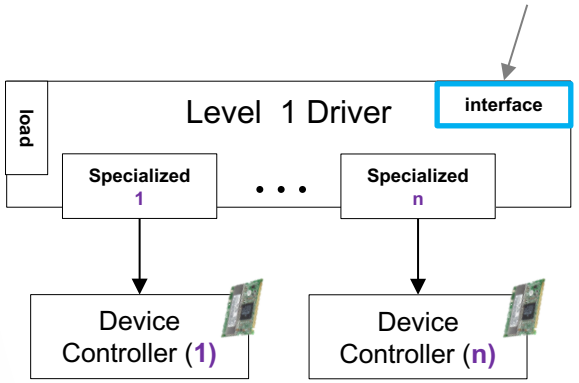
Drivers structure and operation

Operating Systems Design

Contents

- Driver Structure
- Driver Operation
- Driver scheduling example: disk scheduling

Step 1. Interface for system calls



- Interface for system calls:
  - Set of functions providing access to devices.
- Features:
  - Standardization:
    - If a hardware device is valid for a task, the user program or service using the OS might be able to use it without modifying source code.
  - Using shared interfaces built on existing ones:
    - Creating a new call is more expensive than reusing those existing.

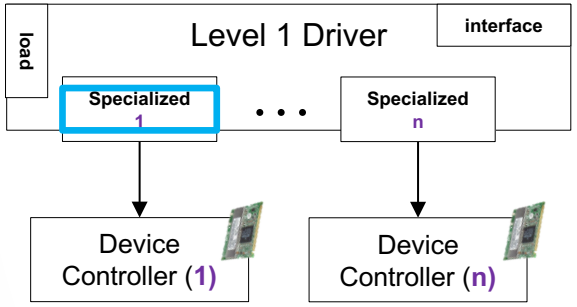
General file operations

```
struct file_operations {
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
                int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *);
        int (*fasync) (int, struct file *, int);
        int (*check_media_change) (kdev_t dev);
        int (*revalidate) (kdev_t dev);
        int (*lock) (struct file *, int, struct file_lock *);
};
```

Linux. Interface for the system calls

- Call to control access to device:
  - Open (name, flags, mode)
  - Close (descriptor)
- Calls to exchange data with the device:
  - Read (descriptor, buffer, size)
  - Write (descriptor, buffer, size)
  - Lseek (descriptor,offset, origin)
- Calls specific for the device:
  - Ioctl (descriptor, no operation, parameters)
    - Allow the execution of any service with any parameters.
    - Operations must be published to avoid conflicts between different drivers.

Step 2. Request to the device controller



- Two functions needed, at least:
  - Operation request
    - Requested from an OS service.
  - Managing device interrupt
    - Executed when the interrupt is received.
- Adapting the hardware controller of the device:
  - Fast device
  - Slow device
    - Independent requests
    - Dependent requests

Installing an Interrupt Handler

- An ISR must be installed before it can be used. This is done using the request_irq() call:

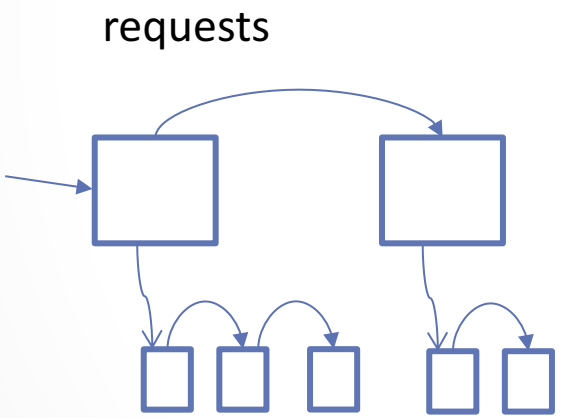
```
int request_irq (unsigned int irq,
                 void (*handler)(int, struct pt_regs *),
                 unsigned long irqflags,
                 const char *devname,
                 void *dev_id)
```

  - irq is the IRQ for which the handler is installed
  - handler is the handler function
  - irq_flags might include SA_INTERRUPT to specify fast IRQs or SA_SHIRQ to specify a shared IRQ (or an ORed combination).
  - devname should be the name of the device driver (used in /proc for accounting)
  - dev_id is meaningless to the kernel, but is passed unchanged to the handler

Request to device controller. Disk write

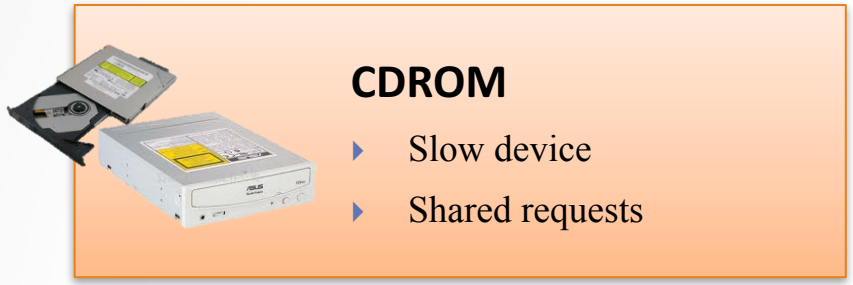
Disk

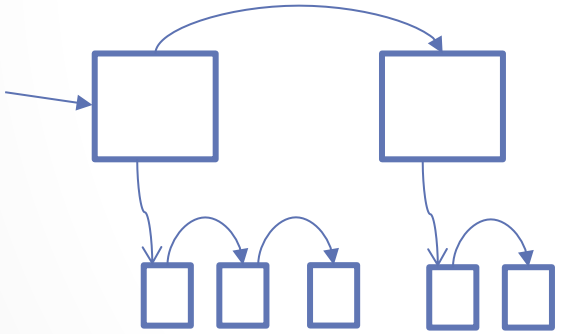
- Slow device
- Shared requests



- Data request:
  - If another process made the request
    - Updating data
    - Sleep and Wait for request service
  - If new request
    - Build a new request
    - Queue the request
    - Sleep and Wait for request service
- Device interrupt manager:
  - Wake up all processes slept waiting for request to end
  - If pending requests
    - Start serving the next request

Request to device controller. CDROM read

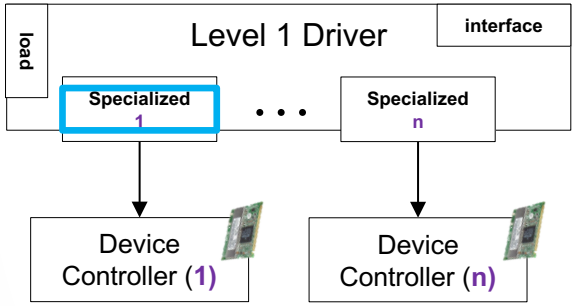




Blocked processes
(n per request)

- Data request:
  - If another process made the request
    - Update data
    - Sleep and Wait for request service
  - If new request
    - Build a new request
    - Queue the request
    - Sleep and Wait for request service
  - Copy read data
- Device interrupt manager:
  - Insert data into the buffer
  - Wake up all processes waiting for request to end
  - If pending requests
    - Serve the next request

Step 3. Driver I/O scheduling



- Queue of requests per device.
- Each driver has an I/O scheduler to minimize the service time.
  - CRITICAL FOR PERFORMANCE
  - Disk blocks are scheduled to minimize time spent in disk head movement..
  - Scheduling policy must be defined
- The I/O scheduler usually makes two basic operations:
  - Ordering: the requests are inserted in the queue followin some criteria.
  - Fussion: several small consecutive requests are joined in a single request.

