Operating System: Introduction to Paging

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

Paging Overview

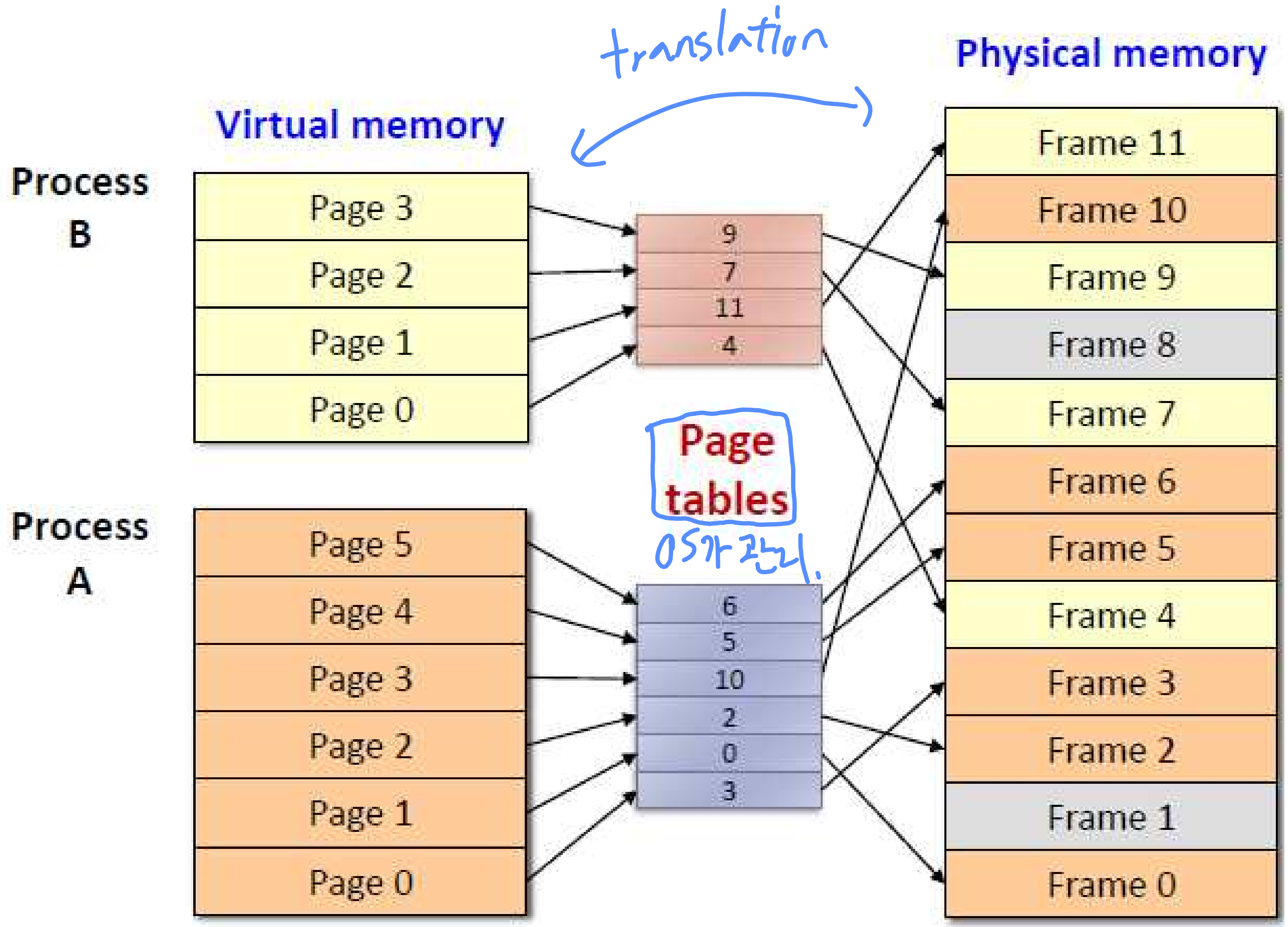
- Two approaches when solving most space-management problem
  - Segmentation: variable size of logical segments (code, stack, heap, etc.)
    - ➢ Fragmentation
    - ➢ Allocation becomes more challenging over time
  - Paging splits up address space into fixed-sized unit called a page

용어의차이

- With paging, physical memory is also split into some number of pages called a page frame

- Page table per process is needed to translate the virtual address to physical address

Paging Overview (Cont.)



