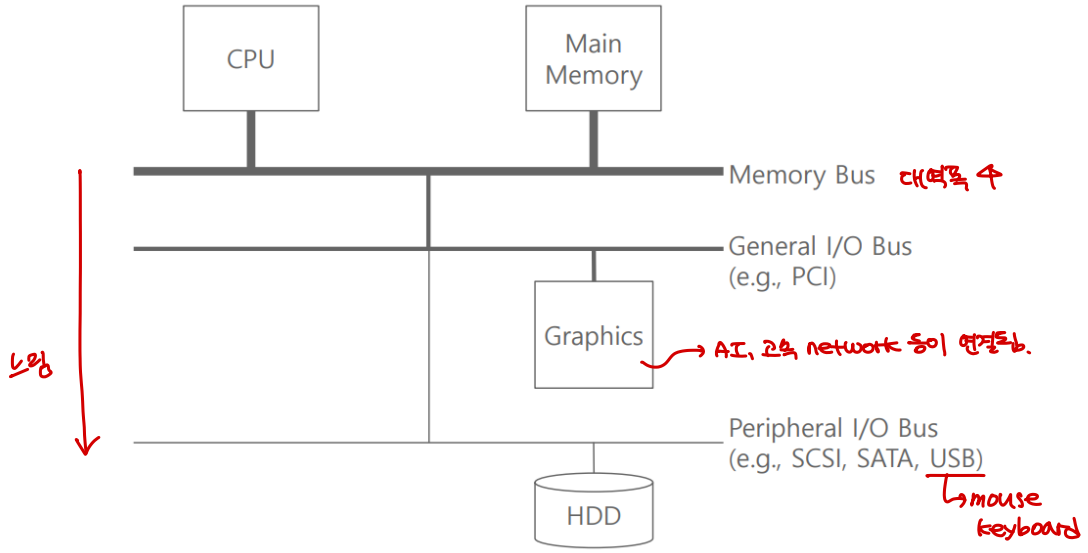
ch3
⇒ persistency

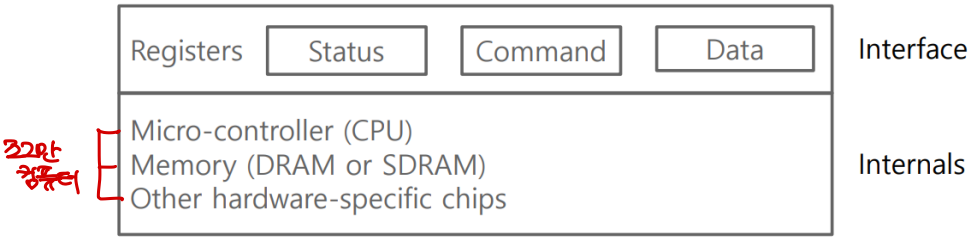
20. I/O Devices and HDD

▼ System Architecture(여기 강의 좀 다시)



▼ I/O Devices

1. interfaces
   ↳ H/W의 명령어를 어떤 식으로 넘겨줄 것인지!
   - allows the system software to control its operation
   - 모든 device → specified interface + typical interaction에 대한 각각의 protocol이 존재 ⇒ 어떤 핸더가 제작했느지에 따라 각각의 protocol이 존재해야 함.
2. internal structure
   - 기기가 system에 제시하는 implementation을 구체적으로 구현



▼ protocol

- device와 device driver 사이의 communication 어떻게?

```
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
```

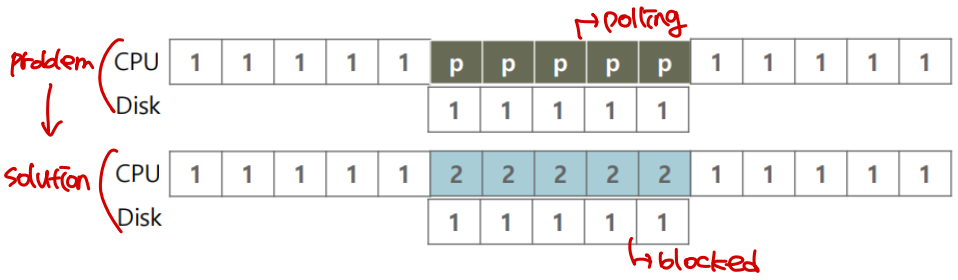
but, 상당히 효율적이지 않은 구현 ver

- inefficiencies and inconveniences
  1. cpu 자원 낭비
     - solution : interrupt
       - interrupts : cpu overhead 낮춰 줌
         1. device를 반복적으로 낭비하는 것 대신에 OS가 요청을 받을 수 있음
         2. 호출한 process가 blocked state로 이동

            ⇒ busy waiting 하는데 사용 x (CPU 더 효율적으로 사용)
         3. 다른 task로 context switch

         - device : operation을 완전히 끝낸 후 h/w interrupt를 발생시킴
           - h/w interrupt : 미리 정의해둔 interrupt service routine(ISR, interrupt handler)로 CPU가 이동함.

