KOÇ UNIVERSITY

Memory Management

Didem Unat

Lecture 17

COMP304 - Operating Systems (OS)

So Far

- Process Management
  - Process Creation/Termination
  - Process State, PCB
  - Multithreading – Pthreads
  - Process Scheduling
  - Synchronization
  - Deadlocks
- Rest of the semester (9th edition)
  - Memory Management – Chapter 8
  - Virtual Memory Management – Chapter 9
  - File System – Chapter 10

Today’s Buzzwords

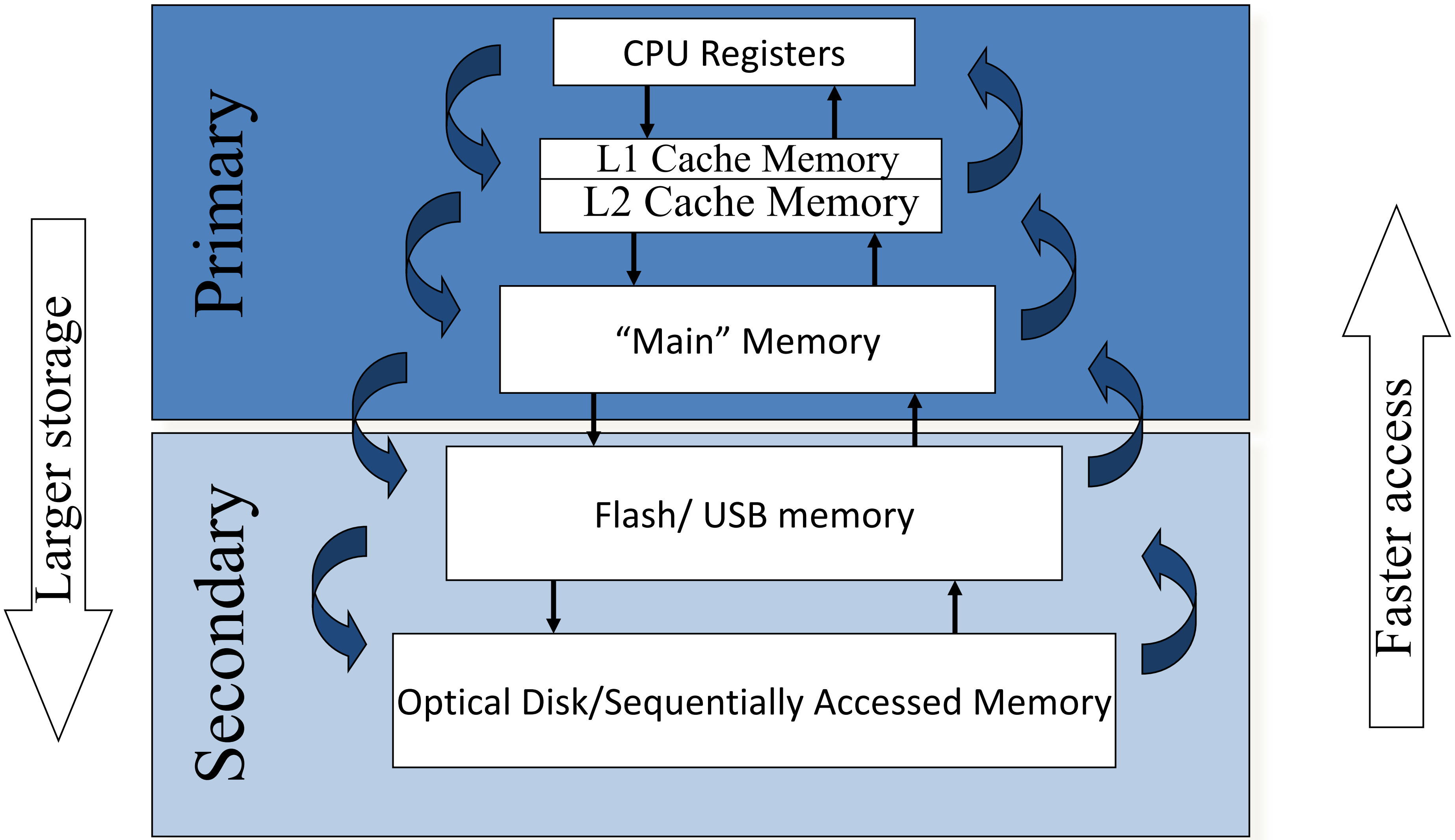
- Base and Limit Registers
- Static vs Dynamic Linking
- Address Binding
- Logical (virtual) and physical addresses
- Relocation Register
- Memory allocation
- Fragmentation
- Segmentation

CPU Access to Memory

- The CPU reads instructions and reads/writes data from/to memory

Functional interface:
value = read(address)
write(address, value)

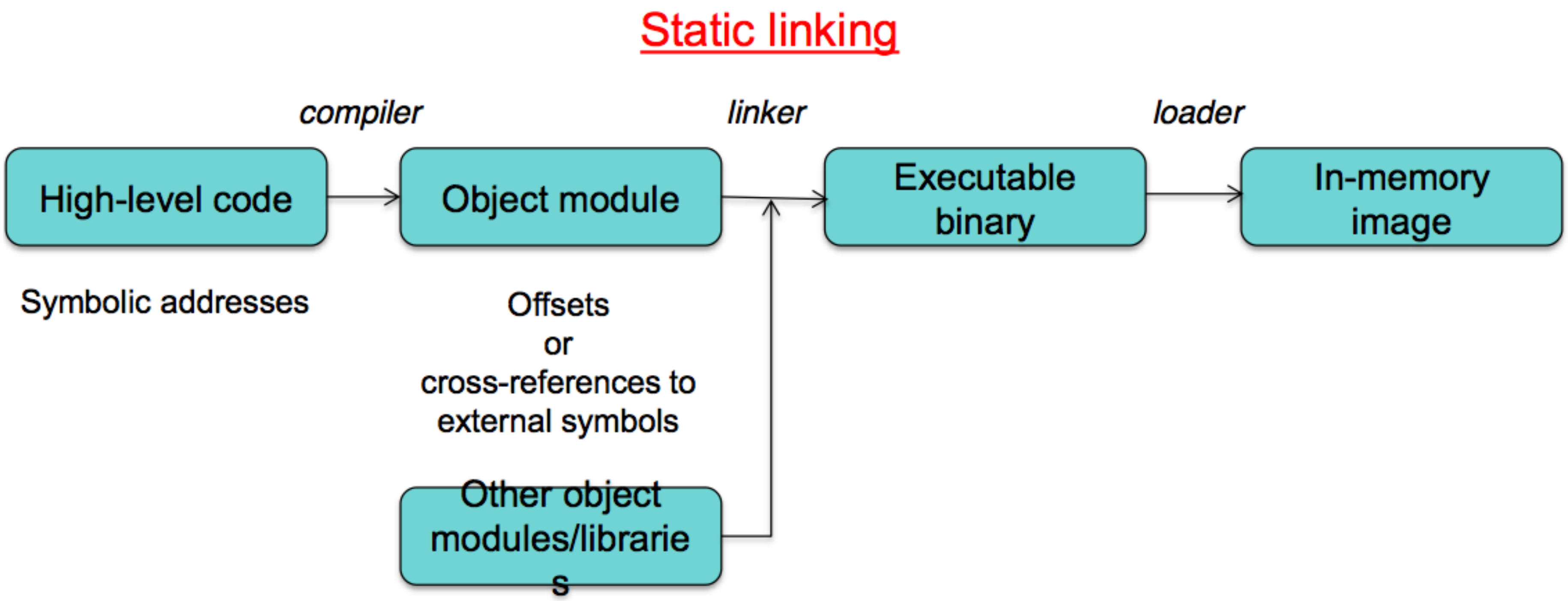
Contemporary Memory Hierarchy



Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a stall
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Programs have references to memory

