NIVEAU D'INTERVENTION DE LA PROGRAMMATION CONCURRENTE

Une application se construit par étapes

1) CAHIER DES CHARGES + ANALYSE FONCTIONNELLE
=> organisation fonctionnelle
(QUE FAIRE)

2) ANALYSE OPERATIONNELLE + CHOIX DES UNITES LOGIQUES
=> architecture logique
(COMMENT FAIRE)

3) ANALYSE ORGANIQUE + CHOIX TECHNOLOGIQUES
=>architecture physique
(AVEC QUELS MOYENS)
architecture matérielle + carte de répartition du logiciel (placement)

processus séquentiel : Entité qui sert à représenter une unité active de l'architecture logique (au niveau de l'expression opérationnelle de l'application). Séquence particularisée d'actions programmées.

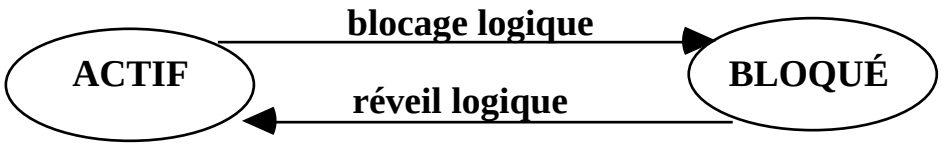
concurrence des processus : Ces processus sont concurrents dans le cadre de l'architecture

LE COMPORTEMENT OPÉRATIONNEL DE L'APPLICATION EST EXPRIMÉ PAR LE COMPORTEMENT CONCURRENTIEL DES PROCESSUS

ÉTATS D'UN PROCESSUS SÉQUENTIEL DANS L'ARCHITECTURE LOGIQUE,

ensemble concurrent de processus séquentiels

•états de synchronisation logique ou intrinsèque : BLOQUÉ <-> ACTIF,
bloqué en attente d'un événement programmé au niveau considéré
actif s'il n'y a pas de blocage logique



événements de l'architecture logique
(événements programmés à ce niveau)

ON IGNORE LES AUTRES ÉTATS

états technologiques introduits par le noyau du système qui doit allouer des ressources au processus
par exemple, états introduits pour une politique globale avec va et vient des processus

ACTIF : {prêt, élu, gelé} **BLOQUÉ : {suspendu, oisif}**

PARADIGMES DE LA CONCURRENCE

Briques de base pour toute étude, analyse ou construction de système ou d'application coopérative.
("concurrency design patterns")

PARADIGMES

• Exemples-type qui permettent de modéliser des classes de problèmes réels fréquemment rencontrés et présents à tous les niveaux dans les systèmes et dans les applications concurrentes.
• Acceptés par la communauté pour leur capacité à fournir des schémas de base de conception

