File Management

IT2758

Operating Systems

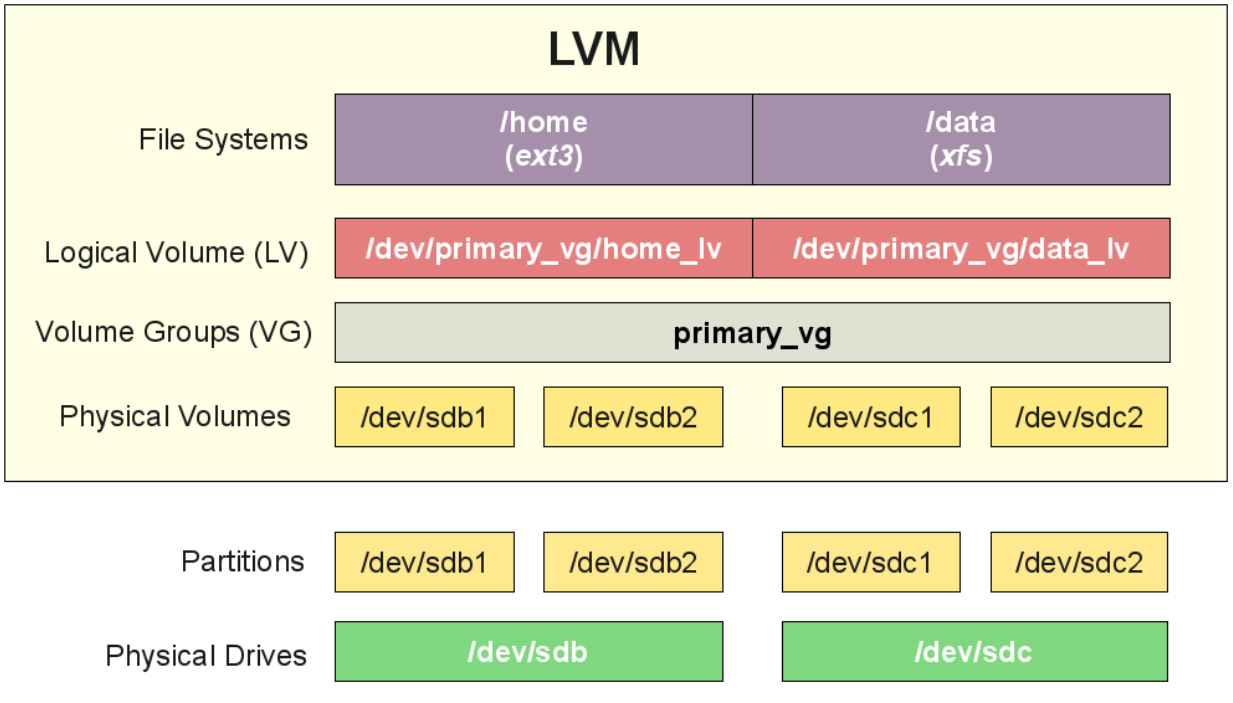
Hierarchy

Linux

- Drives
- Partitions (boot, swap, file)
- Groups
- Volumes
- Filesystems

Windows

- Drives
- Partitions (pri, logical)
- Volumes
- Filesystems



File Systems

File system	Creator	Year introduced	Original operating system	Max File/Vol Size
FAT16	Microsoft	1984	MS-DOS 3.0	2GB, 2GB
XFS	SGI	1994	Irix	8EB
Joliet ("CDFS")	Microsoft	1995	MS Windows, Linux, Mac OS, and FreeBSD	4GB, 8TB
FAT32	Microsoft	1996	Windows 95b	4GB, 2TB
FATX	Microsoft	2002	Xbox	2GB, 2GB
NTFS Version 3.1	Microsoft	2001	Windows XP	16EB, 16EB
exFAT	Microsoft	2006, 2009	Win CE 6.0, Win XP SP3, Win Vista SP1	127PB, 512TB
HFS+	Apple Computer	1998	Mac OS 8.1	8EB, 8EB
ext3	Stephen Tweedie	1999	Linux	2TB, 32TB
ext4	Various	2006	Linux	16TB, 1EB
Google File System	Google	2003	Linux	8EB, 8EB
GFS2	Red Hat	2006	Linux	
MINIX V3 FS	A. S. Tanenbaum	2005	MINIX 3	
VMFS5	VMware	2011	VMware ESXi 5.0	
F2FS	Samsung	2012	Linux	

