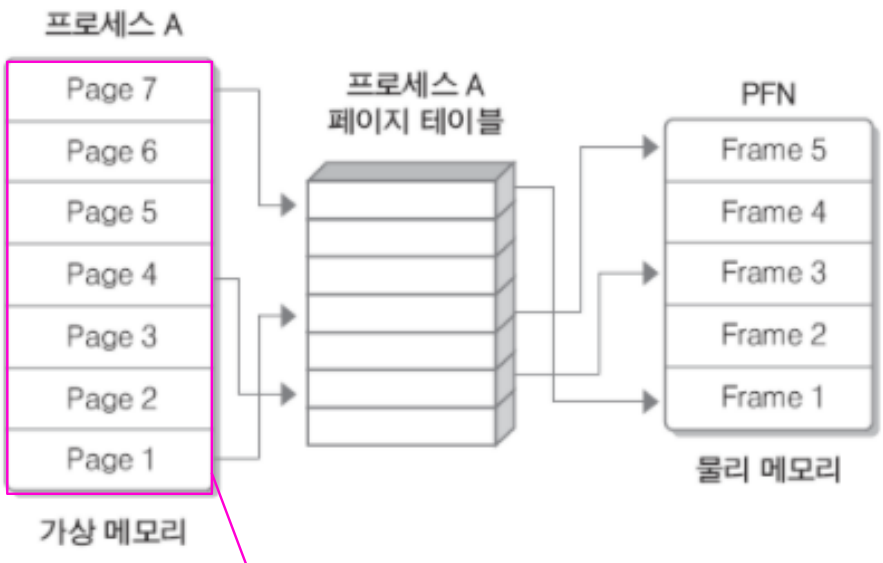
운영체제 21장

- 가상 메모리 -

메인 메모리의 크기가 한정되어 있으므로 물리적인 메모리 크기보다 크기가 큰 프로세스를 실행시킬 수 없다. 예로 100MB 메인 메모리에서 200MB 크기의 프로세스를 실행할 수 없게 되는 것이다. 그렇다면 메인 메모리보다 크기가 큰 프로세스를 실행시키고 싶으면 어떻게 해야 할까? 단순히 메인 메모리가 더 큰 컴퓨터를 사용해야하는가? 이런 방법은 매우 비효율적일 것이다. 그래서 나온 방법이 바로 가상 메모리이다.

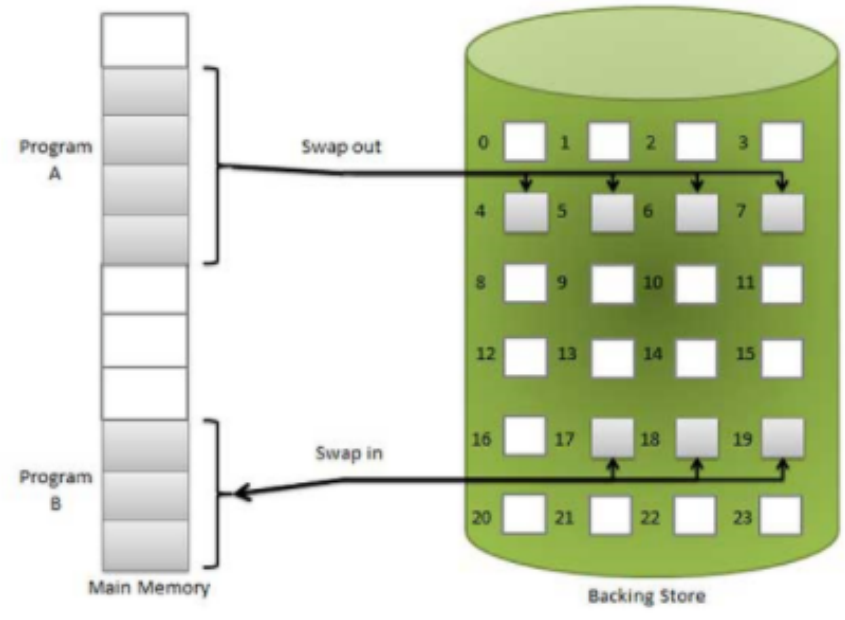
프로세스의 모든 코드는 항상 필요한 것이 아니다. 오류 처리하는 부분이나 필요 없는 배열 부분은 실제로 프로세스가 잘 동작한다면 필요 없는 부분이 된다. 따라서 프로세스는 필요한 부분만 메모리에 올림으로써 메인 메모리에 올라가는 프로세스의 크기를 줄인다. 프로세스 이미지를 모두 메모리에 올릴 필요가 없는 것이다. 동적 적재와 비슷한 개념이라고 알 수 있다.



