

# csci 3411: Operating Systems

## **File Systems**

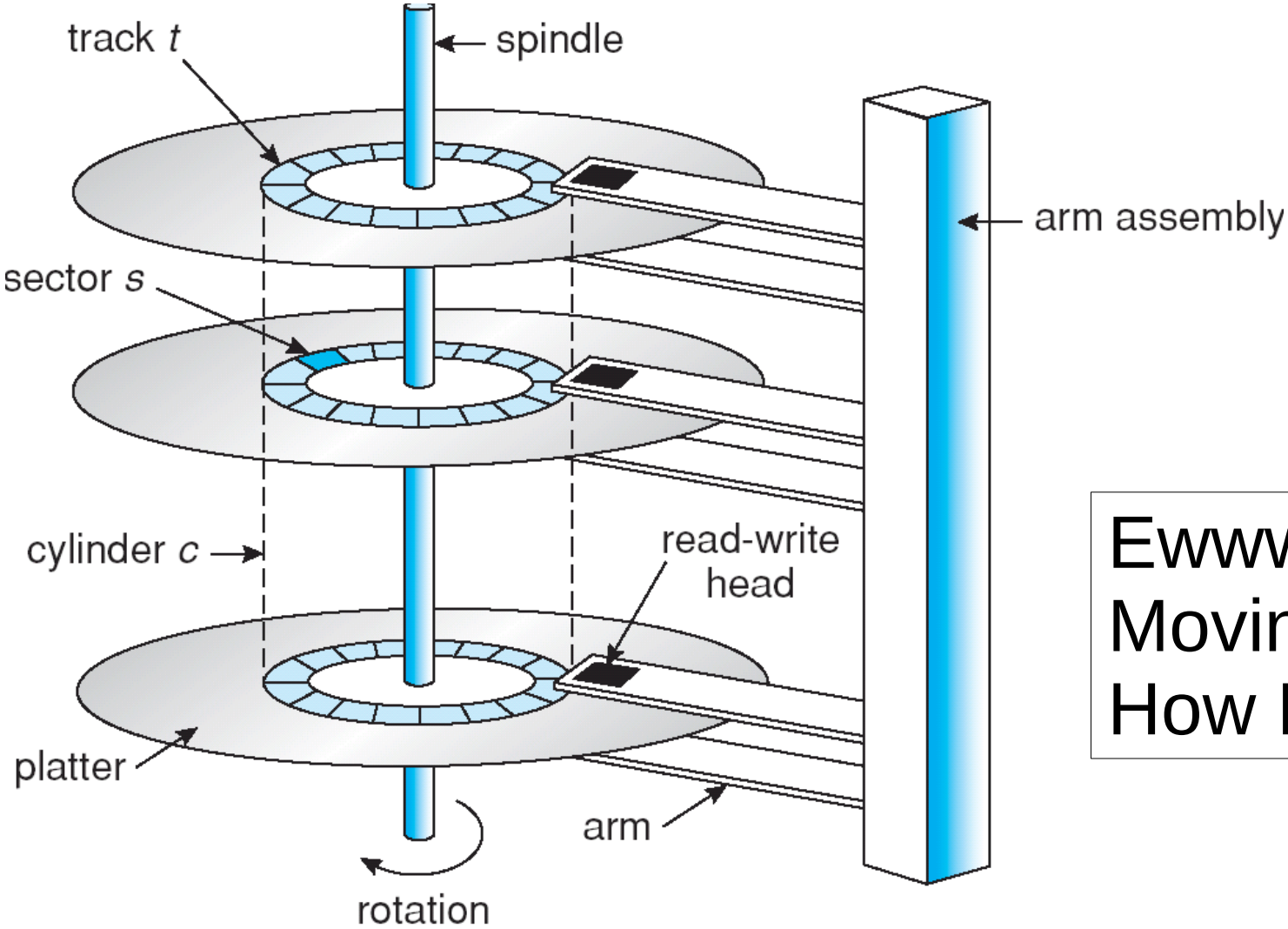
Gabriel Parmer

Slides evolved from Silberschatz

# Today: File System Implementation

- We discussed
  - *abstractions* for organizing persistent storage
  - *interfaces* for programming the storage
- Today: How do we make these abstractions and interfaces efficient
  - Best utilize the capacity of the persistent storage
    - Throughput (MB/sec to/from the storage)
    - Latency to retrieve/store data
    - Space utilization: minimize fragmentation in storage

# Magnetic Media (Disk)



Ewww, Yuck!  
Moving Parts!  
How Barbaric.

# Magnetic Media II

- *Blocks* – granularity of I/O as seen by OS
  - Multiple of sector size
    - (block size = page size) is convenient (i.e. 4KB)
- Accessed by OS as 1d array of blocks
- Disks can process I/O requests in *parallel*
  - Many I/O requests for blocks can be pending at a given time

# Magnetic Media III

- Disks are slow!
  - ~10ms latency to access a block<sup>1</sup>
    - Versus 0.0002ms to access to memory
    - Versus 0.000001ms to access to register
- Minimize frequency of disk access!

