Secondary Storage

Any modern computer system will incorporate (at least) two levels of storage:

primary storage:	random access memory (RAM)
typical capacity	32MB to 1GB
cost per MB	$3.00
typical access time	5ns to 60ns
burst transfer rate	??
secondary storage:	magnetic disk/optical devices/tape systems
typical capacity	4GB to 100GB for fixed media; ∞ for removable
cost per MB	$0.01 for fixed media, more for removable
typical access time	8ms to 12ms for fixed media, larger for removable
burst transfer rate	??

Note: all statistics here are guaranteed invalid by July 1, 2000.

©2000 McQuain WD

Units of Measurement

Spatial units:

byte	8 bits
kilobyte (KB)	1024 or 2^{10} bytes
megabyte (MB)	1024 kilobytes or 2^{20} bytes*
gigabyte (GB)	1024 megabytes or 2^{30} bytes

* **Most, if not all, secondary storage vendors lie about this and call 1,000,000 bytes a megabyte and 1,000,000,000 bytes a gigabyte.**

We will ignore the issue.

Time units:

nanosecond (ns)	one-billionth (10^{-9}) of a second
microsecond (μs)	one-millionth (10^{-6}) of a second
millisecond (ms)	one-thousandth (10^{-3}) of a second

©2000 McQuain WD

Primary versus Secondary Storage

While the particular values given earlier are volatile, the relative performances suggested are actually quite stable:

Primary storage:

- costs several hundred times as much per unit as secondary storage.
- has access times that are 250,000 to 1,000,000 times faster than secondary storage
- has transfer rates that are ?? times faster than secondary storage

Why do WE care (in a data structures class)?

Often data must be first read from disk into memory for processing, and then results must be written back to disk after processing.

In many cases, data sets are too large to store in memory at once.

©2000 McQuain WD

File Storage

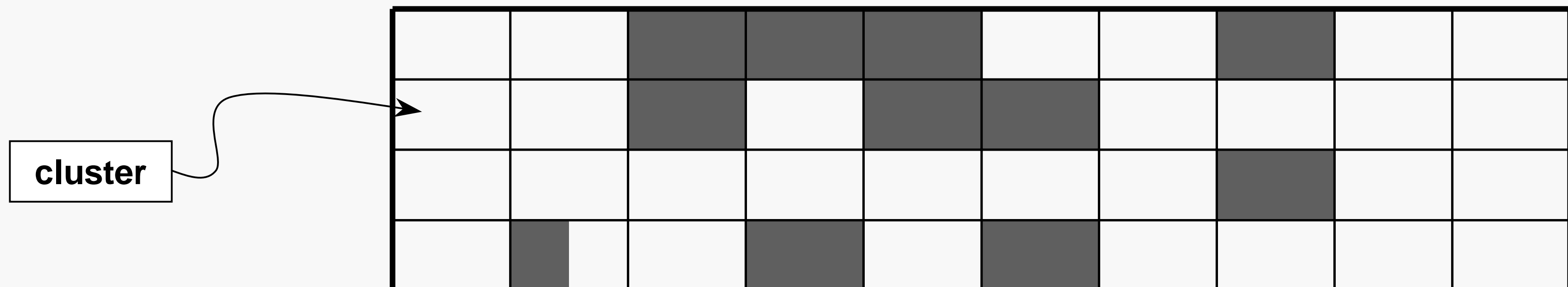
In many file systems, space is allocated in fixed-size chunks called clusters.

Typical cluster sizes range from 1KB up to 32KB, usually equaling a power of 2.

When a file is stored on disk, an integer number of clusters are allocated for the file; since files sizes are typically not a multiple of the cluster size, this means that a certain amount of space is wasted to internal fragmentation.

On average, ½ of a cluster is wasted per file stored. Obviously that adds up… but that's not really our concern in this course.

The clusters are also not usually stored contiguously on the disk. This external fragmentation can cause a serious degradation of performance when a file is being read from or written to disk.



