Chapter 11: File System Implementation

肖 卿 俊

办公室：计算机楼532室

电邮：csqjxiao@seu.edu.cn

主页：http://cse.seu.edu.cn/PersonalPage/csqjxiao

电话：025-52091023

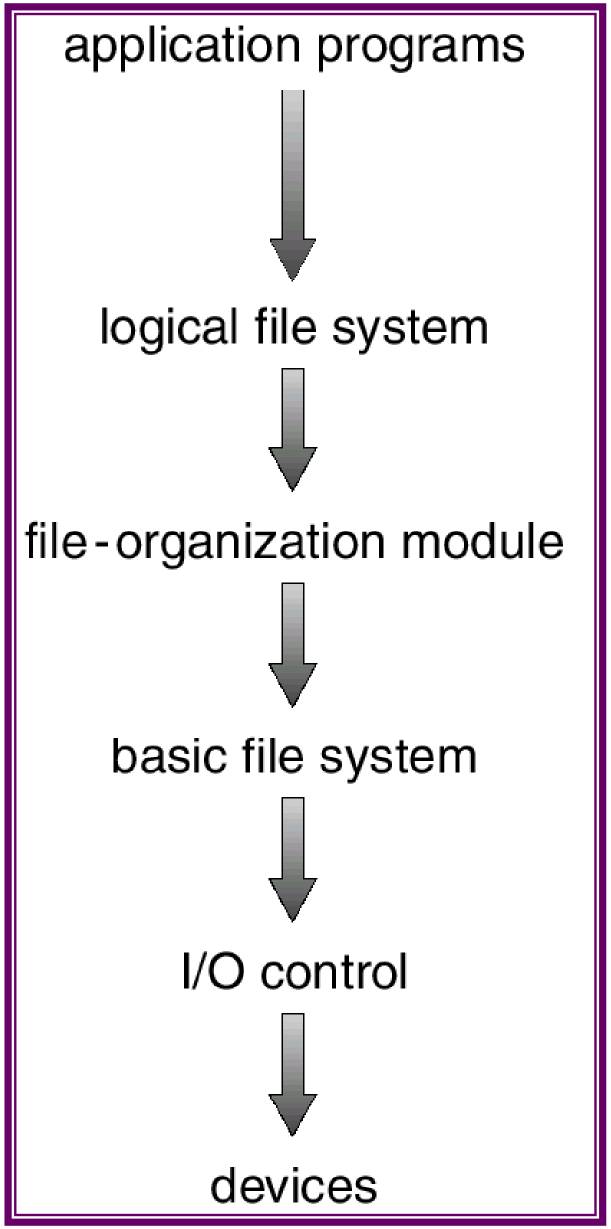
Chapter 11: File System Implementation

- File System Structure
- Free-Space Management
- File System Implementation
- Directory Implementation
- Allocation Methods
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS

File-System Structure

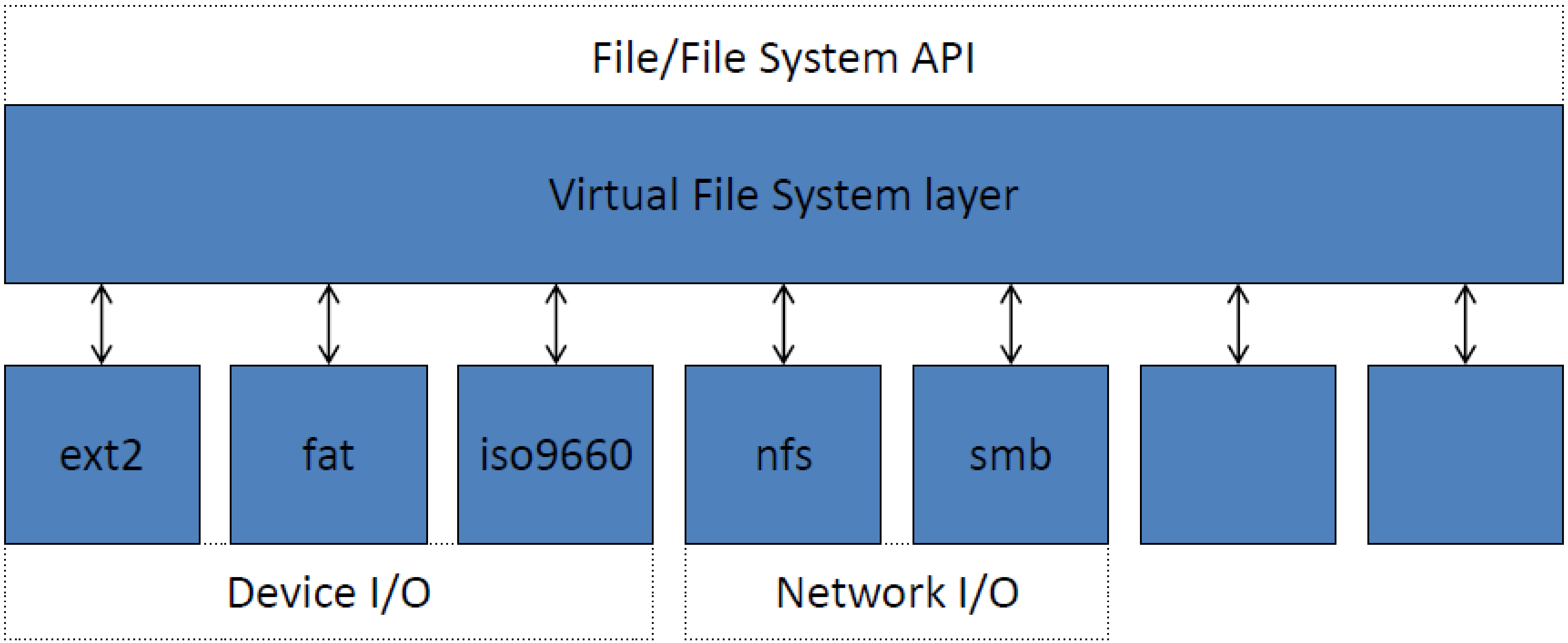
- In this chapter, "file" refers to either an ordinary file or a directory file
- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks).
- File system organized into layers.
- *File control block* – storage structure consisting of information about a file.

Layered File System



Layered File System (cont.)

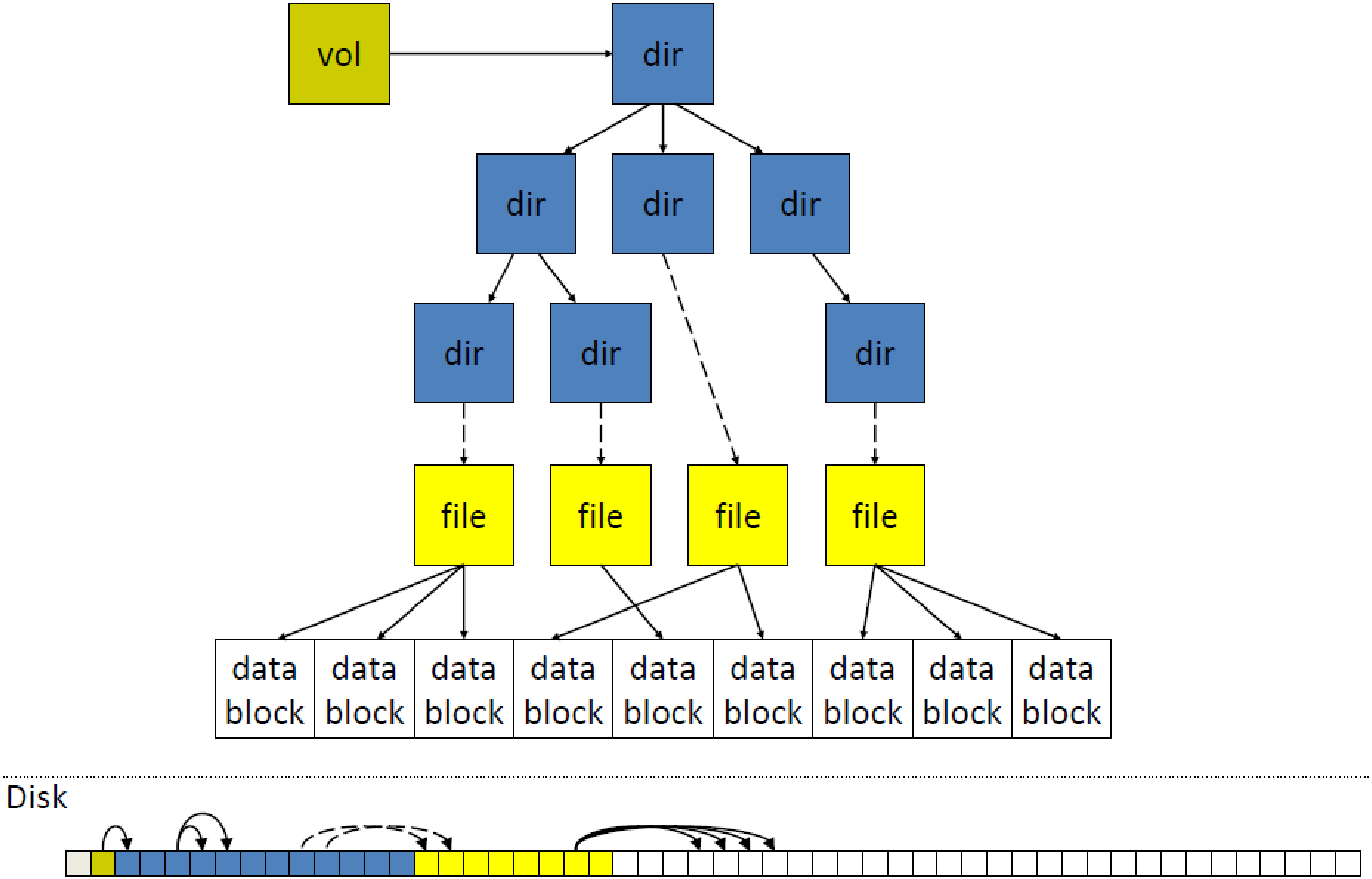
- A layered approach (remember "abstraction" + "layered approach")
  - Upper layer: virtual (logical) file system
  - Lower layer: specific file system modules



File System Basic Data Structures

- Volume Control Block (Unix: "superblock")
  - One per file system
  - Detail information about the file system
  - # of blocks, block size, free-block count/pointer, etc.
- File Control Block (Unix: "vnode" or "inode")
  - One per file
  - Detail information about the file
  - Permission, owner, size, data block locations, etc.
- Directory Node (Linux: "dentry")
  - One per directory entry (directory or file)
  - A tree data structure to encode the directory structure and tree layout
  - Pointer to file control block, parent, list of entries, etc.

