9. I/0 and HDD/SSD

Memory and I/O buses/interconnect, PCI/USB/SATA, micro-controller, I/O device, programmed I/O, interrupt-based I/O, DMA, I/O instructions (isolated I/O), memory-mapped I/O, I/O stack, block device, storage stack, block addresses, sector, HDD, platter, surface, spindle, RPM, track, cylinder, disk arm, disk head, seek time, rotational delay, SSD, SLC/MLC/TLC, NAND flash, flash translation layer, trim, write amplification, wear levelling, RAID 0/1/5, iops, sequential/random read/write

9.1 Input / Output

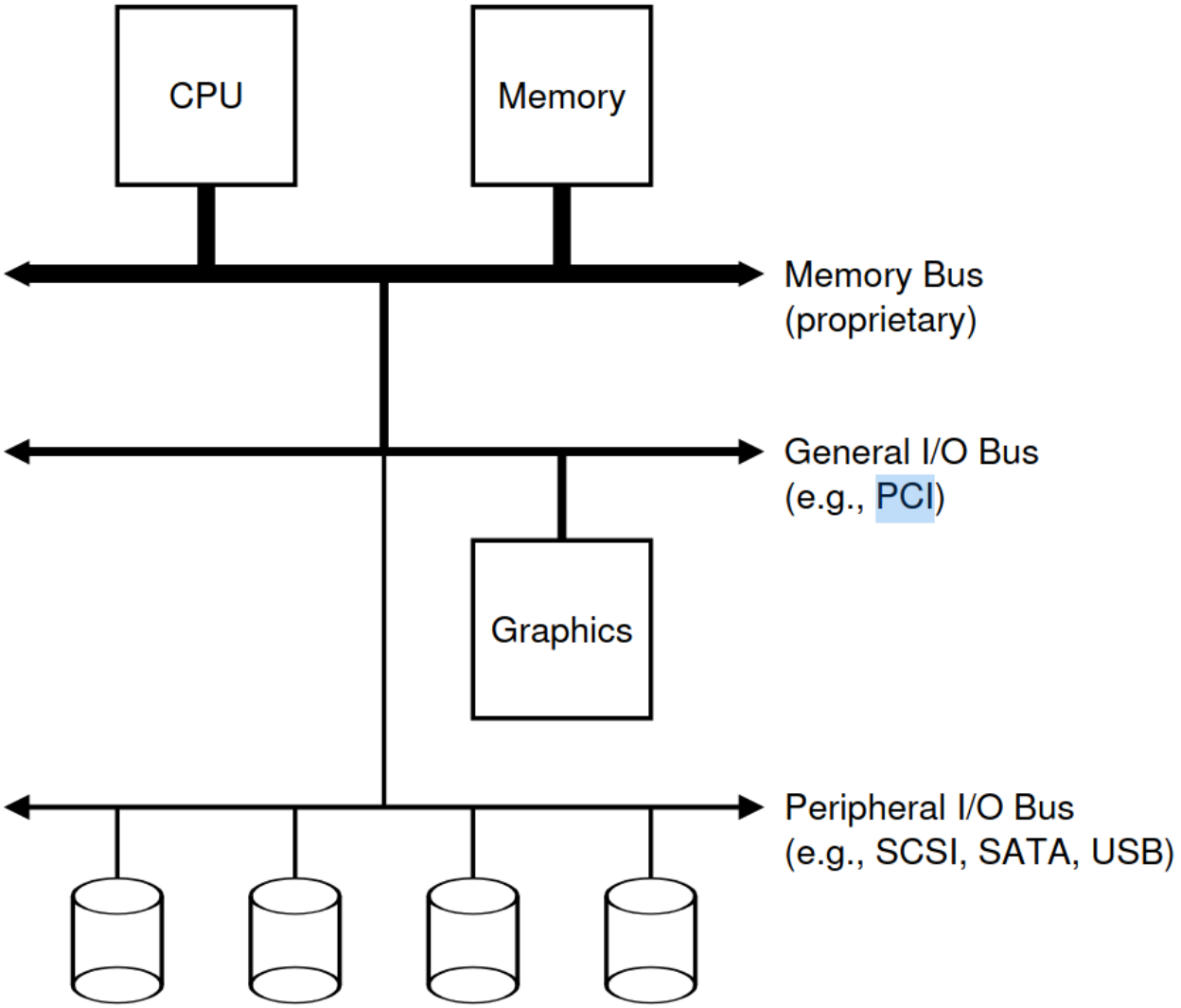


Figure 36.1: **Prototypical System Architecture**

^ Why do we need this hierarchy?

> Because of physics (for high-performance) and cost.

