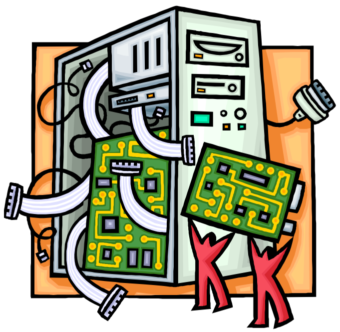
Disks



What we're covering: Bottom-up view

- The device: a disk
- Disk scheduling
- Filesystem structures
- User-level: using a filesystem

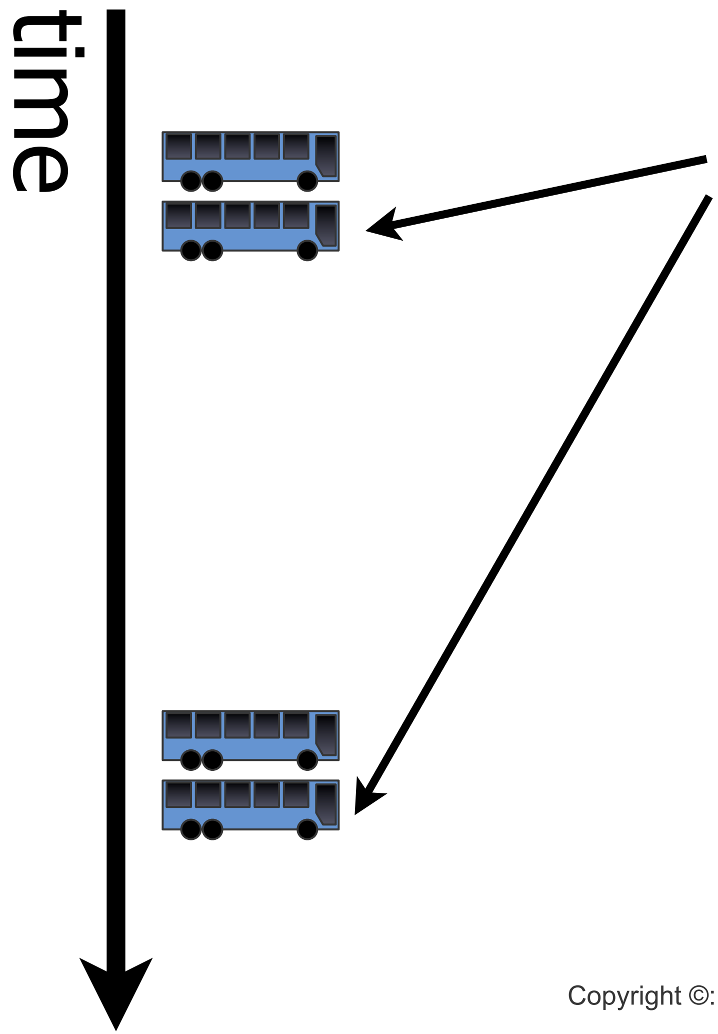
- Today:
  - Finish up disk scheduling
  - Filesystem structures

A detour: waiting for the bus

- Average time between bus arrivals is 10 minutes
- If I arrive at a random time, how long do I expect to wait for the next bus?

- Answer: depends on the pattern of bus arrivals...

A detour: waiting for the bus



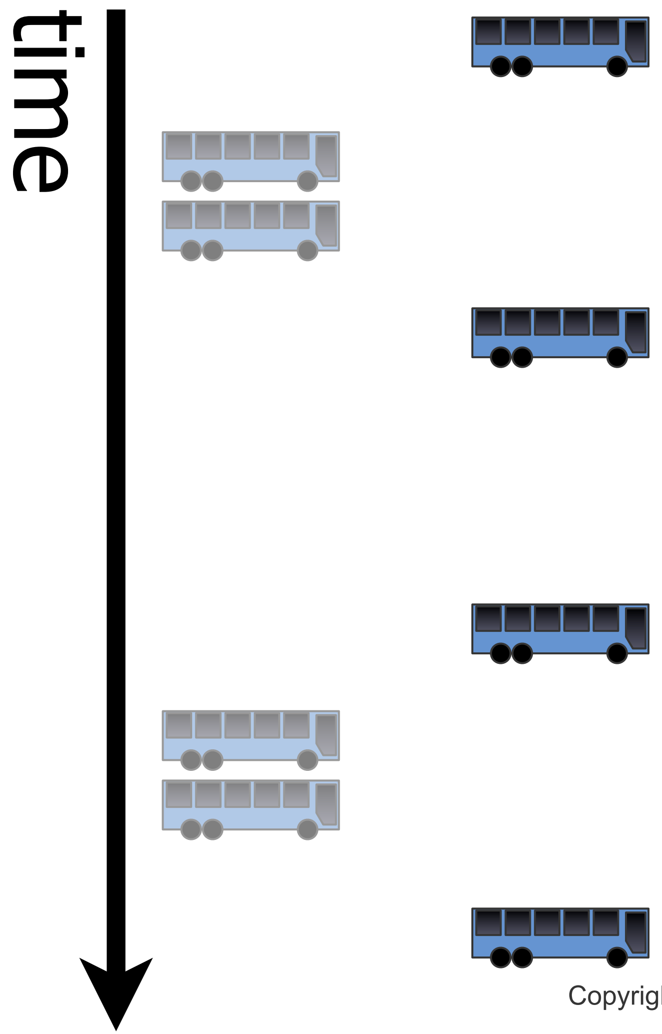
Second bus is essentially useless!
Anyone waiting got on the first bus.

