File Systems

COMP 3361: Operating Systems I

Winter 2015

http://www.cs.du.edu/3361

The World of Abstractions

- CPU, memory and disk are three of the most important shared resources in a computer
- Process for CPU abstraction
  - user programs do not have to know about the presence of other programs
- Logical memory for memory abstraction
  - user programs do not have to know where in physical memory are they located
- **Files for disk abstraction**
  - users do not have to know where data resides on disk

Basic Disk Access

- Think of disk as a linear array of fixed size **blocks**
  - block size (typically 4KB) is a parameter of the file system implementation
- Two operations
  - read from block number k
  - write to block number k
- But then,
  - how do you find information?
  - how do you keep one user from reading another user’s data?
  - how do you know which blocks are free?

File Concept

- A **file** is a logical unit of information on the disk
  - an abstraction from the physical properties of a storage device (do not care ***how*** the actual information is stored on disk)
  - viewed as a contiguous entity (do not care ***where*** the actual information is stored on disk)

- Files are persistent
- Files are managed by an operating system
  - their storage location on disk, naming, discovery, access rights, …
  - the details of how files are organized and managed on disk is a **file system**

User Interface for Files

- **File naming**: how users refer to files
  - a name of variable length
  - an extension to help remember what is in the file
- **File structure**: how is data organized inside a file
  - sequence of bytes: only application programs can interpret the meaning of those bytes
  - sequence of records or tree of variable length records
- **File types**: categorizing what is in a file
  - ASCII files, binary files, directories, links, character/block special files

User Interface for Files

- **File access**: how can files be accessed
  - sequential access or random access
- **File attributes**: metadata on files
  - size, access rights, protection, creation/modifed/accessed times, etc.
- **File operations**: actions allowed on files
  - create, delete, open, close, read, write, append, seek, get attributes, set attributes, rename

6

What is a Directory?

- A special type of file
  - keeps track of other files
  - helps the user manage other files

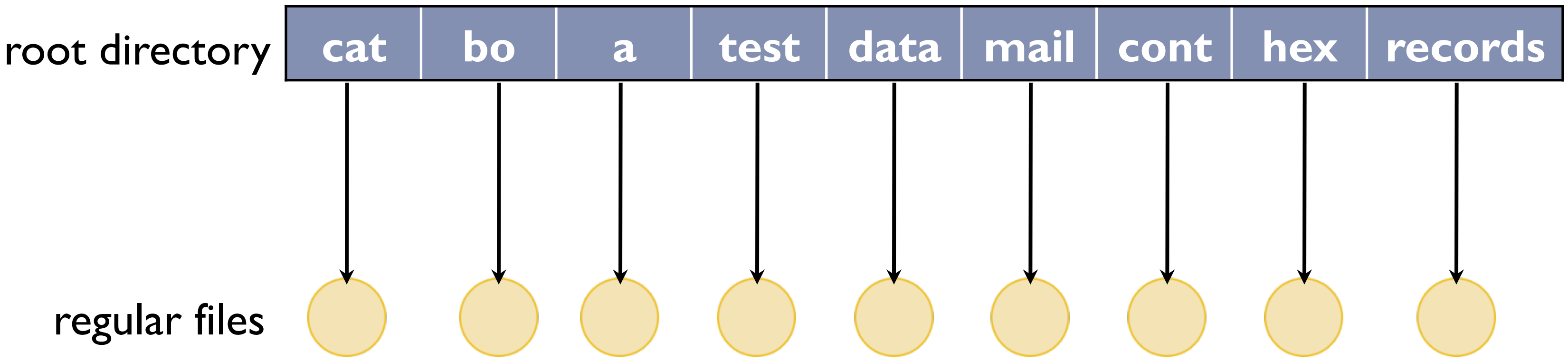
- OS graphical interface shows files and directories differently

