Operating Systems Principles

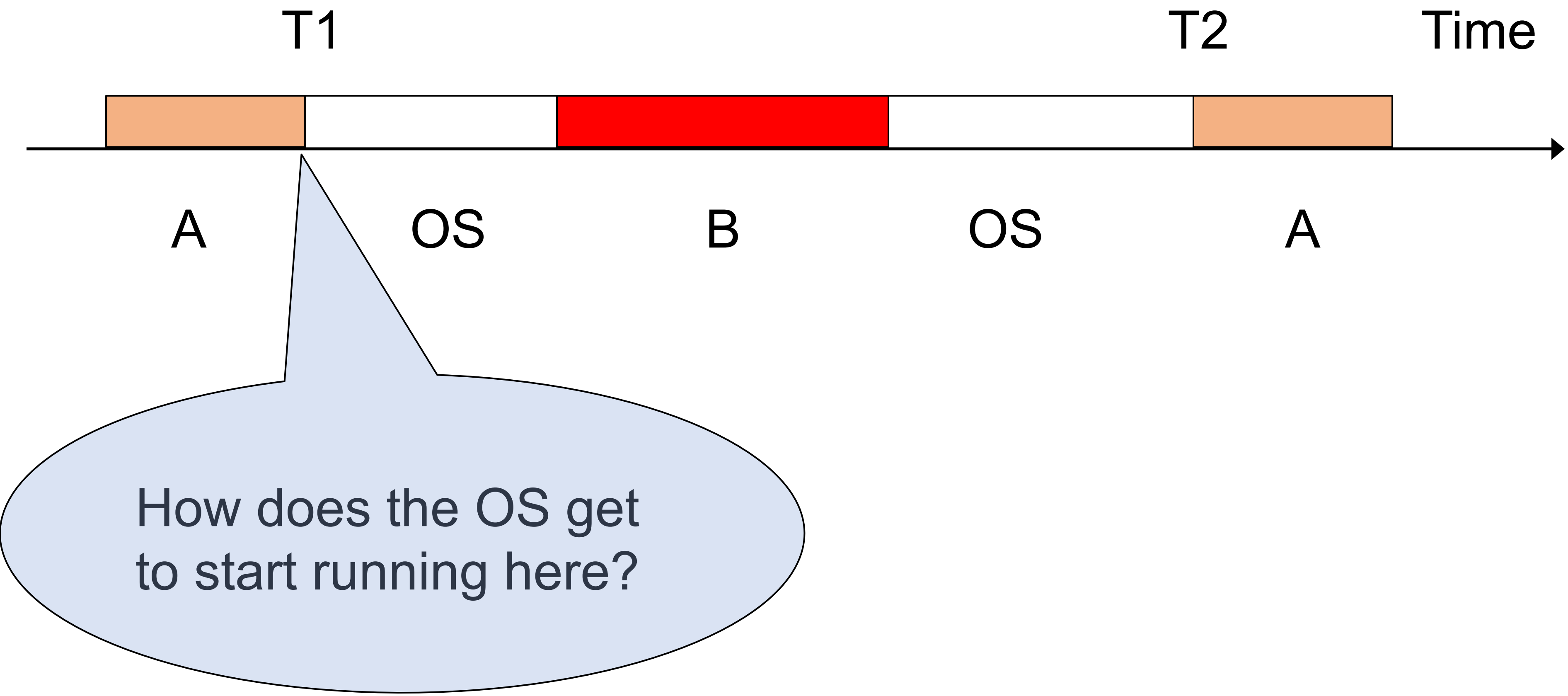
CPU Management (Processes)

Slides are adapted from Prof. Bhuvan Urgaonkar

How Does the OS Help Multiple Processes Share the Same CPU?

