File Allocation

ECE469, Apr 4

Yiying Zhang

Announcement

- Reading: Ch 11

Review: FS hierarchy

- Most file systems use a tree structure (UNIX)
  - Why directories?
  - What is in a directory?
  - How do you find “/dir1/dir2/file3”?
  - How do you find “/”?
  - How to create a file in current dir?
  - How to implement renaming a file?
  - What is a hard link?
  - What is a soft link?

[lec21] File Meta-Data

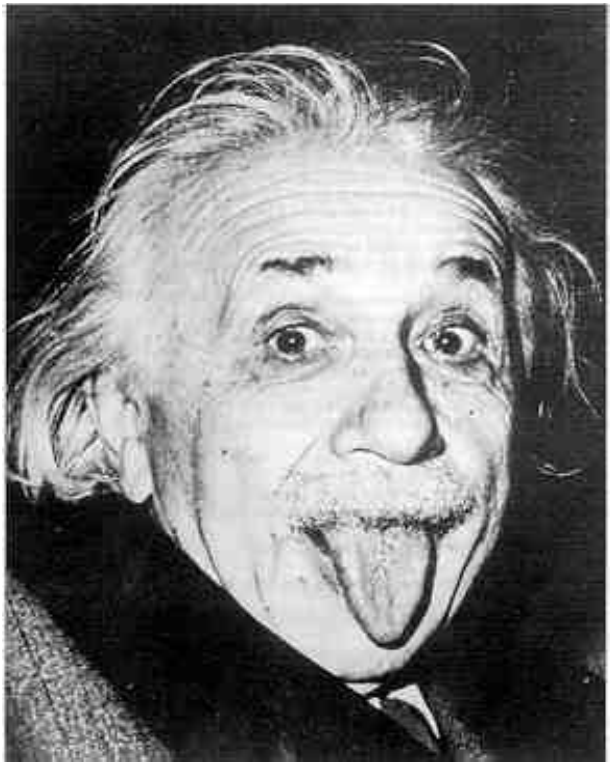
- Meta-data: Additional system information associated with each file
  - Name of file
  - Type of file
  - Pointer to data blocks on disk
  - File size
  - Times: Creation, access, modification
  - Owner and group id
  - Protection bits (read or write)
  - Special file? (directory? symbolic link?)
- Meta-data is stored on disk
  - Conceptually: meta-data can be stored as array on disk

[lec21] Per-file Metadata

- In Unix, the data representing a file is called an inode (for indirect node)
  - Inodes contain file size, access times, owner, permissions
  - Inodes contain information on how to find the file data (locations on disk)
- Every inode has a location on disk.

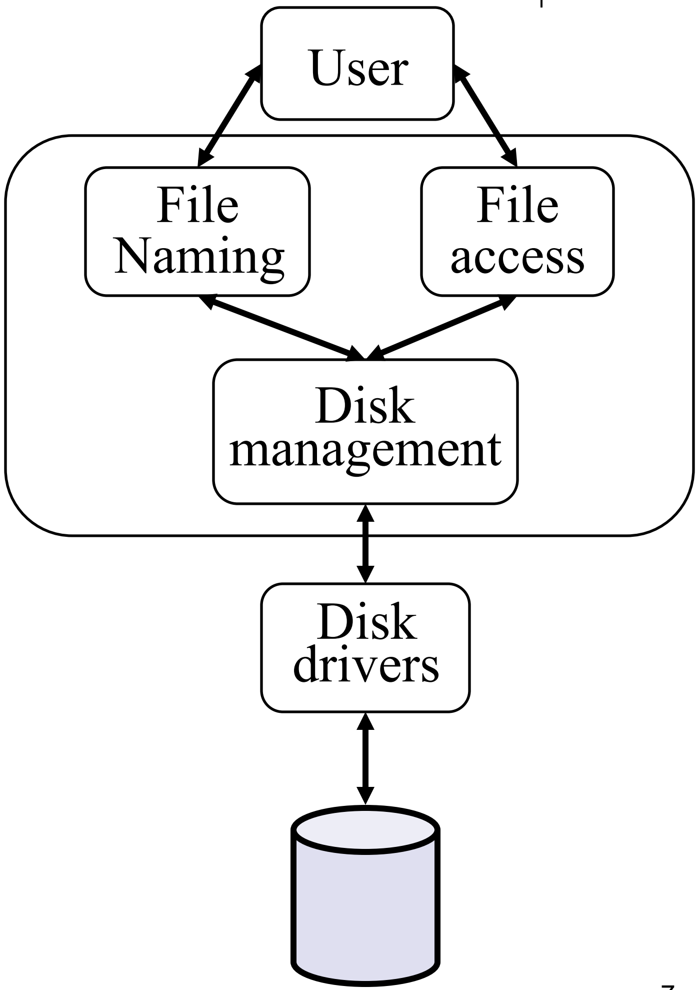
So What Makes File Systems Hard?

- Files grow and shrink
  - Little *a priori* knowledge
  - 6~8 orders of magnitude in file sizes
- Overcoming disk performance behavior
  - Highly nonuniform access
  - Desire for efficiency
- Coping with failure



