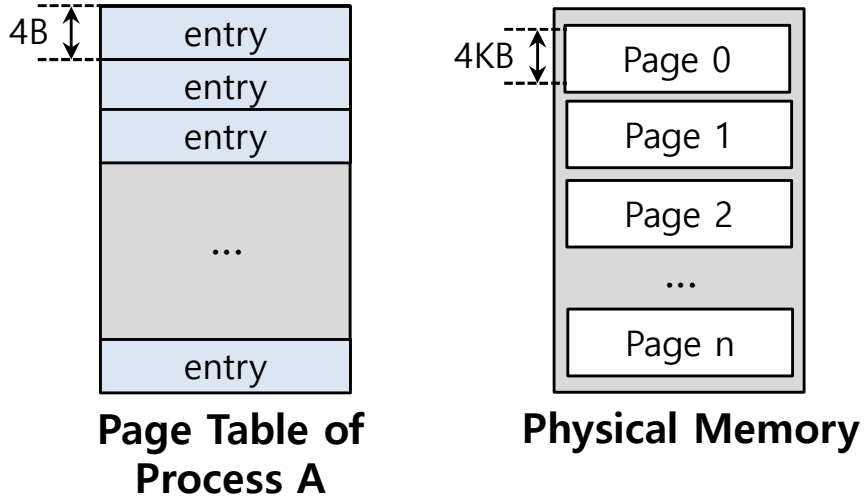
Operating System: Advanced Page Tables

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

Paging: Linear Tables

- We usually have one page table for every process in the system
  - Assume that 32-bit address space with 4KB pages and 4-byte page-table entry

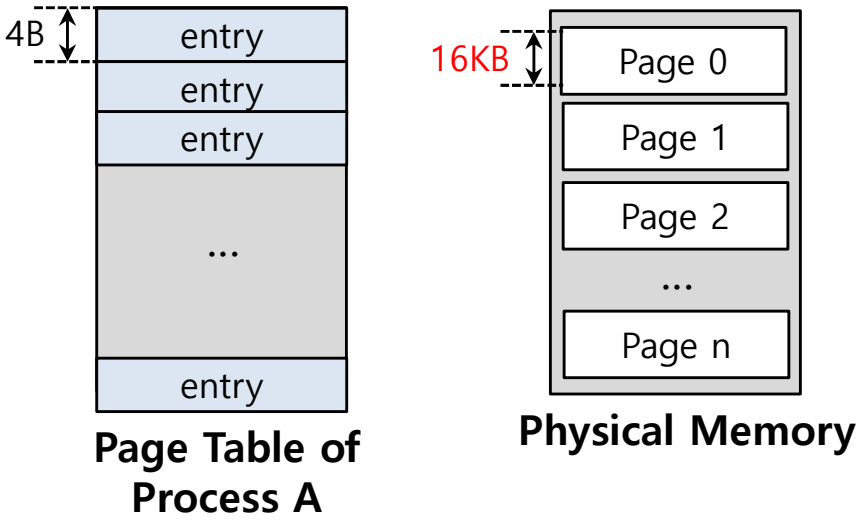


$$\text{Page table size} = \frac{2^{32}}{2^{12}} * 4Byte = 4M\ Byte$$

Page table are too big and thus consume too much memory

Paging: Smaller Tables

- Page table are too big and thus consume too much memory
  - Assume that 32-bit address space with 16KB pages and 4-byte page-table entry



page 크기가 커지면 page 갯수가 줄어드니까 가져갈 갯수↓ → entry↓ → page table size↓

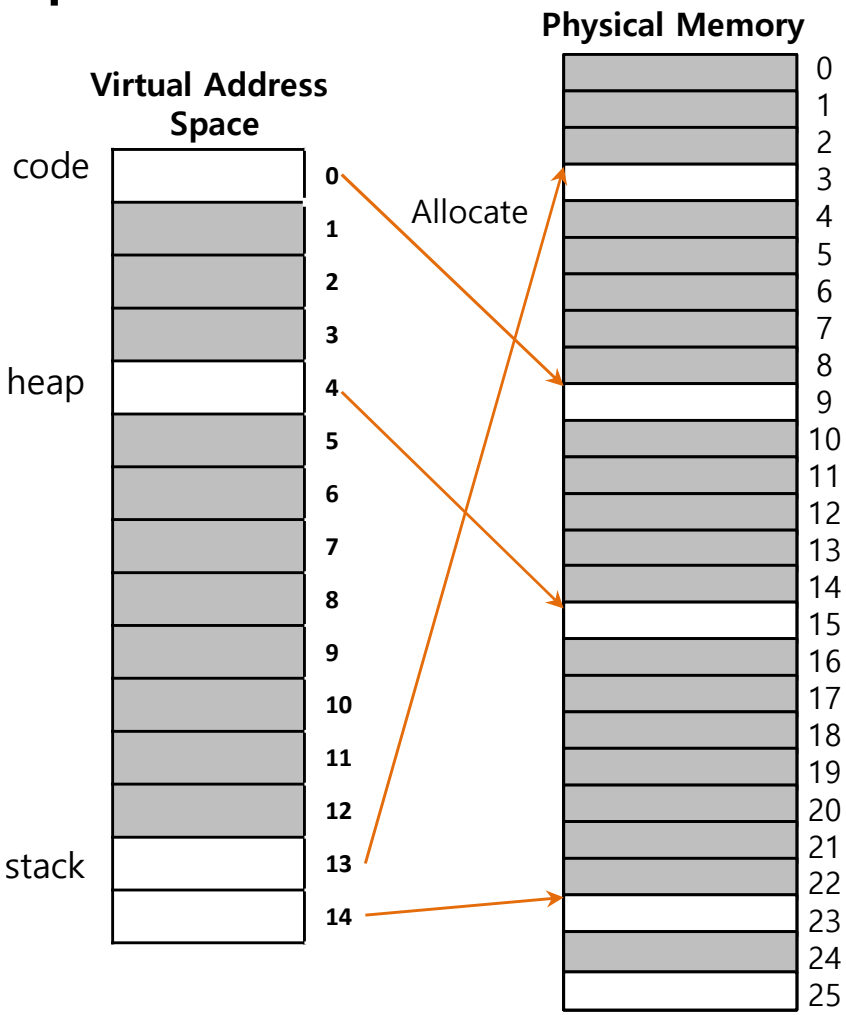
$\frac{2^{32}}{2^{16}} * 4 = 1MB$ per page table

