Lecture 21: Quiz #2 - Review

Instructor: Mitch Neilsen
Office: N219D

Quote of the Day

"One machine can do the work of fifty ordinary men. No machine can do the work of one extraordinary man."

-- Elbert Hubbard

Quiz #2 – Wed., 11/20/2013
(Open Book, Open Notes)

- **Chapters 1-11**
  - Introduction
  - Operating System Structures
  - Processes
  - Threads
  - CPU Scheduling
  - Process Synchronization
  - Deadlocks
  - **Linking**
  - **Memory Management**
  - **Segmentation and Paging**
  - **Virtual Memory**
  - **File Systems**

Linking

- Compiler produces:
  - Symbolic references to external data/functions
  - Table of locally defined data/functions
- Linker
  - Pass 1: Arrange segments in memory
  - Pass 2: Patch addresses (relocate, replace symbols)
- Static vs Dynamic Linking?
- Where do code, data, stack segments come from?
- Sample question:
  - Compare static and dynamic linking:
    - Describe the advantages of dynamic linking over static linking.
    - Describe the problems in dynamic linking that are not present in static linking.

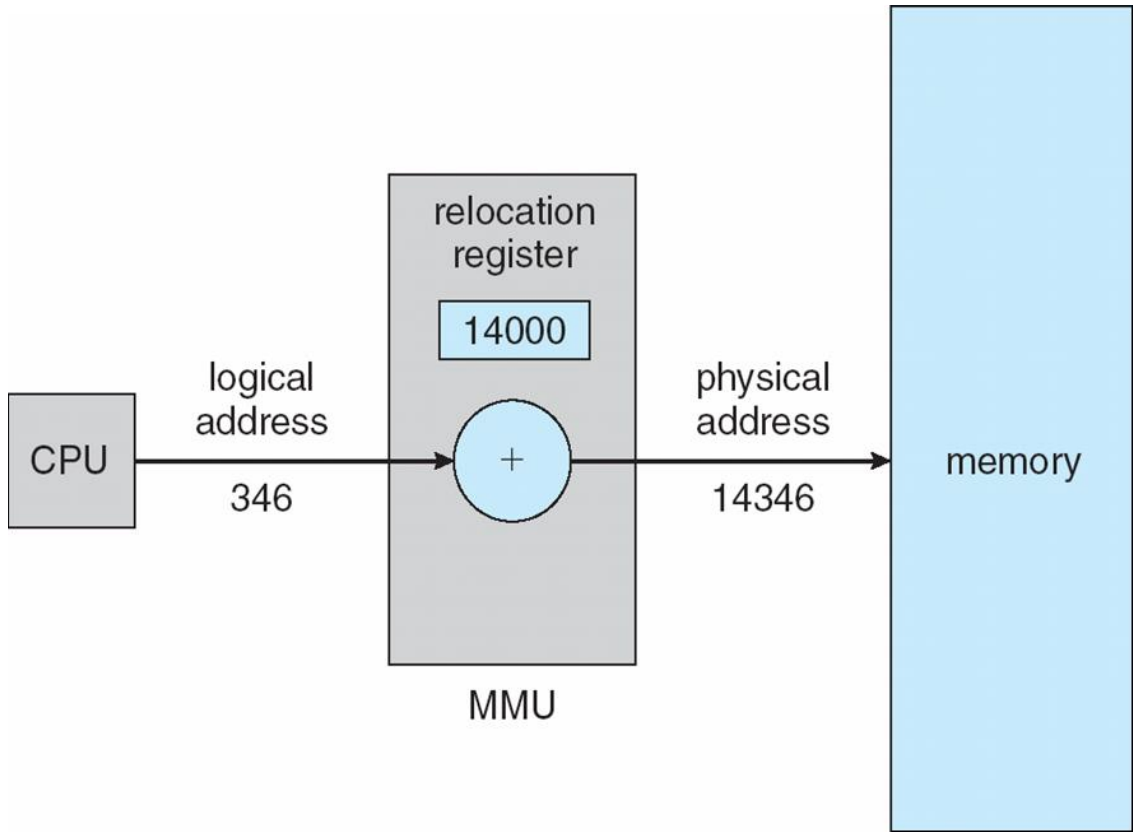
Chapter 8: Memory Management

- Background
- Linking
- **Swapping**
- **Contiguous Memory Allocation**
- **Paging**
- **Structure of the Page Table**
- **Segmentation**

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

Dynamic relocation of segments using a relocation register



Fragmentation

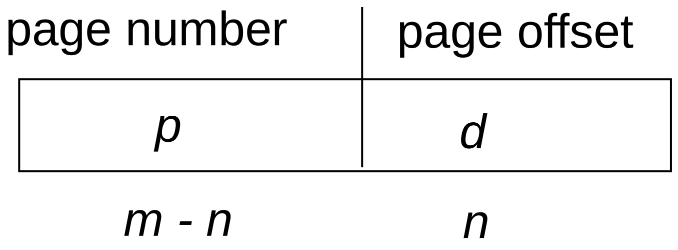
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***n*** pages, need to find *n* free frames and load program
- Set up a page table to translate logical to physical addresses
- Internal fragmentation