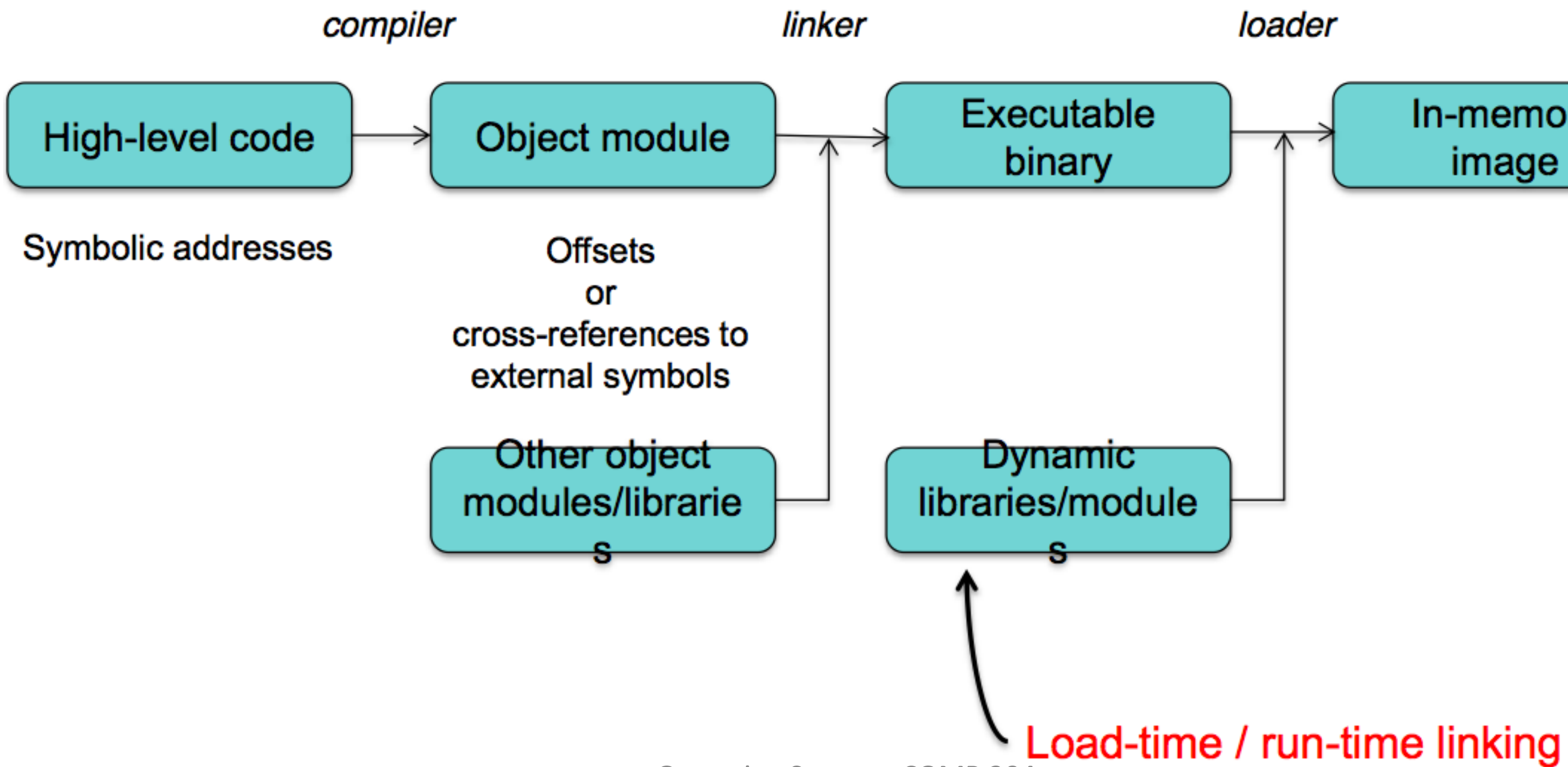


How do programs specify memory addresses?

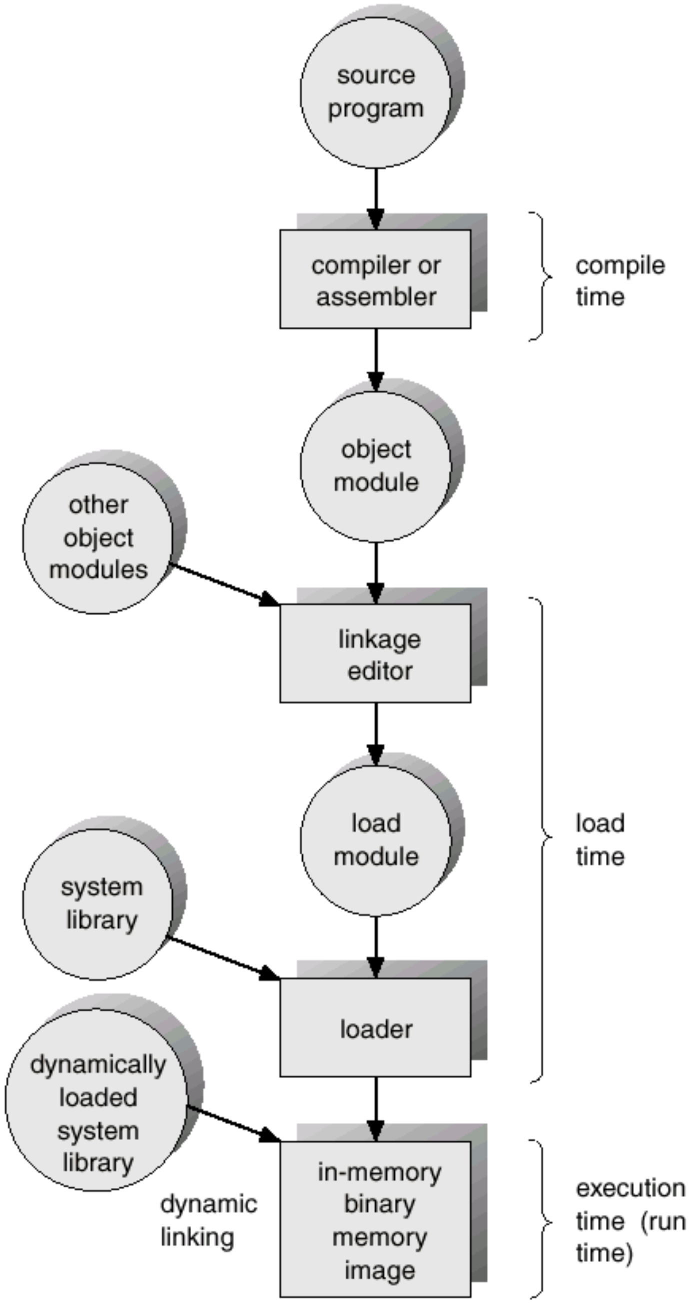
- Absolute code
  - If you know where the program gets loaded (any relocation is done at link time)
- Position independent code
  - All addresses are relative
- Dynamically relocatable code
  - Relocated at load time
- Or ... use logical addresses
  - Absolute code with addresses translated at run time
  - Need special memory translation hardware

Dynamic Linking

- A process loads libraries at load time
  - Symbol references are resolved at load time
- OS loader finds the dynamic libraries and brings them into the process' memory address space



Address Binding



Address binding of instructions and data to memory addresses can happen at three different stages.

