ECE3140 / CS3420
Embedded Systems

Time Sharing

Prof. José Martínez

Cornell University

Time Sharing

- Users often want to run many programs on a system
- Goal: provide an illusion that there is a CPU dedicated to each running program
- How? Virtualize a physical CPU by timing sharing
    - Run one program, stop it and run another, etc.
- Abstraction: 'process' = a running program
    - Abstraction provided to a user
    - Encapsulate the state needed for each program

Outline

- High-level operation
- Process state
  - What needs to be included? Where is it stored?
- OS/scheduler data structures
- Context switch example in ARM
- Memory protection
- Reference for basic concepts
  - “Operating Systems: Three Easy Pieces” (free)
    - Processes: http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-intro.pdf
    - Context switching: http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-mechanisms.pdf
  - Lab 3

Interrupt vs. Context Switch

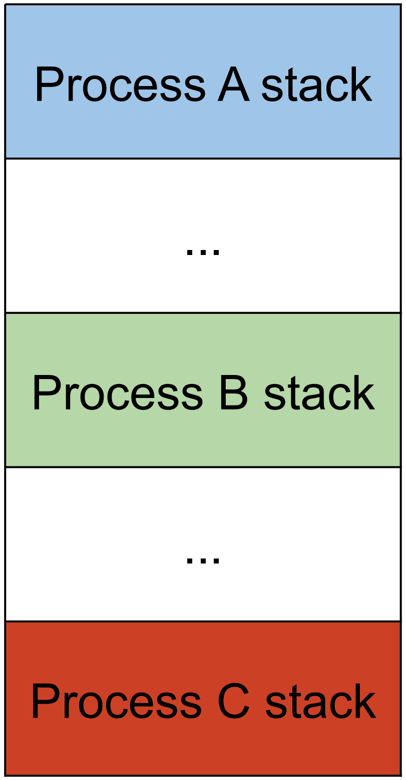
- A context switch is similar to handling an interrupt in terms of saving and restoring process state
  - But, ISRs did not have a notion of multiple processes

Process State

- What does a process need to run? Where is the state stored?

Per-Process Stack

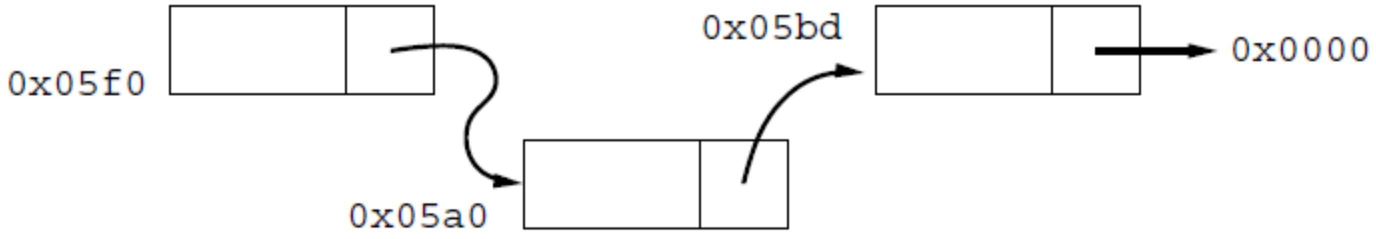
- Separate stack per process
- Process state in PCB
  - SP
  - (scheduling state)
- Rest of state
  - Saved in the stack



Process Queues

- OS/scheduler maintains a queue of processes
  - Often, a separate queue for each scheduling state

```
struct queue {
   process_t *p;
   struct queue *next;
};
```



