Distributed File Systems

- Introduction
- File Service Architecture
- Sun Network File System
- The Andrew File System
- Recent advances
- Summary

Introduction

- File system
  - Sharing of information
  - persistent storage
  - E.g. Web Servers
- Distributed file system
  - Access the information from remote sites.
  - similar (in some case better) performance and reliability
- <u>Various kinds of storage systems</u>

- Consistency: maintenance of consistency between multiple copies of data when updates occurs.
- Caching:
  - Web caching: on client side and on proxy side.
  - Maintained by explicit actions.

Characteristics of file systems

- **Responsibilities**
  - Organization, storage, retrieval, naming, sharing and protection
- **Important concepts related to file**
  - File
    - Include data and attributes
  - Directory
    - A special file that provides a mapping from text names to internal file identifiers
  - Metadata
    - Extra management information; including attribute, directory etc.
- **File system modules**
- **File system operations**
  - Applications access via library procedures

Distributed file system requirements

- **Transparency**
  - access transparency
  - location transparency
  - mobility transparency
  - performance transparency
  - scaling transparency
- **Concurrent file updates**
  - concurrency control
- **File replication**
  - better performance & fault tolerance
- **Hardware and operating system heterogeneity**

Distributed file system requirements … *continuted*

- Fault tolerance
  - *idempotent* operations: support at-least-once semantics
  - stateless server: restart from crash without recovery
- Consistency
  - One-copy update semantics
- Security
  - Authenticate, access control, secure channel
- Efficiency
  - comparable with, or better than local file systems in performance and reliability

Case studies

- SUN NFS
  - First file service that was designed as a product [1984]
  - Adopted as an internet standard
  - Supported by almost platforms, e.g. Windows NT, Unix
- Andrew File System
  - Campus information sharing system in CMU [1986]
  - 800 workstations and 40 servers at CMU [1991]

Distributed File Systems

- Introduction
- File Service Architecture
- Sun Network File System
- The Andrew File System
- Recent advances
- Summary

Three components of a file service

- File service architecture
- Flat file service
  - Operate on the contents of files
  - Unique file identifier (UFID)
- Directory service
  - Provide a mapping between text names to UFIDs
- Client module
  - Support applications accessing remote file service transparently
  - E.g. iterative request to directory service, cache files

Flat file service interface

- Flat file service operations
- Comparison with Unix
  - No open and close
  - Read or write specifying a starting point
- Fault tolerance reasons for the differences
  - Repeatable operations
    - except for create, all operations are idempotent
  - Stateless servers
    - E.g. without pointer when operate on files
    - restart after crash without recovery

Access control in DFS

- Unix File System
  - User's access rights are checked against the access mode requested in the *open* call
- Stateless DFS
  - DFS's interface is opened to public
  - File server can't retain the user ID
  - Two approaches for access control
    - (1) authenticate based on capability
    - (2) attach user ID on each request
- Kerberos in AFS and NFS

Directory service interface

- Main task
  - Translate text names to UFIDs

Lookup(Dir, Name) -> FileId — throws *NotFound*	Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.
AddName(Dir, Name, File) — throws *NameDuplicate*	If *Name* is not in the directory, adds (*Name*, *File*) to the directory and updates the file's attribute record. If *Name* is already in the directory: throws an exception.
UnName(Dir, Name) — throws *NotFound*	If *Name* is in the directory: the entry containing *Name* is removed from the directory. If *Name* is not in the directory: throws an exception.
GetNames(Dir, Pattern) -> NameSeq	Returns all the text names in the directory that match the regular expression *Pattern*.

