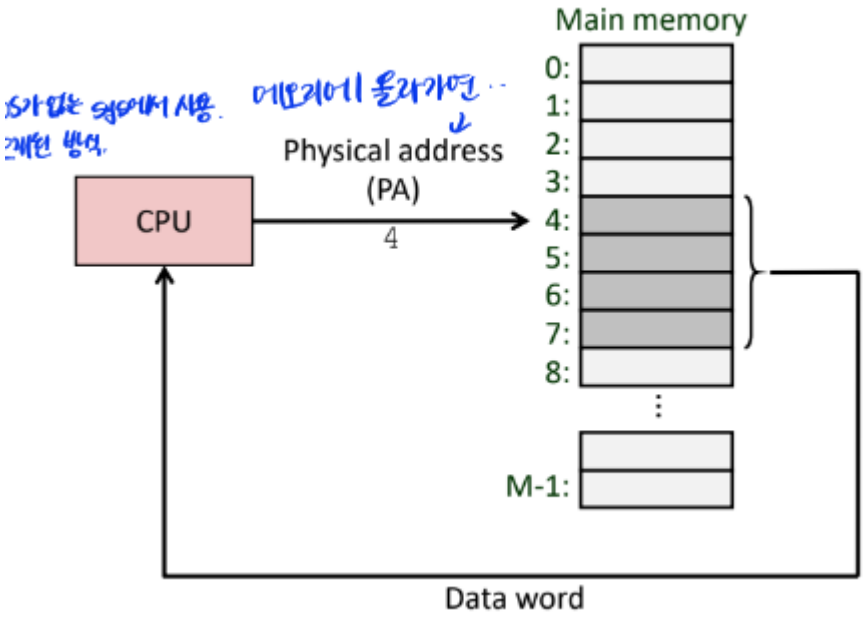
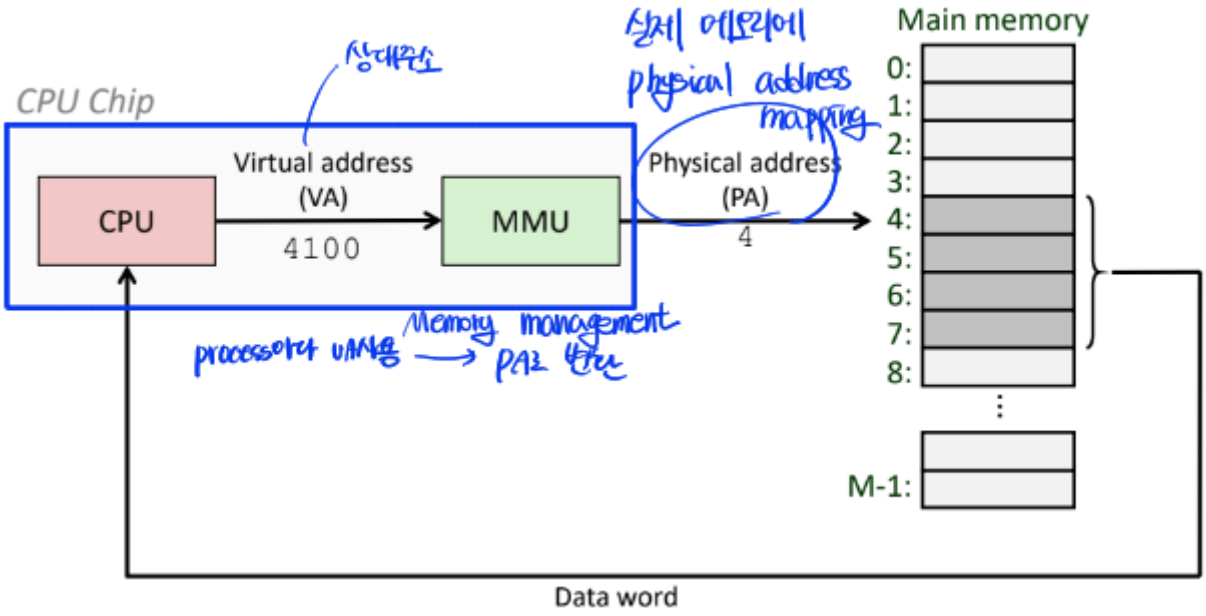
Virtual Memory

System Using Physical Addressing



1. 메모리 구조가 그대로 올라가기 때문에 보안 취약
2. 여러 시스템이 돌아가면 메모리 관리가 어려움
3. os가 없는 시스템에서 사용하던 옛날 버전

System using Virtual Addressing



CPU에서 중간단계인 MMU를 거쳐 VA를 PA로 변환하는 과정.
MMU가 Page Table을 참조해서 VA을 PA로 변환한 후 메모리에 접근한다.

