File Systems (2)

May 30, 2018
Byung-Gon Chun

Acknowledgments. Slides and/or picture in the following are adapted from UW, Columbia, and UC Berkeley slides

Outline

- File system concepts
  - What is a file?
  - What operations can be performed on files?
  - What is a directory and how is it organized?

- File implementation
  - How to allocate disk space to files?

Typical File Access Patterns

- Sequential Access
  - Data read or written in order
    - Most common access pattern
      - E.g., copy files, compiler read and write files,
  - Can be made very fast (peak transfer rate from disk)

- Random Access
  - Randomly address any block
    - E.g., update records in a database file
  - Difficult to make it fast (seek time and rotational delay)

Disk Management

- Need to track where file data is on disk
  - How should we map logical sector # to surface #, track #, and sector #?
    - Order disk sectors to minimize seek time for sequential access
- Need to track where file metadata is on disk
- Need to track free versus allocated areas of disk
  - E.g., block allocation bitmap (Unix)
    - Array of bits, one per block
    - Usually keep entire bitmap in memory

CSE 컴퓨터공학부 Department of Computer Science & Engineering

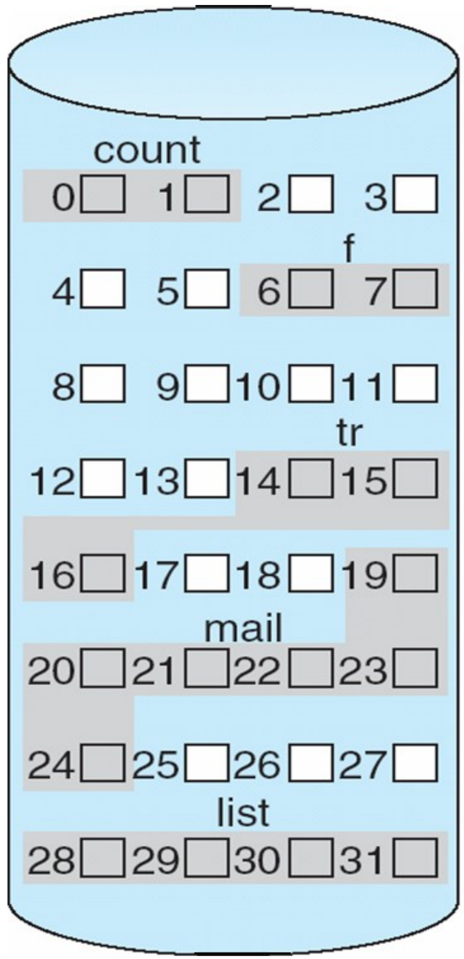
Allocation Strategies

- Various approaches (similar to memory allocation)
  - Contiguous
  - Extent-based
  - Linked
  - FAT tables
  - Indexed
  - Multi-level indexed
- Key metrics
  - Fragmentation (internal & external)?
  - Grow file over time after initial creation ?
  - Fast to find data for sequential and random access?
  - Easy to implement?
  - Storage overhead?

Contiguous Allocation

- Allocate files like continuous memory allocation (base & limit)
  - User specifies length, file system allocates space all at once
  - Can find disk space by examining bitmap
  - Metadata: contains starting location and size of file

Contiguous Allocation Example



directory

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

Pros and Cons

- Pros
  - Easy to implement
  - Low storage overhead
    (two variables to specify disk area for file)
  - Fast sequential access since data stored in continuous blocks
  - Fast to compute data location for random addresses.
  - Just an array index

- Cons
  - Large external fragmentation
  - Difficult to grow file

Extent-based Allocation

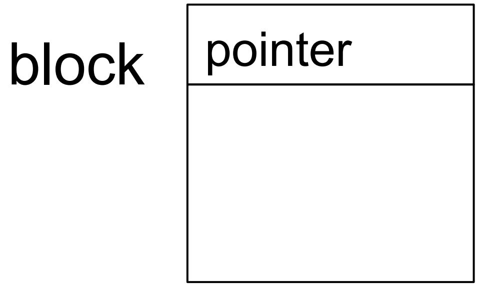
- Multiple contiguous regions per file (like segmentation)
  - Each region is an extent
  - Metadata: contains small array of entries designating extents
    - Each entry: start and size of extent

