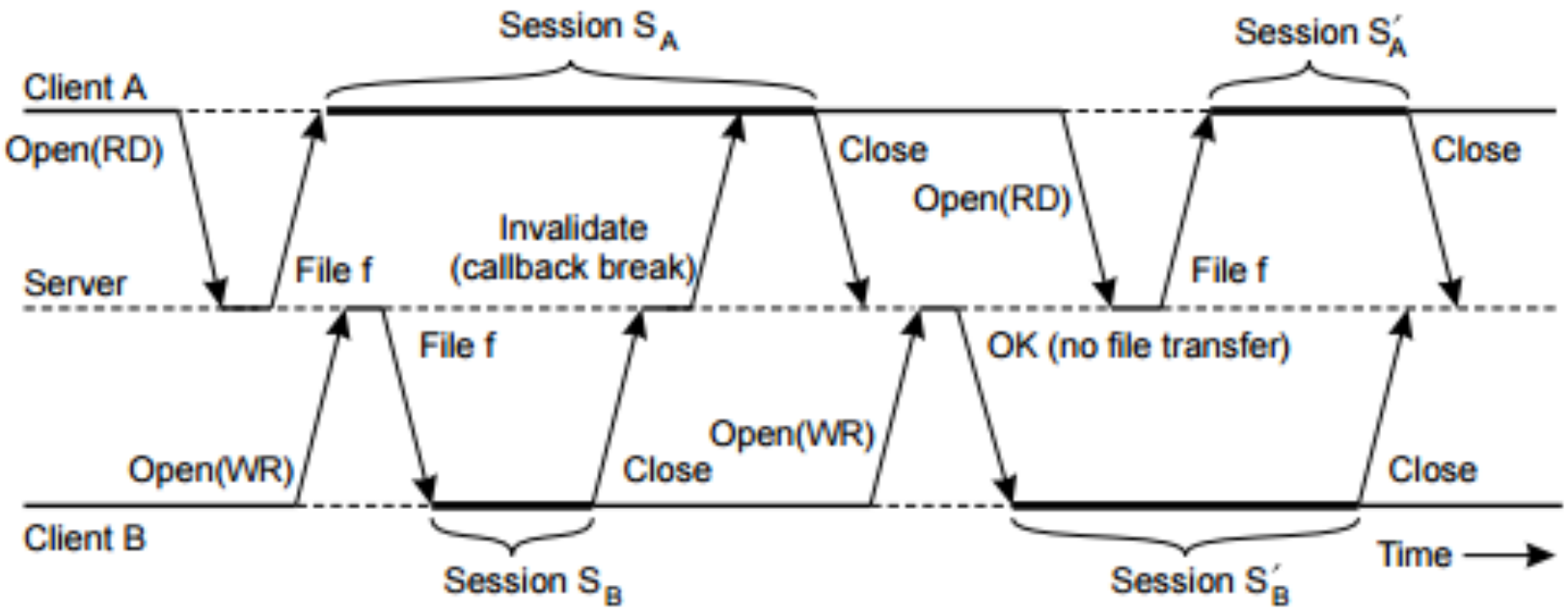
- to read, check cache for file
  - if found, done
  - else
    - get file from PS
    - get file versions from other members of AVSG
    - if conflict, then AVSG members agree on new versions, and update their copies
- to write,
  - send file to all members of AVSG
  - track members of AVSG that have written file

