Real-Time Systems
Systèmes Temps-Réel embarqués

Real-time embedded systems

Embedded systems:

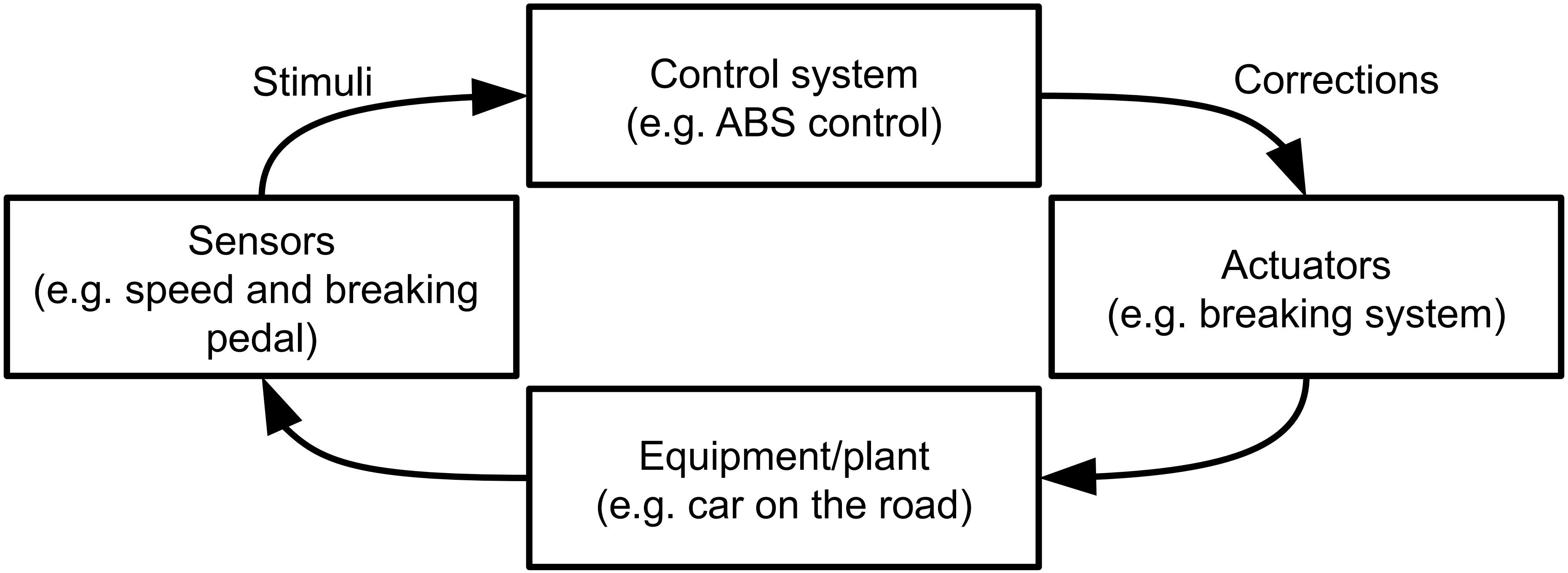
- « An application whose prime function is not that of information processing, but which nevertheless requires information processing in order to carry out their prime function » – Real-Time Systems and programming langages, A. Burns and A. Wellings
- Ex : Microprocessor-controlled washing machine

Real-time systems:

- « Any system in which the time at which output is produced is significant » – Oxford dictionnary of computing
- Outputs must be produced within a certain duration (**deadline**) after input stimuli have been produced by the **environment** (subcategory of **reactive systems**)
- Real-time system = functional correctness + **temporal correctness**
- Ex: Control system for airbags, ABS, etc.

Real-time embedded systems

Ex: embedded control systems:



Real-time embedded systems

Soft Real-time:

- Deadlines can be missed occasionaly, or service can occasionaly be delivered late
- Ex: telemetry report system, car window controller, multimedia application, etc.

Hard Real-Time:

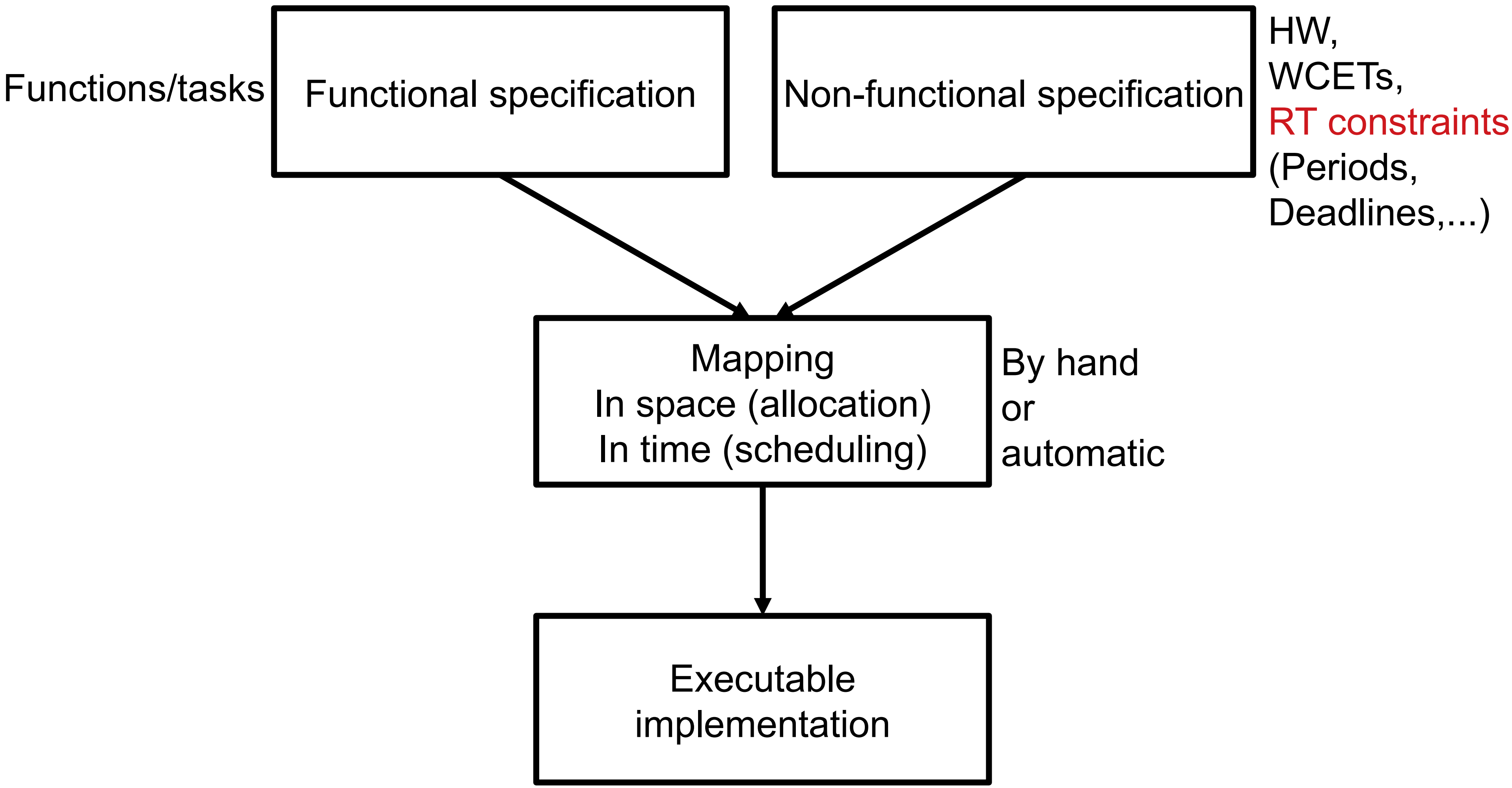
- Any deadline miss is considered a system failure and may be **catastrophic**
- Critical systems : systems for which a timing failure may cause human harm or loss, environmental disaster or expensive mission failure
- Ex: aircraft control system, nuclear plant, etc.

Real-time embedded systems

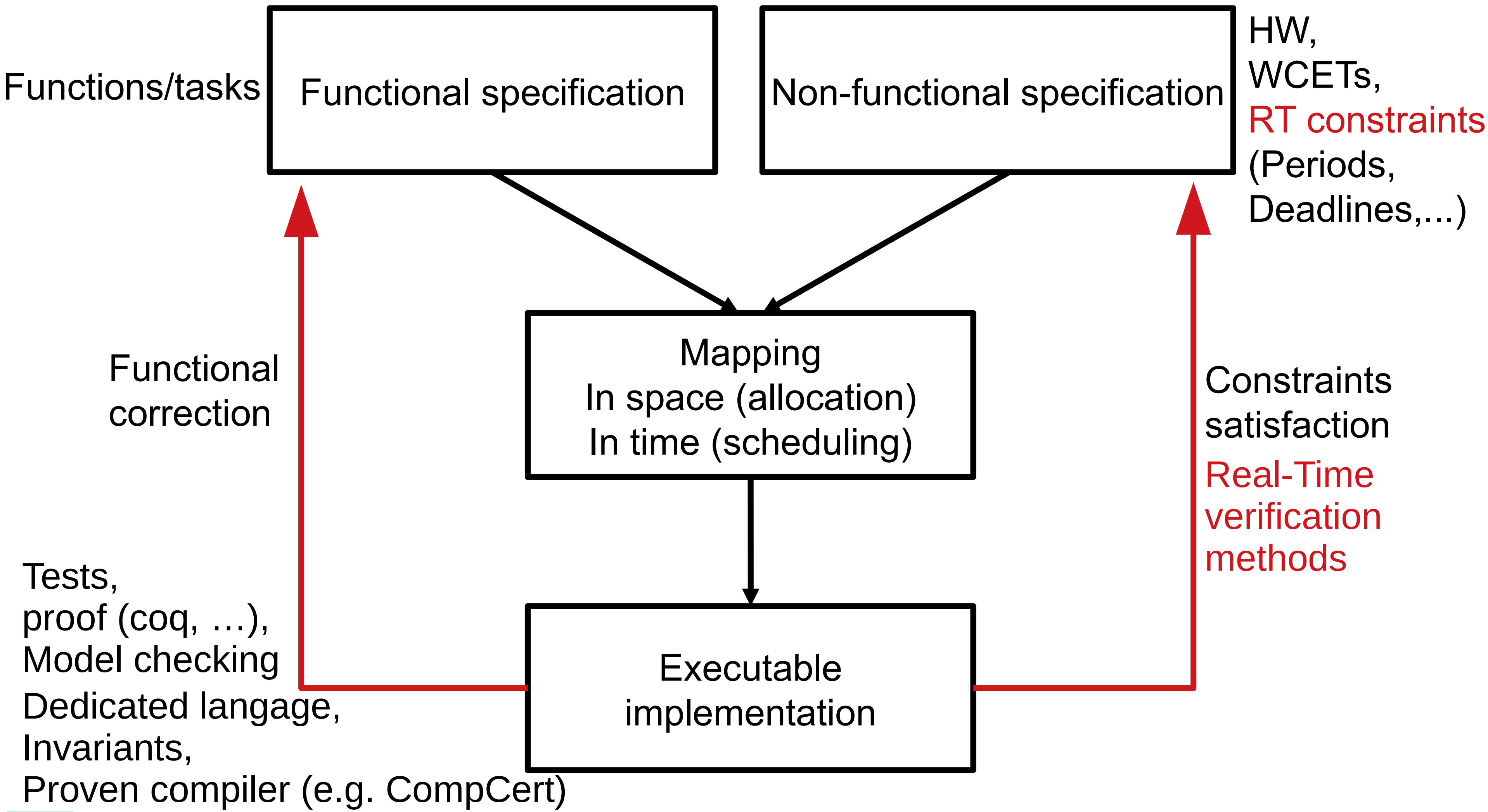
Engineering a real-time system:

- Specification: functional and non-functional requirements
  - Functional: What the system must compute ? Defining **tasks**
  - Non-functional: On what hardware ? With which temporal requirements ?
- Implementation: producing the system with constraints
  - Languages: non-synchronous (C, ADA, …) vs synchronous (lustre/SCADE, esterel, …)
  - Cyclic executive vs concurrent. Linking **specification tasks** to **execution threads/processes**
  - Generating static code vs generating scheduler configuration
- Verification: verifying the requirements on the produced system
  - Functional (not part of this course)
  - Non-functional (**schedulability analysis**)

Real-time embedded systems



Real-time embedded systems



Specification

Real-Time requirements sources

- Laws of nature:
  - Objects in motion, inertia, etc.
  - => End-to-end response time constraints/deadlines
- Control theory:
  - Execution frequency/period
- Hardware limitations:
  - Sensor sampling rate
  - Actuation rate
  - => Periods

Specification

Our running example: a simplified car control system

- Functional requirements:
  - Implement an Engine Control Unit:
    - Samples inputs from a pedal angle sensor, a speed sensor and an engine rotation sensor
    - Writes to a controller in the engine to control the rotation speed
  - Implement an Airbag Control Unit:
    - Samples inputs from a collision detector
    - Triggers the airbag if necessary
- Real-time requirements:
  - Sampling rate for the sensors:
    - Pedal angle, speed: 10ms
    - Engine rotation: 20ms
    - Collision detection: 60ms
  - Actuation rates:
    - ECU function (control theory): 40ms
    - Airbag function (bodies in motion): 60ms

Specification

Most common abstraction: real-time tasks

- Tasks: the functional units of the system (e.g. functions or groups of functions)
- Periodic tasks: mostly used
  - Period (T_i): the duration between two consecutive activations of a task
  - Relative deadline (a.k.a. deadline) (D_i): the duration between a task instance arrival date and the date at which it must complete its execution
  - Execution time budget (C_i) => Worst-case execution time
- Periodicity induces multiple instances (jobs)
  - Arrival date (A_{ij}): the date at which job j of task i is ready for scheduling (a.k.a. release date)
  - Absolute deadline: the date at which a job must complete its execution

Specification

Definitions:

- Tasks: the functional units of the system (e.g. functions or groups of functions)
- Periodic tasks: mostly used
  - Sometimes characterized only with T_i and C_i (when D_i=T_i)

Specification

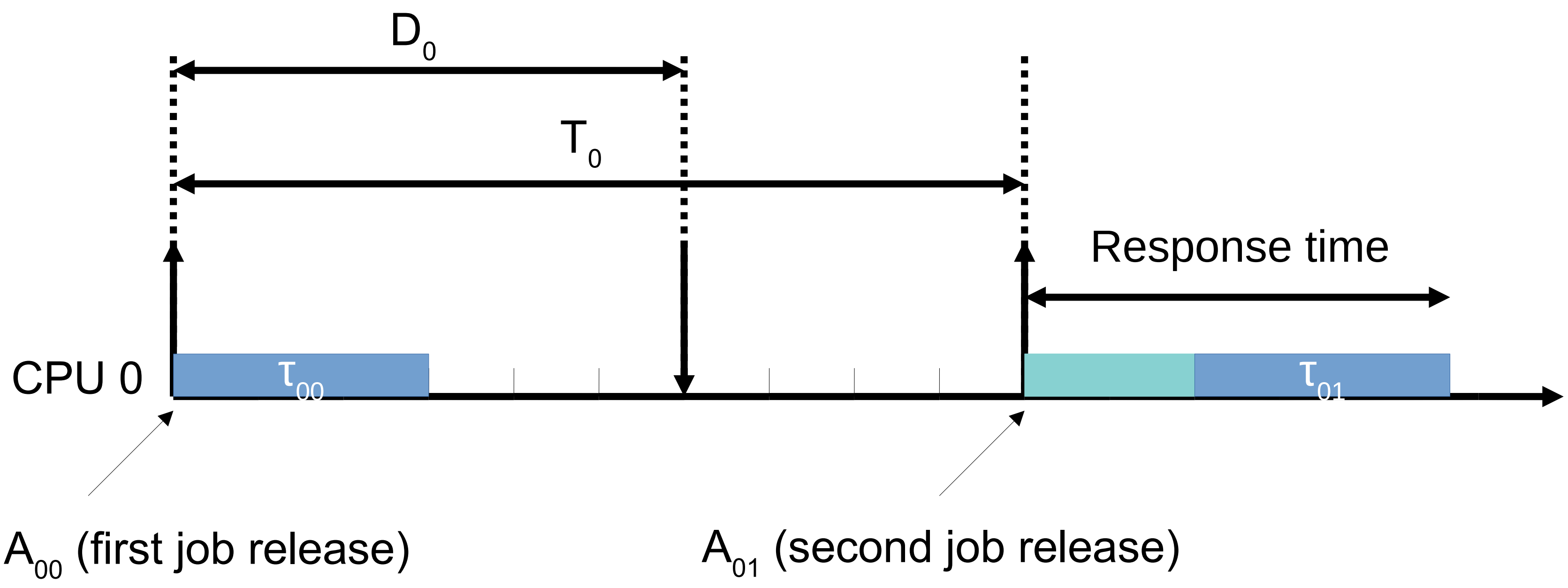
Definitions:

- Tasks: the functional units of the system (e.g. functions or groups of functions)
- Periodic tasks: mostly used
- Sporadic tasks:
  - Not periodic, but minimum inter-arrival duration is known
  - E.g. fault-tolerance routines, some sensor polling tasks
- Aperiodic tasks:
  - Can happen anytime, makes verification hard
  - Can be tackled by adding periodic « server » tasks

Specification

Example:

- τ_0 : T_0=10, D_0=6, C_0=3



Specification

Our running example: a simplified car control system

- Functional requirements:
  - Implement an Engine Control Unit:
    - Samples inputs from a pedal angle sensor, a speed sensor and an engine rotation sensor
    - Writes to a controller in the engine
  - Implement an Airbag Control Unit:
    - Samples inputs from a collision detector
    - Triggers the airbag if necessary
- Real-time requirements:
  - Sampling rate for the sensors:
    - Pedal angle, speed: 10ms
    - Engine rotation: 20ms
    - Collision detection: 60ms
  - Actuation rates:
    - ECU function (control theory): 40ms
    - Airbag function (bodies in motion): 60ms

How to capture the requirements in the periodic task model ?

Specification

Our running example: a simplified car control system

- Real-time requirements:
  - Sampling rate for the sensors:
    - Pedal angle, speed: 10ms
    - Engine rotation: 20ms
    - Collision detection: 60ms
  - Actuation rates:
    - ECU function (control theory): 30ms
    - Airbag function (bodies in motion): 60ms

Engine Control Unit (ECU)

T_0=10ms ?

T_0=30ms ?

