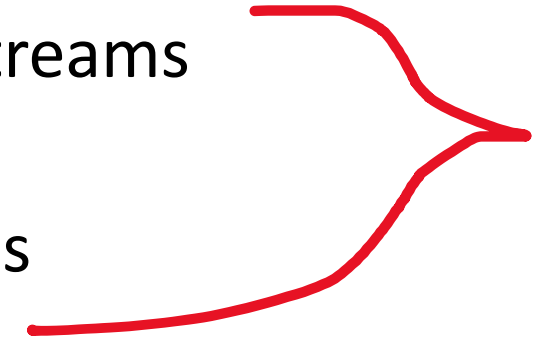
Files and Streams

Streams

- Most file-related classes are in the java.io package (there is also a java.nio which stands for “Network IO”). Two types of streams, Character or Byte, which can also be buffered or unbuffered.

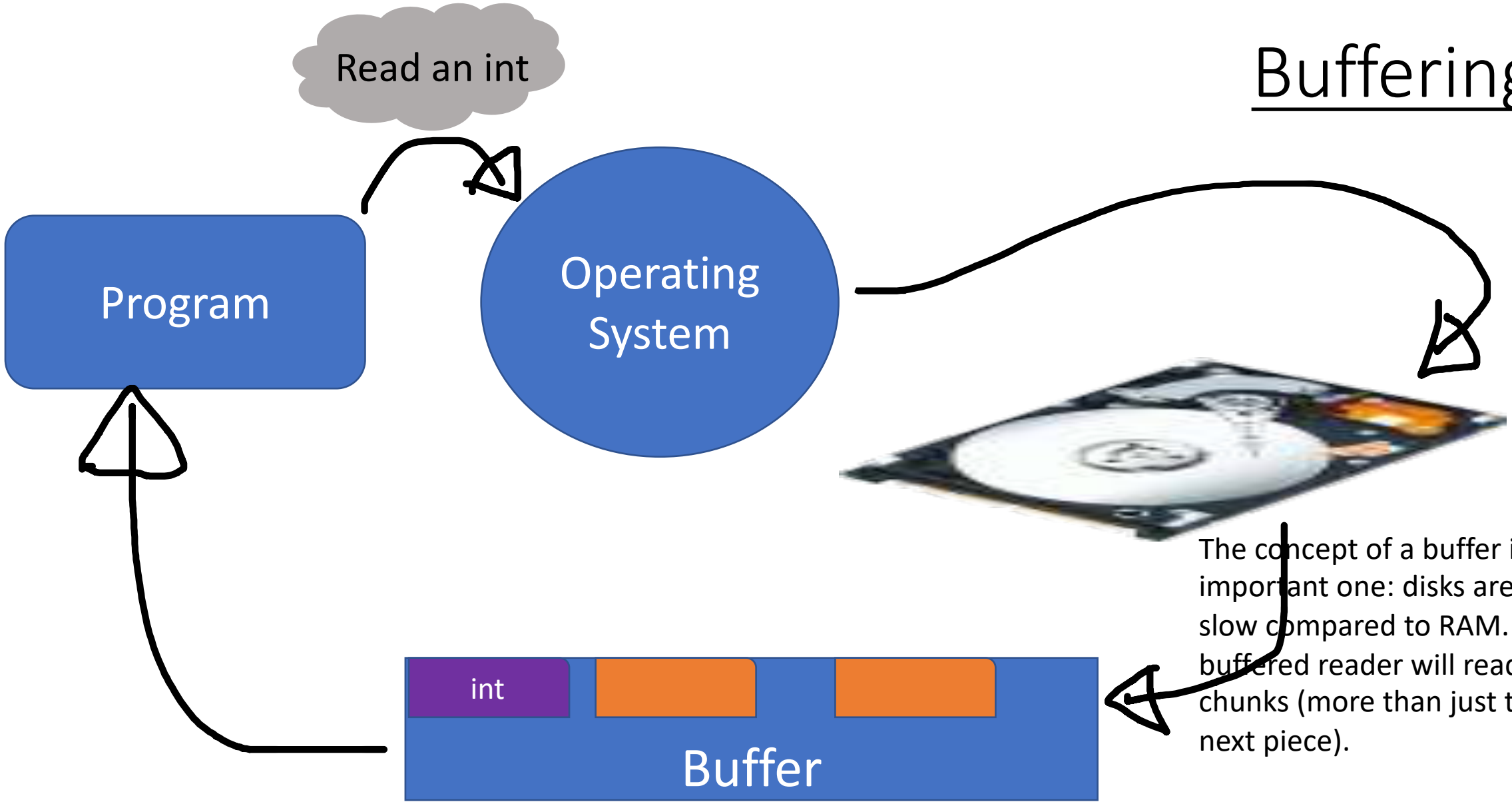
- Character streams
- Byte streams



Buffered or unbuffered

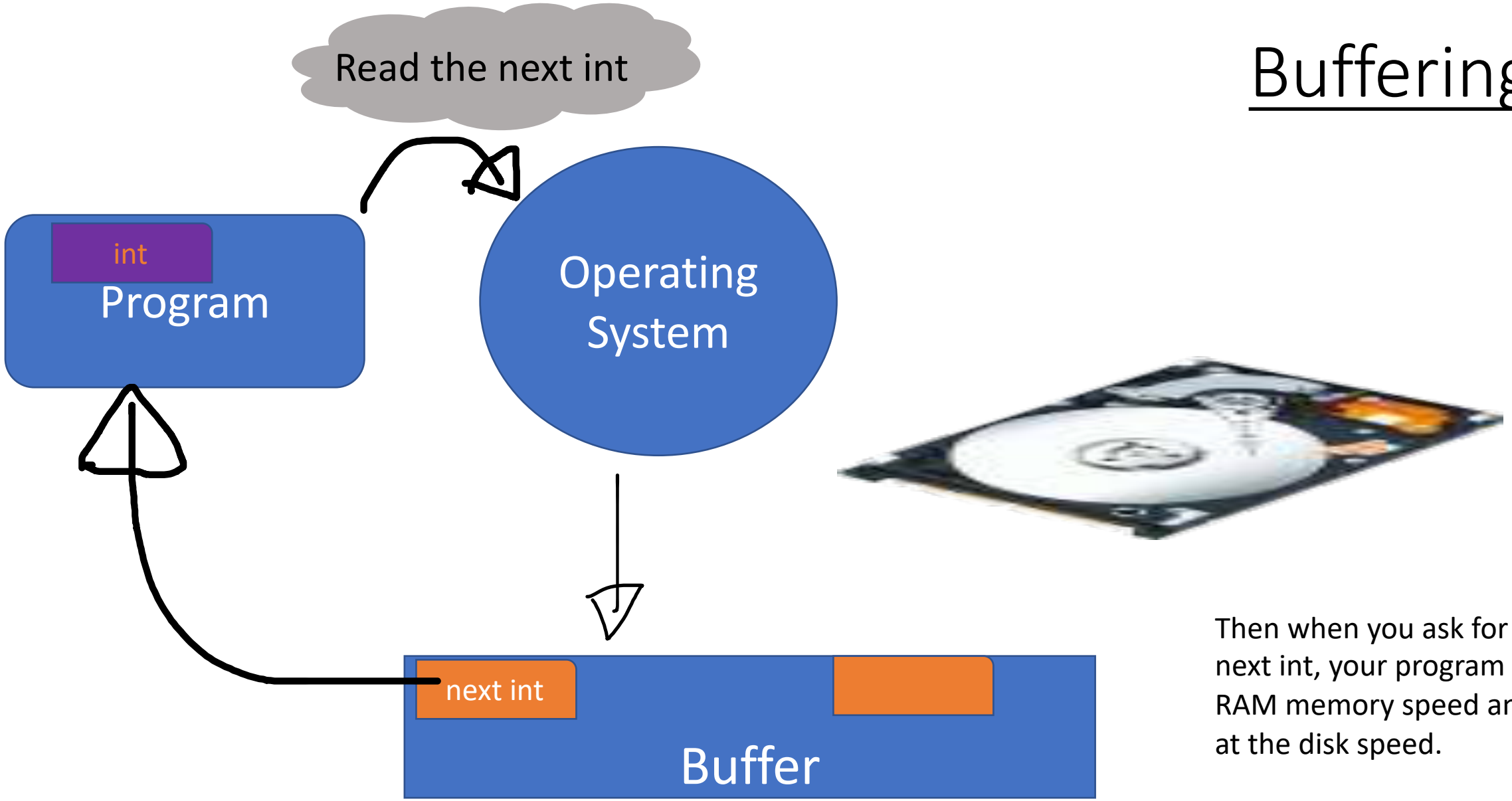
```
import java.io.*
