Threads

Pthreads

- a POSIX standard (IEEE 1003.1c) API for thread creation and synchronization.
- API specifies behavior of the thread library.
- Common in UNIX/Linux operating systems.

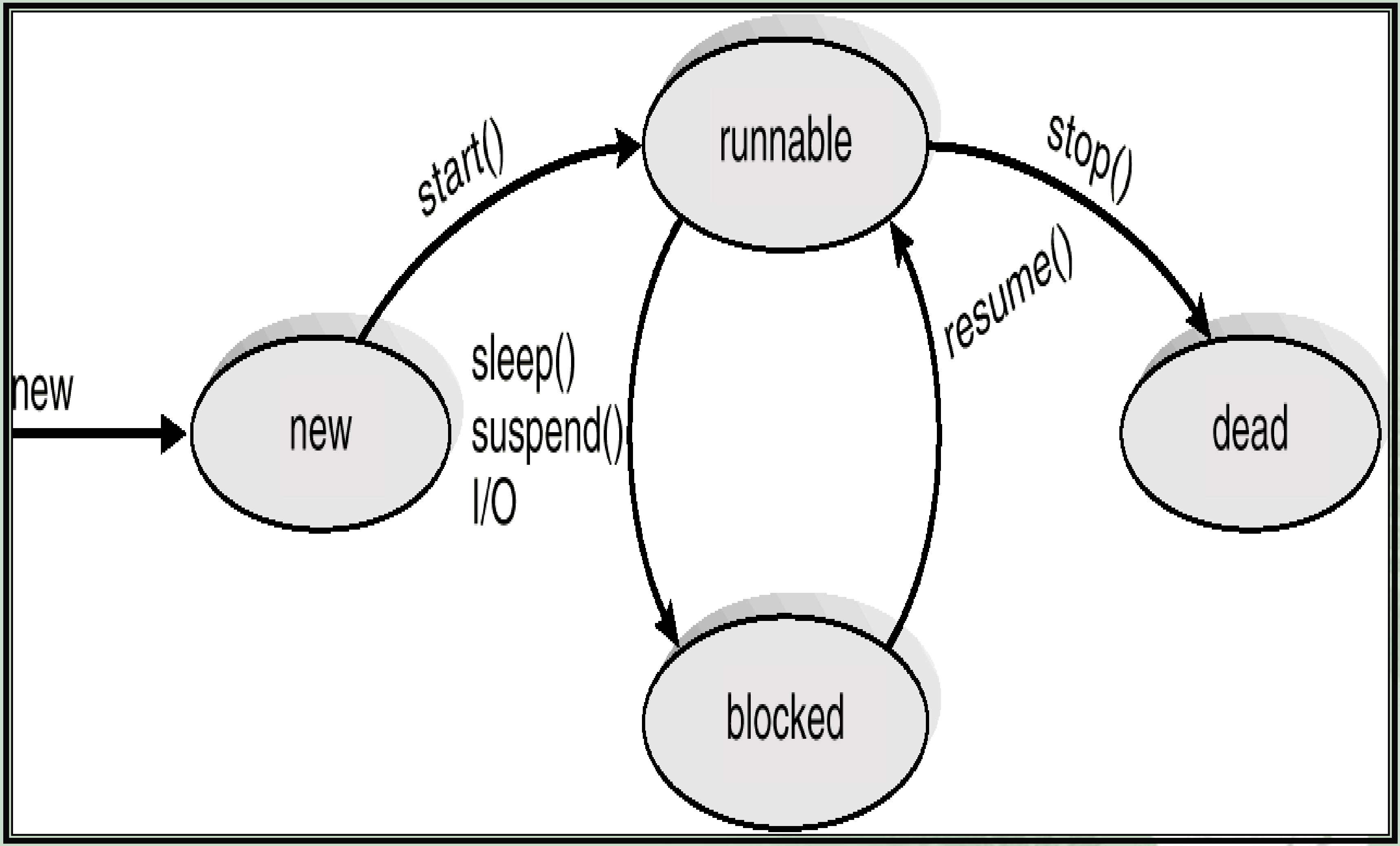
For example, the following table compares timing results for the **fork()** subroutine and the **pthread_create()** subroutine. Timings reflect 50,000 process/thread creations, were performed with the time utility, and units are in seconds, no optimization flags.

