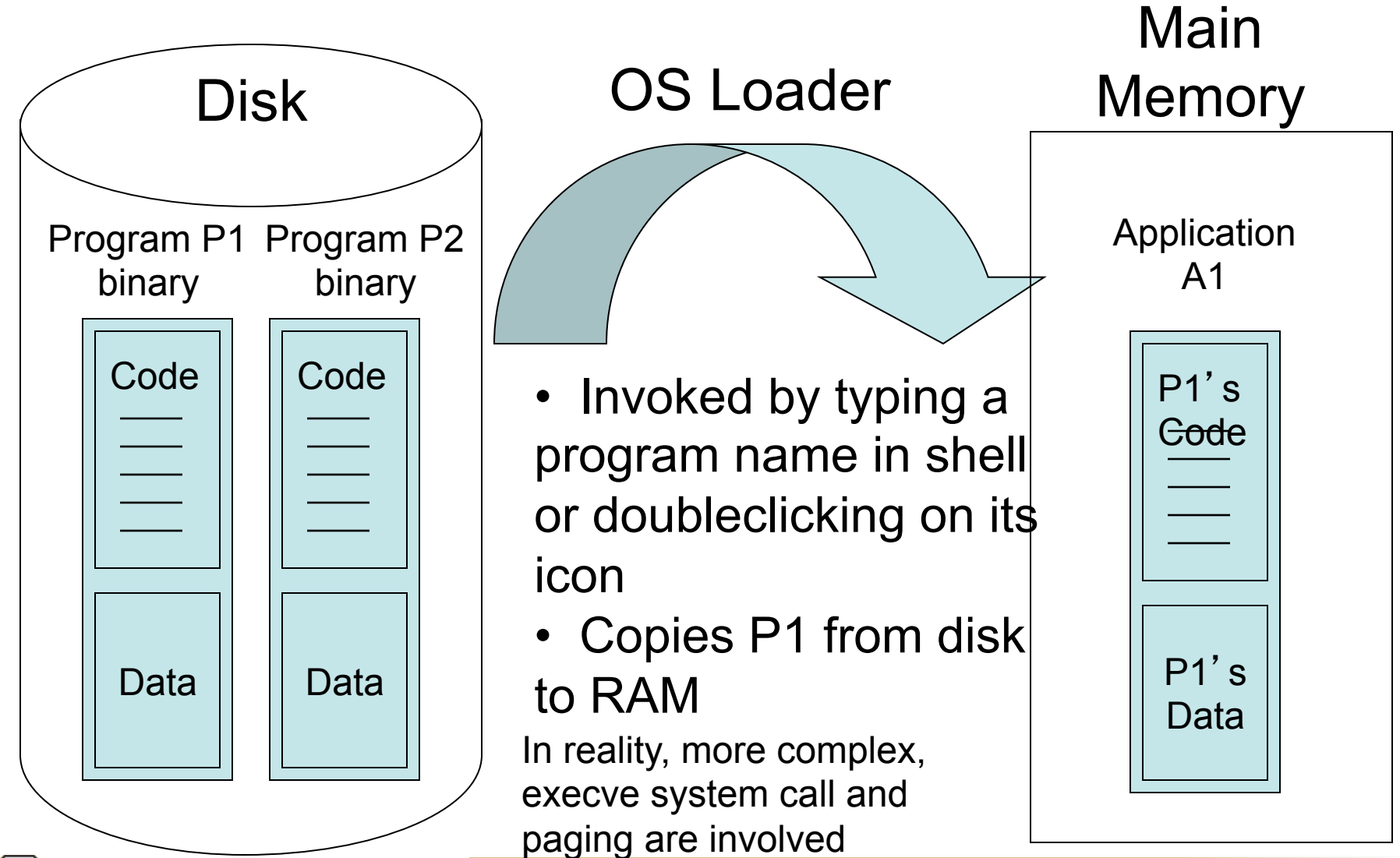
Chapter 3: Introduction to Processes

CSCI 3753 Operating Systems

Prof. Rick Han

University of Colorado at Boulder

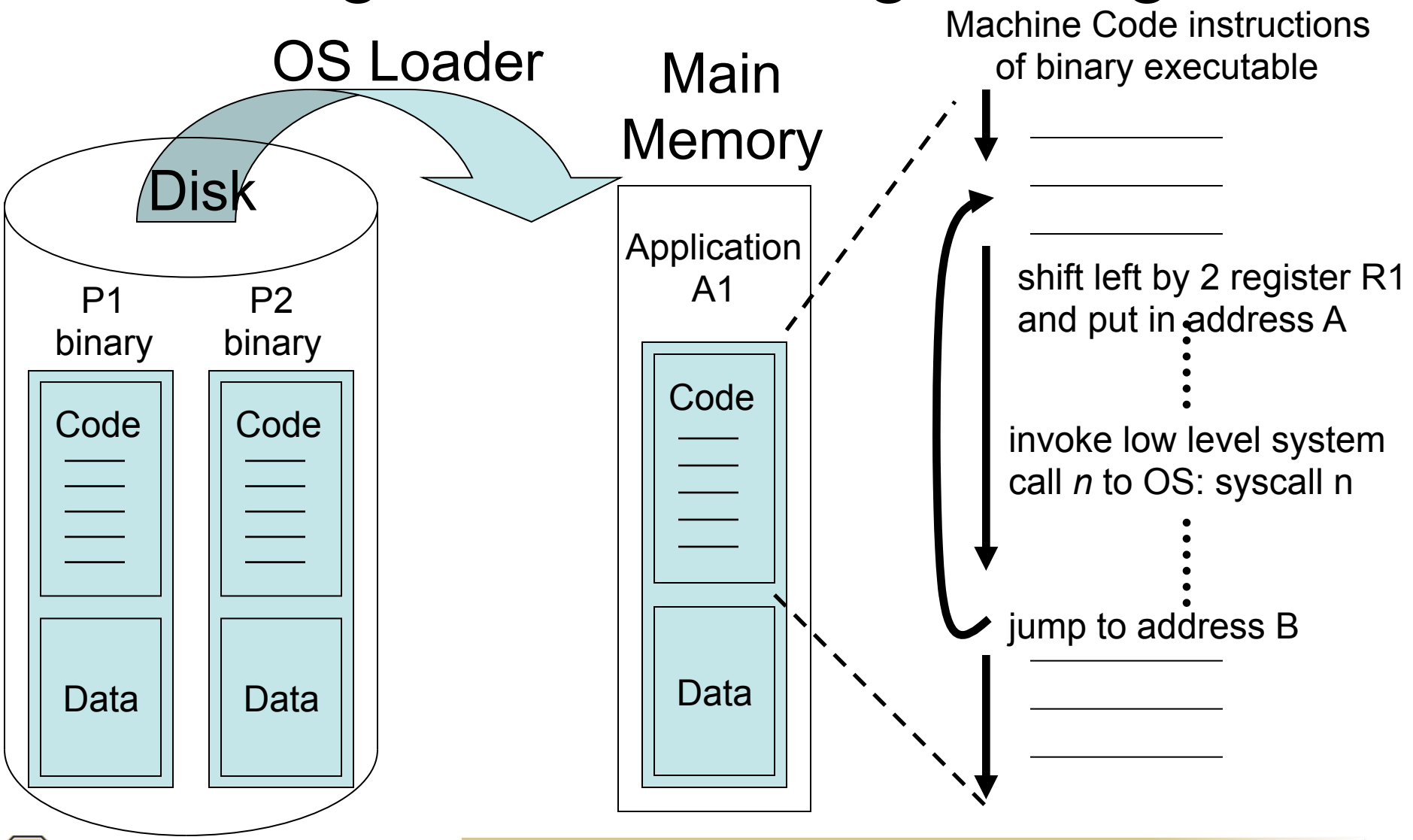
