Corail, ROS2 temps réel

Benoit Varillon, Jean-Baptiste Chaudron, David Doose, Charles Lesire

ONERA/ISAE-Supaero,
Université de Toulouse, France

ROSConFR
Juin 2021

Ce travail a été financé par l'Agence de l'Innovation de Défense (AID) du Minisère des Armées
(projet de recherche CONCORDE N° 2019 65 0090004707501)

Programme Temps Réel

- Interactions avec l'environnement extérieur
- La qualité du résultat dépend du temps de calcul

➔ Contraintes temporelles fortes

- échéances de réponses
- fréquences de traitements

⚠

Répondre à temps ≠ Répondre vite

Programme temps réel

⇒ Vérification fonctionnelle **et Vérification temporelle**

Vérification Temporelle

Vérifier que Toutes les échéances soient respectées
Déterminer le pire temps de réponse de chaque tâche du système

- Être capable de calculer/borner les temps de réponses
  - ➔ Théorie de l'ordonnancement
- Prévoir le comportement du programme
  - ➔ Déterminisme de l'exécution

ROS et le Temps Réel

Gestion de la communication :

- Qualité de Service (QoS)
- DDS
- Inter-process communication

Gestion de l'exécution :

- Tâches périodiques
- Comportements réactifs (subscriptions/services)

Théorie vs ROS2

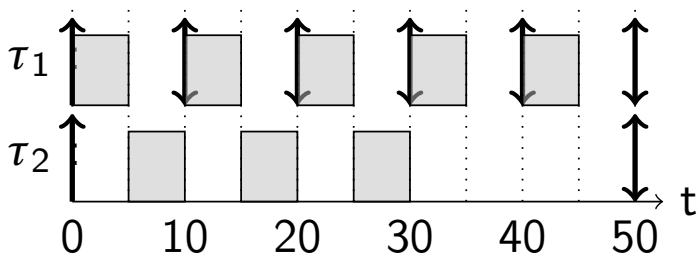
- ex. 2 tâches périodiques
- paramètres suivants :

	T	C	r	P
τ_1	**10**	5	0	1
τ_2	**50**	15	0	2

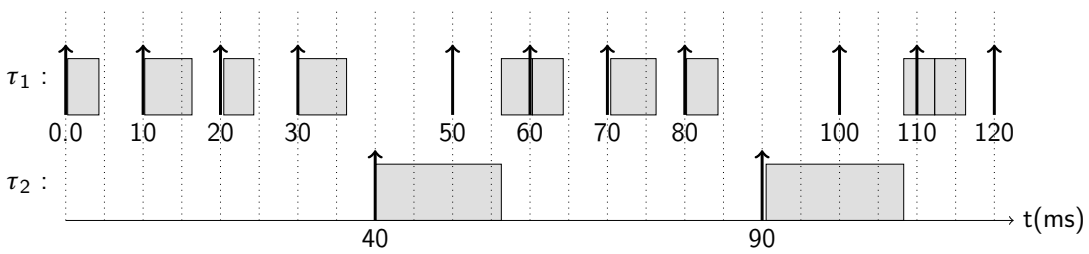
Code ROS2 :

```
class test_node :
        public rclcpp::Node
{
    public :
        test_node(std::string name) :
     Node(name)
        {
        t1_ = create_wall_timer(
            10ms,
     std::bind(&test_node::worker1,this));

        t2_ = create_wall_timer(
            50ms,
     std::bind(&test_node::worker2,this));


        }
```

Timeline théorique :



Timeline réelle :



Problèmes

Execution par défaut (un seul nœud)

- Un seul thread
- Pas de synchronisation des tâches
- Pas de gestion des priorités
- Pas de préemption
  - ➔ non respect des échéances
  - ➔ début d'exécution chaotique
  - ➔ analyse (précise) impossible

