Mass Storage Structure

Hard Disk Drives

Liu yufeng

Fx_yfliu@163.com

Hunan University

I/O Devices

- I/O is **critical** to computer system to **interact with systems.**
- Issue :
  - How should I/O be integrated into systems?
  - What are the general mechanisms?
  - How can we make the efficiently?

I/O Devices

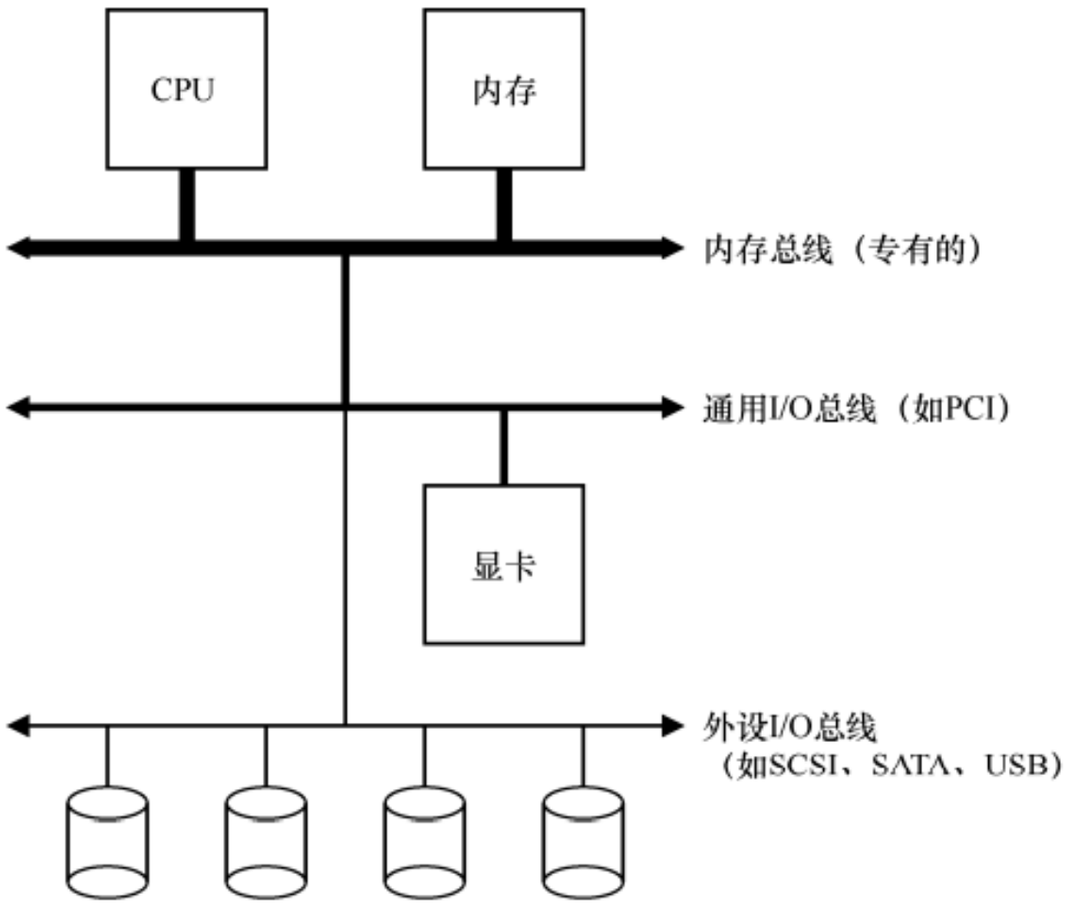


图 36.1　原型系统架构

CPU 通过某种内存总线（memory bus）或互连电缆连接到系统内存。
图像或者其他高性能 I/O 设备通过常规的 I/O 总线（I/O bus）连接到系统，在许多现代系统中会是 PCI 或它的衍生形式。
最下 面是外围总线（peripheral bus），比如 SCSI、SATA 或者 USB。它们将最慢的设备连接到系统，包括磁盘、鼠标及其他类似设备

越快的总线越短，因此高性能的内存总线没有足够的空间连接太多设备。另外，在工程上高性能总线的造价非常高。所以，系统的设计采用了这种分层的方式，这样可以让要求高性能的设备（比如显卡）离 CPU 更近一些，低性能的设备离 CPU 远一些。
将磁盘和其他低速设备连到外围总线的好处很多，其中较为突出的好处就是你可以在外围总线上连接大量的设备

A Canonical Device

图 36.2　标准设备

一个（简化的）设备接口包含 3 个寄存器：一个状态（status）寄存器，可以读取并查看设备的当前状态；一个命令（command）寄存器，用于通知设备执行某个具体任务；一个数据（data）寄存器，将数据传给设备或从设备接收数据。通过读写这些寄存器，操作系统可以控制设备的行为

The Canonical Protocol

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (Doing so starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

第 1 步，操作系统通过反复读取状态寄存器，等待设备进入可以接收命令的就绪状态。我们称之为轮询（polling）设备（基本上，就是问它正在做什么）。
第 2 步，操作系统下发数据到数据寄存器。例如，你可以想象如果这是一个磁盘，需要多次写 入操作，将一个磁盘块（比如 4KB）传递给设备。如果主 CPU 参与数据移动（就像这个示 例协议一样），我们就称之为编程的 I/O（programmed I/O，PIO）。
第 3 步，操作系统将命令写入命令寄存器；这样设备就知道数据已经准备好了，它应该开始执行命令。最后一步， 操作系统再次通过不断轮询设备，等待并判断设备是否执行完成命令（有可能得到一个指示成功或失败的错误码）。

Polling

- Operating system waits until the device is ready by **repeatedly** reading the status register.
  - Positive aspect is simple and working.
  - **However, it wastes CPU time just waiting for the device**.
    - Switching to another ready process is better utilizing the CPU.

进程在 CPU 上运行一段时间（对应 CPU 那一行上重复的 1），然后发出一个 读取数据的 I/O 请求给磁盘。如果没有中断，那么操作系统就会简单自旋，不断轮询设备状 态，直到设备完成 I/O 操作（对应其中的 p）。当设备完成请求的操作后，进程 1 又可以继续运行

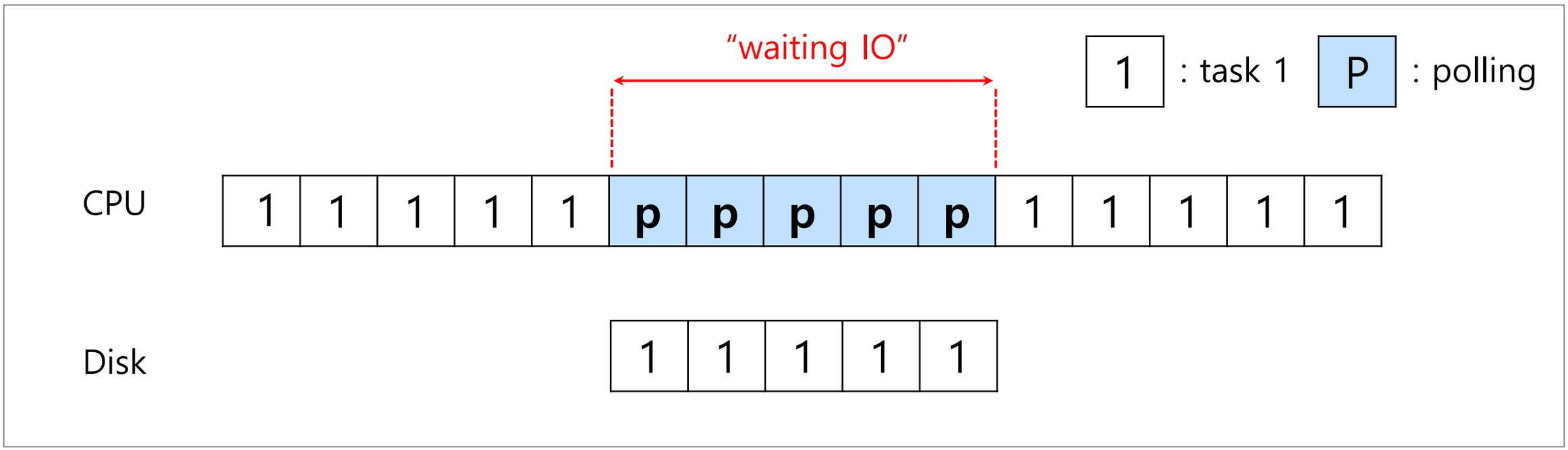


Diagram of CPU utilization by polling

Lowering CPU Overhead With Interrupts

- 有了中断后，CPU 不再需要不断轮询设备，而是向设备发出一个请求，然后就可以让对应进程睡眠，切换执行其他任务。

- 当设备完成了自身操作，会抛出一个硬件中断，引发 CPU 跳转执行操作系统预先定义好的中断处理程序.

