Application Methodology

Written by GT-DISC

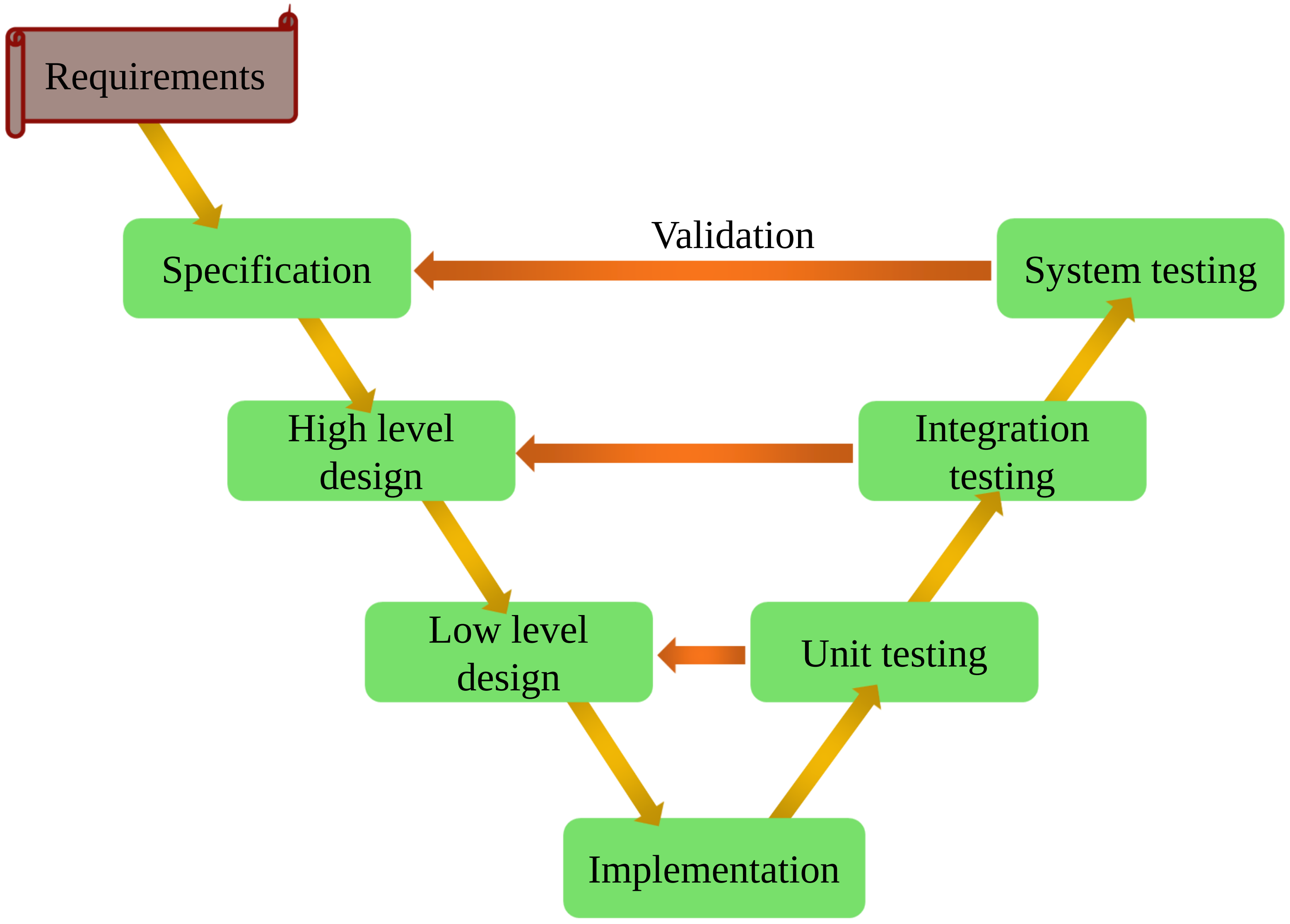
Institut Supérieur de l'Aéronautique et de l'Espace SUPAERO

Table of Contents

1. Introduction to Methodology

- *Life cycle*
- *Functional Specification*
- *SysML reminders*
- *Architecture definition*

Life cycle


Requirements
Specification
Validation
System testing
High level design
Integration testing
Low level design
Unit testing
Implementation

Functional Specification

» From the user requirements, we build a model to :

- Identify and describe the functions.
- Identify and describe the relations between the functions.
- Describe the behavior of the system .
- Describe the temporal constraints of functions execution.