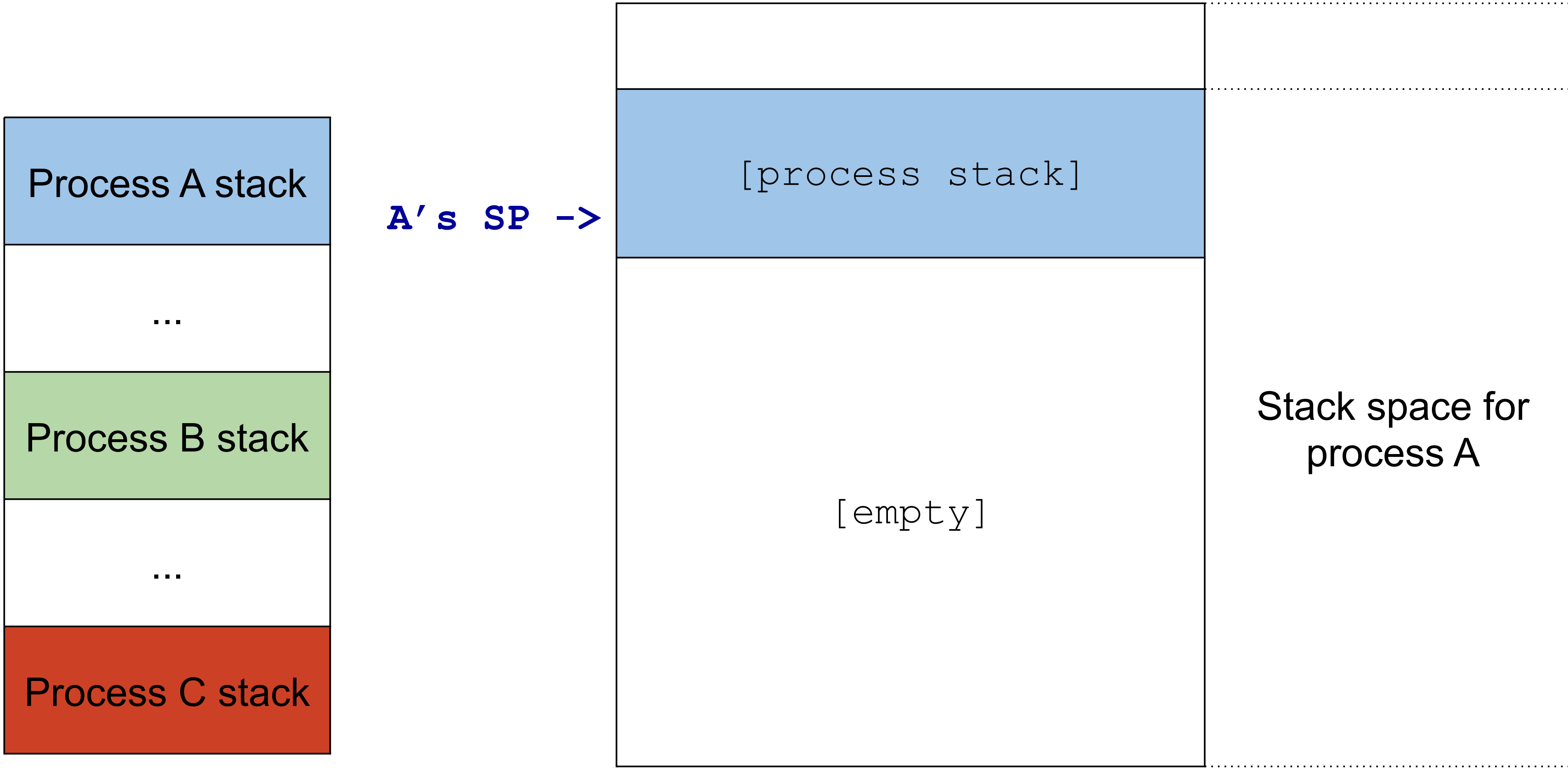
Process Scheduling State

- A process could be:
  - ready
  - waiting/suspended/blocked
  - running