Big pages lead to internal fragmentation

page가 커지니 목적으로 인해 할당된 영역은

Problem

- Single page table for the entries address space of process



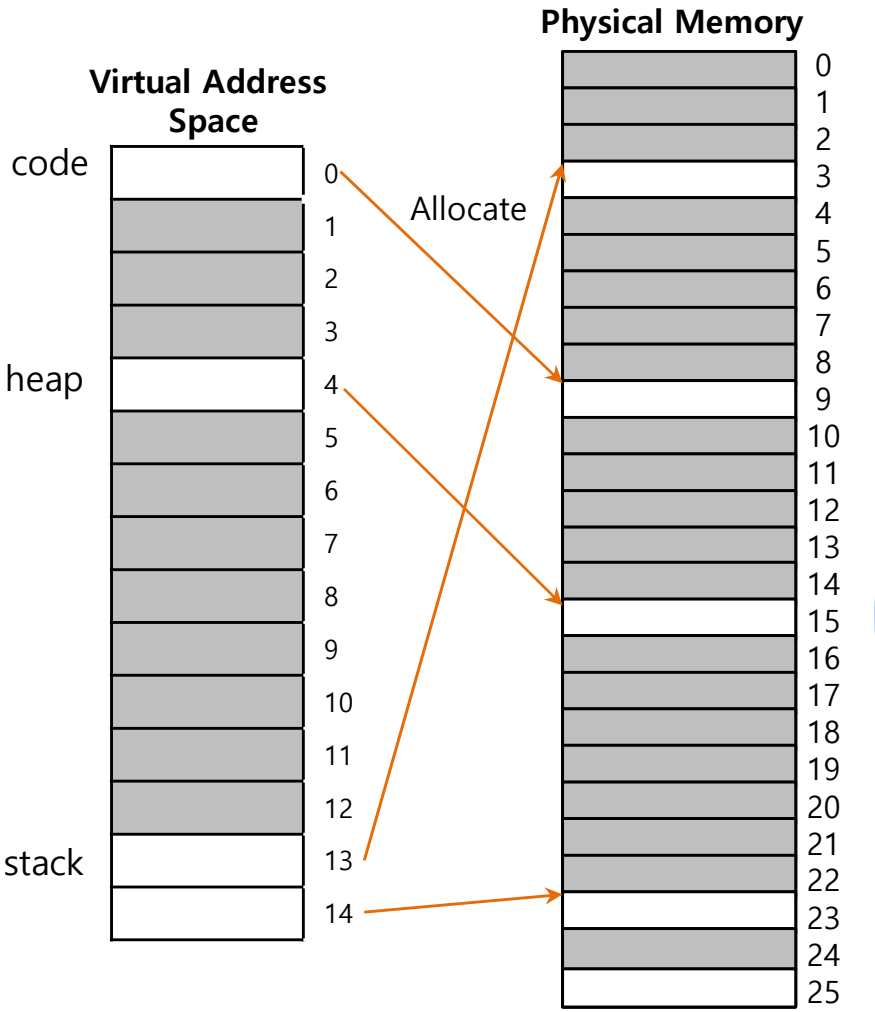
A 16KB Address Space with 1KB Pages

PFN	valid	prot	present	dirty
9	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...	...	...	...	...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

Problem

- Most of the page table is unused, full of invalid entries



A 16KB Address Space with 1KB Pages

PFN	valid	prot	present	dirty
9	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...	...	...	...	...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

데이터이 필요한 2602

많이합고 있음.

