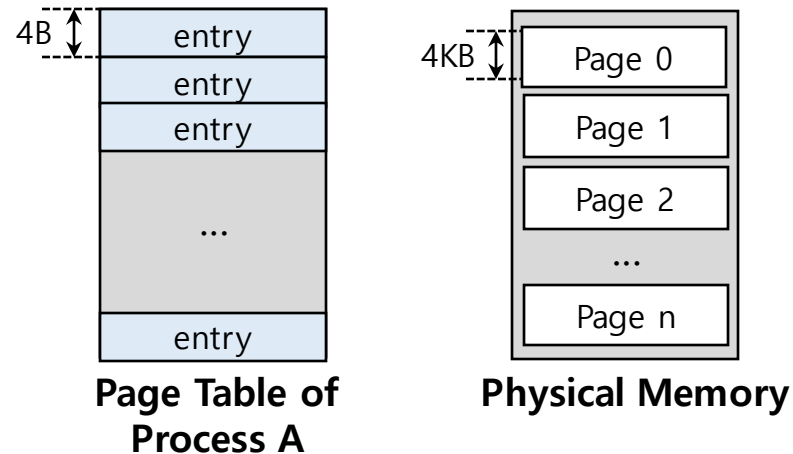


# Smaller Tables

Operating System (OS)  
안인규

# 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.

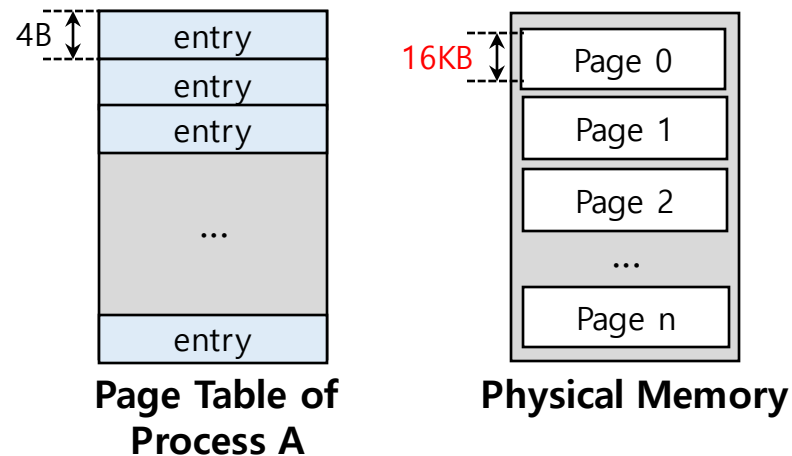


$$\text{Page table size} = \frac{2^{32}}{2^{12}} * 4\text{Byte} = 4\text{MByte}$$

Page table are **too big** and thus consume too much memory.

# Paging: Smaller Tables

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

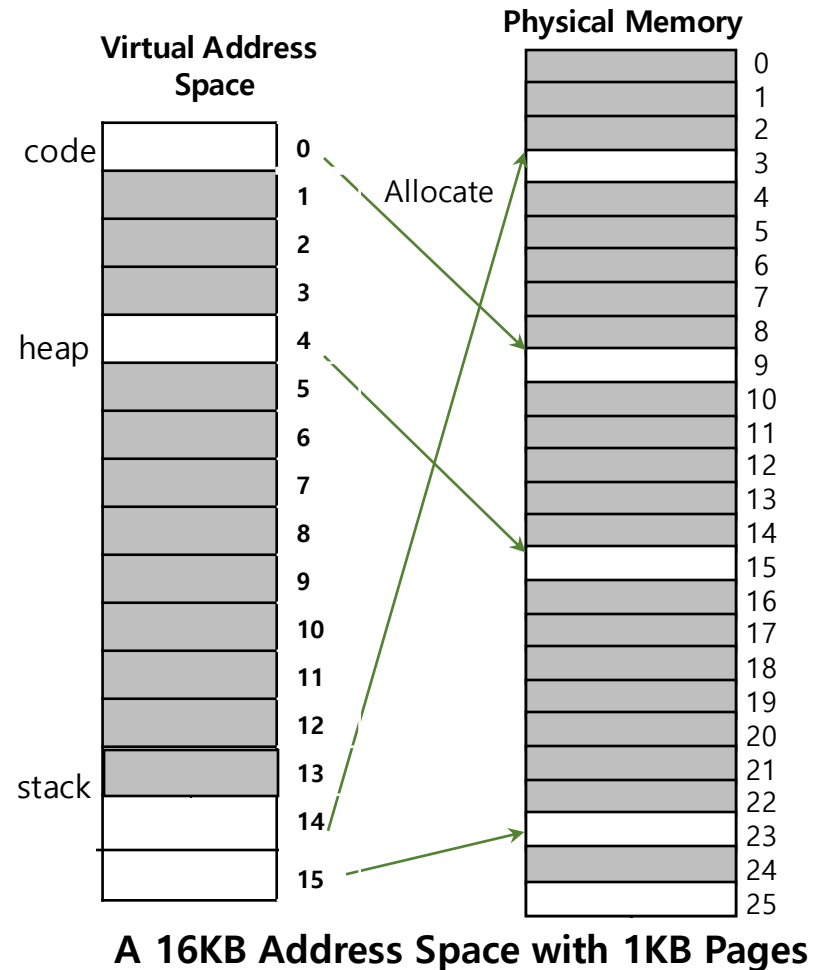


$$\frac{2^{32}}{2^{16}} * 4 = 1MB \text{ per page table}$$

Big pages lead to **internal fragmentation**.

