MÁSTER UNIVERSITARIO EN SISTEMAS ESPACIALES
POR LA UNIVERSIDAD POLITÉCNICA DE MADRID

POLITÉCNICA

2020.04.13

Data handling

Real-time systems

Real-time tasks and scheduling

Juan Antonio de la Puente juan.de.la.puente@upm.es

Some rights reserved. This work is licensed under a
Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License..

STRAST

Real-Time Requirements

- Some actions have to executed within a specified time interval
  - too early or too late is wrong even though the functional behaviour is correct
- Different levels of criticality
  - hard RT requirements: have to be met always
  - soft RT requirements: may fail occasionally
  - firm RT requirements: if not met the result is useless

© 2020 Grupo STRAST - UPM

Task model

T = period
D = deadline
C = processor time
R = response time

- An RTS is composed of a number of *tasks*
  - each task executes a sequence of *jobs,* which are released repeatedly according to an *activation pattern*
    - periodic, sporadic, aperiodic, random, etc.
  - each job has a deadline *D* relative to its release time
  - each consumes a certain amount *C* of processor time
- The time when the job completes, relative to its release is the response time *R* of the job
- The main real-time requirement is that for all jobs of every task $R \leq D$
  - $C \leq R$ is implied

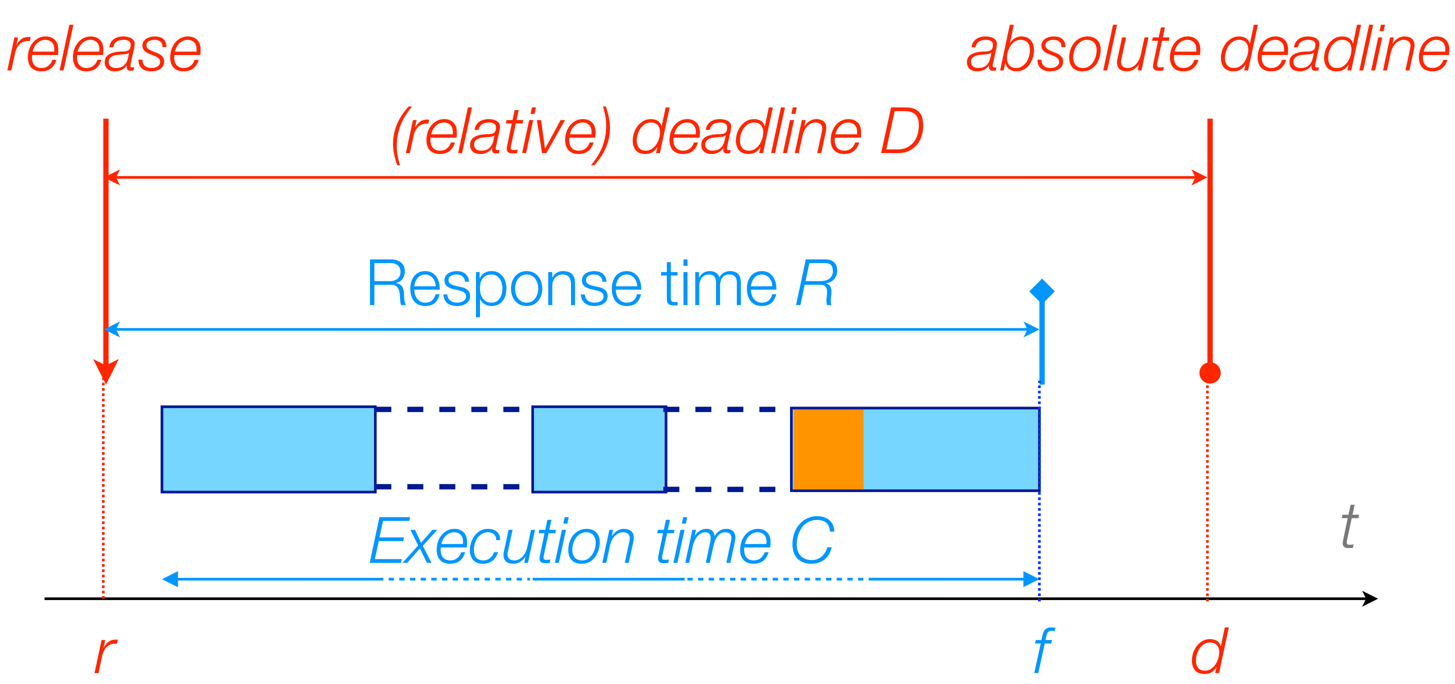
R (RESPONSE) <= D (DEADLINE)

C (processor time) <= R (RESPONSE)

