Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

with thanks to Mike Brady

Scheduler Issues

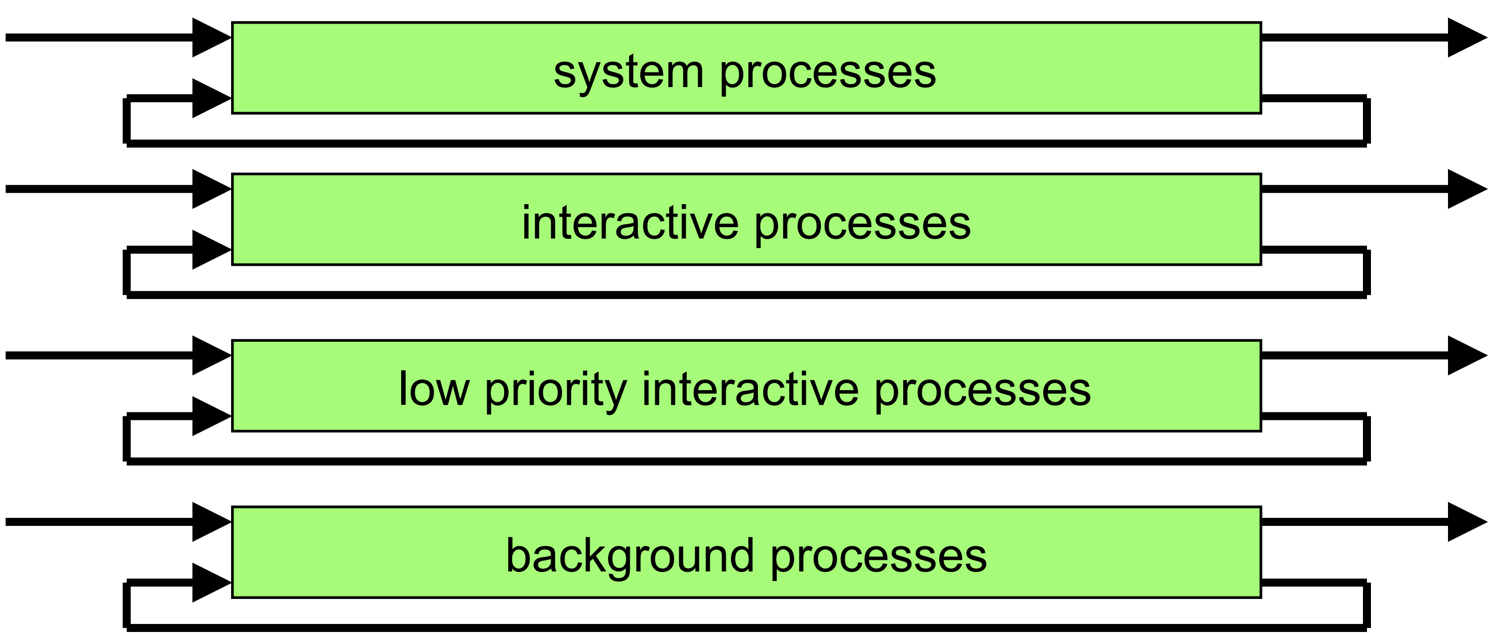
- Fairness
  - Make sure every process gets processor time eventually.
- Efficiency
  - Make best use of the resources available.
- Timeliness
  - Respond quickly to events, e.g. keyboard, etc.
  - Provide processor time reliably, in real-time, over extended periods.
- Requirements may conflict.

Real-Time

- Real-time means the correctness or utility of a program depends on being logically correct but also upon the time by which results or services are provided by the program. Real-time systems have deadlines.
  - *Hard* real-time. If a deadline is missed, the system fails totally.
  - *Firm* real-time. If a deadline is missed, the delayed service is useless, but the overall system can tolerate the occasional failure.
  - *Soft* real-time. If a deadline is missed, the delayed service is degraded, not useless, and the overall system is degraded but still useful/usable.
- Windows, Mac OS X, [Standard] Linuxes & FreeBSD/OpenBSD/NetBSDs are *not* hard real-time-capable OSes.