Platform	fork()		pthread_create()	
	user	sys	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	0.1	2.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	0.4	4.3	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	1.0	12.5	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	2.2	15.7	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	0.6	8.8	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	30.7	27.6	0.6	1.1

Java Threads

- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface
- Java threads are managed by the JVM.

Java Thread States



Thread implementations

```
/* POSIX */
main(){
…
pthread_create(f,arg);
…
}

void *f(void *arg){
 …
 pthread_exit(status);
}
```

```
/* Win32  */
main(){
…
CreateThread(f,arg);
beginthreadex(f,arg);

}

DWORM f(DWORD
arg){
…
ExitThread(status);
endthread(status);

}
```

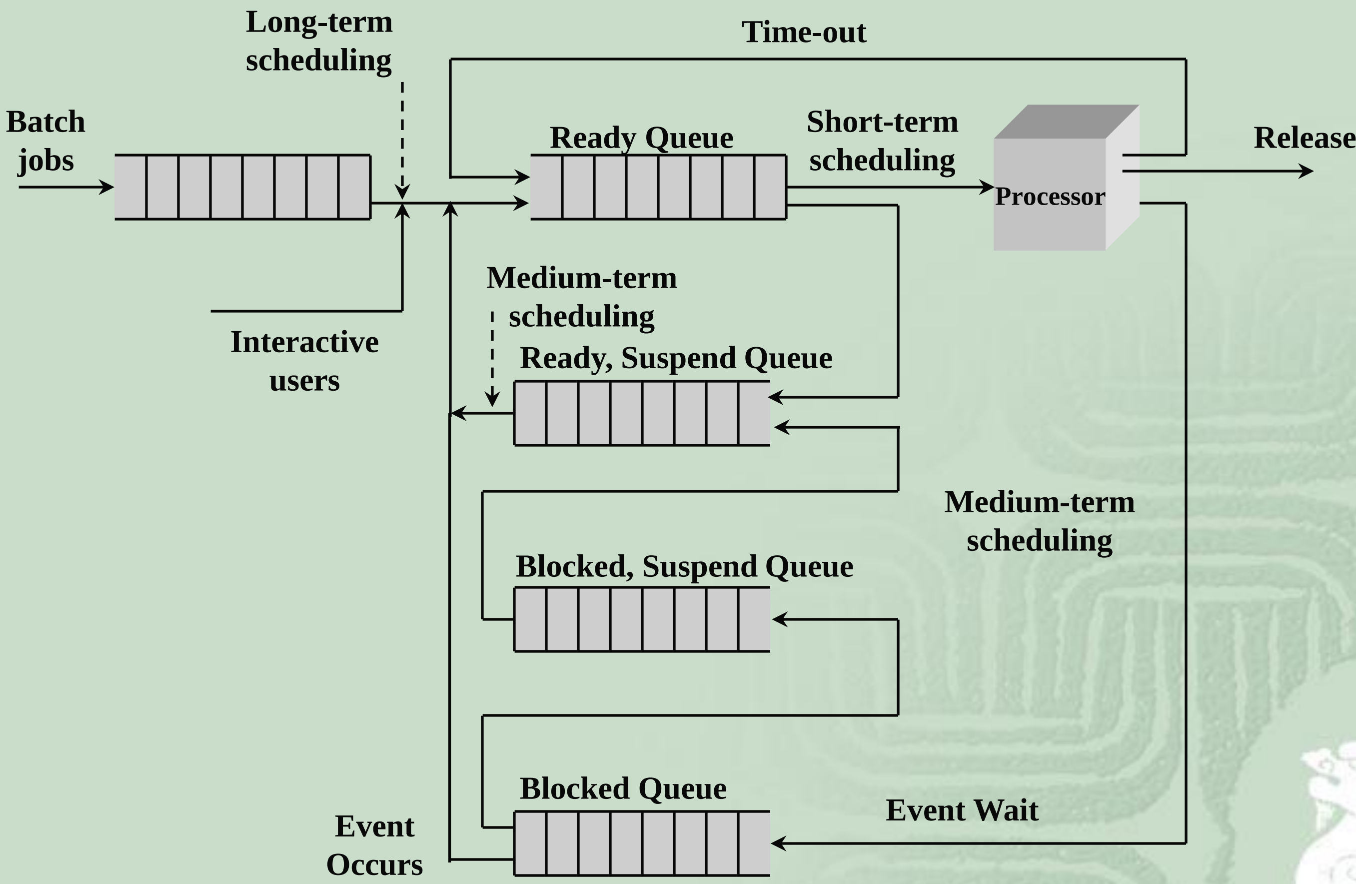
```
/* Java */
main(){
 MyThread t;
 t = new MyThread();
 t.start();
}

class MyThread
     extends Thread{

  public void run(){
…
return;
  }
}
```

