Engineering & Computer Science
Syracuse University

CIS 657 – Principles of Operating Systems

Topic: Persistence – I/O Devices

Endadul Hoque

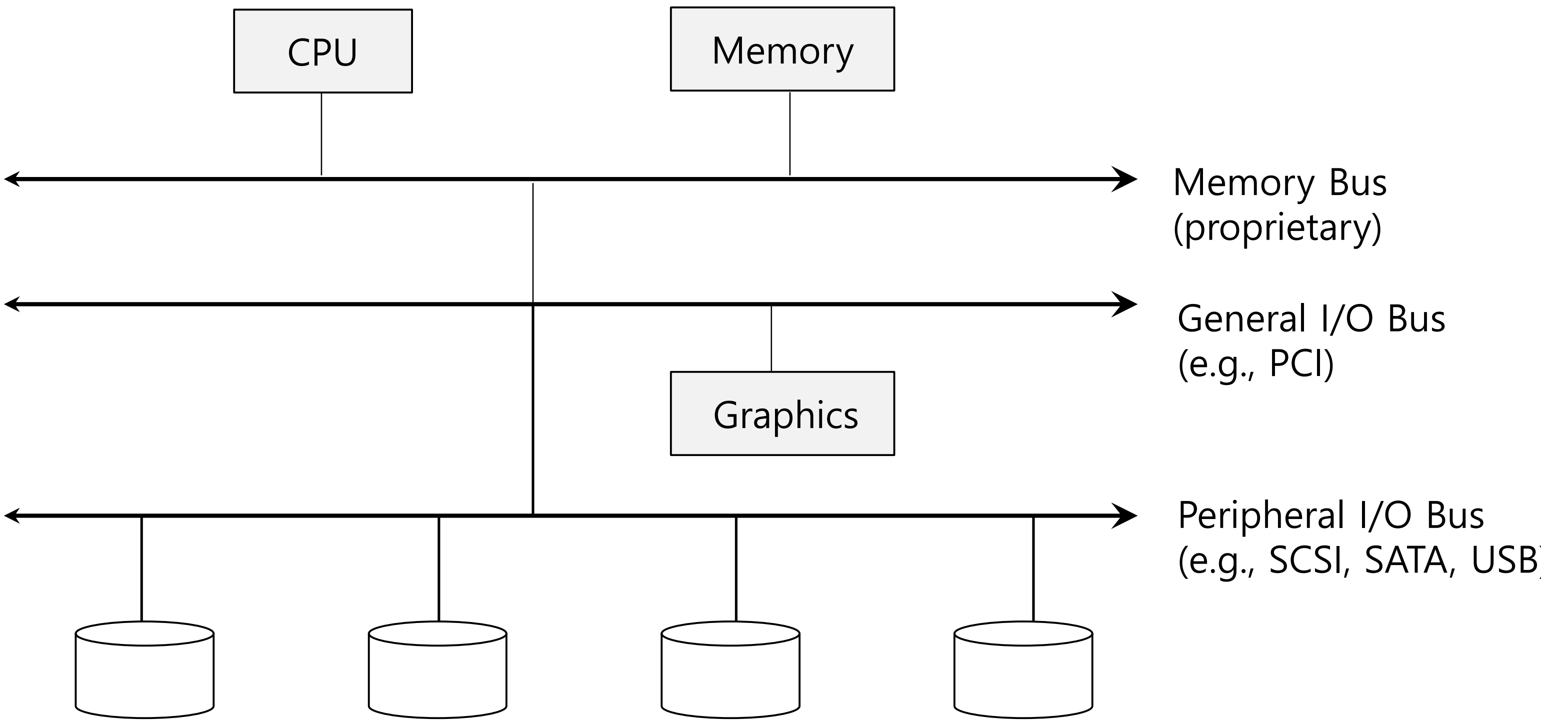
Acknowledgement

- Youjip Won (Hanyang University)
- OSTEP book – by Remzi and Andrea Arpaci-Dusseau (University of Wisconsin)

I/O Devices

- I/O is **critical** to computer system to **interact with systems.**
- Issue :
  - How should I/O be integrated into systems?
  - What are the general mechanisms?
  - How can we make them efficient?

