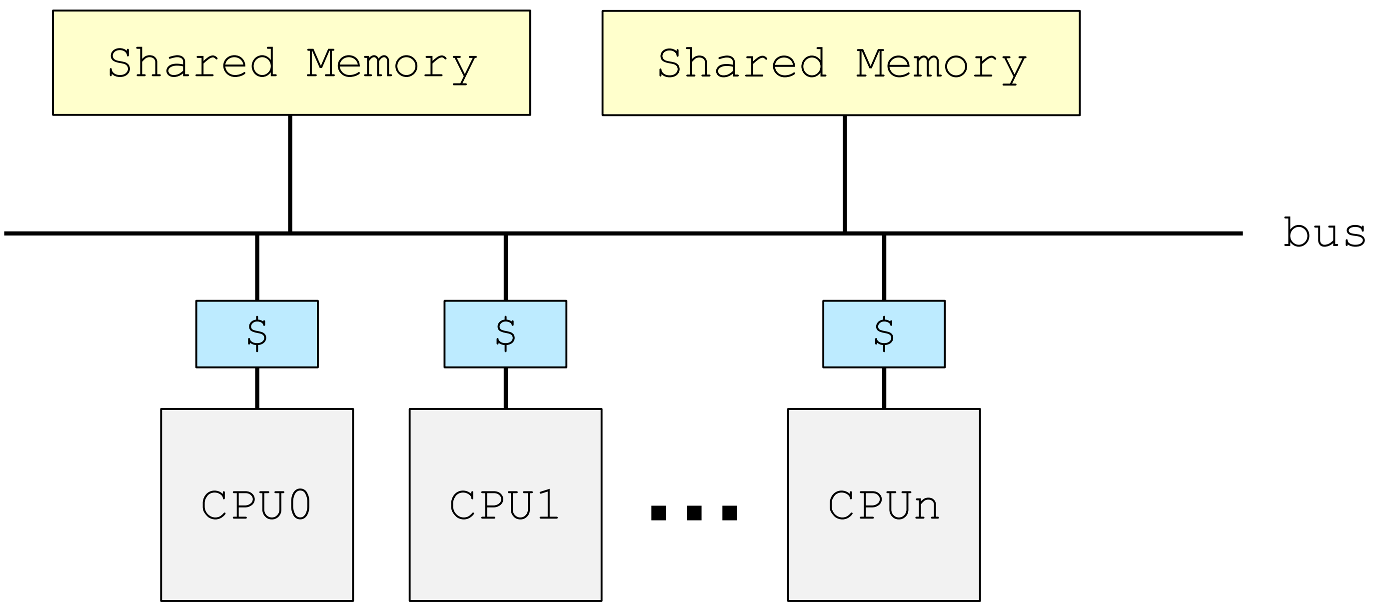
Running Programs on a System

Advanced Process Scheduling

SMP Scheduling

Symmetric Multiprocessor (SMP)

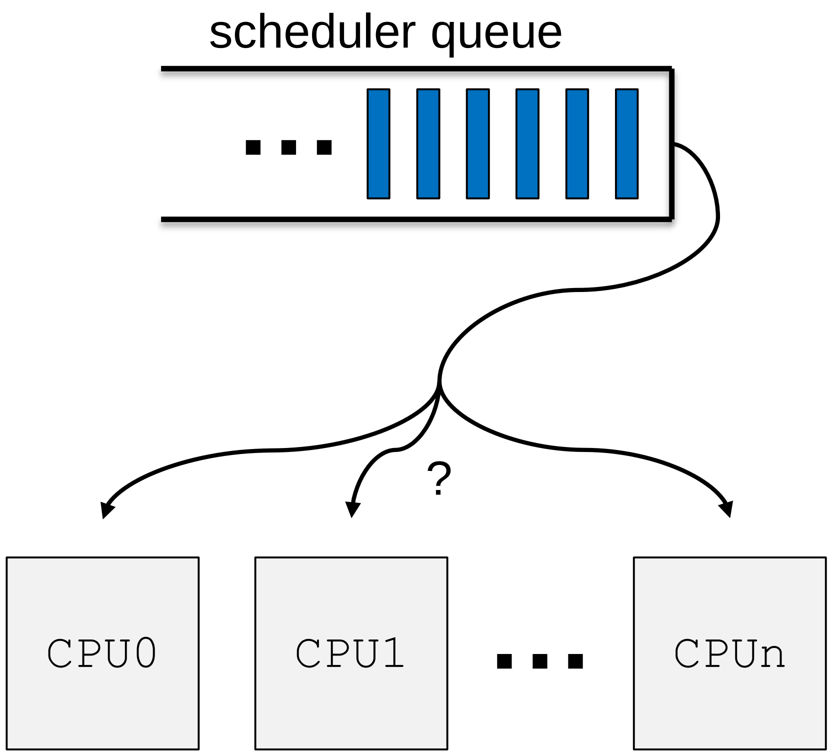
- Multiprocessor with shared memory
  - limited number of homogeneous CPUs with private cache
  - equal/similar access time to shared memory



