University of Stavanger

# I/O Devices (Chapter 36)

How should I/O be integrated into systems?

What are the general mechanisms?

How can we make them efficient?

University of Stavanger

# I/O Devices (Chapter 36)

## 36.1 System Architecture (Generic prototype)

PCI: Peripheral Component Interconnect

SCSI: Small computer System Interface

SATA: Serial Advanced Technology Attachment

USB: Universal Serial Bus

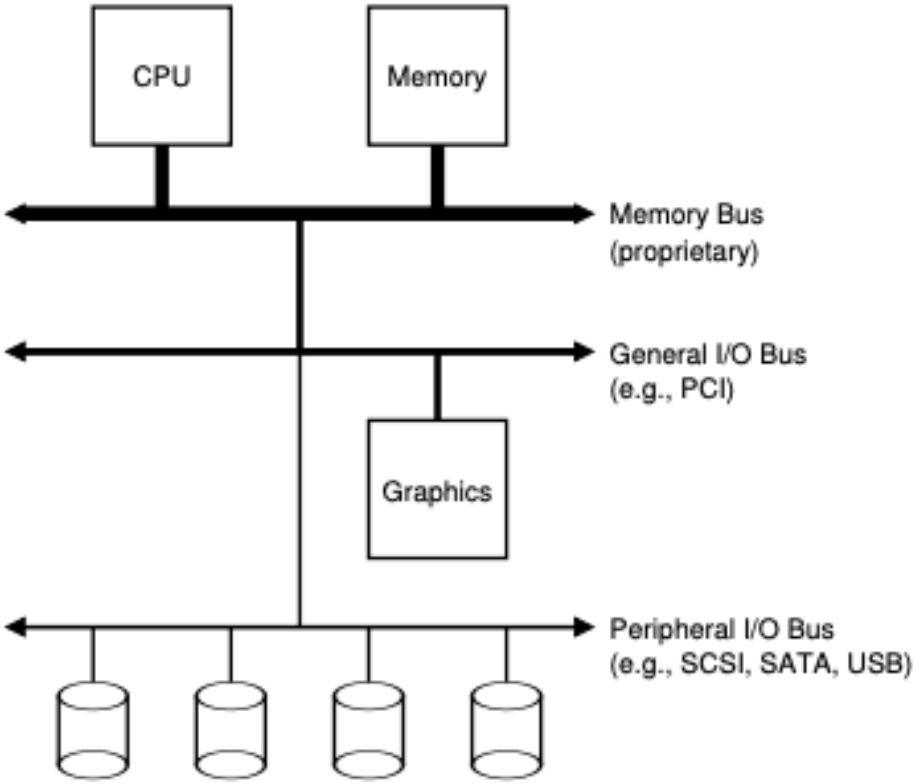



Figure 36.1: **Prototypical System Architecture**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.1 System Architecture Intel's Z270 Chipset

PCIe: Peripheral Component Interconnect Express

eSATA: Express Serial Advanced Technology Attachment

USB: Universal Serial Bus

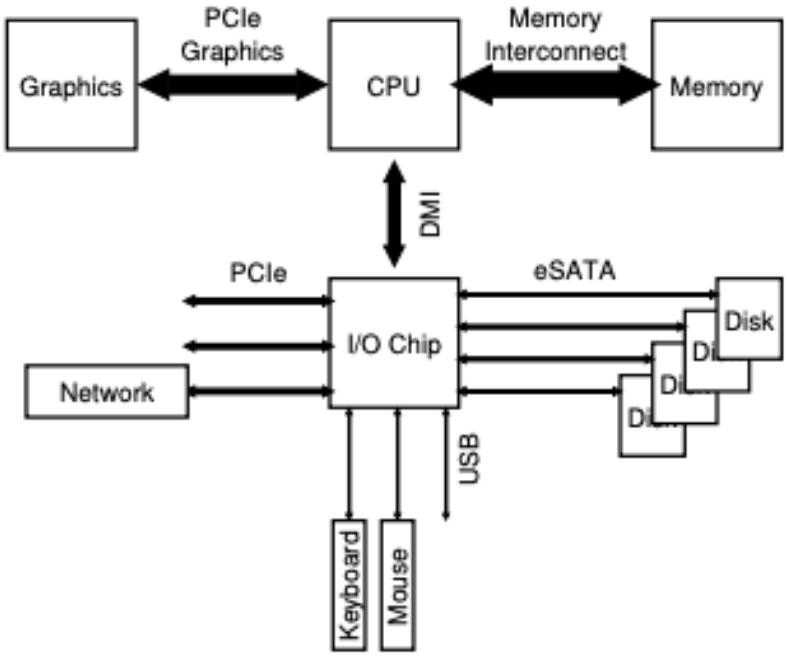



Figure 36.2: **Modern System Architecture**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.2 A Canonical IO Device

