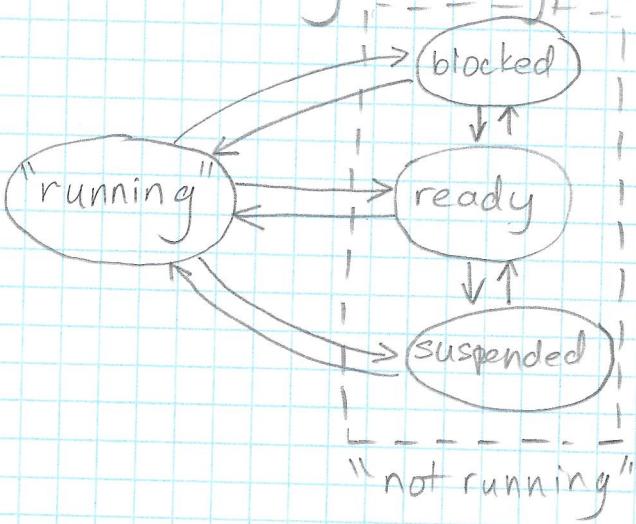
RTOS Scheduling

Recap

- For our development here, we will use the following symbols frequently:
  - C_i : the execution time needed to complete task/service S_i; note that this does not include preemption delays — it is the "raw" execution time needed to complete S_i without interruptions.
  - T_i : the task period; the fixed length of time between task calls or "releases".
  - D_i : the deadline for task S_i; we must have $D_i \leq T_i$ in order to have no missed deadlines.

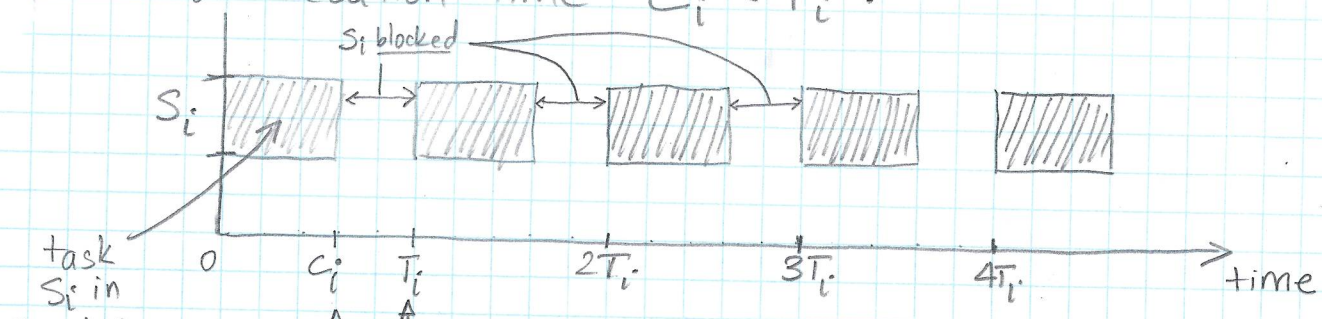
Task States

- Generally tasks are either "running" or "not running" — if a task is not running it may be "blocked", "suspended", or "ready" to run.
- IMPORTANT: When the scheduler is selecting which task to run, it selects from amongst those tasks in the ready state or already running, based on priority.



EXAMPLE:

Consider a task S_i with task period T_i and execution time $C_i < T_i$:



once task S_i completes, it enters the blocked state

as the task period ends, S_i goes from the blocked state to the ready state and may execute again — if it is the highest priority task in the ready state, it begins execution immediately

