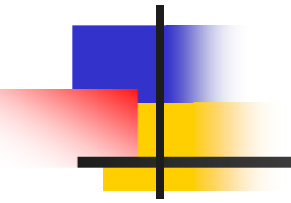
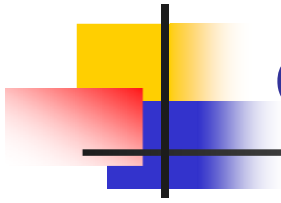


Operating Systems
(Main Memory)

Chapter 9

These lecture materials are modified from the lecture notes written by A. Silberschatz, P. Galvin and G. Gagne.

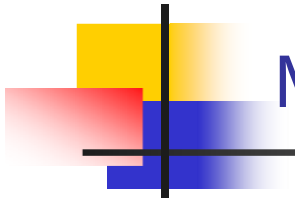
August, 2022



Outline

- Background
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table

Background



Memory

- 거대한 바이트의 배열
- 각 바이트에는 주소 (address)가 저장됨
- CPU는 프로그램 카운터가 지정한 주소에서 명령을 가져와 실행함
- 명령어에서 load나 store 같이 메모리에 접근하는 연산을 수행할 수도 있음

Memory

- 각 프로세스는 자신만의 메모리 공간을 사용해야함
- 이를 위해 베이스 레지스터 (base register), 한계 레지스터 (limit register)가 필요함.

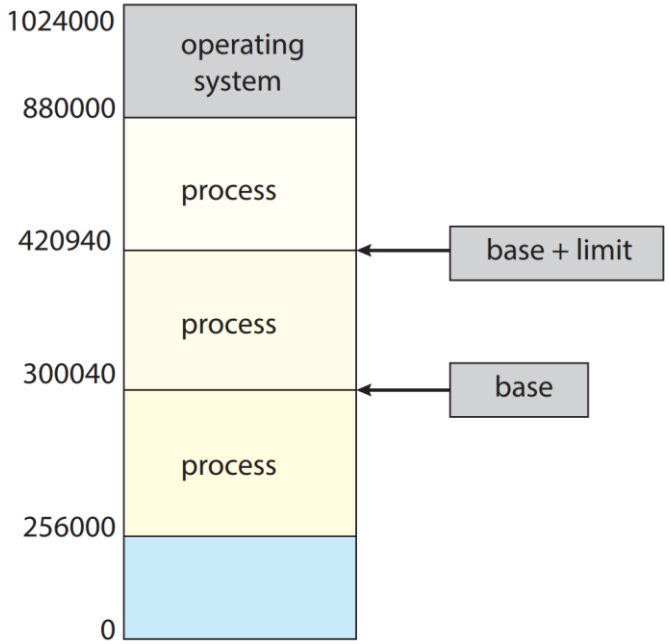


Figure 9.1 A base and a limit register define a logical address space.

Memory

- 메모리 공간의 보호는 CPU 하드웨어가 사용자 모드에서 생성된 모든 주소를 레지스터와 비교하도록 함으로써 달성.
  - 메모리 주소에 접근하고자 할 때 그 주소가 base와 limit 사이에 존재하는 주소 접근일 때에만 접근이 가능함.
  - 잘못된 접근인 경우 OS에 대한 trap이 발생하고, 이는 치명적인 오류로 감지됨.

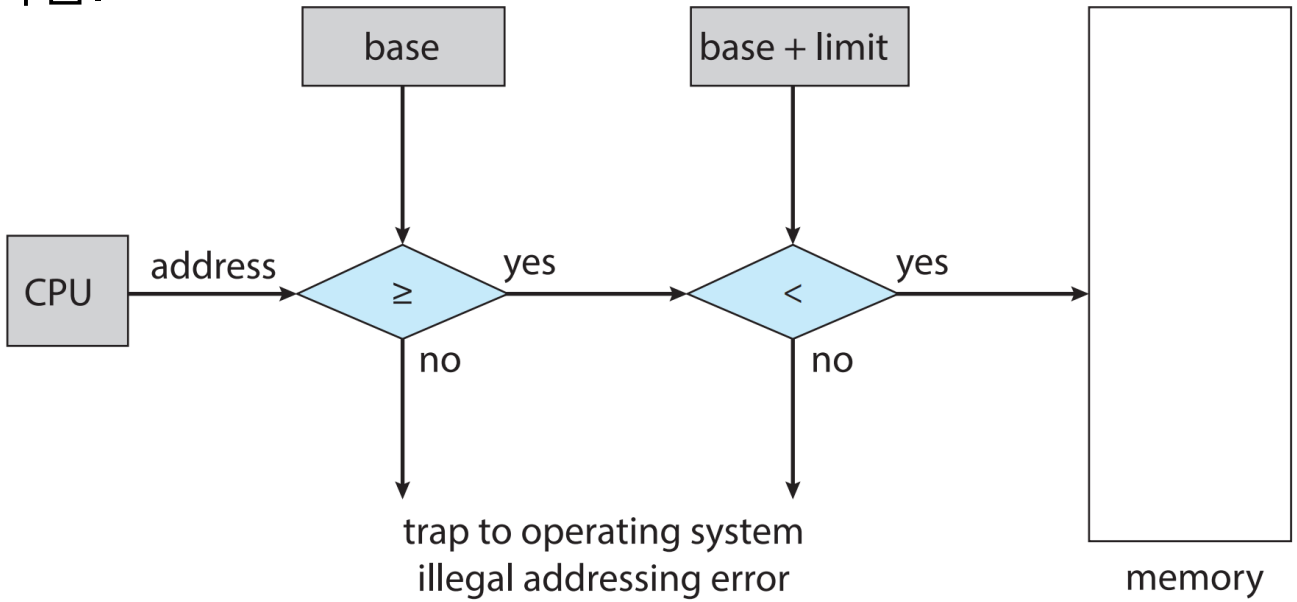
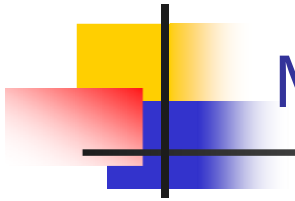


Figure 9.2 Hardware address protection with base and limit registers.



Memory

- 멀티프로세싱 시스템은 context switching을 실행해야 함.
- 메인 메모리에서 레지스터로 다음 프로세스의 context를 로드하기 전에, 레지스터에서 현재 프로세스의 state를 메인 메모리에 저장해야 함.

Address Binding

- Address Binding은 어떤 프로그램이 메모리의 어느 위치에, 즉 어떤 물리적 주소에 load 될지를 결정하는 과정

- 일반적으로 프로그램은 바이너리 실행 파일로 디스크에 상주함.
- 대부분의 시스템은 사용자 프로세스가 물리적 메모리의 어느 부분에나 상주하도록 허용

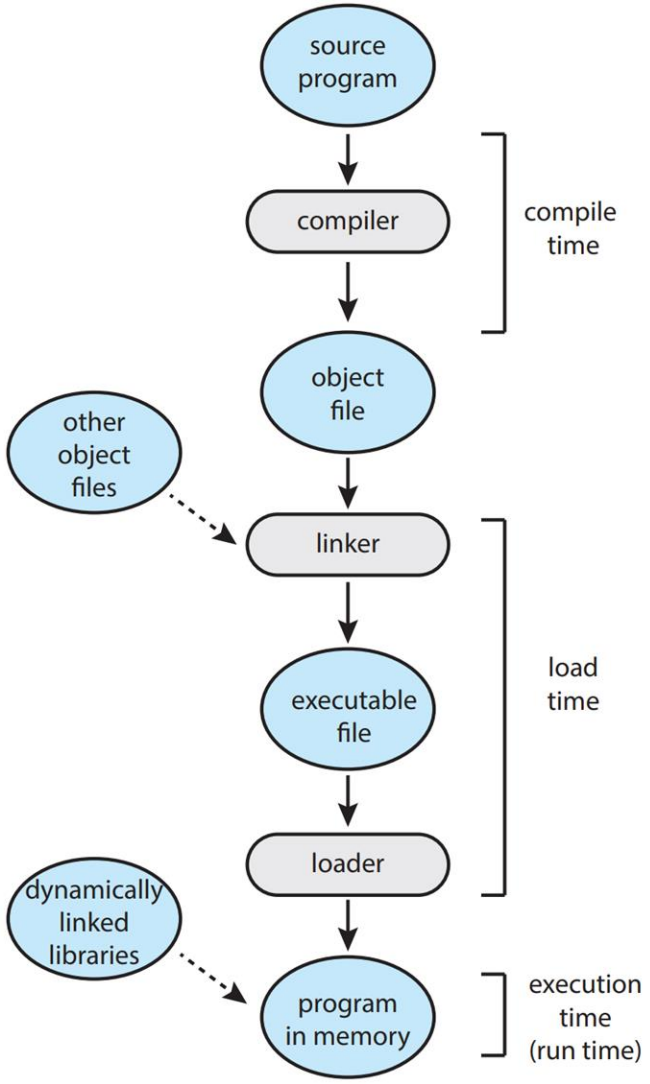
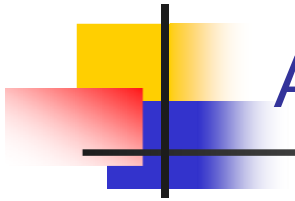
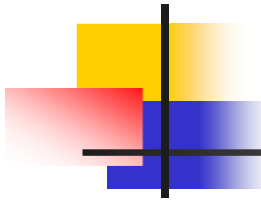


