

**Department of Physics,  
Computer Science & Engineering**

CPSC 410 – Operating Systems I

# Virtualizing Memory: Smaller Page TAbles

Keith Perkins

Adapted from “CS 537 Introduction to Operating Systems”  
Arpaci-Dusseau

# 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)

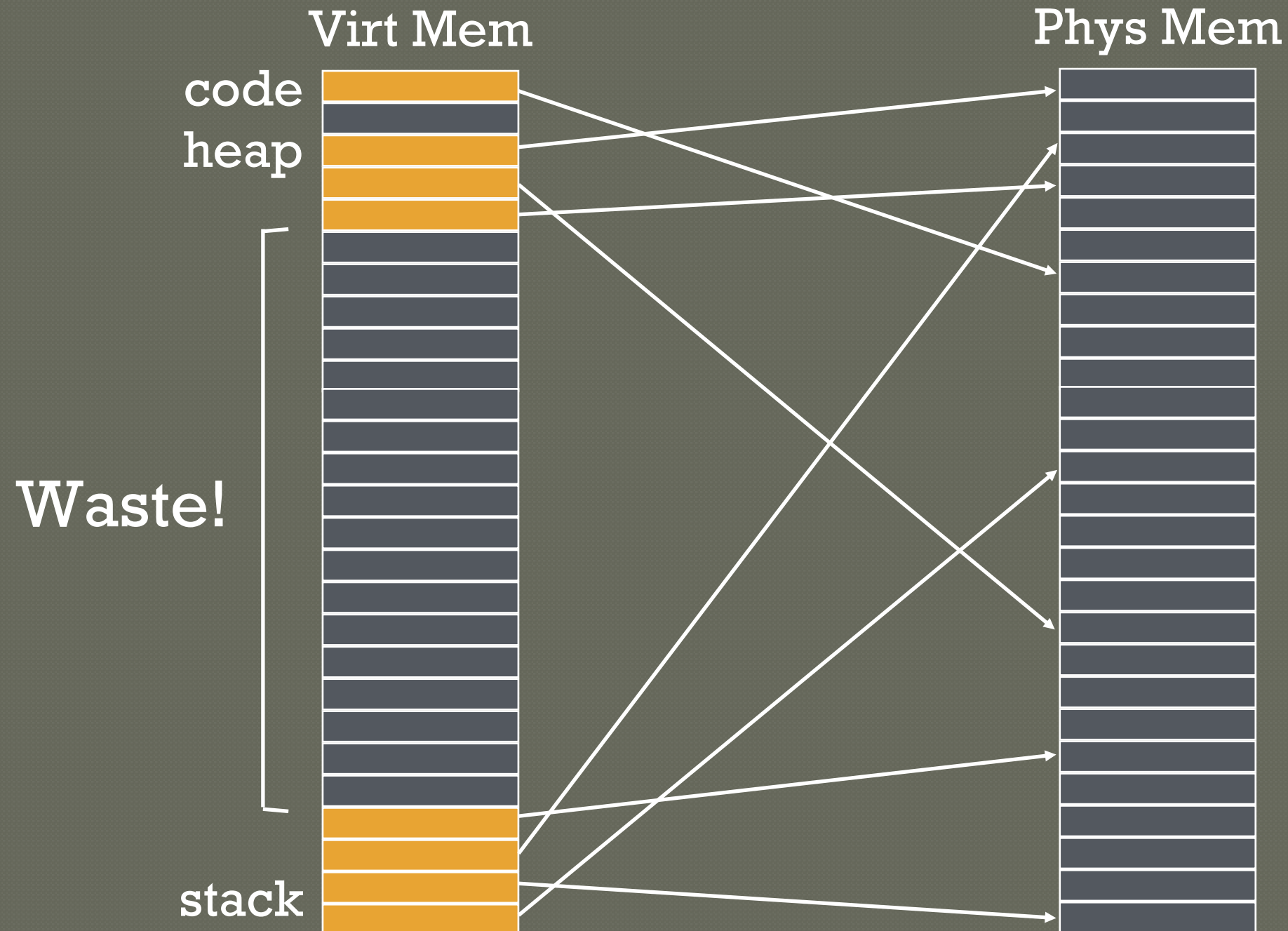
# QUIZ: How big are page Tables?

