운영체제의 기초:
Segmentation and Paging

2023년 5월 4, 9, 11, 16일

홍 성 수
sshong@redwood.snu.ac.kr
SNU RTOSLab 지도교수
서울대학교 전기정보공학부 교수

Agenda

I. Segmentation
II. Paging
III. Paged Segmentation
IV. Enhancing Mechanisms

I. Segmentation

Motivations

- Problem with base and bound relocation
  - *Only one segment per process*
    - How can two processes share code while keeping private data areas (e.g., shared editor)?

Solutions (1)

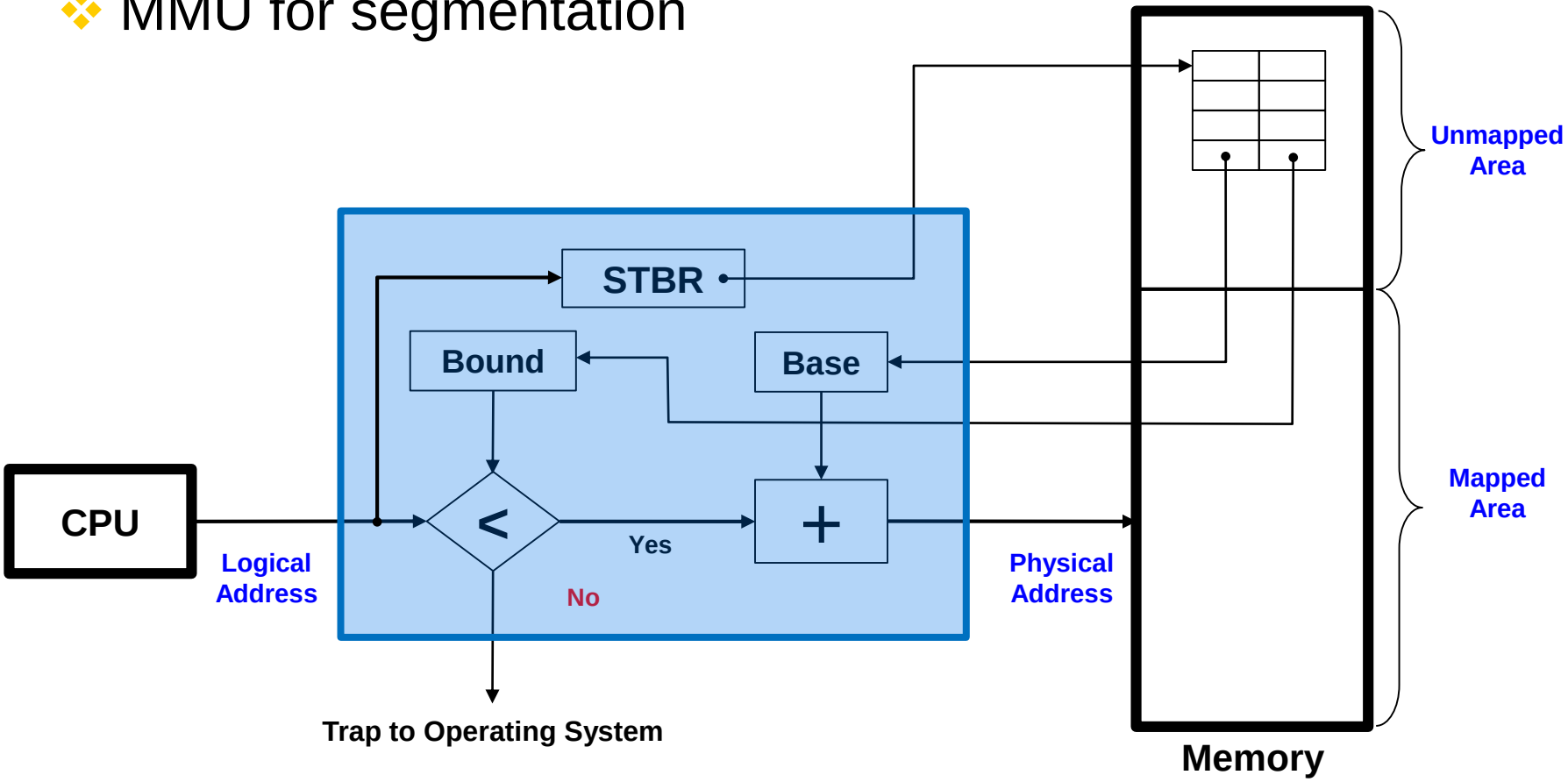
- Multiple segments
  - Permit process to be split between several areas of memory
  - Use a separate base and bound register pair for each segment, and also add two protection bits (read and write)
  - Each memory reference indicates a segment and offset in one or more of three ways
    - Segment table holds the bases and bounds for all the segments of process
    - Top bits of address select segment, low bits the offset
    - Segment is selected implicitly by the instruction
      - Examples: Code vs. data, stack vs. data, or 8086 prefixes

Solutions (2)

- Multiple segments (cont'd)
  - Address mapping gets more complicated
    - Memory mapping procedure consists of "table lookup" + "add" + "compare"
    - Example
      - PDP–10 with high and low segments selected by the high-order address bit
        - Addresses with a 0 top bit used one base register, and higher addresses use another

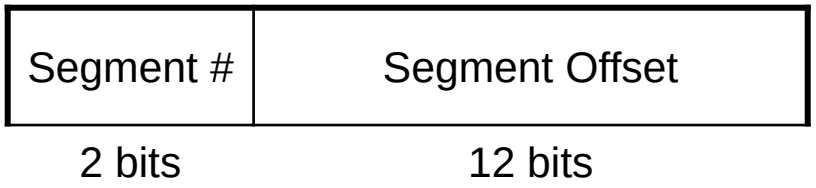
Address Translation (1)

