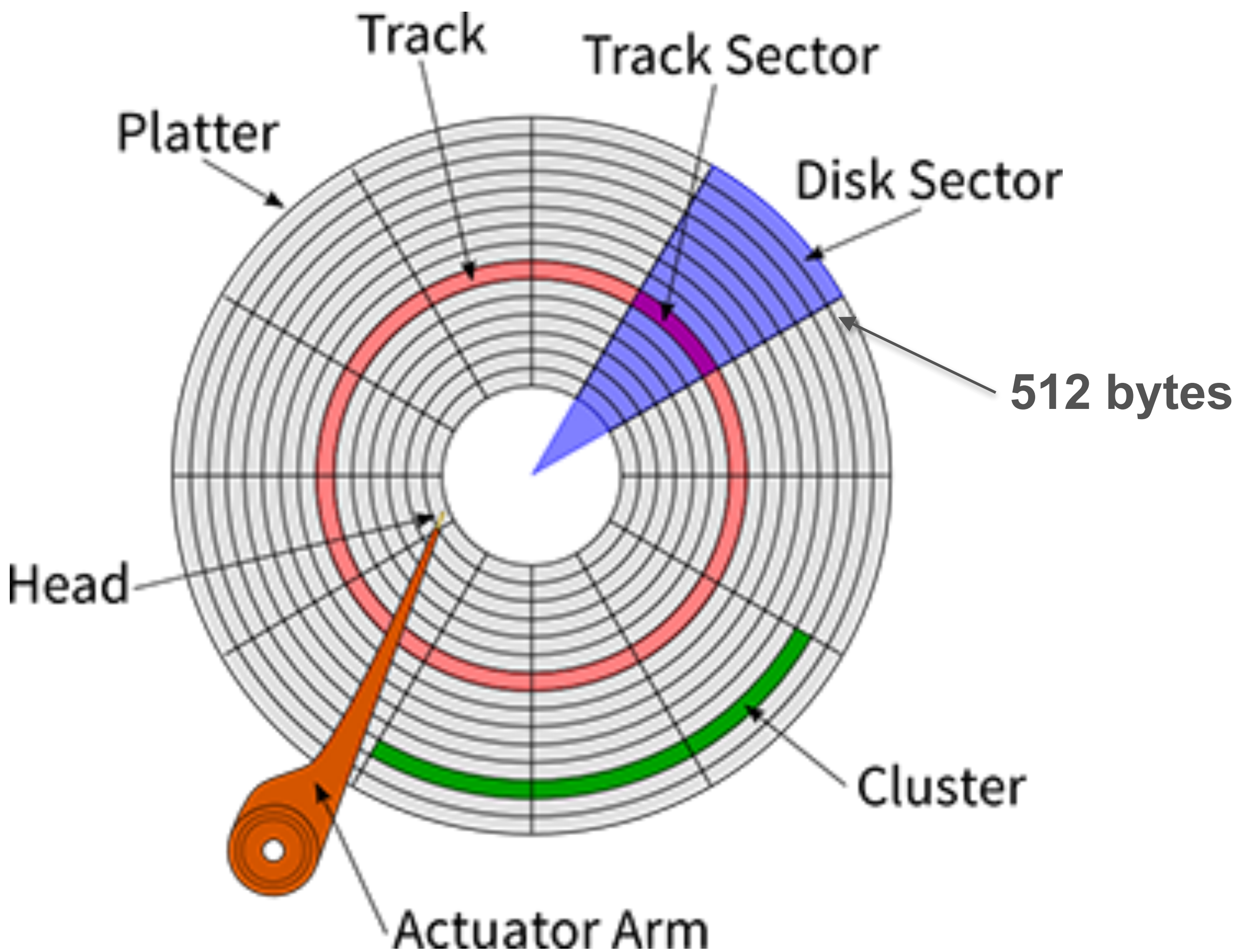
File-System Implementations

Chapter 4: Section 4.3

Disks


Track
Track Sector
Platter
Disk Sector
512 bytes
Head
Cluster
Actuator Arm

File-System Layout

- Master Boot Record (MBR) - Sector 0 of the disk.
  - End of the MBR contains the partition table
  - On boot BIOS reads and executes the MBR.
  - MBR locates the active partition and reads the first block called the boot block

File-System Layout

Structure of a modern standard MBR

Address: Hex	Address: Dec	Description		Size (bytes)
$+000_{hex}$	+0	Bootstrap code area (part 1)		218
$+0DA_{hex}$	+218	0000_{hex}	*Disk timestamp*[3][b] (optional, MS-DOS 7.1–8.0 and Windows 95B/98/98SE/ME. Alternatively, can serve as OEM loader signature with NEWLDR)	2
$+0DC_{hex}$	+220	Original physical drive (80_{hex}–FF_{hex})		1
$+0DD_{hex}$	+221	Seconds (0–59)		1
$+0DE_{hex}$	+222	Minutes (0–59)		1
$+0DF_{hex}$	+223	Hours (0–23)		1
$+0E0_{hex}$	+224	Bootstrap code area (part 2, code entry at $+000_{hex}$)		216 (or 222)
$+1B8_{hex}$	+440	32-bit disk signature	*Disk signature* (optional, UEFI, Windows NT/2000/Vista/7 and other OSes)	4
$+1BC_{hex}$	+444	0000_{hex} ($5A5A_{hex}$ if copy-protected)		2
$+1BE_{hex}$	+446	Partition entry №1	*Partition table* (for primary partitions)	16
$+1CE_{hex}$	+462	Partition entry №2		16
$+1DE_{hex}$	+478	Partition entry №3		16
$+1EE_{hex}$	+494	Partition entry №4		16
$+1FE_{hex}$	+510	55_{hex}	*Boot signature*[a]	2
$+1FF_{hex}$	+511	AA_{hex}		
			Total size: 218 + 6 + 216 + 6 + 4x16 + 2	**512**