PRINCIPAUX PARADIGMES DE LA CONCURRENCE

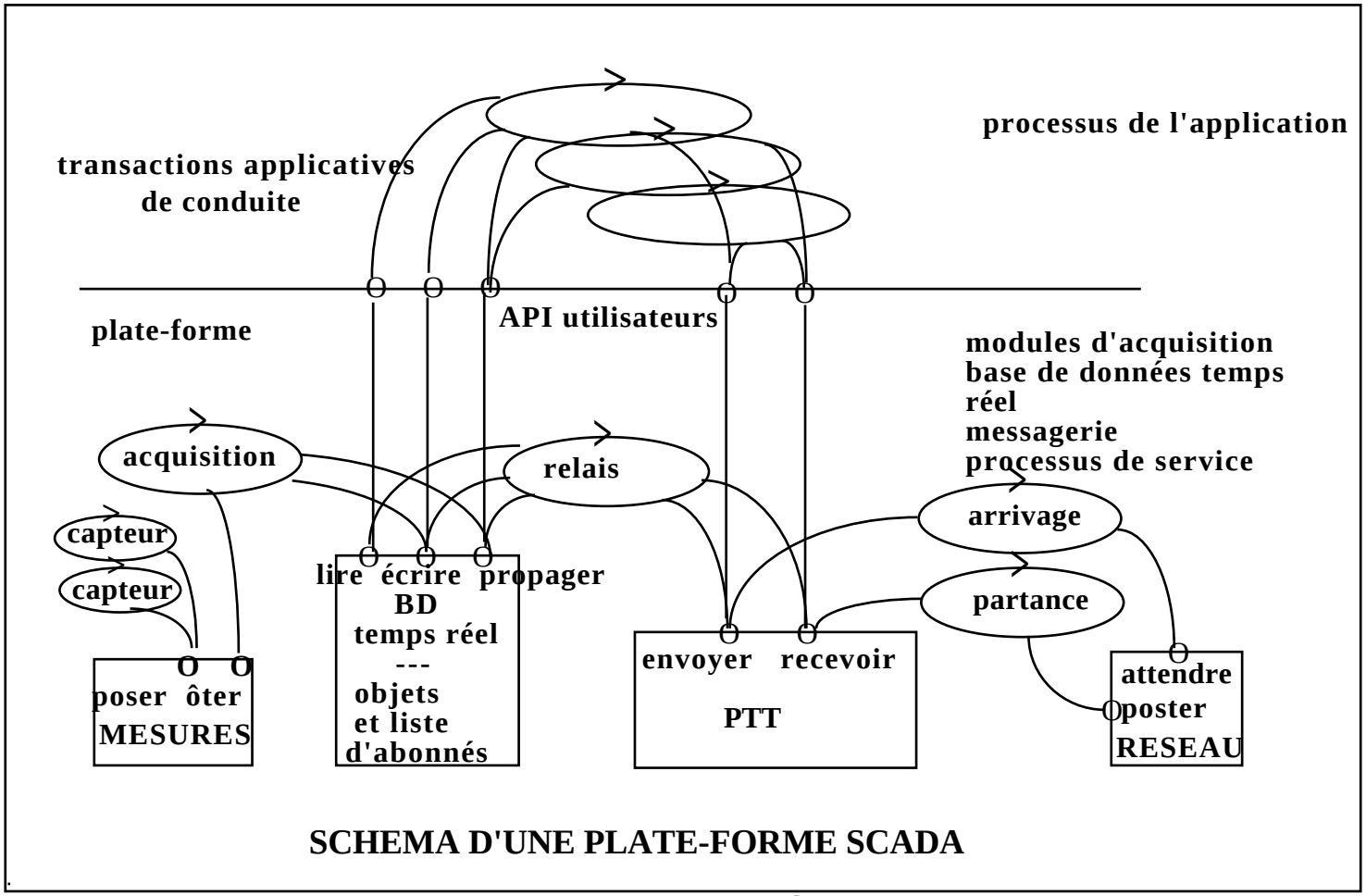
- **l'exclusion mutuelle** qui modélise l'accès cohérent à de ressource partagées,
- **la cohorte** qui modélise la coexistence d'un groupe de taille maximale donnée,
- **le passage de témoin** qui modélise la coopération par découpage des tâches entre les processus
- **les producteurs-consommateurs,** exemple qui modélise la communication par un canal fiable,
- **les lecteurs-rédacteurs** exemple qui modélise la compétition cohérente,
- **le repas des philosophes** qui modélise l'allocation de plusieurs ressources et l'interblocage.

ARCHÉTYPES (ou encore PATRONS)

• Solutions des paradigmes pouvant servir de schémas de programmation de composants concurrents.

La solution d'un paradigme est un archétype qui décrit un comportement acceptable pour l'ensemble des processus concurrents dans le cadre du problème à résoudre, et cela quelle que soit la nature des processus. L'architecture des systèmes et des applications l'utilise soit directement, soit par composition, soit en dérivant de nouvelles solutions à partir du modèle de base.

ÉTUDE DE CAS : UNE PLATE-FORME POUR L'ACQUISITION EN TEMPS RÉEL



SCHEMA D'UNE PLATE-FORME SCADA

paradigmes utilisés :
lecteurs-rédacteurs : base de données, modules d'acquisition
producteurs consommateurs : messagerie, réseau

UNE PLATE-FORME POUR L'ACQUISITION EN TEMPS RÉEL

CODE AVEC OBJETS OU PAQUETAGES (ARCHÉTYPES, COMPOSANTS)
COMPLÉMENT DU SCHÉMA AVEC PARADIGMES (lecteurs-rédacteurs, producteurs-consommateurs)

```
with PLATEFORME; use PLATEFORME;
procedure MAIN_SCADA is                    -- c'est le processus principal qui lance l'application
   NB_TACHES : constant INTEGER := 50;

   task type APPLICATIVE; MON_APPLICATION : array (1..NB_TACHES) of APPLICATIVE;
   task body APPLICATIVE is separate;
   -- décrit les actions des tâches applicatives; celles-ci utilisent les API de la plateforme
begin
   null;
end MAIN_SCADA ; -- cette application comporte (NB_TACHES + 1 + NB_CAPTEURS + 4) processus

      -- on définit l'interface fourni par la plateforme
package PLATEFORME is
   type ELEMENT;
   procedure API_LIRE(X : out ELEMENT);
   procedure API_ECRIRE(X : in ELEMENT);
   procedure API_PROPAGER(X : in ELEMENT);
