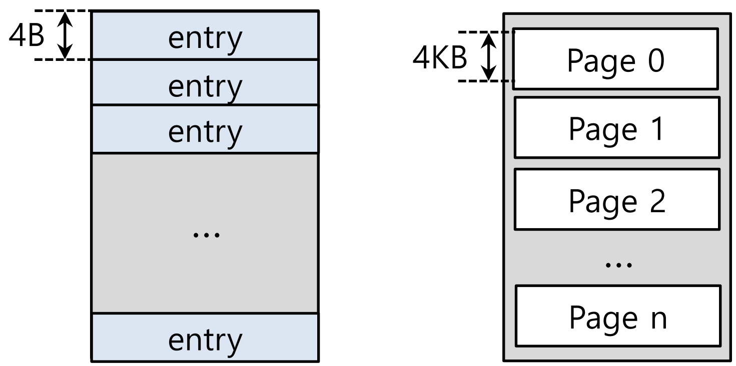
20. Paging: Smaller Tables

Operating System: Three Easy Pieces

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.



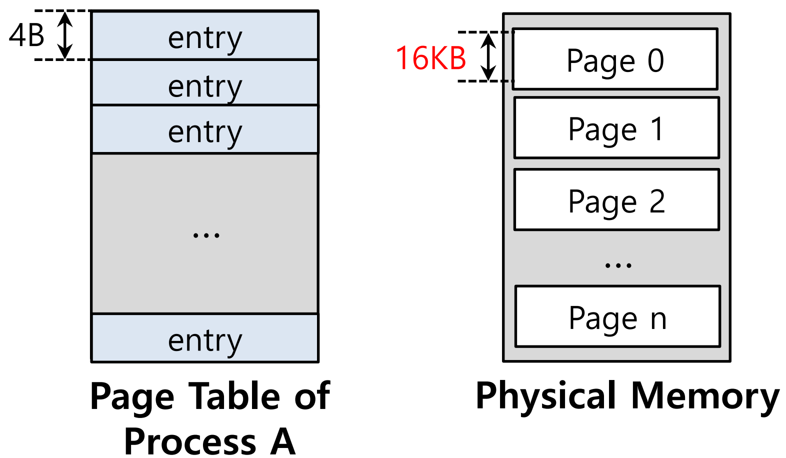
Page Table of Process A **Physical Memory**

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

Page tables 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.

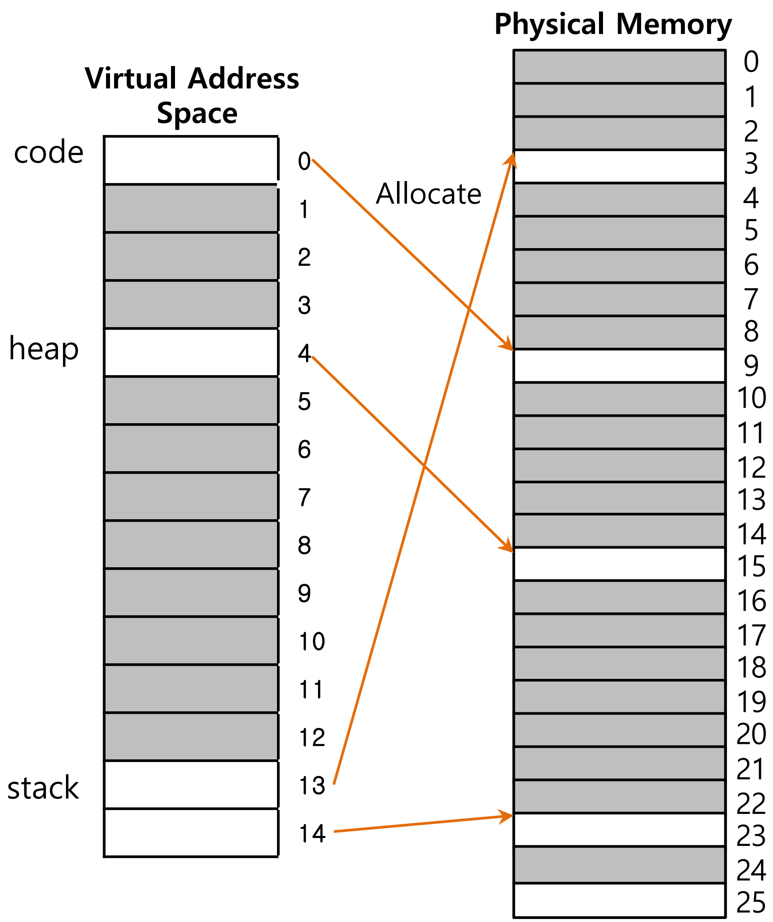


$\frac{2^{32}}{2^{16}} * 4 = 1MB$ **per page table**

But big pages might lead to internal fragmentation.

