EEE499 - Real-Time Embedded System Design

Real-Time Schedulability Part I

Scheduling

- Scheduling algorithms - **Policy**
  - the **rules** under which tasks are assigned to processors are defined by the system scheduling algorithm
- The **scheduler** - **Mechanism**
  - the module which implements these algorithms and protocols

Preemptive Fixed Priority Scheduling

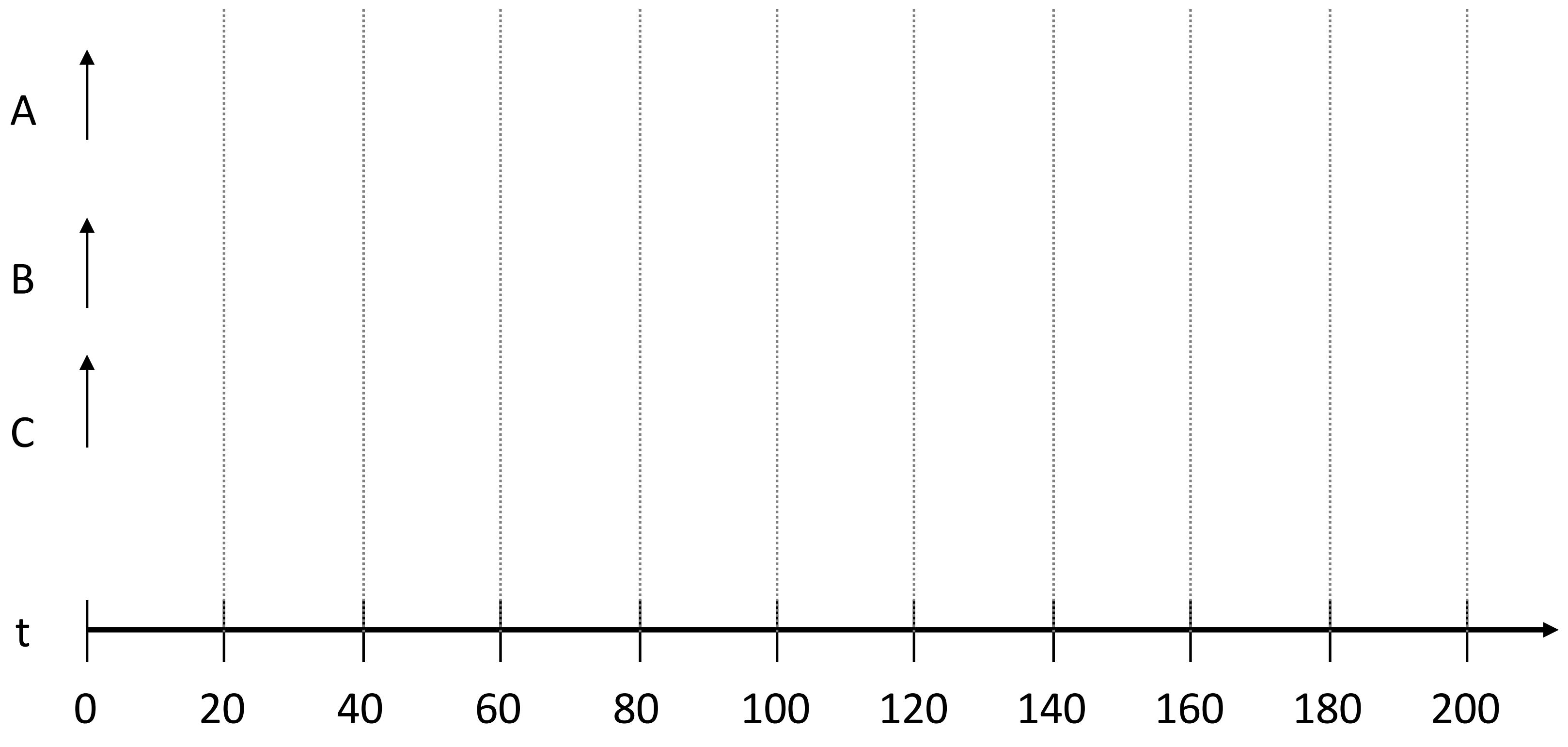
- Priority can be determined by:
  - **Period:** shortest period has highest priority
  - **Deadline:** shortest deadline has highest priority

Preemptive Fixed Priority Scheduling

- Priority can be determined by:
  - **Period:** shortest period has highest priority
  - **Deadline:** shortest deadline has highest priority
- **Rate Monotonic** priority assignation is based on the **period**
- **Deadline Monotonic** priority assignation is based on the **deadline**
- When period = deadline: RM = DM

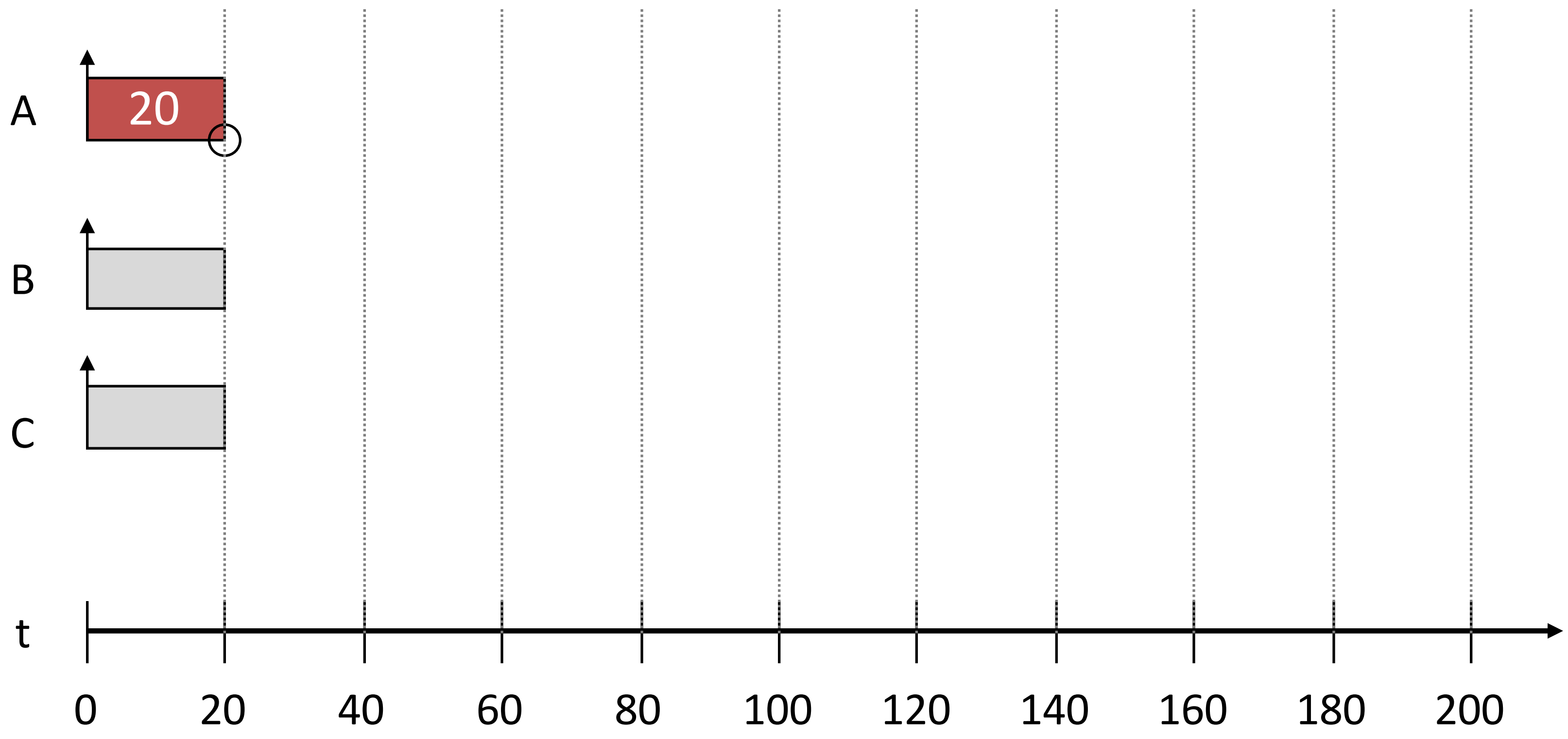
Preemptive Fixed Priority Scheduling

$$\begin{tikzcd} \boxed{\textbf{Ready}} \arrow[r] & \boxed{T_B\ (160,\ 30)} \arrow[r] & \boxed{T_A\ (100,\ 20)} \arrow[r] & \boxed{T_C\ (200,\ 60)} \end{tikzcd}$$

