CT420 REAL-TIME SYSTEMS

SCHEDULING ALGORITHMS FOR RTS

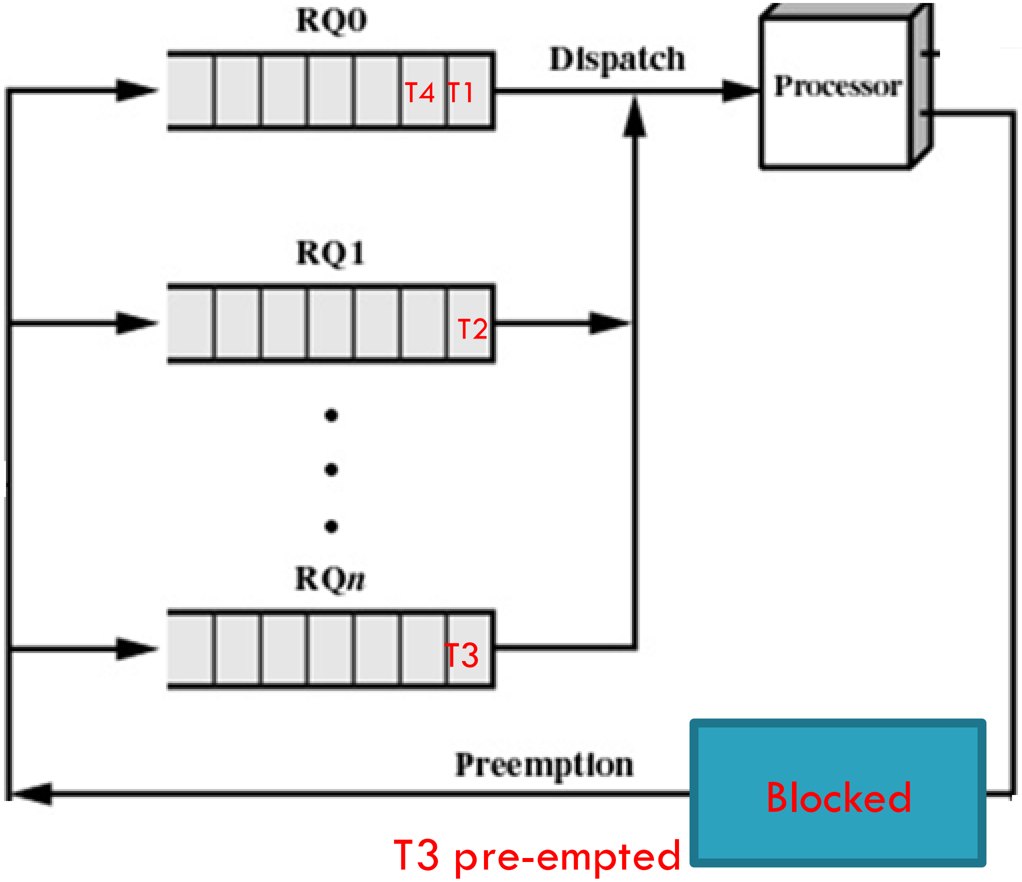
Dr. Michael Schukat

Ollscoil na Gaillimhe
University of Galway

Motivation

- Assume you work as an engineer in the automotive industry
- You are the firmware lead for an engine control unit project (a RTSCS) for a fuel-efficient Diesel engine
- Previous designs you worked on were based on a CE, i.e. based on a manually constructed schedule with well-defined tasks with known WCETs
  - This design worked very well, meeting consistently task time constraints (as exercised in the examples before)
- Now your project manager asks you to go with a modern design, i.e. use the VxWorks RTOS (or OSEK) for the product
  - How can the feasibility of a task schedule be proven?

Recap POSIX FIFO Process Scheduling

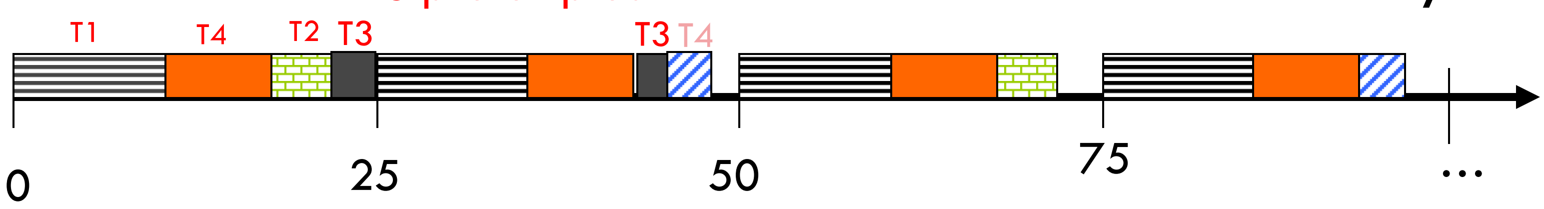


Process Tx:

```
int main() {
// Initialise process
// Setup timer x to notify Tx
// about begin of every cycle, e.g.
// T1: 25ms; T2 = 50ms; T3 = 100ms
while (1) {
  do_something();
  block_until_timer_signal();
}
}
```

Question:

Considering only one task per priority (i.e. T4 and T1 are merged into one task in the example), when is a schedule actually feasible?



