CS 452 Operating Systems

Main Memory

Dr. Denton Bobeldyk

Main Memory

- **Background**
- Swapping
- Contiguous Memory Allocation
- Segmentation

Background

- **Basic Hardware**
- Address Binding
- Logical vs Physical Address Space
- Dynamic Loading
- Dynamic Linking and Shared Libraries

Basic Hardware

- CPU can directly access (for general storage):
  - Registers built into the processor
  - Main memory (also includes cache)
  - If the data isn't in one of these locations, the CPU can not operate on them

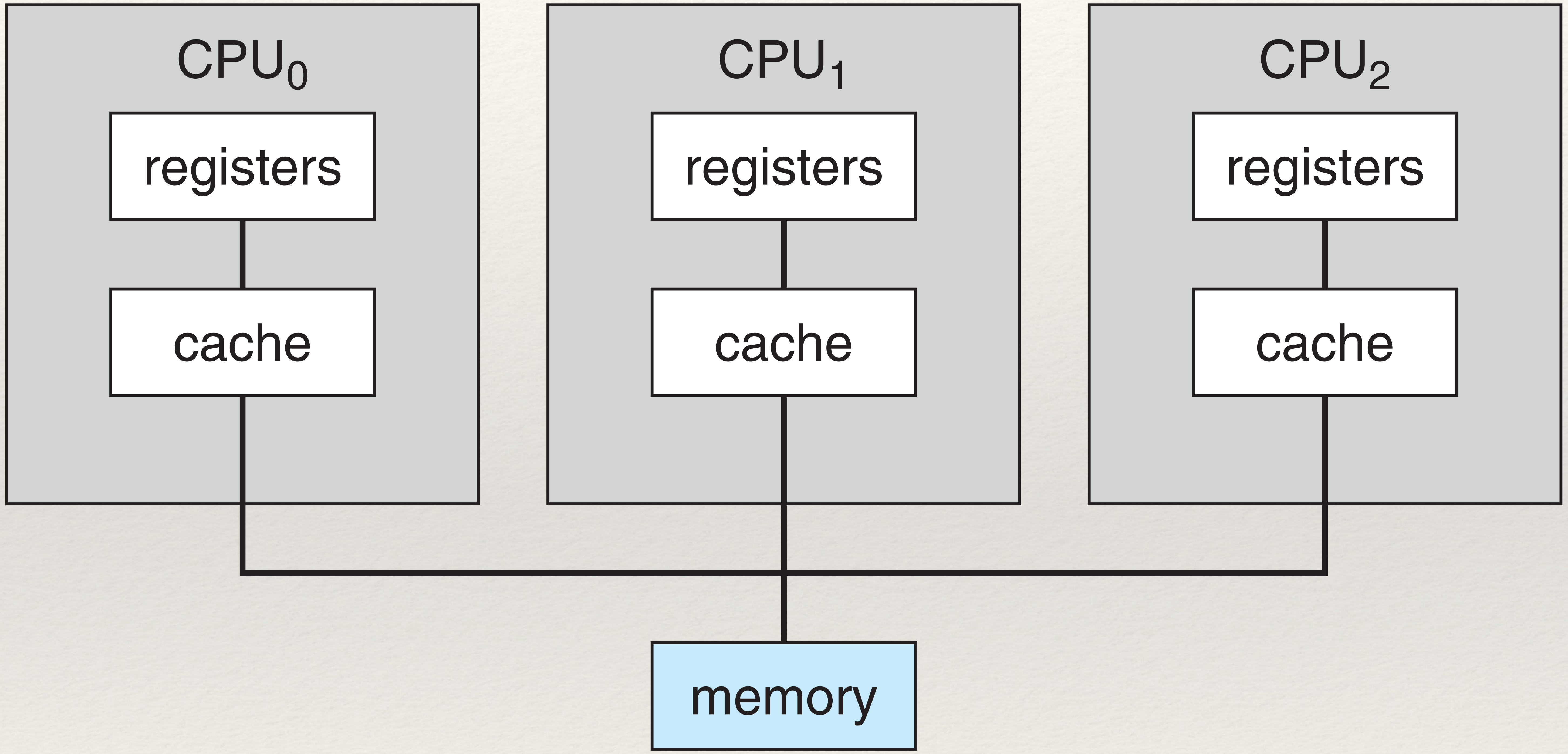
How much main memory on a system?

- Individual exercise:
  - How much RAM for a home PC? Find prices for a few…
  - How much RAM is available for:
    - Nvidia Jetson
    - Nvidia Jetson Nano
    - Raspberry PI

Basic Hardware

- Registers - fast access, generally within one cycle of the CPU clock
- Main Memory - accessed via memory bus. Access time can take many cycles
  - The processor normally needs to **stall** while waiting

Slow Memory Access Solution



Solution: Add cache to speed up memory access

Individual Exercise

- How many cpu cycles does it take to access:
  - Main Memory
  - Cache

Individual Exercise

- How many cpu cycles does it take to access (rough approximates):
  - Main Memory: 100-200 cycles
  - Cache
    - L1: 1-4 cycles
    - L2: 5-12 cycles
    - L3: 12-40 cycles (or more)

Source: ChatGPT (need a better source)

Individual Exercise 5-10 minutes

How much cache is available??

Lookup a CPU manufacturer (e.g., Intel, AMD) and find 3 different CPU models. Determine how much L1/L2/L3 cache they each have. Also include the price in your findings.

