24 - Gérer le paraléllisme

Programmation Concurrente - LI330
Université P. & M. Curie - année scolaire 2013/2014

PrC

QUELQUES PROBLÈMES INDUITS PAR LE PARALLÉLISME

Nous ne pourrons les voir tous!

<table>
<tr><th>Besoins</th><th colspan="2">Mécanismes</th></tr>
<tr><td>Échange d'information</td><td colspan="2">Mécanismes de communication</td></tr>
<tr><td>Accès à des ressources partagées</td><td colspan="2">Synchronisation</td></tr>
<tr><td>Interblocage</td><td colspan="2">Algorithmes de détection ou de prévention</td></tr>
<tr><td>Famine</td><td rowspan="2">Estampillage</td><td>Algorithmes de prévention</td></tr>
<tr><td>Causalité</td><td>Gestion d'horloges</td></tr>
</table>

Mécanismes de communication

- Communication synchrone
  - La notion de rendez-vous
    - Le premier arrivé attend l'autre (se suspend)
    - L'arrivée du second provoque l'échange
    - L'échange terminé, les deux «interlocuteurs» reprennent leur exécution
    - Ca se passe un peu comme un appel de procédure ou de méthode
- Communication asynchrone
  - La notion de message
    - L'émetteur envoie son message et continue son exécution
    - Le récipiendaire lit le message (il attend s'il «arrive le premier»)
    - Cela se passe un peu comme dans une «boîte aux lettres»
  - Sémantique proche de celle des objets/types protégés

Mécanismes plus évolués...

- Pour la culture (sera vu plus tard)

- Multi-rendez-vous (ou barrière de synchronisation)
  - Synchronisation entre N>2 processus

- Diffusion (broadcast)
  - Émission à destinataires multiples

- Diffusion par classe (multicast)
  - Émission à destinataires enregistrés

Accès à des ressources partagées

Considérons l'exécution des deux programmes suivants

Copier R dans A	Copier R dans B
A <- A + 10	B <- B - 10
Copier A dans R	Copier B dans R

Processus 1	A	R	B	Processus 2
copier R dans A	10	10		
A <- A + 10	20	10		
	20	10	10	copier R dans B
	20	10	0	B <- B - 10
	20	0	0	copier B dans R

Résultat : R = 20, est-ce le seul résultat possible?

Ressources: définition et mécanismes d'accès

- Différents types de ressources
  - Ressources critiques
    - Accès exclusif
  - Ressources banalisées
    - Classe de ressources ayant un accès exclusif
- Protection de l'accès aux ressources
  - Section critique
    - Section de code "protégée"
  - Réservation (demande d'entrée)
    - Potentiellement bloquante
  - Libération (notification de sortie)
    - Non bloquante
    - Débloque éventuellement des processus en attente

