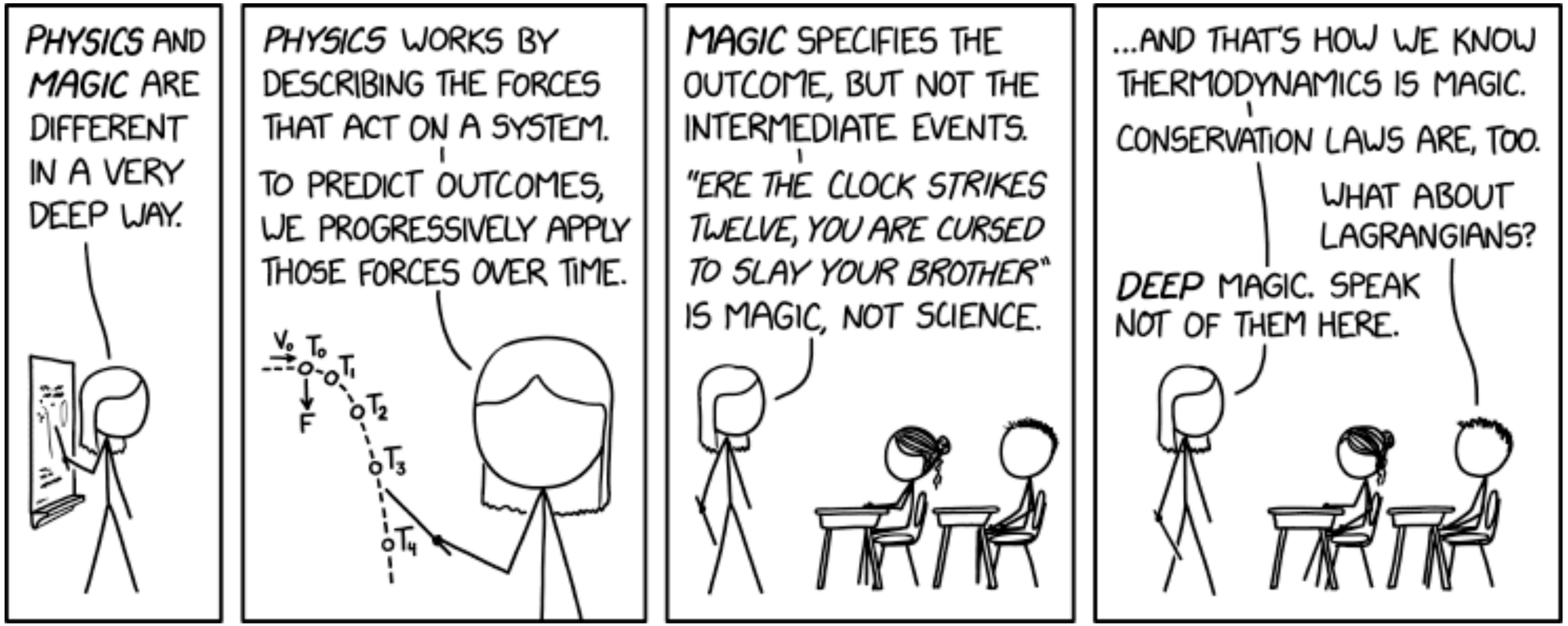
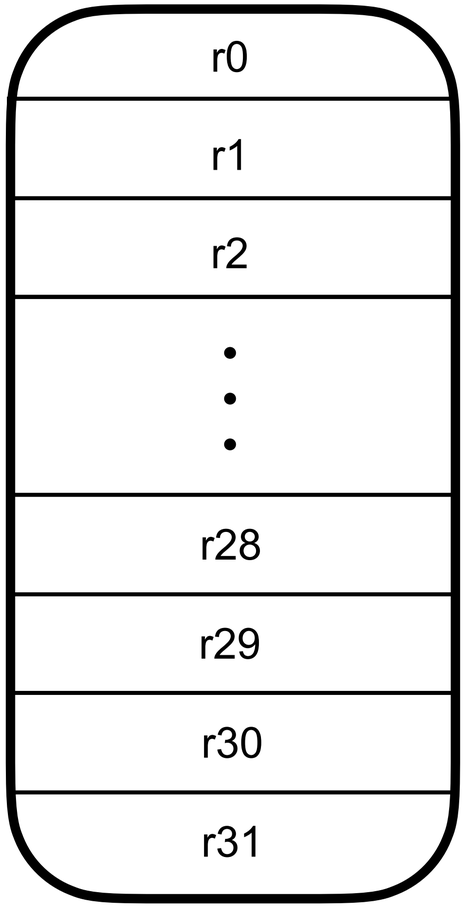
The Magic of Systems



How Do We Multitask?

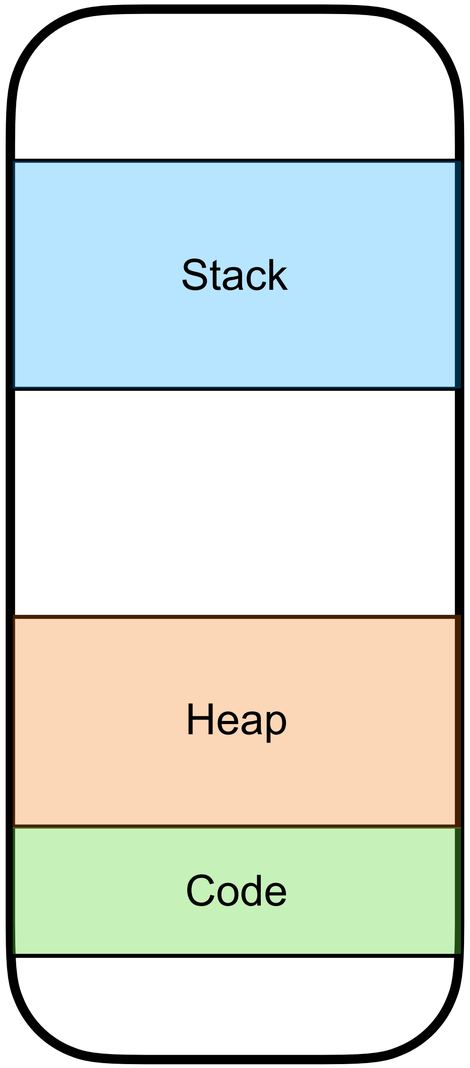
What is a Process?