<sup>1</sup> <http://www.anandtech.com/printarticle.aspx?i=3403>

# Caching!

- Paging (page cache) treats disk as extension of the storage hierarchy
  - When we didn't have enough memory, write out to disk
- *Buffer cache* holds file's data, treats memory as a cache for disk data
  - Minimize frequency of slow disk accesses, so cache file data in memory
  - When read/writes are made to files, cache the file's data
    - Further accesses to that file will check if it is in cache
    - Memory accesses instead of disk accesses

# Disks: Not just slow, *Complex*

- Seeks between cylinders are *expensive*
  - Random vs. sequential access
- Sequential access<sup>1,2</sup> – platter rotation
  - 80MB/s bandwidth
  - 0.16ms latency
- Random access<sup>2</sup> – head seeks
  - 0.5MB/s bandwidth
  - 10ms latency
  - All sorts of fail

<sup>1</sup> <http://www.anandtech.com/printarticle.aspx?i=3403>

<sup>2</sup> <http://it.anandtech.com/printarticle.aspx?i=3532>

# Major Filesystem Questions

- How can we track
  - Where different files are on disk
  - Where the directory structure is stored
- Where free space is on disk to satisfy allocations
  - extending/creating files/directories
- How can we do this while compensating for the physical characteristics of disks?

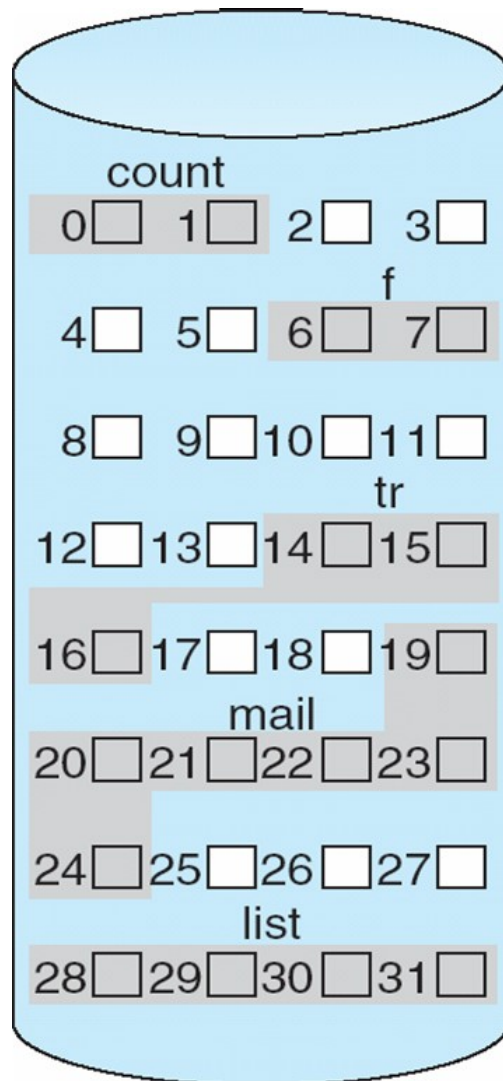
# Files → Disk Blocks

- File – Logical storage unit
  - Sequence of bytes from 0 → FILE\_SIZE
- Disk
  - Essentially large array of bits 0 → VERY\_BIG
- Many files on disk
  - Analogy: many processes with virtual memory share single physical memory
- *How do we map from a location in a file, to a location on disk?*
  - Similar to: How do we map from an address in a virtual address space into a physical address?
  - But we have no MMU/hardware support

# Contiguous Allocation

- Directory entry has
  - Each file described by a <start, length> pair
    - Where is the location of the 100<sup>th</sup> byte of a file?
    - Similar to segmentation

# Contiguous Allocation II



directory

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

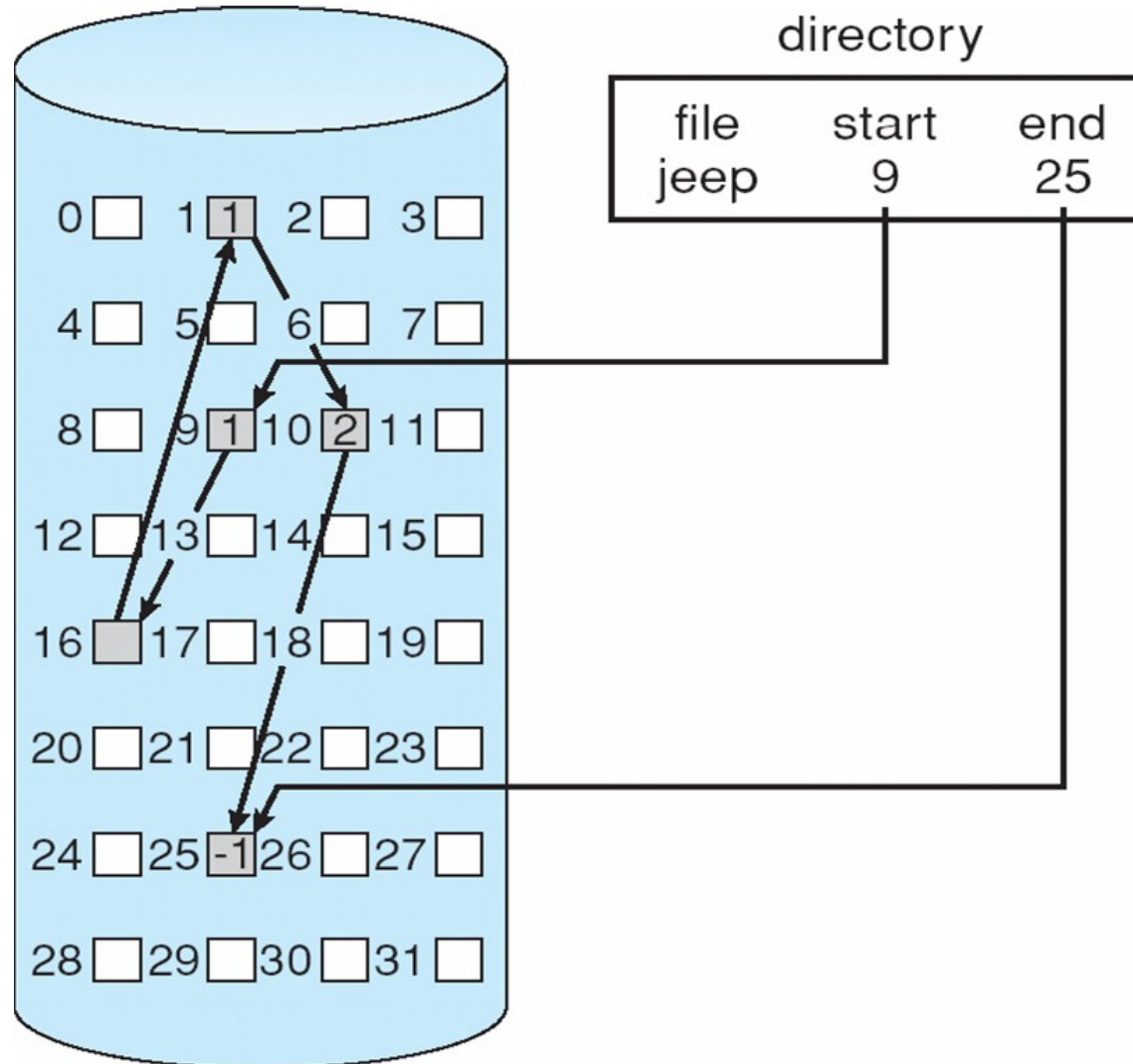
# Contiguous Benefits/Problems?

