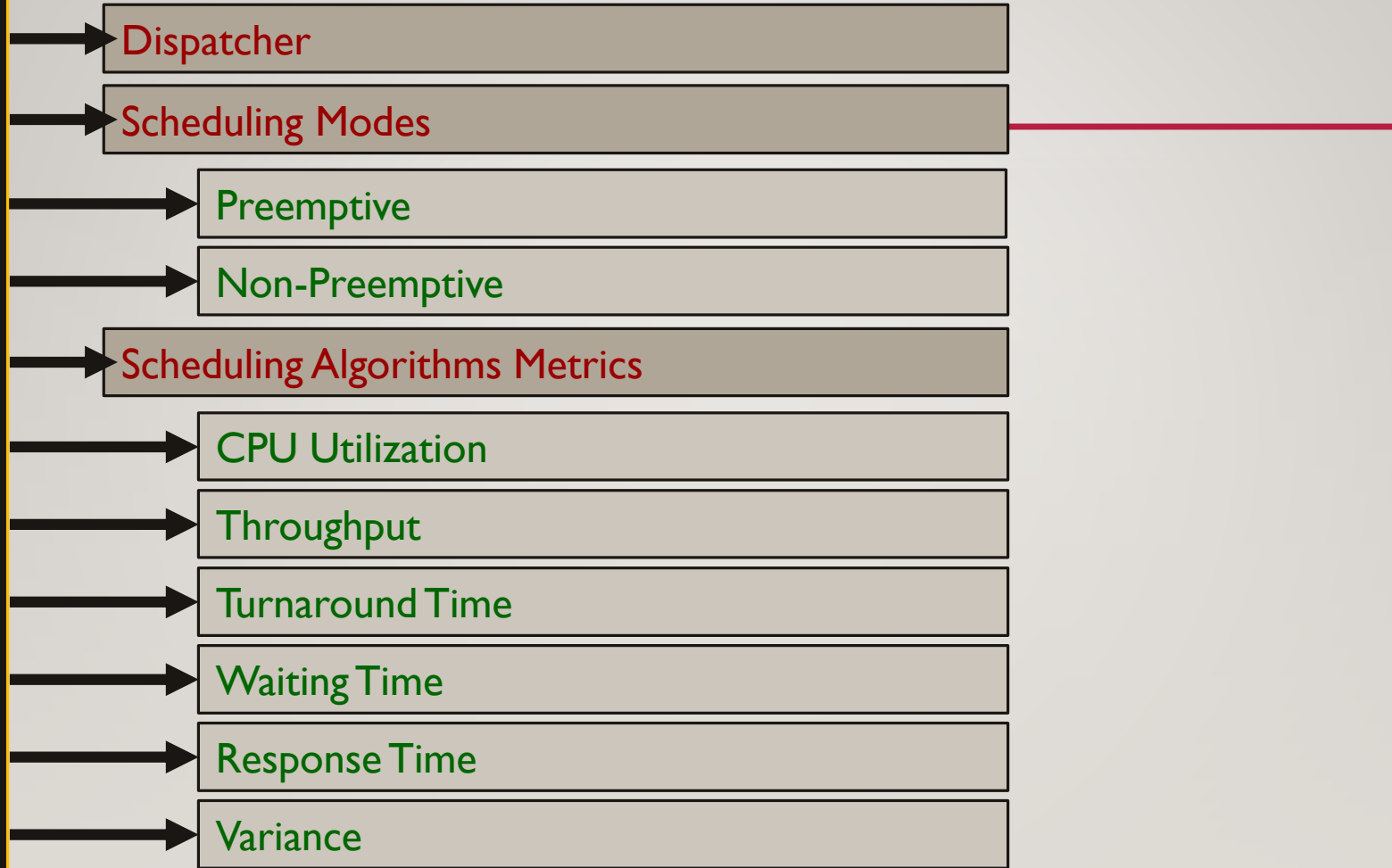


PART 3

# CPU SCHEDULER

gettyimages®

Daniel Grill

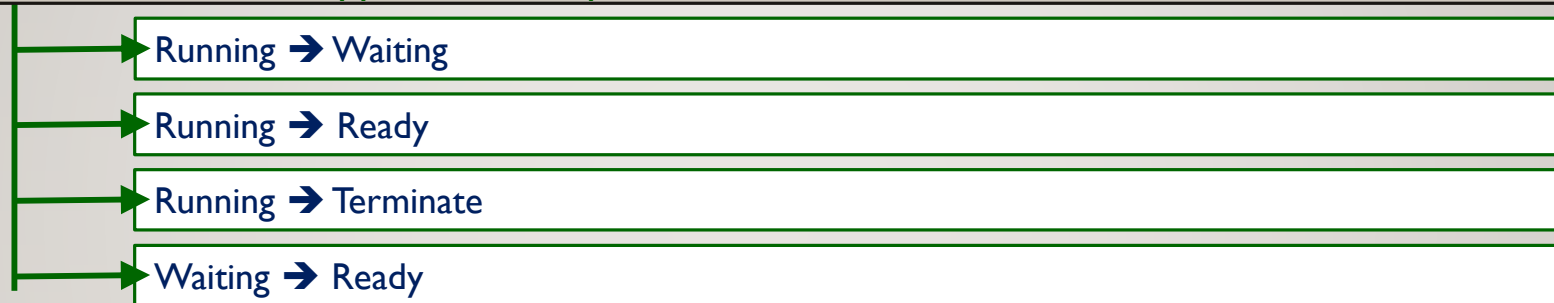


## INTRODUCTION

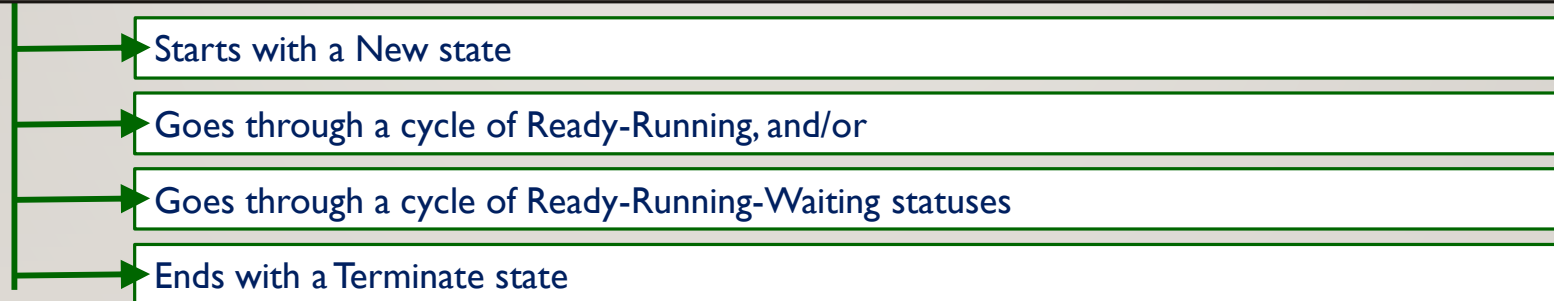
The CPU Scheduler (short-term scheduler) selects one of the processes that are in the ready queue and assigns it to the processor.

The process selection is based on specific criteria defined by the scheduling algorithm.

The CPU Scheduler is invoked when the process running on the CPU is to be replaced by another one. This happens when a process switches from:

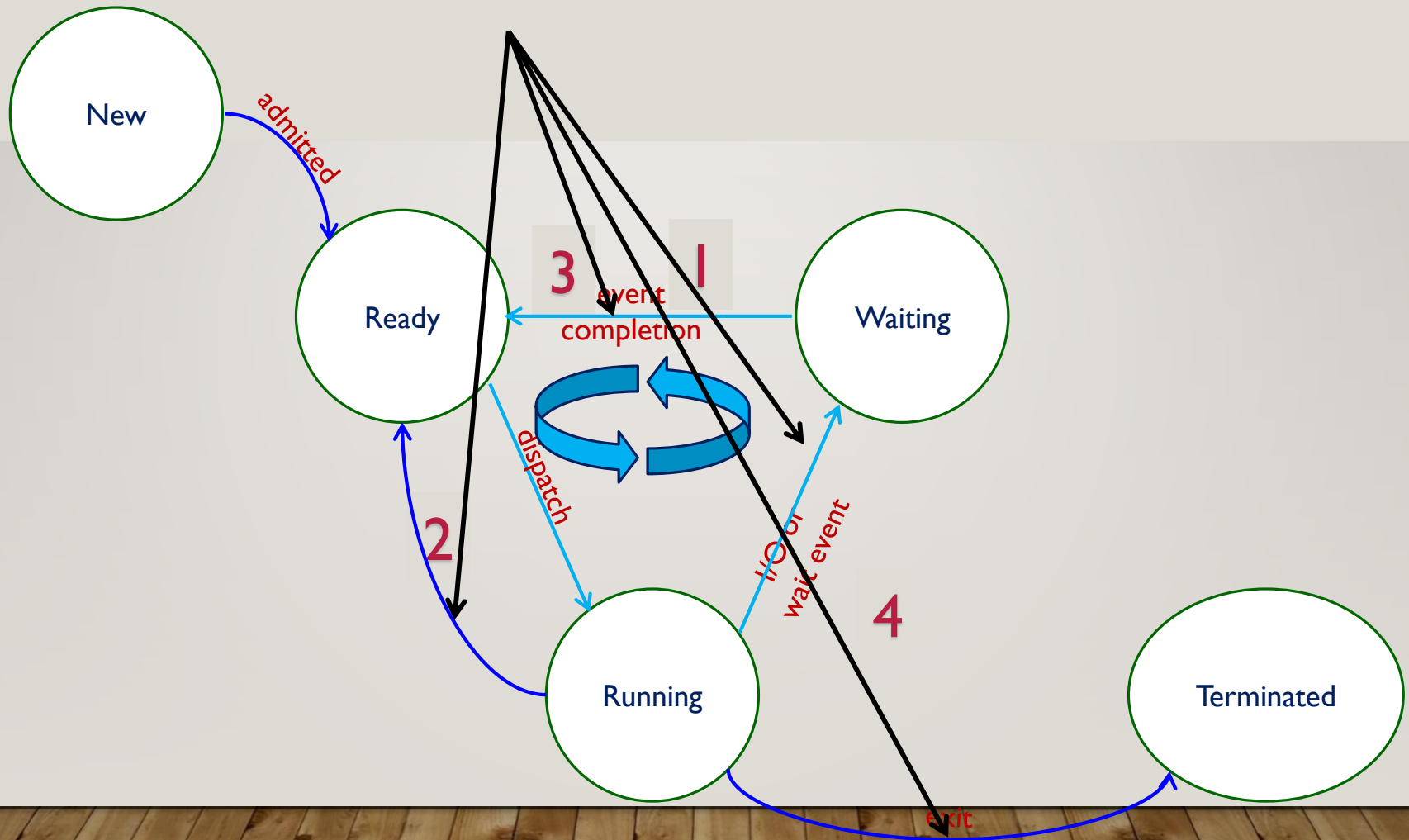


As seen in the figure in the next slide, a process goes through the following phases:



## CPU SCHEDULER INVOCATION

The following simplified process state diagram shows the previously mentioned four cases in which the CPU Scheduler is invoked.

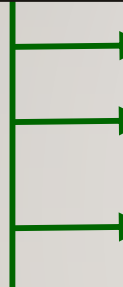


## DISPATCHER

After the short-term schedule selects the process to be assigned to the CPU, the **dispatcher** is invoked.

The dispatcher is the executive module of the CPU Scheduler that gives control of the CPU to the selected process.

The dispatcher performs the following functions in order:

- 
1. Context switching
  2. Switching to the user mode
  3. Jumping to the proper location of the newly selected process, as registered in its Program Counter (PC).

The time taken by the dispatcher to stop one process and start another one is known as the **dispatch latency**.

The dispatcher should be as fast as possible. In other words, the dispatch latency should be minimized as much as possible.

## SCHEDULING MODES

A CPU Scheduler may follow one of two scheduling modes:



## SCHEDULING MODES – PREEMPTIVE SCHEDULING

In **preemptive scheduling**, a process **P1** may be interrupted while running to be replaced by another process **P2** that better satisfies the criteria of the scheduling algorithm.

However, when the running process is interrupted, the currently executing instruction in **P1** is first completed before being replaced by the other process **P2**.

Modern OSs such as Windows 95 and all subsequent versions of Windows, MAC OS X use preemptive scheduling since it is proven to be more efficient.

On the other hand, preemptive scheduling has a few disadvantages such as:

### Race condition

When data are shared between processes, the preemption of a process may result in data inconsistency amongst the processes.

This is to be explained in more details later in *Process Synchronization*.

### Design of OS kernel

A process may be preempted while the OS kernel is modifying its own data. This results in a chaos: the result is unpredictable.

In order to avoid such chaos, some OSs such as most versions of UNIX, disable interrupts while an interrupt handler is executed.

Although this simplifies the design of the OS, but does not support real systems.

On the other hand, an OS should accept interrupts at any time. Otherwise, input may be lost or output might be overwritten.

## SCHEDULING MODES – NON-PREEMPTIVE SCHEDULING

Under **non-preemptive scheduling**, once the CPU is allocated to a process, the latter keeps running until it leaves the CPU voluntarily.

