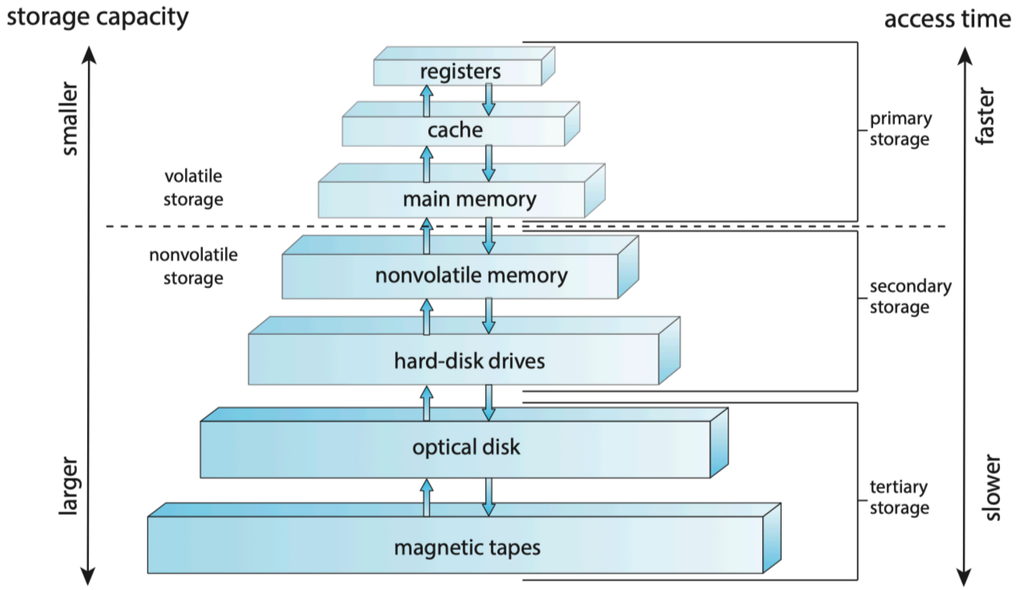
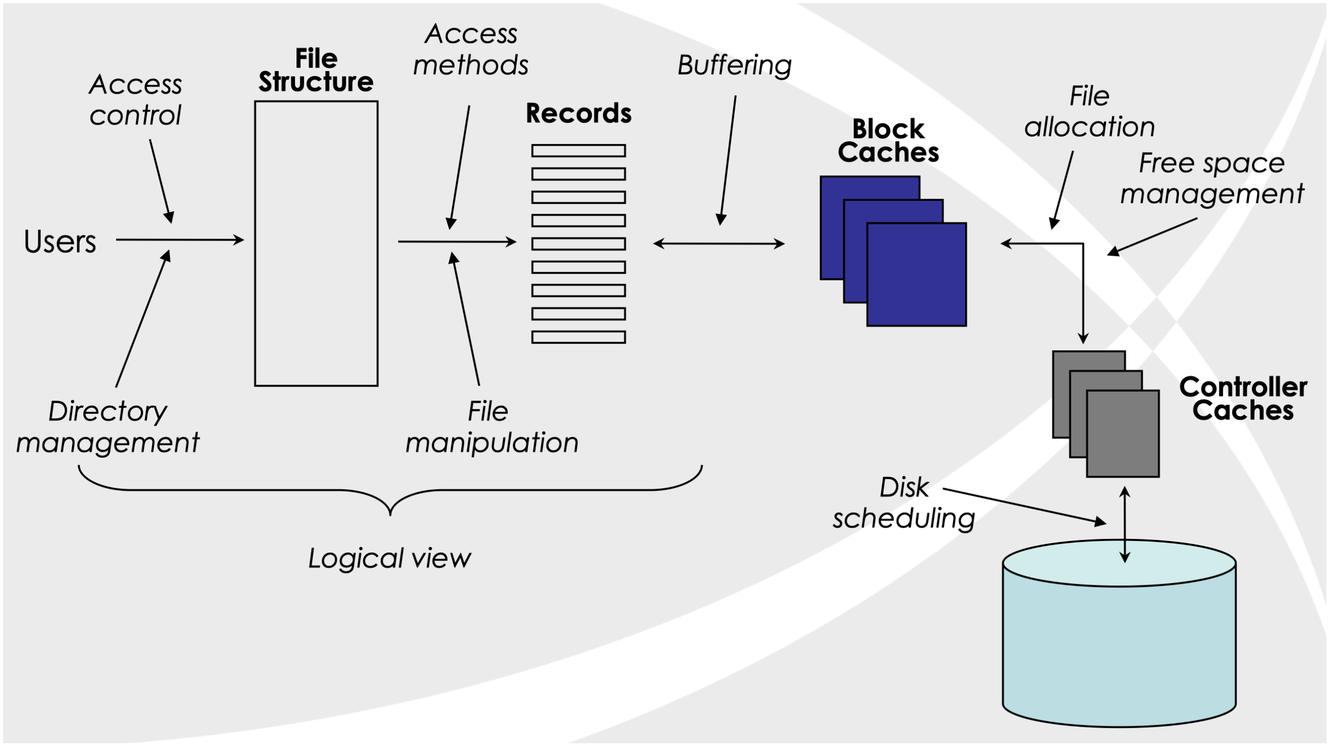
File System Implementation

