Chapter 11: File System Implementation

Chapter 11: File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- Example: FAT32, ISO-9660
- Introduction: NTFS, ZFS

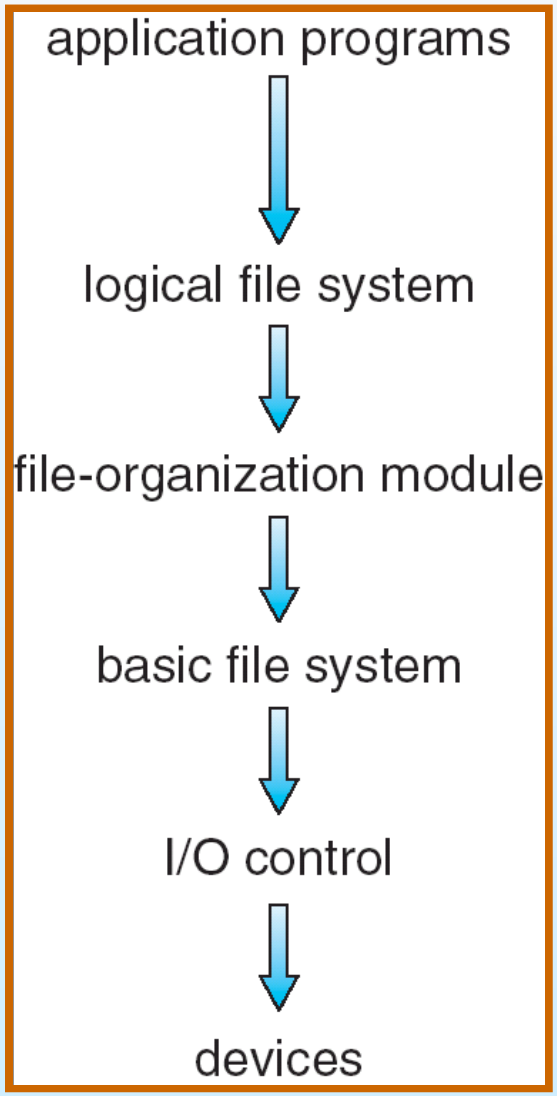
Objectives

- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems
- To discuss block allocation and free-block algorithms and trade-offs

File-System Structure

- File system resides on secondary storage (disks)
- File system organized into layers
- File control block
  - Storage structure consisting of information about a file
  - Permissions, dates(create, access, write), owner, group, ACL, size, data blocks

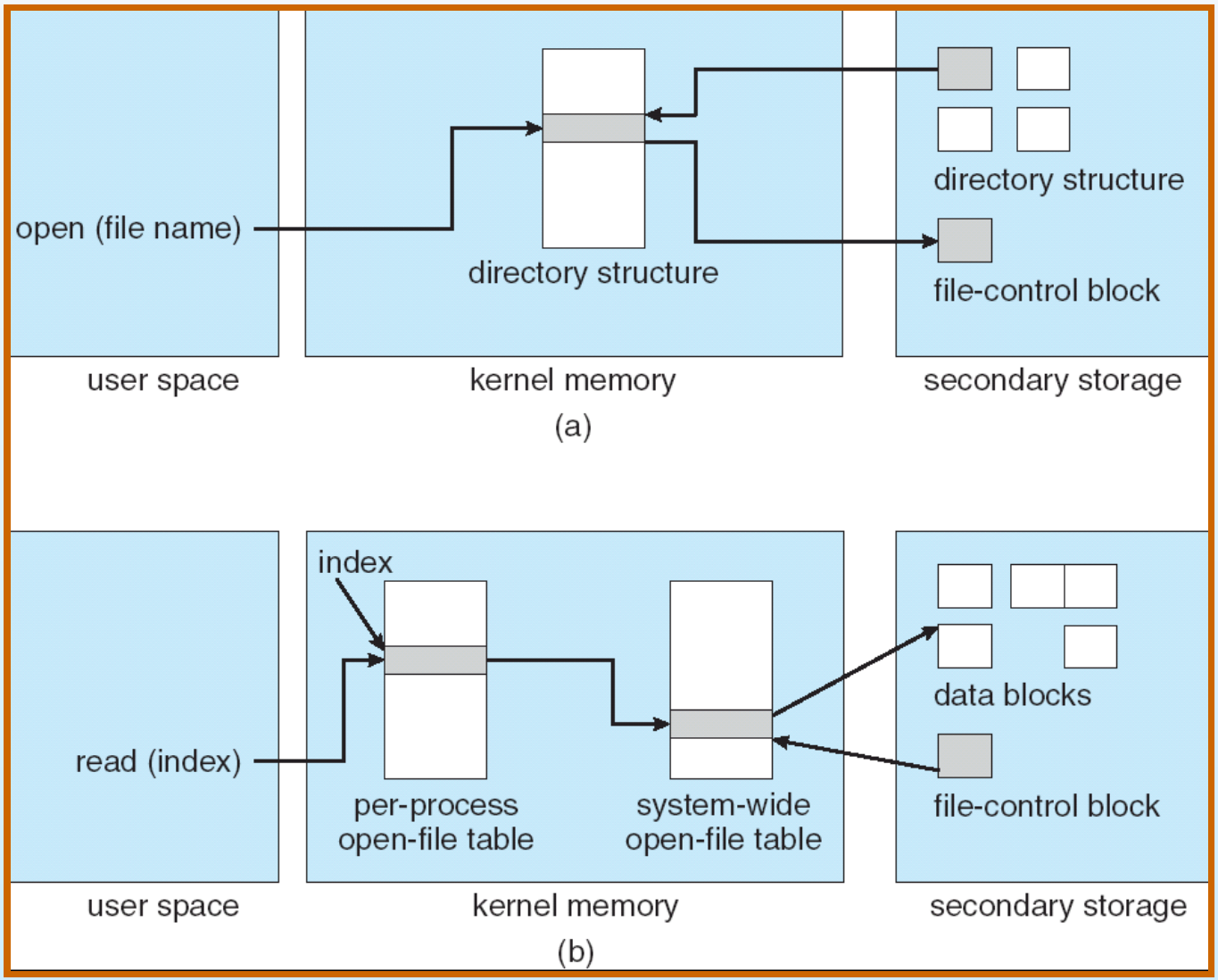
Layered File System



File-system Implementation

- On disk
  - Boot control block
  - Volume control block
  - Directory structure
  - FCB
- In memory
  - Mount table
  - Directory-structure cache
  - System-wide open-file table
  - Per-process open-file table