©2000 McQuain WD

Hard Disk Performance Factors

To illustrate the issues, we will consider the physical organization and behavior of a typical hard disk design.

Note that the presentation here is an over-simplification and that contemporary hard drive designs incorporate control sophistication not discussed here.

Despite advances, the basic performance issues remain the same.

We will consider three primary factors that affect the time required to read requested data from the disk into memory:

seek time	time for the appropriate I/O head to reach the desired track
latency	time for the appropriate sector to rotate to the I/O head
transfer time	time for the data to be read to move past the I/O head

First we must have a clear picture of the hardware…

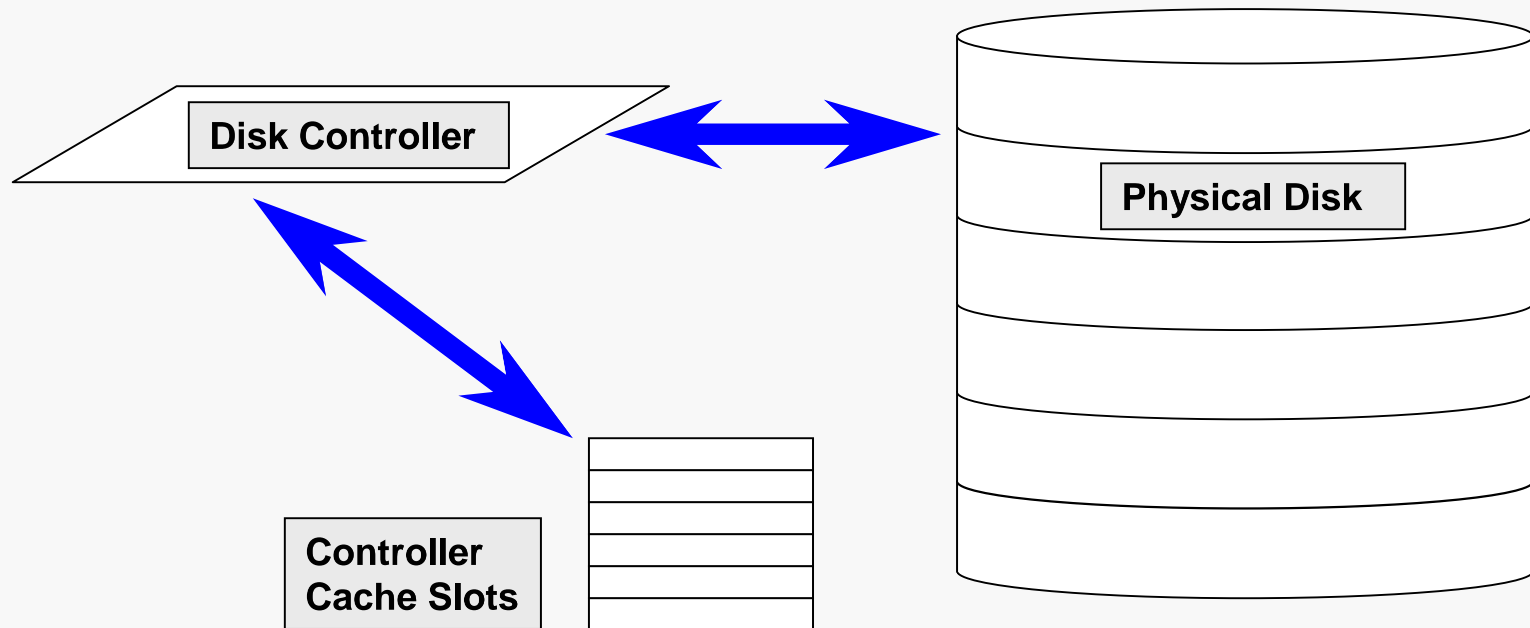
©2000 McQuain WD

Disk Controllers

All disk systems incorporate an on-board controller which acts as an interface between the CPU and the disk hardware.