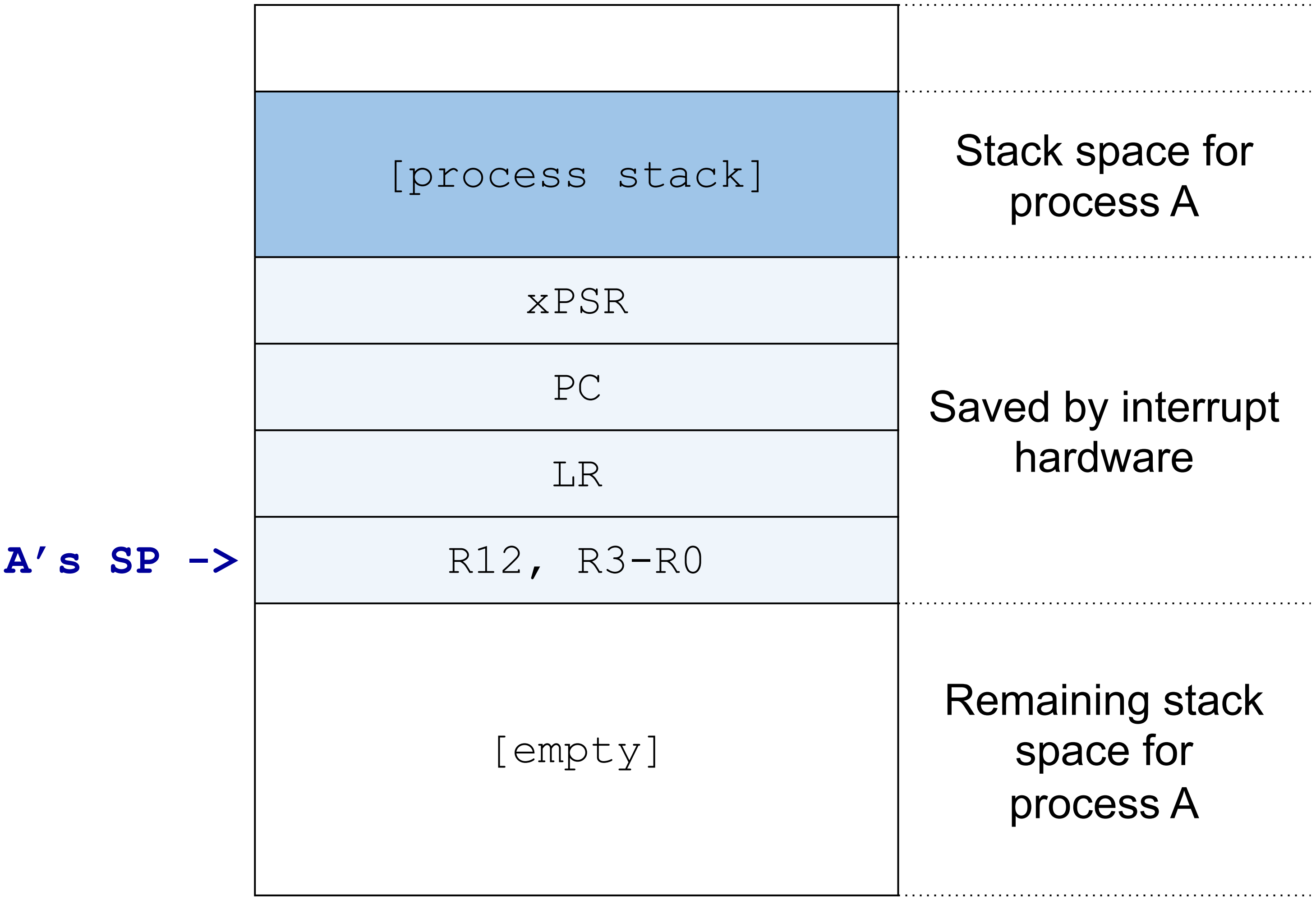
Process Control Block (PCB)

- Need data structures to keep track of processes (process queue) and information on individual processes (PCB)

