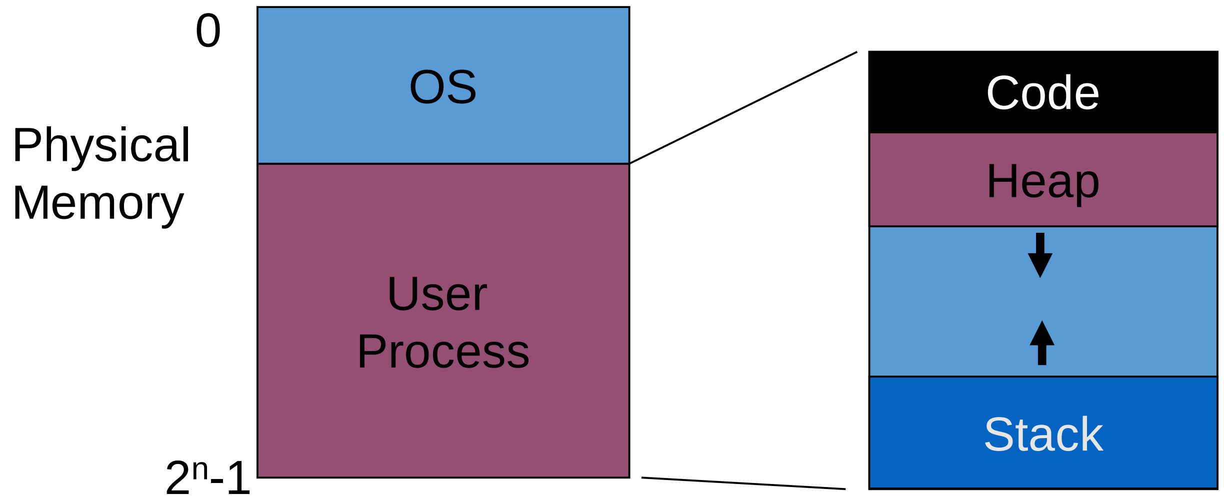
Introduction to
Virtual Memory

Nima Honarmand

(Based on slides by Prof. Andrea Arpaci-Dusseau)

Motivating Virtual Memory

- (Very) old days: Uniprogramming — only one process existed at a time
  - "OS" was little more than a library occupying the beginning of the memory



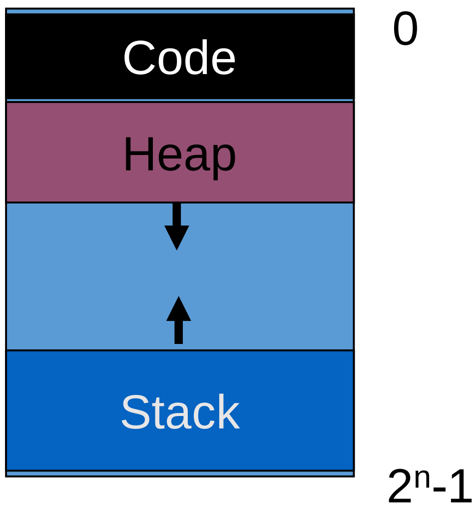
- Advantage:
  - Simplicity — No virtualization needed
- Disadvantages:
  - Only one process runs at a time
  - Process can destroy OS

Goals for Multiprogramming

- Transparency
  - Processes are not aware that memory is shared
  - Works regardless of number and/or location of processes
- Protection
  - Cannot corrupt OS or other processes
  - Privacy: Cannot read data of other processes
- Efficiency
  - Low run-time overhead
  - Do not waste memory resources
- Sharing
  - Cooperating processes should be able to share portions of address space

Abstraction: Address Space

- Address space: Each process' view of its own memory range
  - Set of addresses that map to bytes
- Problem: how can OS provide illusion of private address space to each process?
- Address space has static and dynamic components
  - Static: Code and some global variables
  - Dynamic: Stack and Heap



How to Virtualize Memory?

