CS330: Operating Systems

Process scheduling policies

Recap: MLFQ

- MLFQ: Dynamic priority based on process characteristics
- How does this strategy work for short and interactive jobs?
- Approximates SJF for short jobs, interactive jobs retain higher priority levels
- Does it avoid starvation?
- No. Requires additional mechanism like priority boost.
- Can a user trick the scheduler?
- Yes. Additional history regarding execution is required to be maintained

Today's agenda: Overview of Linux scheduling

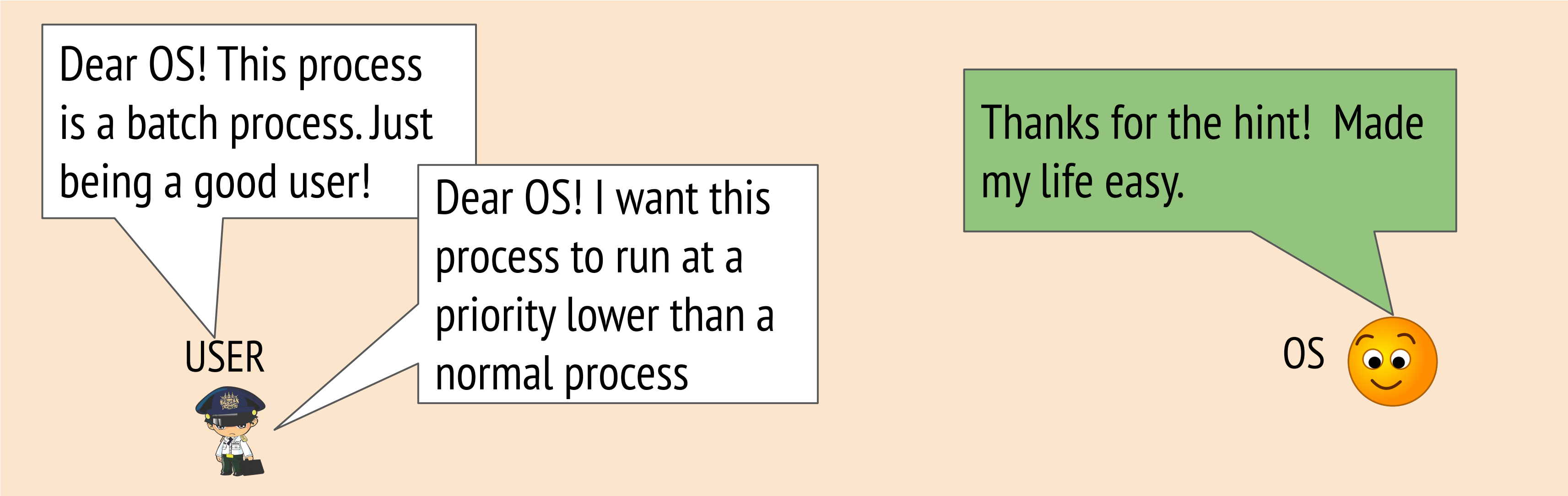
Scheduling is much more complex in a real OS!

- Scheduling requirement of different processes in the system are different
  - Real-time processes: Should meet strict deadlines
  - Interactive processes: Responsive scheduling
  - Batch processes: Starvation free scheduling

Scheduling is much more complex in a real OS!

- Scheduling requirement of different processes in the system are different
  - Real-time processes: Should meet strict deadlines
  - Interactive processes: Responsive scheduling
  - Batch processes: Starvation free scheduling
- Well intentioned users should be able to influence the scheduling policy in a positive manner

Scheduling is much more complex in a real OS!



- Well intentioned users should be able to influence the scheduling policy in a positive manner

Scheduling is much more complex in a real OS!

- Scheduling requirement of different processes in the system are different
  - Real-time processes: Should meet strict deadlines
  - Interactive processes: Responsive scheduling
  - Batch processes: Starvation free scheduling
- Well intentioned users should be able to influence the scheduling policy in a positive manner
- Greed of greedy users should be controlled by the OS

Scheduling is much more complex in a real OS!

