

**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?
  - segmentation + paging, multilevel page tables
- ⦿ What happens on a TLB miss?

# Disadvantages of Paging

---

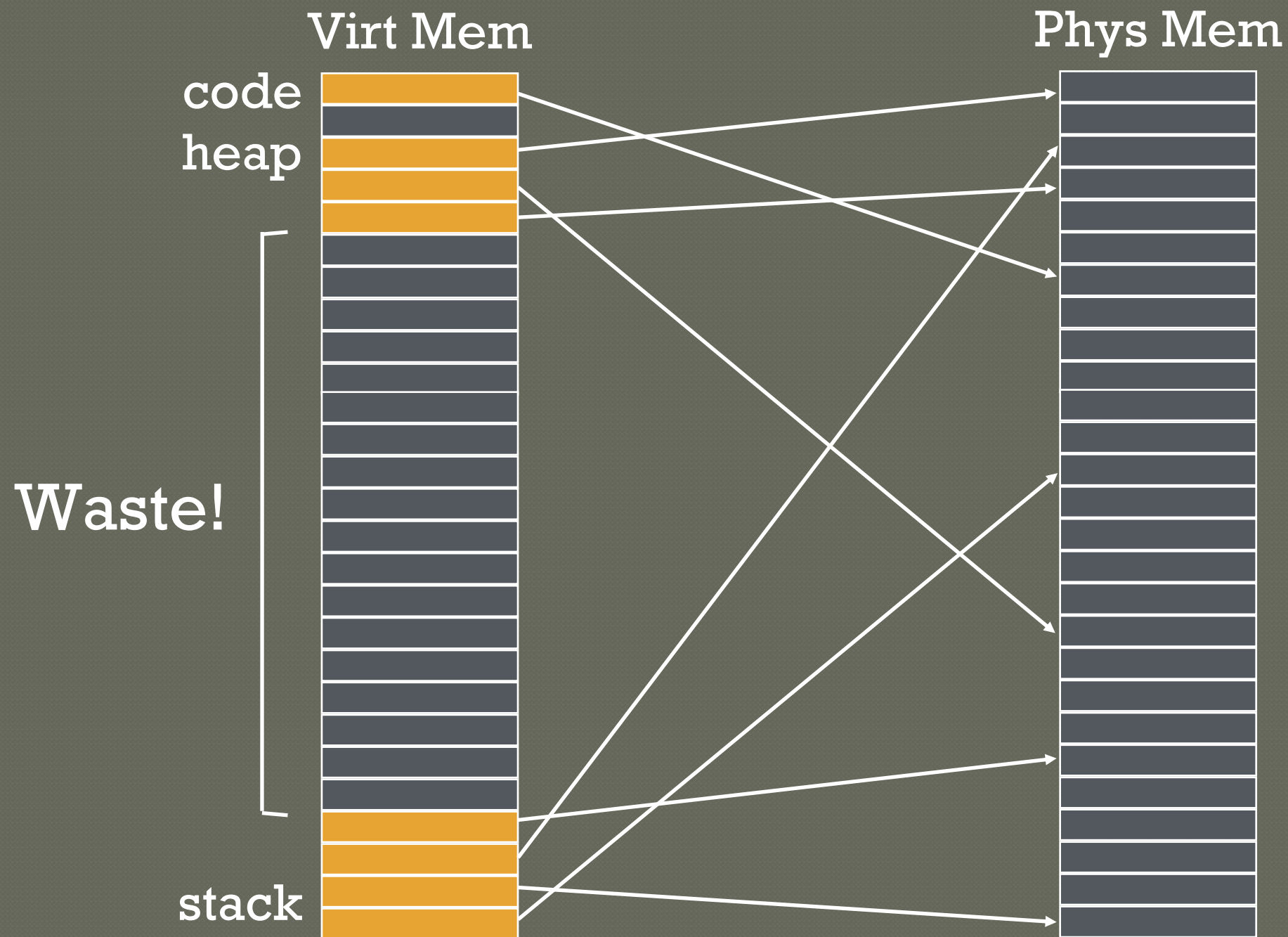
1. Additional memory reference to look up in page table
  - Very inefficient
  - Page table must be stored in memory
  - MMU stores only base address of page table
  - **Avoid extra memory reference for lookup with TLBs (previous lecture)**
2. Storage for page tables may be substantial
  - Simple page 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**  
 $2 \text{ bytes} * 2^{(24 - \lg 16)} = 2 * 2^{21} \text{ bytes} (2 \text{ MB})$
3. PTE's are **4 bytes**, virtual addrs are **32 bits**, and pages are **4 KB**  
 $4 \text{ bytes} * 2^{(32 - \lg 4K)} = 4 * 2^{20} \text{ bytes} (4 \text{ MB})$
4. PTE's are **4 bytes**, virtual addrs are **64 bits**, and pages are **4 KB**  
 $4 \text{ bytes} * 2^{(64 - \lg 4K)} = 4 * 2^{(64-12)} = 2^{54} \text{ byte}$

How big is each page table?

# Why ARE Page Tables so Large?



Need 1 entry per physical frame  
But you are using very few of the entries



# Many invalid PT entries

## Format of linear page tables:

how to avoid  
storing these?

