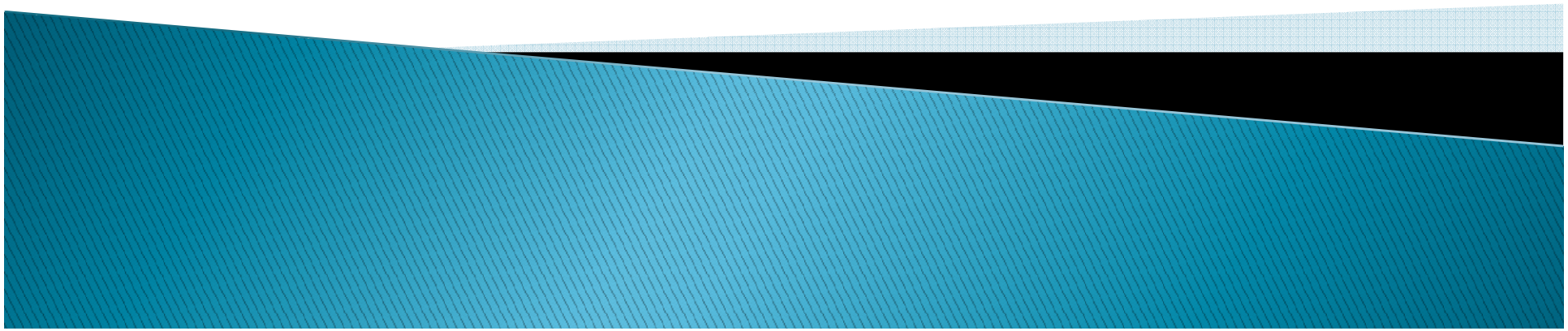
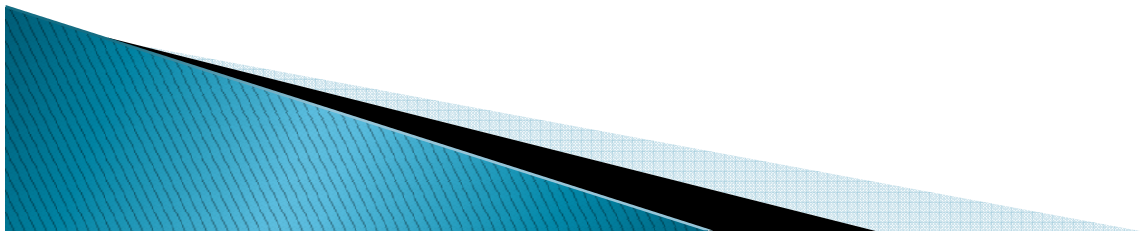
# FILE SERVICE ARCHITECTURE