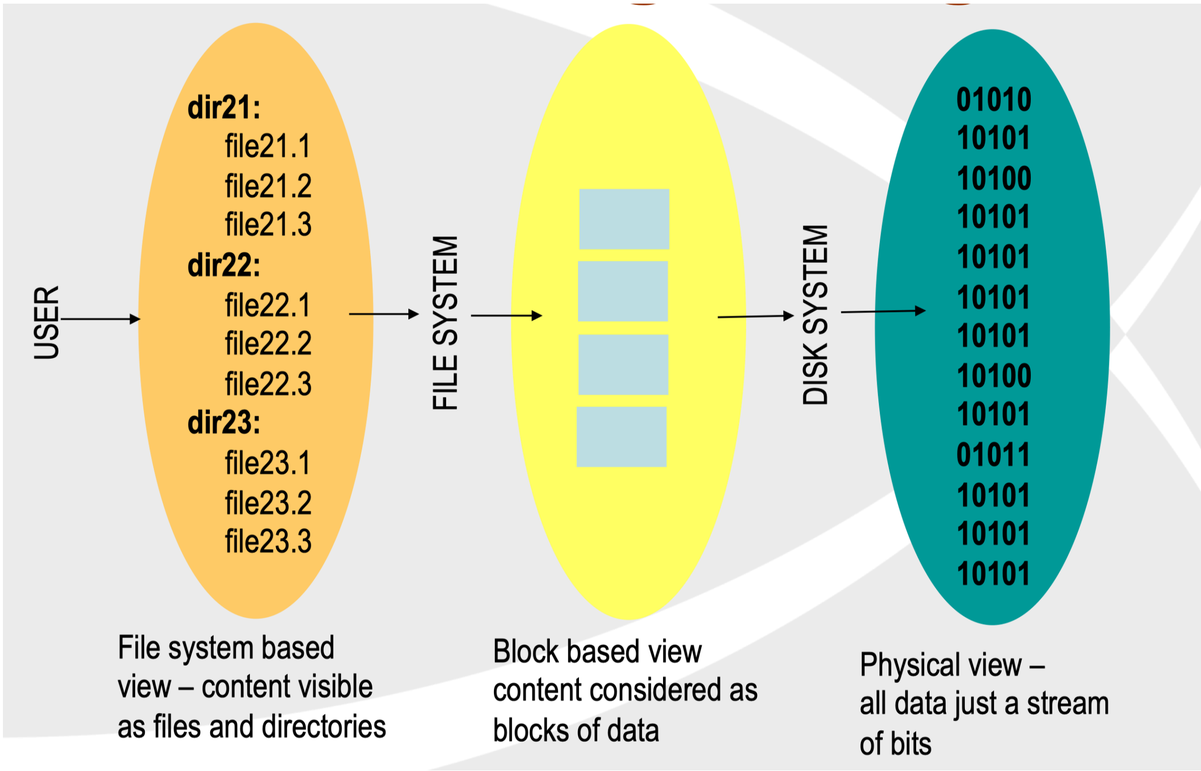
Storage System Overview

- Storage system is the persistent layer in the memory hierarchy
- Storage system is also much slower than the volatile memory
- Storage system is still directly addressable
- We need some way of mitigating the performance gap - use blocking



