Preemptive Scheduling

Operating Systems, Fall 2026

Instructors: Travis McGaha

Head TAs: Shriya Sane

Arnuv Batra
Khoi Dinh
Sydney Fitzgerald
Rania Souissi
Jeffrey Oduman
Abdullah Goher
Monisha Krothapalli
Veer Kakar
Kevin Xue
Shreya Mukunthan
Daniel Price
Wesley Liu
Leo Zhang

Logistics

- Penn-FAT is out!
  - It is due October 29th in pairs!
  - It is a HUGE assignment. And make sure to put in a lot of elbow grease.
  - Some questions on the midterm will ask about concepts enforced by this assignment.
  - Recitation this week is designed to be helpful for it!
- Penn-fat milestone details posted on Tuesday!
  - Please try to meet with your TA by the end of next week!
- Final exam schedule posted:
  - Tuesday, December 15, 2026: 3:00pm to 5:00pm in JMHH G06
- Midterm coming up in class on October 22nd

Any Questions?

Lecture Outline

- **Review**
- Preemptive Algorithms
  - Round Robin
  - Round Robin Variants
  - Completely Fair Scheduling (CFS)
- "Nice" Threads
- Extra: Earliest Eligible Virtual Deadline First, Interactive Tasks, Etc.

Example of SJF

- Example workload with three "jobs"

1 CPU
Job 1, 2, & 3 arrive at the same time

	Time to Finish
Job 1	**24**
Job 2	**3**
Job 3	**3**

- SJF schedule:

- Total wait time: 0 + 3 + 6 = 9
- Average wait time: 9/3 = 3
- Total turnaround time: 3 + 6 + 30 = 39
- Average turnaround time: 39/3 = 13

Poll From last time

- What are the advantages/disadvantages/concerns with **Shortest Job First**

- Things a scheduler should prioritize:
  - Minimizing wait time
  - Minimizing latency
  - Maximizing fairness
  - Maximizing throughput
  - Task priority
  - Cost to schedule things
  - Cost to context switch

- Imagine we have 1 core, and tasks of various (finite) lengths

SJF Analysis

- Advantages:
  - Still relatively simple, low overhead
  - perhaps minimal average turnaround time

- Disadvantages
  - Starvation possible
    - If quick jobs keep arriving, long jobs will keep being pushed back and won't execute
  - How do you know how long it takes for something to run?
    - You can't! You can use a history of past behavior to make a guess.
  - Priority not taken into considerations

Lecture Outline

- Review
- **Preemptive Algorithms**
  - **Round Robin**
  - **Round Robin Variants**
  - **Completely Fair Scheduling (CFS)**
- "Nice" Threads
- Extra: Earliest Eligible Virtual Deadline First, Interactive Tasks, Etc.

Types of Scheduling Algorithms

- **Preemptive:** the thread may be ***interrupted*** after a given time and/or if another thread becomes ready
  - Round Robin
  - Priority Round Robin
  - Completely Fair Scheduling (CFS)
  - Earliest Eligible Virtual Deadline First (EEVDF)

Round Robin

- Sort of a preemptive version of FCFS
  - Whenever a thread is ready, add it to the end of the queue
  - Run whatever job is at the front of the queue

- *But* only let it run for a fixed amount of time (quantum)
  - If it finishes before the time is up, schedule another thread to run
  - If time is up, then send the running thread back to the end of the queue

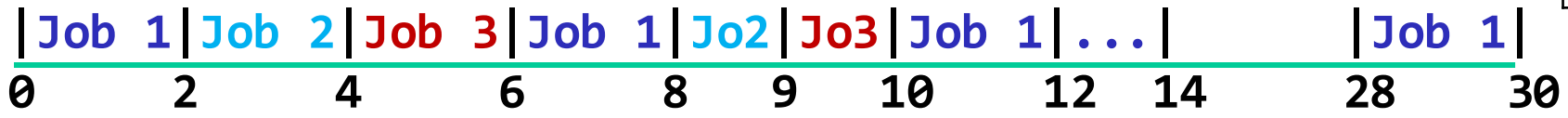
Example of Round Robin

- Example workload with three “jobs”
  - Time quantum: 2

1 CPU
Job 2 arrives slightly after job 1
Job 3 arrives slightly after job 2

	Time to Finish
Job 1	**24**
Job 2	**3**
Job 3	**3**

- Round Robin schedule:



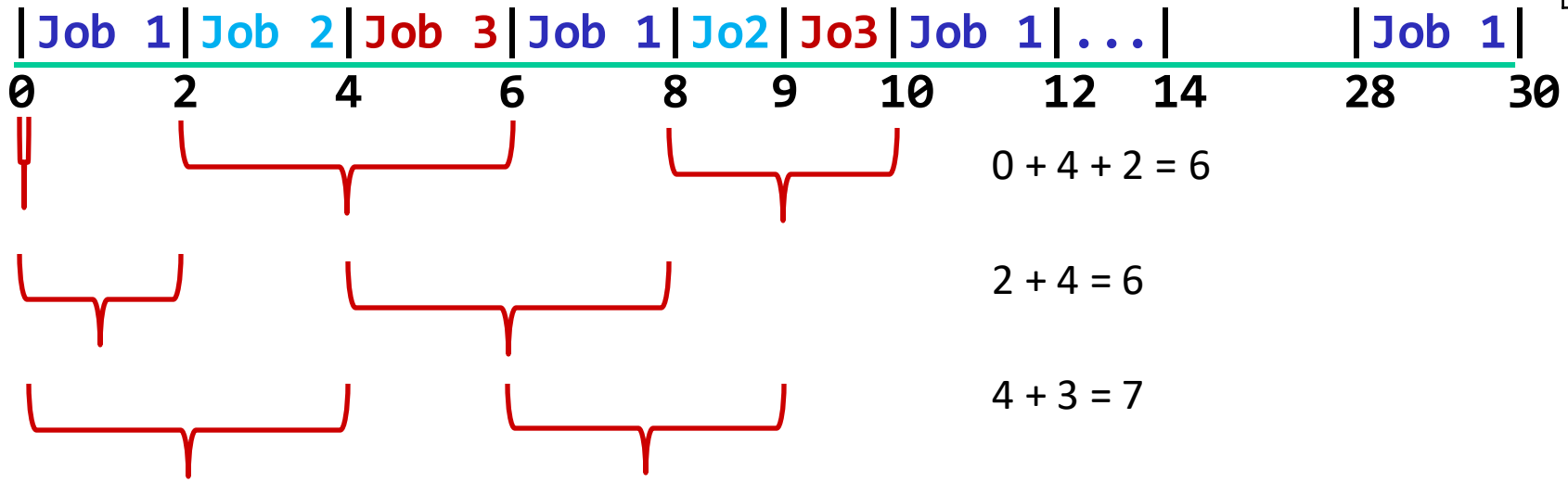
Example of Round Robin

- Wait Time: from ready to running
  - Time quantum: 2

1 CPU
Job 2 arrives slightly after job 1
Job 3 arrives slightly after job 2

	Time to Finish
Job 1	**24**
Job 2	**3**
Job 3	**3**

- Round Robin schedule:



- Total wait time: 6 + 6 + 7 = 19
- Average wait time: 19/3 = ~6.333…

Example of Round Robin

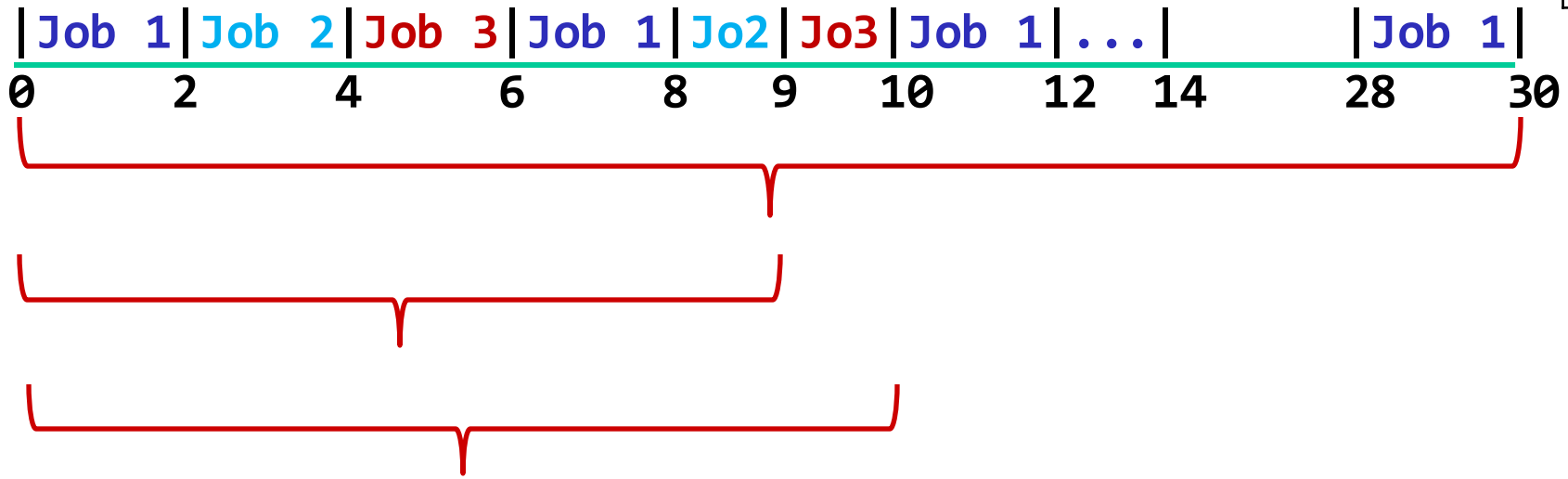
- Turnaround Time: from arrival to finished
  - Time quantum: 2

