Project 1

- Executing multiple PB12 processes with preemption.

- Requires modifications to both the hardware and software components of VM.

  - Beginning to implement Operating System functionality.

Main Memory

- Currently, system can only execute one program at a time.

- The program is loaded into physical memory locations {0 .. 99}, and all addresses generated by the program are from {0 .. 99}.

Main Memory

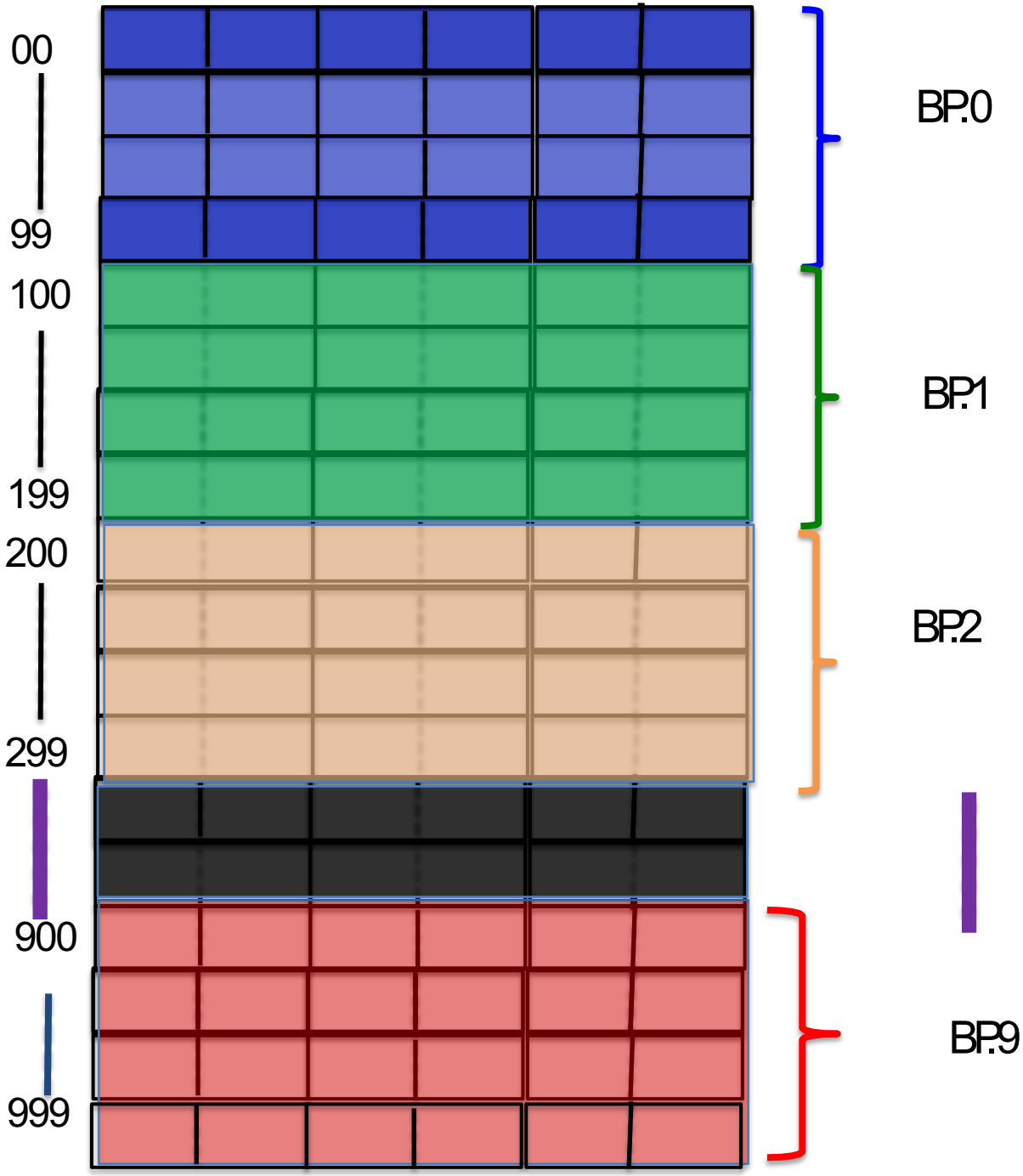
- In Project 1, we now have 1000 x 6 byte memory that can hold up to 10 PB12 programs.

