ECEN 3753: Real-Time Operating Systems

Scheduling

Scheduling

- Why have scheduling?
  - Need ability to share resources between multiple Tasks
  - OS decides what to run next
  - Between decisions, OS needs to prepare the old and new Task
    - Context Switching
- How the scheduler chooses the next Task depends on the algorithm chosen
  - These scheduling decisions are regularly being applied

Scheduling Goals

- Meeting all of the following goals is difficult
- Goals:
  - Maximize throughput
  - Maximize how resources are utilized
  - Minimize response time
  - Minimize wait time
  - Minimize time it takes to start a new task
  - Share resources appropriately
    - What are some examples of resources that need sharing?

Criteria for Scheduling

- CPU Utilization
  - In order to have efficient use of CPU it needs to remain as busy as possible as long as there is work to do
- Response Time
  - Amount of time the task spends waiting to be executed
- Turnaround Time
  - Amount of time it takes from the time a Task was submitted until it finished completion
- Waiting Time
  - Amount of time a tasks waits for resources (CPU, Memory, etc)
- Throughput
  - The amount of "fully completed" work that can be completed in a specific amount of time. [TasksCompleted/Time]

Turnaround = Waiting + Execution.

Non Preemptive Scheduling

- Scheduling algorithm waits for the current Task to let go of the CPU (note: Task either terminates, yields or blocks)
  - Also known as Cooperative multitasking
- Why choose a non-preemptive scheduler?
  - What are the advantages?
  - What are the disadvantages?

Preemptive scheduling

- The OS forces the current Task to let go of the CPU
  - This allows another Task to resume or a new one to be started
  - Also known as preemptive multitasking
- Why choose a preemptive scheduler?
  - What are the advantages?
  - What are the disadvantages?
- What is a human-scale example of "good" line-cutting?

Context Switching

- Required for preemptive scheduling
- Likely required for non-preemptive if non-trivial or long-lived Tasks
- Allows the "pausing" of a Task
- Current Task's information is stored for later retrieval
- Next Task's information is then retrieved
- The time it takes to store and retrieve is known as **Context Switch Time**

Note similarities to Interruptability, Interrupt Latency from CPU context protection for ISR (Critical Sections)

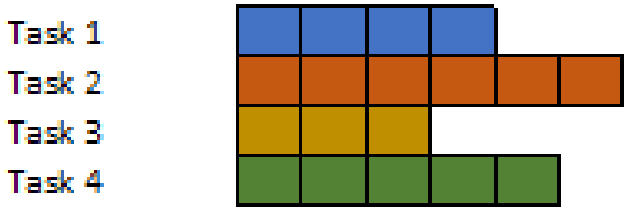
Scheduling Algorithms

- First-Come, First-Served
- Shortest Job First
- Time Slice
- Time Slice with Background Tasks
- Round Robin
- Priority

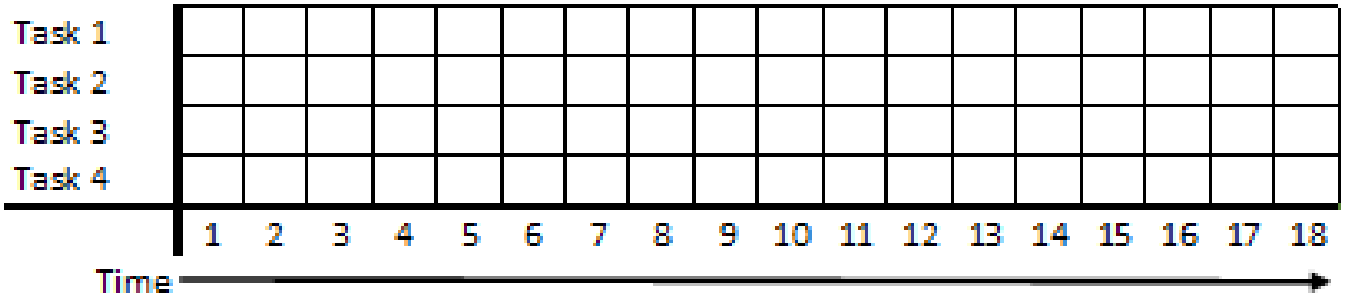
First Come First Served [FCFS]

- Cooperative multitasking (generally)
- What Is It?
  - Also known as First-In, First-Out
  - Select the Task that is at the head of the ready queue FIFO
  - This scheduler is typically non preemptive
- Pros
  - FIFO is easy to implement
  - No context switch needed since the Task runs to completion
- Cons
  - If the first task is large, other tasks will be "stuck" behind it, resulting in low throughput

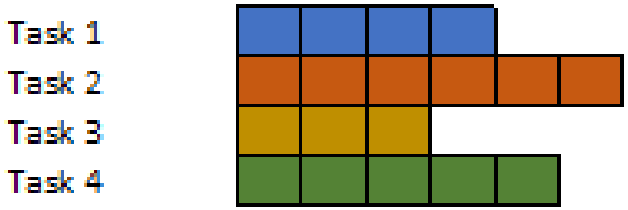
First Come First Served [FCFS] (continued)



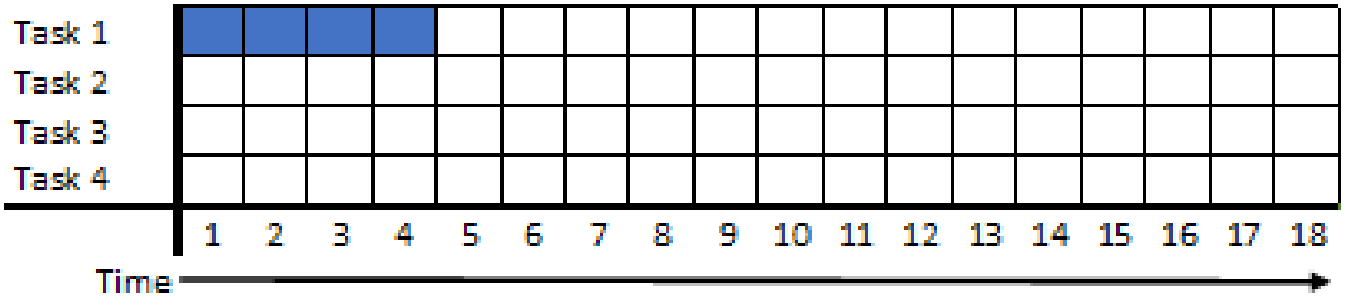
First Come First Served



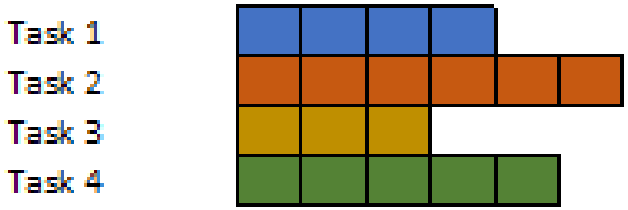
First Come First Served [FCFS] (continued)



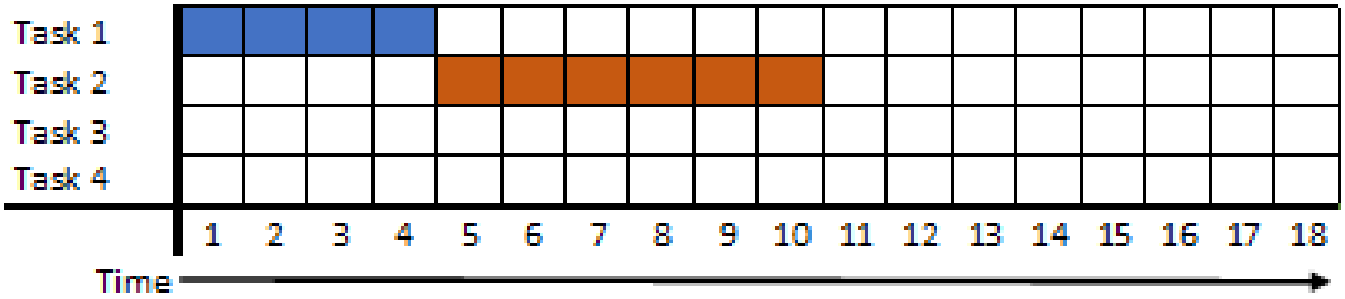
First Come First Served



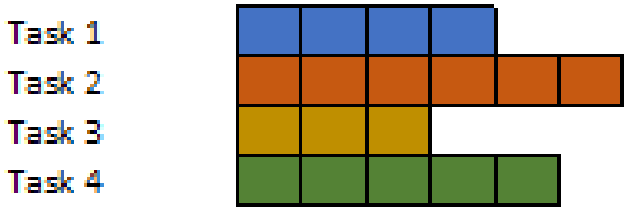
First Come First Served [FCFS] (continued)



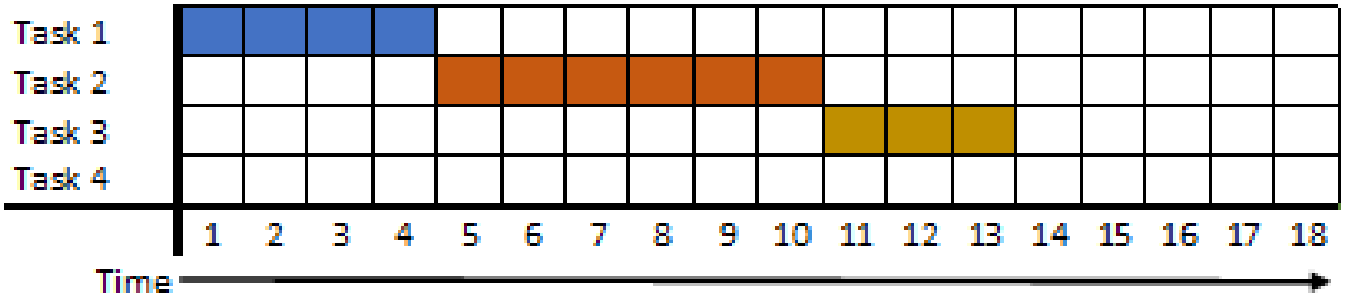
First Come First Served



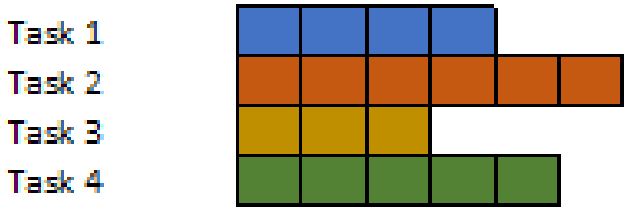
First Come First Served [FCFS] (continued)



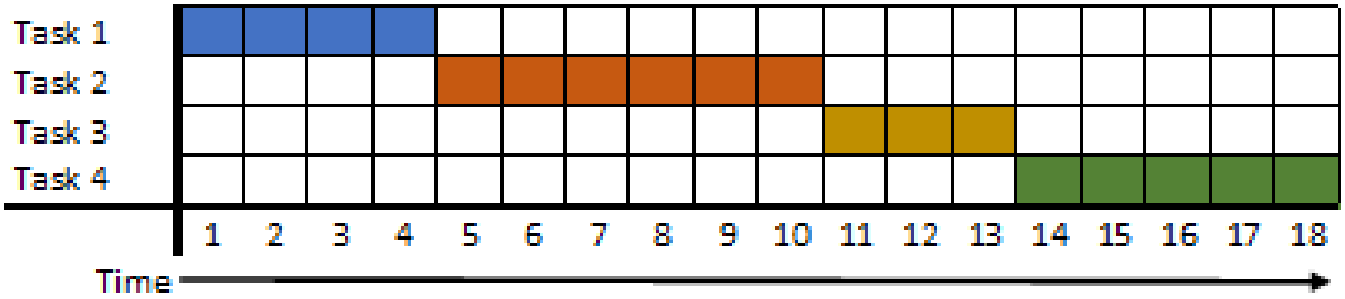
First Come First Served



First Come First Served [FCFS] (continued)



First Come First Served



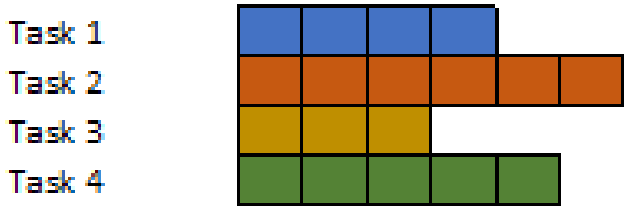
First Come First Served Unit Tests

- Check if everything is initialized correctly
- Check if the first Task has a waiting time of zero
- Check if the second Task has a waiting time the same as the first Task's execution time, etc

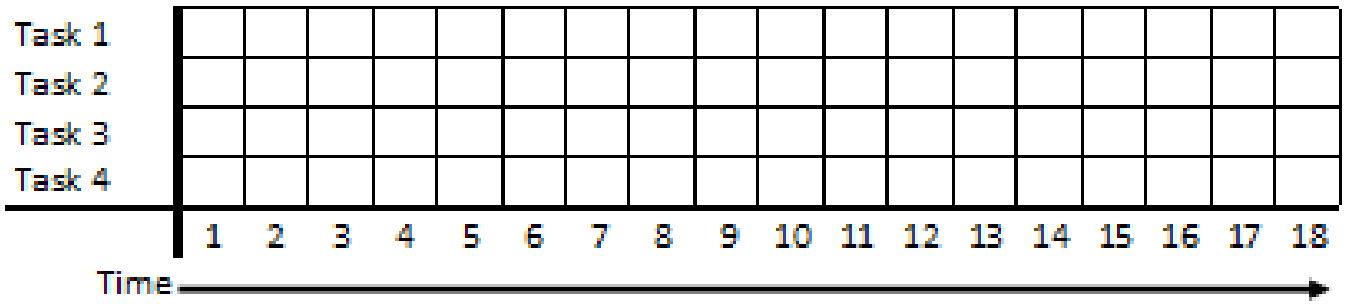
Shortest Job First [SJF]