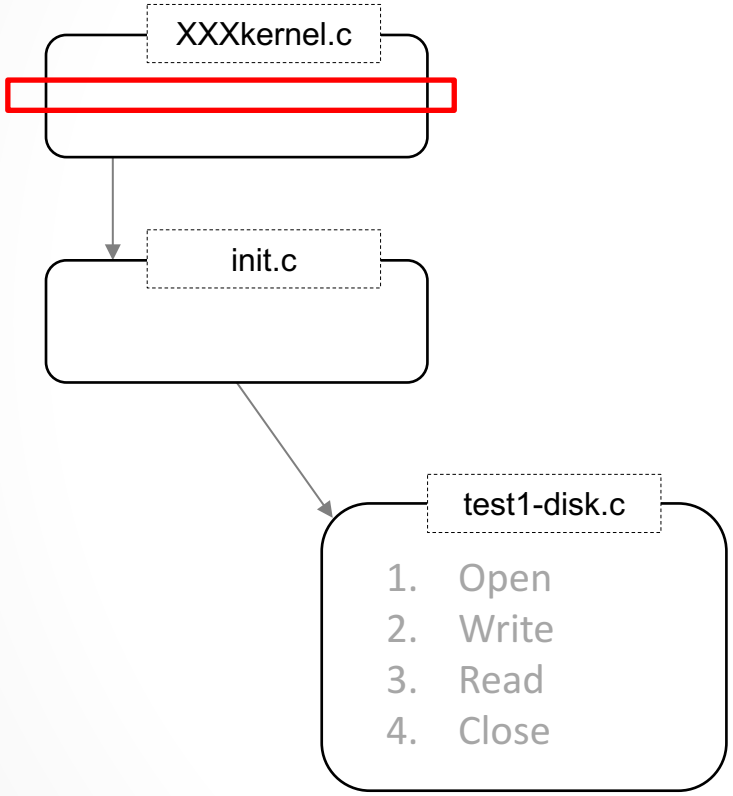
E.g. Disk scheduling algorithms

Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First-in-first-out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest-service-time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of *N* records at a time	Service guarantee
FSCAN	*N*-step-SCAN with *N* = queue size at beginning of SCAN cycle	Load sensitive

Contents

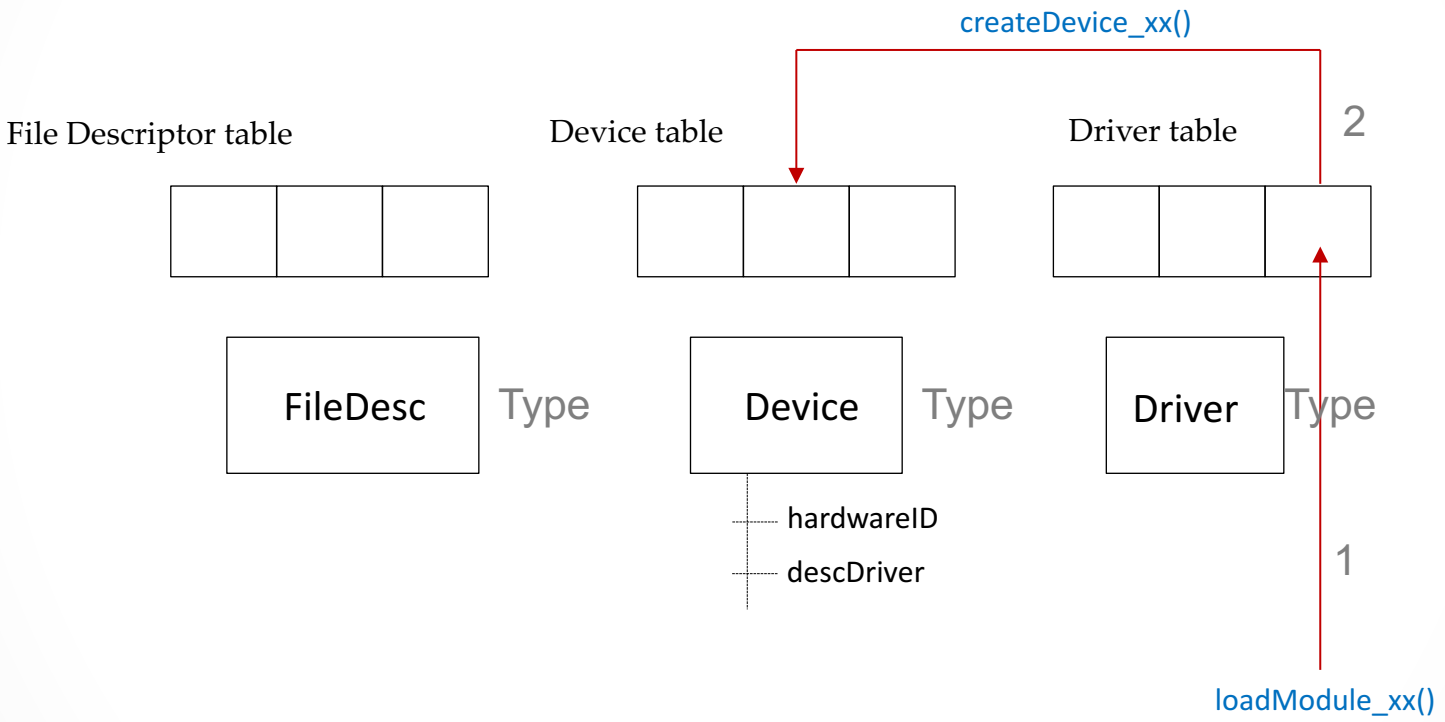
- Driver Structure
- Driver Operation
- Driver scheduling example: disk scheduling