```

Buffering



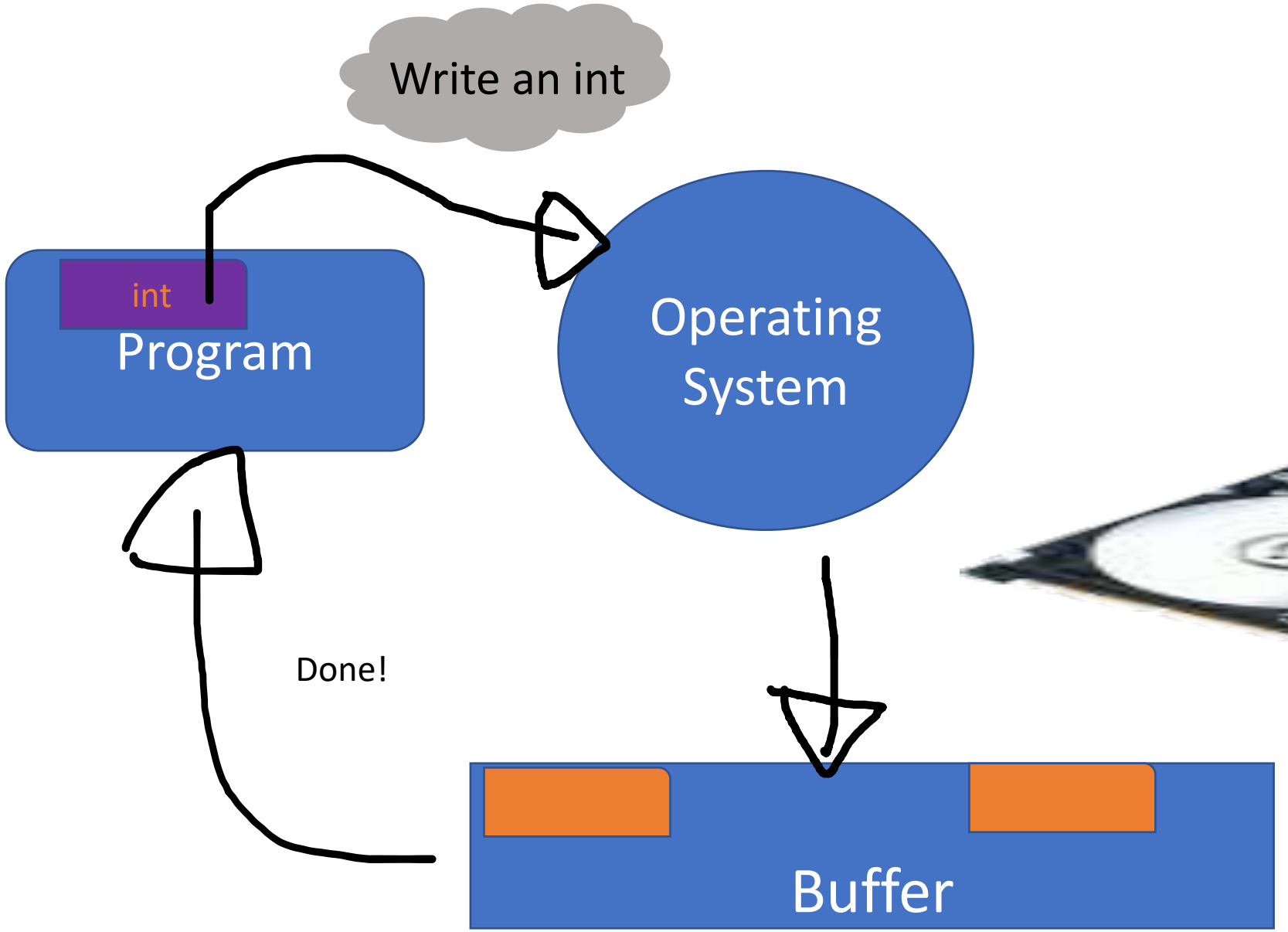
The concept of a buffer is an important one: disks are very slow compared to RAM. A buffered reader will read some chunks (more than just the next piece).