- What Is It?
  - Scheduler places jobs with shortest completion time as the highest priority
  - Non Preemptive:
    - Task with the shortest time gets to execute
  - Preemptive:
    - Task with the shortest remaining time gets to execute
- Pros
  - Maximizes Task throughput
- Cons
  - Larger Task potentially could never run, or at least suffer long latencies
  - Imposes hard requirements on knowledge in regards to time-complexity

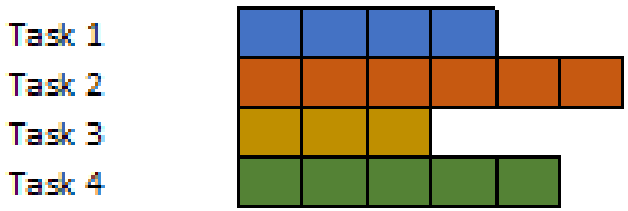
Shortest Job First [SJF] (continued)



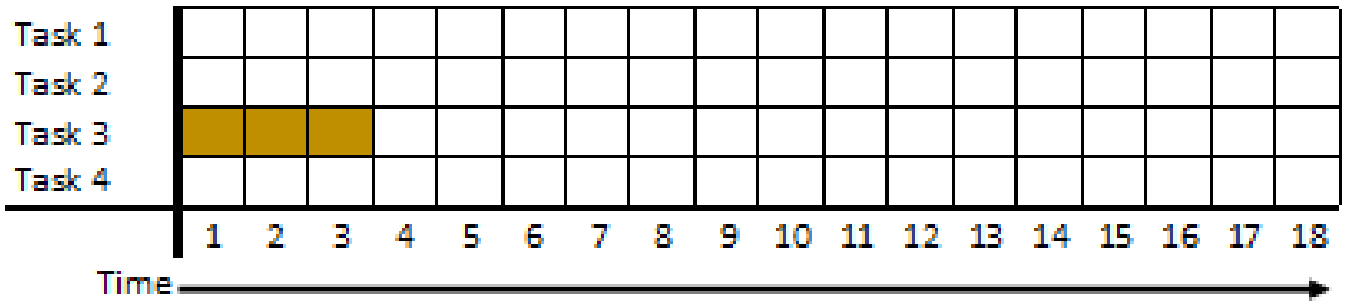
Shortest Job First



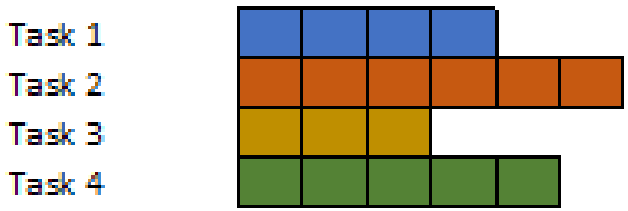
Shortest Job First [SJF] (continued)



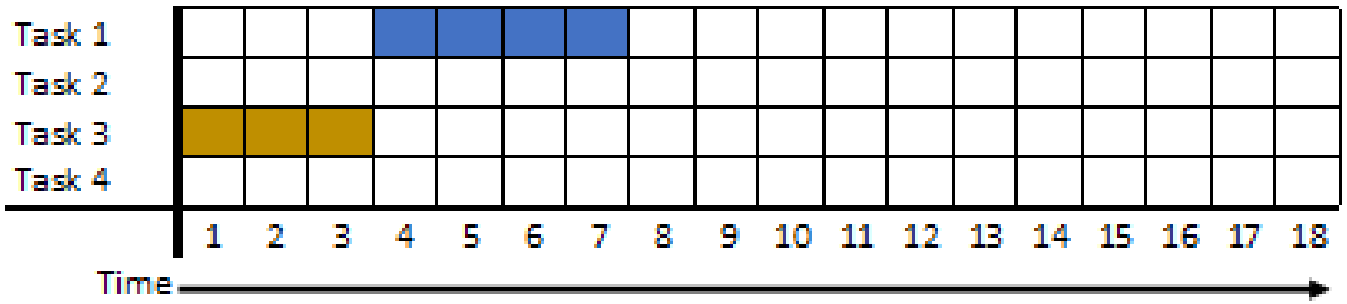
Shortest Job First



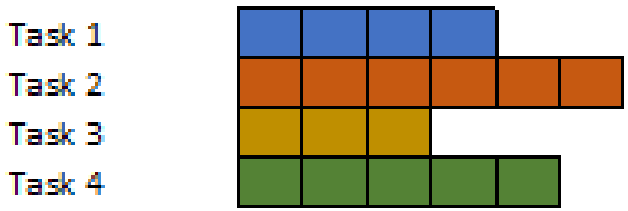
Shortest Job First [SJF] (continued)



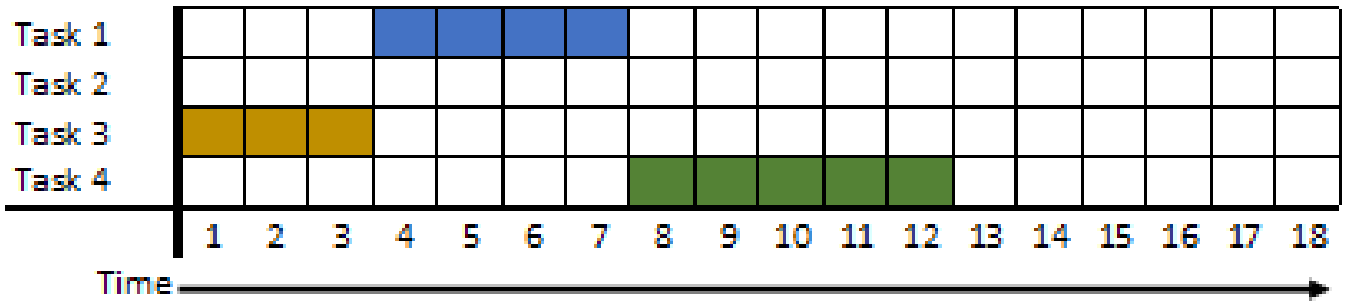
Shortest Job First



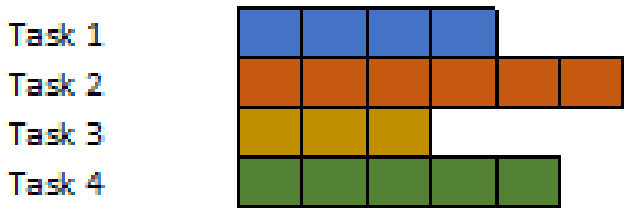
Shortest Job First [SJF] (continued)



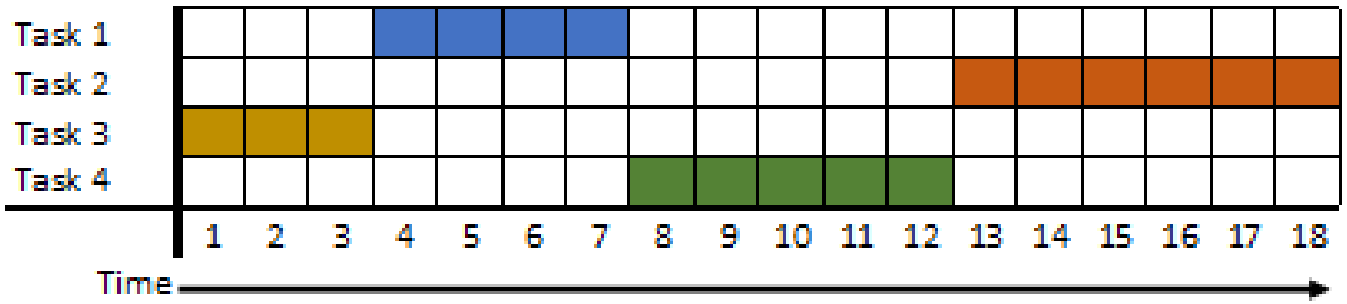
Shortest Job First



Shortest Job First [SJF] (continued)



Shortest Job First



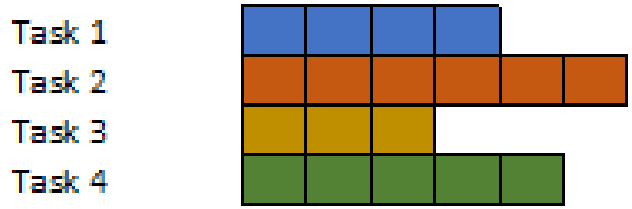
Shortest Job First Unit Tests

- Check if everything is initialized correctly
- Check if the shortest Task has a waiting time of zero
- Check if the second shortest has a waiting time equal to the exeuction time of the first Task, etc

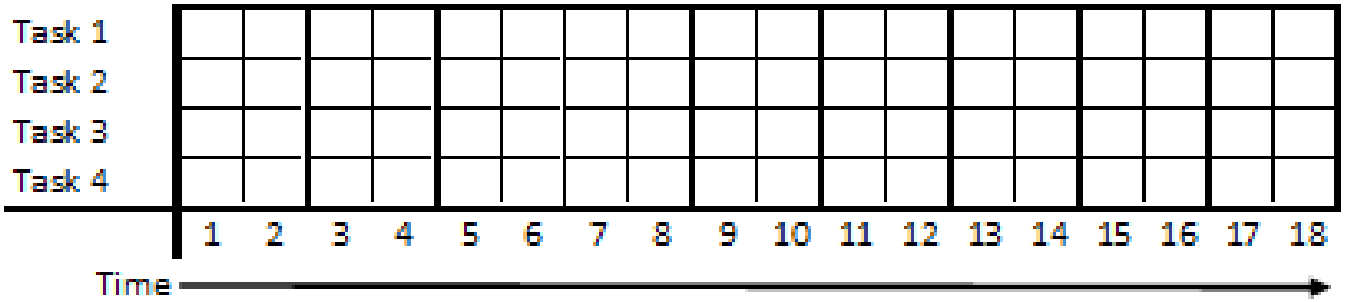
Time Slice [TS]

- Preemptive multitasking
- What Is It?
  - Time slots are created for each Task.
  - Once the "timer" ends an interrupt is thrown for the next task to start
- Pros
  - Fully deterministic
- Cons
  - If the Task has nothing to do during it's time then the slot time is wasted

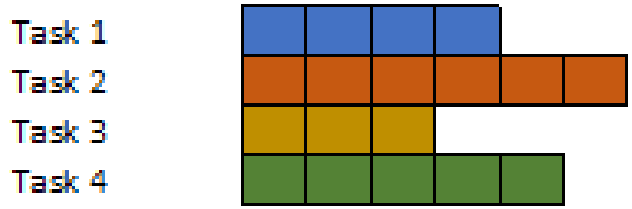
Time Slice (continued)

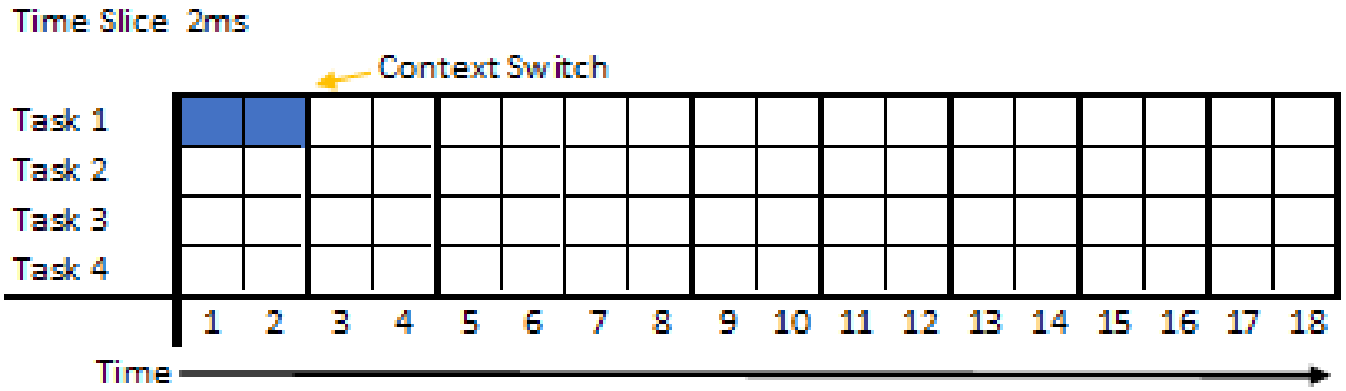


Time Slice 2ms

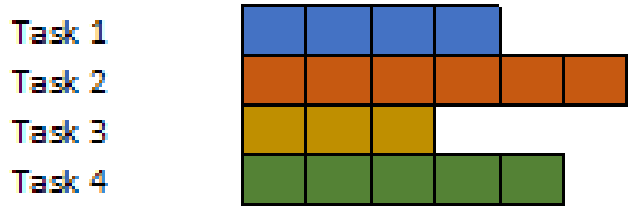


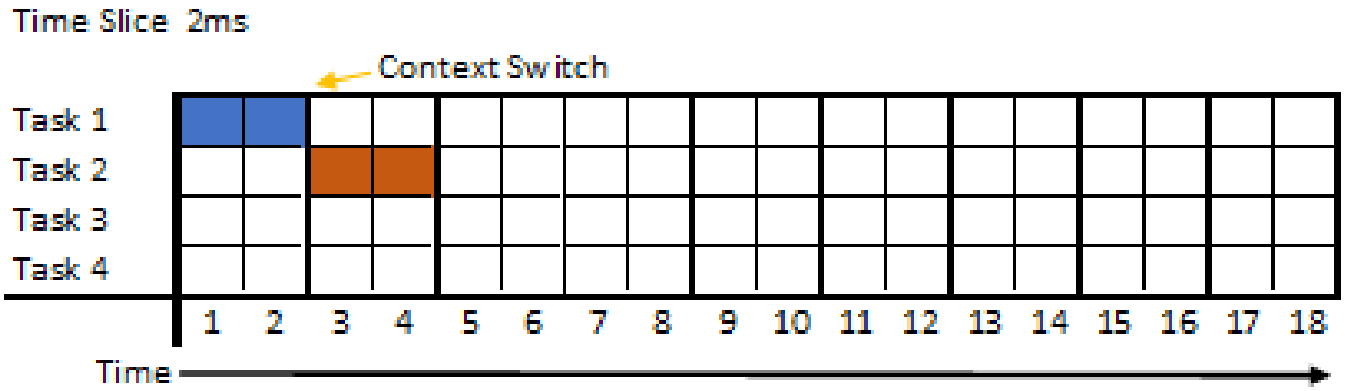
Time Slice (continued)