What is a Process?



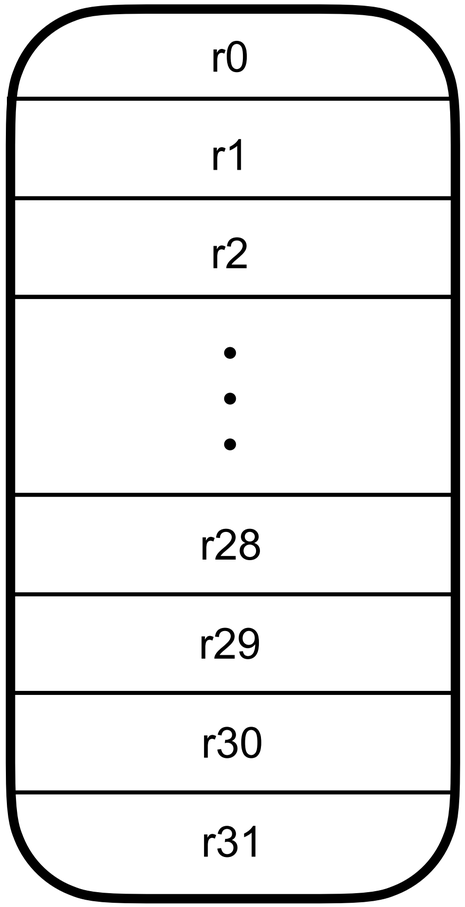
Registers

+

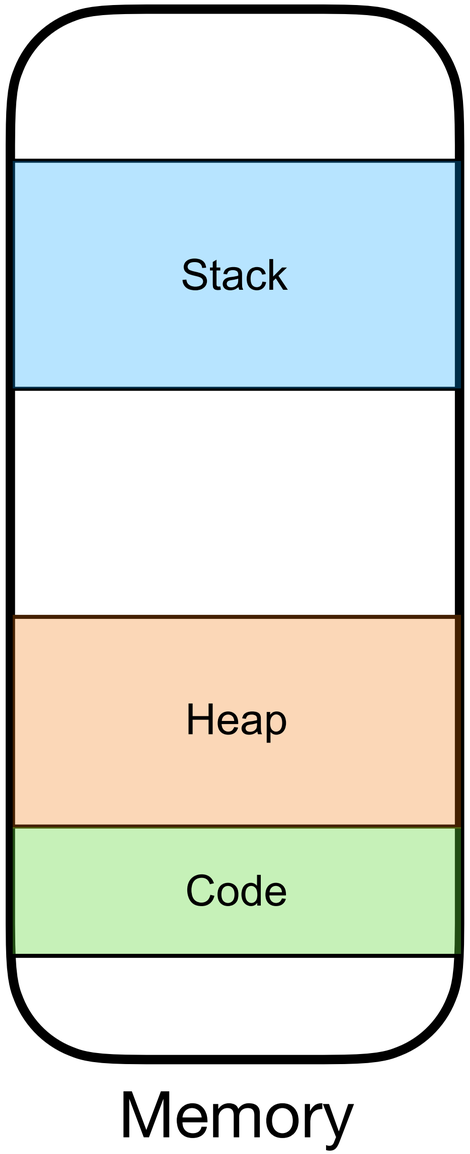


Memory

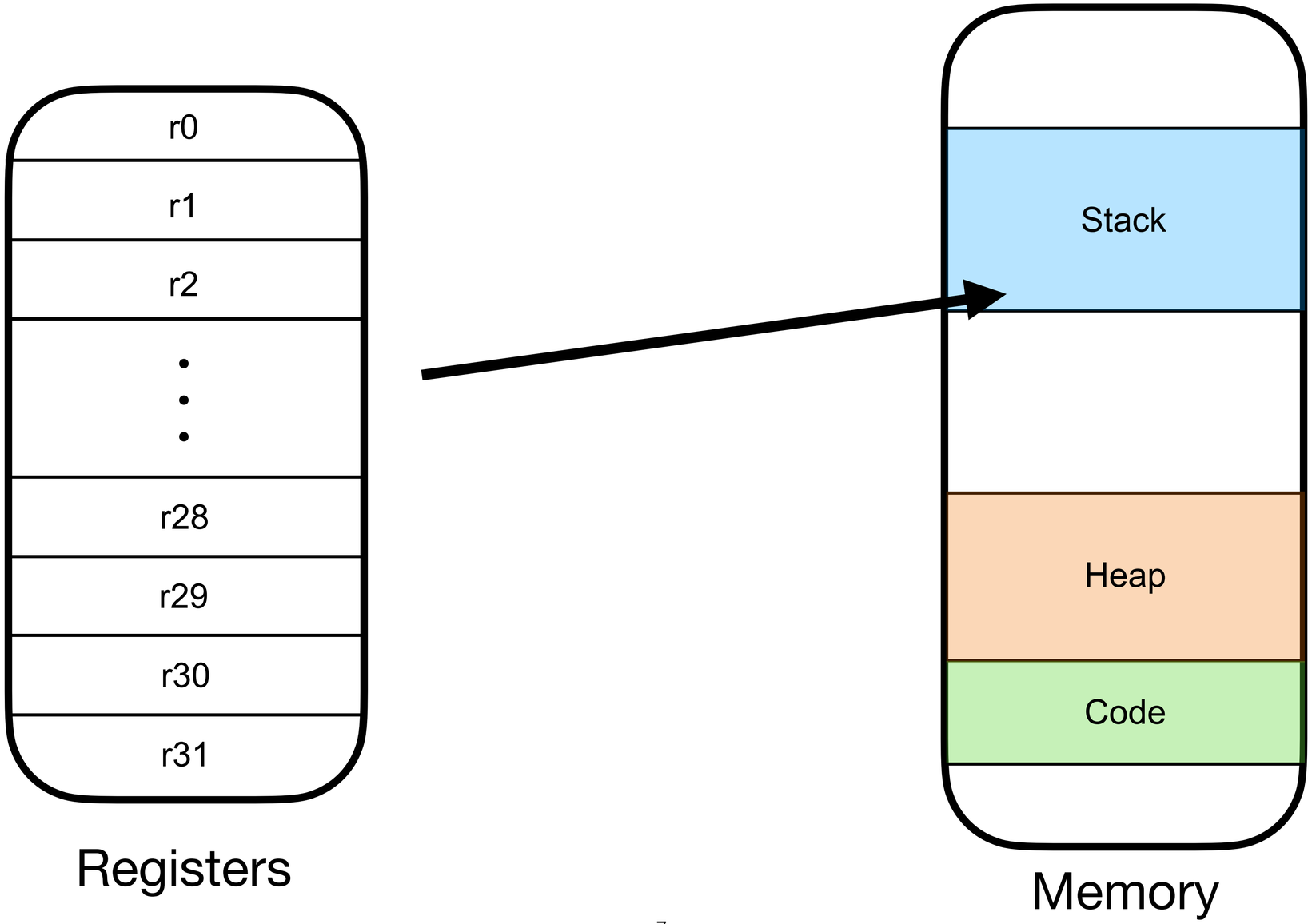
Execution Context

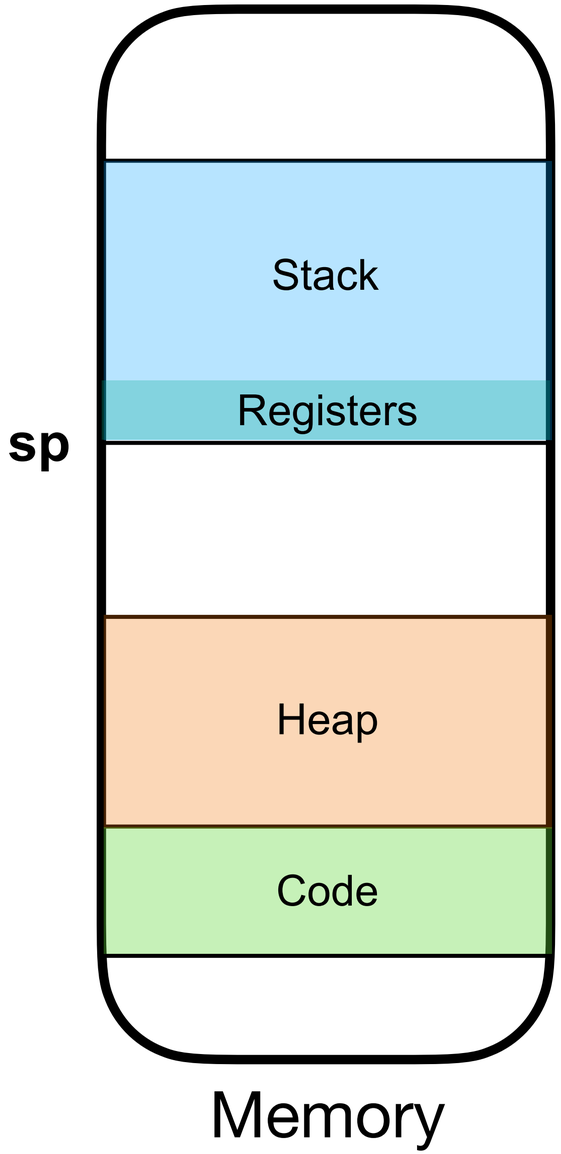


+



we want to “snapshot” our process


r0
r1
r2
r28
r29
r30
r31
Registers
Stack
Heap
Code
Memory