E.g., Two processes on a CPU

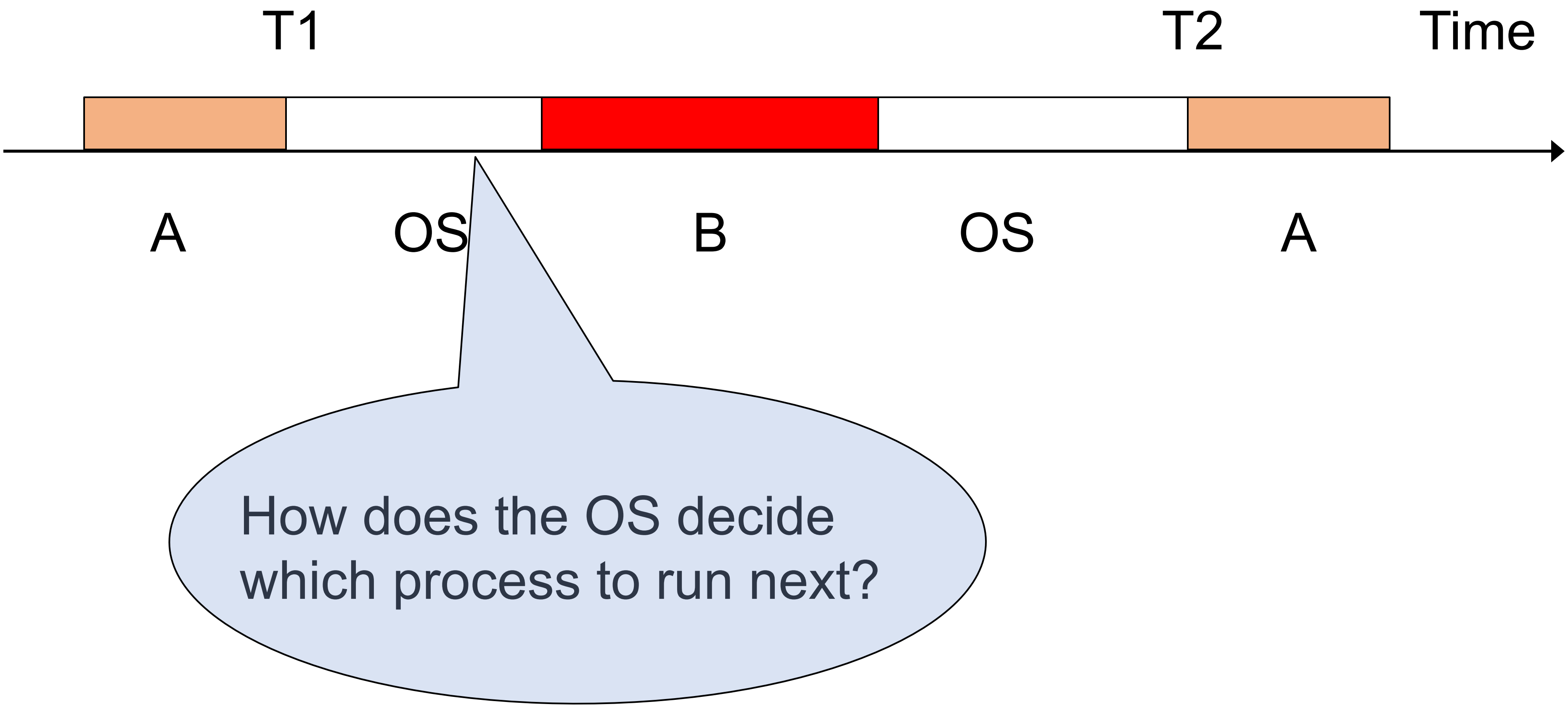
- Lets consider (only) two processes A and B that are running on the same CPU
- Let us look closely at some illuminating events in such a system