- Each program will be loaded into its own 100-word partition.

- Individual PB12 processes still generate addresses in the range of {0 .. 99}. Now these are termed *logical addresses*.

Main Memory

- Each program will be loaded into its own 100 memory word partition.

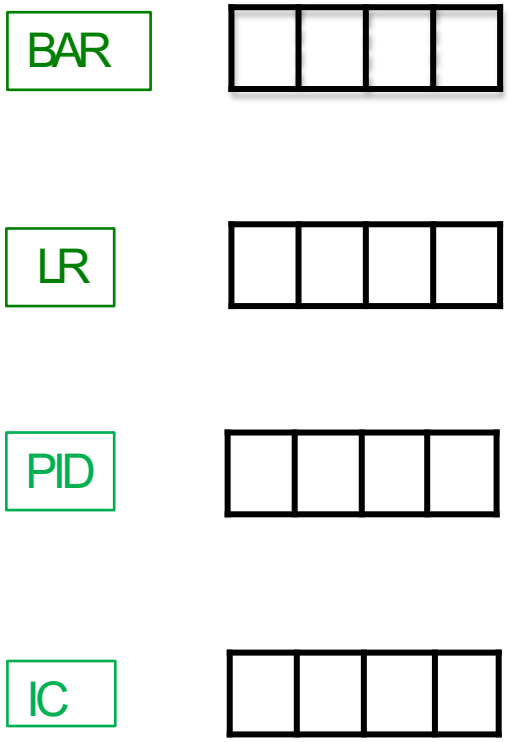

00
99
100
199
200
299
900
999
BP.0
BP.1
BP.2
BP.9

- Individual PB12 processes still generate addresses in the range of {0 .. 99}.
  - termed logical addresses because do not necessarily correlate with actual physical addresses {0 .. 99}

- Requires a Memory Management Unit (MMU) that translates between logical and physical addresses.

- MMU is a hardware unit. Much too slow to perform in software.

New Hardware for our Virtual Machine



Base Address Register: Holds **base address** of the currently executing process
Minimum legal address.

Limit Register: Holds maximum legal address of currently executing process.

Process ID Register: Holds Process ID of currently executing process.

Instruction Counter. Holds time-slice of process.

- The effective address (i.e., actual physical memory location) is

  EA = PC + BAR.

- The Effective Address is used to access physical memory (rather than the PC).

  - note that the PC now holds the *logical*, not *physical* address.

How it Works

- The Base/Limit addresses of each process are determined by the OS when the program is loaded into memory.
    - stored in the hardware BAR/LR when the process is executing, and in the PCB when it is not.
- When a process is executing, your OS must ensure that all addresses it generates are within its legal address space.

Preemptive Multitasking

- Your OS needs to support scheduling and preemptive multitasking
  - Requires Process Control Block (PCB) to maintain state information for a process when it is not executing.
  - Requires implementation of a Ready Queue (RQ), which is a linked list of ready processes (or more precisely their PCBs).
- OS loads in multiple PBRAIN12 programs and multiplexes the CPU between them based on simple scheduling algorithm(s).