interrupts

- 在磁盘处理进程 1 请求的同时，操作系统在 CPU 上运行进程 2。磁盘处理完成后，触发一个中断，然后操作系统唤醒进程 1 继续运行。

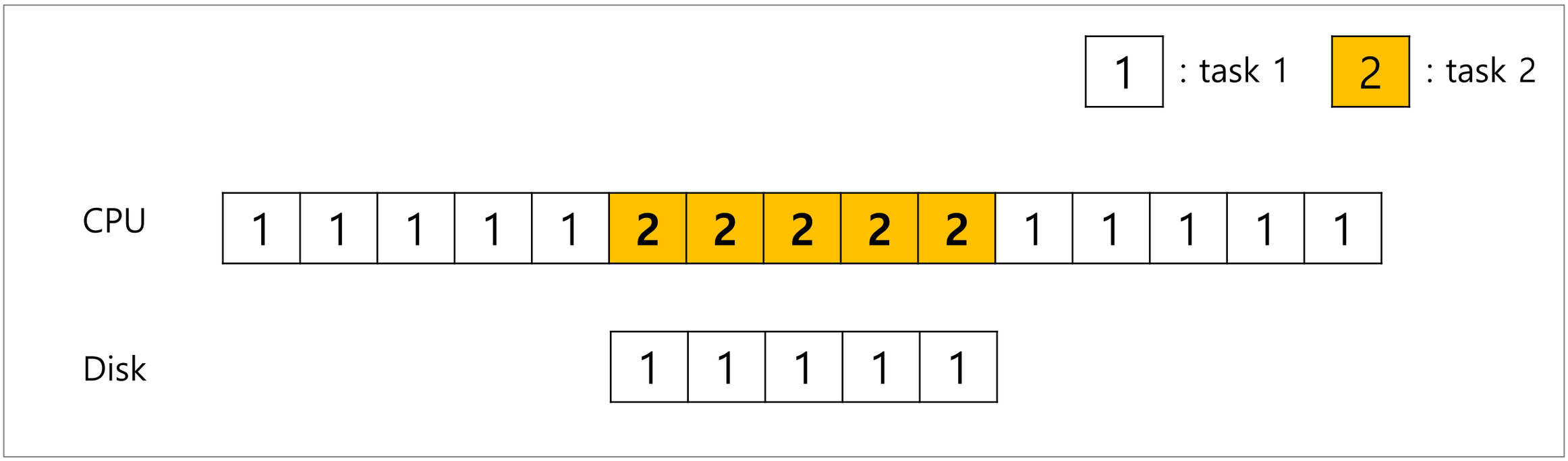


Diagram of CPU utilization by interrupt

Polling vs interrupts

- *However,* “interrupts is not always the best solution”
  - If, device performs very quickly, interrupt will “slow down” the system.
  - Because **context switch is expensive (switching to another process)**
  - 尽管中断可以做到计算与 I/O 的重叠，但这仅在慢速设备上有意义

If a device is fast → poll is best.
If it is slow → interrupts is better.

如果设备的速度未知，或者时快时慢，可以考虑使用混合（hybrid）策略，先尝试轮询一小段时间，如果设备没有完成操作，此时再使用中断。这种两阶段（two-phased）的办法可以实现两种方法的好处

CPU is once again over-burdened

- CPU **wastes a lot of time** to copy the *a large chunk of data* from memory to the device.
- 进程 1 在运行过程中需要向磁盘写一些数据，所以它开始进行 I/O 操作，将数据从内存拷贝到磁盘（其中标示 c 的过程）。拷贝结束后，磁盘上的 I/O 操作开始执行，此时 CPU 才可以处理其他请求。

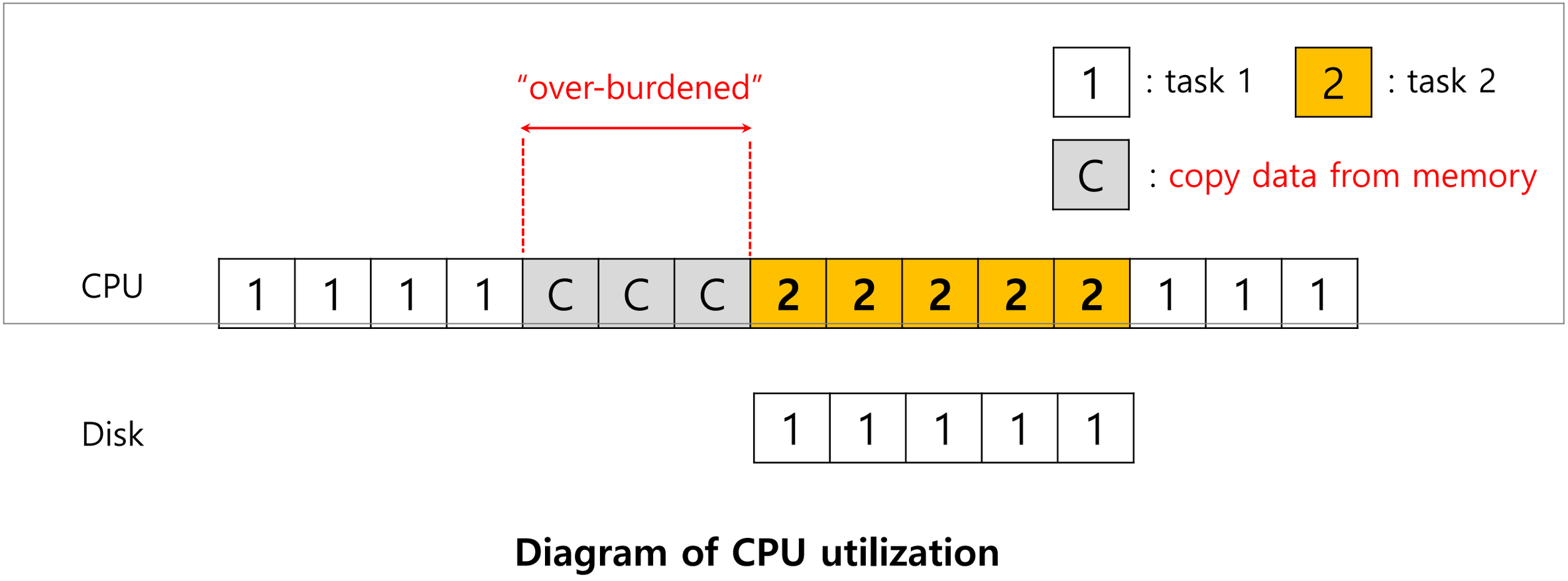


Diagram of CPU utilization

Efficient Data Movement With DMA

- 如果使用编程的 I/O 将一大块数据传给设备，CPU 又会因为琐碎的任务而变得负载很重，浪费了时间和算力。

- 使用 DMA（Direct Memory Access）可以协调完成内存和设备间的数据传递，不需要 CPU 介入。

DMA (Direct Memory Access)

- **Copy data** in memory by knowing “where the data lives in memory, how much data to copy”
- When completed, DMA raises an interrupt, I/O begins on Disk.