Stack
Registers
sp
Heap
Code
Memory

Process Control Block

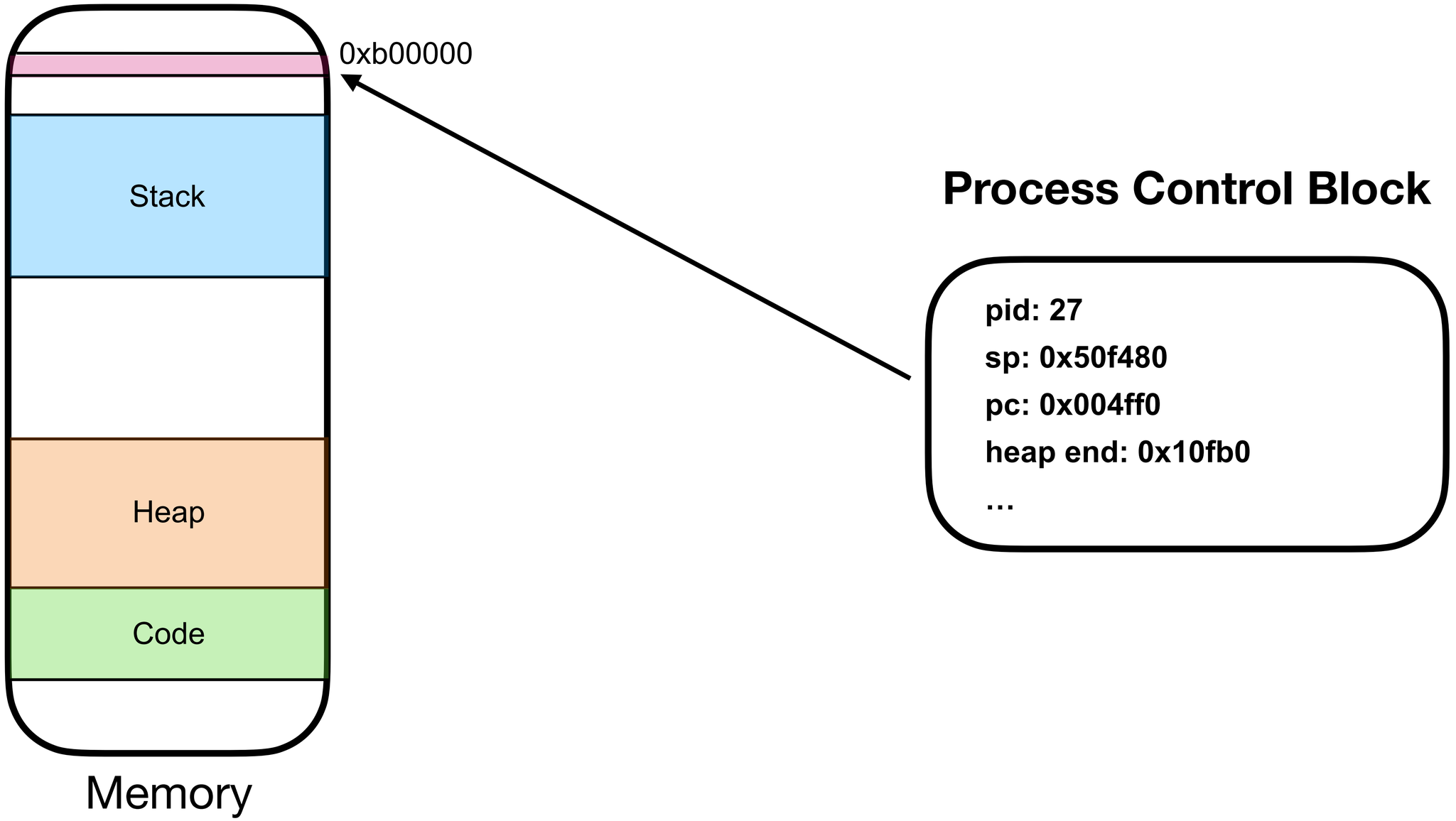
pid: 27

sp: 0x50f480

pc: 0x004ff0

heap end: 0x10fb0

…


0xb00000
Stack
Heap
Code
Memory
Process Control Block
pid: 27
sp: 0x50f480
pc: 0x004ff0
heap end: 0x10fb0
…

Context Switch

Context Switch

- Tricky to get right