File-System Layout

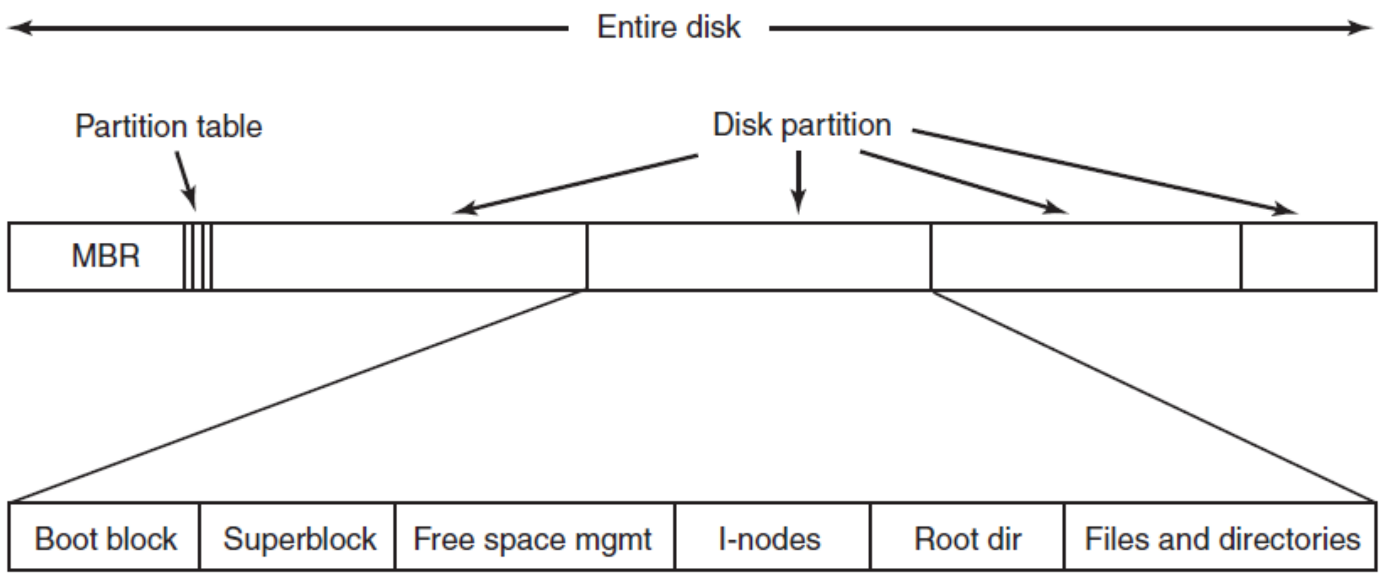
- The boot block loads the OS contained in that partition.
  - Every partition starts with a boot block even if it does not contain a bootable OS.

File-System Layout

Layout of one 16-byte partition entry[13] (all multi-byte fields are little-endian)

Offset (bytes)		Field length	Description
$+0_{hex}$		1 byte	Status or physical drive (bit 7 set is for active or bootable, old MBRs only accept 80_{hex}, 00_{hex} means inactive, and 01_{hex}–$7F_{hex}$ stand for invalid)[c]
$+1_{hex}$		3 bytes	CHS address of first absolute sector in partition.[d] The format is described by three bytes, see the next three rows.
	$+1_{hex}$	1 byte	$h_{7–0}$: x x x x x x x x — head[e]
	$+2_{hex}$	1 byte	$c_{9–8}$: x x; $s_{5–0}$: x x x x x x — sector in bits 5–0; bits 7–6 are high bits of cylinder[e]
	$+3_{hex}$	1 byte	$c_{7–0}$: x x x x x x x x — bits 7–0 of cylinder[e]
$+4_{hex}$		1 byte	Partition type[15]
$+5_{hex}$		3 bytes	CHS address of last absolute sector in partition.[d] The format is described by 3 bytes, see the next 3 rows.
	$+5_{hex}$	1 byte	$h_{7–0}$: x x x x x x x x — head[e]
	$+6_{hex}$	1 byte	$c_{9–8}$: x x; $s_{5–0}$: x x x x x x — sector in bits 5–0; bits 7–6 are high bits of cylinder[e]
	$+7_{hex}$	1 byte	$c_{7–0}$: x x x x x x x x — bits 7–0 of cylinder
$+8_{hex}$		4 bytes	LBA of first absolute sector in the partition[f]
$+C_{hex}$		4 bytes	Number of sectors in partition[f]

File-System Layout



- Superblock contains key parameters about the file system
  - Magic number, number of blocks, etc.

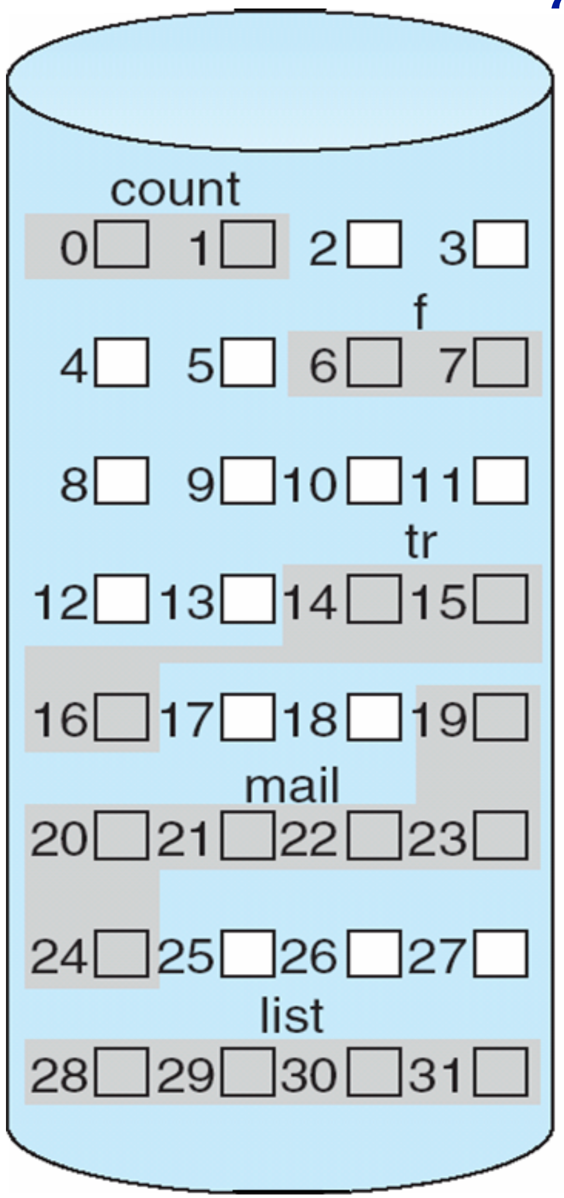
Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- 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