- MMU for segmentation



Address Translation (2)

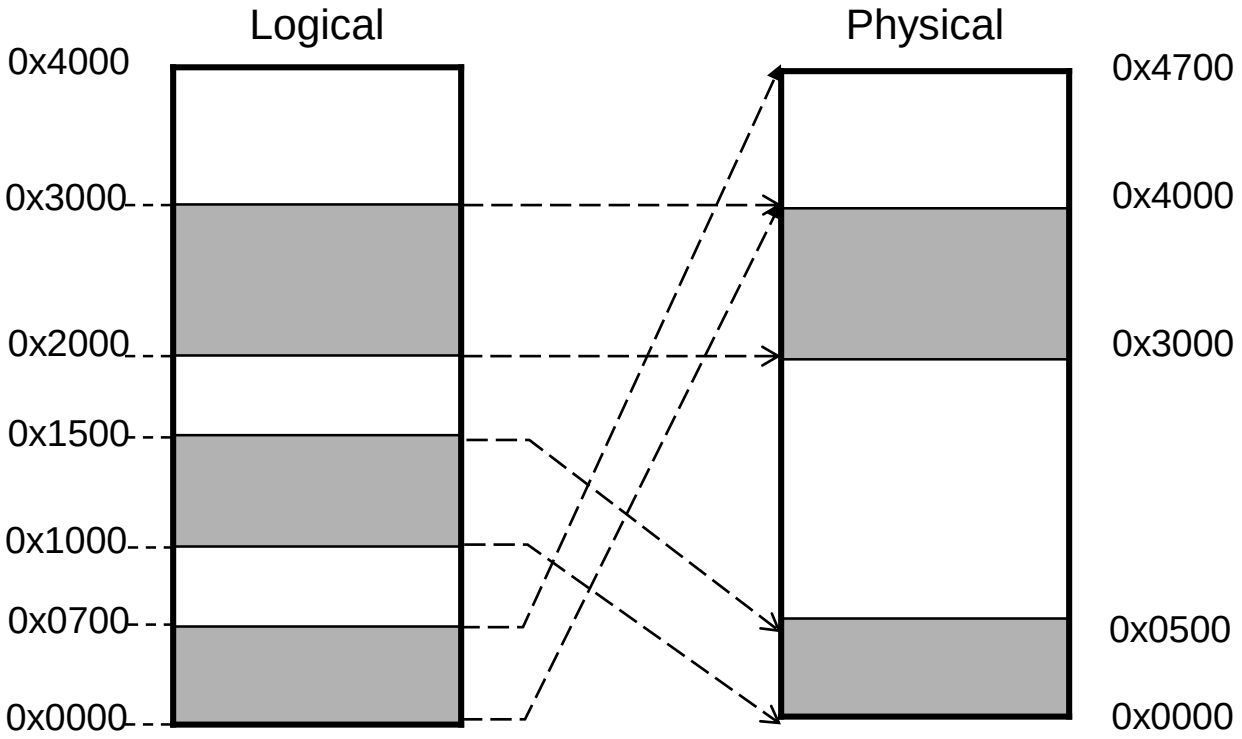
- Segmentation example
  - 2 bits of segment number, 12 bits of offset



Segment	Base	Bounds	RW
0	0x4000	0x6FF	10
1	0x0000	0x4FF	11
2	0x3000	0xFFF	11
3			00

Address Translation (3)

- Address space mapping



- Where is 0x240, 0x1180, 0x265c?
- How about 0x3002, 0x1600?

Address Translation (4)

- Address translation examples
  - 0x240
    - 0x0:240 – seg # 0, offset 0x240 → 0x4240
  - 0x1180
    - 0x1:180 – seg # 1, offset 0x180 → 0x0180
  - 0x3002
    - 0x3:002 – seg # 3, offset 002 → invalid address (used)

Segment	Base	Bounds	RW
0	0x4000	0x6FF	10
1	0x0000	0x4FF	11
2	0x3000	0xFFF	11
3			00

Supporting Operations (1)

- Managing segments
  - Keep copy of segment table in process control block
  - When creating process, allocate space for segment and fill in PCB bases and bounds
  - When process dies, return segments to free pool

- When there's no space to allocate a new segment
  - Compact memory (move all segments, update bases) to get all free space together
  - Or, swap one or more segments to disks to make space and then bring segments back in before letting process run
    - Must then check during context switching and bring segments back in before letting process run

Supporting Operations (2)

- To enlarge segment
  - See if space above segment is free; if so, just update the bound and use that space
  - Or, move the segment above this one to disk, in order to make the memory free
  - Or, move this segment to disk and bring it back into a larger hole (or, maybe just copy it to a larger hole)

Pros and Cons

- Advantage of segmentation
  - Segments can be swapped and assigned to storage independently
- Problems
  - External fragmentation
    - Segments of many different sizes have to be allocated contiguously
    - This problem also applies to base and bound schemes

II. Paging

Motivations

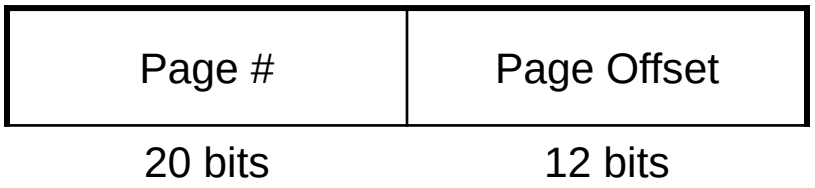
- Problem with segmentation
  - External fragmentation
  - Goal of paging
    - To make allocation and swapping easier
    - To reduce memory fragmentation

