Managing Primary Storage

One of the *key* pieces in the Software Architecture puzzle is the I/O buffer. It is primarily through the intelligent use of buffering that the speed limitations of secondary storage are overcome.

The I/O buffer is what allows application programs (and even the operating system) to use RAM effectively for file I/O.

Topic	***Folk & Zoellick***
• System I/O Buffers	§ 3.5.2, 3.6
• More than one Buffer	§ 3.6.1
• Double Buffering	§ 3.6.2
• Buffer Pools	§ 3.6.2
• The Buffer Index	N/A
• Buffer Replacement Policies	
◦ Optimal	N/A
◦ FIFO	N/A
◦ Second Chance	N/A
◦ LRU	§ 3.6.2
◦ LFU	N/A

The System I/O Buffer

System I/O Buffer

A special area of RAM acting as an image of some portion of a file on secondary storage

When a program reads from a file, the portion of disk containing the data to be read is loaded into a system I/O buffer; the program reads from the buffer.

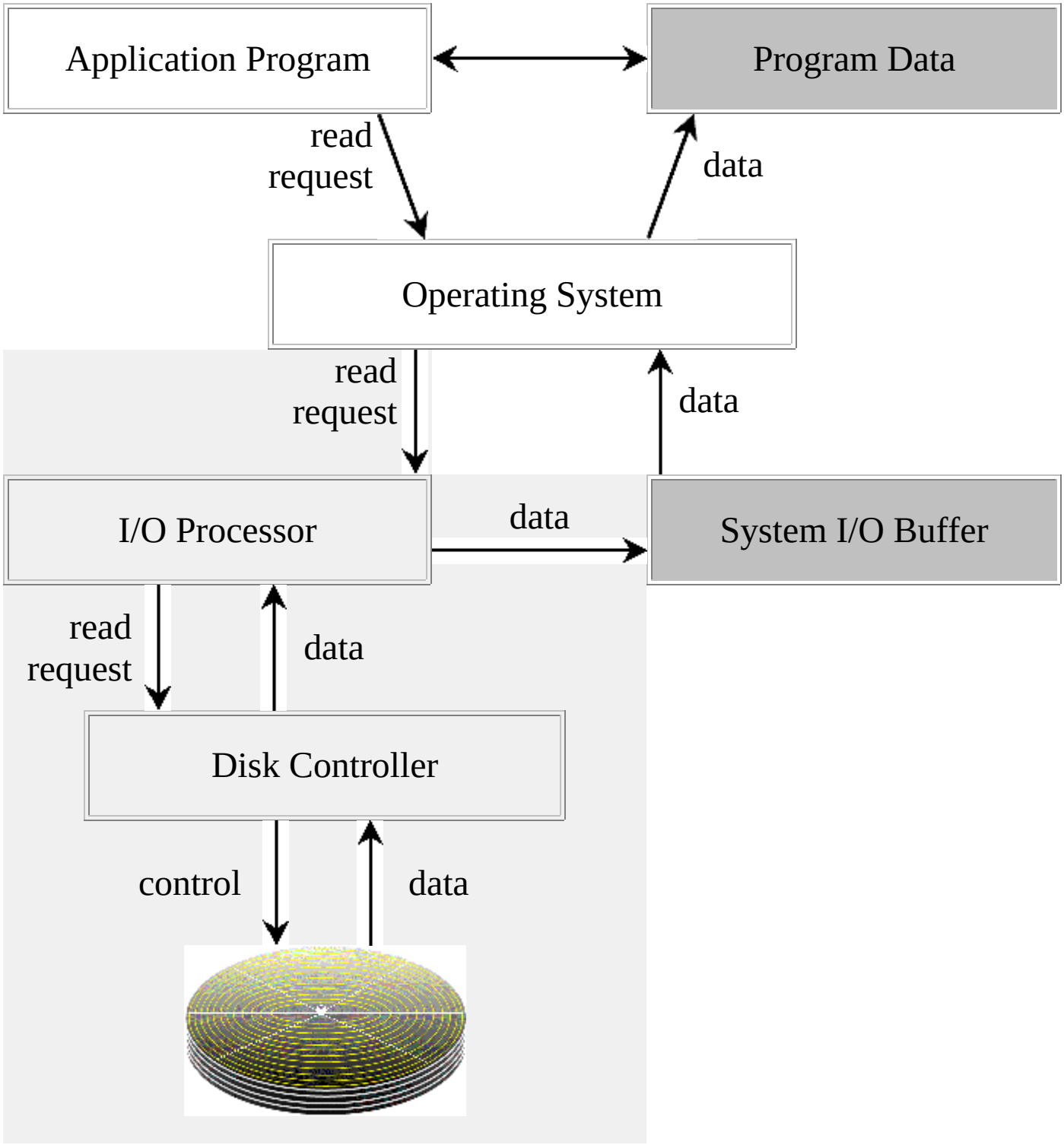
When a program writes to a file, the portion of disk to be modified is loaded into a system I/O buffer; the program writes to the buffer; the buffer is then written back to disk.