	PFN valid	protection
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-



Invalid pages are not used  
but still are in page table

# 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

# Other Approaches

---

1. Segmented Pagedtables
2. Multi-level Pagedtables
  - Page the page tables
  - Page the page tables 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: **err bound=0**  
 0x010424 write: **err 0x004 + 424 > 0xff**  
 0x210014 write: **err 0x003 + 014 > 0x0f**  
 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 needed 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. Segmented Pagedtables
2. Multi-level Pagedtables
  - 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

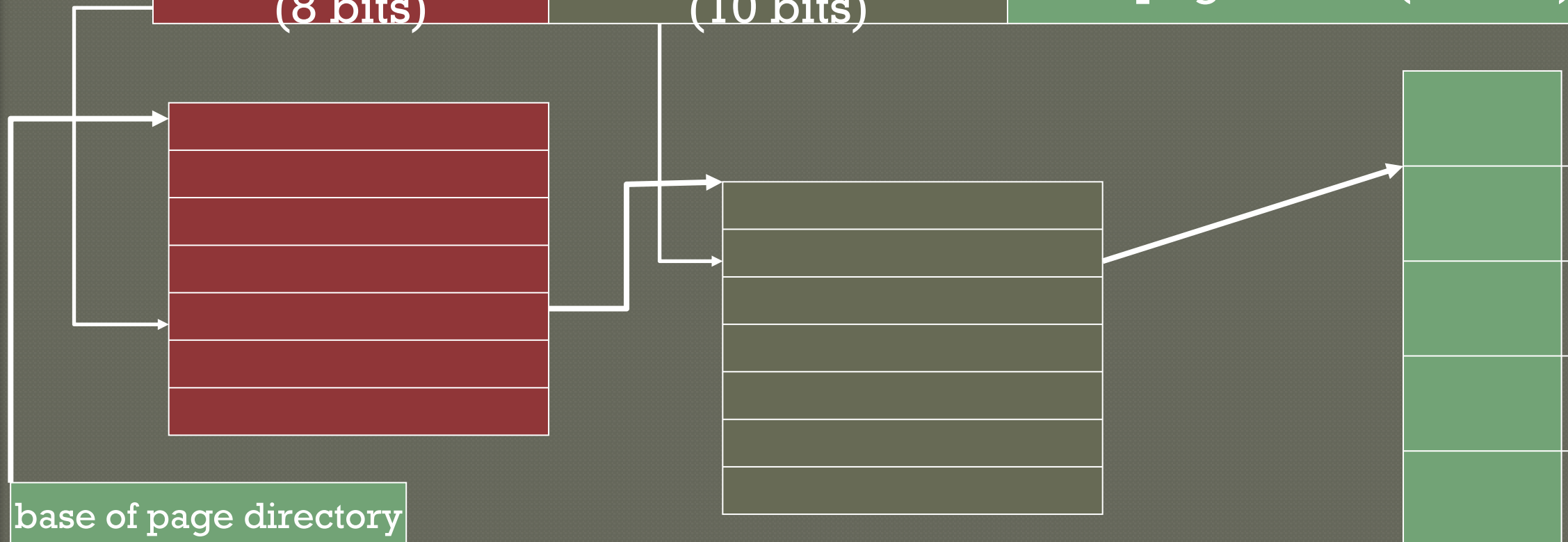
- Creates multiple levels of page tables; outer level “page directory”
- Only allocate page tables for pages in use
- Used in x86 architectures (hardware can walk known structure)

30-bit address:

outer page  
(8 bits)

inner page  
(10 bits)

page offset (12 bits)



8-bit address:

outer page  
(2 bits)

inner page  
(2 bits)

page offset (4 bits)

base of  
page directory

2 bits used to index  
into this table

2 bits used to index  
into these tables

valid

00	1	physical address of inner page table 0
01	0	---
10	0	---
11	1	physical address of inner page table 3

valid

00	1	0x1
01	1	0xF
10	1	0x3
11	0	

valid

00	0	---
01	0	---
10	0	---
11	1	0x2

Physical memory

0x00	
0x10	P0
0x20	P3
0x30	P2
0xF0	P1

Page table has 12 rows not 16

Virtual address    Physical address

0x01    in P0 offset 1    0x11

0x43    => 0100 0011 => invalid

# Quiz: Multilevel

page directory		page of PT (@PPN:0x3)		page of PT (@PPN:0x92)		
PPN	valid	PPN	valid	PPN	valid	
0x3	1	0x10	1	-	0	
-	0	0x23	1	-	0	
-	0	-	0	-	0	translate 0x01ABC
-	0	-	0	-	0	
-	0	0x80	1	-	0	0x23ABC
-	0	0x59	1	-	0	translate 0x00000
-	0	-	0	-	0	
-	0	-	0	-	0	0x10000
-	0	-	0	-	0	
-	0	-	0	-	0	
-	0	-	0	-	0	translate 0xFEED0
-	0	-	0	-	0	
-	0	-	0	-	0	0x55ED0
-	0	-	0	0x55	1	
0x92	1	-	0	0x45	1	

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?

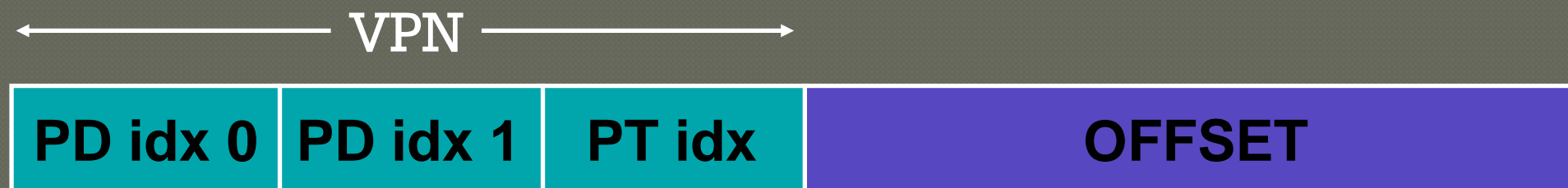
Problem: page directories (outer level) may not fit in a page

**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)

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

**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

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?

•