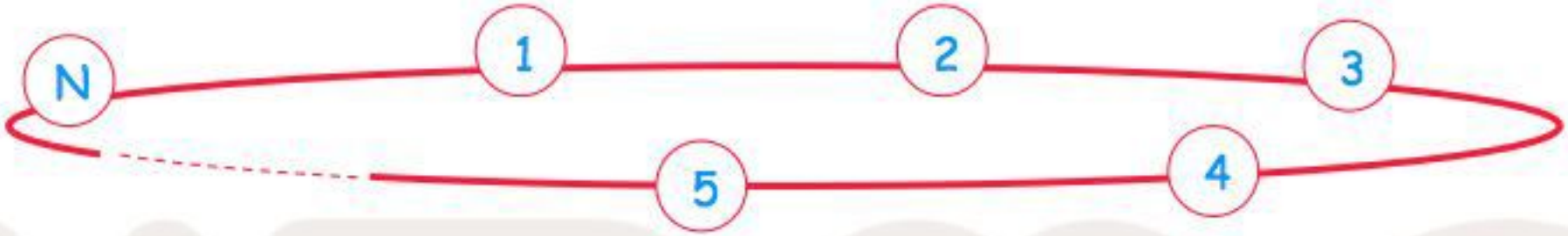
28 - ANNEAU DE PROCESSUS

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

PrC

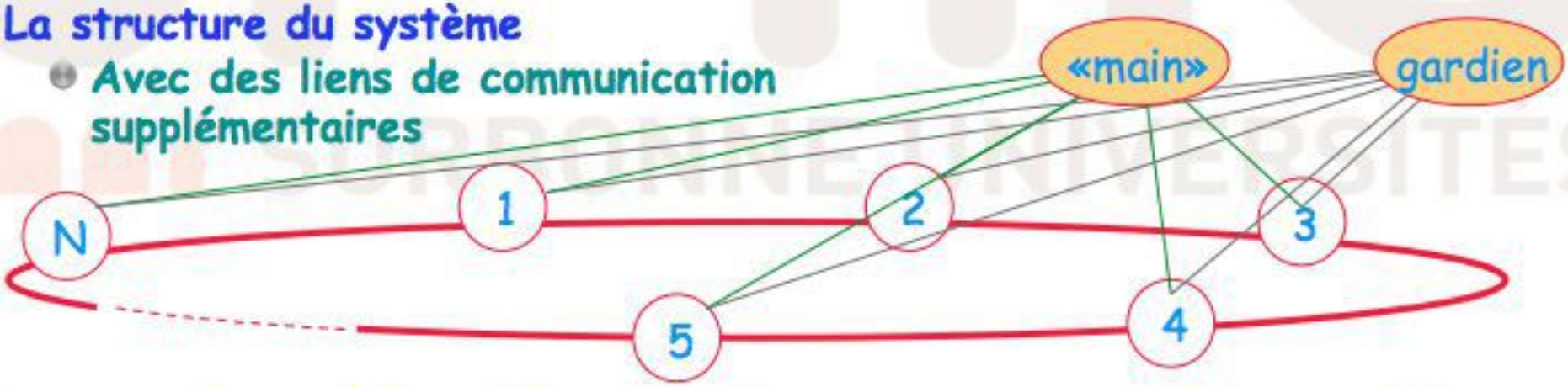
PRINCIPE



- Une façon de relier des tâches entre elles
  - Simplifier les relations entre des processus coopérants
    - Économiser les descripteurs de fichier (maximum autorisé par les OS)
    - Structurer des communications de certaines tâches de l'application
  - Souvent utilisé pour gérer certains moments critiques de l'exécution
    - Initialisation, Terminaison
    - Supervision de certains aspects
- Principe: les communications vont «de proche en proche»
  - Une ou deux directions possibles
  - Routage simplifié (si numérotation adéquate)
- Utilisation de processus à comportement identique
  - Même si à certaines phases, des distinctions sont à faire

Présentation via un exemple

- Phases d'exécution
  - Création de l'anneau (partie non symétrique)
    - Maître prenant en charge la création, le programme principal
  - Circulation de messages sur l'anneau
    - Simulation d'un traitement dont un interlocuteur sait quand il est terminé
  - Terminer le système (partie non symétrique)
    - Maître prenant en charge la détection de la terminaison un gardien
      - Le programme principal n'a pas de points d'entrée...
- Ce que l'on va étudier
  - La structure du système
    - Avec des liens de communication supplémentaires



  - Le comportement (nœud 1 envoie deux messages aux autres nœuds)

