Operating System

Module-2 Notes

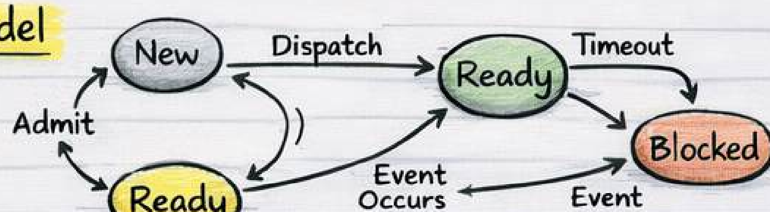
by pyqfort.com

Contents Covered:

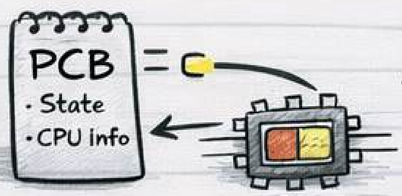
- Program vs Process

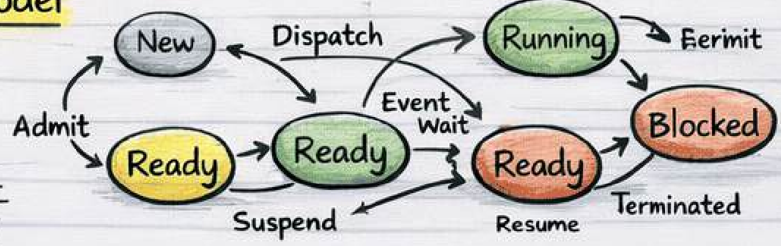
- Different States of a Process

- Five-State Process Model

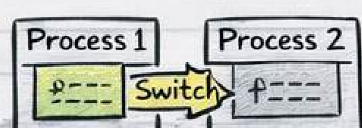


- Seven-State Process Model



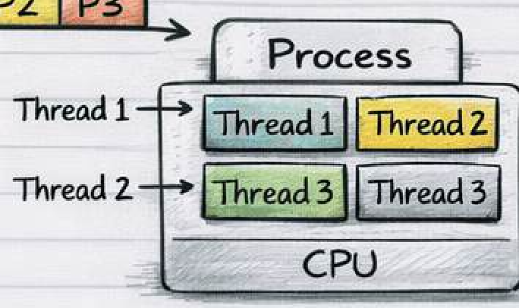


- Process Control Block



- Context Switching
- Process Scheduling

- Schedulers

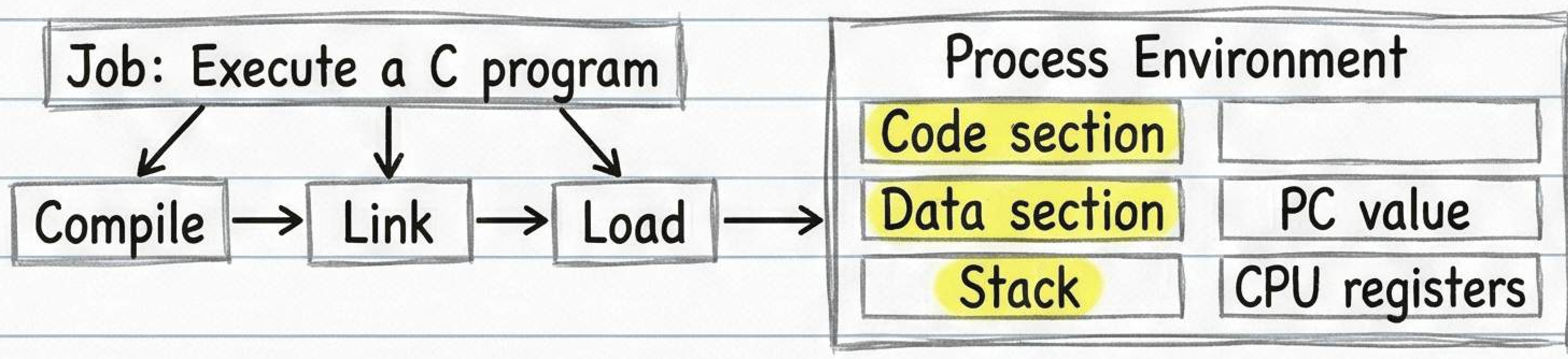


Module 2: Process Management

- Process management is a key component of the OS, handling various user and system processes.

Definition

- A process is a program in execution. Its setup includes an environment with: Code section, Data section, Stack, PC values (Program Counter), and register values.



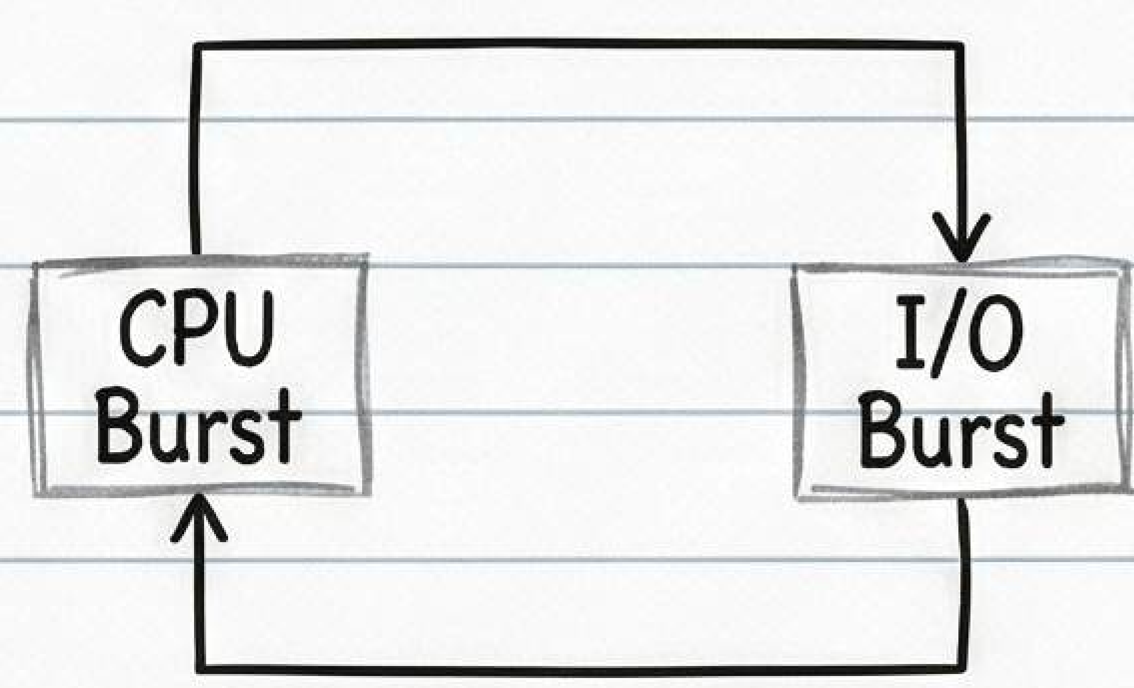
Program vs. Process

Program	Process
Passive / Static entity (on disk)	Active / Dynamic entity (executing)
Cannot compete for resources	Competes for resources (CPU, I/O)
Has only a code section	Has code, data, stack, PC, etc.

Process Execution & Bursts

- Process execution consists of a cycle of CPU execution and I/O wait.

