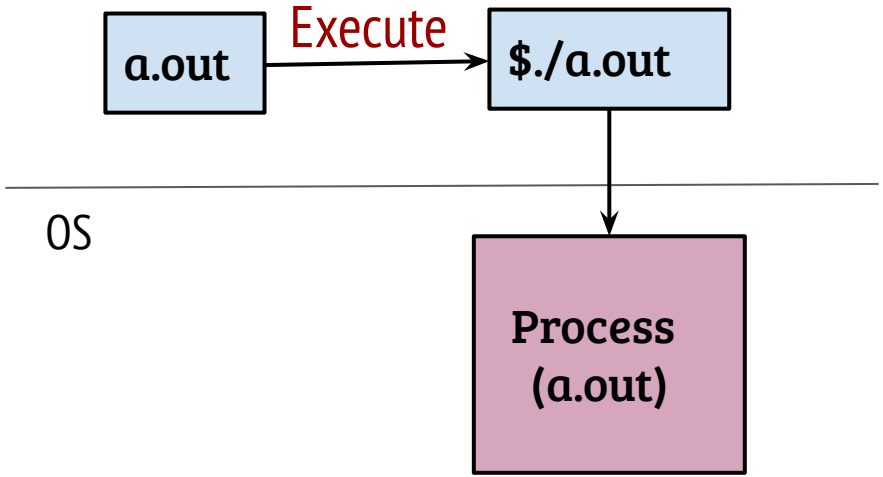
CS330: Operating Systems

Virtual memory: Address spaces

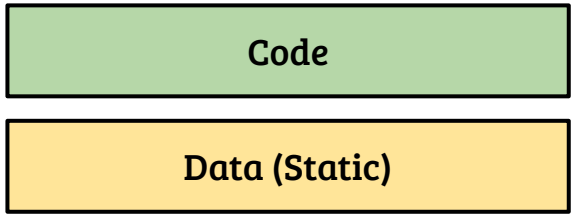
Recap: The process abstraction

- The OS creates a *process* when we run an *executable*



- Executable is a file, stored in a persistent storage (e.g., disk)
- To run, the process code and data should reside in memory
- Run-time memory allocation and deallocation should be supported

Executable file to process memory view



- A typical executable file contains code and statically allocated data
- Statically allocated: global and static variables
- Is loading the program (code and data) sufficient for program execution?

Executable file to process memory view

- A typical executable file contains code and statically allocated data
- Statically allocated: global and static variables
- Is loading the program (code and data) sufficient for program execution?
- No, we need memory for stack and dynamic allocation

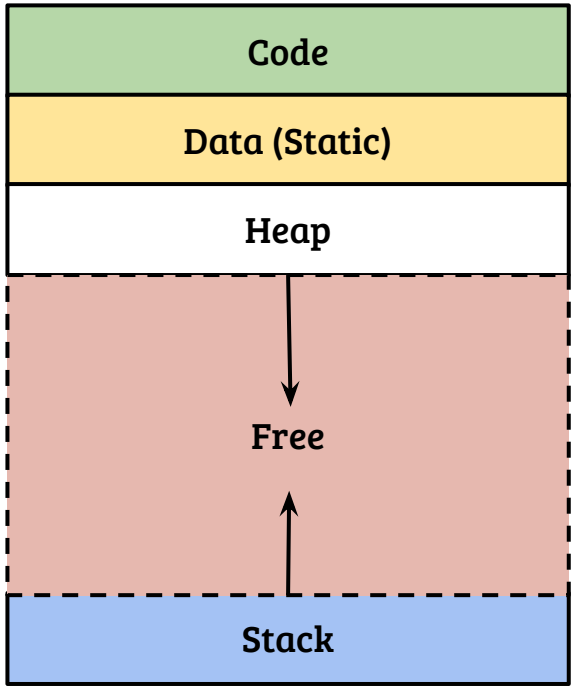
Executable file to process memory view

- A typical executable file contains code and statically allocated data
- Statically allocated: global and static variables
- Is loading the program (code and data) sufficient for program execution?
- No, we need memory for stack and dynamic allocation
- Stack: function call and return, store local (stack) variables

