



ILLINOIS TECH

College of Computing

CS 450 Operating Systems

Multi-level Paging and Swapping

Yue Duan

Disadvantages of Paging



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

Contiguous PTEs

how to avoid storing these?

PFN	valid	prot
10	1	r-x
-	0	-
23	1	rw-
-	0	-
-	0	-
-	0	-
-	0	-
...many more invalid...	0	-
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

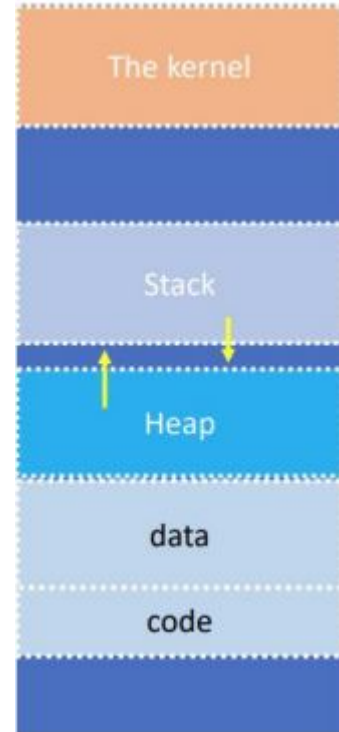
- Note “hole” in addr space
 - valids vs. invalids are clustered
- How did OS avoid allocating holes in phys memory?
 - **Segmentation**

Combine Paging and Segmentation

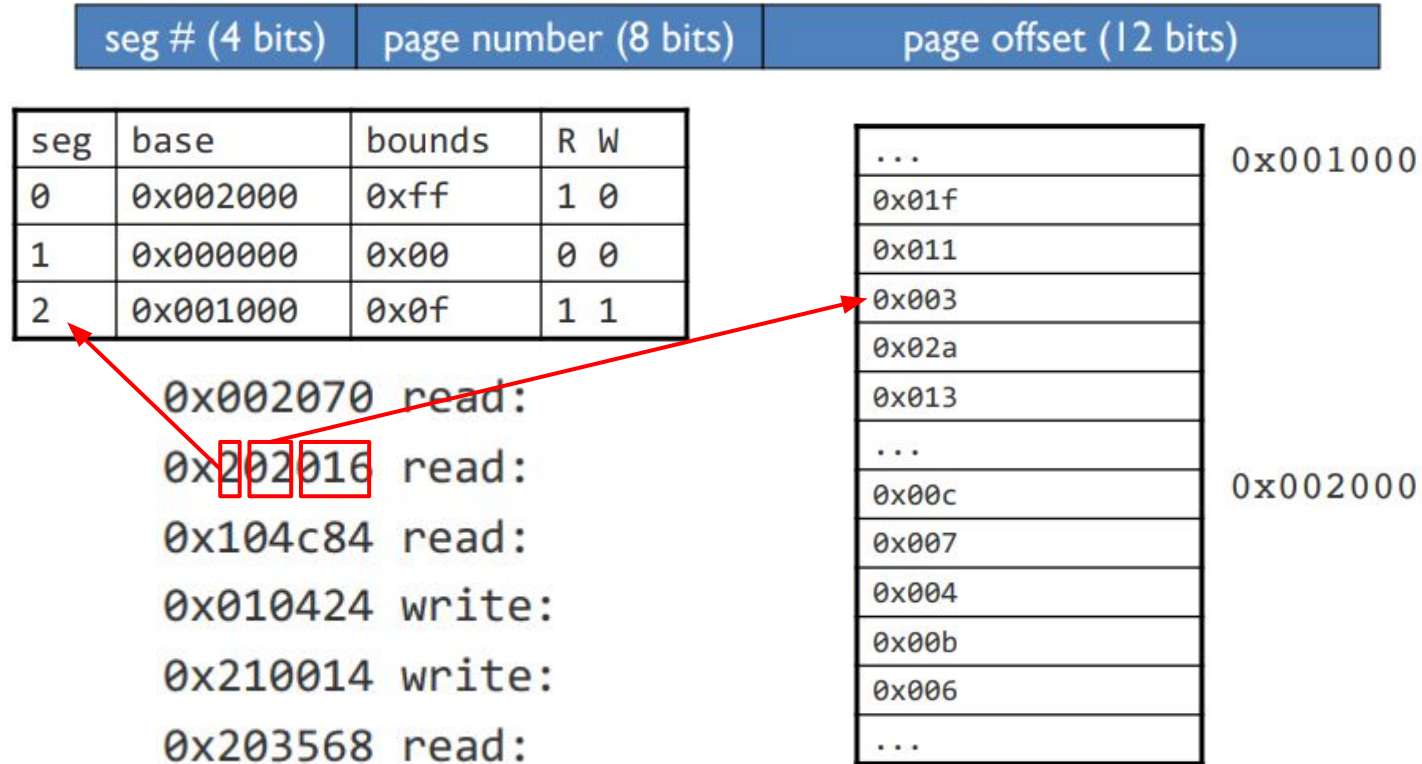
- Divide address space into segments (code, heap, stack)
 - Segments can be variable length
- Divide each segment into fixed-sized pages.
- Logical address divided into three portions