# Problem

- Single page table for the address space of process.

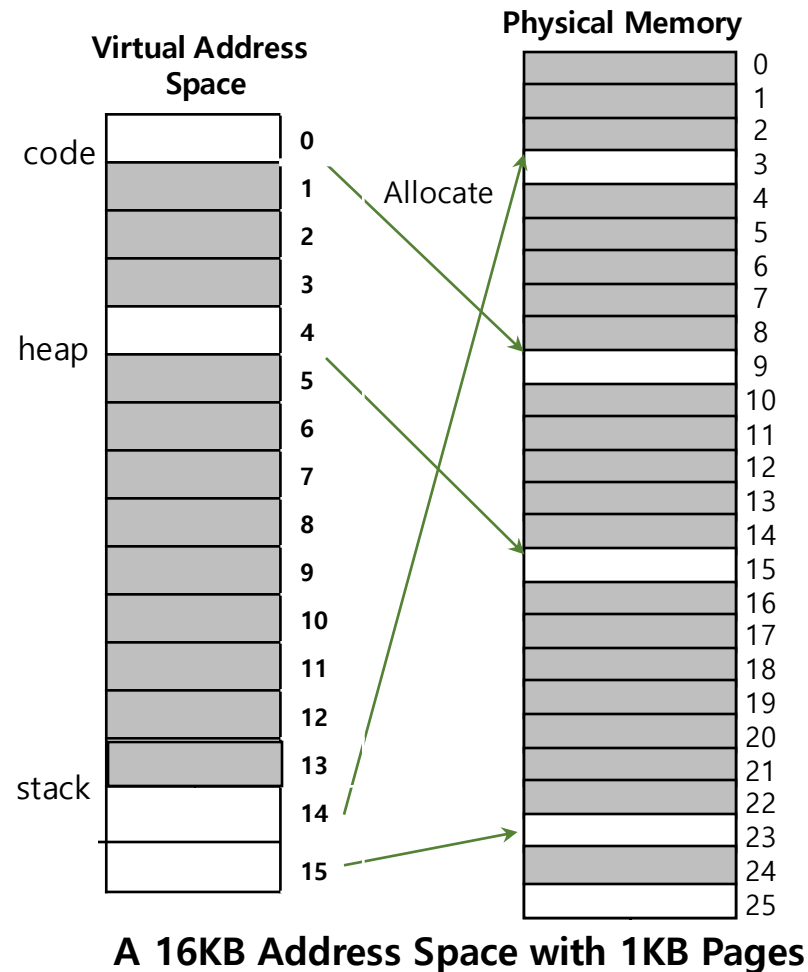


PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...	...	...	...	...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

**A Page Table For 16KB Address Space**

# Problem

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

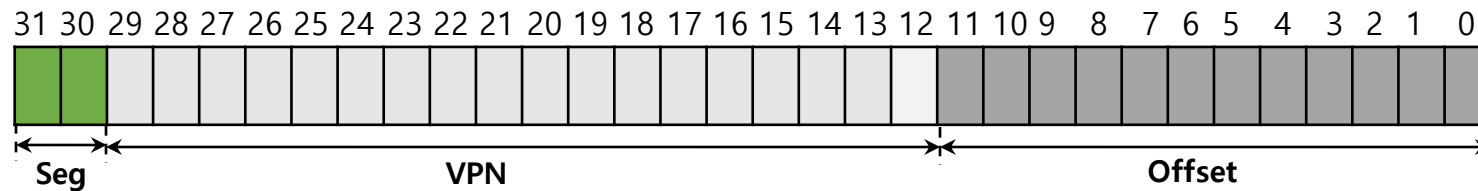


PFN	valid	prot	present	dirty
9	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...	...	...	...	...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

**A Page Table For 16KB Address Space**

# Hybrid Approach: Paging and Segments

- Page table for each segment
  - the base register for each of these segments contains the physical address of a linear page table for that segment.
  - The bound register: indicate the end of the page table.
- Example: Each process has **three** page tables associated with it.