Mean time between buses: 10 min
Time between “bus pairs”: 20 min

On average, we arrive in the middle of one of these intervals.

So mean waiting time: 10 min

A detour: waiting for the bus

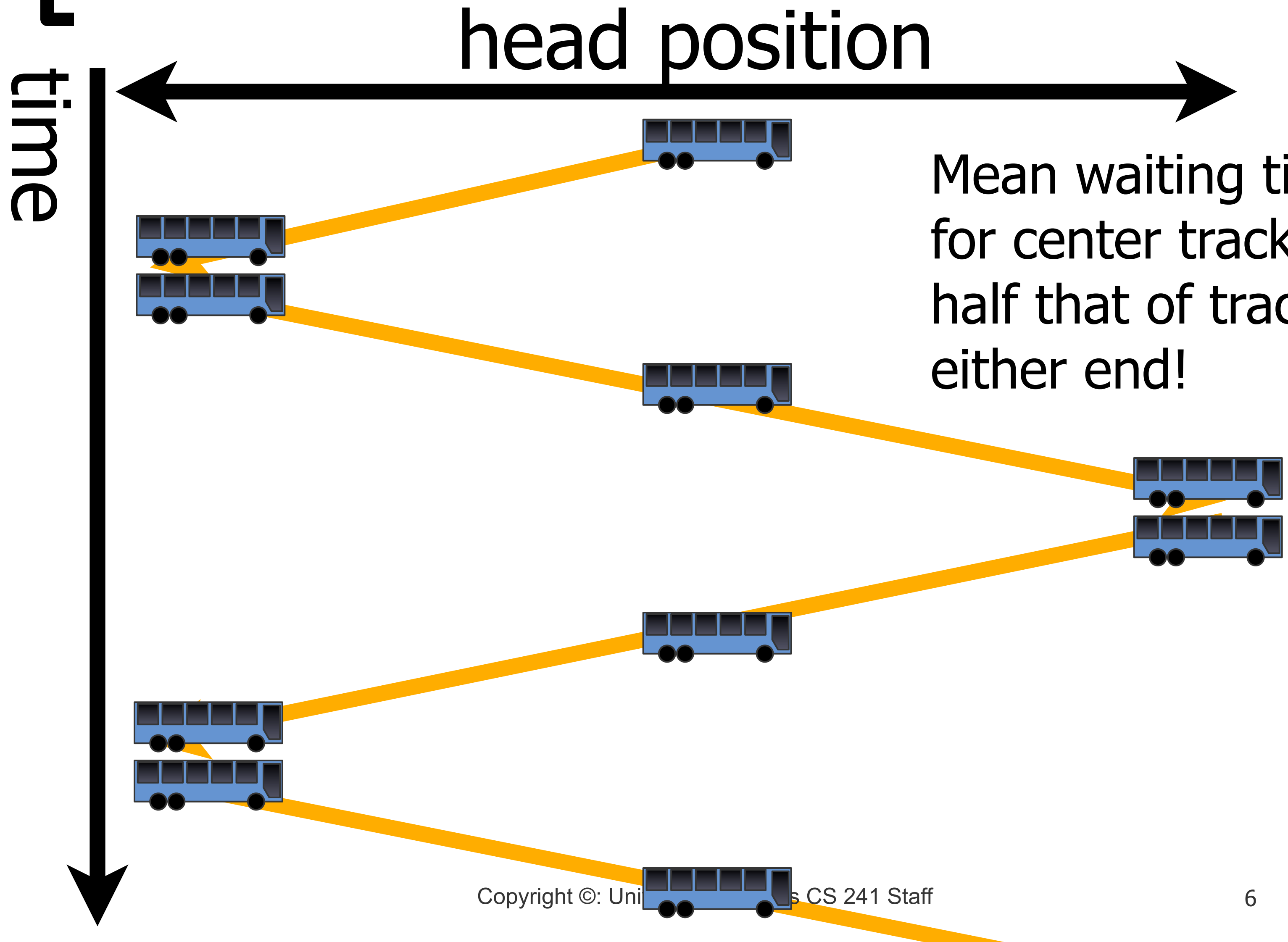


A better arrival pattern: Even spacing between bus arrivals minimizes mean waiting time.

