CPU scheduling

Module 3

Lecture

Operating systems 2019

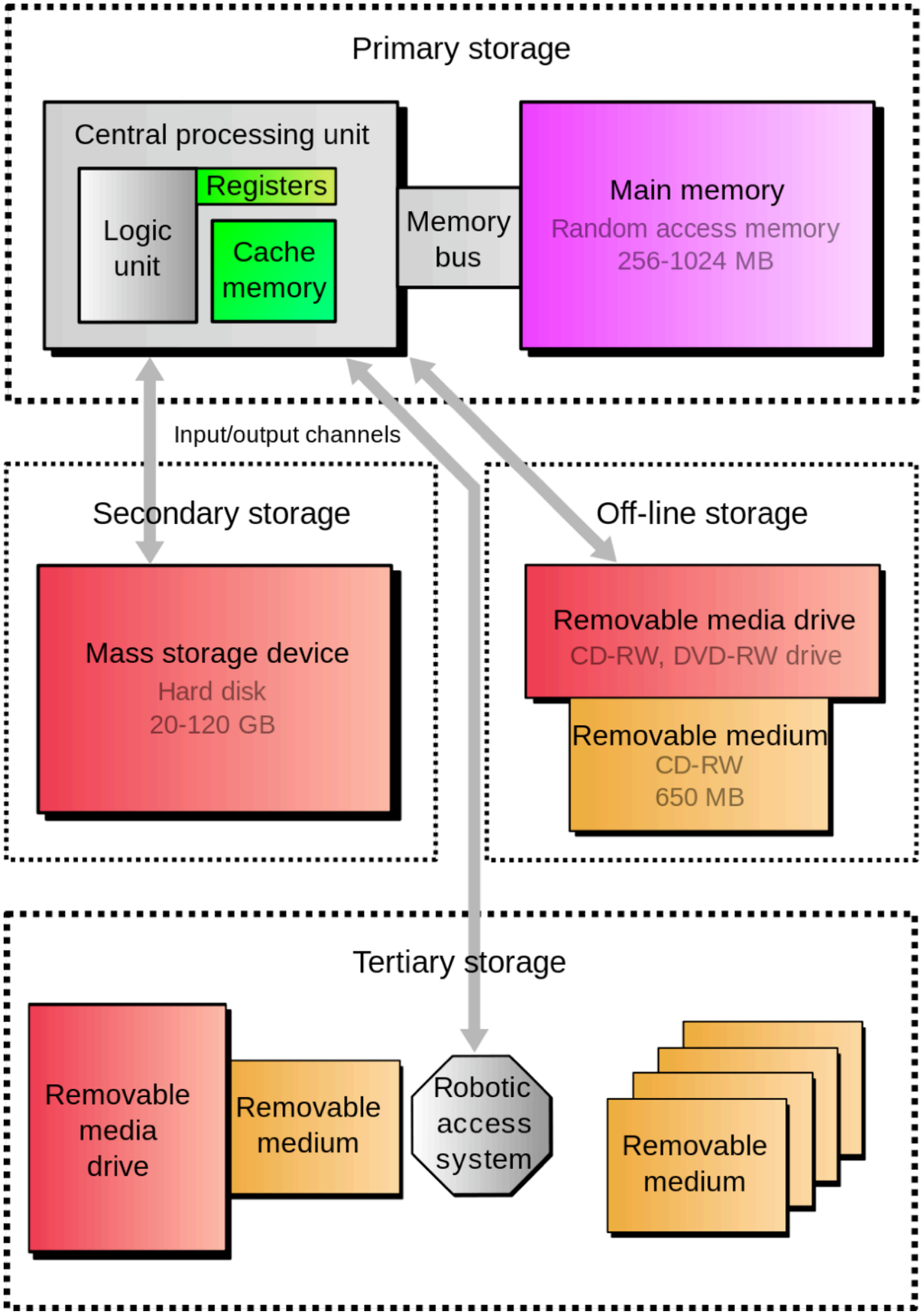
1DT003

2019-02-11 karl.marklund@it.uu.se Uppsala university

Hierarchy of computer data storage

- Primary storage
- Secondary storage
- Tertiary storage
- Off-line storage

Source https://en.wikipedia.org/wiki/Computer_data_storage#Secondary_storage



Directly accessible to the CPU

Not directly accessible to the CPU

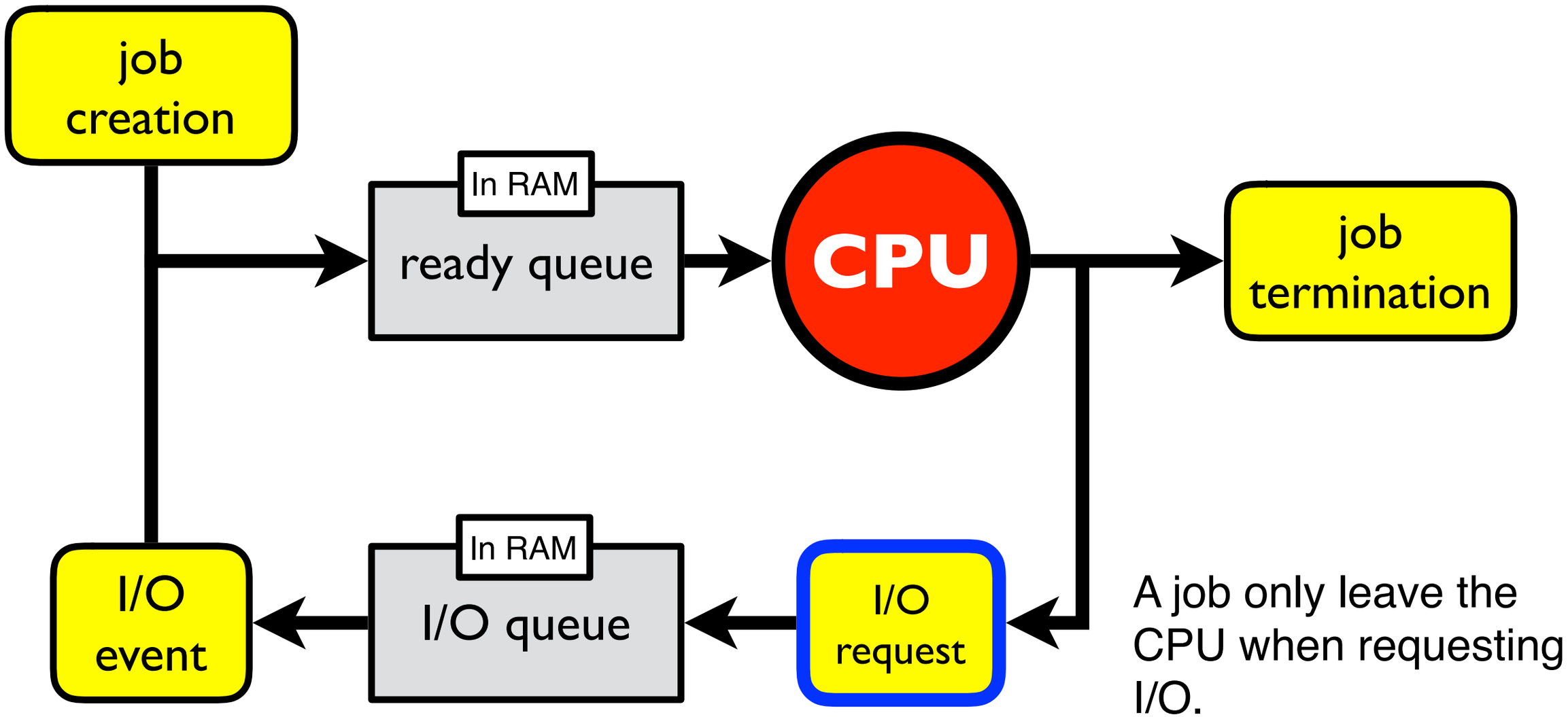
Primarily used for archiving rarely accessed information.

Typically involves a robotic mechanism which will mount(insert) and dismount removable mass storage media into a storage device. Data often copied to secondary storage before use.

Source https://en.wikipedia.org/wiki/Computer_data_storage#Secondary_storage

Multiprogramming

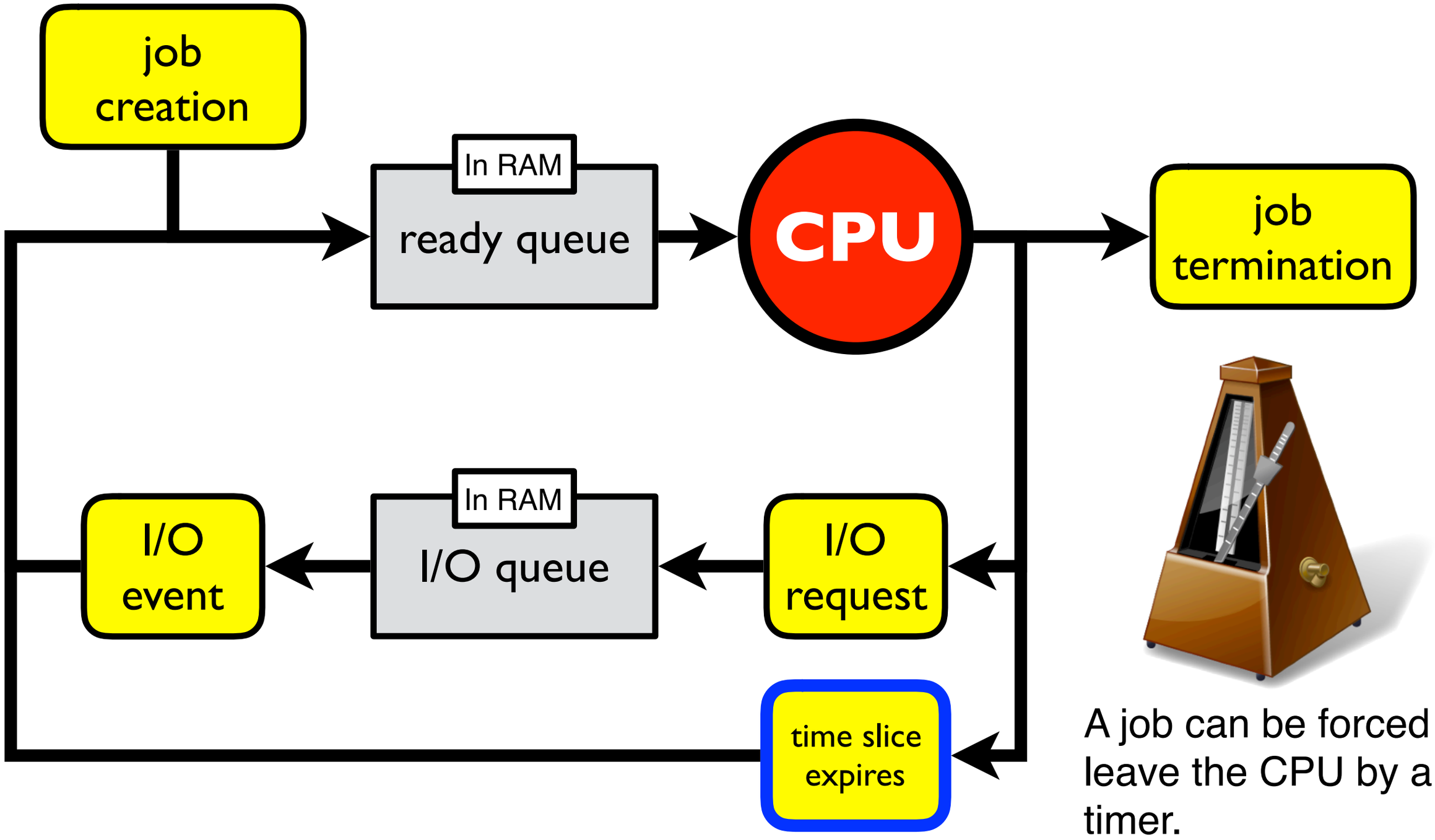
A schematic view of multiprogramming



A job only leave the CPU when requesting I/O.

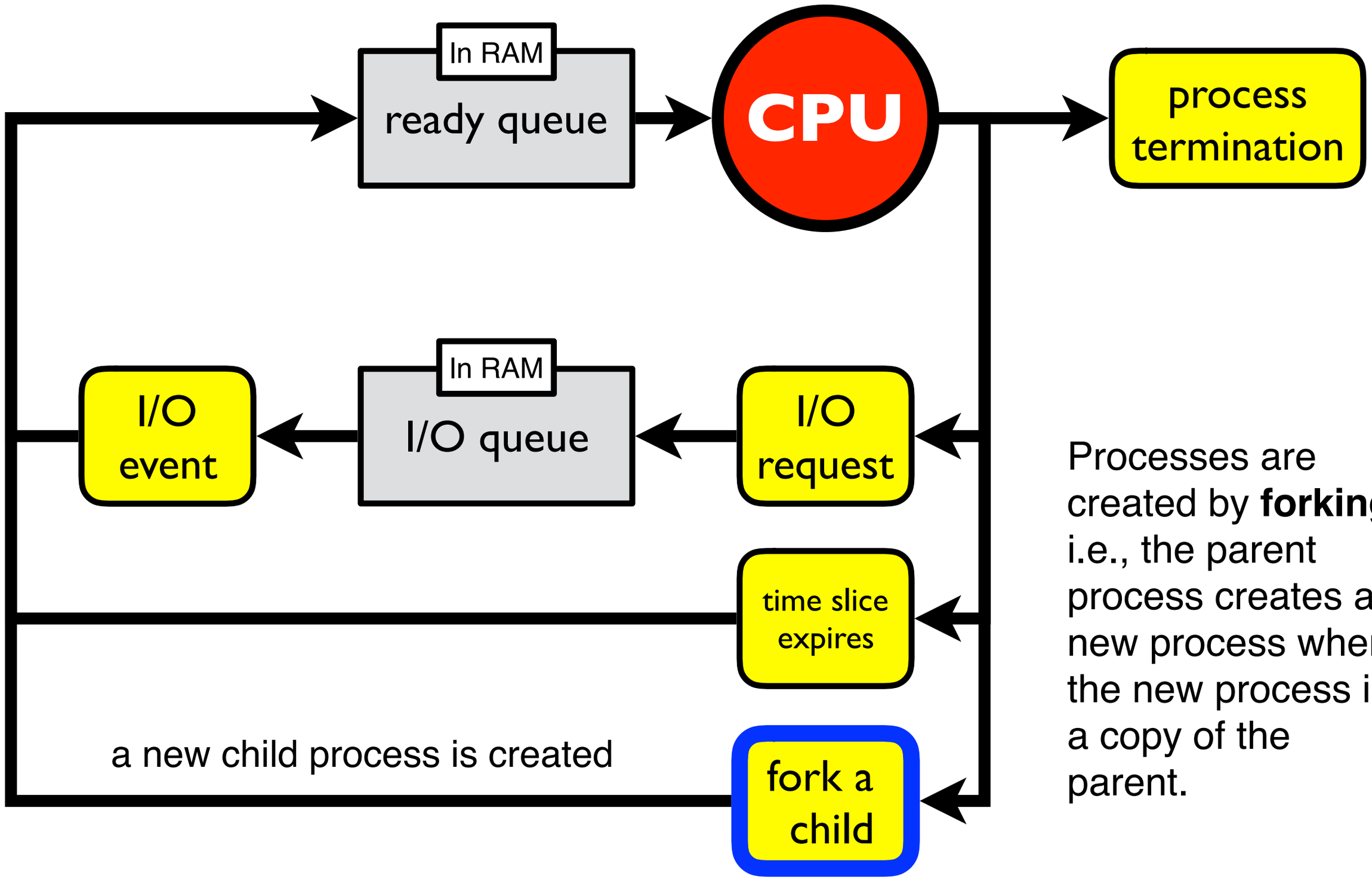
Multitasking

A schematic view of multitasking



A job can be forced to leave the CPU by a timer.

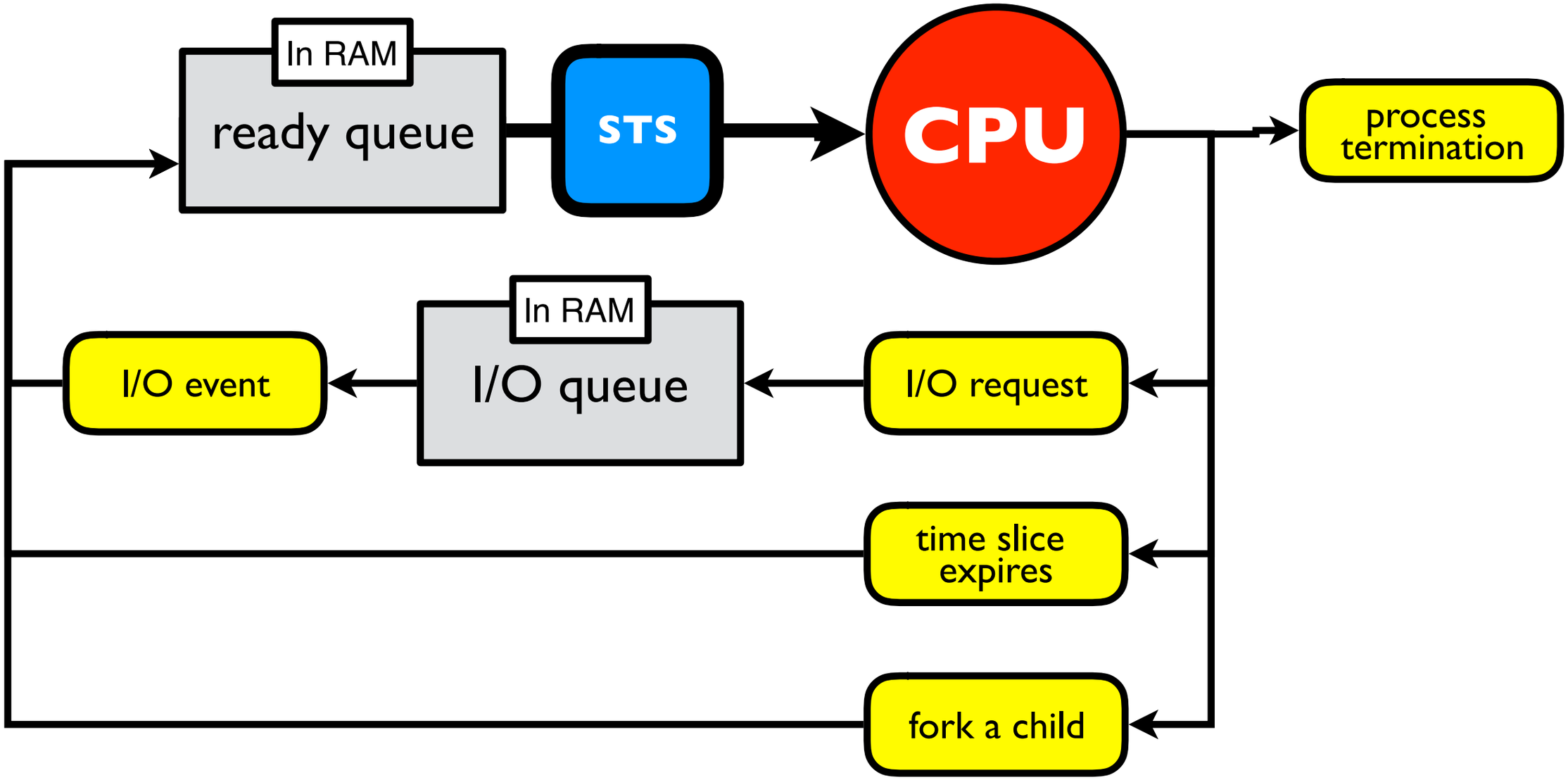
Process creation



Processes are created by **forking**, i.e., the parent process creates a new process where the new process is a copy of the parent.

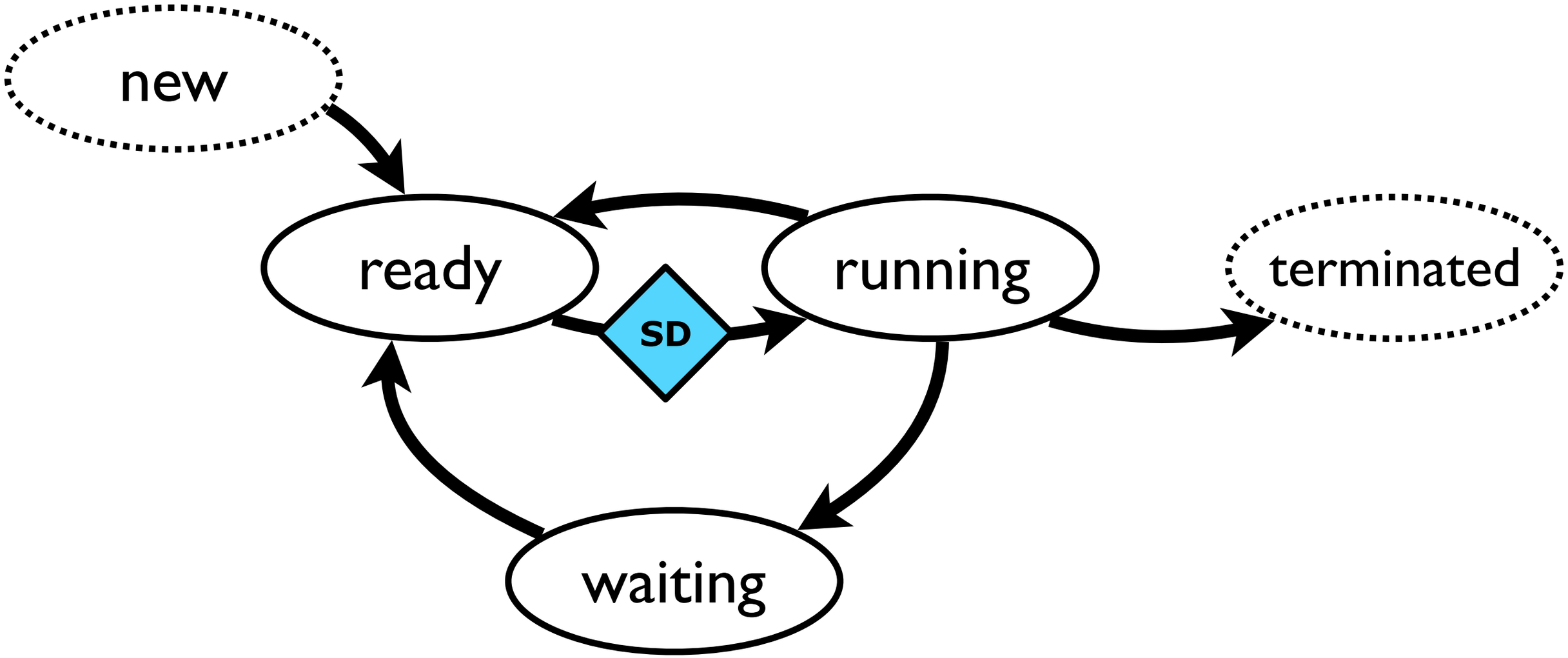
Short-term scheduler

The Short-term scheduler (STS), aka CPU scheduler, selects which process in the in memory ready queue that should be executed next and allocates CPU.



Scheduler dispatch

The CPU scheduler selects one process from among the processes in memory that are READY to execute. The scheduler dispatcher then gives the selected process control of the CPU. This action is called scheduler dispatch (SD).



Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:

- ★ switching context
- ★ switching to user mode
- ★ jumping to the proper location in the user program to resume execution of that program.

Dispatch latency: time it takes for the dispatcher to stop one process and start another.

Process Control Block (PCB)

Process Control Block (PCB)

The process control block (PCB) is a data structure in the operating system kernel containing the information needed to manage a particular process.

Source https://en.wikipedia.org/wiki/Process_control_block 2018-01-21

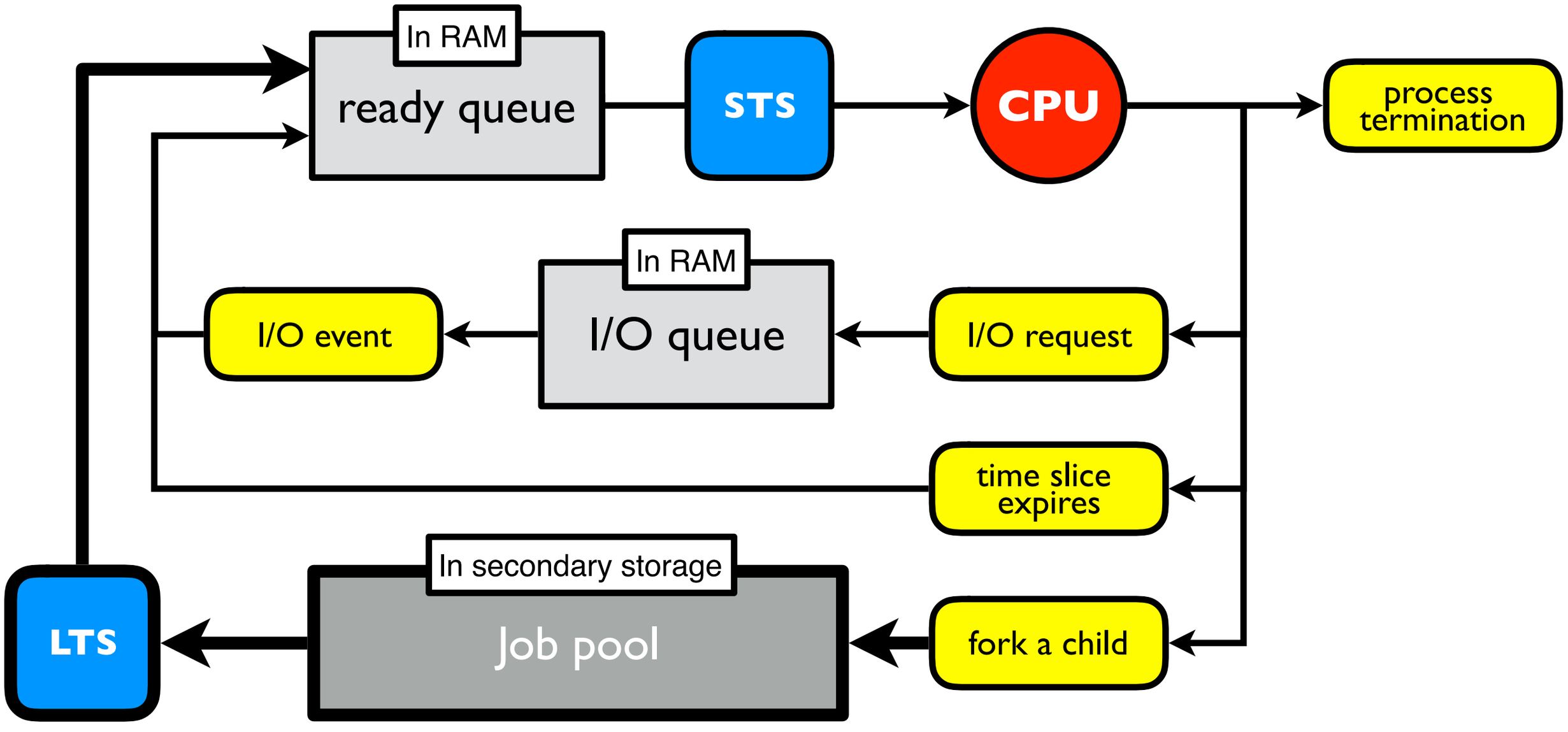
In brief, the PCB serves as the repository for any information that may vary from process to process.