Compile time:

If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes.

Load time:

Must generate **relocatable** code if memory location is not known at compile time.

Execution time:

Binding delayed until run time if the process can be moved during its execution from one memory segment to another.

Need hardware support for address maps (e.g., **base** and **limit registers**).

Logical vs. Physical Address Space

- For proper memory management
  - There is **logical address space and physical address space**
  - **Logical address** – generated by the CPU; also referred to as **virtual address**.
  - **Physical address** – address seen by the memory unit.

- Logical and physical addresses are the same in compile-time and load-time address-binding schemes;

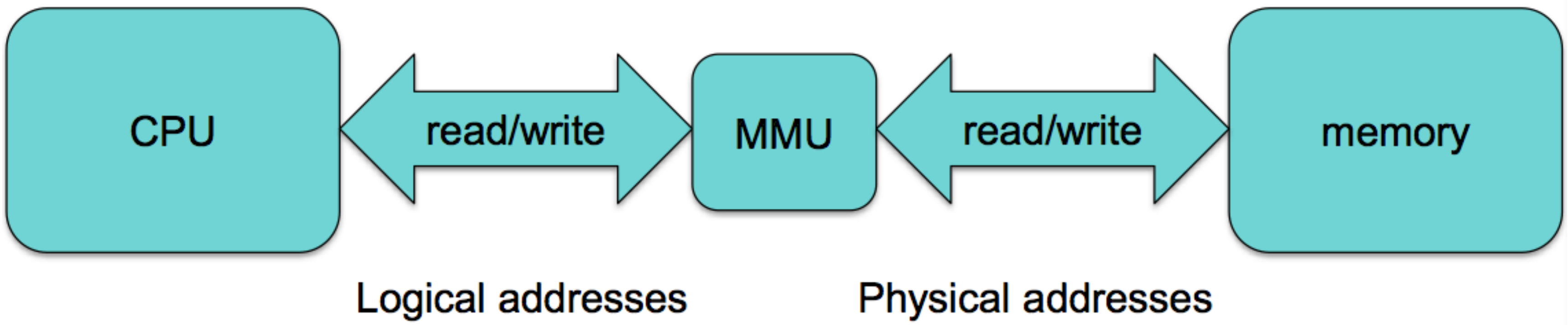
- Logical (virtual) and physical addresses differ in execution-time address-binding scheme.

- The user program deals with logical addresses; it never sees the real physical addresses.
  - Why?

Logical Addressing

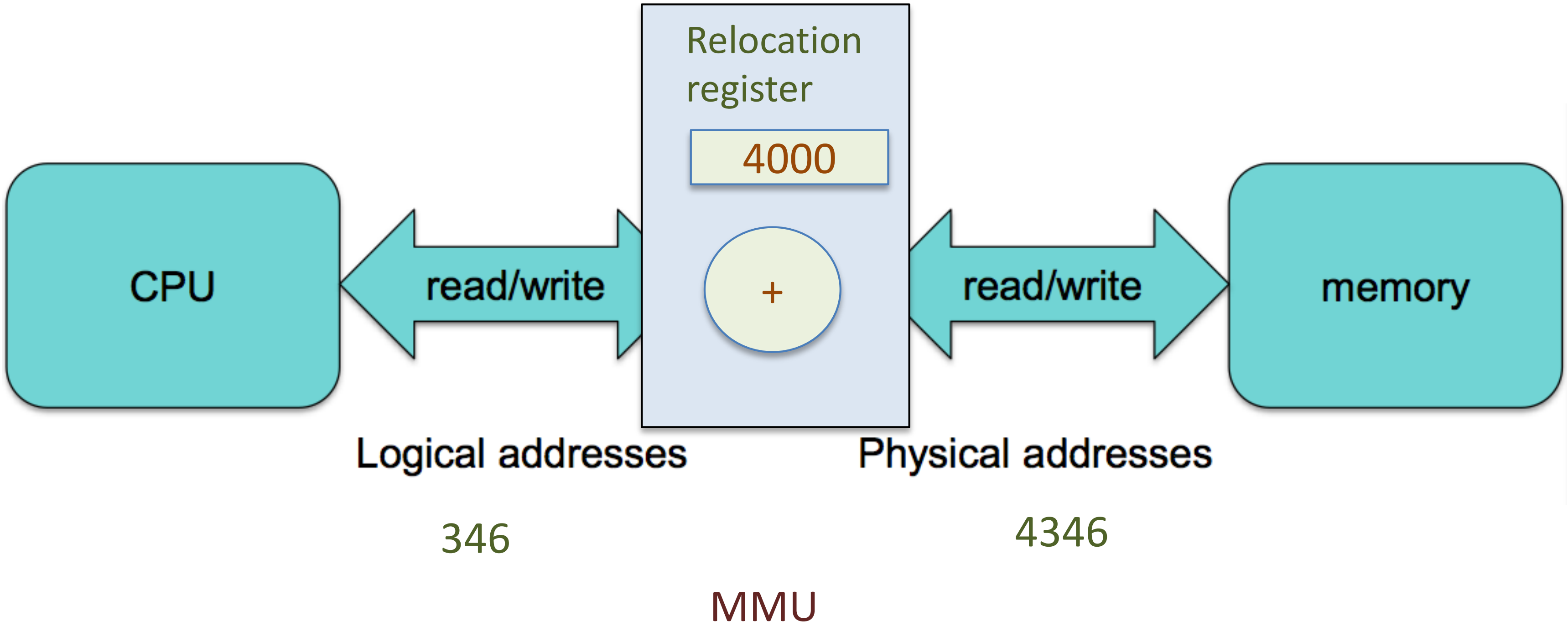
Memory management unit (MMU):

- Real-time, on-demand translation between *logical* (virtual) and *physical* addresses



Relocation Register

- User program only generates logical addresses and thinks that the process runs in location 0 to max.
- In fact, it runs R+0 to R+max - R is the base register (now called relocation register)