Airbag Control Unit

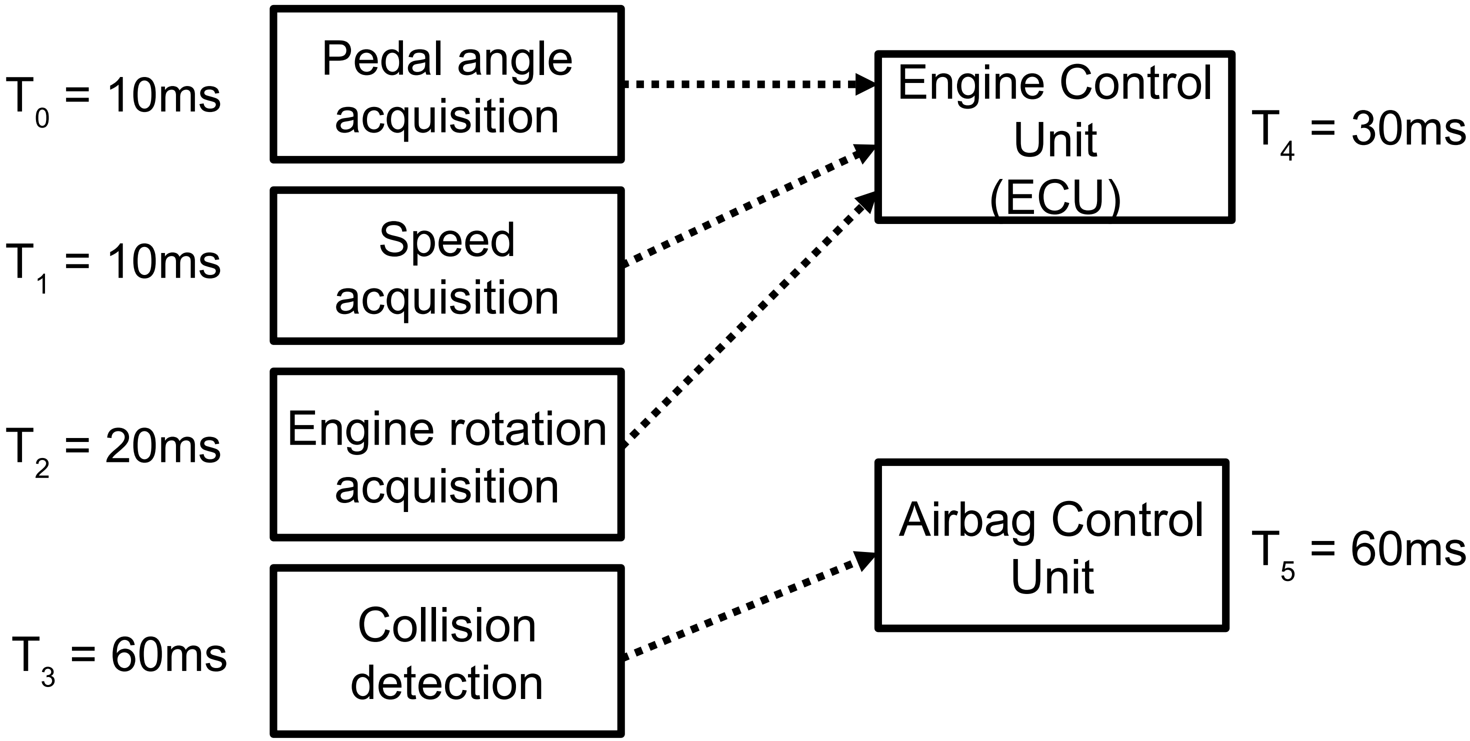
T_1=60ms

- Original model: One task per functionality (coarse-grain tasks)
  - Each is responsible for input acquisition, correction computation and actuation
  - Very simple system design, complexity hidden in the tasks
    - In particular, different sampling rates
    - => Tasks may be harder/sometimes impossible to implement
    - => Tasks WCET are harder to derive
  - => Strong abstraction, usually leads to pessimism overhead in validation phase
  - => Hard to optimize at system-level

Specification

Our running example: a simplified car control system

- Model relaxation: one task per function (fine grain tasks)
  - For now, we do not take into account inter-task communications



Specification

Our running example: a simplified car control system

- Model relaxation: one task per function (fine grain tasks)
  - For now, we do not take into account inter-task communications
  - System design is more complex, tasks are simpler to implement
  - Abstraction is weaker
    - => system-level analysis is harder, but optims are possible
    - => small WCET overhead when analyzing tasks
- Choosing the task system:
  - Transforms requirements into constraints
    - Constraints may be stricter than requirements (cost of abstraction): cf original model in the example
  - Is the first step of implementation (limit between specification and implementation is blurred)

Implementation

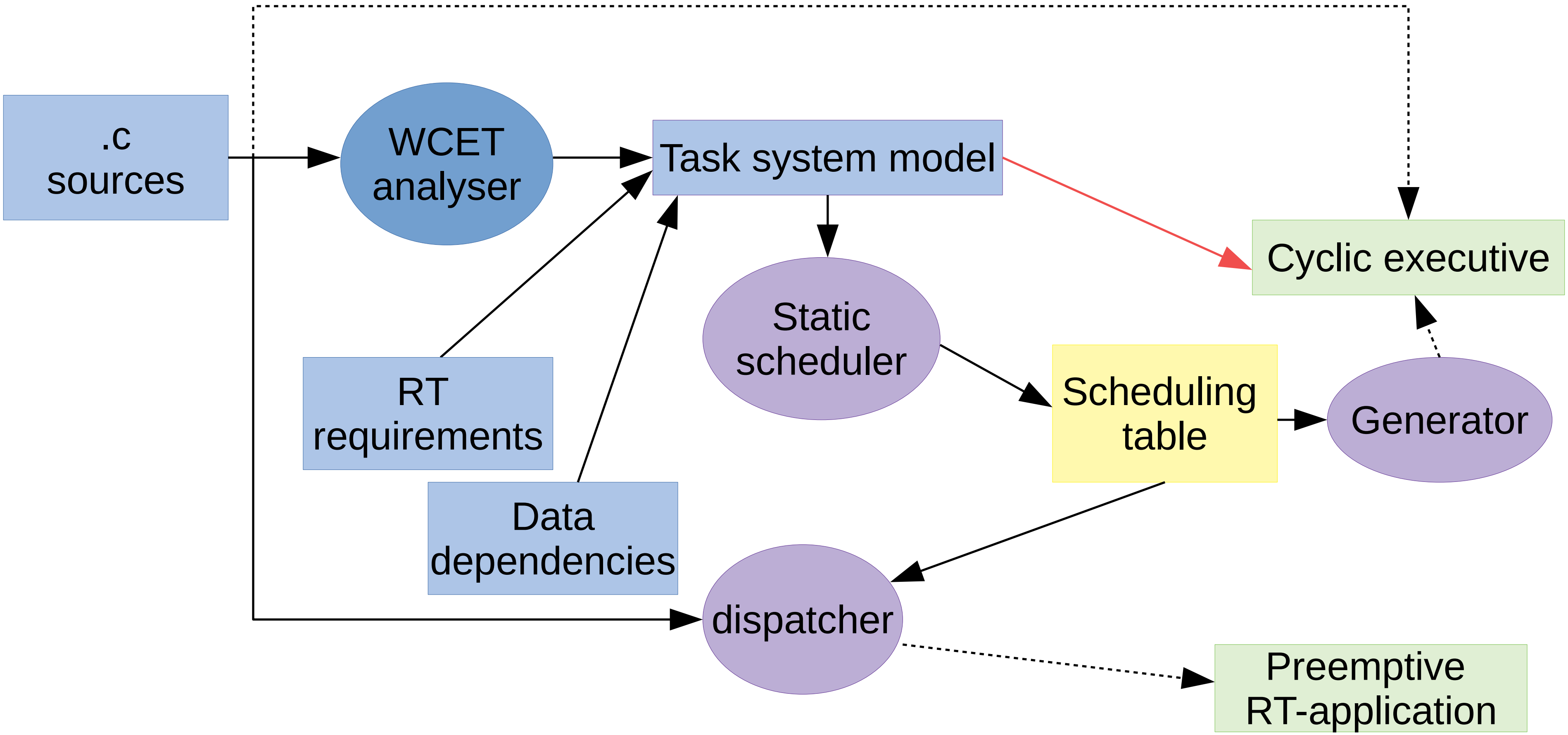
Two main implementation families:

- Static implementation: everything is fixed offline
  - Produce a static schedule i.e. order of execution of the tasks, sometimes also their starting dates
    - Produce a cyclic executive, then check it

    OR
    - Produce a scheduling table with the start date of each task and the code for each task (processes or threads)
  - Easy to verify the RT properties => good for very critical parts
- Dynamic implementation: decisions are made online (during the execution)
  - Implement/choose a scheduler to make the decisions at runtime
    - Choose a scheduling policy
  - Produce tasks code (processes or threads)
  - Verification is harder but implementation is easier, more modular and robust

Static (off-line) scheduling

Static scheduling:



Static (off-line) scheduling

Cyclic executive:

- Basically, one infinite loop which contains calls to the tasks' functions
  - Only one process/thread

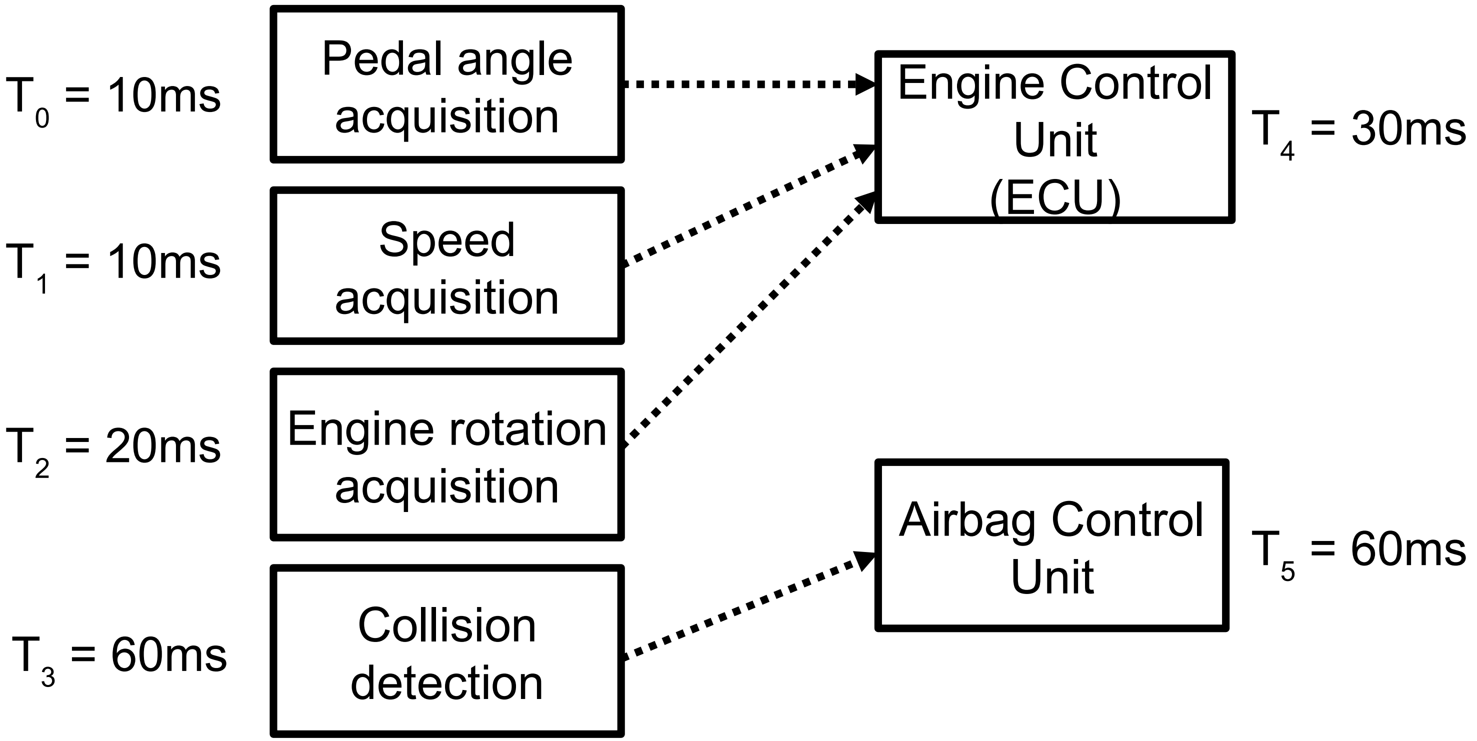
```
Loop:                               //Repeat indefinitely
        acquire_inputs();          //Sensor polling
        compute_corrections();
        write_outputs();           //Update actuators
```

