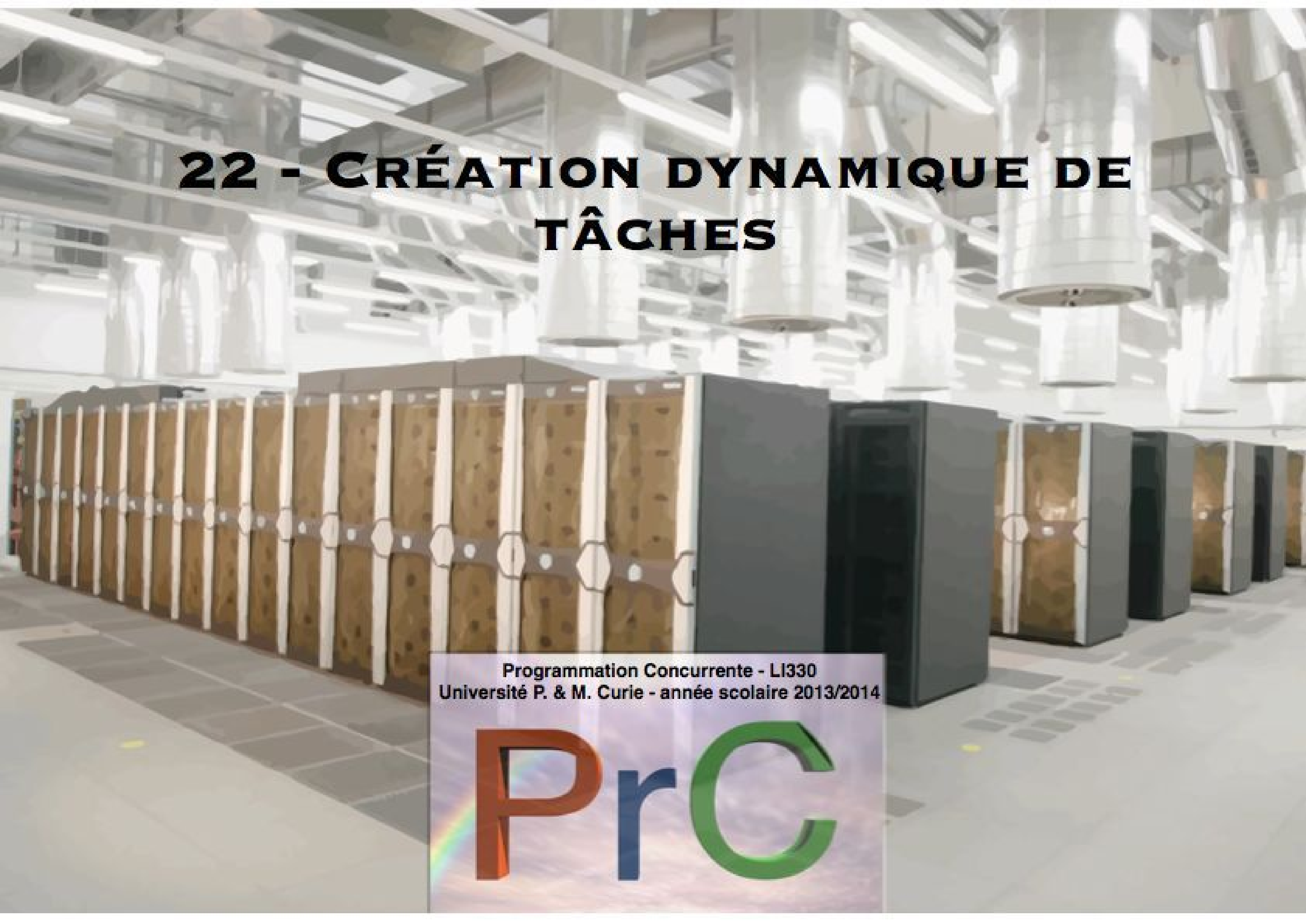
22 - Création dynamique de tâches



Pointeurs sur des tâches

- On peut construire un type pointeur sur un type tâche
  - De la même manière que l'on utilise les pointeurs traditionnels
- On peut allouer un espace à un pointeur sur un type tâche
  - Allocation d'espace mémoire (pile, contexte propre, etc)
  - Création d'une nouvelle tâche (flux d'exécution) de manière explicite
    - Pas de mécanisme «à la fork»
- Attention: une tâche créé dynamiquement dépend de celle qui l'a créé
  - Rappel: le programme principal est une tâche (tâche d'environnement)
    - La «mère de toutes les autres»
    - Une tâche ne se termine que lorsque toutes les tâches qui dépendent d'elle sont terminées