Time between buses: 10 min

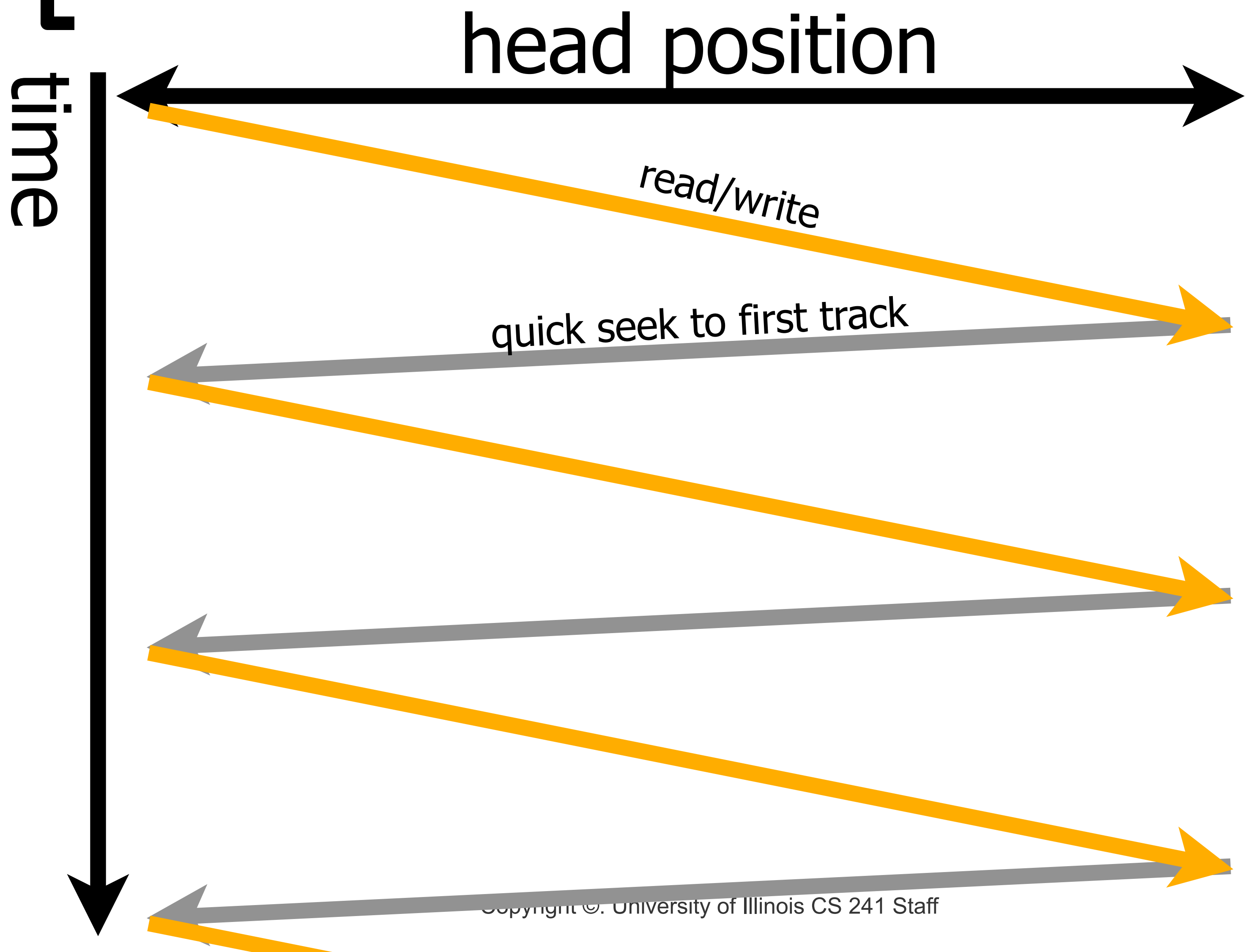
Mean waiting time: 5 min

Back to SCAN disk scheduling



Mean waiting time for center tracks is half that of tracks at either end!

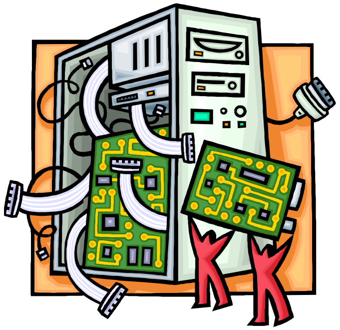
A fix: C-SCAN (circular SCAN)



C-SCAN

- Method
  - Like SCAN
  - But “circle around” to the first track when we get to the last
- Pros
  - Uniform service time
- Cons
  - Do nothing on the return to the first track: some wasted seek time
  - (But it’s faster than if we were reading/writing on the return journey to the first track)