Executable file to process memory view

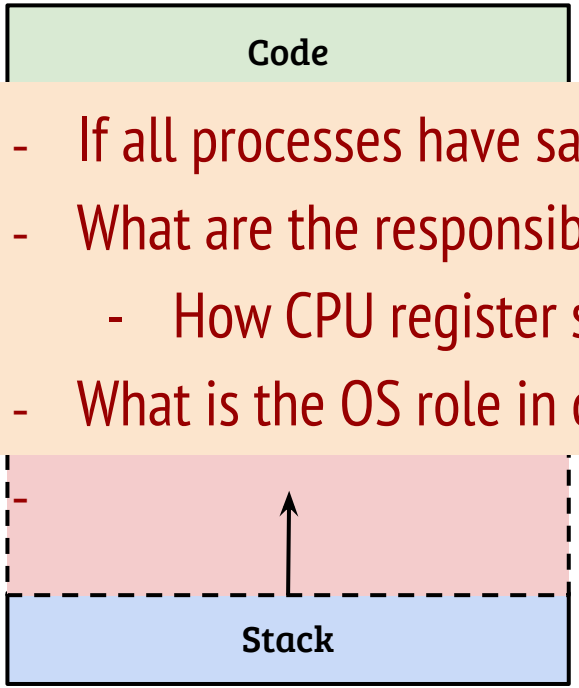
- A typical executable file contains code and statically allocated data
- Statically allocated: global and static variables
- Is loading the program (code and data) sufficient for program execution?
- No, we need memory for stack and dynamic allocation
- Stack: function call and return, store local (stack) variables
- Heap: dynamic memory allocation through APIs like *malloc( )*

The address space abstraction



- Address space represents memory state of a process
- Address space layout is same for all the processes (convenience)
- Exact layout can be decided by the OS, conventional layout is shown

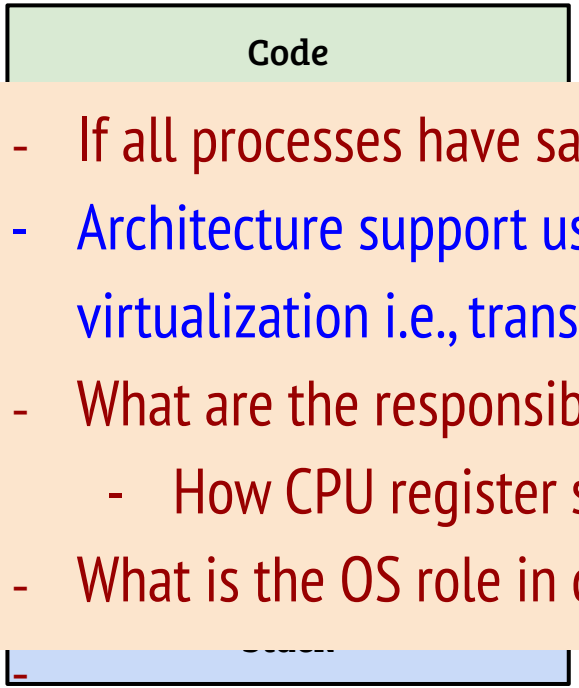
The address space abstraction



- If all processes have same address space, how they map to actual memory?
- What are the responsibilities of the OS during program load?
  - How CPU register state is changed?
- What is the OS role in dynamic memory allocation?