Abstract View



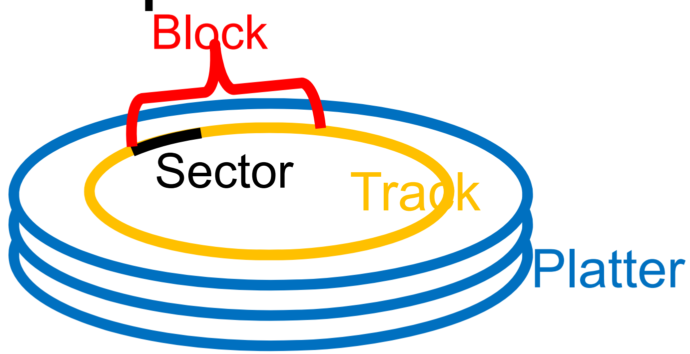
http://www.cs.nmsu.edu/~pfeiffer/classes/474/notes/inodefs.html

Chapter 11: File System Implementation

- File System Structure
- Free-Space Management
- File System Implementation
- Directory Implementation
- Allocation Methods
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS

Free-Space Management

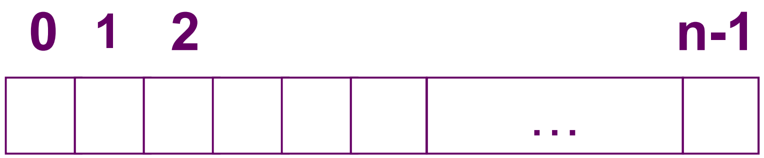
- How do we keep track free blocks on a disk?



- The techniques below are commonly used:
  - Bit Vector
  - Linked List: A free-list is maintained. When a new block is requested, we search this list to find one.
  - Linked List + Grouping
  - Linked List + Address + Count

Bit Vector

- Bit vector (*n* blocks)



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

Block number calculation

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

Free-Space Management

- Advantage of bit vector method: Easy to get contiguous files

- Disadvantage: Bitmap requires extra space.
- An Example:

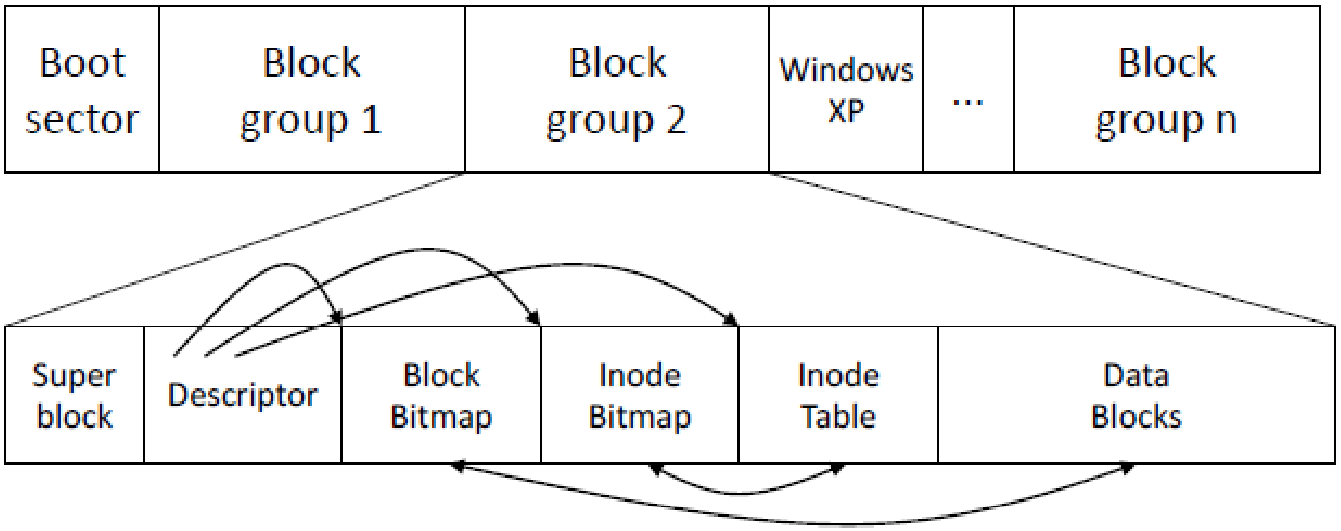
  block size = 2^{12} bytes

  disk size = 2^{30} bytes (1 gigabyte)

  $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

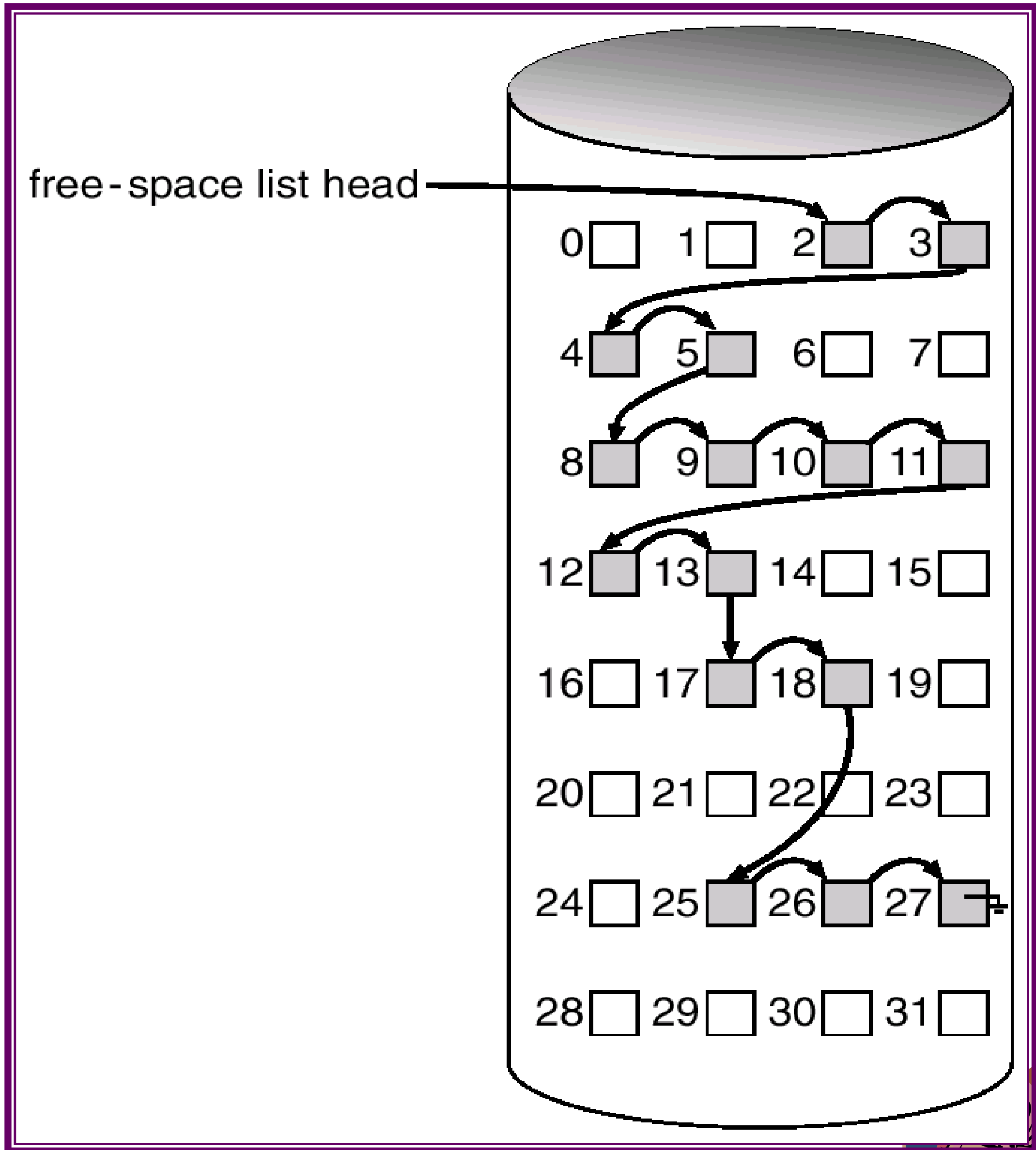
Ext2 Disk Layout

- Block bitmap is used by Ext2 to manage the disk free space.