Example of information stored in the PCB

Process Control Block (PCB)
Process id (PID)
Process state (new, ready, running, waiting or terminated)
CPU Context
I/O status information
Memory management information
CPU scheduling information

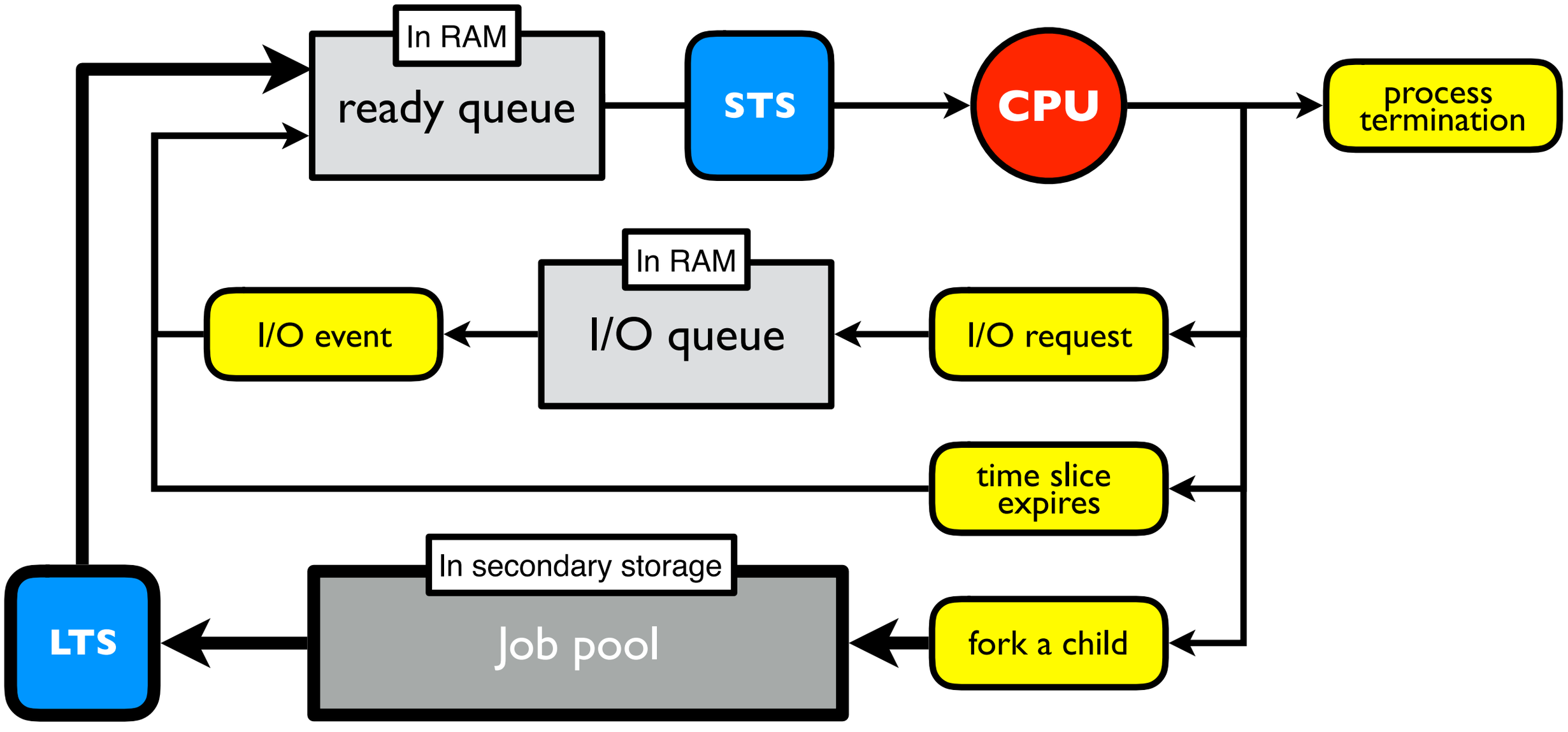
Long-term scheduler (1)

The Long-term scheduler (LTS) (aka job scheduler) decides whether a new process should be brought into the ready queue in main memory or delayed. When a process is ready to execute, it is added to the job pool (on disk). When RAM is sufficiently free, some processes are brought from the job pool to the ready queue (in RAM).



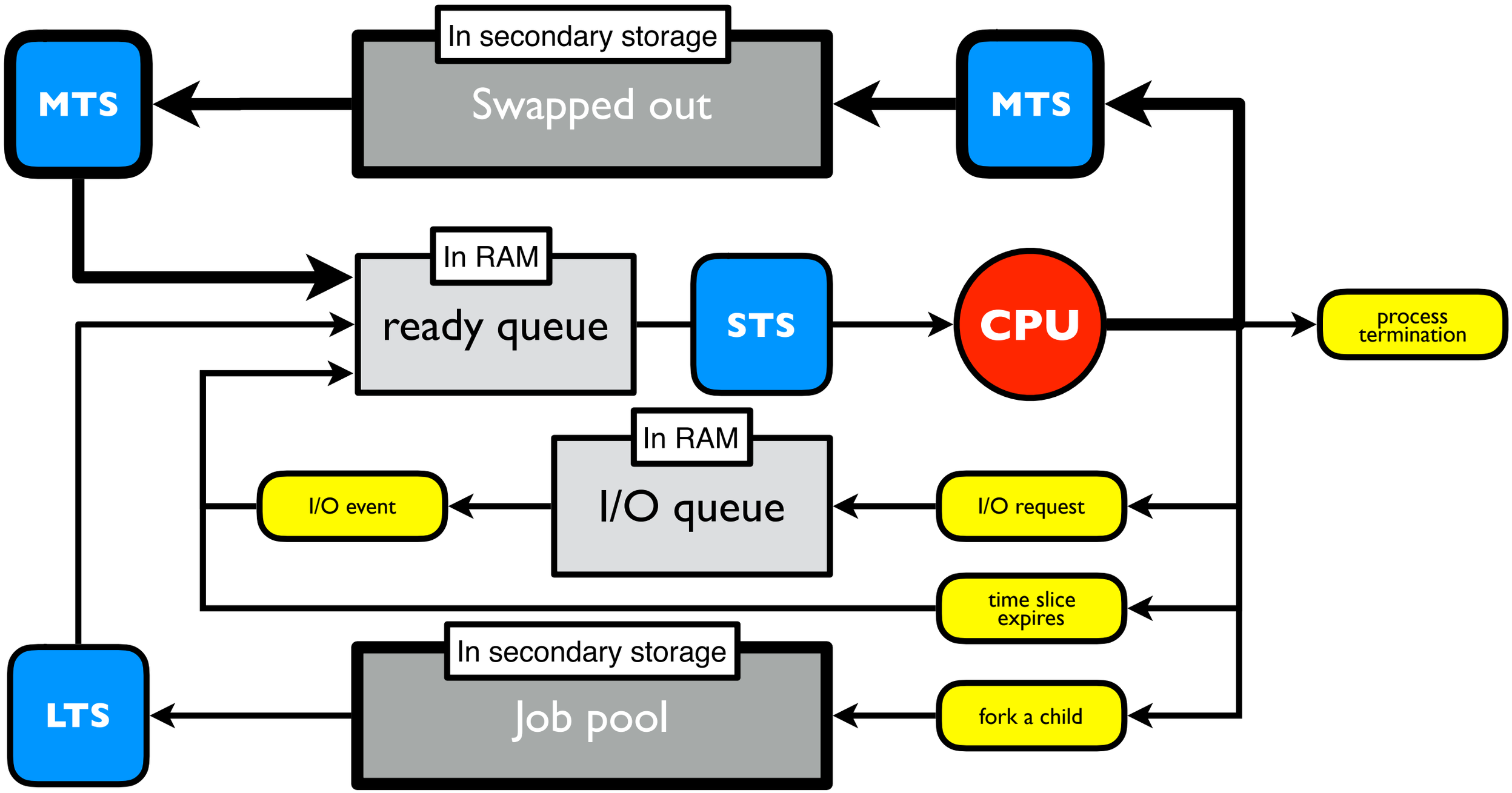
Long-term scheduler (2)

On some systems, the long-term scheduler may be absent or minimal. For example, time-sharing systems such as UNIX and Microsoft Windows systems often have no long-term scheduler but simply put every new process in memory for the short-term scheduler.



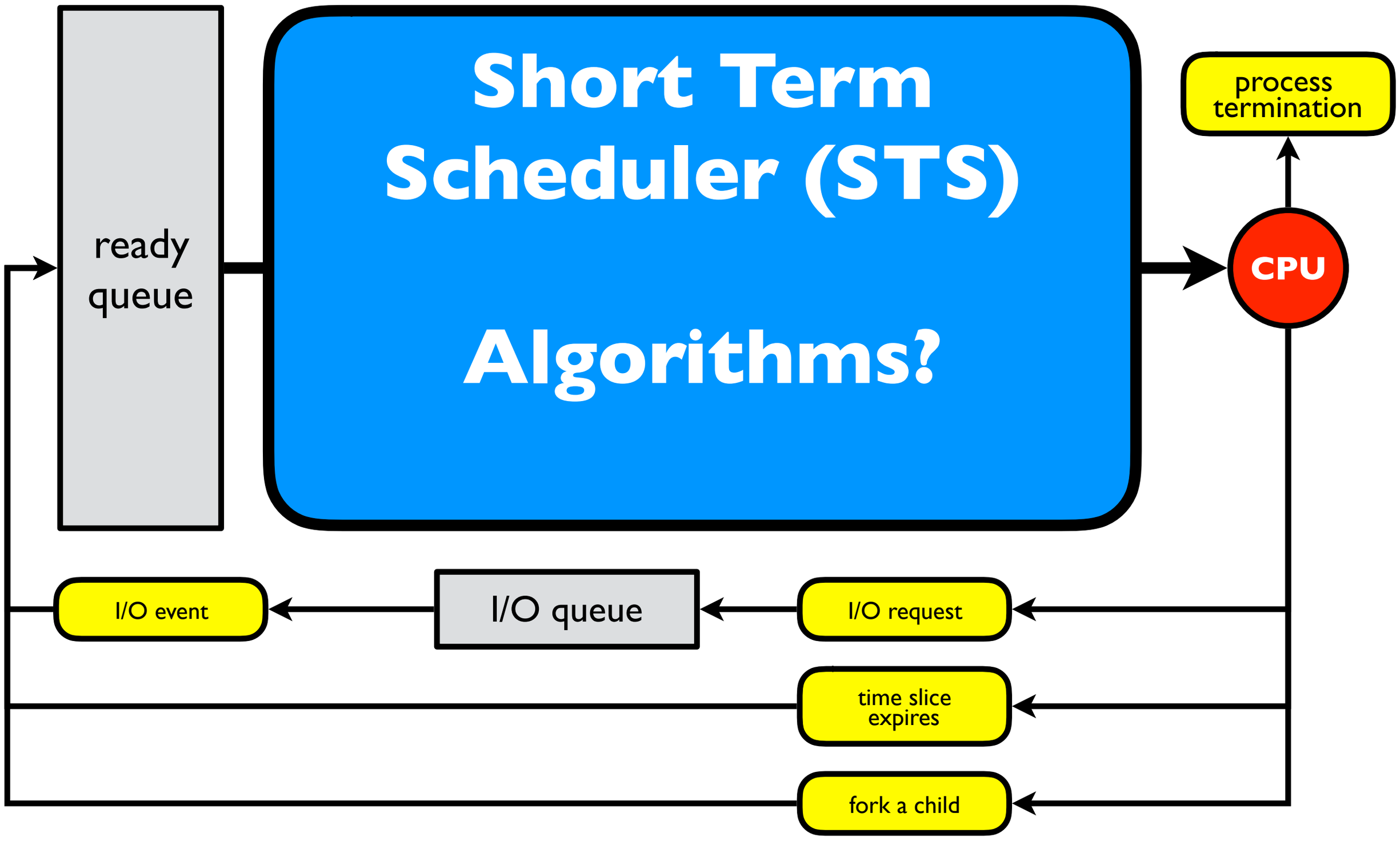
Medium-term scheduler

The medium-term scheduler (MTS) temporarily removes processes from main memory and places them in secondary storage and vice versa, which is commonly referred to as "swapping in" and "swapping out".



Scheduling algorithms

Scheduling algorithms



Perfomance

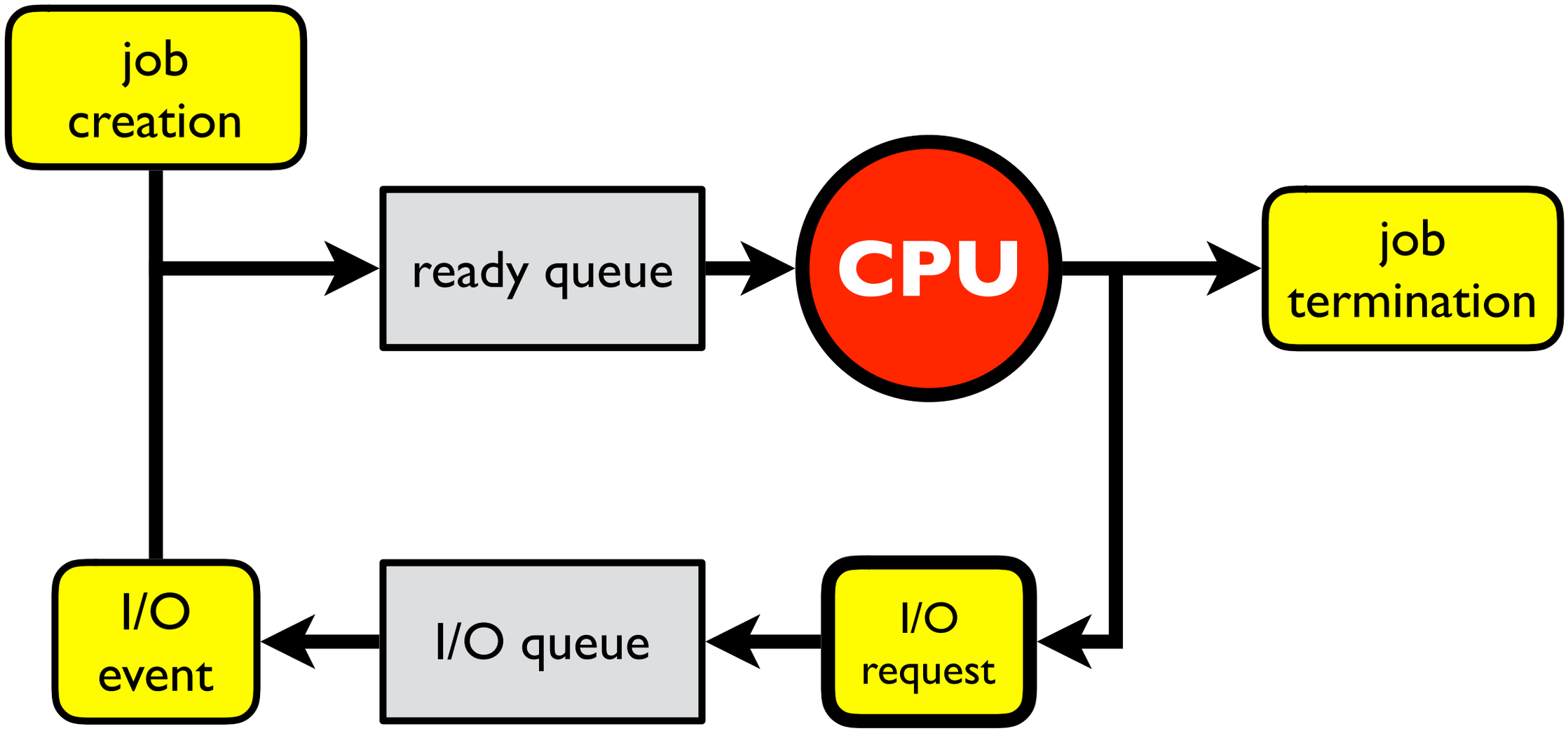
Performance is a context dependent metric.

What do we mean by performance?

Scheduling criteria

Multiprogramming

Multiprogramming **maximises** **CPU utilisation**.



Scheduling criteria

CPU utilisation is not the only criteria ...

Scheduling criteria

Criteria	Definition	Goal
CPU utilization	The % of time the CPU is executing user level process code.	Maximize
Throughput	Number of processes that complete their execution per time unit.	Maximize
Turnaround time	Amount of time to execute a particular process.	Minimize
Waiting time	Amount of time a process has been waiting in the ready queue.	Minimize
Response time	Amount of time it takes from when a request was submitted until the first response is produced.	Minimize

All animals are equal, but some animals are more equal than others.

George Orwell
Animal farm (1945)

(An example of political satire)

Are all processes equal, or are some processes more equal than others?

Karl Marklund

Operating systems 2018

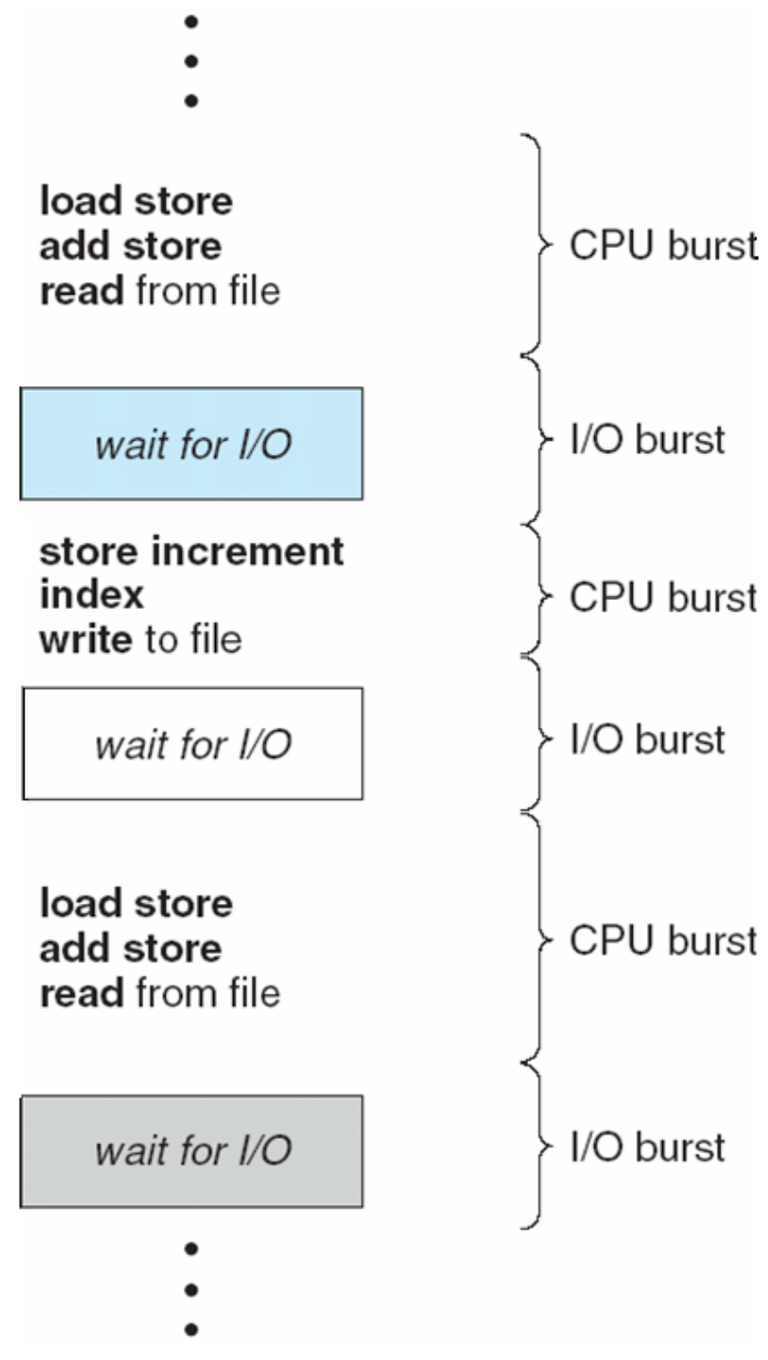
Classification of processes

Do all processes behave the same?

Do all processes have the same needs?

CPU bursts and I/O bursts

When a program executes it alternates between CPU bursts and I/O bursts.



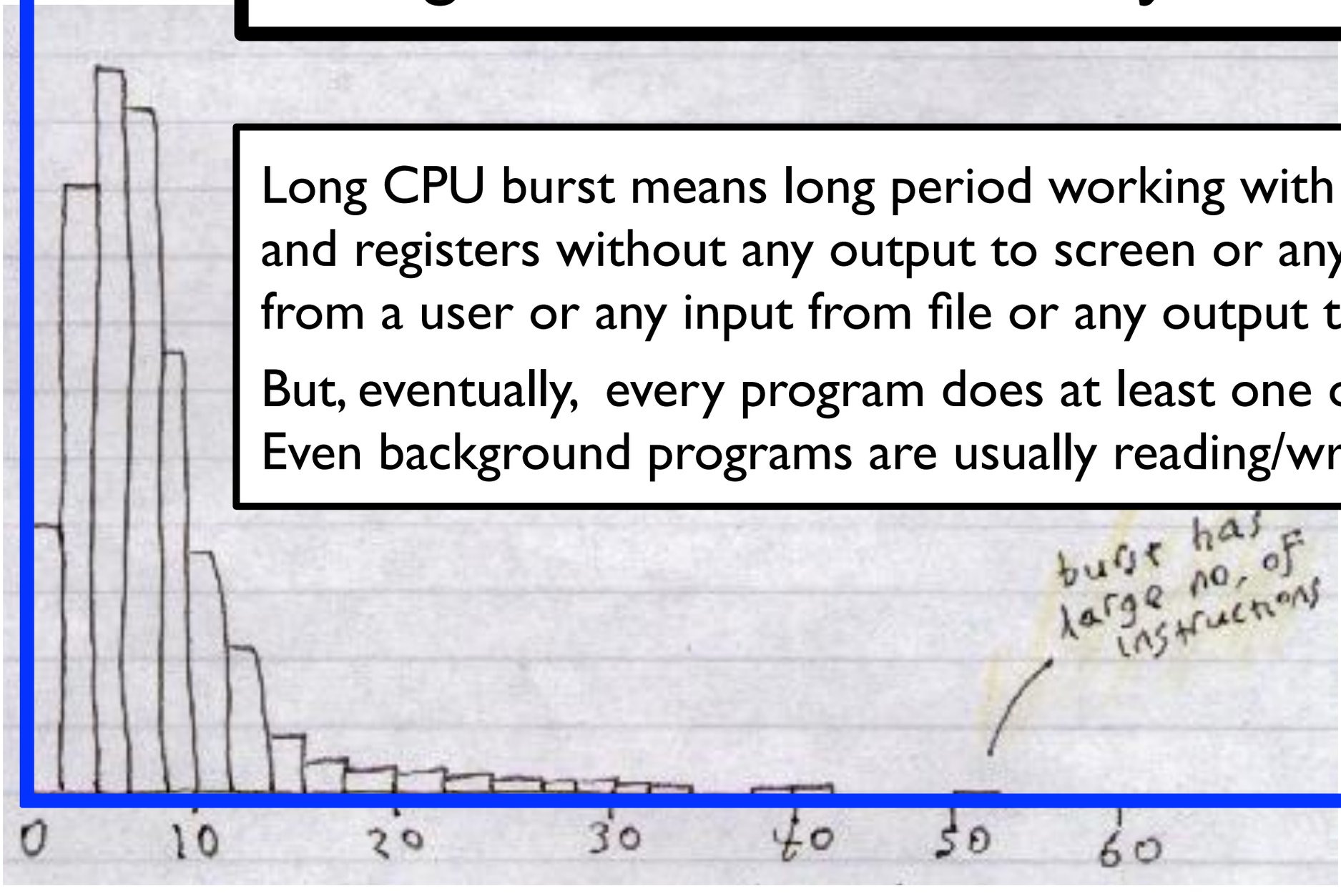
Histogram of CPU-burst times

Number of CPU bursts

Long CPU burst are very rare - why?

Long CPU burst means long period working with memory and registers without any output to screen or any input from a user or any input from file or any output to file.

But, eventually, every program does at least one of these.
Even background programs are usually reading/writing files.

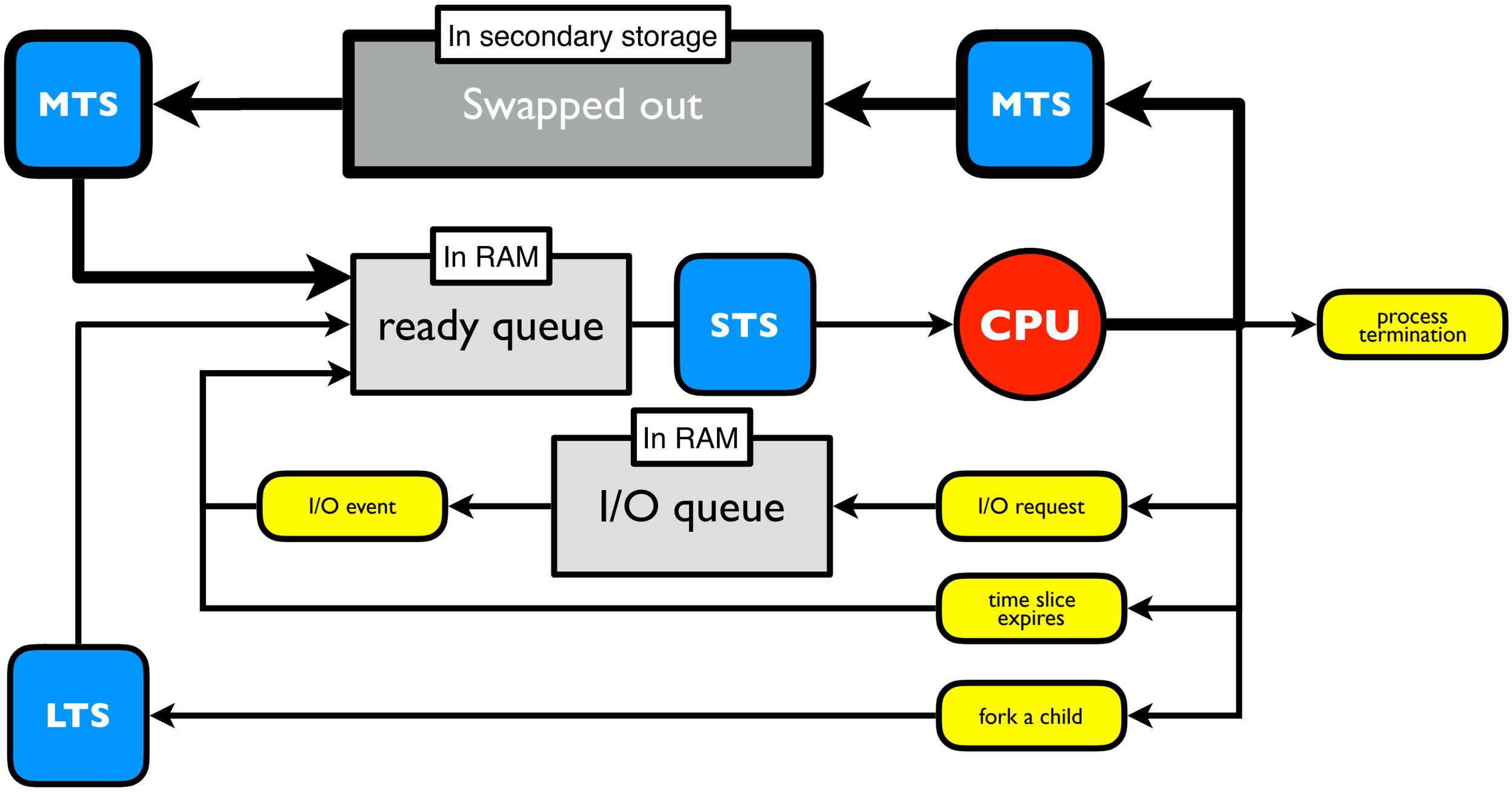


An **I/O-bound** process spends more time doing I/O than computations and is characterised by **many short CPU bursts**.

