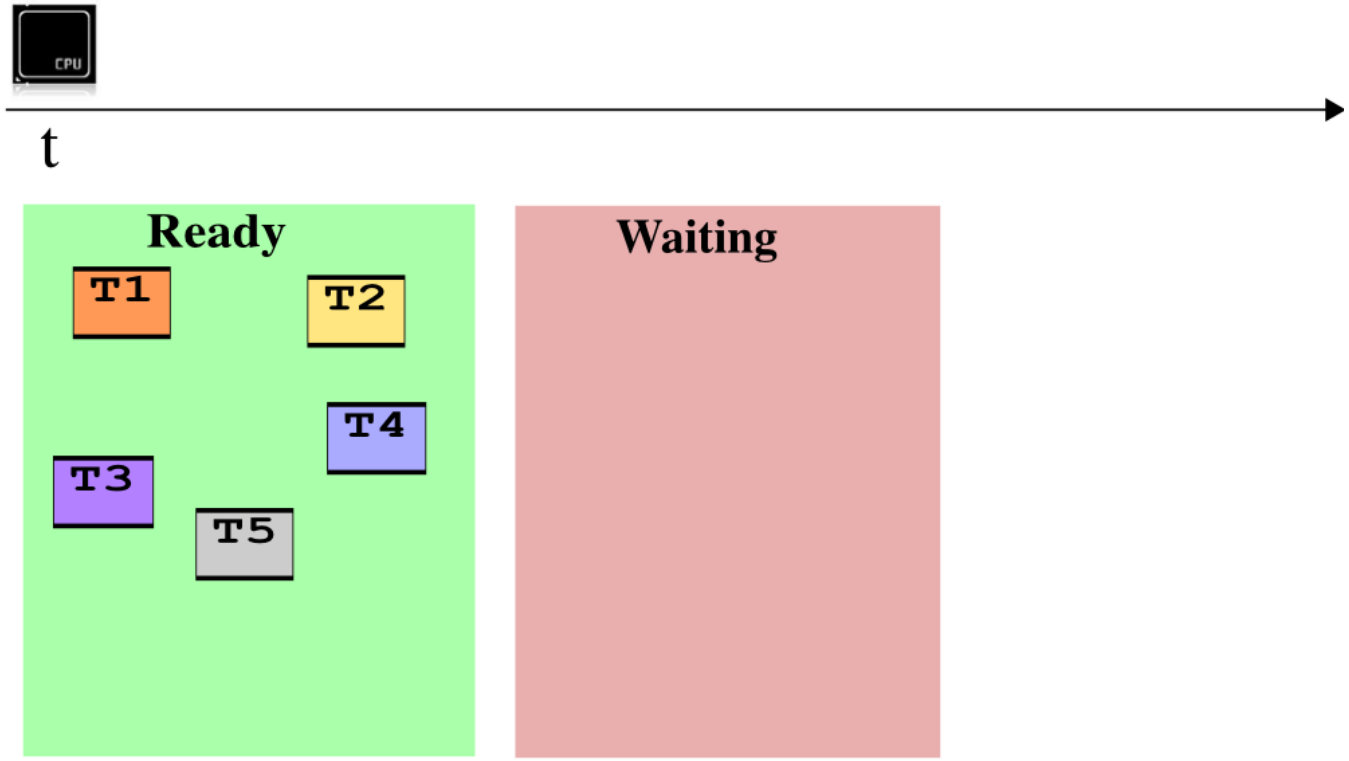
CS 1217

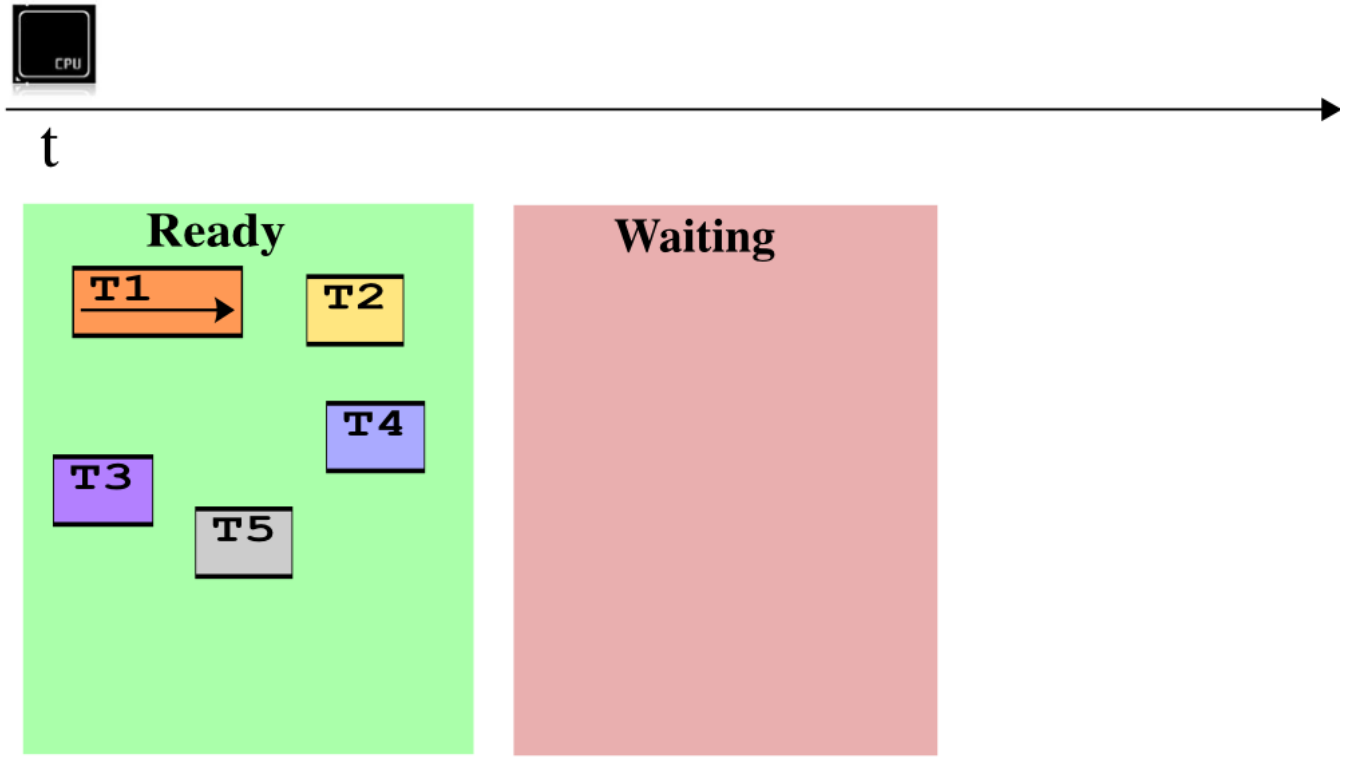
Lecture 11 – Scheduling Wrap; MLFQ; Lottery; Linux Trivia