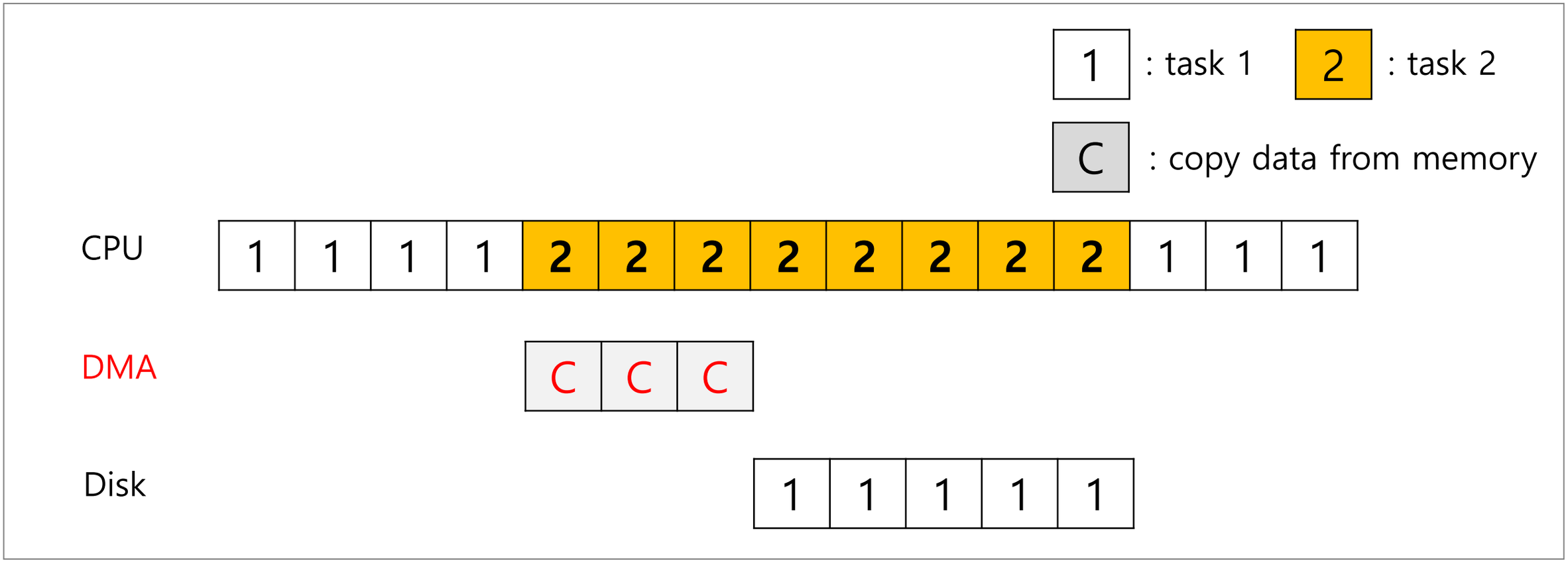
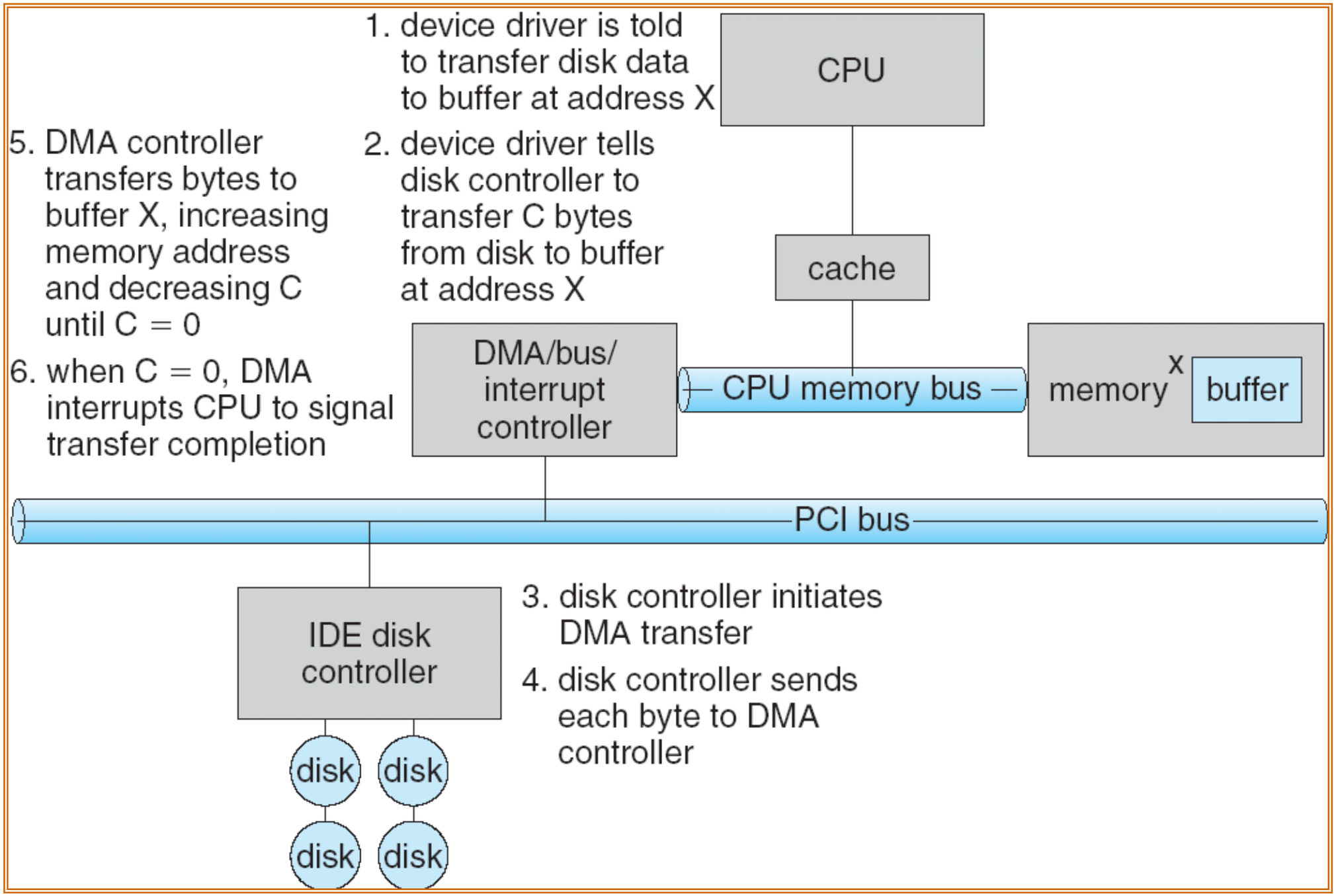


Diagram of CPU utilization by DMA

直接内存访问（DMA）


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

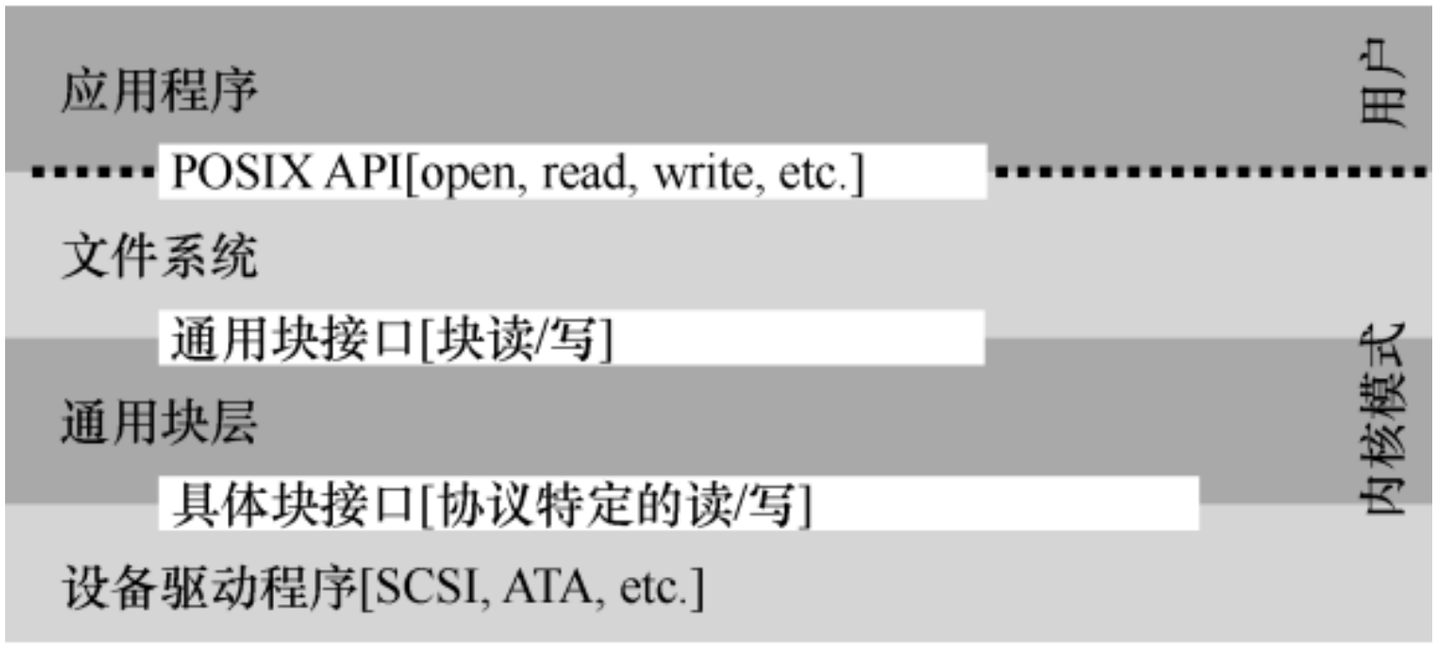
Methods Of Device Interaction

- The first, oldest method is to have **explicit** I/O instructions(直接I/O指令). For example, on x86, the **in** and **out** instructions can be used to communicate with devices.

- The second method to interact with devices is known as **memory-mapped I/O(内存映射指令)**.

