Sharing Resources

Computer Systems Chapter 8.2, 8.4

Abstract View

When I run my program, it has access to the entire computer, including the processor, memory, keyboard, display, disk drives, network connections, etc. etc. etc.

Leaky Abstraction

- In fact, most hardware supports multiple concurrent users
- Each user is often running multiple programs concurrently
- System services (called “deamons”) are often running to provide real-time capabilities
- Even running on a multi-core machine, the number of concurrently running programs almost always exceeds the number of processors.

Resource Sharing Goals

- Ensure each client (e.g. running program) gets a fair share of resources
- Ensure that no client is blocked from continuing
- Ensure that busy clients get priority over idle clients

Early Resource Sharing: Batch Jobs

- Prepare your punch card deck
- Put your deck in the card reader… on top of other students
- Card reader reads the next job
- Computer processes the next job (compile, execute, print results)
- Later, you get your printout from the printer

Batch Processing / Sharing Goals

- Ensure each client gets a fair share of resources
  - Everybody gets a turn
  - First come/ first served
  - Big jobs get more resources than small jobs
- Ensure that no client is blocked from continuing
  - Fails if the student in front of me has an endless loop
- Ensure that busy clients get priority over idle clients
  - Fails when the student in front of me is waiting for IO
- No concurrency!

Naming Clients

- Need a name/handle for each running program
  - Can't be program name, because I can run the same program concurrently
  - Must be created when program starts
  - Must be deleted when program ends

- ***process*** - An <u>invocation</u> of a program
  - Process ID: a numeric identifier associated with a process (PID)
  - C Standard library function calls can create new processes [more later]
  - Ended by "exit" library call (in stdlib.h)

Process Hierarchy

- Processes can create new processes
  - The creator is called the ***parent process*** or "ppid"
  - The spawned process is called a ***child process***
- Parent processes are responsible for their children
- In UNIX, when you log on, the OS process creates a child process and assigns that process to you
  - This is the interactive shell or GUI running on your behalf

Listing Processes

- In UNIX, the “ps” command lists processes
- By default, “ps” lists your process and all of it’s children
- To list all processes owned by you, “ps –u<userid>”
- To list all processes by all owners on this machine, “ps –e”

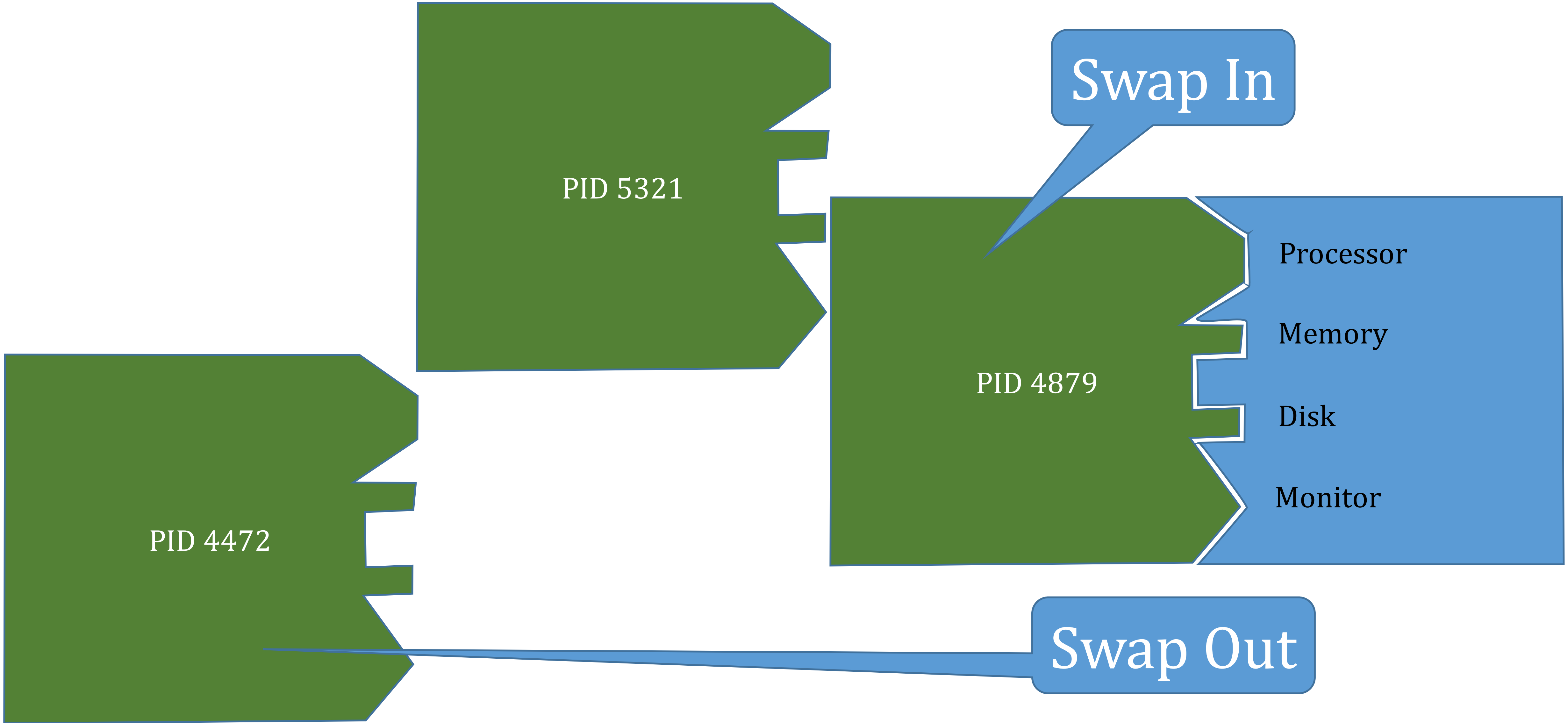
```
alpha:~/CS220> ps
 PID    TTY     TIME     CMD
2933    pts/3   00:00:00 tcsh
3057    pts/3   00:00:00 ps
```

```
alpha:~/CS220> ps -utbartens
 PID    TTY     TIME     CMD
2836    ?       00:00:00 sshd
2837    ?       00:00:00 tcsh
2839    ?       00:00:00 sftp-server
2913    ?       00:00:00 sshd
2914    ?       00:00:00 tcsh
2923    ?       00:00:00 sftp-server
2932    ?       00:00:00 sshd
2933    pts/3   00:00:00 tcsh
3058    pts/3   00:00:00 ps
```