CODA

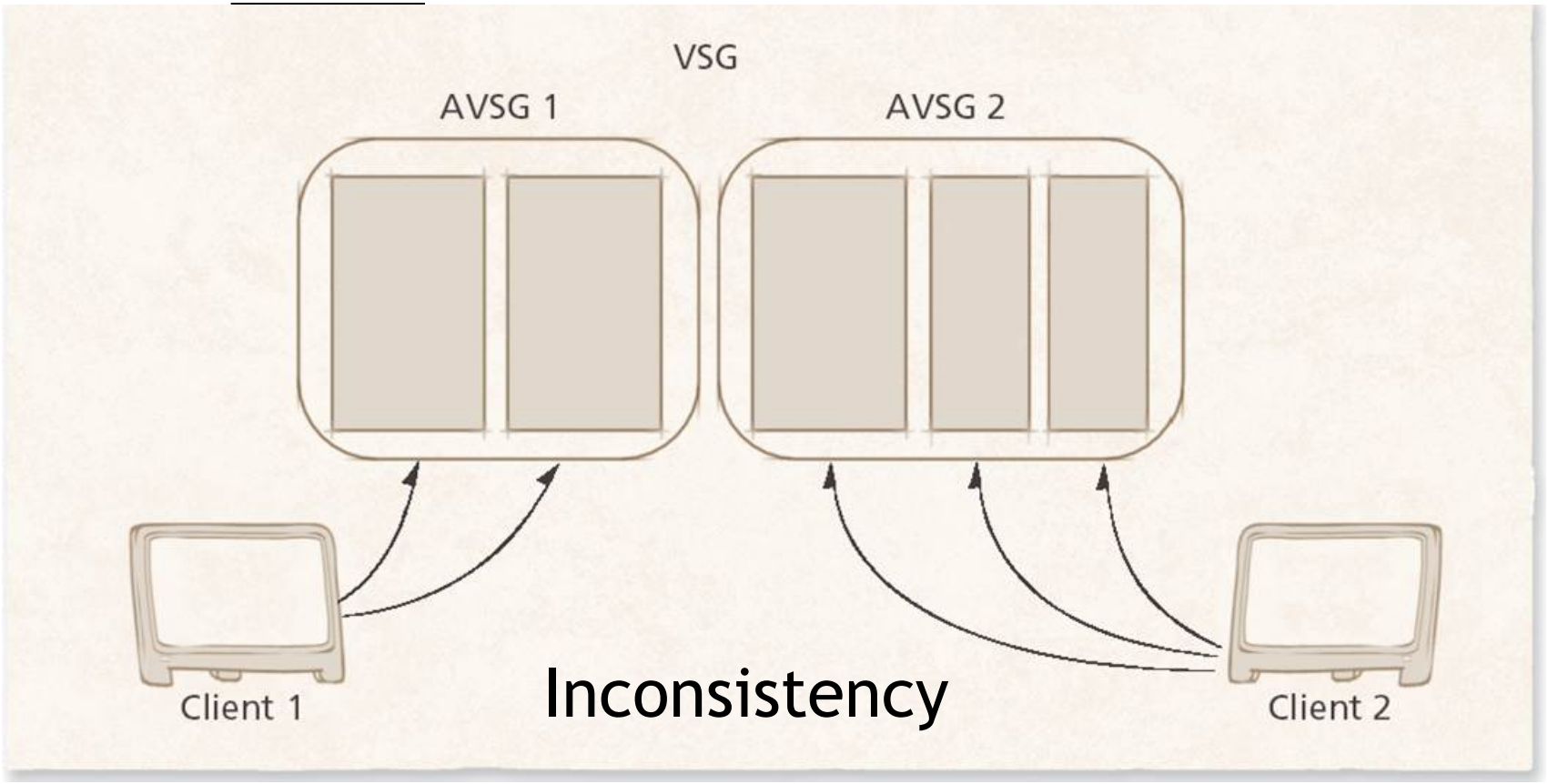
- Coda does not prevent inconsistencies, it detects them.

Client caching in Coda

- Callback promise and Callback break



Server replication



Two clients with different AVSG for the same file ‘f’

Writing in Disconnected Systems

- Each file has a Coda Version Vector (CVV), analogous to vector timestamps, one component per server. Starts at (1, 1, 1).
- Update local component after a file is updated.
- As long as all servers get all updates, all timestamps will be equal

Detecting Inconsistencies

- In the previous example, both A and B will be allowed to open a file for writing.
- When A closes, it will update S1 and S2, but not S3; B will update S3, but not S1, S2.
- The timestamp at S1 and S2 will be [2, 2, 1].
- The timestamp at S3 will be [1, 1, 2].
- It is easy to detect the inconsistency, but knowing how to resolve them is application dependent.