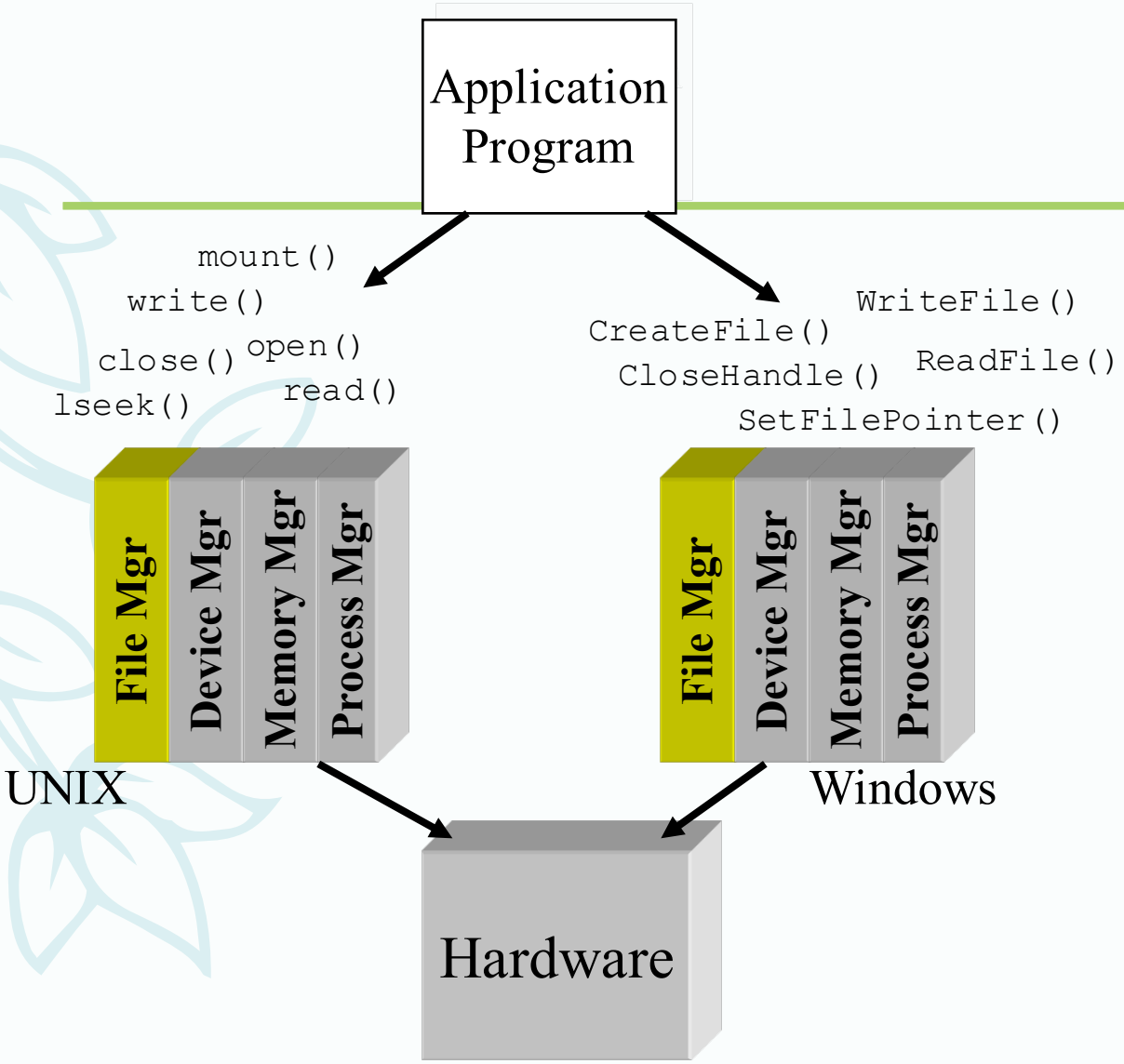
The Need for Files

- A computer system requires to store data in a non-volatile storage system.
- Although the computer system is good at computation, stable storage of data is required in order to perform meaningful tasks, e.g., storing company data, application programs, etc.
- Files are the OS mechanism for organizing and managing the data in a computer system for storage.
- Types of files include executable, data, text (including html), among many others.

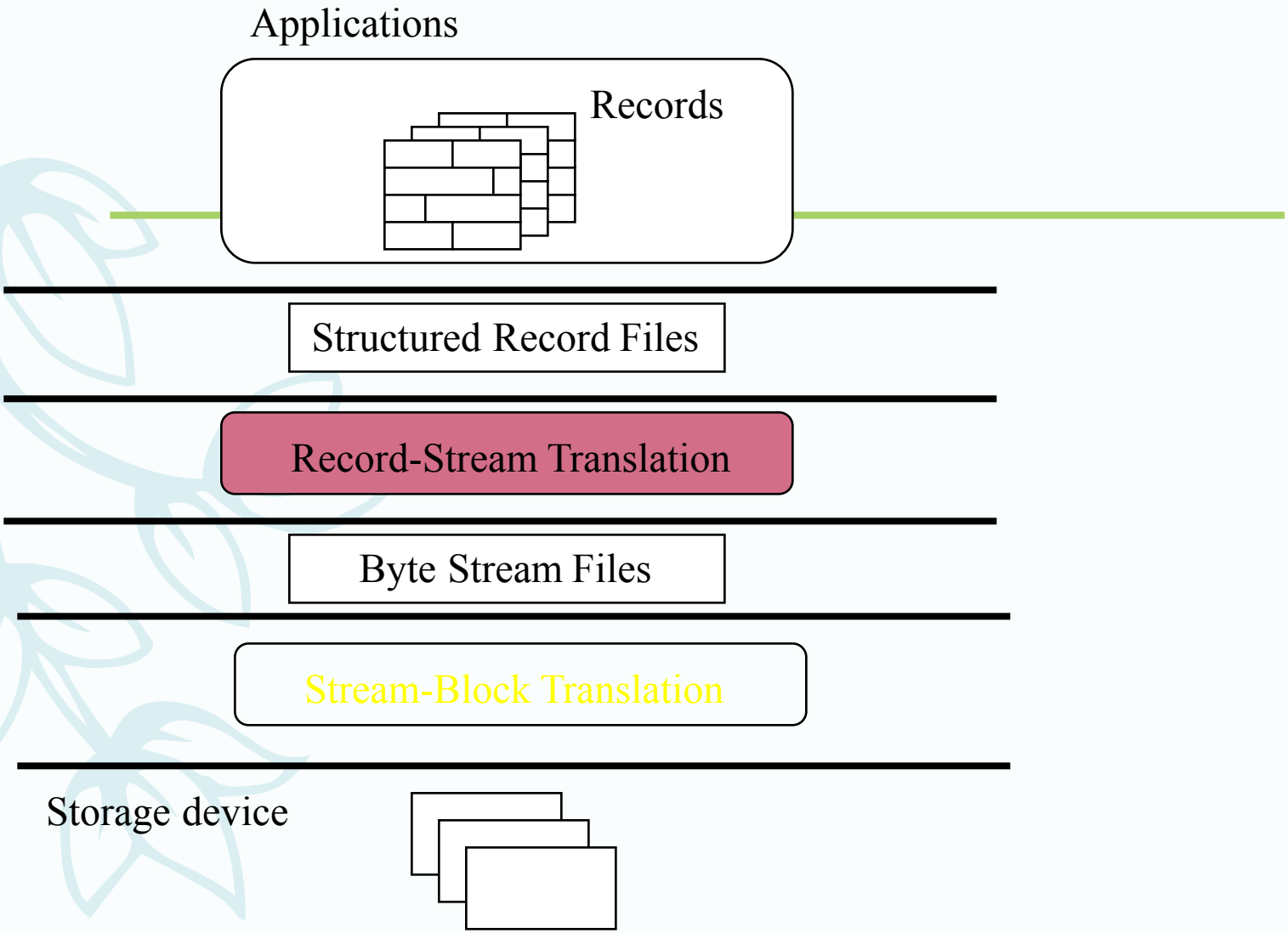
Files

- Almost every application needs to read from or write to files.
- To support file handling, almost all programming languages provide file handling routines.
- These routines, when activated, execute the OS system calls to handle the files.
- The manager in the OS that handles files is the file manager.