struct PCB

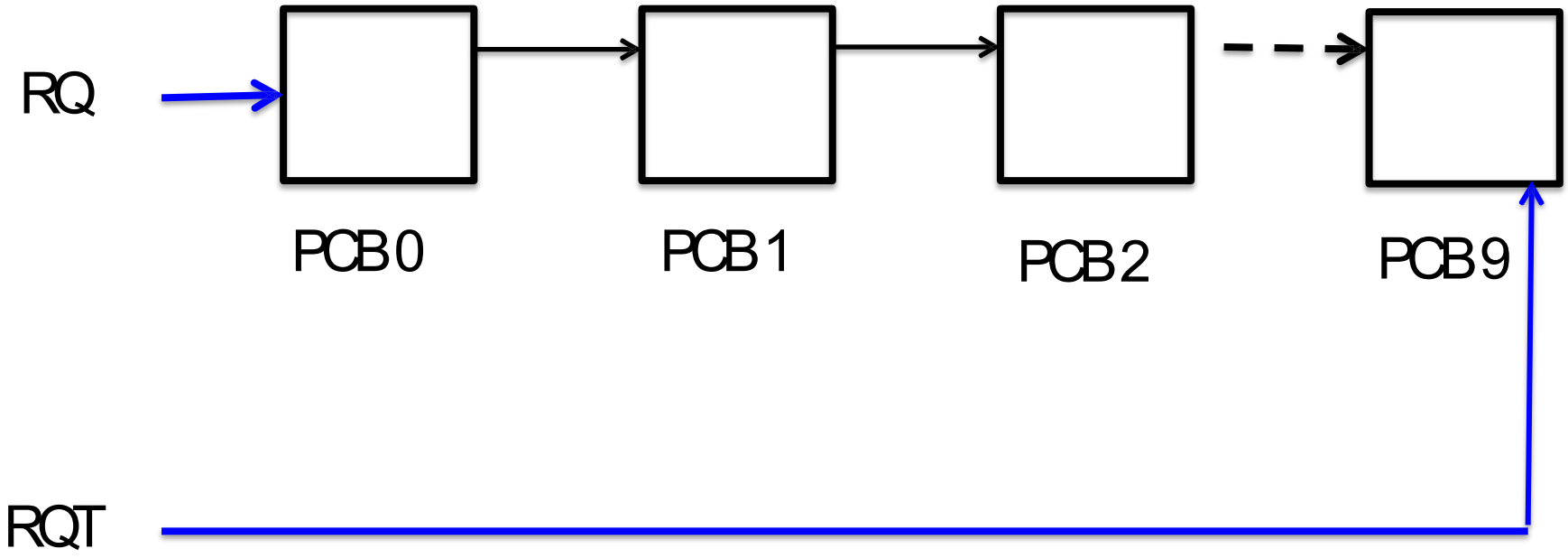
```
{struct PCB *Next_PCB ;          //pointer to next PCB
 int Rregs[4] ;                  //register values
 short  int PC, Pregs[4] ;       //program counter and pointer regs
 int BAR, LR, PID, IC ;          //for multitasking
```



```
} ;
```

OS maintains a Ready Queue with two pointers:

```
struct PCB *RQ ;   //points to the head of the Ready Queue
struct PCB *RQT ; //points to the tail of the Ready Queue
```