장점

1. PA보다 더 큰 범위로 VA를 사용할 수 있다.
2. PA를 고려하지 않아도 된다.
3. VA - 프로그램이 바라보는 메모리,
   PA - 실제 메모리를 분리해서 바라볼 수 있는 장점이 존재한다.

VM을 사용하는 이유?

1. 메인 메모리를 효율적으로 사용할 수 있다.
   VA로 DRAM을 캐시처럼 사용할 수 있다.

2. 메모리 관리를 단순화할 수 있다.
   linear하여 0번지부터 시작됨

3. 메모리 영역 분할 다른 프로세스의 메모리에서 또다른 프로세스의 메모리에 참조하지 못함

4. Process Layout

    - PA: 모든 주소공간에 접근할 수 있지만, 서로 다른 메모리에서 거부한다.
    - VA: 프로세스 자신의 주소공간만 접근할 수 있다.
      -> VM을 통해 PA의 모순성을 해결함

5. Physical Layout

- 기존 문제점
    - PM에서 커널 영역 접근 불가
    - Device들이 메모리 주소 공간에 나타날 수 있다.
    - System마다 RAM 용량이 다를 수도 있다.
- MMU의 해결
  Virtual Address를 이용해 이러한 제약사항을 숨길 수 있다.
  또한 이상 접근을 차단할 수 있다.

Address Spaces

Architecture별 상이
64bit 아키텍처는 48비트만 가능하다

- Linear address space: 0~...
- Virtual address space: 0~n-1
  가장 큰 범위 사용
  sys, platform별 상이
- Physical address space: 0~m-1
  os, architechture별 상이

Address Locations

현재) byte
과거) word 단위 (한 주소에 한 워드만..)

Memory Management Unit (MMU)

Address Translation 과정

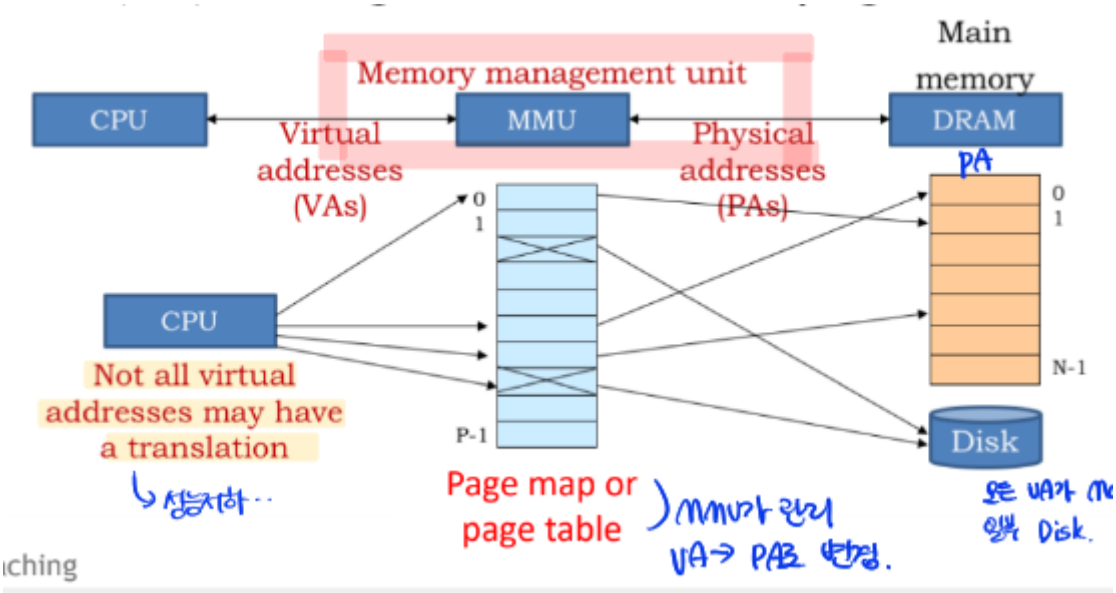
1. MMU가 주소 해석
2. VA -> PA
3. PA로 physical RAM 접근

- Page Table에 VA에 해당하는 PA 올라와있지 않으면 Disk 참조해서 PT 올린 후 재참조
- Physical Memory에 PA 없으면 Disk에서 올리기 (Demand Paging)