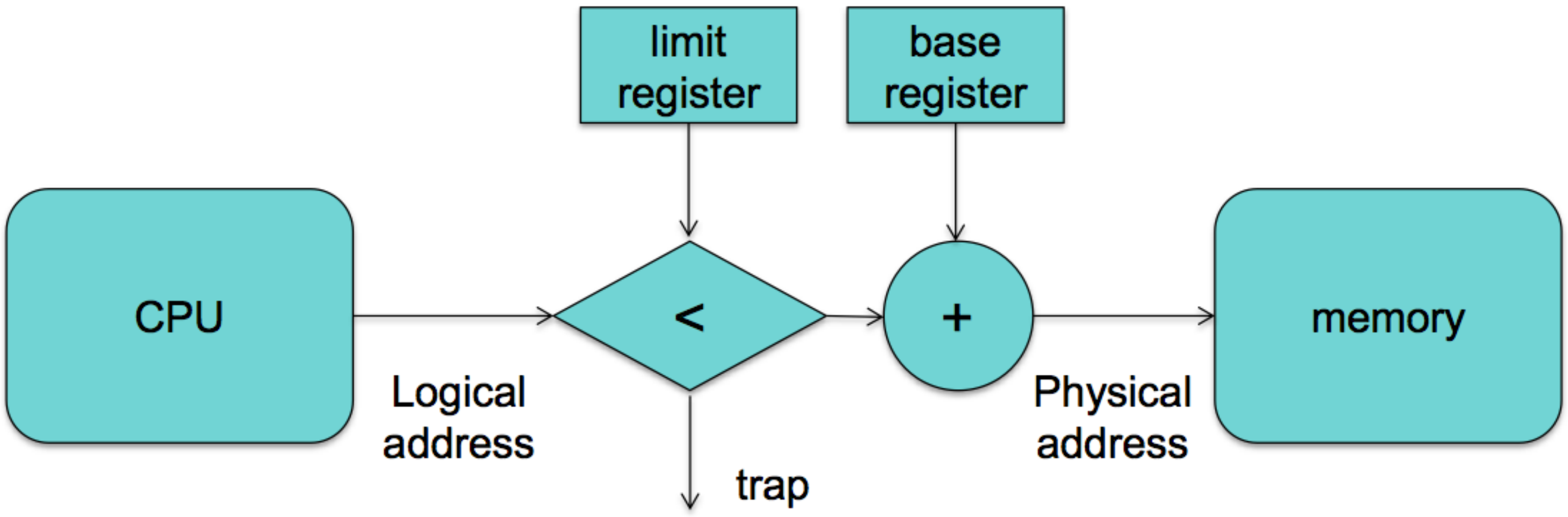


Relocatable Addressing

Base & limit

– *Physical address = logical address + base register*

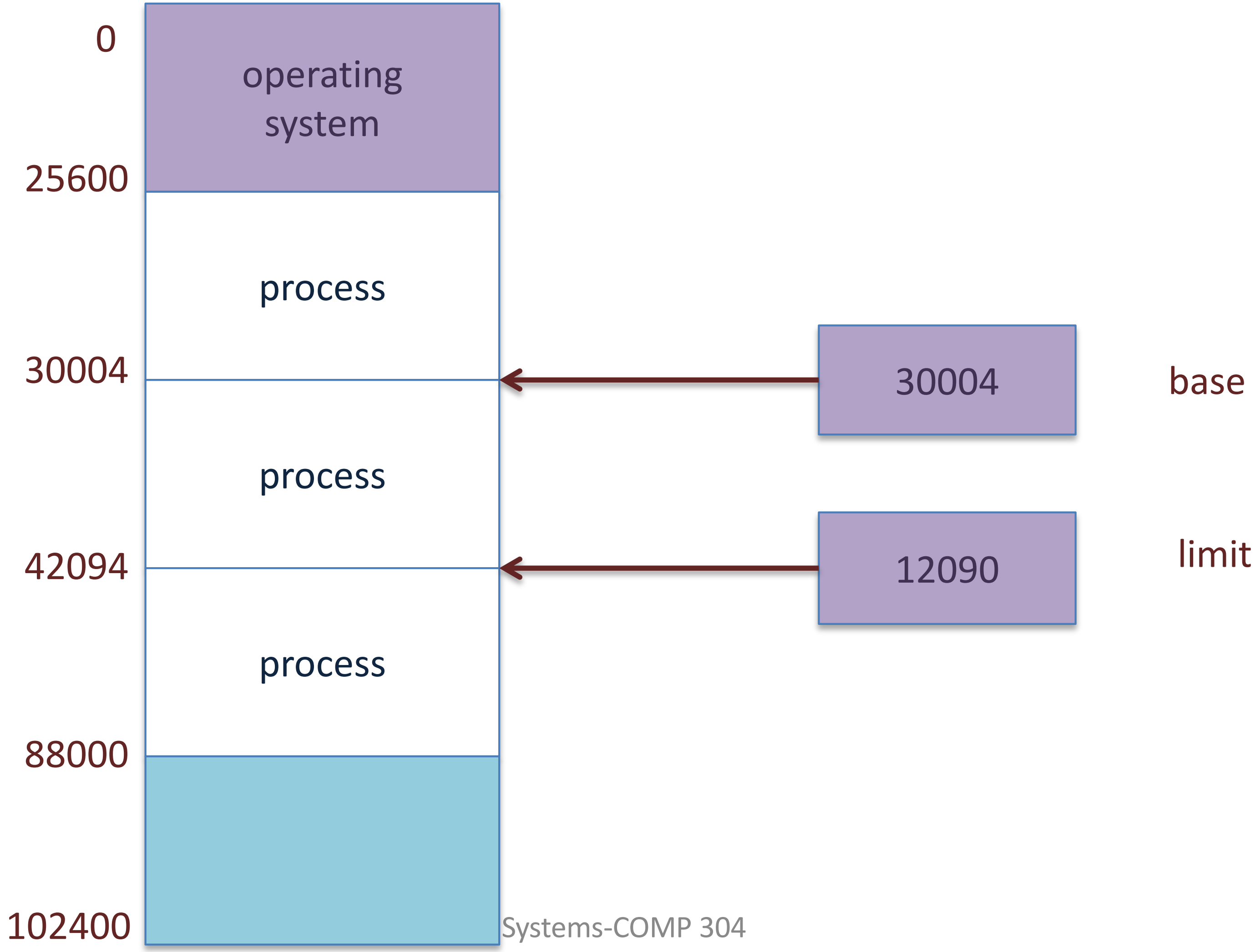
– But first check that: *logical address < limit*



Memory management unit (MMU) maps the logical address dynamically by adding the value in the relocation register.