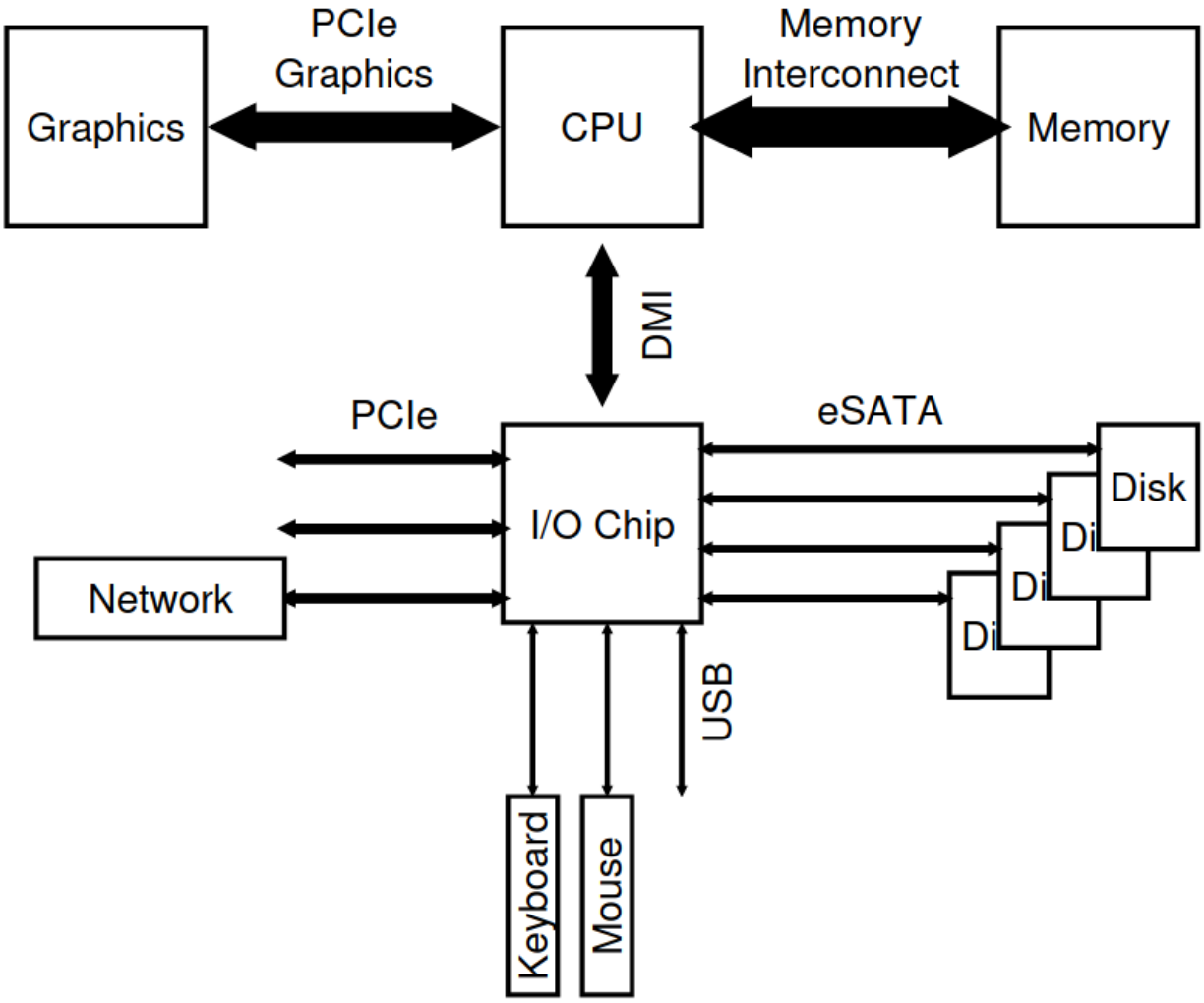


Figure 36.2: **Modern System Architecture**

DMI: Direct Media Interface **eSATA**: External SATA (Serial ATA) (ATA: AT Attachment)

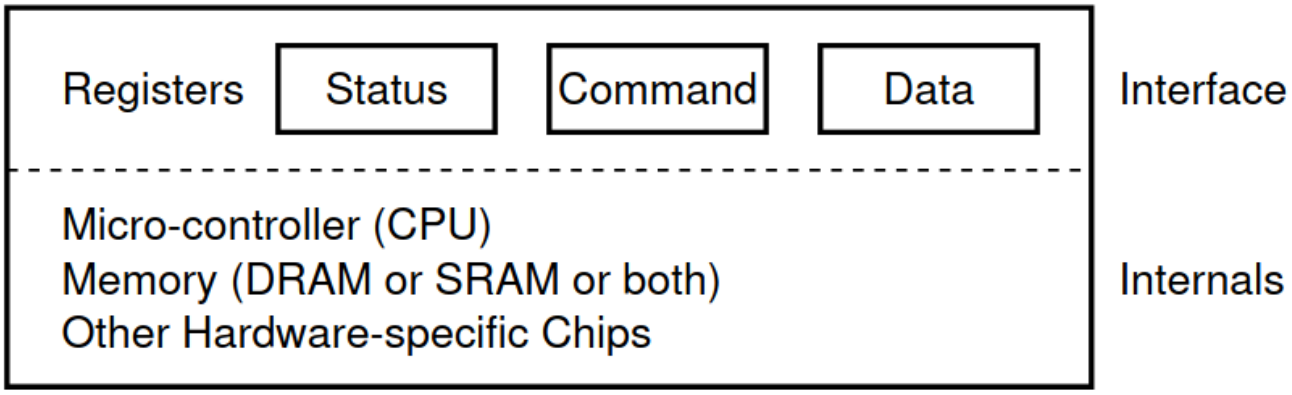


Figure 36.3: **A Canonical Device**

^ The first is the **hardware interface** that it provides. ^ Second is the **internals**, that is implement-specific (containing firmware etc)

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

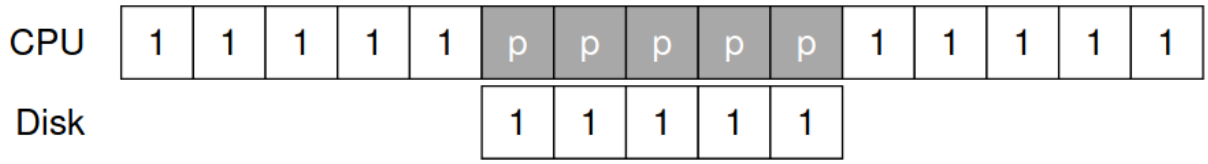
^ Typical how the OS communicates with the device.