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 gets **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))
```

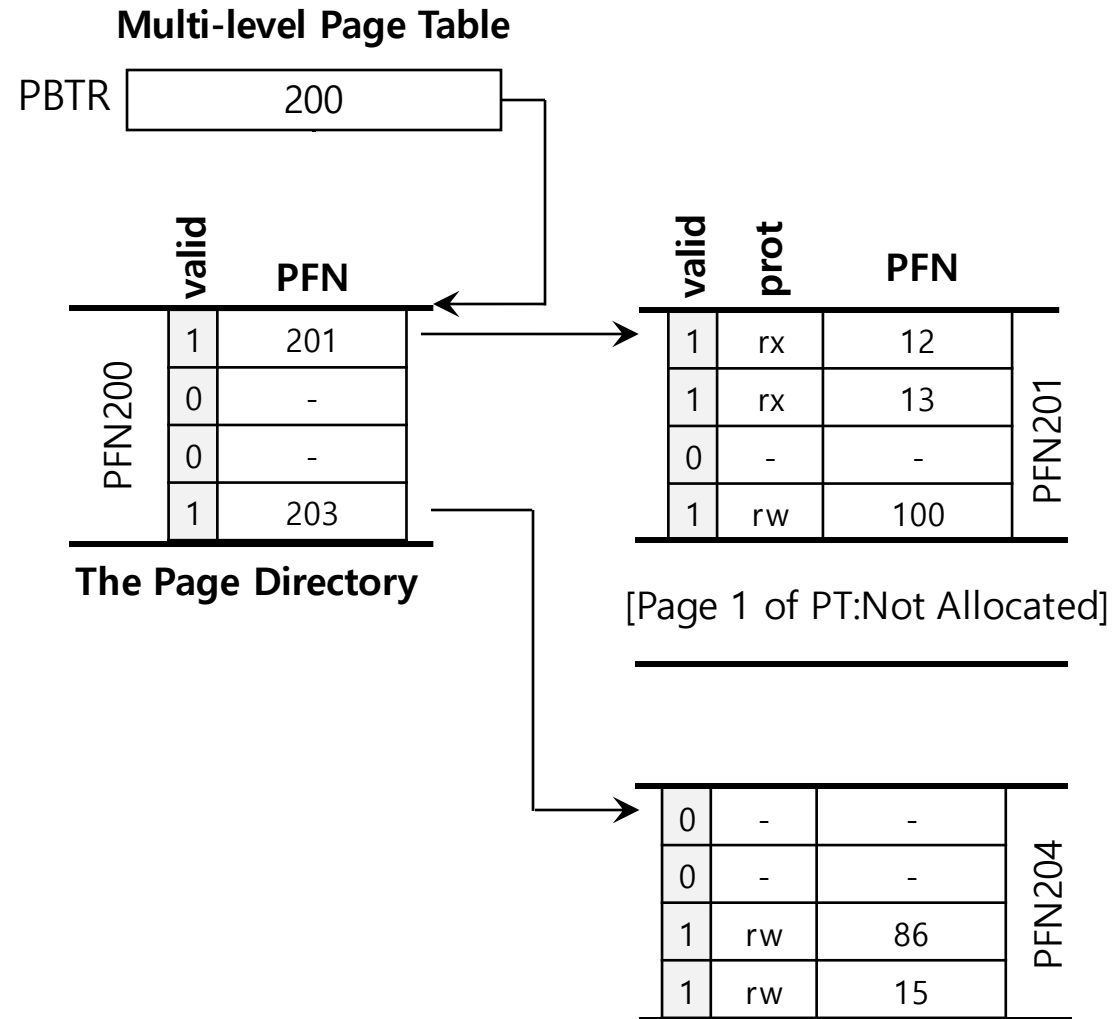
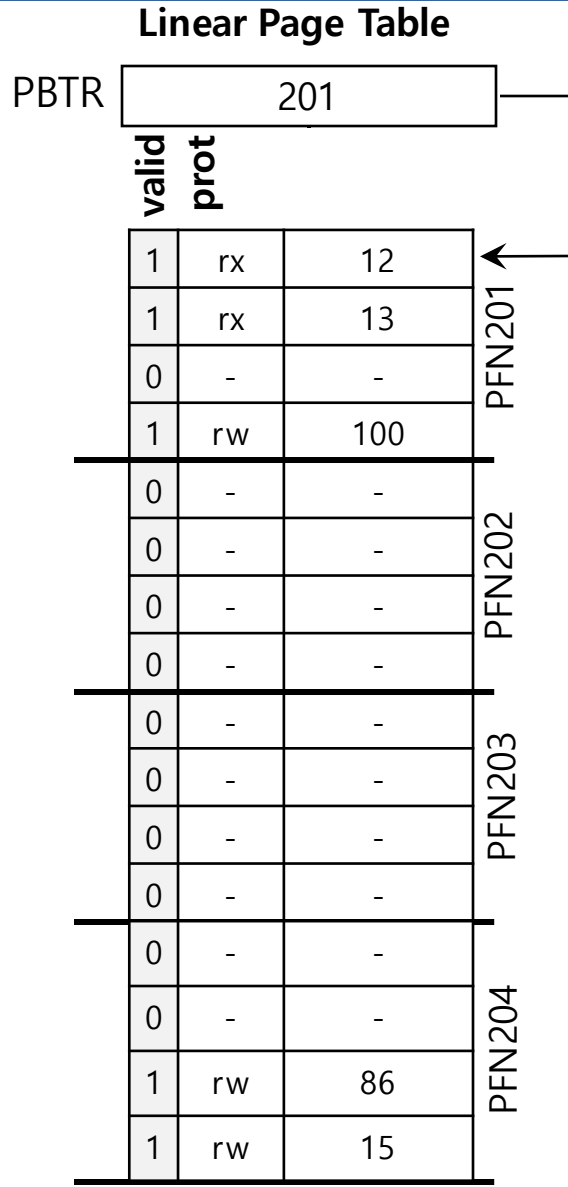
# Problem of Hybrid Approach

- 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.

# Multi-level Page Tables

- Turn 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 a new structure, called [page directory](#)

# Multi-level Page Tables: Page directory



# Multi-level Page Tables

- Page Directory
  - 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)
- 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
- Disadvantage
  - Multi-level table is a small example of a **time-space trade-off**