The controller masks the internal physical organization of the disk (covered in the next few slides) from the CPU and user processes.

The controller has an internal cache (typically 256KB up to several MB) that it uses to buffer data for read/write requests.

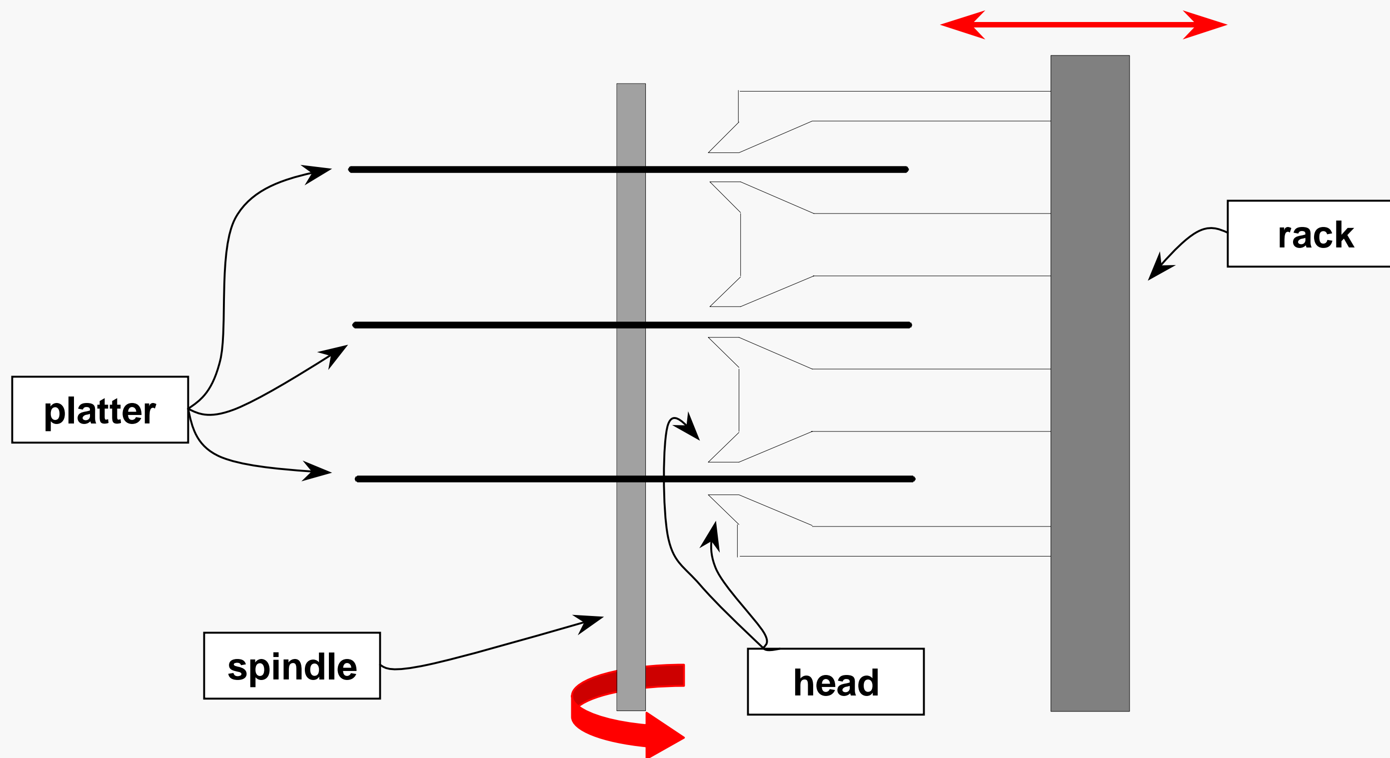


©2000 McQuain WD

Platters and I/O Heads

A typical hard drive contains a number of circular platters, attached to a rotating spindle. Each platter holds data on one or both of its surfaces.