Feasibility Analysis of Task / Process Schedule

- Cyclic executive
  1. Determine minor /major cycle
  2. Determine WCET of all tasks
  3. Align tasks in CE schedule
     - Leave some slack time for ISR handling if needed
  4. Done
- RTOS
  1. Determine execution frequency for each process
  2. Determine WCET of each process
  3. Factor in additional RTOS (i.e. kernel/scheduler) and signal overheads
  4. Assign each process a different priority and link each process to its timer as seen before
  5. Validate that process schedule works, i.e. that all processes can be executed according to their schedule and deadlines?
     - The problem is that in contrast to a cyclic executive process-pre-emption needs to be factored in and a low priority task can be pre-empted by a higher priority task

Overview

- We are looking at analytical methods to determine if a schedule managed by an RTOS is feasible
- Firstly, we'll consider **rate-monotonic scheduling (RMS)**
  - a mathematical model for an optimal static priority scheduling algorithm
  - closely linked to priority-driven pre-emptive scheduling (see pathfinder case study)
- However, RMS is not that straight forward when it comes to guarantee the feasibility of a task schedule
  - Therefore, we also consider a second scheduling algorithm which is much more straight forward when it comes to guarantee / prove a schedule's feasibility
  - Here we consider **earliest deadline first (EDF)**, which is an optimal dynamic priority scheduling algorithm

Scheduling for RTS

- A schedule is <u>feasible</u> if
  - all the tasks/processes start after their release time and
  - complete before their deadlines
- Scheduling Policy may be determined
  - Pre-run-time
    - Schedule created offline
    - See cyclic executive approach
  - Run-time
    - Schedule determined online as tasks arrive
    - Process scheduler determines what process get CPU time

Scheduling for RTS

- Run-time Static versus Run-time Dynamic Priority
  - Static Priority Scheduling Algorithm
    - Task priority does not change
      - Rate Monotonic Algorithm (RM)
  - Dynamic Priority Scheduling Algorithm
    - Process priorities can change over time
      - Earliest Deadline First (EDF)
- Pre-emptive versus non-pre-emptive scheduling
  - Pre-emptive Schedule
    - Task can be pre-empted by other tasks
    - Penalty of context switches
  - Non pre-emptive
    - Task runs to completion unless blocked over resource

Simplifications for our Considerations

- All tasks are periodic
  - Fair enough, but we also have to deal with asynchronous tasks (e.g., ISR)
- Just one task per priority level
  - No big deal either
- No precedence constraints
  - Here, tasks may be merged to implicitly solve precedence constraints
- No task has any non-preemptible sections
  - A good RTOS kernel should accommodate this (e.g. all kernel calls are pre-emptible)
  - Task synchronisation (i.e. semaphores) should be avoided
- Cost of pre-emption is zero
  - Instead, add task pre-emption time overheads (typically known) to task WCET
- Non-CPU resources, e.g. Memory or I/O, are infinite
  - Consider memory locking or better no page swapping at all

Rate Monotonic Scheduling

- Run time, static priority and pre-emptive
- Priority inversely related to period (can be considered as a restriction)
  - Eg. given task T_i and T_j where $p_i < p_j$
    - Priority of task T_i greater than T_j
- In real world, the more critical RTS parameters tend to require faster sample rate/response times of processes controlling those parameters
  - RM is a good match in this regard
- Scheduling decision is to be made when
  - The current task execution is complete
  - A new task is released
- Task T_i utilisation $u_i = e_i / p_i$
  - Overall CPU utilisation $U = \sum_{i=1}^{n} u_i$

RM Example

Task	e	p	u
T_1	1	4	0.25
T_2	2	5	0.4
T_3	5	20	0.25

All Tasks released at time 0; Priority $T_1 < T_2 < T_3$; Overall U = 0.9

Sequence

1st instance Task 1 runs to completion

1st instance Task 2 runs to completion

1st instance Task 3 runs for 1 unit

..at EU=4, Task 1 released → pre-empts Task 3

2nd instance Task 1 runs to completion

..at EU =5, Task 2 released

2nd instance Task 2 runs to completion

1st instance Task 3 runs for 1 unit

.. At EU = 8, Task 1 released → pre-empts Task3

3rd instance Task 1 runs to completion

1st instance Task 3 runs for 1 unit

.. At EU = 10, 3rd instance of Task 2 released → pre-empts 3

..

At EU = 15, 1st instance Task 3 completes.. CPU idle EU 18-20

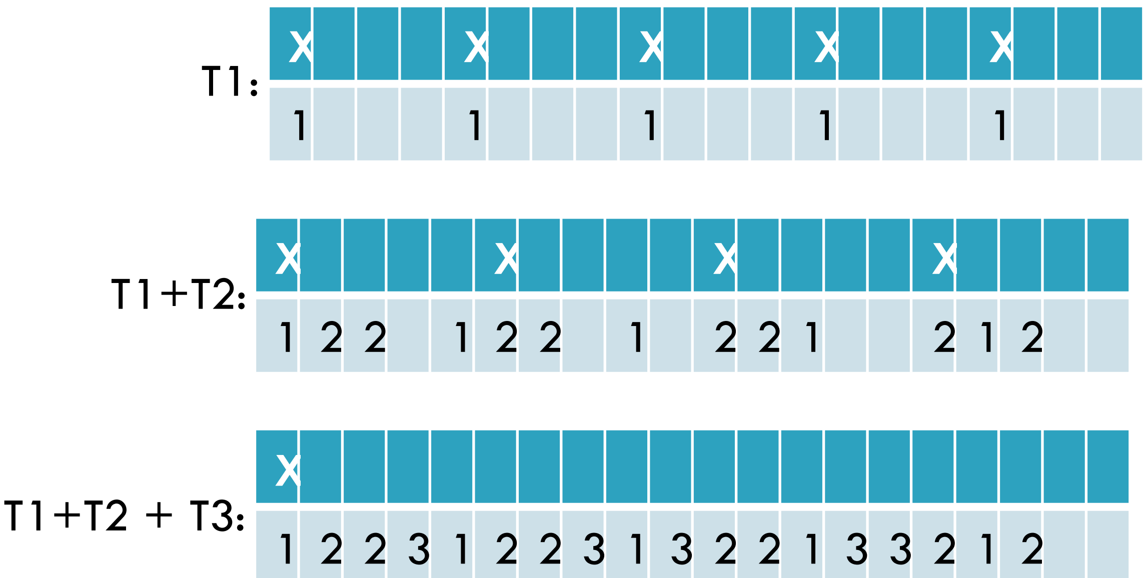
At EU = 20, all 3 tasks released .. Cycle repeats