File System Components

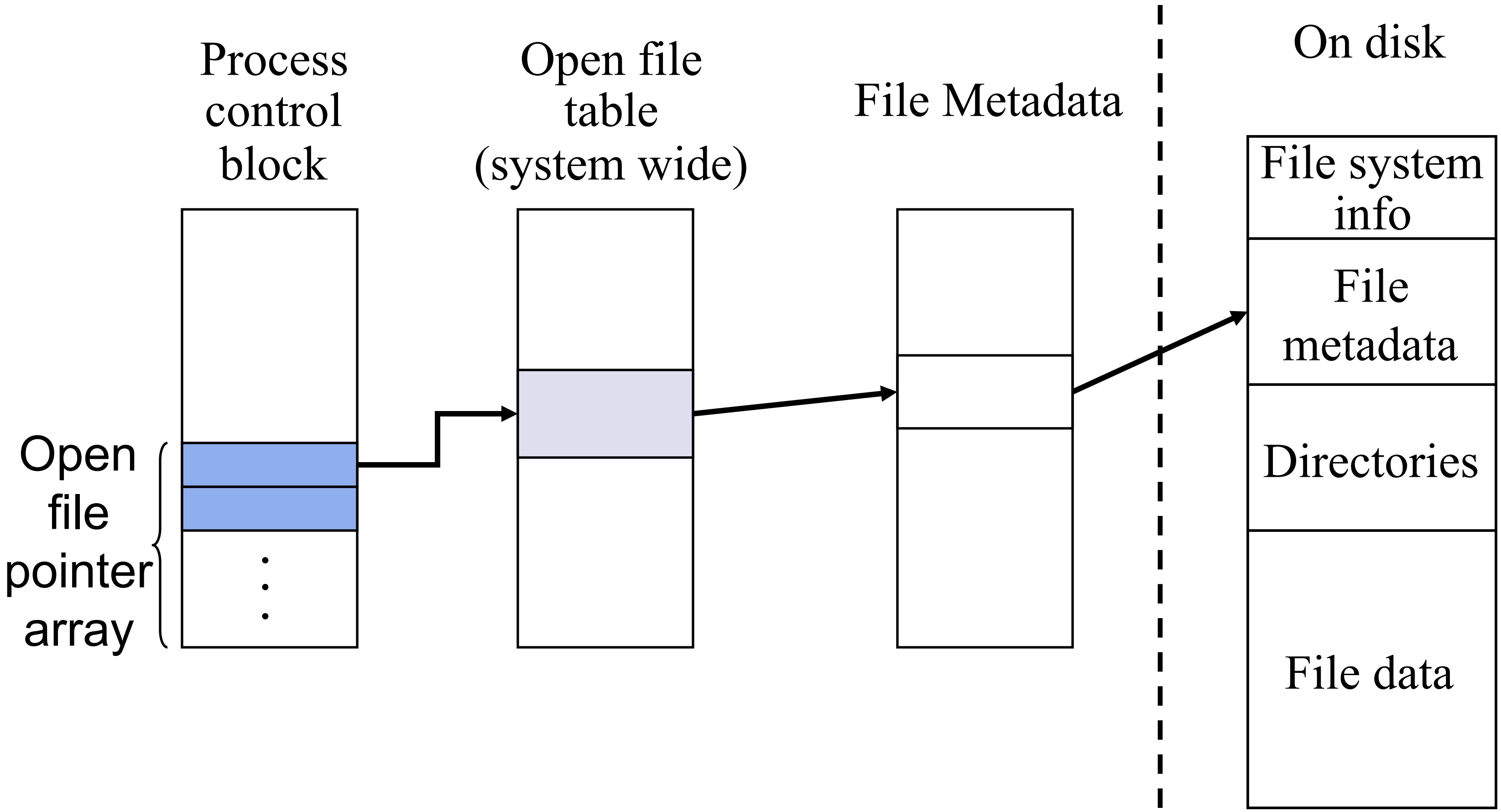
- Naming/Access
  - User gives file name, not track or sector number, to locate data
- Disk management
  - Arrange collection of disk blocks into files
- Protection and permission
  - Protect data from different users
- Reliability/durability
  - When system crashes, lose stuff in memory, but want files to be durable



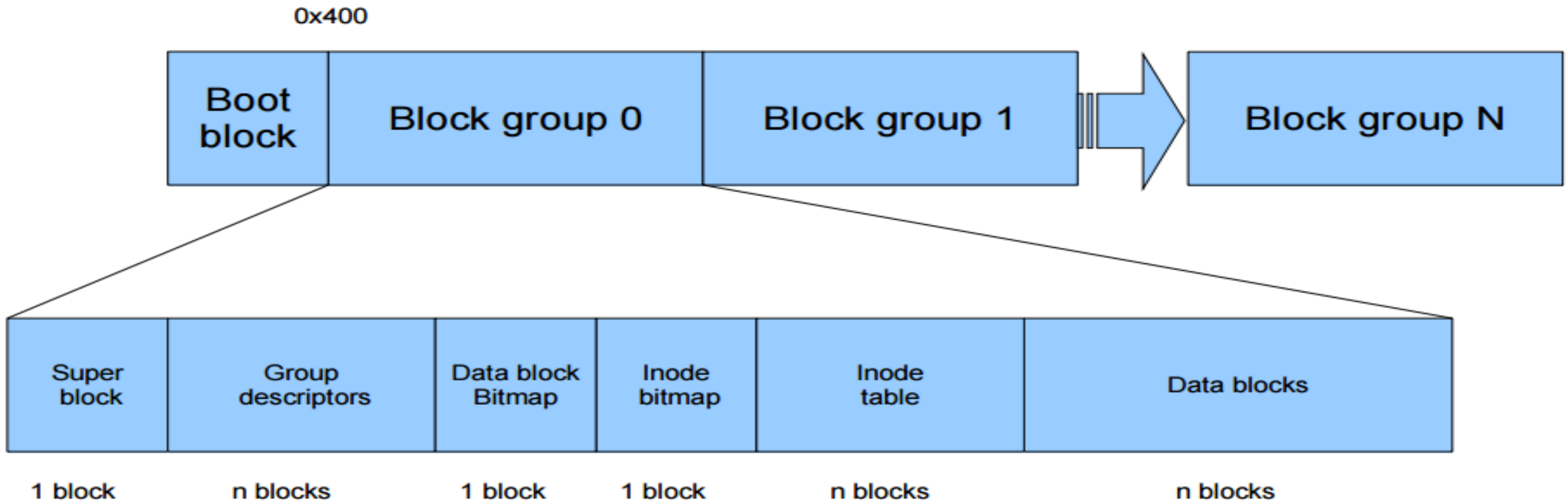
File System Workloads Drive Designs

- Motivation: Workloads influence design of file system
- File characteristics (measurements of UNIX and NT)
  - Most files are small (about 8KB)
  - Most of the disk is allocated to large files
    - (90% of data is in 10% of files)
- Access patterns
  - Sequential: Data in file is read/written in order
    - Most common access pattern
  - Random (direct): Access block without referencing predecessors
    - Difficult to optimize
  - Access files in same directory together
    - Spatial locality
  - Access meta-data when access file
    - Need meta-data to find data