- OS knows know how to read the contents of a directory

- Single-level: only one directory in system
- Hierarchical: multiple level

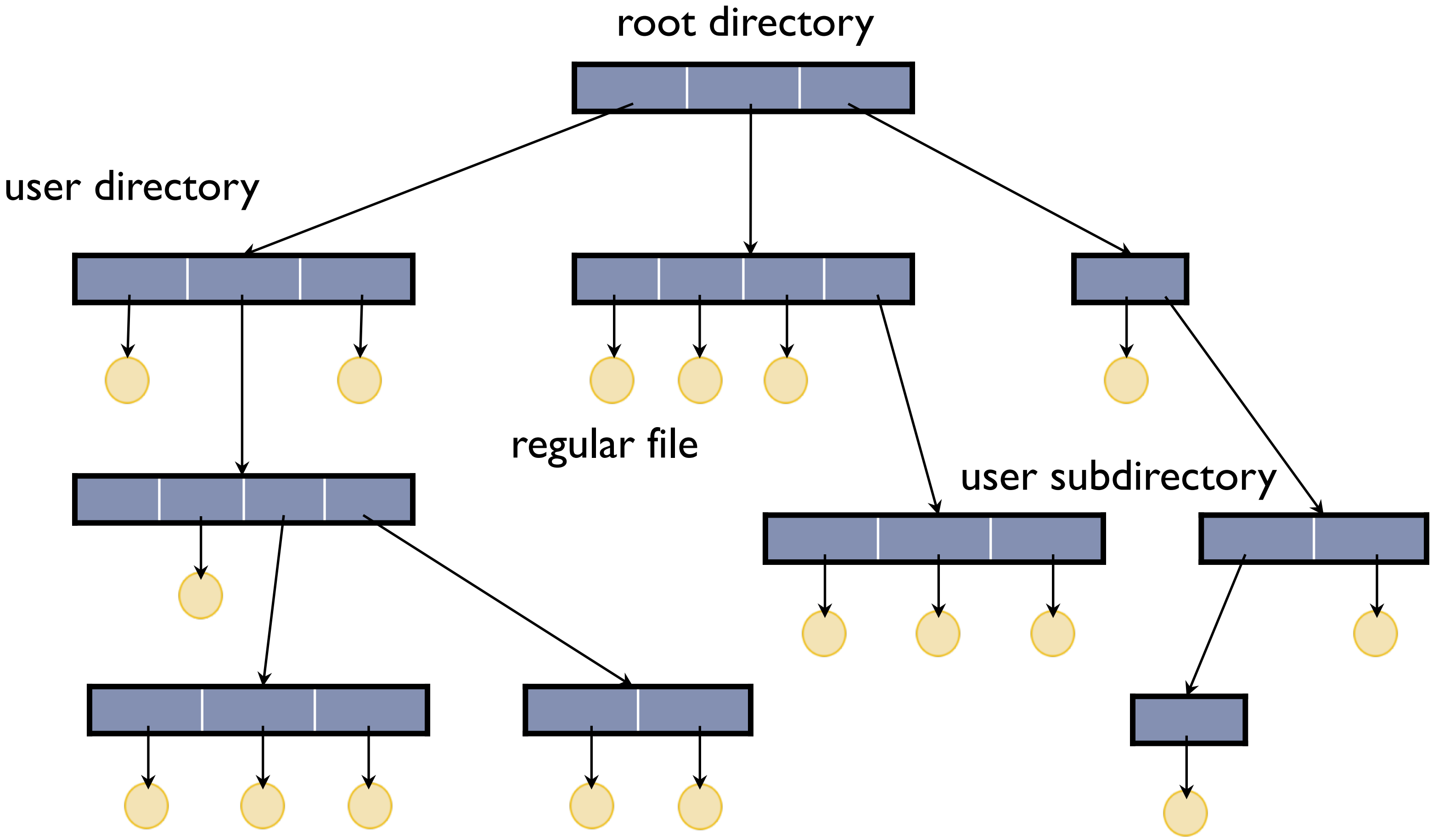
Single-Level Directory

- A single directory in the system



8

Hierarchical Directory Structure



path: the sequence of directories leading to a file
absolute path: sequence begins at root directory
relative path: sequence begins at another directory (current working directory)