---

1. PTE's are **2 bytes**, and **32** possible virtual page numbers  
 $32 * 2 \text{ bytes} = 64 \text{ bytes}$
2. PTE's are **2 bytes**, virtual addrs are **24 bits**, pages are **16 bytes**  
 $2 \text{ bytes} * 2^{(24 - \lg 16)} = 2^{21} \text{ bytes (2 MB)}$
3. PTE's are **4 bytes**, virtual addrs are **32 bits**, and pages are **4 KB**  
 $4 \text{ bytes} * 2^{(32 - \lg 4K)} = 2^{22} \text{ bytes (2 MB)}$
4. PTE's are **4 bytes**, virtual addrs are **64 bits**, and pages are **4 KB**  
 $4 \text{ bytes} * 2^{(64 - \lg 4K)} = 2^{54} \text{ bytes}$

How big is each page table?

# Why ARE Page Tables so Large?



# Many invalid PT entries

	PFNvalid	prot
10	1	1-1
-	0	-
23	1	1W-
-	0	-
-	0	-
-	0	-
-	0	-
...many more invalid...		
-	0	-
-	0	-
-	0	-
-	0	-
28	1	1W-
4	1	1W-

## how to avoid storing these?



# Avoid simple linear Page Table

---

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

# Approach 1: Inverted Page TAbble

---

## Inverted Page Tables

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

Naïve approach:

Search through data structure  $\langle \text{ppn}, \text{vpn} + \text{asid} \rangle$  to find match

- Too much time to search entire table

Better: Find possible matches entries by hashing  $\text{vpn} + \text{asid}$

- 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



# Other Approaches

---

1. Inverted Pagetables
2. Segmented Pagetables
3. Multi-level Pagetables
  - Page the page tables
  - Page the pagetables of page tables...

# 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

# 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

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

## 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

...	0x001000
0x01f	
0x011	
0x003	
0x02a	
0x013	
...	0x002000
0x00c	
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 lecture)

## 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

=  $2^{18} * 4 \text{ bytes} = 2^{20} \text{ bytes} = 1 \text{ MB!!!}$



# Other Approaches

---

1. Inverted Pagetables
2. Segmented Pagetables
3. Multi-level Pagetables
  - Page the page tables
  - Page the pages of page tables...