Filesystems



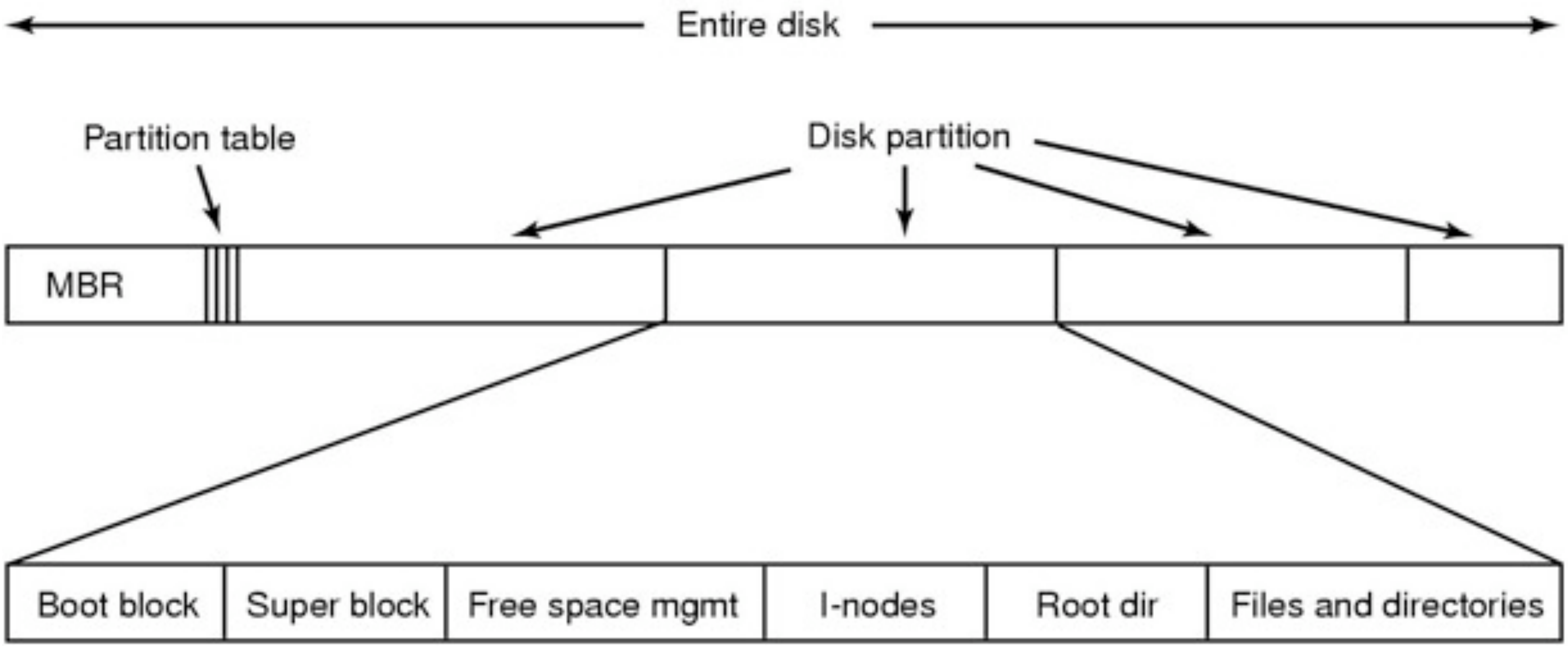
Key terms

- **Sector:** unit of allocation on disk
- **Block:** unit of allocation in filesystem
  - could be several segments
- **Disk address:** index of a block
- **inode:** structure representing a single file or directory, including metadata and pointers to data
  - a directory is like a regular file whose contents happens to be a list of files

Disk Layout

- Master boot record
  - Partition table (start/end of each partition)
  - Active partition
  - BIOS reads MBR and boots (loads OS) from active partition
- Boot record
  - The first block in partition
  - Executable: loads OS
- Followed by file system
  - Superblock
  - Free list
  - File metadata (inodes)
  - Files

File system layout



- A typical file system layout
- MBR = master boot record

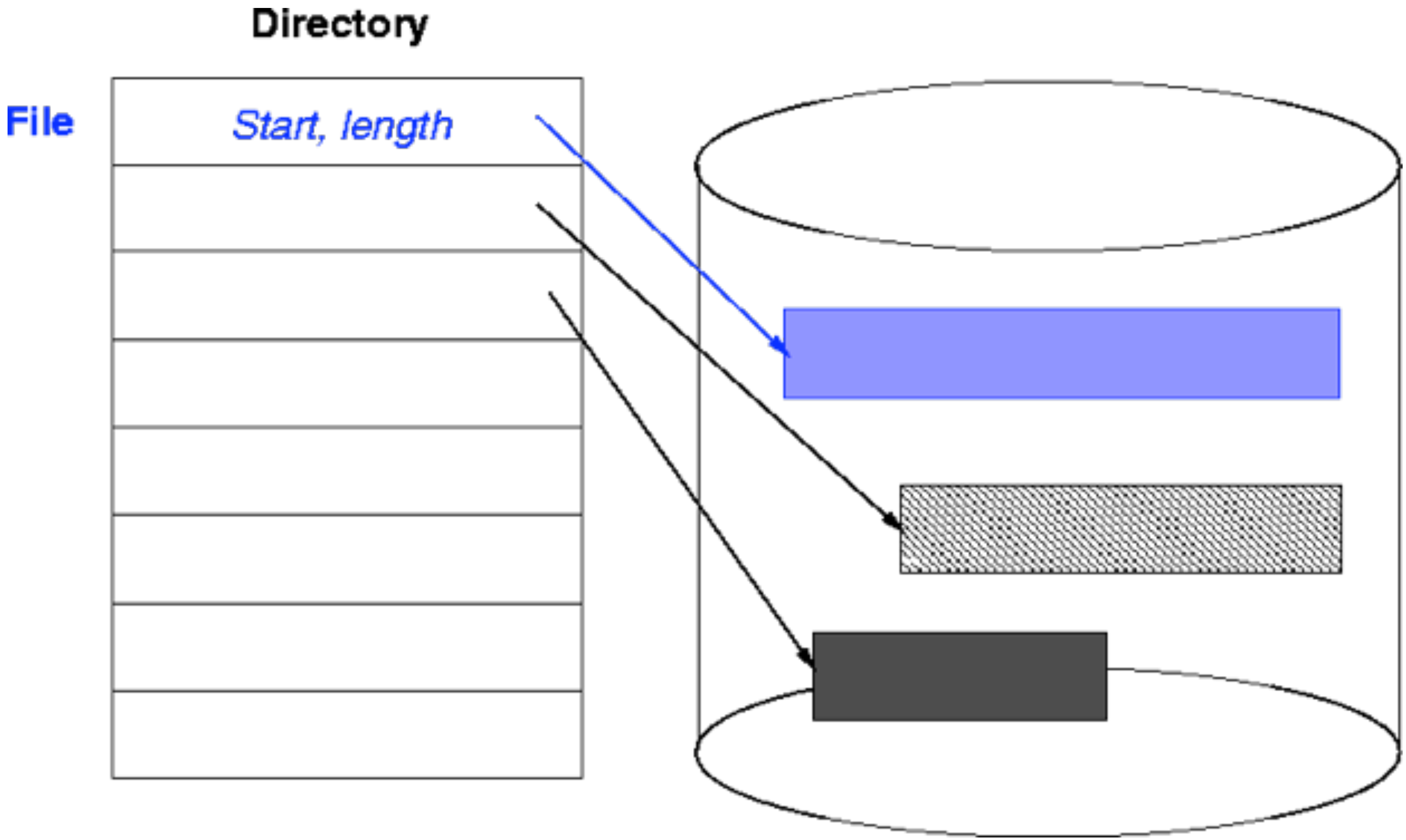
Today's topics

- Allocating blocks for a file on disk
  - Contiguous
  - Linked list
  - Indexed
- Keeping track of free blocks on disk
  - Bitmap
  - Linked list