- Problem: How to run multiple processes concurrently?
- Addresses are “hard-coded” into program binaries
- How to avoid collisions?

How to Virtualize Memory?

- Possible Solutions for Mechanisms:
  1) Time Sharing
  2) Base register
  3) Base + Bound registers
  4) Segmentation
  5) Paging
- We'll first discuss the general ideas
  - To motivate the historical progression
  - Building insight/intuition into why things are the why they are
- Once familiar with the basic concepts, we'll take a look at x86 idiosyncrasies
  - Of which, there are a lot ☺

1) Time Sharing of Memory

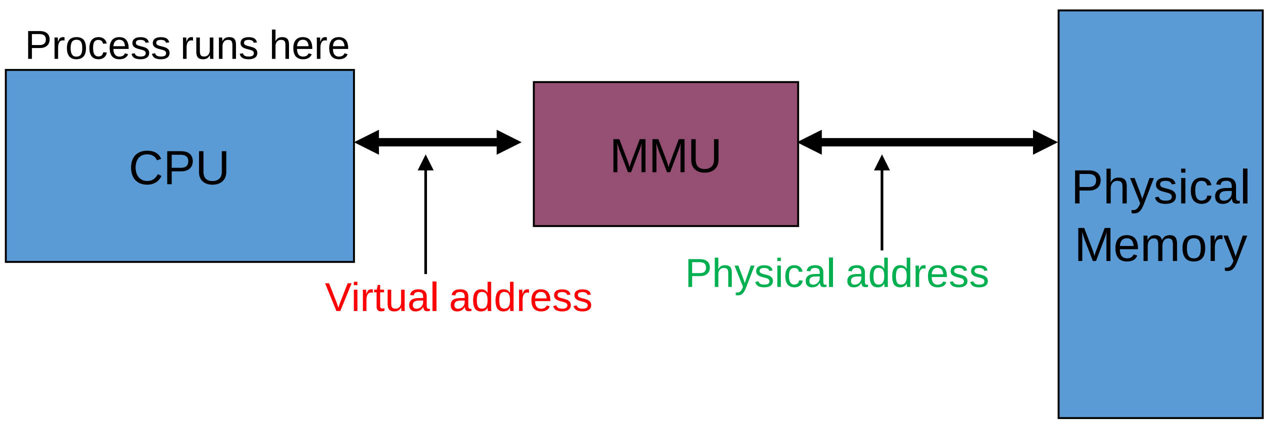
- Try similar approach to how OS virtualizes CPU
- Observation: OS gives illusion of many virtual CPUs by saving CPU registers to memory when a process isn't running
- Could give illusion of many virtual memories by saving memory to disk when process isn't running

Problem w/ Time Sharing Memory

- Problem: Ridiculously poor performance
- Better Alternative: space sharing
  - At same time, space of memory is divided across processes
- Remainder of solutions all use space sharing

2) Per-Process Base Register

- Goal: Allow processes to space-share the physical memory
- Requires hardware support
  - *Memory Management Unit* (*MMU*)
- MMU dynamically changes process-generated address at every memory reference
  - Process generates virtual addresses (in their address space)
  - Memory hardware uses physical addresses