Using

- **SysML = Systems Modeling Language**

SysML Reminders

→**Requirement diagram** :

- *Describes what the system should do*

→**State machine diagram :**

- *Shows the different states of the system*

→**Activity diagram :**

- *Shows the sequence of actions when the system operates*

→**Use case diagram :**

- *Shows the interactions between the actors and the system itself*

Architecture Definition

Function	**Activation**	**Dead line**	**WCET (worst case execution time)**
Function « A »	*Activation for function « A »*	*Dead line For « A »*	*WCET for « A »*
Function « B »	*Activation for function « B »*	*Dead line For « B »*	*WCET for « B »*
Function « C »	*Activation for function « C »*	*Dead line For « C»*	*WCET for « C »*

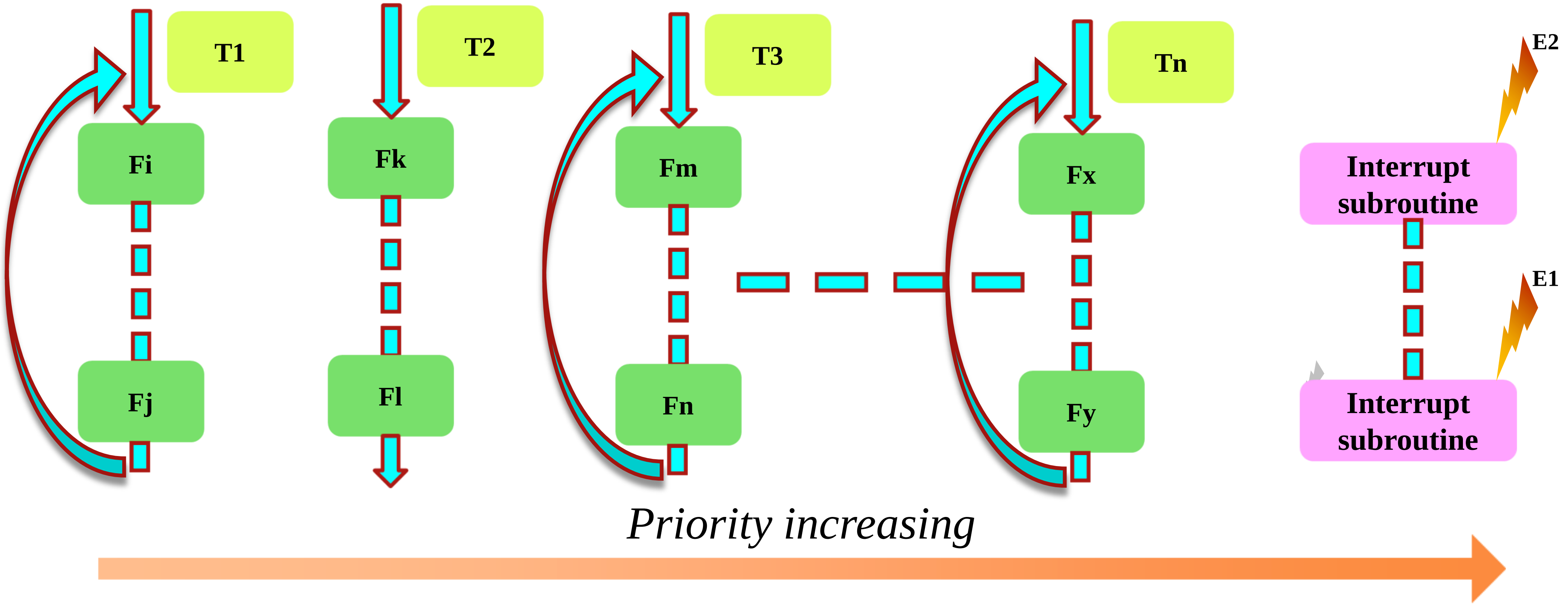
- **The analysis of these datas, the choice with or without preemption can be done :**
  - If the execution time of a function is higher than a deadline, a solution with preemption must be used.
  - If the sum of the execution times of the concurrent functions is lower than the shortest deadline, we can easily implement a solution without preemption.