- Implementation
 - **Each segment has a page table**
 - Track base (physical address) and bounds of the **page table** per segment



Combine Paging and Segmentation



Advantages of Paging and Segmentation



- Advantages from using **Segments**
 - decreases size of page tables
 - if segment not used, no need for page table
- Advantages from using **Pages**
 - no external fragmentation
 - segments can grow without any reshuffling
- Advantages of using both
 - increases flexibility of sharing
 - share either single page or entire segment

Disadvantages of Paging and Segmentation



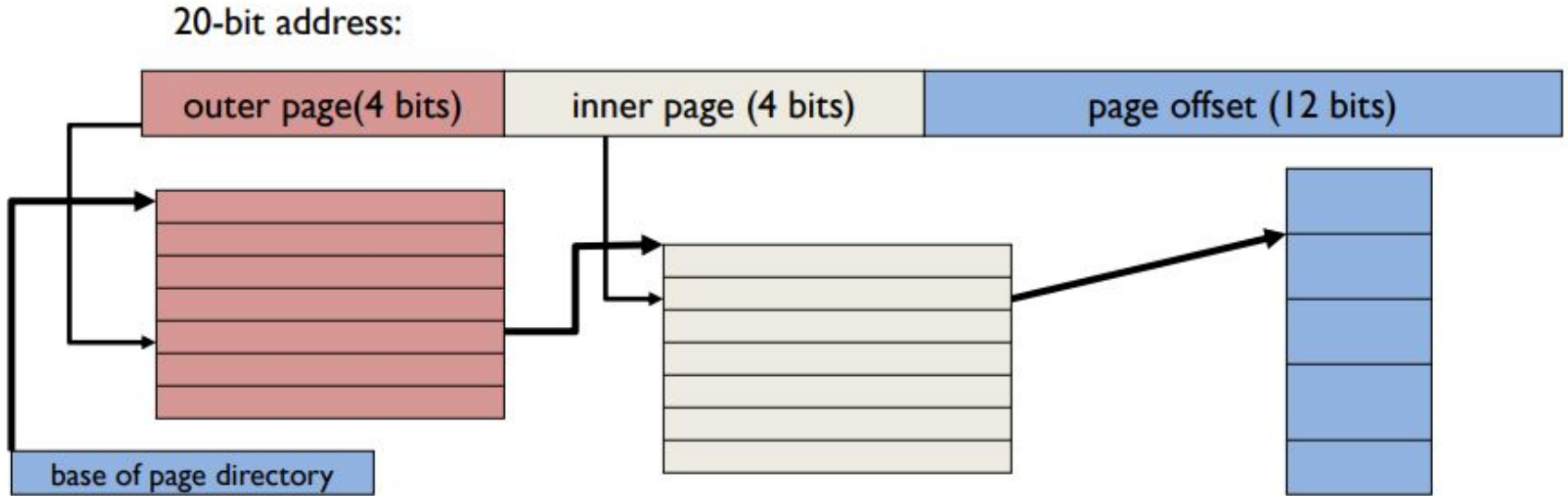
- Potentially large page tables (for each segment)
- Must allocate page table for each segment contiguously
- More problematic with more address bits
- Page table size?
 - Assume 2 bits for segment, 18 bits for vpn, 12 bits for offset
 - Each page table is:
 - = Number of entries * size of each entry
 - = Number of pages * 4 bytes = $2^{18} * 4$ bytes = 2^{20} bytes
 - = 1 MB!

