Master 1 en informatique, UBO

Algorithmes et Systèmes distribués

http://wsn.univ-brest.fr/moodle

Bernard Pottier
pottier@univ-brest.fr
http://wsn.univ-brest.fr/pottier

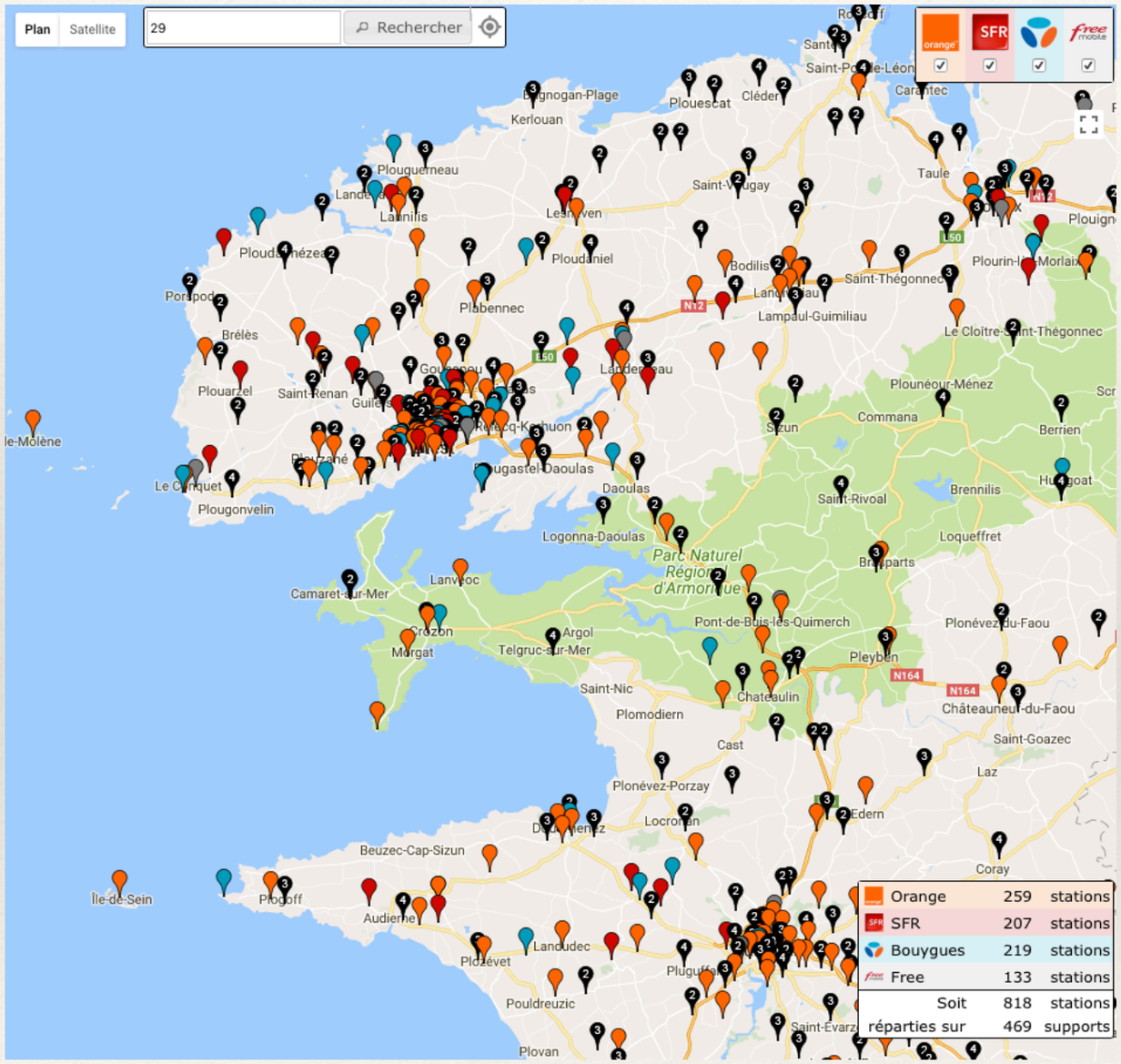
Présentation du domaine

- Omniprésent, et encore en expansion
- Exemples de systèmes distribués
  - réseau internet : routeurs + transmission longue distance + réseaux locaux
  - réseau téléphonique, distribution de l'électricité, contrôle aérien, chemin de fer, transports, environnement
  - réseaux sociaux