Exemple 1

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

procedure Ptr_Taches is

   task type T1 (Id : Positive);
   type A_T1 is access T1;

   task body T1 is
   begin
      for I in 1 .. 3 loop
         Put_Line ("Bonjour de" & Positive'Image(Id));
        delay 0.1;
      end loop;
   end T1;

   Ptr_T1 : A_T1;
   Ptr_T2 : A_T1 := new T1 (2);-- Création de tâche
   Ptr_T3 : A_T1;

begin
   Ptr_T1 := new T1 (1);-- Création de tâche
   Ptr_T3 := Ptr_T2; -- Pas de création de tâche
   PUT_LINE ("Fin du programme principal");
end Ptr_Taches;
```

Se déclare comme un type accès «normal»

Syntaxe usuelle de l'allocation

Recopie de pointeurs

Exemple 2

- On peut faire de la récursion (comme en shell;-)

```
with Ada.Text_Io; use Ada.Text_Io;
procedure Recursion is
   task type T (Id : Positive);
   Type A_T is access T;
   procedure Create_T (Val : in Positive) is
      Ptr : A_T;
   begin
      Ptr := new T (Val);
   end Create_T;
   task body T is
   begin
      Put_Line ("Debut de" & Positive'Image (Id));
      delay 0.1; -- Commutation artificielle
      if Id > 1 then
         Create_T (Id - 1);
      end if;
      delay 0.1; -- Commutation artificielle
      Put_Line ("Fin de" & Positive'Image (Id));
   end T;
begin
   Create_T (5);
   Put_Line ("Je me termine");
end Recursion;
```

Allocation dynamique = la portée n'est pas liée à la procédure

Exemple 3: serveur «multi-threadé»

- Principe
  - Le serveur a une thread qui «écoute» les demandes
  - Le serveur sous-traite les demandes
- Le serveur est donc
  - Une thread qui écoute
  - Une thread par service exécuté
- On peut varier les plaisirs
  - On va voir un exemple simple…

UPMC SORBONNE UNIVERSITÉS

Exemple 3 : les sources (1)

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

procedure Serveur_Mt is
    -- Les clients du système
    type Client;
    type A_Client is access Client;
    task type Client (Id : Positive) is
        entry Qui_Suis_Je (Ptr : A_Client);
        entry Previens_Moi;
    end Client;

    -- Le serveur
    task Serveur is
        entry Service (Pour : A_Client ; Id : Positive);
    end Serveur;
```

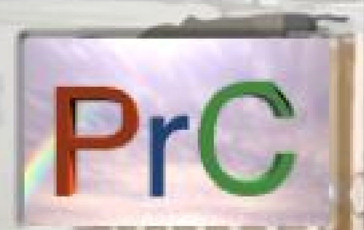
Exemple 3 : les sources (2)

```
-- Les exécutants du serveur
task type Travailleur (Id_T : Positive) is
   entry Travaille (Pour : A_Client ; Id : Positive);
end Travailleur;
type A_Travailleur is access Travailleur;
```

UPMC SORBONNE UNIVERSITÉS

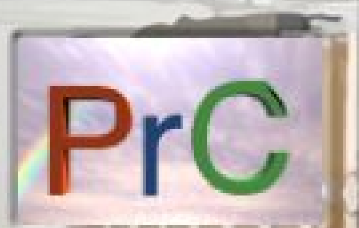
Exemple 3 : les sources (3)

```
-- Le corps du client
task body Client is
   Moi : A_Client;
begin
   -- Initialisation
   accept Qui_Suis_Je (Ptr : A_Client) do
      Moi := Ptr;
   end Qui_Suis_Je;
   delay 0.5 * Duration(Id mod 3);
   -- La demande de service
   Put_Line ((1..Id*2+2 => ' ') & "Client" & Positive'Image (Id) &
             " demande");
   Serveur.Service (Moi, Id);
   -- Attendre que l'on me réveille
   accept Previens_Moi;
   Put_Line ((1..Id*2+2 => ' ') & "Client" & Positive'Image (Id) &
             " se termine");
end Client;
```



Exemple 3 : les sources (4)


```
-- Le corps du serveur
task body Serveur is
   Executant : A_Travailleur;
   Cpt_Trav : Positive := 1;
begin
   loop
      select
         accept Service (Pour : A_Client ; Id : Positive) do
            Executant := new Travailleur (Cpt_Trav);
            Put_Line ("Serveur: traite requete pour" &
                      Positive'Image (Id));
            Executant.all.Travaille (Pour, Id);
            Cpt_Trav := Cpt_Trav + 1;
         end Service;
      or
         terminate;
      end select;
   end loop;
end Serveur;
```