- 硬件将设备寄存器作为内存地址提供。当需要访问设备寄存器时，操作系统装载（读取）或者存入（写入）到该内存地址；然后硬件会将装载/存入转移到设备上，而不是物理内存。

The Device Driver



文件系统栈

文件系统（包括在其之上的应用程序）完全不清楚它使用的是什么类型的磁盘。它只需要简单地向通用块设备层发送读 写请求即可，块设备层会将这些请求发送给对应的设备驱动，然后设备驱动来完成真正的 底层操作。

所有需要插入系统的设备都需要安装对应的驱动程序，所以久而久 之，驱动程序的代码在整个内核代码中的占比越来越大

A Simple IDE Disk Driver

- IDE 硬盘包含 4 种类型的寄存器，即控制、命令块、 状态和错误
- IO指令： in and out

案例研究：简单的 IDE 磁盘驱动程序（自学、了解）

Hard Disk Drives

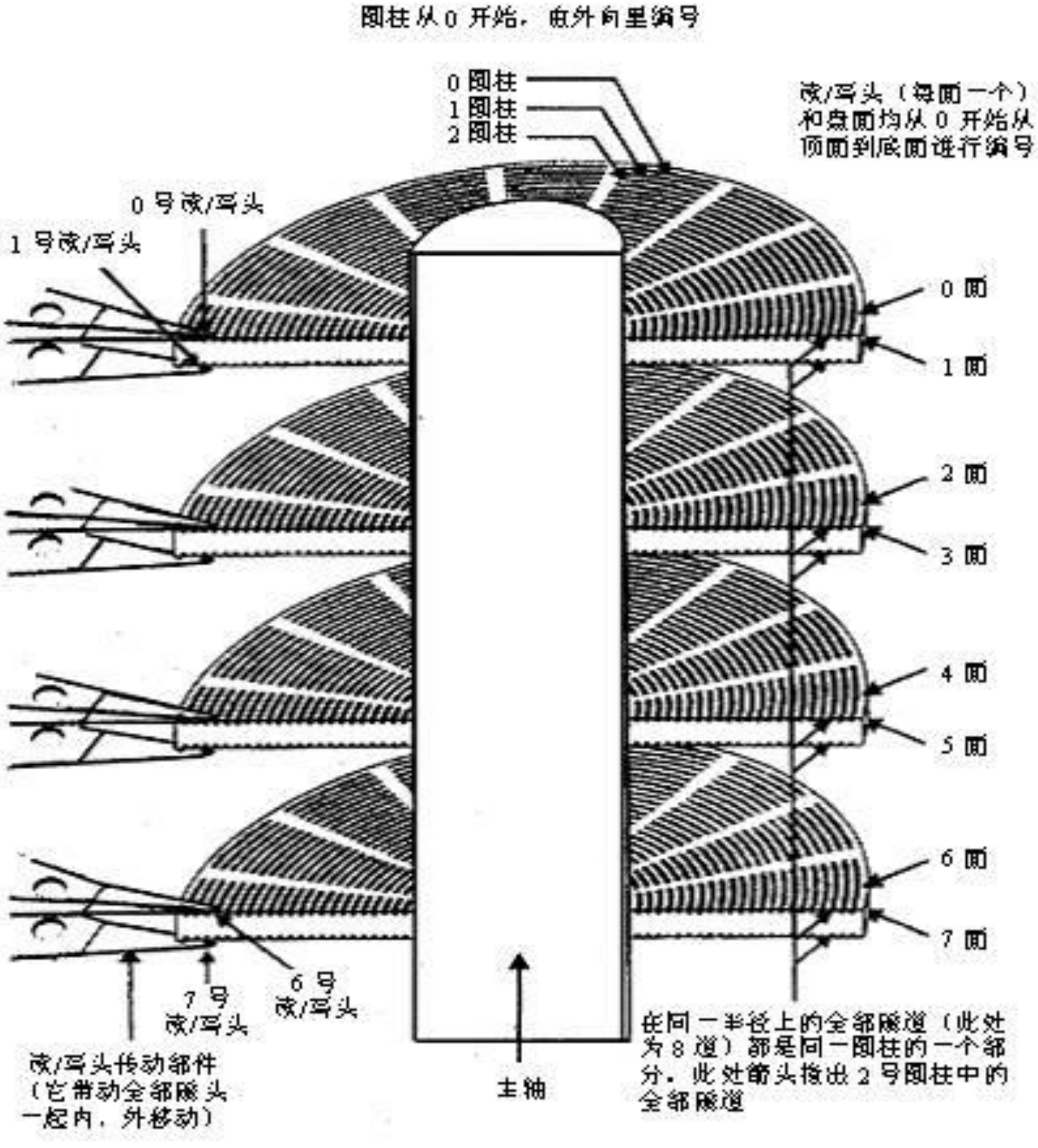
- The **hard disk drive** have been the main form of persistent data storage in computer systems for decades.

- Thus, it is worth understanding the details of a disk’s operation before building the file system software that manages it.


磁盘盘片
读写磁头
主轴
传动手臂
传动轴
反力矩弹簧装置

The Interface

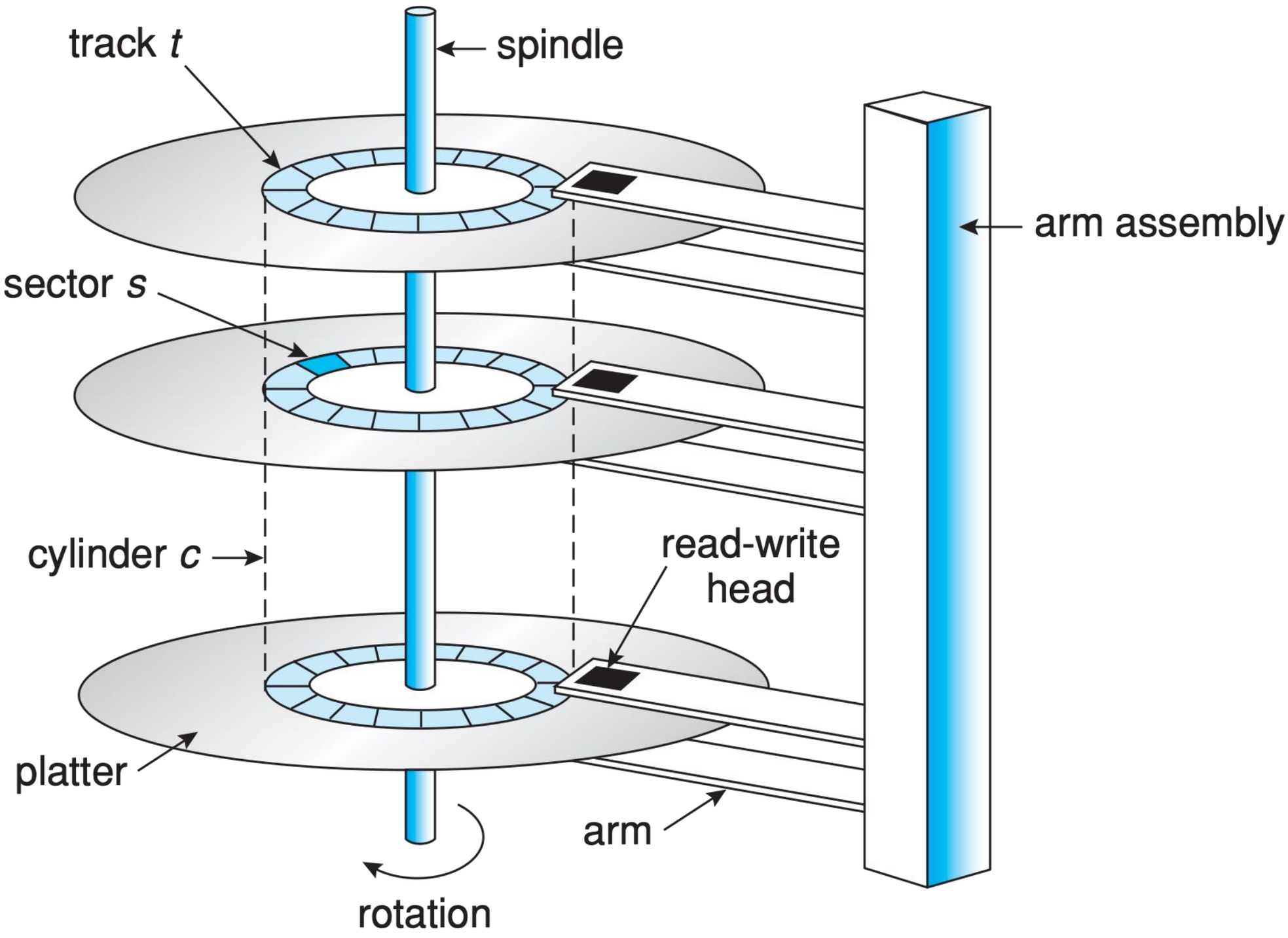
- 驱动器由大量扇区（512 字节块）组成，每个扇区都可以读取或写入。在具有 n 个扇区的磁盘 上，扇区从 0 到 n−1 编号。因此，我们可以将磁盘视为一组扇区，0 到 n−1 是驱动器的地址空间（address space）。
- 许多文件系统一次读取或写入 4KB（或更多数据）。但是，在更新磁盘时，驱动器制造商唯一保证的是单个 512 字节的写入是原子的（atomic，即它将 完整地完成或者根本不会完成）。