Théorie vs ROS2

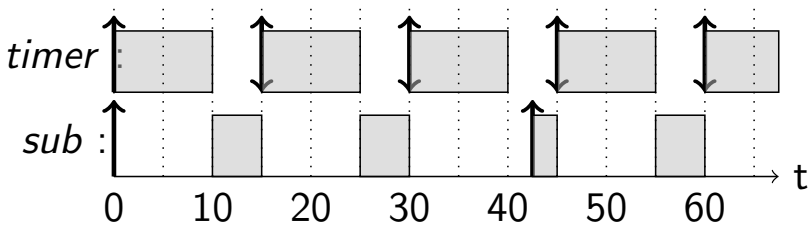
- une tâche périodique
- une subscription

	T	C	r	P
timer	**15**	10	0	1
subscription	∅	10	0	2

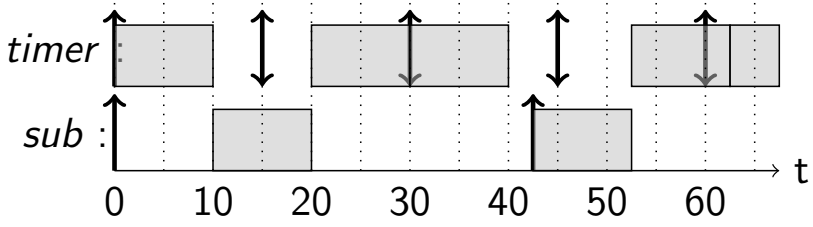
Code ROS2 :

```
class test_node :
    public rclcpp::Node
{
    public :
    test_node(std::string name) : Node(name)
    {
        timer_ = create_wall_timer(
            15ms,
     std::bind(&test_node::worker,this));

        sub_ =
     create_subscription<std_msgs::msg::Int64>
     (
            "test_ros", 10,
     std::bind(&test_node::subCallback,
     this, _1));

    }
```

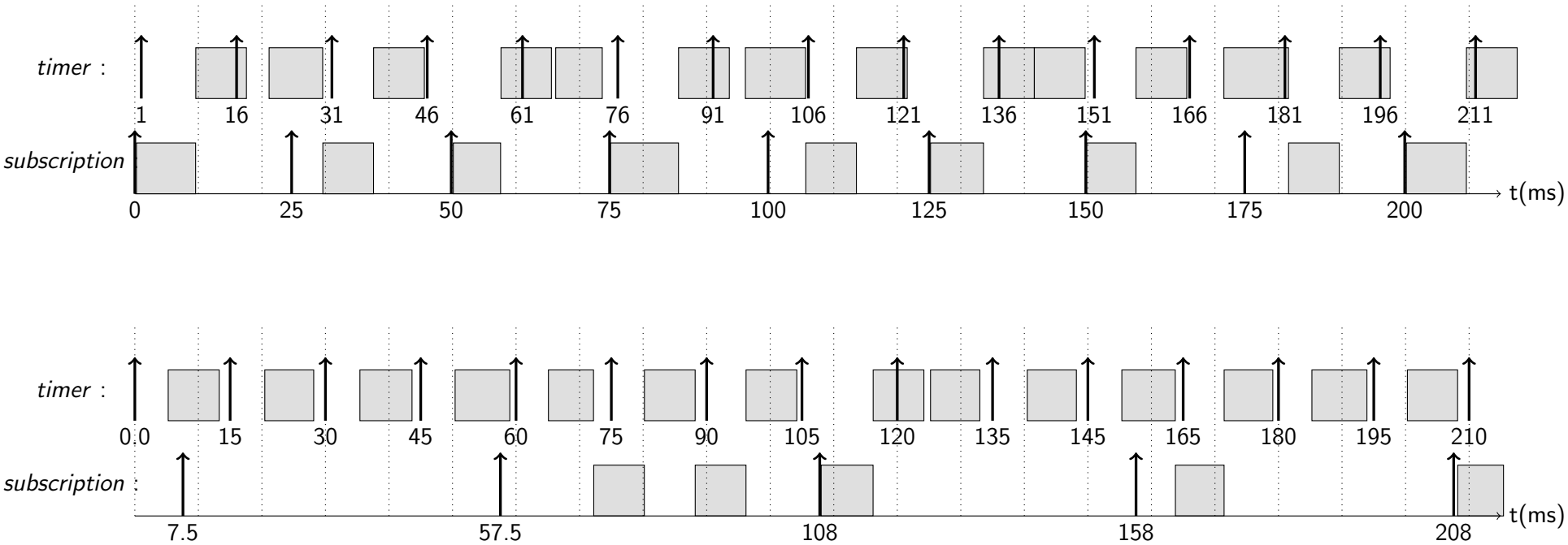
Timeline théorique :