2. Real-Time Application Methodology

- » *Asynchronous approach*
- » *Tasks definition*
- » *Communication between tasks*
- » *Synchronization between tasks*
- » *Periodic activation*
- » *Software architecture plan*
- » *Application Design Steps*

Asynchronous approach (preemptive)

» Execution model :



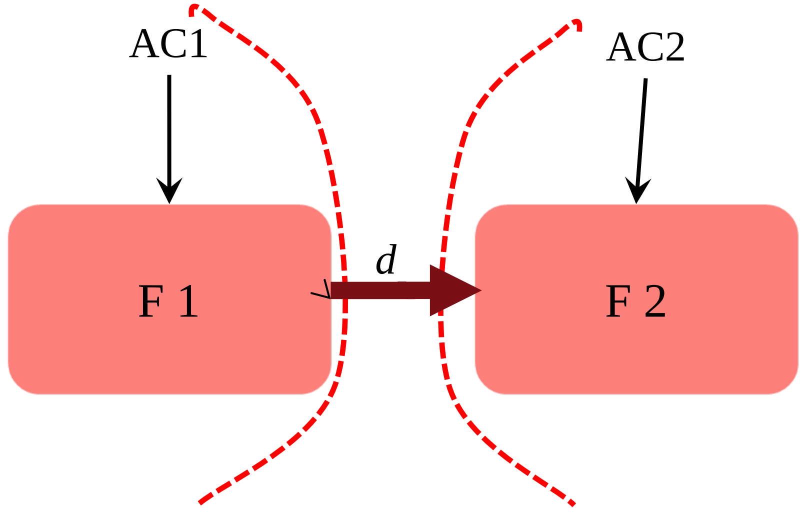
Pseudo parallelism managed by the real time operating system

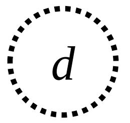
Task definition

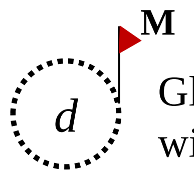
» Execution model :

Question :	**Answer is Yes**
If several functions has periodic activation **and** the periods are multiple **and** the execution duration allowed	» These functions can be grouped into the same task
If 2 or more functions are executed sequentially **and** the execution duration allowed	
If a function has an asynchronous activation condition	» This function has to be in a specific task

Communication between tasks (1)



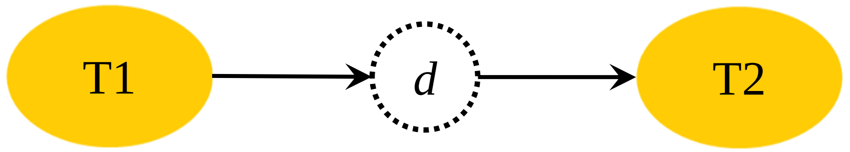

Global variable


Global variable
with mutex protection

Message queue

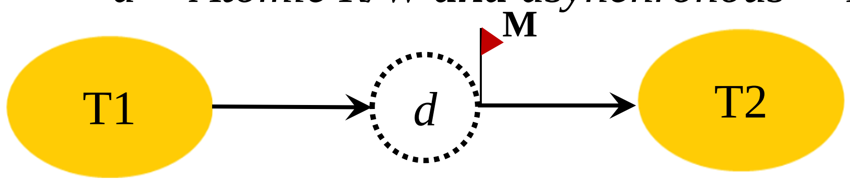
Last occurrence of *d* :

- *d = Atomic R/W*
- *d > Atomic R/W* ***and*** *synchronous* $^{(*1)}$ *R/W*



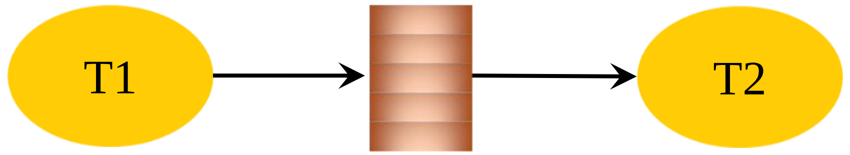
$^{(*1)}$ ***Synchronous :***
Sequential writing and reading

- *d > Atomic R/W* ***and*** *asynchronous*$^{(*2)}$ *R/W*



$^{(*2)}$ ***Asynchronous :***
A writing may be interrupted by a reading

Several occurrences of *d* :



Size = P_{AC2} / P_{AC1}

Communication between tasks (2)

» Definition of communication mechanisms between task.