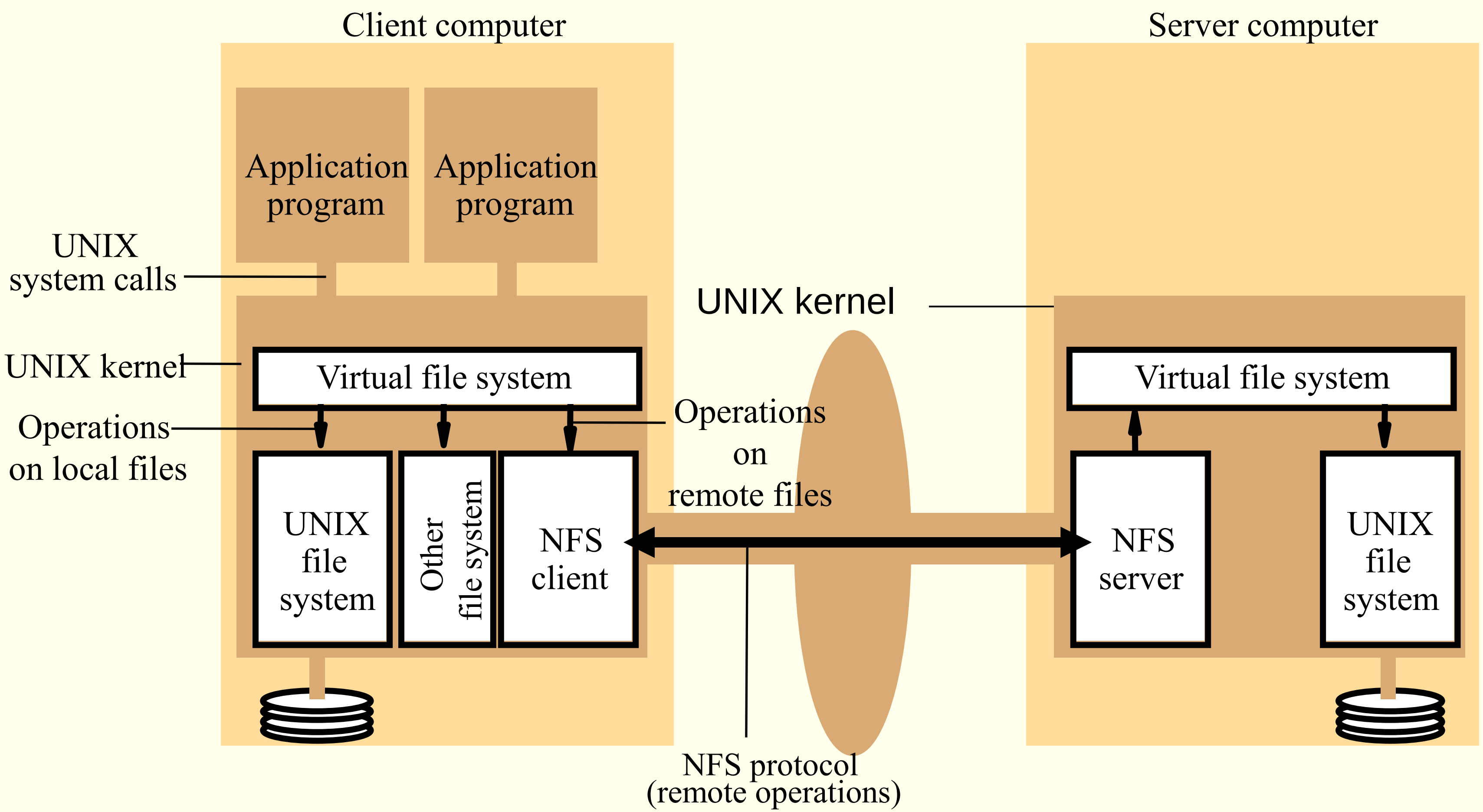
Hierarchic file system

- Directory tree
  - Each directory is a special file
    - holds the names of the files and other directories that are accessible from it
  - Pathname
    - Reference a file or a directory
    - Multi-part name, e.g. “/etc/rc.d/init.d/nfsd”
- Explore in the tree
  - Translate pathname via multiple *lookup* operations
  - Directory cache at the client

Distributed File Systems

- Introduction
- File Service Architecture
- Sun Network File System
- The Andrew File System
- Recent advances
- Summary

NFS architecture



- NFS Protocol: Remote procedure calls
  - NFS Client-Server communication using RPC
  - OS independent but originally designed for UNIX.
- NFS Server: resides in kernel on computer.
  - Port Mapper Service: Enable clients to bind to services in a given host by name.

Virtual file system

- **Keep track of local and remote filesystems.**
- **V-node: indicator to show whether the file is local or remote.**
  - For local file: refer to an i-node
  - For remote file: refer to a file handle
- i-node: no. that serves the identity & locate the file within file system.
- i-node generation no.: i-node nos. can be reused and incremented each time.
- **File handle**
  - The file identifier used in NFS
  - File handles are passed between client and server to refer to a file

fh = file handle:

Filesystem identifier	i-node number	i-node generation

Design points

- **Access control and authentication**
  - User ID is attached to each request
  - Kerberos embedded in NFS
    - Authenticate user when mount
    - *Ticket*, *Authenticator* and *secure channel*
- **Client integration into the kernel**
  - Access files using UNIX system calls without recompilation
  - Single client module serving all user-level processes
  - Retain encryption key used to authenticate user ID passed to the server
- **NFS Server interface**
  - Defined in RFC 1813

Mount service

- File server
  - */etc/exports*: contains the names of local file systems that are available for remote mounting
- Client
  - *mount* command: include location, pathname of the remote directory using mount protocol.
  - *Mount* protocol: takes directory pathname & returns the file handle of the specified directory.
- Example

Mount service … *continued*

- ***Hard-mounted/soft-mounted***
  - *Hard-mounted*: process suspends when the accessing remote directory is unavailable
  - *Soft-mounted*: indicate the error to the process after several tries
- **Automounter**
  - mount dynamically whenever an empty mount point is referenced by a client

Path name translation

- Path name translation
  - From pathname to file handle
- Multi-part pathname translation
  - Client issues several separated lookup requests to server
- Directory cache
  - Cache the results of translation conducted recently

File cache at the server

- *Buffer cache* in UNIX file system: files or directories are retained in main memory buffer cache.
  - *read-ahead: anticipates read accesses & fetches the pages following those that have most recently read.*
  - *delayed-write: optimizes write, when a page has been altered, its new contents are written to disk only when the buffer page is required for another request.*
  - Loss of data handling: *sync* periodically, e.g. 30 seconds
- Cache reading in NFS server
  - Similar to local file system
- Cache writing in NFS server: enhance reliability
  - *write-through:* data in cache at the server and written to disk before a reply is sent to client.
  - *commit* operation : data in cache at the server & written to disk when commit operation is received for the file.