Timeline ROS2 :



Exemples d'exécutions



Problèmes

- Un seul thread
- Pas de synchronisation des tâches
- Pas de gestion des priorités
- Pas de préemption
  - ➔ début d'exécution chaotique
  - ➔ non respect des échéances
  - ➔ analyse (précise) impossible

Début de solution

- les exécuteurs ROS2

- Model réactifs non compatible avec l'analyse RT
- Surcharge si trop de messages
- Priorité entre tâches/subscriptions/services figés par l'exécuteur

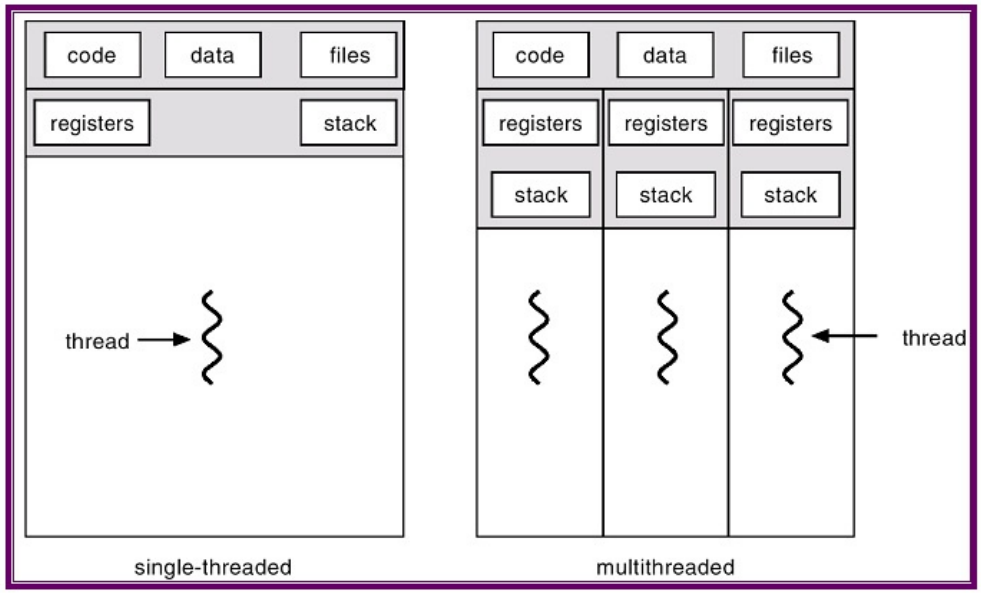
Fonctionnement de ROS2

`rclcpp::Executor`

- `SingleThreadExecutor`
- `MultiThreadExecutor`

`rclcpp::spin()`

➔ `SingleThreadExecutor`



Utilisation de plusieurs threads :

- MultiThreadExecutor (thread pool) → parallélisation multicœurs

➔ Création des threads "à la main"