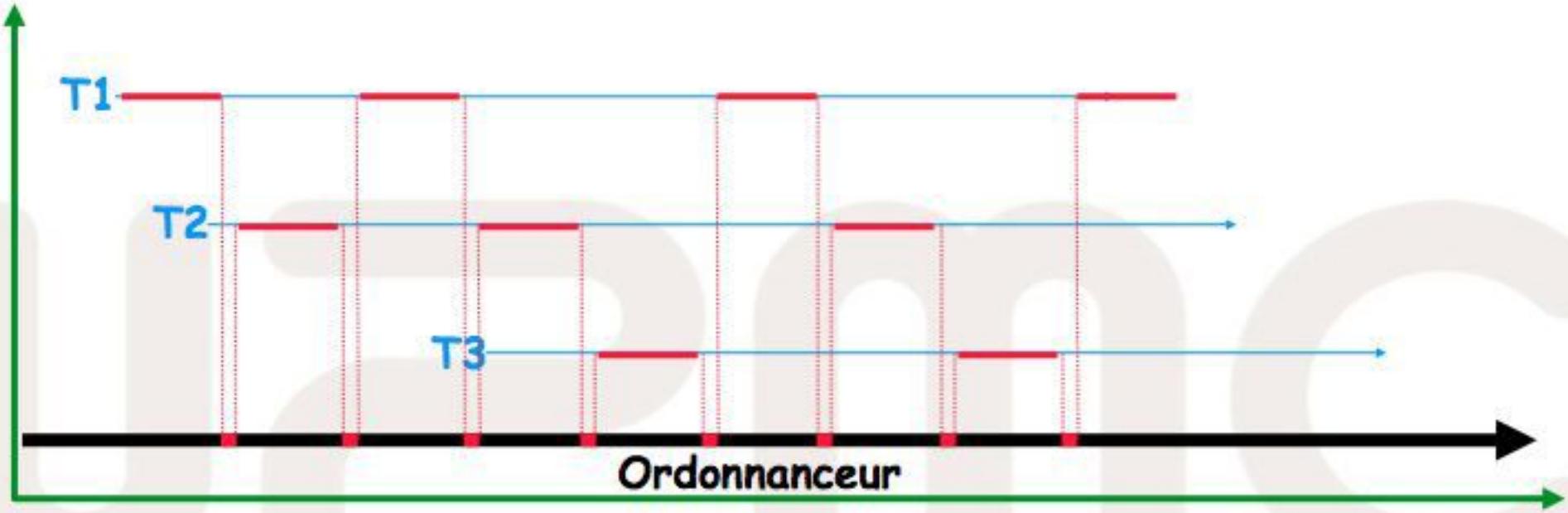
Processus 1	A	R	B	Processus 2
	10	?		réserver R
	10	10		copier R dans B
	10	0		B <- B - 10
réserver R	?	10	0	
Suspension jusqu'à libération par 2	0	0		copier B dans R
		0		libérer R
copier R dans A		0	0	
A <- A + 10		10	0	
copier A dans R		10	10	
libérer R				

Implémentation de ces blocages?
Rappels sur l'exécution multi-tâche...

La réalité du parallélisme se ramène à une notion de «chemin d'exécution» qui passe par différents éléments de programmes...

Le processeur ne «s'arrête jamais» mais partage son exécution

Rappels sur l'exécution multi-tâche: le rôle du «scheduler»



- File d'attente des processus gérée par le «scheduler»
- Différents types de processus
  - Élu
  - Éligible
  - Suspendu
- Choix d'un algorithme d'élection («round robbin», avec priorités, etc)

Besoins pour la protection des ressources

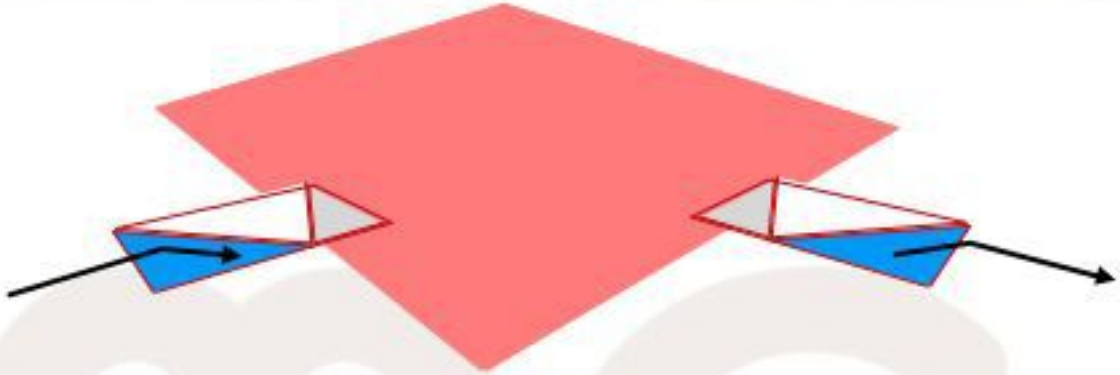
- Une seule solution!!!
  - Bloquer les processus pour séquentialiser les accès
  - Le parallélisme n'est pas possible dans une section critique
    - Ou limité si la section critique est associé à une ressource banalisée
- Il faut donc disposer d'un moyen de bloquer les appelants
  - Suspension de l'exécution d'un processus
  - Le mécanisme peut être local (concurrence)
    - Sémaphores
    - Moniteurs
  - On peut s'appuyer sur l'attente d'une réponse à un message (réparti)
    - Attente bloquante sur un message qui doit arriver...
- Le rôle du scheduler est essentiel!
  - Ca devient délicat sur les architectures multi-cœur...