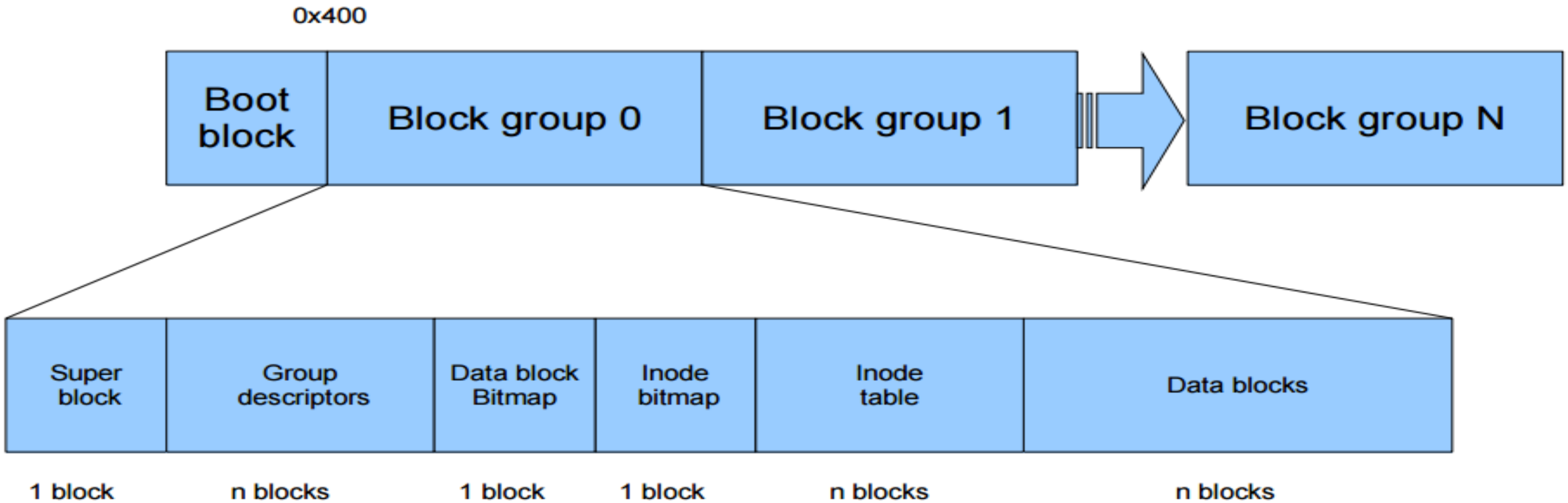
Data Structures for A Typical File System



Disk Layout for a Typical FS



- Boot block: contains info to boot OS
- Block group: next lecture
- Superblock defines a file system
  - type and size of file system
  - size of the file descriptor area
  - free list pointer, or pointer to bitmap
  - location of the file descriptor of the root directory
  - other meta-data such as permission



- What if the superblock is corrupted?
  - What can we do?
  - For reliability, replicate the superblock (in each block group)
- An inode for each file (a header that points to root of the file data blocks) => Inode Table
- Blocks numbered in cylinder-major order, why? (called LBA)
- Data structures to represent free space on disk for both inode and data blocks
  - Bit map: 1 bit per block (sector)
    - How much space does a bit map need for a 4GB disk?
  - Linked list
  - Others?

Data Allocation Problem

- Definition: allocation data blocks (on disk) when a file is created or grows, and free them when a file is removed or shrinks
- Does this sound familiar?
- Shall we approach it like segmentation or paging?
- What kind of locality matters?
  - Compared to page replacement (on demand paging)?

Disk allocation problem

- Two tasks:
  - How to allocate blocks for a file?
  - How to design inode to keep track of blocks?

[lec 20] Disk Bandwidth: Sequential vs Random

- Disk is bandwidth-inefficient for page-sized transfers
  - Sequential vs random accesses

- **Random accesses:**
  - Need seeks, slow (one random disk access latency ~10ms)
  - Randomly reading 4KB pages: ~400KB/sec
- **Sequential accesses:**
  - Stream data from disk (no seeks)
  - 128 sectors/track, 512 B/sector, 6000 RPM
    - 64KB per rotation, 100 rotation/per sec
    - 6400KB/sec → 6.4MB/sec

- Sequential access is ~10x or more bandwidth than random
  - Still no where near the 1GB/sec to 10GB/sec of memory

Disk vs. Memory

Memory

- Latency in 100’s of processor cycles
- Transfer rate ~ 1000 MB/s (DDR SDRAM)
- Contiguous allocation gains ~10x
  - Cache hits
  - RAS/CAS (DRAM)

Disk

- Latency in milliseconds
  - 1ms = 10^6 cycles on 1Ghz machine
- Transfer rate in 30KB/s -- 30MB/s
- Contiguous allocation gains ~1000x

Challenge to disk allocation problem: File Usage Patterns

- How do users access files?
  - Sequential: bytes read in order
  - Random: read/write element out of middle of arrays
  - Whole file or partial file
- How are files used (determines metadata design)?
  - Most files are small
  - Large files use up most of the disk space
  - Large files account for most of the bytes transferred