# 3) Multilevel Page Tables

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

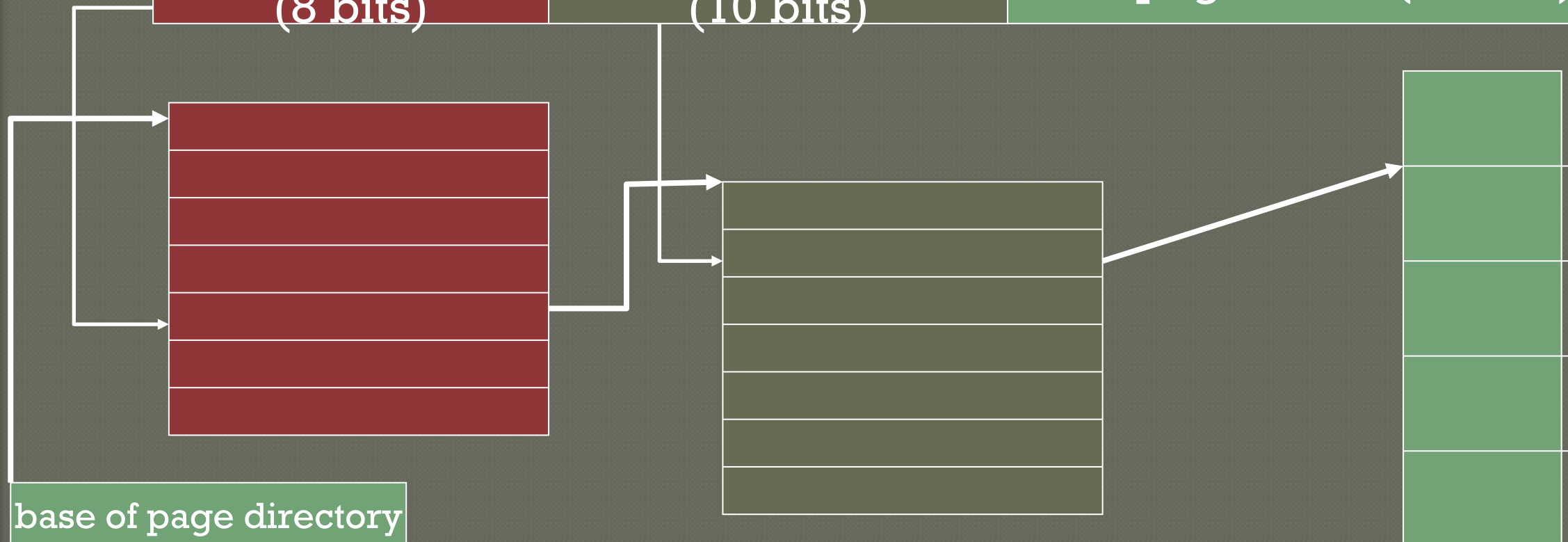
- Creates multiple levels of page tables; outer level “page directory”
- Only allocate page tables for pages in use
- Used in x86 architectures (hardware can walk known structure)

30-bit address:

outer page  
(8 bits)

inner page  
(10 bits)

page offset (12 bits)



# Quiz: Multilevel

page directory

page of PT (@PPN:0x3)

page of PT (@PPN:0x92)

