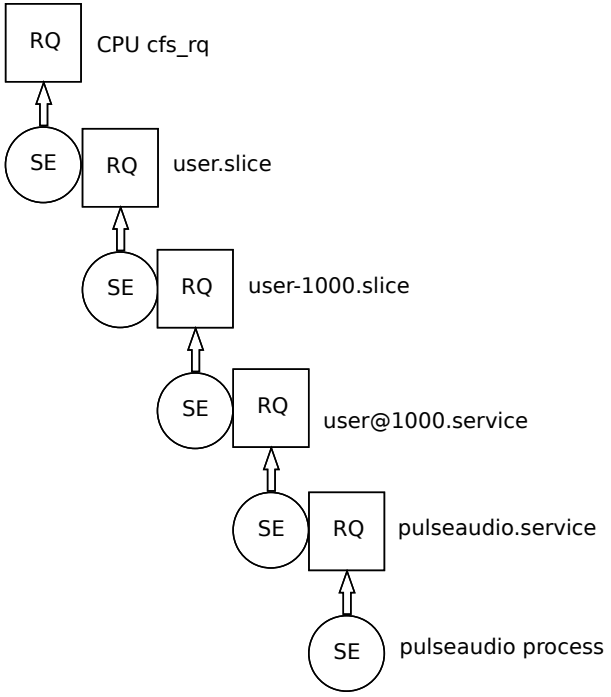
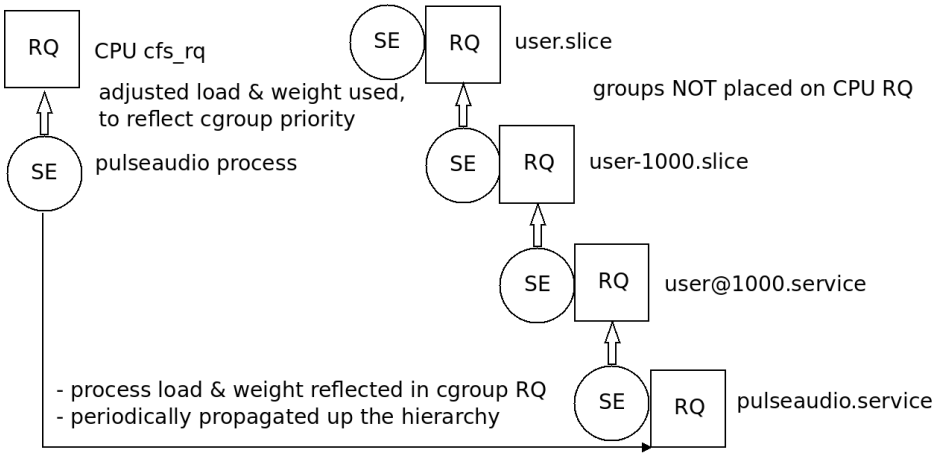
Current cgroup CPU controller

- Task has sched_entity (se)
- Group has se & cfs_rq
- Task se on group cfs_rq
- Group se on parent cfs_rq, etc...
- Build up entire hierarchy on wakeup
  - for_each_sched_entity() loops
  - Put each se on parent's cfs_rq, recalculate priorities
- Tear it back down when task sleeps
- Do vruntime accounting at each level, at every reschedule
- Preemption decisions re-evaluated at every level
- load_avg calculated periodically



