UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Introduction to Operating Systems

Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

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?

Announcements

- P1
  - Due officially by Friday, 5pm; unofficially by Saturday 8am8
  - Lots of test scripts available; run 1a-contest scripts too
  - Lots of office hours through Friday 4:30
- Project 2: Available on Monday
  - Due two weeks later: Monday, Oct 5
  - Can work with project partner in your discussion section (unofficial)
  - Two parts:
    - Linux: Shell -- fork() and exec(), file redirection, history
    - Xv6: Scheduler – simplistic MLFQ
    - Two discussion videos again; watch early and often!
- Exam 1: Two weeks, Thu 10/1 7:15 – 9:15
  - Class time that day for review
  - Look at homeworks / simulations for sample questions
- Reading for today:
  - Chapter 18

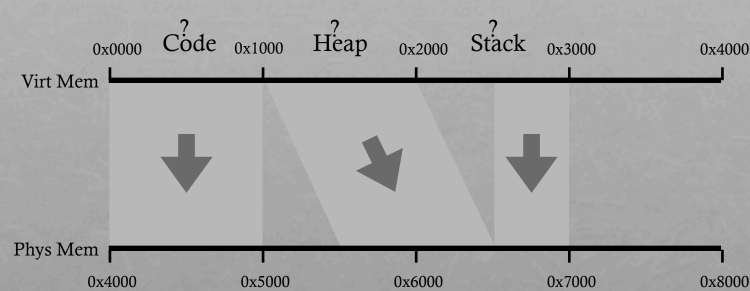
Review: Match Description

Description	Name of approach (covered previous lecture):
one process uses RAM at a time	**Segmentation**
rewrite code and addresses before running	**Static Relocation**
add per-process starting location to virt addr to obtain phys addr	**Base**
dynamic approach that verifies address is in valid range	**Base+Bounds**
several base+bound pairs per process	**Time Sharing**

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

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

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

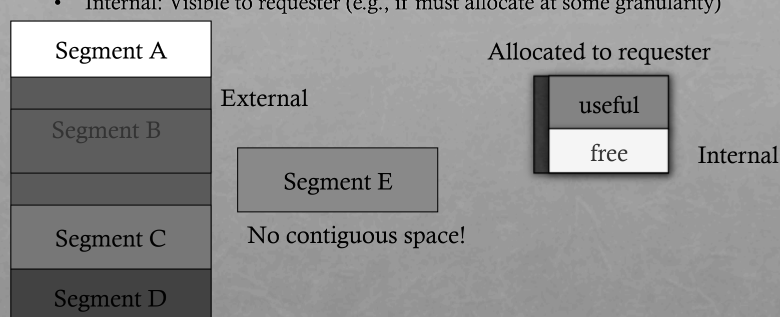
Total of 5 memory references (3 instruction fetches, 2 movl)

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)