Process Resources

- Each process THINKS it owns all machine resources
  - "virtual" processor, virtual memory, virtual keyboard, virtual monitor, virtual disks, virtual network, …
- OS connects VIRTUAL resources to REAL resources

Time Slicing



Time Slicing Concepts

- OS keeps a list of ***active*** processes
  - An active process is a process trying to execute
- OS gives each active process a slice of time to make progress
- When a process gets a slice of time, it is ***swapped in***
  - All other active processes are ***swapped out***
- When a process is swapped in, it can use real resources
  - It can actually make progress in order to complete its job
- When a process is swapped out, it does not have access to resources
  - It remains idle until it gets a time slice

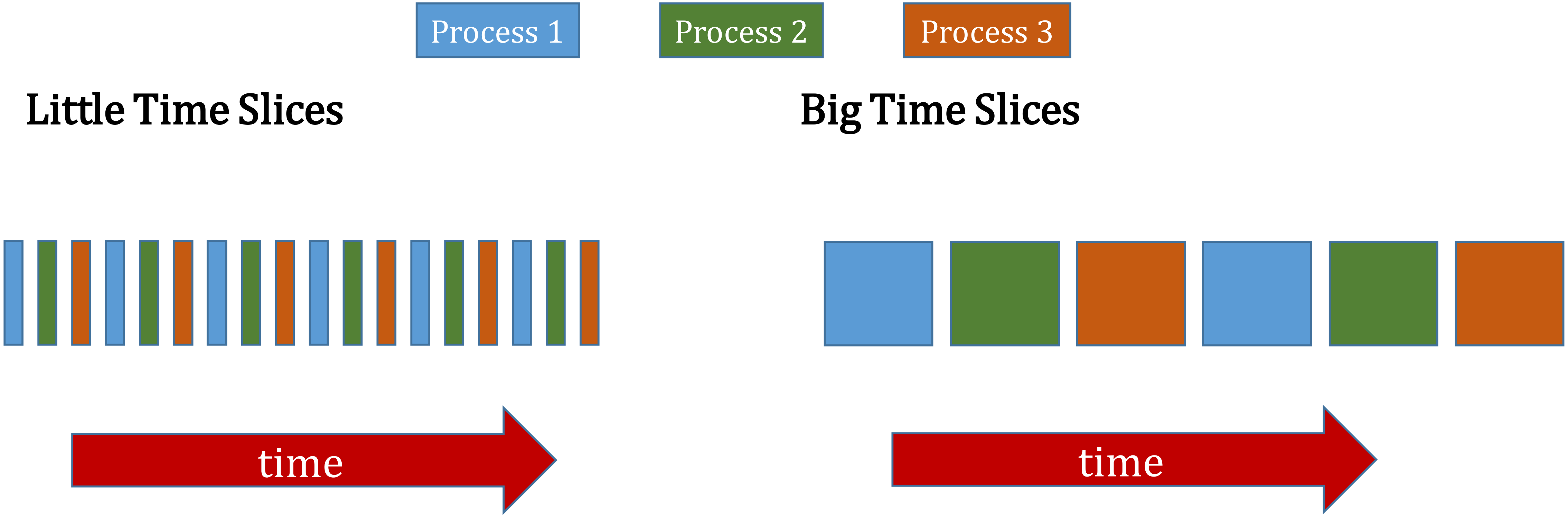
Process Context

- There is information/date associated with each process
  - Register values
  - Values in memory
  - How much data has been read from a file
  - etc.
- The sum of all state for the entire process is called the process *context*
- When a process is active, it has access to its entire context

Time Slicing Issue – Context Swap

- When a process is swapped out, we must save it's context
- When a process is swapped in, we must load it's context
- The process of saving the outgoing context, and loading the incoming context is called a "context swap"
- Context swapping is "overhead" – extra resource needed that does not do the processes work
  - No context swapping required for batch jobs

How Big should a Time Slice be?



