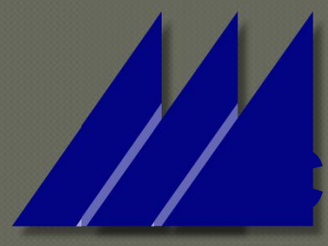


Department of Physics,
Computer Science & Engineering

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

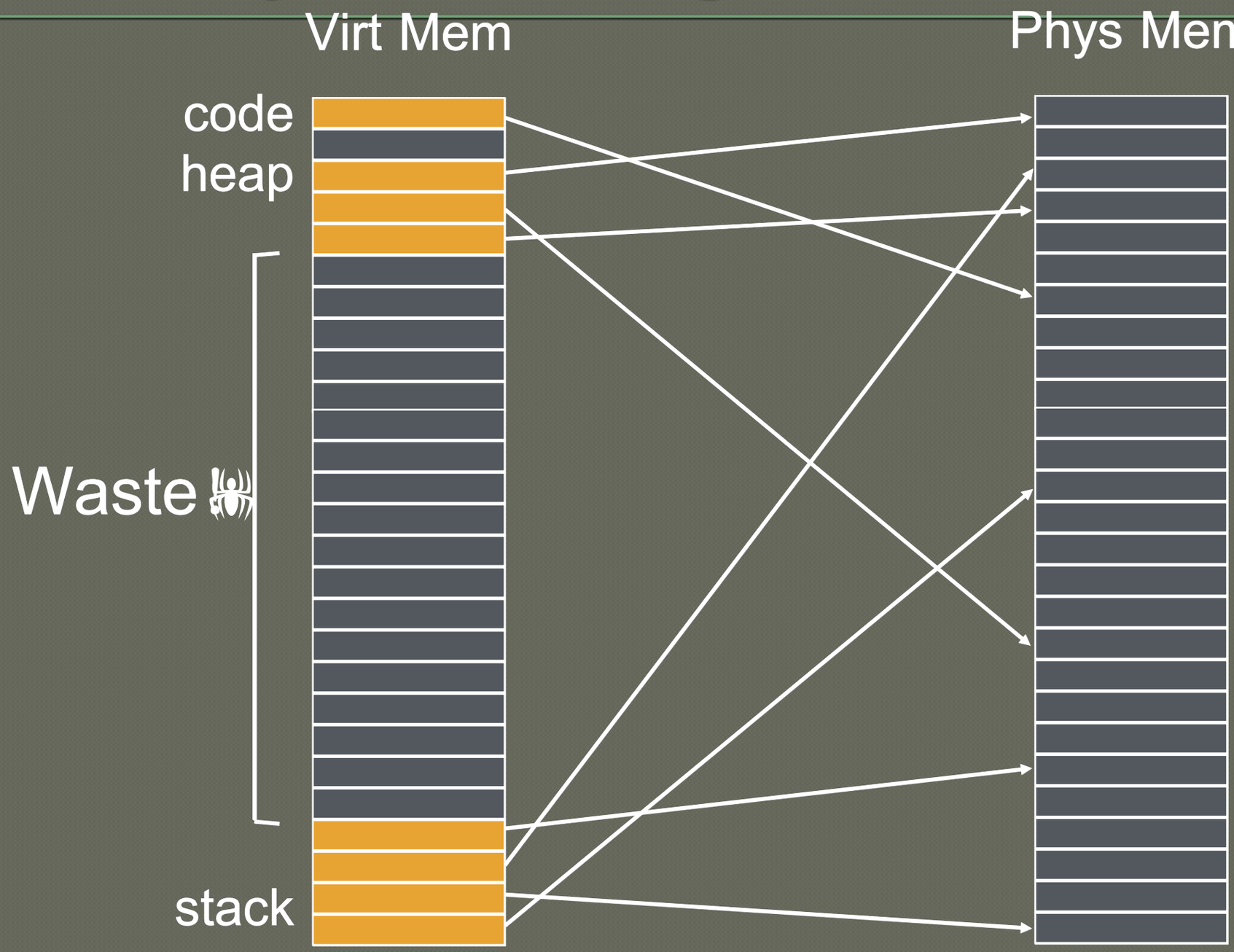
1. Additional memory reference to look up in page table
   - Very inefficient
   - Page table must be stored in memory
   - MMU stores only base address of page table (processor tells it which page table to use)
   - **Avoid extra memory reference for lookup with TLBs (previous lecture)**
2. Storage 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 **bytes** = 64 **bytes**
2. PTE's are **2 bytes**, virtual addrs are **24 bits**, pages are **16 bytes**
   2 **bytes** * 2**^**$(24$ **– lg** $16) = 2$*(**s^**$21)$ **bytes** $(2$ **MB**$)$
3. PTE's are **4 bytes**, virtual addrs are **32 bits**, and pages are **4 KB**
   4 **bytes** * 2**^**$(32$ **– lg** 4**K**$) = 4$*2**^**20 **bytes** $(4$ **MB**$)$
4. PTE's are **8 bytes**, virtual addrs are **64 bits**, and pages are **4 KB**
   8 **bytes** * 2**^**$(64$ **– lg** 4**K**$) = 2$**^**3*2**^**$(64-12)=2$**^**55 **bytes**