Scheduling Arrangements

- Multi-level queue scheduling
  - Often we can group processes together based on their characteristics or purpose
    - system processes
    - interactive process
    - low priority interactive processes
    - background processes



Scheduling Arrangements

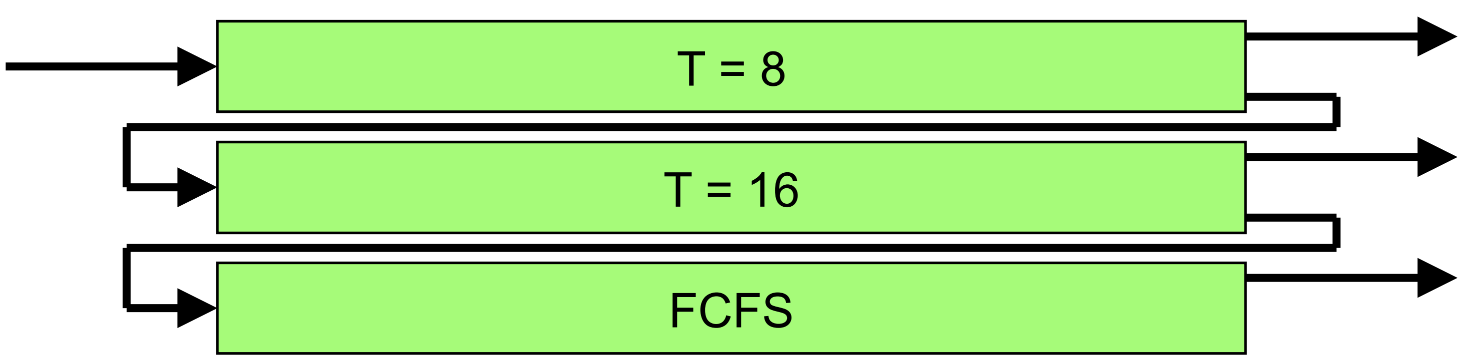
- Multi-level queue scheduling
  - Processes are permanently assigned to these groups and each group has an associated queue
  - Each process group queue has an associated queuing discipline
    - FCFS, SJF, priority queuing, Round-Robin, …
  - Additional scheduling is performed between the queues
    - Often implemented as fixed-priority pre-emptive scheduling
  - Each queue may have absolute priority over lower priority queues
    - In other words, no process in the background queue can execute while there are processes in the interactive queue
    - If a background process is executing when an interactive process arrives in the interactive queue, the background process will be preempted and the interactive process will be scheduled
  - Alternatively, each queue could be assigned a time slice
    - For example, we might allow the interactive queue 80% of the CPU time, leaving the remaining 20% for the other

Scheduling Arrangements

- Multi-level feedback queue scheduling
  - Unlike multi-level queue scheduling, this scheme allows processes to move between queues
  - Allows I/O bound processes to be separated from CPU bound processes for the purposes of scheduling
  - Processes using a lot of CPU time are moved to lower priority queues
  - Processes that are I/O bound remain in the higher priority queues
  - Processes with long waiting times are moved to higher priority queues
    - Aging processes in this way prevents starvation
  - Higher priority queues have absolute priority over lower priority queues
    - A process arriving in a high-priority queue will preempt a running process in a lower priority queue

Scheduling Arrangements

- Multi-level feedback queue scheduling
  - When a process arrives on the ready queue (a new process or a process that has completed waiting for an event), it is placed on the highest priority queue
  - Processes in the highest priority queue have a quantum of 8ms
  - If a high priority process does not end its CPU burst within 8ms, it is preempted and placed on the next lowest priority queue
  - When the highest priority queue is empty, processes in the next queue are given 16ms time slices, and the same behaviour applies
  - Finally, processes in the bottom queue are serviced in FCFS order
  - We can change
    - number of queues
    - queuing discipline
    - rules for moving process to higher or lower priority queues



Multi-processor scheduling

- Flynn's taxonomy of parallel processor systems
  - Single instruction single data (SISD)
  - Single instruction multiple data (SIMD) - e.g., (some) "graphics cards"
    - vector / array processor
  - Multiple instruction single data (MISD)
    - argument over precise meaning (pipelined processor ???)
  - Multiple instruction multiple data (MIMD) - e.g. "multicore"
    - shared memory (tightly coupled)
      - asymmetric (master/slave)
      - symmetric (SMP)
    - distributed memory (loosely coupled)
      - cluster

