



Department of Physics,
Computer Science & Engineering

CPSC 410 – Operating Systems I

Virtualizing Memory: Paging

Keith Perkins

Adapted from “CS 537 Introduction to Operating Systems”
Arpaci-Dusseau

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:

Match Description

DESCRIPTION

- one process uses RAM at a time
- rewrite code and addresses before running
- add per-process starting location to virt addr to obtain phys addr
- dynamic approach that verifies address is in valid range
- several base+bound pairs per process

NAME OF APPROACH

(COVERED PREVIOUS LECTURE):

Segmentation

Static Relocation

Base

Base+Bounds

Time Sharing

Review: Match Description

DESCRIPTION

- one process uses RAM at a time
- rewrite code and addresses before running
- add per-process starting location to virt addr to obtain phys addr
- dynamic approach that verifies address is in valid range
- several base+bound pairs per process

NAME OF APPROACH

(COVERED PREVIOUS LECTURE):

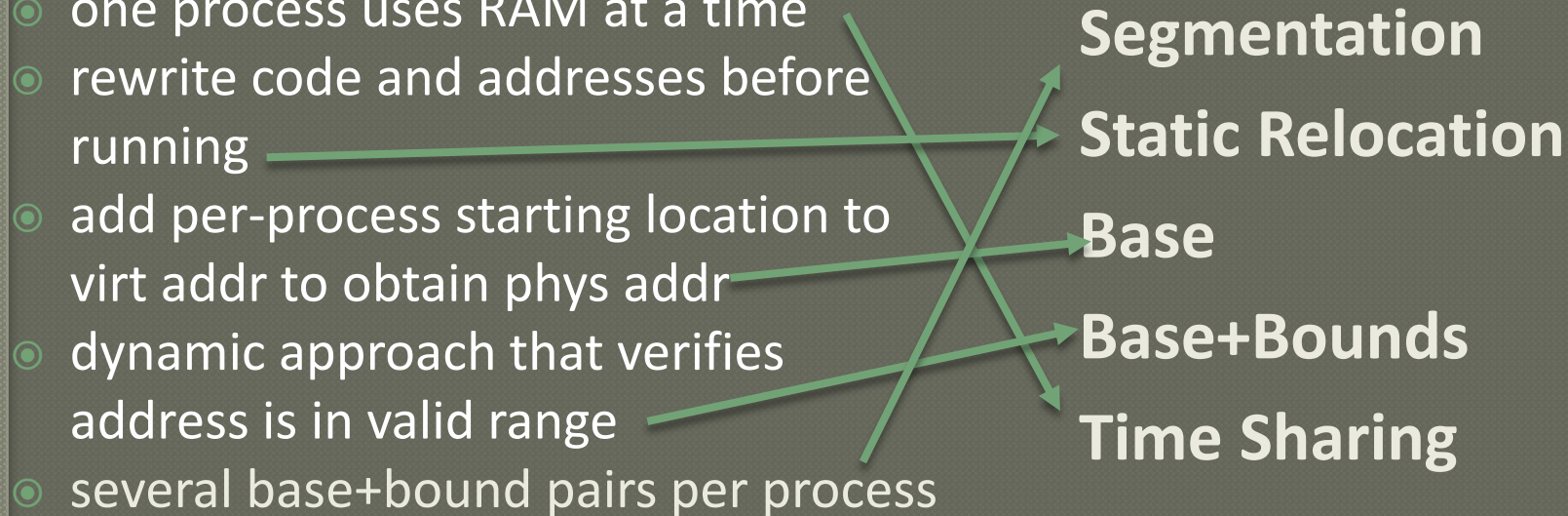
Segmentation

Static Relocation

Base

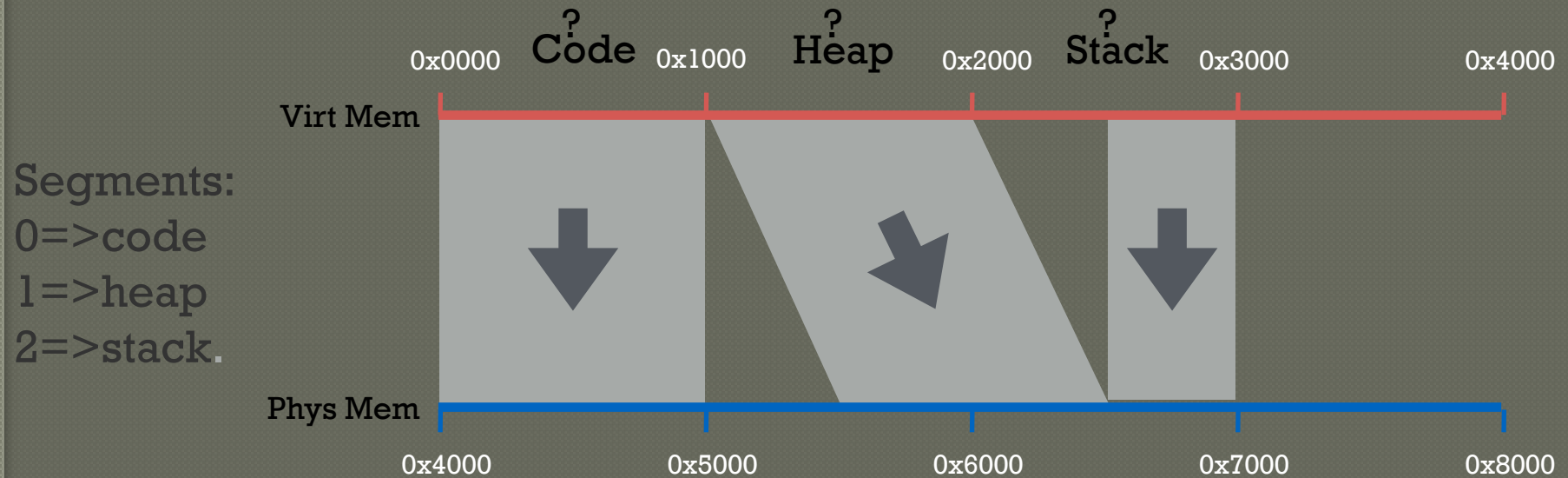
Base+Bounds

Time Sharing



Review: Segmentation

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



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

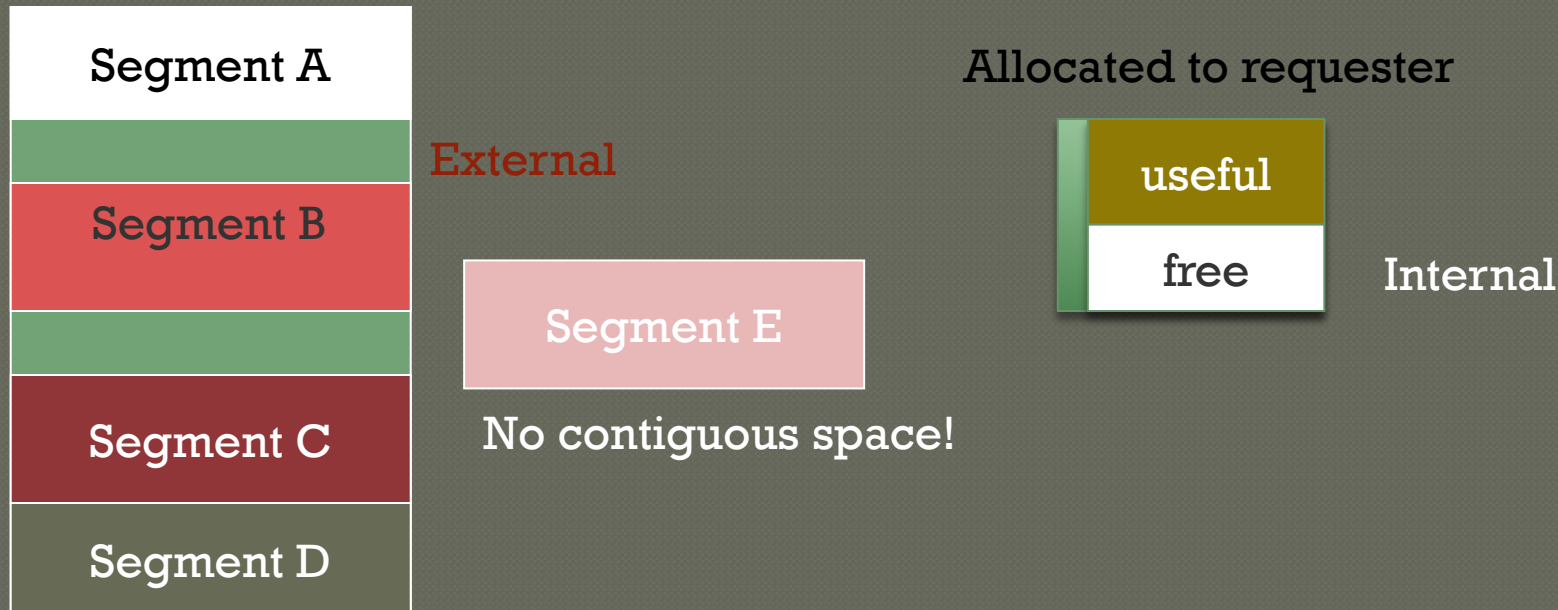
Definition: Free memory that can't be usefully allocated

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)



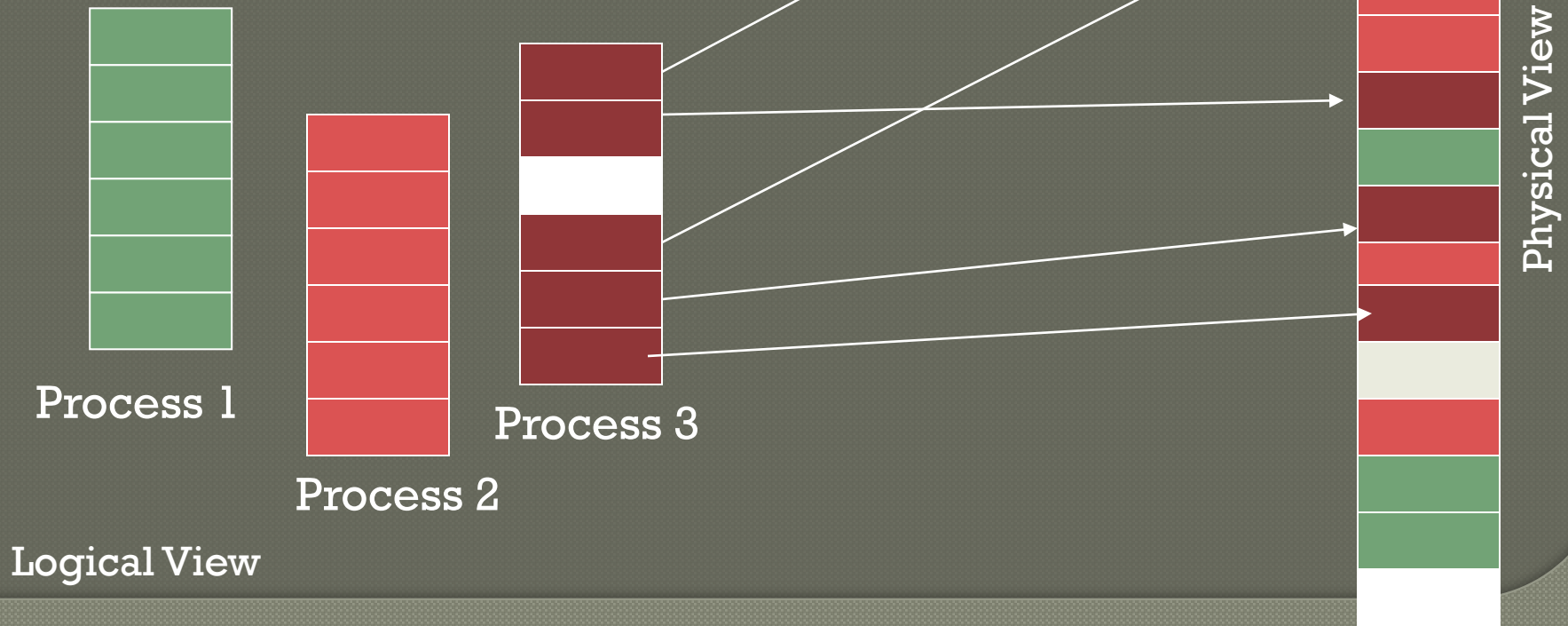
Paging

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 ($n=12$)
- Physical page: page frame



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 ($n=12$)
- Physical page: page frame

| |
|---|
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

Process 1

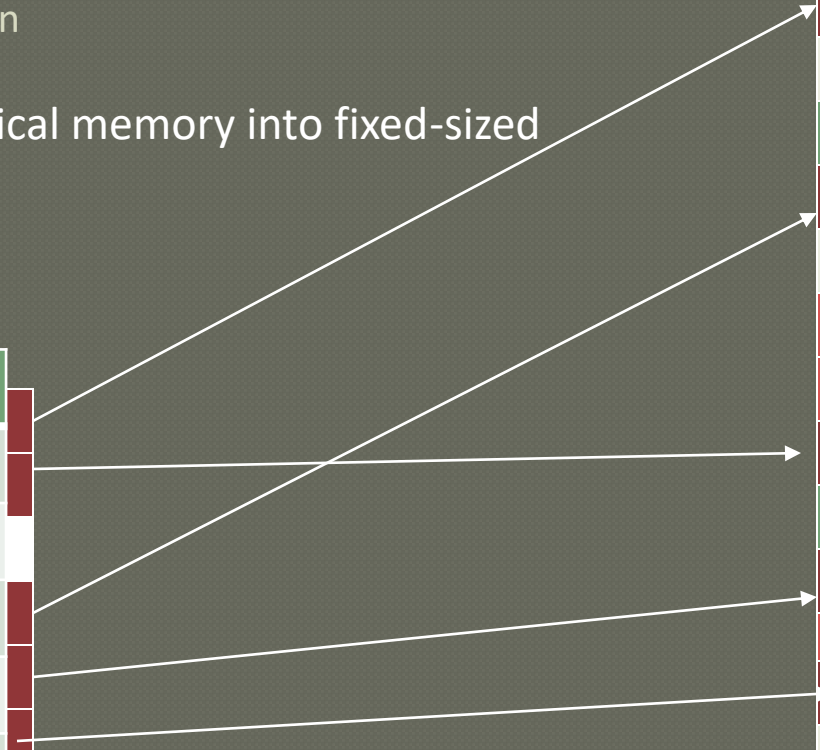
| | |
|---|----|
| 0 | 3 |
| 1 | 0 |
| 2 | 6 |
| 3 | 18 |
| 4 | 19 |
| 5 | 12 |
| | |
| | |

s 3

Logical View

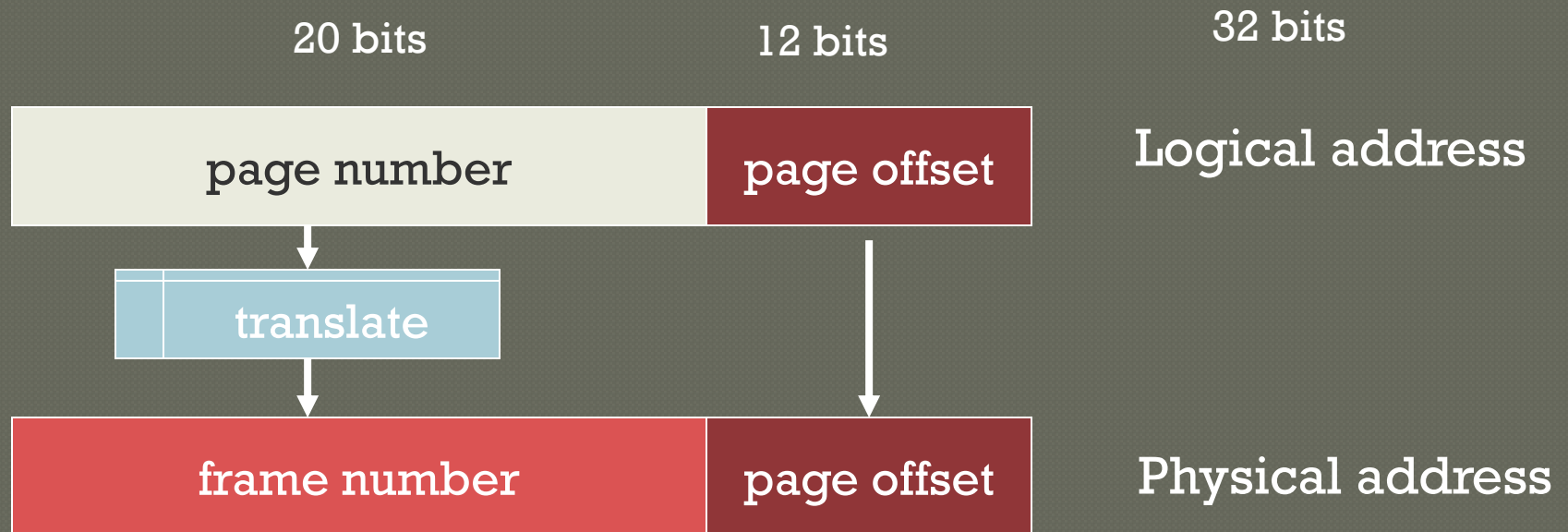
| |
|---|
| 1 |
| |
| |
| 0 |
| |
| |
| 2 |
| |
| |
| |
| |
| 5 |
| |
| |
| |
| |
| |
| 3 |
| 4 |
| |

Physical View



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



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?

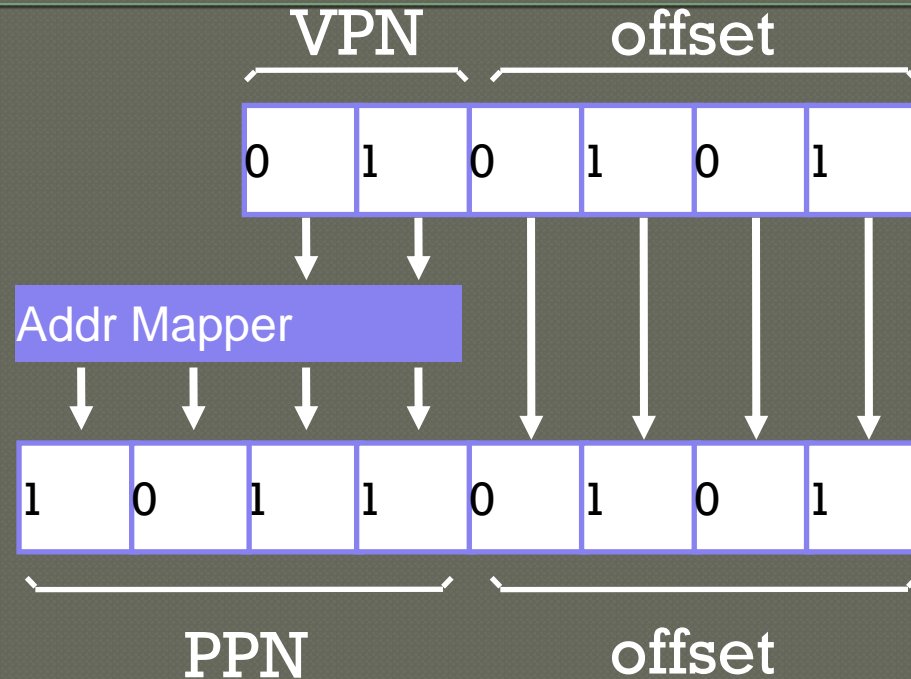
| Page Size | Low Bits (offset) | Virt Addr Bits | High Bits (vpn) |
|-----------|----------------------|----------------|--------------------|
| 16 bytes | 4 | 10 | 6 |
| 1 KB | 10 | 20 | 10 |
| 1 MB | 20 | 32 | 12 |
| 512 bytes | 9 | 16 | 7 |
| 4 KB | 12 | 32 | 20 |

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 | 7 | 128 |
| 4 KB | 12 | 32 | 20 | 1 MB |

VirtUAL => Physical PAGE Mapping



How should OS translate VPN to PPN?

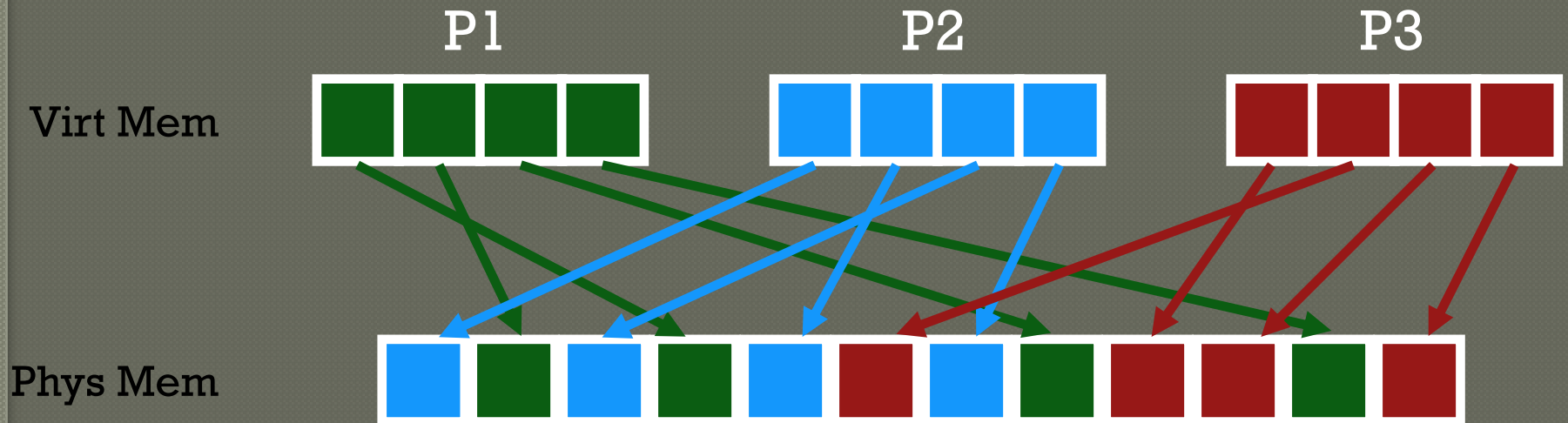
For segmentation, OS used a formula (e.g., $\text{phys addr} = \text{virt_offset} + \text{base_reg}$)

