CS 310 Operating Systems

Lecture 3: Operating System Concepts – Part 4

Ravi Mittal

IIT Goa

Acknowledgements !

- Contents of this class presentation has been taken from various sources. Thanks are due to the original content creators:
  - Class presentation: University of California, Berkeley: David Culler, Anthony D. Joseph, John Kubiatowicz, AJ Shankar, George Necula, Alex Aiken, Eric Brewer, Ras Bodik, Ion Stoica, Doug Tygar, and David Wagner.

Use the following for your learning..

- Operating Systems: Principles and Practice (2nd Edition) Anderson and Dahlin
  - Volume 1, Kernel and Processes
    - Chapter 2
- https://inst.eecs.berkeley.edu/~cs162/su21/
  - CS 162, UCB
- https://youtu.be/itfEcA3TXq4

We will study..

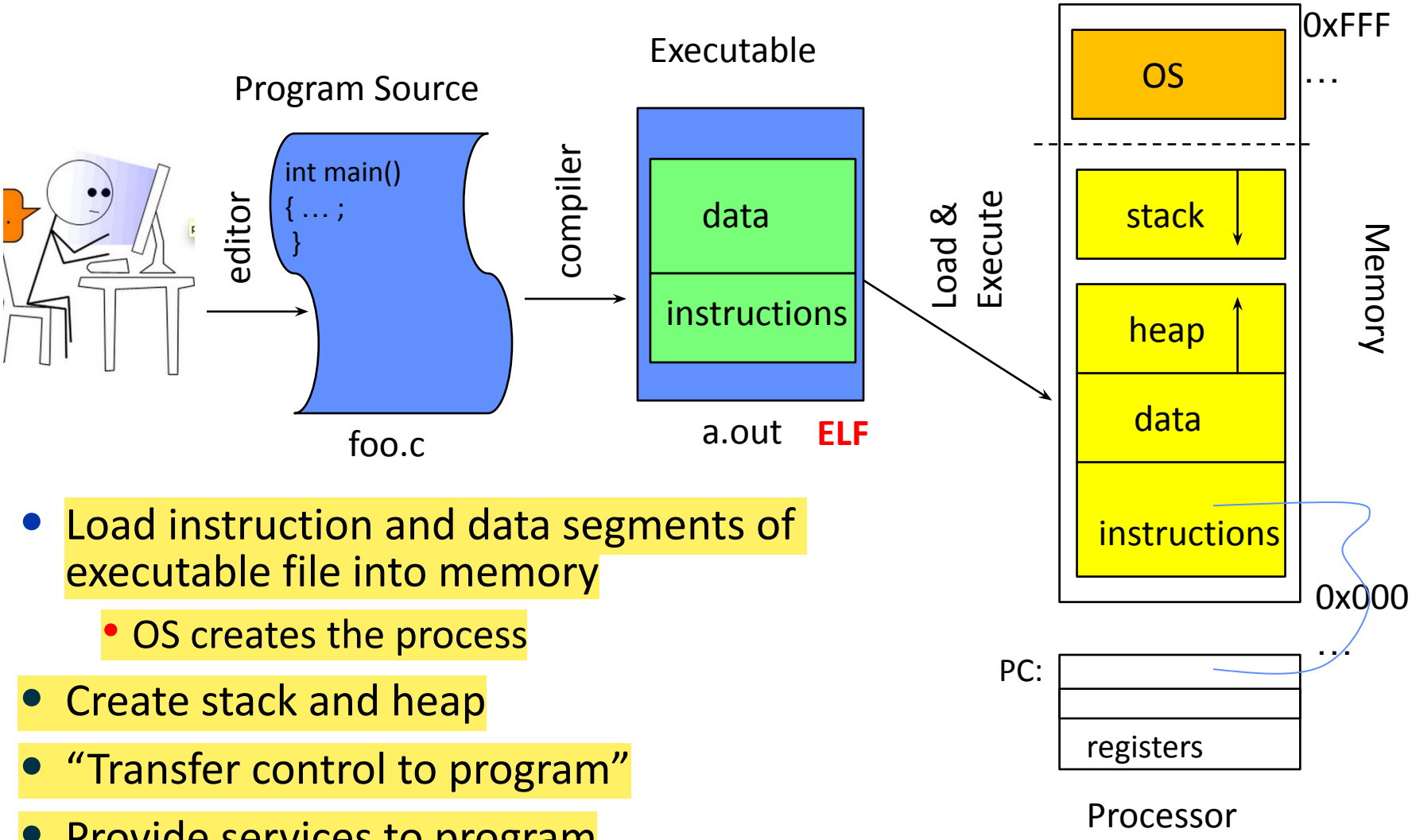
- Revision of concepts learnt so far in the course
- The third OS Concept: Thread
- The Fourth OS Concept: Dual Mode of Operation

Concepts we learnt so far

Four Fundamental OS Concepts

- **Process: an instance of a running program**
  - Protected Address Space + One or more Threads
- **Address space**
  - Set of memory addresses accessible to program (for read or write)
- **Thread: Execution Context**
  - Fully describes program state
- **Dual mode operation / Protection**
  - User / Kernel mode

OS Bottom Line: Run Programs



- Load instruction and data segments of executable file into memory
  - OS creates the process
- Create stack and heap
- "Transfer control to program"
- Provide services to program
- While protecting OS and program

ELF: Executable and Linking Format

What is a process ?