Buffering



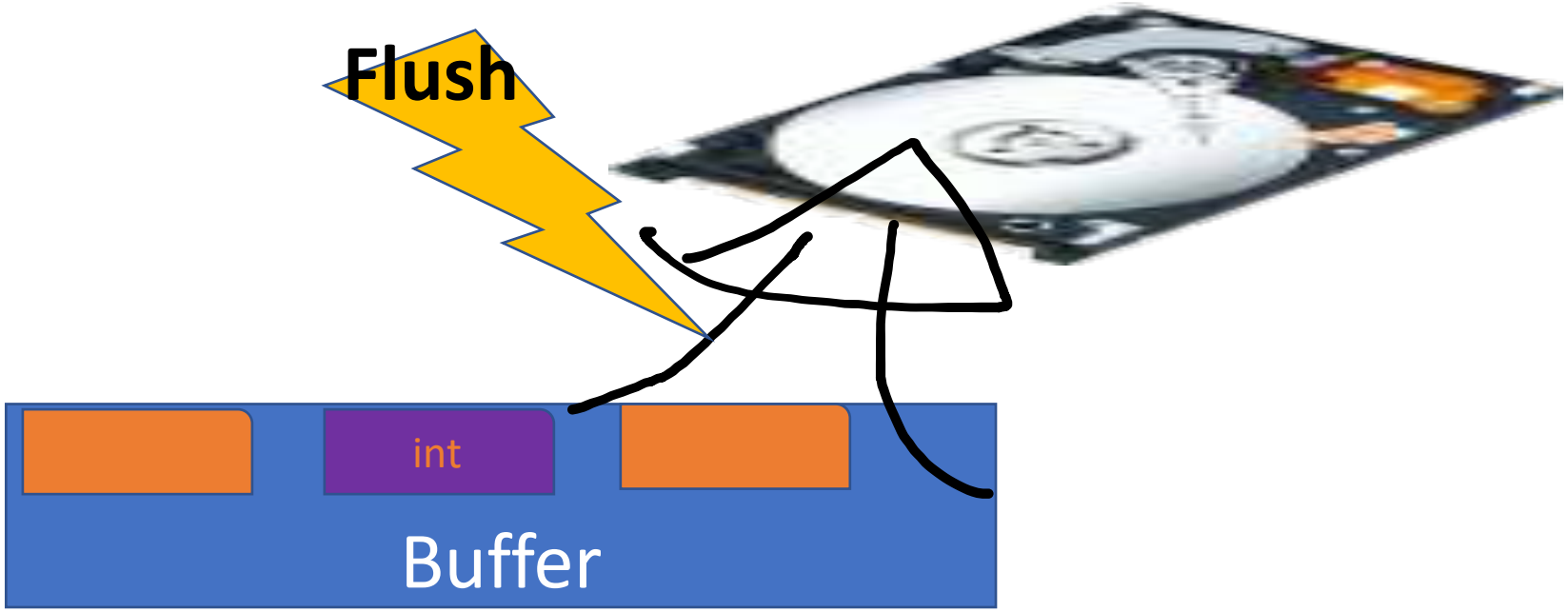
Then when you ask for the next int, your program runs at RAM memory speed and not at the disk speed.

Buffering



In the same way, when you write data, it will be copied to memory, which will be much faster than writing to disk.

Buffering



Data will be bulk transferred to the disk when the buffer is full or you close the file (or you call a method to explicitly flush the buffers).

Buffering

Data will be bulk transferred to the disk when the buffer is full or you close the file (or you call a method to explicitly flush the buffers).