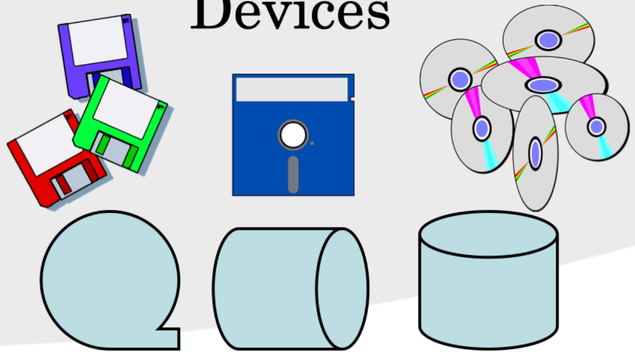
Storage System Overview





File System Abstraction

	Objects	*Typical operations*
Interactive (Shells)	files	copy, delete, rename
Applications and system programs	logical elements (records)	open/close, buffering seek (logical)
File System		file system check soft repair partitioning
Devices	physical elements (head, cylinder, …)	raw read/write, seek (physical) low-level format

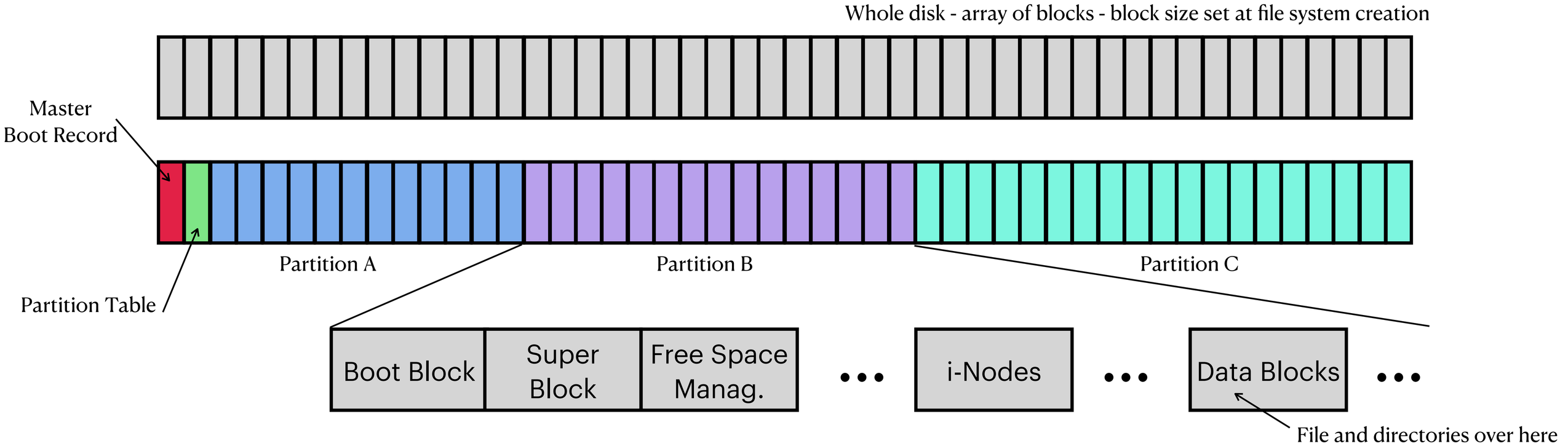


Boot Block: Bootstrapping code for the OS in the partition

File System Layout

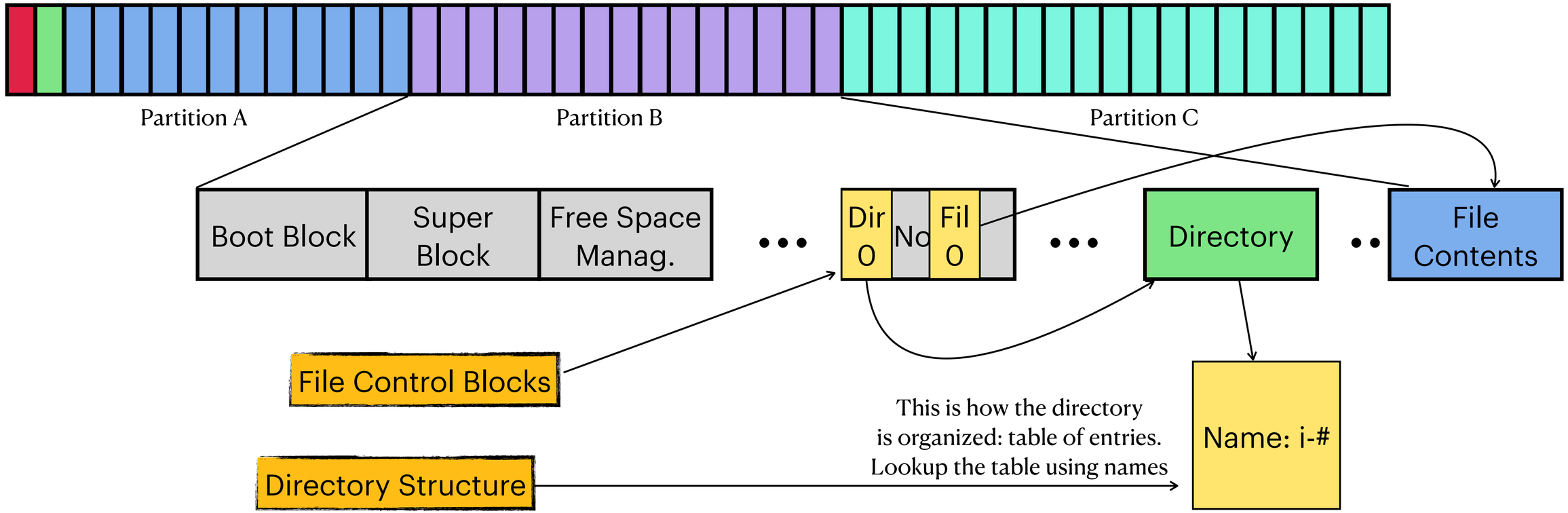
Super Block: Defining the geometry of the partition

- Disk is an array of blocks (e.g., 4KB sized blocks - size determined using performance factors)
- What to store in the different blocks?