Allocation of Disk Space

- Low level access methods depend upon the disk allocation scheme used to store file data
  - Contiguous
  - Linked list
  - Indexed

Contiguous Allocation



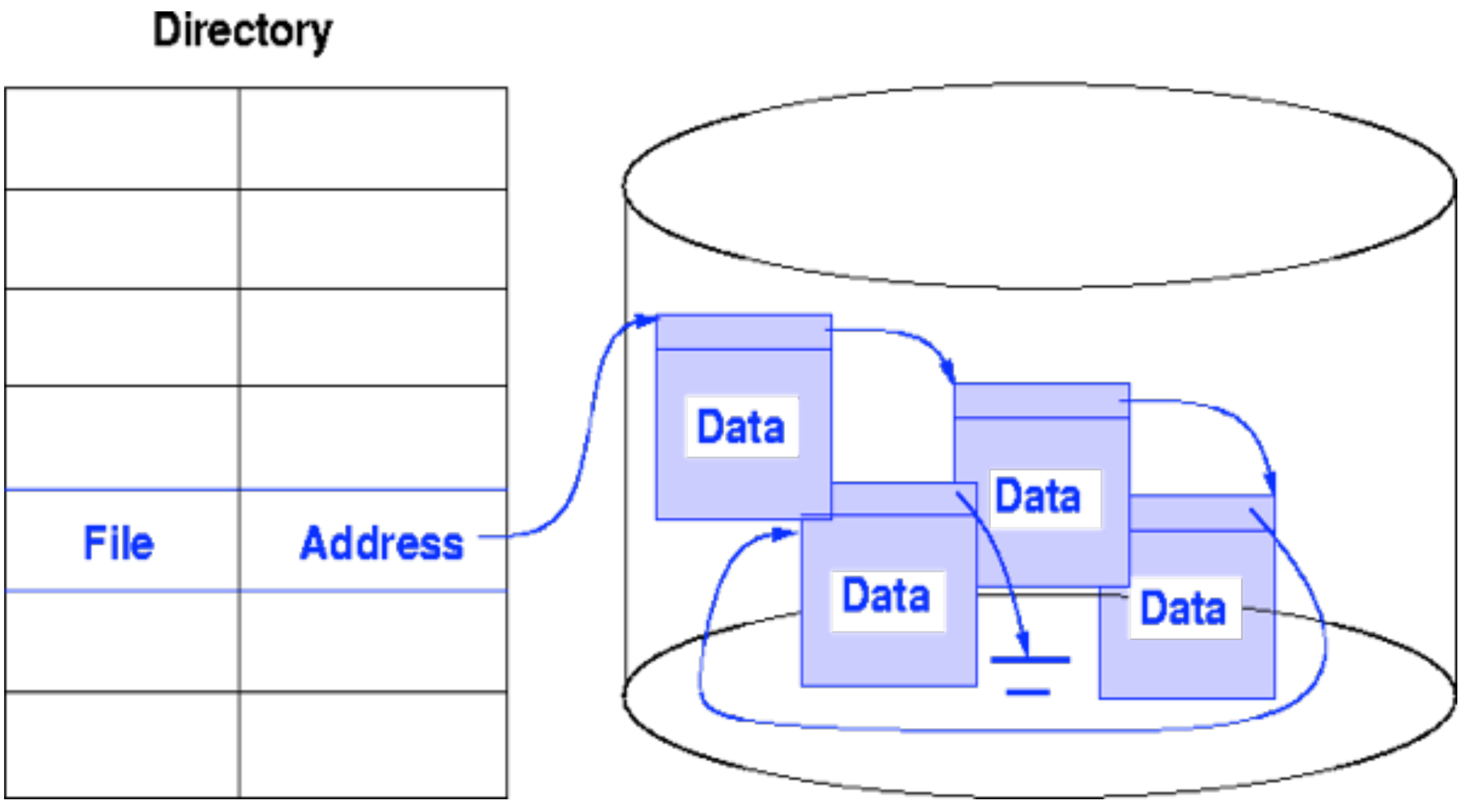
Contiguous Allocation advantages

- Access method suits sequential and direct access
- Easy to recover in event of system crash
- Fast, often requires no head movement and when it does, head only moves one track

Contiguous Allocation problems

- Expanding the file requires copying
- Dynamic storage allocation - first fit, best fit
- **External fragmentation occurs on disk**

Linked Allocation



Linked List Allocation

- Each file is a linked list of chunks
- Pointers in list are not accessible to user
- Directory table maps files into head of list for a file
- A node in the list can be a fixed size physical block or a contiguous collection of blocks
- Easy to use - no estimation of size necessary

Linked List Allocation

- Advantages
  - Can grow in middle and at ends
  - Space efficient, little fragmentation
- Disadvantages
  - Slow for random ("direct") access: need to read through linked list nodes sequentially to find record of interest blocks
  - Suited for sequential access