CPU Scheduling

CPU Scheduling

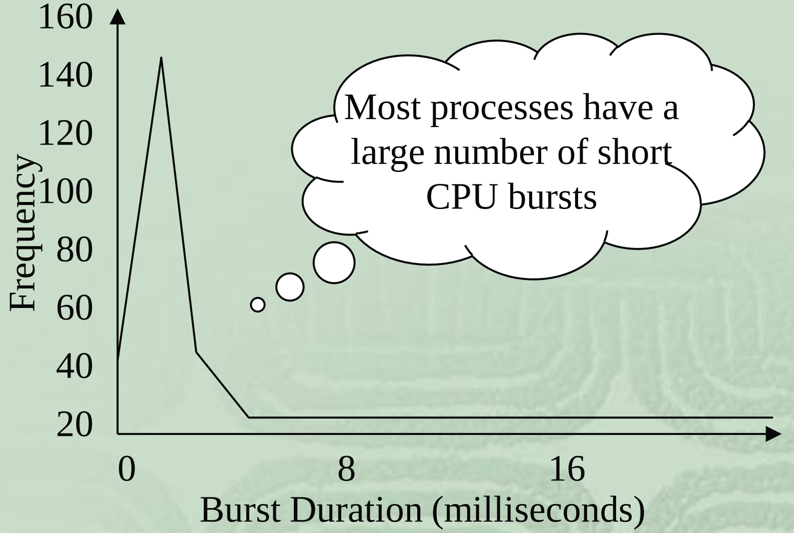
- Fundamentally, scheduling is a matter of managing queues to minimize queuing delay and to optimize performance in a queuing environment.
- Scheduling needs to meet system objectives, such as:
  - minimize response time
  - maximize throughput
  - maximize processor efficiency
  - support multiprogramming
- Scheduling is central to OS design

Queuing Diagram for Scheduling



Observations

- Processes are dependent on I/O
  - dependency level varies
- Processes cycle
  - compute
  - wait for I/O
- Process execution is characterized by
  - length of CPU burst
  - number of bursts



The Scheduler

- Whenever the CPU is idle
  - scheduler must pick another process that is ready
- Scheduling takes place because:
  - I/O request
  - Process interrupted
  - I/O completion
  - Process termination

Scheduling Criteria

- Turnaround time
  - Time from submission to completion
- Response time
  - Time to start responding (interactive users)
- Waiting time:
  - A process spends waiting in the ready queue.
- Throughput
  - Number of jobs processed per unit of time
- Processor utilization
  - Percent of time CPU is busy

Scheduling Criteria (continued…)

- Predictability
  - Same time/cost regardless of load on the system
- Fairness
  - No process should suffer starvation
- Enforcing priorities
  - Favor higher priority processes
- Balancing resources
  - Keep system resources busy

Priorities

- Processes are assigned a priority.
- The scheduler always chooses a process of higher priority over one of lower priority.
  - The highest priority, ready process is always scheduled first.
- Pure priority scheduling may result in starvation of lower-priority processes.
- If unacceptable, priority of a process could change with its age or execution history.