- The main loop gives the activation period for the whole system
  - In the absence of synchronizations, the sum of the WCETs of the tasks' functions gives the response time AND the worst case period

Static (off-line) scheduling

Our running example: a simplified car control system

- Second version: one task per function (fine grain tasks)
  - For now, inter-task communications can be ignored (using buffers)



Static (off-line) scheduling

Cyclic executive:

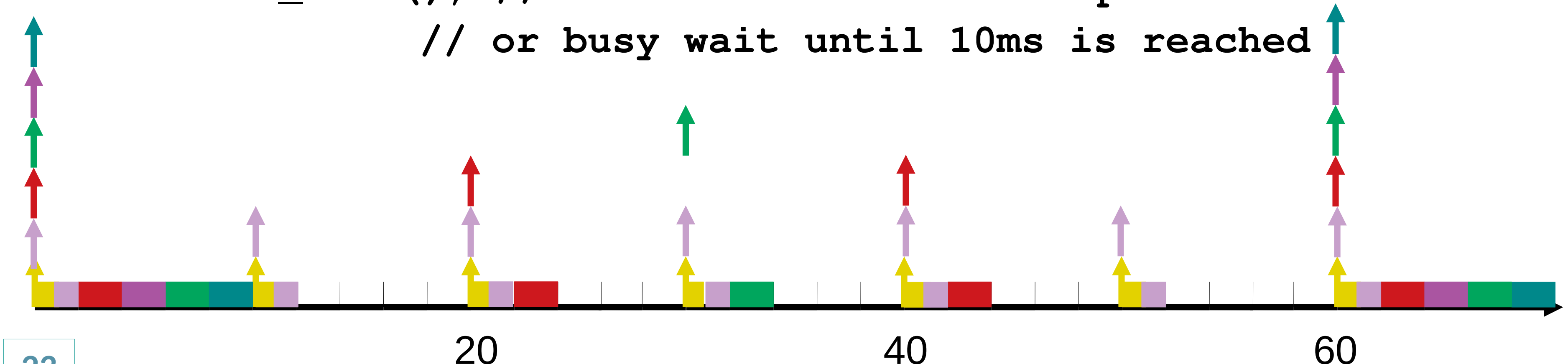
```
loop :
      pedal_angle();
      speed();
      if(i%2==0) engine_rotation();
      if(i%6==0) collision_detection();
      if(i%3==0) ECU();
      if(i%6==0) airbag();
      i++;
      wait_10ms(); // wait for timer interrupt
                   // or busy wait until 10ms is reached
```

Static (off-line) scheduling

Cyclic executive:

```
loop :
        pedal_angle();
        speed();
        if(i%2==0) engine_rotation();
        if(i%6==0) collision_detection();
        if(i%3==0) ECU();
        if(i%6==0) airbag();
        i++;
        wait_10ms(); // wait for timer interrupt
                    // or busy wait until 10ms is reached
```

Period of loop is the period of the fastest task



Static (off-line) scheduling

Cyclic executive:

- Simple verification method: sum of worst-case execution times must be less than the period of the loop (and/or less than the smallest deadline)
  - Regardless of the order of function calls in the loop
  - Sufficient criteria, but not necessary
  - Exact criteria when all tasks have the same period

$$\forall i, \sum_{j} Cj \leq Di$$

$$\forall i, \sum_{j} Cj \leq Ti$$

- Exact verification method: build the worst-case schedule of the task system (one hyperperiod when all tasks start at date 0)
  - If no task exceeds its own deadline, the system is safe

Static (off-line) scheduling

Cyclic executive: sufficient vs exact method

loop :

$C_0=1$ `pedal_angle();`

$C_1=1$ `speed();`

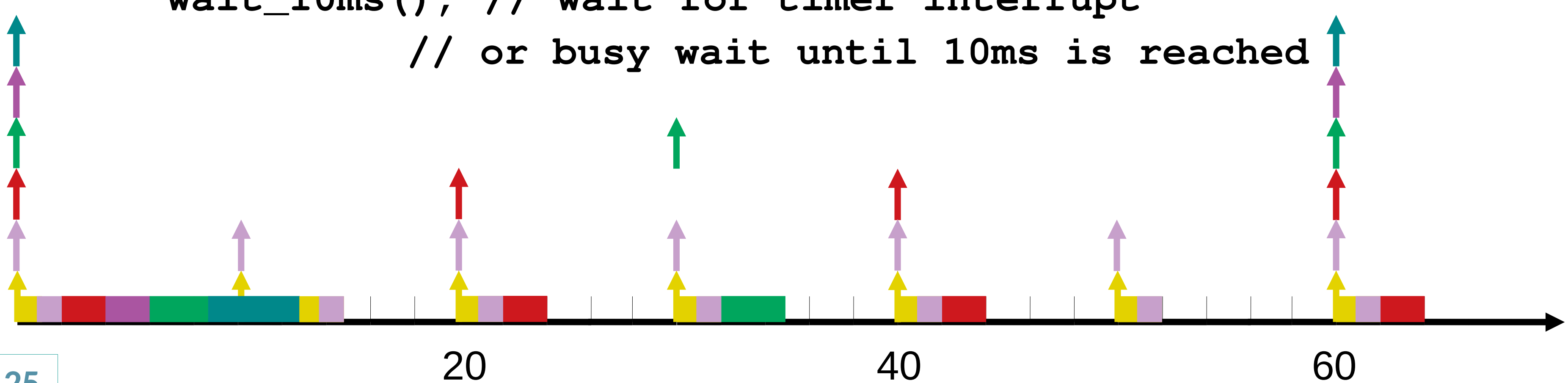
$C_2=2$ `if(i%2==0) engine_rotation();`

$C_3=2$ `if(i%6==0) collision_detection();`

$C_4=3$ `if(i%3==0) ECU();`

$C_5=4$ `if(i%6==0) airbag();`

```
i++;
wait_10ms(); // wait for timer interrupt
             // or busy wait until 10ms is reached
```



Static (off-line) scheduling

Cyclic executive: Problem with long tasks

`loop :`

$C_0=1$ `pedal_angle();`

$C_1=1$ `speed();`

$C_2=2$ `if(i%2==0) engine_rotation();`

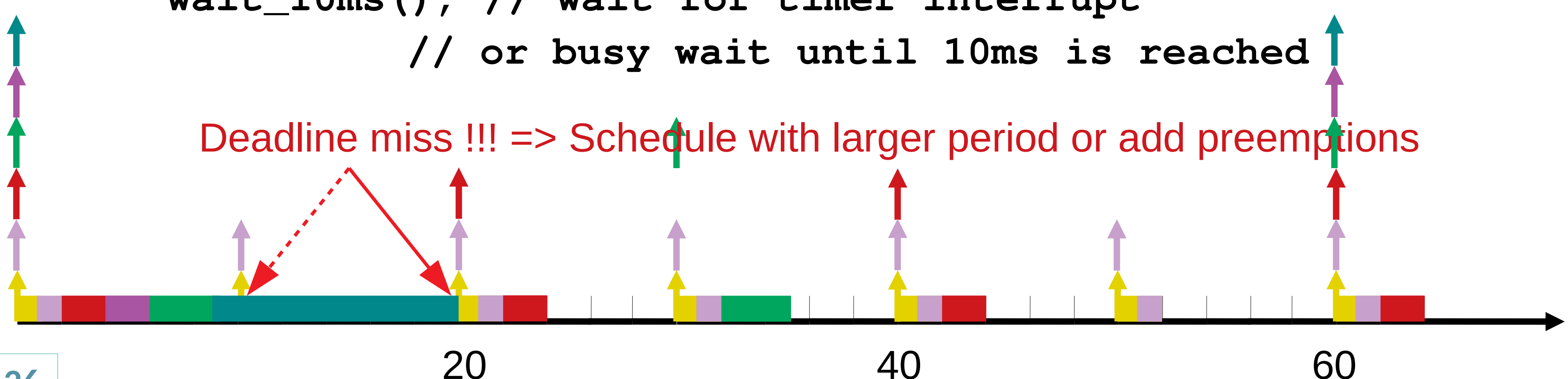
$C_3=2$ `if(i%6==0) collision_detection();`

$C_4=3$ `if(i%3==0) ECU();`

$C_5=12$ `if(i%6==0) airbag();`

```
i++;
wait_10ms(); // wait for timer interrupt
             // or busy wait until 10ms is reached
```

Period of loop is the period of the fastest task=> if a task takes longer than this period we have a deadline miss



Static (off-line) scheduling

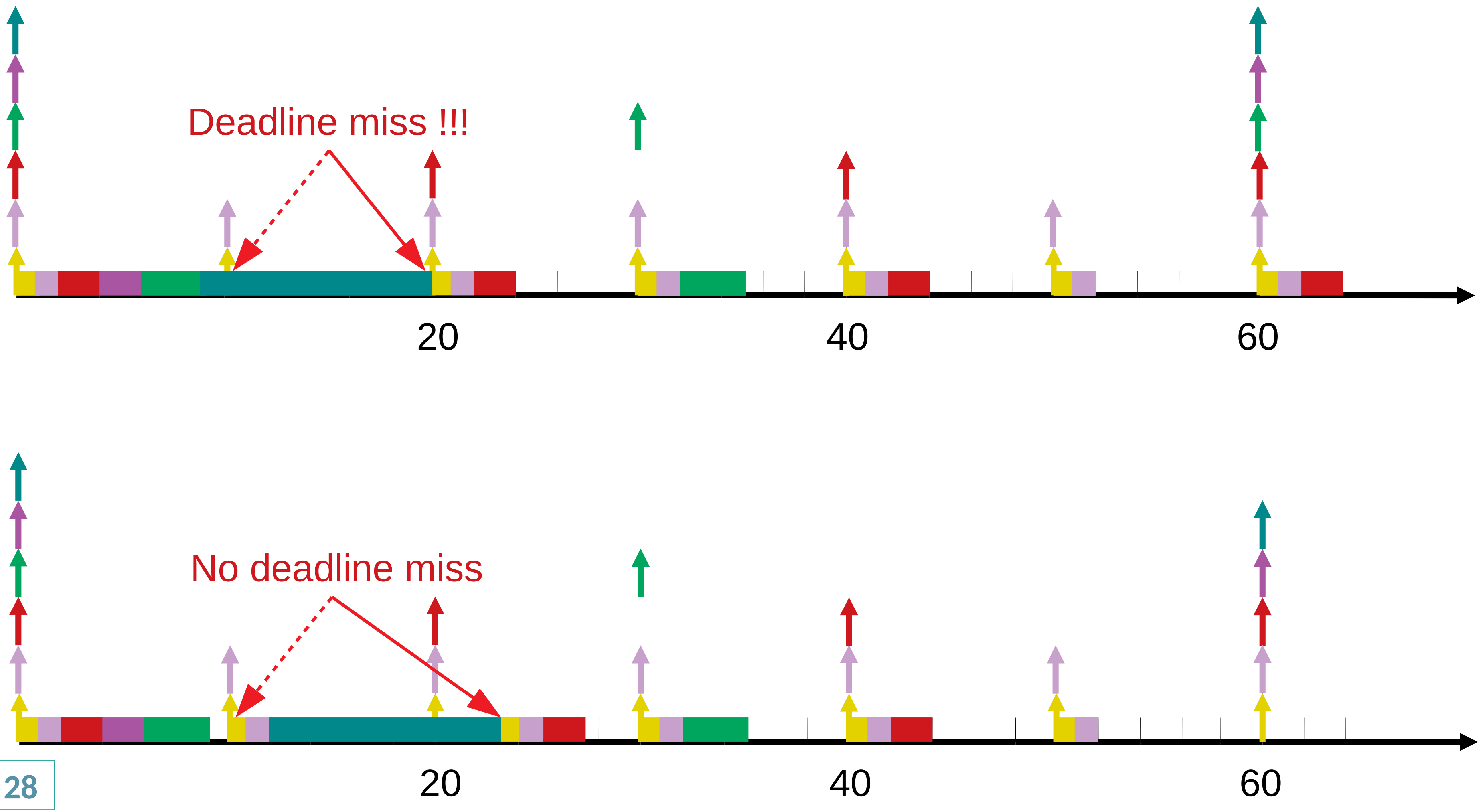
Scheduling table:

- Represents the starting dates of the tasks in each hyperperiod (lcm of tasks periods)
  - Program object
- Used to generate and/or verify a simple cyclic executive
  - Verify: create cyclic executive, then derive corresponding table
  - Generate: create table first, then corresponding executive
- Also used as a parameter to a **dispatcher**
  - Can support preemptions
  - Starting dates and order of computation are fixed at compile time, no decision is made online by the dispatcher

- A scheduling table is its own **validation criteria**: if no task violates its deadline in the table, the implementation is safe

Static (off-line) scheduling

Scheduling table



Static (off-line) scheduling

Scheduling table:

- How to build the scheduling table ?
  - NP-hard problem
  - => for small task systems: by hand, using ILP or heuristics
  - => for large task systems: use heuristics
- When there is no inter-task dependence
  - Simulate an online scheduling policy for one hyperperiod
    - E.g. RM, DM, EDF, LLF, etc.
- When there are inter-task dependencies
  - Exhibit all jobs for one hyperperiod in your task model (including release dates and dependencies to other jobs)

!

  - Schedule ASAP while following the partial order dictated by the dependencies => List scheduling
  - In case of multiple candidate tasks, try all and choose according to a cost function (greedy algorithm)
  - Depending on the size of the task system, use backtracking

Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - RM is initially an online scheduling algorithm
  - In RM, each task is assigned a **static (fixed) priority**
  - Priorities are based on tasks **periods**
  - The smaller the period, the higher the priority
  - $T_i < T_j \Rightarrow priority(\tau_i) > priority(\tau_j)$
  - Under certain hypothesis, RM is optimal (in the sense of RT scheduling)

Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP

T_0 = 10ms
C_0 = 1ms
Pedal angle acquisition

T_1 = 10ms
C_1 = 1ms
Speed acquisition

T_2 = 20ms
C_2 = 2ms
Engine rotation acquisition

T_3 = 60ms
C_3 = 2ms
Collision detection

Engine Control Unit (ECU)
T_4 = 30ms
C_4 = 3ms

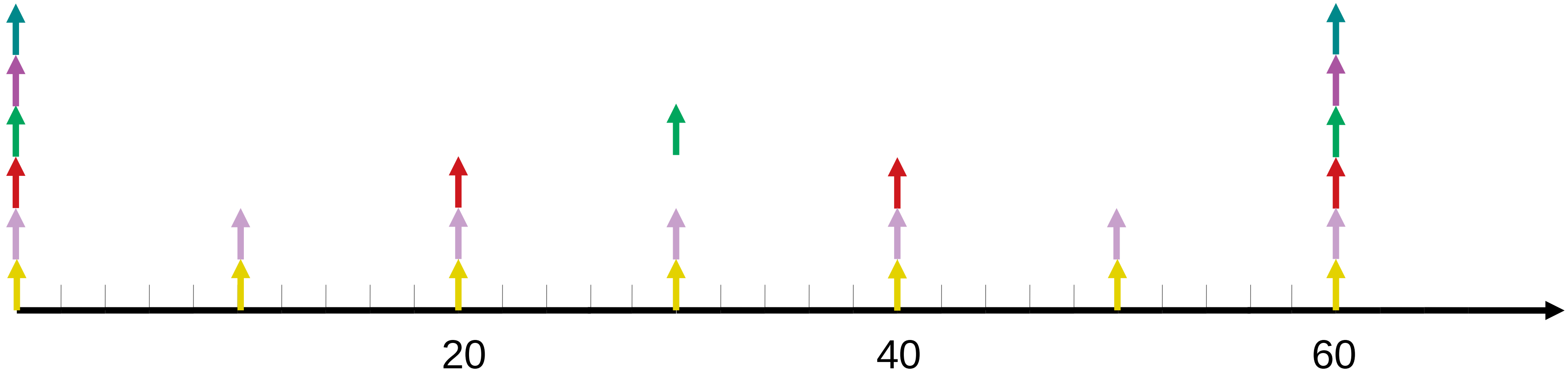
Airbag Control Unit
T_5 = 60ms
C_5 = 12ms

Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP

Start with an empty table of size hyperperiod



Static (off-line) scheduling

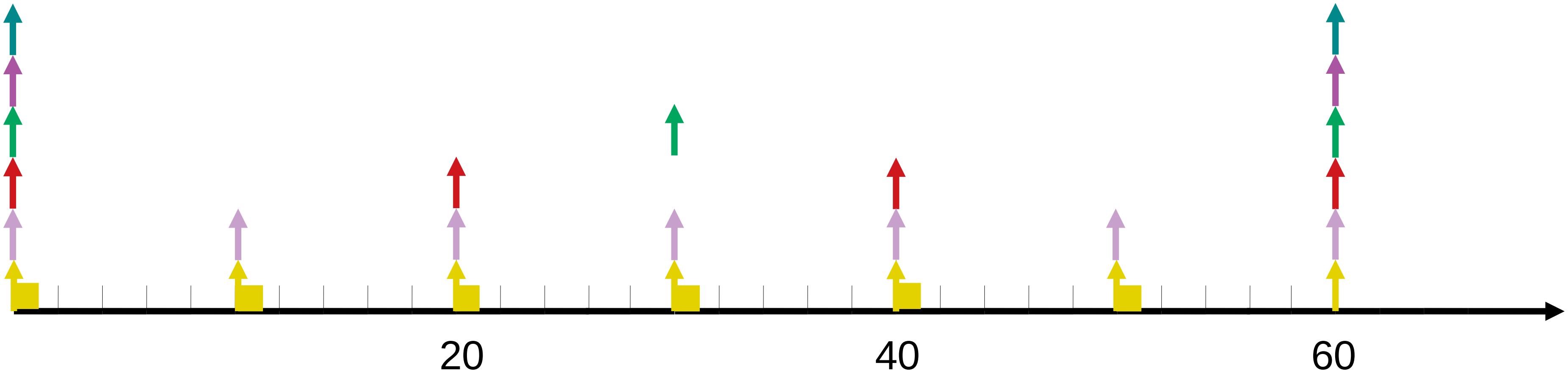
Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP

Select and schedule ASAP task with lowest period

T_0 = 10ms
C_0= 1ms

Pedal angle acquisition



Static (off-line) scheduling

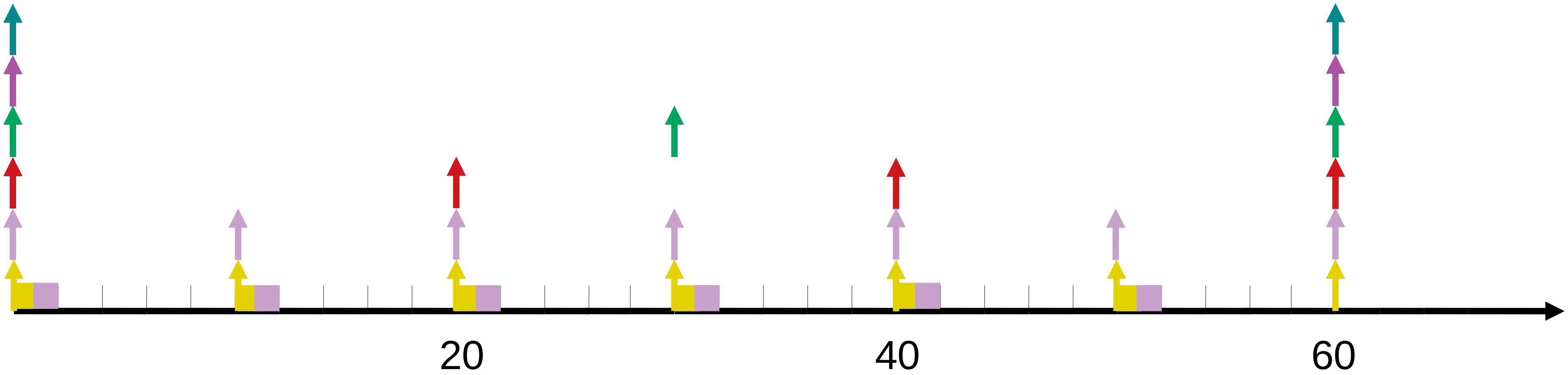
Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP

Select and schedule ASAP task with next lowest period

$T_1 = 10ms$
$C_1 = 1ms$

Speed acquisition



Static (off-line) scheduling

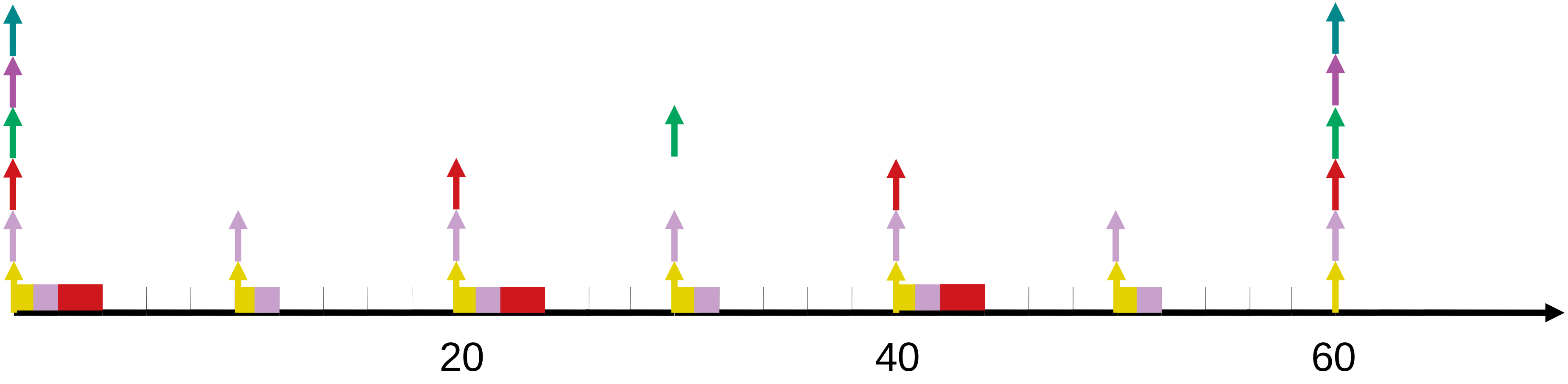
Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP

Select and schedule ASAP task with next lowest period

$T_2 = 20ms$
$C_2 = 2ms$

Engine rotation acquisition



Static (off-line) scheduling

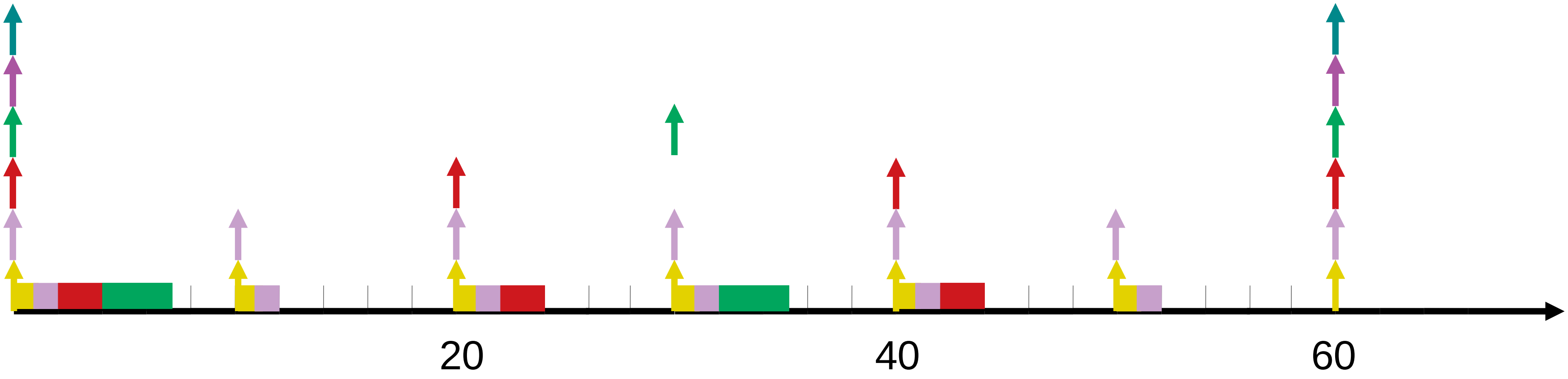
Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP

Select and schedule ASAP task with next lowest period

Engine Control Unit (ECU)

T_4 = 30ms
C_4= 3ms



Static (off-line) scheduling

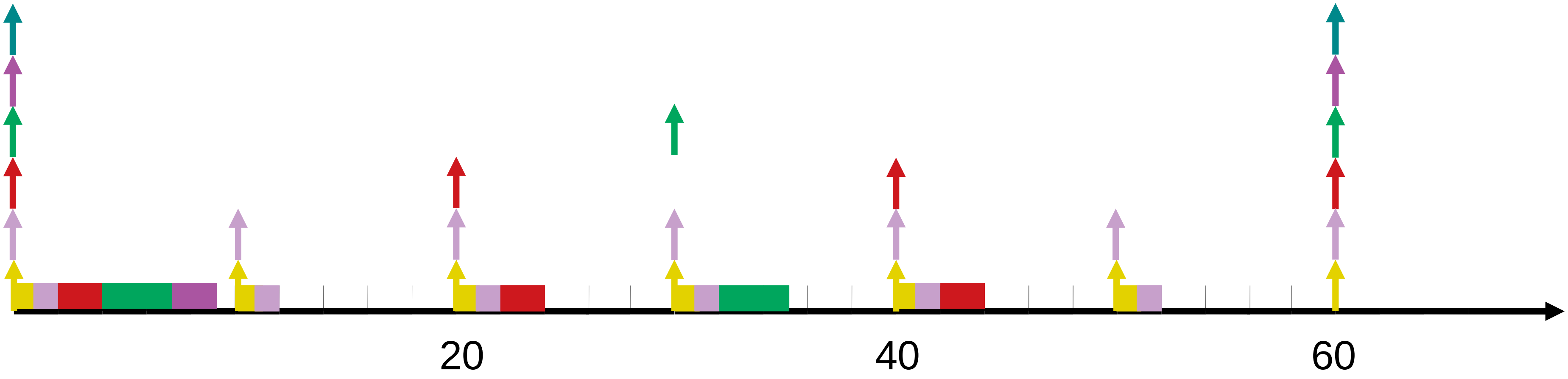
Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP

Select and schedule ASAP task with next lowest period
(I could have selected the airbag control task instead resulting in a different, yet valid, schedule)

T_3 = 60ms
C_3= 2ms

Collision detection



Static (off-line) scheduling

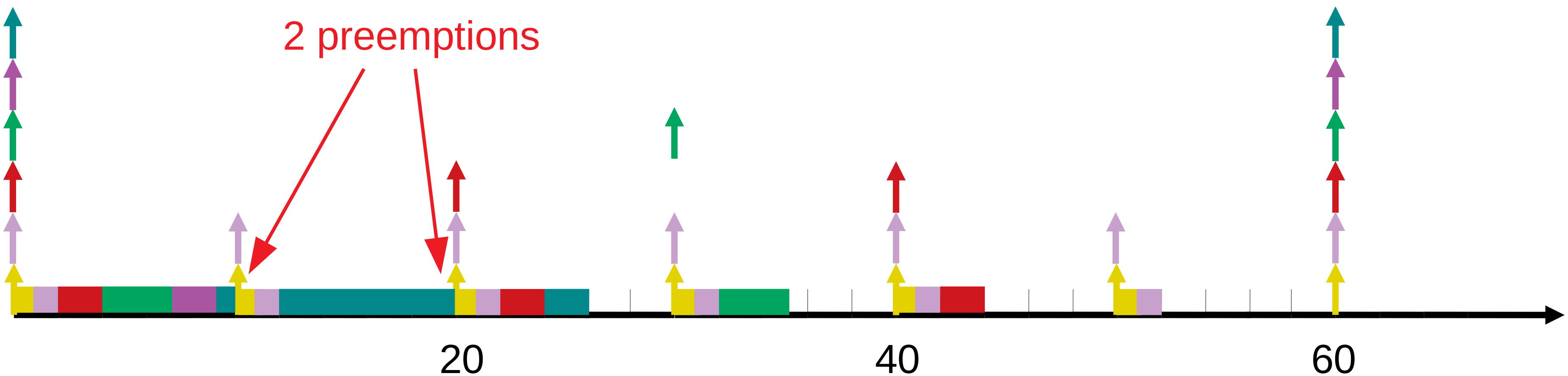
Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP

Select and schedule ASAP task with next lowest period

Airbag Control Unit

T_5 = 60ms
C_5= 12ms



Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Sufficient** schedulability criterion

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

    - $n \to +\infty$: $U \leq \ln 2 \approx 0.69$
    - Only if :
      - Preemptive
      - No shared lock (no critical section)
      - $T_i = D_i$ for all i
      - All preemption costs are accounted for in the C_i (CRPD and context switch)
    - If all periods are harmonic, $U \leq 1$ is the sufficient criterion

Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - For each task, compute response-time of the first generated job
    - If all tasks start at date 0, the worst-case processor demand happens at date 0 (a.k.a synchronous release) => if all jobs generated at date 0 finish before their deadline, then any job will finish before its deadline => the system is schedulable

Static (off-line) scheduling

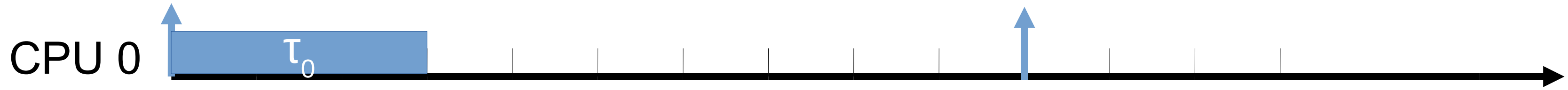
Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - How to compute the WCRT of each job ?
      - A job executes as soon as it is released, but can be delayed by tasks with higher or same priority (all of them in the worst case).
      - The WCRT is thus equal to the job's execution budget (WCET) plus the interference coming from higher than or equal priority tasks.

