CSIS 429 Operating Systems

Lecture 5: Address Translation

September 21st 2020

Textbook chapters

Read “Address Translation” and “Segmentation”

Intro	**Virtualization**		**Concurrency**	**Persistence**	**Appendices**
Preface	3 *Dialogue*	12 *Dialogue*	25 *Dialogue*	35 *Dialogue*	*Dialogue*
TOC	4 Processes	13 Address Spaces	26 Concurrency and Threads code	36 I/O Devices	Virtual Machines
1 *Dialogue*	5 Process API code	14 Memory API	27 Thread API	37 Hard Disk Drives	*Dialogue*
2 Introduction code	6 Direct Execution	15 Address Translation	28 Locks	38 Redundant Disk Arrays (RAID)	Monitors
	7 CPU Scheduling	16 Segmentation	29 Locked Data Structures	39 Files and Directories	*Dialogue*
	8 Multi-level Feedback	17 Free Space Management	30 Condition Variables	40 File System Implementation	Lab Tutorial
	9 Lottery Scheduling code	18 Introduction to Paging	31 Semaphores	41 Fast File System (FFS)	Systems Labs
	10 Multi-CPU Scheduling	19 Translation Lookaside Buffers	32 Concurrency Bugs	42 FSCK and Journaling	xv6 Labs
	11 *Summary*	20 Advanced Page Tables	33 Event-based Concurrency	43 Log-structured File System (LFS)	Flash-based SSDs
		21 Swapping: Mechanisms	34 *Summary*	44 Data Integrity and Protection	
		22 Swapping: Policies		45 *Summary*	
		23 Case Study: VAX/VMS		46 *Dialogue*	
		24 *Summary*		47 Distributed Systems	
				48 Network File System (NFS)	
				49 Andrew File System (AFS)	
				50 *Summary*	

Review: Operating Systems virtualize CPUs

- Operating system scheduler
  - Makes it look like each process has the CPU to itself

- Hard parts
  - How an OS does this:
    - context switch to save register state
    - monitoring each process and adjusting priorities
  - Being fair to all running processes - MLFQ

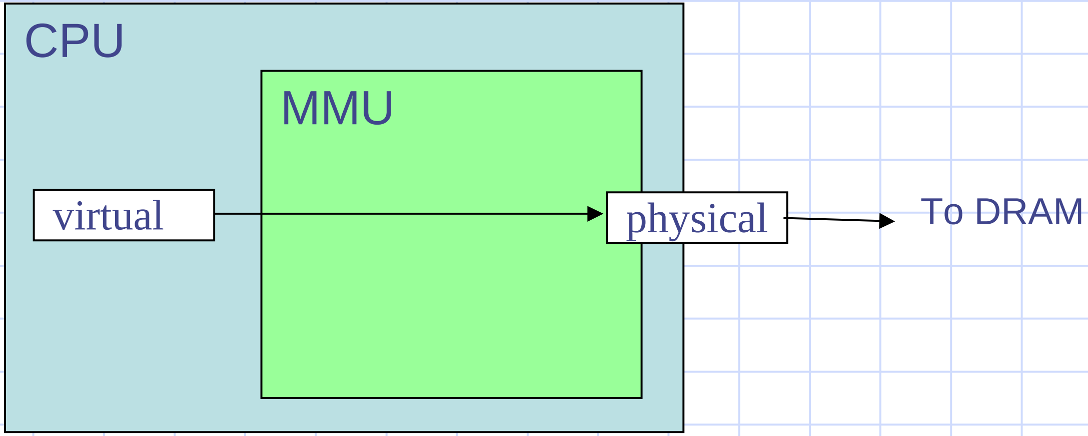
Review: Virtual Addresses

- Every address generated by a process is treated as a “virtual address” by the OS
- OS translates each virtual address to a DRAM address.

- The “crux” of the problem:
- How can the OS build the abstraction of a private address space while actually having many processes use physical memory?
- And doing so securely? And efficiently?

Hardware-based Address Translation

- Every address generated internally by the CPU is treated as a “virtual address” by the Memory Management Unit (MMU) – hardware managed by the OS.



MMU translates each virtual address to a DRAM address.

Virtual vs. Physical Address Space

- The concept of a virtual *address space* that is bound to a separate *physical address space* is central to memory management.
  - *Virtual address* – generated by the CPU internally; also referred to as *logical address*.
  - *Physical address* – address ouput by the memory unit.
- Goal: create the illusion that the program has its own private memory. Behind the virtual reality is the ugly physical truth: there are many processes sharing memory!

Just like …
the Matrix!