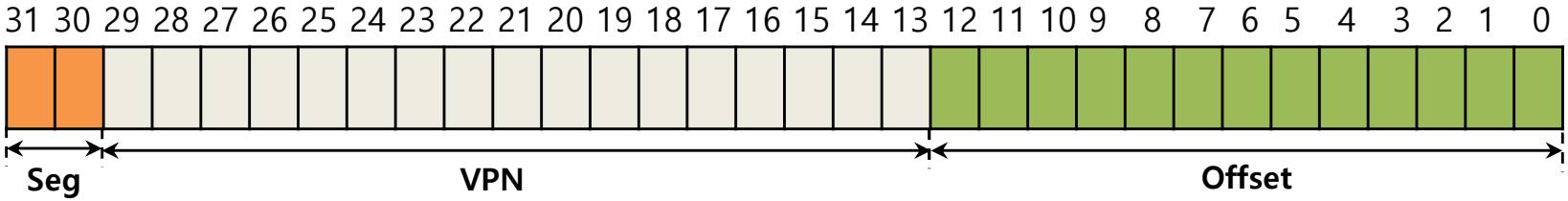
Hybrid Approach: Paging and Segments

두 가지 장점 결합

- In order to reduce the memory overhead of page tables
  - Segmentation:
    - Divide virtual address space into segments
    - Each segment can have variable length
  - Paging:
    - Divide each segment into fixed-sized pages
    - Each segment has a page table
    - Each segment tracks base (physical address) and limit of the page table for that segment
  - Using base not to point to the segment itself but rather to hold the physical address of the page table of that segment
  - The bounds register is used to indicate the end of the page table

Simple Example of Hybrid Approach

- Each process has three page tables associated with it
  - When process is running, the base register for each of these segments contains the physical address of a linear page table for that segment



32-bit Virtual address space with 4KB pages

Seg value	Content
00	unused segment
01	code
10	heap
11	stack

TLB miss on Hybrid Approach

- The hardware get to **physical address** from **page table**
  - The hardware uses the segment bits (SN) to determine which base and bounds pair to use
  - The hardware then takes the physical address therein and combines it with the VPN as follows to form the address of the page table entry (PTE)

```
01:     SN = (VirtualAddress & SEG_MASK) >> SN_SHIFT
02:     VPN = (VirtualAddress & VPN_MASK) >> VPN_SHIFT
03:     AddressOfPTE = Base[SN] + (VPN * sizeof(PTE))
```