Exemple 3 : les sources (5)


```
-- Le corps de l'executant
   task body Travailleur is
      Qui_Prevenir : A_Client;
      Id_Client : Positive;
   begin
      -- J'accepte un travail
      accept Travaille (Pour : A_Client ; Id : Positive) do
         Qui_Prevenir := Pour;
         Id_Client := Id;
      end Travaille;
      -- Je travaille
      Put_Line ("  Travailleur" & Positive'Image (Id_T) &
                "; bosse pour" & Positive'Image (Id_Client));
      delay 0.5 * Duration(Id_Client mod 4);
      -- Je préviens que le travail est terminé
      Qui_Prevenir.all.Previens_Moi;
   end Travailleur;
```

Exemple 3 : les sources (6)

```
-- Les variables globales du système
   Mes_Clients : array (1..8) of A_Client;

begin
    -- On cree les clients
    for I in Mes_Clients'Range loop
        Mes_Clients (I) := new Client (I);
        Mes_Clients (I).all.Qui_Suis_Je (Mes_Clients (I));
    end loop;
end Serveur_Mt;
```

UPMC SORBONNE UNIVERSITÉS

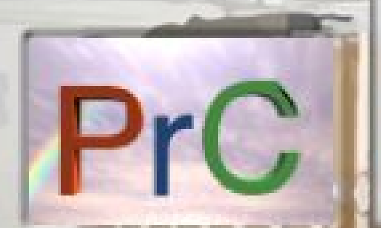
Exemple 3 : exécution

```
fko $ serveur_mt
        Client 3 demande
Serveur: traite requete pour 3
  Travailleur 1: bosse pour 3
            Client 6 demande
Serveur: traite requete pour 6
  Travailleur 2: bosse pour 6
   Client 1 demande
Serveur: traite requete pour 1
          Client 4 demande
              Client 7 demande
  Travailleur 3: bosse pour 1
Serveur: traite requete pour 4
  Travailleur 4: bosse pour 4
Serveur: traite requete pour 7
          Client 4 se termine
  Travailleur 5: bosse pour 7
       Client 2 demande
Serveur: traite requete pour 2
  Travailleur 6: bosse pour 2
            Client 5 demande
Serveur: traite requete pour 5
  Travailleur 7: bosse pour 5
              Client 6 se termine
                Client 8 demande
Serveur: traite requete pour 8
  Travailleur 8: bosse pour 8
                Client 8 se termine
    Client 1 se termine
        Client 3 se termine
            Client 5 se termine
       Client 2 se termine
              Client 7 se termine
```

UPMC SORBONNE UNIVERSITÉS

Chronogramme (pour 2 clients)


```
$ ./serveur_mt
   Client 1 demande
Serveur: traite requete pour 1
  Travailleur 1: bosse pour 1
    Client 2 demande
Serveur: traite requete pour 2
  Travailleur 2: bosse pour 2
   Client 1 se termine
    Client 2 se termine
```



En guise de conclusion...

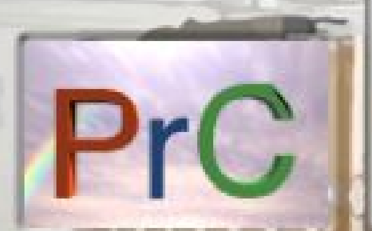
(tout est possible;-)

- On peut utiliser les pointeurs de façon très riche

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

procedure Ptr_Tab is
   task type T;

   task body T is
   begin
      Put_Line ("Coucou");
   end T;

   type T_T is array (1..3) of T;
   type A_T_T is access T_T;

   A_Tab : A_T_T;

begin
   A_Tab := new T_T;  -- Création d'une «batterie entière;-)»
   Put_Line ("Bye");
end Ptr_Tab;
```

- On échappe à une multitude d'appels systèmes
  - Manipulation de processus léger au niveau du langage
  - Attention aux variations dans l'implémentation par les compilateurs
    - L'indéterminisme n'est pas normalisé (normalisable;-)

UPMC SORBONNE UNIVERSITÉS



Remarque importante

- Désallocation d'une tâche?
  - Inutile de disposer d'un mécanisme explicite (comme pour la mémoire)
- Note importante: il existe deux types de gestion de mémoire
  - Allocation/Désallocation explicite
    - On utilise la mémoire comme on veut
    - Il faut être rigoureux
  - On fonctionne sur la base d'hypothèses qui permettent une gestion automatique (pas de risque de perte)
    - Modèle objet (constructeur/destructeur, «ramasse miettes»)
    - FIFO, LIFO
    - Zones de mémoire associées à des tâches (durée de vie liée à celles de la tâche)