Structure du programme

```
with Ada.Text_Io, Ada.Command_Line, Ada.Numerics.Discrete_Random;
use Ada.Text_Io, Ada.Command_Line;

procedure anneau is
   ------------------------------
   -- Elements utiles pour le programme
   ------------------------------

   ------------------------------
   -- Les interlocuteurs du systeme
   ------------------------------

   ------------------------------
   -- Body des interlocuteurs du systeme
   ------------------------------

   ------------------------------
   -- programme principal (le "lanceur")
   ------------------------------
begin
   -- son code...
end anneau;
```

Déclaration de types et de procédures utilitaires

Déclaration des tâches (sauf le «main») + anneau

Corps des tâches déclarées

Types et procédures utilitaires

```
-- Pour générer de l'indéterminisme
package Alea_int is new Ada.Numerics.Discrete_Random(Integer);
Graine :  Alea_int.Generator;

-- Pour avoir des traces propres (une colonne par tache)
procedure Display_Msg (Width : in Natural; S : in String) is
begin
   Put_Line ((1..Width => ASCII.ht) & S);
end Display_Msg;

-- Taille de l'anneau
Taille_Anneau : constant positive := Positive'Value (Argument(1));

-- Type permettant de coder un message
type Code_M is (Vers_Bas, Vers_Haut);
type Un_Message is record
   Dest : Positive;
   Code : Code_M;
end record;
```

Déclaration des tâches

```
-- Une tache correspondant a un noeud dans l'anneau
task type Noeud (Id : Positive) is
   entry Init (Pred, Succ : Positive);
   entry Recevoir (M : in Un_Message);
   entry Finir;
end Noeud;

-- Un pointeur sur la tache
type A_Noeud is access Noeud;
-- Construction d'un tableau de pointeurs sur les noeuds
Un_Anneau : array (1..Taille_Anneau) of A_Noeud;

-- Le gardien
task Gardien is
   entry Jai_Termine;
end Gardien;
```

Initialisation

Communication

Signalement (de terminaison)

Les interlocuteurs : le programme principal


```
-- Initialiser la graine du générateur aléatoire
Alea_int.reset (Graine);
-- Création des tâches
for I in Un_Anneau'Range loop
   Un_Anneau (I) := new Noeud (I);
end loop;
-- Initialisation des tâches
for I in Un_Anneau'Range loop
   if I = Un_Anneau'First then
      Un_Anneau (I).all.Init (Un_Anneau'Last, I+1);
   elsif I = Un_Anneau'Last then
      Un_Anneau (I).all.Init (I - 1, Un_Anneau'First);
   else
      Un_Anneau (I).all.Init (I - 1, I + 1);
   end if;
end loop;
-- On attend la fin...
```

Création d'une tâche-nœud

Cas spécial (voisins de 1)

Cas normal (voisins de 2..N-1)

Les interlocuteurs : le gardien

Attente des signaux de fin (sauf pour 1)

```
task body Gardien is
begin
   -- On compte les processus signalant leur terminaison
   -- sauf le 1 qui n'envoie pas de message au gardien
   for I in 2 .. Un_Anneau'Last loop
      accept Jai_Termine;
   end loop;
   -- On demande a tous les noeuds de se terminer
   for I in Un_Anneau'Range loop
      Un_Anneau(I).all.Finir;
   end loop;
end Gardien;
```

Les interlocuteurs: un nœud de l'anneau (1)


```
task body Noeud is
   P, S : Positive; -- indice des voisins
   Msg_Recu : Natural := 0; -- nombre de messages reçus
   -- Pour les tâches /= 1 (nombre de messages à recevoir)
   Cpt_Dest : Positive := 2;
   -- Pour la tâche 1 (stade de l'émission)
   Round_Msg : Positive := 1;
begin
   -- Phase d'initialisation
   accept Init (Pred, Succ : Positive) do
      P := Pred;
      S := Succ;
      Display_Msg (Id, Integer'Image (Id)
                    & " est initialise avec"
                    & Integer'Image (P) & " et"
                    & Integer'Image (S));
   end Init;
```

Rappel: chaque instance de nœud aura sa propre copie de ces variables => son contexte

Les interlocuteurs: un nœud de l'anneau (2)


```
-- Boucle principale de traitement
loop
   declare
      Loc_M : Un_Message;
   begin
      select
         -- On se termine sur demande du gardien
         accept Finir;
         Display_Msg (Id, "le gardien demande la fin de" &
                          Integer'Image (Id));
         exit;
      or
         -- On recoit un message
         accept Recevoir (M : in Un_Message) do
            Loc_M := M;
         end Recevoir;
         -- Il faut absolument faire les autres appels de point
         -- d'entrée en dehors de l'accept principal sinon on
         -- génère des blocages...
         ...
         code à suivre (page suivante)
```

