OSTEP
Virtualizing Memory: Smaller Page Tables

Questions answered in this lecture:

Review: What are problems with paging?

Review: How large can page tables be?

How can large page tables be avoided with different techniques?

Inverted page tables, segmentation + paging, multilevel page tables

What happens on a TLB miss?

Disadvantages of Paging

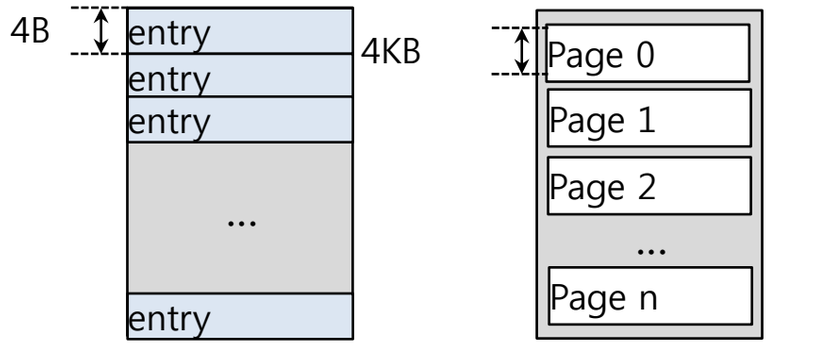
1. Additional memory reference to look up in page table
   - Very inefficient
   - Page table must be stored in memory
   - MMU stores only base address of page table
   - Avoid extra memory reference for lookup with TLBs (previous lecture)

2. Storage for page tables may be substantial
   - Simple page table: Requires PTE for all pages in address space
     - Entry needed even if page not allocated
   - **Problematic with dynamic stack and heap within address space (today)**
     : Too big

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**

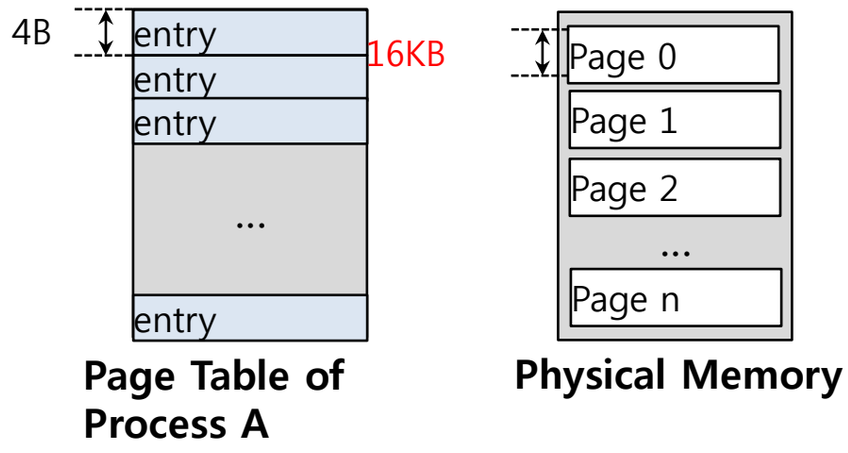
Page table size = $\frac{2^{32}}{2^{12}} * 4Byte = 4MByte$

Page 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.



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

Big pages lead to internal fragmentation.

QUIZ: How big are page Tables?

1. PTE's are **2 bytes**, and **32** possible virtual page numbers

   32 * 2 bytes = 64 bytes
2. PTE's are **2 bytes**, virtual addrs are **24 bits**, pages are **16 bytes**

   2 bytes * 2^(24 – lg 16) = **2^21 bytes** (2 MB)
3. PTE's are **4 bytes**, virtual addrs are **32 bits**, and pages are **4 KB**

   4 bytes * 2^(32 – lg 4K) = **2^22 bytes** (2 MB)
4. PTE's are **4 bytes**, virtual addrs are **64 bits**, and pages are **4 KB**

   4 bytes * 2^(64 – lg 4K) = **2^54 bytes**

How big is each page table?

Motivation

Why do we want big virtual address spaces?
- programming is
- applications need not worry (as much) about fragmentation

Paging goals:
- **space efficiency (don't waste on invalid data)**
- simplicity (no bookkeeping should require contiguous pages)

Approach 1: Change Page Size

- Make pages bigger
- Why are 4 MB pages bad? Internal fragmentation.

Mixed Page Sizes

Some systems support multiple page sizes
- better TLB is bigger motivation, though

Mechanisms: what are implications for
- PTs?
- TLBs?

Policy: when to use large pages?