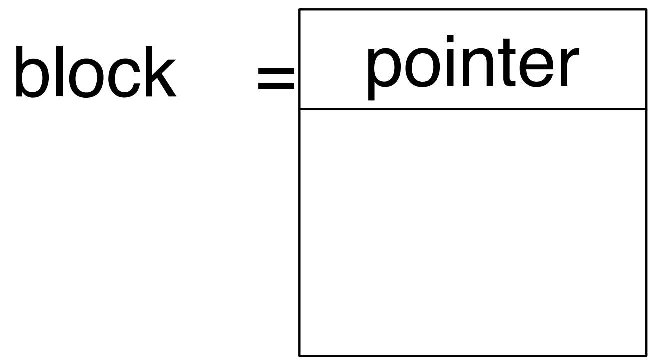


directory

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

Linked Allocation

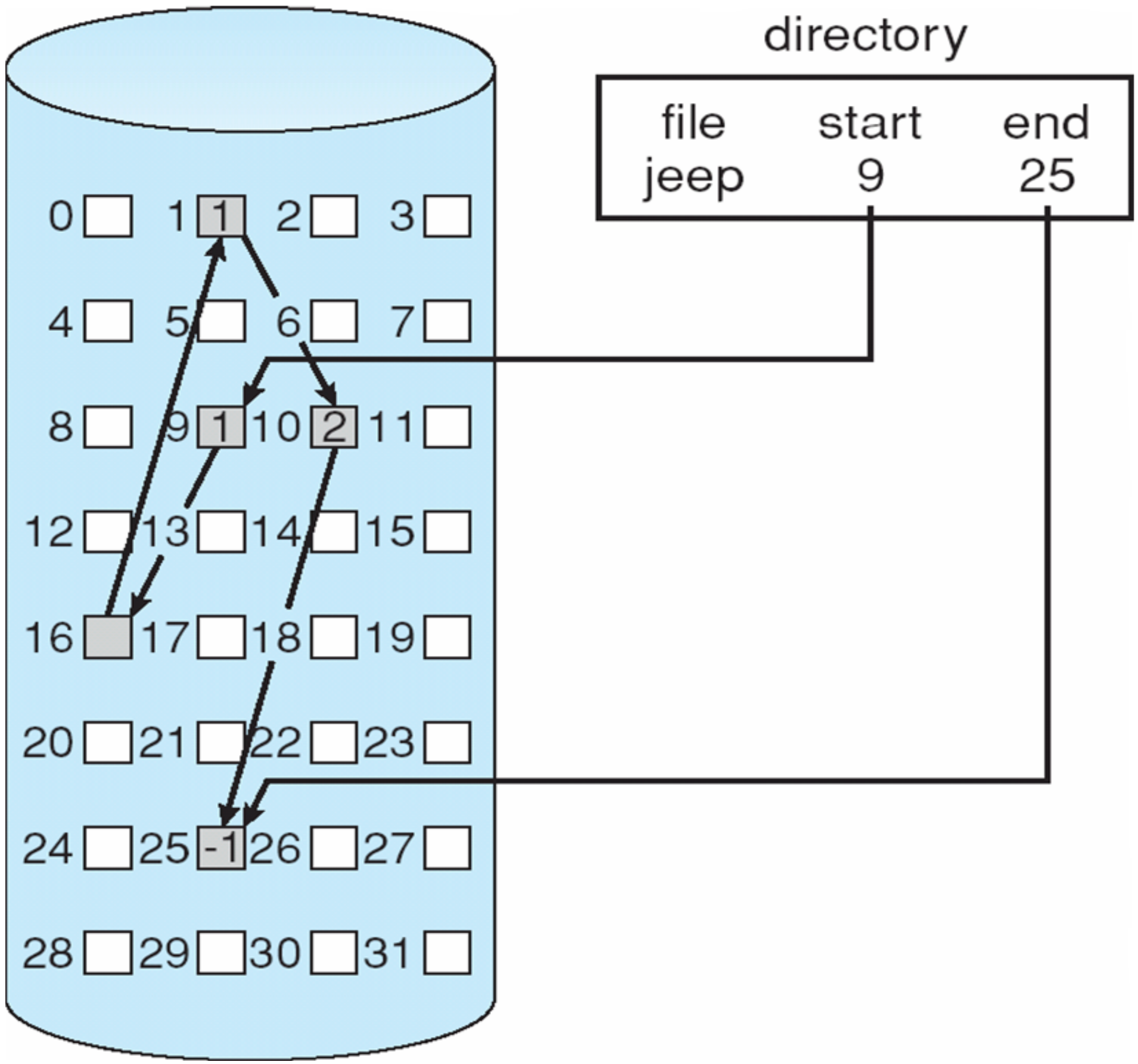
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



