

Memory Virtualization: Smaller Page Tables

OSTEP Chapter 20:

<http://pages.cs.wisc.edu/~remzi/OSTEP/vm-smalltables.pdf>

Jan Reineke
Universität des Saarlandes

Paging: Pros and Cons

Advantages:

- **No** external fragmentation:
 - free memory does not have to be allocated contiguously
- All free (unallocated) pages are “**equal**”:
 - easy to manage, allocate, and free pages

Disadvantages:

- **Too slow**: → TLBs
 - every “virtual” memory access results in two physical ones
- Page table are **too big**: (now)
 - one entry for every page of address space

Quiz: How big are page tables?

PTE = page table entry

1. PTEs are 2 bytes, and 32 possible virtual page numbers

$$2 \cdot 32 \text{ B} = 64 \text{ B}$$

$$\rightarrow 2 \cdot 2^5 \text{ bytes} = 64 \text{ bytes}$$

2. PTEs are 2 bytes, virt. addresses: 24 bits, pages are 16 bytes

$$2^{24} / 2^4 = 2^{20} \text{ PAGES}$$

$$\rightarrow 2 \cdot 2^{20} \text{ bytes} = 2 \cdot 2^{20} = 2 \text{ MB}$$

3. PTEs are 4 bytes, virt. addresses: 32 bits, pages are 4 KB

$$2^{32} / 2^{12} = 2^{20} \text{ PAGES}$$

$$\rightarrow 4 \cdot 2^{20} \text{ bytes} = 4 \cdot 2^{20} = 4 \text{ MB}$$

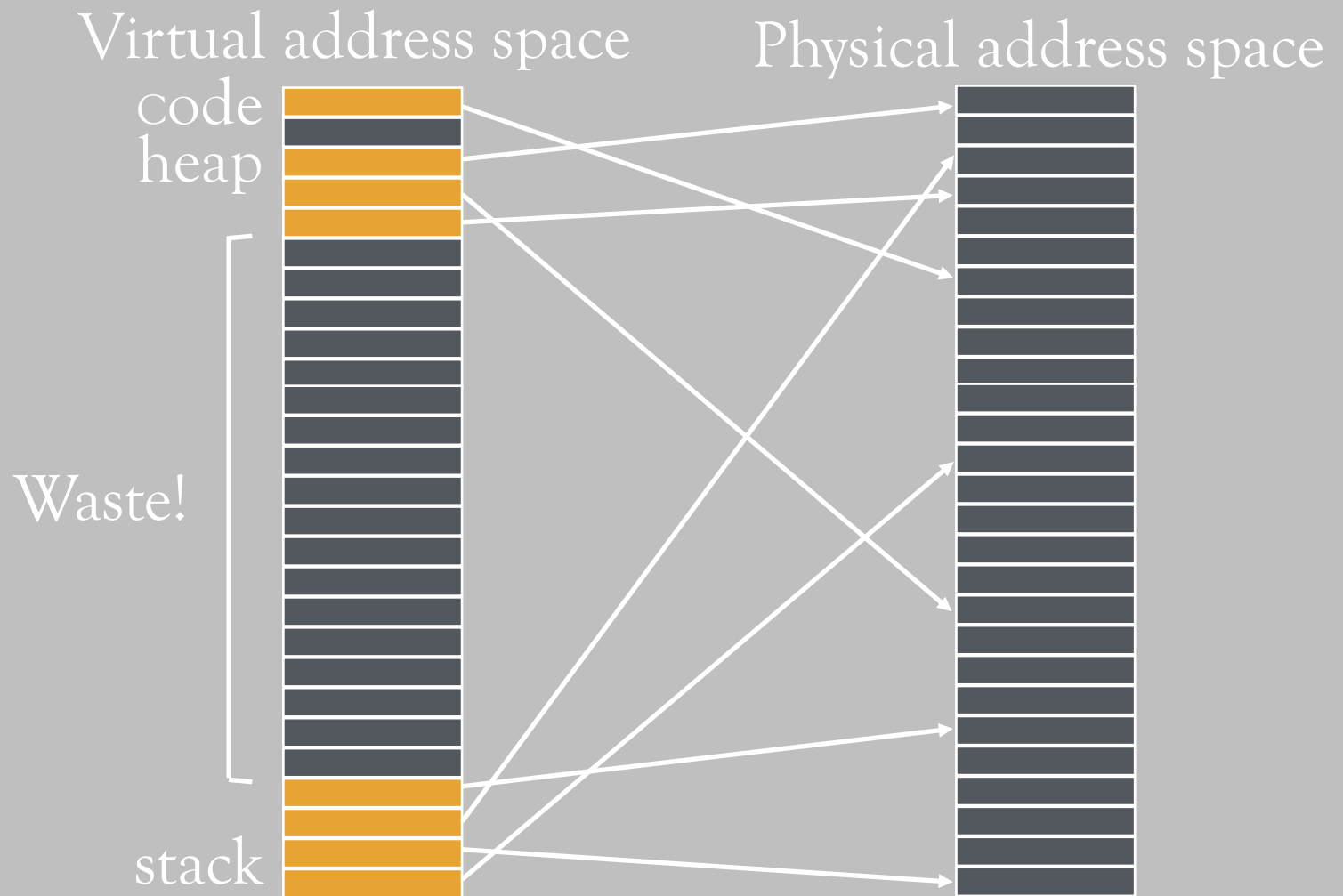
4. PTEs are 4 bytes, virt. addresses: 64 bits, pages are 4 KB