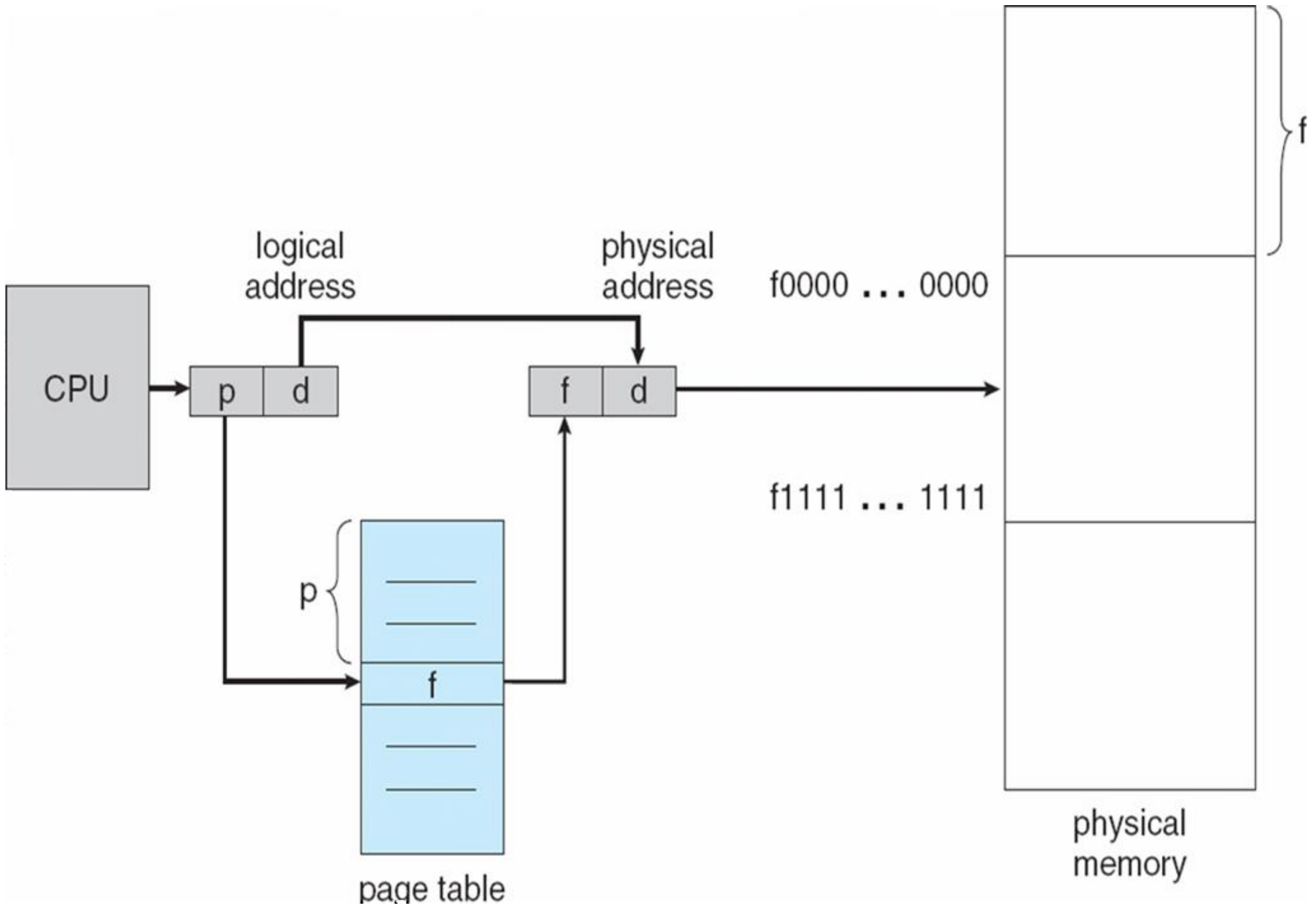
Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory
  - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit



  - For given logical address space 2^m *and page size* 2^n

Paging Hardware



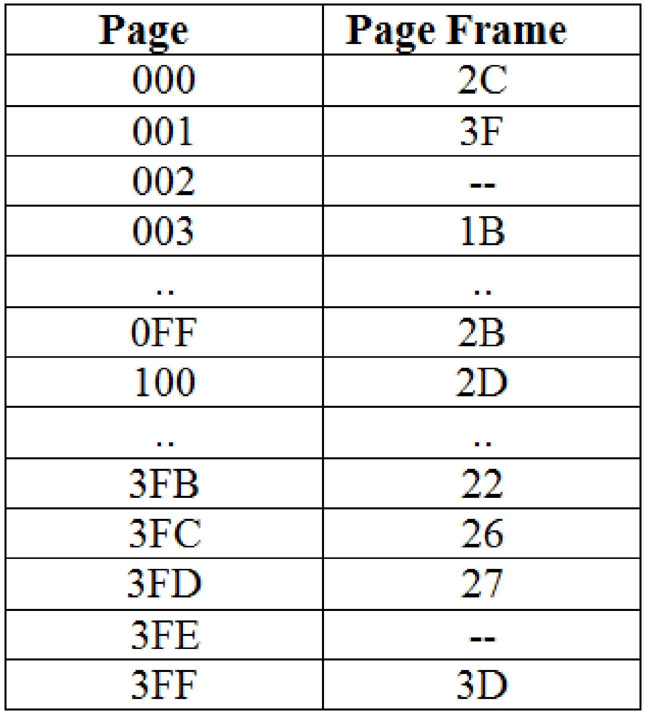
Virtual Address Mapping

- Virtual Address Mapping: Consider the page table shown below for a system with 32-bit virtual addresses and 28-bit physical addresses, and 1 MByte pages; that is, page offsets are represented using 20 bits. Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. If the reference would cause a page fault to be generated, just write “page fault”. If you cannot determine the corresponding physical address or if a page fault will occur, based on the page table entries given, just write “unable to determine”. Note that “--” indicates that the page is currently not loaded into a frame.
  - 3FFFFFFF → ____________________
  - 3FEFFFFF → ____________________
  - 00006FFF → ____________________
  - 00205432 → ____________________
- Also, compute the virtual address corresponding to the given physical addresses.
  - 3FFFFFF → ____________________
  - 1000FFF → ____________________
  - 26F0FAD → ____________________
  - 2C00000 → ____________________

Page	**Page Frame**
000	2C
001	3F
002	--
003	1B
..	..
0FF	2B
100	2D
..	..
3FB	22
3FC	26
3FD	27
3FE	--
3FF	3D