Dear OS! This process requires higher priority than other normal processes. You know what, it is very interactive.
Not really! Just trying to fool you.

Buddy! You can fool me for a little while. I will catch you eventually.

USER

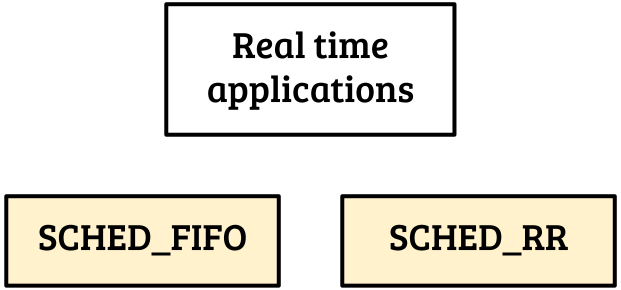
OS

- Greed of greedy users should be controlled by the OS

Scheduling is much more complex in a real OS!

- Scheduling requirement of different processes in the system are different
  - Real-time processes: Should meet strict deadlines
  - Interactive processes: Responsive scheduling
  - Batch processes: Starvation free scheduling
- Well intentioned users should be able to influence the scheduling policy in a positive manner
- Greed of greedy users should be controlled by the OS
- Conclusion: OS scheduling should provide flexibility while being auto-tuning in nature

Linux scheduling classes: Real time applications



- Real time applications are always higher priority than normal processes
- Priority value: 1 to 99 (In Linux, lower value ⇒ higher priority)
- FIFO: Run to completion
- RR: Round robin within a given priority-level
- *sched_setscheduler* system call to define scheduling class and priorities

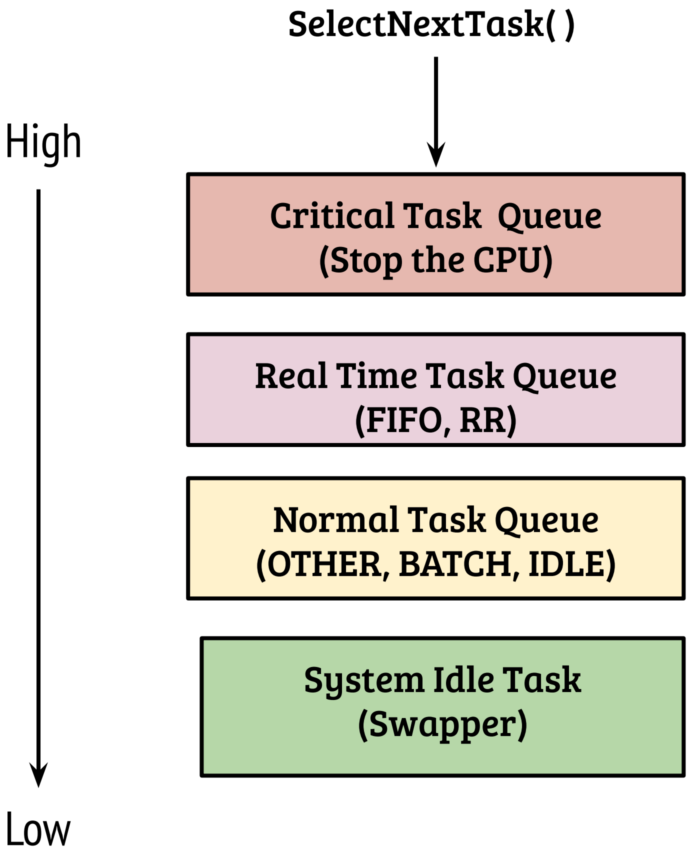
Linux scheduling classes: normal applications

Normal Applications

SCHED_OTHER SCHED_BATCH SCHED_IDLE

- SCHED_OTHER: Default policy, OS dynamic priorities and variable time slicing comes into picture
- SCHED_BATCH: Assume CPU bound while calculating dynamic priorities
- SCHED_IDLE: Very low priority jobs