Static (off-line) scheduling

Scheduling table:

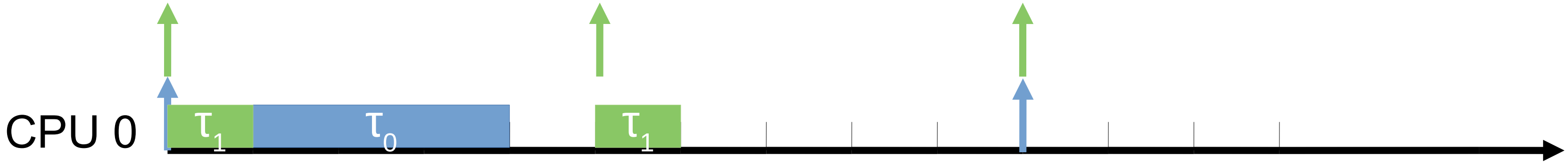
- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - τ_0 : T_0=10, C_0=3
    - In isolation, WCRT $R_0 = C_0$



Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - τ_0 : T_0=10, C_0=3
    - τ_1 : T_1 = 5, C_1=1
    - $R_1 = C_1 = 1$
    - $R_0 = C_0 + C_1 = 4$

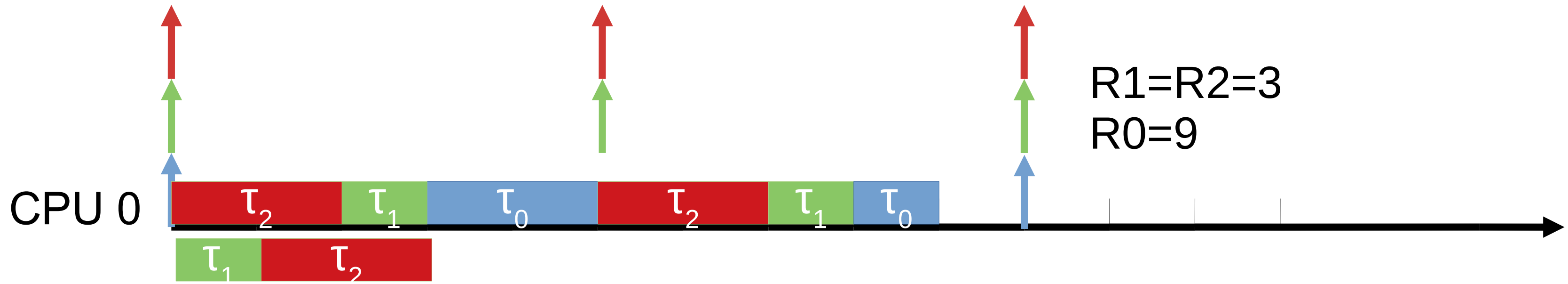


Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - τ_0 : T_0=10, C_0=3
    - τ_1 : T_1 = 5, C_1=1
    - τ_2 : T_2 = 5, C_2=2

sum(Ci/Ti)=9/10
3*(2^(1/3)-1)~0,77
Sufficient condition does not hold



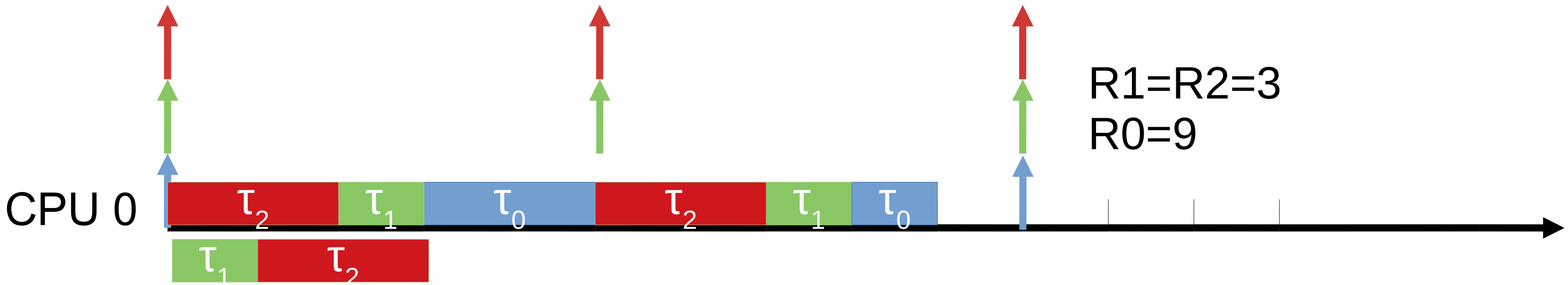
Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - $j>i \iff T_j \leq T_i$

How many jobs from τ_j arrive in an interval of duration R_i ?

$$R_i = C_i + \sum_{j=i+1}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$



Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - $j>i \iff T_j \leq T_i$

$$R_i = C_i + \sum_{j=i+1}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$

How many jobs from τ_j arrive in an interval of duration R_i ?

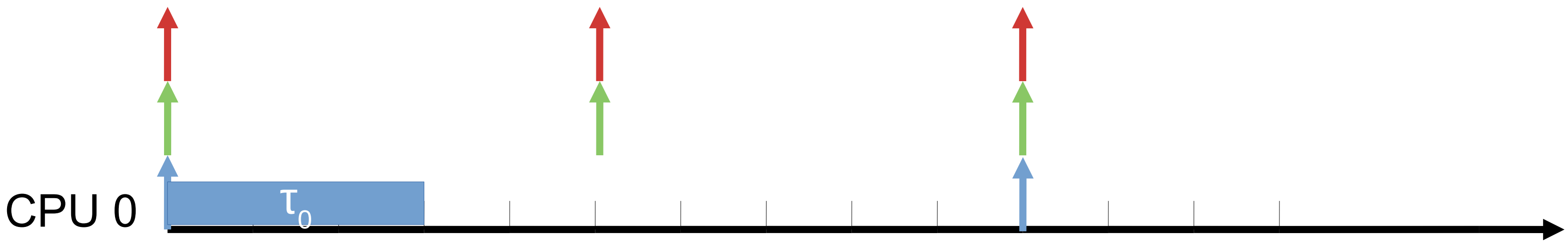
- Must be computed as a fixed-point !

Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - $j>i \iff T_j \leq T_i$
    - Fixed-point computation (step 1):
      - $R_0^1 = C_0 = 3$

$$R_i = C_i + \sum_{j=i+1}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$



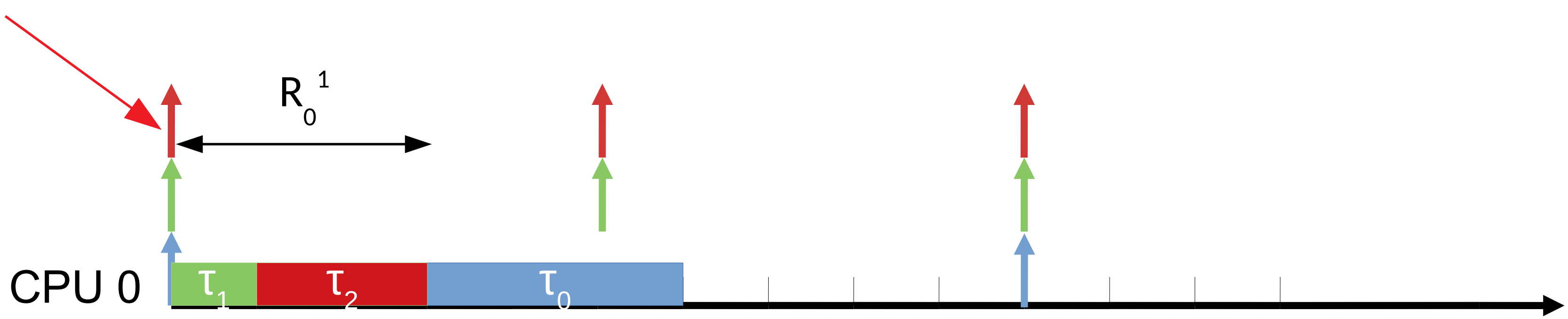
Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - $j > i \iff T_j \leq T_i$
    - Fixed-point computation (step 2):
      - $R_0^2 = C_0 + \text{ceil}(R_0^1/T_1) * C_1 + \text{ceil}(R_0^1/T_2) * C_2$
        $= 3 + \text{ceil}(3/5) * 1 + \text{ceil}(3/5) * 2 = 6$

$$R_i = C_i + \sum_{j=i+1}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$

1 arrival of τ_1 and τ_2 in R_0^1



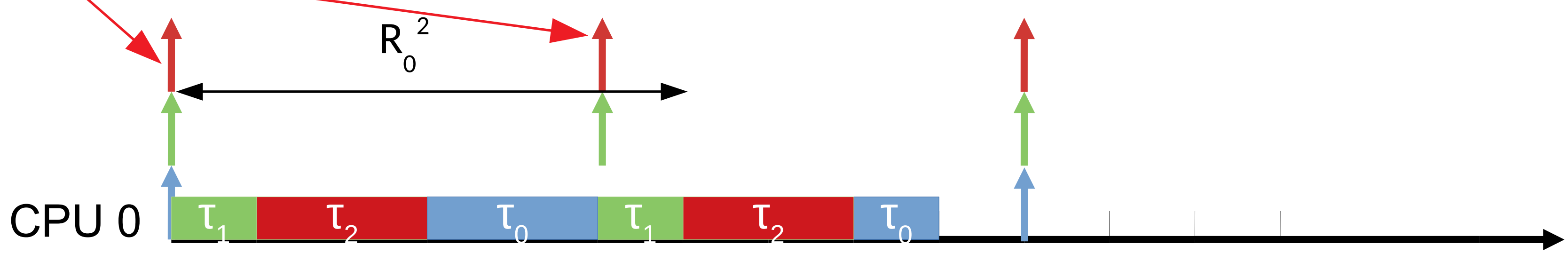
Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - $j>i \iff T_j \leq T_i$
    - Fixed-point computation (step 3):
      - $R_0^3 = C_0 + ceil(R_0^2/T_1)*C_1 + ceil(R_0^2/T_2)*C_2$
        $= 3 + ceil(6/5) * 1 + ceil(6/5)*2 = 9$

$$R_i = C_i + \sum_{j=i+1}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$

2 arrivals of τ_1 and τ_2 in R_0^2



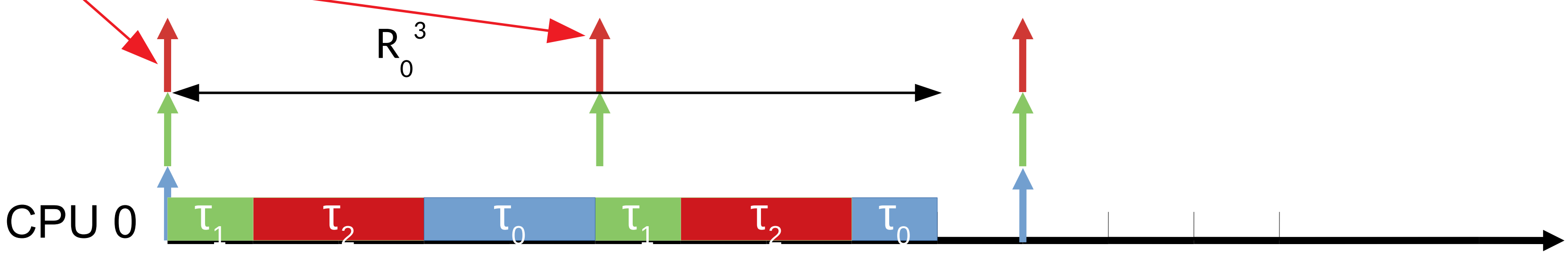
Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
    - $j>i \iff T_j \leq T_i$
    - Fixed-point computation (step 4):
      - $R_0^4 = C_0 + \text{ceil}(R_0^3/T_1)*C_1+\text{ceil}(R_0^3/T_2)*C_2$
      - $= 3 + \text{ceil}(9/5) * 1 + \text{ceil}(9/5)*2 = 9 = R_0^3$
      - We reached a fixed point => $WCRT(\tau_0)=9$

$$R_i = C_i + \sum_{j=i+1}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$

2 arrivals of τ_1 and τ_2 in R_0^3



Static (off-line) scheduling

Scheduling table:

- Preemptive rate-monotonic scheduling
  - Select tasks by increasing activation periods
  - Schedule each job of the task for the hyperperiod ASAP
  - **Exact** schedulability criterion: worst-case response-time analysis
  - Can be extended to account for blocking (critical sections in online schedulers), preemption costs, **multicore interference**

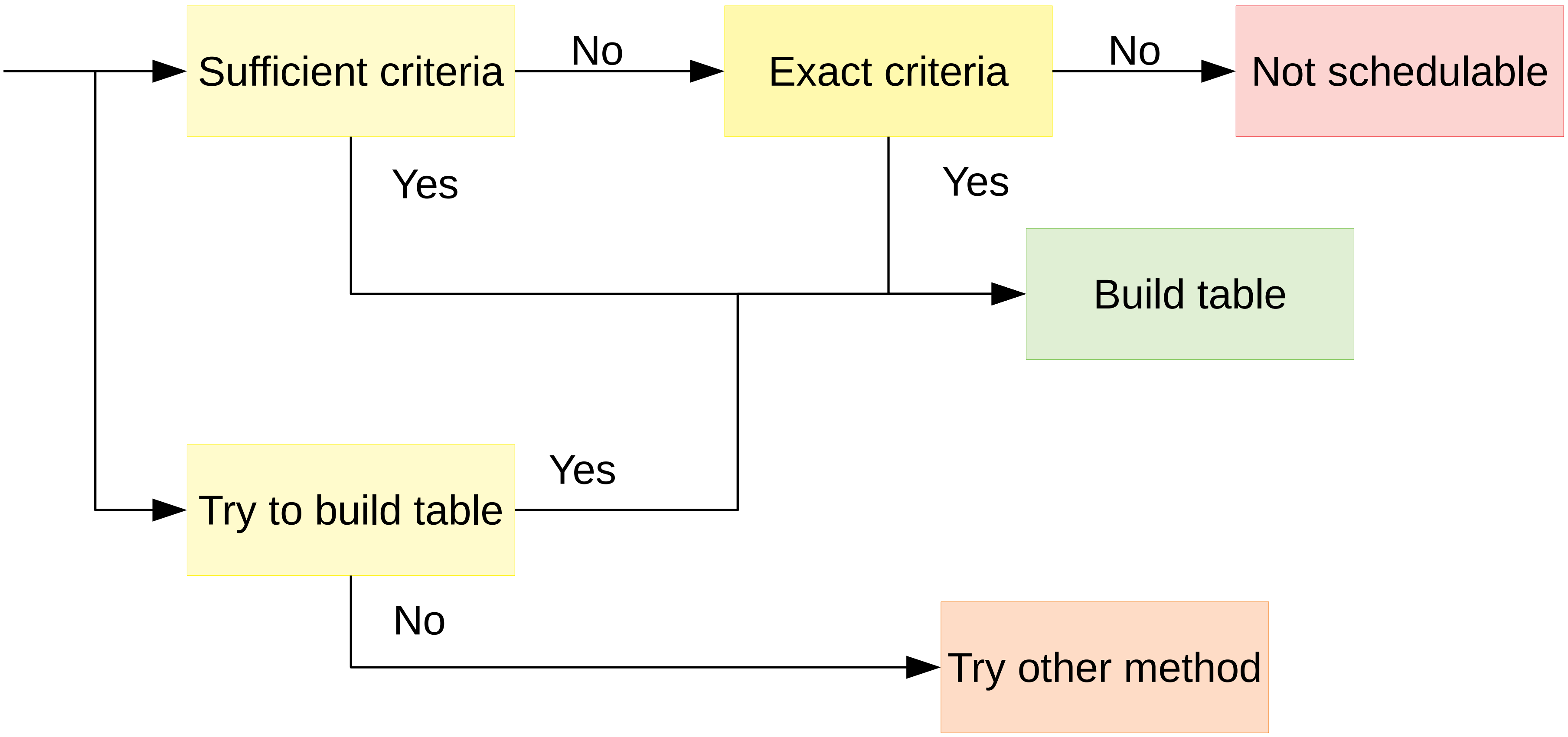
Static (off-line) scheduling

Scheduling table:

- Preemptive deadline-monotonic scheduling
  - Select tasks by increasing deadlines
  - Schedule each job of the task for the hyperperiod ASAP
  - When $D_i=T_i$, exactly the same as RM
  - Response-time analysis criterion can be obtained from the RM one :
    - Same formula
    - j>i <=> Dj ≤ Di (priorities driven by deadlines)

Static (off-line) scheduling

Scheduling table: what do we do of this ?



Static (off-line) scheduling

Exercise 1:

- Rate-monotonic scheduling
  - Is the following task system RM-schedulable ?
    - $\tau_1 : C_1 = 1, T_1 = 10$
    - $\tau_2 : C_2 = 1, T_2 = 10$
    - $\tau_3 : C_3 = 2, T_3 = 20$
    - $\tau_4 : C_4 = 2, T_4 = 60$
    - $\tau_5 : C_5 = 2, T_5 = 60$
    - $\tau_6 : C_6 = 3, T_6 = 30$
    - $\tau_7 : C_7 =4, T_7 = 60$
  - If so, draw the scheduling table for the first hyperperiod of the system, using the RM algorithm.

Static (off-line) scheduling

Exercise 2:

- Rate-monotonic scheduling
  - Is the following task system RM-schedulable ?
    - $\tau_1 : C_1 = 2, T_1 = 10$
    - $\tau_2 : C_2 = 3, T_2 = 20$
    - $\tau_3 : C_3 = 10, T_3 = 50$
    - $\tau_4 : C_4 = 13, T_4 = 100$

Static (off-line) scheduling

Exercise 3:

- Rate-monotonic scheduling
  - Is the following task system RM-schedulable ?
    - $\tau_1 : C_1 = 12, T_1 = 50$
    - $\tau_2 : C_2 = 10, T_2 = 40$
    - $\tau_3 : C_3 = 10, T_3 = 30$

Static (off-line) scheduling

Exercise 4:

- Rate-monotonic scheduling
  - Is the following task system RM-schedulable ?
    - $\tau_1 : C_1 = 1, T_1 = 4$
    - $\tau_2 : C_2 = 1, T_2 = 5$
    - $\tau_3 : C_3 = 1, T_3 = 6$
    - $\tau_4 : C_4 = 2, T_4 = 7$

Static (off-line) scheduling

Exercise 5:

- Rate-monotonic scheduling
  - Is the following task system RM-schedulable ?
    - $\tau_1 : C_1 = 1, T_1 = 5$
    - $\tau_2 : C_2 = 2, T_2 = 10$
    - $\tau_3 : C_3 = 2, T_3 = 20$
    - $\tau_4 : C_4 = 5, T_4 = 10$
  - If so, draw the scheduling table for the first hyperperiod of the system, using the RM algorithm.

Static (off-line) scheduling

Exercise 6:

- Rate-monotonic scheduling
  - Is the following task system RM-schedulable ?
    - $\tau_1 : C_1 = 3, T_1 = 7$
    - $\tau_2 : C_2 = 3, T_2 = 12$
    - $\tau_3 : C_3 = 5, T_3 = 20$

Static (off-line) scheduling

Exercise 7:

- Rate-monotonic scheduling
  - Is the following task system RM-schedulable ?
    - Process P : $C_1 = 1$, $T_1 = 3$
    - Process Q : $C_2 = 2$, $T_2 = 6$
    - Process S : $C_3 = 5$, $T_3 = 18$

Static (off-line) scheduling

Exercise 8:

- Consider three tasks P, Q and S
  - P : C = 30ms, T = 100ms
  - Q : C = 1ms, T = 5ms
  - S : C = 5ms, T = 25ms
- P is the most « important » task in the system, followed by Q and S

- What is the behavior of the scheduler if the priorities are based on the importance ?
- What is the combined utilisation U of tasks P, Q and S ?
- How should the processes be scheduled so that all deadlines are met ?
- Illustrate one of the schemes that allows these processes to be scheduled

Static (off-line) scheduling

Practical work:

- Develop a static scheduler
  - Input: multi-periodic task system described in text file
  - Policy: preemptive RM or DM
  - Output: textual description of the schedule

  - Include a mechanism to guarantee the correctness of the produced schedule

Static (off-line) scheduling

Practical work:

- Develop a static scheduler
  - Input: multi-periodic task system described in text file <= json format
  - Policy: preemptive RM or DM
  - Output: textual description of the schedule <= json format

  - Include a mechanism to guarantee the correctness of the produced schedule

  - Use the python type Task to describe the RT tasks :

```
class Task():
    def __init__(self, name, idx, period, deadline, wcet):
        self.name=name
        self.idx=idx
        self.period=period
        self.deadline=deadline
        self.wcet=wcet
```

Static (off-line) scheduling

Practical work:

- Develop a static scheduler
  - Input: multi-periodic task system described in text file <= json format
  - Policy: preemptive RM or DM
  - Output: textual description of the schedule <= json format

  - Include a mechanism to guarantee the correctness of the produced schedule

  - Use the python type Task to describe the RT tasks
  - Use the python function parse_tasks() to read a task system from a json file and obtain a python list of Tasks

Static (off-line) scheduling

Practical work:

- Develop a static scheduler
  - Input: multi-periodic task system described in text file <= json format
  - Policy: preemptive RM or DM
  - Output: textual description of the schedule <= json format

  - Include a mechanism to guarantee the correctness of the produced schedule

  - Use the python type Task to describe the RT tasks
  - Use the python function parse_tasks() to read a task system from a json file and obtain a python list of Tasks
  - Declare a python type to represent the schedule
  - Write a method to generate a schedule from a list of tasks
  - Write a method which verifies that a generated schedule is valid
  - Write a method to export a schedule to a json file