By
M.Roshini
IIIrd year
CSE

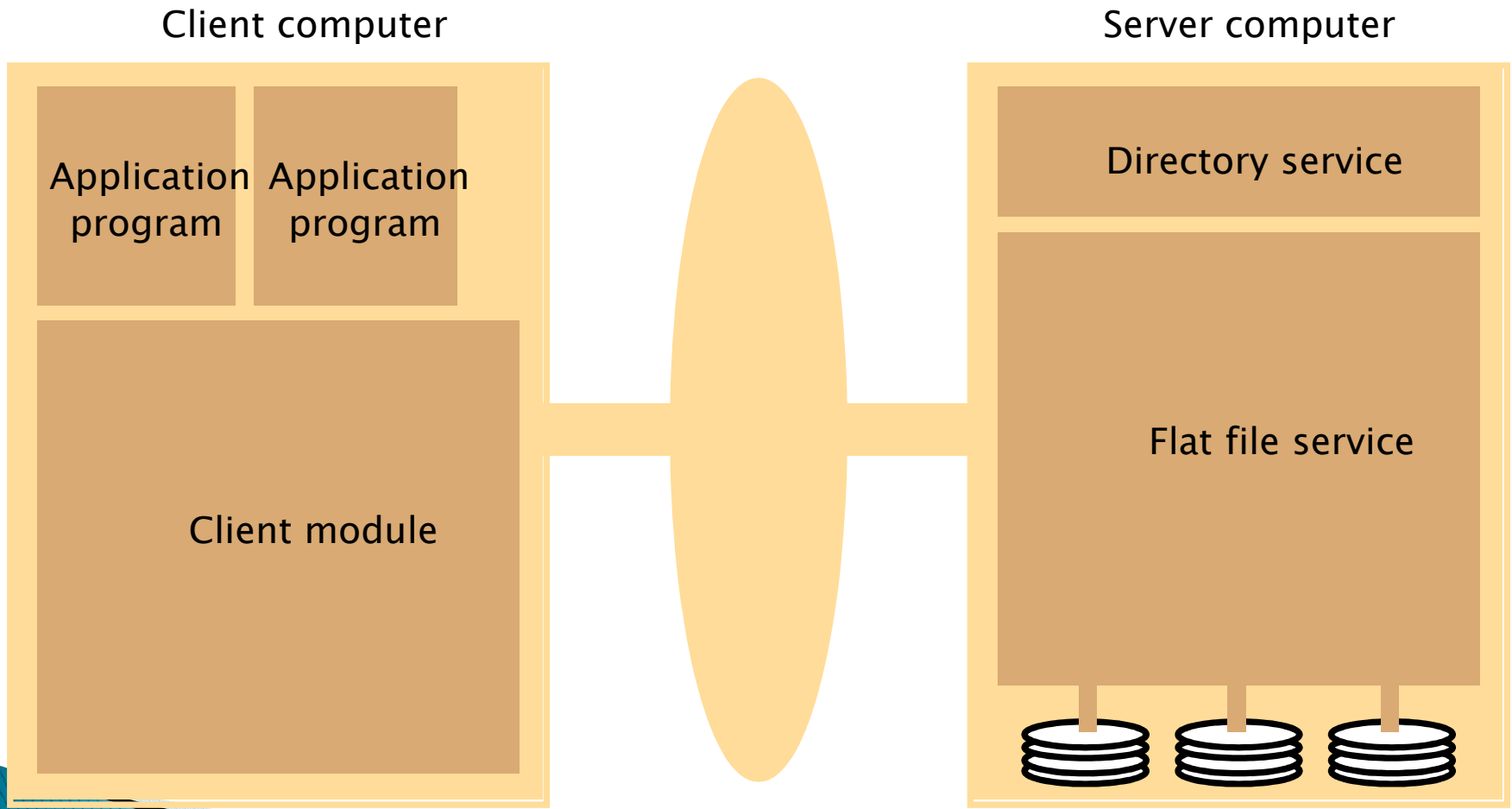


# FILE SERVICE ARCHITECTURE

- 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
  - A client module

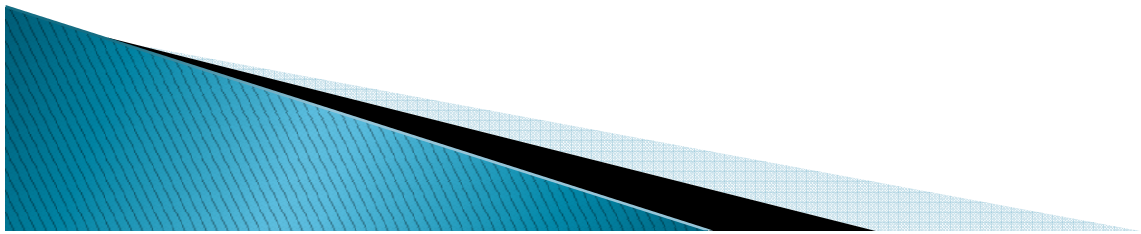


# FILE SERVICE ARCHITECTURE



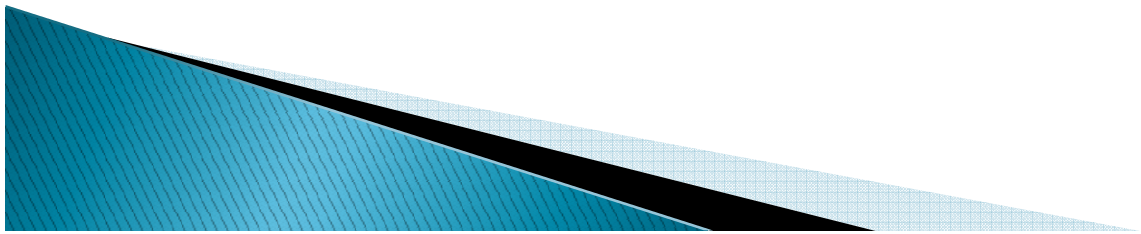
# FILE SERVICE ARCHITECTURE

- The Client module implements exported interfaces by flat file and directory services on server side.
- Responsibilities of various modules can be defined as follows:
  - Flat file service:
    - Concerned with the implementation of operations on the contents of file. Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat file service operations. UFIDs are long sequences of bits chosen so that each file has a unique among all of the files in a distributed system.



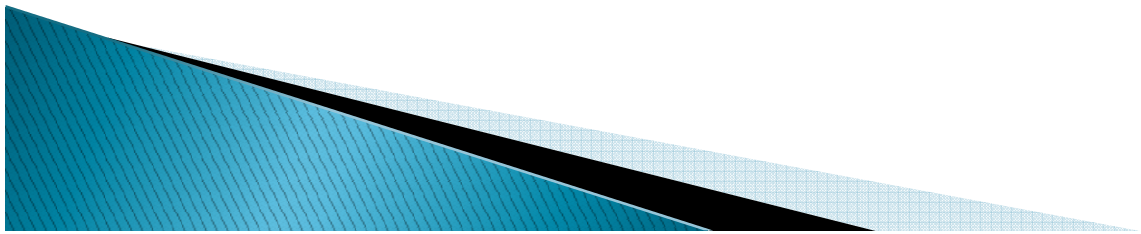