File cache at the client

- **Cache file blocks at the client:**
  - In order to reduce the no. of requests transmitted to the server.
- **Maintain coherence**
  - Client polls server to validate the blocks when using the blocks
- **Validity condition:** synchronization of file contents (*one-copy semantics*) is not guaranteed when two or more clients are sharing the same file.
  - Two timestamp attached to each block in the cache
    - T: the time when request (Current time)
    - T_c: the time when the cache entry was last validated
    - Tm : the time when the block was last modified at the server
  - Valid: $(T\text{-}T_c<t)\vee(Tm_{client}=Tm_{server})$
  - t freshness guarantee which is configurable for each file or directory
    - E.g. 3 sec for file and 30sec for directory

File cache at the client ... *continued*

- Write cache
  - flush result to server when file is closed
  - conduct *sync* periodically
  - Bio-Daemon Process
- Cache semantics
  - Not guarantee the Unix file consistency

NFS Summary

- Access transparency:
  - No modifications to existing programs are required to access remote files.
- location transparency:
  - Uniform name space can be established with appropriate configuration tables in each client.
- Mobility transparency
  - Must remount a filesystem when it migrates.
- Scalability
  - limited performance for hot-spot files (single file that is accessed so frequently that the server reaches a performance limit.
- File replication
  - not support file replication with updates
  - SUN NIS(n/w information service) can be used with NFS.

NFS Summary

- Hardware and operating system heterogeneity
- Fault tolerance
  - stateless server, idempotent operations
- Consistency
- Security
  - be enhanced by kerberos
- Efficiency

Distributed File Systems

- Introduction
- File Service Architecture
- Sun Network File System
- The Andrew File System
- Recent advances
- Summary

Motivation of AFS

- Information sharing
  - Share information among large number of users
- Scalability
  - Large number of users
  - Large amount of files
  - Large number users accessing the "hot spot" files
- Unusual designs
  - *Whole file serving: entire contents of files and directories are transmitted to client computers by AFS Server.*
  - *Whole file caching: Copy of file or chunk has been transmitted to a client computer which will store in Cache.*
  - *Write on Close: Writes are propagated to the server side copy only when client closes the local copy of file.*

Typical scenario of using AFS

- Client *open* an remote file
- Store the file copy in the client computer
- Client *read(write)* on the local copy
- When client *close* the file
  - If the file was updated, flush it to the server

Observation of typical Unix file system

- Files are small
  - Most files are less than 10k in size
- Frequency of Read are more than write
- Sequential access is common, random access is rare
- Most files are accessed by only one user
- Most shared files are modified by one user
- Recently used file is highly probable be used again

Difference between NFS and AFS

- Stateful servers in AFS.
  - Server keeps track of which files are opened by which clients.
- AFS provides location independence and transparency.
  - Location independence: physical storage location of the file can be changed without having to change the path.
  - Location Transparency: file name does not hint its physical storage location.
- Callback and Callback Promise.
  - Callback: Servers will inform to all clients about updates
  - Callback promise: issued by the server to a client when it requests for a copy of file.

Implementation

- System architecture
- Unix kernel – modified BSD
  - Intercept / forward relevant system calls to *Venus*

Implementation … *continued*

- **Venus:**
  - Client side cache manager which acts as an interface between application program and vice.
  - Access files by *fids*
  - step-by-step lookup
    - Translate pathname to *fids*
  - File cache
    - One file partition for cache: Accommodate several hundred average-size files
    - Maintain cache coherence: call-back mechanism
- **Vice:**
  - Server side process that resides on top of the UNIX kernel, providing shared file services to each client.
  - Set of files in one server is referred as a *Volume*.
  - Flat file service
  - Accept requests in terms of *fids*

Cache coherence

- State of a cached file
  - valid/cancelled
  - Token checking: if value is cancelled, then a fresh copy of files must be fetched. Or if value is valid, then the cached copy can opened & used without reference to Vice.
- Open a file
  - Venus fetch the fresh copy of file when the file is cancelled
  - Vice remember each cached file's location
- Close a file
  - Venus flushes the file when application updates the file
  - Vice executes the updates on the file in a sequential order
  - Vice informs all caches of the file to be cancelled

Cache coherence …*continued*

- Validate a file
  - Venus validates the file when client restart or not receive a callback for a time *T*
- Scalability
  - Client-server interaction is reduced dramatically
    - Due to most files are read-only, callback reduces client-server interaction dramatically in contrast to client polling

Update semantics

- **Approximation to one-copy file semantics**
- **AFS-1**
  - After a successful open: *latest(F,S)*
  - After a failed open: *failure(S)*
  - After a successful close: *updated(F,S)*
  - After a failed close: *failure(S)*
- **AFS-2: weaker open semantics**
  - After a successful open
    - *latest(F, S, 0) or (lostCallback(S,T) and inCache(F) and latest(F, S, T))*
- **No concurrent control at server**

Distributed File Systems

- Introduction
- File Service Architecture
- Sun Network File System
- The Andrew File System
- Recent advances
- Summary

NFS enhancements

- Sprite: one-copy update semantics
  - Multiple readers operate on cached copies
  - One writer, multiple readers operate on the same server copy
- NONFS: more precise consistency
  - Maintain cache consistency by *lease*
- WebNFS: access NFS server via Web
- NFS version 4: wide-area networks application

Improvements in storage organization

- Redundant Arrays of Inexpensive Disks (*RAID*)
  - Segmented into fixed-size chunks
  - Stored in stripes across several disks
  - Redundant error-correcting codes
- Log-structured file storage (*LFS*)
  - Accumulate a set of tiny writes in memory
  - Commit the accumulated wirtes in large, continuous, fixed–sized segments

New design approaches

- **xFS (Serverless File System)**
  - Separate file server management task from processing task
  - Distribute file server processing responsibility
  - Software RAID
  - Cooperative cache
- **Frangipani**
  - Separate persistent storage responsibility from other service actions
  - Petal: virtual disk abstraction across many disks located on multiple servers
  - Log-structured data store