Loading a Program into Memory



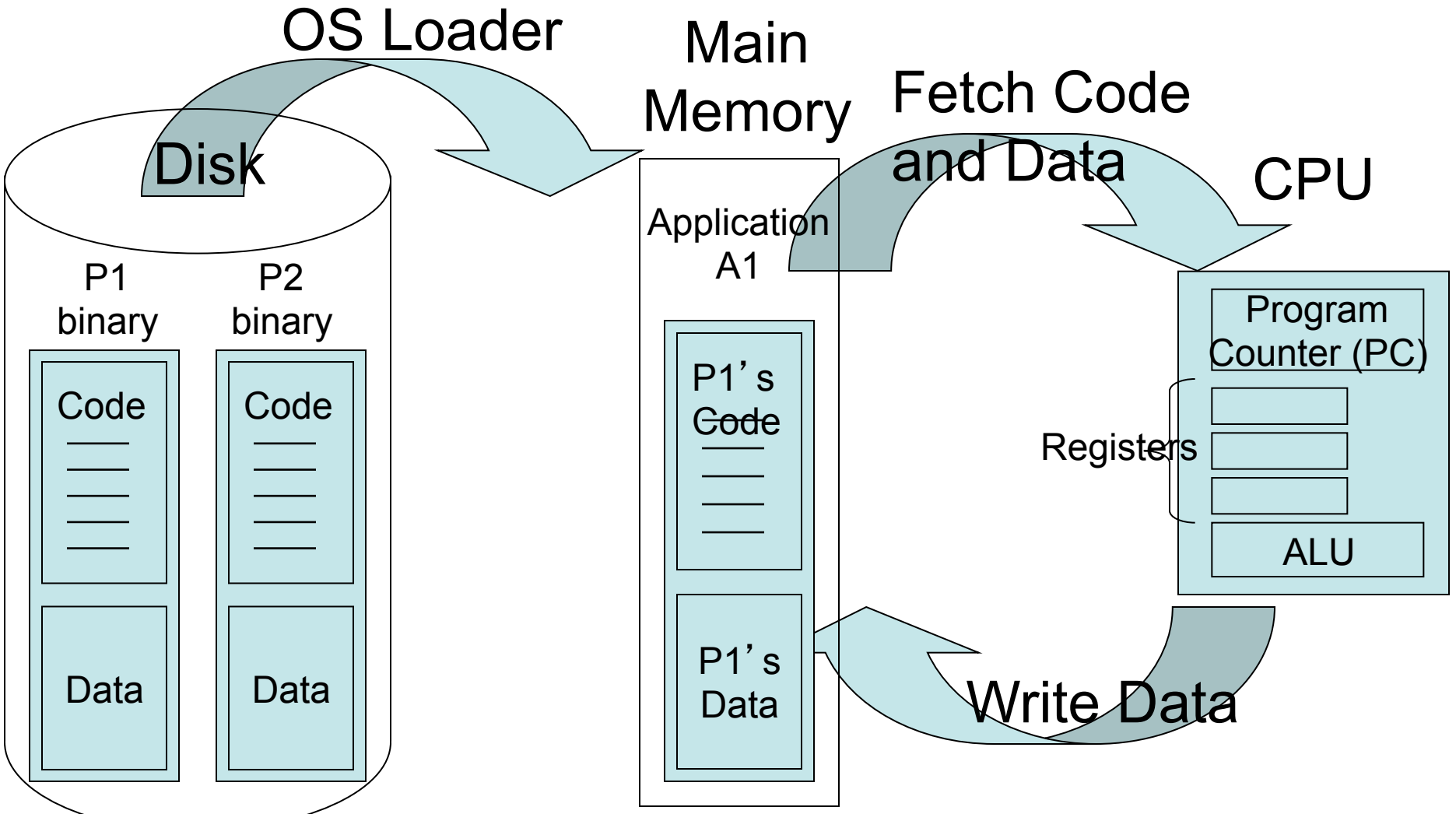
- Invoked by typing a program name in shell or doubleclicking on its icon
- Copies P1 from disk to RAM