# FILE SERVICE ARCHITECTURE

- Directory service:
  - Provides mapping between text names for the files and their UFIDs. Clients may obtain the UFID of a file by quoting its text name to directory service. Directory service supports functions needed generate directories, to add new files to directories.



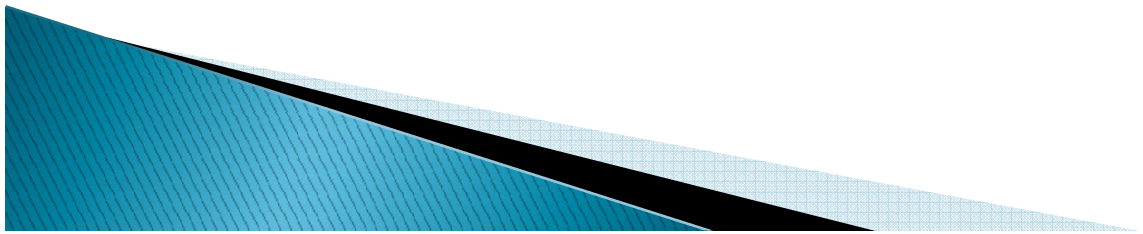
# FILE SERVICE ARCHITECTURE

- Client module:
  - It runs on each computer and provides integrated service (flat file and directory) as a single API to application programs. For example, in UNIX hosts, a client module emulates the full set of Unix file operations.
  - It holds information about the network locations of flat-file and directory server processes; and achieve better performance through implementation of a cache of recently used file blocks at the client.



