Advanced Scheduling

Dr. Naser Al Madi

Project survey results

I appreciate your feedback, all of it!

https://docs.google.com/forms/d/1JXplSwXxmkMHn8MJHQ2KVyo3v0teIkOP9snNHCFteN8/edit#responses

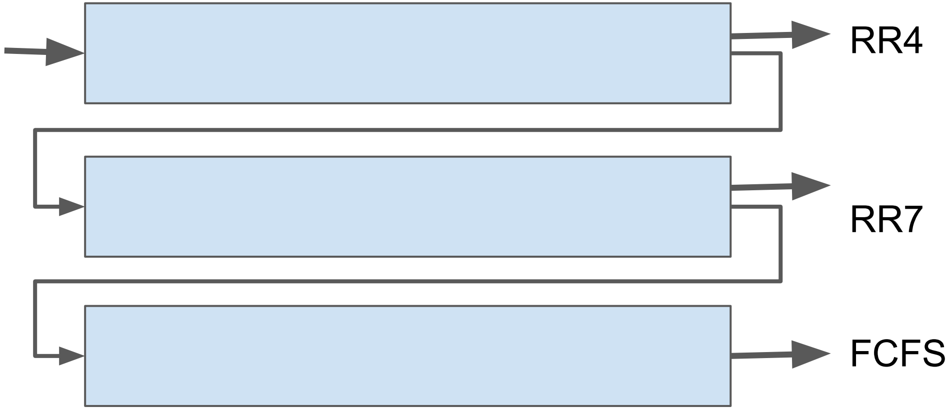
Learning objective

- Multilevel Feedback Queue
- Multi-CPU scheduling
- O(1) Scheduler
- Completely Fair Scheduling
- Project 2

Multilevel Feedback Queue (MLFQ)

Waiting

Process	CPU,I/O,CPU	Arrival	Queue
~~P1~~	~~5, 1, 6, 7~~	~~4~~ ~~28~~	~~1~~ ~~2~~
~~P2~~	~~4, 2, 3~~	~~3~~ ~~10~~	~~1~~
~~P3~~	~~2, 3, 4~~	~~4~~ ~~13~~	~~1~~
~~P4~~	~~5, 1, 2, 7~~	~~14~~ ~~25~~	~~1~~ ~~2~~



P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1	

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Response time – amount of time from when a request was submitted until the first response is produced.

	Response time	Wait time	Turnaround time
P1			
P2			
P3			
P4			

Response time – amount of time from when a request was submitted until the **first response** is produced.

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	first - arrival		
P2			
P3			
P4			

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0		
P2			
P3			
P4			

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1	
0	4	8	10	14	17	21	22	23	25	32	39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0		
P2	1		
P3			
P4			

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1	

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0		
P2	1		
P3	4		
P4			

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1	
0	4	8	10	14	17	21	22	23	25	32	39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

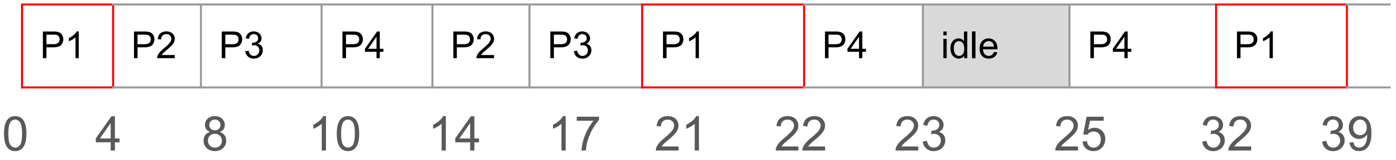
	Response time	Wait time	Turnaround time
P1	0		
P2	1		
P3	4		
P4	3		

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1	
0	4	8	10	14	17	21	22	23	25	32	39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0		
P2	1		
P3	4		
P4	3		



Multilevel Feedback Queue (MLFQ)

Process	**CPU,I/O,CPU**	**Arrival**	**Queue**
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	**Wait time**	**Turnaround time**
P1	0	0 + 17 + 4	
P2	1		
P3	4		
P4	3		

(**6** I/O + **4** waiting in the ready queue)

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	
P2	1	1 +	
P3	4		
P4	3		

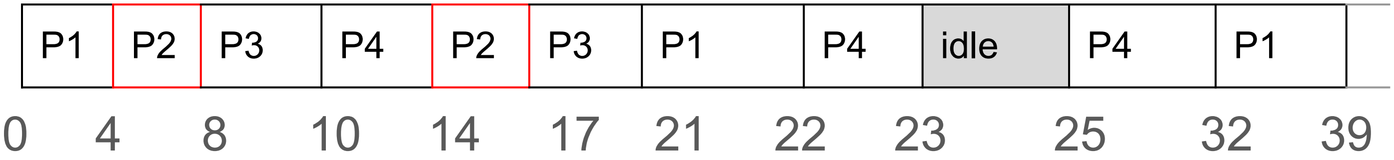
P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	
P2	1	1 + (6 - 2)	
P3	4		
P4	3		



Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	
P2	1	1 + (6 - 2)	
P3	4		
P4	3		

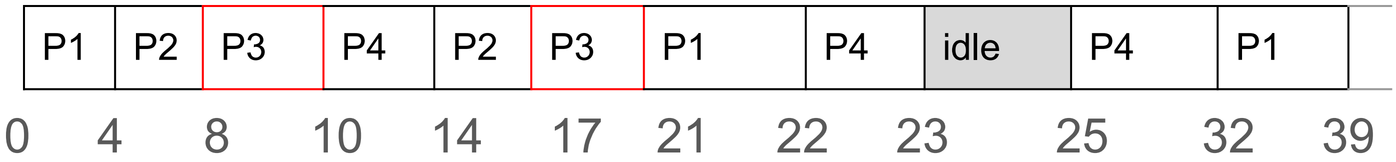
P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	
P2	1	1 + (6 - 2)	
P3	4		
P4	3		



Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	
P2	1	1 + (6 - 2)	
P3	4	4 + (7 - 3)	
P4	3		

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	
P2	1	1 + (6 - 2)	
P3	4	4 + (7 - 3)	
P4	3		

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1	
0	4	8	10	14	17	21	22	23	25	32	39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	
P2	1	1 + (6 - 2)	
P3	4	4 + (7 - 3)	
P4	3		

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	
P2	1	1 + (6 - 2)	
P3	4	4 + (7 - 3)	
P4	3	3 + 8 + 0	

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	21 + 18 = 39
P2	1	1 + (6 - 2)	5 +
P3	4	4 + (7 - 3)	8 +
P4	3	3 + 8 + 0	11 +

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	21 + 18 = 39
P2	1	1 + (6 - 2)	5 + 9 = 14
P3	4	4 + (7 - 3)	8 +
P4	3	3 + 8 + 0	11 +

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	21 + 18 = 39
P2	1	1 + (6 - 2)	5 + 9 = 14
P3	4	4 + (7 - 3)	8 + 9 = 17
P4	3	3 + 8 + 0	11 +

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue (MLFQ)

Process	CPU,I/O,CPU	Arrival	Queue
P1	5, 6, 7	0	1
P2	4, 2, 3	3	1
P3	2, 3, 4	4	1
P4	5, 2, 7	7	1

	Response time	Wait time	Turnaround time
P1	0	0 + 17 + 4	21 + 18 = 39
P2	1	1 + (6 - 2)	5 + 9 = 14
P3	4	4 + (7 - 3)	8 + 9 = 17
P4	3	3 + 8 + 0	11 + 14 = 25

P1	P2	P3	P4	P2	P3	P1	P4	idle	P4	P1

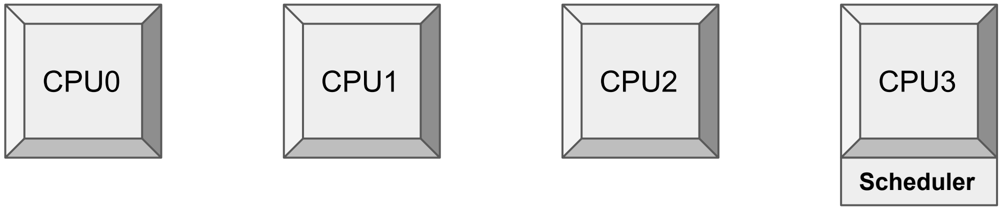
0 4 8 10 14 17 21 22 23 25 32 39

Multilevel Feedback Queue Evaluation

- A process can move between the various queues; aging/decay can be implemented this way
- Multilevel-feedback-queue scheduler is defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to **upgrade** a process
  - method used to determine when to **demote** a process
  - method used to determine which queue a process will enter when that process needs service

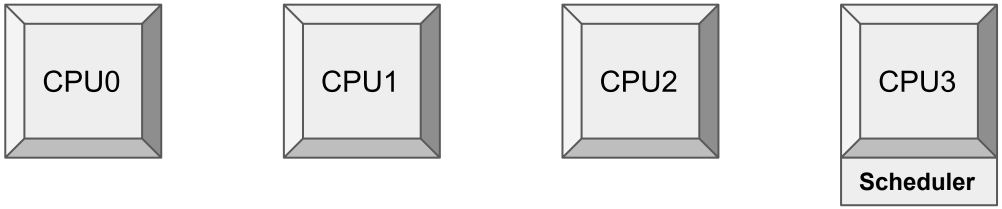
Basic Multi-CPU scheduling

Multi-CPU scheduling (idea 1)



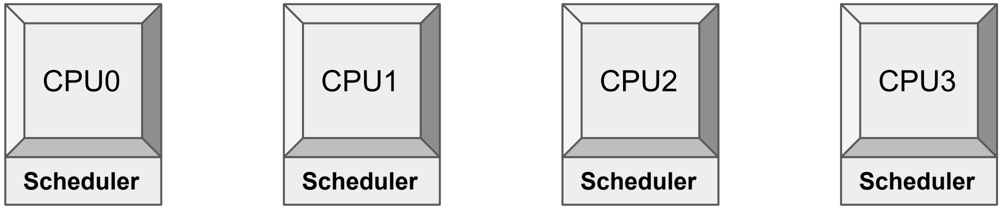
- **Run scheduler on one CPU and schedule processes on others!**
- Limitation?