  - **Complexity**

# Example

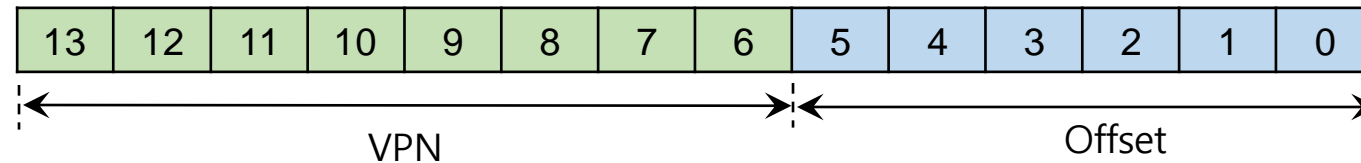
0000 0000	code
0000 0001	code
...	(free)
	(free)
0000 0100	heap
0000 0101	heap
....	(free)
	(free)
1111 1110	stack
1111 1111	stack

- Page 0,1: code
- Page 4,5: heap
- Page 254, 255: stack

# Example

Flag	Detail
Address space	16 KB ( $2^{14}$ Byte)
Page size	<b>64 byte</b>
Virtual address	14 bit
VPN	8 bit
Offset	6 bit
Page table entry	4 Byte

**A 16-KB Address Space With 64-byte Pages**



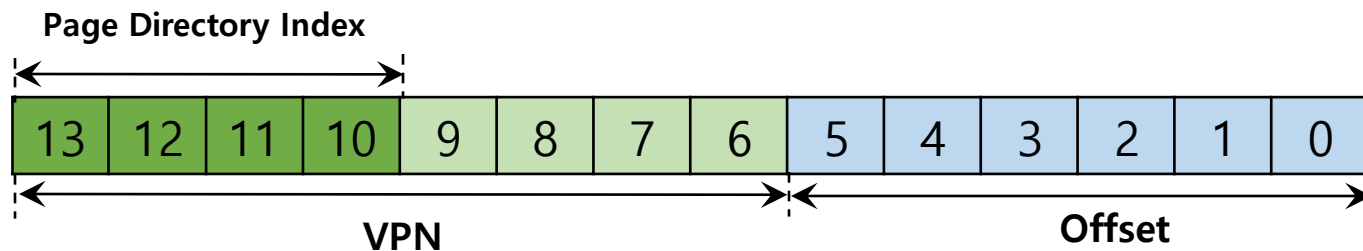
- Single level paging
  - 256 page table entries:  $2^8$  entries
  - Page table size:  $256 * 4 \text{ Byte} = 1 \text{ Kbyte}$