Blast from the Past

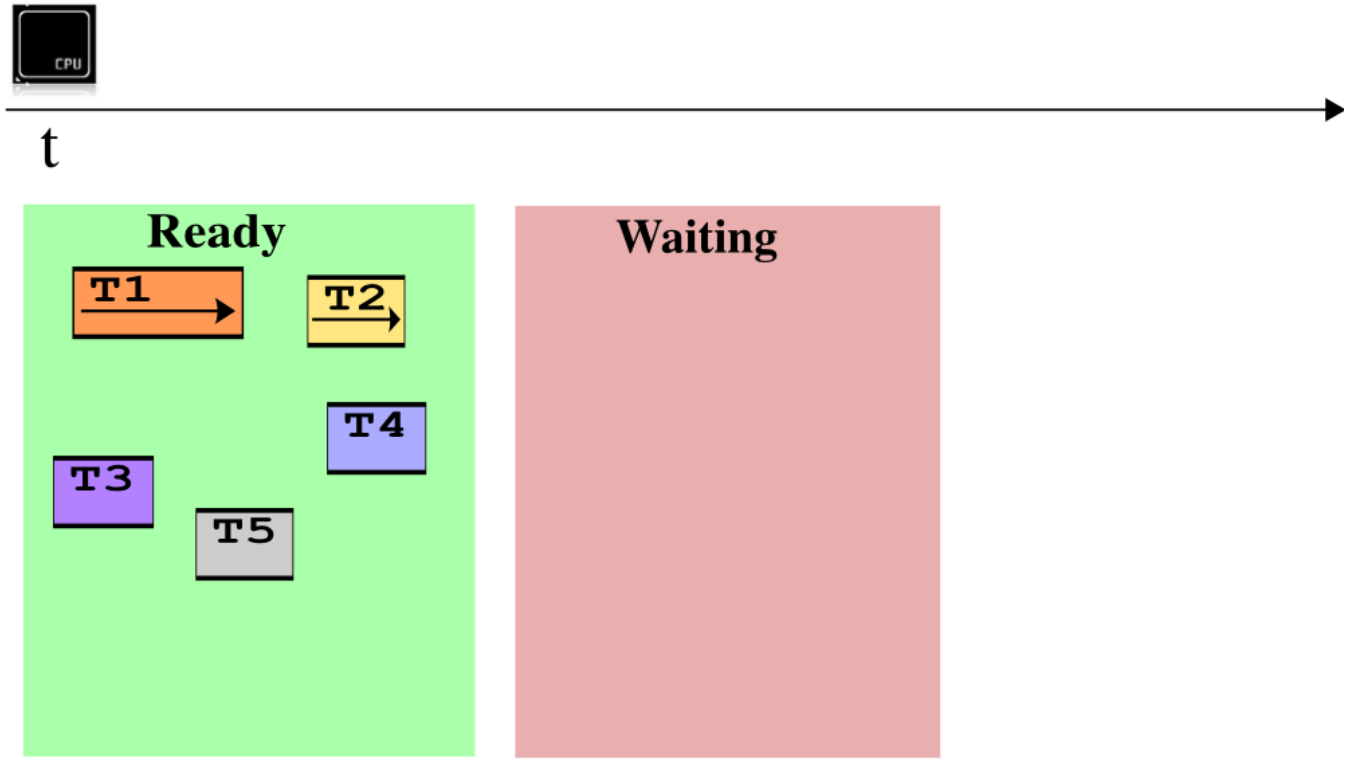
Oracle Scheduling



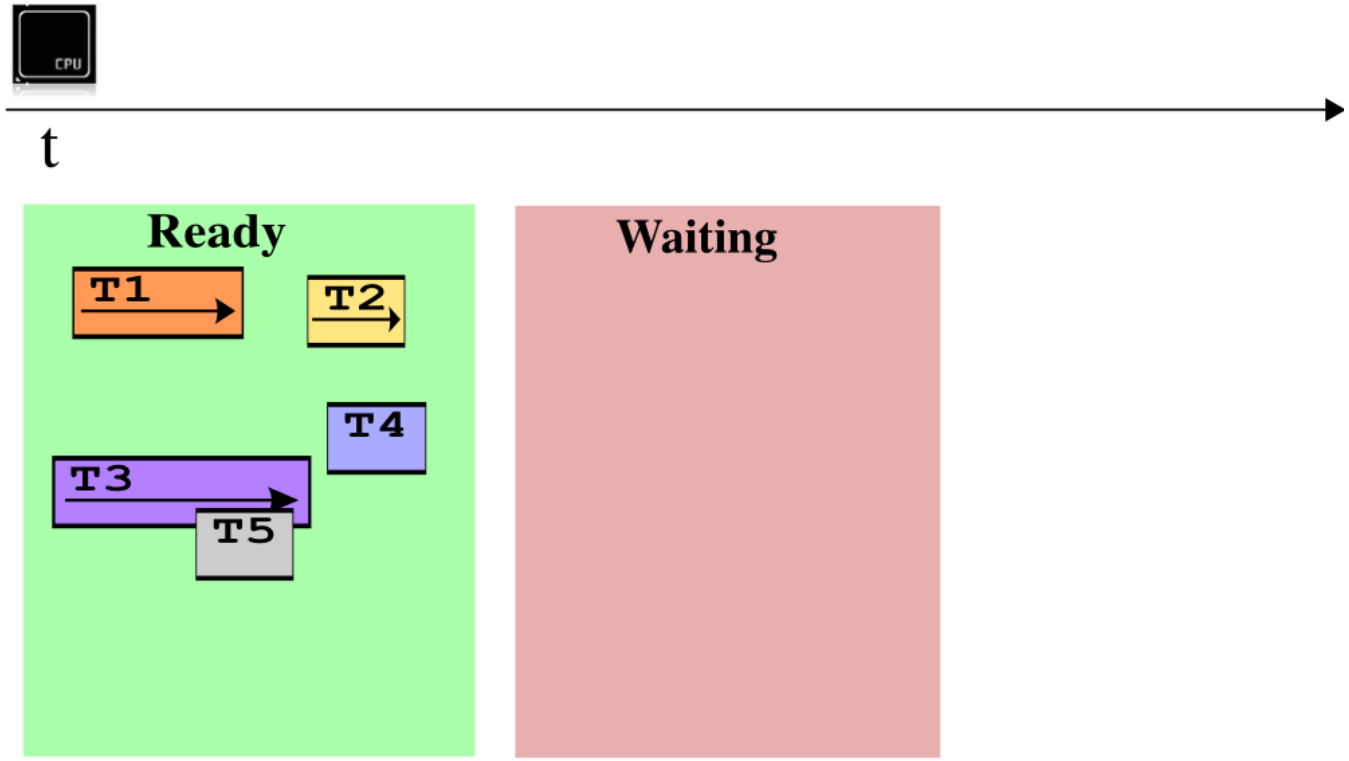
Oracle Scheduling



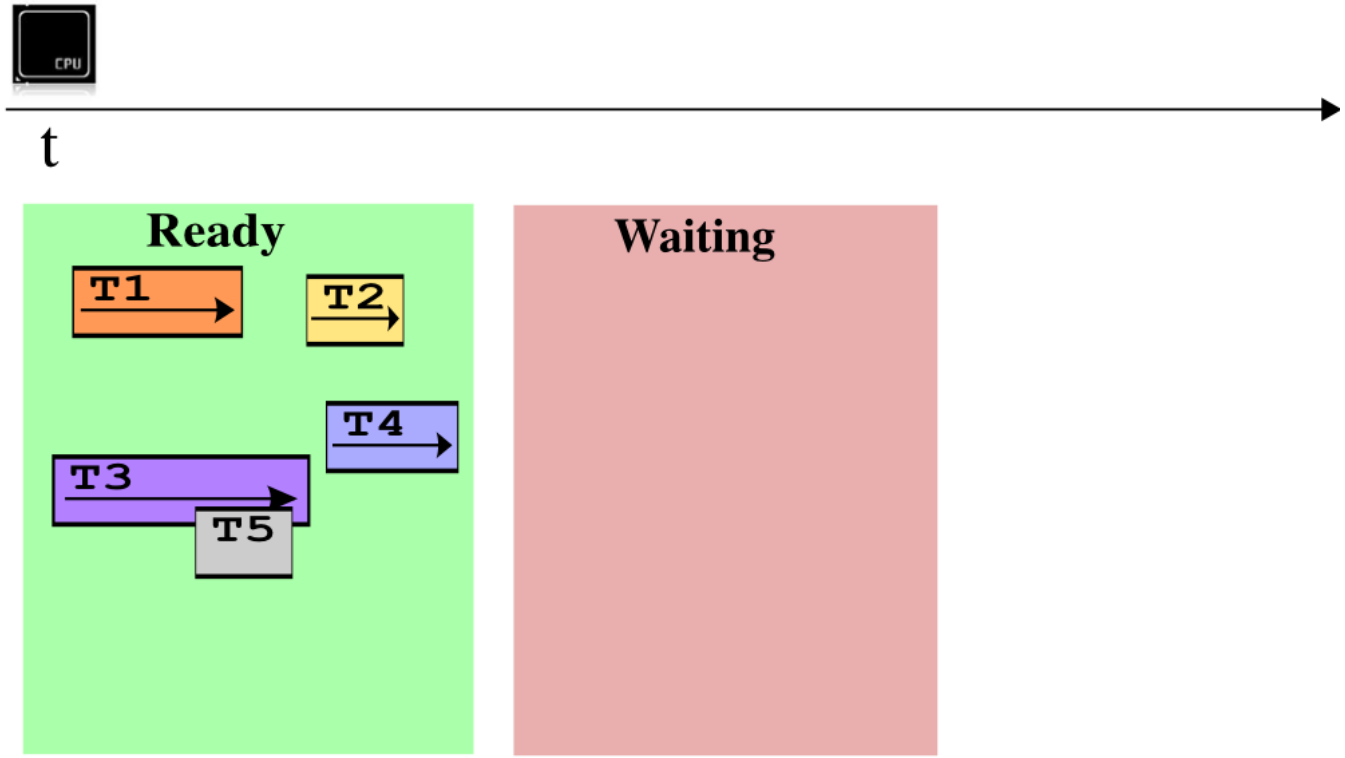
Oracle Scheduling



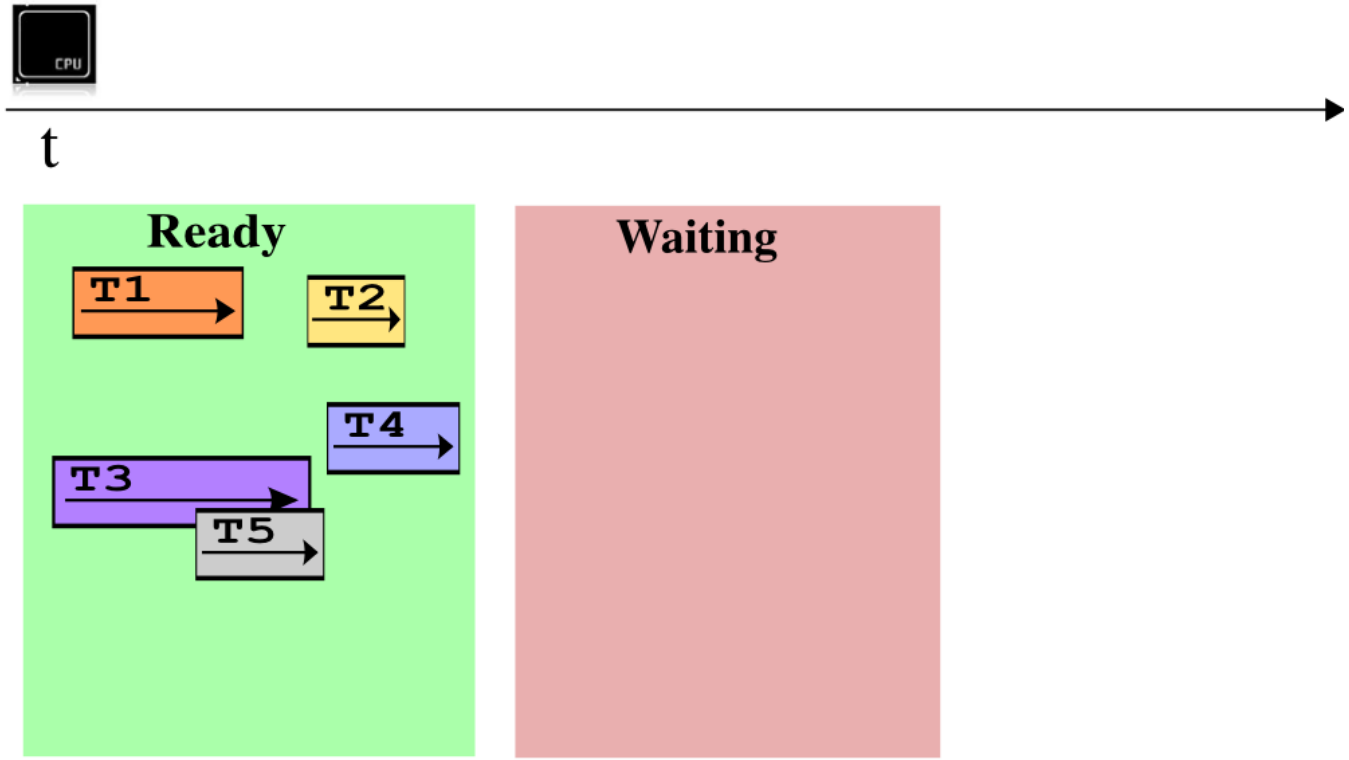
Oracle Scheduling



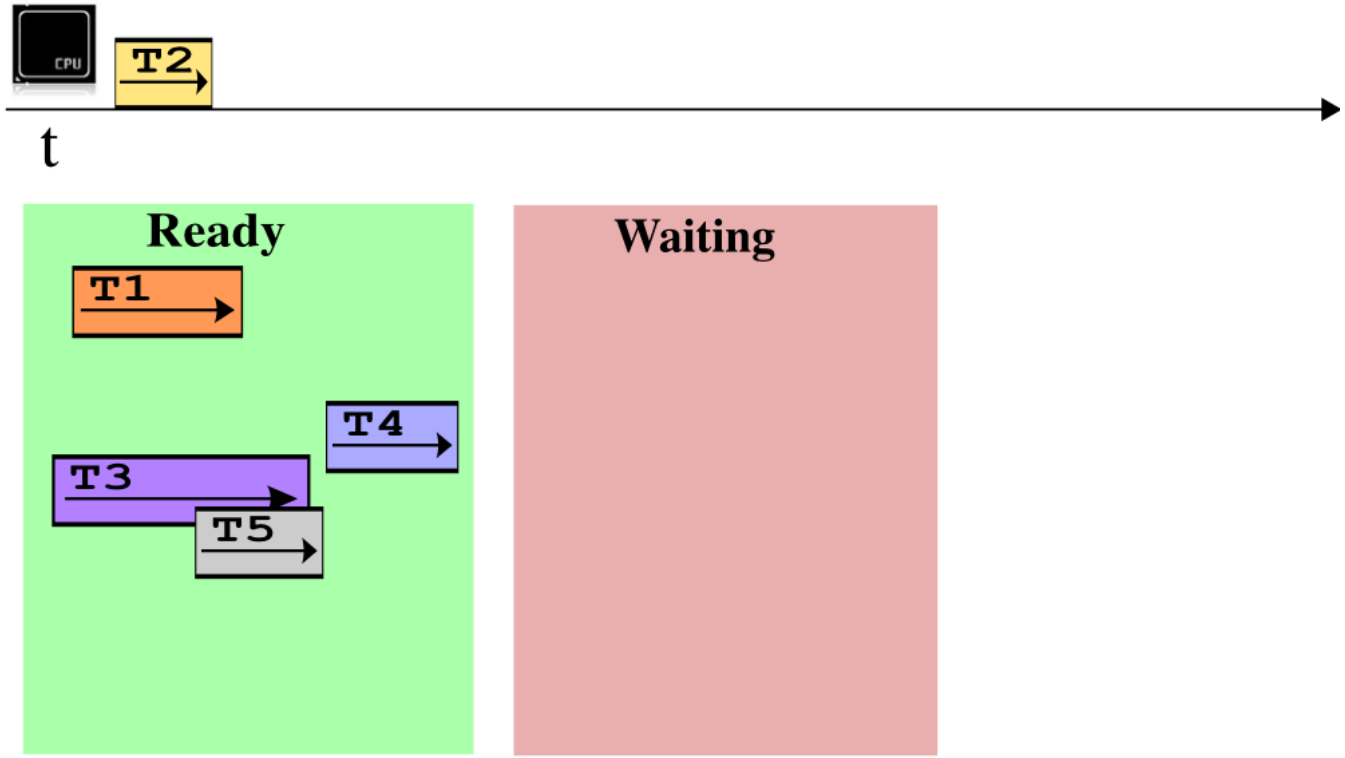
Oracle Scheduling



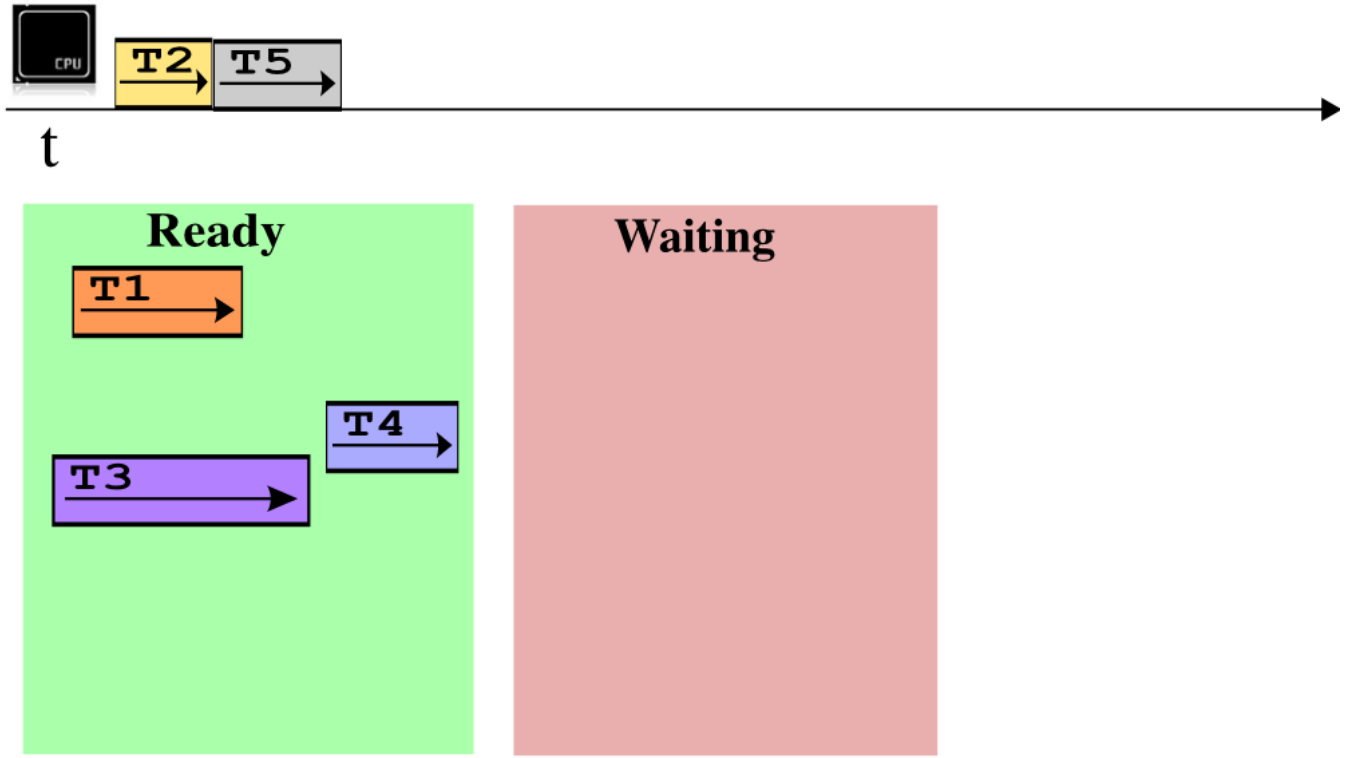
Oracle Scheduling



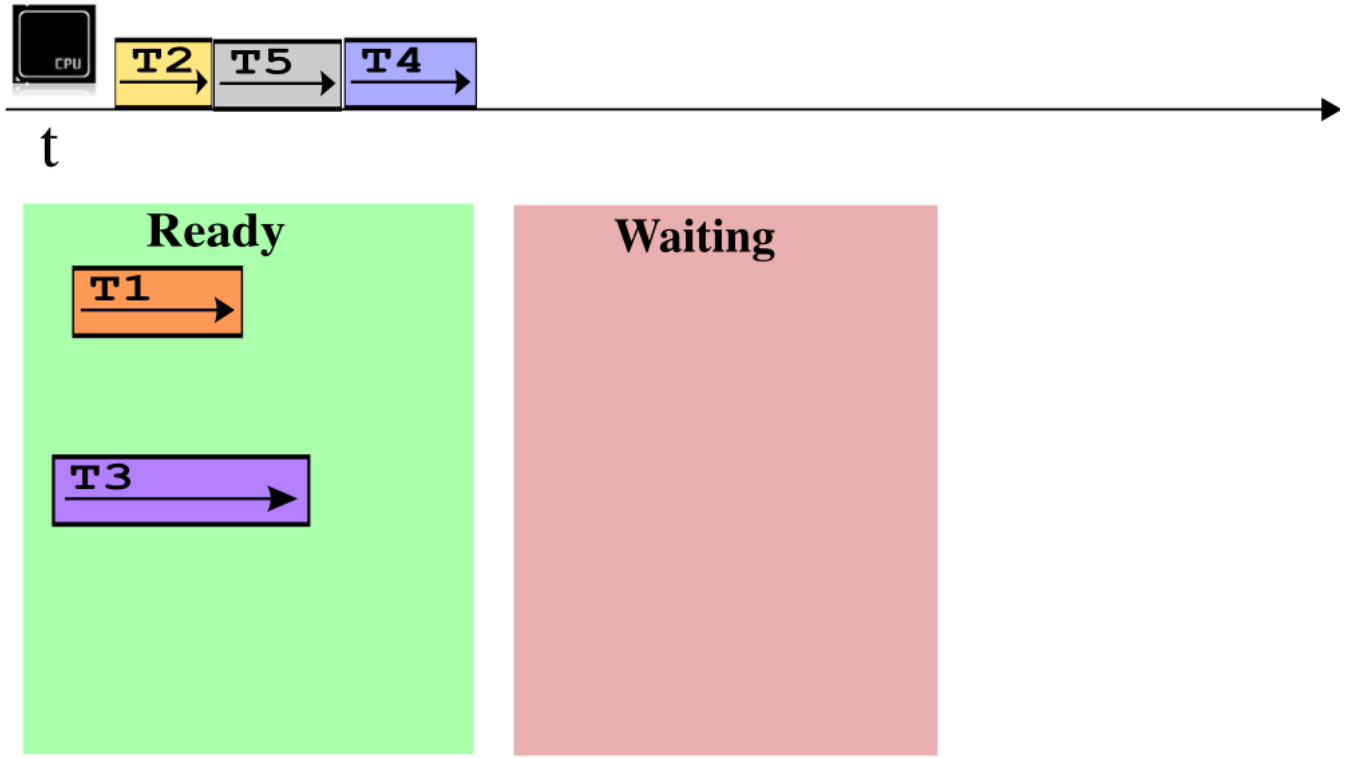
Oracle Scheduling



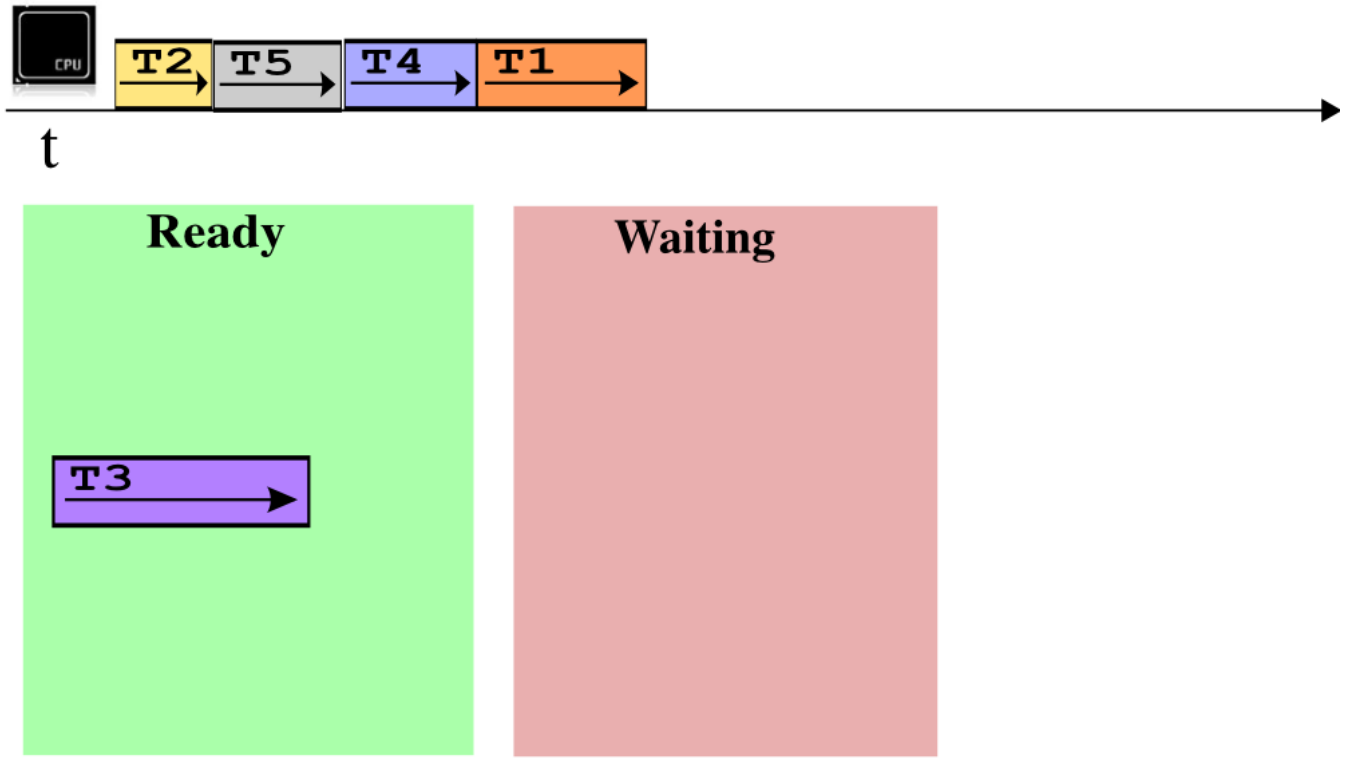
Oracle Scheduling



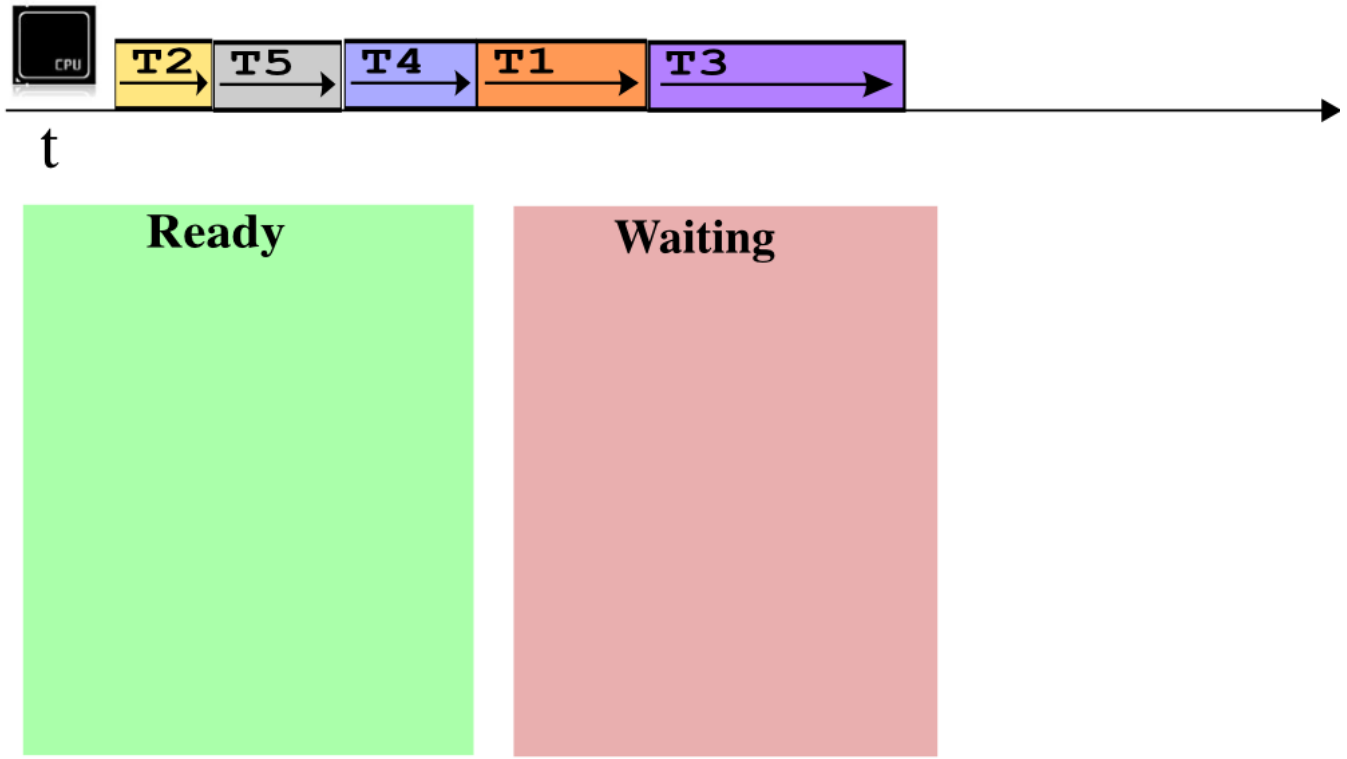
Oracle Scheduling



Oracle Scheduling



Oracle Scheduling