segment 특징

paging 특징

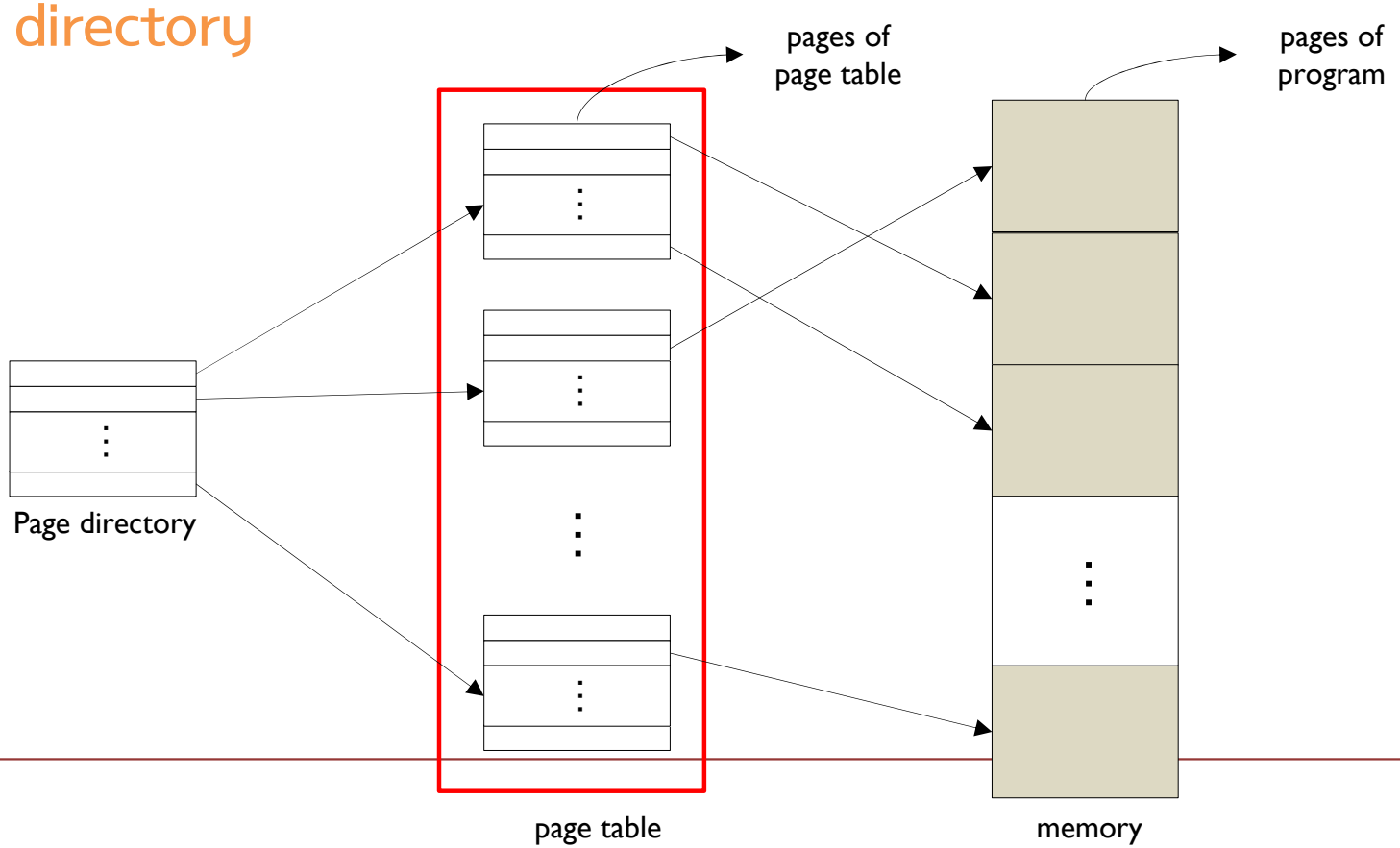
Problem of Hybrid Approach

- Hybrid Approach is not without problems
  - If we have a large but sparsely-used heap, we can still end up with a lot of page table waste
  - Causing external fragmentation to arise again

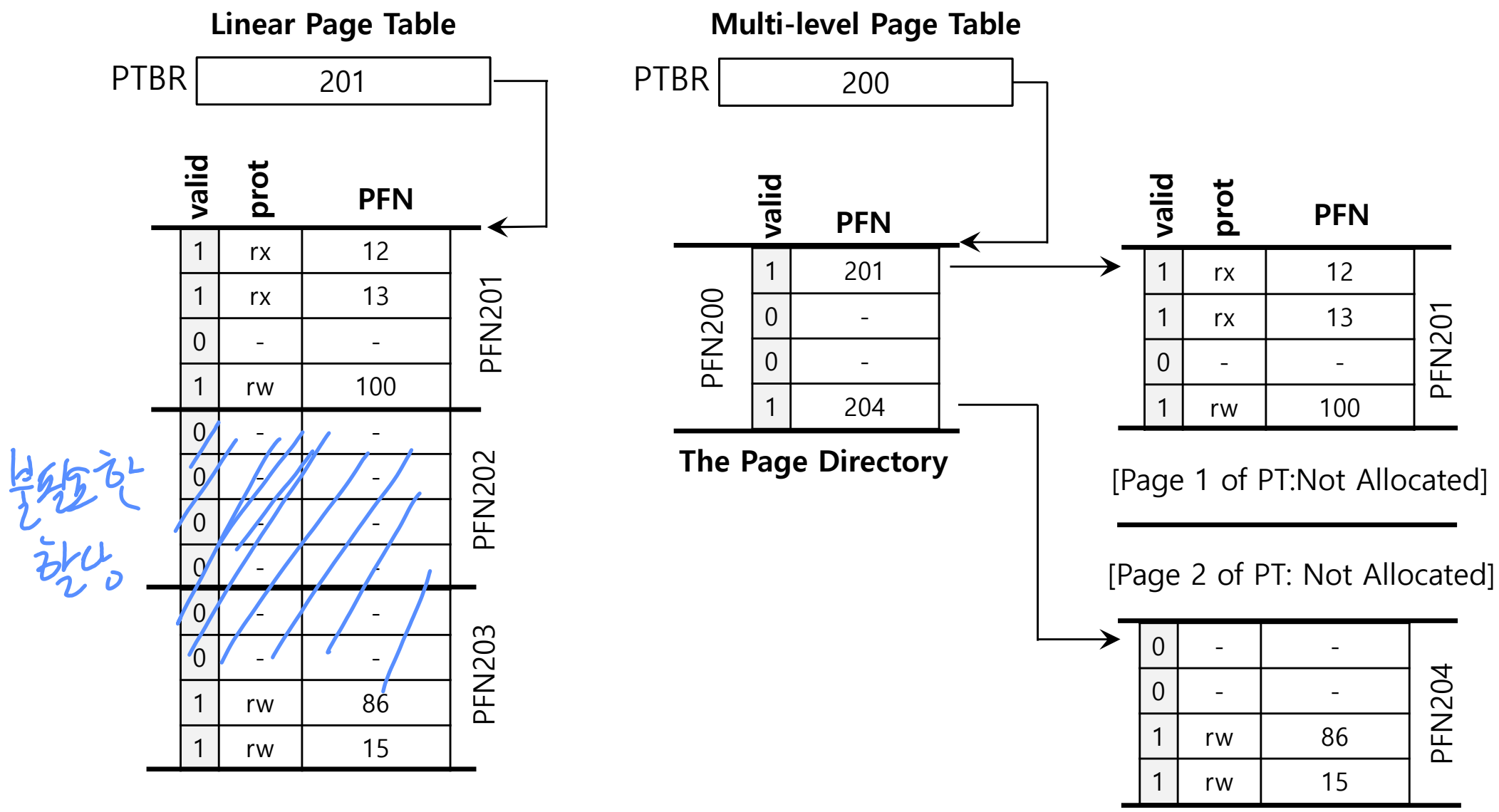
paging에서는 나타나지 않는 특징인 hole이 다시 생길 가능

Multi-level Page Tables

- Turns the linear page table into something like a tree
  - Chop up the page table into page-sized units
  - If an entire page of page-table entries is invalid, don't allocate that page of the page table at all
  - To track whether a page of the page table is valid, use page directory



