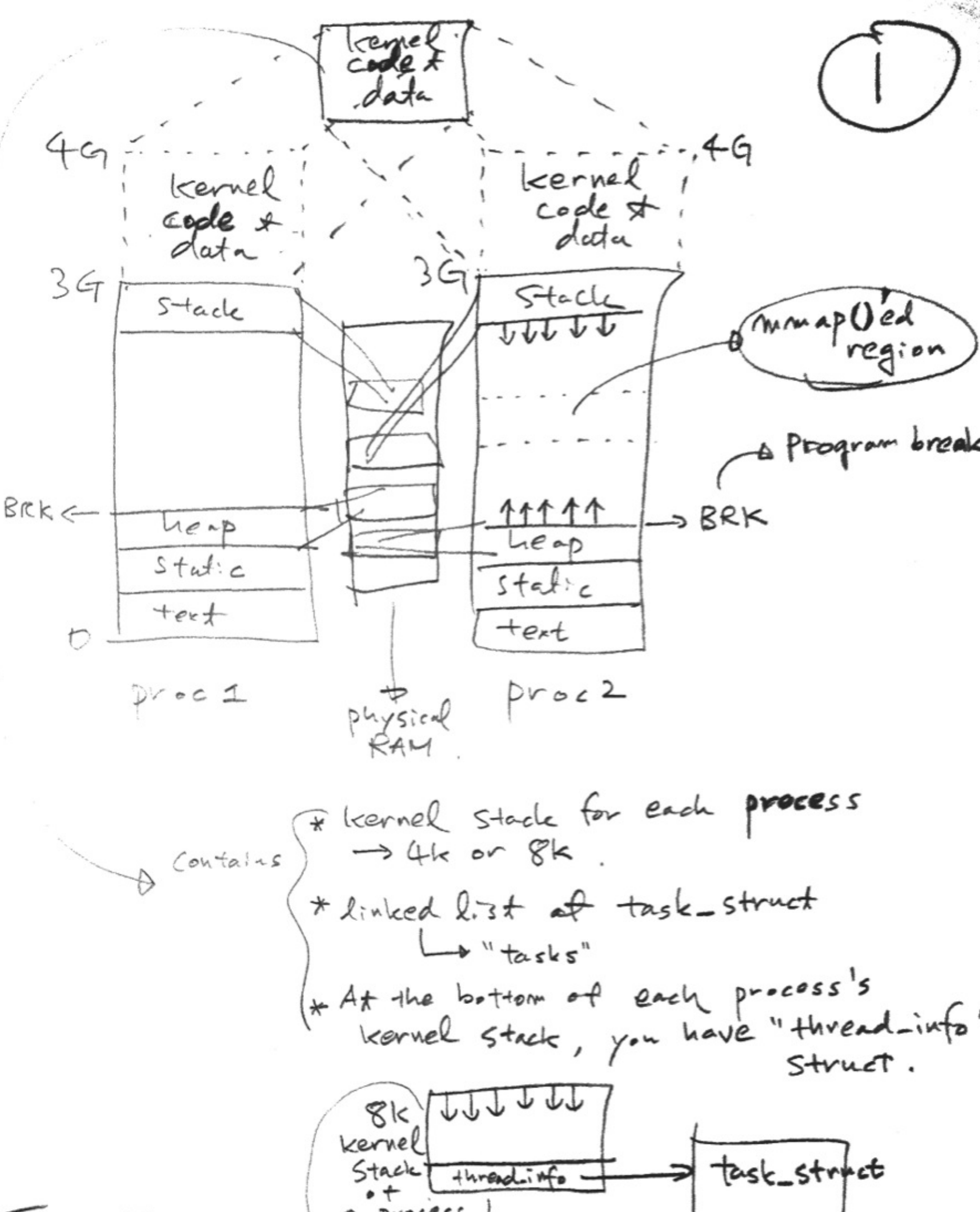