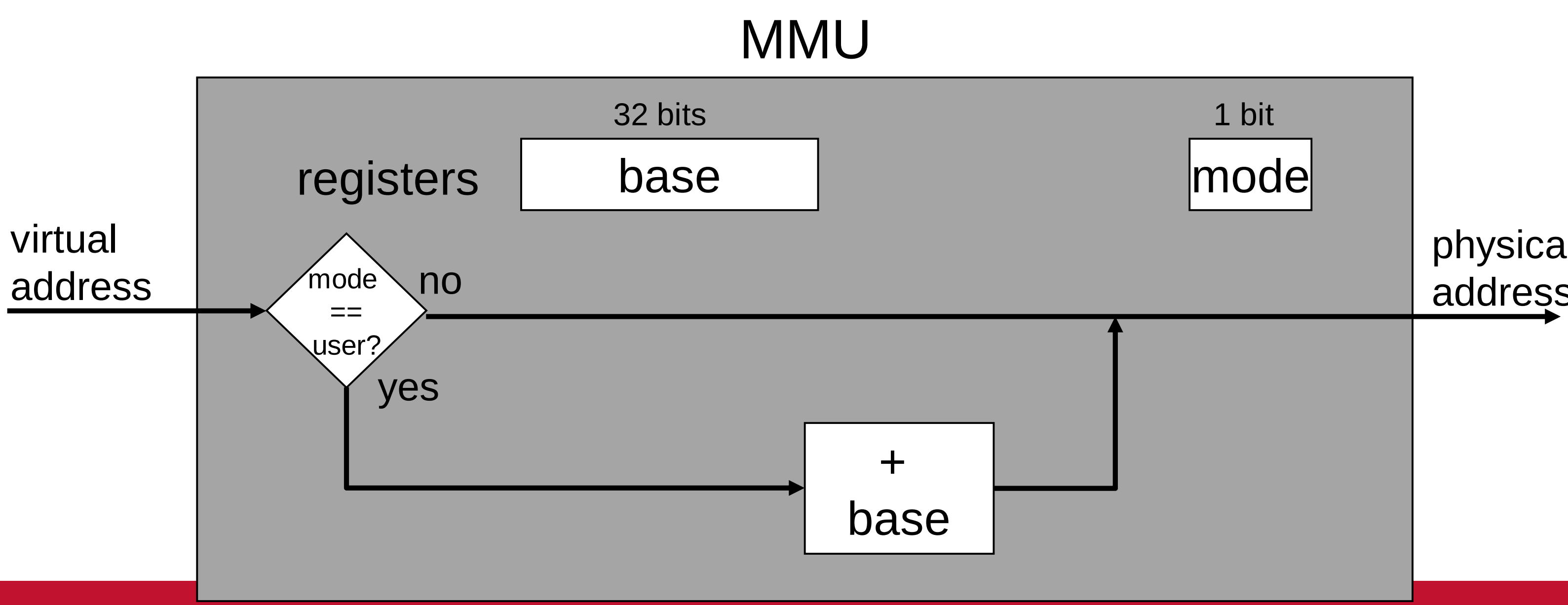
Hardware Support

Two operating modes:

- Privileged (protected, kernel) mode: OS runs
  - When enter OS (trap, system calls, interrupts, exceptions)
  - Allows certain instructions to be executed
    - Can manipulate contents of MMU
  - Allows OS to access all of physical memory
- User mode: User processes run
  - Perform translation of virtual address to physical address
- Minimal MMU contains base register for translation
  - Base: start location for current address space

HW Implementation

- Translation on every memory access of user process
  - MMU adds base register to virtual address to form physical address
  - Each process has different value in base register
    - Saved/restored on a context switch



Quiz: Who Controls Base Register?

- What entity should do the address translation with base register?
  1) User Process
  2) OS
  3) HW
- What entity should modify the base register?
  1) User Process
  2) OS
  3) HW