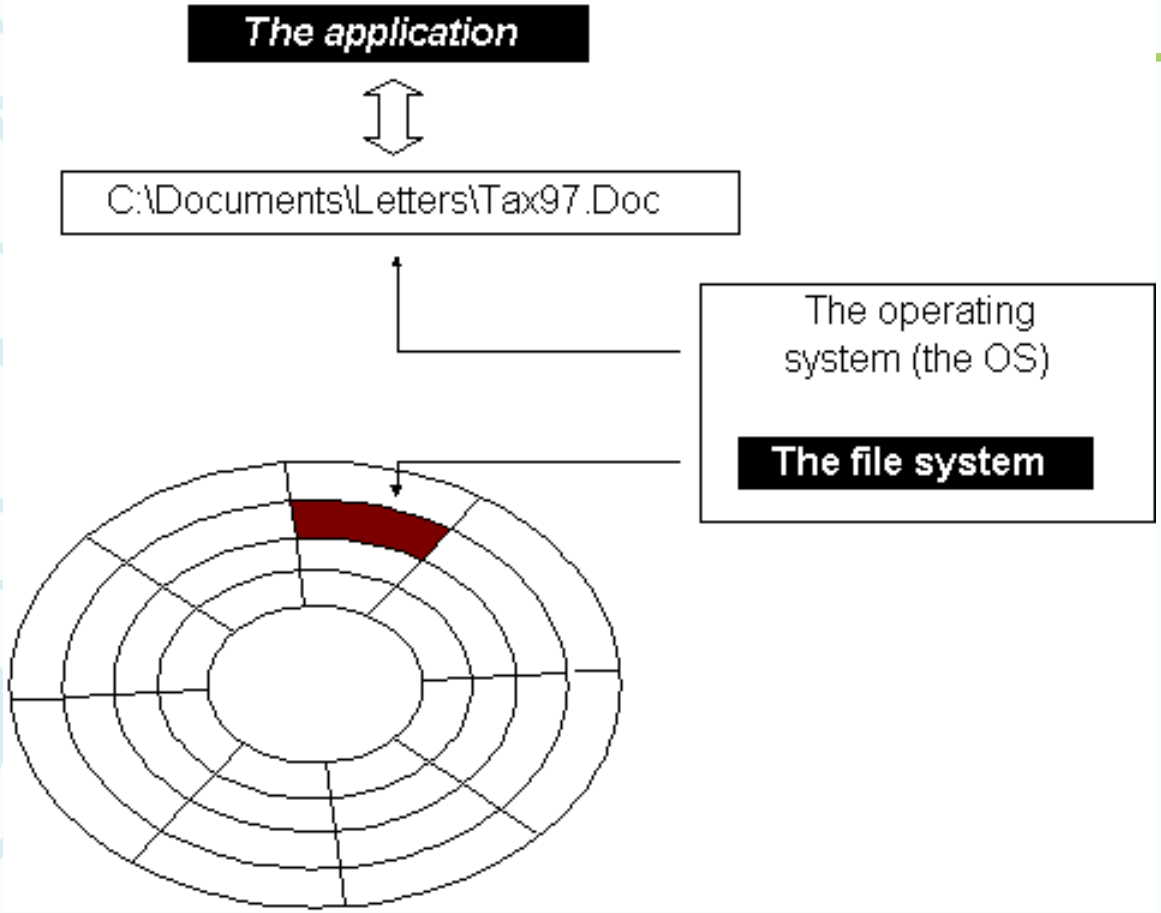
View of the File Manager



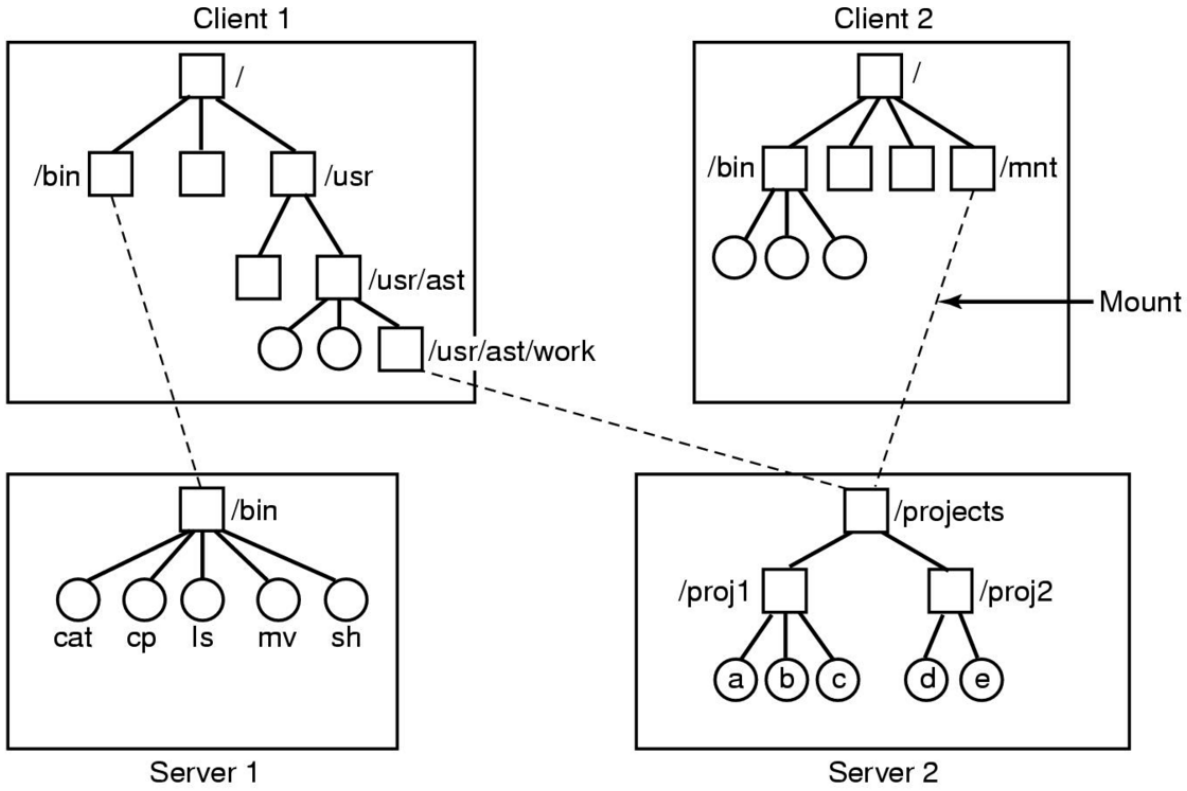
Information Structure



Getting to your file



Network File System (1)



- Examples of remote mounted file systems
- Directories are shown as squares, files as circles