1 CPU
Job 2 arrives slightly after job 1
Job 3 arrives slightly after job 2

Time to Finish

Job 1	**24**
Job 2	**3**
Job 3	**3**

- Round Robin schedule:



- Total turnaround time: 30 + 9 + 10 = 49
- Average turnaround time: 49/3 = ~16.333…

Poll Everywhere

Discuss

- What are the advantages/disadvantages/concerns with **Round Robin**

- Things a scheduler should prioritize:
  - Minimizing wait time
  - Minimizing latency
  - Maximizing fairness
  - Maximizing throughput
  - Task priority
  - Cost to schedule things
  - Cost to context switch

- Imagine we have 1 core, and tasks of various lengths

Round Robin Analysis

- Advantages:
  - Still relatively simple
  - Can work for interactive systems

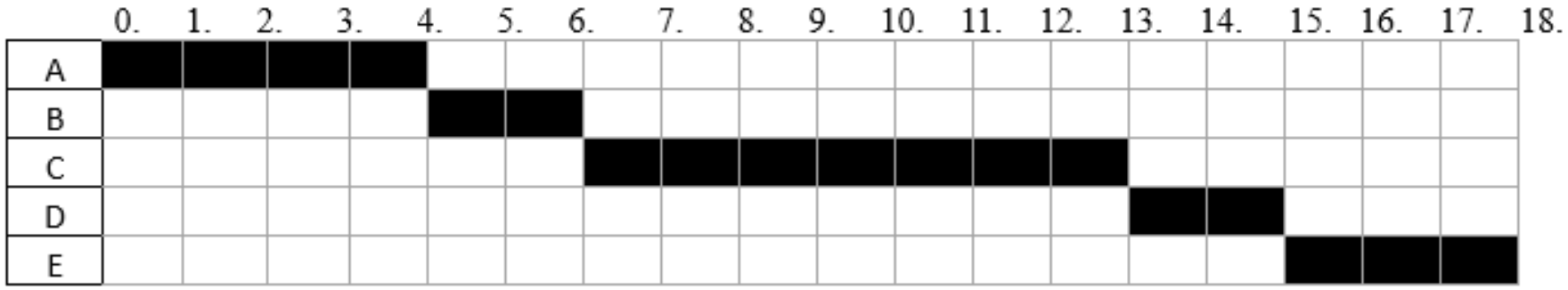
- Disadvantages
  - If quantum is too small, can spend a lot of time context switching
  - If quantum is too large, approaches FCFS
  - Still assumes all processes have the same priority

- Rule of thumb:
  - Choose a unit of time so that most jobs (80-90%) finish in one usage of CPU time

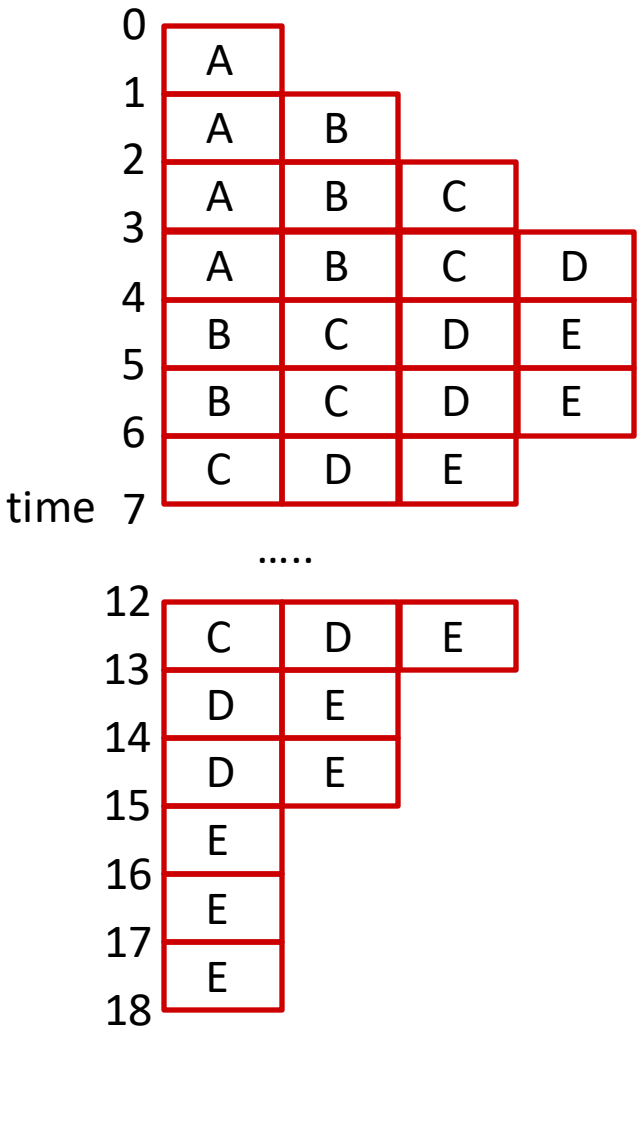
FCFS Example!

- Assume that we had the following set of processes that ran on a single CPU following the First Come First Serve (FCFS) scheduling policy
- If we expressed this with a drawing, where a black square represents that the process is executing, we would get:

Process	Arrival Time	Finishing Time
A	0	4
B	1	6
C	2	13
D	3	15
E	4	18



FCFS Example: Different Visualization



Process	Arrival Time	Finishing Time
A	0	4
B	1	6
C	2	13
D	3	15
E	4	18

= 1 unit of time

Round Robin Practice!

- Let's switch our algorithm to round-robin with a time quantum of 3 time units

Process	Arrival Time	Finishing Time
A	0	4
B	1	6
C	2	13
D	3	15
E	4	18

- You can assume:
  - Context switching and running the Scheduler are instantaneous
  - If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice

	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.	17.	18.
A																			
B																			
C																			
D																			
E																			

Round Robin Practice!

- Let's switch our algorithm to round-robin with a time quantum of 3 time units

Process	Arrival Time	Finishing Time
A	0	4
B	1	6
C	2	13
D	3	15
E	4	18

Process	Arrival Time	Length
A	0	4
B	1	2
C	2	7
D	3	2
E	4	3

Useful to convert to this style of table.

- You can assume:
  - Context switching and running the Scheduler are instantaneous
  - If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice

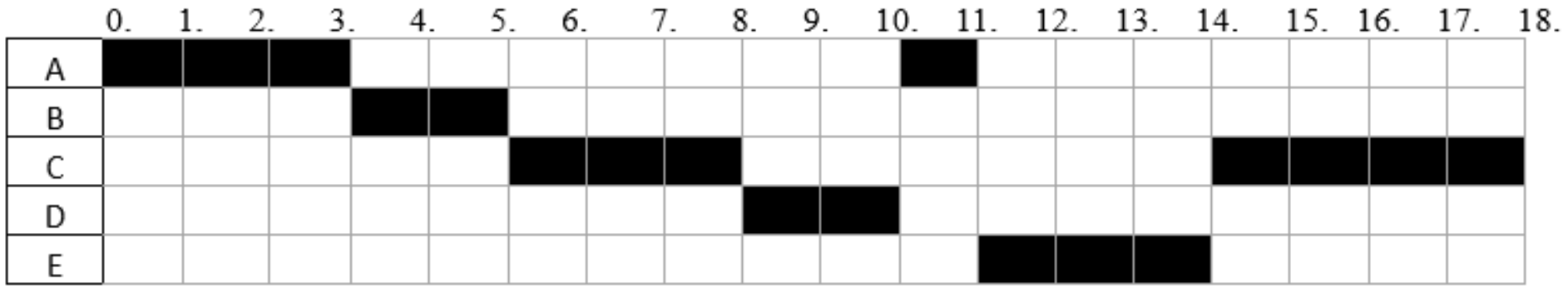
	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.	17.	18.
A																			
B																			
C																			
D																			
E																			

Poll Everywhere

pollev.com/tqm

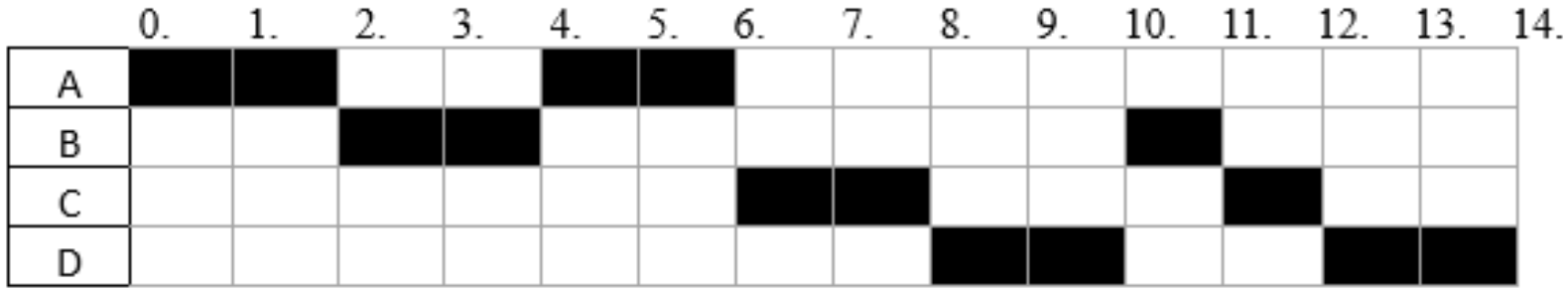
- What is the average wait time?
- What is the average turnaround time?