The Problem

- Single page table for the entries address space of process.



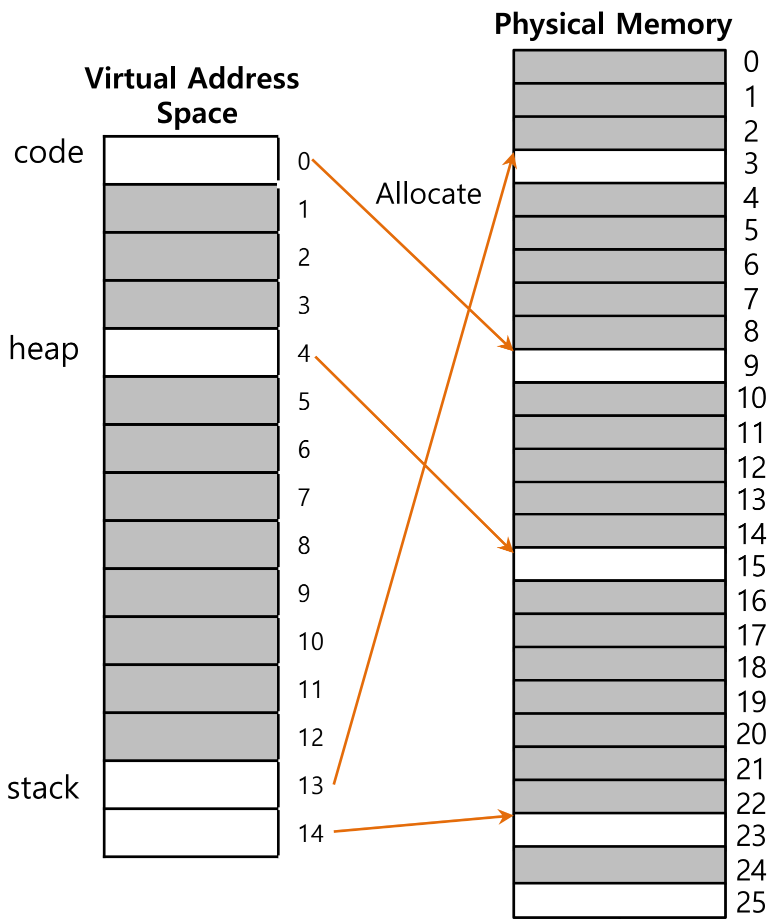
A 16KB Address Space with 1KB Pages

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

The Problem

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



A 16KB Address Space with 1KB Pages

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

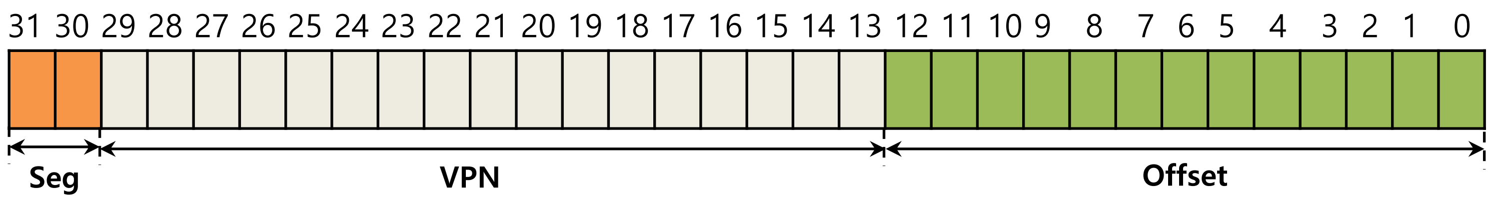
A Page Table For 16KB Address Space

Hybrid Approach to the Problem: Paging and Segments

- In order to reduce the memory overhead of page tables.
  - 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) .

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

Problem of Hybrid Approach

- Hybrid Approach inherits Segmentation issues
  - If we have a large but sparsely-used heap, we can still end up with a lot of page table waste (most of the free space should be tracked)
  - Causing **external fragmentation** to arise again
  - Page Tables have arbitrary size: it’s hard to find space for them (or handle it dynamically)

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 a new structure, called page directory.