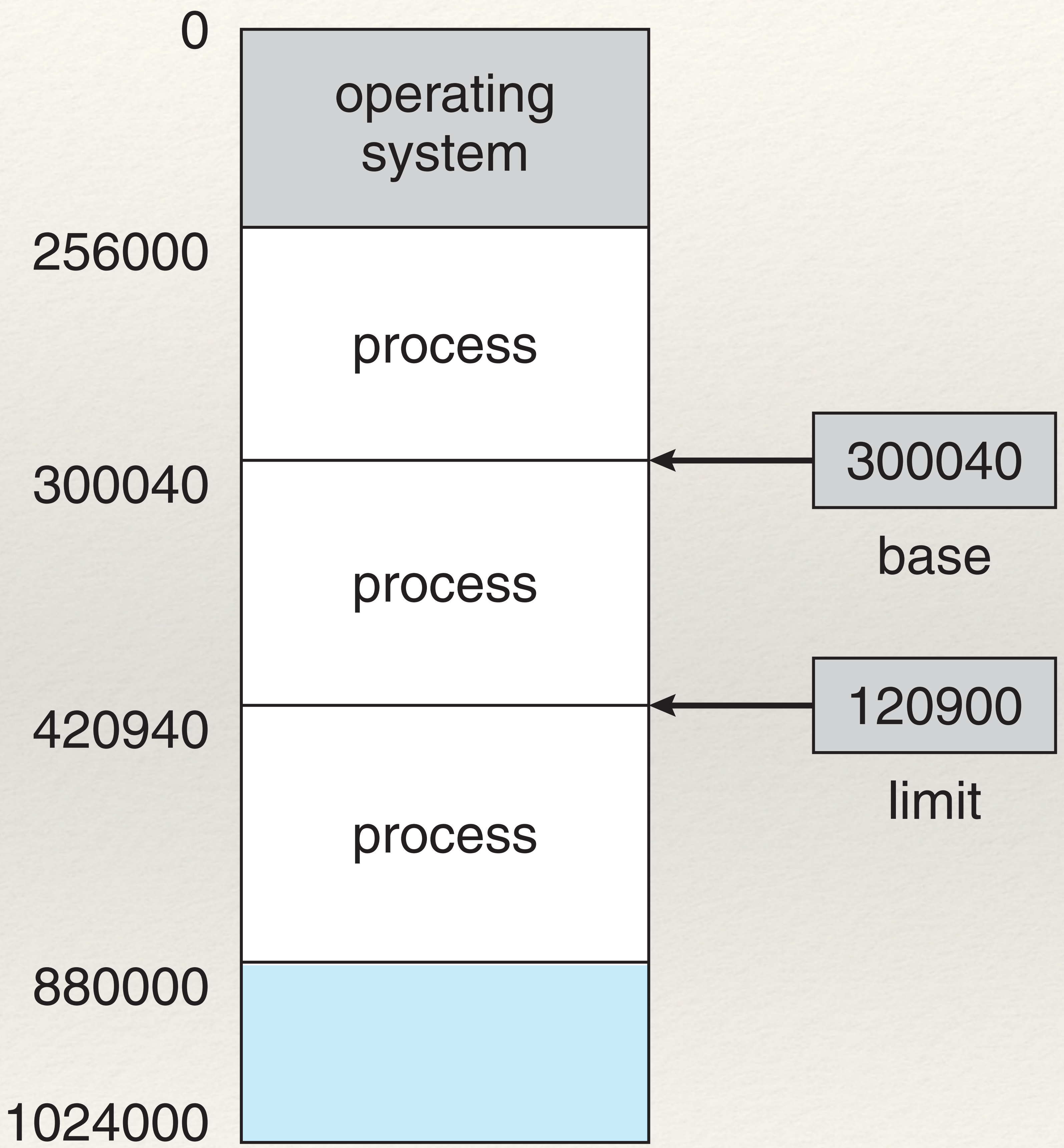
Memory Access - Protection

- Concerned not only about speed, but security
- Need to ensure each process is only accessing it's own memory space
- Create separate memory space for each process

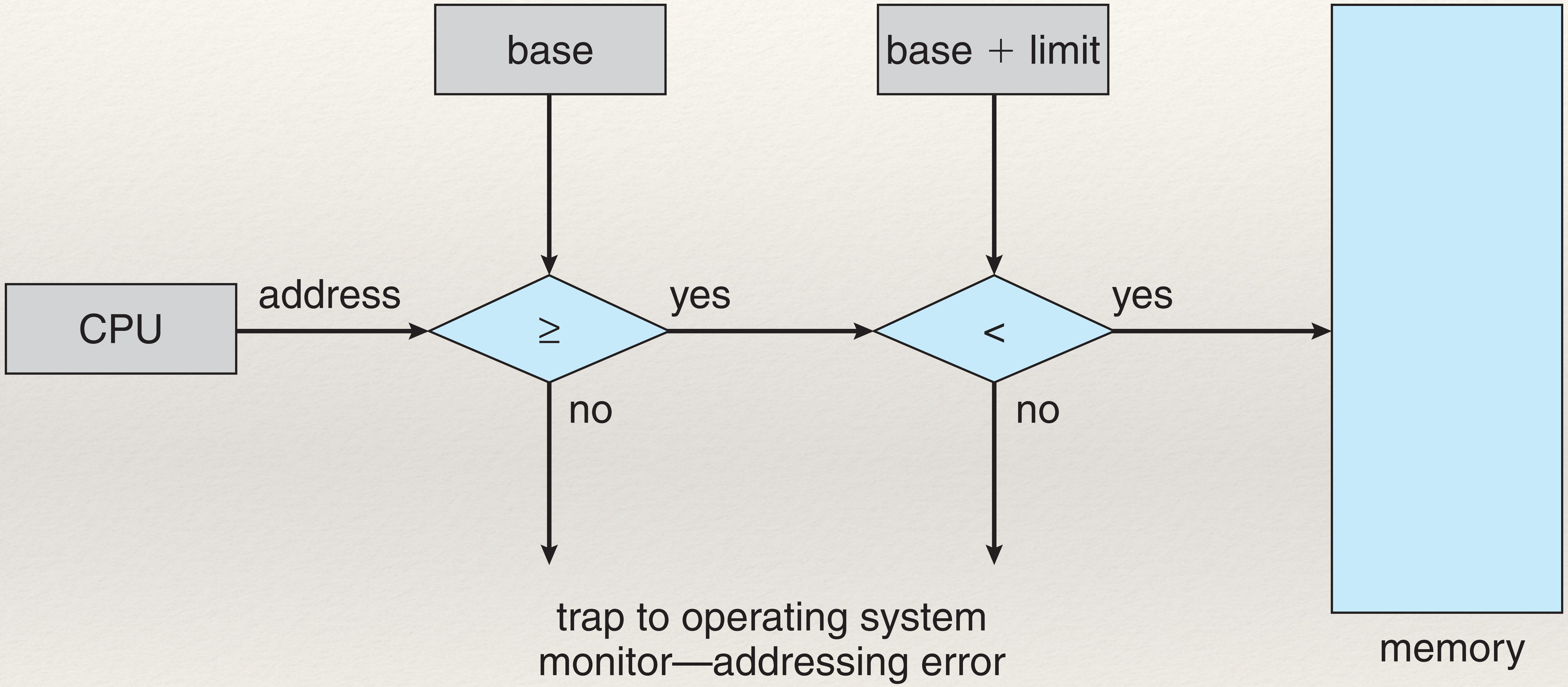
Memory Access - Protection

- Range of address spaces for each process
  - Base register - smallest legal physical memory address
  - Limit register specifies the size of the range