Multi-CPU scheduling (idea 1)



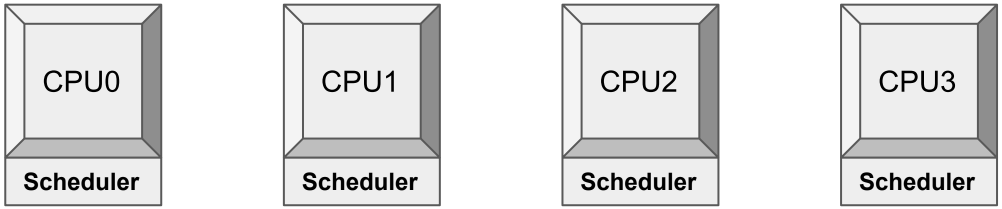
- **Run scheduler on one CPU and schedule processes on others!**
- Limitation: bottle nick where all processors wait for a single processor to make decisions.

Multi-CPU scheduling (idea 2: symmetrical)



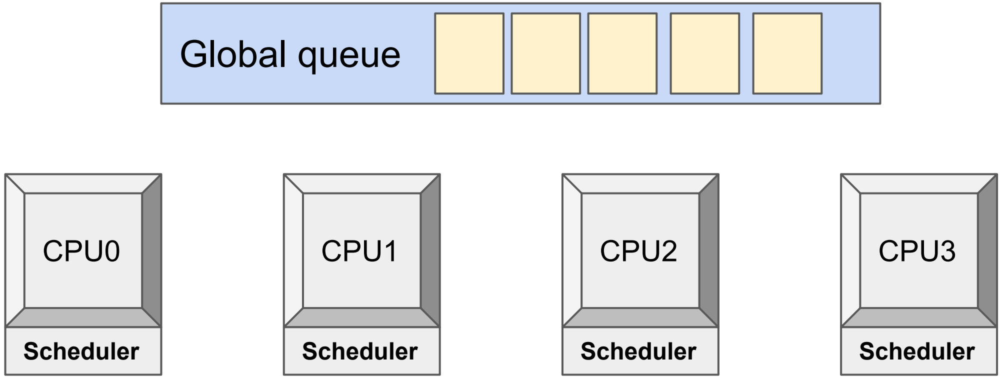
- **Each processor runs its own scheduler** (symmetrical scheduling scheme)

Multi-CPU scheduling (idea 2: symmetrical)



- **Each processor runs its own scheduler** (symmetrical scheduling scheme)
- Come in two variants:
    - Global queue
    - Local queues

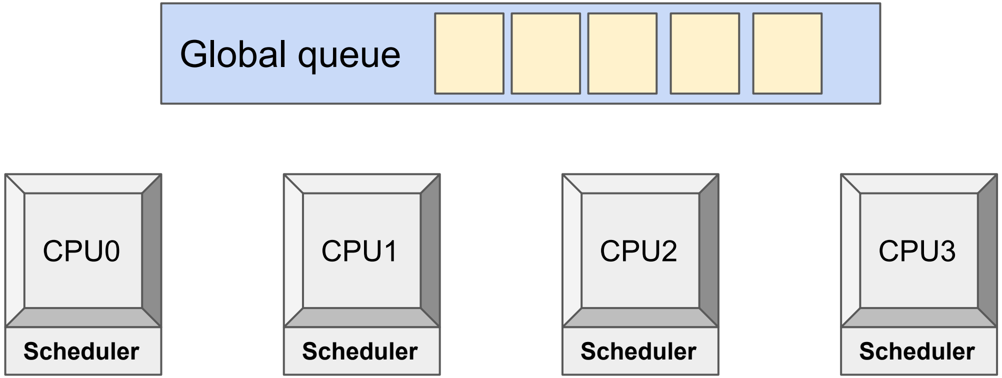
Symmetrical scheduling with global queue



Used in:
Linux 2.4
Xv6

- Advantage:
  - Good CPU utilization.
  - Fairness to all processes.
- Disadvantage:
  - The queue is the bottle nick (synched between processors).
  - Scheduler needs to be really light, since it is synched.
  - No Processor affinity (choosing exact CPU for process).

Symmetrical scheduling with global queue



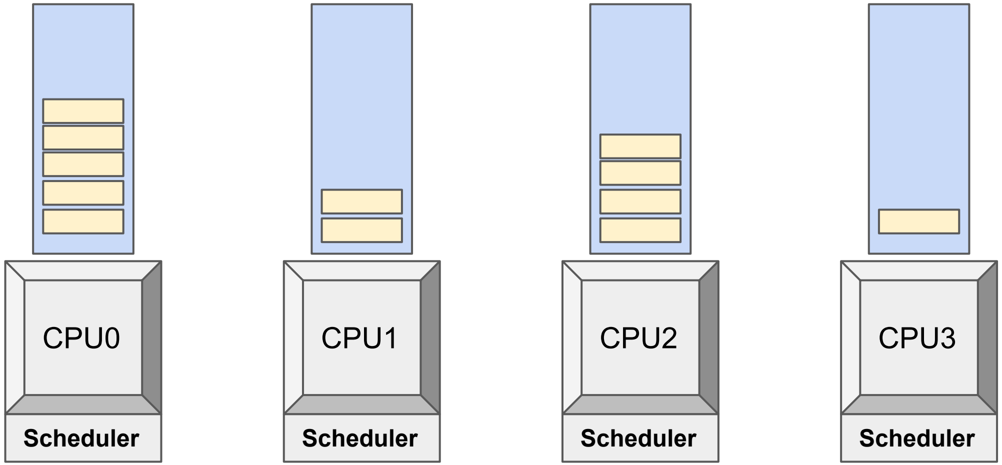
Used in:
Linux 2.4
Xv6

- Advantage:
  - Good CPU utilization.
  - Fairness to all processes.
- Disadvantage:
  - The queue is the bottle nick (synched between processors).
  - Scheduler needs to be really light, since it is synched.
  - No Processor affinity (choosing exact CPU for process).

Biggest problem:

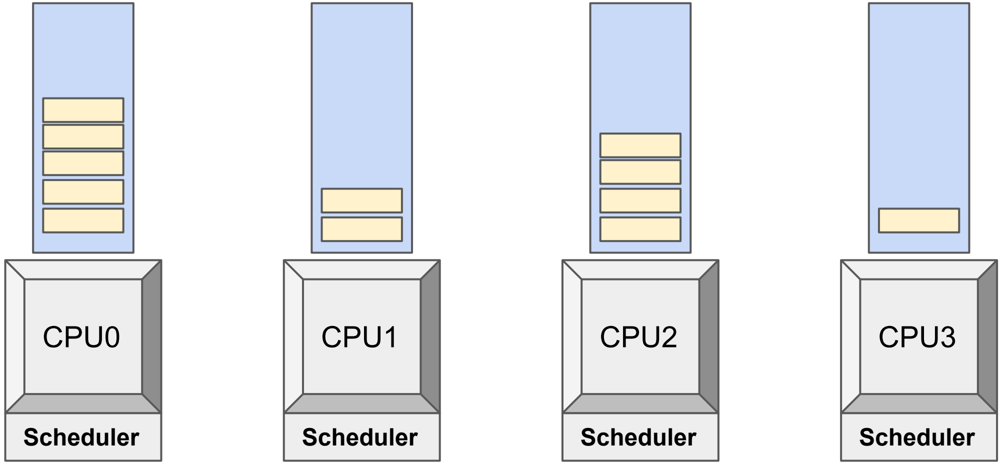
This scheme is not scalable. Synchronization means serialization, defeating the purpose of having multiple CPUs.

Symmetrical scheduling with local queue



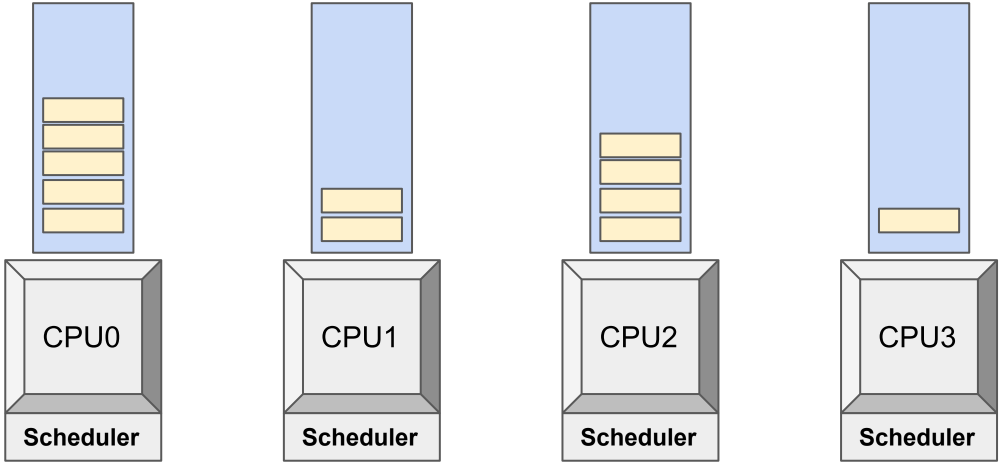
- Advantage:
  - Scalable (no need for synchronization).
  - Locality (TLB and registers)
- Disadvantage:
  - ?