Directory Implementation

- Directory is the symbol table that holds information about all directories and files - points to the file control block - telling you where the data blocks are



Directory Implementation

- **Linear List**
  - Directory is a linear list of names with pointers (as shown in the previous slide)
  - Need to search the whole list at creation time for uniqueness
  - At deletion an entry in the list can be marked and then reused
  - Linear list can be implemented using a linked list (in disk!)
- **Hash table**
  - Name is hashed into a slot in a table
  - Size of the table, handling collisions are some of the issues to deal with here

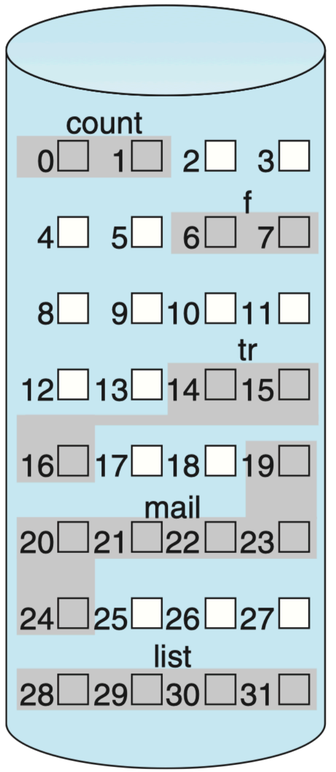
Balanced trees can be used to improve linear list