Process	Arrival Time	Length
A	0	4
B	1	2
C	2	7
D	3	2
E	4	3



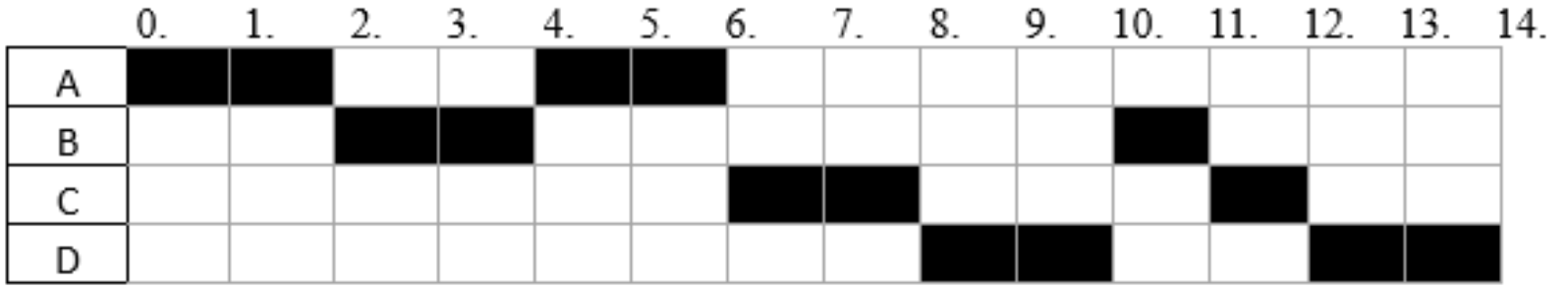
More Round Robin Practice

- Four processes are executing on one CPU following round robin scheduling:



- You can assume:
  - All processes do not block for I/O or any resource
  - Context switching and running the Scheduler are instantaneous
  - If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice

More Round Robin Practice



- What is the earliest time that process C could have arrived?
- Which processes are in the ready queue at time 9?
- If this algorithm used a quantum of 3 instead of 2, how many fewer context switches would there be?

RR Variant: PennOS Scheduler

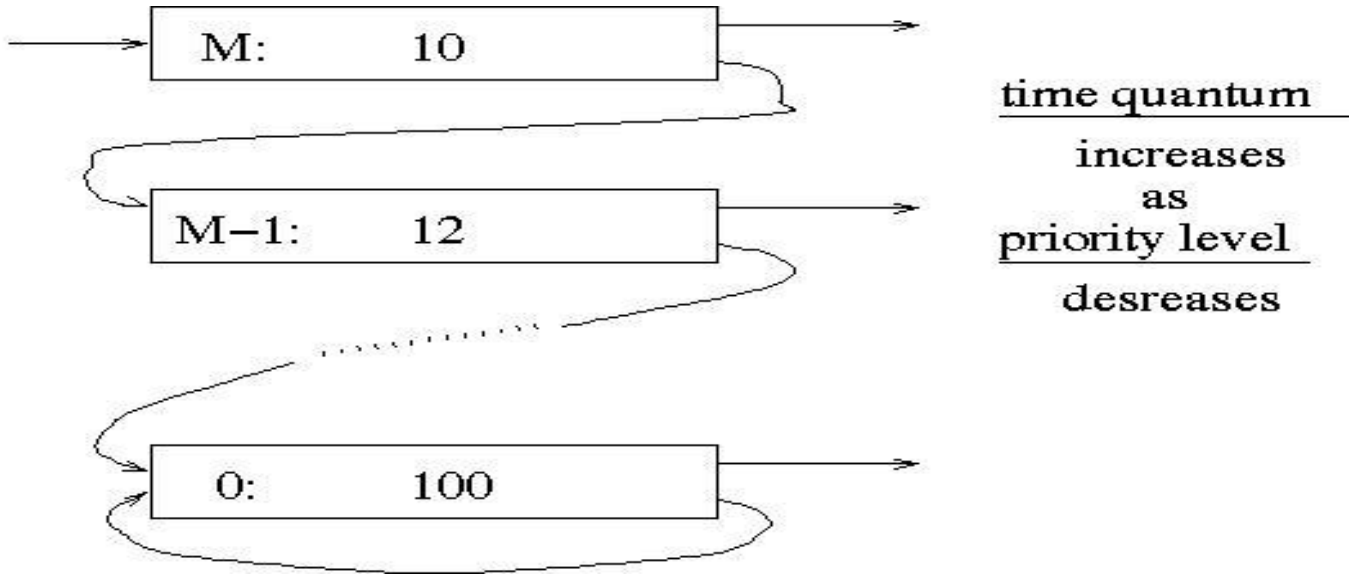
- In PennOS you will implement a priority scheduler based on Round Robin
- You will have 3 queues, each with a different priority (0, 1, 2)
  - Each queue acts like normal round robin within the queue
- You spend time quantum processing each queue proportional to the priority
  - Priority 0 is scheduled 1.5 times more often than priority 1
  - Priority 1 is scheduled 1.5 times more often than priority 2

RR Variant: Priority Round Robin

- Same idea as round robin, but with multiple queues for different priority levels
- Scheduler chooses the first item in the highest priority queue to run
- Scheduler only schedules items in lower priorities if all queues with higher priority are empty

RR Variant: Multi Level Feedback

- Each priority level has a ready queue and a time quantum
- Thread enters highest priority queue initially
  - Move to lower queue with each timer interrupt (i.e. scheduler pre-emption)
- If a thread 'voluntarily' stops using CPU before time is up, it is moved to the end of the current queue
- Threads in a queue are not scheduled until all higher queues are empty
- Bottom queue is standard Round Robin



Multi Level Feedback Analysis

- Threads with high I/O bursts are preferred
  - Makes higher utilization of the I/O devices
  - Good for interactive programs (keyboard, terminal, mouse I/O)
- Threads that need the CPU a lot will sink to lower priority, giving shorter threads a chance to run
- Still must be careful in choosing time quantum
- Must also be careful in choosing how many layers

Multi Level Feedback Variants: Priority

- Can assign tasks different priority levels upon initiation that decide which queue it starts in
  - E.g. the bluetooth daemon should have higher priority than `HelloWorld.java`
- Update the priority based on recent CPU usage rather than overall CPU usage of a task
  - Makes sure that priority is consistent with recent behavior
- Many others that vary from system to system

Multiple Cores

- On a modern machine, we have multiple CPU Cores, each can run tasks
  - ***Generally***, each core has its own run-queue
  - It helps to keep threads in the same process on the same processor
  - Threads in the same process use the same memory: lower overhead
    - If we want to there are ways to make sure a thread/process is “pinned” to a CPU
      - See: Thread Affinity / Processor Affinity / CPU Pinning

- There is other stuff to balance tasks across cores, but I am leaving that out for time ☺

Completely Fair Scheduler

- "Fairness" – making sure that each task gets its fair share of the CPU
  - This is not always achievable
  - "Fairness, it turns out, is enough to solve many CPU-scheduling problems."

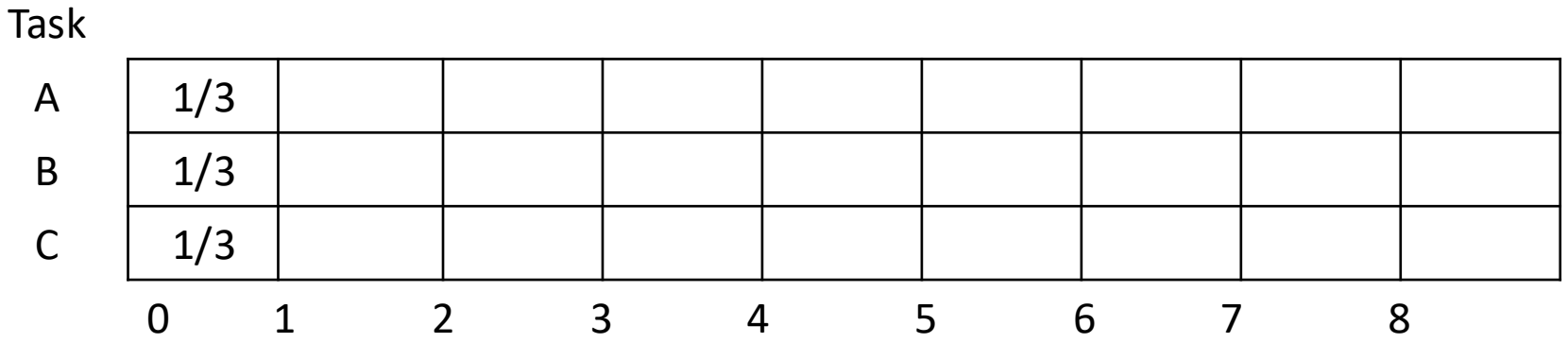
Completely Fair Scheduler

- "Fairness" – making sure that each task gets its fair share of the CPU
  - This is not always achievable
  - "Fairness, it turns out, is enough to solve many CPU-scheduling problems."