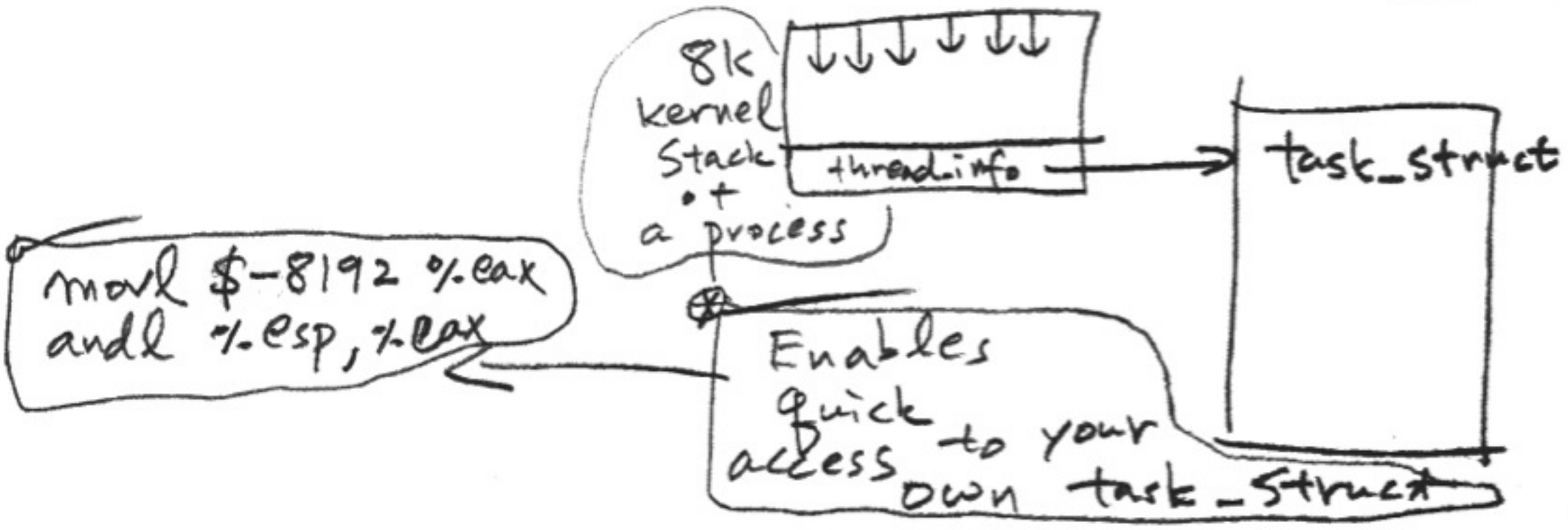
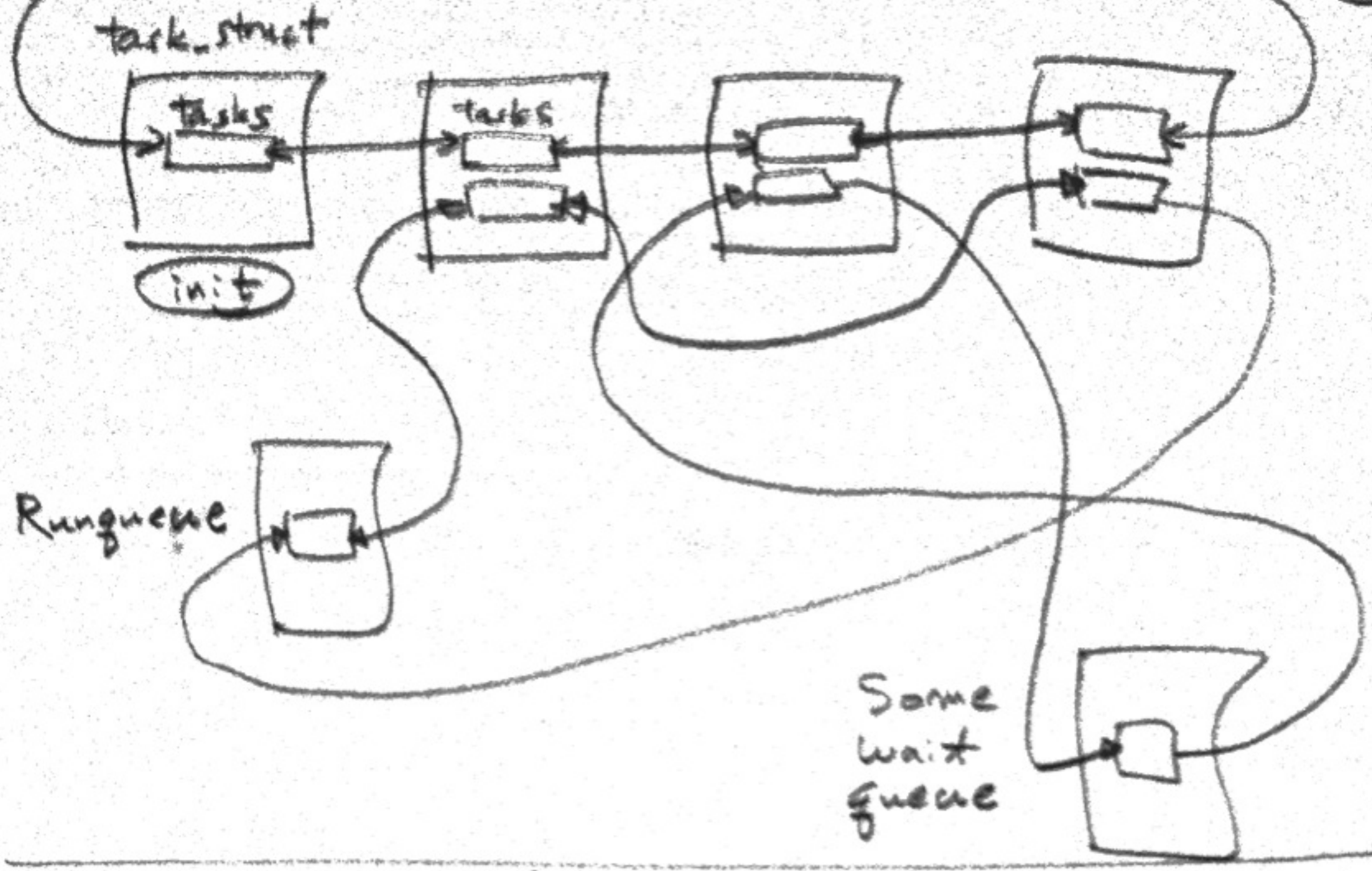
