

# Slides for Chapter 12: Distributed File Systems

---

# Figure 12.1

## Storage systems and their properties

	<i>Sharing</i>	<i>Persis- tence</i>	<i>Distributed cache/replicas</i>	<i>Consistency maintenance</i>	<i>Example</i>
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

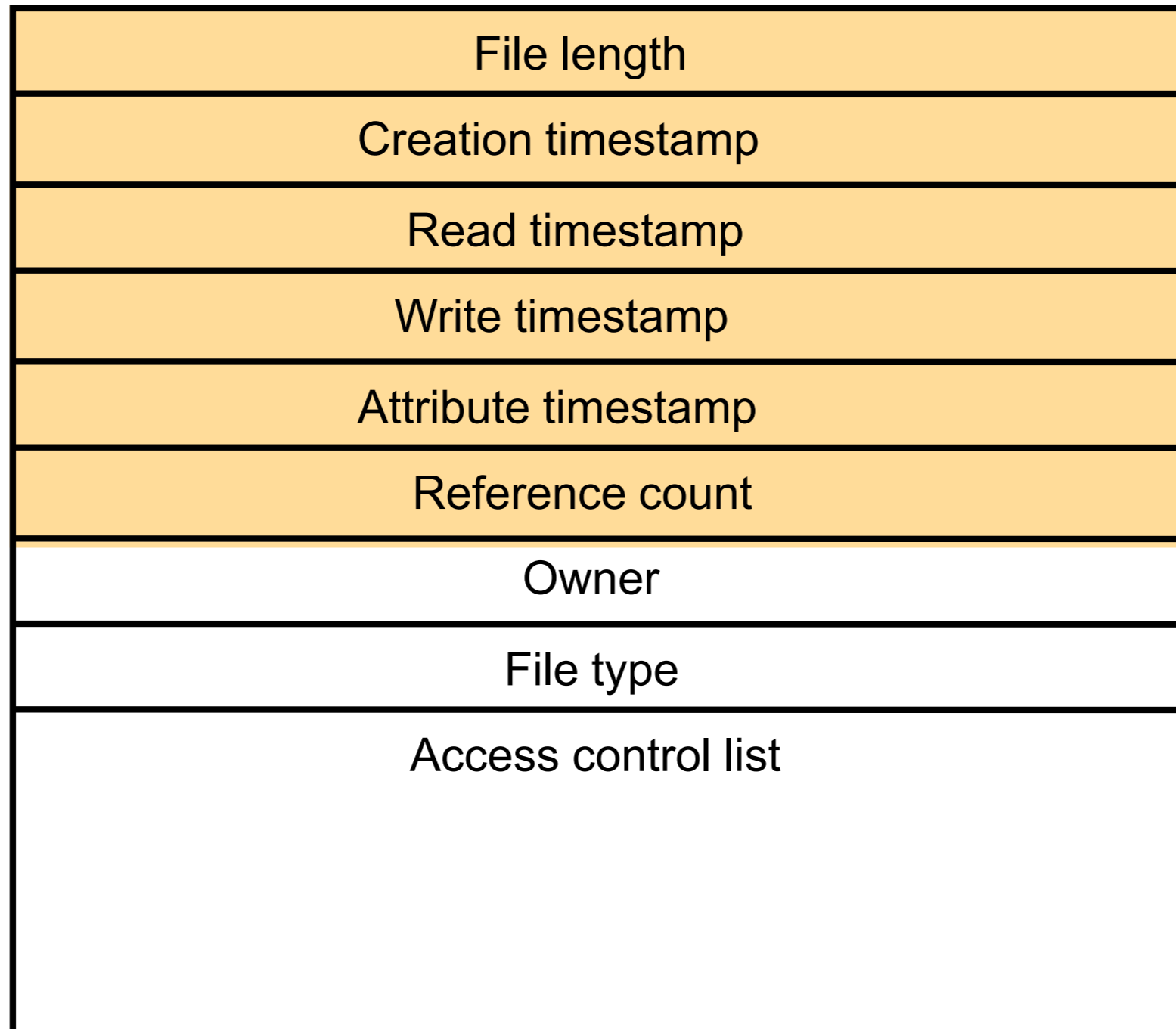
---

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

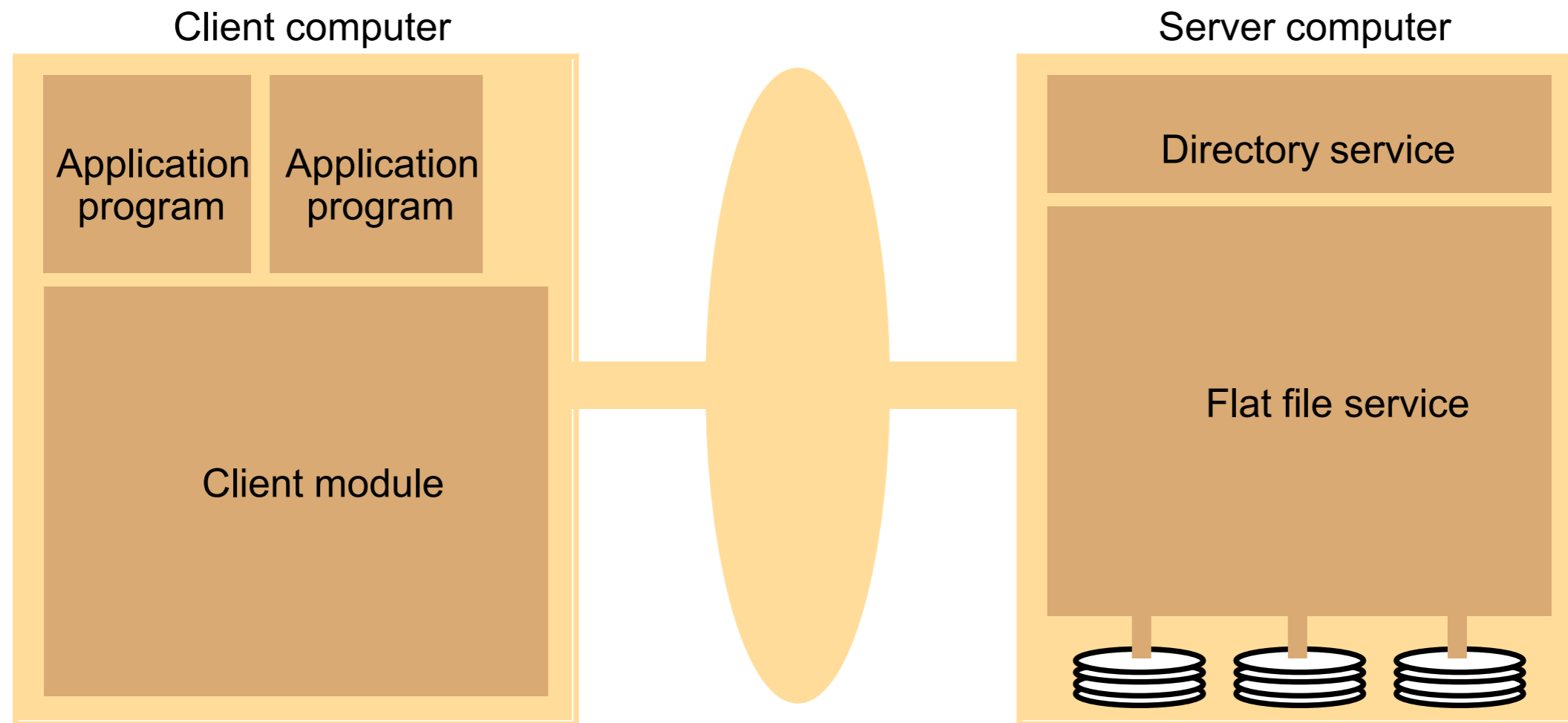
### File attribute record structure

---





## Figure 12.5 File service architecture



## Figure 12.6

### Flat file service operations

---

---

<i>Read(FileId, i, n) -&gt; Data</i> — throws <i>BadPosition</i>	If $1 \leq i \leq \text{Length}(\text{File})$ : Reads a sequence of up to $n$ items from a file starting at item $i$ and returns it in <i>Data</i> .
<i>Write(FileId, i, Data)</i> — throws <i>BadPosition</i>	If $1 \leq i \leq \text{Length}(\text{File})+1$ : Writes a sequence of <i>Data</i> to a file, starting at item $i$ , extending the file if necessary.
<i>Create() -&gt; FileId</i>	Creates a new file of length 0 and delivers a UFID for it.
<i>Delete(FileId)</i>	Removes the file from the file store.
<i>GetAttributes(FileId) -&gt; Attr</i>	Returns the file attributes for the file.
<i>SetAttributes(FileId, Attr)</i>	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.