Block Management / File Space Allocation

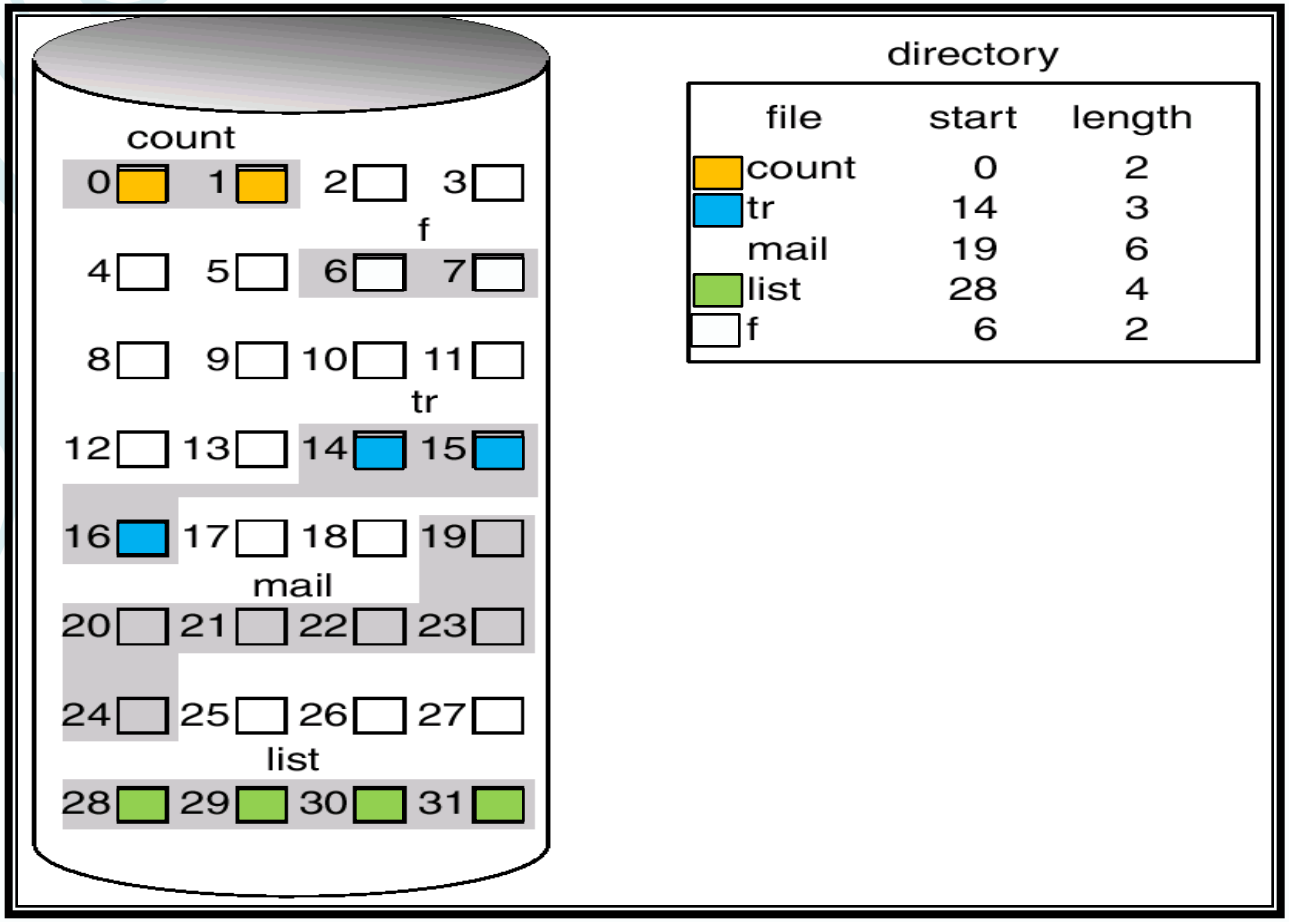
- In an actual storage system, data is stored in blocks/cluster, usually of equal size.
- 1 cluster = multiple blocks
- Three well-known techniques of organizing these blocks :
  - As a **Contiguous** set of blocks.
  - As a **List** of blocks interconnected with links.
  - As a collection of blocks interconnected by a **file Index**.
- http://courses.cs.vt.edu/csonline/OS/Lessons/FileManagement/index.html

Contiguous Allocation

- This method requires each file to occupy a set of contiguous blocks on the disk.
- Disk addresses define a linear ordering on the disk.
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the file.
- Directory entry for each file indicates the address of starting block and length of the area allocated to this file.

Contiguous Allocation

– In the example below, is allocated 2 consecutive blocks while mail is allocated 6 consecutive blocks.



Contiguous Allocation Advantages

- Support both sequential and direct access
  - For sequential access, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block.
  - For direct access to block i of a file that starts at block b, we can immediately access block b + i
- Minimal disk seek time
  - As disk addresses are defined in a linear ordering on the disk, accessing block b+1 after block b normally requires no head movement. Moreover, when head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder) it is only one track. Thus the number of disk seeks required for accessing contiguously allocated files is minimal.

Contiguous Allocation Disadvantages

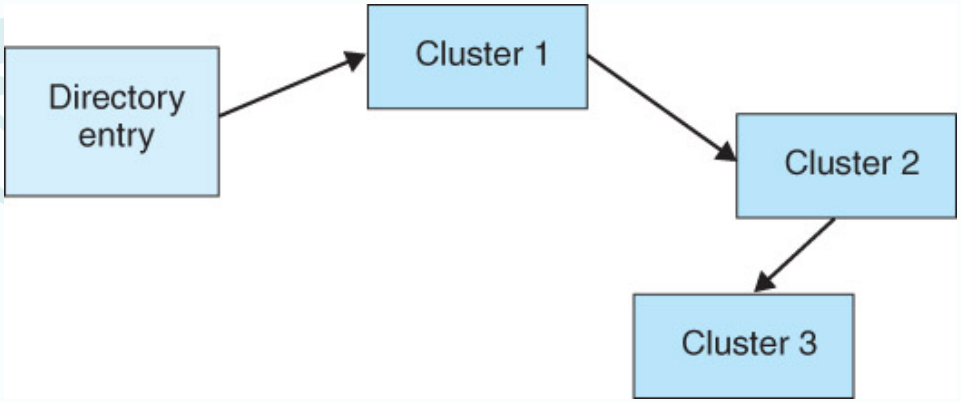
- External Fragmentation
  - As files are allocated and deleted, the free disk space is broken into little pieces thus resulting in non-contiguous blocks.
  - The solution to this problem is to perform compaction on the disk blocks. All the free space is compacted into one large hole. However, it is time consuming.
- Unknown file size
  - When a file is created, the total amount of space it needs must be allocated, but in some cases, example, output file, the size is not known and may be difficult to extend later.
  - One possible solution is to find a bigger hole and copy the contents of the file to the new space, then release the previous space to the free-space list (this should be transparent to the user). However, this is time consuming.

Contiguous Allocation Disadvantages

- **Known file size**
  - Even if the file size is known in advance, pre-allocation may be insufficient because file will grow over a long period of time.
  - If too much space is pre-allocated, system will suffer from internal fragmentation.
  - One possible solution is to allocate a continuous chunk of space to the file initially. If the space is not enough after the file has grown, another chunk of continuous space (an extent) is added to the initial allocation. The location of a file's block is then recorded as a location and a block count, plus a link to the first block of the next extent.