Pros and Cons

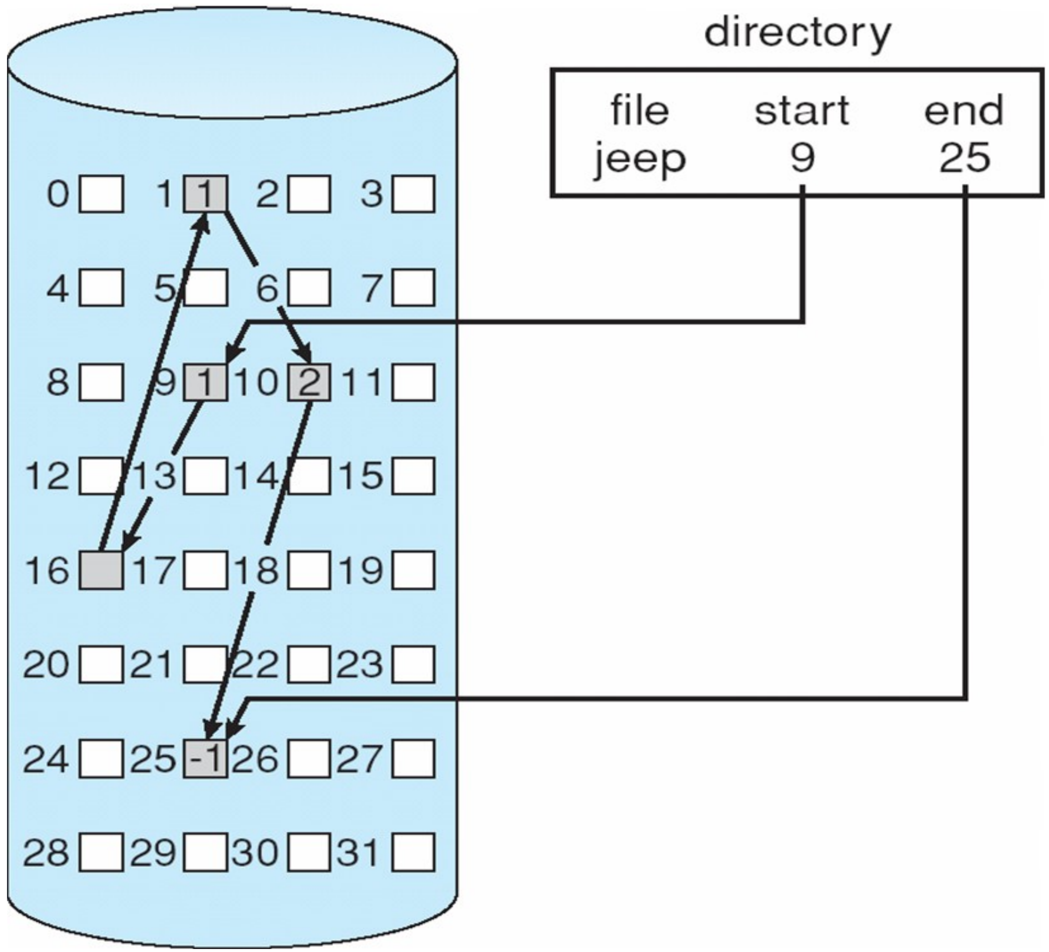
- Pros
  - Easy to implement
  - Low storage overhead (a few entries to specify file blocks)
  - File can grow overtime (until run out of extents)
  - Fast sequential access
  - Simple to calculate random addresses
- Cons
  - Help with external fragmentation, but still a problem

Linked Allocation

- All blocks (fixed-size) of a file on linked list
  - Each block has a pointer to next
  - Metadata: pointer to the first block



