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**

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

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**

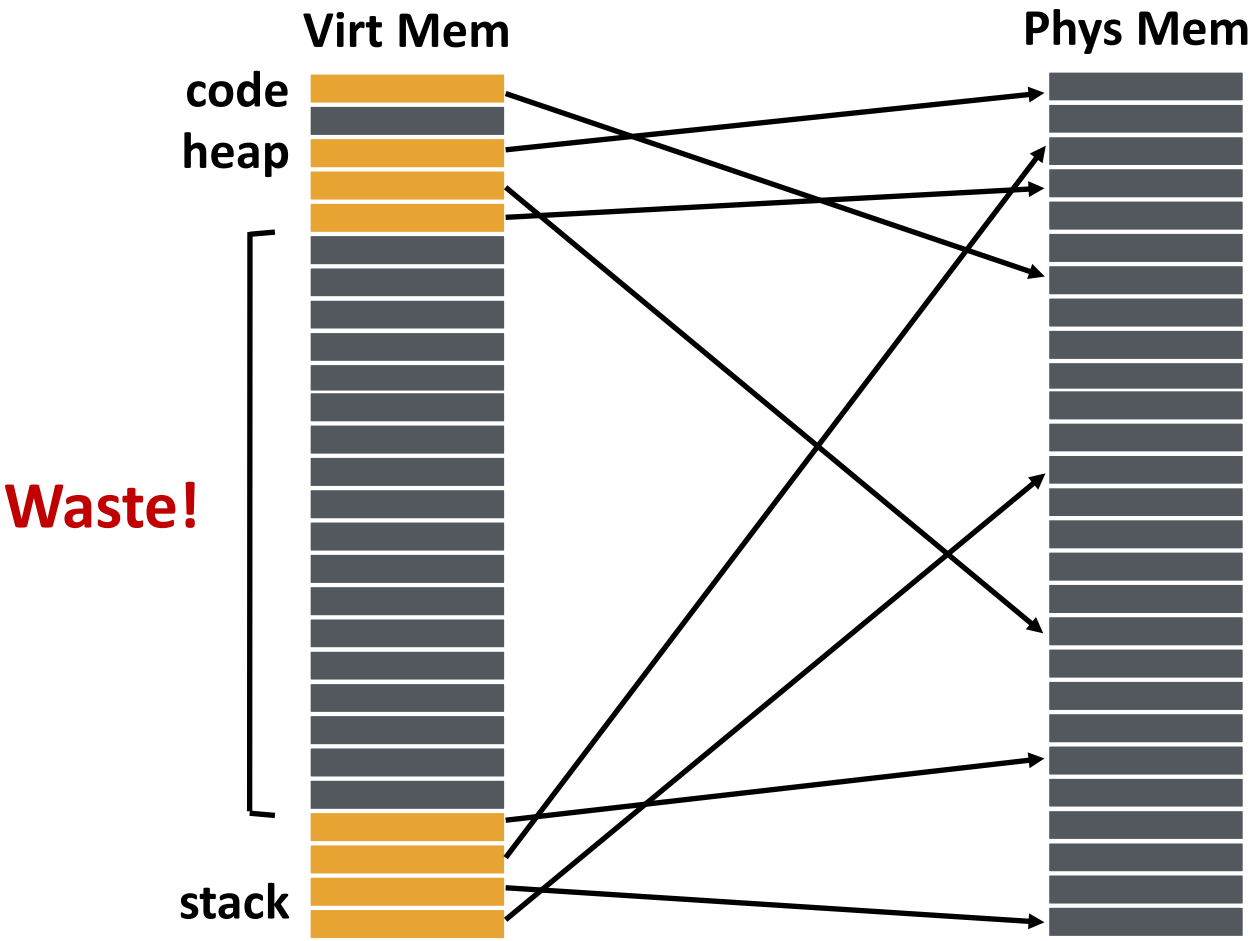
   4 bytes * 2^(32 – lg 4K) = 2^22 bytes (4 MB)

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?

Why are Page Tables so Large?



Many Invalid PT Entries

Note: Code segment cannot be written
exception: JIT & Virus

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 one big array**

- **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

- **Page table is a data structure, requirement of software-managed and hardware–managed TLB are different**

Approaches

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

Approach 1: Inverted Page Table

Never used in implementation

- **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
  - build a hash table, and search through the linked list
  - PowerPC is one example of such architecture
  - demands software-managed TLB
- **For hardware-controlled TLB, need well-defined, simple approach**

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

how to avoid storing these?

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-

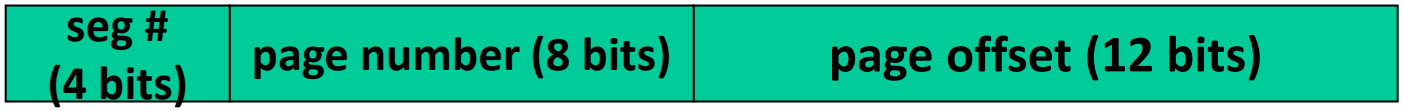
- Note “hole” in addr space: valids vs. invalids are clustered
- How did OS avoid allocating holes in phys memory?