Oracle Schedulers

- Hard to **know** the future; at best you can make a ***prediction*** with some degree of confidence

- Hence, we use the past to predict the future

More Discussion

- Metrics to consider while scheduling: average waiting time
  - Should this be maximized or minimized?

- Longest Job First

- Shortest Job First

Things you'd want to know about a Process

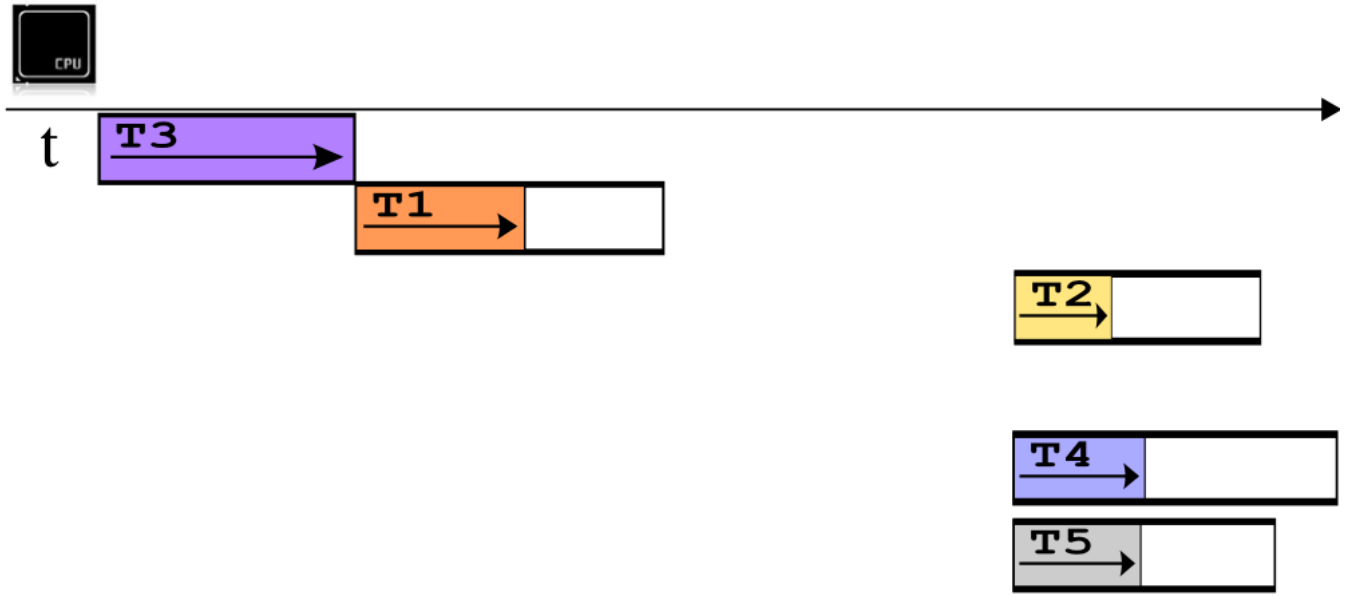
- What might we like to know about the process we are about to execute?
- **How long** is it going to use the CPU!
- Will it **block** or **yield**?
- How **long** will it **wait**?

- And more (list is incomplete)