Simple Virtual Memory System

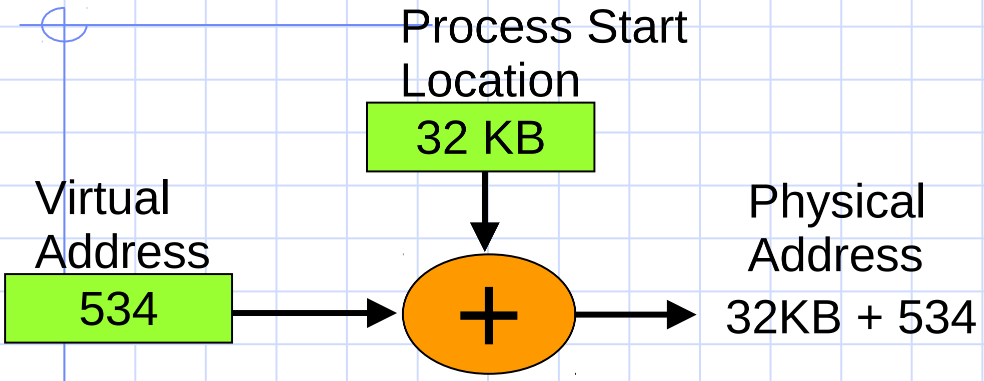
Assume:

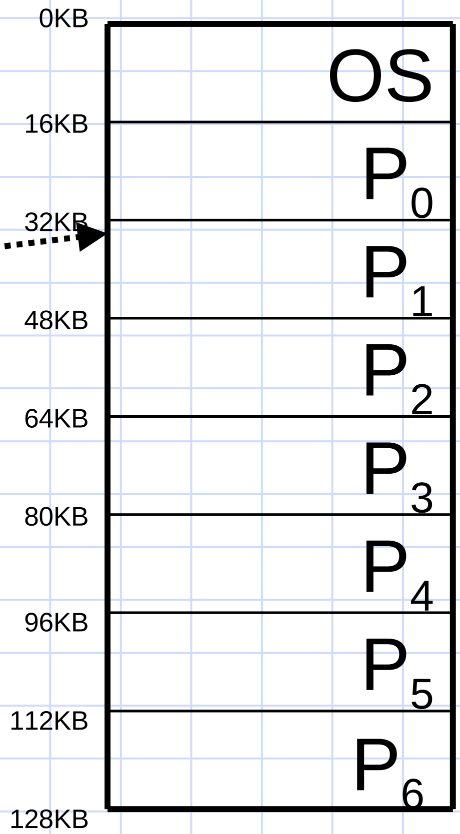
1) User’s address space is contiguous in physical memory
2) Programs can only use a small amount of memory, less than total physical memory
3) Each program is exactly the same size.

→ Physical memory is divided into 16 KB pieces

When a new process is loaded, assign it a new starting location

Simple scheme for relocation





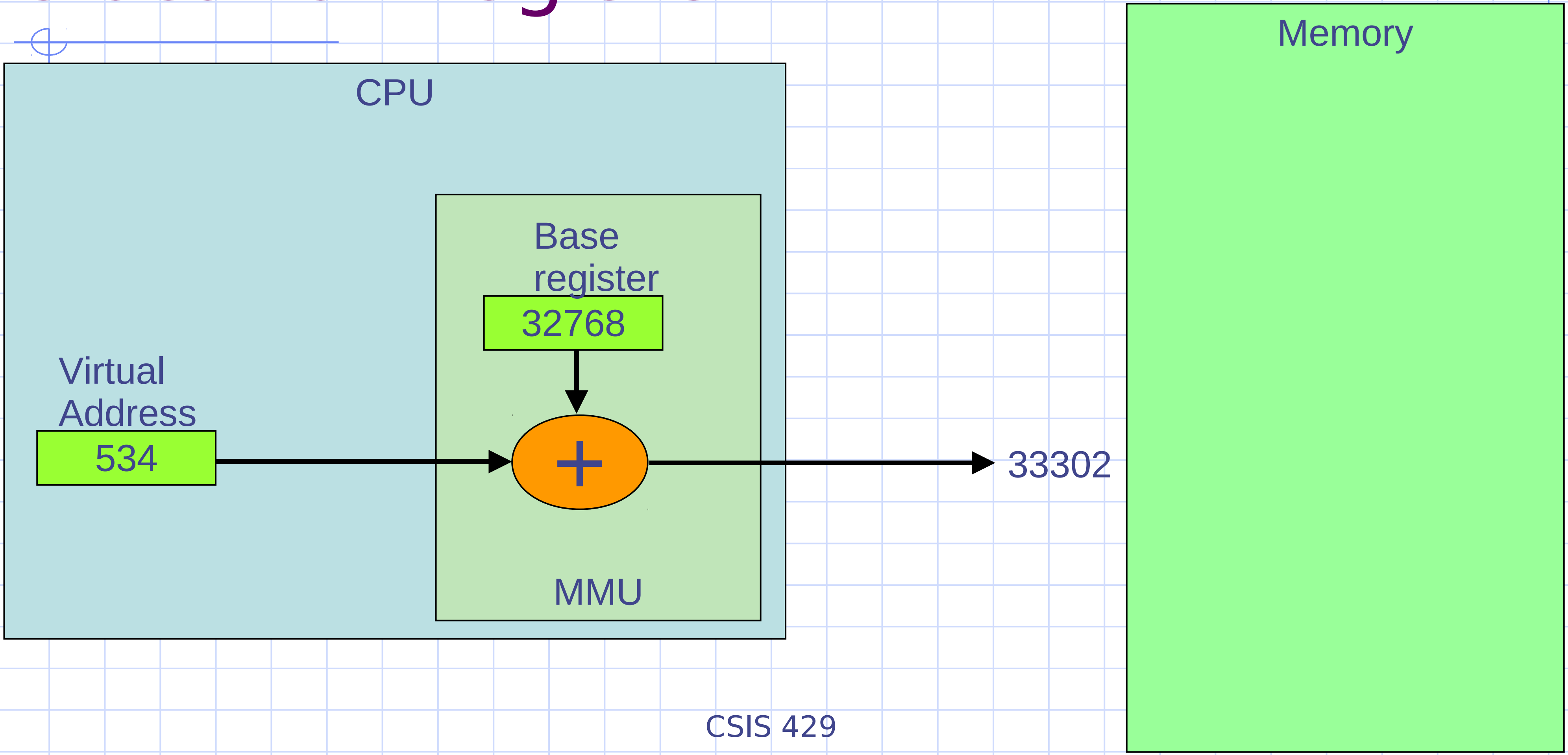
Limitations:
Could be hard to share memory
No big programs
Small programs waste memory