Linked Allocation Example



Pros and Cons

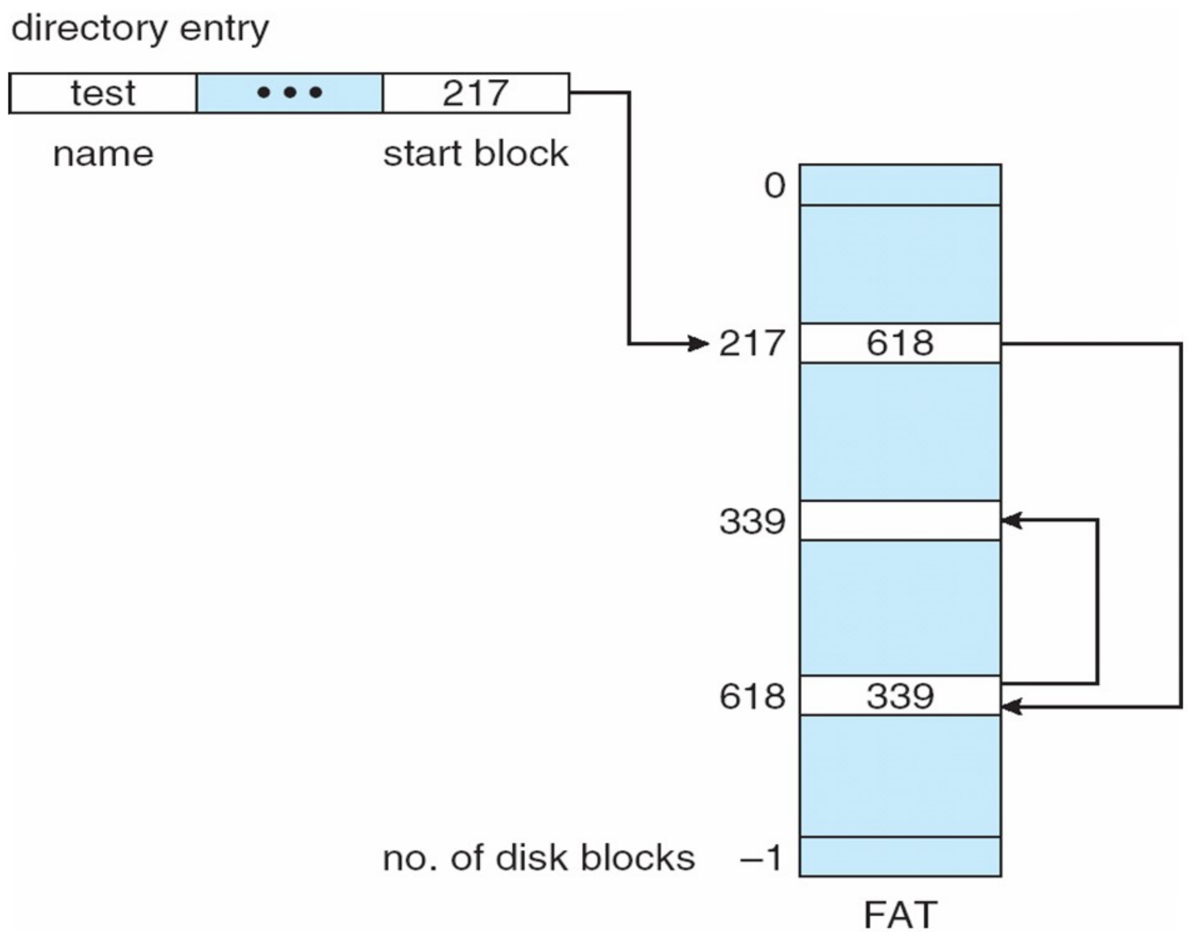
- Pros
  - No external fragmentation
  - Files can be easily grown with no limit
  - Also easy to implement, though awkward to spare space for disk pointer per block
- Cons
  - Large storage overhead (one pointer per block)
  - Potentially slow sequential access
  - Difficult to compute random addresses

Variation: FAT table

- Store linked-list pointers outside block in File-Allocation Table
  - One entry for each block
  - Linked-list of entries for each file

- Used in MSDOS and Windows operating systems

FAT Example



directory entry
test
• • •
217
name
start block
0
217
618
339
339
618
no. of disk blocks
–1
FAT