Multi-level Paging



- Goal: Allow page table to be allocated non-contiguously
- Idea: Page the page tables
 - Creates multiple levels of page tables; outer level “page directory”
 - Only allocate page tables for pages in use
 - Used in x86 architectures (hardware can walk known structure)

Multi-level Paging



Multi-level Paging



- How should logical address be structured?
- How many bits for each paging level?
- Goal?
 - Each inner page table fits within a page
 - $\text{PTE size} * \# \text{ PTE} = \text{page size}$
 - Assume PTE size = 4 bytes, Page size = 2^{12} bytes = 4KB
 - # bits for selecting inner page = **10 bits**
- Remaining bits for outer page: **$30 - 12 - 10 = 8$ bits**

Multi-level Paging

20-bit address:



page directory

PPN	valid
0x3	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
0x92	1

page of PT (@PPN:0x3)

PPN	valid
0x10	1
0x23	1
-	0
-	0
0x80	1
0x59	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

page of PT (@PPN:0x92)

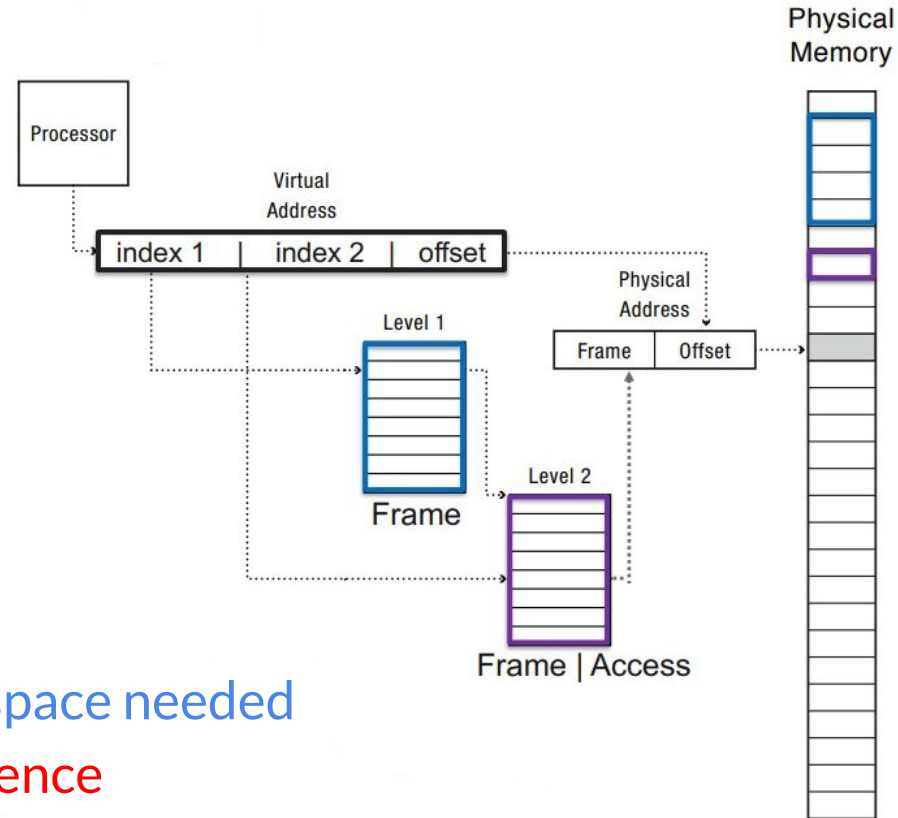
PPN	valid
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
0x55	1
0x45	1

translate 0x01ABC

- extract page directory number 0x0
- go to page table at 0x3
- extract inner page number 0x1
- get PPN 0x 23
- PA: 0x23ABC