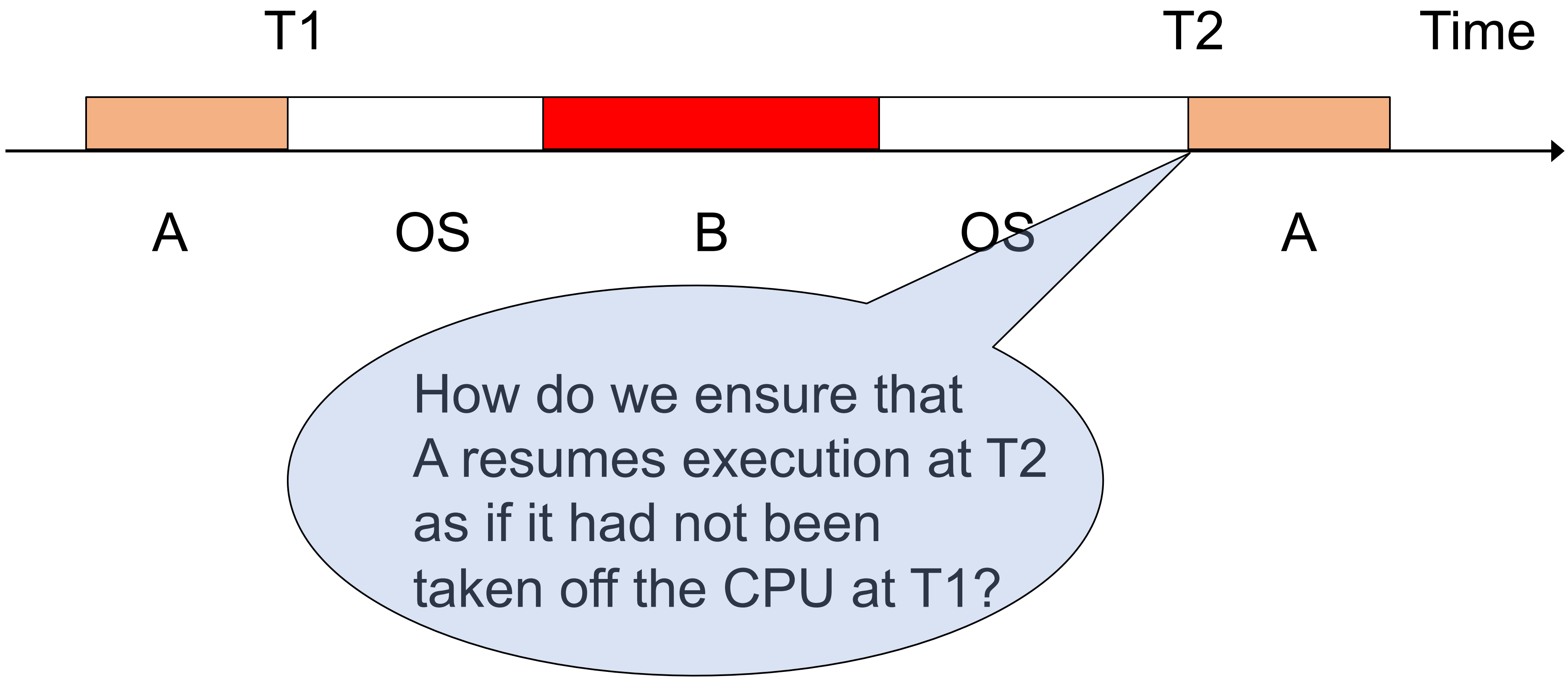
- Recall: user->privileged mode occurs only when a trap is raised or an interrupt occurs
- Can't rely on traps – why?

The Role of Interrupts

- There must be an interrupt mechanism via which the OS gets a chance to run on the CPU every so often
  - A timer interrupt that periodically lets the OS run, typically, once every few milliseconds