- File Access: Random/Sequential access?
- Disk Access: Seeks vs. rotation?
- Allocating files? What size?
  - First-fit, best-fit
- Growing files?
  - Didn't have this with memory mgmt!
- Fragmentation?
  - compaction/defragmentation

# Linked Allocation

- File represented as a linked list of blocks
- Each block holds both
  - Data
  - Block number of next block in file
- Directory contains
  - A file's <start, end> blocks

# Linked Allocation II

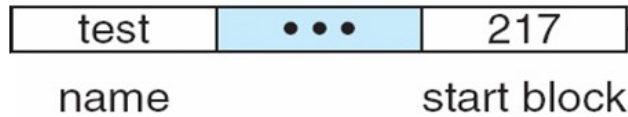


# Linked Benefits/Problems?

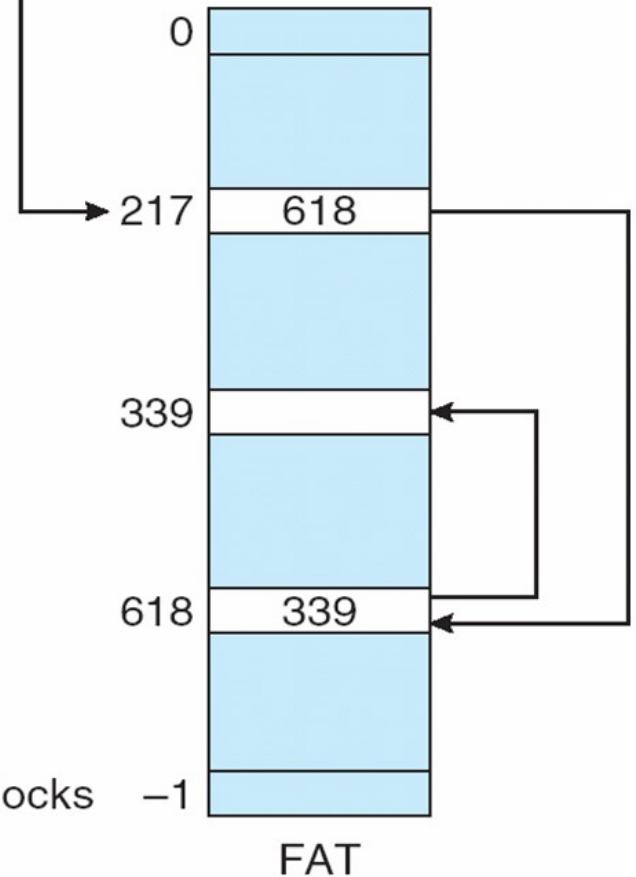
- Space usage for links?
- Fragmentation?
- Random/Sequential Access?
- Seeks vs. Rotation?
- Block allocation?

# Not so FAT

directory entry



- File Allocation Table (FAT)
  - Special section of disk
    - Has one entry per block
    - Entry maps trivially to data block on disk
  - Linked list maintained between these entries
    - Instead of linked list *in* data blocks
- Rest of disk contains data-blocks