Implémenter la protection des ressources

- Encapsuler l'accès à la ressource dans un serveur
  - Processus concurrent (ou processus léger)
    - C'est le seul à accéder à la ressource
    - Il traite des requêtes
  - Gestion «archi-classique» (spooler, serveur de mail, etc...)
    - Convient très bien à une programmation concurrente
    - Moins adapté à une programmation répartie
      - Mauvaise gestion de panne du serveur, traffic sur le réseau, etc.
- Utilisation de «jetons»
  - L'accès à la donnée est matérialisée par un jeton
    - Il peut contenir la valeur de la donnée
    - Il donne le privilège de manipuler la dite donnée
    - Il peut se transmettre d'un processus à l'autre
  - Mécanisme algorithmique très connu
    - Convient très bien à une programmation répartie (base pour le p2p)
    - Moins adapté à la programmation concurrente

Exemple de protection: «le parking» (ressources banalisés)

- Parking de N places
  - Une entrée
  - Une sortie
- Assurer que les véhicules qui rentrent ont forcément une place!



```
Réserver (place)
Entrer
Se garer
Se promener
Libérer (place)
sortir
```

Gestion d'un compteur de places
File d'attente des requêtes suspendues

Le parking en Ada avec des tâches (1)


```
with Ada.Text_Io;
   use Ada.Text_Io;

   procedure Parking is

      -- Constantes
      Parking_Size : constant Positive := 5;
      Nb_Customers : constant Positive := 8;

      -- Declaration des taches
      task Parking_Manager is
         entry Enter_Parking (
               Id : in      Positive );
         entry Exit_Parking (
               Id : in      Positive );
      end Parking_Manager;

      task type Customer (Id : Positive);
      type A_Customer is access Customer;
```

Le parking en Ada avec des tâches (2)


```
-- Le corps du gestionnaire du parking
   task body Parking_Manager is
      Free_Slots : Natural := Parking_Size;
   begin
      loop
         select
            when Free_Slots > 0 =>
            accept Enter_Parking (
                  Id : in     Positive ) do Free_Slots := Free_Slots - 1;
               Put_Line ("Parking:" & Positive'Image (Id) & " entre, il
reste" &
                  Natural'Image (Free_Slots) & " places");
            end Enter_Parking;
         or
            accept Exit_Parking (
                  Id : in     Positive ) do Free_Slots := Free_Slots + 1;
               Put_Line ("Parking:" & Positive'Image (Id) & " sort, il reste"
&
                  Natural'Image (Free_Slots) & " places");
            end Exit_Parking;
         or
            terminate; -- On se termine proprement
         end select;
```