Virtual Address Mapping

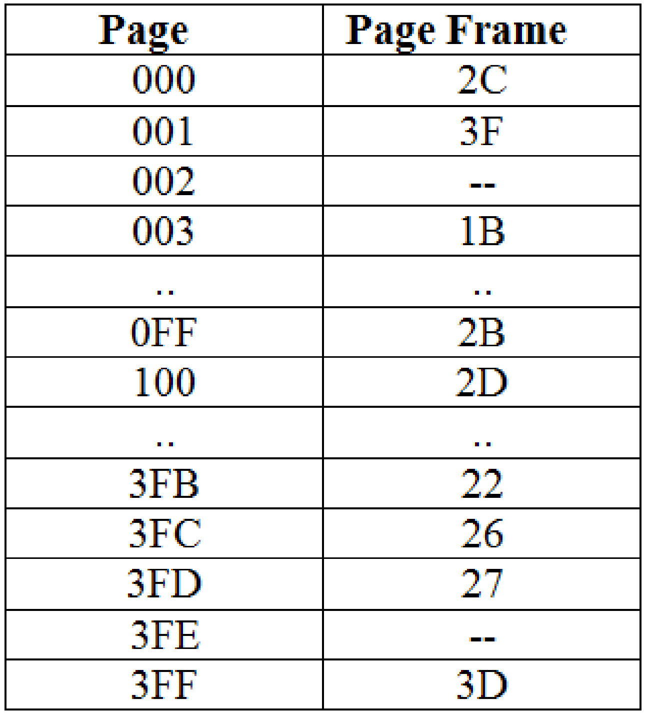
- Virtual Address Mapping: Consider the page table shown below for a system with 32-bit virtual addresses and 28-bit physical addresses, and 1 MByte pages; that is, page offsets are represented using 20 bits. Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. If the reference would cause a page fault to be generated, just write “page fault”. If you cannot determine the corresponding physical address or if a page fault will occur, based on the page table entries given, just write “unable to determine”. Note that “--” indicates that the page is currently not loaded into a frame.
  - **3FF**FFFFF → ___3DFFFFF______
  - **3FE**FFFFF → ___page fault______
  - **000**06FFF → ________________
  - **002**05432 → ________________
- Also, compute the virtual address corresponding to the given physical addresses.
  - **3F**FFFFF → ___001FFFFF_____
  - **10**00FFF → ______N/A_______
  - **26**F0FAD → ________________
  - **2C**00000 → ________________



Page	Page Frame
000	2C
001	3F
002	--
003	1B
..	..
0FF	2B
100	2D
..	..
3FB	22
3FC	26
3FD	27
3FE	--
3FF	3D

Virtual Address Mapping

- Virtual Address Mapping: Consider the page table shown below for a system with 32-bit virtual addresses and 28-bit physical addresses, and 1 MByte pages; that is, page offsets are represented using 20 bits. Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. If the reference would cause a page fault to be generated, just write “page fault”. If you cannot determine the corresponding physical address or if a page fault will occur, based on the page table entries given, just write “unable to determine”. Note that “--” indicates that the page is currently not loaded into a frame.
  - **3FF**FFFFF → ___3DFFFFF______
  - **3FE**FFFFF → ___page fault______
  - **000**06FFF → ___2C06FFF______
  - **002**05432 → ___page fault______
- Also, compute the virtual address corresponding to the given physical addresses.
  - **3F**FFFFF → ___001FFFFF_____
  - **10**00FFF → ______N/A_______
  - **26**F0FAD → ___3FCF0FAD______
  - **2C**00000 → ___00000000______



Page	Page Frame
000	2C
001	3F
002	--
003	1B
..	..
0FF	2B
100	2D
..	..
3FB	22
3FC	26
3FD	27
3FE	--
3FF	3D

Paging and Page Replacement Algorithms

- (5 points) Why are page sizes always a power of 2?
- (10 points) Discuss situations in which the most recently used (MRU) page replacement algorithm generates fewer page faults than the least recently used (LRU) page replacement algorithm. Also, discuss under what circumstances the opposite holds.
- (5 points) Discuss the philosophy behind the second chance (clock) algorithm and how it approximates LRU. Drawings are fine here to help with your explanation.
- (5 points) Give a simple example to show how the LRU algorithm and the second chance (clock) algorithm might select different pages for eviction, or explain why they always select the same page.

Paging and Page Replacement Algorithms

- (5 points) Why are page sizes always a power of 2?

  It is more efficient to break the address up into a fixed number of page number bits and a fixed number of offset bits, rather than performing arithmetic to compute page number and offset.

- (10 points) Discuss situations in which the most recently used (MRU) page replacement algorithm generates fewer page faults than the least recently used (LRU) page replacement algorithm. Also, discuss under what circumstances the opposite holds.

Paging and Page Replacement Algorithms

- (10 points) Discuss situations in which the most recently used (MRU) page replacement algorithm generates fewer page faults than the least recently used (LRU) page replacement algorithm. Also, discuss under what circumstances the opposite holds.

  If the data is processed in a circular queue fashion, then the first page loaded will be the first page to be reused. Example: 3 page frames, pages accessed 0, 1, 2, 3, 0, 1, 2, 3 - MRU: 0F, 1F, 2F, 3F (2 evicted), 0, 1, 2F (1 evicted), 3 = 5 faults, LRU: 0F, 1F, 2F, 3F (0 evicted), 0F (1 evicted), 1F (2 evicted), 2F (3 evicted), 3F (0 evicted) = 8 faults.

- (5 points) Discuss the philosophy behind the second chance (clock) algorithm and how it approximates LRU. Drawings are fine here to help with your explanation.

Paging and Page Replacement Algorithms

- (5 points) Discuss the philosophy behind the second chance (clock) algorithm and how it approximates LRU. Drawings are fine here to help with your explanation.

  The clock algorithm keeps a circular list of pages in memory, with the "hand" pointing to the oldest page in the list. When a page fault occurs and no empty frames exist, then the R (referenced) bit is inspected at the hand's location. If R is 0, the new page is put in place of the page the "hand" points to, otherwise the R bit is cleared. Then, the clock hand is incremented and the process is repeated until a page is replaced.

- (5 points) Give a simple example to show how the LRU algorithm and the second chance (clock) algorithm might select different pages for eviction, or explain why they always select the same page.

Paging and Page Replacement Algorithms

- (5 points) Discuss the philosophy behind the second chance (clock) algorithm and how it approximates LRU. Drawings are fine here to help with your explanation.