© 2020 Grupo STRAST - UPM

Feasible interval

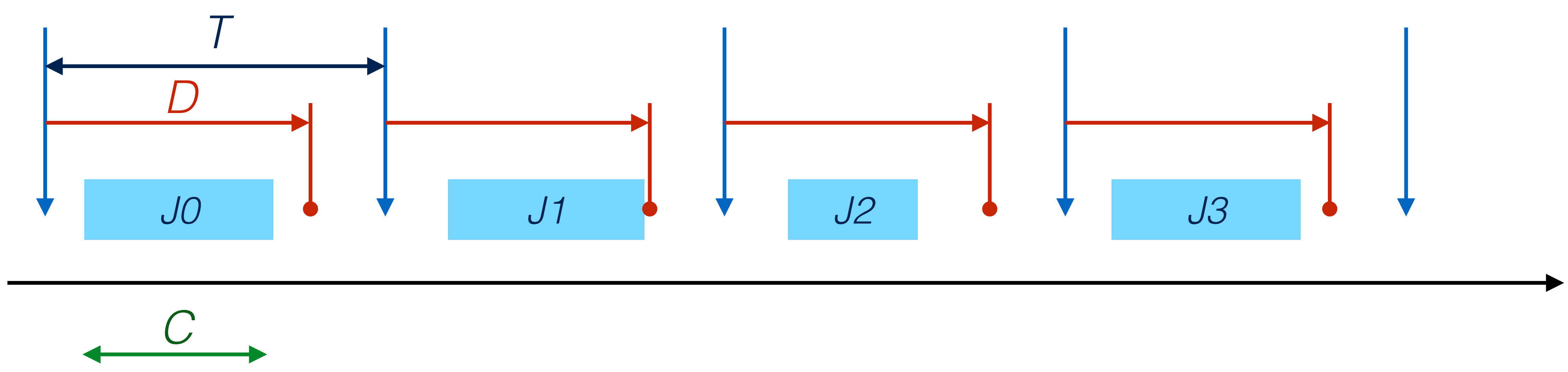
- A *job* has to be executed between its release time and its deadline
- It uses processor time, possibly split into several execution intervals



Periodic tasks

T = period
D = deadline
C = processor time
R = response time

- The jobs of a periodic task are released at regular intervals, with a period *T*
- A periodic task is characterised by (T, C, D)
  - T : period
  - C : computation time
  - D : deadline

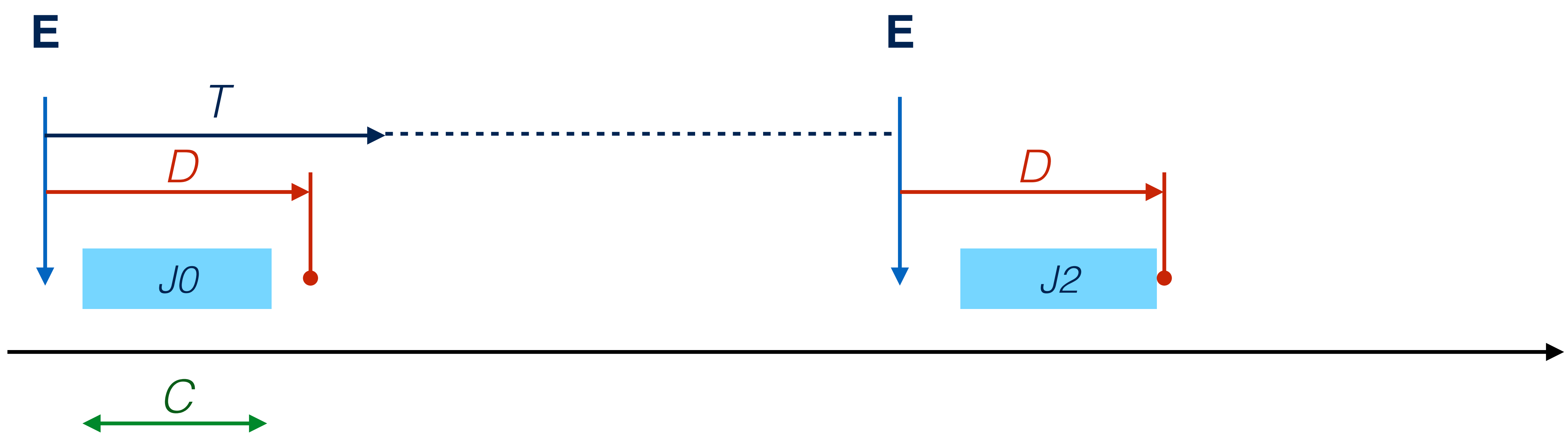


© 2020 Grupo STRAST - UPM

Sporadic tasks

- The jobs of a periodic task are released whenever some event E occurs
  - a minimum separation *T* is usually imposed
- A sporadic task is characterised by (T, C, D)