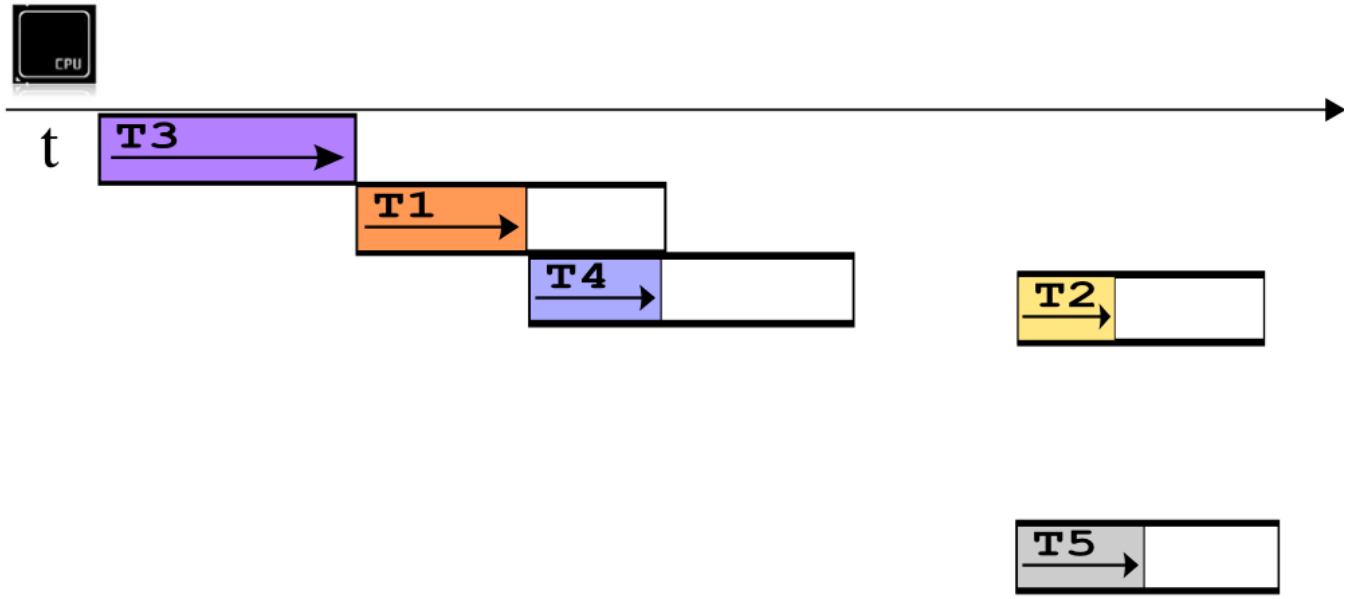
Longest Job First

Longest Job First

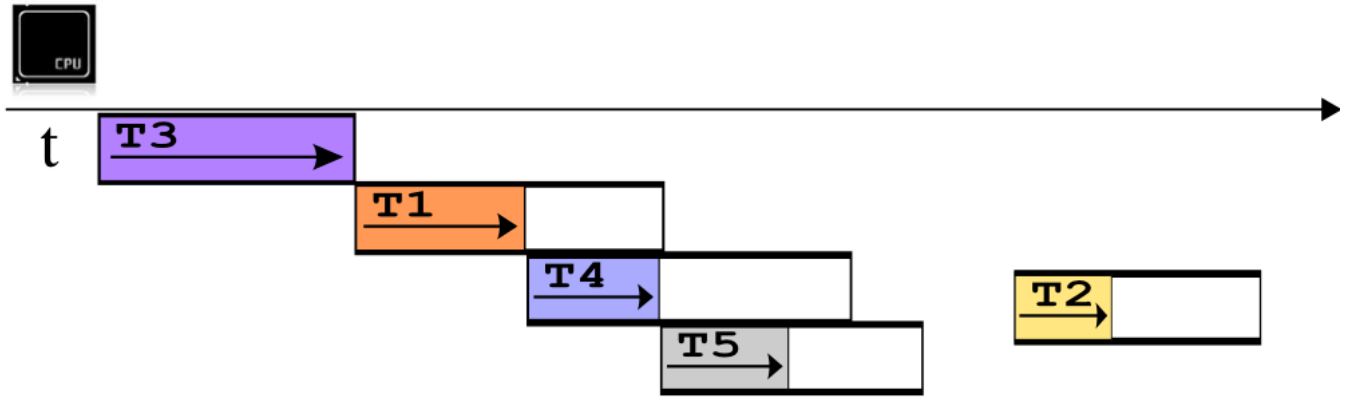
Longest Job First



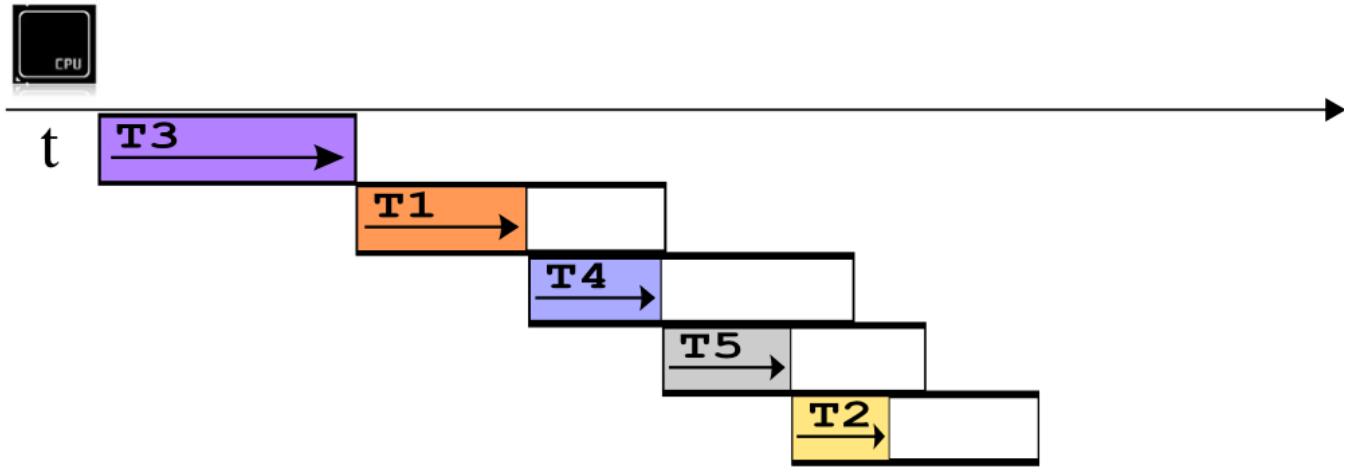
Longest Job First



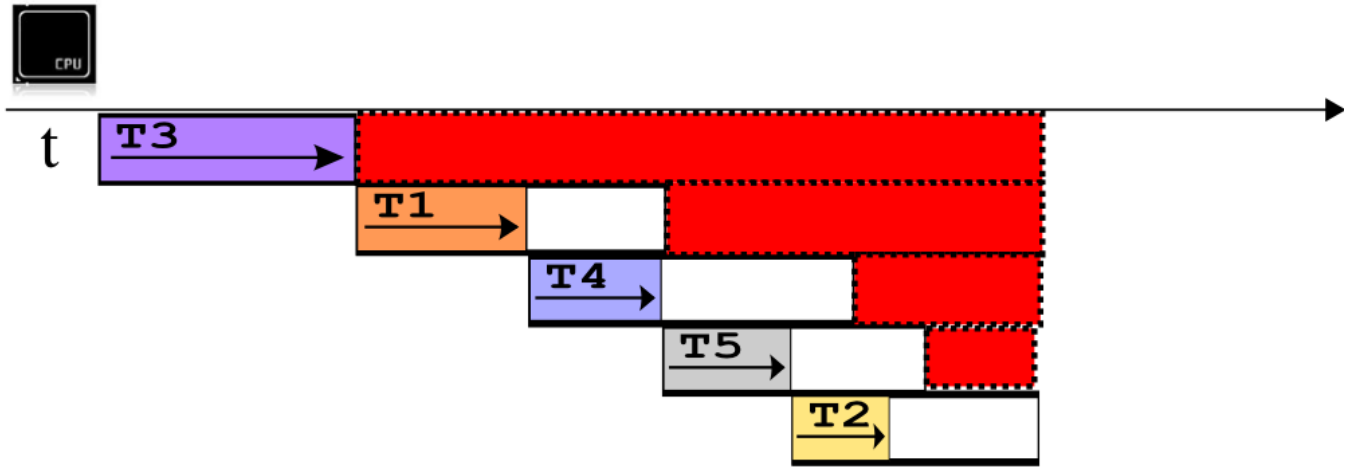
Longest Job First



Longest Job First



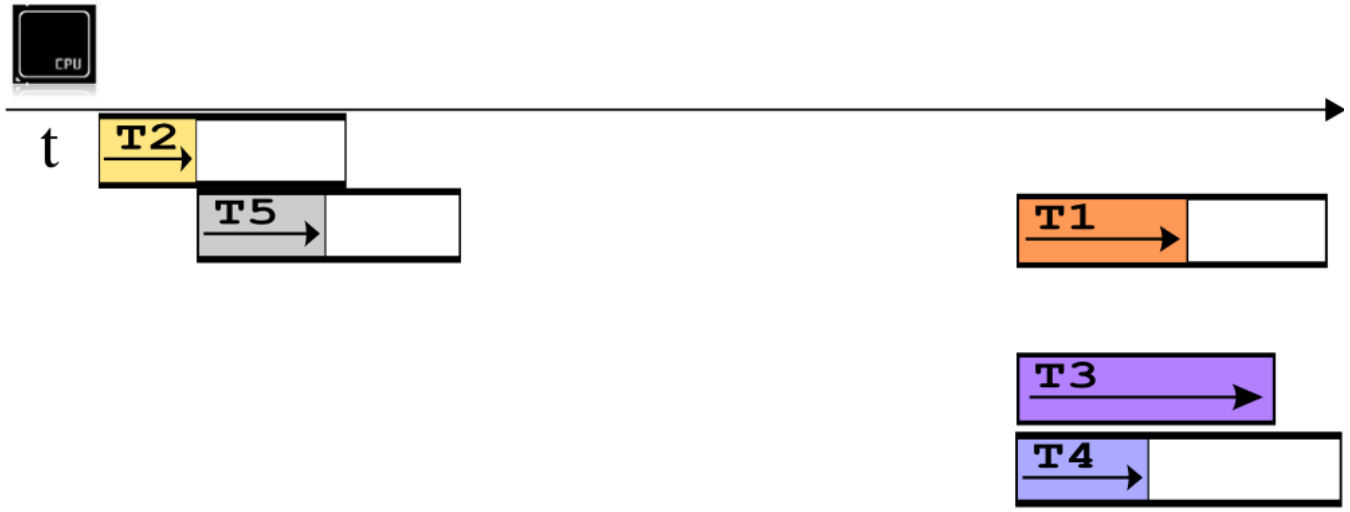
Longest Job First



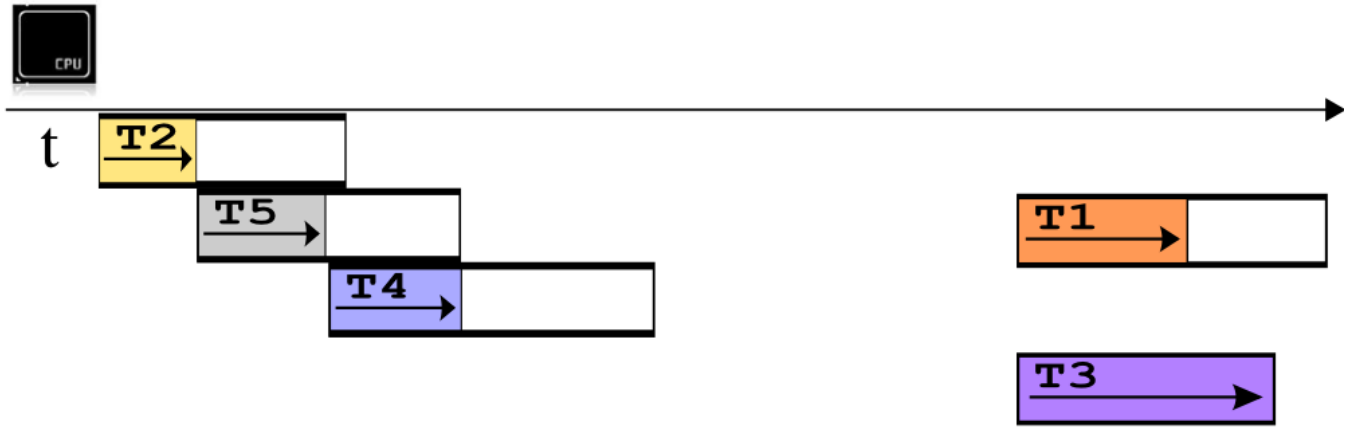
Shortest Job First

Shortest Job First

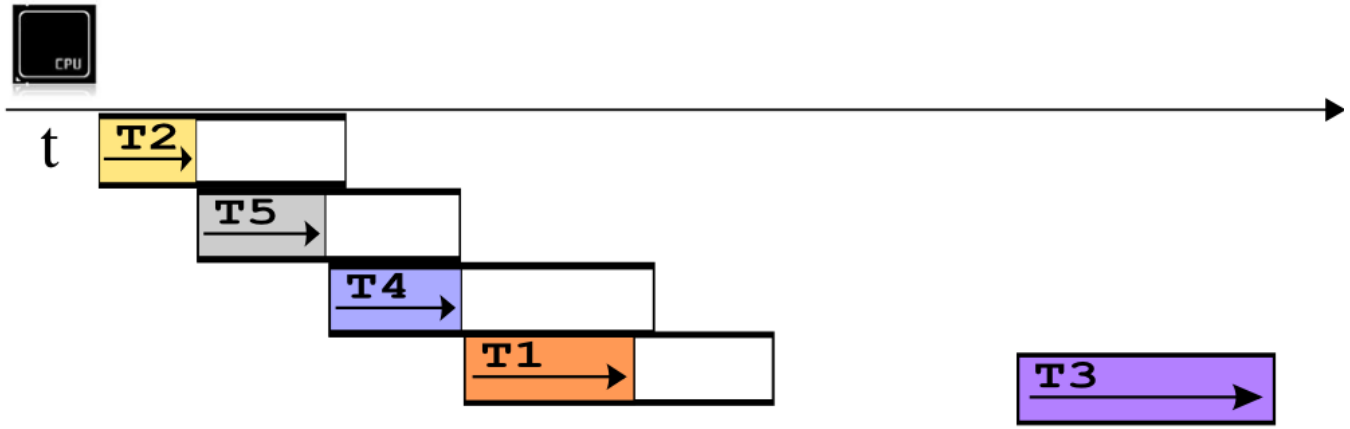
Shortest Job First



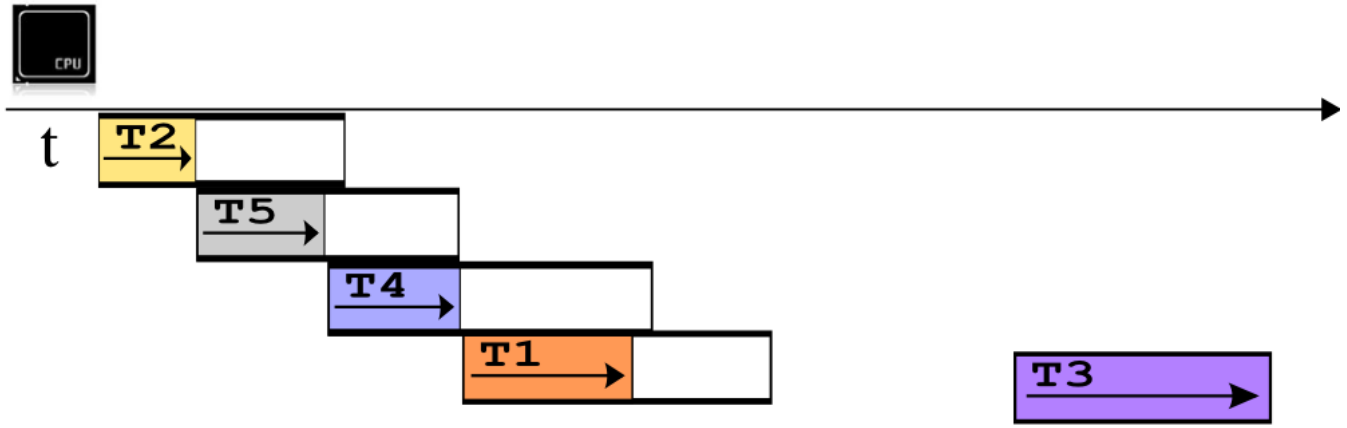
Shortest Job First



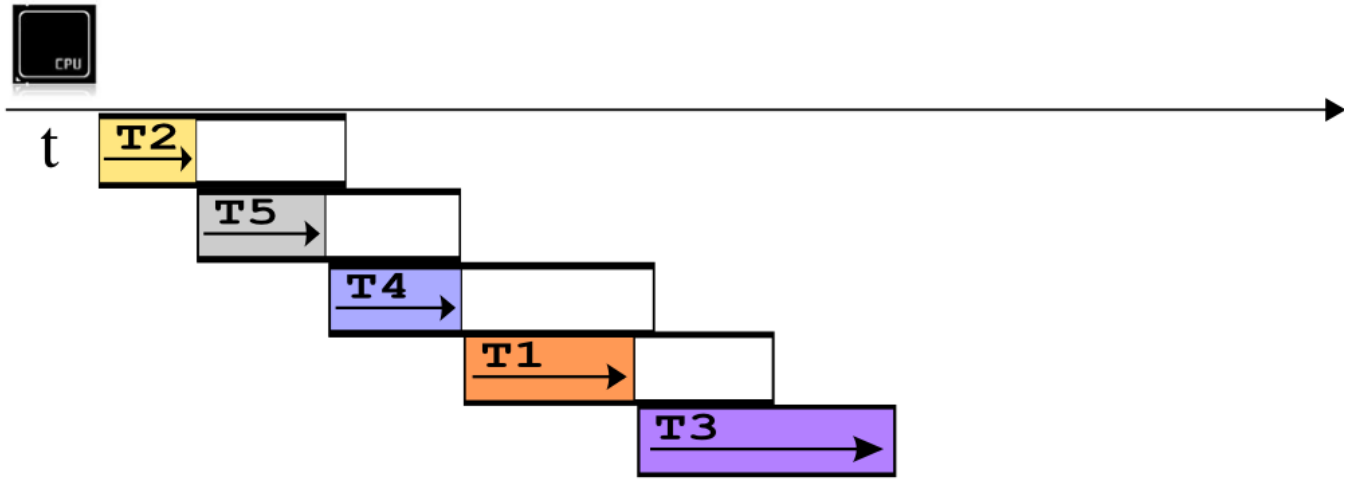
Shortest Job First



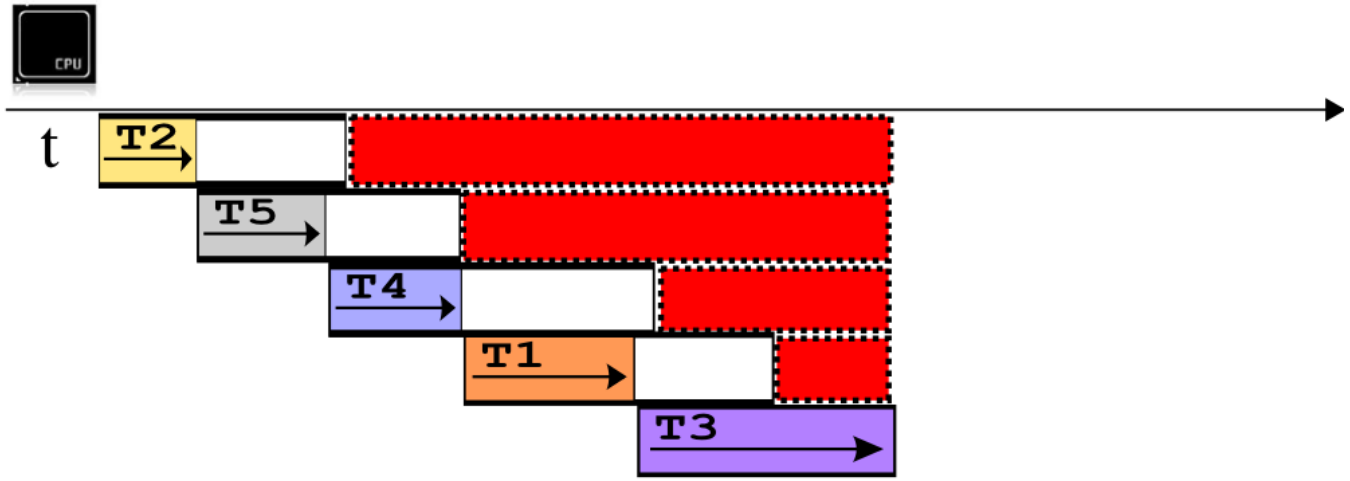
Shortest Job First



Shortest Job First



Shortest Job First



Shortest Job First

- Why would we use this algorithm?
- **Minimizes waiting time!**
- More generally, why would we prefer threads that give up the CPU before their time quantum ends?
- They are probably waiting for **something else**, which can be done in parallel with CPU use.
  - Better utilization of **overall system resources**

Recap

- Scheduling quantum : what?
- When do you make scheduling decisions?
- What needs to be taken into account?

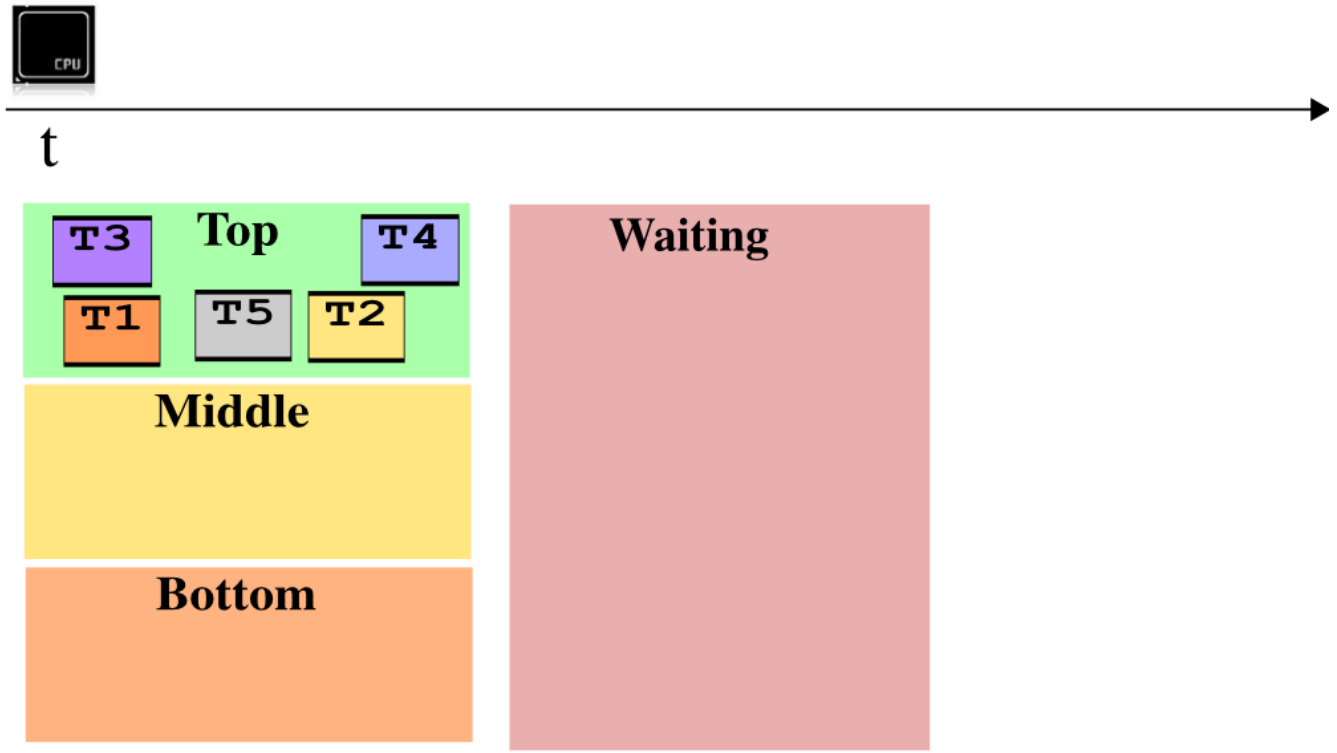
Multi Level Feedback Queue (MLFQ)

- Fernando José (“Corby”) Corbató : Turing Award (1990) for MLFQ (1962)
- Trivia question : How many Turing awards to researchers in India?