An **CPU-bound** process spends more time doing computations and is characterised by **few very long CPU bursts**.

I/O bound and CPU bound processes

The **medium-term scheduler** (MTS) can be used to maintain a good balance between I/O bound and CPU bound processes in the ready queue.



Human user

Human user

Process A

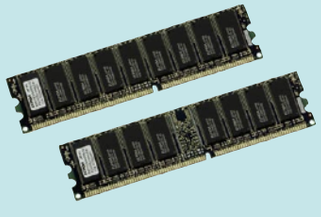
Process B

Not all processes interacts with human users.

Process Z

The operating systems **controls** the **hardware** and **coordinates** its use among the various application programs for the various user. An operating system provides an **environment for the execution of programs**.

Computer hardware



Classification of processes

In general, processes can be classified by the following characteristics.

- ★ Interactive
- ★ Batch
- ★ Real-Time
- ★ I/O Bound
- ★ CPU Bound

Interactive

Interactive processes interact constantly with their human users.

- ★ Spend a lot of time waiting for keypresses and mouse operations.
- ★ When input is received, the process must be woken up quickly, or the user will find the system to be unresponsive.
- ★ Typically, the average delay must fall between 50 and 150 ms. The variance of such delay must also be bounded, or the user will find the system to be erratic.

Typical examples:

- ★ Command shells and interpreters, text editors, graphical applications and games.

Batch

Batch processes do not interact with human users.

- ★ Do not need to be responsive.
- ★ Often run in the background.
- ★ Often penalised by the scheduler.

Typical examples:

- ★ Compilers, database search engines, and scientific computations.

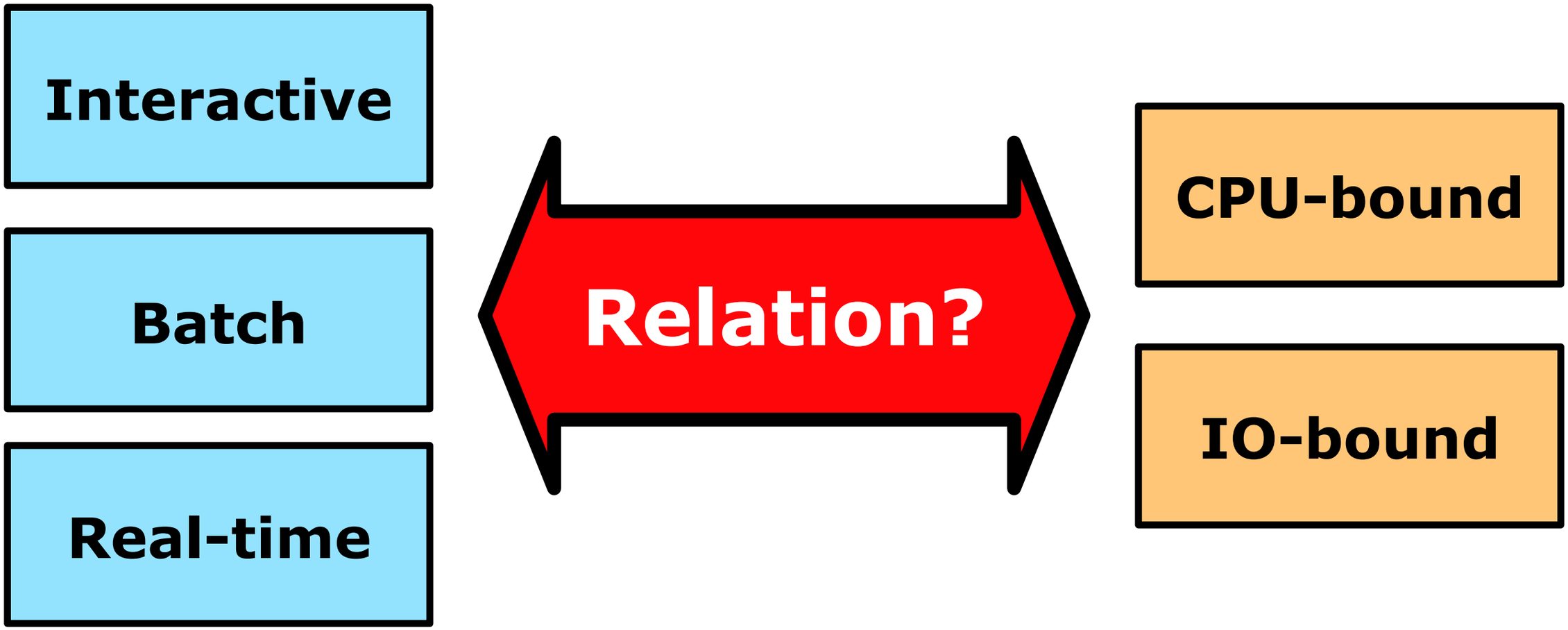
Real-time

Real-time processes have very strong scheduling requirements.

- ★ Such processes should never be blocked by lower-priority processes.
- ★ Should have a short response time.
- ★ Most important, response time should have a minimum variance.

Typical examples:

- ★ Video and sound applications, robot controllers, and programs that collect data from physical sensors.

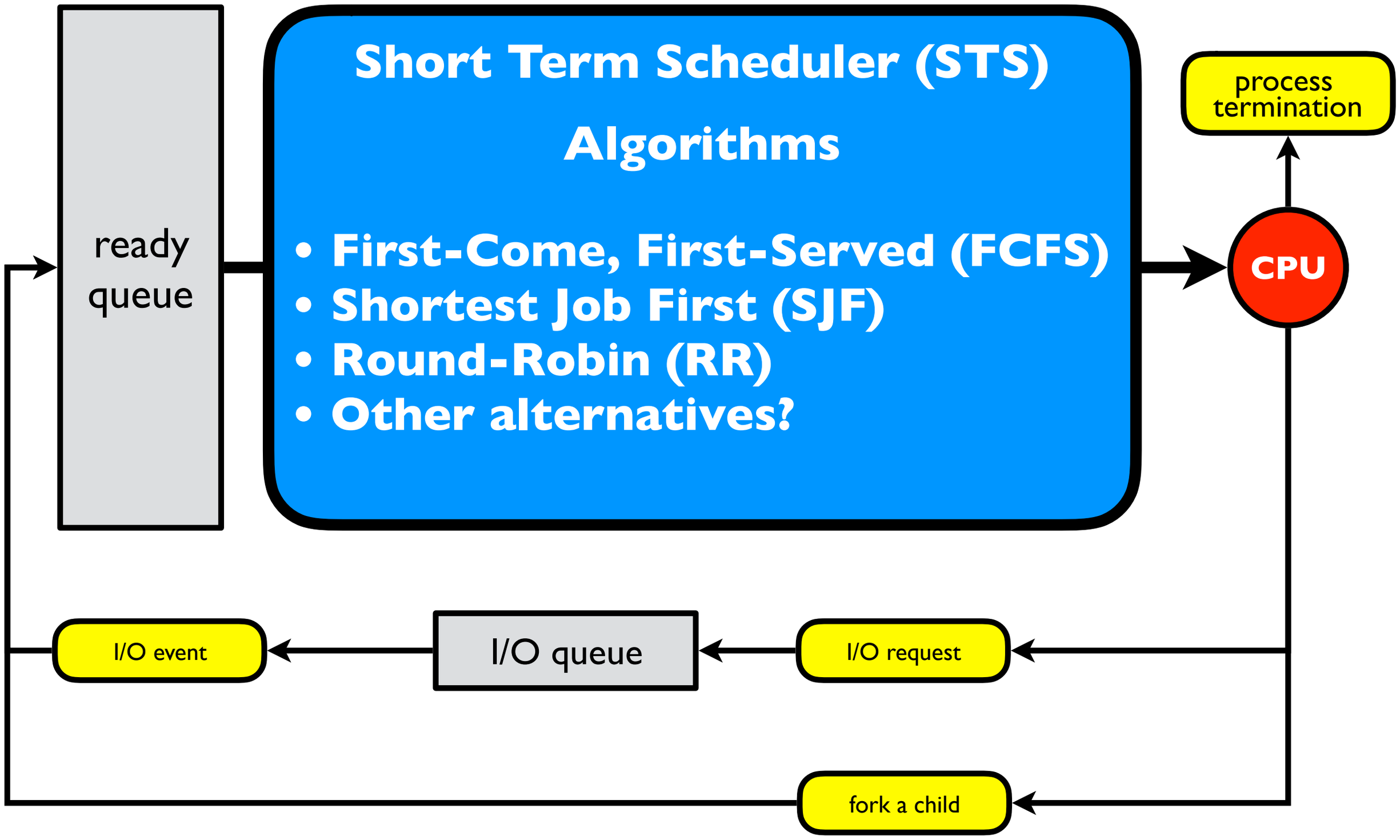


The two classifications above are somewhat independent. For instance, a batch process can be either I/O-bound (e.g., a database server) or CPU-bound (e.g., an image-rendering program).

In general, there is no way to distinguish between interactive and batch programs. In order to offer a good response time to interactive applications, Linux (like all Unix kernels) implicitly favours I/O-bound processes over CPU-bound ones.

Scheduling algorithms

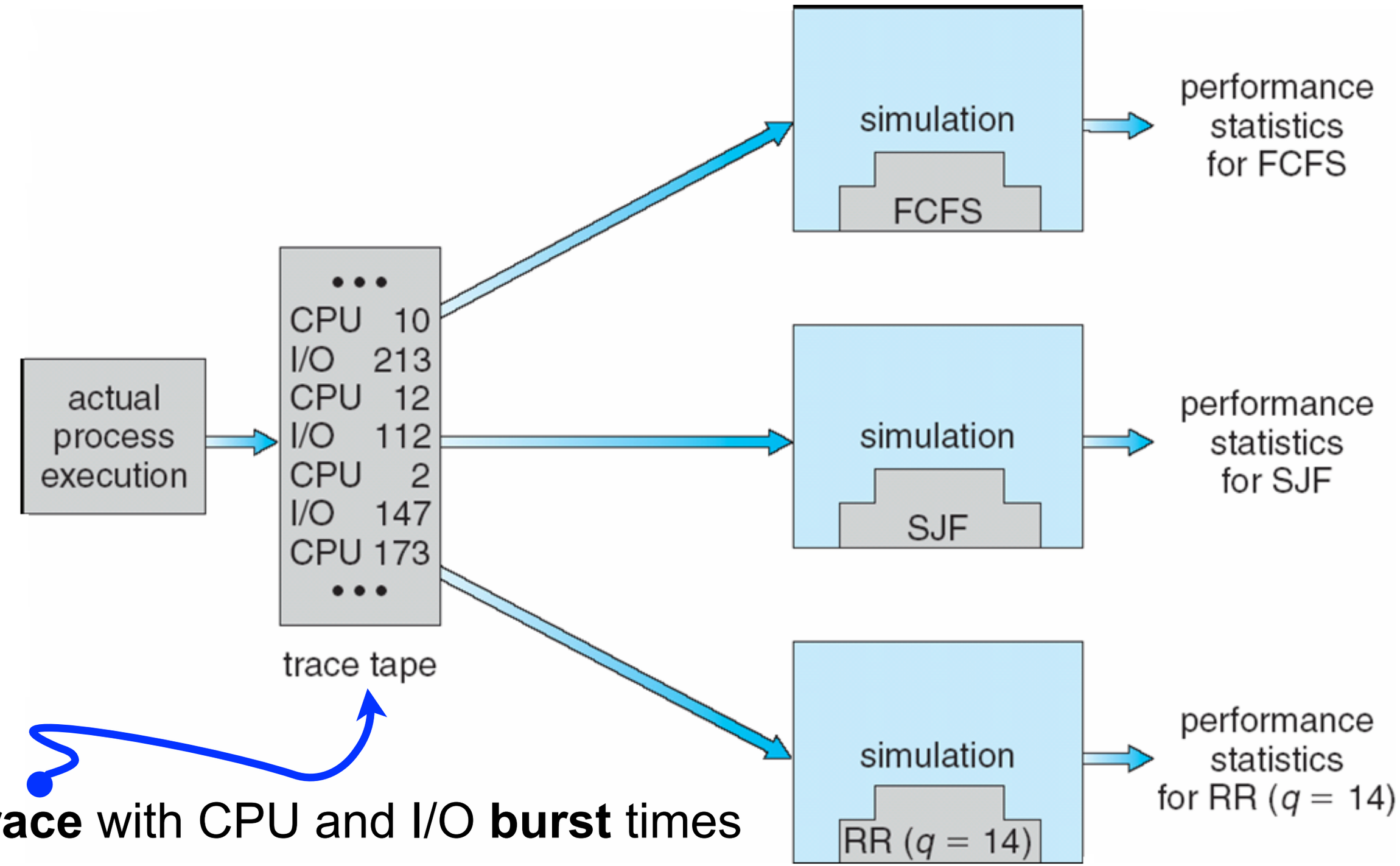
Scheduling algorithms



Evaluation of CPU scheduling algorithms

Evaluation of CPU schedulers by simulation

To evaluate different CPU scheduling algorithms, data obtained from instrumented executables can be used as input in simulations.

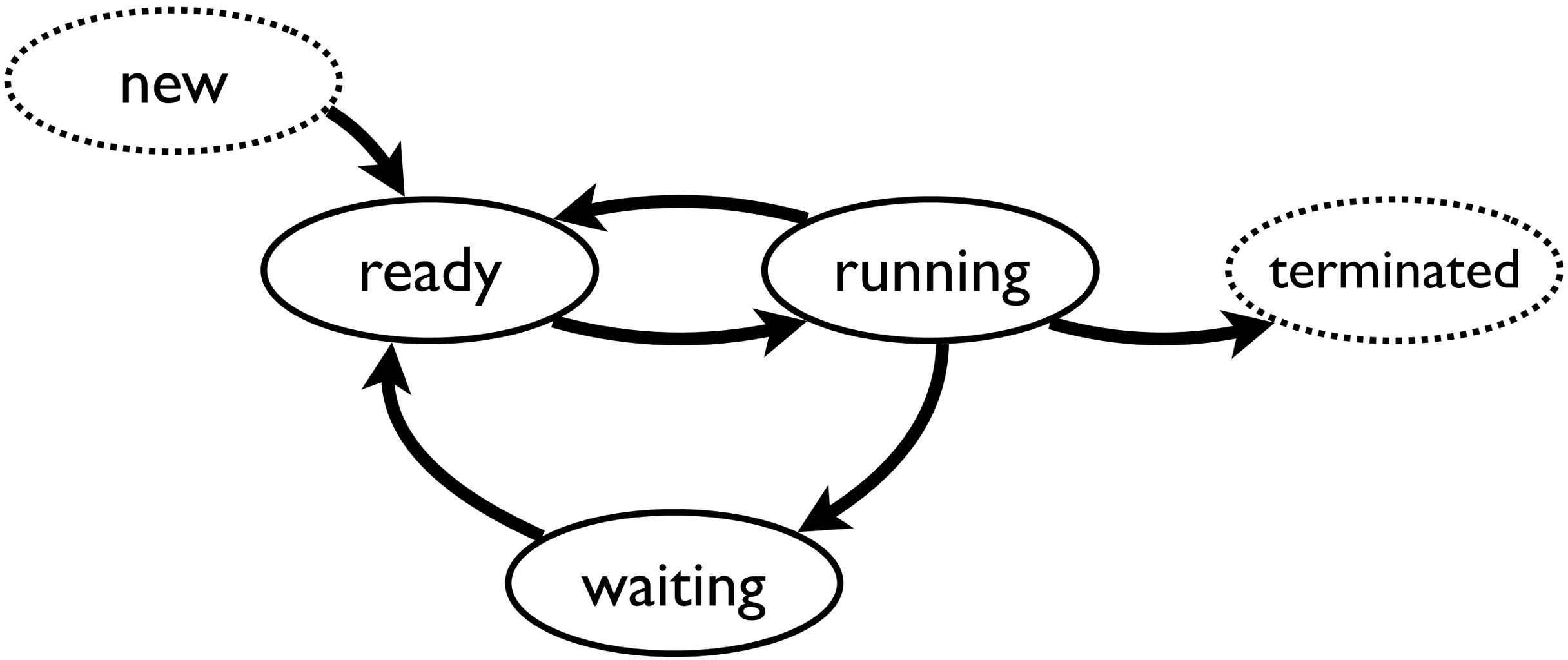


Trace with CPU and I/O **burst** times

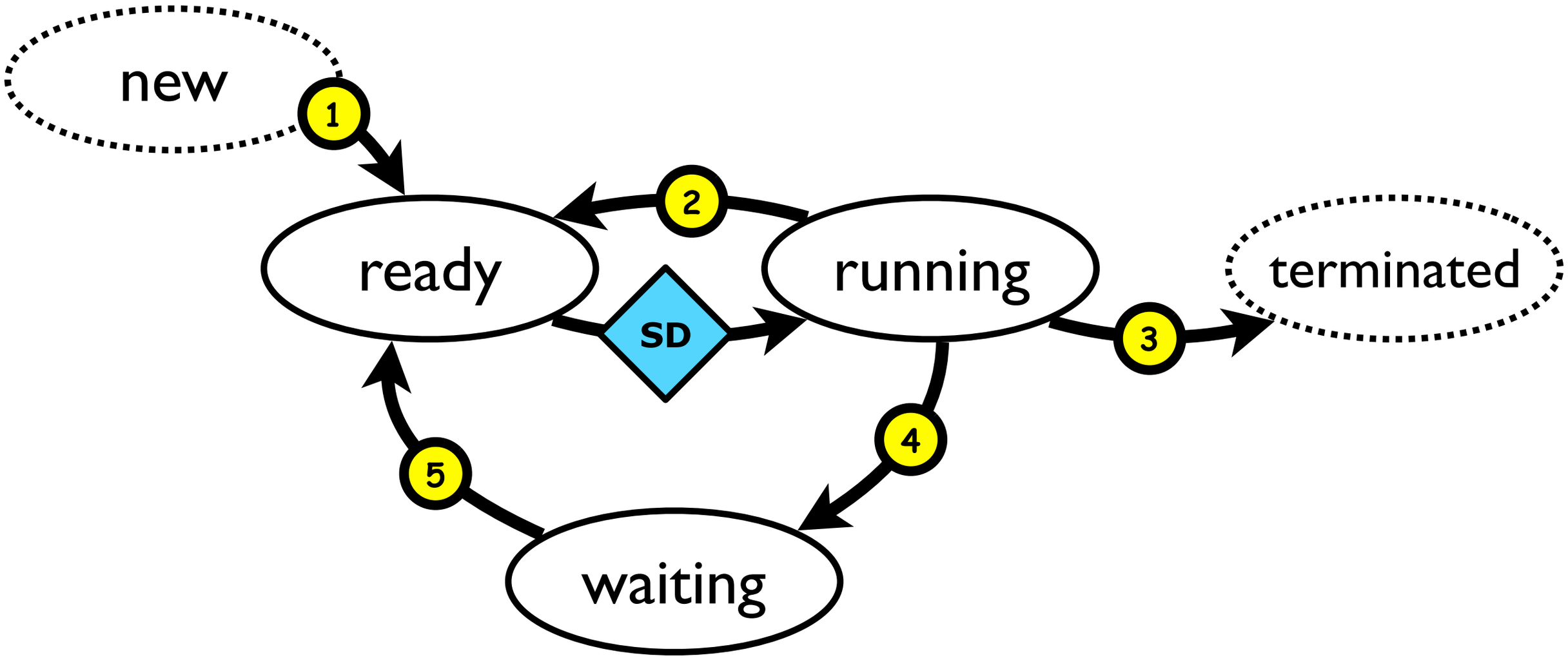
Model of CPU scheduling

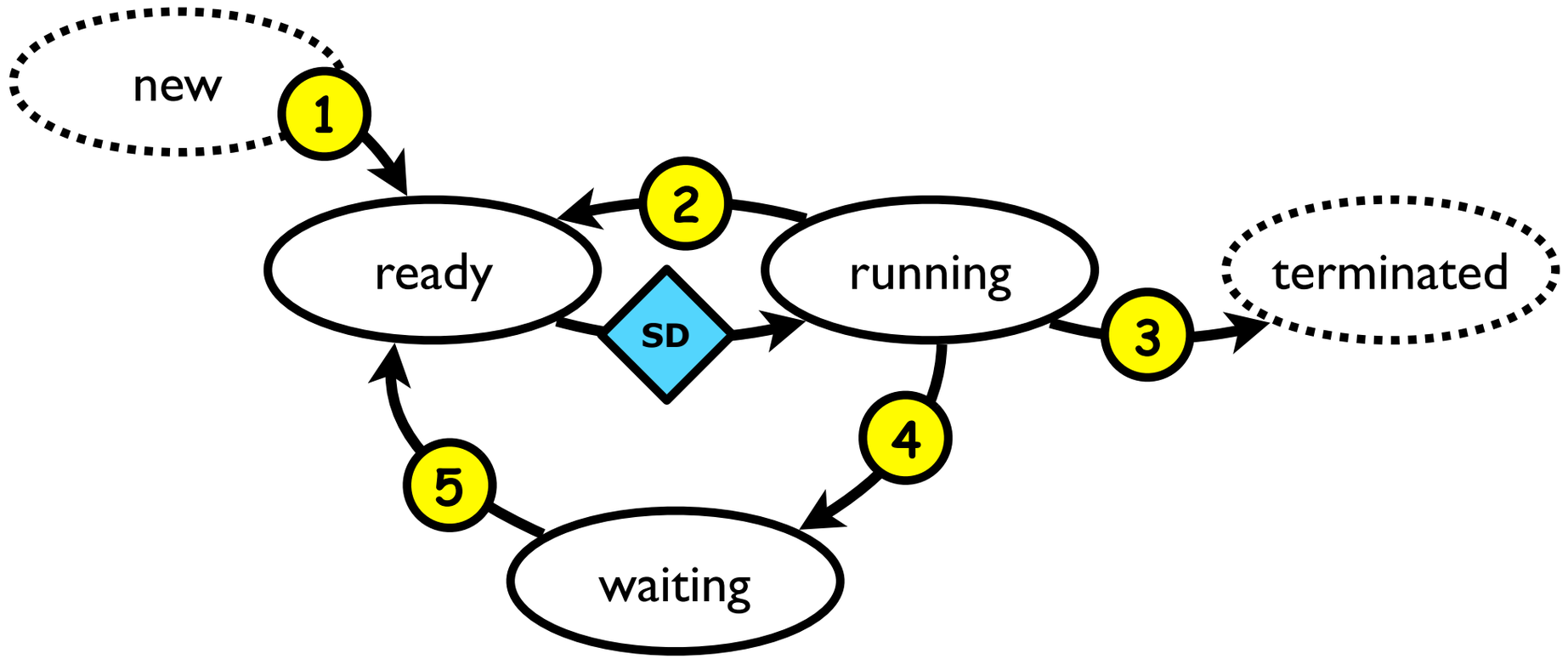
Simplified model of CPU scheduling

To make it easy to reason about CPU scheduling we will use a simplified **model**.



Events causing a scheduler dispatch

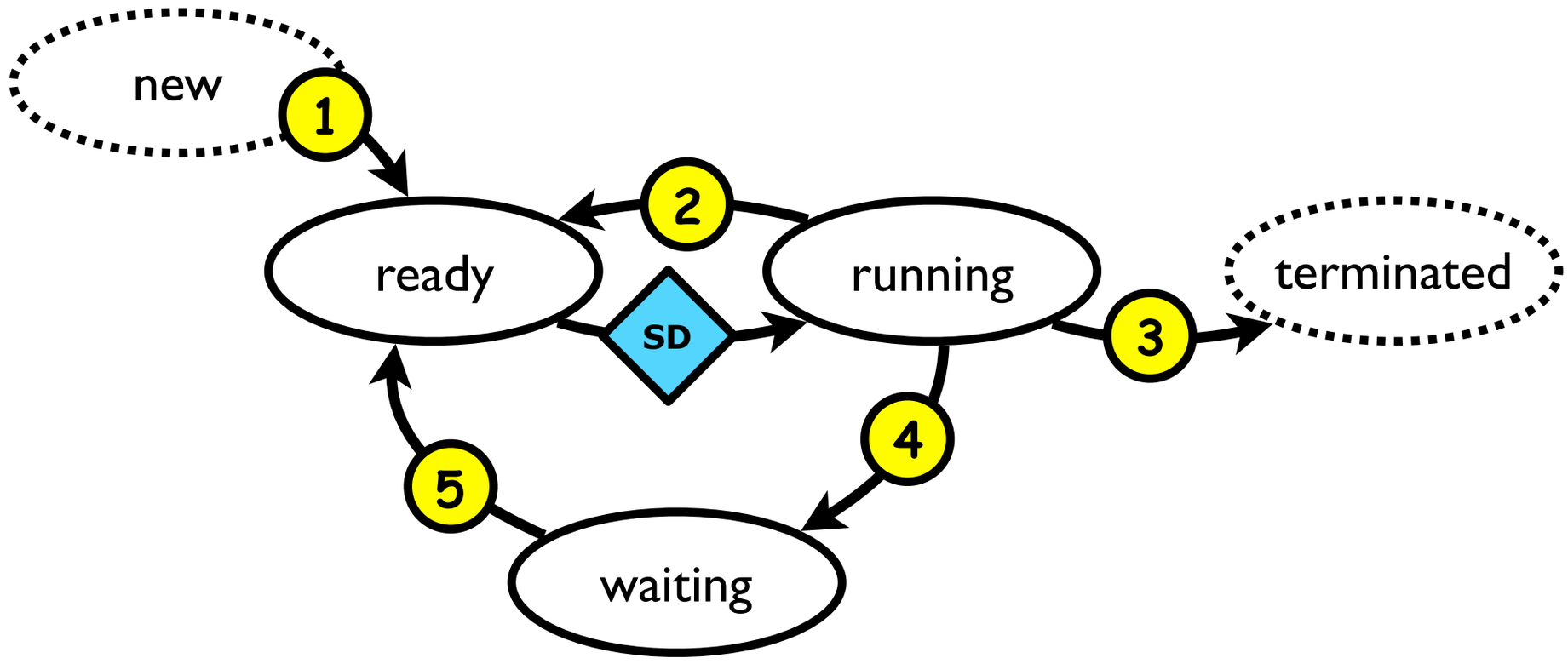




1. A new process arrives at the ready queue.
2. Time slice expires or other forms of **preemption**.
3. The running process terminates.
4. The running process requests I/O.
5. An I/O request completes.