  The clock algorithm keeps a circular list of pages in memory, with the "hand" pointing to the oldest page in the list. When a page fault occurs and no empty frames exist, then the R (referenced) bit is inspected at the hand's location. If R is 0, the new page is put in place of the page the "hand" points to, otherwise the R bit is cleared. Then, the clock hand is incremented and the process is repeated until a page is replaced.

- (5 points) Give a simple example to show how the LRU algorithm and the second chance (clock) algorithm might select different pages for eviction, or explain why they always select the same page.

  LRU keeps track of page usage over a short period of time, while NRU - second chance (clock) algorithms just looks at the usage in the last clock interval. Pages 1, 2, and 3 are loaded and used in the last 1, 2, and 3 clock ticks. LRU evicts 3, clock 2.

Dynamic Memory Management

A dynamic memory allocator uses a linked list to track free blocks. In the allocator, all blocks are allocated in multiples of 4 bytes in length. Answer the following questions assuming that the free list contains just two blocks, of 28 and 16 bytes each, in that order. Ignore any space required by bookkeeping overhead.

- (5 points) If a process calls malloc( ) with a request for 7 bytes, then the allocator returns a pointer to a block of length 8 because all blocks are allocated in full-word (4 byte) units. The last byte in the block is wasted. Is this an example of internal or external fragmentation? Explain briefly.
- (5 points) If the process calls malloc( ) with a request for 24 bytes and the allocator is using the first-fit algorithm, the request for 24 bytes will be satisfied by breaking the block of size 28 into two parts, and returning the remaining 4 bytes to the free list. This small block of size 4 bytes is sometimes referred to as “sawdust” because it probably will never be used. Is this an example of internal or external fragmentation? Explain briefly.
- (5 points) List a sequence of malloc( ) calls that would succeed using first-fit allocation, but fail with best-fit, or explain why such a sequence cannot exist.

Dynamic Memory Management

- (5 points) List a sequence of malloc( ) calls that would succeed with best-fit, but fail with first-fit, or explain why such a sequence cannot exist.

- (5 points) List a sequence of malloc( ) calls that would succeed with worst-fit, but fail with best-fit, or explain why such a sequence cannot exist.

Dynamic Memory Management

The free list contains just two blocks, of 28 and 16 bytes each, in that order. Ignore any space required by bookkeeping overhead.

- (5 points) If a process calls malloc( ) with a request for 7 bytes, then the allocator returns a pointer to a block of length 8 because all blocks are allocated in full-word (4 byte) units. The last byte in the block is wasted. Is this an example of internal or external fragmentation? Explain briefly.

  Internal fragmentation because it is inside the block allocated to the process.

- (5 points) If the process calls malloc( ) with a request for 24 bytes and the allocator is using the first-fit algorithm, the request for 24 bytes will be satisfied by breaking the block of size 28 into two parts, and returning the remaining 4 bytes to the free list. This small block of size 4 bytes is sometimes referred to as “sawdust” because it probably will never be used. Is this an example of internal or external fragmentation? Explain briefly.

  External fragmentation because it is outside the memory allocated to the process.

Dynamic Memory Management

- (5 points) List a sequence of malloc( ) calls that would succeed using first-fit allocation, but fail with best-fit, or explain why such a sequence cannot exist.

  15, 13, 16 - with first-fit, the first free block will be used for the first two requests, and the second block will be used for the third request, with best-fit, the first request will be allocated to the second block, the second will be allocated to the first block, and the third request for 16 cannot be satisfied.

Dynamic Memory Management

The free list contains just two blocks, of 28 and 16 bytes each, in that order. Ignore any space required by bookkeeping overhead.

- (5 points) List a sequence of malloc( ) calls that would succeed with best-fit, but fail with first-fit, or explain why such a sequence cannot exist.

- (5 points) List a sequence of malloc( ) calls that would succeed with worst-fit, but fail with best-fit, or explain why such a sequence cannot exist.

Dynamic Memory Management

The free list contains just two blocks, of 28 and 16 bytes each, in that order. Ignore any space required by bookkeeping overhead.

- (5 points) List a sequence of malloc( ) calls that would succeed with best-fit, but fail with first-fit, or explain why such a sequence cannot exist.

  16, 15, 13 - best fit, request for 16 is allocated from the second block, and the subsequent requests are allocated to the first block, first fit - the request for 16 is allocated to the first block, the request for 15 is allocated to the second block, and the third request for 13 cannot be satisfied.

- (5 points) List a sequence of malloc( ) calls that would succeed with worst-fit, but fail with best-fit, or explain why such a sequence cannot exist.

Dynamic Memory Management

The free list contains just two blocks, of 28 and 16 bytes each, in that order. Ignore any space required by bookkeeping overhead.

- (5 points) List a sequence of malloc( ) calls that would succeed with best-fit, but fail with first-fit, or explain why such a sequence cannot exist.

  16, 15, 13 - best fit, request for 16 is allocated from the second block, and the subsequent requests are allocated to the first block, first fit - the request for 16 is allocated to the first block, the request for 15 is allocated to the second block, and the third request for 13 cannot be satisfied.

- (5 points) List a sequence of malloc( ) calls that would succeed with worst-fit, but fail with best-fit, or explain why such a sequence cannot exist.

  14, 16, 14 - worst-fit allocates first request to the first block, second request can only be satisfied by the second block, and the third request to the first block; best fit - allocates the first request to the second block, and the second to the first block, and the third request cannot be satisfied.

System Calls and Swapping

- Discuss the advantages of using copy-on-write (COW) when creating a new process by using the Unix fork( ) system call.
- Why don’t we typically use LRU for selecting pages to swap, but rather use an algorithm that only approximates LRU, such as the clock algorithm?
- Assume that we have a 2 GHz processor and a hard disk drive with a 10ms average seek time, approximately how many instructions could be executed while waiting for a disk access to complete?
- What is thrashing? What can be done to prevent thrashing from occurring?
- What is Belady’s anomaly? Can the LRU algorithm exhibit Belady’s anomaly?