Memory Protection



Memory Protection



Memory Protection

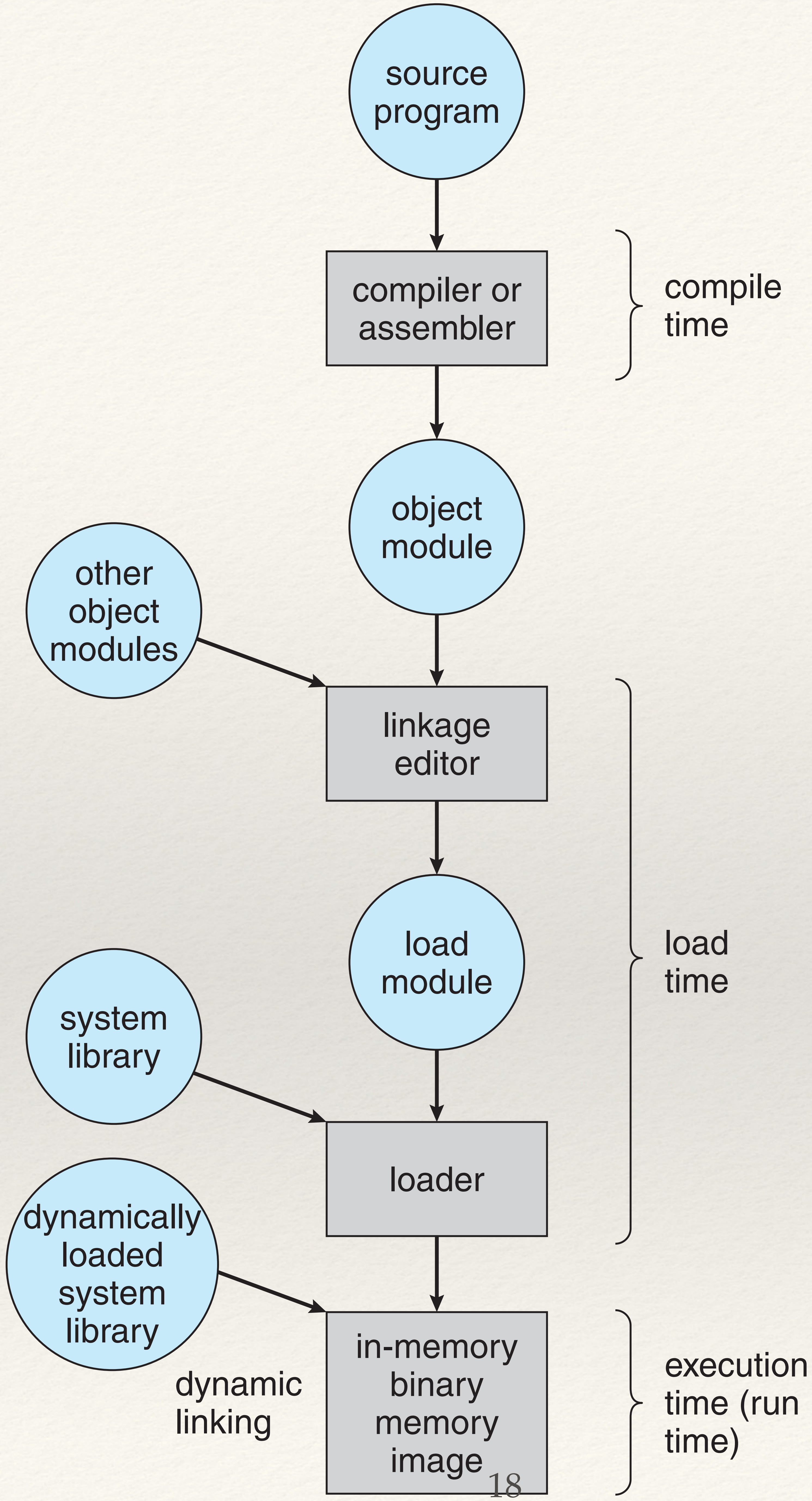
- Only the operating system can change the value of the base and limit
- Operating system, executing in kernel mode, has unrestricted access to all memory locations

Background

- Basic Hardware
- **Address Binding**
- Logical vs Physical Address Space
- Dynamic Loading
- Dynamic Linking and Shared Libraries

Loading a program into memory

- A program must be brought into memory to execute
- The processes on the disk that are waiting to be brought into memory form the **input queue**
- After a process finishes executing, it’s memory is declared available


source program
compiler or assembler
compile time
object module
other object modules
linkage editor
load module
system library
loader
load time
dynamically loaded system library
dynamic linking
in-memory binary memory image
execution time (run time)

Address Binding

- **Compile time** - If you know where the code will reside in memory, then **absolute code** can be generated.
  - If location changes, need to recompile
- **Load time** - compiler generates **relocatable code**. Binding to memory happens at load time.
  - If location changes, reload the user code to incorporate this change
- **Execution time** - process can be moved during its execution from one memory segment to another.
  - Most general-purpose operating system use this method

Memory

- Basic Hardware
- Address Binding
- **Logical vs Physical Address Space**
- Dynamic Loading
- Dynamic Linking and Shared Libraries

Logical vs Physical Address Space

- Logical address - address generated by the CPU
- Physical address - actual address loaded into the memory-address-register of the memory