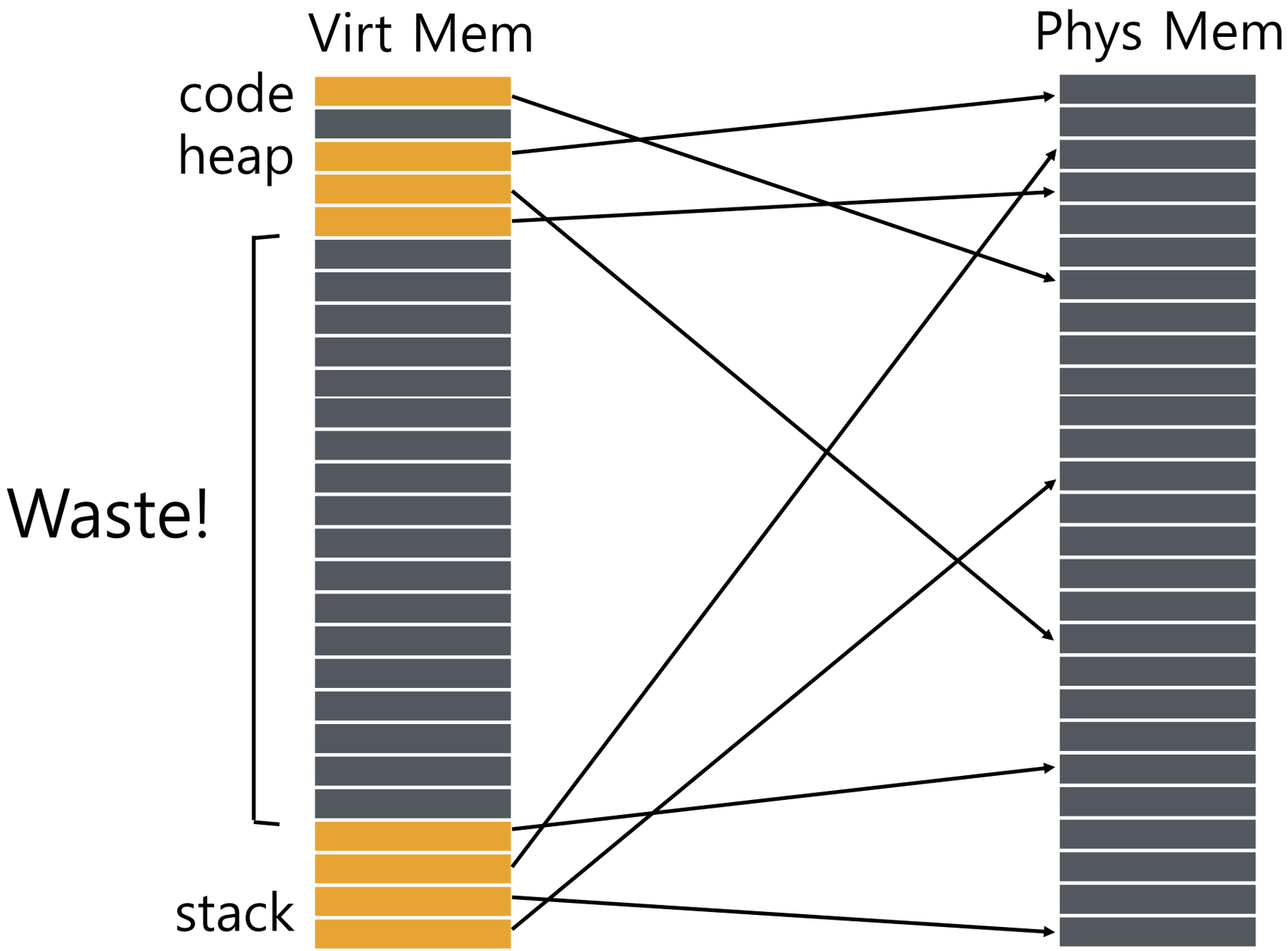
Approach 2: Abandon Simple Linear Page Tables

Use more complex page tables, instead of just big array

Any data structure is possible with software-managed TLB

- Hardware looks for vpn in TLB on every memory access
- If TLB does not contain vpn, TLB miss
- Trap into OS and let OS find vpn->ppn translation
- OS notifies TLB of vpn->ppn for future accesses

Why ARE Page Tables so Large?



Many invalid PT entries

Format of linear page tables:

PFN	valid	prot
10	1	r-x
-	0	-
23	1	rw-
-	0	-
-	0	-
-	0	-
-	0	-
…many more invalid…		
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

how to avoid storing these?

Approach 2a: hash-table lookup

Called an inverted page table.

Pros/Cons?