PPN	value	PPN	value	PPN	value	
0x3	1	0x10	1	0x92	0	
-	0	0x23	1	0x93	0	
-	0	0x36	0	0x94	0	translate 0x01ABC
-	0	0x49	0	0x95	0	
-	0	0x5C	1	0x96	0	<b>0x23ABC</b>
-	0	0x6F	1	0x97	0	translate 0x00000
-	0	0x80	0	0x98	0	
-	0	0x93	0	0x99	0	<b>0x10000</b>
-	0	0xA6	0	0x9A	0	
-	0	0xB9	0	0x9B	0	
-	0	0xC2	0	0x9C	0	
-	0	0xD5	0	0x9D	0	translate 0xFEED0
-	0	0xE8	0	0x9E	0	
-	0	0xFB	0	0x9F	0	<b>0x55ED0</b>
-	0	0x106	0	0xA0	0	
-	0	0x119	0	0xA1	0	
-	0	0x12C	0	0xA2	0	
-	0	0x13F	0	0xA3	0	
-	0	0x152	0	0xA4	0	
-	0	0x165	0	0xA5	1	
-	0	0x178	0	0xA6	1	
0x92	1	0x18B	0	0xA7	0	
		0x19E	0	0xA8	0	
		0x1B1	0	0xA9	0	
		0x1C4	0	0xAA	0	
		0x1D7	0	0xAB	0	
		0x1EA	0	0xAC	0	
		0x1FD	0	0xAD	0	
		0x210	0	0xAE	0	
		0x223	0	0xAF	0	
		0x236	0	0xB0	0	
		0x249	0	0xB1	0	
		0x25C	0	0xB2	0	
		0x26F	0	0xB3	0	
		0x280	0	0xB4	0	
		0x293	0	0xB5	0	
		0x2A6	0	0xB6	0	
		0x2B9	0	0xB7	0	
		0x2CC	0	0xB8	0	
		0x2DF	0	0xB9	0	
		0x2F2	0	0xBA	0	
		0x305	0	0xBB	0	
		0x318	0	0xBC	0	
		0x32B	0	0xBD	0	
		0x33E	0	0xBE	0	
		0x351	0	0xBF	0	
		0x364	0	0xC0	0	
		0x377	0	0xC1	0	
		0x38A	0	0xC2	0	
		0x39D	0	0xC3	0	
		0x3B0	0	0xC4	0	
		0x3C3	0	0xC5	0	
		0x3D6	0	0xC6	0	
		0x3E9	0	0xC7	0	
		0x3FC	0	0xC8	0	
		0x40F	0	0xC9	0	
		0x422	0	0xCA	0	
		0x435	0	0xCB	0	
		0x448	0	0xCC	0	
		0x45B	0	0xCD	0	
		0x46E	0	0xCE	0	
		0x481	0	0xCF	0	
		0x494	0	0xD0	0	
		0x4A7	0	0xD1	0	
		0x4BA	0	0xD2	0	
		0x4CD	0	0xD3	0	
		0x4E0	0	0xD4	0	
		0x4F3	0	0xD5	0	
		0x506	0	0xD6	0	
		0x519	0	0xD7	0	
		0x52C	0	0xD8	0	
		0x53F	0	0xD9	0	
		0x552	0	0xDA	0	
		0x565	0	0xDB	0	
		0x578	0	0xDC	0	
		0x58B	0	0xDD	0	
		0x59E	0	0xDE	0	
		0x5B1	0	0xDF	0	
		0x5C4	0	0xE0	0	
		0x5D7	0	0xE1	0	
		0x5EA	0	0xE2	0	
		0x5FD	0	0xE3	0	
		0x610	0	0xE4	0	
		0x623	0	0xE5	0	
		0x636	0	0xE6	0	
		0x649	0	0xE7	0	
		0x65C	0	0xE8	0	
		0x66F	0	0xE9	0	
		0x682	0	0xEA	0	
		0x695	0	0xEB	0	
		0x6A8	0	0xEC	0	
		0x6BB	0	0xED	0	
		0x6CE	0	0xEE	0	
		0x6E1	0	0xEF	0	
		0x6F4	0	0xF0	0	
		0x707	0	0xF1	0	
		0x71A	0	0xF2	0	
		0x72D	0	0xF3	0	
		0x740	0	0xF4	0	
		0x753	0	0xF5	0	
		0x766	0	0xF6	0	
		0x779	0	0xF7	0	
		0x78C	0	0xF8	0	
		0x79F	0	0xF9	0	
		0x7B2	0	0xFA	0	
		0x7C5	0	0xFB	0	
		0x7D8	0	0xFC	0	
		0x7EB	0	0xFD	0	
		0x7FE	0	0xFE	0	
		0x801	0	0xFF	0	
		0x814	0			
		0x827	0			
		0x83A	0			
		0x84D	0			
		0x860	0			
		0x873	0			
		0x886	0			
		0x899	0			
		0x8AC	0			
		0x8BF	0			
		0x8D2	0			
		0x8E5	0			
		0x8F8	0			
		0x90B	0			
		0x91E	0			
		0x931	0			
		0x944	0			
		0x957	0			
		0x96A	0			
		0x97D	0			
		0x990	0			
		0x9A3	0			
		0x9B6	0			
		0x9C9	0			
		0x9DC	0			
		0x9EF	0			
		0xA02	0			
		0xA15	0			
		0xA28	0			
		0xA3B	0			
		0xA4E	0			
		0xA61	0			
		0xA74	0			
		0xA87	0			
		0xA9A	0			
		0xAAD	0			
		0xAB0	0			
		0xAC3	0			
		0xAD6	0			
		0xAE9	0			
		0xAF2	0			
		0xB05	0			
		0xB18	0			
		0xB2B	0			
		0xB3E	0			
		0xB51	0			
		0xB64	0			
		0xB77	0			
		0xB8A	0			
		0xB9D	0			
		0xBA8	0			
		0xBBB	0			
		0xBCE	0			
		0xBD1	0			
		0xBE4	0			
		0xBF7	0			
		0xC0A	0			
		0xC1D	0			
		0xC30	0			
		0xC43	0			
		0xC56	0			
		0xC69	0			
		0xC7C	0			
		0xC8F	0			
		0xCA2	0			
		0xCB5	0			
		0xCC8	0			
		0xCDB	0			
		0xCE6	0			
		0xCF9	0			
		0xD0C	0			
		0xD1F	0			
		0xD32	0			
		0xD45	0			
		0xD58	0			
		0xD6B	0			
		0xD7E	0			
		0xD91	0			
		0xDA4	0			
		0xDB7	0			
		0xDC0	0			
		0xDD3	0			
		0xDE6	0			
		0xDF9	0			
		0xE0C	0			
		0xE1F	0			
		0xE32	0			
		0xE45	0			
		0xE58	0			
		0xE6B	0			
		0xE7E	0			
		0xE91	0			
		0xEA4	0			
		0xEB7	0			
		0xEC0	0			
		0xED3	0			
		0xEE6	0			
		0xEF9	0			
		0xF0C	0			
		0xF1F	0			
		0xF32	0			
		0xF45	0			
		0xF58	0			
		0xF6B	0			
		0xF7E	0			
		0xF91	0			
		0xFA4	0			
		0xFB7	0			
		0xFC0	0			
		0xFD3	0			
		0xFE6	0			
		0xFF9	0			