T = period
D = deadline
C = processor time
R = response time
E = event



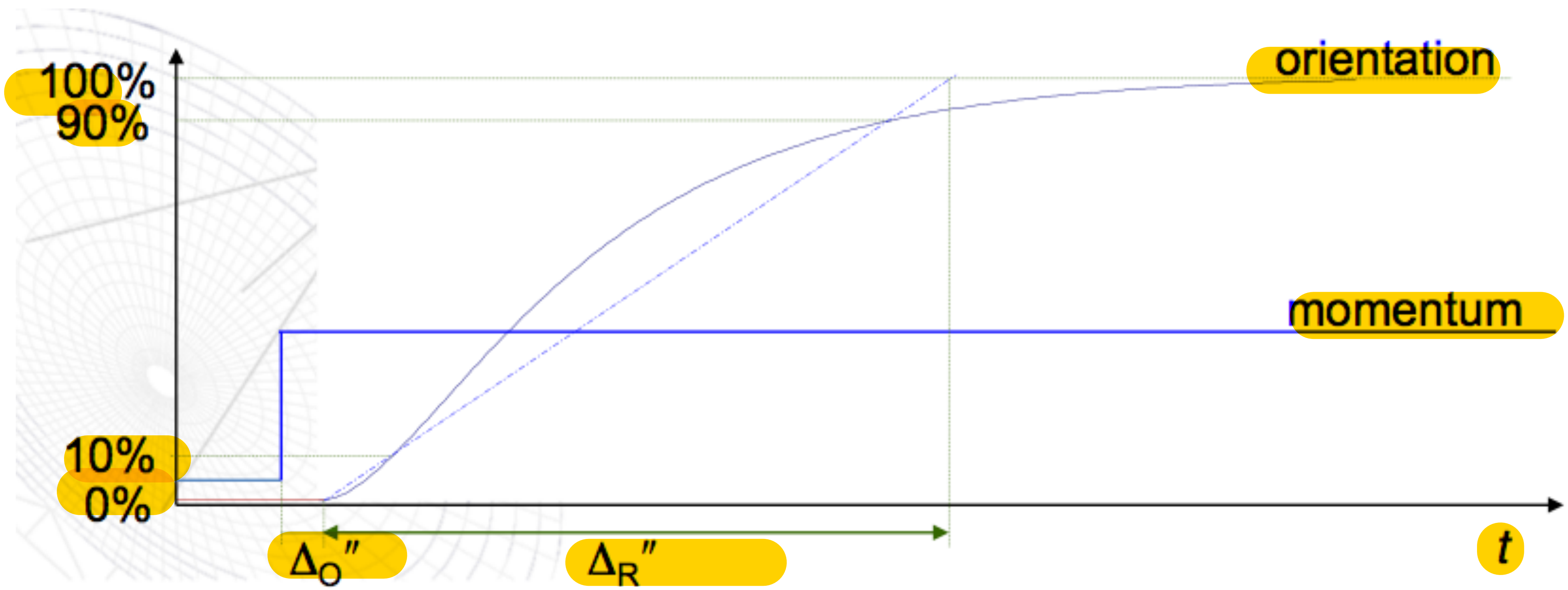
© 2020 Grupo STRAST - UPM

Aperiodic tasks

- Aperiodic tasks are event-driven, but have no real-time requirements
  - no minimal separation between event occurrences
  - no deadline
    - but there is often the requirement that jobs must complete as soon as possible

System dynamics and real-time

- Real-time requirements are derived from the system dynamic characteristics
- Example: attitude control
  - a change in a reaction wheel momentum makes the attitude angles change
  - but the change is not instantaneous



© 2020 Grupo STRAST - UPM

Sampling rate

sampling rate >= 2*bandwidth
Delta(t) < Delta(R) / 10

- Analog variables are sampled when reading them into a computer
- The sampling rate must be at least twice the bandwidth
  - Nyquist-Shannon sampling theorem
    - but faster sampling rates are most common
- Heuristic rules are often used
  - a common rule is $\Delta t < \Delta R/10$
- Control functions are applied on a periodic basis
  - the period is derived from the sampling rate used in the control algorithm

© 2020 Grupo STRAST - UPM

Deadlines

- Deadlines are also derived from system dynamics
- A function must be completed in time for the system state to evolve as specified
- "Too late" is often not admissible
  - the system may enter an undesirable state or become uncontrollable
  - telecommand / telemetry messages may be lost
  - a ground station may lose track of a satellite

© 2020 Grupo STRAST - UPM

Sample requirements

- Nominal mode attitude control
- Keep nominal attitude
  - adjust roll, pitch and yaw angles by acting on reaction wheels
    - every 200 ms
  - calibrate gyro sensors with IR sensor
    - every 1s for 15 min, twice a day
  - gyro sensors deliver data every 100 ms
- Other tasks
  - dump momentum from reaction wheels
    - whenever necessary
  - send TM data when requested by TC
    - not more often than 60 ms
  - respond to TC
    - within 190 ms

© 2020 Grupo STRAST - UPM

Modes of operation

- Many real-time systems have different modes of operation
  - for example, ground, launch, orbit acquisition, nominal
- The set of tasks that are executed, as well as their temporal requirements, may vary from one mode to another
- Mode changes are triggered by specific events
  - e.g. getting to nominal orbit
- They must be carried out in such a way that the temporal integrity of the system is not compromised
  - no deadline misses, no inadmissible release delays