Paging

- Allows the physical address space of a process to be noncontiguous 프로세스 주소공간이 연속적일 이유 X
  - Divide virtual memory into blocks of same size (pages)
  - Divide physical memory into fixed-size blocks (frames)
  - Page (or frame) size is power of 2 (typically 512B - 8KB) 대충 이정도 사이즈.

- Eases memory management

기존 방식은 프로그램들을 위한 알맞는 공간을 hole에서 찾아야했으나. 그럴 필요 X

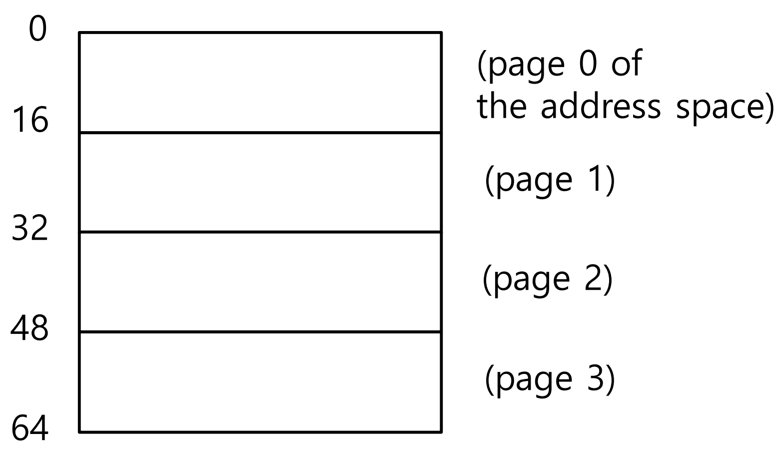
  - OS keeps track of all free frames
  - To run a program of size *n* pages, need to find *n* free frames and load the program
  - Set up a page table to translate virtual to physical addresses
  - No external fragmentation

internal fragmentation

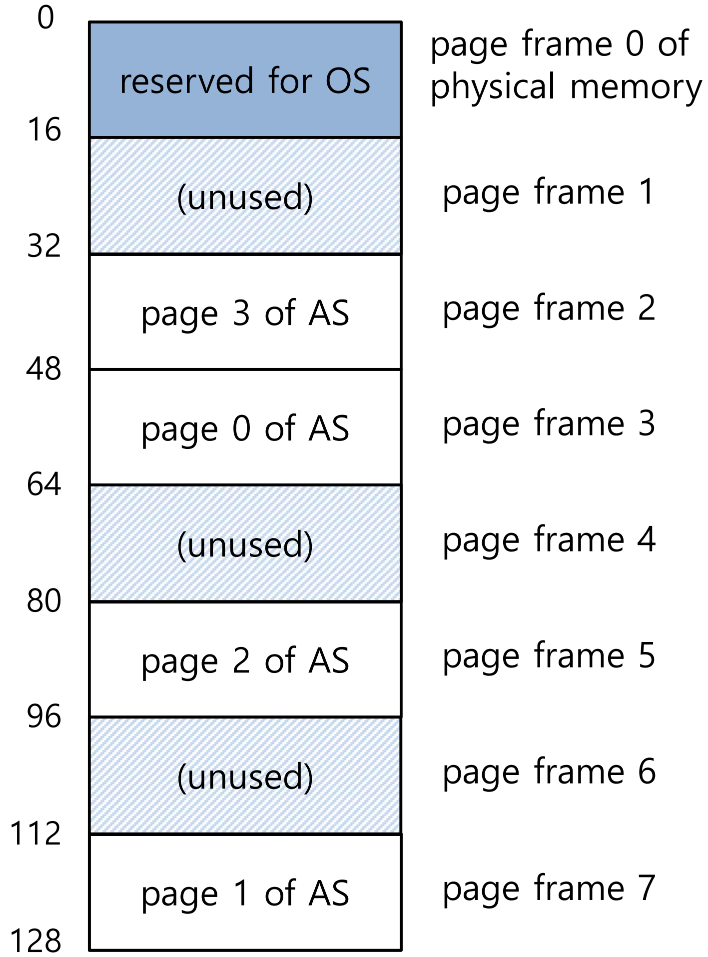
1KB page → page frame 1KB

A Simple Paging Example

- 128-byte physical memory with 16 bytes page frames
- 64-byte address space with 16 bytes pages