CSE 컴퓨터공학부
Department of Computer Science & Engineering

Multiprocessor Scheduling

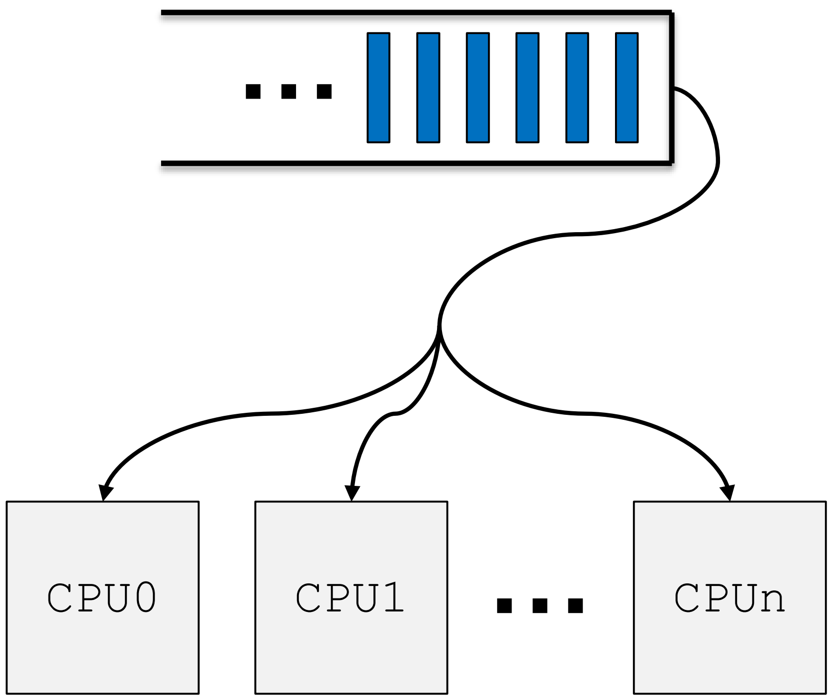
- How should tasks get distributed?



CSE 컴퓨터공학부 Department of Computer Science & Engineering

Global Queue

- each core runs his own scheduler
- global queue shared data structure

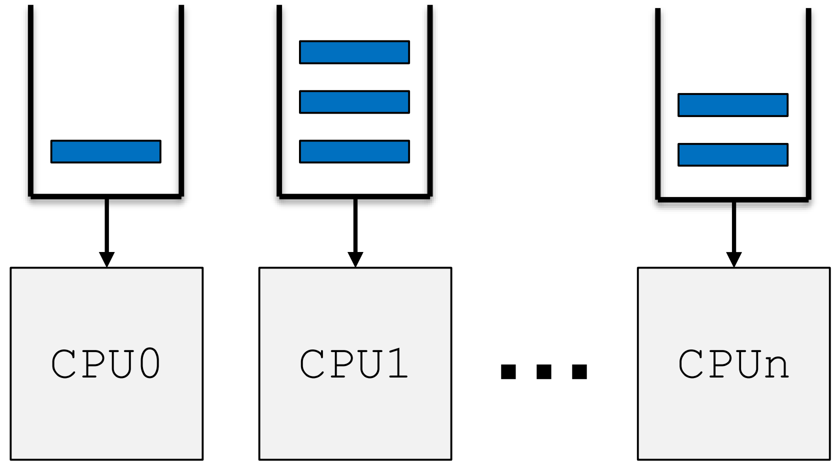


\+ fairness
\+ CPU utilization
\- scalability
\- cache locality

CSE 컴퓨터공학부 Department of Computer Science & Engineering

Local (per-CPU) Queue

- each core runs his own scheduler
- local queue private data structure
- static assignment of processes to CPUs