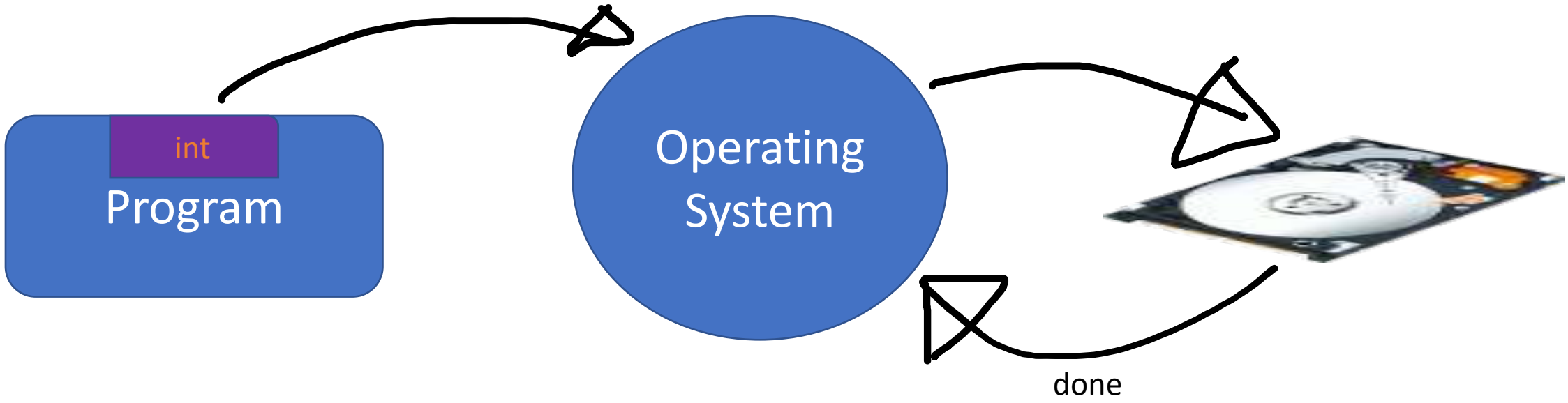
Buffering

- What happens if the system restarts after a crash?

The answer of course is that what was in buffers is lost. Not a problem when reading, big problem when writing. It's not always easy to know what to replay.

No Buffering (I/O direct to disk)

Performance: Should used buffered operations 99.9% of the time. Use unbuffered operations just when writing safely is a critical concern (or for debugging).



Instead can use slow unbuffered operations:

- Used for writing critical information
- Logs
- Messaging

Performance - copying a 11M CSV file char by char

- Test on Mac (internal SSD)*

Unbuffered	approx. 34.5s
Buffered	approx. 1s

- Test on Mac (External USB key)*

Unbuffered	approx. 36s
Buffered	approx. 1.4s

- Consider if its worth it: how often does your computer crash? How bad would it be to rerun the program after restart?

*Tests run by Stephane Faroult in previous years

Sometimes hard to know whats going on…

- Big disk systems have their own battery protected buffers (also called a cache)

Especially with high-end storage you rarely have one level of buffering (in which case unbuffered wouldn't be what it seems). It's a bit hard sometimes to know if the data is on disk or not, and the Cloud doesn't make it any simpler.

Unbuffered

```
InputStream in = null;
OutputStream out = null;


in = new InputStream(...);
out = new OutputStream(...);
```

The basis for all binary Input/Output operations are InputStreams and OutputStreams, which are unbuffered.

One thing that should not be forgotten with file operations if that it's probably the part of a program where everything can fail.

Lots of things can go wrong...

Wrong file directory

Not allowed permissions

Content not as expected

Hardware problem (rare)

Unbuffered

So everything should really be done with exception handling: either with a ”try with resources” or a “finally” block to make sure files are cleanly closed and not left in a “corrupted” state.

```
    InputStream in = null;
    OutputStream out = null;

    try {
        in = new InputStream(...);
        out = new OutputStream(...);

        ...
        } catch (...)  {

        } finally {
            if (in != null)  {
                try {
                    in.close();
                } catch (IOException e)
                    // ignore
                }

    }
}
```

Important: flush everything and close

Unbuffered

```
    FileInputStream in = null;
    FileOutputStream out = null;

    try {
        in = new FileInputStream("filename");
        out = new FileOutputStream("filename");


        } catch (...)  {

        } finally {
            ...
    }
}
```

Looks in the current directory unless you provide a full path

File location is always a practical problem. Use reflection and properties files.

Buffered

To turn an unbuffered stream into a buffered one, just wrap the call to the stream constructor in a call to a buffered stream constructor.

```
    BufferedInputStream in = null;
    BufferedOutputStream out = null;

    try {
        in = new BufferedInputStream(new InputStream(...);
        out = new BufferedOutputStream(new OutputStream(...));


        } catch (...)  {

        } finally {
            ...
    }
}
```

Next: Byte and Character Streams + Object Serialization