A Simple 64-byte Address Space



64-Byte Address Space Placed In Physical Memory

Address Translation

- Two components in the virtual address
  - VPN: virtual page number
  - Offset: offset within the page

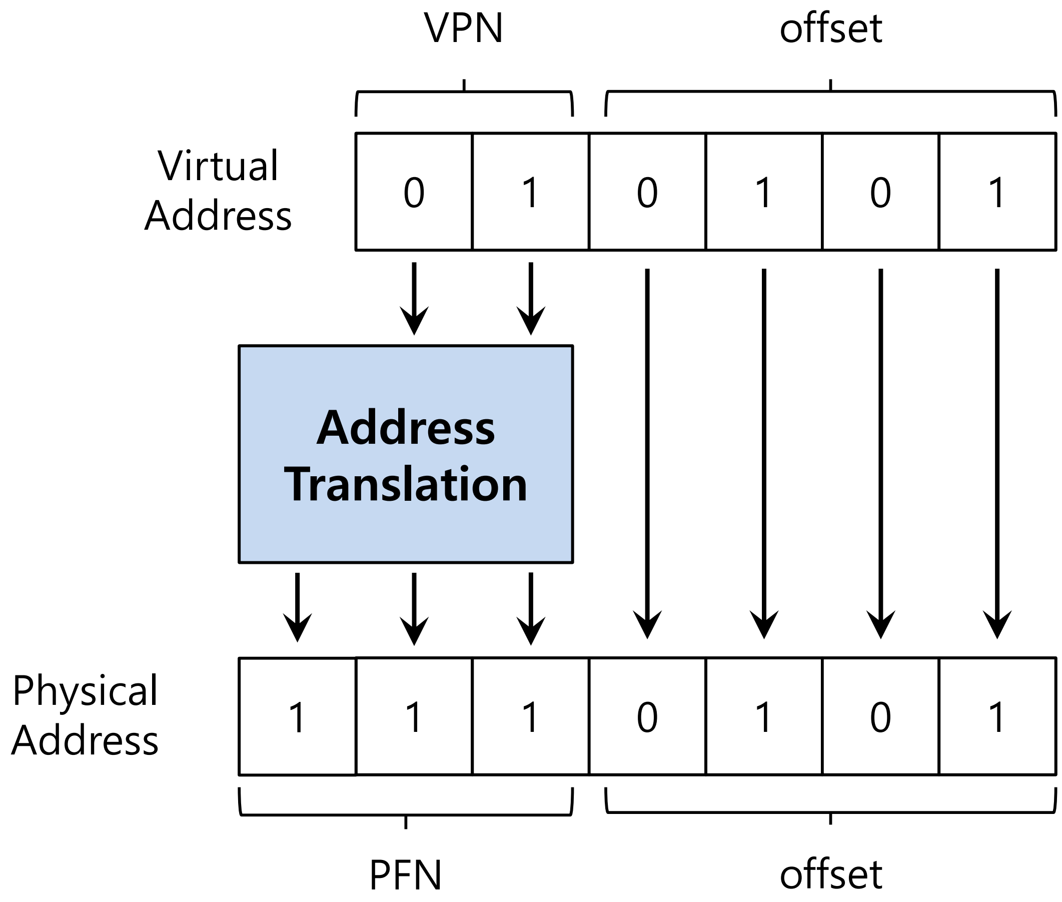
VPN		offset			
Va5	Va4	Va3	Va2	Va1	Va0

- Example: virtual address 21 in 64-byte address space

VPN		offset			
0	1	0	1	0	1

Address Translation (Cont.)

