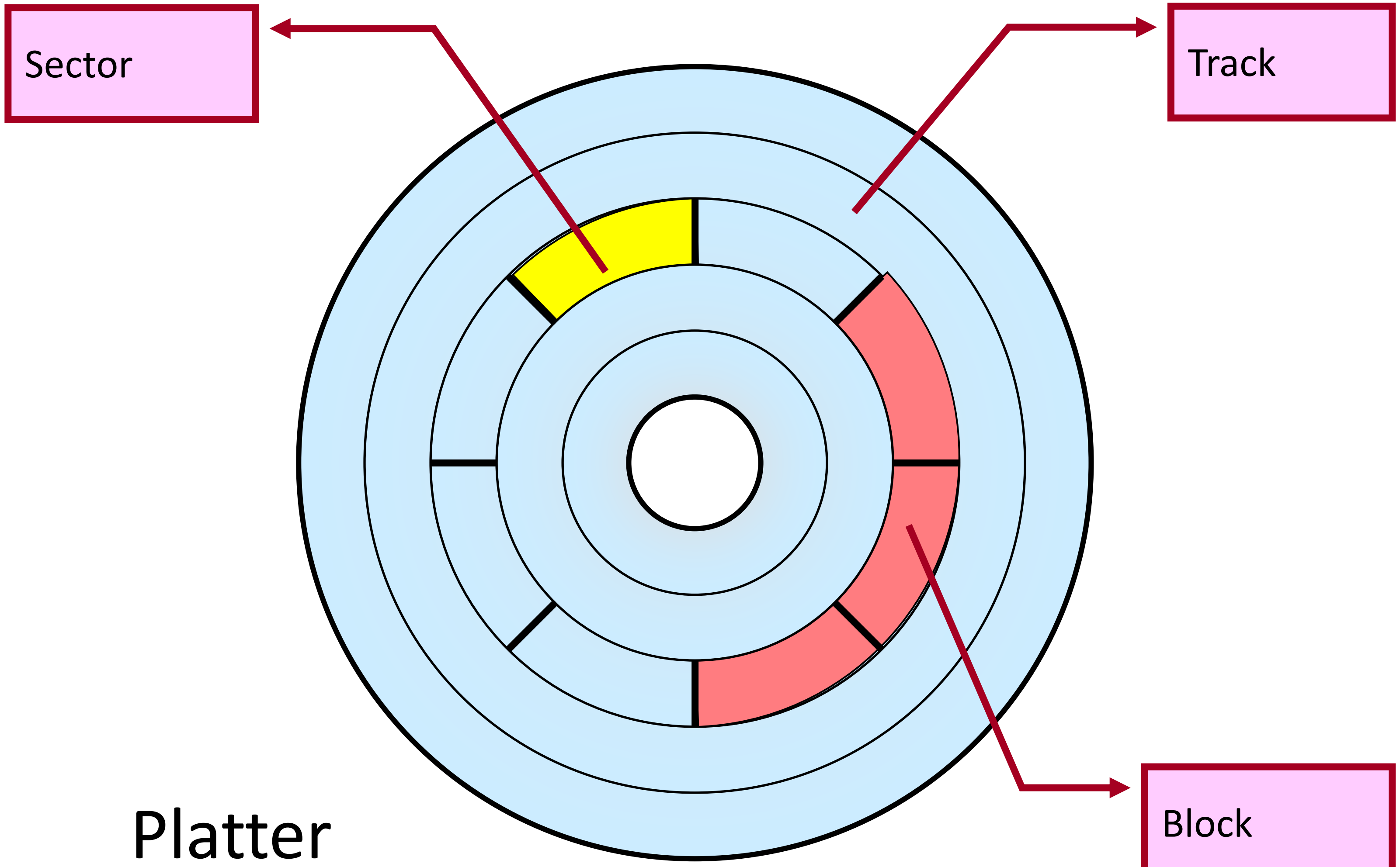
File System

Properties of a Hard Magnetic Disk



How to store data, use data, and share data?

Solution: File

Building a File System

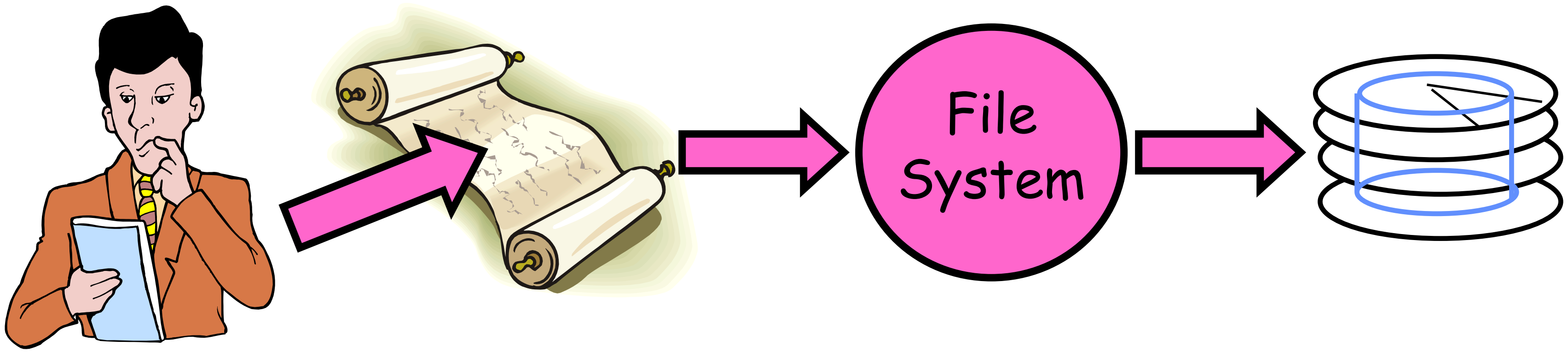
- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.
  - File System Components
  - Disk Management: collecting disk blocks into files
  - Naming: Interface to find files by name, not by blocks
  - Protection: Layers to keep data secure
  - Reliability/Durability: Keeping of files durable despite crashes, media failures, attacks, etc.

Building a File System

- User vs. System View of a File
  - User's view:
    - Durable Data Structures
  - System's view (system call interface):
    - Collection of Bytes
    - Doesn't matter to system what kind of data structures you want to store on disk!

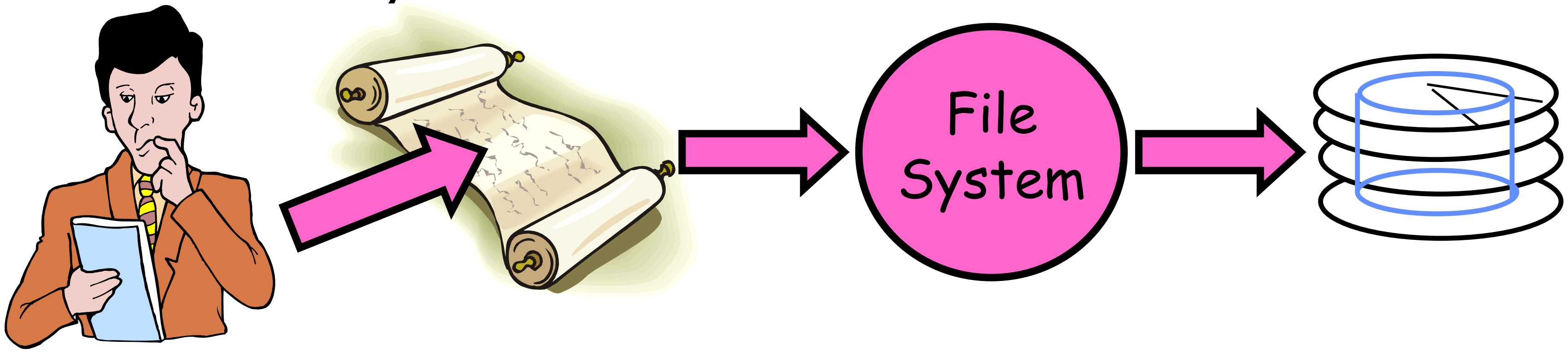
Translating from User to System View

- What happens if user says: give me bytes 2—12?
  - Fetch block corresponding to those bytes
  - Return just the correct portion of the block
- What about: write bytes 2—12?
  - Fetch block
  - Modify portion
  - Write out Block



Translating from User to System View

- Everything inside File System is in whole size blocks
  - For example, **`getc()`**, **`putc()`** $\Rightarrow$ buffers something like 4096 bytes, even if interface is one byte at a time
- From now on, **file** is **a collection of blocks**



Disk Management Policies

- Basic entities on a disk:
  - **File**: user-visible group of blocks arranged sequentially in **logical** space
  - **Directory**: user-visible index mapping names to files (later)

Logical vs Physical

	File	Directory

Logical	File Concept
	File Structure--Logical structure
	    None - sequence of words, bytes
	    Simple record structure
	Access Methods
	    Sequential Access
	    Random Access
		Directory Structure

Physical	Physical structure
	    Contiguous allocation
	    Linked allocation
	        FAT
	    Indexed allocation
	        “inode” in Linux
	FCB
		Table of FCBs

File System

User's View

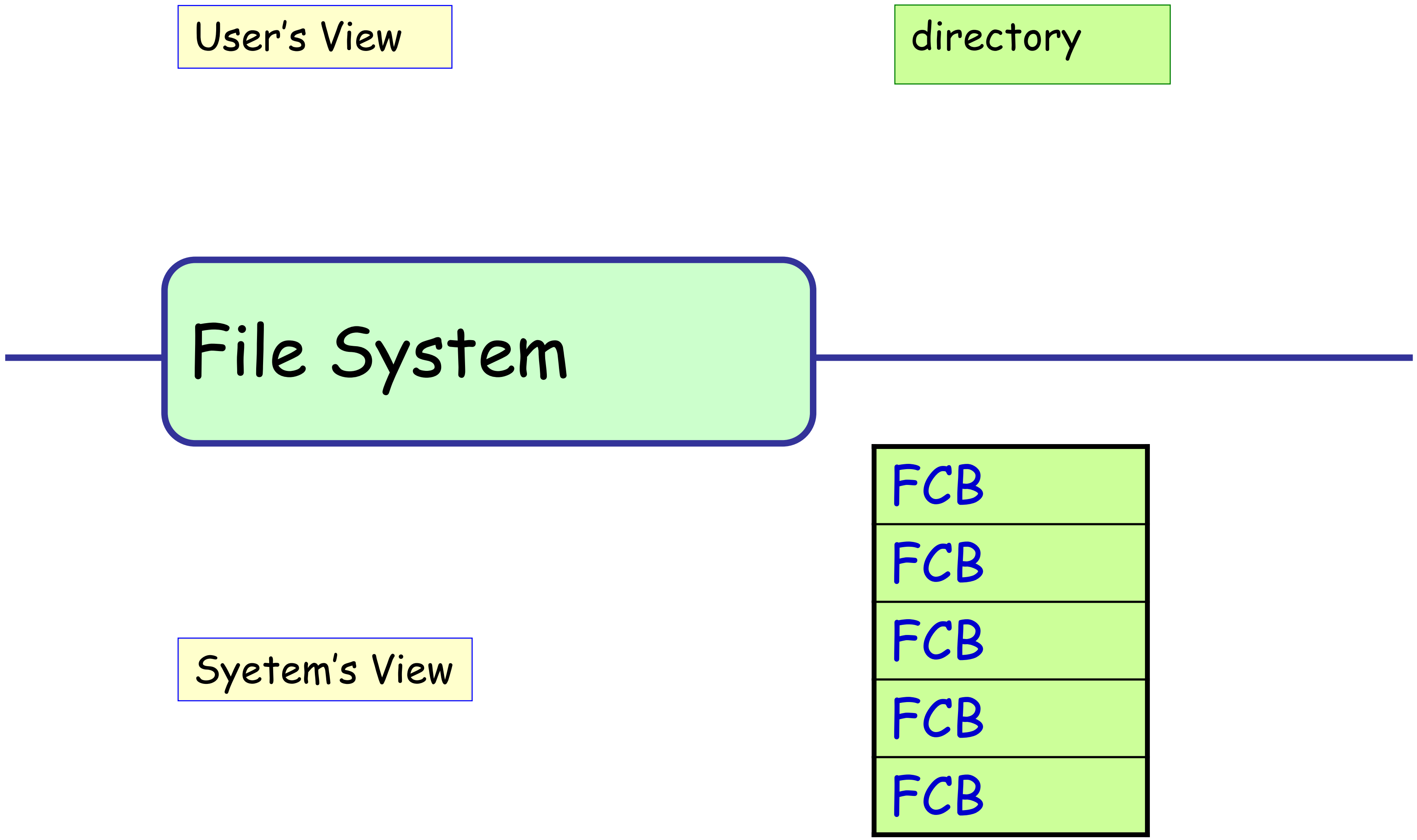
file structure
file attribute
access methods
sizes

File System

physical structure
FCB

Syetem's View

File System


User's View
directory
File System
FCB
FCB
FCB
FCB
FCB
Syetem's View

What is file?

File Concept

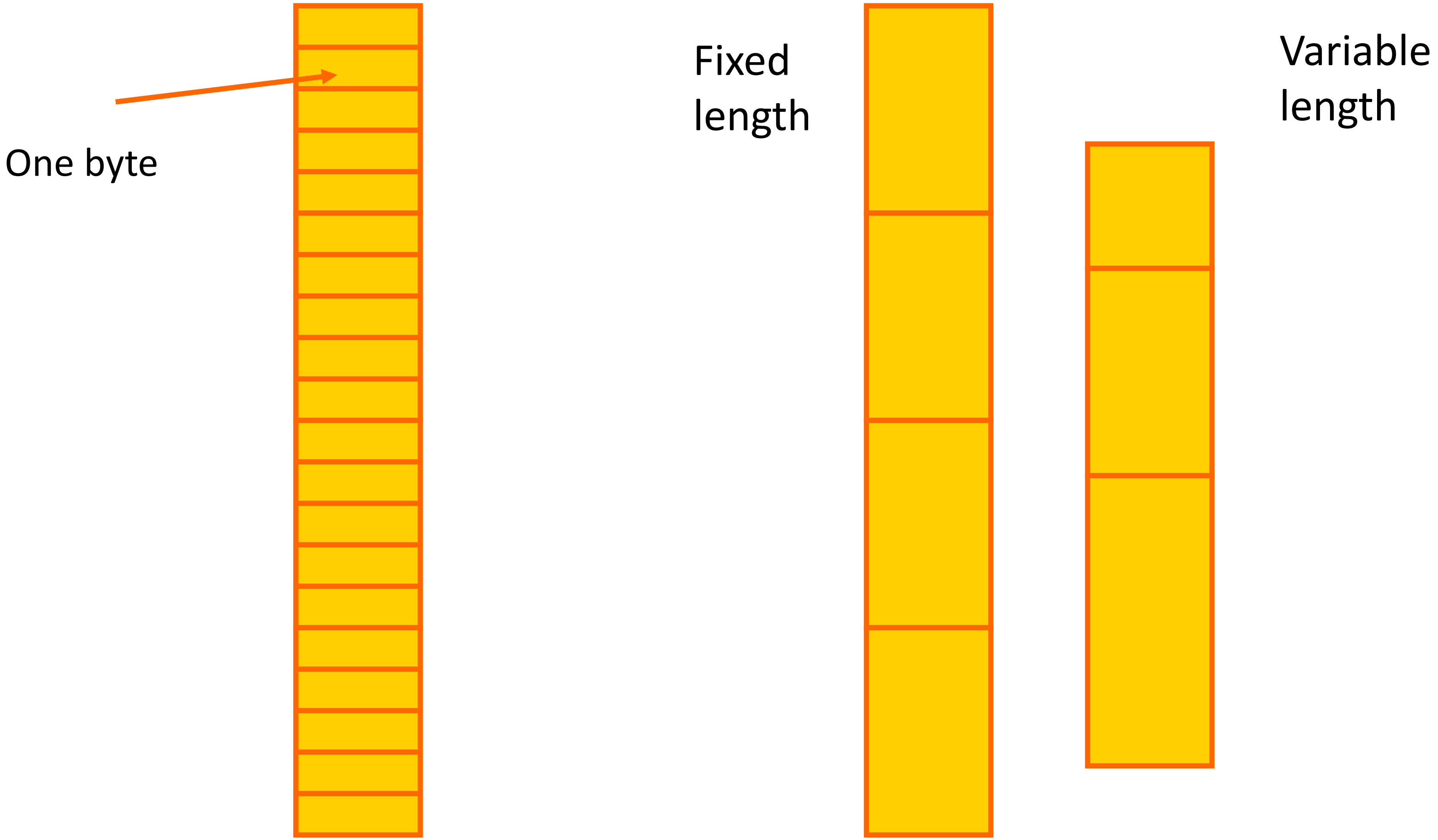
File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program