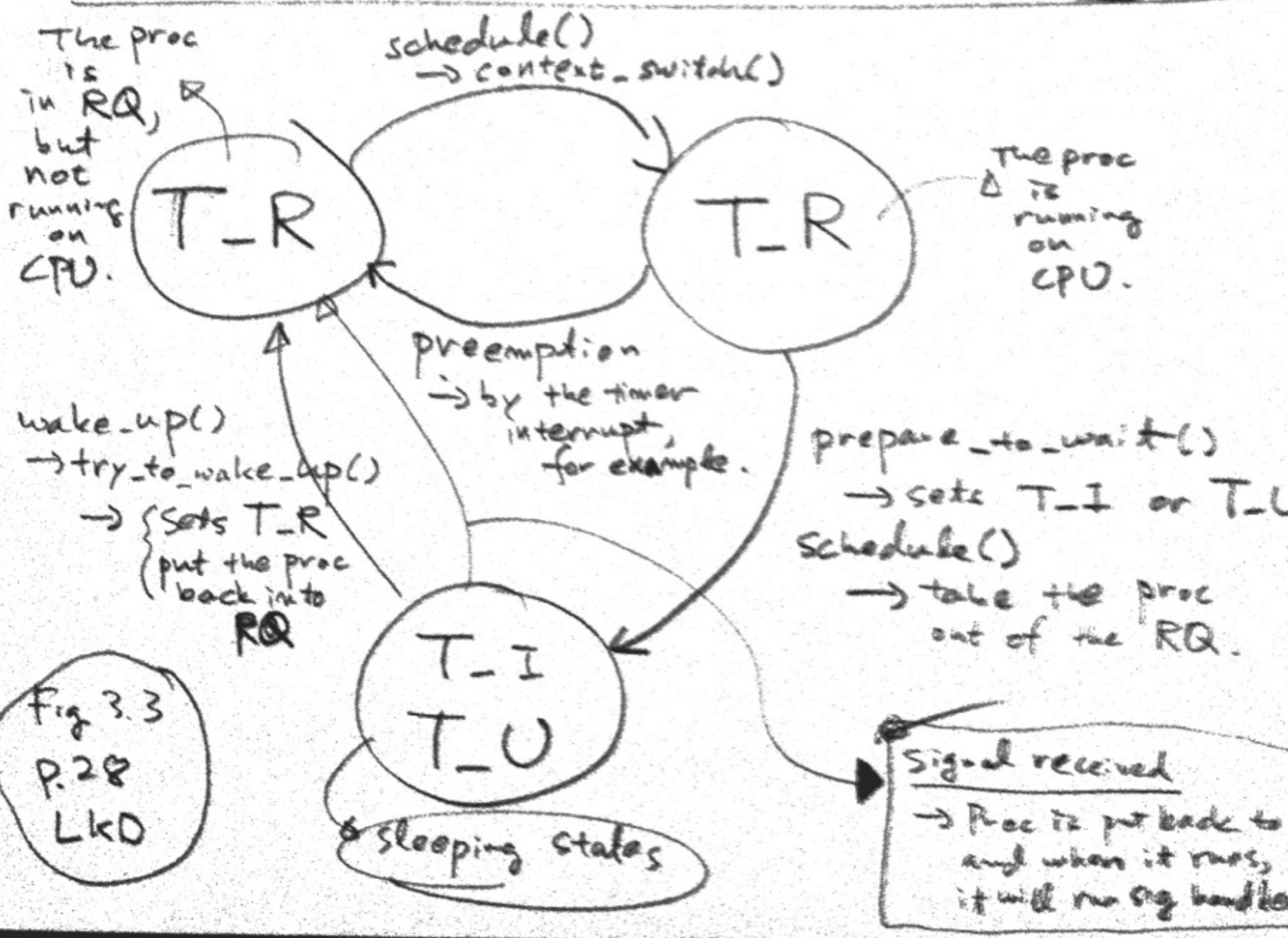