Continued…

- Coda also maintains for each replica (in addition to the CVV)
  - Its ***last store ID (LSID)***
  - The ***length v*** of its update history (in other words, a ***version number***)

Example

- Three copies
  - **A:**
    LSID= 33345 v = 4 CVV = (4 4 3)
  - **B:**
    LSID= 33345 v = 4 CVV = (4 4 3)
  - **C:**
    LSID= 2235 v = 3 CVV = (3 3 3)

Example Continued…

- Coda compares the states of replicas by comparing their LSID's and CVV's
- Four outcomes can be

1. ***Strong equality***: same LSID's and same CVV's
   - Everything is fine
2. ***Weak equality***:
   Same LSID's and different CVV's
   - Happens when one site was never notified that the other was updated
   - Must fix CVV's

Example Continued…

3. ***Dominance /Submission:*** LSID's are different and every element of the CVV of a replica is greater than or equal to the corresponding element of the CVV of the other replica

Example: two replicas ***A*** and ***B***

$\boldsymbol{CVV_A}$ **= (4 3)**

$\boldsymbol{CVV_B}$ **= (3 3)**

Who has the most recent version of the file?

Example Continued…

4. Inconsistency: LSID's are different and some element of the CVV of a replica are greater than the corresponding elements of the CVV of the other replica but other are smaller:

Example: two replicas *A* and *B*

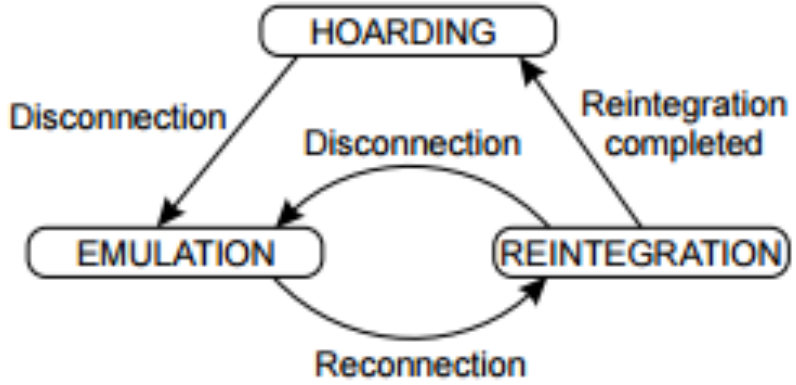
$CVV_A = (4\ 2)$

$CVV_B = (2\ 3)$

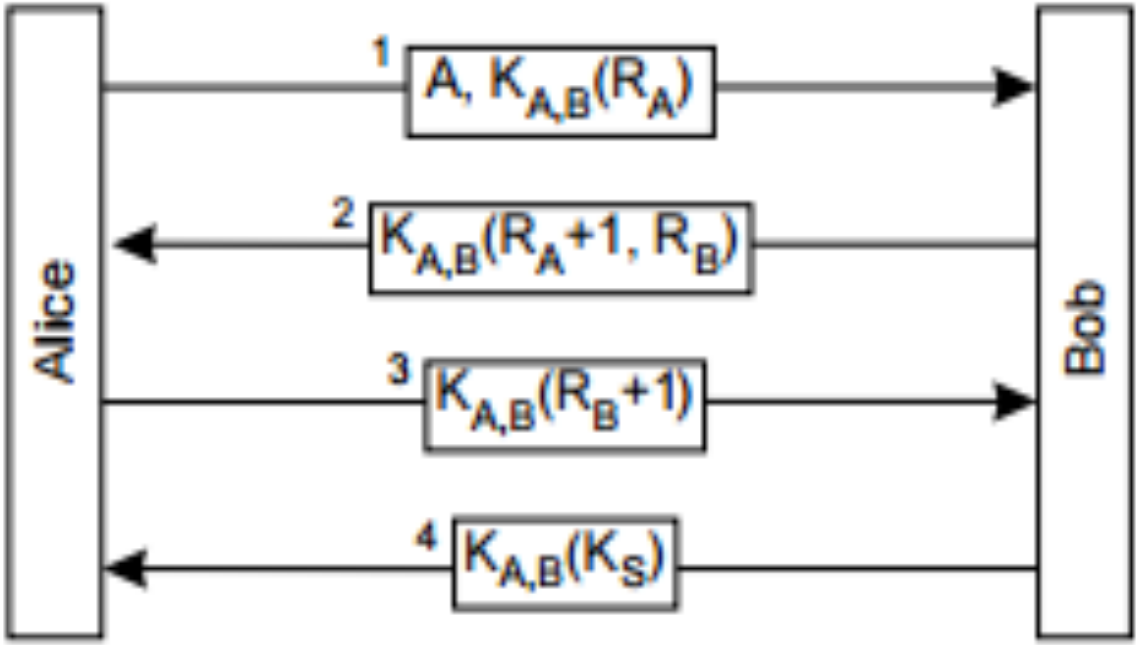
Must fix inconsistency before allowing access to the file.