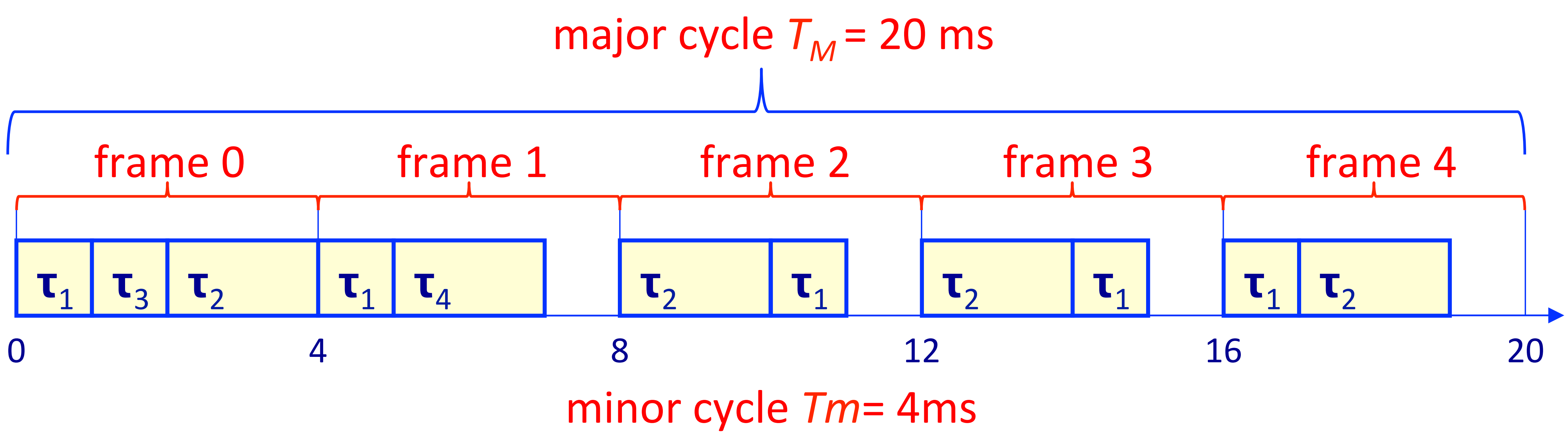
© 2020 Grupo STRAST - UPM

Time-driven implementation

- Let us consider a simple cyclic task model
  - static set of tasks
  - all tasks are periodic
- A static execution scheme can be devised for this simple model
  - the overall system behaviour is periodic
    - all actions are repeated with a hyperperiod $H = \text{mcm } T_i$
    - time is divided into (major) cycles with a duration equal to a hyperperiod
    - major cycles are divided into frames (minor cycles) of equal duration
    - individual jobs are executed within frames
  - the above scheme can be implemented with a simple program called a cyclic executive

Example

Task	T	C	D	f0	f1	f2	f3	f4
τ_1	4	1	4	x	x	x	x	x
τ_2	5	2	5	x		x	x	x
τ_3	20	1	20	x				
τ_4	20	2	20		x			



© 2020 Grupo STRAST - UPM

Cyclic executive

- Each frame is synchronised with a timer interrupt

```
procedure Cyclic_Executive is
   type Frame is mod 5;
   f : Frame := 0;
begin
   loop
      Wait_Timer_Interrupt; -- low-level handler
      case f is
         when 0 => T1; T3; T2;
         when 1 => T1; T4;
         when 2 => T2; T1;
         when 3 => T2; T1;
         when 4 => T1; T2;
      end case;
      f := f + 1;
   end loop;
end Cyclic_Executive;
```

Assessment

- Advantages
  - simple, robust, easy to verify
  - deterministic time behaviour
- Disadvantages
  - static schedule may be complex to build (NP-hard)
    - easier if periods are harmonic
    - long periods are difficult to handle
    - long jobs may have to be split into shorter segments
  - too rigid, difficult to maintain
    - small changes may force the whole plan to be re-computed
  - sporadic tasks are not easily incorporated into a cyclic plan
    - polling servers are commonly used

© 2020 Grupo STRAST - UPM

Event-driven implementation

- Tasks are implemented as threads managed by a multiprogramming kernel
  - e.g. Ada tasks
- Threads await an activation event, then execute their job and suspend until the next activation event
  - may be a clock-driven synchronous event (periodic tasks) or an asynchronous event (aperiodic/sporadic tasks)
  - see examples in chapter on clocks and timers
- Deadline guarantees are provided by scheduling and static analysis
  - next chapter