Multi-level Page Tables: Page directory



Linear (Left) And Multi-Level (Right) Page Tables

Multi-level Page Tables: Page directory entries

- The page directory contains one entry per page of the page table
  - It consists of a number of page directory entries (PDE)

- PDE has a valid bit and page frame number(PFN)

Multi-level Page Tables: Advantage & Disadvantage

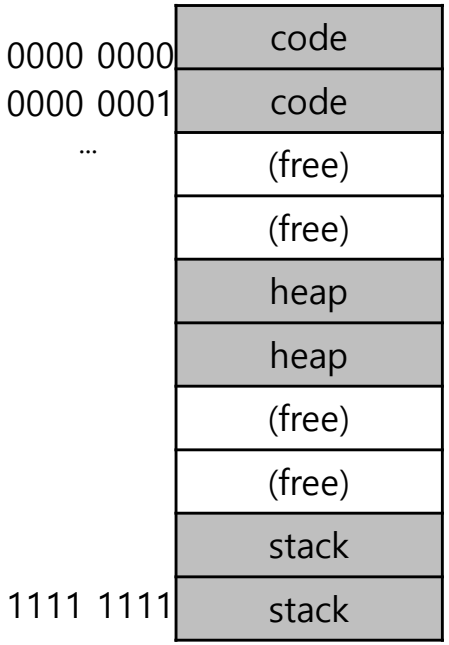
- Advantage
  - Only allocates page-table space in proportion to the amount of address space you are using
  - The OS can grab the next free page when it needs to allocate or grow a page table

OS가 P.T를 paging하니까 관리가 더 용이

- Disadvantage
  - Multi-level table is a small example of a time-space trade-off
  - Complexity

A Detailed Multi-Level Example

- To understand the idea behind multi-level page tables better, let's do an example

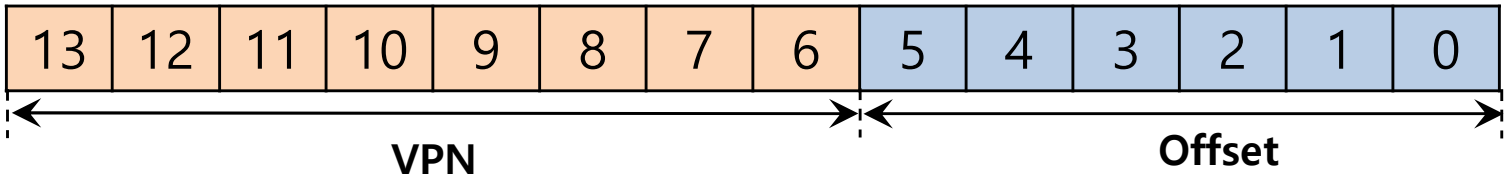


Flag	Detail
Address space	16 KB
Page size	64 byte
Virtual address	14 bit
VPN	8 bit
Offset	6 bit
Page table entry	$2^8(256)$