Most OSes use this idea without the fixed size

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- In MMU, the Process Start Location value is in a "base register" and added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *virtual* addresses; it never sees the *real* physical addresses.

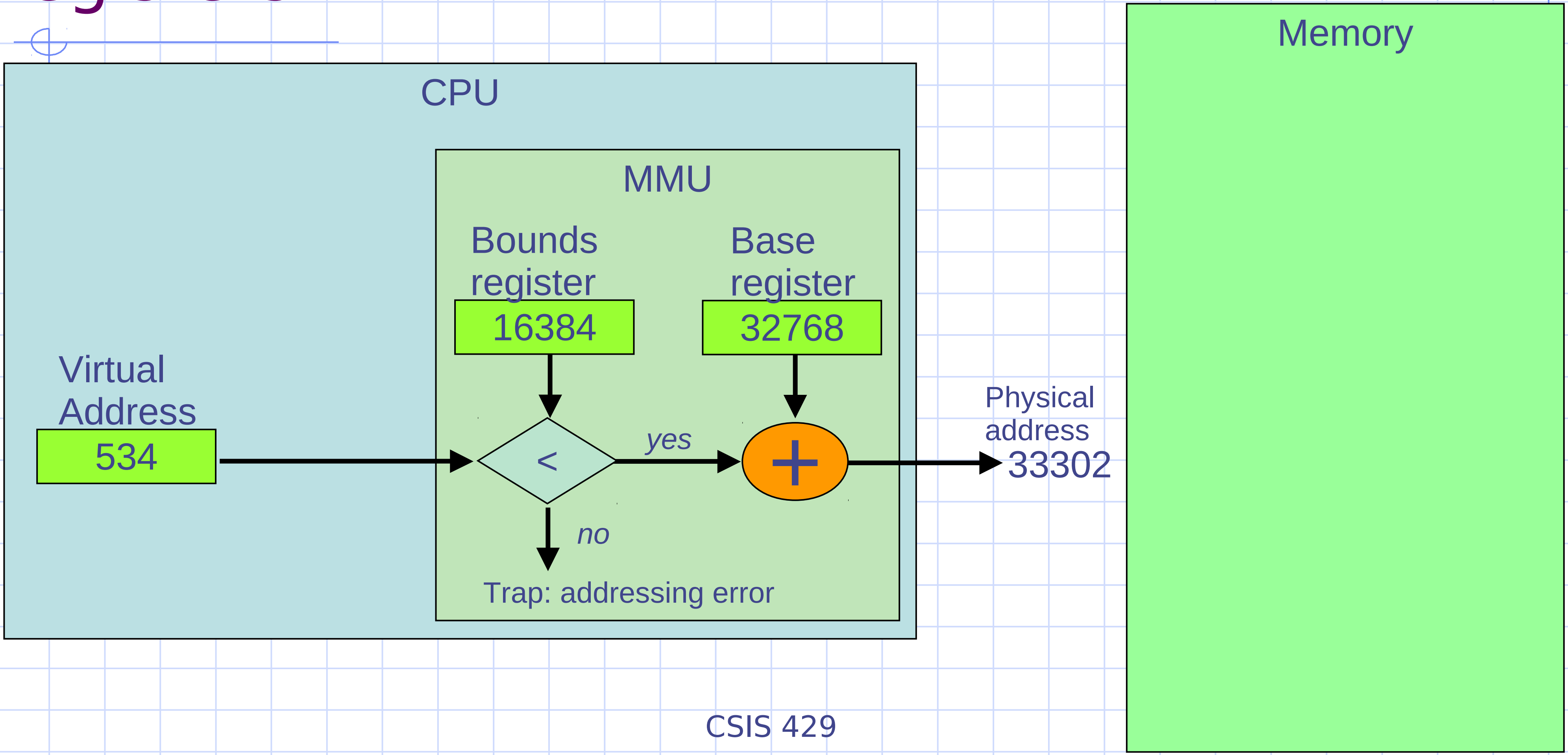
Dynamic relocation using a relocation register



Contiguous Allocation

- Main memory has two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes in high memory.
- Single-partition allocation
  - Base-register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Base register contains value of smallest physical address; bounds register contains range of virtual addresses – each virtual address must be less than the bounds register.

