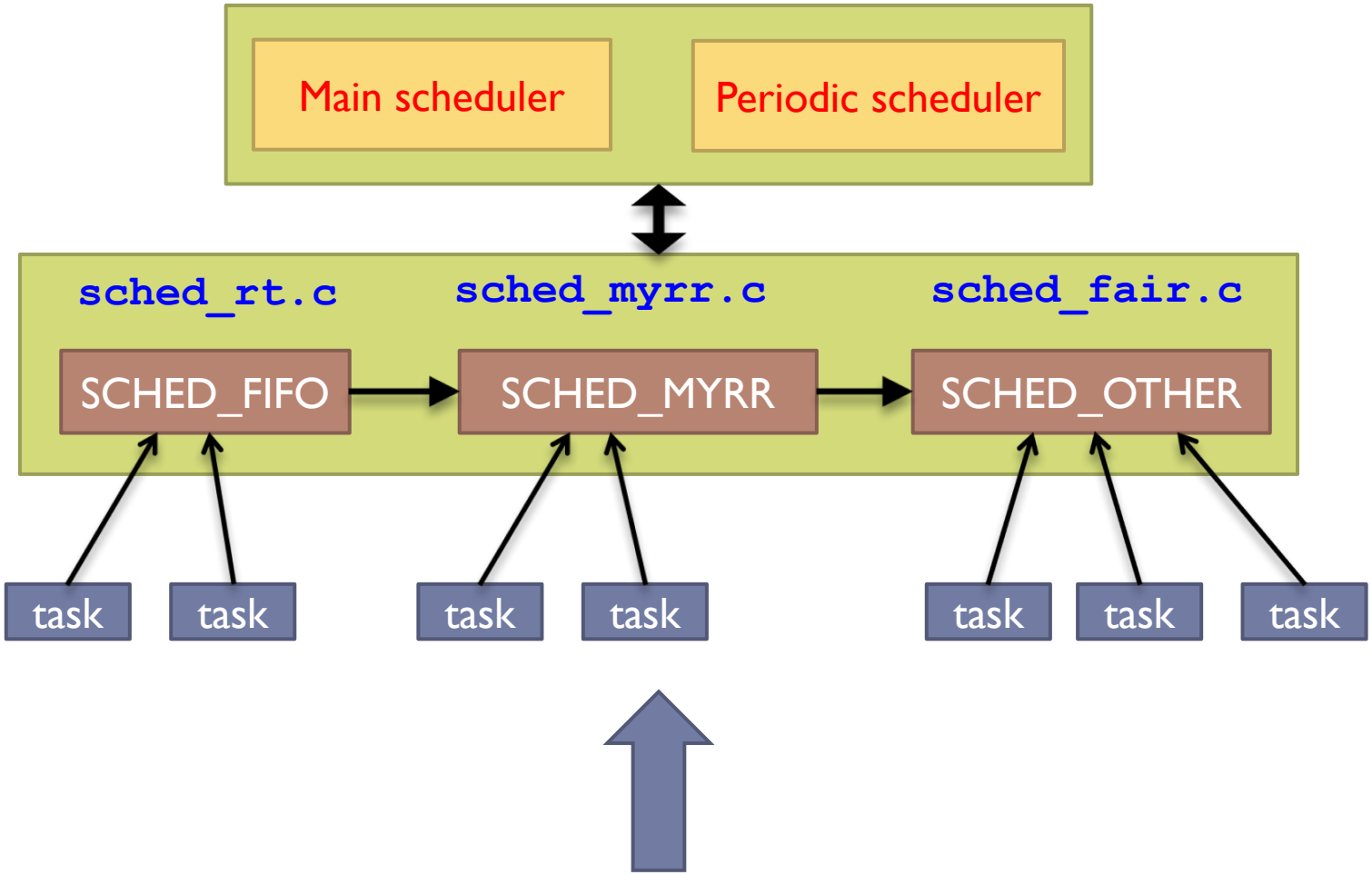
Implement a Round Robin Scheduler

What is Round-robin scheduling

- Round-robin scheduler gives each job a time slice
  - When we implement the entity class (i.e. `sched_myrr_entity`), a new attribute called `time_slice` is added
  - When
    - The given `time_slice` is expired AND
    - There exists some other tasks in the queue
    - The current task will be moved to the end of the queue