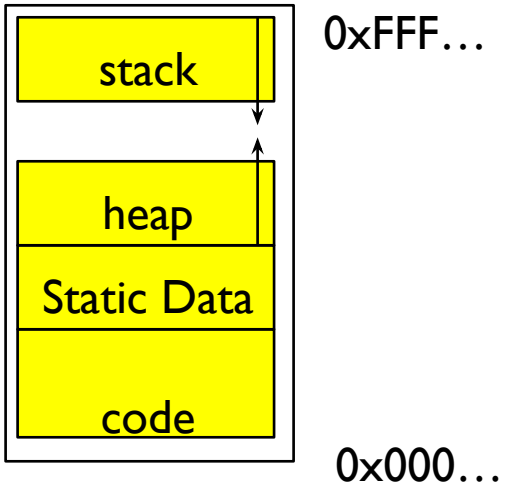
- A process is a program in execution
- The process is the OS's abstraction for execution
- Execution environment with restricted rights
  - (Protected) Address Space with One or More Threads
  - Owns memory (address space), file descriptors, sockets
  - Encapsulate one or more threads sharing process resources
- Simplest (classic) case: a sequential process
  - A single thread of execution (an abstraction of the CPU)
- A sequential process is
  - The unit of execution
  - The unit of scheduling
  - The dynamic (active) execution context
    - vs. the program – static, just a bunch of bytes

Process – Two key abstractions

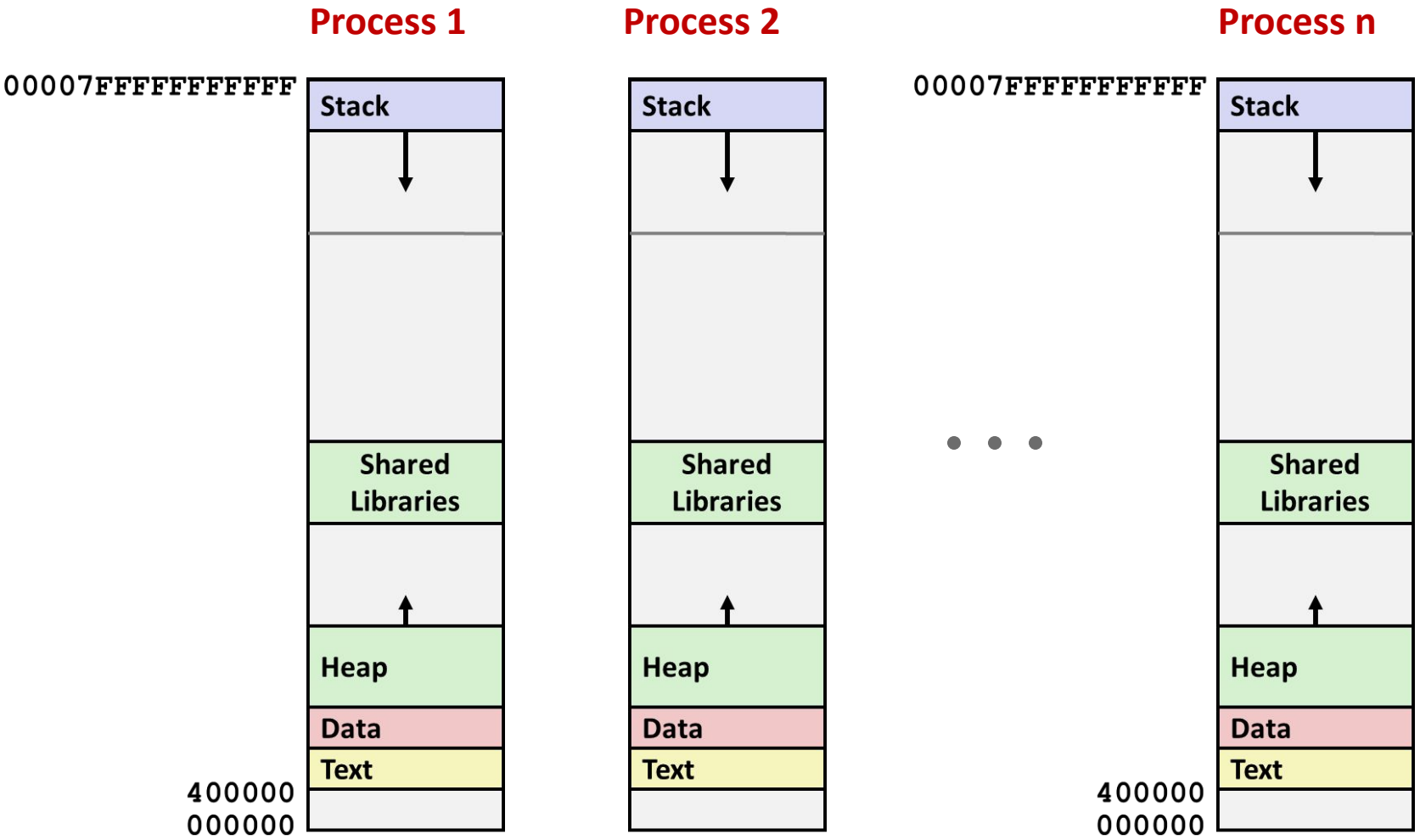
- Process is not the binary source code
  - It is an instance of running program
- Process provides two key abstractions
  - Memory
    - Each process assumes entire system memory to itself
  - Execution
    - Provides abstraction of continued operation
      - There may be hundreds of processes in a system
      - Each process gets impression of continuous operation

Second OS Concept: Address Space

- Address space ⇒ the set of accessible addresses + state associated with them:
  - For 32-bit processor: 2^{32} = 4 billion (10^9) addresses
  - For 64-bit processor: 2^{64} = 18 quintillion (10^{18}) addresses
- The address space of a process contains all of the memory state of the running program
- What happens when you read or write to an address?
  - Perhaps acts like regular memory
  - Perhaps causes I/O operation
    - (Memory-mapped I/O)
  - Perhaps causes exception (fault)
  - Communicates with another program