Selecting the next task



- A task is picked from the non-empty highest priority queue
- Critical task queue contains tasks which require immediate attention: hardware events, restart etc.
- Normal task queue (a.k.a fair scheduling class) implements the heuristics to self-adjust
- If all the queues are empty, swapper task is scheduled (HLT the CPU)

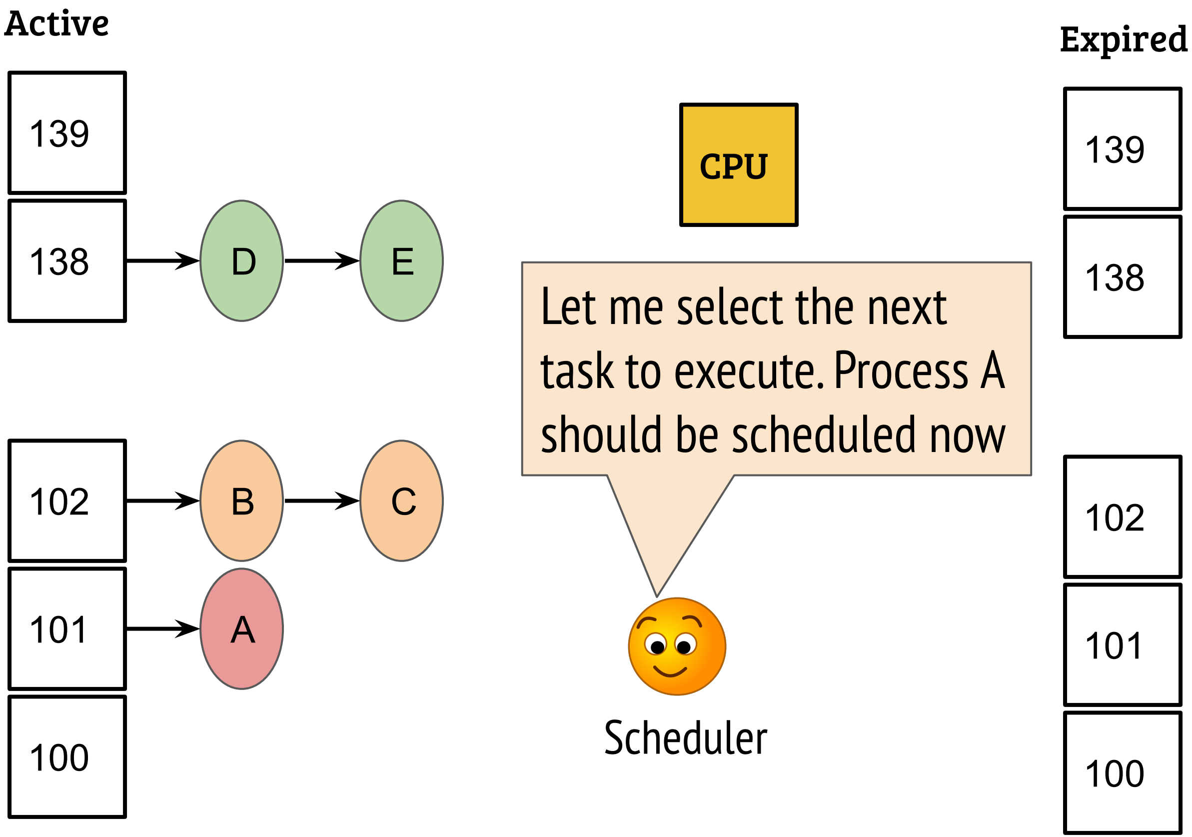
Normal (fair) scheduling class

- 40 priority levels (100 to 139)
- Every process starts with a default priority of 120
- Linux provides *nice* system call to adjust the static priority
    - *nice(int x)*, where x is between 19 to -20
    - nice(19) ⇒ Move the process to lowest priority queue i.e., 139
    - nice(-20) ⇒ Move the process to highest priority queue i.e., 100