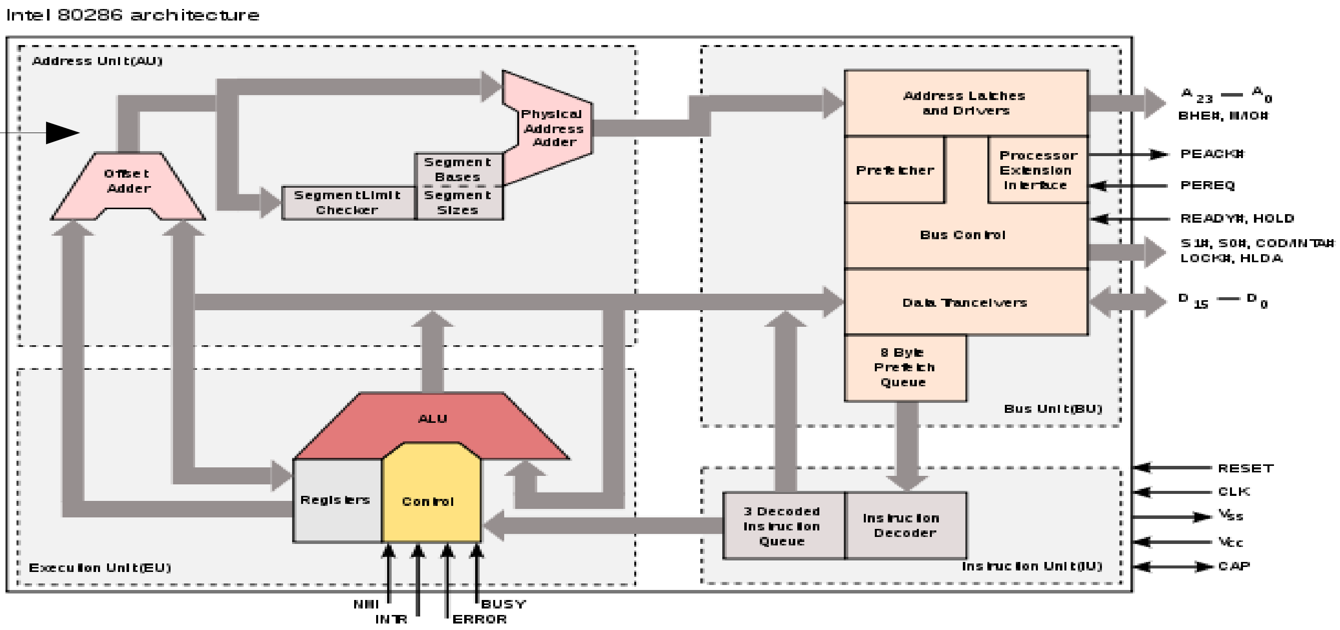
Hardware Support for Base and Bounds Registers



Intel 80286 CPU - MMU

- The Intel 80286 had memory management and protection for segments.

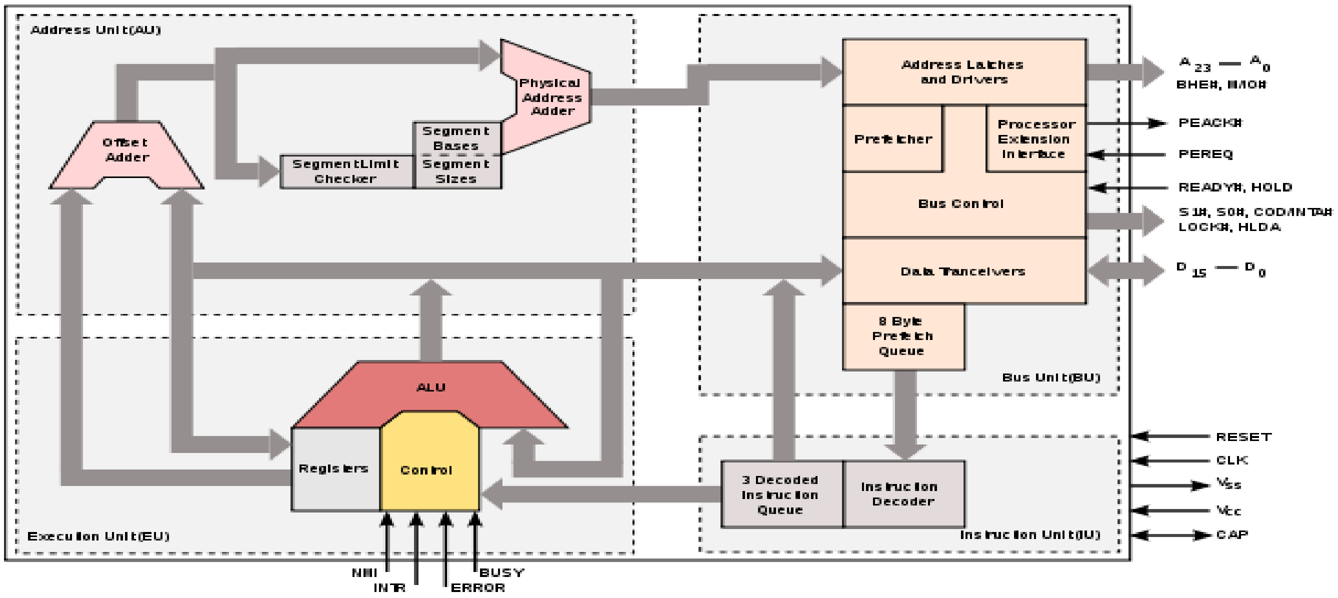
The Address Unit had basic MMU hardware – adders and comparators



Intel 80286 CPU

CPU with 16-bit data and 24-bit addressing.
Floating-point operations were done in a co-processor, the 80287.