How big is each page table?

Why ARE Page Tables so Large?

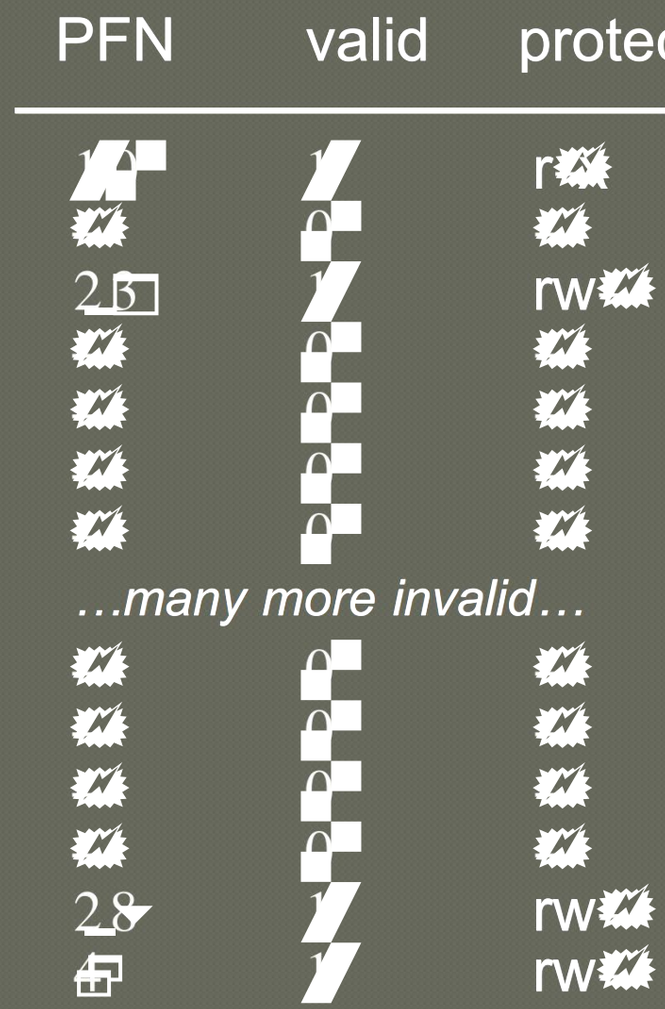


Need 1 entry per physical frame

But you are using very few of the entries

Many invalid PT entries

Format of linear page tables:



BTW where is the code ■

how to avoid storing these ■

Invalid pages are not used but still are in page table

Avoid 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

Approach 1: 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

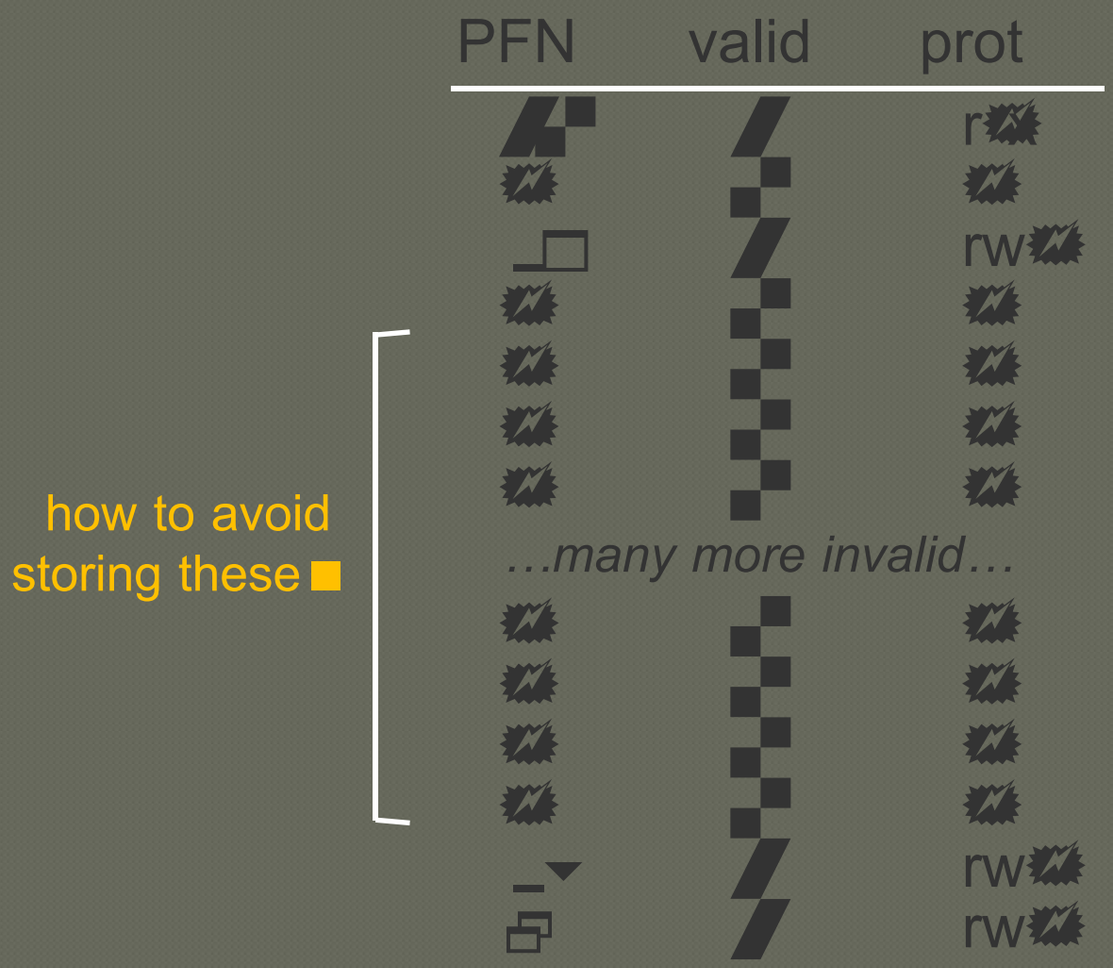
Managing inverted page table requires software-controlled TLB