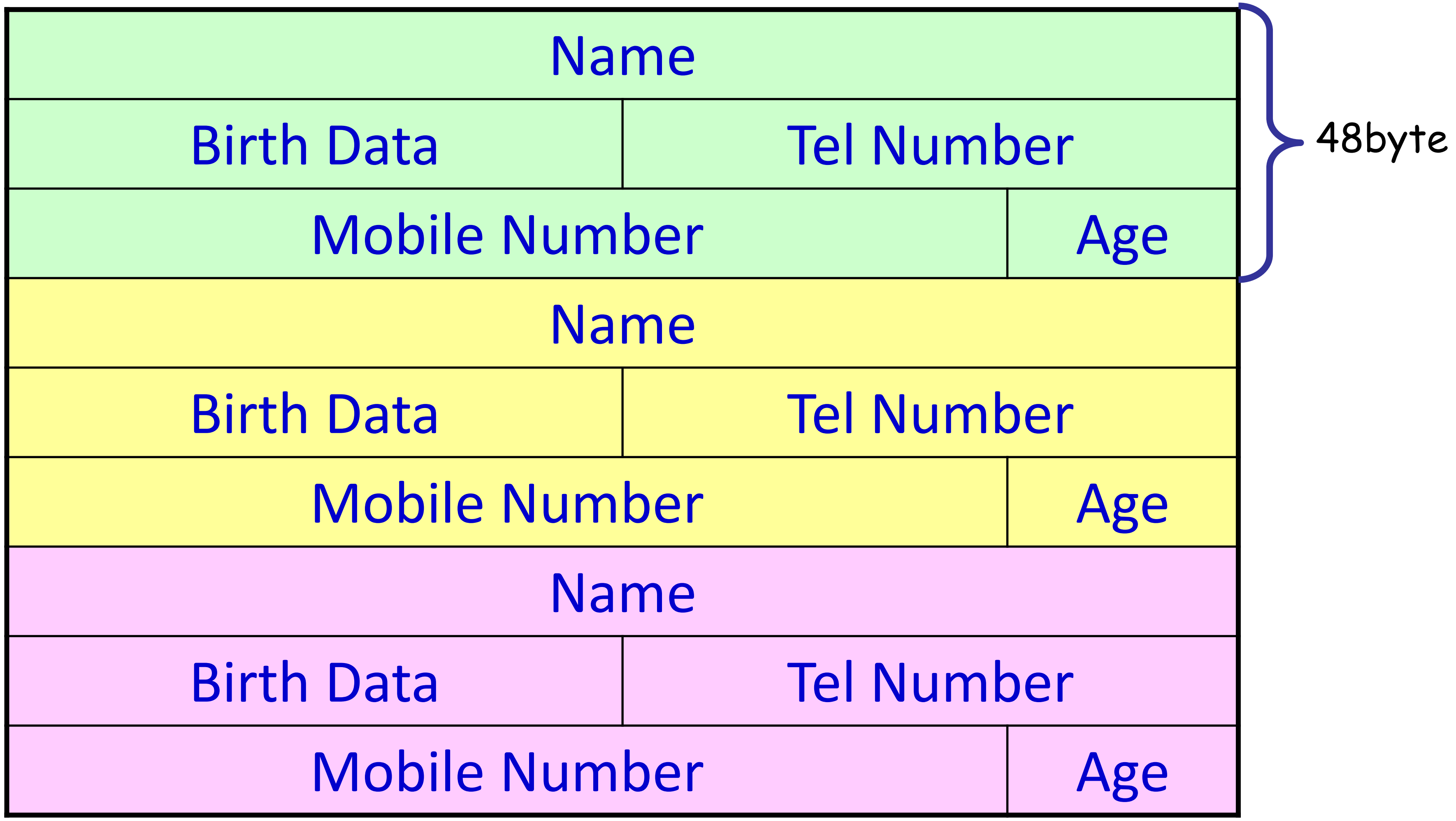
File Structure

- Sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document

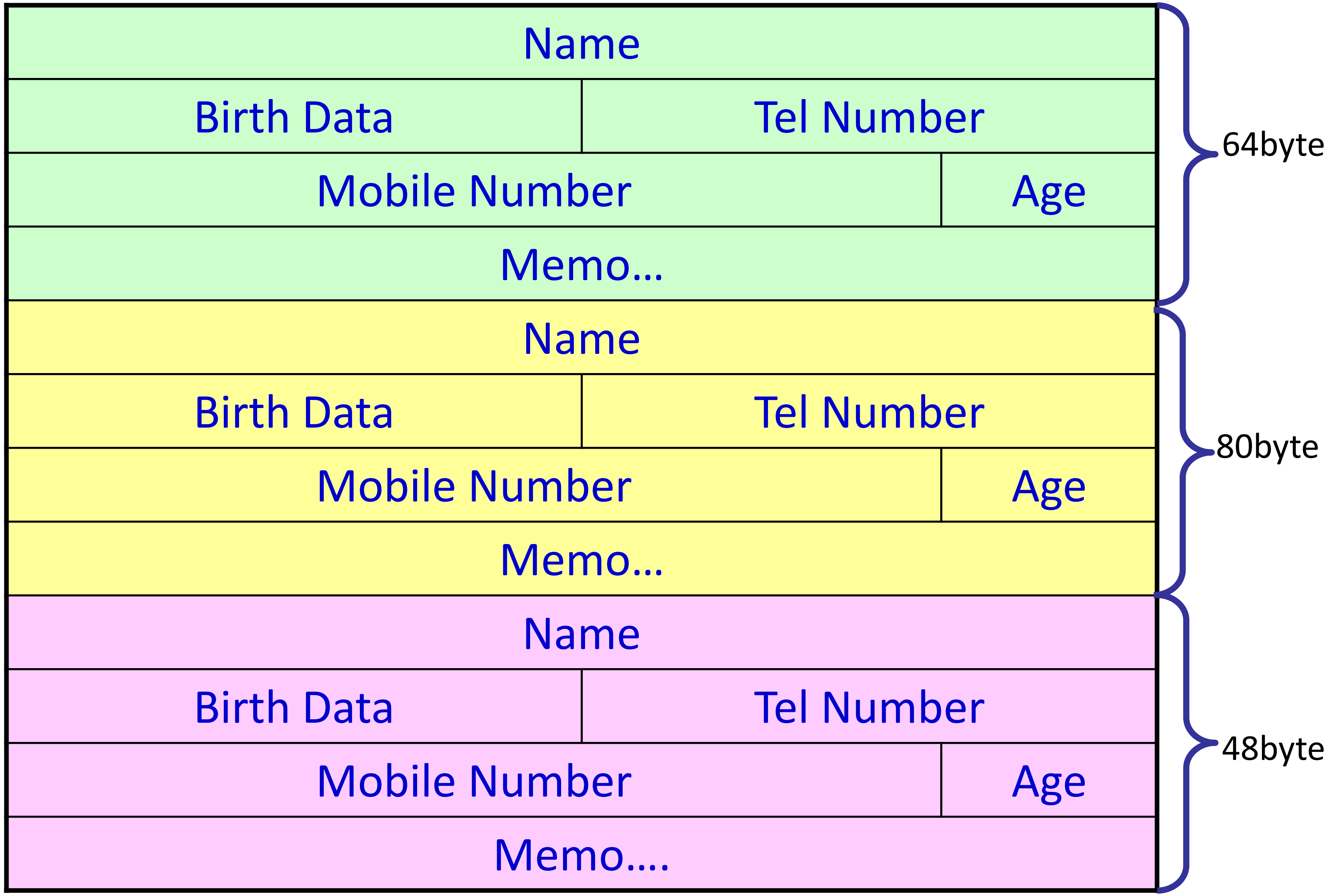
File Structure



File Structure



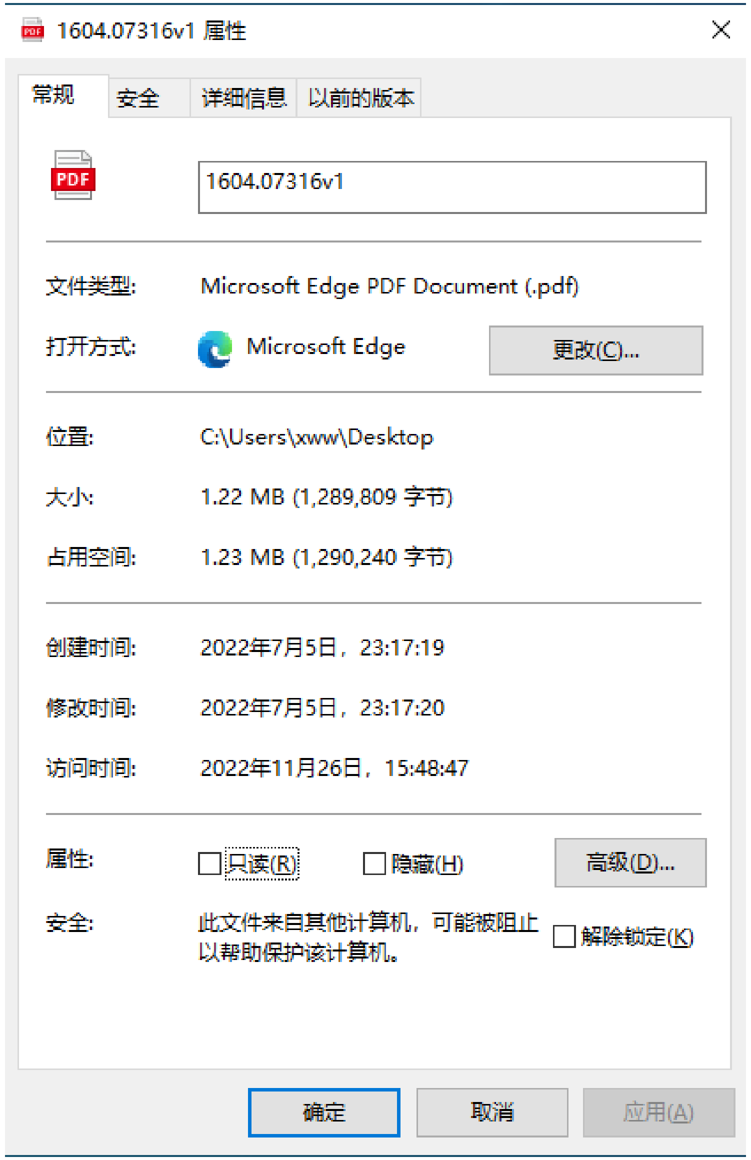
File Structure



File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the **directory structure**, which is maintained on the disk

File Attributes



Designing File System: Usage Patterns

- What are file sizes?
  - Most files are small (for example, .c files)
  - A few files are big – data files, core files, etc.

What can we do with files?

File Operations

- File is an **abstract data type**
  - **Create**
  - **Write**
  - **Read**
  - **Reposition within file**
  - **Delete**
  - **Truncate**
  - **Append**
  - **Seek**
  - **Get Attributes**
  - **Set Attributes**

File Operations

- Open(F_i) – search the directory structure on disk for entry F_i, and move the content of entry to memory
- Close (F_i) – move the content of entry F_i in memory to directory structure on disk

How do users access files?

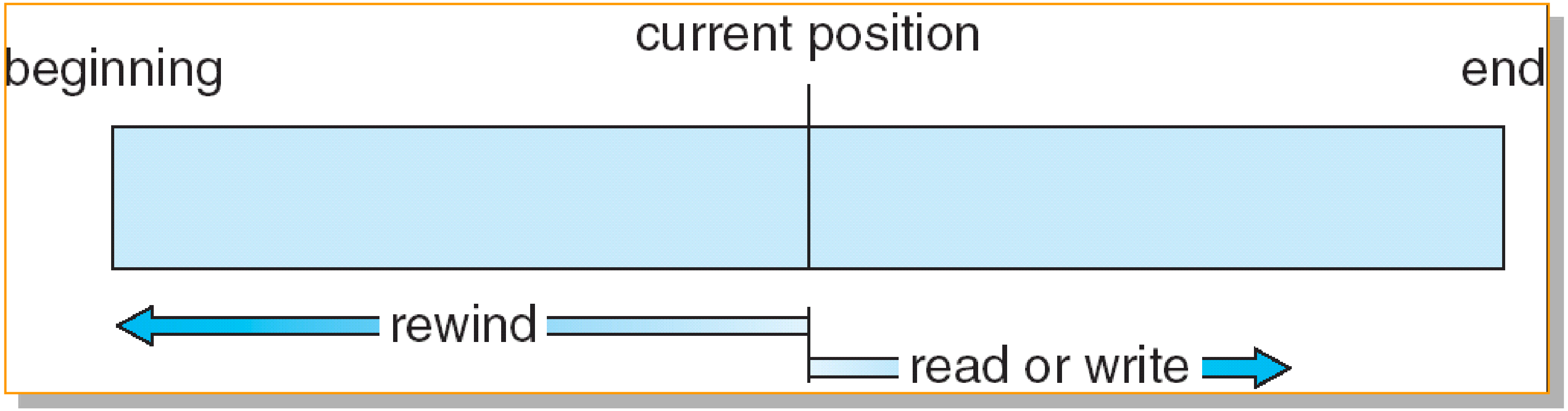
Access Methods

Designing File System: Access Patterns

- How do users access files?
  - Sequential Access
  - Random Access
  - Content-based Access

Sequential Access

- Sequential Access
  - bytes read in order ("give me the next X bytes, then give me next, etc.")
    - Almost all file access are of this flavor



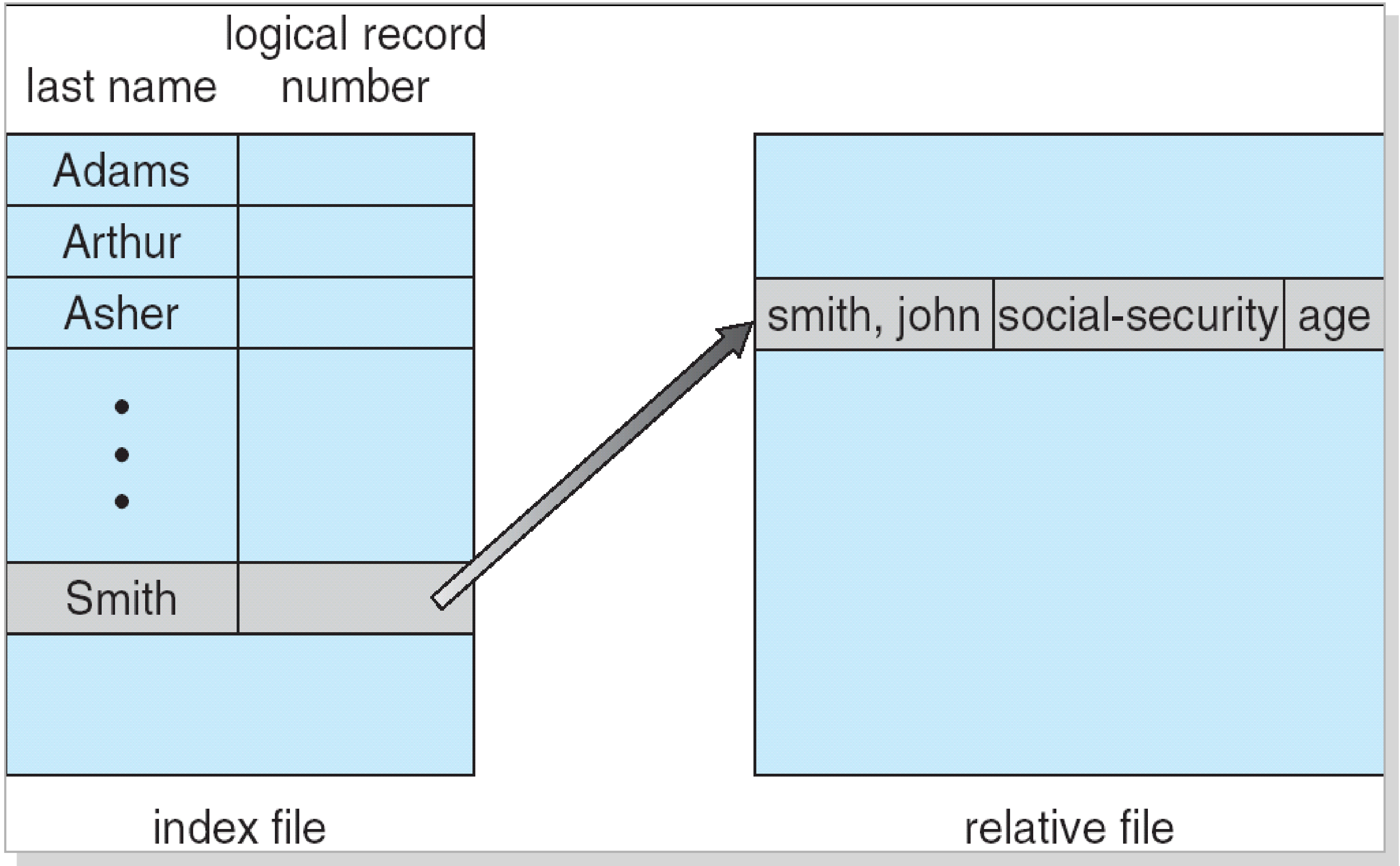
Random Access

- Random Access
  - read/write element out of middle of array ("give me bytes i—j")
    - Also called Direct Access
    - Less frequent, but still important. For example, virtual memory backing file: page of memory stored in file
    - Want this to be fast – don't want to have to read all bytes to get to the middle of the file

Content-based Access

- Content-based Access
  - “find me 100 bytes starting with JOSEPH”)
    - Example: employee records – once you find the bytes, increase my salary by a factor of 2
    - Many systems don’t provide this; instead, databases are built on top of disk access to index content (requires efficient random access)

Example of Index and Relative Files



How to organize files on disk?