Intel 80286 architecture



Example user program fragment

```
void func(int y) {
    int x = y;
    x = x + 3; // translate to assembly
```

In assembly language:

Address of **x** is stored in the **ebx** register

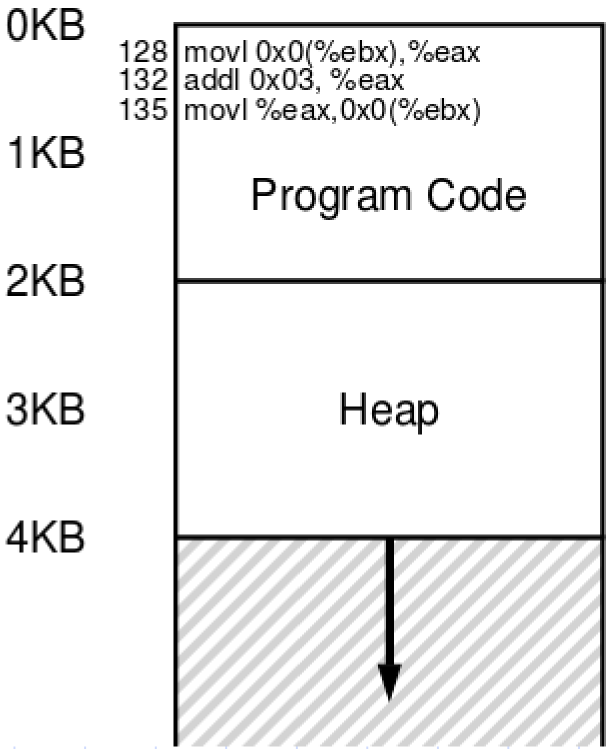
```
128: movl 0x0(%ebx), %eax   ; load (ebx+0) into eax
132: addl $0x03, %eax       ; add 3 to eax
135: movl %eax, 0x0(%ebx)   ; store eax to mem
```

Example code

```
void func(int y) {
    int x = y;
    x = x + 3;
```

In assembly language:

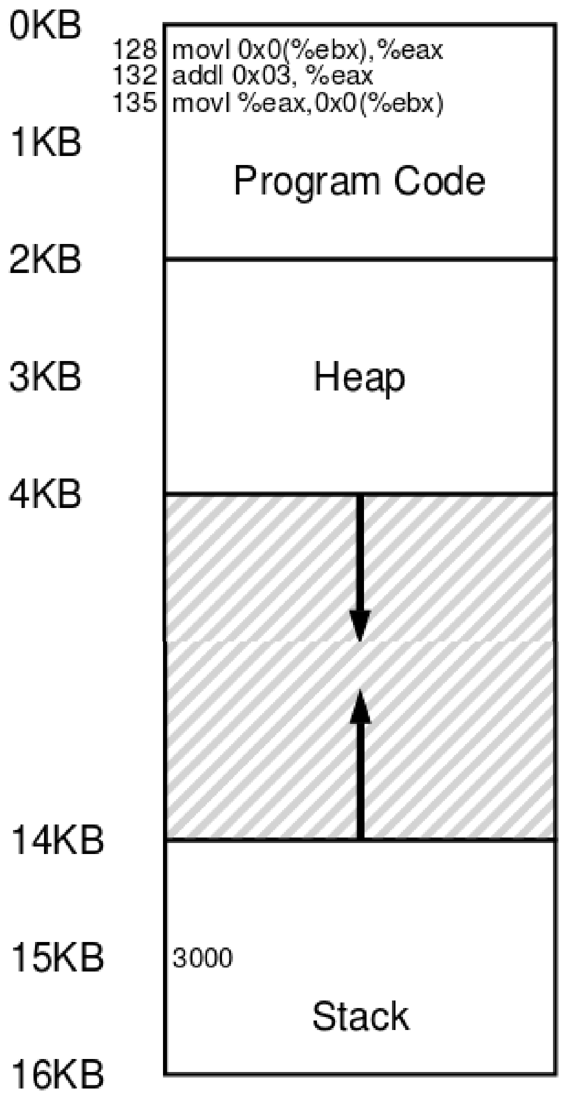
```
128: movl 0x0(%ebx), %eax
132: addl $0x03, %eax
135: movl %eax, 0x0(%ebx)
```