© 2020 Grupo STRAST - UPM

Example: periodic task pattern

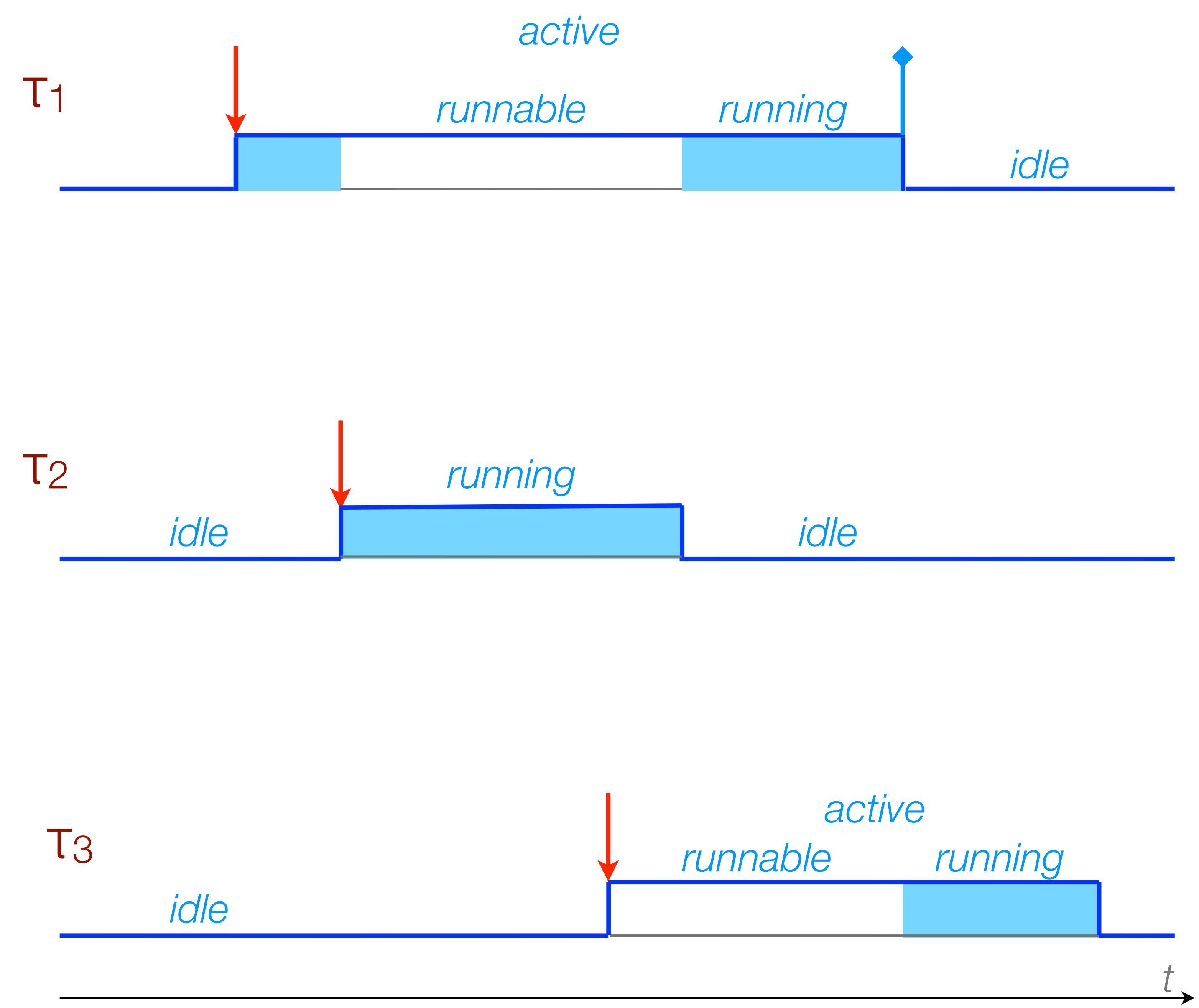
```
task type Periodic (Period : Time_Span);

task body Periodic is
   Next_Time : Time := Clock;
begin
   loop
      delay until Next_Time;  -- activation event
      ... -- periodic job
      Next_Time := Next_Time + Period;
   end loop;
end Periodic;
```


© 2020 Grupo STRAST - UPM

Processor scheduling

- If the number of concurrent tasks is greater than the number of processors, a decision must be made about which task(s) to execute at a given time
- A scheduling algorithm is used to this purpose by the OS/RTS



© 2020 Grupo STRAST - UPM

Some common scheduling methods

- Round-robin scheduling
  - each tasks runs for the duration of a time-slice and then gives way to the next task in a circular queue
  - common in general-purpose OS
    - good for fairness
  - poor performance in RTS
- Priority scheduling
  - each task has a priority
    - a measure of its importance or urgency
  - the highest-priority ready task is dispatched for execution when a processor is available
  - can provide predictable behaviour in real-time systems

© 2020 Grupo STRAST - UPM

Priority scheduling

- Static priorities
  - task priorities do not change over time
  - better predictability

- Dynamic priorities
  - task priorities may change
  - better processor utilisation

- Pre-emptive vs non-pre-emptive scheduling
  - pre-emptive scheduling: if a task with a higher priority the the currently running task becomes ready, the latter is pre-empted from the processor in order to run the highest-priority task
  - non-preemptive scheduling: the running task is allowed to run until it suspends itself, even if a higher priority task has become ready