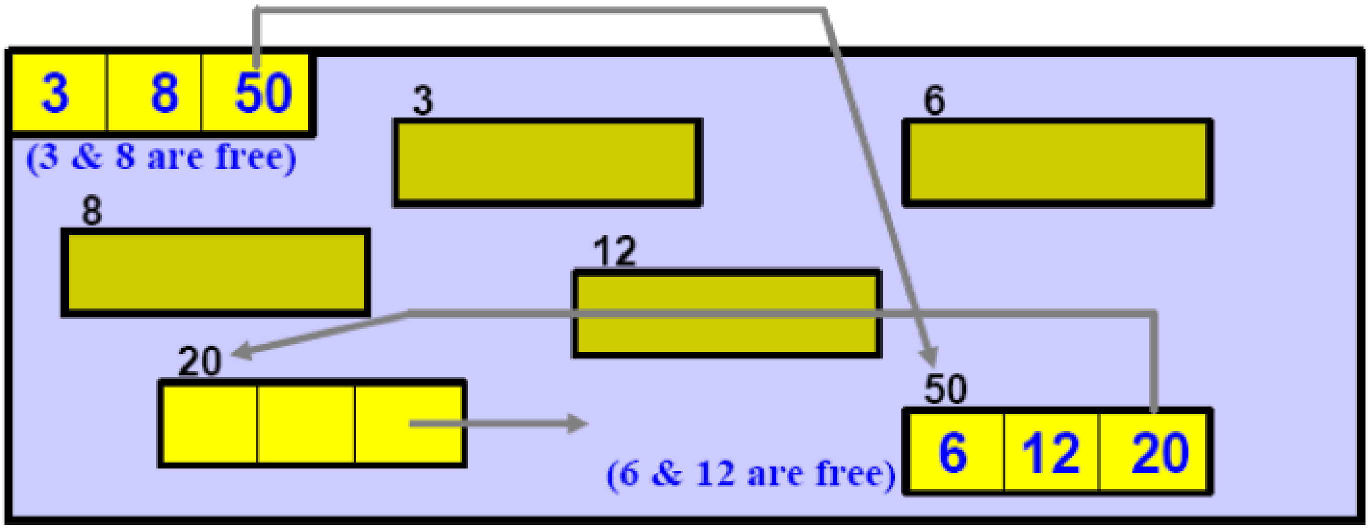
Linked Free Space List on Disk

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space



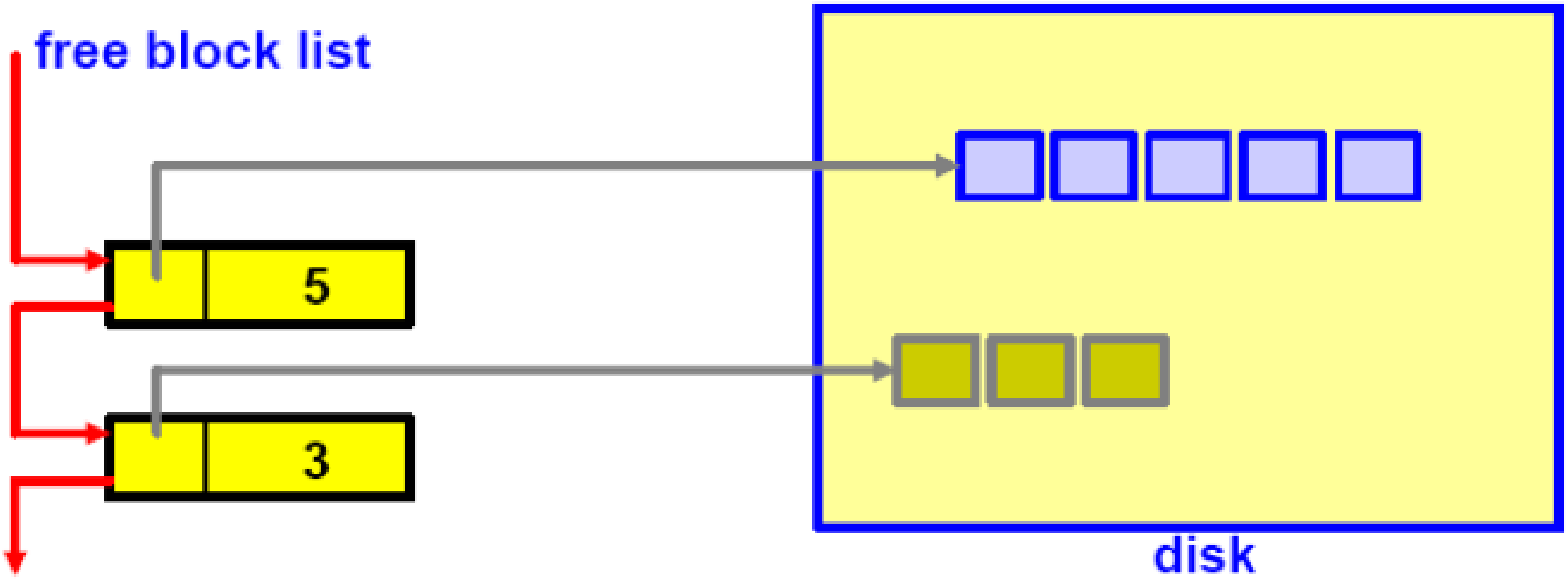
Grouping

- The first free block contains the addresses of *n* other free blocks.
- For each group, the first *n-1* blocks are actually free and the last (i.e., *n-th*) block contains the addresses of the next group.
- In this way, we can quickly locate free blocks.



Address Counting

- We can make the list short with the following trick:
  - Blocks are often allocated and freed in groups
  - We can store the address of the first free block and the number of the following n free blocks.



Chapter 11: File System Implementation

- File System Structure
- Free-Space Management
- File System Implementation
- Directory Implementation
- Allocation Methods
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS

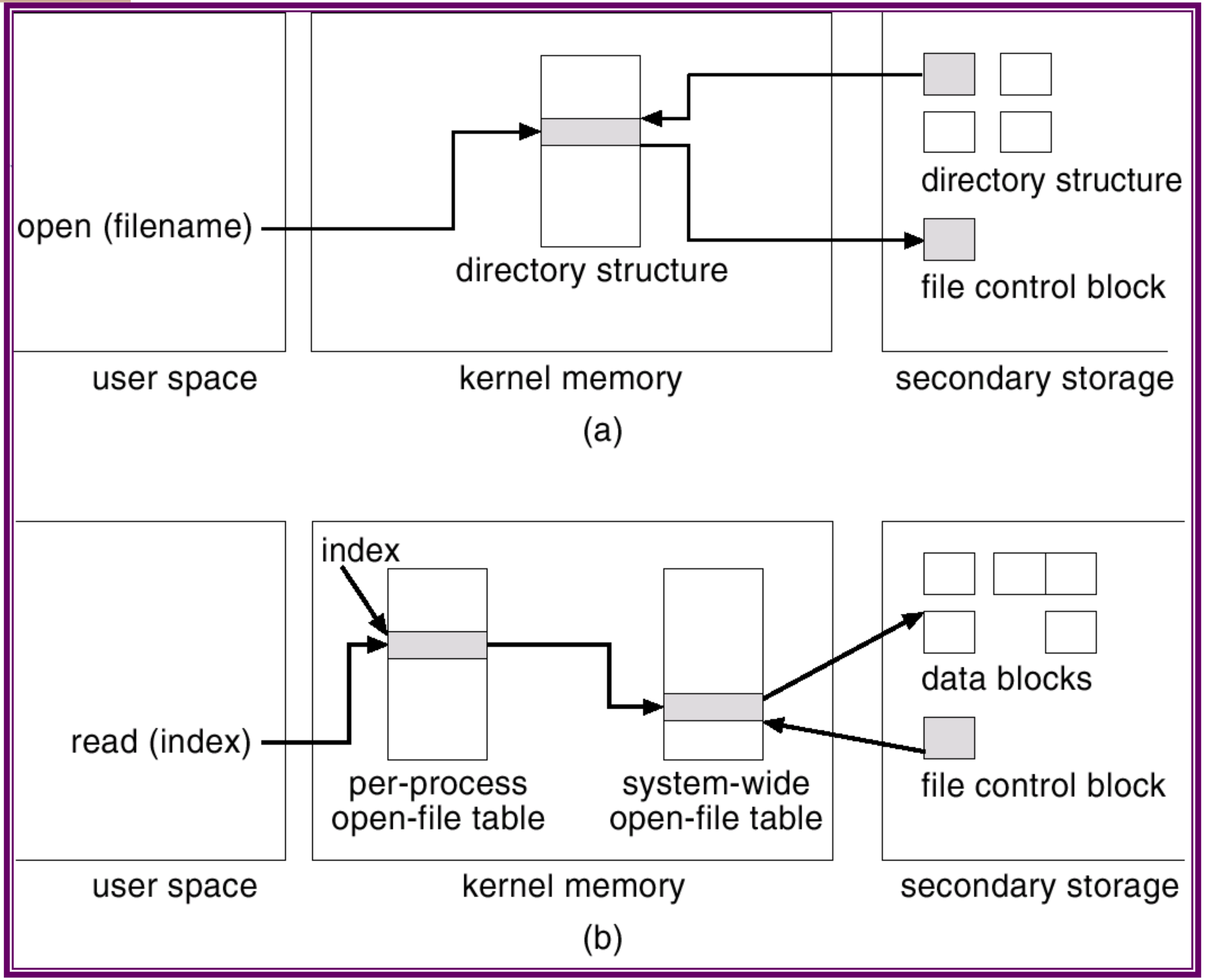
A Typical File Control Block

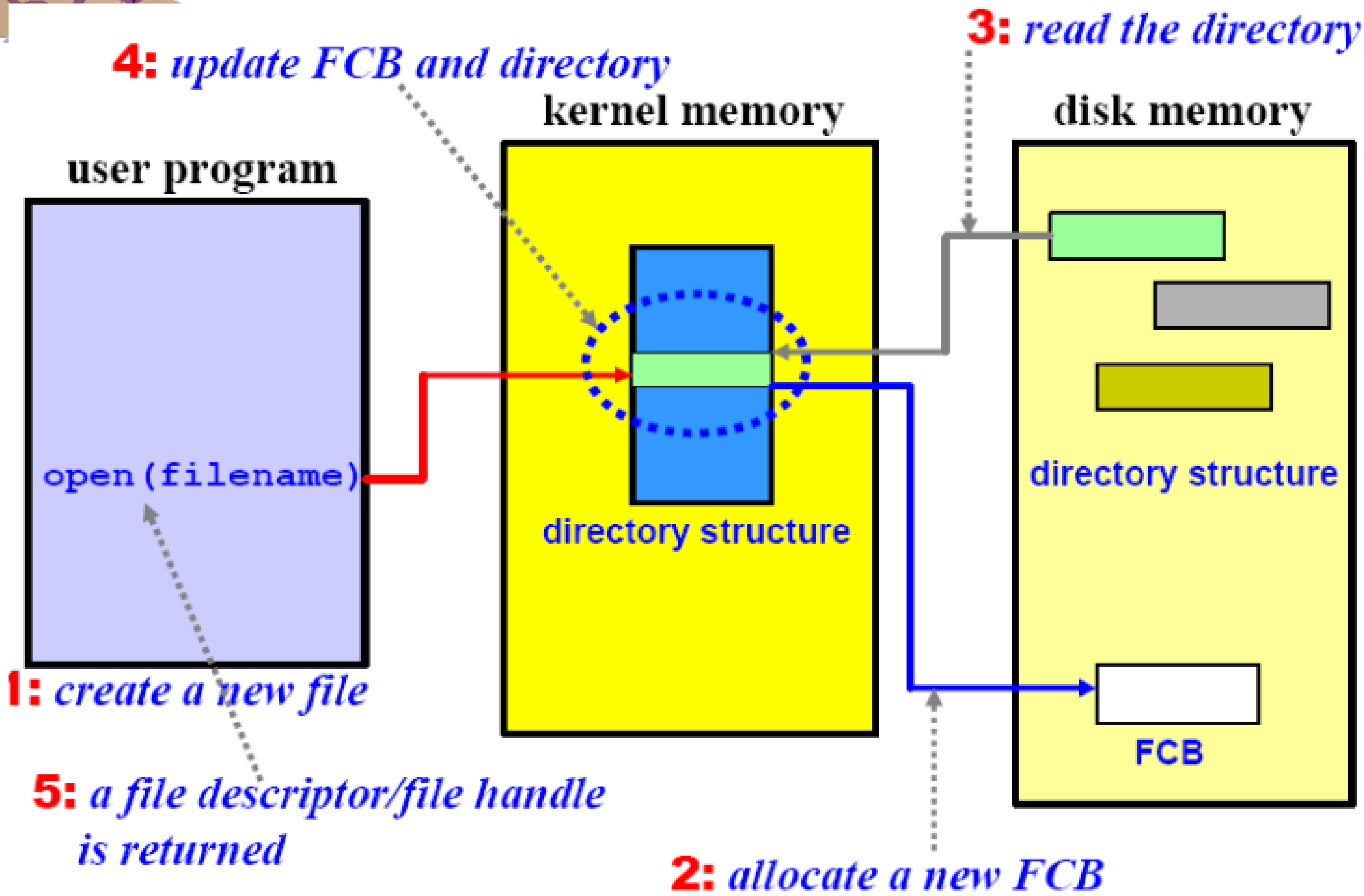
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks

In-Memory File System Structures

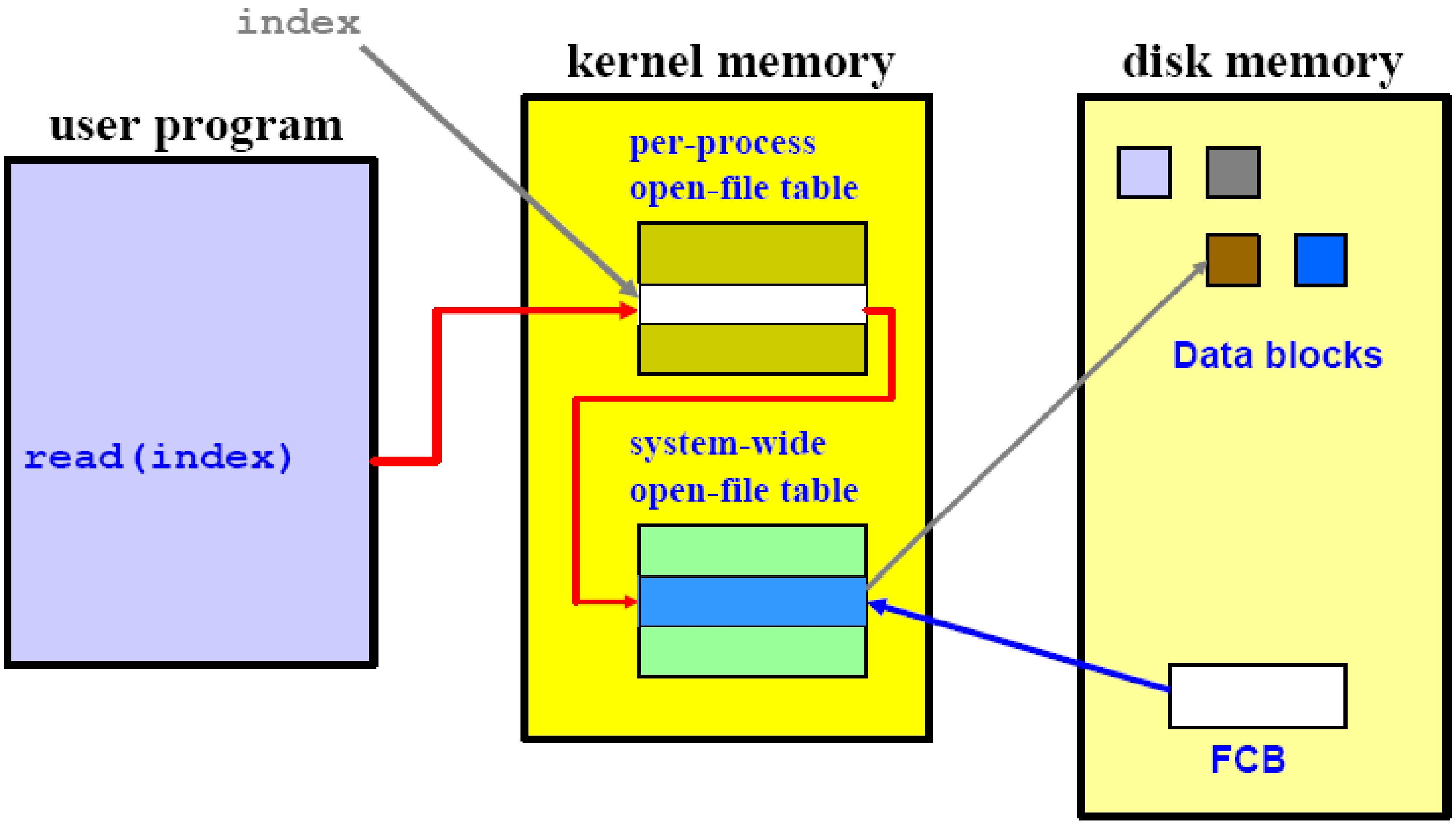
- The following figure illustrates the necessary file system structures provided by the operating systems.
- Figure 12-3(a) refers to opening a file.
- Figure 12-3(b) refers to reading a file.

In-Memory File System Structures





4: update FCB and directory
3: read the directory
kernel memory
disk memory
user program
open(filename)
directory structure
directory structure
FCB
1: create a new file
5: a file descriptor/file handle is returned
2: allocate a new FCB


index
kernel memory
disk memory
user program
per-process
open-file table
read(index)
Data blocks
system-wide
open-file table
FCB

On-demand Loading into Main Memory

- Loaded to memory when needed
  - Volume control block: in memory if file system is mounted
  - File control block: if the file is accessed
  - Directory node: during traversal of a file path

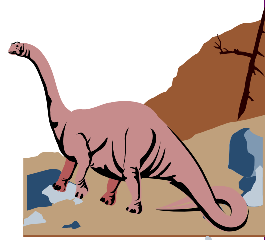
Chapter 11: File System Implementation

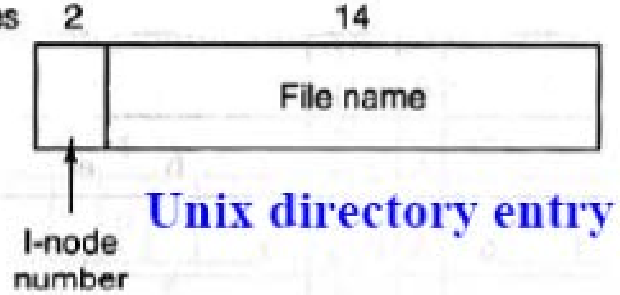
- File System Structure
- Free-Space Management
- File System Implementation
- Directory Implementation
- Allocation Methods
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS

Directory Implementation

- Linear list of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute

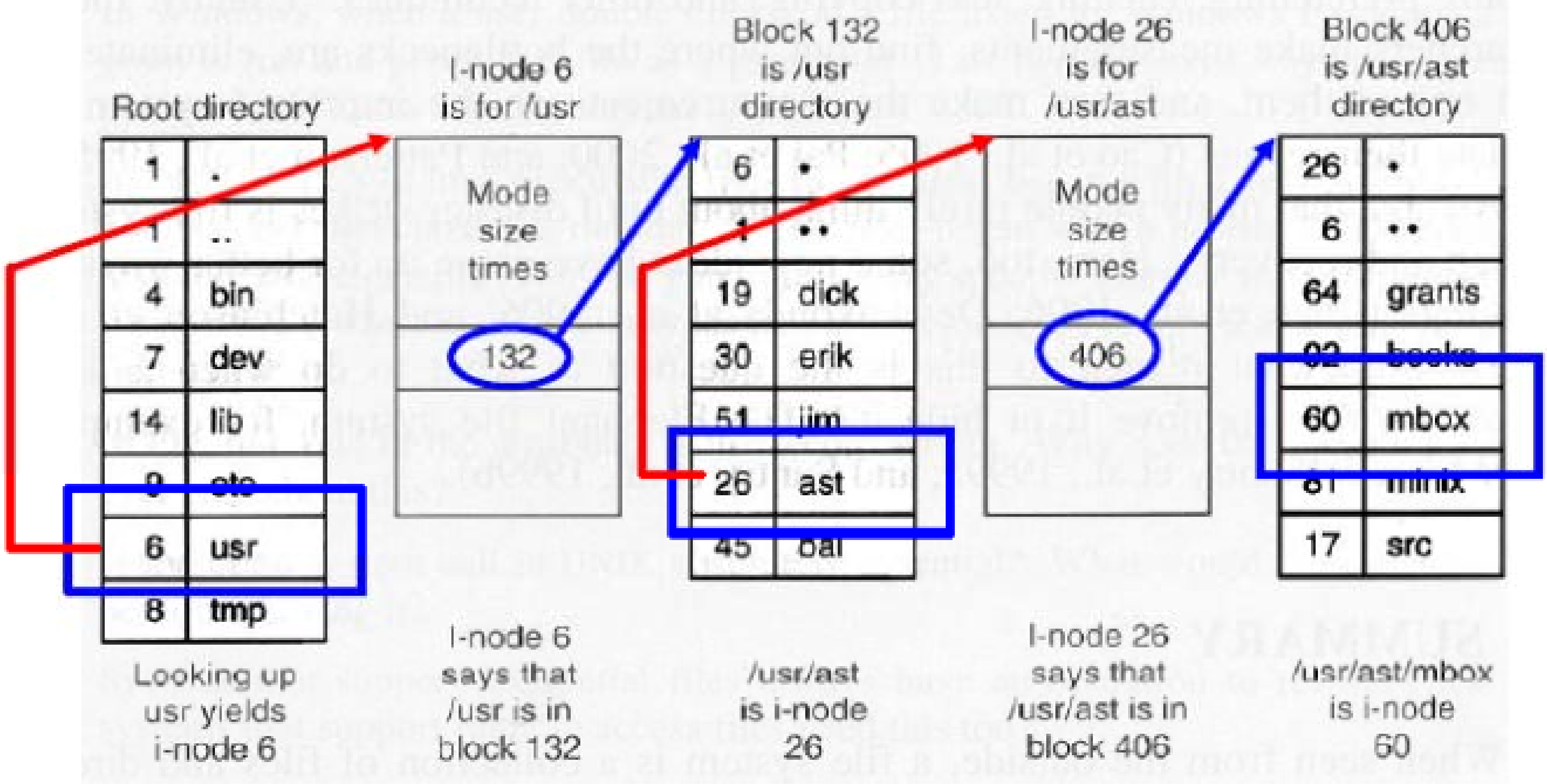
- Hash Table – linear list with hash data structure.
  - decreases directory search time
  - *collisions* – situations where two file names hash to the same location