Allocation Methods

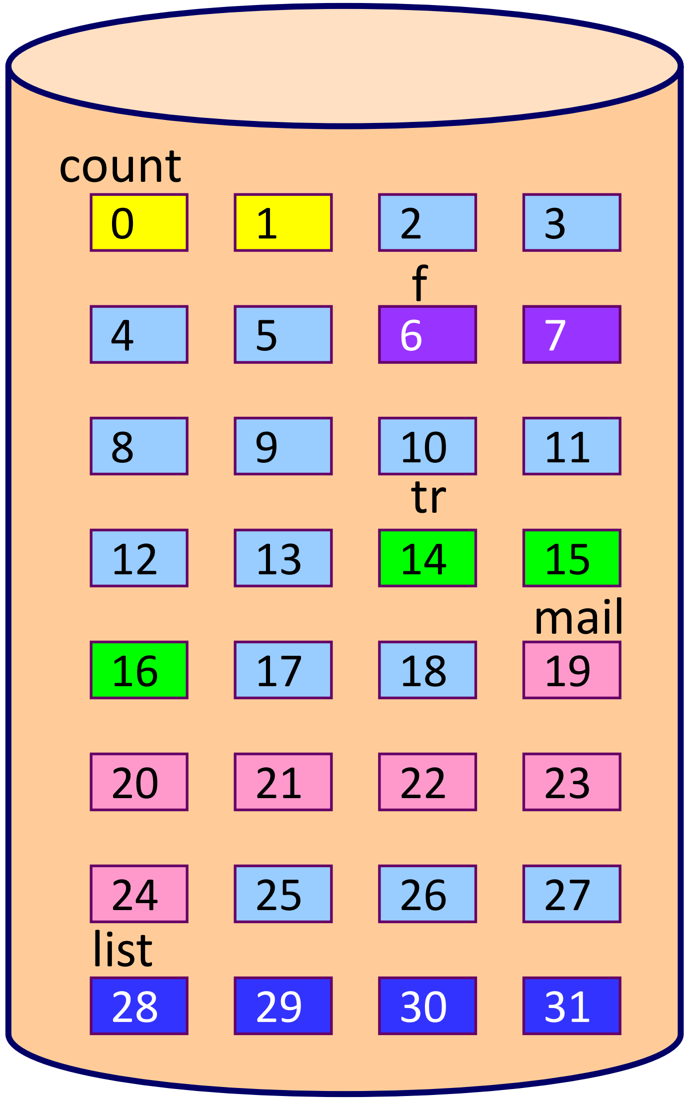
How to organize files on disk

- Goals:
  - Maximize sequential performance
  - Easy random access to file
  - Easy management of file (growth, truncation, etc.)
- An allocation method refers to how disk blocks are allocated for files:
  - Contiguous allocation
  - Linked allocation
    - FAT
  - Indexed allocation
- File physical structure

How to organize files on disk

- First Technique: Continuous Allocation
  - Use continuous range of blocks in logical block space
    - Analogous to base+limit in virtual memory
    - User says in advance how big file will be (disadvantage)
  - File Header Contains:
    - First block
    - File size (# of blocks)

Continuous Allocation



directory

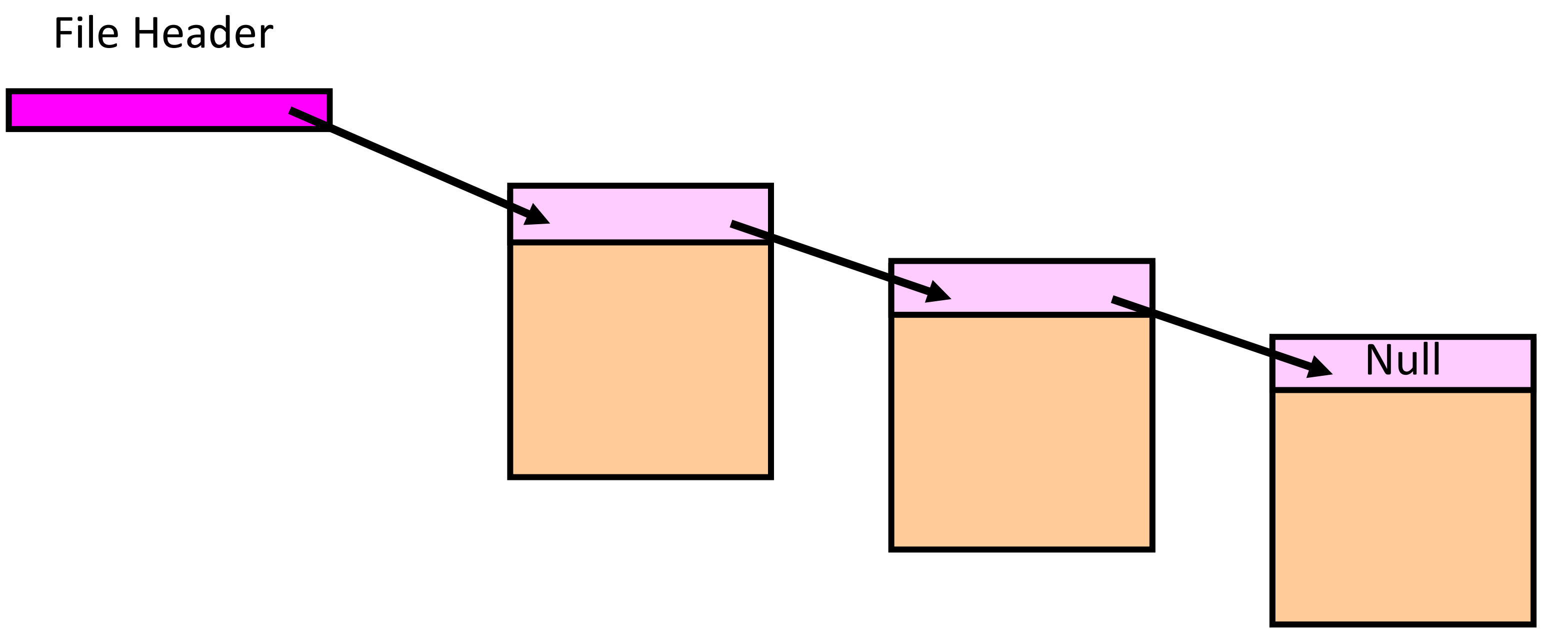
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Continuous Allocation

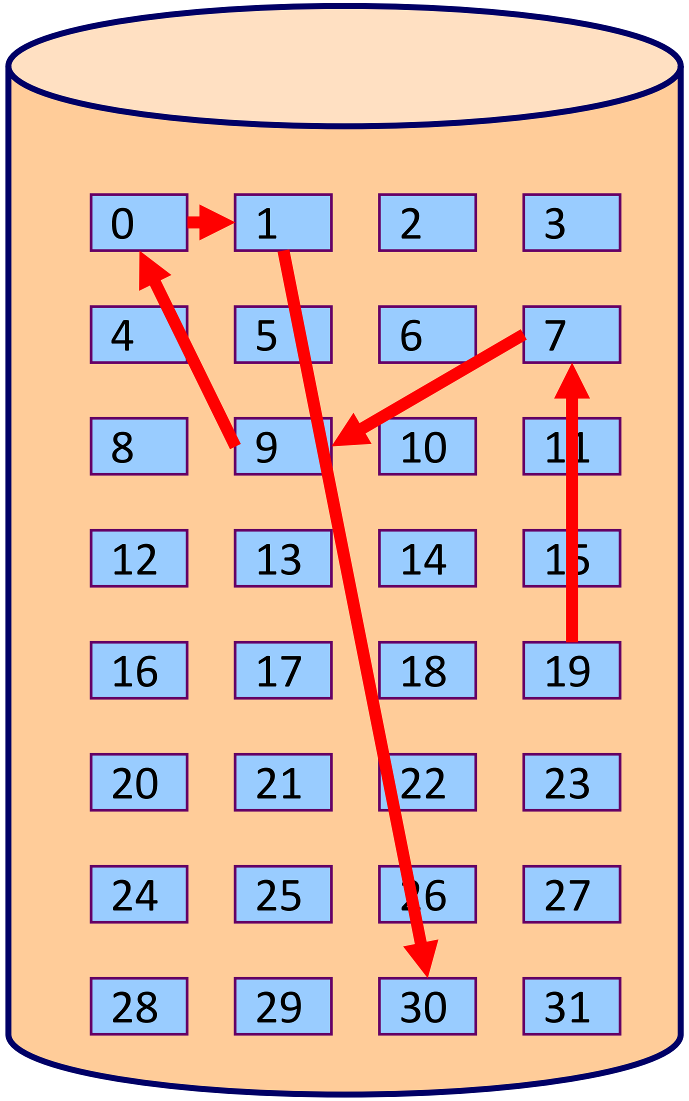
- First Technique: Continuous Allocation
  - Pros:
    - Simple – only starting location (block #) and length (number of blocks) are required
    - Fast Sequential Access, Easy Random access
  - Cons:
    - External Fragmentation/Hard to grow files
    - Free holes get smaller and smaller, wasteful of space
    - Could compact space, but that would be *really* expensive
- CD, DVD

Linked List Allocation

- Second Technique: Linked List Approach
  - Each block, pointer to next on disk

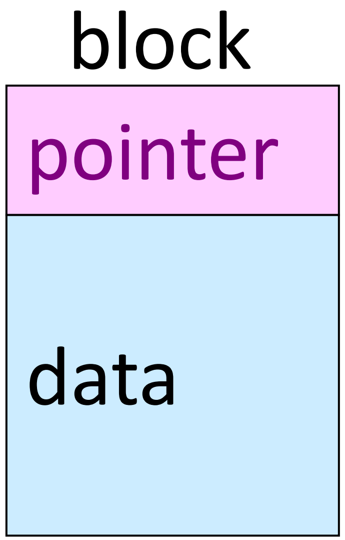


Linked List Allocation



directory

file	start	end
Wise	19	30



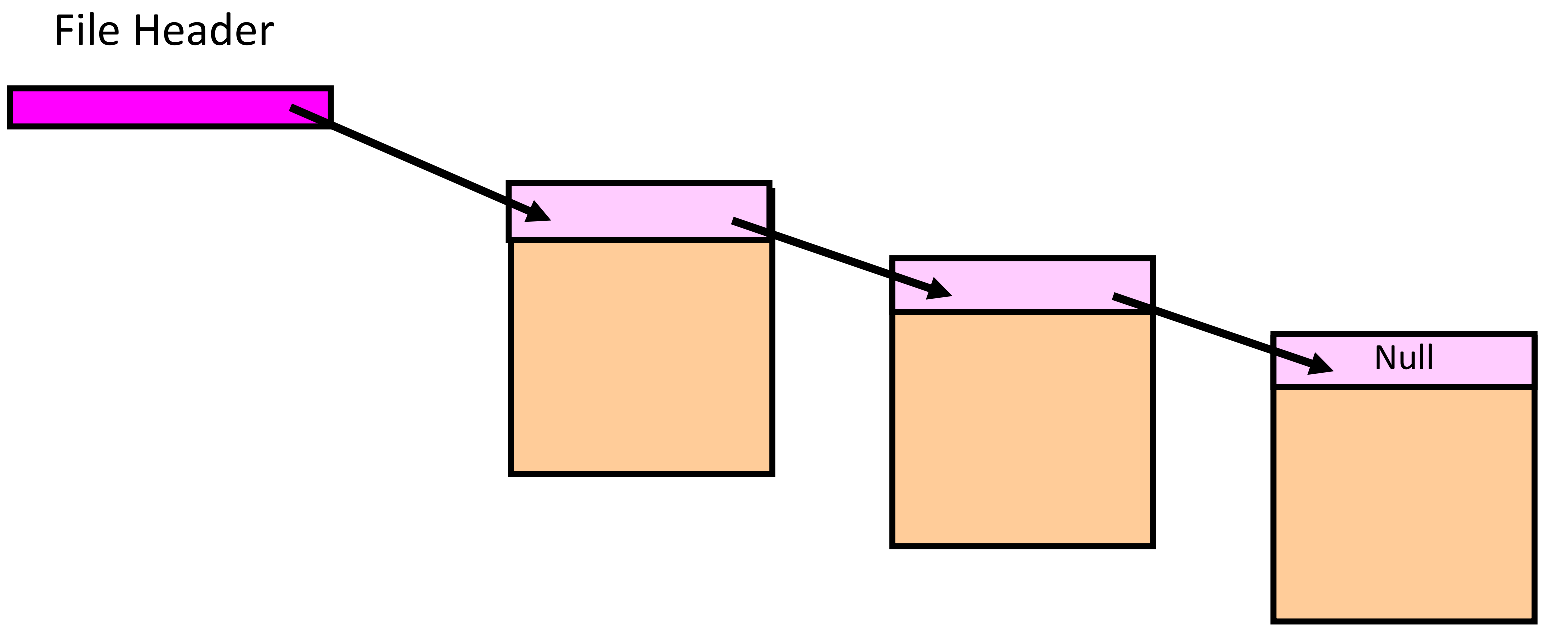
Linked List Allocation

- Second Technique: Linked List Approach
  - Pros: Can grow files dynamically, Free list same as file
  - Cons: Bad Sequential Access (seek between each block), Unreliable (lose block, lose rest of file)
  - Serious Con: Bad random access!!!!

Linked Allocation: File-Allocation Table

- MSDOS links pages together to create a file
  - Links not in pages, but in the File Allocation Table (FAT)
    - FAT contains an entry for each block on the disk
    - FAT Entries corresponding to blocks of file linked together
  - FAT should be cached in memory

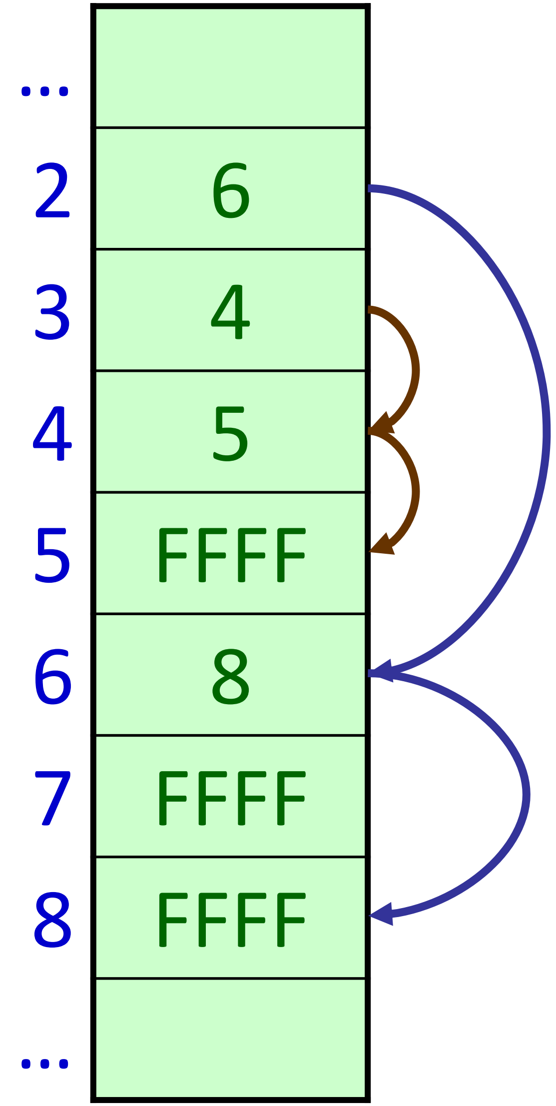
Linked Allocation: File-Allocation Table



File-Allocation Table

file	start
Super	3
Wise	2
Star	7

Directory entry



no. of disk blocks

File-Allocation Table

- Disk size: 32GB
- Block size: 512B

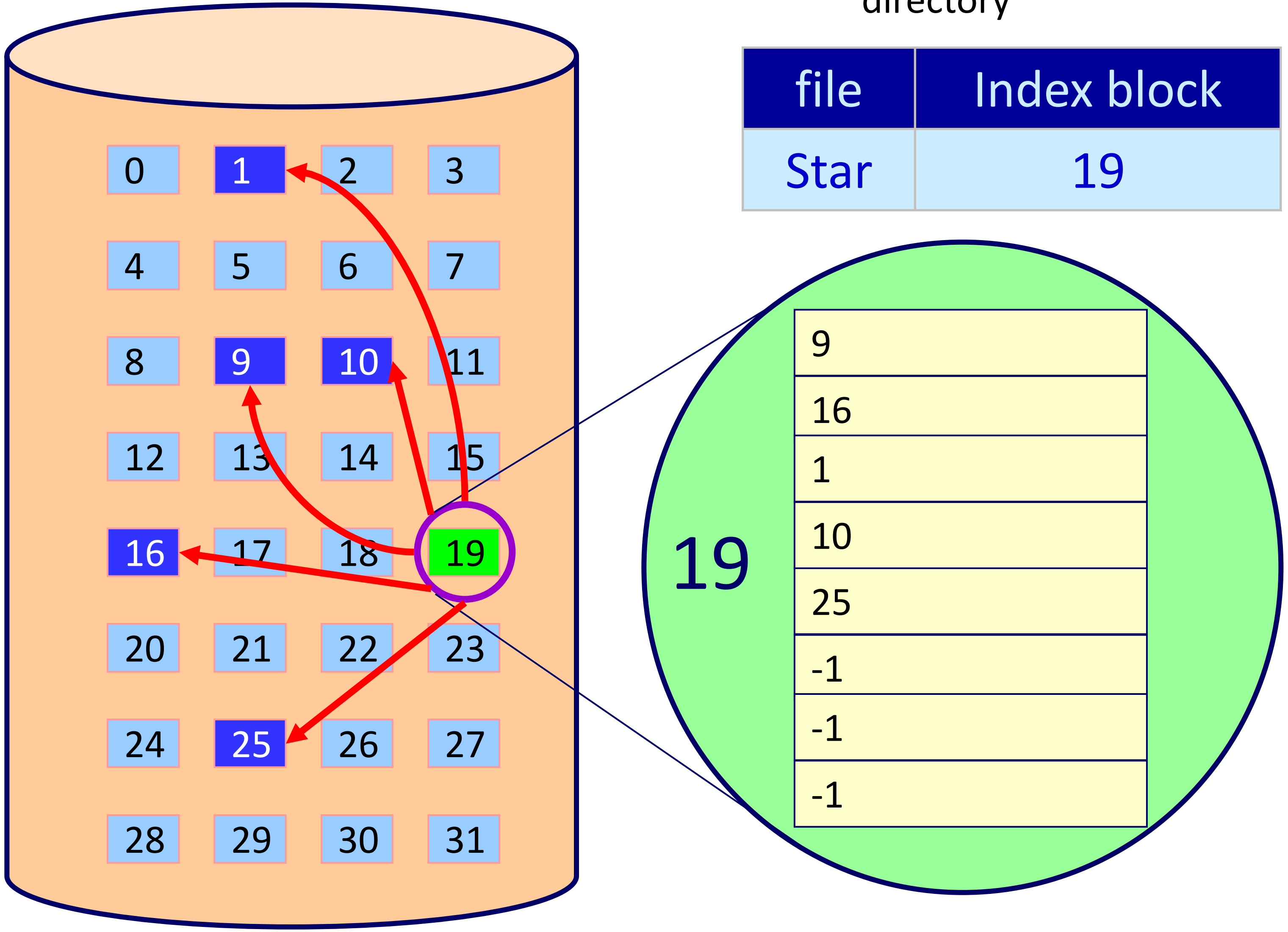
- #Table items: 64M
- Item size: 4B

- Table size: 256M

Indexed Files

- Third Technique: Indexed Files
  - System allocates file header block to hold array of pointers big enough to point to all blocks
    - User pre-declares max file size;