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 track base (physical address) and bounds of page table for that segment (number of pages)**

Paging and Segmentation

Note: Paging after segmentation so that the pages are dense in storage and make it easier to expand segment (just allocate more pages)

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

segment table

seg	Base of page table	Bounds (#page)	R W
0	0x002000	0xff	1 0
1	0x000000	0x00	0 0
2	0x001000	0x0f	1 1

virtual		physical
0x002070	read:	0x004070
0x202016	read:	0x003016
0x104c84	read:	error for bounds
0x010424	write:	error for RW
0x210014	write:	error for #page
0x203568	read:	0x02a568

page table

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

0x001000

Note: But Page table must be continuously stored, which could be problematic.

0x002000

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 bytes = 2^20 bytes = 1 MB

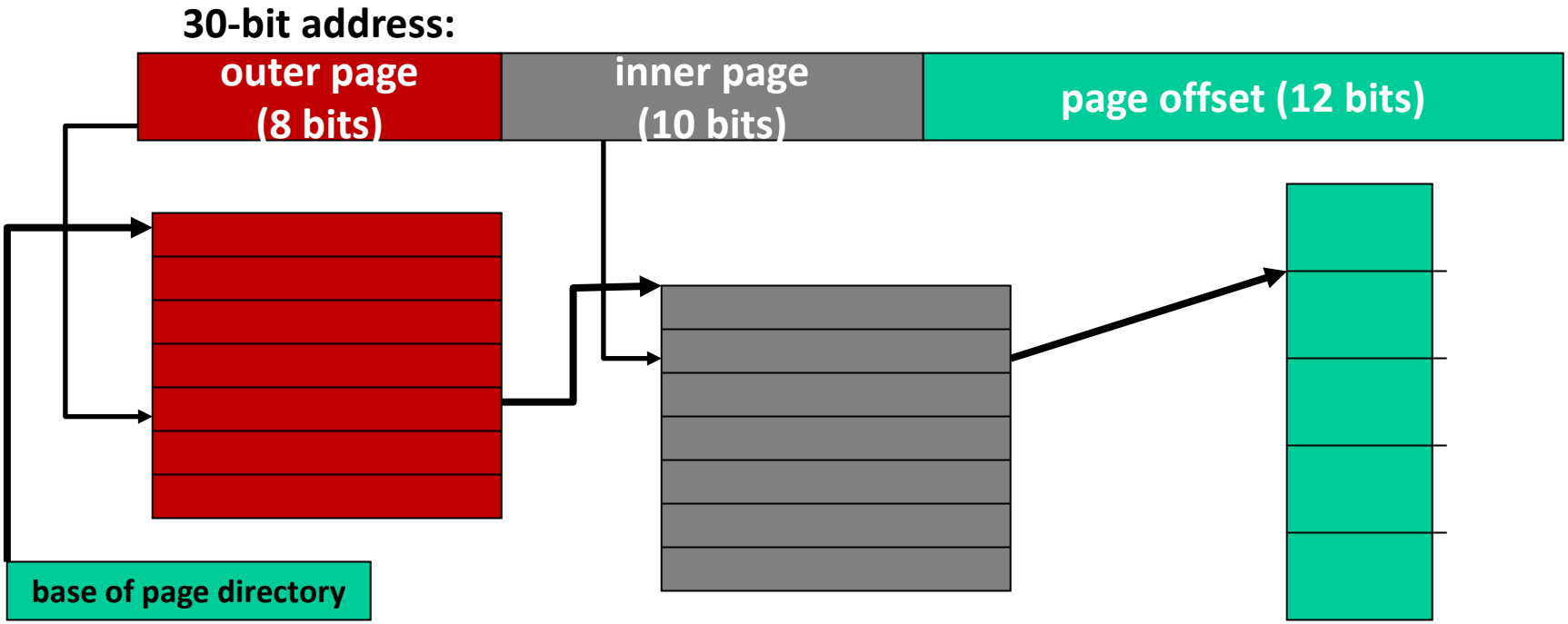
64 bits address? How about a large segment?

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)



Multilevel

page directory

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

page of PT (@PPN:0x92)

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

translate 0x01ABC

dir 0,PT 1: 0x23ABC

translate 0x00000

dir 0, PT 0: 0x10000

translate 0xFEED0

dir F, PT E: 0x55ED0

20-bit address:

Address Format for Multilevel Paging

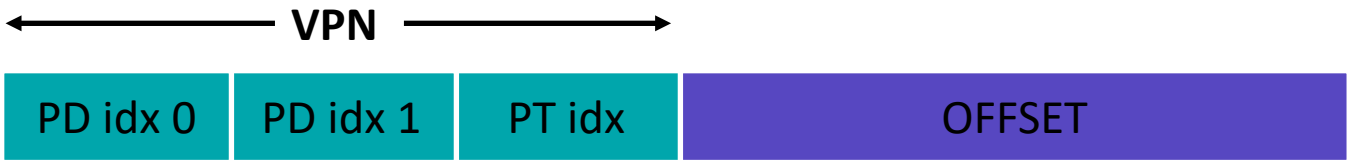
30-bit address:

- **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**
- **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

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 (2^8)
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?