RUTGERS UNIVERSITY
Computer Sciences Department

CS 416 Operating Systems Design

Sudarsun Kannan

Virtualizing Memory: Smaller Page TAbles

Questions answered in this lecture:

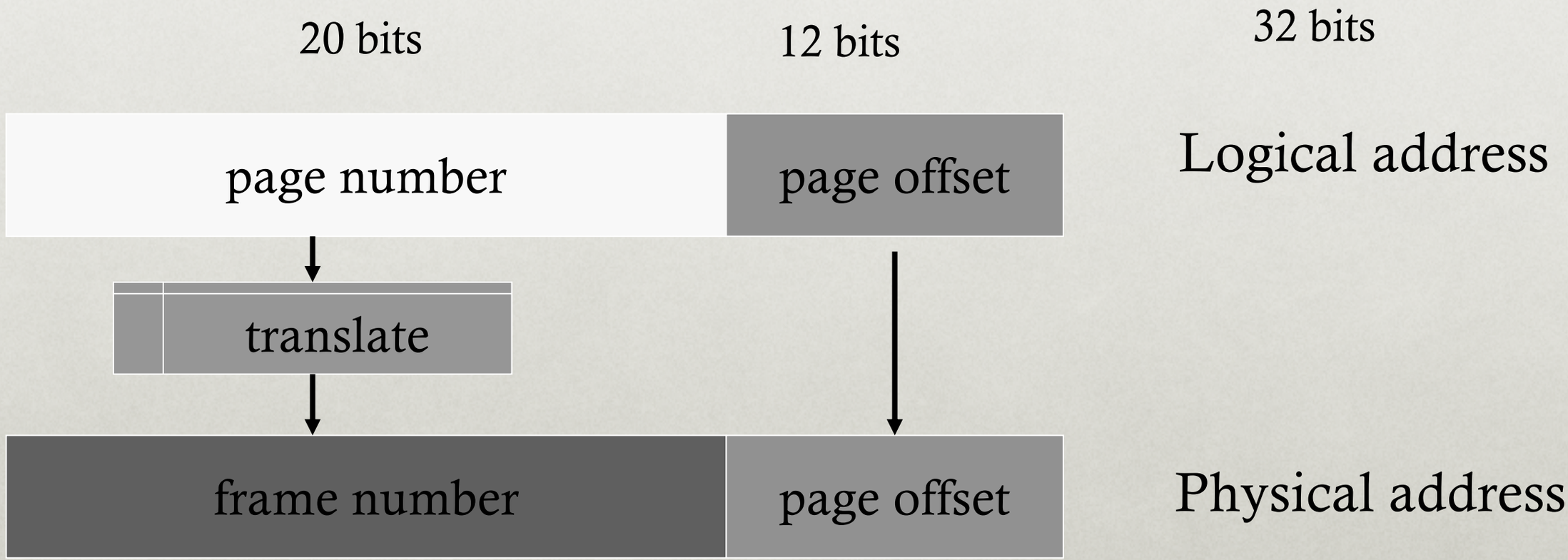
- Review: What are problems with paging?
- Review: How large can page tables be?
- How can large page tables be avoided with different techniques?
  - Inverted page tables, segmentation + paging
  - Multilevel page tables – Current systems

Announcements

- Reading for today:
  - Chapter 18-20
- Project 2 posted, due March 11th
  - Attend recitations to understand about contexts
  - Start early; you need to write a significant amount of good code
  - Make your code as modular as possible (interviewers look for these aspects)
  - Add lot of comments to your code (a great practice to follow)
  - Format your code well https://www.kernel.org/doc/html/v4.10/process/coding-style.html
- Midterm – March 5th
  - Sample mid-term will be posted one week before the exam
- No Class on 02/24
  - No recitations on 02/25 and 02/27

Translation of Page Addresses

- How to translate logical address to physical address?
  - High-order bits of address designate page number
  - Low-order bits of address designate offset within page



No addition needed; just append bits correctly…

How does format of address space determine number of pages and size of pages?

Quiz: Address Format

Given known page size, how many bits are needed in address to specify offset in page?

Page Size	Low Bits (offset)
16 bytes	4
1 KB	10
1 MB	20
512 bytes	9
4 KB	12

Quiz: Address Format

Given number of bits in virtual address and bits for offset, how many bits for virtual page number?