   procedure API_ENVOYER(X : in ELEMENT);
   procedure API_RECEVOIR(X : out ELEMENT);
end PLATEFORME;
```

```
package body PLATEFORME is-- paquetage contenant des tâches et implémentant le service

   procedure API_LIRE(X : out ELEMENT) renames BD.LIRE(X);        -- pseudonyme, alias
   procedure API_ECRIRE(X : in ELEMENT) renames BD.ECRIRE(X);
   procedure API_PROPAGER(X : in ELEMENT) renames BD.PROPAGER(X);
   procedure API_ENVOYER(X : in ELEMENT) renames PTT.ENVOYER(X);
   procedure API_RECEVOIR(X : out ELEMENT) renames PTT.ENVOYER(X);

   package MESURE is          -- spécification d'un objet partagé entre processus
      procedure POSER(X : in ELEMENT);                              -- PARADIGME
      procedure OTER(X : out ELEMENT);                          -- LECTEURS-RÉDACTEURS
   end MESURE;

   package BD is              -- spécification d'un objet partagé entre processus
      procedure ECRIRE(X : in ELEMENT);                             -- PARADIGME
      procedure PROPAGER(X : in ELEMENT);                       -- LECTEURS-RÉDACTEURS
      procedure LIRE(X : out ELEMENT);
   end BD;

   package PTT is             -- spécification d'un objet partagé entre processus
      procedure ENVOYER(X : in ELEMENT);                            -- PARADIGME
      procedure RECEVOIR(X : out ELEMENT);                  -- PRODUCTEURS-CONSOMMATEURS
   end PTT ;

   package RESEAU is          -- spécification d'un objet partagé entre processus
      procedure POSTER(X : in ELEMENT);                             -- PARADIGME
      procedure ATTENDRE(X : out ELEMENT);                  -- PRODUCTEURS-CONSOMMATEURS
   end RESEAU ;

   task ACQUISITION; task RELAIS; task ARRIVAGE; task PARTANCE;

   task CAPTEUR; LES_CAPTEURS / array(1..NB_CAPTEUR) of CAPTEUR; -- déclaration des processus

   package body MESURE is separate; package body BD is separate;
   package body PTT is separate; package body RESEAU is separate;
   task body ACQUISITION is separate; task body RELAIS is separate;
   task body ARRIVAGE is separate; task body PARTANCE is separate; task body CAPTEUR is separate;

begin null;        -- initialisation des variables cachées du paquetage
end PLATEFORME; -- cette plateforme comporte (NB_CAPTEURS + 4) processus. Il n'y a pas de programme principal
```