Interface: What the device exposes

Internal: Device specific hardware and software

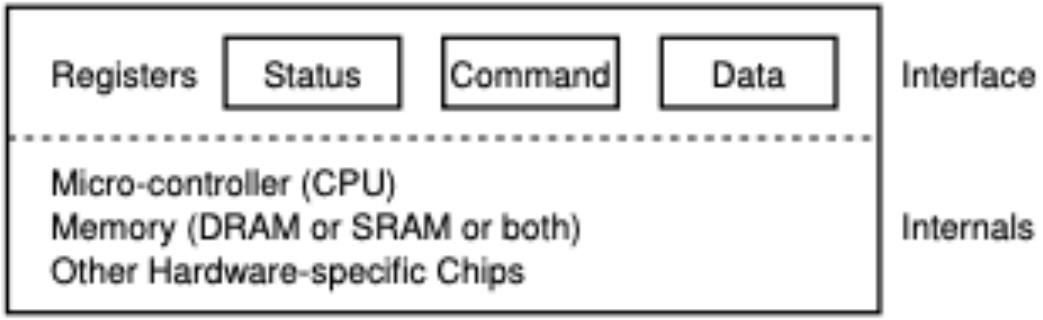



Figure 36.3: **A Canonical Device**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.2 A Canonical IO Device

Story:

- A program p1 wants to send sound to a speaker

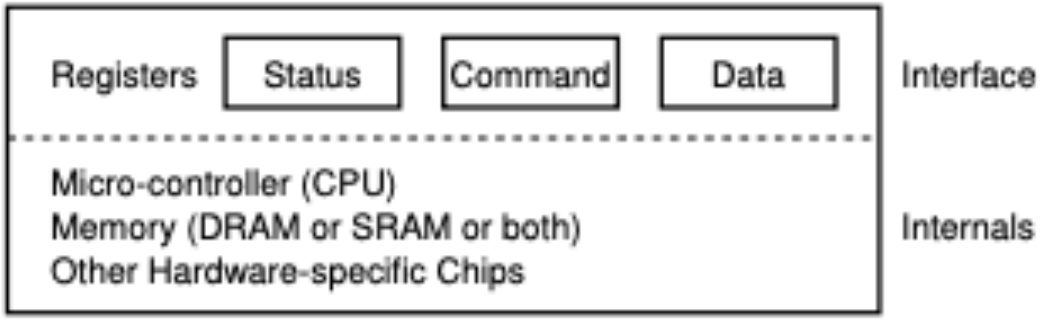



Figure 36.3: **A Canonical Device**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.2 A Canonical IO Device

Story:

- A program p1 wants to send sound to a speaker
- p1 issues a system call
- the system call handler traps in the os
- the os calls that procedure that knows how to communicate with the speaker
- How to communicate with the devices comes later 36.6/7

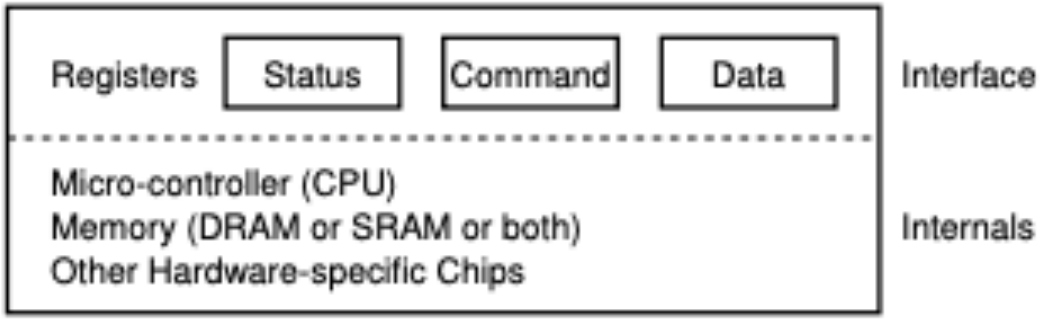



Figure 36.3: **A Canonical Device**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.3 A Canonical Protocol

```
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
```