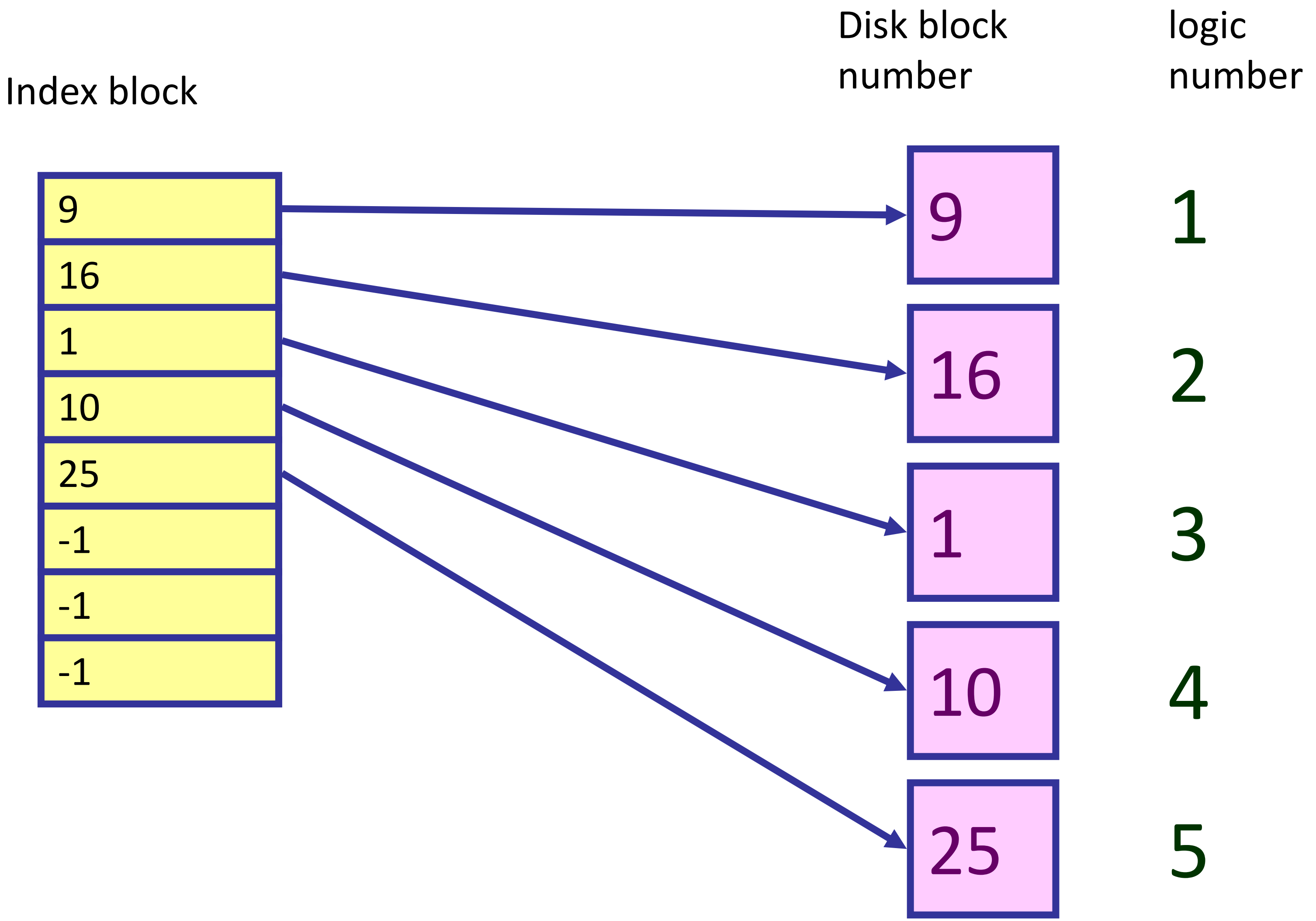
Indexed Files



directory

file	Index block
Star	19

Indexed Files



Indexed Files

- Third Technique: Indexed Files
  - Pros:
    - Can easily grow up to space allocated for index
    - Random access is fast
  - Cons:
    - Clumsy to grow file bigger than table size
    - Still lots of seeks: blocks may be spread over disk

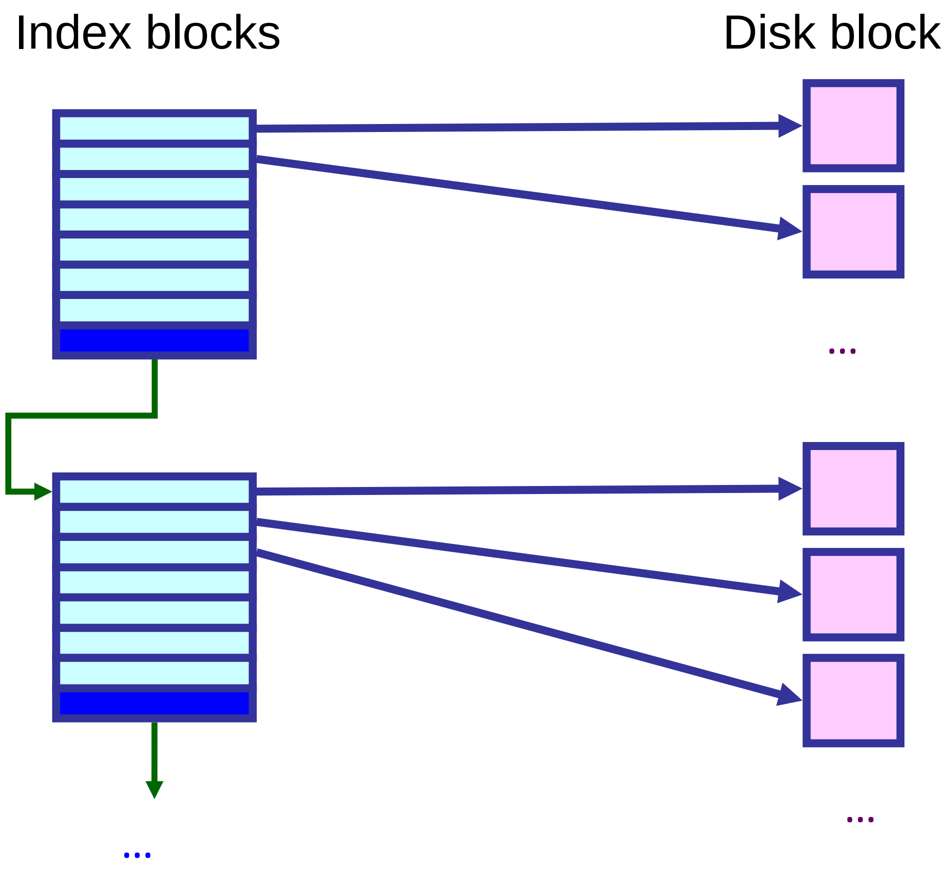
Indexed Files

- Block size: 512B
- Item size: 4B

- Maximum size of a file
  - 64KB

- Is it big enough?

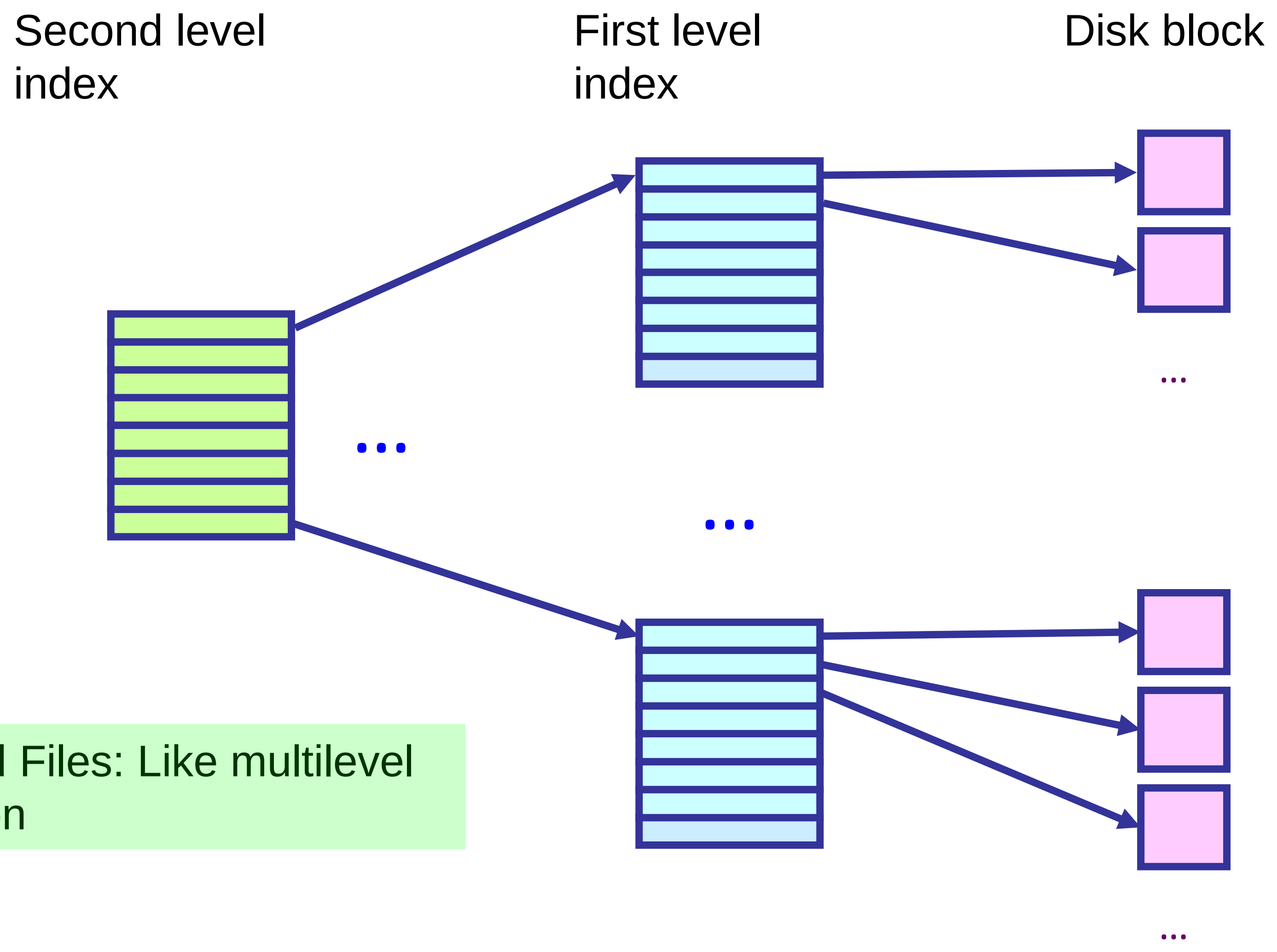
Linked scheme



Mapping from logical to physical in a file of unbounded length

No limit on size

Multilevel index



Multilevel Indexed Files: Like multilevel address translation

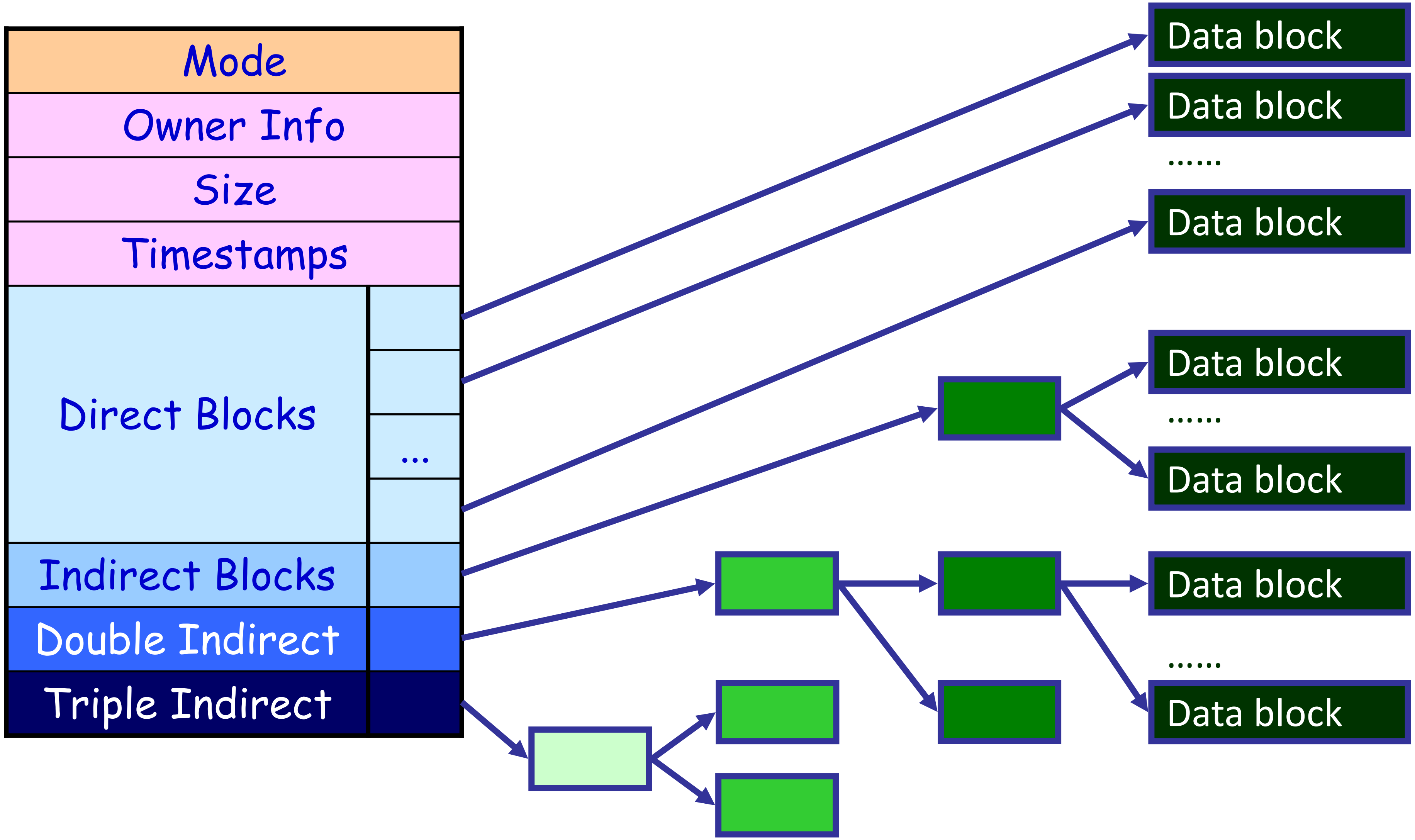
Multilevel Indexed Files (Linux)

- Key idea: efficient for small files, but still allow big files
- "**inode/Inode**" in Linux:
  - 15 pointers
  - Block size: 1KB
  - Pointer length: 4Byte
  - First 12 pointers are to data blocks
  - Pointer 12 points to "indirect block" containing 256 block ptrs
  - Pointer 13 points to "doubly indirect block" containing 256 indirect block ptrs for total of 64K blocks
  - Pointer 14 points to a triply indirect block (16M blocks)
  - MAX file size: 12K+256K+64M+16G

Multilevel Indexed Files (Linux)

Mode
Owner Info
Size
Timestamps
Direct Blocks
Indirect Blocks
Double Indirect
Triple Indirect

Multilevel Indexed Files (Linux)



Example of Multilevel Indexed Files

- Sample file in multilevel indexed format:
  - How many accesses for block #23?
    - Two: One for indirect block, one for data
  - How about block #5?
    - One: One for data
  - Block #340?
    - Three: double indirect block, indirect block, and data

Multilevel Indexed Files

- Pros and cons
  - Pros:
    - Simple (more or less)
    - Files can easily expand (up to a point)
    - Small files particularly cheap and easy
  - Cons:
    - Lots of seeks
    - Very large files must read many indirect block (four I/Os per block!)

How do we actually access files?

File-System Implementation

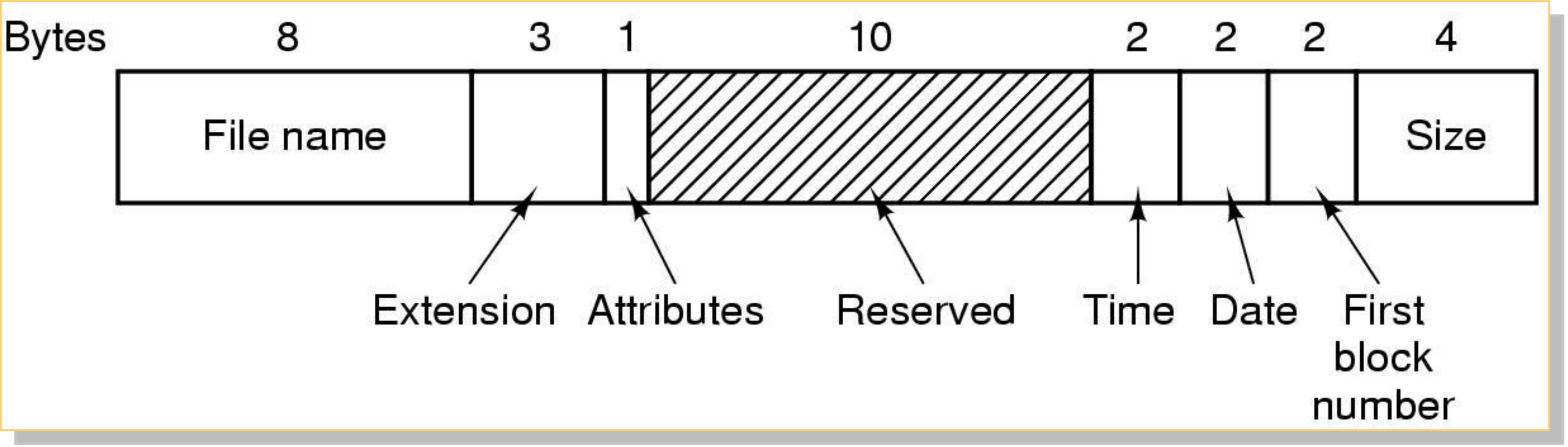
File Control Block (FCB)

- All information about a file contained in its File Control Block (FCB)
- UNIX calls this an “inode”

A Typical File Control Block

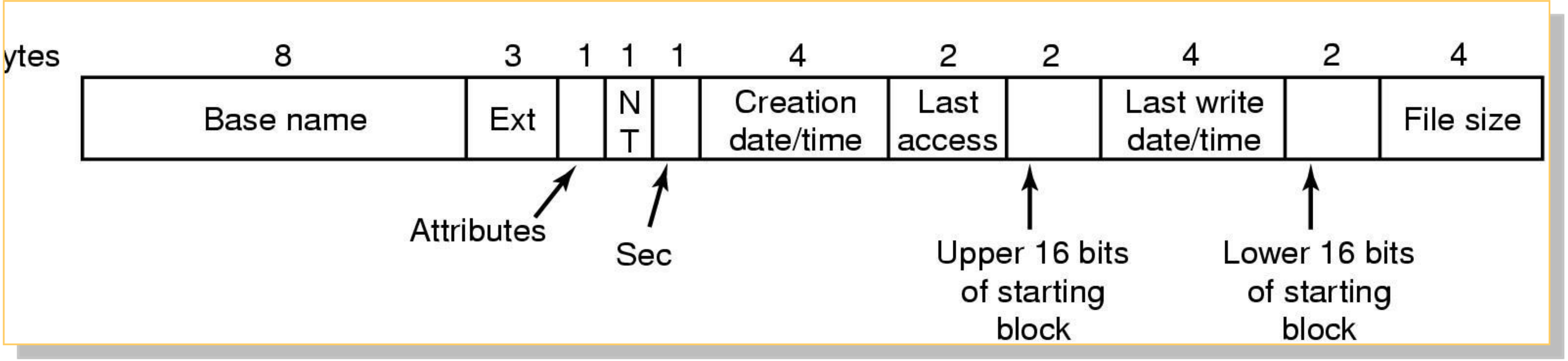
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

FCB of MS-DOS



Bytes
8
3
1
10
2
2
2
4
File name
Size
Extension
Attributes
Reserved
Time
Date
First block number

FCB of Windows



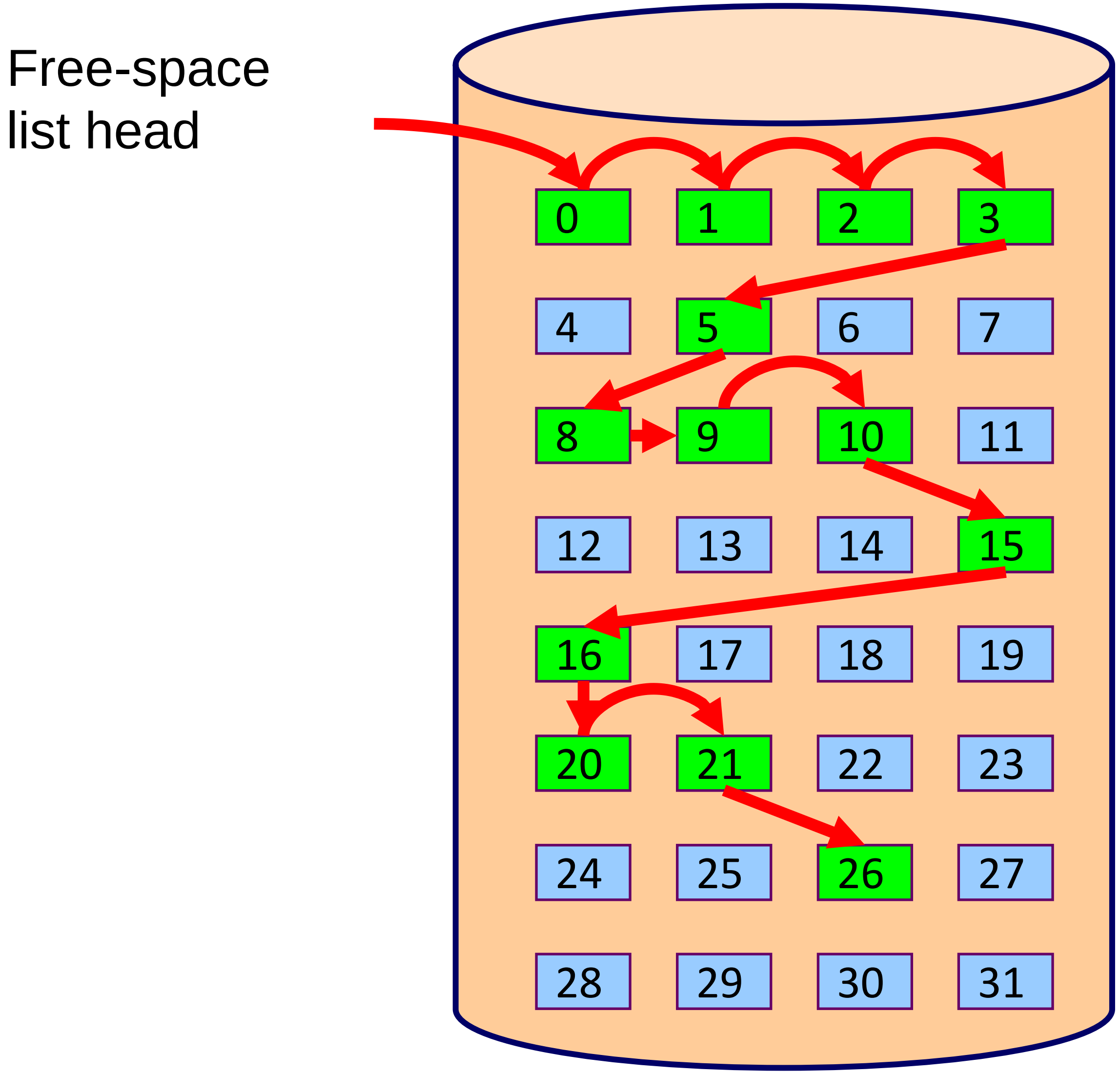
Where do we still have to go?

- Need way to track free disk blocks
- Don't yet know how to name/locate files
  - What is a directory?
  - How do we look up files?

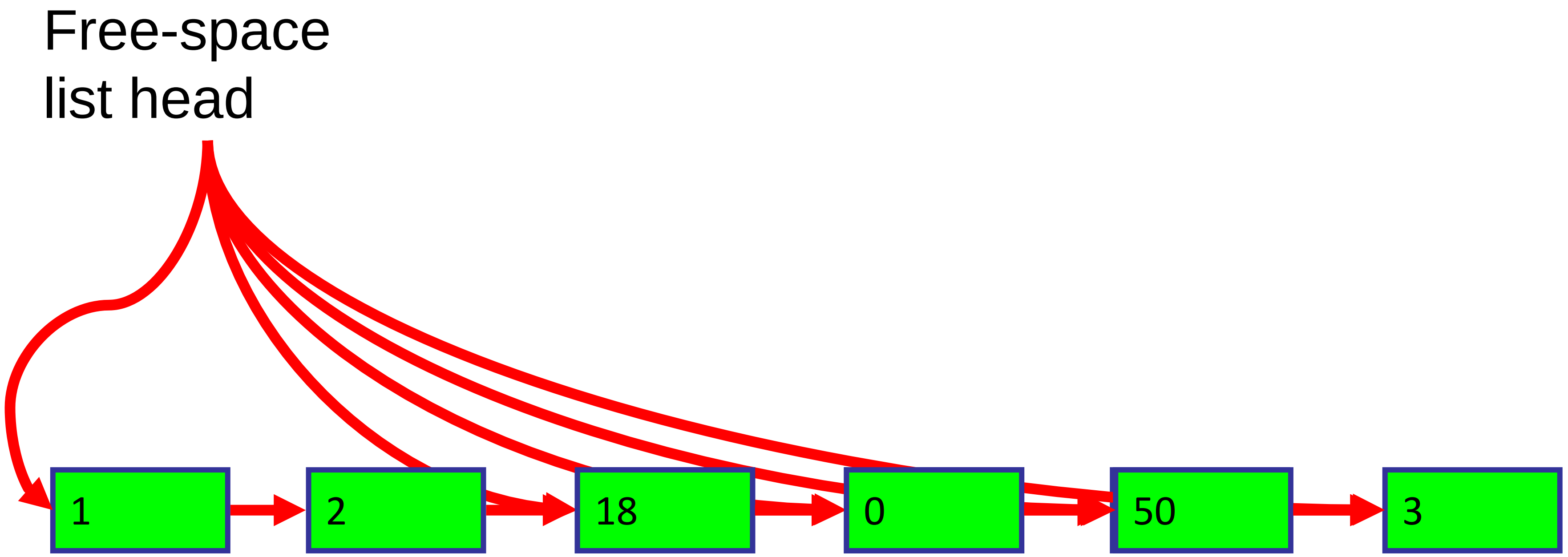
How to track free disk blocks?

Free-Space Management

Linked Free Space List on Disk

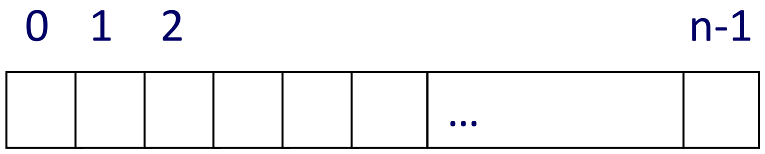


Linked Free Space List on Disk

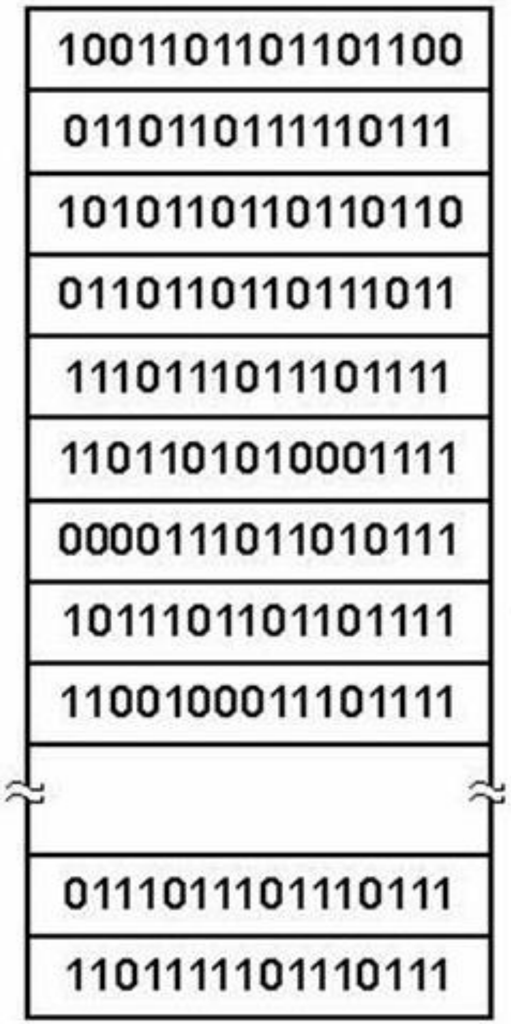


Free-Space Management

- Bit vector (n blocks)



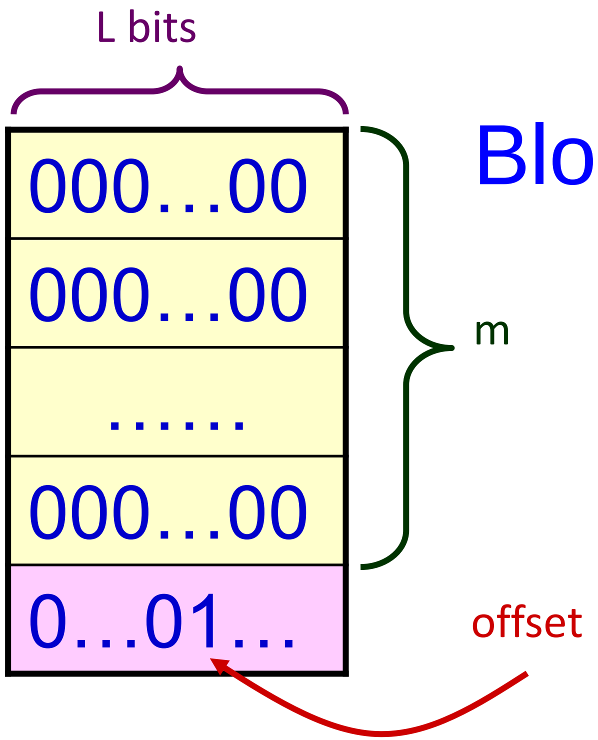
$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$



A bit map

Free-Space Management

- Block number calculation
  - (number of bits per word) * (number of 0-value words) + offset of first 1 bit



Block Number = $L \times m + \text{offset}$

Free-Space Management

- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

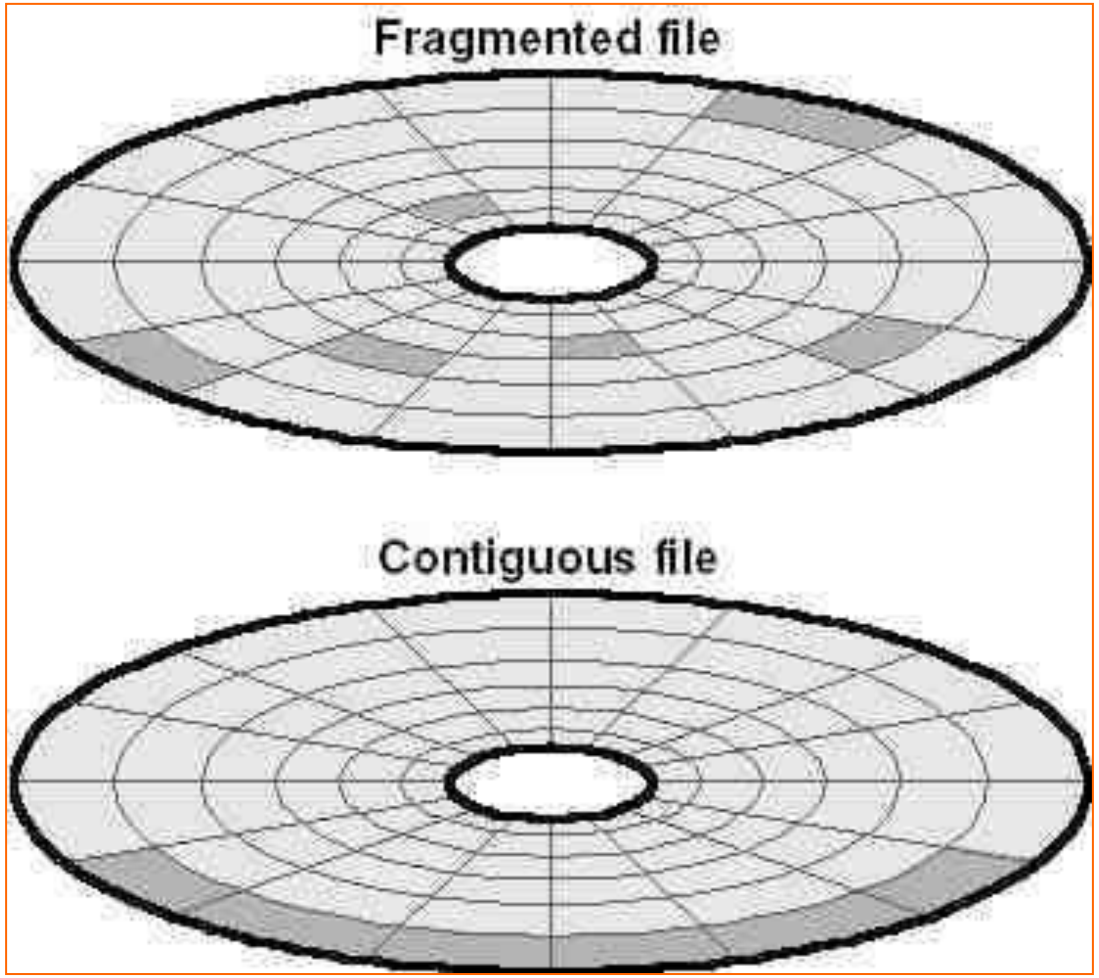
    disk size = 2^{34} bytes (16 gigabyte)

    $n = 2^{34}/2^{12} = 2^{22}$ bits
- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space

Example

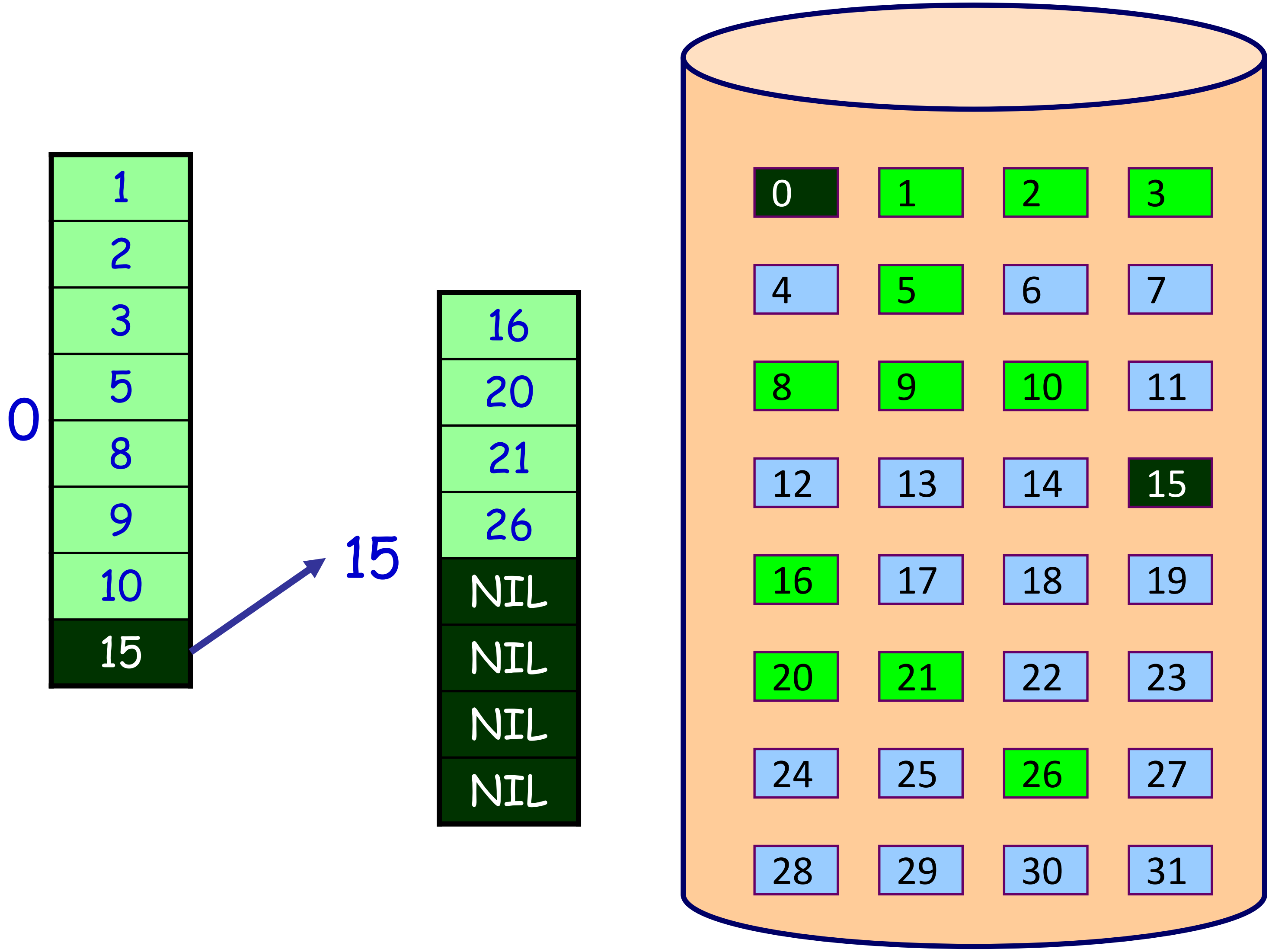
- 某文件管理系统在磁盘上建立了位示图（bitmap），记录磁盘的使用情况。若磁盘的物理块依次编号为：0,1,2,……,系统中字长为32位，每一位对应文件存储器上的一个物理块，取值0和1分别表示空闲和占用。假设将4195号物理块分配给某文件，那么该物理块的使用情况在位示图中的第（　　）个字中描述，系统应该将该字的第（　　）位置为（　　）。

Free-Space Management



Free-Space Management

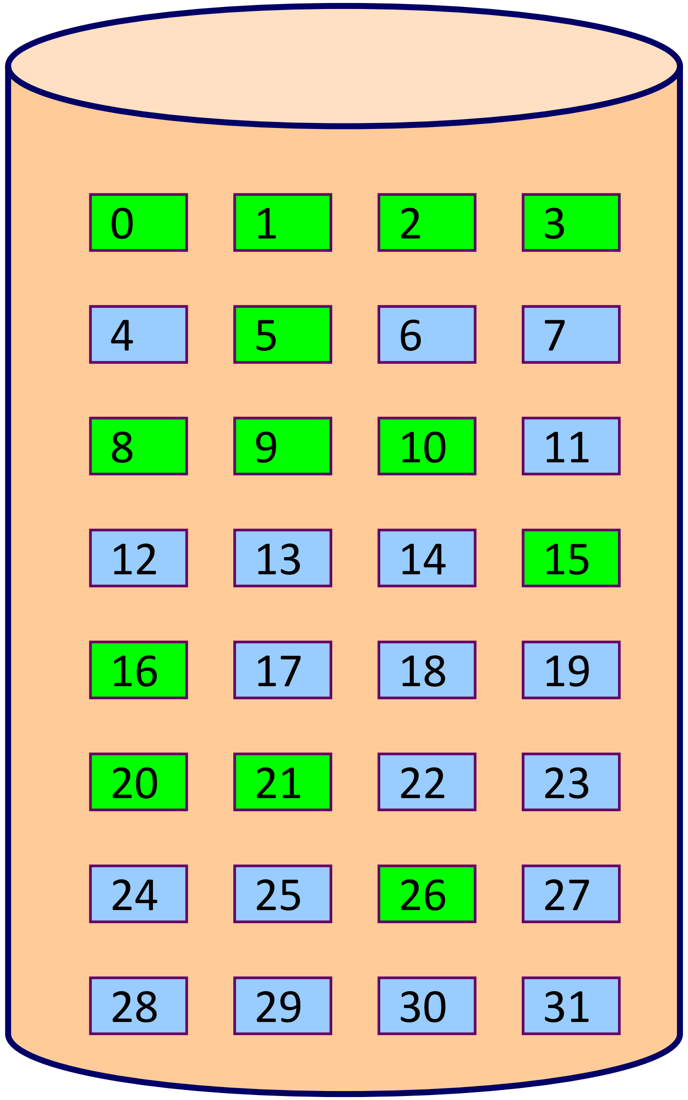
- Grouping



Free-Space Management

- Counting

0	4
5	1
8	3
15	2
20	2
26	1



How to name/locate files?