Corail vs ROS2

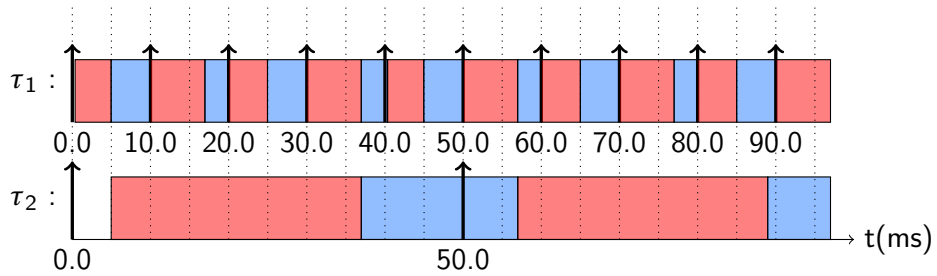
Code Corail :

```
class test_node :
        public corail_core::RealTimeNode
{
    public :
        test_node(std::string name) :
     RealTimeNode(name)
        {
        t1_ = create_rt_timer(
            "task1", 0, 20,
            10ms,
     std::bind(&test_node::worker1,this));

        t2_ = create_rt_timer(
            "task2", 0, 10,
            50ms,
     std::bind(&test_node::worker2,this));
        }
```

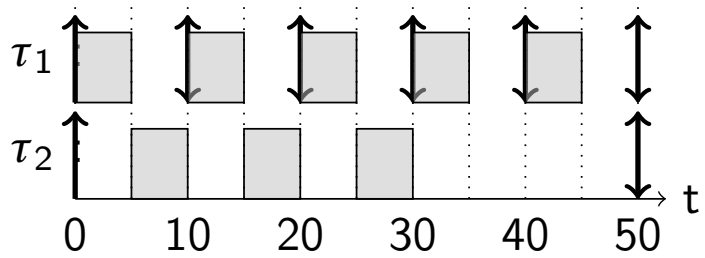
Code ROS2 :

```
class test_node :
        public rclcpp::Node
{
    public :
        test_node(std::string name) :
     Node(name)
        {
        t1_ = create_wall_timer(
            10ms,
     std::bind(&test_node::worker1,this));

        t2_ = create_wall_timer(
            50ms,
     std::bind(&test_node::worker2,this));


        }
```

Timeline Corail :