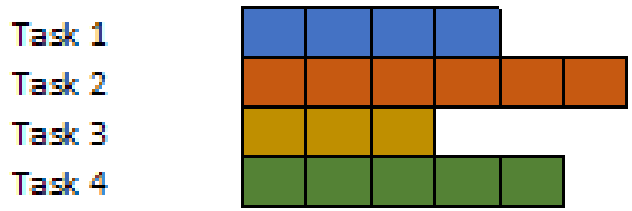


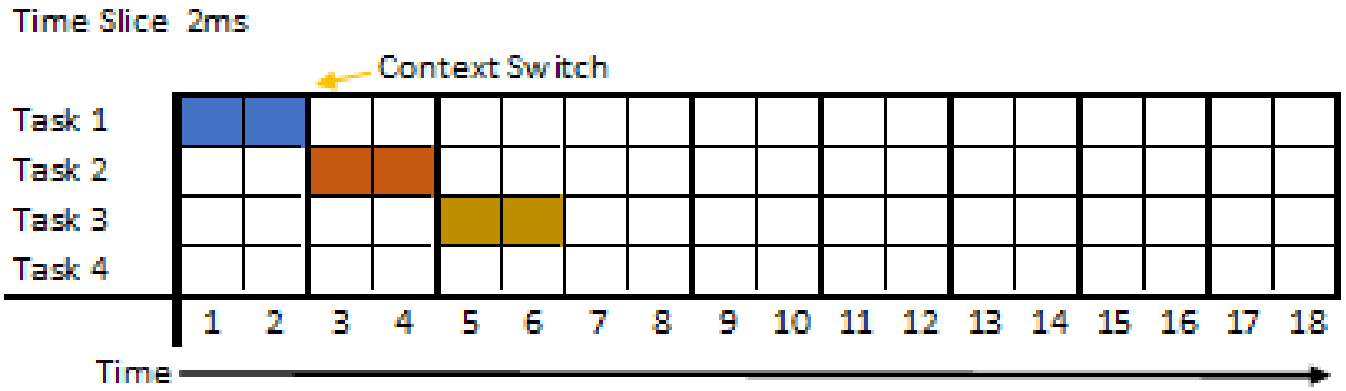
Time Slice (continued)



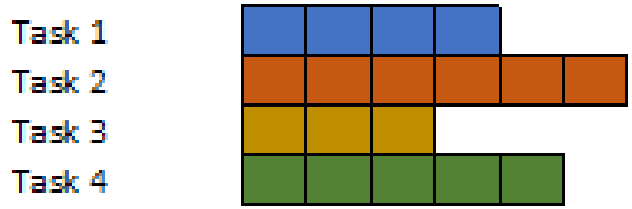


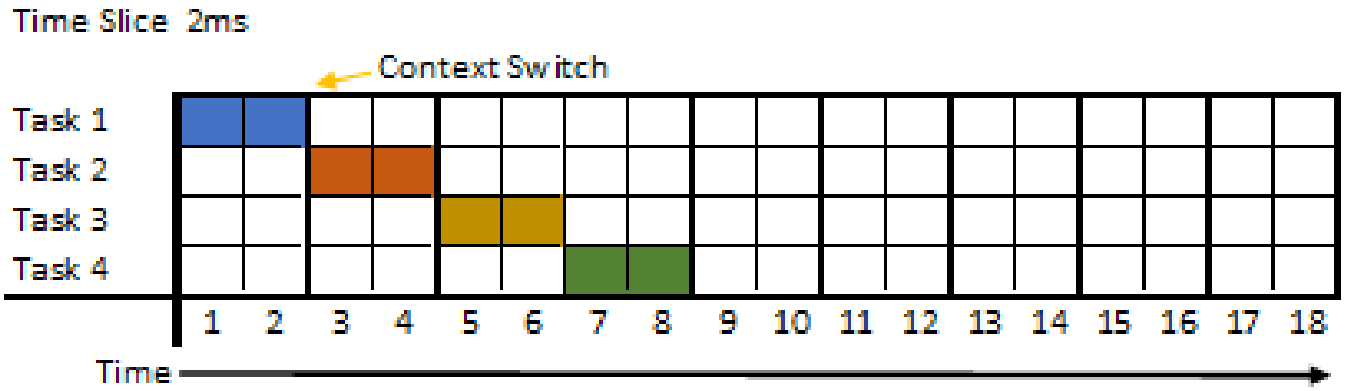
Time Slice (continued)



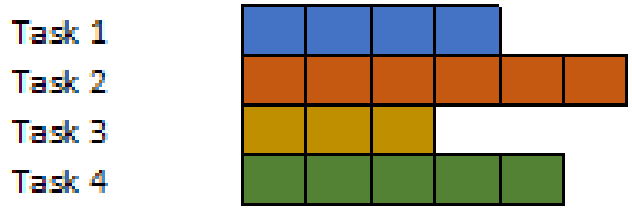


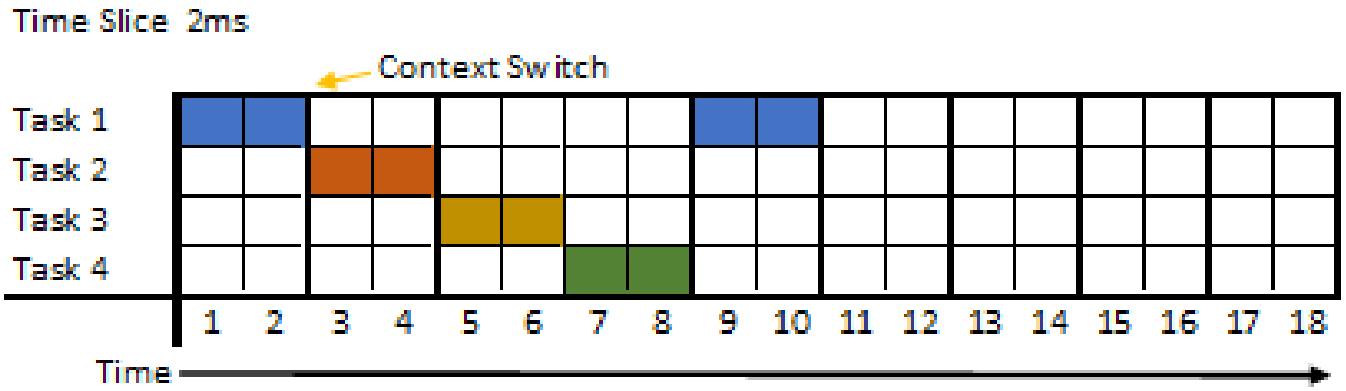
Time Slice (continued)



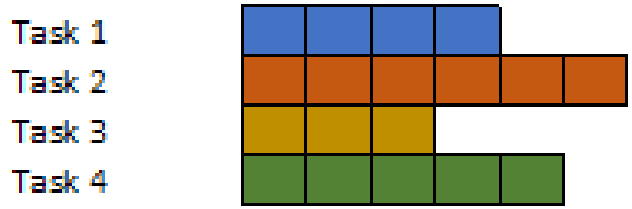


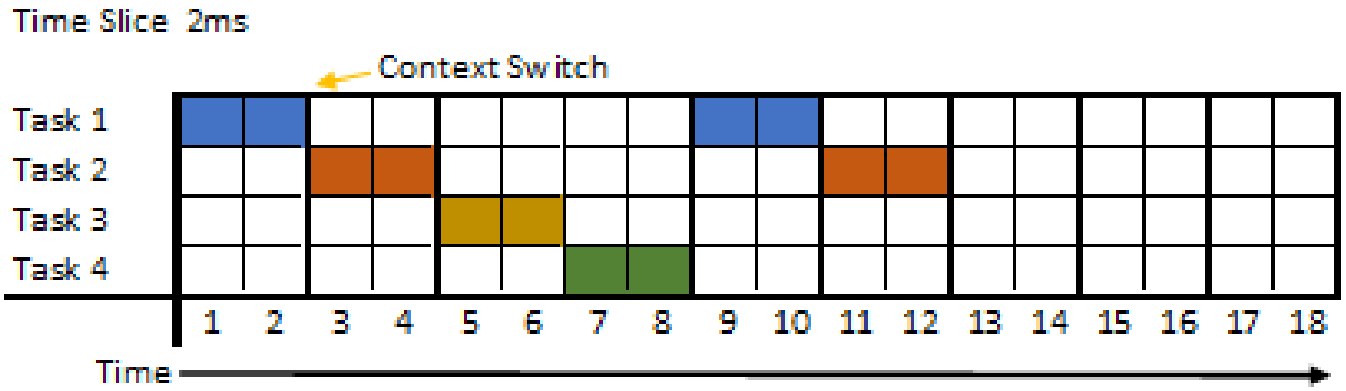
Time Slice (continued)



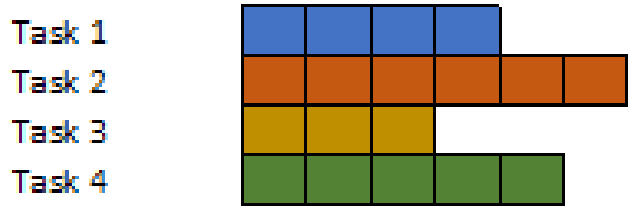


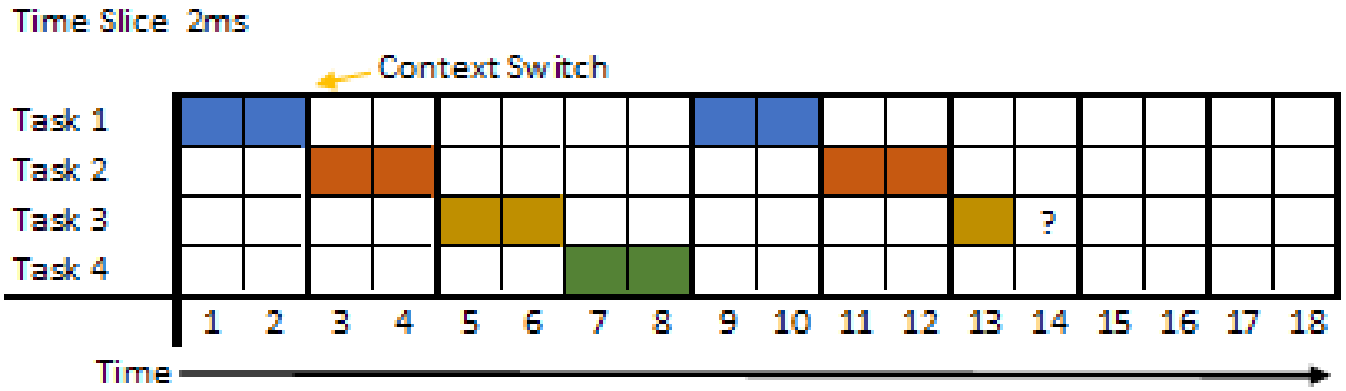
Time Slice (continued)



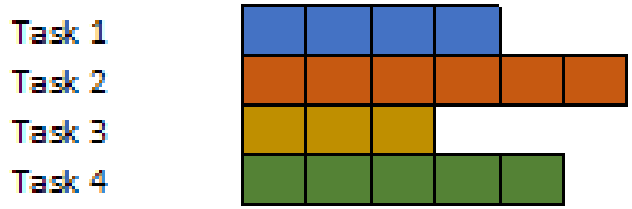


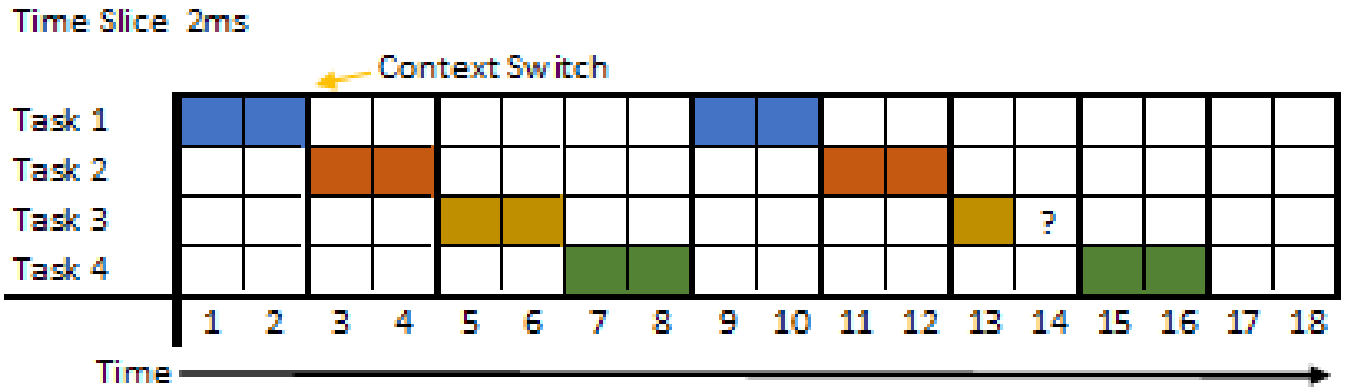
Time Slice (continued)