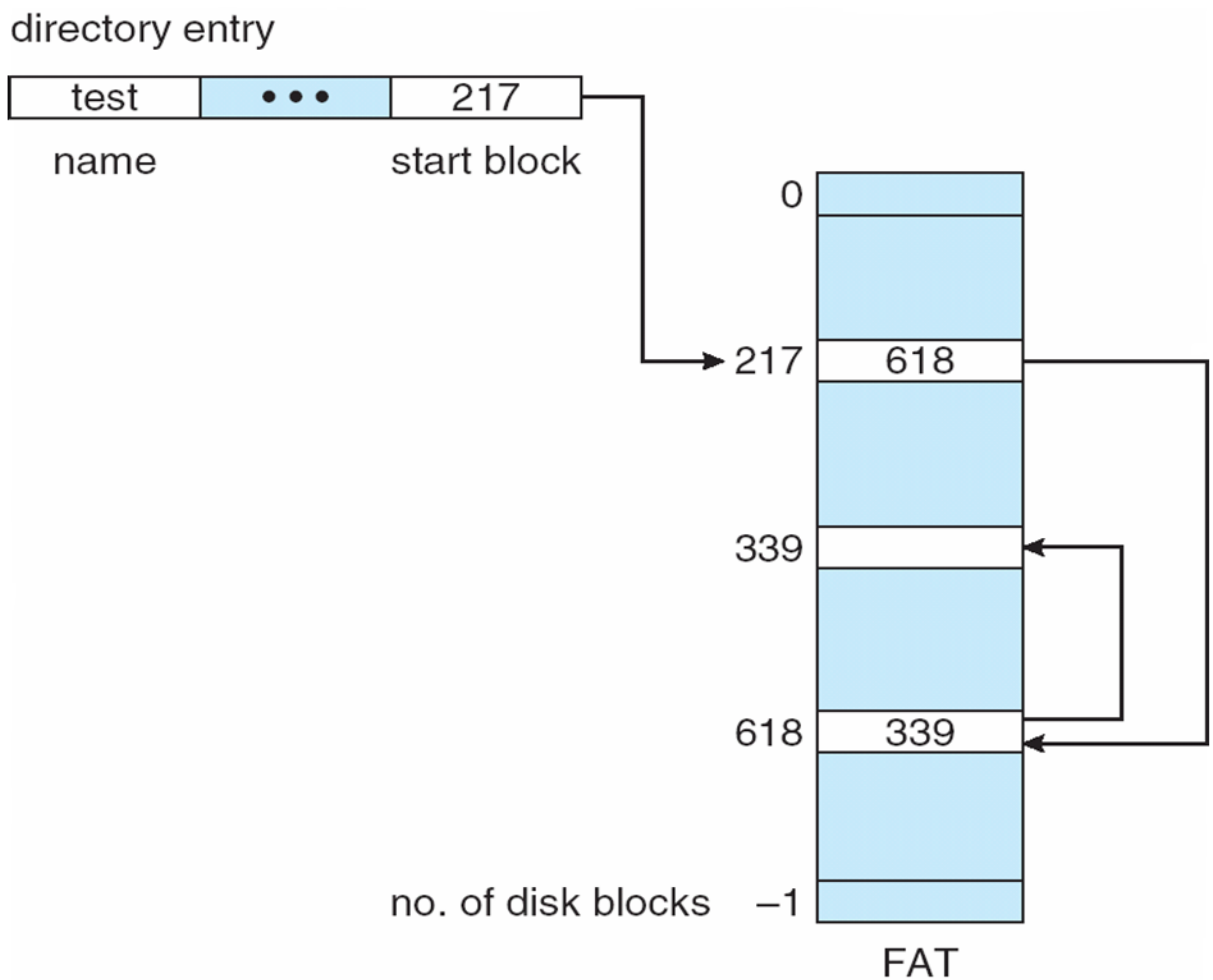
Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system – no waste of space
- No random access

