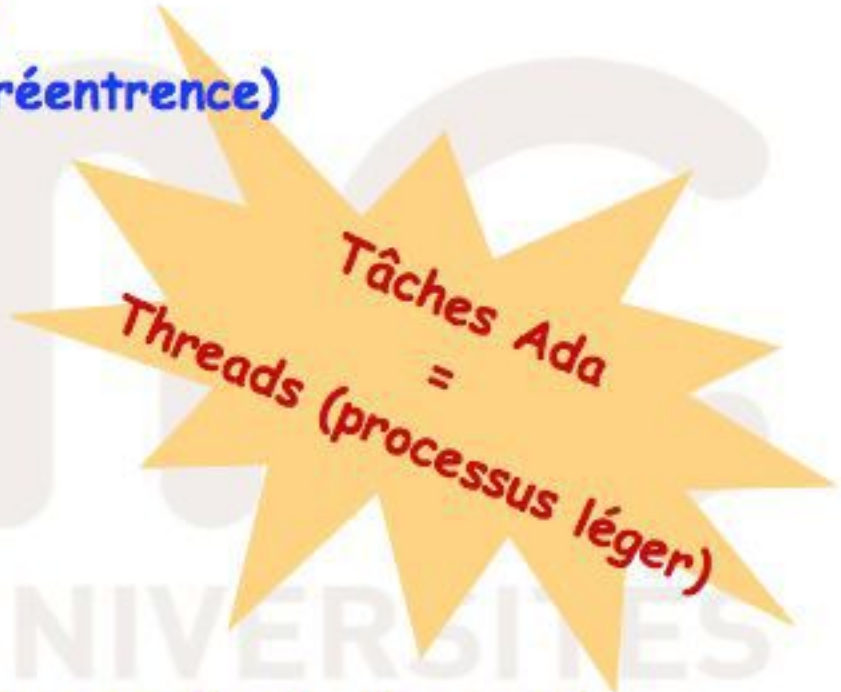
13 - Objets et types tâche

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

PrC

Tâche en Ada

- «task» Ada: expression d'un flux d'exécution en Ada
  - Son propre compteur ordinal
  - Son propre contexte (pile, «registres», etc.)
  - Accès à la mémoire partagée (attention à la réentrence)
  - Points de communication
    - Des files de messages typés («entry»)
    - Notion de requêtes
- Objet tâches
  - Un seul exemplaire de la tâche
  - Niveau de visibilité: le programme principal
- Type tâche
  - Possibilité de créer plusieurs flux d'exécutions associé au même code
    - Cela a un sens car le contexte initial peut-être différent
    - Cela ouvre la possibilité de créer dynamiquement des «task» Ada
  - Niveau de visibilité variable (programme, procédure ou bloc)



Quelques considérations sur l'exécution concurrente en Ada

- Le programme principal est une tâche (tache d'environnement)
  - Pas de file de requêtes
- Terminaison (globale)
  - Le programme se termine lorsque toutes les tâches sont terminées
    - Le «programme principal» (tâche mère)
    - Les tâches qui ont été déclarées dans ce programme
- Cas particulier: les tâches déclarées dans des blocs ou des sous-programmes
  - La sortie du bloc/sous-programme => les tâches qui y sont déclarées sont terminées
  - Sémantique de «co-begin»
    - Co-begin (A; B; C)
      - A exécuté en parallèle de B et C,
      - Synchronisation sur la terminaison de A, B et C

Syntaxe de définition d'une tâche

- **Spécification d'une tâche**
  - Points d'entrée
  - Le discriminant peut servir pour initialiser la tâche
    - Lecture seulement

```
decl_t  task [type] ident [(discrim)] [is
           {déclaration_pt_d_entrée}
        end identificateur];
```

- **Corps de la tâche**
  - Partie déclarative
    - Variables différentiées même dans le cas d'un type tâche
  - Les instructions
    - Points d'acceptation des requêtes
    - Traitement des exceptions

```
body_t task body ident [(discrim)] is
           [partie_déclarative]
       begin
           suite_d_instructions
       [exception
           traite_exception
           {traite_exception}]
       end ident;
```