Symmetrical scheduling with local queue



- Advantage:
  - Scalable (no need for synchronization).
  - Locality (TLB and registers)
- Disadvantage:
  - No fairness (load balancing problems) ☹️

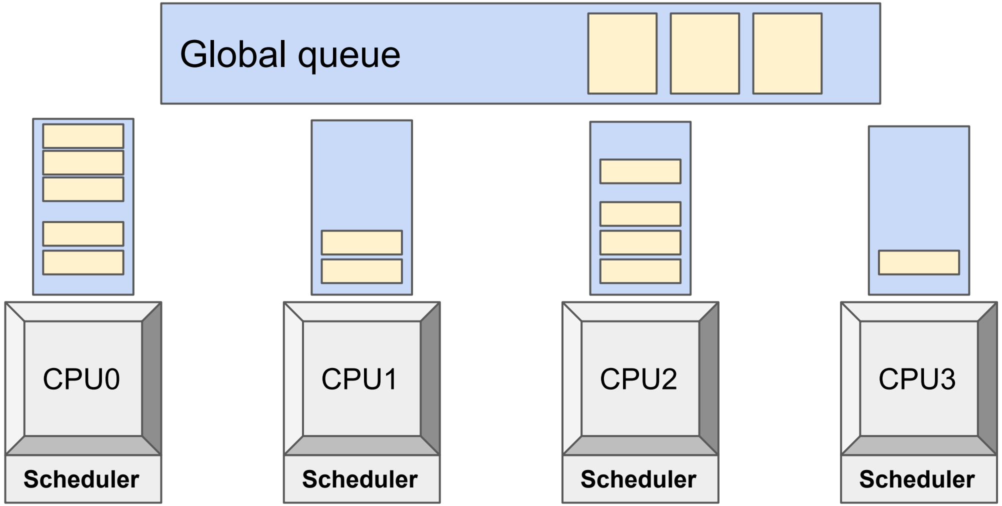
Symmetrical scheduling with <u>local queue</u>



- Advantage:
  - Scalable (no need for synchronization).
  - Locality (TLB and registers)
- Disadvantage:
  - No fairness ☹

The CPU on which a process runs is selected statically before it starts.

<u>Hybrid</u> Symmetrical scheduling



Used in:
Linux 2.6

- Advantage:
  - Global queue ensures load balancing.
  - Local queues ensure locality.

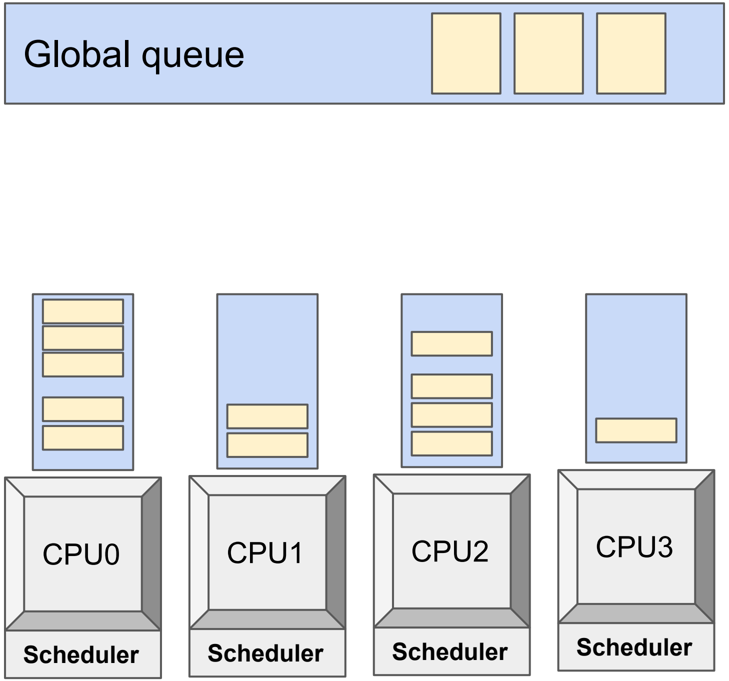
The CPU on which a process runs is selected statically before it starts.

Load balancing

- Push migration:
  - Periodically check processors loads.
  - Add work from global queue to free CPUs.
  - Redistribute all jobs among CPUs.

- Pull migration:
  - Idle processors pull tasks from busy processors.

- Migration is expensive:
  - CPU memory like TLB needs to be repopulated.
  - Migration takes time.



Modern scheduling

Evolution

Extra reading:

Understanding the Linux Kernel, Third Edition 3rd Edition by Daniel P. Bovet

Scheduling Algorithm	**Operating System**
O(n)	Linux 2.4 to 2.6
O(1)	Linux 2.6 to 2.6.22
CFS	Linux 2.6.23 onwards (2007 - 2016)

Process Classification in Linux

- Real time
  - Strict deadlines.
  - Should never be blocked by a low priority process.
- Normal Processes
  - Interactive
    - Constantly interact with their users.
    - When input is received, the process must wake up quickly (with small delay).
  - Batch
    - Do not require any user interaction, often run in the background.

Process Classification in Linux

- Real time
  - Strict deadlines.
  - Should never be blocked by a low priority process.
- Normal Processes
  - Interactive
    - Constantly interact with their users.
    - When input is received, the process must wake up quickly (with small delay).
  - Batch
    - Do not require any user interaction, often run in the background.

A real time process is always a real time process

Process Classification in Linux

- Real time
  - Strict deadlines.
  - Should never be blocked by a low priority process.
- Normal Processes
  - Interactive
    - Constantly interact with their users.
    - When input is received, the process must wake up quickly (with small delay).
  - Batch
    - Do not require any user interaction, often run in the background.

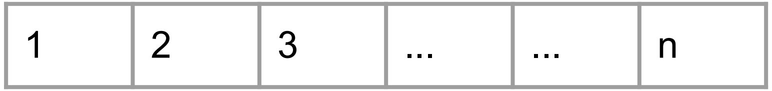
A normal process could act as an interactive process and a batch process interchangeably.

Linux determines type based on **heuristics**

O(n) Scheduler

- At every context switch
  - Scan the list of runnable processes
  - Compute priorities
  - Select the best process to run
- O(n) performance = not scalable
  - Scalability issue observed with JVM creating many tasks
- Used a global run-queue
  - Synchronization needed (again not scalable)

Queue of Ready Processes



O(1) Scheduler

- Constant time required to pick the next process to execute.
  - Easily scales to large number of processes.
- Processes divided into 2 types:
  - **Real time**: priorities from 0 to 99.
  - **Normal processes**: priorities from 100 to 139
    - interactive
    - batch

O(1) Scheduler

- Uses dynamic priority (heuristic)
- Uses variable quantum for each process (heuristic)
- All operations done in constant time
- Combines priority, multilevel feedback queues, round robin, and heuristics to determine priority and quantum.

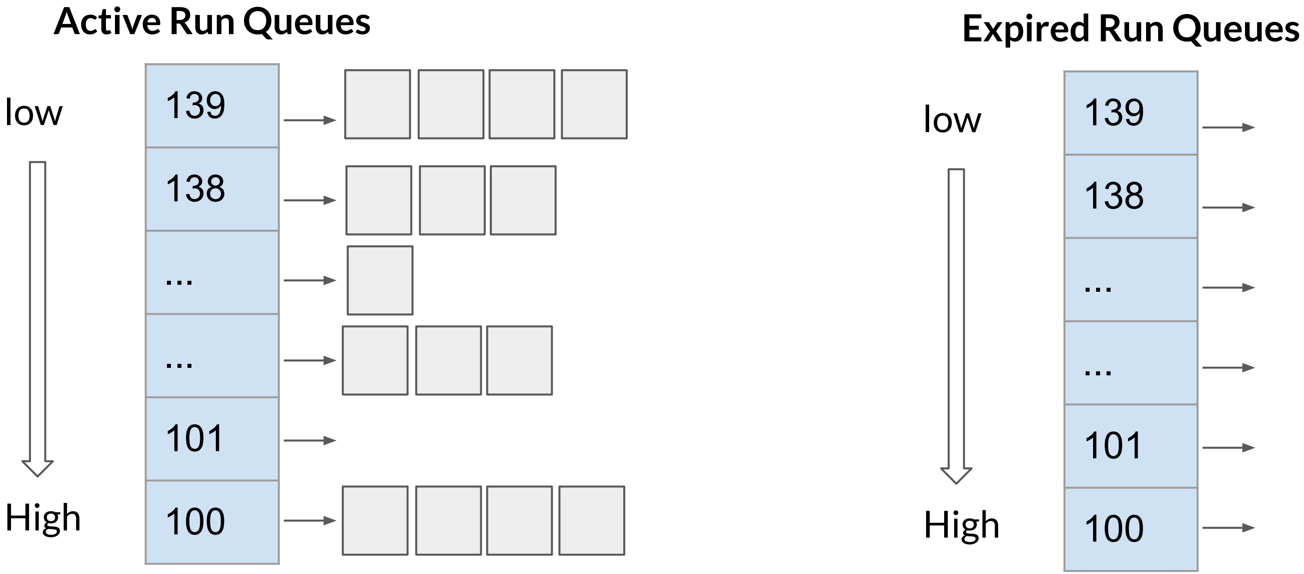
O(1) Scheduler

- Constant time required to pick the next process to execute.
    - Easily scales to large number of processes.
- Processes divided into 2 types:
    - **Real time**: priorities from 0 to 99.
    - **Normal processes**: priorities from 100 to 139
        - interactive
        - batch