Advantages & Disadvantages

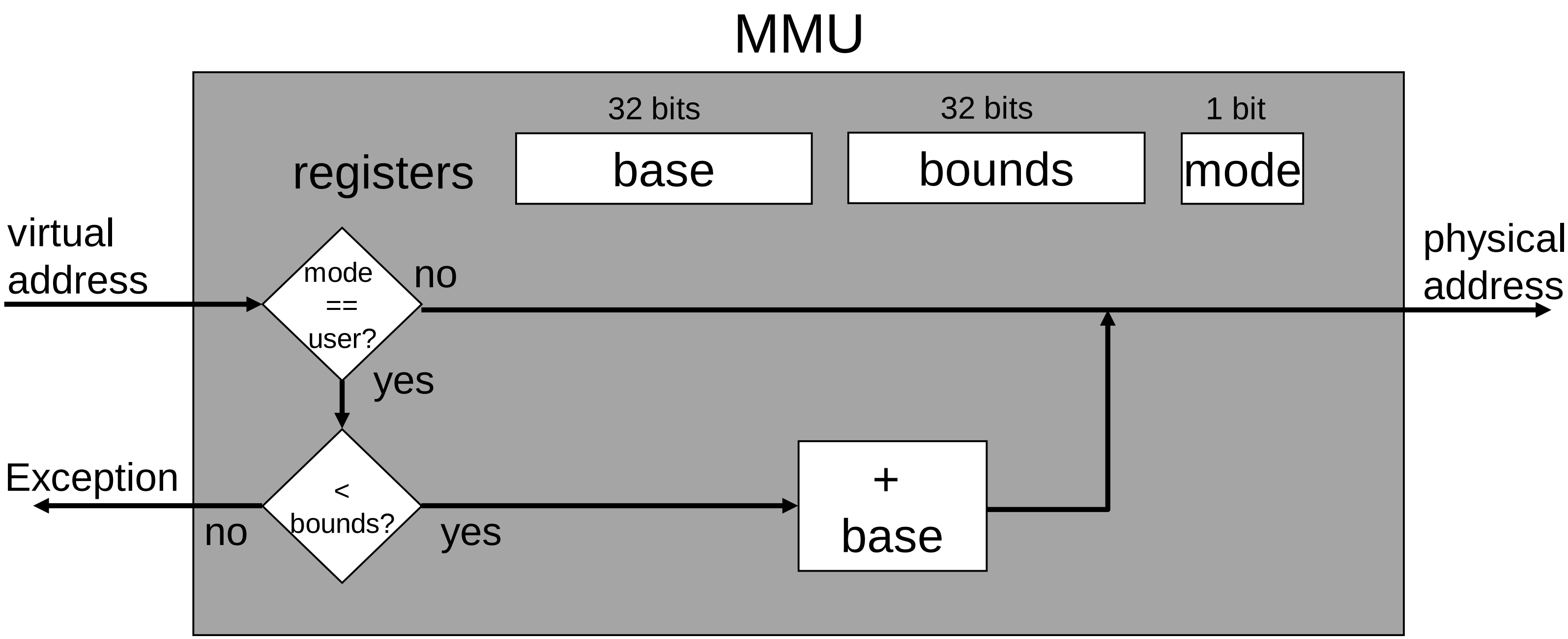
- Advantages:
  - Supports ***Dynamic Relocation***
    - Can place process at different locations in memory everytime
    - Can even move process address space around in memory
  - Simple, fast HW
  - Simple OS support
- Disadvantages:
  - Does not provide protection
  - Does not enable sharing
  - OS has to allocate the maximum address space up front
    - Wastes a lot of memory space

3) Per-Process Base+Bounds Regs

- Idea: limit the address space with a bounds register
  - To provide protection
- **Base register**: smallest physical addr (or starting location)
- **Bounds register**: size of this process's virtual address space
  - Sometimes defined as largest physical address (base + size)
- OS kills process if process loads/stores beyond bounds

HW Implementation

- Translation on every memory access of user process
  - MMU compares virtual address to bounds register
    - if virtual address is greater, then generate exception
  - MMU adds base to virtual addr to form physical address



Managing Processes w/ Base+Bounds

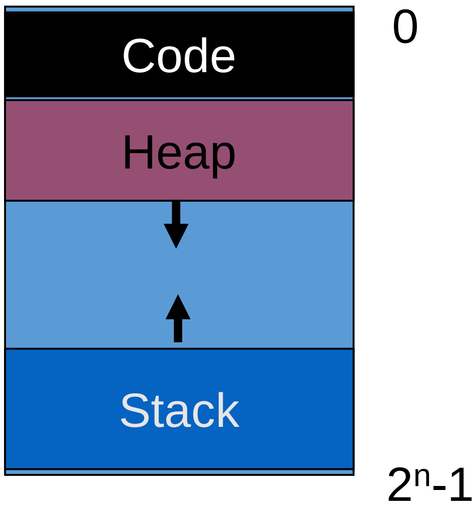
- Context-switch
  - Add base and bounds registers to PCB
  - Steps
    - Change to privileged mode
    - Save base and bounds registers of old process
    - Load base and bounds registers of new process
    - Change to user mode and jump to new process
- Protection requirement
  - User process cannot change base and bounds registers
  - User process cannot change to privileged mode