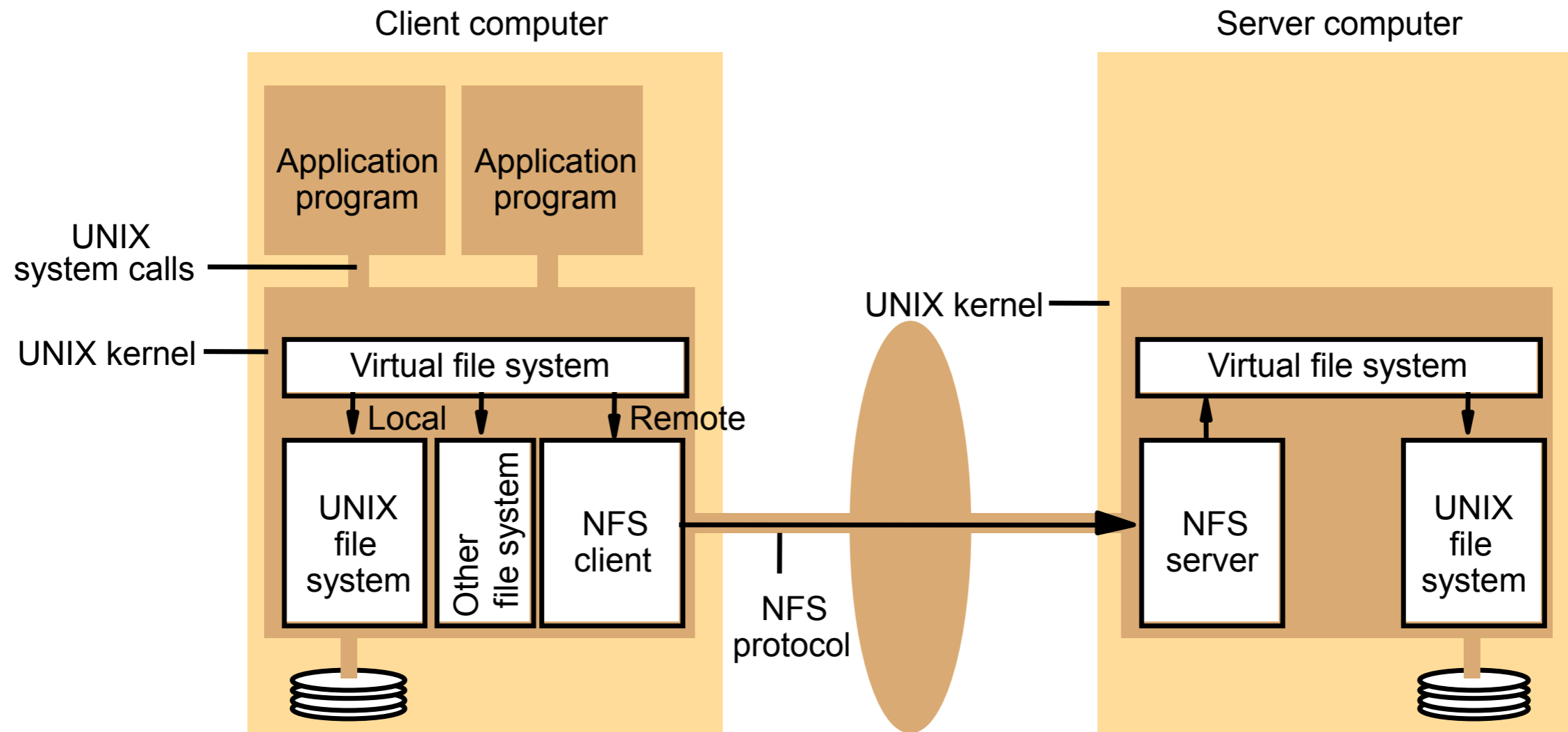
*GetNames(Dir, Pattern) -> NameSeq*

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

---

<i>lookup(dirfh, name) -&gt; fh, attr</i>	Returns file handle and attributes for the file <i>name</i> in the directory <i>dirfh</i> .
<i>create(dirfh, name, attr) -&gt; newfh, attr</i>	Creates a new file name in directory <i>dirfh</i> with attributes <i>attr</i> and returns the new file handle and attributes.
<i>remove(dirfh, name) status</i>	Removes file name from directory <i>dirfh</i> .
<i>getattr(fh) -&gt; attr</i>	Returns file attributes of file <i>fh</i> . (Similar to the UNIX <i>stat</i> system call.)
<i>setattr(fh, attr) -&gt; attr</i>	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.
<i>read(fh, offset, count) -&gt; attr, data</i>	Returns up to <i>count</i> bytes of data from a file starting at <i>offset</i> . Also returns the latest attributes of the file.
<i>write(fh, offset, count, data) -&gt; attr</i>	Writes <i>count</i> bytes of data to a file starting at <i>offset</i> . Returns the attributes of the file after the write has taken place.
<i>rename(dirfh, name, todirfh, toname) -&gt; status</i>	Changes the name of file <i>name</i> in directory <i>dirfh</i> to <i>toname</i> in directory to <i>todirfh</i>
<i>link(newdirfh, newname, dirfh, name) -&gt; status</i>	Creates an entry <i>newname</i> in the directory <i>newdirfh</i> which refers to file <i>name</i> in the directory <i>dirfh</i> .

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.