Exemple 1, tâches

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

procedure Ex1_Taches is
   task T1;
   task T2;

   task body T1 is
   begin
      for I in 1 .. 5 loop
         Put_Line ("Bonjour de T1");
      end loop;
   end T1;

   task body T2 is
   begin
      for I in 1 .. 5 loop
         Put_Line ("Bonjour de T2");
      end loop;
   end T2;

begin
   for I in 1 .. 5 loop
      Put_Line ("Bonjour");
   end loop;
end Ex1_Taches;
```

```
$ ./ex1_taches
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T2
Bonjour de T2
Bonjour de T2
Bonjour de T2
Bonjour de T2
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
```

C'est une exécution possible

Fin du programme

Exemple 1bis (forcer la commutation)


```
with Ada.Text_Io;
use Ada.Text_Io;
procedure Ex1_Taches_Bis is
   task T1;
   task T2;

   task body T1 is
   begin
      for I in 1 .. 5 loop
         Put_Line ("Bonjour de T1");
        delay 0.1;
      end loop;
   end T1;

   task body T2 is
   begin
      for I in 1 .. 5 loop
         Put_Line ("Bonjour de T2");
        delay 0.2;
      end loop;
   end T2;

begin
   for I in 1 .. 5 loop
      Put_Line ("Bonjour");
     delay 0.15;
   end loop;
end Ex1_Taches_Bis;
```

```
$ ./ex1_taches_bis
Bonjour de T1
Bonjour de T2
Bonjour
Bonjour de T1
Bonjour
Bonjour de T2
Bonjour de T1
Bonjour
Bonjour de T1
Bonjour
Bonjour de T2
Bonjour de T1
Bonjour
Bonjour de T2
Bonjour de T2
```

Exemple 2, types de tâches

```
with Ada.Text_Io;
use Ada.Text_Io;
procedure Ex2_Taches is
   task type T1;
   task T2;

   task body T1 is
   begin
      for I in 1 .. 5 loop
         Put_Line ("Bonjour de T1");
      end loop;
   end T1;

   task body T2 is
   begin
      for I in 1 .. 5 loop
         Put_Line ("Bonjour de T2");
      end loop;
   end T2;

   -- Deux tâches de type T1 sont créées
   -- Une tâche anonyme T2 est créée.
   T1_A, T1_B : T1;
begin
   null;
end Ex2_Taches;
```

```
$ ./ex2_taches
Bonjour de T2
Bonjour de T2
Bonjour de T2
Bonjour de T2
Bonjour de T2
Bonjour de T2
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
Bonjour de T1
```

Cette thread est vide!

Blocage tant que T1_A, T1_B et T2 ne sont pas terminées

Exemple 3, types tâches avec discriminant


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

procedure Ex3_Taches is
   task type T1(P : Integer);

   task body T1 is
   begin
      for I in 1 .. P loop
       Put_Line ("coucou" &
                 Integer'image(P));
      end loop;
   end T1;

   T1_A : T1 (2);
   T1_B : T1 (3);

begin
   null;
end Ex3_Taches;
```