- Has to be written in assembly
- Non trivial overhead - more later

But what if the program doesn’t run context switch?


THIS IS WHY
WE CAN'T HAVE
NICE THINGS

Preemption

- Use a timer interrupt to regain control

- Register context switch as your interrupt handler

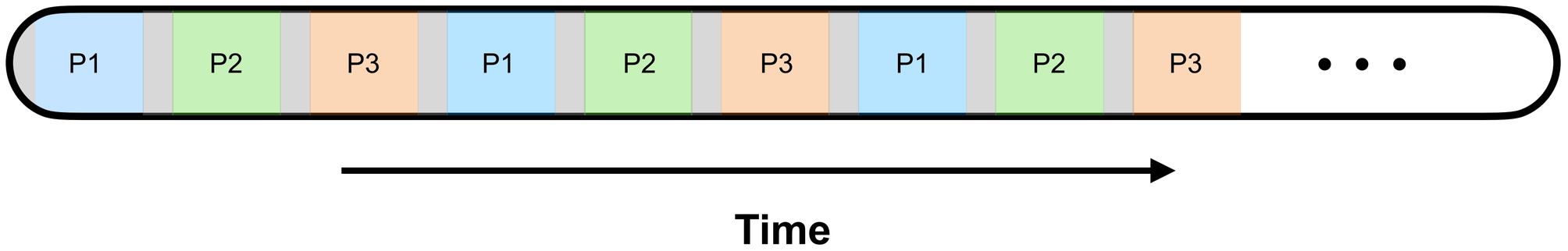
Who do we switch to?