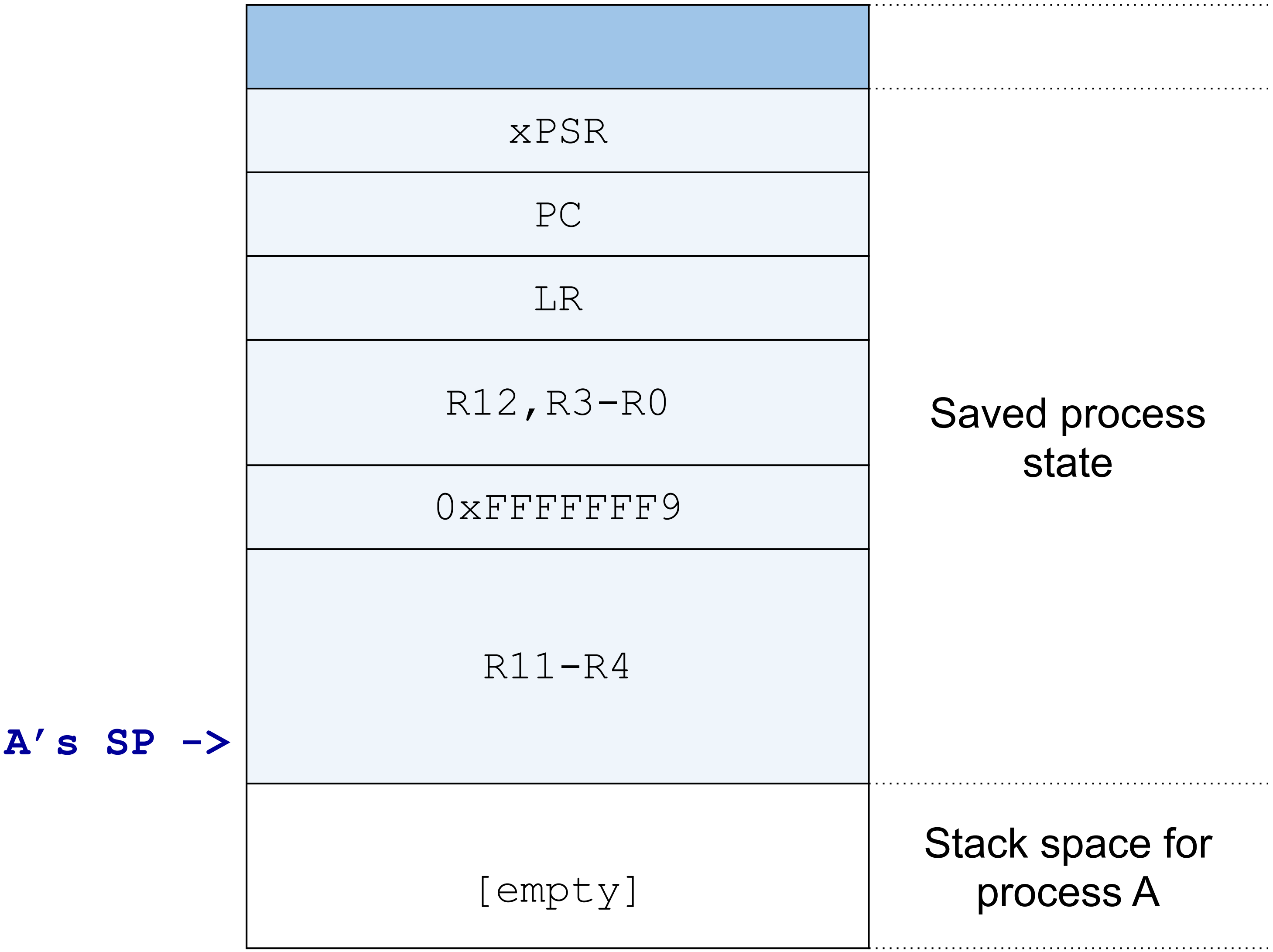
Context Switch Example



Timer Interrupt

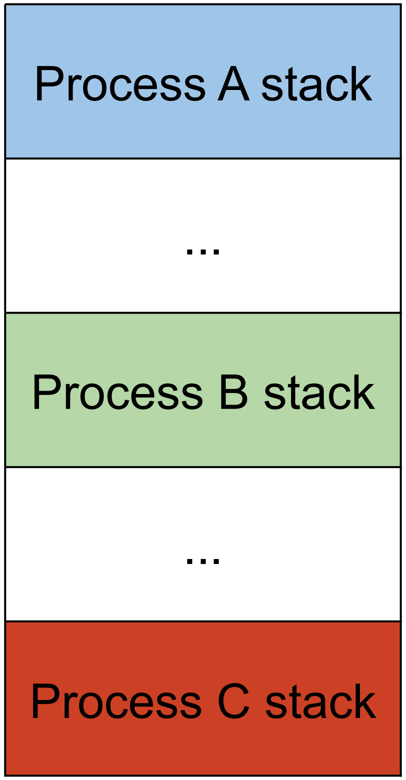

[process stack]
xPSR
PC
LR
R12, R3-R0
[empty]
A's SP ->
Stack space for process A
Saved by interrupt hardware
Remaining stack space for process A