$$2^{22} \cdot 2^{32} \text{ B} = 2^{54} \text{ B}$$

$$\rightarrow 4 \cdot 2^{54} \text{ bytes} = 2^{54} \text{ B} = 2^{14} \text{ TB}$$

How big is each page table?

Waste!



Many invalid page table entries

How to avoid
storing these?

VPN	valid	protection
10	1	r-x
-	0	-
23	1	rw-
-	0	-
-	0	-
-	0	-
-	0	-
...many more invalid entries...		
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

Avoid linear page table

Approach:

Use hierarchical data structure instead of “flat” array

Any data structure is possible with software-managed TLB:

- HW looks for VPN on every memory access
- If TLB does not contain VPN, TLB miss
 1. HW generates exception, traps into OS
 2. OS finds PPN for given VPN
 3. OS enters PPN \rightarrow VPN into TLB
 4. Instruction that generated TLB miss is repeated (“exception return” without change of “epc” in MIPS)

Approaches

1. Segmented page tables
2. Multi-level page tables

Observation:

Valid PTEs are contiguous

How to avoid
storing these?

VPN	valid	protection
10	1	r-x
-	0	-
23	1	rw-
-	0	-
-	0	-
-	0	-
-	0	-
...many more invalid entries...		
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

Idea:
Combine
segmentation
and **paging**

1. Combine paging and segmentation

Divide address space into segments (code, heap, stack)

- Segments can be **variable length**

Divide each segment into fixed-size pages

Virtual address divided into three portions:

segment no. (4 bits)	page number (8 bits)	page offset (12 bits)
-------------------------	----------------------	-----------------------

Implementation: per segment

- Each segment has a page table (only as large as necessary)
- Base address and size of page table

Quiz: Paging and segmentation

segment no. (4 bits)		page number (8 bits)		page offset (12 bits)		
Seg	Base	Bounds	R W			
0	0x002000	0xff	1 0	00	0x01f	0x001000
1	0x000000	0x00	0 0	01	0x011	
2	0x001000	0x0f	1 1	02	0x003	
				03	0x02a	
					0x013	
					...	0x002000
				00	0x00c	
				01	0x007	
				02	0x004	
					0x00b	
					0x006	
					...	

0x002070	read:	0x004070
0x202016	read:	0x003016
0x104c84	read:	error
0x010424	write:	error
0x210014	write:	error
0x203568	read:	0x02a568

Handwritten notes:
- Red box around 0x002070 in the first table, with "S VPN" and "offset" written above it.
- Red box around 0x0010424 in the second table.
- Red box around 0x02a568 in the second table.