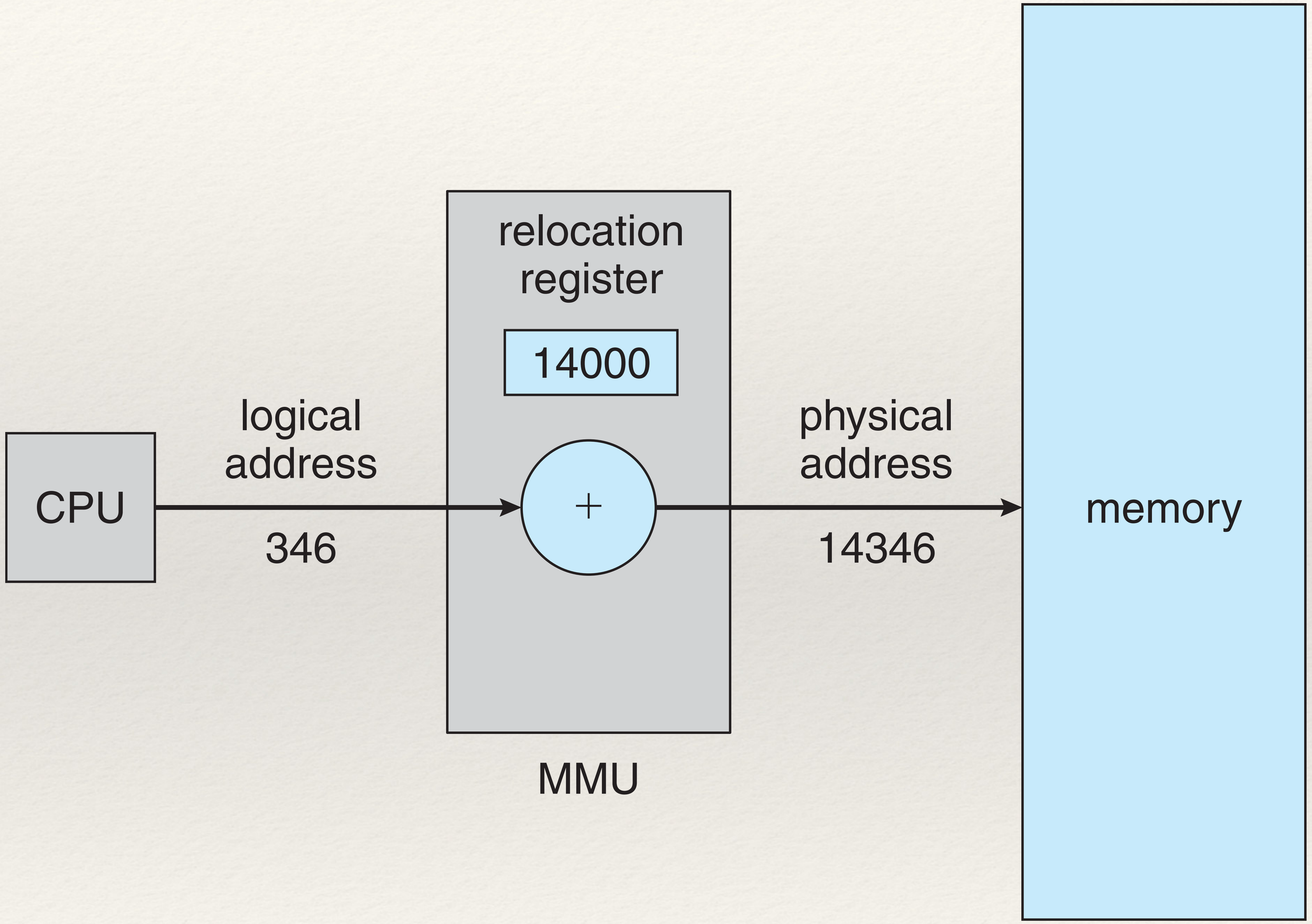
Logical vs Physical Address Space

- Same address: compile-time and load-time address binding
- Different logical and physical addresses: execution-time address-binding.

Logical vs Physical Address Space

- Logical address space: set of all addresses generated by a program
- Physical address space: set of all physical addresses corresponding to the logical addresses.
- Mapping is performed by the memory-management unit (MMU)

Logical vs Physical Address Space



Logical vs Physical Address Space

- The user program never sees the real physical address
- Logical address range: 0 to *max*
- Physical address range: R + 0 to R+*max*

Memory

- Basic Hardware
- Address Binding
- Logical vs Physical Address Space
- **Dynamic Loading**
- Dynamic Linking and Shared Libraries

Dynamic Loading

- The main program is loaded into memory and executed
- A routine isn't loaded until it is called
- The calling routine checks to see if it's in memory, if not the relocatable linking loader is called to load it into memory

Dynamic Loading

- Advantageous especially when large amounts of code handle infrequently occurring cases (e.g., error routines)
- Programmers can design their program to take advantage of this method

Memory

- Basic Hardware
- Address Binding
- Logical vs Physical Address Space
- Dynamic Loading
- **Dynamic Linking and Shared Libraries**

Dynamic Linking and Shared Libraries

- Dynamically linked libraries - system libraries linked to a user program when they are run
- Static linking - libraries that are combined by the loader into the binary image
- Dynamic linking - linking is postponed until execution time
    - Useful for system libraries, so every program doesn't have to load the same library into memory

Dynamic Linking and Shared Libraries

- Stub - a small piece of code that indicates how to locate the appropriate memory-resident library routine (or how to load if it's not already present)
- The first time the stub is executed, it will check to see if the routine is already in memory, if it's not, it will load it into memory
- When the stub is executed the second time, it replaces itself with the address of the routine and executes the routine.
    - No cost for dynamic linking the second time

Dynamic Linking and Shared Libraries

- Library updates - can be replaced with a new version and all programs that reference the library can automatically update to use the new version
- Without dynamic linking, all programs would need to be relinked to gain access to the new library

Dynamic Linking and Shared Libraries

- Version information can be included, so programs will not accidentally use a library they may be incompatible with