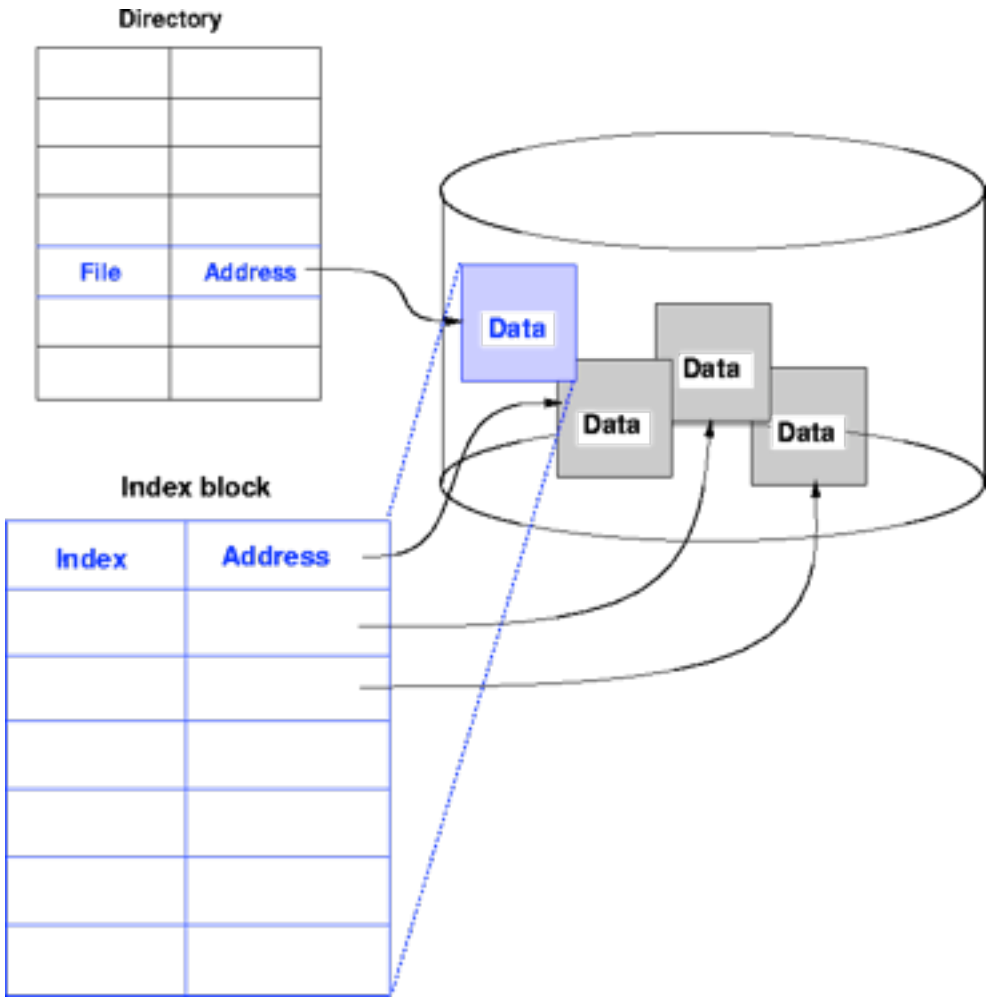
Linked List Allocation Issues

- Disk space must be used to store pointers (if disk block is 512 bytes, and disk address requires 4 bytes, then the user sees blocks of 508 bytes)
- Not very reliable. System crashes can scramble files being updated
- Important variation on linked allocation method: `file-allocation table' (FAT) - OS/2 and MS-DOS
  - Pull all linked list pointers to one part of the disk
  - Faster than reading one block at a time to scan through file

Linked List Allocation Issues

- Summary: linked allocation solves the external fragmentation and size-declaration problems of contiguous allocation,
- However, it can't support efficient direct access