256개 page

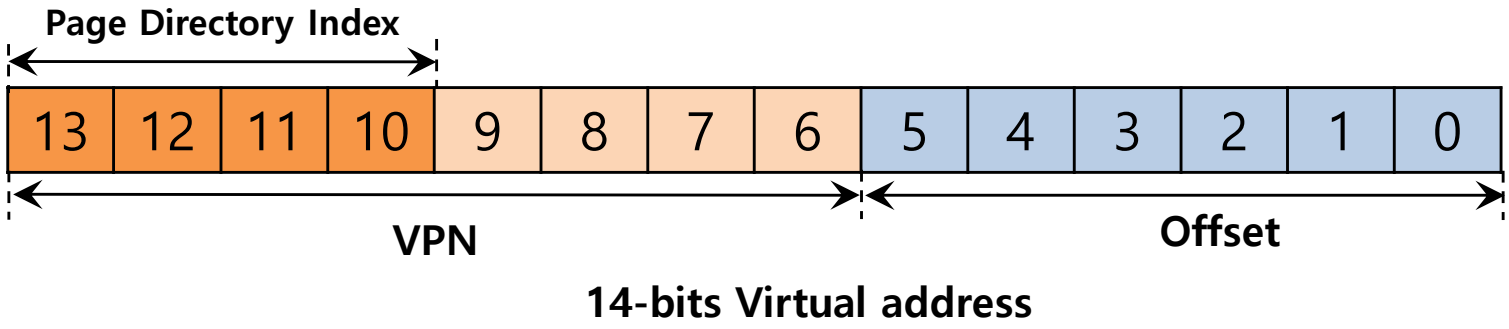
256개 존재

A 16-KB Address Space With 64-byte Pages



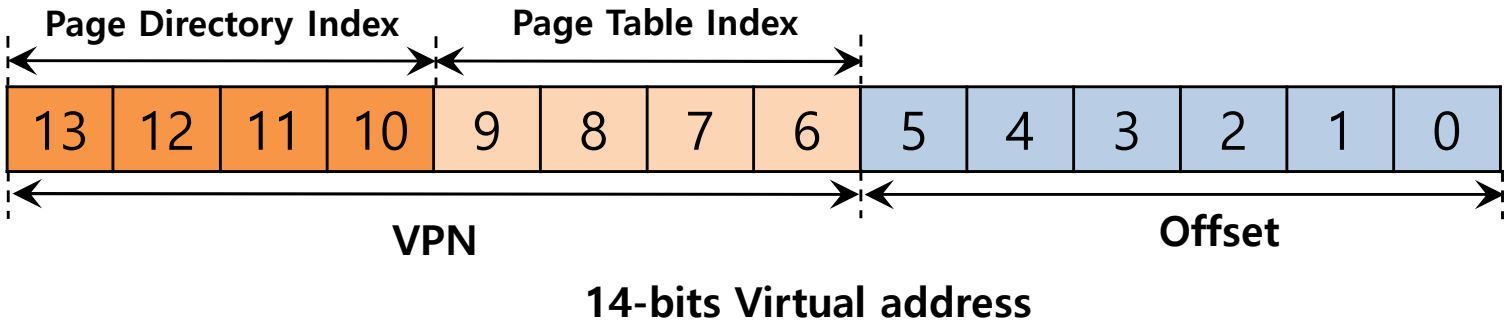
A Detailed Multi-Level Example: Page Directory Idx

- The page directory needs one entry per page of the page table
  - it has 16 entries
- The page-directory entry is invalid → Raise an exception (The access is invalid)



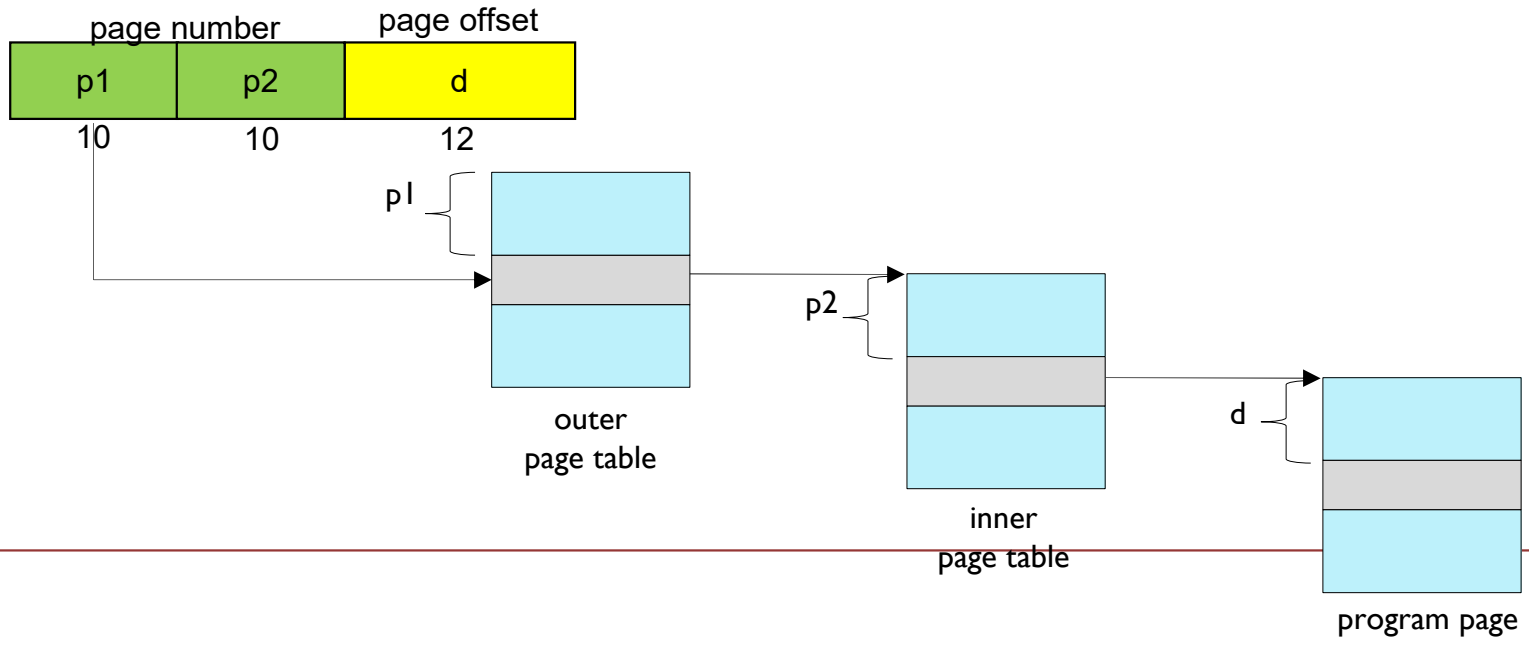
A Detailed Multi-Level Example: Page Table Idx

- The PDE is valid, we have more work to do
  - To fetch the page table entry(PTE) from the page of the page table pointed to by this page-directory entry
- This page-table index can then be used to index into the page table itself