When the main CPU is involved in the data movement, it is called **programmed I/O**.

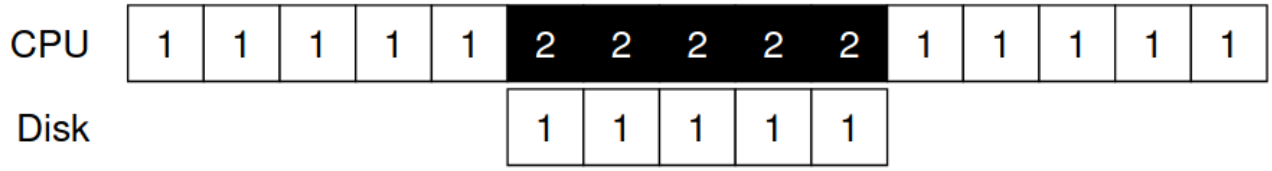
^ This implementation (polling) is not efficient, because the CPU is waiting for the device

> Instead, we can use **interrupts**, which lets the calling process to sleep.

That is, we go from this:



To this:

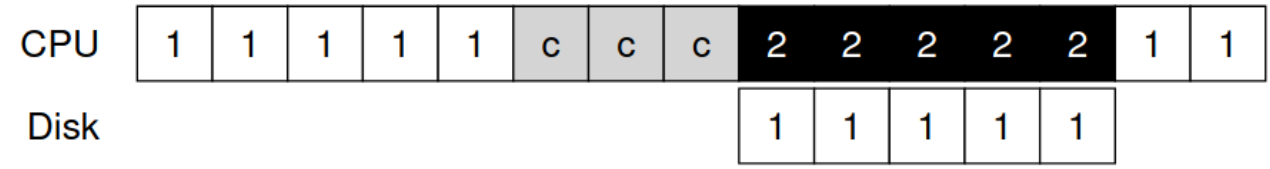


However:

> If a device is fast, it may be best to *poll*; if it is slow, *interrupts*, which allows for overlaps

And if the device is both fast and slow, it may be best to use a **hybrid** approach where

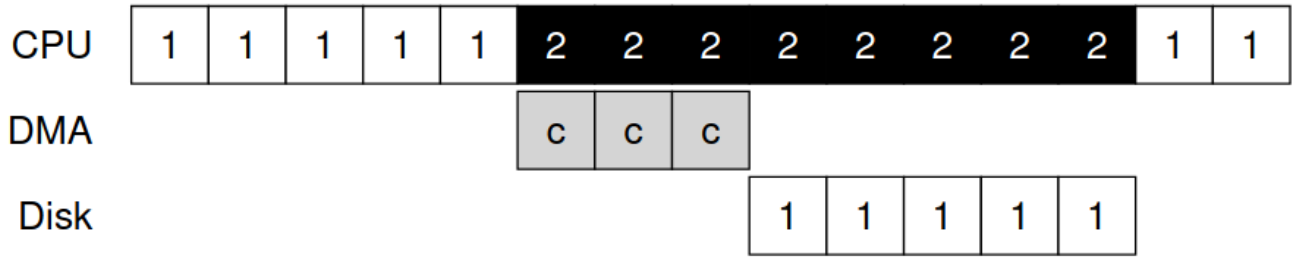
> Poll for a little, if it's not finished, then sleep and wait for an interrupt.



^ When the CPU is moving data to and from Memory, it's quite slow

Instead we can use **DMA** (Direct Memory Access), where the device can directly access the memory

With DMA:



It works as follows:

1. The CPU tells the DMA engine where data lives in memory, how much data to move, and which device to send it to.
2. When the DMA is complete, the DMA controller raises an interrupt and the OS knows the transfer is complete.

How to communicate with the device?

1. The oldest way is to use **I/O instructions** (isolated I/O) (in/out instructions)
    - use `in` and `out` instructions with an address space based on ports (similar to TCP/UDP ports)
2. **Memory-mapped I/O**:
    - use physical addresses (those not used by RAM) and map those to registers on I/O-devices, then we can reuse instructions like `mov`

How to fit every device into the OS (each with its own interface)

We used the good-old technique called **abstraction**:

- We create a **device driver** for each device

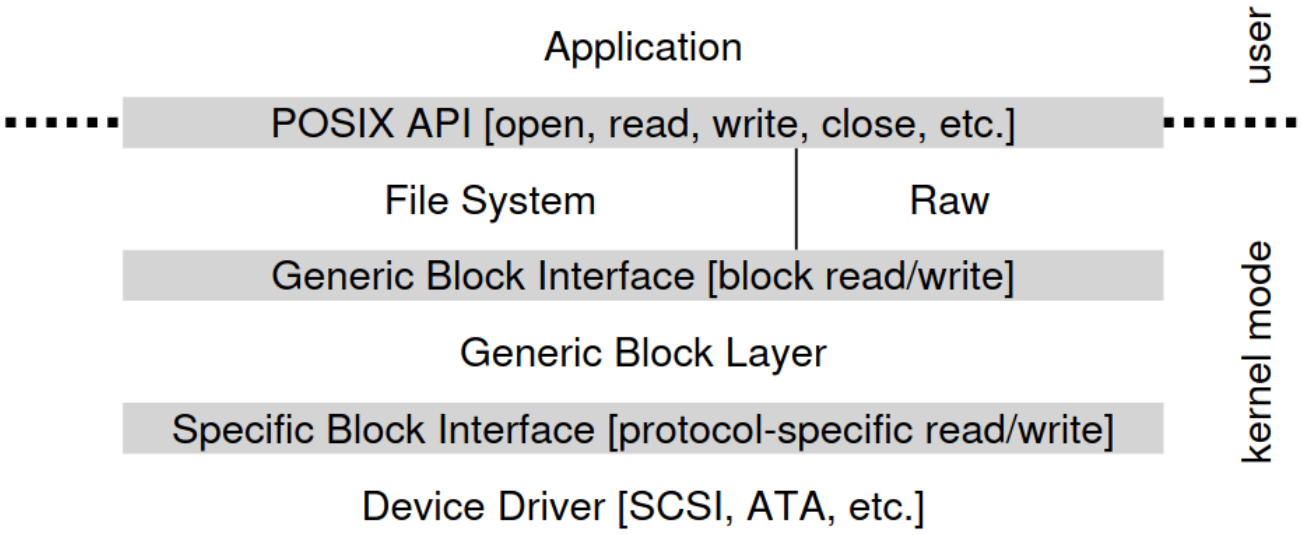


Figure 36.4: **The File System Stack**

9.2 Storage

Addressing

The (HDD) drive consist of a large number of *n* **sectors** (512 bytes blocks) that are numbered from 0 to *n-1*.

This is the **address space** of the drive.

9.2.1 HDD

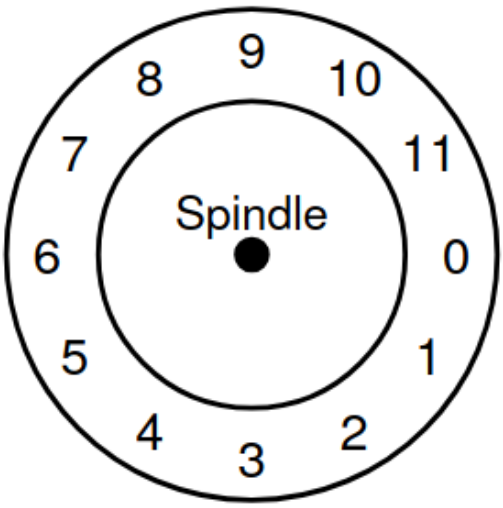


Figure 37.1: **A Disk With Just A Single Track**

A HDD consists of:

- platter with surface grouped in a spindle
- rotation measured in RPM

- a circle on a surface is a track, the set of all tracks above each other is a cylinder
- a disk arm accesses a sector with its disk head
- each platter have two surfaces, there are many platters and each platter have a disk arm for top and bottom surface
- e.g. eight platters with two surfaces means 16 disk arms (they all move to- gether, not independently)

HDD Access Times

- **Seek time**: time to move the disk arm to the right track
- **Rotational delay**: time to wait for the sector to rotate under the head
- **Accessing Sectors**: time to read/write the sector

Total time = $T_{I/O} = T_{seek} + T_{rotational} + T_{transfer}$

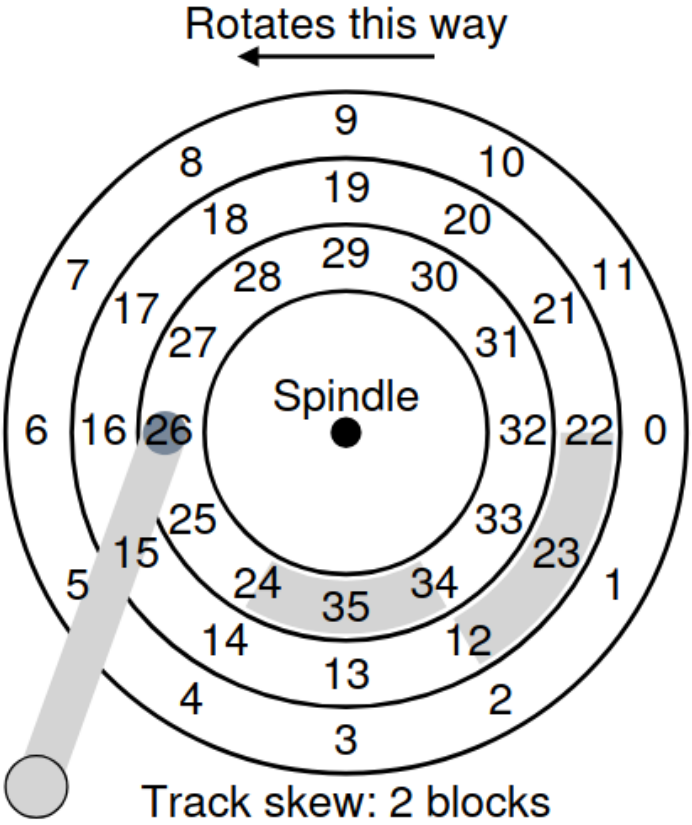


Figure 37.4: **Three Tracks: Track Skew Of 2**

Calculations

Compute $T_{rotation}$ for a 7200 RPM:

$$T_{rotation} = \frac{1minute}{7200rot} \cdot \frac{60seconds}{1minute} \cdot \frac{1000ms}{1seconds} = \frac{8.33ms}{rot}$$