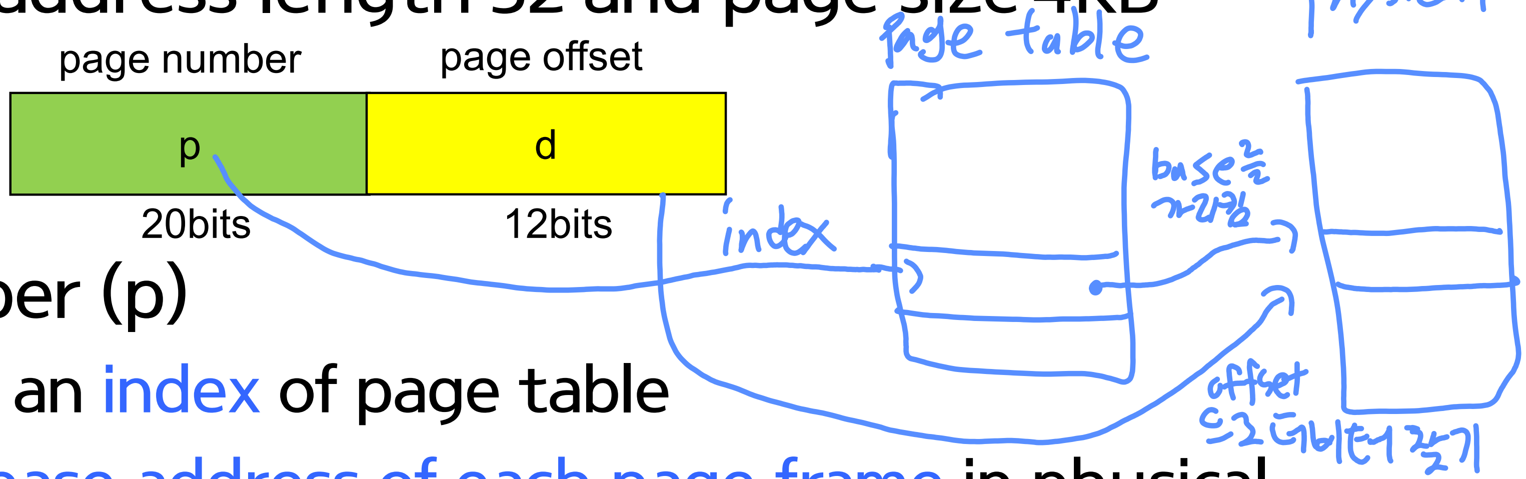
- The virtual address 21 in 64-byte address space



Page
Frame Number

Address Translation (Cont.)

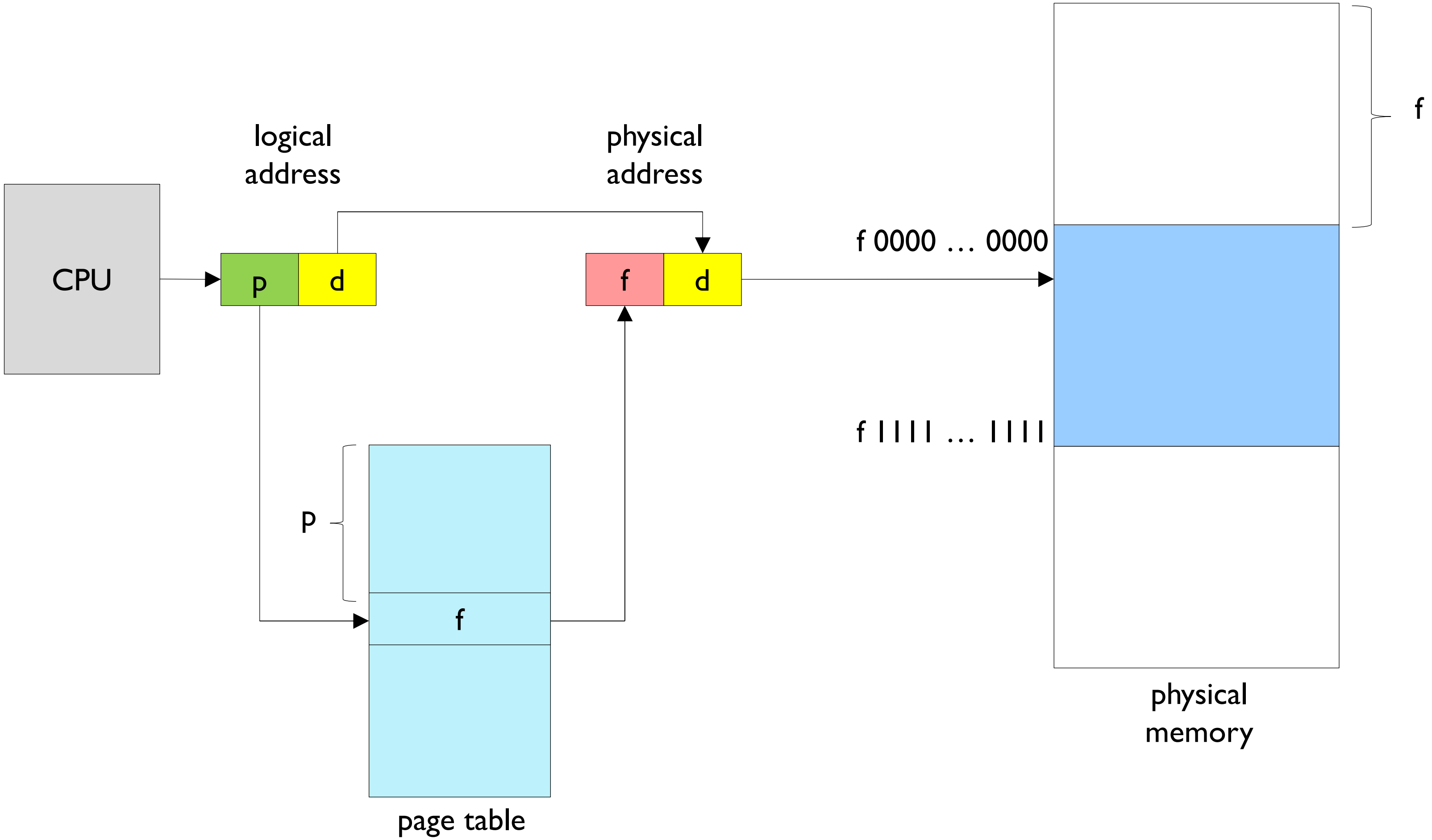
- Another example
  - The logical address generated by CPU is divided into two parts
  - E.g. Given address length 32 and page size 4KB