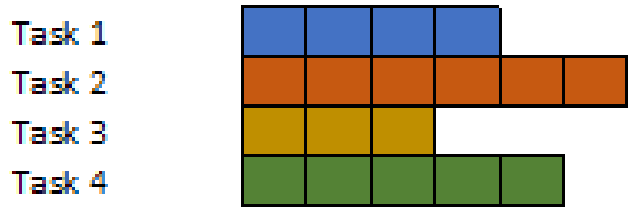


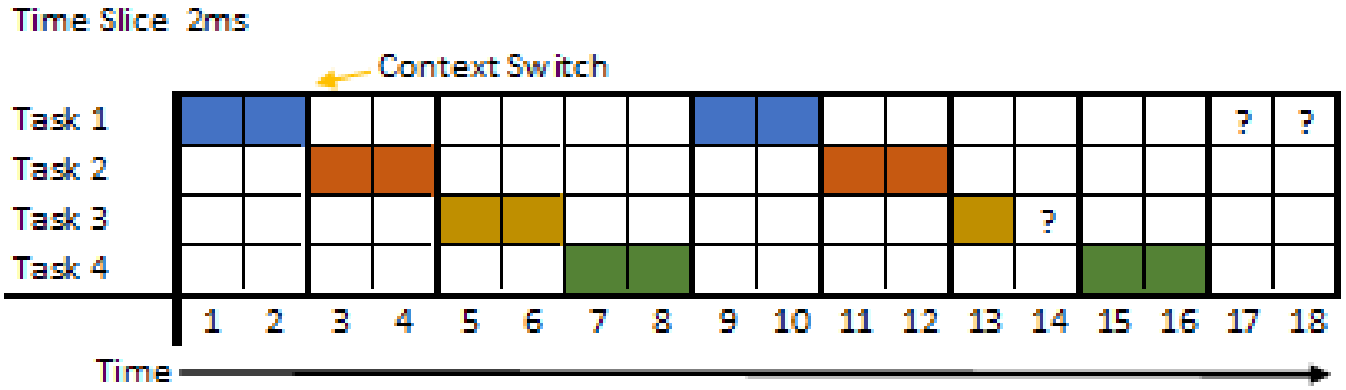
Time Slice (continued)





Time Slice (continued)





During '?':

- spin CPU and wait?
- suspend CPU to reduce power?
- Try to start other tasks sooner?

Time Slice Unit Tests

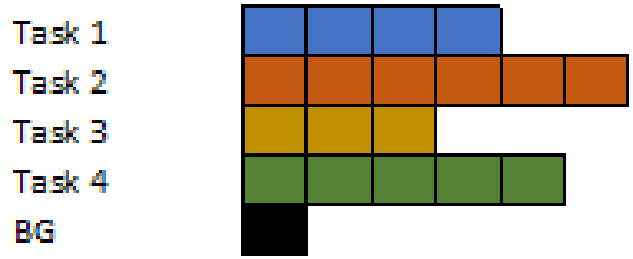
- Check if everything is initialized correctly
- Test that each Task runs for a max time each iteration
- Test with only 1 Task first then add more

Time Slice with Background Task [TSBG]

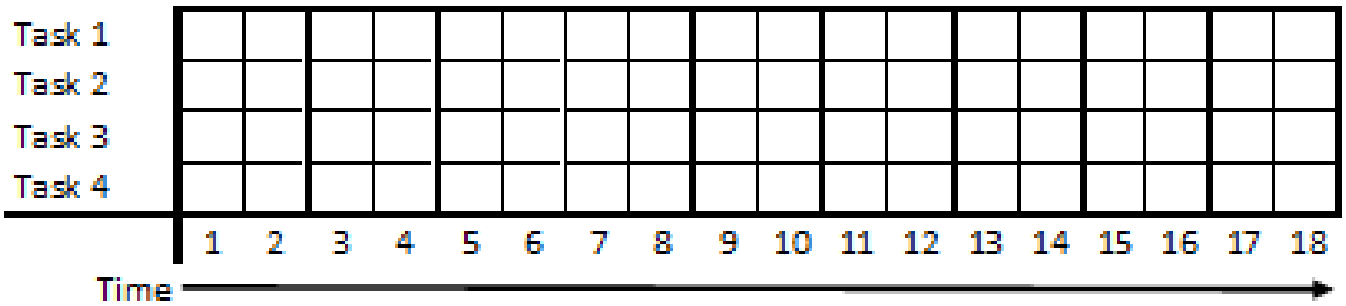
- Preemptive multitasking
- What Is It?
  - Time Slice but adds in ability to run background tasks if a slot is free
- Pros
  - Fully Deterministic
  - Improves Time Slice
- Cons
  - There is a chance that background tasks will never be scheduled
  - Overhead due to context-switching

Real-life example: HDD, early 1990s: servo on timer interrupt, then R/W channel/sequencer, then background (host/ queueing)...degenerate "preemption"

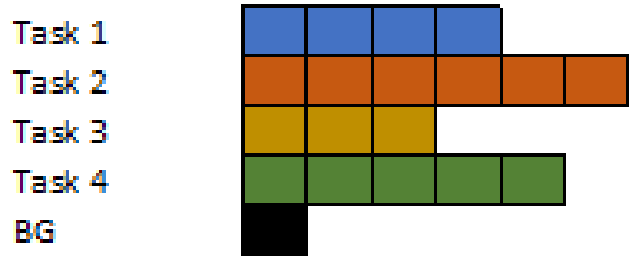
Time Slice with Background Task [TSBG] (continued)

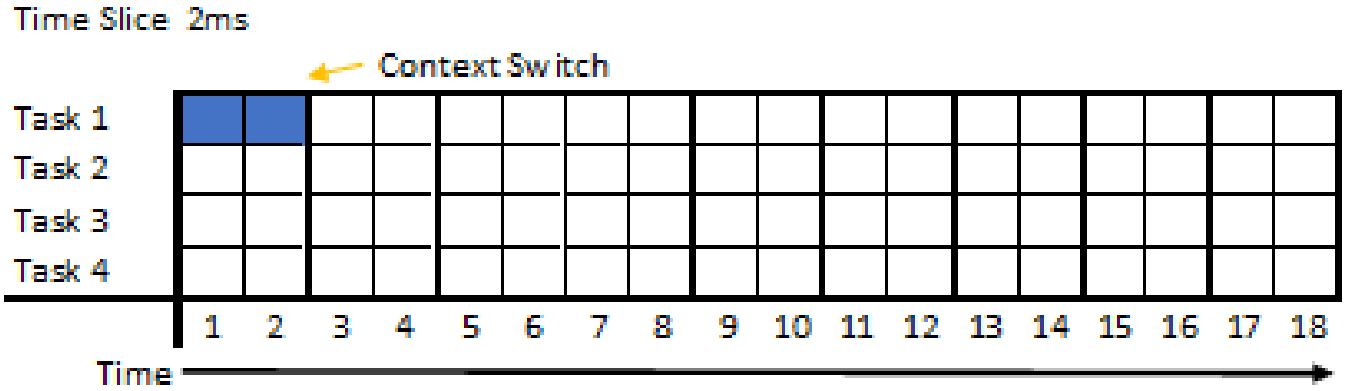


Time Slice 2ms

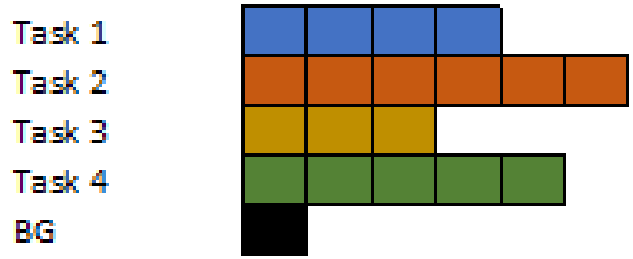


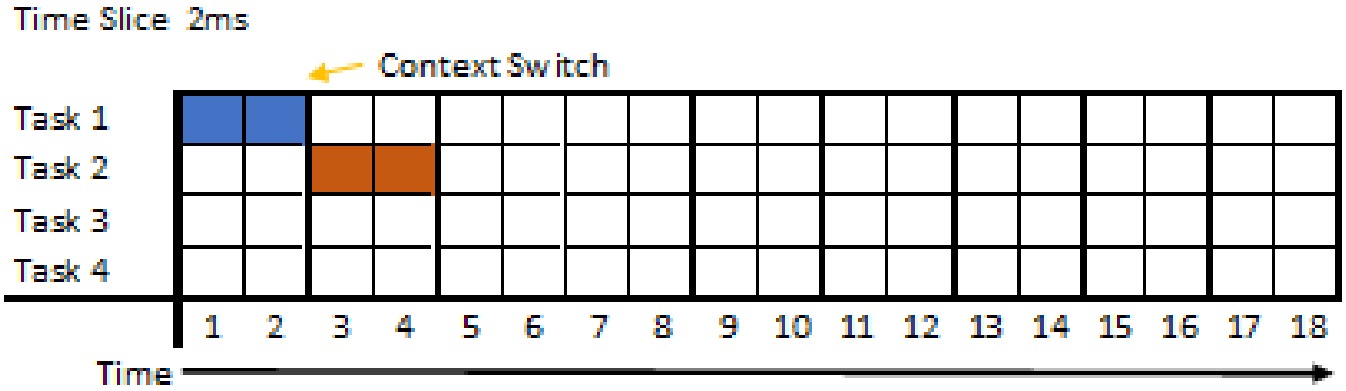
Time Slice with Background Task [TSBG] (continued)



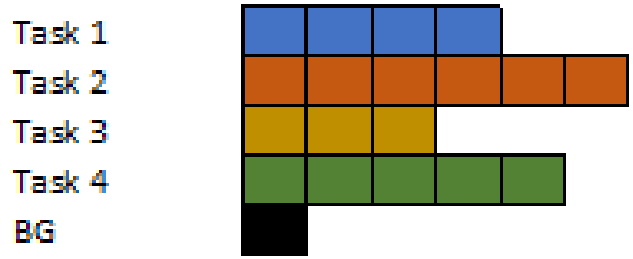


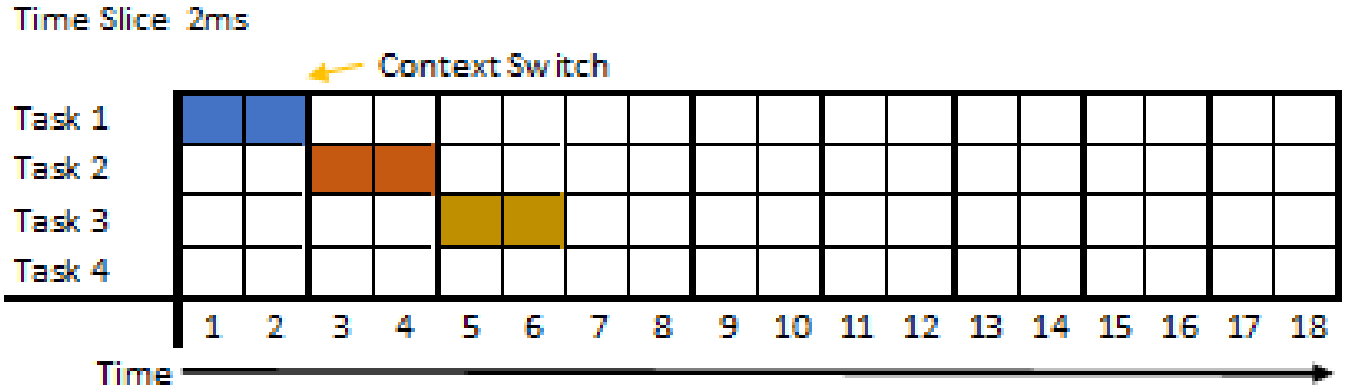
Time Slice with Background Task [TSBG] (continued)



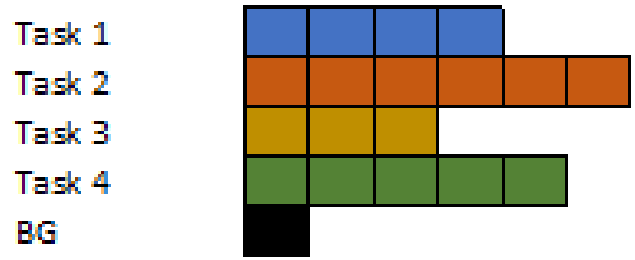


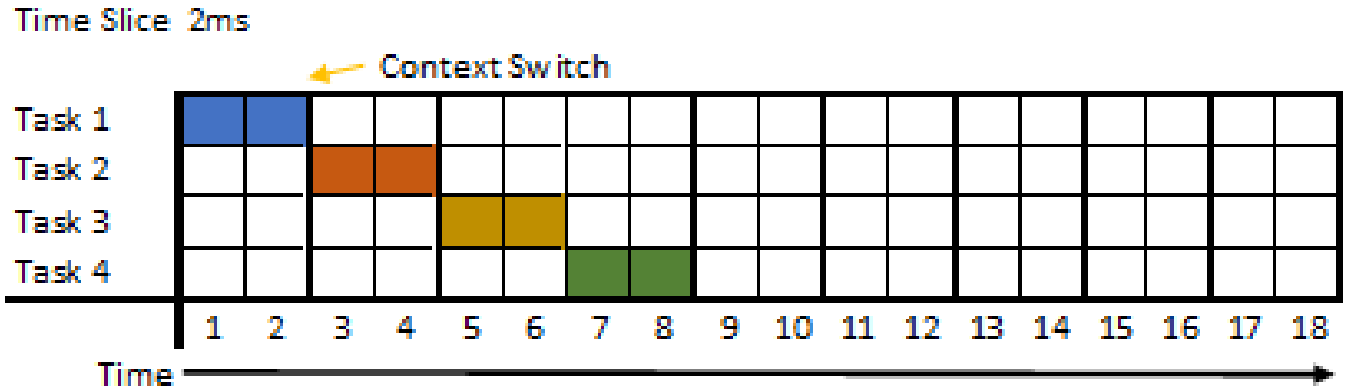
Time Slice with Background Task [TSBG] (continued)