Les interlocuteurs: un nœud de l'anneau (3)


```
-- Il faut absolument faire les autres appels de point
-- d'entrée en dehors de l'accept principal sinon on
-- génère des blocages...
if Loc_M.Dest = Id then
   Display_Msg (Id, "Message " & Code_M'Image (Loc_M.Code) &
                    " recu par" & Integer'Image(Id));
   Msg_Recu := Msg_Recu + 1;
   if Msg_Recu = 2 then
      Gardien.Jai_Termine;
   end if;
else
   Display_Msg (Id, "Message " & Code_M'Image (Loc_M.Code) &
                    " pour" & Integer'Image(Loc_M.Dest) &
                    " transmis par " & Integer'Image(Id));
   case Loc_M.Code is
      when Vers_Bas =>
         Un_Anneau(P).Recevoir (Loc_M);
      when Vers_Haut =>
         Un_Anneau(S).Recevoir (Loc_M);
   end case;
end if;
-- fin de la seconde alternative du select
```

Le message est pour le nœud

Le message n'est pas pour le nœud

Les interlocuteurs: un nœud de l'anneau (4)


```
else
   -- Si on n'a pas de message à transmettre et que l'on est le nœud 1
   if Id = 1 and then Cpt_Dest <= Un_Anneau'Last then
      if Round_Msg = 1 then
         select
            Un_Anneau(P).Recevoir((Cpt_Dest, Vers_Bas));
            Display_Msg (Id, "J'ai envoye un message a" &
                         Integer'Image(Cpt_Dest) & " Vers_Bas");
            Round_Msg := Round_Msg + 1;
         or
            delay Duration (Alea_int.random(Graine) mod 10)*0.1;
         end select;
      end if;
      if Round_Msg = 2 then
         select
            Un_Anneau(S).Recevoir((Cpt_Dest, Vers_Haut));
            Display_Msg (Id, "J'ai envoye un message a" &
                         Integer'Image(Cpt_Dest) & " Vers_Haut");
            Round_Msg := Round_Msg + 1;
         or
            delay Duration (Alea_int.random(Graine) mod 10)*0.1;
         end select;
      end if;
      if Round_Msg = 3 then
         Round_Msg := 1;
         Cpt_Dest := Cpt_Dest + 1;
      end if;
```

Tenter d'envoyer le message «vers le bas»

Tenter d'envoyer le message «vers le haut»

Passer au nœud suivant

Les interlocuteurs: un nœud de l'anneau (5)


```
                -- Si on n'a pas de message à transmettre et que l'on est le nœud N /= 1
                else
                   delay Duration(Alea_int.random(Graine) mod 10)*0.1;
                end if;
             end select;
          end;
       end loop;
    end;
```

Exécution (1)

- Pour un anneau a un nœud, le programme se bloque
  - Normal, un tel anneau n'a pas de sens!
    - Le nœud 2 n'existe pas

- Exécution

```
$ ./anneau 1
        1 est initialise avec 1 et 2
^C
```

Exécution (2)