Unix directory entry

find /usr/ast/mbox



Chapter 11: File System Implementation

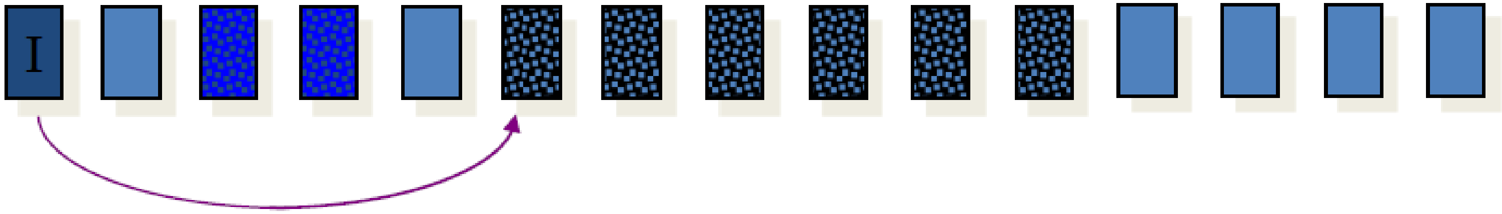
- File System Structure
- Free-Space Management
- File System Implementation
- Directory Implementation
- Allocation Methods
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS

File Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- Allocation methods
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

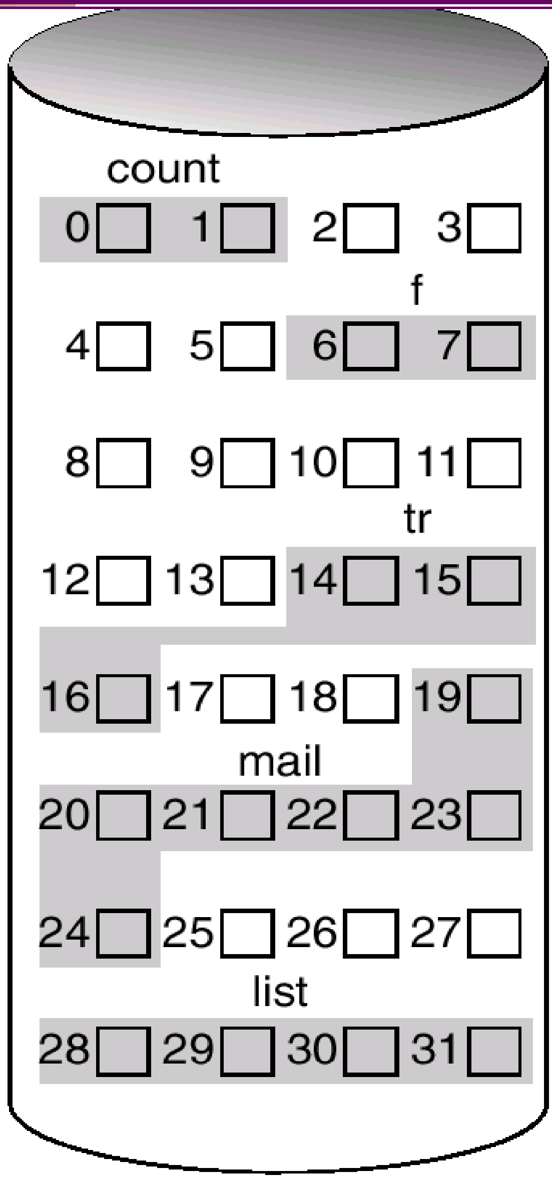
Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk.



- Simple – only starting location (block #) and length (number of blocks) are required.
- Random access.
- Wasteful of space (dynamic storage-allocation problem).
- Files cannot grow.

Contiguous Allocation of Disk Space



directory

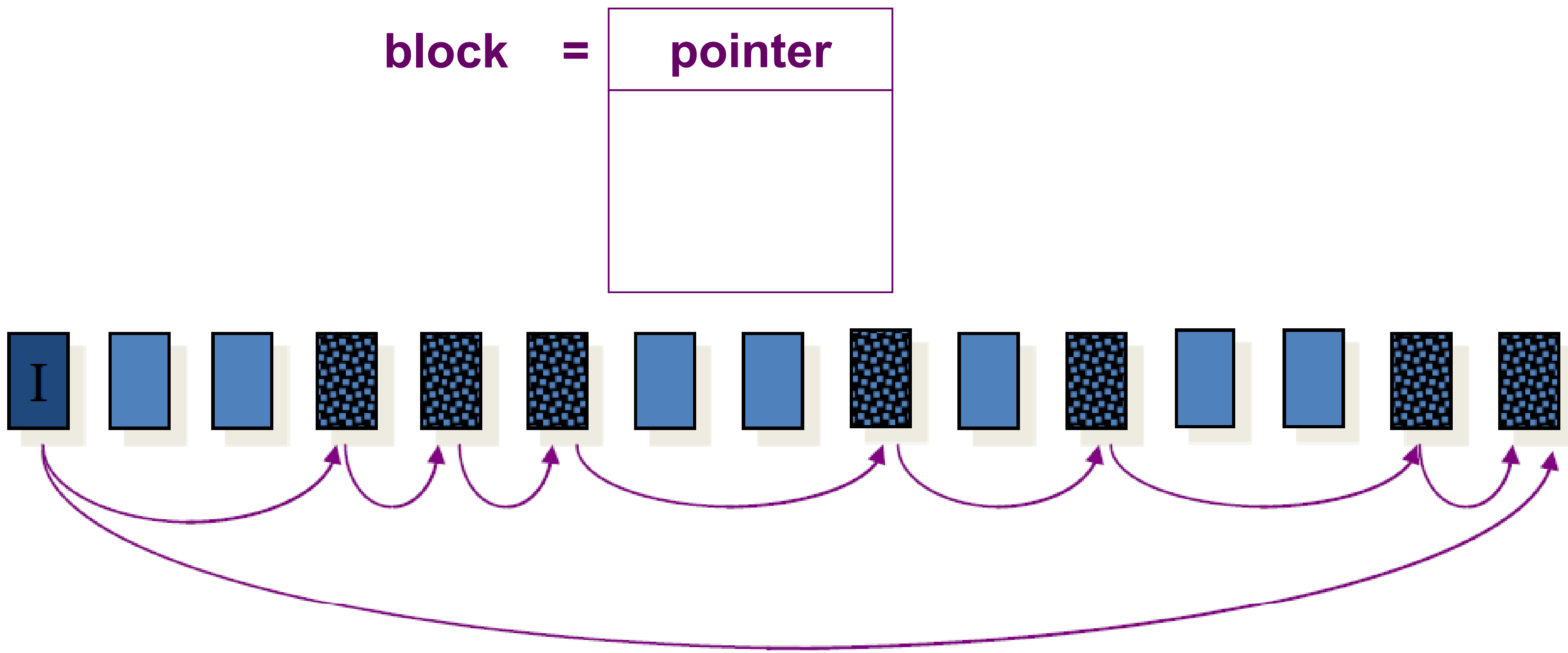
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Extent-Based Systems

- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme.
  - 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.
- Basic idea is similar to the **slab-based** kernel memory management

Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.