RQ
PCB0
PCB1
PCB2
PCB9
RQT

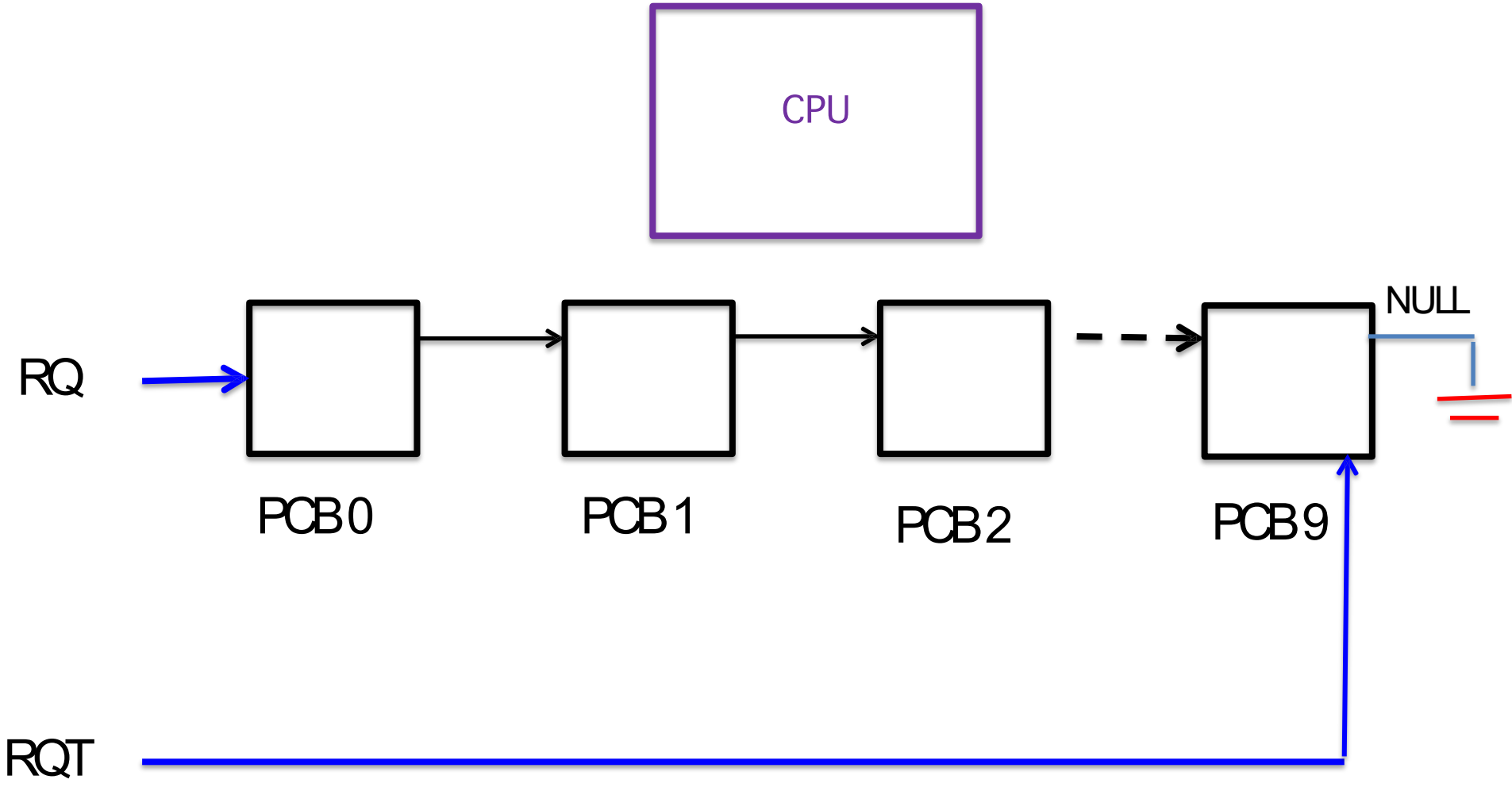
First, Simple RR Scheduling Algorithm

- Your OS implements the Round Robin scheduling algorithm.
  - The process at the head of the RQ is always selected to execute next.