Preemptive and nonpreemptive

Scheduler dispatch can be **preemptive** or **nonpreemptive**.

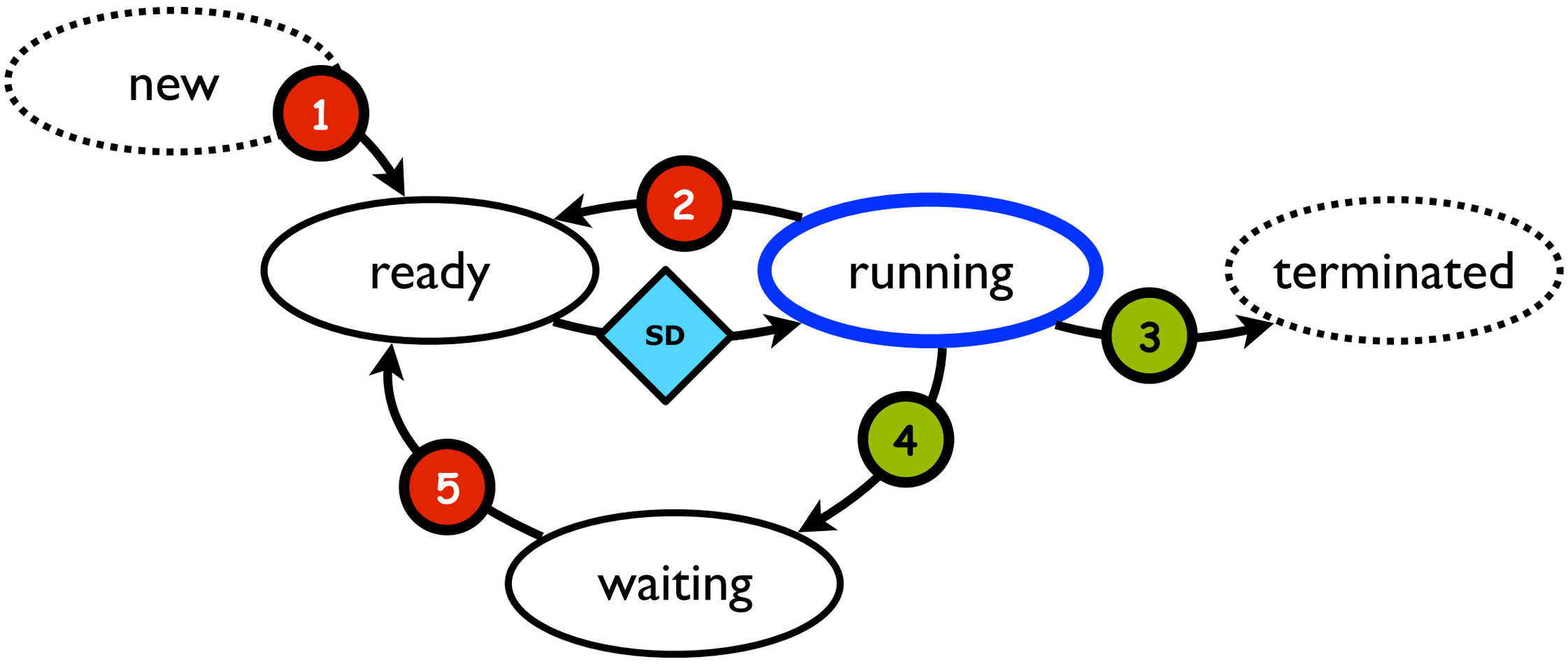


A **preemptive** dispatch is caused by an event external to the running running process.

A **nonpreemptive** dispatch is caused by the running process itself.

Preemptive and nonpreemptive

Scheduler dispatch can be **preemptive** or **nonpreemptive**.

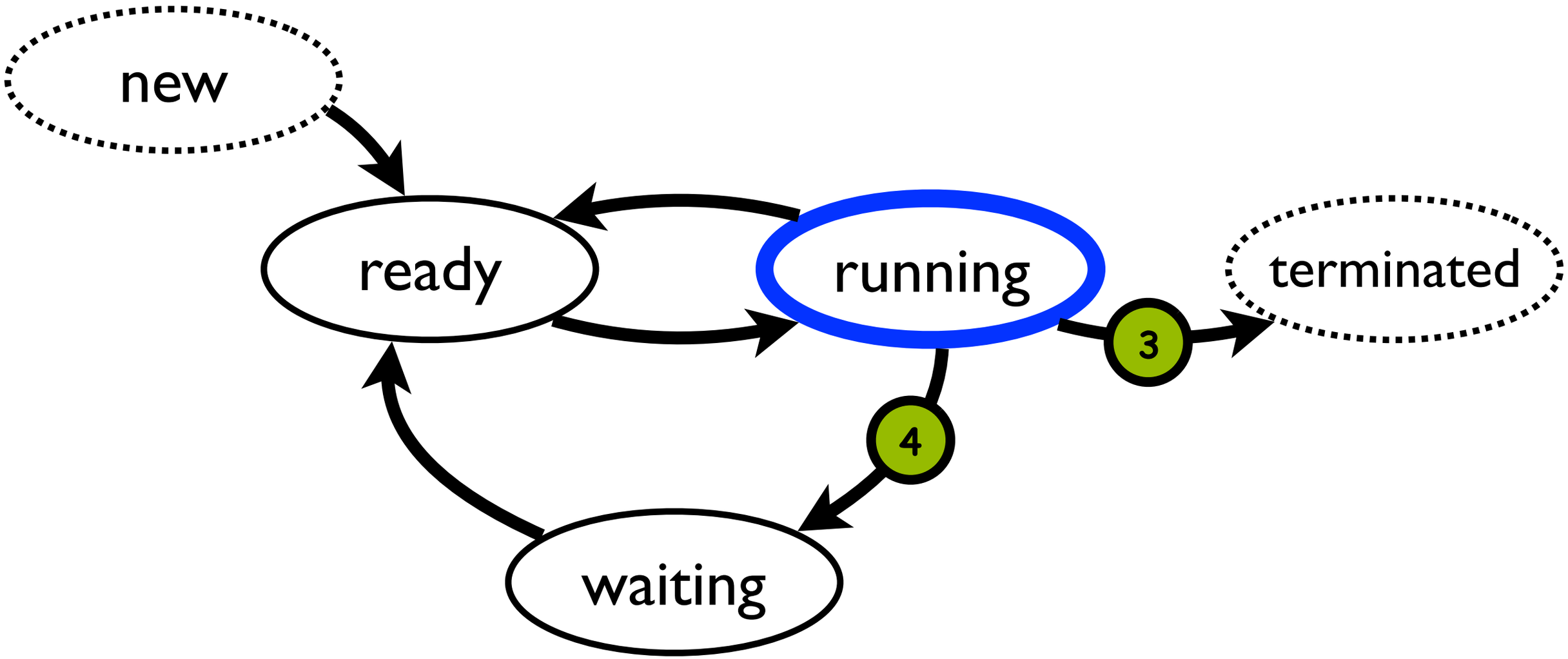


Events causing a **preemptive** scheduler dispatch: 1 2 5

Events causing a **nonpreemptive** scheduler dispatch: 3 4

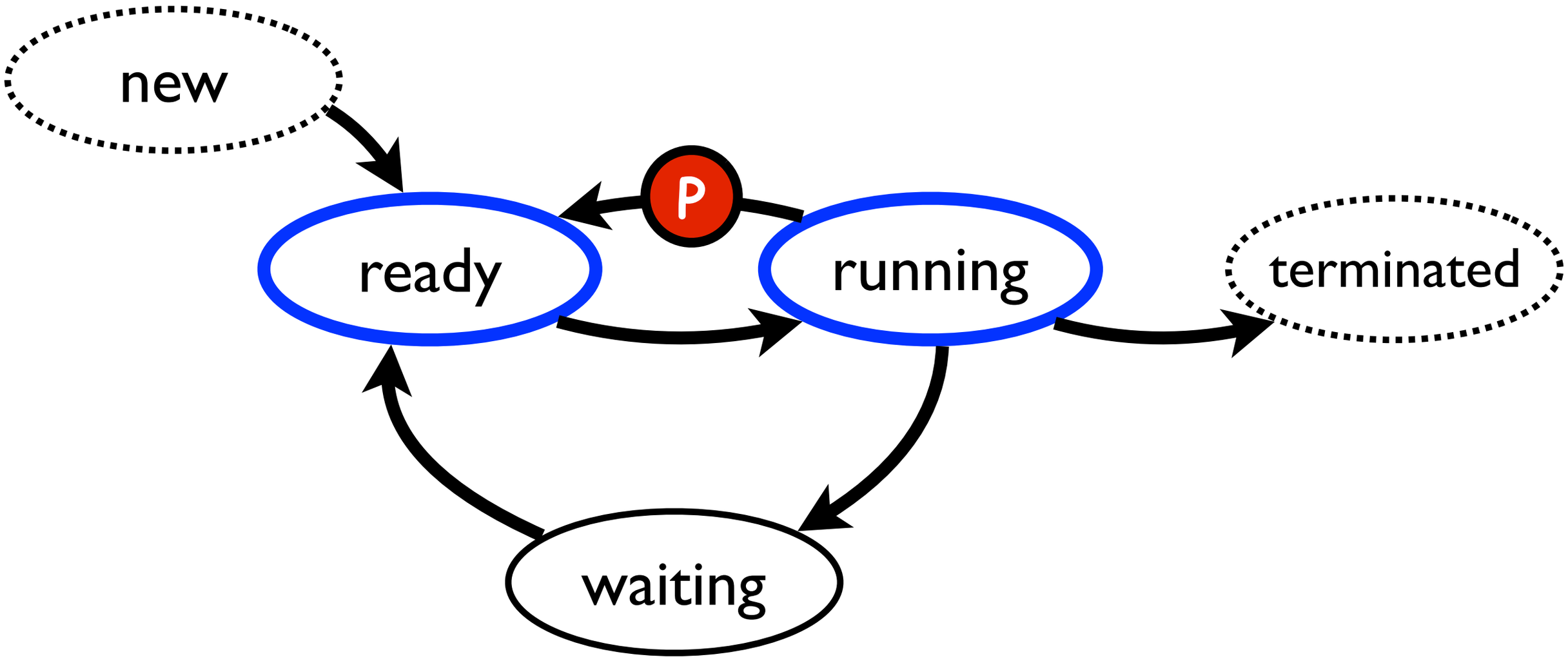
CPU burst

With CPU burst we mean the time spent by a running process using the CPU before (3) terminating or (4) performing a blocking system call.



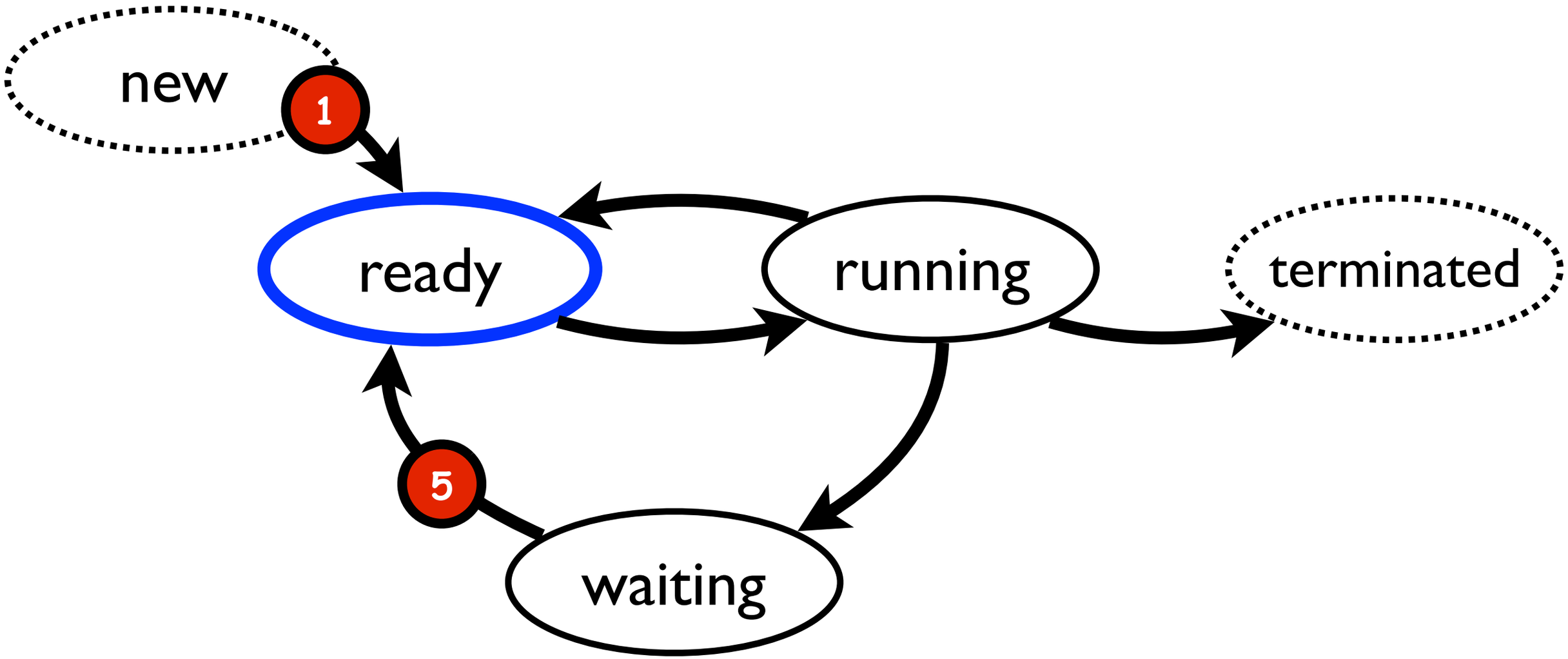
Preemption

A process may get preempted P, i.e., forced off the CPU and put back in the ready queue before completing its CPU burst.



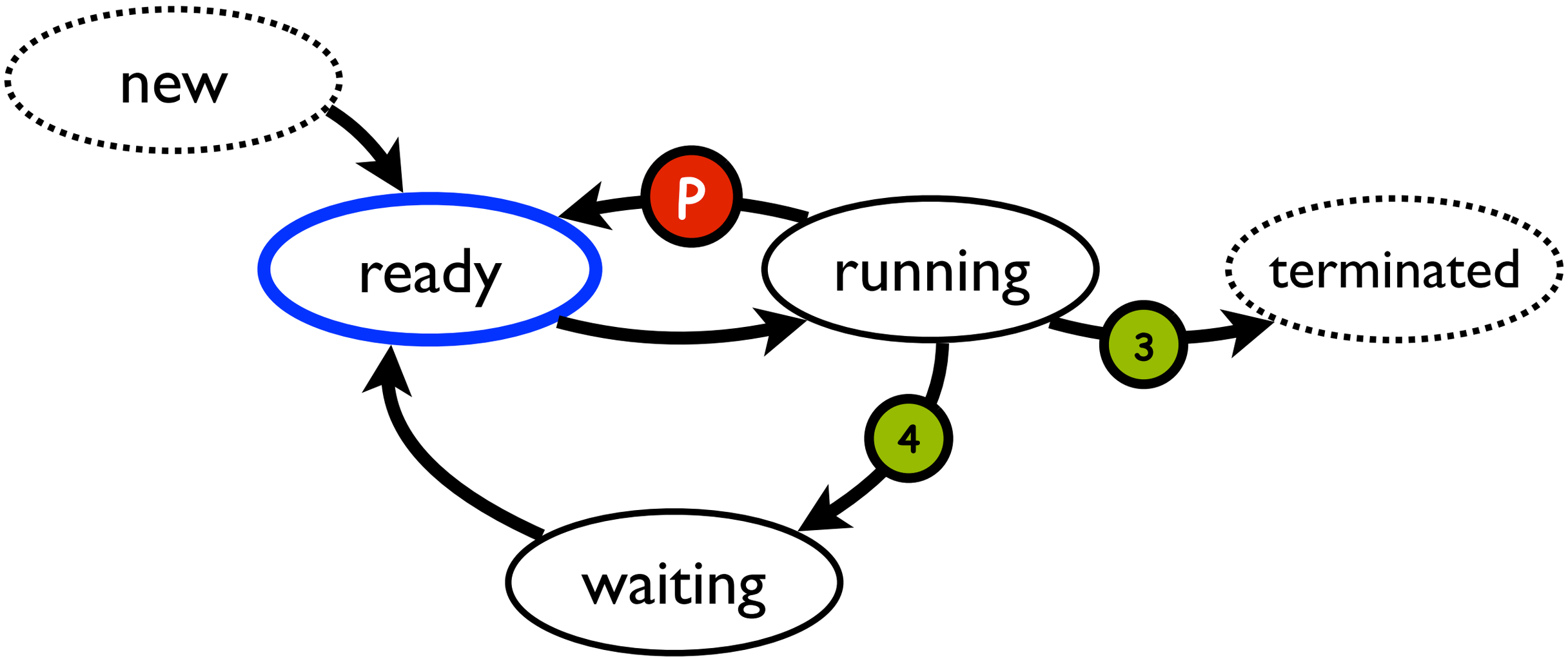
Process arrival

We make no difference between 1 a new process arriving to the ready queue and 5 a process coming back to the ready queue after completion of a blocking system call.



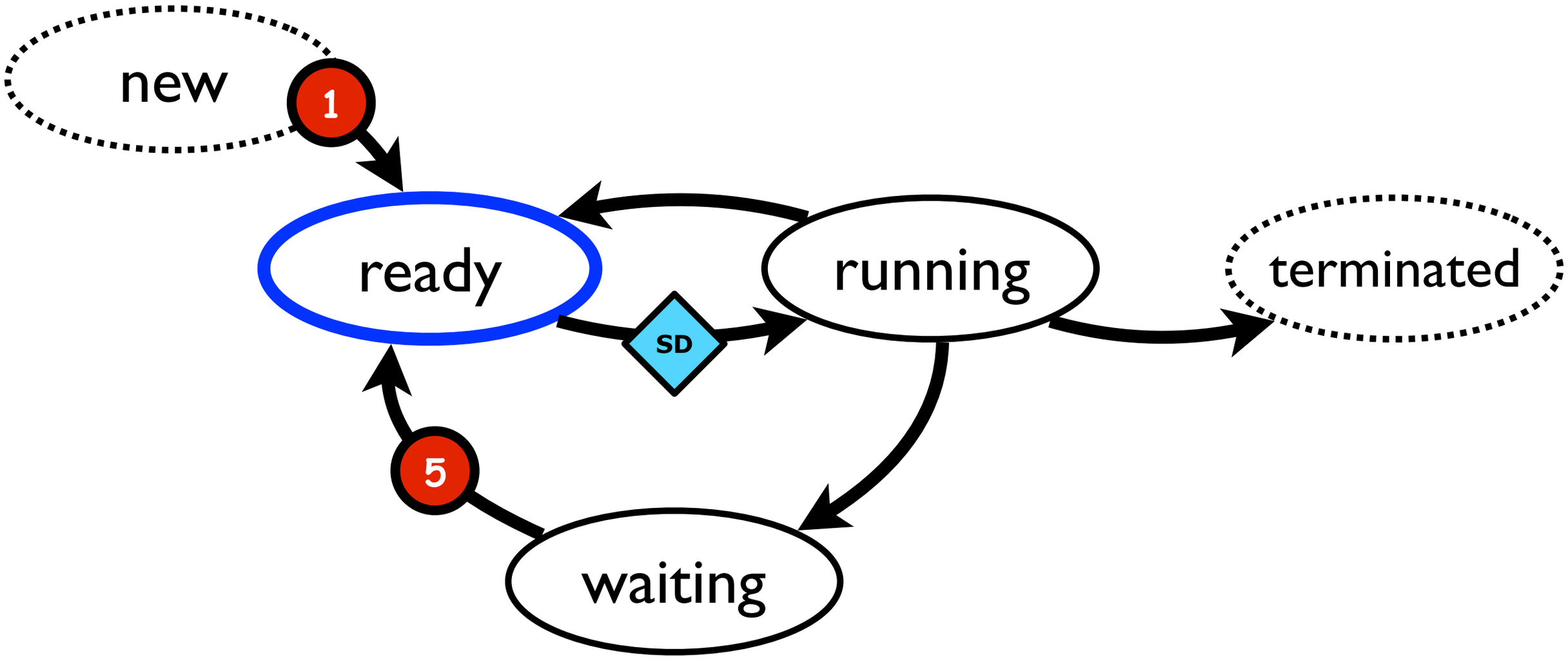
Waiting time

With waiting time we mean the total time a process has been waiting in the ready queue until 3 terminating or 4 performing a blocking system call. A process may get preempted P and forced off the CPU and put back in the ready queue before completing its CPU burst and have more waiting time added.



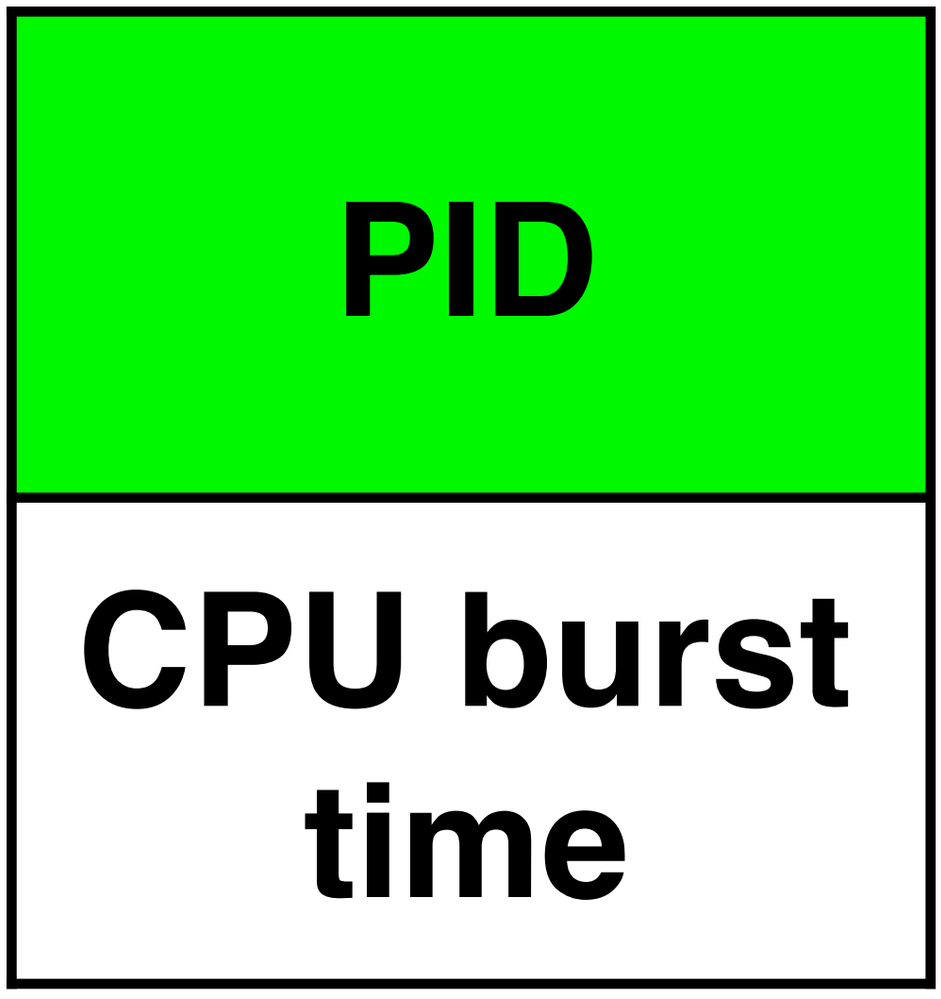
Response time

The model we use to study and compare different CPU scheduling algorithms is abstract and don't take into account what a response is and that it may take time to produce a response once a task gets to execute on the CPU. In this model response time is defined as the time from when a task enters the ready queue (1 or 5) to the time the task first gets to execute on the cpu SD .



Process representation

When studying CPU scheduling a process will be represented by process ID (PID) and the next CPU burst time.