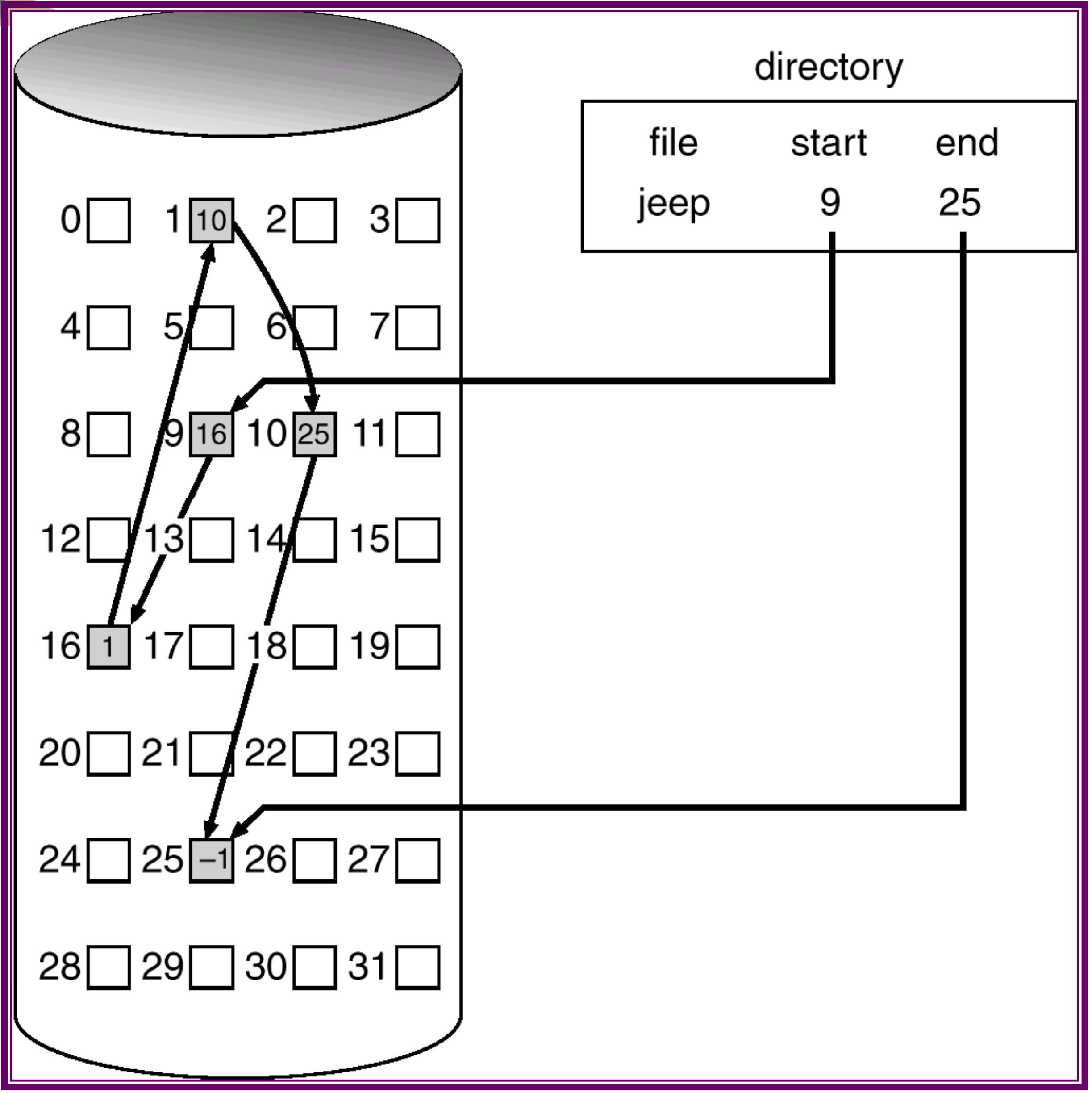


Linked Allocation (Cont.)

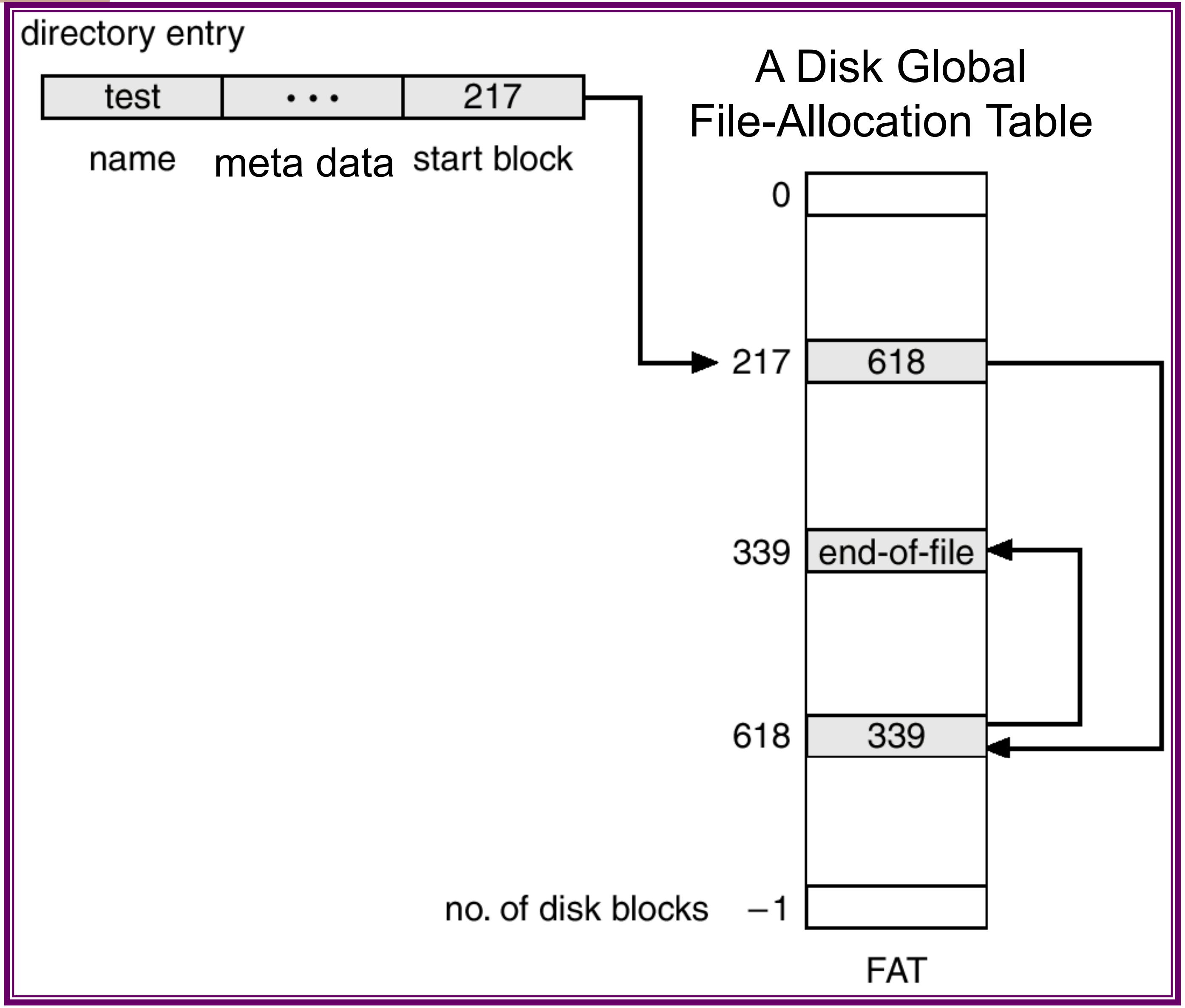
- Simple – need only starting address
- Free-space management system – no waste of space
- Files can grow

- No random access
- Each block contains a pointer, wasting space
- Blocks scatter everywhere and a large number of disk seeks may be necessary
- Reliability: what if a pointer is lost or damaged?

Linked Allocation



File-Allocation Table (FAT)



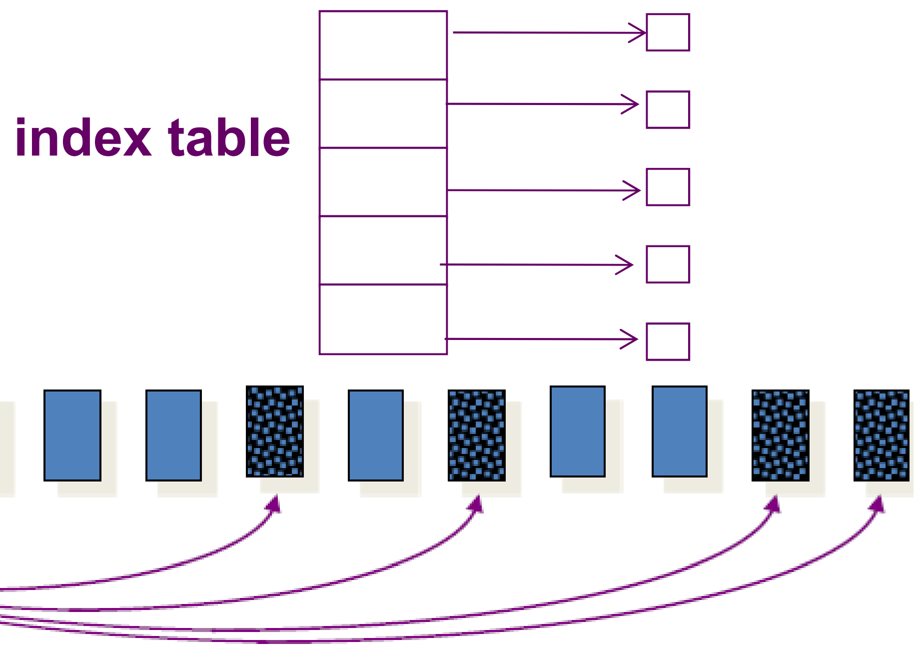
Question about FAT

- Given the values in the FAT, mark the block addresses that start a file

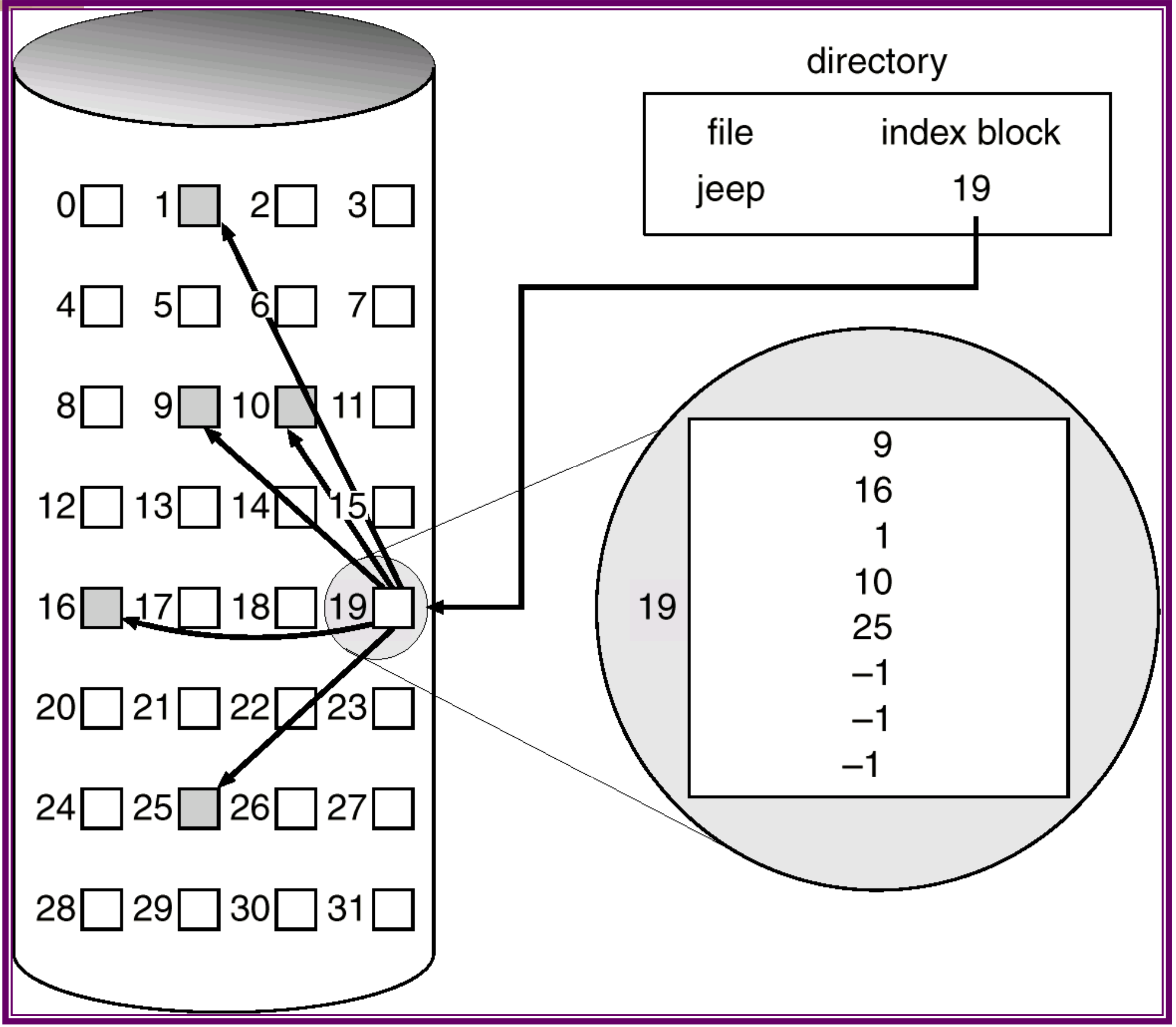
	Busy	Next	
0	0		
1	1	6	
2	1	-1	
3	1	1	✓
4	0		
5	1	-1	✓
6	1	-1	
7	1	2	✓