Given transfer rate of 100 MB/s, calculate how long it takes to transfer 512 KB block in ms:

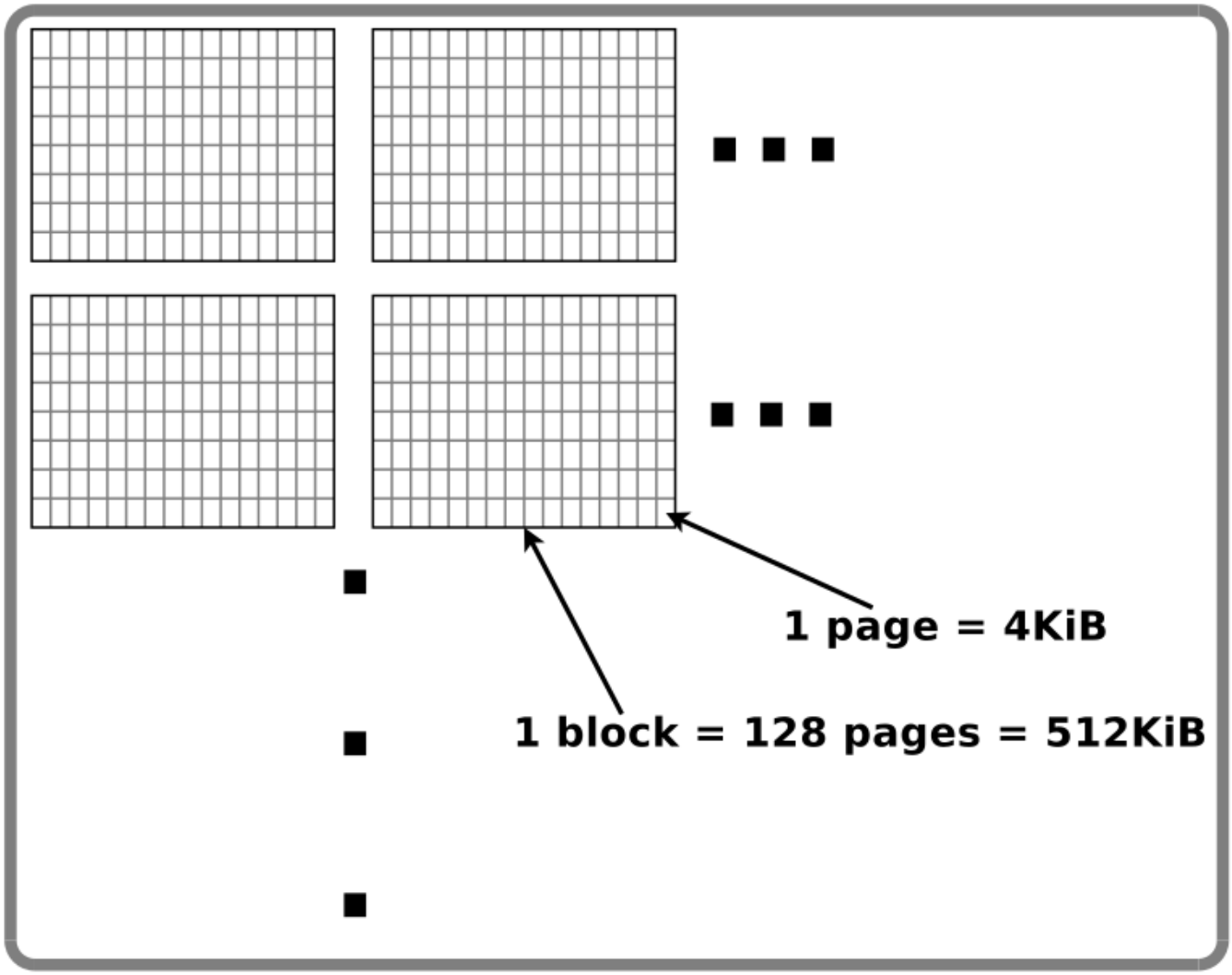
$$\frac{ms}{1request} = \frac{512KB}{1request} \cdot \frac{1MB}{1024KB} \cdot \frac{1s}{100MB} \cdot \frac{1000ms}{1second} = \frac{5ms}{request}$$

E.g. if 1MB per track, rotation time 8.33ms (7200rpm) (have to divide by two since on average we have to rotate a platter half a round to find our data), average seek time 5ms and block size 4KB:

$$T_{I/O} = 5\,\text{ms} + \frac{8.33\,\text{ms}}{2} + (\frac{4\,\text{kB}}{1\,\text{MB}} \times 8.33\,\text{ms}) = 9.20\,\text{ms}$$

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

9.2.2 SSD



Solid State Drive

- No moving parts
- Faster than HDD
- More expensive

> - Limited write cycles

9.2.2.1. Storing bits

- **SLC (Single Level Cell)**: 1 bit per transistor
- **MLC (Multi Level Cell)**: 2 bits per transistor
- **TLC (Triple Level Cell)**: 3 bits per transistor

Three low-level mechanisms

- **Read**: read a page
- **Erase**: (before writing) erase a block
- **Program**: write a sector