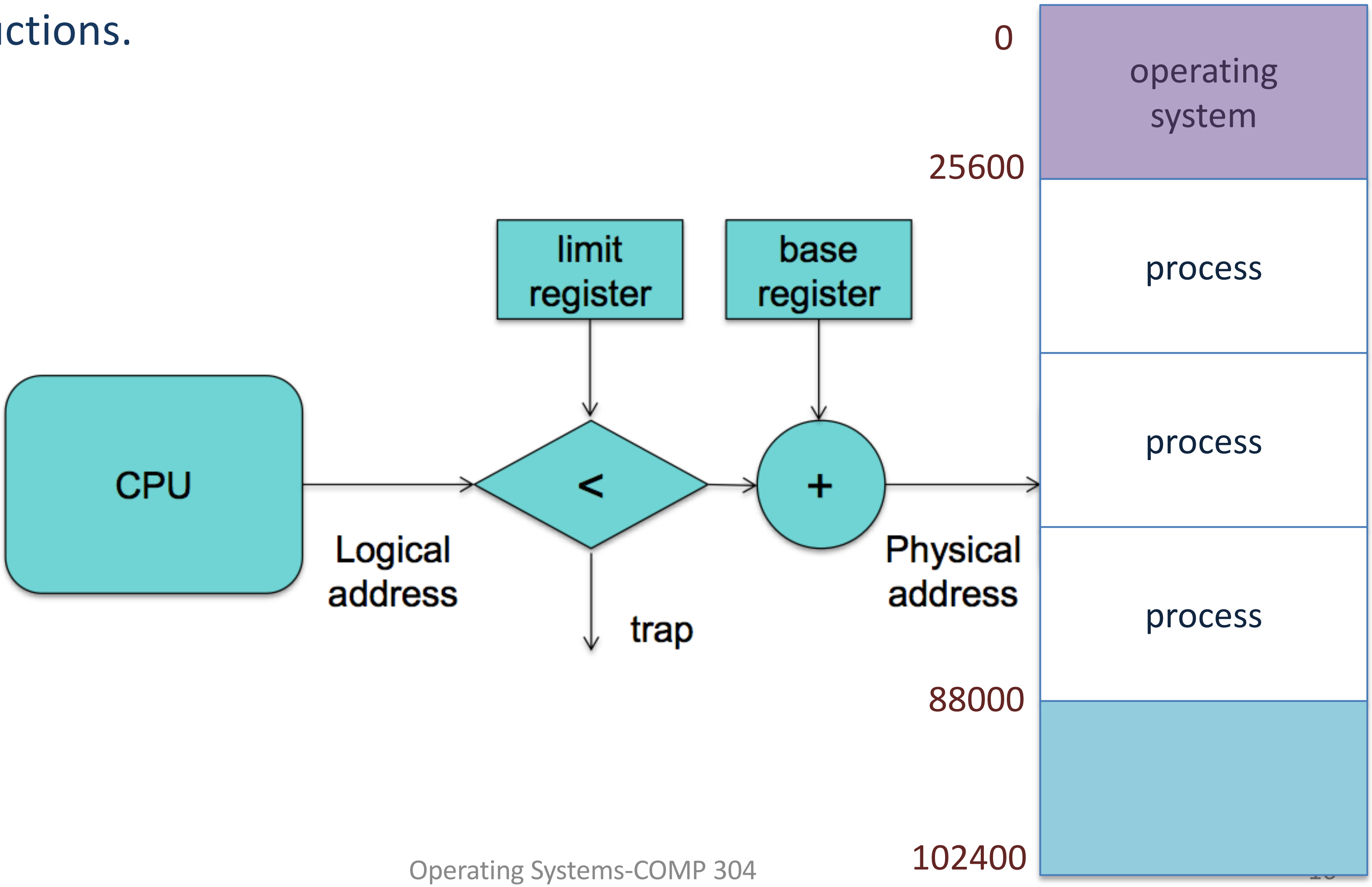
Base and Limit Registers

- A pair of base and limit registers define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between in base and limit for that user



Hardware Protection

- When executing in kernel mode, the operating system has unrestricted access to both kernel and user’s memory.
- The load instructions for the *base* and *limit* registers are privileged instructions.



Memory Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Three methods:
  - Contiguous memory allocation
  - Segmentation
  - Paging

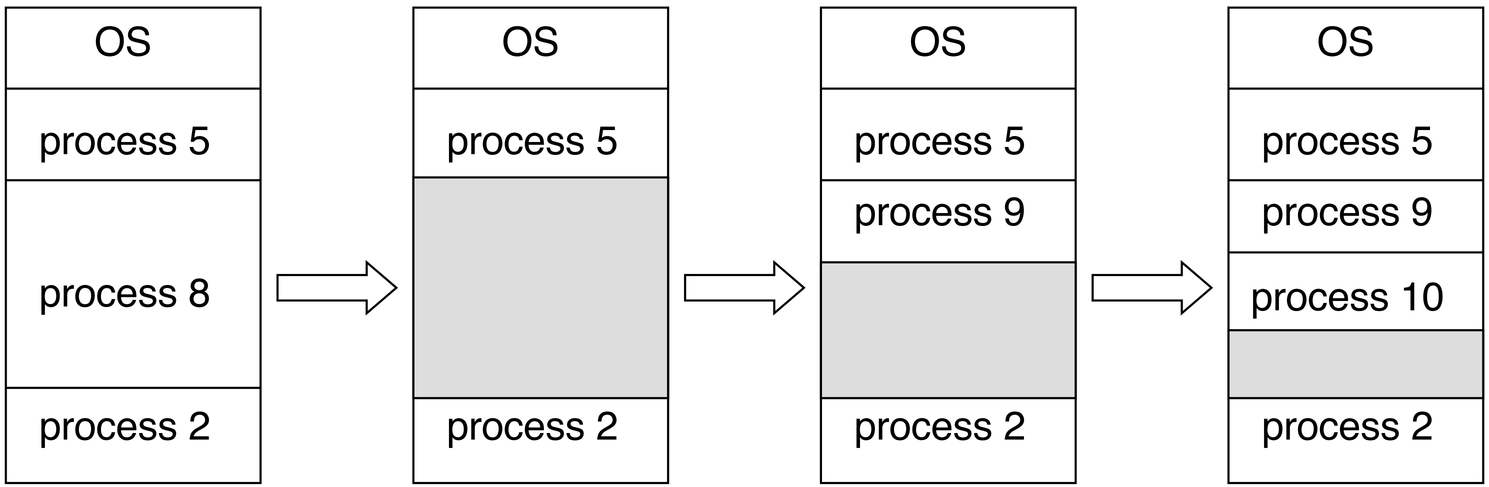
Contiguous Allocation

- Main memory is usually divided into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.

- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base (relocation) register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU (memory management unit) maps logical address dynamically

Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - **Hole** – block of available memory; holes of various sizes are scattered throughout memory.
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it.
  - Operating system maintains information about:
    allocated partitions and free partitions (holes)



Dynamic Storage-Allocation Problem

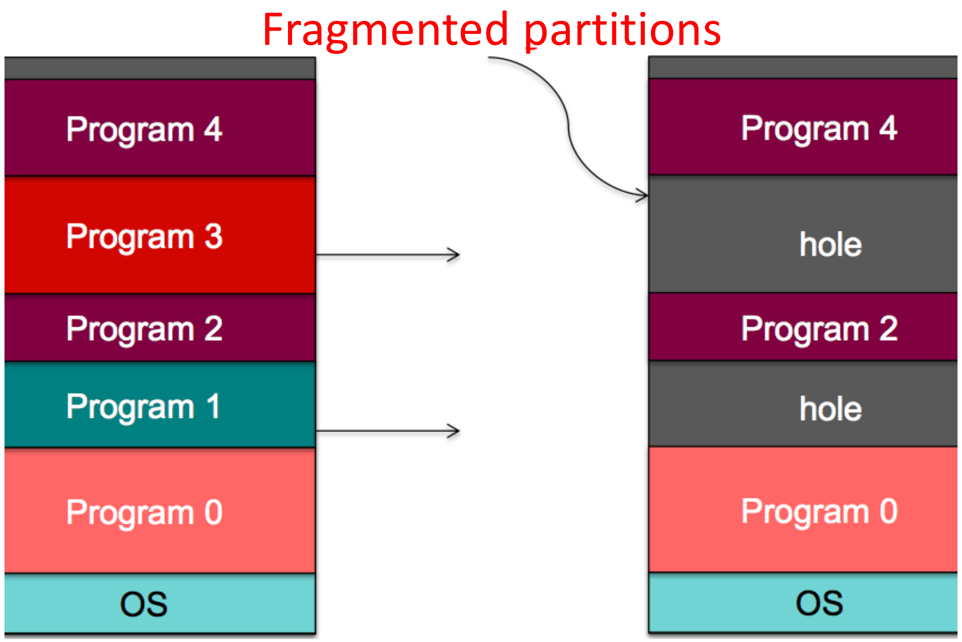
How to satisfy a request of size n from a list of free holes?