Advantages of paging and segmentation

Advantages of segments

- Supports sparse address spaces
 - decreases sizes of page tables
 - no need for page table if segment not used

Advantages of paging

- no external fragmentation
- segments can grow without any reshuffling
- can run process when some pages are swapped to disk (later)

Advantages of both

- Increases flexibility of sharing
 - share either single page or entire segment

Disadvantages of paging and segmentation

- Potentially large page tables (for each segment)
- Must allocate each page table contiguously

Approaches

1. Segmented page tables
2. Multi-level page tables

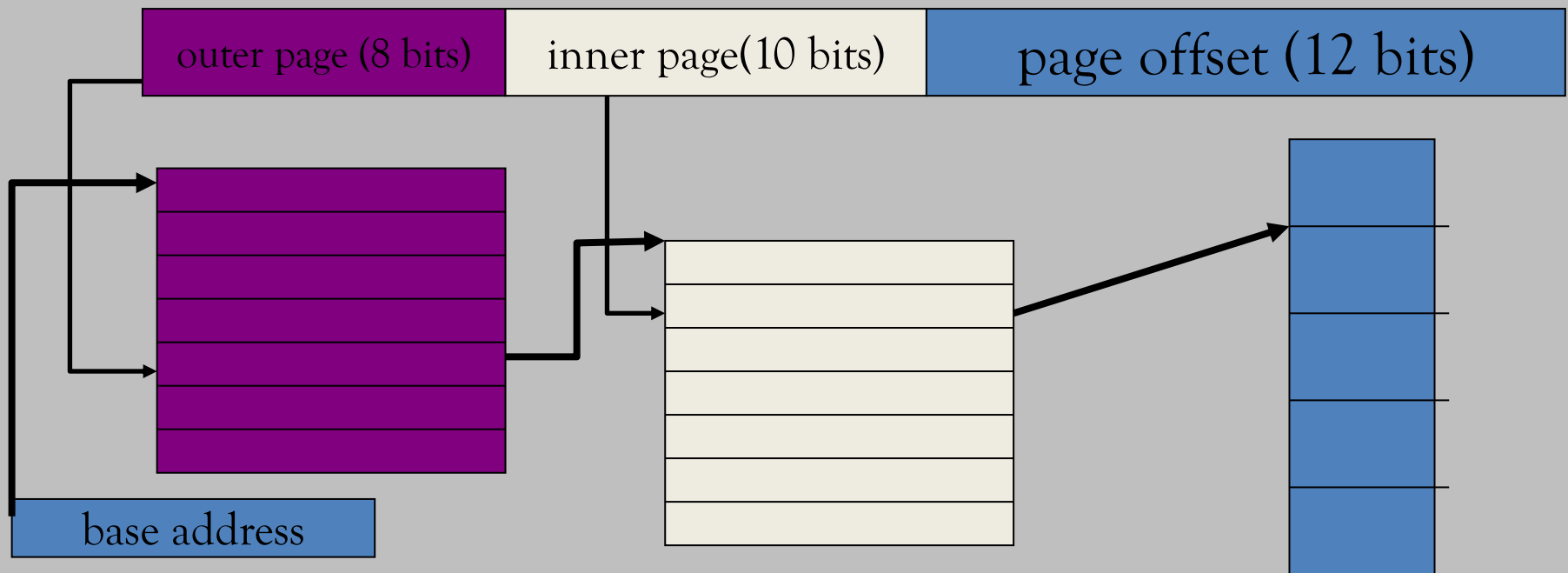
2. Multi-level page tables

Goal: Allow each page table to be allocated non-contiguously

Idea: **Hierarchical page tables**

- Several translation levels, inner tables stored in pages
- Only allocate page tables for pages in use
- Used in x86 architectures (hardware can walk known structure)

30-bit address:



Quiz: Multi-level page table

Page directory

@PPN:0x3

@PPN:0x92

PPN	valid
0x3	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
0x92	1

PPN	valid
0x10	1
0x23	1
-	0
-	0
0x80	1
0x59	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

PPN	valid
-	0
-	0
-	0
-	0
-	0
-	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:

outer page
(4 bits)

inner page
(4 bits)

page offset (12 bits)

Quiz: Address format for multi-level paging

30-bit address:



How should virtual addresses be structured?

- How many bits for each paging level?