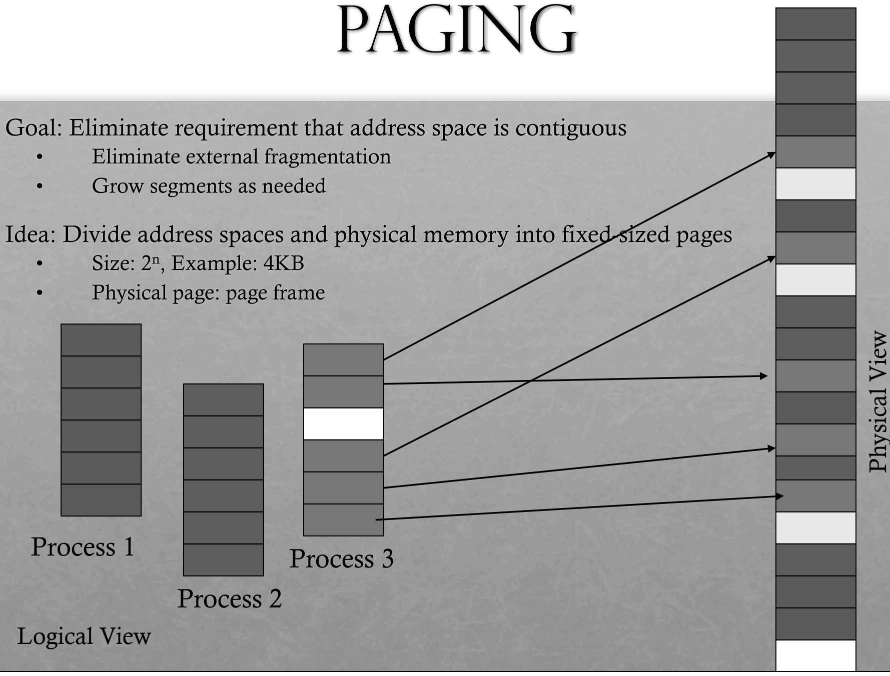
Paging

Goal: Eliminate requirement that address space is contiguous

- Eliminate external fragmentation
- Grow segments as needed

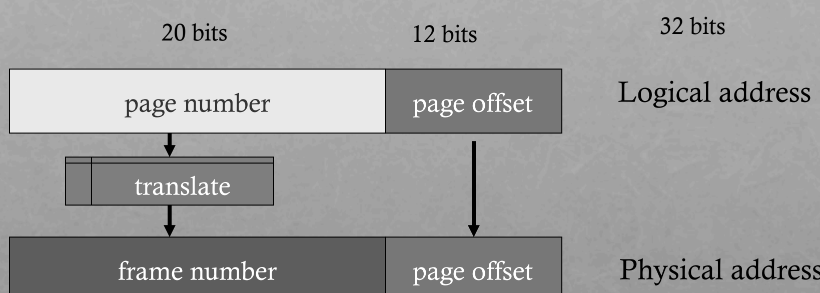
Idea: Divide address spaces and physical memory into fixed-sized pages

- 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



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	7
4 KB	12	32	20	

Correct?

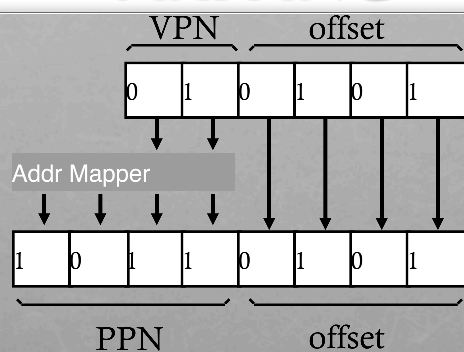
Quiz: Address Format

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

Virtual => Physical Page Mapping

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



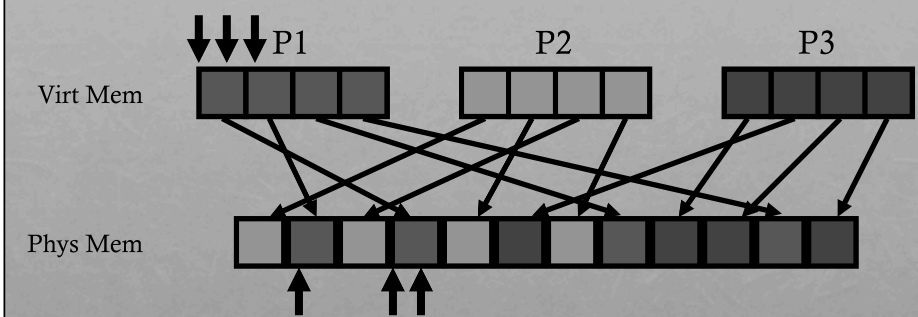
How should OS translate VPN to PPN?

For segmentation, OS used a formula (e.g., phys addr = virt_offset + base_reg)

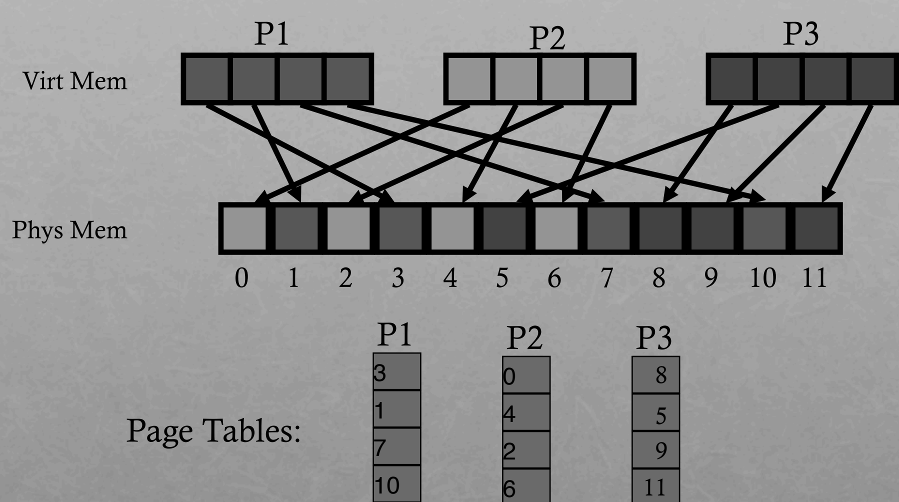
For paging, OS needs more general mapping mechanism

What data structure is good? Big array: pagetable

The Mapping



Quiz: Fill in Page Table



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

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

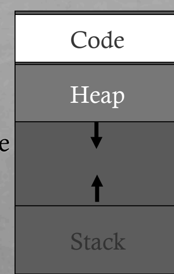
- Wasted memory grows with larger pages
- **Tension?**

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)



Homework Exercises

- Look at relocation.py
  - Base+bounds dynamic relocation
- Look at page-linear-translate.py
  - Basic page tables