Two-Level Page Tables

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



Problem with Two-Levels

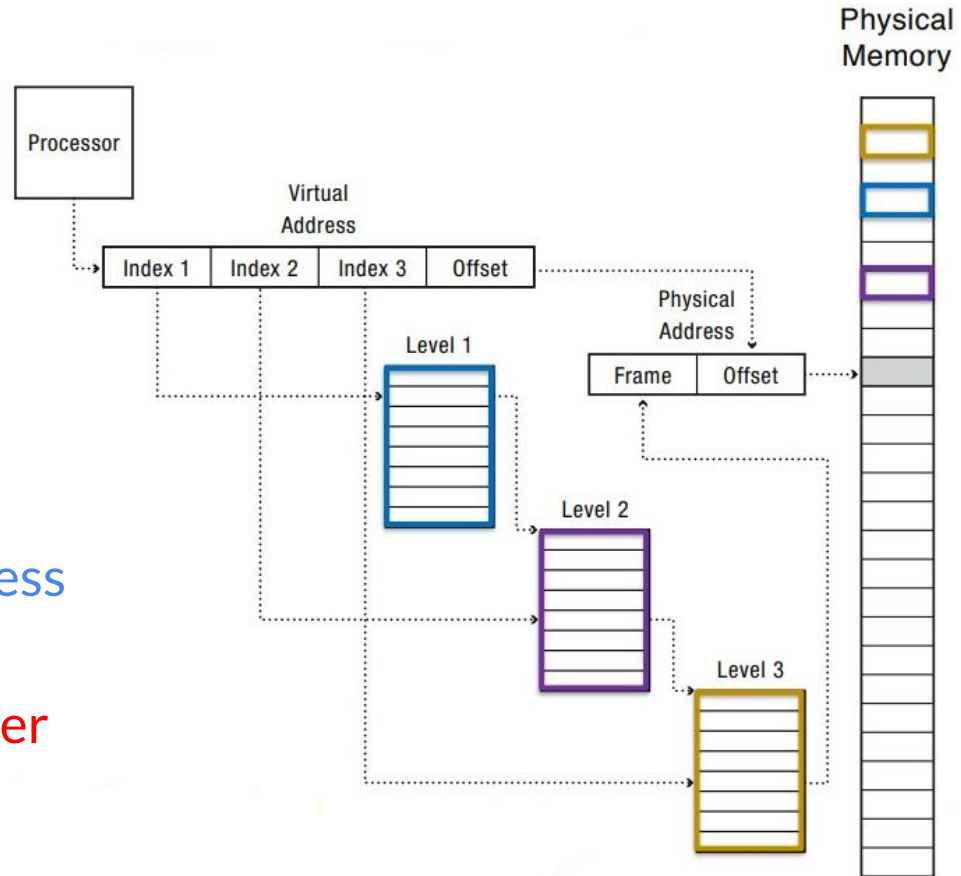
64-bit address



- page directories (outer level) may not fit in a page
- Solution: add more levels
 - Split page directories into pieces
 - Use a higher-level page dir to refer to the page dir pieces

Three-Level Paging

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



Inverted Page Table



- Only store entries for virtual pages w/ valid physical mappings
- Naïve approach:
 - Search through data structure to find match
 - Too much time to search entire table
- Better:
 - Find possible matches entries by hashing vpn+asid
 - Smaller number of entries to search for exact match
- Managing inverted page table requires **software-controlled TLB**