Operation flow

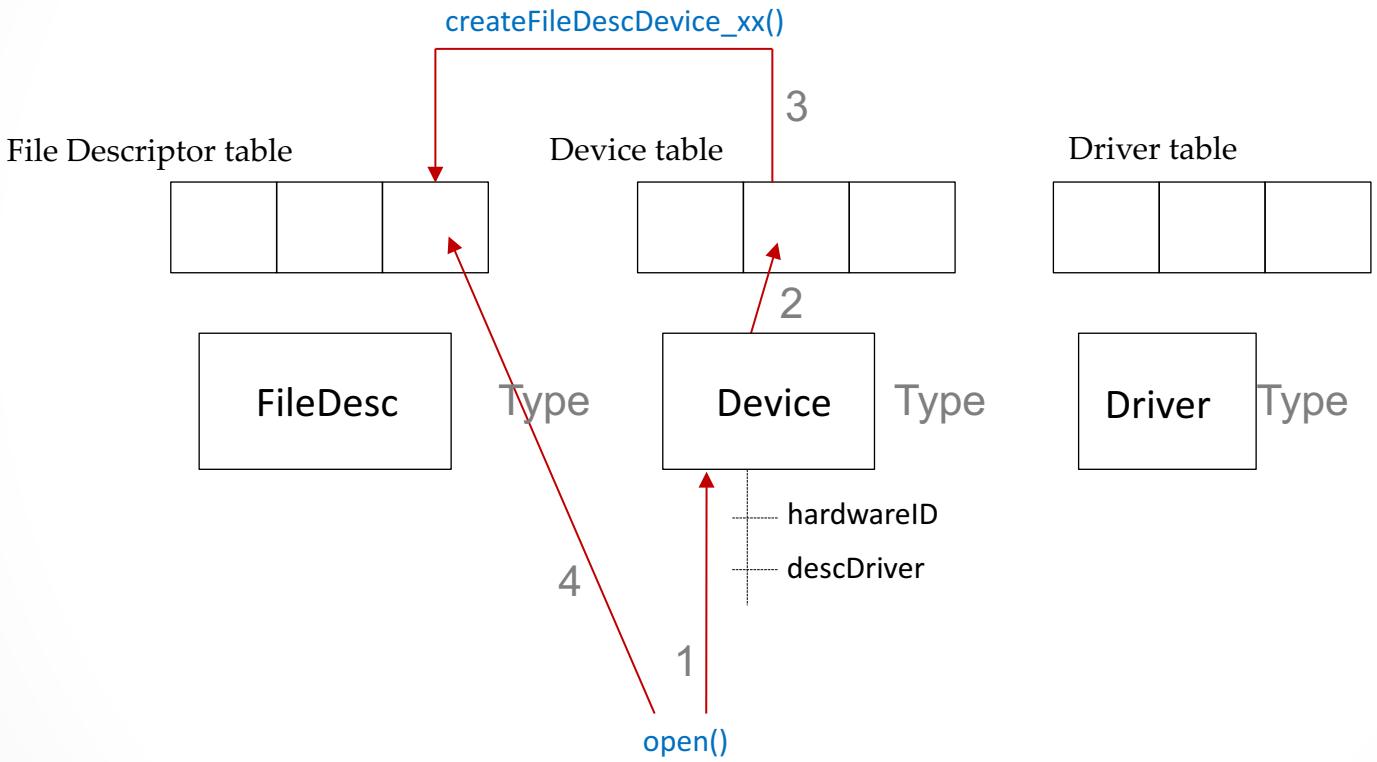


1. Load
2. Open
3. Write
4. Read
5. Close

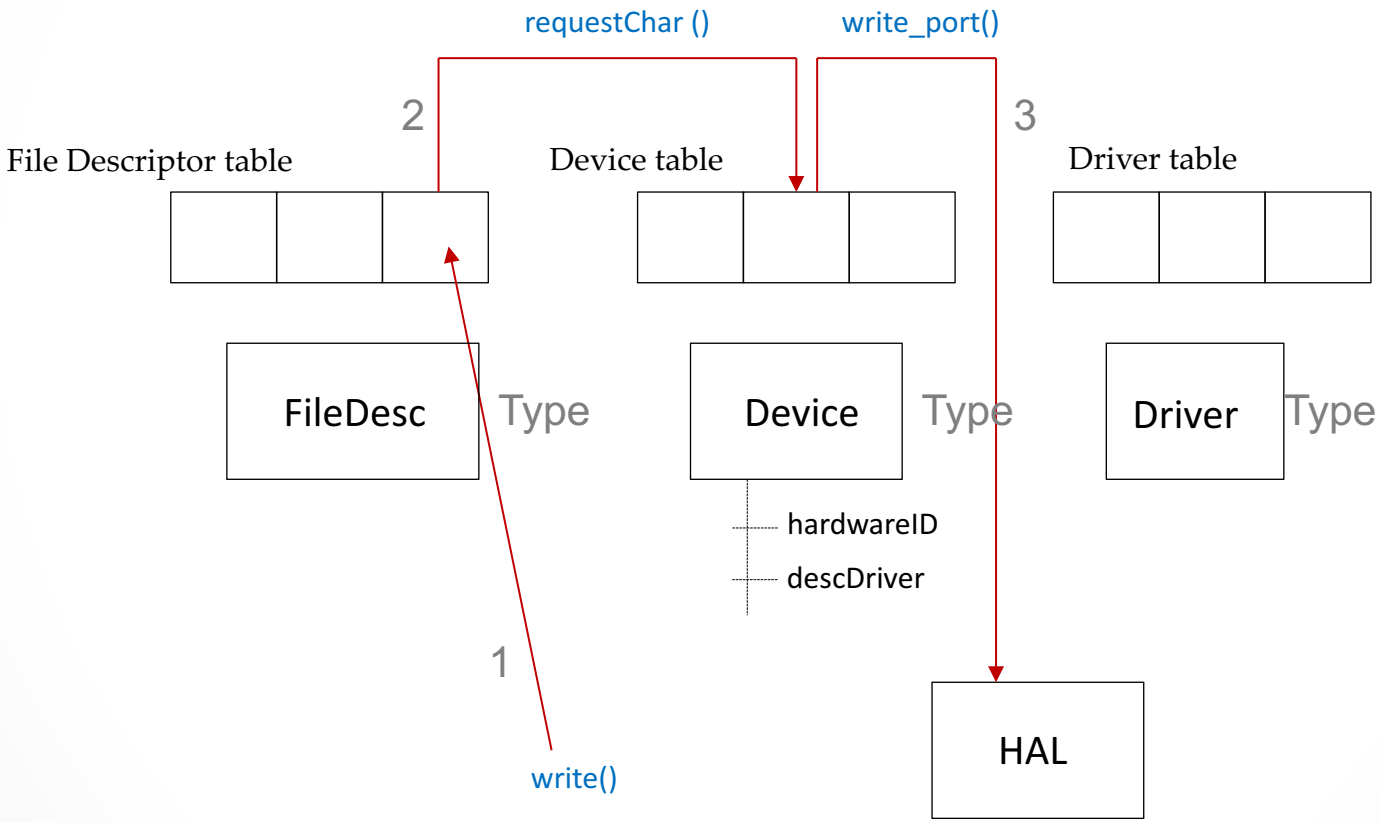
Loading


createDevice_xx()
File Descriptor table
Device table
Driver table
2
FileDesc
Type
Device
Type
Driver
Type
hardwareID
descDriver
1
loadModule_xx()