UTILISATION DES PARADIGMES ET DES ARCHÉTYPES

ÉTUDE DE CAS : SERVICE AVEC RÉPONSES MULTIFORMES

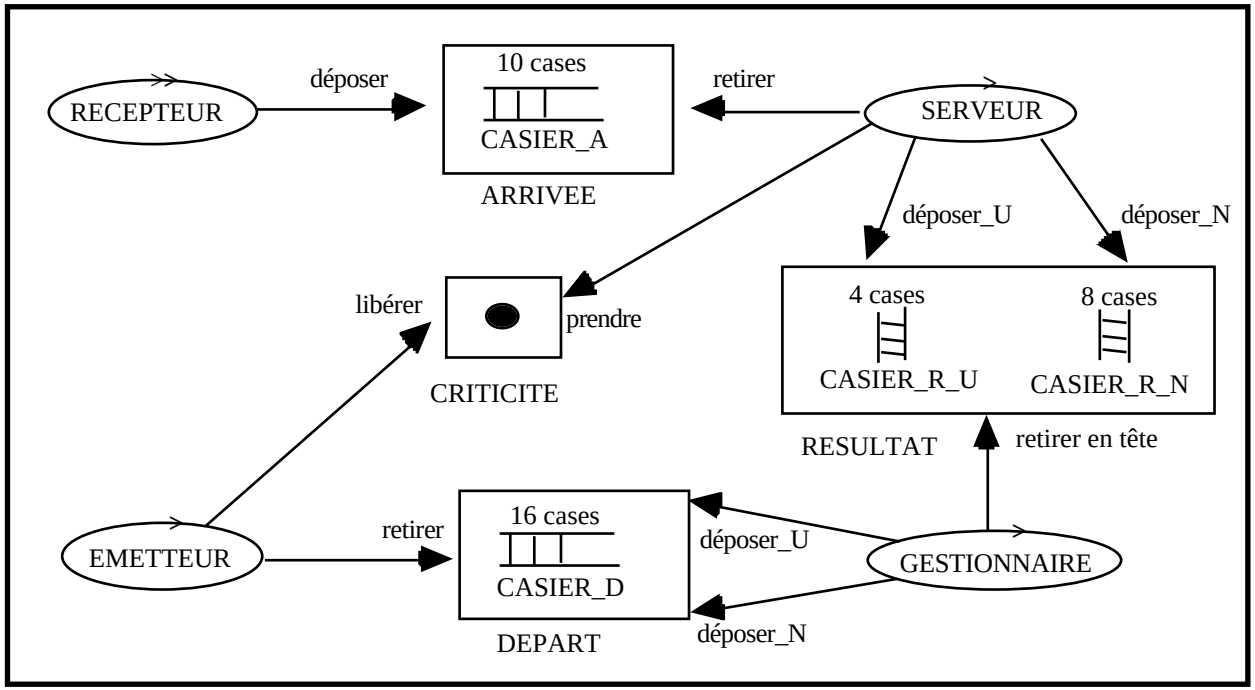


SCHÉMA AVEC PARADIGMES (exclusion mutuelle, producteurs-consommateurs)
COMPLÉMENT DU CODE AVEC OBJETS OU PAQUETAGES (ARCHÉTYPES, COMPOSANTS)

```
package COMMUN is                                   -- définition des types communs
   type REQUETE; type ELEMENT; type MESSAGE is record
                                                    URGENT, DERNIER : boolean;
                                                    VALEUR : ELEMENT; end record;
end COMMUN;

with COMMUN; use COMMUN;
procedure MAIN_SRI_B_SEPT_96 is                     --c'est le processus principal qui lance l'application

   package ARRIVEE is                               -- spécification d'un objet partagé entre processus
      procedure DEPOSER(X : in REQUETE);                                          -- PARADIGME
      procedure RETIRER(X : out REQUETE);                                -- PRODUCTEURS-CONSOMMATEURS
   end ARRIVEE;

   package RESULTAT is                              -- spécification d'un objet partagé entre processus
      procedure DEPOSER_U(X : in MESSAGE);                                        -- PARADIGME
      procedure DEPOSER_N(X : in MESSAGE);                               -- PRODUCTEURS-CONSOMMATEURS
      procedure RETIRER(X : out MESSAGE);
   end RESULTAT;

   package DEPART is                                -- spécification d'un objet partagé entre processus
      procedure DEPOSER_U(X : in MESSAGE);                                        -- PARADIGME
      procedure DEPOSER_N(X : in MESSAGE);                               -- PRODUCTEURS-CONSOMMATEURS
      procedure RETIRER(X : out MESSAGE);
   end DEPART;

   package CRITICITE is                             -- spécification d'un objet partagé entre processus
      procedure PRENDRE;                                                          -- PARADIGME
      procedure LIBERER;                                                 -- EXCLUSION MUTUELLE
   end CRITICITE;

   task RECEPTEUR, task SERVEUR; task GESTIONNAIRE; task EMETTEUR; -- déclaration des processus

   package body ARRIVEE is separate; package body RESULTAT is separate; package body DEPART is separate;
   package body CRITICITE is separate; task body RECEPTEUR is separate; task body SERVEUR is separate;
   task body GESTIONNAIRE is separate; task body EMETTEUR is separate;

begin    null;          - les 4 tâches déclarées ci-dessus démarrent à ce point et l'application comporte 5 processus
end MAIN_SRI_B_SEPT96;  -- le seul rôle de ce programme principal est de créer les objets et les tâches
```

```
separate(MAIN_SRI_B_SEPT96) task body RECEPTEUR is-- expression de l'activité du processus
      R : REQUETE;
   begin
      loop
         attendre_requête_du_réseau(R);  ARRIVEE.DEPOSER(R);
      end loop;                  -- boucle infinie car processus cyclique
   end RECEPTEUR;
```

```
separate(MAIN_SRI_B_SEPT96) task body SERVEUR is
      R : REQUETE; M : MESSAGE; DERNIER : boolean; URGENT : boolean;
   begin
      loop
         CRITICITE.PRENDRE;
         ARRIVEE.RETIRER(R); ENCORE := true;
         while ENCORE loop
            continuer_à_traiter_la_requête(DERNIER, URGENT, M);  -- chaque étape élabore M, DERNIER et URGENT
            M.DERNIER := DERNIER; -- le message M est ou non URGENT, et est ou non le dernier
            if URGENT then
               M.URGENT := true; RESULTAT.DEPOSER_U(M);
            else
               M.URGENT := false; RESULTAT.DEPOSER_N(M);
            end if;
         end loop;          -- boucle contrôlée par le while
      end loop;                -- boucle infinie car processus cyclique
   end SERVEUR;
```

```
separate(MAIN_SRI_B_SEPT96) task body GESTIONNAIRE is M : MESSAGE;
   begin loop
            RESULTAT.RETIRER(M);  traitement_local(M);
            if M.URGENT then DEPART.DEPOSER_U(M); else DEPART.DEPOSER_N(M); end if;
         end loop;               -- boucle infinie car processus cyclique
   end GESTIONNAIRE;
```

```
separate(MAIN_SRI_B_SEPT96) task body EMETTEUR is  M : MESSAGE;
   begin loop
            DEPART.RETIRER(M); envoyer_sur_le_réseau_et_attendre_acquittement(M);
            if M.DERNIER then CRITICITE.LIBERER; end if;
         end loop;               -- boucle infinie car processus cyclique
   end EMETTEUR;
```

EXCLUSION MUTUELLE ET COHERENCE DE SUITES D'ACTION CONCURRENTES

EXEMPLE 1 : des processus P et Q produisent des suites de lignes pour une imprimante partagée

P1 P2 Q1 P3 Q2 P4 P5 Q3 P6 Q4 Q5 --mauvais
Q1 Q2 Q3 Q4 Q5 P1 P2 P3 P4 P5 P6 --bon
P1 P2 P3 P4 P5 P6 Q1 Q2 Q3 Q4 Q5 --bon