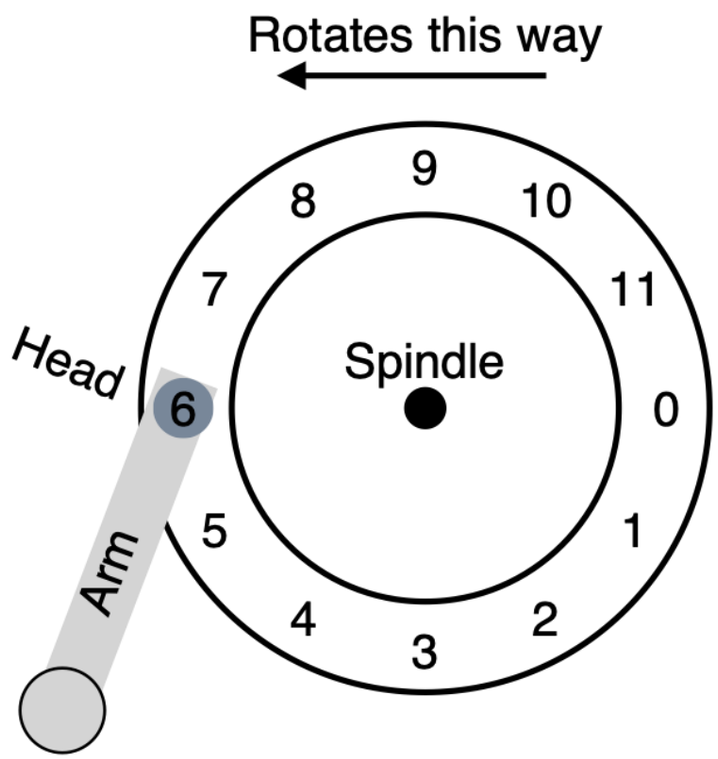
- 硬盘首先在逻辑上被划分为磁道、柱面以及扇区
- 磁盘在格式化时被划分成许多同心圆，这些同心圆轨迹叫做磁道。数百个磁道只有头发的宽度
- 磁道从外向内从0开始顺序编号。硬盘的每一个盘面有500～2500个磁道，新式大容量硬盘每面的磁道数更多。
- 所有盘面上的同一磁道构成一个圆柱，通常称做柱面。
- 这些同心圆不是连续记录数据，而是被划分成一段段的圆弧，每段圆弧叫做一个扇区。

Basic Geometry

旋转速率通常以每分钟转数（Rotations Per Minute，RPM）来测量，典型的现代数值在 7200～15000 RPM 范围内

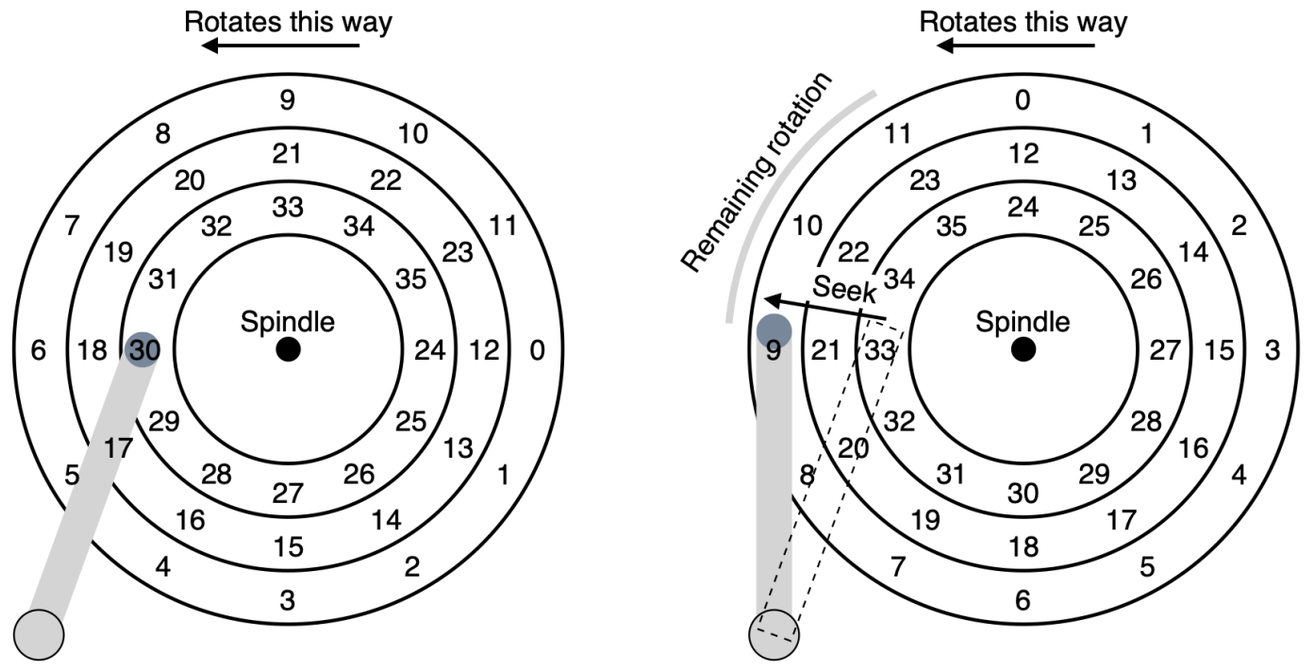


The Rotational Delay



- This track has just 12 sectors, each of which is 512 bytes in size and addressed by the numbers 0 through 11.
- The disk head, attached to the end of the arm, is positioned over sector 6, and the surface is rotating **counter-clockwise（逆时针）**.
- 如果完整的旋转延迟是 R，那么磁盘必然产生大约为 R/2 的旋转延迟，以等 待 0 来到读/写磁头下面（如果我们从 6 开始）。对这个单一磁道，最坏情况的请求是第 5 扇 区.

Seek Time



示例磁盘表面有 3 条磁道（左图）。 在该图中，磁头当前位于最内圈的磁道上（它包含扇区 24～35）

为了读取扇区 11，驱动器必须首先将磁盘臂移动到正确的磁道（在这种情况下，是最外面的磁道），通过一个所谓的寻道（seek）过程。寻道，以及旋转，是 最昂贵的磁盘操作之一。

寻道有许多阶段：首先是磁盘臂移动时的加速阶段。然后随着磁盘臂 全速移动而惯性滑动。然后随着磁盘臂减速而减速。最后，在磁头小心地放置在正确的磁 道上时停下来。停放时间（settling time）通常不小，例如 0.5～2ms，因为驱动器必须确定 找到正确的磁道

若磁盘的转速提高一倍，则（ ）

A. 平均存取时间减半　　B. 平均寻道时间减半

C. 存储道密度提高一倍　　D. 平均寻道时间不变

若磁盘的转速提高一倍，则（ ）
A. 平均存取时间减半　　B. 平均寻道时间减半
C. 存储道密度提高一倍　D. 平均寻道时间不变

- 磁盘从接到命令到向目标扇区读取或写入数据完毕共经历三个阶段:
- 第一阶段，磁头沿径向移动，移到目标扇区所在磁道的上方（注意，不是目标扇区，而是目标扇区所在的磁道），这段时间称为寻道时间，目标扇区所在磁道跟0磁道的远近不同寻道时间也不一样。这个时间是随机变化的，因此用平均值来表示。最大寻道时间和最小寻道时间的平均值称为平均寻道时间，有的书上称为平均定位时间，约为10毫秒。
- 第二阶段，找到目标磁道后通过盘片的旋转，使得要目标扇区转到磁头的下方，这段时间称为旋转延迟时间，取最大最小的平均值即旋转半周的时间作为平均旋转延迟时间，有的书上称为平均等待时间。比如，一个7200（转/每分钟）的硬盘，每旋转一周所需时间为60×1000÷7200＝8.33毫秒，则平均旋转延迟时间为8.33÷2＝4.17毫秒。
- 第三阶段，向目标扇区读取或写入数据，时间约为零点几个毫秒。 由此可见，第三阶段所耗时间相对第一、二阶段可以忽略不记。于是将平均寻道时间与平均旋转延迟时间之和称为平均存取时间。决定一个磁盘读写速度的是它的平均存取时间。
- 可见，当磁盘的转速改变时，由于平均寻道时间与磁盘转速无关，所以它不变。磁盘的转速提高一倍，只是平均旋转延迟时间减半。从上述分析可以看出，答案为D

Some Other Details

- 任何现代磁盘驱动器都有一个重要组成部分，即缓存（cache），由于历史原因有时称为磁道缓冲区（track buffer）。该缓存只是少量的内存（通常大约 8MB 或 16MB），驱动器可以使用这些内存来保存从磁盘读取或写入磁盘的数据。

- 在写入时，驱动器面临选择：回写（write back，在数据更新时只写入缓存Cache。只在数据被替换出缓存时，被修改的缓存数据才会被写到后端存储）和透写（write through，在数据更新时，同时写入缓存Cache和后端存储。）。