① 우선 우리가 실행을 시키고자 하는 프로세스들을 페이징 과정을 먼저 실행한다. 메인 메모리의 외부 단편화 문제를 해결하여 메모리 낭비를 줄이는 데 페이징을 사용하면 매우 효율적이기 때문이다. 따라서 페이징 과정을 거쳐 특정 단위인 페이지 단위로 프로세스를 자른다. 그러면 여기서 페이지들마다 필요한 부분과 필요 없는 부분으로 나눌 수 있다. ② 여기서 필요한 페이지만 메모리에 적재를 하면 많은 메모리의 낭비를 막고 우리가 필요한 프로세스들을 모두 메인 메모리에서 실행을 시킬 수 있게 되는 것이다. 이를 요구 페이징이라고 한다.

요구 페이징(Demand Paging)은 프로세스의 이미지를 backing store에 저장한다. backing store는 swap device로 하드웨어의 부분인데 페이지를 임시로 보관하는 공간이다. 프로세스는 페이지의 조합이기 때문에 필요한 페이지만 메모리에 올린다. 이를 요구되는 페이지만 메모리에 올린다는 의미로 요구 페이징이라고 부른다.

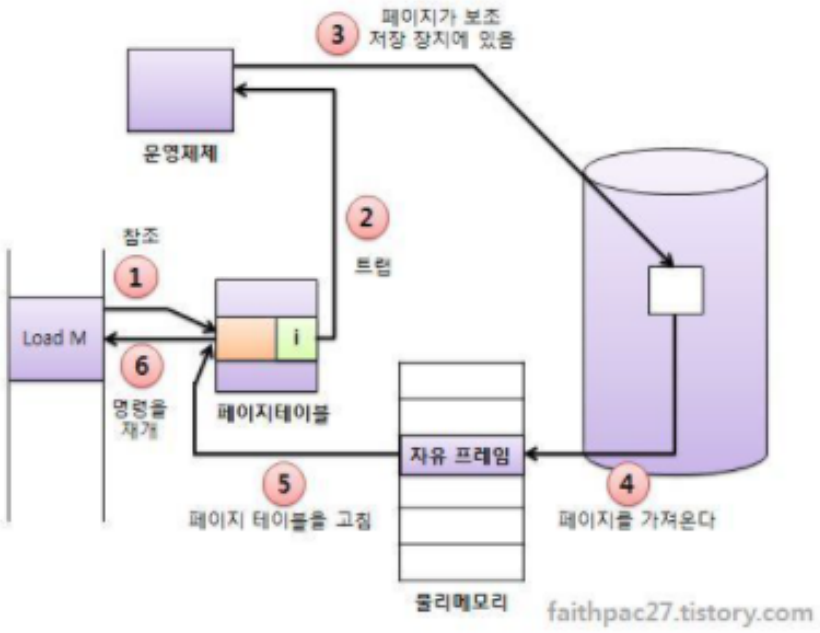
하드디스크에 존재

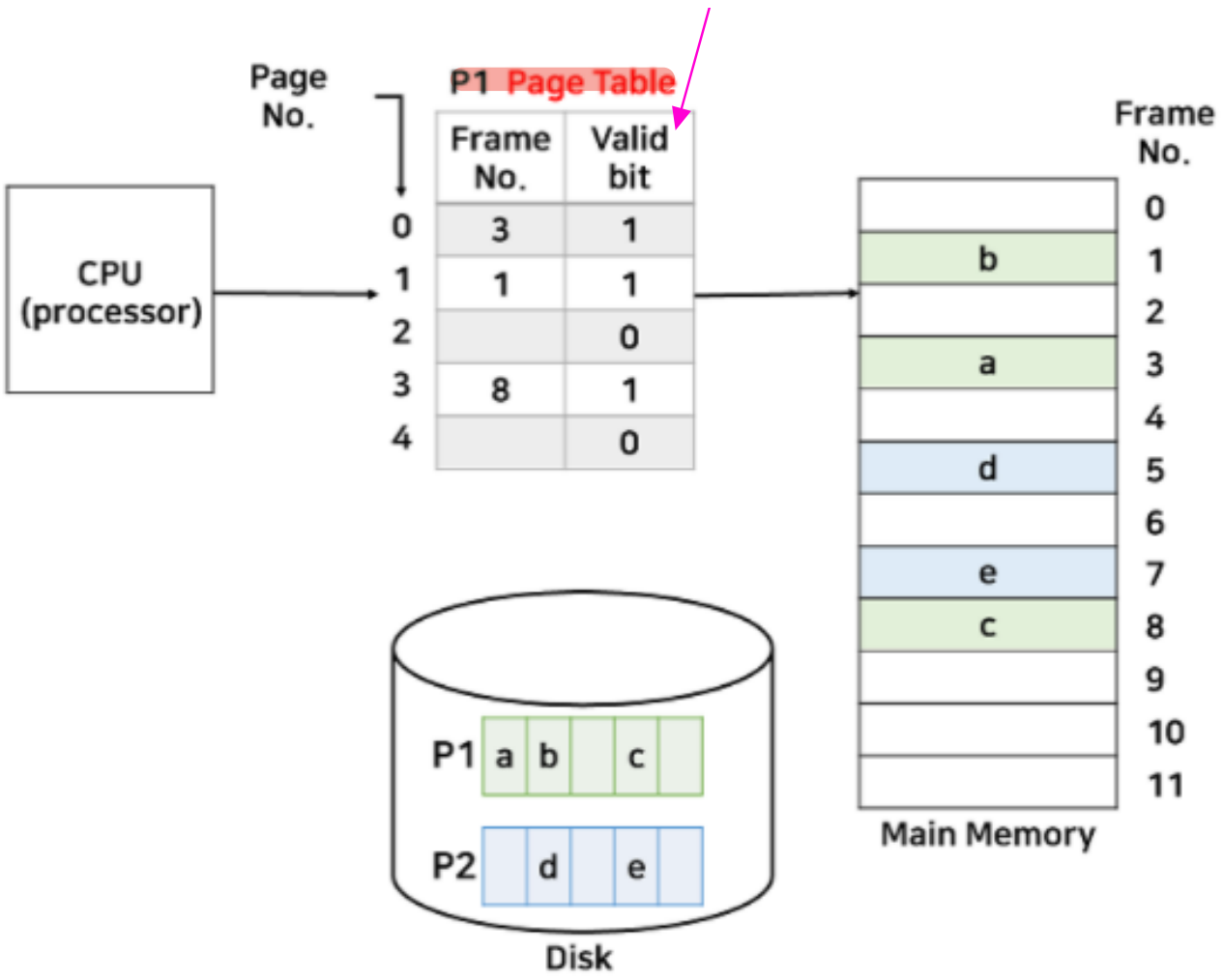


Memory Management Unit : CPU가 메모리에 접근하는 것을 관리하는 컴퓨터 하드웨어 부품.

페이징 기법을 사용할 때 페이지 테이블이라는 부분을 놔두게 된다. MMU의 재배치 레지스터를 통해 논리 주소를 물리 주소로 바꾸어주는 주소 변환 과정을 거쳐 CPU가 프로세스는 연속적으로 할당되어져 있다고 속게 만드는 작업을 한다. 그런데 요구 페이징 기법을 사용하면 페이지가 메모리에 올라와있는 것도 있고 올라가지 않고 backing store에 보관되어 있는 것도 존재한다. 따라서 페이지 테이블을 작성할 때 이를 구분해줄 도구가 필요하다. 그래서 valid 비트 필드를 페이지 테이블에 추가한다. 1과 0의 값으로 메모리에 적재되어 있는지 없는지를 구분할 수 있다.