© 2020 Grupo STRAST - UPM

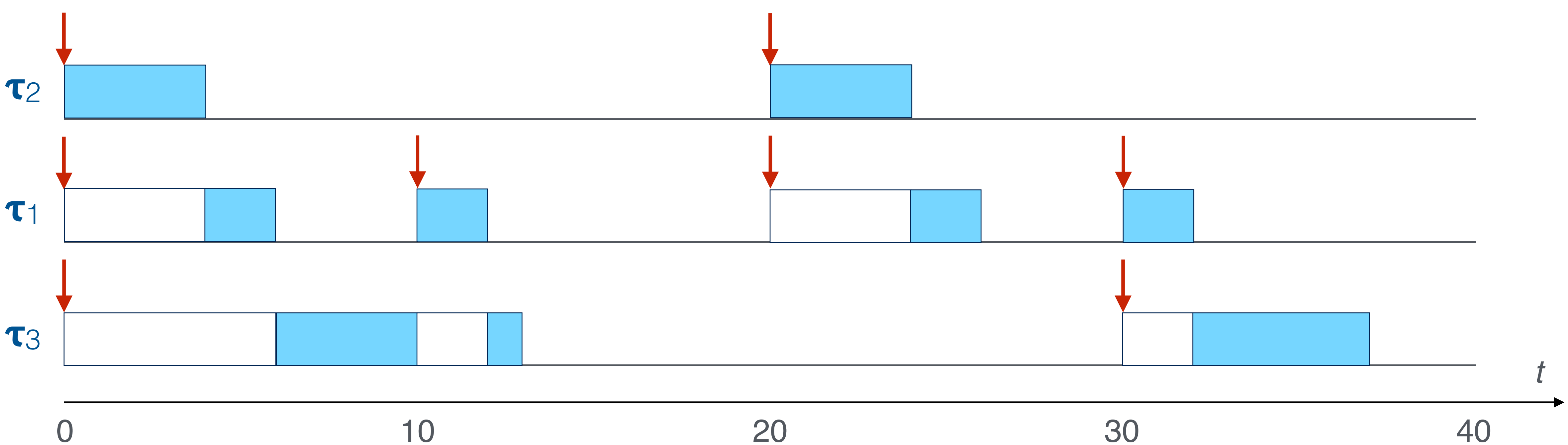
Fixed-priority pre-emptive scheduling

- FPPS is a common scheduling method for real-time systems
  - task priorities are fixed
    - except perhaps on mode changes and other discrete events
  - the highest priority active task is always running
- Problem: how to assign priorities to tasks
  - Deadline-monotonic scheduling (DMS) is optimal
    - tasks with shorter deadlines have higher priorities
    - if $D = T$ for all tasks, it is the same as rate-monotonic scheduling (RMS)

© 2020 Grupo STRAST - UPM

Example

Task	T	C	D	P
τ_1	10	2	10	2
τ_2	20	4	8	3
τ_3	30	5	20	1



© 2020 Grupo STRAST - UPM

Task scheduling in Ada

- Tasks have priorities

```
subtype Any_Priority is Integer range ...;
subtype Priority is Any_Priority
  range Any_Priority'First .. value;
subtype Interrupt_Priority is Any_Priority
  range Priority'Last + 1 .. Any_Priority'Last;
```

- The basic priority of tasks is defined by

```
task type T
   with  Priority => P
is ...
```

- It can be a task attribute

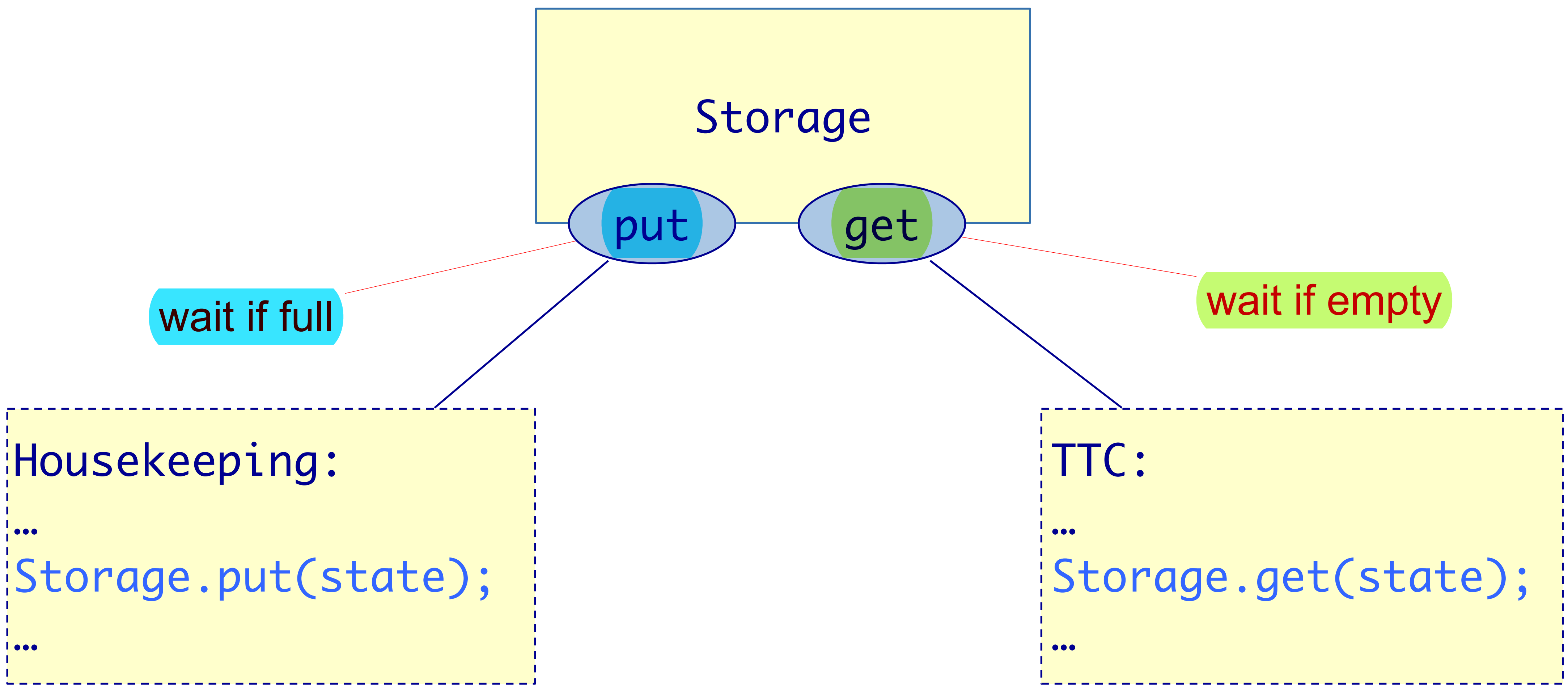
```
task type T (P : System.Priority)
   with Priority => P
is ...

Housekeeping : T(25);
```