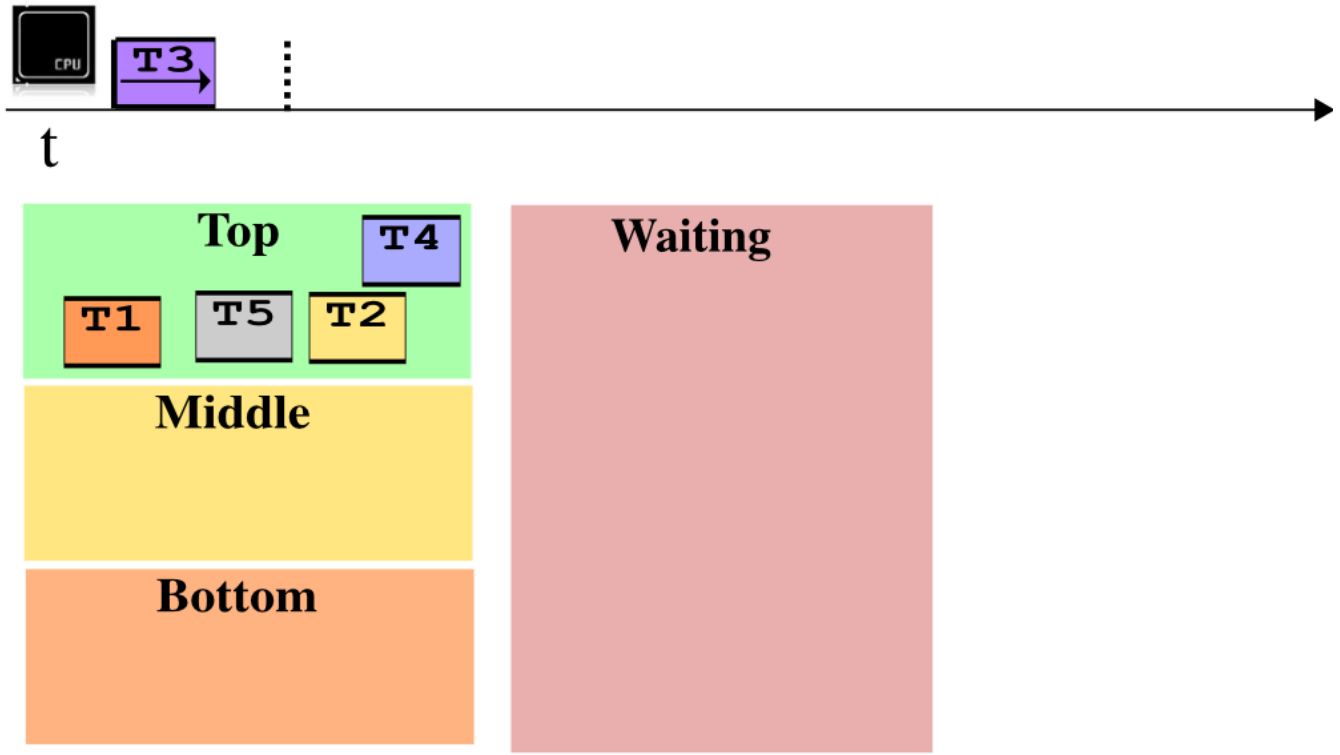
MLFQ

- Choose a **scheduling quantum**.
- Establish some number of queues; each represents a **level**.
- Processes from the highest-level queues are chosen first
- Then:
  - Choose and run a process from the highest-level **non-empty** queue.
  - If the process **blocks** or **yields**, promote it to a higher level queue.
  - If the process must be **preempted** at the end of a quantum, demote it to a lower level queue.