#3. Indexed Allocation



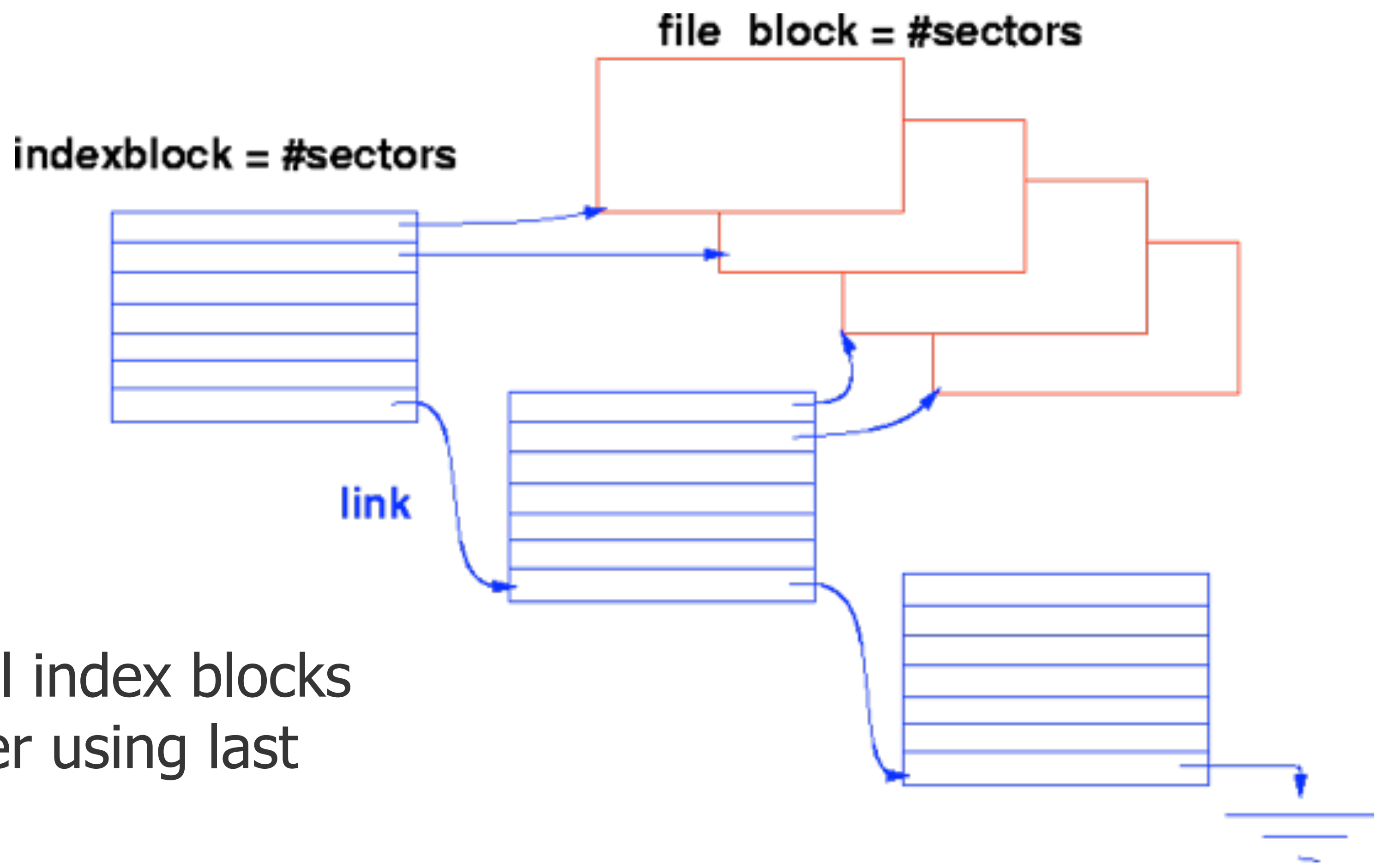
Indexed Allocation

- Solves external fragmentation
- Supports sequential and direct access
- Access requires at most one access to index block first. This can be cached in main memory

Indexed Allocation

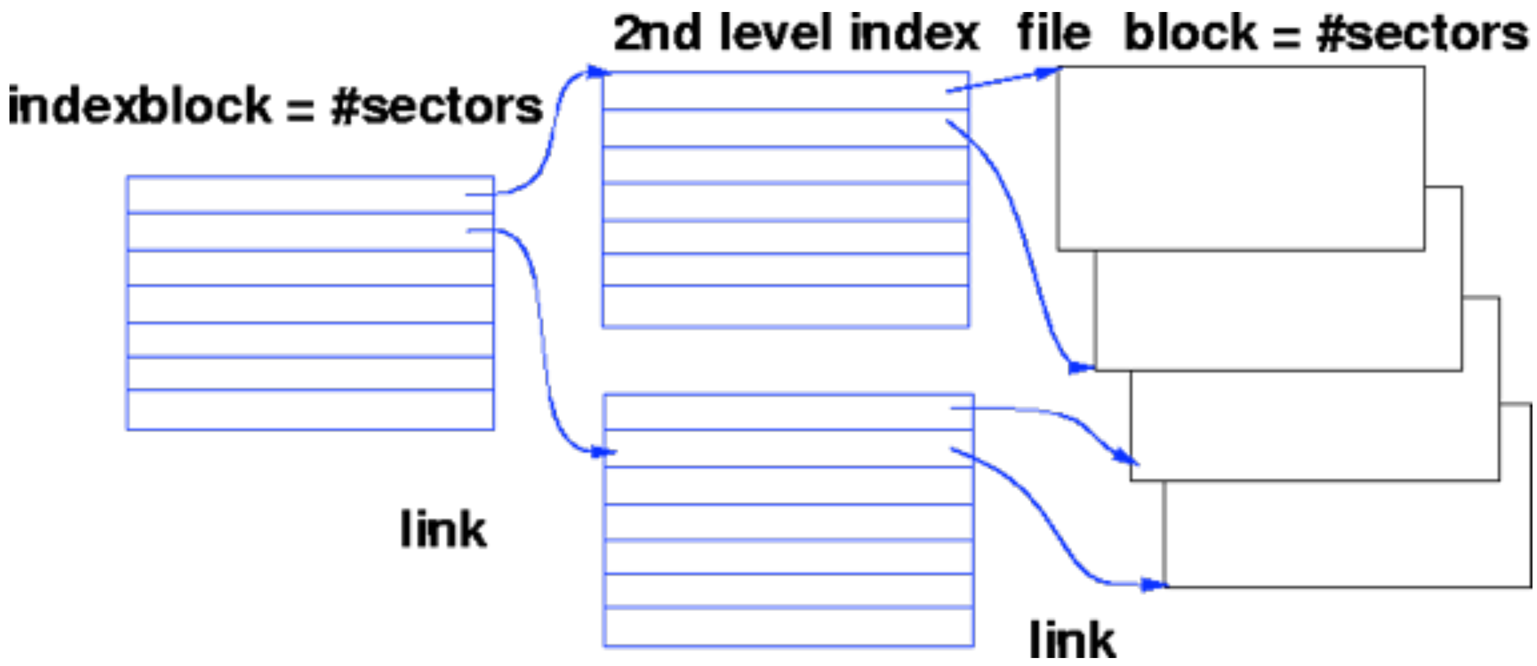
- File can be extended by rewriting a few blocks and index block
- Requires extra space for index
- What if we have a big file whose blocks can't all be listed in a single index block?