© 2020 Grupo STRAST - UPM

Shared data

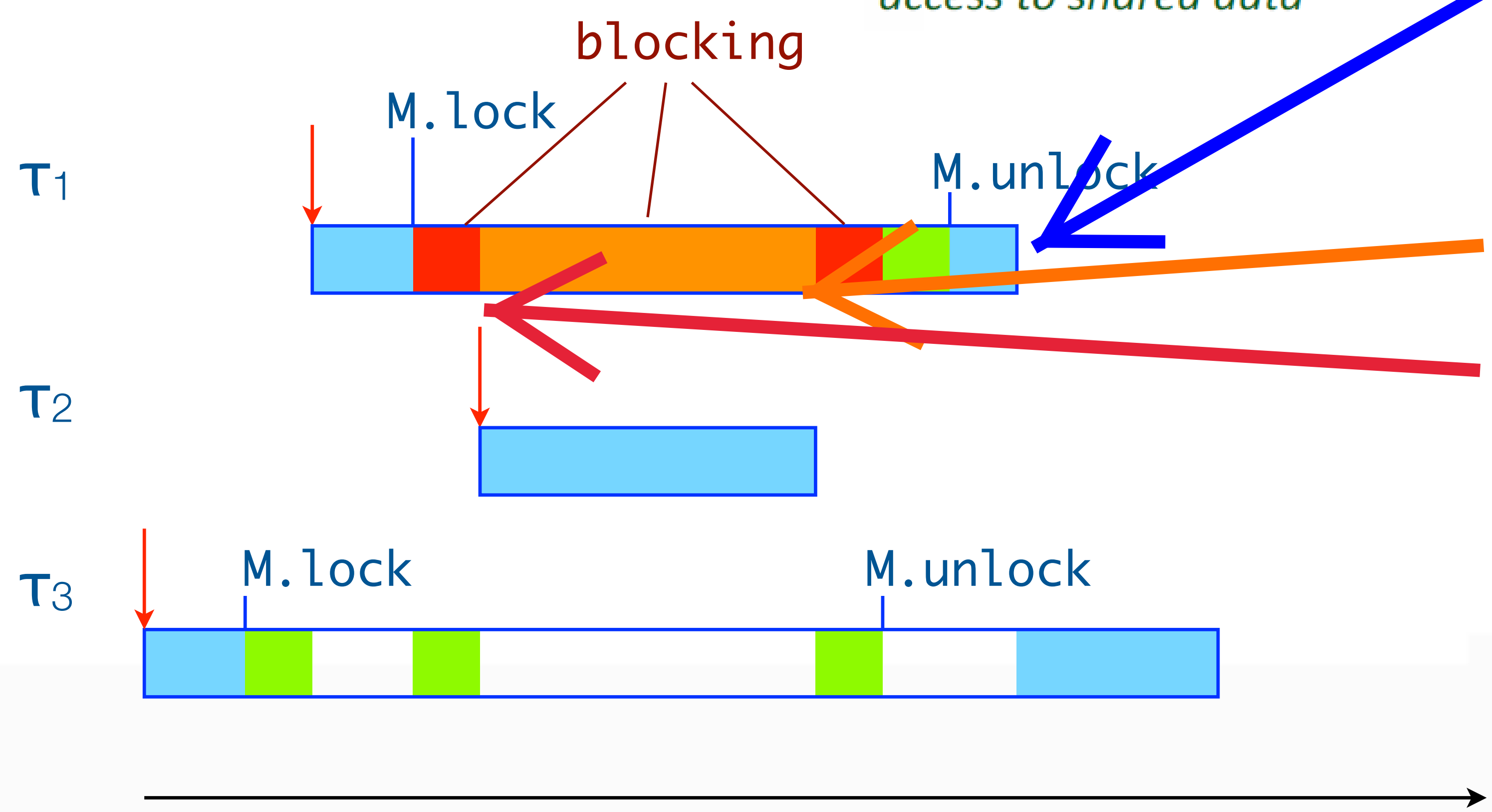
- When tasks access a shared object in mutual exclusion priority inversion may arise



© 2020 Grupo STRAST - UPM

Priority inversion

- When two tasks access shared data in mutual exclusion, priority inversion may arise
  - a high-priority task is delayed by a low-priority task
  - Example



En los azules el intervalo de la tarea en la que se esta ejecutando.