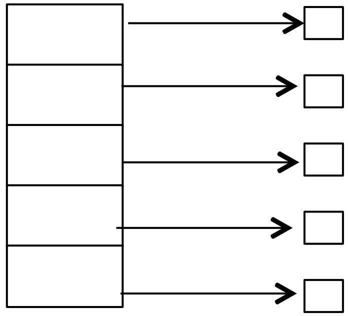
Pros and Cons

- Pros
  - Fast random access. Only search cached FAT
- Cons
  - Large storage overhead for FAT table
  - Potentially slow sequential access

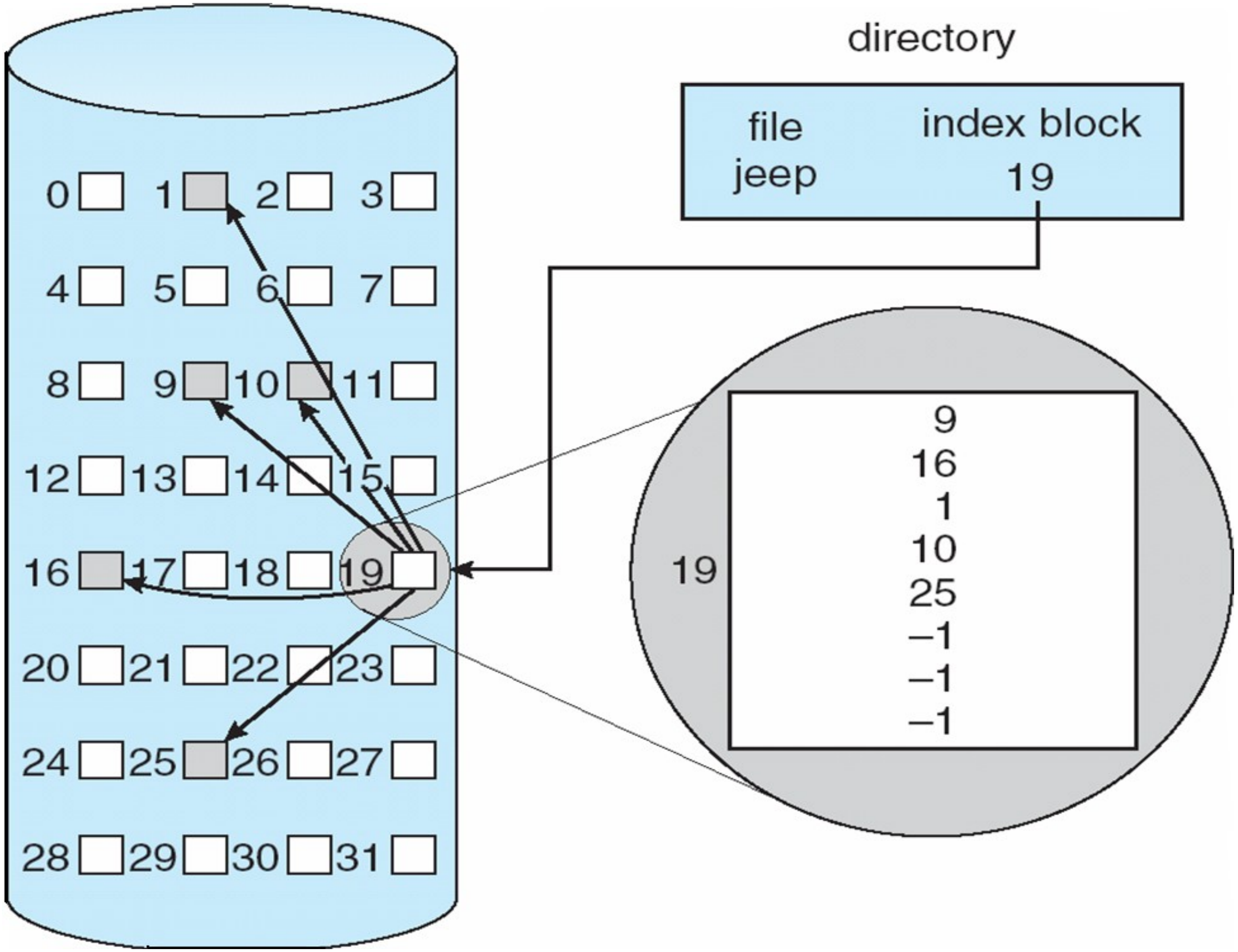
Indexed Allocation

- File has array of pointers (index) to block
  - Allocate block pointers contiguously in metadata
    - Must set max length when file created
    - Allocate pointers at creation, allocate blocks on demand
    - Cons: fixed size, same overhead as linked allocation
  - Maintain multiple lists of block pointers
    - Last entry points to next block of pointers
    - Cons: may need to access a large number of pointer blocks

