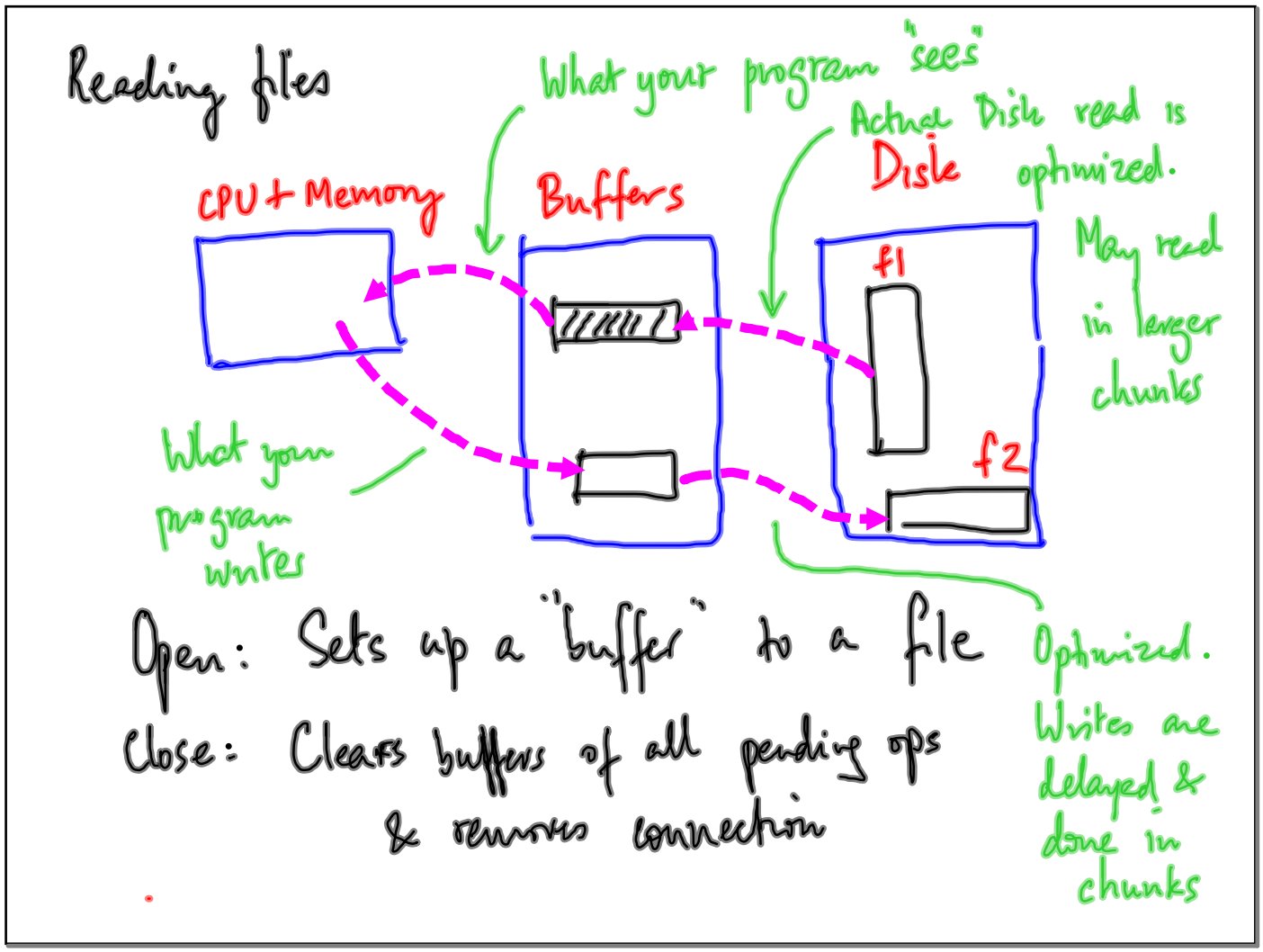
Reading files
What your program "sees"
Actual Disk read is optimized.
CPU + Memory
Buffers
Disk
f1
f2
May read in larger chunks
What your program writes
Optimized. Writes are delayed & done in chunks
Open: Sets up a "buffer" to a file
Close: Clears buffers of all pending ops & removes connection

File handles

Special datatype – name that refers to a buffer

```
f = open(filename, mode)
```

filename: strings, e.g. 'xyz.txt'

mode:
- 'r' read
- 'w' write (overwrite)
- 'a' append

```
f.close()
```

Disconnects file from handle
Flushes buffer if needed

Actual reading/writing w.r.t. file handles

Reading

`x = f.read()` ⟶ All of the file is returned as a single string

`f.read(n)` Reads n characters

Returns '' at end of file

Each successive read starts where previous one ended - close, open to reset position

f.readline() return a string upto (and including) next '\n'

f.readlines() returns each line as a string, collected in a list

```
for l in f.readlines():
    ...
```

Does not work

```
for l in f:
    ...
```

✓

Writing a file

`f.write(x)` — x must be a string. Write at current position. Must insert '\n' ourselves to create new lines.

`f.flush()` — Writes all pending stuff in buffer onto disk (f.close() flushes buffer)

`f.writelines(l)` — Write a list of strings. Must insert '\n' ourselves

File input/output typically generates errors

open('xyz.txt', 'r') fails with IOError

if xyz.txt does not exist

Similarly 'w' will fail if disk is full etc

f.write()

Typically need try.. except.. for file I/O

Stripping '\n'

```
x = f.readline()
```

I want x without trailing \n

```
x = x[:-1]
```

```
x.strip()
```

← Removes all leading & trailing white space ' ', '\t', '\n'

↓

returns a new string

```
x.lstrip(), x.rstrip()
```

Another useful string function

s.split() - splits s into a list, separating at white space

Use with dictionary to "process" a text file

s.split("ab")

↑ split on "ab" instead of white space