Notions fondamentales

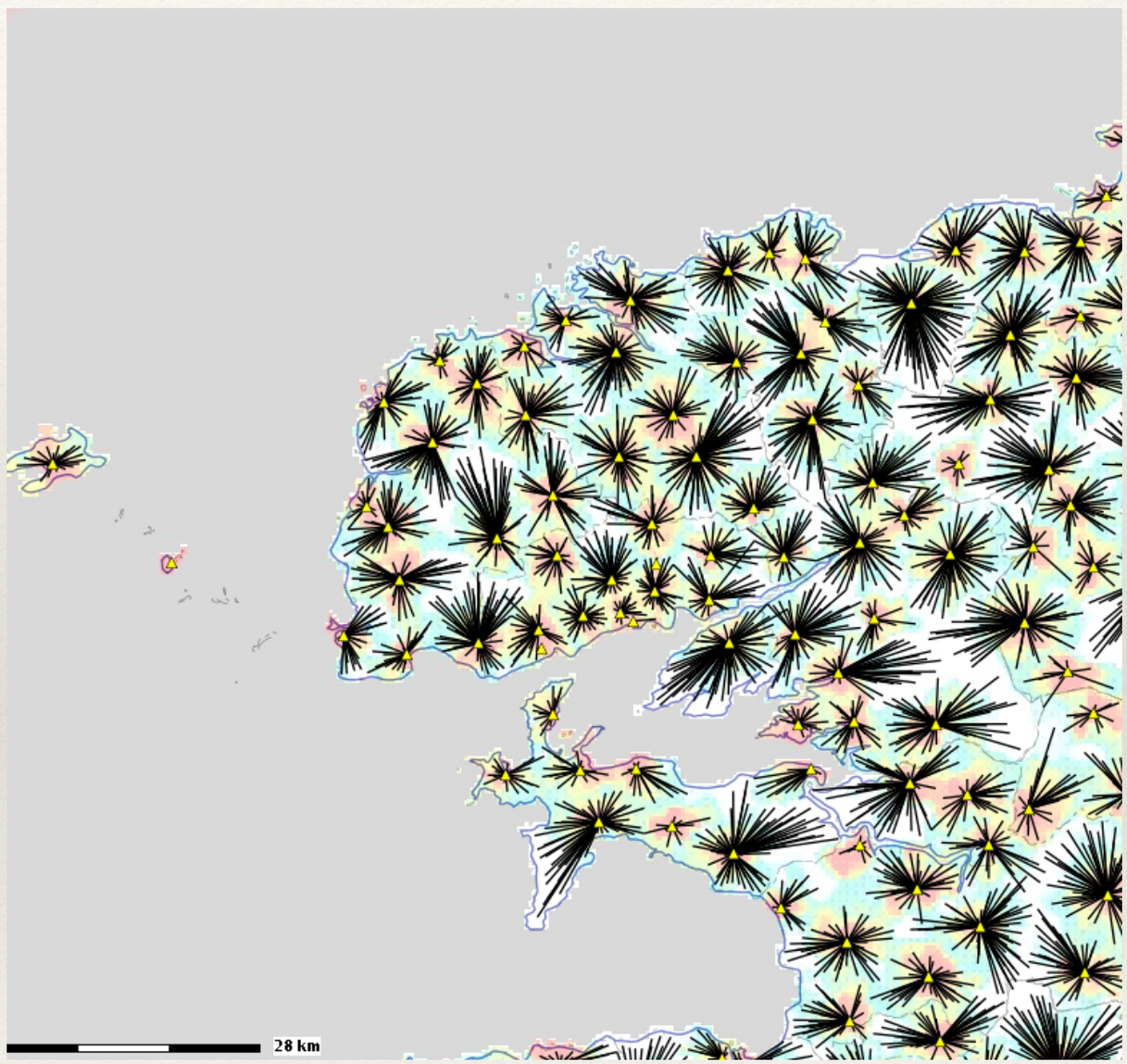
- **Noeuds**, machines, systèmes, processus
  - entités ayant un comportement local et collectif
- **Communications**
  - liens matériels, liens radio, influences physiques:
    - dépendances abstraites (circulation de l'information)
    - dépendances physiques (l'eau coule)
- **Systèmes:** Assemblage de processus et de communications

