File Systems

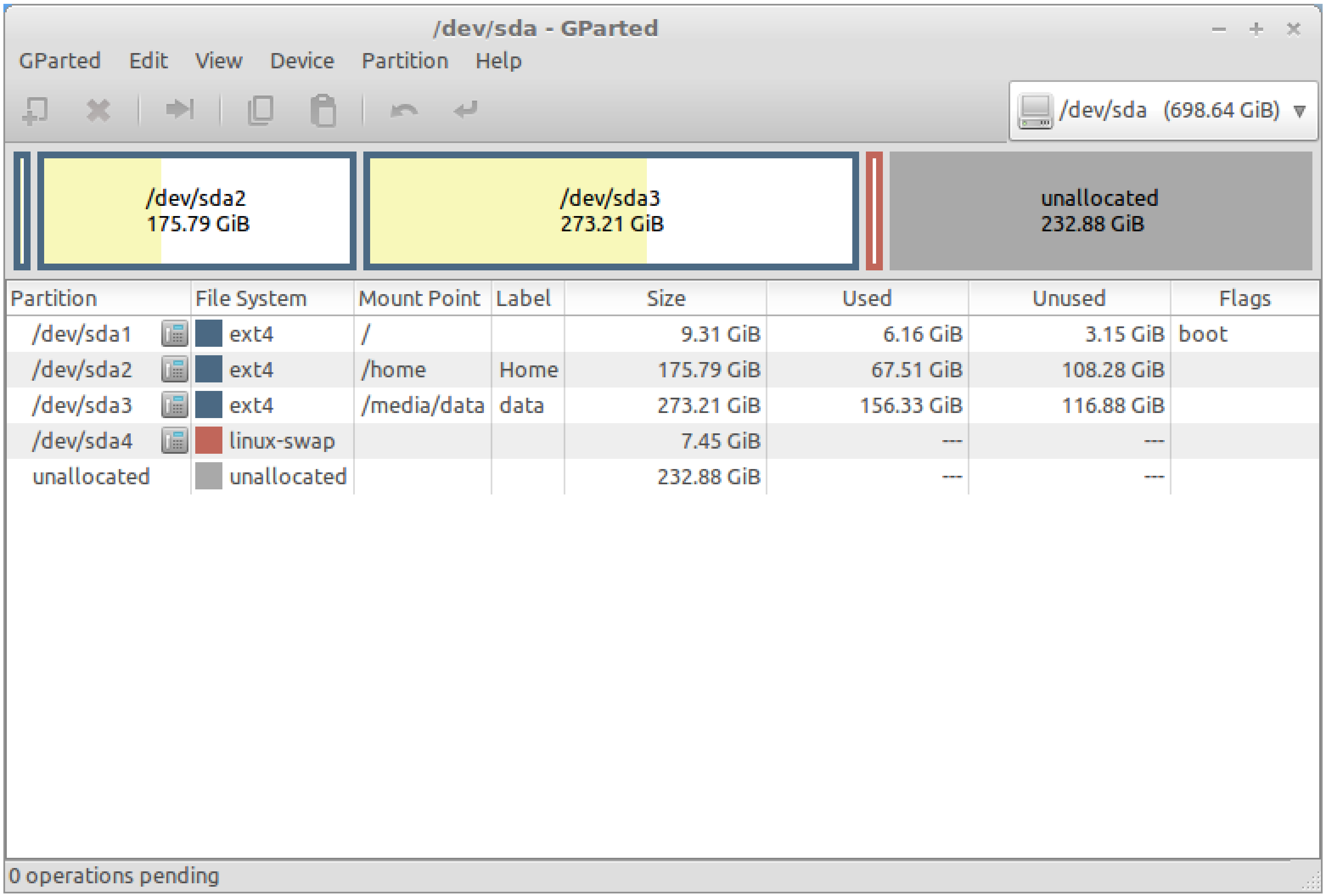
File System Layout

- Filesystem refer to an entire hierarchy of directories, or directory tree, that is used to organize files on a computer system.
- File systems usually are stored on disks
- Most disks can be divided up into partitions
  - Independent file systems on each partition
- Sector 0 of the disk is the Master Boot Record (MBR)
  - By the way a sector is a disk unit (more later)
- The end of the MBR contains the partition table
  - Gives the start and end of each partition
  - One of the partitions is marked as active