Indexed Allocation

- Brings all pointers together into the *index block.*
- A file's directory entry contains a pointer to its index block.
- Hence, the index block of an indexed allocation plays the same role as the page table.
- Logical view.



Example of Indexed Allocation

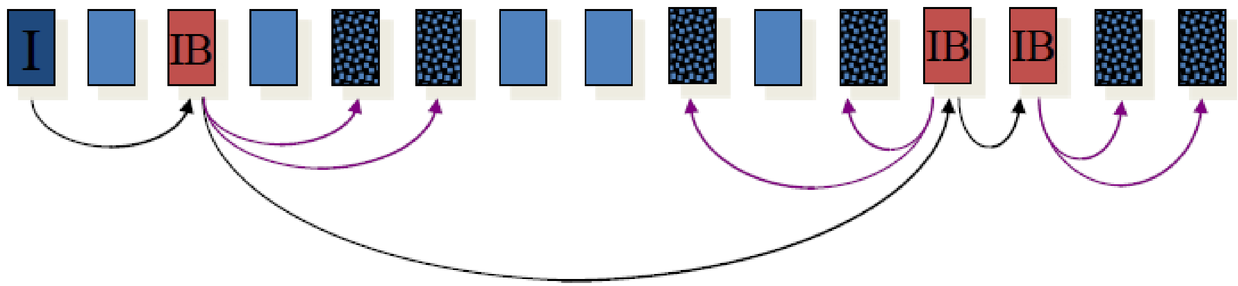


Indexed Allocation (cont.)

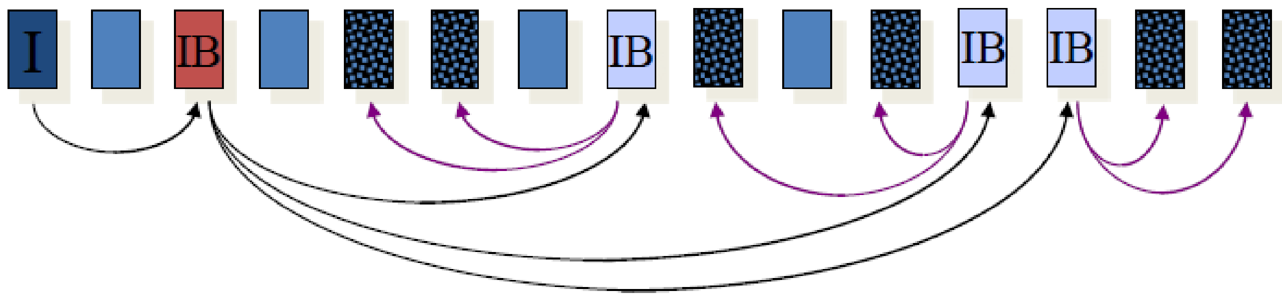
- Support the random access
- The indexed allocation suffers from wasted space. The index block may not be fully used (i.e., internal fragmentation).
- The number of entries of an index table determines the upper bound for the size of a file. But the file size can be extra large.
- To overcome this problem, we must extend the indexed allocation method.

Indexed Allocation (cont.)

- Index Allocation for Large File
  - multiple index blocks, chain them into a linked-list

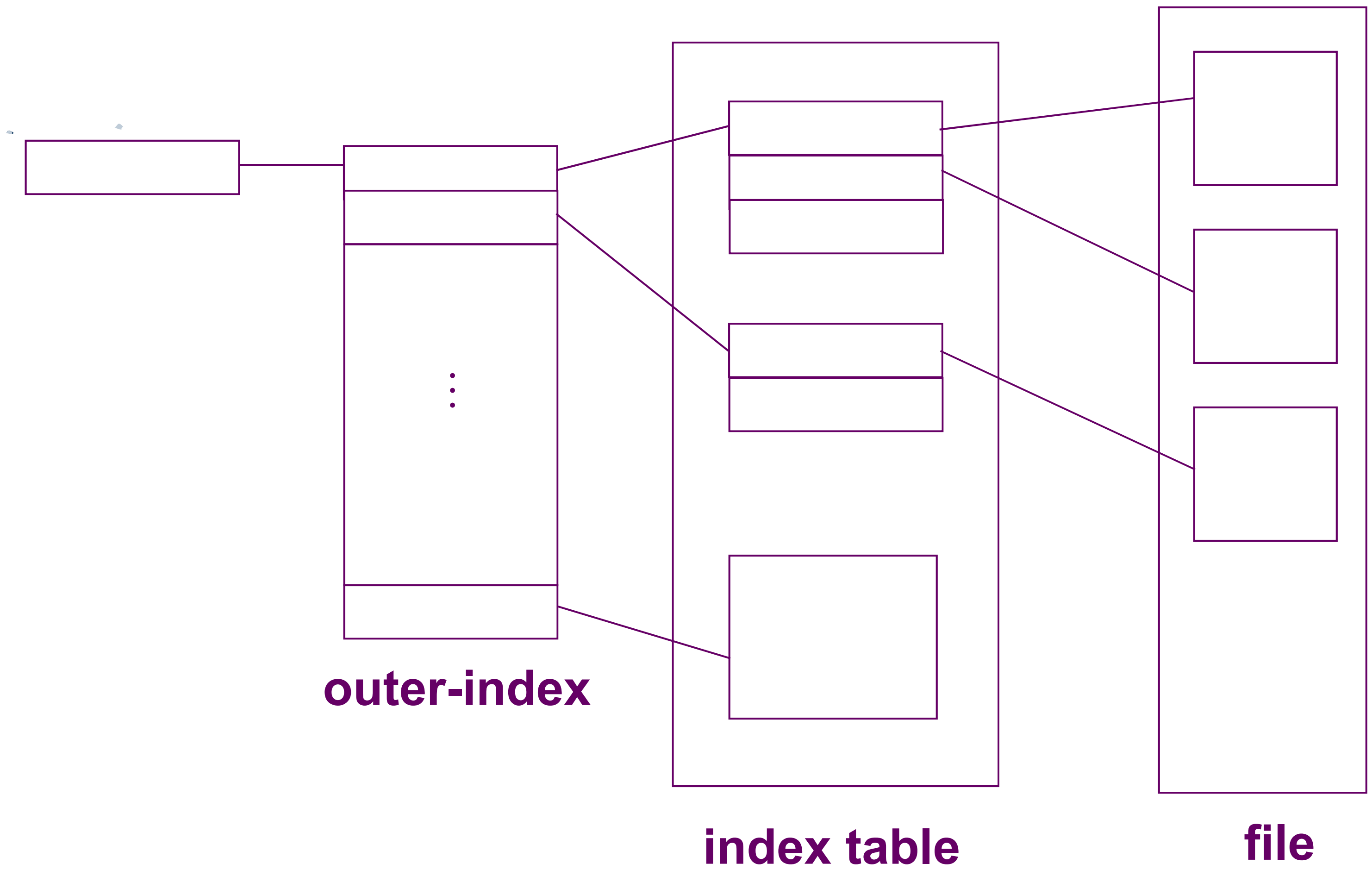


  - multiple index blocks, but make them a tree just like the indexed access method