	Priority
low	139
	...
	100
	99
	...
High	0

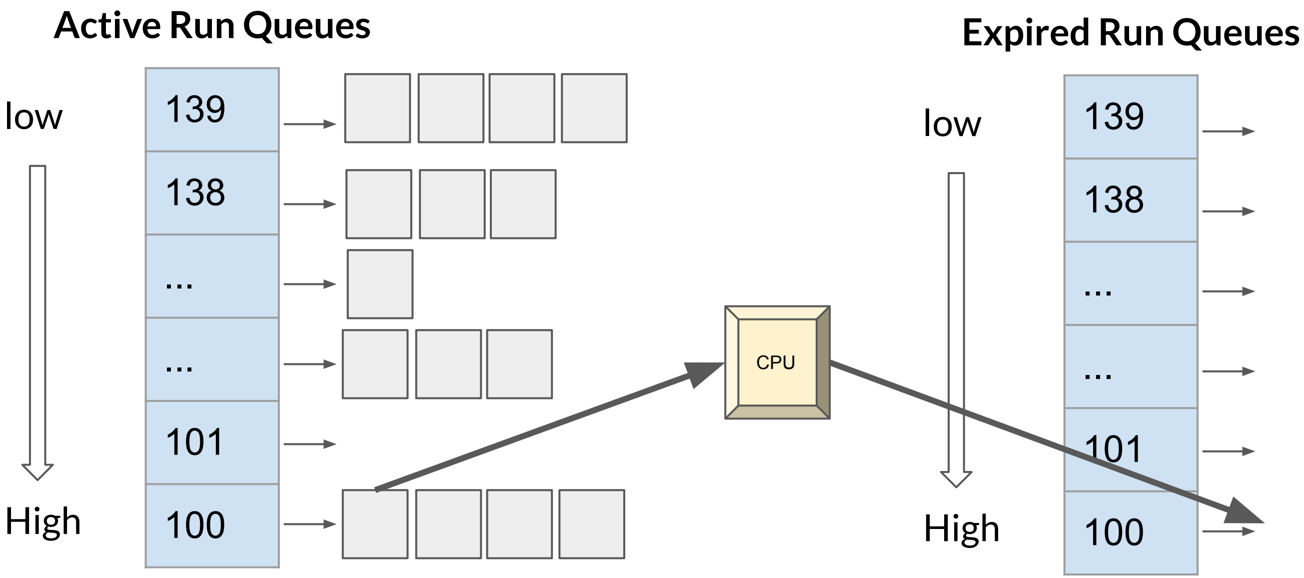
Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



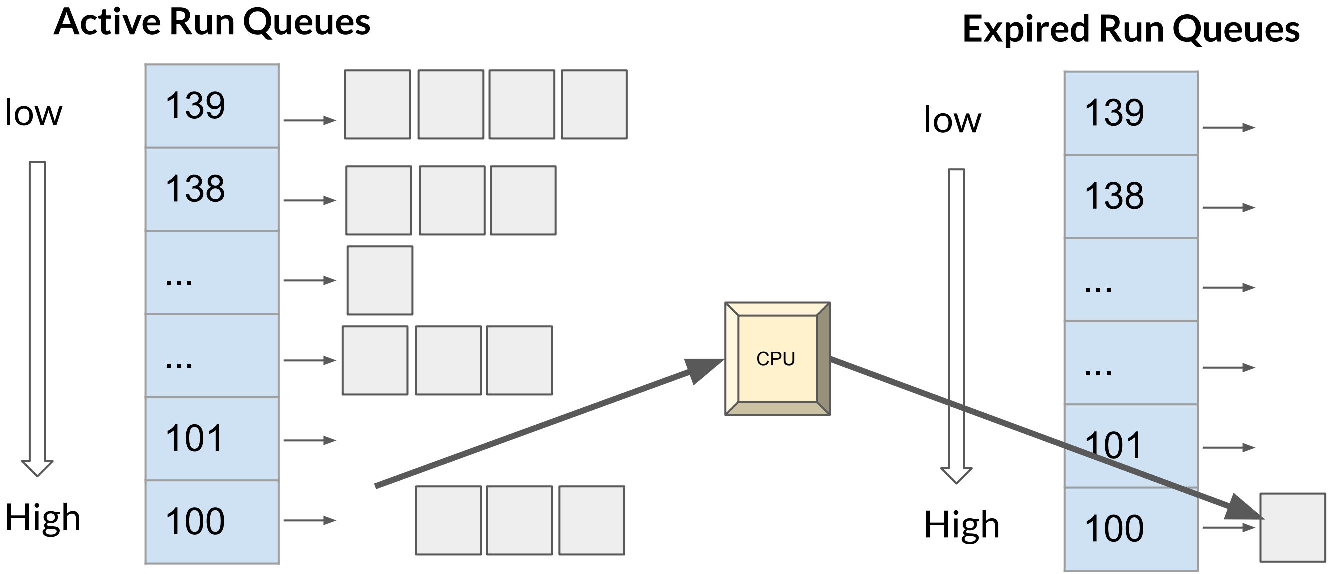
Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



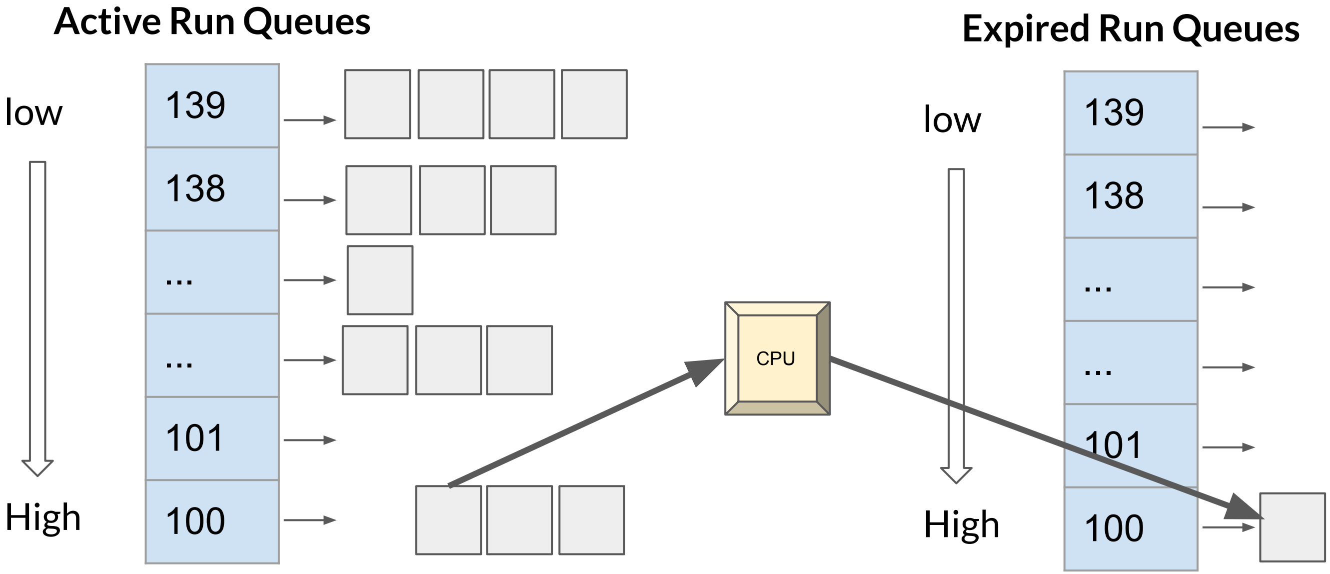
Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