block
pointers

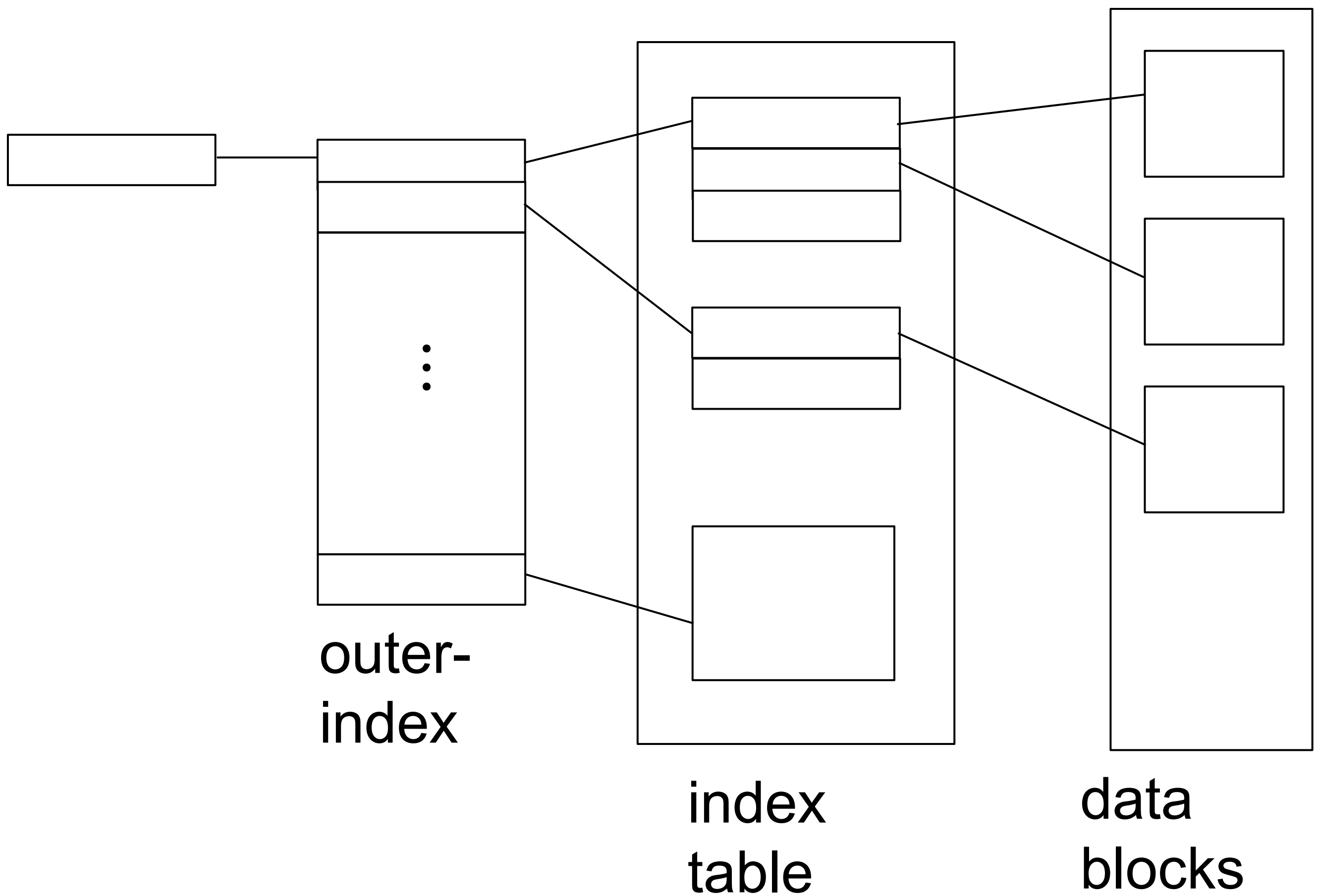


Indexed Allocation Example

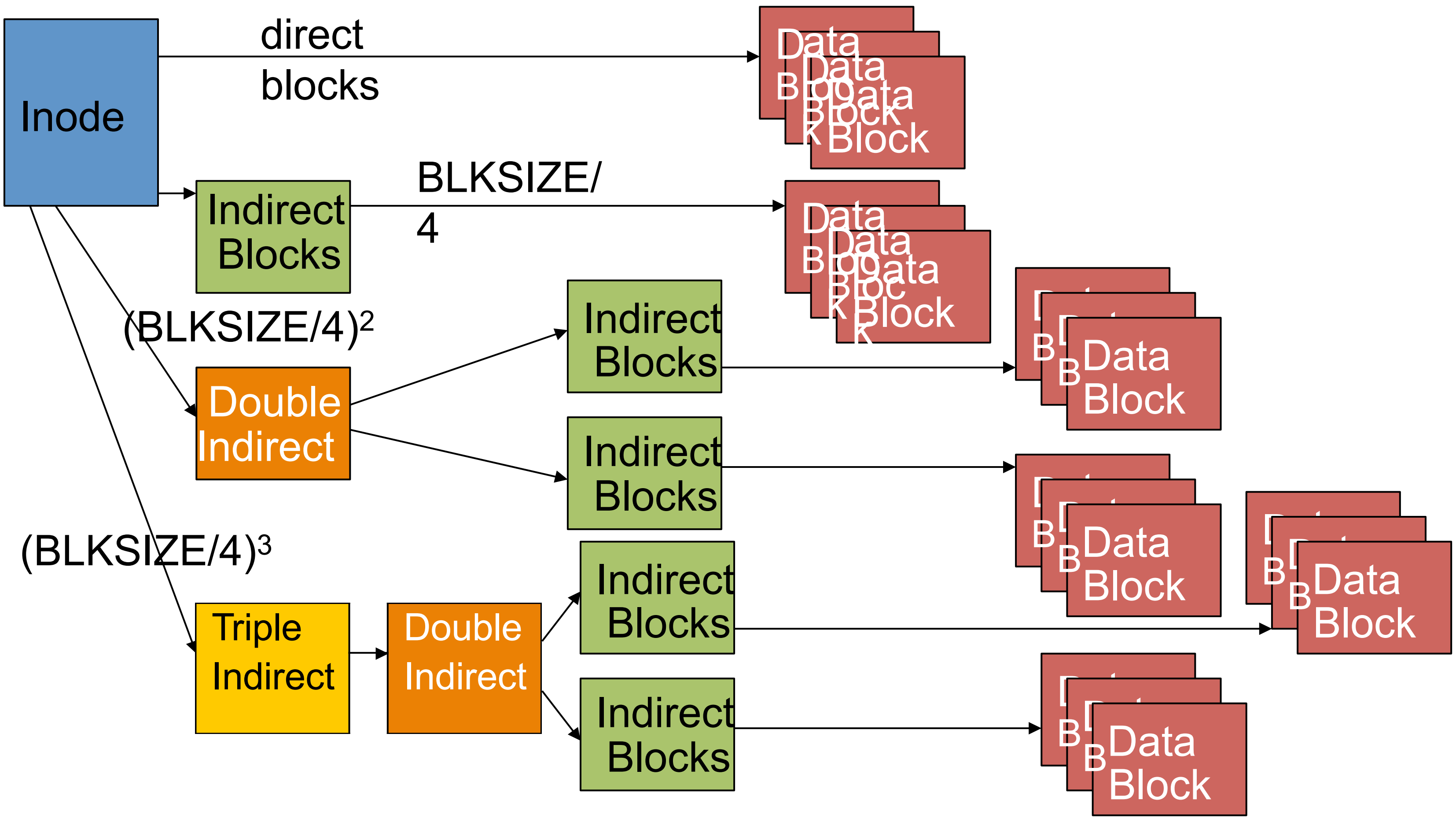


Multi-level Indexed Files

- Block index has multiple levels

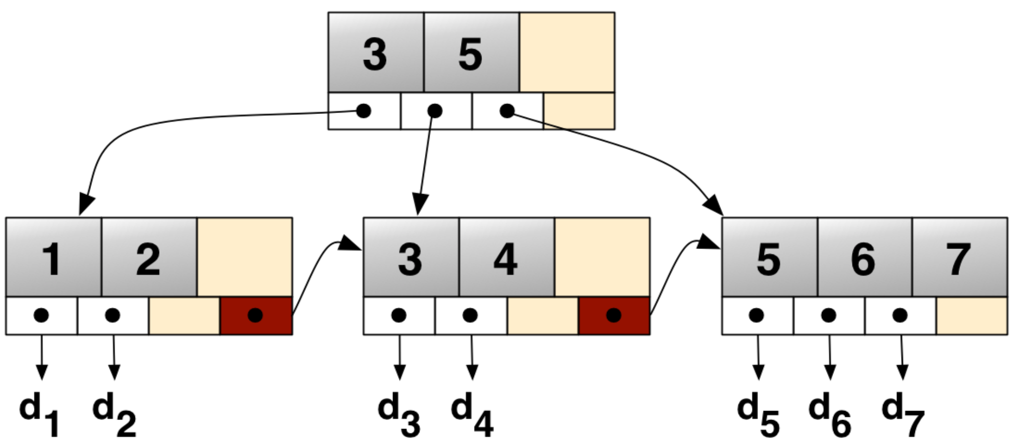


Multi-level Indexed Allocation (UNIX FFS, and Linux ext2/ext3)



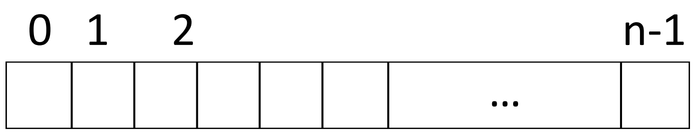
Advanced Data Structures

- More sophisticated data structures
- B+ Trees