- Bad news
  - Want everything to be efficient

Hints

- OS allocates LBAs (logical block addresses) to meta-data, file data, and directory data
  - Workload items accessed together should be close in LBA space
- Implications
  - Large files should be allocated sequentially
  - Files in same directory should be allocated near each other
  - Data should be allocated near its meta-data
- Meta-Data: Where is it stored on disk?
  - Embedded within each directory entry
  - In data structure separate from directory entry
    - Directory entry points to meta-data

Design goal and expectation

- Optimize I/O performance
  - What can we do for random access patterns?
  - What can we do for sequential access patterns?
- Also want to minimize file metadata size
  - Metadata for file info and for data block mapping
  - Ideally fit in inode

Allocation Strategies

- Progression of different approaches
  - Contiguous
  - Extent-based
  - Linked
  - File-allocation Tables
  - Indexed
  - Multi-level Indexed
- Questions
  - Amount of fragmentation (internal and external)?
  - Ability to grow file over time?
  - Seek cost for sequential accesses?
  - Speed to find data blocks for random accesses?
  - Wasted space for pointers to data blocks?

Contiguous Allocation

- Allocate each file to contiguous blocks on disk
  - Meta-data: Starting block and size of file
  - OS allocates by finding sufficient free space
    - Must predict future size of file; Should space be reserved?
  - Example: IBM OS/360

		A	A	A		B	B	B	B	C	C	C		

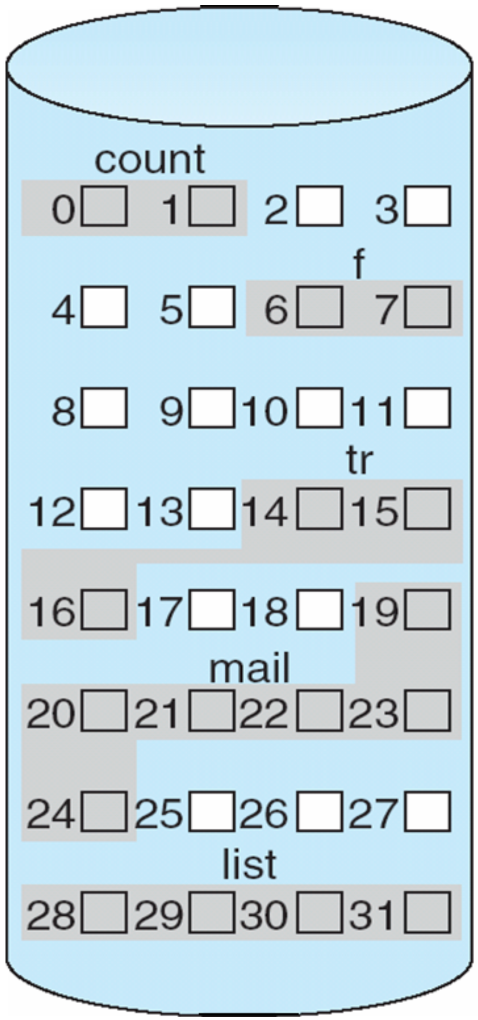
Advantages

- Little overhead for meta-data
- Excellent performance for sequential accesses
- Simple to calculate random addresses

Drawbacks

- Horrible external fragmentation (Requires periodic compaction)
- May not be able to grow file without moving it

Contiguous Allocation of Disk Space



directory

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

Analogy in memory management?

Extent-Based Allocation

- Allocate multiple contiguous regions (extents) per file
  - Meta-data: Small array (2-6) designating each extent
    - Each entry: starting block and size

D	D	A	A	A	D	B	B	B	B	C	C	C	B	B

Improves contiguous allocation

- File can grow over time (until run out of extents)
- Helps with external fragmentation

Advantages

- Limited overhead for meta-data
- Very good performance for sequential accesses
- Simple to calculate random addresses

Disadvantages (Small number of extents):

- External fragmentation can still be a problem
- Not able to grow file when run out of extents

Remzi and Andrea Arpaci-Dusseau

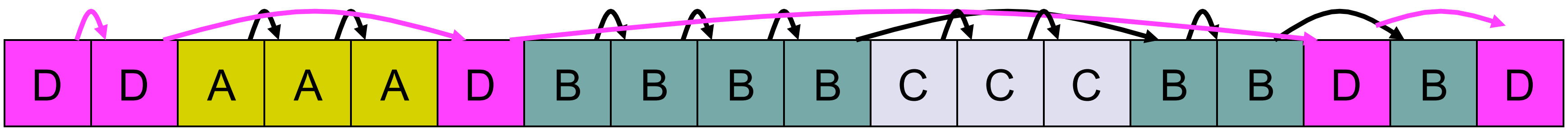
- Systems/Storage
- File and storage systems - reliability
- Wisconsin

Kai Li

- Systems/Storage
- DSM
- Data deduplication
- Founded Data Domain (acquired by EMC with $2.4B, EMC now acquired by Dell with $67B)
- Princeton

Linked Allocation

- Allocate linked-list of fixed-sized blocks
  - Meta-data: Location of first block of file
    - Each block also contains pointer to next block