VM as a Tool for Caching

Address

- CPU가 VA 이용
- 메인 메모리가 PA 이용



MMU가 VA를 PA로 바꿀 때 Page map or table 이용

- 모든 VA가 Page table에 있는 것은 아니다.
- 모든 PA가 메인 메모리 (DRAM)에 올라가지 못하기 때문에 일부는 Disk 저장

서로다른 프로세스에서 lib같은 공유자원 이용시 PM에 하나만 올림

DRAM Cache Organization

속도 SRAM > DRAM >> Disk(SSD)

- SRAM
  캐시처럼 작은 용량의 고속 메모리
- DRAM
  싸고 SRAM 보다는 느리지만 용량 큼 / 주로 메인 메모리로 사용
- Disk, SSD
  액세스 속도 느리며 용량 큼

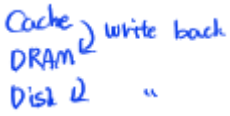
VM 입장에서는 DRAM을 cache로 생각하고 접근

단위

- CPU <-> Cache: word(4byte)
- Main Memory에서 데이터를 주고받는 단위: page(4kb / 4mb)

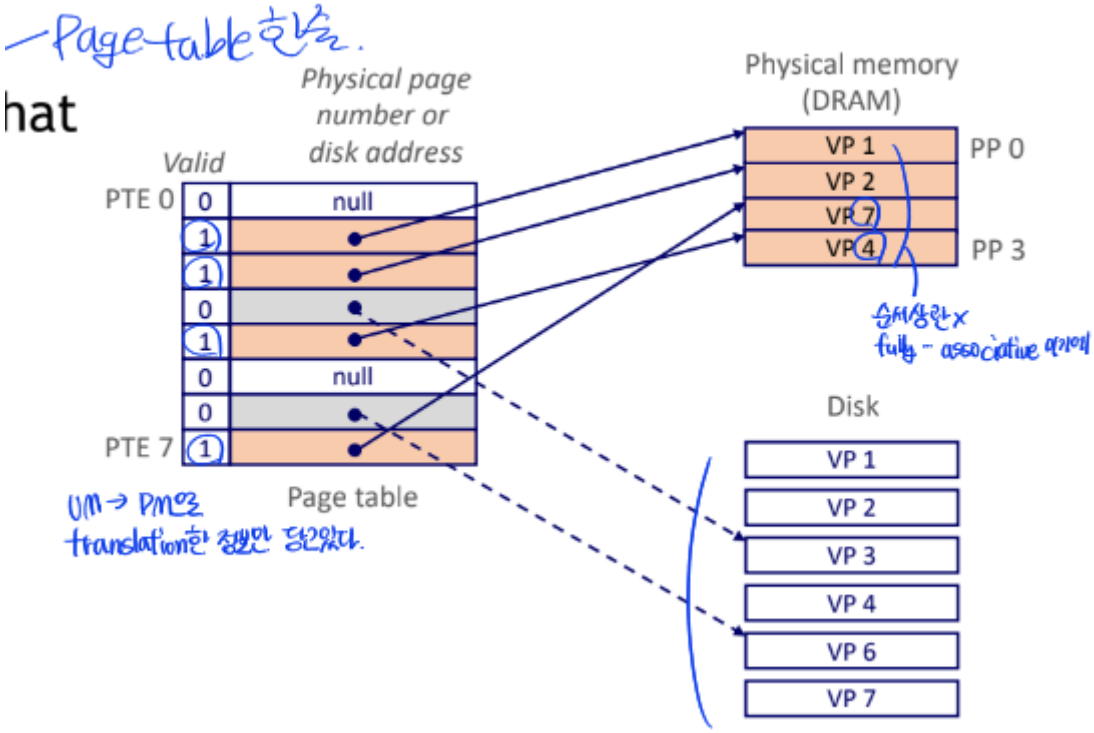
Consequences (결과)

- 큰 page 단위
- Fully Associative - 메모리의 어느 주소든 캐시 메모리의 아무 곳에 올릴 수 있다.
  - mapping 함수, replace 함수같은 비싼 교체 알고리즘이 필요하다. (PM <-> Disk)
- Write-back rather than write-through
  - write back 수정사항이 존재해도 Physical Memory가 디스크에 내려갈 때만 디스크에 쓴다.
    SRAM에 비해 DRAM의 성능 향상을 위한 방법



  - write through
    은행같은 경우, 실시간 동기화가 필요하기 때문에 PM에서 수정사항 발생하여도 Disk까지 바로 반영한다.

