Processes

What is a process?

- “An execution stream in the context of a particular process state.”
- State variables
  - Instruction Pointer
  - Registers
  - Stack Variables
  - Memory
  - etc.

Processes

- A process is similar to a program…

- A **program** is a set of instructions to perform, in a certain order
- A **process** is one thing that is executing that program
- But there can be **multiple processes** running the **same program**

Processes

- Internally, processes **pretend** that they are running on a uni-programming machine
  - Only one CPU, and we run on it 100% of the time
  - Absolute control over memory
  - Never blocked

- Syscalls break this illusion, but we mostly treat them **like function calls**

Processes

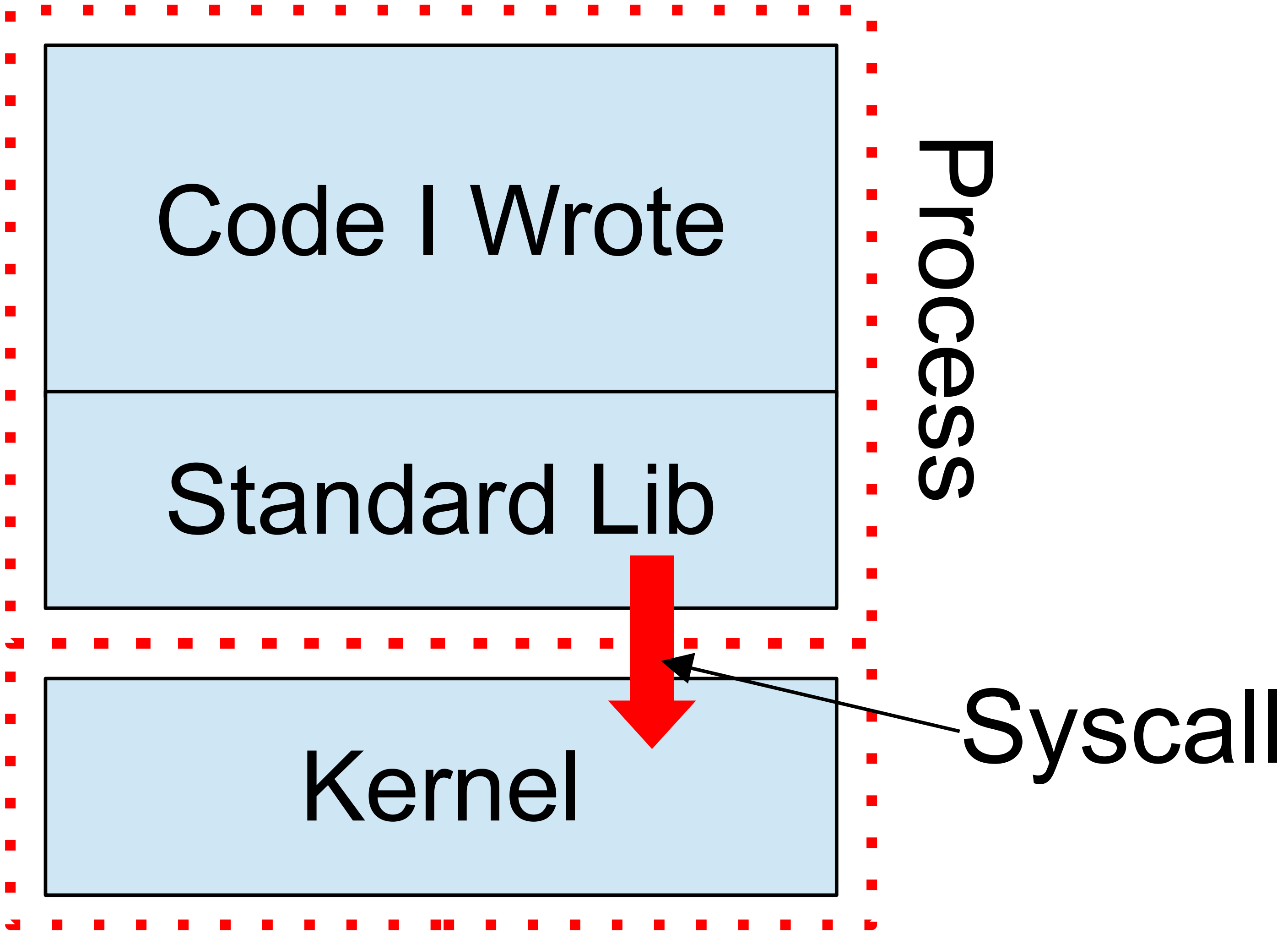
- Processes routinely link to **shared libraries**
  - Classic code, often provided as part of the OS. Or can be custom libraries, just for you