System Calls and Swapping

- Discuss the advantages of using copy-on-write (COW) when creating a new process by using the Unix fork( ) system call.

  Copy-on-write allows a child process to share the same memory as the parent until it is absolutely necessary to make a copy. This speeds up the creation of a child process, and reduces the amount of memory required.

System Calls and Swapping

- Discuss the advantages of using copy-on-write (COW) when creating a new process by using the Unix fork( ) system call.

  Copy-on-write allows a child process to share the same memory as the parent until it is absolutely necessary to make a copy. This speeds up the creation of a child process, and reduces the amount of memory required.

- Why don't we typically use LRU for selecting pages to swap, but rather use an algorithm that only approximates LRU, such as the clock algorithm?

  LRU is too computationally expensive (takes too much time) to be used for paging, so an algorithm that closely approximates it is used instead to exploit temporal and spatial locality.

- Assume that we have a 2 GHz processor and a hard disk drive with a 10ms average seek time, approximately how many instructions could be executed while waiting for a disk access to complete?

System Calls and Swapping

- Assume that we have a 2 GHz processor and a hard disk drive with a 10ms average seek time, approximately how many instructions could be executed while waiting for a disk access to complete?

  2 GHz = 2 billion cycles/sec, if each instruction takes a single cycle, then 2 billion instructions per second, or 20 million instructions in 1/100 sec = 10 msec.

System Calls and Swapping

- What is thrashing? What can be done to prevent thrashing from occurring?

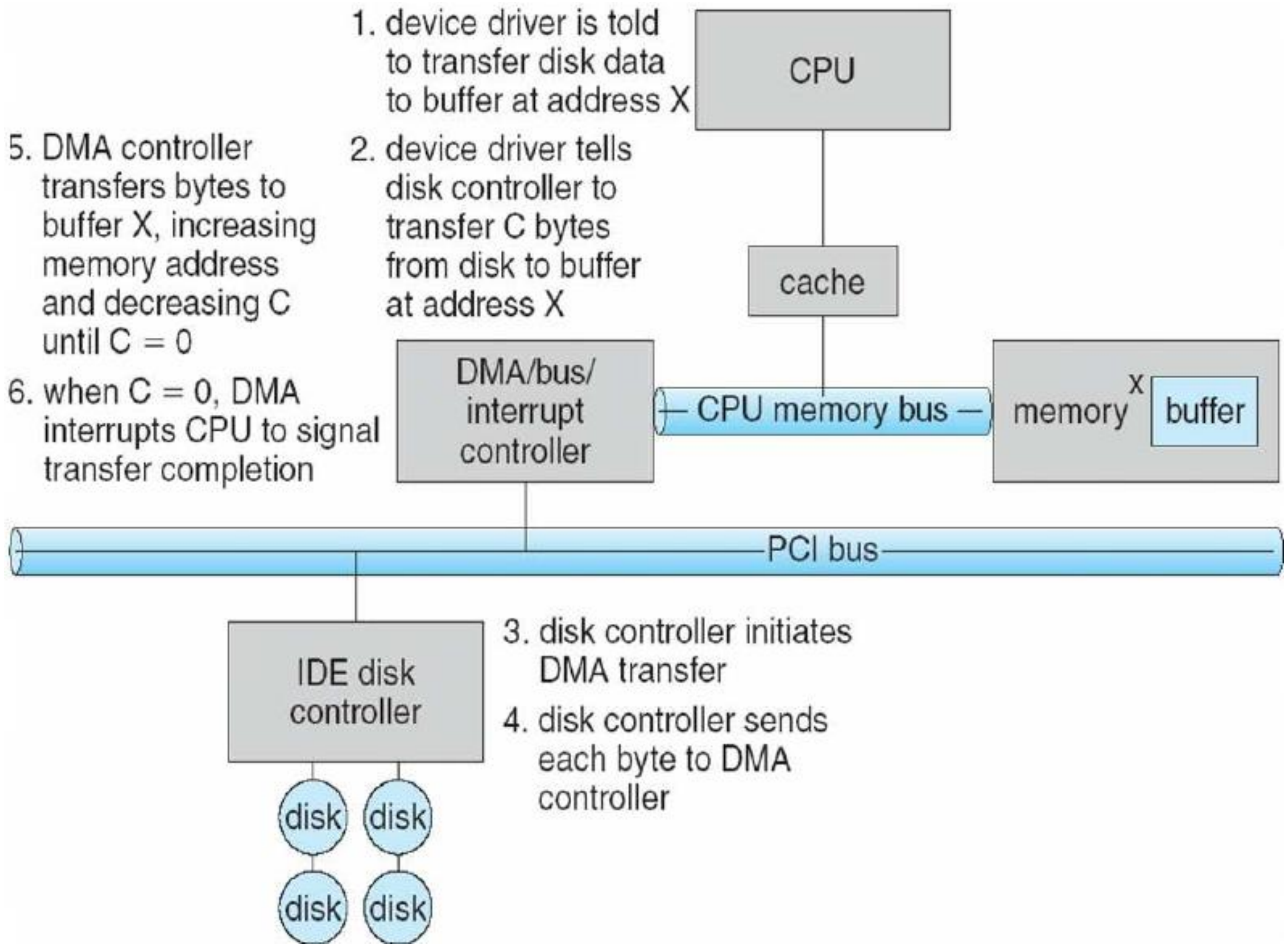
  Thrashing occurs when two or more processes spend a disproportionate time just accessing a shared resource such as memory, and consequently don't get very much useful work done. For example, a printer that doesn't have enough memory, may thrash bringing in pages of a document to be printed from different print jobs. To avoid thrashing, increase the amount of shared resource; e.g., increase the amount of memory on the printer.

- What is Belady’s anomaly? Can the LRU algorithm exhibit Belady’s anomaly?