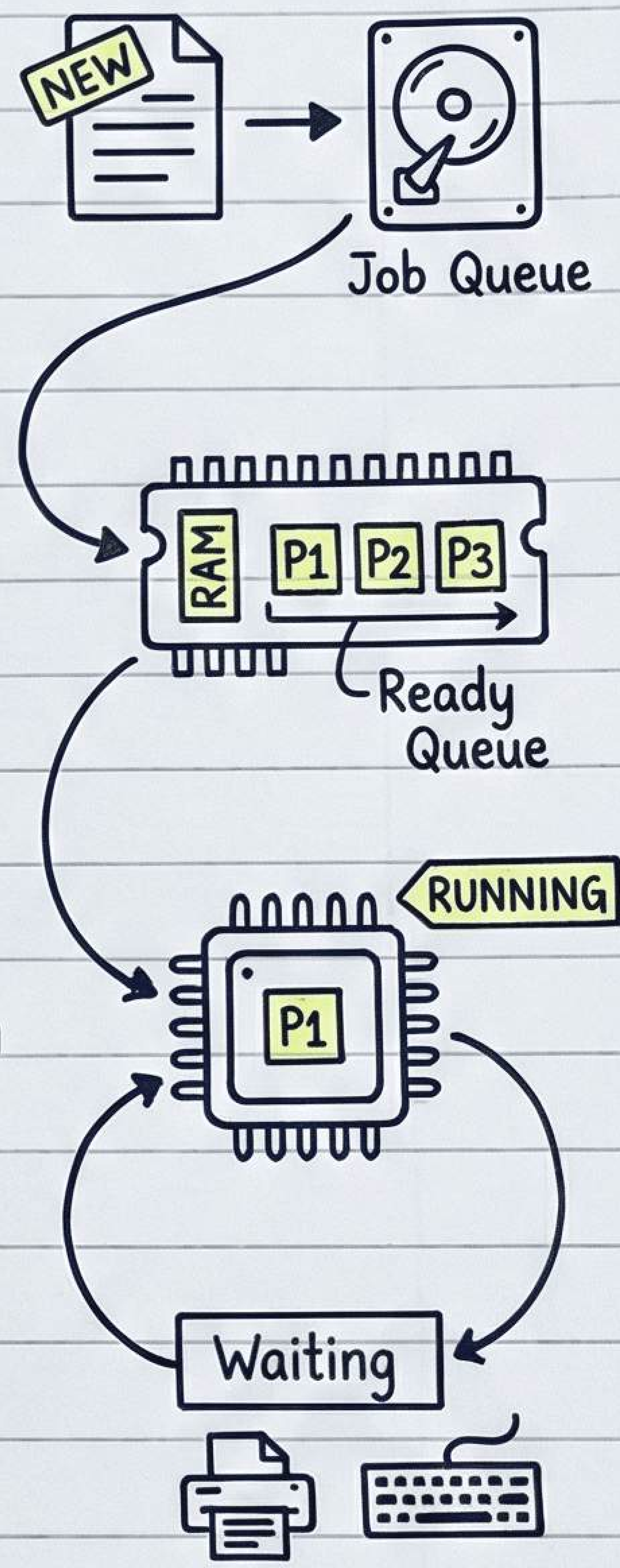
```
Printf("...");          } I/O-Burst
scanf("...");
if(x>y) {               }
    if(x>z) ...         } CPU-Burst
}   else ...            }
else {                  }
    if(y>z) ...         } CPU-Burst
    else ...            }
}
printf("...");          } I/O-Burst
```



Different States of a Process

A process is an active entity that changes its state with time from creation to termination.

1. **New State**: A process is in a new state when it is first created as a program/job. In batch systems, it's stored in a job pool/queue on the hard disk.

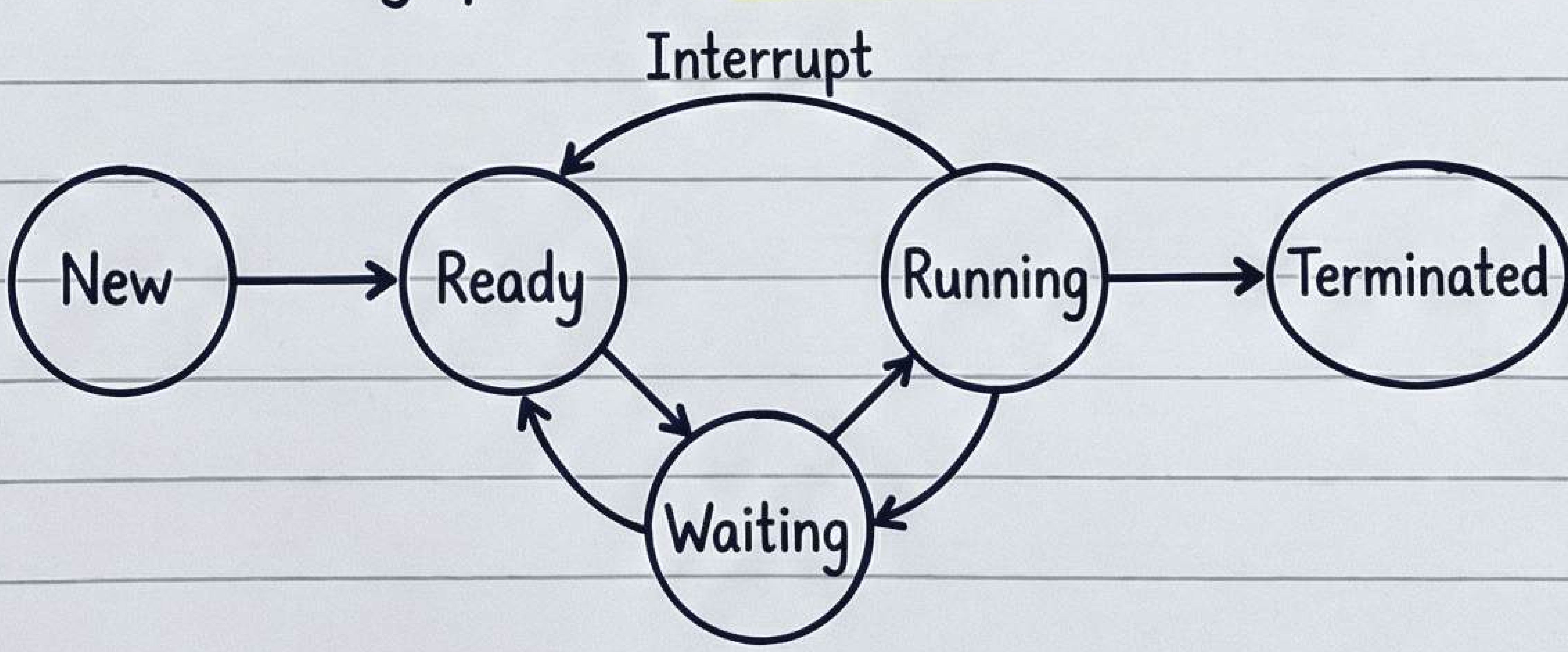


2. **Ready State**: When a job is loaded into main memory (via job scheduling), it enters the ready queue, waiting for the CPU. It is ready for execution and competes for resources.

3. **Running State**: The process is selected (via process/CPU scheduling) and sent for execution (via process dispatching). It gets CPU time and executes its code.

4. **Waiting State**: A process transitions to a wait state if it is interrupted or needs to access an I/O device.

