Operating System: I/O devices

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

Persistence

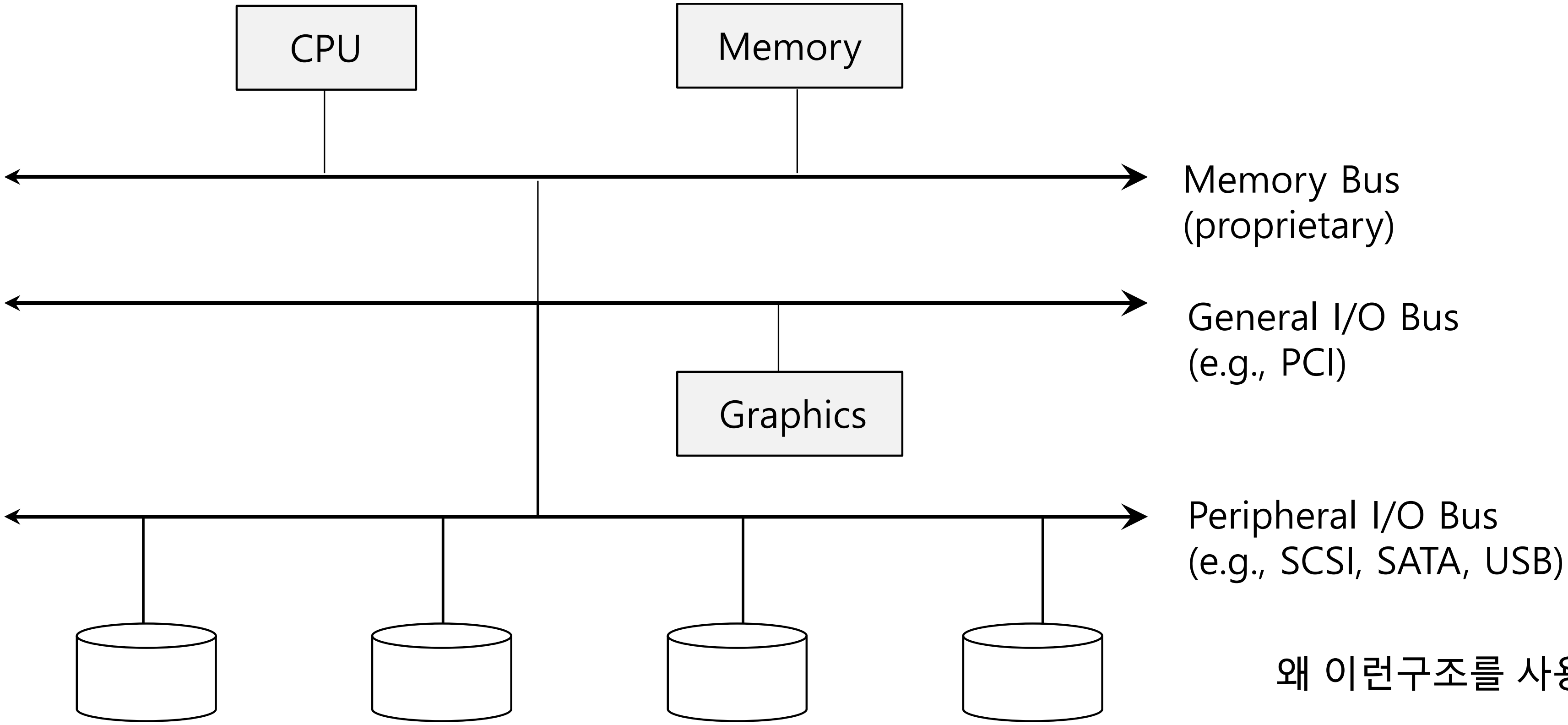
A Dialogue on Persistence

- *Professor*: Anyhow, you pick a peach; in fact, you pick many many peaches, but you want to make them last for a long time. Winter is hard and cruel in Korea, after all. What do you do?

You have to do <u>a lot more work</u> to making the peach persist

Hardware support for I/O

- A typical system structure



왜 이런구조를 사용할까?

Prototypical System Architecture

Why use hierarchical buses?