Figure 9.3 Multistep processing of a user program.



Address Binding

- 실행하려면 프로그램을 메모리로 가져와야 하고, 사용 가능한 CPU에서 실행.
- 프로세스가 실행되면 메모리에 저장된 명령 및 데이터에 액세스함.
- 결국 프로세스는 종료되고 메모리는 다른 프로세스에서 사용할 수 있도록 회수됨.

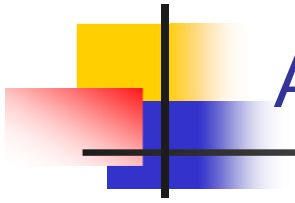


Address Binding

- **물리 주소 (physical address):** 실제 하드웨어 상의 주소. 논리 주소와는 아무런 관계가 없어야 하며, 메모리 주소 레지스터 (memory-address register)에 저장되는 주소.

- **논리 주소 (logical address):** 사용자 프로세스에서 사용하기 위해 CPU에서 생성된 주소.

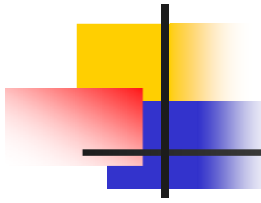
- 논리 주소를 물리 주소에 매핑하기 위해서는 논리적 주소 공간 (logical address space)와 물리적 주소 공간 (physical address space)이 분리되어야 함.



Address Binding

- 일반적으로 명령어와 데이터를 메모리 주소에 바인딩하는 작업은 모든 단계에서 수행할 수 있음.

- 1. Compile Time
- 2. Load Time
- 3. Execution Time (Run time)



Address Binding

- **1. Compile Time**
  - 프로세스의 물리적 주소는 컴파일 때 정해짐.
  - 프로세스가 메모리의 어느 위치에 들어갈지 미리 알고 있다면 컴파일러가 절대 주소(Absolute address), 즉 고정된 주소를 생성.
  - 컴파일 타임 주소 할당은 프로세스 내부에서 사용하는 논리적 주소와 물리적 주소가 동일하다.
  - 문제점: 주소가 고정되어 있기 때문에 메모리 상에 빈 공간이 많이 발생할 수 있어 비효율적이고, 로드하려는 위치에 이미 다른 프로세스가 존재할 수 있음.

Address Binding

- **2. Load Time**
  - 프로세스가 메모리의 어느 위치에 들어갈지 미리 알 수 없다면 컴파일러는 Relocatable code를 생성해함.
  - Relocatable code: 메모리의 어느 위치에서나 수행될 수 있는 기계 언어 코드.
  - Loader가 프로세스를 메모리에 load 하는 시점에 물리적 주소를 결정함. 따라서 논리적 주소와 물리적 주소가 다름.
  - 문제점: 프로세스 내에 메모리를 참조하는 명령어들이 많아서 이 주소를 다 바꿔줘야 하기 때문에, 로딩할 때의 시간이 매우 커질 수 있음

Address Binding

- **3. Execution Time (Run time)**
  - 프로세스 수행이 시작된 이후, 프로세스가 실행될 때 메모리 주소를 바꾸는 방법
    - 즉, Runtime때 물리적 주소가 결정되며 실행 도중에 주소가 바뀔 수 있음.
    - CPU가 주소를 참조할 때마다 address mapping table을 이용하여 binding을 점검.
  - MMU(Memory Management Unit)라는 하드웨어 장치를 사용하여 논리적 주소를 물리적 주소로 바꿔줌.
    - 프로세스가 CPU에서 수행되면서 생성해내는 모든 주소값에 대해서 base register의 값을 더해주어 물리적 주소를 생성.
    - base register는 하나이므로 프로세스끼리 공유함.

Address Binding

- **3. Execution Time (Run time)**

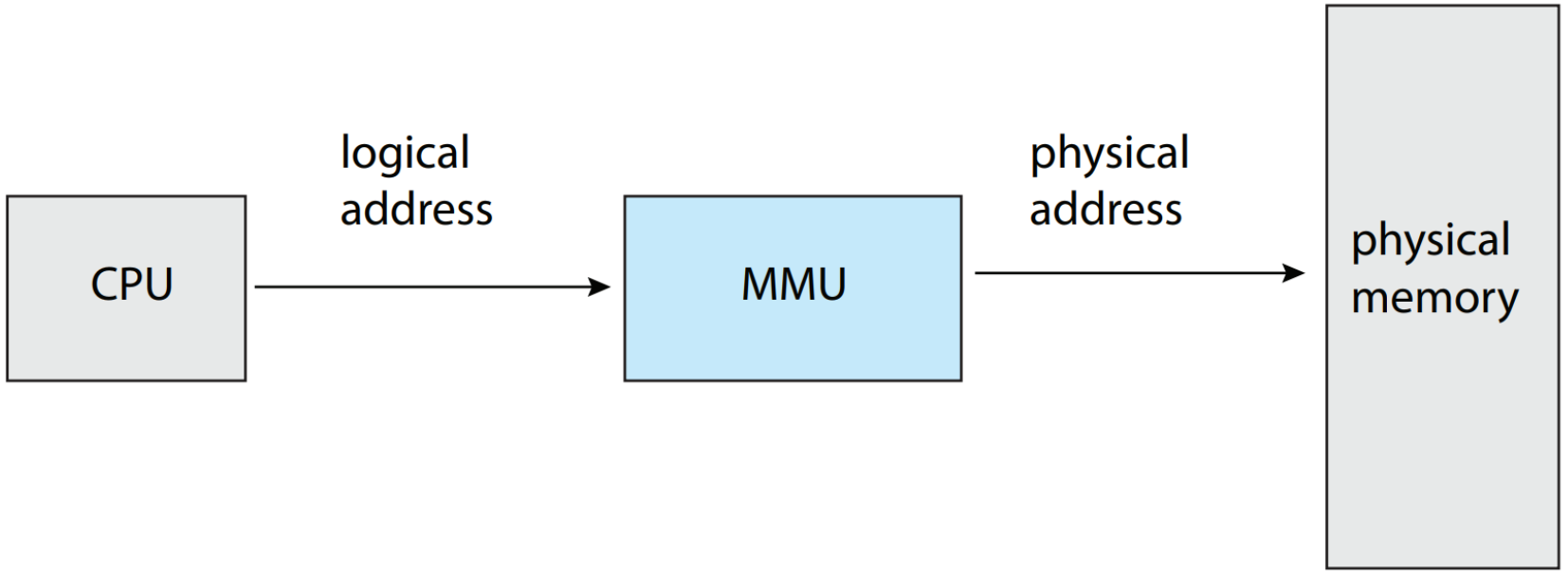


Figure 9.4 Memory management unit (MMU).

MMU(Memory Management Unit)

- 가상에서 물리적 주소로의 런타임 매핑을 수행하는 장치. 여기서는 base register를 relocation register라고 함.