System Architecture



Prototypical System Architecture

CPU is attached to the main memory of the system via some kind of memory bus.

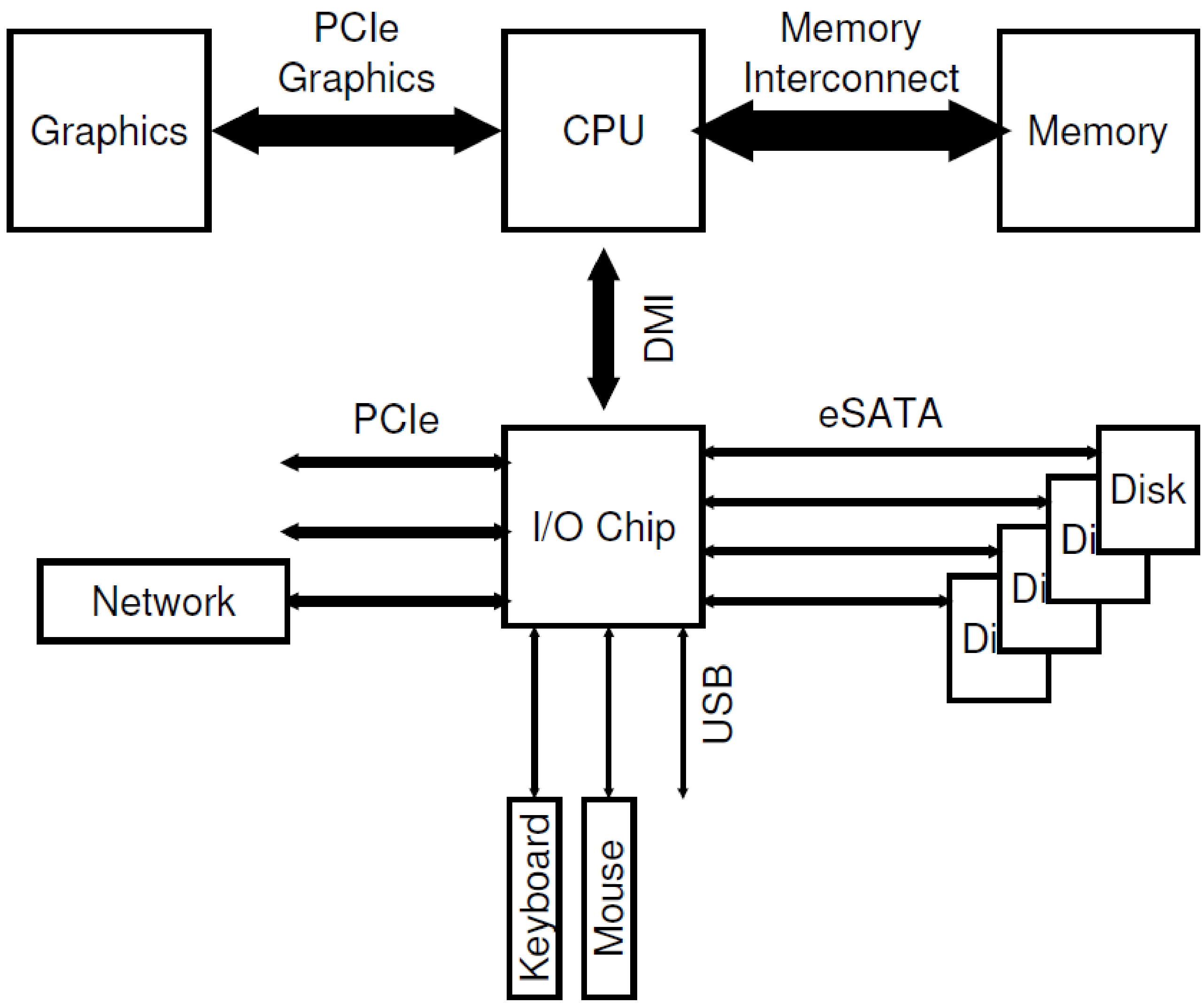
Some devices are connected to the system via a general I/O bus.