Thread Schedulers

Thread Schedulers

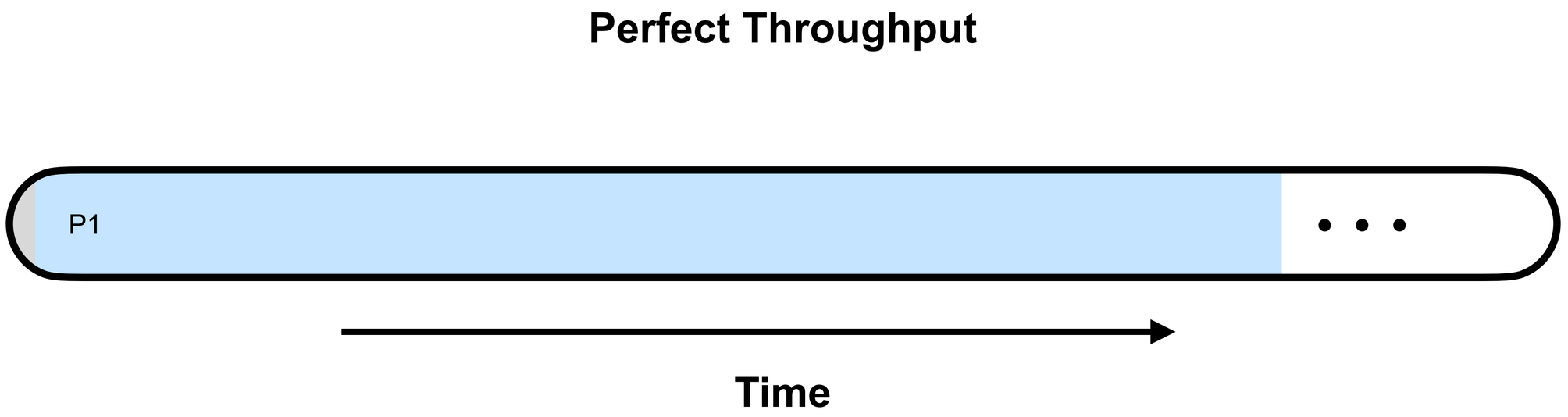
- Not itself a process, just a bit of code to determine who runs next
- Very interesting problem, lots of solutions, needs to be fast
- Latency vs Throughput