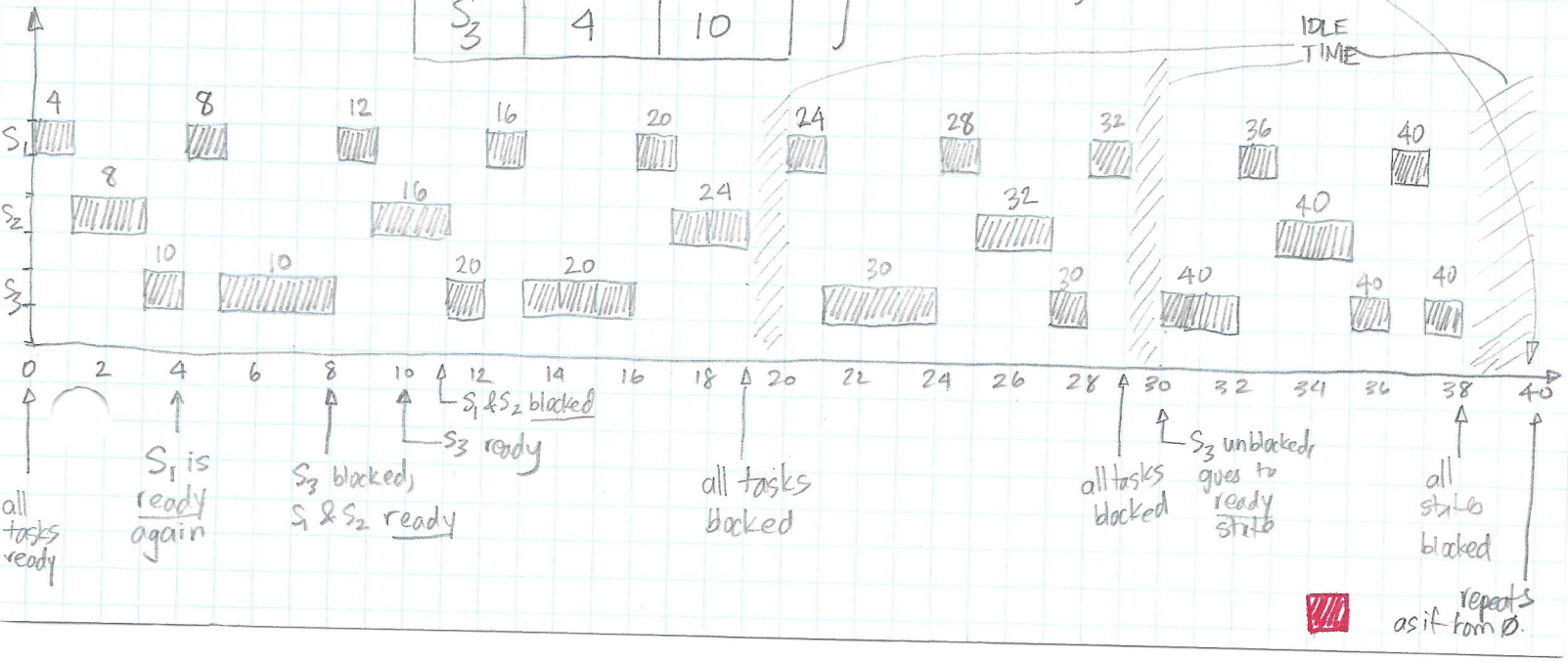
EXAMPLE:

Apply the RM fixed priority assignment policy to the following set of tasks:

	C_i	T_i
S_1	1	4
S_2	2	8
S_3	4	10

prior (S_1) > prior (S_2) > prior (S_3)

LCM (4, 8, 10) = 40



- in this example, we see the presence of idle time, suggesting that the CPU is not fully utilized

- also note that the behaviour of this system is captured completely over the LCM time of all task periods — we can see that there are no missed deadlines

Estimating Feasibility

- In RTOS theory, there is a sufficient (but not necessary) simple test of scheduling feasibility; that is, given N tasks S_i, can scheduling be achieved with no missed deadlines using the RM policy? This test is called the RM LUB (least upper bound):

- Theorem: Given the utility of the CPU, $U := \sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right)$, if $U \leq N(2^{1/N} - 1)$, then RM priority assignment will produce no missed deadlines. ▨

  (stated w/o proof)

- in the previous example,

  - $$U = \sum_{i=1}^{3} \left(\frac{C_i}{T_i}\right) = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} = \frac{1}{4} + \frac{2}{8} + \frac{4}{10}$$
    $$= \frac{1}{2} + \frac{2}{5} = \frac{5+4}{10} = \frac{9}{10} = 90\%$$

and

  - $$N(2^{1/N} - 1) = 3(2^{1/3} - 1) \simeq 0.78$$

and so

$$U = 0.90 \not\leq 0.78$$

showing that the RM LUB is conservative, because we have shown that RM policy, in fact, works with no missed deadlines.