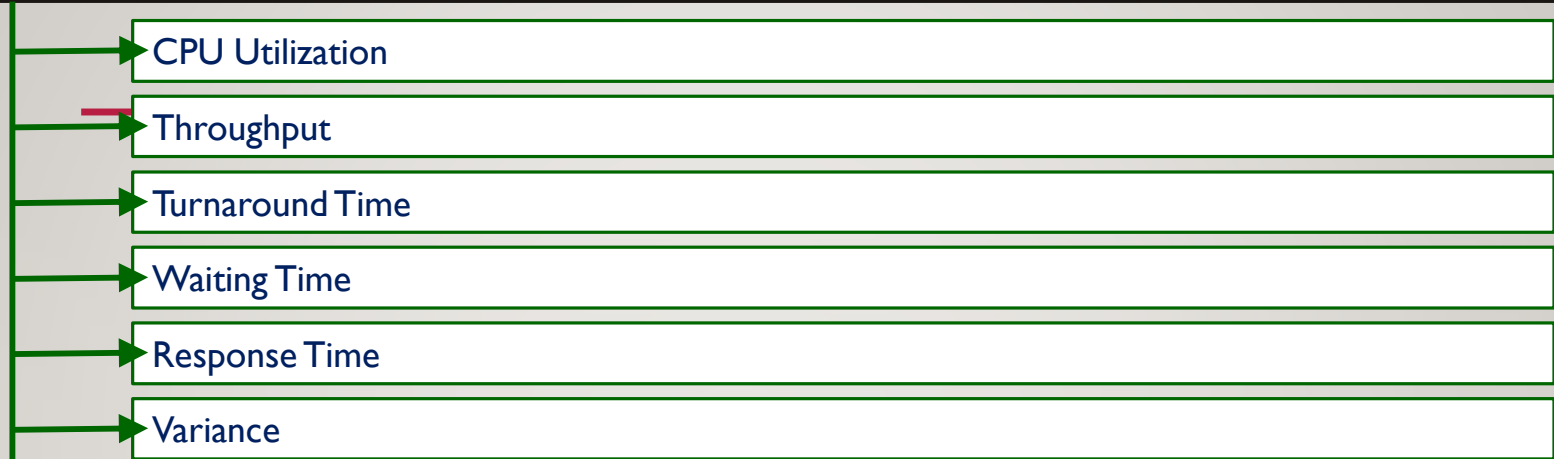
In other words, the process keeps running until it either terminates or it switches to the Wait status.

Non-preemptive scheduling is the only available choice on certain hardware platforms that do not have the special hardware needed for preemption such as the timer.



## SCHEDULING ALGORITHMS METRICS (I) – INTRODUCTION

In order to compare the performance of different scheduling algorithms, some **metrics** are used such as:



## SCHEDULING ALGORITHMS METRICS (2) – CPU UTILIZATION

CPU Utilization indicates how much the CPU is busy.

CPU utilization is equal to the ratio of the time in which the CPU is busy to the total time (busy time + idle time) \* 100%.

CPU utilization should be maximized.

## SCHEDULING ALGORITHMS METRICS (3) – THROUGHPUT

The **throughput** is the number of processes that are completed per unit time.

The throughput is affected by the processor speed and the process length.

The throughput should be maximized.

## SCHEDULING ALGORITHMS METRICS (4) – TURNAROUND TIME

The **Turnaround time** of a process is its lifetime.

The turnaround time of a process is calculated as the interval time between the process was in the New state till it is terminated (Terminated state).

Therefore, the time that the process spends in waiting an I/O device is also taken into consideration.

Accordingly, the turnaround time of a process is also affected by the speed of the I/O device.

The turnaround time of a process should be minimized.

## SCHEDULING ALGORITHMS METRICS (5) – WAITING TIME

The **Waiting Time** of a process  $P_i$  is the sum of the periods spent by  $P_i$  waiting in the ready queue.

As previously mentioned, a process  $P_i$  goes through many cycles of Ready-Running and Ready-Running-Wait cycles.

The **Waiting Time** measures how much time the process spent waiting for the CPU in the ready queue.

The waiting time of a process should be minimized.

## SCHEDULING ALGORITHMS METRICS (6) – RESPONSE TIME

The **Response Time** of a process  $P_i$  is used in time-sharing environment.

The Response Time is the time elapsed between issuing a command and the time at which the process starts to respond.

As the number of processes competing for the CPUs in a time-sharing environment increases, as the response time is subject to increase.

The response time of a process should be minimized.

## SCHEDULING ALGORITHMS METRICS (7) – VARIANCE

The **Variance** measures the constancy (stability) of a process in a time-sharing environment.

The variance should be minimized.

A system with minimum variance and longer response time, is more desirable than that which is faster on the average but is highly variant (unstable).