Linked Allocation

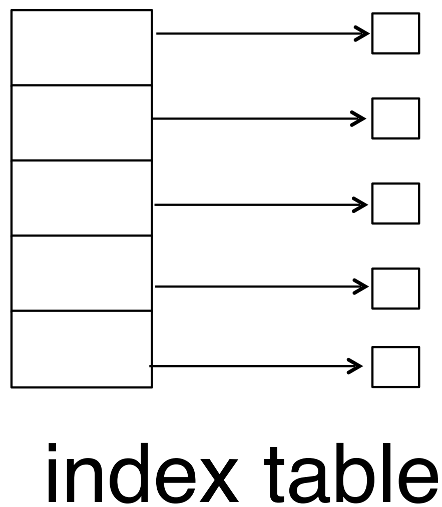


File-Allocation Table

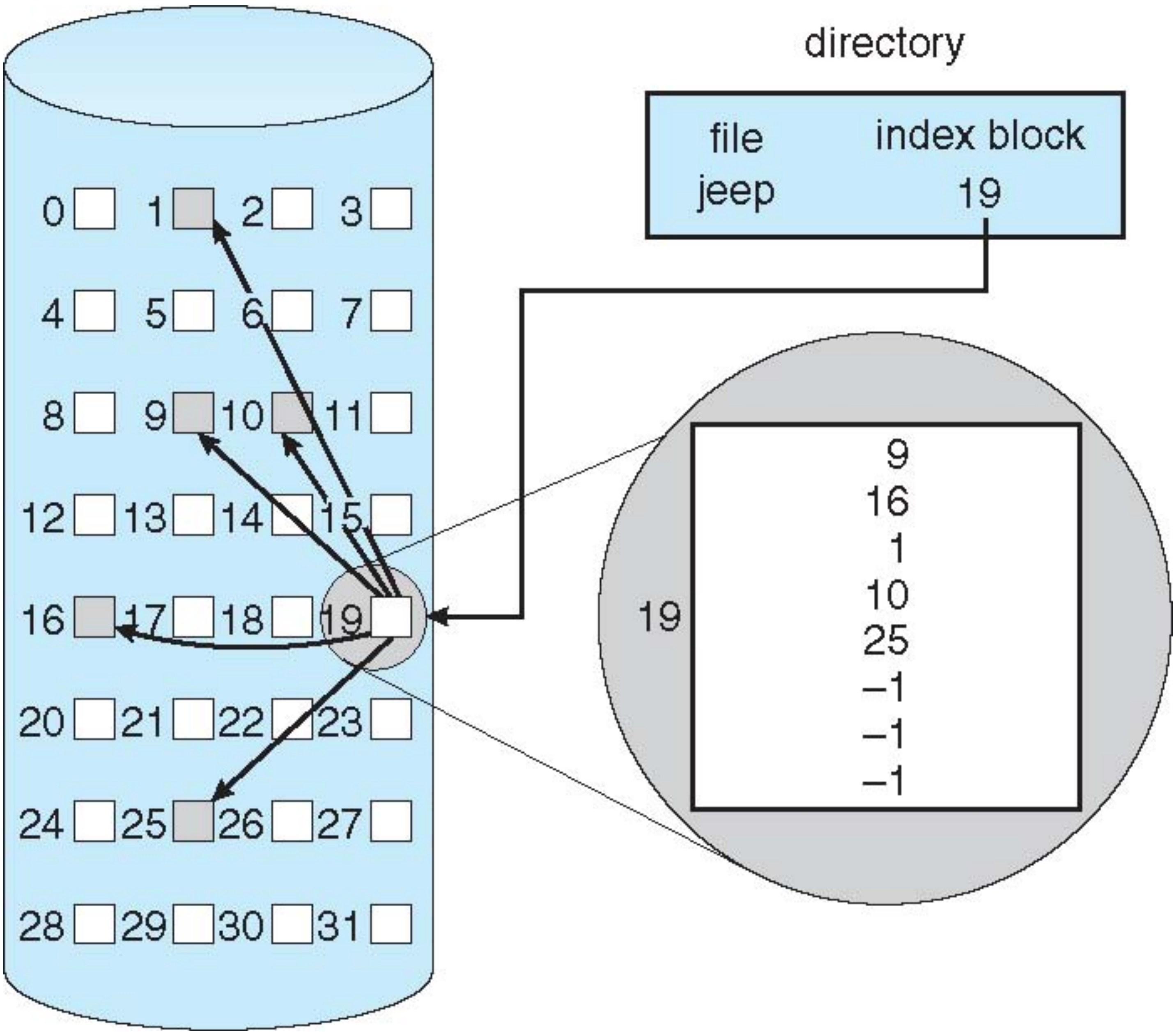


Indexed Allocation

- Brings all pointers together into the **index block**
- Logical view



