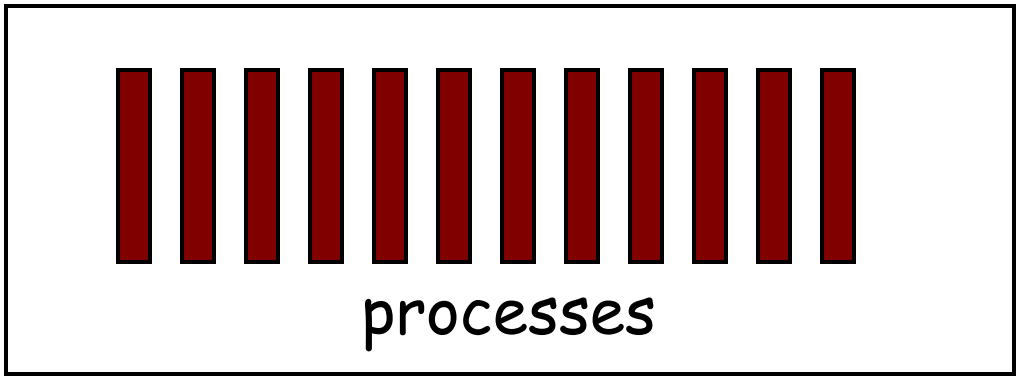
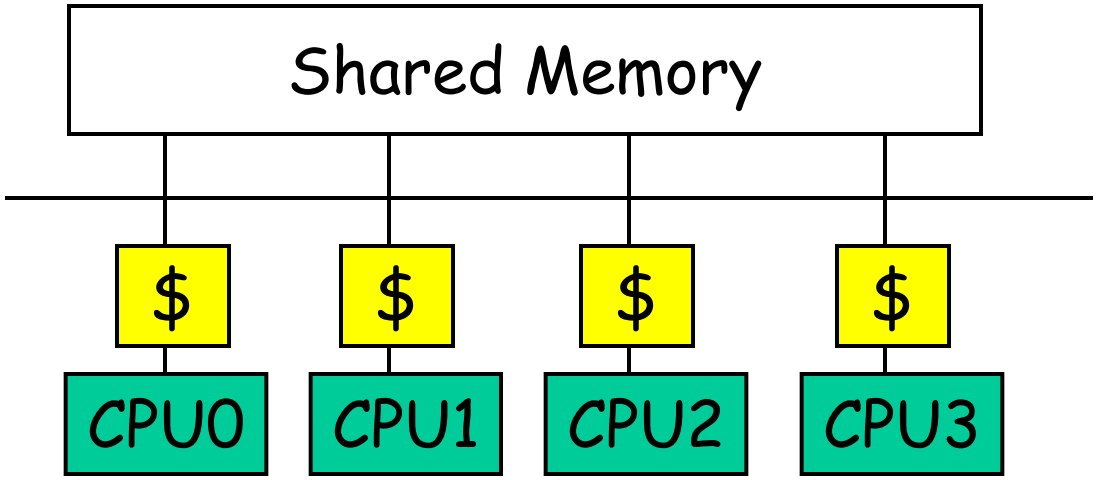
Scheduling II

- Multiprocessor scheduling issues
- Real-time scheduling
- Linux scheduling
- Linux scheduler architecture

How to allocate processes to CPUs?



