

LINUX OPERATING SYSTEM

YANG

LINUX操作系统（双语）





双语课→课件内容中英混排

|Lecture 16

Mass Storage



本讲内容

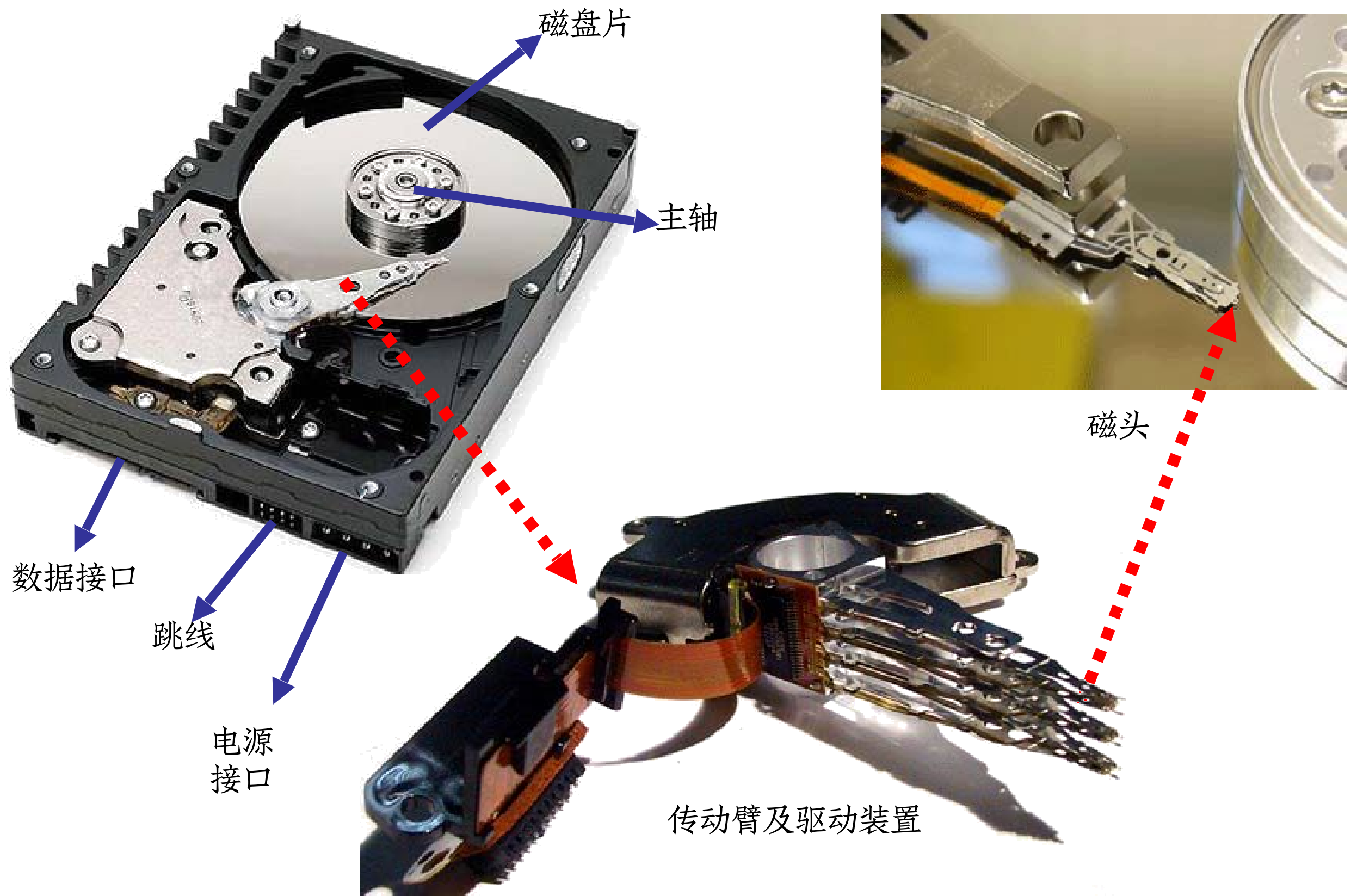
 磁盘结构

 磁盘性能

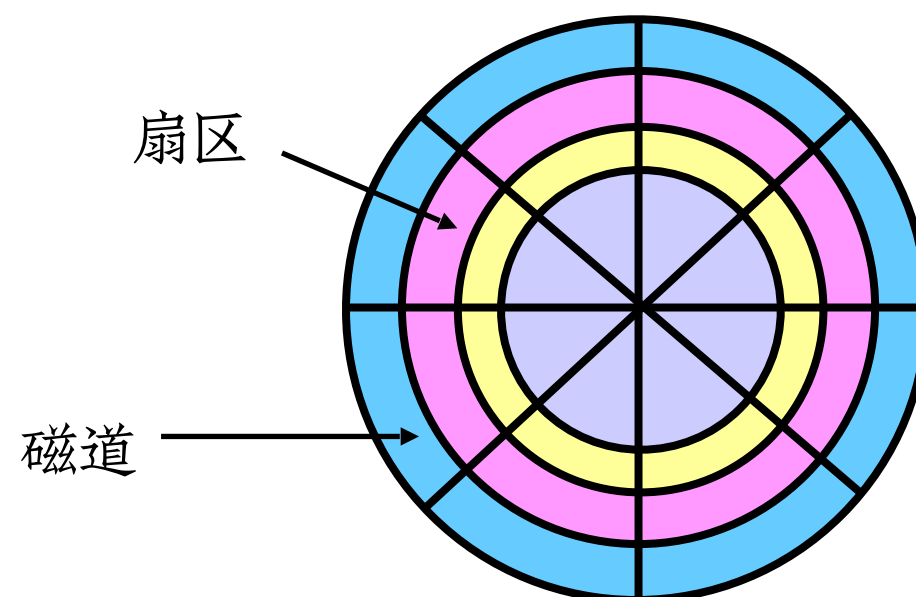
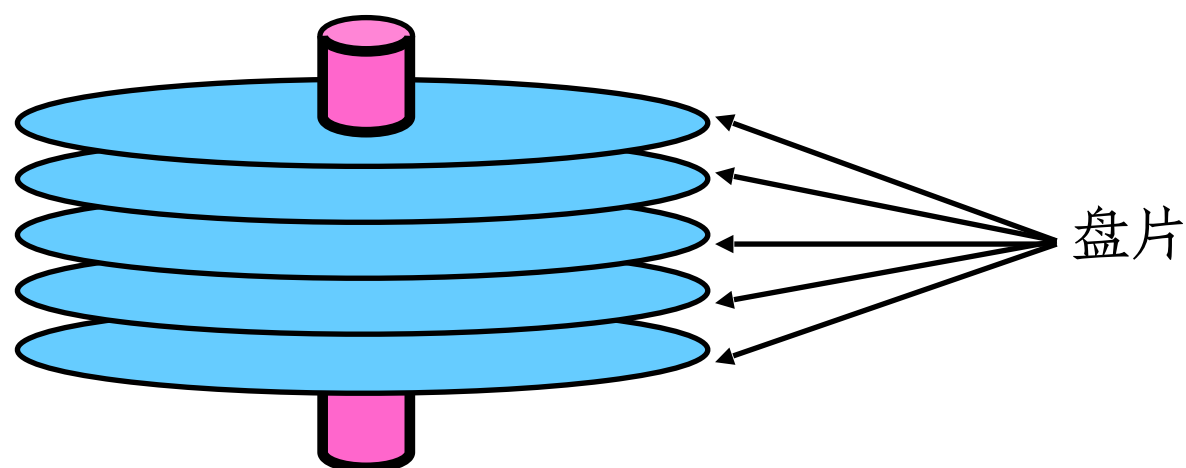