The system I/O buffer is maintained by the operating system. Buffering at higher levels is possible.

Q: How could an application program do its own buffering?

A:

The System I/O Buffer (cont.)



Pages, Blocks, Clusters

Pages, Blocks, Clusters

The "portion" of disk loaded into a buffer is often called a *block*. A block typically consists of some number of clusters.

Whatever the size of the portion, the data being loaded into a buffer is often called a *page*.

For this course it does not serve us to be concerned about the details of blocks and pages. We will assume the portions of disk being loaded into buffers are clusters, though we may call them blocks or pages from time to time.

One Buffer

Hearken back to our old `copy` program:

```
main()
{
 char ch;
 FILE *ifil, *ofil;

 ifil = fopen("foo.dat", "r");

 ofil = fopen("bar.dat", "w");

 ch = getc(ifil);
 while(ch != EOF)
    {
     putc(ch, ofil);
     ch = getc(ifil);
    }

 fclose(ifil);
 fclose(ofil);
}
```

Let's say our input file `foo.dat` occupies cluster 31 on disk and our output file `bar.dat` occupies cluster 87.

Q: What happens if we have only one I/O buffer?

A:

One Buffer (cont.)

C	*I/O requests*
`ch = getc(ifil);`	`load cluster 31 into the buffer` `read first byte in the buffer`

`while(ch != EOF)`
`{`
`putc(ch, ofil);`
	`load cluster 87 into the buffer`
	`write byte to the buffer`

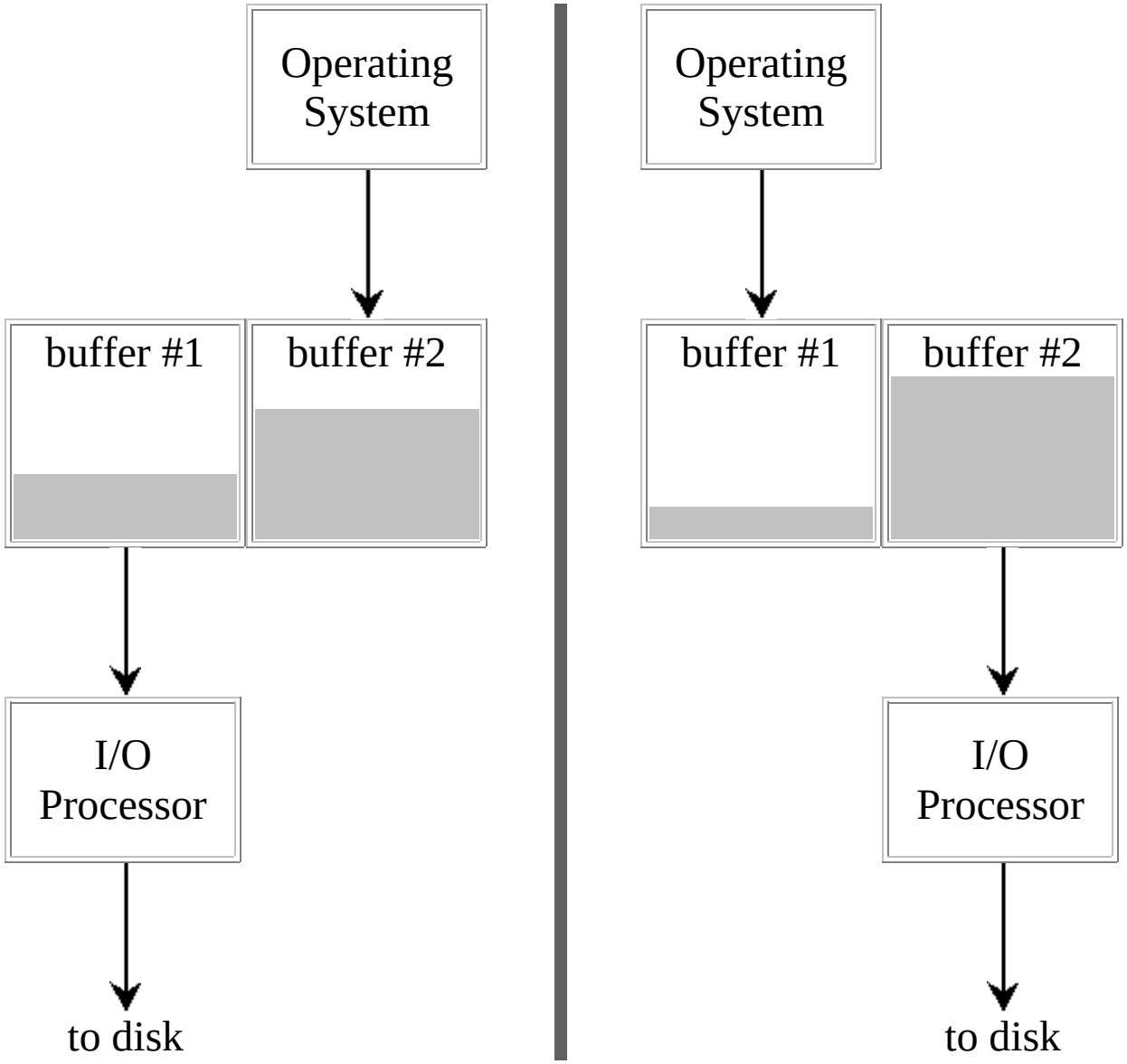
`ch = getc(ifil);`	`load cluster 31 into the buffer` `read next byte in the buffer`
`}`	