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

Approaches

1. Segmented Pagetables
2. Multi-level Pagetables
   - Page the page tables
   - Page the page tables of page tables…

valid PTEs are Contiguous



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

How did OS avoid allocating holes in phys memory ■

Segmentation

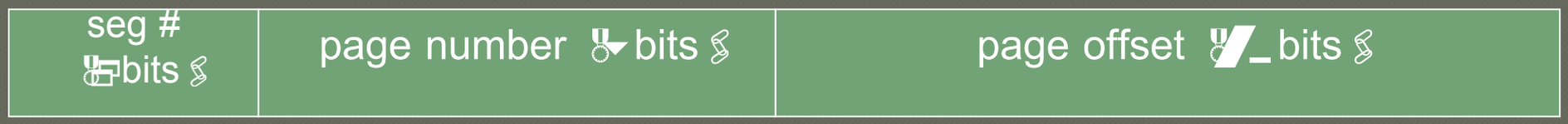
Combine Paging and Segmentation

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



Implementation

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

Quiz: Paging and Segmentation

seg # bits	page number bits	page offset bits

seg	base	bounds	R W
0			
1			
2			

read:
read:
read:
write:

read:

Quiz: Paging and Segmentation

seg # [illegible] bits	page number [illegible] bits	page offset [illegible] bits

seg	base	bounds	R W
[illegible]	[illegible]	[illegible]xff	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]

[illegible] read: [illegible]

[illegible] read: [illegible]

[illegible] read: [illegible]

[illegible] write: [illegible]

[illegible] read: [illegible]

Go to seg 0 get base
0x002000
02<=0xff ?
Yes go to entry here
And build address
0x004070

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]a
[illegible]
[illegible]
[illegible]e
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

[illegible]

[illegible]

Quiz: Paging and Segmentation

seg # bits	page number bits	page offset bits

seg	base	bounds	R W
		ff	

read

read

read

write

err exceeded bounds

read

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
    - Worst case [illegible] bounds register [illegible]
    - Each page table is [illegible]
      - [illegible] Number of entries [illegible] size of each entry
      - [illegible] Number of pages [illegible] bytes
      - [illegible] bytes [illegible] bytes [illegible] MB

Other Approaches

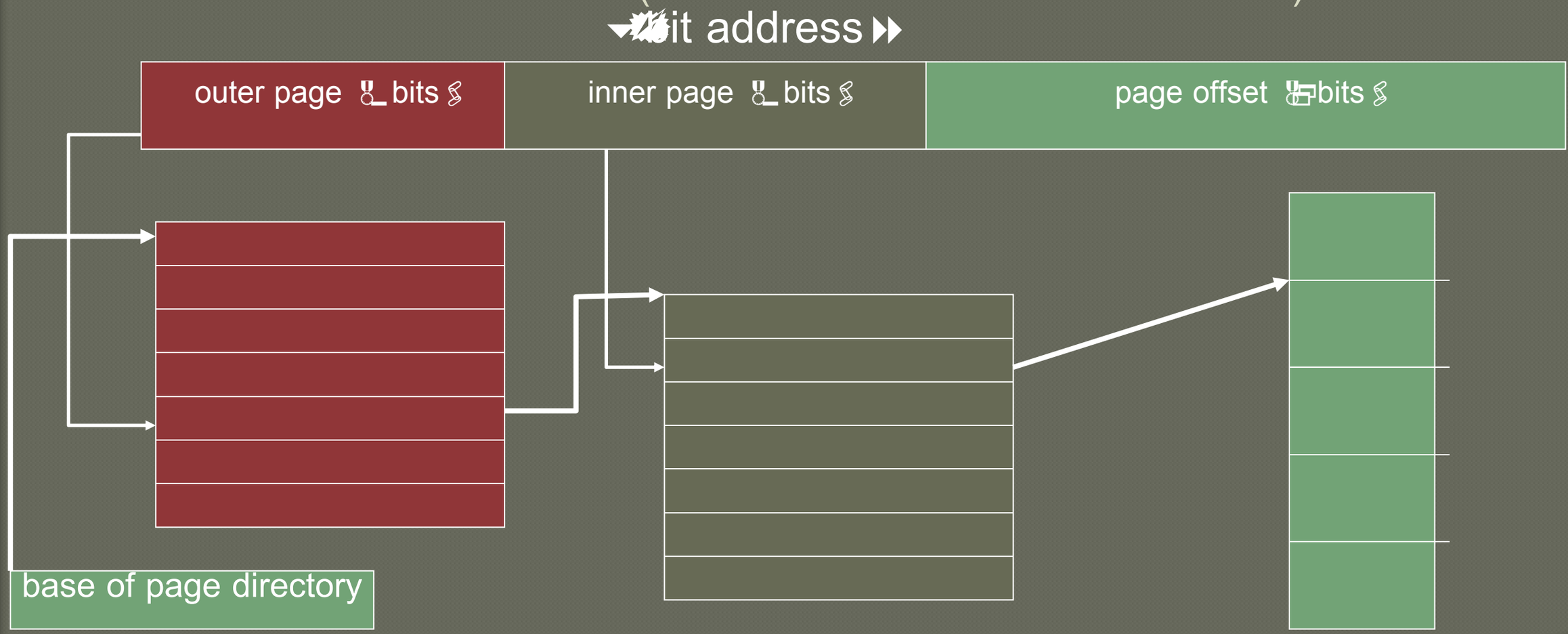
1. Segmented Pagetables
2. Multi-level Pagetables
   - Page the page tables
   - Page the pages of page tables…