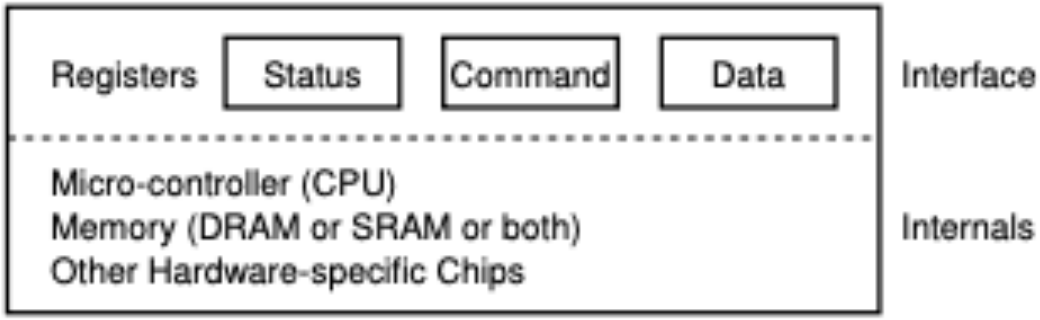



Figure 36.3: **A Canonical Device**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.3 A Canonical Protocol

```
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
```

**What is wrong with this approach ?**

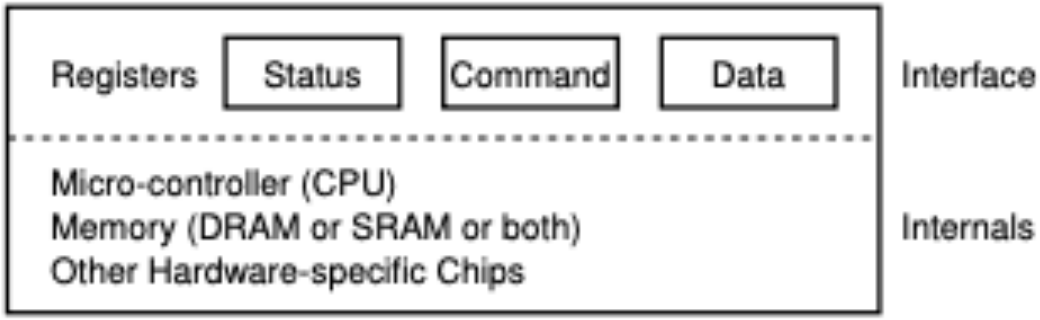



Figure 36.3: **A Canonical Device**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.3 A Canonical Protocol

**Polling for Ready (CPU)**

```
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
```

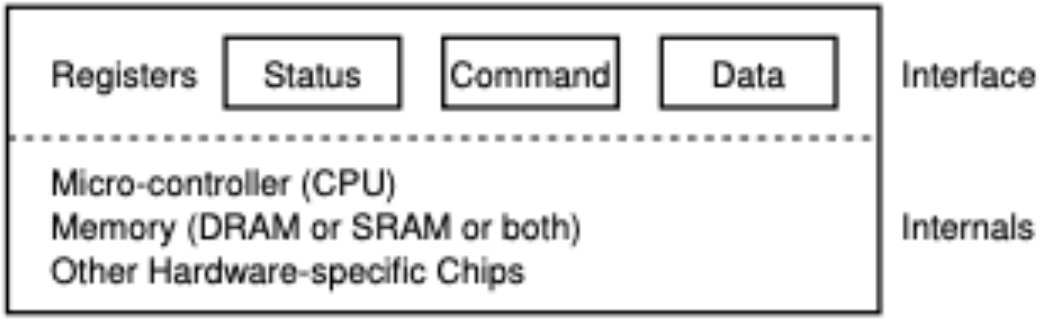



Figure 36.3: **A Canonical Device**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.3 A Canonical Protocol

```
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
```