Signalisation de la fin d'exécution au gardien

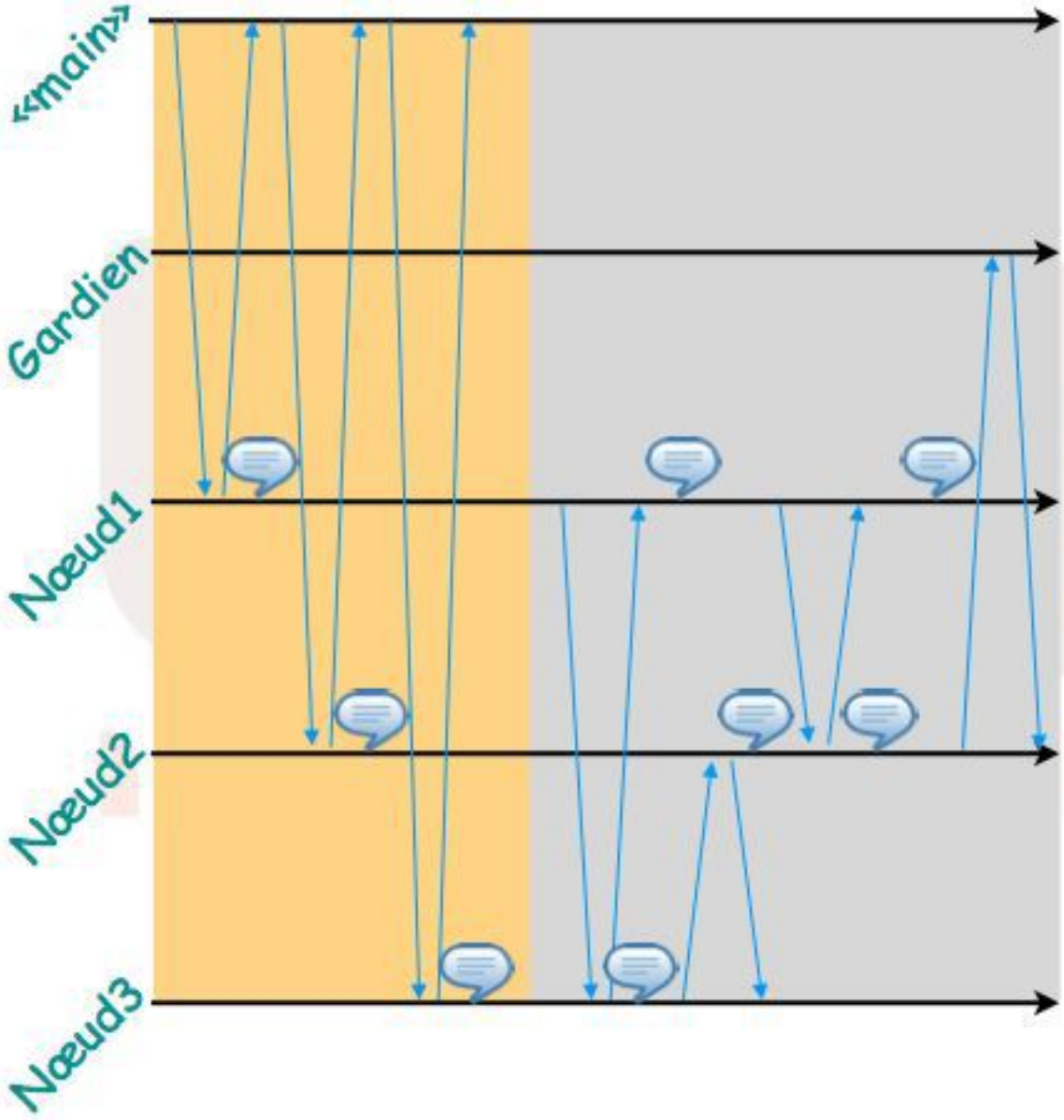
«lanceur» «nœud 1» «nœud 2» «gardien»

```
$ ./anneau 2
1 est initialise avec 2 et 2
2 est initialise avec 1 et 1
                    Message VERS_BAS recu par 2
          J'ai envoye un message a 2 Vers_Bas
                    Message VERS_HAUT recu par 2
          J'ai envoye un message a 2 Vers_Haut
                              le gardien demande la fin de 1
                              le gardien demande la fin de 2
```

Exécution (3)