File Space Allocation

- Directory links the name of a file or subdirectory with a file control block (FCB)
- FCB points to all data blocks
- We need a way of allocating data blocks - file space allocation problem
- File space allocation can be done in three major ways
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

Contiguous Allocation

- File must occupy a set of contiguous blocks
- Accessing is easy for any part of the file: both sequential and direct access are supported
- Finding space for a new allocation is hard due to external fragmentation
- Growing a file is also hard - no space to grow where the file is located
- Can copy the whole file to a new location and release the old blocks
- Compacting (a time consuming activity) is necessary to reduce external fragmentation

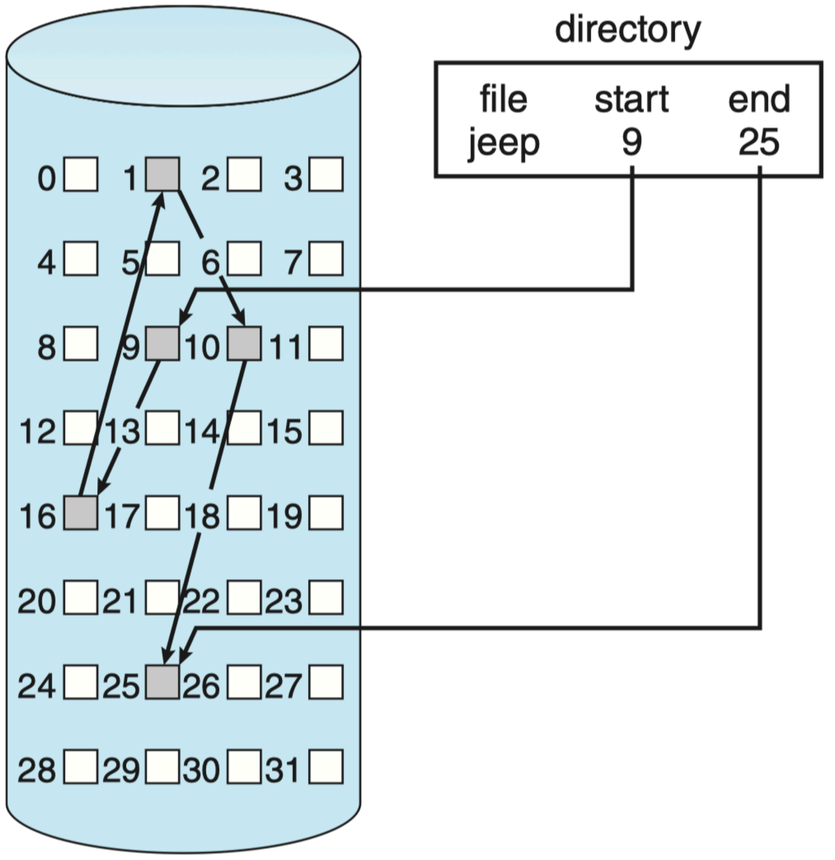


directory

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