**Writing (CPU)**

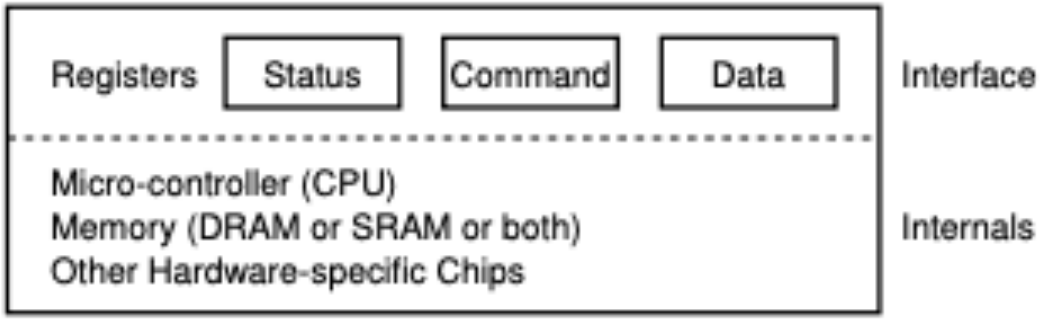



Figure 36.3: **A Canonical Device**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.3 A Canonical Protocol

```
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
```

**Polling again (CPU)**

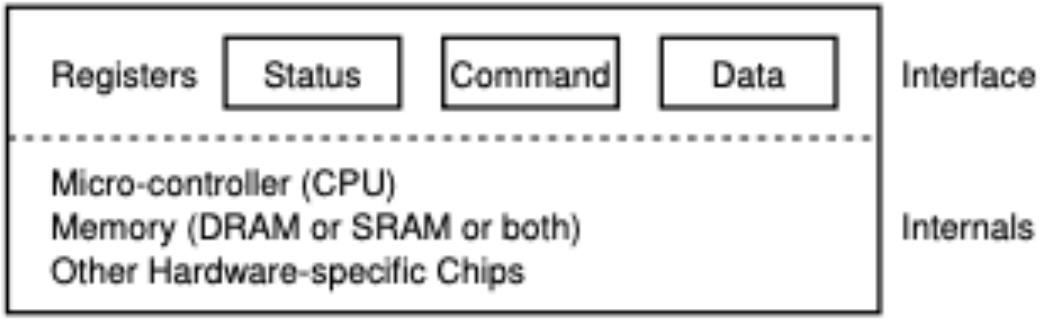



Figure 36.3: **A Canonical Device**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.4 Lowering CPU Overhead with Interrupts