- **First-fit**: Allocate the **first** hole that is big enough.
- **Best-fit**: Allocate the **smallest** hole that is big enough; must search entire list, unless ordered by size.
  - Produces the smallest leftover hole.
- **Worst-fit**: Allocate the **largest** hole; must also search entire list.
  - Why? Produces the largest leftover hole.

First-fit and best-fit perform better than worst-fit in terms of speed and storage utilization, however it causes fragmentation.

Fragmentation

- **External Fragmentation**
  - total memory space exists to satisfy a request, but it is not contiguous
  - Also a common problem in disk as well



- **Internal Fragmentation**
  - Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

Need more memory?

- What if a process needs more memory?
  – Always allocate some extra memory just in case
  – Find a hole big enough to relocate the process

- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible **only** if relocation is dynamic, and is done at execution time.

Segmentation

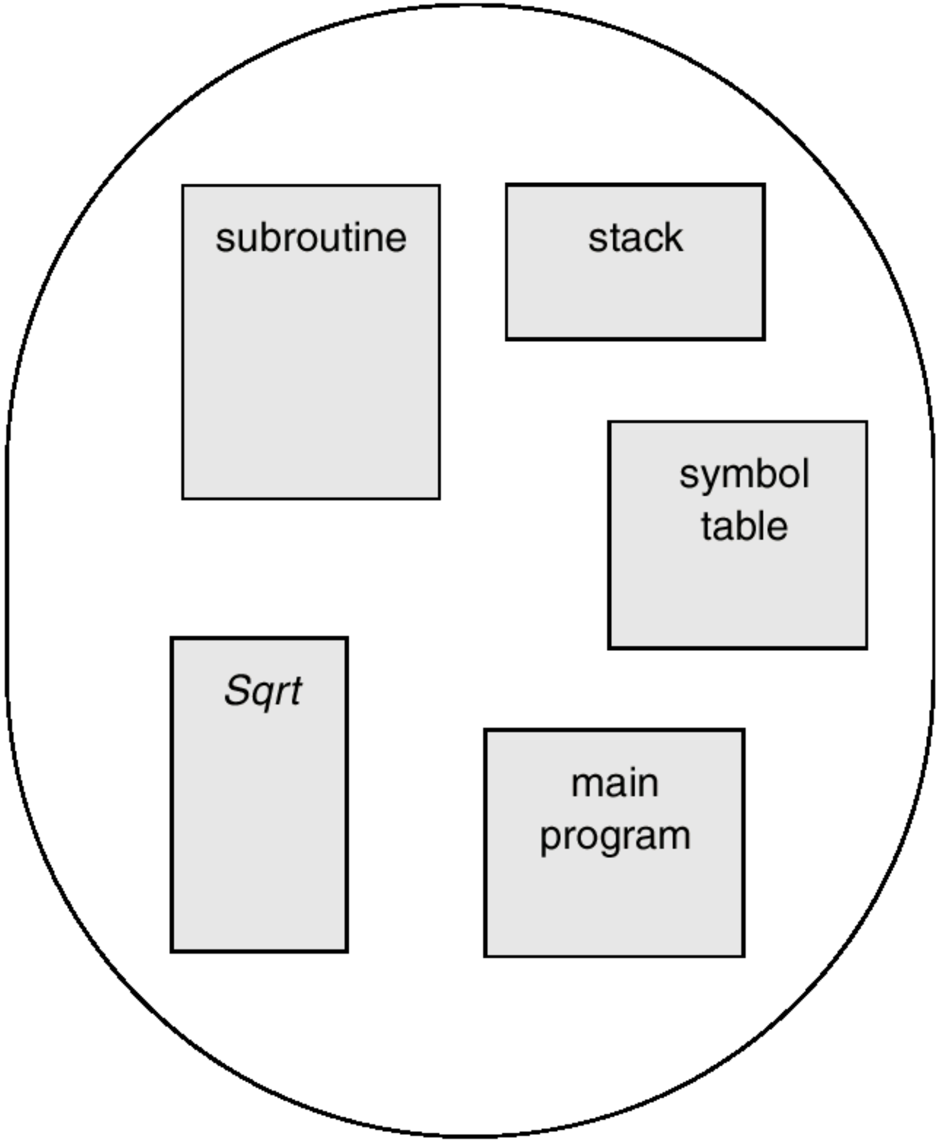
- Memory allocation mechanism that supports user view of memory.

- Users prefer to view memory as a collection of variable-sized segments – similar to programmer's view of memory

- A program is a collection of segments. A segment is a logical unit such as:

  main program,
  function,
  method,
  object,
  local variables, global variables,
  common block,
  stack,
  symbol table, arrays

