MEMORY: PAGING AND TLBS

Shivaram Venkataraman

CS 537, Spring 2020

PAGING

Goal: Eliminate requirement that address space is contiguous

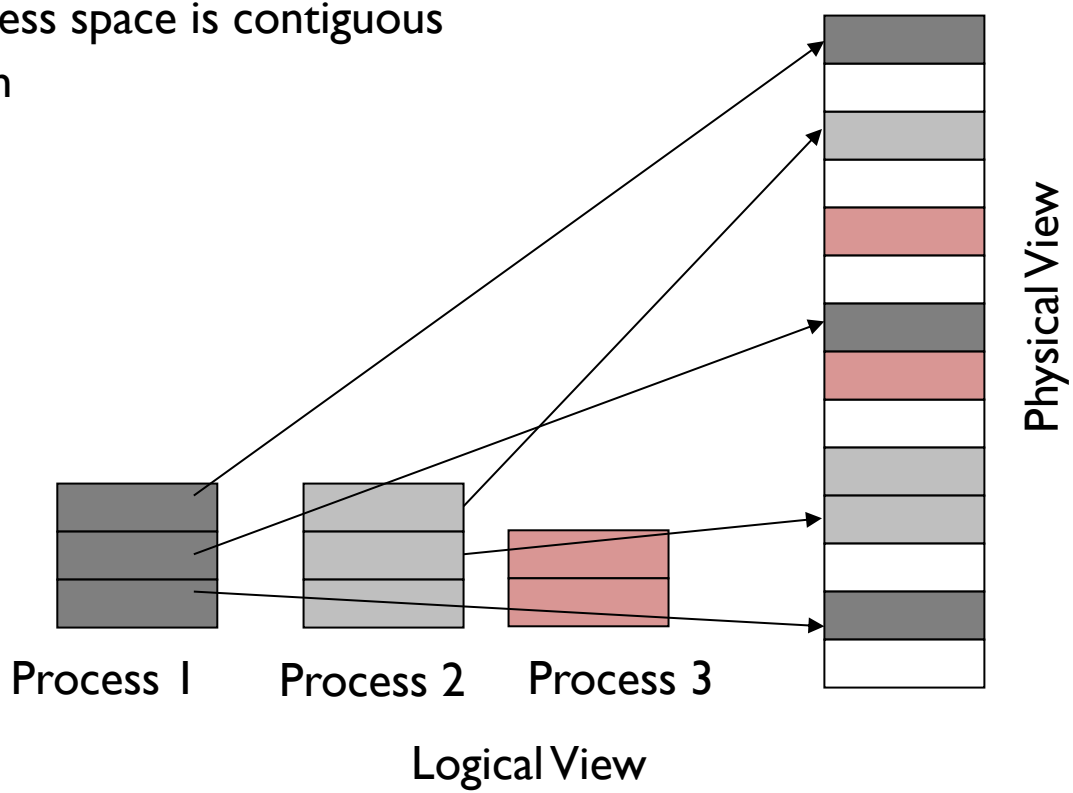
Eliminate external fragmentation

Grow segments as needed

Idea:

Divide address spaces and physical memory into fixed-sized pages

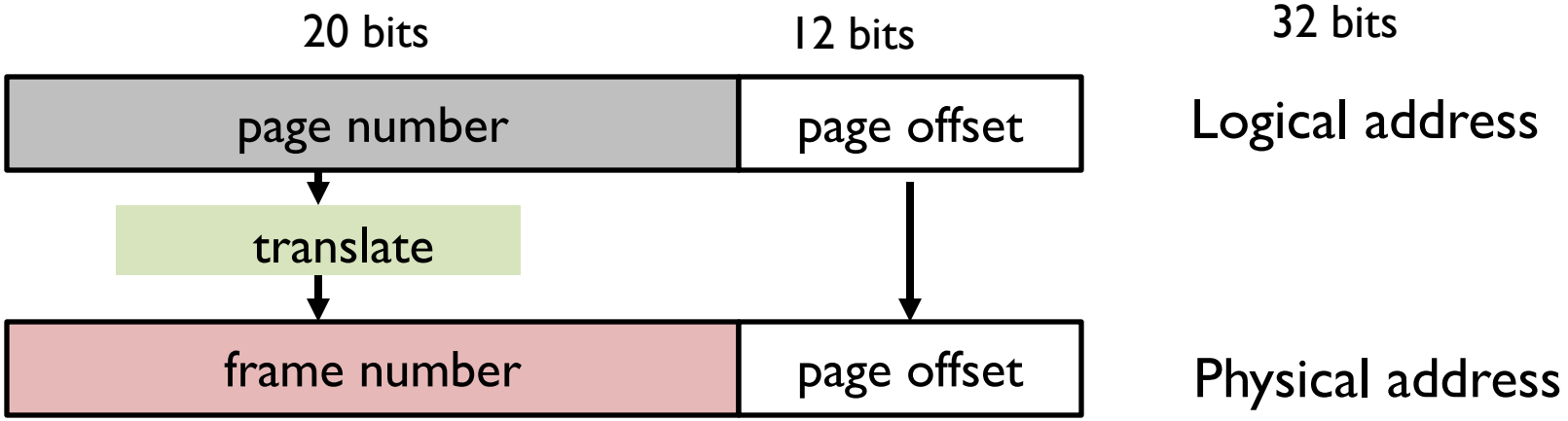
Size: 2^n, Example: 4KB



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!

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	
1 KB	
1 MB	
512 bytes	
4 KB	

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 Total Bits	High Bits(vpn)
16 bytes	4	10	
1 KB	10	20	
1 MB	20	32	
512 bytes	9	16	
4 KB	12	32	

ADDRESS FORMAT

Given number of bits for vpn, how many virtual pages can there be in an address space?

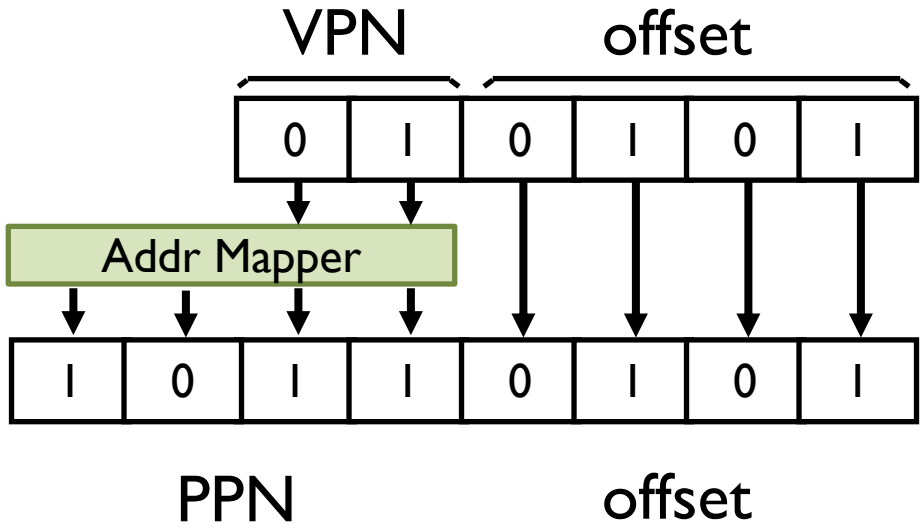
Page Size	Low Bits (offset)	Virt Addr Bits	High Bits (vpn)	Virt Pages
16 bytes	4	10	6	
1 KB	10	20	10	
1 MB	20	32	12	
512 bytes	9	16	7	
4 KB	12	32	20	

VIRTUAL → PHYSICAL PAGE MAPPING

Number of bits in virtual address

need not equal

number of bits in physical address

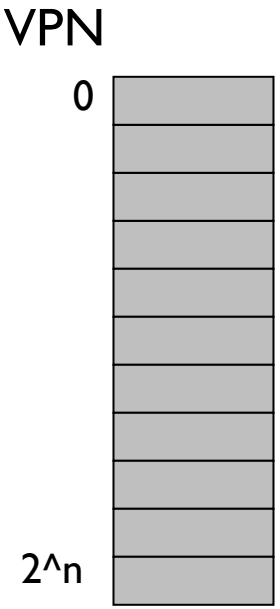


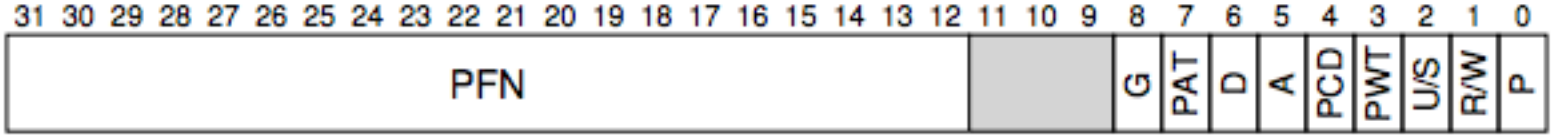
How should OS translate VPN to PPN?

PAGETABLES

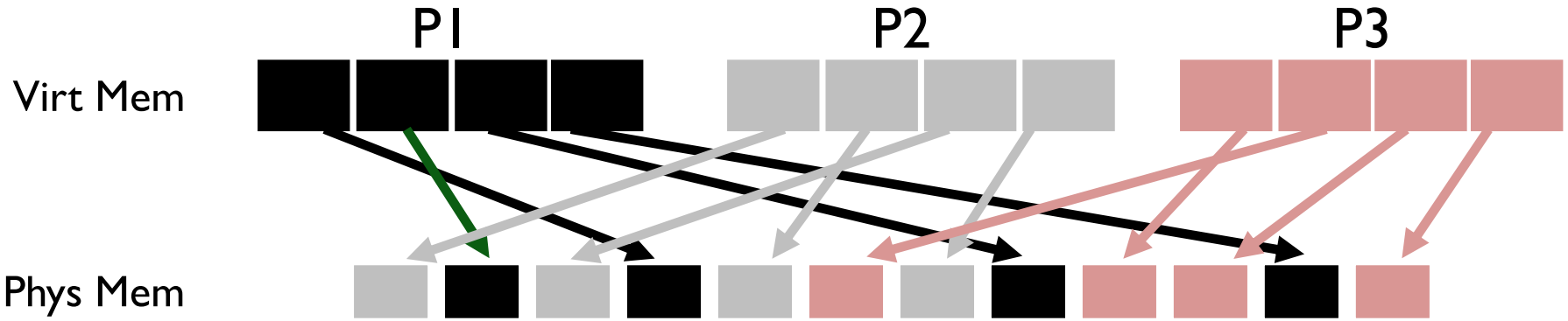
What is a good data structure ?