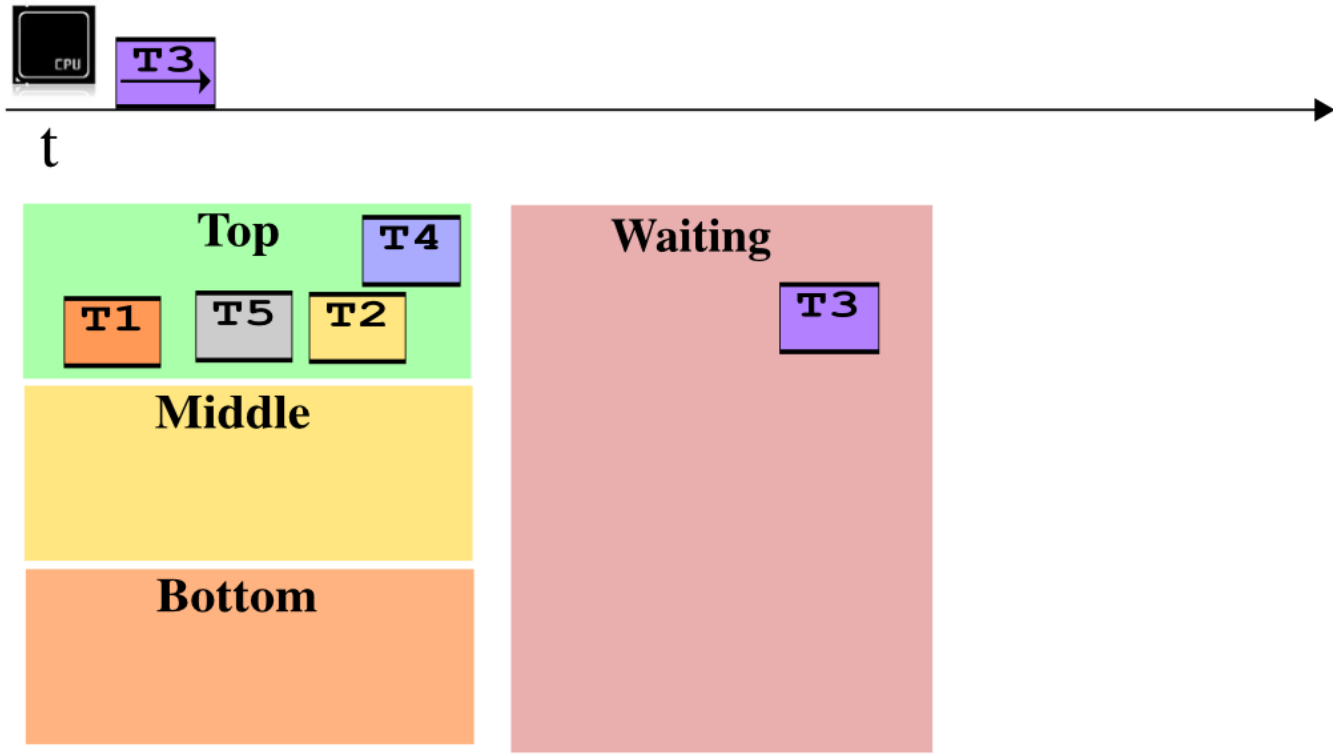
MLFQ



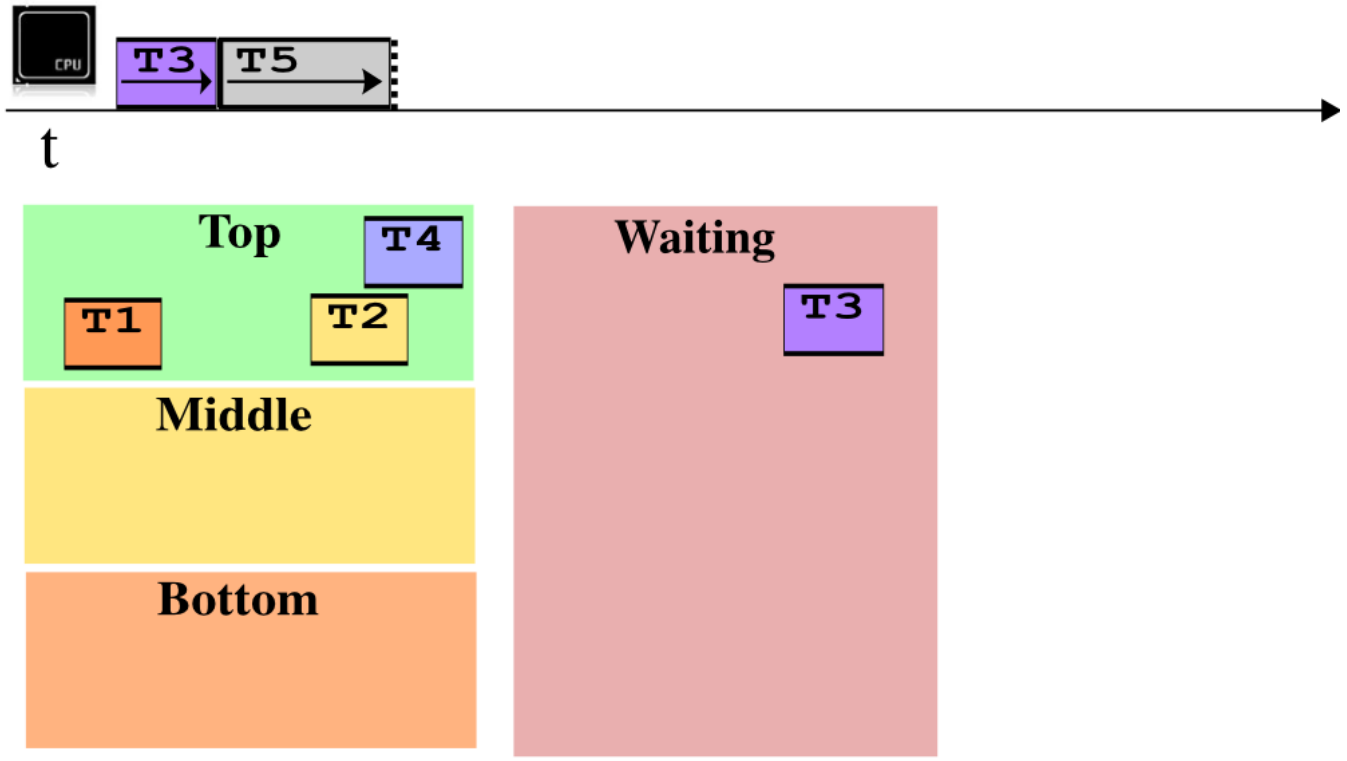
MLFQ



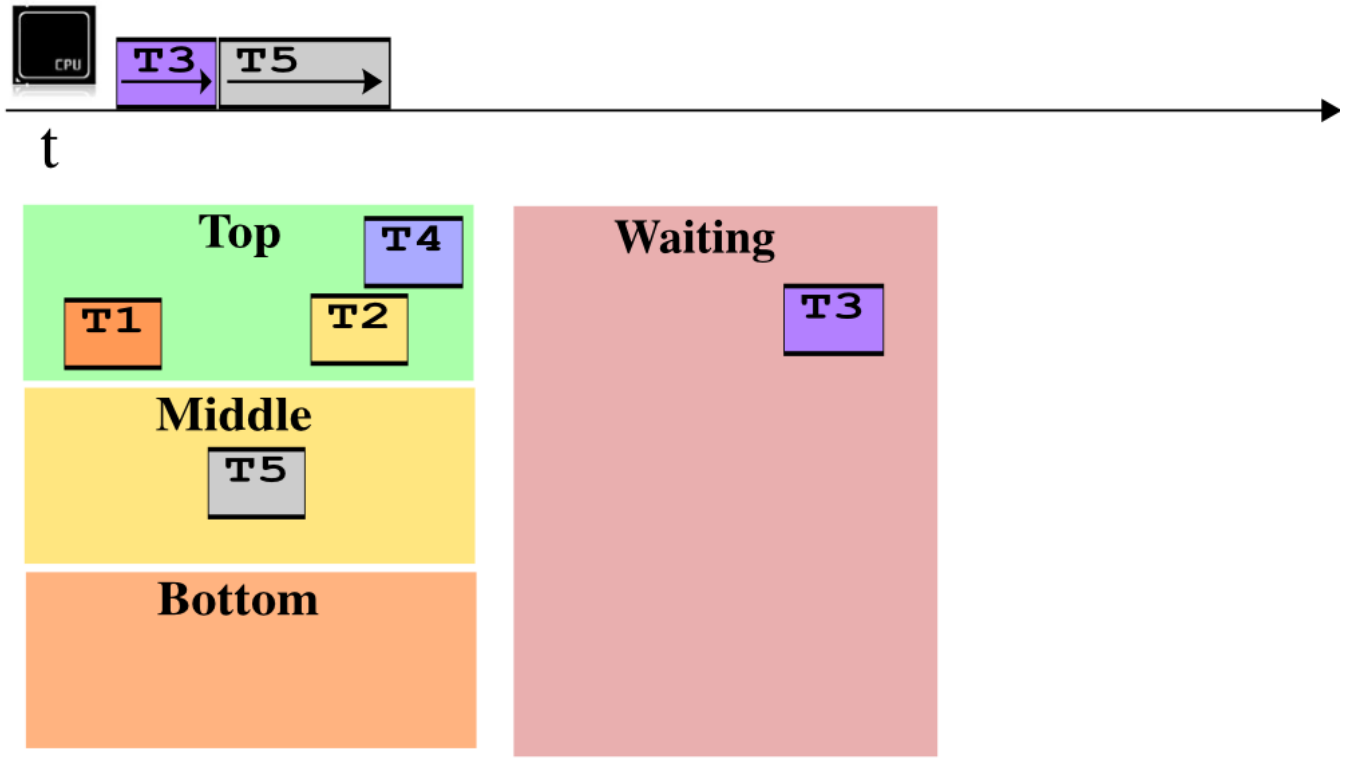
MLFQ



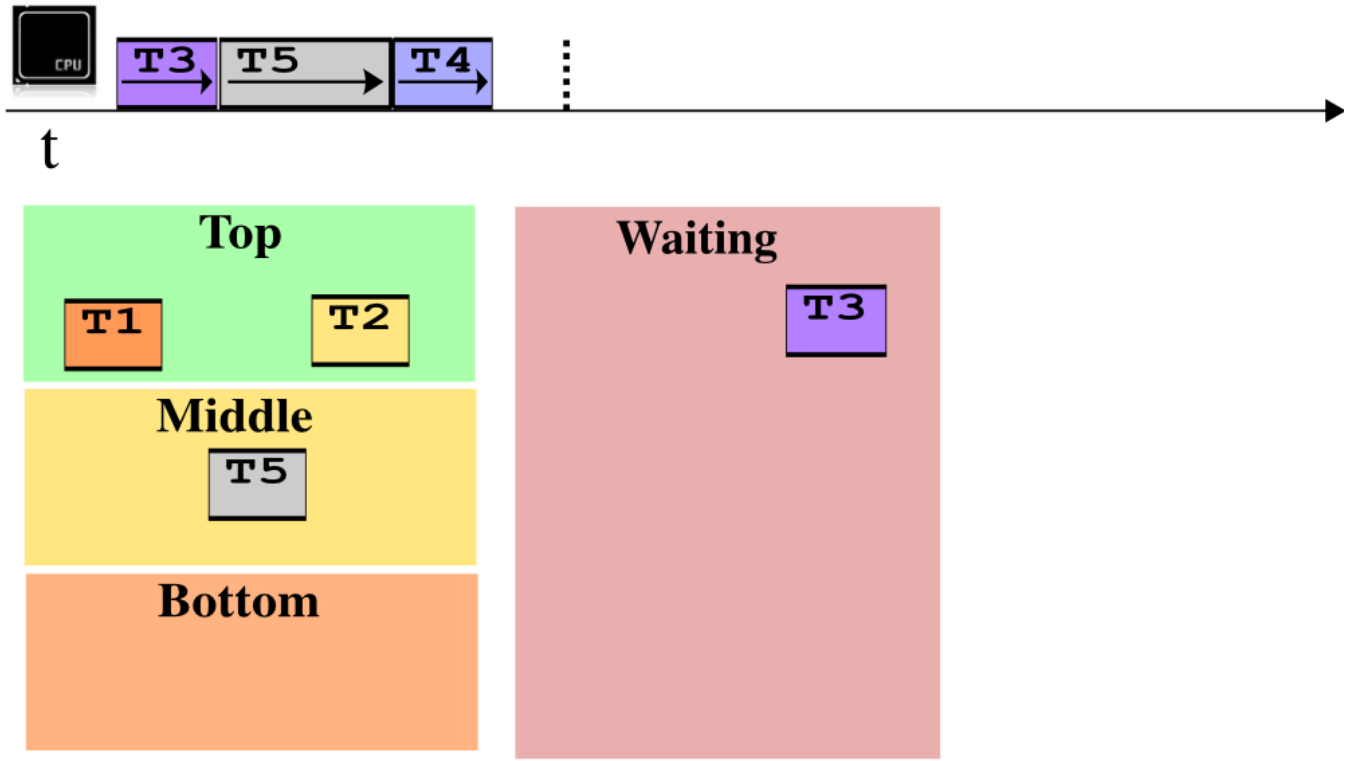
MLFQ



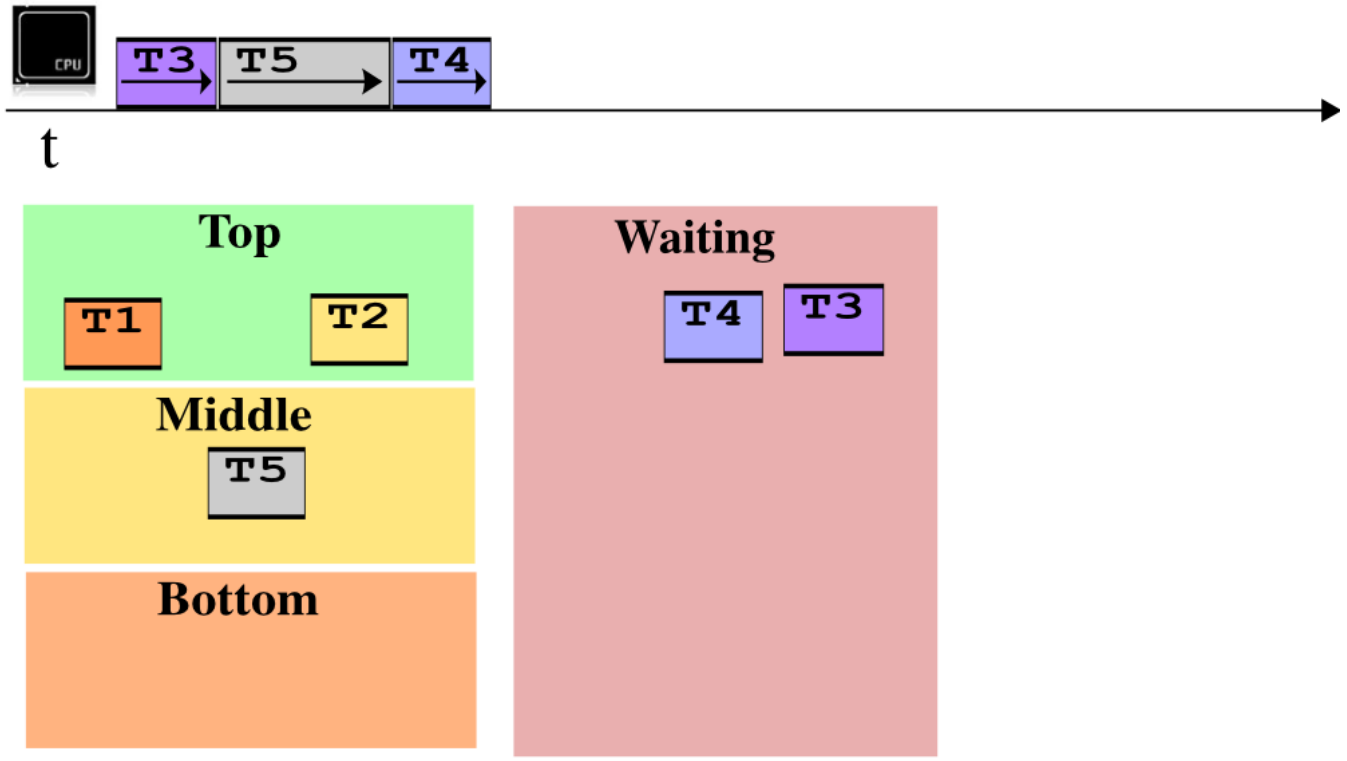
MLFQ



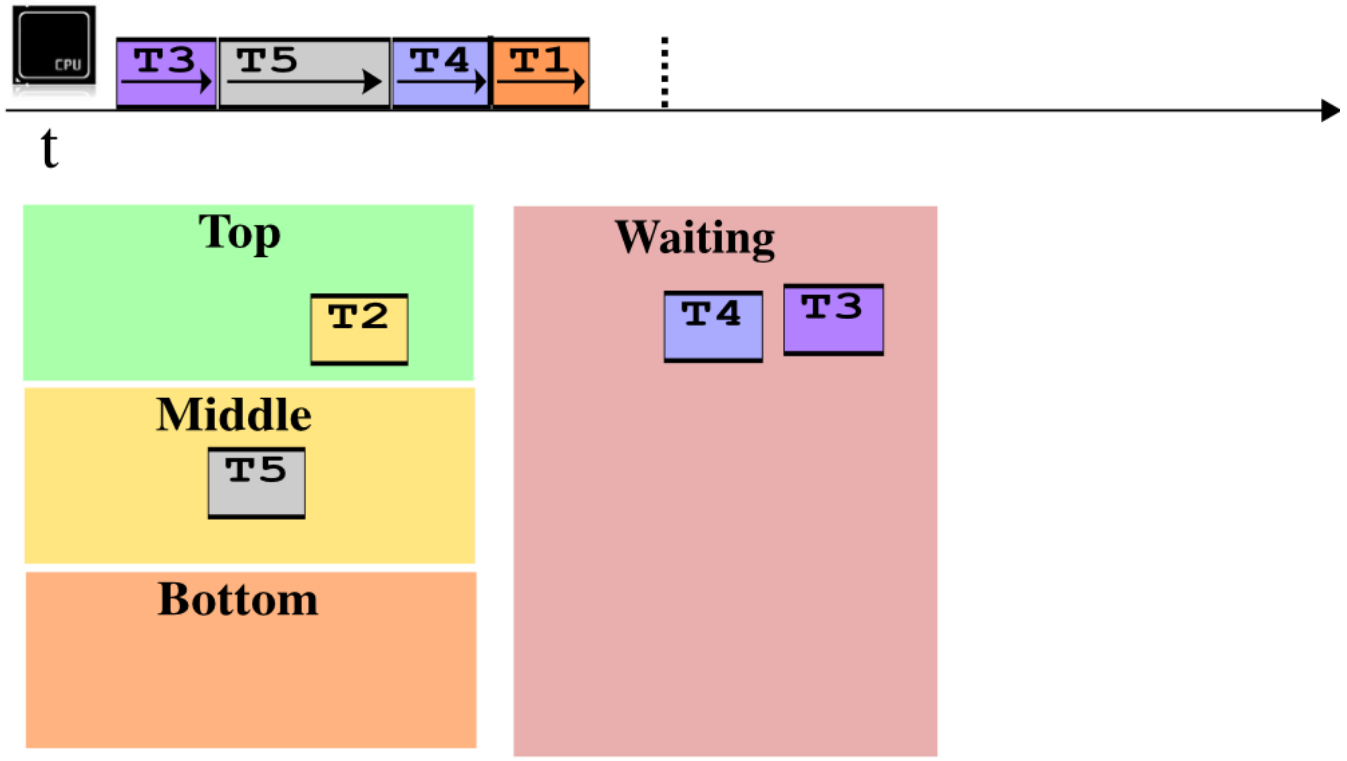
MLFQ



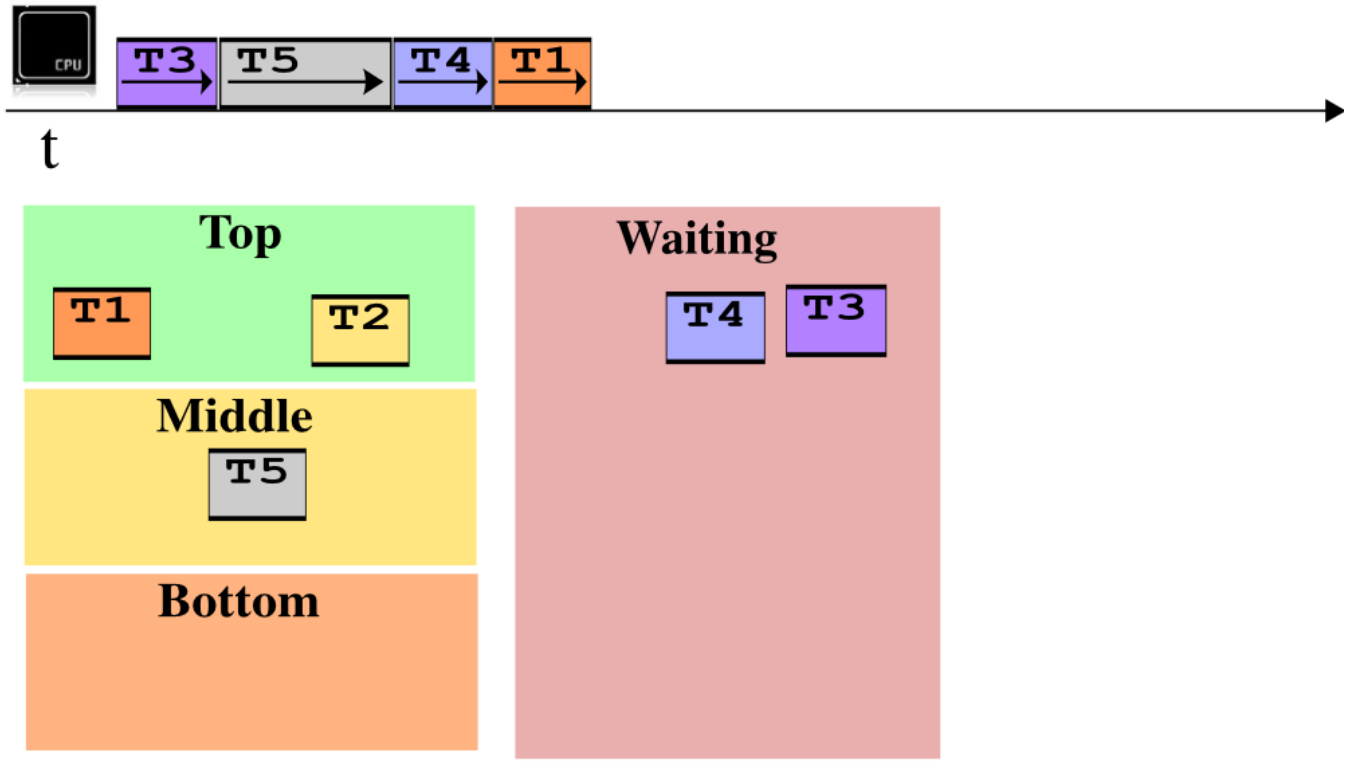
MLFQ



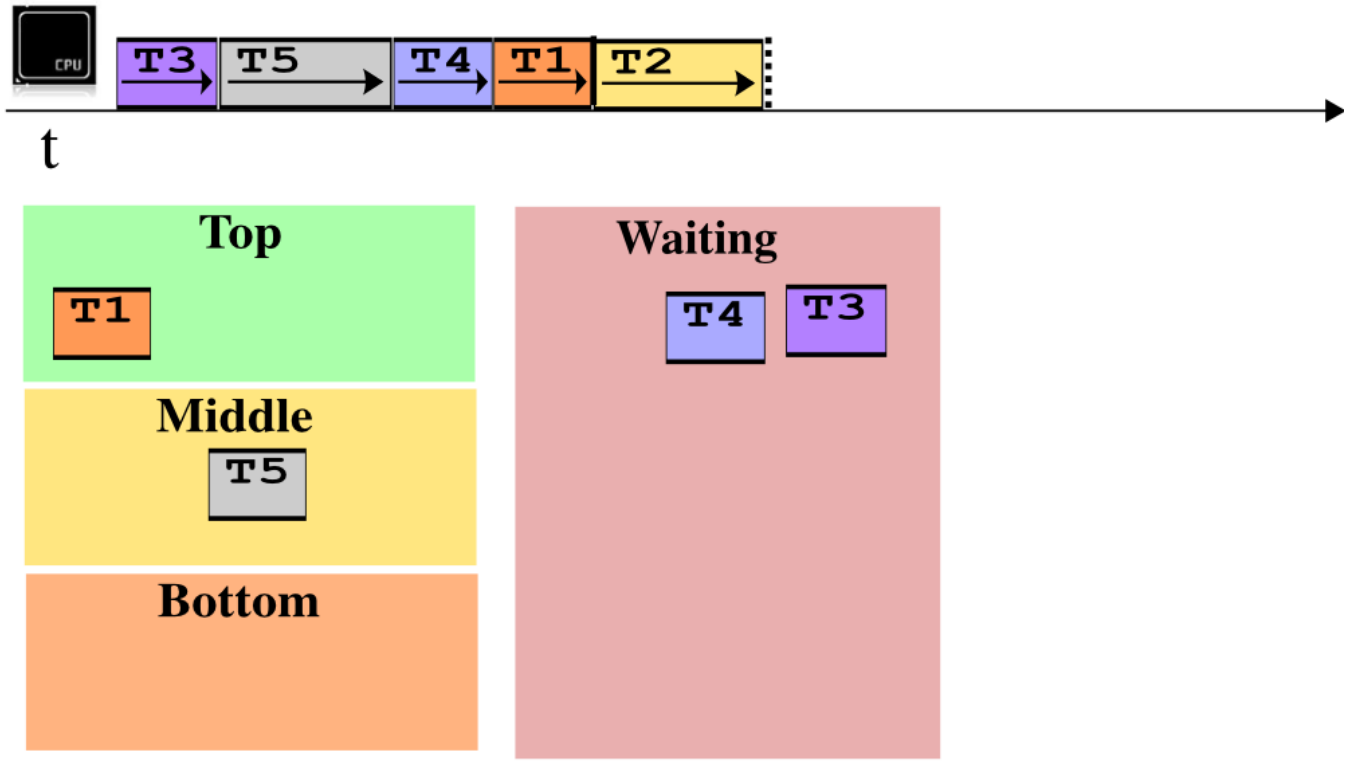
MLFQ



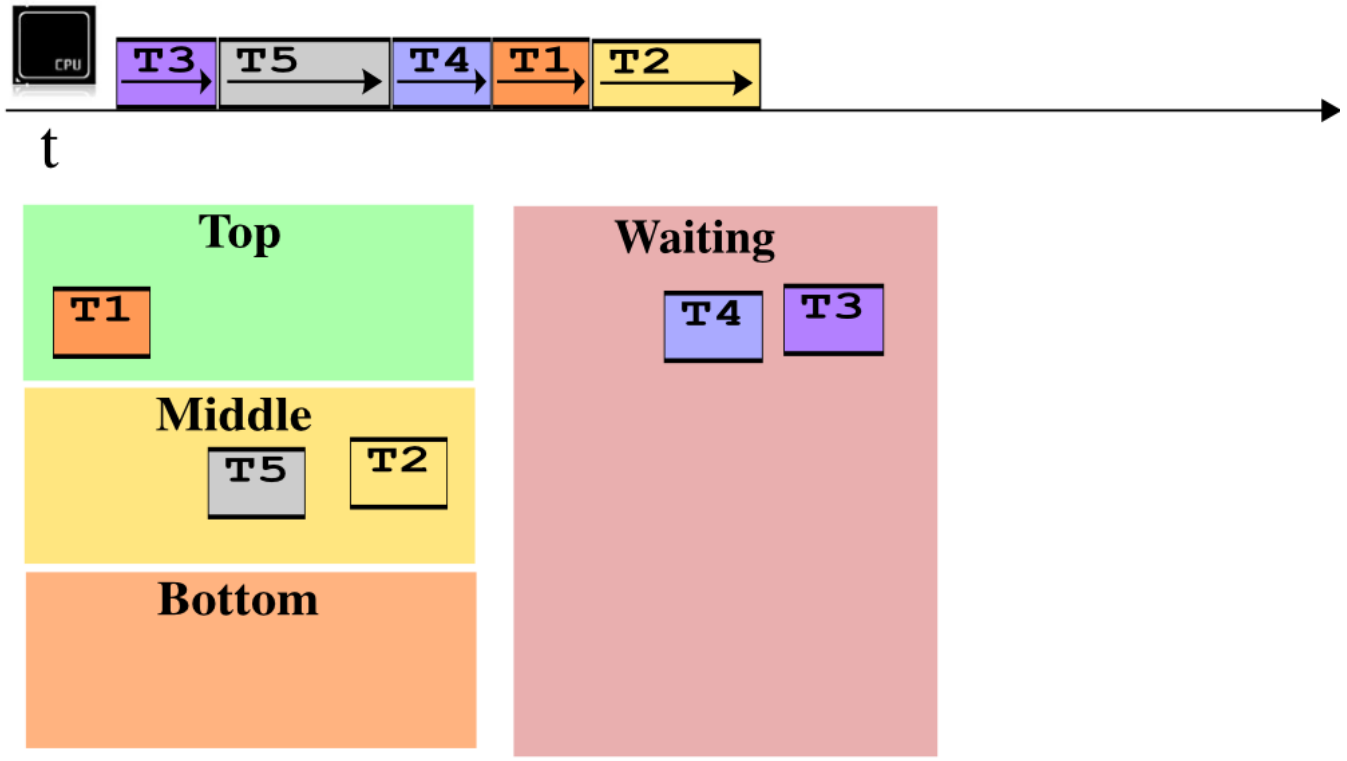
MLFQ



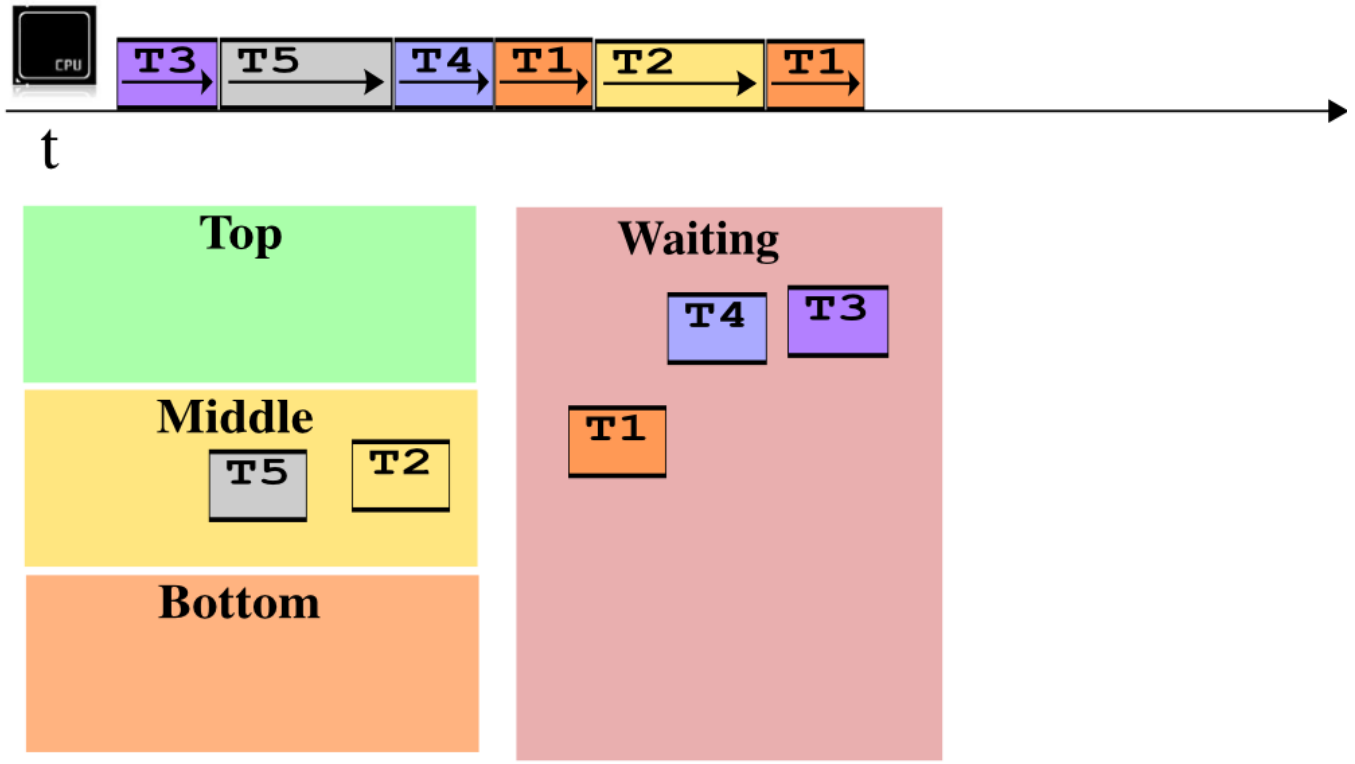
MLFQ



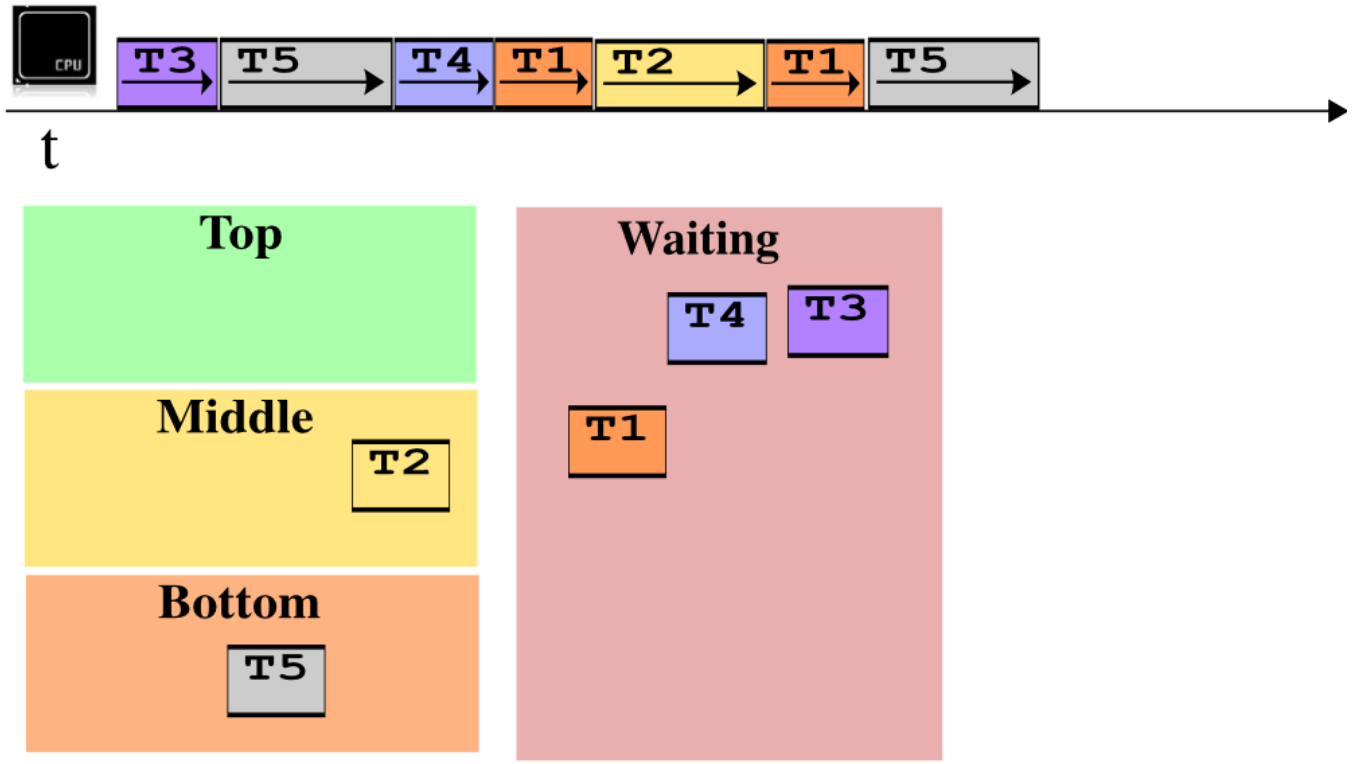
MLFQ



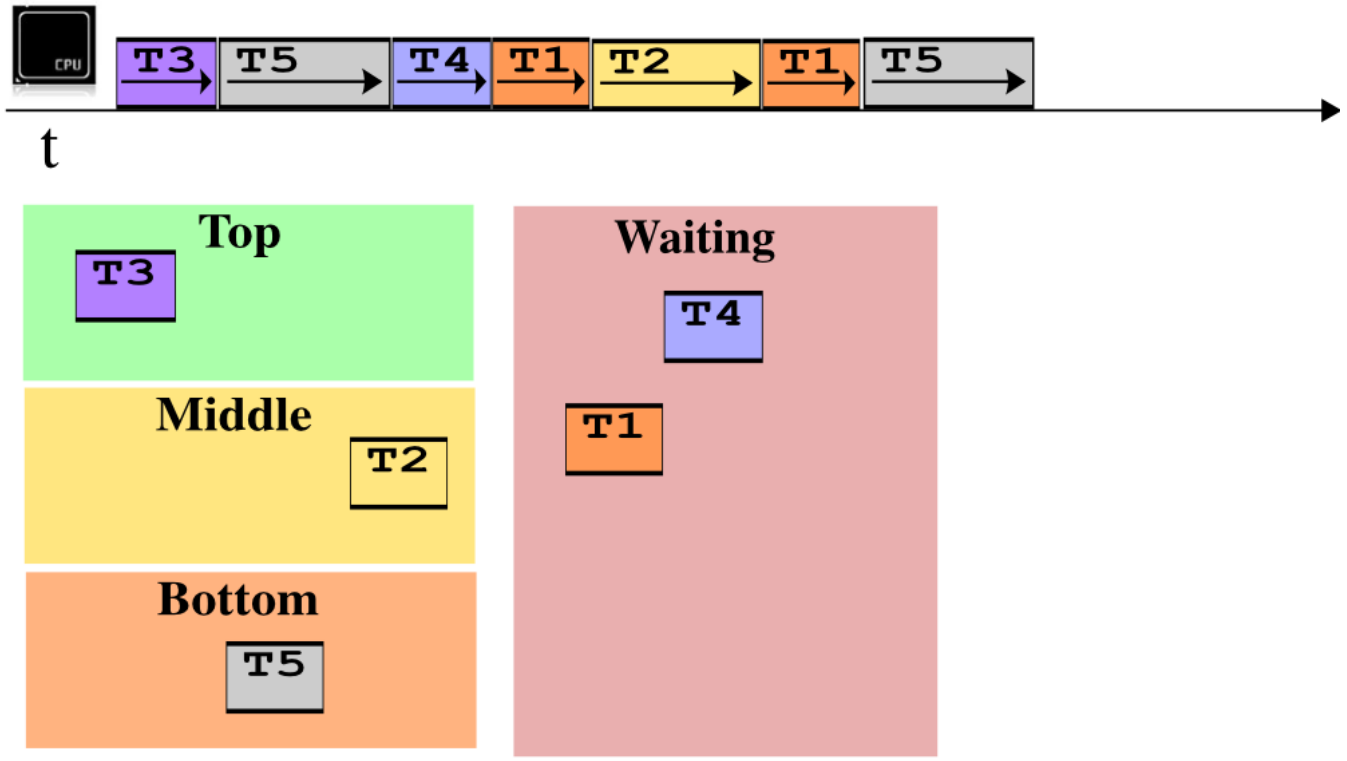
MLFQ



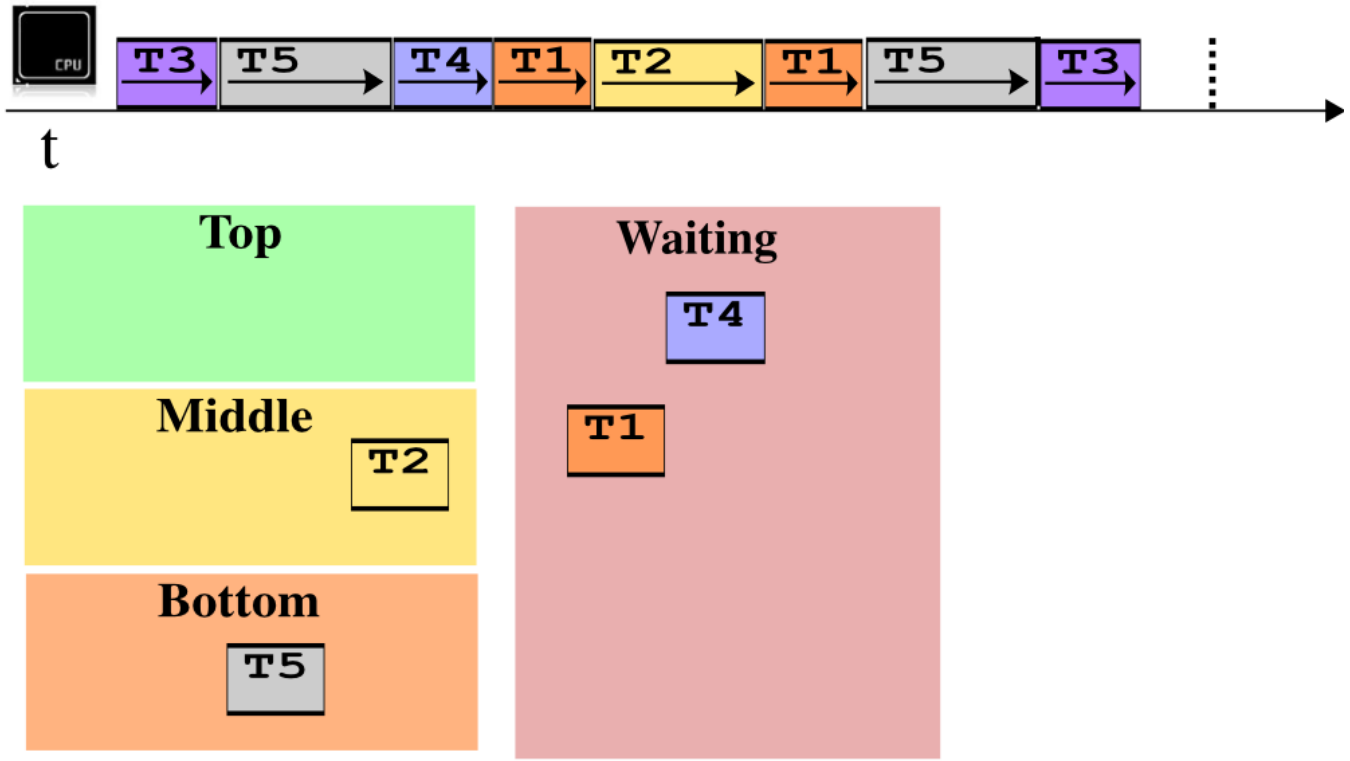
MLFQ


CPU
T3
T5
T4
T1
T2
T1
T5
t
Top
Middle
T2
Bottom
T5
Waiting
T4
T3
T1

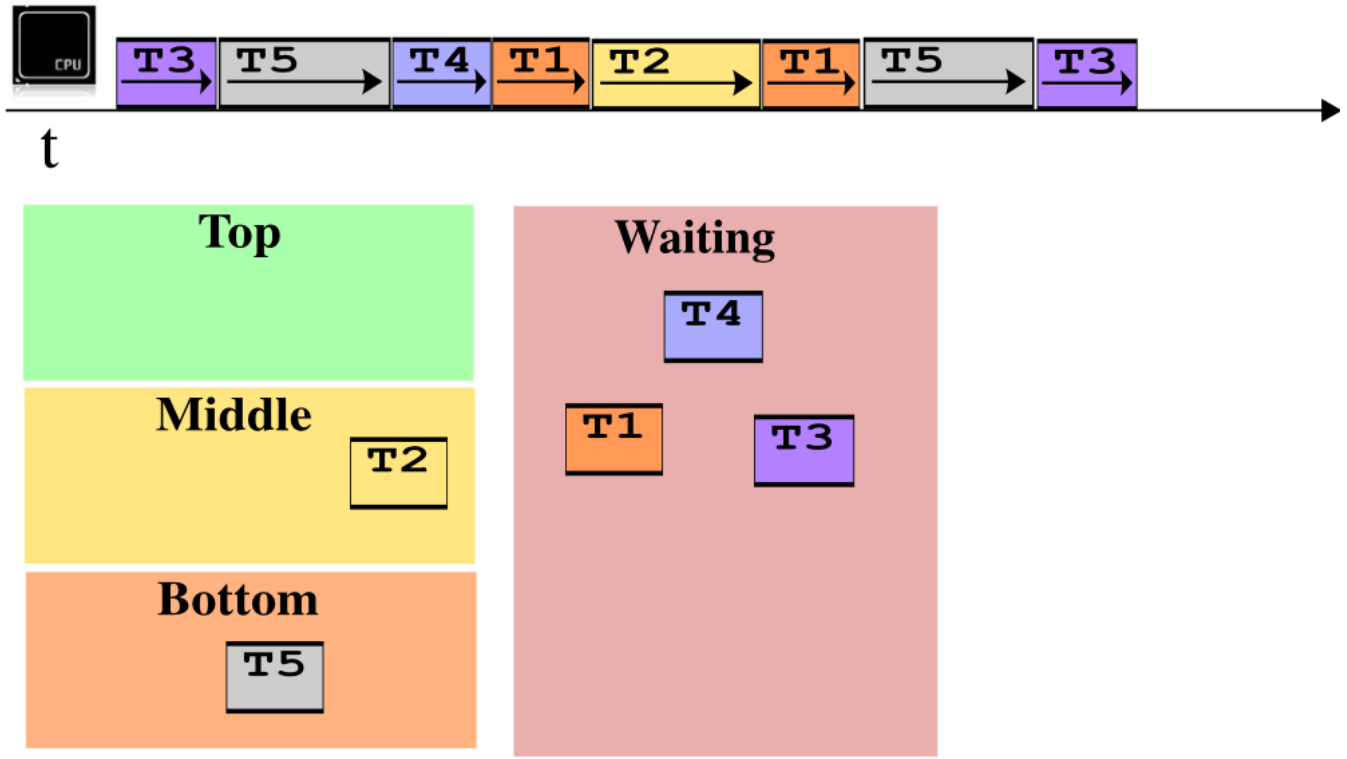
MLFQ


CPU
T3
T5
T4
T1
T2
T1
T5
t
Top
T3
Middle
T2
Bottom
T5
Waiting
T4
T1

MLFQ


CPU
T3
T5
T4
T1
T2
T1
T5
T3
t
Top
Middle
T2
Bottom
T5
Waiting
T4
T1

MLFQ


CPU
T3
T5
T4
T1
T2
T1
T5
T3
t
Top
Middle
T2
Bottom
T5
Waiting
T4
T1
T3

MLFQ Problems?

- What happens to:
  - CPU-bound threads? Keep going down.
  - I/O-bound threads? Keep rising up.
- What problems can this cause?
- **Starvation!** Threads trapped in the lower queues may never have a chance to run.
- One solution is to periodically **rebalance** the levels by periodically tossing everyone back to the top level.

Establish Priorities

- Priority : Deign some tasks more important than others, so that they can get more consideration while making scheduling decisions
- Relative for a given system
- On a desktop
  - Chrome rendering task
  - File indexing task
- On a smartphone
  - Candy Crush
  - Video playback

Linux Schedulers

- Prior to Linux 2.6, the scheduler scaled poorly : required O(n) time to make scheduling decisions
- For Linux 2.6, Ingo Molnar implemented an O(1) scheduler
- O(1) scheduler combines:
  - Static Priority: this is set by the user or system using nice
  - Dynamic Priority: Boosting priority at runtime to reward interactive threads.
    - This got real complex, real fast

Lottery Scheduling