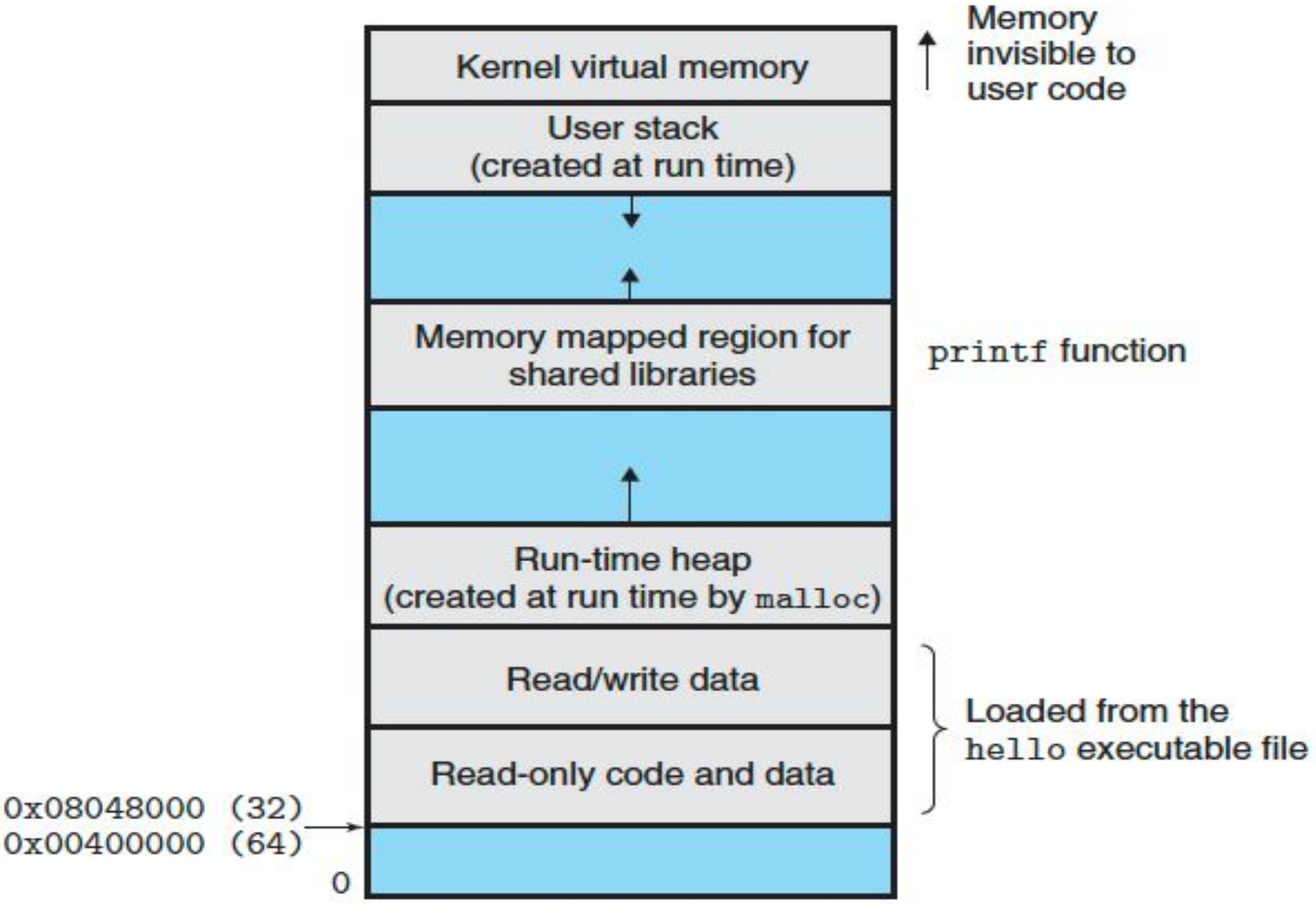


Recall: Each process can have the same address space



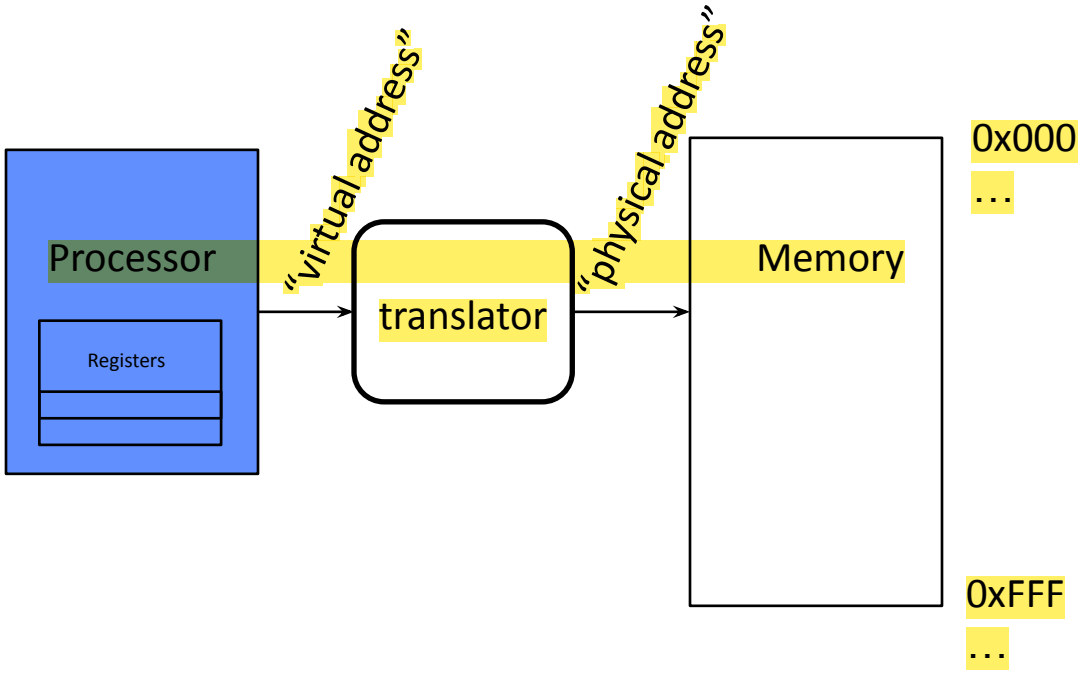
Solution: Virtual Memory

Address Space