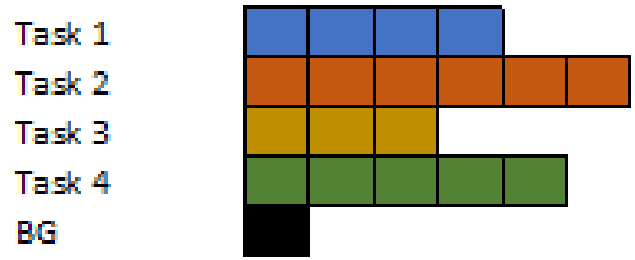


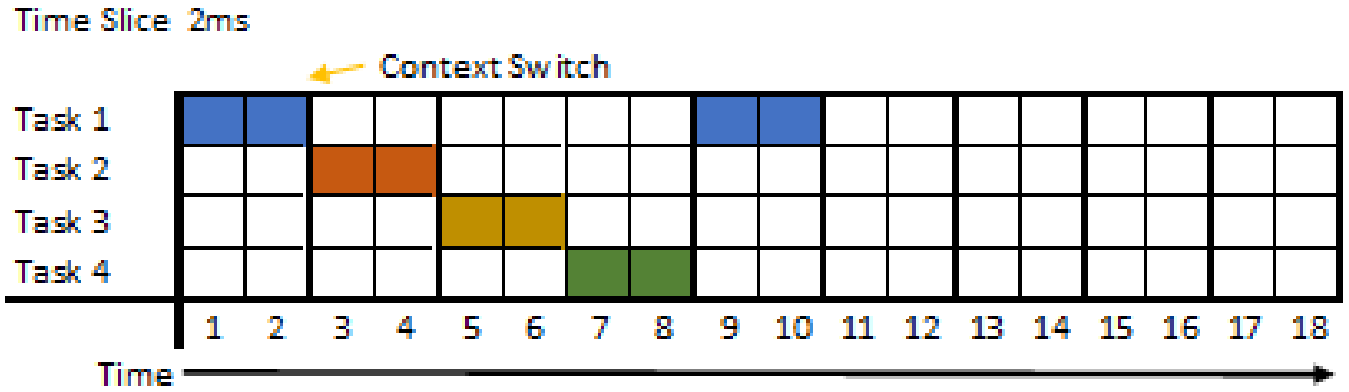
Time Slice with Background Task [TSBG] (continued)



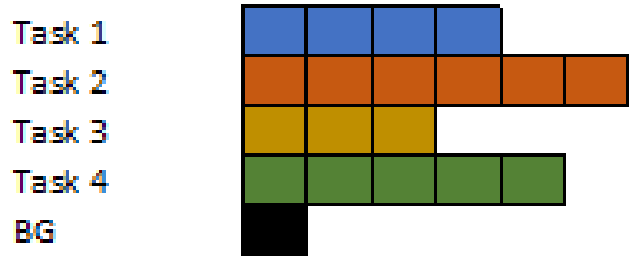


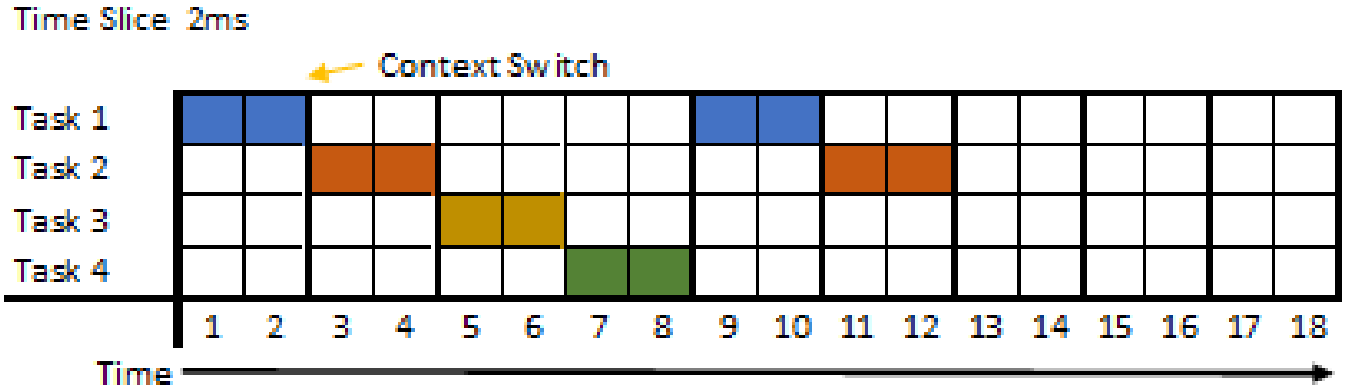
Time Slice with Background Task [TSBG] (continued)



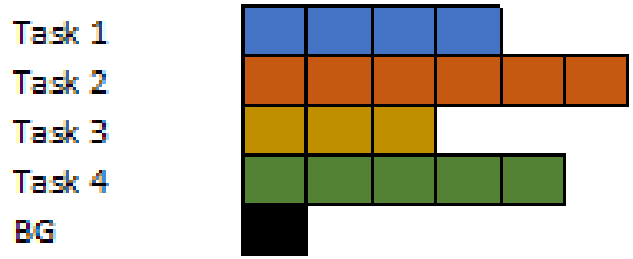


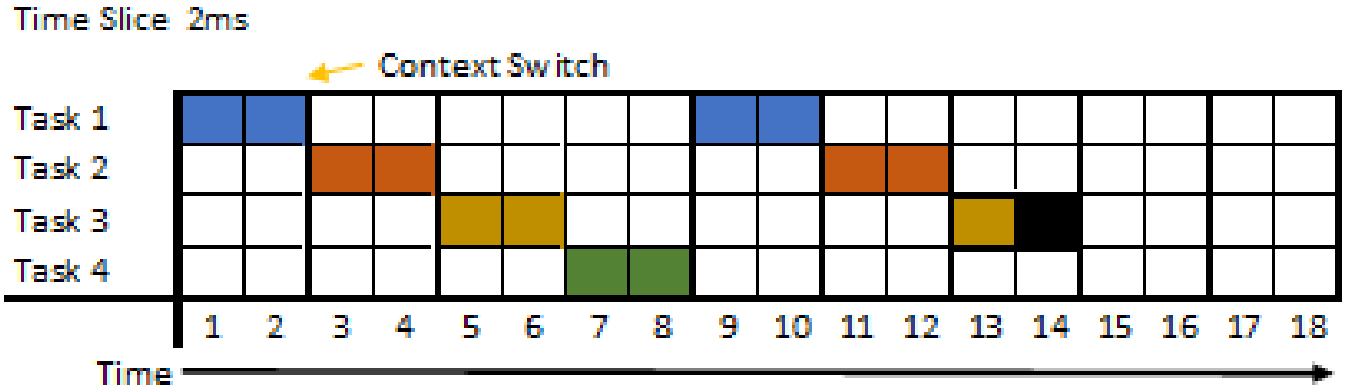
Time Slice with Background Task [TSBG] (continued)



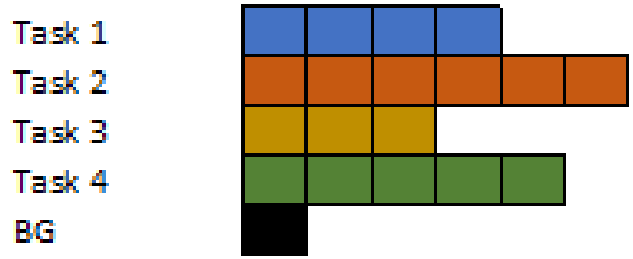


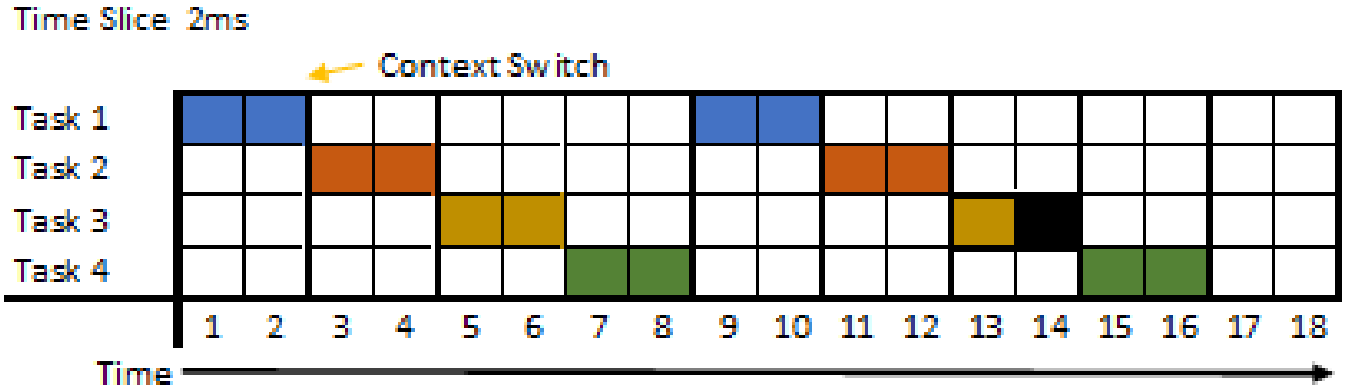
Time Slice with Background Task [TSBG] (continued)



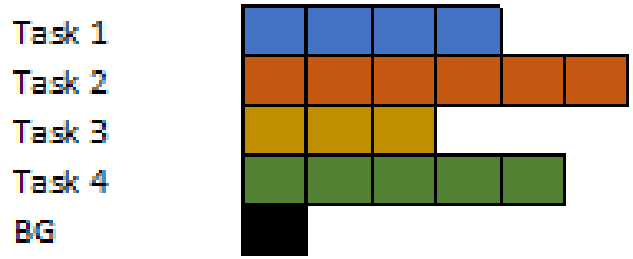


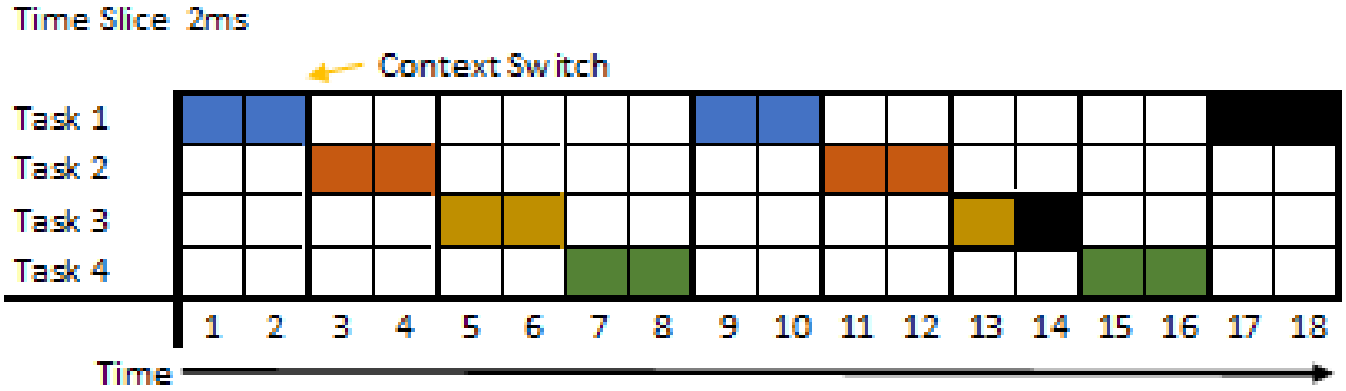
Time Slice with Background Task [TSBG] (continued)





Time Slice with Background Task [TSBG] (continued)





Time Slice with BG Unit Tests

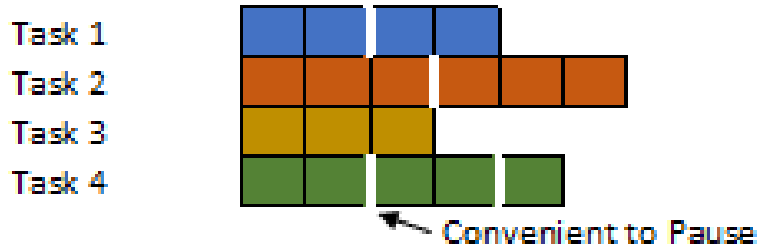
- Check if everything is initialized correctly
- Test that each Task runs for a max time each iteration
- Check that a background Task runs when a Task runs out of things to do

Round Robin [RR]