Preemptive vs. Non-preemptive

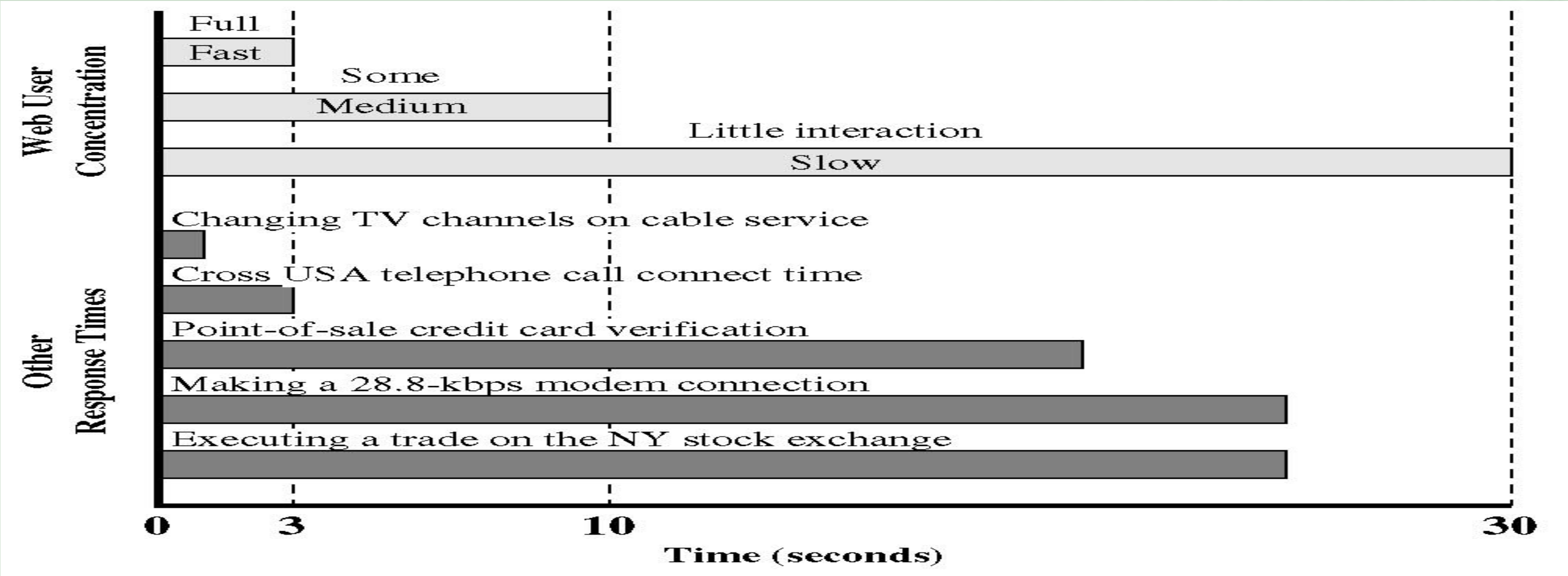
- Scheduling that only takes place due to I/O or process termination is non-preemptive.
- Preemptive scheduling allows the operating system to interrupt the currently running process and move it to the ready state.
  - new process arrives
  - clock interrupt
- Preemptive scheduling:
  - incurs greater overhead (context switch)
  - provides better service to the total population of processes
  - may prevent one process from monopolizing the processor

Response Time

- User productivity tends to improve with a more rapid response time.
  - Becomes very noticeable as response time drops below 1 second
- User time or “think time” – Time user spends thinking about the response.
- System time – Time system takes to generate its response.
  - Short response times are important
  - If the system gets too slow to a user, the user may abort the operation