Solutions

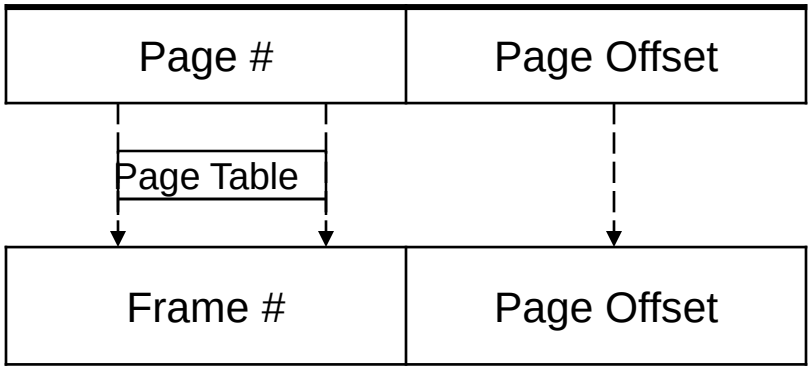
- Paging
  - Make all chunks of memory the same size called pages
    - Typical sizes range from 512 to 16 Kbytes
  - For each process, a page table defines
    - The base address of each of that process' pages and protection (read-only) and existence bits
  - Translation process is introduced

Address Translation (1)

- Example: SPARCstation

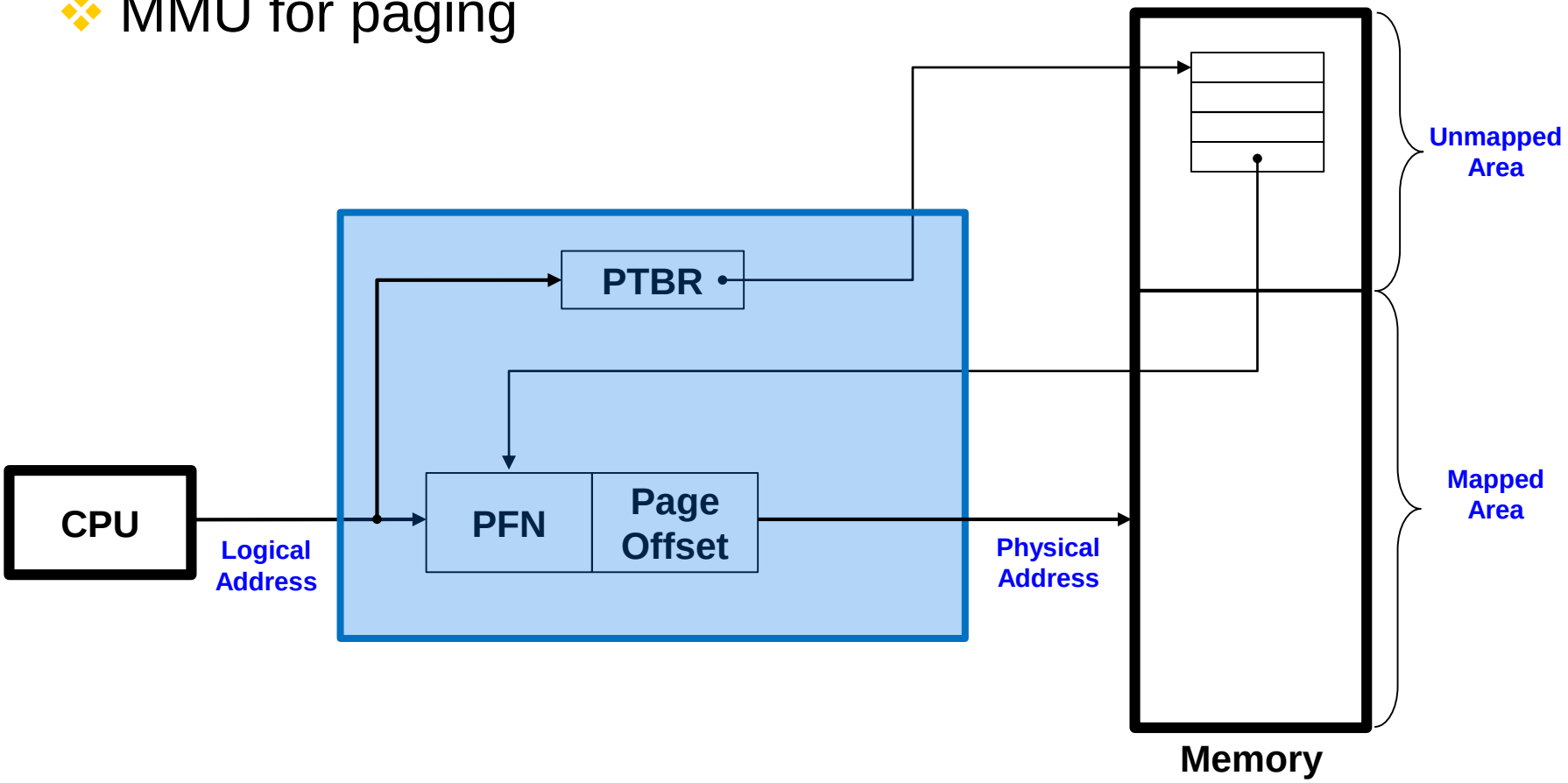


  - Translation process
    - Page number always comes directly from the address



Address Translation (2)