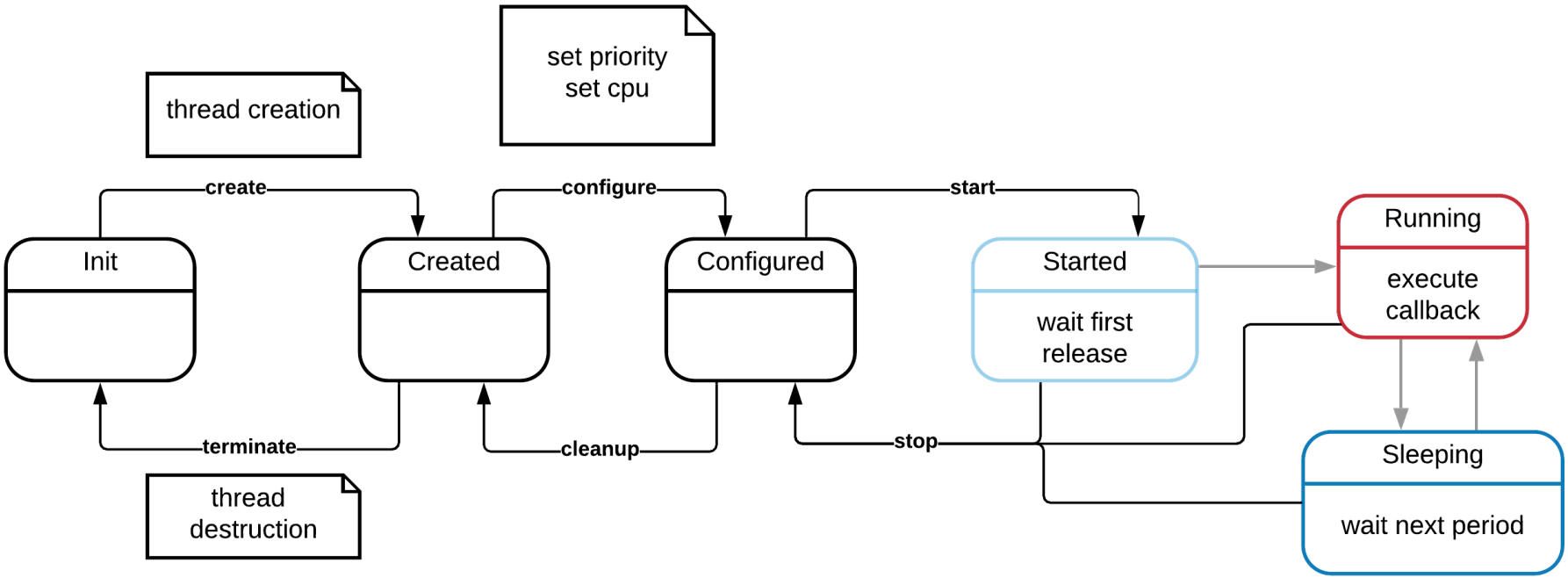


Timeline théorique :



Fonctionnement Corail

- Nouvel exécuteur
- Une tâche ⇔ un thread (POSIX) → priorités(SCHED_FIFO) et cpu
- Diagramme d'état :



Corail vs ROS2

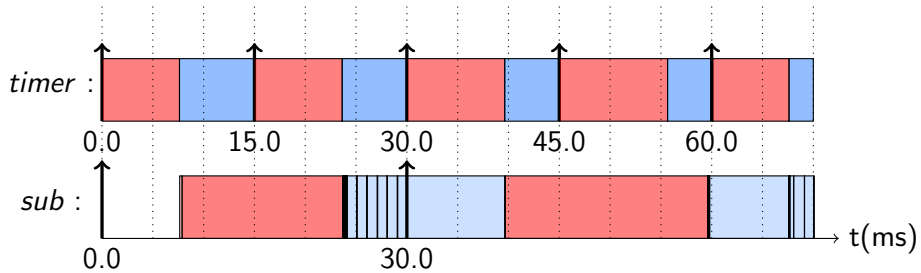
Code Corail :

```
class test_node :
        public corail_core::RealTimeNode
{
    public :
    test_node(std::string name) :
     RealTimeNode(name)
    {
        timer_ = create_rt_timer(
            "timer", 0, 20,
            15ms,
     std::bind(&test_node::worker,this));

        sub_ =
     create_rt_subscription<std_msgs::msg::Int
            "sub", 0, 10, 2ms, 10ms,
            "test_ros", 1,
     std::bind(&test_node::subCallback,
     this, _1));
    }
```

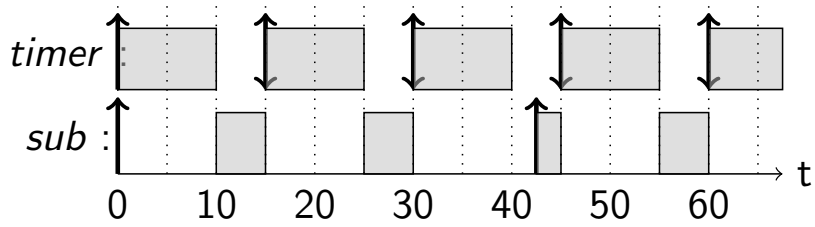
Code ROS2 :

```
class test_node :
    public rclcpp::Node
{
    public :
    test_node(std::string name) :
     Node(name)
    {
        timer_ = create_wall_timer(
            15ms,
     std::bind(&test_node::worker,this));

        sub_ =
     create_subscription<std_msgs::msg::Int64>
     (
            "test_ros", 10,
     std::bind(&test_node::subCallback,
     this, _1));

    }
```