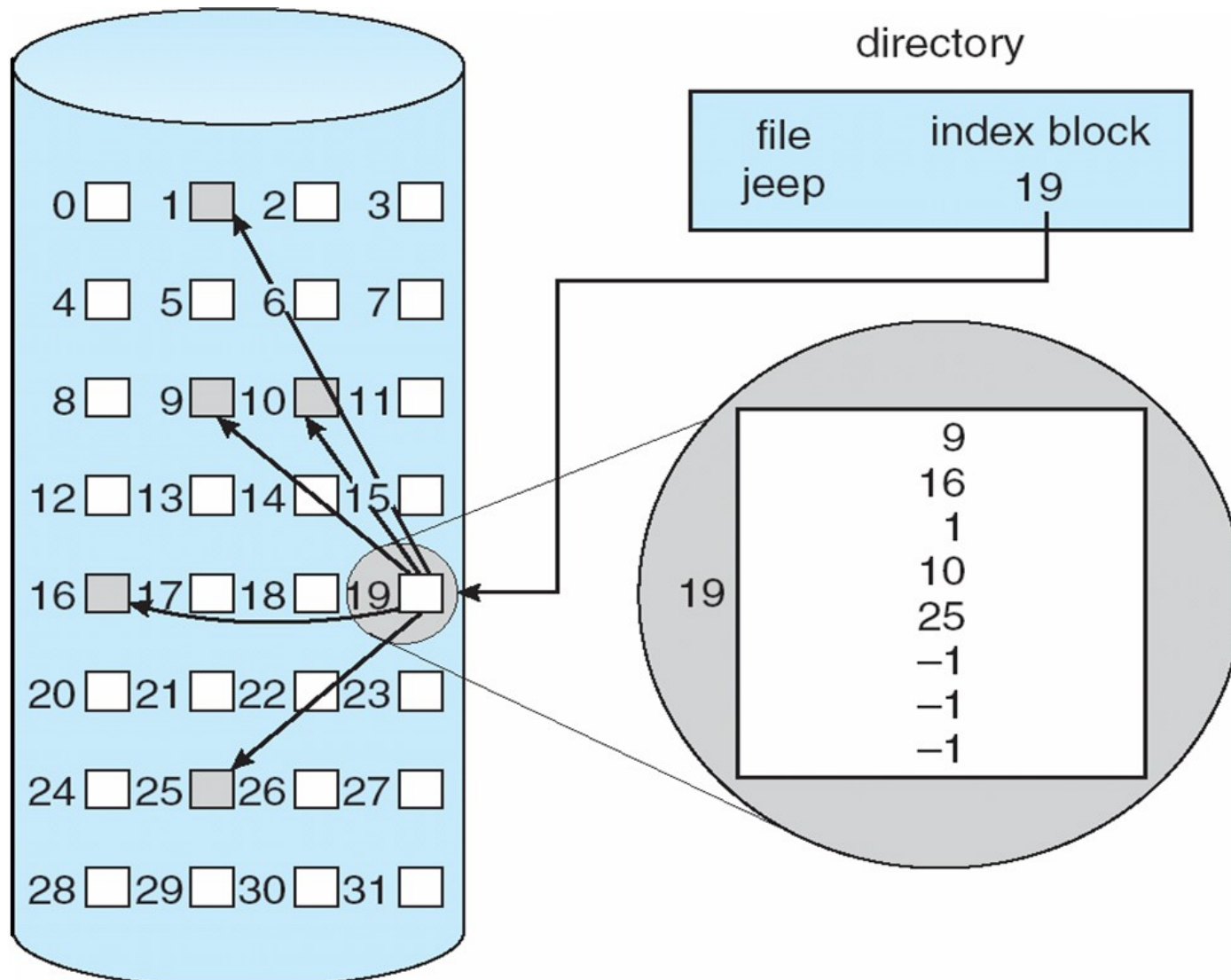
# FAT Benefits/Problems?

- Space usage?
- Number of disk accesses to read
  - One data block?
  - Two data blocks?
- Sequential/Random Access?
- Seeks vs. Rotation?

# Indexed Allocation

- Directory includes block location of *index*
- *Index* is an array entries containing the locations of the blocks of that file
  - Like a single-level page-table

# Indexed Allocation II



# Indexed Allocation III

- Block size = 100B, index holds 10 entries
  - File can be  $100 * 10 = 1000\text{B}$  long
  - Which block contains file offset 365?
- Block size =  $2^{12}\text{B}$ , index holds  $2^8$  entries
  - File can be  $2^{12} * 2^8 = 2^{(12+8)} = 2^{20}$  bytes long
  - To find the block containing the 0xBEEF offset in the file
    - Values larger than  $2^{12} = 0\text{x}B000$
    - Look for entry number  $0\text{x}B = 11$  in the index to find the block
    - Desired data at  $0\text{x}0EEF$  offset into block