Page Table



VA를 PA로 변환한 결과를 담고있는 테이블

- Valid: VPN이 참조하고 있다.
  0이면 해당 PNN이 Disk를 참조하고 있어, page fault 발생 후 PM에 올려 다시 접근한다.
- Physical page number: PNN을 참조하고 있다.

Page Table:
PM에 올라와있음 - valid(page hit) Disk에 저장되어있음 - unvalid (page fault)

page fault handler

disk에 있는 페이지를 올리기 위해 PT에서 valid하고 PM에 올라와있는 하나를 내려야함.
-> 내려온 것이 victim

내린 후 다시 올려서 VA에서부터 다시 접근 retry -> page hit

1. valid:0, page fault
2. victim 골라서 Disk의 pa를 pm에 올리기 -> valid:1
3. victim은 PT의 valid:0이 되고 disk로 내려간다.
4. 접근 retry -> page hit

key point

- demand paging
  miss난 페이지 DRAM(PM)에 올릴 때까지 기다리기
  요청할때만 / 필요할때만 PM에 올라간다.(page fault 시에만 올리기)
    - lazy swapper
      메모리 전체를 변경(swap)하지 않고 특정 페이지만 교체한다
      (victim <-> page fault page)
    - demand paging과 write-back은 독립적
      deman paging: 필요할때 올리기
      write-back: 수정사항 있어도 내려야 할 때만 disk에 쓰기

중요

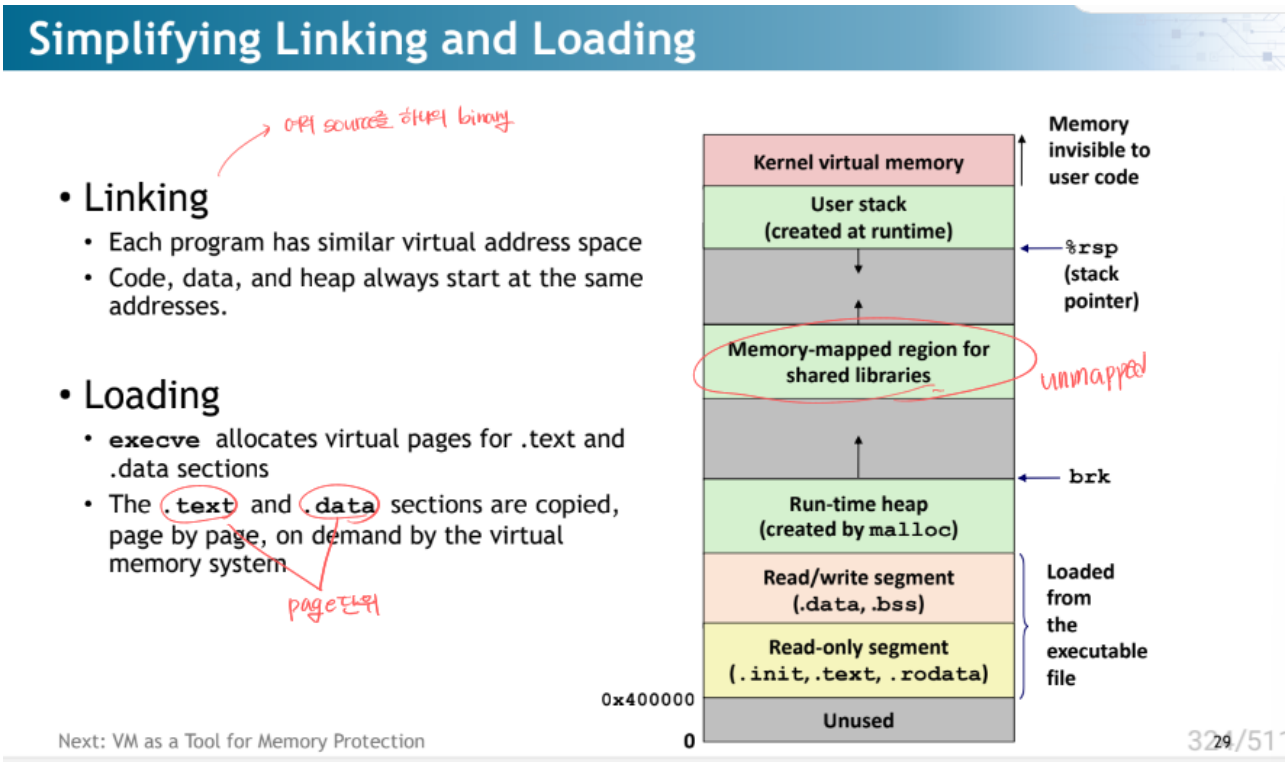
VM이 비효율적인 것처럼 보이지만 locality(지역성)로 동작함

> **Temporal locality**
> 한 번 참조된 데이터는 다시 참조될 가능성이 높다.