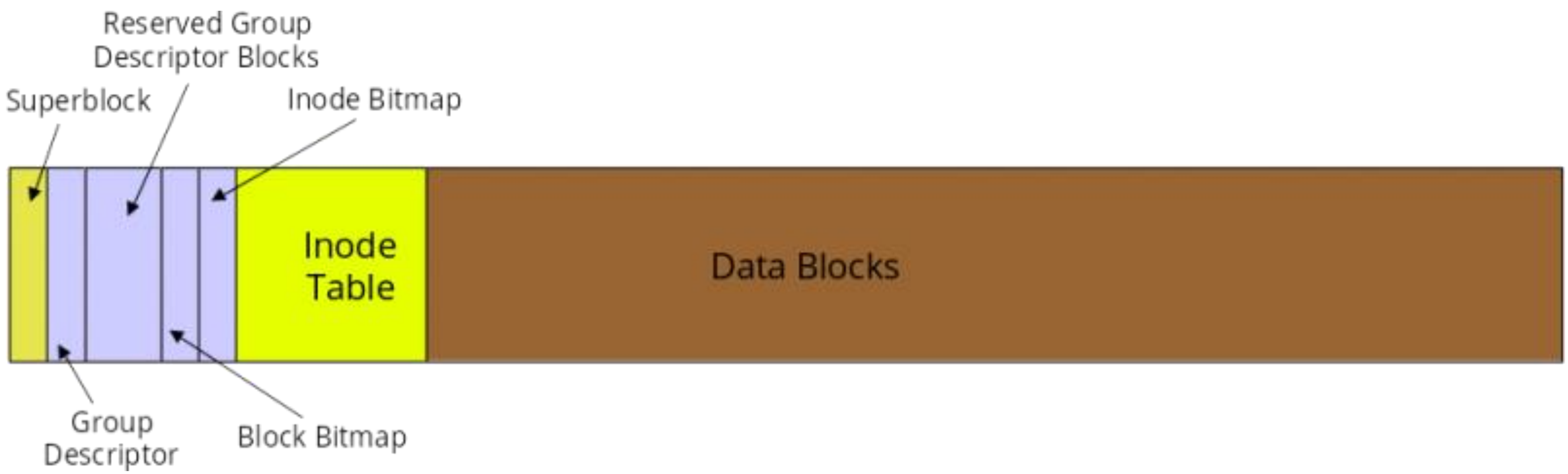
File System Layout

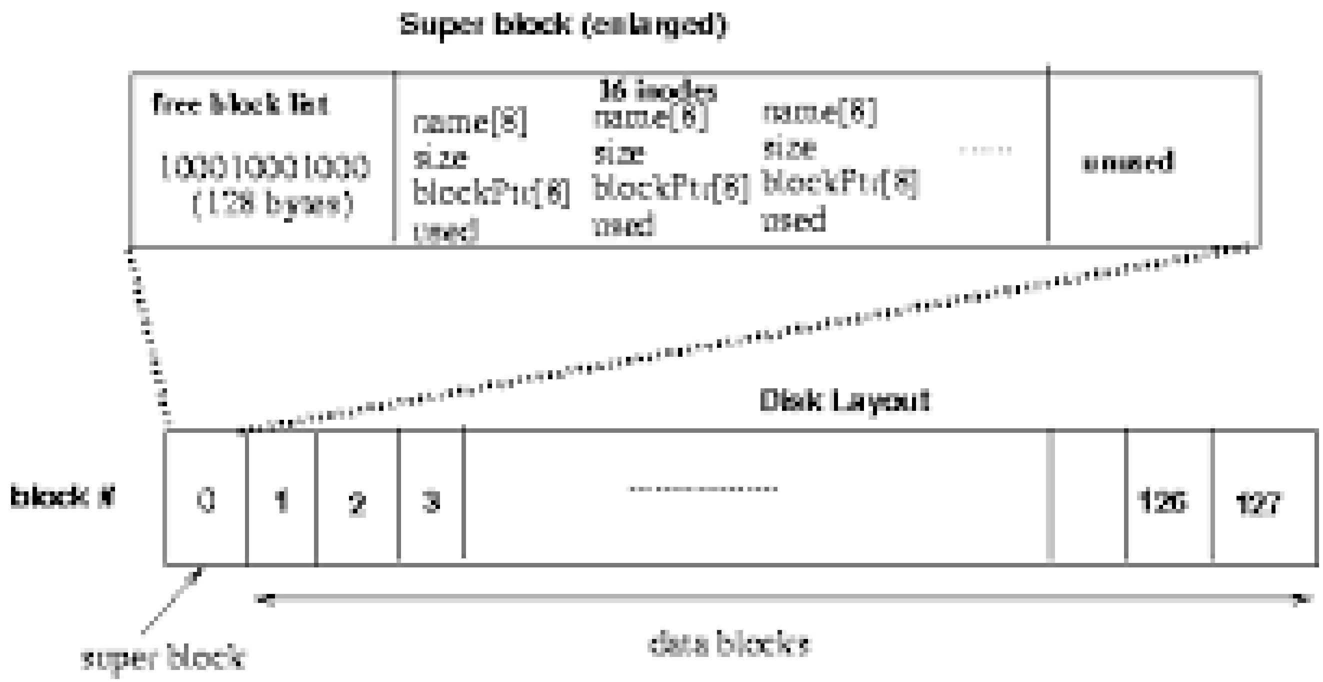
- The MBR program reads in and executes the code in the MBR
- This first thing that is determined is the active partition
  - The first block of the active partition is read in (boot block)
  - This program loads the OS contained in that partition

File System Layout-GParted

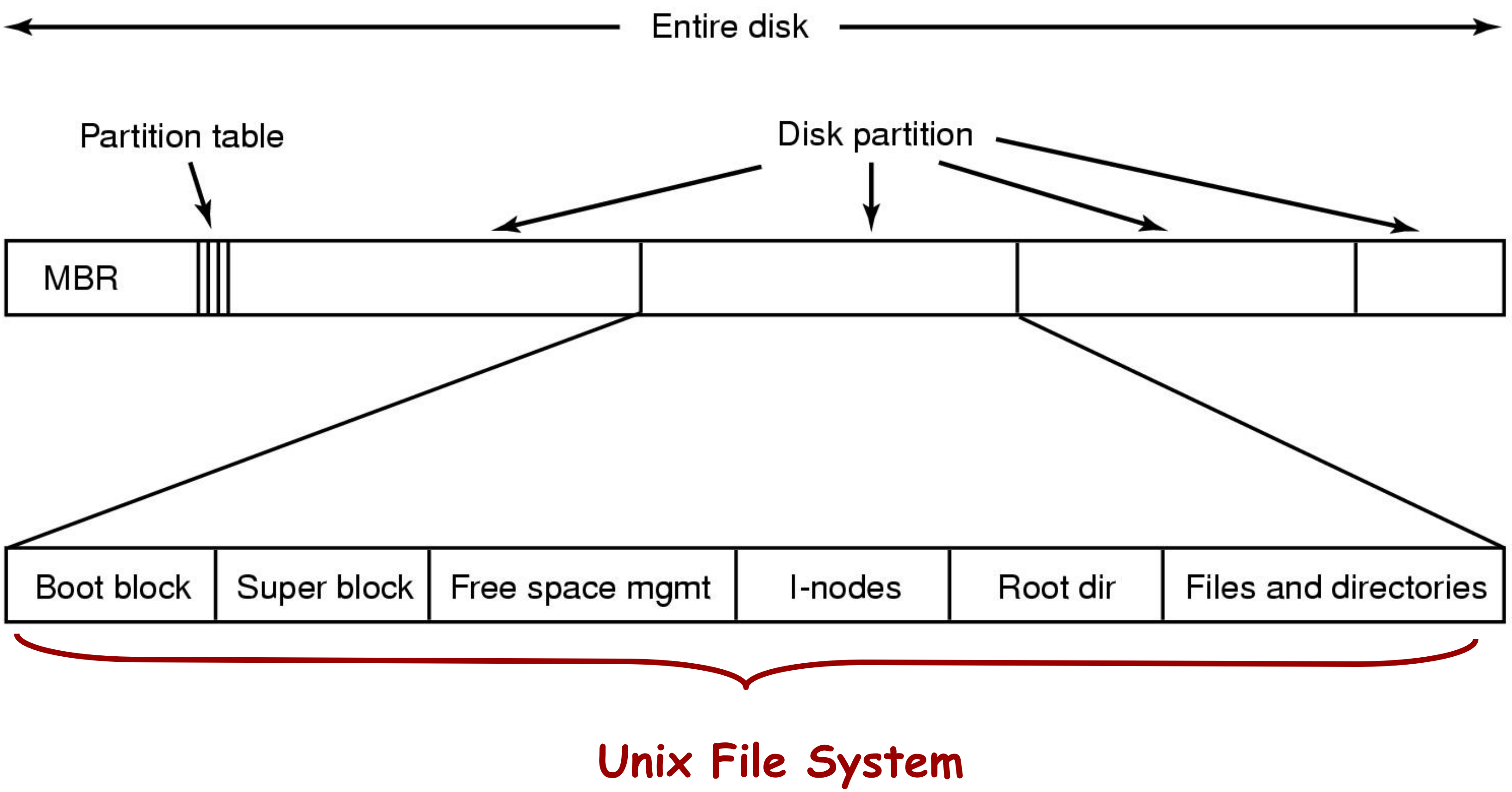


File System Layout





File System Layout



File Bytes vs Disk Sectors

- Files are sequences of bytes
  - Granularity of file I/O is byte
- Disks are arrays of sectors (512 bytes)
  - Granularity of disk I/O is sector
  - File data must be stored in sectors

File Bytes vs Disk Sectors

- File systems may also define a block size
  - Block size usually consists of a number of sectors
  - Contiguous sectors are allocated to a block
- File systems view the disk as an array of blocks
  - Must allocate blocks to file
  - Must manage free space on disk