EXEMPLE 2 : deux processus P et Q modifient une variable commune COMPTE_CLIENT

{COMPTE_CLIENT = 0}	**P1 Q1 Q2 P2 Q3 P3**	**{COMPTE_CLIENT = 1} --mauvais**
{COMPTE_CLIENT = 0}	**Q1 Q2 P1 P2 P3 Q3**	**{COMPTE_CLIENT = 1} --mauvais**
{COMPTE_CLIENT = 0}	**Q1 Q2 Q3 P1 P2 P3**	**{COMPTE_CLIENT = 2} --bon**
{COMPTE_CLIENT = 0}	**P1 P2 P3 Q1 Q2 Q3**	**{COMPTE_CLIENT = 2} --bon**

EXEMPLE 3: plusieurs processeurs communiquent via une voie unique, un bus de communication, par exemple Ethernet. Il faut éviter la cacophonie qui résulterait du mélange de messages.
EXEMPLE 4: SNCF avec section de ligne de chemin de fer à voie unique. Trains dans un sens ou dans l'autre, mais pas dans les deux sens en même temps.

TERMINOLOGIE

RESSOURCE CRITIQUE : entité partagée dont l'utilisation ne doit être faite que par un processus à la fois -en accès exclusif- : imprimante, COMPTE_CLIENT, bus ethernet, section à voie unique.
EXCLUSION MUTUELLE: condition de fonctionnement garantissant à un processus l'accès exclusif à une ressource pendant une suite d'opérations avec cette ressource
SECTION CRITIQUE : séquence d'actions d'un processus pendant lesquelles il doit être le seul à utiliser la ressource critique

P1 P2 P3 P4 P5 P6 ou Q1 Q2 Q3 Q4 Q5 de l'exemple 1
P1 P2 P3 ou Q1 Q2 Q3 de l'exemple 2

REMARQUE : une exclusion mutuelle, une section critique pour chaque ressource critique utilisée

EXCLUSION MUTUELLE : EXEMPLE 1

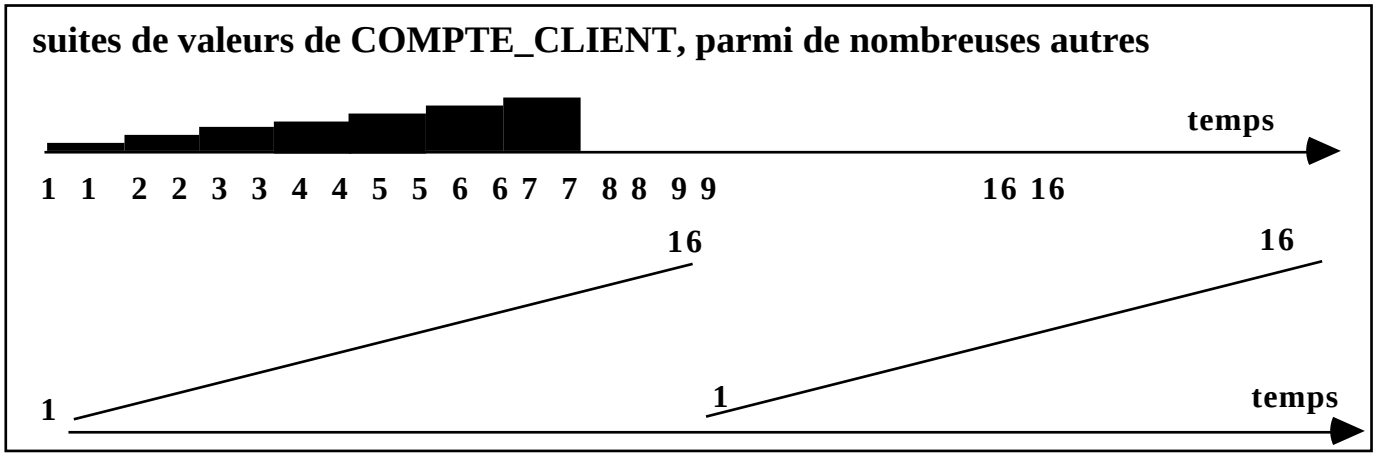
```
procedure LES_MAUX_DE_L_IMPRESSION is
  pragma TIME_SLICE (0.001);
  task P is
  begin
    PUT ("Terre arable "); PUT ("du songe! "); PUT ("Qui parle "); PUT ("de batir? "); NEW_LINE;
    PUT ("J'ai vu la terre "); PUT ("distribuee "); PUT ("en de vastes espaces "); NEW_LINE;
    PUT ("et "); PUT ("ma pensee "); PUT ("n'est point distraite ");PUT ("du navigateur."); NEW_LINE;
     PUT ("                    "); PUT ("Saint-John "); PUT ("Perse"); NEW_LINE(2);
  end P;

  task Q is
  begin
    PUT ("L'homme "); PUT ("est capable de faire "); PUT ("ce qu'il est "); PUT ("incapable d'imaginer. ");
    NEW_LINE;
    PUT ("Sa tete "); PUT ("sillonne "); PUT ("la galaxie "); PUT ("de l'absurde. "); NEW_LINE;
     PUT ("                    "); PUT ("Rene "); PUT ("Char"); NEW_LINE(2);
  end Q;

begin
  PUT ("IMPRESSION "); PUT ("DE DEUX POEMES "); PUT ("CONTEMPORAINS");
  NEW_LINE (2);
 end LES_MAUX_DE_L_IMPRESSION;
```