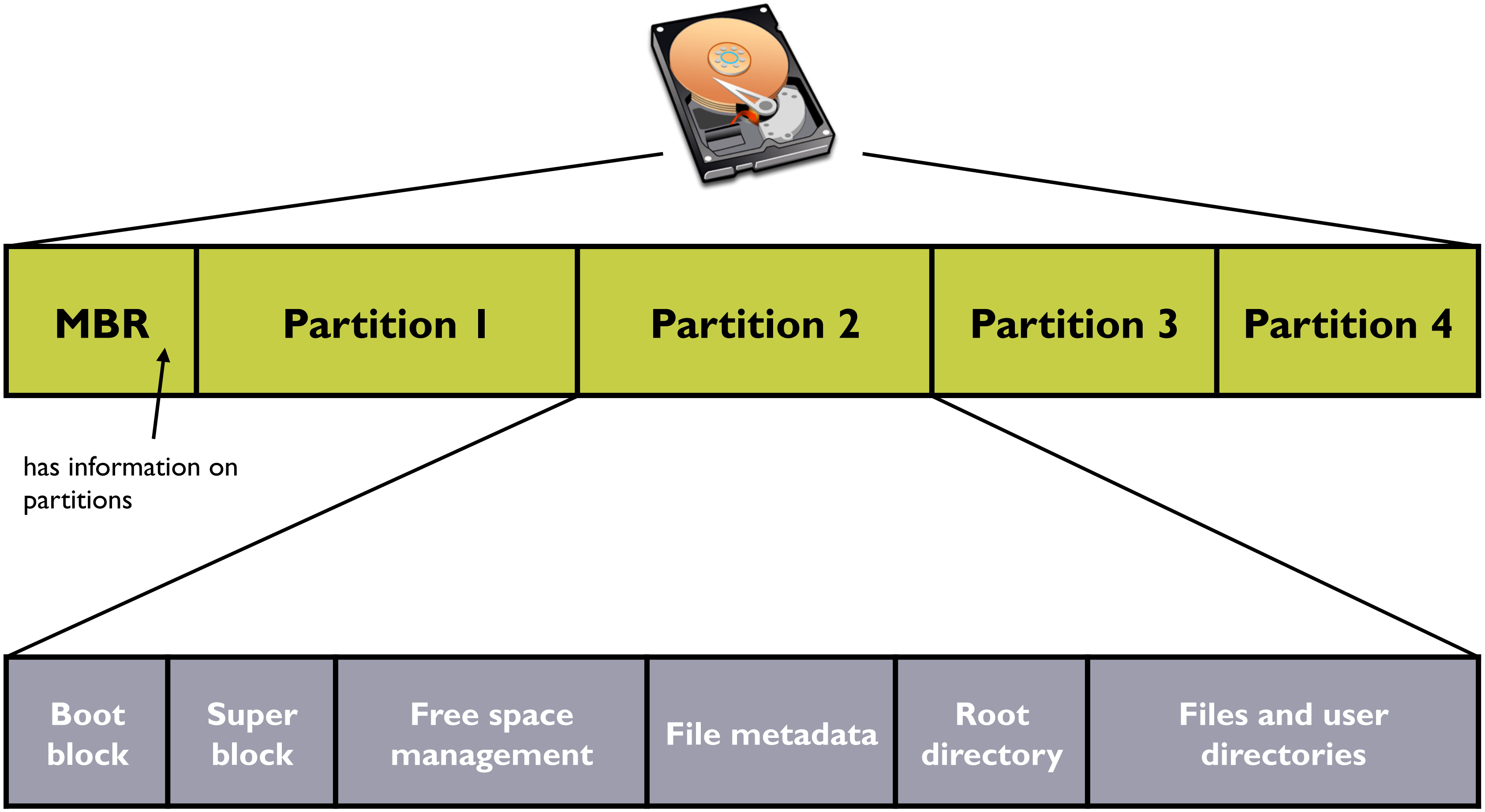
Directory Operations

- Same like files
  - create, delete, opendir, closedir, readdir, rename, **link, unlink**

- **Symbolic link**
  - a special file that contains the path of another file
  - space allocated to store the pathname
  - OS reads the path information to reach the real file

- **Hard link**
  - no space allocated
  - maintain a reference count for the file pointed to by the links

File System Layout



MBR | Partition 1 | Partition 2 | Partition 3 | Partition 4

has information on partitions

Boot block | Super block | Free space management | File metadata | Root directory | Files and user directories

File System Components

- **Boot block**: contains information needed by the system to boot an OS in the partition, if any
- **Superblock**: contains parameter information on the layout
  - number of blocks in the partition, size of blocks, free-block count, location of root directory, …
  - called a *master file table* in Windows NTFS
- **Free space management**: tracking free blocks
- **File metadata**: which blocks go with which file, and other file attributes
- **Root directory**: files/directory in root directory

12

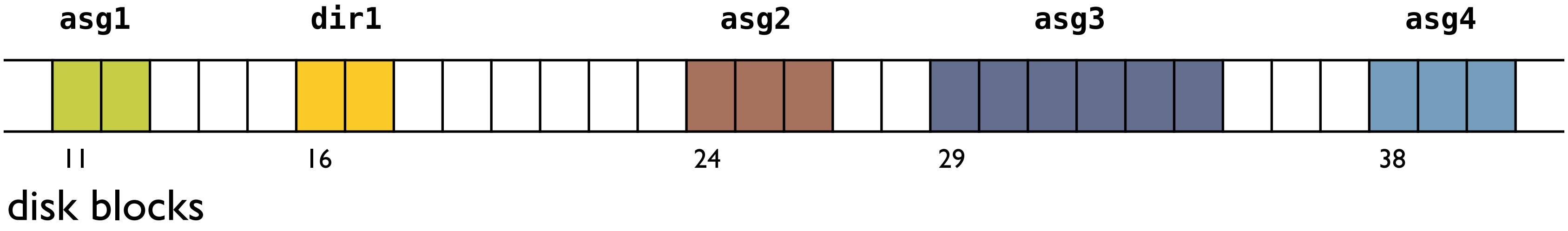
File-to-Block Mapping

- How to track which blocks belong to a file?
- **Contiguous allocation**
  - files occupy contiguous blocks on disk
- **Linked allocation**
  - each file is a linked list of blocks
- **Indexed allocation**
  - maintain an array of disk-block addresses

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Only starting location (block number) and length (number of blocks) are required
- Random access
- Dynamic storage-allocation problem
  - external fragmentation
- How to handle the growth of files?

Contiguous Allocation Example



File metadata

	ID	start block	length	other attributes
asg1	1	11	2	…
asg2	2	24	3	…
asg3	3	29	6	…
asg4	4	38	3	…
dir1	5	16	2	…

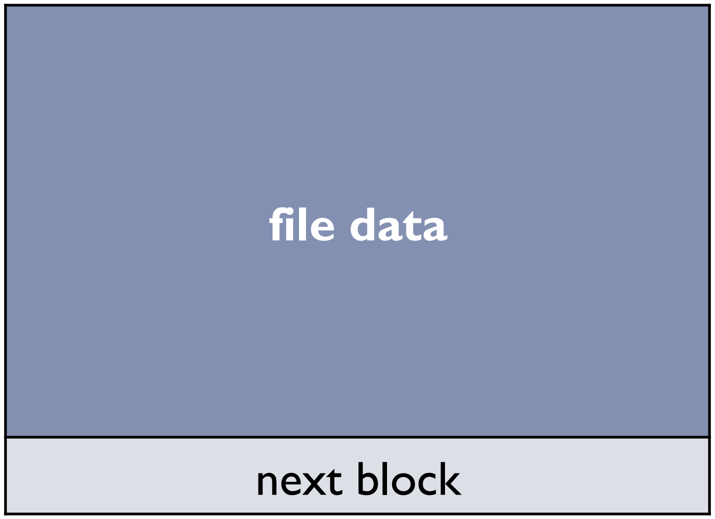
15

Extent Based Systems

- A modified contiguous allocation scheme is used to handle problems due to file size increase
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
  - extents are allocated for file allocation
  - a file consists of one or more extents