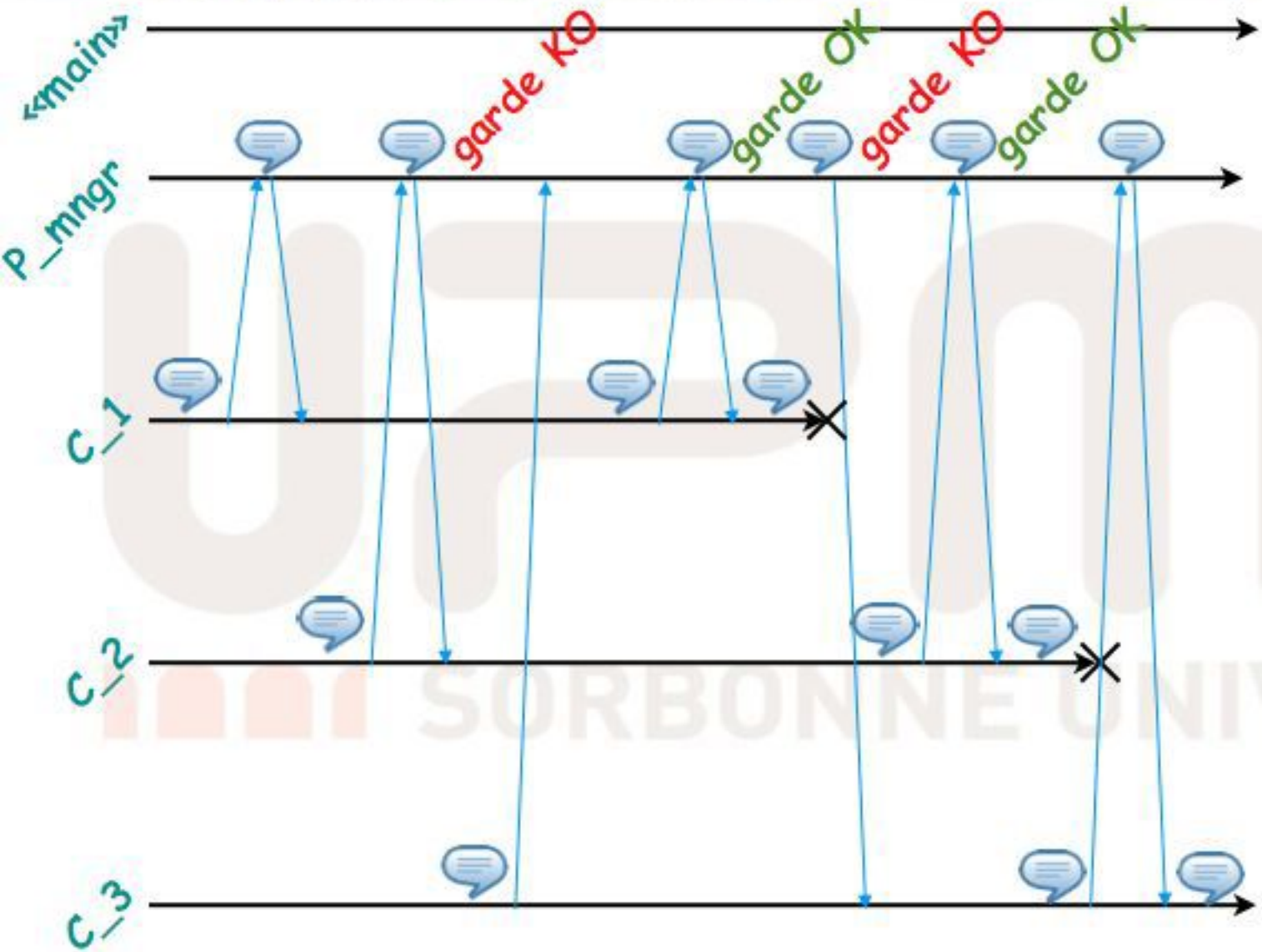
Le parking en Ada avec des tâches (3)


```
-- Le corps d'un usager normal
   task body Customer is
   begin
      Put_Line ((1.. Id => ' ') & Positive'Image(Id) &
                " veut entrer");
      Parking_Manager.Enter_Parking (Id);
      delay 1.0 * duration (id mod 3); -- attente de duree variable
      Put_Line ((1.. Id => ' ') & Positive'Image(Id) &
                " veut sortir");
      Parking_Manager.Exit_Parking (Id);
   end Customer;

   Ptr : A_Customer;

begin -- Parking
   for I in 1.. Nb_Customers loop
      Ptr := new Customer (I);
   end loop;
end Parking;
```

