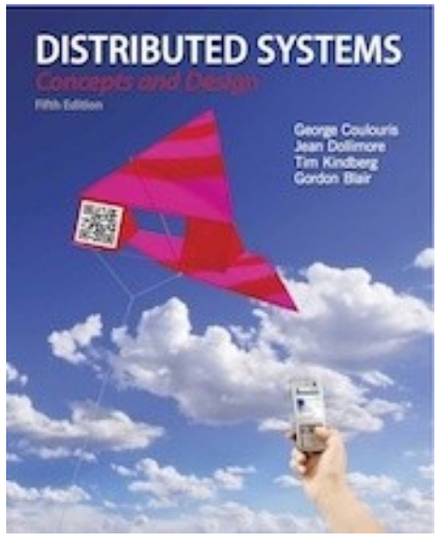
Slides for Chapter 12: Distributed File Systems



From **Coulouris, Dollimore, Kindberg and Blair**

Distributed Systems: Concepts and Design

Figure 12.1
Storage systems and their properties

	Sharing	*Persis-tence*	*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. 18)
Remote objects (RMI/ORB)	✓	×	×	1	CORBA
Persistent object store	✓	✓	×	1	CORBA Persistent Object Service
Peer-to-peer storage system	✓	✓	✓	2	OceanStore (Ch. 10)

Types of consistency:
1: strict one-copy. 3: slightly weaker guarantees. 2: considerably weaker guarantees.