In-Memory File System Structures



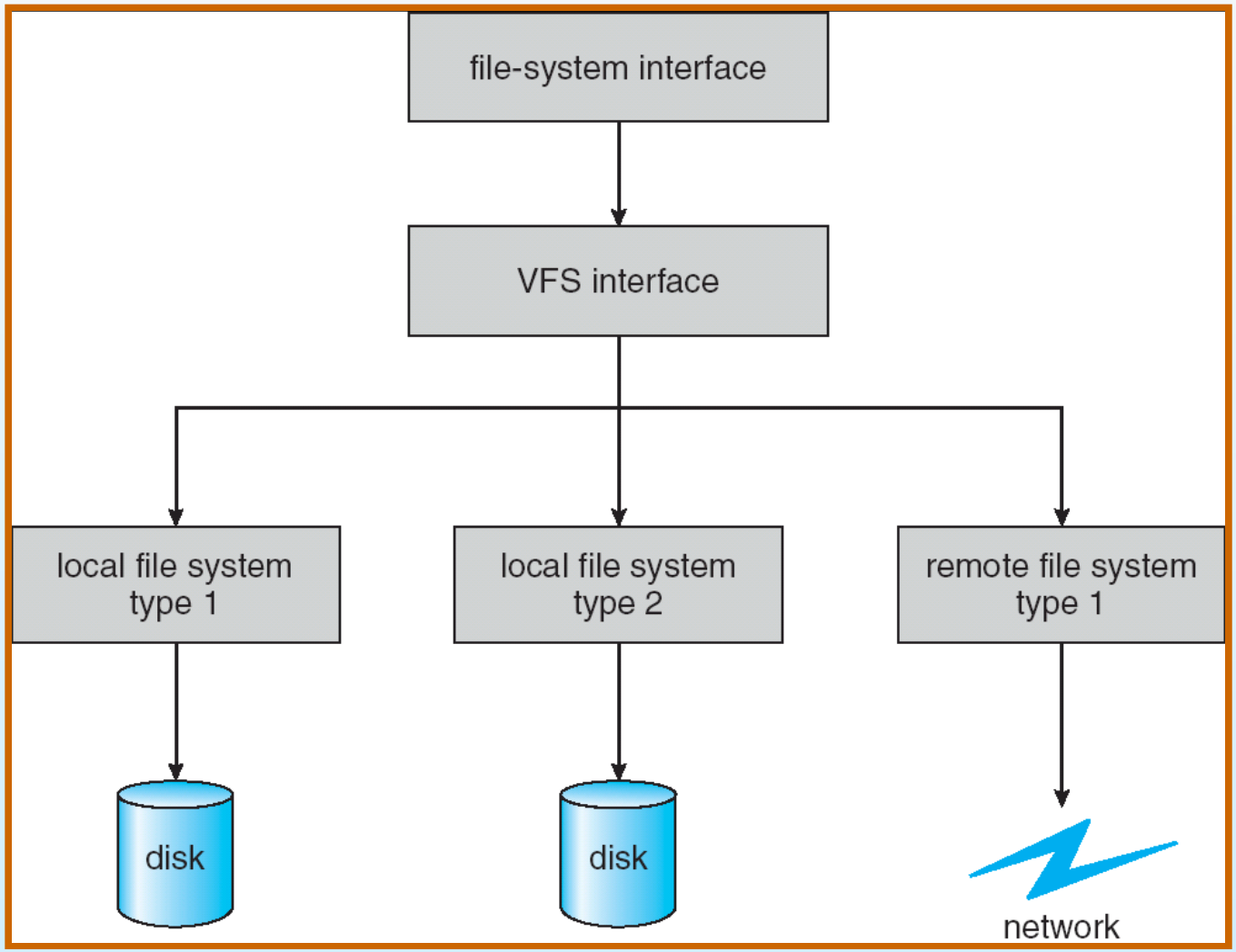
Partitions and Mounting

- A disk can be sliced into multiple partitions
  - Raw partition
  - Formatted partition
- A volume can include
  - Single partition
  - Multiple partitions
  - Multiple partitions on multiple disks
- Dual-boot
  - Different OS on different partition
- Root partition
  - Auto mount at boot time

Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Schematic View of Virtual File System



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

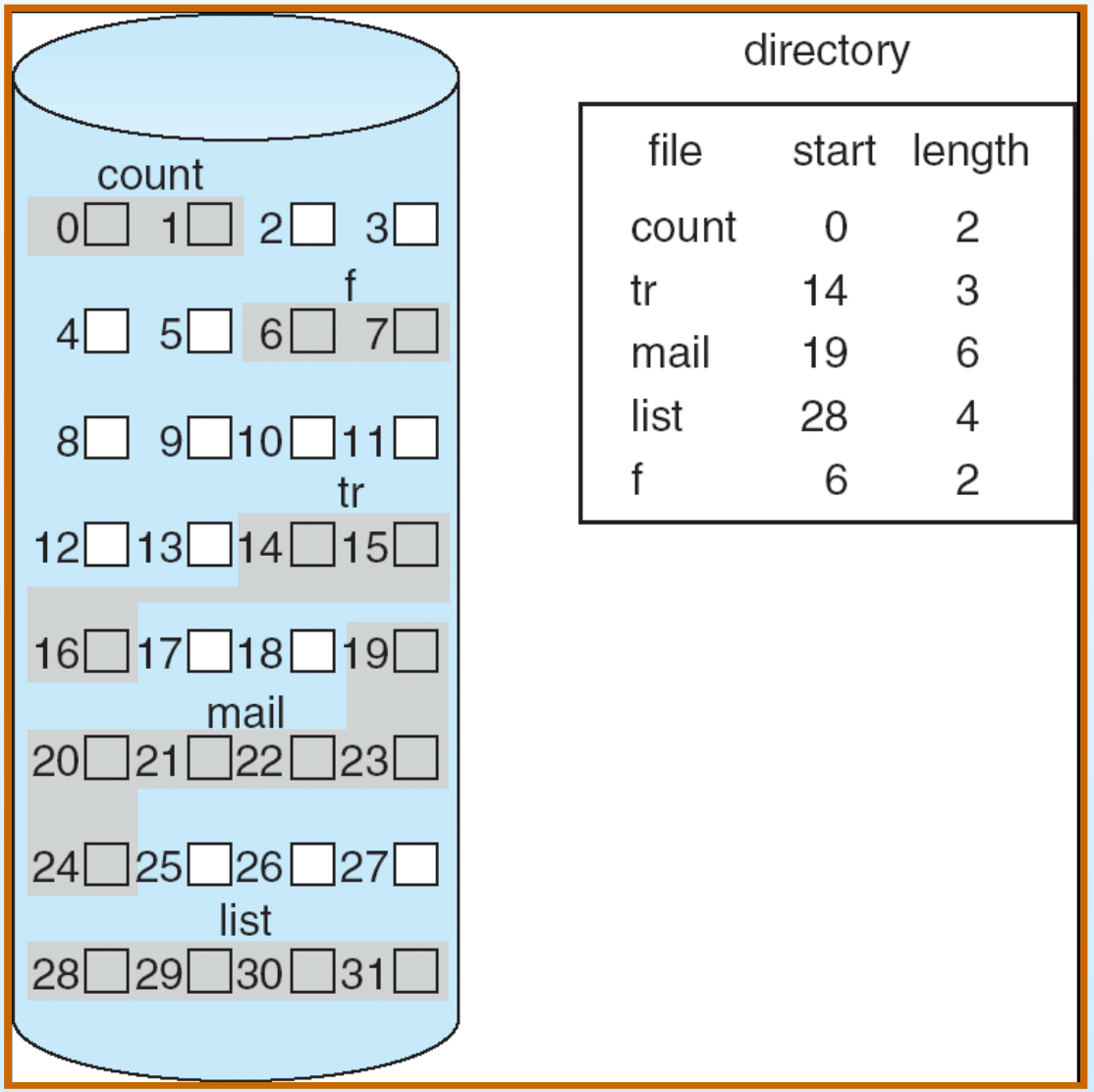
Allocation Methods

- How disk blocks are allocated for files:
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks
- 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