Distributed File Systems

- Introduction
- File Service Architecture
- Sun Network File System
- The Andrew File System
- Recent advances
- Summary

Summary

- **Key design issues for DFS**
  - Effective use of client cache
  - Cache coherence
  - Recovery after client or server failure
  - High throughput for reading and writing
  - Scalability
- **NFS**
  - Stateless, efficient, poor scalability
- **AFS**
  - high scalability
- **DFS state-of-the-art**
  - Support application in wide area network and pervasive computing

Storage systems and their properties

	Sharing	*Persistence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	✗	✗	✗	1	RAM
File system	✗	✓	✗	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	✗	Web server
Distributed shared memory	✓	✗	✓	✓	Ivy (DSM, Ch. 6)
Remote objects (RMI/ORB)	✓	✗	✗	1	CORBA
Persistent object store	✓	✓	✗	1	CORBA Persistent State Service
Peer-to-peer storage system	✓	✓	✓	2	OceanStore (Ch. 10)

File system modules

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

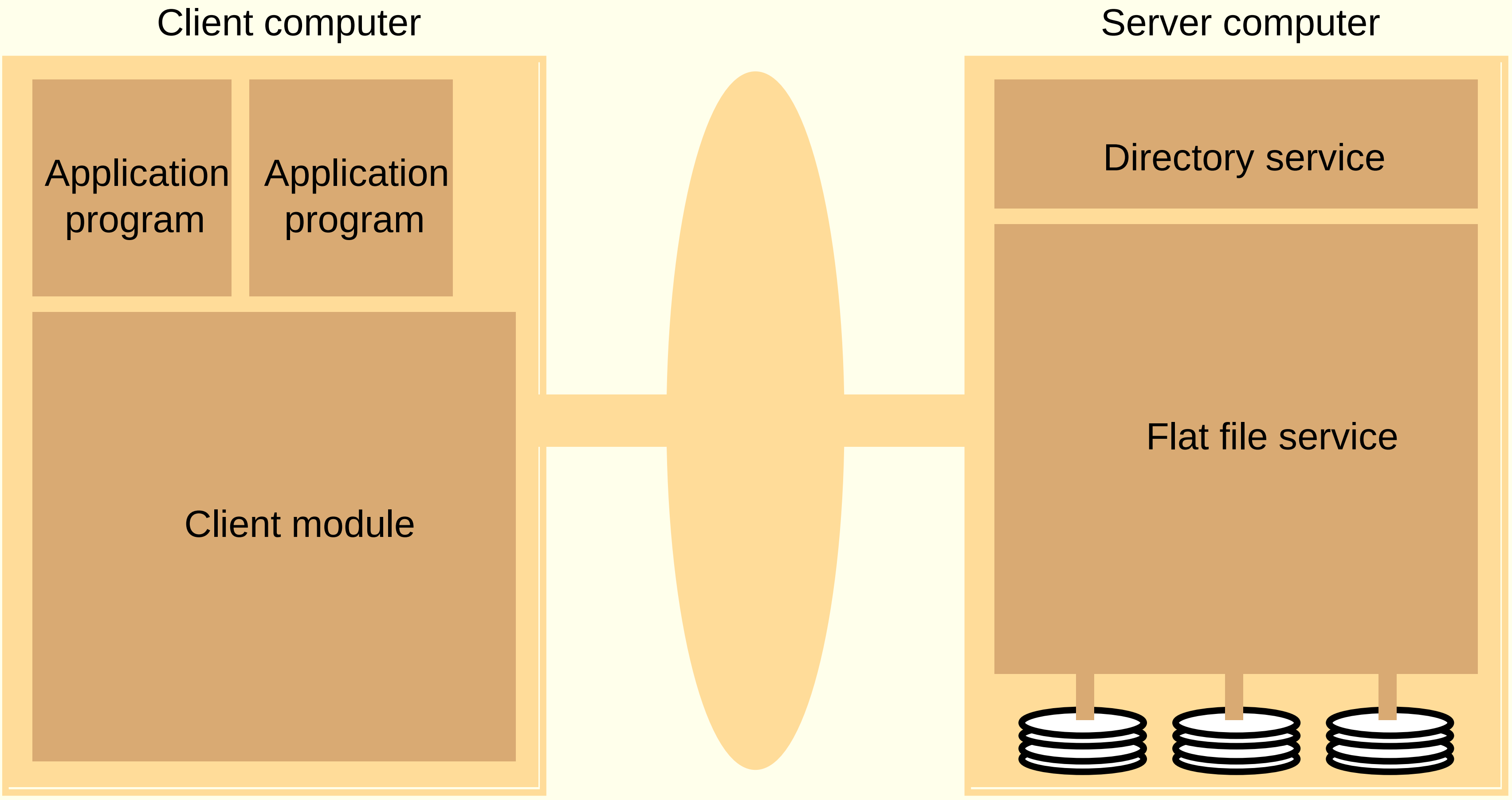
File attributed record structure

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list

Unix file system operations

filedes = open(name, mode) *filedes = creat(name, mode)*	Opens an existing file with the given *name*. Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The *mode* is *read, write* or both.
status = close(filedes)	Closes the open file *filedes*.
count = read(filedes, buffer, n) *count = write(filedes, buffer, n)*	Transfers *n* bytes from the file referenced by *filedes* to *buffer*. Transfers *n* bytes to the file referenced by *filedes* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
pos = lseek(filedes, offset, whence)	Moves the read-write pointer to offset (relative or absolute, depending on *whence*).
status = unlink(name)	Removes the file *name* from the directory structure. If the file has no other names, it is deleted.
status = link(name1, name2)	Adds a new name (*name2*) for a file (*name1*).
status = stat(name, buffer)	Gets the file attributes for file *name* into *buffer*.