Nice if we trapped on TLB misses…

Inverted Page TAble

Inverted Page Tables

- Only need entries for virtual pages w/ valid physical mappings

Naïve approach:
Search through data structure <ppn, vpn+asid> to find match

- Too much time to search entire table

Better: Find possible matches entries by hashing vpn+asid

- Smaller number of entries to search for exact match

Managing inverted page table requires software-controlled TLB

For hardware-controlled TLB, need well-defined, simple approach

Valid PTEs are Contiguous

PFN	valid	prot
10	1	r-x
-	0	-
23	1	rw-
-	0	-
-	0	-
-	0	-
-	0	-
…many more invalid…		
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

how to avoid storing these?

Note “hole” in addr space: valids vs. invalids are clustered

How did OS avoid allocating holes in phys memory?

- Segmentation
- Paging

Other Approaches

2a. Inverted Pagetables (hashtable)
2b. Segmented Pagetables (hybrid)
2c. Multi-level Pagetables

- Page the page tables (PTs over PTs)

2d. Multi-level Pagetables

- Page the pagetables of page tables… (PTs over Pts over PTs)

Combine Paging and Segmentation

Divide address space into segments (code, heap, stack)

- Segments can be variable length

Divide each segment into fixed-sized pages

Logical address divided into three portions

Implementation

- Each segment has a page table
- Each segment track base (physical address) and bounds of **page table** for that segment

Quiz: Paging and Segmentation

seg # (4 bits)	page number (8 bits)	page offset (12 bits)

seg	base	bounds	R W
0	0x002000	0xff	1 0
1	0x000000	0x00	0 0
2	0x001000	0x0f	1 1

0x002070 read: 0x004070

0x202016 read: 0x003016

0x104c84 read: error

0x010424 write: error

0x210014 write: error

0x203568 read: 0x02a568

...	
0x01f	0x001000
0x011	
0x003	
0x02a	
0x013	
...	
0x00c	0x002000
0x007	
0x004	
0x00b	
0x006	
...	

Advantages of Paging and Segmentation

Advantages of Segments

- Supports sparse address spaces
  - Decreases size of page tables
  - If segment not used, not need for page table

Advantages of Pages

- No external fragmentation
- Segments can grow without any reshuffling
- Can run process when some pages are swapped to disk (next l ecture)

Advantages of Both

- Increases flexibility of sharing
  - Share either single page or entire segment
  - How?

Disadvantages of Paging and Segmentation

Potentially large page tables (for each segment)

- Must allocate each page table contiguously
- More problematic with more address bits
- Page table size?
  - Assume 2 bits for segment, 18 bits for page number, 12 bits for offset

Each page table is:

= Number of entries * size of each entry

= Number of pages * 4 bytes

= 2^18 * 4 bytes = 2^20 bytes = 1 MB!!!