1

Contains
* kernel stack for each process
  → 4k or 8k.
* linked list of task_struct
  → "tasks"
* At the bottom of each process's kernel stack, you have "thread_info" struct.



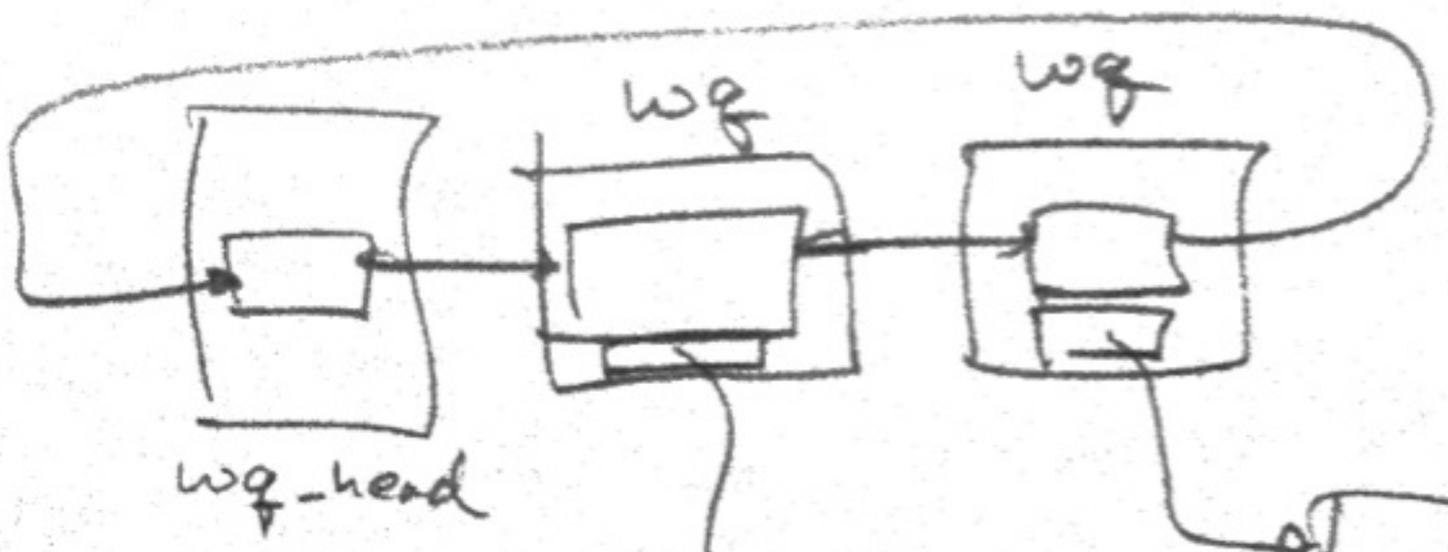

task_struct
tasks
tasks
init
Runqueue
Some wait queue

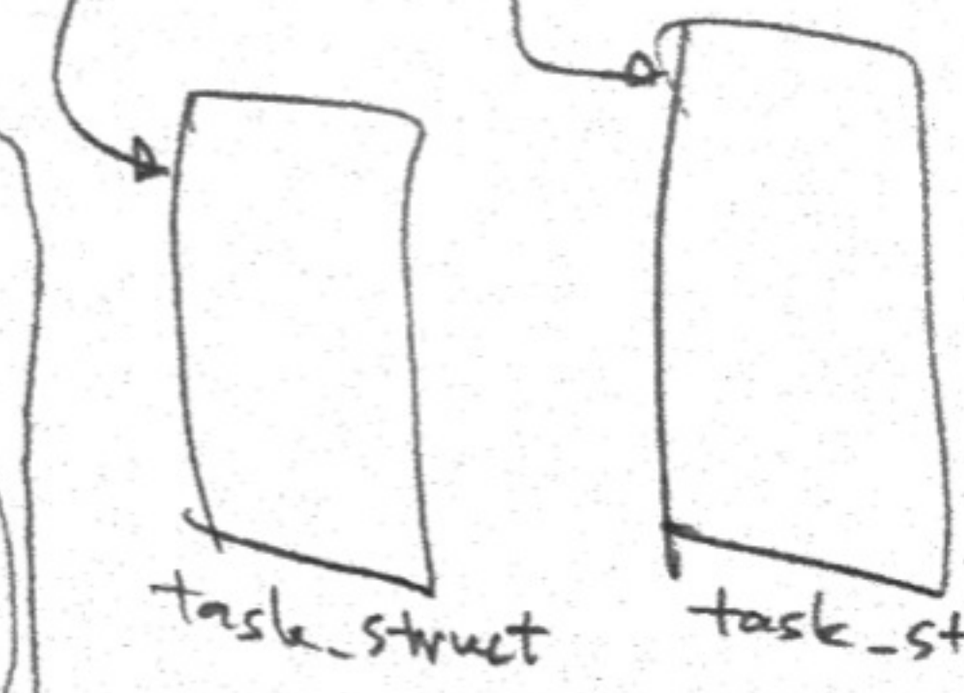

The proc is in RQ, but not running on CPU.
schedule()
→ context_switch()
T_R
T_R
The proc is running on CPU.
preemption
→ by the timer interrupt, for example.
wake_up()
→ try_to_wake_up()
→ {Sets T_R
put the proc back into RQ
prepare_to_wait()
→ sets T_I or T_U
schedule()
→ take the proc out of the RQ.
T_I
T_U
Sleeping states
Signal received
→ Proc is put back to RQ and when it runs, it will run sig handler.
Fig 3.3
P.28
LkD

Wait queues (pseudo code)


```
struct wq_head {
        spin_lock_t lock;
        list_head   task_list;
};

struct wq {
        task_struct * task;
        wq_func  func; // call back function
        list_head  task_list;
};
```






```
Void wait_event(wq, cond)
{
    DEF_WAIT(wait);
    for (;;) {
        prepare_to_wait(
                &wq,
                &wait,
                T_I    );
        if (cond)
            break;
        schedule();
    }
    finish_wait(&wq, &wait);
}
```

See LKD p59 for explanation

"wait_event" is a real function in Linux kernel. Skeleton pseudocode is shown here