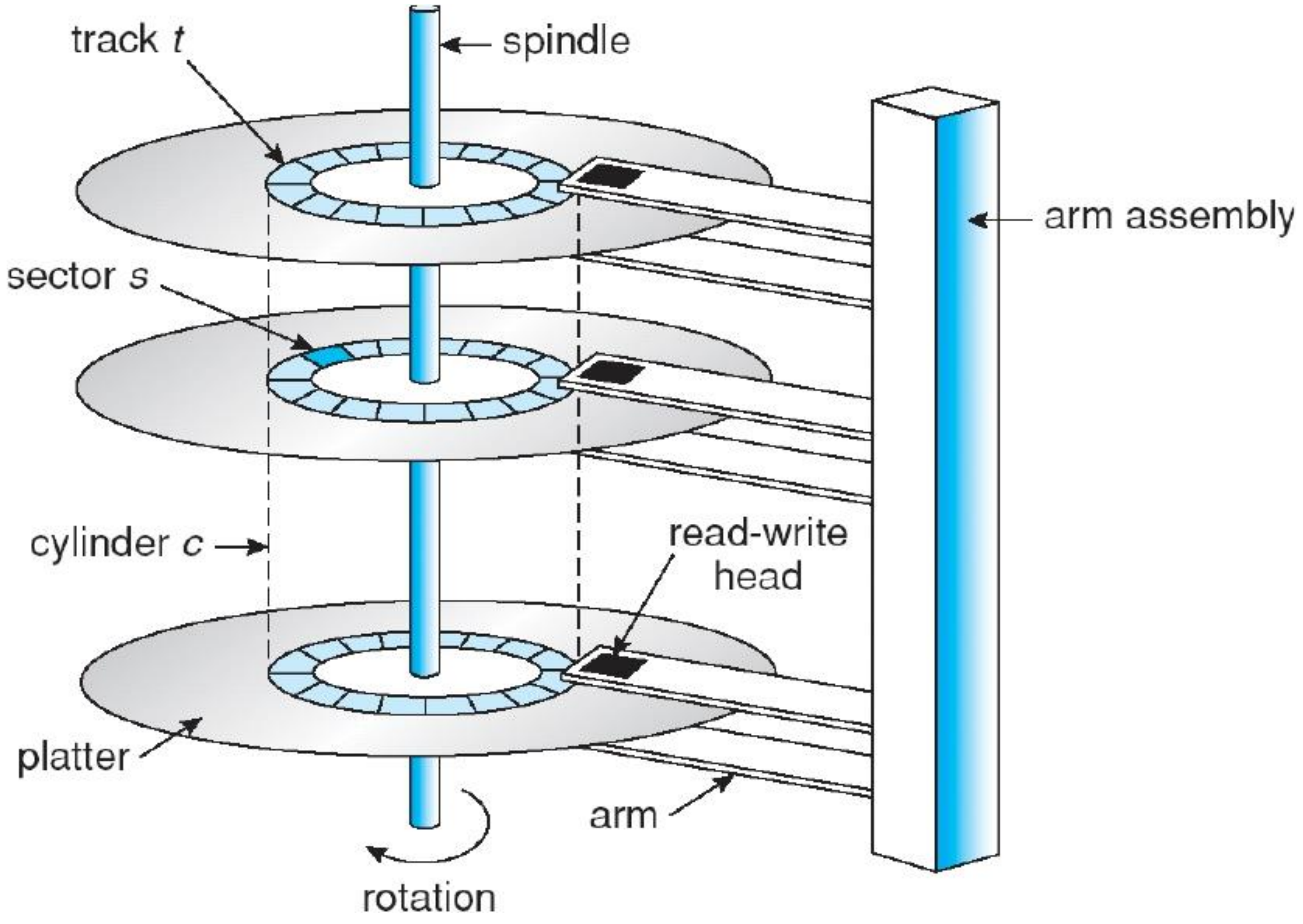
  Belady's anomaly occurs when increasing the number of page frames doesn't result in the same or fewer page faults for a given reference string.

  No. The LRU algorithm cannot exhibit Belady's anomaly, but the FIFO algorithm can.

File Systems: disk read w/ DMA


1. device driver is told to transfer disk data to buffer at address X
CPU
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
cache
6. when C = 0, DMA interrupts CPU to signal transfer completion
DMA/bus/ interrupt controller
CPU memory bus
memory
X
buffer
PCI bus
IDE disk controller
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
disk
disk
disk
disk