Coda client disconnect scenario: Fault tolerance

- When connected to Coda, clients cache files so they can be accessed when disconnected (hoarding stage)
- When disconnected, clients enter the emulation stage where all file requests are serviced from the cache, if the file is resident (error otherwise)
- When reconnected, file updates are sent to the server asynchronously (reintegration stage)



Secure Channels in Coda



Mutual authentication

The X-kernel logical File System

- Experimental one
- Allows uniform access to resources on a nationwide internet
- Provides only a directory service
- Provides an interface with which we can access heterogeneous physical file systems like SUN NFS , AFS, Coda etc.
- It simply maps a file name to the location where it can be found , under its directory services.
- Each user defines his private file system out of existing physical ones.

The X-kernel logical File System

- Experimental one
- What does it mean?
- It can be implemented in lab and code can be modified based on requirement. No a commercial one.

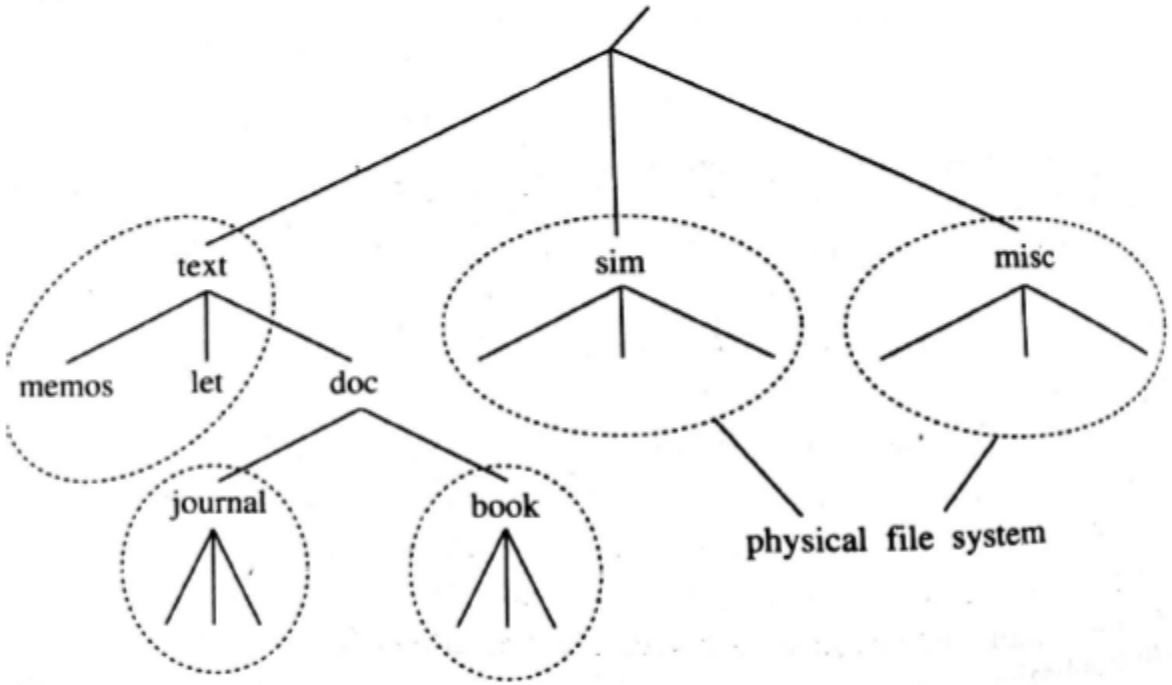
- Motivation
- Allows uniform access to resources on a nationwide internet

- Provides only a directory service
- What does it mean??
- No read, write, lseek into a file is allowed in x-kernel.
- relies entirely on an existing physical file system for storage

Contd.