- MMU for paging



Pros and Cons (1)

- Advantage of paging
  - Easy to allocate
    - Keep a free list of available pages and grab the first one
    - Easy to swap since everything is the same size, which is usually the same size as disk blocks to and from which pages are swapped

Pros and Cons (2)

- Problems
  - Efficiency of access
    - Even small page tables are generally too large to get loaded into fast memory in the MMU
    - Page tables are kept in main memory and the MMU has only the page table's base address
    - It thus takes one overhead reference for every real memory reference

Pros and Cons (3)

- Problems (cont'd)
  - Table space
    - If pages are small, the table space could be substantial
      - Example: 32-bit address space with 1 KB pages
        - Page table size = 16 Megabytes
    - What if the whole table has to be present at once?
      - Partial solution: Keep base and bound for page table, so only large processes have to have large tables
    - Internal fragmentation
      - Page size doesn't match up with information size
      - The larger the page, the worse this is

III. Paged Segmentation

Motivations

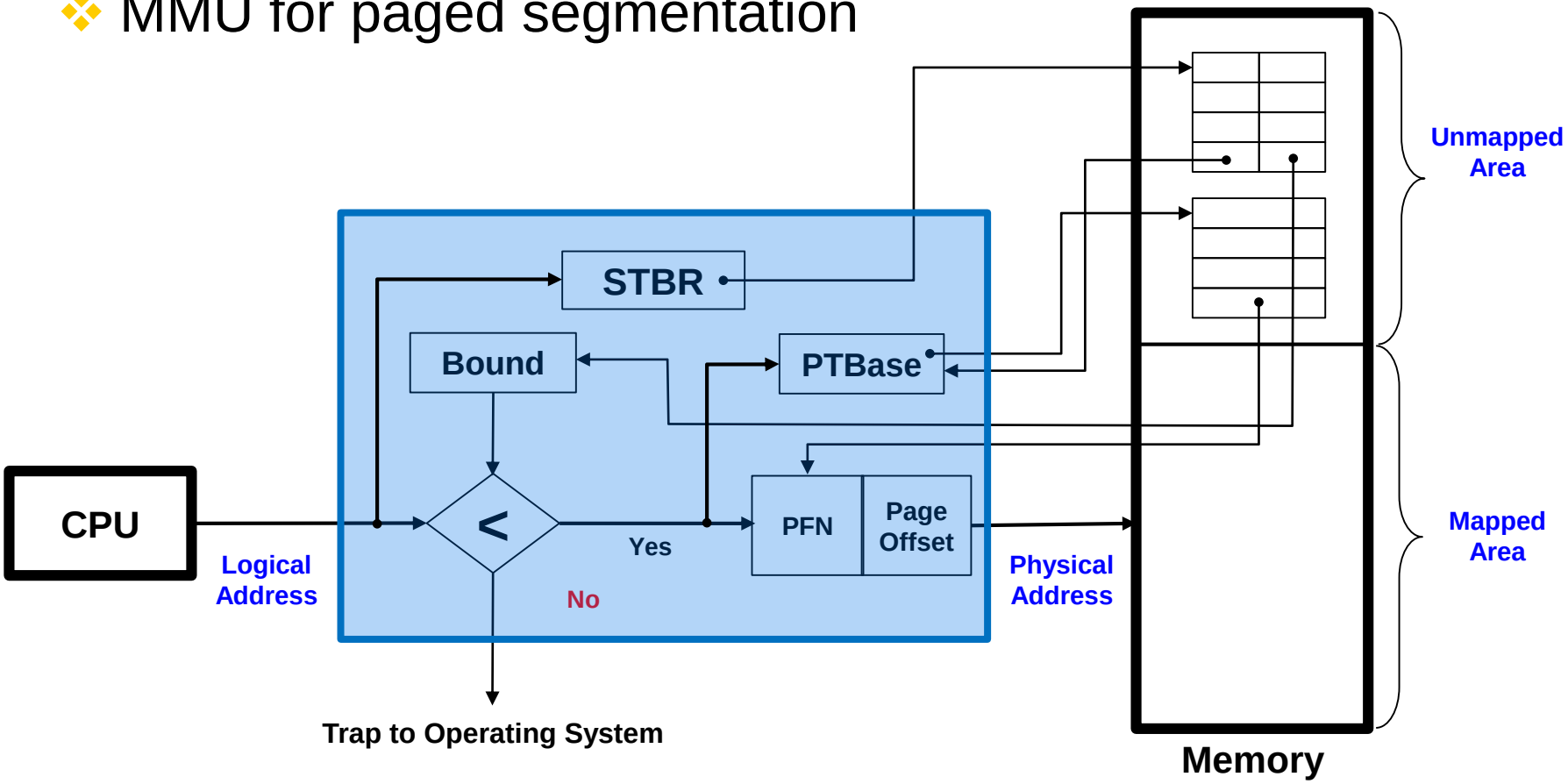
- Problems with paging
  - The same as single segment per process
    - But not always true, either
  - Going from paging to P+S is like going from single segment to multiple segments, except at a higher level

Solutions

- Paging and segmentation combined
  - Instead of having a single page table, have many page tables with a base and bound for each
  - Call the stuff associated with each page table a segment
    - Use two levels of mapping to make tables manageable
    - Each segment contains one or more pages
    - Segments correspond to logical units: Code, data, stack
      - Segments vary in size and are often large
  - Pages are for the use of the OS
    - They are fixed-sized to make it easy to manage memory