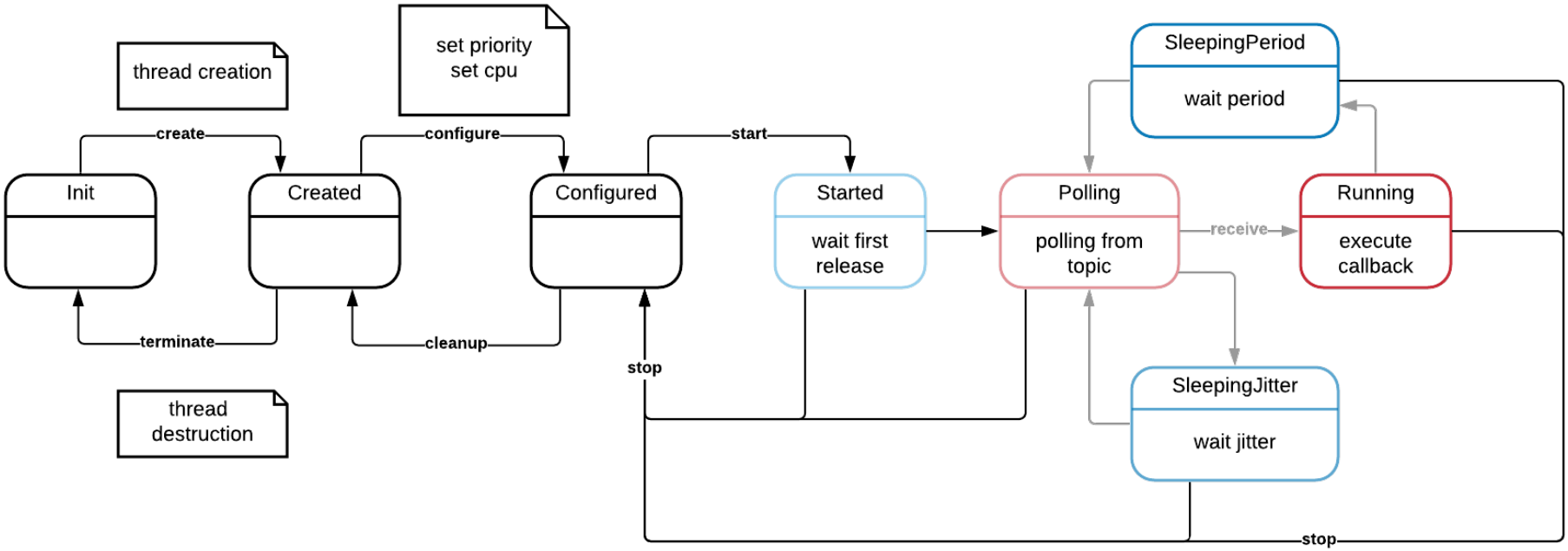
Timeline Corail :



Timeline théorique :