Cylinders, Tracks, & Sectors



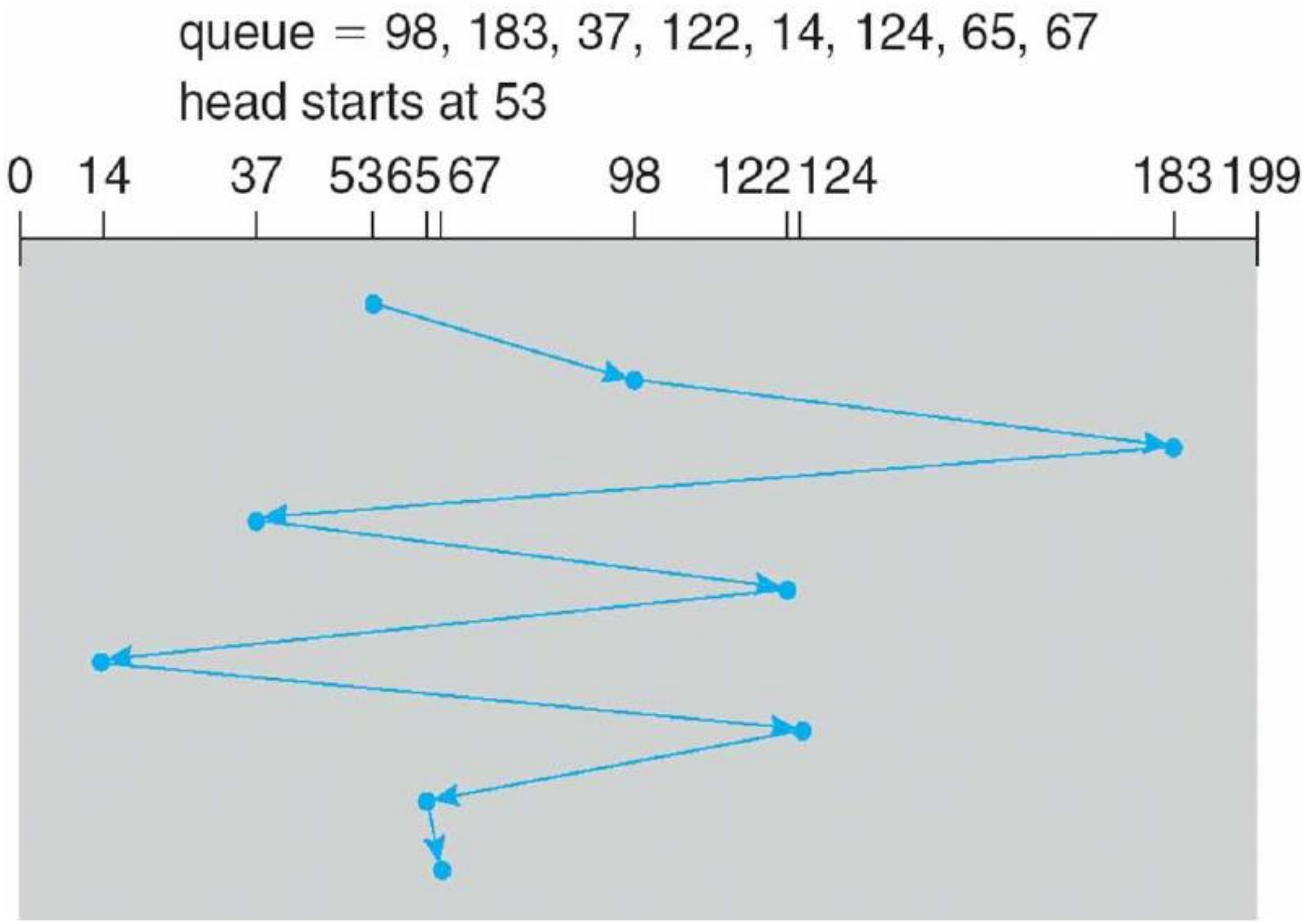
Disk performance

- **Placement & ordering of requests a huge issue**
  - Sequential I/O much, much faster than random
  - Long seeks much slower than short ones
  - Power might fail any time, leaving inconsistent state
- **Must be careful about order for crashes**
  - More on this in next two lectures
- **Try to achieve contiguous accesses where possible**
  - E.g., make big chunks of individual files contiguous
- **Try to order requests to minimize seek times**
  - OS can only do this if it has a multiple requests to order
  - Requires disk I/O concurrency
  - High-performance apps try to maximize I/O concurrency
- **Next: How to schedule concurrent requests**

Scheduling: FCFS

- **“First Come First Served”**
  - Process disk requests in the order they are received
- **Advantages**
  - Easy to implement
  - Good fairness
- **Disadvantages**
  - Cannot exploit request locality
  - Increases average latency, decreasing throughput