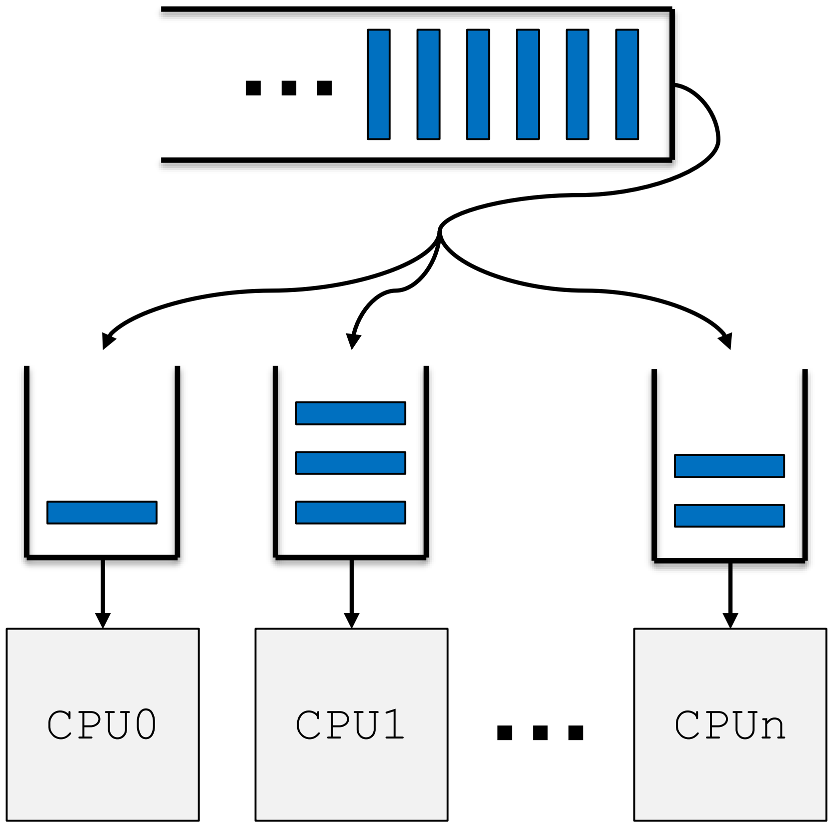


+ scalability
+ cache locality
+ simplicity
- load balancing

CSE 컴퓨터공학부 Department of Computer Science & Engineering

Local (per-CPU) Queue with Load Balancing

- hybrid approach between global & local queues
- schedule locally, balance globally
- processor affinity