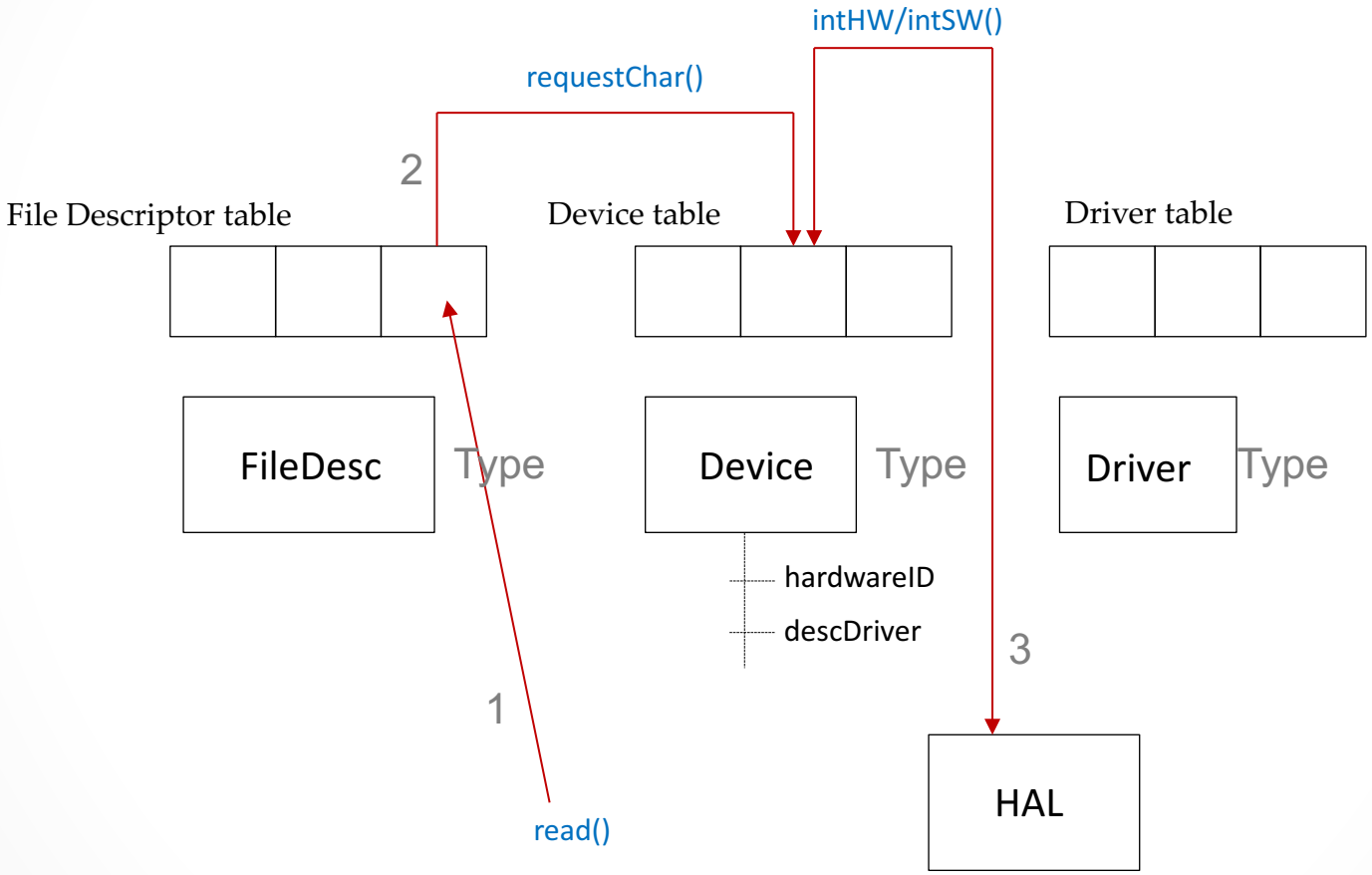
Open the device



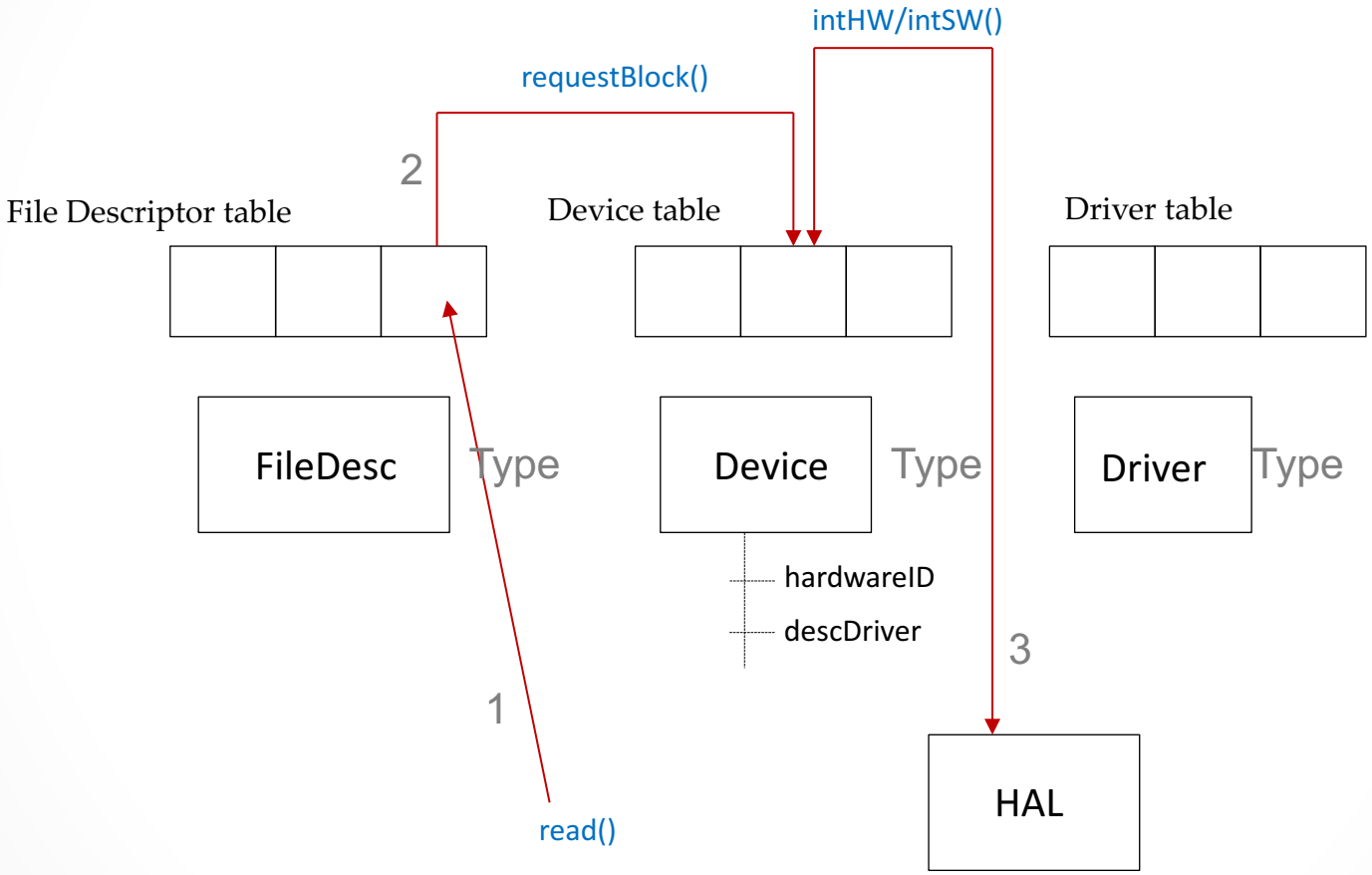
Write request. E.g. screen



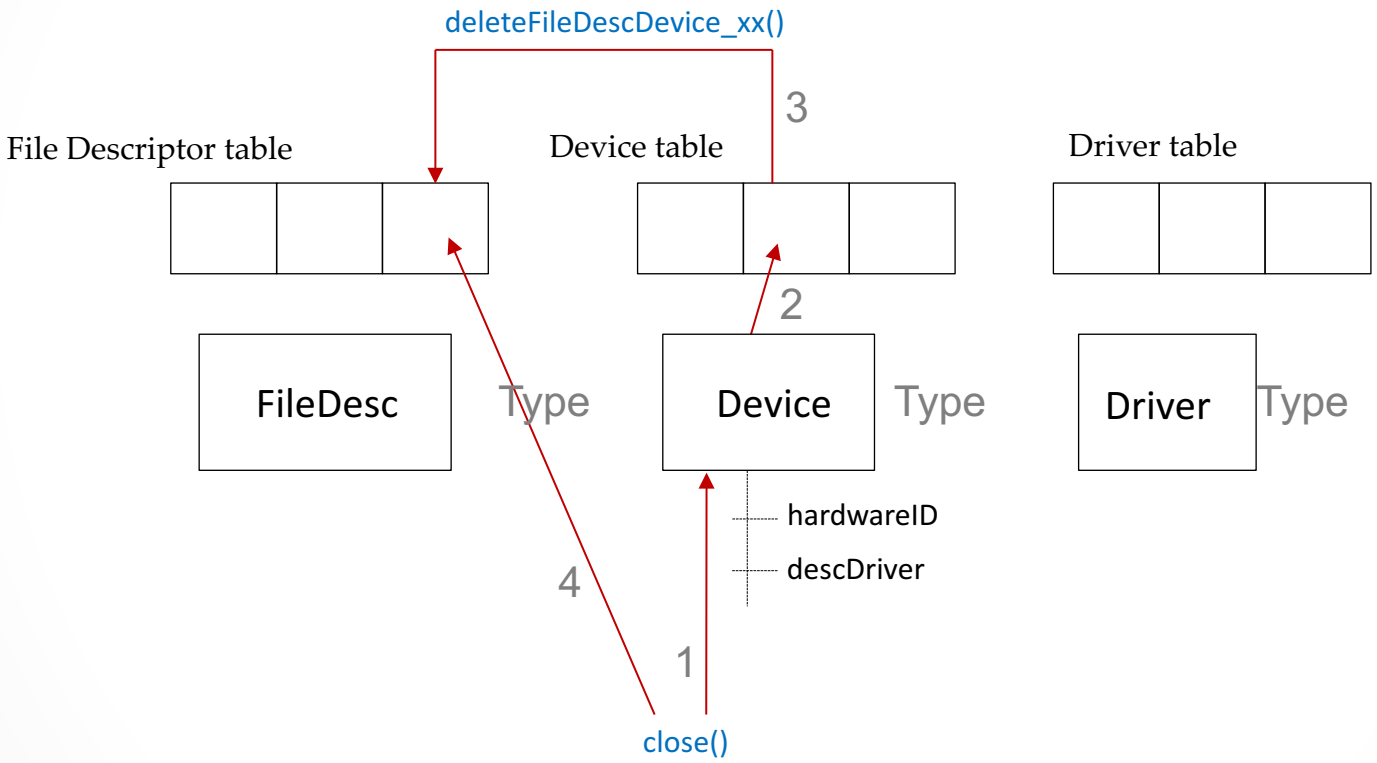
Read request. E.g. Keyboard



Read request. E.g. CD-ROM



Closing the device



Contents

- Driver Structure
- Driver Operation
- Driver scheduling example: disk scheduling

Overview of Mass Storage Structure

- **Magnetic disks** provide bulk of secondary storage of modern computers
  - Drives rotate at 60 to 250 times per second
  - **Transfer rate** is rate at which data flow between drive and computer
  - **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
  - **Head crash** results from disk head making contact with the disk surface
    - That's bad