		`iiii`	*Initial: pages in block are invalid (`i`)*
Erase()	→	`EEEE`	*State of pages in block set to erased (`E`)*
Program(0)	→	`VEEE`	*Program page 0; state set to valid (`V`)*
Program(0)	→	**error**	*Cannot re-program page after programming*
Program(1)	→	`VVEE`	*Program page 1*
Erase()	→	`EEEE`	*Contents erased; all pages programmable*

Device	Read (μs)	Program (μs)	Erase (μs)
SLC	25	200-300	1500-2000
MLC	50	600-900	~3000
TLC	~75	~900-1350	~4500

Figure 44.2: **Raw Flash Performance Characteristics**

Major issues with SSDs

- **Wear out**: limited number of write cycles
- **Disturbance**: when writing to a cell, the surrounding cells may be disturbed

---

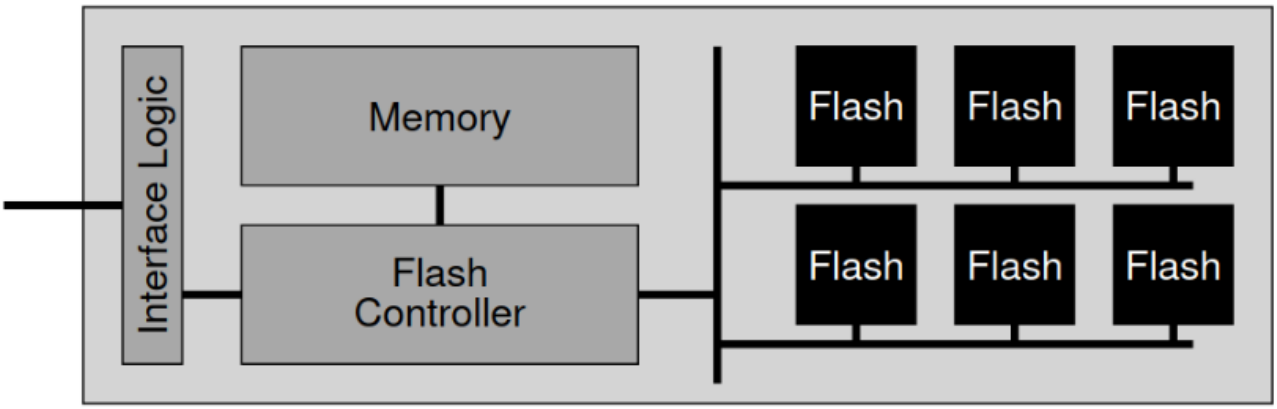


Figure 44.3: **A Flash-based SSD: Logical Diagram**

- **FTL**: Flash Translation Layer (maps logical addresses to physical addresses)
  - should be fast and reliable

For even better performance, we can use **parallelism** of the flash chips.

Another performance goal is to reduce **write amplification** (the amount of data written to the flash memory is more than the amount of data written by the host)

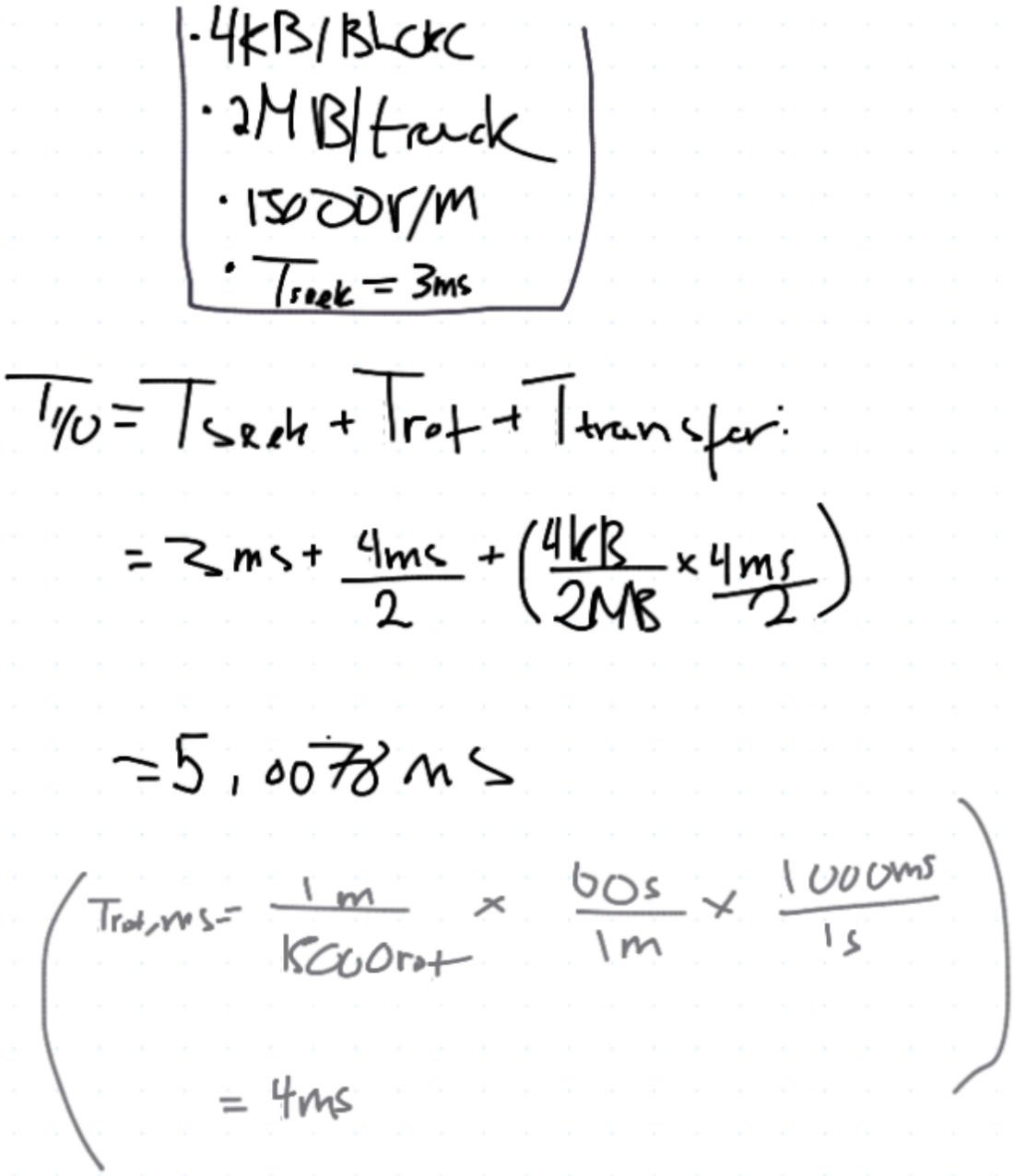
wear leveling: spread out the writes over the entire disk to reduce wear out (done so by the *FTL*)