- Shared libraries are **part of the process**
  - Even though you didn't write them, they are integrated into your program
  - No special rights or powers (or limits)
  - OS treats library code the same as main

Syscalls

- **syscalls** are special routines that are part of the kernel
  - Often, wrapped with nice functions to make them easy to use
  - But the system call itself is actually known as a “synchronous interrupt”
    - Blocks the CPU, forces you into the kernel
    - But has to happen ***now,*** to ***this*** process
      - Not on any other CPU

Syscalls



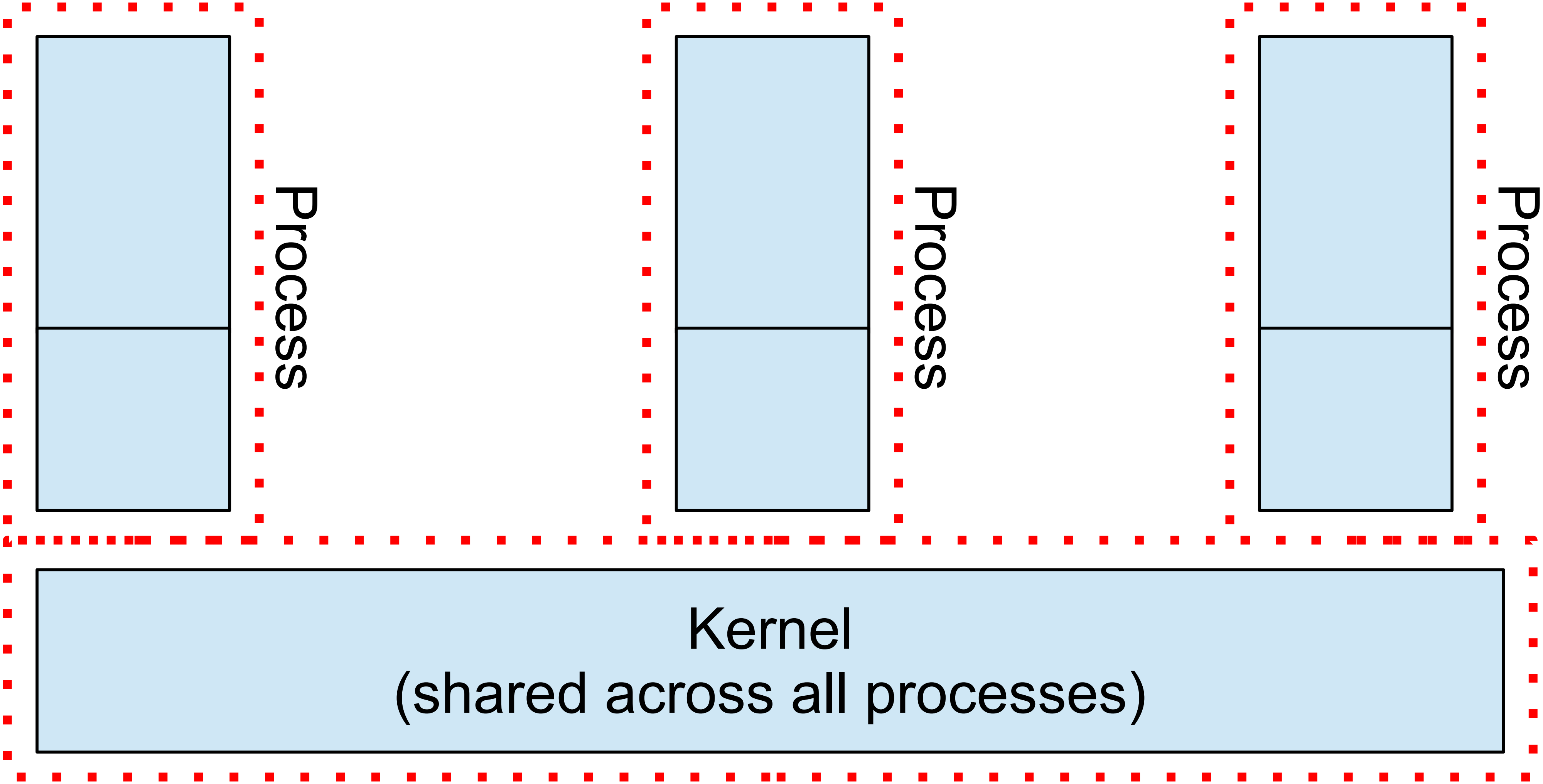
Processes

Context Switching

Multiprocessing

- In truth, many processes are on the box at the same time (100s)
  - Have to share CPU
  - Have to (safely!) share memory