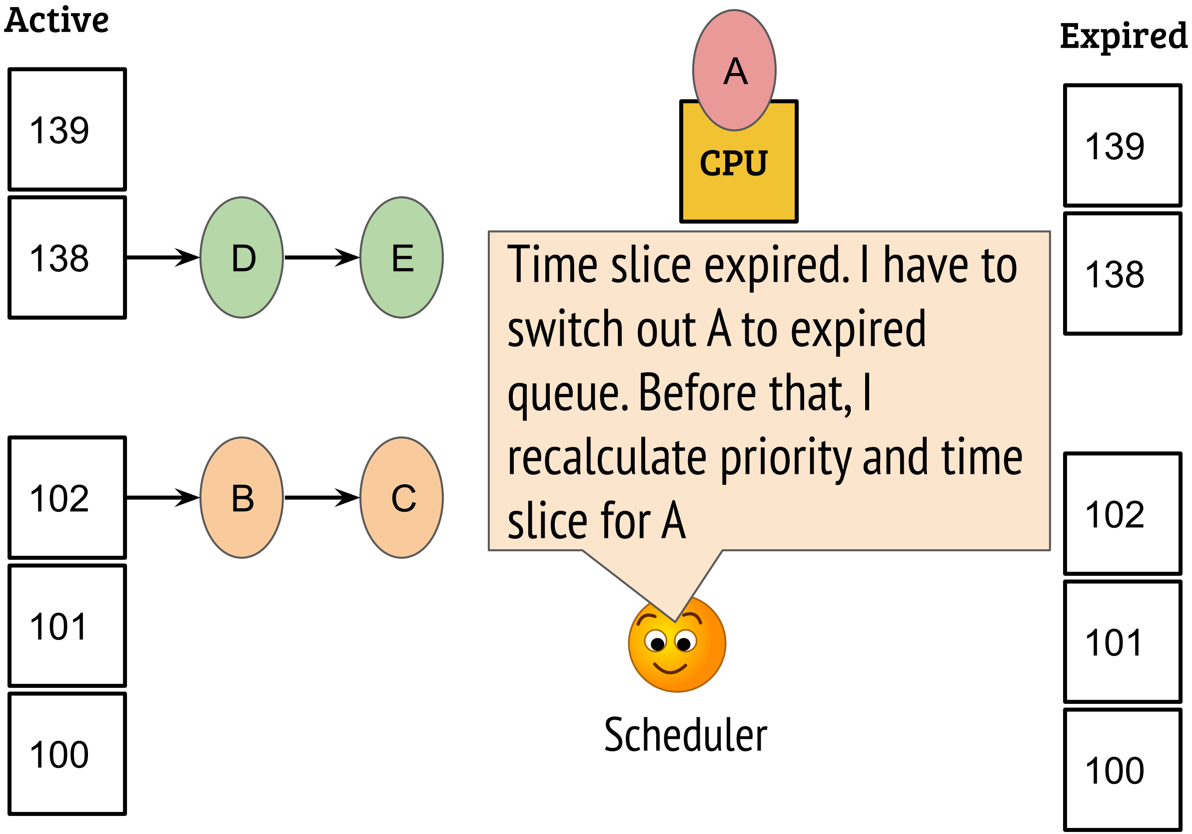
Normal (fair) scheduling class

- 40 priority levels (100 to 139)
- Every process starts with a default priority of 120
- Linux provides *nice* system call to adjust the static priority
  - *nice(int x)*, where x is between 19 to -20
  - nice(19) ⇒ Move the process to lowest priority queue i.e., 139
  - nice(-20) ⇒ Move the process to highest priority queue i.e., 100
- Dynamic priority is calculated by the Linux kernel considering the interactiveness of the process
  - More interactive processes move towards the priority level 100