direct mapped FTL: is bad in terms of performance and reliability.

Summary

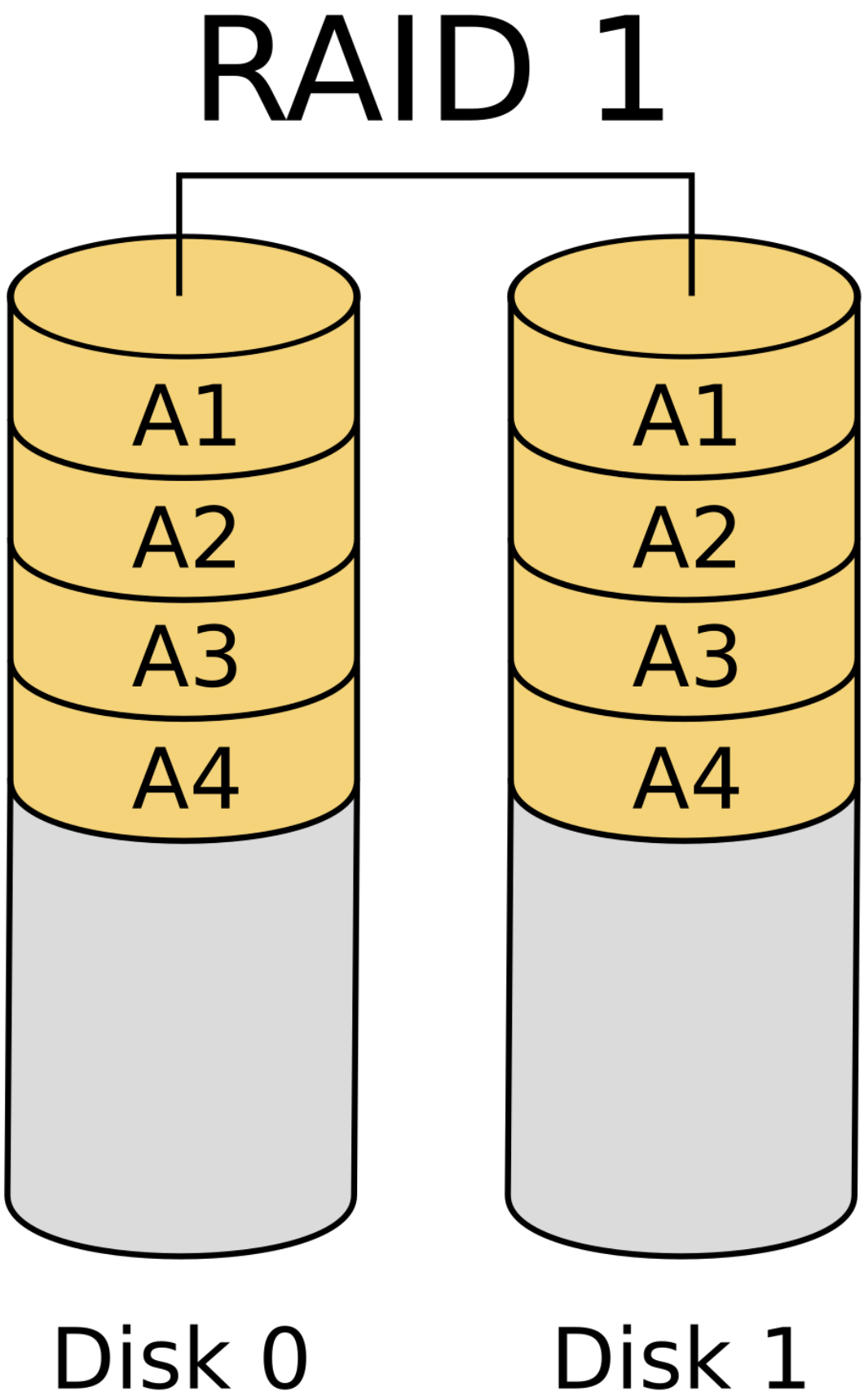
Review Questions and Problems

1. What is the difference between memory-mapped I/O and isolated I/O?
   - Memory-mapped I/O uses the same address space as the memory, while isolated I/O uses a separate address space.
   - Memory-mapped uses the same instructions as memory, while isolated I/O uses special I/O instructions.
2. (KEY PROBLEM) On a hard drive, how many bytes are in a sector? How long would you estimate it takes to fetch a 4KB block at a random location on the disk if the disk has 2MB per track, is 15000rpm and has an average seek time of 3ms?
   - 512 bytes or 4KB