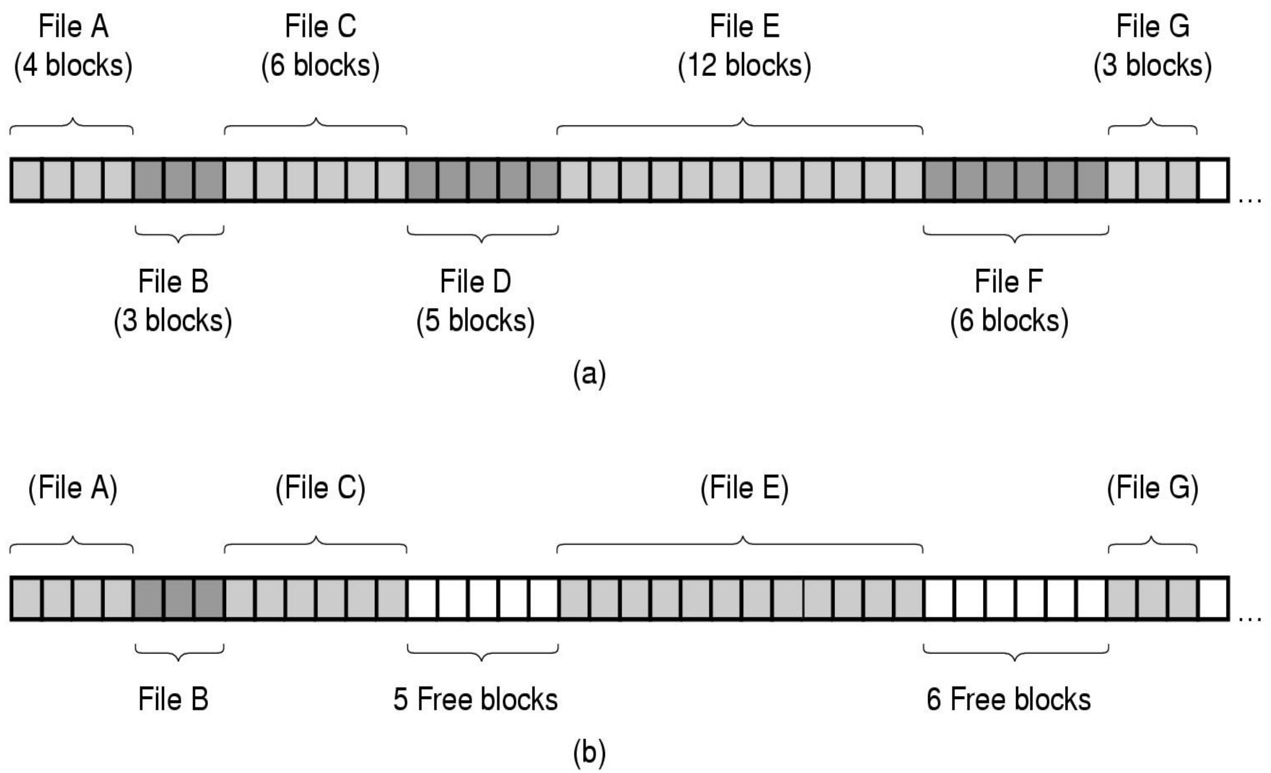
File System Implementation

- Approaches to allocating blocks to a file
  - Contiguous Allocation
  - Linked List Allocation
  - Linked List Allocation using Index
    - FAT used by WINDOWS
  - I-nodes
    - Used by UNIX

Contiguous Allocation

- Store each file as a contiguous run of disk blocks
- Assume a disk of 1-KB blocks
  - 50 KB file is allocated 50 consecutive blocks
- Advantages:
  - Easy to implement
    - Two numbers needed for each file: disk address of the first block and the number of blocks in the file
  - Read performance is excellent

Contiguous Allocation



(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files *D* and *E* have been removed

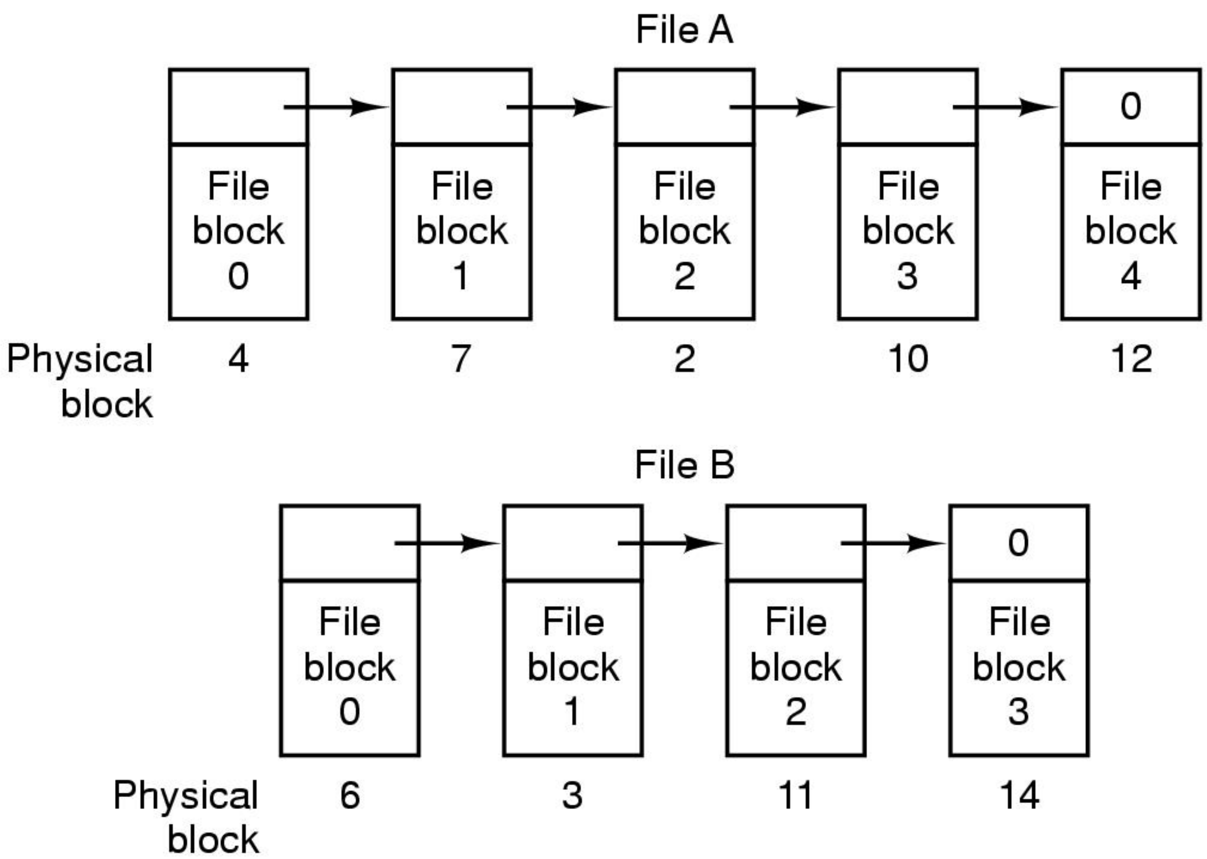
Contiguous Allocation

- Disadvantages
  - Fragmentation
  - Will need periodic compaction (time-consuming)
  - Will need to manage free lists
  - If new file is put at end of disk
    - No problem
  - If new file is put into a "hole"...
    - Have to know a file's maximum possible size ... at the time it is created!

Contiguous allocation

- Good for CD-ROMs, DVDs
  - All file sizes are known in advance
  - Files are never deleted