- Done by the CPU scheduler within the OS
- Will study later

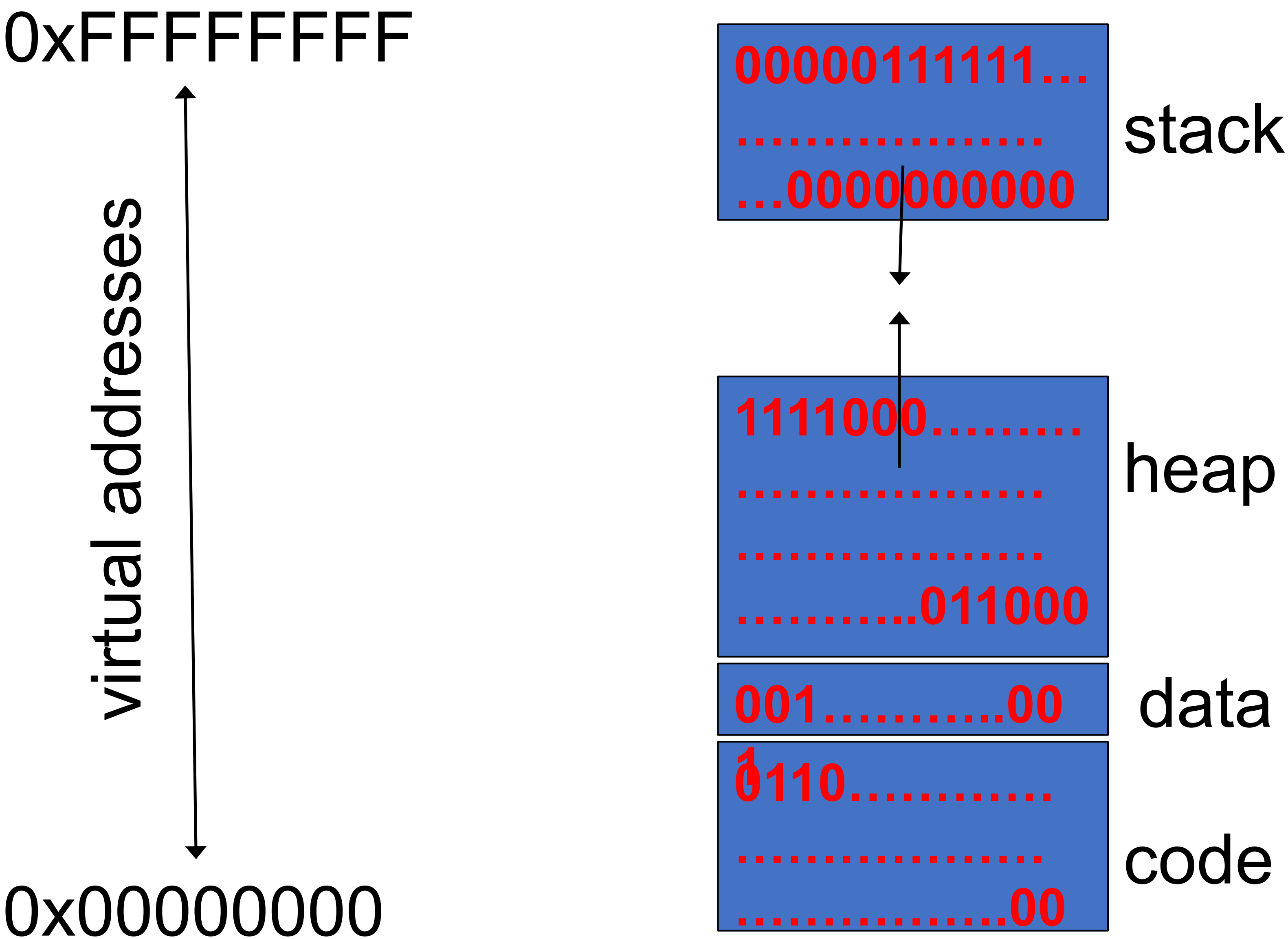


- By ensuring that we save the entire “state” of A at T1 and can resume it from this state at T2
- state(A, T1) == state(A, T2)
- What does the state of A at T1 consist of?

State of A at time T1 (1)

- #1: Contents of A's address space
  - What the code, data, heap, and stack contain at T1