  - Page number (p)
    - ➢ Is used as an index of page table
    - ➢ Contains base address of each page frame in physical memory
  - Page offset (d)
    - ➢ Is combined with base address to define the physical memory address

Address Translation (Cont.)

- Address Translation Scheme



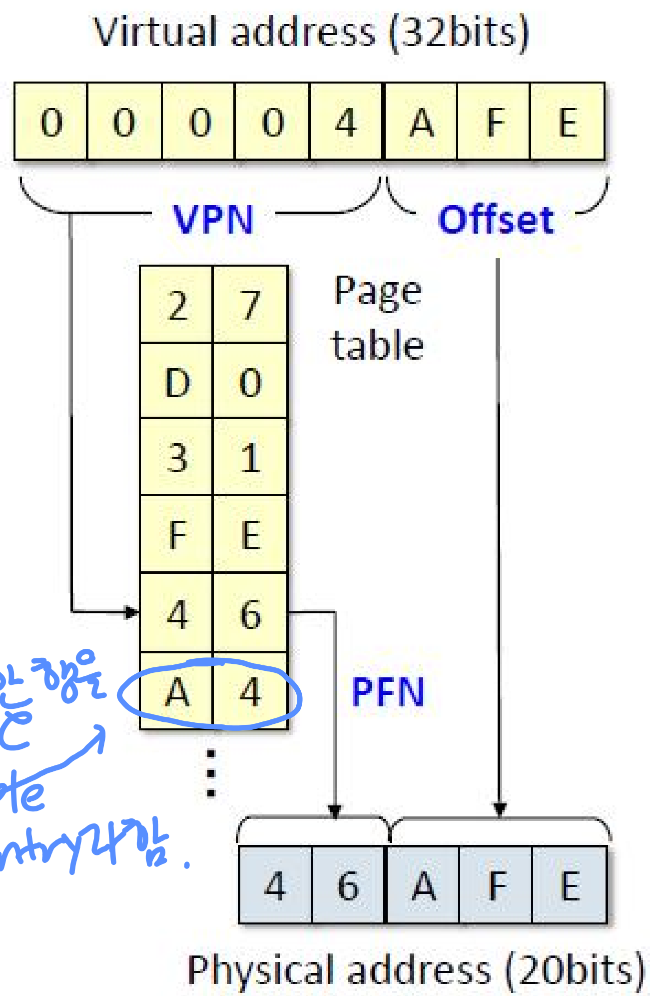
Address Translation (Cont.)

- Example
  - Virtual address: 32 bits
  - Physical address: 20 bits
  - Page size: 4KB

  - Offset: 12 bits
  - VPN: 20 bits
  - Page table entries: 2^{20}
  - Page table size?

=1MB

한 행을 page table entry라함.