- Strict priorities can lead to **starvation** when low-priority tasks are constantly blocked by high-priority tasks with work to do.

- Lottery scheduling
- Give each process a number of **tickets** proportional to their **priority**.
- Choose a ticket at random. The process holding the ticket gets to run

- Priorities may also be used to determine **how long** processes are allowed to run

Linux Schedulers

- Prior to Linux 2.6, the scheduler scaled poorly : required O(n) time to make scheduling decisions
- For Linux 2.6, Ingo Molnar implemented an O(1) scheduler
- O(1) scheduler combines:
  - Static Priority: this is set by the user or system using nice
  - Dynamic Priority: Boosting priority at runtime to reward interactive threads.
    - This got real complex, real fast

A Scheduler Story

- Con Kolivas is an Australian anaesthetist
- Got interested in kernel programming, particularly CPU Scheduling
- Known for developing Rotating Staircase Deadline (RSDL) and BFS
- His work inspired Ingo to write CFS

- Moral of the story: If an anaesthetist can do it, so can you

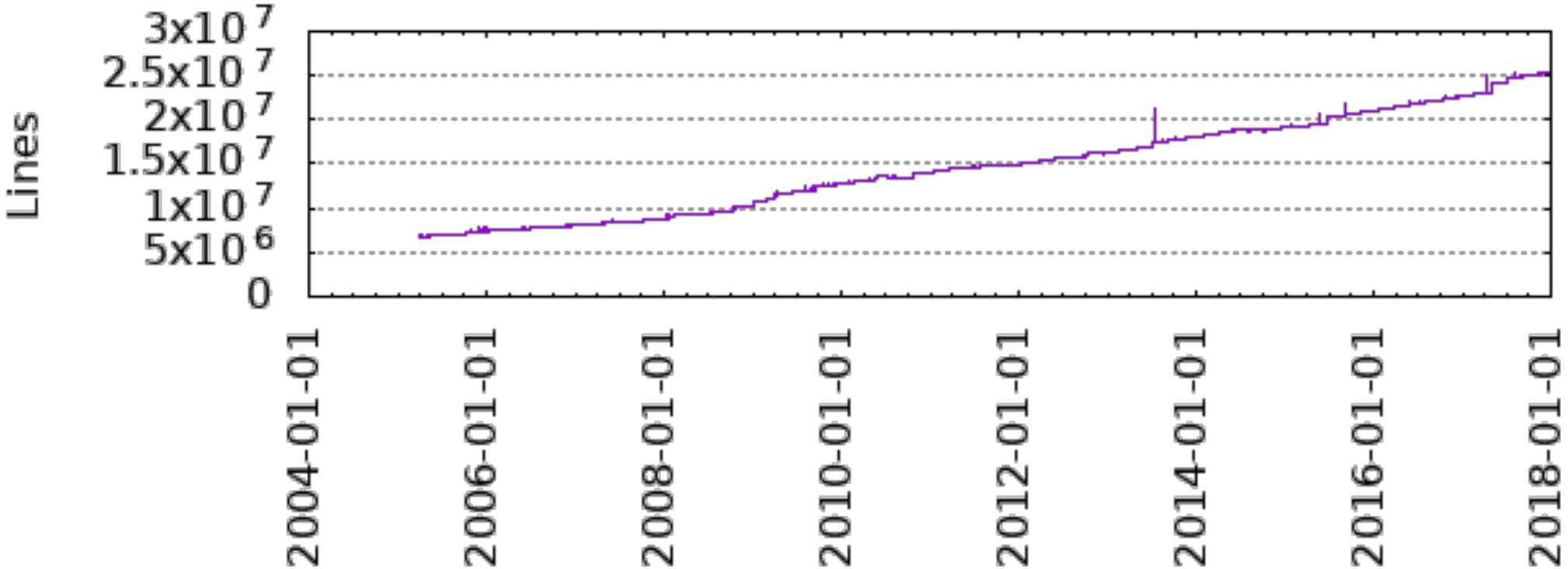
https://en.wikipedia.org/wiki/Con_Kolivas

Completely Fair Scheduler (CFS)

- CFS models an “ideal, precise multitasking CPU”
- Calculates how long a process should run as a function of the total number of currently runnable processes
- Higher priority jobs get larger weights => more CPU-time
- CFS achieves fairness by letting a task run for a period
  - Run time is proportional to its weight divided by the total weight of all runnable processes
- Default scheduler since Linux 2.6
- Maintained by Ingo Molnár

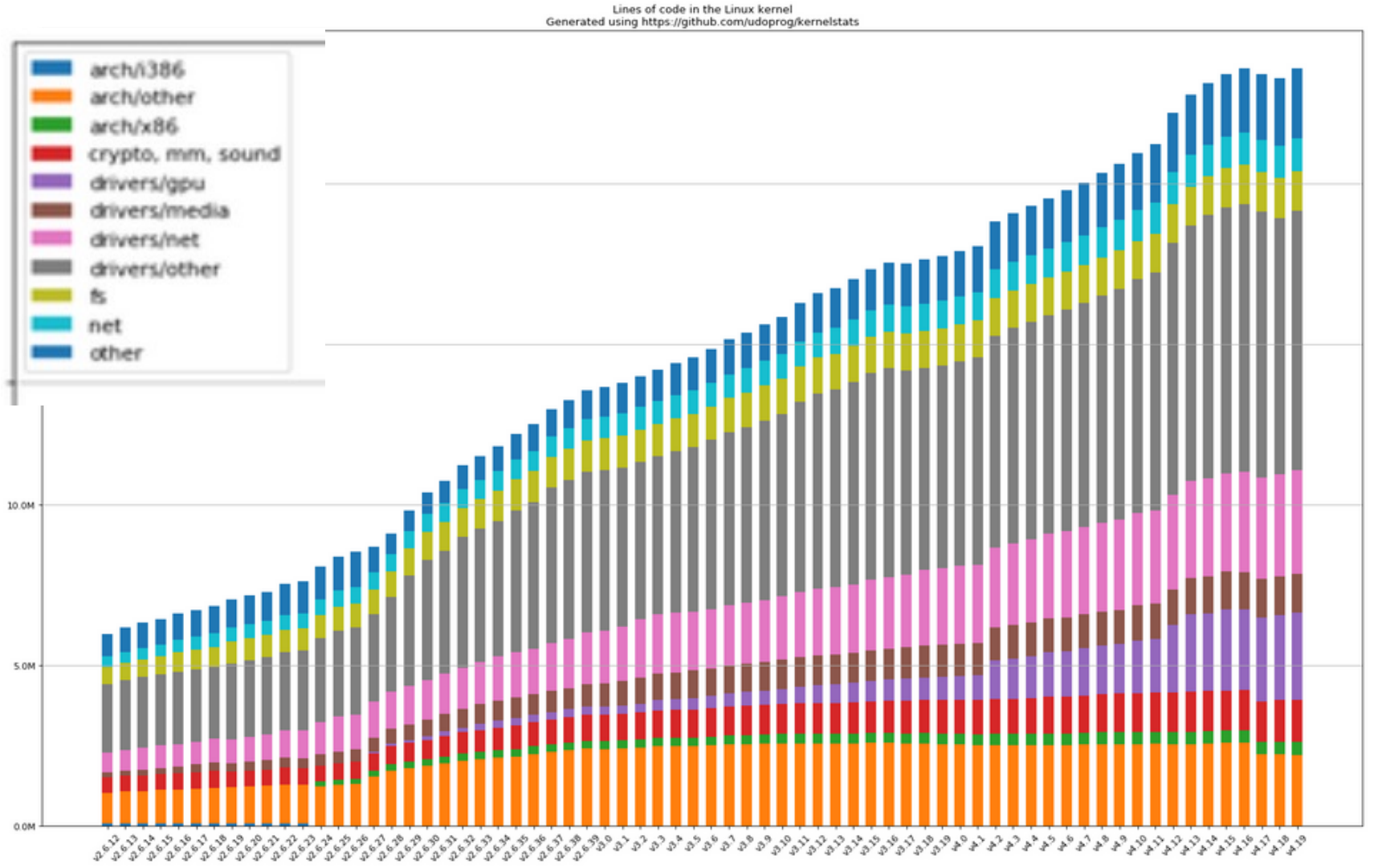
https://www.linuxjournal.com/node/10267

Linux Facts



https://www.phoronix.com/scan.php?page=news_item&px=Linux-Kernel-Commits-2017

Linux Facts



https://www.reddit.com/r/linux/comments/9uxwli/lines_of_code_in_the_linux_kernel/

Linux Facts

GitStats - linux

General | Activity | Authors | Files | Lines | Tags

Project name:
linux
Generated:
2018-01-01 12:05:12 (in 15451 seconds)
Generator:
GitStats (version 2014-12-09), git version 2.13.6, gnuplot 5.0 patchlevel 5