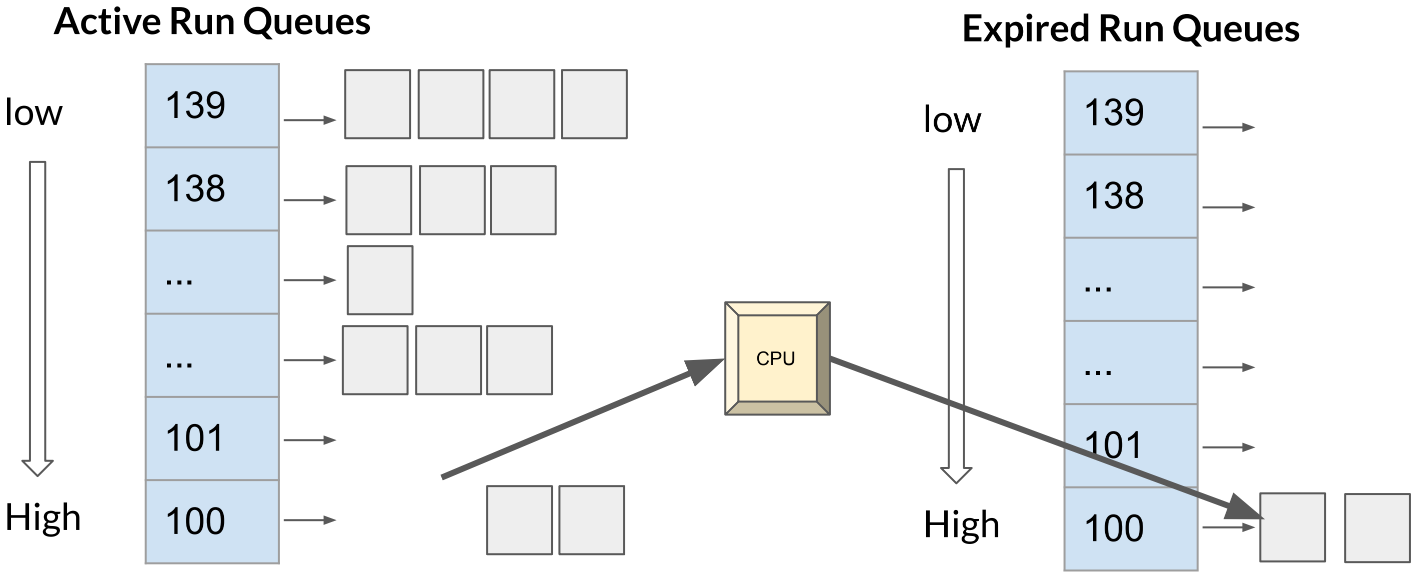
Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



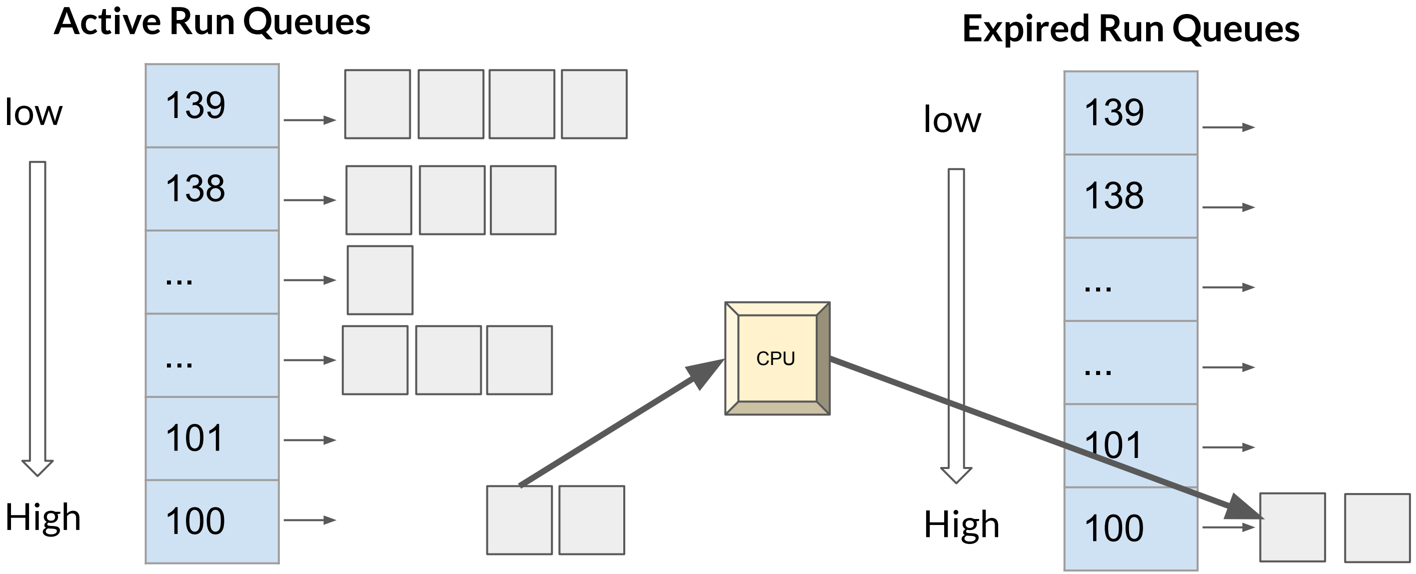
Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



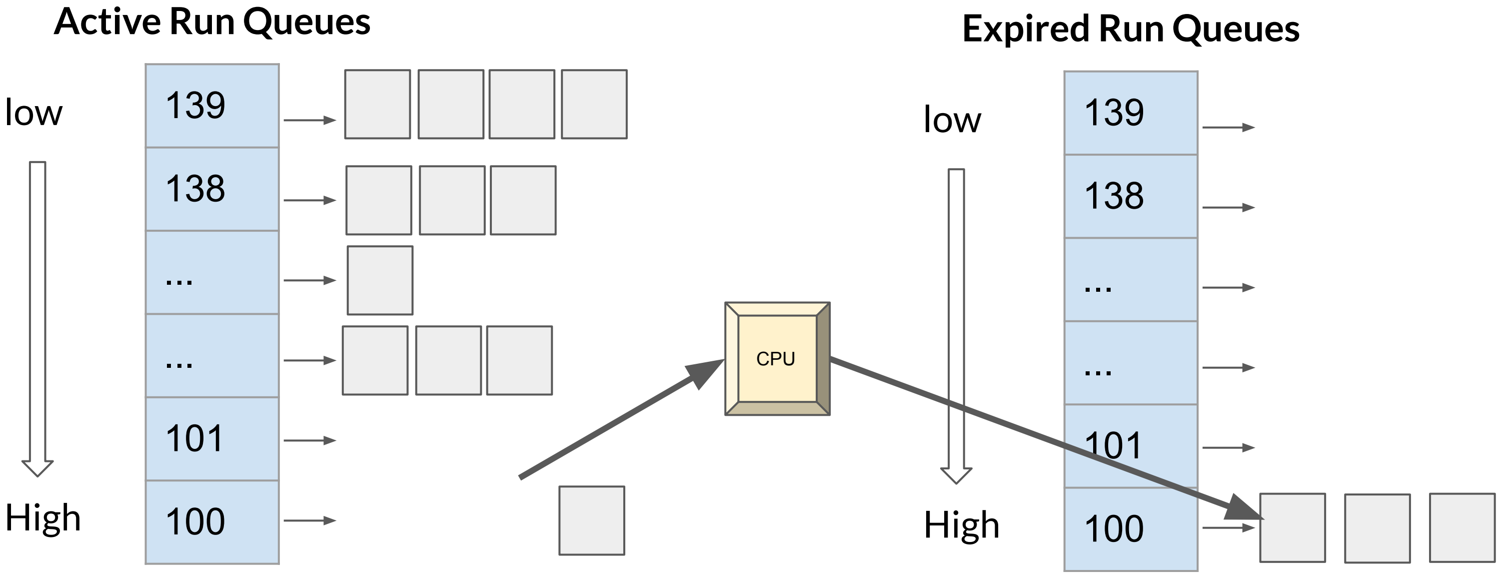
Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



Scheduling Normal Processes

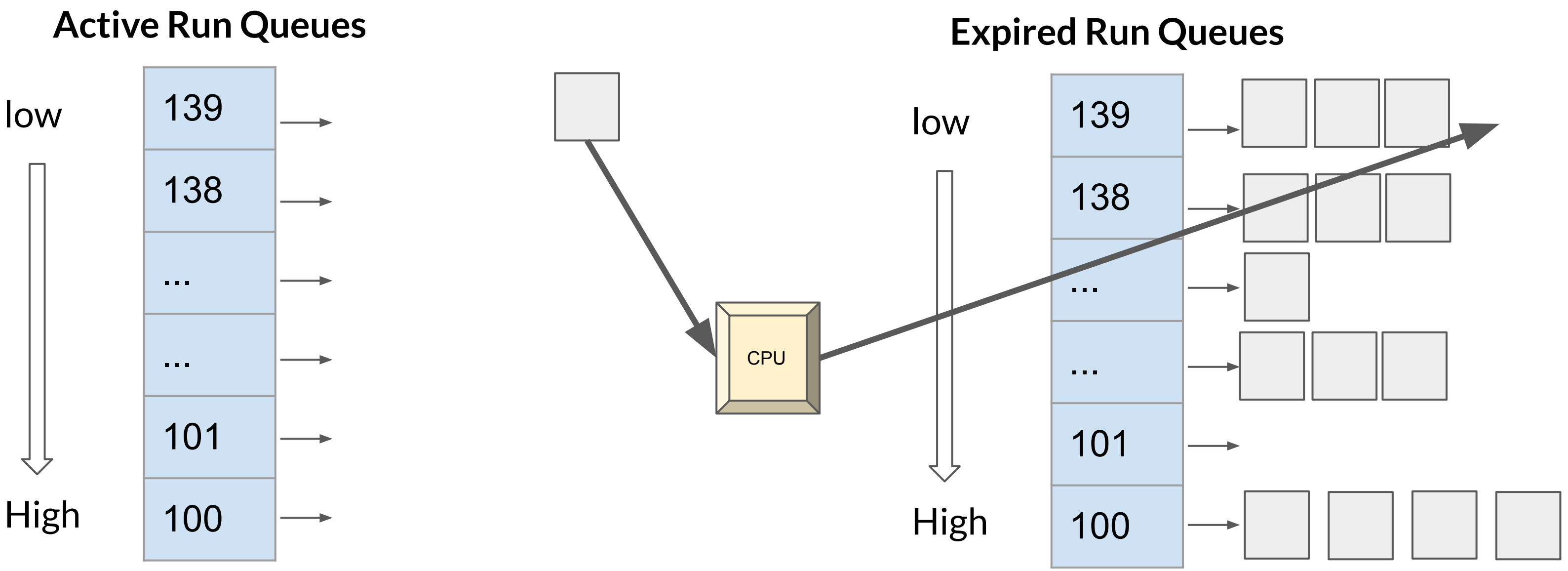
- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



This continues for a while….