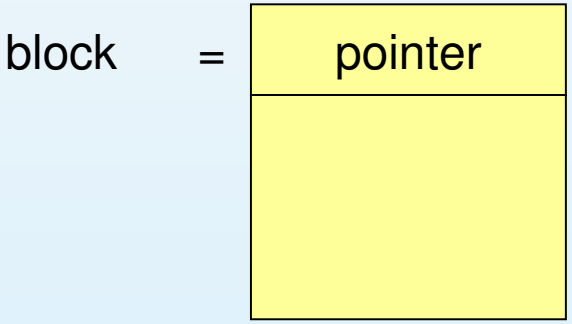


directory

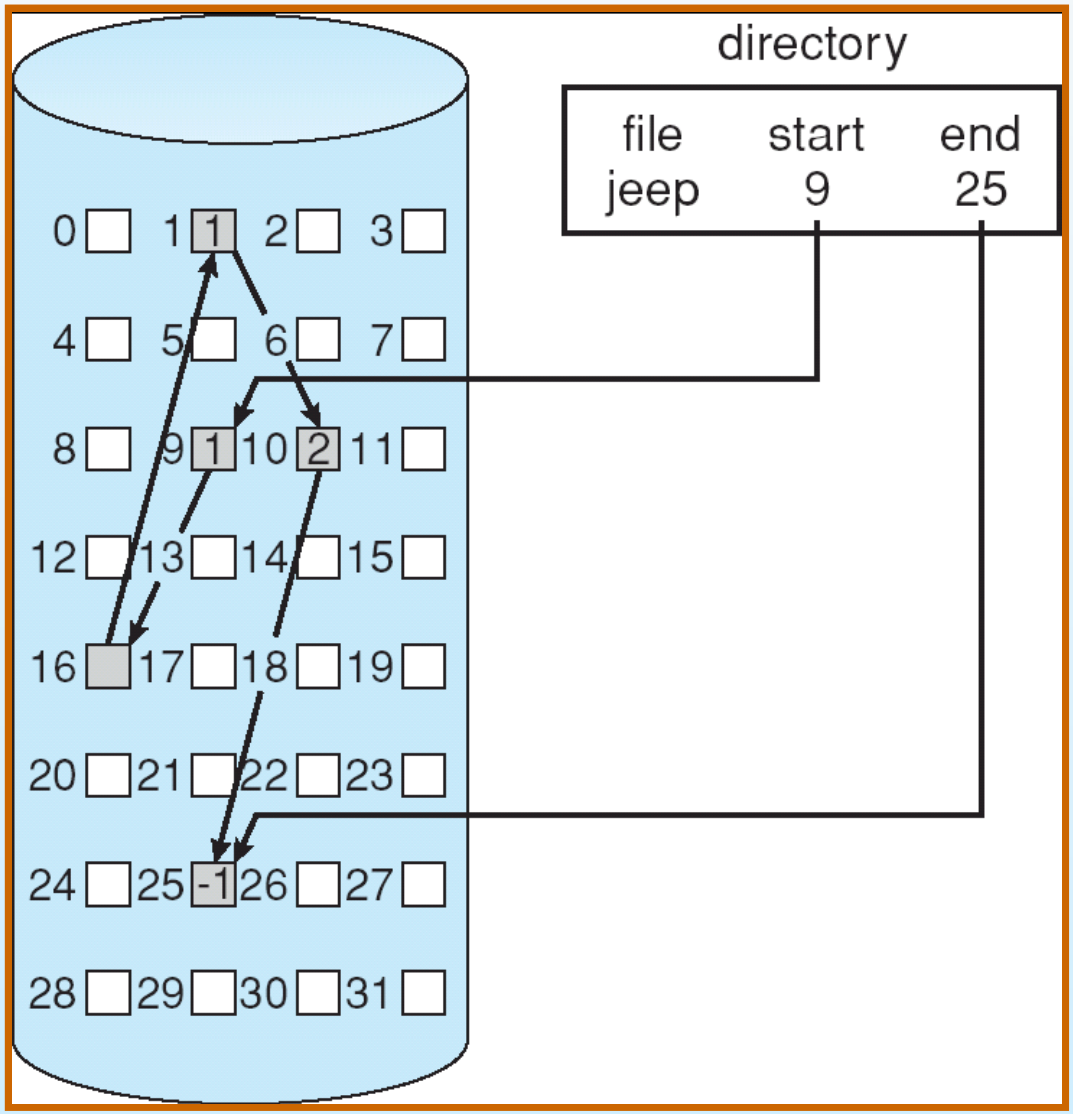
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked Allocation