  - Sixteen pages (64 byte each):  $1024/64 = 16$



# Example: two level paging

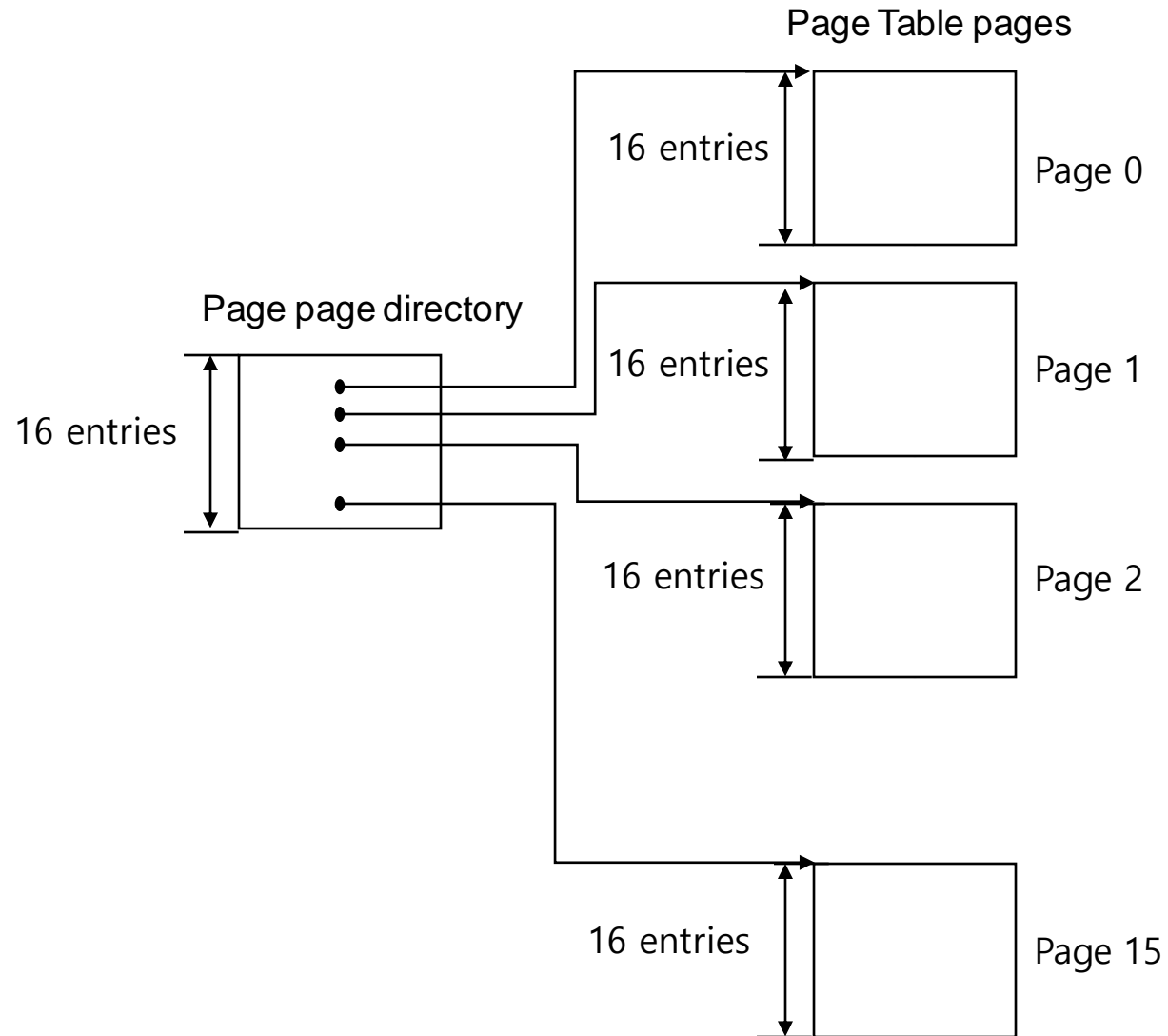
- Page directory index
  - A page table consists of 16 pages.
  - 16 entries for page directory: one entry per page of the page table.
  - $16 * 4 \text{ byte} = 64 \text{ byte}$  is required for page directory. → it can fit in a page.
  - 4 bits for page directory index.

$$\text{PDEAddr} = \text{PageDirBase} + (\text{PDIndex} * \text{sizeof}(\text{PDE}))$$



- If the page-directory entry is **invalid**, raise an exception (The access is invalid).

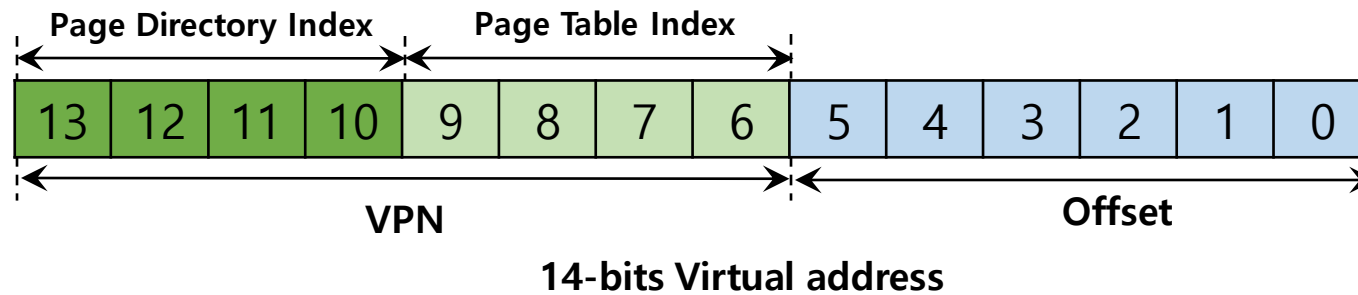
# Example: two level paging



# Example: two level paging

- Page table index