Réseau cellulaire, réseau ADSL



couverture-mobile.fr

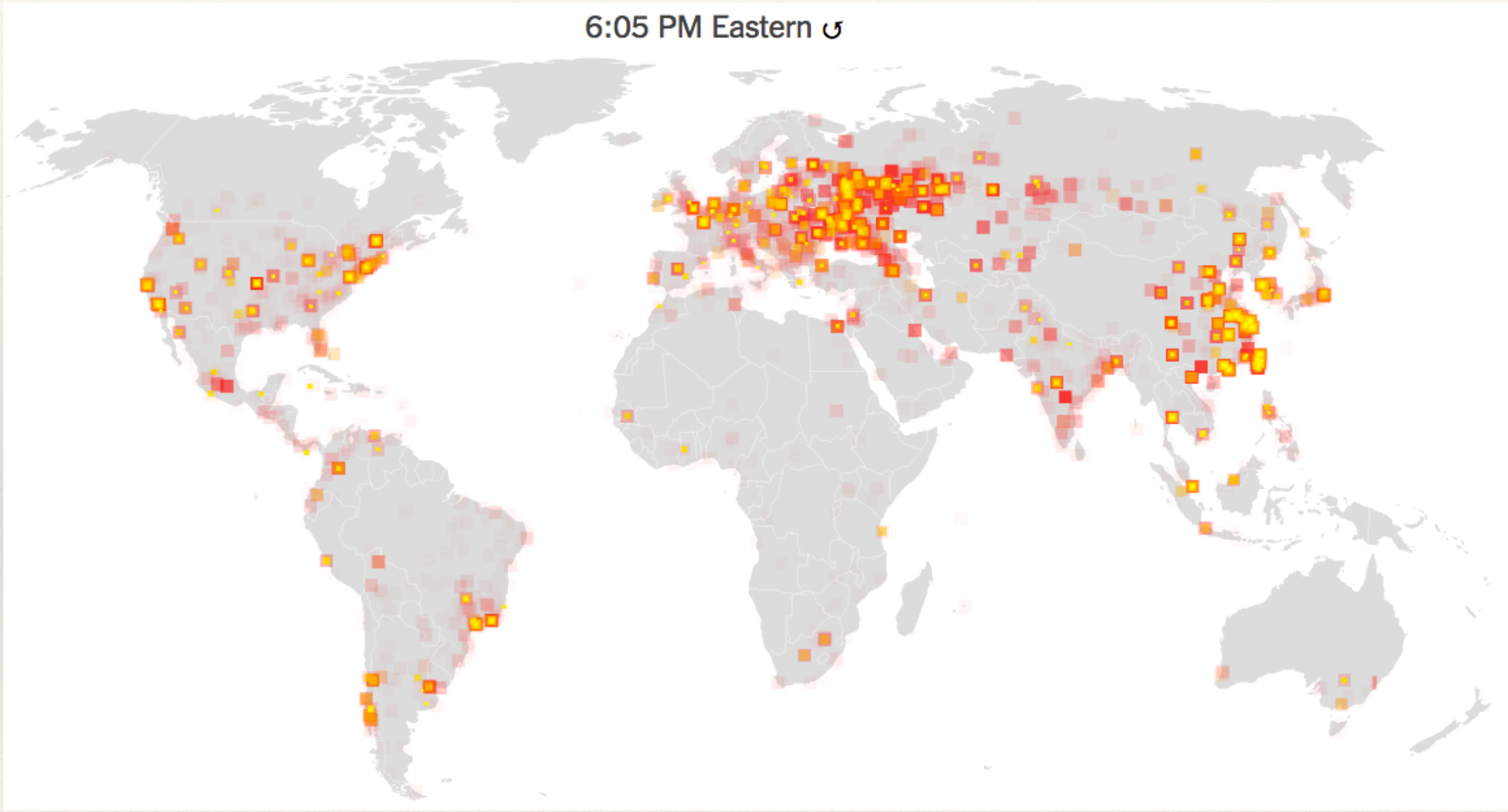
Antennes de réseaux cellulaires



cartelie couverture ADSL

NRA: Noeuds Raccordements abonnés

Réseau abstrait: communautés



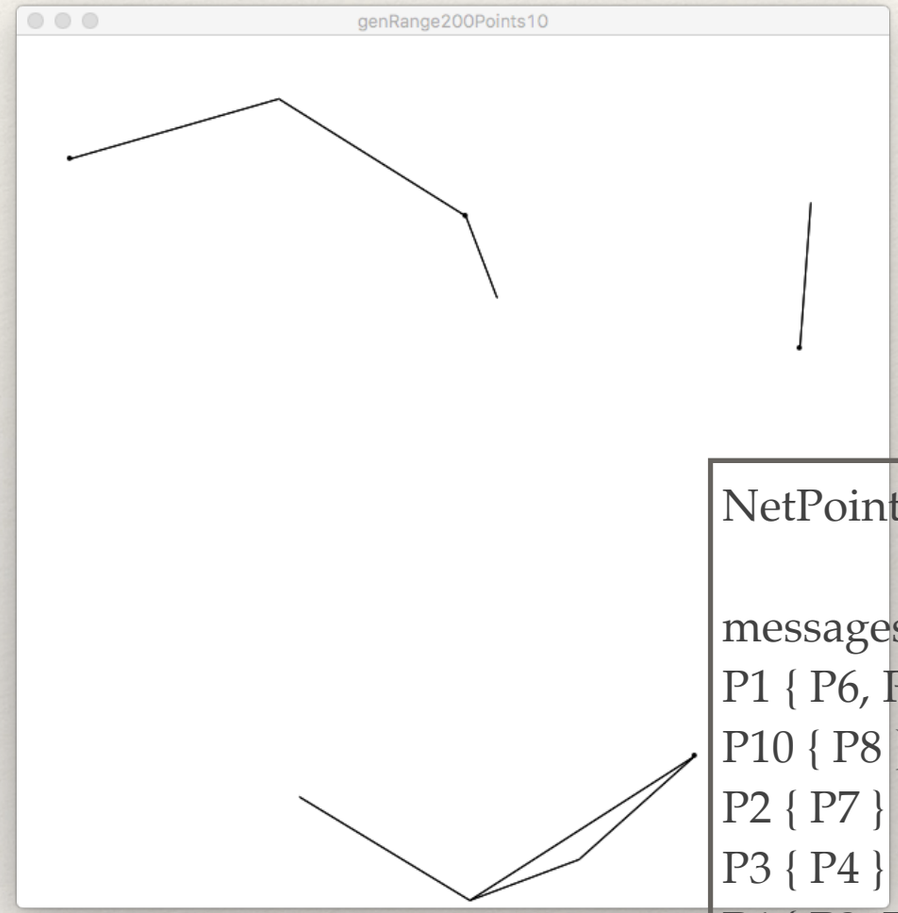
NY Times, Ransomware

Virtuel : réseau dynamique créé par un virus

Pourquoi le *distribué* est particulier ?

- **Inconnues prédominantes:**
  - Topologie, nombre de partenaires
- **Communications**
  - voisins directs ? ou indirects ? à quelle « distance » ?
  - topologie particulière (grille, anneau), ou aléatoire ?
  - séquencement : ordre des messages perturbé ?
  - capacité et rétention dans le réseau (tampons ? latences ?)
  - pannes, attaques malicieuses ?