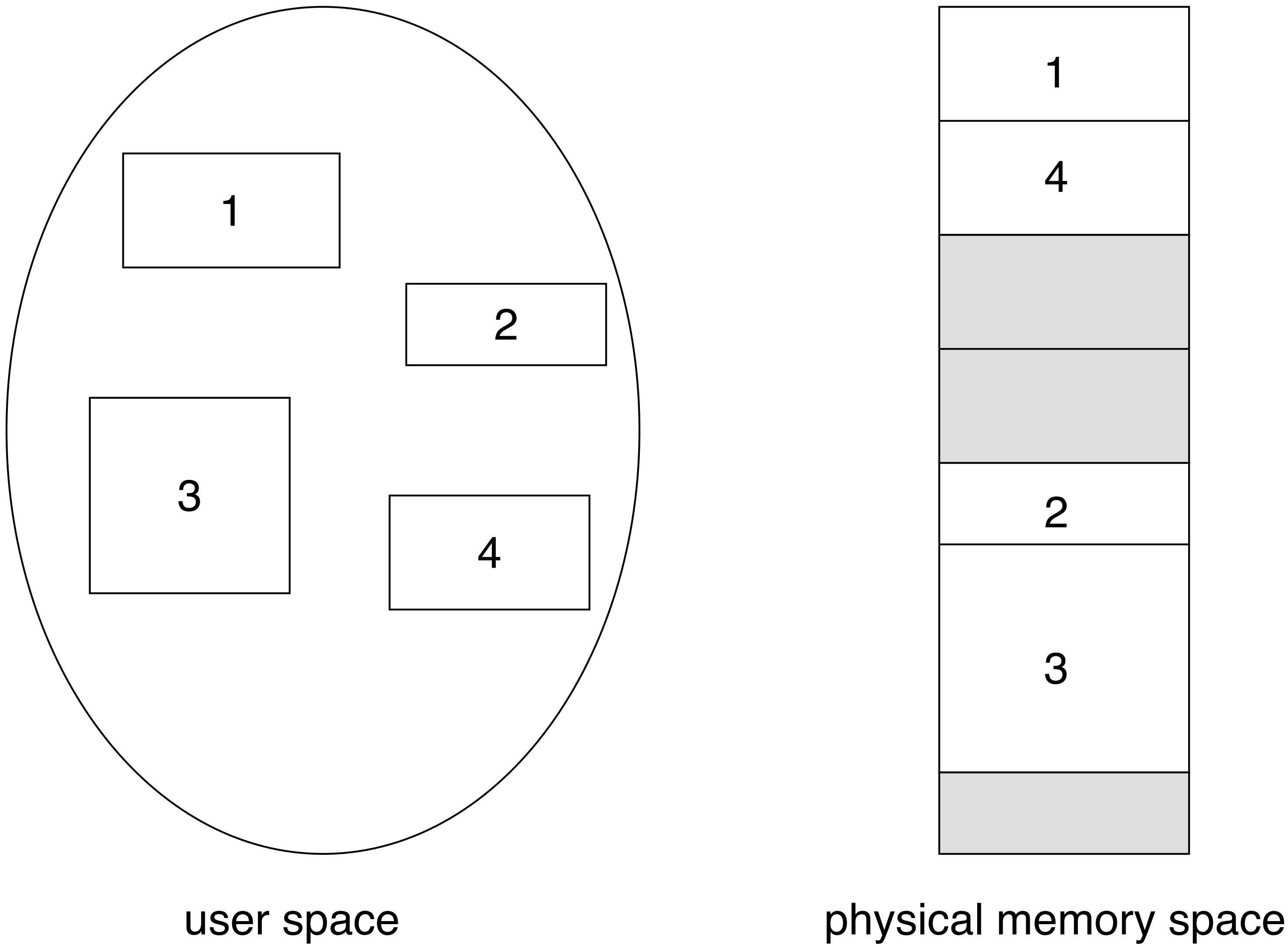
User's View of a Program



logical address space

Logical address space is a collection of segments

Logical View of Segmentation



Segmentation Architecture

- Logical address consists of a two tuple:

  `<segment-number, offset>`

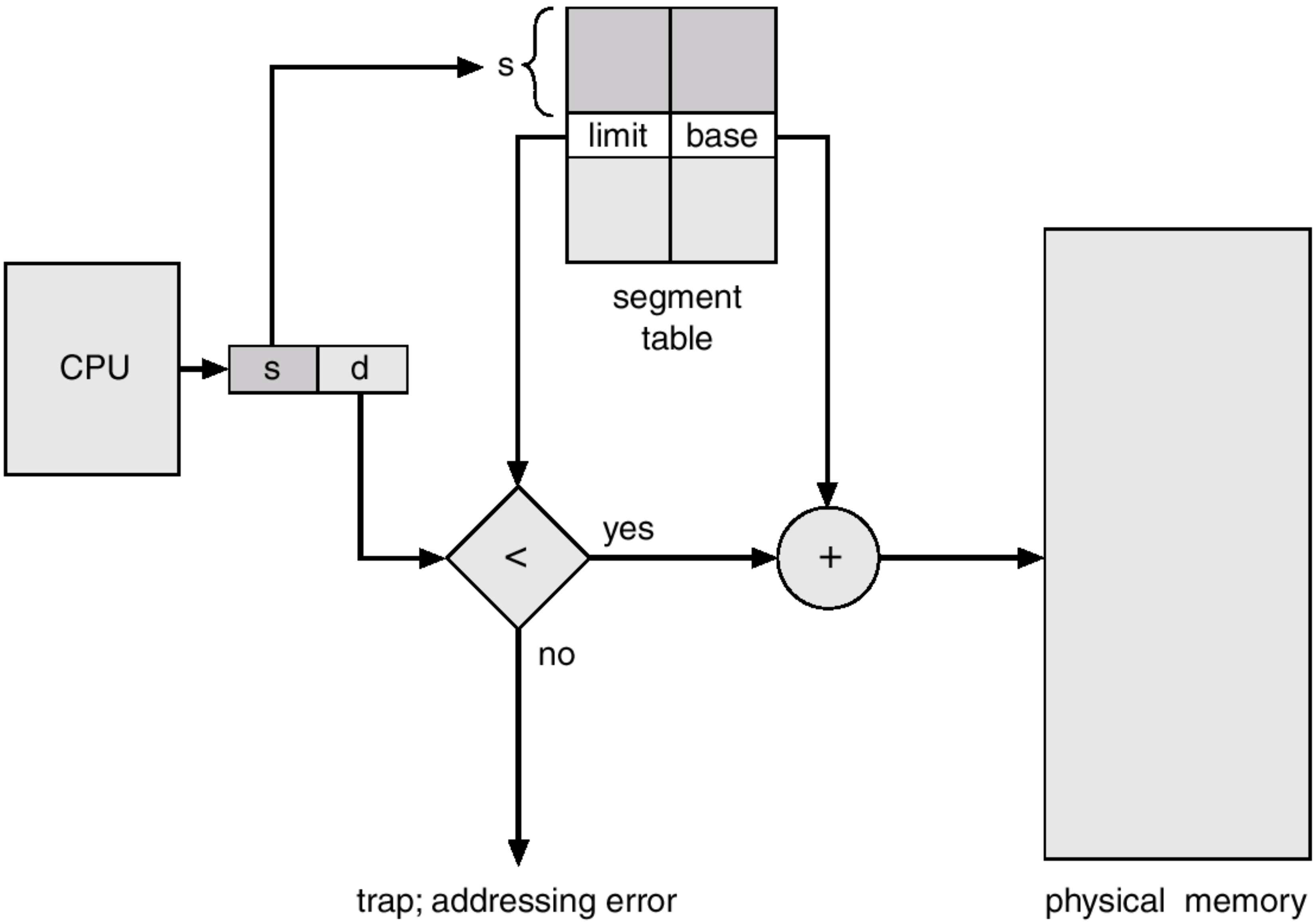
- Segment table: maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the **segments** reside in memory.
  - limit – specifies the length of the segment.
- Segment-table base register (STBR) points to the segment table's location in memory.
- Segment-table length register (STLR) indicates number of segments used by a program;

  segment number s is legal if s < STLR.