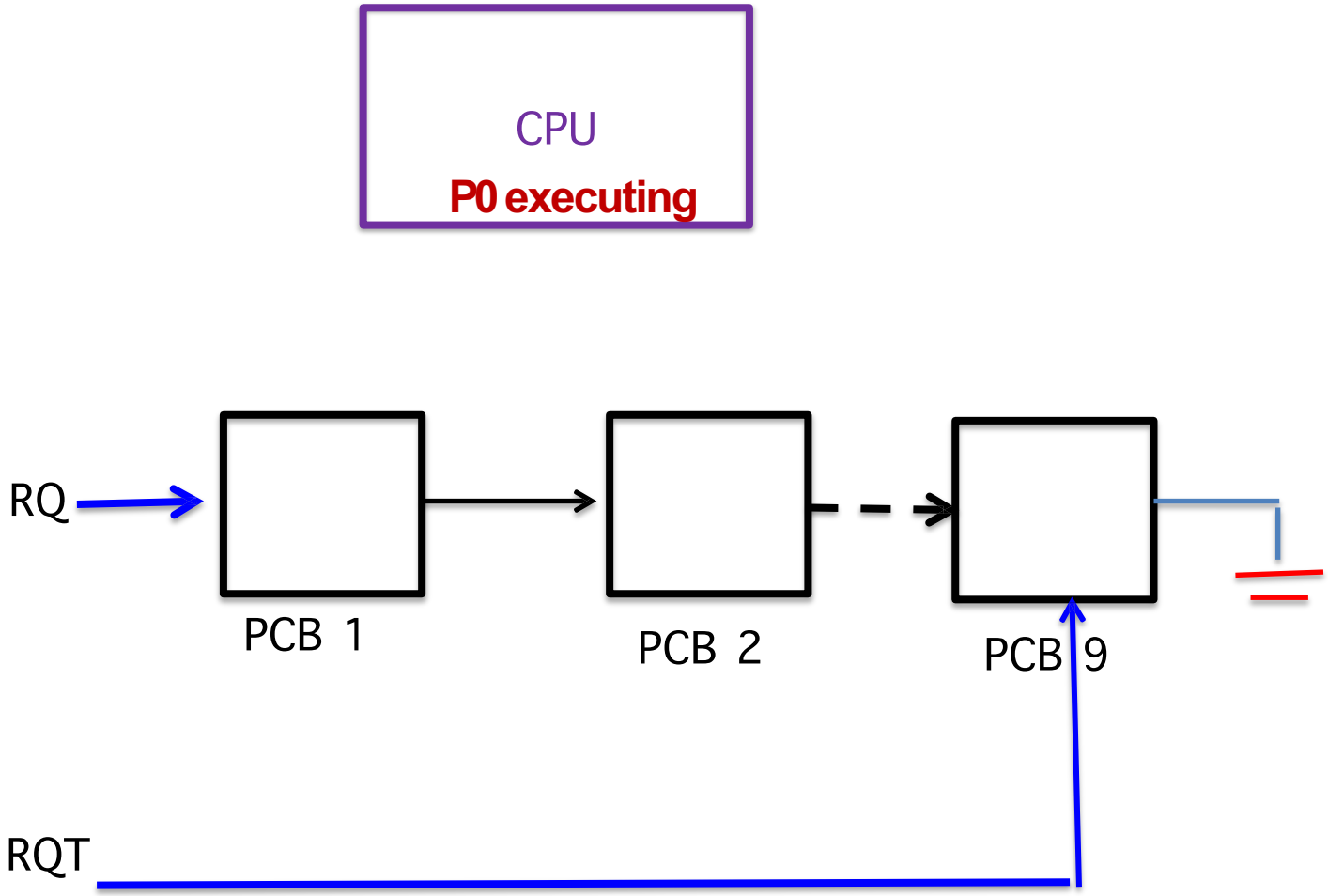
RR Scheduling Algorithm

- Before allowing a process to execute the OS selects a *time-slice*:
  - The length of time the process can execute before being preempted by the OS to give another process a chance to execute.
- Sets a "timer" to generate an interrupt after the time slice has elapsed.
- On the interrupt, your OS saves the state of currently executing process in its PCB, places the PCB at the *tail* of the RQ, and selects the next process to execute.