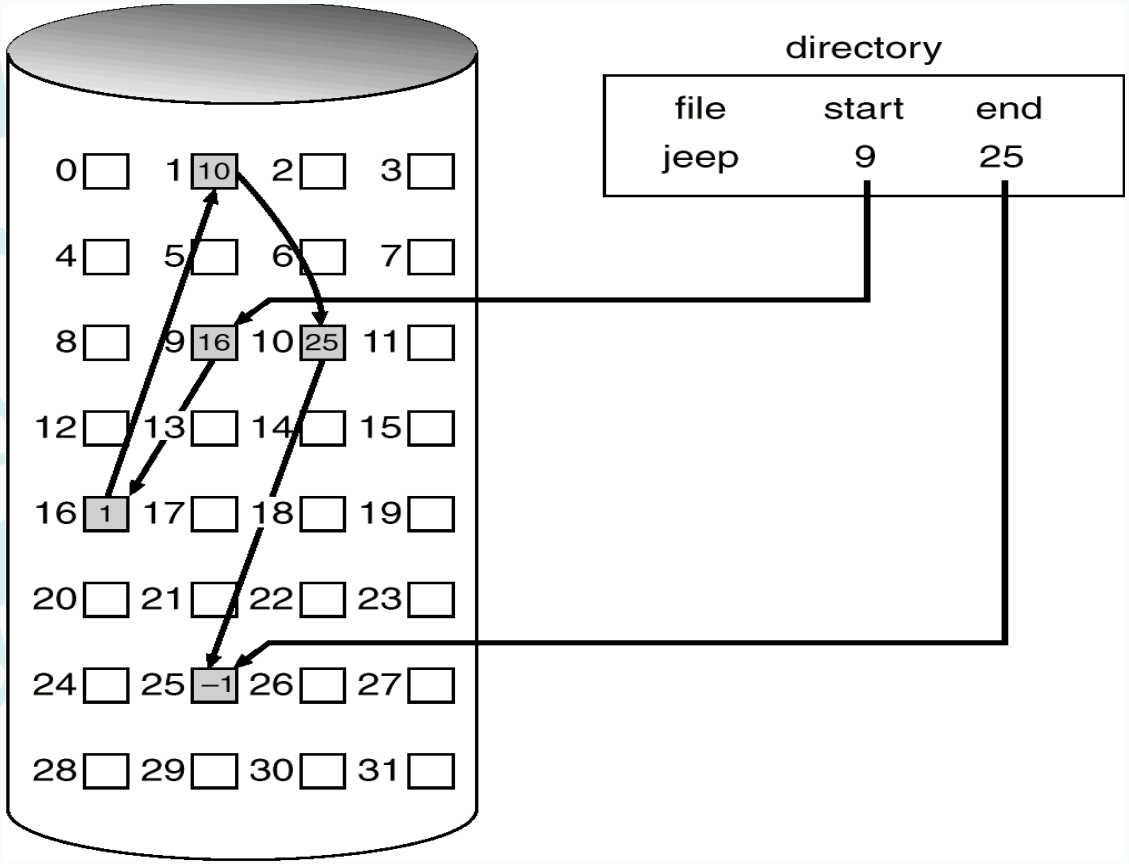
Linked Allocation

- Linked allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of blocks; the disk blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block.
- To create a file, we simply create a new entry in the directory and the pointer to the first disk block of the file is initialized to nil (the end-of-list pointer value) to signify an empty file.
- A write to a file removes the first free block from the free-space list and linked to the end of the file.
- To read a file, simply follow the pointers from block to block.



Linked Allocation - Example

– In the example below, a file of five blocks might start at block 9, continue at block 16, then block 1, block 10, and finally block 25.



Linked Allocation - Advantages

- Disk space need not be contiguous
- No external fragmentation
  - Any free block on the free-space list can be used to satisfy a request, since all blocks are linked together.
    - There is no necessity to compact disk space.
- No need to declare the size of file
  - The size of a file can grow as long as there are free blocks

Linked Allocation - Disadvantages

- Inefficient for direct access
  - To find the ith block of a file, we must start at the beginning of that file, and follow the pointers until we get to the ith block.
- Pointer space
  - Extra space in each block is required to store the pointers. Thus, each file would require slightly more space.
  - One possible solution is to collect blocks into multiples, called clusters, and to allocate clusters rather than blocks. However, the cost of this approach is an increase in internal fragmentation.

Linked Allocation - Disadvantages

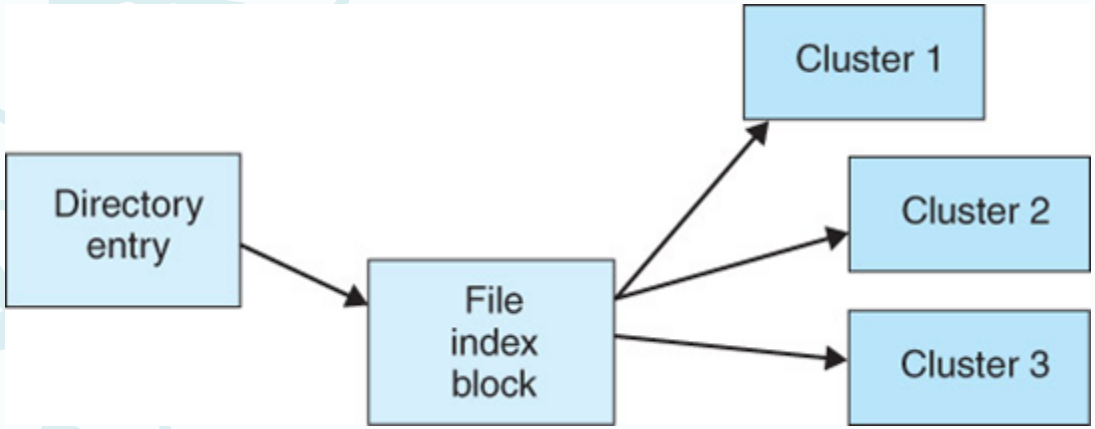
- Reliability problem
  - As the files are linked together by pointers scattered all over the disk, consider what would happen if a pointer was lost or damaged. A bug in the operating-system software or a disk hardware failure might result in picking up the wrong pointer thus accessing the wrong block. A partial solution is to implement double linked list. However, this increases the overhead.

Indexed Allocation

- Linked allocation does not support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order. Indexed allocation solves this problem by **bringing all the pointers together into one location**: the index block.
- Each file has its own index block, which is an array of disk-block addresses. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block.
- To read the ith block, use the pointer in the ith index block entry to find and read the desired block.
- When the file is created, all pointers in the index block are set to nil. When the ith block is first written, a block is removed from the free-space list and its address is put in the ith index-block entry.

Indexed Allocation - Advantages

- Supports direct access
- No external fragmentation as any free block on the disk may satisfy a request for more space