In reality, more complex, execve system call and paging are involved

Loading and Executing a Program

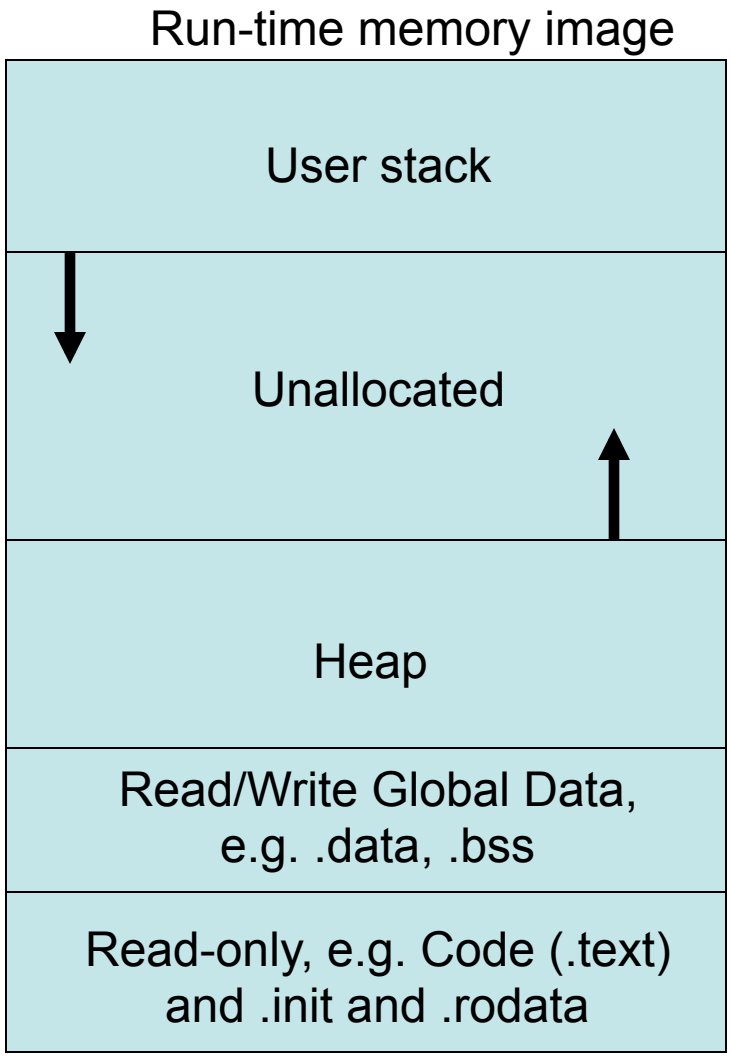


Loading and Executing a Program



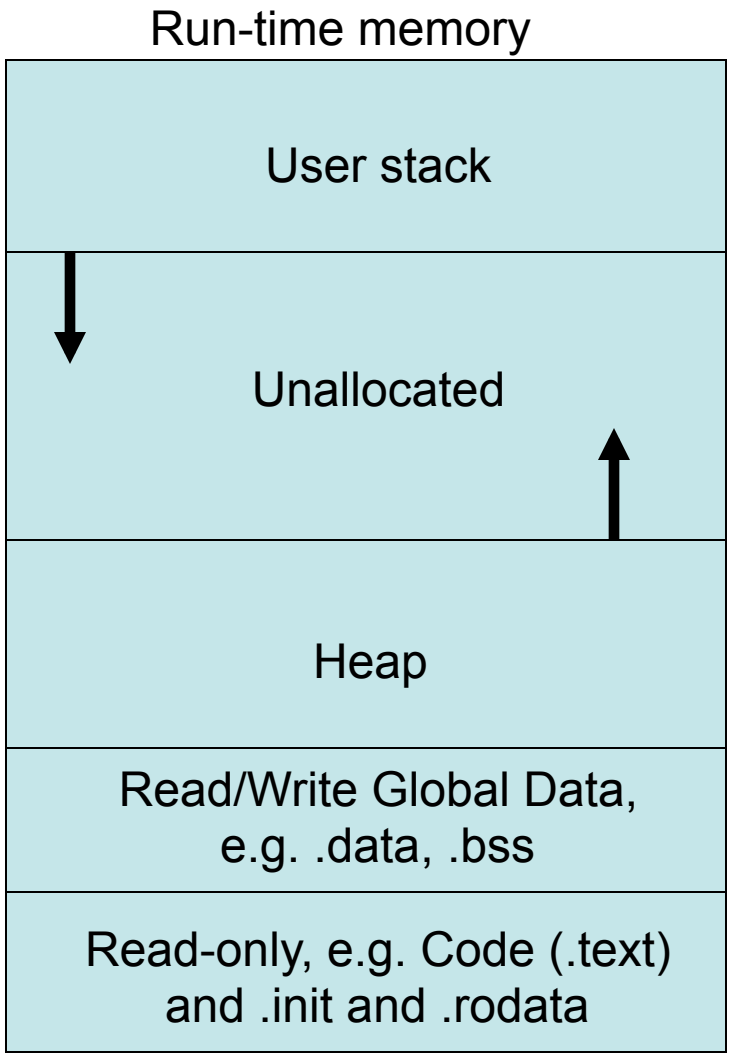
Loading Executable Object Files

- When a program is loaded into RAM, it becomes an actively executing application
- The OS allocates a stack and heap to the app in addition to code and global data.
  - A call stack is for local variables
  - A heap is for dynamic variables, e.g. malloc()
  - Usually, stack grows downward from high memory, heap grows upward from low memory , but this is architecture-specific