1 2 2 3 1 2 2 3 1 3 2 2 1 3 3 2 1 2 1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Execution Units EU

RM Example



Task	e	p	u
T_1	1	4	0.25
T_2	2	5	0.4
T_3	5	20	0.25

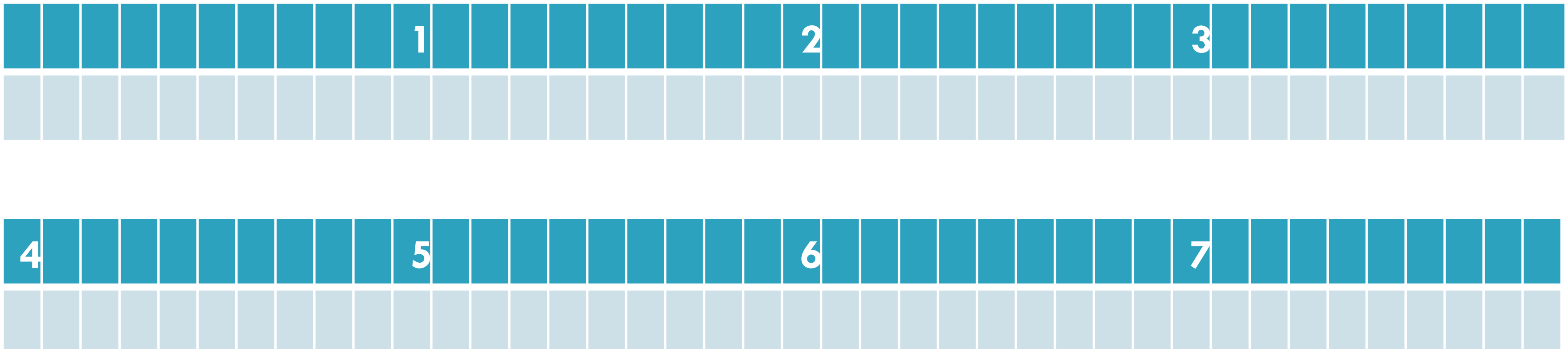
U = 0.9

RM Schedulability?

- Consider Task set
- U = 1/5 + 1/6 + 1/3+ 1/4 = 57/60
- Does this schedule work too?

i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

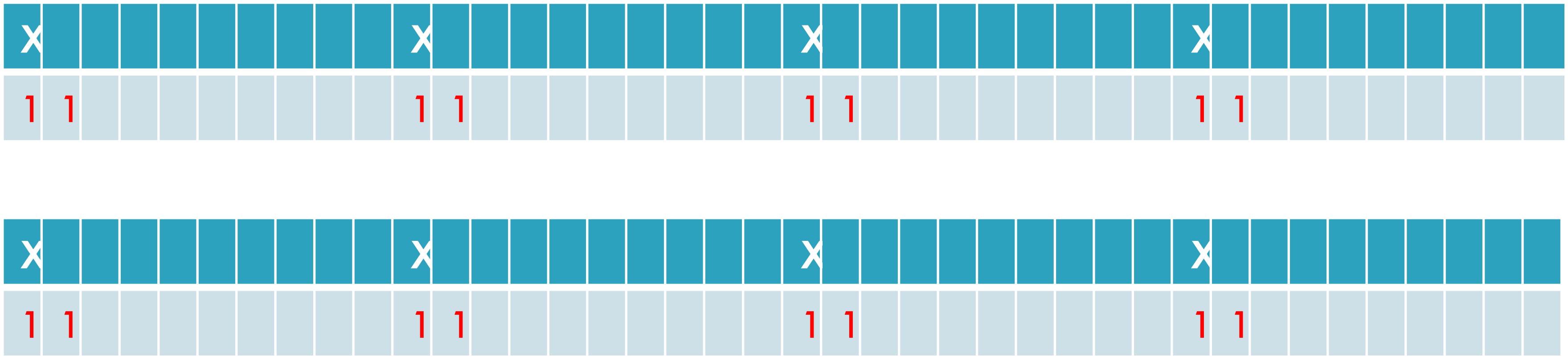
RM Schedulability?



i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

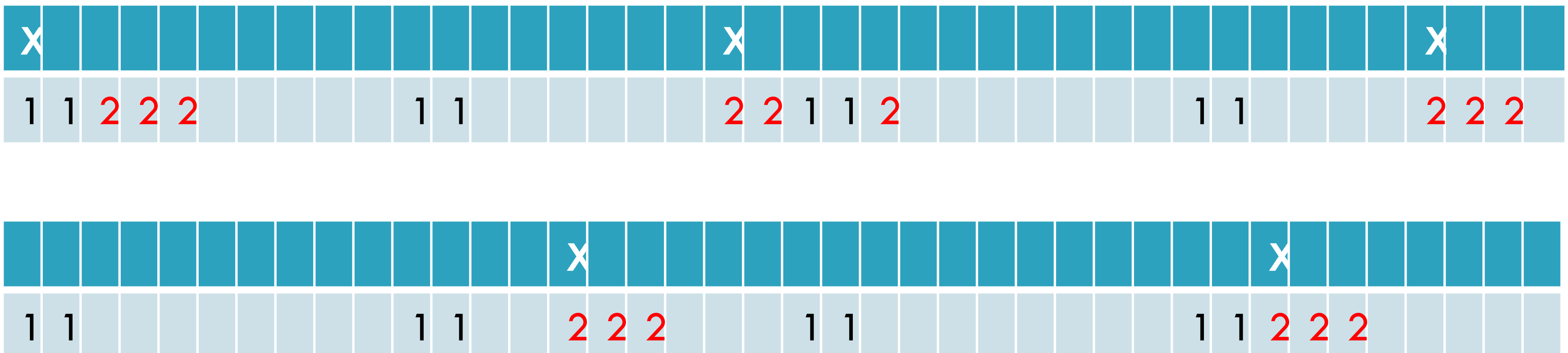
Please use the worksheet on Blackboard to complete this exercise

RM Schedulability?



i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

RM Schedulability?



i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

RM Schedulability?

x																								x															
1	1	2	2	2	3	3	3	3	3	1	1	3	3	3				2	2	1	1	2		3	3	3	3	3	3	1	1	3	3			2	2	2	

								x																								x							
1	1							3	3	1	1	3	3	2	2	2	3	3	3	1	1	3								1	1	2	2	2	3	3	3	3	3

i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

RM Schedulability?

x																																							
1	1	2	2	2	3	3	3	3	3	1	1	3	3	3	4	4	4	2	2	1	1	2	4	3	3	3	3	3	3	1	1	3	3	4	4	2	2	2	4

1	1							3	3	1	1	3	3	2	2	2	3	3	3	1	1	3								1	1	2	2	2	3	3	3	3	3

i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

RM Schedulability?

x																																							
1	1	2	2	2	3	3	3	3	3	1	1	3	3	3	4	4	4	2	2	1	1	2	4	3	3	3	3	3	3	1	1	3	3	4	4	2	2	2	4

x																																							
1	1	4	4	4	4	4	4	3	3	1	1	3	3	2	2	2	3	3	3	1	1	3	4	4	4	4	4	4	4	1	1	2	2	2	3	3	3	3	3

i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

RM Scheduling

- General schedulability test
  - If **U <= n($2^{1/n}$ -1)**
      - where n = number of tasks
    - RM will definitely produce feasible schedule
    - No need for further analysis
  - However
    - RM may produce feasible schedule when
      - U > n($2^{1/n}$ - 1)
    - i.e. Sufficient but **not** necessary condition
    - Recall Example: CPU U = 0.9 but still schedulable
      - Depends on particular task characteristics
    - If U > n($2^{1/n}$ - 1)
      - need to perform further schedulability analysis
  - As n increases, bound ➔ 69%

RM Schedulability Analysis

- Consider taskset T_1 T_2 T_3 T_4 with
  - $p_1 < p_2 < p_3 < p_4$
- Task 1
  - Highest priority.. never pre-empted
  - Will run immediately once released
  - For Task 1 to be feasibly scheduled
    - Only condition is that $e_1 <= p_1$
- Include Task 2 in task set
  - Can only be pre-empted by Task 1
  - Will be executed iff one can find sufficient time e_2 over period $[0, p_2[$
  - Say Task 2 completes at time t within $[0, p_2[$
  - How many times did Task 1 run over [0,t] ?

RM Schedulability Analysis

- Over interval [0,t], Task 1 is released $\left\lceil \frac{t}{p_1} \right\rceil$
- Time t to complete task 2 must satisfy condition
  - $t = e_2 + e_1 \left\lceil \frac{t}{p_1} \right\rceil$
- Need to find t over interval $[\,0, p_2[$
- Find integer k such that:
  - $k\,p_1 >= k\,e_1 + e_2$
  - $k\,p_1 <= p_2$

Rounded up, e.g.
$\lceil 10 / 3 \rceil = 4$

RM Schedulability Analysis

- Include Task 3
  - Can be pre-empted by Task 1 and 2
  - Need to find t over [0, p_3[such that

$$t = \left\lceil t/p_1 \right\rceil e_1 + \left\lceil t/p_2 \right\rceil e_2 + e_3$$

  - Need to check only at multiples of p_1 and / or p_2
- Similar analysis for Task 4
  - Can be pre-empted by Task 1,2,3

RM Schedulability Analysis

- General Rule
- $W_i(t) =$ $$\sum_{j=1}^{i} e_j \left\lceil {t}/{p_j} \right\rceil$$

  = total work carried out by tasks $T_1 T_2 T_3 \ldots T_i$ initiated in interval [0,t]
- If $W_i(t) \le t$, then schedule is feasible
  - → $(W_i(t) / t) \le 1$
- $W_i(t)$ only changes at finite number of points when tasks are released
  - Check points defined by $$\tau_i = \left\{ l p_j \,\middle|\, j = 1, .., i; l = 1, .., \left\lfloor {p_i}/{p_j} \right\rfloor \right\}$$

RM Schedulability Analysis

- Consider Task set
- General schedulability test:
  - $n = 4 \rightarrow n(2^{1/n} - 1) = 0.76$
  - Note U = 0.95 (0.2+0.166+0.33+0.25)
  - ➔ further analysis required

i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

Example: RM Schedulability Analysis

- Check points
  - t_1: {100}
  - t_2: {100,180}
  - t_3: {100,180,200,240}
  - t_4: {100,180,200,240,300,360,400}

i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

Example: RM Schedulability Analysis

- $W_i(t) = \sum_{j=1}^{i} e_j \left\lceil t/p_j \right\rceil$
- W_1:
  - Interval [0,100[
    - $W_1(t) = e_1 = 20$
- W_2 : checkpoints {100,180}
  - Interval [0,100[; $W_2(t) =$ $e_1 \left\lceil 100/p_1 \right\rceil + e_2 \left\lceil 100/p_2 \right\rceil$

    =20(1) +30(1) = 50
  - Interval [0,180[; $W_2(t)=$ $e_1 \left\lceil 180/p_1 \right\rceil + e_2 \left\lceil 180/p_2 \right\rceil$

    20(2) +30(1) = 70

Example: RM Schedulability Analysis

- W_3 : checkpoints {100,180,200,240}
  - $W_3(t) = e_1\left\lceil t/p_1 \right\rceil + e_2\left\lceil t/p_2 \right\rceil + e_3\left\lceil t/p_3 \right\rceil$
  - Interval [0,100[
    - $W_3(t) = 20(1) + 30(1) + 80(1) = 130$
  - Interval [0,180[
    - $W_3(t) = 20(2) + 30(1) + 80(1) = 150$
  - Interval [0,200[
    - $W_3(t) = 20(2) + 30(2) + 80(1) = 180$
  - Interval [0,240[
    - $W_3(t) = 20(3) + 30(2) + 80(1) = 200$

Example: RM Schedulability Analysis

- Task 1 is RM Schedulable iff
  - $e_1 <= 100$ (True)
- Task 1/2 is RM Schedulable iff
  - $e_1 + e_2 <= 100$ or ....(True: 50)
  - $2 e_1 + e_2 <= 180$ .... (True: 70)
- Task 1/2/3 is RM Schedulable iff
  - $e_1 + e_2 + e_3 <= 100$ or .... (False: 130)
  - $2 e_1 + e_2 + e_3 <= 180$ or ..... (True: 150)
  - $2 e_1 + 2e_2 + e_3 <= 200$ or ...(True: 180)
  - $3 e_1 + 2e_2 + e_3 <= 240$ .... (True: 200)

Example: RM Schedulability Analysis

- Task 1/2/3/4 is RM Schedulable iff
  - $e_1 + e_2 + e_3 + e_4 <= 100$ or …. (False: 230)
  - $2e_1 + e_2 + e_3 + e_4 <= 180$ or ….. (False: 250)
  - $2e_1 + 2e_2 + e_3 + e_4 <= 200$ or …(False: 280)
  - $3e_1 + 2e_2 + e_3 + e_4 <= 240$ or …. (False: 300)
  - $3e_1 + 2e_2 + 2e_3 + e_4 <= 300$ or …. (False: 380)
  - $4e_1 + 2e_2 + 2e_3 + e_4 <= 360$ or …. (False: 400)
  - $4e_1 + 3e_2 + 2e_3 + e_4 <= 400$ …. (False: 430)
- By including Task 4, not RM schedulable
- Can also plot results
  - Check whether $W_i(t)$ falls on or below $W_i(t) = t$ line

RM Schedulability?

X																																							
1	1	2	2	2	3	3	3	3	3	1	1	3	3	3	4	4	4	2	2	1	1	2	4	3	3	3	3	3	3	1	1	3	3	4	4	2	2	2	4

4										5										6										7									
1	1							3	3	1	1	3	3	2	2	2	3	3	3	1	1	3								1	1	2	2	2	3	3	3	3	3

i	e_i	p_i
1	20	100
2	30	180
3	80	240
4	100	400

RM Schedulability Analysis

- Sporadic Tasks
  - So far have only considered periodic tasks
    - ➔ unrealistic
  - Can view sporadic task as infrequent periodic task if can specify
    - Minimum interarrival time between release of successive sporadic tasks
    - Maximum execution time
    - ➔ Simply treated as additional task in RM analysis

Earliest Deadline First (EDF)

- Run-time, dynamic and preemptable
- Ready task whose absolute deadline is the earliest is given highest priority
- Task priorities are re-evaluated when tasks released / completed
- EDF is an optimal single-processor scheduling algorithm
  - If all tasks are periodic
    - Task 1…n ; CPU U = $\sum_{i=1}^{n} u_i$
    - **If U <=1, then task set is EDF schedulable!**

EDF Example

Task	e	p	u
T_1	1	4	0.25
T_2	2	5	0.4
T_3	5	20	0.25

All Tasks released at time 0; Overall U = 0.9

Sequence

1st instance Task 1 runs 1st as earliest deadline of 4

1st instance Task 2 runs to completion

1st instance Task 3 runs for 1 unit ..note: Deadline is 20

..at EU=4, Task 1 rel. → pre-empt Task 3 as deadline is 8

2nd instance Task 1 runs to completion

..at EU =5, Task 2 released

2nd instance Task 2 runs to completion as deadline is 10

1st instance Task 3 runs for 1 unit

.. At EU = 8, Task 1 released → pre-empts Task3

3rd instance Task 1 runs to completion

1st instance Task 3 runs for 1 unit

.. At EU =10, Task 2 released.. pre-empts task 3 as deadline is 15

At EU =12, Task 1 runs as deadline 16 < 20

At EU =15 Task 2 released and runs with deadline 20

At EU =16 Task 1 released with deadline 20 → no pre-emption

1 2 2 3 1 2 2 3 1 3 2 2 1 3 3 2 2 1 1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Execution Units EU

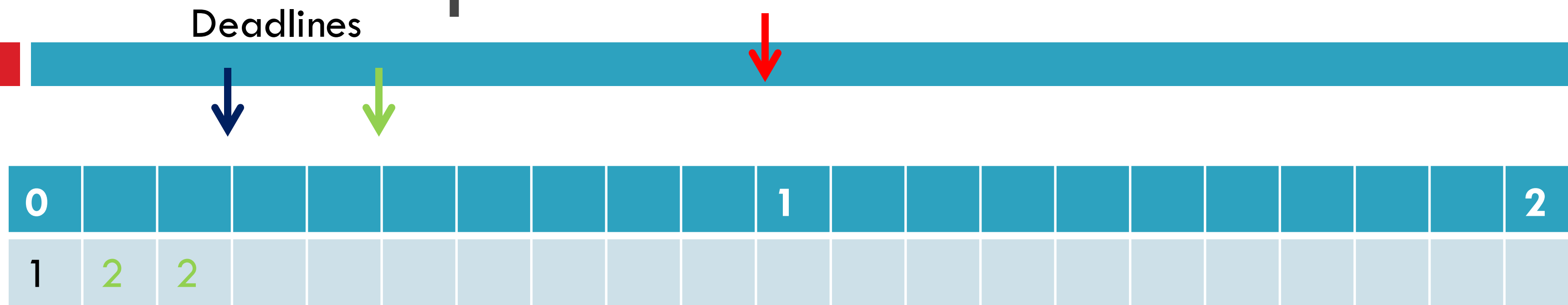
EDF Example

0										1										2

Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

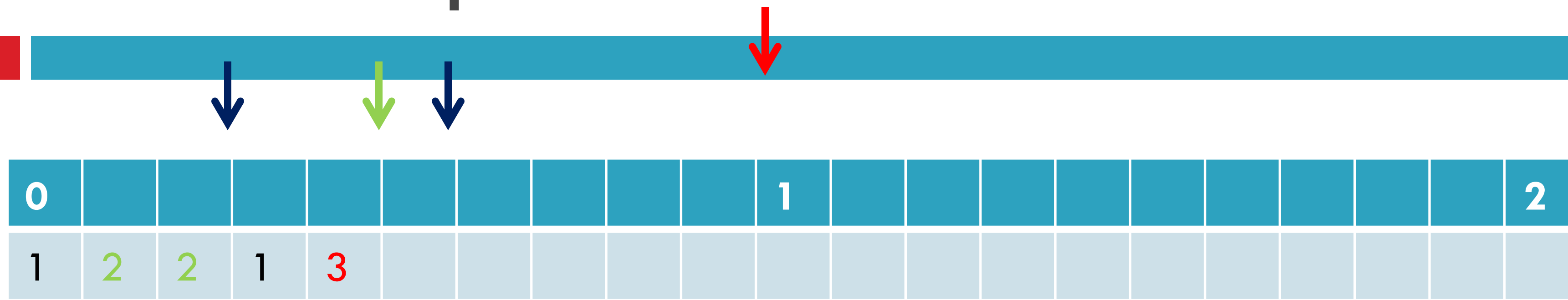
Please use the worksheet on Blackboard to complete this exercise

EDF Example



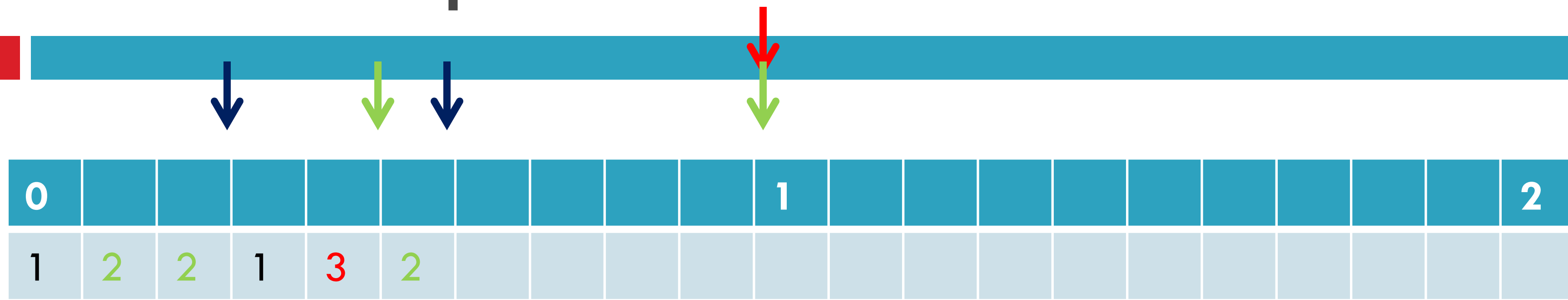
Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

EDF Example



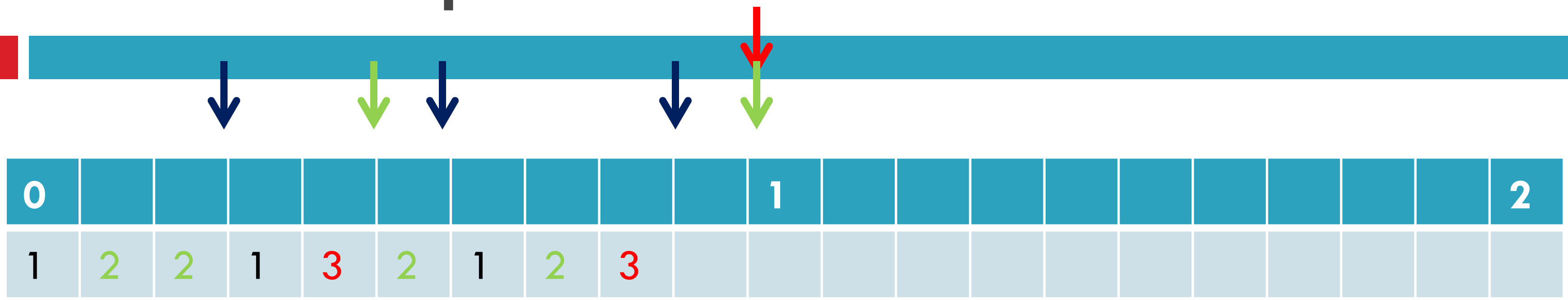
Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

EDF Example



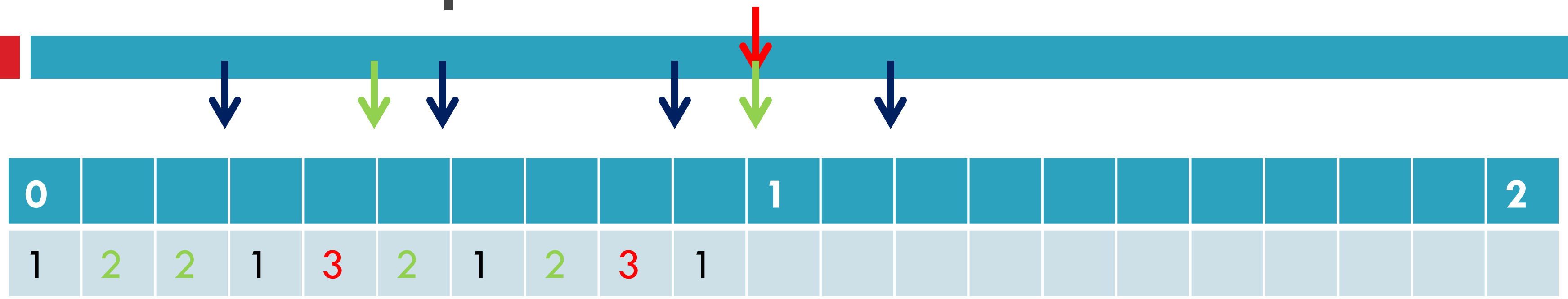
Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

EDF Example



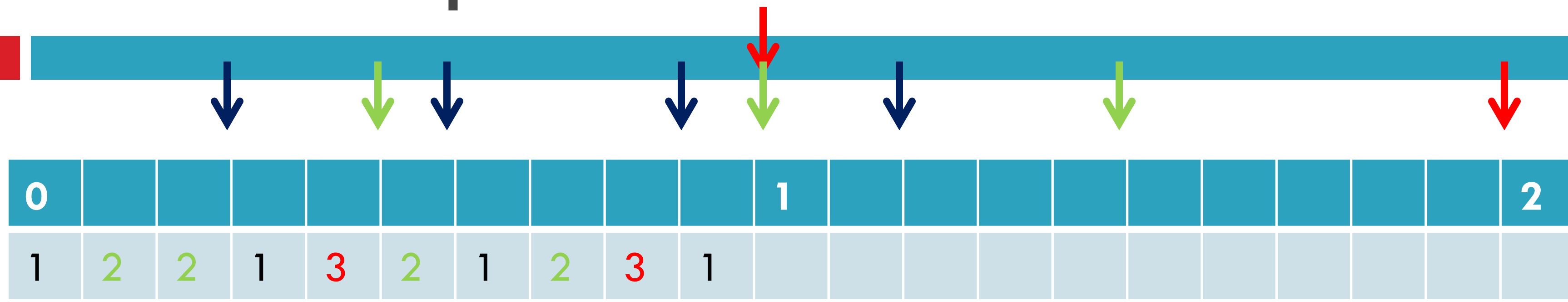
Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

EDF Example



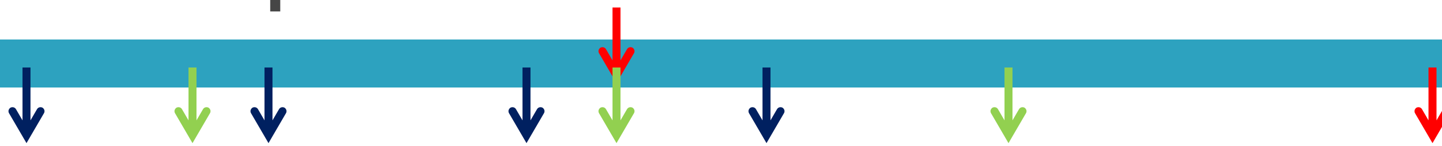
Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