Linked List Allocation



Storing a file as a linked list of disk blocks

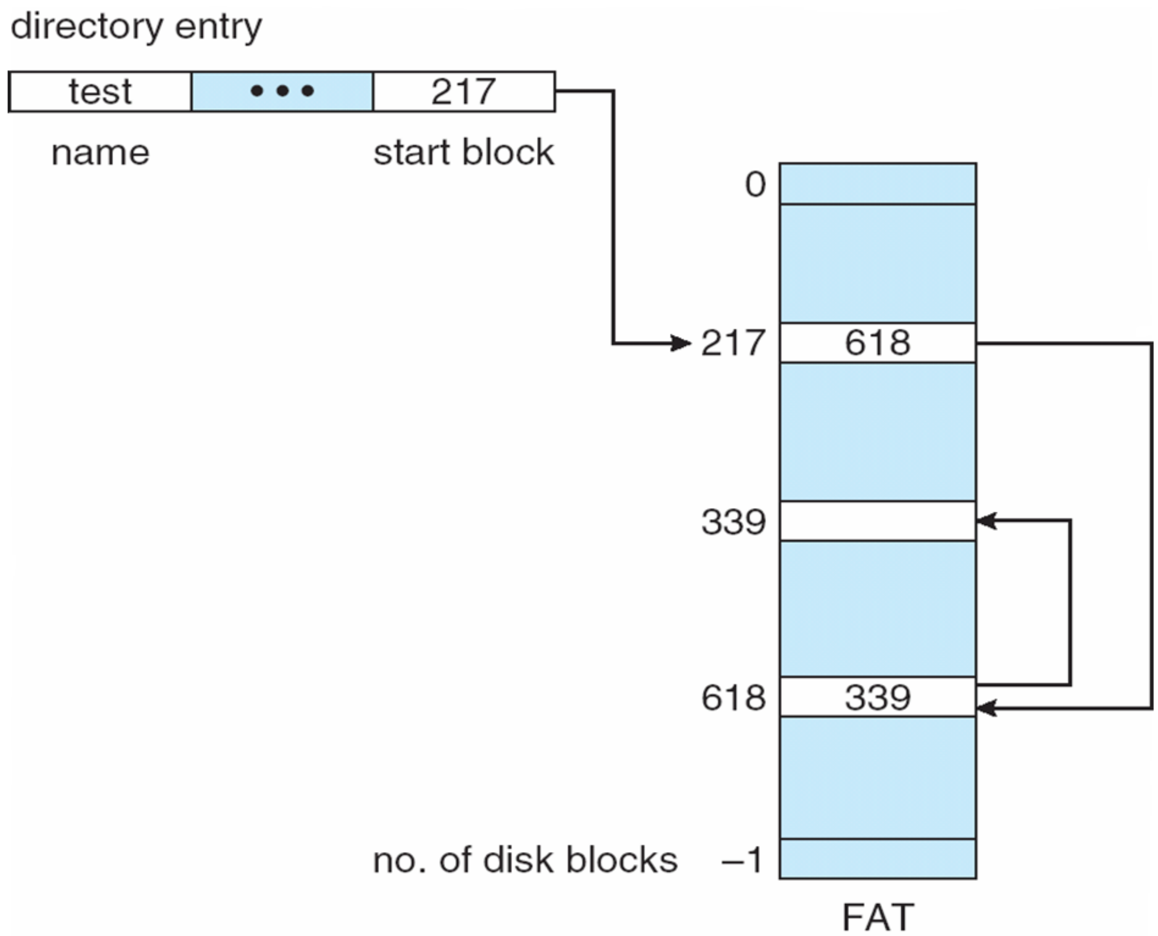
Linked List Allocation

- Advantage
  - No fragmentation (except internal fragmentation)
- Disadvantage
  - Random access is slow
    - To get to block n, the operating system has to start at the beginning and read the n-1 blocks prior to it

Linked List Allocation Using Index

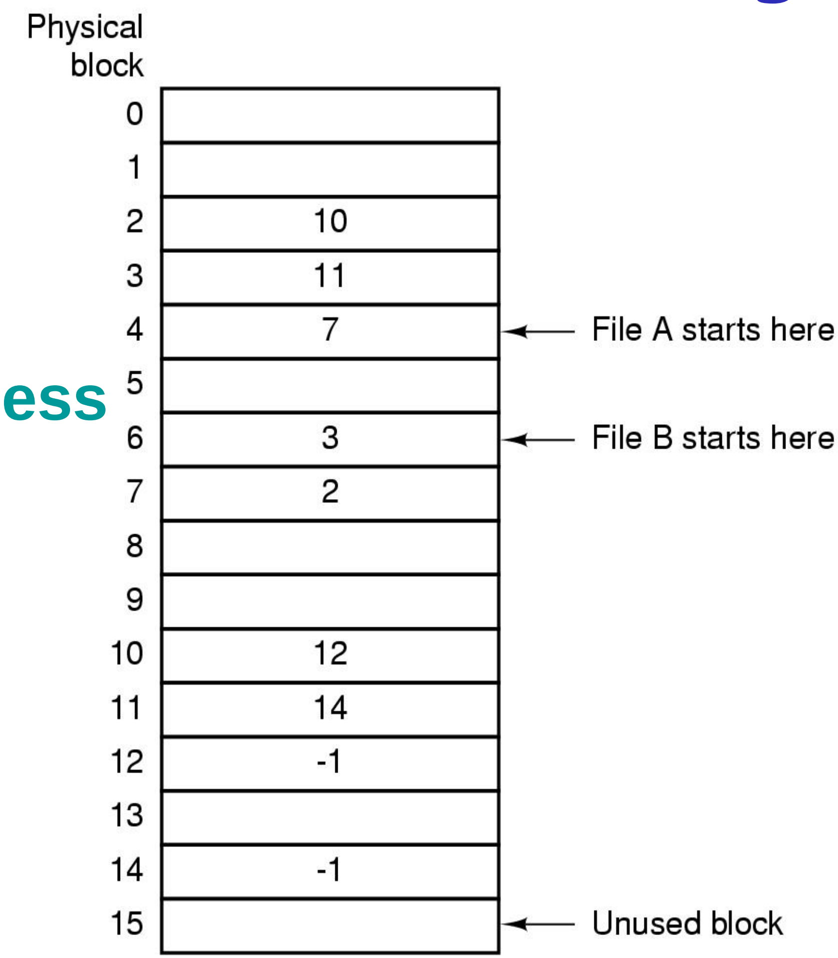
- Take the pointer word from each disk block and put it in a table in memory
- The figure on the next page is based on the previous figure
- The chains are terminated with a special marker (-1).
- The table in main memory is called a FAT (File Allocation Table)