New CPU controller

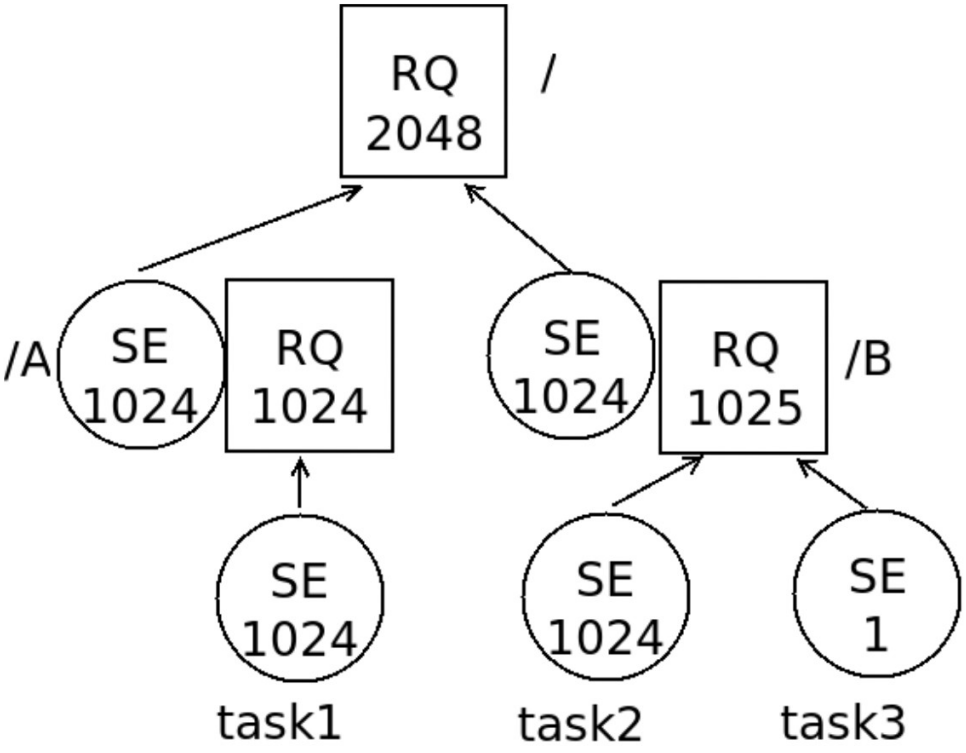
- All tasks on rq get same amount of vruntime
- Basic design
  - All tasks in root cfs_rq
  - Groups not placed on root cfs_rq
  - Rate limit hierarchy walks as much as possible
  - Use hierarchical load & weight for task priority
  - Scale vruntime with hierarchical task weight
  - Slight variation on vruntime formula




```
se→vruntime += (NICE_0_LOAD / task_se_h_weight(se)) *
delta_exec;
```

Corner case: uneven subgroups

- Top level same priority
- Subgroups/tasks different
- Tasks 1, 2 & 3 running
  - Groups A & B equal priority
  - Task 3 lower than 2
  - Task 3 runs, vruntime advanced
  - Task 2 goes to sleep, task 3 still waiting?!
  - Task 3 equal prio as task 1 when task 2 sleeps…
  - Need fast convergence



Solution: overflow runqueue

- Vruntime =+ delta_exec / prio
- Limit amount of vruntime accounted at once (to sched_latency?)
- Task cannot have all its delta_exec moved vruntime?
  - Move task to overflow/overloaded runqueue heap
  - Sorted by vruntime
  - In pick_next_entity, bring left-most entity in overloaded heap up to current vruntime, re-insert if still delta_exec remaining
  - If that task has all delta_exec accounted, move back to main rq
  - Skip that task for now, first run a task that was already on the main rq
- Move one task back at a time
- Do not starve tasks already on main rq

Issue: thundering herd wakeups

- Scenario:
  - 1 task running in cgroup A
  - 100 tasks waking up in cgroup B
  - How to keep task in cgroup A from starvation?
- Solution: admission control
  - Piggyback on overflow/overloaded rq heap
  - If, at wakeup time, a task's priority is such that it cannot run for sched_min_granularity_ns and account it all as vruntime …
  - … move it straight onto the overflow/overloaded rq heap
  - Apply same rules to this task as to other tasks on that heap
- Thanks to sched_slice and __sched_period this only applies to tasks with below average priority

CFS bandwidth plan

- When a cgroup is throttled, mark cgroup cfs_rqs as throttled (do not touch tasks)
- When pick_next_entity finds a task from a throttled cgroup
  - Remove from root cfs_rq, place on cgroup cfs_rq
  - Keep task vruntime intact, adjust cgroup min_vruntime
- When a cgroup is unthrottled
  - Mark cgroup cfs_rq unthrottled
  - Place unthrottled group on overflow/overloaded rq heap, using min_vruntime
- In pick_next_entity, if left-most entity on overflow/overloaded rq heap is a group
  - Grab task with smallest min_vruntime, remove cgroup cfs_rq from heap if empty
  - Adjust that task's vruntime to root cfs_rq min_vruntime + ½ a timeslice, place on root cfs_rq
  - Run smallest vruntime task on the root cfs_rq (may be other task than just woken one)
- Slow wakeup avoids "thundering herd" issues and minimizes work done
- Seems reasonable? What did I overlook?