Multi-level Page Tables: Page directory

Linear Page Table

PTBR: 201

valid	prot	PFN	
1	rx	12	PFN 201
1	rx	13	
0	-	-	
1	rw	100	
0	-	-	PFN 202
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 203
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 204
0	-	-	
1	rw	86	
1	rw	15	

Multi-level Page Table

PBTR: 200

The Page Directory (PFN200)

valid	PFN
1	201
0	-
0	-
1	203

PFN201

valid	prot	PFN
1	rx	12
1	rx	13
0	-	-
1	rw	100

[Page 1 of PT:Not Allocated]

[Page 2 of PT: Not Allocated]

PFN204

valid	prot	PFN
0	-	-
0	-	-
1	rw	86
1	rw	15

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 (minimally) has a valid bit and page frame number (PFN).

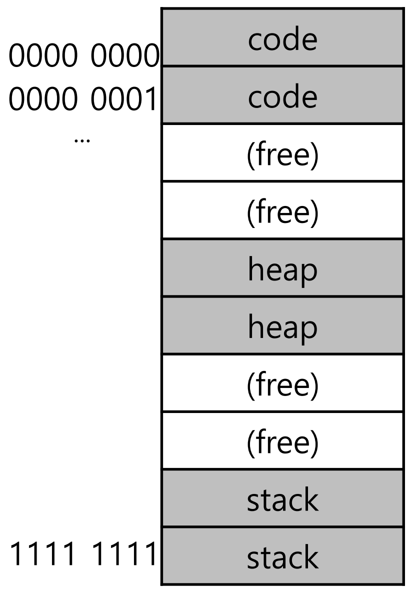
Multi-level Page Tables: Advantage & Disadvantage

- 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.

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

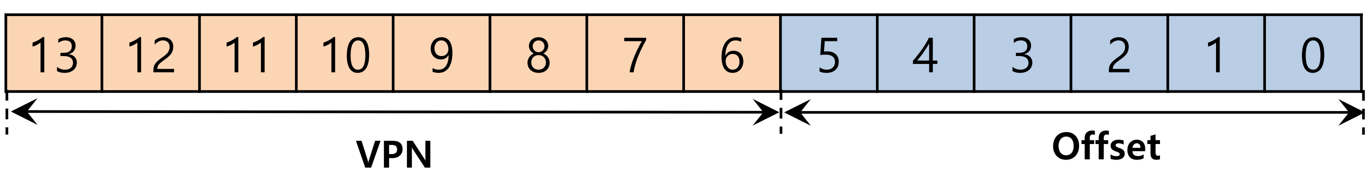
A Detailed Multi-Level Example

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



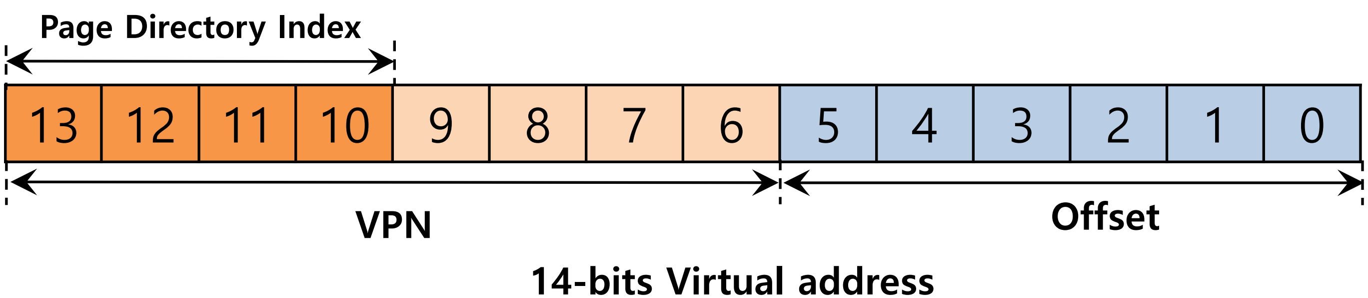
Flag	Detail
Address space	16 KB
Page size	64 byte
Virtual address	14 bit
VPN	8 bit
Offset	6 bit
# Page table entry	2^8(256)