Page Size	Low Bits (offset)	Virt Addr Bits	High Bits (vpn)
16 bytes	4	10	6
1 KB	10	20	10
1 MB	20	32	12
512 bytes	9	16	5
4 KB	12	32	20

7

Correct?

Quiz: Address Format

Given number of bits for vpn, how many virtual pages can there be in an address space?

Page Size	Low Bits (offset)	Virt Addr Bits	High Bits (vpn)	Virt Pages
16 bytes	4	10	6	64
1 KB	10	20	10	1 K
1 MB	20	32	12	4 K
512 bytes	9	16	5	32
4 KB	12	32	20	1 MB

Where Are Pagetables Stored?

How big is a typical page table?

- assume **32-bit** address space
- assume 4 KB pages
- assume 4 byte entries

Final answer: 2 ^ (32 - log(4KB)) * 4 = **4 MB**

- Page table size = Num entries * size of each entry
- Num entries = num virtual pages = 2^(bits for vpn)
- Bits for vpn = 32– number of bits for page offset
  = 32 – lg(4KB) = 32 – 12 = 20
- Num entries = 2^20 = 1 MB
- Page table size = Num entries * 4 bytes = 4 MB

Implication: Store each page table in memory

- Hardware finds page table base with register (e.g., CR3 on x86)

What happens on a context-switch?

- Change contents of page table base register to newly scheduled process
- Save old page table base register in PCB of descheduled process

Other PT info

What other info is in pagetable entries besides translation?

- valid bit
- protection bits
- present bit (needed later)
- reference bit (needed later)
- dirty bit (needed later)

Pagetable entries are just bits stored in memory

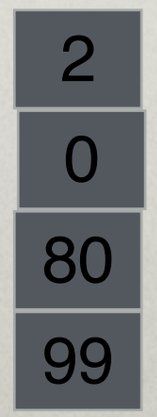
- Agreement between hw and OS about interpretation

Memory Accesses with Pages

```
0x0010: movl 0x1100, %edi
0x0013: addl $0x3, %edi
0x0019: movl %edi, 0x1100
```

Assume PT is at phys addr 0x5000
Assume PTE's are 4 bytes
Assume 4KB pages
How many bits for offset? 12

Simplified view of page table



Old: How many mem refs with segmentation?

5 (3 instrs, 2 movl)

Physical Memory Accesses with Paging?

1) Fetch instruction at logical addr 0x0010; vpn?
   - Access page table to get ppn for vpn 0
   - Mem ref 1: 0x5000
   - Learn vpn 0 is at ppn 2
   - Fetch instruction at 0x2010 (Mem ref 2)

Exec, load from logical addr 0x1100; vpn?

- Access page table to get ppn for vpn 1
- Mem ref 3: 0x5004
- Learn vpn 1 is at ppn 0
- Movl from 0x0100 into reg (Mem ref 4)

Pagetable is slow!!! Doubles memory references

Advantages of Paging

No external fragmentation

- Any page can be placed in any frame in physical memory

Fast to allocate and free

- Alloc: No searching for suitable free space
- Free: Doesn't have to coallesce with adjacent free space
- Just use bitmap to show free/allocated page frames

Simple to swap-out portions of memory to disk (later lecture)

- Page size matches disk block size
- Can run process when some pages are on disk
- Add "present" bit to PTE

Disadvantages of Paging

Internal fragmentation: Page size may not match size needed by process

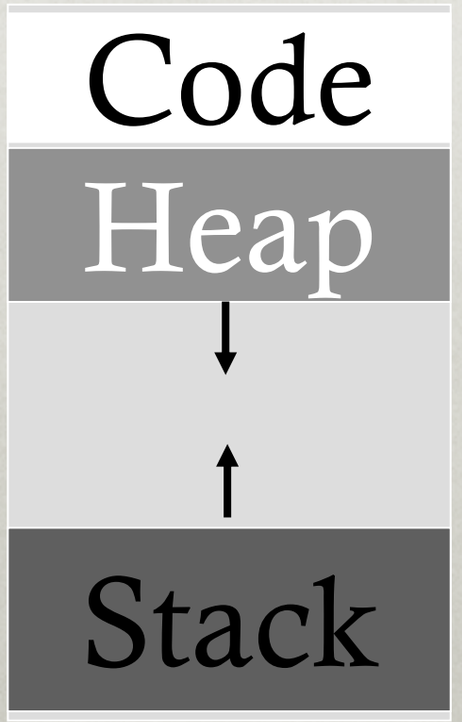
- Wasted memory grows with larger pages
- **Tension?**