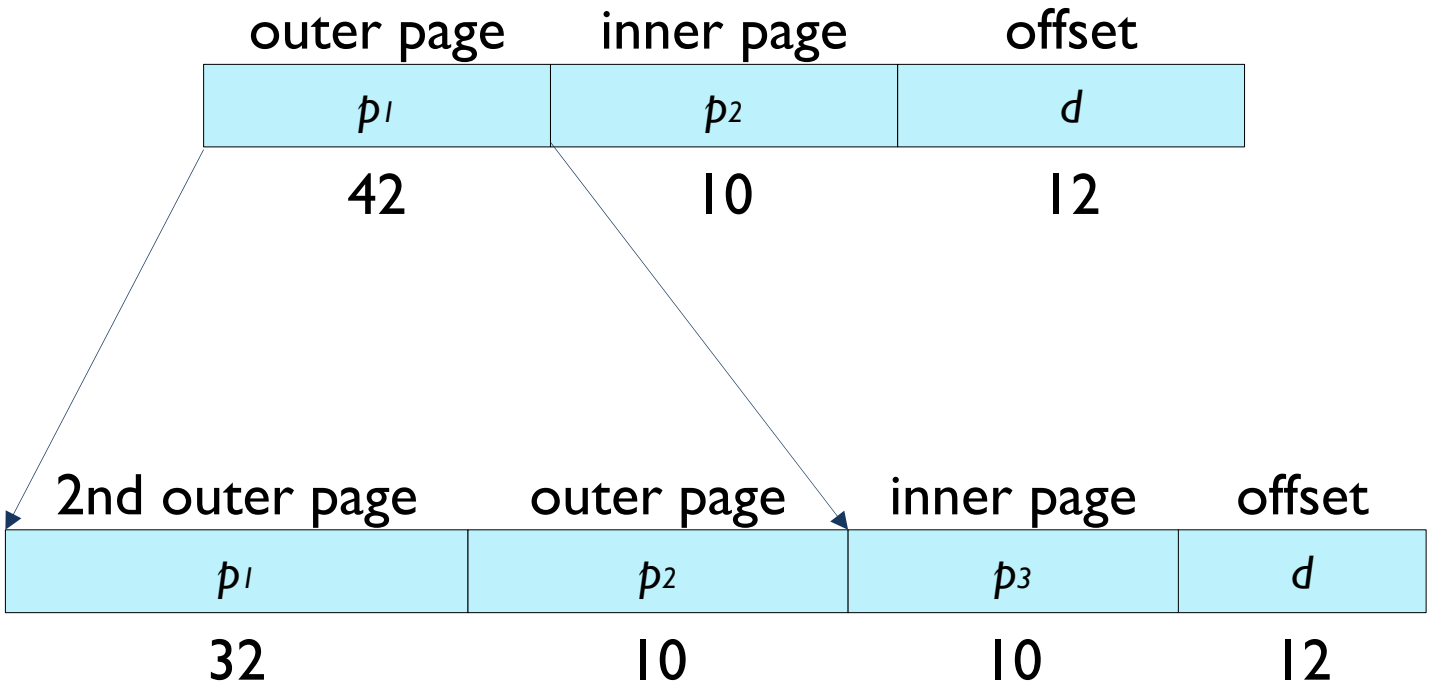
Two-level Page Table Scheme

- Address translation for 32 bits processor
  - A logical address (on 32-bit with 4K page size) is divided
    - 20-bits page number
    - 12-bits page offset
  - Since page table itself is paged, page number is further divided
    - 10-bits for index in page directory (outer page table)
    - 10-bits for index in page table (inner page table)



Three-level Page Table Scheme

- Three level paging for 64 bits processor



Multi-level Page Table Control Flow

```
01:     VPN = (VirtualAddress & VPN_MASK) >> SHIFT
02:     (Success,TlbEntry) = TLB_Lookup(VPN)
03:     if(Success == True)        //TLB Hit
04:       if(CanAccess(TlbEntry.ProtectBits) == True)
05:             Offset = VirtualAddress & OFFSET_MASK
06:             PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
07:             Register = AccessMemory(PhysAddr)
08:       else RaiseException(PROTECTION_FAULT);
09:     else // perform the full multi-level lookup
```

- (1 lines) extract the virtual page number (VPN)
- (2 lines) check if the TLB holds the translation for this VPN
- (5-8 lines) extract the page frame number from the relevant TLB entry, and form the desired physical address and access memory