The data is read and written by I/O heads, typically one per data surface. These I/O heads are mounted in a rack and all move in and out in unison. Typically, only one I/O head can be active (reading or writing data) at any given time.



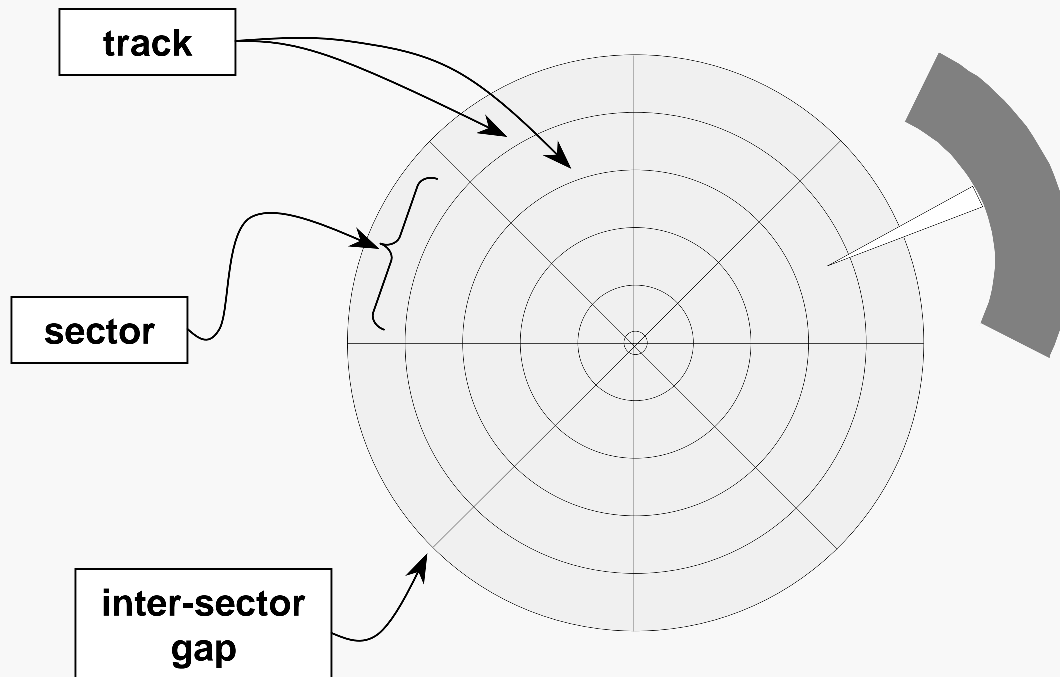
©2000 McQuain WD

Tracks and Sectors

Each data surface is organized into concentric rings of data, called tracks.

Each track is divided into a number of segments, called sectors.

We will assume that each sector contains the same amount of data, regardless of which track it belongs to. (That is actually true for some older disks.)



The basic unit of disk I/O is the sector.

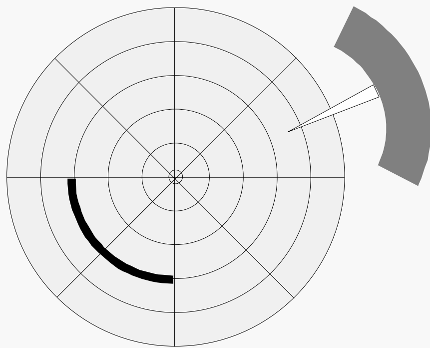
We will see shortly that reading an entire sector from disk takes only slightly longer than reading a single byte.