A 16-KB Address Space With 64-byte Pages



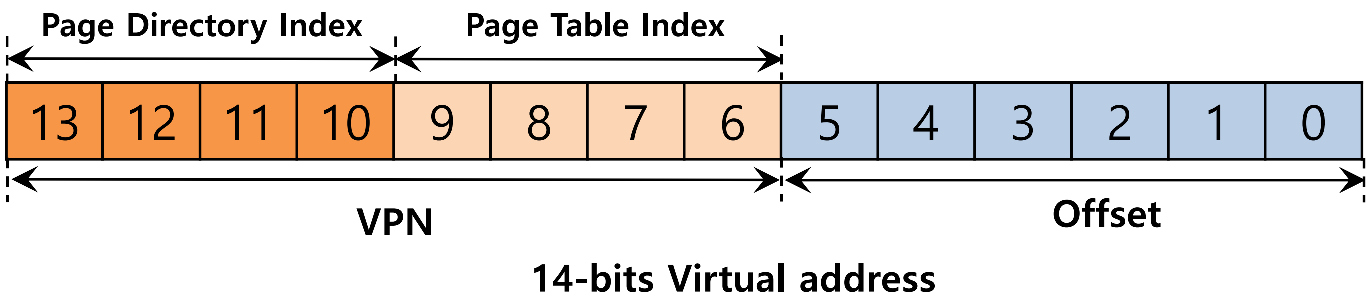
A Detailed Multi-Level Example: Page Directory Idx

- The page directory needs one entry per page of the page table
  - it has 16 entries.
- The PDE 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.



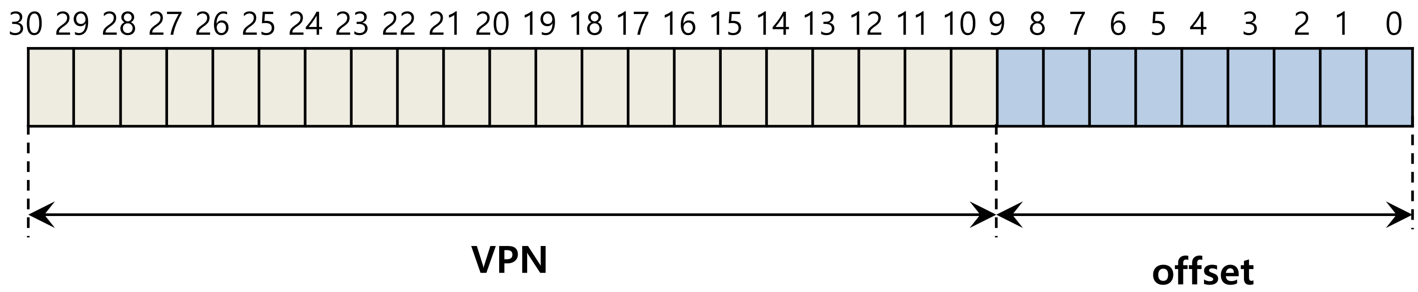
Example

0000 0000	code
0000 0001	code
0000 0010	(free)
0000 0011	(free)
0000 0100	heap
0000 0101	heap
0000 0110	(free)
0000 0111	(free)
....	... all free ...
1111 1100	(free)
1111 1101	(free)
1111 1110	stack
1111 1111	stack

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-

More than Two Levels

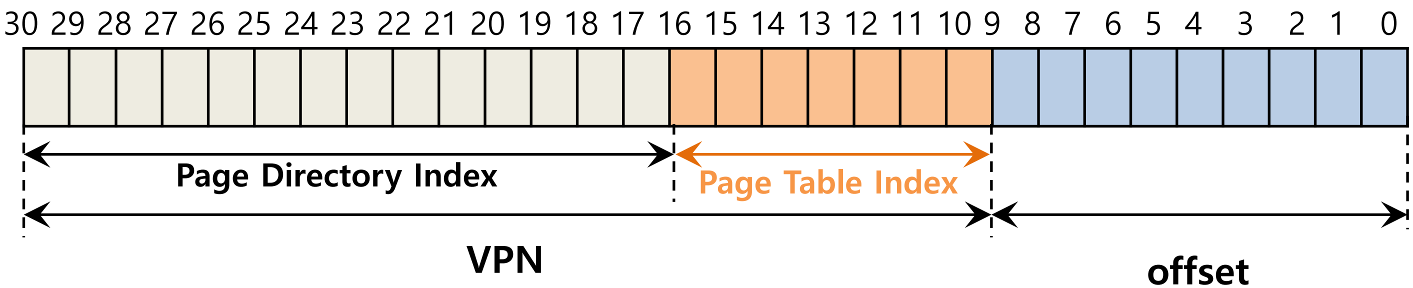
- In some cases, a deeper tree is possible (and needed).