Goal?

- Each page table fits within a page
- PTE size * number of PTEs = page size
 - Assume: PTE size = 4 bytes
 - Page size = 2^{12} byte = 4 KB
 - 2^2 byte * number of PTEs = 2^{12} bytes
 - number of PTEs (per inner page table) = 2^{10}
 - #bits for selecting inner page = 10

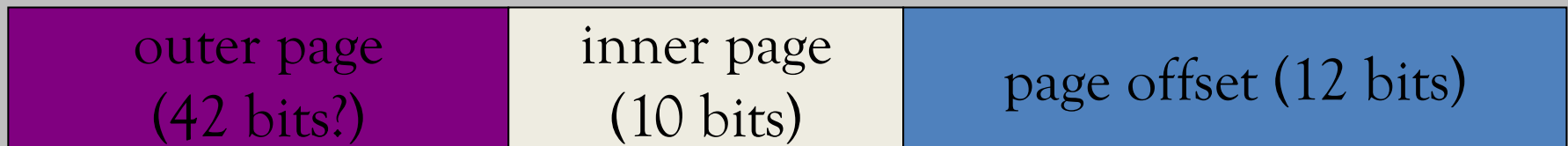
Remaining bits for outer page:

- $30 - 10 - 12 = 8$ bits

Problem with 2 levels?

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

64-bit address:



Solution: Additional translation levels



Size of the virtual address space

How large is the virtual address space with 4 KB pages, 4 byte PTEs, every page table fits into a page?

4KB / 4 bytes \rightarrow 1K entries per level

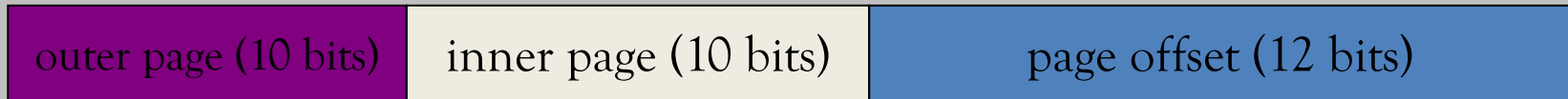
1 level: $1K * 4K = 2^{22} = 4 \text{ MB}$

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

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

Quiz: How much space is “used” by a multi-level page table in memory?

Example: 32-bit address:



PTE size 4 Byte

How much memory is required **minimally** for a multi-level page table?

$2^{10} \cdot 4 \text{ bytes} = 4 \text{ KB}$ for outer page table
+ $2^{10} \cdot 4 \text{ bytes} = 4 \text{ KB}$ for **one** inner page table

How much memory is required **maximally** for a multi-level page table?

$2^{10} \cdot 4 \text{ bytes} = 4 \text{ KB}$ for outer page table
+ $2^{10} \cdot 2^{10} \cdot 4 \text{ bytes} = 4 \text{ MB}$ for **1024** inner page tables

Quiz: Full system with TLBs

On TLB miss: lookups with more levels more expensive
How much does a miss cost?

Assumptions:

- 3-level page table
- 256-byte pages
- 16-bit addresses
- ASID of current process is 211

ASID	VPN	PFN	Valid
211	0xBB	0x91	1
211	0xFF	0x23	1
122	0x05	0x91	1
211	0x05	0x12	0

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

(a) 0xAA10: movl 0x1111, %edi

0xAA: (TLB miss -> 3 for addr. translation) + 1 instruction fetch
0x11: (TLB miss -> 3 for addr. translation) + 1 movl Total: 8

(b) 0xBB13: addl \$0x3, %edi

0xBB: (TLB hit -> 0 for addr. translation) + 1 instr. fetch Total: 1

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

0x05: (TLB miss -> 3 for addr. translation) + 1 instr. fetch
0xFF: (TLB hit -> 0 for addr. translation) + 1 movl Total: 5

Summary:

Better page tables

Problem:

Simple linear page tables require too much memory

Many options for efficiently organizing page tables

If OS traps on TLB miss, OS can use any data structure

If HW handles TLB miss, page tables must follow specific format:

- Multi-level page tables used in x86 architecture
- Each page table fits within a page