File service architecture



Flat file service operations

Read(FileId, i, n) -> Data — throws *BadPosition*	If $1 \le i \le Length(File)$: Reads a sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If $1 \le i \le Length(File)+1$: Writes a sequence of *Data* to a file, starting at item *i*, extending the file if necessary.
Create() -> FileId	Creates a new file of length 0 and delivers a UFID for it.
Delete(FileId)	Removes the file from the file store.
GetAttributes(FileId) -> Attr	Returns the file attributes for the file.
SetAttributes(FileId, Attr)	Sets the file attributes (only those attributes that are not shaded in).

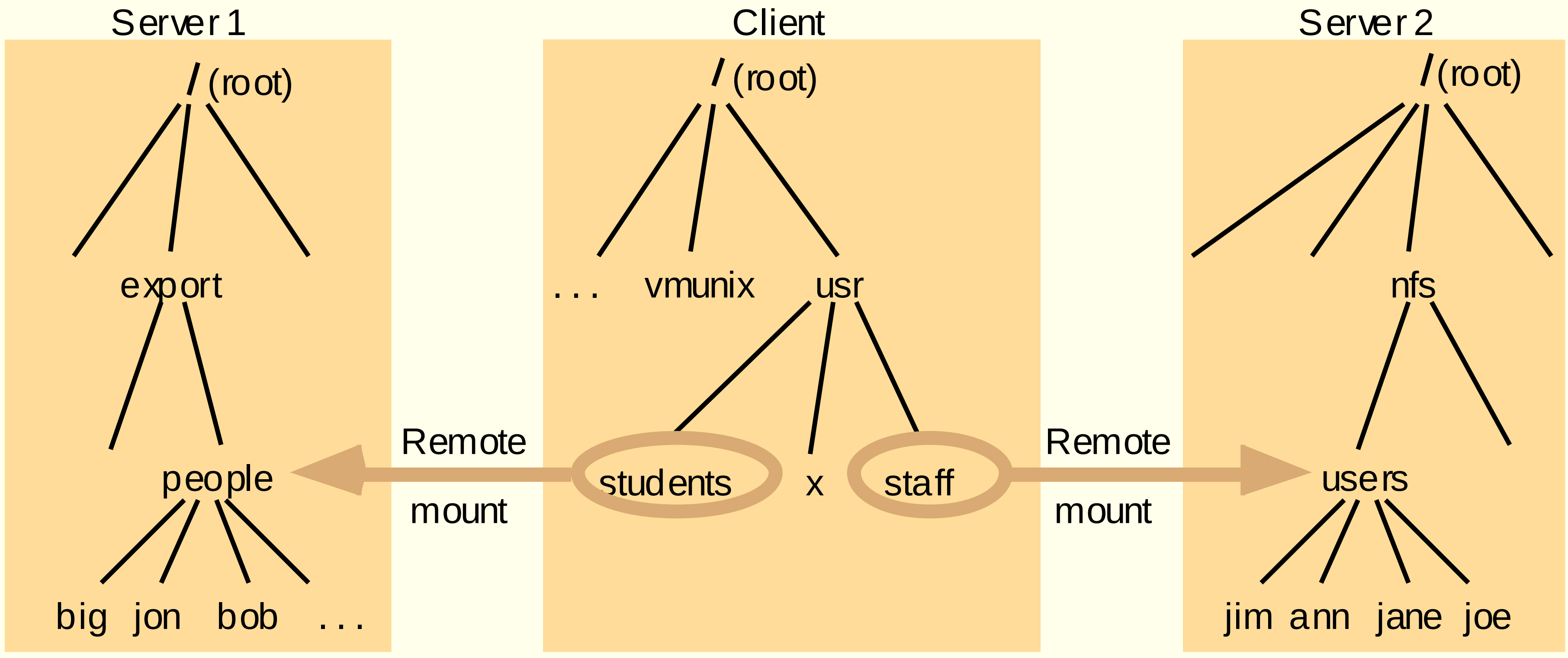
NFS server operations (simplified) - 1

lookup(dirfh, name) -> fh, attr	Returns file handle and attributes for the file *name* in the directory *dirfh*.
create(dirfh, name, attr) -> newfh, attr	Creates a new file name in directory *dirfh* with attributes *attr* and returns the new file handle and attributes.
remove(dirfh, name) status	Removes file name from directory *dirfh*.
getattr(fh) -> attr	Returns file attributes of file *fh*. (Similar to the UNIX *stat* system call.)
setattr(fh, attr) -> attr	Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file.
read(fh, offset, count) -> attr, data	Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file.
write(fh, offset, count, data) -> attr	Writes *count* bytes of data to a file starting at *offset*. Returns the attributes of the file after the write has taken place.
rename(dirfh, name, todirfh, toname) -> status	Changes the name of file *name* in directory *dirfh* to *toname* in directory to *todirfh*
link(newdirfh, newname, dirfh, name) -> status	Creates an entry *newname* in the directory *newdirfh* which refers to file *name* in the directory *dirfh*.