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

More than Two Level : Page Table Index

- In some cases, a deeper tree is possible (and needed).



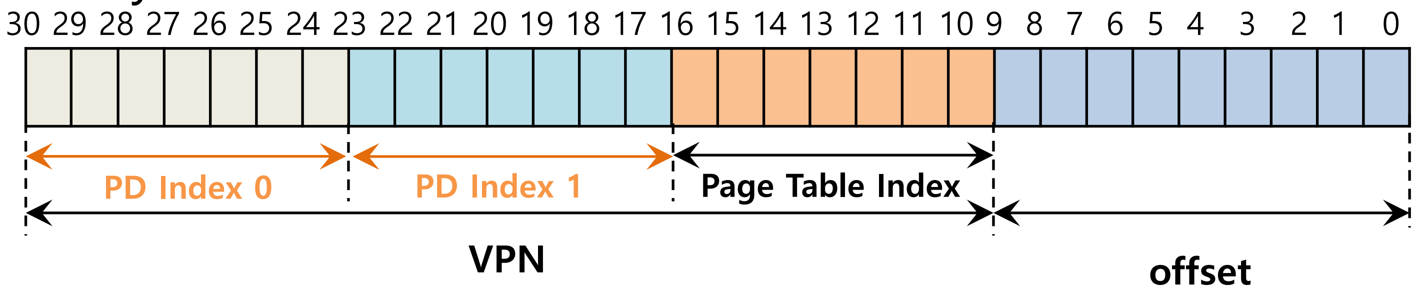
Flag	Detail
Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit
Page entry per page	128 PTEs

$\log_2 128 = 7$
4bytes per PTE

$PD\ has\ 2^{14}\ entries = 16K \cdot 4 = 64KB$

More than Two Level : Page Directory Within a Page

- If our page directory has 2^{14} entries, it spans not one page but 128
  assuming size_of(PDE) == size_of(PTE)
- To remedy this problem, we build a **further level** of the tree, by splitting the page directory itself into multiple pages of the page directory.