©2000 McQuain WD

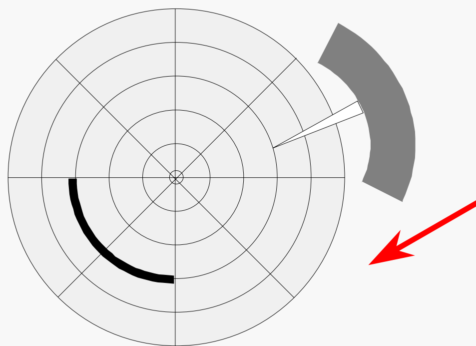
Reading a Sector

When a disk read is requested the following actions must be carried out:

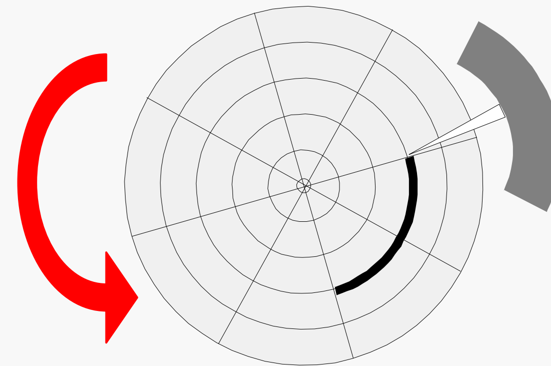
1. **Determine what sector(s) must be read — the disk controller is responsible for this.**



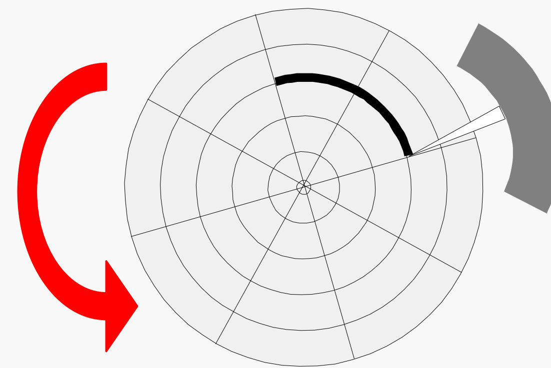
2. **Move the read head to the track containing the targeted sector(s).**



3. **Wait for the beginning of the first sector to rotate to the head.**



4. **Read the data as it rotates beneath the head.**



©2000 McQuain WD

Seek Time

seek time — the time required to move the head to the appropriate track

The seek time depends only on the speed with which the head rack moves, and the number of tracks that the head must move across to reach its target.

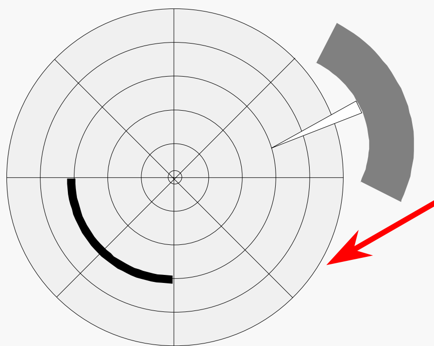
Given the following (which are constant for a particular disk):

H_s = the time for the I/O head to start moving

H_T = the time for the I/O head to move from one track to the next

then the time for the head to move n tracks is:

$$\text{Seek}(n) = H_s + H_T n$$



QTP: On average, the head will move 1/3 of the way across the platter. Why?

©2000 McQuain WD

Rotational Latency Time

latency — the time required for the appropriate sector to rotate to the position of the I/O head

The rotational latency time depends only on the speed at which the spindle rotates, and the angle through which the track must rotate to reach the I/O head.

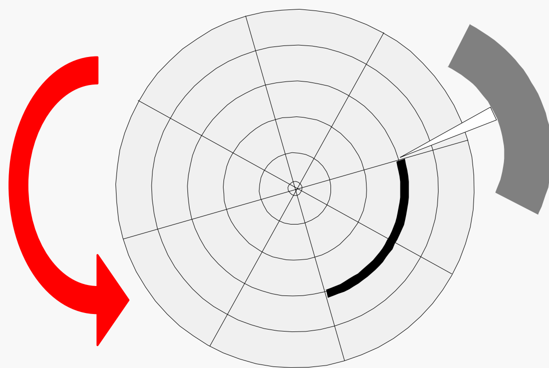
Given the following:

R = the rotational speed of the spindle (in rotations per second)

θ = the number of radians through which the track must rotate

then the rotational latency is:

$$\text{Latency}(\theta) = \left(\frac{\theta}{2\pi}\right) \times \left(\frac{1000}{R}\right)$$



On average, the latency time will be the time required for the platter to complete 1/2 of a full rotation.