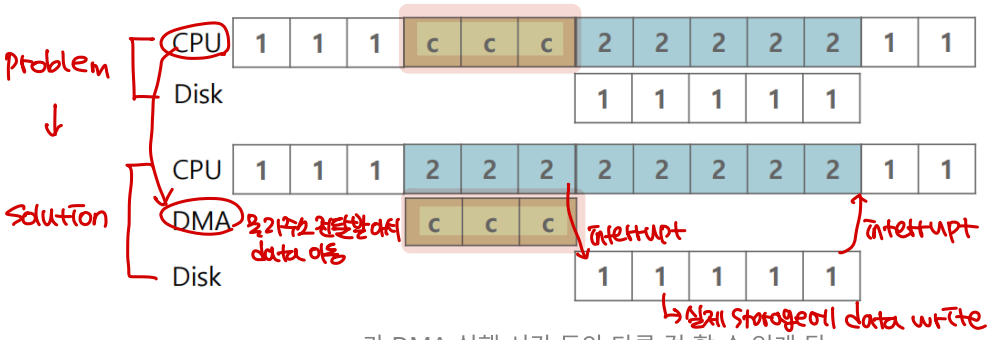
         - interrupts : computation, I/O가 overlap 가능



  2. programmed I/O(PIO) : device 쪽에 data 옮기기 (BUSY가 아닐 때까지 spining)
     - CPU → data movement 하는 동안 cpu 낭비

- solution : Direct Memory Access(DMA)
  - Direct Memory Access(DMA) : 더 효율적인 data 이동
    - DMA engine : CPU 개입 없이 device와 main memory 사이의 data 전송을 조정할 수 있는 장치
    - OS : data가 memory 어디에 있고 copy하는데 얼마나 걸리는지 DMA engine에게 말해줌으로써 program