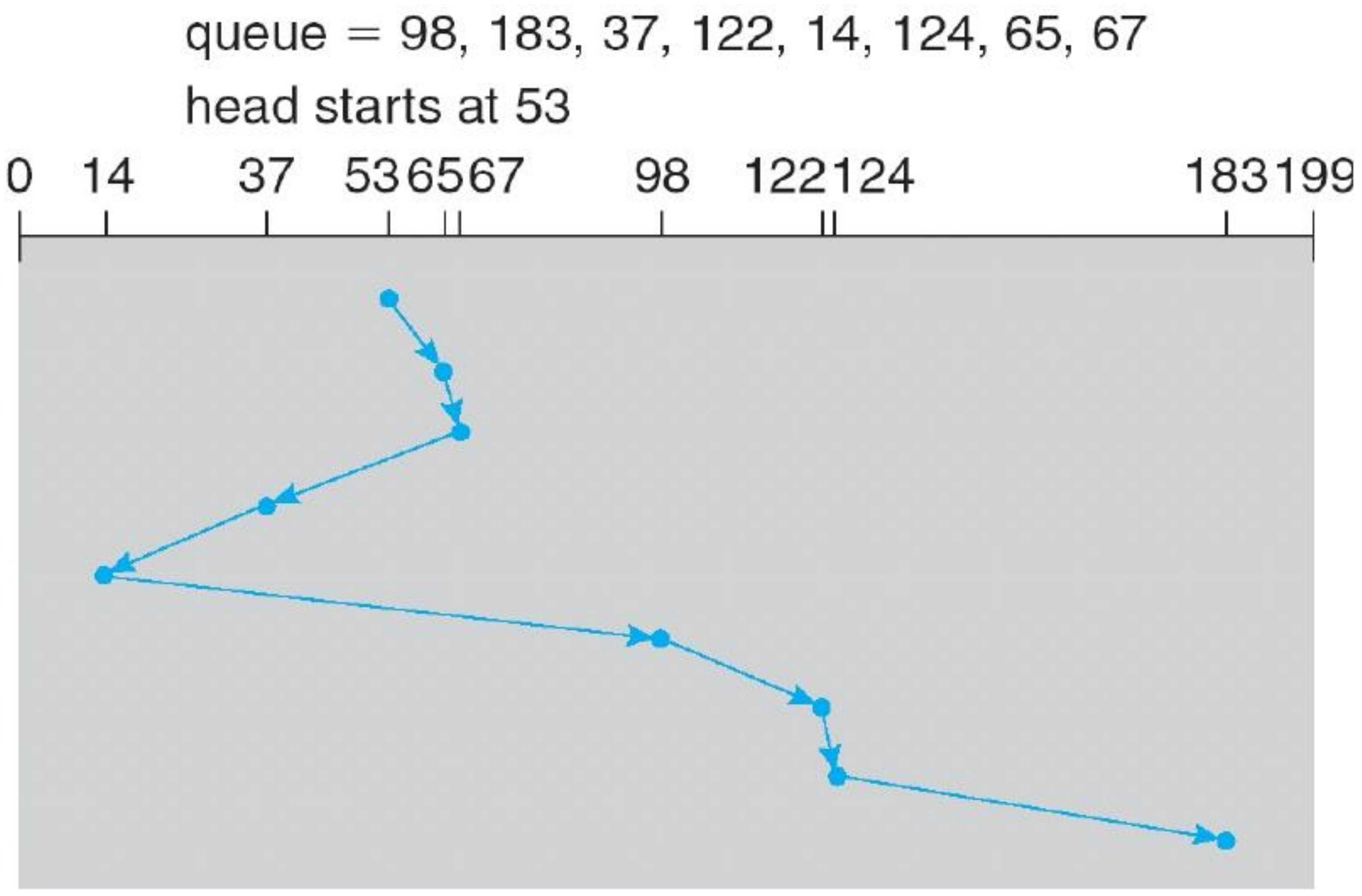
FCFS example



Shortest positioning time first (SPTF)

- **Shortest positioning time first (SPTF)**
  - Always pick request with shortest seek time
- **Advantages**
  - Exploits locality of disk requests
  - Higher throughput
- **Disadvantages**
  - Starvation
  - Don’t always know what request will be fastest
- **Improvement: Aged SPTF**
  - Give older requests higher priority
  - Adjust “effective” seek time with weighting factor:
    $T_{\text{eff}} = T_{\text{pos}} - W \cdot T_{\text{wait}}$
- **Also called Shortest Seek Time First (SSTF)**

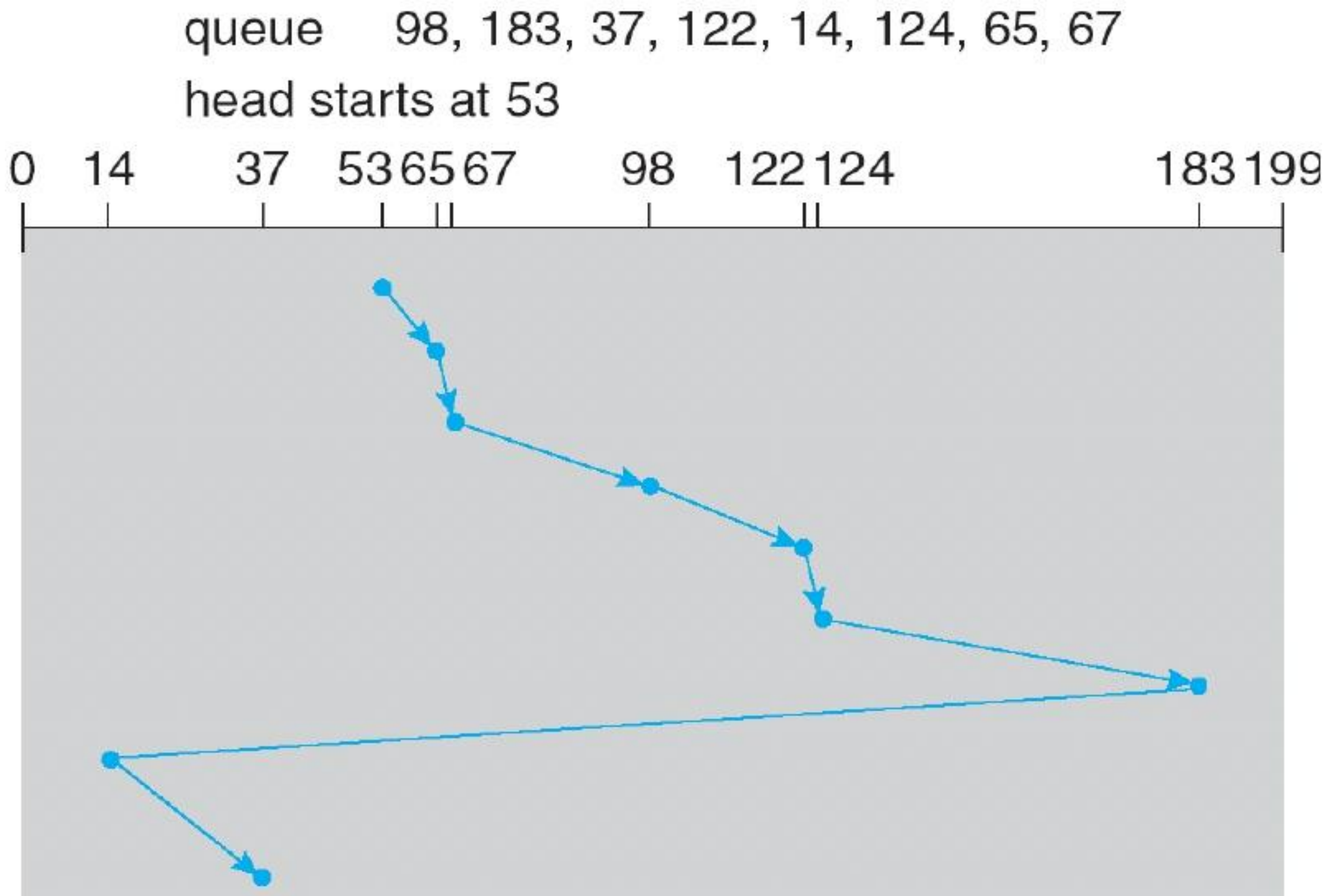
SPTF example



“Elevator ” scheduling (SCAN)

- **Sweep across disk, servicing all requests passed**
  - Like SPTF, but next seek must be in same direction
  - Switch directions only if no further requests
- **Advantages**
  - Takes advantage of locality
  - Bounded waiting
- **Disadvantages**
  - Cylinders in the middle get better service
  - Might miss locality SPTF could exploit
- **CSCAN: Only sweep in one direction**
  Very commonly used algorithm in Unix
- **Also called LOOK/CLOOK in textbook**

CSCAN example



Other

- Review Project #2 - System Calls
  - Data structures and synchronization primitives needed
  - Stack set-up and parameter passing
- Review File Systems
  - Unix File System (UFS)
  - Berkeley Fast File System (FFS)