This is what CFS is trying to replicate as if we are on an "Ideal Processor"

- Here is an example of fairness:
  - Within some "slice" of time, each task gets an equal proportion of the processor

TASK	Run Time
A	1
B	5
C	2



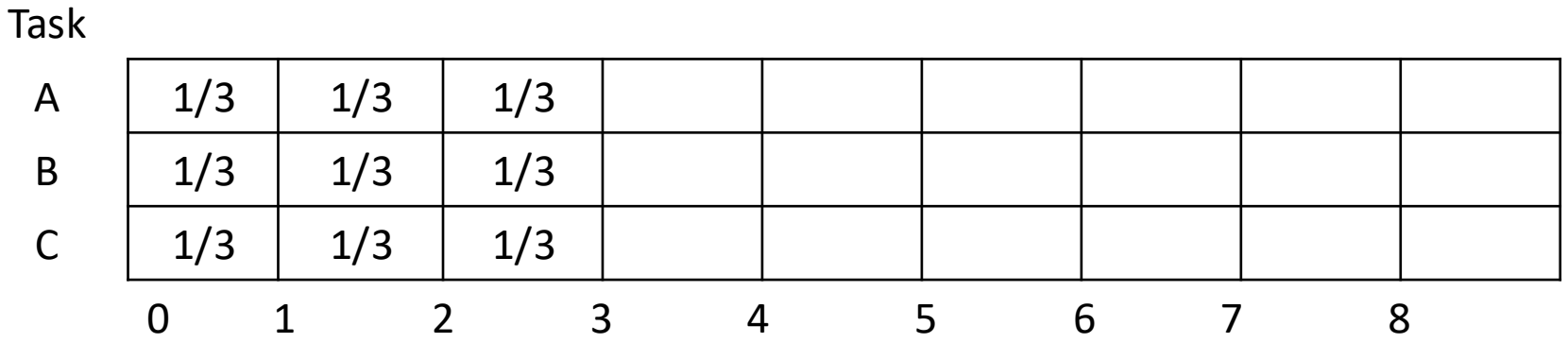
Completely Fair Scheduler

- "Fairness" – making sure that each task gets its fair share of the CPU
  - This is not always achievable
  - "Fairness, it turns out, is enough to solve many CPU-scheduling problems."

This is what CFS is trying to replicate as if we are on an "Ideal Processor"

- Here is an example of fairness:
  - Within some "slice" of time, each task gets an equal proportion of the processor

TASK	Run Time
A	1
B	5
C	2



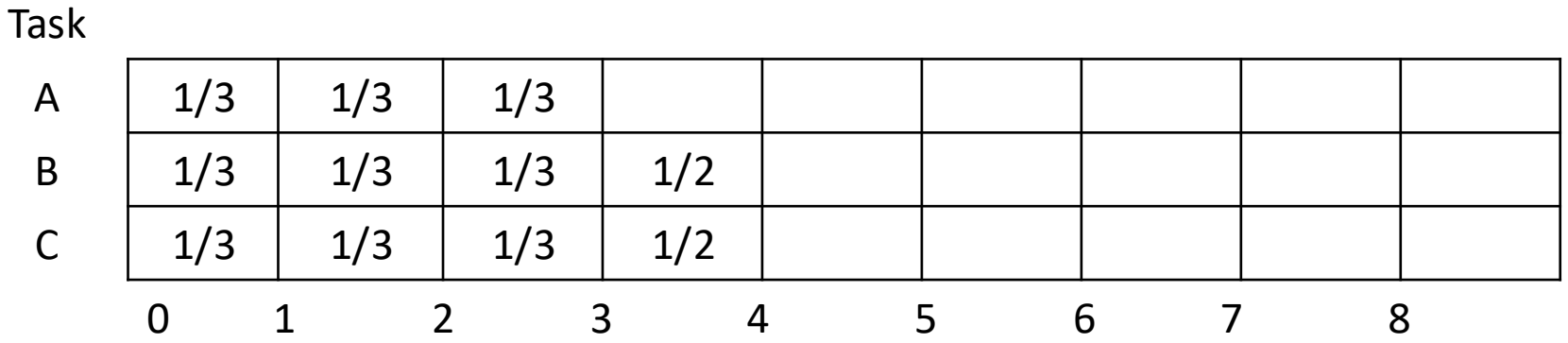
Completely Fair Scheduler

- "Fairness" – making sure that each task gets its fair share of the CPU
  - This is not always achievable
  - "Fairness, it turns out, is enough to solve many CPU-scheduling problems."

This is what CFS is trying to replicate as if we are on an "Ideal Processor"

- Here is an example of fairness:
  - Within some "slice" of time, each task gets an equal proportion of the processor

TASK	Run Time
A	1
B	5
C	2



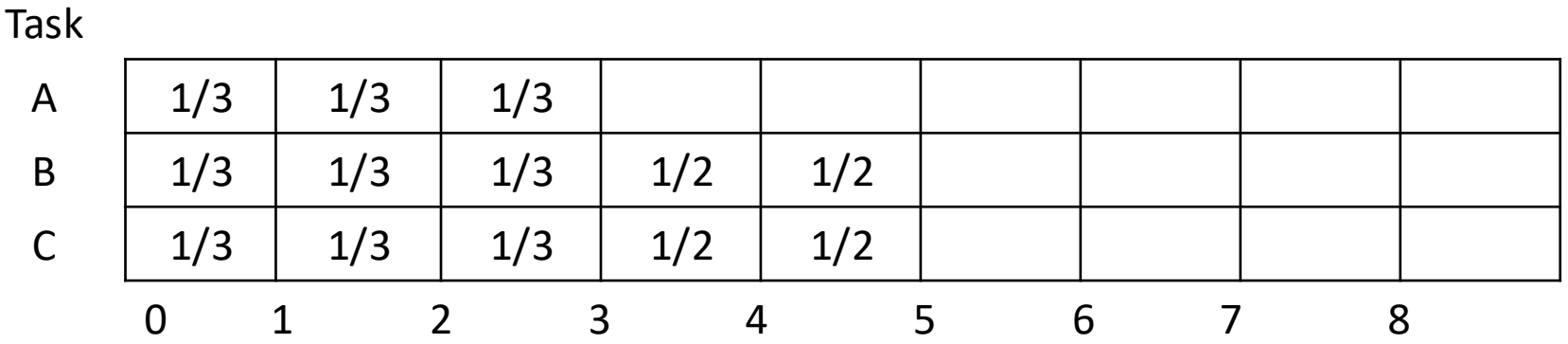
Completely Fair Scheduler

- "Fairness" – making sure that each task gets its fair share of the CPU
  - This is not always achievable
  - "Fairness, it turns out, is enough to solve many CPU-scheduling problems."

This is what CFS is trying to replicate as if we are on an "Ideal Processor"

- Here is an example of fairness:
  - Within some "slice" of time, each task gets an equal proportion of the processor

TASK	Run Time
A	1
B	5
C	2



Completely Fair Scheduler

- "Fairness" – making sure that each task gets its fair share of the CPU
  - This is not always achievable
  - "Fairness, it turns out, is enough to solve many CPU-scheduling problems."

This is what CFS is trying to replicate as if we are on an "Ideal Processor"

- Here is an example of fairness:
  - Within some "slice" of time, each task gets an equal proportion of the processor

TASK	Run Time
A	1
B	5
C	2

Task									
A	1/3	1/3	1/3						
B	1/3	1/3	1/3	1/2	1/2	1	1	1	
C	1/3	1/3	1/3	1/2	1/2				
	0	1	2	3	4	5	6	7	8

CFS – Reality

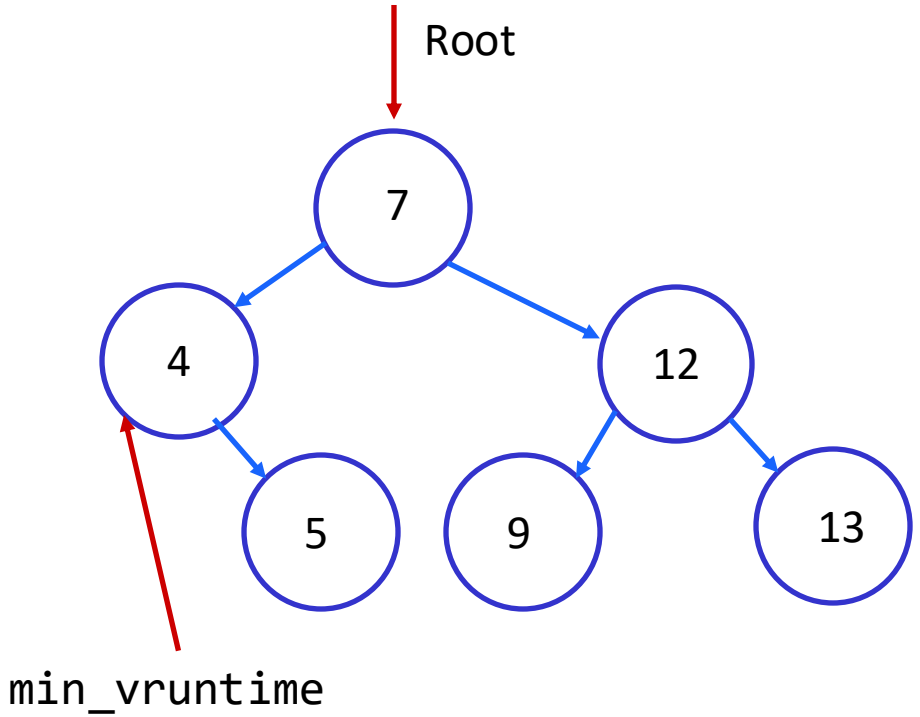
- In reality, many things prevent us from having a “perfect multi-tasking processor”
  - Time to context switch
  - Time for the scheduler run
  - Time spent running other things in the kernel that don't really belong to a single task
  - **Task may not be pre-emptible sometimes and we need to wait for the task to become pre-emptible**
  - Etc.