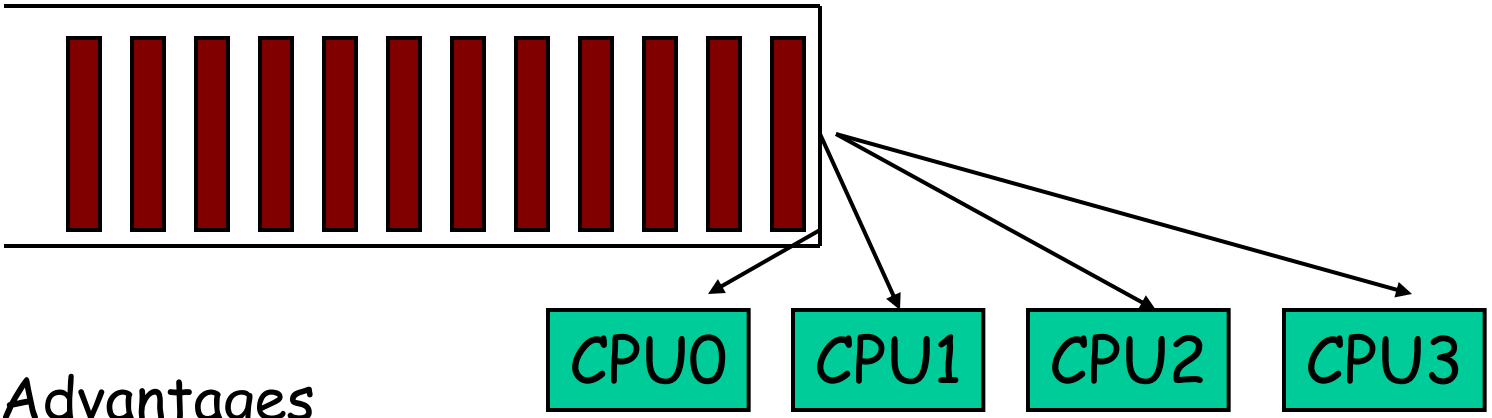
Symmetric multiprocessing (SMP)



- Multiple CPUs
- Same access time to main memory
- Private cache

Global queue of processes

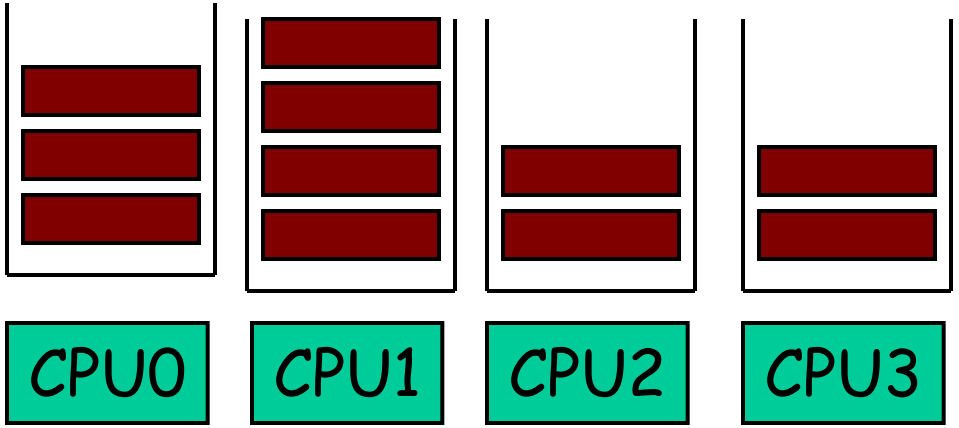
- One ready queue shared across all CPUs



- Advantages
  - Good CPU utilization
  - Fair to all processes
- Disadvantages
  - Not scalable (contention for global queue lock)
  - Poor cache locality
- Linux 2.4 uses global queue

