Virtualizing Memory: Paging

Questions answered in this lecture:

Review segmentation and fragmentation

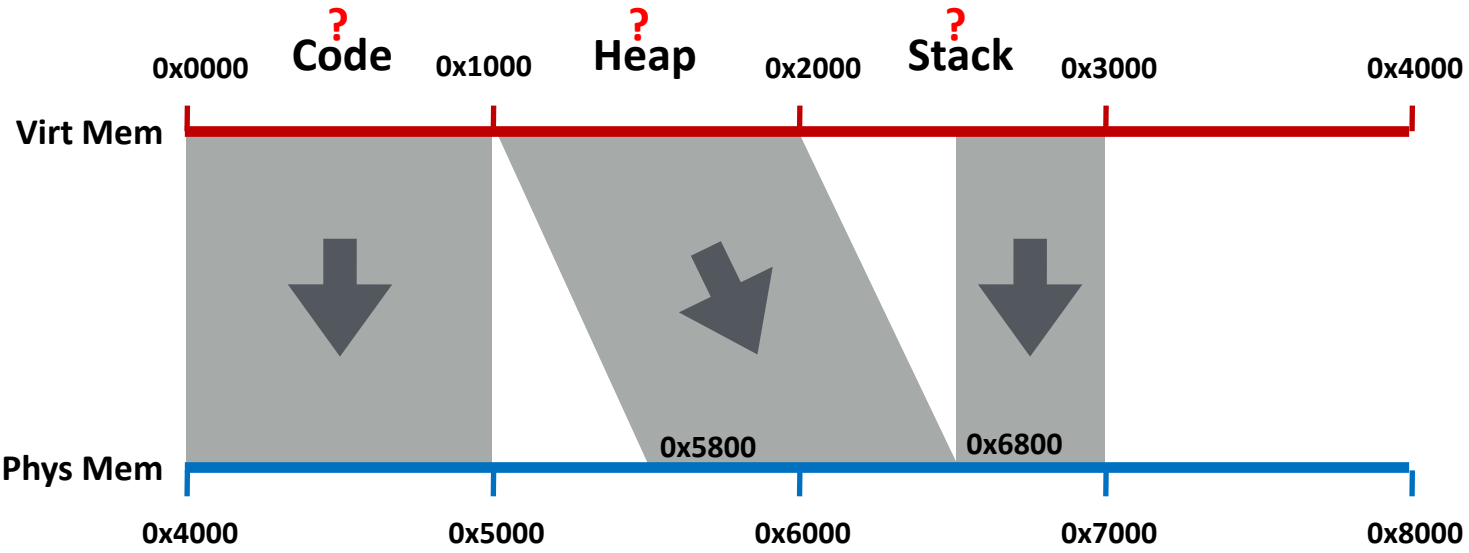
What is paging?

Where are page tables stored?

What are advantages and disadvantages of paging?

Review: Segmentation

Assume 14-bit virtual addresses, high 2 bits indicate segment



Segments:
0=>code
1=>heap
2=>stack.

Where doe segment table live?

All registers, MMU

Seg	Base	Bounds
0	**0x4000**	**0xfff**
1	**0x5800**	**0xfff**
2	**0x6800**	**0x7ff**

Review: Memory Accesses

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

%rip: 0x0010

Physical Memory Accesses?

1) Fetch instruction at logical addr 0x0010

- Physical addr: **0x4010**

Exec, load from logical addr 0x1100

- Physical addr: **0x5900**

2) Fetch instruction at logical addr 0x0013

- Physical addr: **0x4013**

Exec, no load

3) Fetch instruction at logical addr 0x0019

- Physical addr: **0x4019**

Exec, store to logical addr 0x1100

- Physical addr: **0x5900**

Total of 5 memory references (3 instruction fetches, 2 movl)