Paging: Spatial Overhead

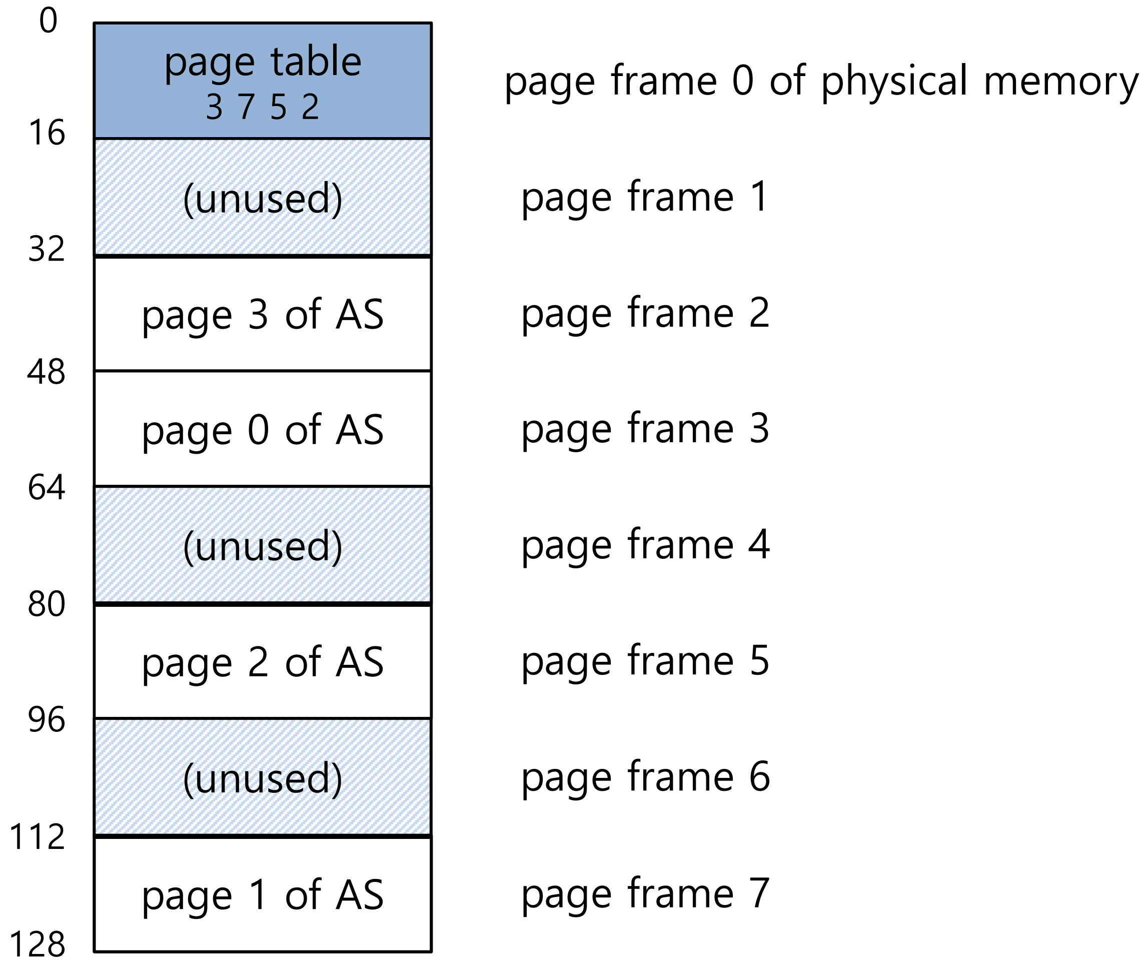
- Where Are Page Tables Stored?
  - Page tables for each process are stored in memory

각각의 프로세스마다 생성되는 건데 너무 커지면 X.

- Page tables can get awfully large
  - 32-bit address space with 4-KB pages, 20 bits for VPN
    - $4MB = 2^{20}\ entries \ * 4\ Bytes\ per\ page\ table\ entry$

Page Table in Kernel Physical Memory

- Page tables for each process are stored in memory



Physical Memory

What Is In The Page Table?