- relocation register의 값은 주소가 메모리로 전송될 때 사용자 프로세스에 의해 생성된 모든 주소에 추가됨.
  - 예를 들어, 베이스가 14000에 있는 경우, 위치 0을 지정하려는 사용자의 시도는 위치 14000으로 동적으로 재배치됨.
  - 그러므로 위치 346에 대한 액세스는 위치 14346에 매핑됨.

- 사용자 프로그램은 실제 물리적 주소에 액세스하지 않음.

MMU(Memory Management Unit)

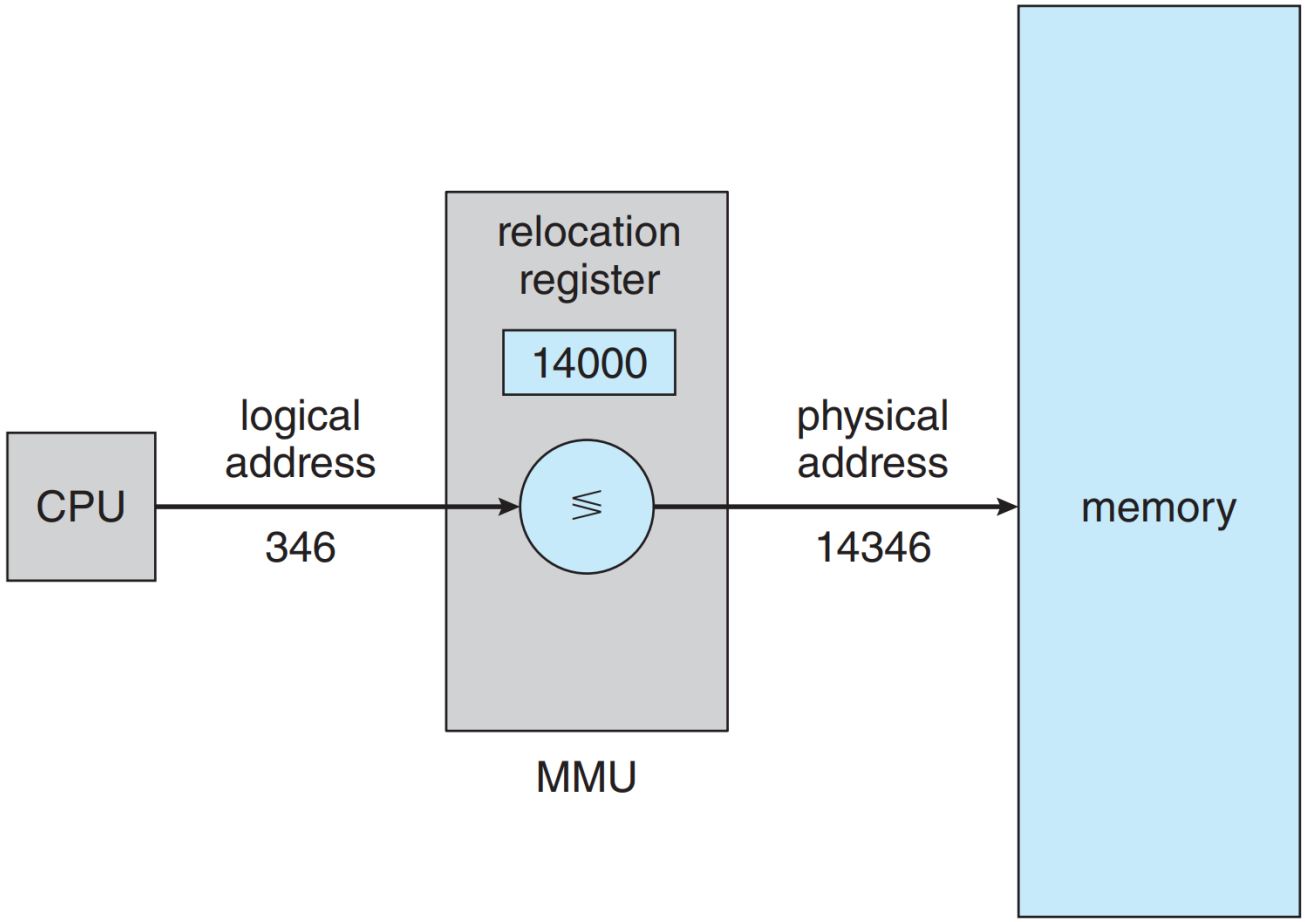
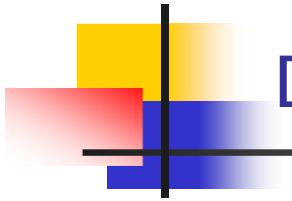
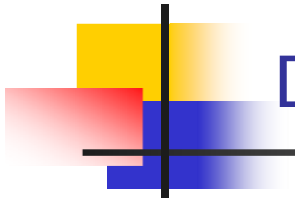


Figure 9.5 Dynamic relocation using a relocation register.



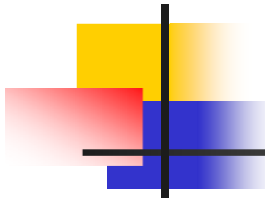
Dynamic loading

- 메모리 공간을 효율적으로 사용하기 위해 프로그램, 루틴을 한꺼번에 로딩하지 않고 필요할 때마다 로딩하는 방식.
- relocatable linking loader는 필요할 때마다 호출되어 주소 테이블 (address table)을 업데이트할 수 있음.



Dynamic loading

- **linking:** 다른 프로그램이나 외부 라이브러리를 가져와 연결하는 것.
- **DLL (dynamically linked library):** 프로그램 실행 중간에 사용자 프로그램에 linking 되어 사용할 수 있는 시스템 라이브러리.

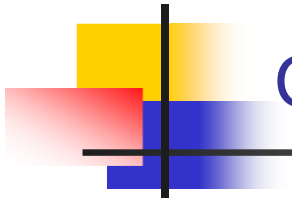


Linking

- **정적 링킹 (static linking):** 실행 파일을 만들때, 모든 라이브러리의 모듈을 복사하는 방식.
- 모두 한 실행파일에 포함되어 동적 링킹보다 빠름.

- **동적 링킹 (dynamic linking):** 실행 파일을 만들때, 모든 라이브러리 모듈의 주소만 가지고 있다가, linking 작업을 실행 시간 (execution time)에 수행.
- 매번 runtime때 마다 링킹해야하는 오버헤드가 존재하여 느림.

Contiguous Memory Allocation



Contiguous Memory Allocation

- 연속 메모리 할당은 프로세스를 단일 구역 (single section)에 할당하는 방식
- 여러 곳에 나눠서 할당되어있지 않고 한 구역에 할당되어 연속적임.

Contiguous Memory Allocation

- 연속 메모리 할당은 프로세스를 단일 구역 (single section)에 할당하는 방식
  - 여러 곳에 나눠서 할당되어있지 않고 한 구역에 할당되어 연속적임.