©2000 McQuain WD

Data Transfer Time

<u>transfer time</u> the time required for the appropriate sector(s) to rotate under the I/O head (for contiguous sectors on the same track)

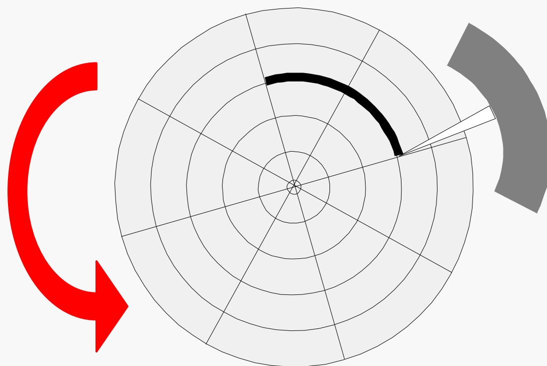
The transfer time depends only on the speed at which the spindle rotates, and the number of sectors that must be read.

Given:

S_T = the total number of sectors per track

the transfer time for n contiguous sectors on the same track is:

$$\text{Transfer}(n) = \left(\frac{n}{S_T}\right) \times \left(\frac{1000}{R}\right)$$



For sectors on different tracks, each track must be analyzed separately, allowing for seek and latency.

©2000 McQuain WD

Total Read/Write Time

The total time to read/write data is from/to disk is then the sum of the seek time, the rotational latency time, and the transfer time.

This ignores the time for controller logic, delays due to multitasking queues, etc.; that is fair because those times are normally orders of magnitude less that the times considered.

One note: we have assumed in the analysis of the data transfer time that the I/O heads are capable of reading/writing data as fast as the sector moves beneath the head. On older disk systems that was often not the case, and interleaving was necessary to optimize read/write performance. On most contemporary disk systems, the I/O heads are fast enough that such tricks are unnecessary.

©2000 McQuain WD

Example: Contiguous File Read

Assume a disk system with the following parameters:

Total capacity:	**675 MB**
# of platters:	**15**
	(30 data surfaces)
# of tracks per platter:	**576**
# of sectors per track:	**160**
cluster size:	**8 KB**
spindle speed:	**3600 RPM**
head start time:	**3 ms**
head track traversal time:	**0.08 ms**

Question: how much time would be required, on average, to read a file of size 140 KB, assuming the file is stored in contiguous sectors on adjacent tracks (since one track won't hold the whole file in this case)?

©2000 McQuain WD

Example: Contiguous File Read

We must compute some values from the given disk parameters:

rotations per second:	**60**
capacity of one platter:	**45 MB**
capacity of one track:	**80 KB**
capacity of one sector:	**512 bytes**
sectors per cluster:	**16**
clusters per track:	**10**

So a 140 KB file would occupy 18 clusters, or one full track plus 8 clusters on an adjacent track. The last cluster is only partially filled, but we will ignore that in our calculations.

For reading the first (full) track, assuming average values:

- seek time = 3 + (576/3)*0.08 = 18.36 ms
- latency time = (1/2)*(1000/60) = 8.33 ms
- transfer time = (160/160)*(1000/60) = 16.67 ms

total: 43.36 ms

©2000 McQuain WD

Example: Contiguous File Read

For reading the relevant sectors from the second track, assuming average values:

- seek time = 3 + (1)*0.08 = 3.36 ms
- latency time = (1/2)*(1000/60) = 8.33 ms
- transfer time = (128/160)*(1000/60) = 13.33 ms

total: 24.74 ms

So the total time to read the file into memory* would be about 68.1 ms.