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))
```

Other Approaches

2a. Inverted Pagetables (hashtable)

2b. Segmented Pagetables

2c. Multi-level Pagetables

- Page the page tables (PTs over PTs)

2d. Multi-level Pagetables

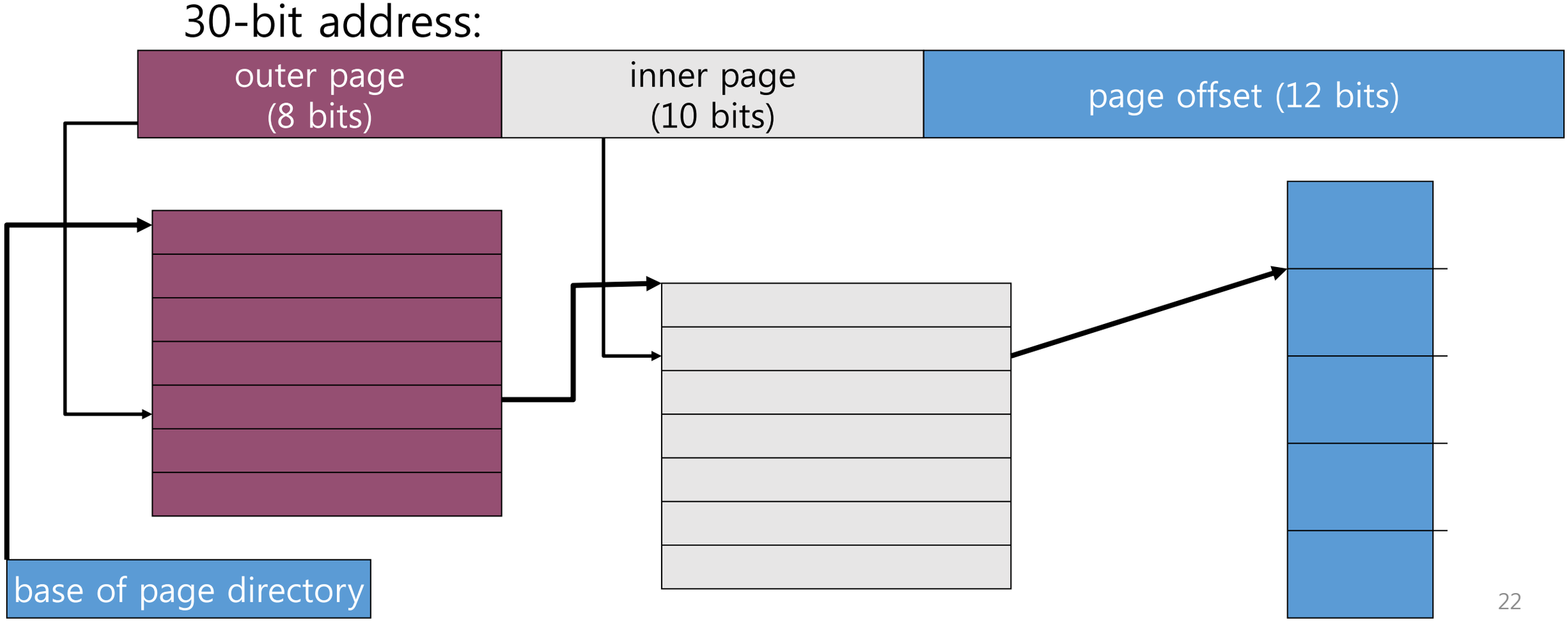
- Page the pagetables of page tables…(PTs over Pts over PTs)

3) Multilevel Page Tables

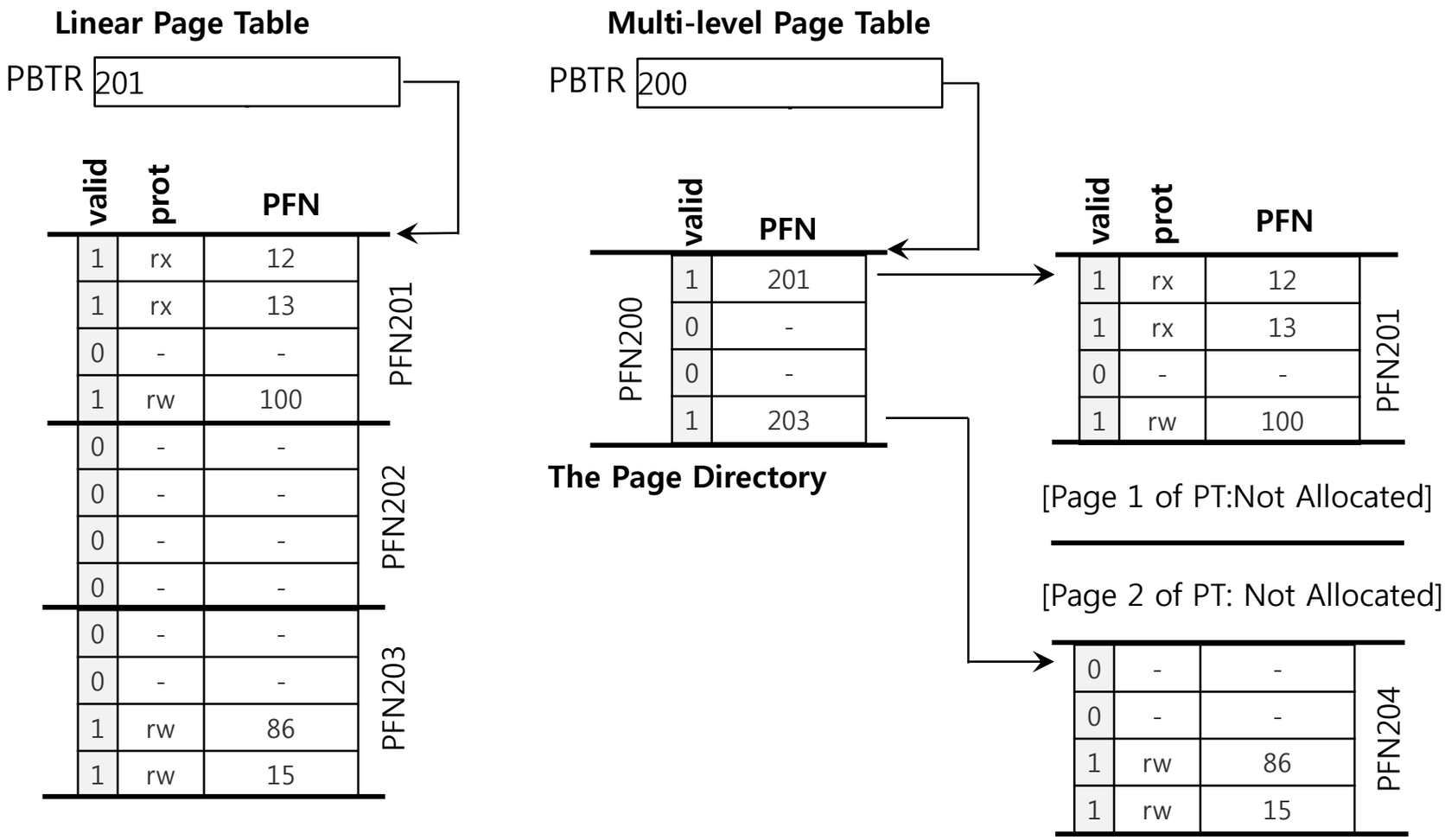
Goal: Allow each page tables to be allocated non-contiguously

Idea: Page the page tables

- Creates multiple levels of page tables; outer level "page directory"
- Only allocate page tables for pages in use
- Used in x86 architectures (hardware can walk known structure)



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).

Quiz: Multilevel

page directory

PPN	valid
0x3	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
0x92	0
	1

page of PT (@PPN:0x3)

PPN	valid
0x10	1
0x23	1
-	0
-	0
0x80	1
0x59	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