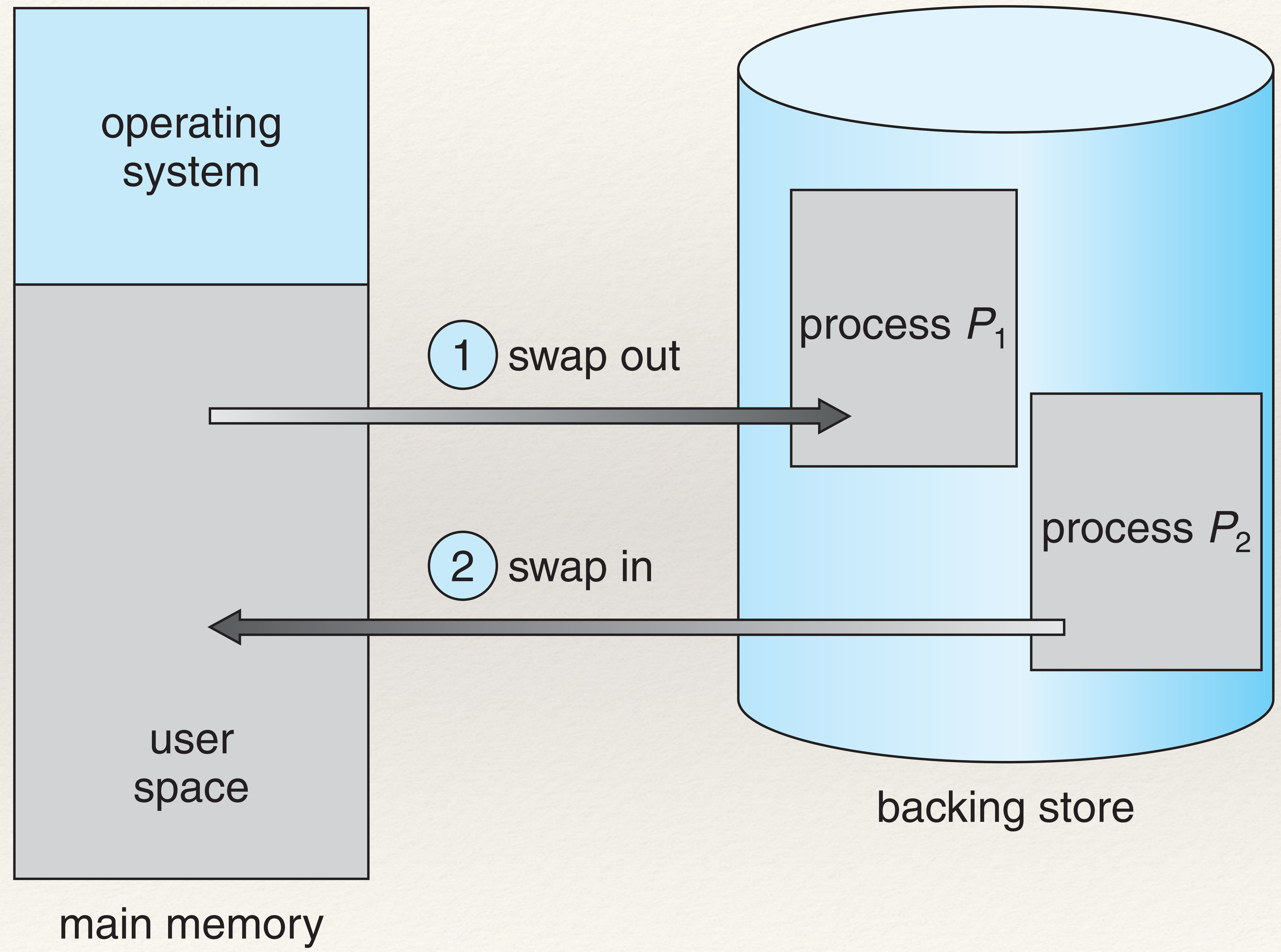
Dynamic Linking and Shared Libraries

- Operating system needs to be utilized to check another process's memory space for routines as user processes can not access another processes memory space

Main Memory

- Background
- **Swapping**
- Contiguous Memory Allocation
- Segmentation

Swapping


operating
system
user
space
main memory
1 swap out
2 swap in
process P_1
process P_2
backing store

Swapping

- System maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory ready to run
- Dispatcher loads the next process into memory, if there is not enough free memory for it, the dispatcher swaps out a process currently in memory

Swapping

- Swapping to the backup store is expensive!
- For example, 100MB program transfers to a standard hard disk at the transfer rate of 50MB/second. That takes 2 seconds!

Swapping

- A process should be completely idle
- A process should not be waiting for I/O??

Swapping

- A process awaiting I/O could still be swapped out if:
  - Only execute I/O operations into operating system buffers
  - Transfer then occurs from the OS buffer to the process memory.
    - This is called **double buffering**
      - This causes additional overhead!!

Swapping

- Modified versions of swapping exist:
  - Swap a portion of the process, rather than the entire process
  - Swapping can be disabled but will start if the amount of free memory falls below a certain threshold

Swapping on Mobile Systems

- Mobile systems do not typically support swapping
  - Generally use flash memory rather than more spacious hard disks for persistent storage (limiting the space to swap)
  - Flash memory will tolerate a limited number of writes before it starts to become unreliable

Swapping on Mobile Systems

- Apple iOS asks applications to relinquish memory
- Applications can be terminated if they don't free up the memory…

Swapping on Mobile Systems

- Android does not support swapping
- May terminate a process if insufficient memory is available
  - Android writes its application state to flash memory so that it can be quickly restarted

Main Memory

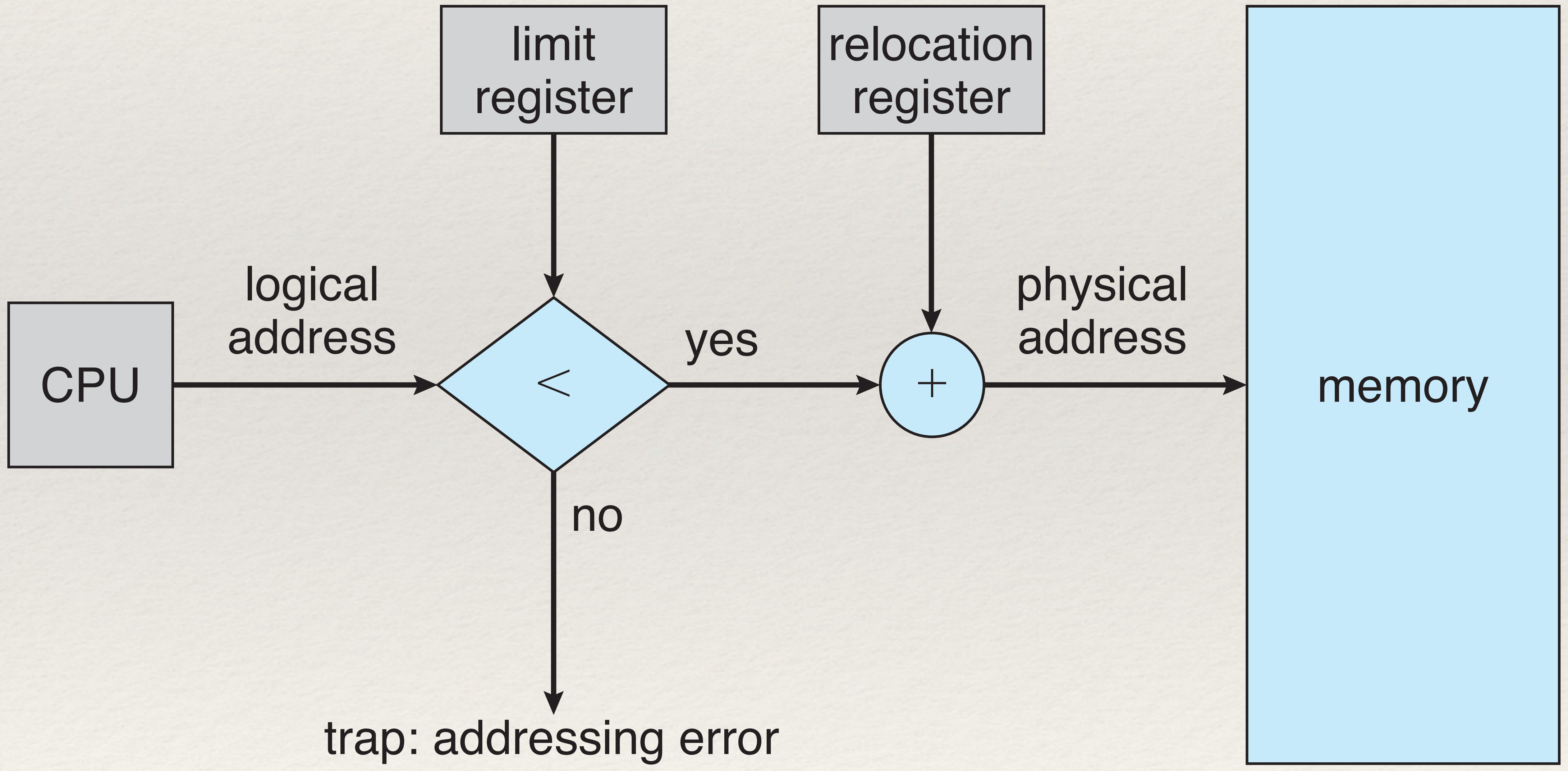
- Background
- Swapping
- **Contiguous Memory Allocation**
- Segmentation