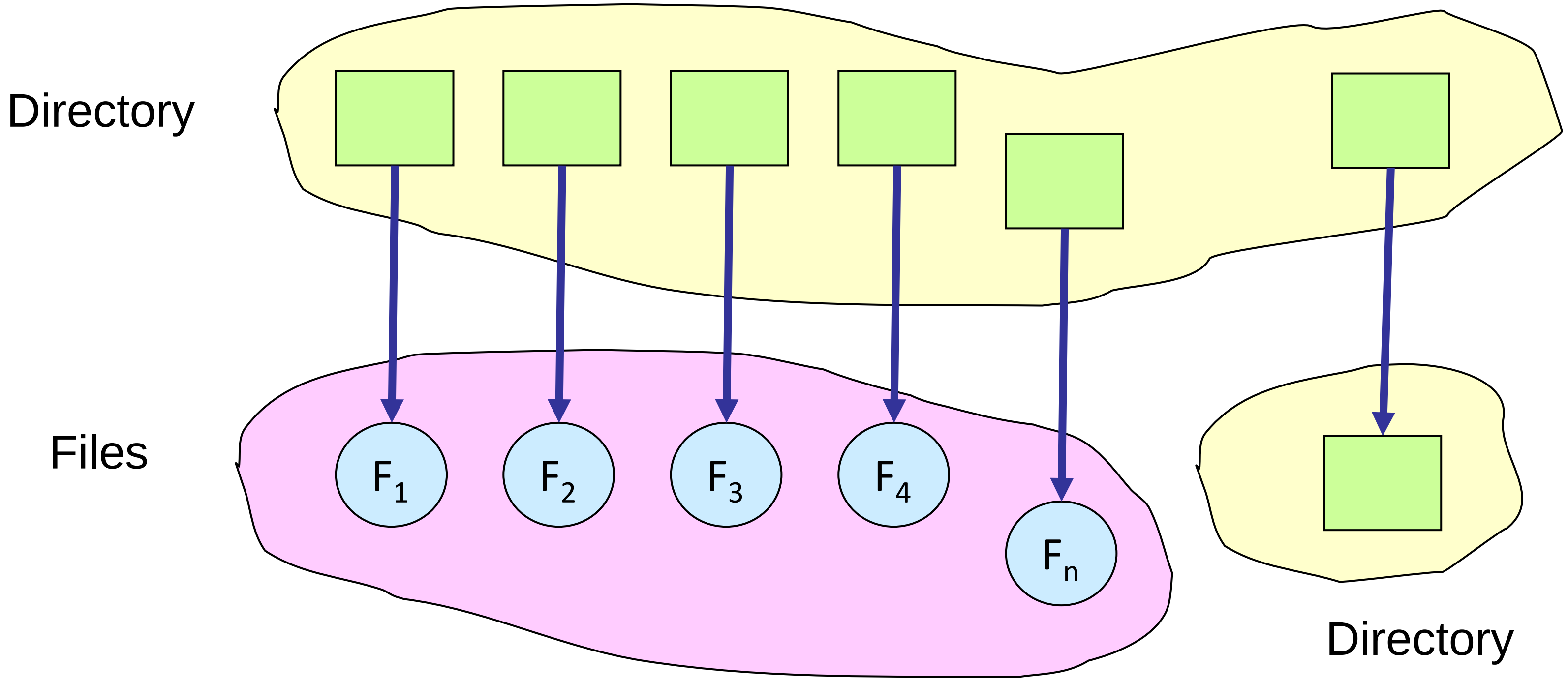
Directory Structure

How to name/locate files?

- Question: how does the user ask for a particular file?
  - One option: user specifies an inode by a number (index).
    - Imagine: open("14553344")
  - Better option: specify by textual name
    - Have to map name→inumber
  - Another option: Icon
    - This is how Apple made its money. Graphical user interfaces. Point to a file and click.
- Naming: The process by which a system translates from user-visible names to system resources

Directory Structure

- A collection of nodes containing information about all files
- Entries in directory can be either files or directories

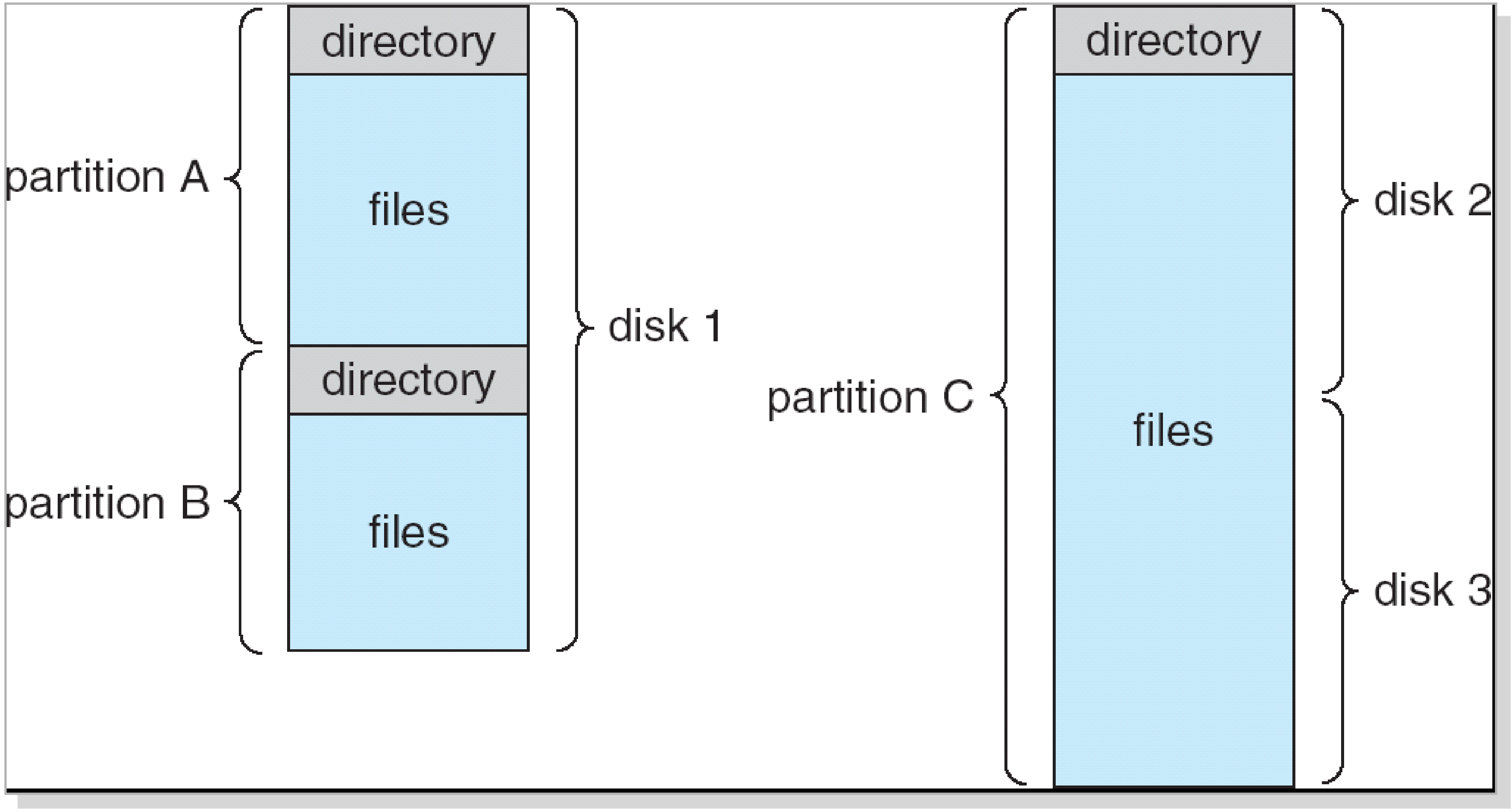


Both the directory structure and the files reside on disk

Disk Structure

- Typical file-system organization
  - Disk
    - **partitions** (IBM: minidisk, PC: volume)
    - files (including directories)

A Typical File-system Organization



Disk Structure

- Directory structures
  - Single-level directory
  - Two-level directory
  - Tree-structured directory (more common)
  - Acyclic-graph directory (cycle-detection is expensive)

flexible
complicated

Operations Performed on Directory

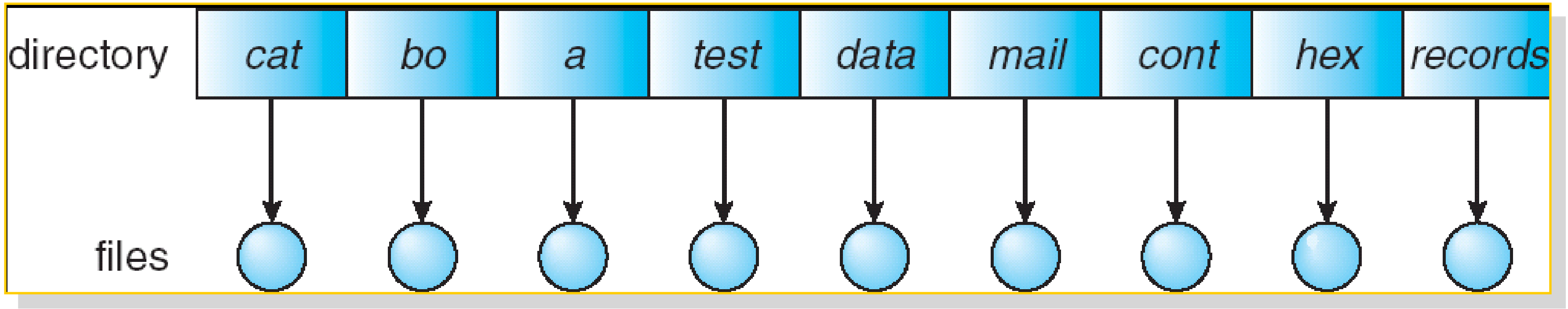
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Organize the Directory (Logically) to Obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties (e.g., all Java programs, all games, …)

Single-Level Directory

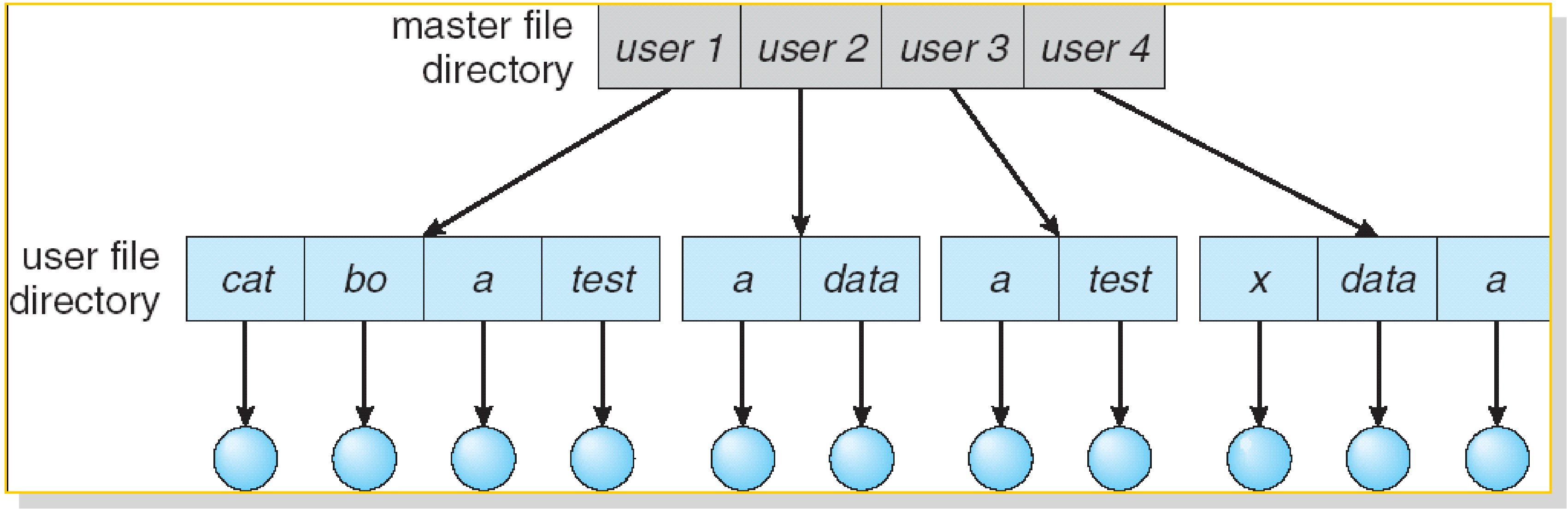
- A single directory for all users



- Naming problem
  - difficult to remember all names
  - not easy to give a new name
  - confusion between different users
- Grouping problem

Two-Level Directory

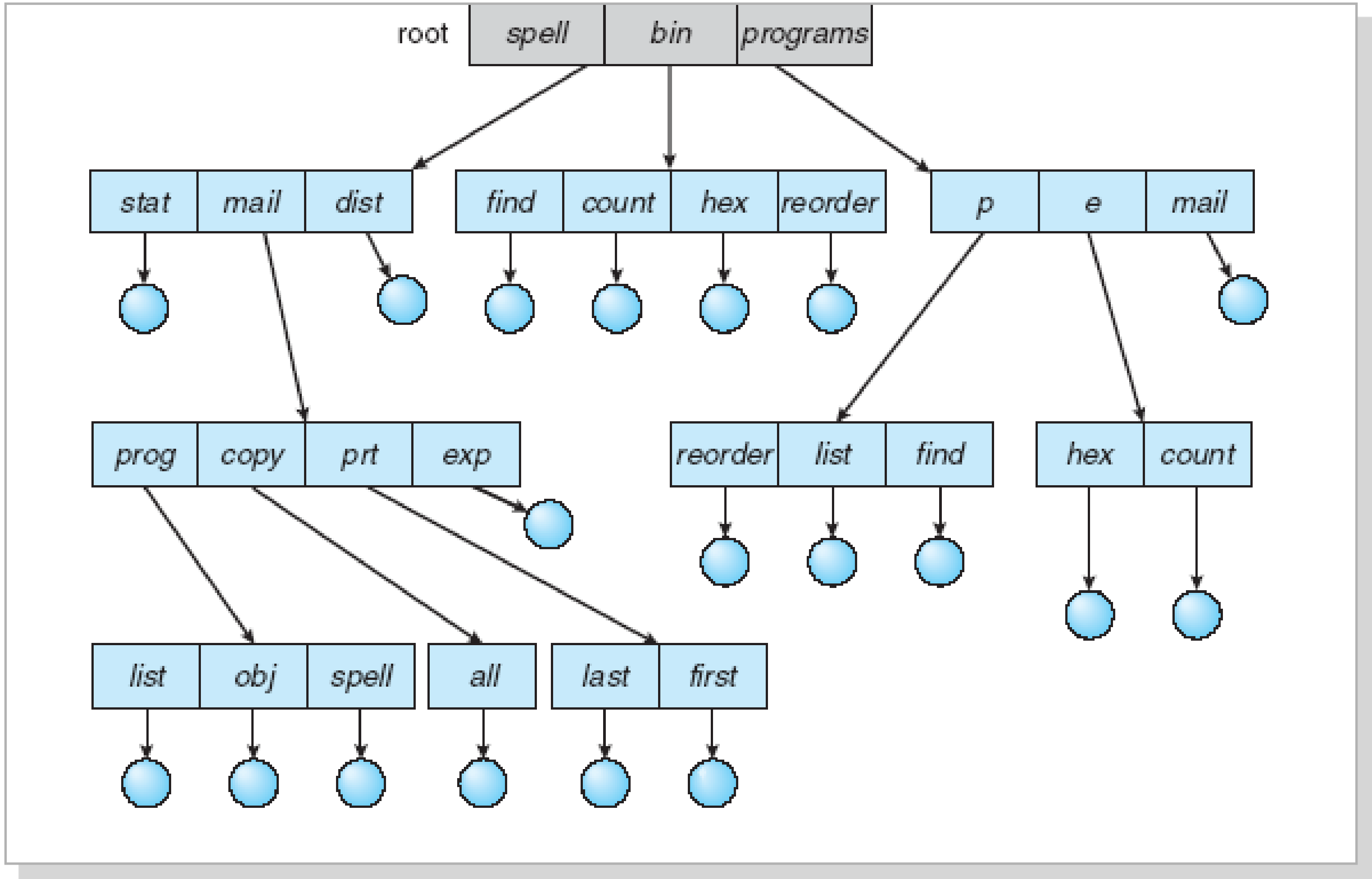
- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories

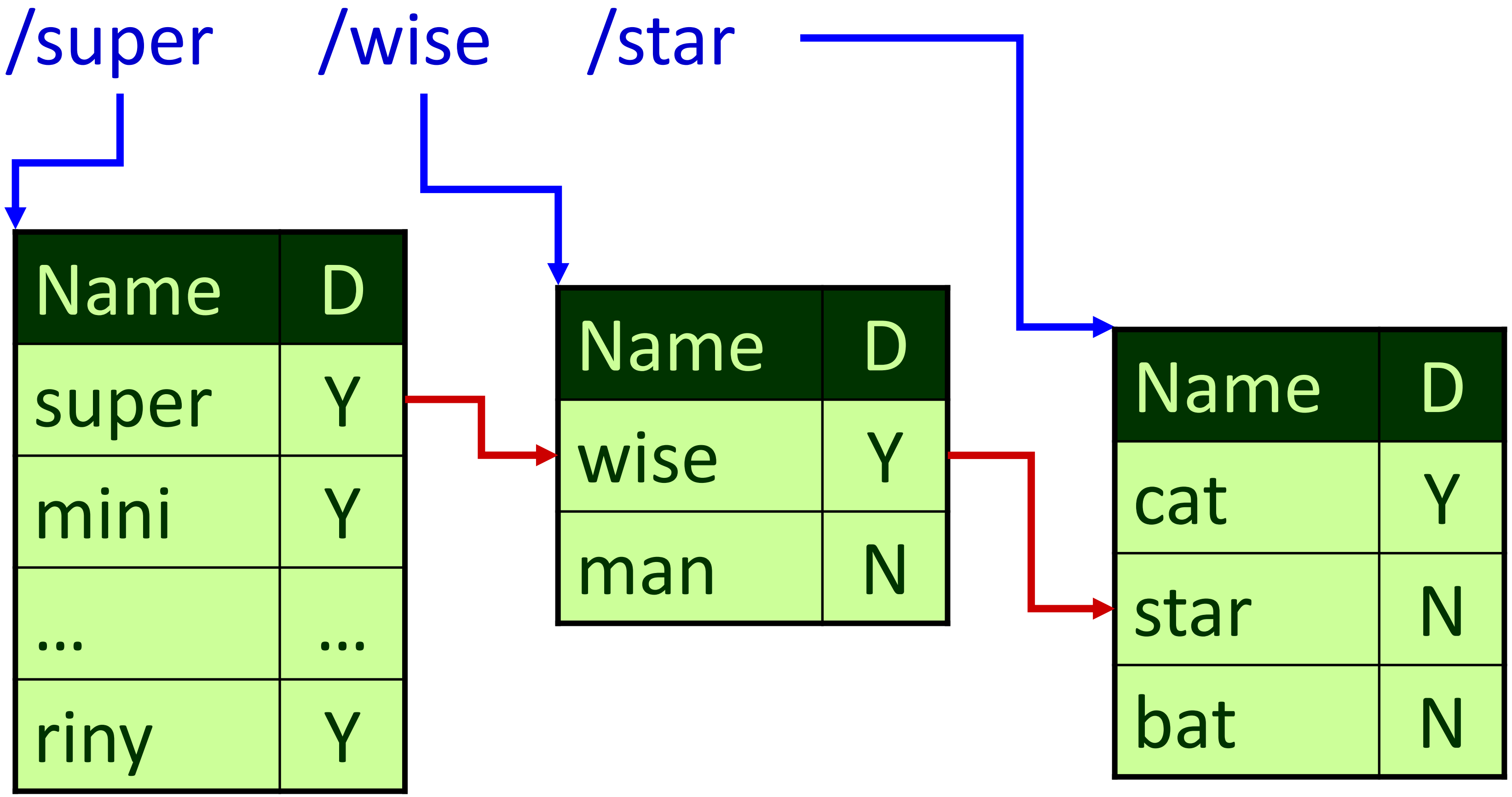
- Efficient searching/Grouping Capability



Tree-Structured Directories

- Name Resolution: The process of converting a logical name into a physical resource (like a file)
  - Traverse succession of directories until reach target file

Tree-Structured Directories

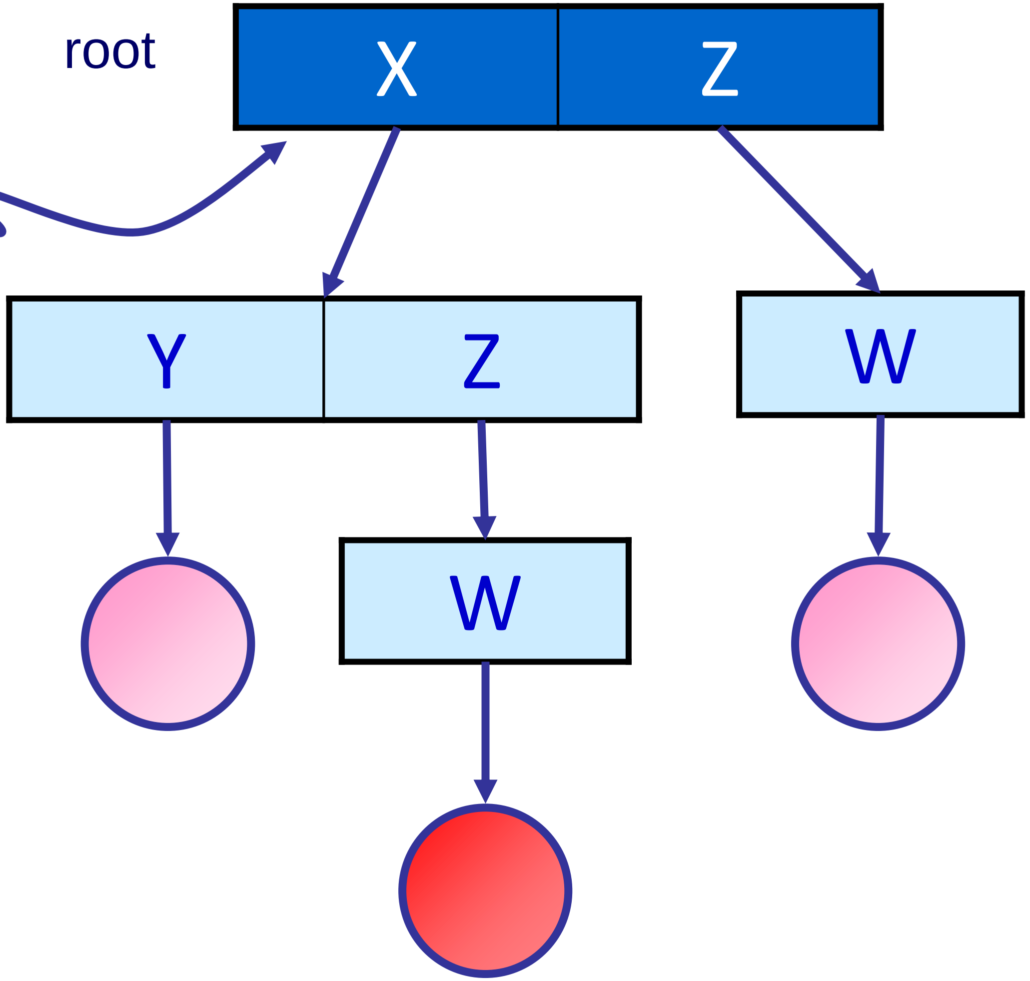


Tree-Structured Directories

- **Current working directory**: Per-address-space pointer to a directory (inode) used for resolving file names
- **Absolute** and **relative** path name

Tree-Structured Directories

Current working directory
root\x



Relative path name:

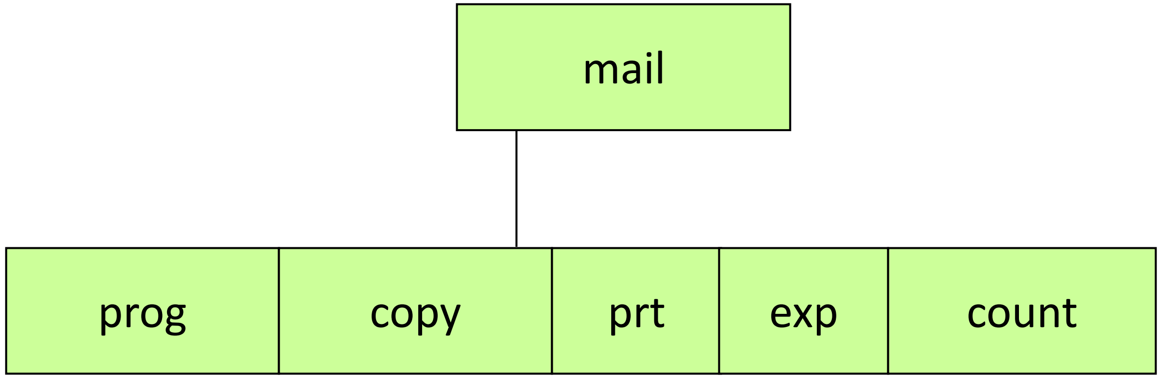
\z\w

Absolute path name:

root\x\z\w

Tree-Structured Directories

- Creating a new file/ Deleting a file is done in current directory
- Creating a new subdirectory is done in current directory
- Delete a subdirectory

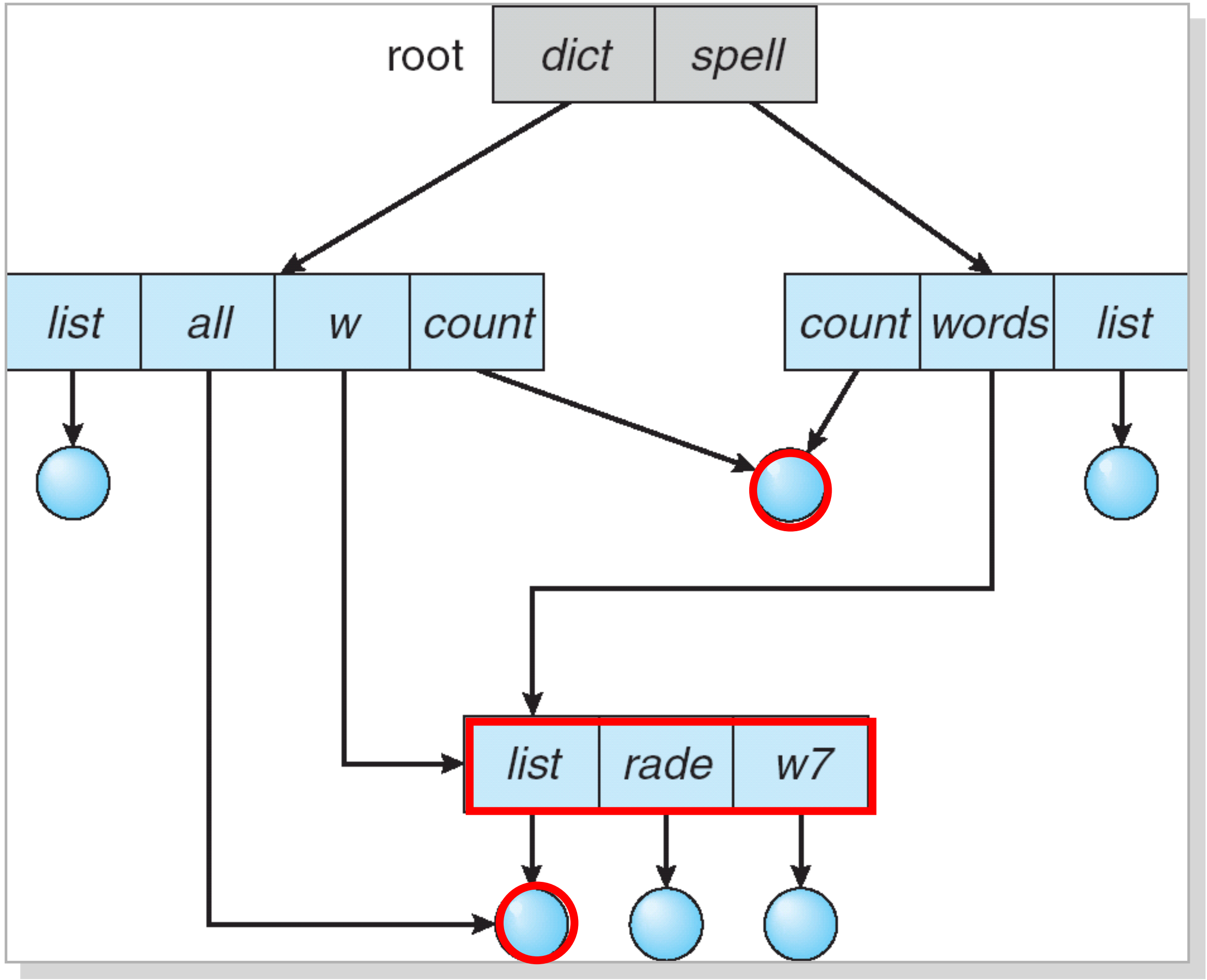


Deleting "mail"

$\Rightarrow$ deleting the entire subtree rooted by "mail" ?

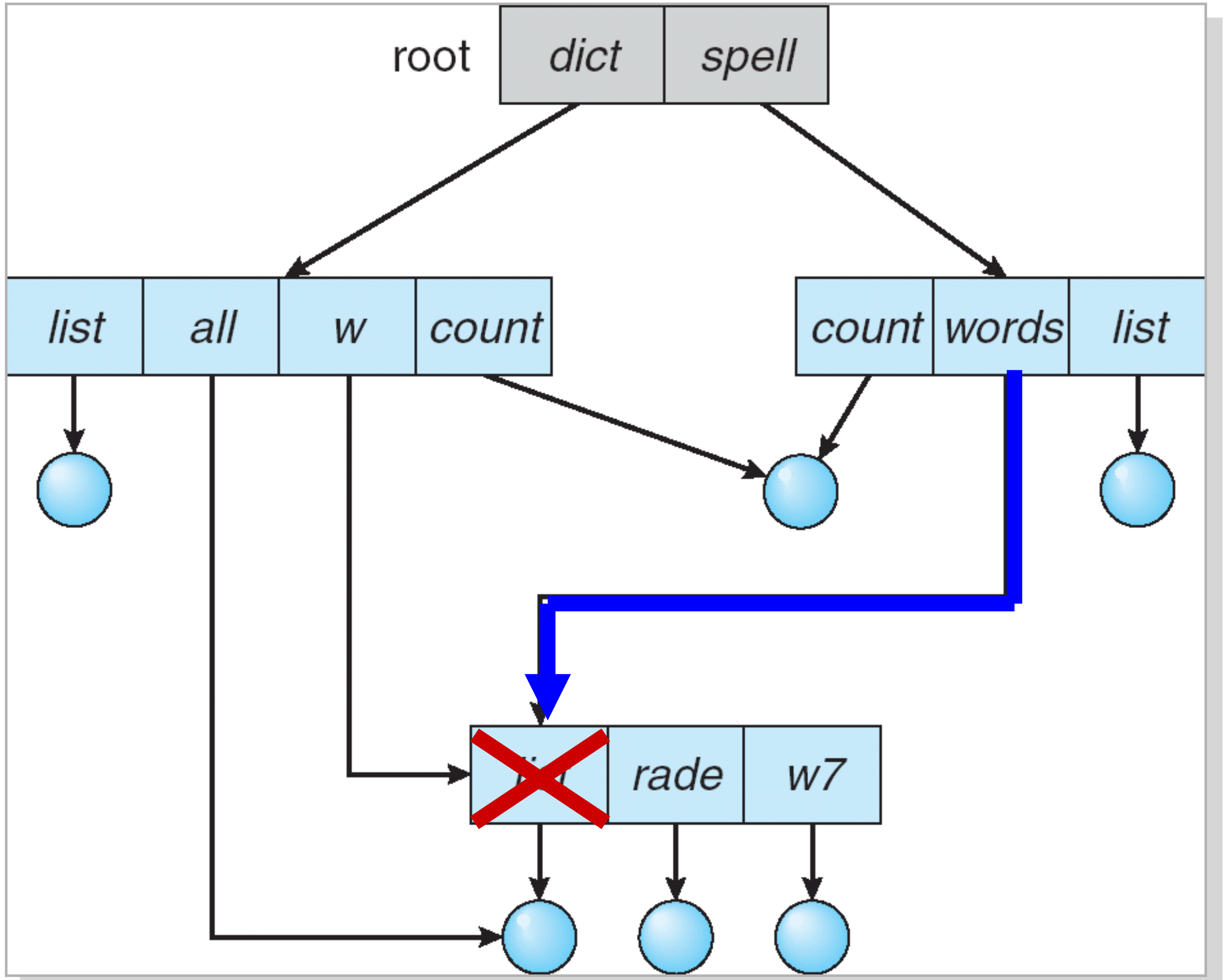
Acyclic-Graph Directories

- Have shared subdirectories and files



Acyclic-Graph Directories

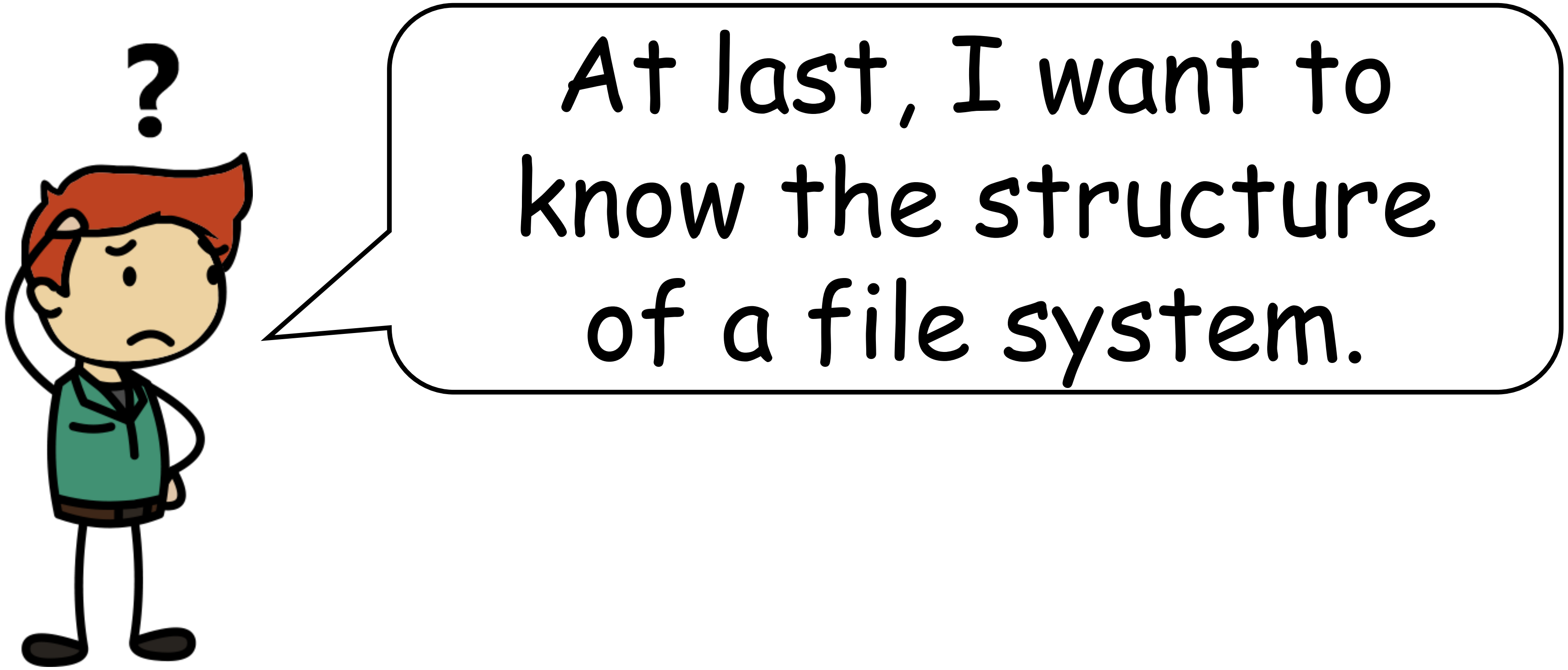
- If *dict* deletes *list* ⇒ dangling pointer



Acyclic-Graph Directories

- If *dict* deletes *list* $\Rightarrow$ dangling pointer

  Solutions:
  - Backpointers, so we can delete all pointers
  - Keep a count of the number of references


?
At last, I want to know the structure of a file system.