Figure 12.2
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

Figure 12.3
File attribute record structure

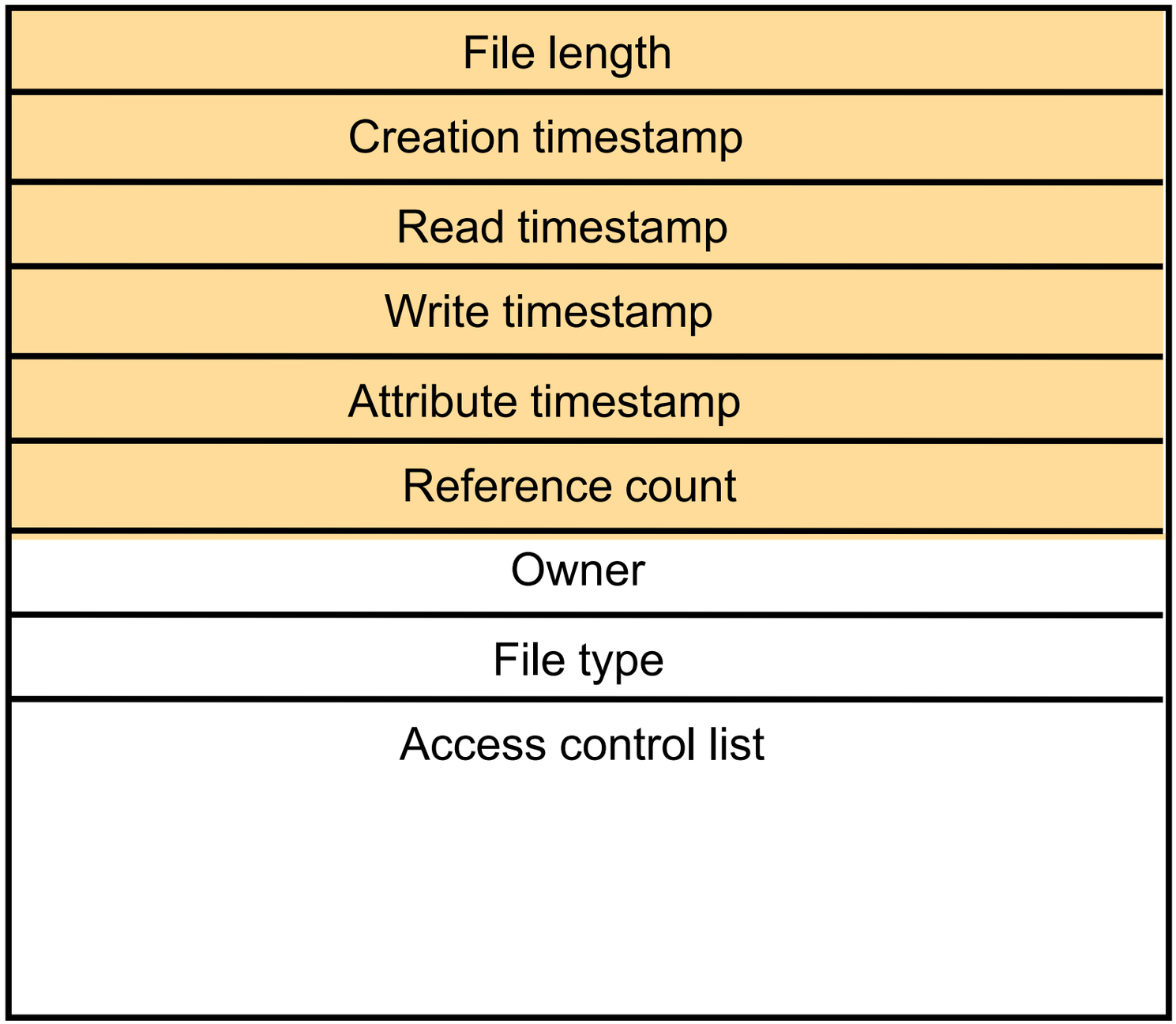


Figure 12.4
UNIX file system operations

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*.