Contents of A's Address Space



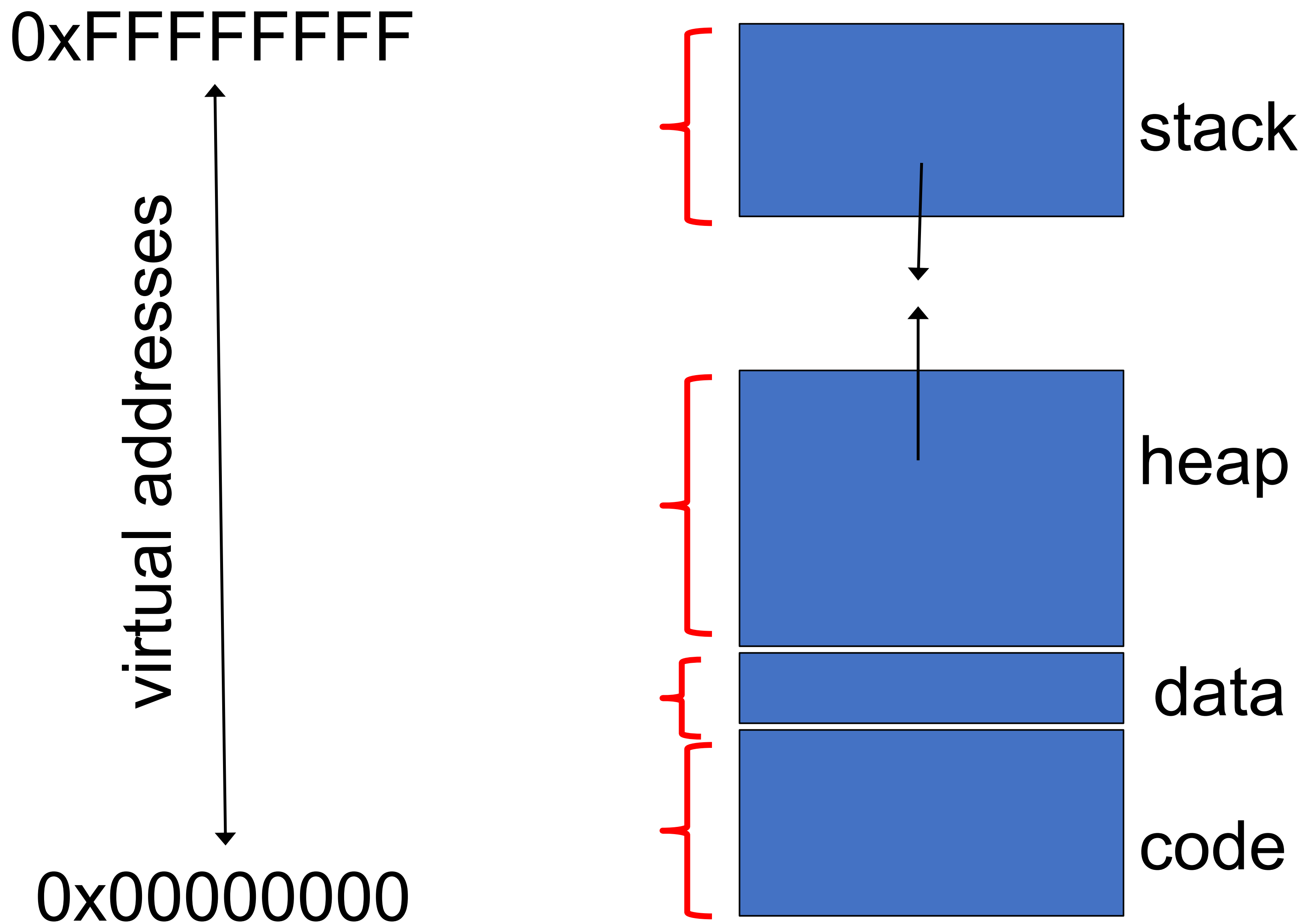
State of A at time T1 (1)

- #1: Contents of A's address space
  - What the code, data, heap, and stack contained at T1
- Q: Where do these contents reside at time T1?
  - In portion of main memory set aside for A
  - We rely on memory manager to ensure they remain unchanged by other processes during [T1, T2]
    - More details when we study virtual memory management

State of A at time T1 (2)

- #2: Layout of A's address space
  - The address ranges the code, data, heap, stack span

Layout of Address Space

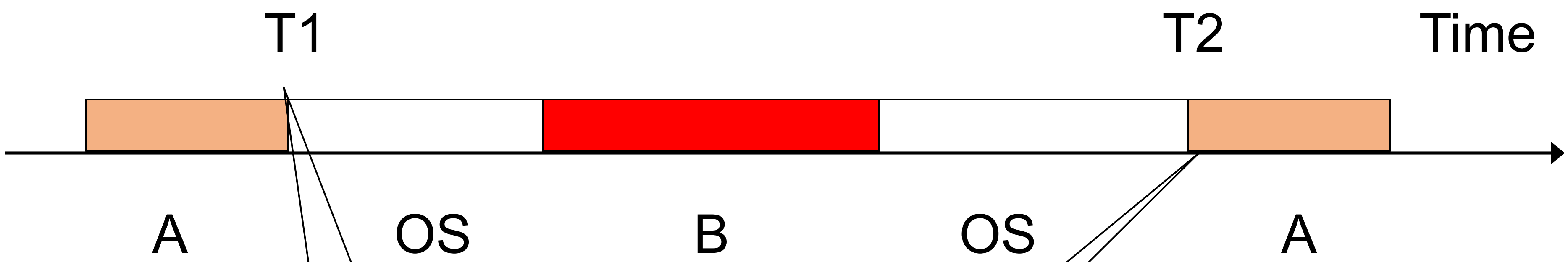


State of A at time T1 (2)

- Layout of A's address space
  - The address ranges the code, data, heap, stack span
- Q: Where are these address ranges stored?
  - Somewhere in memory
  - In whose address space? Lets say A's own address space for now (e.g., its heap or data segment)
  - **OS will need to maintain at least one address, say where A's address space begins in main memory**

State of A at time T1 (3)

- Anything else?