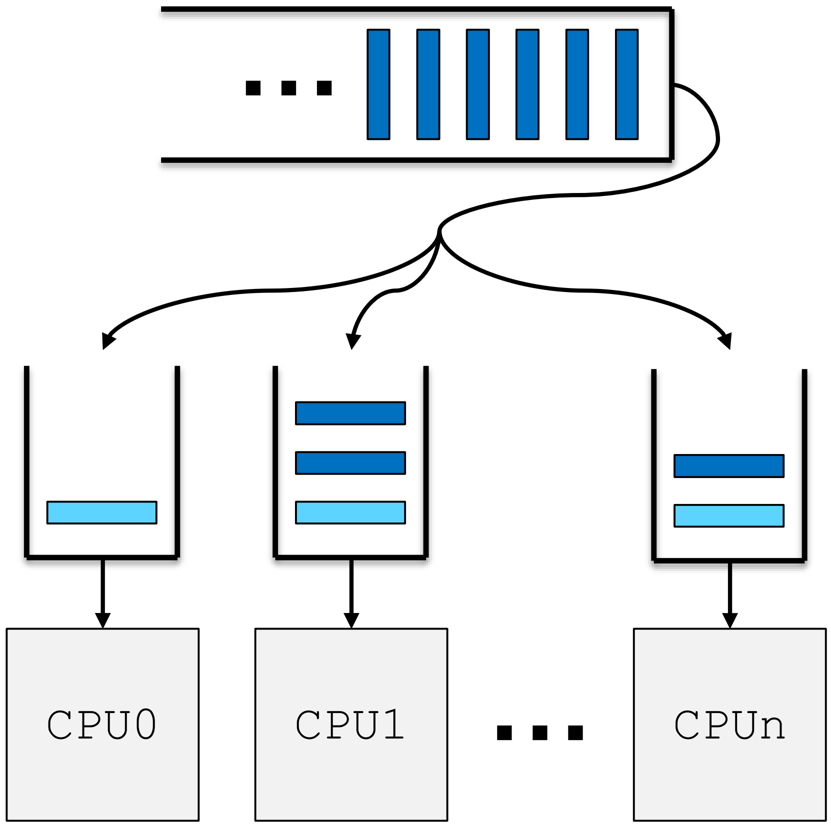


+ best of both worlds
- complexity

CSE 컴퓨터공학부 Department of Computer Science & Engineering

Gang Scheduling

- parallel tasks may need coordination and should be scheduled simultaneously
- local scheduler queue does not act independently
- global context switch



History of the Linux Scheduler

CSE 컴퓨터공학부
Department of Computer Science & Engineering

History of the Linux Scheduler

Original Scheduler
Linux 1.0 (1995)

- circular runqueue
- round robin scheduling

Scheduling Classes
Linux 2.2 (1999)

- rt, non-rt, non-preemptable, normal scheduling classes
- support for SMP

O(N) Scheduler
Linux 2.4 (2001)

- epochs and timeslices
- epoch and task selection complexity O(N)

O(1) Scheduler
Linux 2.6 (2003)

- run queues that allow task selection in O(1)
- lots of heuristics to determine a task's CPU/IO-boundness

Completely Fair Scheduler (CFS)
Linux 2.6.23 (2008)

- models an ideal, precise multi-tasking CPU on real hardware
- completely fair balancing of processor time between tasks

The O(N) Scheduler

Linux 2.4 (2001)

- **timeslice**: assigned to each task at the beginning of an epoch
  - depends on priority and unused timeslice in last epoch
  - fork(): parent/child share remaining timeslice (each gets half)
- **epoch**: scheduling "generation"
  - initialization: sum of all timeslices
  - end of epoch: no ready-to-run tasks has timeslice > 0
- **goodness()**: selection of next task to run

```
max_goodness = 0
max_t = idle_task

foreach runnable task t
  if ((g = goodness(t)) > max_goodness)
    max_goodness = g
    max_t = t

dispatch(max_t)
```


CSE 컴퓨터공학부 Department of Computer Science & Engineering

The O(N) Scheduler

Linux 2.4 (2001)

- **issues**
  - does not scale well
    - epoch calculation: O(# tasks)
    - task selection: O(# runnable tasks)
  - favors I/O bound tasks
    - maybe okay for server, less so for desktops
  - predefined minimal quantum is too long
    - long latency under high system load
  - weak support for real-time processes

The O(1) Scheduler

Linux 2.6 (2003)

- **two main ideas**
  - *active* and *expired* runqueues
    - active runqueue: tasks with timeslice > 0
    - expired: tasks that have expired their timeslice
    - active → expired: task moved when timeslice reaches 0
      - new timeslice is computed upon insertion into expired
    - active runqueue empty → array swap
  - pair of (active,expired) runqueues *per priority level*
    - scheduling: pick first task from first non-empty active runqueue of highest priority level → O(1)

CSE 컴퓨터공학부 Department of Computer Science & Engineering

The O(1) Scheduler

Linux 2.6 (2003)