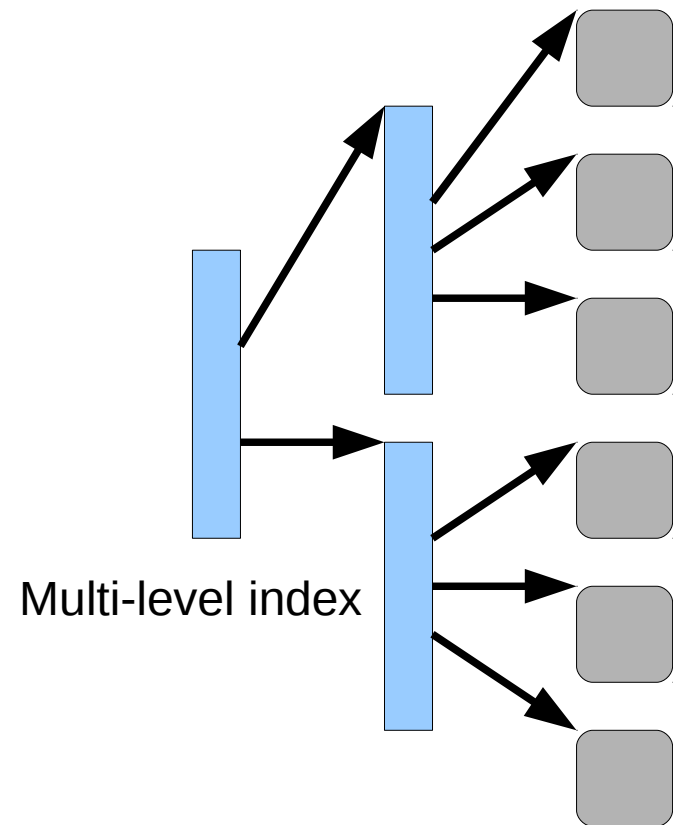
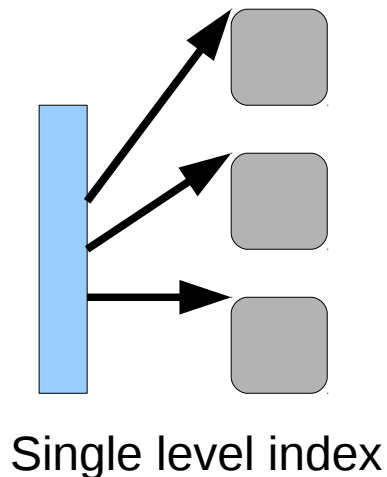
# Indexed Allocation Benefits/Probs?

- File Size?
  - Can support larger file sizes by linking together indices
    - *linked indices*
  - One entry in an index is the location of the *next* index
- Space overhead?
  - Space used – One block index with  $N$  entries
    - file w/ 1 block? File with  $N$  blocks?
- How many disk accesses must we make to retrieve data blocks?
- Seeks vs. Rotation?
- Random Access within file?
- Fragmentation/Block Allocation?

# Multilevel Index

- Like multilevel page-tables
- An index block contains entries of locations of
  - other index blocks which contain entries of locations of

- File data



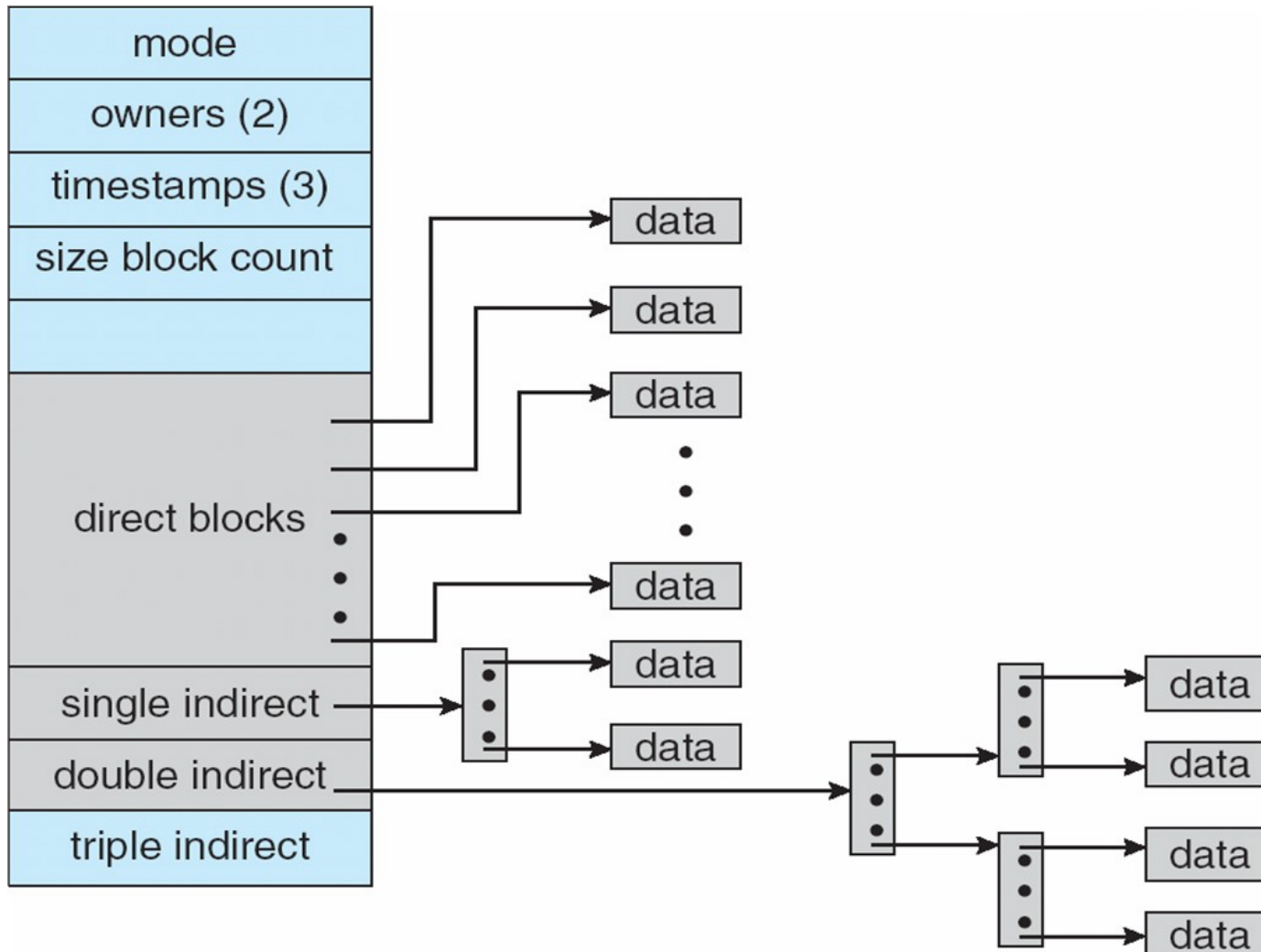
# Multilevel Index Pros/Cons?