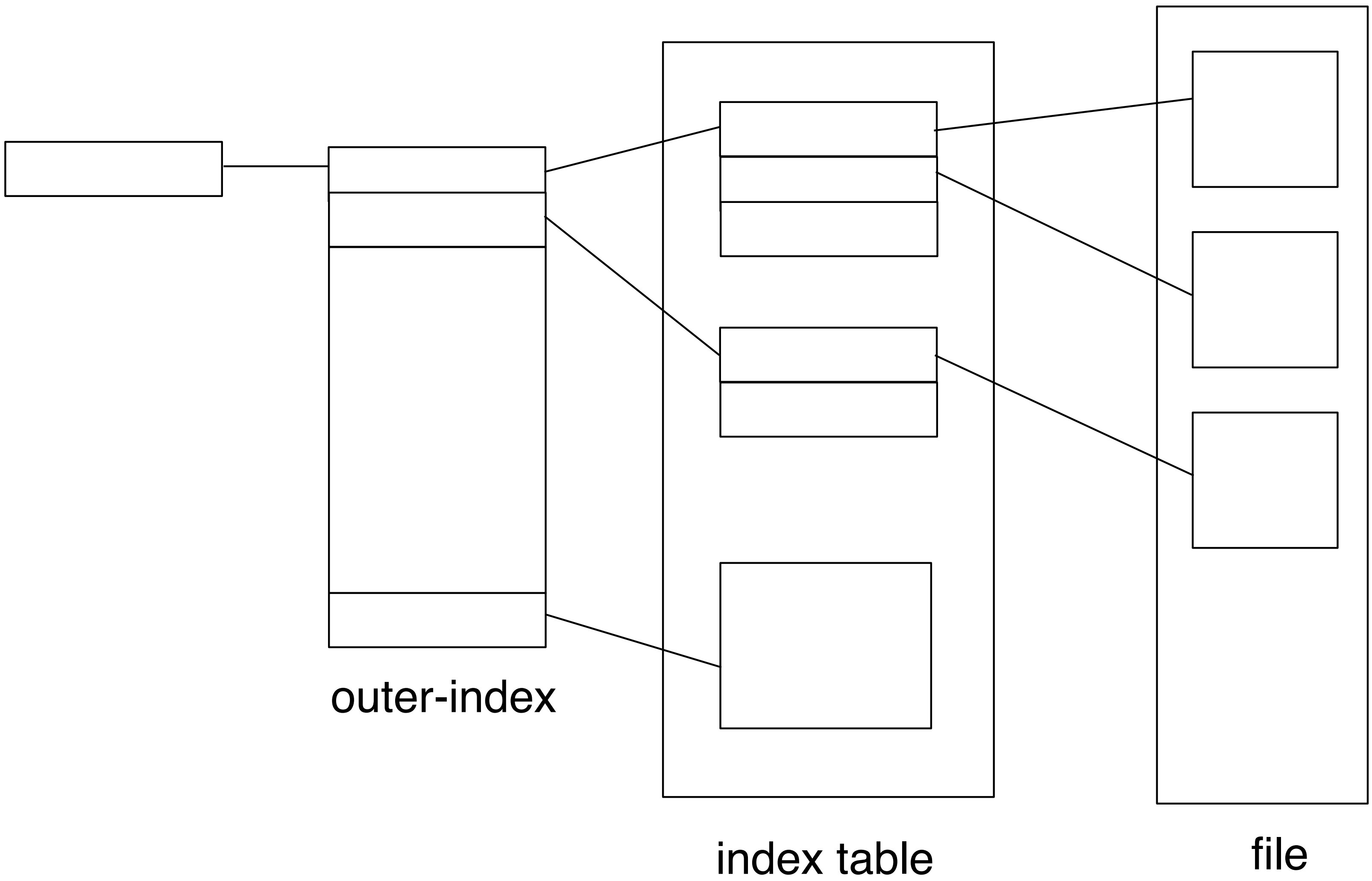
Example of Indexed Allocation



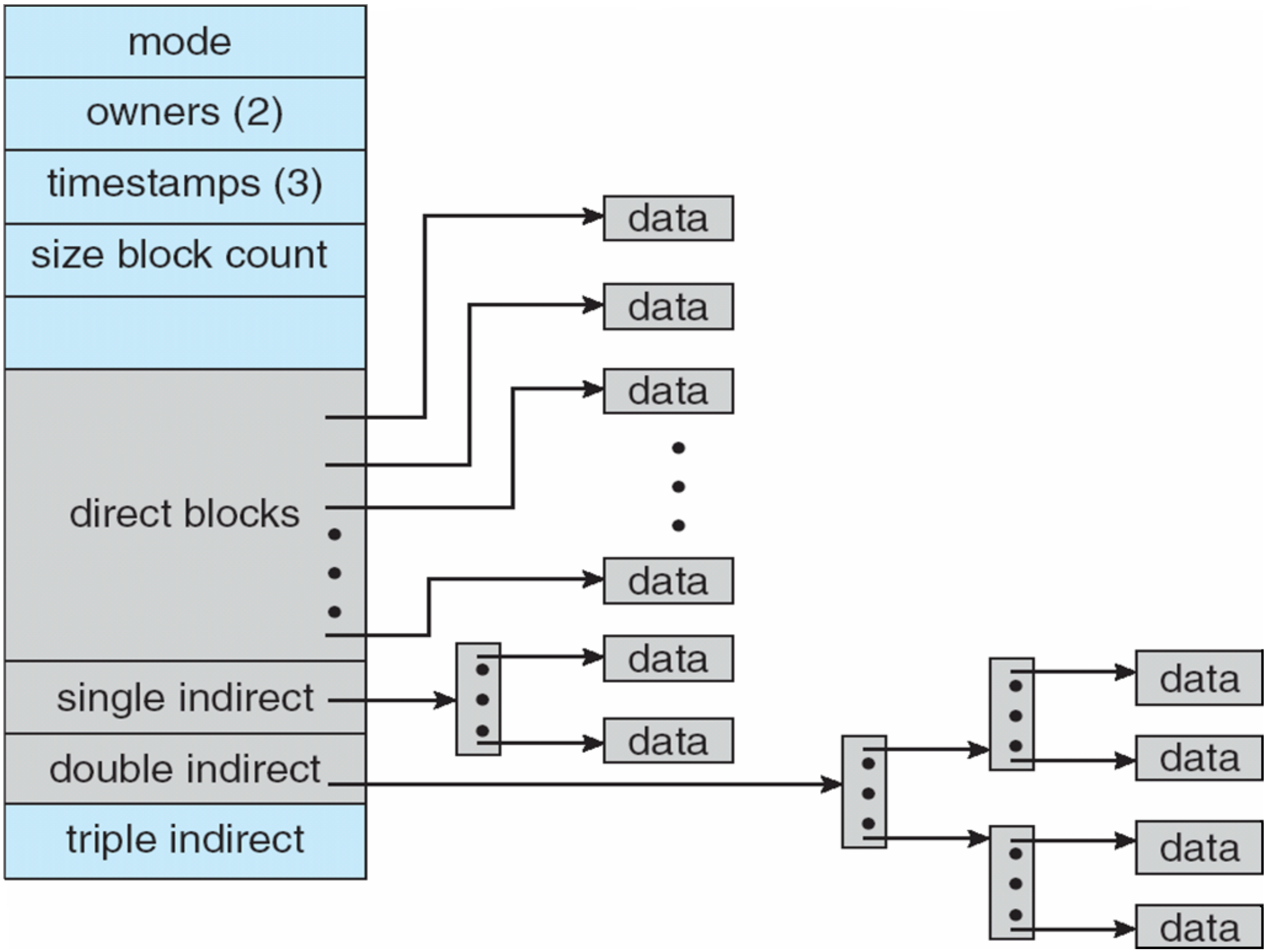
Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block

Indexed Allocation – Mapping (Cont.)