- 回写缓存有时会使驱动器看起来“更快”，但可能有丢失数据危险。

I/O Time: Doing The Math

- We can now represent I/O time as the sum of three major components:

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

- 通常比较驱动器用 I/O 速率（$R_{I/O}$）更容易（如下所示），它很容易从时间计算 出来。只要将传输的大小除以所花的时间:

$$R_{I/O} = \frac{Size_{Transfer}}{T_{I/O}}$$

表 37.1　磁盘驱动器规格：SCSI 与 SATA

	Cheetah 15K.5	Barracuda
容量	300GB	1TB
RPM	15000	7200
平均寻道时间	4ms	9ms
最大传输速度	125MB/s	105MB/s
磁盘	4	4
缓存	16MB	16/32MB
连接方式	SCSI	SATA

硬盘转速以每分钟多少转来表示，单位表示为RPM

- 根据这些数据，我们可以开始计算驱动器在下列两个工作负载下的性能

  随机（random）工作负载，它向磁盘上的随机位置发出小的（例如 4KB）读取请求。随机工作负载在许多重要的应用程序中很常见，包括数据库管理系统。
  顺序（sequential）工作负载，只是从磁盘连续读取大量的扇区，不会跳过。顺序访问模式也很常见。

I/O Time Example

- **Random workload**: Issue 4KB read to random locations on the disk
- **Sequential workload**: Read 100MB consecutively from the disk

15000 RPM 等于 250 RPS（每秒转速）。因此，每次旋转需要 4ms。平均而言，磁盘将会遇到半圈旋转，因此平均时间为 2ms。

		Cheetah 15K.5	Barracuda
T_{seek}		4 ms	9 ms
$T_{rotation}$		2 ms	4.2 ms
	$T_{transfer}$	4KB/125MBs=32 us	4KB/105MBs=38 us
Random	$T_{I/O}$	$T_{seek}+T_{rotation}+T_{transfer}$ =6 ms	$T_{seek}+T_{rotation}+T_{transfer}$ =13.2 ms
	$R_{I/O}$	Size/$T_{I/O}$=0.66 MB/s	Size/$T_{I/O}$=0.31 MB/s
Sequential	$T_{transfer}$	100MB/125MBs=800 ms	100MB/105MBs=950 ms
	$T_{I/O}$	$T_{seek}+T_{rotation}+T_{transfer}$ =806 ms	$T_{seek}+T_{rotation}+T_{transfer}$ =963.2 ms
	$R_{I/O}$	Size/$T_{I/O}$=125 MB/s	Size/$T_{I/O}$=105 MB/s

Disk Drive Performance: SCSI Versus SATA

There is a huge gap in drive performance between random and sequential workloads

如何加速对磁盘的访问

磁盘访问一个磁盘块平均要用 10ms,如果磁盘访问请求的到达个数超过 10ms 一次，那么这些请求就会被无限的阻塞，调度延迟将会变的非常大

按柱面组织数据：寻道时间占平均块访问时间的一半，如果我们选择在一个柱面上连续的读取所有块，那么我们只需要考虑一次寻道时间，而忽略其它时间。这样，从磁盘上读写数据的速度就接近于理论上的传输速率。

使用多个磁盘：如果我们使用多个磁盘来替代一个磁盘，只要磁盘控制器、总线和内存能以 n 倍速率处理数据传输，则使用 n 个磁盘的效果近似于 1 个磁盘执行了 n 次操作。因此使用多个磁盘可以提高系统的性能。