Multiprocessing



Kernel
(shared across all processes)

Blocking

- Processes will sometimes block for a while
  - Sleeping
  - Waiting for I/O
  - Waiting for communication
    - Inter-process communication
    - Sockets & network

- Other times, they run endlessly
  - Useful or a bug???

Time Sharing

- What happens when multiple processes want the CPU?
- Old, broken idea: hope for the best
  - Can switch processes at any syscall or interrupt. But what if none happen?
- Better: timer interrupt

Pre-emptive Multitasking

- **Timer interrupt** fires 100-1000 times per second
  - Each interrupt is a chance to switch processes, if we wish
  - Processes run for a while (efficiency)
  - But not forever (correctness)
- CPU chooses **scheduling policy**
  - Many options
  - Round-robin common

Pre-emptive Multitasking

- From inside a process it is **impossible** to reliably predict your speed
  - Cache hits & misses
  - Page faults
  - Random interrupts (now including timer!)
  - OS code could run for short or long
  - etc.

Pre-emptive Multitasking

- From inside a process all interrupts feel **random** (even if they are predictable)

- **Any instruction** can be interrupted
- **Impossible** to guarantee “I’ll do this quickly”
  - Could be context-switched out for long time

Pre-emptive Multitasking

More correct:

- From inside a process, you ***never know*** that interrupts occur
  - Process only cares about its own ***internal, logical state***
  - Individual instructions run fast or slow, and **nobody cares**

Context Switching

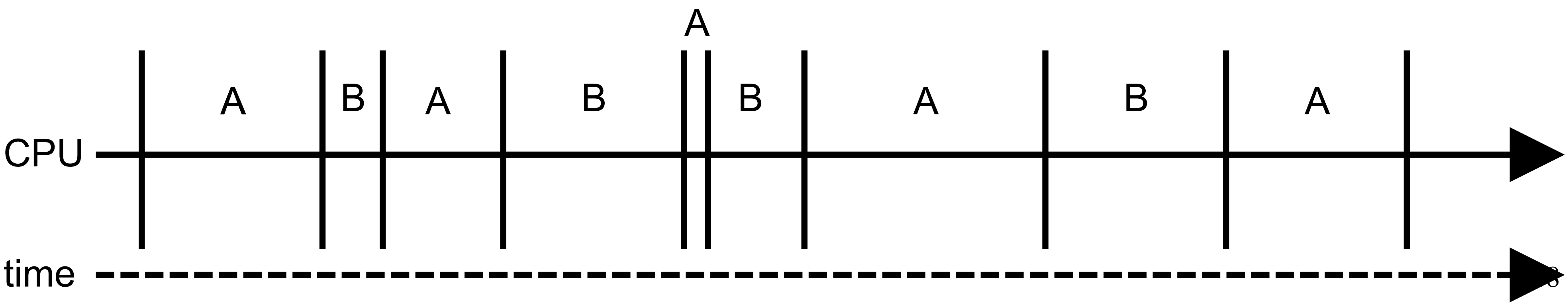
- When the OS chooses to remove a process from the CPU, it performs a **context switch**
  - Saves all CPU registers, including program counter
  - Changes virtual memory config
  - Then loads CPU with new info

Context Switching

- CPU perspective
  - Process A was running. Now process B is running
- Process perspective
  - I was running, and then I was frozen in time
  - Later, I was restored to the **exact same state**
- ***Truer*** process perspective
  - I never stopped running