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



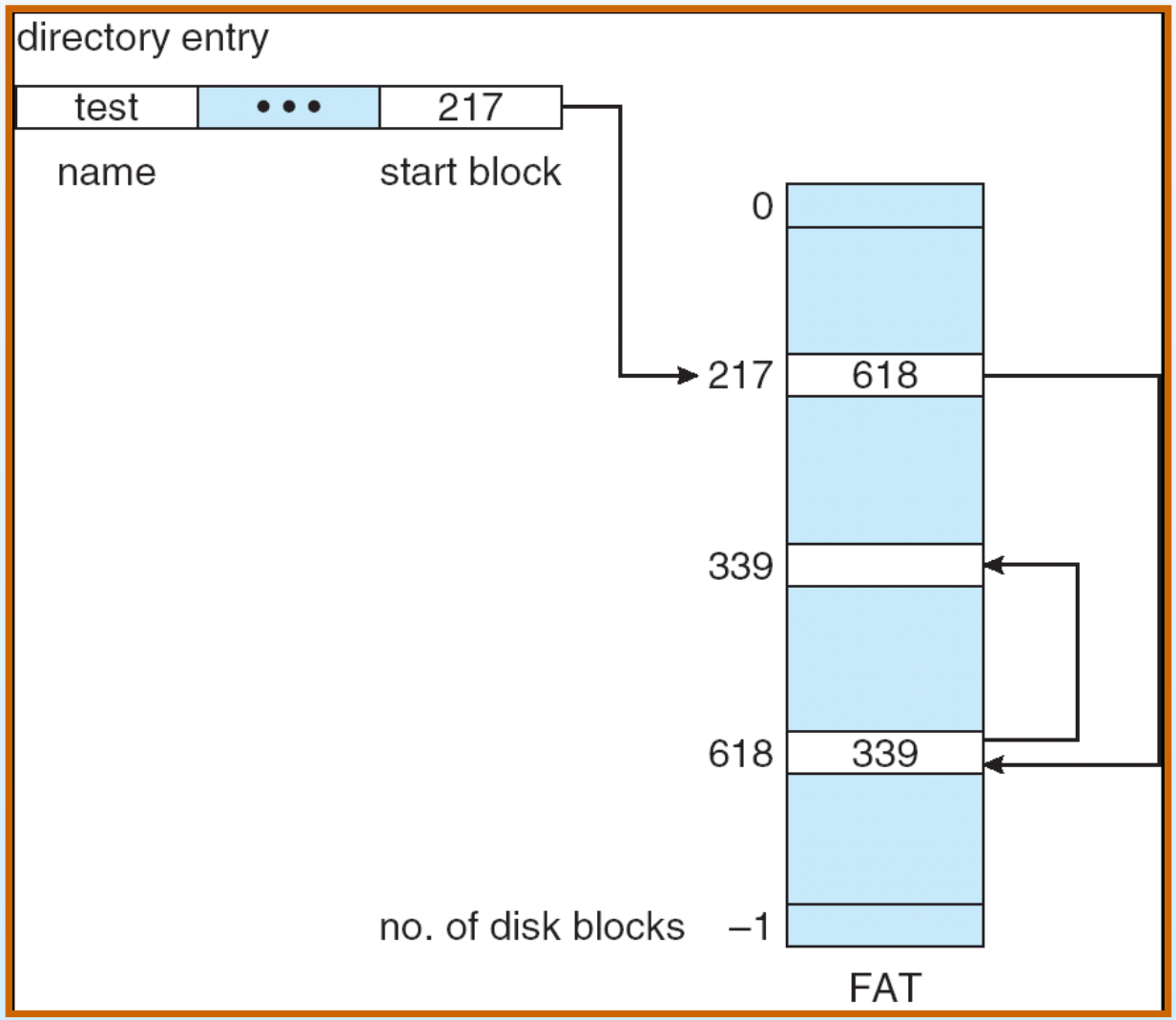
Linked Allocation



Linked Allocation (Cont.)

- Simple
  - Need only starting address
- Free-space management system
  - No waste of space
- No random access
- Unreliable
- Mapping

File-Allocation Table

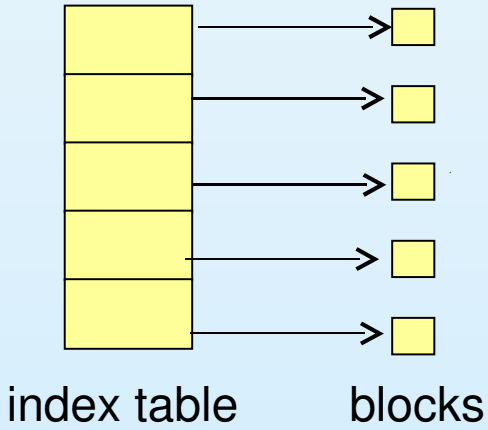


File-Allocation Table (Cont.)

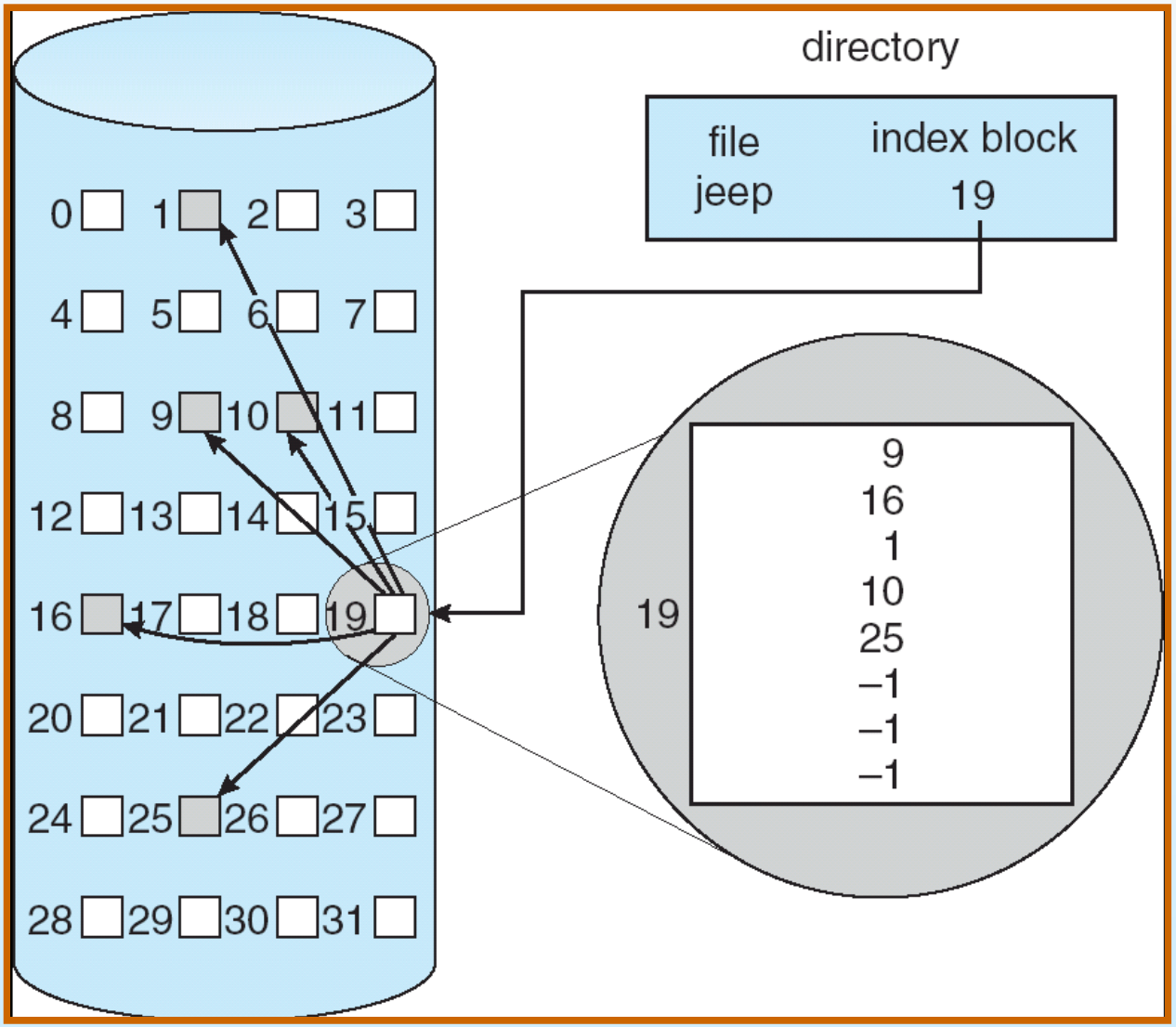
- Used by MS-DOS, OS/2 and Windows
  - Widely known as FAT12, FAT16 and FAT32
- Double FAT for reliability
  - Overhead
- Cache FAT for performance
- Random access

Indexed Allocation

- Brings all pointers together into the *index block.*
  - Widely known as inode
  - Popular in Unix/Linux
- Logical view.