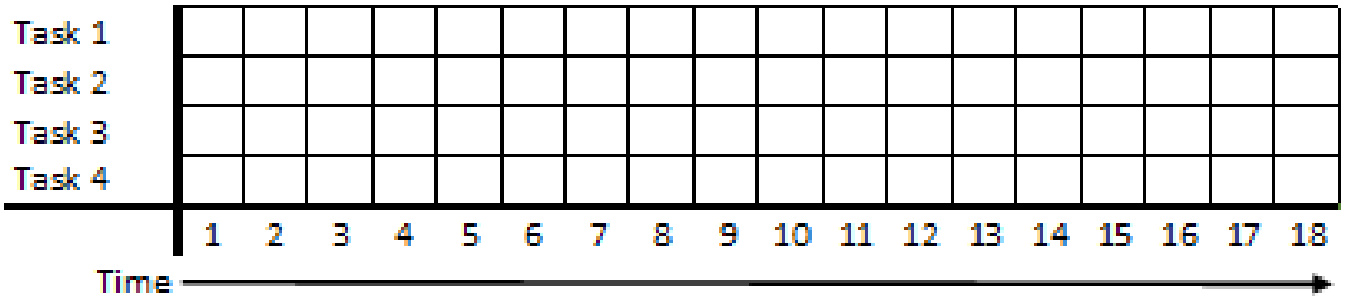
- Cooperative multitasking
- What Is It?
  - Strictly-ordered execution of Task segments
  - Typically the ready queue (FIFO) is a circular queue
  - Each Task gets the resources for a specific amount of time
  - The Task runs until it blocks, finishes, or uses alloted time
- Pros
  - Great for small Tasks
- Cons
  - Highest priority task is not run immediately

Real-life use: Network router with heavy hardware pipelining for execution context pre-fetch

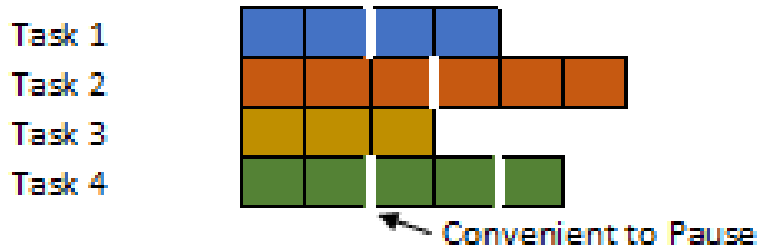
Round Robin [RR] (continued)

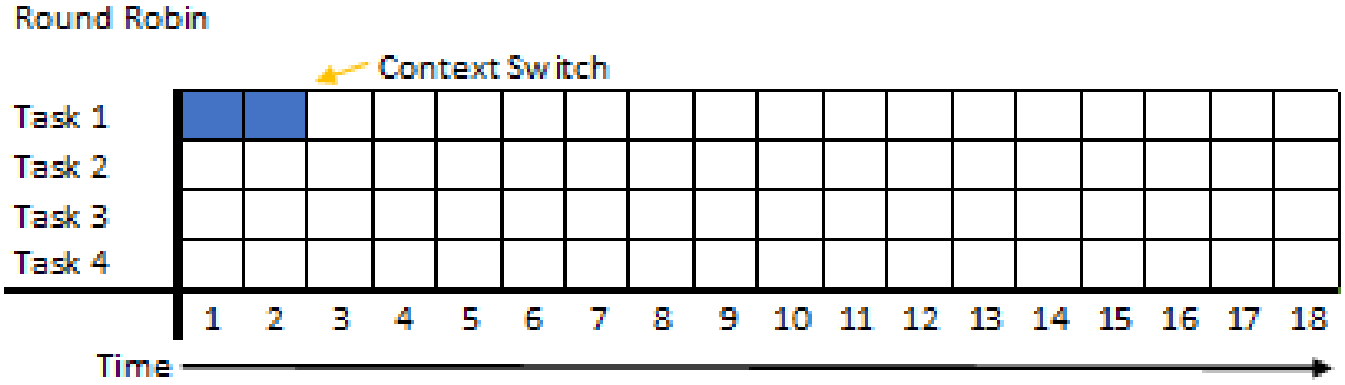


Round Robin

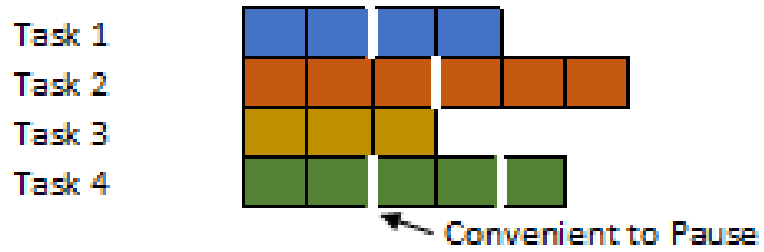


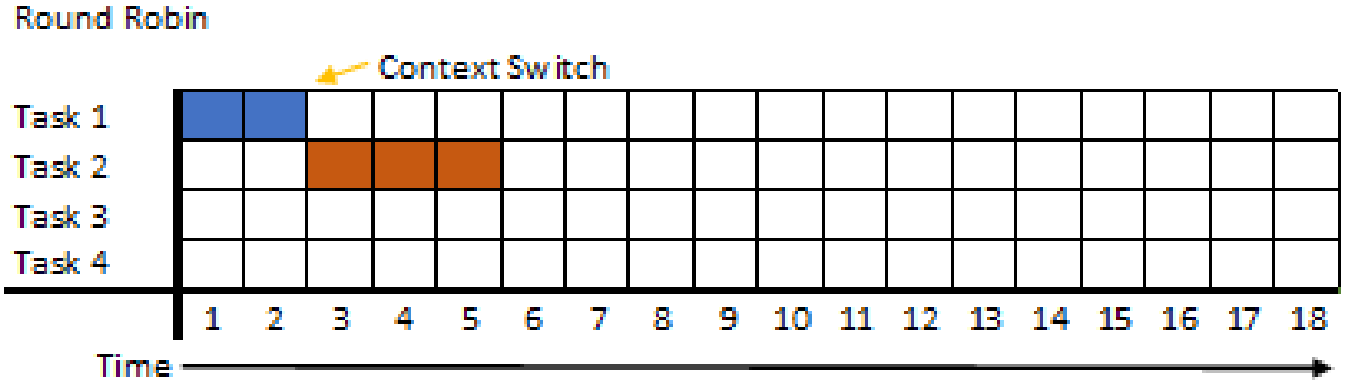
Round Robin [RR] (continued)



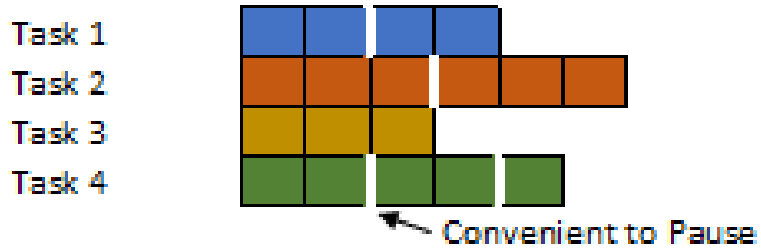


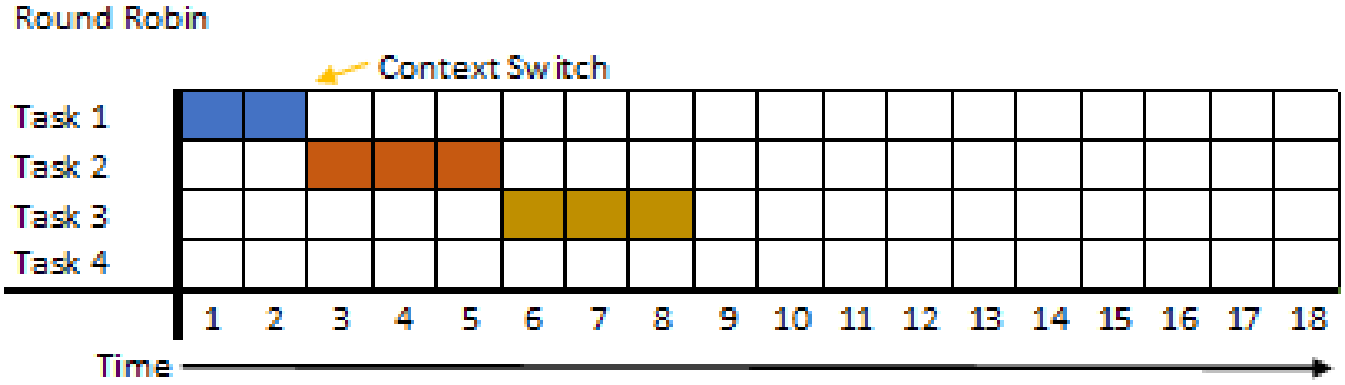
Round Robin [RR] (continued)



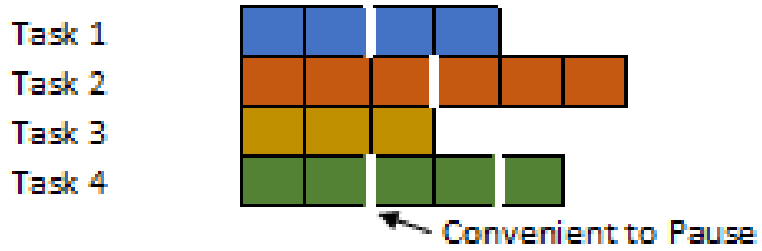


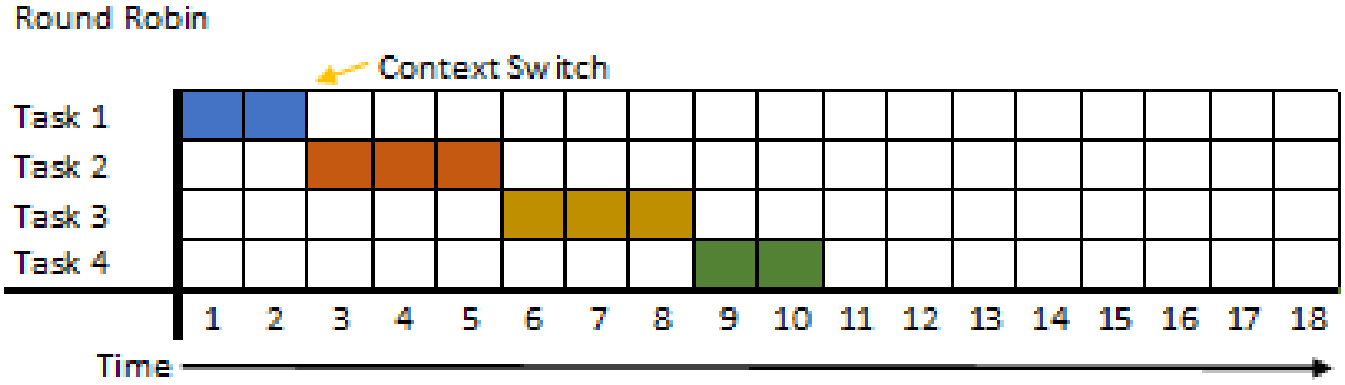
Round Robin [RR] (continued)



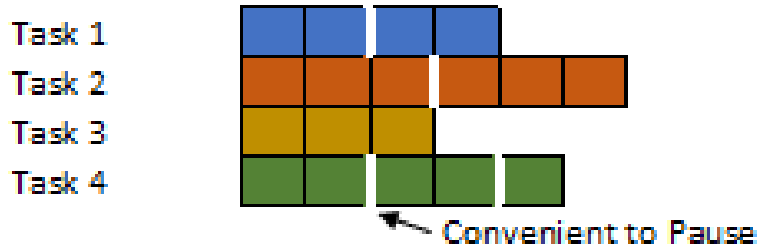


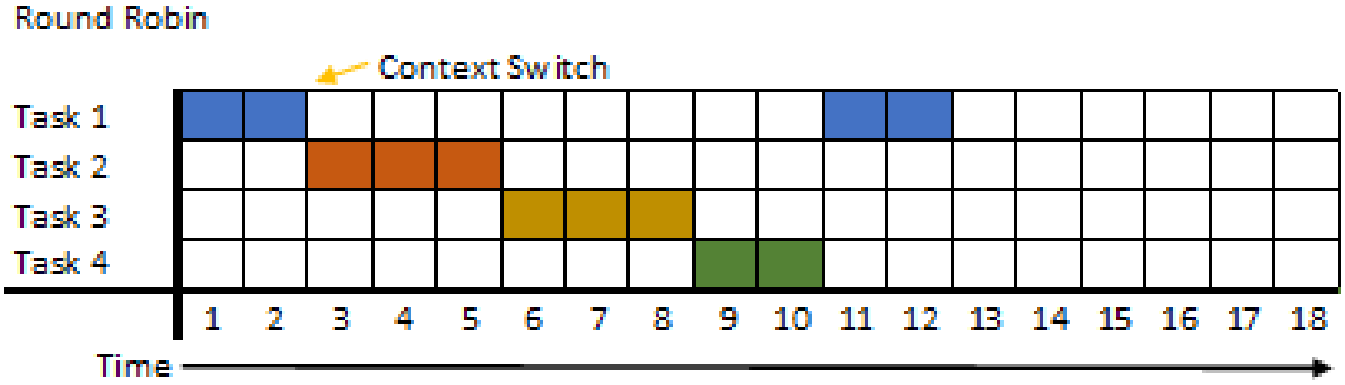
Round Robin [RR] (continued)