- Hint: Suppose A's instruction executed just before and after T1 are:

Load R1, 100

Add R1, R2, R3

State of A at time T1 (3)

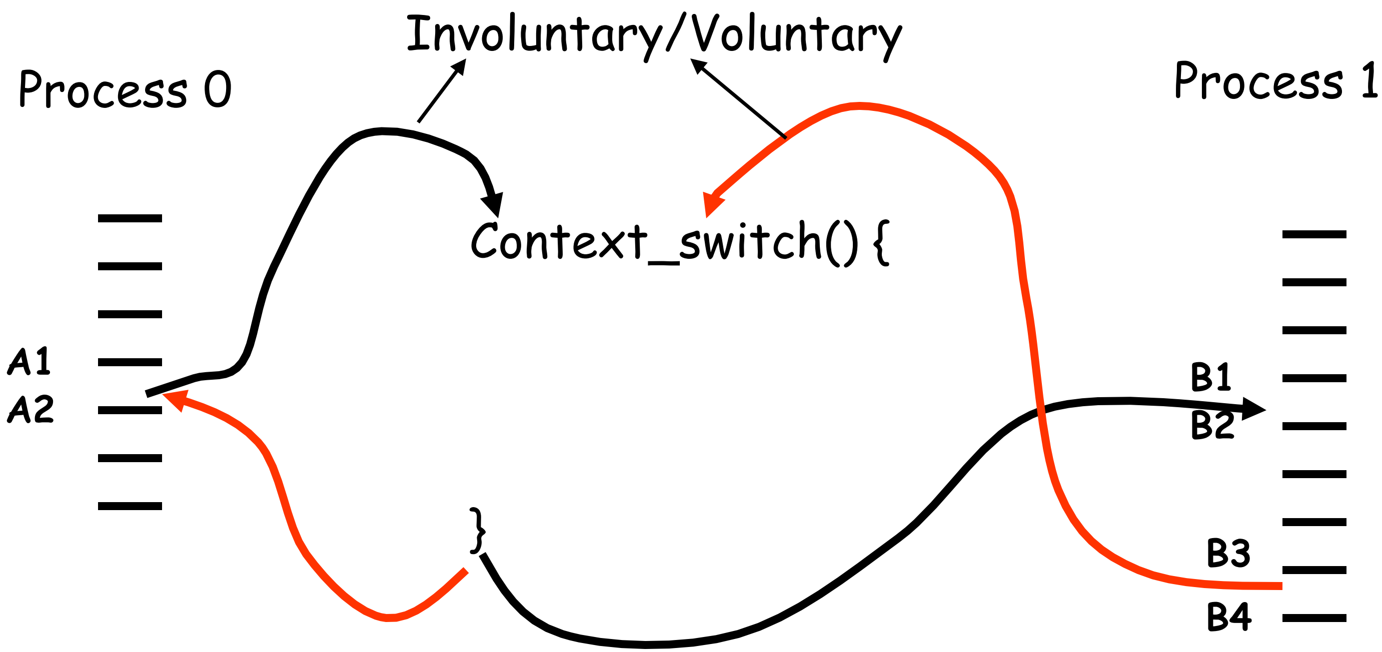
- #3: All the register values at time T1 need to be saved in main memory and restored at time T2

- Called the **hardware context** of process A
- Typically, the hardware context also has information about address space layout
  - E.g., Stack Pointer Register
  - Why does the CPU provide this facility?
    - Because registers are accessed much faster than memory

Process Control Block (PCB)

- A data structure in the operating system's address space where it maintains a variety of information about the process
  - Example we learnt today: Values held in various registers when the process was context switched out

Context Switch



```
context_switch() {

  push R0, R1, …                // save regs on its stack
  PCB[curr].SP = SP             // save stack pointer
  PCB[curr].PT = PT             // save ptr(s) to address space

  next =  schedule()            // find next process to run

  PT = PCB[next].PT
  SP = PCB[next].SP
  pop Rn, … R0

  return                        // NOTE: Ctrl returns to another process
}
```

Overheads of Process Switch

- Direct
  - “Wasted” instructions of outgoing process
  - The time spent on switching context

- Indirect
  - Cache pollution
  - TLB flush