Présentation des architectures

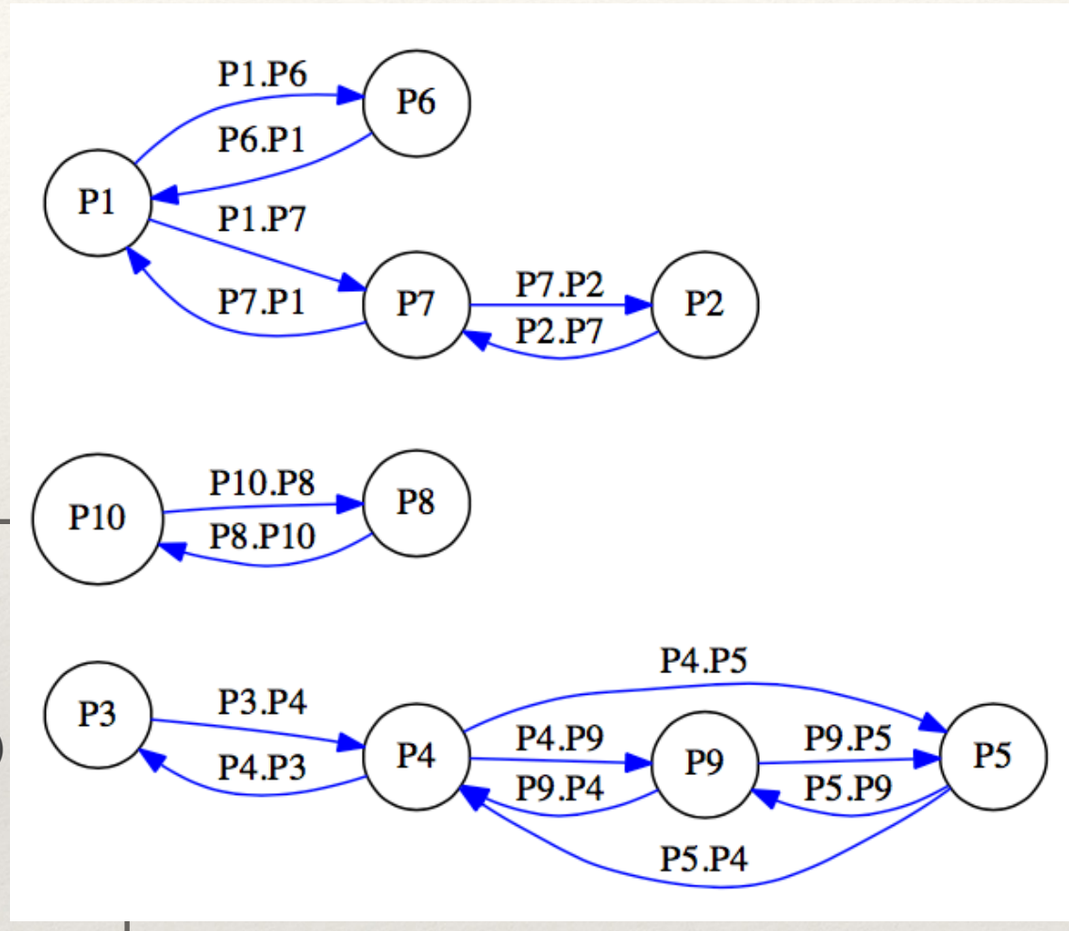


Graphique

NetPoints10

messages none defined.
P1 { P6, P7 } Node (192 @ 46) (200)
P10 { P8 } Node (574 @ 229) (200)
P2 { P7 } Node (352 @ 192) (200)
P3 { P4 } Node (207 @ 558) (200)
P4 { P3, P5, P9 } Node (332 @ 634) (200)
P5 { P4, P9 } Node (412 @ 604) (200)
P6 { P1 } Node (39 @ 90) (200)
P7 { P1, P2 } Node (329 @ 132) (200)
P8 { P10 } Node (582 @ 122) (200)
P9 { P4, P5 } Node (497 @ 528) (200)

Textuelle



Logique

Architectures et algorithmes (1)

Algorithme distribué:

- Description ***locale*** d'un traitement utilisant
  - des ***données locales,***
  - des données partagées *par communications*
- L'algorithme transforme un ***état local,*** et produit des données communiquées aux voisins immédiats

Exemple d'algorithme: calcul du maximum d'une valeur

- La composition des comportements permet de synthétiser un état général *indépendamment* de l'architecture, produisant une valeur *connue de tous*

Architectures et algorithmes (2)

Caractéristique de l'algorithme distribué:

- La composition des comportements permet de synthétiser un état général indépendamment de l'architecture, peut-être une valeur connue de tous
- On ne peut pas accéder à l'état des autres noeuds
- On ne peut que compter sur les communications pour faire progresser l'état du système