- note that our example, however, had close to 100% CPU utilization (almost no idle time).
- if we "relax" our parameters further, say, set $C_3 = 2$, in order to give the scheduler "more room", then U becomes

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} = \frac{1}{4} + \frac{2}{8} + \frac{\textcircled{2}}{10}$$

(② reduced from 4)

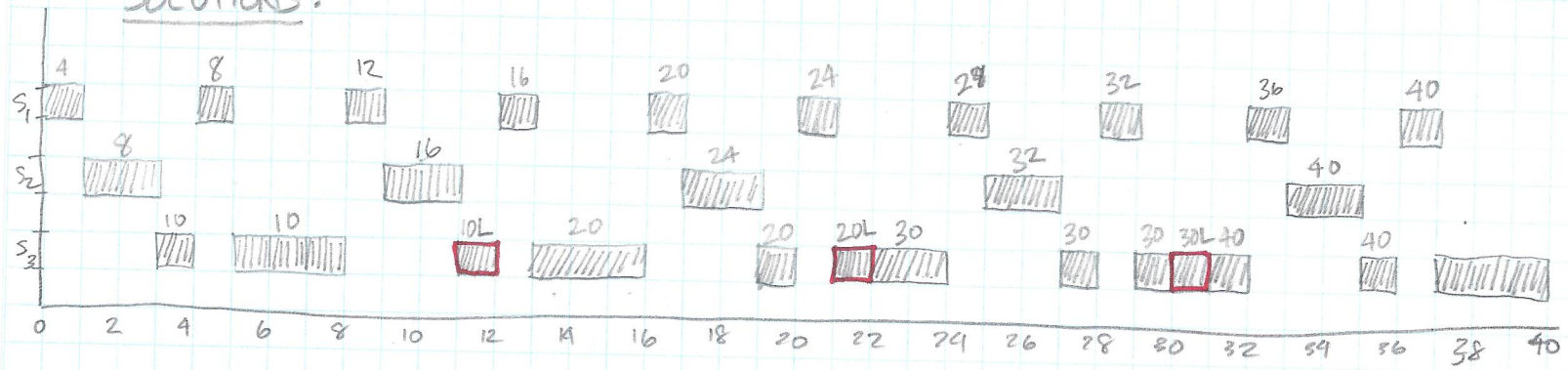
$$= \frac{1}{2} + \frac{2}{10} = \frac{5+2}{10} = \frac{7}{10} = 0.70.$$

in this case, $U = 0.70 \leq 0.78$.

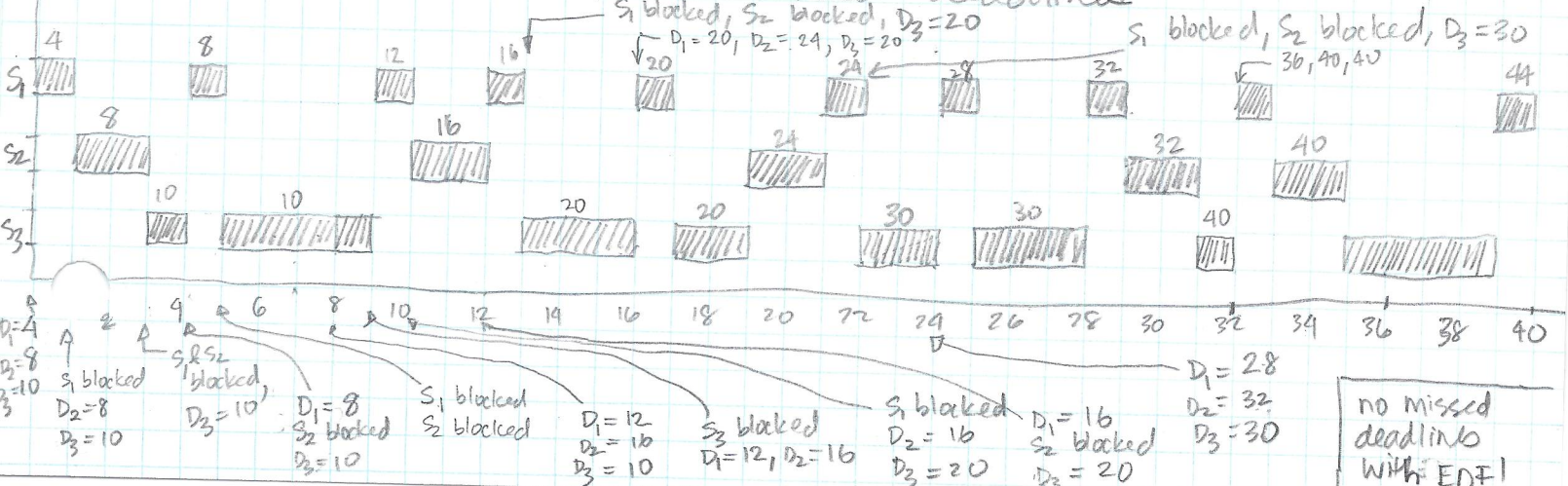
EXERCISE (HOMEWORK/LAB):