Contiguous Memory Allocation

- Memory is usually divided into two partitions
  - Resident Operating System
  - User Processes
- Contiguous memory allocation
  - Each process is brought into a single section of memory

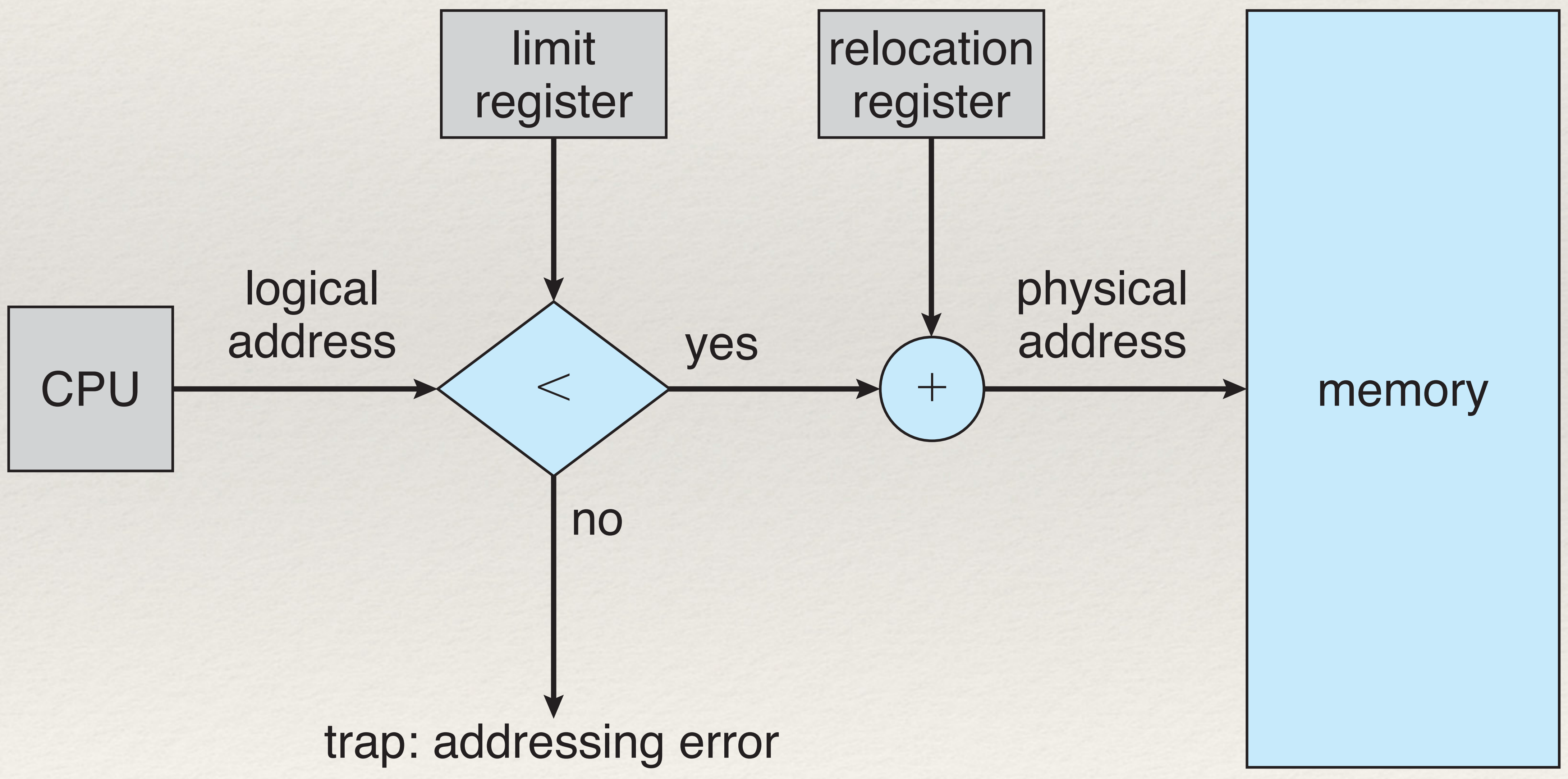
Memory Protection

- Need to prevent processes from accessing memory they don't own
  - Define a limit and relocation register



Memory Protection

- During the context switch, the dispatcher (OS) loads the values for the relocation and limit registers



Memory Allocation

- Divide memory into fixed size partitions
    - Each partition contains one process
    - Number of processes is bound by number of partitions?
- Variable partition scheme
    - Initially one large 'hole', eventually a set of 'holes'

Memory Allocation

- Processes are allocated space (loaded into memory) and can then compete for CPU time
- When a process terminates, it releases its memory

Memory Allocation

- Memory is allocated to each process from the input queue until no more memory or no more process are available
- If more processes are available, the OS can wait until a large enough memory block (hole) frees up, or move onto the next process in the input queue that may have a smaller memory requirement.

Memory Allocation

- Dynamic storage-allocation problem:
  - How to satisfy a request of size n, from a list of free holes
    - First-fit
    - Best-fit
    - Worst-fit

Memory Allocation

- First-fit
  - Allocate the first hole that is big enough
  - Search can be from the beginning or end of the list
- Best-fit
  - Allocate the smallest hole that is big enough
- Worst-fit
  - Allocate the largest hole

Fragmentation

- External fragmentation
  - Enough memory to satisfy a request, but the memory isn’t contiguous
- Amount of external fragmentation that exists will depend on the system and the algorithm used to assign memory holes
- Up to one third of memory may be unusable
  - Known as 50 percent rule
    - Given N allocated blocks, another 0.5 N blocks will be lost to fragmentation

Fragmentation

- Internal fragmentation
  - Unused memory internal to a partition
- How is it caused?
  - Processes can be allocated a fixed size block of memory
  - Process dynamically allocated a block slightly larger than its request
    - Request 18,462 bytes, available hole of 18,464. More overhead to keep track of those 2 bytes than to just allocate them to the process