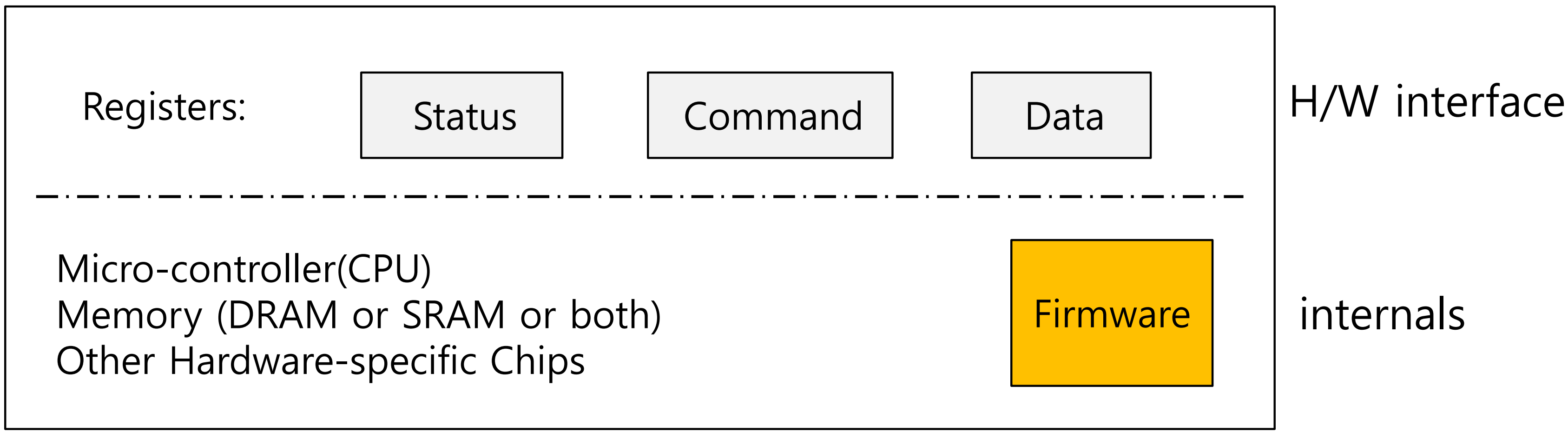
I/O architecture

- Buses 정보 교환 통로
  - Data paths that provided to enable 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 three hardware components: I/O ports, interfaces and device controllers

Canonical device

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



Canonical Device

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

Hardware interface of Canonical Device

- A typical interaction

```
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
```

A Typical I/O Device

- Status check 상태 체크
  - Polling
  - Interrupts
- Data transfer 데이터 전달 방법
  - Programmed I/O (PIO)
  - DMA direct memory access
- Control 제어 방법
  - Special instructions (e.g. in & out in x86)
  - memory-mapped I/O (e.g. load & store)

Polling

cpu를 누구에게도 할당하지 않고 입출력요청이 완료될때까지 기다리는 거

- Operating system waits until the device is ready by repeatedly reading the status register
  - It is simple and working
  - It wastes CPU time just waiting for the device
  - Switching to another ready process is better utilizing the CPU