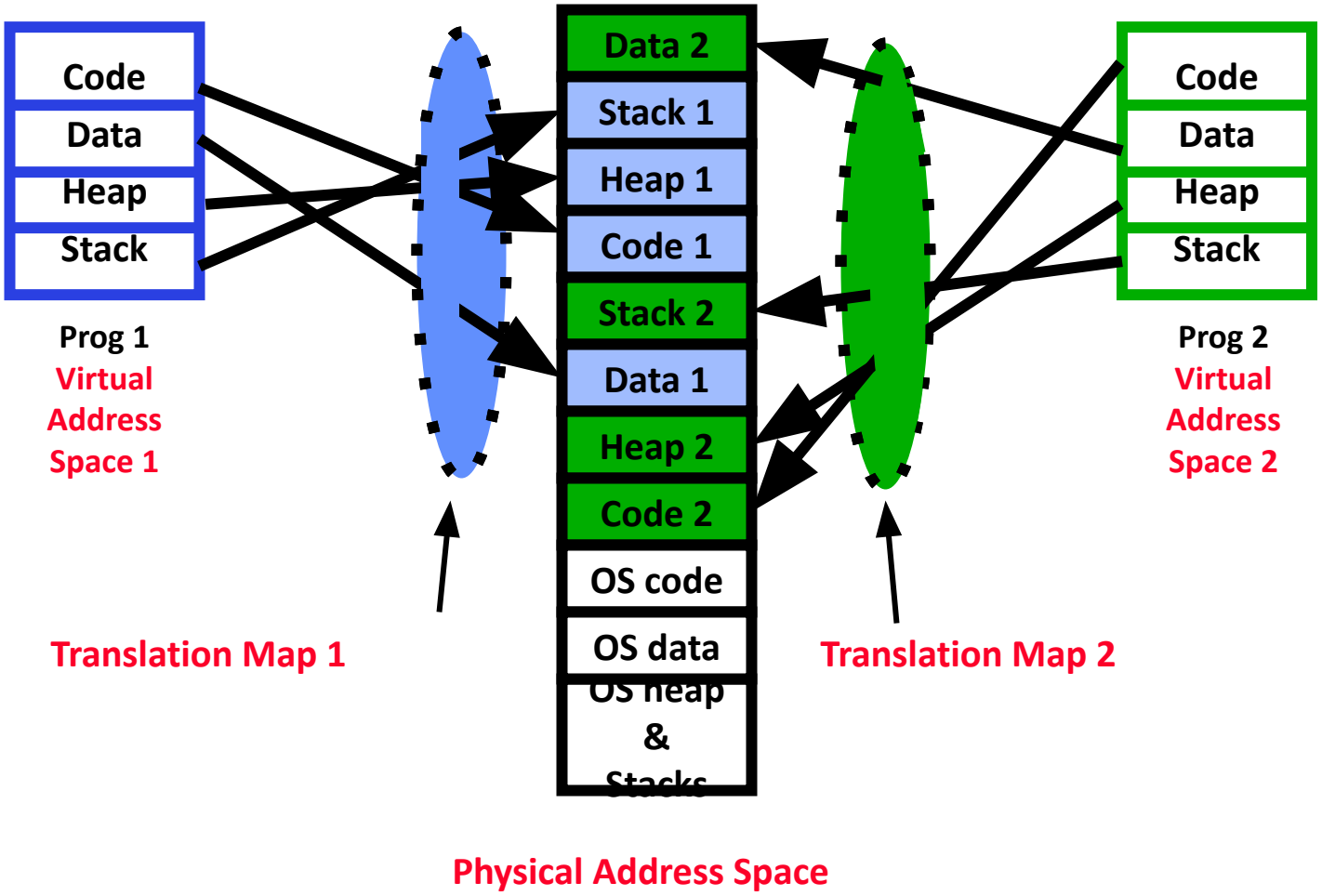


Address Space Translation

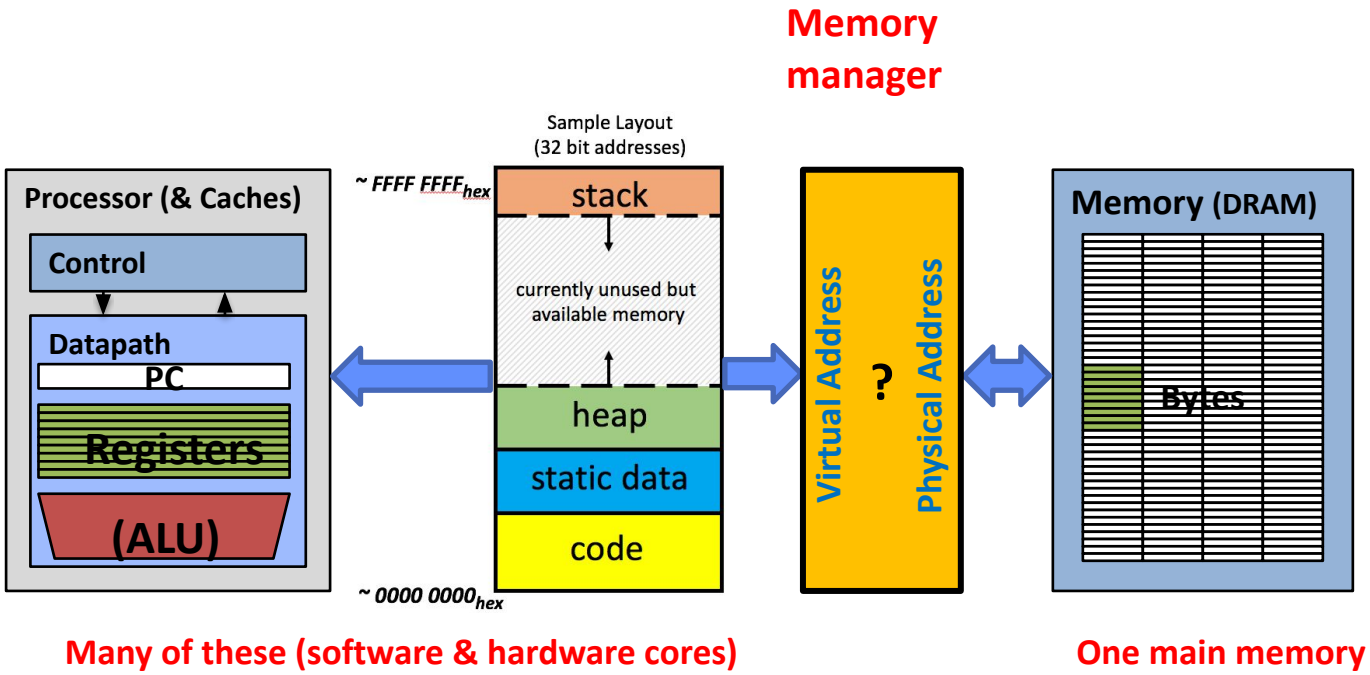
- Program operates in an address space that is distinct from the physical memory space of the machine