Per-CPU queue of processes

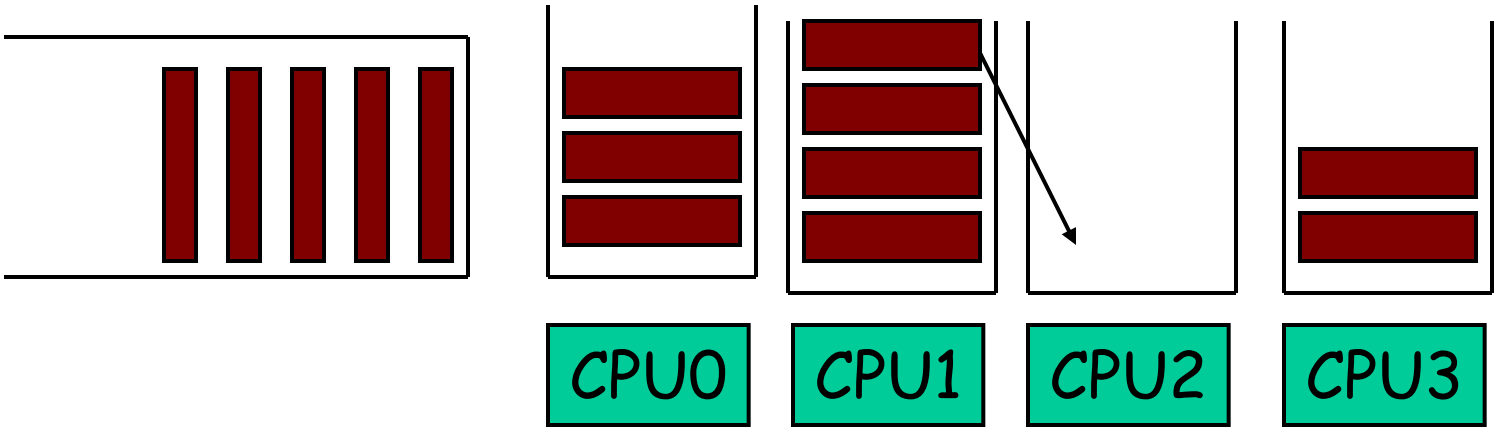
- Static partition of processes to CPUs



- Advantages
  - Easy to implement
  - Scalable (no contention on ready queue)
  - Better cache locality
- Disadvantages
  - Load-imbalance (some CPUs have more processes)
    - Unfair to processes and lower CPU utilization

Modern OSes take hybrid approaches

- Use both global and per-CPU queues
- Migrate processes across per-CPU queues



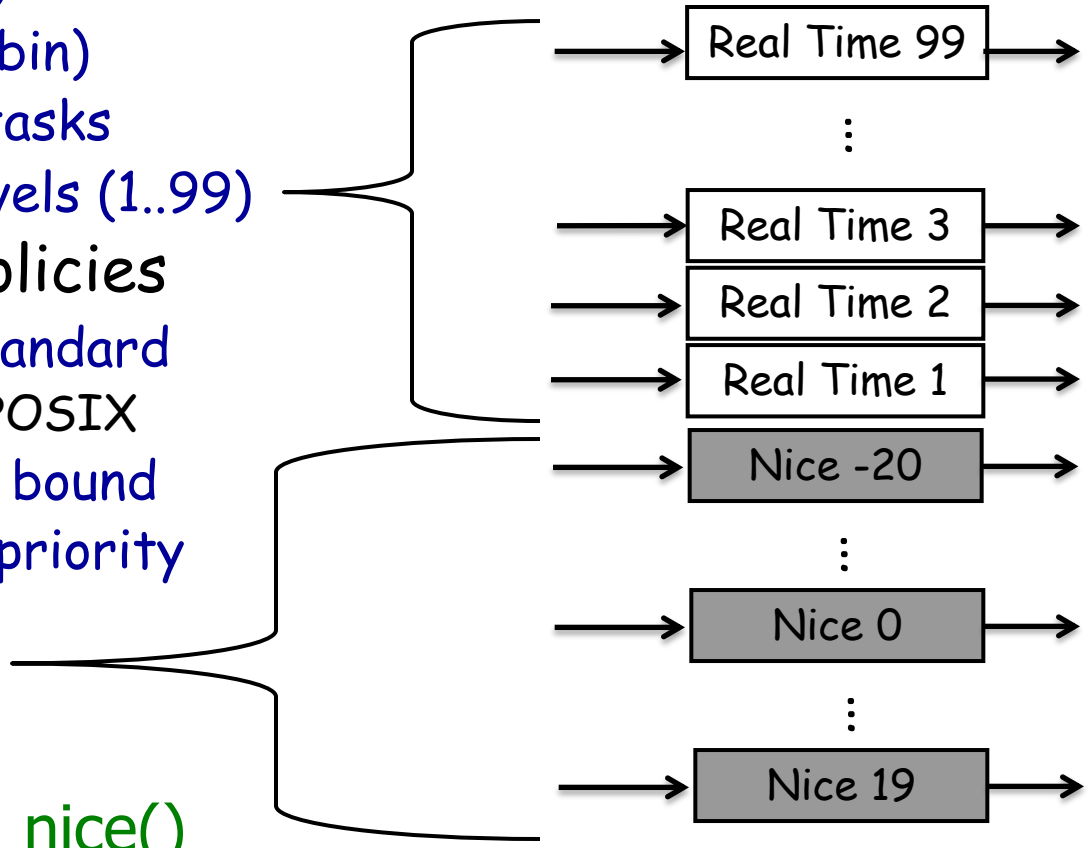
- Processor Affinity
  - Add process to a CPU's queue if recently run on the CPU
    - Cache state may still present