Communication link's name	Activating data	Last or several occurences	Asynchonous R/W (if dataSize >1 byte)	Communication mechanism
Data #1	No	Last	No	Global variable
Datat #2	No	Last	Yes	Global variable with mutex
Data #3	Yes	X	X	Message queue
Data #3	X	Several	X	Message queue

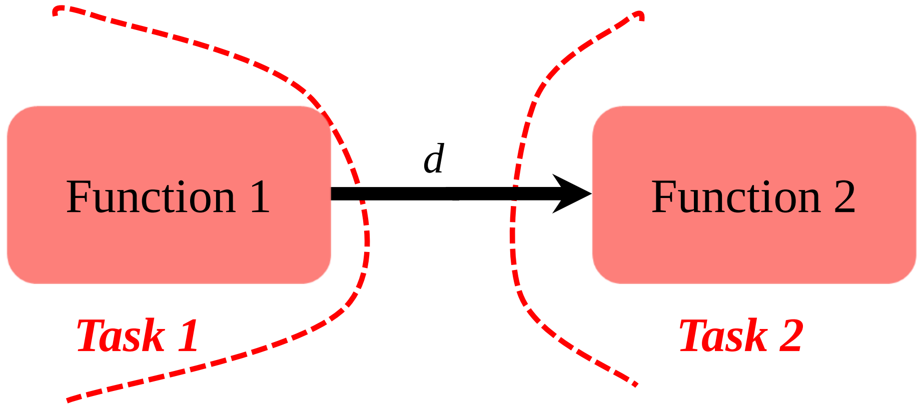
Functionnal needs

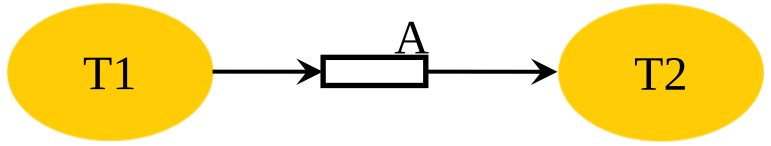
Solution

Synchronization between tasks (1)

» Unique activation conditions :