Thread Schedulers: Round Robin

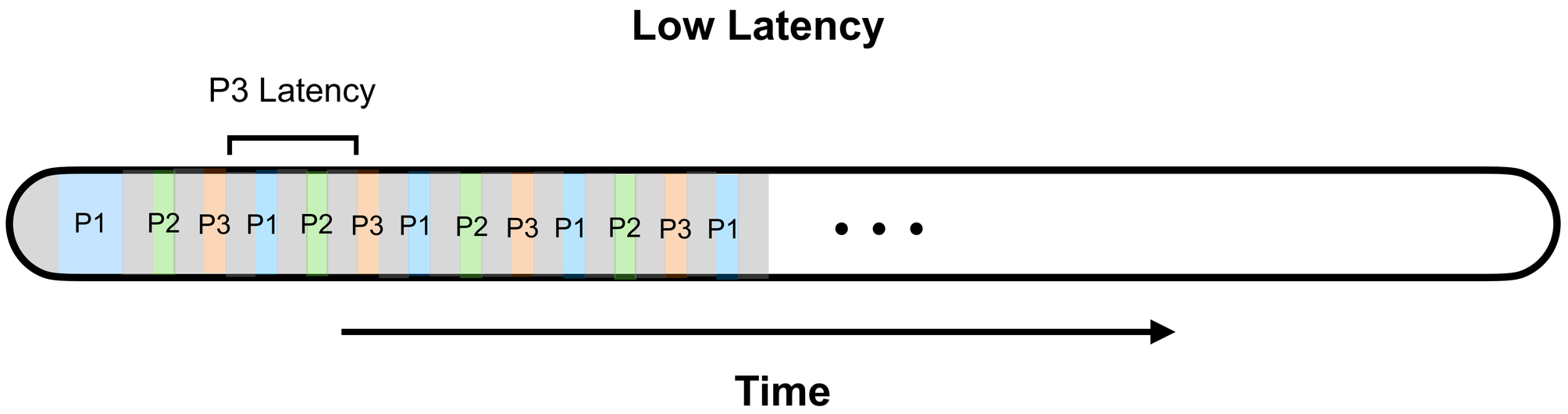


Latency vs Throughput

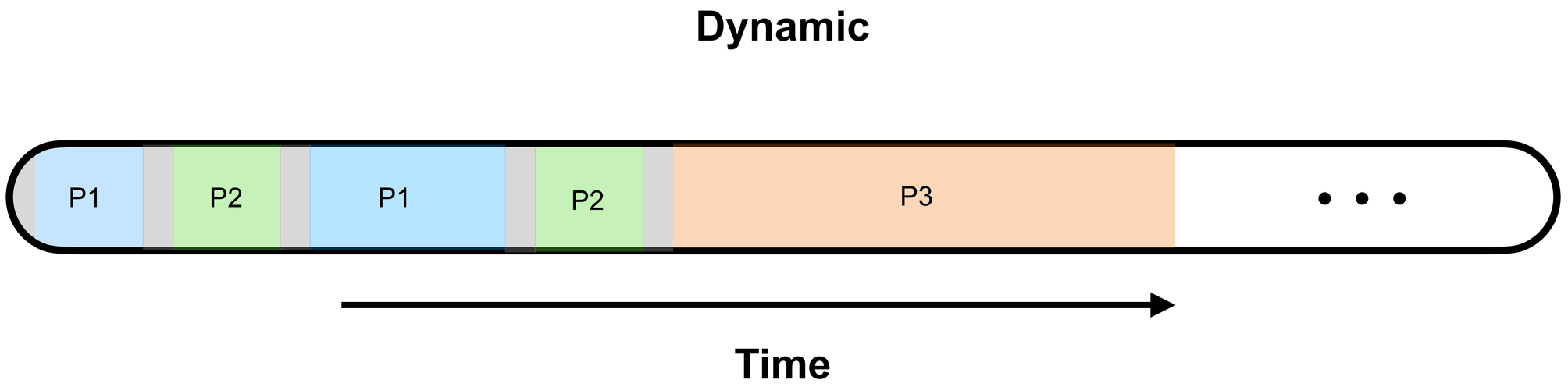
Latency vs Throughput



Latency vs Throughput

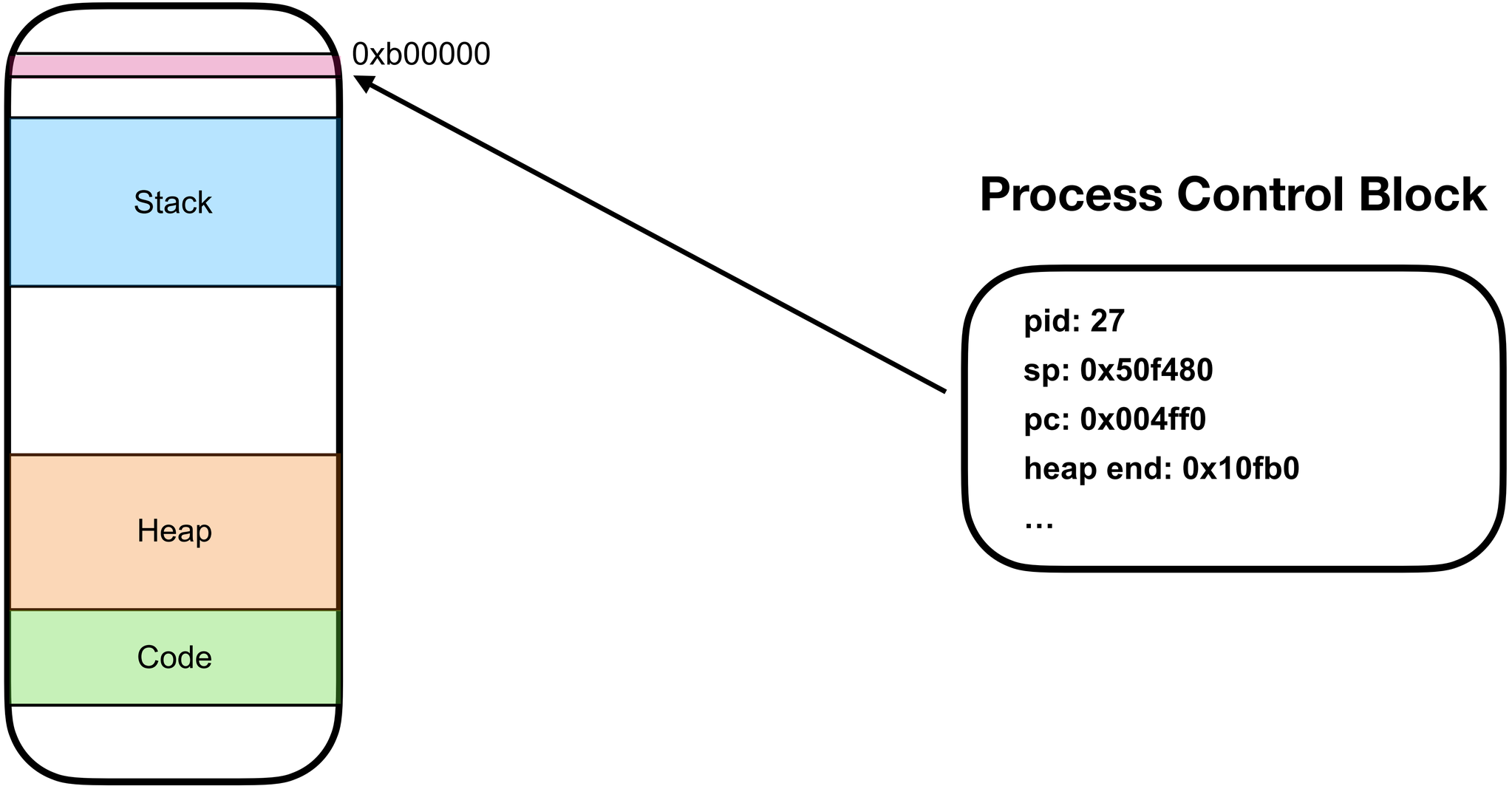


Latency vs Throughput