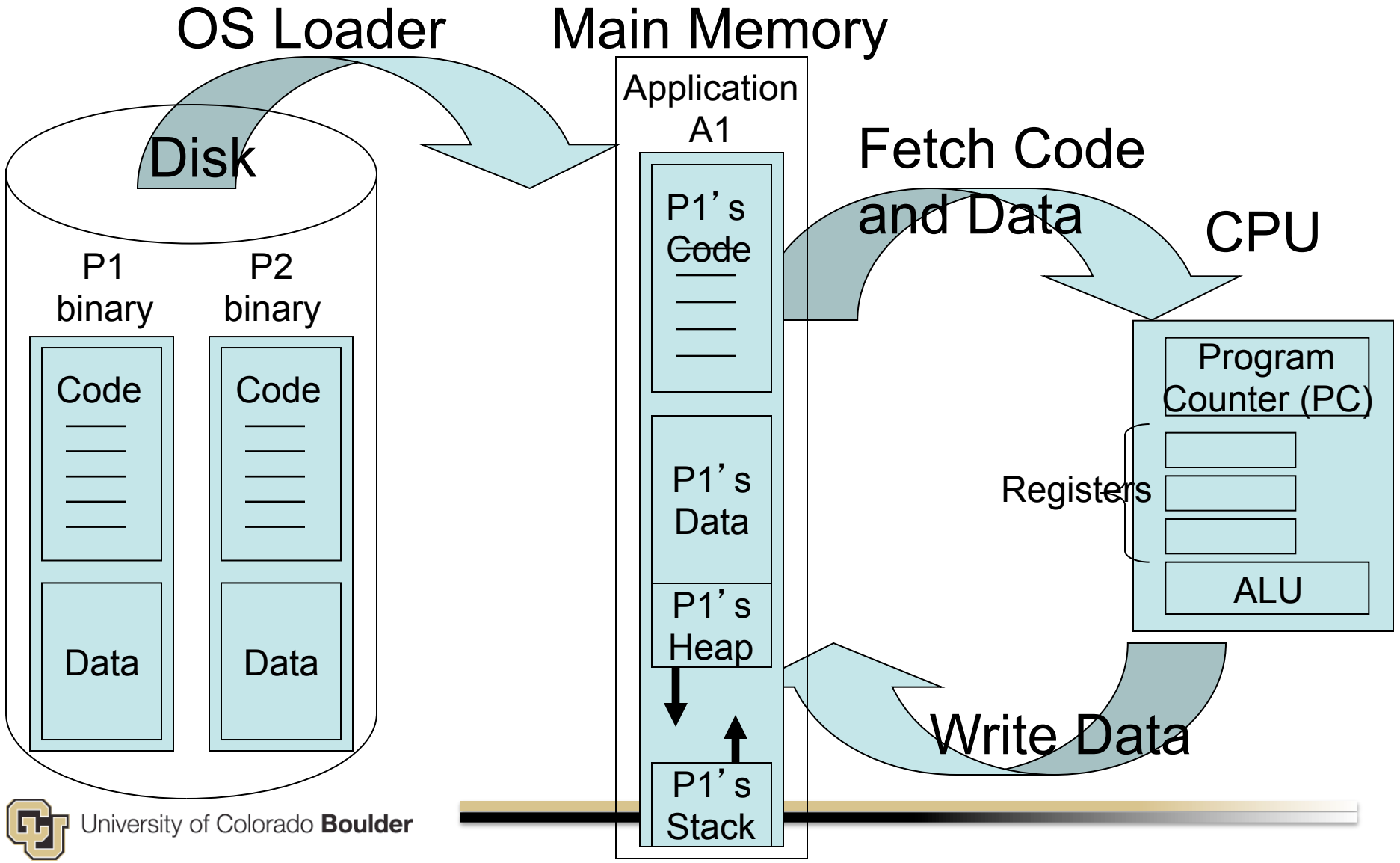


Running Executable Object Files

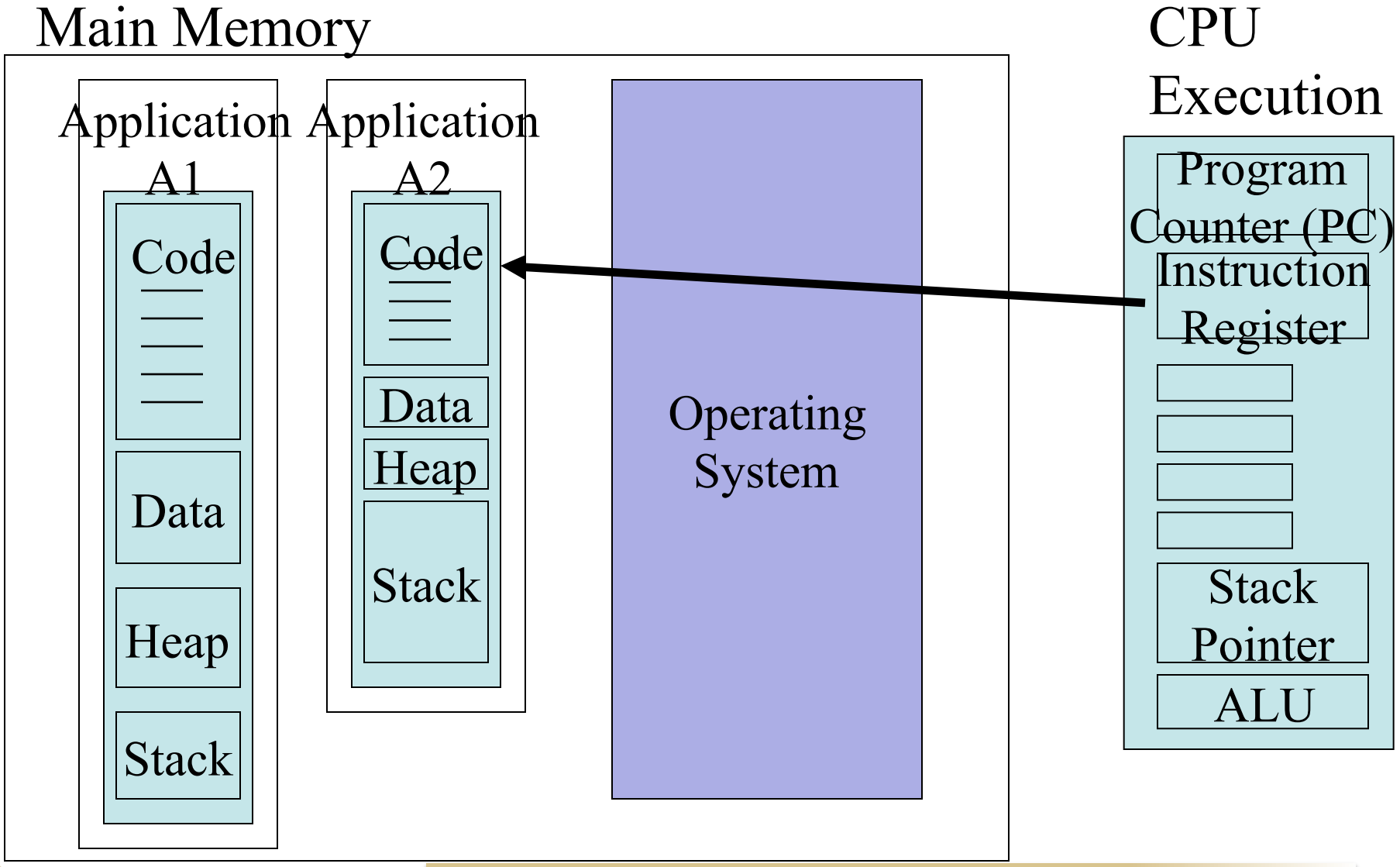
- Stack contains local variables
  - As main() calls function f1, we allocate f1's local variables on the stack
  - If f1 calls f2, we allocate f2's variables on the stack below f1's, thereby growing the stack, etc…
  - When f2 is done, we deallocate f2's local variables, popping them off the stack, and return to f1
- Stack dynamically expands and contracts as program runs and different levels of nested functions are called
- Heap contains run-time variables/ buffers
  - Obtained from malloc()
  - Program should free() the malloc'ed memory
- Heap can also expand and contract during program execution