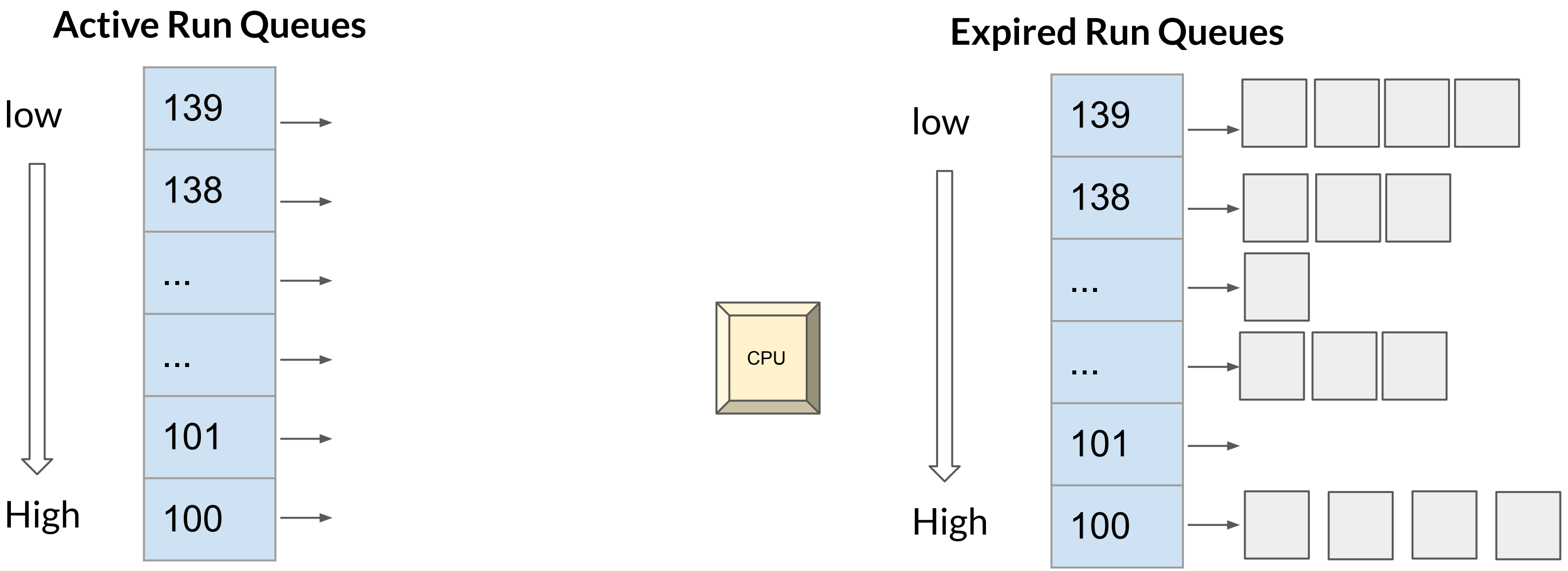
Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



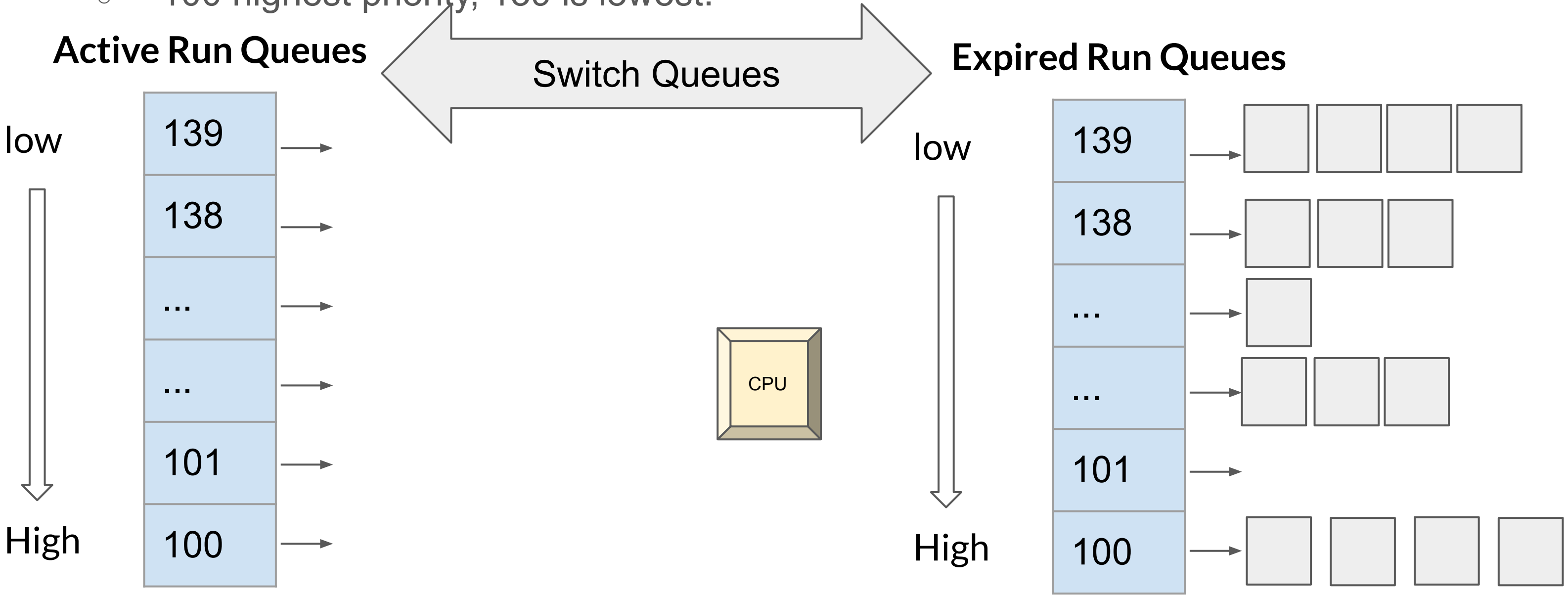
Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



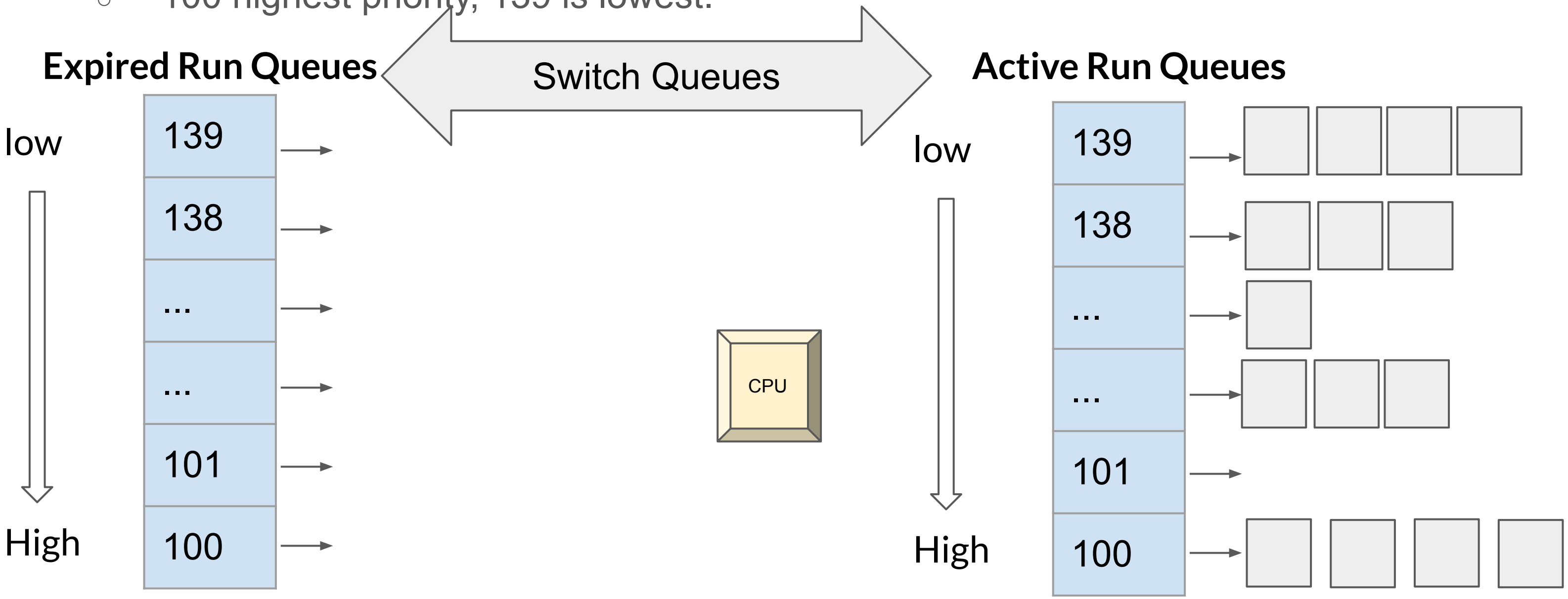
Scheduling Normal Processes

- Two ready queues for each CPU
    - Each queue has 40 priority classes (100 - 139)
    - 100 highest priority, 139 is lowest.