20-bit address:

outer page  
(4 bits)

inner page  
(4 bits)

page offset (12 bits)

# QUIZ: Address format for multilevel Paging

30-bit address:

outer page	inner page	page offset (12 bits)
------------	------------	-----------------------

## How should logical address be structured?

- How many bits for each paging level?

### Goal?

- Each page table fits within a page
- $\text{PTE size} * \text{number PTE} = \text{page size}$ 
  - Assume PTE size = 4 bytes
  - Page size =  $2^{12}$  bytes = 4KB
  - $2^2 \text{ bytes} * \text{number PTE} = 2^{12} \text{ bytes}$
  - $\rightarrow \text{number PTE} = 2^{10}$
- $\rightarrow \# \text{ bits for selecting inner page} = 10$

### Remaining bits for outer page:

- $30 - 10 - 12 = 8 \text{ bits}$

# Problem with 2 levels?

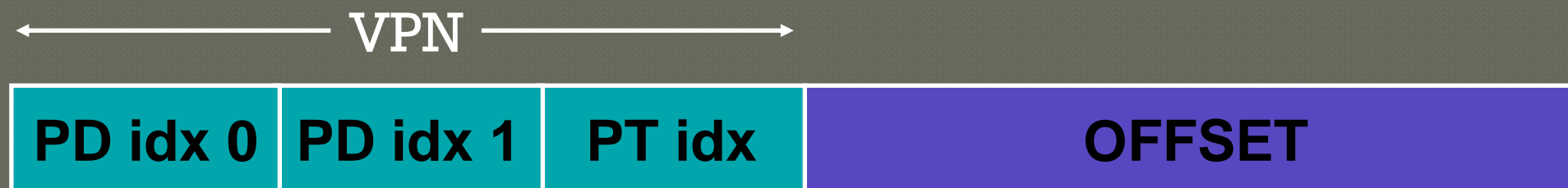
Problem: page directories (outer level) may not fit in a page

**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:  $1K * 4K = 2^{22} = 4 \text{ MB}$

2 levels:  $1K * 1K * 4K = 2^{32} \approx 4 \text{ GB}$

3 levels:  $1K * 1K * 1K * 4K = 2^{42} \approx 4 \text{ TB}$



# QUIZ: FULL SYSTEM WITH TLBS

On TLB miss: lookups with more levels more expensive

How much does a miss cost?

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

Assume 3-level page table

Assume 256-byte pages

Assume 16-bit addresses

Assume ASID of current process is 211

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

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

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

Total: 8

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

Total: 1

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

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

Total: 5



# 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?

•