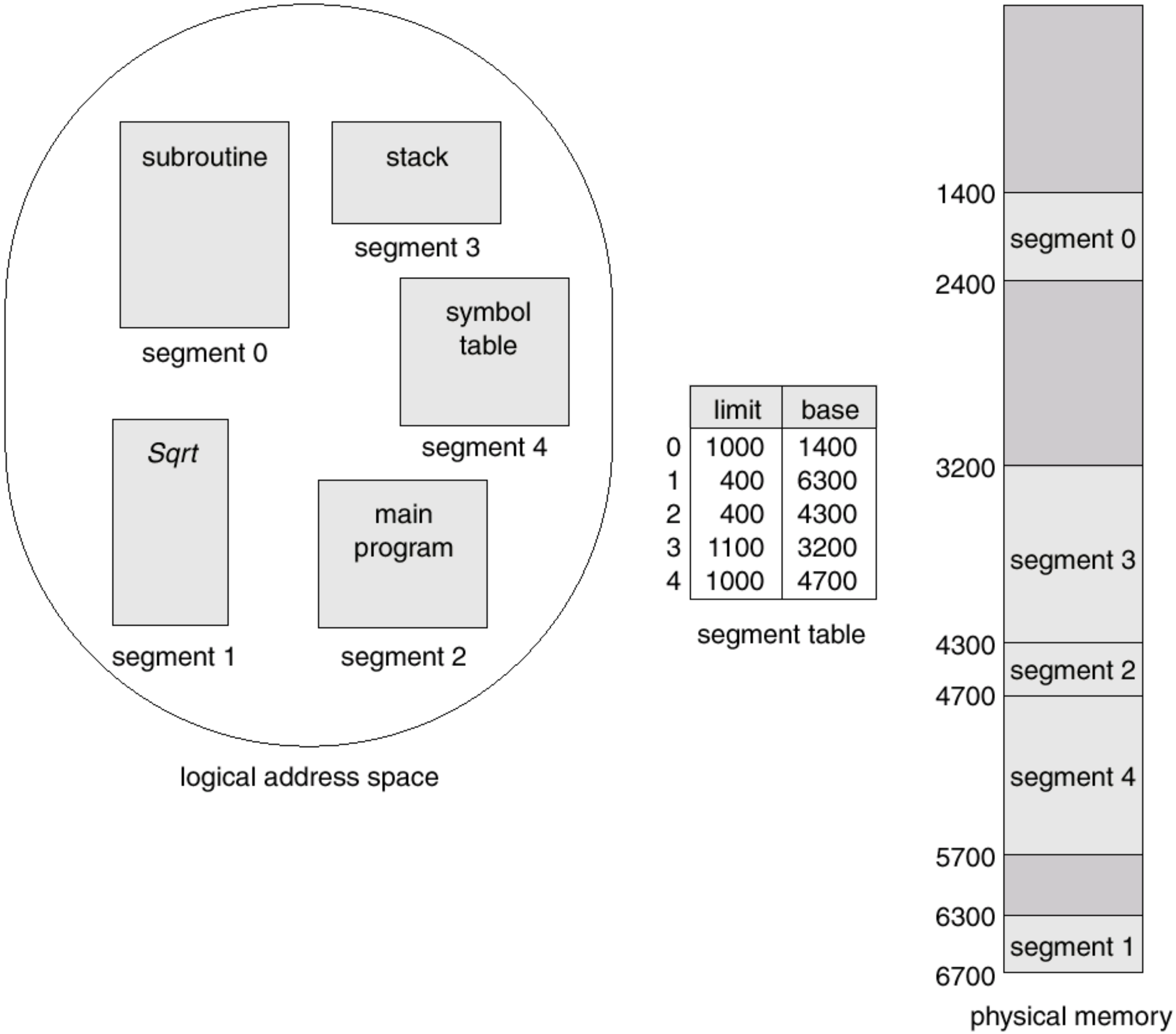
Segmentation Architecture

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

Segmentation Hardware



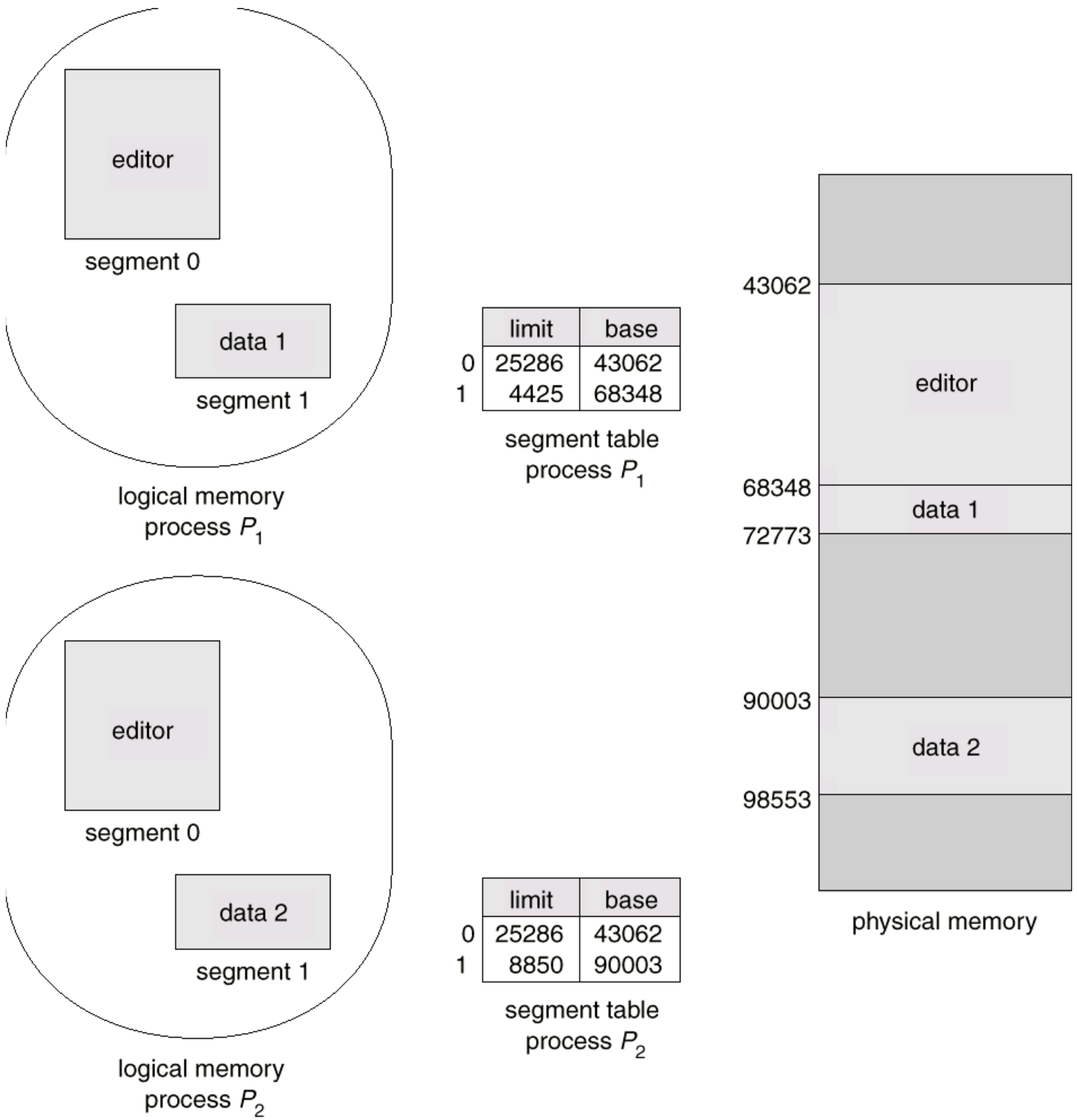
Example of Segmentation



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

segment table

Sharing of Segments



	limit	base
0	25286	43062
1	4425	68348

segment table
process P_1

	limit	base
0	25286	43062
1	8850	90003

segment table
process P_2

Acknowledgments

- These slides are adapted from
  - Öznur Özkasap (Koç University)
  - Operating System and Concepts (9th edition) Wiley
  - Paul Krzyzanowski (Rutgers University)