DISTRIBUTED FILE SYSTEM

DILIP KUMAR SHRESTHA

INTRODUCTION

A distributed file system enables programs to store and access remote files exactly as they do local ones, allowing users to access files from any computer on a network. The performance and reliability experienced for access to files stored at a server should be comparable to that for files stored on local disks.

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)

Types of consistency:

1: strict one-copy ✓: slightly weaker guarantees 2: considerably weaker guarantees

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:	performs disk I/O and buffering

FILE ATTRIBUTE RECORD STRUCTURE

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

DISTRIBUTED FILE SYSTEM REQUIREMENTS

- Transparency
  - Access transparency
  - Location transparency
  - Mobility transparency
  - Performance transparency
  - Scaling transparency
- Concurrent file updates
- File replication
- Heterogeneity
- Fault tolerance
- Consistency
- Security
- Efficiency

FILE SERVICE ARCHITECTURE

An architecture that offers a clear separation of the main concerns in providing access to files is obtained by structuring the file service as three components – a flat file service, a directory service and a client module.

Flat file service: concerned with implementing operations on the contents of files (works on UFIDs)

Directory service: provides a mapping between text names for files and their UFIDs

Client module: runs in each client computer, integrating and extending the operations of the flat file service and the directory service

FILE OPERATIONS: UNIX

filedes = open(name, mode)	Opens an existing file with the given *name*.
filedes = creat(name, mode)	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)	Transfers *n* bytes from the file referenced by *filedes* to *buffer*.
count = write(filedes, buffer, n)	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)	Puts the file attributes for file *name* into *buffer*.