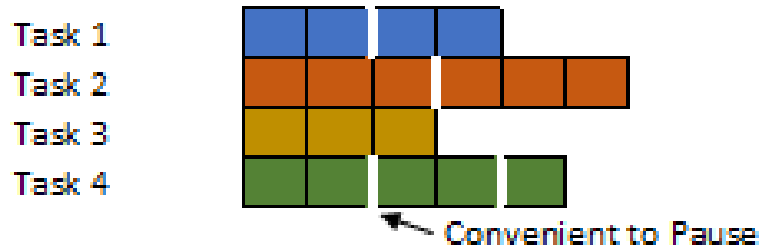


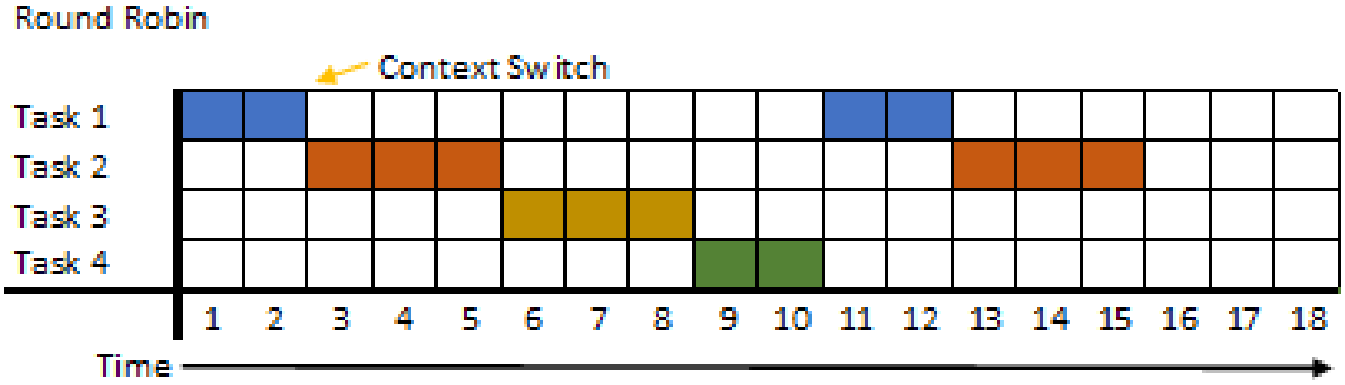
Round Robin [RR] (continued)



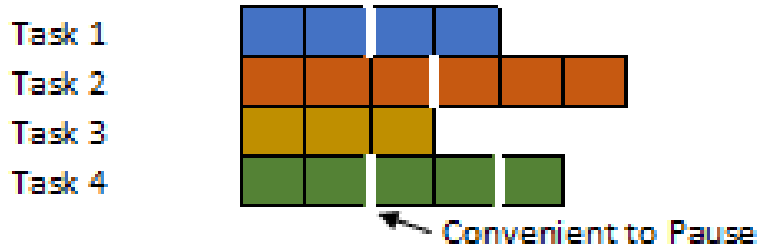


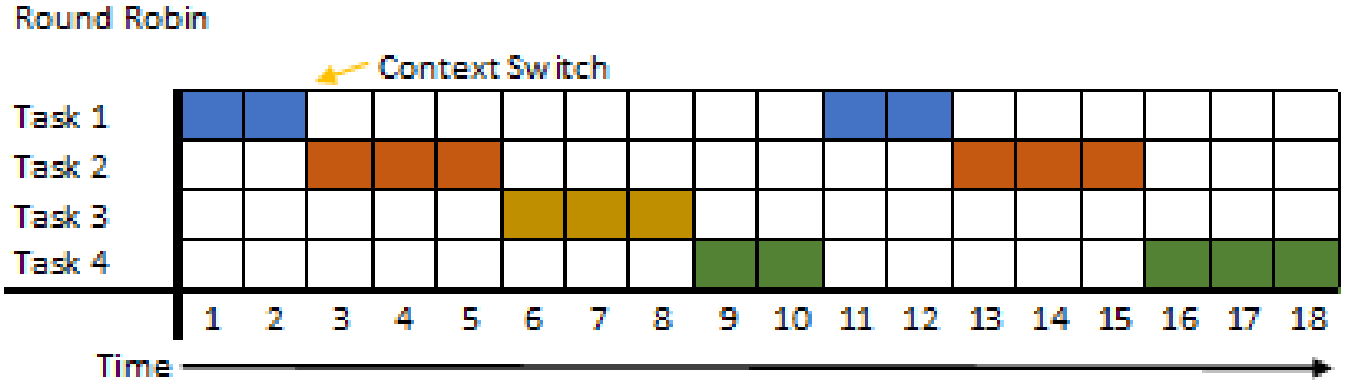
Round Robin [RR] (continued)





Round Robin [RR] (continued)





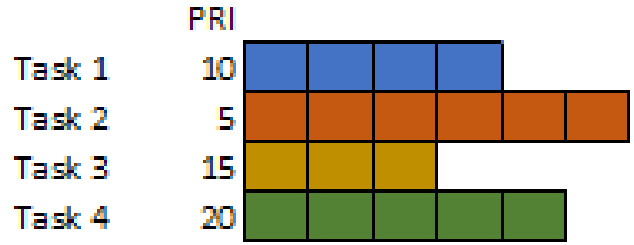
Round Robin Unit Tests

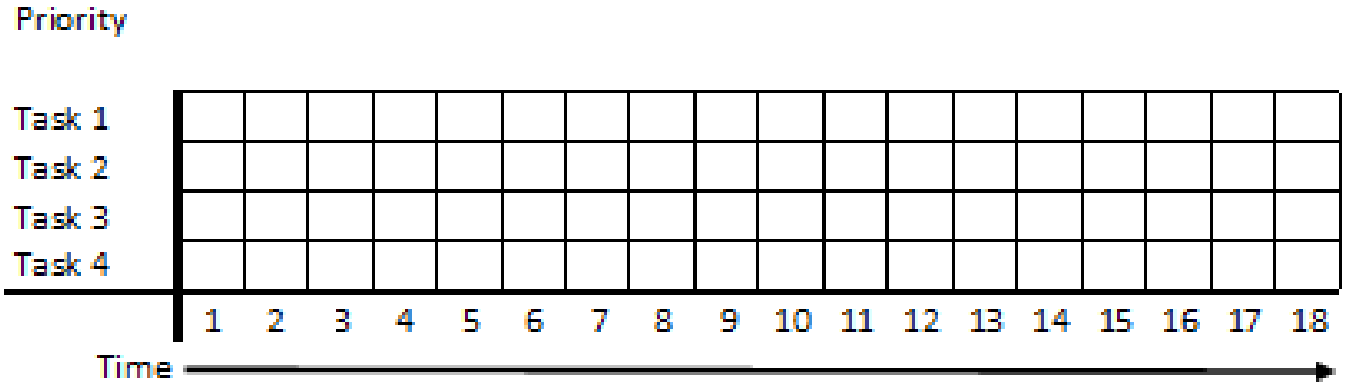
- Check if everything is initialized correctly
- Verify that each Task "pauses" during specific times
- Verify that each Task finished execution
- Verify that each Task didn't hog resources and therefore finish early

Priority

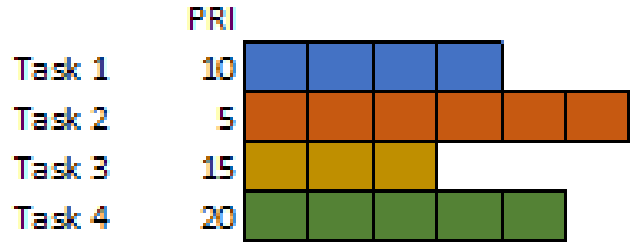
- What Is It?
  - Task is chosen with the highest priority
  - Priority can be based on memory requirements, CPU Time, Owner, etc
  - Can be non-preemptive or preemptive
  - Note: priorities can change during execution
- Pros
  - Highest priority jobs are run
- Cons
  - Low priority jobs can potentially wait forever
  - Overhead due to context-switching

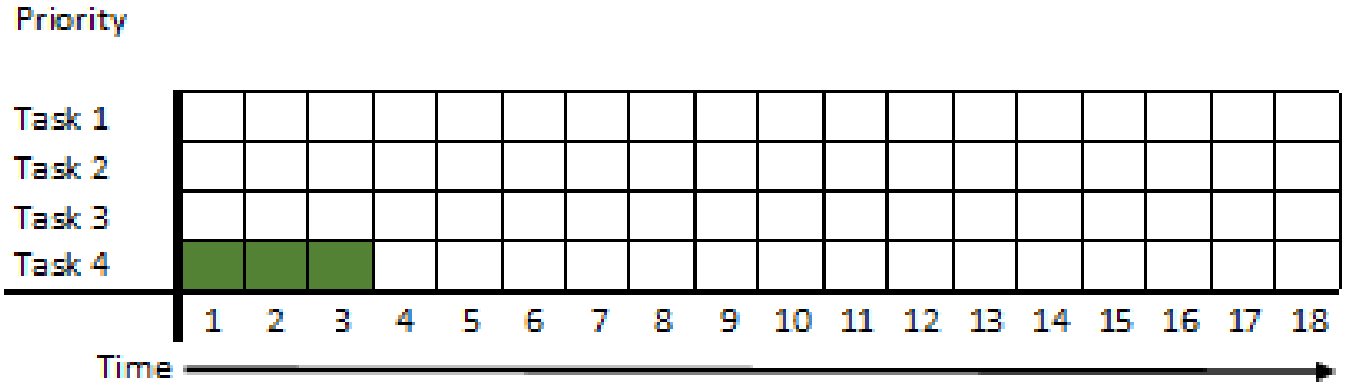
Priority (continued)



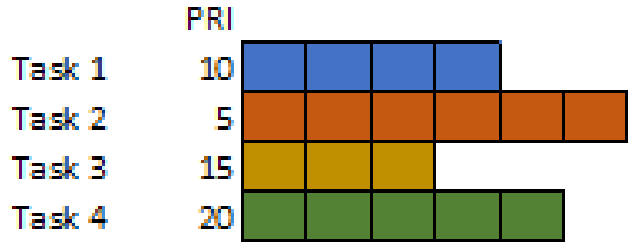


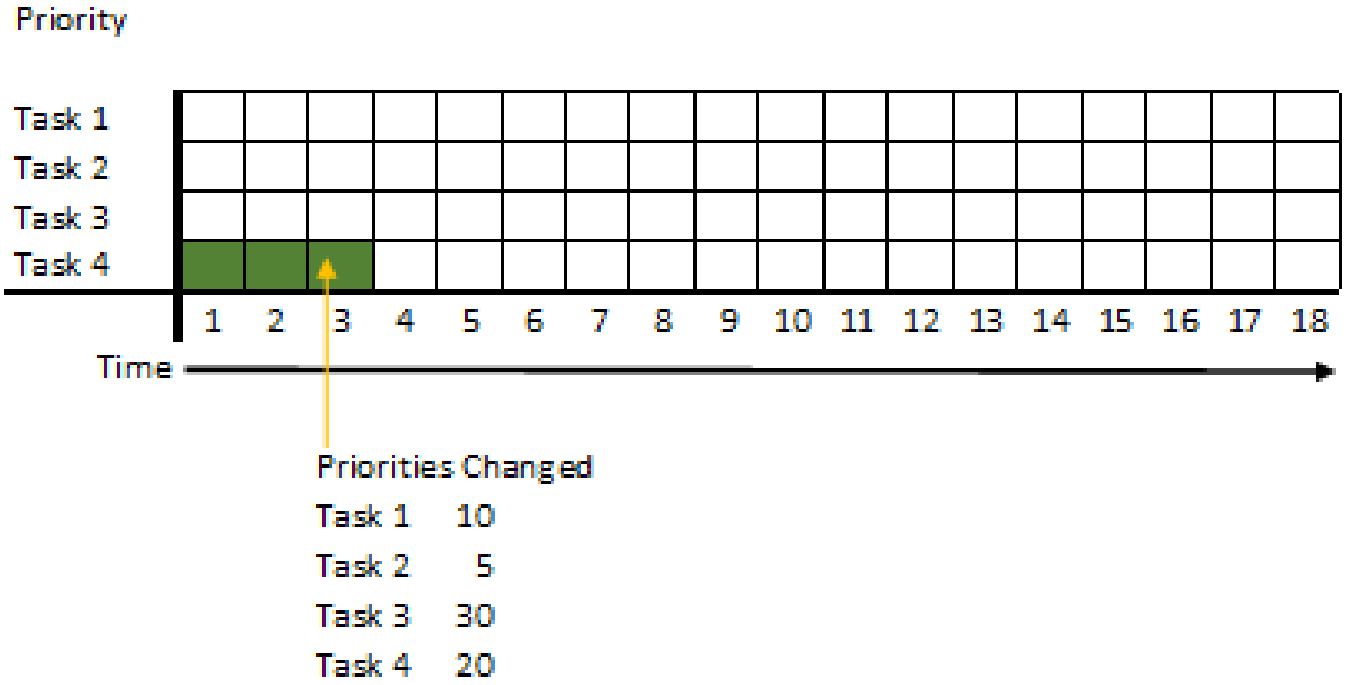
Priority (continued)