  - It is used to find the address of the page table entry.

$$\text{PTEAddr} = (\text{PDE.PFN} \ll \text{SHIFT}) + (\text{PTIndex} * \text{sizeof}(\text{PTE}))$$

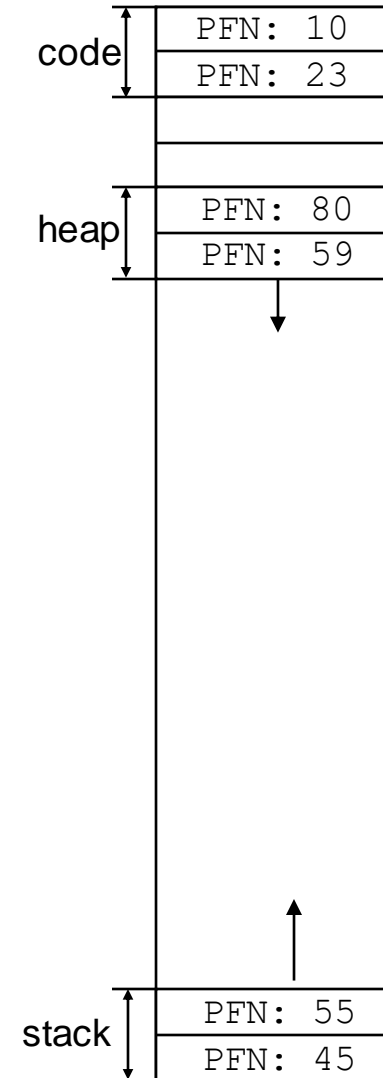


# Example

Page Directory		Page of PT (@PFN:100)			Page of PT (@PFN:101)		
PFN	valid?	PFN	valid	prot	PFN	valid	prot
100	1	10	1	r-x	—	0	—
—	0	23	1	r-x	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	80	1	rw-	—	0	—
—	0	59	1	rw-	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	55	1	rw-
101	1	—	0	—	45	1	rw-

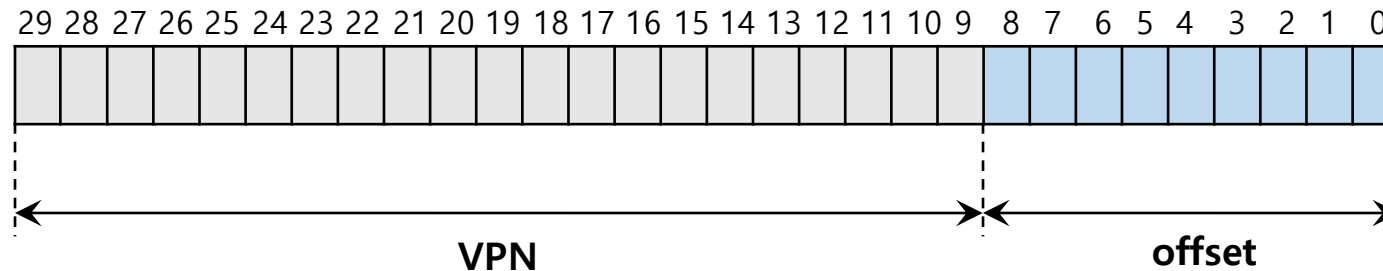
Single level paging: 16 pages

→ Two level paging: 3 pages



# More than Two Levels

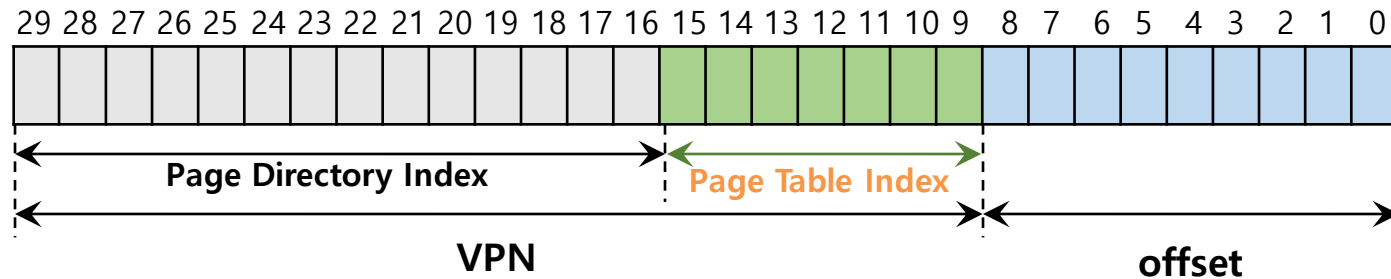