Multilevel Page Tables

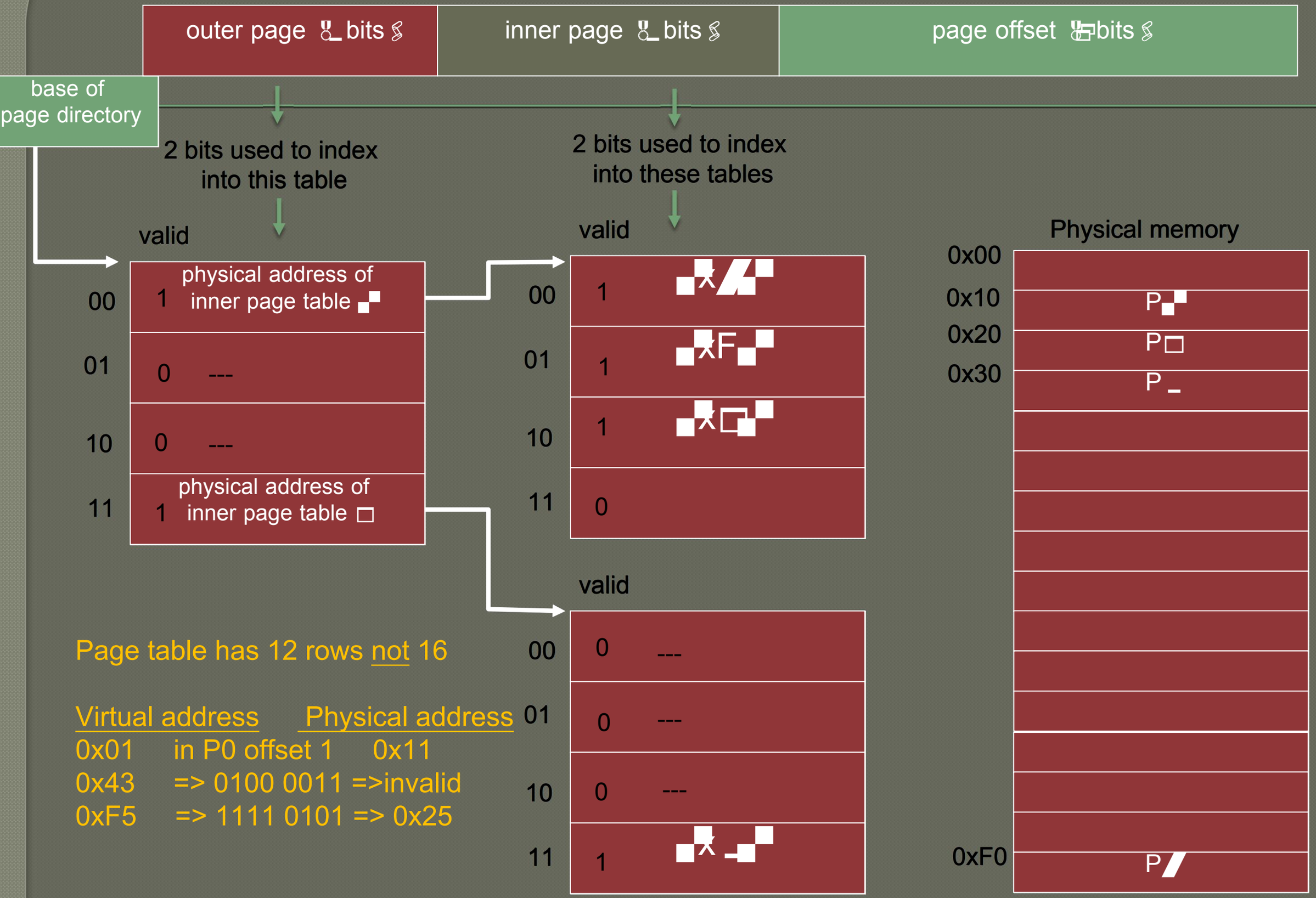
Goal: Allow page tables to be allocated non-contiguously