File-Allocation Table



Linked List Allocation using Index

Fast Random Access

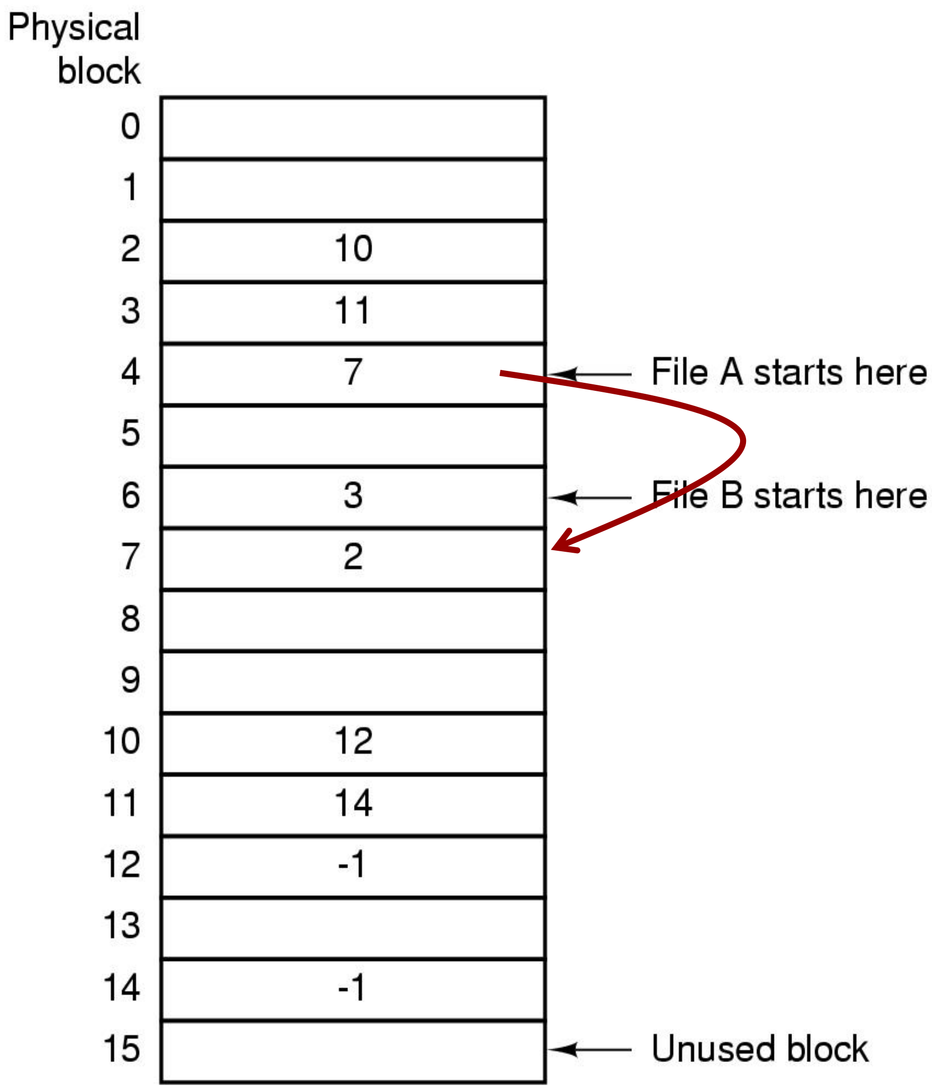


Linked list allocation using a file allocation table in RAM

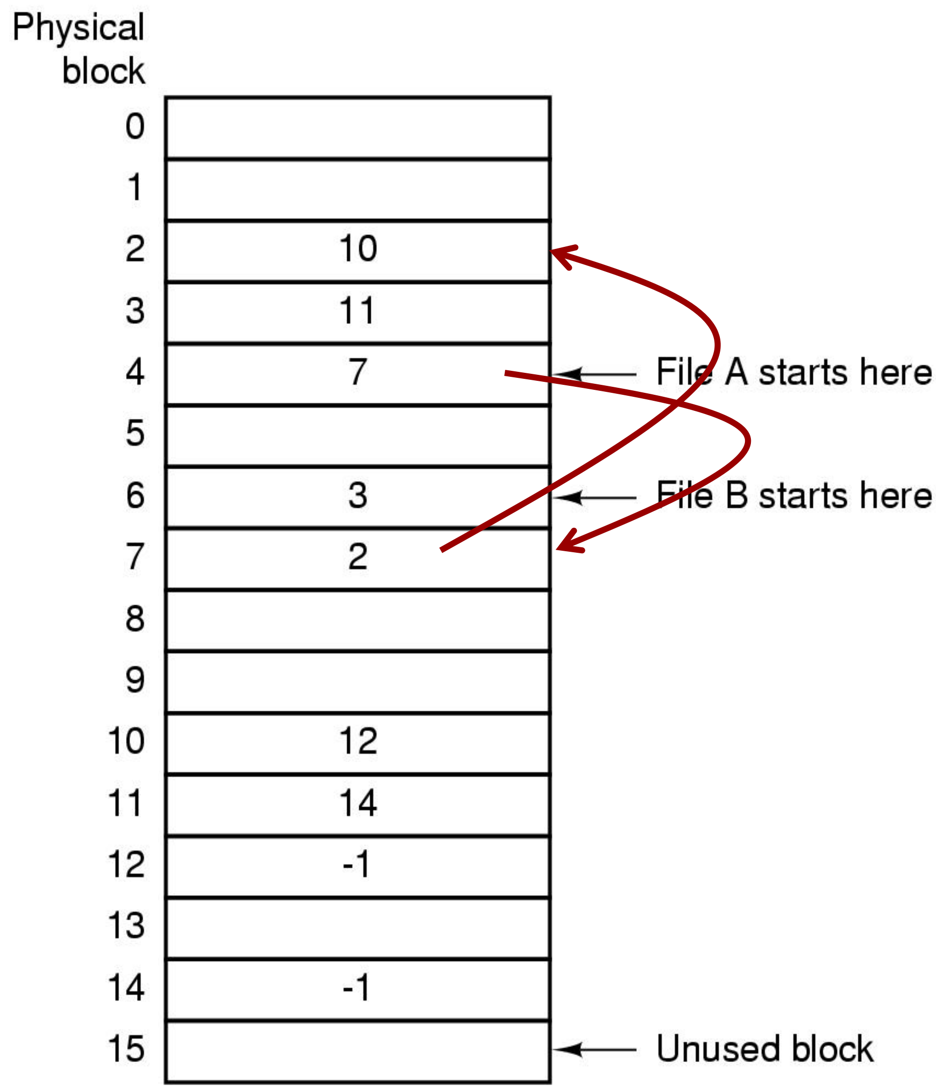
File allocation table (FAT)

Physical block		
0		
1		
2	10	
3	11	
4	7	File A starts here
5		
6	3	File B starts here
7	2	
8		
9		
10	12	
11	14	
12	-1	
13		
14	-1	
15		Unused block

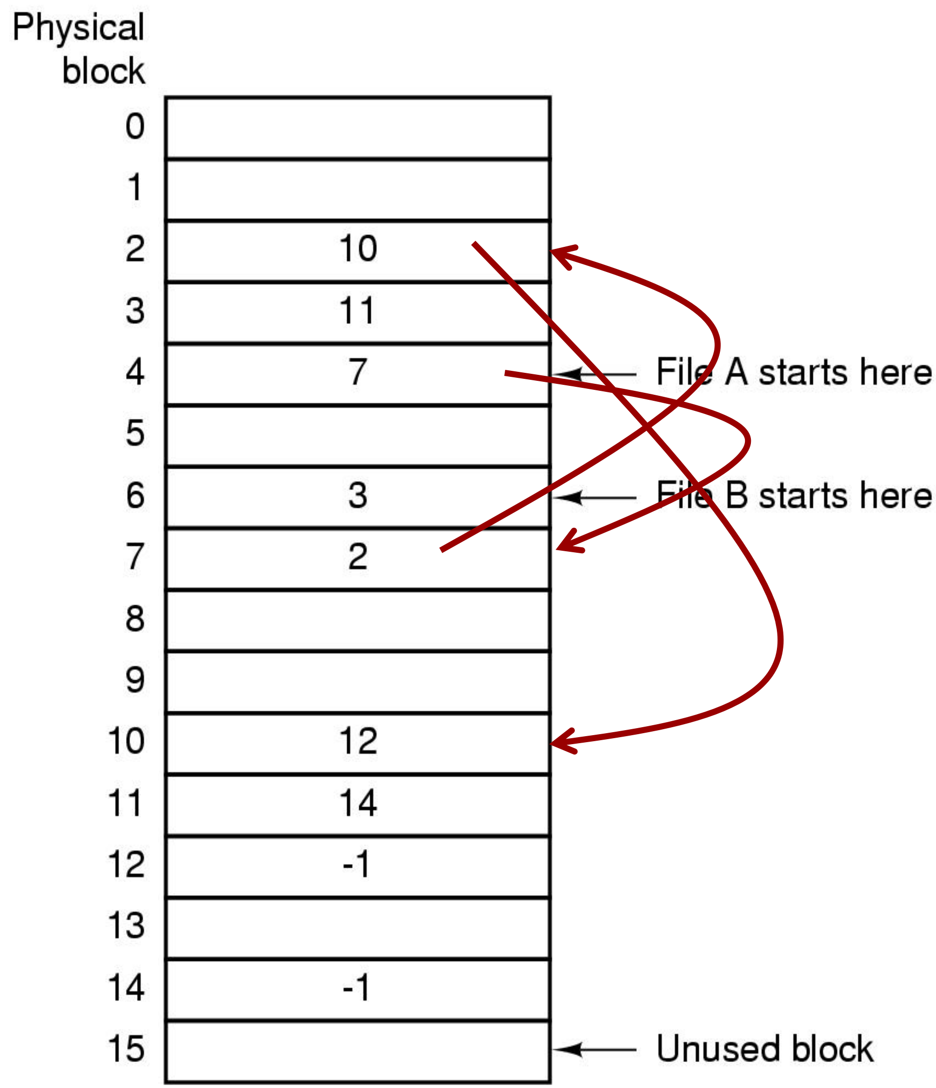
File allocation table (FAT)