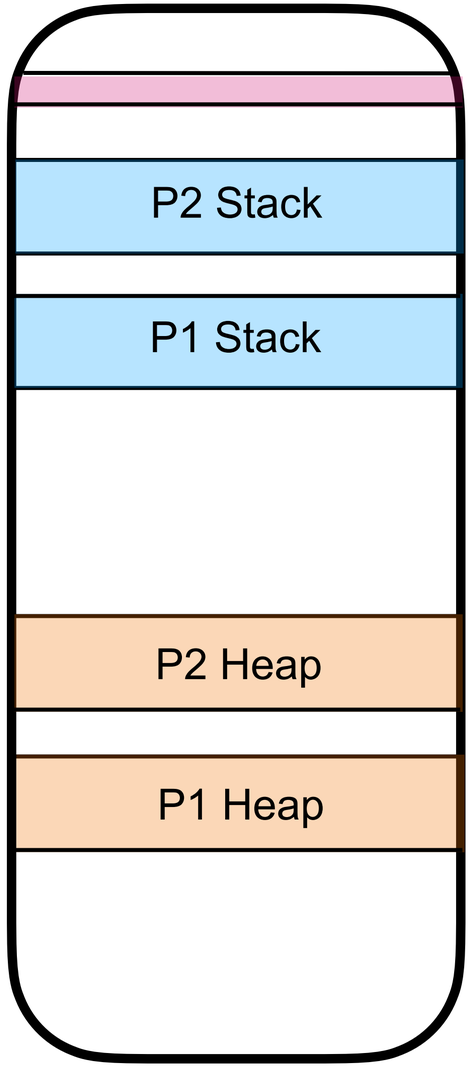


Sorry I lied a bit

Sorry I lied a bit



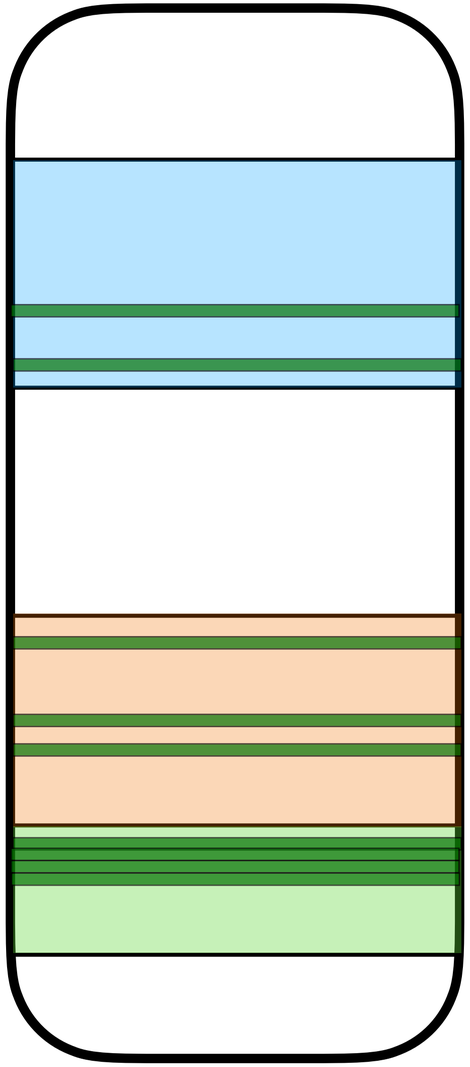
Sorry I lied a bit



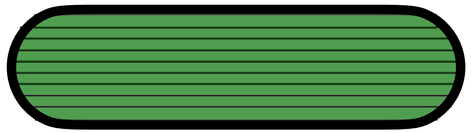
Memory

Lets not support all our programs in the same address space

what memory do we really need?