CFS – Implementation

- CFS maintains a current count for "how long has a task run" called `vruntime`
- The runtimes of all tasks are stored by the scheduler
- Unlike round robin, a thread is not run for a fixed amount of time
  - Run a task till there is some thing with a lower `vruntime`
  - To avoid constantly switching back and forth between two tasks there is a minimum "granularity" (~2.25 milliseconds iirc)

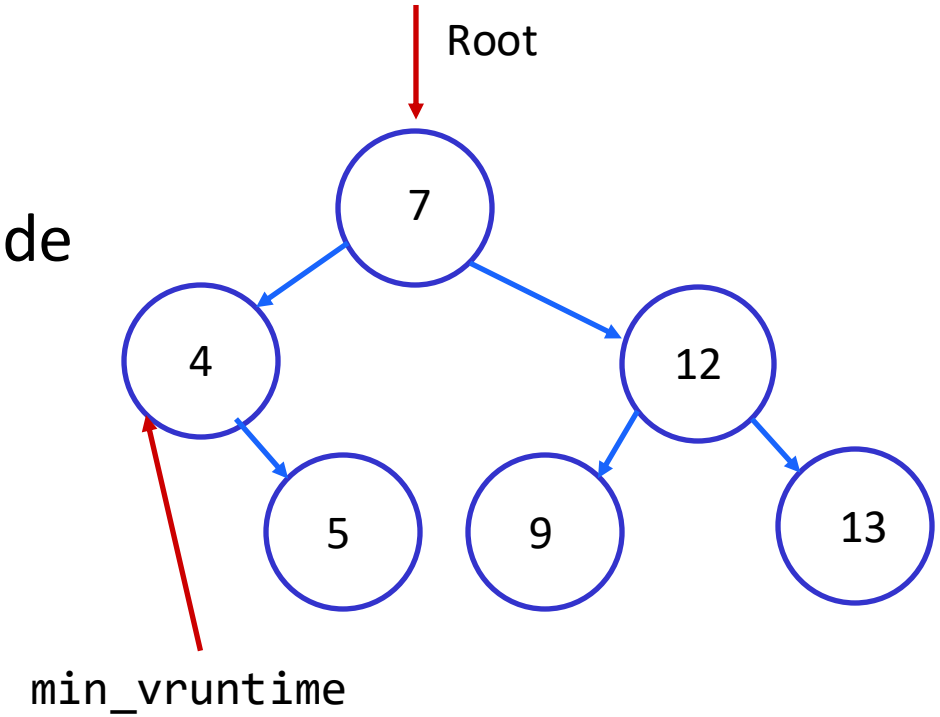
CFS – Implementation Details

- CFS maintains a current count for "how long has a task run" called `vruntime`
- The runtimes of all tasks are stored by the scheduler inside of a Red-Black Tree
  - Red-Black Tree is a self balancing binary tree
  - Sorted on the `vruntime` for each task
  - Smallest `vruntime` task is the leftmost node
- Adding a node is O(log N) operation
- Pointer to leftmost node is maintained, so looking up is O(1)



CFS – Implementation Details

- CFS maintains a current count for "how long has a task run" called `vruntime`
- On each scheduler "tick" the processor compares the current running task to the leftmost task
- If the `min_vruntime` is less than the current node (and granularity has passed) then start running the minimum task



CFS – New Tasks

- New tasks haven't run on the CPU, so are their `vruntime` 0 when they are created?
  - No, instead new tasks start with their `vruntime` equal to the `min_vruntime`
  - This way fairness is maintained between newer and older tasks.

CFS – I/O Bound Tasks

- CFS will also maintain whether a job is sleeping or blocked
  - Won't schedule to run those tasks, stores them in a separate structure
- CFS handles I/O bound tasks pretty well :)
- Tasks with many I/O bursts will have small usage of CPU, so they also have a low `vruntime` and have higher priority

Lecture Outline

- Review
- Preemptive Algorithms
  - Round Robin
  - Round Robin Variants
  - Completely Fair Scheduling (CFS)
- **"Nice" Threads**
- **Extra: Earliest Eligible Virtual Deadline First, Interactive Tasks, Etc.**

nice

- Linux has a way to set priority with a nice value
  - Each process starts with a nice value of 0
  - Nice is clamped to [-20, 19]

- Higher nice score → lower priority
- Lower nice score → higher priority

- The higher your nice score, the “nicer” you are
  (the task runs less often thus letting other tasks run instead of it)

CFS – Vruntime

- CFS uses `vruntime` as the dominant metric
  - **V** stands for **virtual** (e.g. not real runtime)
- You may have thought:
  - `curr_task→runtime += time_running`
  - This is false
- `vruntime` takes other things (like nice scores) into consideration
  - `curr_task→vruntime += (time_running * weight_based_on_nice)`
- CFS takes other things into consideration that make it more complex :)

Earliest Eligible Virtual Deadline First (EEVDF)

- New Linux scheduler!
  - Replaced CFS about two years ago (April 2024)
  - Still aims for fairness, just with some different metrics

- Utilizes a new concept called “lag” (in addition to `vruntime`)
  - A measurement for how much time a task is “owed” if it did not get its fair share of time
  - Tasks that took more CPU time than its fair share have negative “lag”
    - Will not be considered “eligible” so will not be run until lag >= 0
    - Sleeping / blocked tasks will not get free lag increases

Earliest Eligible Virtual Deadline First (EEVDF)

- Some Lore:

Wrote the CFS Schedular

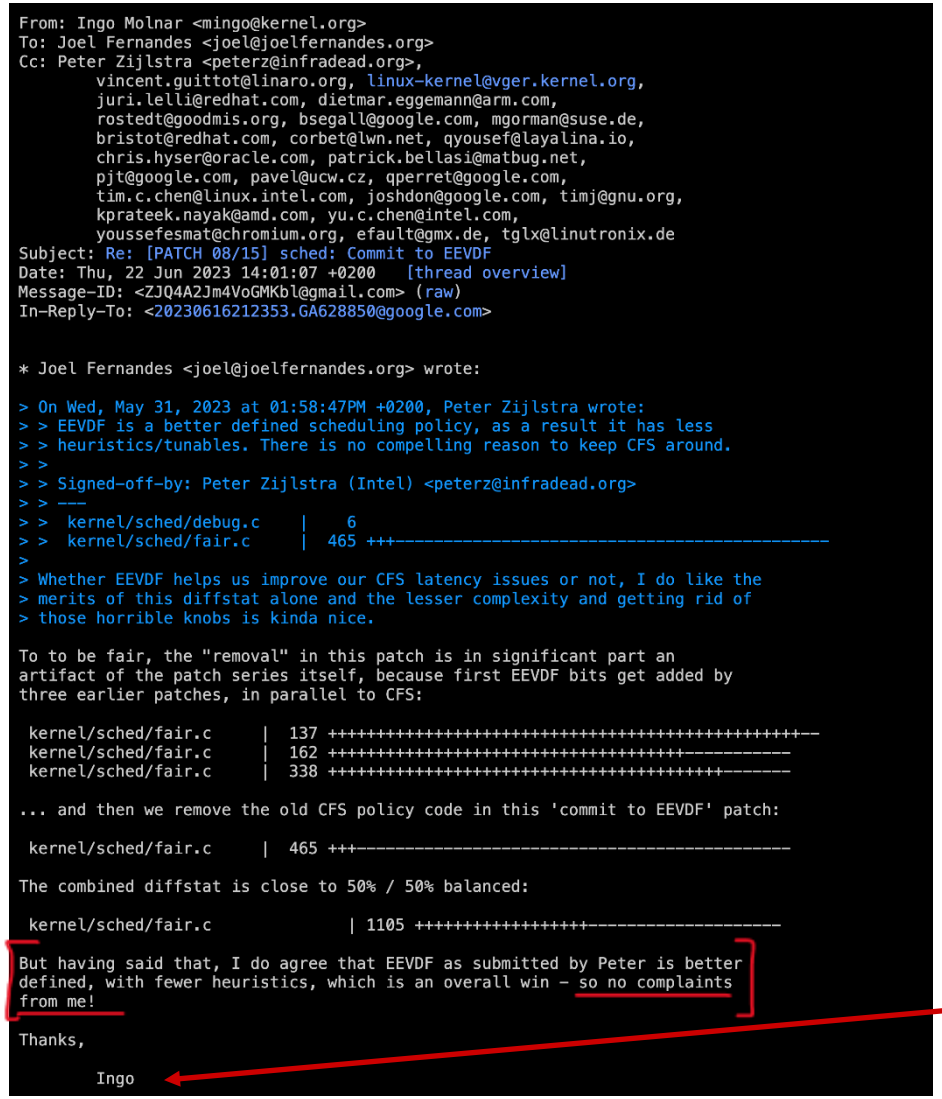
```
From: Peter Zijlstra <peterz@infradead.org>
To: mingo@kernel.org, vincent.guittot@linaro.org
Cc: linux-kernel@vger.kernel.org, peterz@infradead.org,
        juri.lelli@redhat.com, dietmar.eggemann@arm.com,
        rostedt@goodmis.org, bsegall@google.com, mgorman@suse.de,
        bristot@redhat.com, corbet@lwn.net, qyousef@layalina.io,
        chris.hyser@oracle.com, patrick.bellasi@matbug.net,
        pjt@google.com, pavel@ucw.cz, qperret@google.com,
        tim.c.chen@linux.intel.com, joshdon@google.com, timj@gnu.org,
        kprateek.nayak@amd.com, yu.c.chen@intel.com,
        youssefesmat@chromium.org, joel@joelfernandes.org, efault@gmx.de,
        tglx@linutronix.de
Subject: [PATCH 08/15] sched: Commit to EEVDF
Date: Wed, 31 May 2023 13:58:47 +0200   [thread overview]
Message-ID: <20230531124604.137187212@infradead.org> (raw)
In-Reply-To: 20230531115839.089944915@infradead.org

EEVDF is a better defined scheduling policy, as a result it has less
heuristics/tunables. There is no compelling reason to keep CFS around.

Signed-off-by: Peter Zijlstra (Intel) <peterz@infradead.org>
```