Multi-processor scheduling

- Processor affinity
  - If a process can execute on any processor, what are the implications for cache performance?
  - Most SMP systems implement a processor affinity scheme
    - soft affinity: an attempt is made to schedule a process to execute on the CPU it last executed on
    - hard affinity: processes are always executed on the same CPU

Multi-processor scheduling

- Load balancing
  - Scheduler can use a common ready queue for all processors or use private per-processor ready queues
  - If per-processor ready queues are used (and no affinity or soft-affinity is used), then load-balancing between CPUs becomes an issue
  - Two approaches:
    - pull-migration: idle processors pull ready jobs from other processors
    - push-migration: load on each processor is monitored and jobs are pushed from overloaded processors to under-loaded processors
- Load-balancing competes with affinity

Scheduling in Linux (2.6…)

- Three categories of process:
  - Interactive
  - Batch
  - Real-time

Process scheduling in Linux

- Scheduling goals
  - Fast response time
  - High throughput
  - Avoid starvation
  - Satisfy the needs of both high- and low-priority processes
  - ...

Process scheduling in Linux

- Pre-emptive scheduling
  - When quantum of the executing process expires, another process may be executed in its place

Process scheduling in Linux

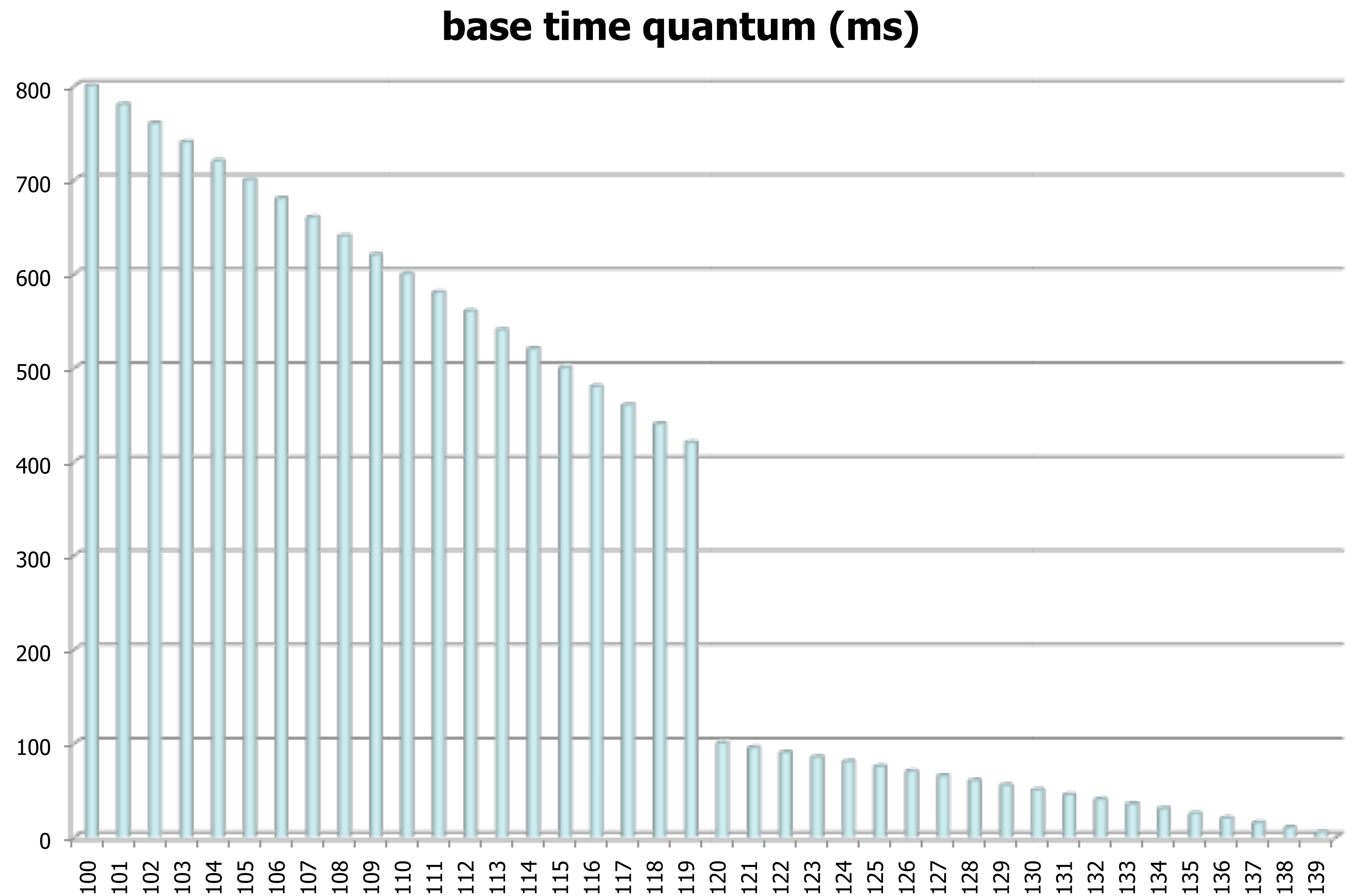
- Processes are ranked according to priorities
  - Priorities are dynamic: the operating system scheduler adjusts process priorities based on behaviour (e.g. decrease priority of long-running processes, boost priority when on I/O completion, etc.)
  - This results in complex schemes for determining the current priority for a process but simpler schemes for selecting the next process to execute.