El naranja, el tiempo de bloqueo en la tarea 1 por la tarea 2

El rojo el tiempo de bloqueo en la tarea 1 por la tarea 3

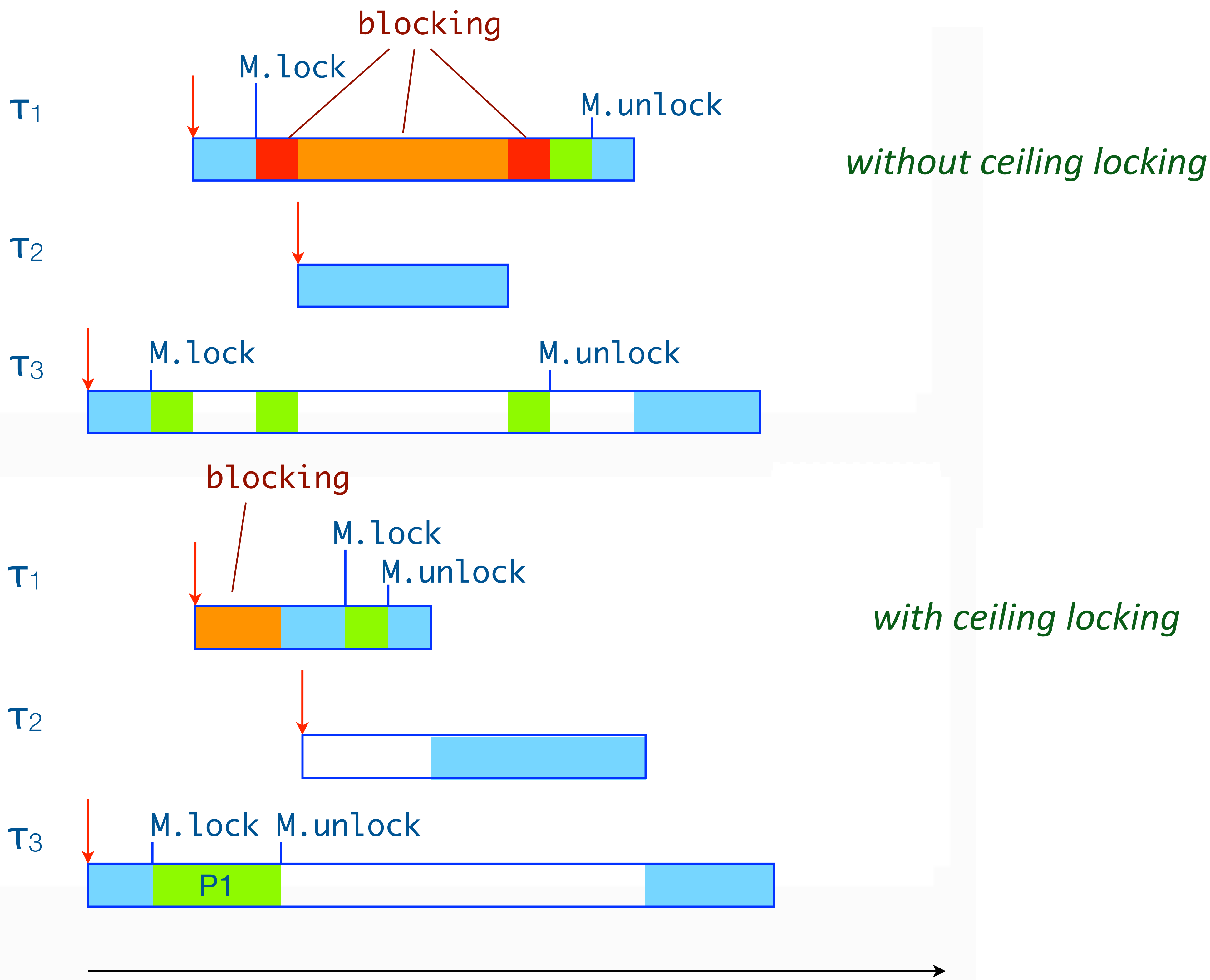
Estan todos en accediendo al mismo recurso. Se activa la inversion de prioridad a la tarea 1 pasa a ultimo plano

Access protocols

- Priority inversion cannot be totally eliminated
- But it can be bounded by using an appropriate access protocol for shared objects
- The immediate ceiling priority protocol (ICPP) is often used
  - each shared resource has a ceiling priority
    - the highest of the tasks that use it
  - a task trying to get exclusive access to a shared object immediately inherits its ceiling priority
  - the blocking a task may incur is bounded by the longest operation on the shared object
  - also known as *ceiling locking* or *highest locker protocol*

© 2020 Grupo STRAST - UPM

Example



© 2020 Grupo STRAST - UPM

Ceiling priority

- The ceiling priority of protected objects can be specified with an annotation

```
protected type Manager (P : System.Priority)
   with Priority => P
is
   ...
private
   ...
end Manager;

System_Manager : Manager(30);
```

Scheduling methods in Ada

- Scheduling is defined for the whole program
  - in a configuration file
  - FPPS is specified by

```
pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
```

  - ICPP for shared objects is specified by

```
pragma Locking_Policy (Ceiling_Locking);
```

- Other scheduling methods are also available
  - dynamic priorities, earliest-deadline first, round-robin, etc.

© 2020 Grupo STRAST - UPM