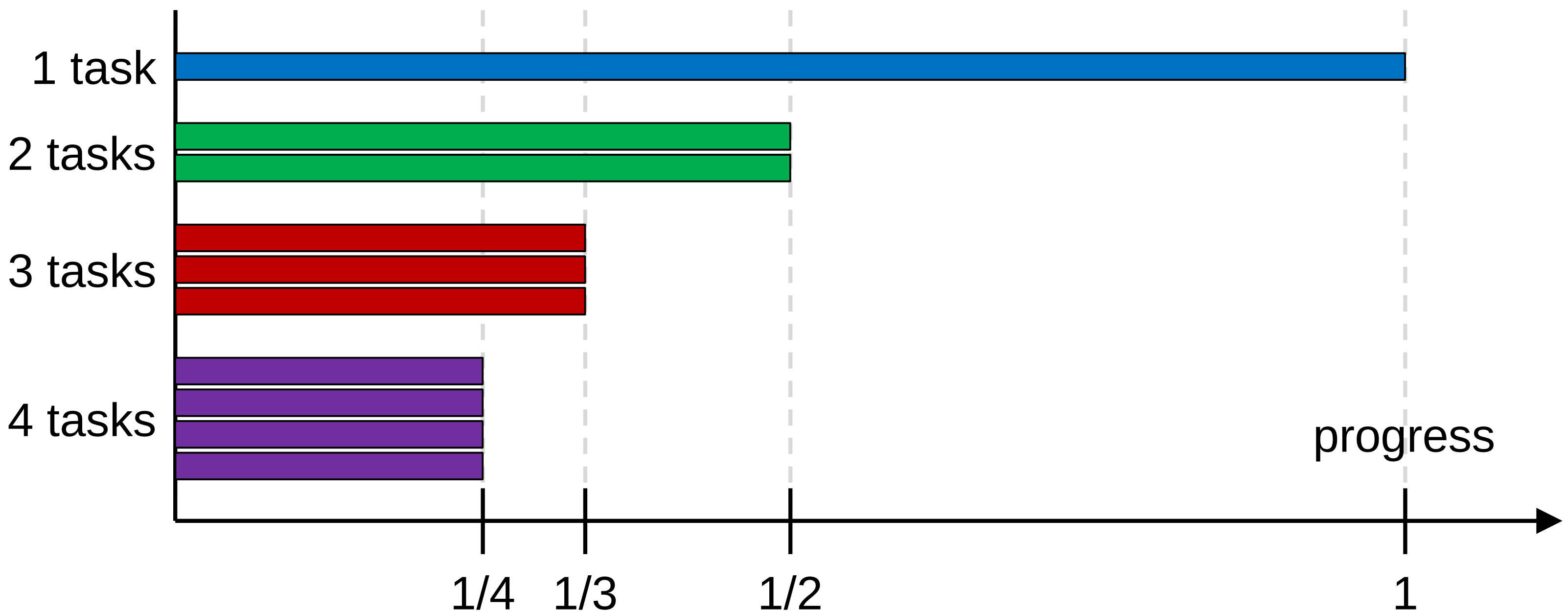
- **issues**
  - heuristics to determine CPU- resp. I/O-boundness of task
    - complex and error-prone
    - permitted attacks on the scheduler

CSE 컴퓨터공학부
Department of Computer Science & Engineering

The Completely Fair Scheduler

Linux 2.6.23 (2008)

- Models an ideal, precise multitasking CPU
  - simultaneous process of tasks
    infinitesimally small timeslices, no task switching overhead
  - *n* runnable tasks progress uniformly at $1/n^{th}$ of the CPU speed