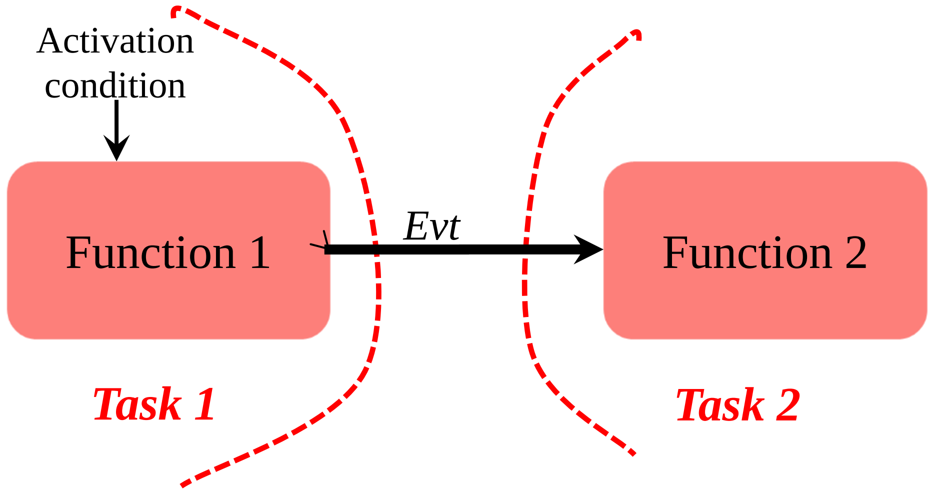
- **Activating data:**

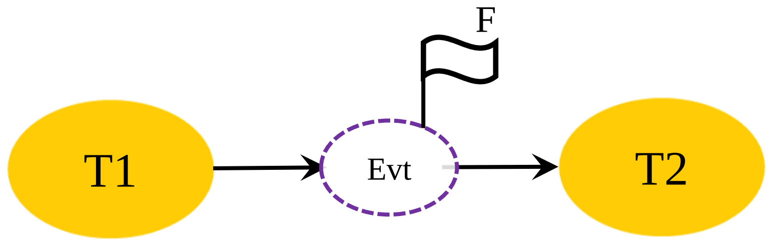


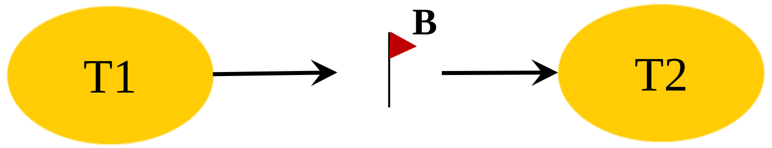


- **Tested event :**

- **Activating event :**

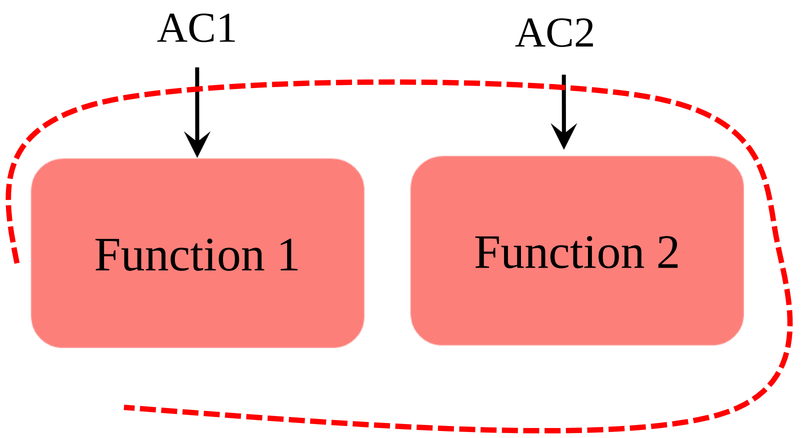




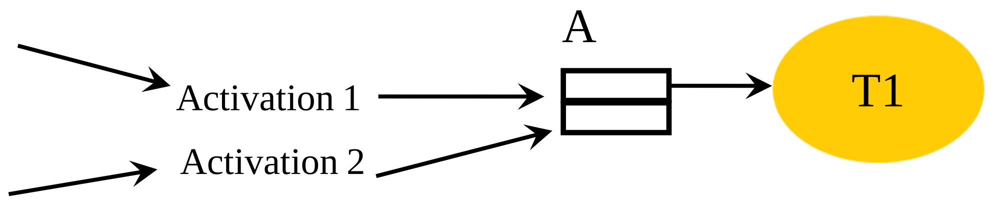


Synchronization between tasks (2)

- **Multiple activation conditions**

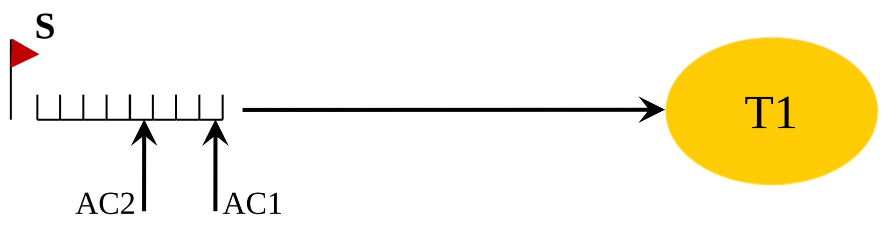


- » Activating message queue



OR

- » Signals



Synchronization between tasks (3)

» Definition of Synchronization mechanisms between task

Synchronization link's name	**Activating data**	**One Several Activation conditions**	**Tested or waited event**	**Synchronization mechanism**
Event #1	Yes	One	waited	Activation message queue (1 elt)
Event #2	No	One	Tested	Flag
Event #3	No	One	Waited	Binary semaphore
Event #4	No	Several	Waited	Activating message queue
Event #5	No	X	X	Signals
Event #6	X	X	X	Message queue ⚠

Functionnal needs **Solution**

⚠ A message queue can be used for any synchronization, but that's not the best solution.