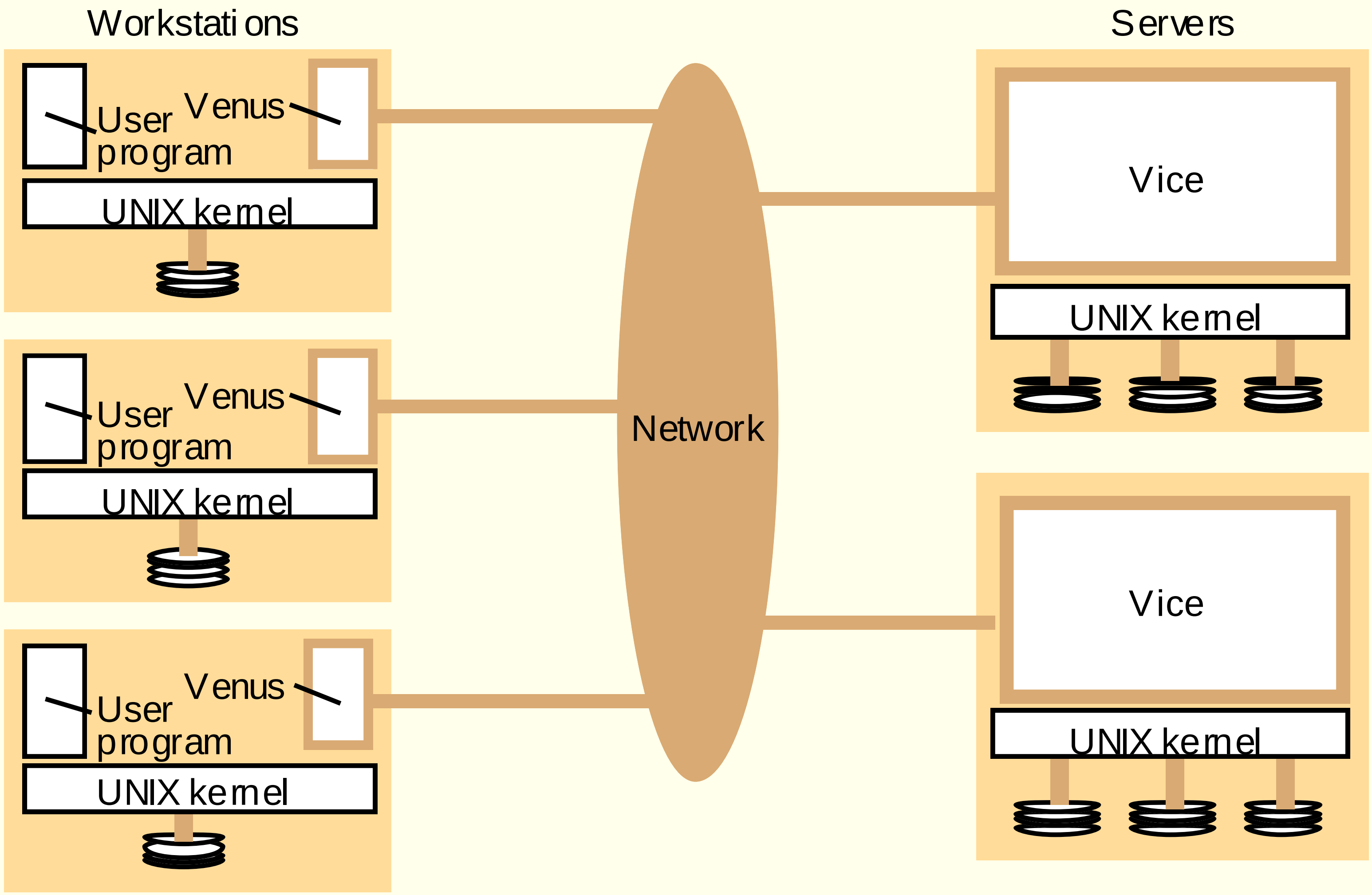
NFS server operations (simplified) - 2

symlink(newdirfh, newname, string) -> status	Creates an entry *newname* in the directory *newdirfh* of type symbolic link with the value *string*. The server does not interpret the *string* but makes a symbolic link file to hold it.
readlink(fh) -> string	Returns the string that is associated with the symbolic link file identified by *fh*.
mkdir(dirfh, name, attr) -> newfh, attr	Creates a new directory *name* with attributes *attr* and returns the new file handle and attributes.
rmdir(dirfh, name) -> status	Removes the empty directory *name* from the parent directory *dirfh*. Fails if the directory is not empty.
readdir(dirfh, cookie, count) -> entries	Returns up to *count* bytes of directory entries from the directory *dirfh*. Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a *cookie*. The *cookie* is used in subsequent *readdir* calls to start reading from the following entry. If the value of *cookie* is 0, reads from the first entry in the directory.
statfs(fh) -> fsstats	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file *fh*.