 磁盘调度

磁盘结构

MAGNETIC DISK

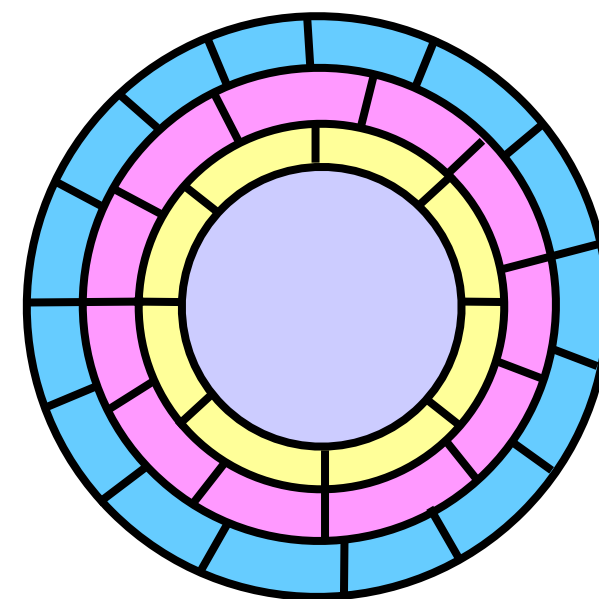


磁盘结构



每条磁道上的扇区数相等


- 磁道：能被磁头访问的一组同心圆
- 扇区：磁道上的区域，数据存放的基本单位
- 柱面：所有盘片同一磁头下的磁道集合
- 恒角速度CAV
 - 不同磁道密度不同，但转速恒定
- 恒线速度CLV
 - 磁道密度相同，但转速不断变化



