Operating System Project3

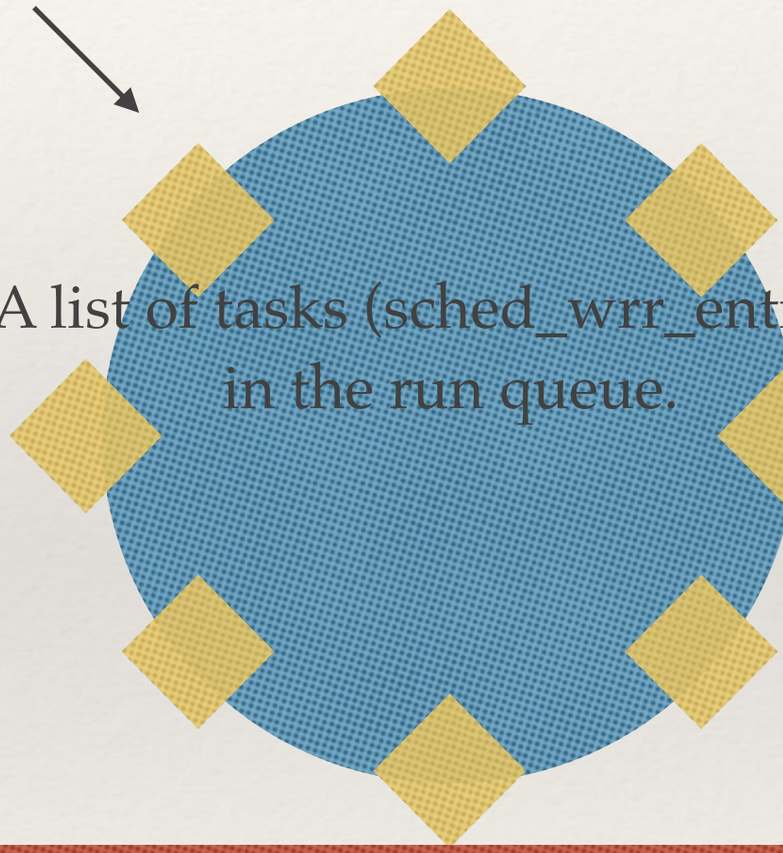
WRR Scheduler

Team 1
Sung-Yun Hur
Eun-Hyang Kim
Yeon-Woo Kim

wrr_rq Structure

- struct wrr_rq {
- unsigned long total_weight;
- struct list_head run_queue;
- struct task_struct* curr;
- };

A cursor that points to the currently running task.



Total weight of the tasks in the run queue.

sched_wrr_entity Structure

- struct sched_wrr_entity {
- struct list_head run_list;
- unsigned int weight;
- unsigned int time_slice;
- };