Une impression obtenue parmi d'autres, toutes aussi probables :

```
L'homme est capable de faire ce qu'il est incapable d'imaginer.
Terre arable IMPRESSION Sa tete du songe! DE DEUX POEMES sillonne Qui parle CONTEMPORAINS
de batir? la galaxie


de l'absurde J'ai vu la terre distribuee                    Rene en de vastes espaces

Char et ma pensee n'est point distraite du navigateur
                        Saint-John Perse
```

EXCLUSION MUTUELLE : EXEMPLE 2

```
procedure ACCES_BRUTAL is
                    COMPTE_CLIENT : Integer := 0;        -- variable commune

  task P is                                          •        task Q is
    X : INTEGER := 0;  -- variable locale à P        •          Y : INTEGER := 0;  -- locale à Q
  begin                                              •          begin
    for I in 1 .. 16 loop                            •            for I in 1 .. 16 loop
      X := COMPTE_CLIENT;     -- P1                  •              Y := COMPTE_CLIENT;     -- Q1
      X := X + 1;                        -- P2       •              Y := Y + 1;             -- Q2
      COMPTE_CLIENT := X;     -- P3                  •              COMPTE_CLIENT := Y;     -- Q3
    end loop;                                        •            end loop;
  end P;                                             •          end Q;

                    -- les deux processus P et Q en plus du programme principal ACCES_BRUTAL
begin
        null;  -- cette partie peut servir à initialiser des variables globales
    end ACCES_BRUTAL;
```



ENCORE UN EXEMPLE POUR L'EXCLUSION MUTUELLE

NICOLAS et ROSE, auditeurs au CNAM Paris,

• ont un compte bancaire joint : CC **• et ont chacun une carte bancaire donnant accès à ce compte**

LEUR BANQUE A INSTALLE DES GAB :

• C : consultation du compte **• R : retrait avec mise à jour du solde**

OPERATIONS CONCURRENTES AVEC LA BANQUE

• C1 : Nicolas consulte CC depuis le GAB Saint-Martin
• R1 : Nicolas retire 800 F ssi le compte est alimenté (CC >= 800 F)

• C2 : Rose consulte CC depuis le GAB Montgolfier
• R2 : Rose retire 600 F ssi le compte est alimenté (CC >= 600 F)

CONCURRENCE MAL GEREE (sans exclusion mutuelle)

{CC = 1000 F} C1; C2; R1; R2 {CC = -400 F donc découvert}
{CC = 1000 F} C1; C2; R2; R1 {CC = -400 F donc découvert}

CONCURRENCE BIEN GEREE (avec exclusion mutuelle)

{CC = 1000 F} C2; R2; C1; R1 {CC = 400 F}
{CC = 1000 F} C1; R1; C2; R2 {CC = 200 F}

RESSOURCE CRITIQUE : CC
SECTIONS CRITIQUES POUR ACCES A CC : C1; R1; ou C2; R2

EXCLUSION MUTUELLE

HYPOTHÈSES

H1: Les vitesses relatives des processus sont quelconques.
H2: Les priorités ou les droits des processus sont quelconques.
H3: Tout processus sort de sa section critique au bout d'un temps fini; en particulier ni panne ni blocage perpétuel ne sont permis en section critique.

SPÉCIFICATION DE COMPORTEMENT

C1: Un processus au plus en section critique. Peu importe l'ordre d'accès.
C2: Pas d'interblocage actif ou passif. Si aucun processus n'est en section critique et que plusieurs processus attendent d'entrer dans leur section critique, alors l'un d'eux doit nécessairement y entrer au bout d'un temps fini. (contrexemple: déclaration et politesse)
C3: Un processus bloqué en dehors de section critique, en particulier un processus en panne, ne doit pas empêcher l'entrée d'un autre processus dans sa section critique. (contrexemple: accès à l'alternat)
C4: La solution ne doit faire d'hypothèse ni sur les vitesses, ni sur les priorités relatives des processus. De ce point de vue la solution doit être symétrique.