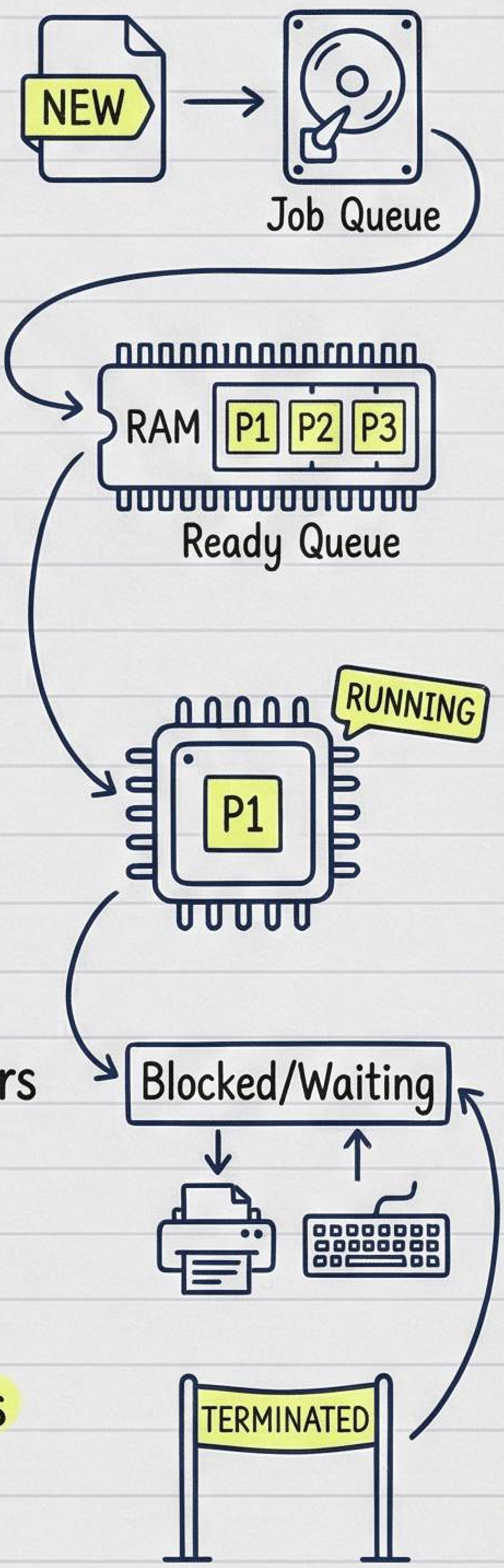
5. **Terminated State**: After executing its full code, the running process terminates.



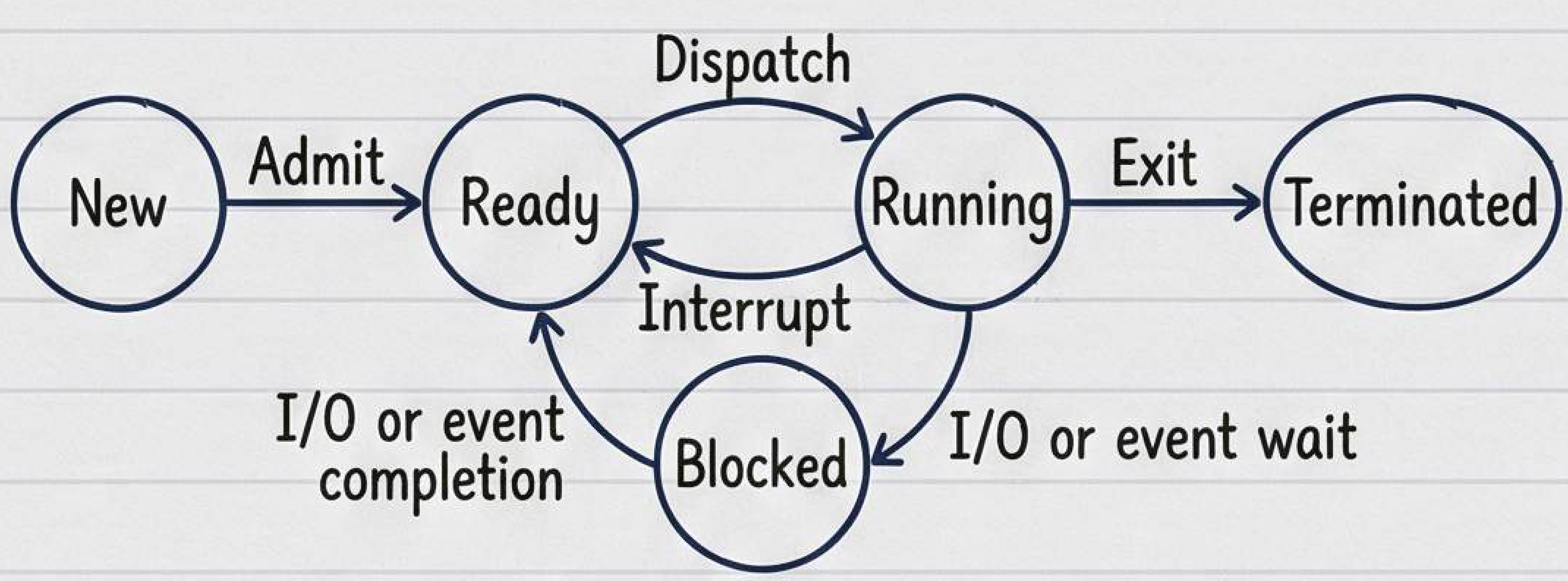
Process State Transition Diagram

Five-State Process Model

A process is an active entity that changes its state with time from creation to termination. This model manages processes in a multi-programming environment.



1. **New State:** A process is in a new state when it is first created as a program/job. It's stored in a job pool/queue on the hard disk (secondary storage) in a batch system.
2. **Ready State:** When a job is loaded into main memory (via job scheduling), it enters the ready queue, waiting for the CPU. It is ready for execution, competes for resources, but is waiting for its turn.
3. **Running State:** The process is selected (via process/CPU scheduling) and sent for execution (via process dispatching). It gets CPU time and executes its code.
4. **Blocked/Waiting State:** A running process enters this state if it has to wait for an I/O device or another event. The CPU is taken away, and the process waits in a separate blocked queue.
5. **Terminated State:** The process has executed its full code and finished. It releases all allocated resources.



Process State Transition Diagram

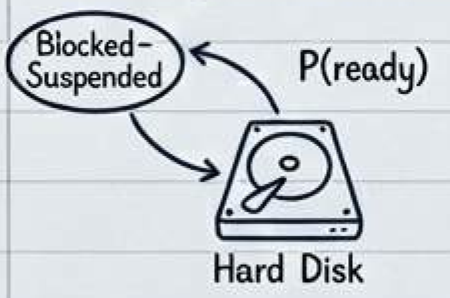
Seven-State Process Model

This model extends the five-state model by adding two suspended states . This is done to free up main memory when it's full by swapping processes out to secondary storage (disk)

The Two New States:

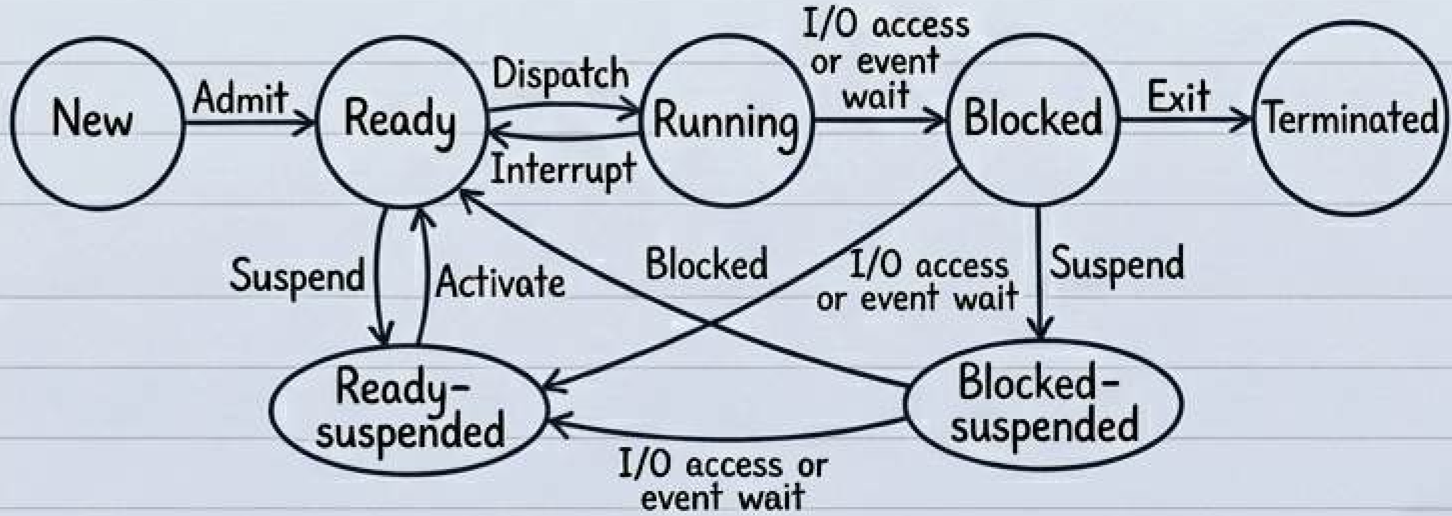
- Blocked-Suspended: A process in the blocked state (waiting for I/O) is swapped out to the suspended queue on the disk . It is still waiting for the event but is now on disk

- Ready-Suspended: A process that is ready to execute but is stored on the disk . It can come from the Blocked-Suspended state when its I/O event completes , or directly from the Ready state to free up memory



Key Swapping Transitions:

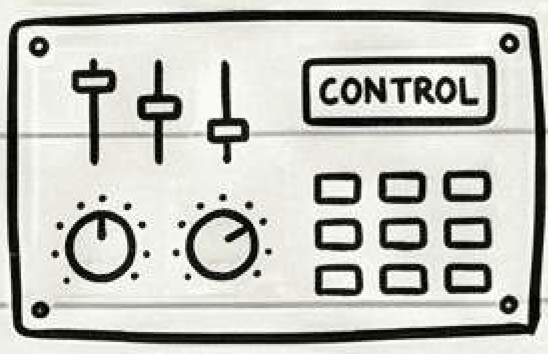
- Suspend: Moving a process from memory (Ready/Blocked) to disk (Ready-Suspended/Blocked-Suspended)
- Activate: Moving a process from disk (Ready-Suspended) back to the ready queue in memory



Seven-State Transition Diagram

Process Control Block (PCB)

A data structure used to control the execution of processes in a multi-programming environment.



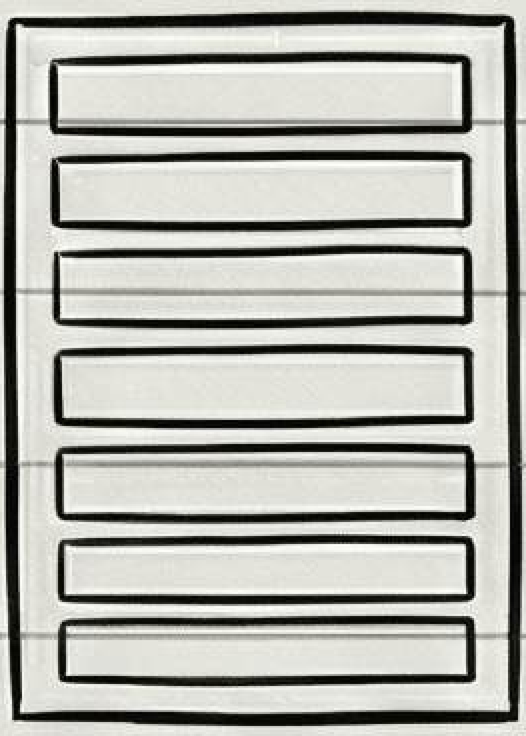
It stores various attributes related to the process.

A new PCB is created for every new process.

When an interrupt stops a process, its current status is saved in its PCB. This is a context save operation.

Key Attributes:

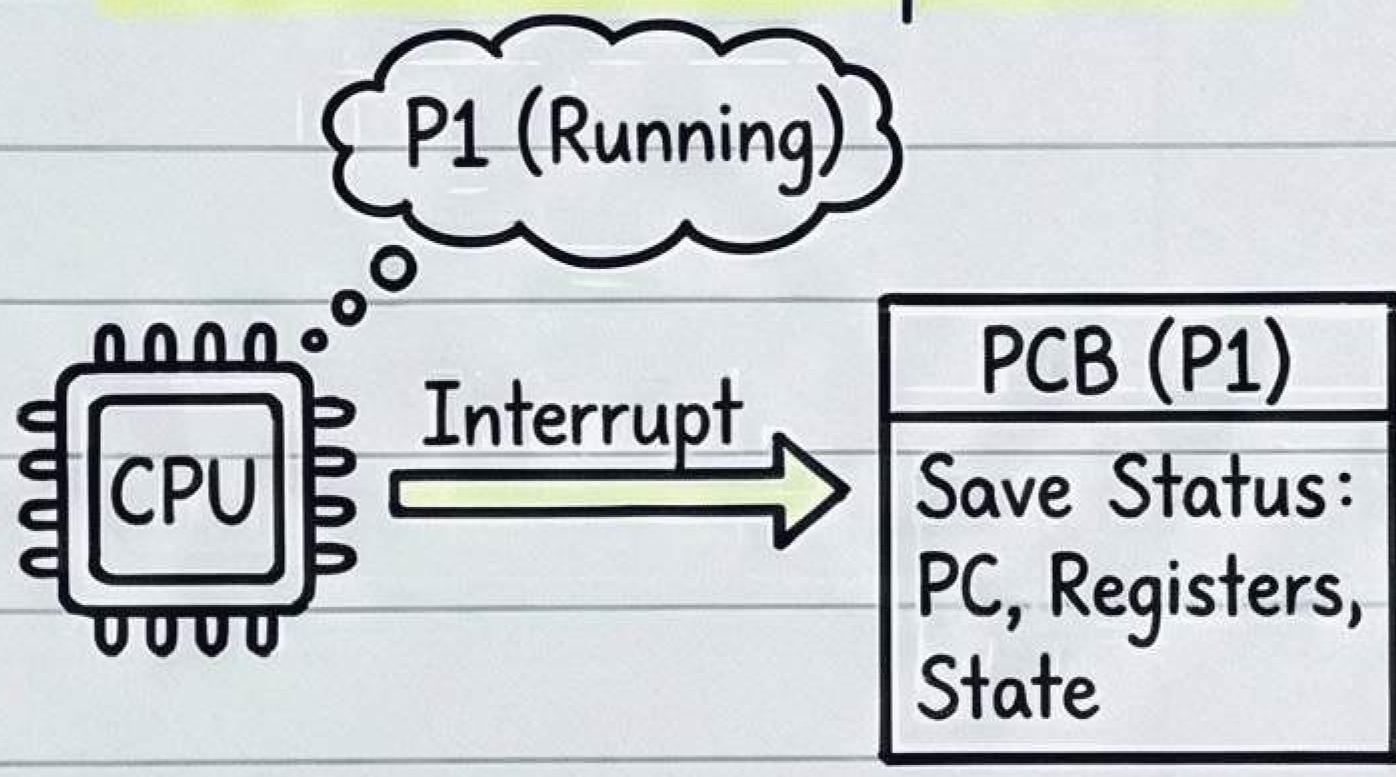
- PID (Process ID)
- PC and CPU registers
- Process state
- Process priority
- Event information
- Memory-related information
- Resource-related information
- Scheduling-related information
- Various pointers