  - Used by many high perf filesystems for directories and/or data
  - E.g., XFS, ReiserFS, ext4, MSFT NTFS and ReFS, IBM JFS, brzs
  - Can support very large files (including sparse files)
  - Can give very good performance (minimize disk seeks to find block)

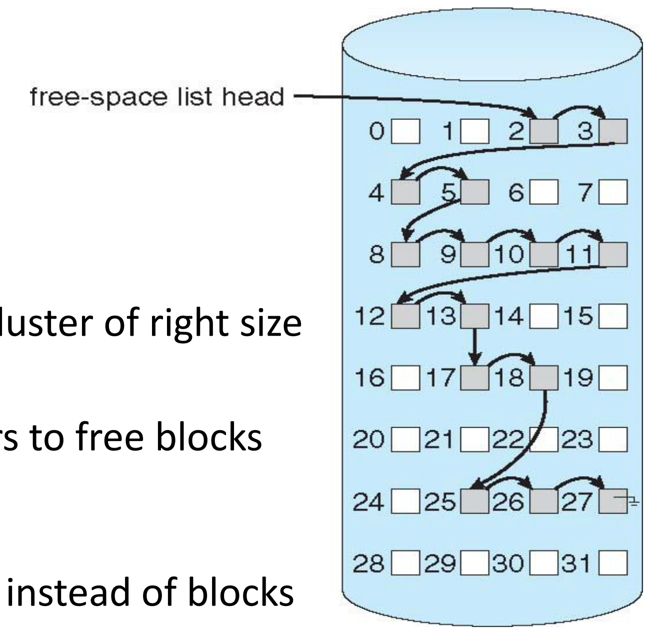


Free Space Management

- File system maintains **free-space list** to track available blocks/clusters
  - **Free bitmap** stored in the superblock



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



  - Linked free list
    - Pros: space efficient
    - Cons: requires many disk reads to find free cluster of right size
  - Grouping
    - Use a free index-block containing n-1 pointers to free blocks and a pointer to the next free index-block
  - Counting
    - Free list of variable sized contiguous clusters instead of blocks
    - Reduces number of free list entries