- Provides an interface with which we can access heterogeneous physical file systems like SUN NFS , AFS, Coda etc.
- It simply maps a file name to the location where it can be found , under its directory services.
- Each user defines his private file system out of existing physical ones.
- Logical file system simply maps a filename to the location where it can be found.
- Physical file system is must even for local file access

Private file system in x-kernel



Logical file system.

Circles represent physical file system. Can also be logical file system

What are the operations possible in this file system?

Contd.

- File system has two unique features

1. Each user defines his or her own private file system out of the existing physical file systems
   - User constructs a file system by using logical make directory operation,
   - Lmkdir (<name>,<physical file directory>) to mount a file system into a directory
2. Logical file read directory operation, Lreaddir, allows the reading of logical file system

- Apart from these two special operations, user's private file system behaves like a UNIX file system.
- Lmkdir (name, physical file directory) is used to mount a file system to a directory.
- Lreaddir allows reading of a directory.

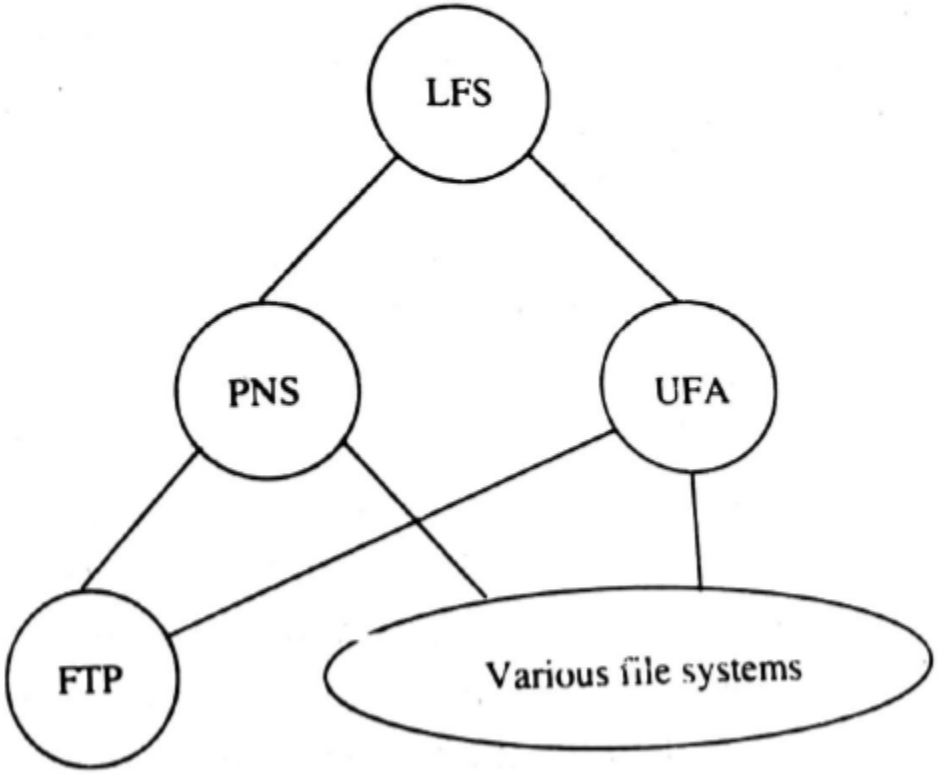
- Two protocols are used to separate directory functions from storage function