Earliest Eligible Virtual Deadline First (EEVDF)

- Some Lore:



From: Ingo Molnar <mingo@kernel.org>
To: Joel Fernandes <joel@joelfernandes.org>
Cc: Peter Zijlstra <peterz@infradead.org>,
vincent.guittot@linaro.org, linux-kernel@vger.kernel.org,
juri.lelli@redhat.com, dietmar.eggemann@arm.com,
rostedt@goodmis.org, bsegall@google.com, mgorman@suse.de,
bristot@redhat.com, corbet@lwn.net, qyousef@layalina.io,
chris.hyser@oracle.com, patrick.bellasi@matbug.net,
pjt@google.com, pavel@ucw.cz, qperret@google.com,
tim.c.chen@linux.intel.com, joshdon@google.com, timj@gnu.org,
kprateek.nayak@amd.com, yu.c.chen@intel.com,
youssefesmat@chromium.org, efault@gmx.de, tglx@linutronix.de
Subject: Re: [PATCH 08/15] sched: Commit to EEVDF
Date: Thu, 22 Jun 2023 14:01:07 +0200 [thread overview]
Message-ID: <ZJQ4A2Jm4VoGMKbl@gmail.com> (raw)
In-Reply-To: <20230616212353.GA628850@google.com>

* Joel Fernandes <joel@joelfernandes.org> wrote:

> On Wed, May 31, 2023 at 01:58:47PM +0200, Peter Zijlstra wrote:
> > EEVDF is a better defined scheduling policy, as a result it has less
> > heuristics/tunables. There is no compelling reason to keep CFS around.
> >
> > Signed-off-by: Peter Zijlstra (Intel) <peterz@infradead.org>
> > ---
> > kernel/sched/debug.c | 6
> > kernel/sched/fair.c | 465 +++--
>
> Whether EEVDF helps us improve our CFS latency issues or not, I do like the
> merits of this diffstat alone and the lesser complexity and getting rid of
> those horrible knobs is kinda nice.

To to be fair, the "removal" in this patch is in significant part an
artifact of the patch series itself, because first EEVDF bits get added by
three earlier patches, in parallel to CFS:

kernel/sched/fair.c | 137 ++--
kernel/sched/fair.c | 162 ++
kernel/sched/fair.c | 338 ++

... and then we remove the old CFS policy code in this 'commit to EEVDF' patch:

kernel/sched/fair.c | 465 +++--

The combined diffstat is close to 50% / 50% balanced:

kernel/sched/fair.c | 1105 ++++++++++++++++++++++++++++++++++++++

Earliest Eligible Virtual Deadline First (EEVDF)

- Not going over it due to:
  - Time in lecture, looks like it may be more complex and take longer to explain
  - It is new! Not as much information out there on it
    - I could read the Linux kernel source code, but that takes time :)))))))
    - I invite you to read the 13718 lines of the scheduler implementation. ☺

- Look at these articles from LWN.net if you want to learn more about EEVDF
  - https://lwn.net/Articles/925371/
  - https://lwn.net/Articles/969062/

Why did we talk about this?

- Scheduling is fundamental towards how computer can multi-task
- This is a great example of how “systems” intersects with algorithms :)
- It shows up ‘occasionally’ in the real world :)
  - What really happened on Mars Rover Pathfinder, Mike Jones.

Consideration: Interactive Tasks

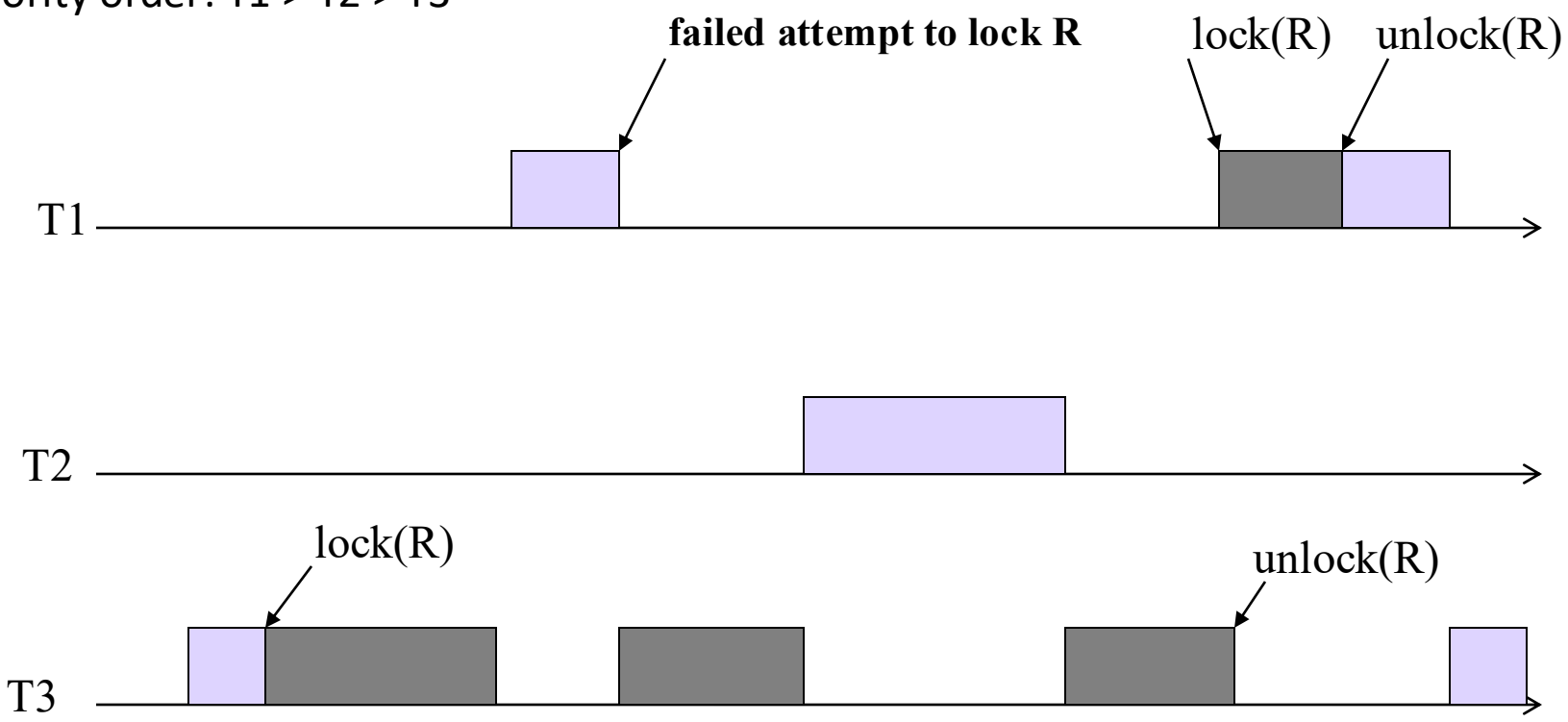
- There is still ongoing work to make schedulers that are better
  - "Better" either in general or to specific situations

- Example: People are already working on EEVDF upgrades. VARD scheduler for SteamOS

  https://youtu.be/xJjZ5tzlHOY?si=lgGNWaQe03qSgCP2&t=1682

Another Issue: The Priority Inversion Problem

Priority order: T1 > T2 > T3



T2 is causing a higher priority task T1 wait!