Example of Indexed Allocation



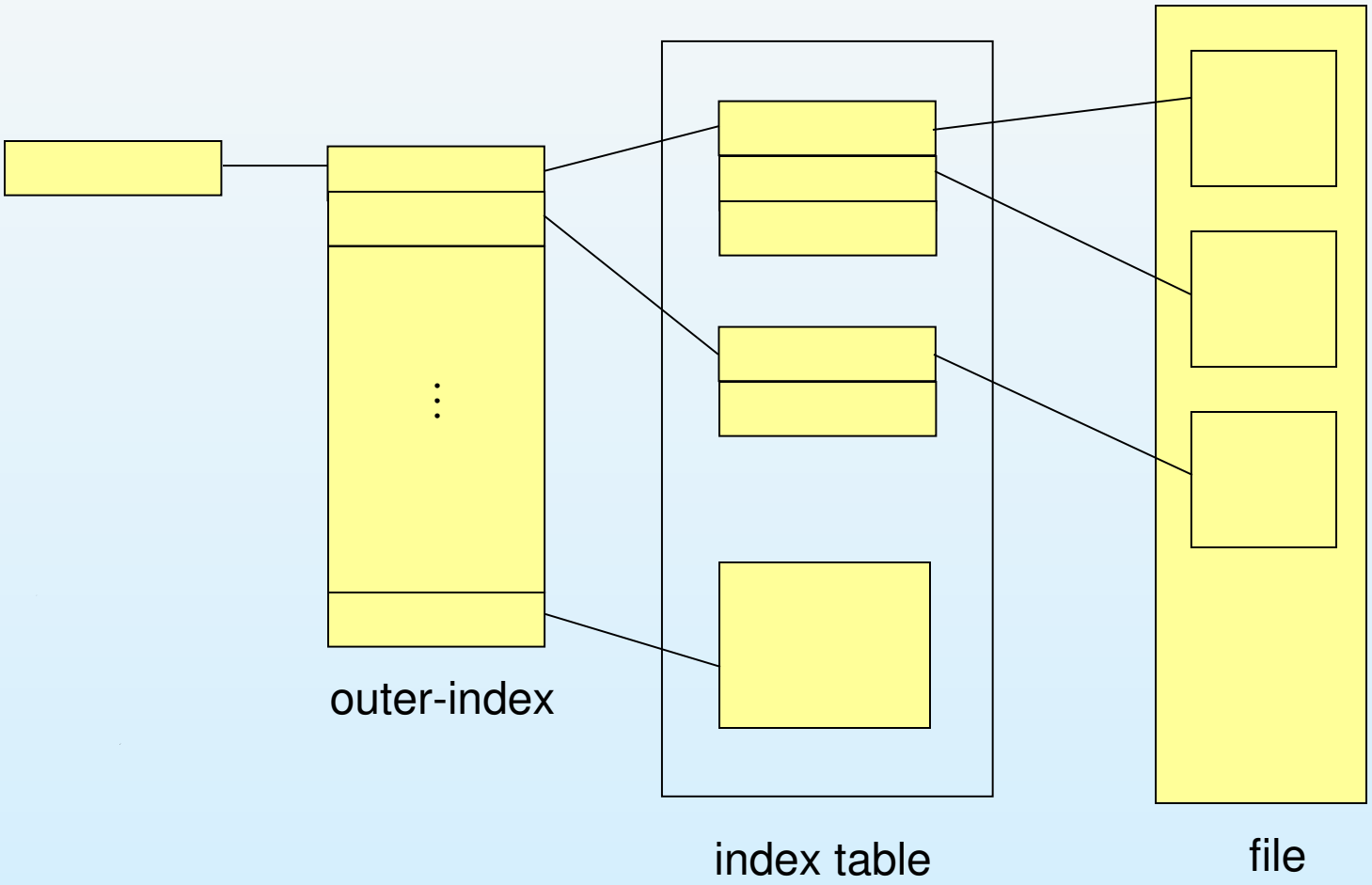
Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access
- No external fragmentation
- Overhead of index block

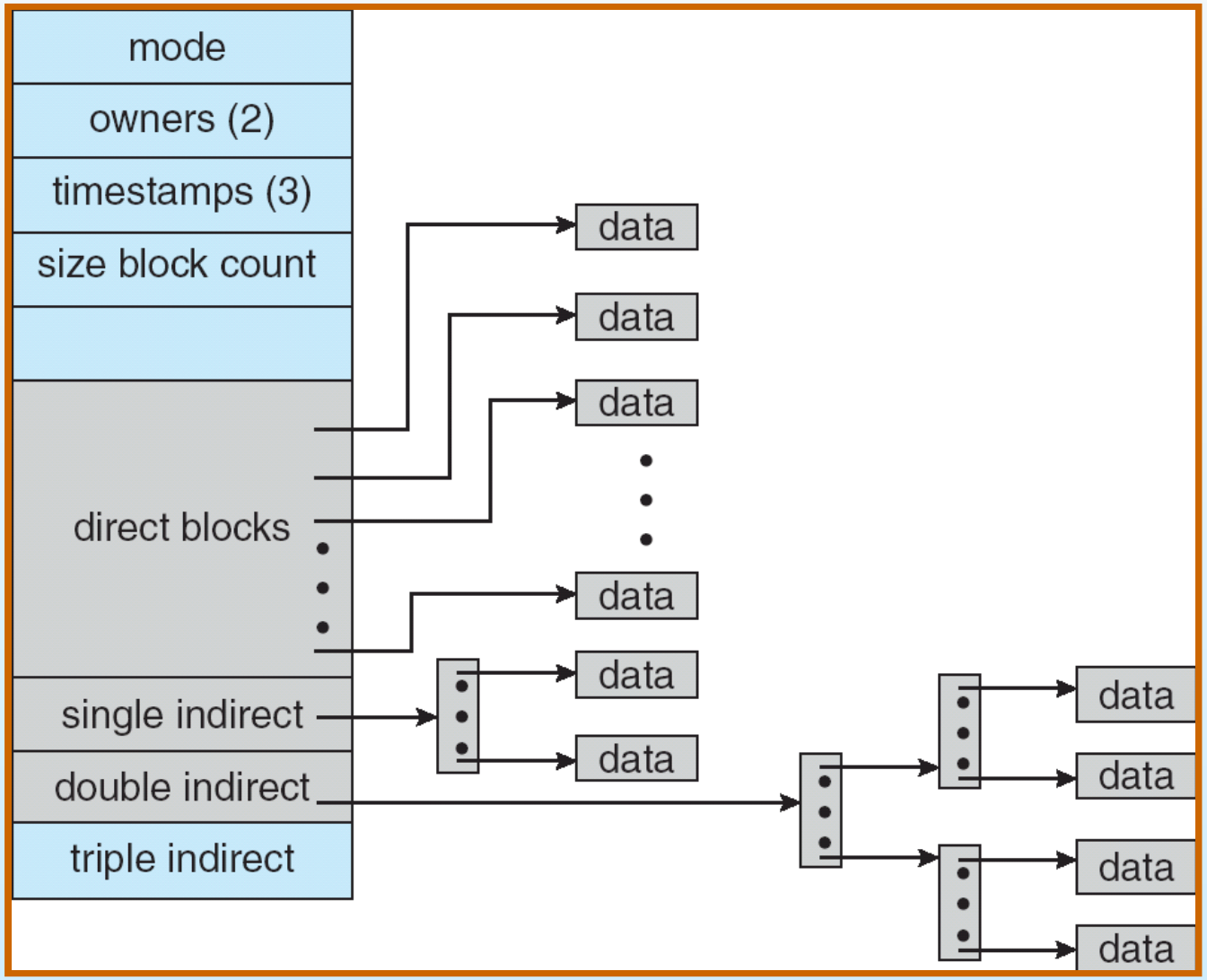
Indexed Allocation – Mapping (Cont.)