Loading and Executing a Program – a more complete picture

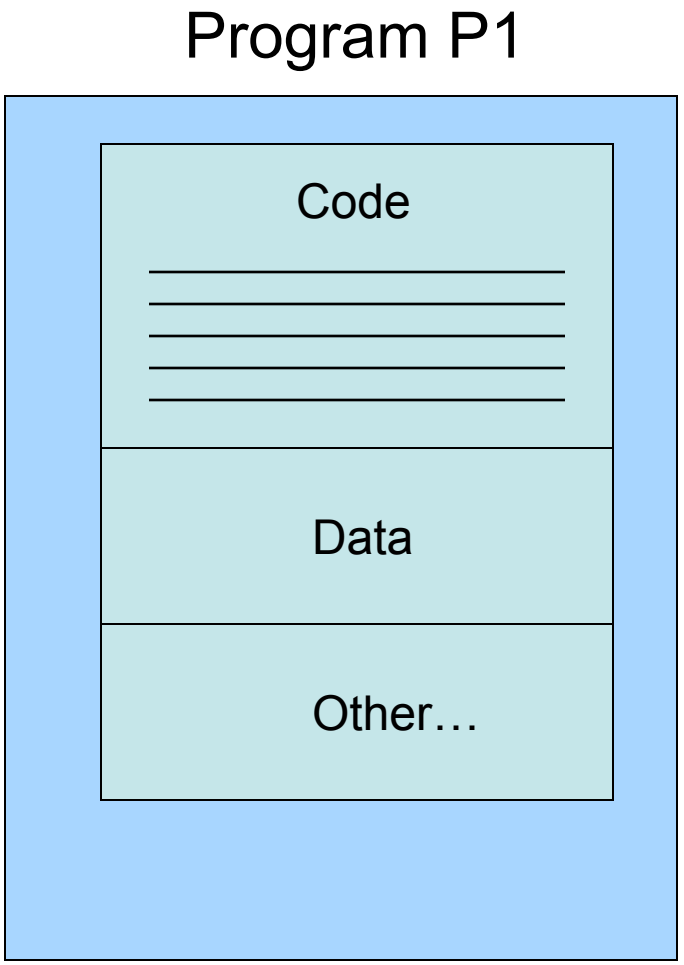


Multiple Applications + OS

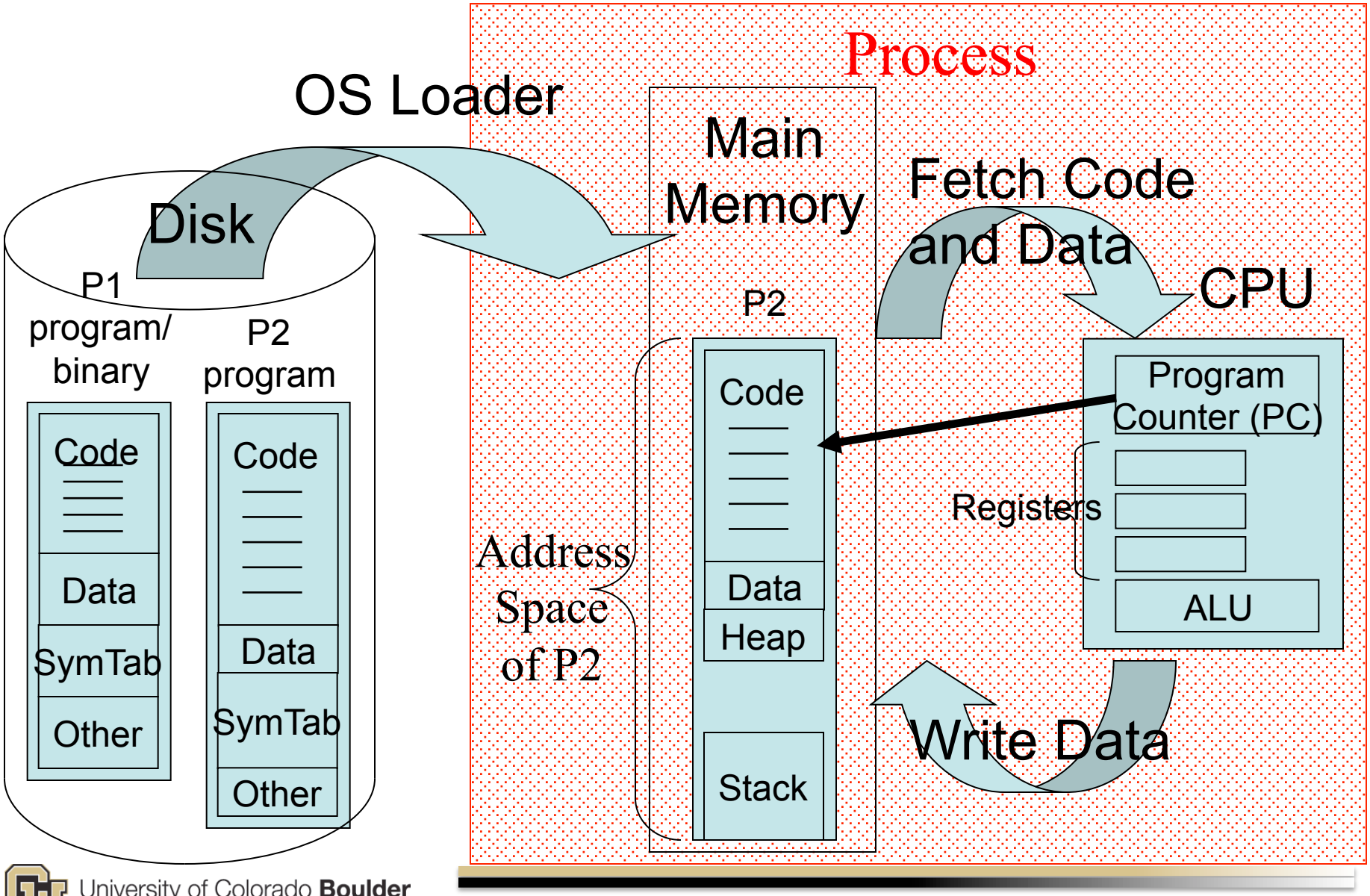


Chapter 3: What is a Process?