* **Actually this is the time to read the file into the disk buffer memory.**

©2000 McQuain WD

Example: Fragmented File Read

What if the file is scattered all over the disk?

Then it will take a lot longer… for each cluster:

- seek time = 3 + (576/3)*0.08 = 18.36 ms
- latency time = (1/2)*(1000/60) = 8.33 ms
- transfer time = (16/160)*(1000/60) = 1.67 ms

total: 28.36 ms

Since the file occupies (part or all of) 18 clusters, the total read time for a completely fragmented copy of the file would be about 510.48 ms.

©2000 McQuain WD

Sector Read vs Single Byte Read

The programs you have implemented simply perform read/write actions, usually on single variables which may store from 1 to a few hundred bytes of data.

Why do typical disk controllers perform read/write operations in sector-sized chunks? Performance… given the disk system described earlier:

time to read one full track: 43.36 ms (slide 14)

time to read one full sector: 26.79 ms

18.36 ms + 8.33 ms + (1/160)*(1000/60)

time to read one byte: 26.69 ms

18.36 ms + 8.33 ms + (1/512)*(1/160)*(1000/60)

The time to read one sector is only slightly more than the time to read a single byte, largely because the seek and latency times dominate the transfer time.

©2000 McQuain WD

Locality of Reference

In view of the previous slide, it makes sense to design programs so that data is read from and written to disk in relatively large chunks… but there is more.

Spatial Locality of Reference

In many cases, if a program accesses one part of a file, there is a high probability that the program will access nearby parts of the file in the near future.

Moral: grab a larger chunk than you immediately need.

Temporal Locality of Reference

In many cases, if a program accesses one part of a file, there is a high probability that the program will access the same part of the file again in the near future.

Moral: once you've grabbed a chunk, keep it around.