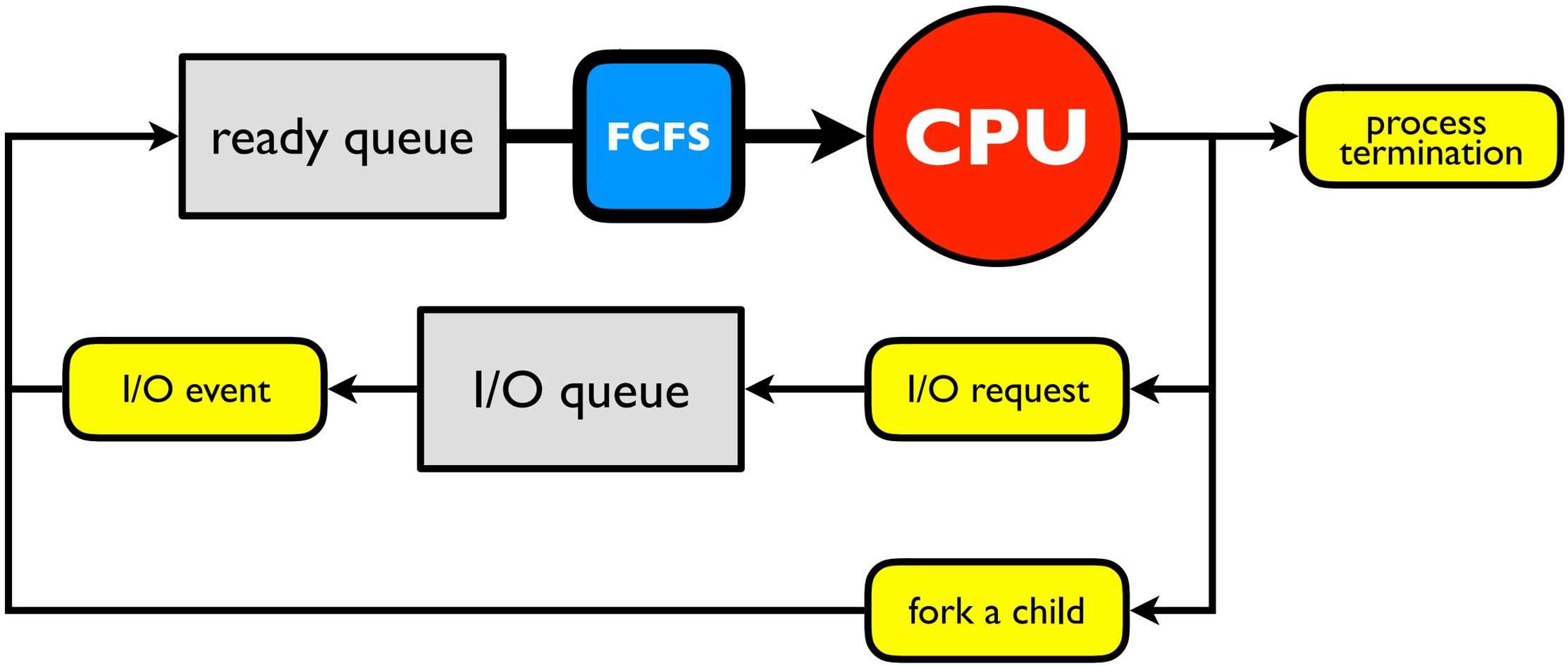


FCFS

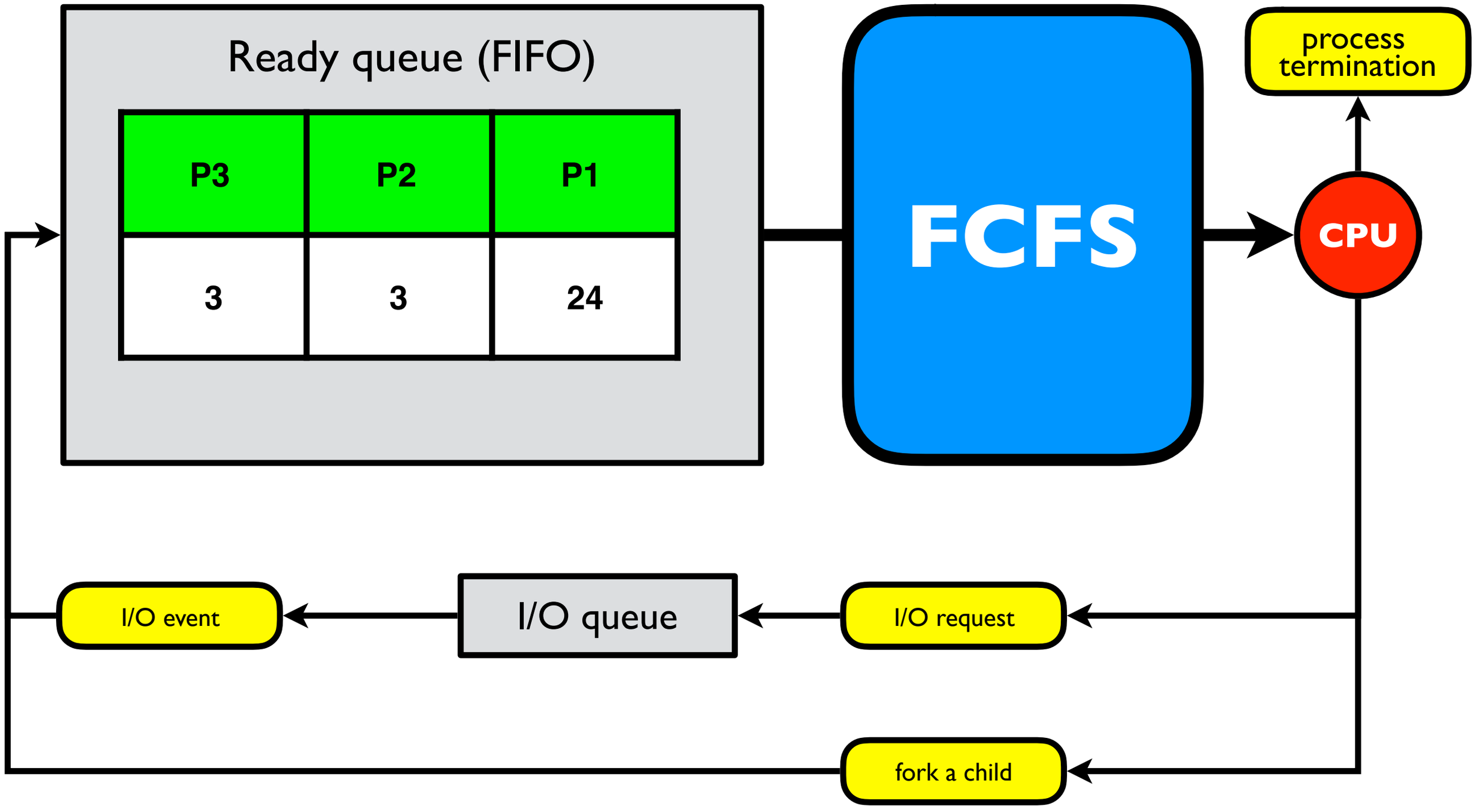
First-Come, First-Served

First-Come, First-Served (FCFS)

The first come, first served (commonly called FIFO - first in, first out) process scheduling algorithm is the simplest process scheduling algorithm. Processes are executed on the CPU in the same order they arrive to the ready queue.



Scheduling algorithms



PID	P3	P2	P1
CPU burst time	3	3	24

The processes arrive to the ready queue in the order: P1, P2, P3.

Gantt Chart for the FCFS schedule

PID	Waiting time
P1	0
P2	24
P3	27

Average waiting time

(0 + 24 + 27)/3 = 17

PID	P1	P3	P2
CPU burst time	24	3	3

What if the same processes arrive to the ready queue in the order: P2, P3, P1.

Gantt Chart for the FCFS schedule

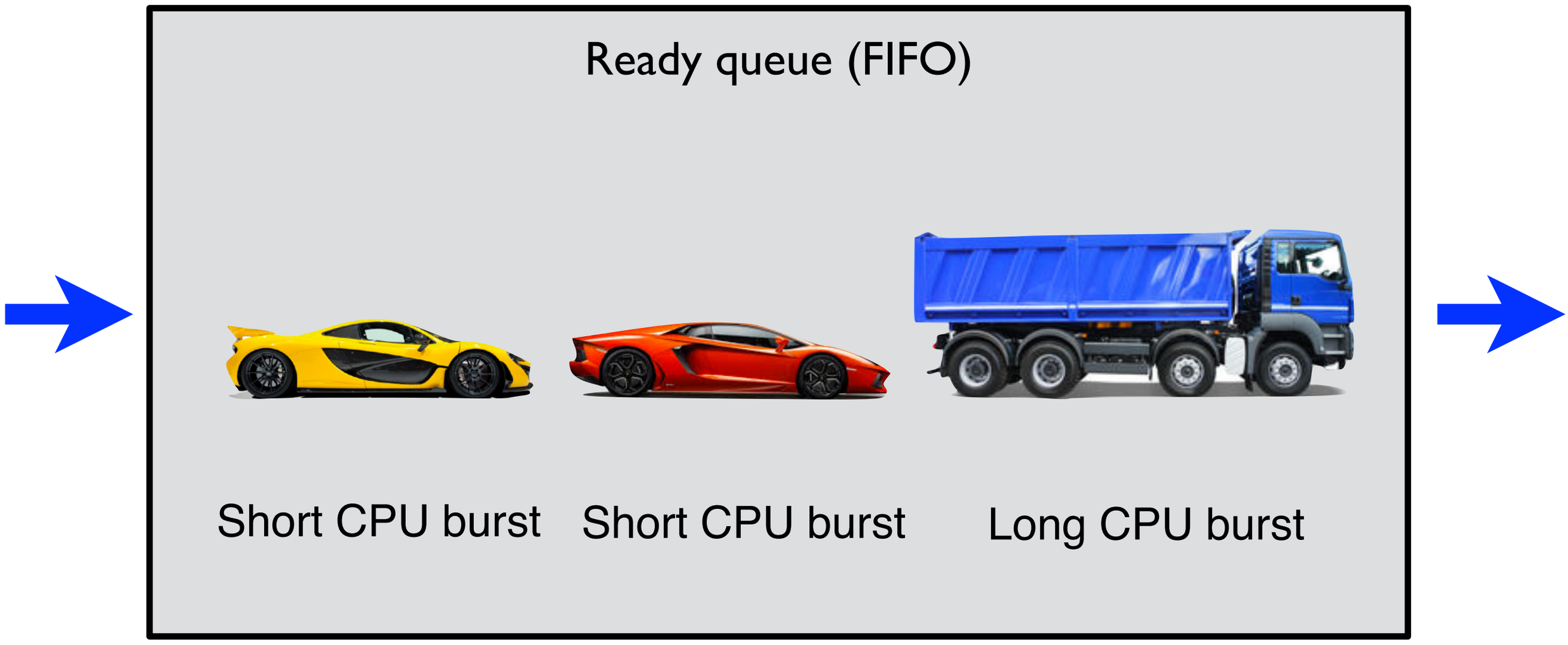
PID	Waiting time
P1	6
P2	0
P3	3

Average waiting time

(6 + 0 + 3)/3 = 3

The convoy effect

When using FCFS scheduling, if I/O bound (short CPU burst) processes are scheduled after CPU bound (long CPU burst) processes, the average waiting time increases.



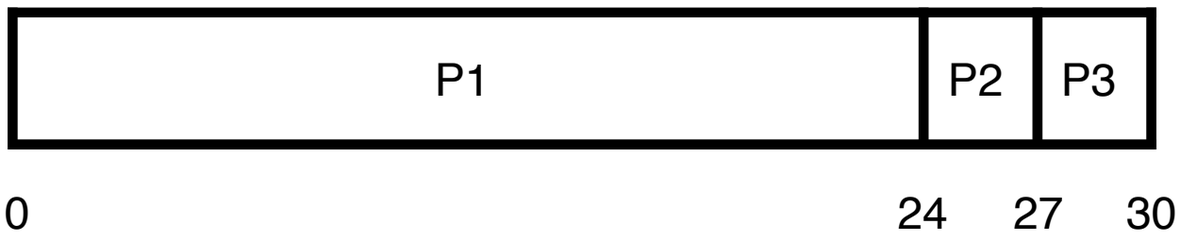
The convoy effect

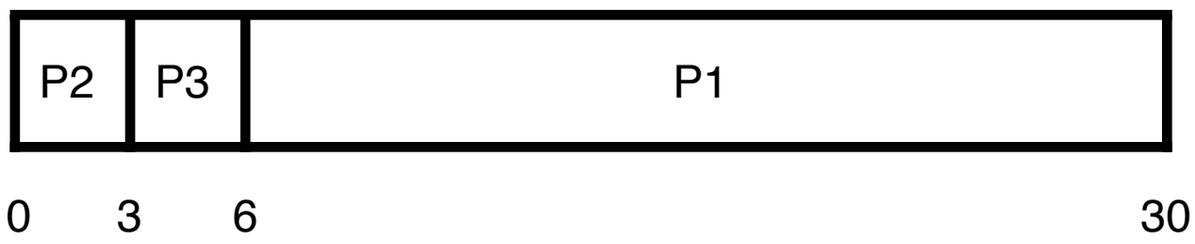
When using FCFS scheduling, if I/O bound (short CPU burst) processes are scheduled after CPU bound (long CPU burst) processes, the average waiting time increases.

PID	P3	P2	P1
CPU burst time	3	3	24

PID	P1	P3	P2
CPU burst time	24	3	3

Gantt Charts for the FCFS schedules





PID	Waiting time
P1	0
P2	24
P3	27

Average waiting time

(0 + 24 + 27)/3 = 17

PID	Waiting time
P1	6
P2	0
P3	3

Average waiting time

(6 + 0 + 3)/3 = 3

First-Come, First-Served (FCFS)

Advantages

- ★ Simple
- ★ Easy
- ★ First come, first served

Disadvantages

- ★ This scheduling method is **nonpreemptive**, that is, a process will execute its CPU burst until it finishes.
- ★ Because of this nonpreemptive scheduling, short processes which are at the back of the queue have to wait for the long process at the front to finish making the average waiting time increase - the **convoy effect**.

What is the optimal schedule?

To answer this question we must first define what we mean with optimal.

A better question

What schedule minimises the average waiting time?

In general, if we have CPU bursts $x_1, \ldots x_n$, calculate the total waiting time T_{wait}.

$$
\begin{aligned}
T_{wait} = {} & 0 + \\
& x_1 + \\
& x_1 + x_2 + \\
& x_1 + x_2 + x_3 + \\
& \quad \ldots \quad + \\
& x_1 + x_2 + x_3 + \ldots + x_{n-1} \\
= {} & (n-1)x_1 + (n-2)x_2 + \ldots + x_{n-1}
\end{aligned}
$$

Now, calculate the average waiting time.

$$\mathbf{Average}(T_{wait}) = [(n-1)x_1 + (n-2)x_2 + \ldots + x_{n-1}]/n$$

The average waiting time is reduced if the x_i's that are multiplied the most times are the smallest ones.

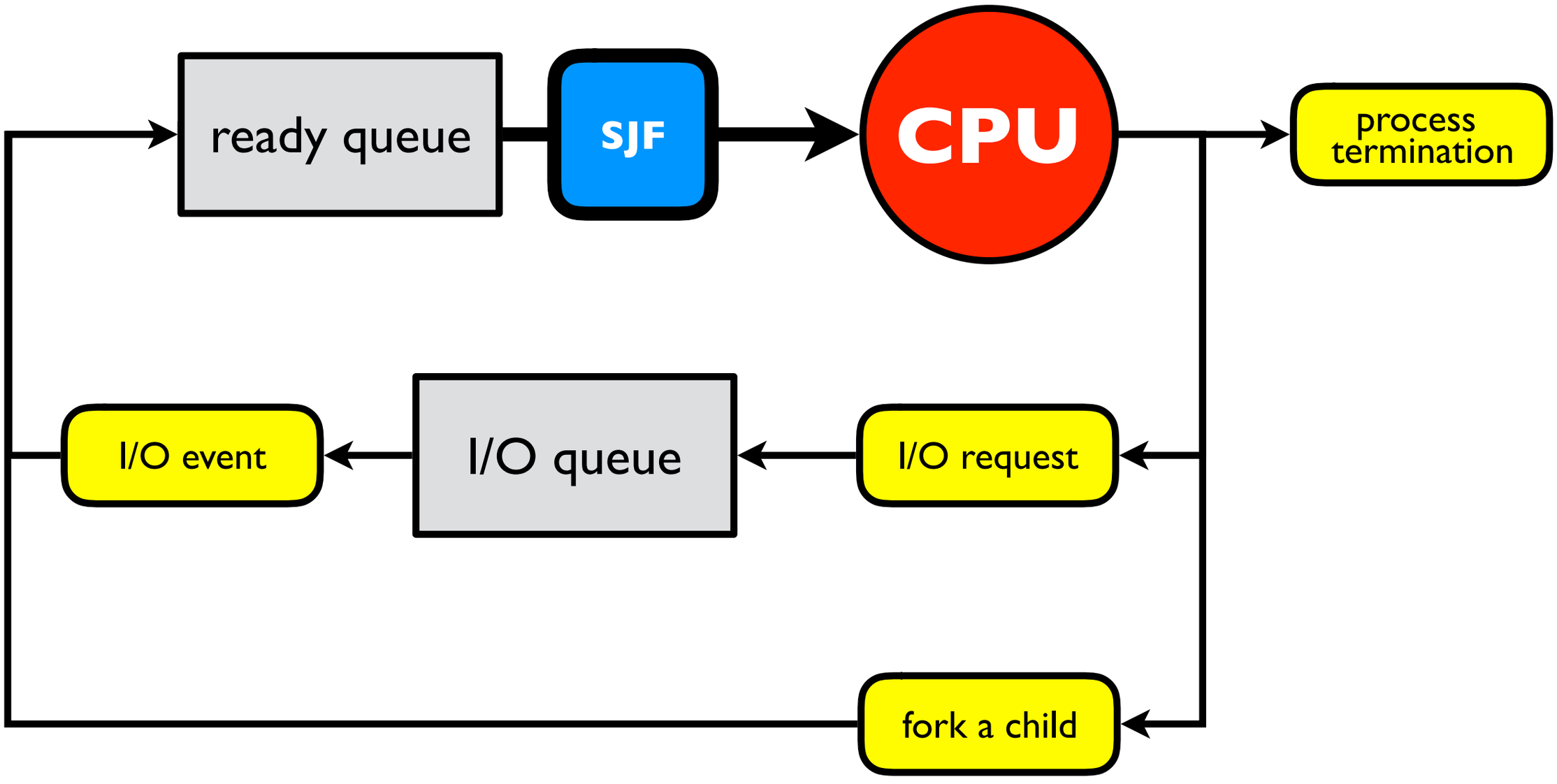
Scheduling the shortest job first gives the minimal average waiting time.

SJF

Shortest Job First

Shortest Job First (SJF)

Shortest Job First (SJF) scheduling assigns the process estimated to complete fastest, i.e, the process with shortest CPU burst, to the CPU as soon as CPU time is available.



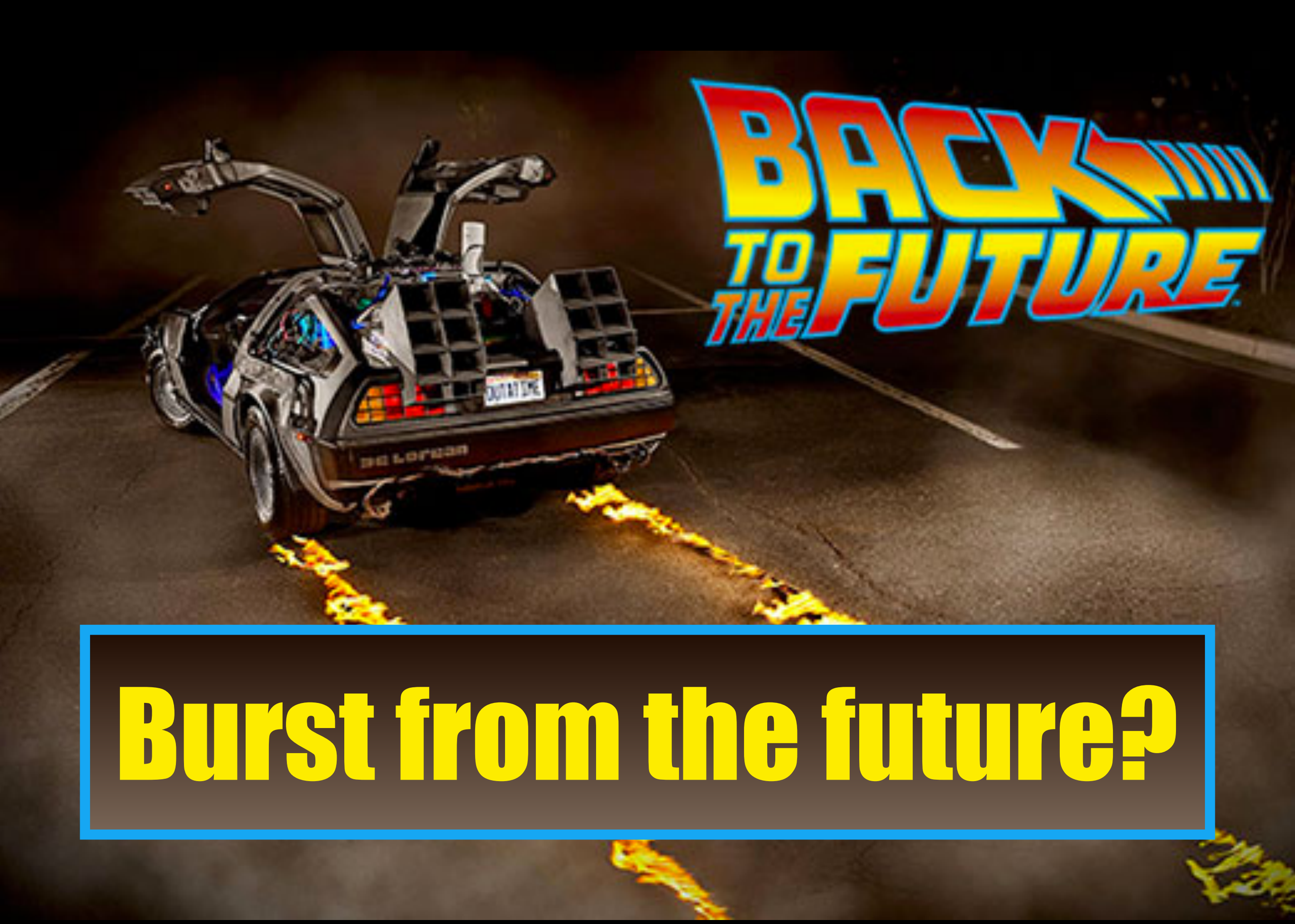
Shortest Job First (SJF)

Shortest Job First (SJF) scheduling assigns the process estimated to complete fastest to the CPU as soon as CPU time is available.

- ★ Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest burst time.
- ★ Also knows as Shortest Process Next (SPN) scheduling.
- ★ Also known as Shortest job next (SJN) scheduling.
- ★ SJF is **optimal** – gives **minimum average waiting** time for a given set of processes.

The difficulty is knowing the length of the next CPU burst.

Source: https://en.wikibooks.org/wiki/Operating_System_Design/Scheduling_Processes/SPN 2016-02-02
https://en.wikipedia.org/wiki/Shortest_job_next 2016-02-02


BACK TO THE FUTURE
Burst from the future?

Can the the length of the next CPU burst be determined dynamically?

Exponential averaging

Can only estimate the length of the next CPU burst. Estimation can be done by using the length of previous CPU bursts, using exponential averaging.

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define : $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

Analysis: what happens when $\alpha \rightarrow 0$ or when $\alpha \rightarrow 1$?

$$\alpha = 0 \Rightarrow \tau_{n+1} = \tau_n$$

Recent history does not affect the estimate.

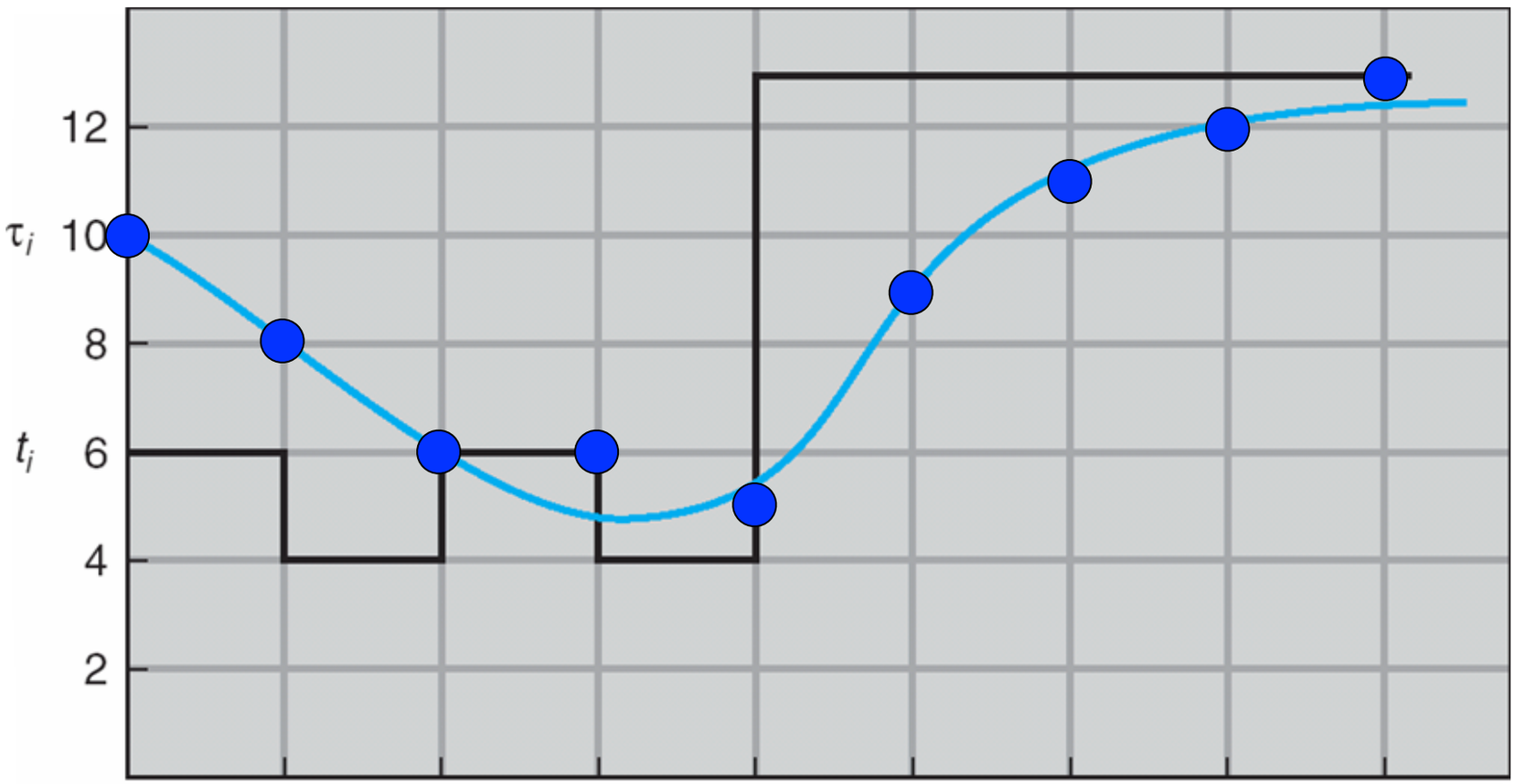
$$\alpha = 1 \Rightarrow \tau_{n+1} = \alpha\, t_n$$

Only the actual last CPU burst affect the estimate.