Advantages

- No external fragmentation
- Files can be easily grown, with no limit

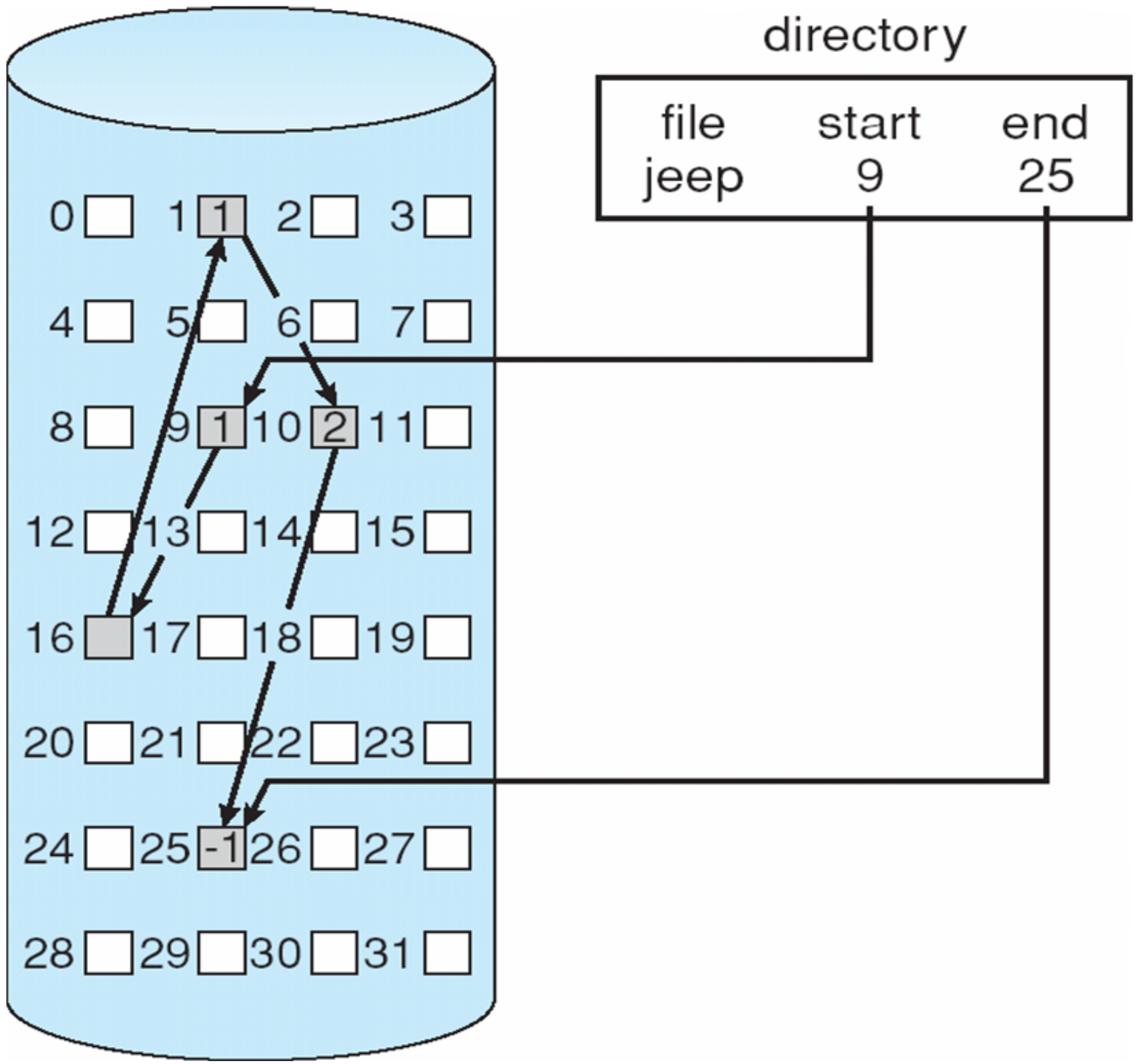
Disadvantages

- Cannot calculate random addresses w/o reading previous blocks
- Sequential bandwidth may not be good
  - Try to allocate blocks of file contiguously for best performance
- unreliable: losing a block means losing the rest

Trade-off: Block size (doesn't need to be the same as sector size)

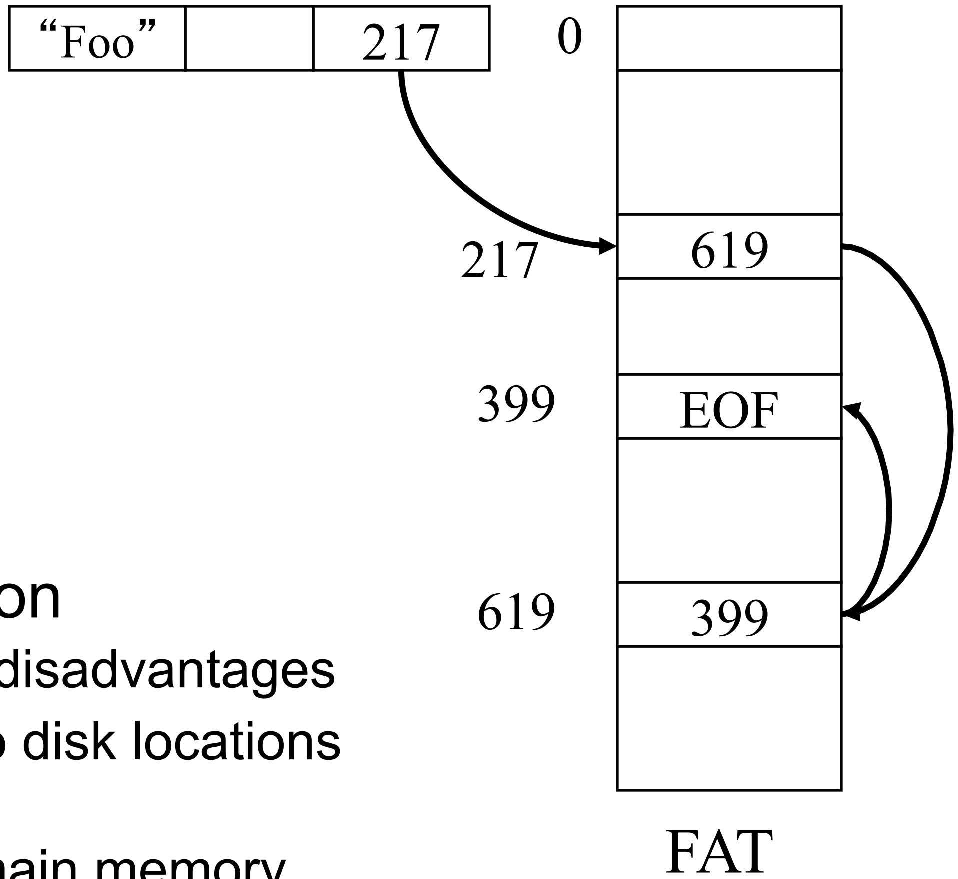
- Larger --> ??
- Smaller --> ??