Layered File System

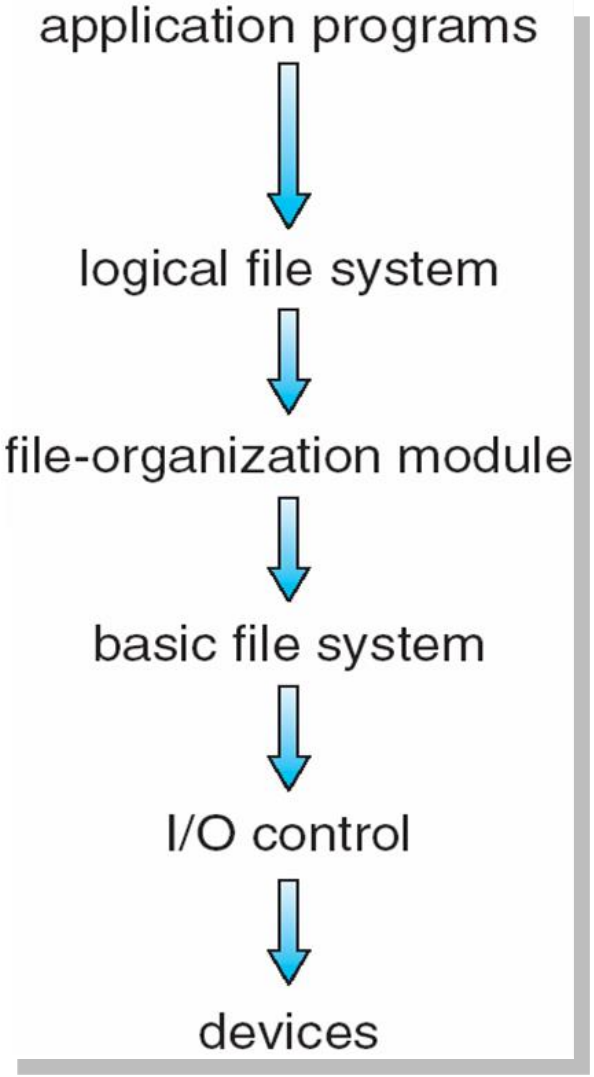
Layered File System

file name
write(data_file, item)

information
of data_file

2144th block

driver 1, cylinder 73,
surface 2, sector 10



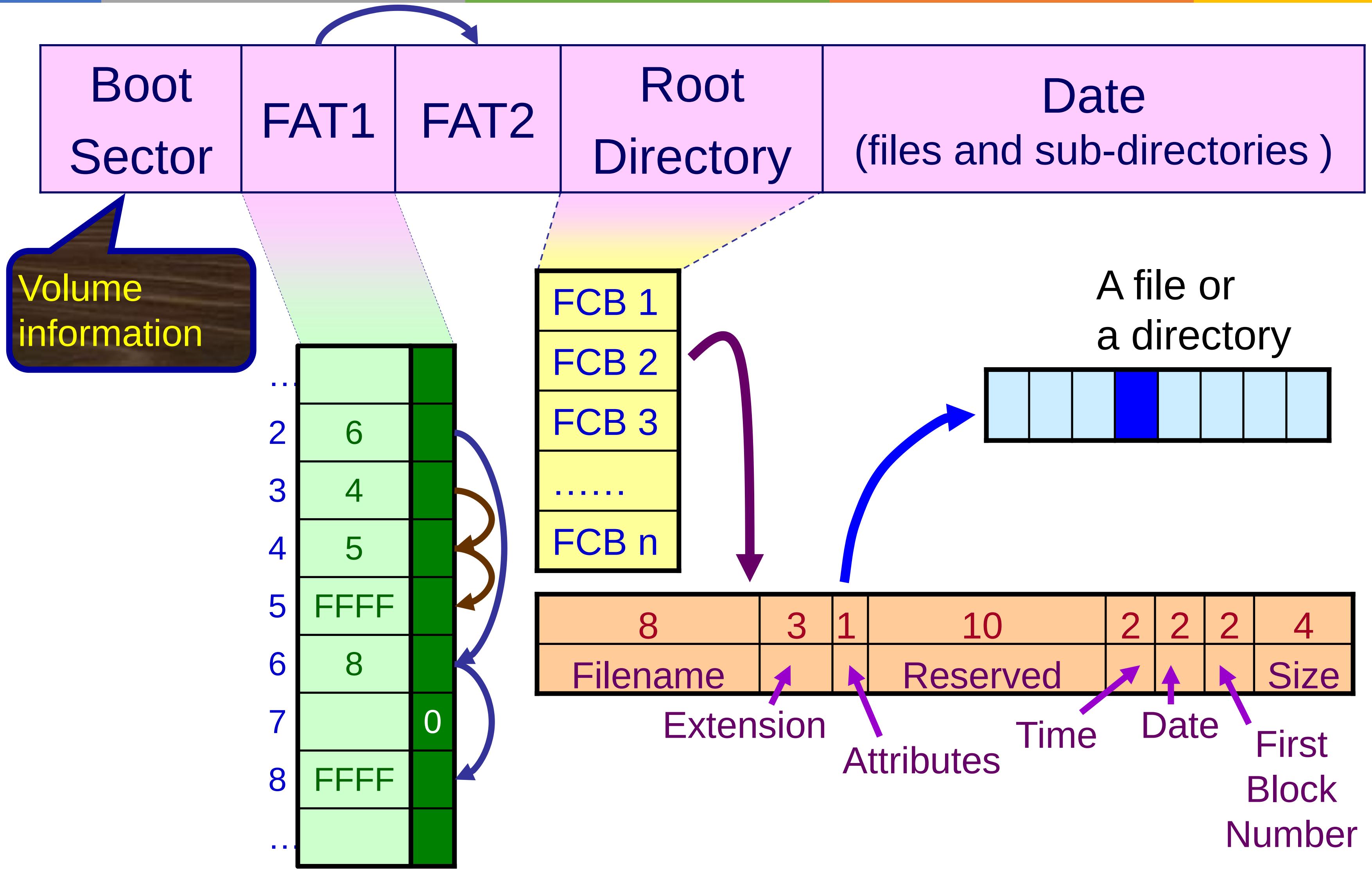
Given a symbolic file name, use directory to provide values needed by FOM

Transform logical address to physical block address

Numeric disk address

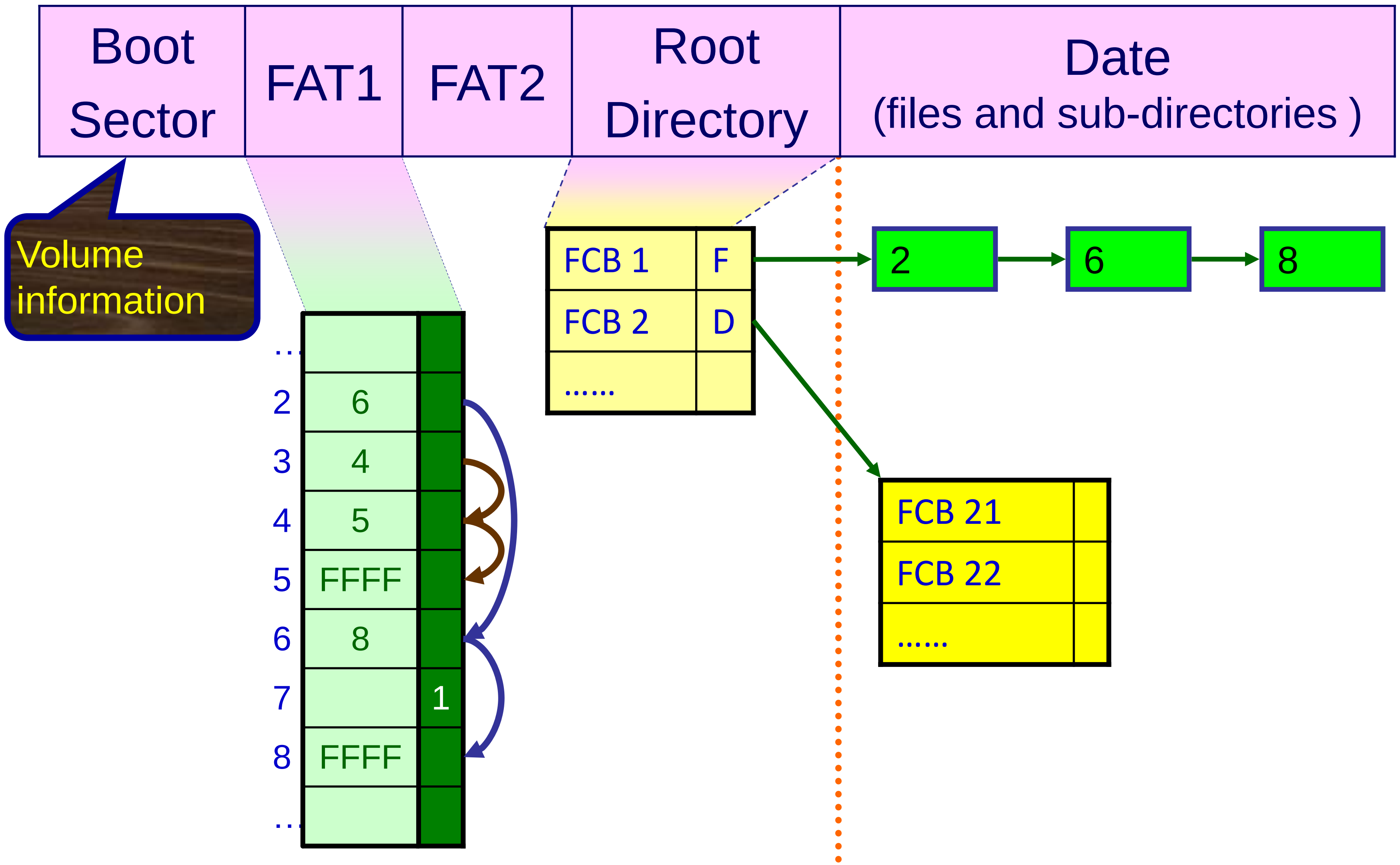
Driver: hardware instructions

Example: DOS



Boot Sector
FAT1
FAT2
Root Directory
Date (files and sub-directories)
Volume information
…
2 6
3 4
4 5
5 FFFF
6 8
7 0
8 FFFF
…
FCB 1
FCB 2
FCB 3
……
FCB n
A file or a directory
8 Filename
3
1
10 Reserved
2
2
2
4 Size
Extension
Attributes
Time
Date
First Block Number

Example: DOS



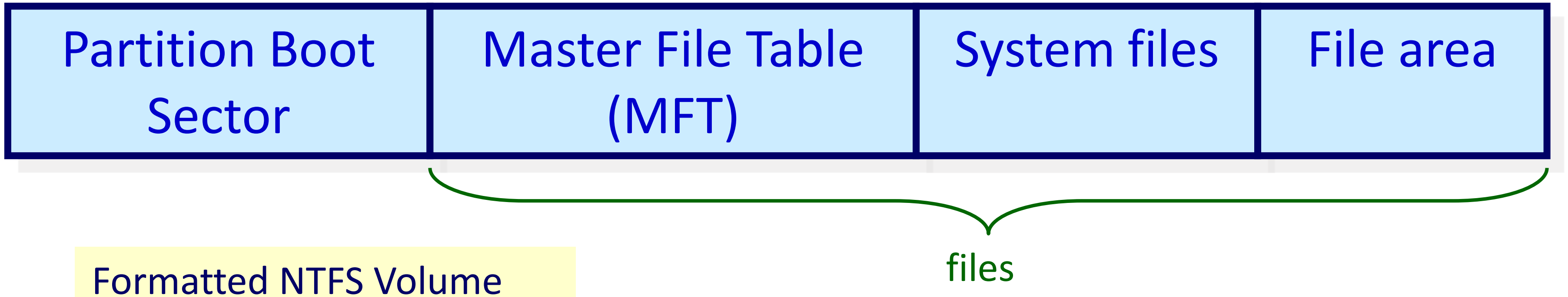
Boot Sector
FAT1
FAT2
Root Directory
Date (files and sub-directories)
Volume information
…
2 6
3 4
4 5
5 FFFF
6 8
7 1
8 FFFF
…
FCB 1 F
FCB 2 D
……
2
6
8
FCB 21
FCB 22
……

Example: NTFS

- The Windows NT file system (NTFS) provides a combination of performance, reliability, and compatibility not found in the FAT file system.
- It is designed to quickly perform standard file operations such as read, write, and search - and even advanced operations such as file-system recovery - on very large hard disks.

Example: NTFS

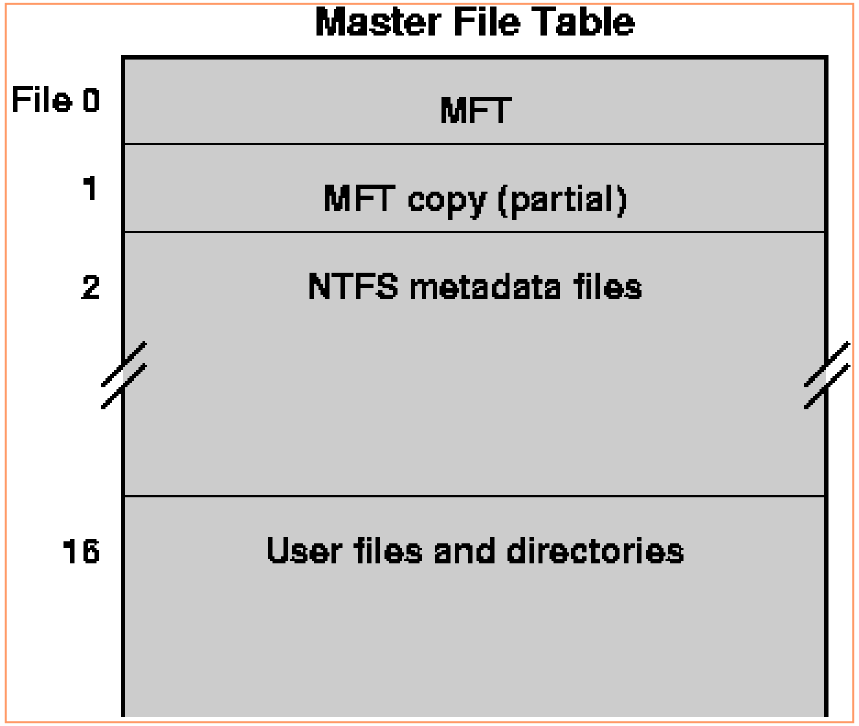
- Formatting a volume with the NTFS file system results in the creation of several system files and the Master File Table (MFT), which contains information about all the files and folders on the NTFS volume.
- The first file on an NTFS volume is the Master File Table (MFT).



Formatted NTFS Volume

Example: NTFS

- **NTFS Master File Table (MFT)**
  - Each file on an NTFS volume is represented by a record in a special file called the master file table (MFT).
  - NTFS reserves the first 16 records of the table for special information.



WIN32 APIs on Files

- CreateFile
- ReadFile
- WriteFile
- CloseHandle
- GetFileTime
- GetFileSize
- GetFileAttributes
- SetFileAttributes
- GetFileInformationByHandle
- GetFullPathName
- CopyFile
- MoveFileEx
- DeleteFile
- GetTempPath
- GetTempFileName
- SetFilePoint

- LockFile
- UnlockFile
- LockFileEx
- UnlockFileEx
- LZOpenFile
- LZSeek
- LZRead
- LZClose
- LZCopy
- GetExpandedName
- CreateFileMapping
- MapViewOfFile
- UnmapViewOfFile
- FlushViewOfFile

Summary

- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.
- File System Design Goals:
  - Maximize sequential performance
  - Easy random access to file
  - Easy management of file (growth, truncation, etc.)

Summary

- **File Concept**
- **File Structure**
  - Logical structure
    - None - sequence of words, bytes
    - Simple record structure
  - Access Methods
    - Sequential Access
    - Random Access
  - Physical structure
    - Contiguous allocation
    - Linked allocation
      - FAT
    - Indexed allocation
  - FCB
- **Directory Structure**