How Big should a Time Slice be?

Little Time Slices

- Makes progress seem continuous to the user
- Increases the number of context switches required (more overhead)
- Smaller delta to swap in/out (faster, less overhead)

Big Time Slices

- Makes progress seem jerky to the user
- Decreases the number of context switches required (less overhead)
- Larger delta to swap in/out (slower, more overhead)

Process Queue

Process 1 | Process 2 | Process 3

- List of processes competing for resources
- New processes can be added to the queue
- When a program is done, it’s process can be removed from the queue

Process Swapping / Context Switch

- Wait for Instruction to End
- Save context of swap out process
  - Registers (especially EIP) & flags
  - Main Memory (stack and heap)
  - I/O status
- Restore swapped in context
  - Registers and Memory and I/O status
- Restart instruction processing cycle

Swapping Memory

Bad Idea:

Write Swap Out address space from memory to disk

Read Swap In address space from disk to memory

- A 32 bit address space is 4G
- Writing 4G to disk takes ~1G/sec or 4 seconds
- Times slices are MUCH smaller than 1 second
- You would spend 99.9999% of the time reading/writing memory!

Solution: Stay Tuned

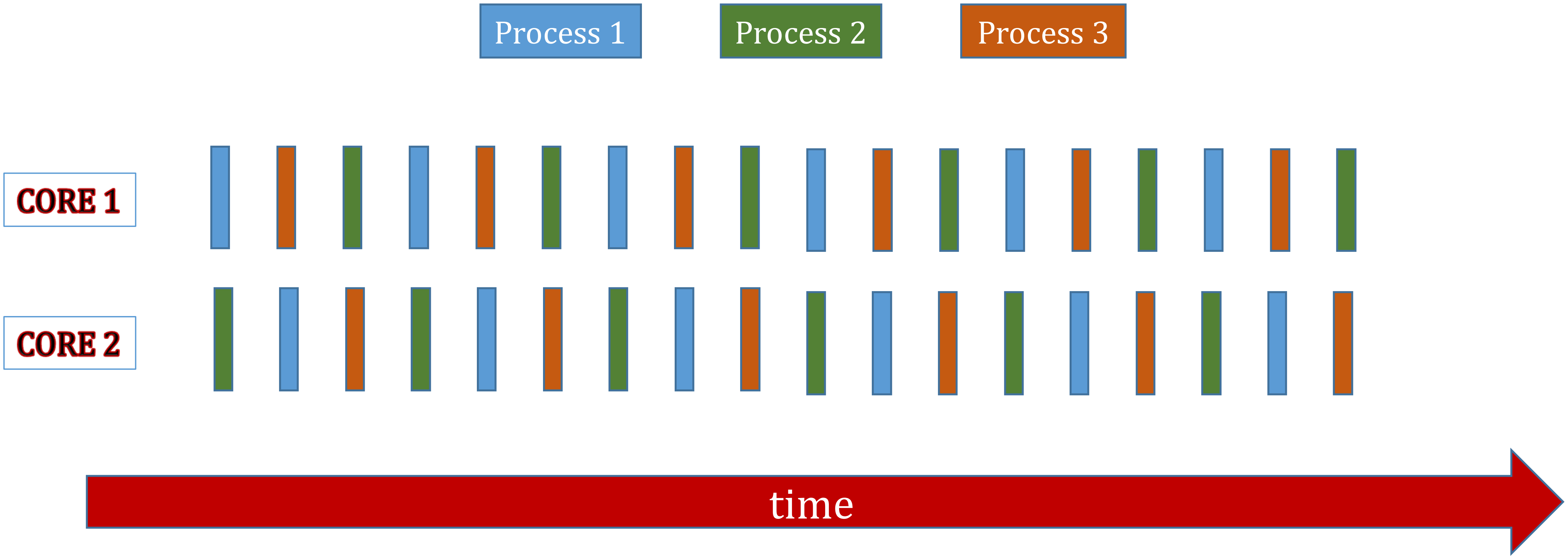
Process Swapping and Interrupts

- Often a process must request a resource, and wait until that resource is available
  - E.g. request a disk read, and wait until that data is available
- Process cannot use the processor while it is waiting
- Identified as an "idle" process
- Idle processes swapped out, and kept off of the process queue
- When request is satisfied, an "interrupt" occurs to wake up that process
- When a process wakes up, it goes back on the process queue

Multi-Core Time Slicing

- Allows multiple processes to execute simultaneously
- Each core has it's own unshared resource pool
  - Processor
  - Registers
- Cores may share common resources
  - Disk
  - Monitor
  - Memory
- If fewer processes than cores, some cores stay idle (rare)
- If more processes than cores, processes swapped in and out of next available core (swap-out time smaller)

Multi-core Time Slicing



Time Slicing Pro's and Con's

Advantages

- Fair resource sharing policy
- Enables multiple (seemingly) concurrent processing

Disadvantages

- Context swapping overhead – loss of efficiency