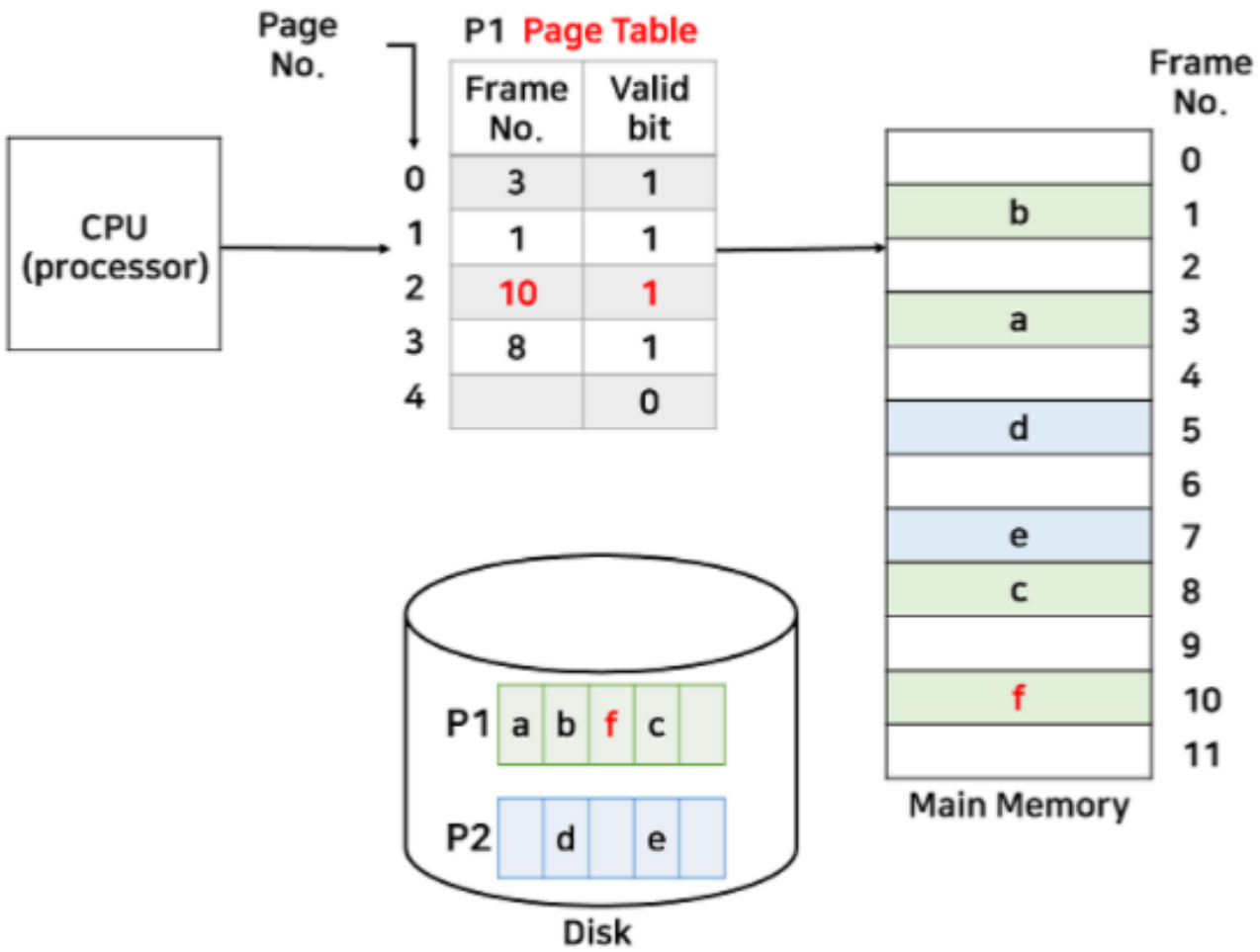
만약 오류가 발생한다면 오류를 제어할 수 있는 코드를 실행해야한다. CPU에서 해당 메모리를 가져오라고 논리 주소를 보냈는데 페이지 테이블에서 접근하려는 페이지가 메모리에 없다고 표시가 되어 있다. 이는 valid 비트 필드에 의해서 결정된다. 그러면 Backing store에서 해당 페이지를 가져와야한다. 이를 수행하기 위해서 CPU는 잠시 하는 일을 멈추고 운영체제가 나서서 Backing store를 뒤져 필요한 페이지를 메모리에 적재하게 된다. 그리고 valid 비트를 올라와 있다고 바꾸어준다. 이런 현상을 페이지 결함, 페이지 부재(Page Fault)라고 부른다.





위 그림은 요구 페이징의 모습이다. 두 프로세스 P1, P2는 각각 필요한 페이지만 메모리에 할당하였다. 여기서 위 그림의 테이블은 P1이 수행 중일 때의 페이지 테이블이다. 기존의 페이지 테이블과 다른 점은 **valid bit** 가 추가되었다. 이는 현재 메모리에 페이지가 있는지 없는지를 나타내는 비트이다. 현재 페이지가 메모리에 있다면 1, 없다면 0값을 갖는다.

만약, CPU에서 P1의 ~~3~~ 2번째 페이지에 접근하는데, valid bit값이 0이다. 그러면 **CPU에 인터럽트 신호를 발생하여 운영체제 내부의 ISR로 점프**한다. 여기서 디스크 내부의 프로세스 P1에 있는 2번째 페이지를 메모리에 할당하는 작업을 처리한다.



위 그림은 P1의 3번째 페이지를 메모리에 올린 후 모습이다.

가상 메모리를 만드는 방법은 대표적으로 두 가지가 존재하지만, 대부분 **요구 페이징을 사용**하므로 가상 메모리와 요구 페이징을 같은 용어로 사용하는 경우가 많다.

요구 페이징을 할 때 두 가지의 종류가 있다. 처음부터 모든 페이지를 적재시키지 않고 CPU가 요구할 때 valid를 바꾸어 페이지를 적재하는 방법과 우선 필요할 것 같은 페이지를 적재시키고 필요할 때 다른 페이지를 적재시키는 방법이 있다. 전자는 pure demand paging이고 후자는 prepaging 기법이다. pure 기법을 사용하면 페이지를 요구할 때만 메모리에 적재하므로 메모리의 낭비는 매우 줄일 수 있다. 하지만 요구에 의해 앞선 페이지 부재의 현상을 처리하려고 하면 많은 부담이 발생한다. 이에 반해 미리 올라와져 있는 prepaging은 처리하는 속도는 빠르겠지만 메모리가 낭비될 수 있도 있다.