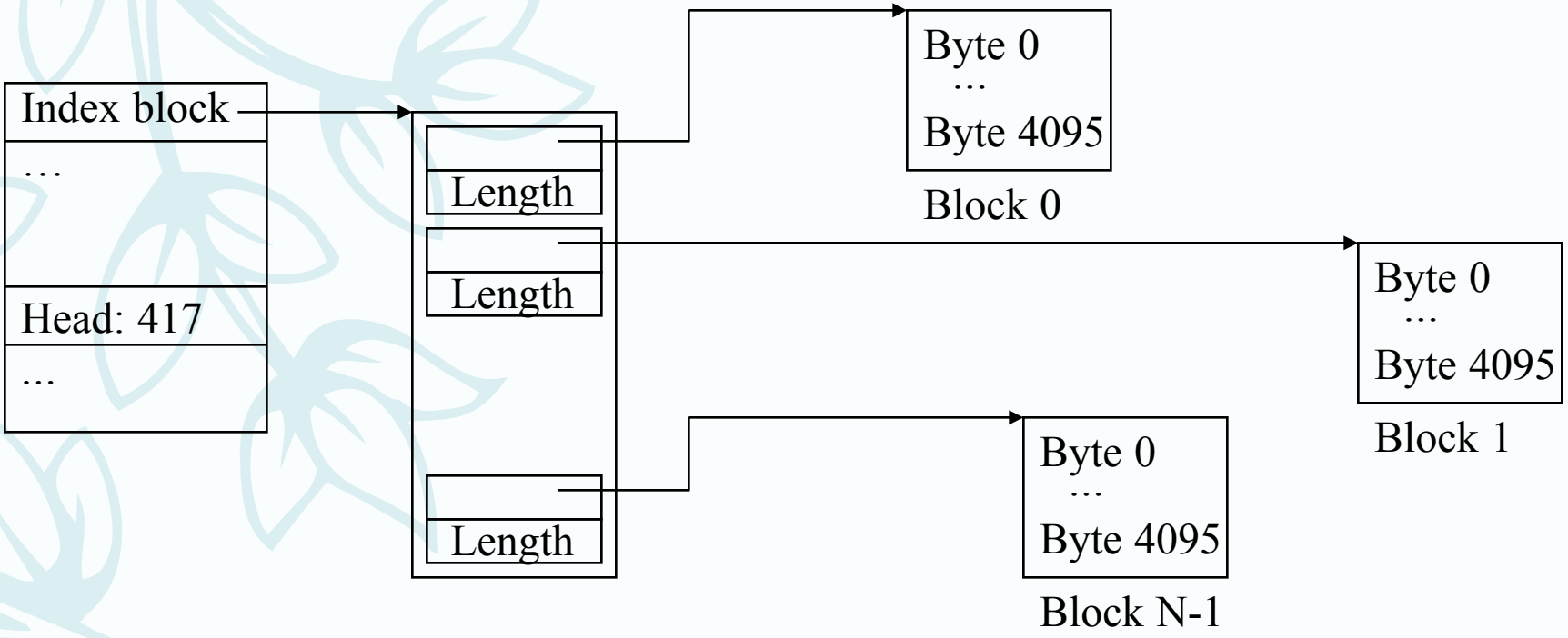


Indexed Allocation - Disadvantage

- Wasted space
  - The pointer overhead of index block is generally greater than the pointer overhead of linked allocation.
  - An entire index block is allocated to a file even if only one or two pointers are used.
- Inevitably, we have to decide on the size of the index block (If index block is too large, space are wasted. Too small an index block will not be able to hold enough pointers for large file).
- There are some mechanisms deal with this issue, namely,
  - linked index
  - multi-level index
  - combined scheme

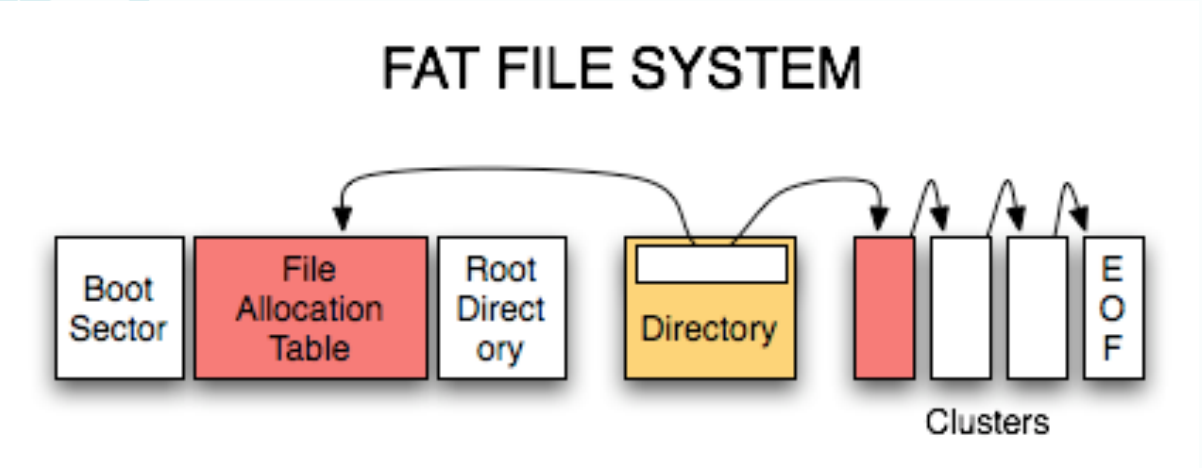
Indexed Allocation

- Extract headers and put them in an index
- Simplify seeks
- May link indices together (for large files)



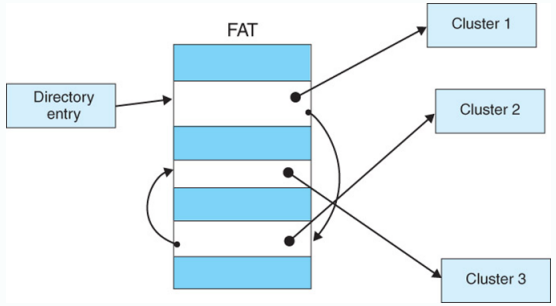
Linked Allocation - FAT

- An important variation on the **linked allocation** is the use of File-Allocation Table (FAT).
- Using FAT is a simple efficient method of disk-space allocation.
- Used by MS-DOS(Windows) and OS/2 operating systems.
- A section of disk at the beginning of each partition is set aside to contain the table. The table has one entry for each disk block and is indexed by block number. The FAT is used much like a linked list.