Base+Bounds Advantages

- Provides protection across address spaces
- Supports dynamic relocation
- Simple, inexpensive, fast HW implementation
  - Few registers, little logic in MMU
- Simple context switching logic
  - Save and restore a couple of registers

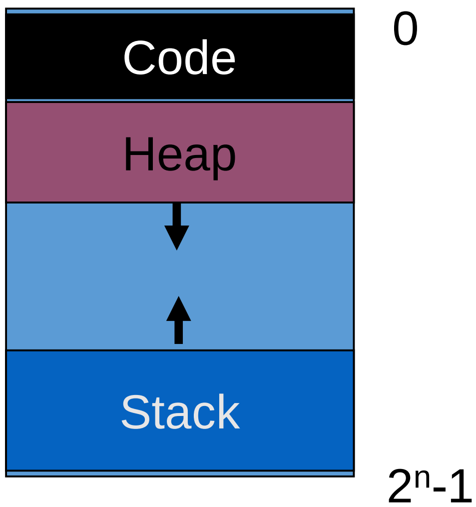
Base+Bounds Disadvantages

- Each process must be allocated contiguously in physical memory
  - Must allocate memory that may not be used by process
- No partial sharing: Cannot share limited parts of address space



4) Segmentation

- Divide address space into logical segments
  - Each segment corresponds to logical entity in address space
    - E.g., code, stack, heap
- Each segment can independently:
  - be placed in physical memory
  - grow and shrink
  - be protected (separate read/write/execute protection bits)



Segmented Addressing

- Code now specifies **segment** and **offset within segment** for every memory access

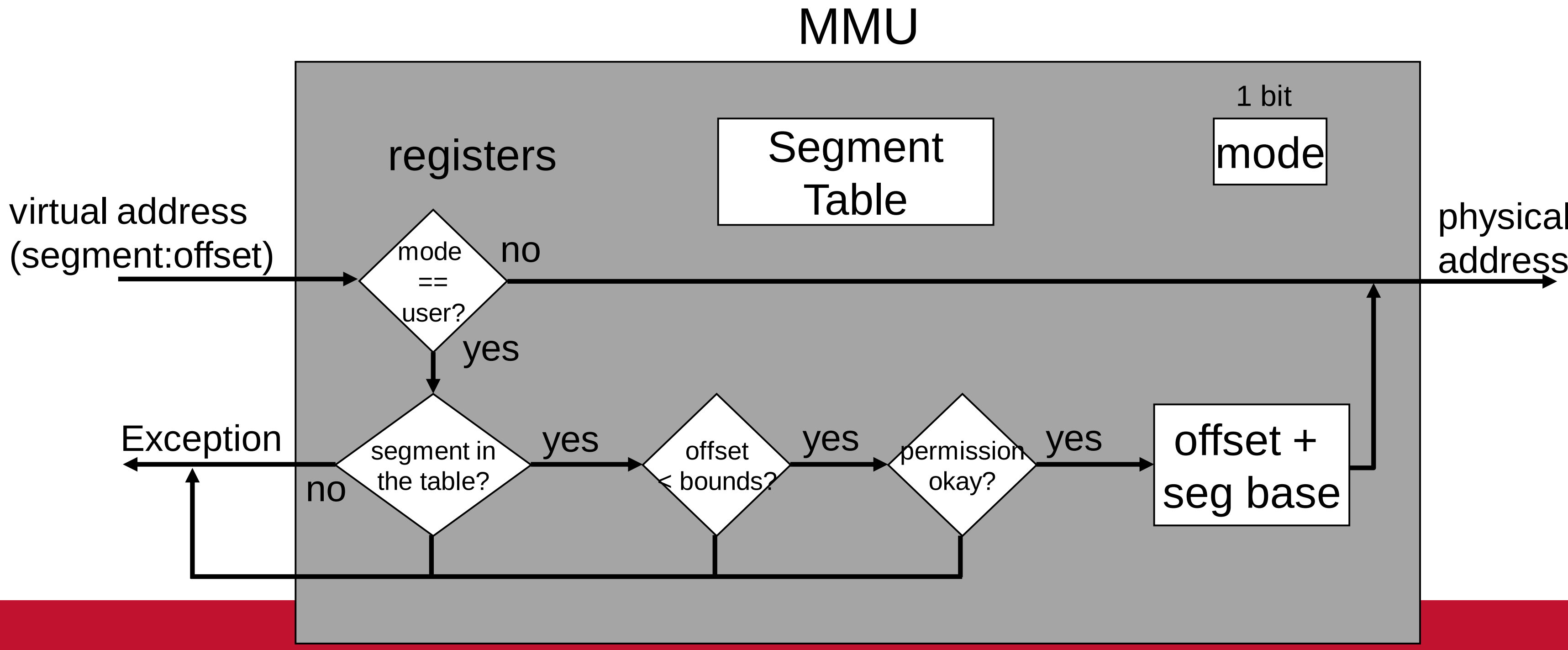
- How to designate a particular segment?
  - Use part of virtual address
    - Top bits of virtual address select segment
    - Low bits of virtual address select offset within segment