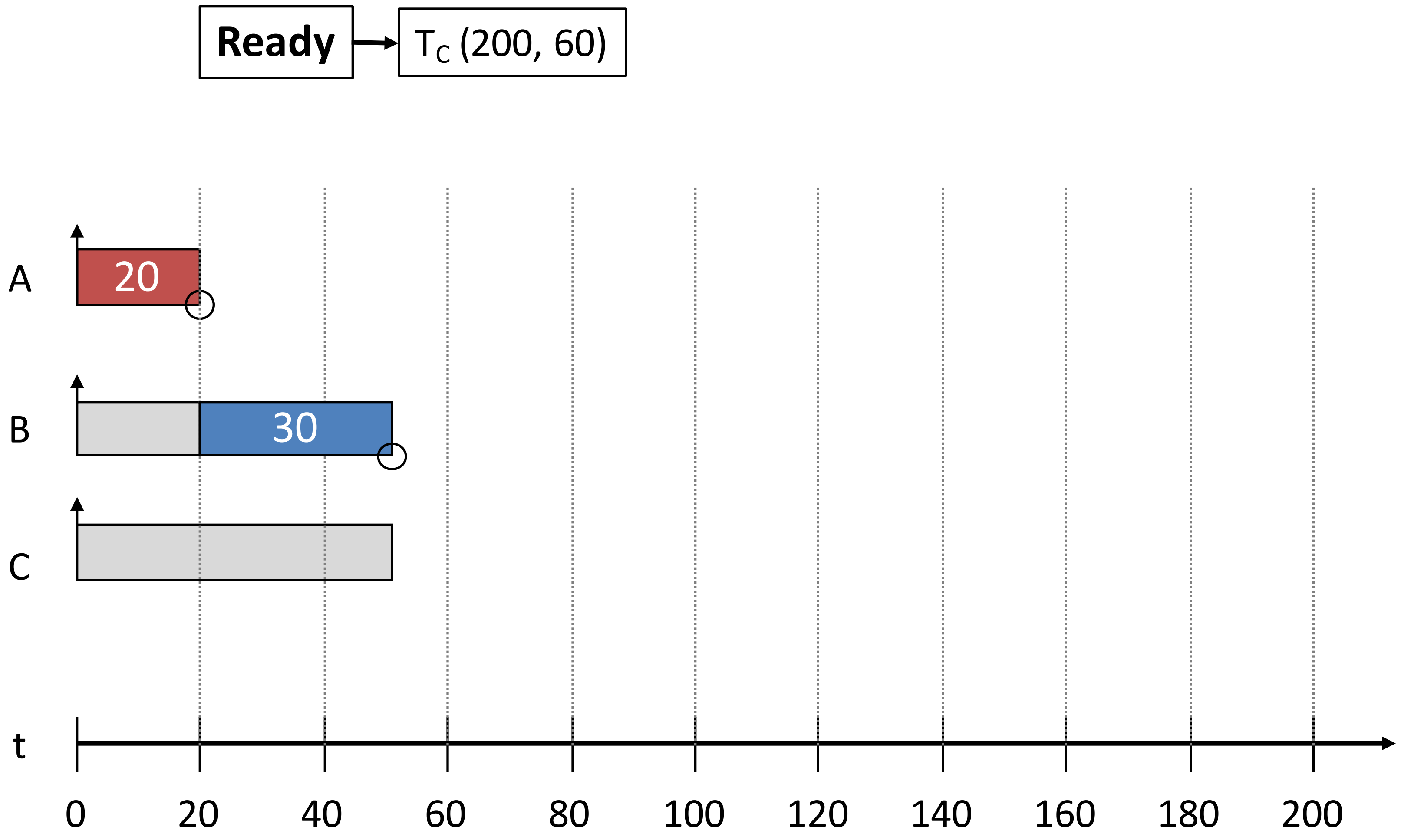


Preemptive Fixed Priority Scheduling

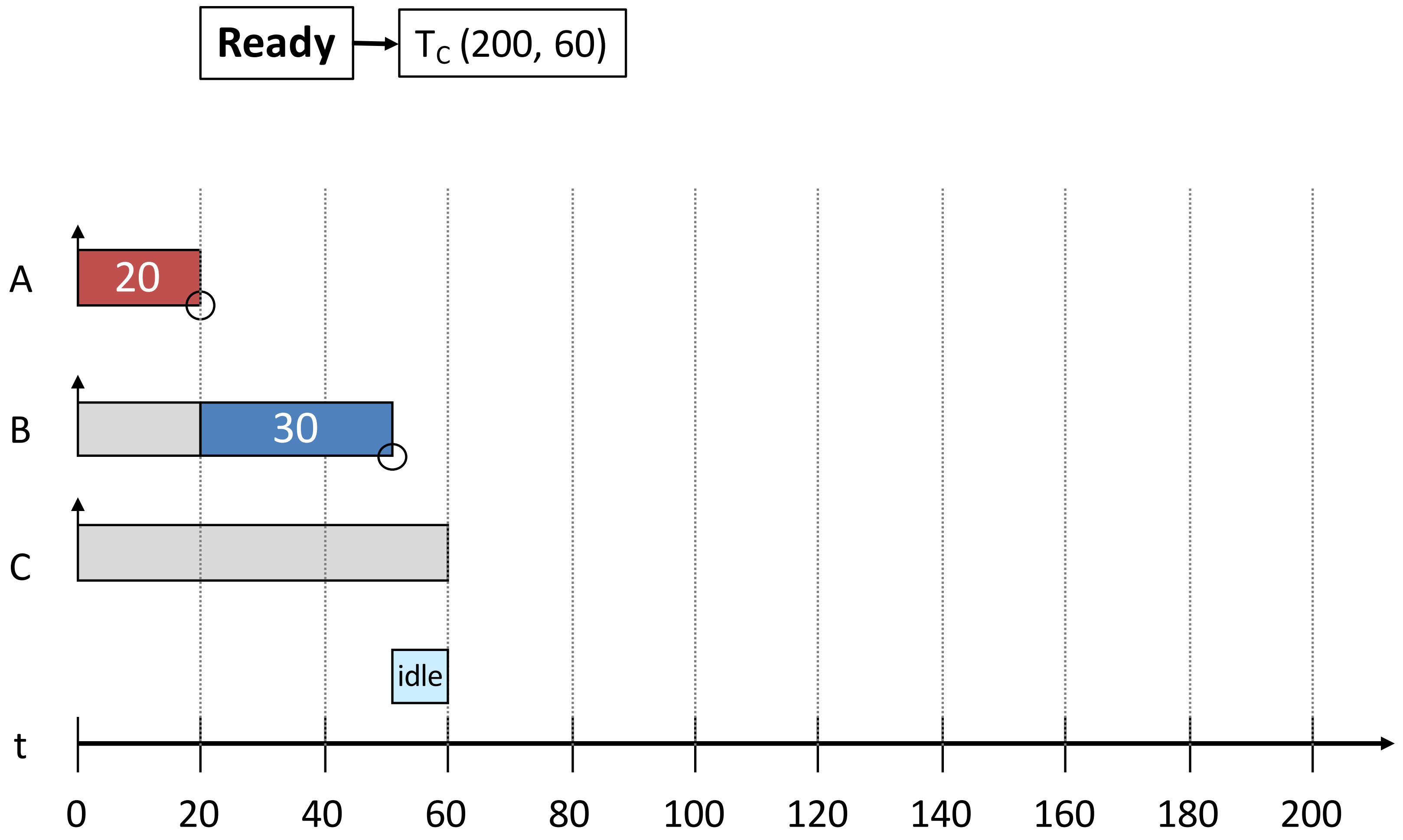
$$\begin{tikzcd} \textbf{Ready} \arrow[r] & T_B\ (160,\ 30) \arrow[r] & T_C\ (200,\ 60) \end{tikzcd}$$