- Mapping from logical to physical in a file of unbounded length
  - Linked scheme
    - Link blocks of index table (no limit on size).
  - Multilevel index
    - First-level index points to a set of second-level index and so on...
  - Combined scheme
    - Point to data block or lower-level index

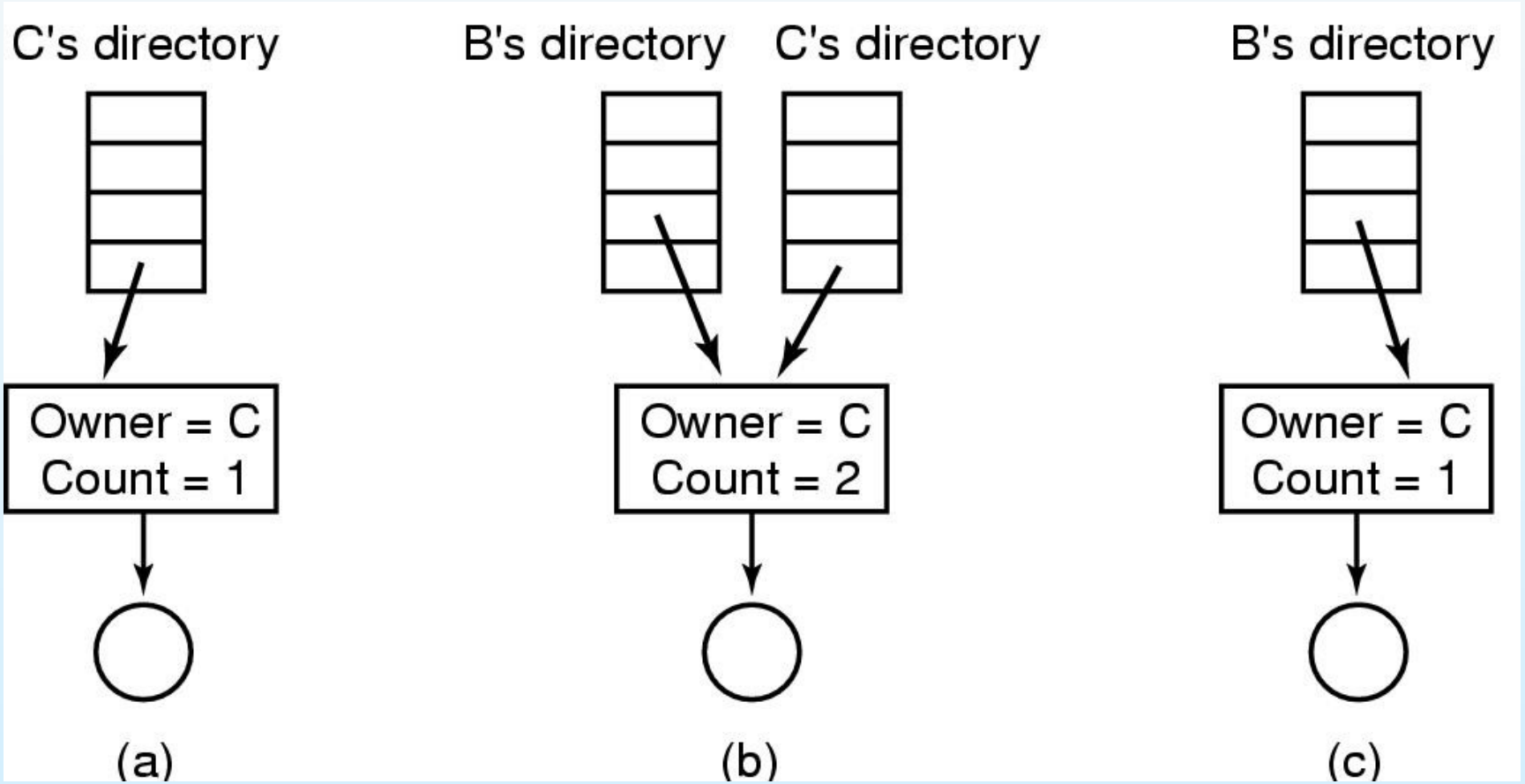
Indexed Allocation – Multilevel Index



Combined Scheme: UNIX (4K bytes per block)

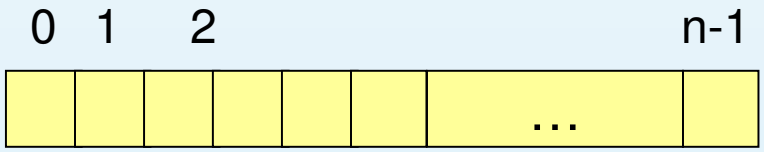


Shared Files



Free-Space Management

- Bitmap (*n* blocks)
  - bit[i] = 0, block i is free
  - bit[i] = 1, block i is occupied