Interrupt Handler Saves Registers

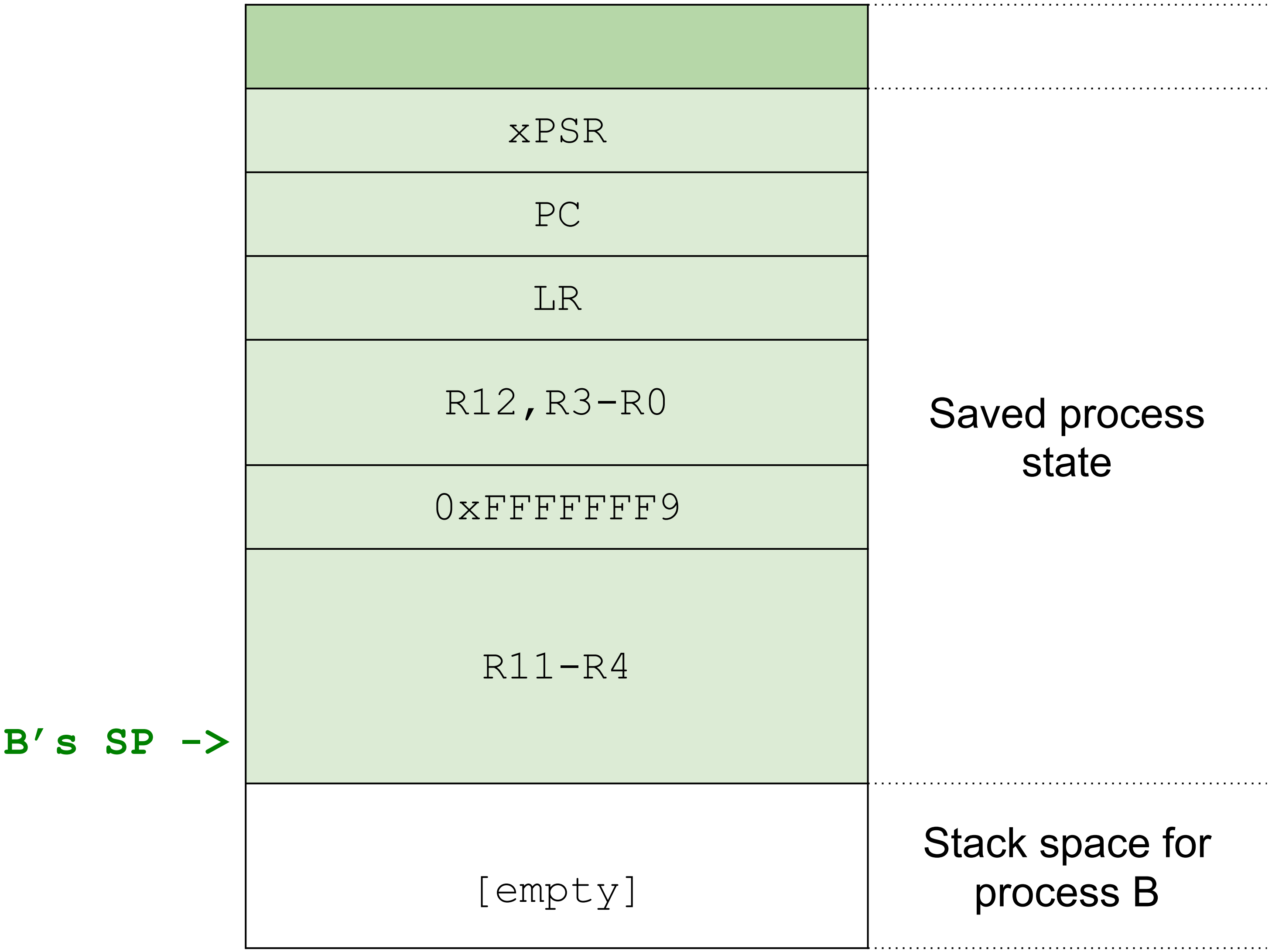


Switch the Stack Pointer

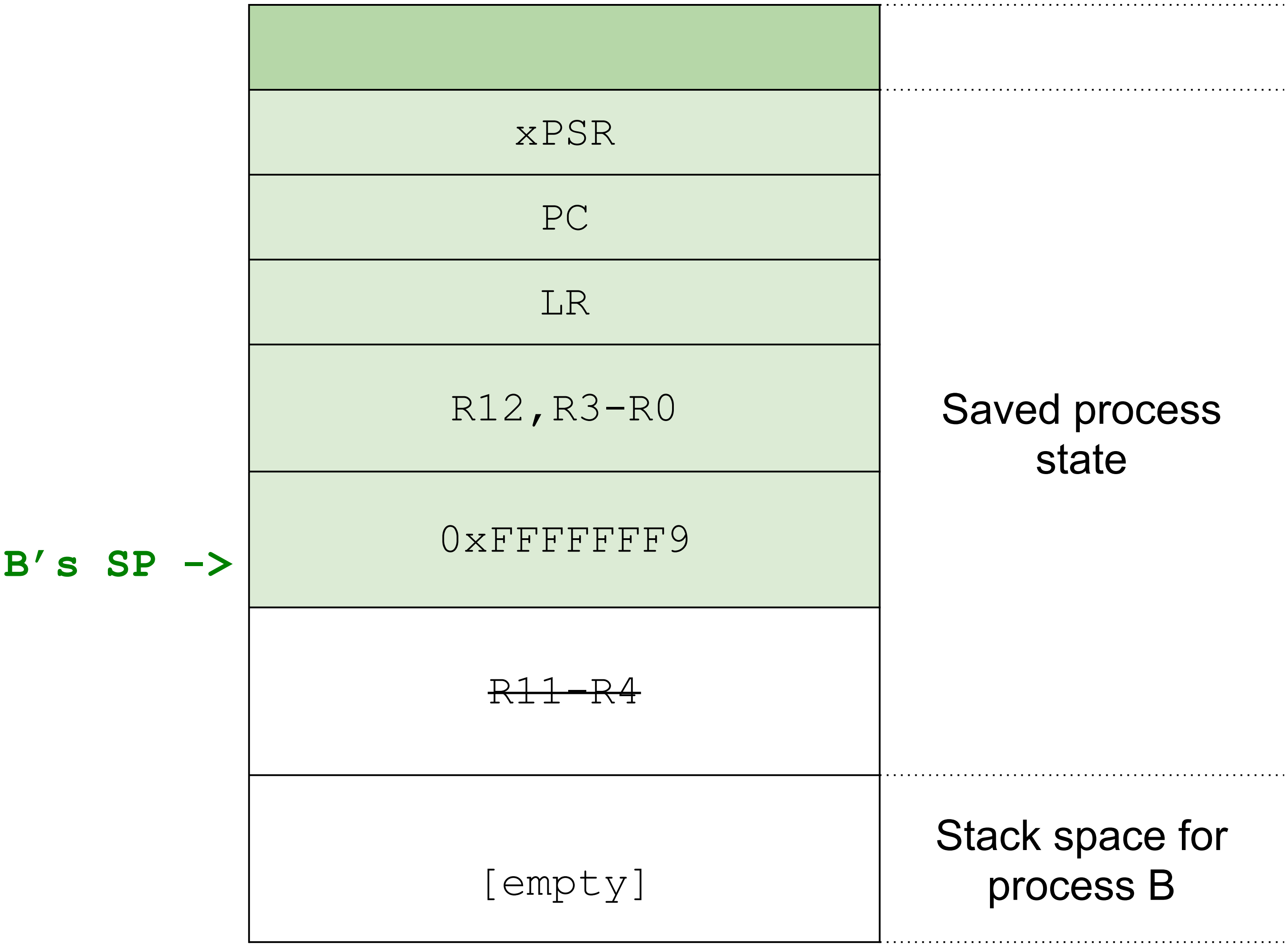
- Select the next process to run
  - For example, Process B
- Save SP for Process A
  - Put SP in A's PCB
- Set SP for the next process (B)
  - Read SP from B's PCB
- A CPU will use Process B's stack going forward