Spécification dans un langage

Architecture

- Description syntaxique de l'organisation du réseau: processus et liens de communication

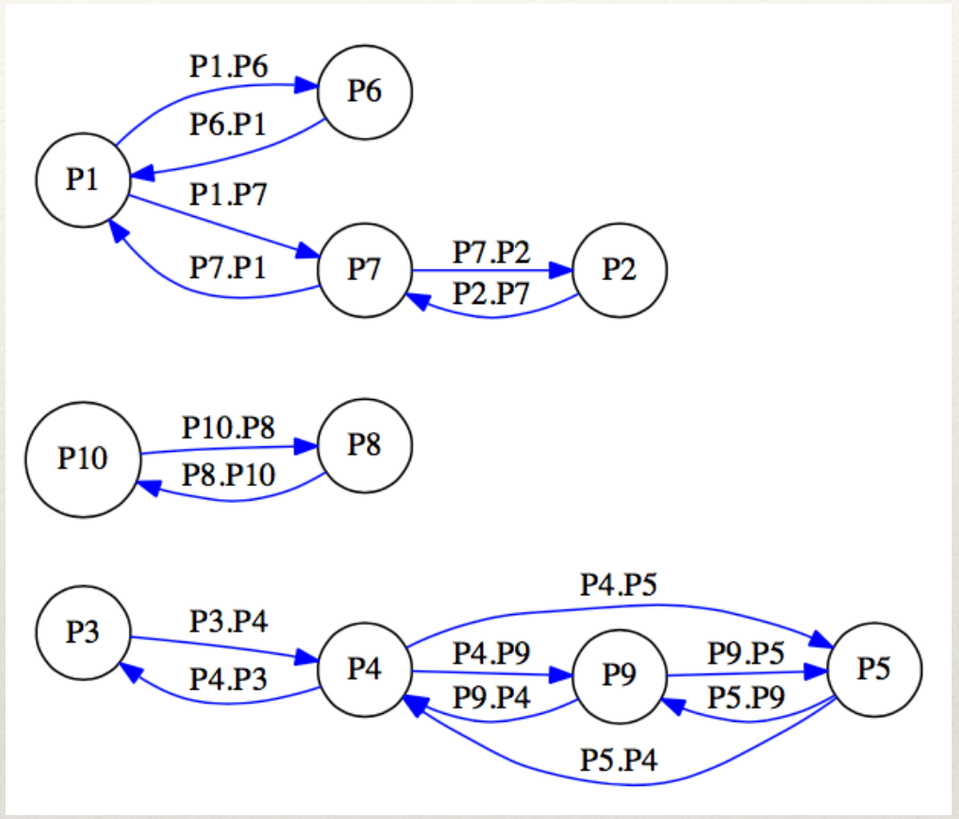
Algorithme

- Programme opérant sur les liens entrant et sortant, et opérant sur l'état local

Exemple de spécification

Processus

- entités exécutant le programme Node
- P1, P2, .. P10



Canaux

- communication directionnelle bloquante de processus à processus
- P1.P6, P6.P1, P1.P7, P7.P1, P2.P7, P7.P2

Langage de spécification: Occam

- Notion de processus communicants, constructions parallèles (PAR)
- Canaux bloquant avec deux opérations in (?) et out (!)
- Langage structuré, algorithmique, proche de Pascal
- Compilateur de l'Université de Kent (KROC)
- Installable aisément sur Linux et MacOSX

https://github.com/concurrency/kroc

Support du cours

- Moodle pour les planches de CM, les exercices, et les corrigés
- TP en salles de machine du département d'informatique
- Manuel en ligne disponible,
- notice d'installation.

https://wsn.univ-brest.fr/moodle

Motivation du modèle synchrone

- Dans un système distribué, il est difficile, ou impossible, de disposer d'une horloge commune.
- Les algorithmes parallèles sont plus facilement exprimés et vérifié si on dispose d'une référence de temps
- Une solution est d'induire cette référence des communications effectuées entre les processus
- Le *modèle synchrone* est général et englobe les exécutions sur horloge commune.