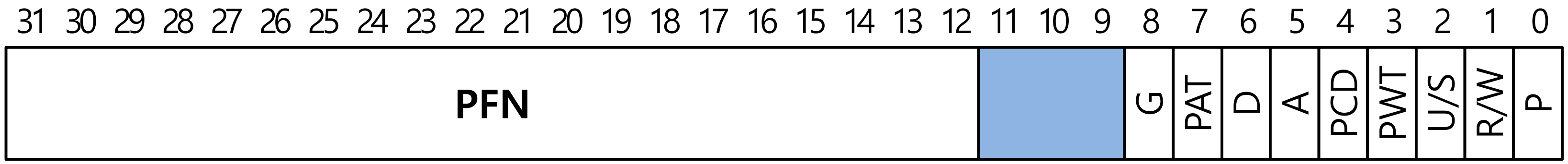
- The page table is just a data structure that is used to map the virtual address to physical address
  - Simplest form: a linear page table, an array

- The OS indexes the array by VPN, and looks up the page-table entry

Common Flags of Page Table Entry

- Valid Bit
  - Indicates whether the particular translation is valid
  - It is checked each time a virtual address is used
- Protection Bit
  - Indicates whether the page could be read from, written to, or executed from
- Present Bit
  - Indicates whether this page is in physical memory or on disk(swapped out)
- Dirty Bit
  - Indicates whether the page has been modified since it was brought into memory
- Reference Bit (Accessed Bit)
  - Indicates that a page has been accessed

Example: x86 Page Table Entry



An x86 Page Table Entry(PTE)

- P: present
- R/W: read/write bit
- U/S: supervisor
- A: accessed bit
- D: dirty bit
- PFN: the page frame number

Paging: Temporal Overhead

- To find a location of the desired PTE, the starting location of the page table is needed

- For every memory reference, paging requires the OS to perform one extra memory reference

page table 또한 메모리에 있고
이를 참조하는데 추가적인 오버헤드가 발생.

Accessing Memory With Paging

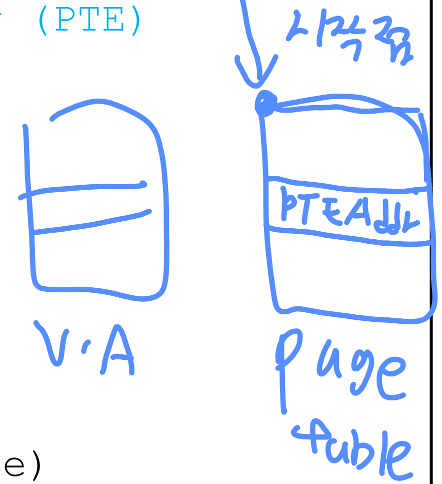
page table base register

```
// Extract the VPN from the virtual address
VPN = (VirtualAddress & VPN_MASK) >> SHIFT

// Form the address of the page-table entry (PTE)
PTEAddr = PTBR + (VPN * sizeof(PTE))

// Fetch the PTE
PTE = AccessMemory(PTEAddr)

// Check if process can access the page
if (PTE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
else if (CanAccess(PTE.ProtectBits) == False)
        RaiseException(PROTECTION_FAULT)
else
        // Access is OK: form physical address and fetch it
        offset = VirtualAddress & OFFSET_MASK
        PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
        Register = AccessMemory(PhysAddr)
```



page Frame Number.

Demand Paging

- Divide memory and image into fixed size, respectively
  - page frame and page
  - Size is power of 2, between 512B and 8KB
- Entire program image resides on disk
- When the program starts, just load the first page only
- The rest of pages are loaded in memory on-demand
- A particular page X of the program can be either
  - already loaded in memory page frame Y, or
  - never been loaded before, it is in disk
- Pages can be placed anywhere in memory
- Whenever CPU presents an address, MMU looks up page table
  - For translating logical address to physical address

disk 에 일단두고
첫 페이지만
메모리에 로드.