x86 Segmentation Implementation

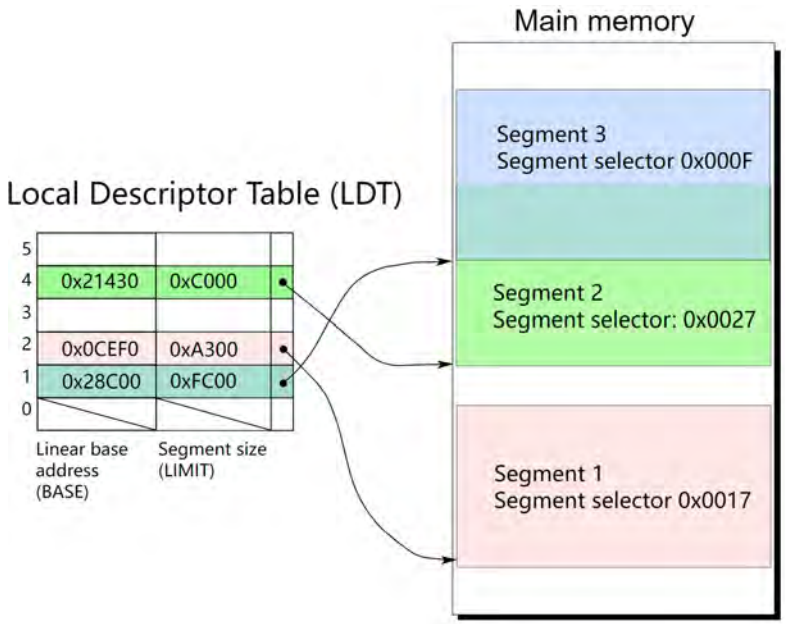
- **Real Mode**

segment selector
(16 bit register)

```
func:
        mov eax, [ss:esp+4]
        mov ebx, [ds:eax]
        add ebx, 8
        mov [ds:eax], ebx
        ret
```

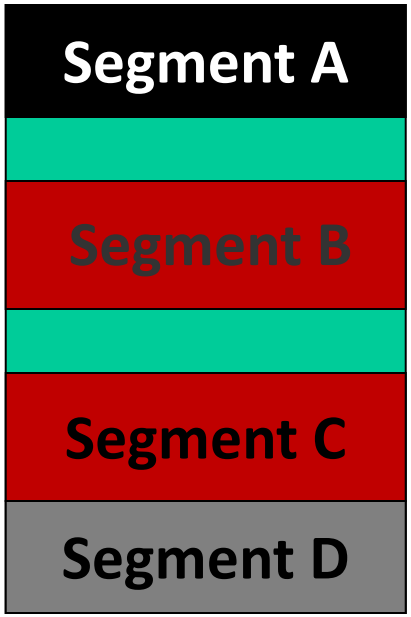
	0000 0110 1110 1111 0000	**Segment**	16 bits, shifted 4 bits left (or multiplied by 0x10)
+	0001 0010 0011 0100	**Offset**	16 bits
	0000 1000 0001 0010 0100	**Address**	20 bits

- **Protected Mode**



Problem: Fragmentation

- **Definition: Free memory that can't be usefully allocated**
- **Why?**
  - Free memory (hole) is too small and scattered
  - Rules for allocating memory prohibit using this free space
- **Types of fragmentation**
  - **External**: Visible to allocator (e.g., OS)
  - **Internal**: Visible to requester (e.g., if must allocate at some granularity)



External

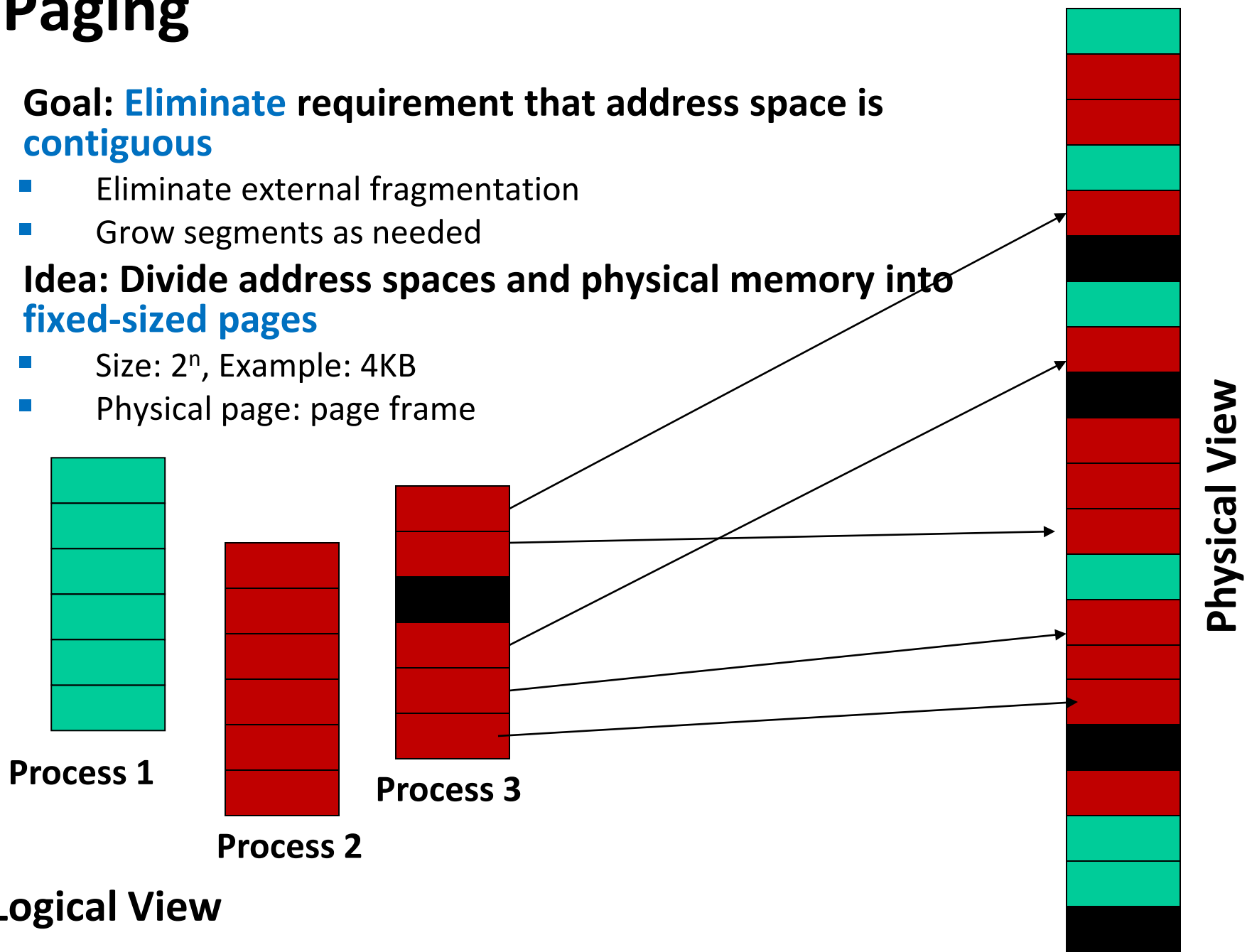
No contiguous space!