PCB Attributes

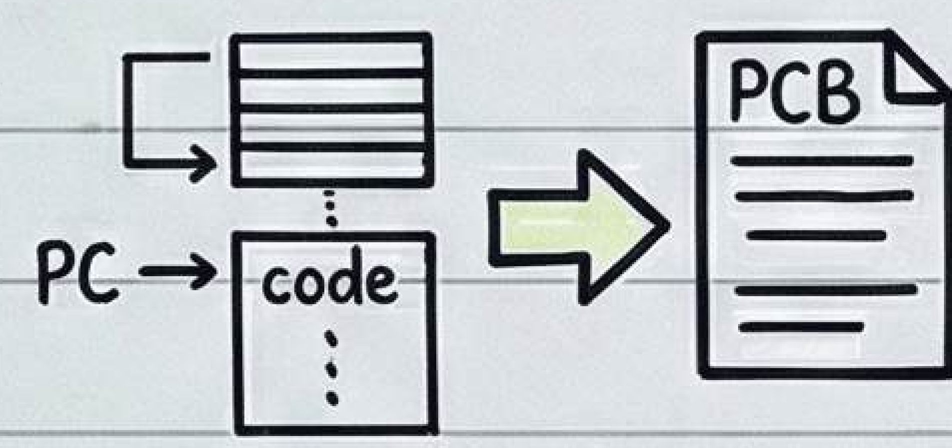
Context Switching

When a running process is interrupted (e.g., for I/O or time slice), the OS must save its current status This information is stored in the process's Process Control Block (PCB). This is known as a context save operation.

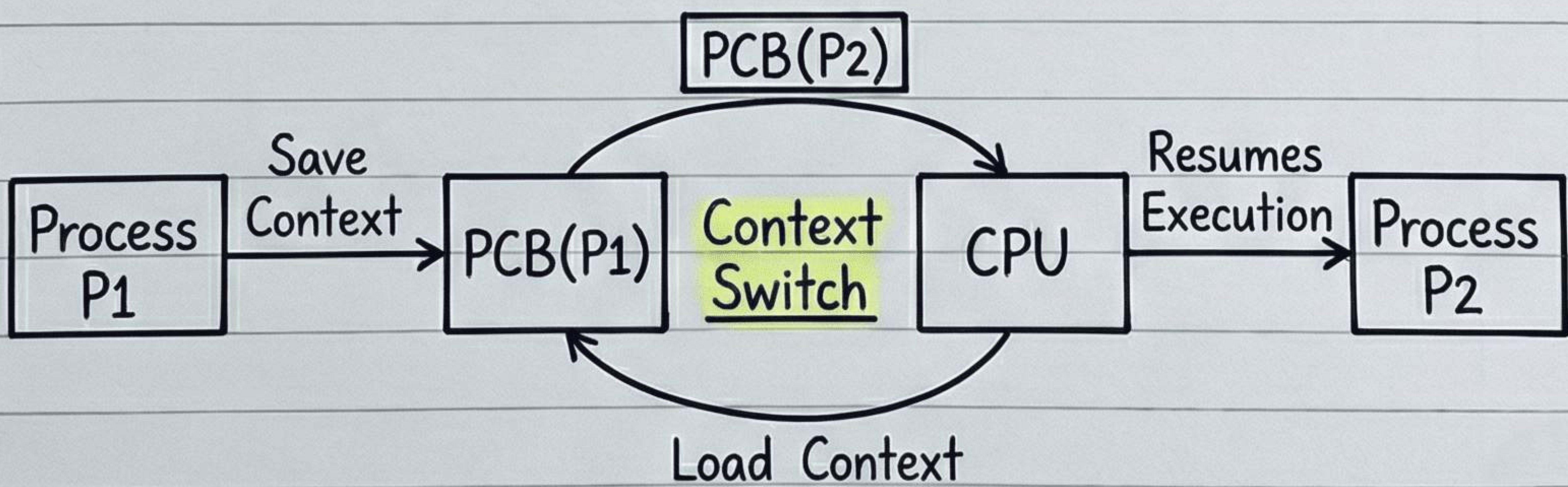


What is the "Context"?

- The context includes the process's vital information, as stored in the PCB:
  - ∗ Program Counter (PC) and CPU registers.
  - ∗ Process state.
  - Memory management information, etc.

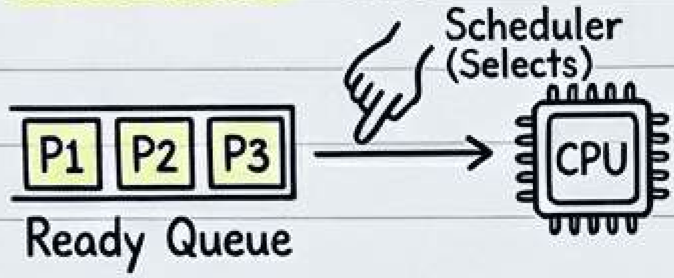


After saving the old process's context, the OS loads the saved context of the new process (selected by the scheduler) from its PCB. This is a context restore operation. The CPU then resumes execution of the new process.



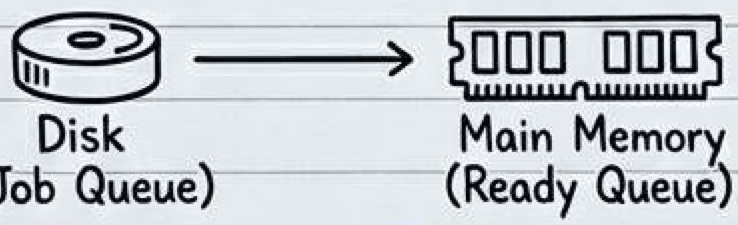
Process Scheduling

In a multiprogramming environment, multiple processes in the ready queue compete for the CPU.
The OS must select which process to run next. This task is performed by the scheduler and dispatcher.

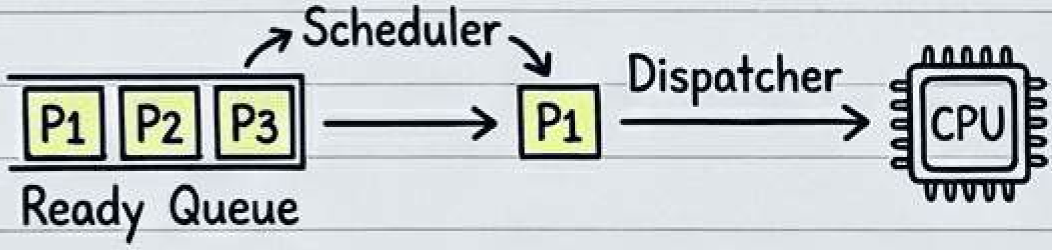


Key Components & Stages:

1. Job Scheduling: Selecting a job from the job queue (on disk) and bringing it into the ready queue (in main memory).



2. Process (or CPU) Scheduling: Selecting a process from the ready queue for the next execution on the CPU.
3. Process Dispatching: Sending the selected process to the CPU for execution.