- 활성화된 VP를 working set이라고 하며, 시간적 지역성이 좋을수록 더 작은 working set을 가진다.
    - working set size < main memory
      = 메인 메모리에 VP 다 올라가 성능 좋다.
    - sum(working set size) > main memory
      = 메인 메모리에 다 올리지 못하기 때문에 page swap이 많이 일어나며, disk 접근이 잦아진다.

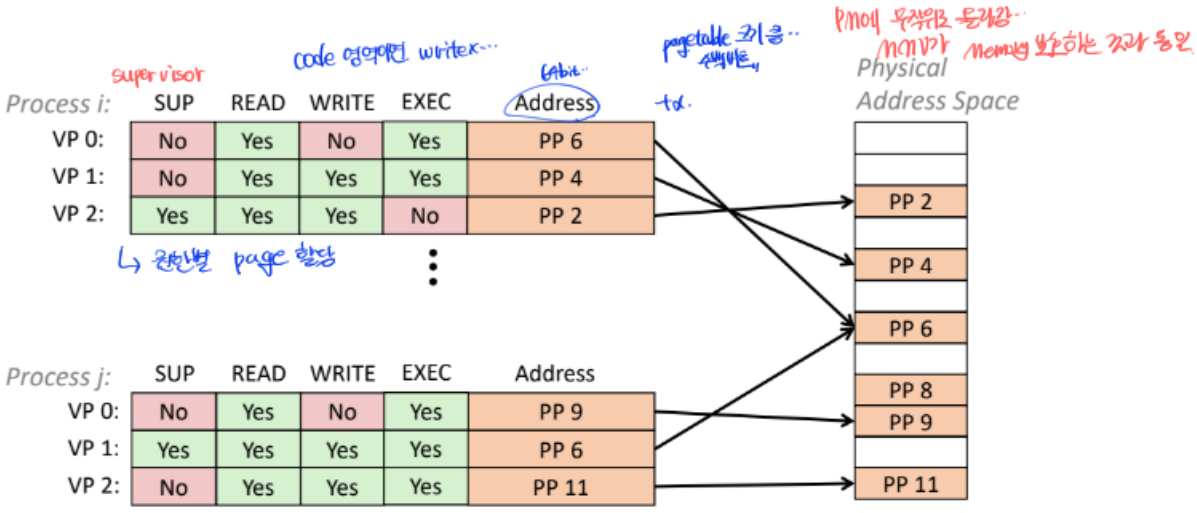
VM as a Tool for Memory Management

- 각 프로세스는 각 virtual address space를 가지고 있다.
- VA는 linear(0부터 순차적)이지만 PA는 그렇지 않다.
- 프로세스 간 shared data(공유자원 (read-only))는 PM에 하나만 저장, 여러 VA가 같은 곳을 참조함
    - shared lib 등 이용시 stack<-> heap 사이 memory-mapped region 영역에 저장된다.



VM as a Tool for Memory Protection

- 한 프로세스 안이더라도 VP마다 권한을 달리 설정할 수 있다.
  Supervisor, read, write, exec가 추가적으로 존재하여 Extended PTE를 지원한다.
  permission bit + PTE -> extended PTE
  - MMU가 각 접근마다 권한을 확인한다.
- VP가 0~n까지 순서대로 있어도 PP에는 랜덤 할당 (fully-associative)하여 메모리를 보호할 수 있다.



Simplify Linkin and Loading

- Linking
  VM이기에 다 Code, data, heap이 전부 같은 주소에서 시작한다.
- Loading
  text(코드), data(초기화된 변수)가 page 단위이기 때문에 demand paging의 영향..

Summary...

- Virtual Memory
    - MMU를 이용해 VA를 PA로 변경한다
    - CPU가 실제 PA를 몰라도 VA만보고 프로그램 구동 가능하다
- Page 단위로 데이터 교환 (4kb) -> CPU <-> cache
- VA 주소 배열과 PA주소 배열은 다르다.. linear / random
- page fault시 unmapped 메모리 맵핑하기 (disk에서 PM으로 올리기)
- page table은 MMU가 VA -> PA로 바꾸는 데 활용된다