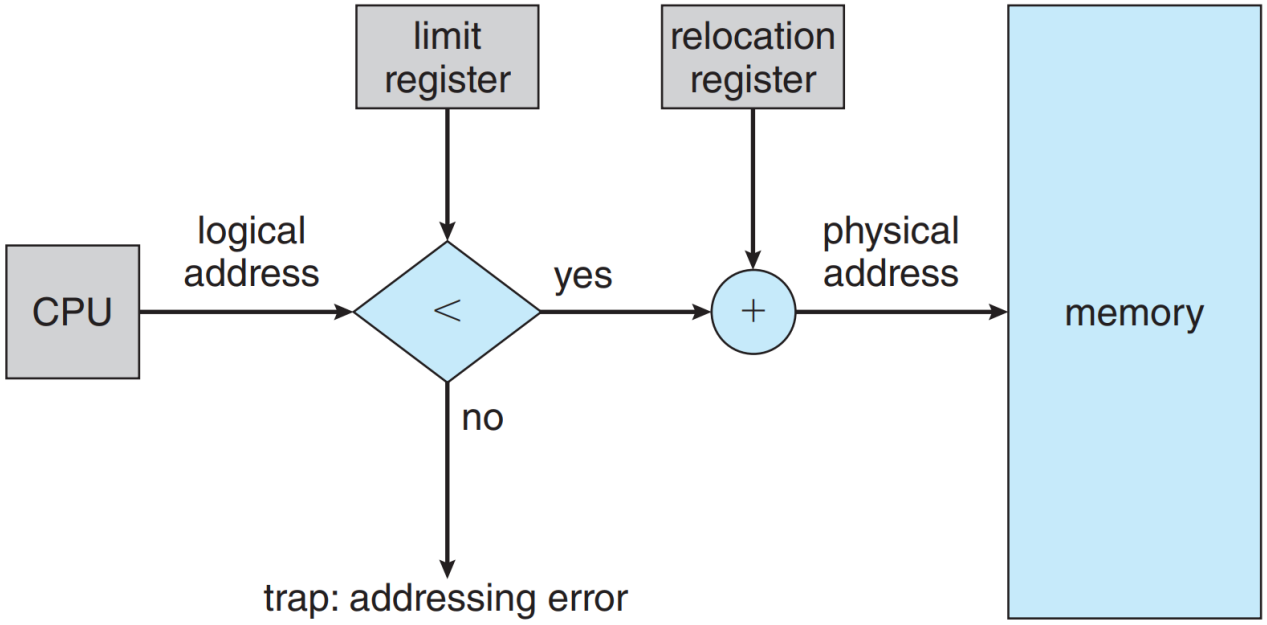


Figure 9.6 Hardware support for relocation and limit registers.

Memory Protection

- relocation register와 limit register를 통해 프로세스가 소유하지 않은 메모리에 액세스하는 것을 방지할 수 있음.
- 메모리 주소에 접근하고자 할 때 그 주소가 base와 limit 사이에 존재하는 주소 접근일 때에만 접근이 가능함. 그렇지 않으면 trap 발생.

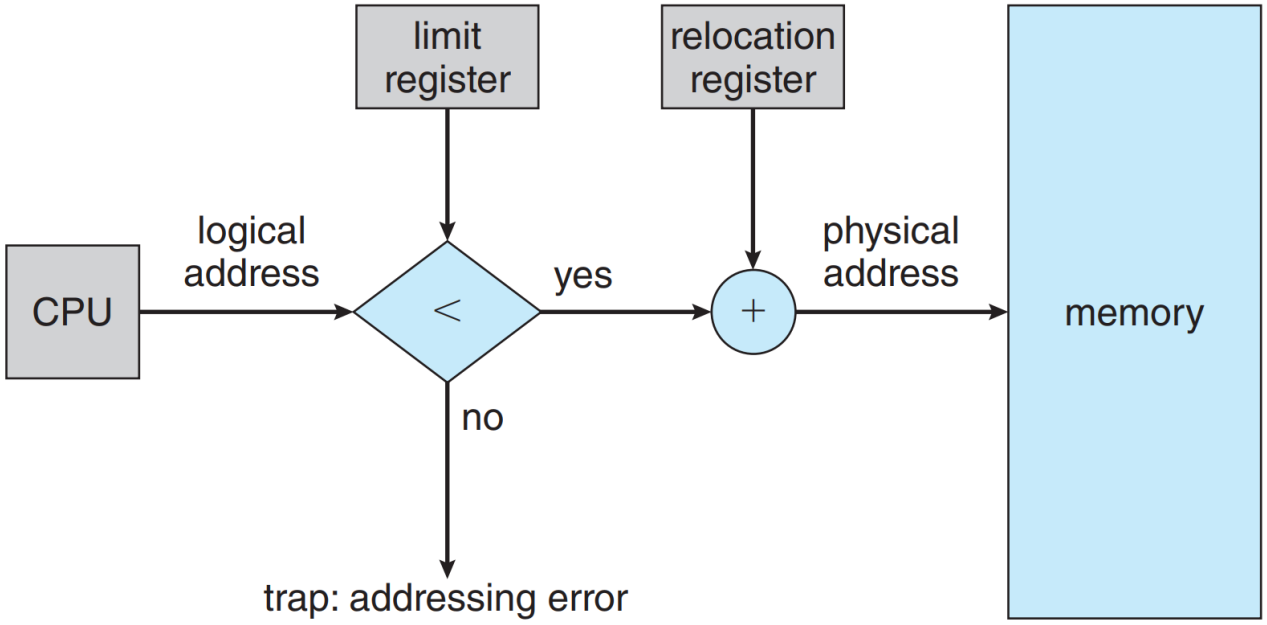


Figure 9.6 Hardware support for relocation and limit registers.

Memory Allocation

- 프로세스의 크기가 제각각이기 때문에 어떤 프로세스를 어떻게 할당할 것인지가 중요함.
- 프로세스의 할당 & 해제를 반복하다보면 빈 공간이 생기는데, 이를 구멍 (hole)이라고 부름. 이 구멍을 잘 관리하는 것이 중요함.

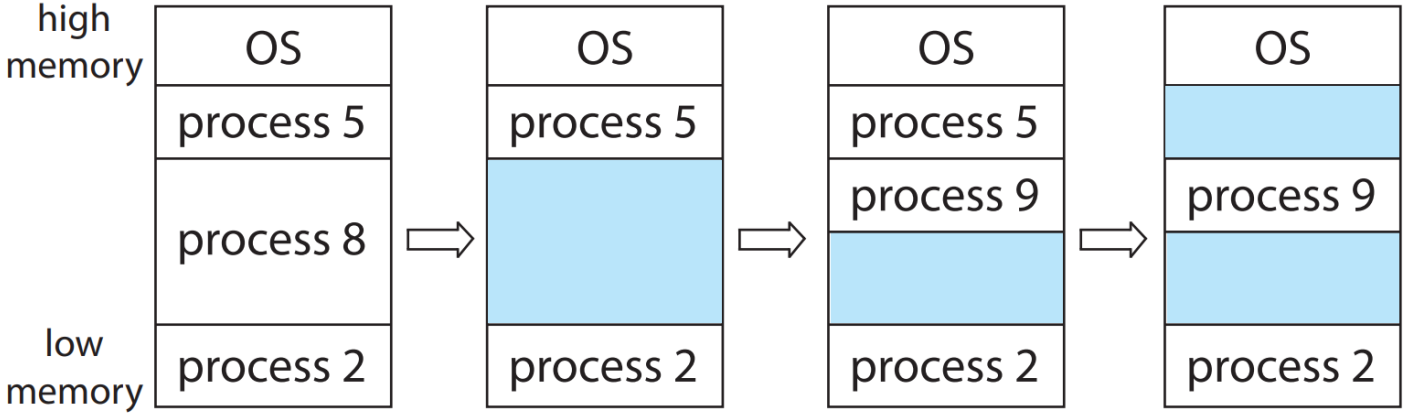
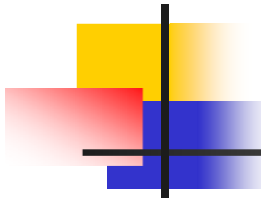


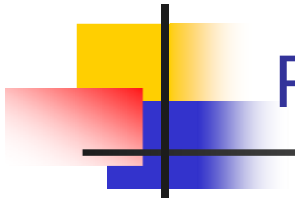
Figure 9.7 Variable partition.



Memory Allocation

- 저장 공간에 동적으로 메모리를 할당할 때 여러 구멍들 중에 크기 n만큼의 메모리를 어느 구멍에 할당할지에 대한 문제가 발생함.

- 3가지 해결 방법
  - **최초 적합 (first-fit):** 구멍들을 탐색하다가 할당할 수 있는 가장 첫 번째 구멍에 할당. 즉, 크기만 맞으면 할당.
  - **최적 적합 (best-fit):** 할당할 수 있는 가장 작은 구멍에 할당
  - **최악 적합 (worst-fit):** 할당할 수 있는 가장 큰 구멍에 할당



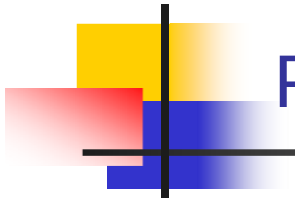
Fragmentation

- **외부 단편화 (external fragmentation):**
  hole의 크기가 하나의 프로세스의 크기보다 작아, 메모리 할당이 불가능한 상태.

- **내부 단편화 (internal fragmentation):**
  빈 공간에 메모리를 할당하고 난 뒤 남은 작은 공간이 너무 많아, 다른 메모리를 할당할 수 없는 상태

- Paging을 하면 내부 단편화 문제가,
  Contiguous Memory Allocation을 하면 외부 단편화 문제가 발생함.

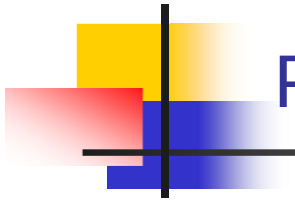
Paging



Paging

- 프로세스의 물리적 주소 공간을 불연속적 (non-contiguous)으로 쪼개서 관리하는 것.
- 페이징의 기본적인 방법은 메모리를 동일한 크기로 쪼개는 것.

- **프레임 (frame):** 물리 메모리를 고정 크기의 블록으로 나눈 것
- **페이지 (page):** 논리 메모리를 같은 크기로 나눈 것

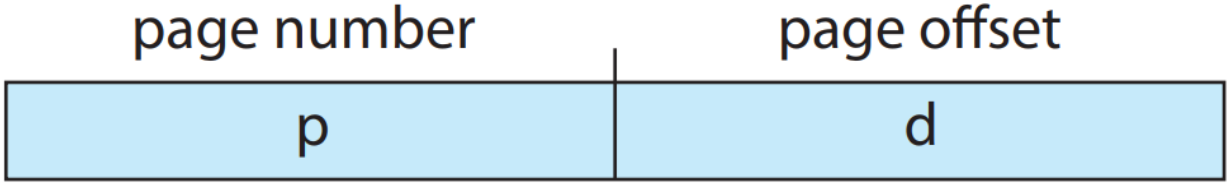


Paging

- 페이징에서는 논리 메모리에 있는 프로그램을 물리 메모리에 올릴 때, Contiguous Memory Allocation처럼 프로그램 하나를 통째로 올릴 필요 없음.
- 페이지를 프레임에 올리기만 하면 됨.
- 논리 주소 공간과 물리 주소 공간이 완벽하게 분리됨.

Paging

- CPU에서 생성된 모든 주소는 page number(p)와 page offset (d)의 두 부분으로 나뉨.
  - 주소를 넘겨줄 때 page number(p)와 page offset (d) 만 넘겨주면 됨.



Paging

- 프로세스마다 페이지의 개수가 다르기 때문에 페이지 테이블(page table)을 통해 관리해 주어야 함.
- 페이지 번호를 통해 페이지 테이블에서 프레임 번호를 알 수 있음.

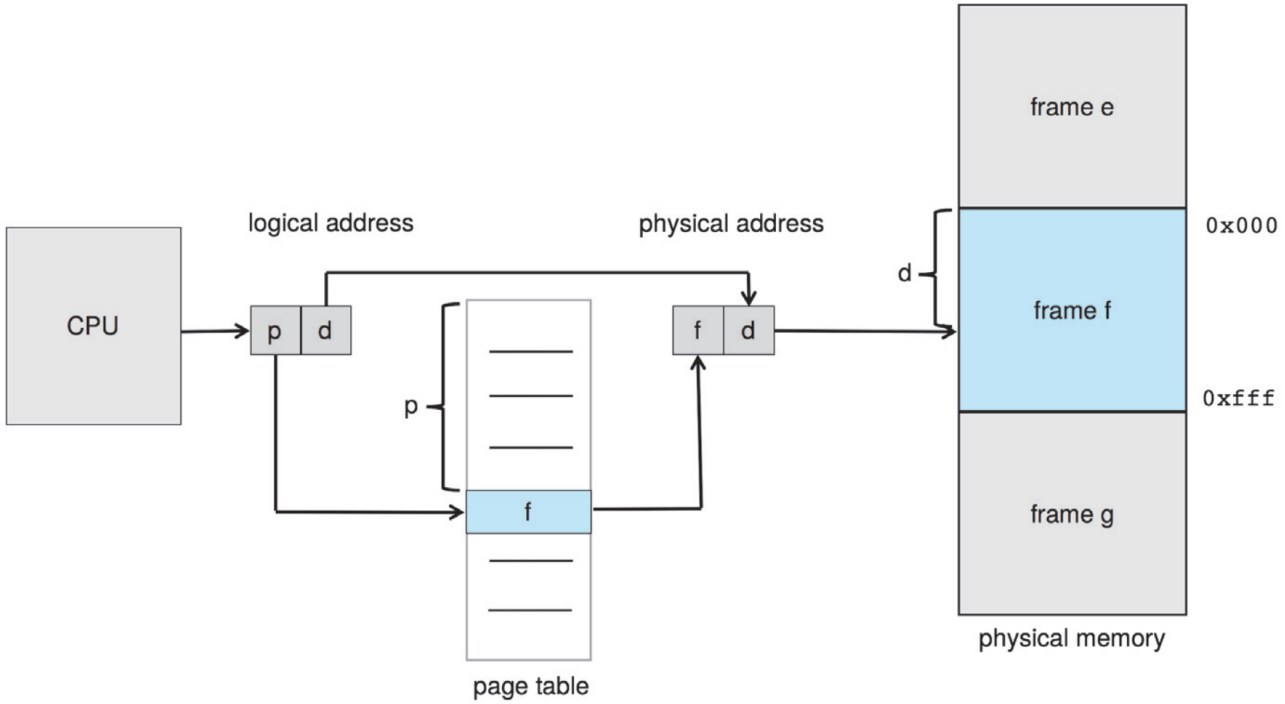


Figure 9.8 Paging hardware.

Paging

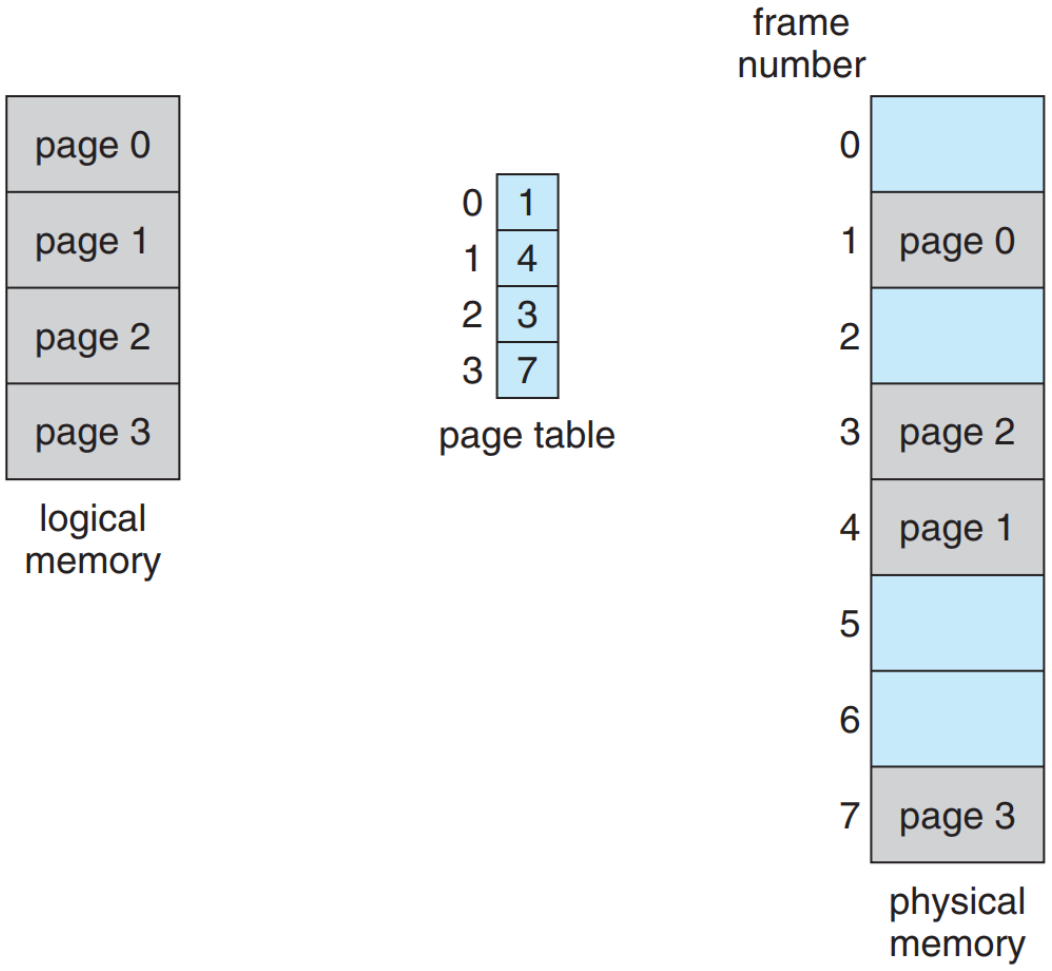
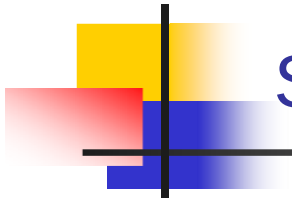


Figure 9.9 Paging model of logical and physical memory.

Structure of the Page Table



Structure of the Page Table

- 현대적인 시스템의 경우 논리 주소 공간이 매우 크기 때문에 페이지 테이블 또한 매우 커짐.
- 따라서 페이지 테이블을 구조화하는 것이 필요함.

Hierarchical paging

- 페이지 테이블의 페이지 테이블을 만들어 관리하는 것.
- 페이지 테이블 자체가 너무 커지는 것을 방지하기 위해 선택함.

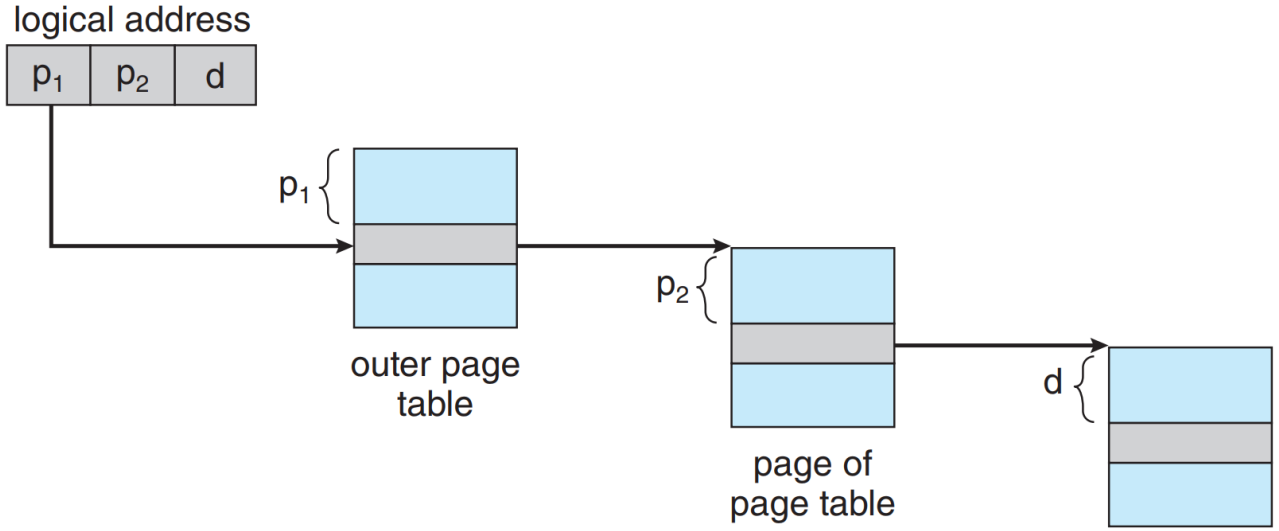
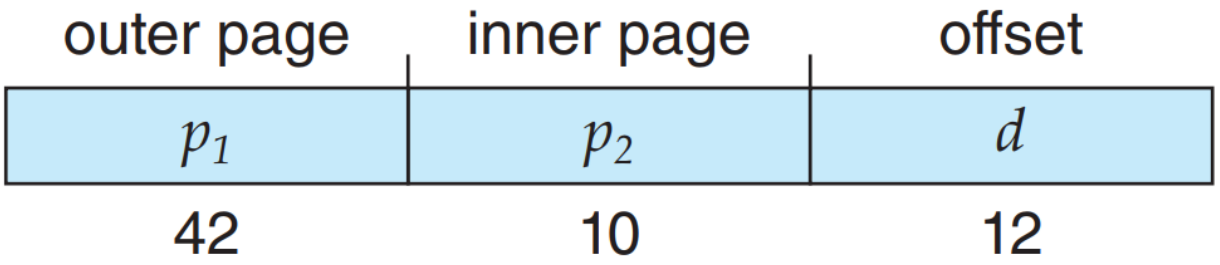
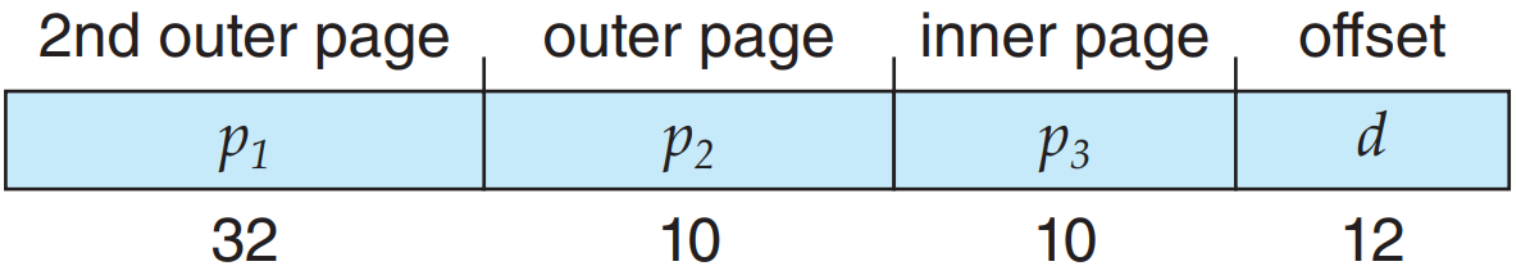


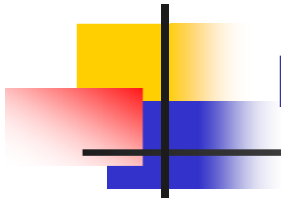
Figure 9.16 Address translation for a two-level 32-bit paging architecture.

Structure of the Page Table

64-bit address space







Hashed page table

- 해싱 자료구조가 O(1)의 시간 복잡도를 갖기 때문에 효율적임. 해싱을 하드웨어적으로 구현한 것이 해시 페이지 테이블.
- 논리 주소의 p, d 값으로 해시 함수를 통해 해시 값을 생성한 뒤, 이 값을 물리 주소의 값으로 사용하여 매핑함.
- 주소 공간의 크기가 32 bit를 넘어갈 때 자주 사용하는 방법.

Hashed page table

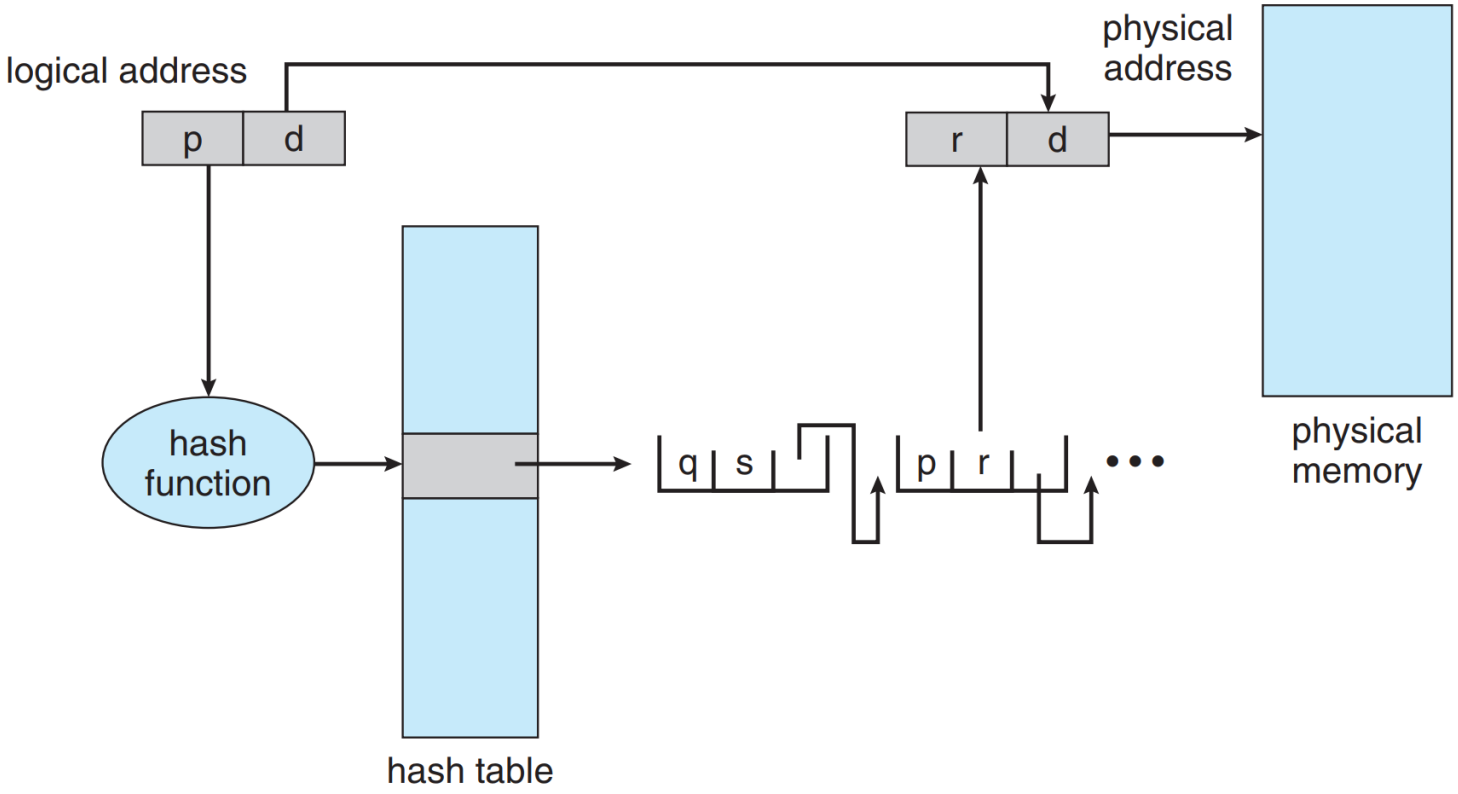


Figure 9.17 Hashed page table.

Inverted page table

- 메모리에 하나의 고정크기 페이지 테이블만 두는 방법. 즉 모든 프로세스는 한 개의 페이지 테이블만 사용.
- 페이지 테이블 엔트리 갯수 = 프레임 수
- 페이지 테이블 하나가 메인 메모리에 그대로 매핑됨.
- pid (process identifier)를 추가하여 해당 pid의 페이지를 역으로 저장함.

Inverted page table

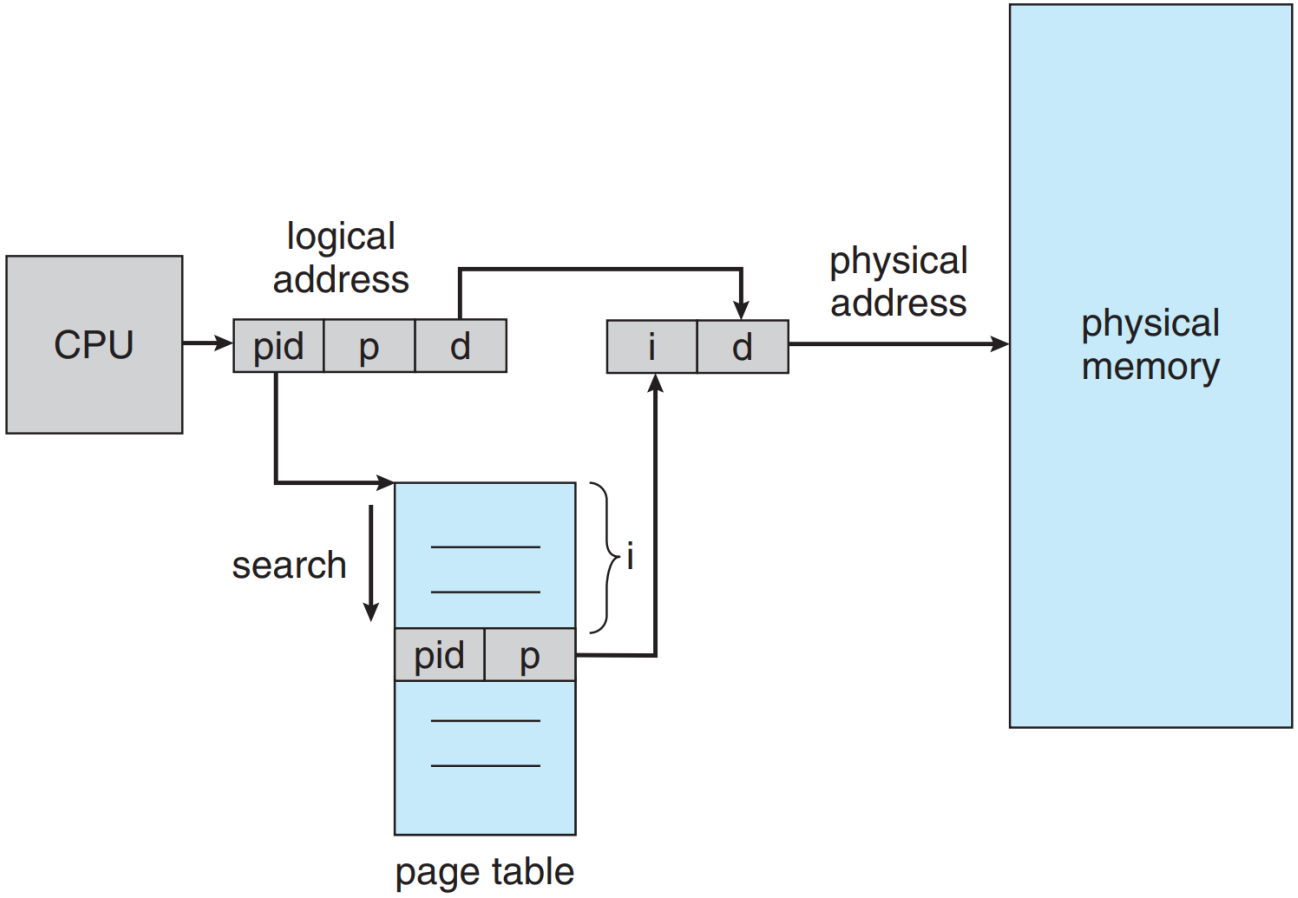
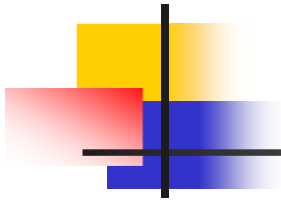
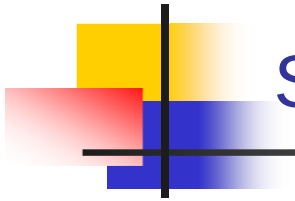


Figure 9.18 Inverted page table.



Inverted page table

- **단점 :**
  - 1. 프로세스간 메모리 공유 불가함. 같은 프로세스 내에서도 페이지가 다르면 메모리 공유 불가
  - 2. 페이지 테이블에서 pid와 page number을 찾기 위해 탐색하는 데에 오버헤드 발생.



Swapping

- 프로세스 명령어와 해당 명령어가 작동하는 데이터는 실행될 메모리에 있어야 함.
- 그러나 프로세스 또는 프로세스의 일부를 일시적으로 메모리에서 백업 저장소로 스왑한 다음 계속 실행하기 위해 메모리로 다시 가져올 수 있음.
- 스와핑은 모든 프로세스의 전체 물리적 주소 공간이 시스템의 실제 물리적 메모리를 초과할 수 있게 할 수 있게 함.