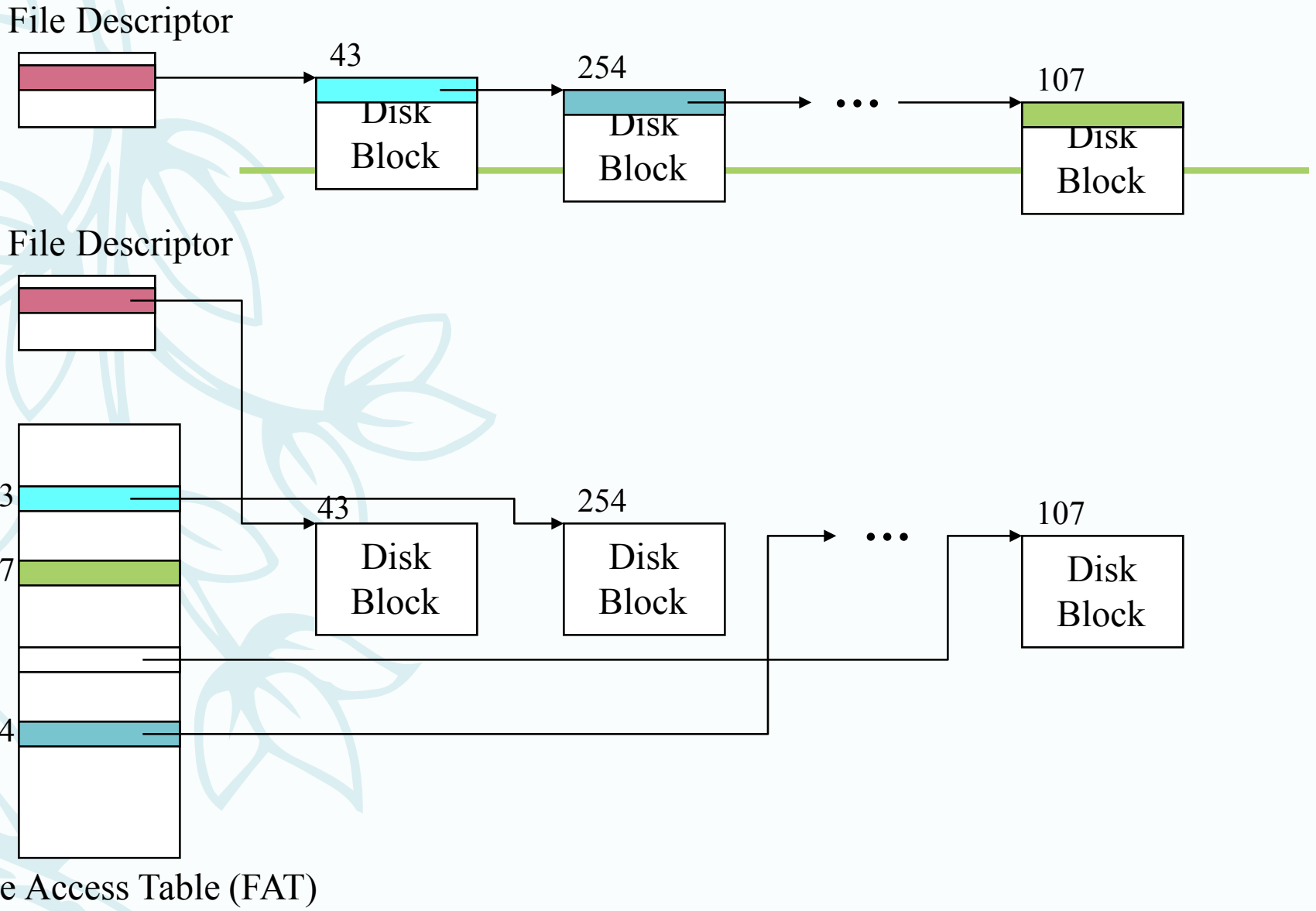


Linked Allocation - FAT

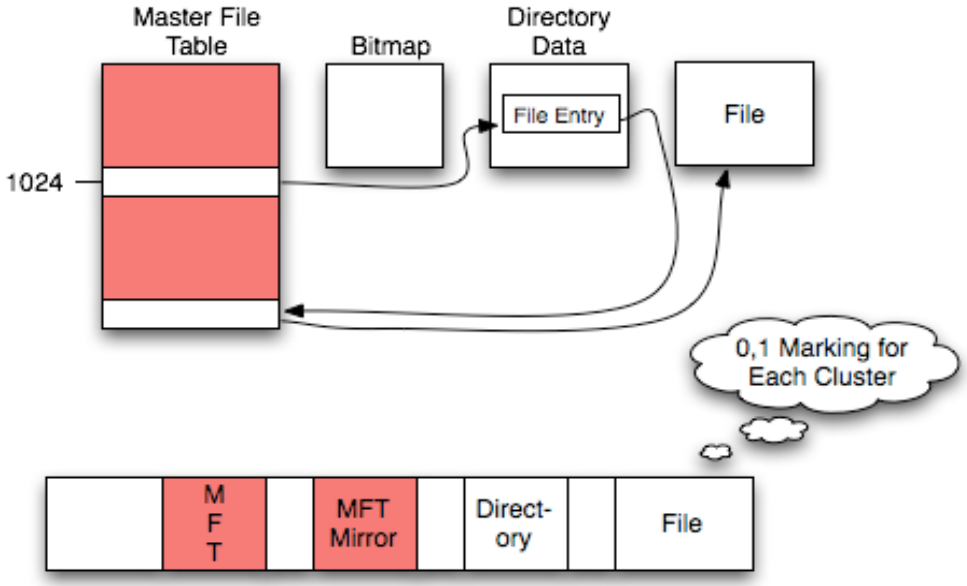
- The directory entry contains the block number of the first block of the file. The table entry indexed by the block number then contains the block number of the next block in the file. This chain continues until the last block, which has a special end-of-file value as the table entry. Unused blocks are indicated by a 0 table value.
- FAT allocation scheme results in significant number of disk head seek, unless the FAT is cached.
- Important limitation is the size of the file system is limited. This has resulted in various improvements in the FAT system, including FAT32. But this requires one to upgrade the file system, which can be painful.



