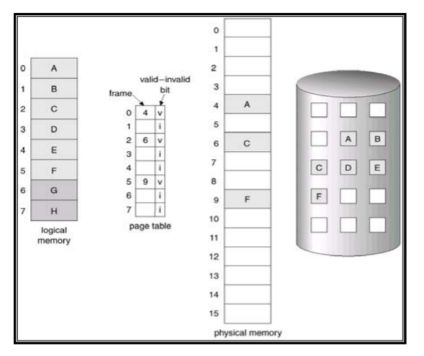
**OS 기말고사 정리**

Chapter 9. Virtual Memory Management

**Demand Paging** 

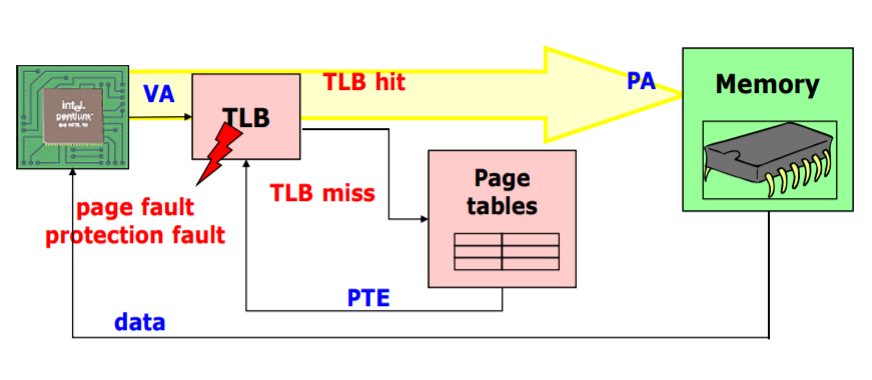
가상 메모리에 실행 과정 중에 필요한 페이지를 적재한다. (swapping 개념)

페이지가 메모리에 page in되어있는지 구별 여부는 valid-invalid bit를 사용한다.

**Page fault**: invalid(0)인 page(메모리에 없고 디스크에 존재) 참조 시 **Trap** 발생.

Free frame이 있는 경우 page in.

Free frame이 없는 경우 page replace.



TLB 먼저 검사, TLB miss일 경우 Page table 참조, Page fault 발생 시 Trap

Page fault는 낮게 유지해야 한다. 왜? Thrashing 발생 가능!

Page fault를 줄이는 방법 1. 메모리 추가 구입 (한계 존재)

2. 동작하는 프로그램을 줄이기.

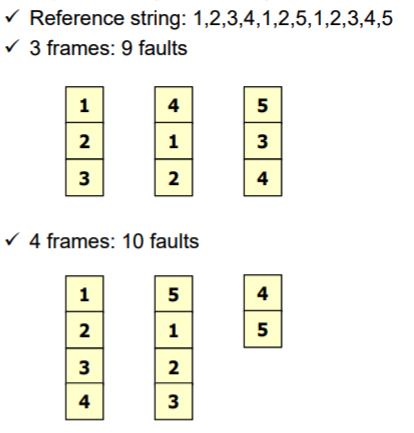
Locality: 프로그램 수행 시간의 일정 부분이 프로그램의 어떤 부분을 수행하는 데 걸리는 것

Temporal locality: 내가 참조된 후에 얼마 지나지 않아 또 참조될 확률

Spatial locality: 내가 참조됐으면 나와 근접한 것이 참조될 확률

**FIFO**

들어온 순서대로 페이지 replace!

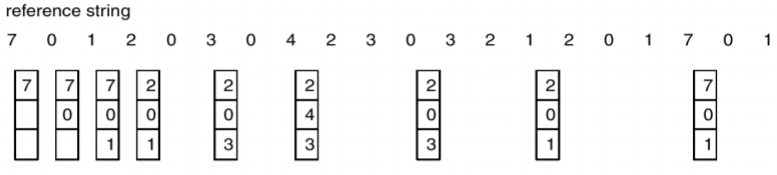


\* 프레임이 늘어났는데도 page fault가 늘어남!

* 잘못된 알고리즘!