Allocated to requester

Internal

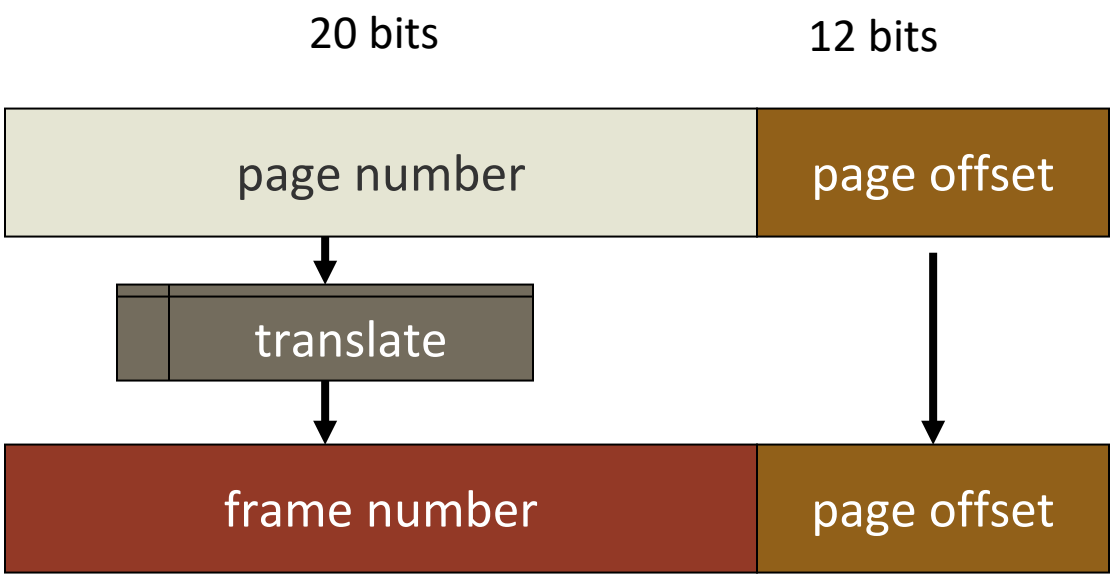
Paging

- **Goal: Eliminate requirement that address space is contiguous**
  - Eliminate external fragmentation
  - Grow segments as needed
- **Idea: Divide address spaces and physical memory into fixed-sized pages**
  - Size: 2^n, Example: 4KB
  - Physical page: page frame