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

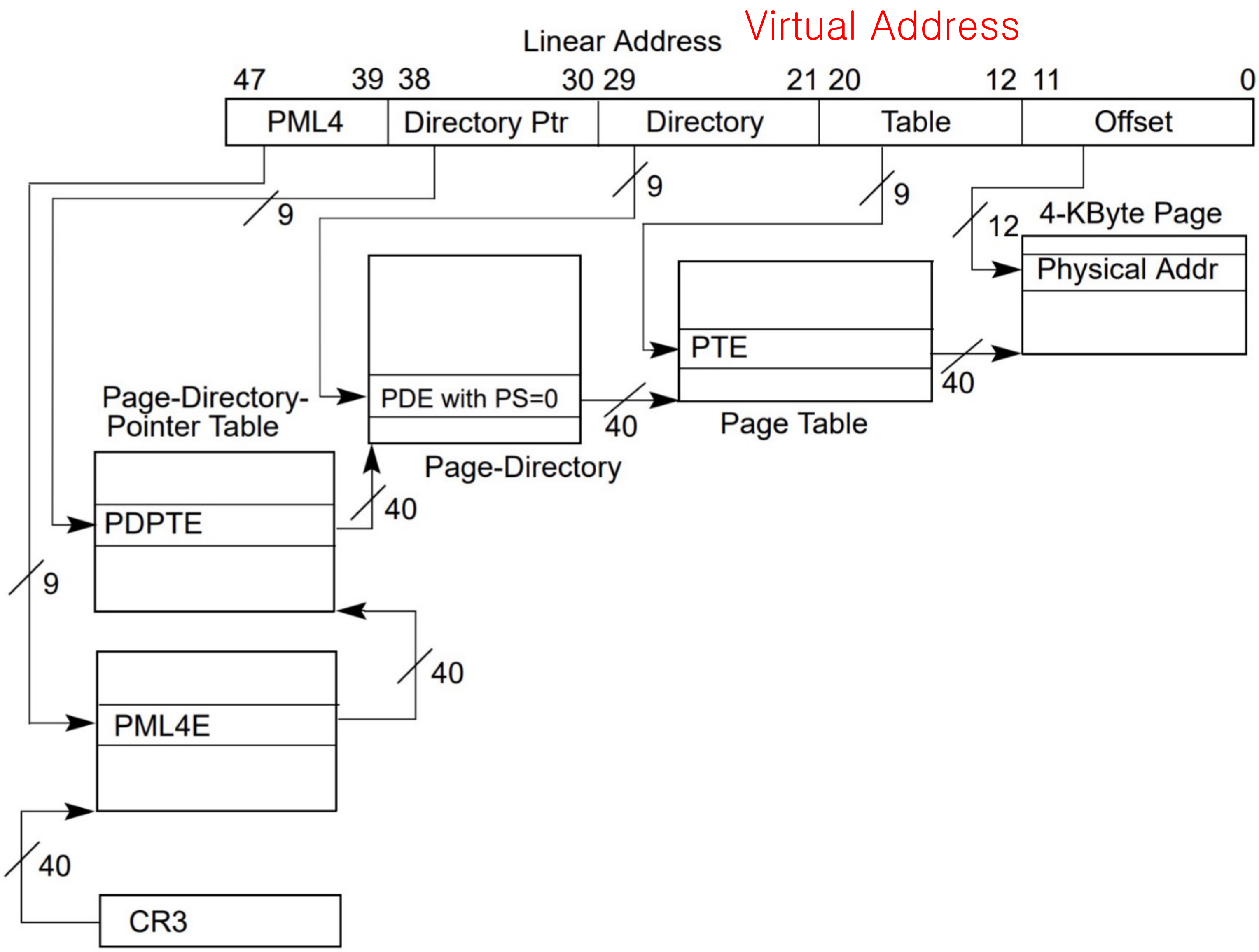
```
11:     else
12:             PDIndex = (VPN & PD_MASK) >> PD_SHIFT
13:             PDEAddr = PDBR + (PDIndex * sizeof(PDE))
14:             PDE = AccessMemory(PDEAddr)
15:             if(PDE.Valid == False)
16:                     RaiseException(SEGMENTATION_FAULT)
17:             else // PDE is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

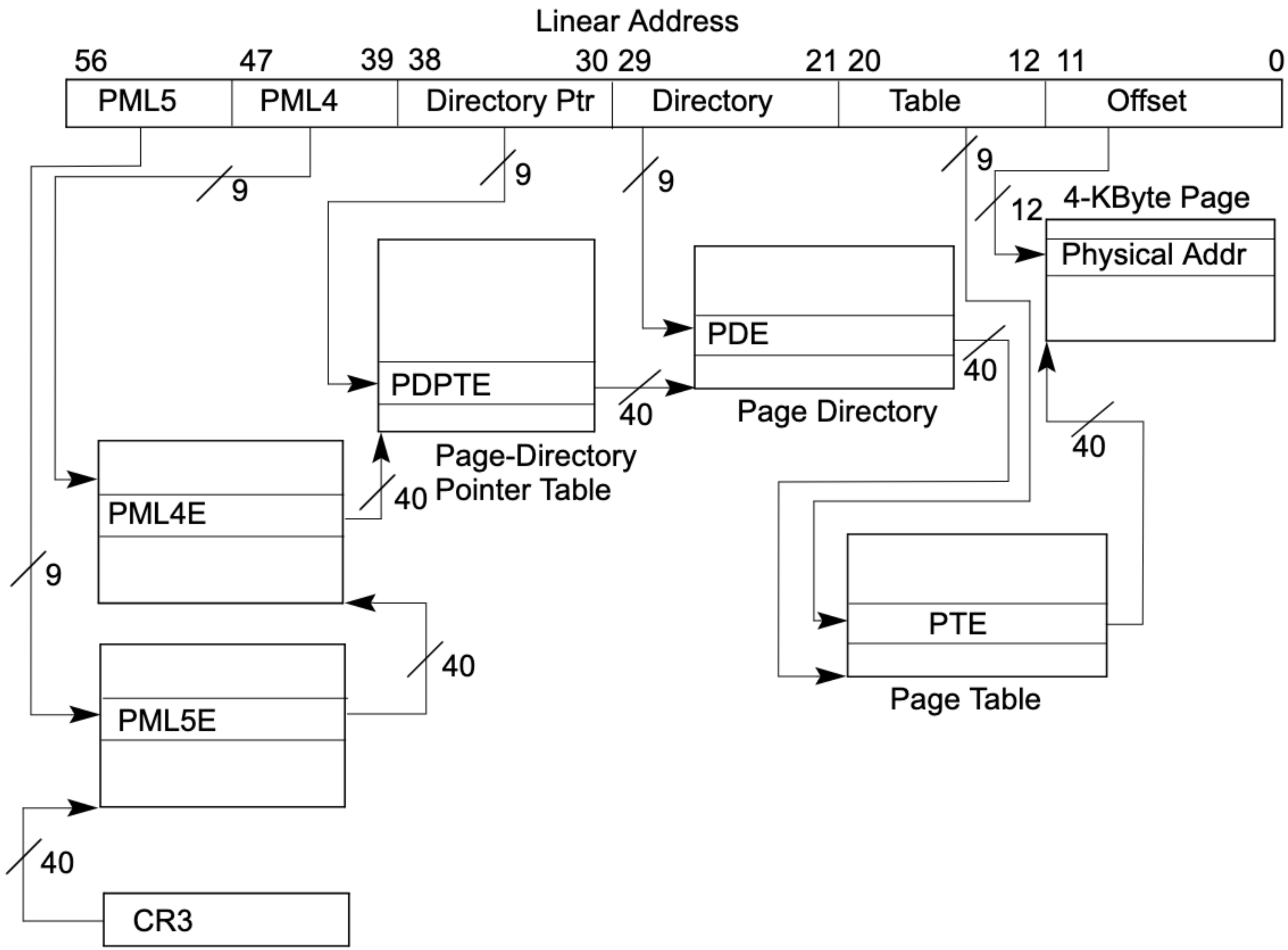
The Translation Process: Remember the TLB