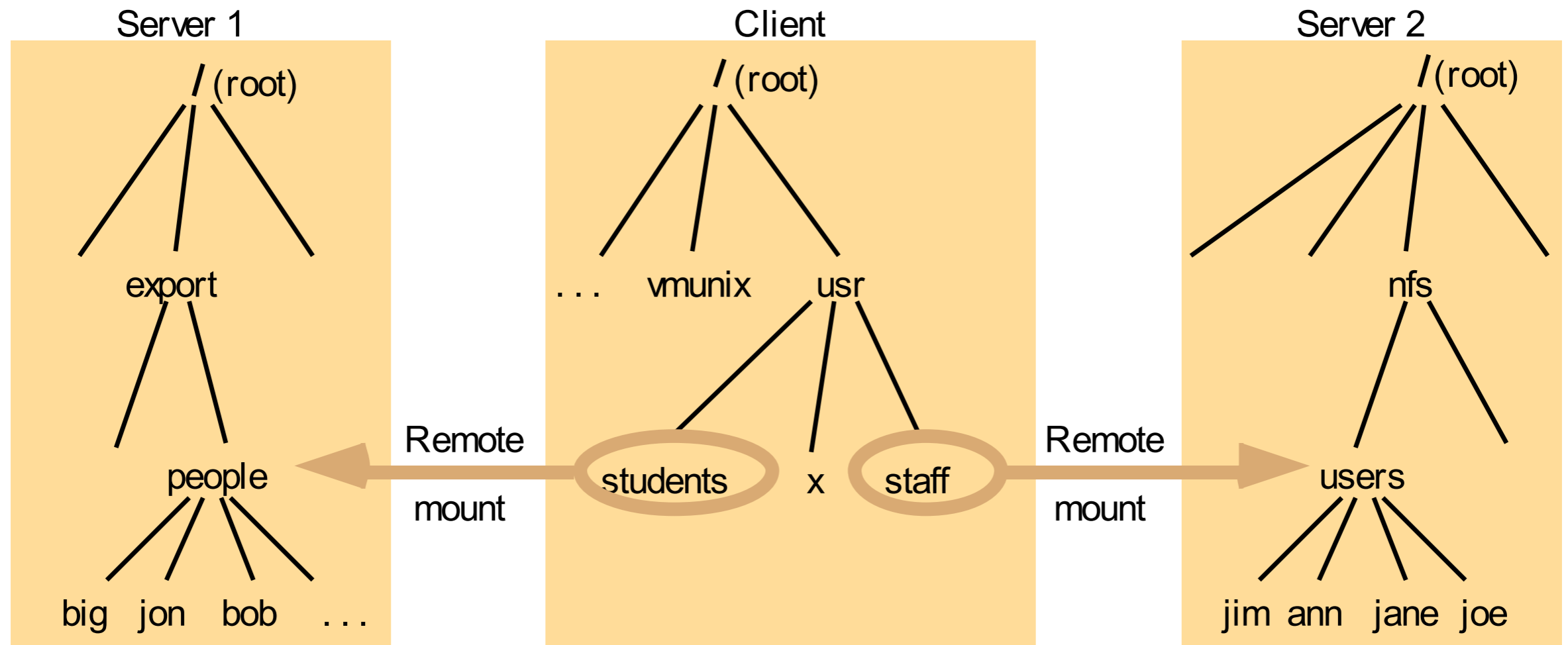
*statfs(fh)* -> *fsstats*

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

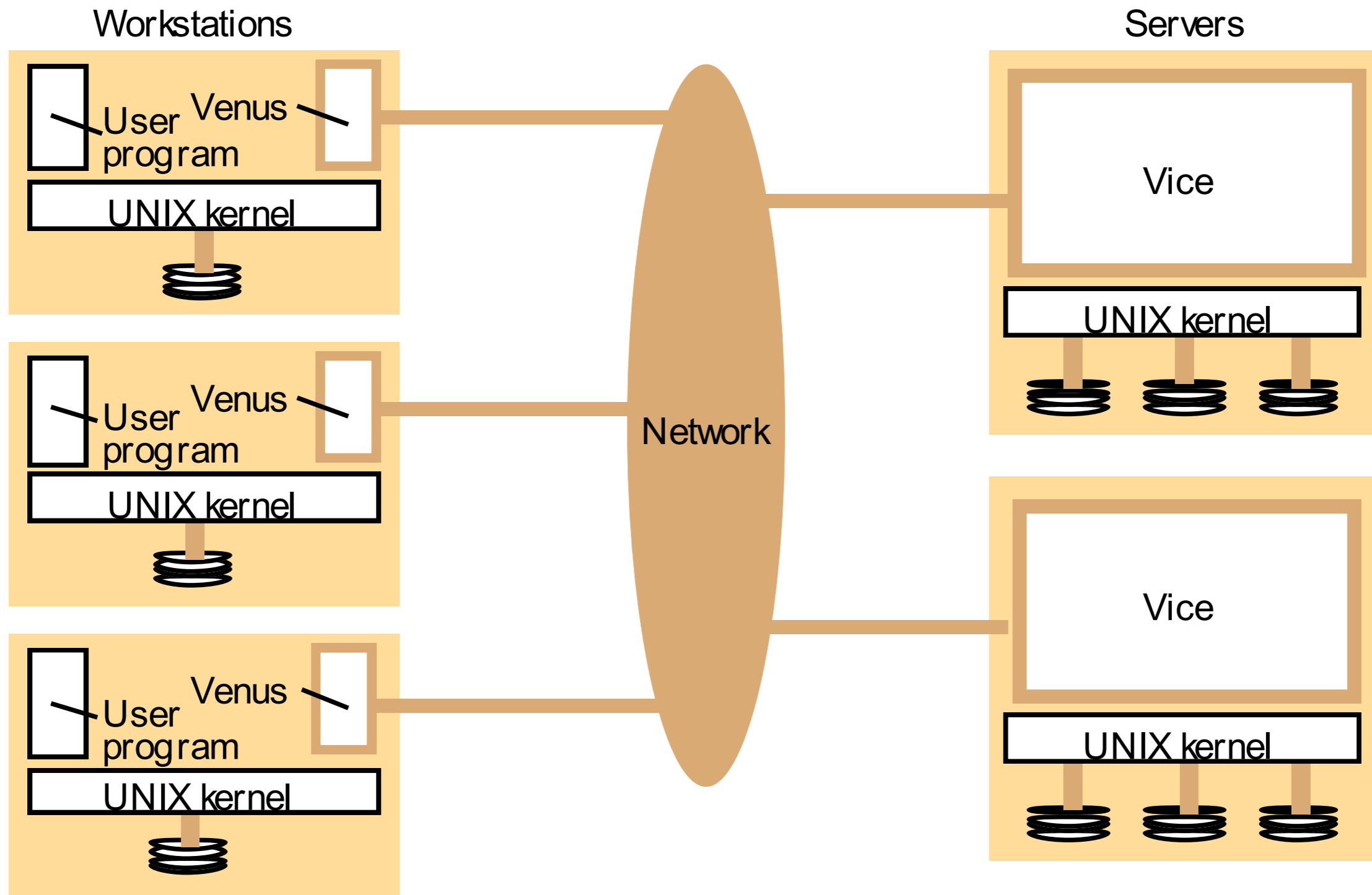


Figure 12.12  
File name space seen by clients of AFS

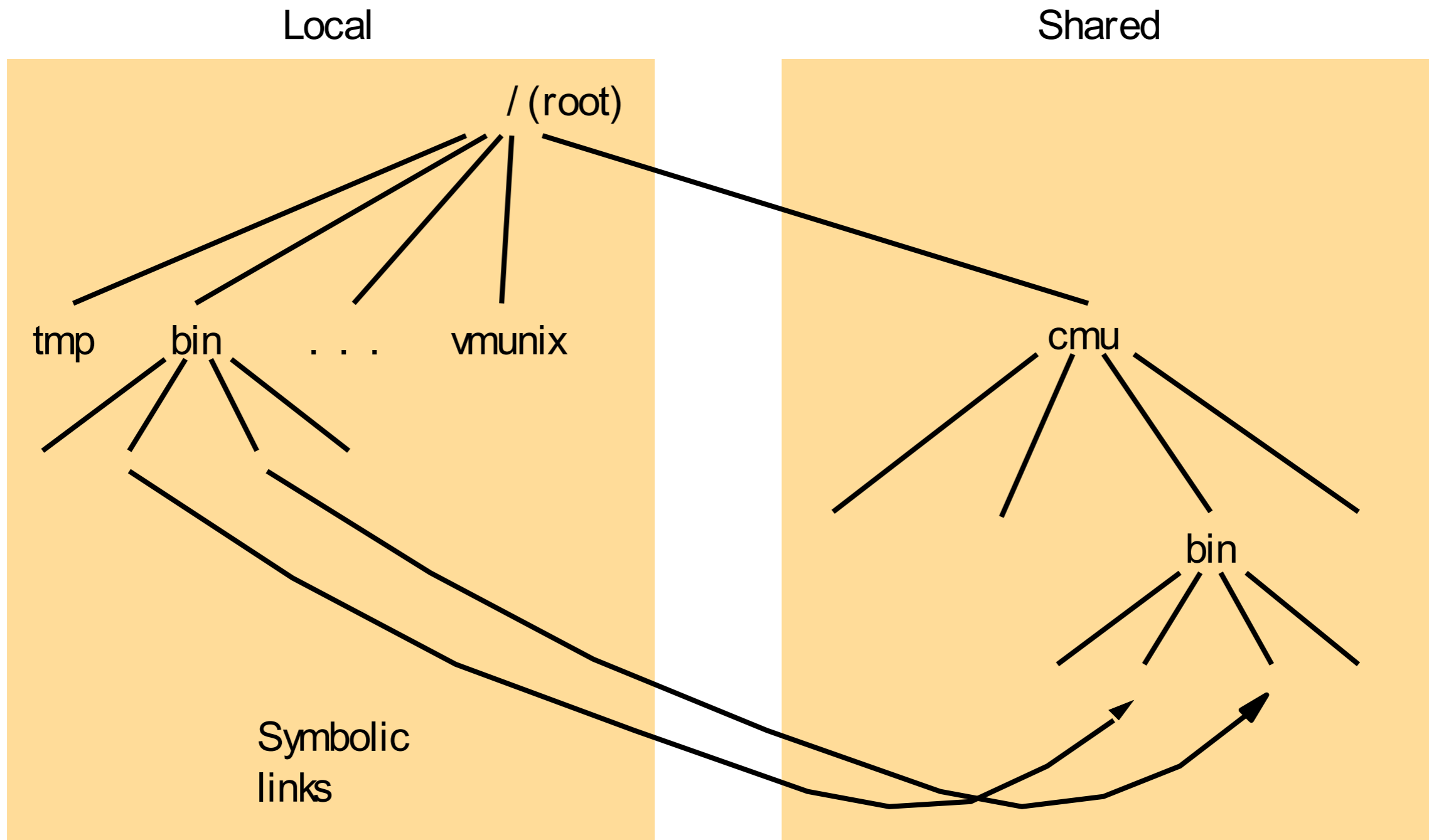
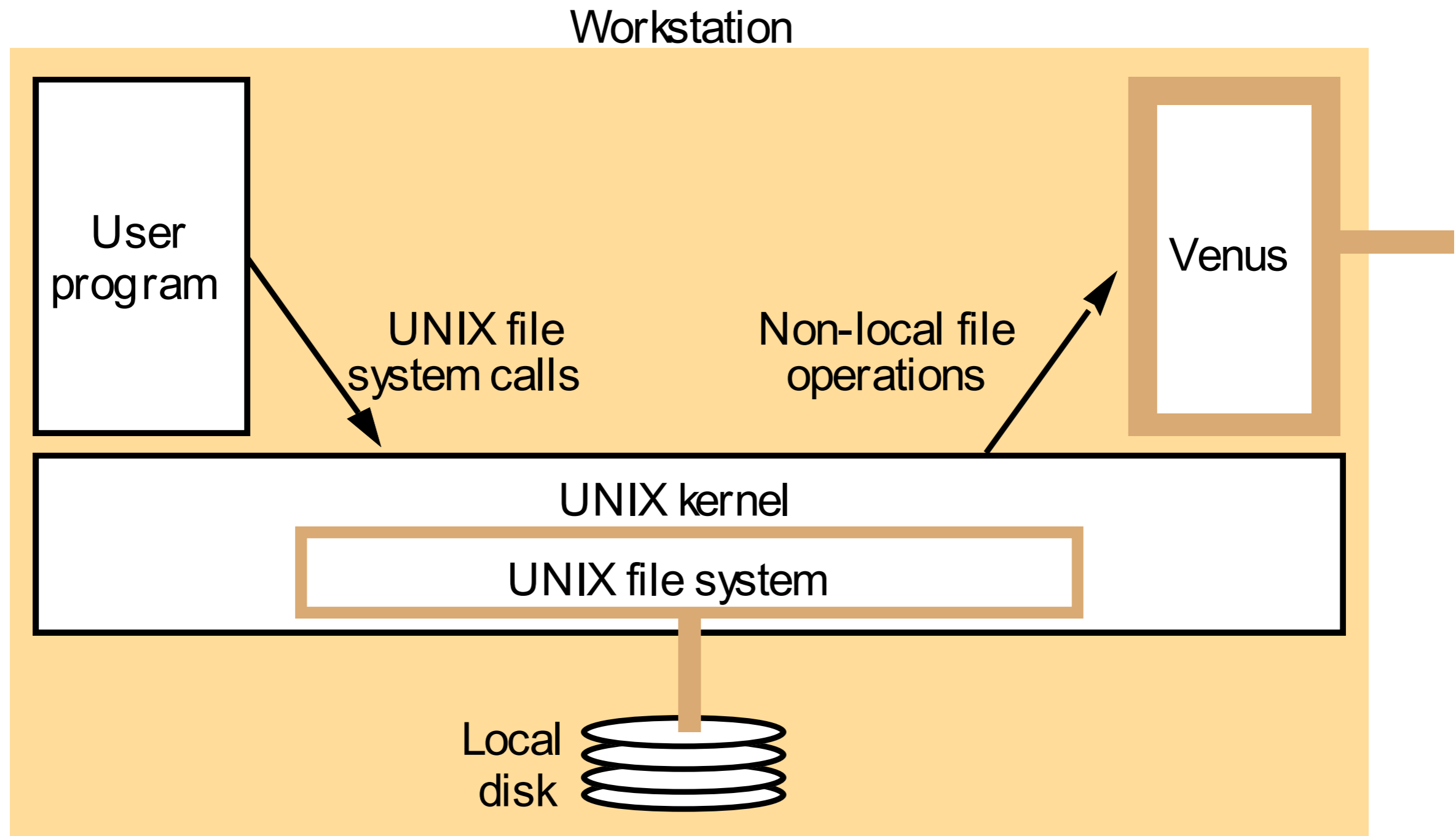


Figure 12.13  
System call interception in AFS



# Figure 12.14

## Implementation of file system calls in AFS

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



## Figure 12.15

### The main components of the Vice service interface

---

---

<i>Fetch(fid) -&gt; attr, data</i>	Returns the attributes (status) and, optionally, the contents of file identified by the <i>fid</i> and records a callback promise on it.
<i>Store(fid, attr, data)</i>	Updates the attributes and (optionally) the contents of a specified file.
<i>Create() -&gt; fid</i>	Creates a new file and records a callback promise on it.
<i>Remove(fid)</i>	Deletes the specified file.
<i>SetLock(fid, mode)</i>	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.
<i>ReleaseLock(fid)</i>	Unlocks the specified file or directory.
<i>RemoveCallback(fid)</i>	Informs server that a Venus process has flushed a file from its cache.
<i>BreakCallback(fid)</i>	This call is made by a Vice server to a Venus process. It cancels the callback promise on the relevant file.

---