The goal is to switch the CPU among processes to maximize utilization and provide interactivity.

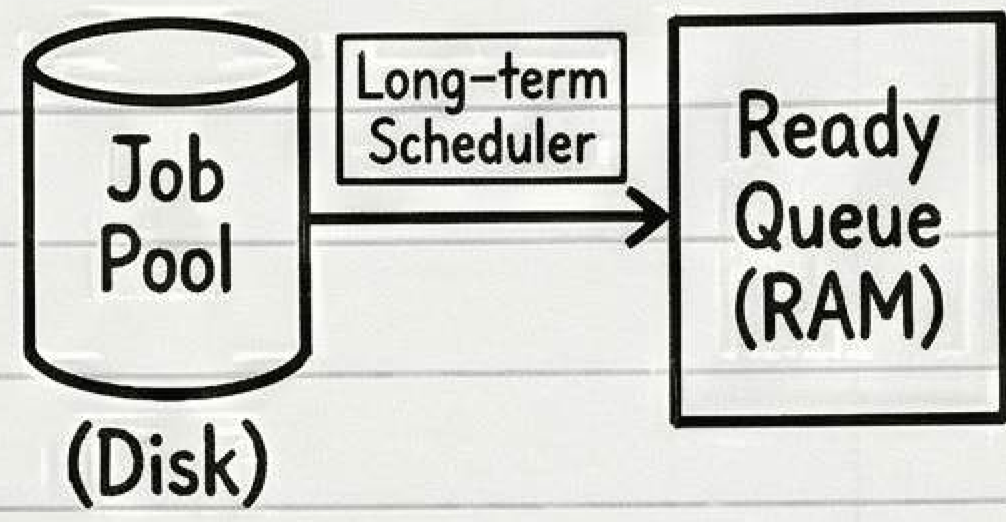
SCHEDULERS: Long, Short, & Medium Term

1. Long-term Scheduler (Job Scheduler)

Selects a job from the job pool and sends it to the ready queue. This controls the degree of multi-programming

It is not invoked frequently

Needed in batch processing but absent in time-sharing systems (jobs go directly to ready queue)

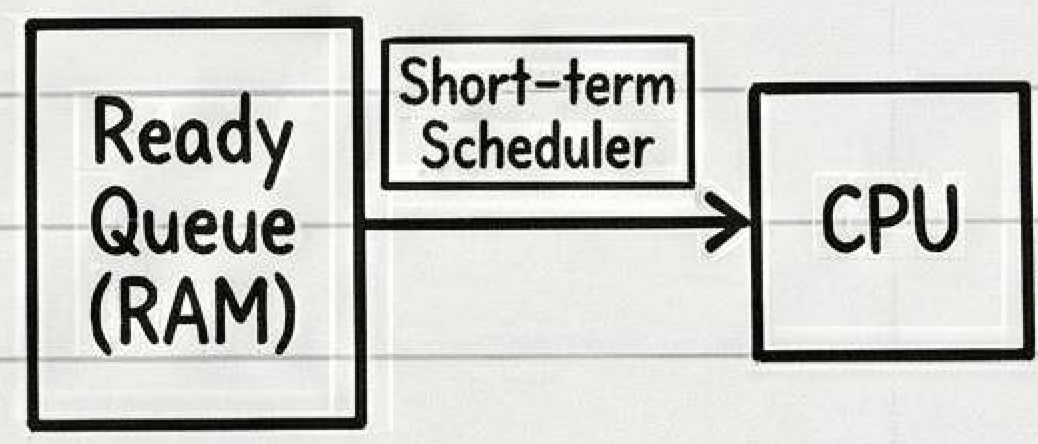


2. Short-term Scheduler (Process Scheduler)

Selects a process from the ready queue to be dispatched to the CPU

Invoked very frequently, like after every interrupt

Performs process scheduling

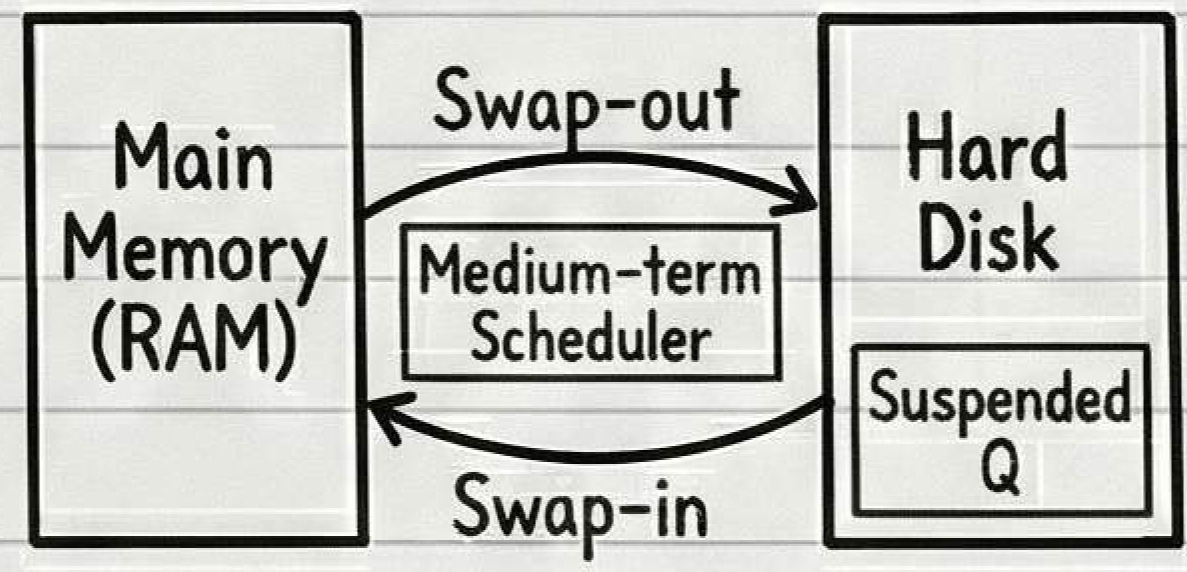


3. Medium-term Scheduler

Invoked to swap out blocked processes from main memory to the hard disk (blocked-suspended queue) when memory is needed

Later, it swaps in the process back to the ready queue when its I/O is complete

Performs swap-in and swap-out functions

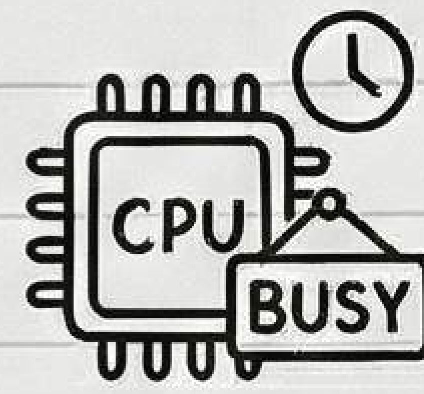


Type	Purpose
Long-term	Job scheduling
Short-term	Process scheduling
Medium-term	Swap-in/Swap-out