Real-time scheduling

- Real-time processes have timing constraints
  - Expressed as deadlines or rate requirements
  - Ex) gaming, video/music player, autopilot
- Hard real-time systems – required to complete a critical task within a guaranteed amount of time
- Soft real-time computing – requires that critical processes receive priority over others
- Linux supports soft real-time

Linux: multi-level queue with priorities

- Soft real-time scheduling policies
  - SCHED_FIFO (FCFS)
  - SCHED_RR (round robin)
  - Priority over normal tasks
  - 100 static priority levels (1..99)
- Normal scheduling policies
  - SCHED_NORMAL: standard
    - SCHED_OTHER in POSIX
  - SCHED_BATCH: CPU bound
  - SCHED_IDLE: lower priority
  - Static priority is 0
    - 40 dynamic priority
    - "Nice" values
- sched_setscheduler(), nice()
- See "man 7 sched" for detailed overview

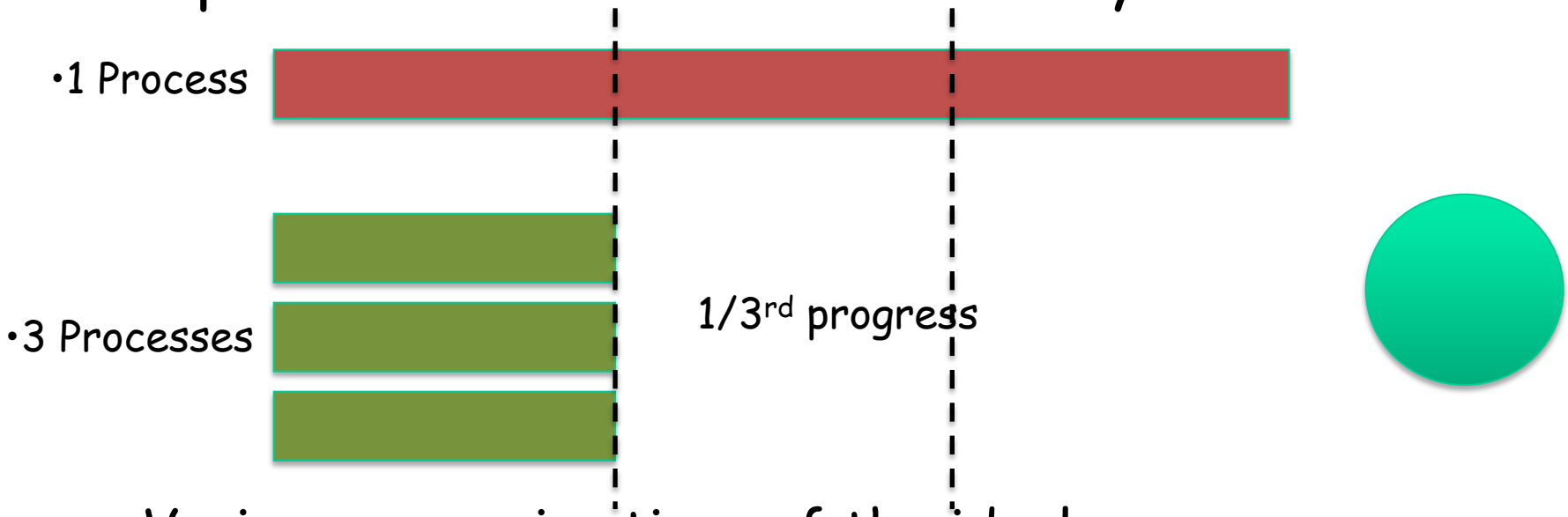


Linux scheduler history

- O(N) scheduler up to 2.4
  - Simple: global run queue
  - Poor performance on multiprocessor and large N
- O(1) scheduler in 2.5 & 2.6
  - Good performance: per-CPU run queue
  - Complex and error prone logic to boost interactivity
  - No fairness guarantee
- Completely Fair Scheduler (CFS) in 2.6 and later
  - Currently default scheduler for SCHED_NORMAL
  - Processes get fair share of CPU
  - Naturally boosts interactivity
- BFS and MuQSS
  - Linux scheduler for hippies
  - Available as kernel patches on the street