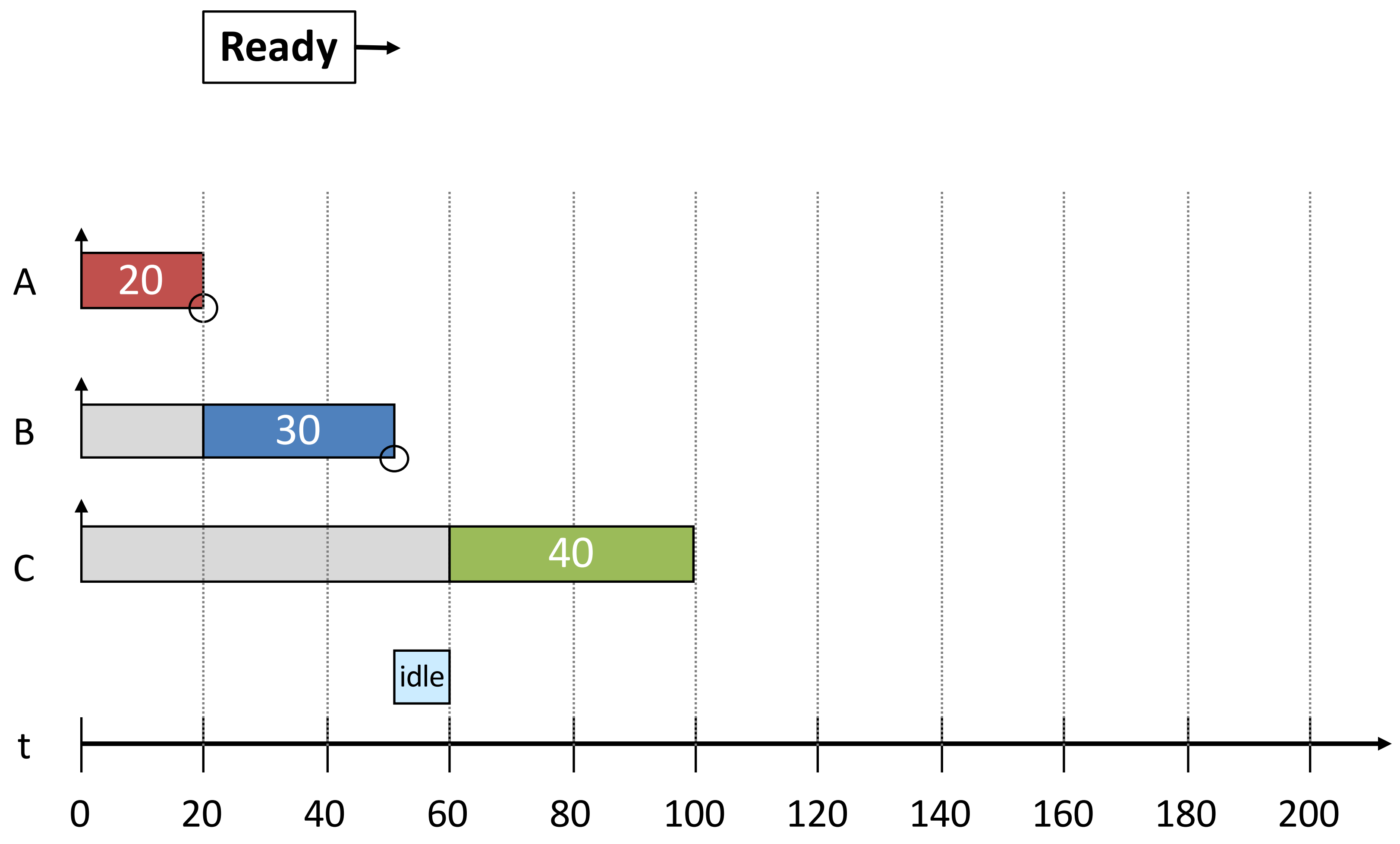
Preemptive Fixed Priority Scheduling



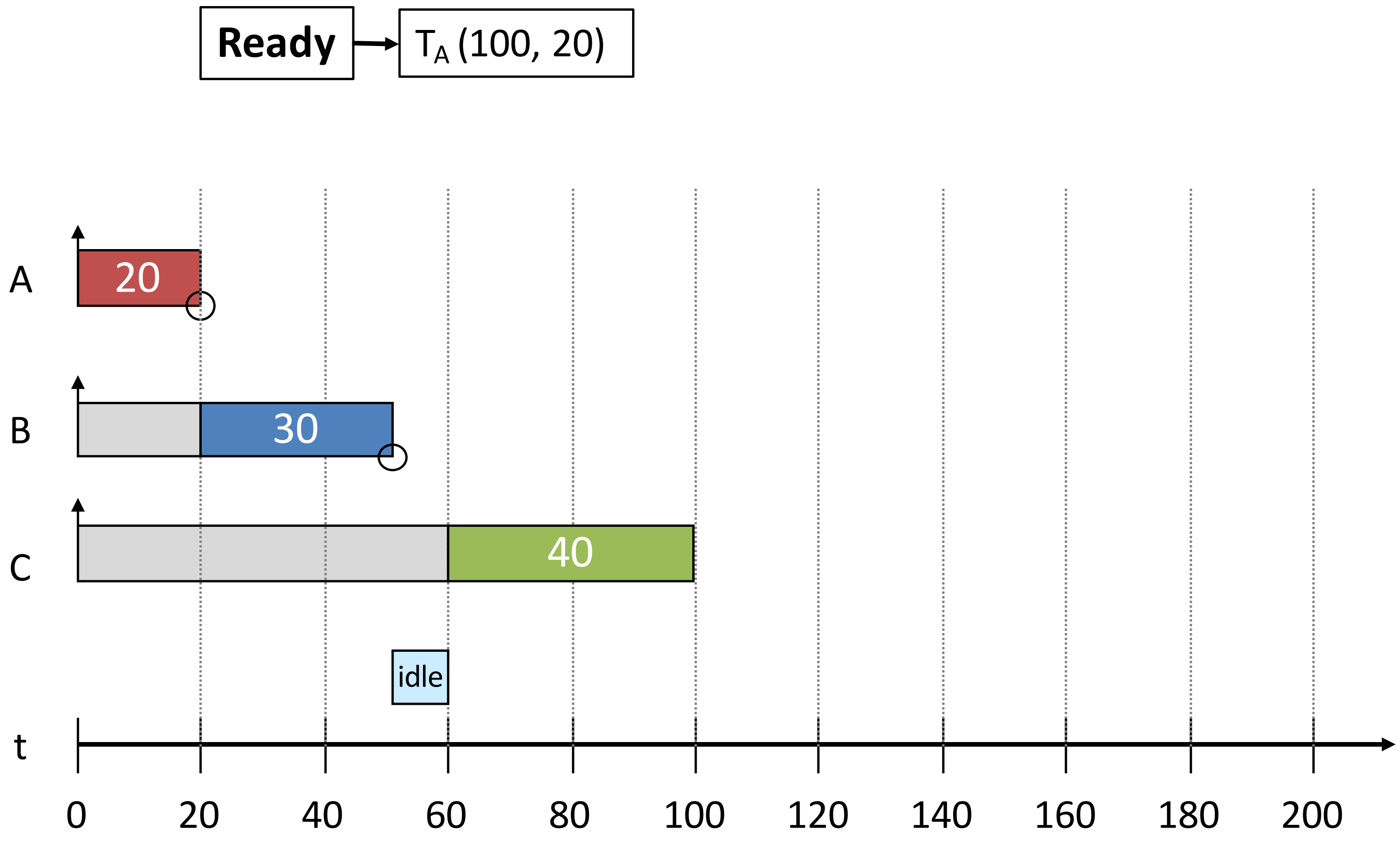
Preemptive Fixed Priority Scheduling



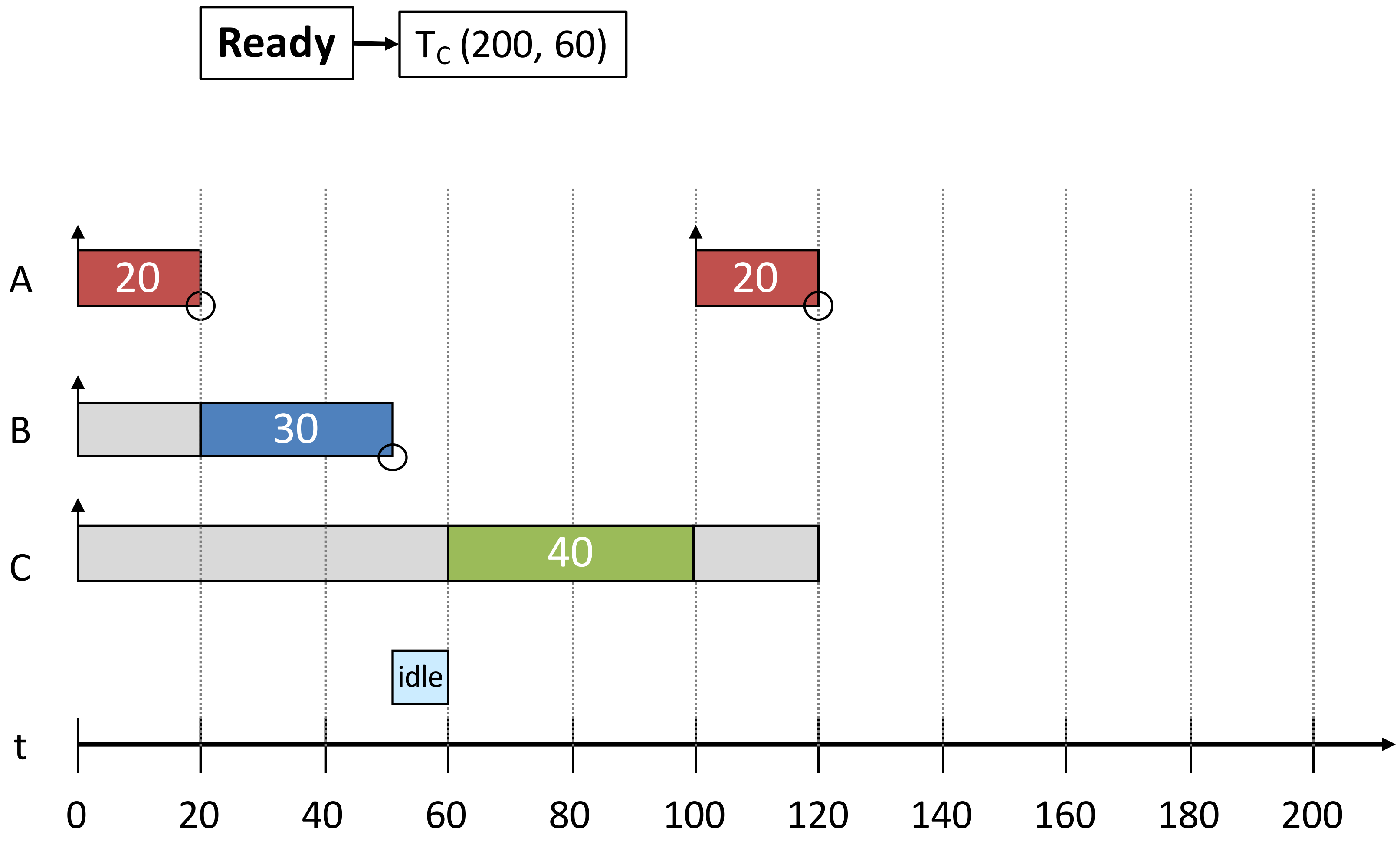
Preemptive Fixed Priority Scheduling



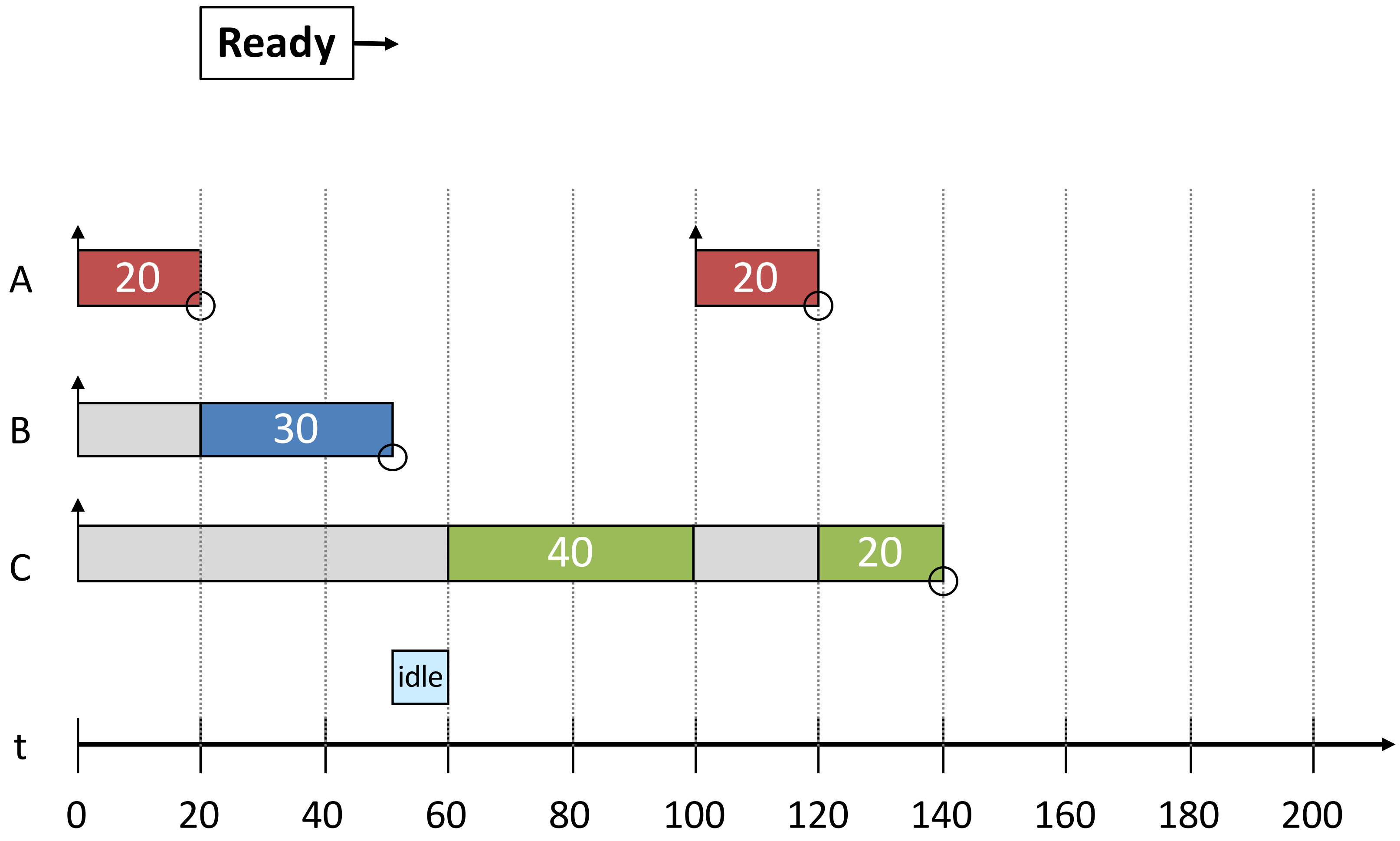
Preemptive Fixed Priority Scheduling



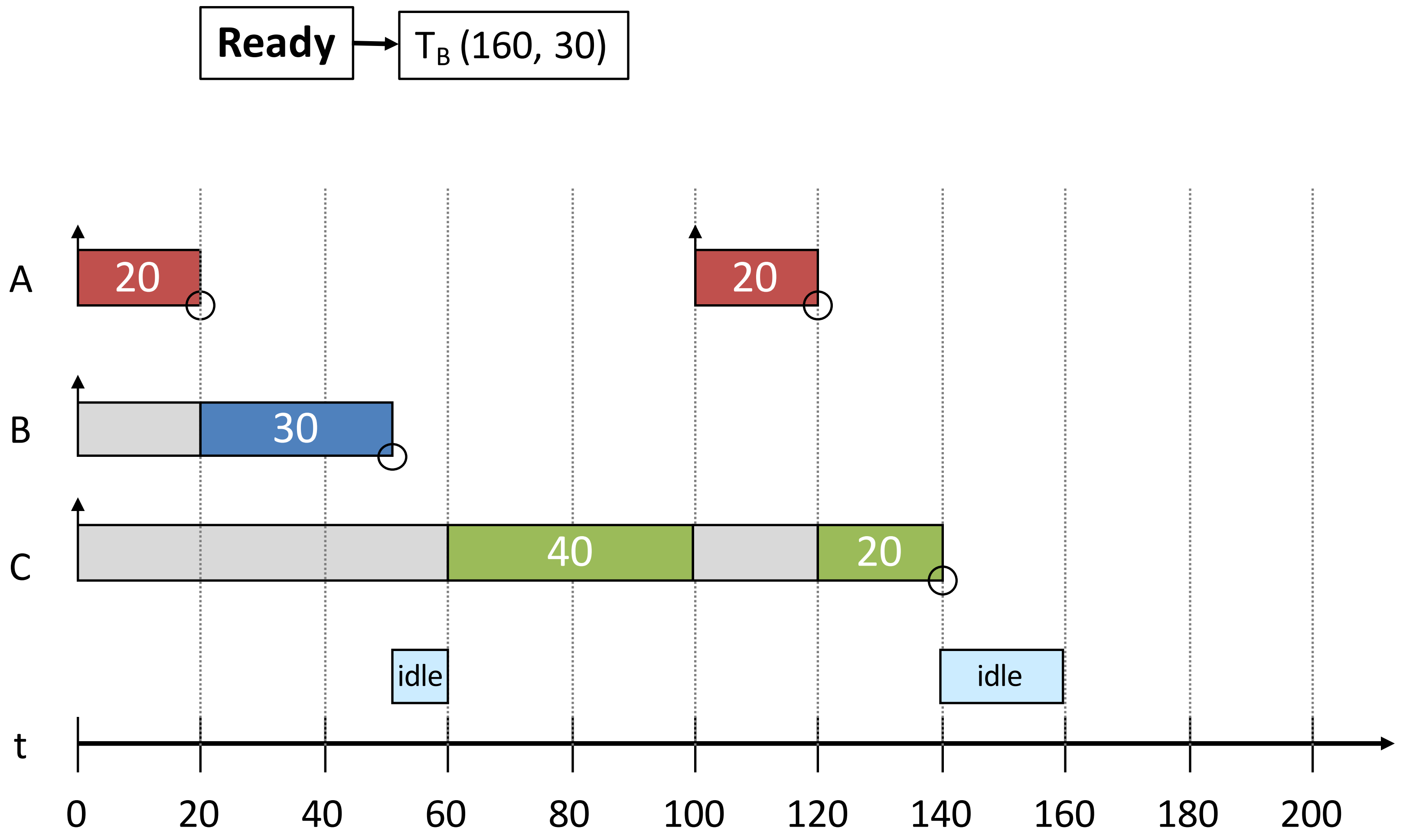
Preemptive Fixed Priority Scheduling



Preemptive Fixed Priority Scheduling