Translation through Page Table



Recall: Processors operate in Virtual Address Space



- Each Process uses it's own address space
- Processes use virtual addresses
- Many processes, all using same (conflicting) addresses
- Memory uses physical addresses (also, e.g., 0 ... 0xffff,ffff)
- ***Memory manager maps virtual to physical addresses***

The Third OS concepts – Threads

Thread

- A single-execution stream of instructions that represents a separately schedulable task
  - OS can run, suspend, resume a thread at any time
  - bound to a process (lives in an address space)
- Finite Progress Axiom: execution proceeds at some unspecified, non-zero speed
- Each thread is very much like a separate process, except for one difference: they *share* the same address space and thus can access the same data
- A **multi-threaded** program has more than one point of execution (i.e., multiple PCs, each of which is being fetched and executed from)

Thread of Control

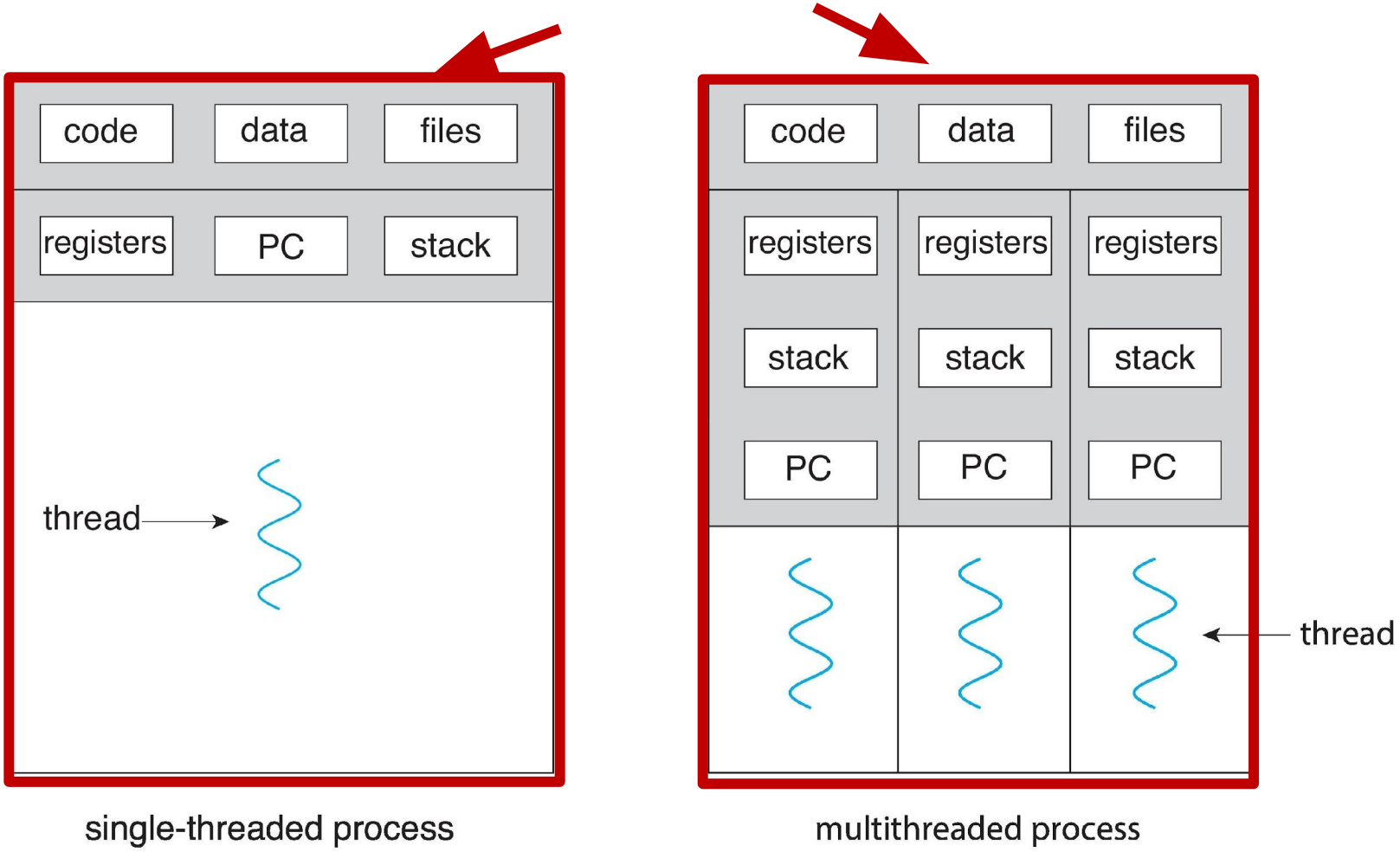
- Thread: Single unique execution context
  - Program Counter, Registers, Execution Flags, Stack, Memory State
- A thread is executing on a processor (core) when it is *resident* in the processor registers
- Resident means: Registers hold the root state (context) of the thread:
  - Including program counter (PC) register & currently executing instruction
    - PC points at next instruction in memory
  - Including intermediate values for ongoing computations
    - Can include actual values (like integers) or pointers to values in memory
  - Stack pointer (register) holds the address of the top of stack (which is in memory)