- How many disk accesses must we make to retrieve data blocks?
  - Sequential access, random access?
- File size?
  - $M$  entries per index
  - $N$  levels of index nodes
  - How many blocks can be addressed?

# Combined Index Scheme

- Index contains
  - $N$  *direct* references to data blocks
  - $M$  *indirect (or single indirect)* references to second level indices
    - That each refer to data blocks
  - $X$  *double indirect* references to second lvl indexes
    - That refer to third level indexes
    - That refer to data blocks
  - $Y$  *triple indirect* references to second level indexes
    - That refer to third level indexes
    - That refer to fourth level indexes
    - That refer to data blocks

# UFS/ext2/... Indices



# Combined Index Pros/Cons?

- File size (blocks with  $W$  entries, 4K per block)?
  - $N$  *direct* references to data blocks
  - $M$  *single indirect* entries
  - $X$  *double indirect* entries
  - $Y$  *triple indirect* entries
  - $Z$  entries per “multilevel” index
- Random/Sequential Access?
- Space overhead?

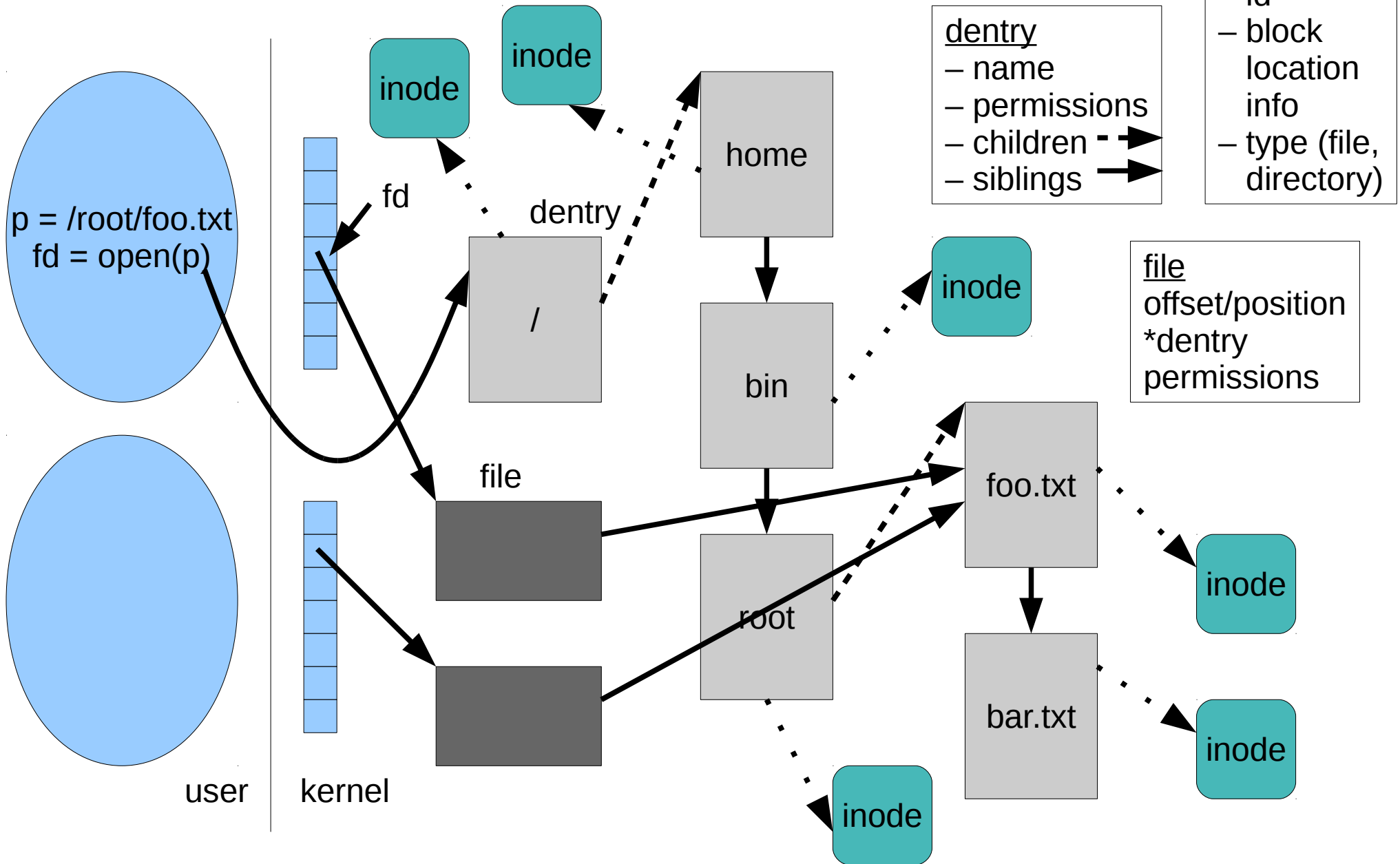
# Extents

- Technique can be added to each of the preceding data-structures
- Rotation vs. seek benefit of contiguous allocation
  - None of the other methods had that benefit!!!
- Don't just refer to blocks (index or data) in index
  - Index entry: <base, length> pair
    - instead of just <base>
  - Can read *length* blocks without seeks!
- Can complicate random access
- Makes index entries larger to store *length*