- In some cases, a deeper tree is required.
- Consider the following address space.



Flag	Detail
Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit

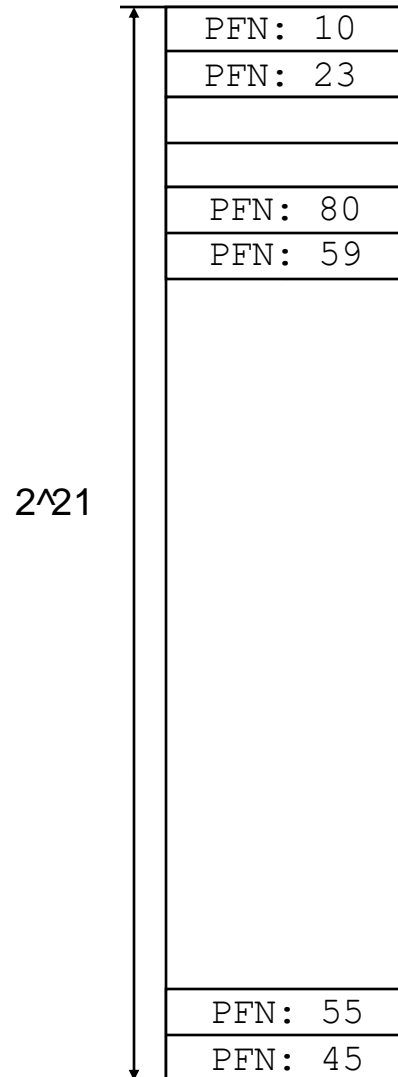
# More than Two Level : Page Table Index

- Number of pages in a page table
  - $2^{21}$  page table entries
  - $2^7$  (128) PTE's in a single page (512/4, 512 page size, 4 byte pte size)
  - $2^{14}$  pages in a page table
- Number of page directory entries:  $2^{14}$

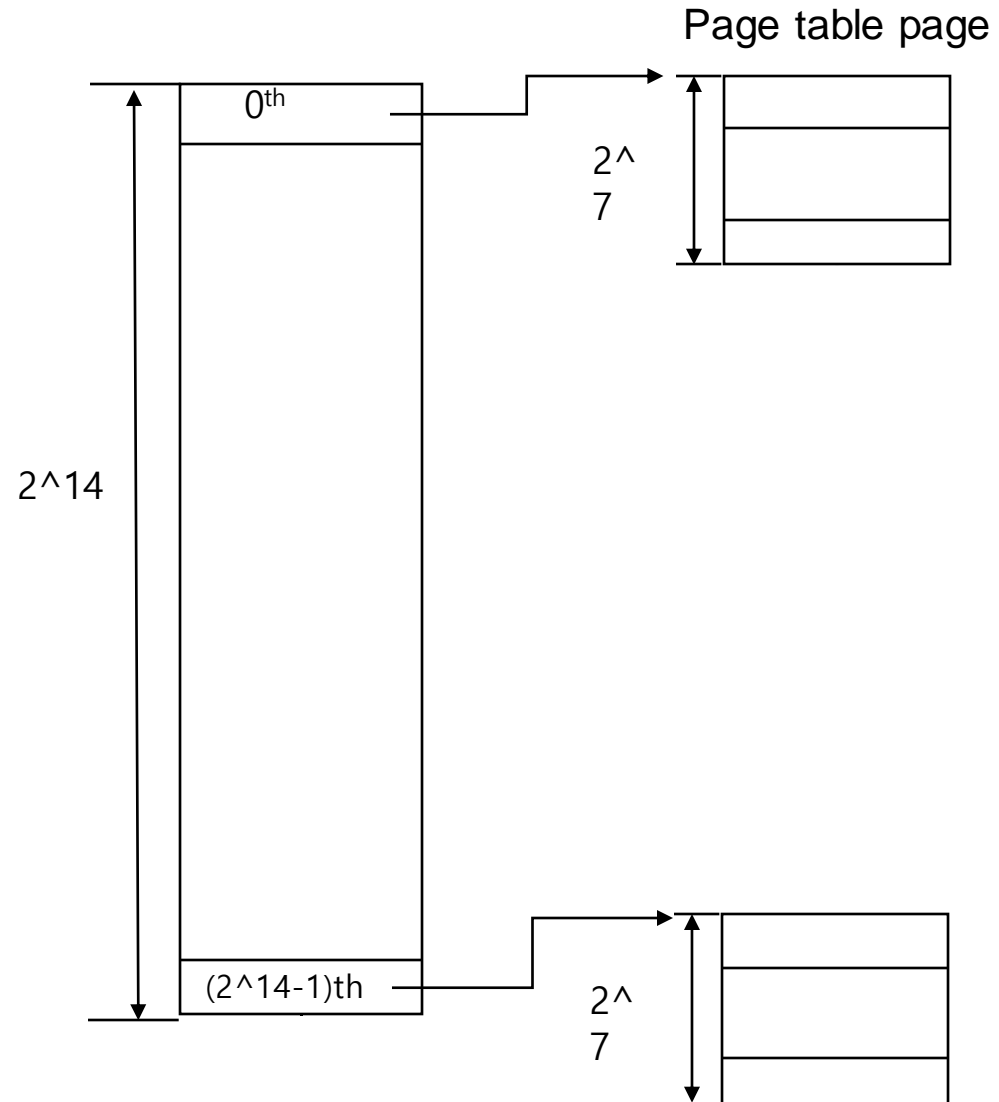


Flag	Detail
Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit
Page entry per page	128 PTEs (512/4)

# Single level page table

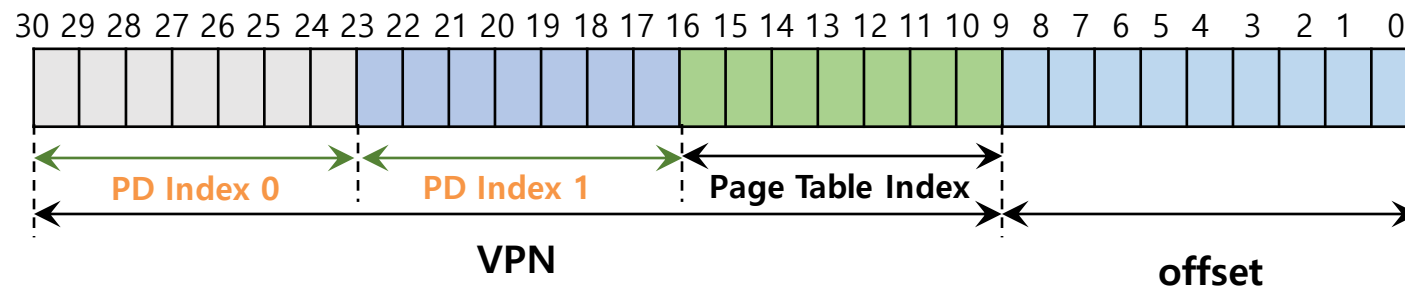