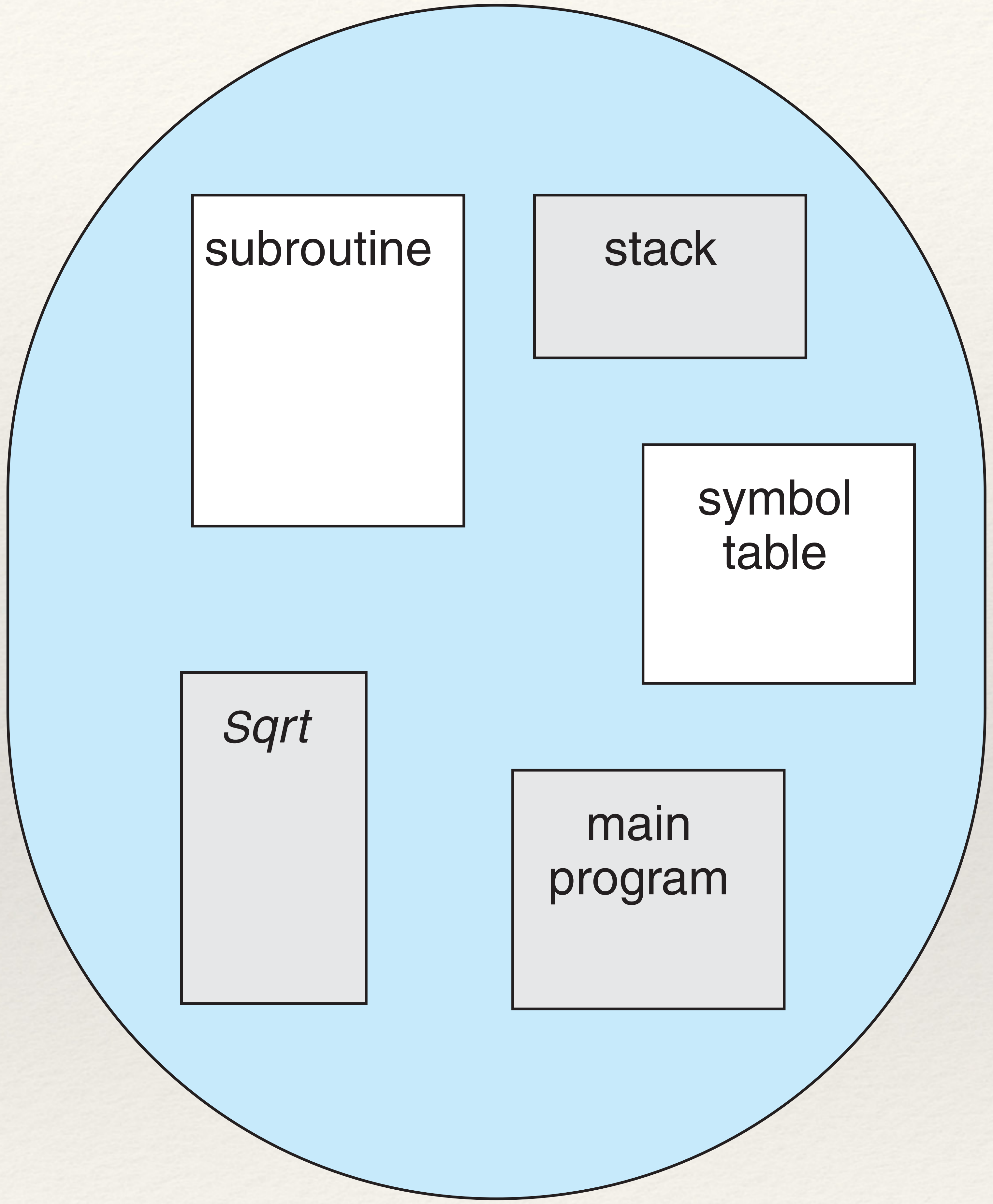
Main Memory

- Background
- Swapping
- Contiguous Memory Allocation
- **Segmentation**

Segmentation

- Memory management scheme where the address space is viewed as a collection of segments
- Each logical address consists of a two tuple:
    - <segment-number, offset>

Segmentation



logical address

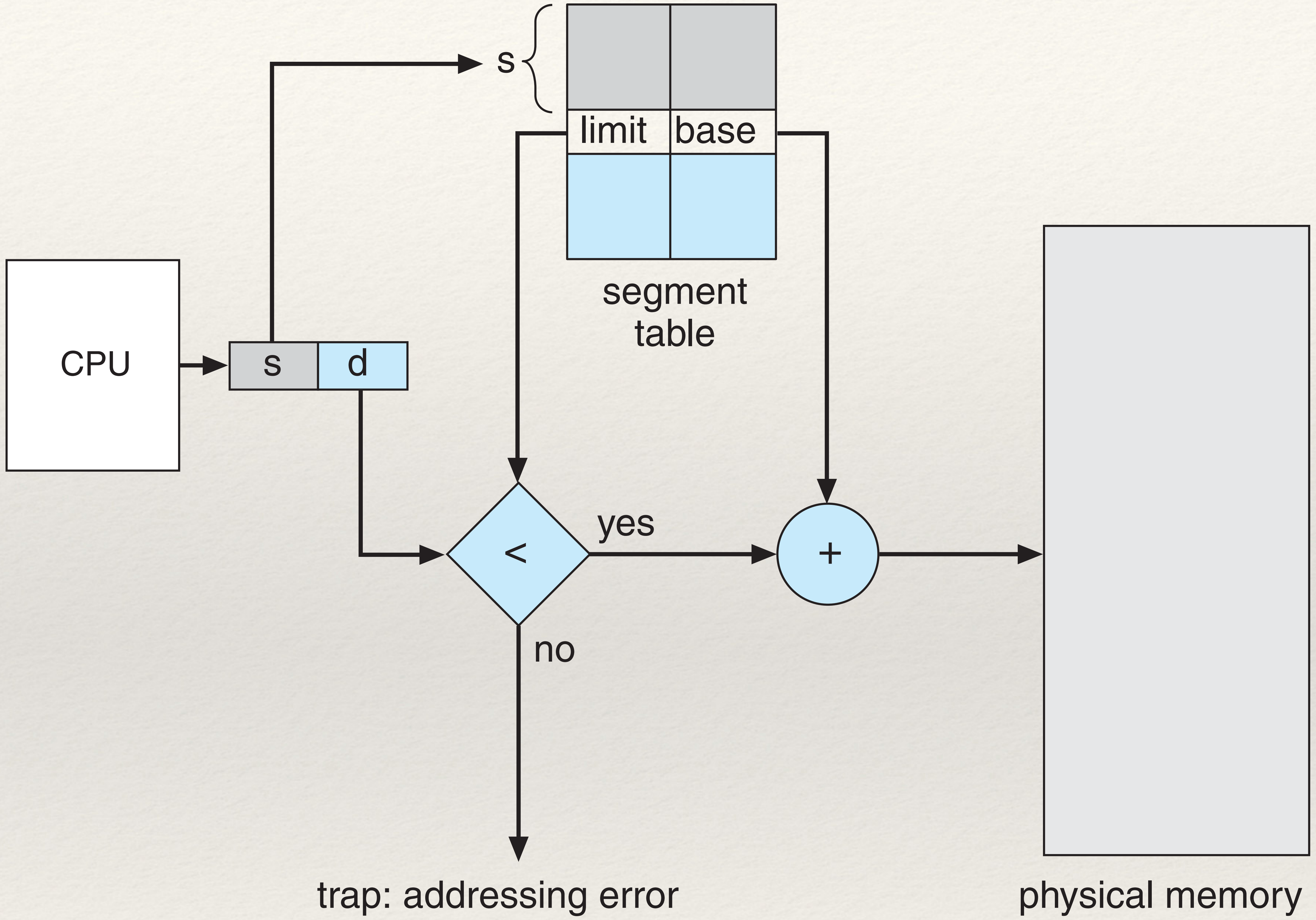
Segmentation

- C compiler might create separate segments for:
  - The code
  - Global variables
  - The heap, from which memory is allocated
  - The stacks used by each thread
  - The standard C library