More

- For those curious, there was a LOT left out

- RTOS (Real Time Operating Systems)
  - For real time applications
  - CRITICAL that data and events meet defined time constraints
  - Different focus in scheduling. Throughput is de-prioritized

- Fair-share scheduling
  - Equal distribution across different users instead of by processes

- Etc.

That's all!

- See you next week for Threads!

Round Robin Practice!

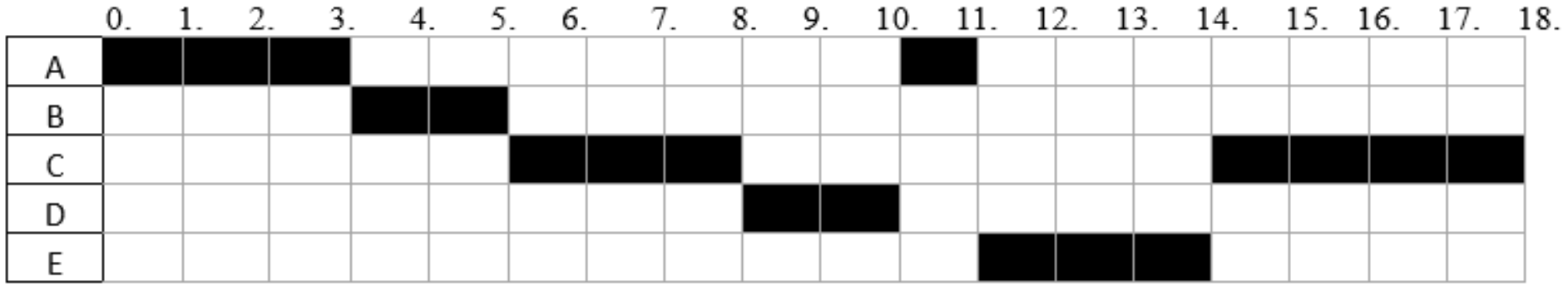
- Let's switch our algorithm to round-robin with a time quantum of 3 time units

Process	Arrival Time	Finishing Time
A	0	4
B	1	6
C	2	13
D	3	15
E	4	18

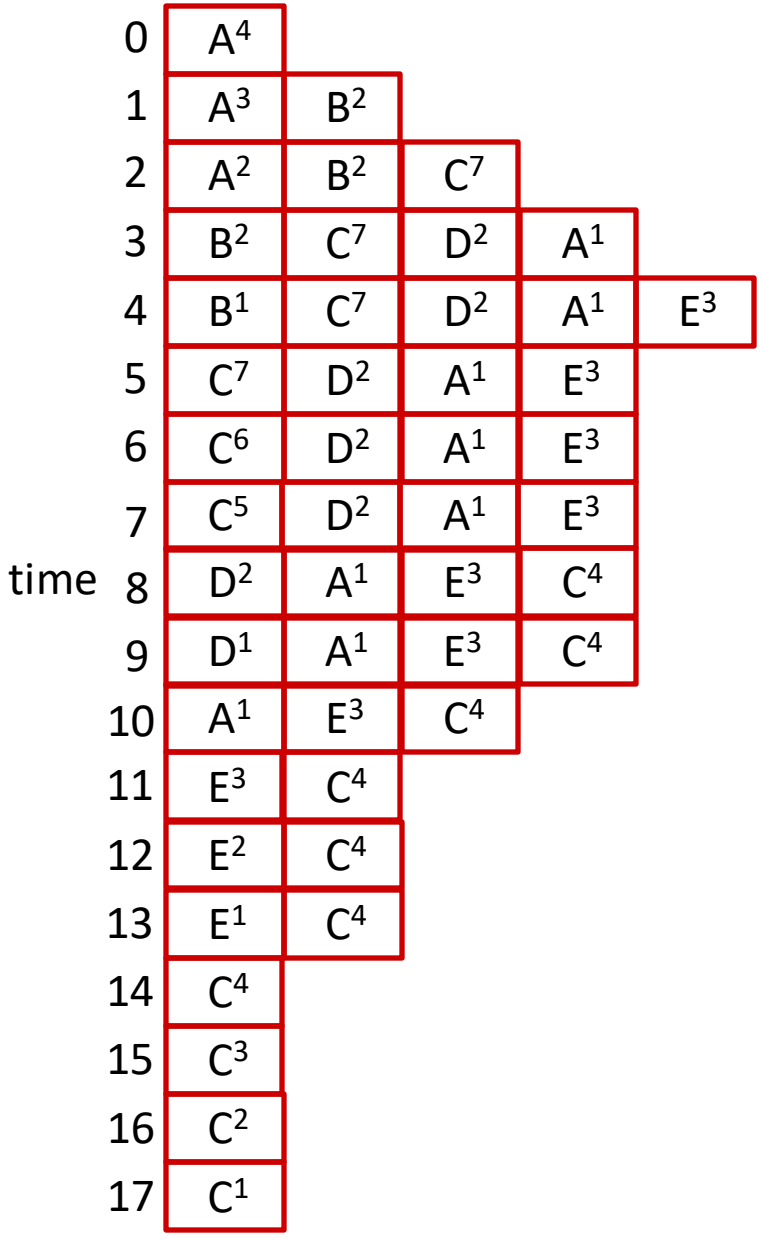
→

Process	Arrival Time	Length
A	0	4
B	1	2
C	2	7
D	3	2
E	4	3

- You can assume:
  - Context switching and running the Scheduler are instantaneous
  - If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice

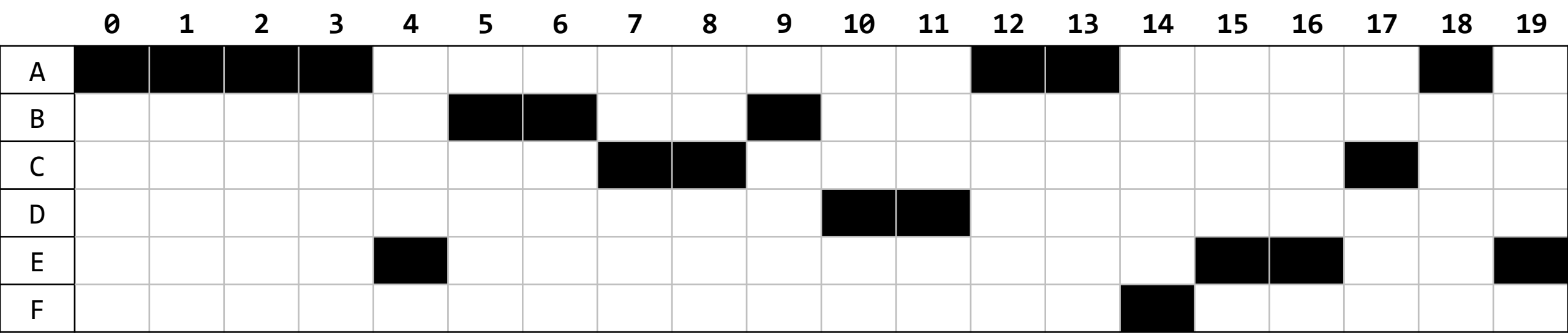


RR Example: Different Visualization



Process	Arrival Time	Length
A	0	4
B	1	2
C	2	7
D	3	2
E	4	3

= 1 unit of time

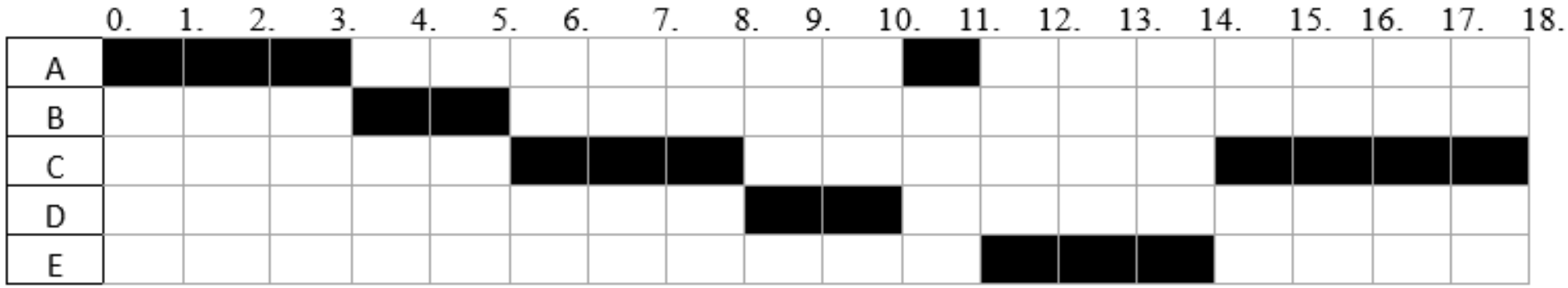

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
A
B
C
D
E
F

Poll Everywhere

pollev.com/tqm

- What is the average wait time?
  - A: 0 + 7 = 7
  - B: 2
  - C: 3 + 6 = 9
  - D: 5
  - E: 7
  - Average: (7 + 2 + 9 + 5 + 7)/5 = 30/5 = 6