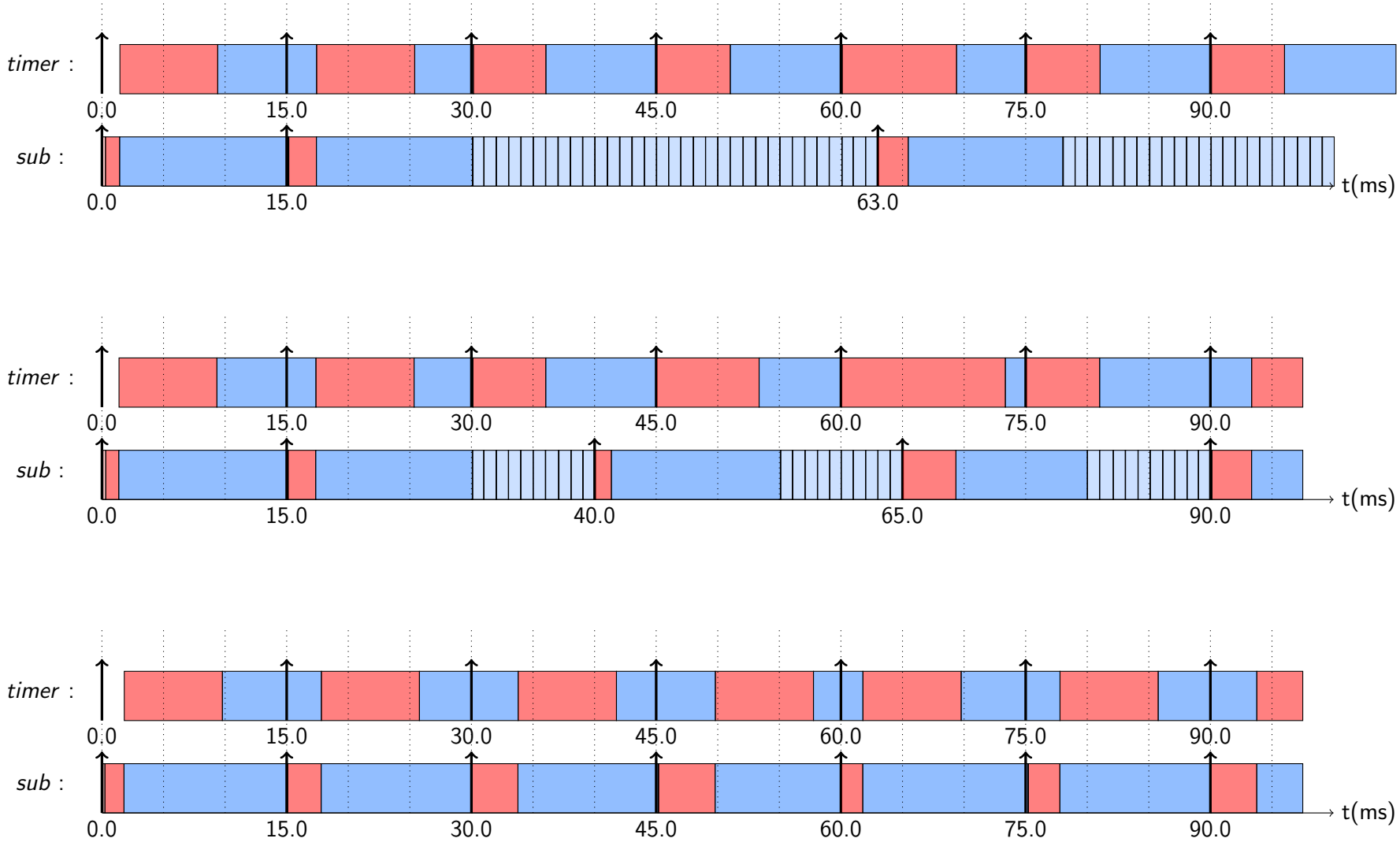
Details subscriptions et services



Deux periodes différentes :

- jitter : période de polling
- period : temps minimal entre deux exécution

➔ Plus de model réactifs


timer :
0.0
15.0
30.0
45.0
60.0
75.0
90.0
sub :
0.0
15.0
63.0
t(ms)
timer :
0.0
15.0
30.0
45.0
60.0
75.0
90.0
sub :
0.0
15.0
40.0
65.0
90.0
t(ms)
timer :
0.0
15.0
30.0
45.0
60.0
75.0
90.0
sub :
0.0
15.0
30.0
45.0
60.0
75.0
90.0
t(ms)

RealTimeExecutor

- POSIX (pthread)
- Un thread par tâche
- Synchronisation des tâches via des barrières
- Gestion des priorités
- Utilisation de l'ordonnanceur temps réel (SCHED_FIFO)
  - ➔ Respect des échéances
  - ➔ Exécution déterministe
  - ➔ Méthode d'analyse disponible

Conclusion/Perspectives

- API simple et proche de rclcpp
- Exécution temps réel robuste et déterministe
- Traces LTTng permettant l'étude et la validation du système
- Outils d'analyses temps réel
  - Analyse des traces
  - Analyse de l'ordonnançabilité
- Autres stratégies d'exécution (process au lieu de thread)
- Gestion de systèmes distribués

`https://corail1.gitlab.io/`

`https://gitlab.com/corail1/roscon_fr_21`