Summary: Multi-level Paging



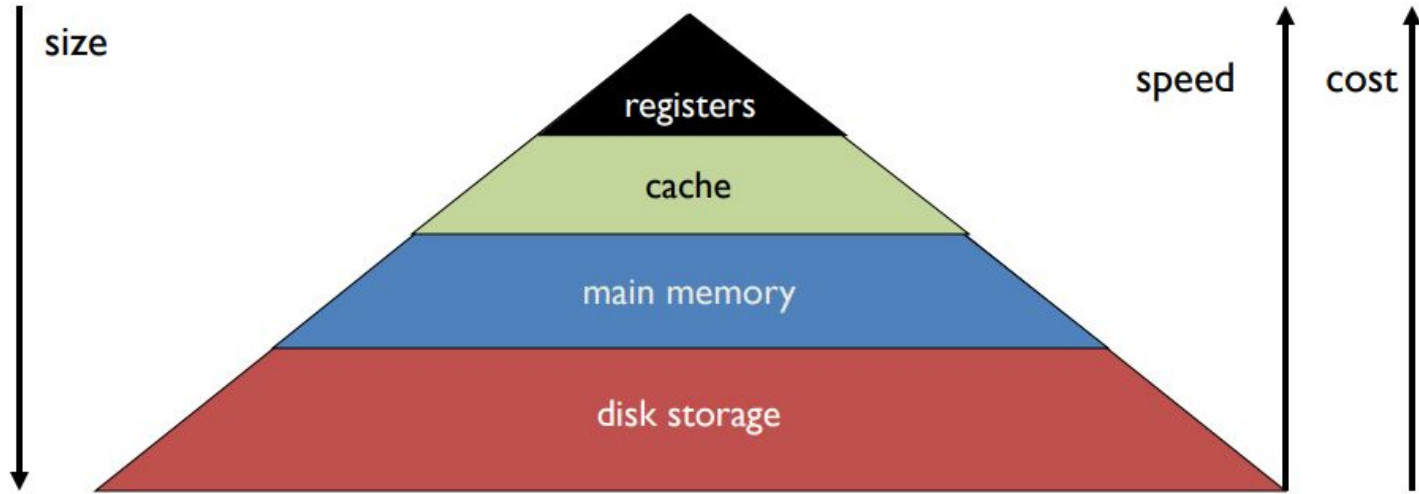
- Problem: Simple linear page tables require too much contiguous memory
- Many options for efficiently organizing page tables
- If OS traps on TLB miss, OS can use any data structure
 - Inverted page tables (hashing)
- If Hardware handles TLB miss, page tables must follow specific format
 - Multi-level page tables used in x86 architecture
 - Each page table fits within a page

Swapping



- OS goal: Support multiple processes
 - Single process with very large address space
 - Multiple processes with combined address spaces
- User code should be independent of amount of physical memory
 - Correctness, if not performance
- Virtual memory: OS provides illusion of more physical memory
- Reality: many processes, limited physical memory

Memory Hierarchy



- Leverage memory hierarchy of machine architecture
- Each layer acts as “backing store” for layer above

Swapping Intuition



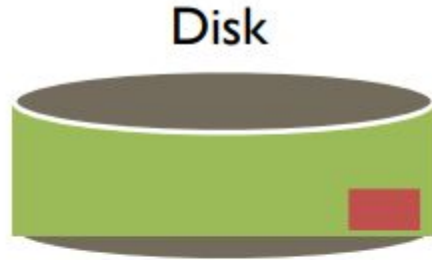
- Idea: OS keeps unreferenced pages on disk
 - Slower, cheaper backing store than memory
- Process can run when not all pages are loaded into main memory
- OS and hardware cooperate to make large disk seem like memory
 - Same behavior as if all of address space in main memory
- Requirements:
 - OS must have **mechanism** to identify location of each page in address space ==> in memory or on disk
 - OS must have **policy** to determine which pages live in memory and which on disk

Virtual Address Space Mechanisms



- Each page in virtual address space maps to one of three locations:
 - Physical main memory: Small, fast, expensive
 - Disk (backing store): Large, slow, cheap
 - Nothing (error): Free
- Extend page tables with an extra bit: present
 - permissions (r/w), valid, present
 - Page in memory: present bit set in PTE
 - Page on disk: present bit cleared
 - PTE points to block on disk
 - Causes trap into OS when page is referenced
 - **Trap: page fault**

Virtual Address Space Mechanisms



Phys Memory



PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28	1	rw-	0
4	1	rw-	1

What if access vpn 0xb?

Virtual Address Space Mechanisms



- First, hardware checks TLB for virtual address
- if TLB hit, address translation is done; page in physical memory
- Else
 - Hardware or OS walk page tables
 - If PTE designates page is present, then page in physical memory page fault (i.e., present bit is cleared)
 - Else
 - Trap into OS (not handled by hardware)
 - OS selects victim page in memory to replace
 - Write victim page out to disk if modified (use dirty bit in PTE)
 - OS reads referenced page from disk into memory
 - Page table is updated, present bit is set
 - Process continues execution

Swapping Policies



- Goal: Minimize number of page faults
 - Page faults require milliseconds to handle (reading from disk)
 - Implication: Plenty of time for OS to make good decision
- OS has two decisions
 - Page selection
 - When should a page (or pages) on disk be brought into memory?
 - Page replacement
 - Which resident page (or pages) in memory should be thrown out to disk?



THANK YOU!