# two level page table

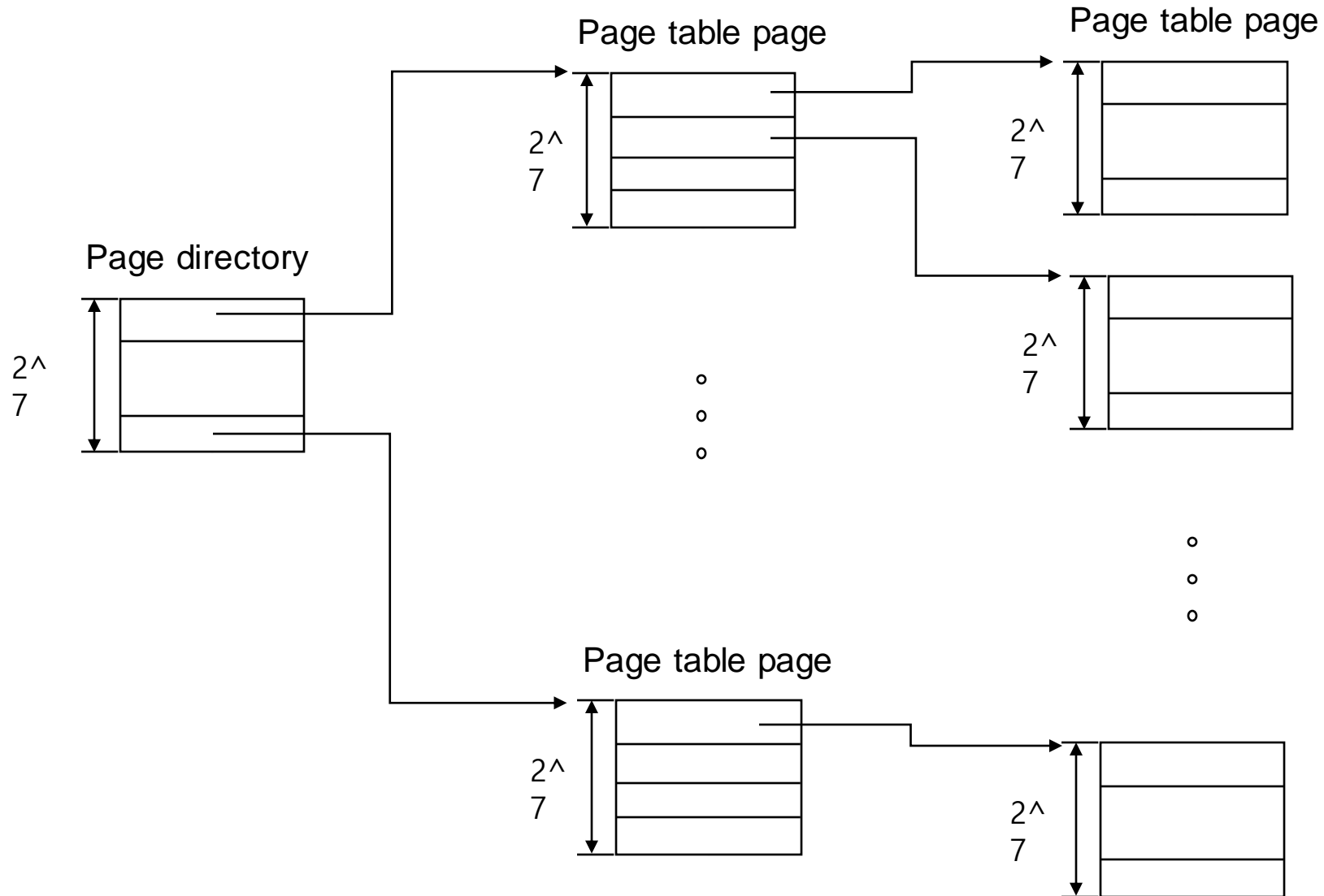


# More than Two Level : Page Directory

- The number of pages in a page directory:  $2^{14}/2^7 = 2^7$  (128)
- Page the page directory



# three level page table



# 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 line) 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 Tables

- Keeping a single page table that has an entry for each physical page of the system.
- The entry tells us which process is using this page, and which virtual page of that process maps to this physical page.

# Summary

- Reducing the page table size
- Per segment page table
- Multi-level paging
  - Efficient use of memory
  - Severe TLB miss