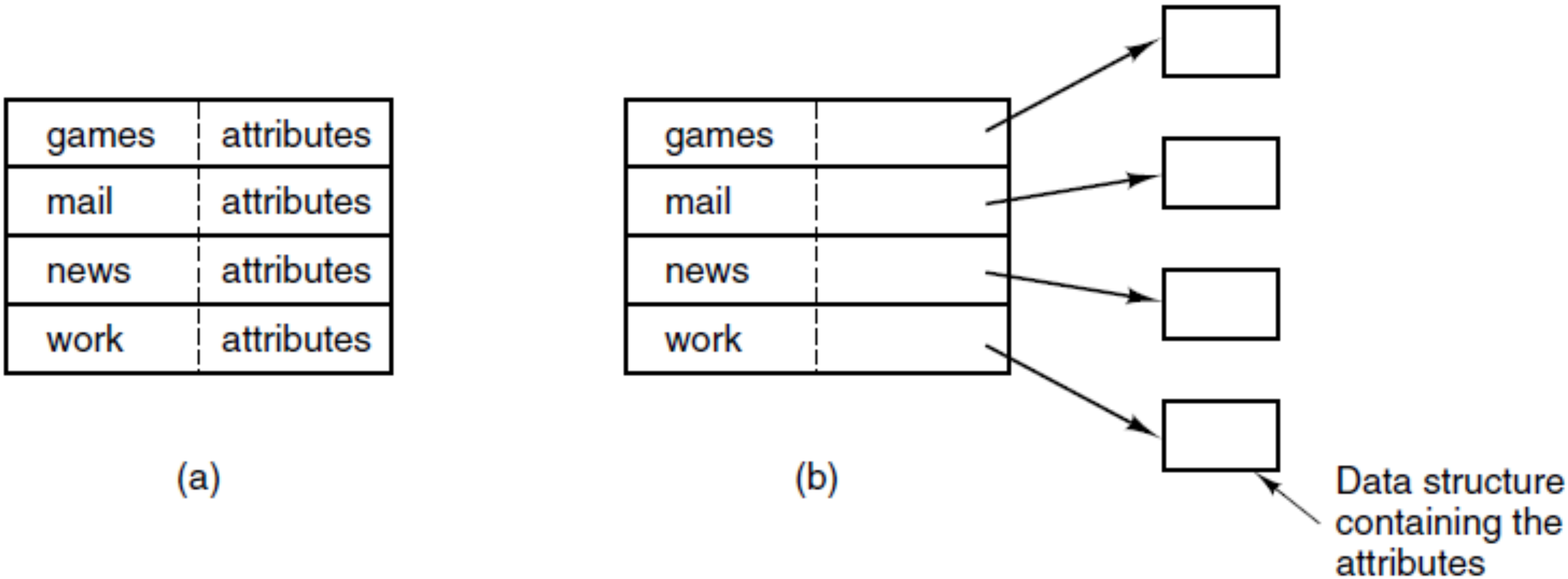
Combined Scheme: UNIX UFS
(4K bytes per block)



Directories

- File systems need to track not only locations but other file metadata as well:
  - owner
  - creation time
- Obvious location is the directory
  - inode is also alternative

Directories



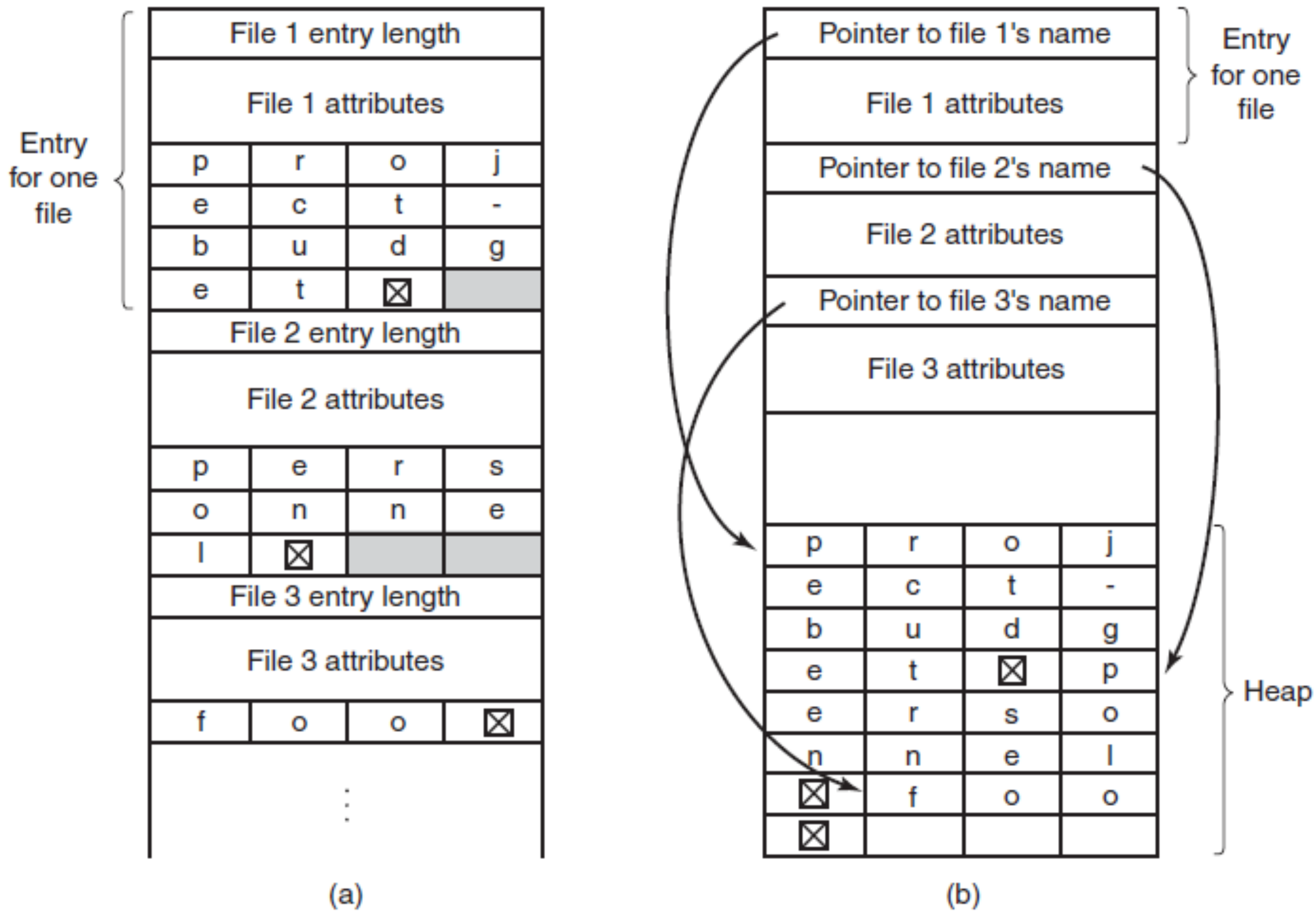
(a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry.