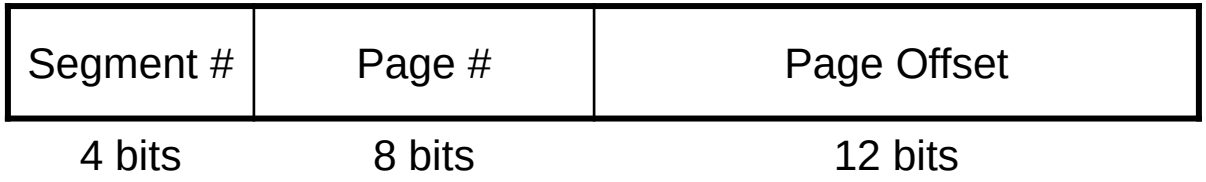
Address Translation (1)

- MMU for paged segmentation



Address Translation (2)

- System 370 example
  - 24-bit logical address space
    - 4 bits of segment number, 8 bits of page number, 12 bits of offset



  - Segment table contains physical address of page table along with the length of the page table (a sort of bounds register for the segment)
    - Page table entries are only 12 bits long occupying 16 bits
    - Real addresses are 24 bits long

Address Translation (3)

- Example of S/370 paging

Segment table		
Base	Bound	Prot
0x2000	0x14	R
0x0000	0x00	
0x1000	0x0D	RW

	Memory
0x2022	0x001F
0x2020	0x0011
…	...
0x2004	0x0003
0x2002	0x002A
0x2000	0x0013
0x1022	0x000C
0x1020	0x0007
…	...
0x1004	0x0004
0x1002	0x000B
0x1000	0x0006

Address Translation (4)

- Address translation examples
  - 0x002070 read: 0x0:02:070
    - Seg 0, page 2; PTE @ 0x002004; phy addr = 0x003070
  - 0x202016 read: 0x2:02:016
    - Seg 2, page 2; PTE @ 0x001004; phy addr = 0x004016
  - 0x104C84 read: 0x1:04:C84
    - Seg 1, page 4; PTE @ (Protection Error)
  - 0x011424 read: 0x0:11:424
    - Seg 0, page 17; PTE @ 0x002022; phy addr = 0x01F424
  - 0x210014 write: 0x2:10:014
    - Seg 2, page 16; PTE @ (Bounds Violation)

Pros and Cons (1)

- Advantages of P+S
  - If a segment isn't used, then there's no need to even have a page table for it
  - Can share at two levels
    - Single page
    - Single segment (whole page table)
  - Pages eliminate external fragmentation and make it possible for segments to grow without any reshuffling
  - If page size is small compared to most segments, then internal fragmentation is not too bad
  - User is not given access to the paging tables

Pros and Cons (2)

- Problems
  - Too big table space if pages are too small
    - Page tables are contiguous in physical memory
  - If translation tables are kept in main memory, overhead could be very high
    - 1 or 2 overhead references for every reverence

Solutions to P+S Problems (1)

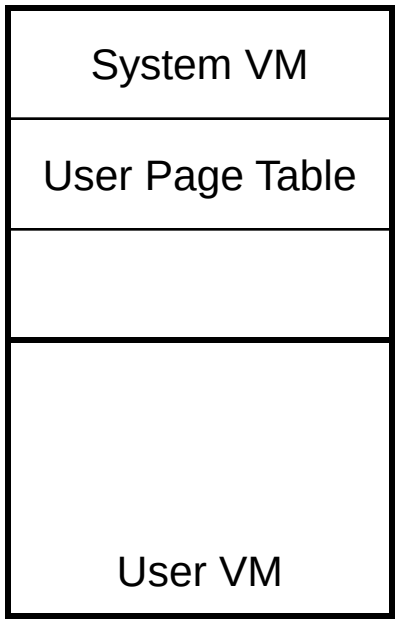
- Large page table size: The VAX case
  - Address is 32 bits, top two select segment
    - Four base-bound pairs define page tables
      - System, P0, P1, unused
      - One segment contains operating system stuff, two contain stuff of current user process
  - Read-write protection information is contained in the page table entries, not in the segment table
  - Pages are 512 bytes long
    - Too small

Solutions to P+S Problems (2)

- Large page table size: The VAX case (cont'd)
  - Use the system page table to map the user page tables so the user page tables can be scattered
    - System base-bound pairs are physical addresses, system tables must be contiguous
    - User base-bound pairs are virtual addresses in the system space
      - This allows the user page tables to be scattered in non-contiguous pages of physical memory

Solutions to P+S Problems (3)

- Large page table size: The VAX case (cont'd)
  - The result is a two level scheme