System Architecture

- Buses
  - Data paths that are provided to exchange information between CPU(s), RAM, and I/O devices.

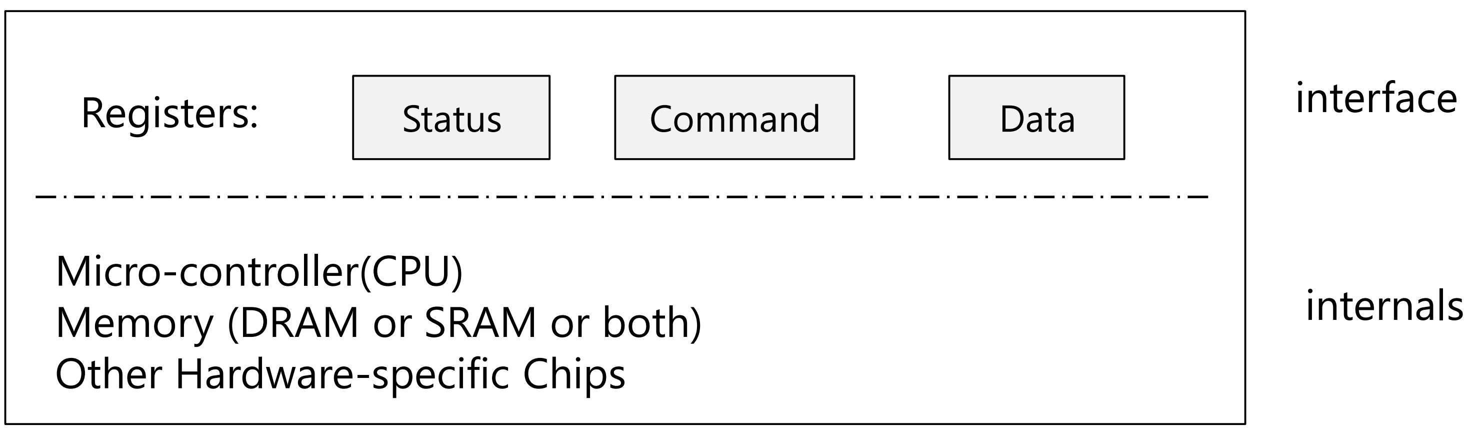
- I/O bus
  - Data path that connects a CPU to an I/O device.
  - I/O bus is connected to I/O device by hardware components: I/O ports, interfaces and device controllers.

Modern System Architecture



Canonical Device

- Canonical Devices has two important components.
  - **Hardware interface** allows the system software to control its operation.
  - **Internals** which is implementation specific.



Canonical Device

Hardware interface of Canonical Device

- **status register**
  - See the current status of the device
- **command register**
  - Tell the device to perform a certain task
- **data register**
  - Pass data to the device, or get data from the device

By reading and writing above three registers,
the operating system can control device behavior.

Hardware interface of Canonical Device

- Typical interaction example

```
while ( STATUS == BUSY)
    ; //wait until device is not busy
write data to data register
write command to command register
    Doing so starts the device and executes the command
while ( STATUS == BUSY)
    ; //wait until device is done with your request
```

Polling

- Operating system waits until the device is ready by **repeatedly** reading the status register.
  - Positive aspect is simple and working.
  - **However, it wastes CPU time just waiting for the device**.
    - Switching to another ready process is better utilizing the CPU.