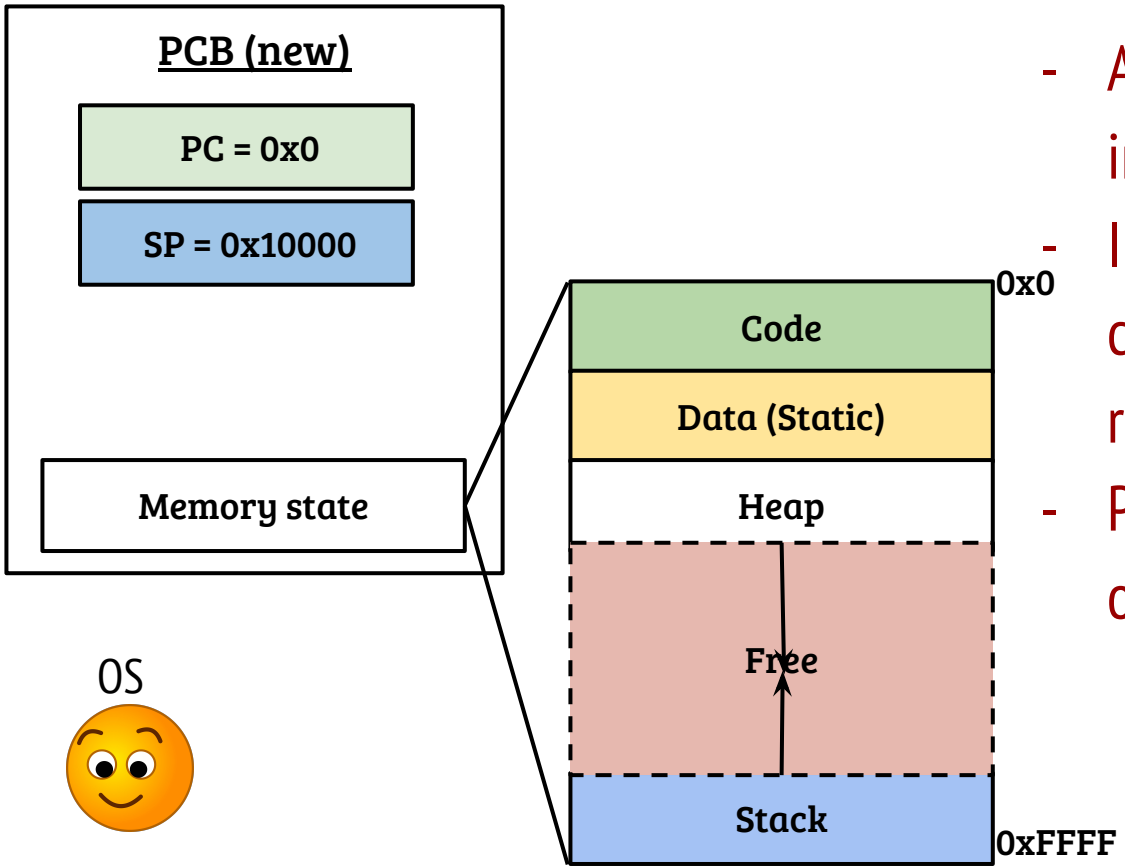
conventional layout is shown

The address space abstraction



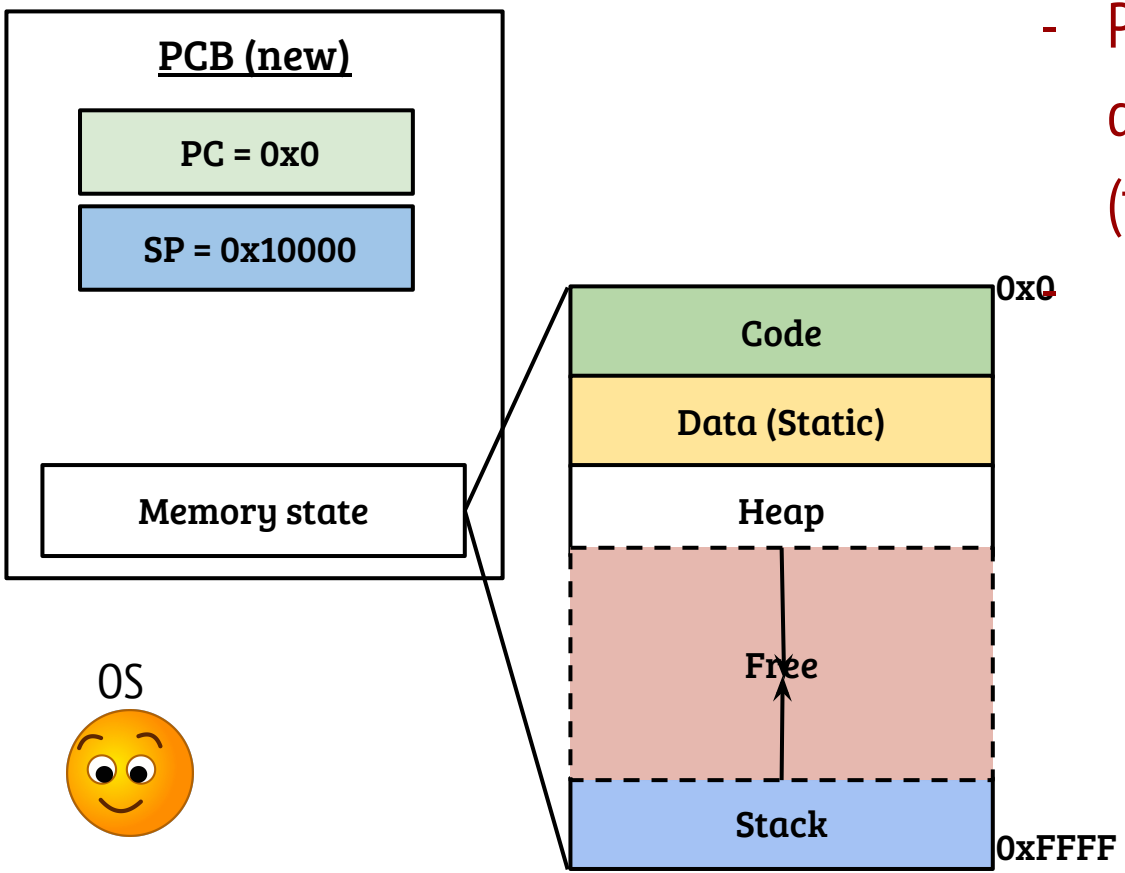
- If all processes have same address space, how they map to actual memory?
- Architecture support used by OS techniques to perform memory virtualization i.e., translate virtual address to physical address (will revisit)
- What are the responsibilities of the OS during program load?
  - How CPU register state is changed?
- What is the OS role in dynamic memory allocation?

OS during program load (exec)

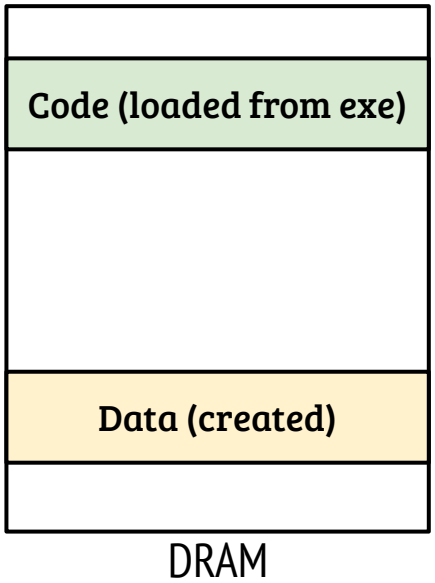


- A fresh address space is initialized
- In reality, parent address space copied at the time of fork( ) is reset
- PC and SP are set with addresses of code and stack, respectively