磁盘调度：提高磁盘系统吞吐率的另一个有效方法是让磁盘控制器在若干个请求中选择一个来首先执行。

预取数据：在一些应用中，我们是可以预测从磁盘请求块的顺序的。因此我们就可以在需要这些块之前就将它们装入主存。

Disk Scheduling

OS的任务之一就是有效地使用硬件。对磁盘驱动器，访问时间有以下两个主要部分

寻道时间：磁臂将磁头移动到包含目标扇区的柱面的时间。

旋转延迟：磁盘需要将目标扇区转动到磁头下的时间。

最小化寻道时间

寻道时间可以近似的用寻道距离来表示

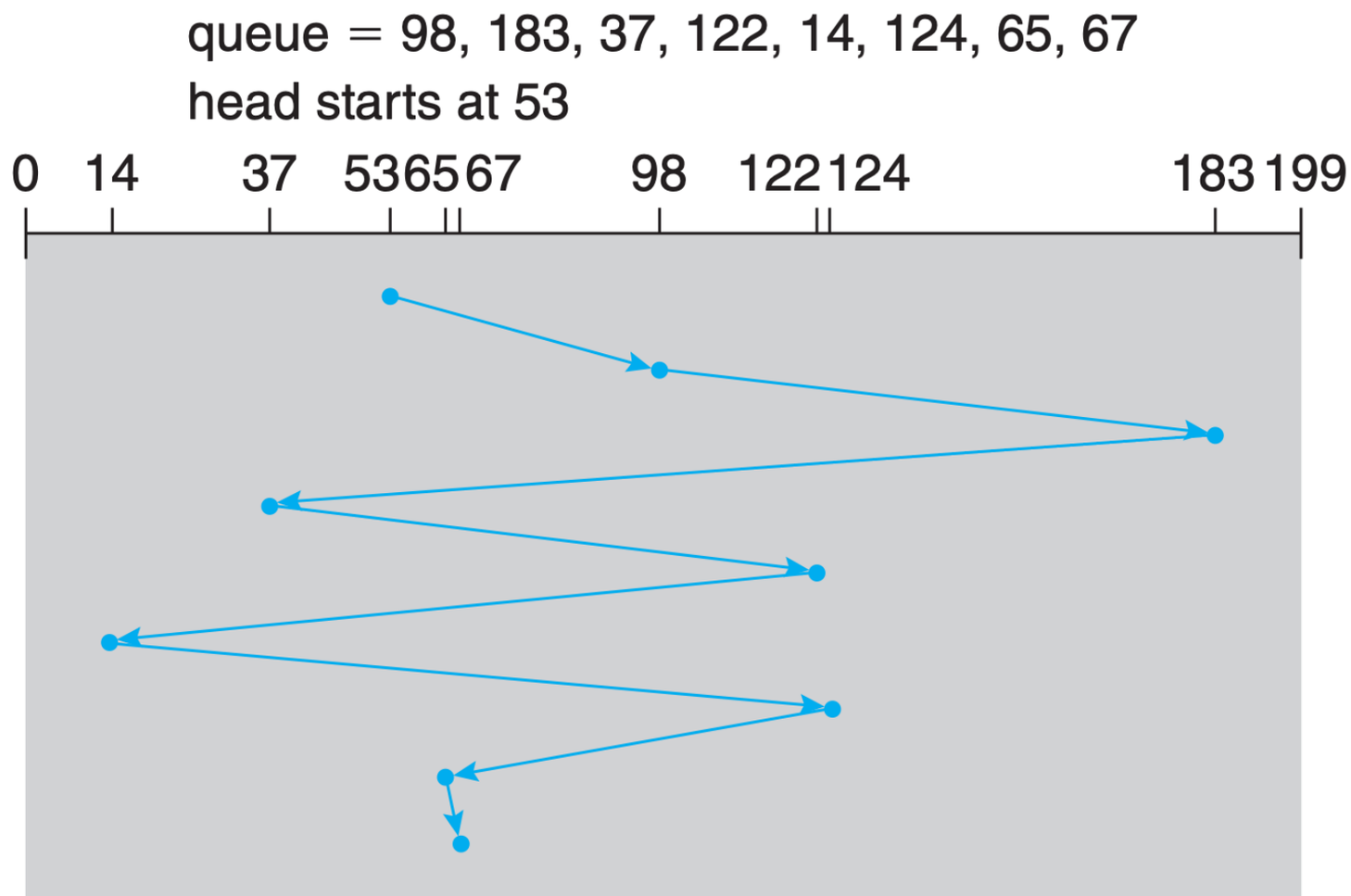
- 假定有以下磁盘请求（磁道编号从0—199）

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

- 当前磁头位置为53

FCFS Scheduling

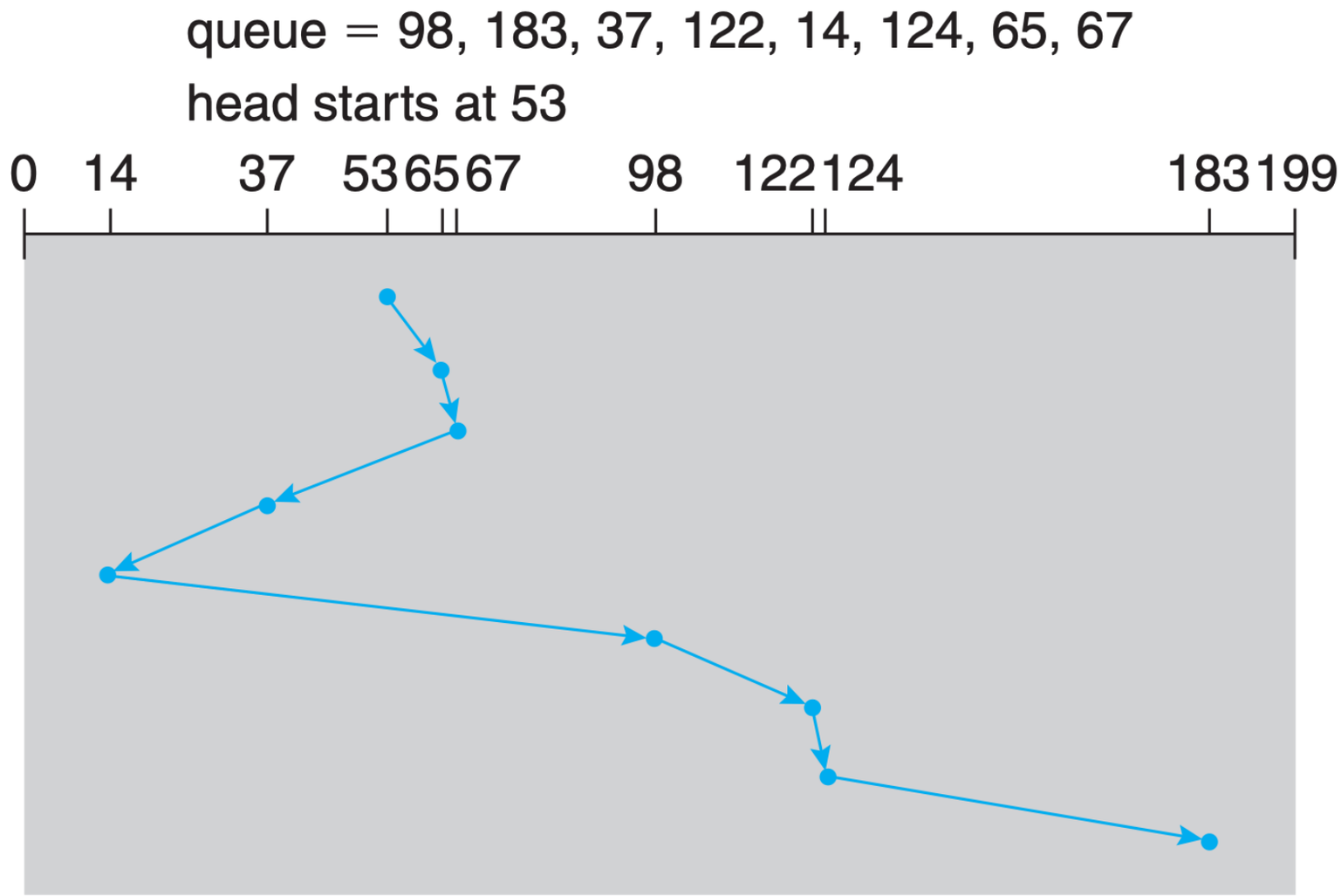
Illustration shows total head movement of 640 cylinders.



优点：简单，公平；
缺点：效率不高，相邻两次请求可能会造成最内到最外的柱面寻道，使磁头反复移动，增加了服务时间，对机械也不利

SSTF Scheduling

Illustration shows total head movement of 236 cylinders.



优先选择距当前磁头最近的访问请求进行服务，主要考虑寻道优先

优点：改善了磁盘平均服务时间；

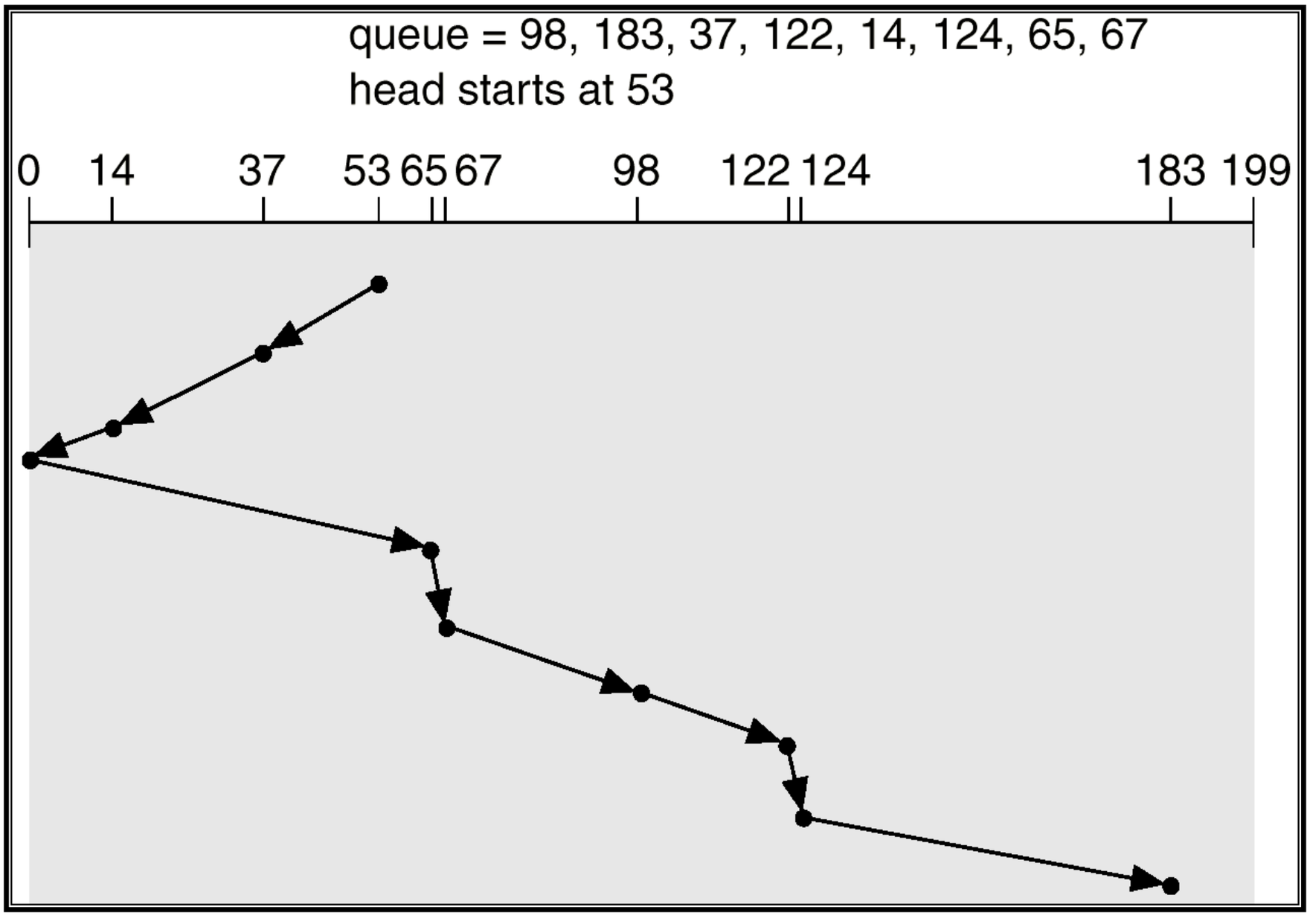
缺点：造成某些访问请求长期等待得不到服务

SCAN（电梯调度）

- 克服了最短寻道优先的缺点，既考虑了距离，同时又考虑了方向
- 磁臂从磁盘的一端向另一端移动，同时当磁头移过每个柱面时，处理位于该柱面上的服务请求。当到达另一端时，磁头改变移动方向，处理继续。磁头在磁盘上来回扫描。有时也叫做“电梯(Elevator)”算法

SCAN (Cont.)

Illustration shows total head movement of 208 cylinders.