Systèmes distribués

Bernard Pottier

[1]LabSTICC, UMR CNRS 6185
Université de Bretagne Occidentale -
name@univ-brest.fr

September 15, 2018

Part I

Présentation du langage Occam

Objectifs du langage

Il se situe dans la lignée des langages concurrents provenant de *Communicating Sequential Processes* (CSP). L'initiateur de cette branche est Tony Hoare, un informaticien anglais éminent titulaire du *Turing Award*.

`https://amturing.acm.org/byyear.cfm`

- chaque opération élémentaire est un processus
- les processus se composent hiérarchiquement
- toutes les opérations de communication et de synchronisation sont menées sur des *canaux*.
- Ces opérations sont *bloquantes* et impliquent un rendez-vous.

- Occam définit des primitives essentielles : alternatives, timers .. et des constructions Record, Protocol, Réplications.
- Ces langages s'utilisent en logiciel, matériel (Handel-C) ou distribué.

Le compilateur kroc

Il existe plusieurs compilateurs ou interpréteurs disponibles. Il y a aussi eu des versions matérielles de ces interpréteurs. Nous utilisons kroc de l'Université de Kent qui est disponible sur plusieurs plateformes.

- Indications sur http://wsn.univ-brest.fr/moodle, cours SysDis.
- écrire un petit programme kroc suffixé en .occ: exo1.occ
- compiler : kroc exo1.occ
- exécuter : ./exo1
  vous avez un programme multi-treadé en binaire i386.

Dans le cas où le source utilise la librairie 'course', par exemple pour les impressions formatées:

- kroc exo1.occ -lcourse
- les entrées-sorties sont associées aux entrée-sorties standard Unix qui doivent être déclarées dans le processus principal.

Le compilateur kroc: illustration

ssh -X 172.12.18.154

```
PROC Sys(CHAN OF BYTE in,out,err)

  BYTE ch:
  SEQ
    ch := '0'
    SEQ i=0 FOR 10
      out ! ch + (BYTE i)
    out ! '*n'
:
~
~
~
~
~
~
~
~
~
:wq
```

ssh -X 172.12.18.154

```
kroc: Assuming you meant to compile sys1.occ
Warning-occ21-sys1.occ(3)- parameter err is not used
Warning-occ21-sys1.occ(3)- parameter in is not used
kroc: Ignoring course (no course.occ found)
bernard@joyeux:~$ kroc sys1 -lcourse
kroc: Assuming you meant to compile sys1.occ
Warning-occ21-sys1.occ(3)- parameter err is not used
Warning-occ21-sys1.occ(3)- parameter in is not used
bernard@joyeux:~$ !v
vi sys1.occ
bernard@joyeux:~$ !k
kroc sys1 -lcourse
kroc: Assuming you meant to compile sys1.occ
Warning-occ21-sys1.occ(3)- parameter err is not used
Warning-occ21-sys1.occ(3)- parameter in is not used
bernard@joyeux:~$ ls -l sys1
-rwxr-xr-x 1 bernard users 7357 2011-09-16 13:18 sys1
bernard@joyeux:~$ ./sys1
0123456789
bernard@joyeux:~$
```

Notation syntaxique du manuel Occam

C'est l'occasion de se familiariser avec une notation formelle de la grammaire des langages de programmation.

assignment = variable:=expression
*x1 := 1.2*1.8*
x2 := x1
est un exemple de spécification grammaticale pour une règle d'*affectation* utilisant *variable* et *expression*. La notation est proche de la notation dite BNF, elle même proche des outils de spécification de grammaires (yacc, bison, . . .)

sequence = **SEQ**
{ *process* }
définit une séquence comme le mot-clé SEQ suivi par zero ou plus instances de process.
A noter : la séquence de processus doit être en retrait de 2 blancs par rapport à l'alignement de **SEQ**.

Commentaires, noms

- Un commentaire démarre par deux traits et s'étend jusque la fin de la ligne
- x := x + 1 `-- ceci est un commentaire`
  y := y - 1
- les noms commencent par des minuscules, majuscules et se poursuivent par des lettres, chiffres ou points (.).:

PACKETS
vector6
LinkOut

terminal.in
pointu..