磁盘格式化

低级格式化 (Low-level formatting)


Physical formatting

 为每个扇区使用特殊的数据结构进行填充，包括一个头部、数据区域和一个尾部。

 头部和尾部包含一些控制信息，如扇区号、ECC码等。

高级格式化 (High-level formatting)

Logical formatting

 构建文件系统，在磁盘上初始化文件系统数据结构，如空闲和已分配空间表、一个空目录等。

磁盘性能

磁盘性能指标

🧠 查找一个物理块的顺序：柱面号、磁头号 and 扇区号

🧠 寻道时间 T_s ：将磁头定位到正确磁道(柱面)上所花的时间，与盘片直径和传动臂速度相关，平均20ms。

🧠 旋转延迟 T_r ：所查找的扇区转到磁头下所用的时间，与磁盘的旋转速度有关，一个10,000 r/m的磁盘平均旋转延迟为3ms。

🧠 传送时间 T ：传送扇区内的数据的时间，同样取决于磁盘的旋转速度， $T = b/(rN)$ (b 为要传送的字节数， N 为一个磁道中的字节数， r 为转速)

🧠 总的平均存取时间 $T_a = T_s + T_r + T$

实例分析：磁盘性能

💡 条件假设:

💡 平均寻道时间为 5 ms, 平均旋转延迟为 4 ms

💡 传输速率为 4 MByte/s, 扇区大小是 1 KByte

💡 如果随机访问一个扇区

💡 $T_a = 5\text{ms} + 4\text{ms} + 0.25\text{ms} \approx 10\text{ ms}$

💡 总的存取速率为 100 KByte/sec

💡 如果要访问的扇区在同一个柱面

💡 $T_a = 4\text{ms} + 0.25\text{ms} \approx 5\text{ ms} \rightarrow 200\text{ KByte/sec}$