Process	Arrival Time	Length
A	0	4
B	1	2
C	2	7
D	3	2
E	4	3

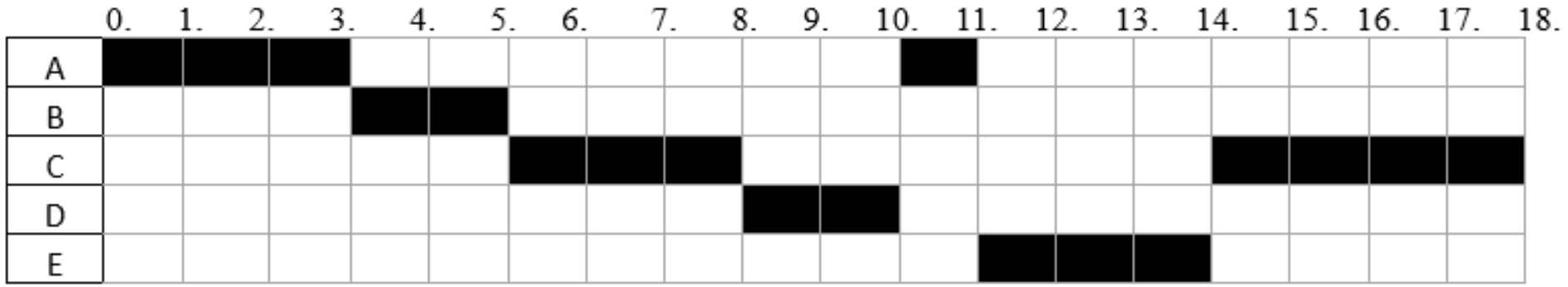


Poll Everywhere

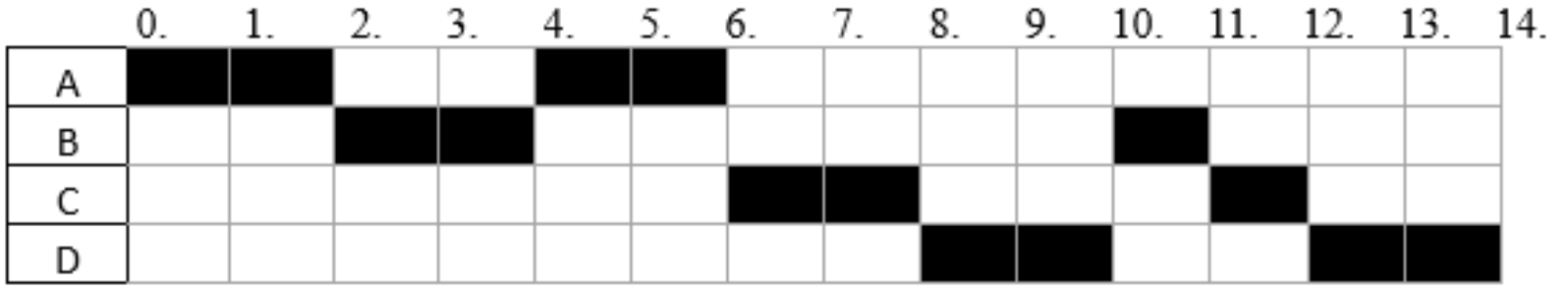
pollev.com/tqm

- What is the average turnaround time?
  - A: 11 – 0 = 11
  - B: 5 – 1 = 4
  - C: 18 – 2 = 16
  - D: 10 – 3 = 7
  - E: 14 – 4 = 10
  - Average: (11 + 4 + 16 + 7 + 10)/5 = 48/5 = 9.6

Process	Arrival Time	Length
A	0	4
B	1	2
C	2	7
D	3	2
E	4	3

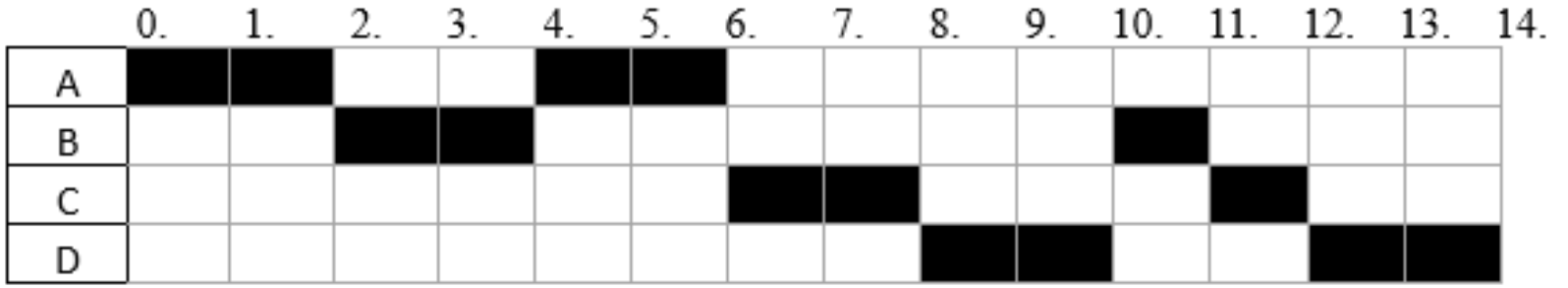


More Round Robin Practice



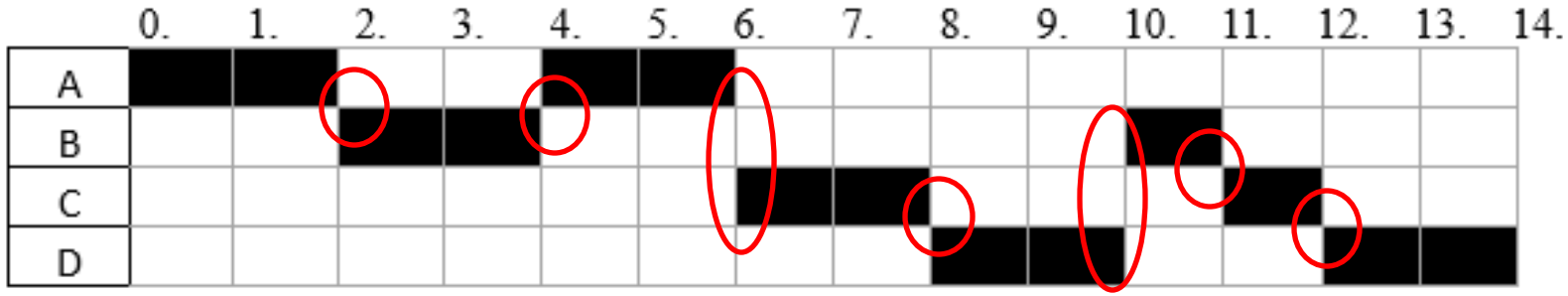
- What is the earliest time that process C could have arrived?
  - If C arrived at time 0, 1, or 2, it would have run at time 4
  - C could have shown up at time 3 and come after A in the queue
  - C showed up at time 3 at earliest

More Round Robin Practice



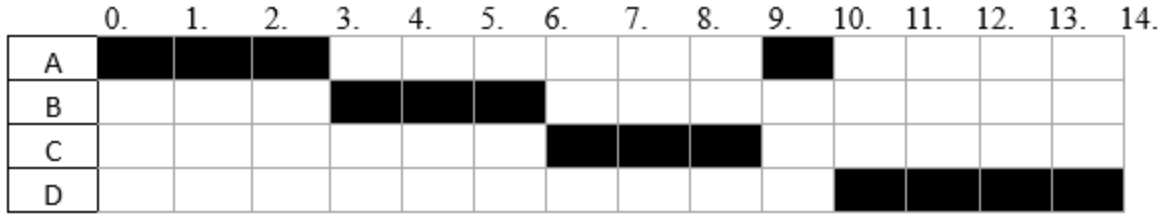
- Which processes are in the ready queue at time 9?
  - D is running, so it is not in the queue
  - A has finished
  - B and C still must finish, so they are in the queue

More Round Robin Practice

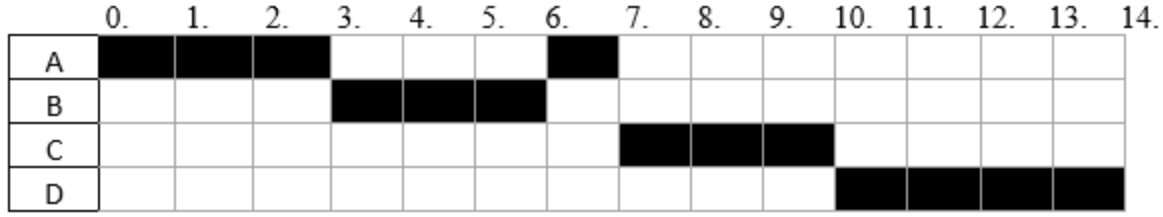


- If this algorithm used a quantum of 3 instead of 2, how many fewer context switches would there be?
  - Currently there are 7 context switches
  - If quantum was 3:

Depends on if C shows up at time 3 or 4



  - Or:



Either way, only 4 context switches, so 3 less than quantum = 2