page of PT (@PPN:0x92)

PPN	valid
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
0x55	1
0x45	1

translate 0x01ABC

0x23ABC

translate 0x00000

0x10000

translate 0xFEED0

0x55ED0

20-bit address:

QUIZ: Address Format for Multilevel Paging

30-bit address:

How should logical address be structured?

- How many bits for each paging level?

Goal?

- Each page table fits within a page
- PTE size * number PTE = page size
  - Assume PTE size = 4 bytes
  - Page size = 2^12 bytes = 4KB
  - 2^2 bytes * number PTE = 2^12 bytes
  - → number PTE = 2^10
- → # bits for selecting inner page = 10

Remaining bits for outer page:

- 30 – 10 – 12 = 8 bits

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

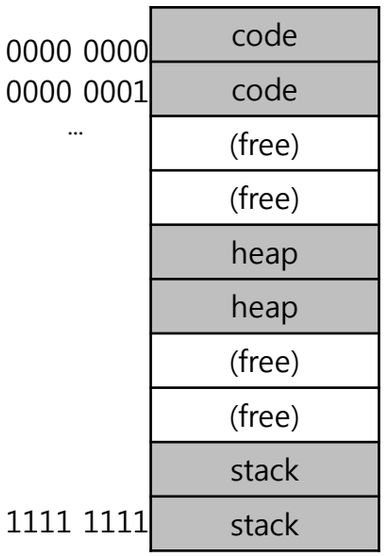
- Multi-level table is a small example of a time-space trade-off.
- Complexity.

Multi-level Page Table: Level of indirection

- A multi-level structure can adjust level of indirection through use of the page directory.