- in the previous example, set $C_3 = 5$. Is RM policy still optimal? How many missed deadlines are there over LCM time? Which task is most affected?
- apply EDF dynamic priority policy to this problem; are there missed deadlines?

SOLUTIONS:



∴ there are three missed deadlines



Free RTOS Overview

- Free RTOS application form:

```
void Task1 (...) {
  while 1 {

  }
  return 1;
}
```

```
void Task 2 (...) {
  while 1 {

  }
  return 1;
}
```

Tasks are created as infinite loops

```
int main ( ) {
  prv.Setup Hardware ( );
  xTaskCreate (Task1, ...);
  xTaskCreate (Task2, ...);
  :
  vStartScheduler ( );
  return 1;
}
```

main code in every Free RTOS app looks something like this

← hardware initialization

tasks are "setup" (not executed) here, including allocation of stack space on the heap and the setting of priorities

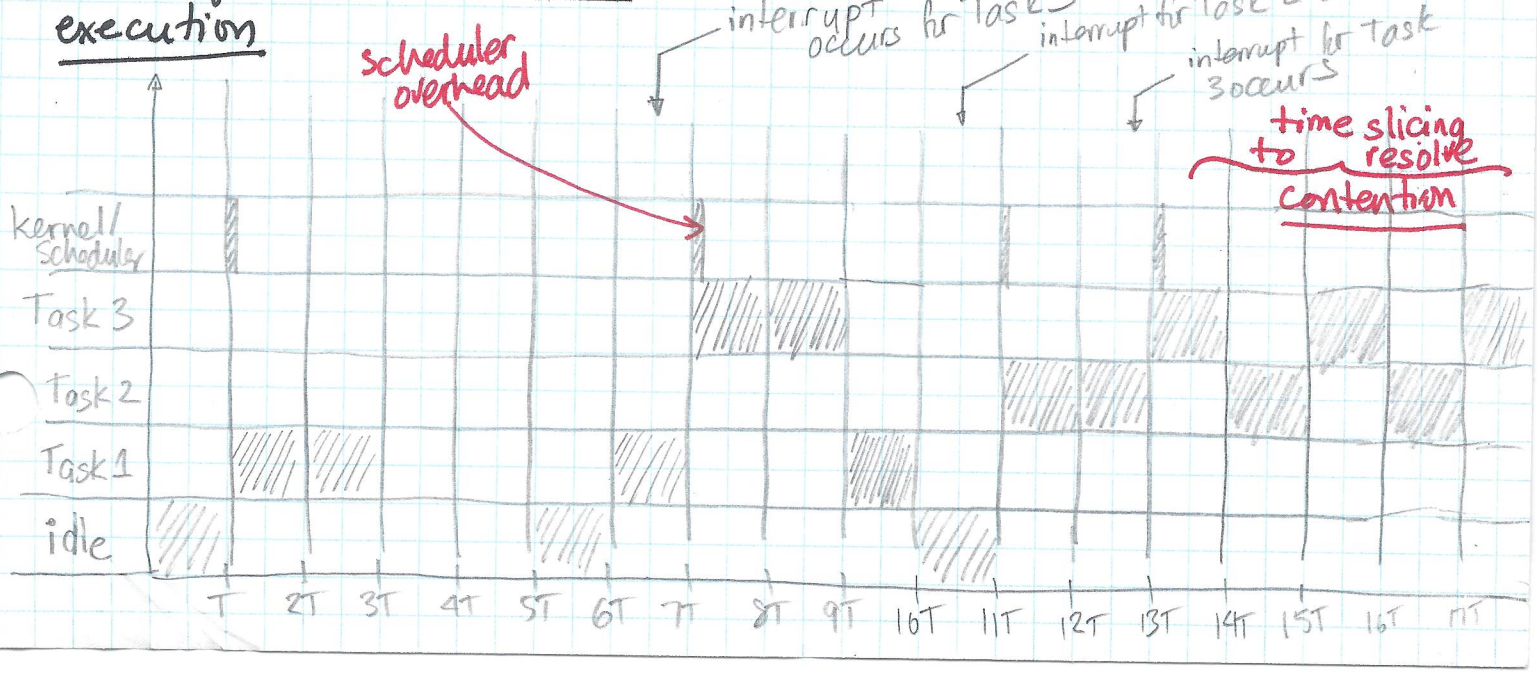
this is where the program hands control over to the Scheduler, which never relinquishes control