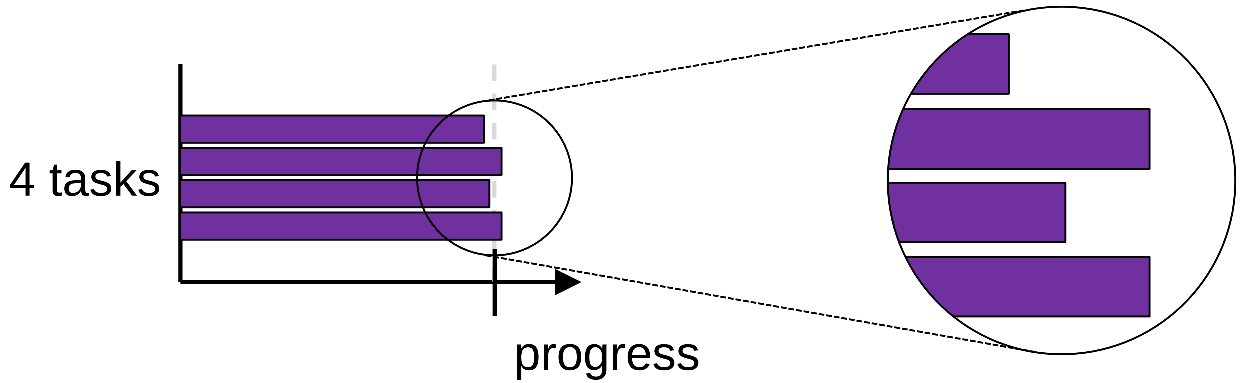


The Completely Fair Scheduler

Linux 2.6.23 (2008)

- Each task's (of equal priority) timeslice = 1/n of the default task latency
  - task latency = maximum wait time
  - default: 20ms
  - lower bound timeslice (default: 4ms) may force increasing the task latency
  - priorities modeled with proportional sharing

- Selection of next task to run = pick task with minimum runtime so far
  - runtime accounting: vruntime (nanosecond granularity)



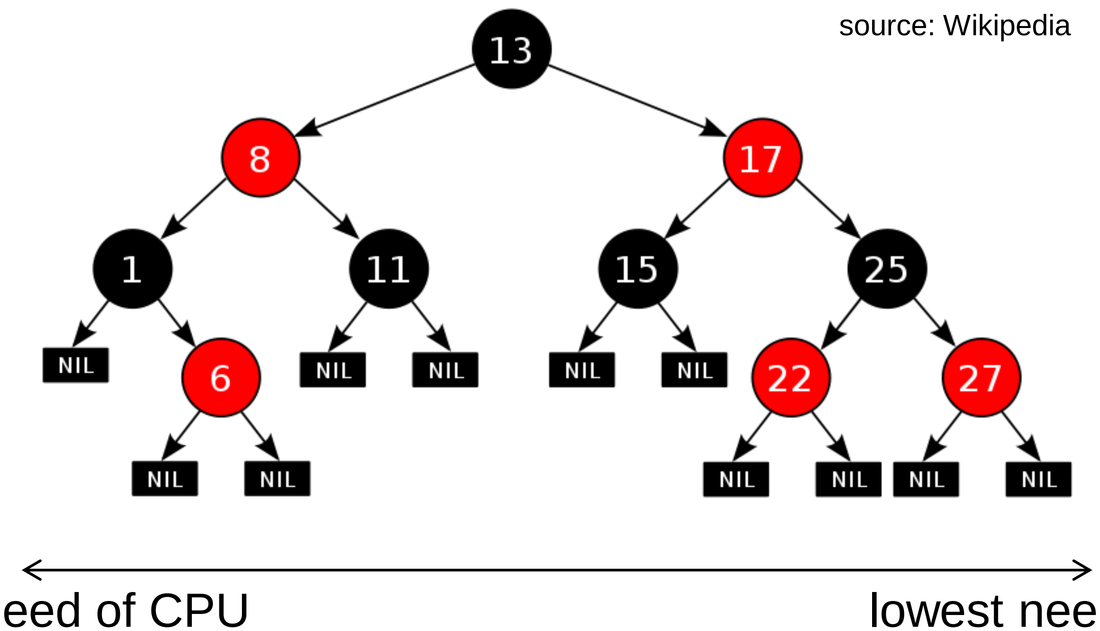
CSE 컴퓨터공학부 Department of Computer Science & Engineering

The Completely Fair Scheduler

Linux 2.6.23 (2008)

- Efficiently finding the task with minimum runtime
  - red-black tree
  - insertion & deletion: O(log N), find minimum: O(1)
  - ordered by runtime
  - task with minimum runtime is always left-most node

