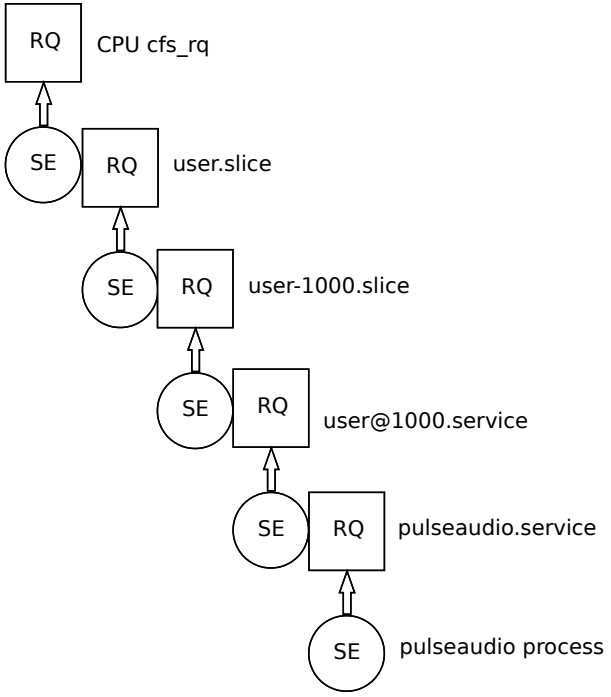
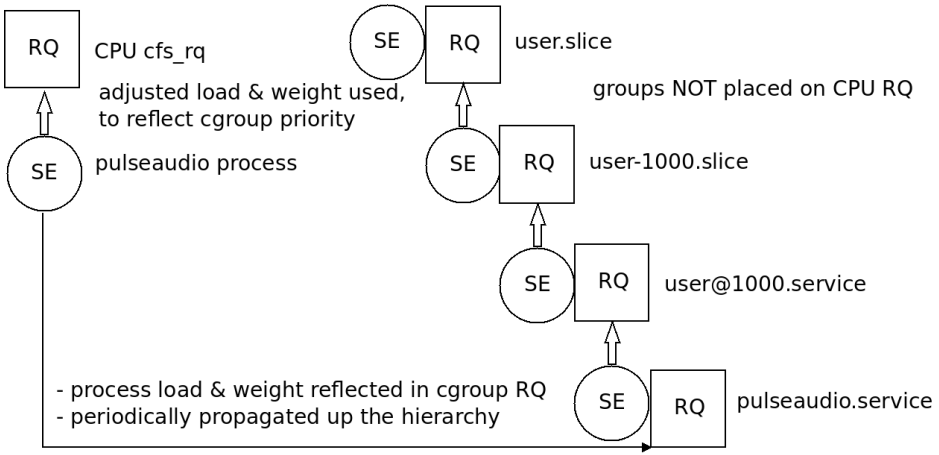
Current cgroup CPU controller

- 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

- 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;
```

Preemption

- Some problems
  - vdiff influenced by tasks other than curr
  - High priority task can preempt anyone, even higher priority tasks!
  - Not a big problem today, but with flat rq will have wildly different priority tasks involved more often
- What should preempt do?
  - Lets figure out “what” before “how”

```
vdiff = curr→vruntime –
woken→vruntime;

gran =
calc_delta_fair(sched_wak
eup_granularity, woken);

if (vdiff > gran)

    return 1; /* preempt curr */
```

Remaining hierarchy walk stuff

- Ramp-up logic in calc_group_shares
  - Need similar logic in update_cfs_rq_h_load for h_load/h_weight?
  - Proper priority & vruntime scaling when task is woken up
  - Avoid load balancer confusion with zero weight task?
- Enqueue_task_fair forces propagate_entity_cfs_rq to walk hierarchy every time
  - Avoids zero group shares on wakeup
  - Most remaining CPU controller overhead in this path with memcache style workload
  - How can we reduce this?
    - Skip when DO_ATTACH, but group has non-zero shares already?
    - Skip when DO_ATTACH, but last cfs_rq / shares decay happened when group nr_running > 0?
    - ...

CFS bandwidth issues

- Old implementation
  - Tasks are on cgroup runqueues
  - Remove entire runqueues when group is throttled
- New implementation
  - Tasks are on root runqueue
  - How to avoid / reduce O(N) issues?
    - Be very, very, very lazy

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
  - Have unthrottled cgroup cfs_rqs in heap on root cfs_rq sorted by min_vruntime
- In pick_next_entity, check for non-empty unthrottled heap
  - 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?