# FILE SERVICE ARCHITECTURE

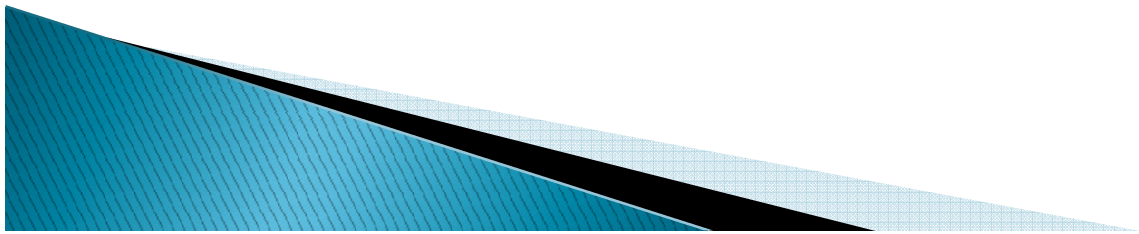
- Flat file service interface:
  - It contains a definition of the interface to a flat file service



# FILE SERVICE ARCHITECTURE

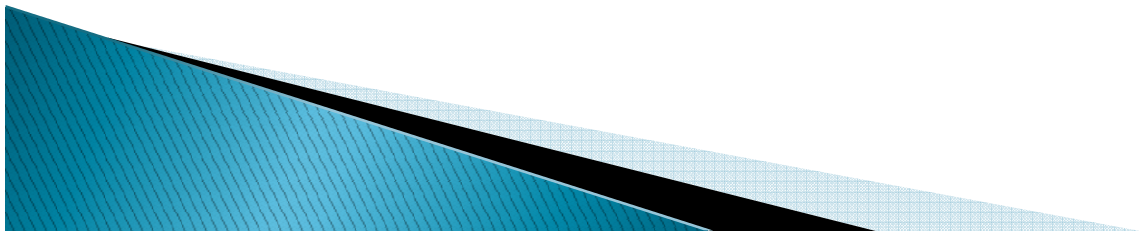
**Figure 12.6** Flat file service operations

| | |
|---|---|
| *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 Figure 12.3). |



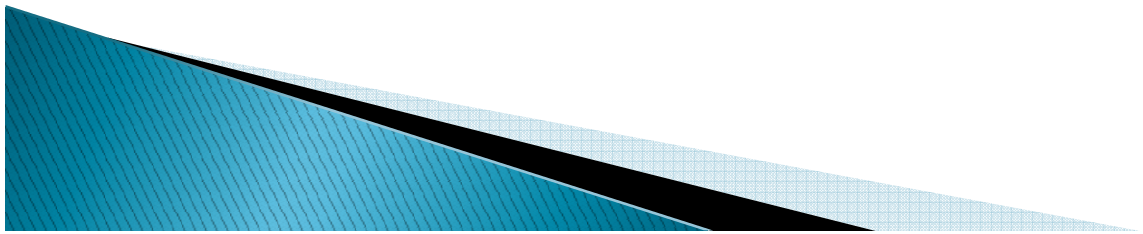
# SPECIFICATIONS

- ‘i’ –specify the position in the file.
- Read operation-copies the sequence of n data items beginning at item I from the specified file into Data,which is then returned to the client.
- Write operation-copies the sequence of n data items in Data into the specified file beginning at item i,replacing the previous content of the file.



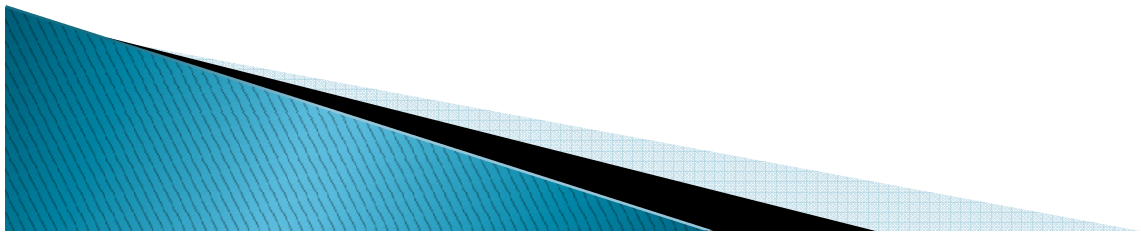
# REASONS FOR DIFFERENCE BETWEEN FLAT FILE SERVICE INTERFACE & UNIX FILE SYSTEM INTERFACE

- Repeatable operations-client can repeat the calls in which they receive no reply.
- Stateless servers-restarted after a failure and resume operation without any need for client and server to restore any state.