Figure 12.5
File service architecture

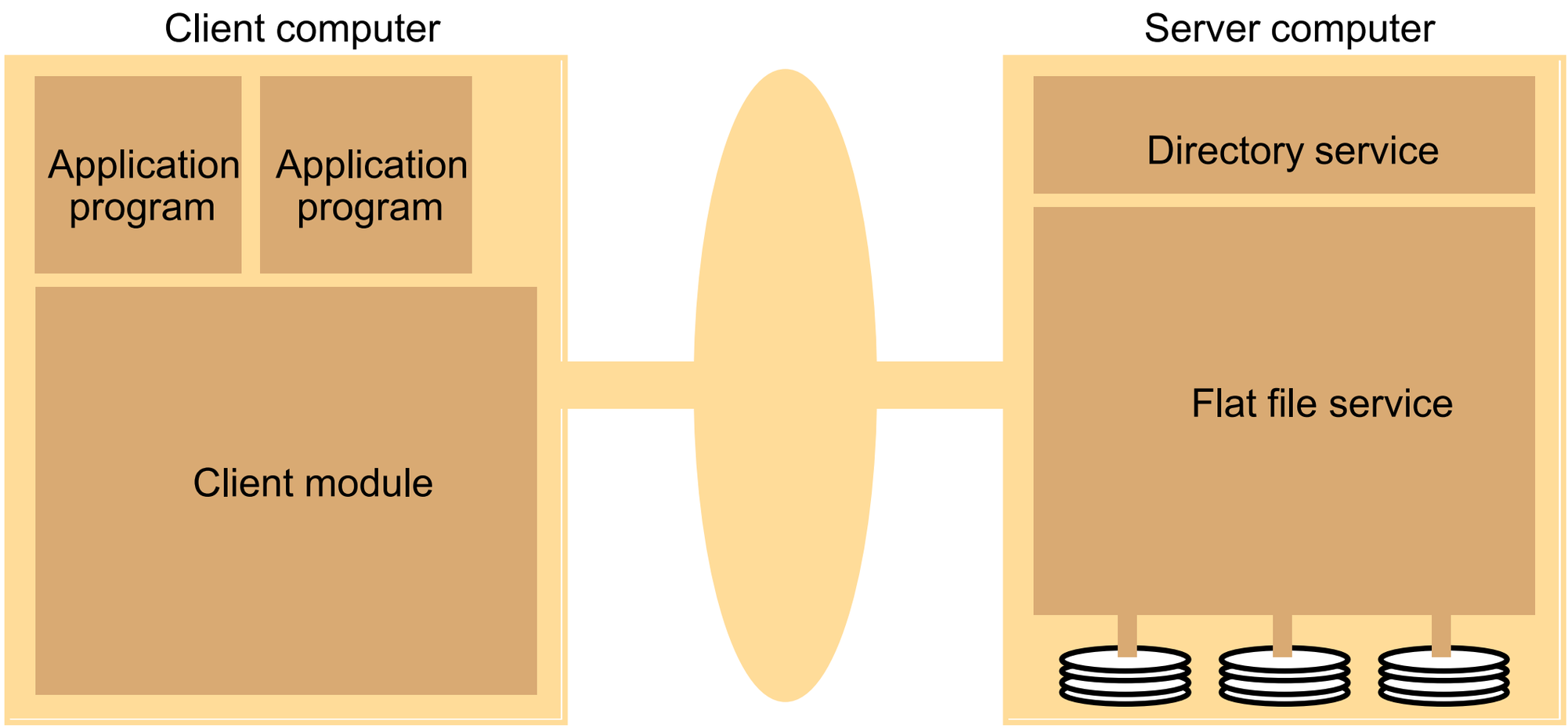


Figure 12.6
Flat file service operations

Read(FileId, i, n) -> Data — throws *BadPosition*	If $1 \leq i \leq Length(File)$: 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 $1 \leq i \leq Length(File)+1$: 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 Figure 12.3).

Figure 12.7
Directory service operations

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, FileId) — 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*.