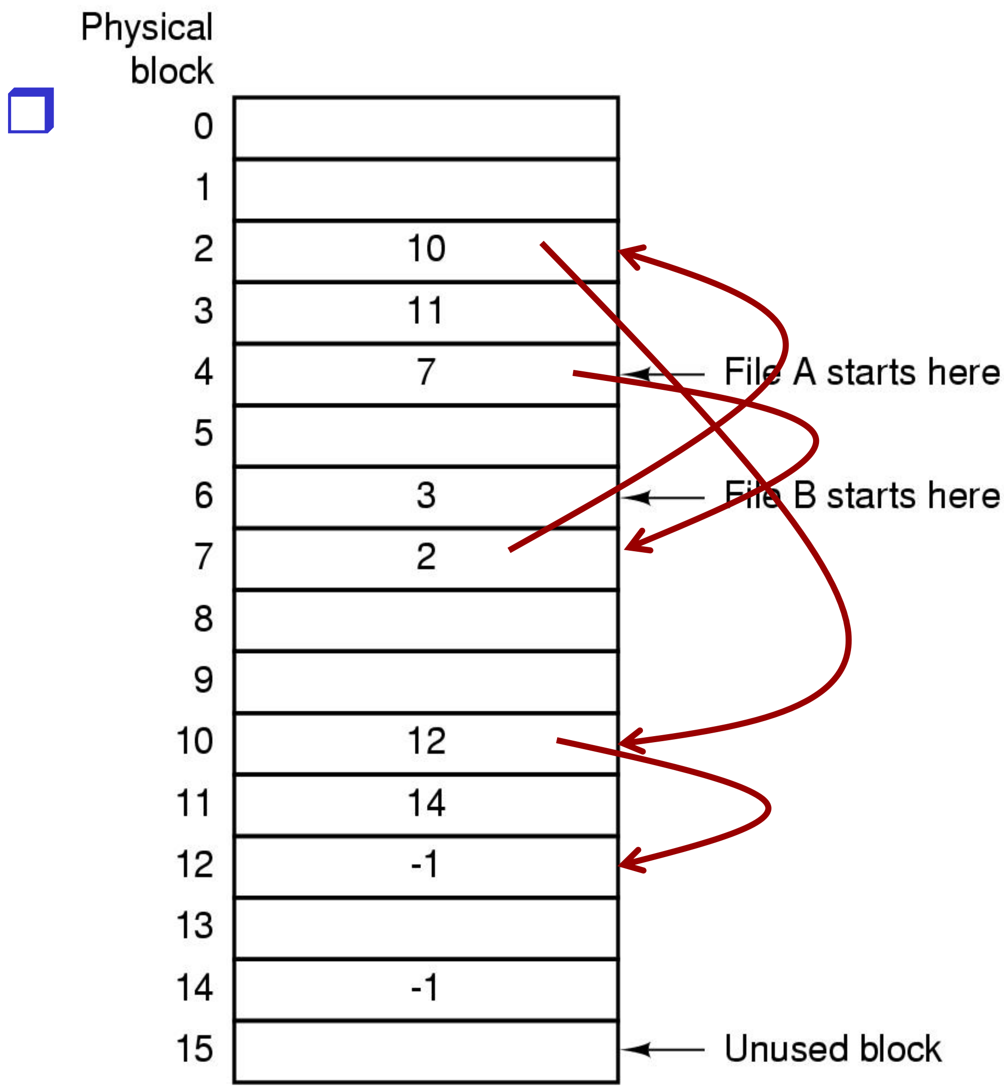
File allocation table (FAT)



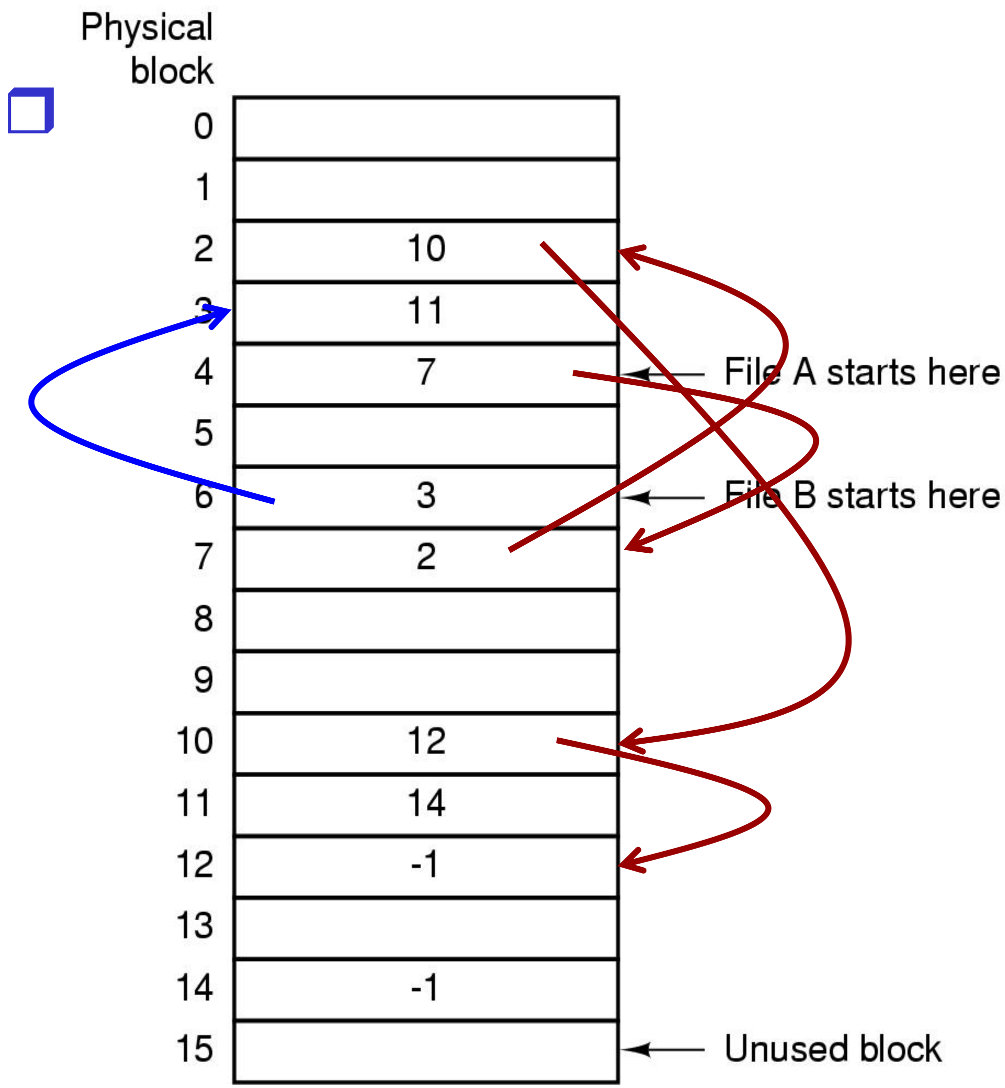
File allocation table (FAT)