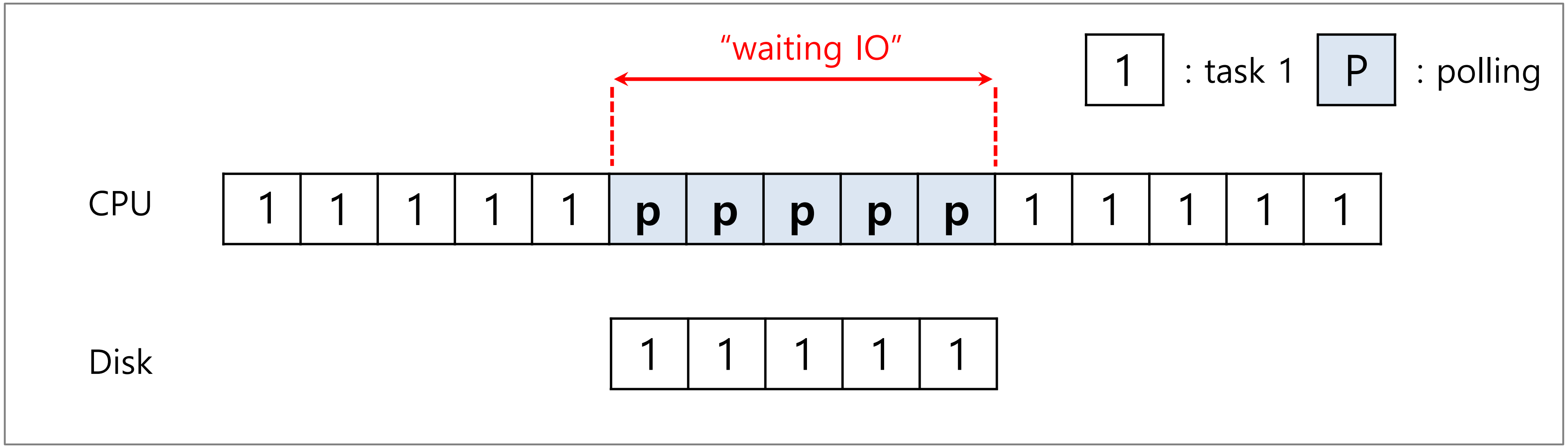


Diagram of CPU utilization by polling

Interrupt

polling과 대비되는 방법으로 입출력 요청 process는 sleep시키고 다른 process로 문맥교환하는 방식

- 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 is allow to CPU **and the disk are properly utilized**

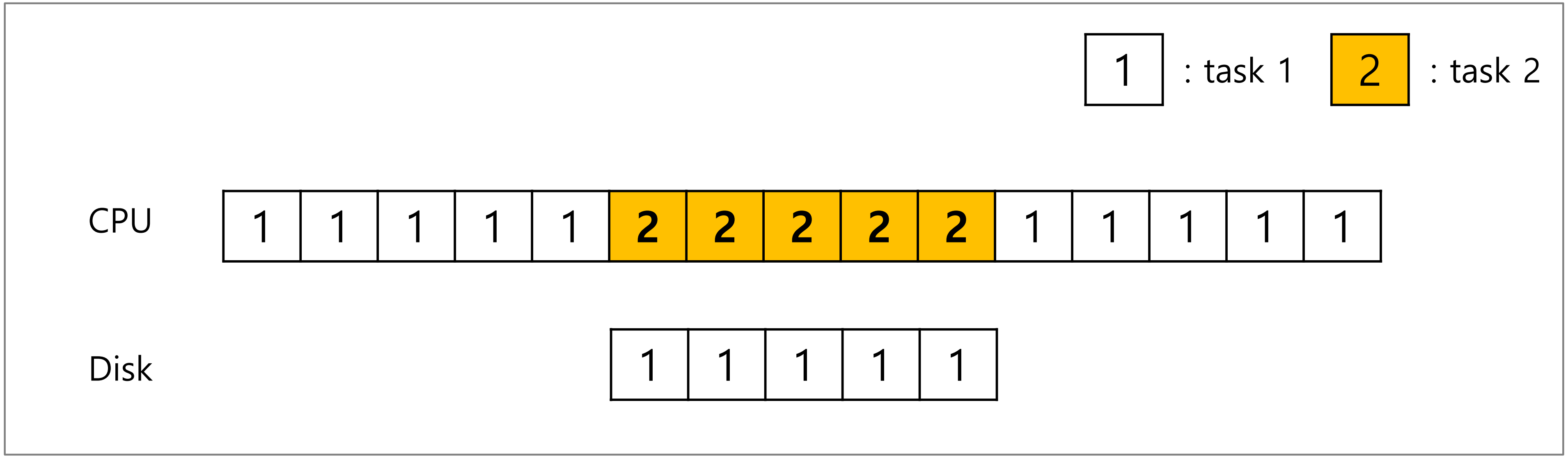


Diagram of CPU utilization by interrupt

Polling vs interrupts

- Interrupt 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 → polling is better
If a device is slow → interrupt is better

interrupt가 항상 좋은 건 아님 context switch가 오버헤드가 커서 입출력 요청이 빠르게 끝낼 수 있는 경우에는 polling이 더 효율적일 수 있다.