SCHED_MYRR



Now, we replaced MYFIFO with MYRR

Implementation of SCHED_MYRR

Step-1: define scheduling policy no.

- Locate the following line

```
#define SCHED_IDLE 5
```

in `include/uapi/linux/sched.h`

- After the above line, add the following line

```
#define SCHED_MYRR  6
```

- If you want to keep the previous `MY_FIFO` scheduler policy number, you can set `SCHED_MYRR` as 7

Step-2: define the scheduler entity

- In `include/linux/sched.h`, locate the declaration of `struct sched_rt_entity`, after which we define `struct sched_myrr_entity` to encapsulate needed information to help implement the new scheduling policy

```
struct sched_myrr_entity {
        struct list_head run_list;
        int time_slice;  /* new field is introduced */
};
```

- Locate the definition of `struct task_struct`, add the following line

```
struct sched_myrr_entity myrr;
```

after the line

```
struct sched_rt_entity rt;
```

Step-3: define a new runqueue

- In `kernel/sched/sched.h`, we define `struct myrr_rq`

```
struct myrr_rq {
        struct list_head queue;
};
```

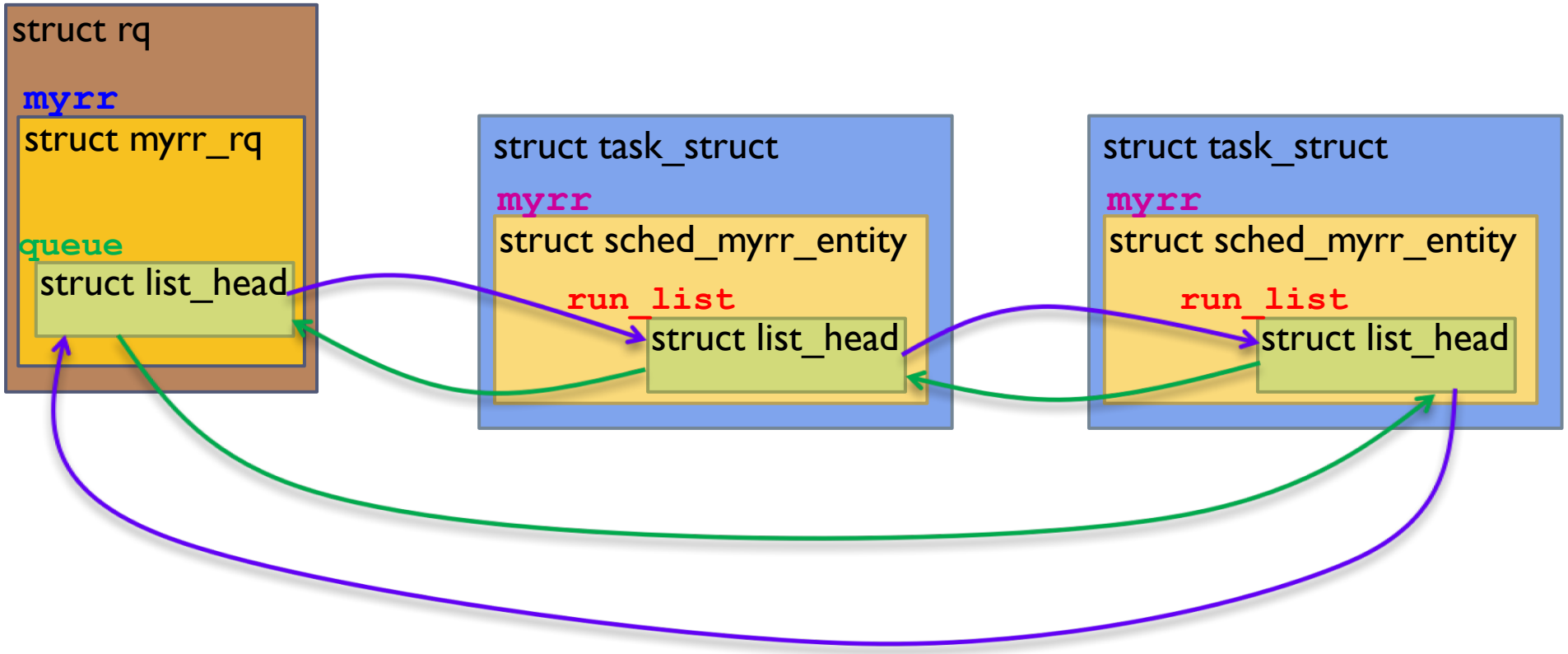
- Locate the definition of `struct rq`, add the following line

```
struct myrr_rq myrr;
```