**Go to sleep and continue later**

```
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
```

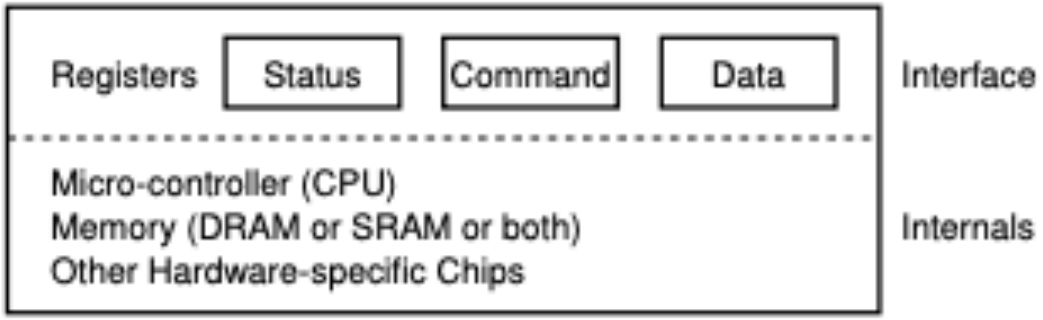



Figure 36.3: **A Canonical Device**

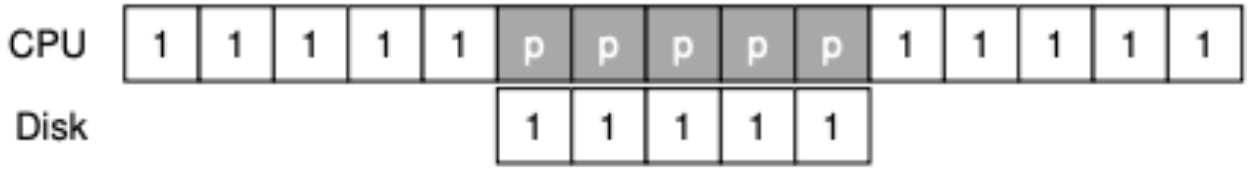
University of Stavanger

# I/O Devices (Chapter 36)

## 36.4 Lowering CPU Overhead with Interrupts

**Without interrupt busy spinning**

```
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
```



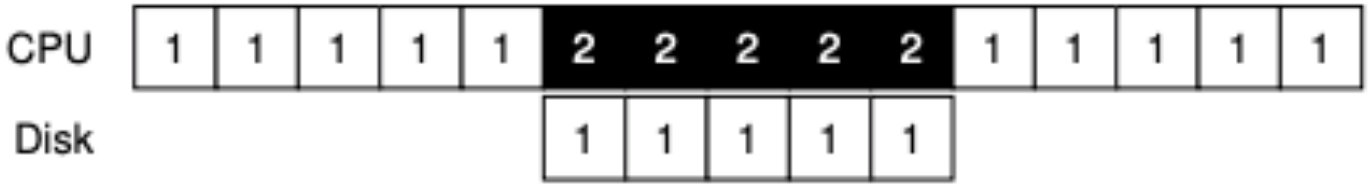
University of Stavanger

# I/O Devices (Chapter 36)

## 36.4 Lowering CPU Overhead with Interrupts

**With interrupt another process can use the CPU**

```
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
```



University of Stavanger

# I/O Devices (Chapter 36)

## 36.4 Lowering CPU Overhead with Interrupts

```
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
```

**However,**

if the device is fast, then no need to go to sleep and wait for interrupt

University of Stavanger

# I/O Devices (Chapter 36)

## 36.4 Lowering CPU Overhead with Interrupts

```
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
```

**However,**

if the device is fast sometimes, spin sometime and then go to sleep

University of Stavanger

# I/O Devices (Chapter 36)

## 36.4 Lowering CPU Overhead with Interrupts

```
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
```

**Issue,**

When many requests (Stream of IO), many interrupts will cause a "livelock" all time goes to processing interrupts

University of Stavanger

# I/O Devices (Chapter 36)

## 36.4 Lowering CPU Overhead with Interrupts

```
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
```

**Issue,**

When many requests (Stream of IO), many interrupts will cause a "livelock" all time goes to processing interrupts

University of Stavanger

# I/O Devices (Chapter 36)

## 36.4 Lowering CPU Overhead with Interrupts

```
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
```

**Trick,**

The device waits a while before sending interrupt signal to inform that it is done "coalescing" several "done" into one.

University of Stavanger

# I/O Devices (Chapter 36)

## 36.5 Direct Memory Access (DMA)

```
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
```

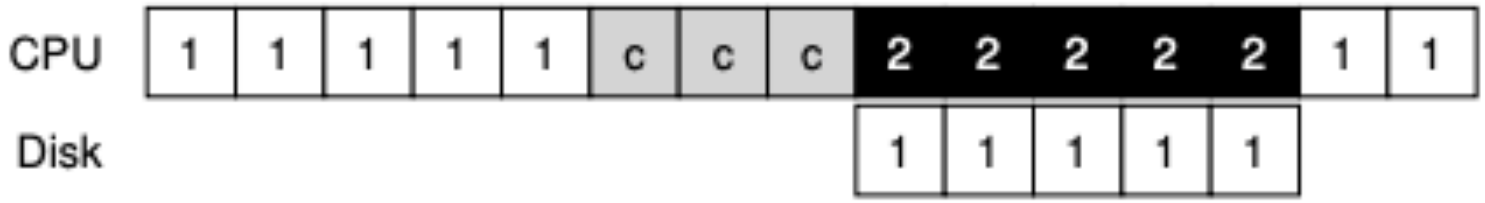
**Problem,**

writing large chunk of data is still a CPU task. Must copy the data from memory to the device one word at the time "not good"

University of Stavanger

# I/O Devices (Chapter 36)

## 36.5 Direct Memory Access (DMA)



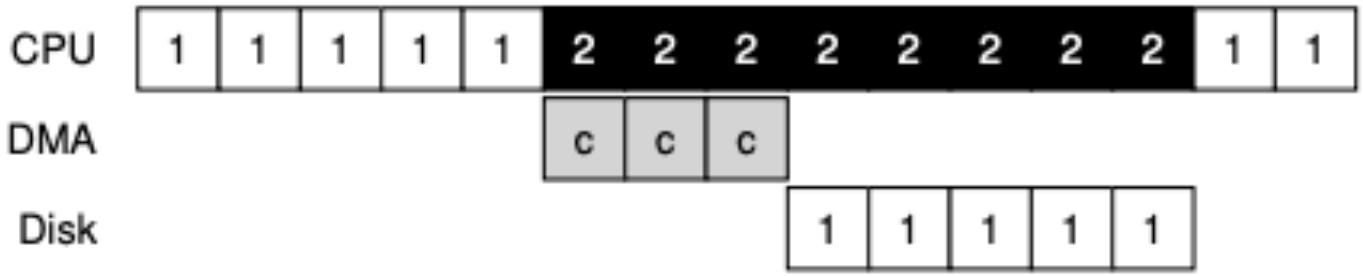

**Problem,**

writing large chunk of data is still a CPU task. Must copy the data from memory to the device one word at the time "not good"