Exécution du parking

```
$ ./parking
  1 veut entrer
   2 veut entrer
Parking: 1 entre, il reste 4 places
Parking: 2 entre, il reste 3 places
     3 veut entrer
Parking: 3 entre, il reste 2 places
      4 veut entrer
     3 veut sortir
Parking: 4 entre, il reste 1 places
Parking: 3 sort, il reste 2 places
       5 veut entrer
Parking: 5 entre, il reste 1 places
        6 veut entrer
Parking: 6 entre, il reste 0 places
        6 veut sortir
Parking: 6 sort, il reste 1 places
         7 veut entrer
Parking: 7 entre, il reste 0 places
            8 veut entrer
   1 veut sortir
Parking: 1 sort, il reste 1 places
Parking: 8 entre, il reste 0 places
      4 veut sortir
Parking: 4 sort, il reste 1 places
          7 veut sortir
Parking: 7 sort, il reste 2 places
    2 veut sortir
Parking: 2 sort, il reste 3 places
       5 veut sortir
Parking: 5 sort, il reste 4 places
            8 veut sortir
Parking: 8 sort, il reste 5 places
```

Le Chronogramme (2 places, 3 clients)



1 veut entrer
Parking: 1 entre, il reste 1 places
2 veut entrer
Parking: 2 entre, il reste 0 places
3 veut entrer
1 veut sortir
Parking: 1 sort, il reste 1 places
Parking: 3 entre, il reste 0 places
3 veut sortir
Parking: 3 sort, il reste 1 places
2 veut sortir
Parking: 2 sort, il reste 2 places

Pour rire... le parking en Ada avec un object protégé (1)


```
with Ada.Text_Io;
   use Ada.Text_Io;

   procedure Parking_2 is

      -- Constantes
      Parking_Size : constant Positive := 5;
      Nb_Customers : constant Positive := 8;

      -- Declaration des taches
      protected Parking_Manager is
         entry Enter_Parking (Id : in Positive);
         procedure Exit_Parking (Id : in Positive);
      private
         Free_Slots : Natural := Parking_Size;
      end Parking_Manager;

      task type Customer (Id : Positive);
      type A_Customer is access Customer;
```