Switch the Stack Pointer



Restoring Registers



xPSR

PC

LR

R12,R3-R0

0xFFFFFFF9

B's SP ->

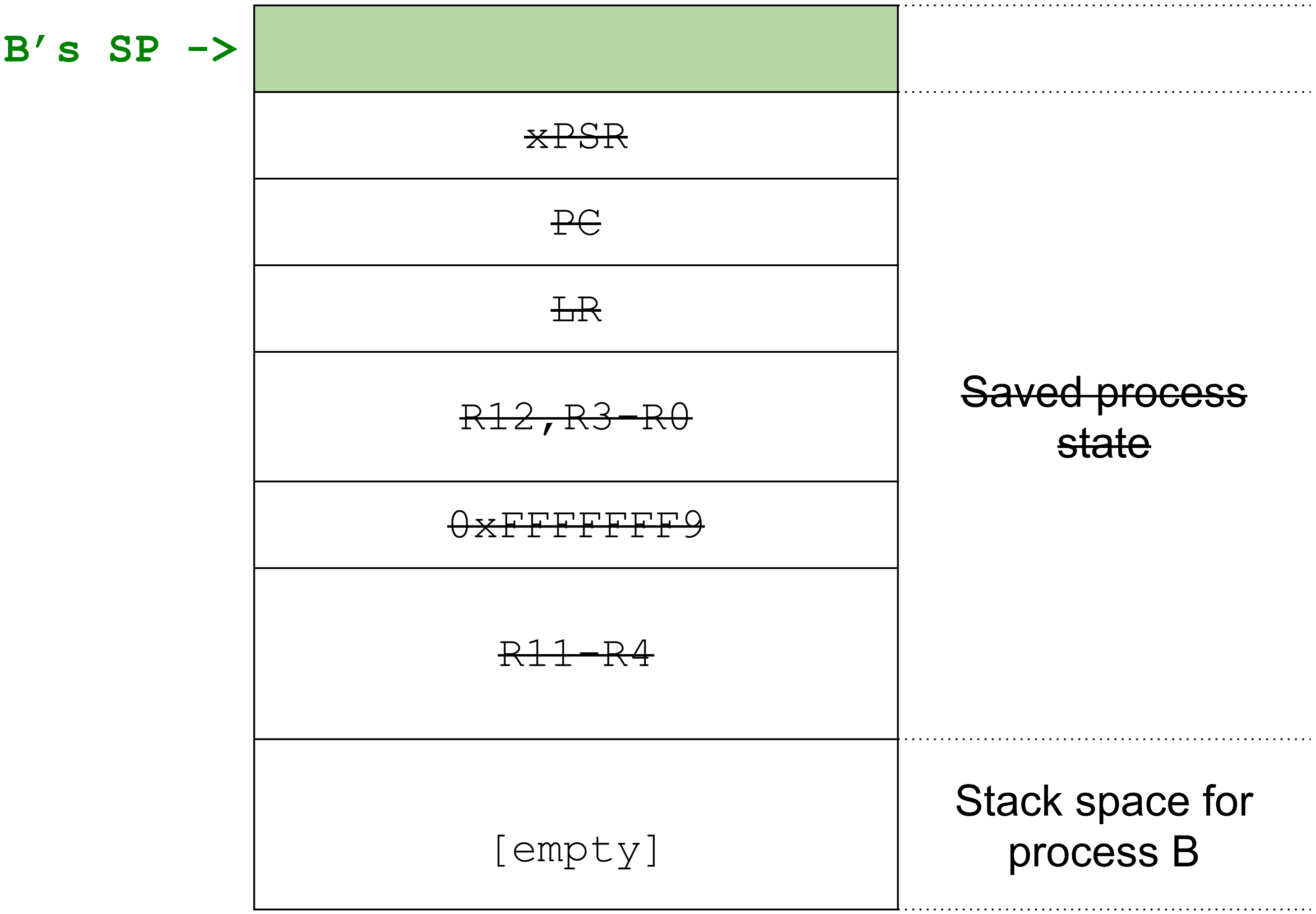
~~R11-R4~~

[empty]