after the line

```
struct rq_rq rt;
```

Structure


struct rq
myrr
struct myrr_rq
queue
struct list_head
struct task_struct
myrr
struct sched_myrr_entity
run_list
struct list_head
struct task_struct
myrr
struct sched_myrr_entity
run_list
struct list_head

Step-4: Add implement schedule_myrr.c

- Add the definition of myrr_sched_class in `kernel/sched/sched.h`
  - Add `extern const struct sched_class myrr_sched_class`
  - After `extern const struct sched_class idle_sched_class`

Step-4: implement sched_myrr.c

- Create a new file **kernel/sched/sched_myrr.c**

```
#include "sched.h"
static const struct sched_class myrr_sched_class = {
    .next = &fair_sched_class,
    .enqueue_task = enqueue_task_myrr,
    .dequeue_task = dequeue_task_myrr,
    .yield_task = yield_task_myrr,
    .check_preempt_curr = check_preempt_curr_myrr,
    .pick_next_task = pick_next_task_myrr,
    .put_prev_task = put_prev_task_myrr,

#ifdef CONFIG_SMP
    .select_task_rq = select_task_rq_myrr,
    .load_balance = load_balance_myrr,
    .move_one_task = move_one_task_myrr,
#endif

    .set_curr_task = set_curr_task_myrr,
    .task_tick = task_tick_myrr,
    .get_rr_interval = get_rr_interval_myrr,
    .prio_changed = prio_changed_myrr,
    .switched_to = switched_to_myrr,
};
```

We should have disabled SMP in the previous lab, we can ignore this part

Step-4: implement sched_myrr.c

```
static void enqueue_task_myrr(struct rq *rq, struct task_struct *p,
                              int wakeup, bool head)
{
    struct sched_myrr_entity *myrr_se = &p->myrr;
    list_add_tail(&myrr_se->run_list, &rq->myrr.queue);

    printk(KERN_INFO"[SCHED_MYRR] ENQUEUE: Process-%d\n", p->pid);
}

static void dequeue_task_myrr(struct rq *rq,
                              struct task_struct *p, int sleep)
{
    struct sched_myrr_entity *myrr_se = &p->myrr;
    list_del(&myrr_se->run_list);
    printk(KERN_INFO"[SCHED_MYRR] DEQUEUE: Process-%d\n", p->pid);
}
```

Step-4: implement sched_myrr.c

```
static void yield_task_myrr(struct rq *rq)
{
    struct sched_myrr_entity *myrr_se = &rq->curr->myrr;
    struct myrr_rq *myrr_rq = &rq->myrr;
    list_move_tail(&myrr_se->run_list, &myrr_rq->queue);
}

static void check_preempt_curr_myrr(struct rq *rq,
    struct task_struct *p, int flags)
{
    if (rq->curr->policy == SCHED_FIFO || rq->curr->policy == SCHED_RR)
        return ;
    if (rq->curr->policy == SCHED_MYRR)
        return ;
    /* preempt normal tasks */
    resched_task(rq->curr);
}
```