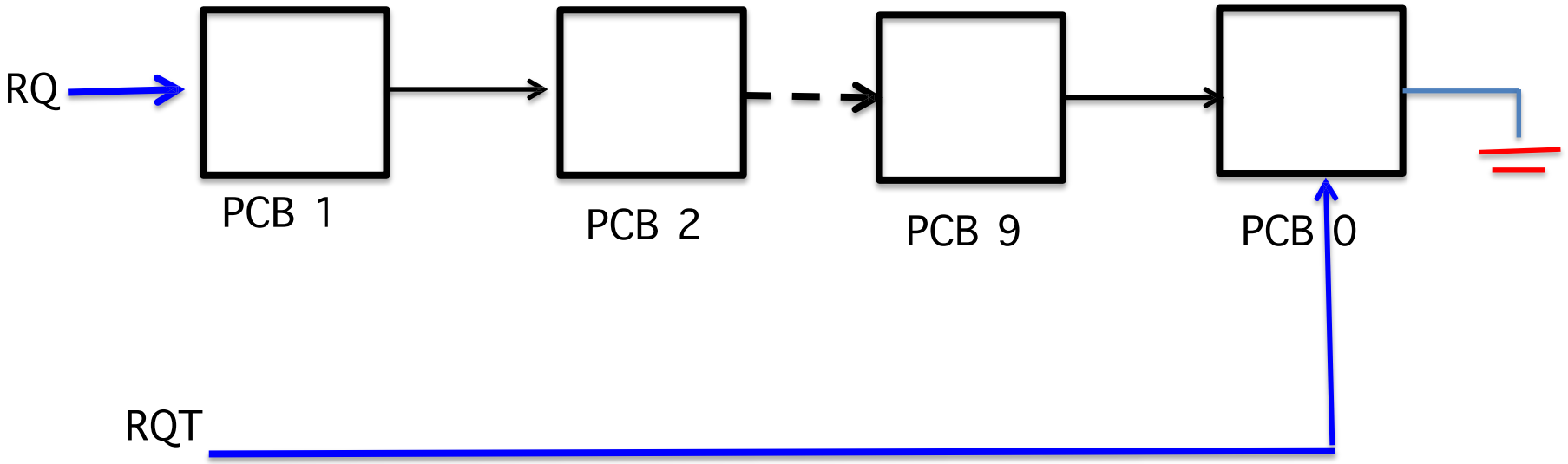
- Use a simple Instruction Counter register (IC) as proxy for the timer.
  - The OS sets the IC to the number of instructions a process can execute before it is preempted.
    - Set to a random value between 8 – 15
  - The IC is decremented by one after each instruction is executerd.
  - When the IC hits 0, the currently executing process is preempted, its state is stored in its PCB, and the PCB is placed at the tail of the Ready Queue.
  - The state of the process at the head of the RQ is then restored and it is given control of the CPU.