Figure 12.8
NFS architecture

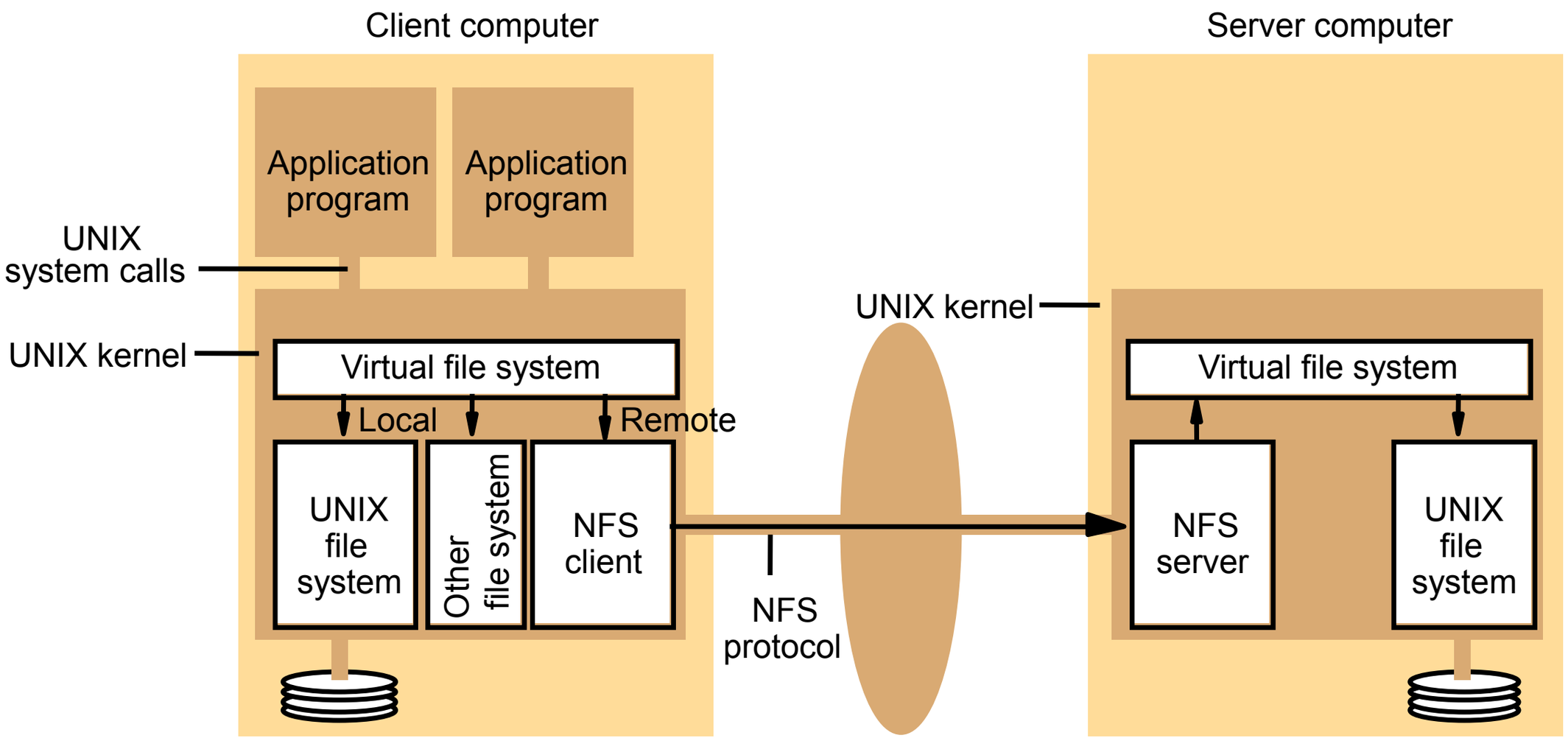


Figure 12.9
NFS server operations (simplified) – 1

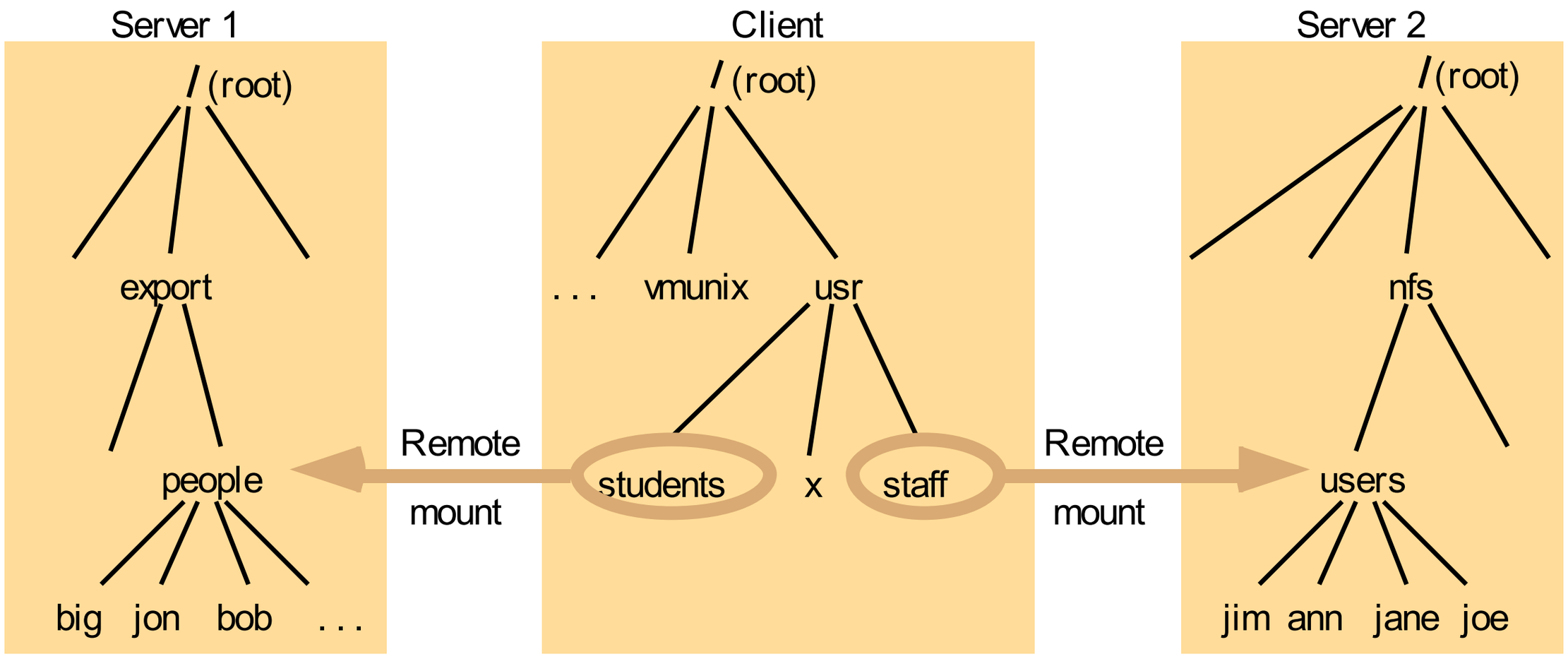
lookup(dirfh, name) -> fh, attr	Returns file handle and attributes for the file *name* in the directory *dirfh*.
create(dirfh, name, attr) -> newfh, attr	Creates a new file name in directory *dirfh* with attributes *attr* and returns the new file handle and attributes.
remove(dirfh, name) status	Removes 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*.

Continues on next slide ...

Figure 12.9
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*.

Figure 12.10
Local and remote file systems accessible on an NFS client



Note:
The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server 1; the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.