Process scheduling in Linux

- Every (conventional) process has a static priority
  - in the range 100 (highest priority) ... 139 (lowest priority)
- Static priority determines base time quantum

```
if (static priority < 120) then
    base time quantum = (140 – static priority) x 20
else
    base time quantum = (140 – static priority) x 5
```

Process scheduling in Linux



Process scheduling in Linux

- Dynamic priority determines which process to execute:
  - dynamic priority = max(100, min(static priority – bonus + 5,139))
  - bonus in the range 0 … 10 but the effect of a "bonus" less than 5 is to lower the dynamic priority of the process
  - bonus is based on average sleep time and is generated from a lookup table that gives a bigger bonus to processes that sleep for longer

Process scheduling in Linux

Average sleep time	**Bonus**
$0 \leq$ average sleep time < 100	0
$100 \leq$ average sleep time < 200	1
$200 \leq$ average sleep time < 300	2
$300 \leq$ average sleep time < 400	3
$400 \leq$ average sleep time < 500	4
$500 \leq$ average sleep time < 600	5
$600 \leq$ average sleep time < 700	6
$700 \leq$ average sleep time < 800	7
$800 \leq$ average sleep time < 900	8
$900 \leq$ average sleep time < 1000	9
$1000 \leq$ average sleep time	10