Simple solution: Linear page table aka *array*

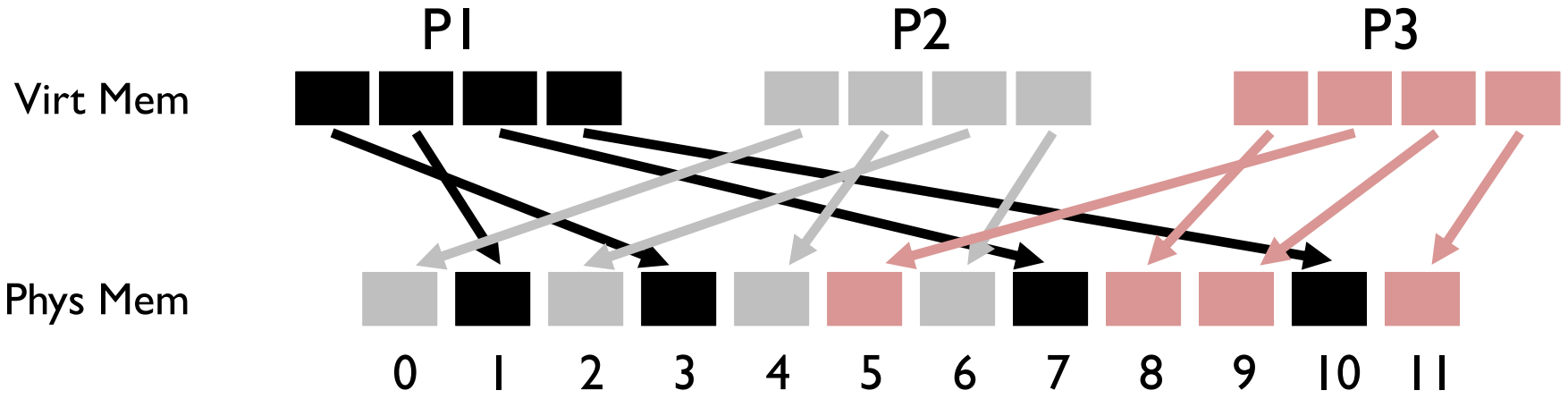




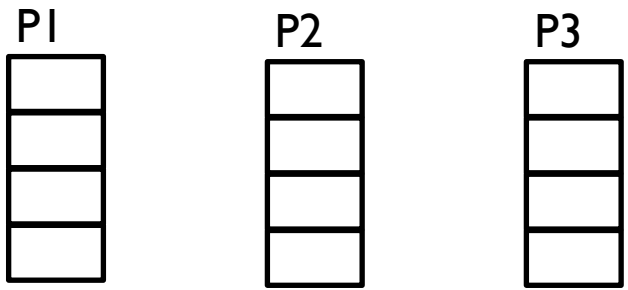
PER-PROCESS PAGETABLE



FILL IN PAGETABLE



Page Tables:



QUIZ 9

https://tinyurl.com/cs537-sp20-quiz9

Description

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

Name of approach

Candidates: Segmentation, Static Relocation, Base, Base+Bounds, Time Sharing

QUIZ9: HOW BIG IS A PAGETABLE?

Consider a **32-bit** address space with 4 KB pages. Assume each PTE is 4 bytes

How many bits do we need to represent the offset within a page?

How many virtual pages will we have in this case?

What will be the overall size of the page table?

WHERE ARE PAGETABLES STORED?

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 PAGETABLE 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 PAGING

14 bit addresses

```
0x0010: movl  0x1100, %edi
```

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

Fetch instruction at logical addr 0x0010

Access page table to get ppn for vpn 0

Mem ref 1:

Learn vpn 0 is at ppn ___

Fetch instruction at _____ (Mem ref 2)

Exec, load from logical addr 0x1100

Access page table to get ppn for vpn 1

Mem ref 3:

Learn vpn 1 is at ppn ___

Movl from _____ into reg (Mem ref 4)

MEMORY ACCESSES WITH PAGING

14 bit addresses

```
0x0010: movl  0x1100, %edi
```

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

Fetch instruction at logical addr 0x0010

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

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)

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

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

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 ?

SUMMARY: PAGE TRANSLATION STEPS

For each mem reference:

1. extract **VPN** (virt page num) from **VA** (virt addr)
2. calculate addr of **PTE** (page table entry)
3. read **PTE** from memory
4. extract **PFN** (page frame num)
5. build **PA** (phys addr)
6. read contents of **PA** from memory into register

Which steps are expensive?