Exponential averaging example

$\tau_o = 10, \alpha = 0.5$

$\tau_{n+1} = 0.5 \cdot t_n + 0.5 \cdot \tau_n$ = 0.5*(previous burst + previous estimate)

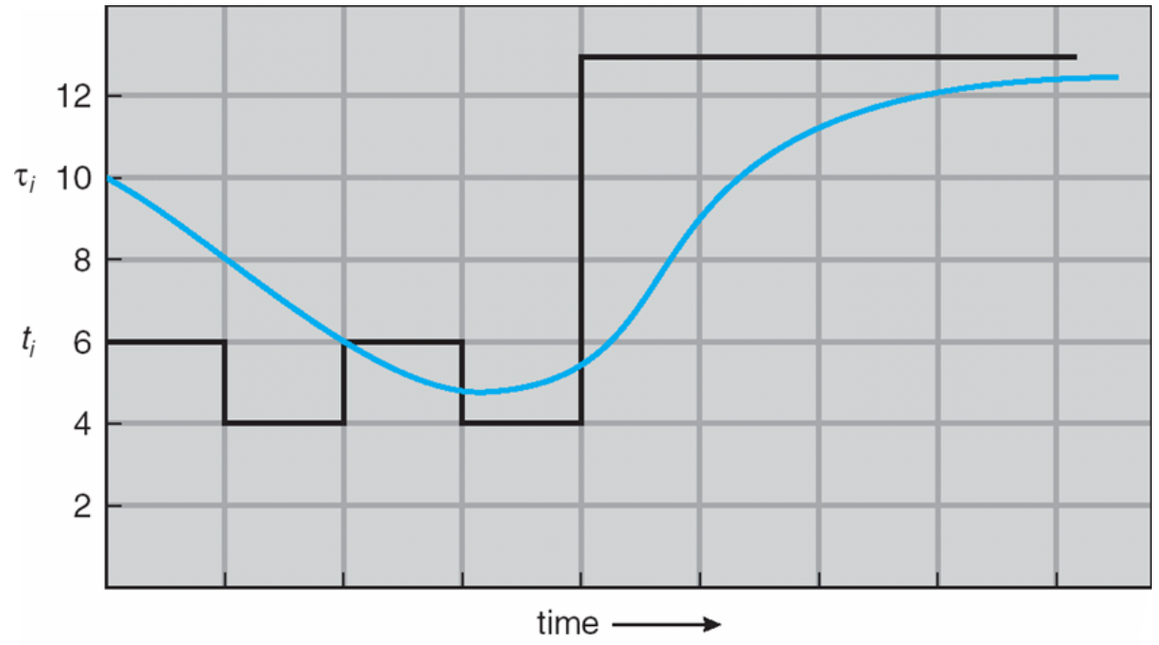


Time	0	1	2	3	4	5	6	7	8
CPU burst (t_i)	6	4	6	4	13	13	13	13	13
"guess" (τ_i)	10	8	6	6	5	9	11	12	13

$\tau_1 = 0.5*(t_0 + \tau_0) = 0.5*(6 + 10) = 8$

Exponential averaging

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$



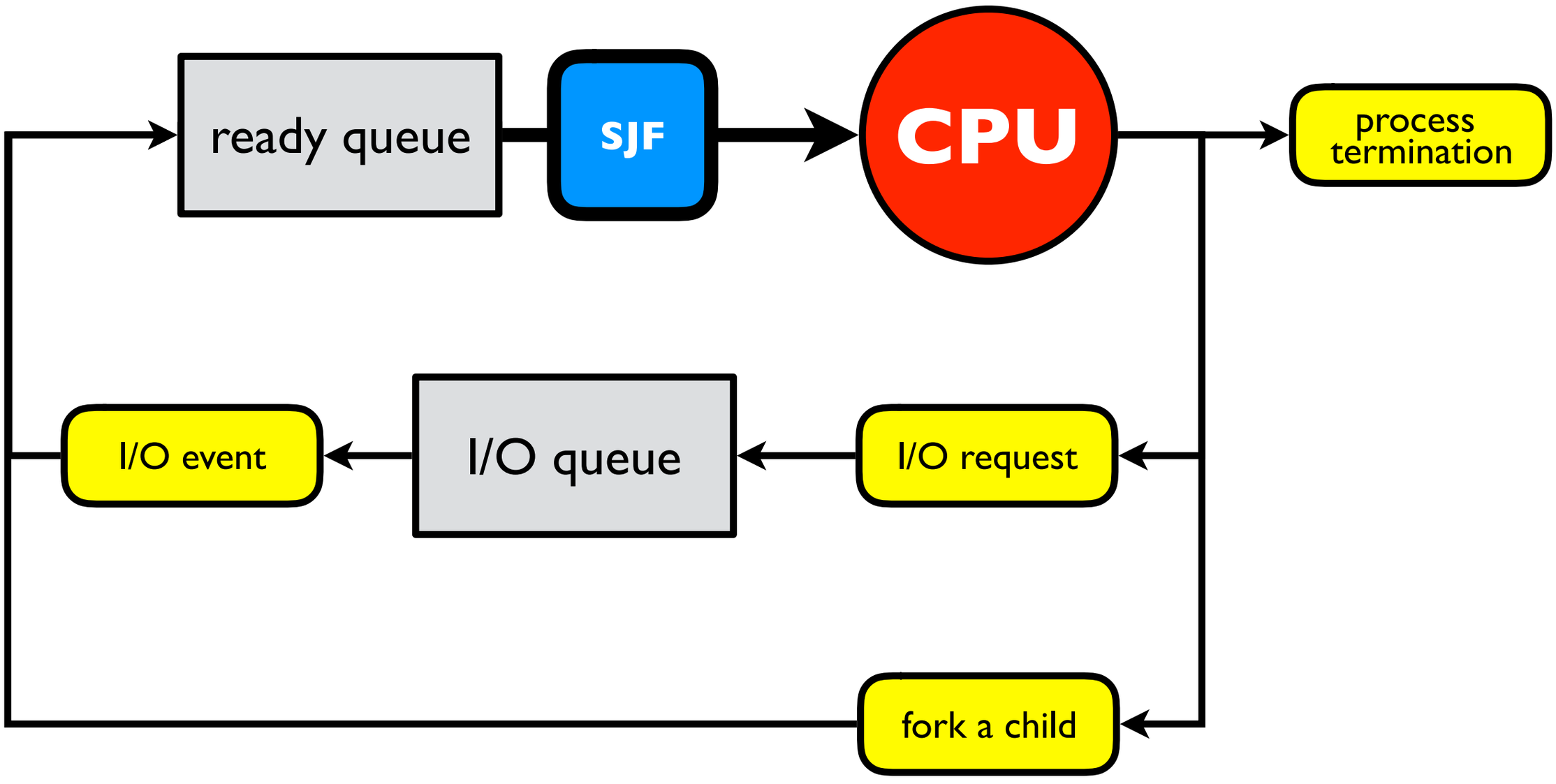
If we expand the formula for τ_{n+1} by substituting for τ_n we get:

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \ldots + (1-\alpha)^j \alpha t_{n-j} + \ldots + (1-\alpha)^{n+1}\tau_0$$

Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

Shortest Job First (SJF)

Can use exponential averaging to estimate the next CPU burst for each process.

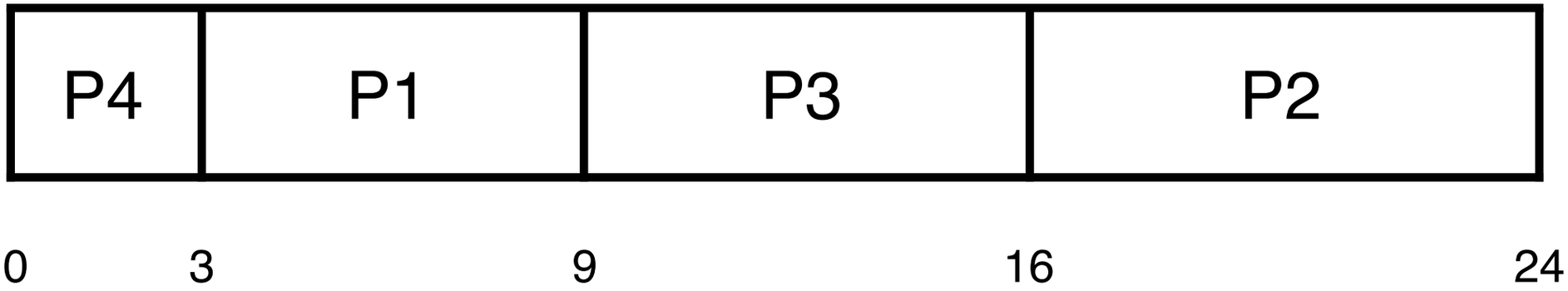


PID	P4	P3	P2	P1
CPU burst time	3	7	8	6

Estimated values of CPU bursts

Processes in the ready queue.

Gantt Chart for the SJF schedule

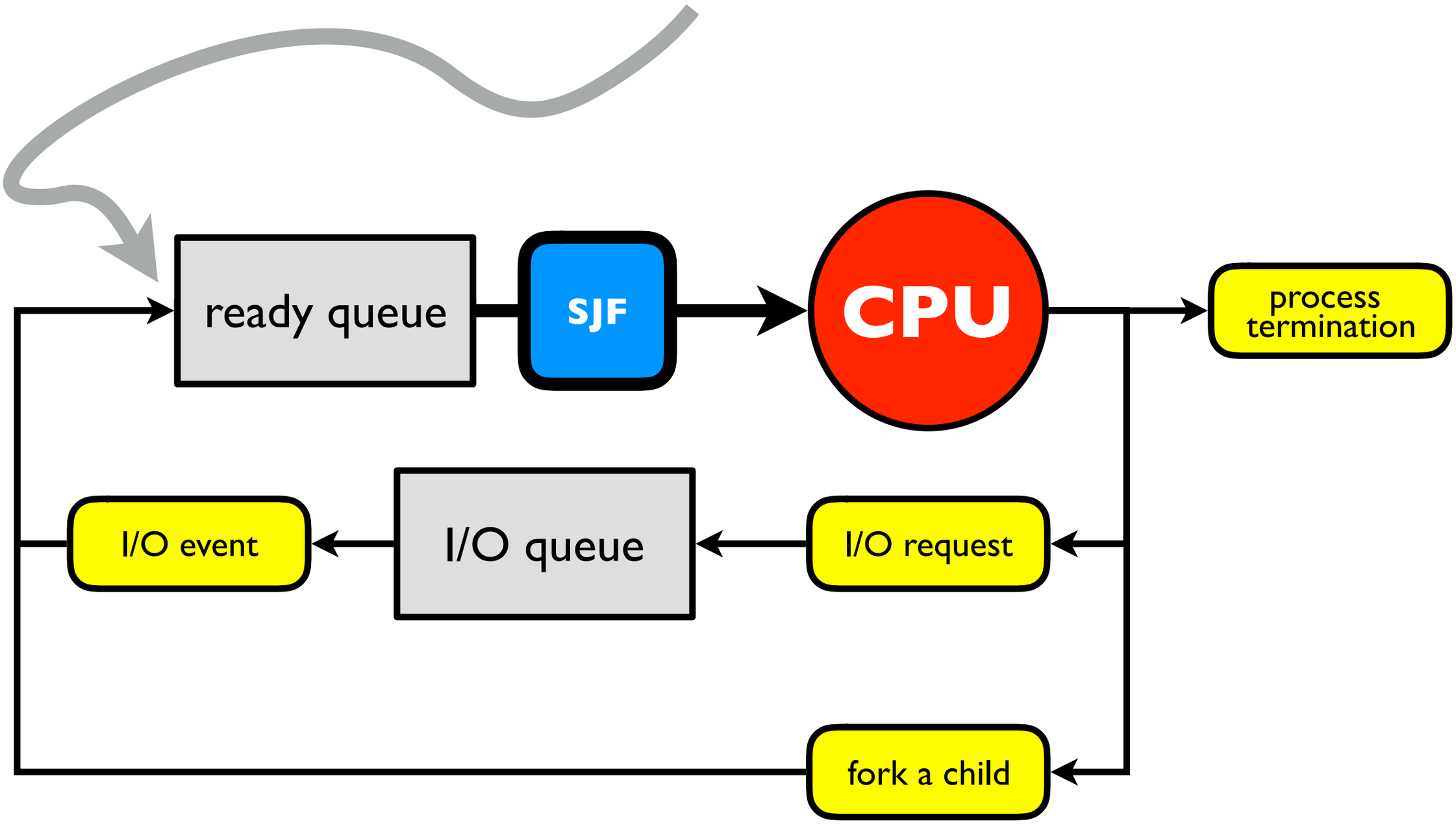


Process	Waiting time
P1	3
P2	16
P3	9
P4	0

Average waiting time

(3 + 16 + 9 + 0)/4 = 28/4 = 7

But does all processes arrive at the same time to the ready queue?



Arrival time

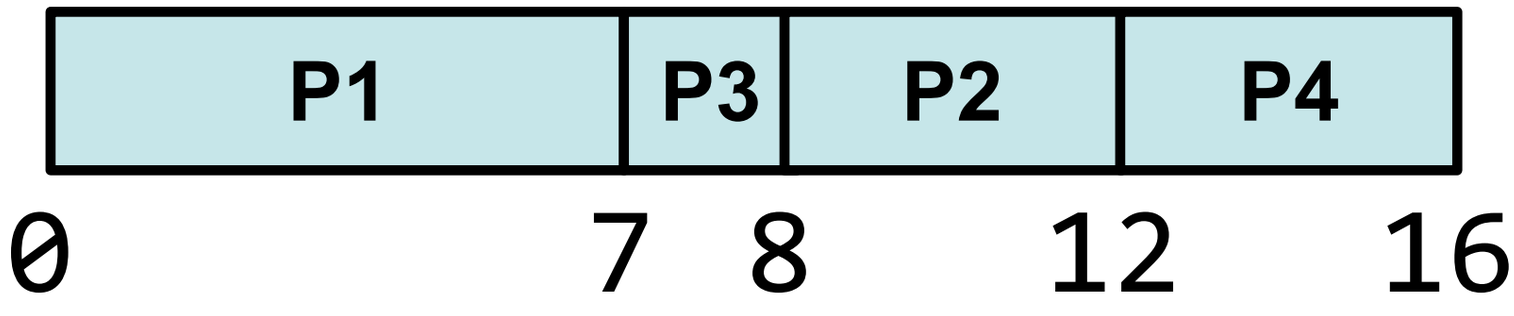
Must keep track of when a process arrives to the ready queue

Example of SJF

Ready queue		
Process	**Arrival time**	**Burst Time**
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Time	Action
0	Only P1 ready to execute
7	When P1 finish, the process in the ready queue with the shortest burst time is selected for dispatch, in this example P3.
8	When P3 finish, both P2 and P4 have burst time 4. Use FIFO to break ties. In this example P2 arrives before P4, hence P2 is selected for execution
12	When P2 finish, only P4 remains in the ready queue.
16	The ready queue is now empty, all processes done.

Gantt Chart for the SJF schedule



Example of SJF - Average waiting time

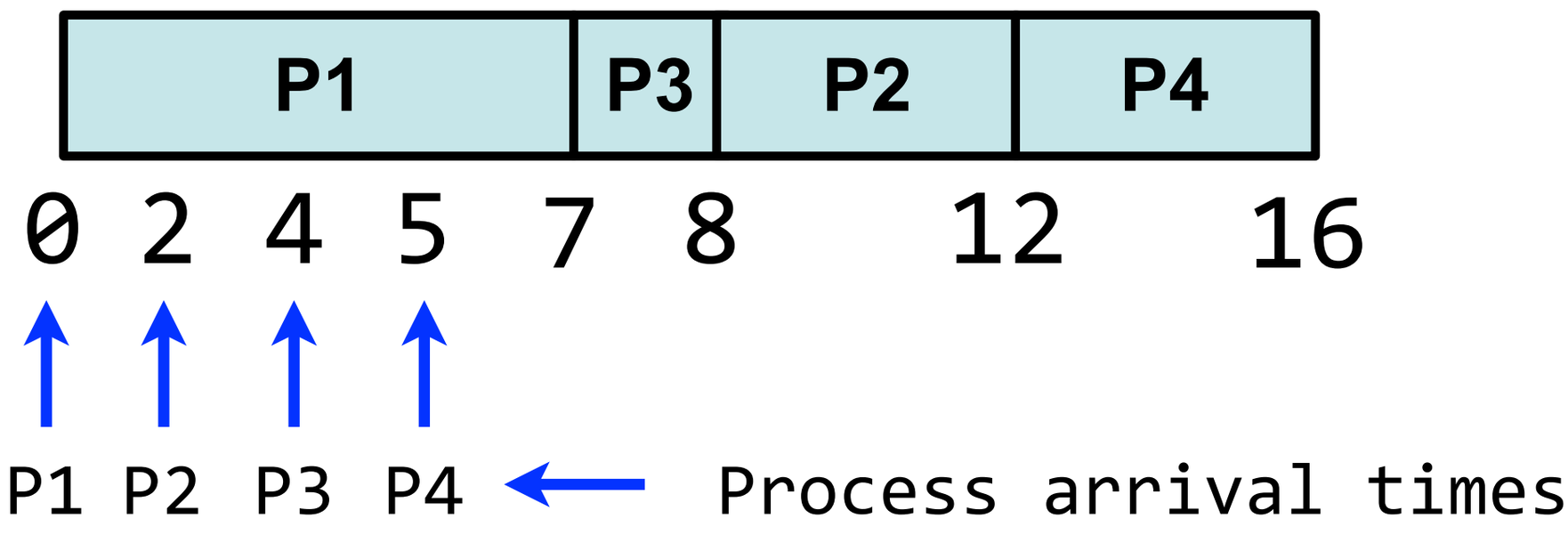
Ready queue			SJF				
Process	**Arrival time**	**Burst Time**	$T_{dispatch}$	-	$T_{arrival}$	=	**Waiting time**
P1	0	7	0	-	0	=	0
P2	2	4	8	-	2	=	6
P3	4	1	7	-	4	=	3
P4	5	4	12	-	5	=	7

Average waiting time

(0 + 6 + 3 + 7) / 4
=

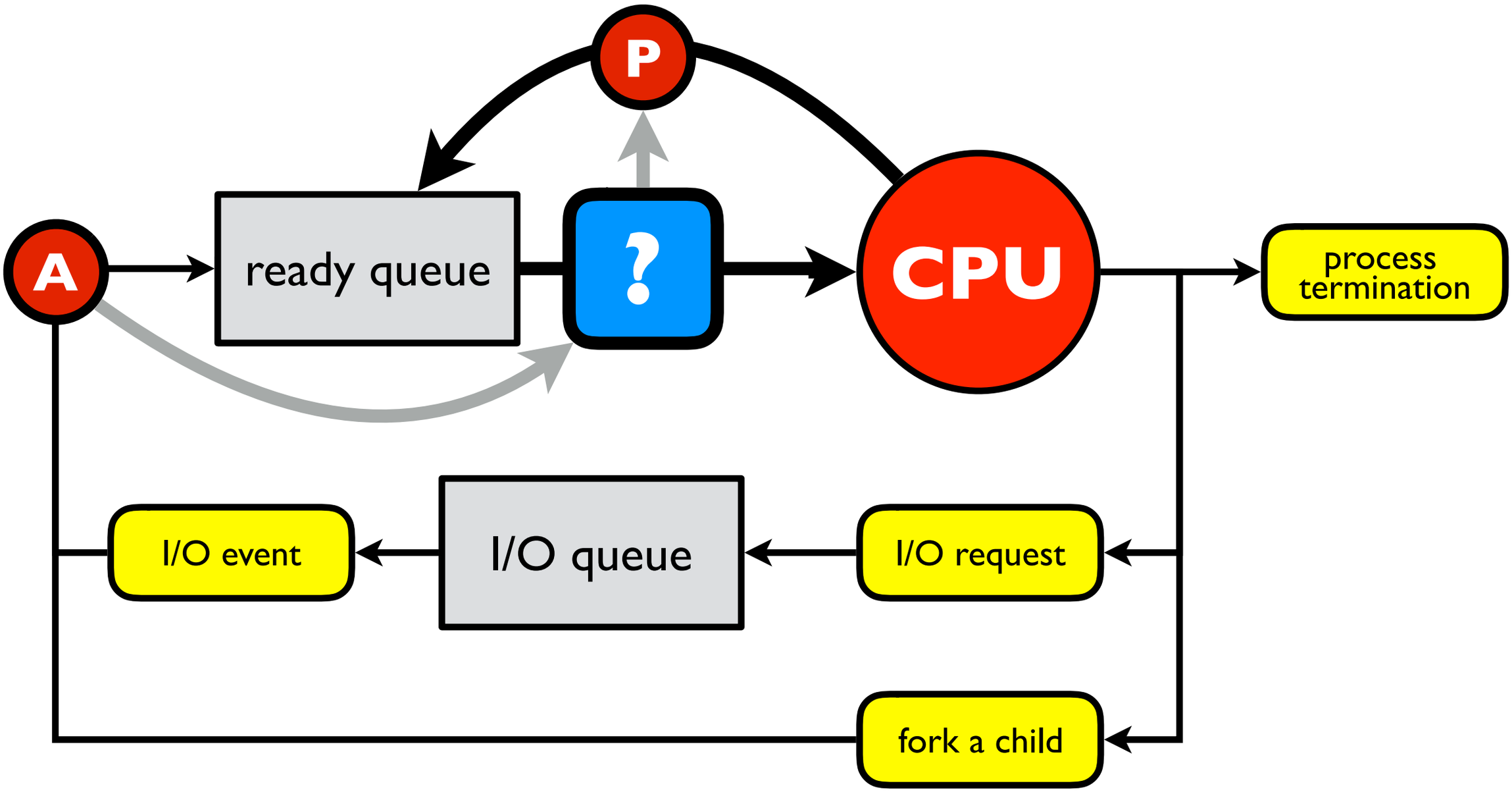
16/4 = 4

Gantt Chart for the SJF schedule



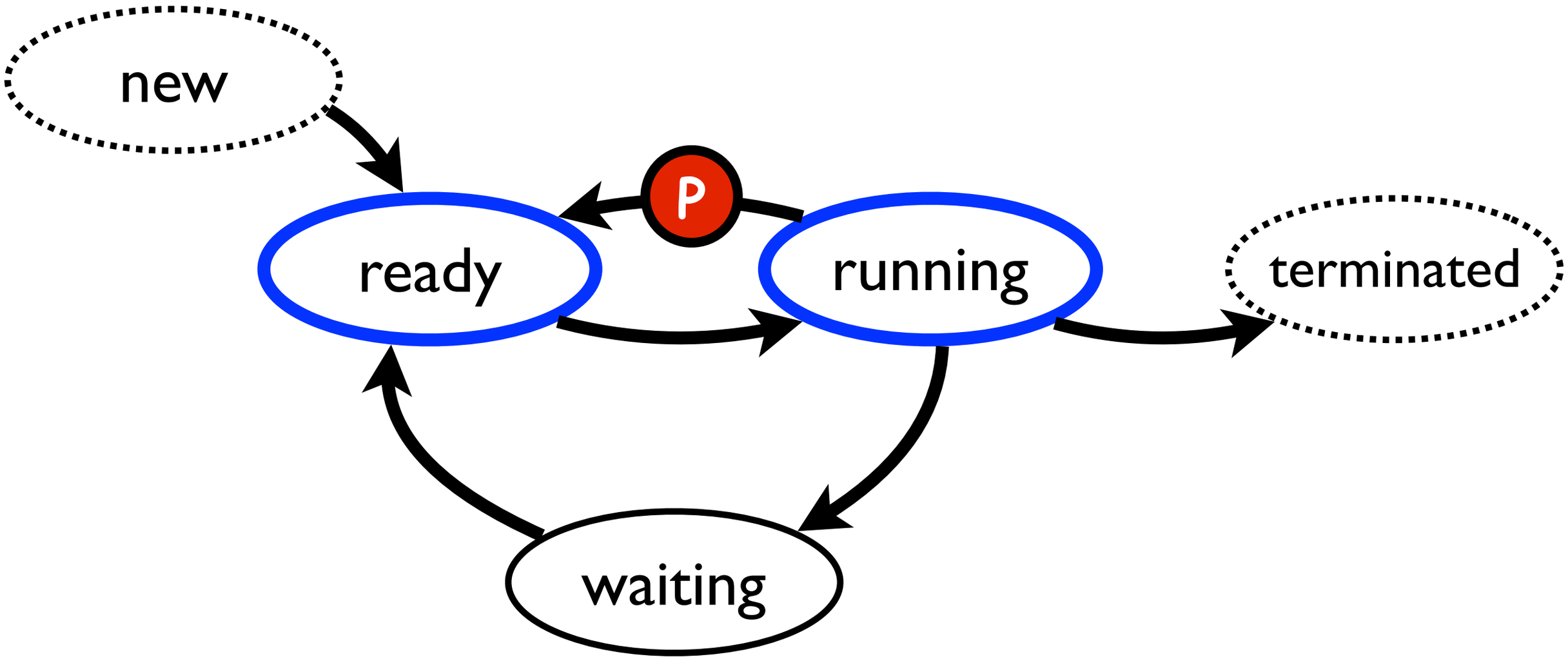
A When a process with a shorter burst time compared to the currently scheduled process arrives to the ready queue ...

P ... wouldn't it be more optimal (minimising the average waiting time) to preempt the running process and switch to newly arrived process?



Preemption

A process may get preempted P, i.e., forced off the CPU and put back in the ready queue before completing its CPU burst.

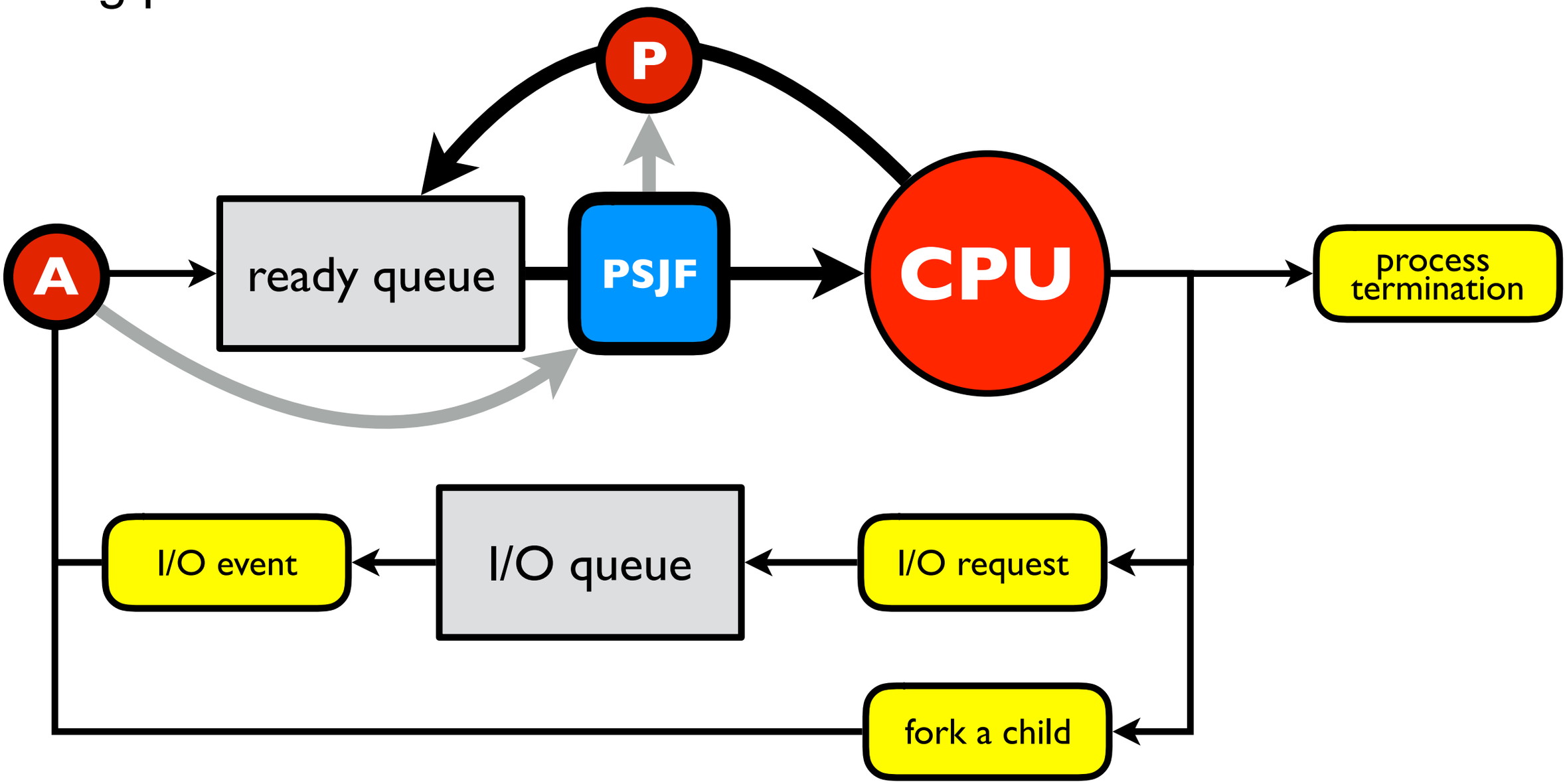


PSJF

Preemptive Shortest Job First

Preemptive Shortest Job First (PSJF)

An extension of SJF where the currently running process is P preempted if the CPU burst of a process A arriving to the ready queue is shorted than the remaining CPU burst of the currently running process.