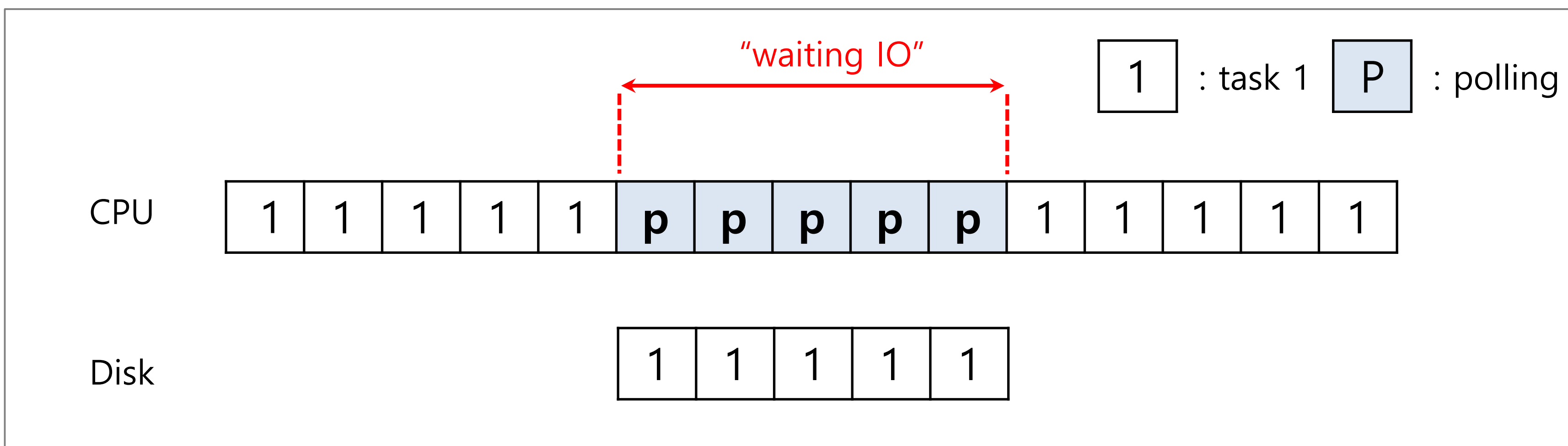


Diagram of CPU utilization by polling

Interrupts

- **Put the I/O request process to sleep** and context switch to another.
- When the device is finished, wake the process waiting for the I/O by **interrupt**.
  - Positive aspect : **CPU and the disk are properly utilized.**