- A software *program* consist of a sequence of code instructions and data stored on disk
  - A program is a *passive* entity
- A *process* is a program *actively executing* from main memory within its *own address space*

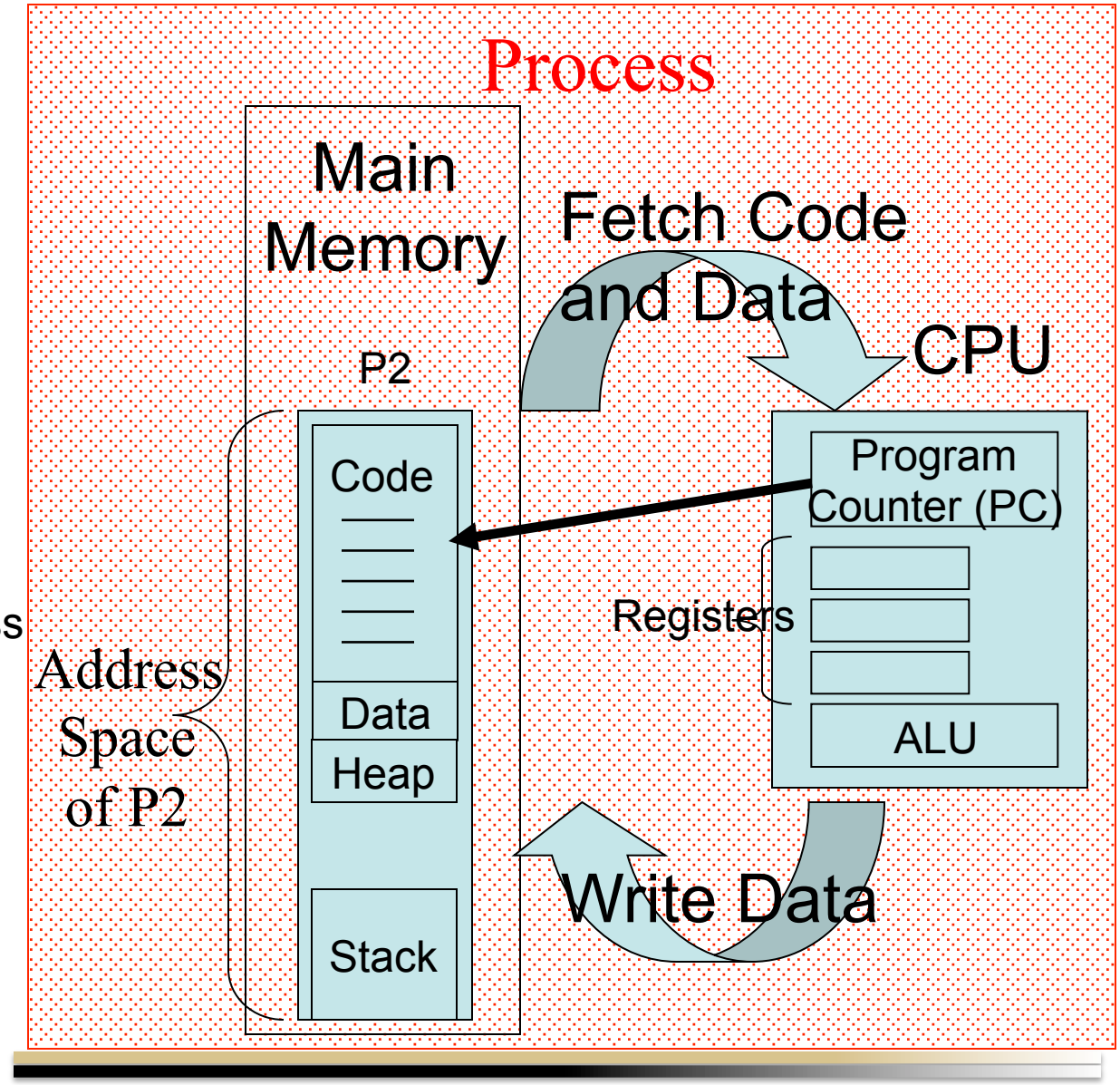


What Is a Process? (2)