Idea: add at least one more buffer giving us one for I, one for O!

Double Buffering

Even if we're only reading or only writing to a single file, it may be useful to have more than one buffer.

Recall that the I/O processor is an independent microprocessor that can execute instructions while the CPU is busy doing something else.



Buffer Pools

If two buffers are better than one, then N buffers must be even better, right?

A group of N system I/O buffers maintained by the operating system is called a *buffer pool*. The operating system must do several things to keep track of multiple buffers:

- keep track of which buffers are in use, which are empty and available
- keep track of the cluster in each buffer in use
- keep track of the mode of the buffer (read or write)
- if a buffer is for writing, have the contents of the buffer been changed since it was loaded from disk (*i.e.*, is the cluster on disk out of date?)

When a program requests access to a certain cluster on disk, the operating system must check if that cluster is already in a buffer. If not, it loads it into an empty buffer not being used.

Obvious Q: What if there *are* no empty buffers available?

A:

Buffer Index

Multiple buffers can be very handy, but they require extra work to maintain. Several kinds of information must be kept about each buffer in the pool.

Buffer index
Buffer #1
pointer to the buffer in RAM
cluster# of the cluster in the buffer
"dirty" bit (data modified?)
sharing information?
usage statistics?
...
Buffer #2
pointer to the buffer in RAM
...
Buffer ...