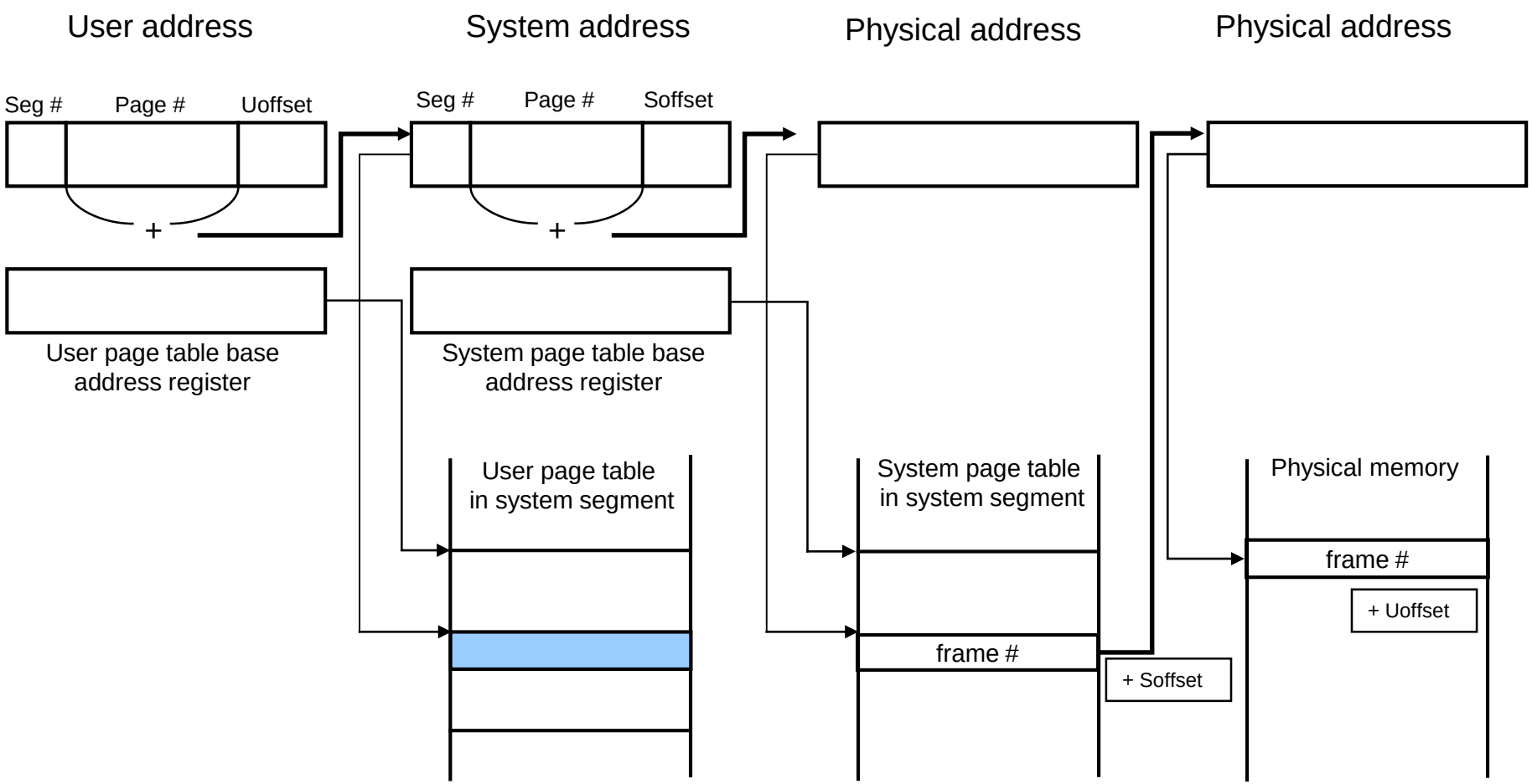
① User generates address

② Lookup in User Page Table

③ Lookup in System Page Table

④ Access physical address

Solutions to P+S Problems (4)



Solutions to P+S Problems (5)

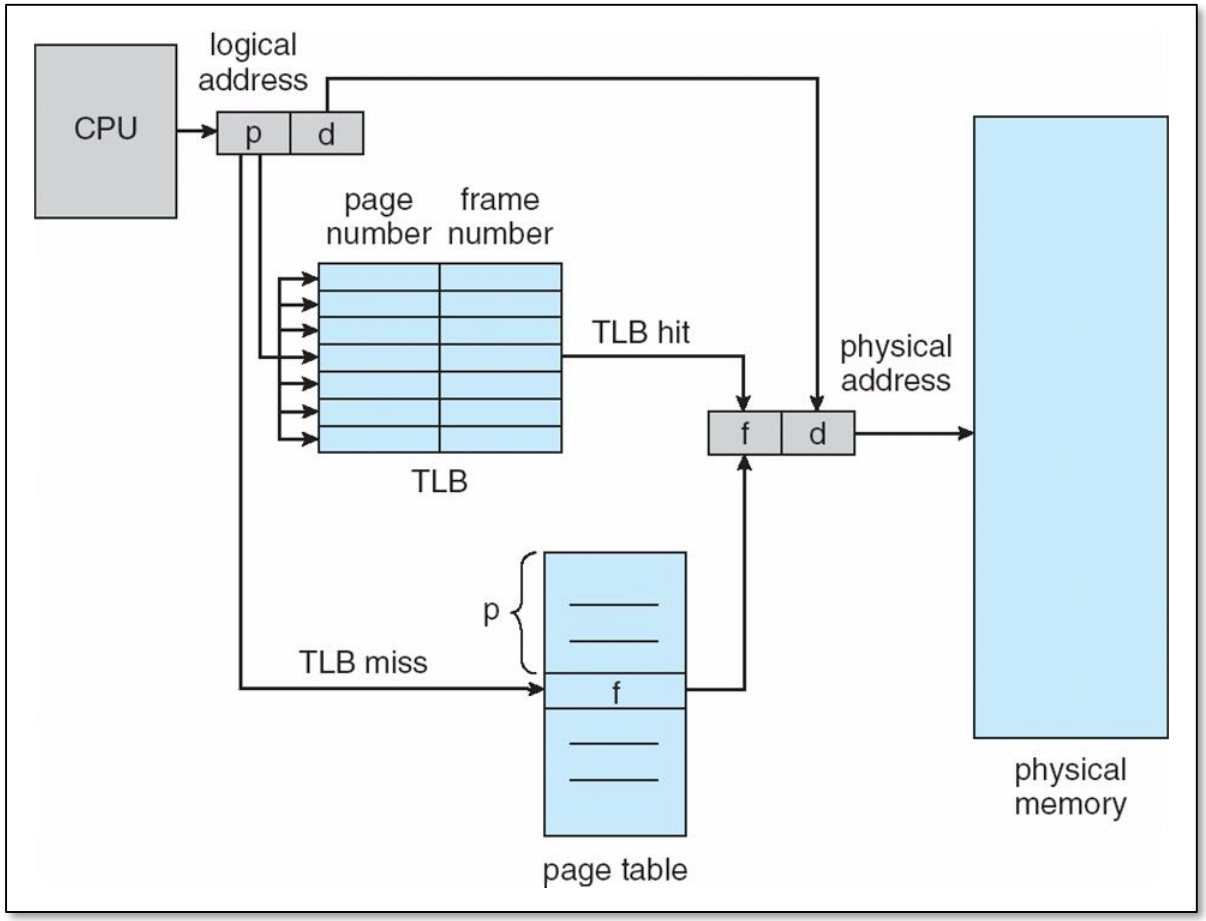
- Address translation overhead
  - Extra memory references to access translation tables can slow programs down by a factor of two or three
  - Too many entries in translation tables to keep them all loaded in fast processor memory
    - Remember notion of locality
      - At any given time a process is only using a few pages or segments
  - Solution: Translation lookaside buffer (TLB)