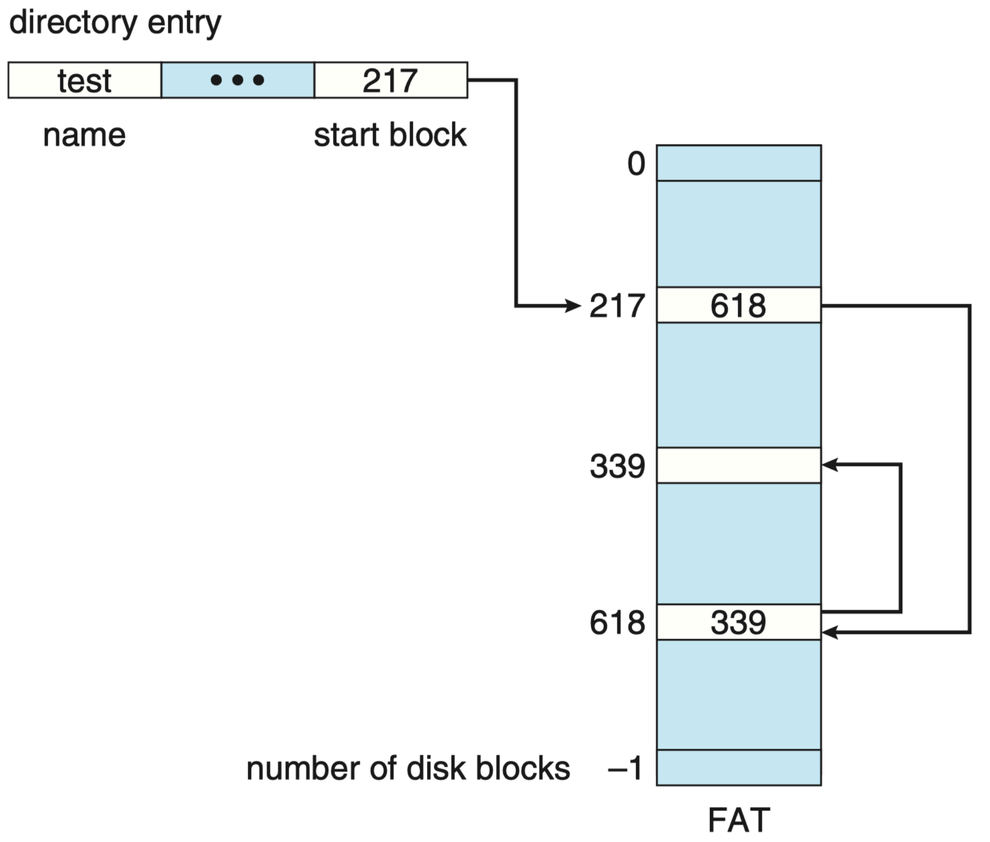
Linked Allocation

- File is a linked list of storage
- Directory contains the first and last blocks of the file
- Each block contains a pointer to the next block
- Linked allocation is effective for sequential access, but not for direct access
- No external fragmentation
- Block includes a pointer - small storage is lost for pointers in each block
- File system integrity is lost quite easily - single corrupt block can destroy files and directories



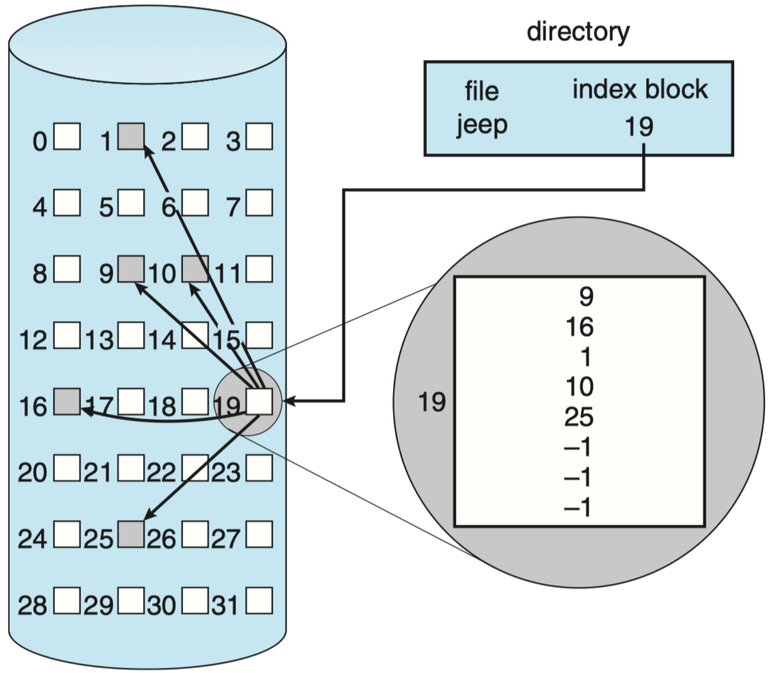
File Allocation Table

- Some blocks on disk are set aside for a file allocation table
- Directory entry points to the FAT entry for the file
- Add a new block - insert the new block into the linked list (could be at the end or in the middle)
- FAT can be cached in memory and disk seeks can be avoided
- FAT allows efficient direct access