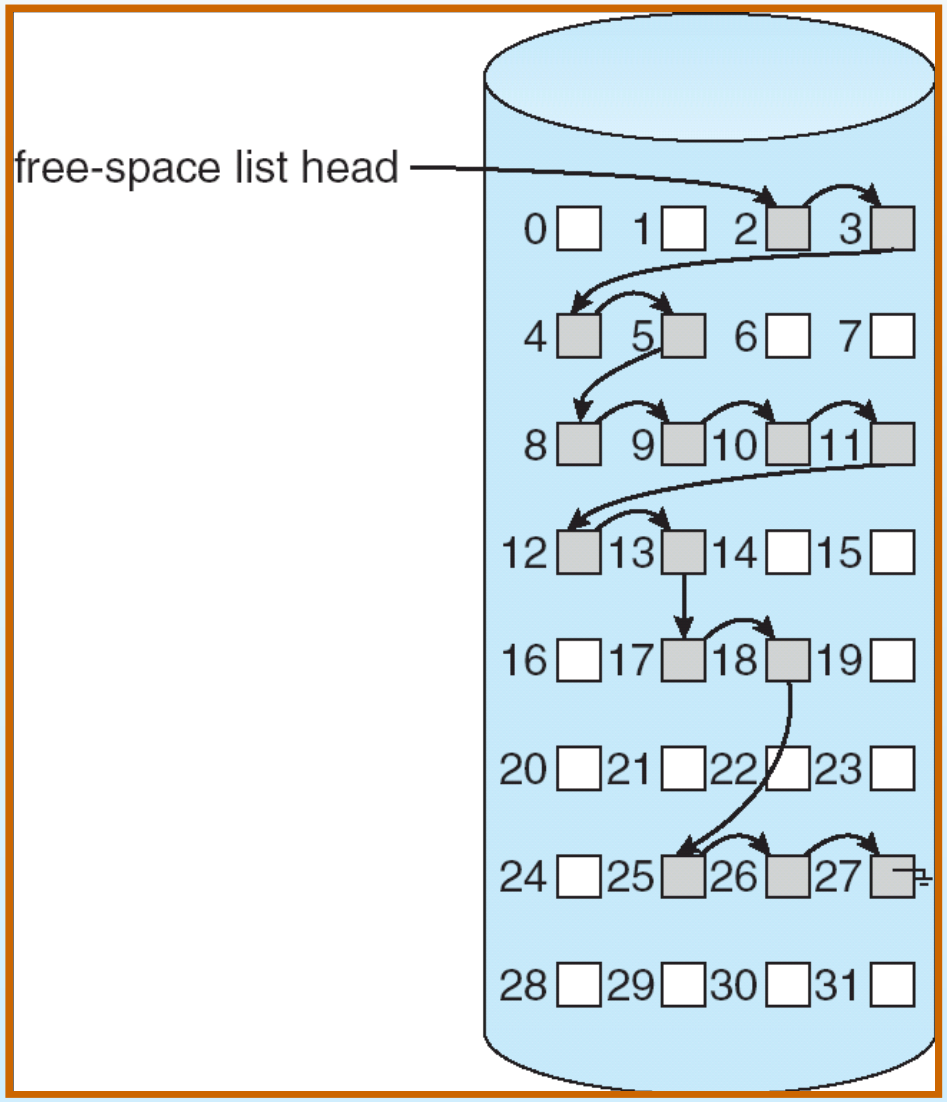


- Easy to get contiguous files
- Bitmap requires extra space

Free-Space Management (Cont.)

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
  - Store the addresses of n free blocks in one free block
- Counting
  - Keep the address of block *i*
  - And the number of free contiguous blocks following *i*

Linked Free Space List on Disk

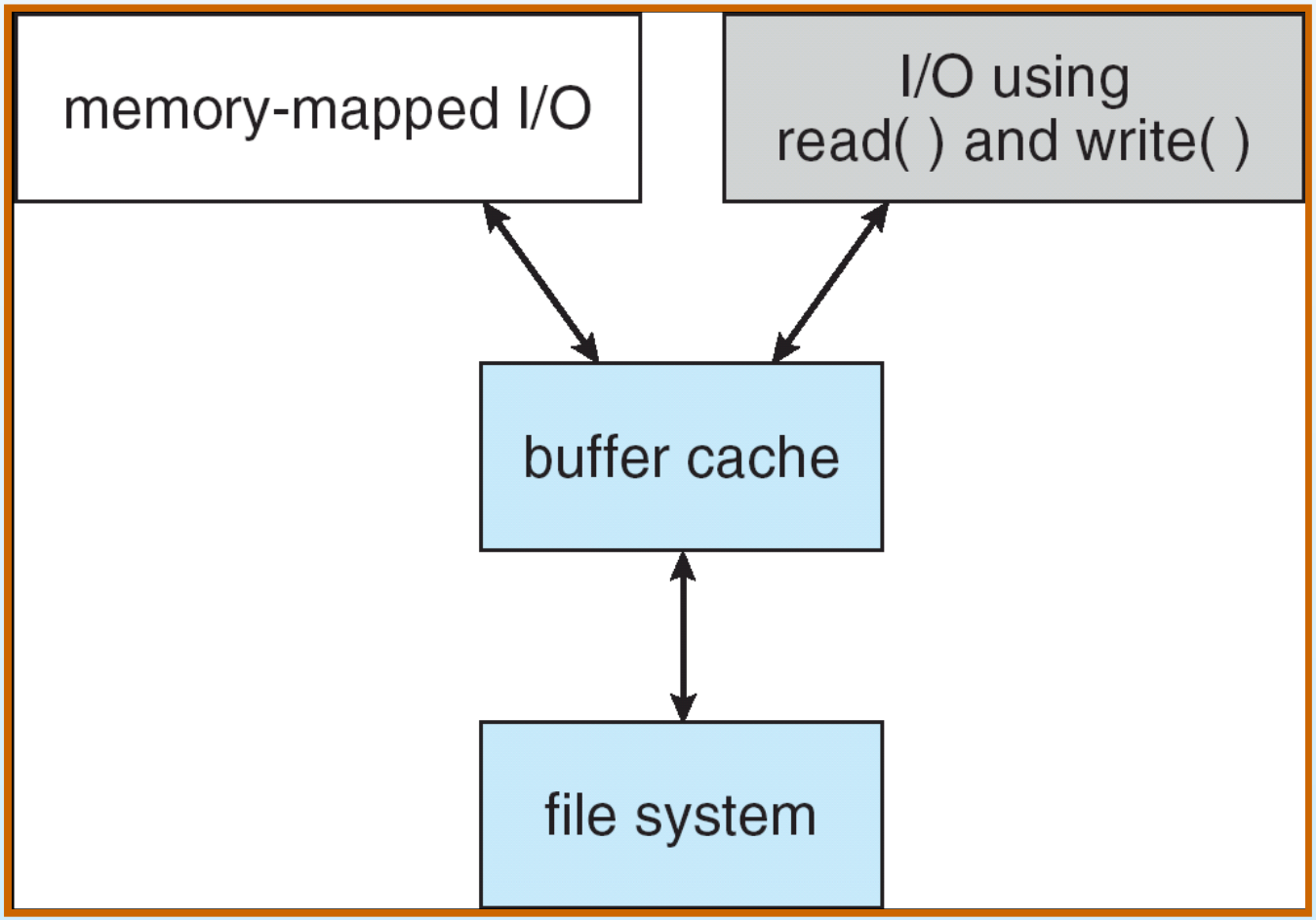


Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
    - Reduce seek time
  - types of data kept in file's directory entry
- Performance
  - disk cache
    - Synchronous write and asynchronous write
  - free-behind and read-ahead

Unified Buffer Cache

- Cache both memory-mapped pages and ordinary file system I/O



Recovery

- Consistency checking
  - Cache and computer crash makes data inconsistency
  - Compares data in directory structure with data blocks, and tries to fix inconsistencies
    - fsck in Unix/Linux, chkdsk in Windows
- Use system programs to back up data to another storage device
  - floppy disk, magnetic tape, other magnetic disk, optical, network
  - Full backup and incremental backup
- Recover by restoring data from backup
- Recover by accessing disk directly

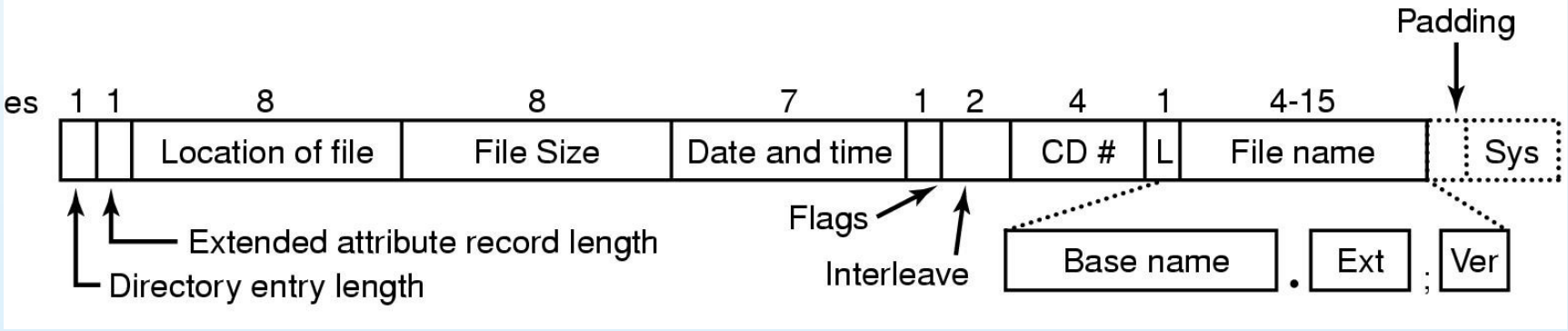
Log Structured File Systems

- Log structured (or journaling) file systems record each update to the file system as a transaction
  - NTFS, ext3, ZFS and so on...
- All transactions are written to a log
  - The transactions in log are asynchronously written to the file system
  - When the file system is modified, the transaction is removed from the log
    - No half progress
- If the system crashes, all remaining transactions in the log must still be performed