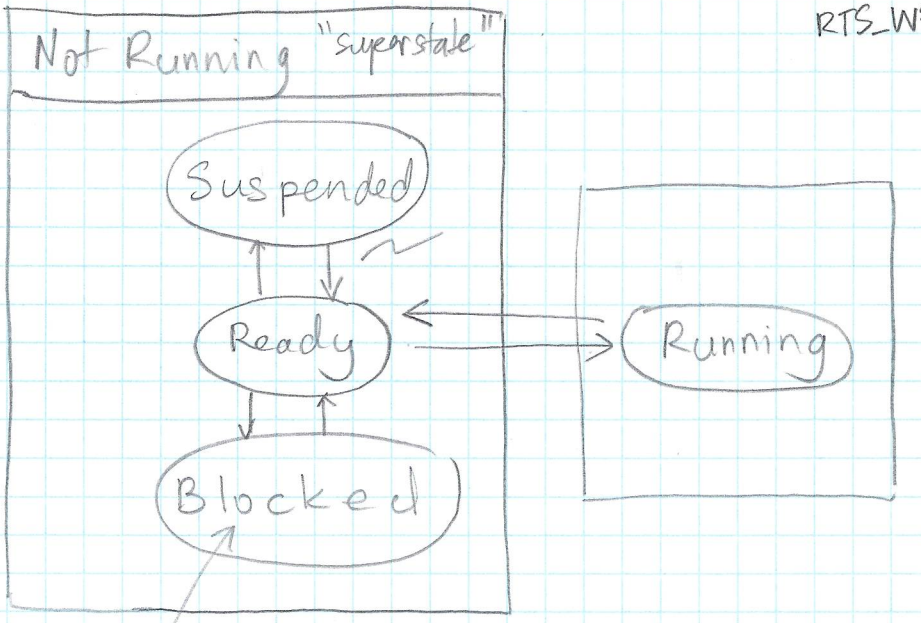
this statement is never reached

priorities

task	priority
⋮	⋮
Task 3	4
Task 2	4
Task 1	1
idle	0

- thus far we have only discussed setup → what happens before we hand control over to the scheduler
- now we can discuss execution → we assume: preemptive mode of operation (FreeRTOS can do cooperative as well), a single core, and a fixed set of priorities
- in addition we assume a tick period which is many times (1000 to 1000 000 times the CPU clock period) — FreeRTOS has a default of 10 ms.





blocking occurs when the task is waiting for certain kinds of events, such as the expiration of a delay

→ why we use vTaskDelay(), a FreeRTOS function instead of another C-language delay implementation (which would keep the task in the running state).

- vTaskDelayUntil(·):