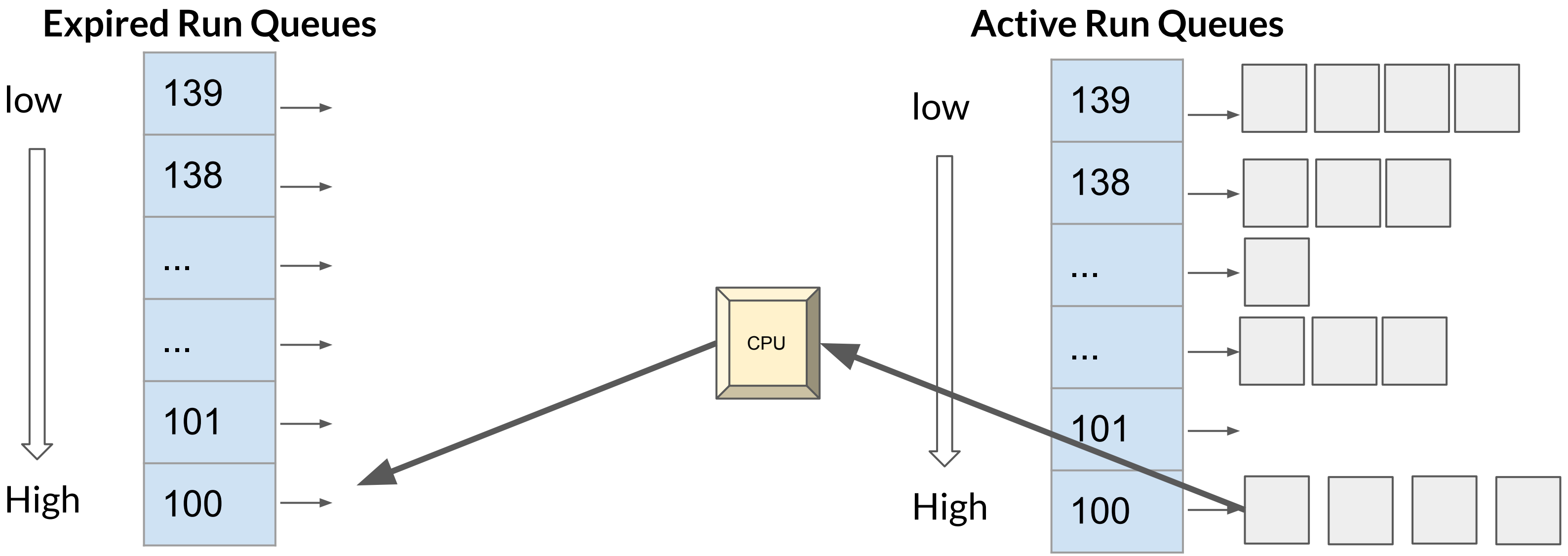
Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



Scheduling Normal Processes

- Two ready queues for each CPU
  - Each queue has 40 priority classes (100 - 139)
  - 100 highest priority, 139 is lowest.



More on priorities

- Static priorities
  - 120 is the default priority
  - Nice: command line to change default priority of a process.
    Nice -n [NUMBER] ./a.out

    NUMBER is a value from +19 (nice) to -20 (selfish)

- Dynamic priorities (heuristic)

Dynamic Priority

- To distinguish between batch and interactive processes
- Uses a “bonus” which changes based on a heuristic

Dynamic_priority = MAX(100, MIN(static_priority - bonus + 5, 139))

Dynamic Priority

- To distinguish between batch and interactive processes
- Uses a “bonus” which changes based on a heuristic

Dynamic_priority = MAX(100, MIN(static_priority - bonus + 5, 139))

Bonus has value between 0 and 10

If **bonus < 5**, implies less interaction with the user, thus CPU bound process. Priority is decreased in this case.

If **bonus > 5**, implies more interaction with the user, thus more of an interactive process. Priority is increased in this case.

Dynamic Priority

- To distinguish between batch and interactive processes
- Uses a “bonus” which changes based on a heuristic

Dynamic_priority = MAX(100, MIN(static_priority - bonus + 5, 139))

e.g.:

Dynamic_priority = MAX(100, MIN(120 - 8 + 5, 139))
Dynamic_priority = MAX(100, MIN(115, 139))
Dynamic_priority = MAX(100, 115)
Dynamic_priority = 115

Dynamic Priority

- To distinguish between batch and interactive processes
- Uses a “bonus” which changes based on a heuristic

Dynamic_priority = MAX(100, MIN(static_priority - bonus + 5, 139))

e.g.:

Dynamic_priority = MAX(100, MIN(120 - 8 + 5, 139))
Dynamic_priority = MAX(100, MIN(115, 139))
Dynamic_priority = MAX(100, 115)
Dynamic_priority = 115 **<120**

Dynamic Priority

- To distinguish between batch and interactive processes
- Uses a “bonus” which changes based on a heuristic

Dynamic_priority = MAX(100, MIN(static_priority - bonus + 5, 139))

e.g.:

Dynamic_priority = MAX(100, MIN(120 - 1 + 5, 139))
Dynamic_priority = MAX(100, MIN(124, 139))
Dynamic_priority = MAX(100, 124)
Dynamic_priority = 124

Dynamic Priority

- To distinguish between batch and interactive processes
- Uses a “bonus” which changes based on a heuristic

Dynamic_priority = MAX(100, MIN(static_priority - bonus + 5, 139))

e.g.:

Dynamic_priority = MAX(100, MIN(120 - 1 + 5, 139))
Dynamic_priority = MAX(100, MIN(124, 139))
Dynamic_priority = MAX(100, 124)
Dynamic_priority = 124 **>120**

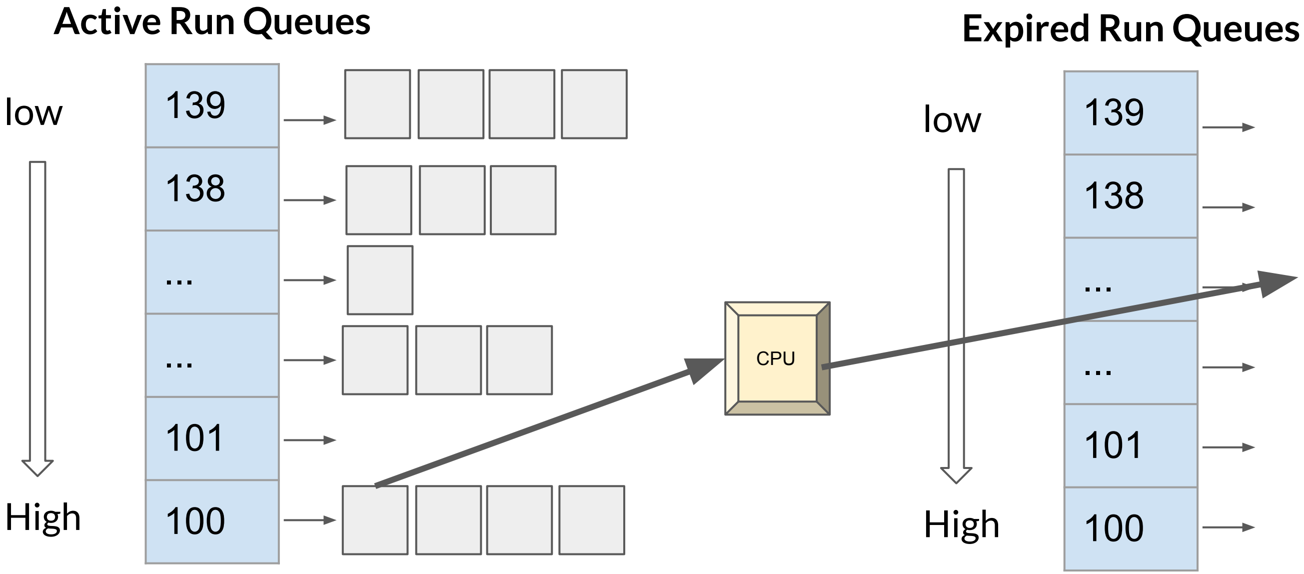
How does the OS set the bonus?

- Based on average sleep time:
  - I/O bound process will sleep more therefore should get a higher priority.
  - CPU bound process will sleep less, therefore should get lower priority.

Average sleep time	Bonus
>= 0 but < 100 ms	0
>= 100 ms but < 200 ms	1
>= 200 ms but < 300 ms	2
>= 300 ms but < 400 ms	3
>= 400 ms but < 500 ms	4
>= 500 ms but < 600 ms	5
>= 600 ms but < 700 ms	6
>= 700 ms but < 800 ms	7
>= 800 ms but < 900 ms	8
>= 900 ms but < 1000 ms	9
1 second	10

Dynamic Priority and Run Queues

- Dynamic priority used to determine on which queue a process is placed.
- No starvation, and everyone waits.