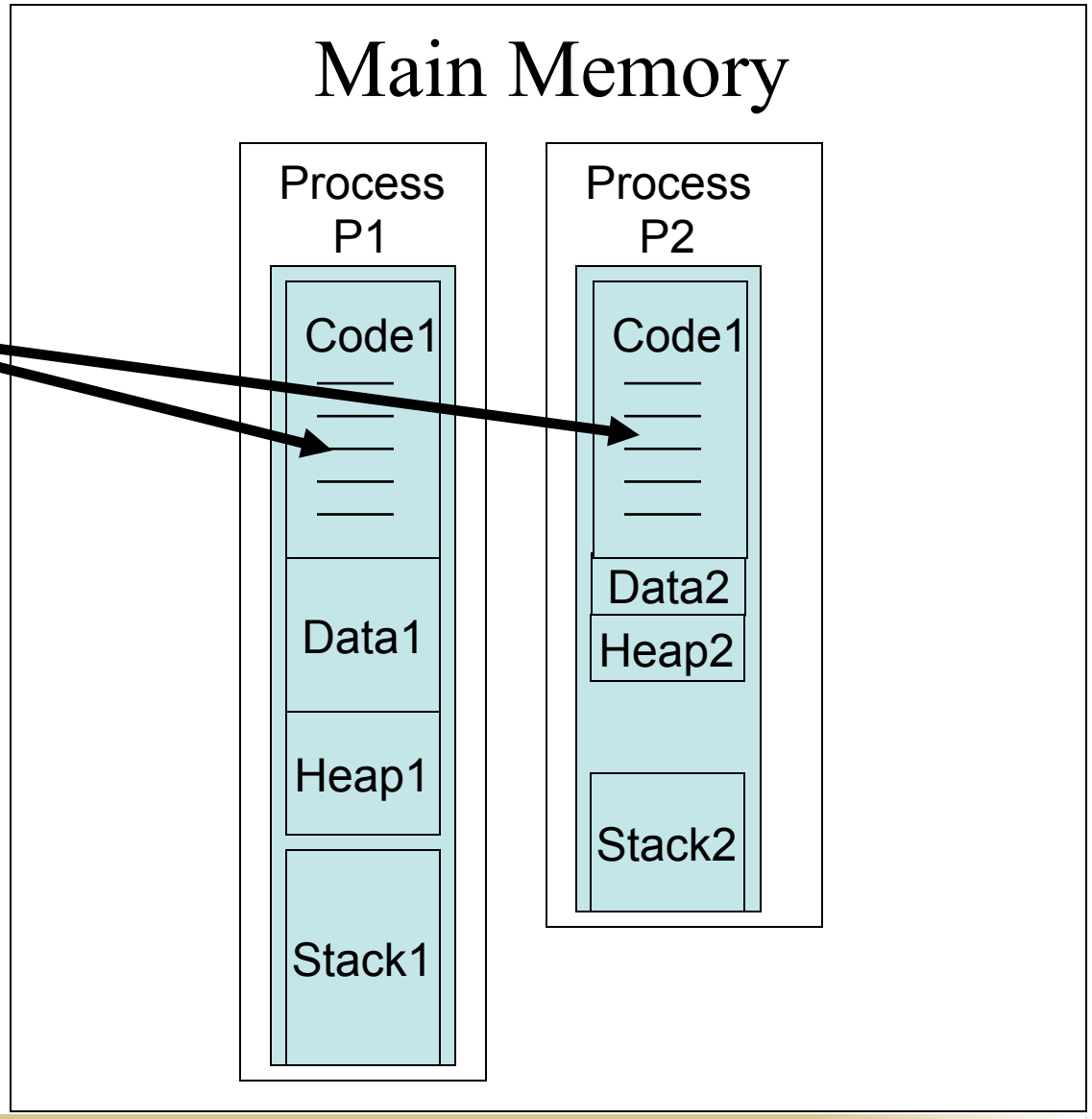
What is a Process? (3)

- A *process* is a program *actively executing* from main memory
  - has a Program Counter (PC) and execution state associated with it
    - CPU registers keep state
    - OS keeps process state in memory
    - it's alive!
  - Owns its own *address space*
    - a limited set of (virtual) addresses that can be accessed by the executing code

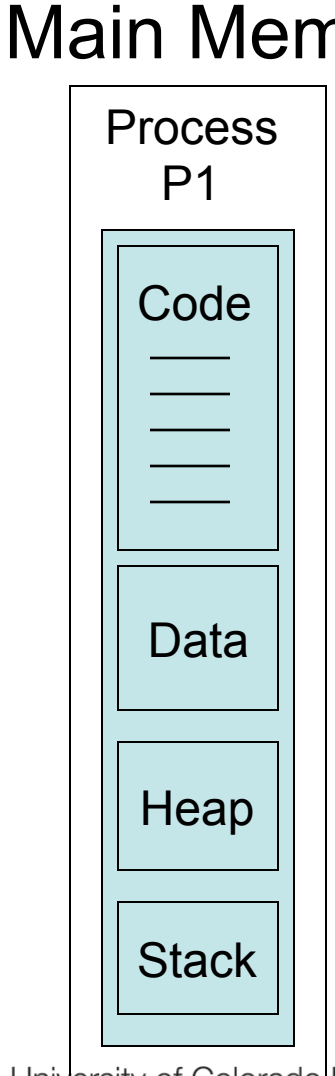


What is a Process? (4)

- 2 processes may execute the same program code, but they are considered *separate execution* sequences
  - e.g. two shell terminals