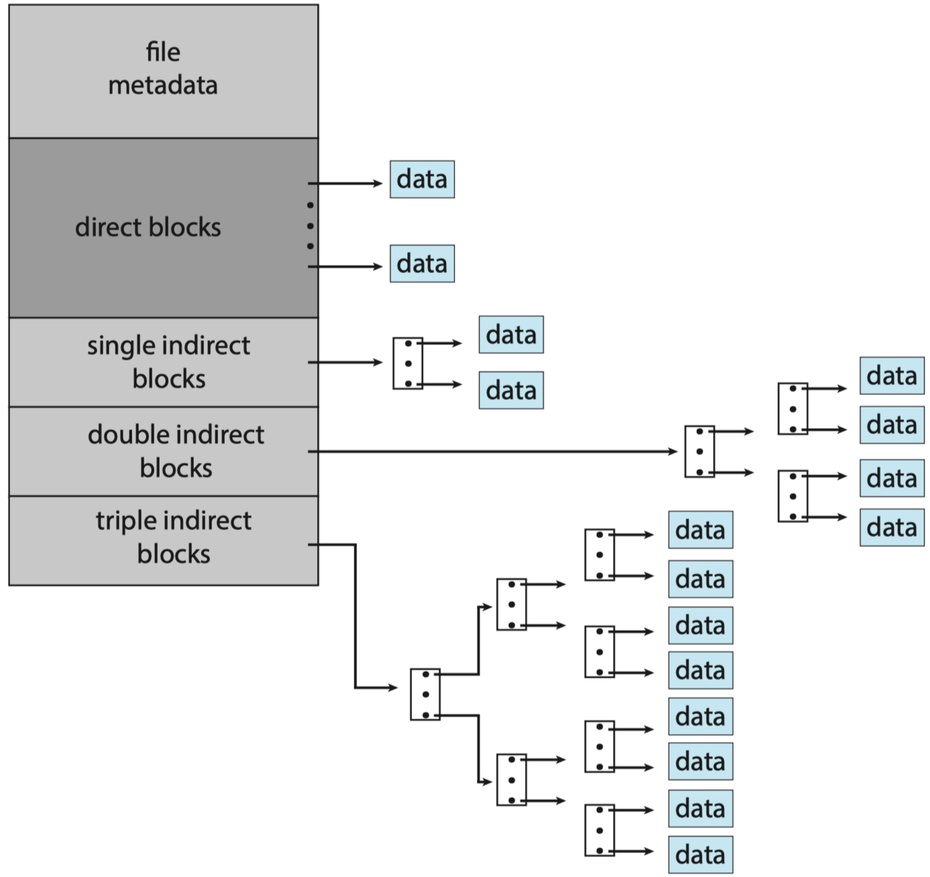
Indexed Allocation

- All the indexes are collected into an index block
- Indexed allocation supports direct access, no external fragmentation
- Indexed allocation can waste space - internal fragmentation inside data blocks and index blocks
- Pointer overhead for indexed allocation is more than linked allocation