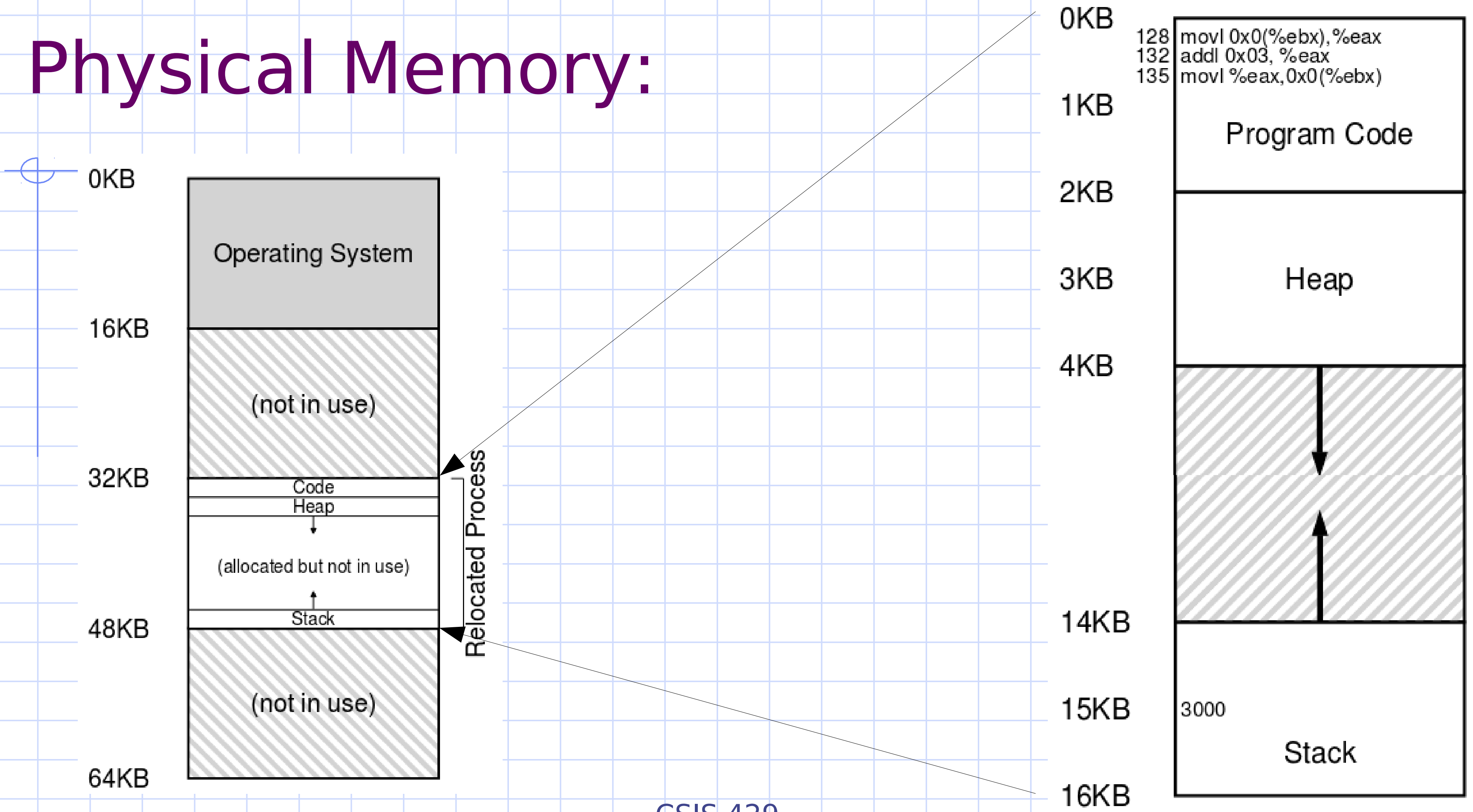
Example code execution:

- Fetch instruction at 128-131
- Execute (load from addr 15KB)
- Fetch instruction at 132-134
- Execute (use ALU to add)
- Fetch instruction at 135-138
- Execute (store to addr 15KB)