IV. Enhancing Mechanisms

TLB (1)

- *Translation Lookaside Buffer* (TLB)
  - Used to cache a few of translation table entries
  - It's very fast, but only stores a small number of entries
  - On each memory reference,
    - First ask TLB if it knows about the page
    - If so, the reference proceeds fast
    - If TLB has no info for the page,
      MMU must go through page tables to get the info
      - Reference takes a long time, but TLB is given the info for this page so it will know it for the next reference
    - TLB must forget one of its current entries
      in order to record a new one
  - Also called a *Translation Buffer* (TB)

TLB (2)



Source: Silberschatz, Galvin and Gagne, Operating System Concepts, 2008

TLB (3)

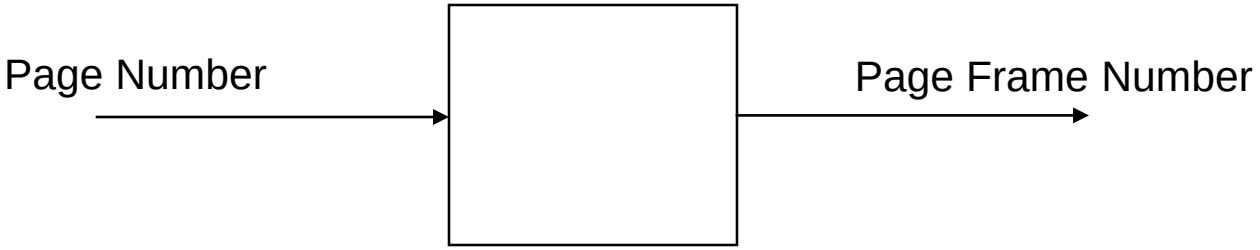
- *Effective memory access time*: τ

$$\tau = (\varepsilon + \mu)\,\alpha + (\varepsilon + 2\,\mu)(1 - \alpha)$$
$$= \varepsilon + (2 - \alpha)\,\mu$$

  - "ε": *associative lookup time* (10 *ns*)
  - "μ": *memory cycle time* (100 *ns*)
  - "α": *TLB hit ratio*
    - Percentage of times that a page number is found in TLB
    - Dependent on the number of TLB lines

TLB (4)

- TLB organization
  - TLB is *cache memory*
    - Cache is memory with some comparators
  - Typical sizes of memory: 64 to 2K entries
  - Each entry holds a virtual page number (VPN) and the corresponding physical page number (PPN)



TLB (5)

- TLB organization (cont'd)
  - How can cache lines be organized to find an entry quickly?

1. One possibility (*naïve*)
   - Search the whole table from the start on every reference
   - Sequential table search
   - Not acceptable due to low performance

TLB (6)

- TLB organization (cont'd)

2. Second possibility (*direct-mapped cache*)
   - Restrict the page table entry for any given virtual page to fall in exactly one location in the cache memory
     - Use the "*low-order*" bits of the virtual page numbers as the index into the memory (like an array)
     - Then only need to check that one location with a *tag*

3. Third possibility (*fully associative cache*)
   - Check all entries in parallel (expensive but fast)

TLB (7)

- TLB organization (cont'd)

4. Another approach (*m-way set-associative cache*)
   - Restrict the page table entry for any given virtual page to fall in exactly one of the 2^n sets in the cache memory
     - Use the "*low-order*" bits of the virtual page numbers as the index into the sets
   - Each set has 2^m lines (*m*-way)
     - The page table entry can be in any of them
     - Use *m* comparators and the rest of the bits of the page number as a *tag* to find the matching entry
   - About as fast as the simple scheme, but a bit more expensive
     - The *m* comparators instead of one
     - Have to decide which entry to replace to bring in a new entry
   - This is between the second and third possibility

TLB (8)