Additional memory reference to page table --> Very inefficient

- Page table must be stored in memory
- MMU stores only base address of page table
- Solution: Add TLBs (future lecture)

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
- Page tables must be allocated contiguously in memory
- Solution: Combine paging and segmentation (future lecture)



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 (next 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) = **2^21 bytes** (2 MB)

3. PTE's are **4 bytes**, virtual addrs are **32 bits**, and pages are **4 KB**

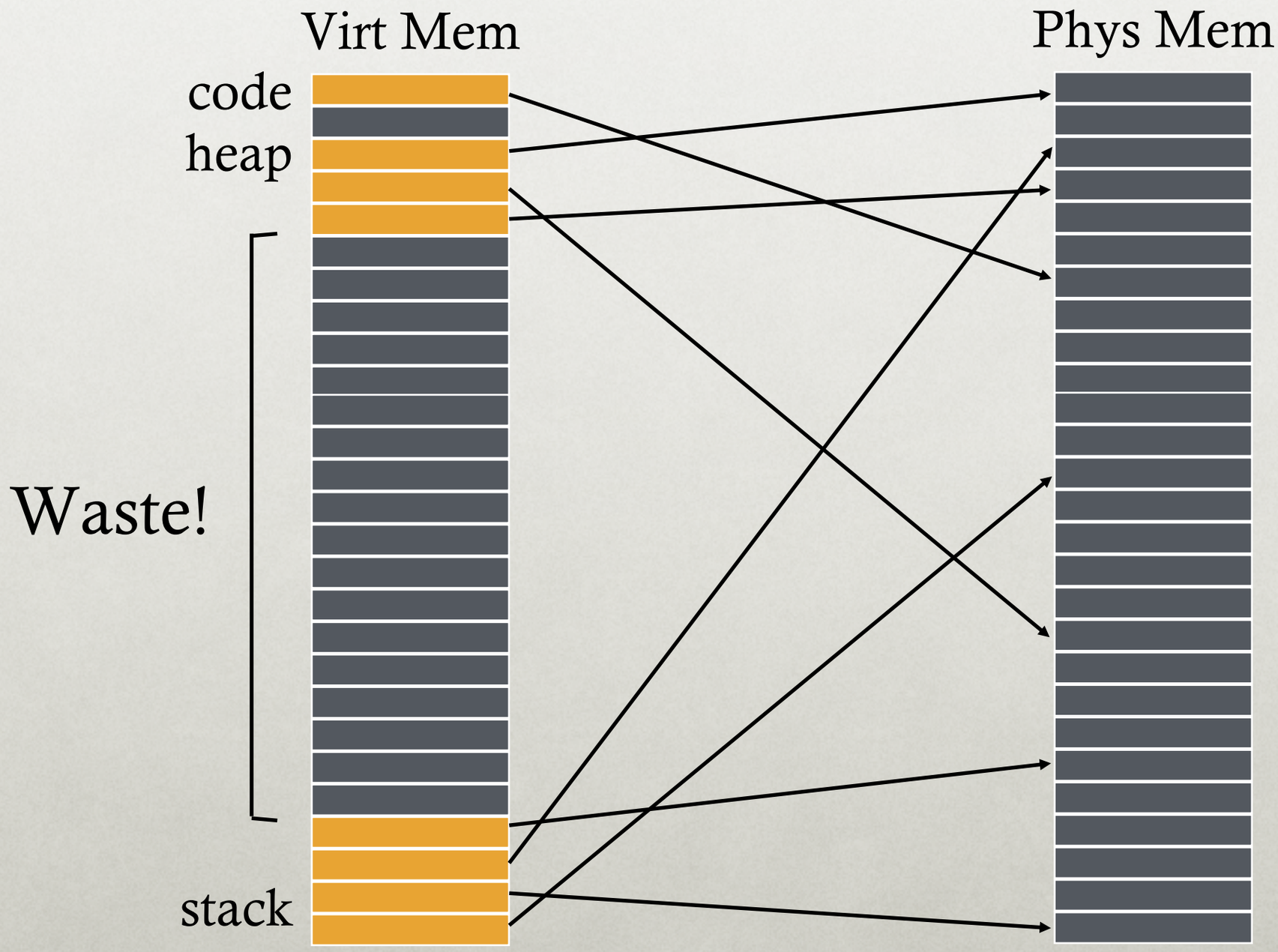
   4 bytes * 2^(32 – lg 4K) = **2^22 bytes** (2 MB)

4. PTE's are **4 bytes**, virtual addrs are **64 bits**, and pages are **4 KB**

How big is each page table?