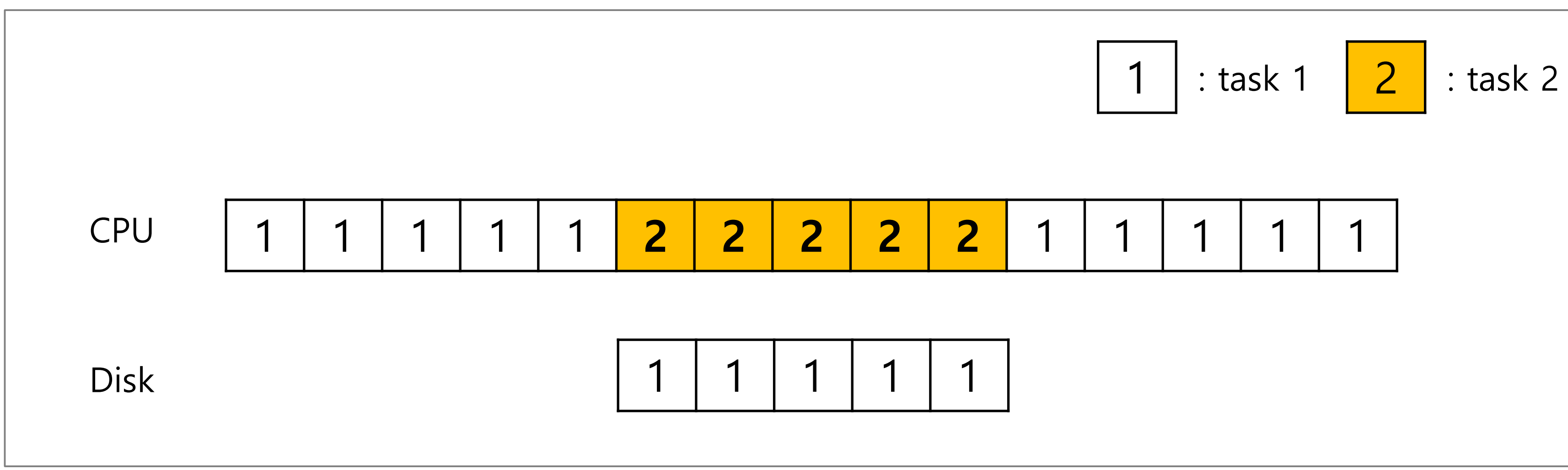


Diagram of CPU utilization by interrupt

Polling vs. Interrupts

- *However,* **"interrupts is not always the best solution"**
  - If, device performs very quickly, interrupt will "slow down" the system.
  - Because **context switch is expensive (switching to another process)**

If a device is fast → poll is best.
If it is slow → interrupt is better.

CPU is once again over-burdened

- CPU **wastes a lot of time** to copy the *a large chunk of data* from memory to the device.

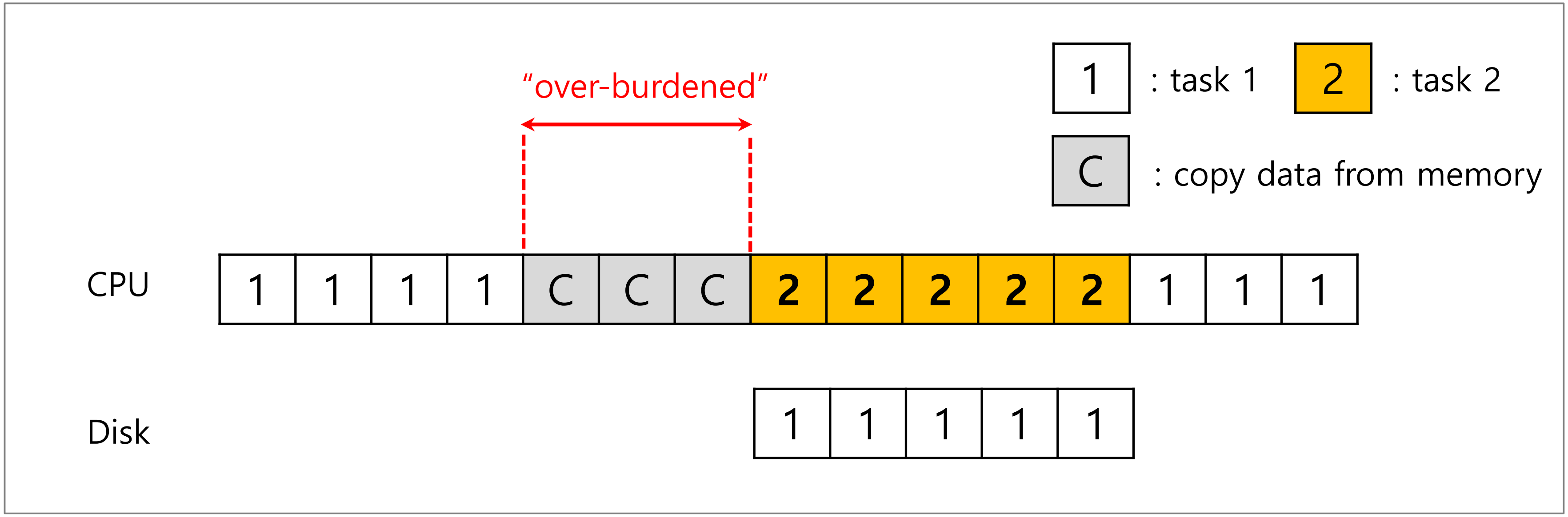


Diagram of CPU utilization

DMA (Direct Memory Access)

- **Copy data** in memory by knowing “where the data lives in memory, how much data to copy”
- When completed, DMA raises an interrupt, I/O begins on Disk.