# inodes

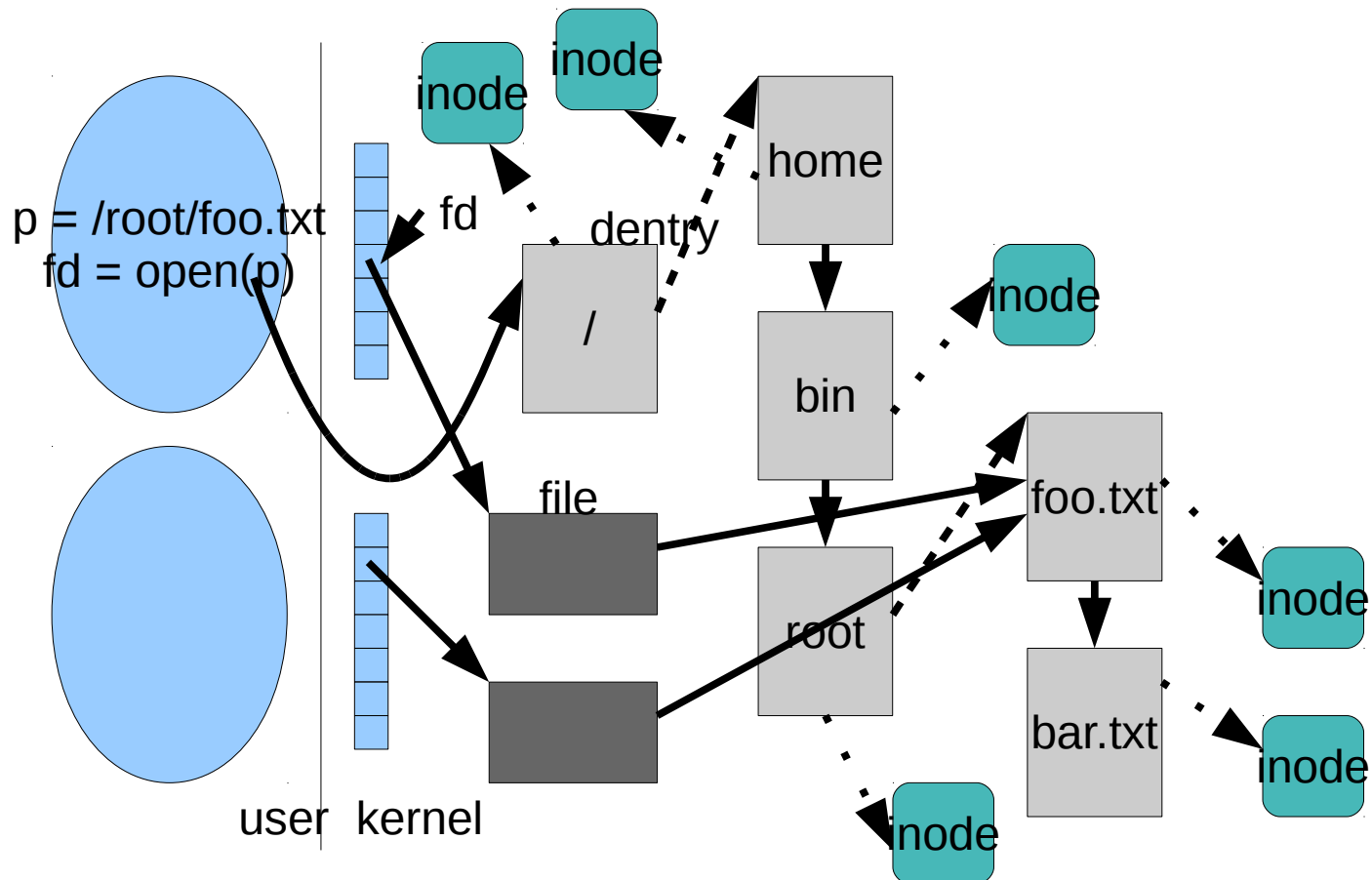
- Both file contents and directory structure are stored using these techniques
  - Directory data blocks include data about subdirectories
- UNIX/Linux in-memory data structure used for tracking the location of on-disk objects
  - Individual inodes represent either files or directories
  - Each inode has a unique identifier
    - `ls -i`
    - Often the location on disk that inode is located at
  - Includes location of that file's/directory's blocks

# Data structures including inodes



# Perspective

- How many disk accesses can it take to open `/root/bar.txt`?
- Disks satisfy about 100 serial accesses per second