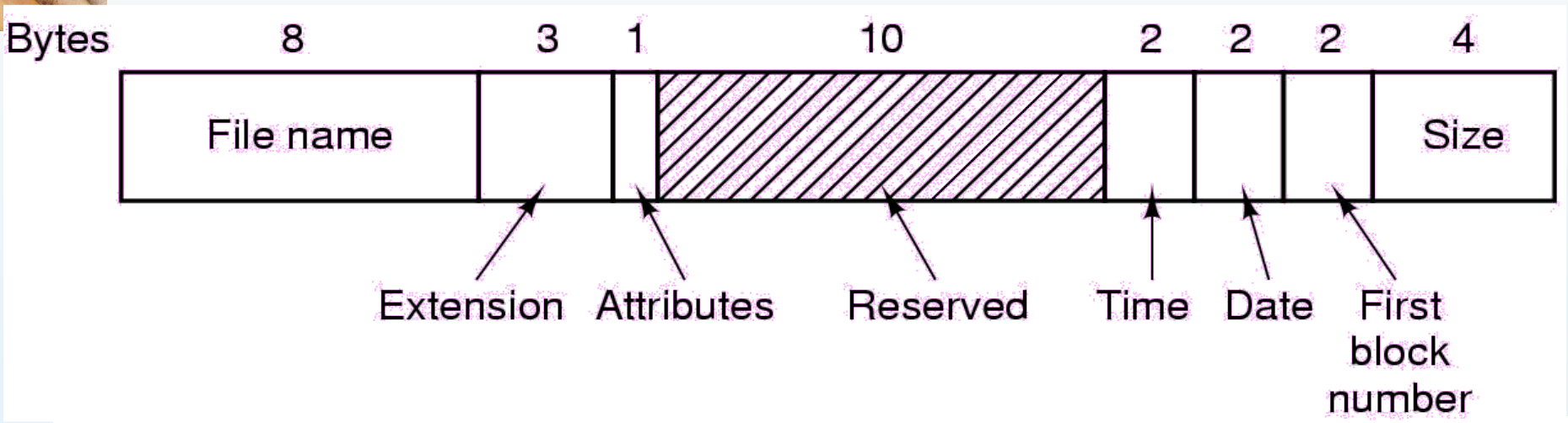
Performance of Log Structured FS

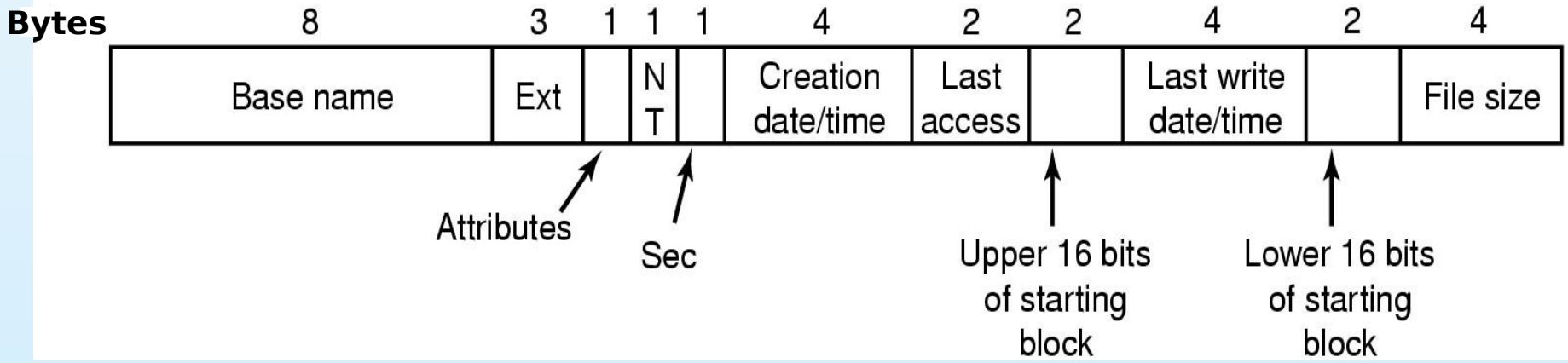
- Log space is continuous allocation
  - Synchronous write
  - Fast
- Transaction is submitted to disk when I/O is idle
  - Asynchronous write
  - Slow but can't feel
- Append data, no overwrite
  - Easier to recover
  - Sequential write
  - Free pages when idle

Example: ISO-9660 CD-ROM File Systems



Example: FAT16 & FAT32





FAT32 Long File Name

Bytes																															
68	d		o		g							A	0	CK											0						
3	o		v		e							A	0	CK	t		h		e			l		a	0		z		y		
2	w		n				f		o			A	0	CK	x				j		u		m	p	0		s				
1	T		h		e				q			A	0	CK	u		i		c		k			b	0		r		o		
T	H	E	Q	U	I	~	1					A	NT	S	Creation time					Last acc		Upp		Last write			Low		Size			

Introduction to NTFS

- Standard file system of Windows NT family
- NTFS v3.1 introduced since Windows XP
  - Vista gives some improvement
- Partly supported in Linux kernel
- Full supported in Linux through NTFS-3G
  - NTFS-3G is built on file system in userspace (FUSE)

Features of NTFS

- Alternate data stream (ADS)
  - One file with multi contents
  - Security feature
- Quotas
  - Limit the total space a user can use
- Sparse files
  - Allowing an application to specify regions of empty data
- Volume mount point
- Hard links and symbolic links
- Volume shadow copy
  - Keeps historical versions of files and directories

Features of NTFS (Cont.)

- File compression
  - Auto zip
- Single instance storage (SIS)
  - Save the same files as one copy
  - Copy-on-write
- Encrypting file system (EFS)
  - Real safe!
- Transactional NTFS
- USN Journal

Introduction to ZFS

- Developed by Sun in Solaris
  - Open source
- The first 128-bit file system on earth
  - Project leader Bonwick said,
  - "Populating 128-bit file systems would exceed the quantum limits of earth-based storage. You couldn't fill a 128-bit storage pool without boiling the oceans."
- Ported to BSD, Mac OS X and Linux
  - Unfinish

Features of ZFS

- Built on top of virtual storage pools called *zpools*
  - Constructed of vdevs (files, partitions or disks)
  - May be configured as non-redundantly, a mirror...
- Copy-on-write transactional model
  - Active blocks are never overwritten
- Snapshots and clones
  - You can know the history of any file
- Dynamic striping
  - Dynamically add vdevs and auto load balance
- Variable block sizes
- Filesystem encryption in a beta stage

End of Chapter 11