Preemptive Fixed Priority Scheduling



RM is optimal

Optimal algorithms: an algorithm which always produces a feasible schedule, if one exists, is said to be optimal.

Rate monotonic priority assignment is indeed optimal.

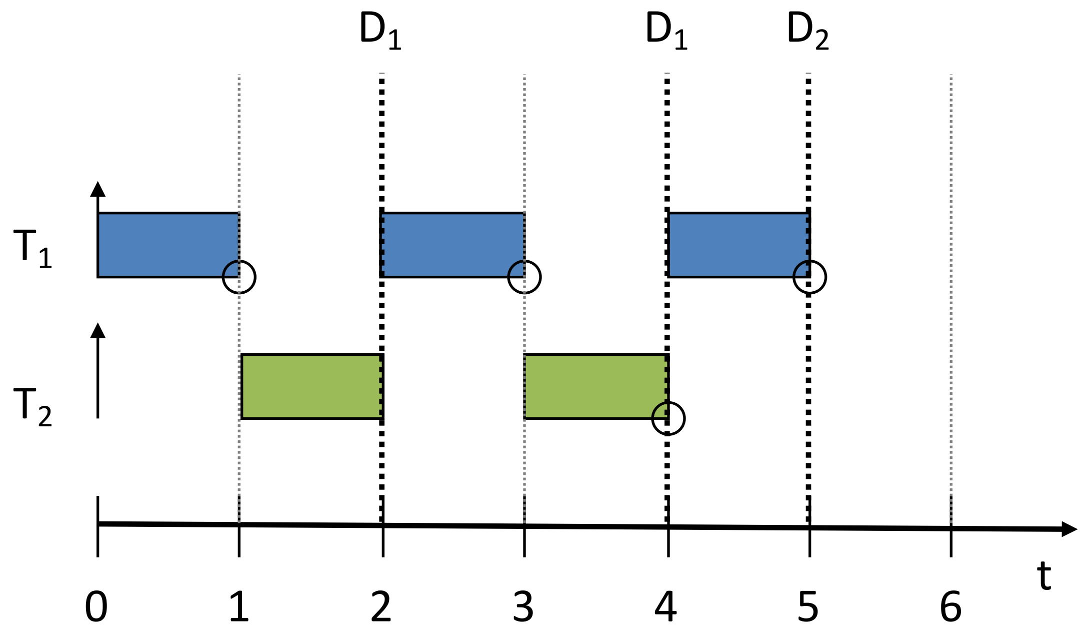
RM is optimal

- Assumptions:
  1. Tasks are periodic and the period is constant
  2. Completion-time < period
  3. Tasks are independent
  4. Runtime is known and deterministic