Basically the same as MY_FIFO

Step-4: implement sched_myrr.c

```
static struct task_struct *pick_next_task_myrr(struct rq *rq)
{
    struct sched_myrr_entity *myrr_se = NULL;
    struct task_struct *p = NULL;
    struct myrr_rq *myrr_rq = &rq->myrr;
    if (list_empty(&myrr_rq->queue))
        return NULL;
    myrr_se = list_entry(myrr_rq->queue.next,
                         struct sched_myrr_entity,
                         run_list);
    p = container_of(myrr_se, struct task_struct, myrr);
    return p;
}

static void put_prev_task_myrr(struct rq *rq, struct task_struct *p)
{
    /* it is the place to update the current task's
     * runtime statistics */
}
```

Step-4: implement sched_myrr.c

```
#ifdef CONFIG_SMP
static int select_task_rq_myrr(struct rq *rq, struct task_struct *p,
                               int sd_flag, int flags)
{
        return task_cpu(p);
}

static unsigned long load_balance_myrr(struct rq *this_rq,
        int this_cpu, struct rq *busiest, unsigned long max_load_move,
        struct sched_domain *sd, enum cpu_idle_type idle,
        int *all_pinned, int *this_best_prio)
{
        return 0;
}

static int move_one_task_myrr(struct rq *this_rq, int this_cpu,
        struct rq *busiest, struct sched_domain *sd,
        enum cpu_idle_type idle)
{
        return 0;
}
#endif
```

We should have disabled SMP in the previous lab, we can ignore this part

Step-4: implement sched_myrr.c

```
static void set_curr_task_myrr(struct rq *rq)
{
}

static void task_tick_myrr(struct rq *rq,
                           struct task_struct *p, int queued)
{
    if (p->policy != SCHED_MYRR) return ;

    /* Do your implementation here!!!!! */
    /* Hint: make use of time_slice attribute
       Hint: How can you access this attribute via p?
       Hint: How can you move a task to the end of the queue?
       Hint: Once the task is moved, a special function call
         set_tsk_need_resched(p) is required to be invoked!
    */

}
```

Function	Purpose
set_tsk_need_resched(p)	Set the `need_resched` flag in the given process

Step-4: implement sched_myrr.c

```
unsigned int get_rr_interval_myrr(struct rq *rq,
                                  struct task_struct *p)
{
   /* ToDo: return a default timeslice */
}

static void prio_changed_myrr(struct rq *rq, struct task_struct *p,
                              int oldprio, int running)
{
}

static void switched_to_myrr(struct rq *rq, struct task_struct *p,
                             int running)
{
}
```

Step-5: merge into the kernel

- Please follow the similar steps in the previous lab (i.e. `SCHED_MYFIFO`) to merge the new scheduler to the kernel
- Do we need to change the Makefile(s)?
- The steps may not be exactly the same (e.g. you need to change from myfifo to myrr in many places)
- Important:
  - When you initialize `myrr_entity` inside the `__sched_fork` function, you also need to initialize the attribute of `time_slice`
  - For example:
    - `INIT_LIST_HEAD(&p->myrr.run_list);`
    - `p->myrr.time_slice = /* Your default time slice */;`

Step-6: write the test program

- Make some changes to the test program in the previous lab. For example:
  - `ret = sched_setscheduler(0, SCHED_MYFIFO, &sp);`
  - `/* Now, we should replace it by what? */`

Lab demo:

- After running the test program:
  - Show the kernel log message (we should have added appropriate `printk` function calls in `enqueue/dequeue` functions) to demonstrate that the round-robin scheduler is executed