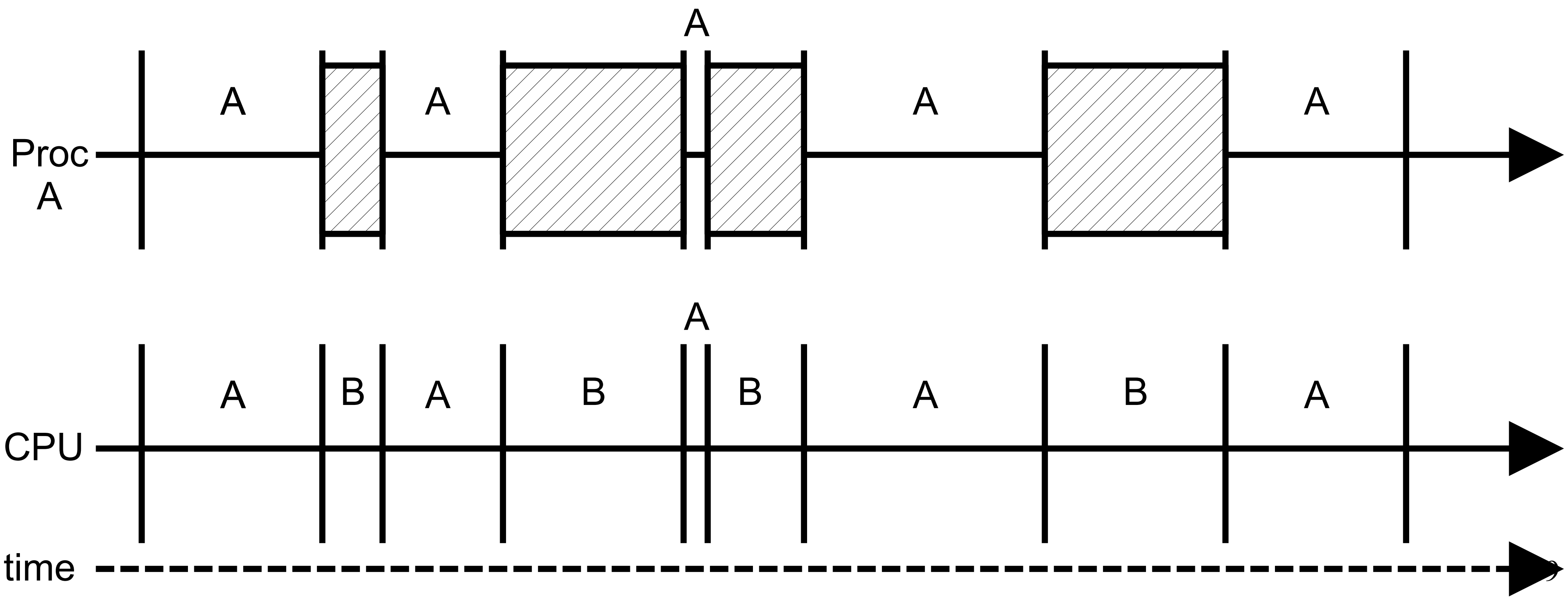
Context Switching

Over time, a single CPU does work for two different processes.