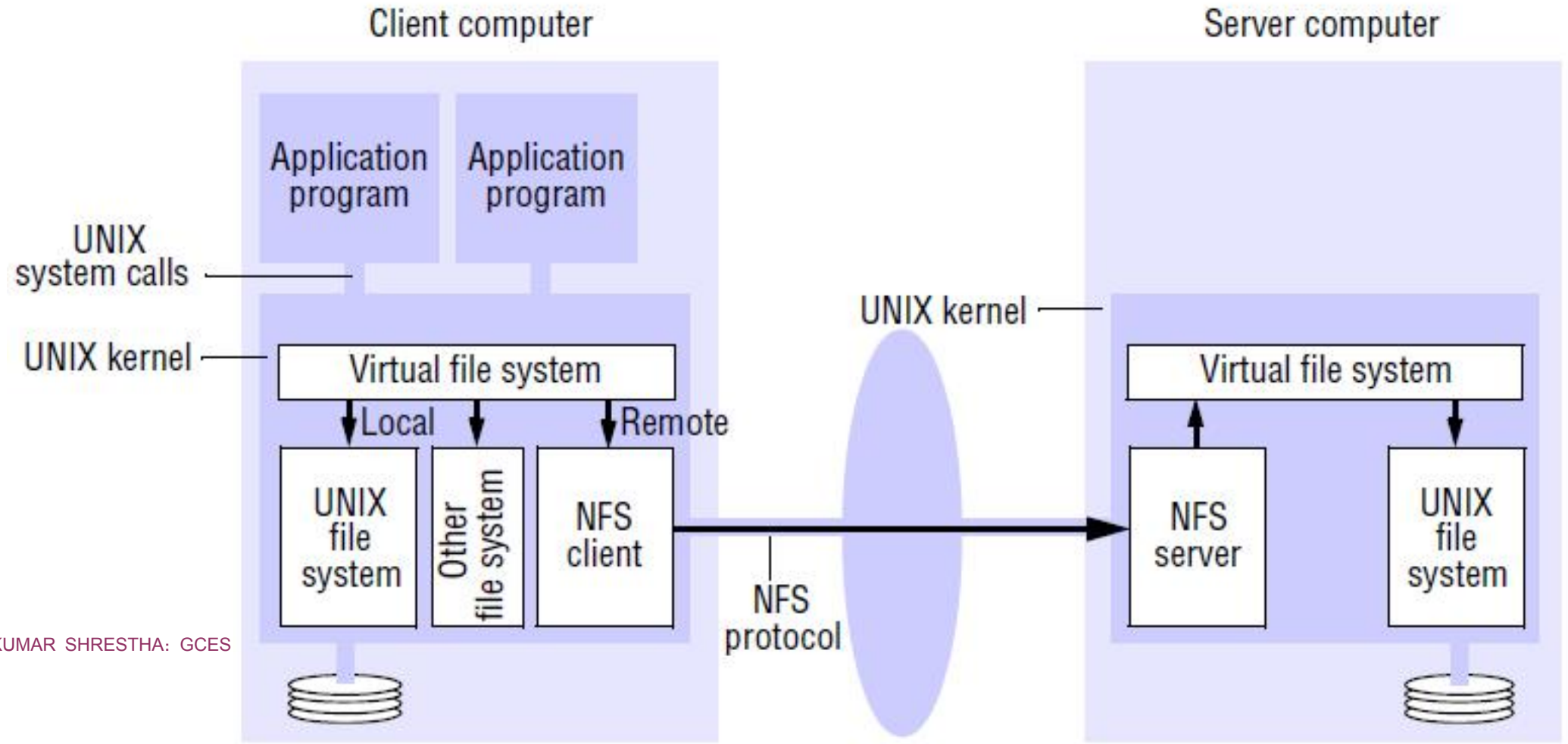
FILE OPERATIONS: FLAT FILE SERVICE

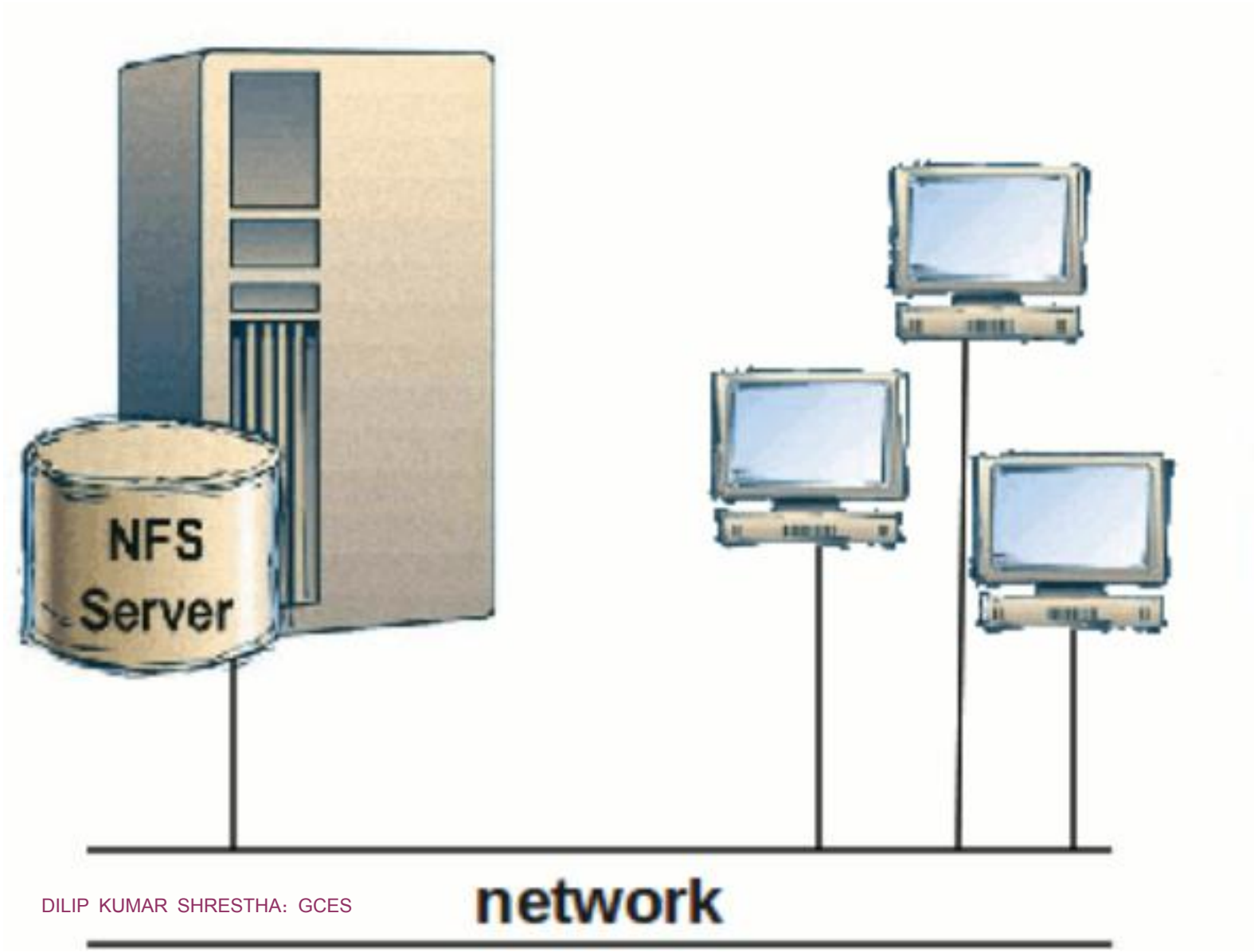
Read(FileId, i, n) → *Data* — throws *BadPosition*	If $1 \le i \le Length(File)$: Reads a sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If $1 \le i \le Length(File)+1$: Writes a sequence of *Data* to a file, starting 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 Figure 12.3).

FILE OPERATIONS: FLAT DIRECTORY SERVICE

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, removes the entry containing *Name* 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*.

SUN NETWORK FILE SYSTEM





GENERAL USAGES

TERMINOLOGIES

- Virtual File System
- Client Integration
- Access control and authentication
- NFS server interface
- Mount service
- Path name translation
- Automounter
- Server cashing
- Client caching

SERVER OPERATIONS

- GETATTR: Get file attributes
- SETATTR: Set file attributes
- LOOKUP: Lookup filename
- ACCESS: Check access permission
- READLINK: Read from symbolic link
- READ: Read from file
- WRITE: Write to file
- CREATE: Create a file
- MKDIR: Create a directory
- SYMLINK: Create a symbolic link
- MKNOD: Create a special device
- REMOVE: Remove a file
- RMDIR: Remove a directory
- RENAME: Rename a file or directory
- LINK: Create link to an object
- READDIR: Read From directory
- READDIRPLUS: Extended read from directory
- FSSTAT: Get dynamic file system information
- FSINFO: Get static file system information
- PATHCONF: Retrieve POSIX information

USEFUL REFERENCES

Protocol v3 specification	• https://tools.ietf.org/html/rfc1813
Paper	• http://people.cs.pitt.edu/~manas/courses/2510/nfs3.pdf
Installation	• https://help.ubuntu.com/lts/serverguide/network-file-system.html.en

THANKYOU