Part 1: The Foundation

1. The CPU-I/O Burst Cycle Programs do not just use the CPU continuously. They cycle between two states:

- **CPU Burst:** Doing calculations or processing data.
- **I/O Burst:** Waiting for something external, like reading a file or waiting for the user to type something. Because the CPU sits idle during an I/O burst, the operating system uses multiprogramming to switch to another program to keep the CPU as busy as possible.

2. The Scheduler vs. The Dispatcher

- **Short-Term Scheduler:** This is the decision-maker. It looks at all the programs waiting in the "ready queue" and selects which one gets to use the CPU next.
- **Dispatcher:** This is the action-taker. Once the scheduler chooses a program, the dispatcher actually gives that program control of the CPU. It saves the state of the old program, loads the new one (context switching), and jumps to the right spot to restart it. The time it takes to do this swap is called **Dispatch Latency**.

3. The 5 Rules for Success (Scheduling Criteria) When the scheduler is deciding who goes next, it tries to optimize these five metrics:

- **Maximize CPU Utilization:** Keep the CPU as busy as possible.
- **Maximize Throughput:** Complete as many processes as possible within a given time unit.
- **Minimize Turnaround Time:** The total time it takes from starting a process until it completely finishes.
- **Minimize Waiting Time:** The total time a process spends just sitting in the ready queue.
- **Minimize Response Time:** The time it takes from a user submitting a request until the very first response is produced.

Part 2: Basic Algorithms

Let’s look at our first two scheduling algorithms: FCFS and SJF.

1. First-Come, First-Served (FCFS) This is the simplest algorithm: the process that requests the CPU first gets the CPU first.

Let's calculate the **Average Waiting Time** using an example from your slides:

- Imagine three processes arrive at the exact same time in this order: P1 (takes 24ms), P2 (takes 3ms), and P3 (takes 3ms).
- **P1** goes first. It waits **0ms**.
- **P2** goes second. It must wait for P1 to finish, so it waits **24ms**.

- **P3** goes third. It waits for P1 and P2, so it waits **27ms** (24 + 3).
- The average waiting time is: (0 + 24 + 27) / 3 = **17ms**.

The major problem with FCFS: It suffers from the **Convoy Effect**. If a very long process gets the CPU first, all the short processes get stuck waiting behind it, resulting in high waiting times. If P2 and P3 had simply gone first, the average wait would have dropped dramatically down to just 3ms!

2. Shortest-Job-First (SJF) To fix the Convoy Effect, SJF looks at all the available processes and gives the CPU to the one that needs the *least* amount of time to finish.

- **The Benefit:** It is mathematically optimal because it guarantees the minimum possible average waiting time for a given set of processes.
- **The Problem:** The computer cannot easily look into the future, making it very difficult to know exactly how long a process's next CPU request will be before it actually runs.

SJF can be run in two ways:

- **Non-preemptive:** Once a process starts, it keeps the CPU until it finishes its burst.

For **Non-preemptive SJF**, a process never stops once it starts. The formula is simply: **Waiting Time = Start Time - Arrival Time**. *Example:* If a process arrives at 2ms but the CPU doesn't start running it until 8ms, its waiting time is 6ms (8 - 2)

- **Preemptive:** If a new process arrives in the queue and its time is shorter than the *remaining* time of the currently running process, the CPU will interrupt the current process and switch to the new one. This specific method is called **Shortest-Remaining-Time-First (SRTF)**.

For **Preemptive SJF (SRTF)**, a process might be paused and restarted multiple times, so we have to account for the total time it spent in the system minus the time it actually spent running. The easiest formula is: **Waiting Time = Finish Time - Arrival Time - Total Burst (Execution) Time**. *Example:* Process 1 arrives at 0ms and needs 7ms of CPU time. It runs for a bit, gets interrupted by a shorter job, and finally finishes at 16ms. Its waiting time is 9ms (16 finish time - 0 arrival time - 7 run time).

To get the **Average Waiting Time**, you simply calculate the waiting time for each individual process, add them all together, and divide by the total number of processes.

Response Time:

Response time is the amount of time it takes from when a process arrives until it gets its very first turn on the CPU.

You calculate it with this simple formula: **Response Time = First Start Time - Arrival Time**.

Here is a helpful trick to remember for your exam:

- For **non-preemptive** algorithms (like FCFS and standard SJF), response time is exactly the same as waiting time. This is because once a process gets its first turn, it never stops.
- For **preemptive** algorithms (like SRTF and Round Robin), response time is usually much smaller than total waiting time. A process gets a quick first turn, but then might be paused and forced to wait again.

Part 3: Priority Scheduling and Round Robin

1. Priority Scheduling Every process is given a priority number. The CPU always selects the process with the highest priority first (usually, a smaller number means a higher priority).

- **The Problem (Starvation):** If high-priority jobs keep arriving, low-priority jobs might wait forever in the queue and "starve".
- **The Solution (Aging):** The operating system slowly increases the priority of processes the longer they wait, guaranteeing they eventually get to run.

2. Round Robin (RR) This algorithm is all about equal turns. Every process gets a small, fixed slice of CPU time called a **time quantum** (usually 10 to 100 milliseconds).

- Once a process uses up its time slice, a timer interrupts it, and it is put at the very back of the ready queue to wait for its next turn.
- **The Balance:** If the time slice is too huge, RR just acts like the First-Come, First-Served (FCFS) algorithm. If the time slice is too tiny, the CPU wastes all its energy constantly switching between tasks instead of doing actual work.

In Round Robin, processes are handled in the order they wait in the "ready queue" (like P1, then P2, then P3), and every single process is given the exact same maximum time block, which is called a **time quantum**.

Here is a detailed breakdown of how that works:

- **Taking Turns:** The scheduler gives P1 its time block (for example, 10 milliseconds). Once that time runs out, a timer interrupts P1, pauses it, and puts it at the very back of the line.
- **Moving Down the Line:** The CPU immediately moves to P2 and gives it the exact same 10ms block.
- **Finishing Early:** If a process needs *less* time than the time quantum, it simply finishes its task and gives up the CPU early. The scheduler then immediately moves to the next process in line.

Because of this rotating system, if you have *n* processes waiting, no process ever has to wait more than (n-1)q time units before getting its next turn. Everyone gets a fair, predictable slice of the pie.

3. Multilevel Queues Instead of one big line, the main ready queue is split into separate smaller lines (for example, one for interactive foreground tasks and one for background tasks). In a **Multilevel Feedback Queue**, processes can even drop down into slower lines if they take up too much CPU time.

A **Multilevel Queue** works by splitting the main "ready queue" into several separate, smaller lines. Instead of one big line, you might have one line for "foreground" tasks (like an app you are actively using) and another for "background" tasks. Each line can even use its own scheduling rule, like Round Robin for the foreground and First-Come, First-Served (FCFS) for the background.

The "slow" lines are the background lines, which often use the FCFS algorithm. In a more advanced version called a **Multilevel Feedback Queue**, processes can actually be forced into these slow lines. If a task takes too much time in a fast, high-priority line, the system pushes it down into a slower, lower-priority line. The absolute slowest line is usually the bottom FCFS queue, where long tasks just sit and wait their turn to finish.