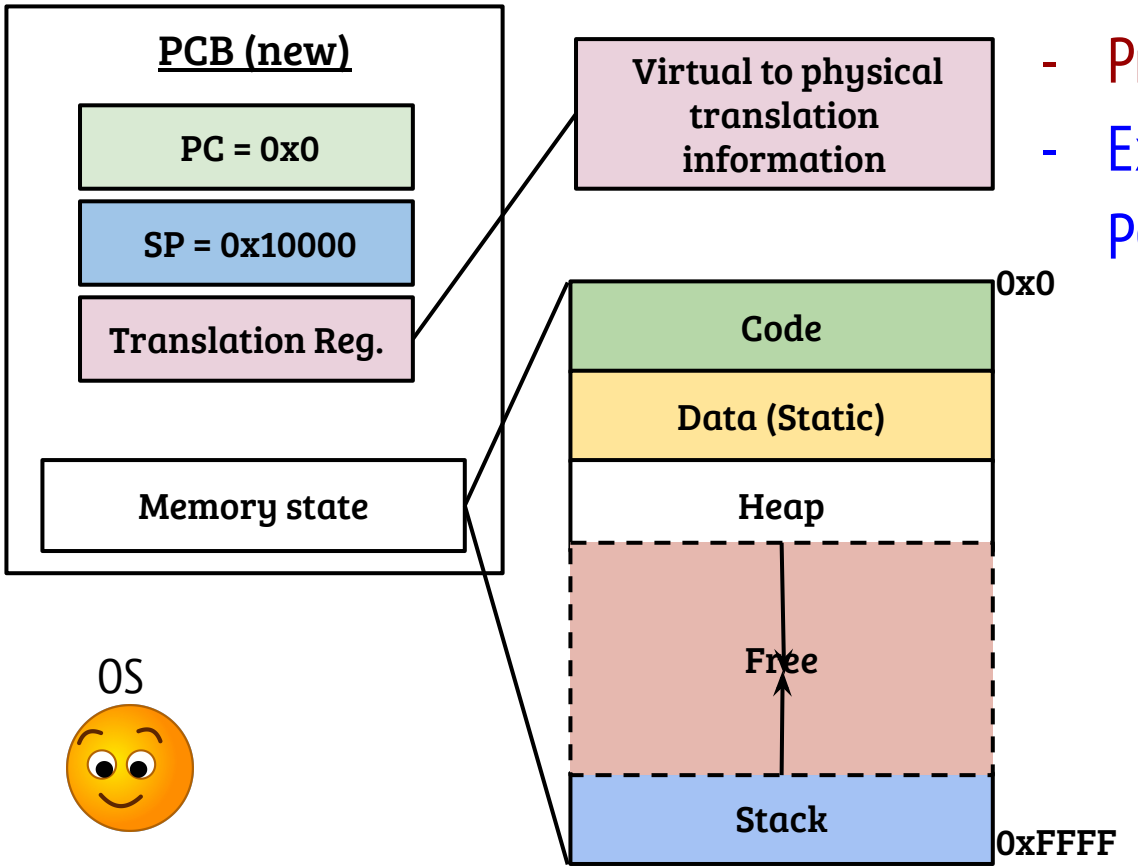
OS during program load (exec)



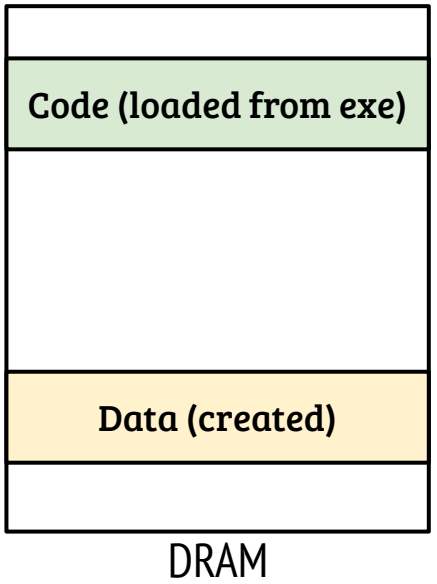
- Physical memory for code and data allocated, executable code (text section) is loaded



OS during program load (exec)