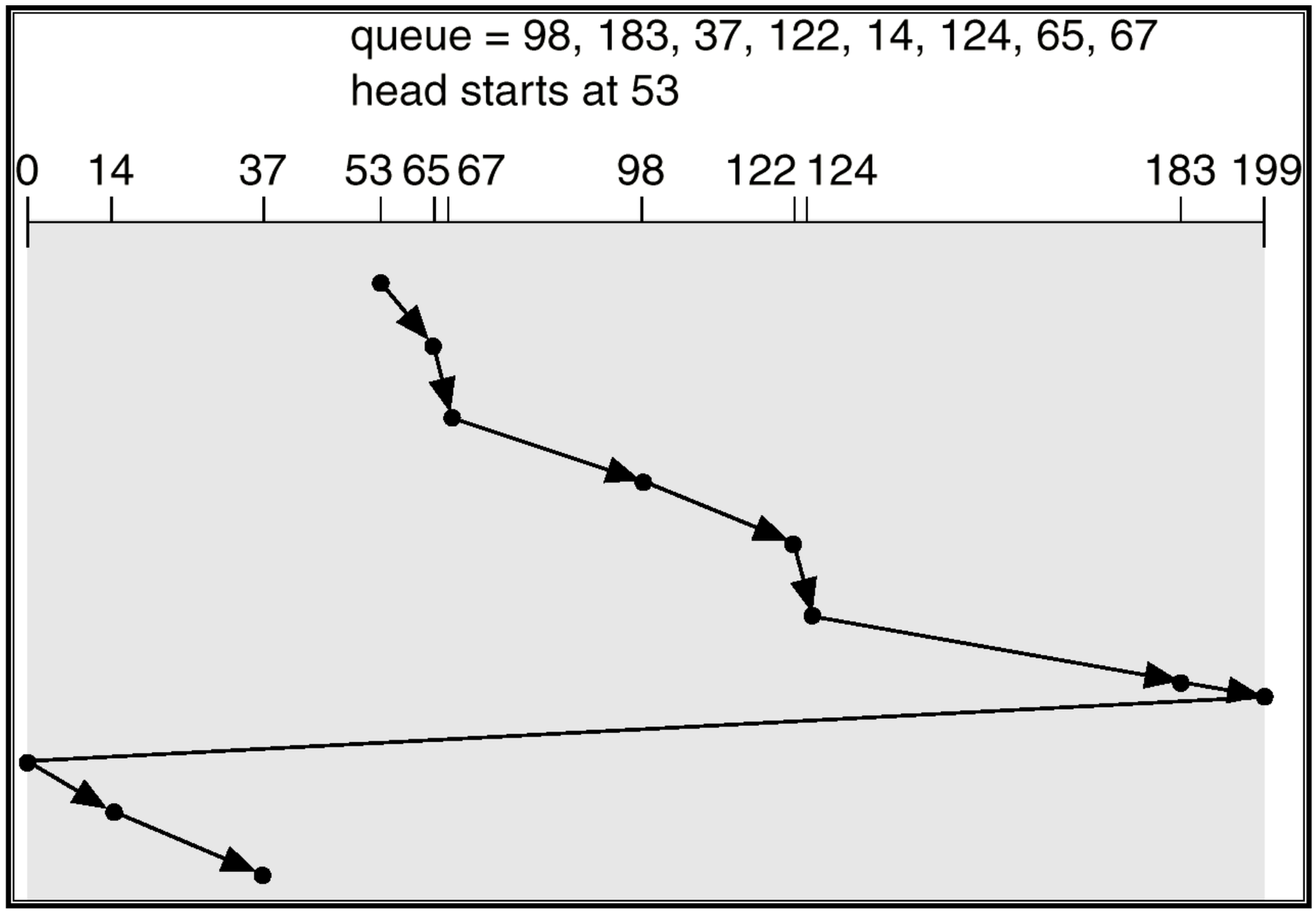


C-SCAN

- 也称单向扫描算法。

循环扫描算法是对扫描算法的改进。如果对磁道的访问请求是均匀分布的，当磁头到达磁盘的一端，并反向运动时落在磁头之后的访问请求相对较少。这是由于这些磁道刚被处理，而磁盘另一端的请求密度相当高，且这些访问请求等待的时间较长，为了解决这种情况，循环扫描算法规定磁头单向移动。例如，只自里向外移动，当磁头移到最外的被访问磁道时，磁头立即返回到最里的欲访磁道，即将最小磁道号紧接着最大磁道号构成循环，进行扫描。

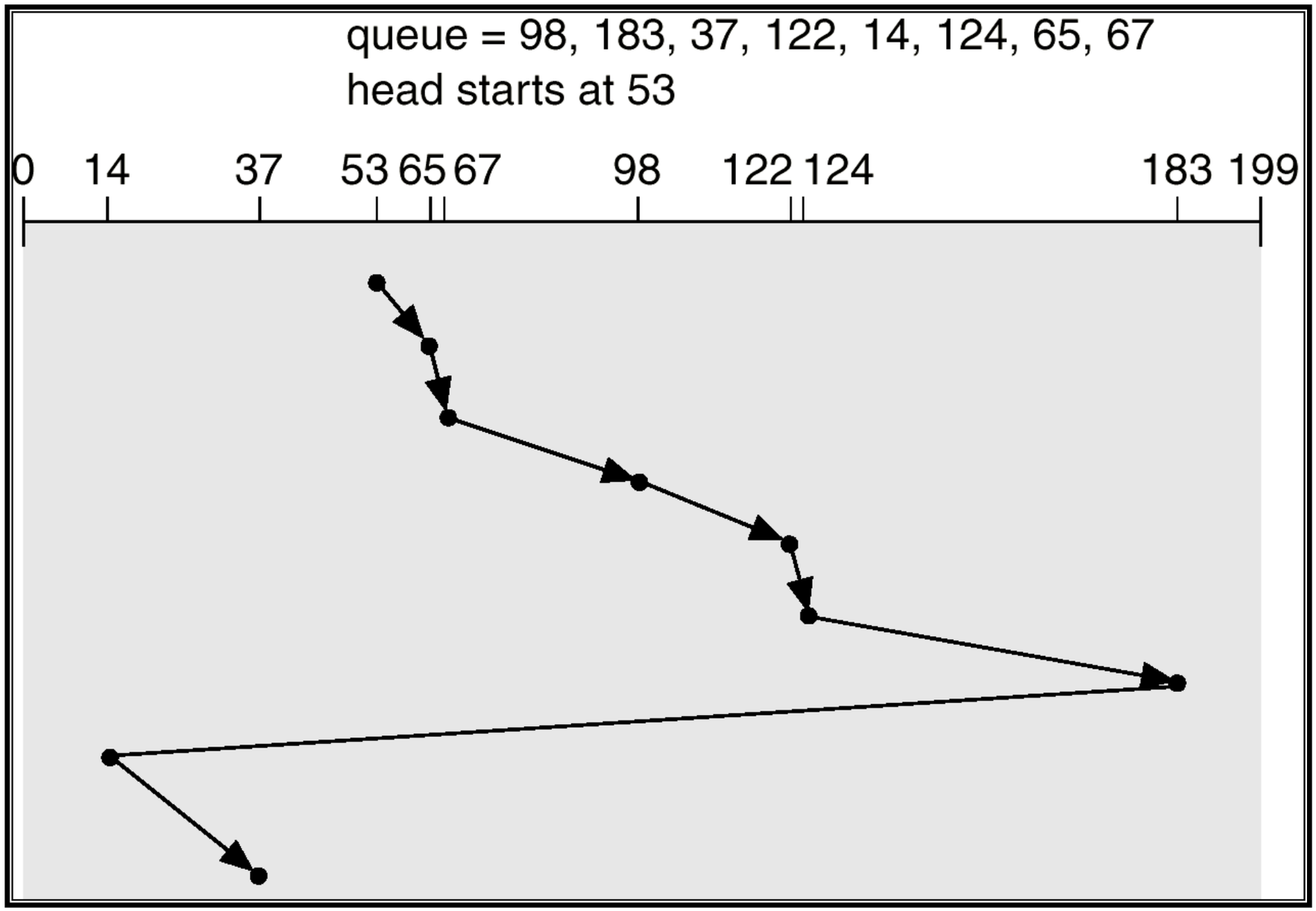
C-SCAN (Cont.)


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

- SCAN与C－SCAN算法的改进版
- 通常，磁头只移动到一个方向上最远的请求为止。接着，它马上回头，而不是继续到磁盘的尽头。这种形式的SCAN和C-SCAN称为LOOK和C-LOOK调度

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

驱动器有 5000 个柱面，从 0 到 4999，驱动器正在为柱面 143 的一个请求提供服务，且前面的一个服务请求是在柱面 125.按 FIFO 顺序，即将到来的请求队列是86，1470，913，1774，948，1509，1022，1750，130从现在磁头位置开始，按照下面的磁盘调度算法，要满足队列中即将到来的请求要求磁头总的移动距离（按柱面数计）是多少?

FCFS

SSTF

SCAN

LOOK

C-SCAN

C-LOOK

FCFS 的调度是 143， 86， 1470， 913， 1774，948，1509，1022，1750，130。总寻道距离是 7081。

SSTF 的调度是 143，130，86，913，948，1022，1470，1509，1750，1774。总寻道距离是 1745。

SCAN 的调度是 143，913，948，1022，1470，1509，1750，1774，4999，130，86。总寻道距离是 9769。

LOOK 的调度是 143， 913， 948， 1022， 1470，1509，1750，1774，130， 86。总寻道距离是 3319。

C-SCAN 的调度是 143， 913， 948， 1022， 1470，1509，1750，1774，4999，0，86，130。总寻道距离是　　。

C-LOOK 的调度是 143， 913， 948， 1022，1470，1509，1750，1774，86，130。总寻道距离是 3363。

End