First OS Concept: Thread of Control

- At the time of context switching, the register state of the running thread is saved into Thread Control Block (TCB)
  - Will study it later

- A thread is suspended (not executing) when its state is not loaded (resident) into the processor
  - Processor state pointing at some other thread
  - Program counter register is not pointing at next instruction from this thread
  - Often: a copy of the last value for each register stored in memory

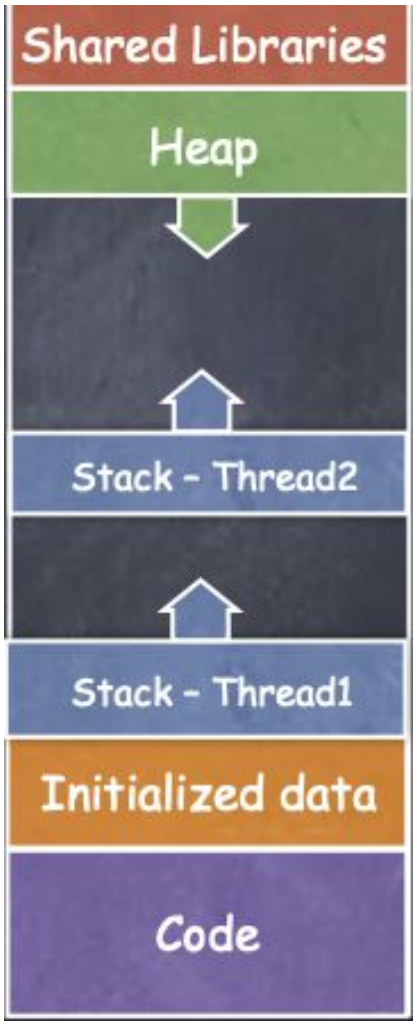
Single and Multithreaded Processes



- Threads encapsulate concurrency: "Active" component
- Address spaces encapsulate protection
- Threads share memory; In processes – sharing memory is complex

Thread

- All threads within a process share
  - Heap
  - Global/static data
  - Libraries
- Each thread has a separate
  - Program Counter
  - Stack
  - Registers
- Note two threads of the same process

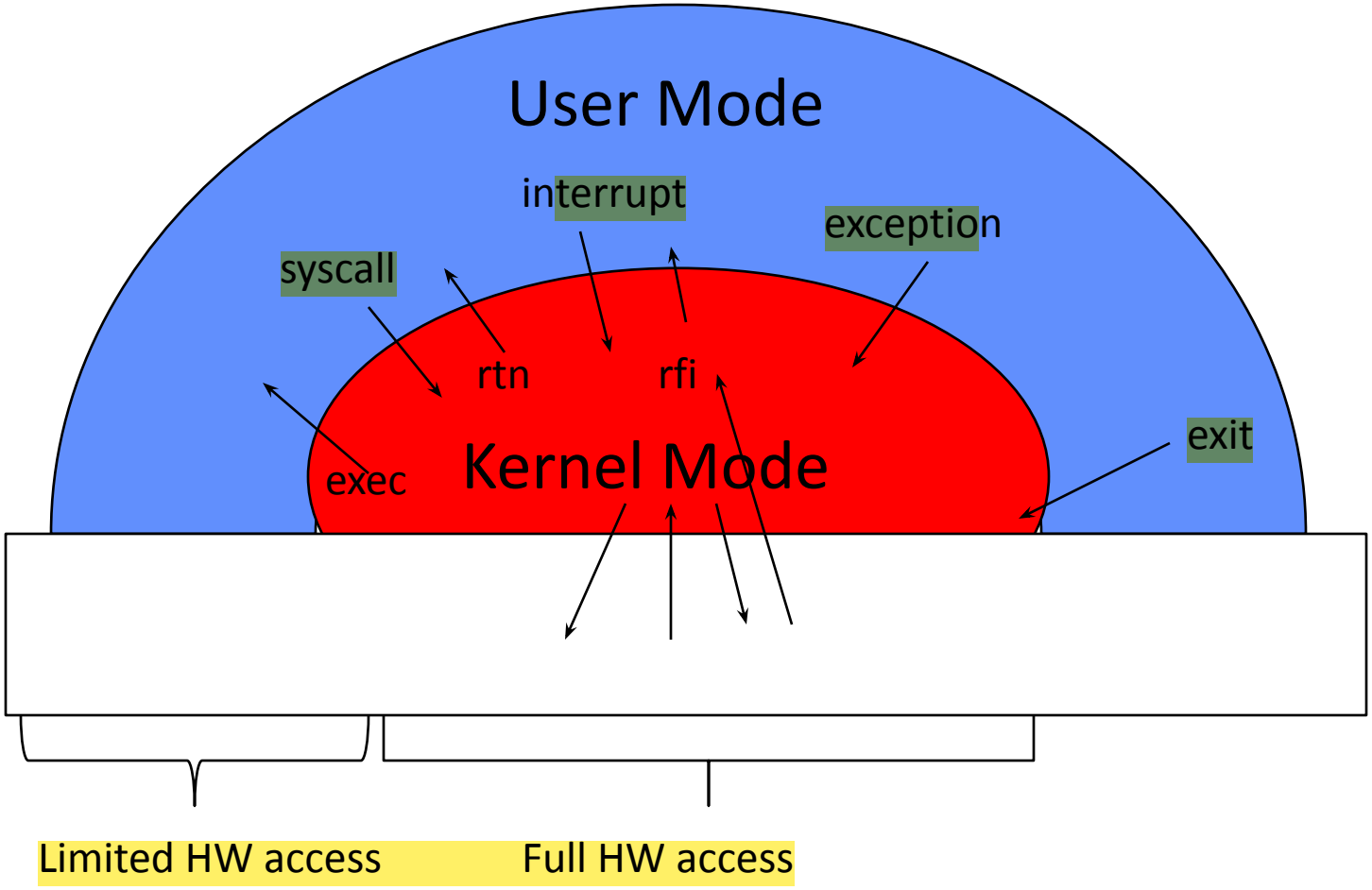


The Fourth OS concepts – Dual Mode of Operation

Why do we need two (or more) modes of operation?