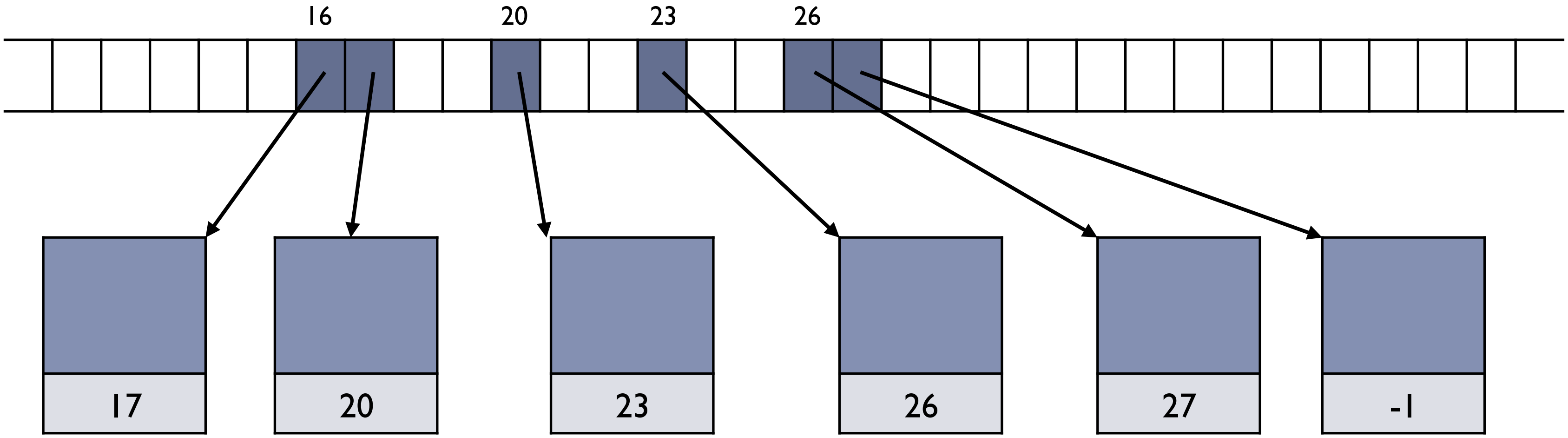
Linked Allocation

- Part of each disk block stores the address of the file's next block
  - blocks may be scattered anywhere on the disk



structure of a block

Linked Allocation Example



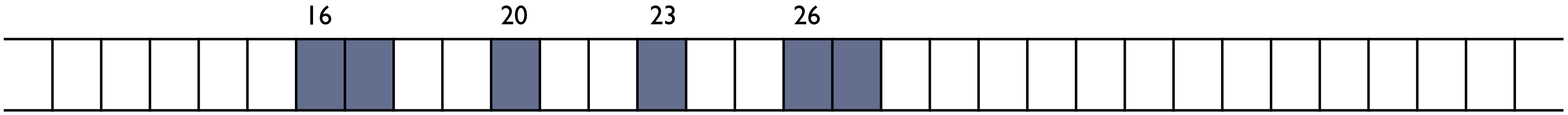
	ID	start block	other attributes
...		...	...
asg	3	16	...
...		...	...

Linked Allocation

- Need only starting address (block) of a file
- No external fragmentation
- No random access
  - the n^{th} block can be reached only by traversing the (n-1) previous blocks of the file
- Part of the space in each block is taken by the pointer
- What happens if one of the file blocks is corrupted?

File Allocation Table (FAT)

- Reserve a section of the disk to contain a table of block addresses
  - each entry in the table indicates where the next block of data for the file can be found



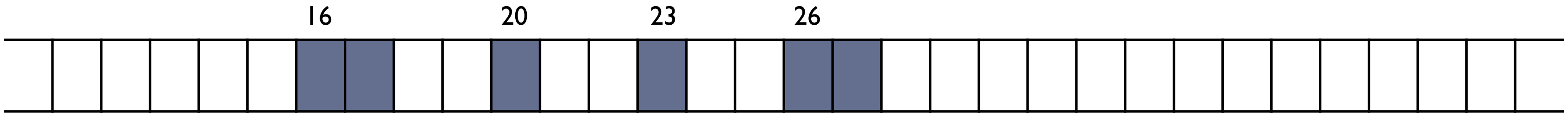
File Allocation Table

						17	20		
23			26			27	-1		

	ID	start block	other attributes
...		...	...
asg	3	16	...
...		...	...