💡 如果下一个要访问的扇区正好和上次访问的扇区相邻

💡 $T_a = 0.25\text{ms} \rightarrow 4\text{ MByte/sec}$

磁盘调度

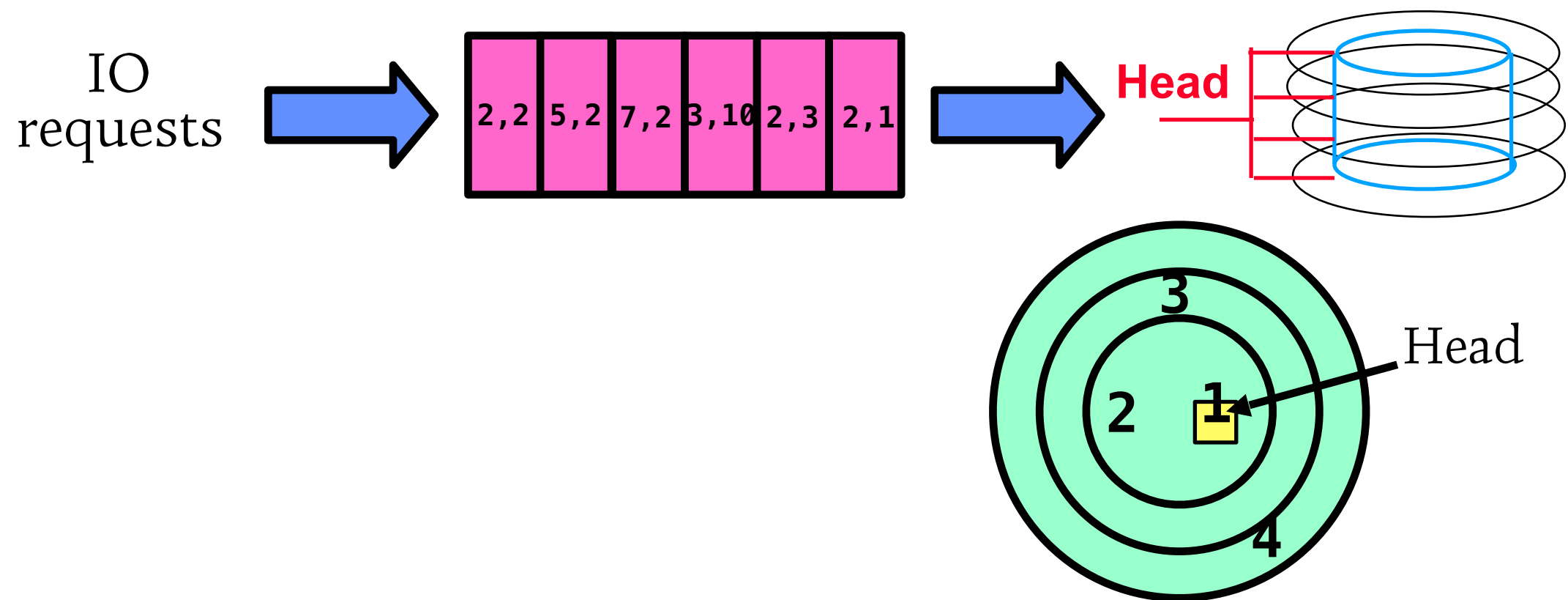
DISK I/O REQUEST

- 🧠 Whenever a process needs I/O to or from the disk, it issues a system call to the operating system. The request specifies several pieces of information:
 - 🧠 Whether this operation is input or output
 - 🧠 What the disk address for the transfer is
 - 🧠 What the memory address for the transfer is
 - 🧠 What the number of sectors to be transferred is

DISK SCHEDULING

- 💡 For a multiprogramming system with many processes, the disk queue may often have several pending requests. Thus, when one request is completed, the operating system chooses which pending request to service next.

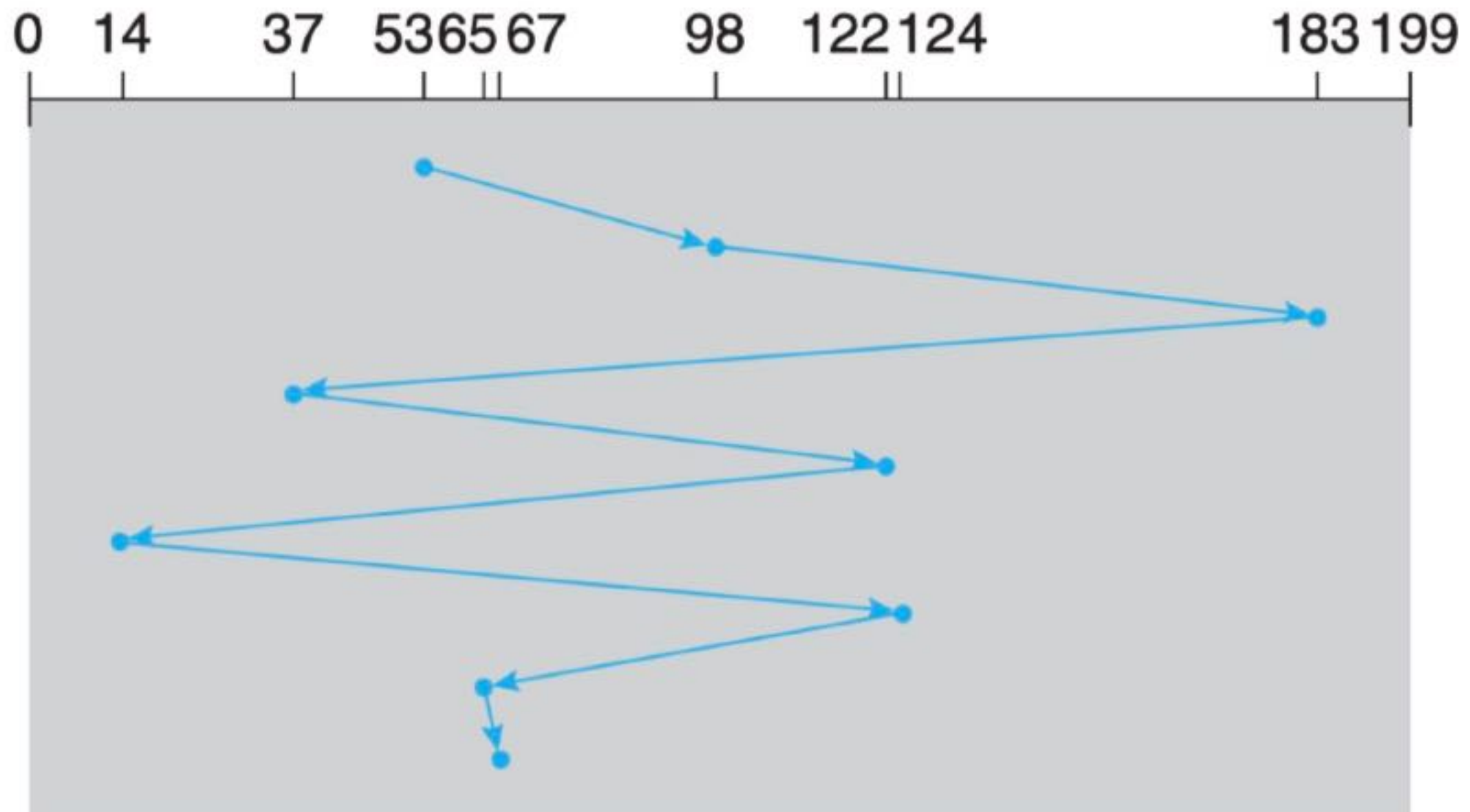
How does the operating system make this choice?