SCHEDULING CRITERIA

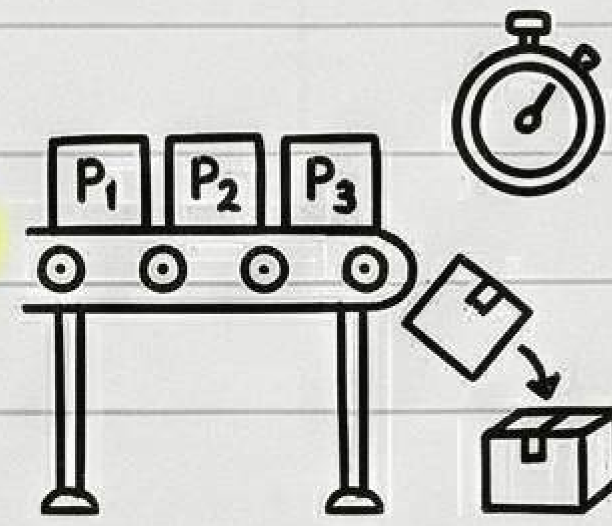
CPU Utilization

CPU utilization is the percentage of time that the CPU is busy executing the processes



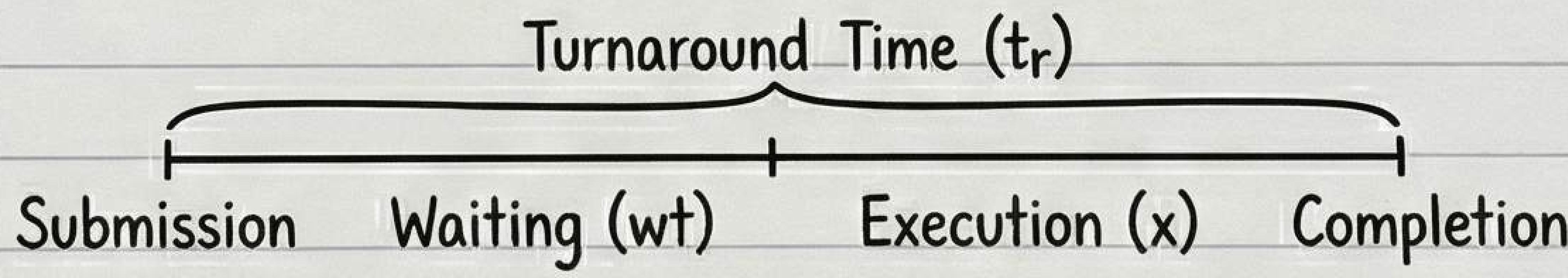
Throughput

Throughput is the number of processes completed in a unit time



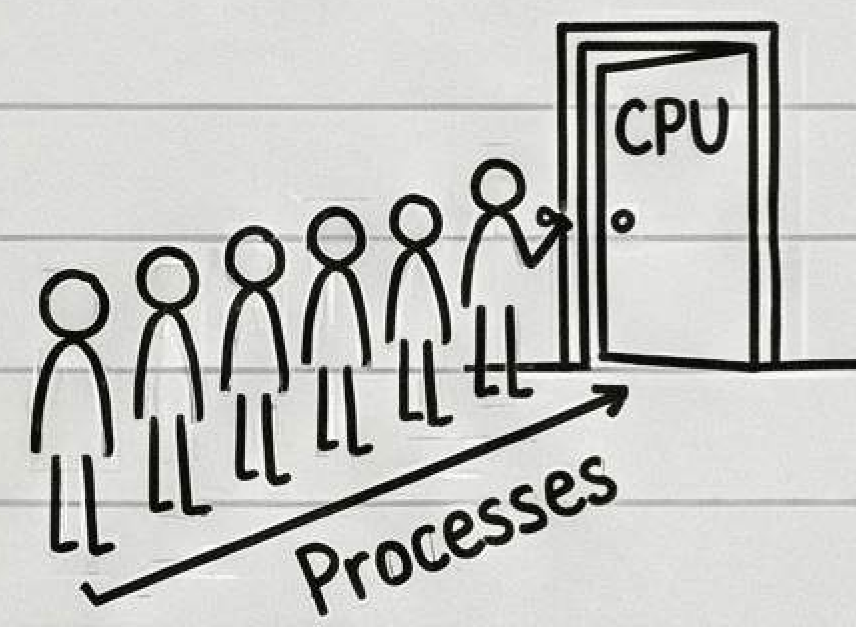
Turnaround Time

Turnaround time is the total time spent by a process in the system
Formula: $t_r = wt + x$, where wt is waiting time in ready queue, and x is total execution time



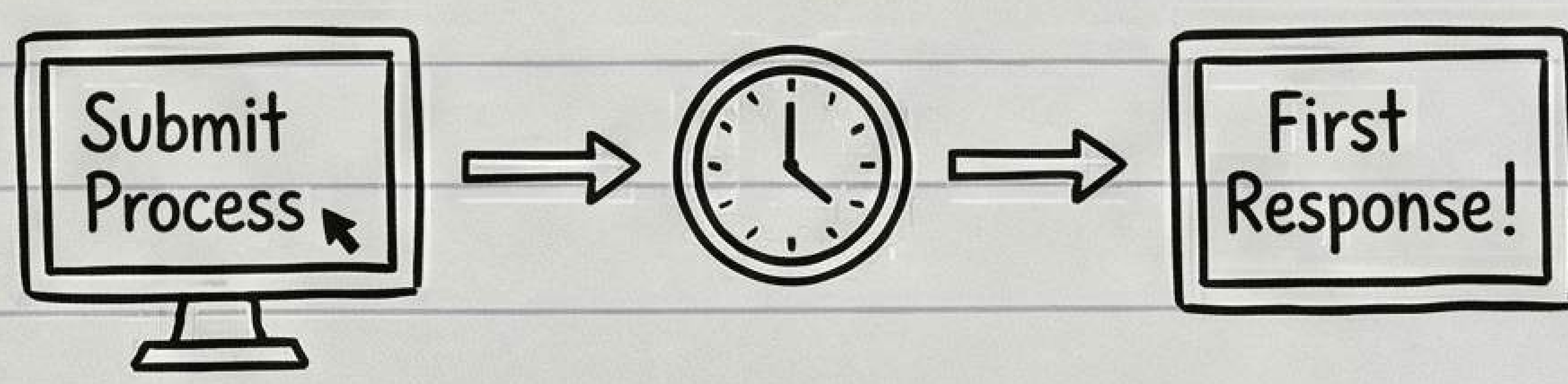
Waiting Time

The waiting time is the total time spent by a process in the ready queue

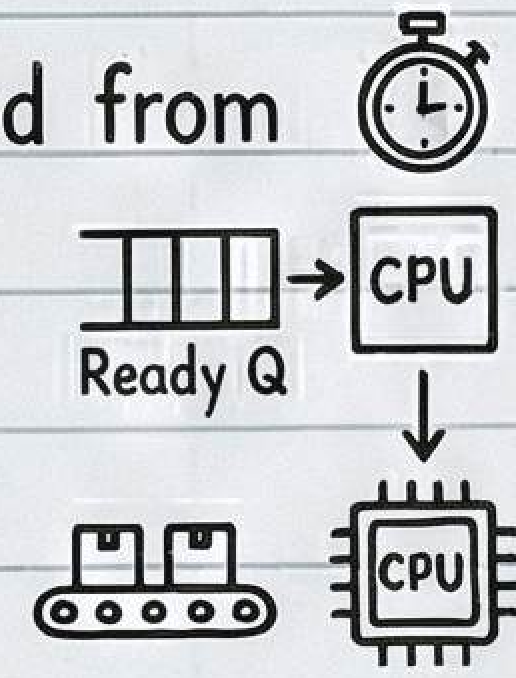


Response Time

Response time is the time period between the time of submission of a process and the first response given by the process to the user