Programmed I/O

- CPU is once again over-burdened
  - CPU wastes a lot of time to copy *a large chunk of data* from memory to the device

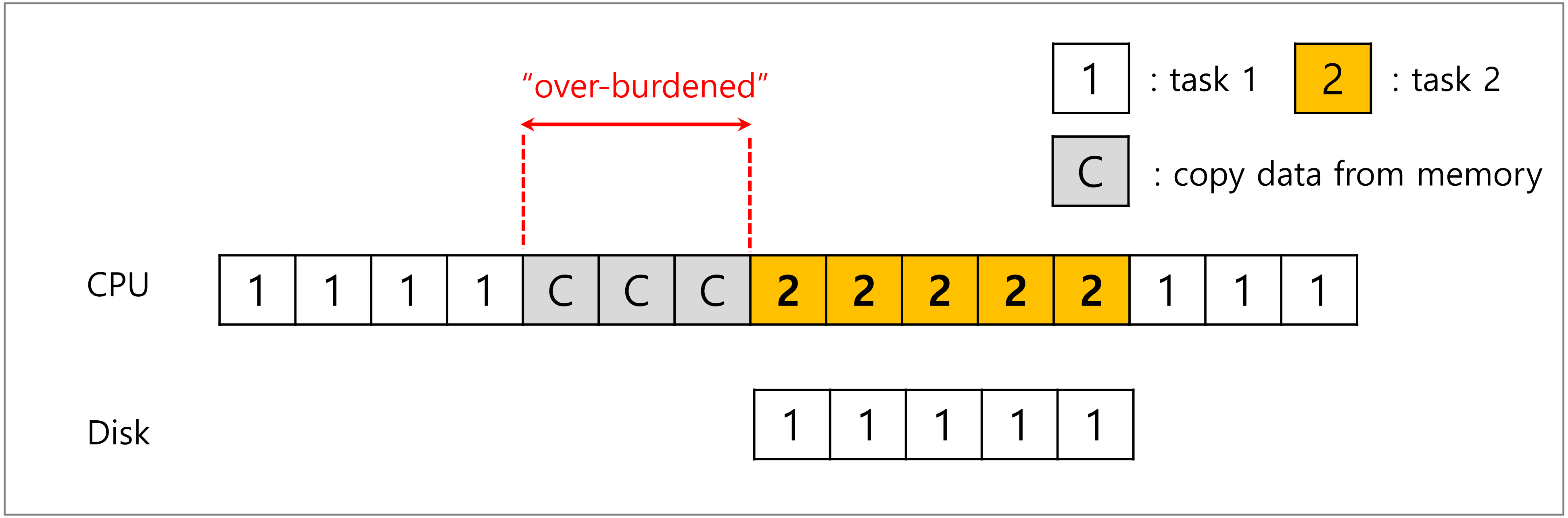


Diagram of CPU utilization

메모리에 데이터를 저장할때 CPU를 사용하는 방법 이렇게 되면 적은 데이터면 상관없지만 큰 데이터를 보내게될때 복사시 너무 많은 시간을 소비하게 된다.