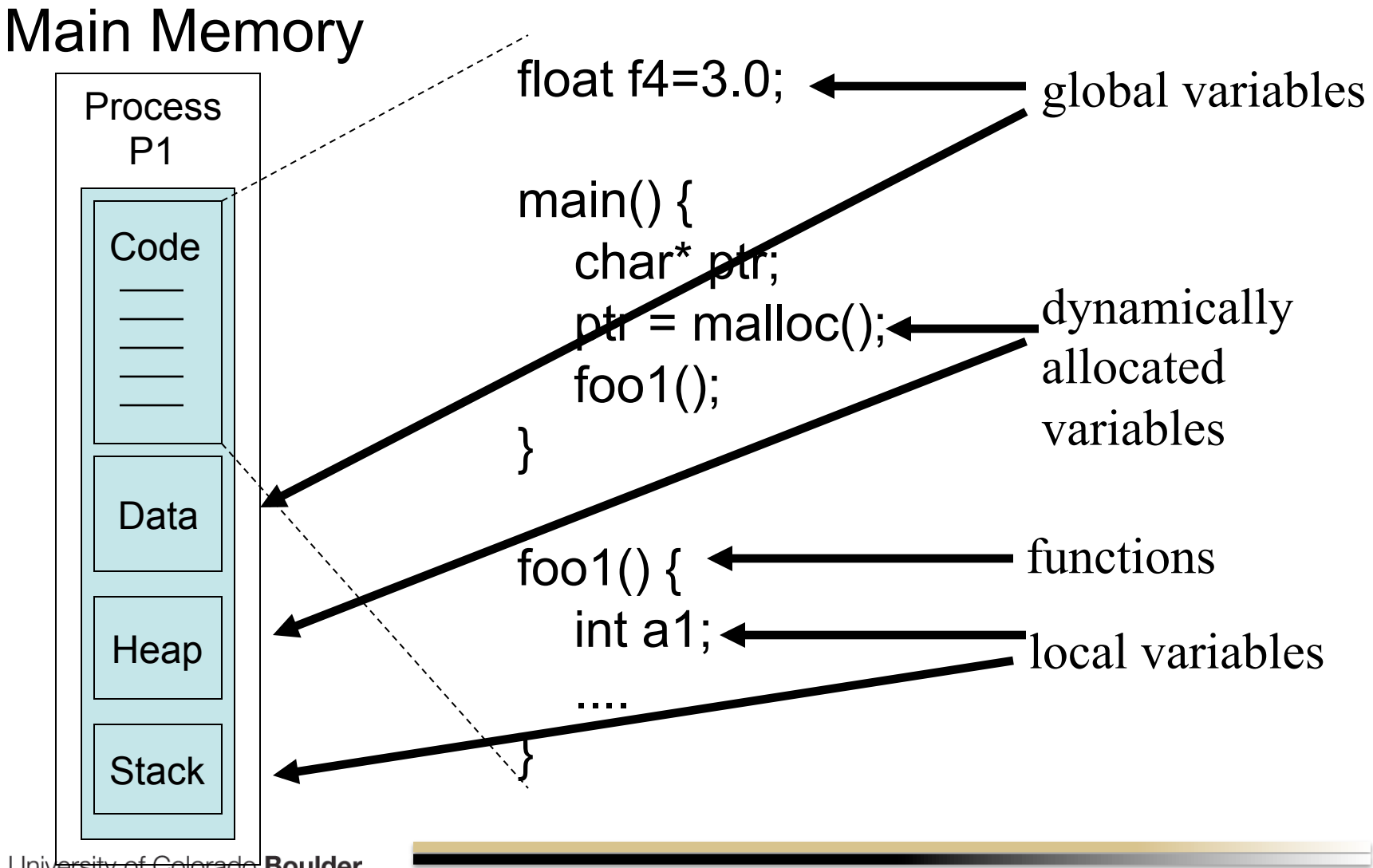


A Process Executes in its Own Address Space



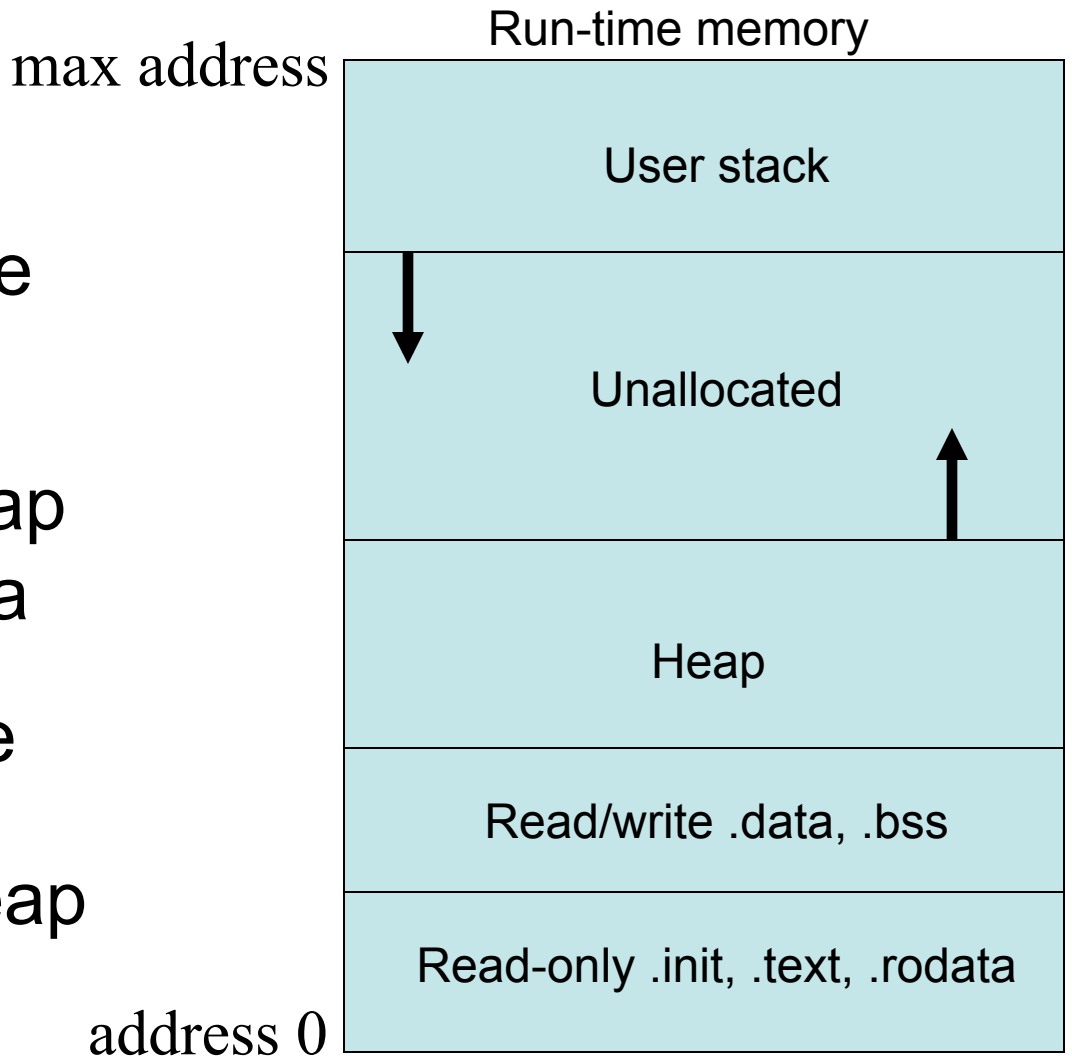
- OS tries to provide the illusion or *abstraction* to the process that it executes on its own abstract machine
  - in its own subset of RAM, i.e. its own address space – achieved using virtual memory paging
  - on its own subset (time slice) of the CPU – achieved by preemptive multitasking

How is a Process Structured in Memory?



How is a Process Structured in Memory?

- Run-time memory image
- Essentially code, data, stack, and heap
- Code and data loaded from executable file
- Stack grows downward, heap grows upward



Applications and Processes

- Application = $\sum$ Processes$_i$
  - e.g. a server could be split into multiple processes, each one dedicated to a specific task (UI, computation, communication, etc.)
  - The Application's various processes talk to each other using Inter-Process Communication (IPC). We'll see various forms of IPC later.