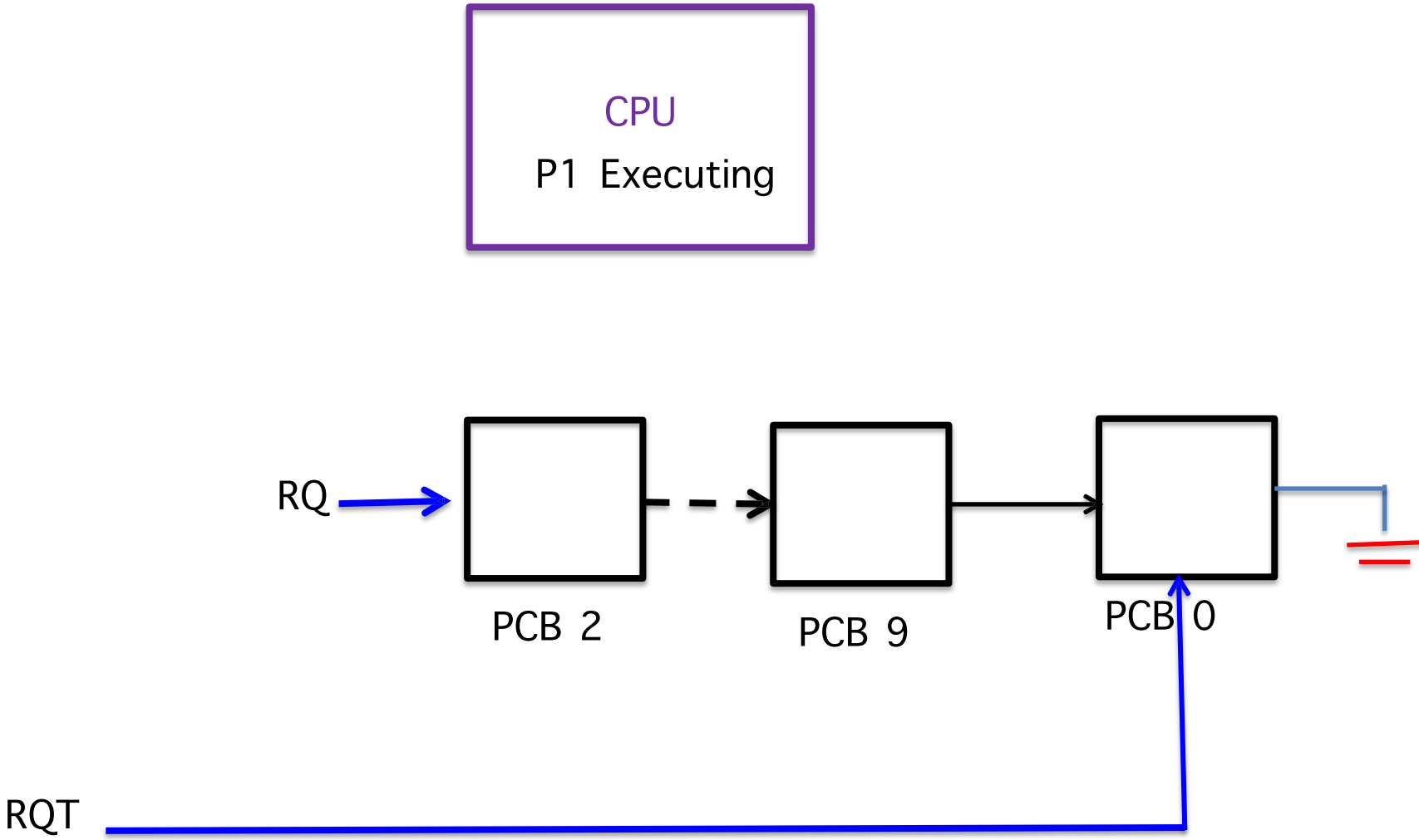
Process 0 selected to execute. Taken off of RQ and given control of CPU.



After the time slice is consumed, the "timer" generates an interrupt. OS places P0 at the tail of the RQ



It then selects P1 to execute, removes it rom the RQ., restores its state, and gives it control of the CPU.



This cycle continues until all processes have finished their program execution.

Requirements

- Provided with a program skeleton and asked to write key functions that implement preemptive multi-tasking.

- Tested with 10 copies of the Fibonacci code, loaded into 10 memory partitions.

- Required documentation as program executes

Documenting Program Flow

- Just before a process is given control of the CPU the OS needs to print out information about the process as follows:
  - Process (Process ID) ready to begin execution (PID assigned as programs are loaded into memory, where program 0 is given PID 0 and so forth).
  - It has a time slice of x instructions.
- As the process is executing, it should call the PrintRegs function to document execution flow. The PrintRegs function should be updated to include the Process ID, Base/Limit Registers, ***and number of instructions remaining in time-slice***.

Documenting Program Flow

- When the IC hits 0, the OS must print out a message to the effect:
  - Process (PID) completed time slice. Placing at tail of the ready queue.
- When a program completes its execution (i.e., executes the HALT instruction), the OS must:
  - print out a message indicating the process is to be terminated.
  - remove its PCB from the RQ,
  - free the PCB
  - Give control of the CPU to the current Head of the RQ.
  - Print the contents of the updated RQ.

- Your project will be tested using 10 copies of the Fibonacci code (to be provided).
- It is due by Friday, October 23rd, 5:00 PM.

- Up to 12 points of extra credit will be given to those projects with output that is very easy to read, clear, well organized, and perhaps even visually appealing.