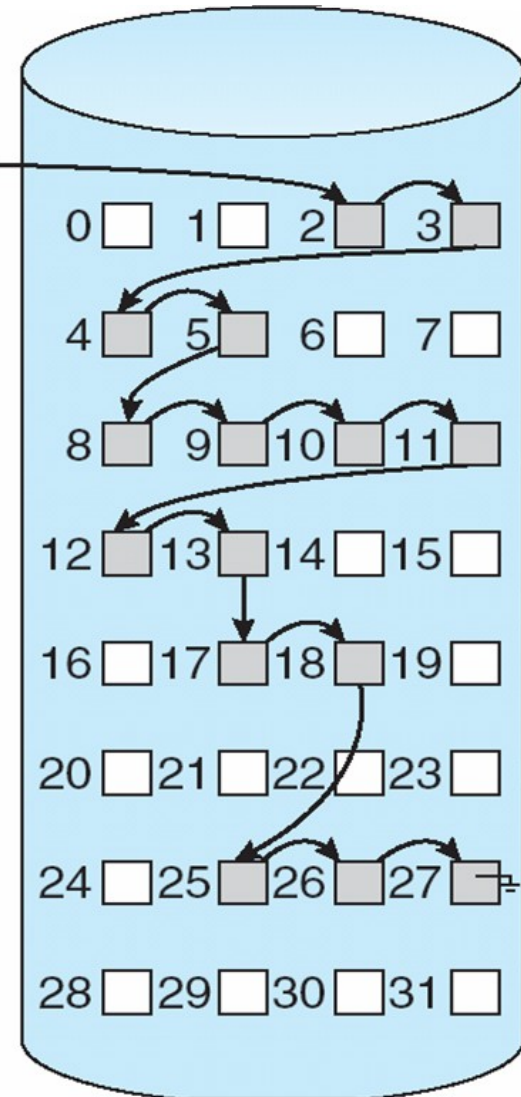
# Free Space Tracking

- A file/directory is created or grows when written to
- How do we find free blocks to allocate to it?

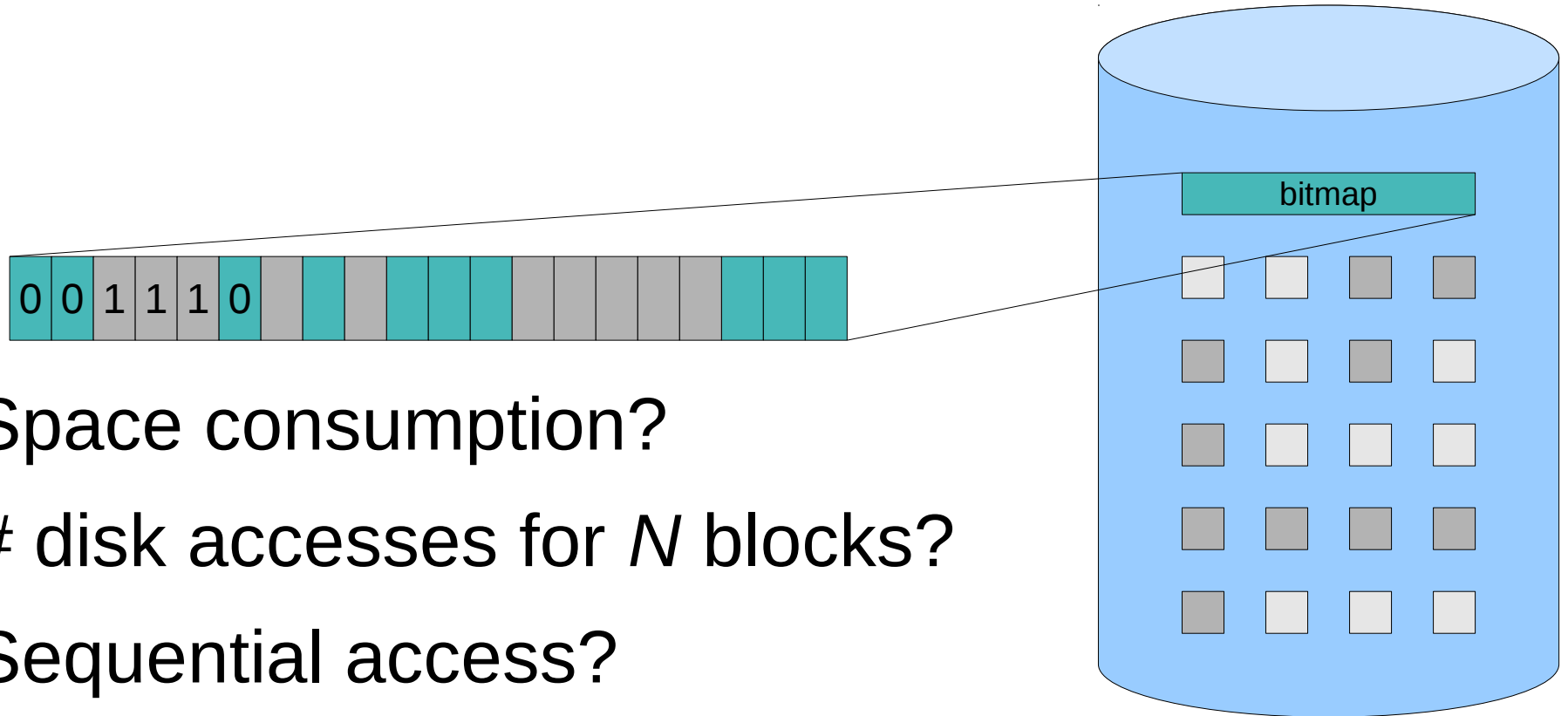
# Free Space Tracking: Linked List

- Space consumption?
- # disk access to find  $N$  blocks?
- Promote sequential access
  - Easy to allocate sequential blocks?
  - Is the freelist always “sorted”?
- Optimization: extents of free blocks

free-space list head



# Free Space Tracking: Bitmaps



- Space consumption?
- # disk accesses for  $N$  blocks?
- Sequential access?
  - Find contiguous blocks easily?