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

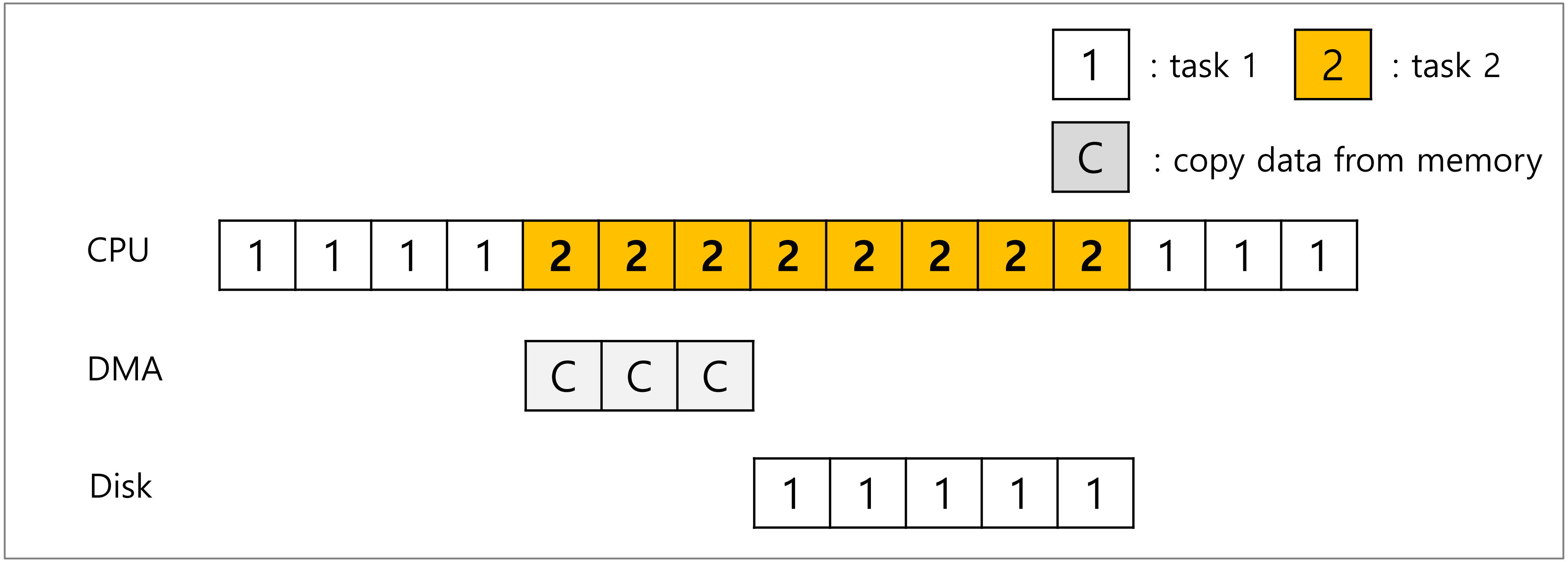


Diagram of CPU utilization by DMA

DMA라는 추가 하드웨어 장치를 이용해서 데이터 복사를 맡기고 복사가 끝나면 disk와 입출력을 한다.
DMA가 복사할 데이터가 어디있는지 얼마나 복사를 해야하는지와 같은 정보를 알고 있어야한다.

Device control

- 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 device register가 실제 메모리라고 생각하고
    - The OS `load` (to read) or `store` (to write) to the device instead of main memory

      read나 store같은 명령어 뒤에 장치를 적어서 main memory로 보내는 게 아니라 device 쪽으로 바로 보냄

How to build a device-neutral OS

- How the OS interact with different specific interfaces?
  - It is good to keep as general as possible
  - E.g. We'd like to build a file system that worked on top of SCSI disks, IDE disks, USB keychain drivers, and so on 무수히 많은 디바이스와 상호작용하기 위한 인터페이스를 모두 가지고 있는건 힘들다. 따라서 어떤 일반적인 것이 필요해
- Solutions: Abstraction
  - Abstraction encapsulate any specifics of device interaction