University of Stavanger

# I/O Devices (Chapter 36)

## 36.5 Direct Memory Access (DMA)




**Solution,**

OS (via a system call) tells DMA where the data is in memory, and how much to copy to which device. The device issues an interrupt when done.

University of Stavanger

# I/O Devices (Chapter 36)

36.6 Methods of Device Interaction

Basically how CPU interacts with devices

I/O instructions: uses dedicated commands to communicate with devices

Memory mapped I/O: uses memory operations to communicate with devices

University of Stavanger

# I/O Devices (Chapter 36)

36.6 Methods of Device Interaction

Basically how CPU interacts with devices

Explicit privileged I/O instructions: for example “in” “out” and port (device)

Memory mapped I/O: device registers are seen as memory locations. Load (read) or Store (write) to those locations. The hardware routes that to the device and not to memory

University of Stavanger

# I/O Devices (Chapter 36)

## 36.7 The Device Driver

An abstraction make life easier for the OS, but dealing with the device details

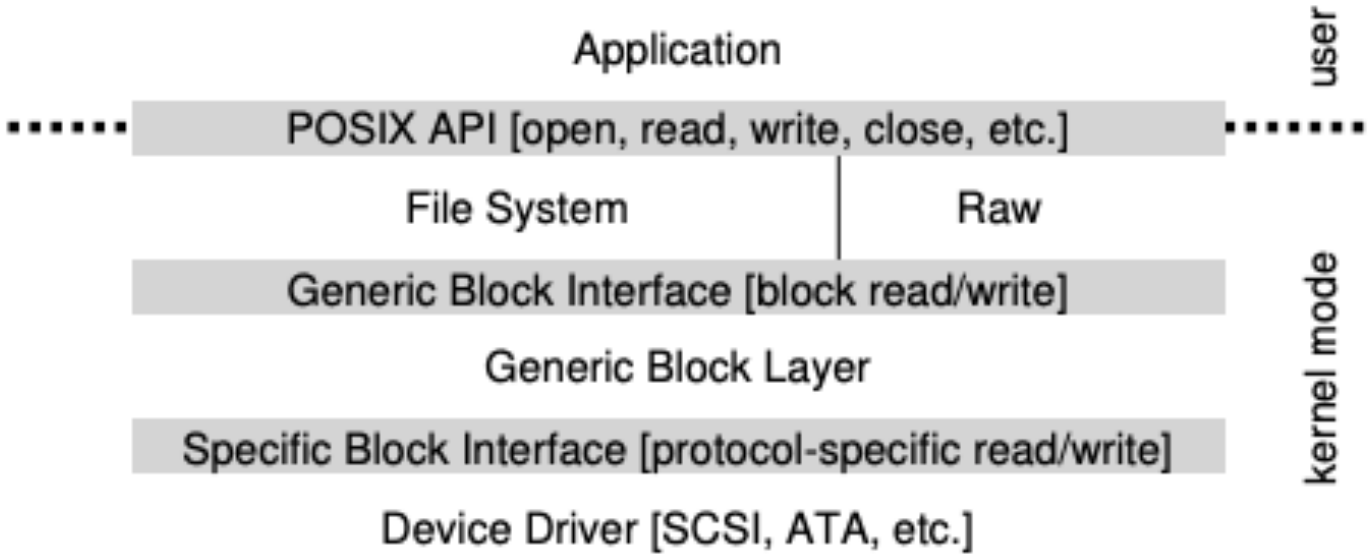



Figure 36.4: **The File System Stack**

University of Stavanger

# I/O Devices (Chapter 36)

## 36.10 Summary

- About how OS interacts with a device
- Two techniques for device efficiency: Interrupt and DMA
  - Interrupt makes sense when device is slow
  - DMA makes sense for large data chunks copying
- Two techniques for the cpu to access devices: explicit I/O instructions or memory-mapped I/O
- Device drivers to make it easier for the os to build os in a device neutral fashion.