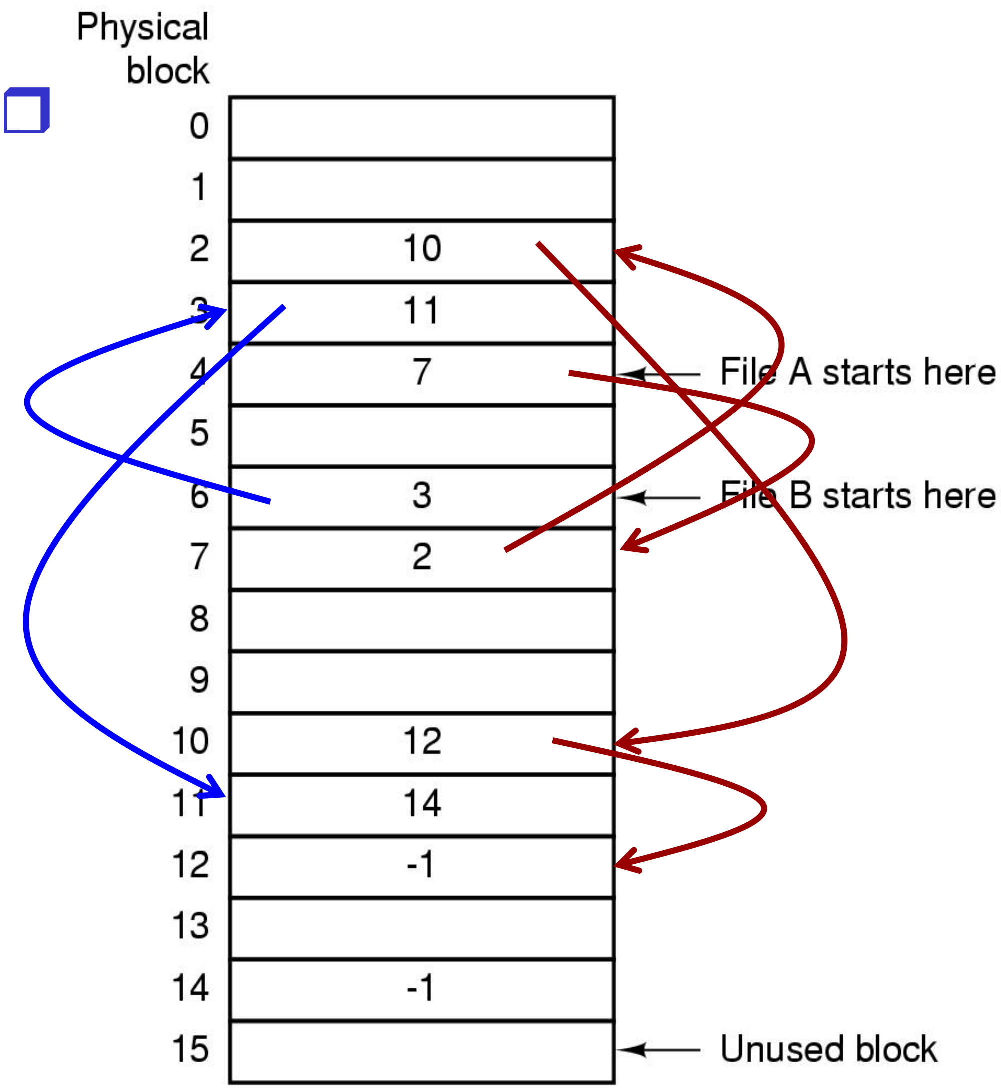
File allocation table (FAT)



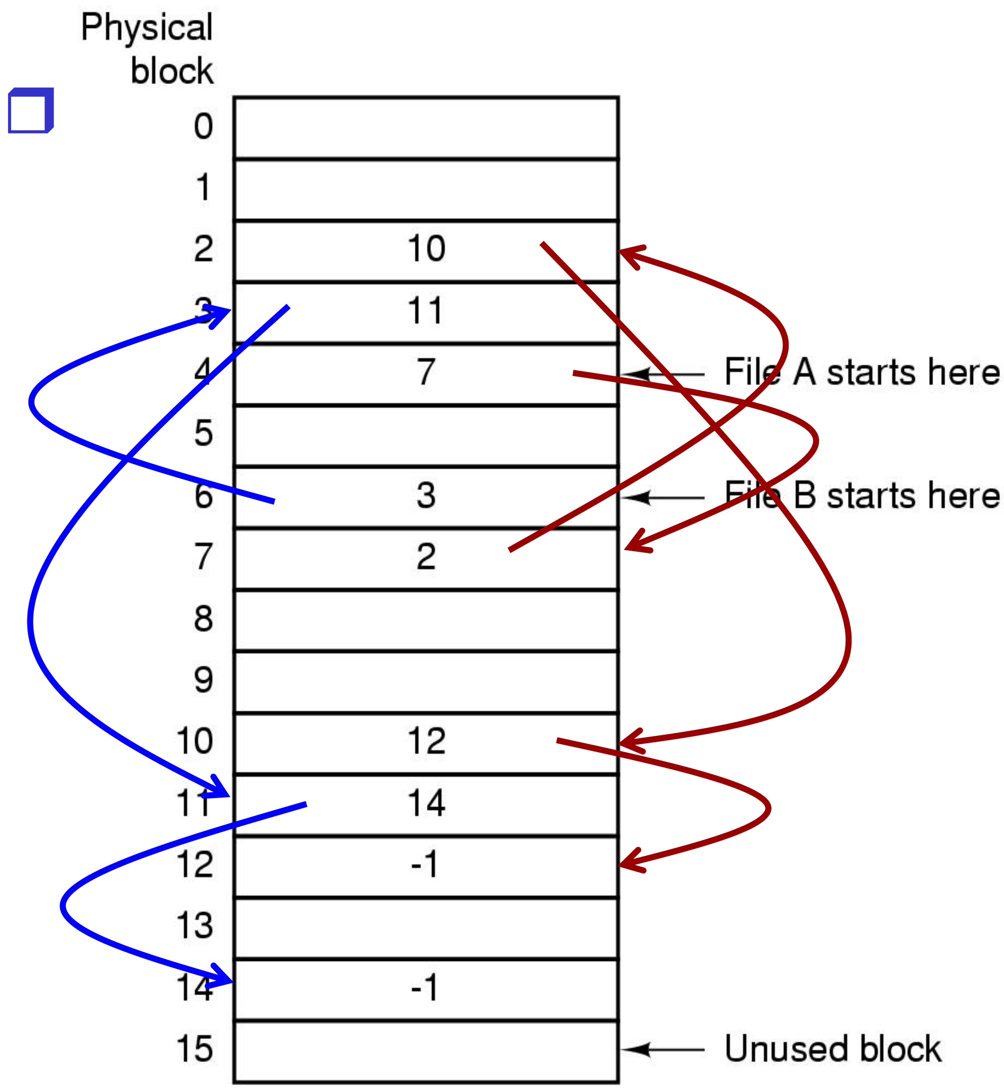
File allocation table (FAT)



File allocation table (FAT)



File allocation table (FAT)