```
PTIndex = (VPN & PT_MASK) >> PT_SHIFT
PTEAddr = (PDE.PFN << SHIFT) + (PTIndex * sizeof(PTE))
PTE = AccessMemory(PTEAddr)
if(PTE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
else if(CanAccess(PTE.ProtectBits) == False)
        RaiseException(PROTECTION_FAULT);
else
        TLB_Insert(VPN, PTE.PFN , PTE.ProtectBits)
        RetryInstruction()
```

Real Multilevel Page Tables (2^{48}B=256TB)

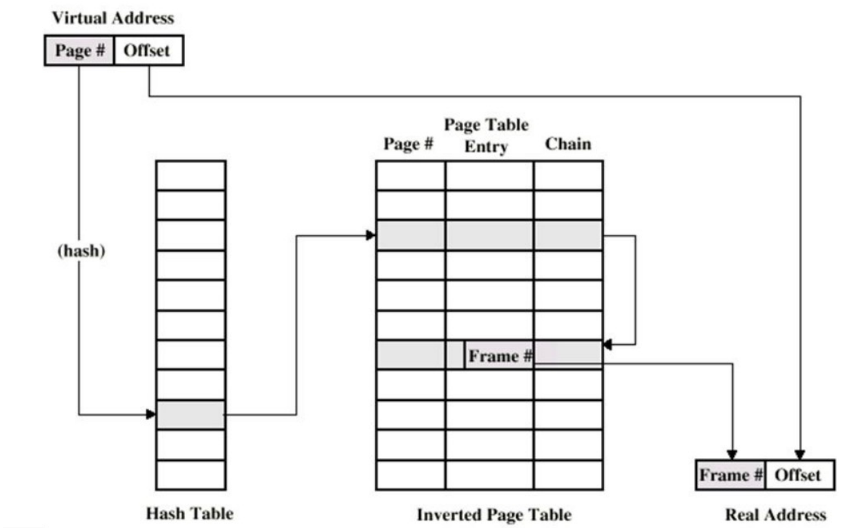


Since 2021 (2^{56}B=128PB)



Inverted Page Table

- Keeping a single page table that has an entry for each <u>physical page</u> 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.
- Used with a hashing table (<u>hash anchor table</u>) in order to allow practical implementations



Swapping Page Tables to Disk

- When memory conditions are harsh, kernel allocated pages can be swapped to disk

- Page tables (a portion of it) might reside only in disk at a given time

- Ignored for sake of simplicity until now (we have no clue about swapping yet)

- This lecture slide has been adapted for AOS course at University of Cantabria by V.Puente. Was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book written by Remzi and Andrea Arpaci-Dusseau (at University of Wisconsin)