- Disks can be removable
- Drive attached to computer via **I/O bus**
  - Busses vary, including **EIDE**, **ATA**, **SATA**, **USB**, **Fibre Channel**, **SCSI**, **SAS**, **Firewire**
  - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

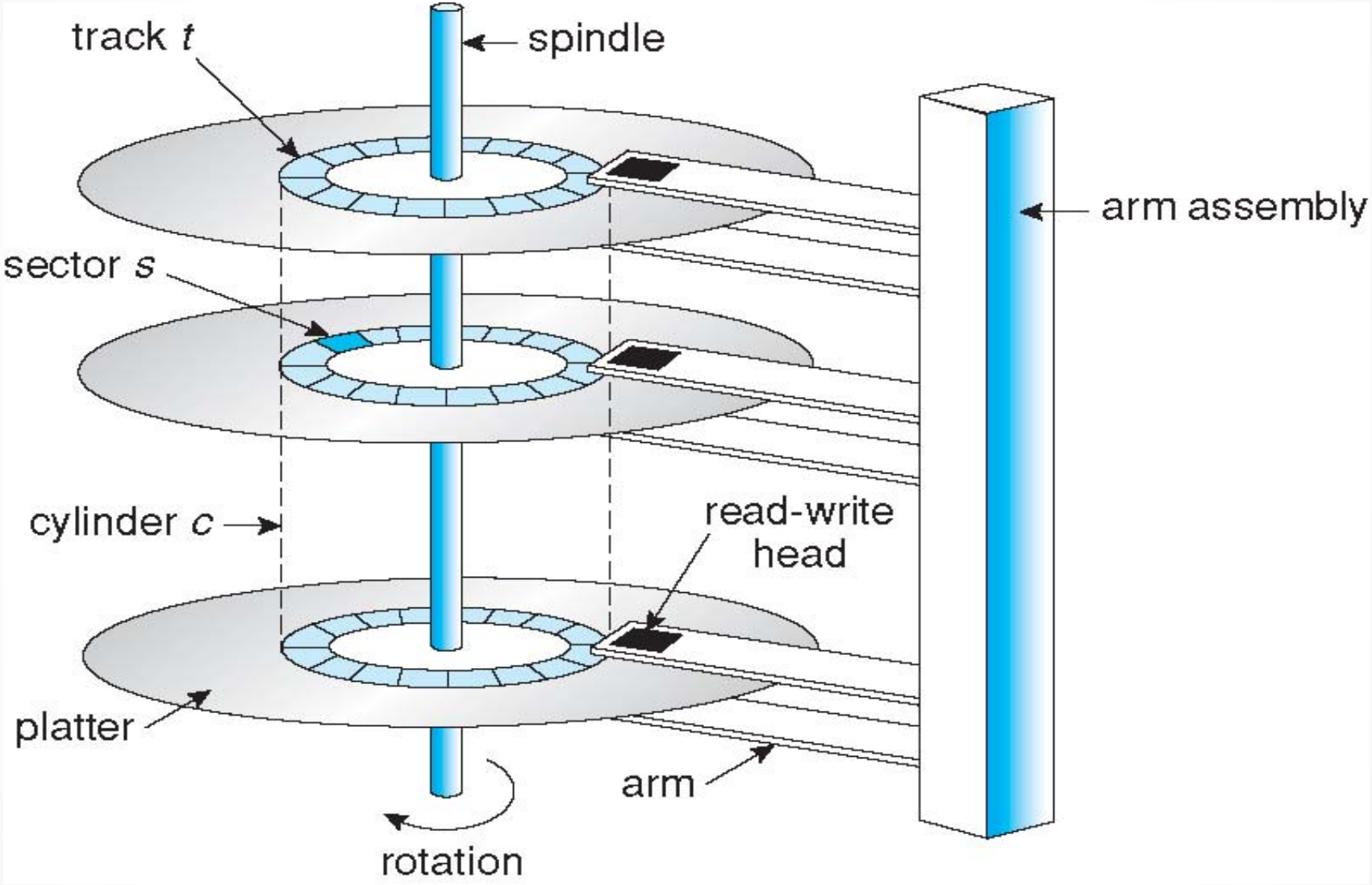
Magnetic Disks

- Platters range from .85" to 14" (historically)
  - Current: 3.5" and 2.5"
- Largest developed 12 Tbytes (2017)
- Performance
  - Transfer Rate – theoretical – 6 Gb/sec
  - Effective Transfer Rate – real – 1Gb/sec
  - Seek time from 3ms to 12ms – 9ms common for desktop drives
  - Average seek time measured or calculated based on 1/3 of tracks
  - Rotational latency based on spindle speed
    - 1/(RPM / 60)
  - Average latency = $\frac{1}{2}$ latency