My timeslice

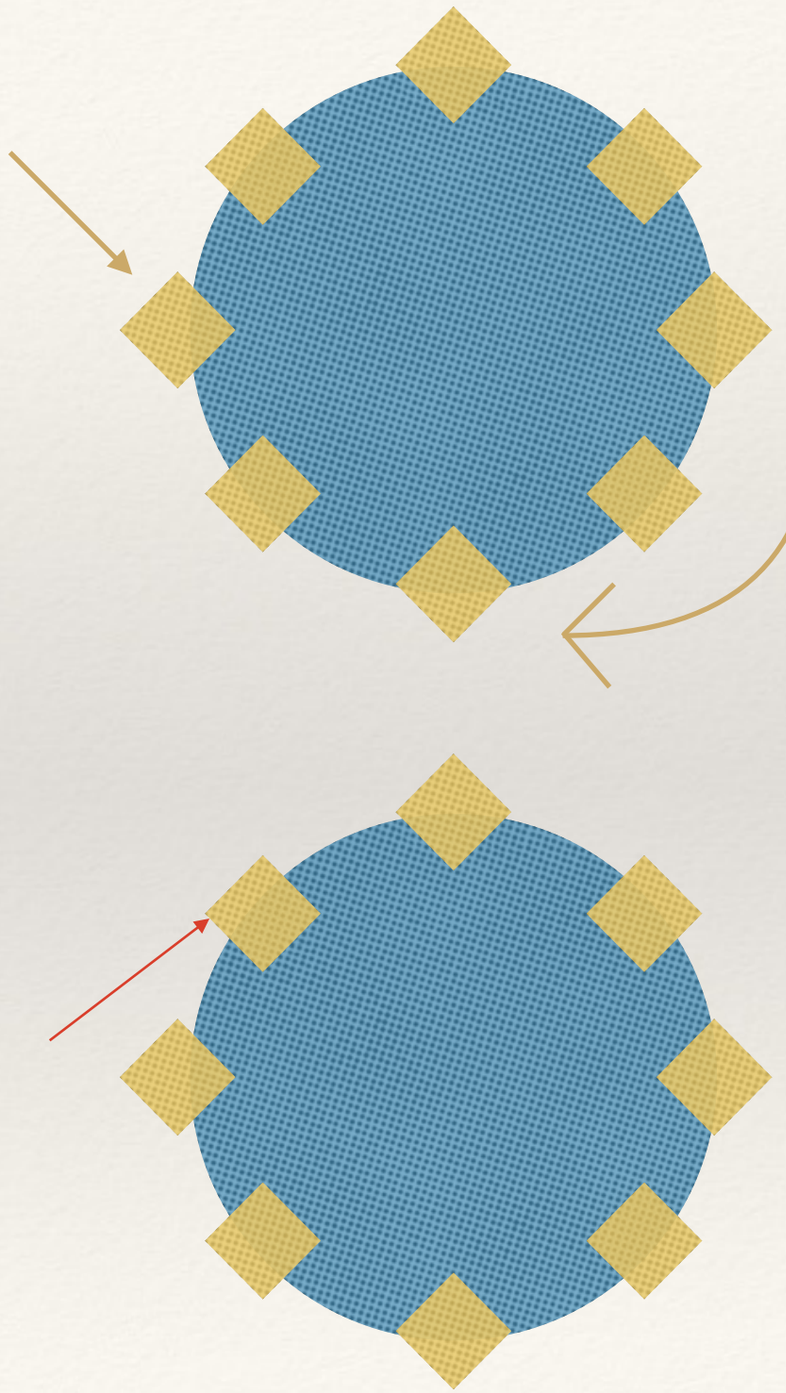
My weight

En/Dequeue and pick next task

- Enqueue: if no task in the run queue, add the task in to the run queue and set the cursor to the added task. Else add the task right before the cursor.
- Dequeue: if the task that the cursor is pointing to needs to be deleted, update the cursor to the next task. Else, simply delete the task.
- Pick next task: return the task that the cursor is pointing to with updated time slice.

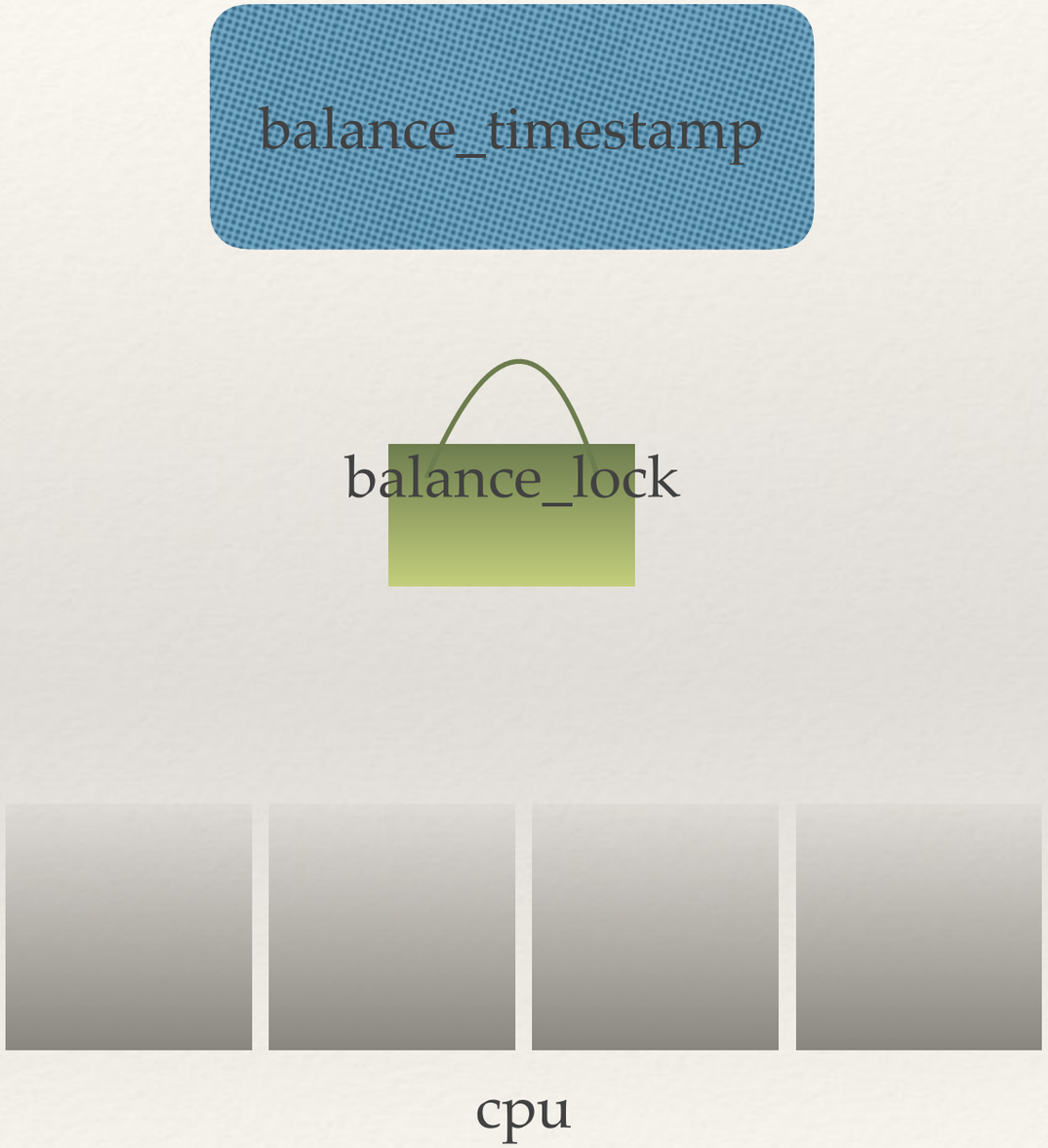
Task tick!