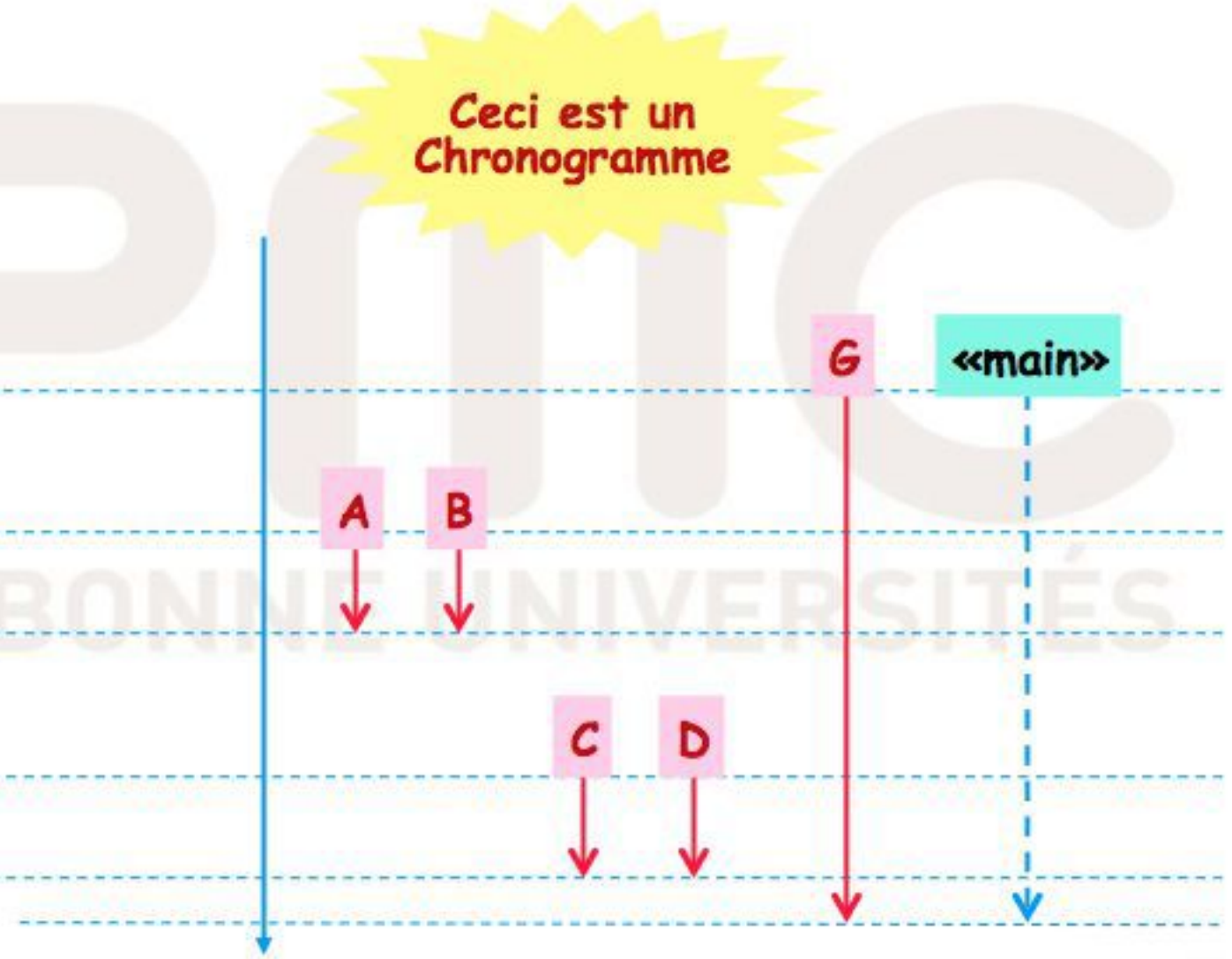
```
$ ./ex3_taches
coucou 2
coucou 2
coucou 3
coucou 3
coucou 3
```

Différenciation du contexte au moyen du discriminant

Contrôler la portée du parallélisme (co-begin/co-end)


```
with Ada.Text_Io;
use Ada.Text_Io;
procedure Dep_Taches is
   task type T1;
   task body T1 is
   begin
      Put_Line ("coucou");
   end T1;

   G: T1;
begin
   declare
      A, B : T1;
   begin
      null; -- Attente de A et B
   end;
   declare
      C, D: T1;
   begin
      null; -- Attente de C et D
   end;
end Dep_Taches; -- Attente de G
```



Mécanismes de communication

- Rendez-vous
  - Via la notion de point d'entrée
  - Communication de type synchrone
- Objets protégés
  - Variables munies d'un système de protection
    - Similaire aux moniteurs de Hoare
  - Communication asynchrone
- Variables partagées
  - Pas de protection (sauf mécanisme explicitement implémenté)
  - Accès à la mémoire commune
  - Exige en général la définition de ressources critiques
    - Rappel: synchroniser = suspendre un flux d'exécution, donc...
      ...on peut utiliser les objets protégés ou les rendez-vous pour cela