Linked Allocation



A variation of linked files: File Allocation Table (FAT) (MS-DOS, OS2)

- Variation of Linked allocation
  - Keep linked-list information for all files in on-disk FAT table
  - Meta-data: Location of first block of file
    - And, FAT table itself

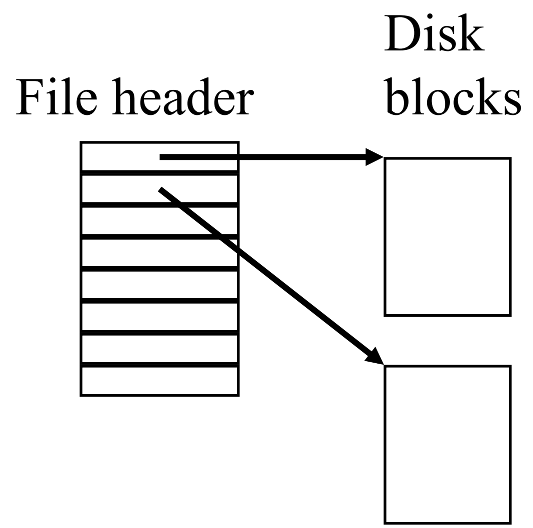


Comparison to Linked Allocation

- Same basic advantages and disadvantages
- Disadvantage: Read from two disk locations for every data read
- Optimization: Cache FAT in main memory
  - Advantage: Greatly improves random accesses

Indexed Allocation

- Allocate fixed-sized blocks for each file
  - Meta-data: Fixed-sized array of block pointers
    - Allocate space for ptrs at file creation time



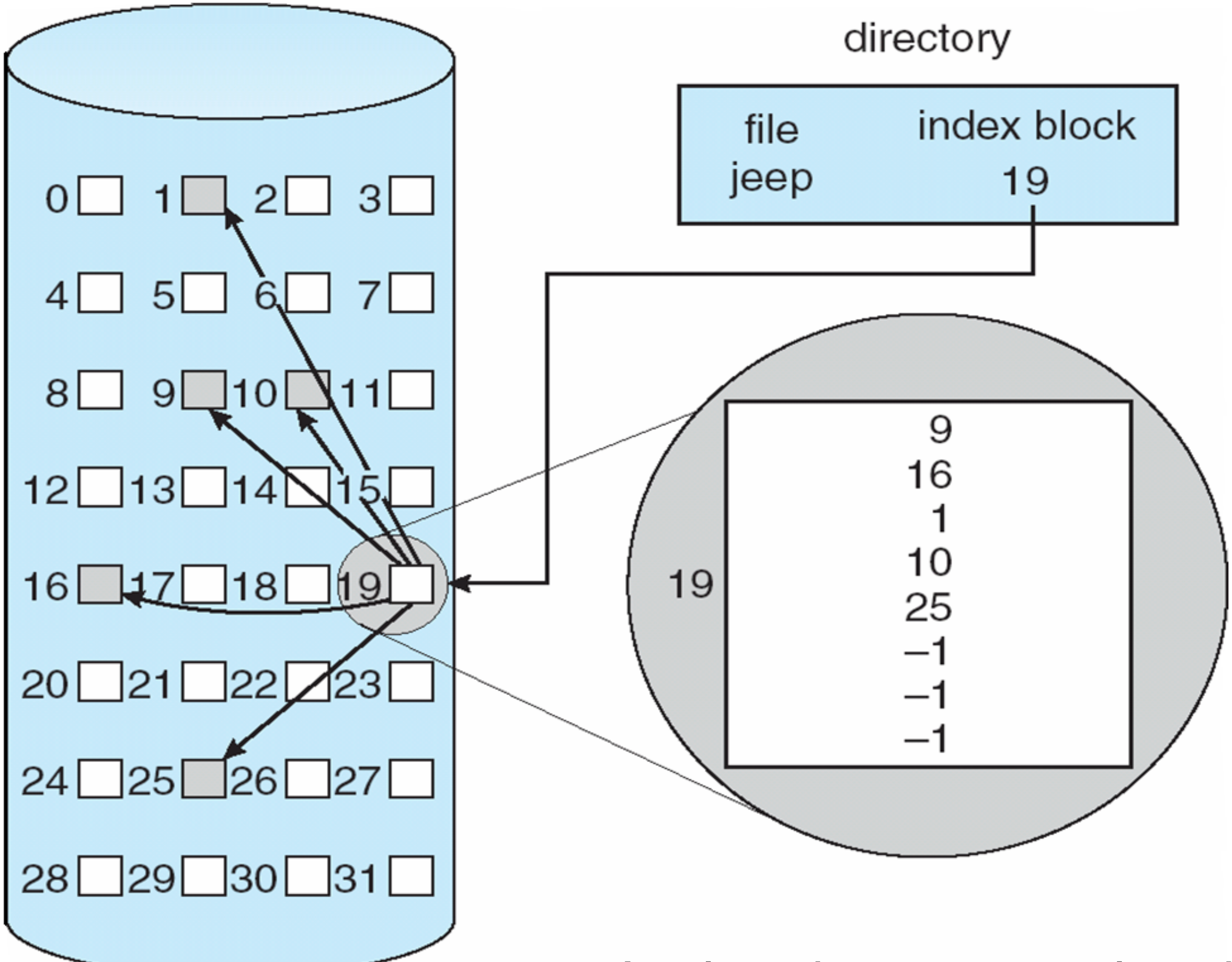
Advantages

- No external fragmentation
- Files can be easily grown, with no limit
- Supports random access

Disadvantages

- Large overhead for meta-data:
  - Wastes space for unneeded pointers (most files are small!)