Indexed Allocation

- Store all disk blocks of a file in one place
  - in this case, each row in the table below is called an **index-node (i-node)** and the table is called the **i-node table**



	ID	file blocks	other attributes
...		…	…
asg	3	16,17,20,23,26,27	…
...		…	…

Index Block for Growing Files

file **asg** blocks: 16,17,20,23,26,27,28,32,33,34,35,37,38,39,40

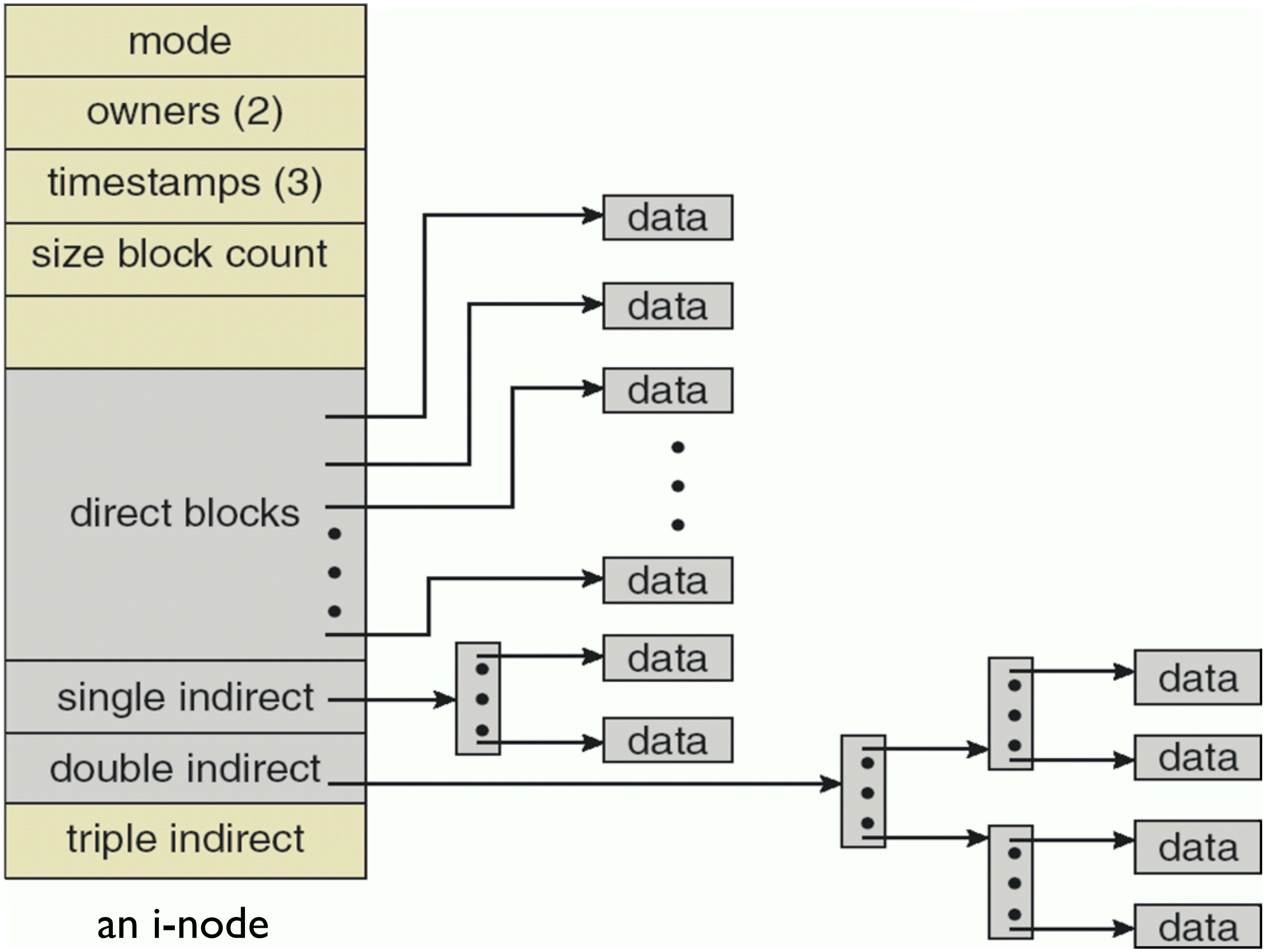
	ID	file blocks (5 max)	other attributes
…		…	…
asg	3	16,17,20,23,**150**	…
…		…	…