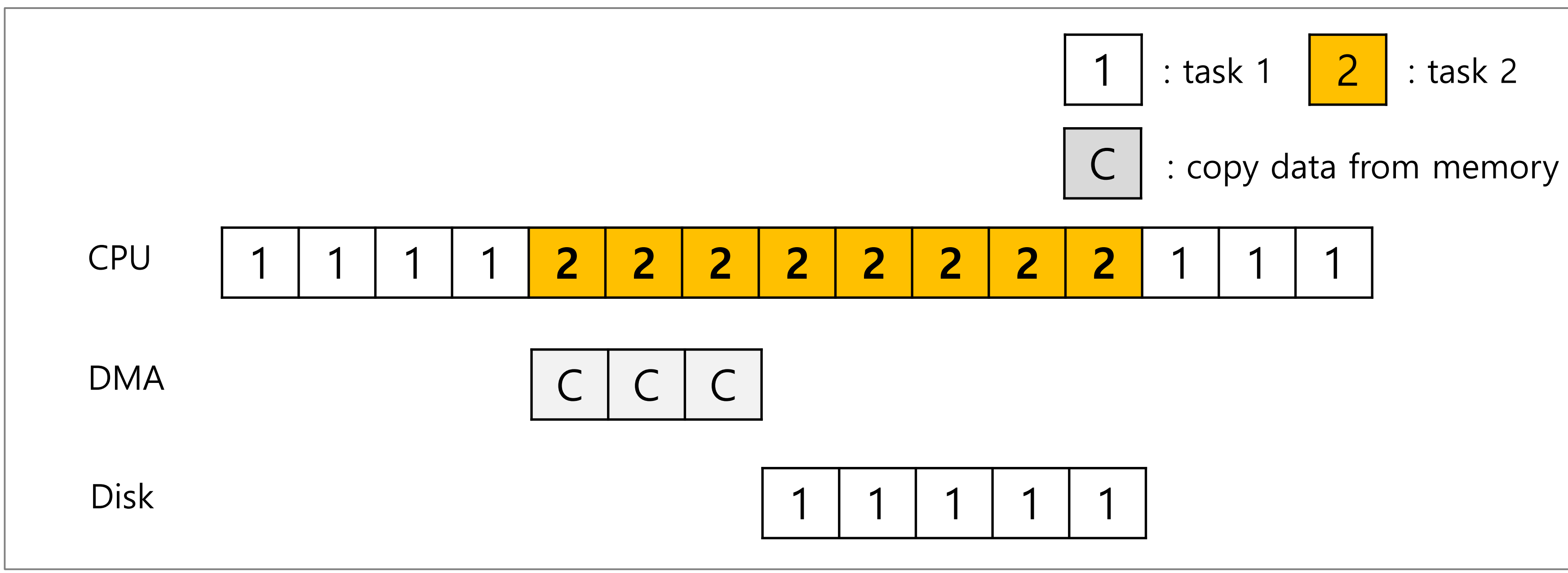


Diagram of CPU utilization by DMA

Device interaction

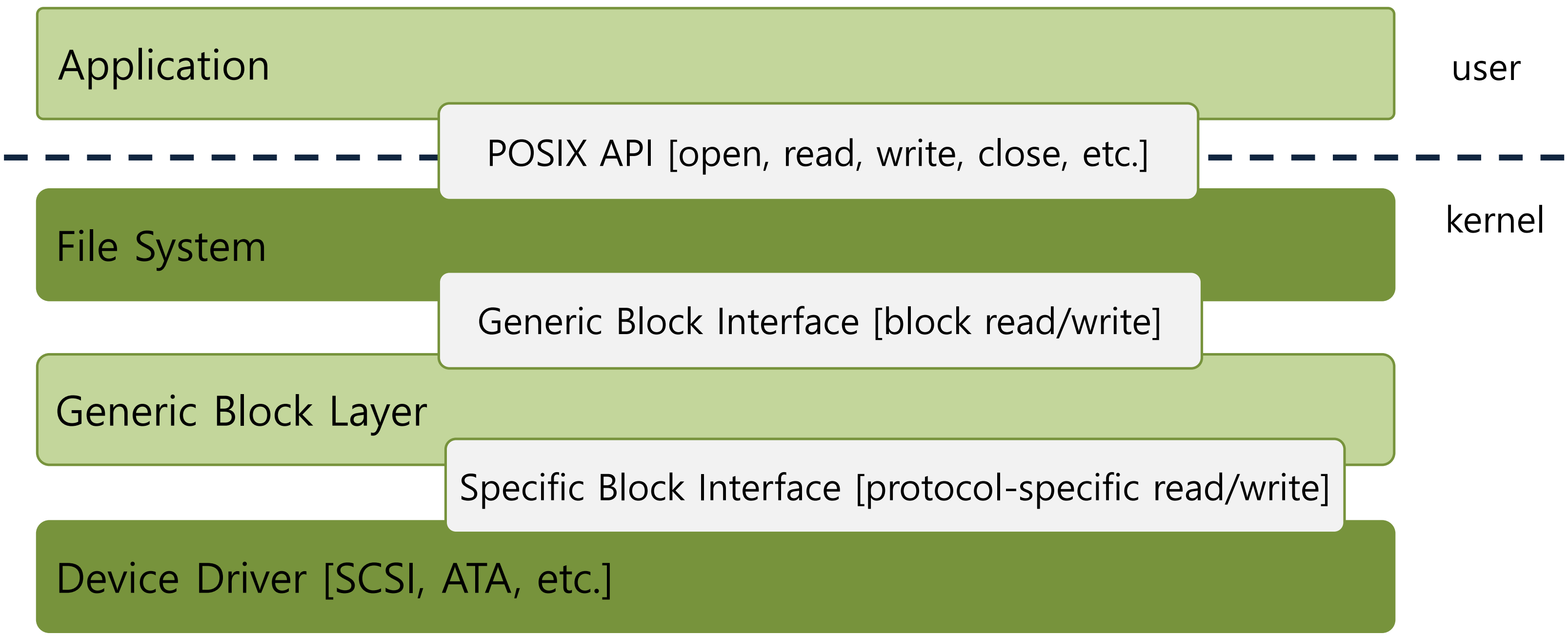
- How the OS communicates with the **device**?
- Solutions
  - I/O instructions: a way for the OS to send data to specific device registers.
    - Ex) `in` and `out` instructions on x86
  - memory-mapped I/O
    - Device registers available as if they were memory locations.
    - The OS `load` (to read) or `store` (to write) to the device instead of main memory.

Device interaction

- How the OS interact with devices **specific interfaces**?
  - Ex) We'd like to build a file system that worked on top of SCSI disks, IDE disks, USB keychain drivers, and so on.
- Solutions: Abstraction
  - Abstraction encapsulate any specifics of device interaction.

File system Abstraction

- File system specifics of which disk class it is using.
  - Ex) It issues **block read** and **write** request to the generic block layer.



The File System Stack

Problem of File System Abstraction

- If there is a device having many special capabilities, these capabilities will go unused in the generic interface layer.

- Over 70% of OS code is found in device drivers.
  - Many device drivers are needed because you might plug it to your system.
  - They are primary contributor to **kernel crashes**, making **more bugs**.

Reading Material

- **Chapter 36** of OSTEP book – by Remzi and Andrea Arpaci-Dusseau (University of Wisconsin) http://pages.cs.wisc.edu/~remzi/OSTEP/file-devices.pdf

Questions?