  - Indirection place page-table pages wherever we would like in physical memory.

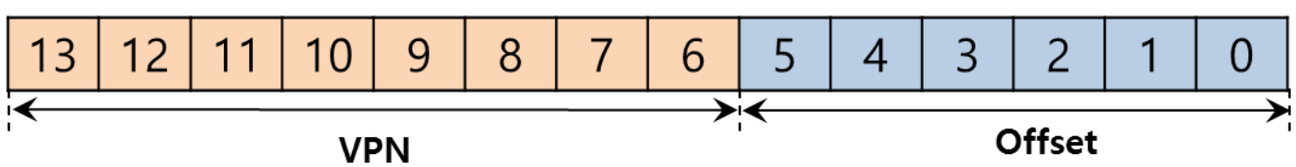
A Detailed Multi-Level Example

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

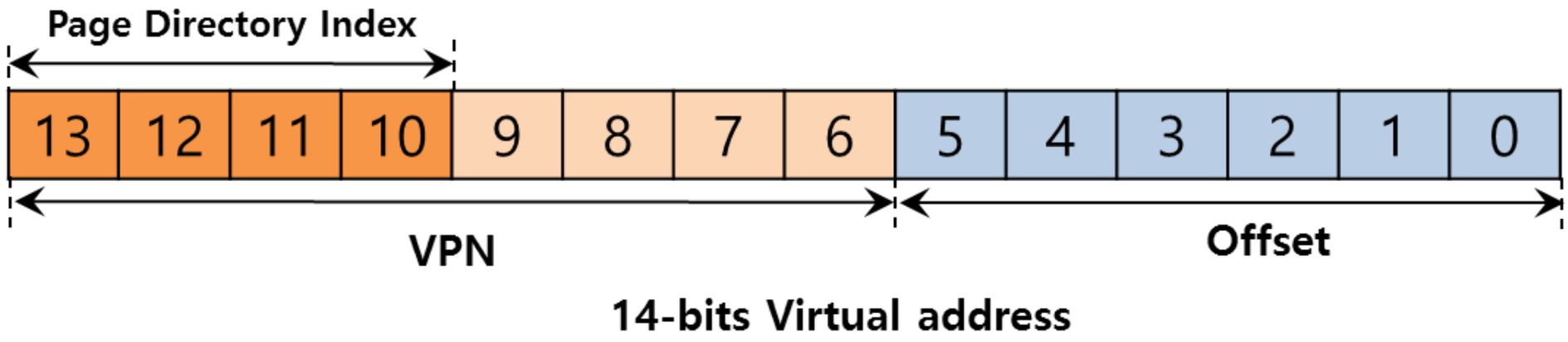


A Detailed Multi-Level Example: Page Directory IDX

The page directory needs one entry per page of the page table

- it has 16 entries.

The page-directory entry is invalid → Raise an exception (The access is invalid)



Other Approaches

2a. Inverted Pagetables (hashtable)

2b. Segmented Pagetables

2c. Multi-level Pagetables

- Page the page tables (PTs over PTs)

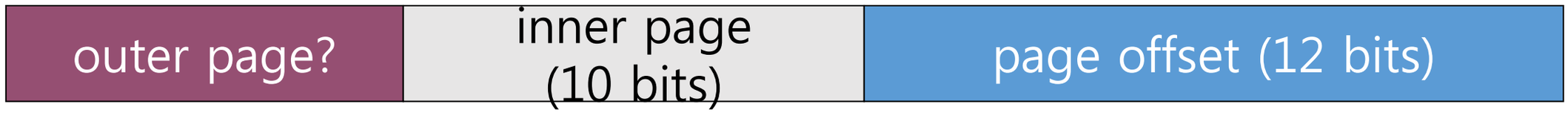
2d. Multi-level Pagetables

- Page the pagetables of page tables… (PTs over Pts over PTs)

Problem with 2 levels?

Problem: page directories (outer level) may not fit in a page

64-bit address:



Solution:

- Split page directories into pieces
- Use another page dir to refer to the page dir pieces.