Multi-level Page Table Control Flow

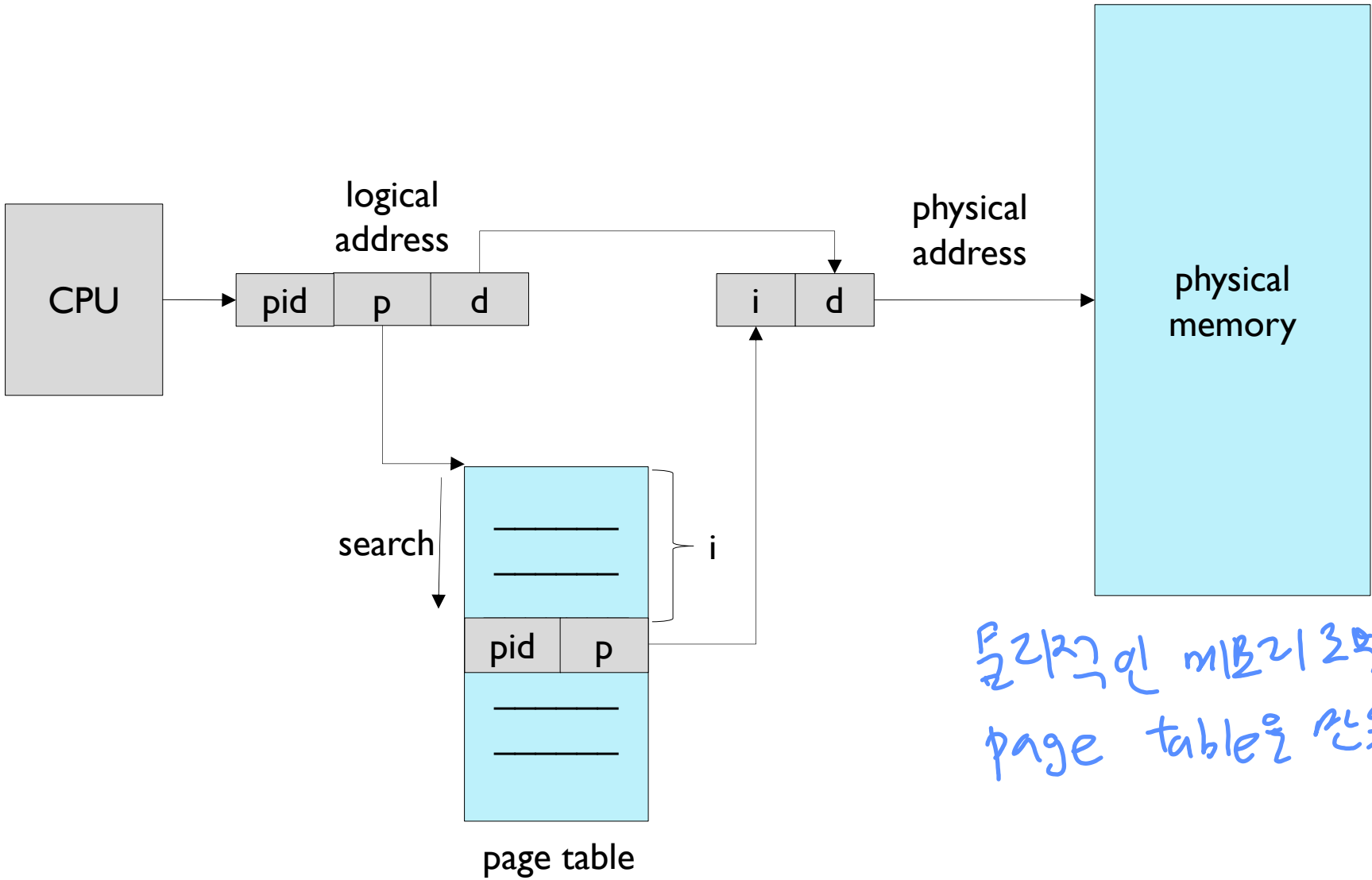
```
11:       else
12:                 PDIndex = (VPN & PD_MASK) >> PD_SHIFT
13:                 PDEAddr = PDBR + (PDIndex * sizeof(PDE))
14:                 PDE = AccessMemory(PDEAddr)
15:                 if(PDE.Valid == False)
16:                           RaiseException(SEGMENTATION_FAULT)
17:                 else // PDE is Valid: now fetch PTE from PT
```

- (11 lines) extract the Page Directory Index (PDIndex)
- (13 lines) get Page Directory Entry (PDE)
- (15-17 lines) Check PDE valid flag. If valid flag is true, fetch Page Table entry from Page Table

The Translation Process: Remember the TLB

```
18:     PTIndex = (VPN & PT_MASK) >> PT_SHIFT
19:     PTEAddr = (PDE.PFN << SHIFT) + (PTIndex * sizeof(PTE))
20:     PTE = AccessMemory(PTEAddr)
21:     if(PTE.Valid == False)
22:             RaiseException(SEGMENTATION_FAULT)
23:     else if(CanAccess(PTE.ProtectBits) == False)
24:             RaiseException(PROTECTION_FAULT);
25:     else
26:             TLB_Insert(VPN, PTE.PFN , PTE.ProtectBits)
27:             RetryInstruction()
```

Inverted Page Table



물리적인 메모리로부터 page table을 만든다

Inverted Page Table

- The space overhead of page table
  - The size of page table is proportional to program size
    - One page table entry for each page
  - Actually, just some pages are required at a time
- Inverted page table
  - One page table entry for each page frame
    - Each page table entry should contain process-id
  - Just one page table is sufficient in system
    - Entries are required as many as the number of page frames
  - It is an associative search, and need to search entire table
    - decreases memory needed to store each page table, but
    - increases time needed to search the table
    - So, a hash table or associative registers should be used