Saved process state

Stack space for process B

Return-from-interrupt



B's SP ->

~~xPSR~~

~~PC~~

~~LR~~

~~R12, R3-R0~~

~~0xFFFFFFF9~~

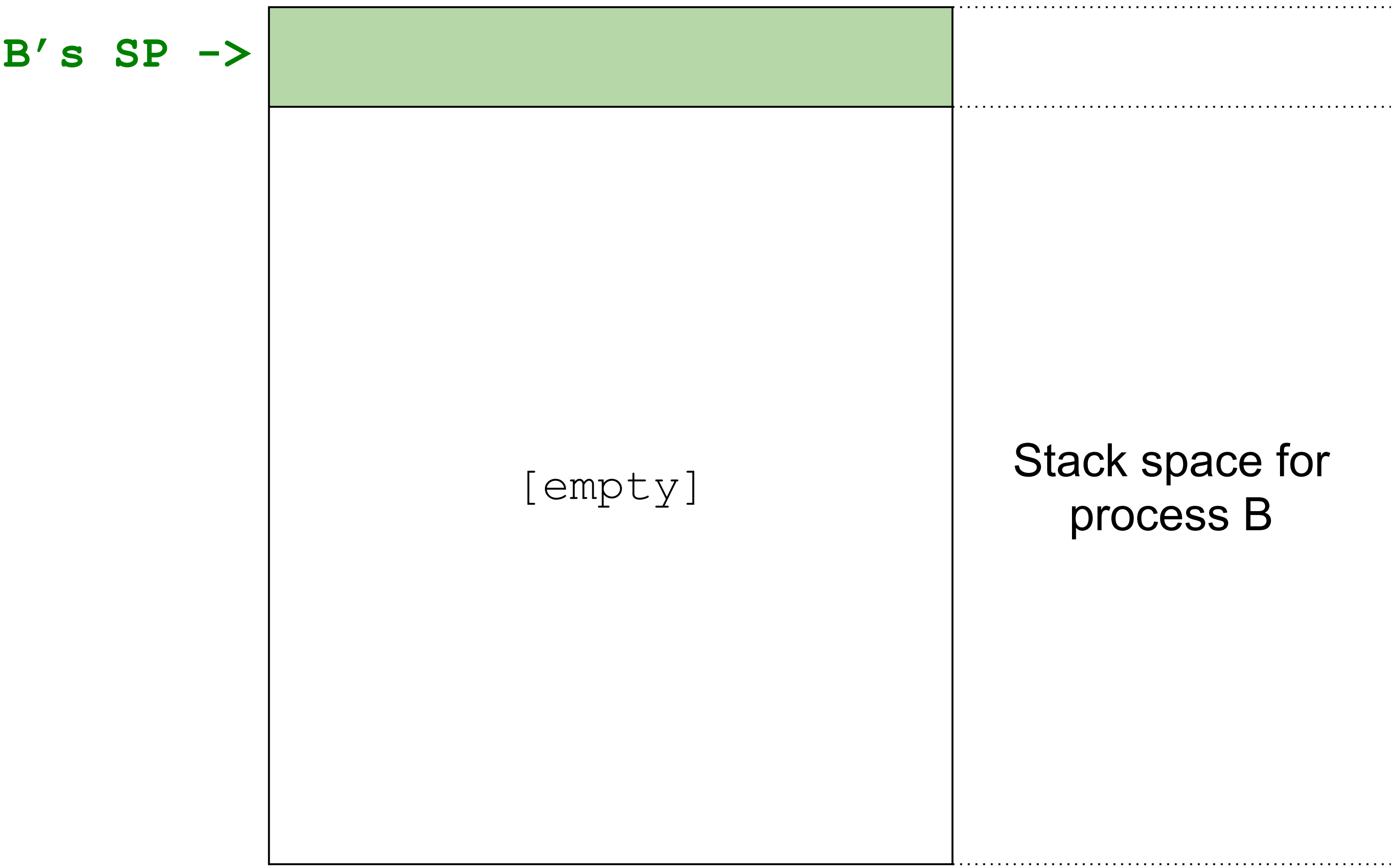
~~R11-R4~~

[empty]

~~Saved process state~~

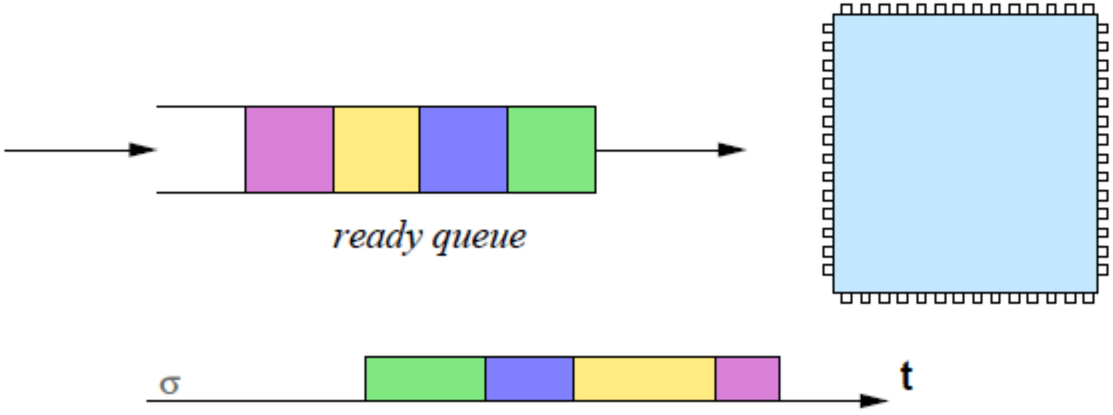
Stack space for process B

Process B Starts Running



Simple Scheduling Policy

- First-Come First-Serve (FCFS)

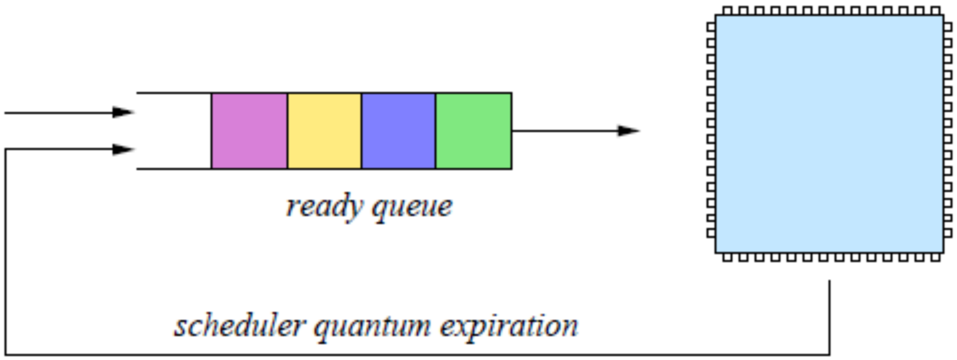


- Non-preemptive
  - Each programs runs until it voluntarily gives up a CPU
  - Also called cooperative multi-tasking
- What if a program is malicious or buggy?

Round-Robin Scheduling

- Round Robin (RR):
  - The ready queue is FCFS

- However . . .
  - A program cannot execute more than Q time units, often called a time quantum
  - When Q time units have elapsed, the program is interrupted and is put back into the ready queue → Preemptive scheduling



- More on scheduling algorithms later

Memory Protection

Virtual Memory (Concept)