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



32 bits

Logical address

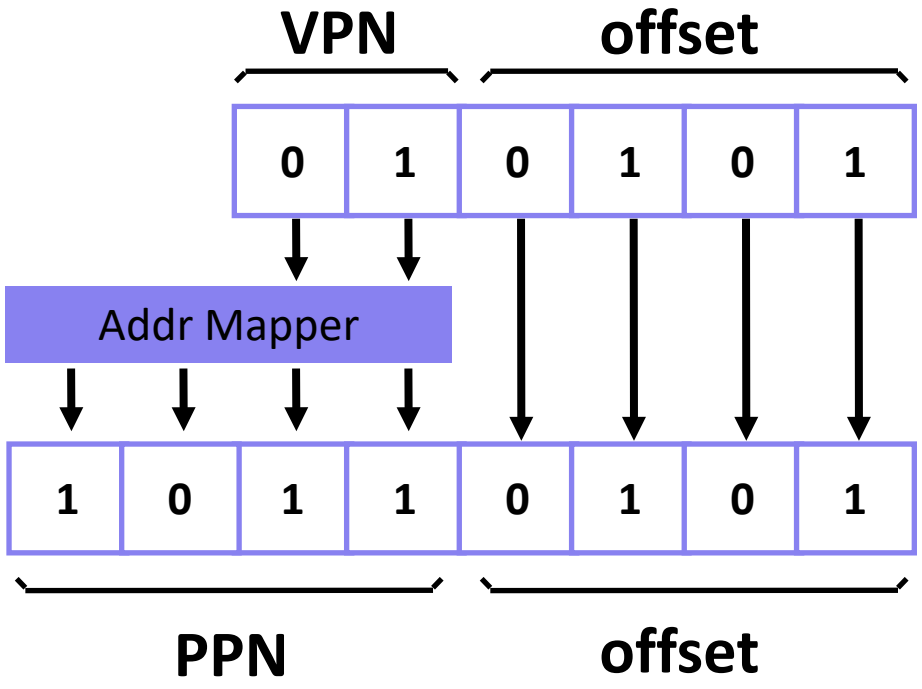
Physical address

No addition needed; just append bits correctly…

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

Virtual => Physical Page Mapping

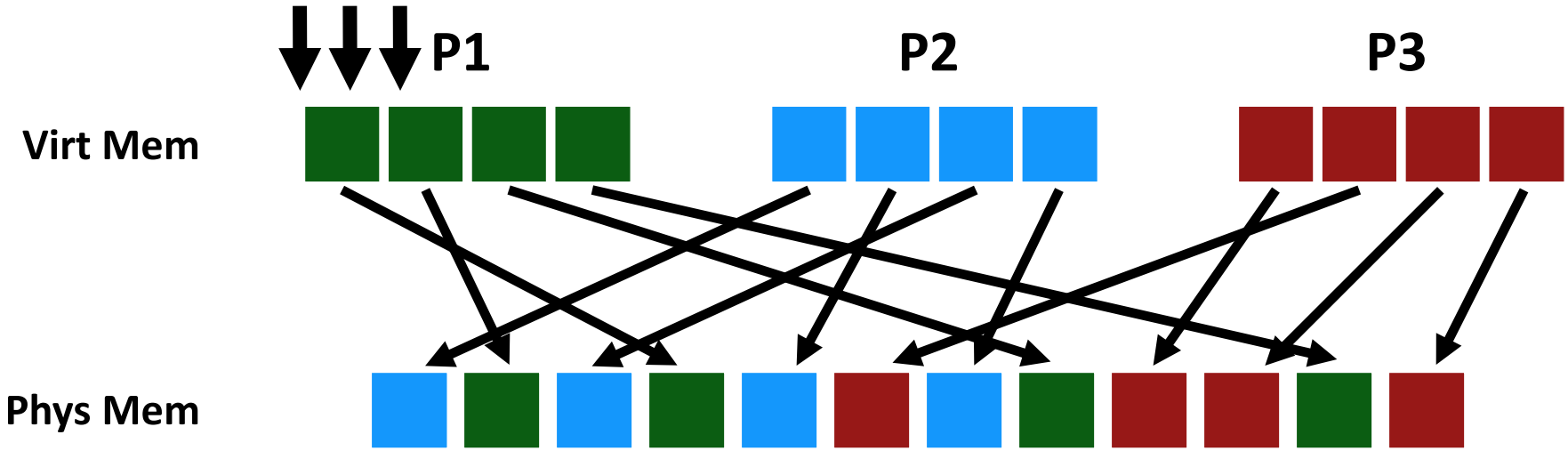
Number of bits in virtual address format does not need to equal number of bits in physical address format



- How should OS translate VPN to PPN?
- For segmentation, OS used a formula (e.g., phys addr = virt_offset + base_reg)
- For paging, OS needs more general mapping mechanism
- What data structure is good?

Big array: pagetable

The Mapping



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 M
  - 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?

Simplified view of page table

2
0
80
99

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