Setting timeslice (quantum) for each process

- Interactive processes have high priorities.
  - But likely to not complete their timeslice.
  - Given largest timeslice to ensure that CPU burst is completed without preemption.

```
If priority < 120:

        Timeslice = (140 - priority)  *  20 milliseconds
Else:
        Timeslice = (140 - priority)  *  5 milliseconds
```

Setting timeslice (quantum) for each process

- Interactive processes have high priorities.
  - But likely to not complete their timeslice.
  - Given largest timeslice to ensure that CPU burst is completed without preemption.

If priority < 120:

Timeslice = (140 - priority) * 20 milliseconds

Else:

Timeslice = (140 - priority) * 5 milliseconds

Priority:	Static Pri	Niceness	Quantum
Highest	100	-20	800 ms
High	110	-10	600 ms
Normal	120	0	100 ms
Low	130	10	50 ms
Lowest	139	19	5 ms

Limitations of O(1) Scheduler

- Heuristics too complex to distinguish process type.
- Dependence between time slice and priority.
- Priority and time slice values not uniform (fairness?).

How is it constant time?

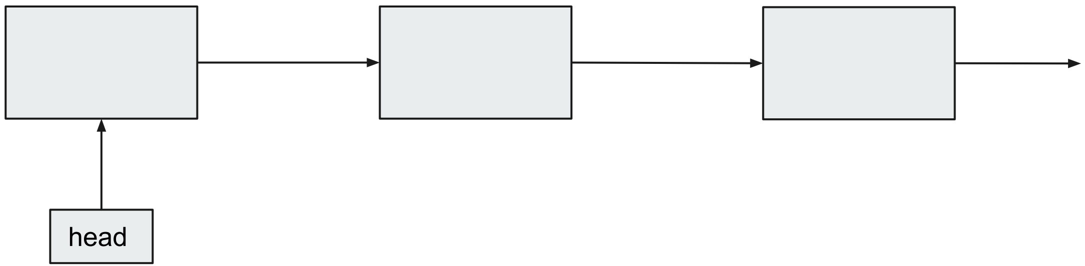
O(1) scheduler has two operations:

1. Find the queue with the lowest number with at least one task.
2. Pick a process from the front of that queue.

How is it constant time?

O(1) scheduler has two operations:

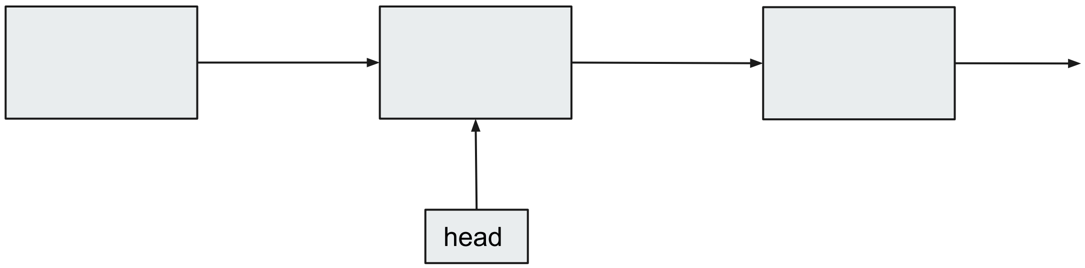
1. Find the queue with the lowest number with at least one task.
2. Pick a process from the front of that queue.



How is it constant time?

O(1) scheduler has two operations:

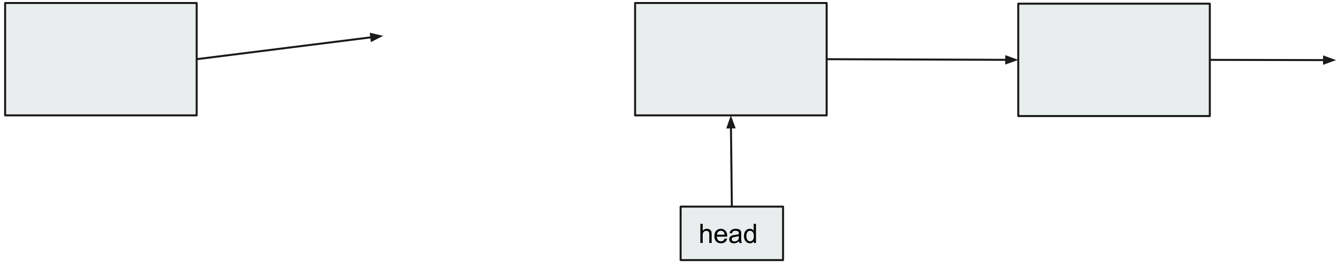
1. Find the queue with the lowest number with at least one task.
2. Pick a process from the front of that queue.



How is it constant time?

O(1) scheduler has two operations:

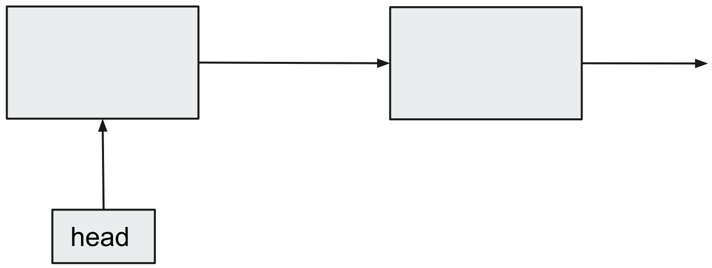
1. Find the queue with the lowest number with at least one task.
2. Pick a process from the front of that queue.



How is it constant time?

O(1) scheduler has two operations:

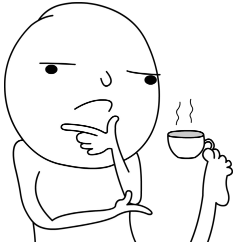
1. Find the queue with the lowest number with at least one task.
2. Pick a process from the front of that queue.



How is it constant time?

O(1) scheduler has two operations:

1. Find the queue with the lowest number with at least one task.
2. Pick a process from the front of that queue.



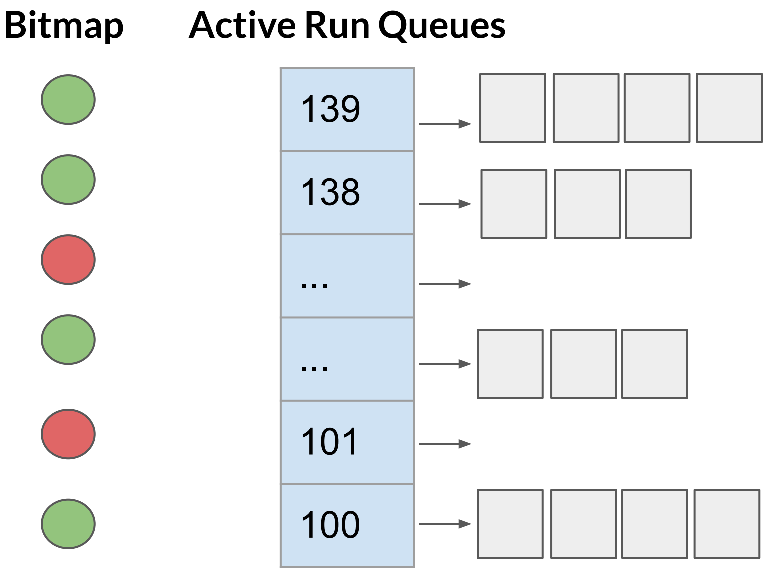
How is that constant time?

How is it constant time?

O(1) scheduler has two operations:

1. Find the queue with the lowest number with at least one task.
2. Pick a process from the front of that queue.

Maintain a bitmap of run queues with non-zero length



How is it constant time?

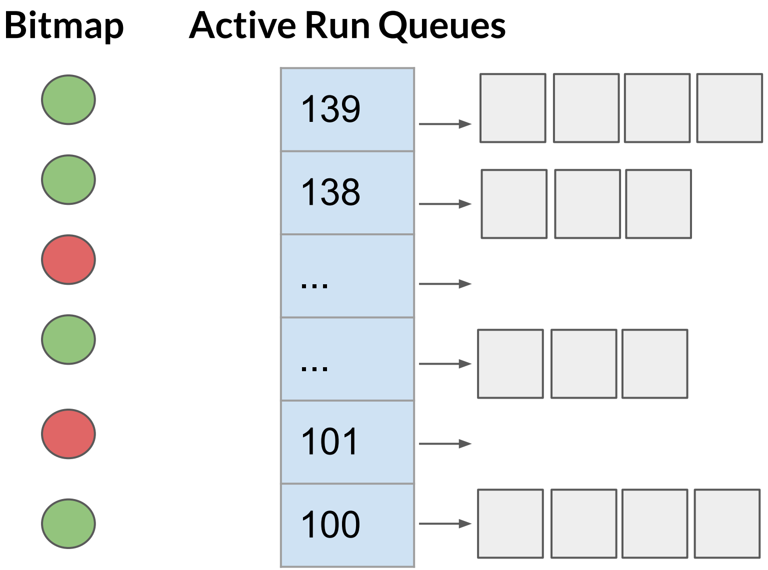
O(1) scheduler has two operations:

1. Find the queue with the lowest number with at least one task.
2. Pick a process from the front of that queue.

Use special intel instruction ***bsfl***
Find-first-bit-set

It returns the **index** of the first non-zero bit in the bitmap, in constant time.

That index is the run queue number!



Go over project 2

References

- Introduction to Operating Systems (Prof. Chester Rebeiro, IIT Madras)
- Understanding the Linux Kernel, 3rd Edition By Daniel P. Bovet, Marco Cesati
- Linux Kernel Development By Robert Love