Big file solution 1: Linked Indexed Files



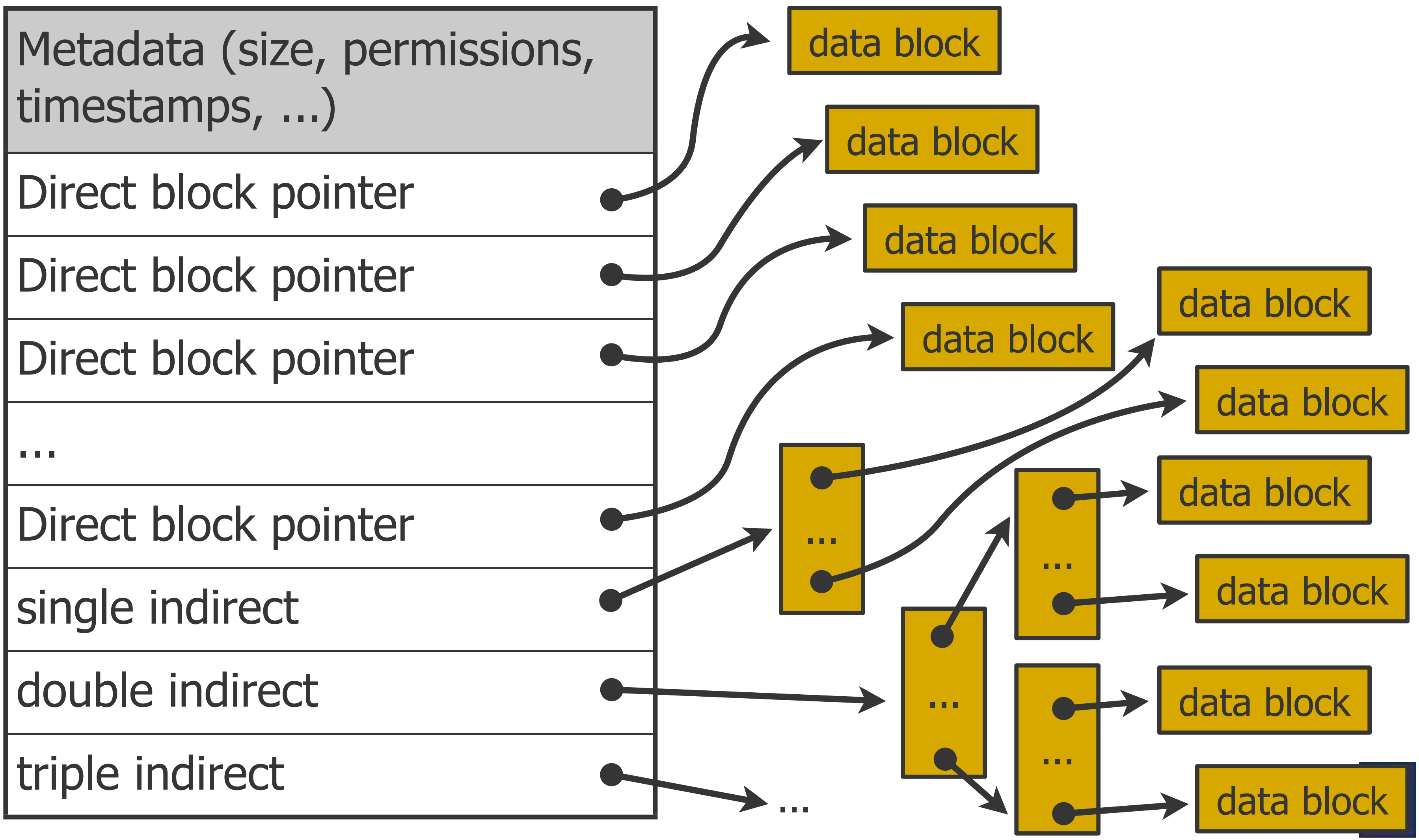
Link full index blocks together using last entry.

Multilevel Indexed File



Multiple levels of index blocks

Multilevel indexed example: UNIX inodes



Free Space Management

- When we want to add a block to a file, how do we find a free one?
  - (does this sound familiar...?)

Option #1: bit vector

- A bit map is kept of free blocks
- Each bit in a vector represents one block
- If the block is free, the bit is zero
- Simple to find *n* consecutive free blocks
- Overhead is bit map
- Example: BSD file system

Option #2: free list

- Keep a linked list of free blocks
- Not very efficient to traverse linked list
- But typical operation is just to add/ remove the first element: no traversal

- Which one needs more space: bit map or linked list?

Option #2: free list

- Keep a linked list of free blocks
- Not very efficient to traverse linked list
  - E.g., we need to read a whole block just to update the list head pointer

- Question to ponder: Which one needs more space -- bit map or linked list?

Option #2a: list of indices

- Linked list of indices
  - A linked list of index blocks is kept
  - Each index block contains addresses of free blocks and a
  - Pointer to the next index block
- A large number of free blocks can be found quickly

Variation: list of contiguous free blocks

- Linked list of contiguous blocks that are free
  - The free list node consists of a pointer and the number of free blocks starting from that address
  - Blocks are joined together into larger blocks as necessary

Final word

- There are **many** optimizations implemented in filesystems!
- 7500 RPM drive will take 4 ms just for mean rotational delay
- How many instructions does a 2 GHz processor execute in that time?
  - 8,000,000
- Conclusion: Disk is the bottleneck. Even complex optimizations may be worthwhile.

HW3 Part 4 Question C

- The question is supposed to ask for the total time spent seeking, not the total time overall. In other words, ignore rotational delay and read time.