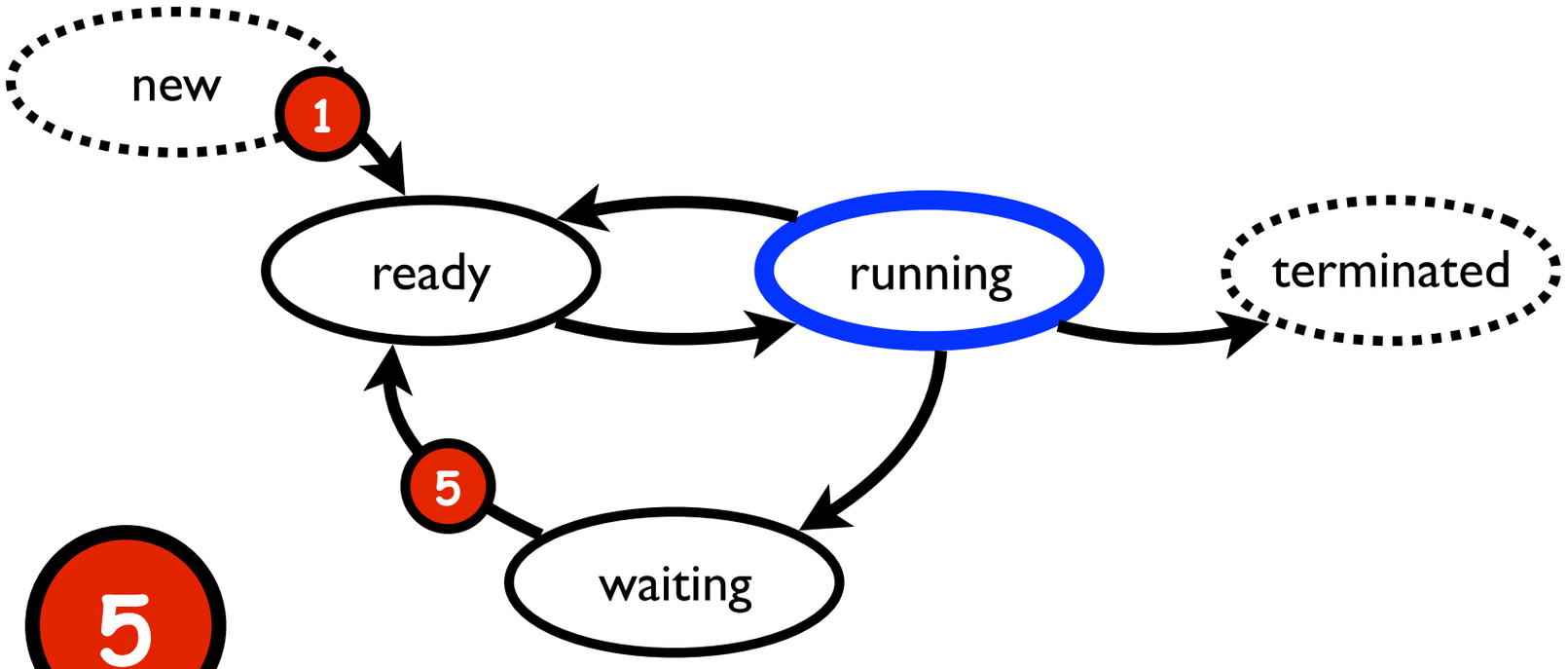


Preemptive Shortest Job First (PSJF)

The currently running process is preempted if the CPU burst of a process arriving to the ready queue is shorted than the remaining CPU burst of the currently running process.

- ★ Also known as shortest remaining time first (SRTF).
- ★ The currently executing process will always run until completion or until a new process is added to the ready queue that requires a smaller amount of time to complete.
- ★ Shortest remaining time is advantageous because **short processes are handled very quickly**.
- ★ Requires very **little overhead** since a decision is made only when a process completes or a new process is added, and when a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently in the ready queue.

SJV vs PSJF



1 & 5

SJF: The currently running process is allowed to continue to execute.

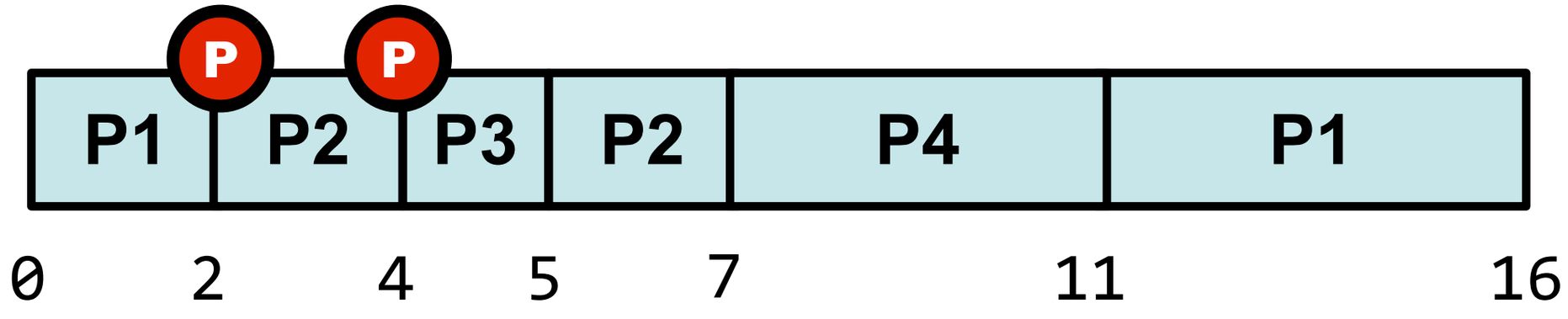
PSJF: The currently running process is preempted if the CPU burst of the newly arrived process is shorter than the remaining CPU burst of the currently running process.

Example of PSJF

Ready queue		
Process	**Arrival time**	**Burst time**
P1	0	7
P2	2	4
P3	4	1
P4	5	4

T	Action
0	P1 is the only process ready to run.
2	When P2 arrives, P1 is preempted since the burst time of P2 (4) is smaller than the remaining burst time of P1 (5).
4	When P3 arrives, P2 is preempted since the burst time of P3 (1) is smaller than the remaining burst time of P1 (5) and P2 (2).
5	P3 is done and P4 (4) arrives to the ready queue where P1 (5) and P2 (2) already waits. P2 has the smallest remaining burst time and is selected to run next.
7	P2 is done. P1 (5) and P4 (4) waits in the ready queue. P4 (4) has the smallest remaining burst time and is selected to run next.
11	P4 is done. Only P1 (5) waits in the ready queue and is selected to run next.
16	The ready queue is now empty, all processes done.

Gantt Chart for the PSJF schedule



Average waiting time

READY QUEUE			PSJF				
Process	Arrival time	Burst time	$T_{dispatch}$	-	$T_{arrival}$	=	Waiting time
P1	0	7	0	-	0	=	0
			11	-	2	=	9
P2	2	4	2	-	2	=	0
			5	-	4	=	1
P3	4	1	4	-	4	=	0
P4	5	4	7	-	5	=	2

Processes may have to wait in the ready queue more than once.

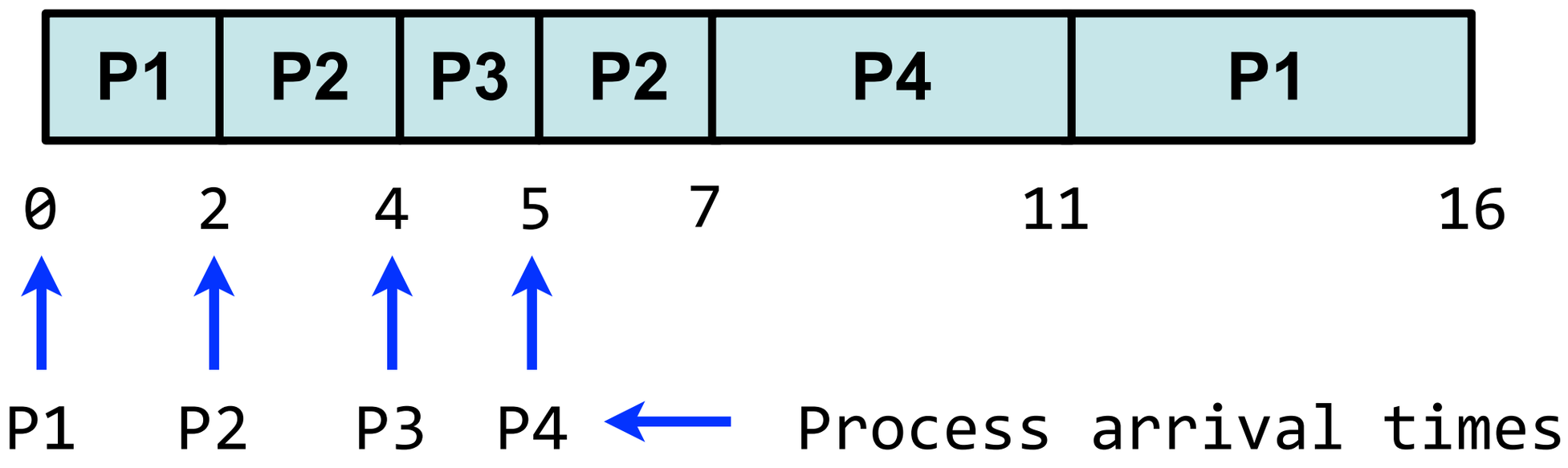
Average waiting time

(9 + 1 + 0 + 2) / 4

=

12/4 = 3

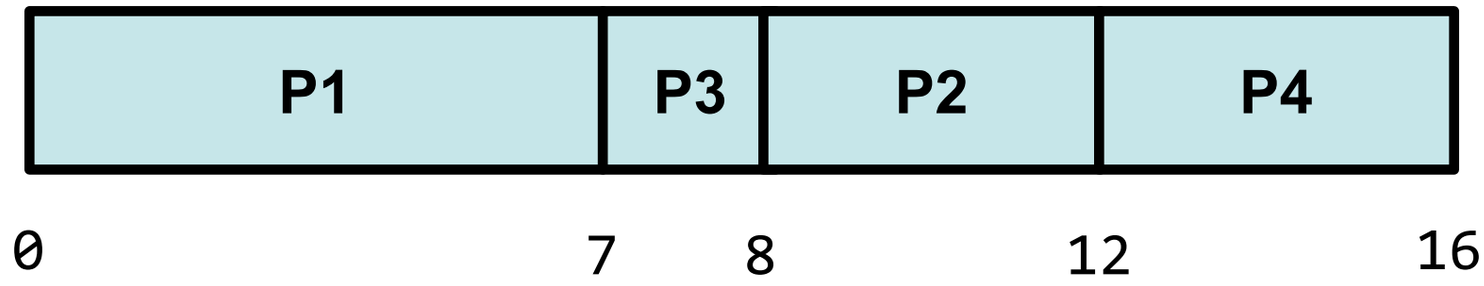
Gantt Chart for the PSJF schedule



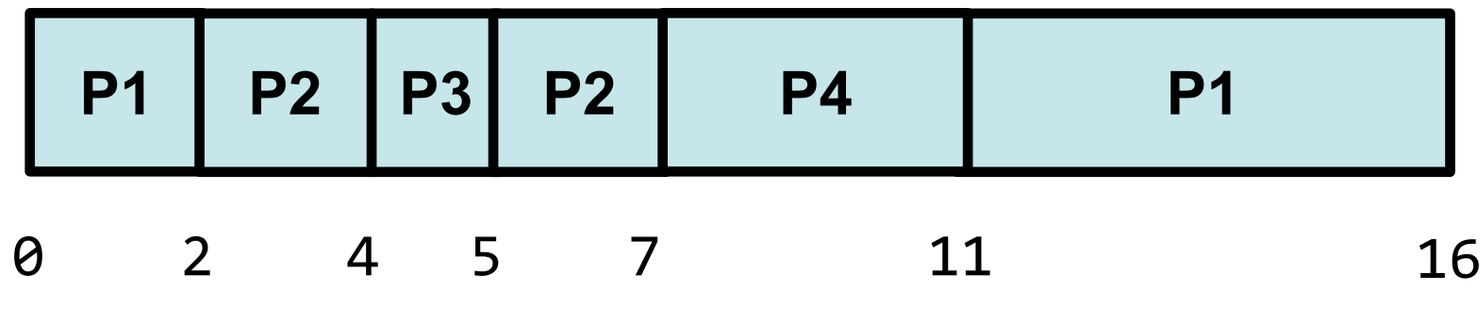
SJF vs PSJF

READY QUEUE		
Process	Arrival time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

SJF, average waiting time = 4



PSJF, average waiting time = 3



SJF gives the optimal average waiting time for a given set of processes currently in the ready queue.

PSJF aims at decreasing the average waiting time by allowing a newly arriving process to preempt the currently running process if the CPU burst of the new process is shorter than what remains of the currently running process.

Priority Scheduling

Priority scheduling

A priority number (integer) is associated with each process. The CPU is allocated to the process with the highest priority (smallest integer = highest priority).

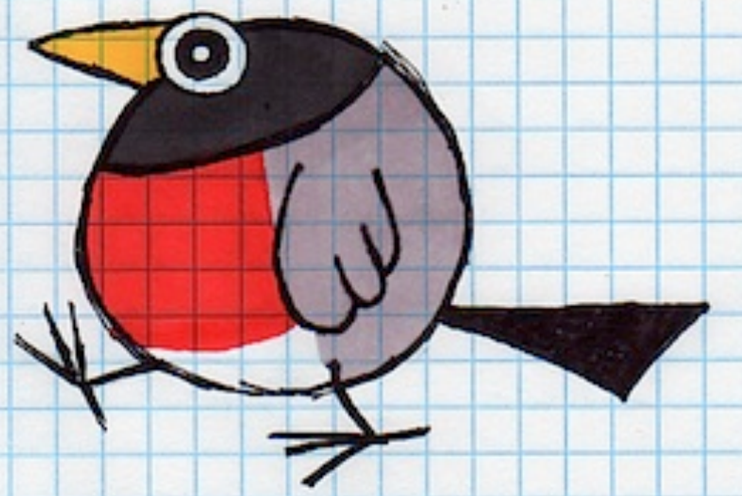
- ★ **Preemption** – should a higher priority process be allowed to preempt a running process with lower priority?
- ★ **Starvation** – low priority processes may never execute.
- ★ **Ageing** – ensure that jobs with lower priority will eventually complete their execution. Ageing can be implemented by increasing the priority of a process as time progresses.
- ★ **SJF** is a priority scheduling algorithm where priority is the predicted next CPU burst time.

RR

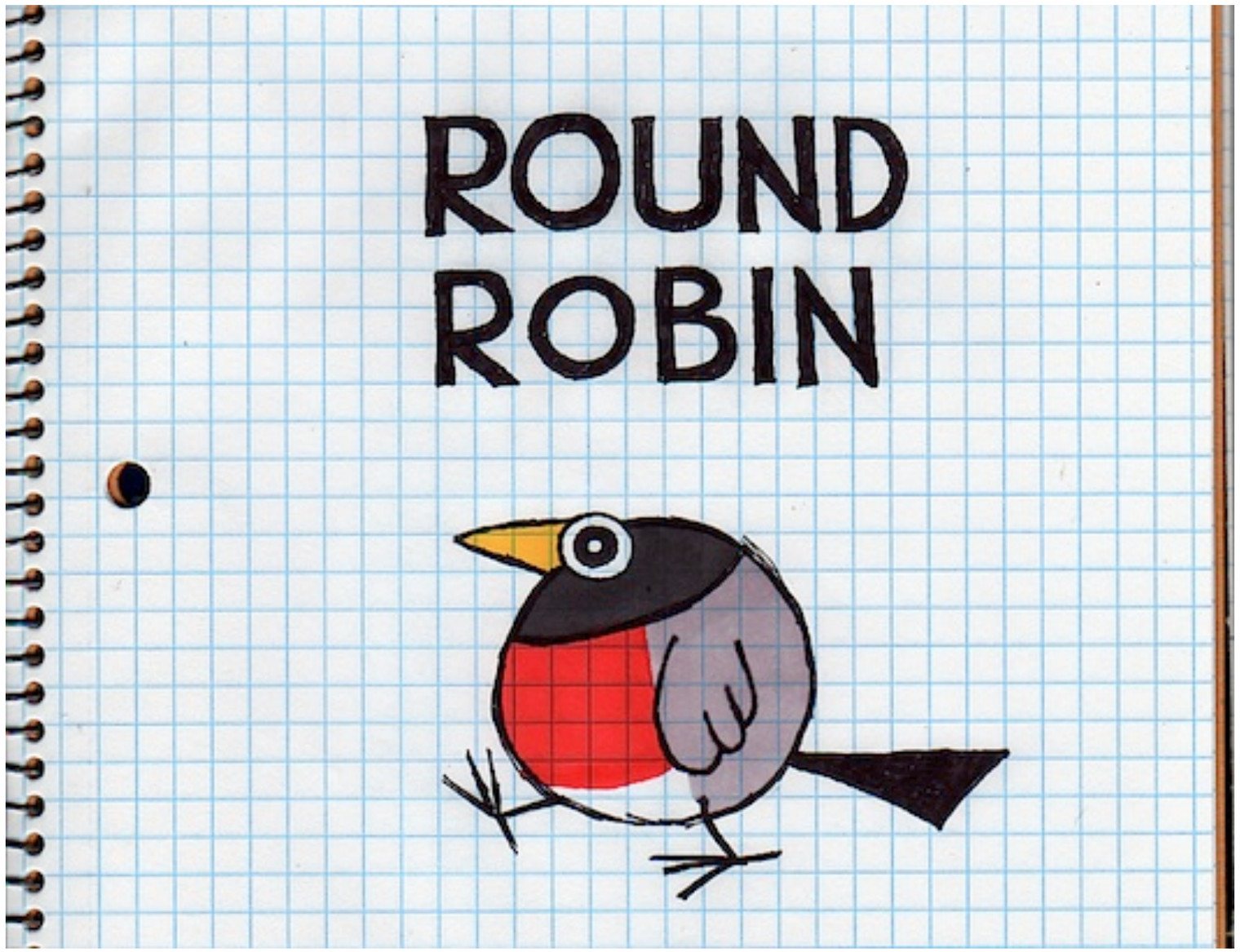
Round Robin

ROUND ROBIN

Robin = *Rödhake* in Swedish

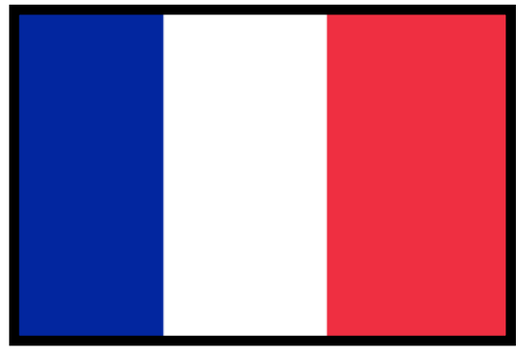


In general, round-robin refers to a pattern or ordering whereby items are encountered or processed sequentially, often beginning again at the start in a circular manner.



Round Robin is one of the simplest **CPU scheduling** algorithms that also **prevents starvation**.

Etymology

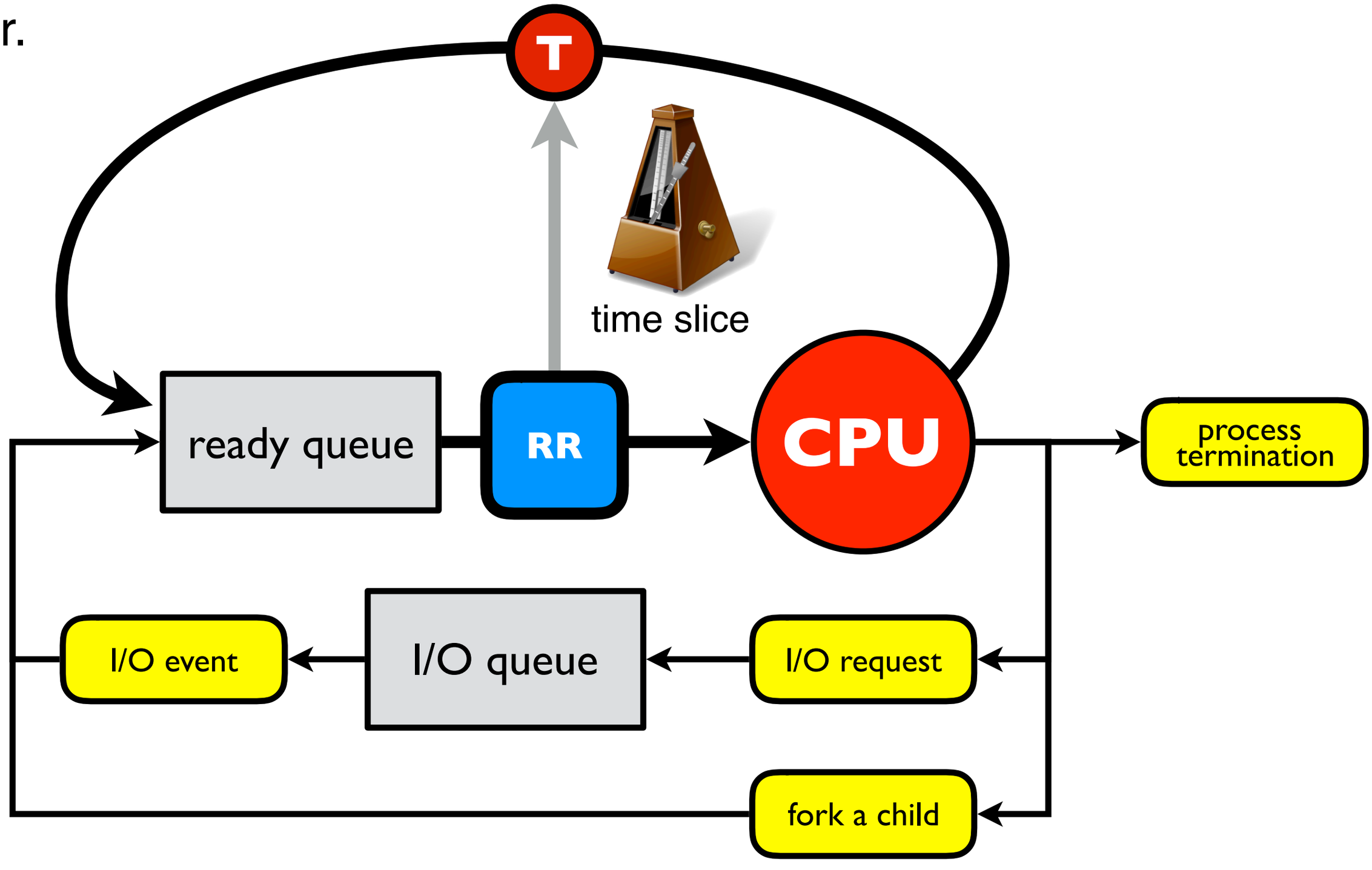


The phrase round-robin actually has nothing whatever to do with a bird, robin or any other kind.

- ★ The term round-robin dates from the 17th-century French **ruban rond** (round ribbon).
- ★ Originally, round-robin is a **document signed by multiple parties in a circle**..
- ★ Round-robin described the practice of signatories to petitions against authority (usually Government officials petitioning the Crown) appending their names on a document in a non-hierarchical circle or ribbon pattern (and so disguising the order in which they have signed) so that none may be identified as a ringleader.

Round Robin (RR)

Round Robin (RR) is a scheduling algorithm where time slices are assigned to each process in equal portions and in circular order.



Characteristics of CPU scheduling algorithms

Metric	Description
Performance	A context dependent metric. What do we mean by performance?
Waiting time	Amount of time a process has been waiting in the ready queue.
Turn around time	Amount of time to execute a particular process.
Response time	Amount of time it takes from when a request was submitted until the first response is produced.

Round Robin (RR)

Each process gets a small unit of CPU time (time quantum), usually 10-100 milliseconds. After this time has elapsed, the process is **preempted** and added to the end of the ready queue.

If there are **n** processes in the ready queue and the time quantum is **q**, then each process gets **1/n** of the CPU time in chunks of at most **q** time units at once.

Response time and extreme behaviours (RR)

A nice property of RR is that there is an upper bound the response time.