Device abstraction

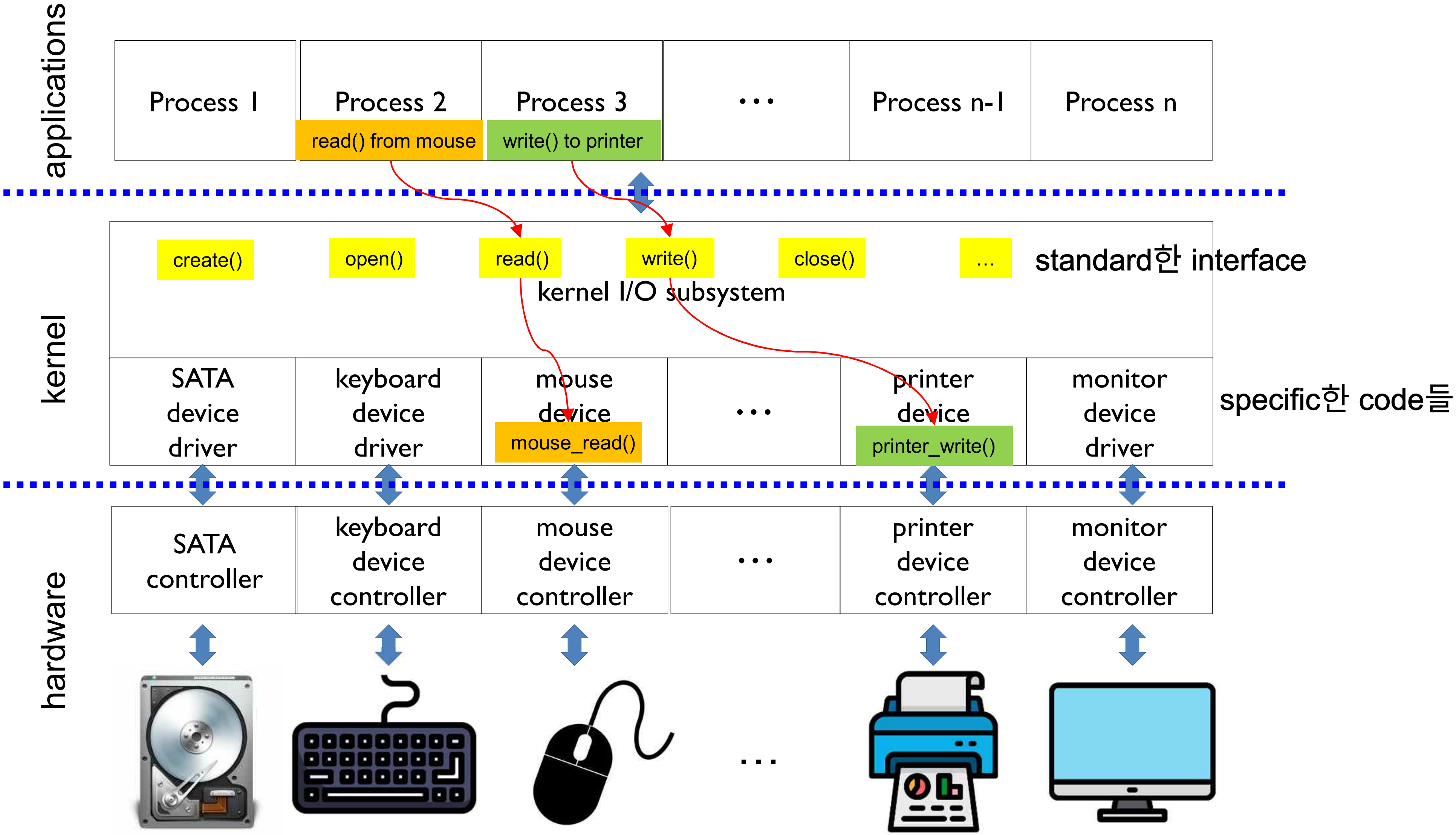
- I/O subsystem
  - A simple example for output to device

```
int fd = open("/dev/something");
for (int i = 0; i < 10; i++) {
        fprintf(fd,"Count %d\n",i);
}
close(fd);
```

  - This code works on many different devices
  - I/O subsystem provides the standard interface to different devices 아래와 같은 standard한 interface를 이용해서 다양한 device와 상호작용이 가능하다.
    - create, open, read, write, close, etc.
  - I/O subsystem provides a framework for cooperating with device drivers
- Device Driver
  - Device-specific code in the kernel that interacts directly with the device hardware

Device abstraction (Cont.)

- A kernel I/O structure



Problem of device abstraction

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

  device마다 가지는 특별한 기능들은 general한 interface에서 호환되지 않을 수 있다.
  ex) 카메라에서 데이터를 읽는 건 가능해도 줌인아웃을 하는 법은 없을 수 있는거지

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

    따라서 OS측에서 이러한 기능을 지원하기 위해서 코드의 크기가 증가하고 있음 이러면 더 많은 기능을 지원할 수 있으나 무거워진 kernel에서 crash가 발생하거나 bug가 발생할 수 있다는 거

Other I/O issues

추가적으로 제공하는 기능들

- I/O Scheduling

  I/O 요청 성능을 올리기 위한 스케줄링 기법
  - Reorders the I/O requests for I/O performance improvement
  - E.g. disk I/O scheduling - SSTF, SCAN, C-SCAN, etc.
- Buffering

  메모리에서 device로 이동시킬때 속도차이 혹은 전송 크기의 차이때문에 버퍼링이라는 것을 제공함
  - Stores data in memory while transferring between devices
    - to cope with device speed mismatch
    - to cope with device transfer size mismatch
- Caching

  더 빠른 메모리 쪽에 데이터를 복사해두어서 속도를 증가시킴
  - Holds copies of data in fast memory for I/O performance improvement
- Spooling

  주로 프린터에서 확인 가능한 기능으로 여러 인쇄요청이 들어왔을 때, 하나의 인쇄(task)가 완료될때까지 나머지 요청들이 끼어들지 않고 기다리게끔 하는 기능
  - Holds output for a device that cannot accept interleaved data streams
  - E.g. printer