Spindle [rpm]	Average latency [ms]
4200	7.14
5400	5.56
7200	4.17
10000	3
15000	2

(From Wikipedia)

Moving-head Disk Mechanism



Magnetic Disk Performance

- **Access Latency** = **Average access time** = average seek time + average rotational latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead

- For example to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =
  - 5ms + (1/(7200/60))/2*1000ms + 4KB / 1Gb/sec + 0.1ms =
  - 5ms + 4.17ms + 9.27ms + 4 / 131072 sec =
  - 9.27ms + .12ms = 9.39ms

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer

- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost
  - Logical to physical address should be easy
    - Except for bad sectors
    - Non-constant # of sectors per track via constant angular velocity

Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time ≈ seek distance
- Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Disk Scheduling (Cont.)

- There are many sources of disk I/O request
  - OS
  - System processes
  - Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists
- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying "depth")

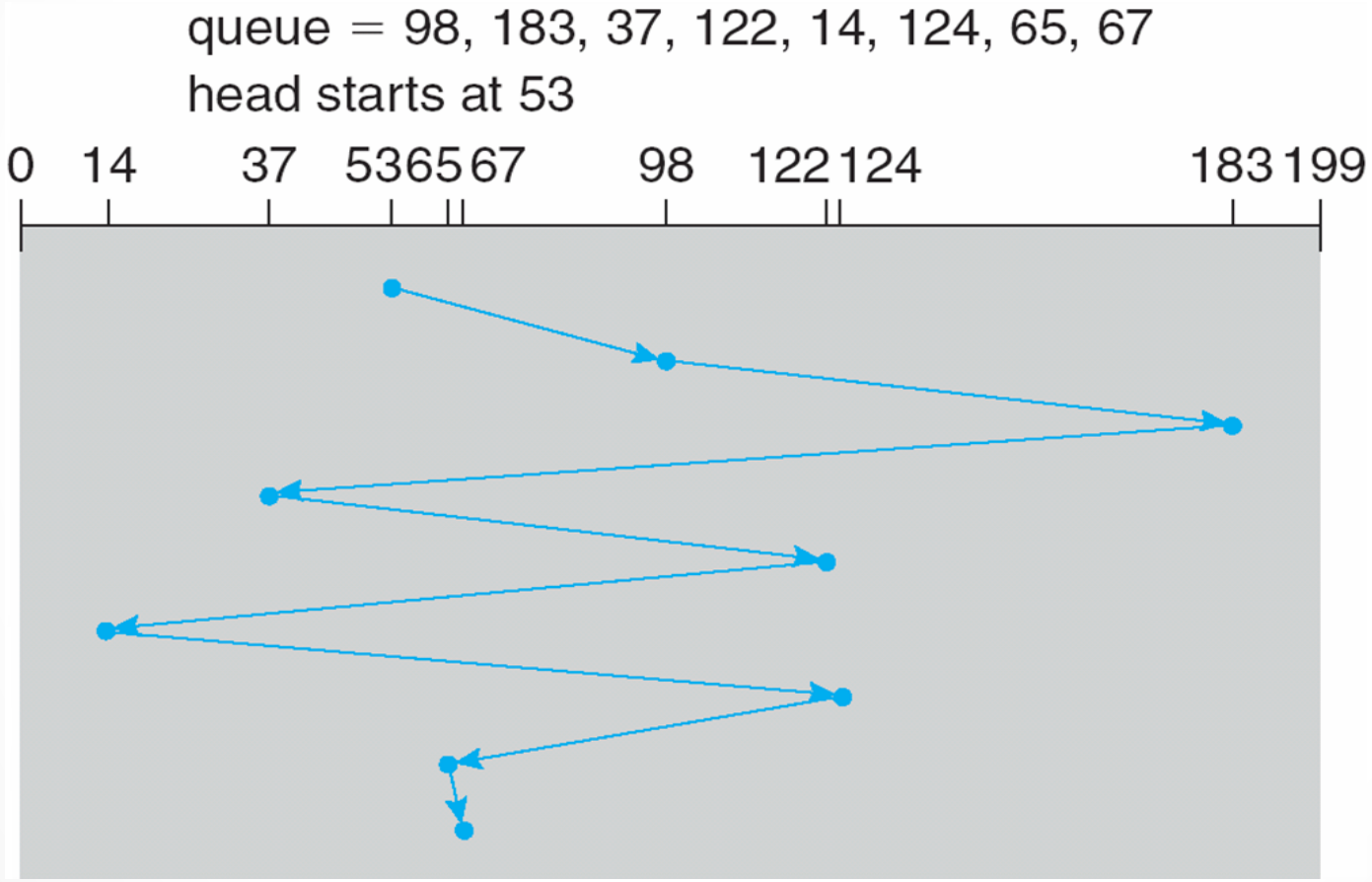
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