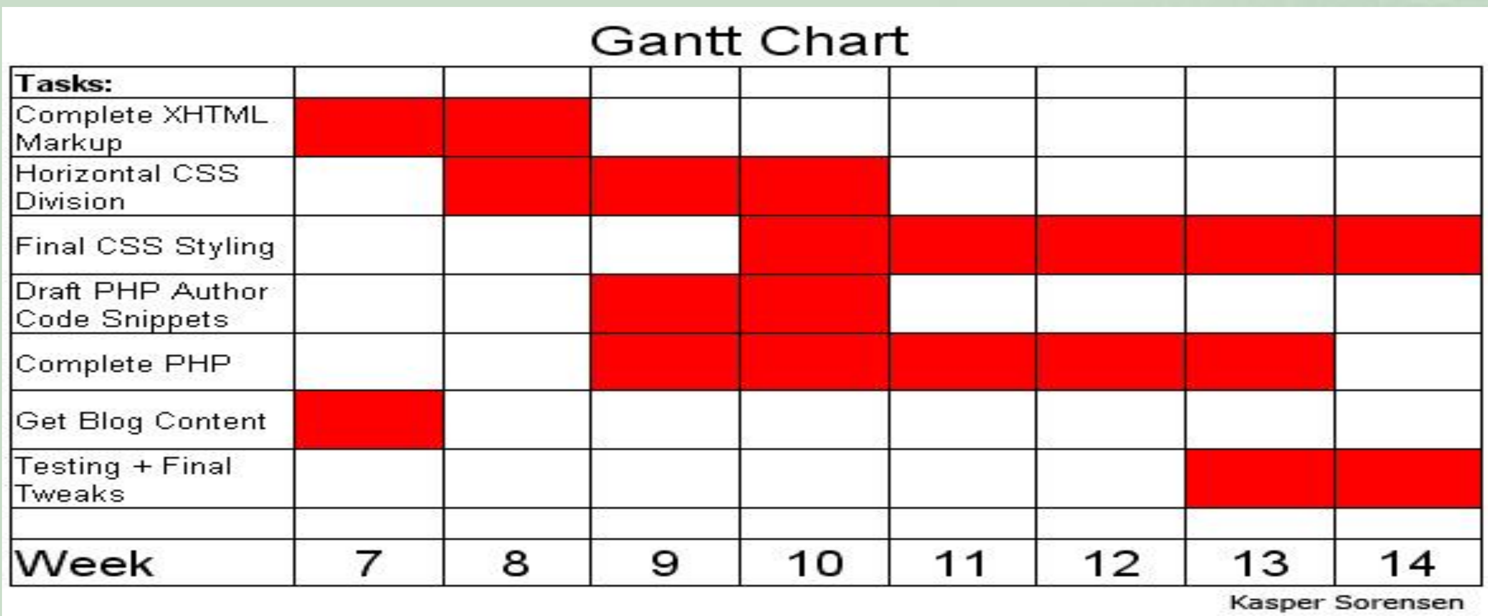
Response Time (continued…)

- Web Pages – Loading a page in 3 seconds or less increases the user's attention
  - User may abort after 10s or more
- Must balance response time with the cost required
  - Faster/more expensive hardware may be required



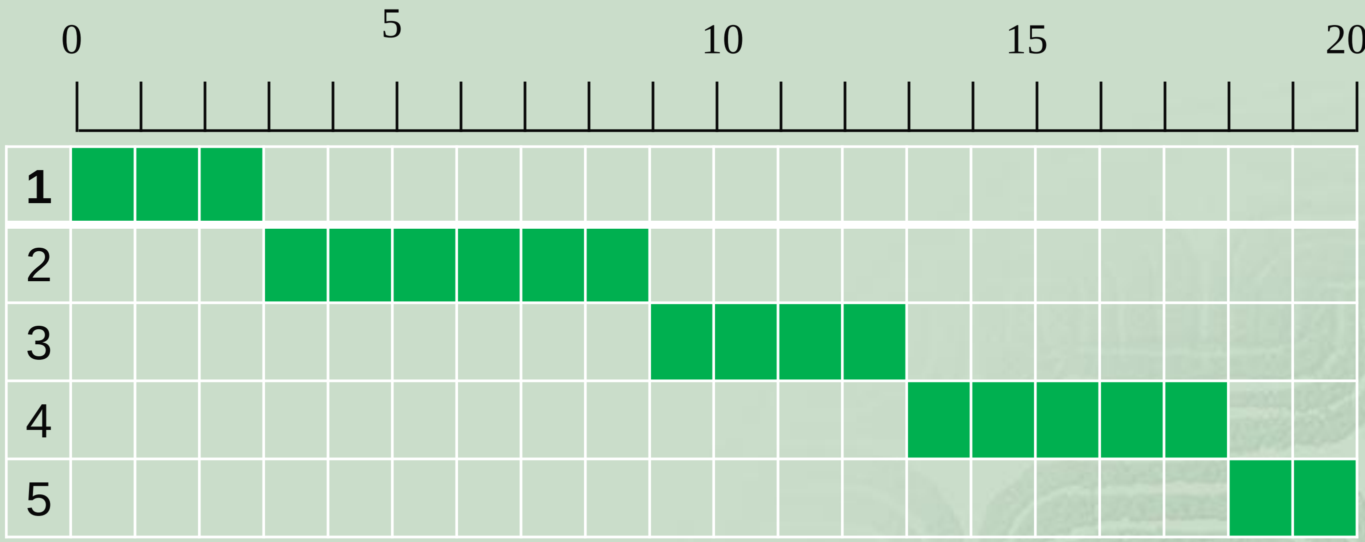
Gantt Chart

- A **Gantt chart** is a type of bar chart, developed by Henry Gantt, that illustrates a project schedule.



First-Come-First-Served (FCFS)

Process	Arrival	Service
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



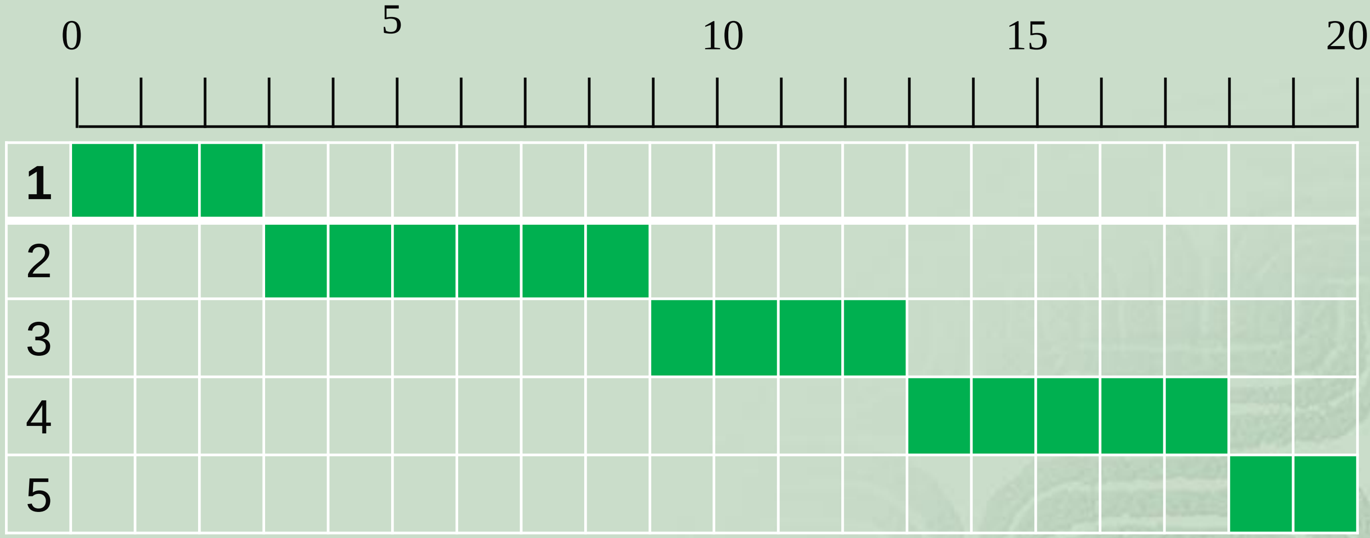
- First-Come-First-Served - Each process joins the Ready queue
- When the current process ceases to execute, the oldest process in the Ready queue is selected

FCFS (continued…)

- A short process may have to wait a very long time before it can execute.
- Favors CPU-bound processes over I/O-bound processes.
- FCFS, although not an attractive alternative for a uniprocessor system, when combined with a priority scheme may prove to be an effective scheduler.

First-Come-First-Served (FCFS)

Process	Arrival	Service
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Turnaround times: p1: 3-0 = 3; p2: 9-2 = 7; p3: 13-4 = 9;
p4: 18-6 = 12; p5: 20-8 = 12;

Ave Turnaround time: (3+7+9+12+12)/5 = 8.6

Throughput (within 1000 time units): 1000/8.6 ≈ 116

Total time required: 20

FCFS (continued…)

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive at time 0 in the order: P_1 , P_2 , P_3
  The simplified Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS (continued…)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1 .$$

- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case.

Convoy Effect of FCFS

- All the other processes wait for the one big process to get off CPU

- Non-preemptive: Once the CPU has been allocated to a process, that process keeps the CPU until it gets off