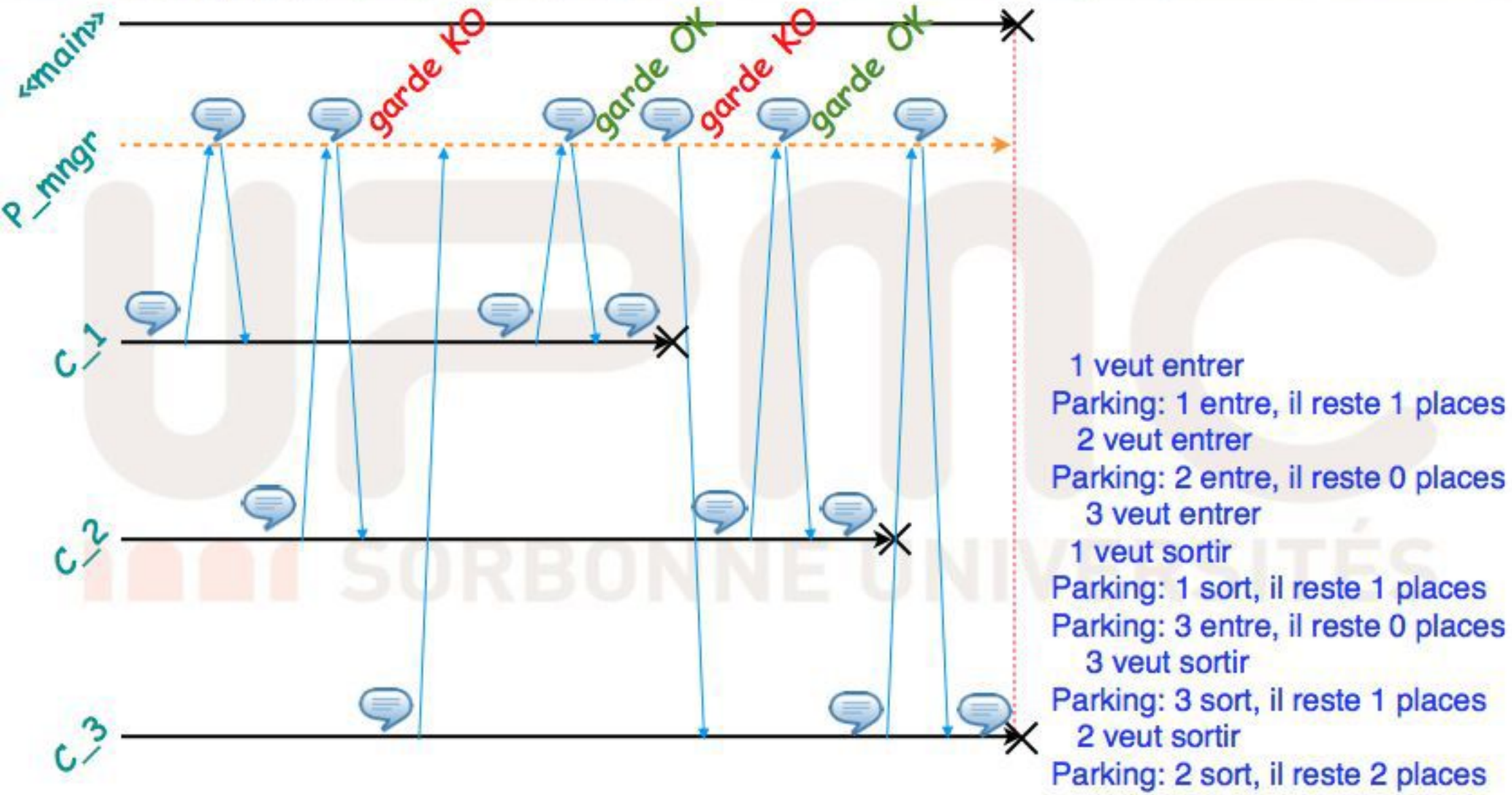
Pour rire... le parking en Ada avec un object protégé (2)


```
-- Le corps du gestionnaire du parking
protected body Parking_Manager is

   entry Enter_Parking (Id : in Positive) when Free_Slots > 0 is
   begin
      Free_Slots := Free_Slots - 1;
      Put_Line ("Parking:" & Positive'Image (Id) & " entre, il reste" &
         Natural'Image (Free_Slots) & " places");
   end Enter_Parking;

   procedure Exit_Parking (Id : in Positive) is

   begin
      Free_Slots := Free_Slots + 1;
      Put_Line ("Parking:" & Positive'Image (Id) & " sort, il reste" &
         Natural'Image (Free_Slots) & " places");
   end Exit_Parking;
end Parking_Manager;
```