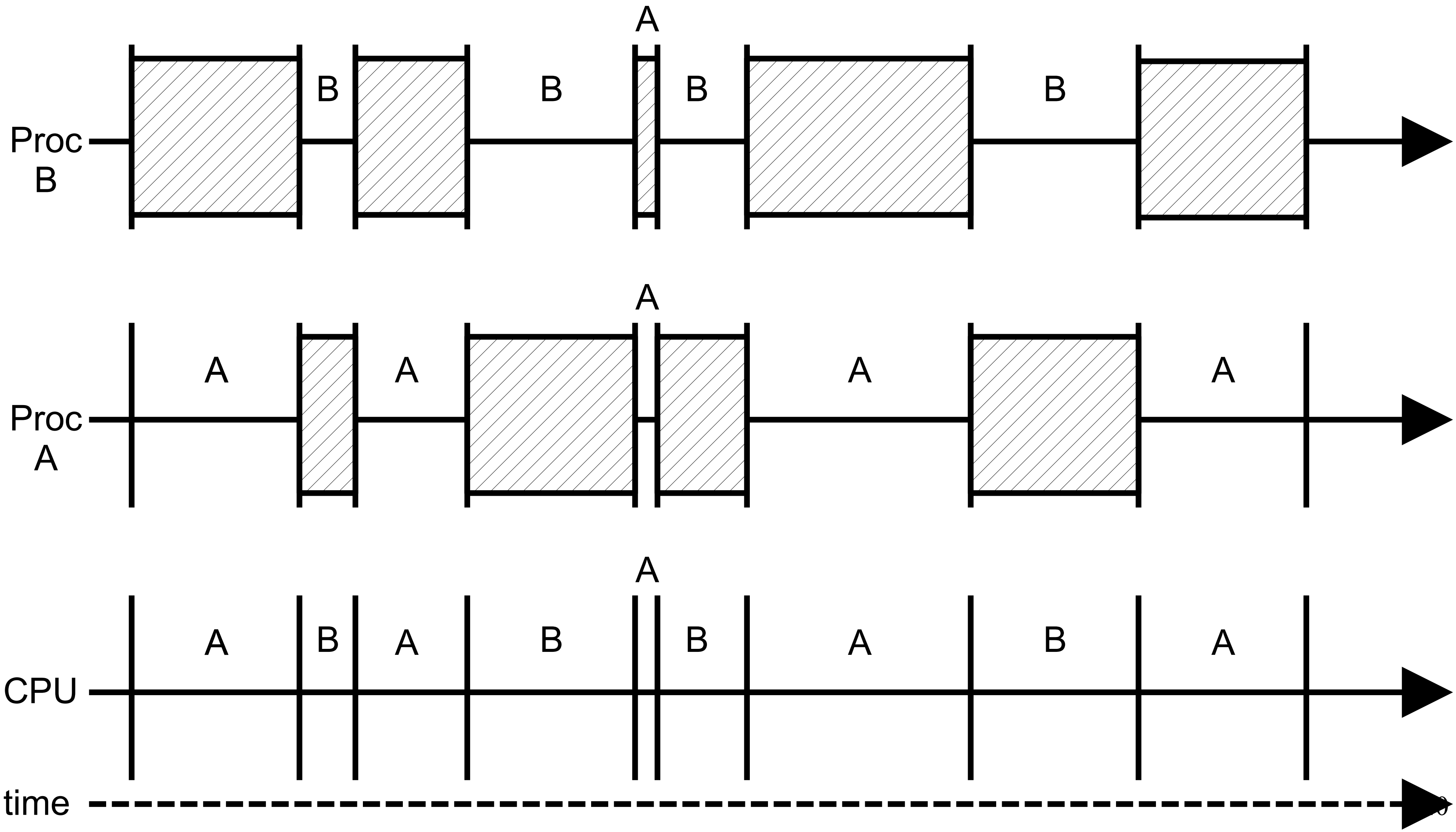


Context Switching

From the perspective of Process A, it looks like there were long pauses.



Context Switching



Context Switching - A Kernel Perspective

- Context switching in the user is always **implicit**
  - Ask for an operation to be performed, but it will take time
  - Interrupt occurs unexpectedly

- In the kernel, it has to be **explicit**
  - Some functions will (or might) perform a context switch ***away from your process***

Context Switching -
A Kernel Perspective

```
void some_func(…)
{
  USLOSS_Context *oldCtx = ... ;
  USLOSS_Context *newCtx = ... ;
  USLOSS_Console(“Before\n”);
  USLOSS_ContextSwitch(oldCtx,newCtx);
  USLOSS_Console(“After\n”);
}
```

When does `Before` get printed?
When does `After` get printed?

Q: When does `Before` get printed?

A: Immediately!

Q: When does `After` get printed?

A: A long time from now!

Why is that???

```
                                                            ;
                                                            ;
    USLOSS_Console("Before\n");
    USLOSS_ContextSwitch(oldCtx,newCtx);
    USLOSS_Console("After\n");
}
```

Processes

- When we restore a process after it’s been frozen for a while…
  - Restore CPU registers
  - Set up virtual memory

- Process should pick up where it left off
- What does it execute next?
  - The line **after** it performed the context switch!

Context Switching - A Kernel Perspective

```
void some_func(…)
{
  USLOSS_Context *oldCtx = ... ;
  USLOSS_Context *newCtx = ... ;
  USLOSS_Console(“Before\n”);
  USLOSS_ContextSwitch(oldCtx,newCtx);
  USLOSS_Console(“After\n”);
}
```

When we come back…

When did `Before` get printed?

When does `After` get printed?

When we come back…

Q: When did `Before` get printed?

A: A long time ago

Q: When does `After` get printed?

A: Immediately

```
                               ;
                               ;
  USLOSS_Console("Before\n");
  USLOSS_ContextSwitch(oldCtx,newCtx);
  USLOSS_Console("After\n");
}
```

Processes

Process Control Block

Process Control Block

- A **process control block** stores everything that the kernel knows about a process
- The **process table** is a fixed-length array of PCBs

Process Control Block

- The **process control block** stores:
  - Process ID & name
  - Runnable state
  - Context variables (for saving state)
  - Accounting information
  - Open files & allocated resources
  - … anything that the kernel knows about the process ...