FCFS

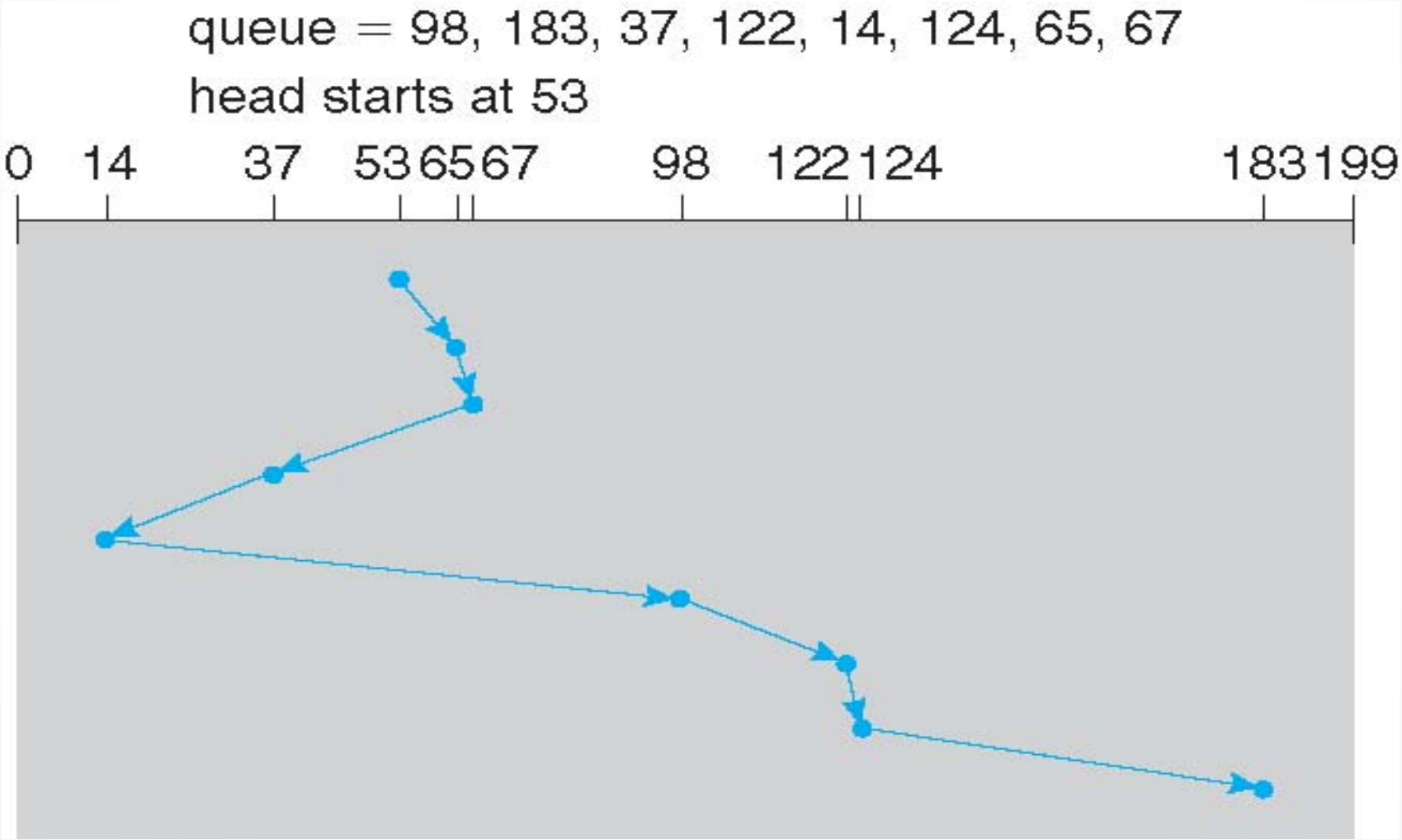
Illustration shows total head movement of 200 cylinders



SSTF

- Shortest Seek Time First selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Illustration shows total head movement of 236 cylinders

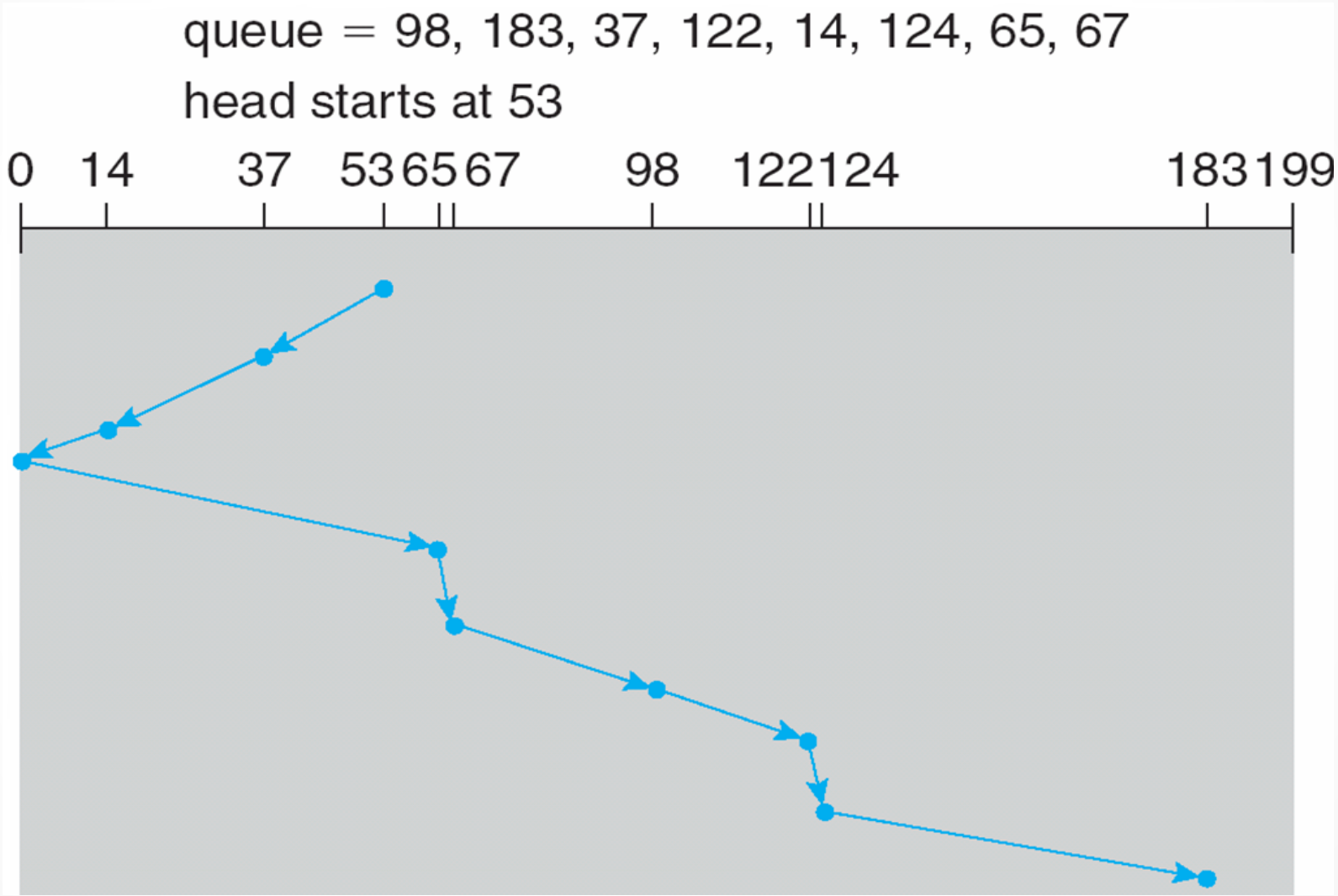
SSTF (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- **SCAN algorithm** Sometimes called the **elevator algorithm**
- Illustration shows total head movement of 208 cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