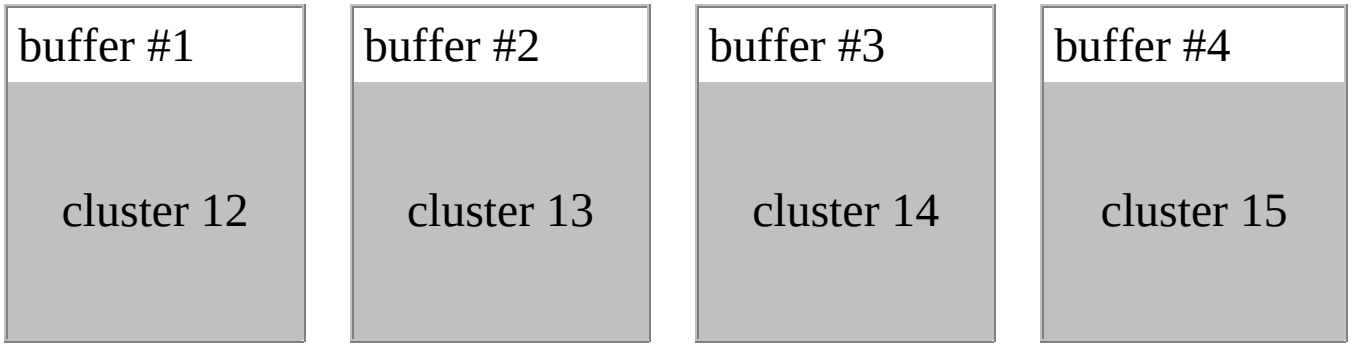
Buffer Replacement Policy

- Assume we have four system I/O buffers.
- Suppose we have a program that needs to read from 5 different clusters throughout its execution.

  The program (for some good reason) needs to read the data in clusters 12 to 16 in the following order:

12 13 14 12 15 16 12 13 14 15 12 16

The system loads clusters 12 to 15 into buffers 1 to 4 respectively.



Q: Where does the system put cluster 16 when the program needs to read it (*i.e.*, which cluster gets *replaced*)?

Optimal Replacement Policy

- Replace the cluster that will be requested again further in the future than any other cluster.

request cluster	12	13	14	12	15	16	12	13	14	15	12	16
replace cluster	__	__	__	__	__	__	__	__	__	__	__	__
buffer 1	12	12	12	12	12							
buffer 2		13	13	13	13							
buffer 3			14	14	14							
buffer 4					15							

Total Replacements: ____

Q: Could the Optimal Replacement Policy be implemented?

A:

Q: What's the point?

A:

First In First Out (FIFO)

- Replace the first cluster that was loaded into a buffer.

request cluster	12	13	14	12	15	16	12	13	14	15	12	16
replace cluster	__	__	__	__	__	__	__	__	__	__	__	__
buffer 1	12	12	12	12	12							
buffer 2		13	13	13	13							
buffer 3			14	14	14							
buffer 4					15							

Total Replacements: ____

Second Chance

- Associate an "access" bit with the cluster in each buffer. When the cluster is accessed, set the bit to ’1’. Replace the first cluster that was loaded into a buffer (as in FIFO) only if its access bit is ’0’. If it is cluster N’s turn to be replaced but its bit is ’1’, set its bit to ’0’ and proceed to the next cluster to check for replacement.

request cluster	12	13	14	12	15	16	12	13	14	15	12	16
replace cluster	—	—	—	—	—	—	—	—	—	—	—	—
buffer 1	12 0	12 0	12 0	12 1	12 1							
buffer 2		13 0	13 0	13 0	13 0							
buffer 3			14 0	14 0	14 0							
buffer 4					15 0							

Total Replacements: ____

Least Recently Used (LRU)

- Replace the cluster that has not been accessed in the longest time.

request cluster	12	13	14	12	15	16	12	13	14	15	12	16
replace cluster	__	__	__	__	__	__	__	__	__	__	__	__
buffer 1	12	12	12	12	12							
buffer 2		13	13	13	13							
buffer 3			14	14	14							
buffer 4					15							

Total Replacements: ____

Least Frequently Used (LFU)

- Replace the cluster that has been accessed the fewest number of times.

request cluster	12	13	14	12	15	16	12	13	14	15	12	16
replace cluster	__	__	__	__	__	__	__	__	__	__	__	__
buffer 1	12 1	12 1	12 1	12 2	12 2							
buffer 2		13 1	13 1	13 1	13 1							
buffer 3			14 1	14 1	14 1							
buffer 4					15 1							

Total Replacements: ____

Choice of Buffer Replacement Policy

- The "best" buffer replacement policy will depend on how data is being accessed by any given program.
- The buffer replacement policy is usually set in the design of the operating system. It is rare that users have any control over the policy.
- If you know in advance how your program will access data on disk, it may be worthwhile to implement your own buffering (DBMSs often do this).

Some Final Thoughts on Buffers

Q: In all our examples we assumed cluster requests were for reading. How does buffer replacement differ when *writing*?

A:

Q: When writing to a buffer, what considerations affect when the buffer should be written to disk?

A:

Q: How do you think your favourite operating system handles requests from multiple processes for the same disk cluster?

A: