Operating System

Memory Mangement(1/2)

교재 **Chap 8, 9 265~356**

강의 #09

Contents

1. 주소 공간

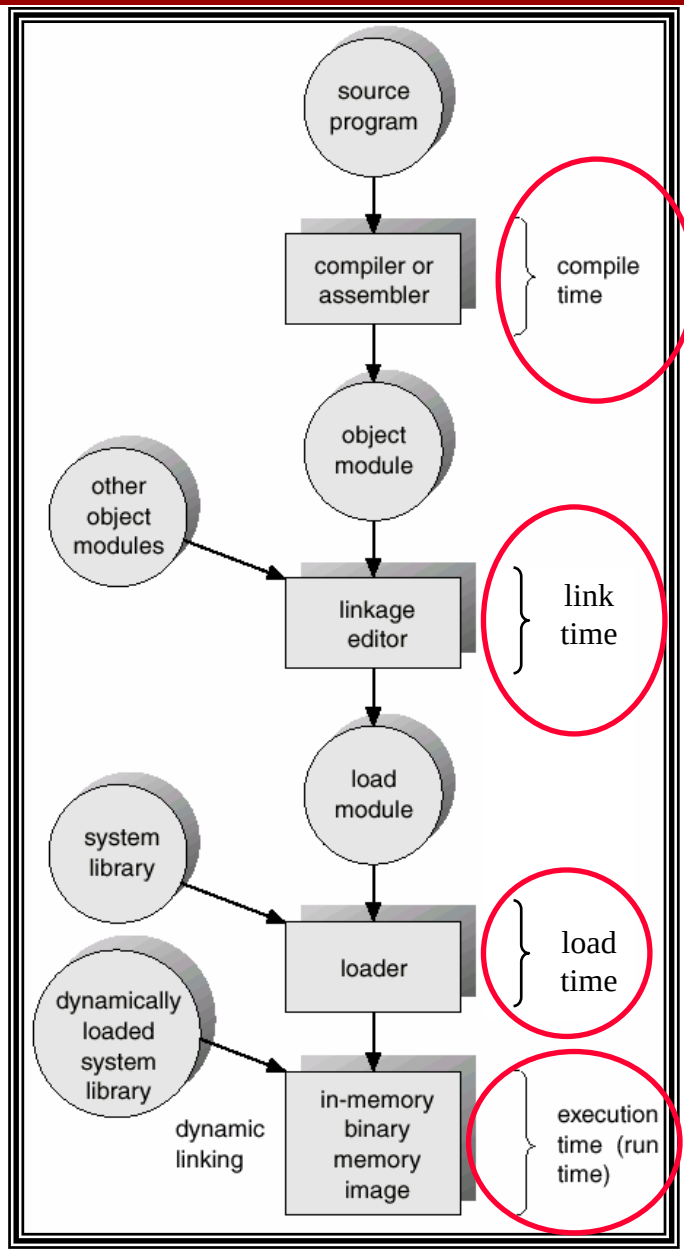
- 정의 (address space)
  - 프로세스에서 참조 할 수 있는 주소들의 범위(집합)
  - 프로세스와 1-1의 관계
  - 사용자 쓰레드는 주소공간을 공유함
- 주소 공간의 크기
  - CPU의 주소 버스(address bus)의 크기에 의해 의존
  - 주소 버스가 32bit 인 시스템에서 주소 공간의 크기
    - 2^{32}개의 서로 다른 주소에 대한 식별자를 만들 수 있으므로
    - 0부터 (2^{32}-1) 까지의 주소 범위를 addressing 할 수 있다.

2. 물리 주소와 가상 주소

- 물리 주소 (physical address)
  - 컴퓨터의 메인 메모리를 접근할 때 사용되는 주소
  - 기억 장치의 주소 레지스터에 적재되는 주소
- 가상 주소 (logical address or virtual address)
  - 프로세스의 관점에서 사용하는 주소
  - CPU 관점의 주소는 물리 주소도 될 수 있고 가상 주소도 될 수 있다.
    - 어느 메모리 모델을 사용하느냐에 따라 달라진다.
  - Logical 이기 때문에 주소공간을 의미 있는 단위로 나누어 사용할 수 있음

초창기 컴퓨터의 주소 관리

- 물리주소를 Compile time에 생성
  - 프로세스가 물리메모리에서 실행되는 주소를 compile time에 알아서 compiler는 절대 코드를 생성한다.
  - 시작 주소의 위치가 바뀌어야 할 경우에는 다시 compile을 해야 한다.
  - 예) MS-DOS의 .COM 형식의 프로그램
- 다양한 프로그램이 돌아가게 됨에 따라 compile time에 물리 주소를 정하기 어려움 → 가상 주소를 생성하기 시작함.


source program
compiler or assembler
compile time
object module
other object modules
linkage editor
link time
load module
system library
loader
load time
dynamically loaded system library
dynamic linking
in-memory binary memory image
execution time (run time)

다양한 가상주소의 생성

Compile및 Link 가상 주소

- Compile time
  - Compiler가 symbol table을 만들고 주소는 symbol table relative한 주소로 이루어짐.
  - 컴파일된 object 파일은 주소 0 부터 시작함 (relocatable)
- Link time
  - object 파일들과 시스템에서 제공하는 library들을 묶어서 symbol table에 의존적이 아닌 주소를 만들어 냄 (address resolution)
  - Link의 결과로 하나의 executable파일이 만들어짐 주소는 0 부터 시작함
  - Executable은 하나의 주소 공간으로 0 부터 시작함

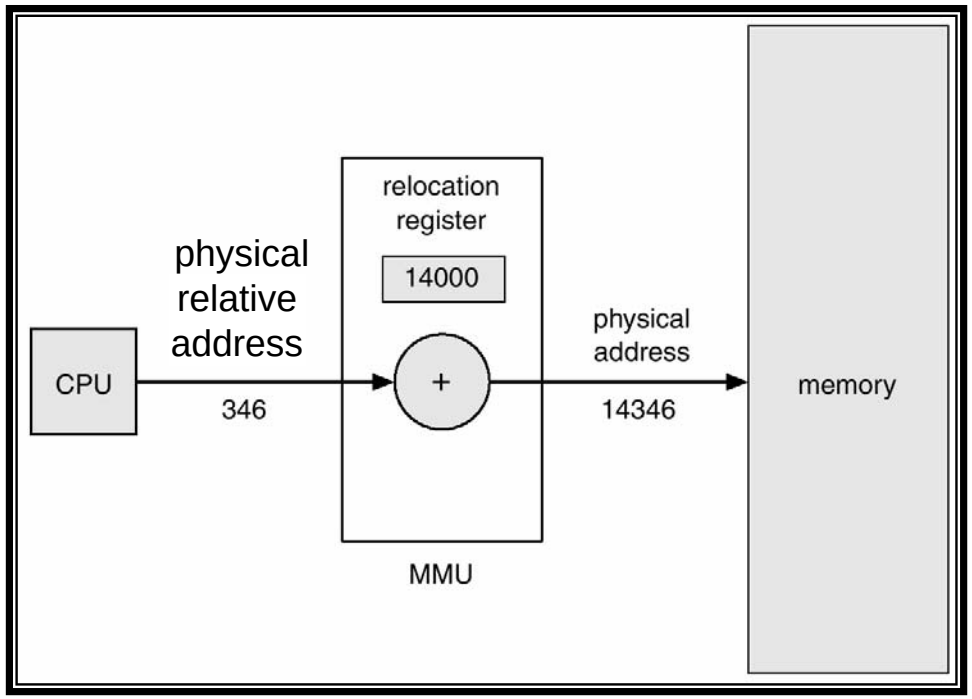
Load및 수행시 가상 주소

- Load time
  - 프로그램이 실행을 위해 loader는 executable을 메모리로 load한다.
  - 주소공간 전체가 메모리에 올라간다면, load시에 물리주소에 대한 binding이 일어난다.
    - 프로그램은 relocatable 주소로 되어 있기 때문에 뒤에 배울 base register를 통해서 물리 주소로 바뀌어 실행 하게 된다.
  - 만일 프로그램의 시작 주소를 바꾸려면, 다시 load를 해야 한다.

- Execution time
  - 프로세스가 실행할 때 물리 주소가 바뀌는 경우, 물리 주소에 대한 binding은 프로세스가 실행 할 때 일어난다. (나중에 배울 paging이나 swapping을 통해서 프로세스가 메모리에 올려지는 물리 주소는 바뀌어 질 수 있다.)
  - 이러한 형태의 주소 결정 방법을 사용하기 위해서는 MMU와 같은 특별한 하드웨어가 필요하게 된다.
  - 대부분의 general-purpose 운영체제에서는 이 방식을 사용

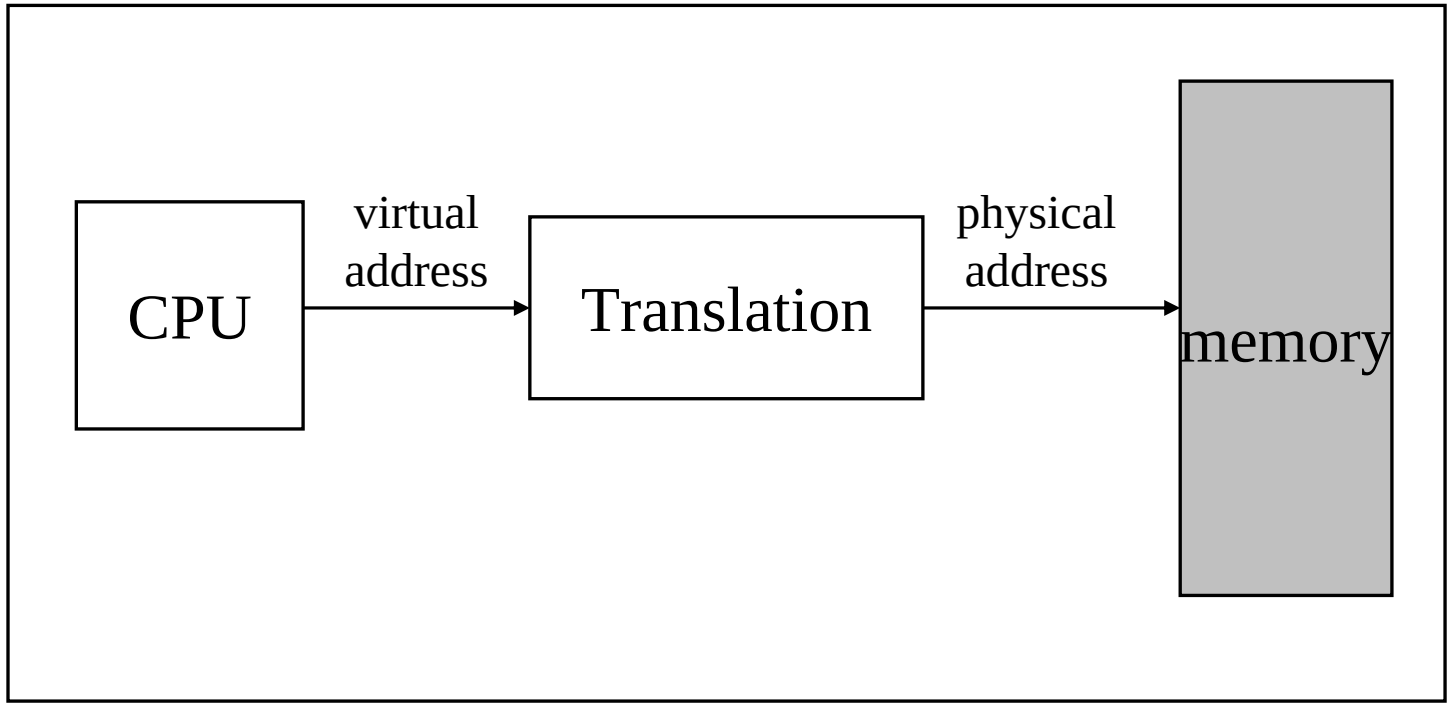
2.2 CPU에서 사용하는 주소에 따른 변환 방법

- CPU에서 physical relative address를 사용하는 경우
  - 프로그램내의 instruction들의 주소를 시작 주소(base address)부터 상대적인 offset으로 표현한 방법
  - 시작 주소가 결정 되면 시작 주소 + 상대 주소 의 합으로 절대 주소를 생성 할 수 있다.



2.2 CPU에서 사용하는 주소에 따른 변환 방법(cont.)

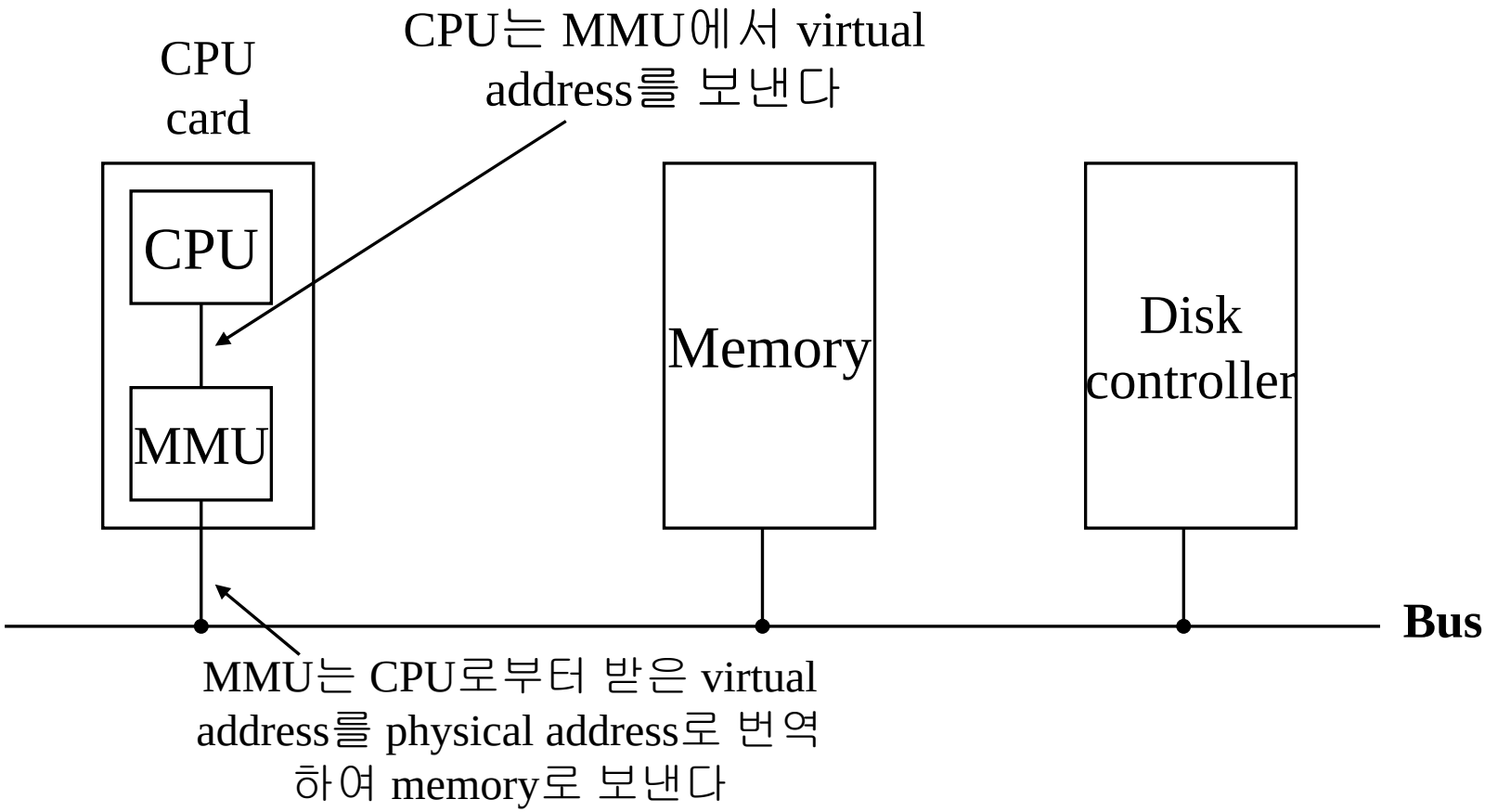
- CPU에서 Virtual address를 사용하는 경우



- Translation의 속도가 중요한 요소가 된다.

2.3 Memory Management Unit (MMU)

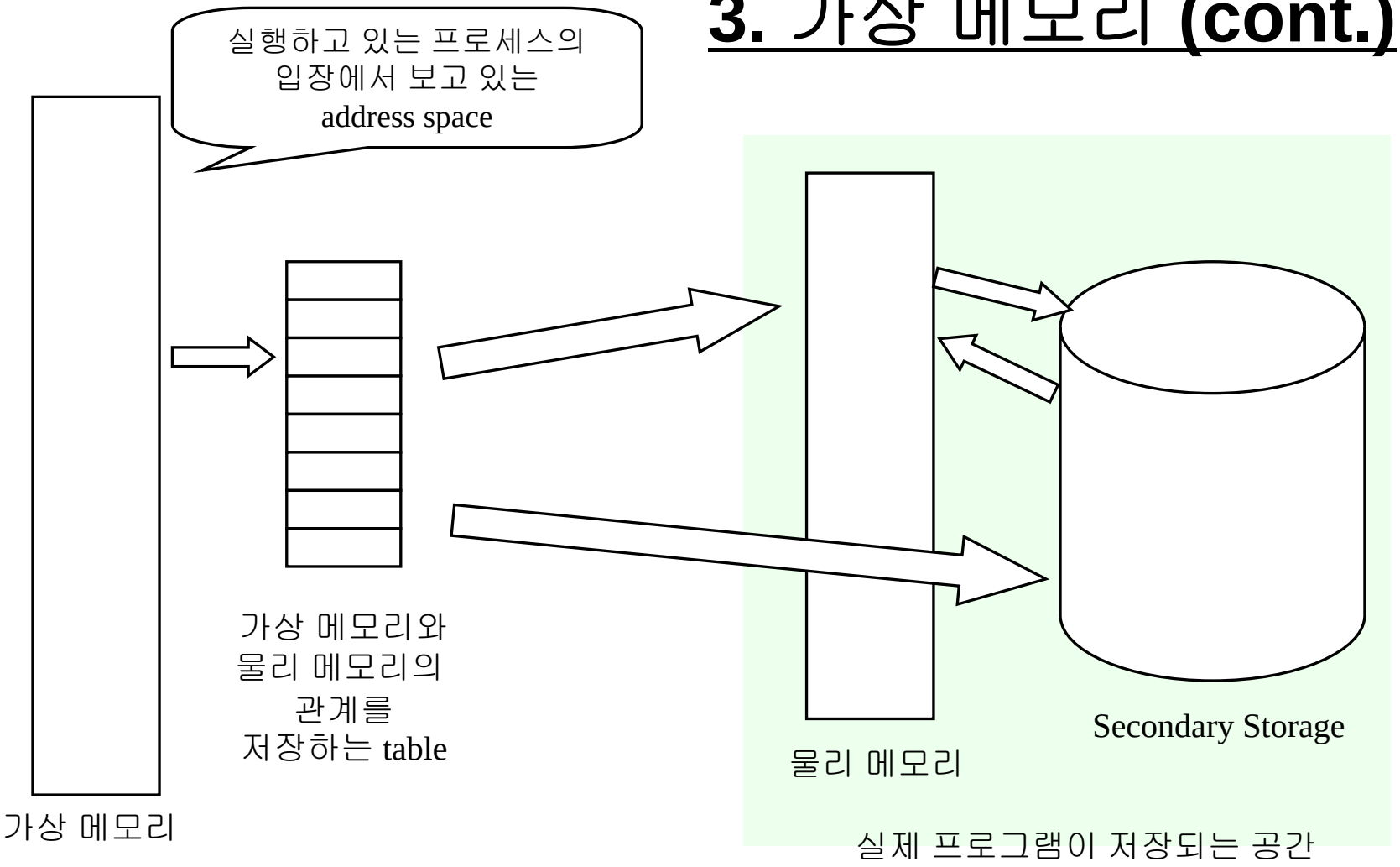
- Virtual address와 Physical address 간의 변환을 수행하는 hardware 장치



3. 가상 메모리

- 정의
  - 메모리로서 실제 존재하지는 않지만 사용자에게 메모리로써의 역할을 하는 메모리 (virtual)
- Basic Idea
  - 프로세스가 수행 되기 위해서 프로그램의 모든 부분이 실제 메모리(physical memory)에 있을 필요는 없다.
    - 현재 실행되고 있는 code 부분만이 실제 메모리에 있으면 프로세스는 실행이 가능

3. 가상 메모리 (cont.)



물리 메모리 보다 큰 가상 메모리의 diagram

4. Paging

- 메모리를 여러 개의 공간으로 나누어 관리
  - 프레임 (Frame)
    - 물리 메모리를 고정된 크기로 나누었을 때, 하나의 블록
  - 페이지 (Page)
    - 가상 메모리를 고정된 크기로 나누었을 때, 하나의 블록
  - 각각의 프레임 크기와 페이지 크기는 같다.

- 페이지가 하나의 프레임을 할당 받으면, 물리 메모리에 위치하게 된다.
  - 프레임을 할당 받지 못한 페이지들은 외부 저장장치 (Backing store)에 저장된다.
    - Backing store도 페이지, 프레임과 같은 크기로 나누어져 있다.

4. Paging (cont.)

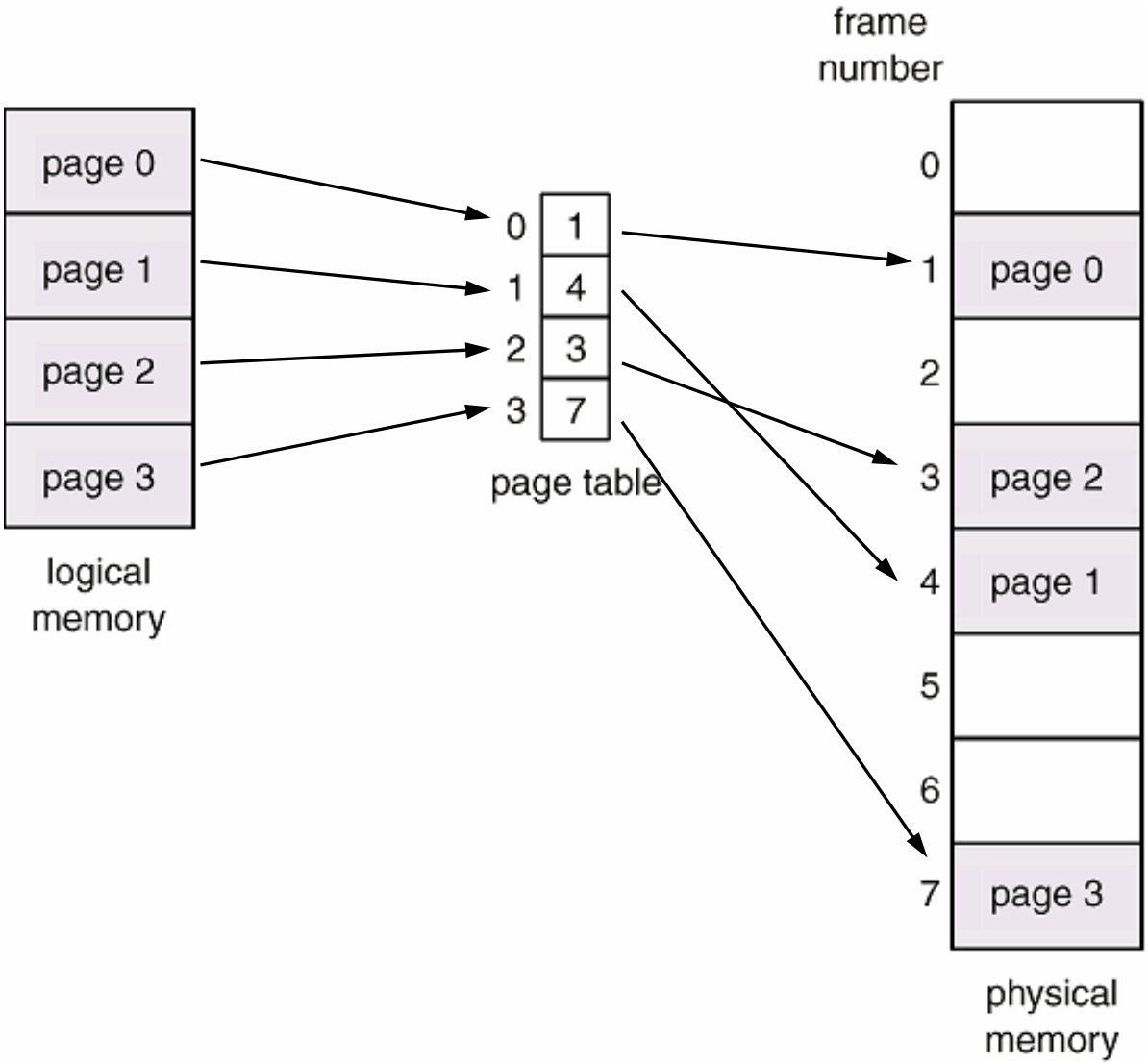
- CPU가 관리하는 모든 주소는 두 부분으로 나뉜다.
  - 페이지 번호 (Page number)
    - 각 프로세스가 가진 페이지 각각에 부여된 번호
    - 예) 1번 프로세스는 0부터 63번까지의 페이지를 가지고 있음
  - 페이지 주소 (Page offset)
    - 각 페이지의 내부 주소를 가리킴
    - 예) 1번 프로세스 12번 페이지의 34번째 데이터

4.1 Page table

- 페이지 테이블 (Page Table)
  - 각 프로세스의 페이지 정보를 저장함
    - 프로세스마다 하나의 페이지 테이블을 가짐
  - 인덱스 : 페이지 번호
  - 내용 : 해당 페이지에 할당된 물리 메모리(프레임)의 시작 주소
    - 이 시작 주소와 페이지 주소를 결합하여 원하는 데이터가 있는 물리 메모리 주소를 알 수 있다.

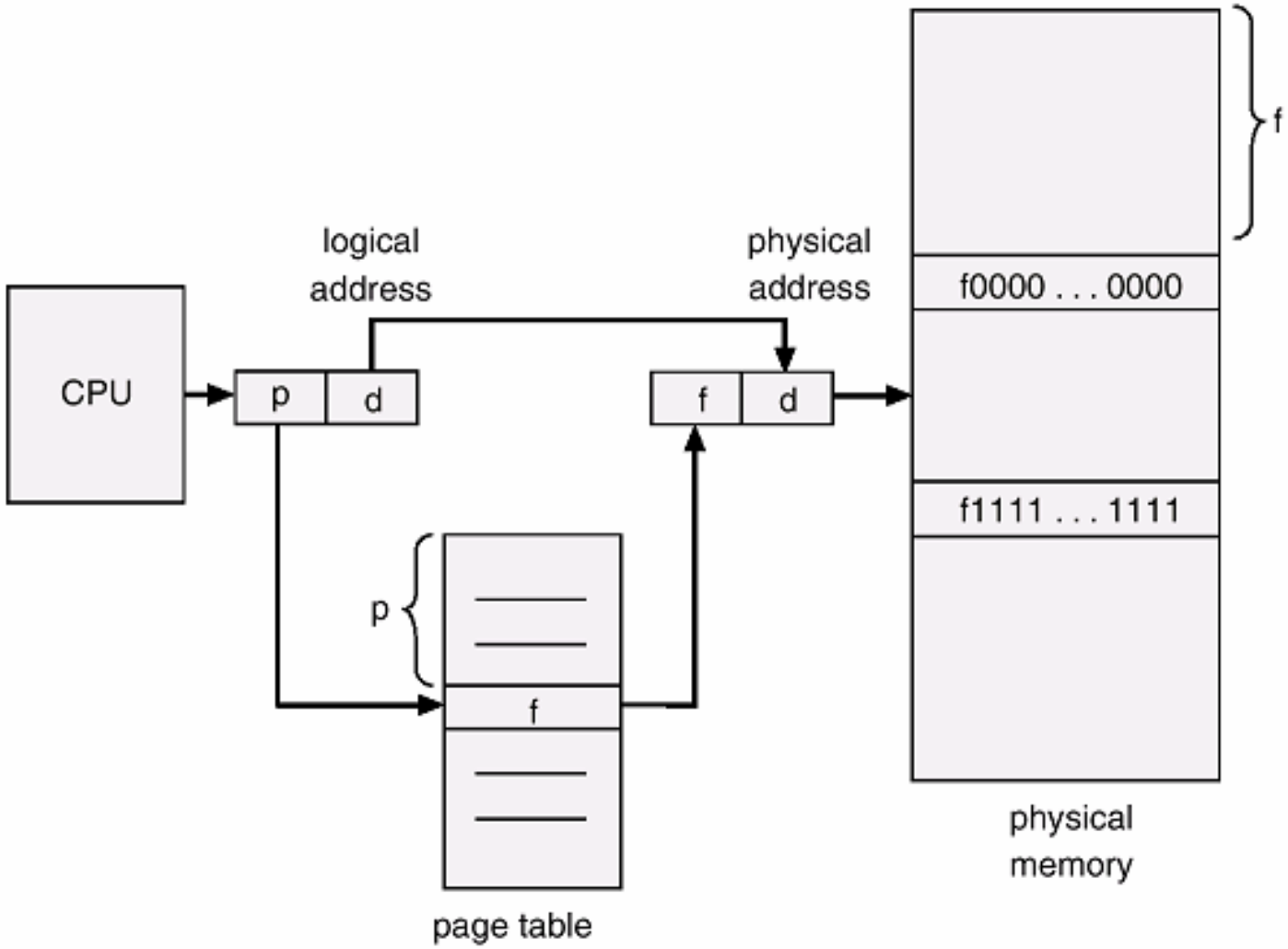
- 페이지 테이블의 구현
  - 페이지 테이블은 물리 메모리에 위치함
  - 페이지 테이블 기준 레지스터 (PTBR: Page-table base register)가 물리 메모리 내의 페이지 테이블을 가리킴
  - 페이지 테이블 길이 레지스터 (PRLR: Page-table length register)가 페이지 테이블의 사이즈를 나타냄

4.1 Page table (cont.)



- 가상 메모리와 물리 메모리 사이의 페이징 모델

4.1 Page table (cont.)



- 페이지 테이블을 이용해 주소를 해석하여 물리 메모리에 접근

4.2 Page Table Entry (PTE)

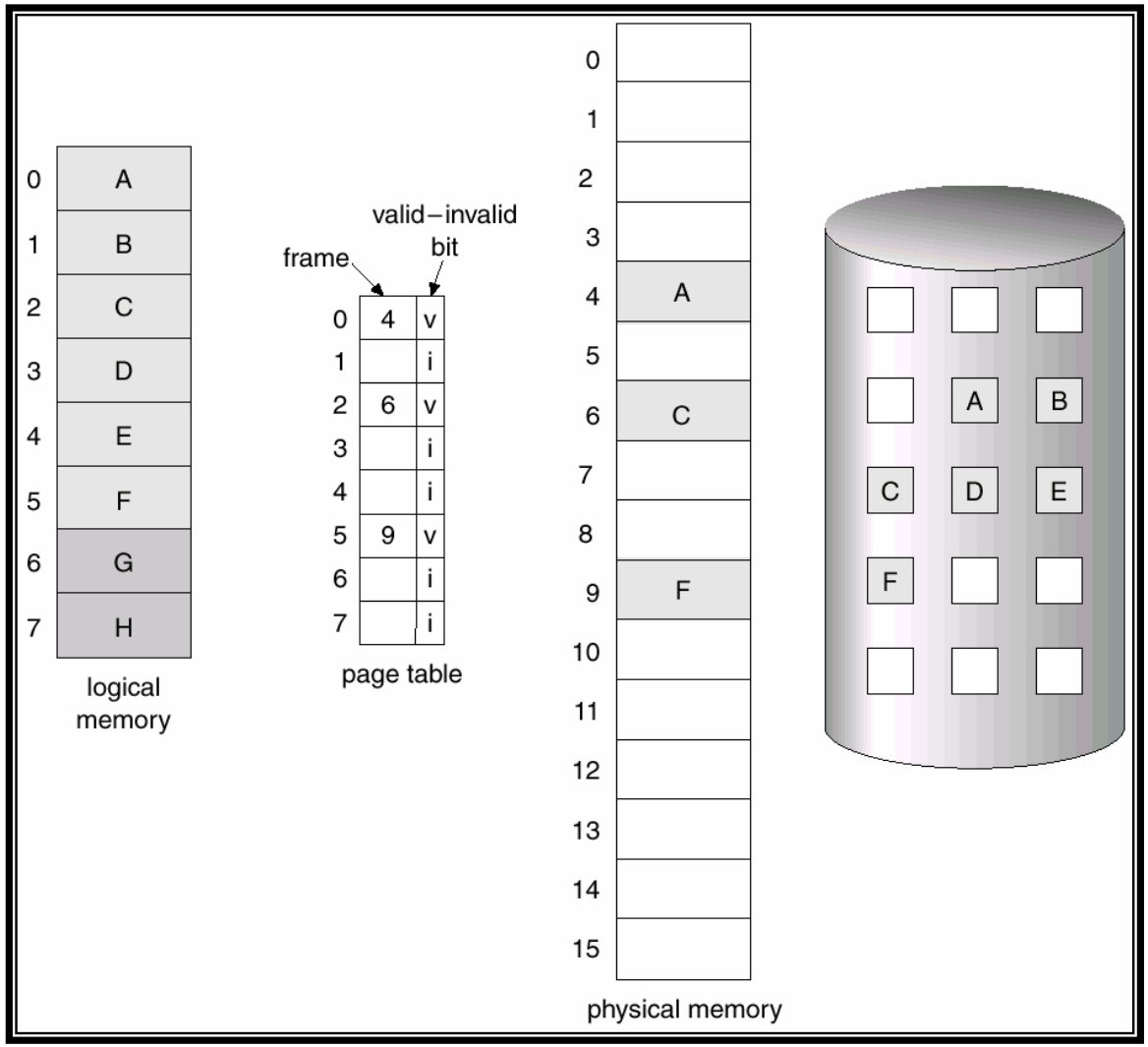
- PTE란?
  - 페이지 테이블의 레코드
- PTE의 각 필드 내용
  - Page base address
    - 해당 페이지에 할당된 프레임의 시작 주소
    - 이를 통해 물리 메모리에 접근할 수 있음
  - Flag bits
    - Accessed bit
      - 페이지에 대한 접근이 있었는지
    - Dirty bit
      - 페이지 내용의 변경이 있었는지
    - Present bit
      - 현재 페이지에 할당된 프레임이 있는지
    - Read/Write bit
      - 읽기 쓰기에 대한 권한 표시
  - 그 외 구성에 따라 여러 가지 내용이 포함될 수 있다.

4.3 Demand Paging

- Demand paging이란
  - 프로세스의 실행을 위한 모든 page를 메모리에 올리지 않고, 필요한 page의 요청이 발생할 때 메모리에 올리는 paging 기법
- Paging 서비스를 통해서 한 프로세스에 필요한 page를 memory와 secondary space 사이를 이동 시킴
- Valid and invalid page
- Demand paging의 장점
  - 실행을 위한 물리 메모리 구성의 시간이 줄어든다.
  - 프로세스의 전체 이미지를 메모리에 올리지 않기 때문에 실제 필요한 전체 물리 메모리의 양을 줄일 수 있다.

4.3 Demand Paging (cont.)

- Demand paging의 단점
- 프로세스가 page를 참조할 때의 가능한 상황
  - 참조하려는 page가 valid한 경우
    - 참조하고자 하는 page가 실제 물리 메모리에 있기 때문에 정상적인 참조가 일어남
  - 참조하려는 page가 invalid한 경우
    - 참조하고자 하는 page가 실제 물리 메모리에 없으므로 이에 대한 처리가 필요
    - Page fault 발생



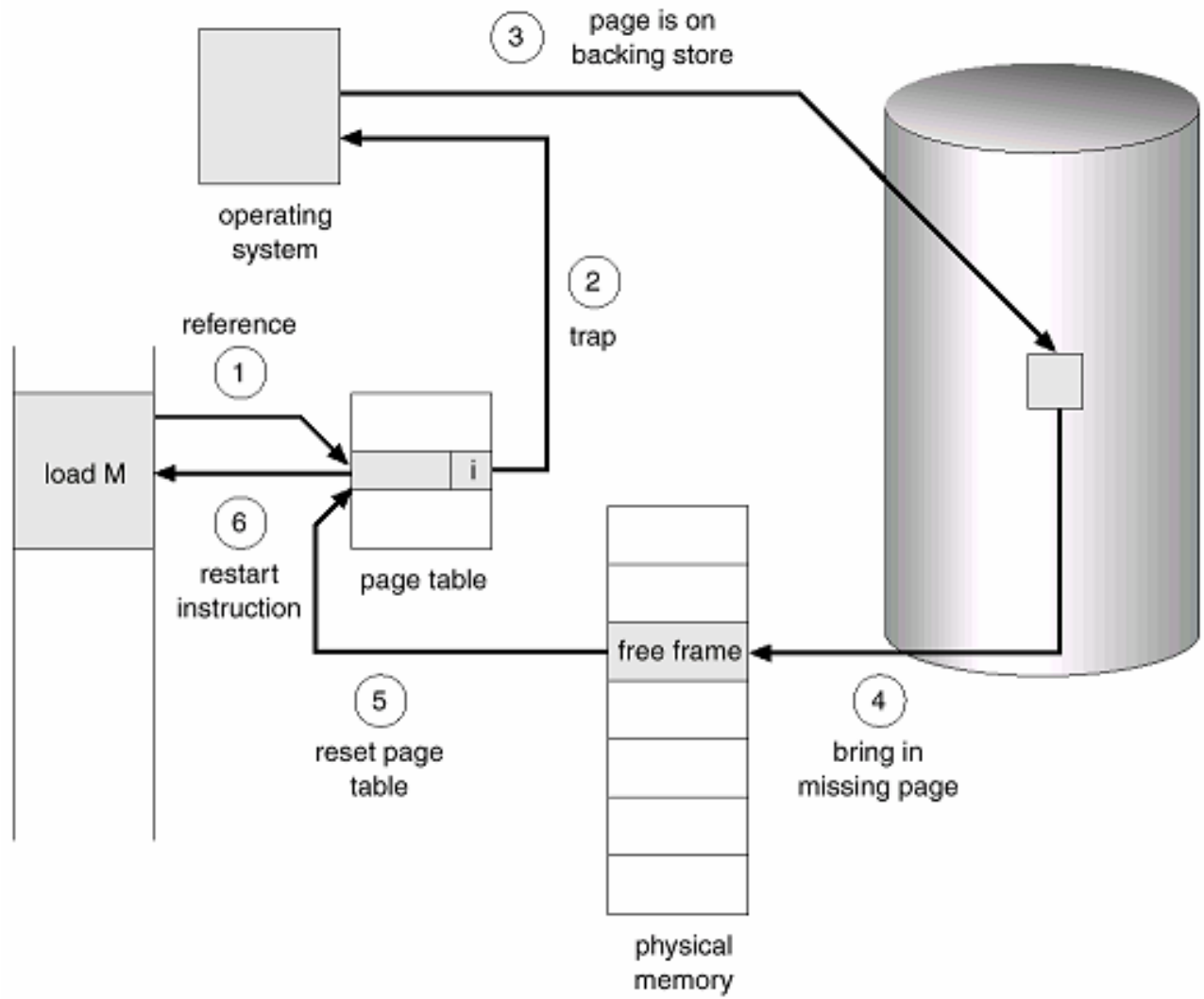
Demand paging 기법을 사용한 page table

4.4 Page Fault

- 프로세스가 페이지를 참조하였을 때 해당 페이지가 할당받은 프레임이 없는 경우
  - 있는 경우 (present bit == true)
    - page base address를 통해 해당 프레임에 접근
  - 없는 경우 (present bit == false)
    - **Page Fault 발생**
      - 프레임을 새로 할당받아야 한다.
    - Page Fault handler 수행

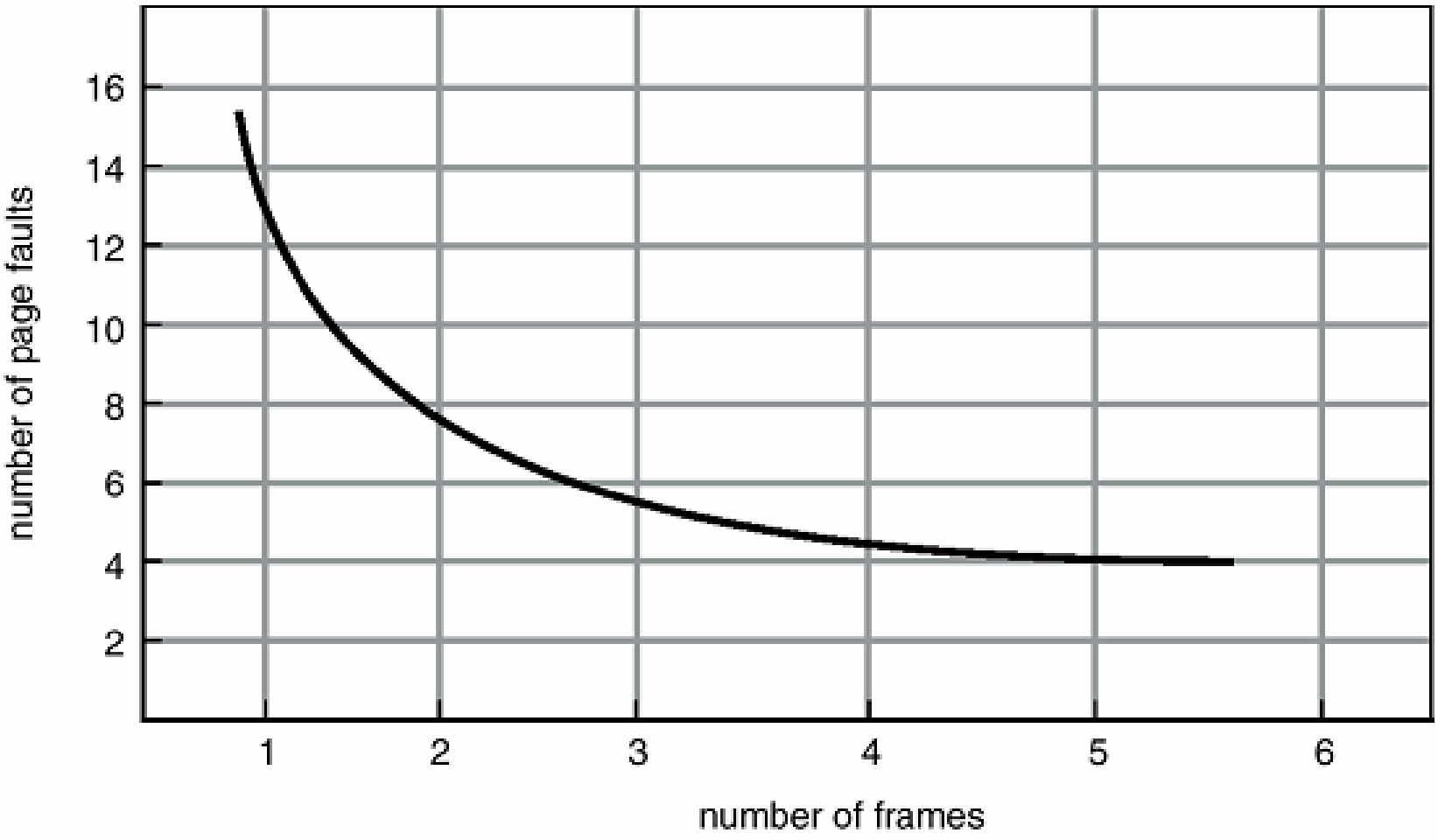
- Page Fault handler가 수행하는 내용
  1. 새로운 프레임을 할당받음
  2. Backing store에서 페이지의 내용을 다시 프레임에 불러들인다.
  3. 페이지 테이블을 재구성한다.
  4. 프로세스의 작업을 재시작한다.

4.4 Page Fault (cont.)



- Page fault handler의 처리 과정

4.4 Page Fault (cont.)



- Page-fault 발생 빈도는 프레임의 개수와 반비례함