FCFS SCHEDULING

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

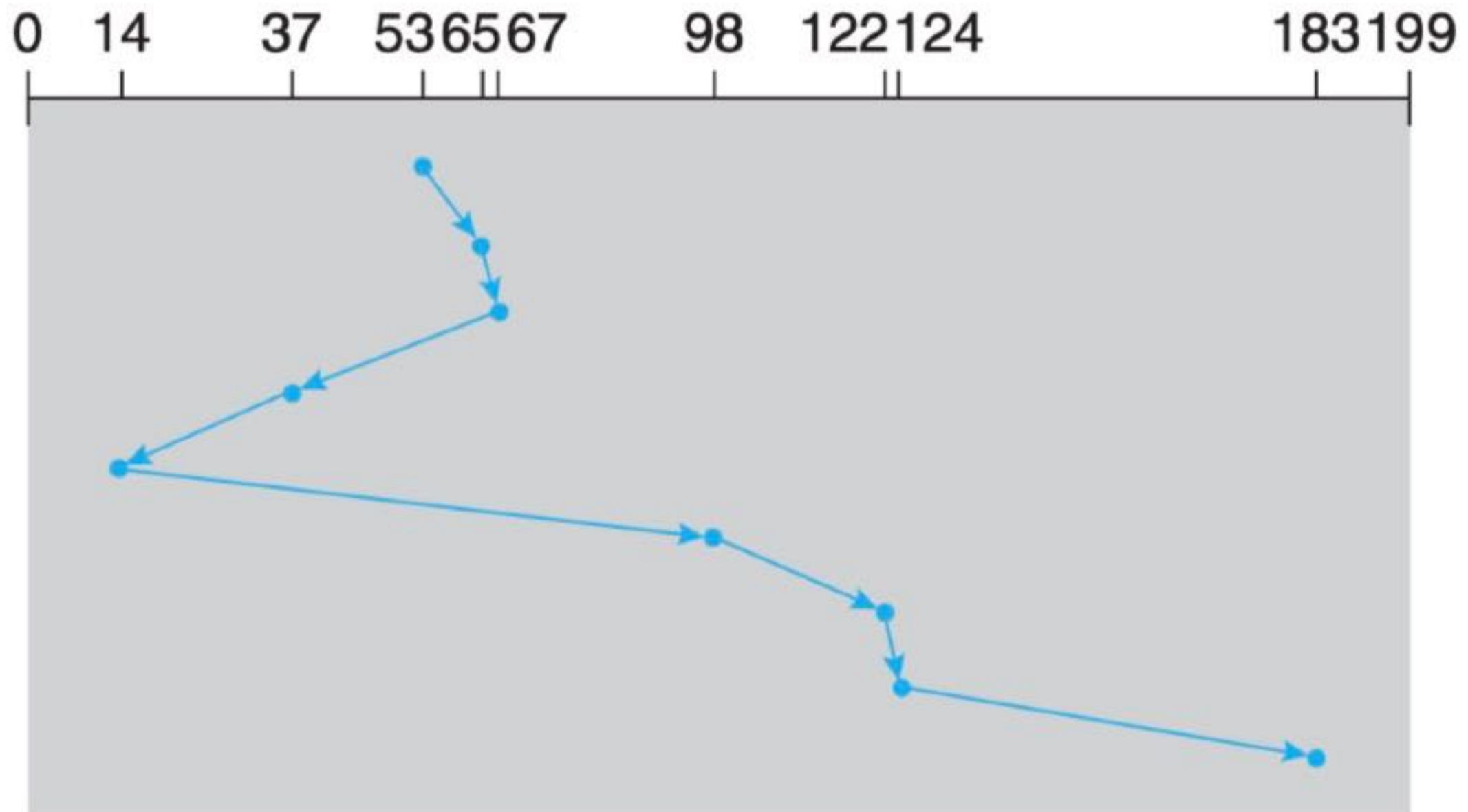
head starts at 53



SSFT SCHEDULING

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

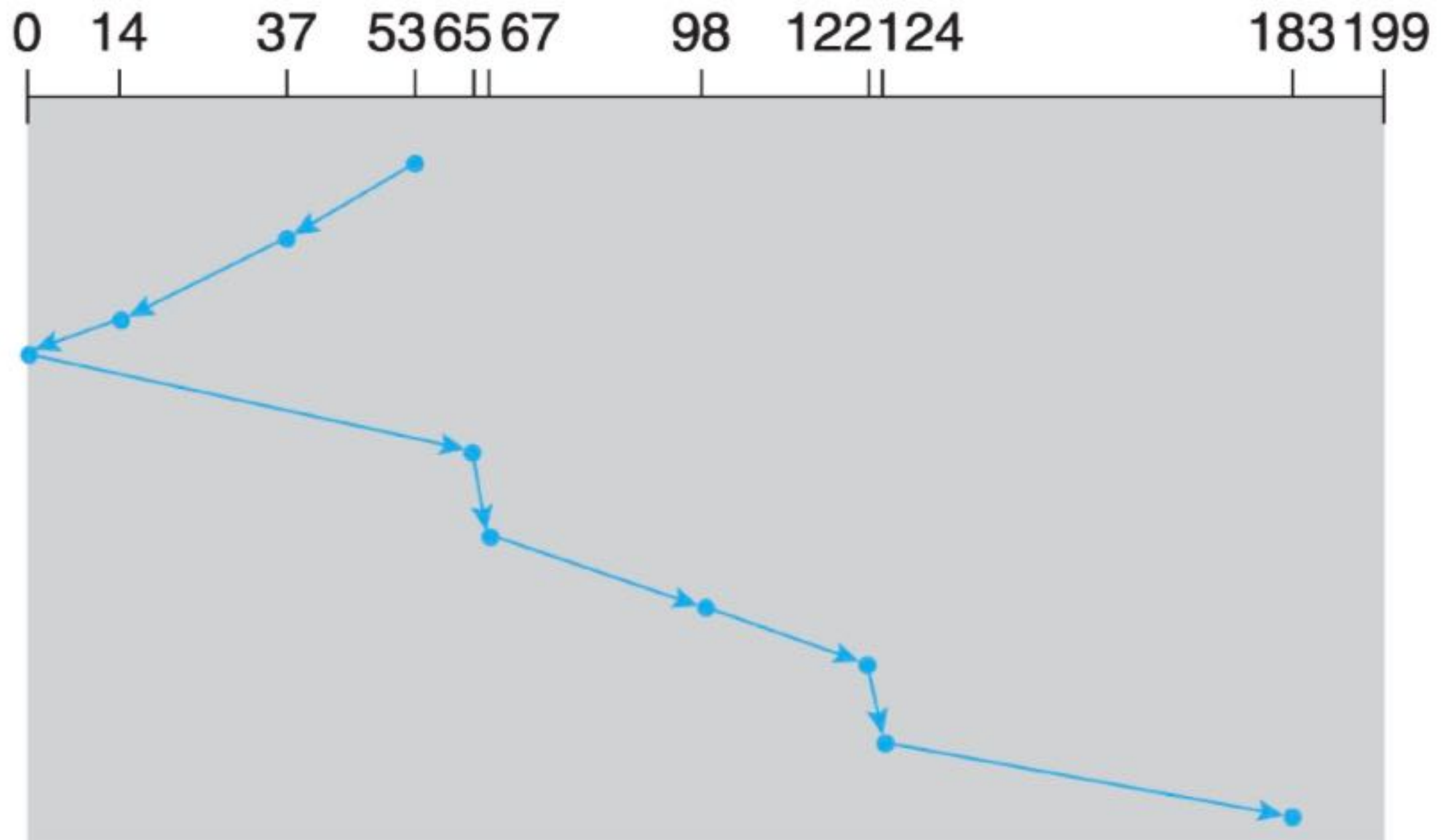
head starts at 53



SCAN SCHEDULING

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

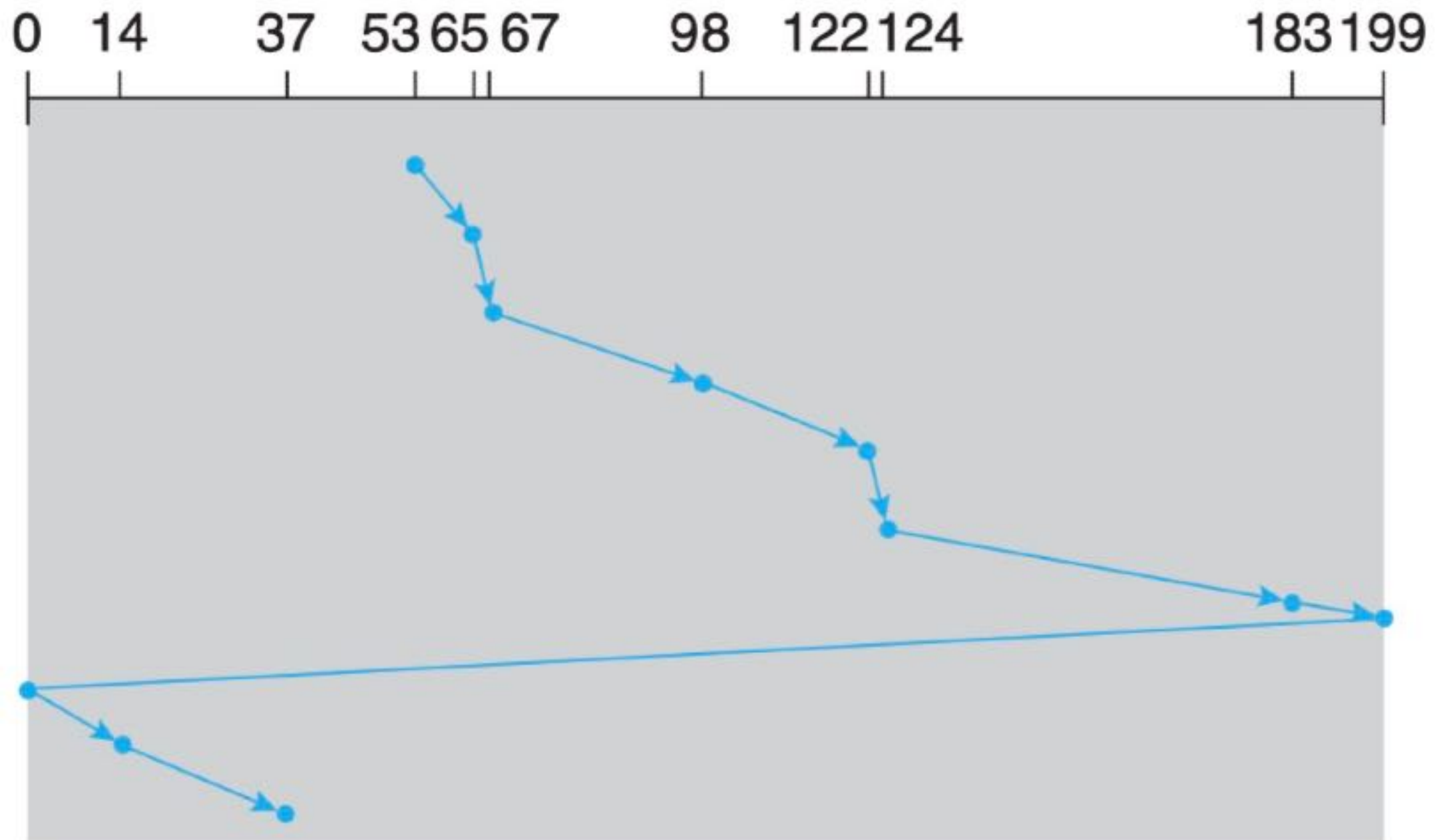
head starts at 53



C-SCAN SCHEDULING