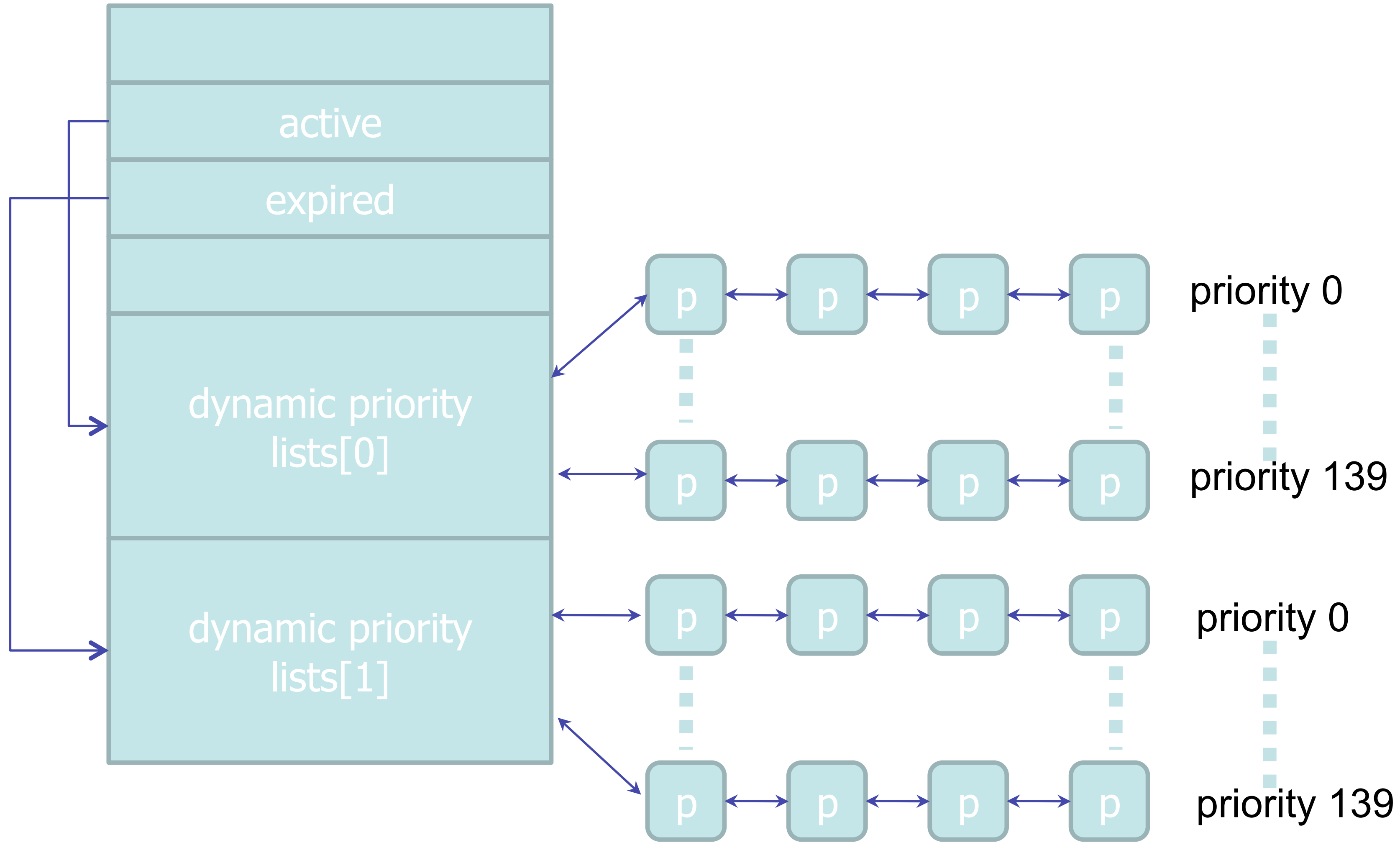
Process scheduling in Linux

- Linux differentiates between interactive and batch processes
- A process is interactive if (bonus – 5 ≥ static priority / 4 – 28)
  - It is easier for a high priority process to become interactive
  - Very low priority processes cannot become interactive
- To avoid starvation of low-priority process by high-priority processes, all runnable processes are classified as active or expired
  - active processes have not exhausted their quantum and are allowed to run
  - expired processes have exhausted their quantum and are not allowed to run until there are no more active processes
  - in practice the situation is a little more complex (e.g. interactive processes can get placed back on the active list as long as there are no “starving” expired processes)

Process scheduling in Linux

- Real-time process scheduling
  - (Soft) real-time processes are assigned priorities in the range 1 (highest priority) ... 99 (lowest priority)
  - There are two classes of real-time process: FIFO and "round-robin"
  - The priority is not dynamic
- FIFO processes
  - continue executing until they block or until they are pre-empted by a higher priority process
  - time slice is irrelevant
- "Round-Robin" processes
  - continue executing until they block, are pre-empted or their time slice expires
  - when the time-slice of a Round-Robin process expires, it is placed at the tail of the priority list

Process scheduling in Linux



Process scheduling in Linux

- Periodically, the remaining time slice of the currently executing process is decremented
- If the time slice has been exhausted …
    - the process is removed from the active queue
    - its time slice is restored
- … and it is placed …
    - on the active list if it is interactive and there are no starving expired processes
    - on the expired list otherwise
- Schedule the next process to execute from the head of the highest priority active queue
- If all processes are expired, the expired and active lists are swapped (cheaply, by changing pointers)