- Why not have just one mode: User mode ?
- Security?
- Can a user access OS code and data?

User/Kernel (Privileged) Mode



Fourth OS Concept: Dual Mode Operation

- Hardware provides at least two modes:
  - "Kernel" mode (or "supervisor" or "protected")
  - "User" mode: Normal programs executed
- What is needed in the hardware to support "dual mode" operation?
  - A bit of state (user/system mode bit)
  - Certain operations / actions only permitted in system/kernel mode
    - In user mode they fail or trap
  - User Kernel transition *sets* kernel mode AND saves the user PC
    - Operating system code carefully puts aside user state then performs the necessary operations
  - Kernel User transition *clears* kernel mode AND restores appropriate user PC
    - Example: return-from-interrupt

Multiprogramming, Context Switching, and Multiprocessing

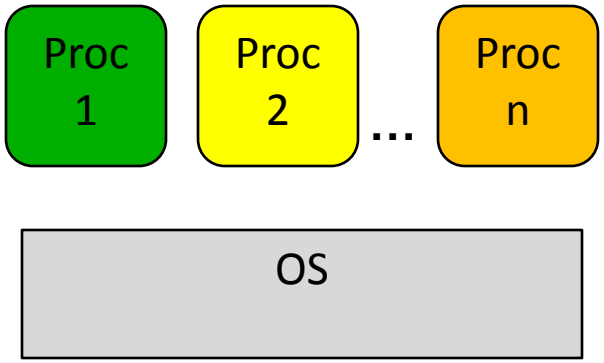
Multiprogramming

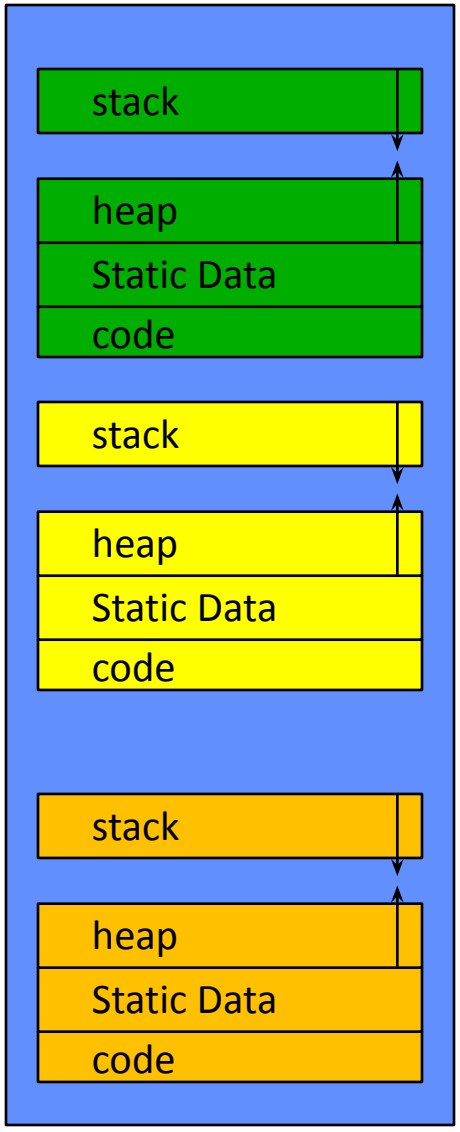
- On a modern computer, there are dozens of different processes running simultaneously -- but only a few CPUs

- In the period of microseconds, the OS rapidly switches between all processes to allow each process to run on a CPU
  - In the multi-programming system, one or multiple programs can be loaded into its main memory for execution

- When the OS swaps out one process from one CPU and allows a new process to run, this is called a context switching