block 150

26,27,28,32,33,34,35
,37,38,39,40,-1

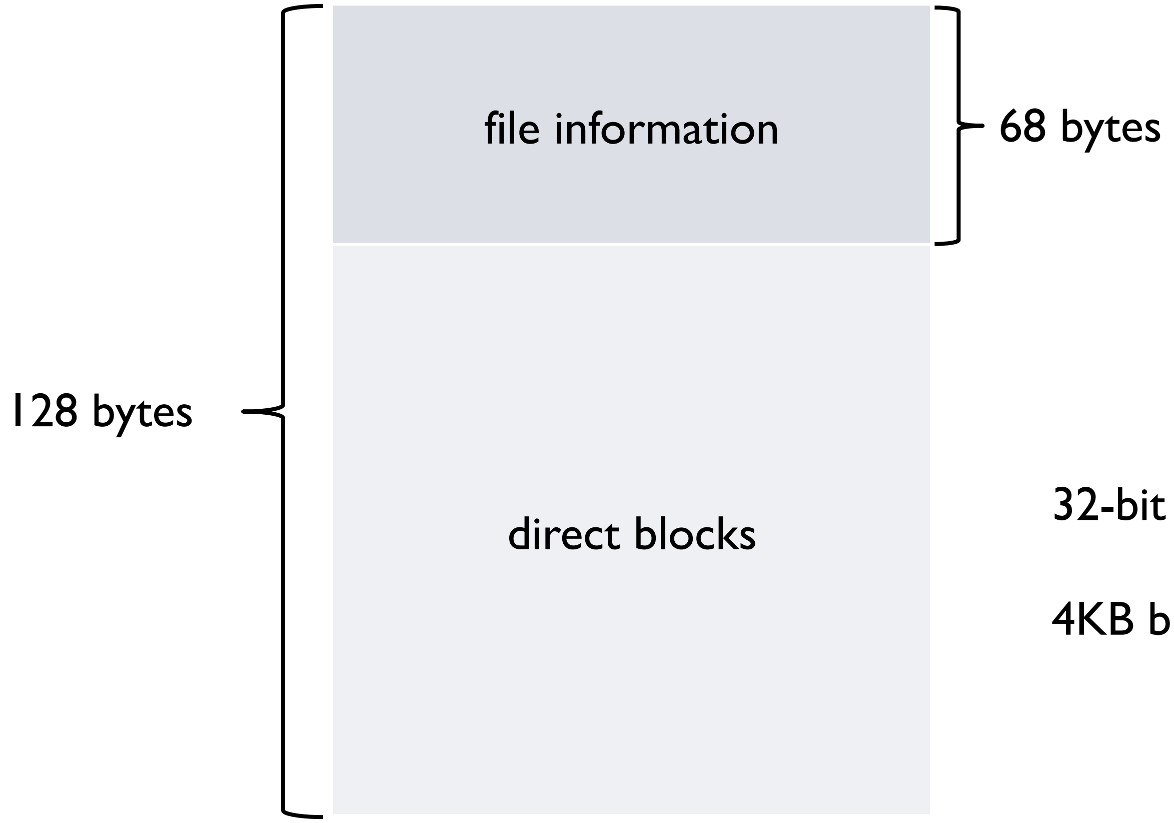
last (fifth here) block is **index block**: block containing remaining blocks of file

Using Multilevel Index Blocks



an i-node

What is the Maximum Size of a File?



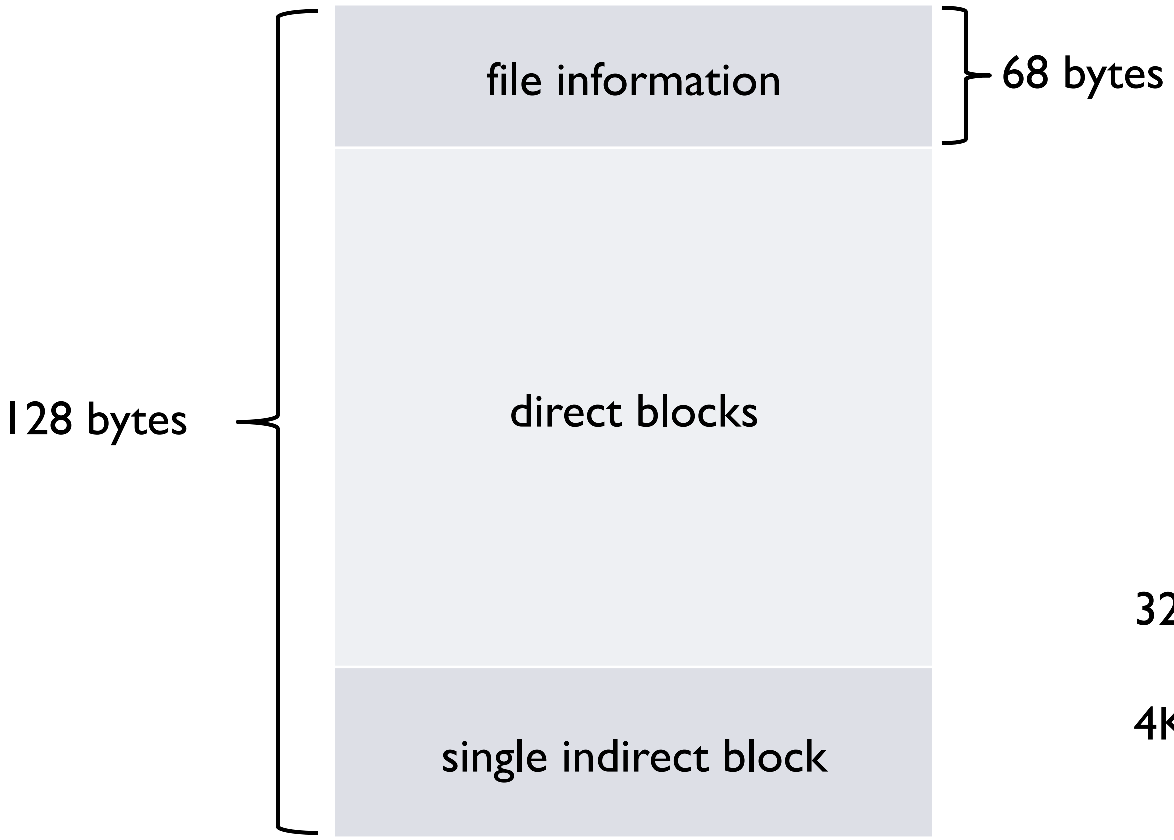
32-bit (4 byte) block addresses

4KB blocks

(128-68) = 60 bytes for direct blocks = (60 / 4) direct block addresses
= 15 direct block addresses