Report Period:
1969-12-31 18:00:01 to 2037-04-25 03:08:26
Age:
24587 days, 4744 active days (19.29%)
Total Files:
62296
Total Lines of Code:
25359556 (44680251 added, 19320695 removed)
Total Commits:
723641 (average 152.5 commits per active day, 29.4 per all days)
Authors:
17888 (average 40.5 commits per author)

https://phoronix.com/misc/linux-2017/index.html

Linux Development Process

- Each file has a maintainer.
- Each subsystem has a maintainer.
- And then there’s Linus

Apparently, not a very nice guy

The e-mails of the celebrated programmer Linus Torvalds land like thunderbolts from on high onto public lists, full of invective, insults, and demeaning language. "Please just kill yourself now. The world will be a better place," he wrote in one.

AFTER YEARS OF ABUSIVE E-MAILS, THE CREATOR OF LINUX STEPS ASIDE

By Noam Cohen September 19, 2018

After years of verbally abusing programmers who contribute to the Linux operating-system kernel he created, the celebrated coder Linus Torvalds is stepping aside and says he is getting help.

Photograph by Kimmo Mäntylä / REX / Shutterstock

https://www.newyorker.com/science/elements/after-years-of-abusive-e-mails-the-creator-of-linux-steps-aside

Linux Development Process

- Linus and others maintain the **mainline** Linux kernel tree used to generate official releases.

- Other developers maintain their own trees to experiment with new features.
  - Some features ***may*** move into the mainline.

- Git was built to facilitate this working model

The Rotating Staircase Deadline Scheduler

- To begin a scheduling epoch:
- Put all threads in a queue determined by their priority.
- Then:
  - Run threads from the highest non-empty queue round-robin
  - If a thread **blocks** or **yields**, it remains at that level
  - If a thread runs out of its quota, it moves to the **next level down.**
  - If the level runs out of *its* quota, **all threads move to the next level down.**
  - Continue until all quotas are exhausted or no threads are runnable, then restart another epoch