memory management Unit

Paging: Pros

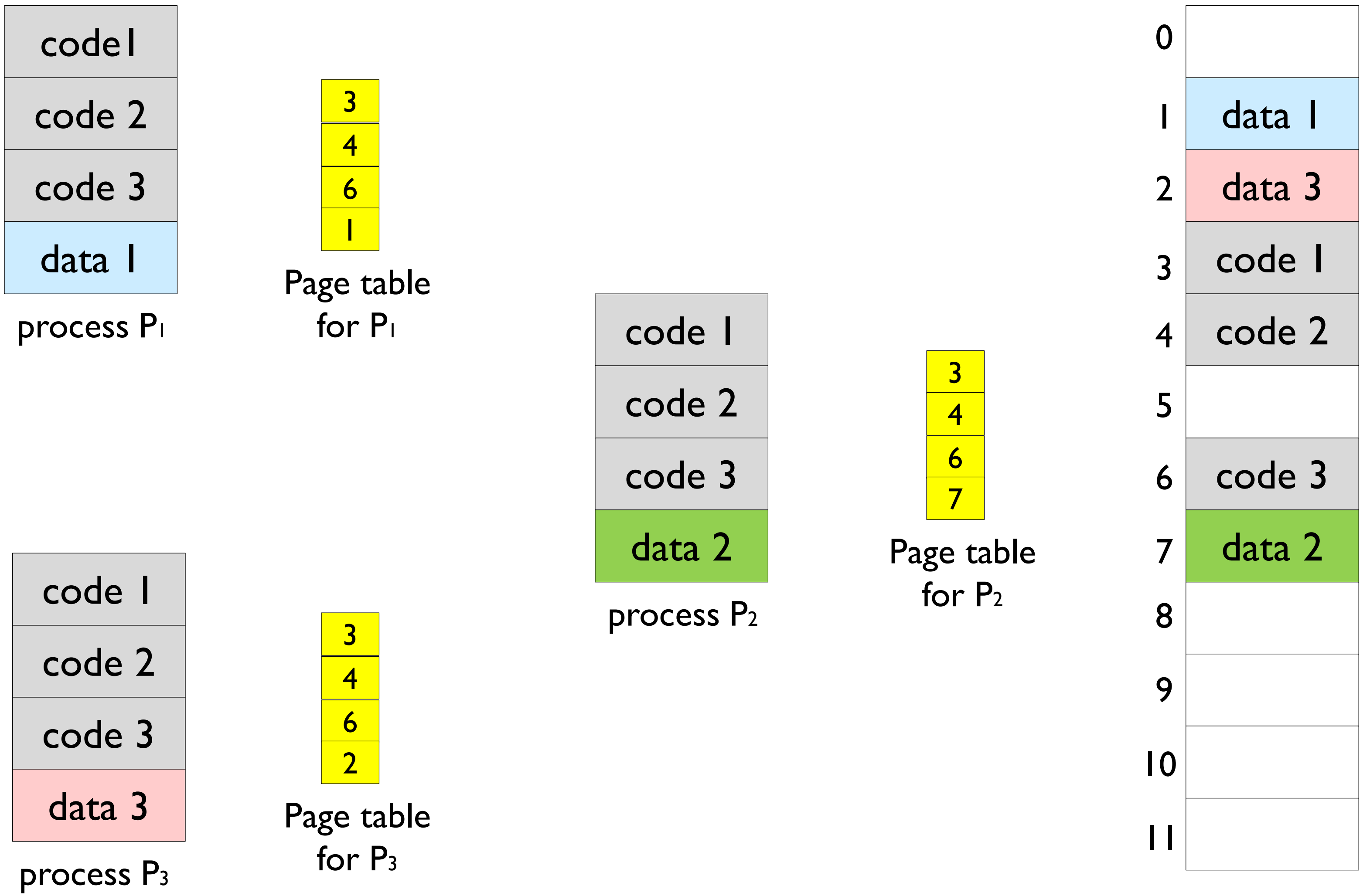
- No external fragmentation
- Fast to allocate and free
  - A list or bitmap for free page frames
  - Allocation: no need to find contiguous free space
  - Free: no need to coalesce with adjacent free space
- Easy to "page out" portions of memory to disk

  필요없어서 다시 디스크로 보내는거
  - Page size is chosen to be a multiple of disk block sizes
  - Use valid bit to detect reference to "paged-out" pages
  - Can run process when some pages are on disk
- Easy to protect and share pages

Paging: Pros (Cont.)

다시보기

- Shared pages



Paging: Cons

- Internal fragmentation
  - Wasted memory grows with larger pages
- Memory reference overhead (temporal overhead)
  - Doubles the number memory references per instruction
  - Solution: get hardware support (TLBs)
- Storage needed for page tables (spatial overhead)
  - Needs one PTE for each page in virtual address space
    - 32-bit address space with 4KB page size: 2^{20} PTEs
    - 4 bytes/PTE: 4MB per page table
    - 100 processes in the system: total 400MB of page tables
  - Solution: store valid PTEs only