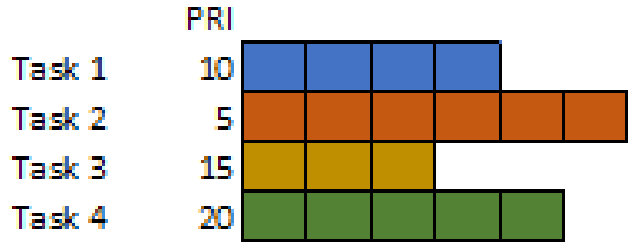


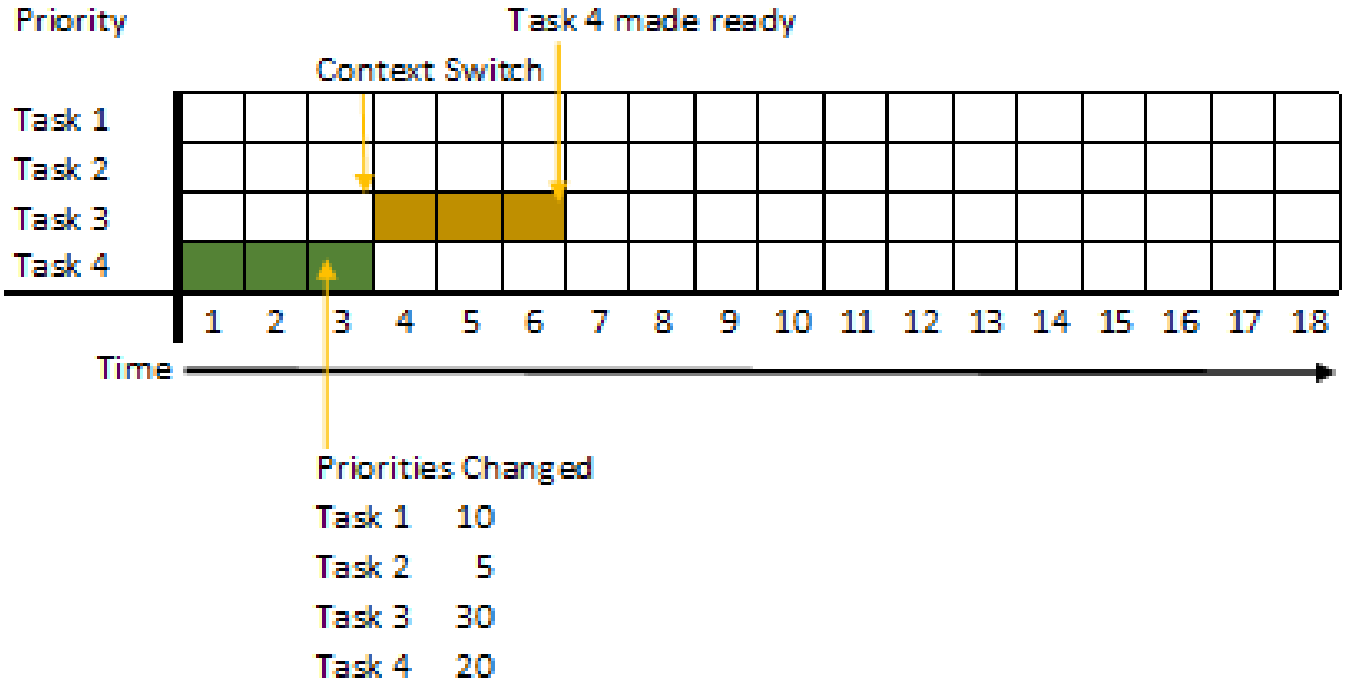
Priority (continued)



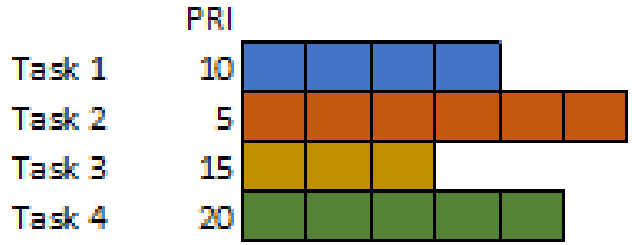


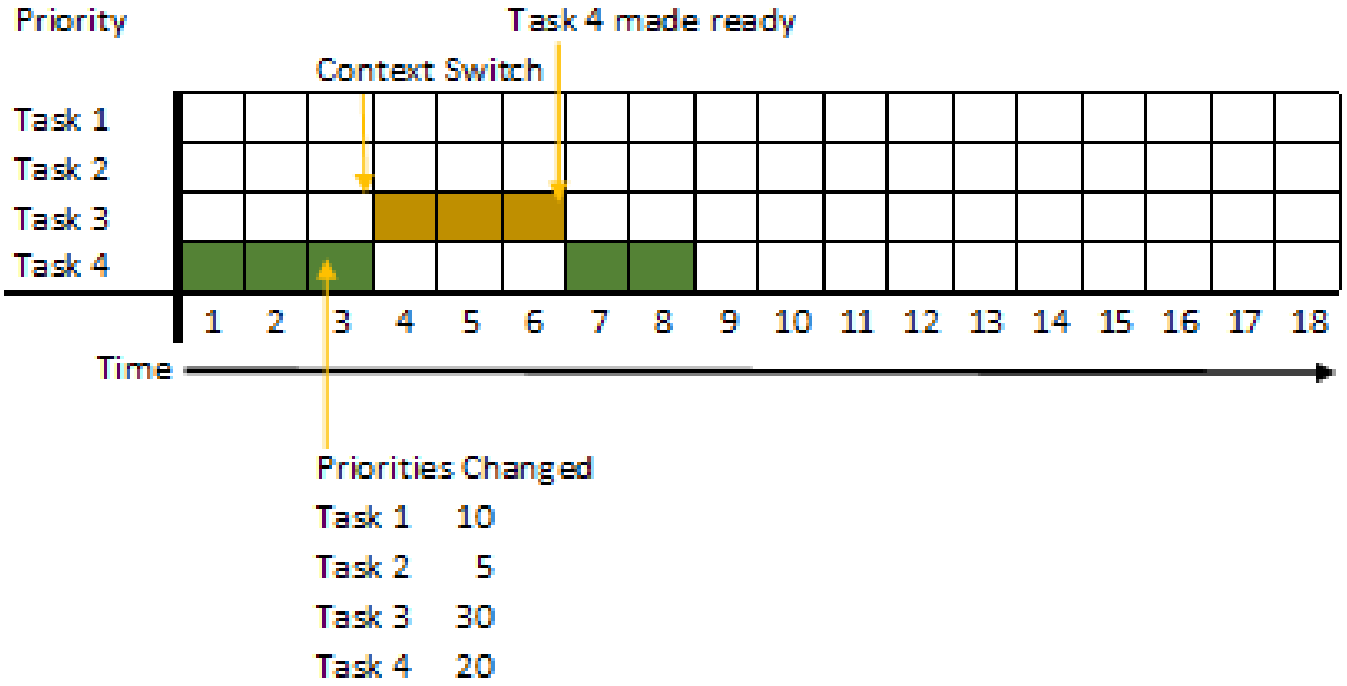
Priority (continued)



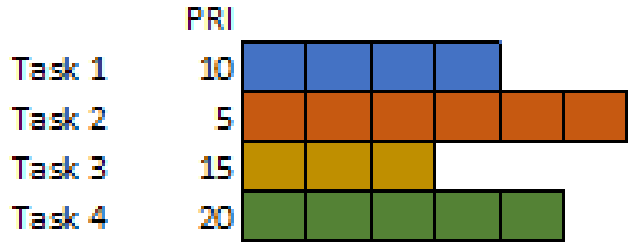


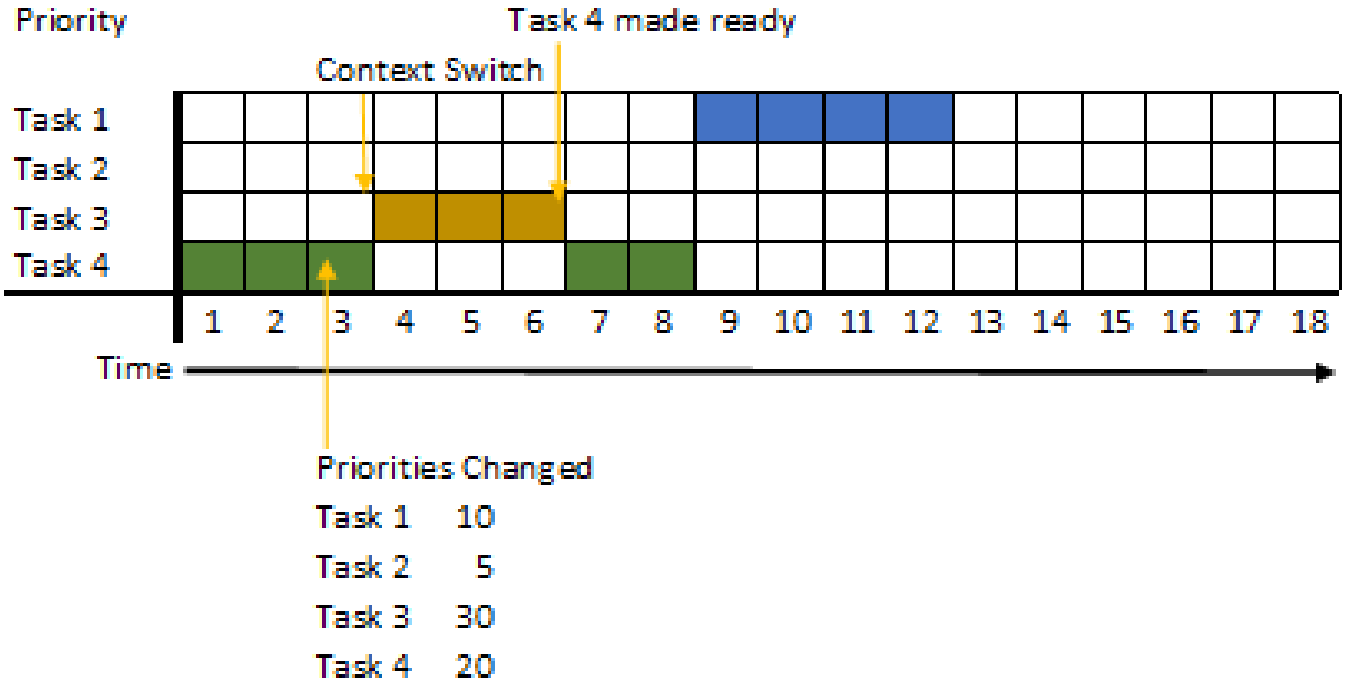
Priority (continued)



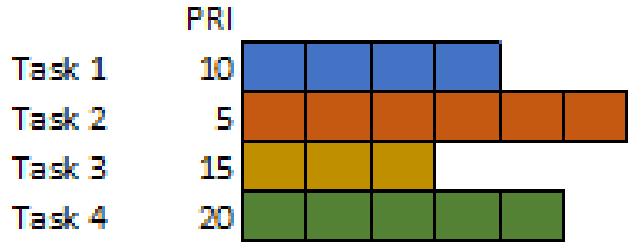


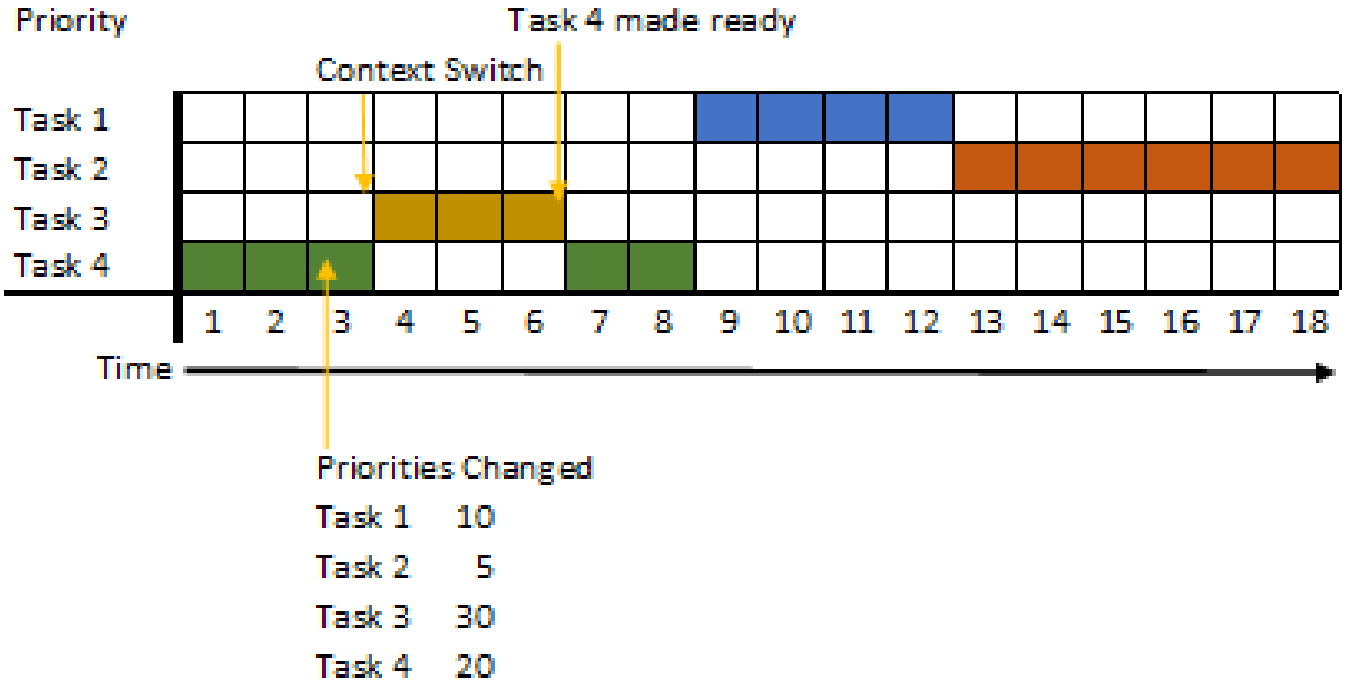
Priority (continued)





Priority (continued)





Priority Unit Tests

- Check if everything is initialized correctly
- Verify that the highest priority Task is always picked next
- Verify all Tasks finished execution
- Verify that new higher priority Tasks are executed in the right order

Scheduling

- Which Scheduling algorithm should you use?

Scheduling

- Which Scheduling algorithm should you use?
- It Depends.