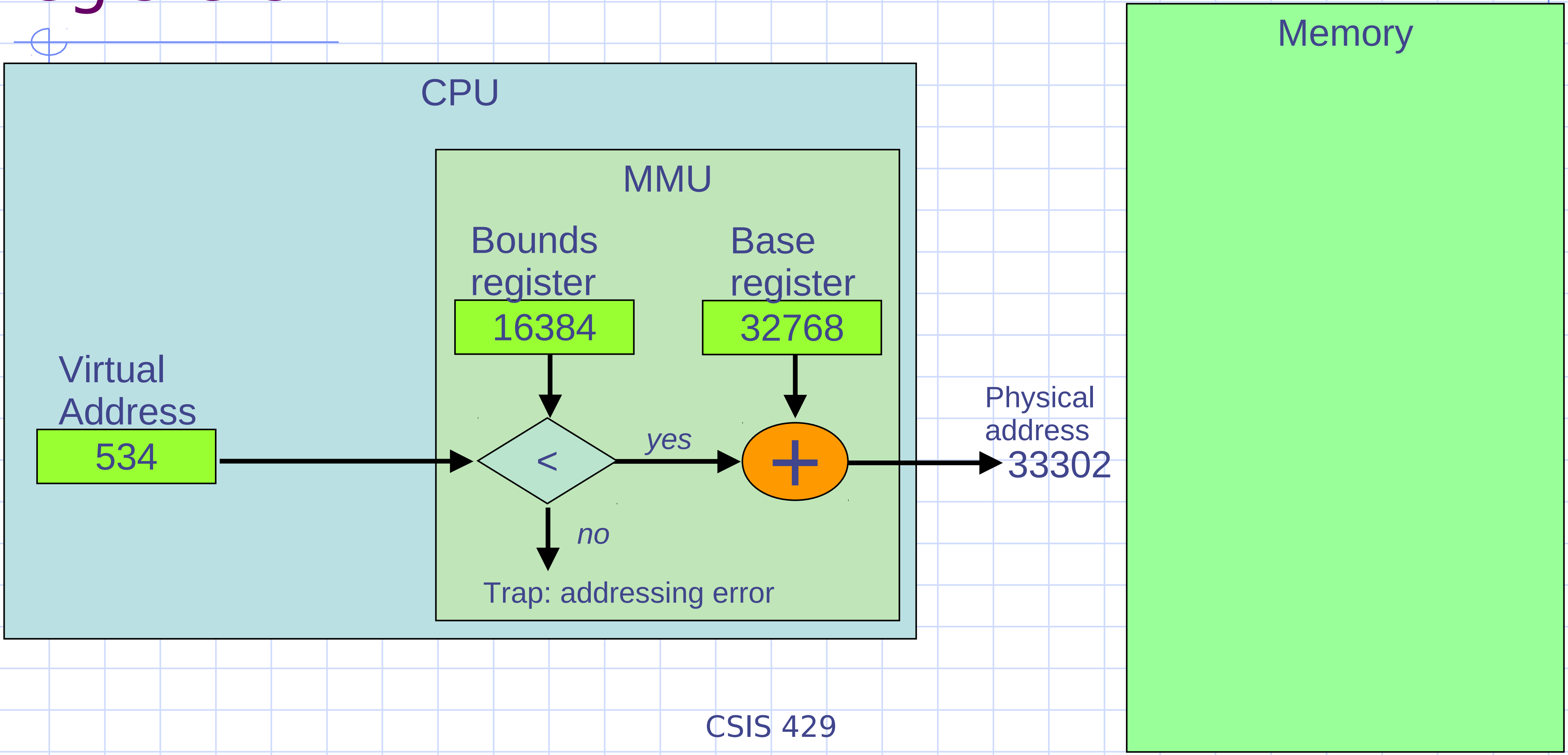
Program thinks its address starts at 0 → addresses generated inside CPU



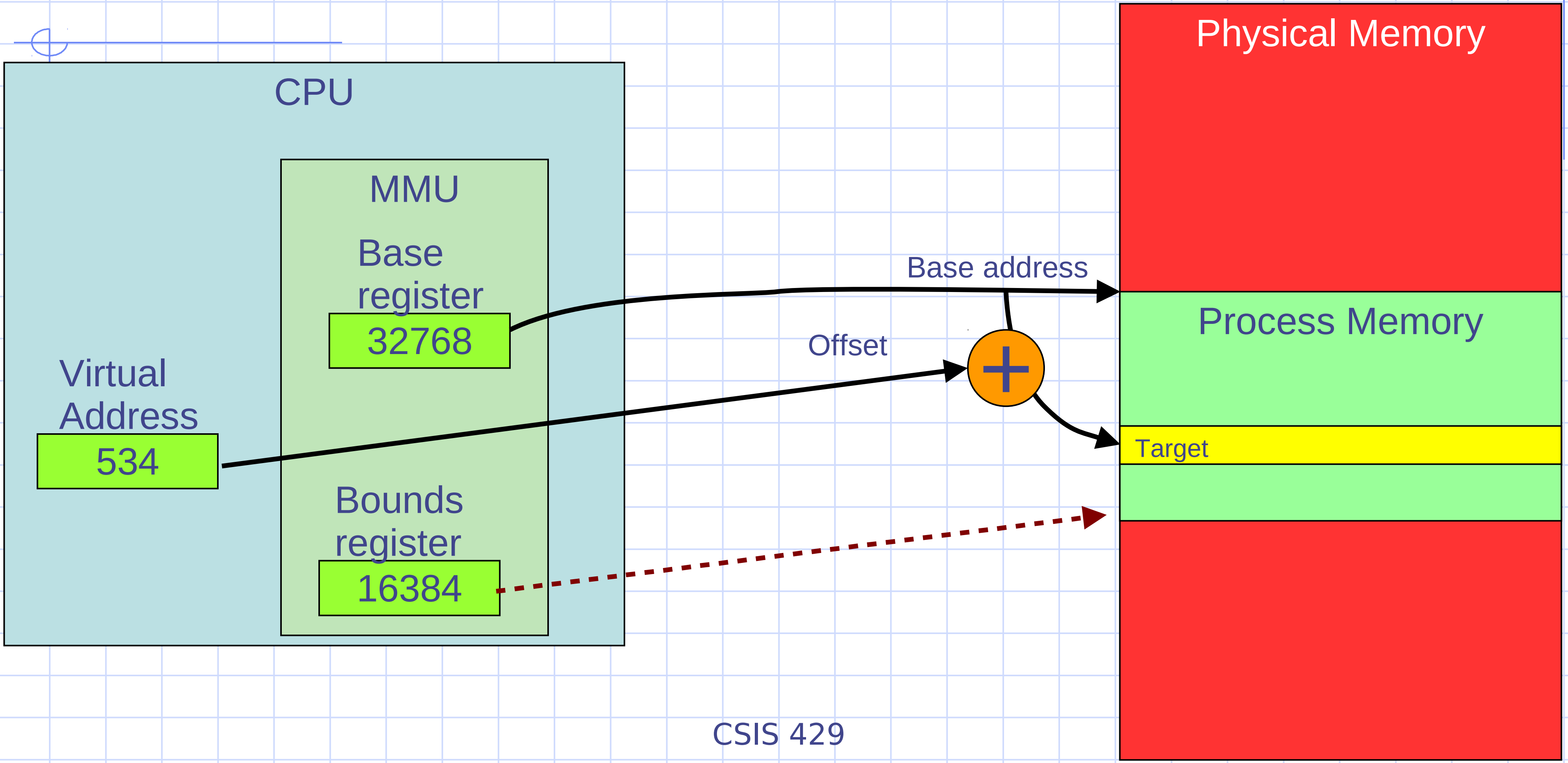
Physical Memory:



Hardware Support for Base and Bounds Registers



Abstract view of Base and Bounds



Managing processes w/ Base+Bounds:

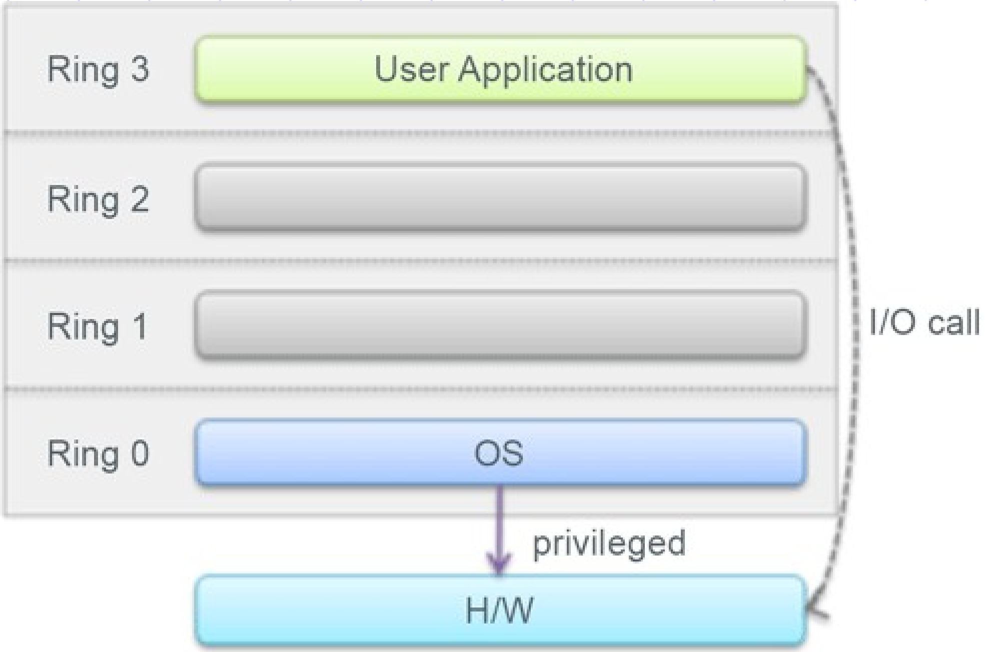
Context switch

- Base and bounds registers are part of CPU state
- During context-switch:
  - Change to “privileged mode” in kernel
  - Save base, bounds registers of old process
  - Load base, bounds registers of new process
  - Change to user – non-privileged – mode and resume

What would happen if we did not change base, bounds during context switch?

Intel x86 Hardware Access Rings

Intel x86 has 4 levels of hardware access authority: Ring 0 is most privileged and Ring 3 is "user space" - less privileged.



Cubrid.org

Pros/Cons of Base+bounds:

Advantages:

- ✔ Dynamic relocation is possible
- ✔ Processes are isolated – secure
- ✔ Inexpensive: 2 registers + some logic, ALU
- ✔ Fast: Compare, Add can be done in 2 clock cycles.

Disadvantages:

- ✗ Memory for each process has to be contiguous.
- ✗ Must allocate memory that may not be used.
- ✗ Sharing will need additional memory “segments” & logic
- ✗ Process address space may be smaller than necessary

Other OS issues with Dynamic Reloc.

OS will need to do other accounting:

- "Free list" to keep track of which parts of memory are available to be allocated to a new process
- When a process ends, put its memory into free list.
- When a process starts, find a block of memory and take it out of the free list.
- When all of memory has been allocated and a new process starts, OS could save the memory of an "inactive" process to disk and use that for the new process → swapping.