  5. all system overheads are negligible or deemed to be included in task computation times
  6. Critical instant - defined as the maximum load condition when all tasks release together

- Constraints
  1. Deadline = period
  2. fixed set of tasks
  3. Preemptive

RM is optimal

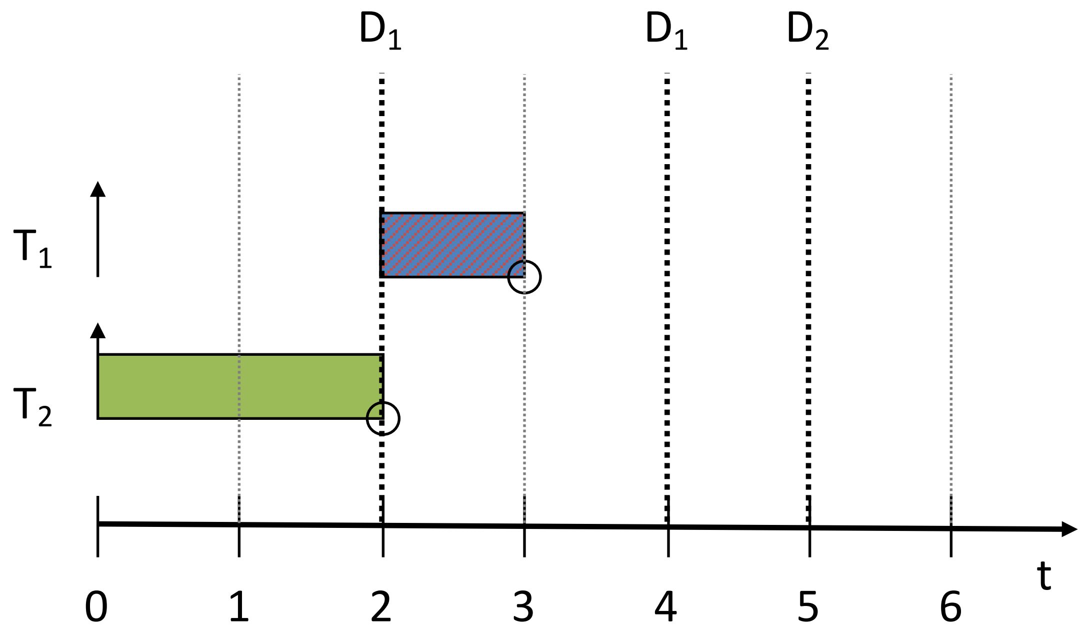
$T_1 = (2,1)$ ①
$T_2 = (5,2)$ ②



RM is optimal

$T_1 = (2,1)$ 2

$T_2 = (5,2)$ 1



RM is optimal

But it does not tell us anything as to whether a set of tasks is schedulable.