How large is virtual address space with 4 KB pages, 4 byte PTEs, each page table fits in page given 1, 2, 3 levels?

4KB / 4 bytes → 1K entries per level

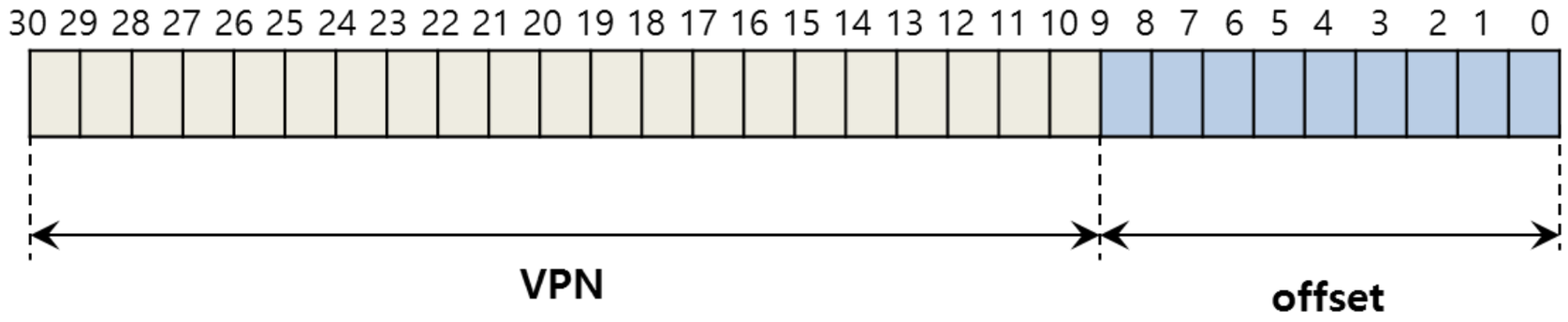
1 level: 1K * 4K = **2^22** = 4 MB

2 levels: 1K * 1K * 4K = **2^32** ≈ 4 GB

3 levels: 1K * 1K * 1K * 4K = **2^42** ≈ 4 TB

More than Two Level

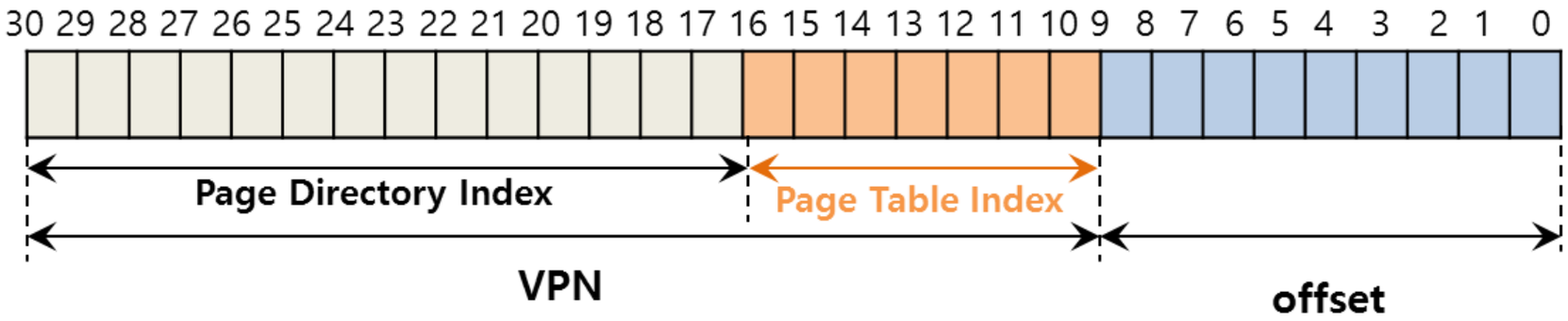
In some cases, a deeper tree is possible.



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.



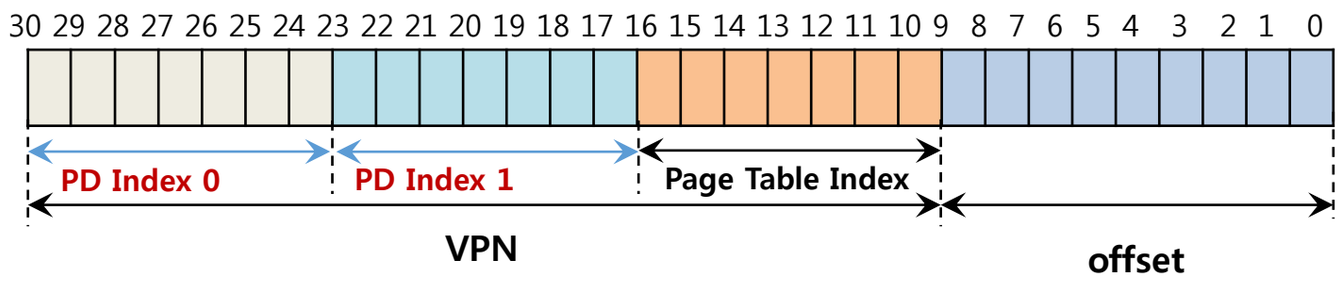
Flag	Detail
Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit
Page entry per page	128 PTEs

→ $\log_2 128 = 7$

More than Two Level : Page Directory

If our page directory has 2^{14}entries, it spans not one page but 128.

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 transalation 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

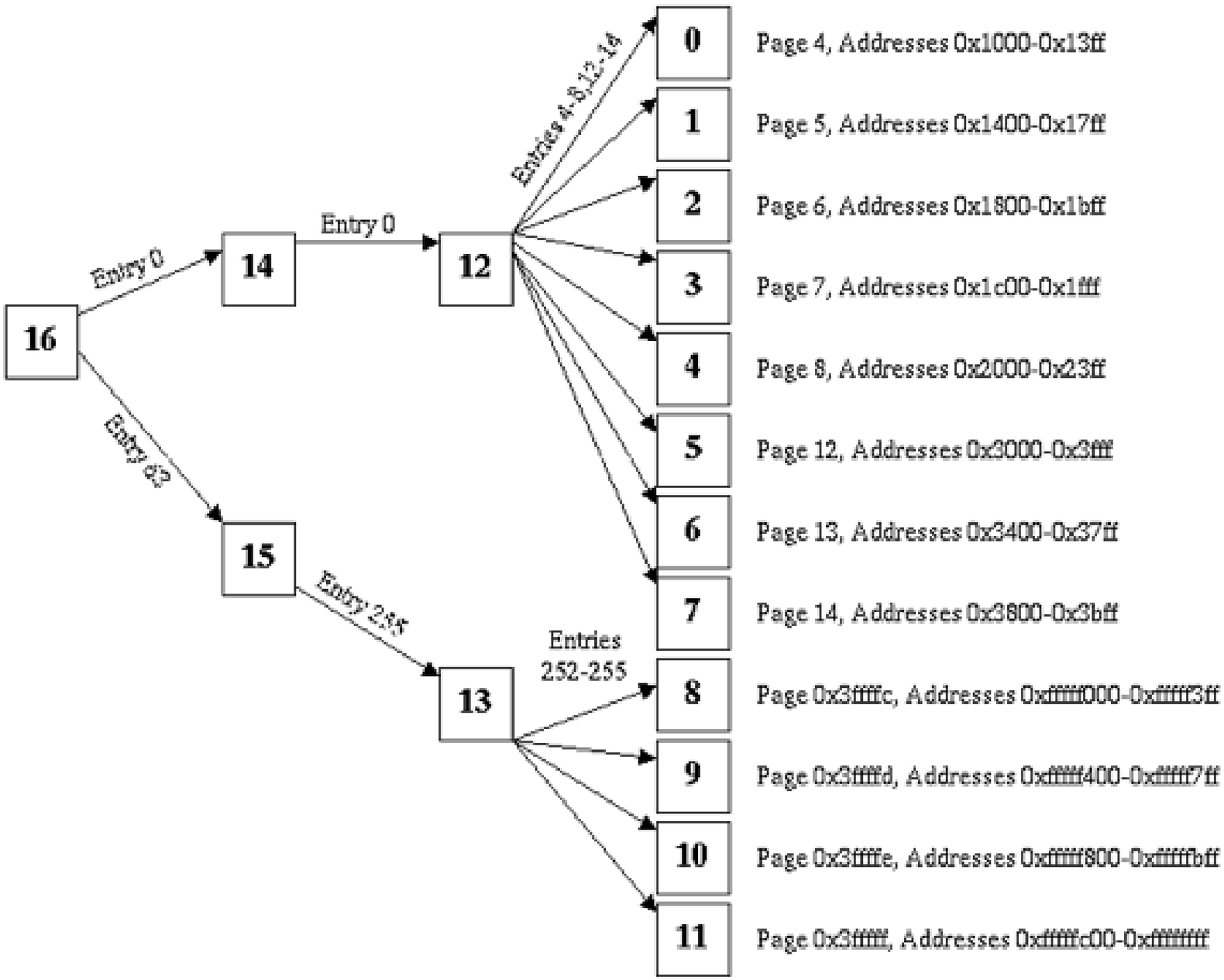
Multi-level Page Table Control Flow