- Decrease the time slice of the task the cursor is pointing to.
- If all the time slice is consumed, move the cursor to next and reschedule.
- If currently running task is the only task in the queue, refill the time slice and return.



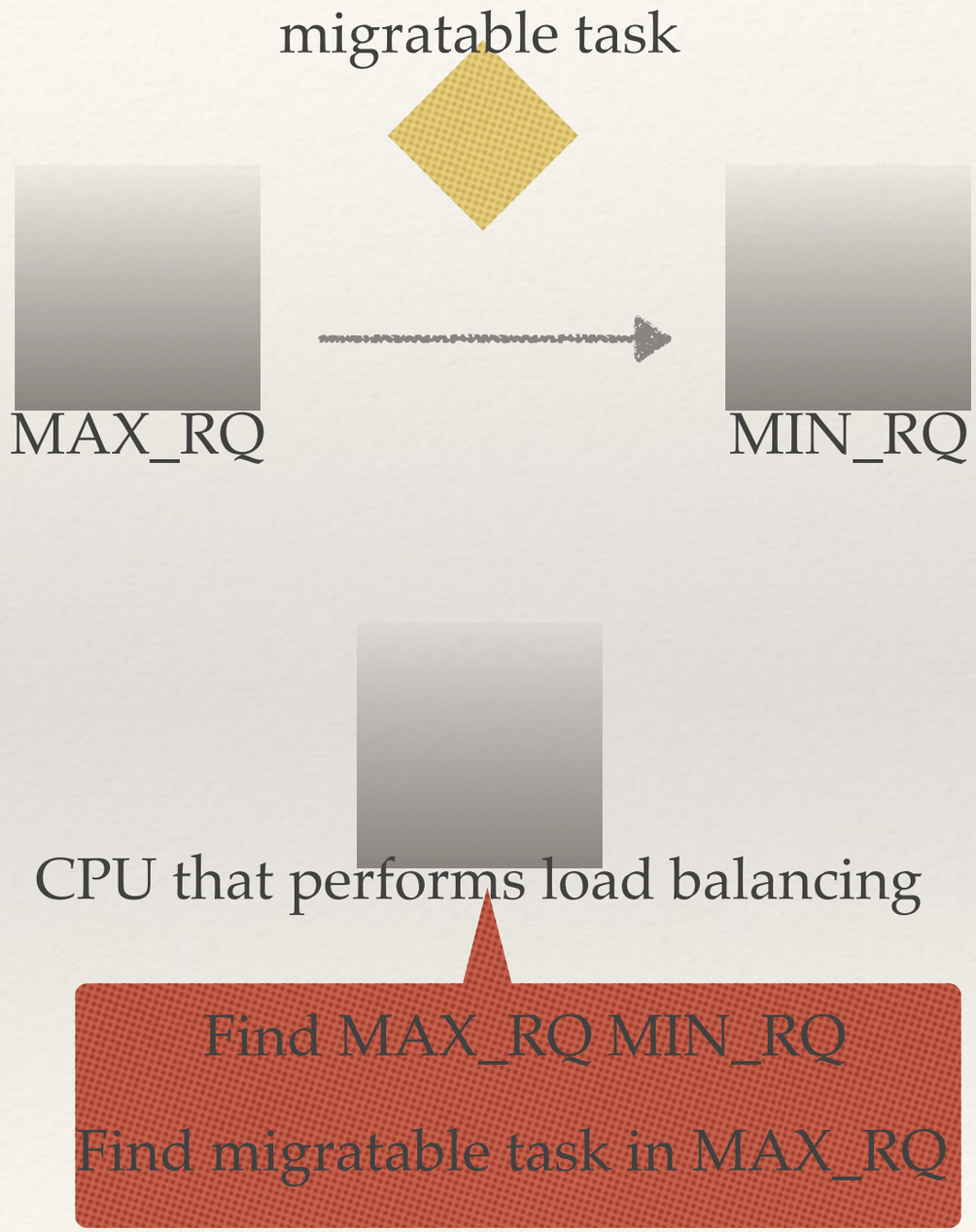
Load Balancing

- A global variable stores the time of last load balancing.
- At each tick, cpus that are online try to get the lock, check if it is time to do next load balancing, and update the timestamp to now.



Load Balancing contd.

- The cpu that succeeded in getting the lock and updating the timestamp then starts load balancing
- It finds max_rq and min_rq, checks if there is a migratable task; if there is, it acquires lock for the two run queues and migrates the task.
- Additionally, we have an internal lock in wrr_rq and wrr_rq gets the lock whenever a task is to be enqueued or dequeued.



Questions?