    → DMA가 완료되면 DMA controller가 interrupt 발생시킴



cpu가 DMA 실행 시간 동안 다른 걸 할 수 있게 됨

  - Methods of Device Interaction

    1. I/O instructions

       ex) in and out (x86) ⇒ usually privileged

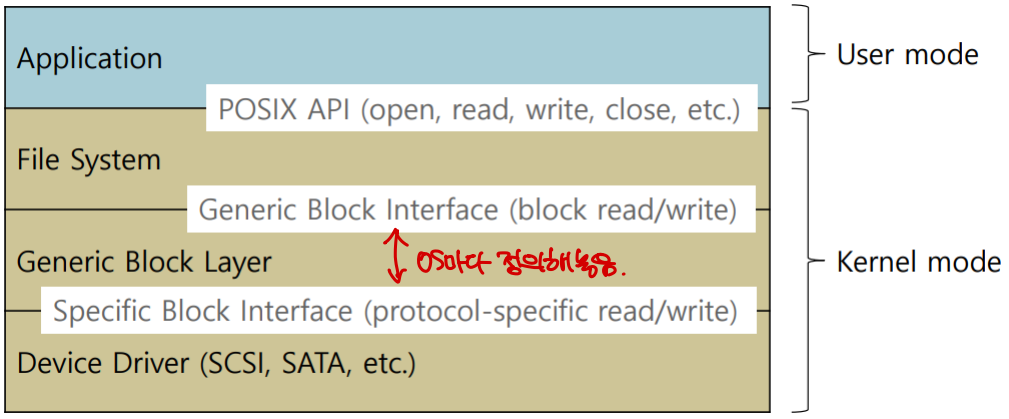
    2. memory-mapped I/O

       h/w : memory location처럼 device register 사용

       → 특정 register에 접근할 때 OS가 그 주소에 load/store

    ⇒ 어느 하나 유독 잘난 건 없음. 때에 따라 사용

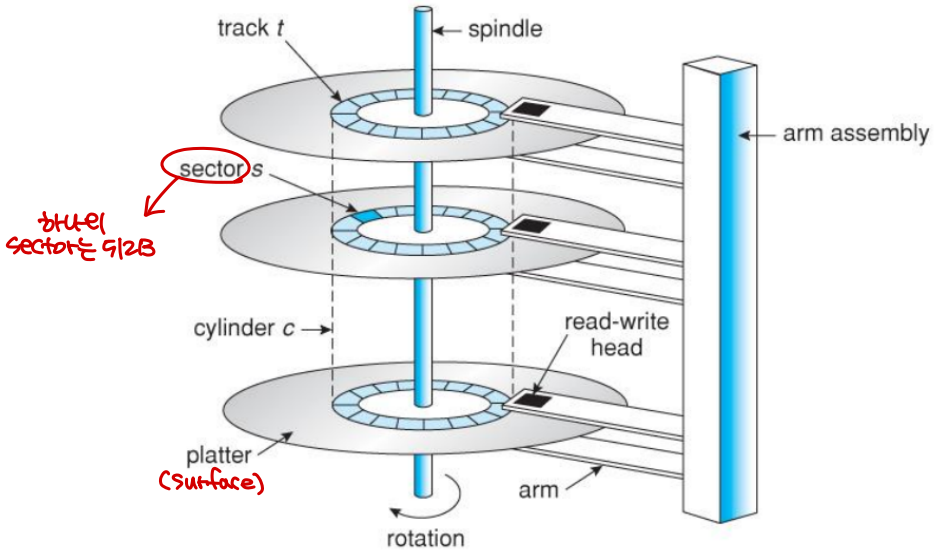
▼ Device Drivers



▼ Hard Disk Drivers(HDD)

▼ HDD

- Basic geomerty → 물리적인 모양이 어떻게 생겼는지 확인해보자!



1. Platter : data가 저장되어 있는 판때기

   → 하나의 disk는 여러 or 하나의 platter(surface)가 존재하며 양면에 저장 가능

2. Spindle : platter들이 연결되어 있는 중심축

   → platter를 일정한 속도로 회전시키는 motor에 연결

   - rotation 속도는 분당 속도로 측정됨 (RPM)(ex: 7,200~15,000 RPM) ↳ 높을수록 좋음

3. Track : data는 sector의 동심원들 중 각 surface에 encoding됨

4. Disk head and Disk arm : reading, writing은 disk head에 의해 표시됨

   - drive의 platter(surface) 마다 하나의 head가 존재 → 양면 각각 존재
   - disk head : 각 disk arm마다 붙어 있으며 track에 따라 head의 위치가 platter 가로질러 움직임

- I/O Time ⇒ 특정 sector 접근 시에 걸리는 시간

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

↳ sector가 head를 지나갈때의 data 전송 속도

  - seek time : 정확한 track으로 disk arm(head)가 이동하는 시간
  - rotational delay : disk head 밑에 원하는 sector가 회전할 때까지 기다리는 시간 ↳ track이 도는데 걸리는 시간

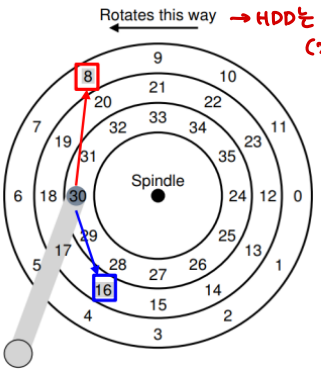
▼ Disk Scheduling

- OS: disk에 접근하는 I/O의 순서를 결정
  - I/O request가 주어지면 disk scheduler는 요청을 검사 + 다음에 schedule될지 결정함.

- SSTF : shortest seek time first ⇒ seek time이 젤 적은 놈부터 read
  - track에 의해 I/O 요청을 queue로 정렬 → seak time이 가장 먼저 끝나는 놈부터 pick
  - problem
    - ①drive geometry → host OS에 not availabe, NBF(nearest-block-first) block 배열이 나타남
    - ②starvation : 멀리에 위치하는 놈들은 자꾸 우선순위에서 밀림
- Example(OS : track에 대한 구조는 모른다고 가정)
  - 98, 183, 37, 122, 14, 124, 65, 67 (Head starts at 53)
  - ~~SSTF~~ : 65 → 67 → 37 → 14 → 98 → 122 → 124 → 183
  - FCFS (FIFO) : 98 → 183 → 37 → 122 → 14 → 124 → 65 → 67
- Elevator
  - SCAN
    - disk service 요청 사이에 track 간에 순서대로 앞뒤로 움직임
  - C-SCAN
    - 밖에서 안으로만 sweep 한 뒤 바깥 track부터 다시 시작하도록reset

problem
⇒ seek만 고려하기에 rotation cost가 비쌈

- Example(OS : track에 대한 구조는 모른다고 가정)
  - 98, 183, 37, 122, 14, 124, 65, 67 (Head starts at 53)
  - SCAN : 37 → 14 → 65 → 67 → 98 → 122 → 124 → 183
  - CSCAN : 65 → 67 → 98 → 122 → 124 → 183 → 14 → 37
- SPTF(Shortest Positioning Time First)



  - seek과 rotation을 동일하게 고려
    - rotation > seek : 80이 더 가까움 → track이 2개만 회전하면 됨

    - rotation < seek : 가까움 ⇒ SSTF 적용
- OS에 구현해놓기 훨씬 더 어려움
    - 현대의 system에서는 disk가 여러 개의 미해결 요청을 해결할 수 있고 정교한 내부 scheduler 자체를 가지고 있음.