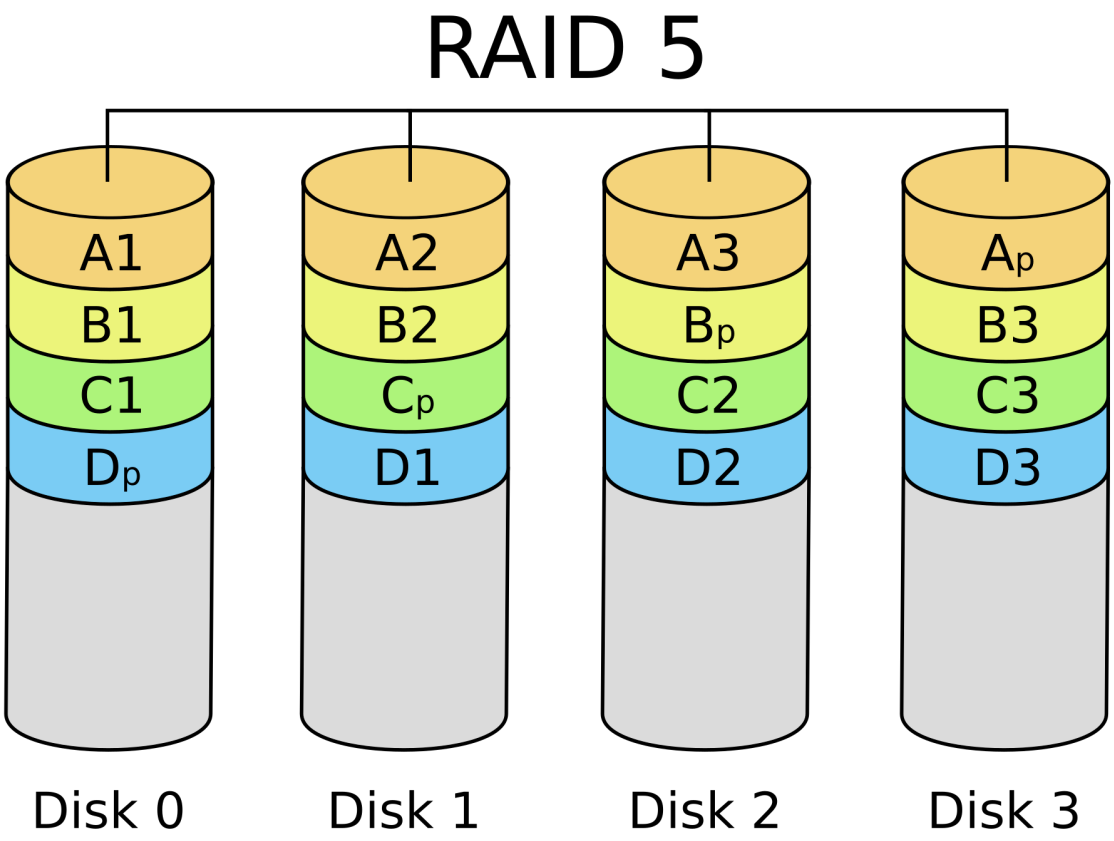


3. What is the benefit of organizing disks in a RAID? How are the disks organized at RAID level 1? How are the disks organized at RAID level 5?
    - Share data across multiple disks for performance and reliability.

- RAID 1: Mirrored



- RAID 5: Striped with parity



4. (KEY PROBLEM) Explain the difference between HDD and SSD in terms of reading, writing/overwriting and deleting files. What is the point of the TRIM command?

Reading, writing and erasing are the same (sector on HDD, and pages on SSD), the big one the difference is overwritten since SSDs must erase the contents of the cells before they can has been written to, and deletion can only take place on entire blocks. Since HDD and SSD do not understand what can be deleted before the operating system tries to overwrite something takes overwriting long on the SSD, this should be solved with the TRIM command, which is a way to The OS will notify the SSD that data may be deleted.

- **TRIM**: Siden HDD og SSD ikke skjønner hva som kan slettes før OSet forsøker overskrive noe tar overskriving lang til p˚a SSD, dette skal løses med TRIM kommandoen som er en m˚ate for OSet ˚a gi SSD'n beskjed om at data kan slettes.

5. Why is it beneficial that data is stored continuously (in sequence) on a HDD? Does this also apply to a SSD? Justify your answer.

    - Because the disk arm can move to the next sector without having to wait for the disk to rotate. This does not apply to SSDs because there is no moving parts.

6. (KEY PROBLEM) Run the command `iostat` on your linux. What is TPS? What is the difference between running just iostat and iostat 1 ?

- TPS: Transactions per second (IOPS)
- `iostat` shows the average since boot, `iostat 1` shows the average for the last second.