- What if small address space, not enough bits?
  - Implicitly by type of memory reference
    - To fetch an instruction: user code segment
    - To push/pop stack: use stack segment
    - For everything else: use data segment
  - Use special *segment registers* if you need to override the above

Mechanism used in x86

HW Implementation

- MMU contains **Segment Table**
  - Each segment has own base, bounds, protection bits
  - MMU sores the segment table (or has a special register pointing to the segment table stored in memory)



Example

- 14 bit virtual address, 4 segments
  - How many bits for segment? How many bits for offset?

Segment	Base	Bounds	R W
0	0x2000	0x6ff	1 0
1	0x0000	0x4ff	1 1
2	0x3000	0xfff	1 1
3	0x0000	0x000	0 0

- Translate virtual addresses (in hex) to physical addresses
  - 0x0240: Seg 0; phys-addr: 0x2000 + 0x240 = 0x2240
  - 0x1108: Seg 1; phys-addr: 0x0000 + 0x108 = 0x108
  - 0x265c: Seg 2; phys-addr: 0x3000 + 0x65c = 0x365c
  - 0x3002: Seg 3: out of bounds — no translation

Advantages of Segmentation

- Enables **sparse** allocation of address space
  - Stack and heap can grow independently
  - Heap: If no objects on free list, dynamic memory allocator requests more from OS (e.g., UNIX: malloc calls sbrk())
  - Stack: OS recognizes reference outside legal segment, extends stack implicitly
- Different protection for different segments
  - Read-only status for code
- Enables sharing of selected segments
- Supports dynamic relocation of each segment

Disadvantages of Segmentation

- If only a few segments allowed per process: **coarse-grained** segmentation
  - Not very flexible
  - Cannot easily accommodate sharing
- If many segments allowed per process: **fine-grained** segmentation
  - Has to save/restore a large segment table on every context switch
    - Remember: each process has its own set of segments
- Makes physical memory management much more complex
  - Causes ***fragmentation***

Conclusion

- HW & OS work together to virtualize memory
  - Give illusion of private address space to each process
- Adding MMU registers for base+bounds so translation is fast
  - OS not involved with every address translation, only on context switch or errors
- Dynamic relocation w/ segments is good building block
- Next lecture: Solve fragmentation with paging