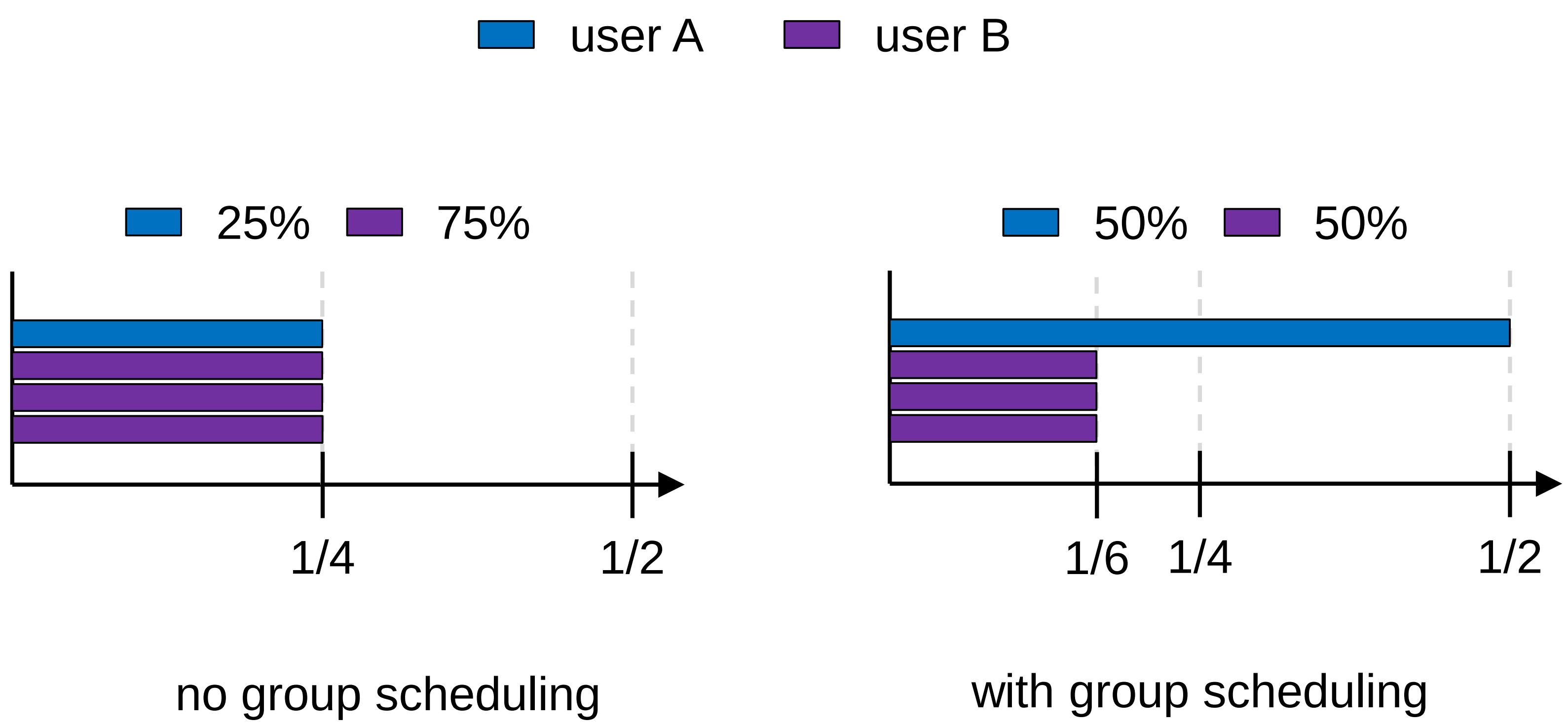
source: Wikipedia



CSE 컴퓨터공학부 Department of Computer Science & Engineering

The Completely Fair Scheduler

- Group Scheduling
  - divide CPU time equally between *groups* instead of *tasks*



The Completely Fair Scheduler

- Load balancing
  - active balancing
    periodically pull tasks over from busiest CPU
  - idle balancing
    as soon as there is no runnable task
    - migrated only if average idle time > migration cost

CSE 컴퓨터공학부 Department of Computer Science & Engineering