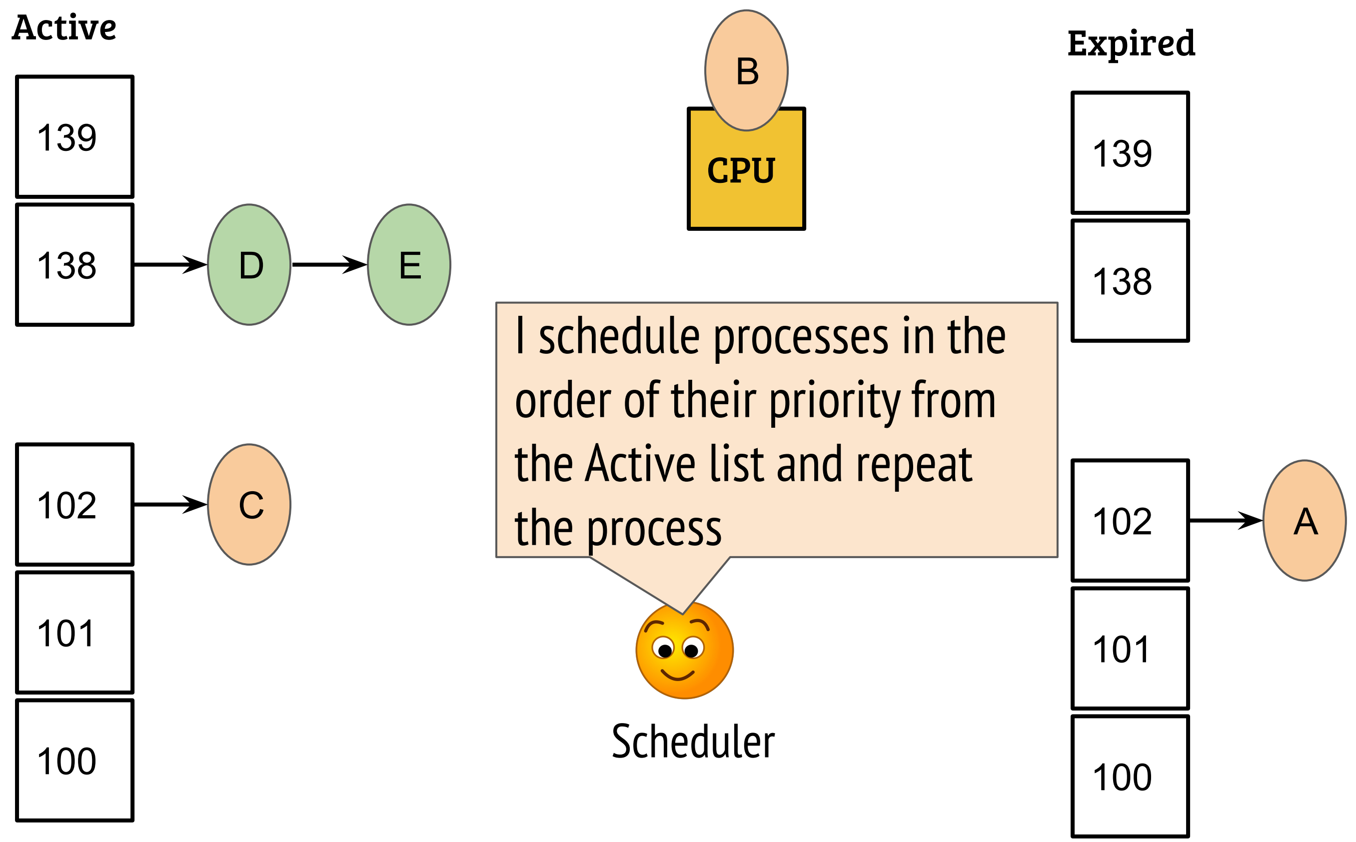
Linux O(1) scheduler



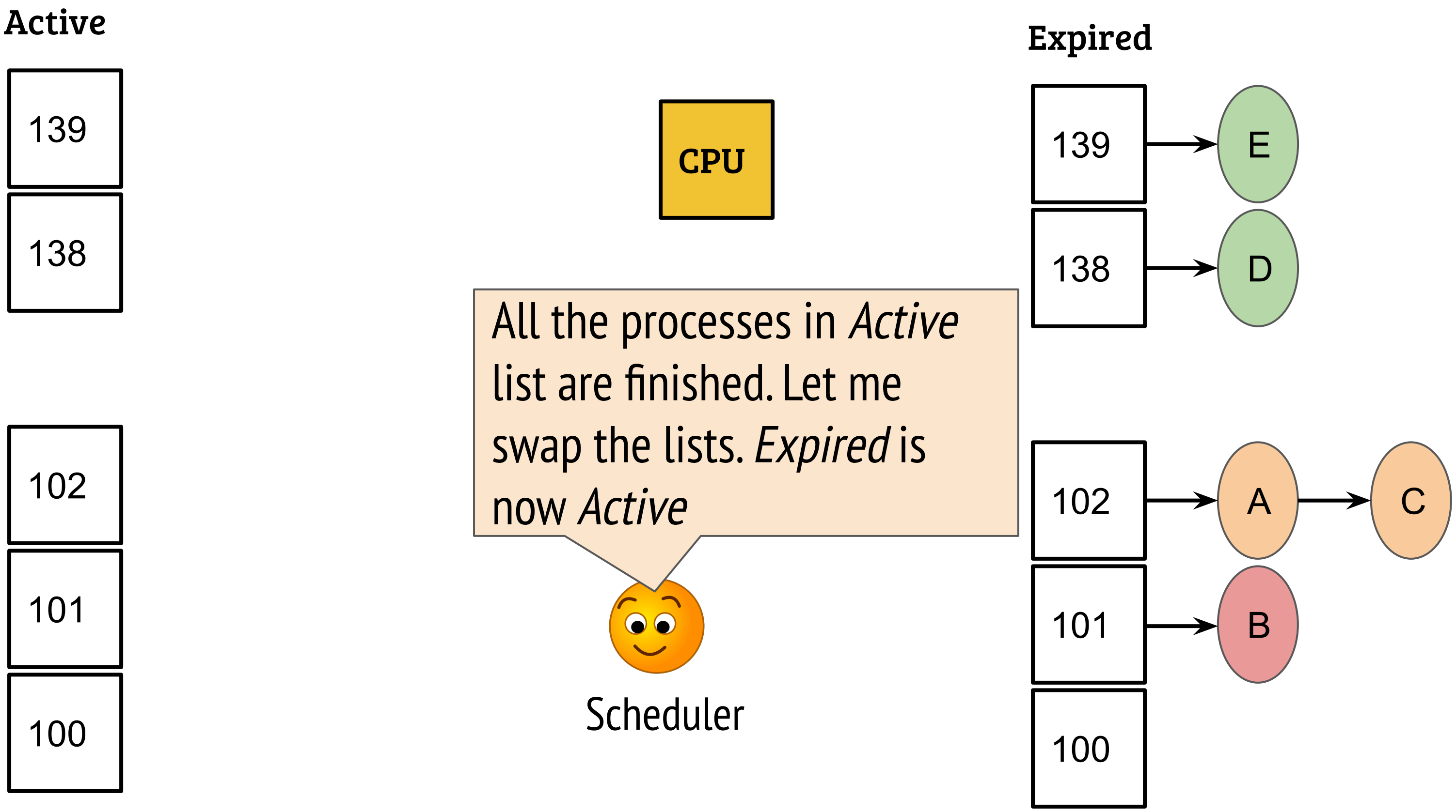
Linux O(1) scheduler



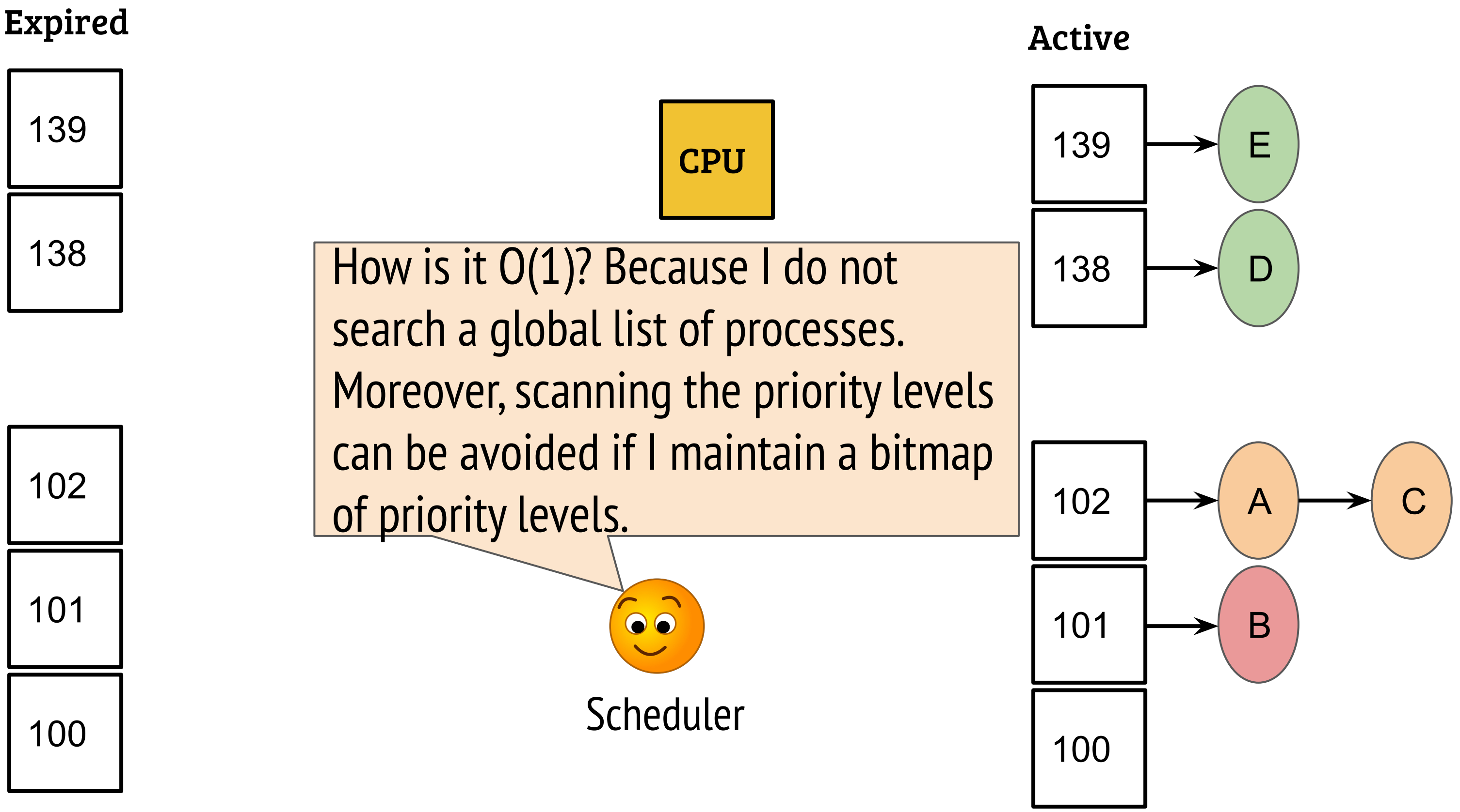
Linux O(1) scheduler



Linux O(1) scheduler



Linux O(1) scheduler



O(1) scheduler: value of time slice

- Objective: reduce timer interrupts (tickless system)
- High priority processes are given big time slices
  - Interactive processes relinquish CPU before the quantum expiry
- Low priority processes are given small time slices
  - Should not starve the interactive applications

O(1) scheduler: value of time slice

- Objective: reduce timer interrupts (tickless system)
- High priority processes are given big time slices
    - Interactive processes relinquish CPU before the quantum expiry
- Low priority processes are given small time slices
    - Should not starve the interactive applications
- Result: In a busy system, low priority processes execute less frequently resulting in few timer interrupts