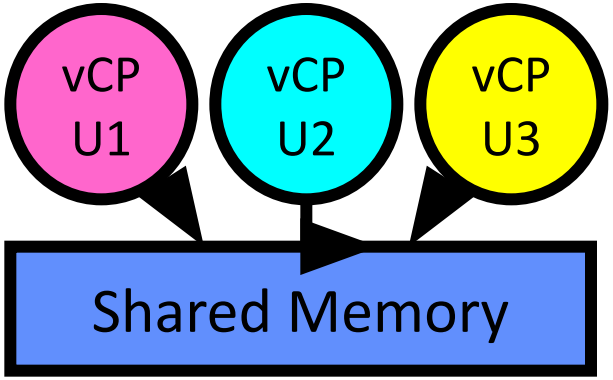
Multiprogramming

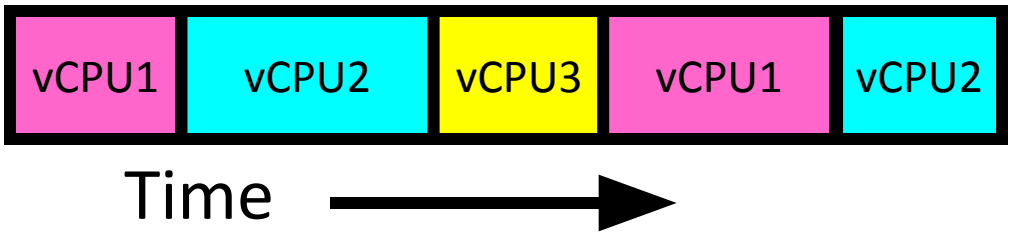




OS timeshares CPU across multiple Processes

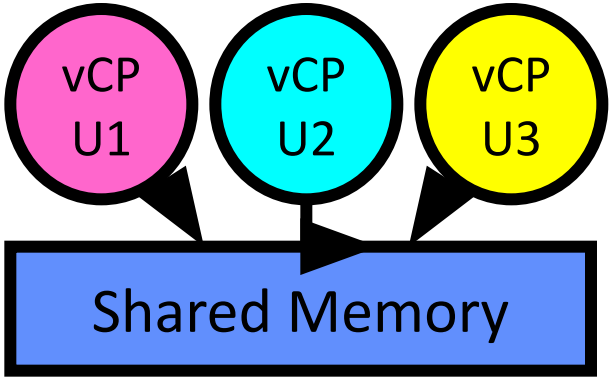
Illusion of Multiple Processors

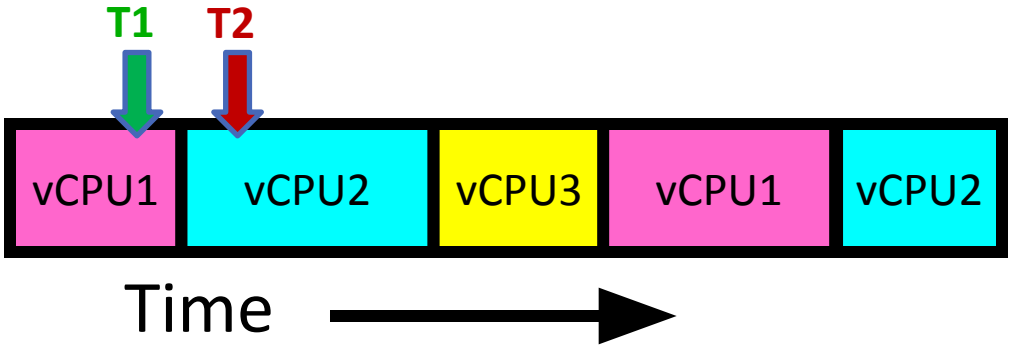




- Assume a single processor (core). How do we provide the illusion of multiple processors?
  - Multiplex in time!
- Threads are *virtual cores*
- Contents of virtual core (thread):
  - Program counter, stack pointer
  - Registers
- Where is "it" (the thread)?
  - On the real (physical) core, or
  - Saved in chunk of memory – called the *Thread Control Block (TCB)*

Illusion of Multiple Processors (Continued)





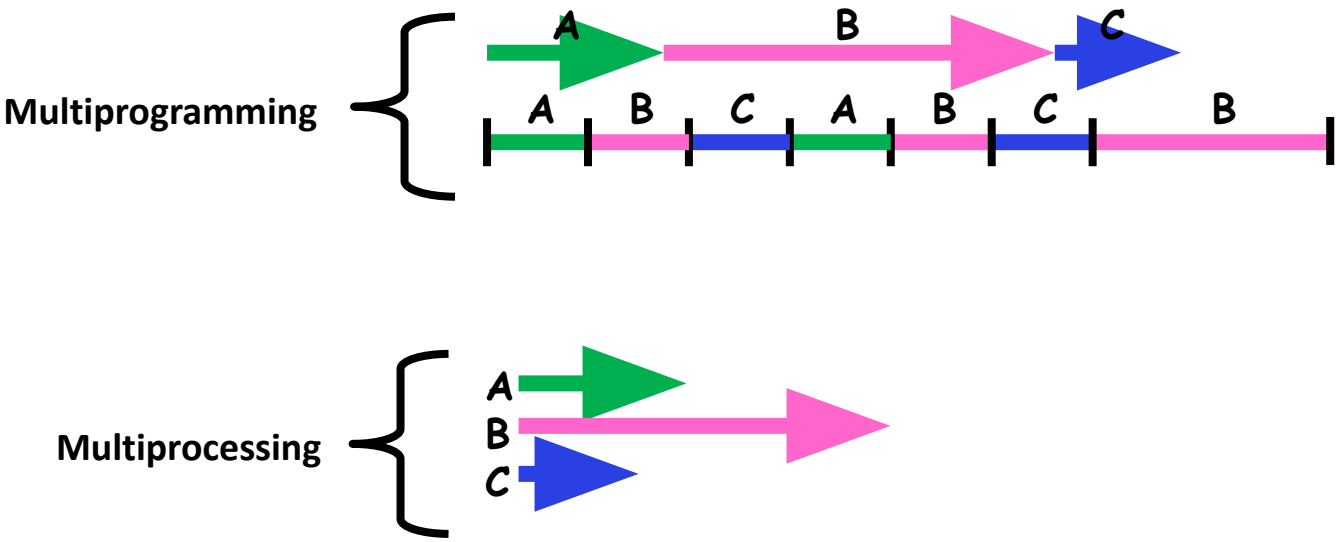
- Consider:
  - At T1: vCPU1 on real core, vCPU2 in memory
  - At T2: vCPU2 on real core, vCPU1 in memory
- What happened?
  - OS ran, triggering context switch
  - Saved PC, SP, … in vCPU1's thread control block (memory)
  - Loaded PC, SP, … from vCPU2's TCB, jumped to PC
- What triggered this switch?
  - Timer, voluntary yield, I/O, other things
- OS has a CPU scheduler that picks up on of the process/thread
  - Policy: Which Process
  - Mechanism: How to context switch

What is required during context switching?

- CPU
  - Save all CPU registers
- Caches
  - Save all CPU caches specific to the process
- Page Table
  - Page tables are unique to each process
  - Switch to the new page table (for the process to be executed)
- Overall Cost
  - Expensive
  - OS must minimize time for context switching
    - A lot of progress is made in the few decades
      - Still it is a few microseconds
- Each process contains one or more threads that can

Multiprocessing vs. Multiprogramming

- Multiprocessing: Multiple CPUs(cores)
- Multiprogramming: Multiple jobs or multiple processes
- Multithreading: Multiple threads per process



Concurrency is not Parallelism

- Concurrency is about handling multiple things at once
- Parallelism is about doing multiple things *simultaneously*
- Example: Two threads on a single-core system...
  - … execute concurrently …
  - … but *not* in parallel
- Parallel => concurrent, but not the other way round!

Lecture Summary

- Program in execution is a process
- In multiprogramming systems, one or multiple processes are resident in the main memory and they run concurrently
- The process of switching the CPU from one process to another (stopping one and starting the next) is the context switch
  - Context switching is relatively expensive operation
- An important OS data-structure for multiprogramming/context switching is Process Control Block
  - Will study PCB later in the course