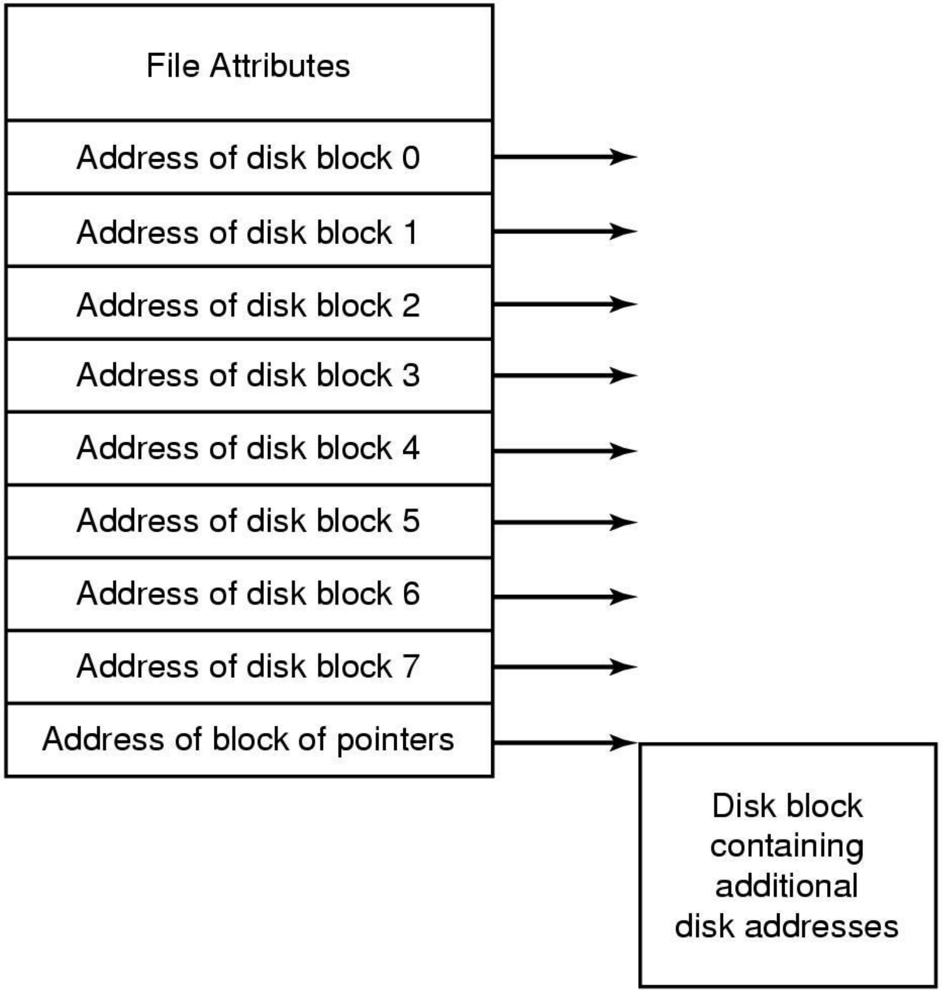
Linked List Allocation Using Index

- Advantage
  - Chain must still be followed to find a given offset within the file, but the chain is entirely in memory
    - No disk references are needed
- Disadvantage
  - Entire table must be in memory all the time
  - What if you have a 20-GB disk and a 1-KB block size?
    - The table needs 20 million entries, one for each of the 20 million disk block
    - Each entry is a minimum of 3 bytes which means 60 MB of memory
- This technique was used in MS-DOS and Windows-98

I-node

- Associate with each file a data structure called an i-node which maintains this information:
  - File attributes
  - Disk addresses of the file's blocks
- Given an i-node it is possible to find all the blocks of the file
- Fixed size
- Unix systems use i-nodes

I-node

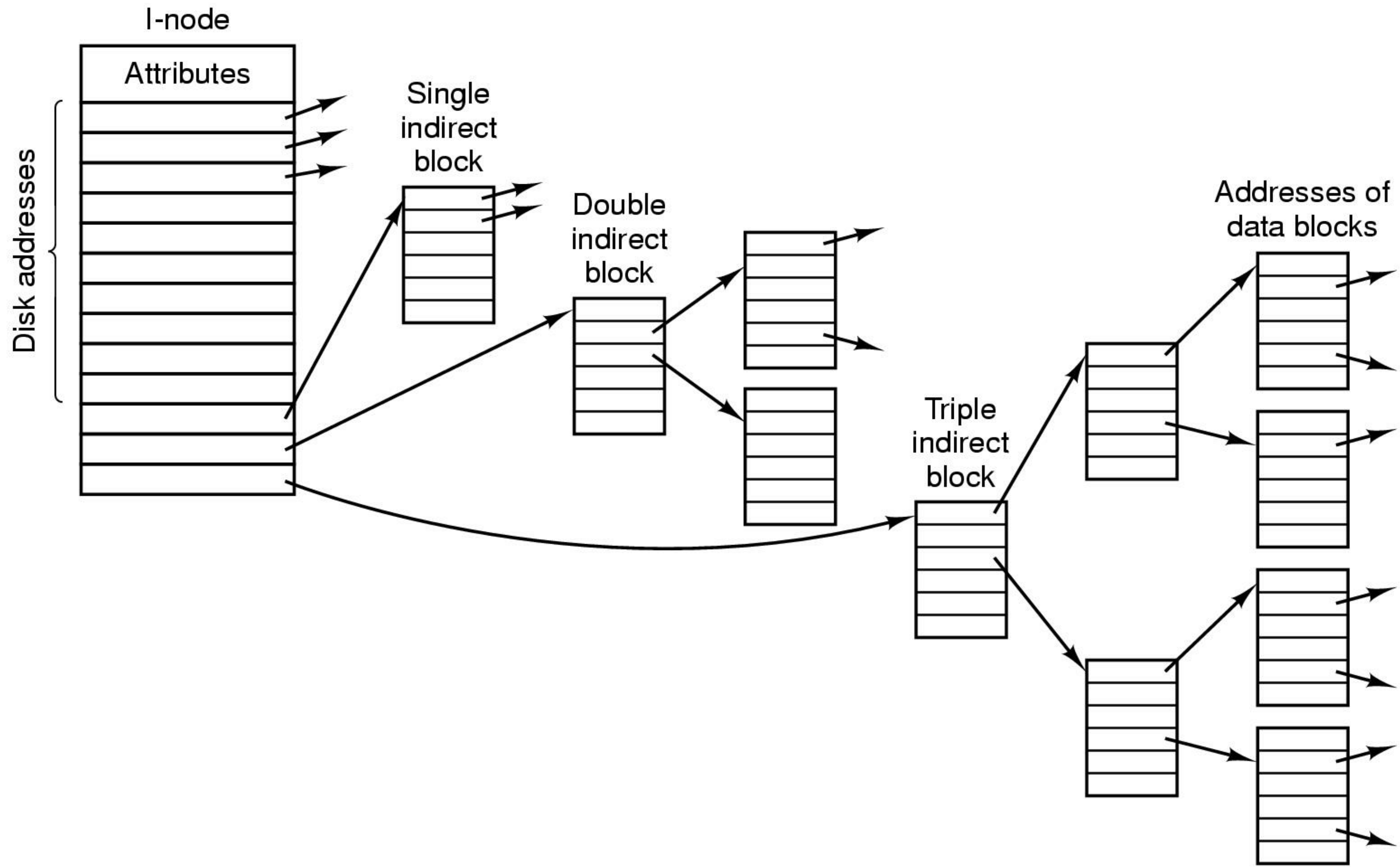


An example i-node

I-node

- What happens when a file needs more blocks?
  - Reserve the last disk address for the address of a block containing more disk block addresses

I-node



A UNIX i-node

The UNIX I-node entries

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

Structure of an I-Node: Attributes and addresses of disk blocks

I-node

- Advantage
  - Only the i-node needs to be in memory when the corresponding file is open
- Disadvantage
  - Updating the structure is more complex

Implementing Directories

- Same as files but containing an array of 16-byte directory entries where each entry consists of the following: file name, inode number
  - The file name can refer to another directory

Implementing Directories

- Opening a file requires that the OS needs the pathname
- The OS uses the pathname supplied by the user to locate the directory entry
- How can we locate the root directory (the start of all paths)?

Implementing Directories

- In Unix systems
  - *Superblock* is a record of the characteristics of a filesystem, including its size, the *block* size, the empty and the filled blocks and their respective counts. It is file system metadata.
  - The superblock (among other things) has the location of the i-node which represents the root directory.
- Once the root directory is located a search through the directory tree finds the desired directory entry
- The directory entry provides the information needed to find the disk blocks for the requested file

Entry Lookup

- When a file is opened, the file system must take the file name supplied and locate its disk blocks
- Let's see how this is done for the path name */usr/ast/mbox*

Entry Lookup

- First the system locates the root directory
  - There is information for each file and directory within the root directory
  - A directory entry consists of the file name and i-node number
- The file system looks up the first component of the path, *usr*, to find the i-node for */usr* which is i-node 6
- From this i-node the system locates the directory for */usr/* which is in block 132

Entry Lookup

- The system then searches for *ast* within the */usr* directory which is block 132
- The entry gives the i-node for */usr/ast/*
- From this i-node the system can find the directory itself and lookup *mbox*
  - The i-node for this file is then read into memory and kept there until the file is closed

Looking up for an entry

Root directory

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

Looking up usr yields i-node 6

I-node 6 is for /usr

Mode size times
132

I-node 6 says that /usr is in block 132

Block 132 is /usr directory

6	•
1	••
19	dick
30	erik
51	jim
26	ast
45	bal

/usr/ast is i-node 26

I-node 26 is for /usr/ast

Mode size times
406

I-node 26 says that /usr/ast is in block 406

Block 406 is /usr/ast directory

26	•
6	••
64	grants
92	books
60	mbox
81	minix
17	src

/usr/ast/mbox is i-node 60

The steps in looking up /usr/ast/mbox

Summary

- We have examined how files are implemented and how one particular implementation is used to support some of the file operations.