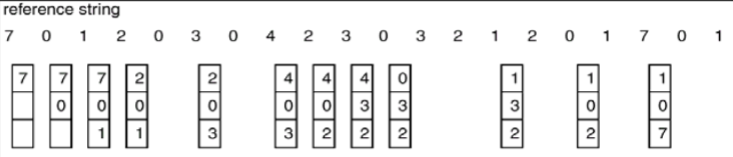
**Optimal Algorithm**

오랫동안 쓰이지 않을 페이지 replace! (미래를 알고 있다는 가정)



**LRU (Least Recently Used)**

오랫동안 쓰이지 않은 페이지 replace! (과거를 이용해 추측)



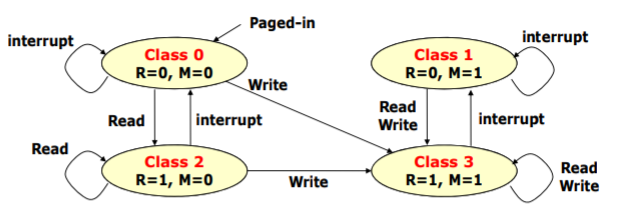
장점: 거의 Optimal에 가까운 성능을 보임.

단점: 페이지 별로 참조된 시간을 기록해야 하기 때문에 오버헤드가 크다.

\* approximation을 해야 한다. → 여러 비트 말고 1비트(reference bit, 참조 시 1)만 사용하자.

\* second chance: 1이면 0으로 바꾸고 다음 것을 확인 (1번 더 기회 부여)

**NRU (Not Recently Used)**

R(reference)과 M(modify)을 가지고 최근에 변경되지 않았고, 참조되지 않은 페이지 replace! 

\* interrupt: 주기적으로 R을 0으로 초기화.

\* class 0이 replace하기 좋다. (0, 1, 2, 3순으로 replace!)

**Counting Algorithm**

자주 사용되는 것은 앞으로도 자주 사용하기 때문에 count로 비교 후 replace!

**LFU (Least Frequently Used):** 가장 참조 횟수가 작은 페이지 교체.

**MFU** **(Most Frequently Used):** 가장 참조 횟수가 많은 페이지 교체.

\* 그러나 여러 비트가 필요하고, counter가 필요하기 때문에 오버헤드가 크다.

**Allocation of Frames**

**Fixed Algorithm**

1. Equal Allocation: 모든 프로세스에게 똑같이 프레임을 할당하는 방법.

남은 프레임은 가용 프레임 버퍼 풀로 사용.

2. Proportional Allocation: 각 프로세스의 크기에 맞추어 할당하는 방법

**Priority Algorithm: 프로세스에게 우선순위를 부여해서 할당 및 교체**

1. Global Allocation: 프로세스가 교체할 프레임을 다른 프로세스의 프레임에서 가져올 수 있다.

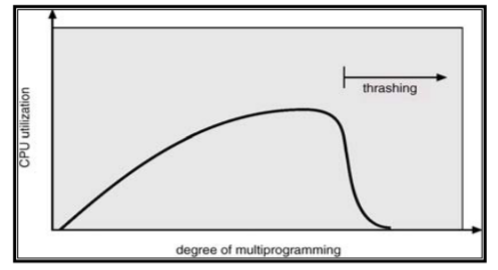
2. Local Allocation: 자신에게 할당 된 프레임에서 교체한다.

**Thrashing**

충분한 프레임을 할당 받지 못했음을 가정.

모든 page가 활발히 참조되므로 page fault가 지속적으로 일어난다.

* 실행 시간보다 페이징에 시간을 더 많이 사용하는 과도한 페이징 작업을 말한다.

OS는 이용률을 올리기 위하여 새로운 프로세스를 증가시키지만 physical memory는 한정되어 있으므로 할당된 프레임이 부족해지므로 thrashing 발생!

**해결 방안:** 프로세스의 개수를 줄인다.

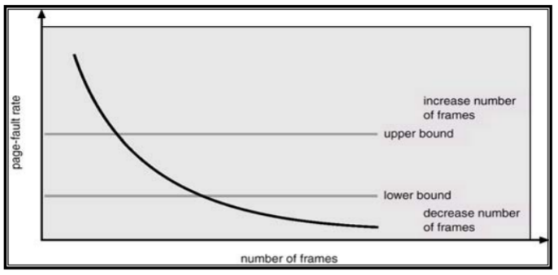
메모리를 추가한다.

**Working-Set Model**

Working-Set: 자주 참조되는 페이지들.

자주 참조되는 페이지의 개수에 따라 프레임을 할당.

**Page-Fault Frequency Scheme**

전체 프로세스의 page fault가 lower bound와 upper bound 사이로 유지시키는 방법.

Prepaging: page fault가 발생했을 때, 또 발생할 확률이 높기 때문에 간 김에 옆에 것까지 읽어옴.

Page size

클수록 fragmentation 증가 (작을수록 좋다.)

클수록 table size 증가 (클수록 좋다.)

클수록 I/O overhead 감소 (클수록 좋다.)

클수록 locality 증가 (클수록 좋다.)