(b) A directory in which each entry just refers to an i-node.

File Naming

- Simplest approach: allocate and limit to 255
  - Wastes space
- Instead, give up on directory entries being the same size.

File Naming



(a) In-line. (b) In a heap.

File Naming

» In-line disadvantage: when a file is removed a variable sized gap is introduced into the directory.

» Can also span a page entry and a page fault may occur when the filename is read.

File Naming

» In-line disadvantage: filename heap must be managed.

» Can also span a page entry and a page fault may occur when the filename is read.

Directory Structure

- So far we've looked at linear structures
- Hash it instead
  - Table size n, has the file from 0-n
  - O(1)
  - Only reasonable option when directories can hold thousands of files.

Log Structured File Systems

- » CPU getting faster
- » Disks becoming bigger and cheaper
- » RAM growing exponentially
- » Disk seek time is not improving by leaps and bounds
  - » Except SSDs
- » Bottleneck

Log Structured File Systems

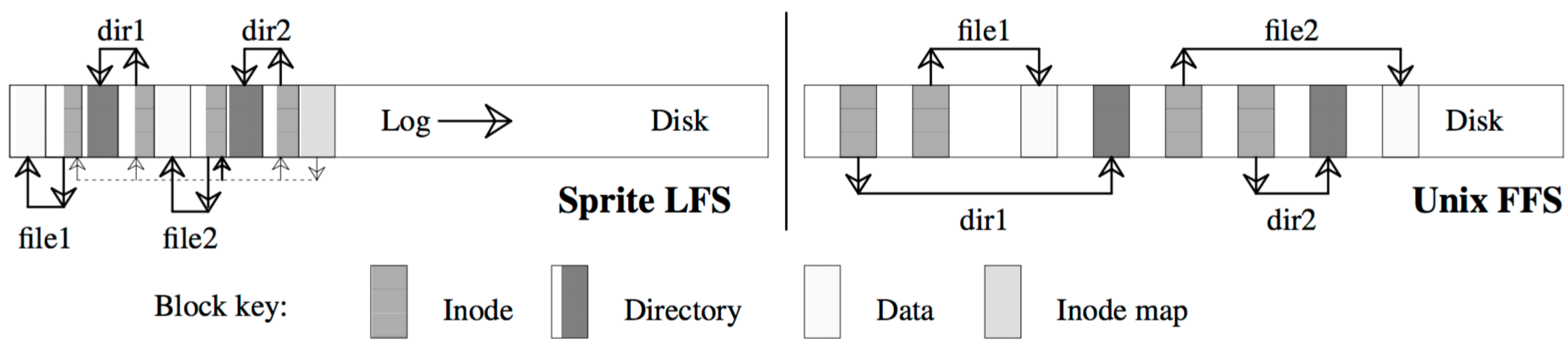
» Log Structured File System (LFS)

» Conventional file systems tend to lay out files with great care for spatial locality and make in-place changes to their data structures.

» I/O becoming write-heavy because reads almost always satisfied from memory cache.

Log Structured File Systems

» A log-structured file system thus treats its storage as a circular log and writes sequentially to the head of the log



Log Structured File Systems

- Important side effects:
- Write throughput on optical and magnetic disks is improved because they can be batched into large sequential runs and costly seeks are kept to a minimum.
- Writes create multiple, chronologically-advancing versions of both file data and meta-data. Some implementations make these old file versions nameable and accessible, a feature sometimes called time-travel or snapshotting.
- Recovery from crashes is simpler. Upon its next mount, the file system does not need to walk all its data structures to fix any inconsistencies, but can reconstruct its state from the last consistent point in the log.

Log Structured File Systems

- Disadvantage
  - Must reclaim free space from the tail of the log to prevent the file system from becoming full when the head of the log wraps around to meet it.
  - To reduce the overhead incurred by this garbage collection, most implementations avoid purely circular logs and divide up their storage into segments.

Journaling File System

» A journaling file system is a file system that keeps track of changes not yet committed to the file system's main part by recording the intentions of such changes in a data structure known as a "journal", which is usually a circular log

Journaling File System

» For example, deleting a file on a Unix file system involves three steps:

1. Removing its directory entry.
2. Releasing the inode to the pool of free inodes.
3. Returning all used disk blocks to the pool of free disk blocks

Journaling File System

» Detecting and recovering from such inconsistencies normally requires a complete walk of its data structures, for example by a tool such as fsck.

» This must typically be done before the file system is next mounted for read-write access.

Journaling File System

- Journal allocated
  - Records the changes it will make ahead of time.
  - After a crash
    - Read the journal from the file system
    - Replay changes from this journal until the file system is consistent again

Journaling File System

- Physical Journals
  - Logs an advance copy of every block that will later be written to the main file system.
    - If crash during write then replay from journal
  - Significant performance penalty because every changed block must be committed twice to storage

Journaling File System

- Logical Journals
  - A logical journal stores only changes to file metadata in the journal, and trades fault tolerance for substantially better write performance.
  - A file system with a logical journal still recovers quickly after a crash, but may allow unjournaled file data and journaled metadata to fall out of sync with each other, causing data corruption.

Journaling File System

» For example, appending to a file may involve three separate writes to:

1. The file's inode, to note in the file's metadata that its size has increased.
2. The free space map, to mark out an allocation of space for the to-be-appended data.
3. The newly allocated space, to actually write the appended data.

» In a metadata-only journal, step 3 would not be logged. If step 3 was not done, but steps 1 and 2 are replayed during recovery, the file will be appended with garbage.