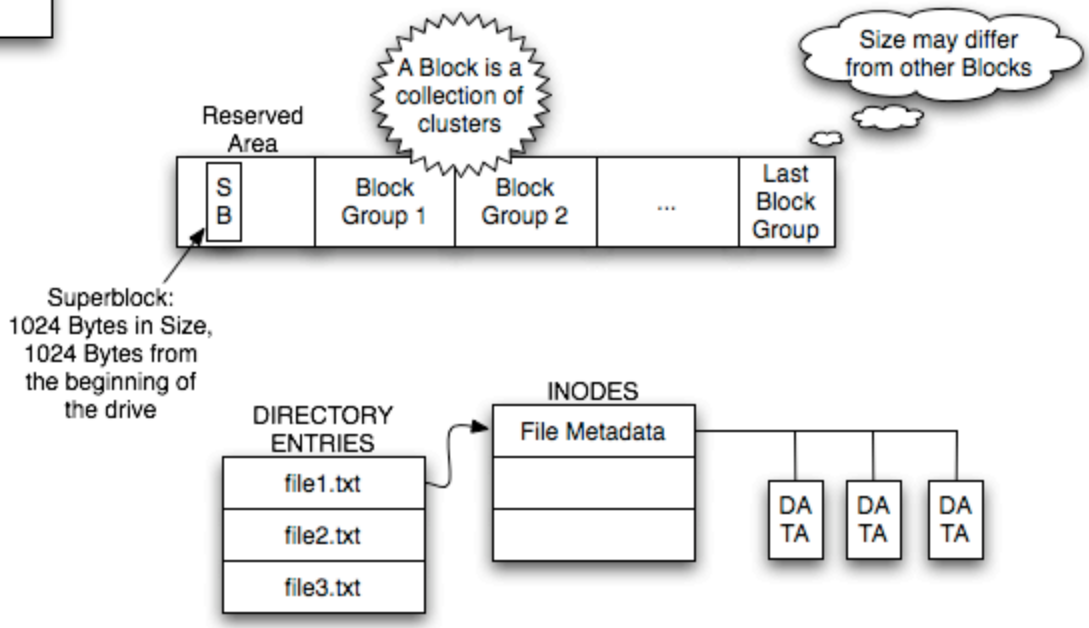
DOS FAT Files



NTFS FILE SYSTEM



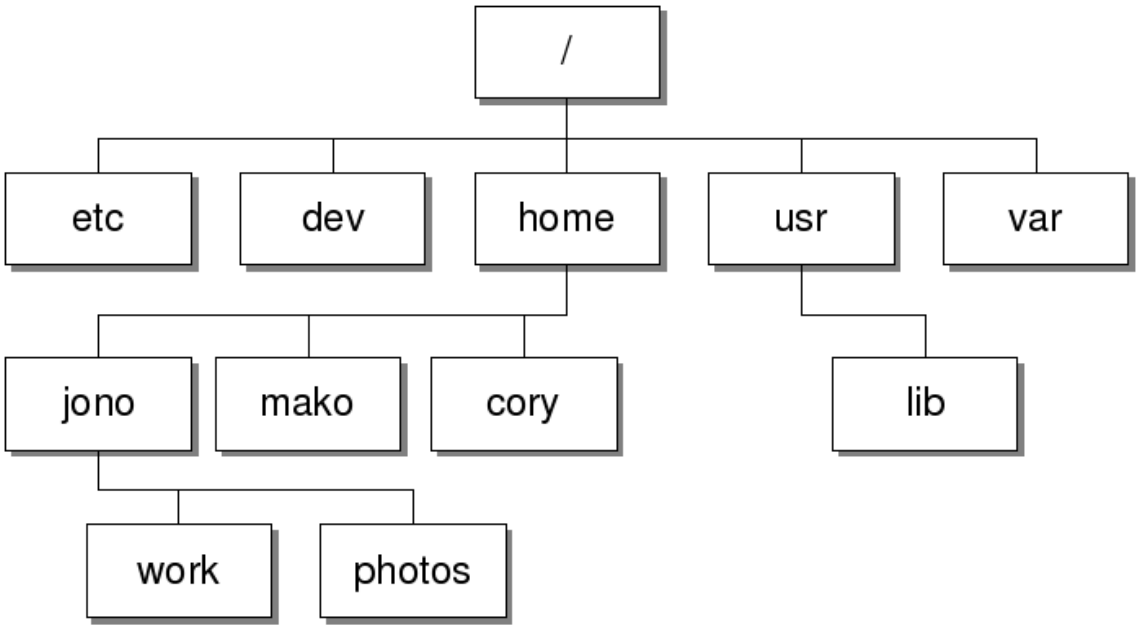
EXT2/3 FILE SYSTEM



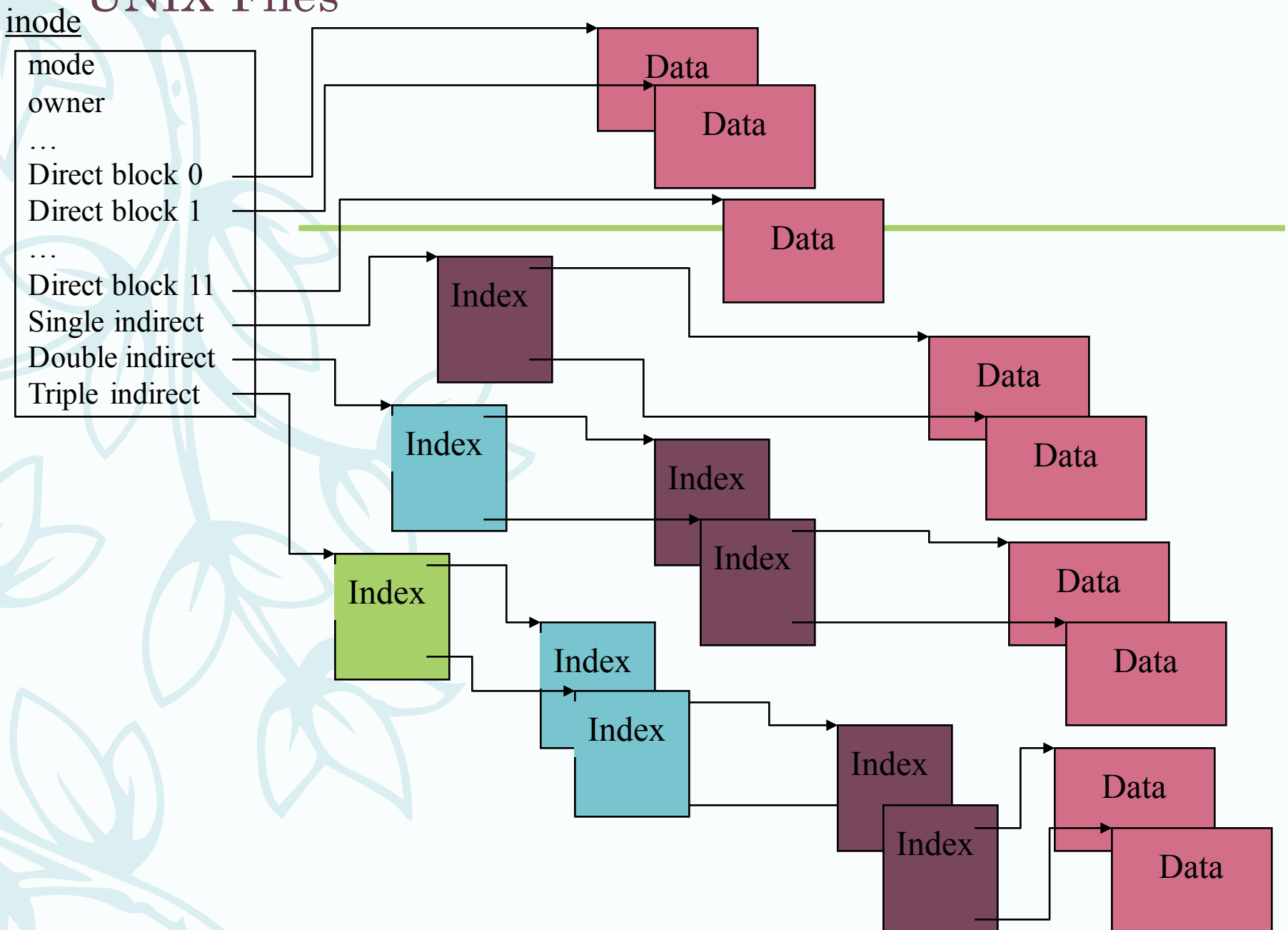
Indexed Allocation – UNIX File Structure

- UNIX/Linux uses a variant of the indexed allocation scheme.

ar, the



UNIX Files


inode
mode
owner
…
Direct block 0
Direct block 1
…
Direct block 11
Single indirect
Double indirect
Triple indirect
Data
Data
Data
Index
Data
Data
Index
Index
Index
Data
Data
Index
Index
Index
Data
Index
Data
Index

Disk Allocation Method - Performance

- Some systems support contiguous allocation for direct access files and linked allocation for sequential access files. For this case, a file created for sequential access cannot be used for direct access. A file created for direct access will be contiguous and can support both sequential and direct access, but its maximum length must be declared when it is created.
- Some systems combine contiguous allocation with indexed allocation by using continuous allocation for small files and indexed allocation for large files. In this system, as the file size grows, the allocation method will automatically switch from contiguous to indexed allocation.

File Descriptors

- The file manager, in addition to storing the files, also manages the various information that describe the files. (file descriptors).
  - External name
  - Current state
  - Sharable
  - Owner
  - User
  - Locks
  - Protection settings
  - Length
  - Time of creation
  - Time of last modification
  - Time of last access
  - Reference count
  - Storage device details

Directories

- File manager not only needs to provide mechanisms for applications to create, write, read files, it also needs to provide ways to manage *collections* of files.
- A file directory is a set of logically associated files and other sub-directories of files.
- We use directories to help us organize *logically* the different files that we may handle.
- For e.g., we can store the files associated with a project in one directory and those with another project in another directory.

Directories

- The file manager provides a set of commands to allow users to administer directories, including :
  - Enumerate: Returns a list of all the files and nested directories. E.g., dir in Windows, or ls in UNIX/Linux.
  - Copy: Create a duplicate of an existing file.
  - Rename: Changes the name of the file.
  - Delete: Removes the file from the directory. Can also be used to delete directories.
  - Traverse: Allows a user to explore the directories by 'moving' into or out of directory structures.

Conclusion

- File system defines the ability of the computer system to store data.
- OS provides the interfaces required by application programs to manage data in files.
- File manager also provides mechanisms to organize files into folders called directories.
- Different methods of implementing files and directories result in differing performance.