EDF Example



Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

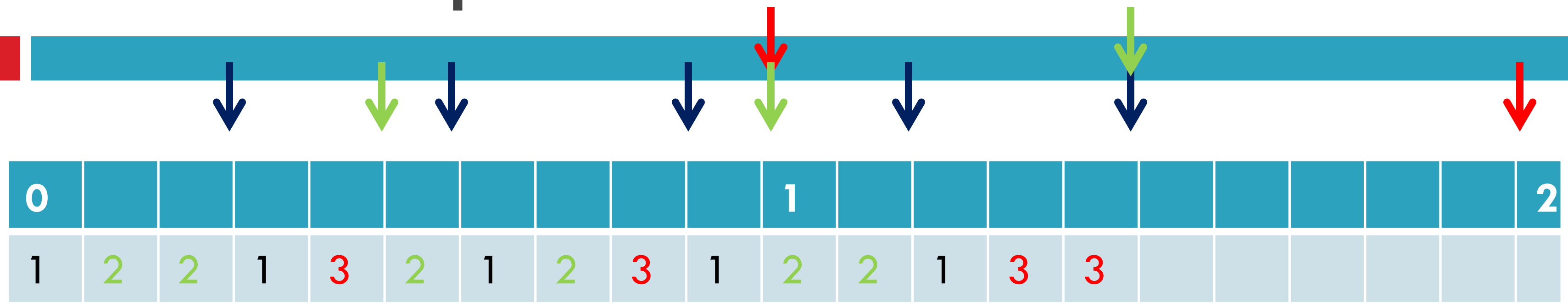
EDF Example



0										1										2
1	2	2	1	3	2	1	2	3	1	2	2									

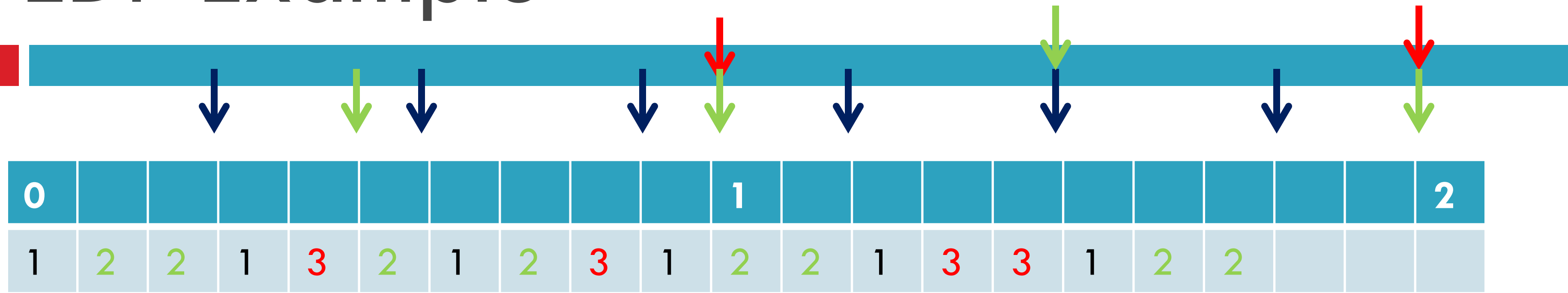
Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

EDF Example



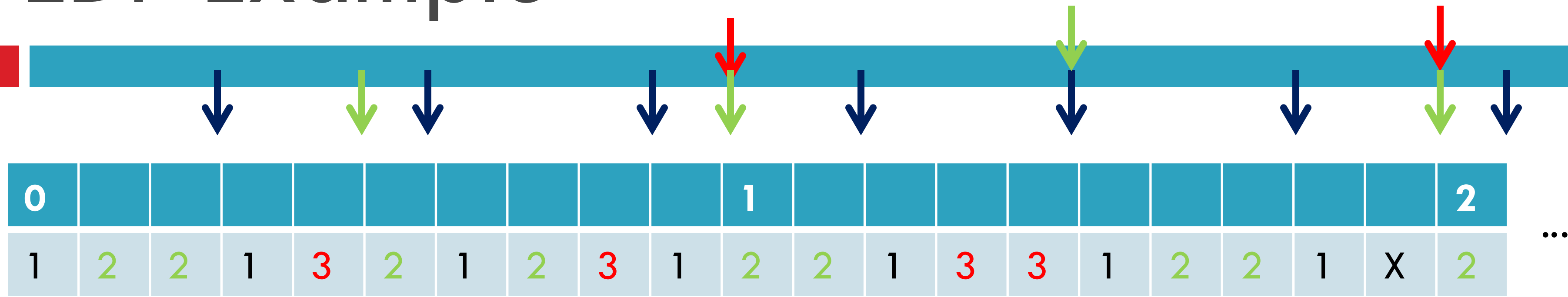
Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

EDF Example



Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

EDF Example



Task	e	p	u
T_1	1	3	0.33
T_2	2	5	0.4
T_3	2	10	0.2

EDF vs RM

- With RM, priorities fixed
  - High priority tasks guaranteed CPU time
    - Good mapping to priority-driven pre-emptive scheduling
  - In overload conditions, lower priority tasks lose out
  - Bound on CPU utilisation must be considered
    - Necessary but not sufficient
- EDF, dynamic priority
  - More flexible, but less predictable
  - In overload conditions, all tasks may miss deadlines
  - Schedulable if CPU U <=1