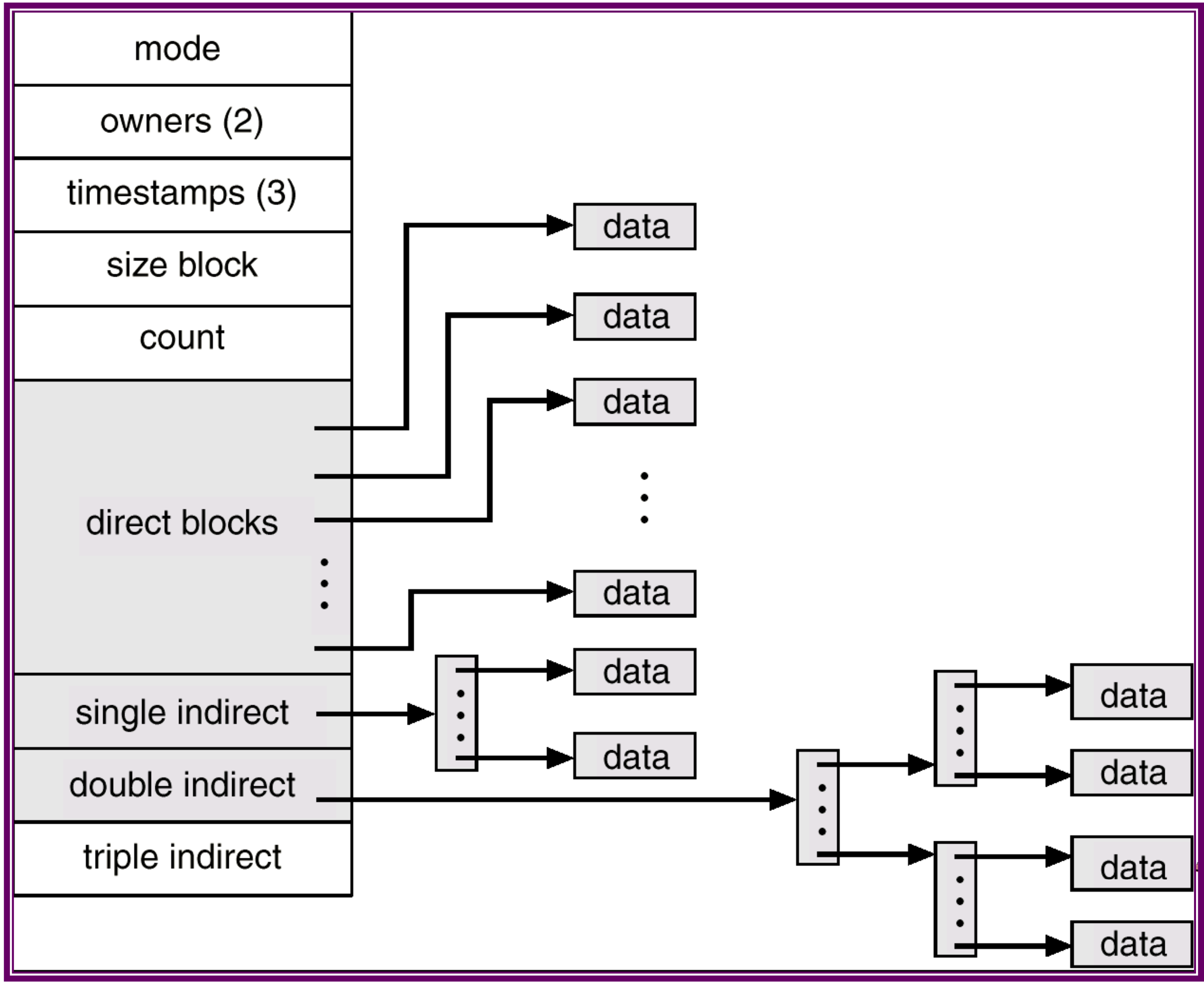


  - A combination of both

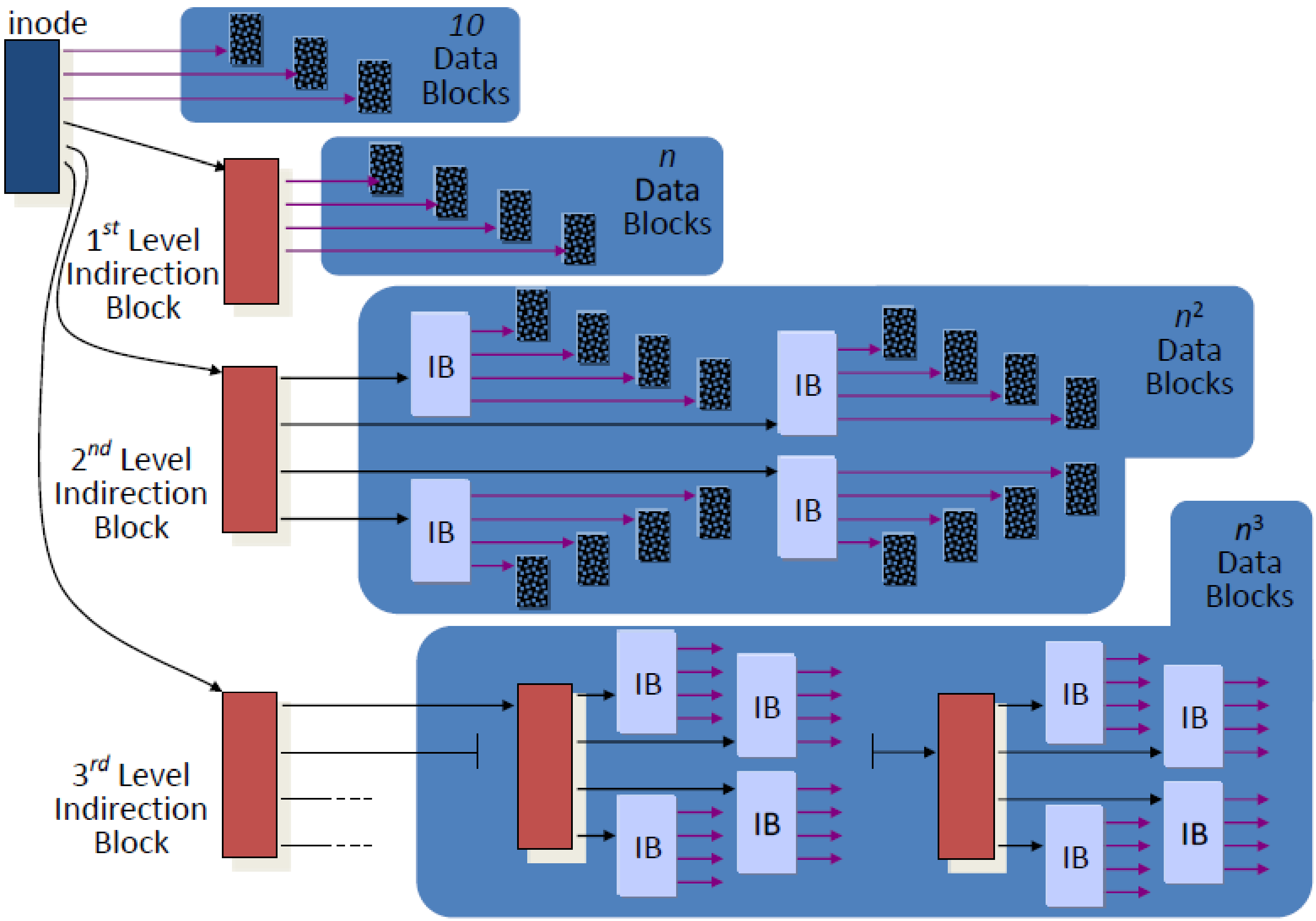
Indexed Allocation (cont.)



Combined Scheme: UNIX inode (4K Bytes per Block)



Multi-level Indexed Allocation in UNIX



Chapter 11: File System Implementation

- File System Structure
- Free-Space Management
- File System Implementation
- Directory Implementation
- Allocation Methods
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS

Efficiency and Performance

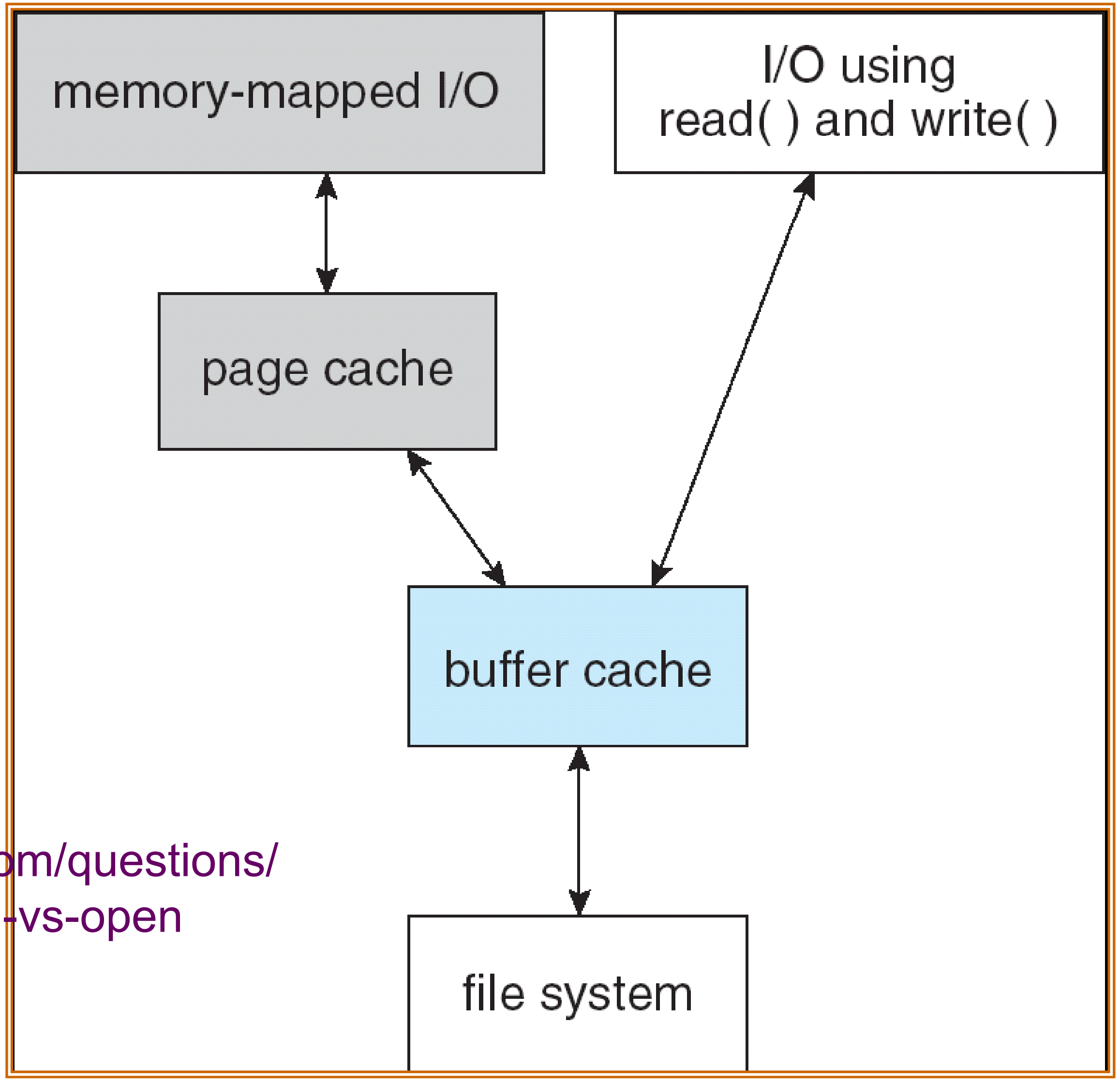
- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure

I/O Without a Unified Buffer Cache

open和fopen接口的区别是什么？



http://stackoverflow.com/questions/1658476/c-fopen-vs-open

Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O

I/O Using a Unified Buffer Cache