Maximum file size = 15 x 4KB = 60KB

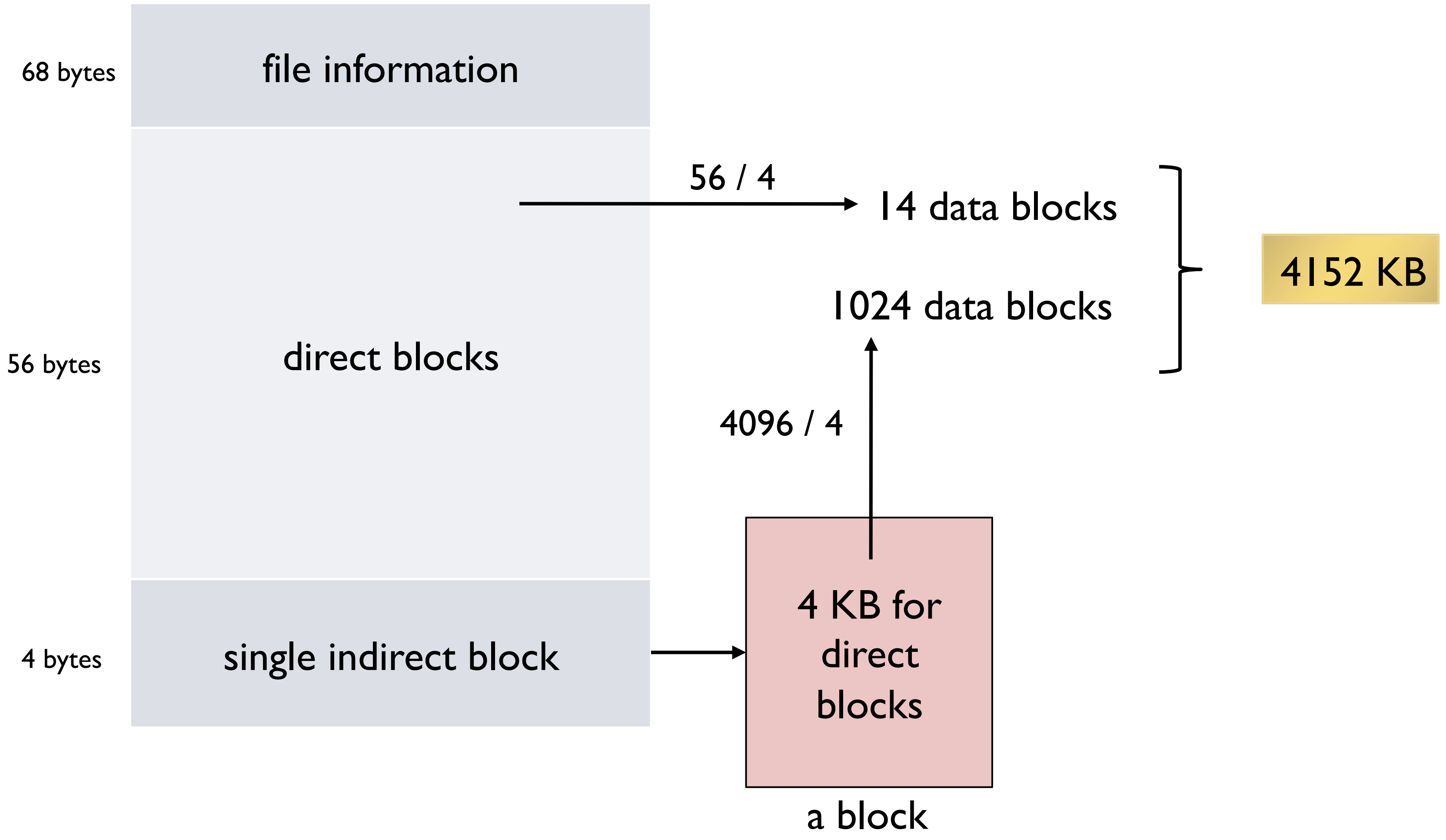
What is the Maximum Size of a File?



32-bit block addresses

4KB blocks

What is the Maximum Size of a File?



Directory Implementation

- A directory is a special file containing information on the contents inside it
  - root directory is always at fixed location on disk

Filename	ID in metadata table
.	<self>
..	<parent>
asg1	1
asg2	2
asg3	3
asg4	4
dir	5

an implementation that keeps pointers (**inode numbers**) to entries in the file metadata table

Filename	start block	length	other attributes
.	…	…	…
..	…	…	
asg1	10	2	…
asg2	24	3	…
asg3	29	6	…
asg4	38	3	…
dir	16	2	…