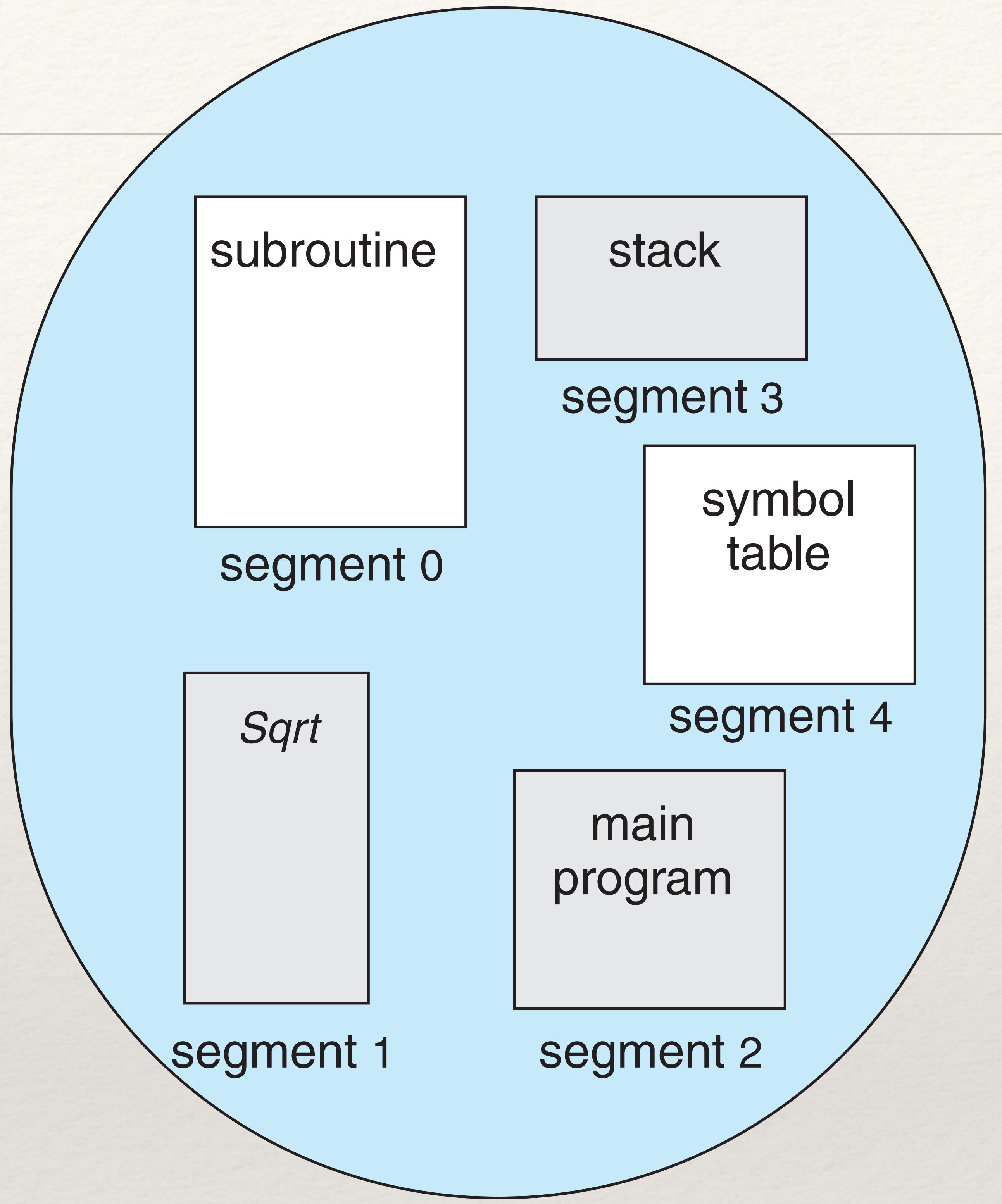
Segmentation

- Segment table keeps track of mapping logical to physical address space
- Table consists of:
  - Segment base: starting physical address
  - Segment limit: length of the segment

Segmentation



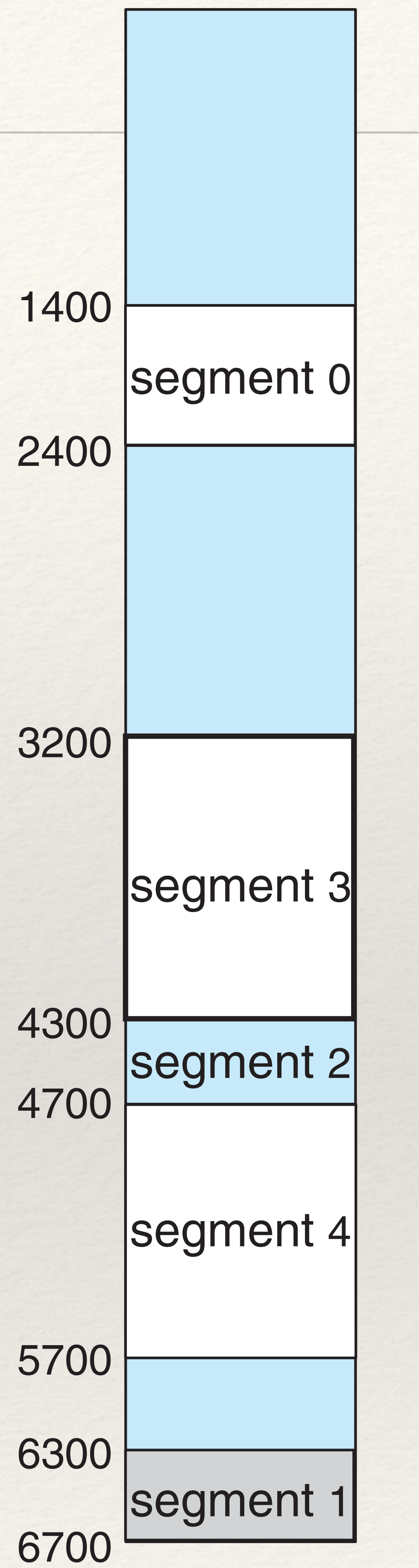
Segmentation



logical address space

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



physical memory

End