Ideal fair scheduling

- Infinitesimally small time slice
- n processes: each runs uniformly at $1/n^{th}$ rate



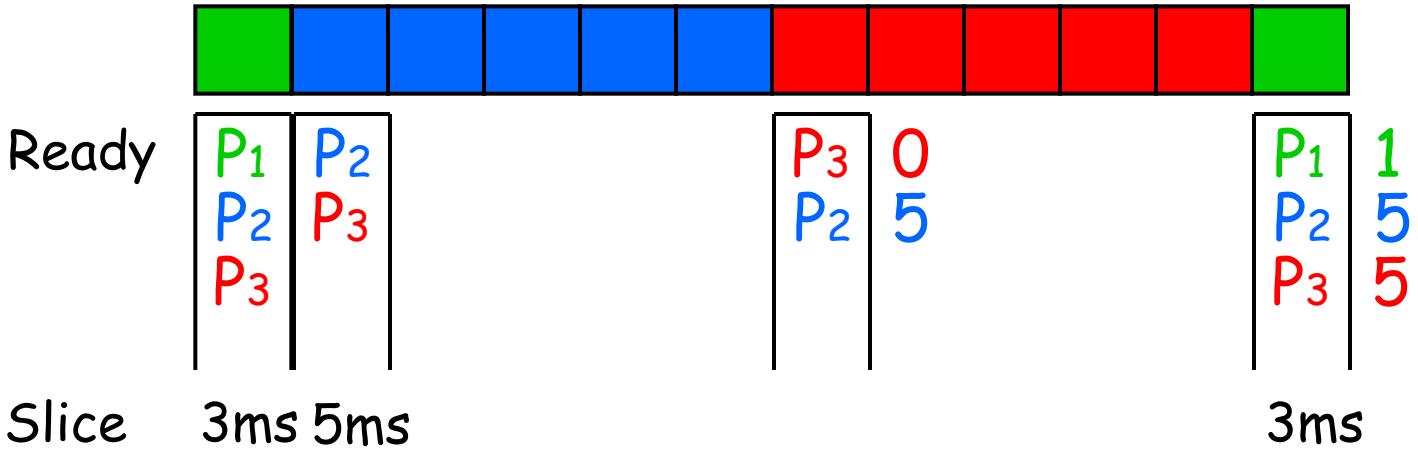
- Various approximations of the ideal
  - Lottery scheduling
  - Stride scheduling
  - Linux CFS

Completely Fair Scheduler (CFS)

- Approximate fair scheduling
  - Run each process once per schedule latency period
    - sysctl_sched_latency
  - Time slice for process Pi: T * Wi/(Sum of all Wi)
    - sched_slice()
- Too many processes?
  - Lower bound on smallest time slice
  - Schedule latency = lower bound * number of procs
- Introduced in Linux 2.6.23