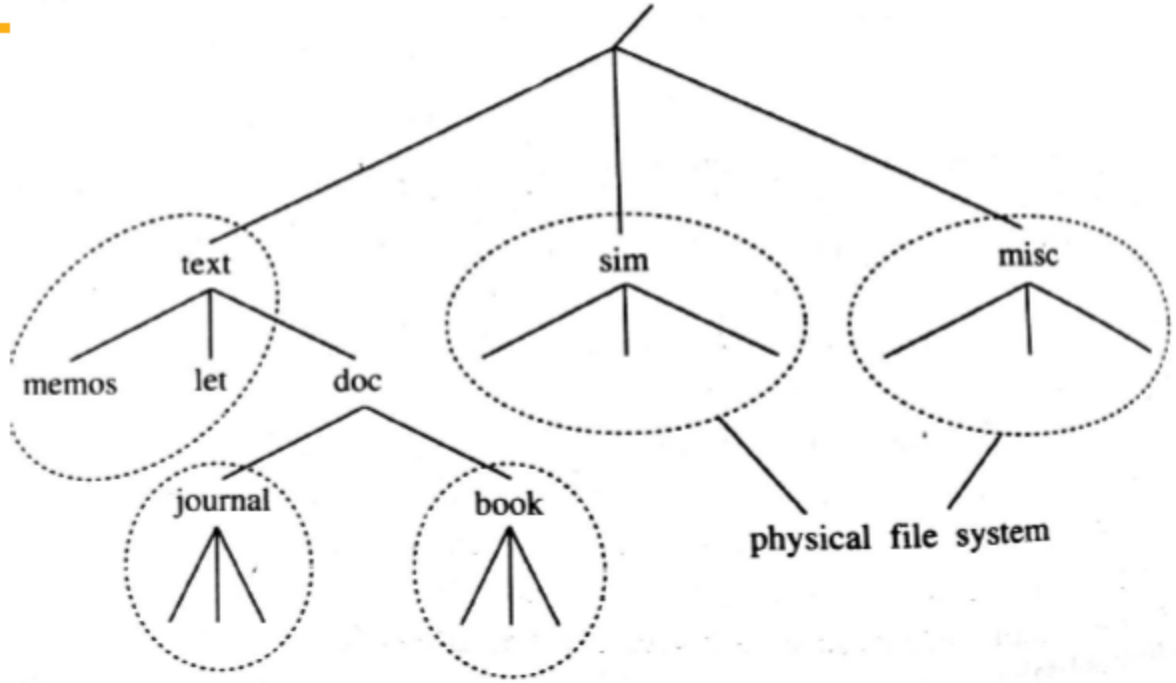
1. Private name space:

- implements the directory function
- Maintains a Skelton directory
- Superimposes a logical hierarchy over a collection of existing file hierarchies
- Directory entry contains three parts
  - Contains Logical file name: corresponds to the logical file directory or logical file name
  - Associates the logical directory to the physical directory. Identifies the physical file system and server maintaining the physical directory
  - a pointer to the private directory

Uniform file access:

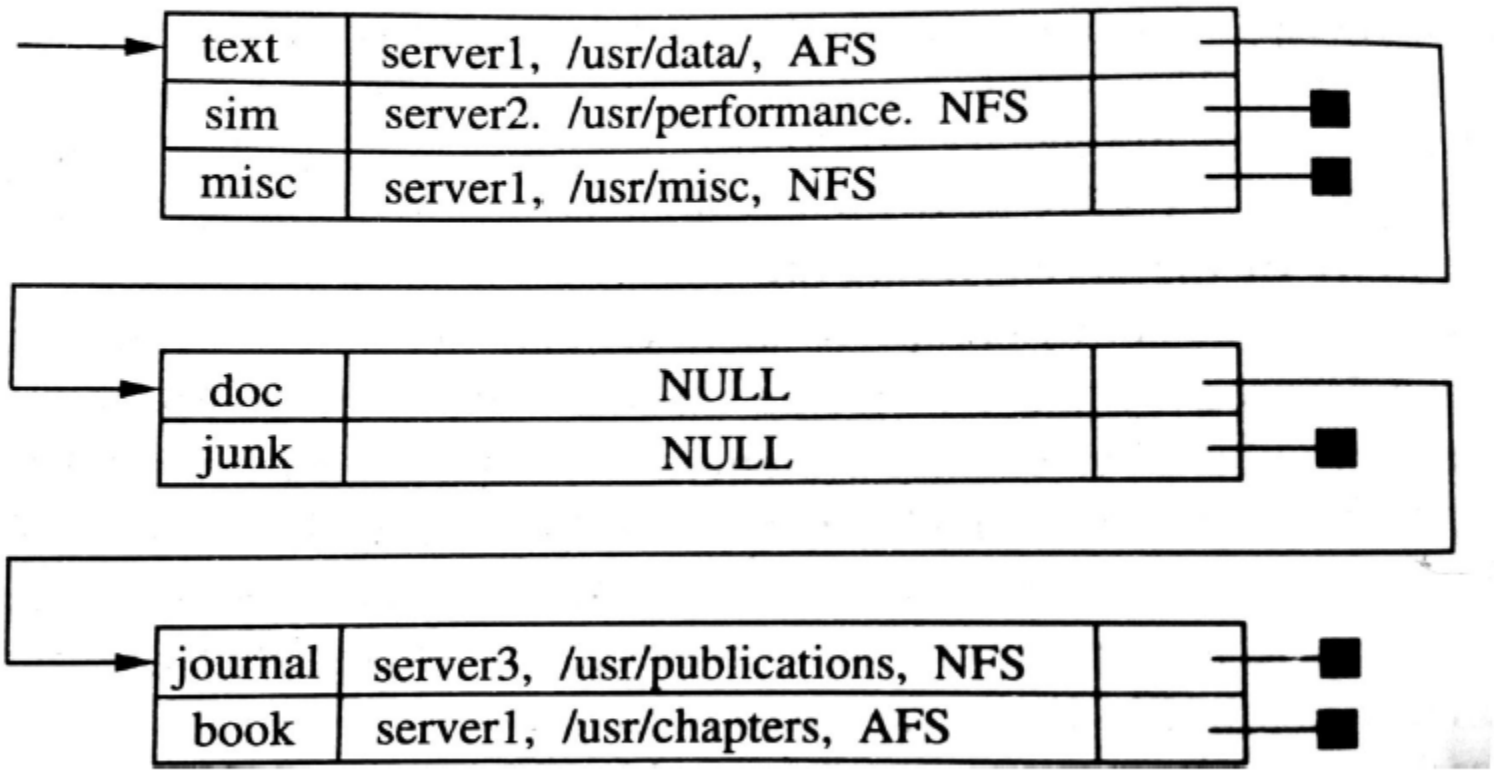
- Translates file system operations (read, write) into appropriate low level commands used by underlying physical filesystem
- Handles caching
- If any particular file system is not supported by UFA, it calls FTP to transfer disk file into an existing one and handles it.
- UFA caches the retrieved file as temporary file to one of the supported file systems.
- Operations on the file are appropriately translated.
- On close, FTP updates the remote copy.


LFS
PNS
UFA
FTP
Various file systems



- Directory entry contains three parts
  - Contains Logical file name: corresponds to the logical file directory or logical file name
  - Associates the logical directory to the physical directory. Identifies the physical file system and server maintaining the physical directory
  - a pointer to the private directory

Skeleton directory



Search for /text/doc/journal/paper/1 in PNS protocol

Invokes server 3 to resolve /paper/1 in its physical file system (Entry directory)

Issues with X-kernel

1. File system are unaware that they might be participating in some other user's private file system
2. The pathname of the logical file system can take precedence over the pathnames of physical system
3. PNS would not recognize /usr/data/doc as it is hidden by private directory /text/doc.

Log structured file systems

- By caching frequently accessed data, file systems can satisfy most read requests without communicating with disk subsystems
- Caching serves as buffers
- Number of modified blocks can be collected before writing to disk.
- Improvement in data transfer bandwidth and storage capacity
- Not much in access time/seek time, as it depends on mechanical motion

Existing bottlenecks

- Improving file system performance has following challenges

1. The existing file systems physically spread files belonging to same directory on disk sector.
   - Attributes are stored separately from files
   - Requires many seek operations for modification of file
   - Seek time is highest consuming operation
2. Write synchronously, waiting for previous write to finish before it can proceed.
   - Instead of continuing execution and let write execute in background.
   - Data transfer is asynchronous but writes to directories and files attributes are still synchronous.

- An application performing small disk transfers, separated by many seeks
- Not likely to experience speed up in near future
- Despite an increase in processor speed, and use of buffered writes

Solution??

Log-structured file systems

Log-structured file systems

- Files are cached in main memory
- File updates are carried out in the main memory
- Updates are asynchronously written to disk in sequential structure, called log
- Updates written in a single write operation
- Information written includes
  - data blocks
  - Directories
  - Attributes
  - Other information required to manage a file system

Contd..