Figure 12.11
Distribution of processes in the Andrew File System

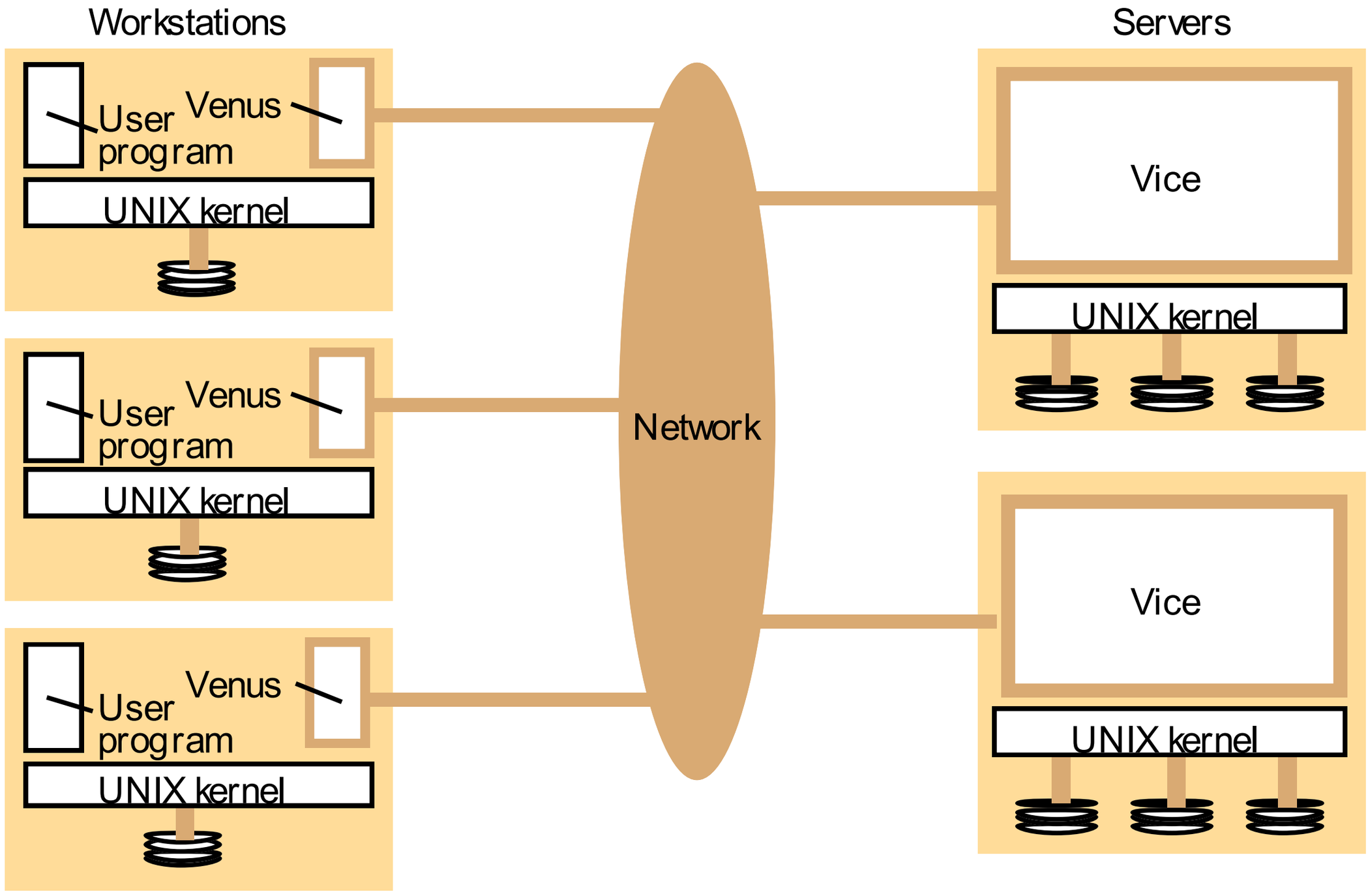


Figure 12.12
File name space seen by clients of AFS

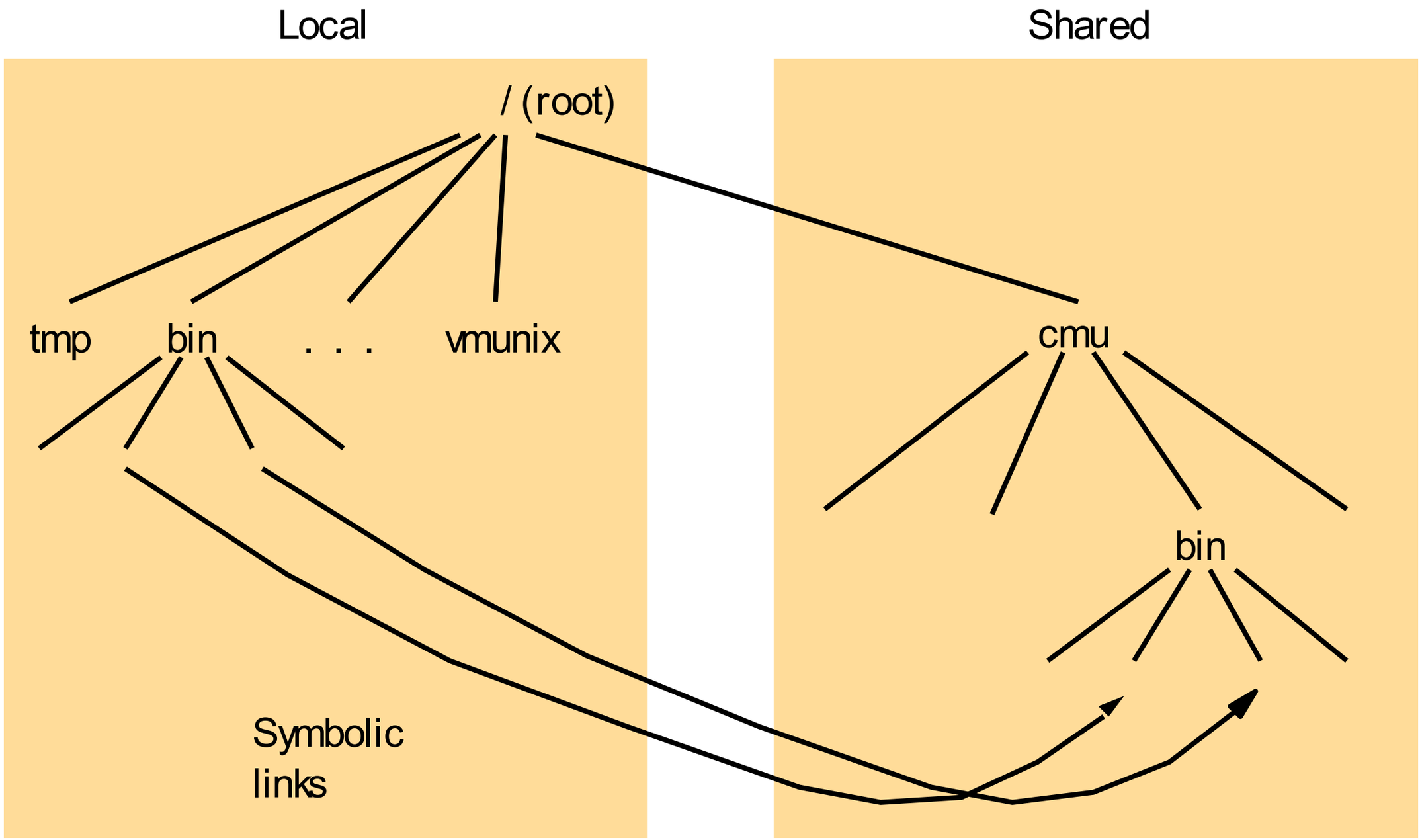


Figure 12.13
System call interception in AFS

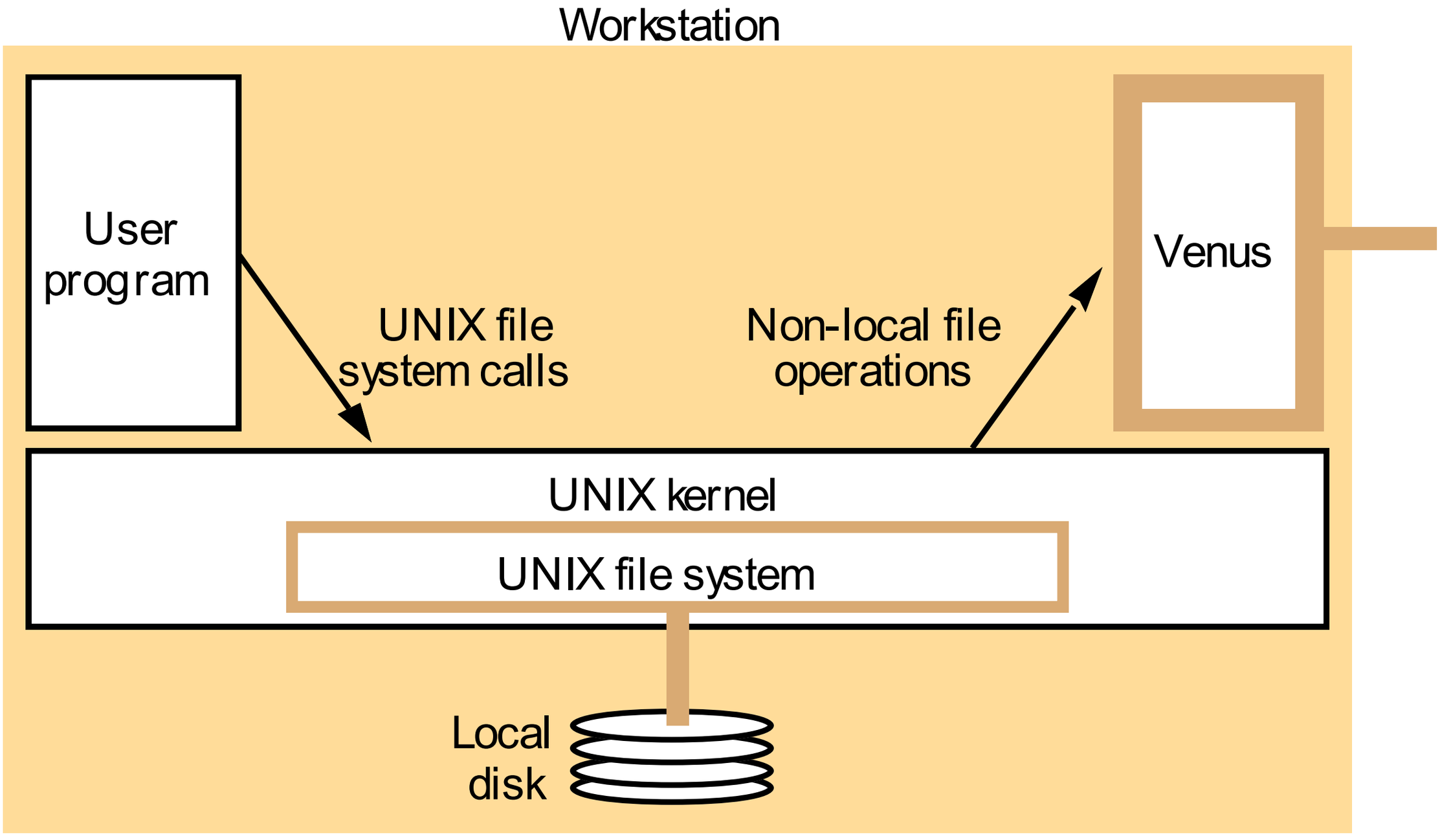


Figure 12.14
Implementation of file system calls in AFS

User process	*UNIX kernel*	*Venus*	*Net*	*Vice*
open(FileName, mode)	If *FileName* refers to a file in shared file space, pass the request to Venus. Open the local file and return the file descriptor to the application.	Check list of files in local cache. If not present or there is no valid *callback promise*, send a request for the file to the Vice server that is custodian of the volume containing the file. Place the copy of the file in the local file system, enter its local name in the local cache list and return the local name to UNIX.	→ ←	Transfer a copy of the file and a *callback promise* to the workstation. Log the callback promise.
read(FileDescriptor, Buffer, length)	Perform a normal UNIX read operation on the local copy.			
write(FileDescriptor, Buffer, length)	Perform a normal UNIX write operation on the local copy.			
close(FileDescriptor)	Close the local copy and notify Venus that the file has been closed.	If the local copy has been changed, send a copy to the Vice server that is the custodian of the file.	→	Replace the file contents and send a *callback* to all other clients holding *callback promises* on the file.

Figure 12.15
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.