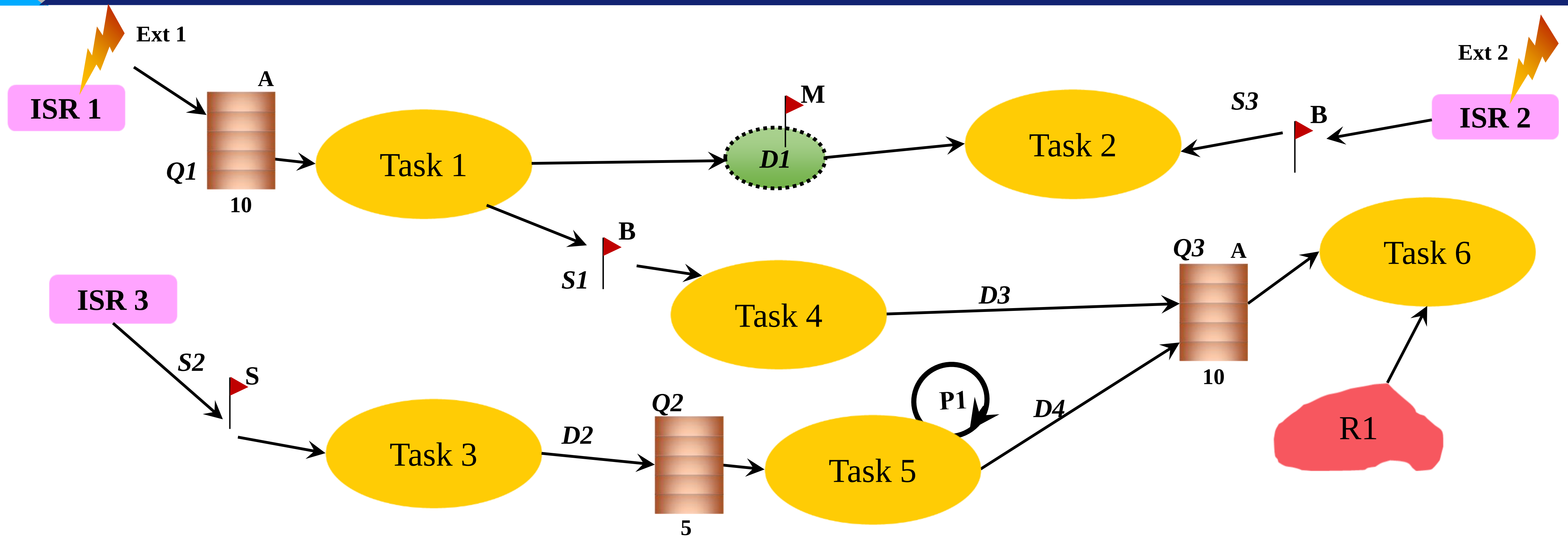
Periodic activation of tasks

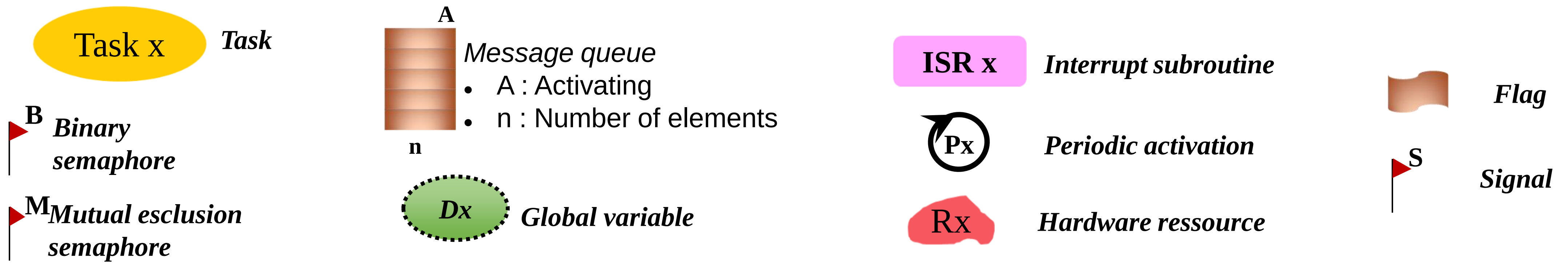
» Attitude modification

rt_task_set_periodic : *Define The period in nano-second*

rt_task_wait_period : *Wait for the periodic activation,*

Software architecture plan





Application Design Steps (1)

» Determination of the relative priority between tasks

- **Rate monotonic**

if

- The tasks are periodic
- The tasks are independant
- The CPU load factor is less than 69% ($n(2^{(1/n)}-1)$)

then

- The priorities are inversely proportional to the activation period.

(short period □ high priority)

Application Design Steps (2)

» High level tasks algorithm example

Task T1

```
Begin
    While (true)

        Wait for a message in the queue
        Function F1
        V ( Binary semaphore)  S1
        Function F2
        Acquire Mutex
        D1 □ new value
        Release Mutex
    end  while
end
```

Task T4

```
Begin
    While (true)

        P ( Binary semaphore) S1
        Function F2
        Write into queue Q3

    end  while
end
```

Application Design Steps (2)

» Application programming

Advice :

→ **Do not** implement compile and run

→ Build the application step by step (C language)

» Tests and Validation

Advice :

→ Use of **absolute time** to measure the real CPU load of each task, to validate the relative priority between tasks.

Institut Supérieur de l'Aéronautique et de l'Espace

10, avenue Edouard Belin – BP 54032
31055 Toulouse Cedex 4 – France
T +33 5 61 33 80 80

www.isae-supaero.fr