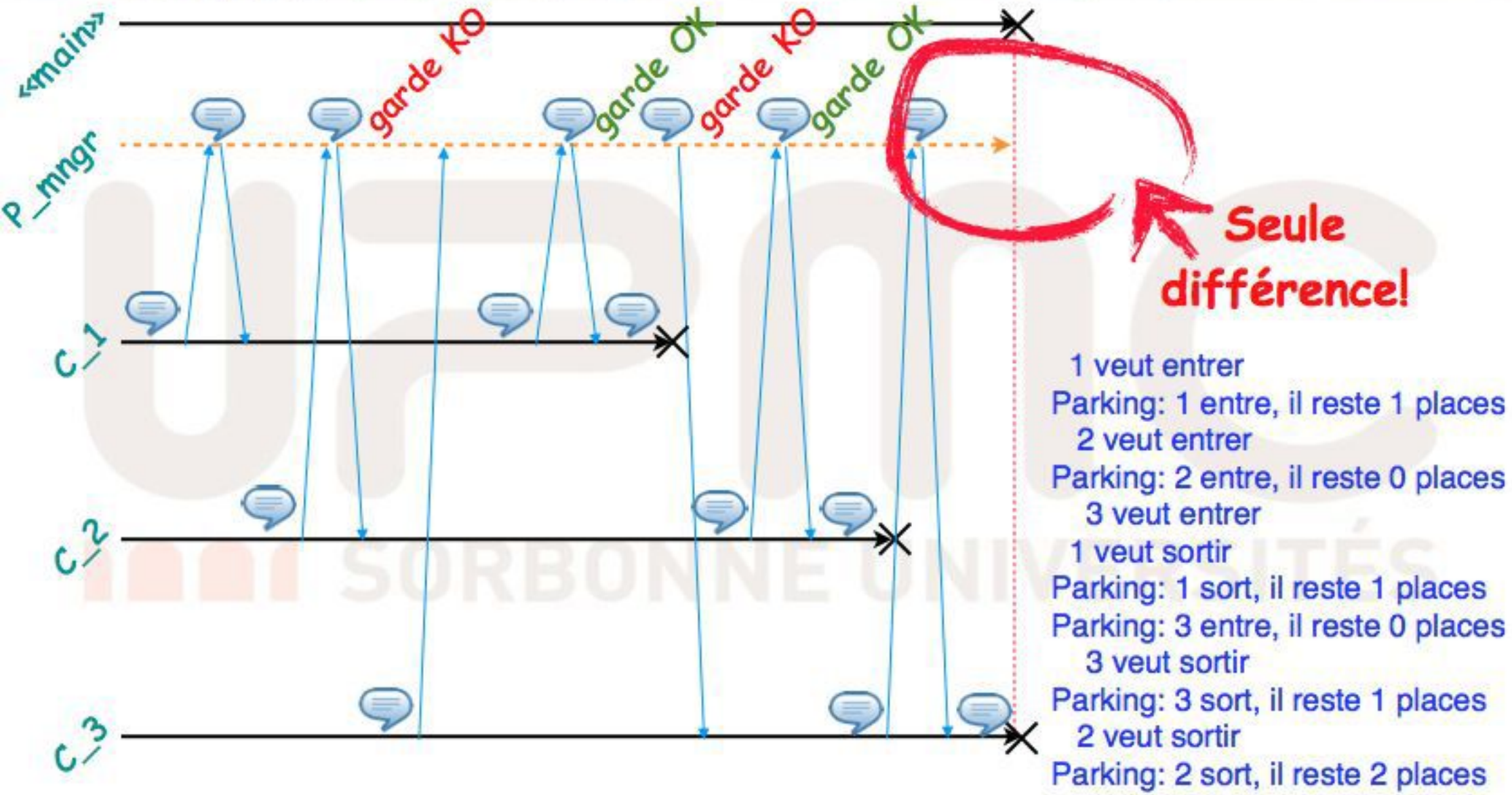
Pour rire... le parking en Ada avec un object protégé (1)


```
-- Le corps d'un usager normal
   task body Customer is
   begin
      Put_Line ((1.. Id => ' ') & Positive'Image(Id) & " veut entrer");
      Parking_Manager.Enter_Parking (Id);
      delay 1.0 * Duration (Id mod 3); -- attente de duree variable
      Put_Line ((1.. Id => ' ') & Positive'Image(Id) & " veut sortir");
      Parking_Manager.Exit_Parking (Id);
   end Customer;

   Ptr : A_Customer;

begin -- Parking
   for I in 1.. Nb_Customers loop
      Ptr := new Customer (I);
   end loop;
end Parking_2;
```

Le Chronogramme (2 places, 3 clients), la même séquence qu'avant



Le Chronogramme (2 places, 3 clients), la même séquence qu'avant



Remarques à ce stade

La synchronisation est un mécanisme de protection des données

Rendez-vous et objets protégés servent à synchroniser les tâches