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

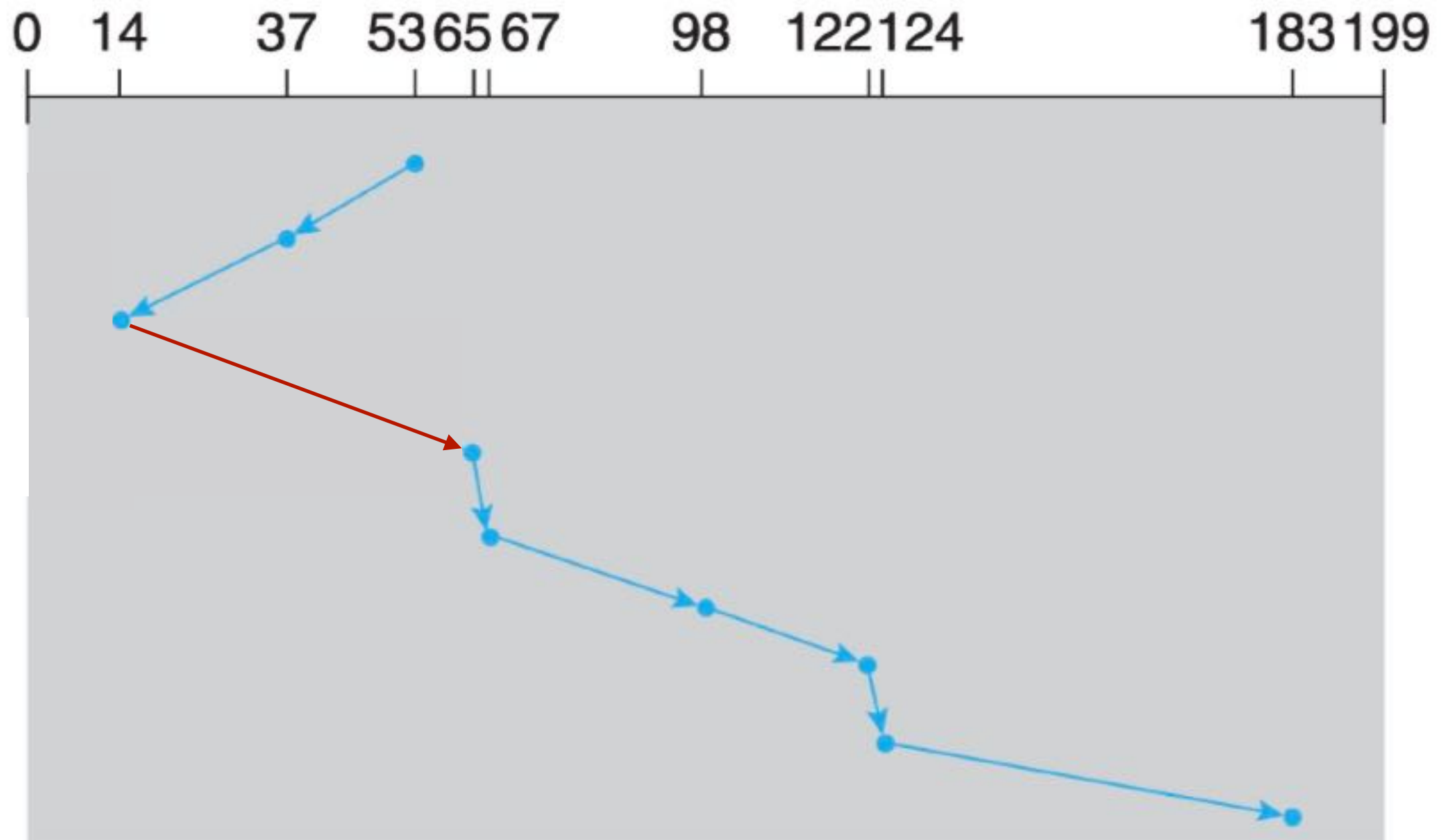
head starts at 53



LOOK SCHEDULING

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

head starts at 53



SELECTION OF A ALGORITHM

- 💡 FCFS is the simplest.
- 💡 SSTF is common and has a natural appeal but it may cause a starvation problem.
- 💡 SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
- 💡 How to know which algorithm is chosen by Linux?

```
root@youngyt-PC:/# cat /sys/block/sda/queue/scheduler  
[noop] deadline cfq
```

LINUX I/O SCHEDULER

- 🧠 **noop**: it performs FCFS policy which is good enough for SSD.
- 🧠 **deadline**: it works by creating two queues: a read queue and a write queue. Each I/O request has a time stamp associated that is used by the kernel for an expiration time. When an I/O request reaches its deadline, it is pushed to the highest priority.
- 🧠 **cfq**: Complete Fairness Queueing works by creating a per-process I/O queue. The goal of this I/O scheduler is to provide a fair I/O priority to each process. While the CFQ algorithm is complex, the gist of this scheduler is that after ordering the queues to reduce disk seeking, it services these per-process I/O queues in a round-robin fashion.

|Lecture 16

The End



下期预告

 下次直播时间：4月2日 上午9:30

 课程内容

 Lecture 17 File System

 Q&A