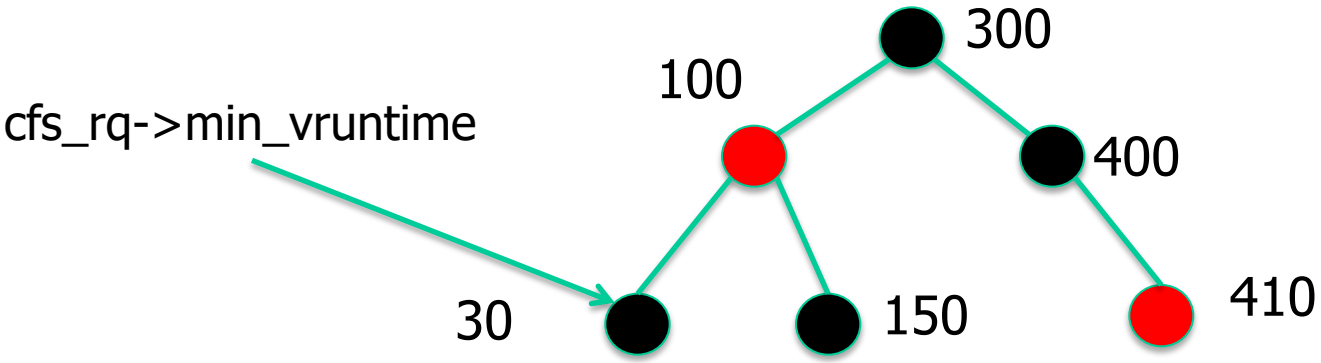
Picking the next process

- Pick proc with weighted minimum runtime so far
  - Virtual runtime: task->vruntime += executed time / Wi
- Example
  - P1: 1 ms burst per 10 ms (schedule latency)
  - P2 and P3 are CPU-bound
  - All processes have the same weight (1)



Finding proc with minimum runtime fast

- Red-black tree
  - Balanced binary search tree
  - Ordered by vruntime as key
  - O(lgN) insertion, deletion, update, O(1): find min



- Tasks move from left of tree to the right
- min_vruntime caches smallest value
- Update vruntime and min_vruntime
  - When task is added or removed
  - On every timer tick, context switch

Converting nice level to weight

- Table of nice level to weight
  - static const int prio_to_weight[40] (kernel/sched/sched.h)
- Nice level changes by 1 ➔ 10% weight
- Pre-computed to avoid
  - Floating point operations
  - Runtime overhead

Fsck all that…

Enter BFS

The scheduler that shall not be named

(now replaced by MuQSS, sadly…)

Hierarchical, modular scheduler

- Code from kernel/sched/core.c:

```
class = sched_class_highest;
for ( ; ; ) {
        p = class->pick_next_task(rq);
        if (p)
                return p;
        /*
         * Will never be NULL as the idle class always
         * returns a non-NULL p:
         */
        class = class->next;
}
```

sched_class Structure

```
static const struct sched_class fair_sched_class = {
        .next                   = &idle_sched_class,
        .enqueue_task           = enqueue_task_fair,
        .dequeue_task           = dequeue_task_fair,
        .yield_task             = yield_task_fair,
        .check_preempt_curr     = check_preempt_wakeup,
        .pick_next_task         = pick_next_task_fair,
        .put_prev_task          = put_prev_task_fair,
        .select_task_rq         = select_task_rq_fair,
        .load_balance           = load_balance_fair,
        .move_one_task          = move_one_task_fair,
        .set_curr_task          = set_curr_task_fair,
        .task_tick              = task_tick_fair,
        .task_fork              = task_fork_fair,
        .prio_changed           = prio_changed_fair,
        .switched_to            = switched_to_fair,
}
```

The runqueue

- All run queues available in array runqueues, one per CPU
- struct rq (kernel/sched/sched.h)
  - Contains per-class run queues (RT, CFS) and params
    - E.g., CFS: a red-black tree of task_struct (struct rb_root tasks_timeline)
    - E.g., RT: array of active priorities
    - Data structure rt_rq, cfs_rq,
- struct sched_entity (include/linux/sched.h)
  - Member of task_struct, one per scheduler class
  - Maintains struct rb_node run_node, other per-task params
- Current scheduler for task is specified by task_struct.sched_class
  - Pointer to struct sched_class
  - Contains functions pertaining to class (object-oriented code)

Adding a new Scheduler Class

- The Scheduler is modular and extensible
  - New scheduler classes can be installed
  - Each scheduler class has priority within hierarchical scheduling hierarchy
  - Linked list of sched_class sched_class.next reflects priority
  - Core functions: kernel/sched/core.c, kernel/sched/sched.h, include/linux/sched.h
  - Additional classes: kernel/sched/fair.c, rt.c
- Process changes class via sched_setscheduler syscall
- Each class needs
  - New runqueue structure in main struct rq
  - New sched_class structure implementing scheduling functions
  - New sched_entity in the task_struct