Memory



Working Set

we want to support only the memory being used

Virtual Memory

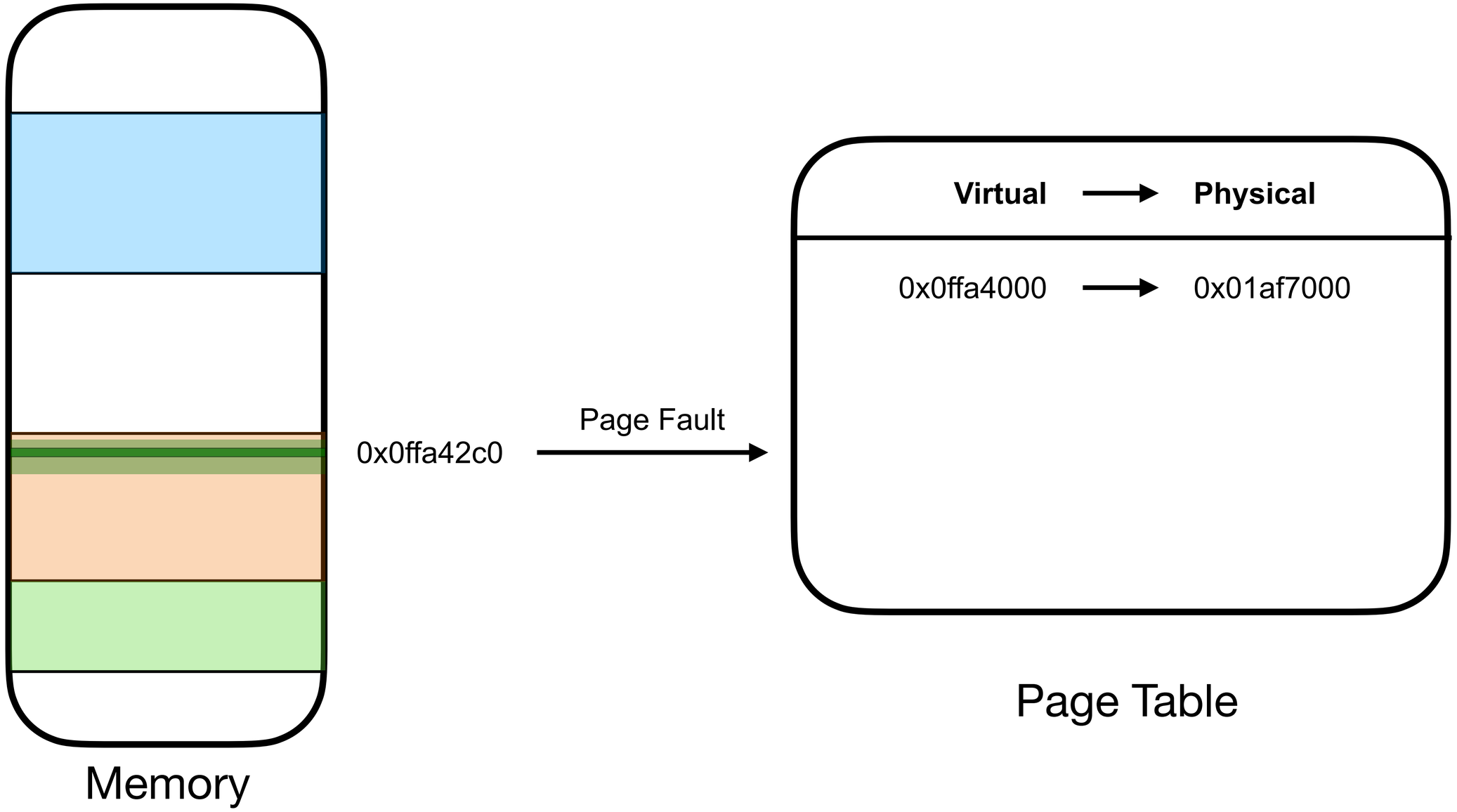
Virtual Memory

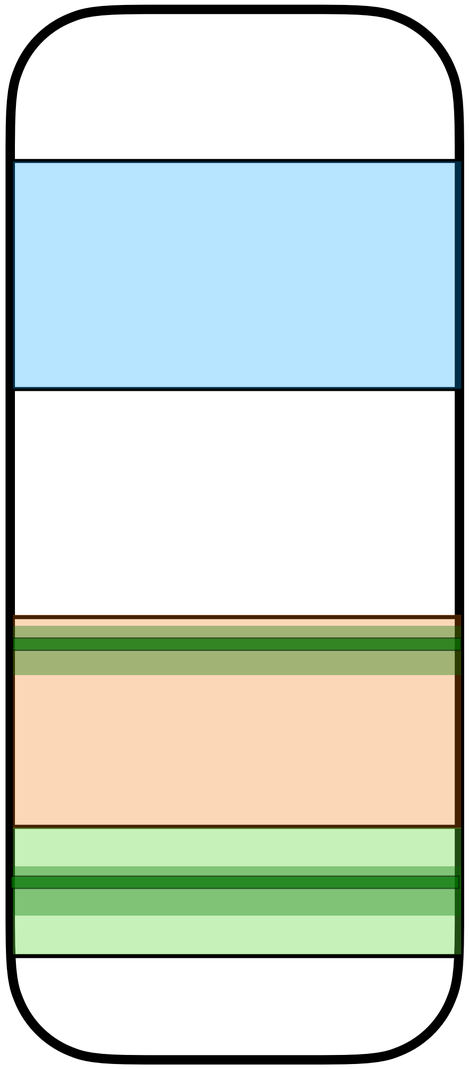
Virtual Memory

- We have two address spaces,
  - the program’s (virtual) and the machine’s (physical)
- We map pages (4KB) of virtual memory to physical pages as needed

Virtual Memory

- We have two address spaces,
  - the program’s (virtual) and the machine’s (physical)
- We map pages (4KB) of virtual memory to physical pages as needed
- Store these mappings in a Page Table, one Page Table per process
- Use interrupts (Page Faults) to service this


Virtual
Physical
0x0ffa4000
0x01af7000
Page Fault
0x0ffa42c0
Memory
Page Table

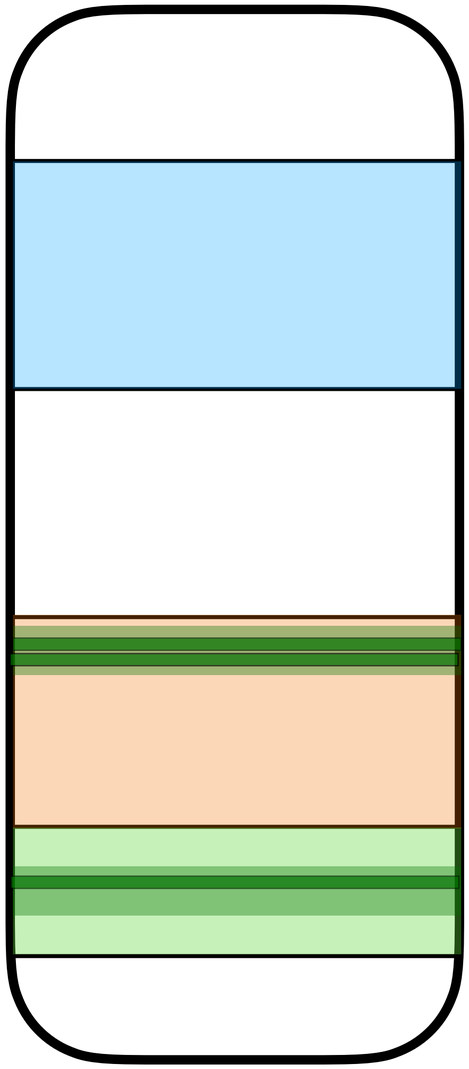


Memory

Page Fault

Virtual		Physical
0x0ffa4000	→	0x01af7000
VA	→	PA

Page Table



Memory

Page Fault

Virtual	Physical
0x0ffa4000	0x01af7000
VA	PA

Page Table

Aren’t all those page faults really annoying?

Translation Lookaside Buffer (TLB)

- Hardware implemented buffer
- Holds recently used page mappings, so you don’t have to take a page fault
- Really speeds things up!
- Needs to be flushed when we swap programs

Translation Lookaside Buffer (TLB)

- Hardware implemented buffer
- Holds recently used page mappings, so you don’t have to take a page fault
- Really speeds things up!
- Needs to be flushed when we swap programs

An aside on caching

- Memory is slow, really far away from processor
- Keep a copy of the most frequently used stuff close to the processor

Pause for Questions

The Abstractions we have so far

The Abstractions we have so far

- Infinite computing resources - multithreading and thread scheduler

The Abstractions we have so far

- Infinite computing resources - multithreading and thread scheduler
- Infinite memory - virtual memory

Creating an OS

Creating an OS

User Space

- -

Operating System

(Kernel)

Creating an OS

User Programs Libraries

User Space

Operating System

(Kernel)

Device Drivers Networking Stack Thread Scheduler File Management

Creating an OS

How do we enforce this line in the sand?

System Calls

System Calls

- The OS’s api to its users. Functions like sbrk, open, fork
- Use software interrupts to ensure correct privileges
  - Machine mode vs User mode

Page Protection

- Our virtual memory provides a powerful abstraction
  - Placing our code between arbitrary code and memory
- Can mark pages as read only, execute only, etc.

I call that a success

Whats Next in the Systems Track