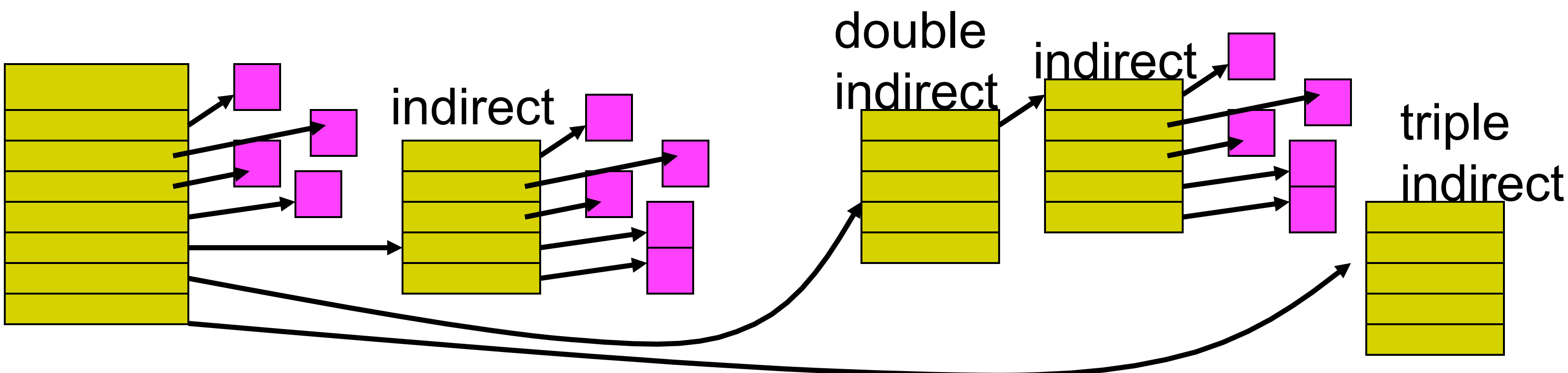
Example of Indexed Allocation



Analogy in memory relocation?

Multi-Level Indexed Files

- Variation of Indexed Allocation
  - Dynamically allocate hierarchy of pointers to blocks as needed
  - Meta-data: Small number of pointers allocated statically
    - Additional pointers to blocks of pointers
  - Examples: UNIX FFS-based file systems

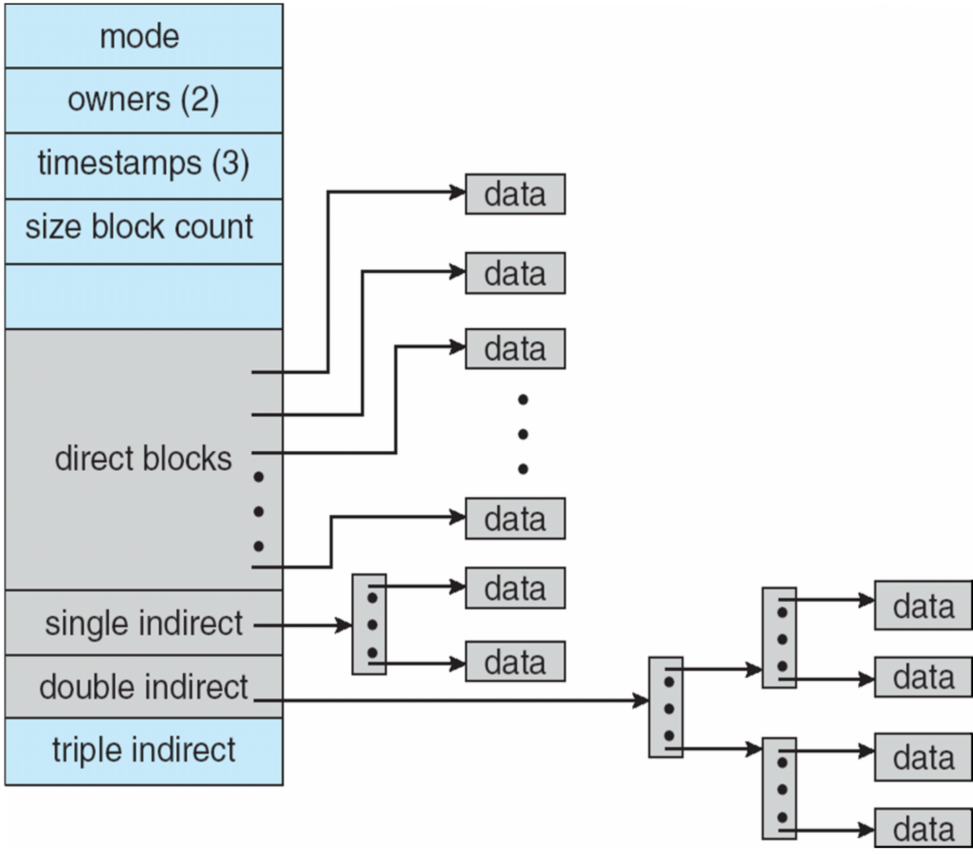


Comparison to Indexed Allocation

- Advantage: Does not waste space for unneeded pointers
  - Still fast access for small files
  - Can grow to what size??
- Disadvantage: Need to read indirect blocks of pointers to calculate addresses (extra disk read)
  - Keep indirect blocks cached in main memory