RÈGLES D'ATTENTE--PARFOIS CONTRADICTOIRES

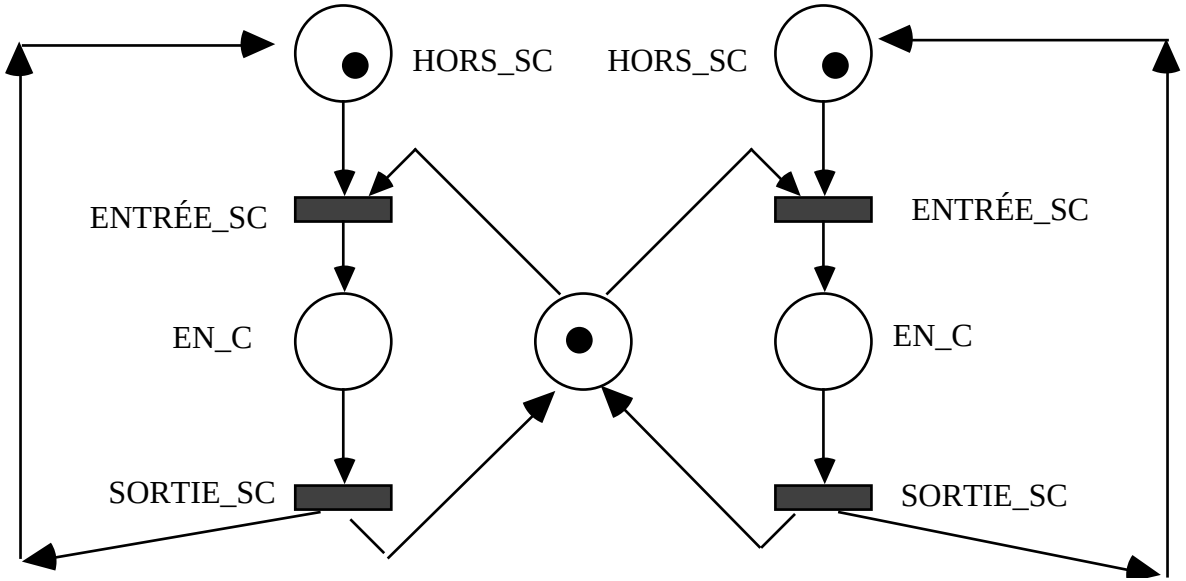
R1: Pas de coalition voulue ou fortuite entrainant la famine d'un processus. Aucun processus qui demande à entrer en section critique ne doit attendre indéfiniment d'y entrer. C'est la propriété d'équité. (Danger présent avec des priorités fixes)
R2: Le processus le plus prioritaire du système entre en premier en section critique
R3: Pas de règle particulière d'équité ou de temps de réponse. C'est le cas le plus fréquent
REMARQUE 1. Toute solution viable doit supposer le respect permanent de l'hypothèse H3 ou plutôt en reconstituer les conditions, en cas de panne matérielle ou logicielle. Ce rôle est assuré par le noyau du système. Par exemple, les fichiers ouverts par un programme utilisateur qui est arrêté pour cause d'erreur ou de dépassement de temps doivent être fermés. Quand le noyau tombe en panne, le système est aussi en panne.
REMARQUE 2. Il est souhaitable que l'exécution d'une section critique soit traitée comme une action atomique: elle est exécutée soit entièrement, soit pas du tout.

SPÉCIFICATION FORMELLE DE L'EXCLUSION MUTUELLE

1. SCHÉMA D'EXECUTION

Protocole d'initialisation
procédures de contrôle: ENTREE_SCi; section_critique_i; SORTIE_SCi;

2. RÉSEAU DE PETRI POUR DEUX PROCESSUS



COHORTE OU GROUPE DE TAILLE MAXIMALE DONNÉE

LA SITUATION CONCRËTE

Une procédure (une méthode) exécutable par plusieurs processus concurrents, N au plus
Un service multiprogrammé avec au plus N serveurs coopératifs
Partage d'une ressource à N points d'accès
Partage de N ressources banalisées d'une même classe

HYPOTHÈSES

H1: Les vitesses relatives des processus sont quelconques.
H2: Les priorités ou les droits des processus sont quelconques.
H3: Tout processus quitte la cohorte au bout d'un temps fini.

SPÉCIFICATION DE COMPORTEMENT

N processus au plus, N fixé, peuvent coopérer de façon asynchrone pour :

- **se répartir une tâche ou un service à fournir**
- **partager une ressource banalisée en N exemplaires**

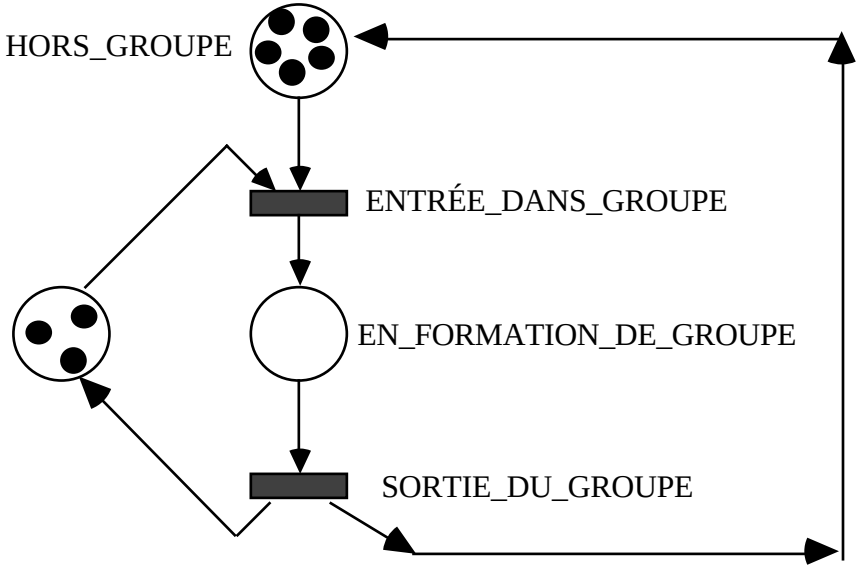
Remarque : avec N = 1, on a l'exclusion mutuelle