```
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()
```



http://web.eecs.utk.edu/~mbeck/classes/cs560/560/oldtests/t2/2003/Answers.html

QUIZ: FULL SYSTEM WITH TLBS

On TLB miss: lookups with more levels more expensive

How much does a miss cost?

Assume 3-level page table
Assume 256-byte pages
Assume 16-bit addresses
Assume ASID of current process is 211

ASID	VPN	PFN	Valid
211	0xbb	0x91	1
211	0xff	0x23	1
122	0x05	0x91	1
211	0x05	0x12	0

How many physical accesses for each instruction? (Ignore previous ops changing TLB)

(a) 0xAA10: movl 0x1111, %edi

0xaa: (TLB miss -> 3 for addr trans) + 1 instr fetch

0x11: (TLB miss -> 3 for addr trans) + 1 movl

Total: 8

(b) 0xBB13: addl $0x3, %edi

0xbb: (TLB hit -> 0 for addr trans) + 1 instr fetch from 0x9113

Total: 1

(c) 0x0519: movl %edi, 0xFF10

0x05: (TLB miss -> 3 for addr trans) + 1 instr fetch

0xff: (TLB hit -> 0 for addr trans) + 1 movl into 0x2310

Total: 5

What about TLBs?

Lookups in multiple levels more expensive.

How much does a miss cost?

Time/Space tradeoffs.

Summary: Better Page Tables

Problem:
Simple linear page tables require too much contiguous memory

Many options for efficiently organizing page tables

If OS traps on TLB miss, OS can use any data structure

- Inverted page tables (hashing)

If Hardware handles TLB miss, page tables must follow specific format

- Multi-level page tables used in x86 architecture
- Each page table fits within a page

Next Topic:
What if desired address spaces do not fit in physical memory?