RM Least Upper Bound

$$U = \sum_{i=0}^{m} (e_i / p_i) \leq m(2^{\frac{1}{m}} - 1)$$

U: Utilization of the CPU that is achievable

e_i: Execution time of task i

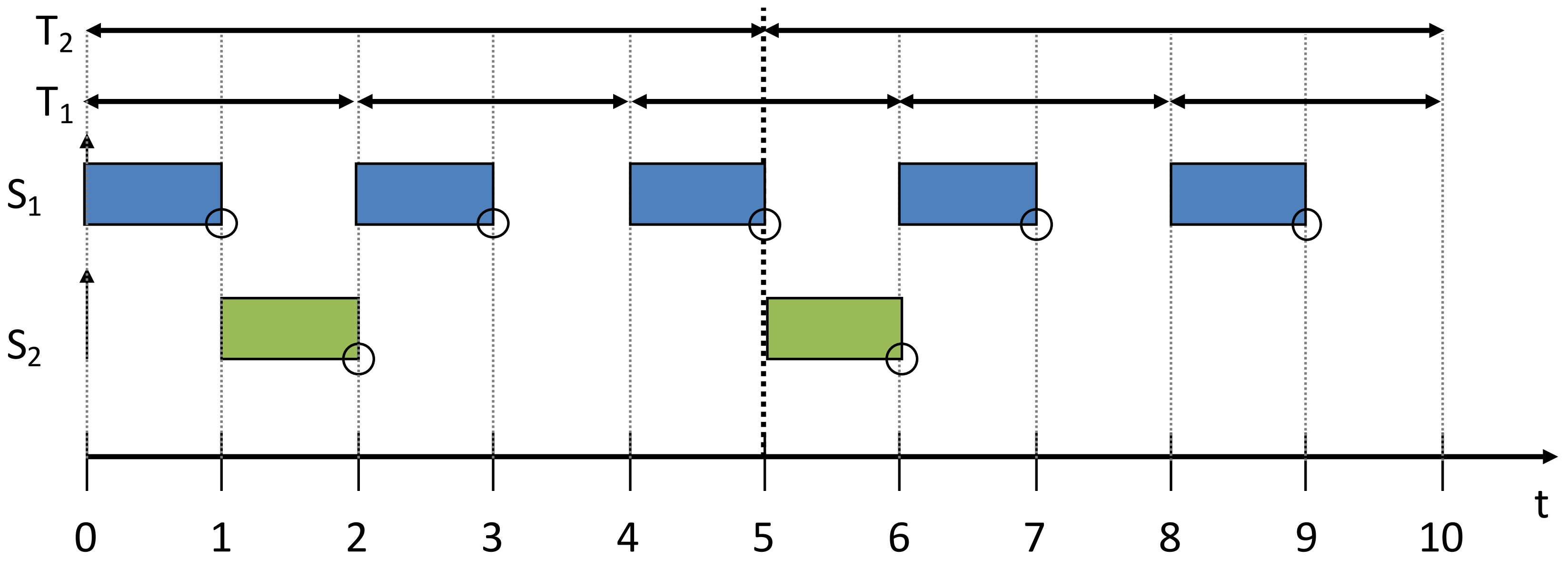
m: Total number of tasks sharing common CPU resources

p_i: Release period of task i

RM Least Upper Bound

$S_1 = (2,1)$
$S_2 = (5,1)$

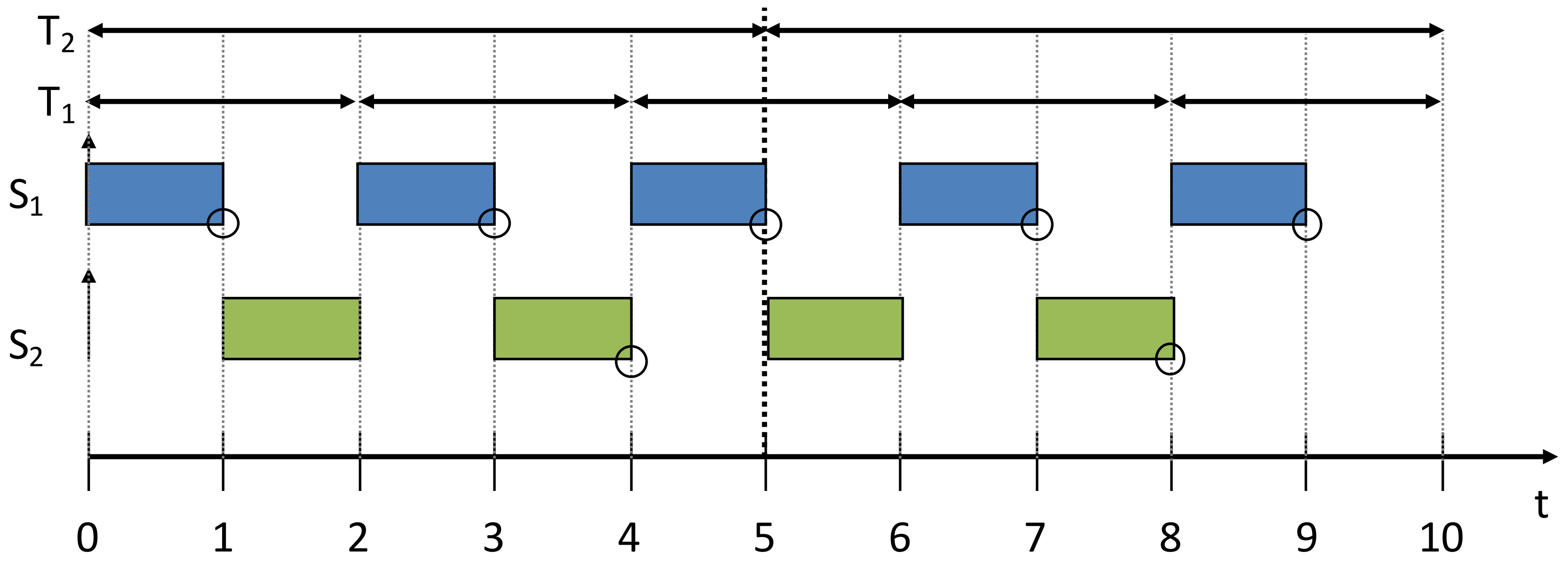
$U = 1/2 + 1/5 = 0.7$
$U = 0.7 < 2(2^{1/2} - 1) = 0.83$