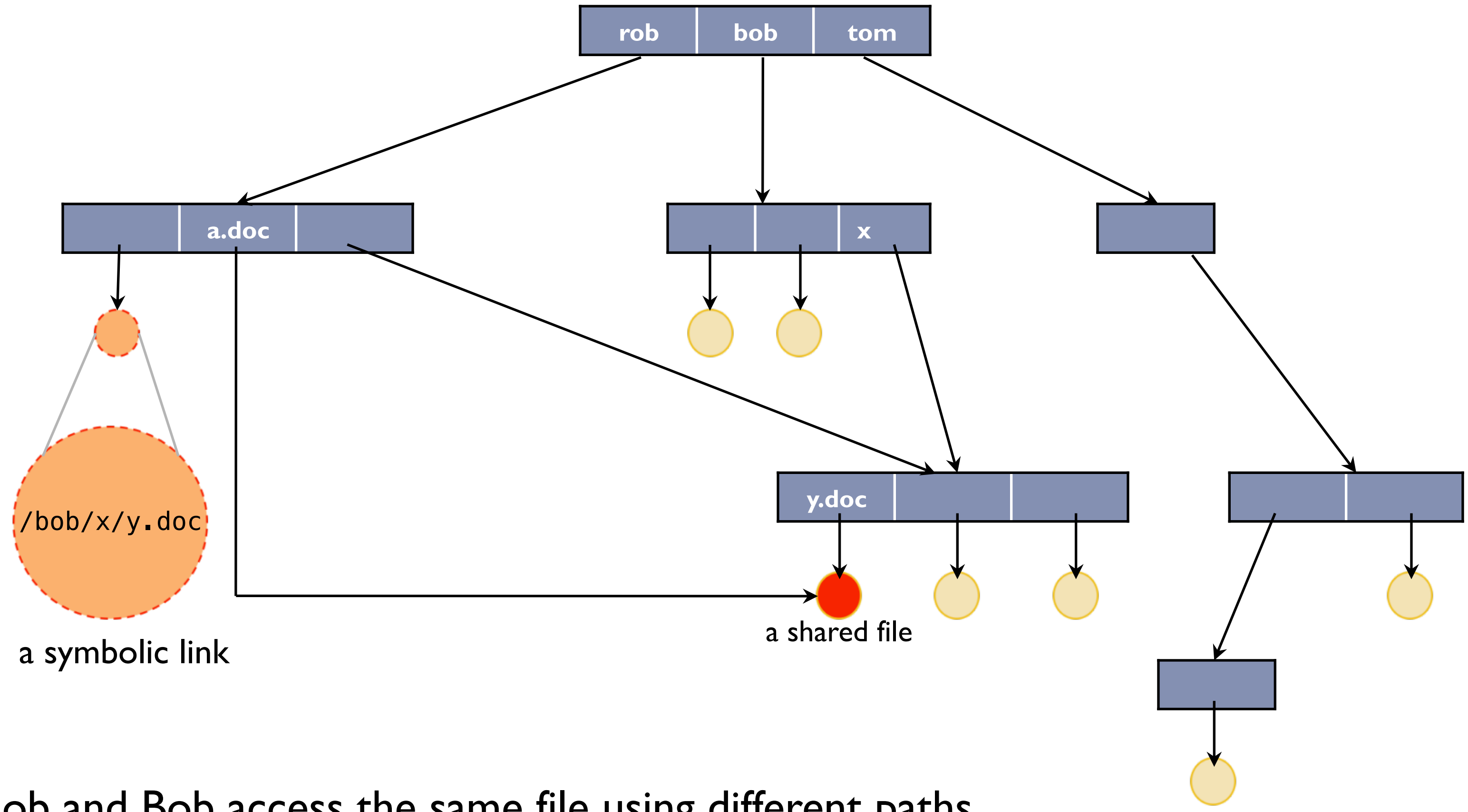
an implementation that keeps all details about the files in their directory entries; in this case the file metadata table is redundant

Finding a File

- `/usr/ast/mbox`
- Look at entries in root directory
  - find i-node number of `usr`
- Check details of `usr` in i-node table
  - is it a directory? where is it on disk? …
- Read contents of `usr` directory file
  - find i-node number of `ast`
- Check details of `ast` in i-node table
- Read contents of `ast` directory file
  - find i-node number of `mbox`
- Check details of `mbox` in i-node table, and read it

28

Acyclic-Graph Directories



Rob and Bob access the same file using different paths

`/rob/a.doc`

`/bob/x/y.doc`

Sharing Files with Links

- If directories contain disk block addresses of files,
  - entry **`a.doc`** is updated when Rob adds to file
  - entry **`y.doc`** is updated when Bob adds to file
- Hard linking solution
  - directories contain only i-node numbers
  - keep count of how many references exist to an i-node
- Soft (symbolic) linking solution
  - Rob creates a special file (called a *link* file) in one of its sub-directories and puts `/bob/x/y.doc` in that file
  - when Rob accesses the link file, OS reads it and fetches the path stored in the file

Journaling File System

- File system operations are lengthy
  - remove a file
    - remove file’s directory entry → release i-node used for file → release all blocks used by file
  - what if system crashes in between?
- **Journaling**: write a log entry before doing a sequence of operations; remove entry once done
  - if an entry exists in log after system recovery, then operations did not finish
  - repeat them
    - necessary that all operations are **idempotent** (can be repeated without any side-effects)

References

- Chapter 4.1-4.3, Modern Operating Systems, A. Tanenbaum and H. Bos, 4th Edition.