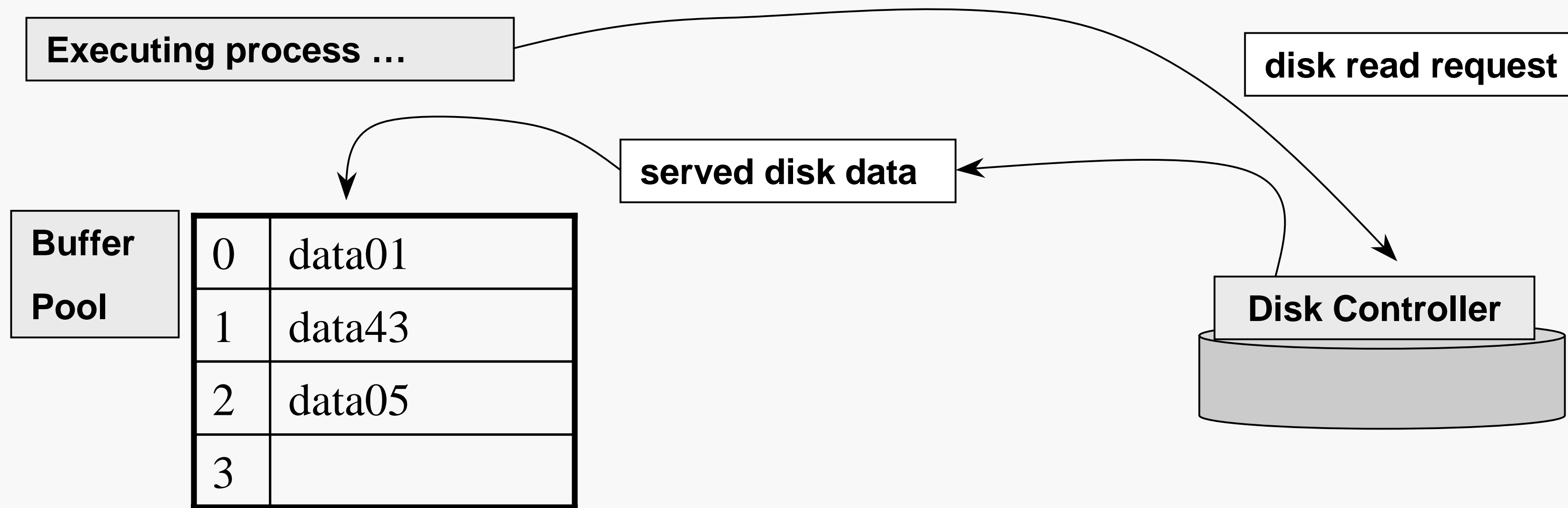
©2000 McQuain WD

Buffer Pools

buffer pool — a series of buffers (memory locations) used by a program to cache disk data

A program that does much disk I/O can often improve its performance by employing a buffer pool to take advantage of locality of reference.

Basically, the buffer pool is just a collection of data chunks. The program reads and writes data in buffer-sized chunks, storing newly-read data chunks into the pool, replacing currently stored chunks as necessary.



©2000 McQuain WD

Replacement Strategies

The buffer pool must be organized physically and logically.

The physical organization is generally an ordered list of some sort.

The logical organization depends upon how the buffer pool deals with the issue of replacement — if a new data chunk must be added to the pool and all the buffers are currently full, one of the current elements must be replaced.

If the replaced element has been modified, it (usually) must be written back to disk or the changes will be lost.

Common buffer replacement strategies:

- FIFO (first-in is first-out) organize buffers as a queue
- LFU (least frequently used) replace the least-accessed buffer
- LRU (least recently used) replace the longest-idle buffer

©2000 McQuain WD

FIFO Replacement

Logically the buffer pool is treated as a queue:

Requested "chunks":

Request					
0	0				
4	0	4			
17	0	4	17		
4	0	4	17		
13	0	4	17	13	
7	4	17	13	7	
8	17	13	7	8	
17	17	13	7	8	✔
0	13	7	8	0	

. . .

Takes no notice of the access pattern exhibited by the program. Consider what would happen with the sequence:

0 1 0 2 0 3 0 4 0 5 0 6 …

©2000 McQuain WD

LFU Replacement

For LFU we must maintain an access count for each element of the buffer pool. It is also useful to keep the elements sorted by that count.

Requested "chunks":

Request				
0	0,1			
4	0,1	4,1		
17	0,1	4,1	17,1	
4	0,1	4,2	17,1	
13	0,1	4,2	17,1	13,1
7	4,2	17,1	13,1	7,1
8	4,2	13,1	7,1	8,1
17	4,2	7,1	8,1	17,1
0	4,2	8,1	17,1	0,1
. . .				

Aside from cost of storing and maintaining counter values, and searching for least value, consider the sequence:

0 (500 times)

1 (500 times)

2 3 4 2 3 4 …

©2000 McQuain WD

LRU Replacement

With LRU, we may use a simple list structure. On an access, we move the targeted element to the front of the list. That puts the least recently used element at the tail of the list.

Requested "chunks":

Request					
0	0				
4	4	0			
17	17	4	0		
4	4	17	0		
13	13	4	17	0	
7	7	13	4	17	
8	8	7	13	4	
17	17	8	7	13	
0	0	17	8	7	

. . .

Consider what would happen with the sequence:

0 1 2 3 4 0 1 2 3 4 …

©2000 McQuain WD

Comparison

In general, none of these replacement strategies is best in all cases.

All are used with some frequency.

Intuitively, LRU and LFU make more sense than FIFO.

The performance you get is determined by the access pattern exhibited by the running program, and that is often impossible to predict.

Optimal buffer replacement strategy:

replace the buffer element whose next access lies furthest in the future.

©2000 McQuain WD