- TLB example
  - MIPS R2000/R3000
    - CPU used in DecStations and SGI machines
    - Addresses are 32 bits: 12-bit page offset (i.e., 4K pages)
    - TLB entry format: 64 bits
    - 64 TLB entries

TLB (9)

- TLB example (cont'd)
  - MIPS R2000/R3000

20	6	6	20	1	1	1	1	8
VPN	PID	0	PFN	N	D	V	G	0

G — Global, valid for any PID
V — Entry is valid
D — Dirty bit, page has been modified
N — Don't cache the page
PFN — Physical address of the page
PID — Process ID for entry (or called ASID)
VPN — Virtual page number

TLB (10)

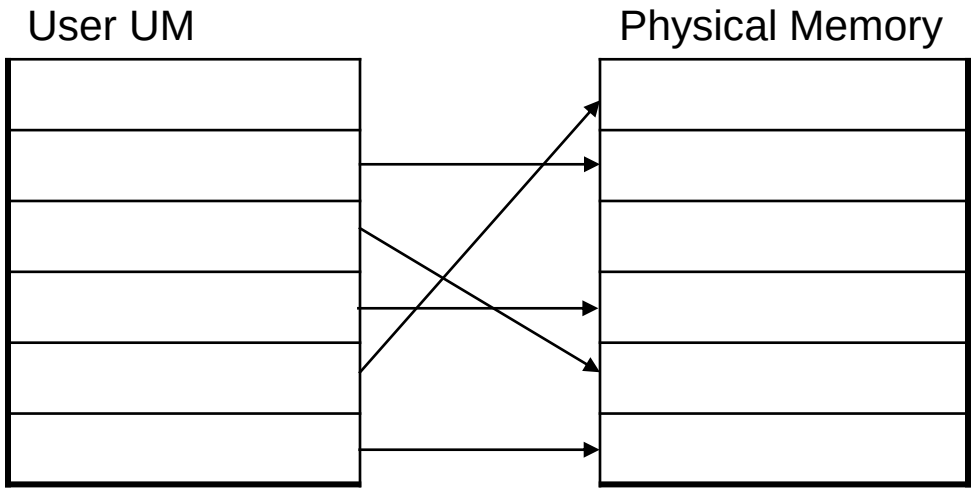
- In practice, TLB's have been extremely successful
  - 98% ratio is typical for 128 entries
- Interactions with OS
  - TLB can be mostly hidden from OS
    - Possible exception: Context switches
      - Must either:
        - Flush TLB during each context switch
        - Or, store a Process ID (PID) in the TLB entry

TLB (11)

- Obtaining TLB entries
  - Software-managed TLB
    - OS loads the entries via a TLB miss fault
      - Issued when a process tries to access an address that is not in the TLB
  - Hardware-managed TLB
    - Hardware searches page table itself to get the missing entry
      - OS never sees TLB miss faults: they are handled in hardware
    - Pro
      - Faster
    - Con:
      - OS must set up the page table in a fixed way that the hardware understand

User Memory Access in OS (1)

- How does OS get information from user memory?
  - Example: I/O buffers, parameter blocks
  - Note that the user passes the OS virtual addresses
  - Note addresses that are contiguous in the virtual address space may not be contiguous physically
    - I/O operations may have to be split up into multiple blocks



User Memory Access in OS (2)

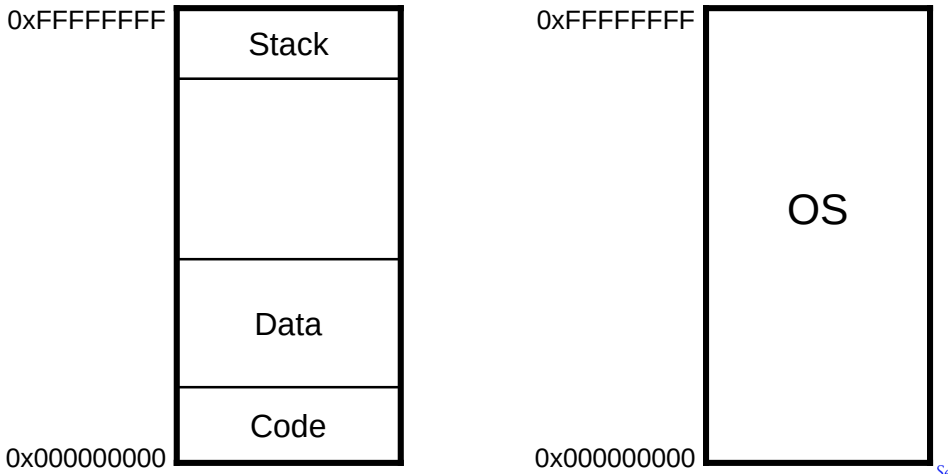
- Varies depending on OS/user memory configurations

- Three different cases
  ① Run the OS unmapped
  - OS reads the page tables and translate user addresses in software
  - I/O operations may have to be split up into multiple blocks

User Memory Access in OS (3)

② Run the OS mapped to a separate address space

- OS and user run mapped in different address spaces
- It must generate a page table entry for the user area
- Some machines provide special instructions to get the user stuff
  - Note that under no circumstances should users be given access to mapping tables (sun4u SPARC V9)



User Memory Access in OS (4)

③ Run the OS mapped to users' address spaces

- Both OS and user run mapped in the same address space
- Translate both OS and I/O addresses thru the TLB
  - Ex: SunOS on a SPARC Station (SPARC V7)
- Both system and user information visible at once
  - Can't touch system stuff unless running with protection bit set
- IO devices DMA into virtual addresses