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)



8-bit address:



Page table has 12 rows not 16

Virtual address		Physical address
0x01	in P0 offset 1	0x11
0x43	=> 0100 0011	=>invalid
0xF5	=> 1111 0101	=> 0x25

Quiz: Multilevel

QUIZ: Address format for multilevel Paging

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 ← Want entire page table to fit in 4kb
  - 2^2 bytes * number PTE = 2^12 bytes
  - number PTE = 2^10 ← can have 1024 4byte rows
- # bits for selecting inner page = 10

Remaining bits for outer page:

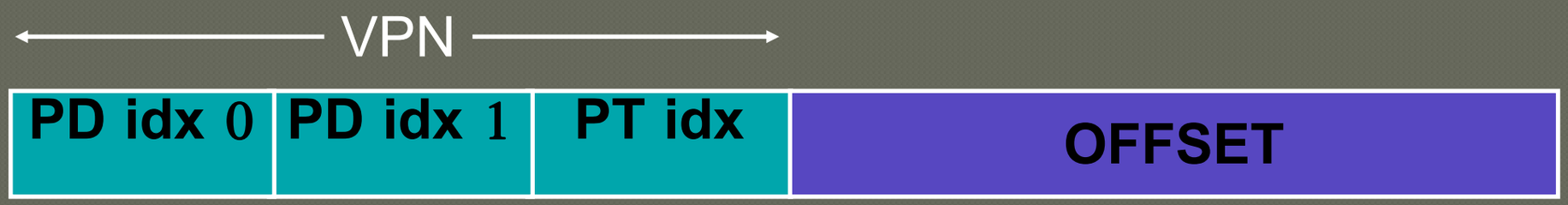
- 30 – 10 – 12 = 8 bits

Problem with 2 levels?

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

Solution:

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



How large is virtual address space with KB pages byte PTEs each page table fits in page given levels Assume bits level K

KB bytes K entries per level

level K K 2^22 MB

levels K K K 2^32 ≈ GB

levels K K K K 2^42 ≈ 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
	bb		
	ff		

Assume level page table

Assume byte pages bits

Assume it addresses so bits for levels

Assume ASID of current process is

How many physical accesses for each instruction ignore previous ops changing TLB

AA movl edi

aa TLB miss for addr trans instr fetch

Total

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

BB addl edi

bb TLB hit for addr trans instr fetch from

Total

movl edi FF

TLB miss for addr trans instr fetch

Total

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

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?

.

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?