SPÉCIFICATION DE LA COHORTE

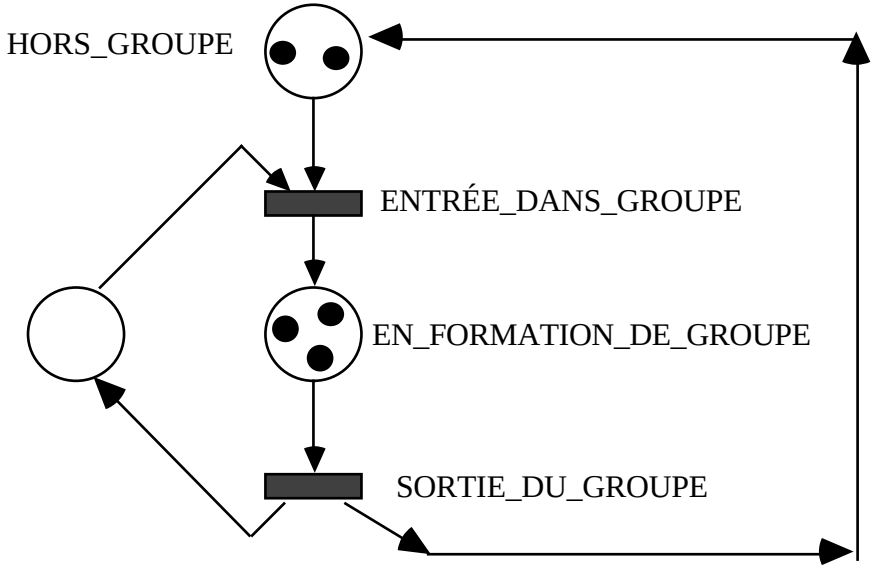
1. SCHÉMA D'EXECUTION

Protocole d'initialisation
procédures de contrôle: ENTREE_DANS_GROUPE; action coopérative; SORTIE_DU_GROUPE;

2. RESEAU DE PETRI POUR GROUPE DE TROIS PROCESSUS AU PLUS



groupe vide



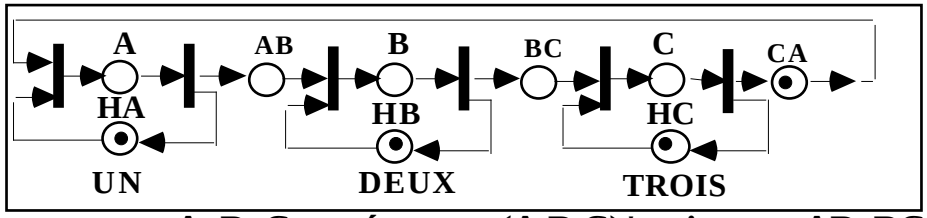
groupe plein

groupe des serveurs coopératifs ou groupe des utilisateurs qui ont reçu une ressource banalisée

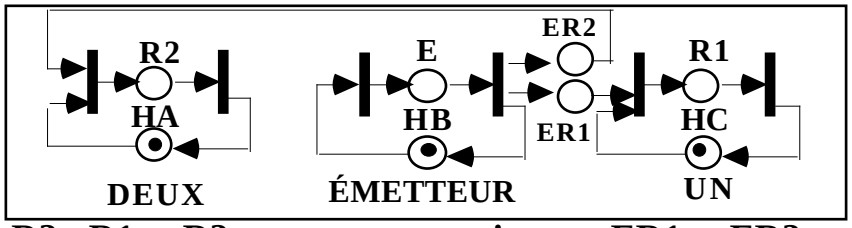
PASSAGE DE TÉMOIN

coopération par division du travail entre les processus
Envoi d'un signal, témoin de la fin d'une action.

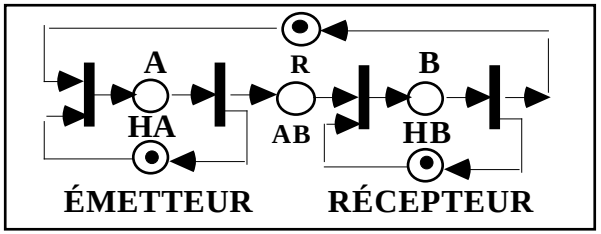
Signaux mémorisés ou non. Point à point ou diffusion vers p.



1) Actions : HA, HB, HC concurrentes ; A, B, C en séquence (A,B,C)* ; signaux AB, BC, CA



2) Précédences : E -> R1 et E -> R2 ; R1 et R2 concurrentes ; signaux ER1 et ER2



3) Envoi du droit d'accès à une ressource en exclusion mutuelle
Utilisation en section critique durant A puis durant B ; signal AB ; B libère la section critique