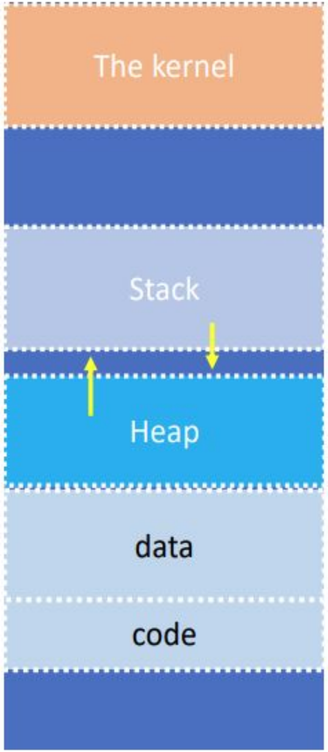
ILLINOIS TECH

College of Computing

CS 450 Operating Systems
Intro to Paging

Yue Duan

Recap



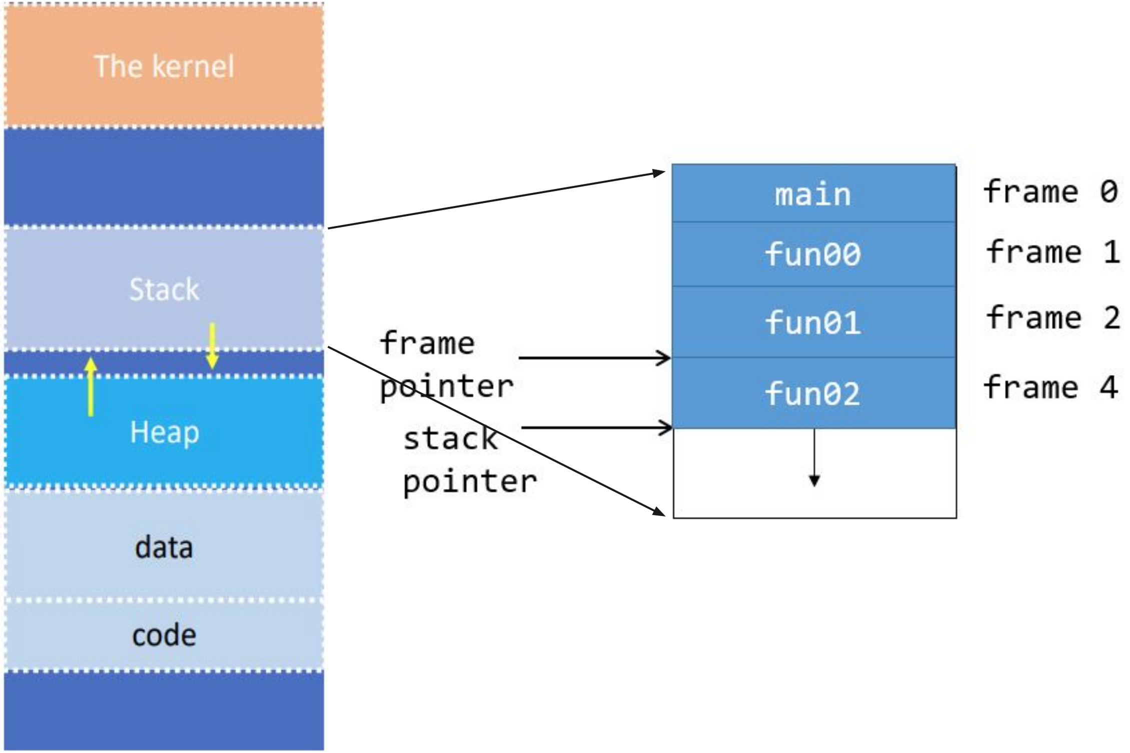
stack segment: grows downward; contains local variables, arguments to functions, return values, etc

heap segment: grows upward, contains malloc'd data, dynamic data structures

data segment: where global variables live

text segment: where instructions live

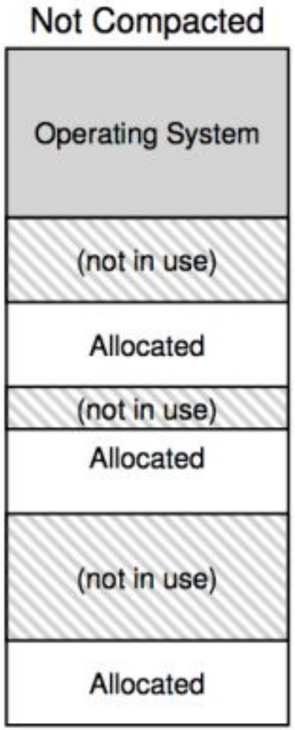
Recap

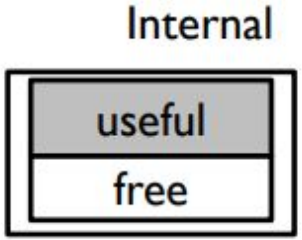



```c
void fun02(int a) {
    int z = 10;
}

void fun01(int b) {
    int y = 20;
    fun02(y);
}

void fun00(int c) {
    int x = 20;
    fun01(x);
}
```

Fragmentation

- Definition: Free memory that can’t be usefully allocated
- Types of fragmentation
  - External:
    - Visible to allocator (e.g., OS)
  - Internal:
    - Visible to requester