# FILE SERVICE ARCHITECTURE

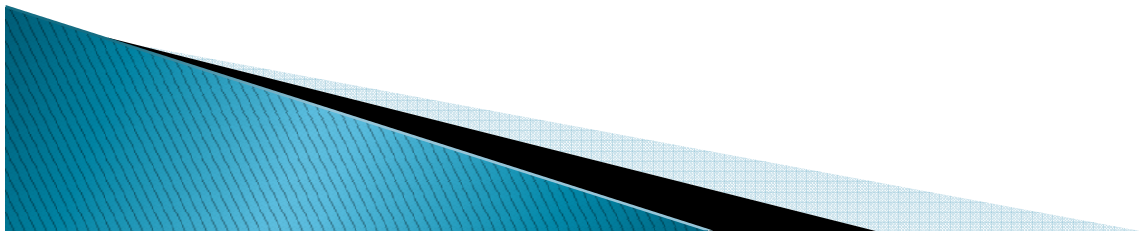
- Access control
  - In distributed implementations, access rights checks have to be performed at the server because the server RPC interface is an otherwise unprotected point of access to files.
- Directory service interface
  - The following figure contains a definition of the RPC interface to a directory service



# DIRECTORY FILE OPERATIONS

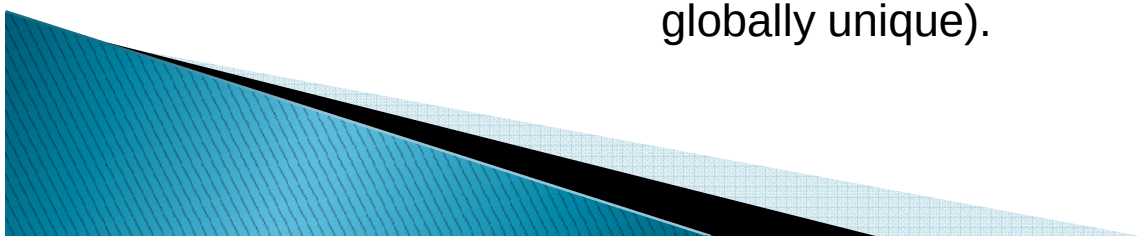
**Figure 12.7** Directory service operations

| | |
|---|---|
| *Lookup(Dir, Name)* → *FileId*<br>— 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)*<br>— throws *NameDuplicate* | If *Name* is not in the directory, adds *(Name, File)* to the directory and updates the file's attribute record.<br>If *Name* is already in the directory, throws an exception. |
| *UnName(Dir, Name)*<br>— throws *NotFound* | If *Name* is in the directory, removes the entry containing *Name* from the directory.<br>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*. |



# FILE SERVICE ARCHITECTURE

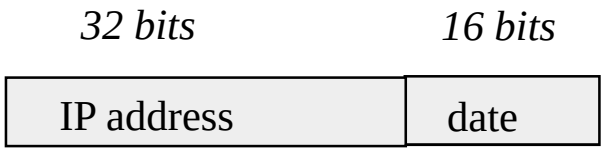
- Hierarchic file system
  - A hierarchic file system such as the one that UNIX provides consists of a number of directories arranged in a tree structure.
  - Each directory holds the names of the files and other directories that are accessible from it.
- File Group
  - A file group is a collection of files that can be located on any server or moved between servers while maintaining the same names.
    - A similar construct is used in a UNIX file system.
    - It helps with distributing the load of file serving between several servers.
    - File groups have identifiers which are unique throughout the system (and hence for an open system, they must be globally unique).



# FILE SERVICE ARCHITECTURE

To construct a globally unique ID we use some unique attribute of the machine on which it is created, e.g. IP number, even though the file group may move subsequently.

File Group ID:



# THANK YOU