Swapping

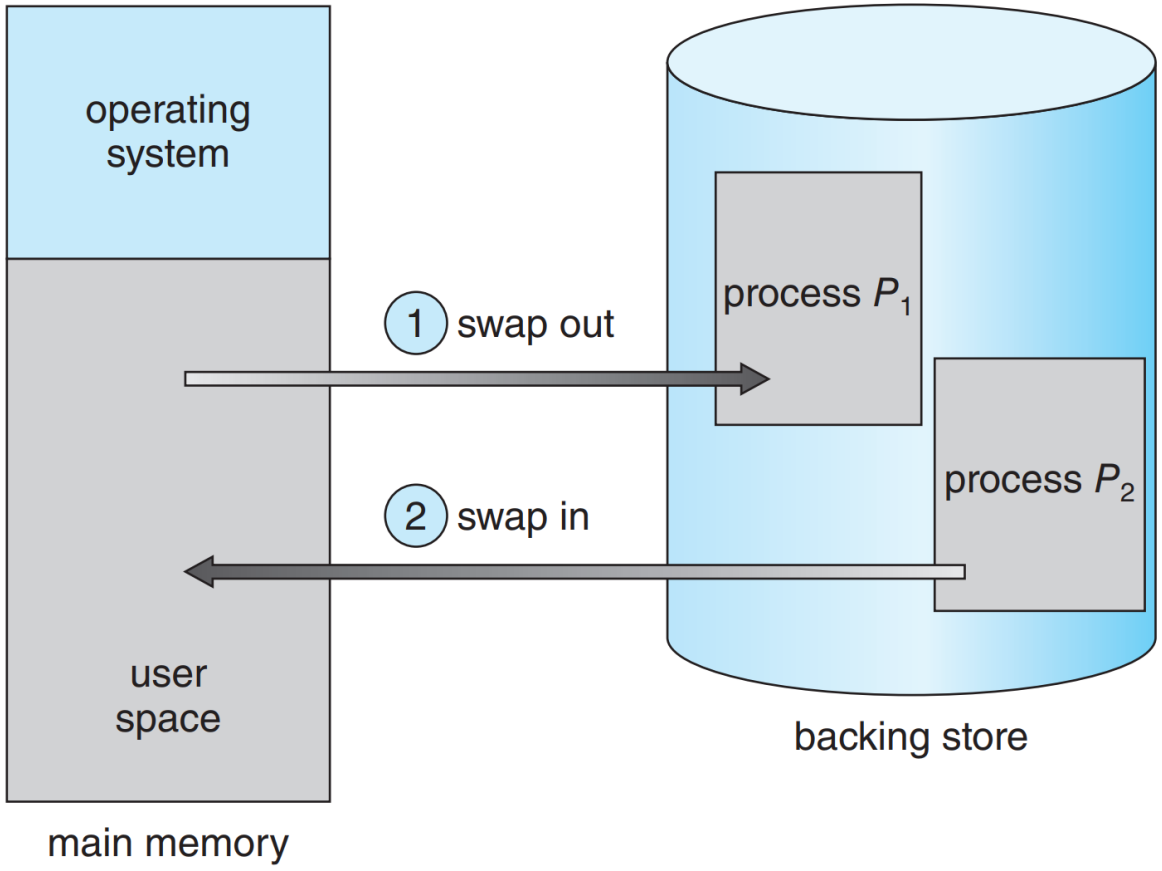
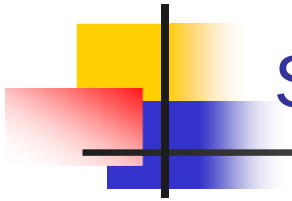


Figure 9.19 Standard swapping of two processes using a disk as a backing store.



Standard Swapping

- Standard Swapping은 메인 메모리와 백업 저장소 사이에서 전체 프로세스를 이동 시키는 것.
- 백업 저장소는 일반적으로 빠른 secondary storage.

- Standard Swapping의 장점은 물리적 메모리를 초과 할당할 수 있으므로 시스템이 프로세스를 저장할 실제 물리적 메모리 보다 더 많은 프로세스를 수용할 수 있다는 것
- idle 상태이거나 대부분이 idel 상태인 프로세스는 스와핑에 적합한 후보

Swapping with Paging

- Standard Swapping은 최신 OS들은 보통 사용하지 않음
- 메모리와 백업 저장소 간에 전체 프로세스를 이동하는 데 필요한 시간이 길기 때문임.

- 보통 전체 프로세스가 아닌 **프로세스의 페이지를 스왑**할 수 있는 스와핑의 변형을 사용함.
- 이 전략은 물리적 메모리를 훨씬 더 많이 할당되도록 허용하지만 스와핑에 포함되는 페이지 수는 적기 때문에 전체 프로세스를 스와핑하는 비용이 발생하지 않음.

Swapping with Paging

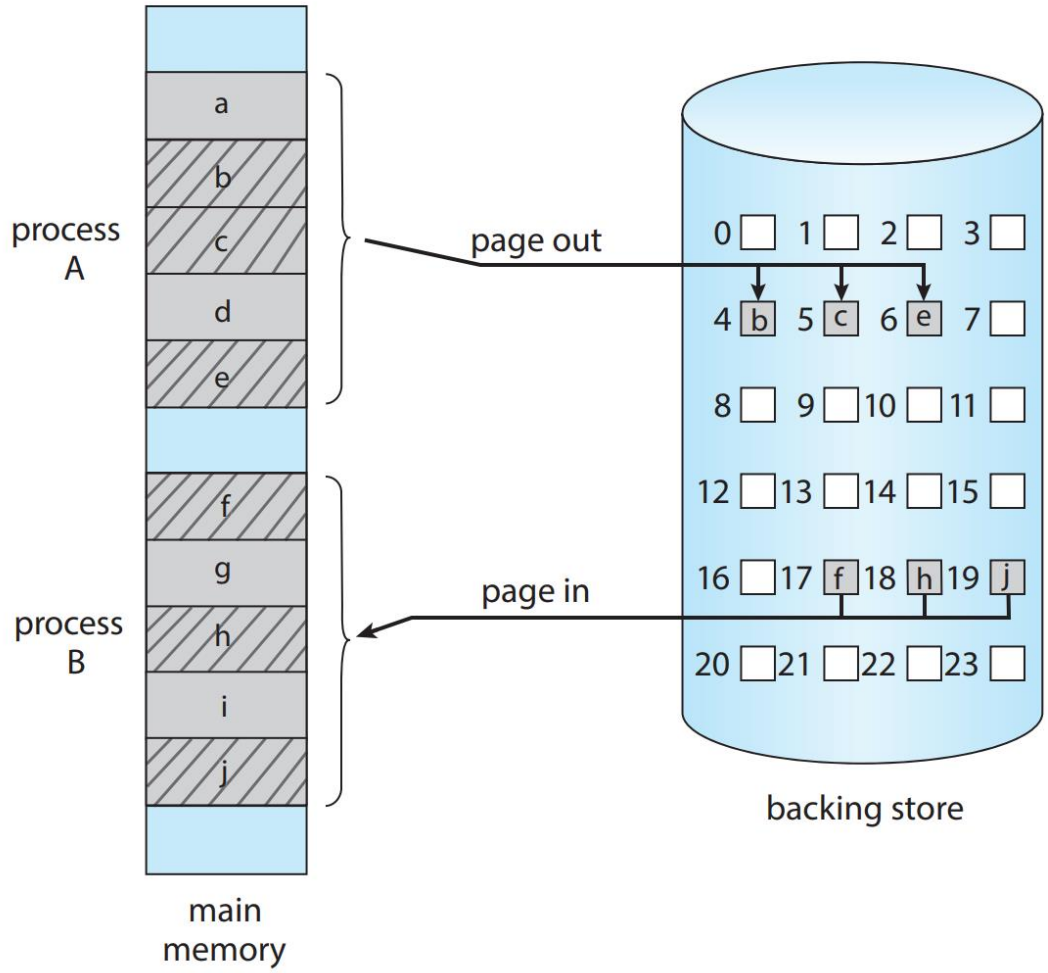


Figure 9.20 Swapping with paging.

Chapter 9
Finish