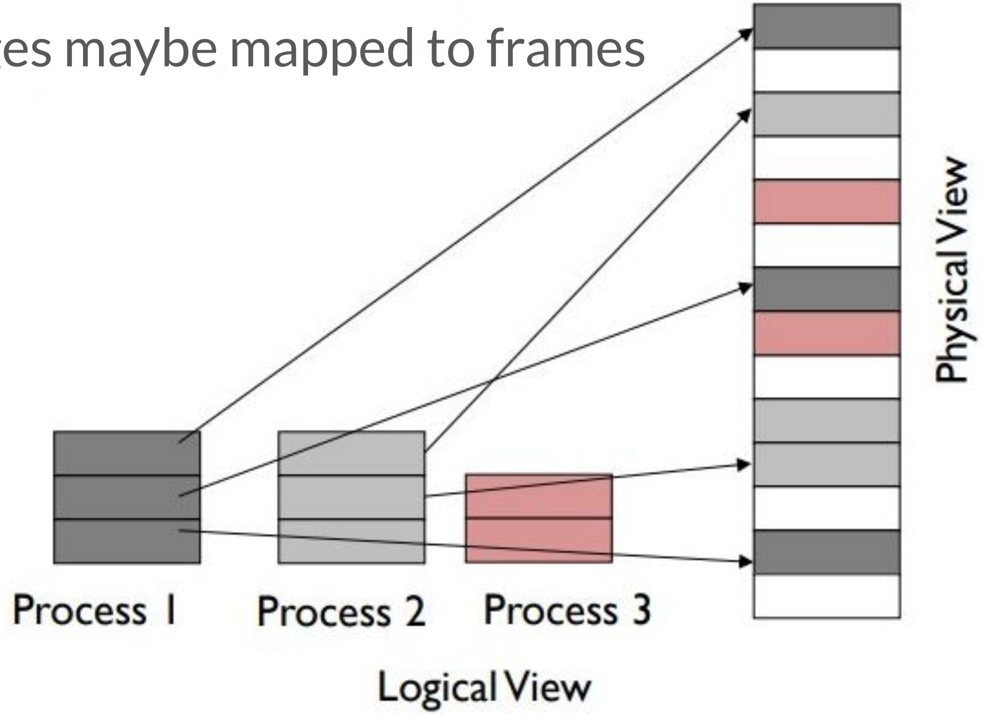


Paging

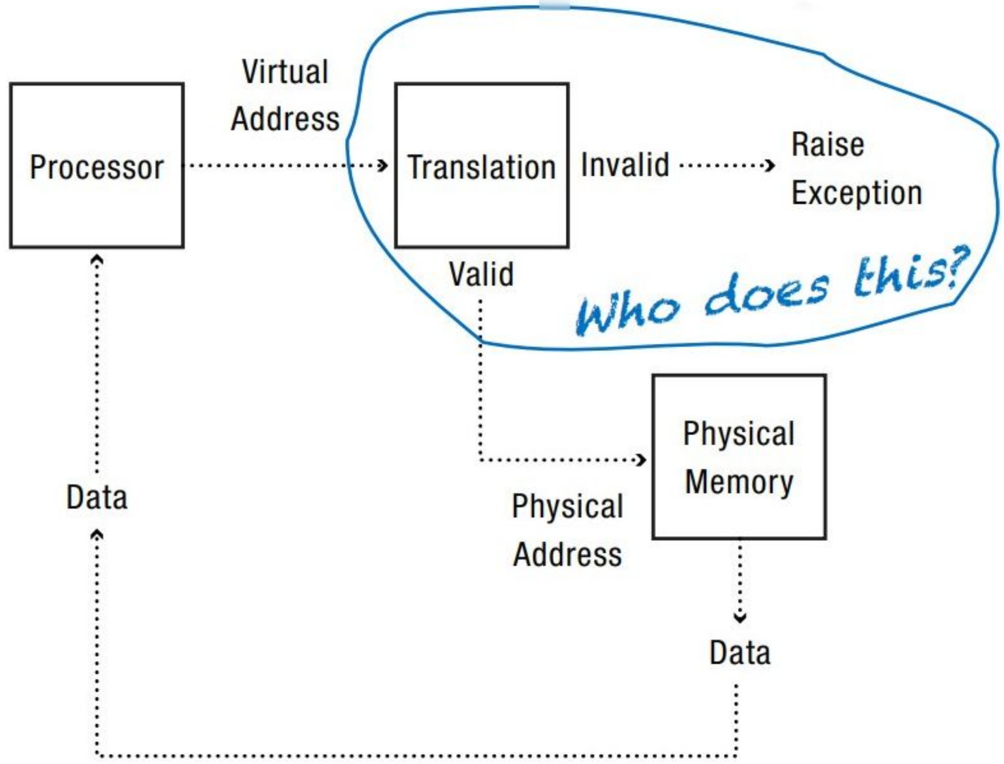
- Goal:
  - Eliminate requirement that address space is contiguous
  - Eliminate external fragmentation
  - Grow segments as needed
- Idea:
  - Physical memory into fixed-sized blocks called **frames**
  - Virtual memory into blocks of same size called **pages**
- Management:
  - Keep track of which pages are mapped to which frames
  - Keep track of all free frames

Paging

Note: Not all pages maybe mapped to frames



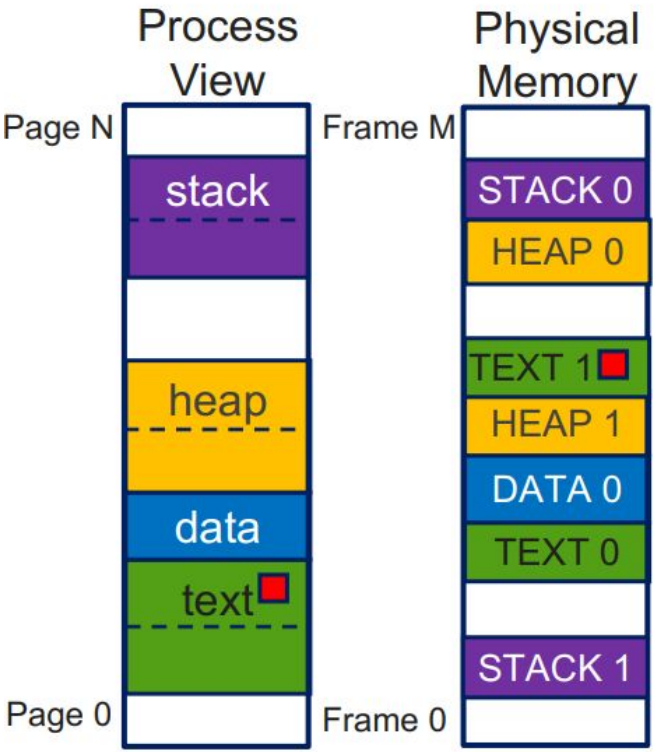
Address Translation



Memory Management Unit (MMU)

- Hardware device
- Maps virtual to physical address (used to access data)
- User Process:
  - Deals with virtual addresses
  - Never sees the physical address
- Physical Memory:
  - Deals with physical addresses
  - Never sees the virtual address

High-Level Address Translation



- red cube is 255th byte in page 2
- where is the red cube in physical memory?

Virtual Address Components

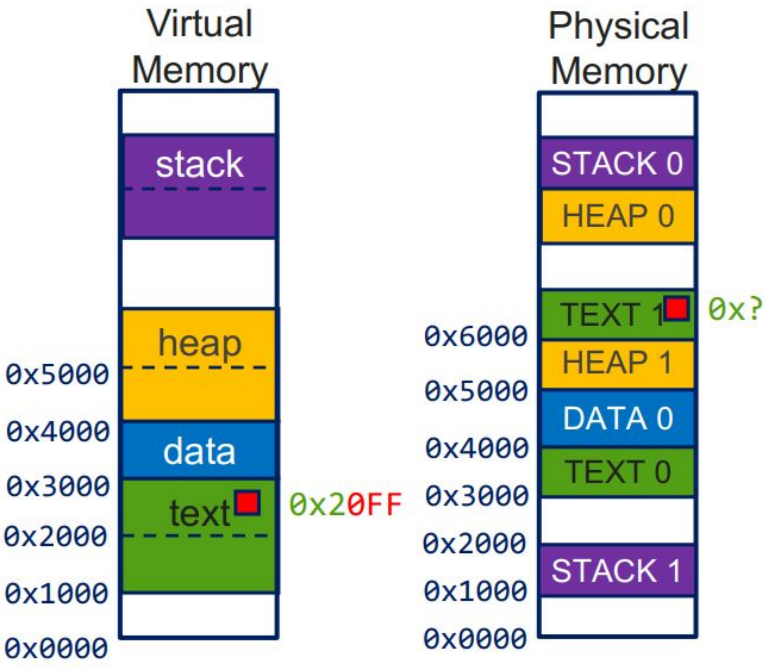
- Page number – Upper bits
  - Must be translated into a physical frame number
- Page offset – Lower bits
  - Does not change in translation

page number	page offset
$m - n$	n

For given logical address space 2^m and page size 2^n

High-Level Address Translation

Who keeps track of the mapping?

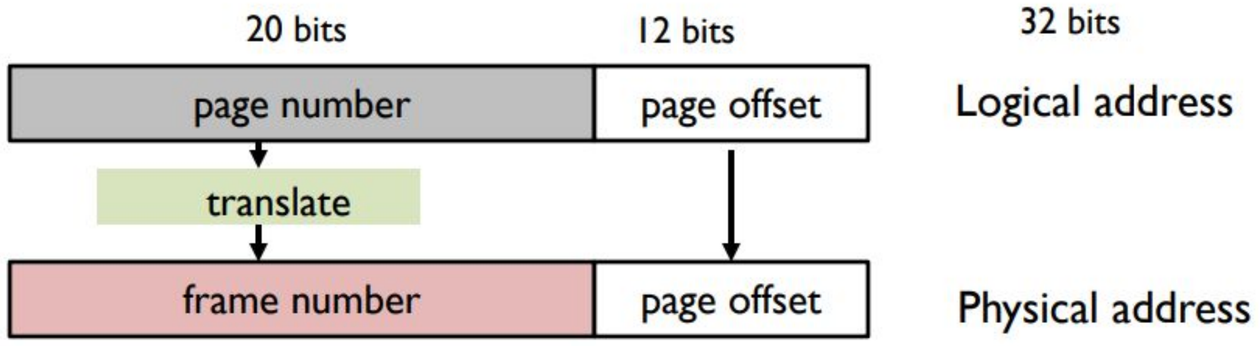


→ Page Table

0	-
1	3
2	6
3	4
4	8
5…	5

High-Level Address Translation

- 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



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	

Page Size	Low Bits(offset)
16 bytes	4
1 KB	10
1 MB	20
512 bytes	9
4 KB	12

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	

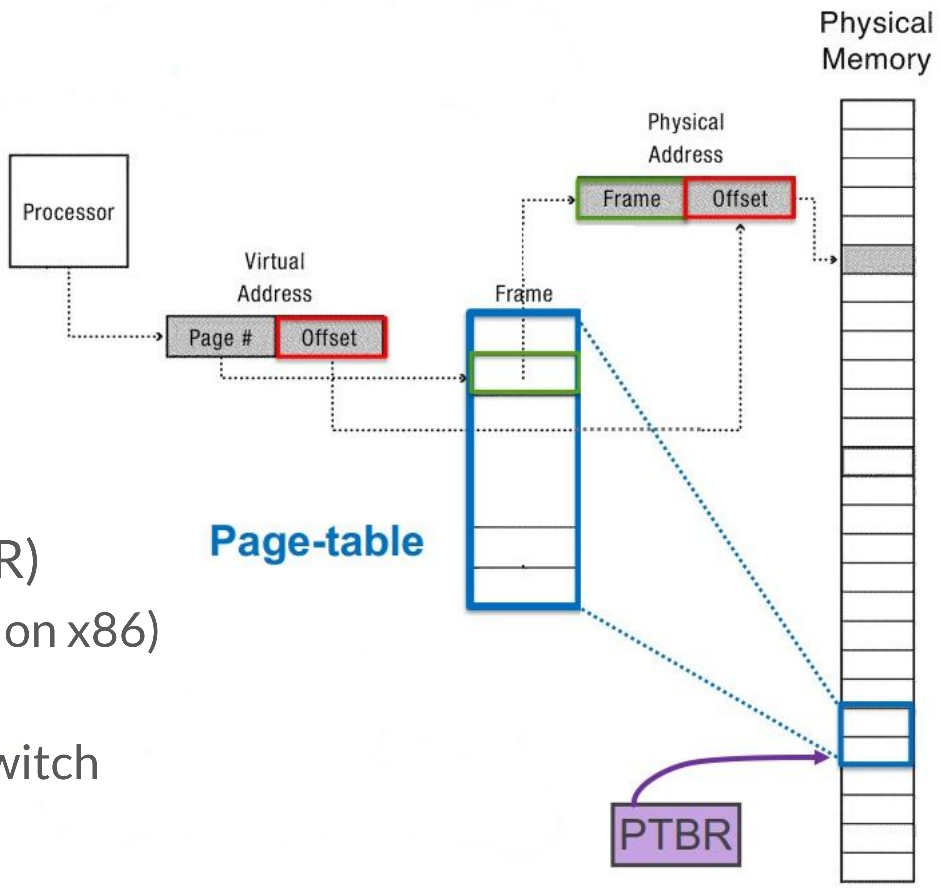
Page Size	Low Bits (offset)	Virt Addr Bits	High Bits (vpn)
16 bytes	4	10	6
1 KB	10	20	10
1 MB	20	32	12
512 bytes	9	16	7
4 KB	12	32	20

Address Format

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

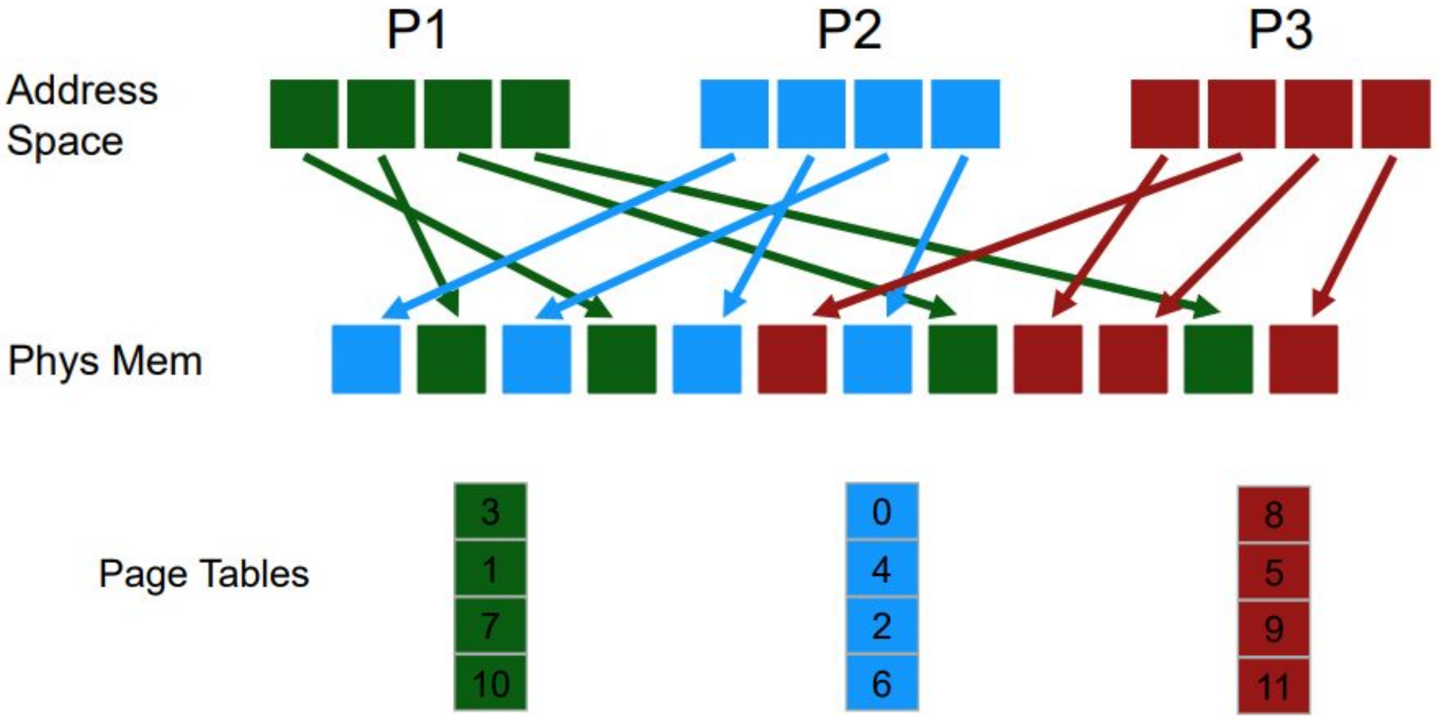
Page Size	Low Bits (offset)	Virt Addr Bits	High Bits (vpn)	Virt Pages
16 bytes	4	10	6	
1 KB	10	20	10	
1 MB	20	32	12	
512 bytes	9	16	7	
4 KB	12	32	20	

Simple Page Table



- Lives in Memory
- Page-table base register (PTBR)
  - dedicated register (e.g., CR3 on x86)
  - Points to the page table
  - Saved/restored on context switch

Example: Fill in the Page Tables



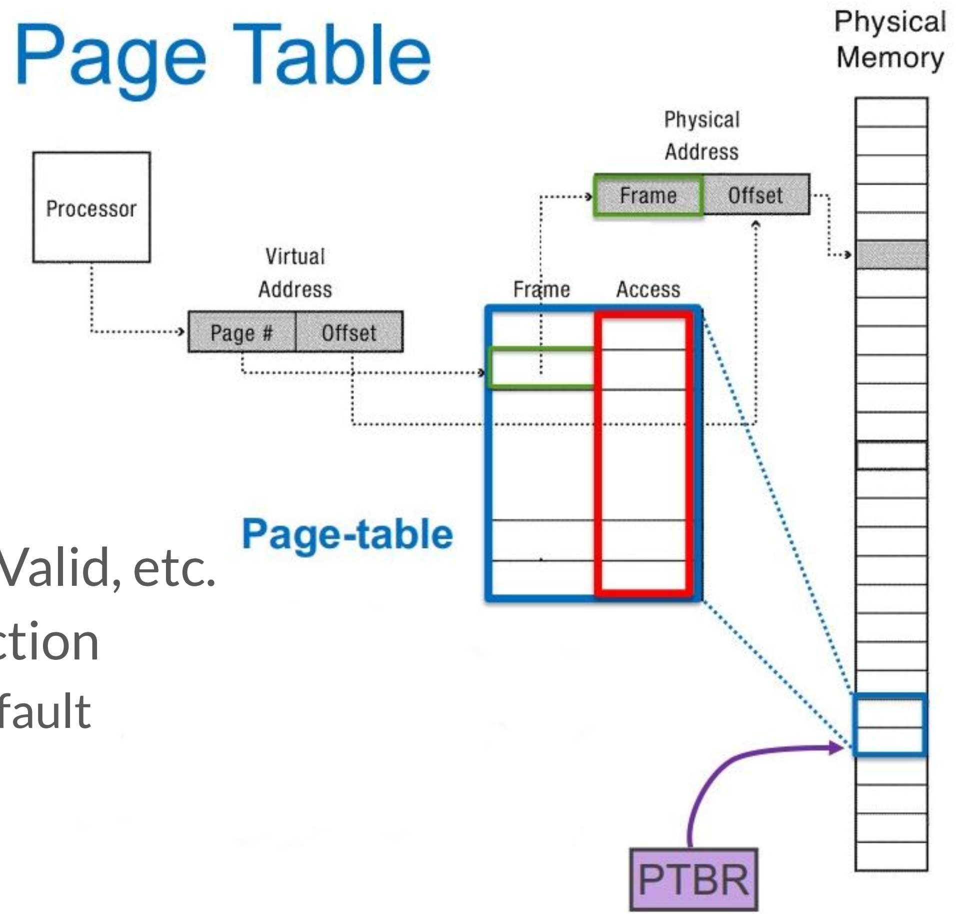
Storing Page Tables

- How big is a typical page table?
  - Assume 32-bit address space, 4KB pages and 4 byte PTEs
- 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 – log(4KB) = 32 – 12 = 20
  - Num entries = 2^20 = 1 MB
  - Page table size = Num entries * 4 bytes = 4 MB

Page Table Entries (PTE)

- PTE Info
  - PFN
  - valid bit
  - protection bit
  - present bit
  - referenced bit
  - dirty bit
- Page table entries are just bits stored in memory
  - Agreement between HW and OS about interpretation

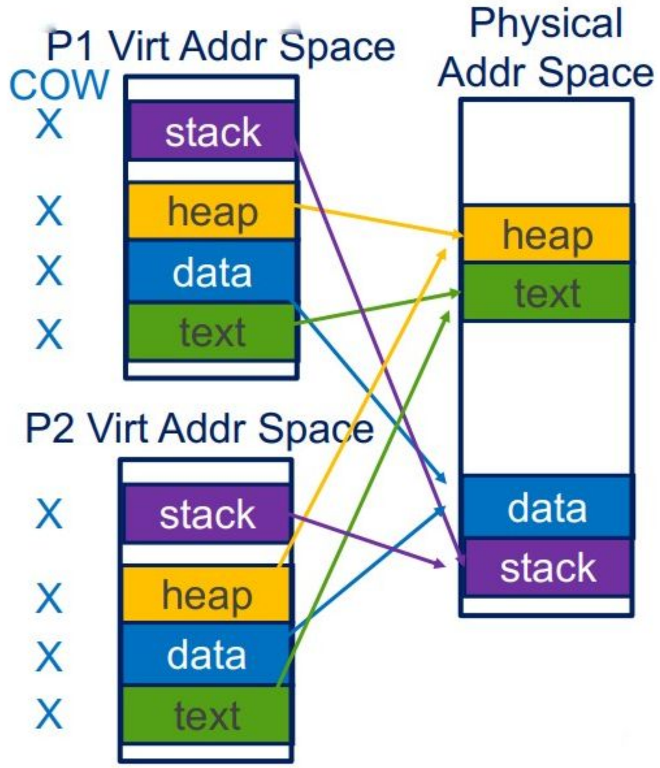
Protection



- Meta Data about each frame
- Protection R/W/X, Modified, Valid, etc.
- MMU Enforces R/W/X protection
  - illegal access throws a page fault

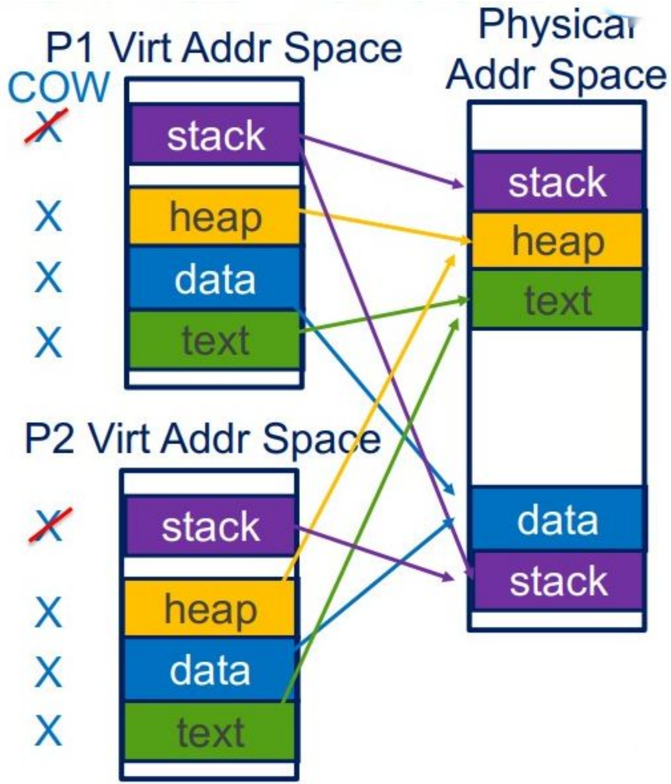
Copy-On-Write (COW)

- P1 forks()
- P2 created with
  - own page table
  - same translations
- All pages marked **COW** (in Page Table)



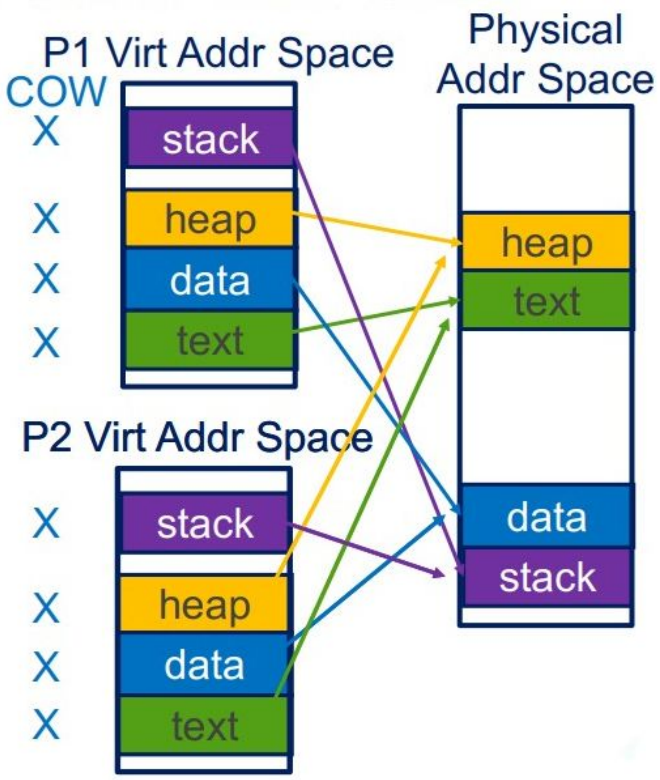
Option 1: fork, then keep executing

- Now P1 tries to write to the stack (for example):
  - Page fault
  - Allocate new frame
  - Copy page
  - Both pages no longer **COW**



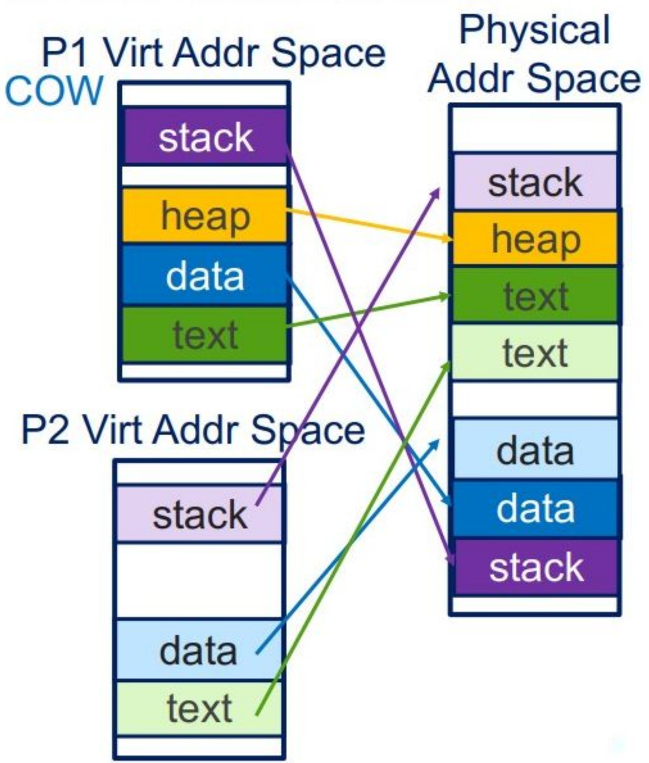
Option 2: fork, then call exec

- Before P2 calls exec()



Option 2: fork, then call exec

- After P2 calls exec()
  - Allocate new frames
  - Load in new pages
  - Pages no longer **COW**



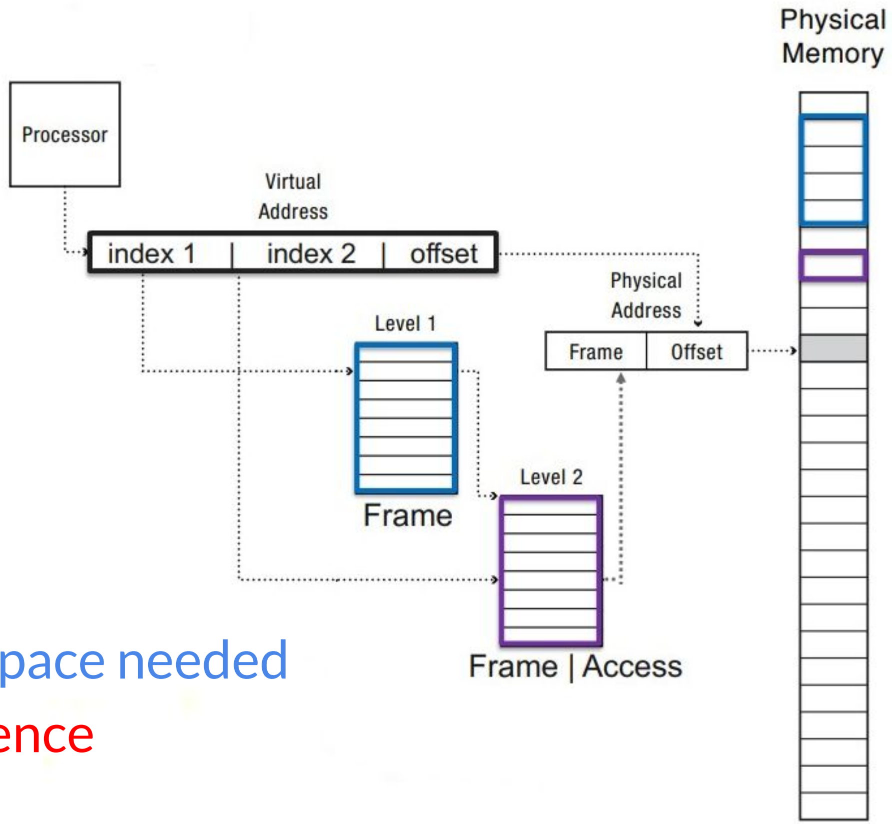
Advantages of Paging

- Easily accommodates transparency, isolation, protection and sharing
- No external fragmentation
- Fast to allocate and free page frames
  - Alloc: No searching for suitable free space; pick the first free page frame
  - Free: Doesn’t have to coalesce with adjacent free space; just add to the list of free page frames
  - Simple data structure (bitmap, linked list, etc.) to track free/allocated page frames

Disadvantages of Paging

- Internal fragmentation
  - Page size may not match size needed by process
  - Make pages smaller? But then…
- Storage overhead: page tables may be large
  - Simple page table: requires PTE for all pages in address space
  - Entry needed even if page not allocated ?
- Inefficiency: memory references to page table
  - Page table must be stored in memory
  - MMU stores only base address of page table

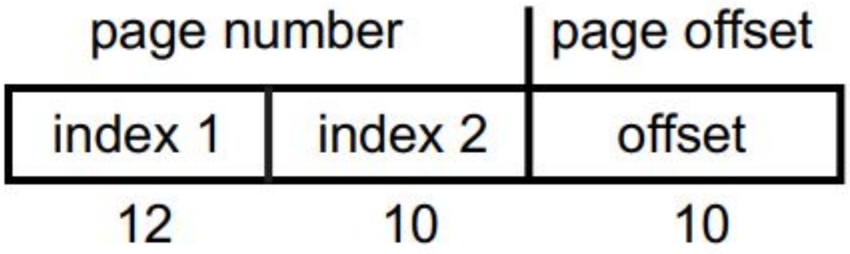
Multi-Level Page Tables



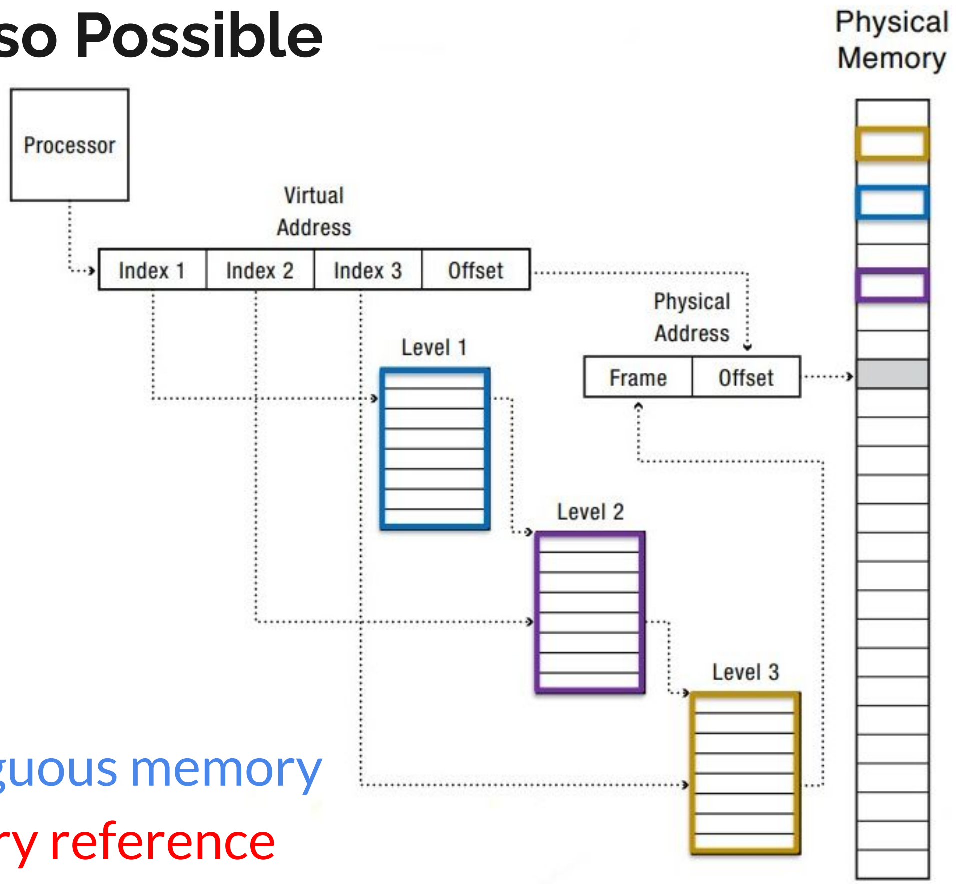
- + Allocate only PTEs in use
- + Simple memory allocation
  - no large continuous memory space needed
- − more lookups per memory reference

Two-Level Paging Example

- Assume 32-bit machine, 1KB page size
- Logical address is divided into:
  - a page offset of 10 bits (1024 = 2^10)
  - a page number of 22 bits (32-10)
- The page number is further divided into:
  - a 12-bit first index
  - a 10-bit second index



Three-Level Paging is also Possible



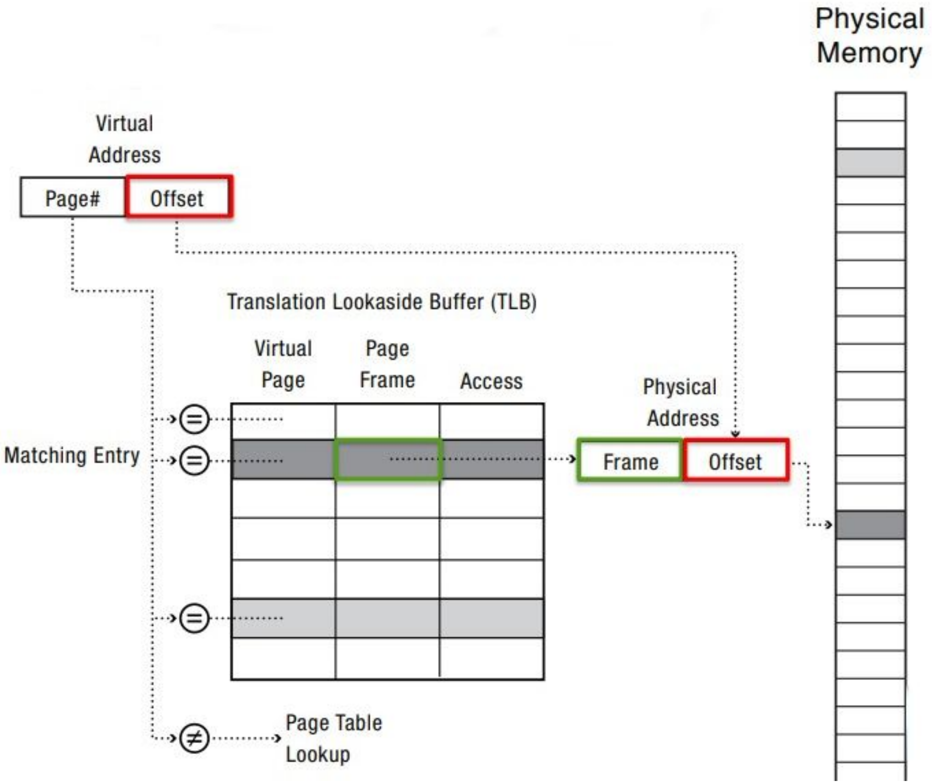
- + First Level requires less contiguous memory
- − even more lookups per memory reference

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

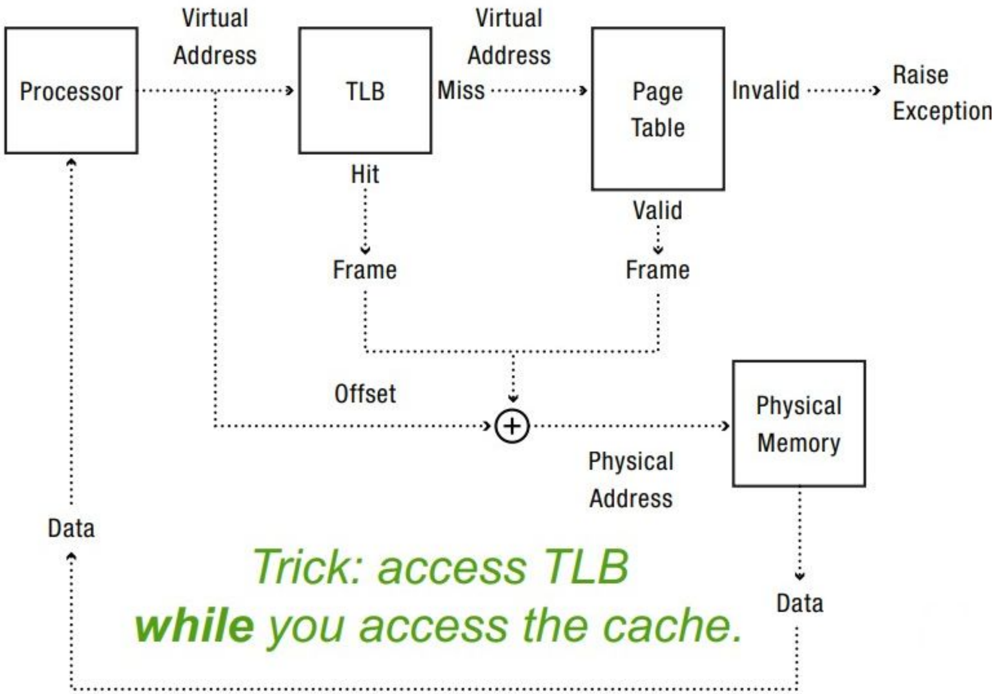
Strategy: Cache Page Translations

- TLB: Translation Lookaside Buffer
    - part of MMU
    - Associative cache of virtual to physical page translations



Address Translation with TLB

- Access TLB before you access memory.



THANK YOU!