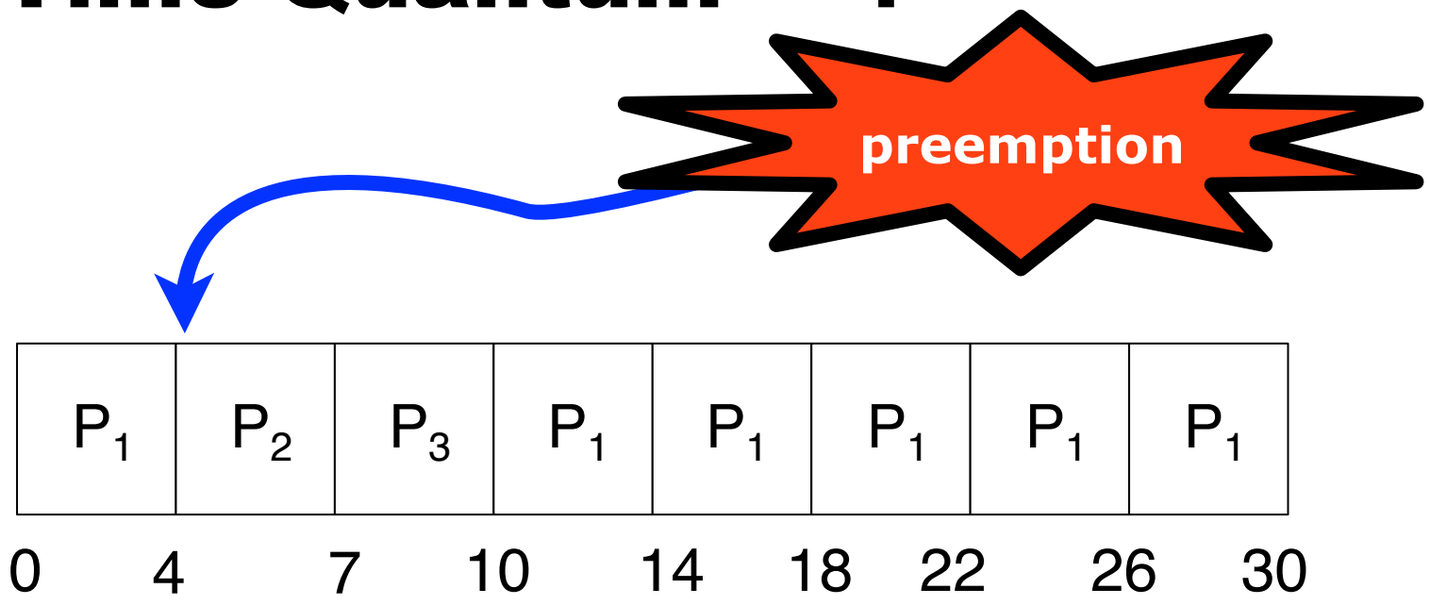
Upper bound for response time: (n - 1)q time units.

Extreme behaviours

- ★ *q* large ⇒ FCFS/FIFO
- ★ *q* small ⇒ **q** must be large with respect to context switch time, otherwise the overhead is too high.

Example of RR with Time Quantum = 4

READY QUEUE	
Process	**Burst Time**
P1	24
P2	3
P3	3



Turnaround time

Amount of time to execute a particular process.

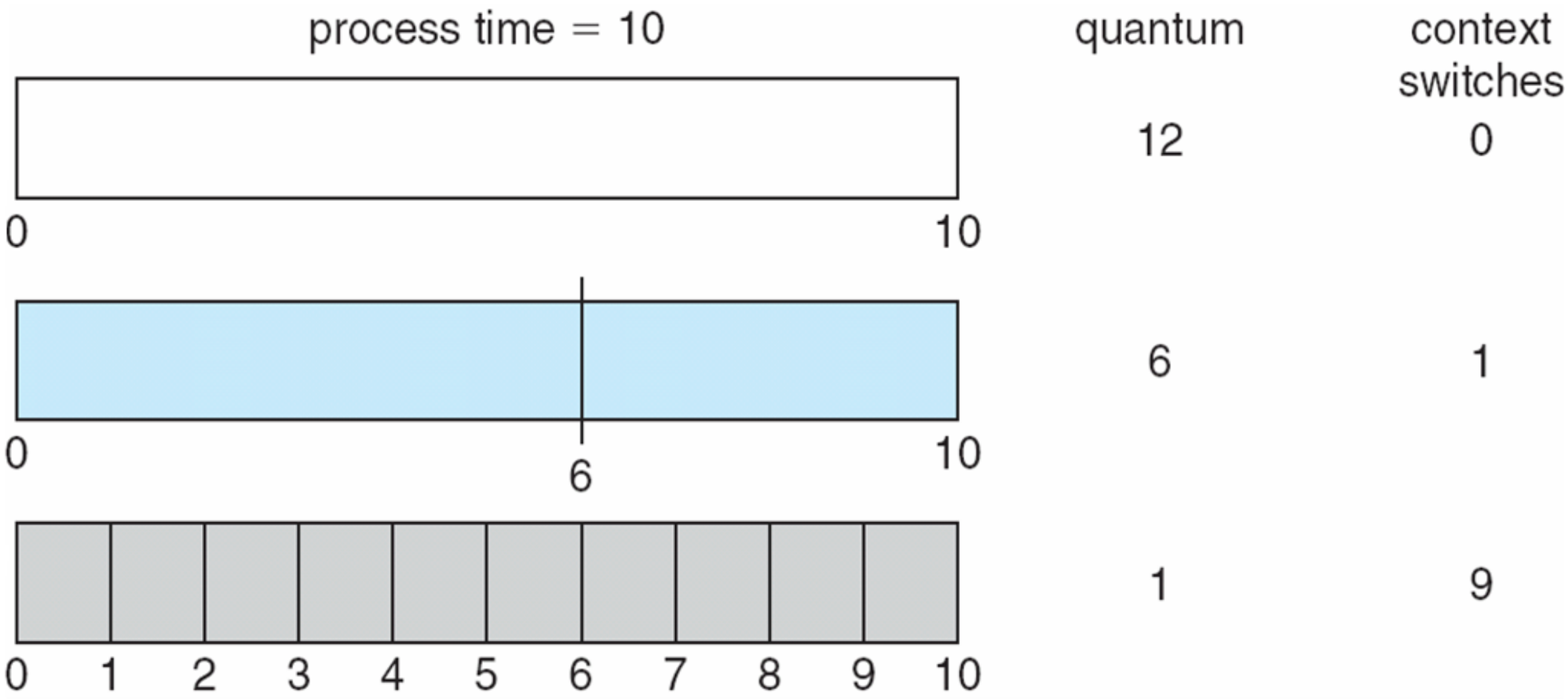
Response time

Amount of time it takes from when a request was submitted until the first response is produced.

Round Robin typically have **higher** average **turnaround** times than **SJF**, but **better** average **response time**.

Time Quantum and Context Switch Time

The RR time quantum (q) affects the number of context switches for a process.



Turnaround time varies with the time quantum

Turnaround time = amount of time to execute a particular process. When using Round Robin scheduling, the average turnaround time will depend on the time quantum (q).

process	time
P_1	6
P_2	3
P_3	1
P_4	7

q = 3

0	P1	3	P2	6	P3	7	P4	10	P1	13	P4	16	P4	17

Average Turnaround Time = (13 + 6 + 7 + 17)/4 = 10.75

q = 5

0	P1	5	P2	8	P3	9	P4	14	P1	15	P4	17

Average Turnaround Time = (15 + 8 + 9 + 17)/4 = 12.25

q = 7

0	P1	6	P2	9	P3	10	P4	17

Average Turnaround Time = (6 + 9 + 10 + 17)/4 = 10.5

Foreground and background processes

Foreground and background processes

Sometimes process can easily be classified into two groups, one set of processes that interacts with users and one set that doesn’t.

Background process (batch)

A process that don’t interacts with any user is called a background process or a batch process.

Foreground process (interactive)

A process that interacts with users is callad a foreground process or an interactive process.

Multilevel queue scheduling

Multilevel queue scheduling (1)

A multi-level queue scheduling algorithm is used in scenarios where the processes can be classified into groups based on properties like process type, CPU time, IO access, memory size, etc

General classification of processes:

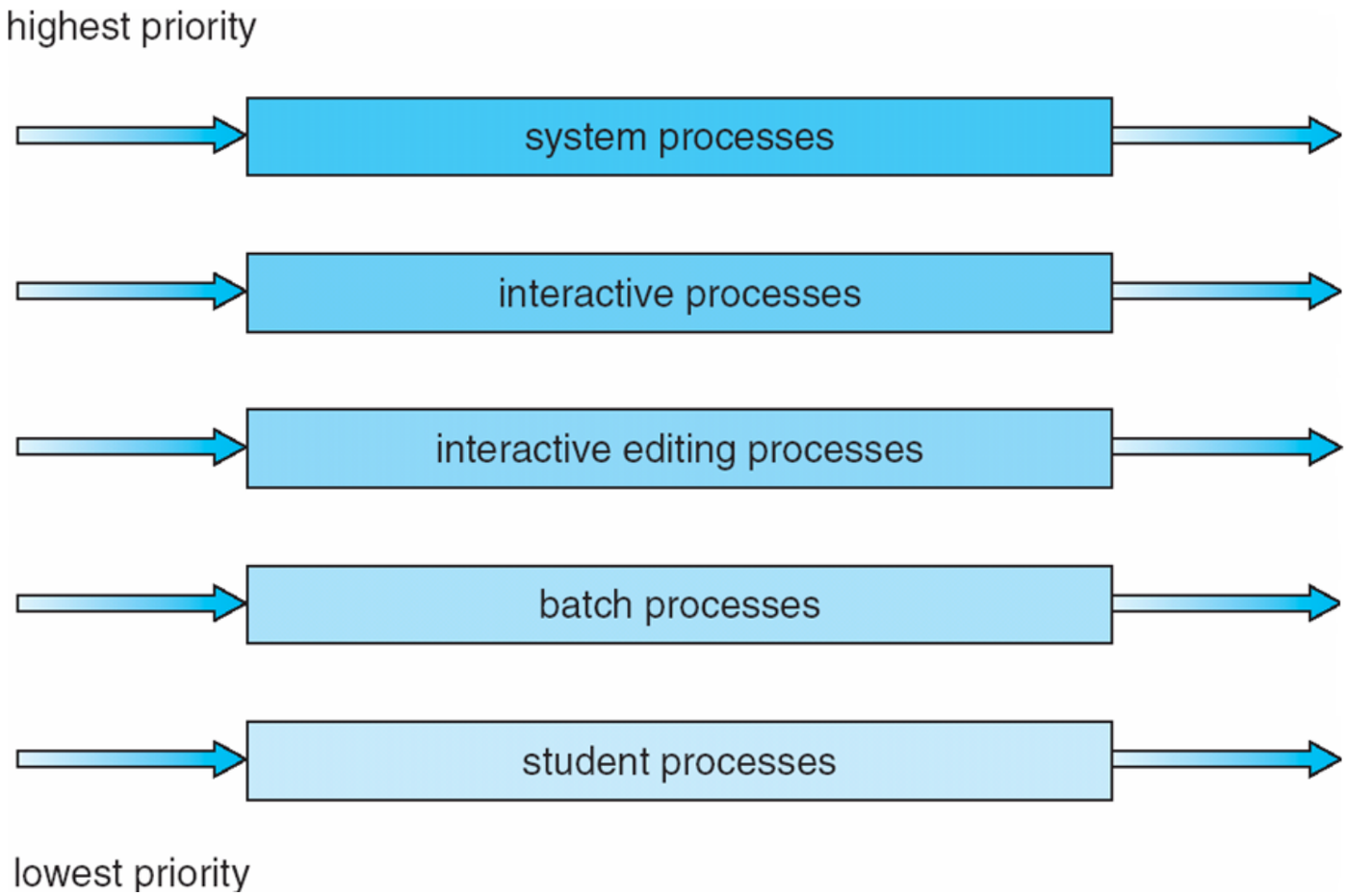
- ★ foreground (interactive)
- ★ background (batch)

In a multi-level queue scheduling algorithm, there will be **n** number of **queues**, where **n** is the number of groups the processes are classified into.

- ★ Each queue will be assigned a priority and will have its own scheduling algorithm like Round-robin scheduling or FCFS.

Multilevel queue scheduling (2)

Use several ready queues. A process is permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.



Each queue has its own scheduling algorithm.

There must be scheduling among the queues.

For example, the foreground queue may have absolute priority over the background queue. If an interactive process enters the ready queue while a batch process is running, the batch process will be preempted.

Multilevel queue scheduling (3)

How to select scheduling algorithms for the various queues?

Ready queue is partitioned into separate queues:

- ★ foreground (interactive)
- ★ background (batch)

Each queue has its own scheduling algorithm

- ★ foreground – RR
- ★ background – FCFS

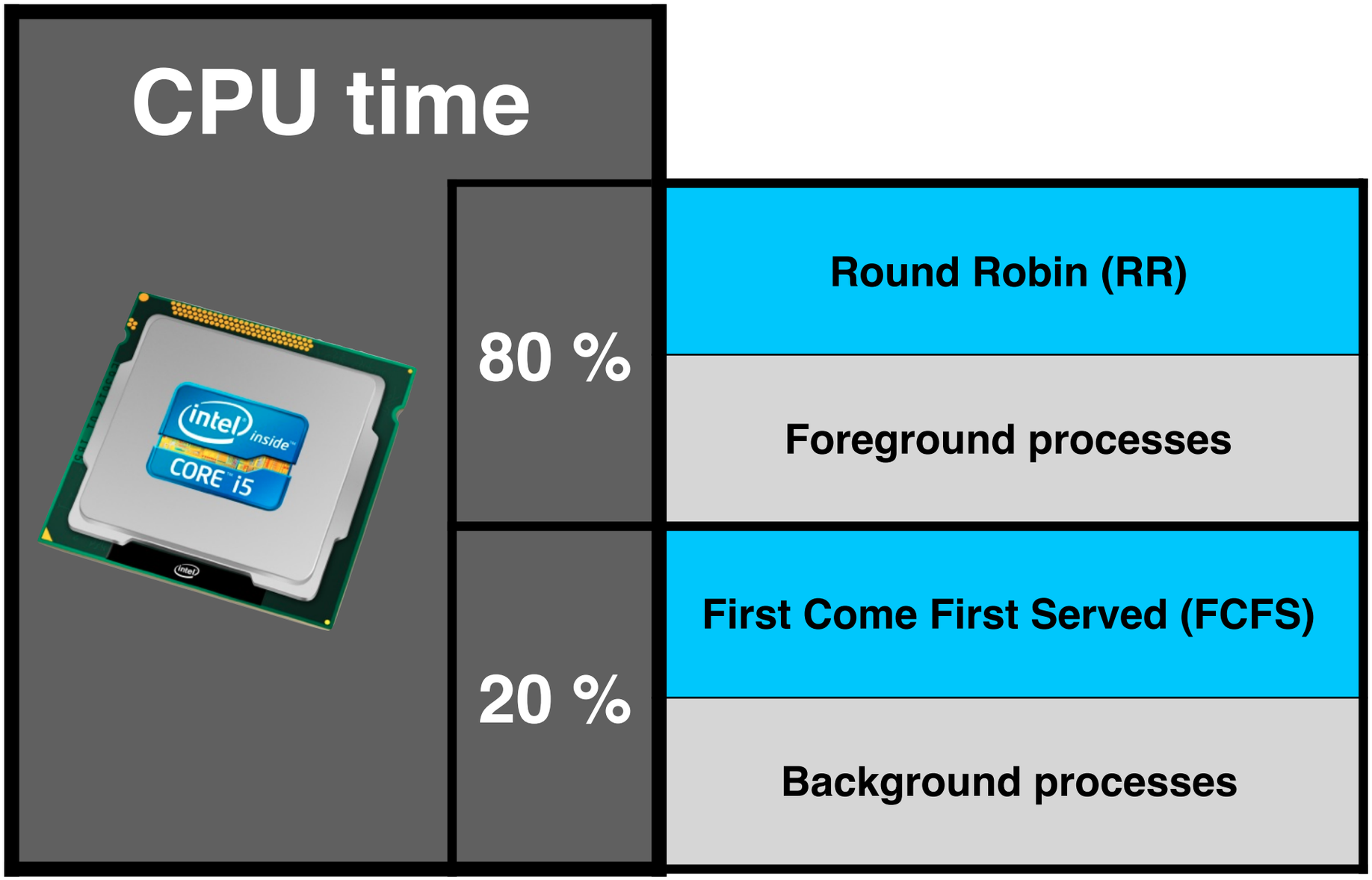
Multilevel queue scheduling (4)

Scheduling must be done between the queues.

- **Fixed priority scheduling**; (i.e., serve all from foreground then from background). Possibility of starvation.
- **Time slice** – each queue gets a certain amount of CPU time which it can schedule amongst its processes.

Multilevel queue scheduling (5)

Example of time slicing among multilevel queues. Foreground process are given 80 % of the CPU time for RR scheduling and background processes are given 20 % of the CPU time for FCFC scheduling.



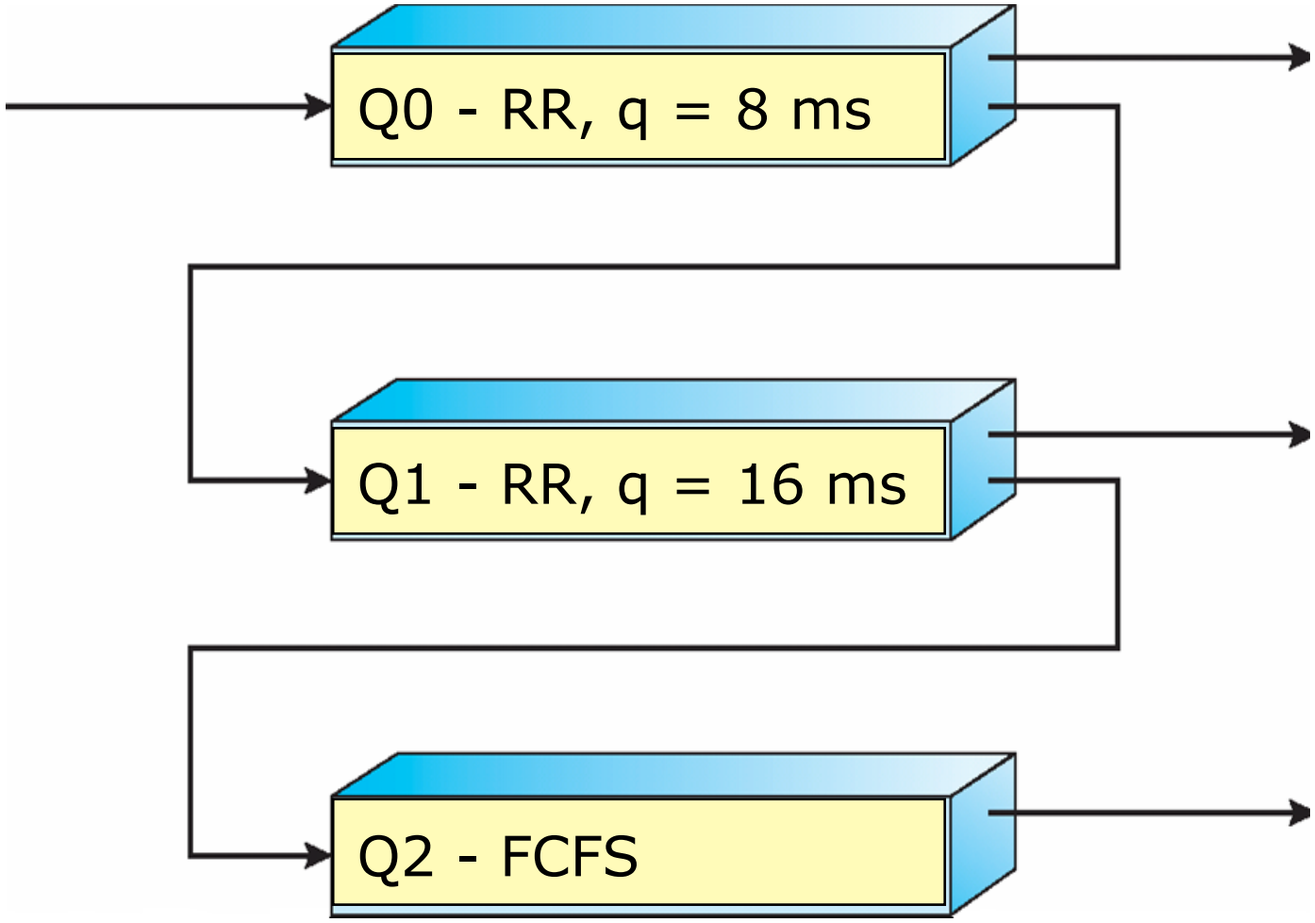
Multilevel feedback queue scheduling

The idea is to separate processes according to the characteristics of their CPU bursts. If a process uses too much CPU time it will be moved to a lower-priority queue.

Design objectives

Multilevel feedback queue scheduling design objectives.

- ★ Give preference to short CPU bursts.
- ★ Give preference to I/O bound processes.
- ★ Separate processes into categories based on their need for the CPU.



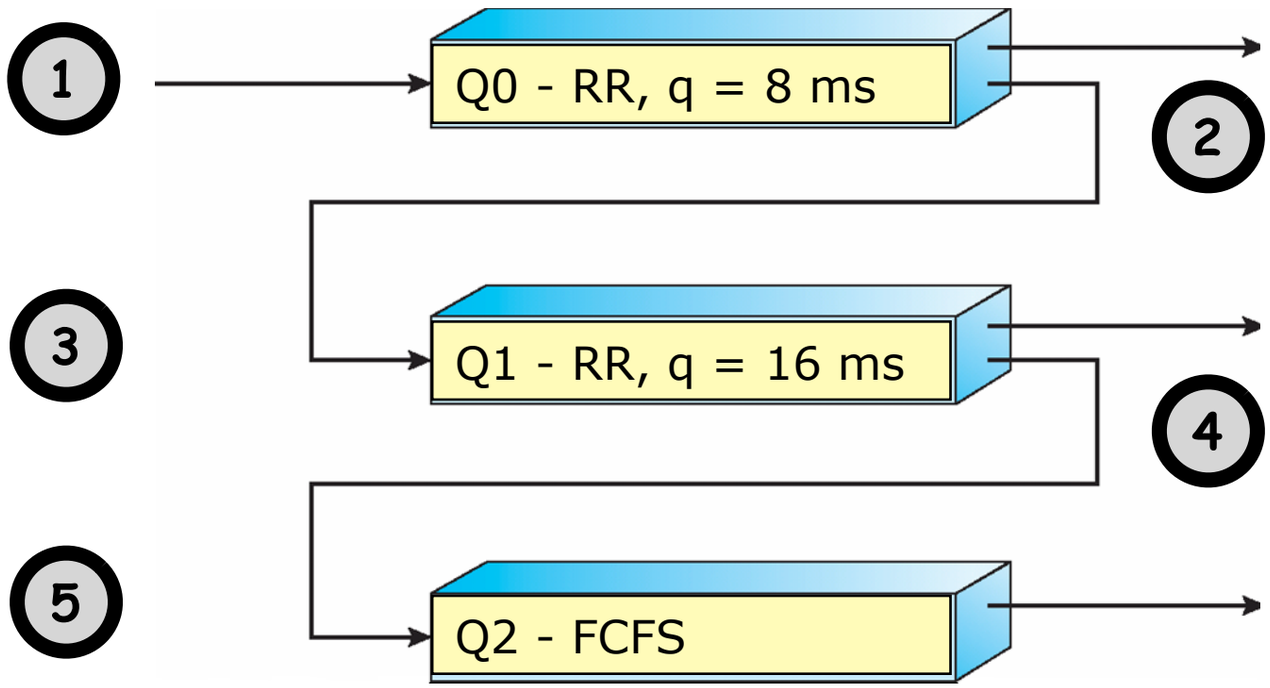
Priorities

- The scheduler first executes all processes in Q0.
- Only when Q0 is empty will it execute processes in Q1.
- Similarly, processes in Q2 will only be executed if Q0 and Q1 are empty.

Preemption

- A process that arrives at Q1 will preempt a process in Q2.
- A process in Q1 will in turn be preempted by a process arriving at Q0.

Example



1. A new job enters queue Q0 which is served FCFS. Each job gets at most 8 milliseconds of CPU time.
2. If it does not finish in 8 milliseconds, the job is moved to queue Q1.
3. At Q1 the job is again served FCFS and receives 16 additional milliseconds.
4. If it still does not complete, it is preempted and moved to queue Q2.
5. Once in Q2, processes are scheduled using FCFS but are run only when Q0 and Q1 are empty.

Indefinite blocking (starvation)

Indefinite blocking (starvation)

In computer science, starvation is a problem encountered in multitasking where a process is perpetually denied necessary resources. Without those resources, the program can never finish its task.

Problem with any sort of priority scheduling?

A process that is ready to run but waiting for the CPU can be considered blocked.

A priority scheduling algorithm can leave some low-priority processes waiting indefinitely.

A steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

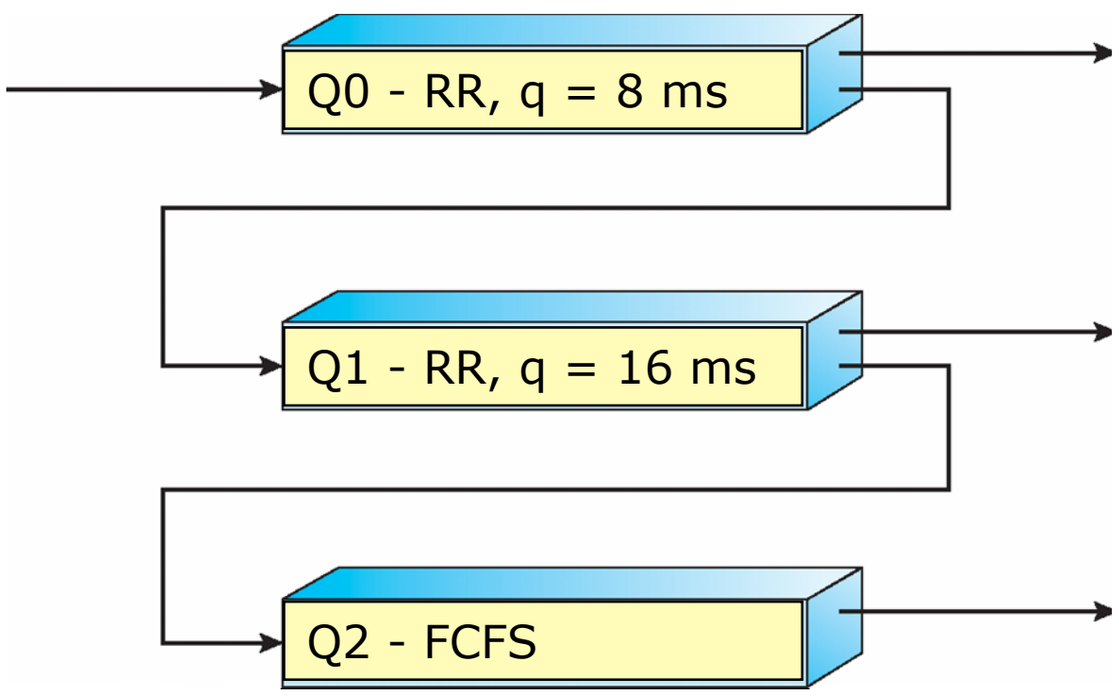
Ageing

Ageing is a scheduling technique used to avoid starvation.

Ageing

Ageing is used to ensure that jobs with lower priority will eventually complete their execution.

- ★ Ageing can be implemented by increasing the priority of a process as time progresses.



Multilevel feedback queue scheduling

A process can move between the various queues.

- ★ Ageing can be implemented this way.

Source: https://en.wikipedia.org/wiki/Aging_(scheduling)