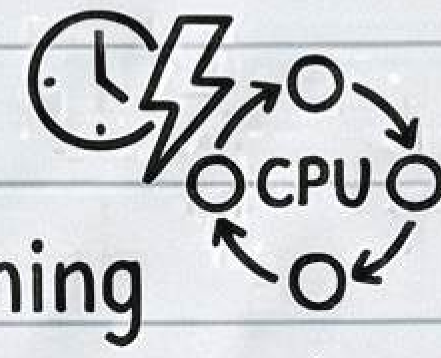


SCHEDULING ALGORITHMS

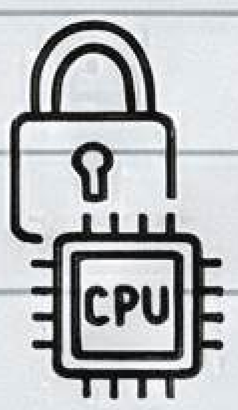


The scheduling mechanisms, by which a process is selected from the ready queue, are called scheduling algorithms
The goal is to have minimum turnaround time, minimum waiting time, maximum throughput, and maximum CPU utilization

Pre-emptive and Non-preemptive



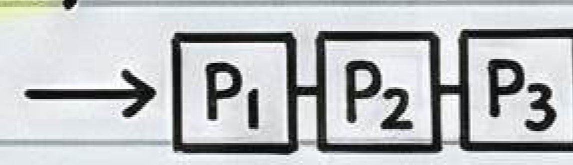
Non-preemptive Scheduling:
Once a process gets the CPU, it keeps it until it voluntarily gives it up (finishes or waits for I/O)
The OS can't force it off :ite:



Preemptive Scheduling:
The OS can interrupt a running process (force it to give up the CPU) . Usually done after a fixed time slice
Allows for better sharing

Specific Algorithms

FCFS (First-Come, First-Served): Used in batch systems, works on a FIFO queue



Round Robin (RR): Provides equal chance to every process by using a time quantum periodically
- If quantum is too large, RR acts like FCFS
- If quantum is too small, there's large context switch time

Rules for quantum:
1. 80% of CPU bursts should be smaller than the quantum
2. Context switch time is nearly 10% of the quantum

Max waiting time: $w = (n-1) * q$

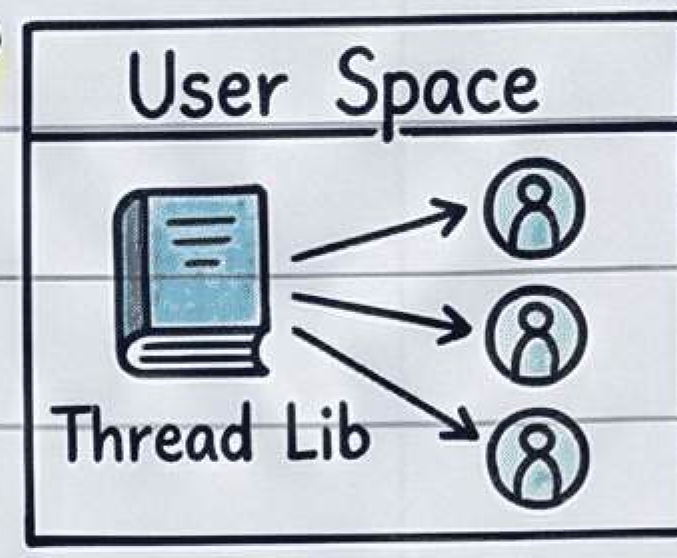
CPU consumption ratio $= \frac{\text{Actual CPU time}}{\text{total estimated time}}$

RR is appropriate for a multi-user environment

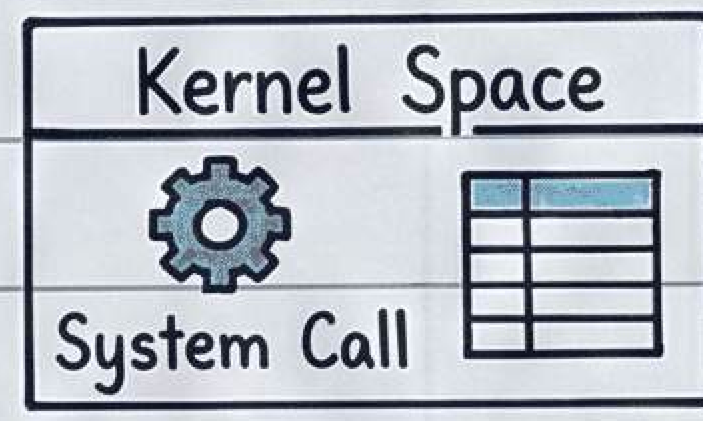
Concept of Threads & Multithreading

User vs. Kernel Threads

- 'User threads' are managed in 'user space' by the 'application' via a 'thread library' from the OS.



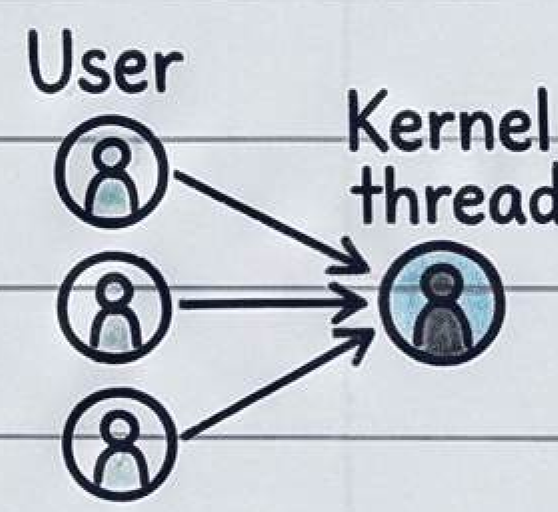
- A 'thread table' with pointers to each thread's TCB is maintained.
- They are 'easier & faster' to create/manage and 'more portable' (OS independent).
- 'Kernel threads' are managed in 'kernel space' via 'system calls'.



- Managed through a 'single-thread table' in the kernel.

Thread Mappings

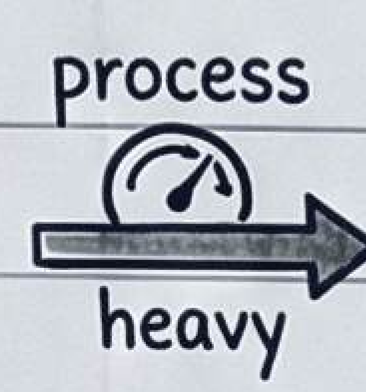
- 'User-level (Many-to-One)': OS maps 'all threads' in a process to a 'single-execution context'.



- 'Kernel-level (One-to-One)': Provides a 'one-to-one mapping' between user and 'kernel thread'.

Why Multithreading?

- Arose due to 'high context switch overhead' in process switching.



- An app with 'multiple threads' responds & executes 'quickly' and 'fast'.