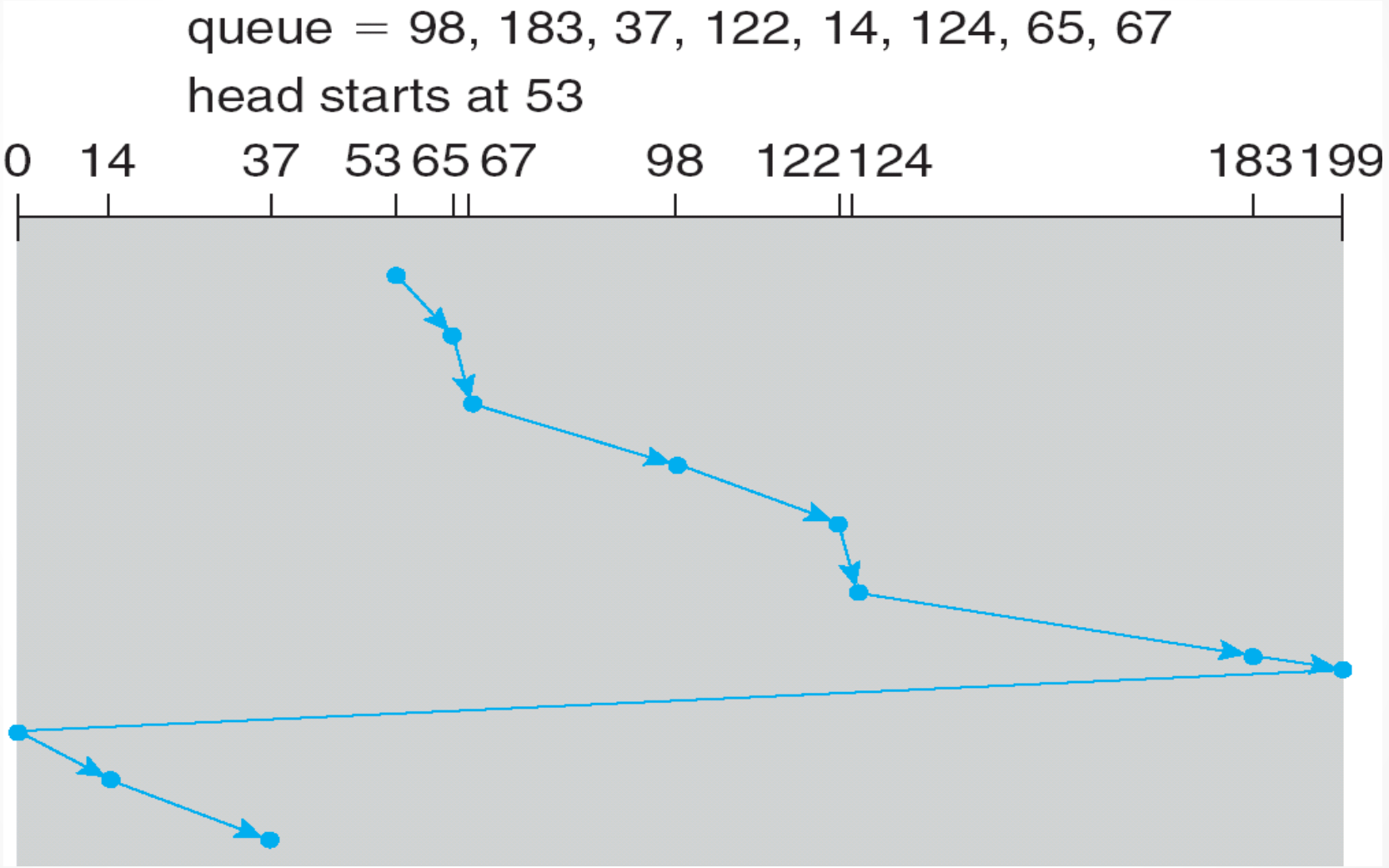
SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-SCAN

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
  - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one
- Total number of cylinders?

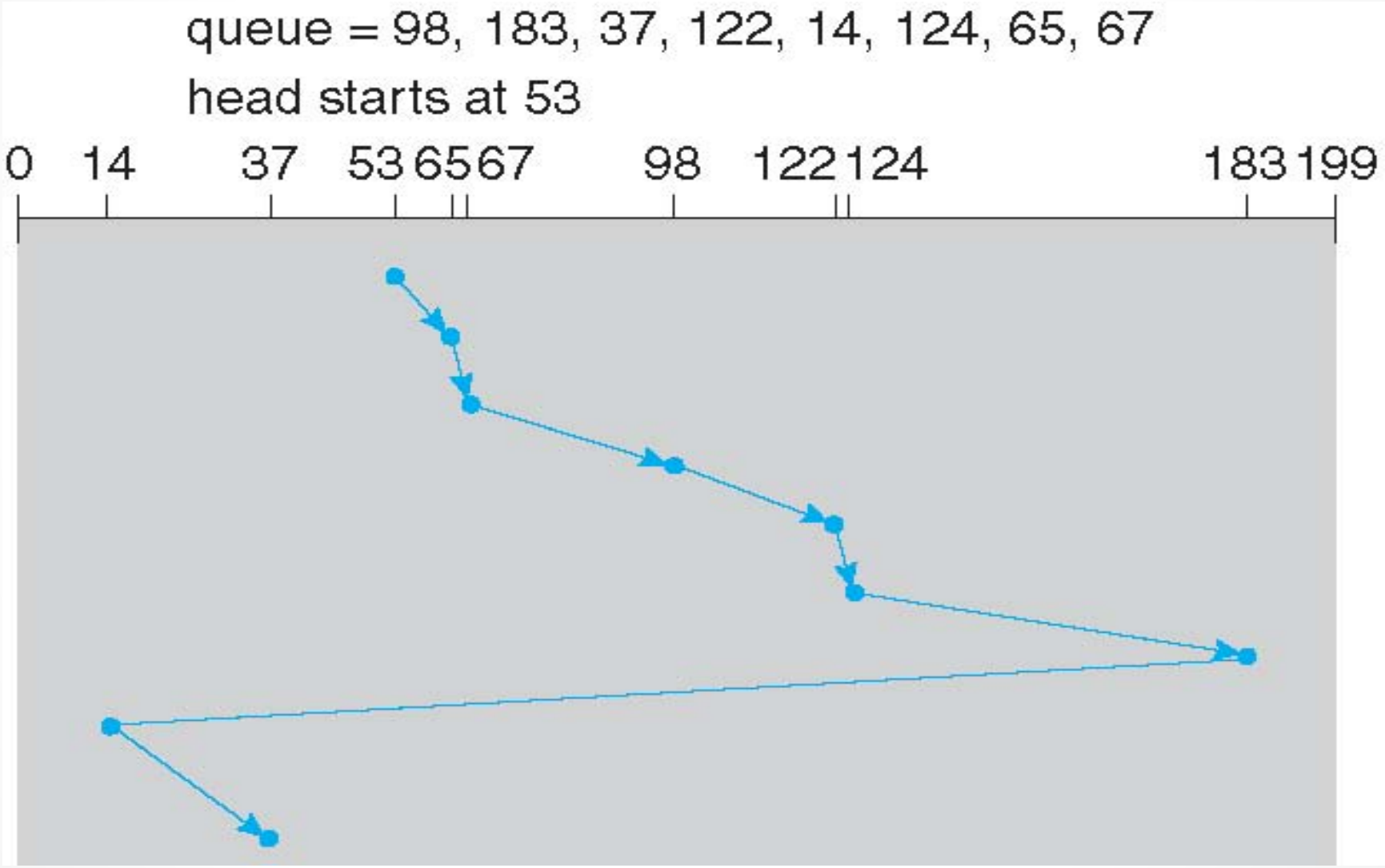
C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-LOOK

- LOOK a version of SCAN, C-LOOK a version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk
- Total number of cylinders?

C-LOOK (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
  - And metadata layout
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for the default algorithm

- What about rotational latency?
  - Difficult for OS to calculate

- How does disk-based queuing effect OS queue ordering efforts?

Grupo ARCOS

Operating Systems Design
Grado en Ingeniería Informática
Universidad Carlos III de Madrid