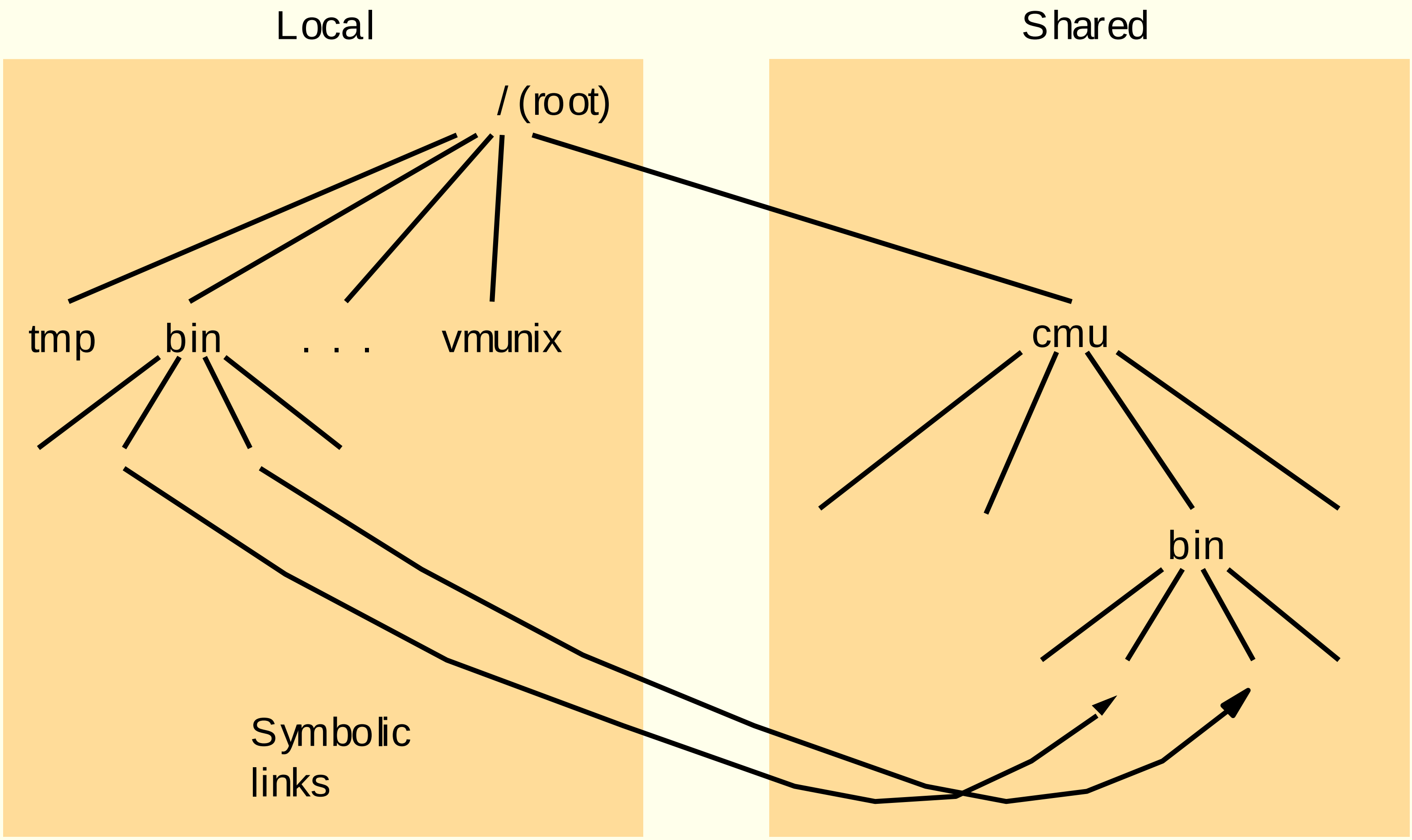
Local and remote file systems accessible on an NFS client



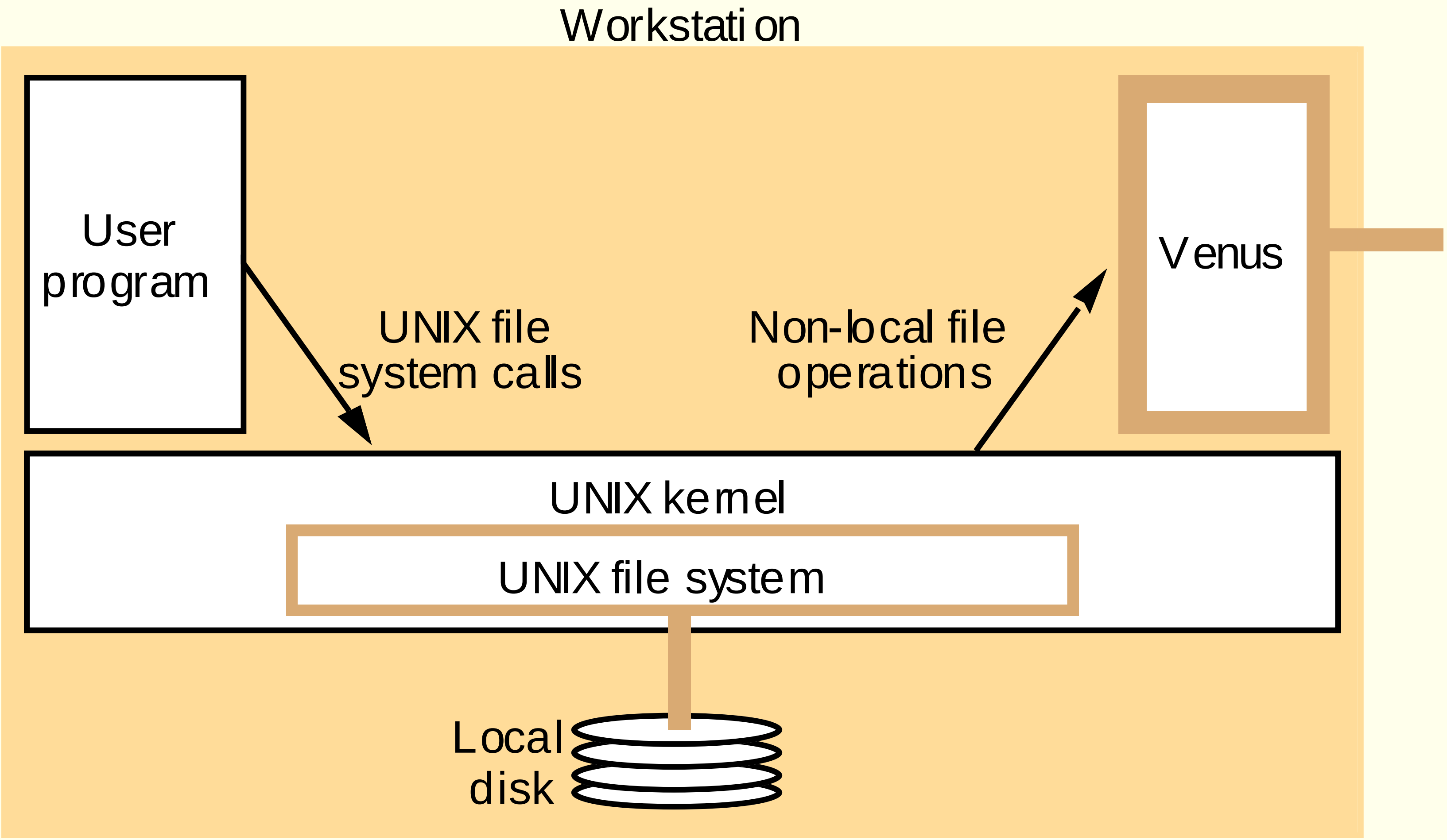
Distribution of processes in the Andrew File System