Example of Multi-Level Index: Linux ext2 (and ext3)

- ext2 has 15 pointers.
  - Pointers 1 to 12 point to direct blocks
  - pointer 13 points to an indirect block
  - pointer 14 points to a double indirect block
  - pointer 15 points to a triple indirect block



Theoretical ext2 limits under Linux

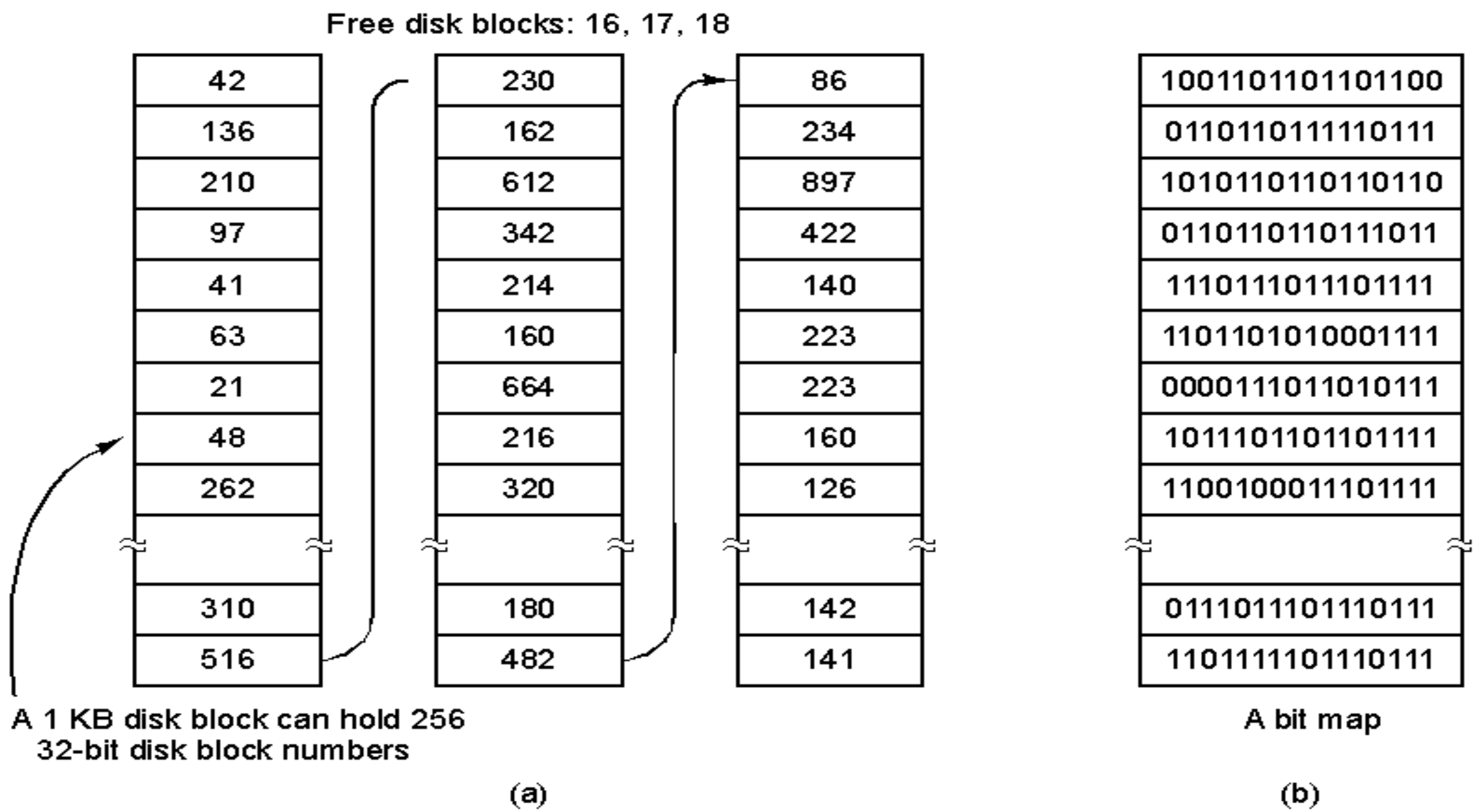
Block size:	1 KB	2 KB	4 KB	8 KB
max. file size:	16 GB	128 GB	1 TB	8 TB
max. filesystem size:	4 TB	8 TB	16 TB	32 TB

Deep thinking

- What about sequential access in multi-level indexed scheme?
- Can we try multi-level indexing in page table design?

Managing Free Disk Space

- 2 approaches to keep track of free disk blocks
  - Linked list and bitmap approach



Free-Space Management Tradeoffs

- Bit Map:
  - Pro: Easy to get contiguous files
  - Con: Bit map requires extra space
    - Example:

      block size = 2^{12} bytes

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

      $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Linked list:
  - Pro: no wasted extra space
    - Use free blocks themselves to store free block list
  - Con: Cannot get contiguous space easily

Summary

- Seeks kill performance → exploit spatial locality

- Extent-based allocation optimizes sequential access
- Single-level indexed allocation has speed
- Multi-level index has great flexibility

- Bitmaps show contiguous free space
- Linked lists easy to search for free blocks

Analogy

Memory Management

- Virtual address per process
- Page table maps virtual pages to physical pages
- Different schemes:
  - Base& bound
  - Segmentation
  - 1-level paging
  - 2-level paging
  - Segment+paging
  - Inverse paging

Disk Allocation

- 1-D logical bytes per file
- File header maps logical bytes to disk blocks
- Different schemes:
  - Contiguous
  - Extent-based
  - Linked files / FAT
  - Single-level indexing
  - Multi-level indexing