RM Least Upper Bound

$S_1 = (2,1)$
$S_2 = (5,2)$

$U = 1/2 + 2/5 = 0.9$
$U = 0.9 > 2(2^{1/2} - 1) = 0.83$

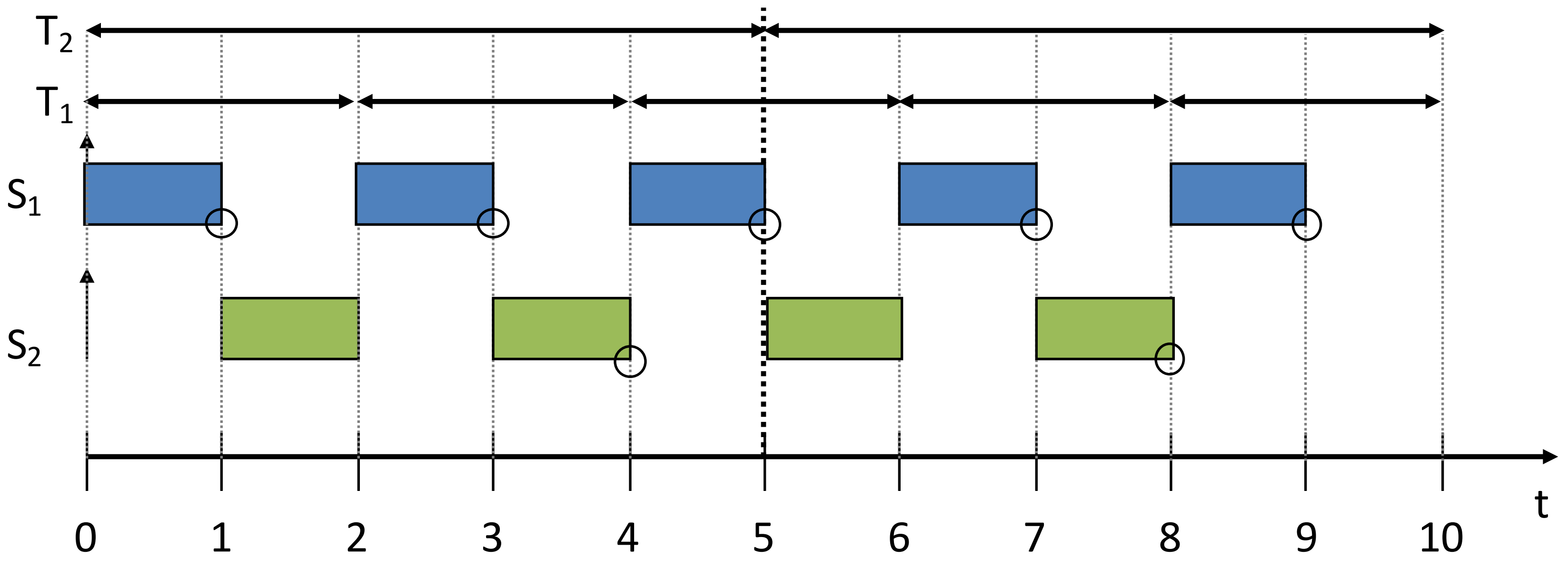


RM Least Upper Bound

$S_1 = (2,1)$
$S_2 = (5,2)$

$U = 1/2 + 2/5 = 0.9$
$U = 0.9 > 2(2^{1/2} - 1) = 0.83$

?



RM Least Upper Bound

a set of *m* independent periodic tasks scheduled by the rate monotonic least upper bound algorithm will <u>always</u> meet its deadlines, if

$$U = \sum_{i=0}^{m} (e_i / p_i) \leq \underbrace{m(2^{\frac{1}{m}} - 1)}_{U_{RM}}$$

U_{RM} converges to 69.3% for large *m*

- a **sufficient** but **not necessary** test
  - i.e. we **cannot** say that a system with higher utilization is **not** schedulable with this scheduling algorithm

RM LUB Example 1

Task	e_i	p_i	U_i
A	20	100	
B	30	150	
C	60	200	

RM LUB Example 1

Task	e_i	p_i	U_i
A	20	100	0.2
B	30	150	0.2
C	60	200	0.3

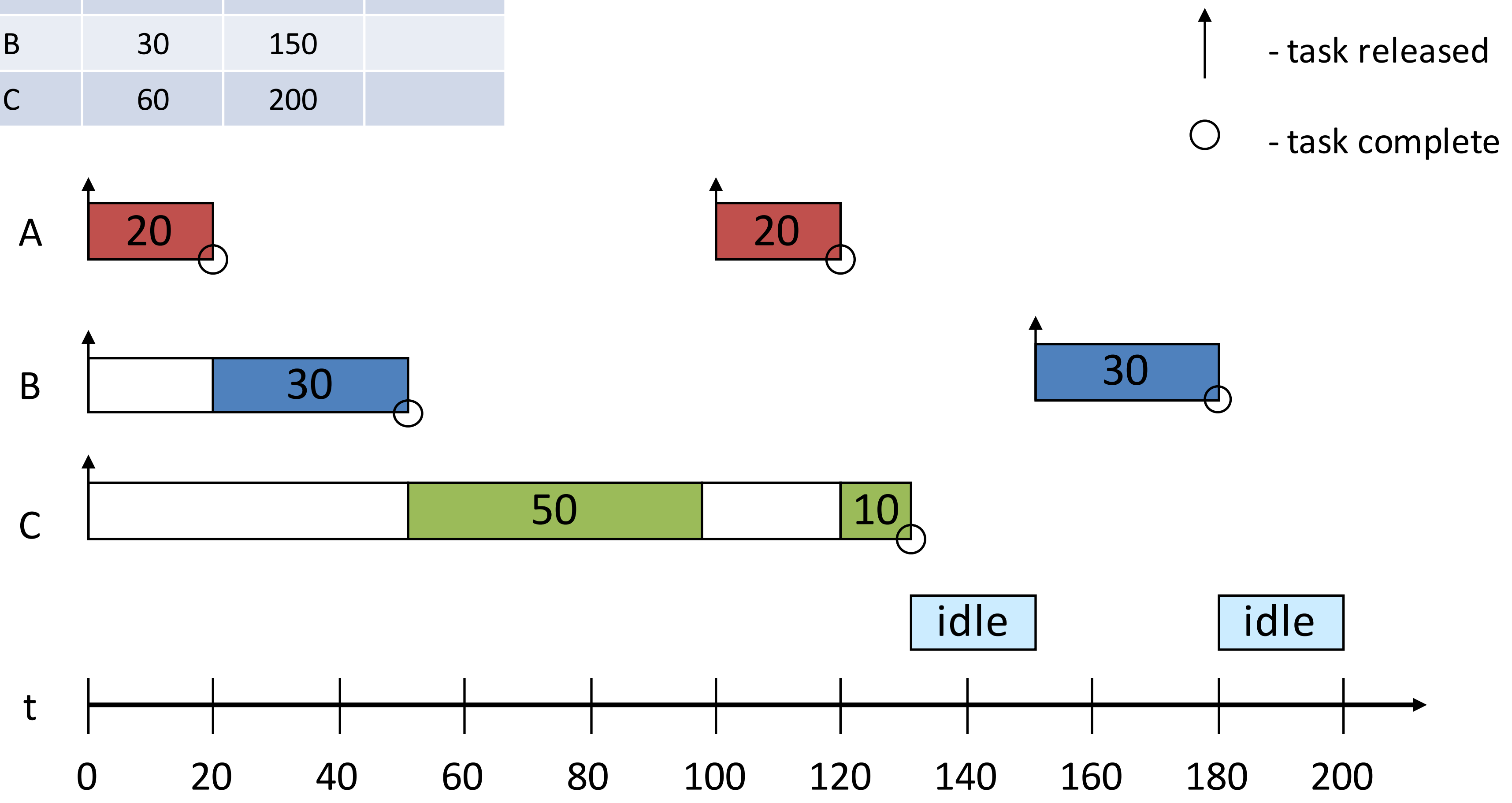
$U_1 + U_2 + U_3 = 0.7$, and is smaller than $U_{RM}(3)$
where $U_{RM}(3) = 3*(2^{1/3} - 1) = 0.779$

Therefore these tasks will always meet their deadlines under the scheduling scheme:

$P(A) = 1, \; P(B)=2, \; P(C)=3$

RM LUB Example 1

Task	e_i	p_i	U_i
A	20	100	
B	30	150	
C	60	200	



RM LUB Example 2

Task	e_i	p_i	U_i
A	12	50	
B	10	40	
C	10	30	

RM LUB Example 2

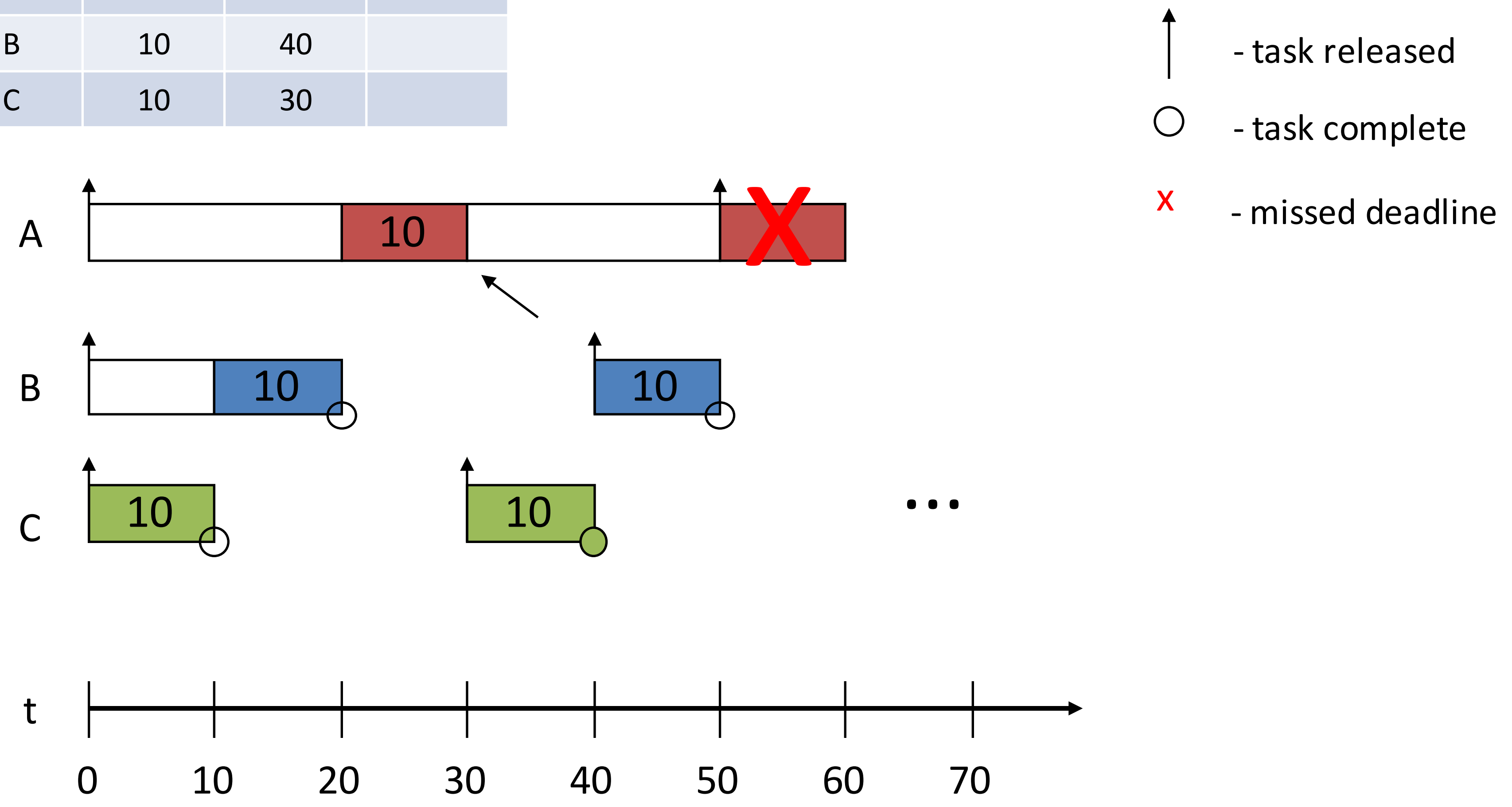
Task	e_i	p_i	U_i
A	12	50	0.24
B	10	40	0.25
C	10	30	0.33

$U_1 + U_2 + U_3 = 0.82$, and is not $\leq U_{RM}(3)$
where $U_{RM}(3) = 3*(2^{1/3} - 1) = 0.779$

Therefore these tasks fail the utilization test, and by examination of their Gantt chart, it can be seen that all deadlines will not be met

RM LUB Example 2

Task	e_i	p_i	U_i
A	12	50	
B	10	40	
C	10	30	



RM LUB Example 3

Task	e_i	p_i	U_i
A	40	80	
B	10	40	
C	5	20	

RM LUB Example 3

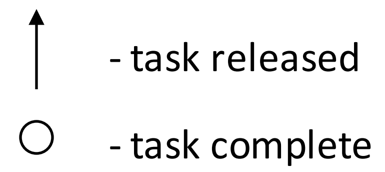
Task	e_i	p_i	U_i
A	40	80	0.5
B	10	40	0.25
C	5	20	0.25

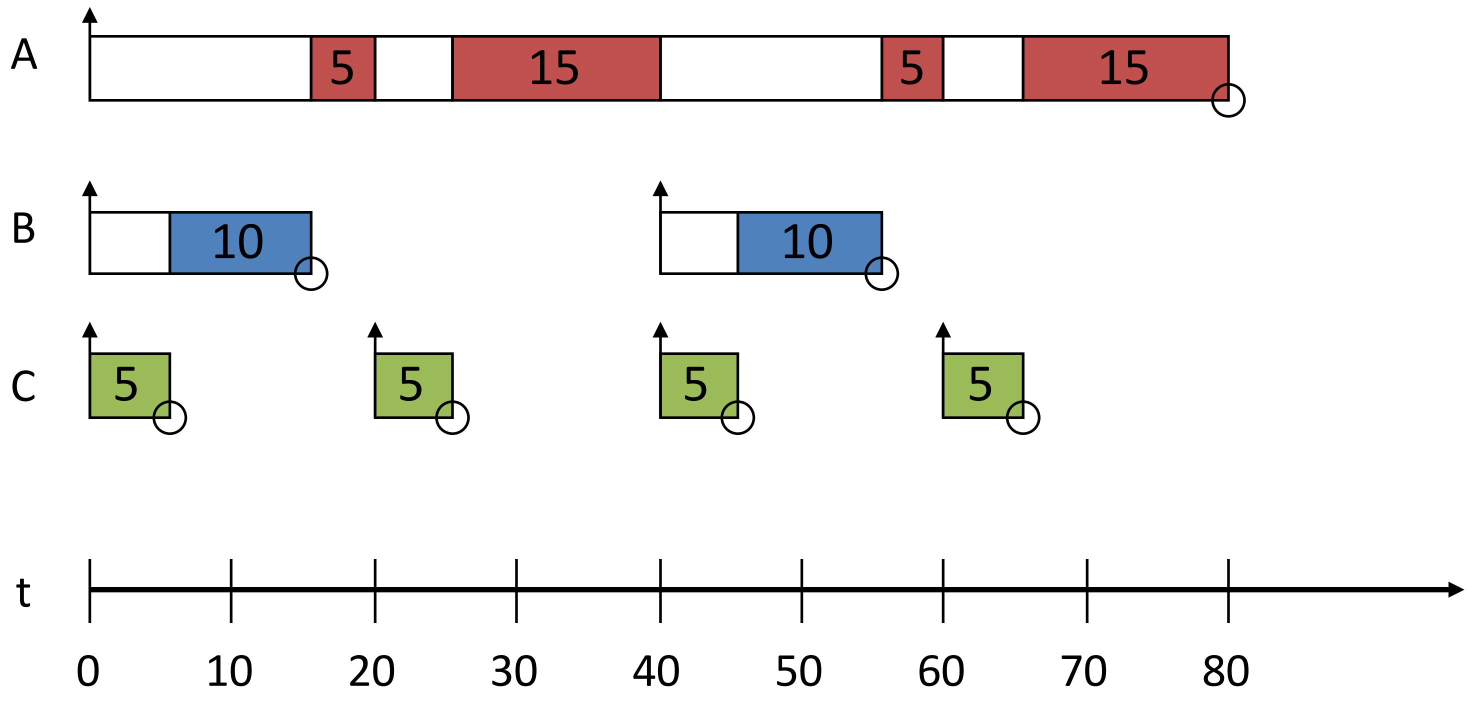
$U_1 + U_2 + U_3 = 1.00$, and is not $\leq U_{RM}(3)$
again where $U_{RM}(3) = 3*(2^{1/3} - 1) = 0.779$

These tasks clearly fail the utilization test, but after examination of their Gantt chart, it can be seen that all deadlines will be met ????

RM LUB Example 3

Task	e_i	p_i	U_i
A	40	80	
B	10	40	
C	5	20	





Additional Points on RM LUB

- RM LUB has the advantage of being stable in conditions of transient overload
  - a subset of the tasks (those with the highest priorities) will still meet their deadlines even in a temporary overload condition.

- <u>Completion Time Theorem</u>
  - for a set of independent periodic tasks, if each task meets its first deadline when all tasks are started at the same time (critical instant), then the deadlines will be met every time.

Weaknesses of RM LUB

- utilization test is not exact
  - sufficient, but not necessary
  - therefore other tests must be used (besides manually checking the feasibility)
- the simple task model is too restrictive
  - only allows for periodic tasks
  - all tasks must be independent
    - implies no shared resources (no mutexes)

References

[1] Siewert, S. Pratt, J. Real-Time Embedded Components and Systems with Linux and RTOS. Mercury Learning and Information, 2016.

[2] Burns, A. and Wellings, A., "*Real-Time Systems and Programming Languages*", Chapter 13, Addison Wesley, 1997

[3] TimeSys Corp, "*The Concise Handbook of Real-Time Systems*", Version 1.0, 1999