For paging, OS needs more general mapping mechanism

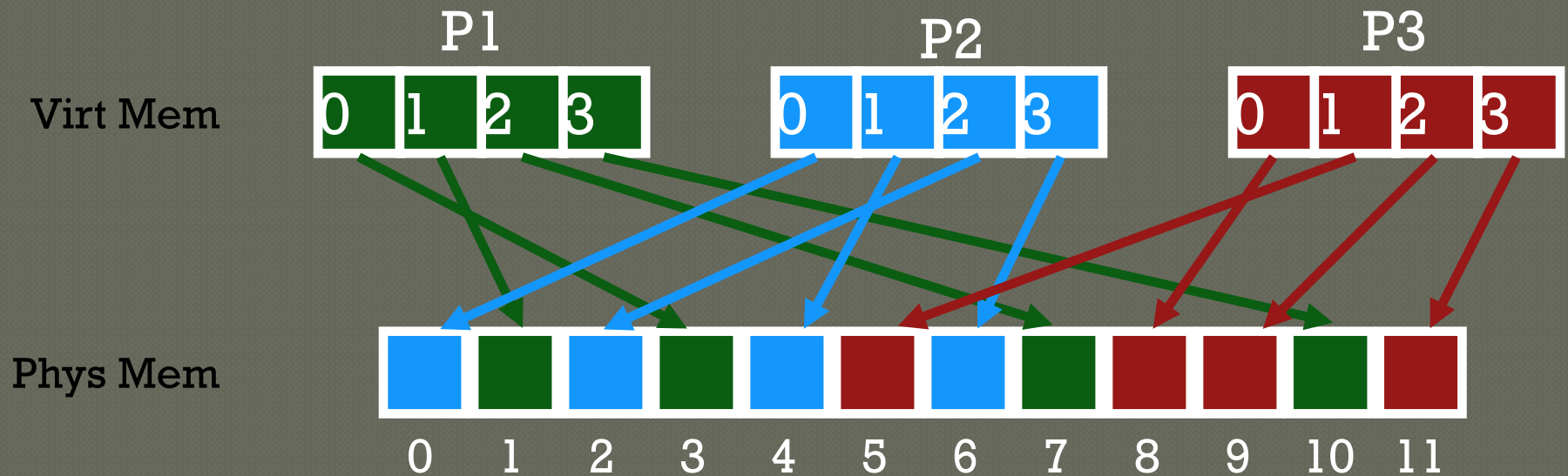
What data structure is good?

An Array: **pagetable**

The Mapping



Quiz: Fill in Page Table



Page Tables:

| P1 | P2 | P3 |
|----|----|----|
| 3 | 0 | 8 |
| 1 | 4 | 5 |
| 7 | 2 | 9 |
| 10 | 6 | 11 |

Where Are Pagetables Stored?

How big is a typical page table?

- assume **32-bit** address space
- assume 4 KB pages
- assume 4 byte entries per page table row

Final answer: $2^{(32 - \log(4KB))} * 4 = 4 \text{ MB}$

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

Other PT info

What other info is in pagetable entries besides translation?

- valid bit
- protection bits(R/W)
- present bit (is it memory or on disk?)
- reference bit (used for page eviction algorithms)
- dirty bit (has a page changed and thus needs to be rewritten to disk)

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

So how many bits for offset? 12

Simplified view
of page table

| | |
|--------|----|
| 0x5000 | 2 |
| 0x5004 | 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 (0x0010--0x2010)
- 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 (0x1100--0x0100)
- Movl from 0x0100 into reg (Mem ref 4)
- And so on...

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 coalesce 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 (a fast cache...and a 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 (its an array)
- Solution: Combine paging and segmentation (future lecture)