- To locate data efficiently on the disk, structures that store disk addresses of files are stored at a fixed location on disk.
- These structures are cached
- Eliminates the necessity to sequential search a log to retrieve a file
- E.g. sprite log structured file systems stores inodes on the log
- An inode map that maintains the current location of each inode is also written at a fixed location on the disk
- Inode map is compact to be cached in the main memory.

Crash recovery

- Inode stores

1. File attributes including type, owner, permissions, etc.
2. Disk addresses of the first ten direct blocks
3. One or more indirect blocks

- To restore the consistency of files, most recent portion of log need to be examined
- In conventional file systems, entire disk must be searched to restore the consistency of files
- Files are physical spread out on the disk

Disk space management

- Large extents of free disk space must always be available to write new data
- Over time free disk space is fragmented as files are deleted and overwritten
- To reclaim the free space two approaches are used

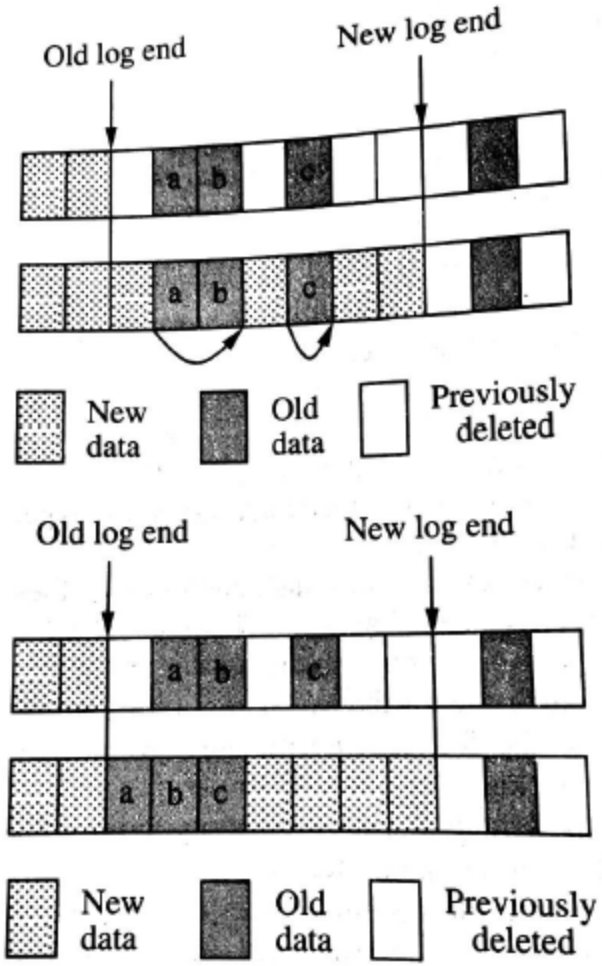
1. Threading
2. copying

Threading

- Live data is left in its place
- Log is spread across the free spaces
- Disk space eventually becomes severely fragmented
- Requires many disk seeks to write a log
- The performance of log-structured file system is no faster than traditional file systems

copying

- Live data is copied out of logs into a compact form
- Freeing up large contiguous disk space
- Disadvantage??
- Cost of copying
- Long lived files can be copied over and over again
- When should copying begin?
- How much disk space should be freed up at a time?
- Which blocks are selected for copying data out?


Old log end
New log end
a
b
c
a
b
c
New data
Old data
Previously deleted
Old log end
New log end
a
b
c
a
b
c
New data
Old data
Previously deleted

Sprite log-structured file system

By same author J.K. Ousterhout in 1992

Other at UCB in 1988

- Uses both threading and copying
- Disk divided into large segments
- Reading and writing at segments more expensive than seeking at the beginning of segment
- Logs threaded through segments
- Segments overwritten contiguously from the beginning to the end
- Before a segment is rewritten all live data is copied out of the segment

Copy operation in sprite log-structured file system

- Copy operation to free disk space
- When available segments drops below threshold
- Till number of free segments exceeds other threshold
- Number of selected segments are read to memory
- Live blocks are sorted by age before writing them out
- Separate cold blocks (accessed infrequently) from hot blocks (accessed frequently)
- Hot blocks contain little data as they are created repeatedly
- Little benefit from copying cold blocks out of cold blocks