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

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?

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 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~~

Other Approaches

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

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 (255)	1 0
1	0x000000	0x00	0 0
2	0x001000	0x0f (15)	1 1

0x002070 read: 0x004070

0x202016 read: 0x003016

0x104c84 read: error

0x010424 write: error

0x210014 write: error

0x203568 read: 0x02a568

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

0x001000

0x002000

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 (discussed soon)

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!!!

Other Approaches

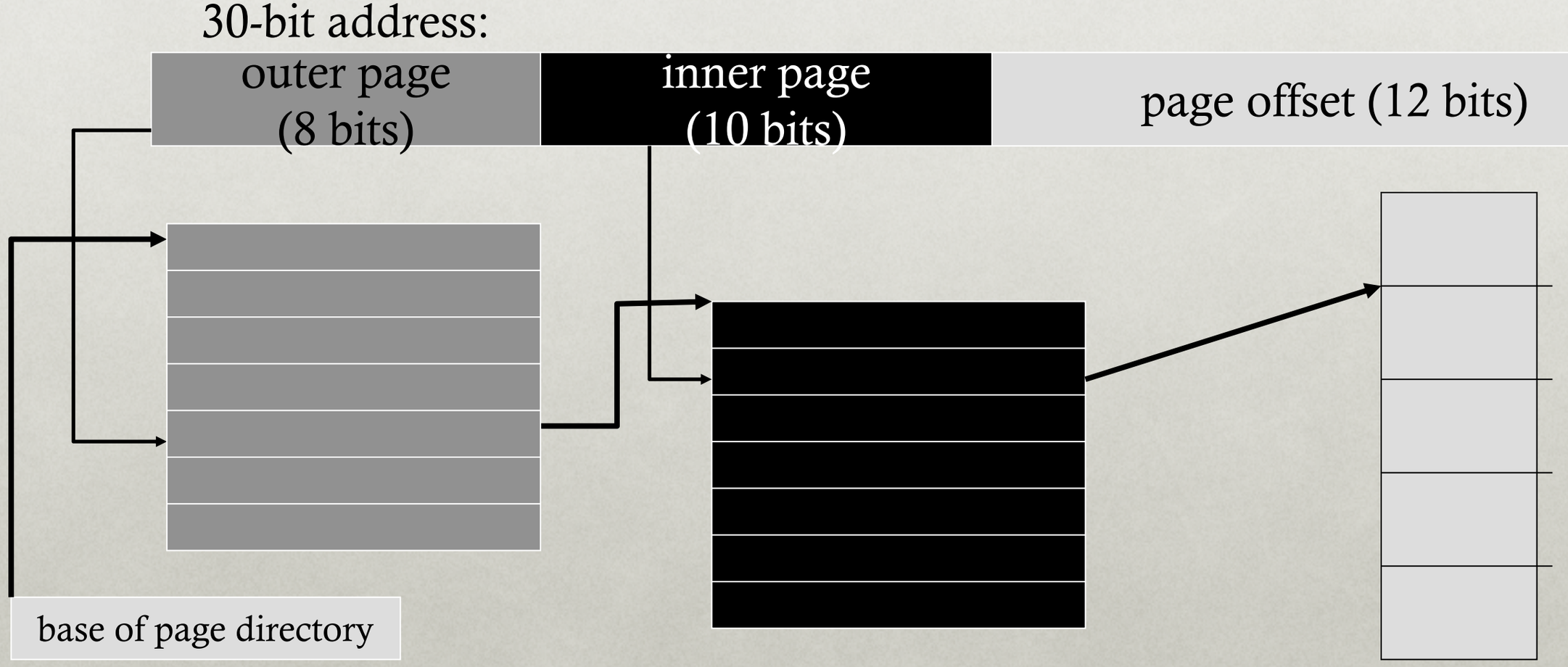
1. Segmented Pagetables
2. Multi-level Pagetables
   - Page the page tables
   - Page the pagetables of page tables…
3. Inverted Pagetables

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)



Quiz: Multilevel

page directory

VPN	PPN	valid
0	0x3	1
1	-	0
2	-	0
-	-	0
-	-	0
-	-	0
-	-	0
-	-	0
-	-	0
-	-	0
-	-	0
-	-	0
-	-	0
-	-	0
15	0x92	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
-	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

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

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

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

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?