File name space seen by clients of AFS



System call interception in AFS



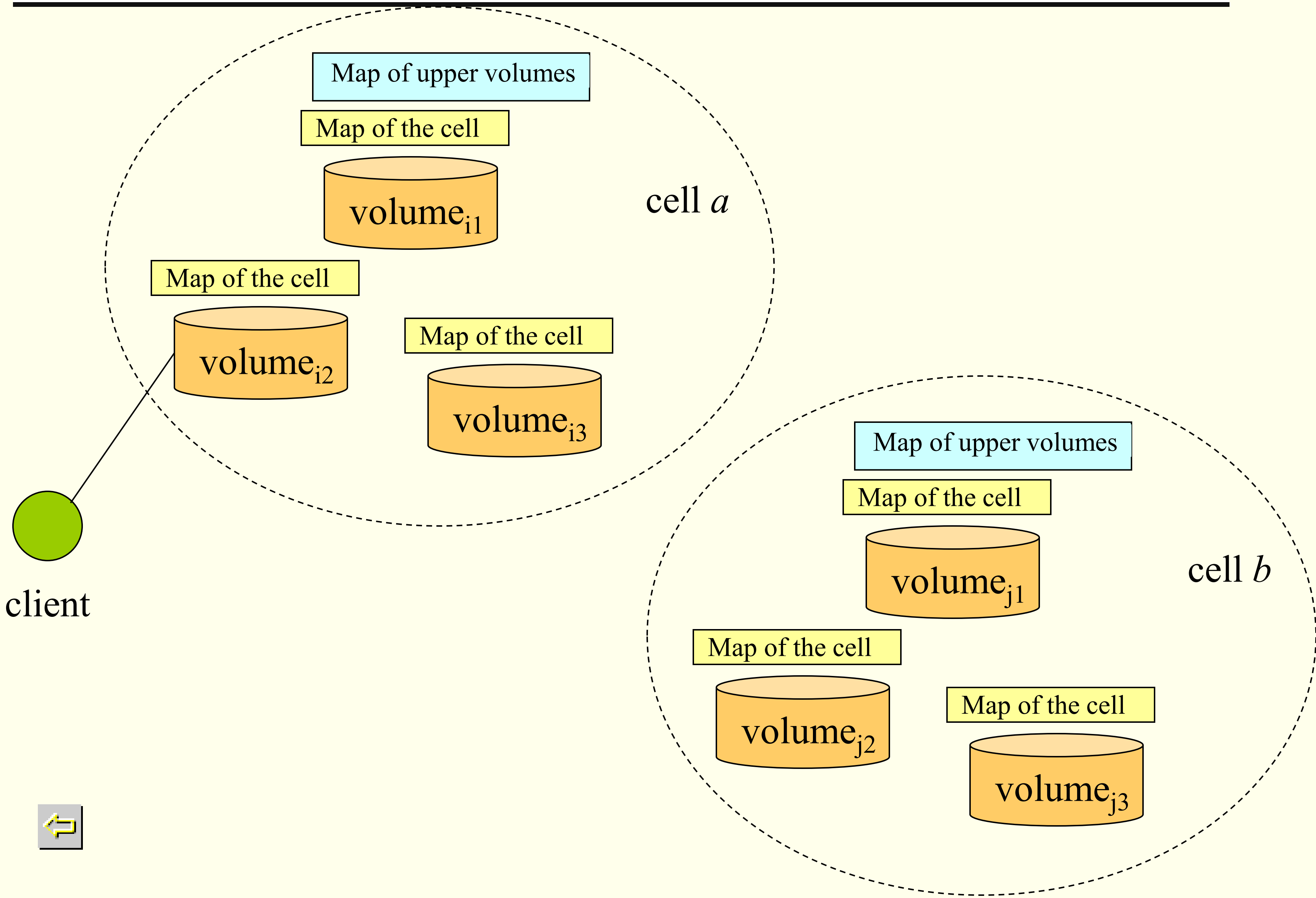
AFS *fid*

32 bits	32 bits	32 bits
Volume number	File handle	Uniquifier

The main components of the Vice service 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.

Location database in AFS



Xfs architecture