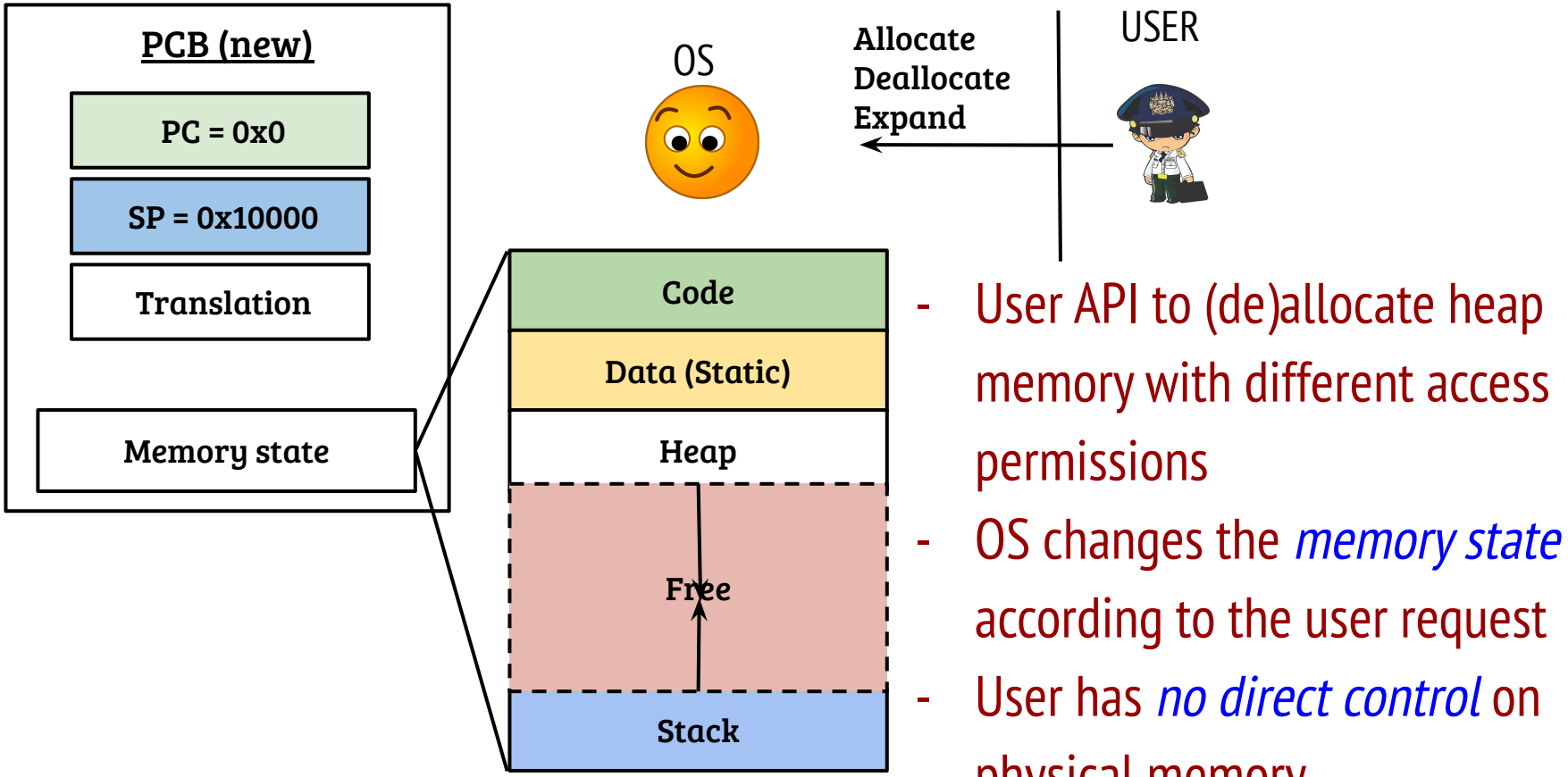
- Translation information updated
- Process is ready to execute
- Executes when register state in PCB is loaded onto the CPU



The address space abstraction

- If all processes have same address space, how they map to actual memory?
- Architecture support used by OS techniques to perform memory virtualization i.e., translate virtual address to physical address (will revisit)
- What are the responsibilities of the OS during program load?
  - How CPU register state is changed?
- Creating address space, loading binary, updating the PCB register state
- What is the OS role in dynamic memory allocation?

User API for memory management



- User API to (de)allocate heap memory with different access permissions
- OS changes the *memory state* according to the user request
- User has *no direct control* on physical memory