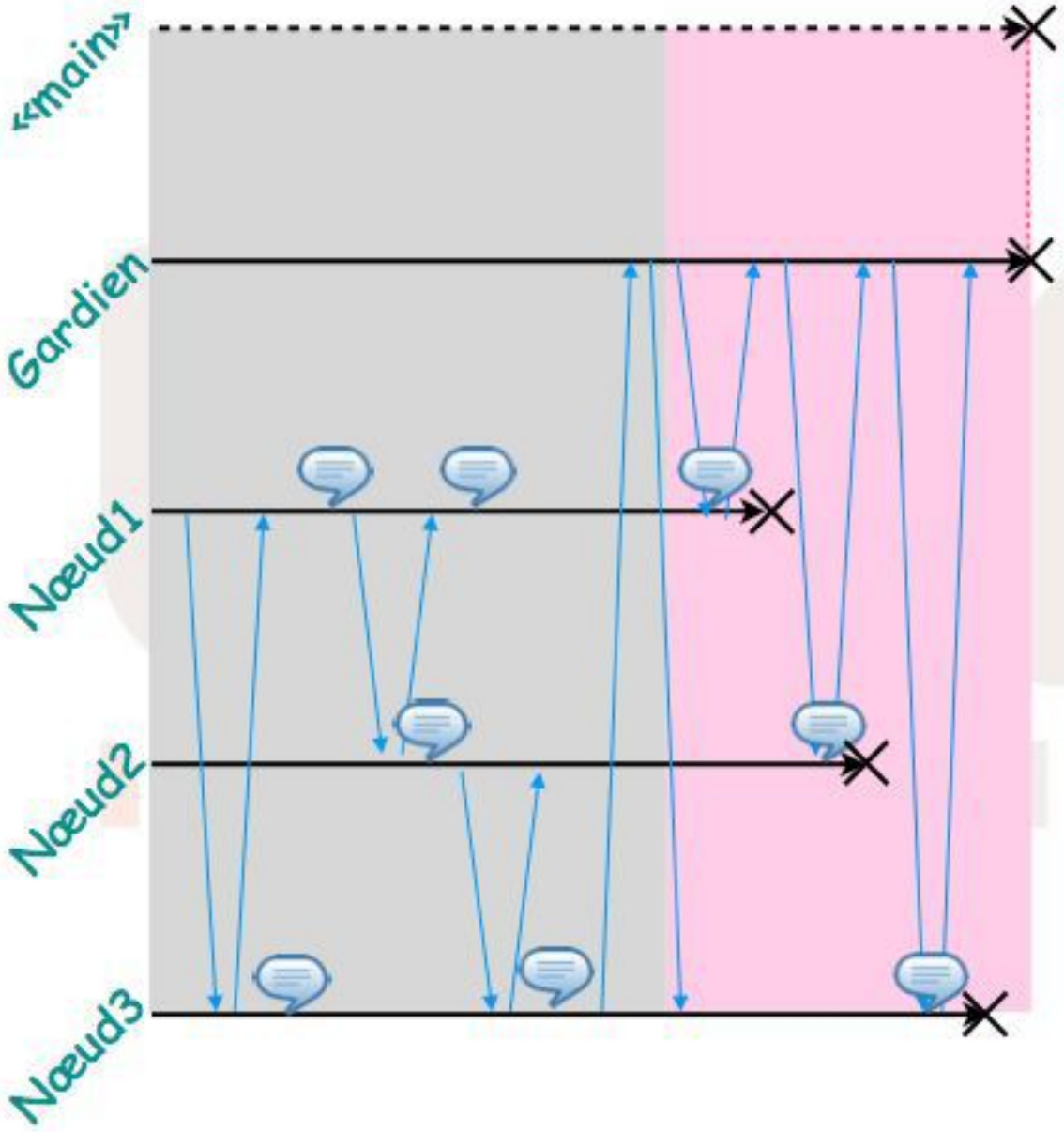

```
«lanceur»   «nœud 1» «nœud 2»  «nœud 3»  «nœud 4»  «gardien»
$ ./anneau 4
1 est initialise avec 4 et 2
2 est initialise avec 1 et 3
3 est initialise avec 2 et 4
4 est initialise avec 3 et 1
                                          Message VERS_BAS pour 2 transmis par  4
            J'ai envoye un message a 2 Vers_Bas
                      Message VERS_HAUT recu par 2
            J'ai envoye un message a 2 Vers_Haut
                                 Message VERS_BAS pour 2 transmis par  3
                                           Message VERS_BAS pour 3 transmis par  4
            J'ai envoye un message a 3 Vers_Bas
                      Message VERS_BAS recu par 2
                                 Message VERS_BAS recu par 3
                      Message VERS_HAUT pour 3 transmis par  2
            J'ai envoye un message a 3 Vers_Haut
                                 Message VERS_HAUT recu par 3
                                          Message VERS_BAS recu par 4
            J'ai envoye un message a 4 Vers_Bas
                      Message VERS_HAUT pour 4 transmis par  2
            J'ai envoye un message a 4 Vers_Haut
                                 Message VERS_HAUT pour 4 transmis par  3
                                           Message VERS_HAUT recu par 4
                                                     le gardien demande la fin de 1
                                                     le gardien demande la fin de 2
                                                     le gardien demande la fin de 3
                                                     le gardien demande la fin de 4
```

Un petit chronogramme (3 nœuds) Partie 1



1 est initialise avec 3 et 2
2 est initialise avec 1 et 3
3 est initialise avec 2 et 1
Message VERS_BAS pour 2 transmis par 3
J'ai envoye un message a 2 Vers_Bas
Message VERS_BAS recu par 2
Message VERS_HAUT recu par 2
J'ai envoye un message a 2 Vers_Haut

Un petit chronogramme (3 nœuds) Partie 2



Message VERS_BAS recu par 3
J'ai envoye un message a 3 Vers_Bas
Message VERS_HAUT pour 3 transmis par 2
J'ai envoye un message a 3 Vers_Haut
Message VERS_HAUT recu par 3
le gardien demande la fin de 1
le gardien demande la fin de 2
le gardien demande la fin de 3