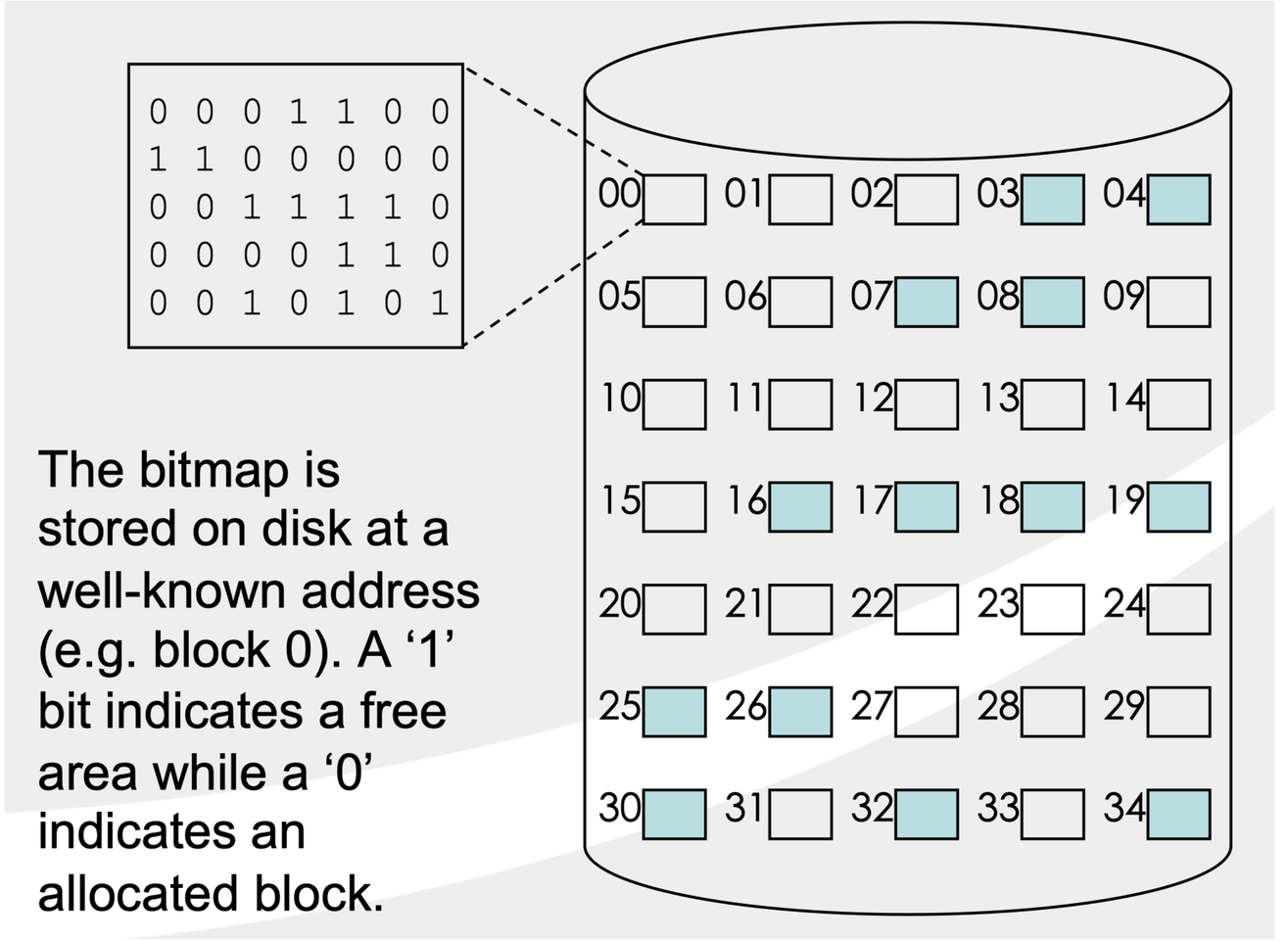
i-Nodes (Index Allocation in UNIX)

- UNIX i-node is organized using the indexed allocation scheme
- 12 direct block pointers, 1 single indirect block pointer, 1 double indirect block pointer, 1 triple indirect block pointer
- Lets say block size is B and a disk pointer takes 4 bytes, the largest file size is given by
  $12 \times B + B/4 \times B + (B/4)^2 \times B + (B/4)^3 \times B$



Free Space Management

- Secondary storage has finite space - so we need to reclaim storage
- Keep track of free storage blocks and reuse them in future allocations
- Bitmap or bit vector - each block is represented by 1 bit
- Many CPUs have bit manipulation instructions that can scan for free blocks in the bit map efficiently
- 1 TB disk with 4KB blocks would require 32 MB to store bitmap $2^{40}/2^{12} = 2^{28}$ bits $= 2^{25}$ bytes $= 2^{5}$ MB



Linked List of Free Blocks

- First free block is pointing to the next block
- Scheme is not efficient - free block traversal is not done all the time
- Adding a block is quite efficient
- Need some modification to support contiguous allocation
- Free block pointer + count of free contiguous blocks at that location

For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bitmap would be

00111100111111000110000011100000 ...