Processus primitifs: SKIP et STOP

SKIP

- mot-clé représentant un processus qui démarre, ne fait rien et termine immédiatement
- exemple:

```
SEQ
  clavier ? char
  SKIP
  ecran ! char
```

STOP

- mot-clé représentant un processus qui démarre, ne fait rien et ne termine jamais
- exemple:

```
SEQ
  clavier ? char
  STOP
  ecran ! char
   - - jamais exécuté
```

Processus construits: présentation

Constructions:

- **SEQ** : séquence d'instructions exécutées séquentiellement.
- **PAR** : séquence d'instructions exécutées en parallèle.
- **ALT** : alternative portant sur une série d'entrées.
- **IF** : branchement conditionnel
- **CASE** : sémection dans une série de valeurs
- **WHILE** : bouclage

- Les processus *concurrents* sont démarrés par les instructions PAR et ALT.
- Les constructions SEQ, IF, PAR et ALT peuvent être répliquées par des itérateurs

```
SEQ i=0 FOR 5 .
```

Processus construits: Séquence

- **SEQ** : définit une suite de processus réalisés les uns après les autres.
- *exemple* :

```
SEQ
    clavier ? char
    screen ! char
```

- - réalise une entrée, puis une sortie.

- Combinaisons structurantes: les SEQ peuvent s'imbriquer
- *exemple* :

```
SEQ
  SEQ
    screen ! '?'
    clavier ? char
  SEQ
    screen ! char
    clavier ? cr
    clavier ? lf
```

Processus construits: Séquence répliquée

Une réplication permet de développer un code séquentiel (SEQ i=0 FOR 2) sur la base d'un itérateur (i).

- **SEQ** : définit une suite de processus réalisés les uns après les autres.
- *exemple* :

```
SEQ i=0 FOR array.size
   stream ? data.array[i]
- - réalise une entrée, pour chacun des éléments
- - d'un tableau, indexé par i.
```

- 0 désigne la valeur initiale de l'index, et array.size désigne le nombre de pas.

exemple :

```
SEQ i=10 FOR 2
   ecran ! string[i]
```

équivaut à :

```
SEQ
   ecran ! string[10]
   ecran ! string[11]
```

Processus construits: Condition

Se substitue au `if (cond) ...` mais avec une sémantique plus précise

- *exemple* :

  IF

  $x < y$

  x := x + 1

  $x >= y$

  SKIP

- *ou* :

  IF

  $x < y$

  x := x + 1

  TRUE

  SKIP

- La condition groupe une série de processus *gardé* par des conditions évaluées séquentiellement.
- Si une condition est vraie, le processus est exécuté et la condition est terminée.
- Si aucune garde n'est vraie, le processus STOP est exécuté.

Processus construits: Condition répliquée

La réplication développe toutes les instances de l'itérateur (i). i est un entier (INT) qui peut etre lu, mais pas écrit.

- *exemple* (length=2):

```
IF
  IF i=1 FOR length
    string[i]<>objet[i]
      found := FALSE
  TRUE
    found:=TRUE
```

- *va se lire:*

```
IF
    string[1] <> objet[1]
      found := FALSE
    string[2] <> objet[2]
      found := FALSE
    TRUE
    found:=TRUE
```

- a la premiere inégalité, on renvoit FALSE, sinon, TRUE.

Si le nombre de pas est zero, on considère qu'aucune condition n'est vraie (STOP).

Processus construits: Sélection

CASE compare une variable avec des ensembles d'options.

- *exemple* :
  CASE direction
    up
      x:=x+1
    down
      x:=x-1
- si aucun branchement n'est vrai, on exécute STOP.

On peut couvrir les cas non précisé avec le mot-clé ELSE

- *cas multiples et par défaut*:
  CASE letter
    'a', 'e', 'i', 'o', 'u'
      vowel := TRUE
    ELSE
      vowel := FALSE

Processus construits: Boucles

WHILE répète un processus tant qu'une condition est vraie

- *exemple* :

```
WHILE char <> eof
  SEQ
    clavier ? char
    screen ! char
```

- ce qui suppose une valeur initiale correcte de char

```
SEQ
  char := '.'
  WHILE char <> eof
    SEQ
      clavier ? char
      screen ! char
```

  - la condition suivant WHILE doit rendre un booléen.
  - le processus controlé peut être parallèle (PAR)

Processus construits: Concurrence

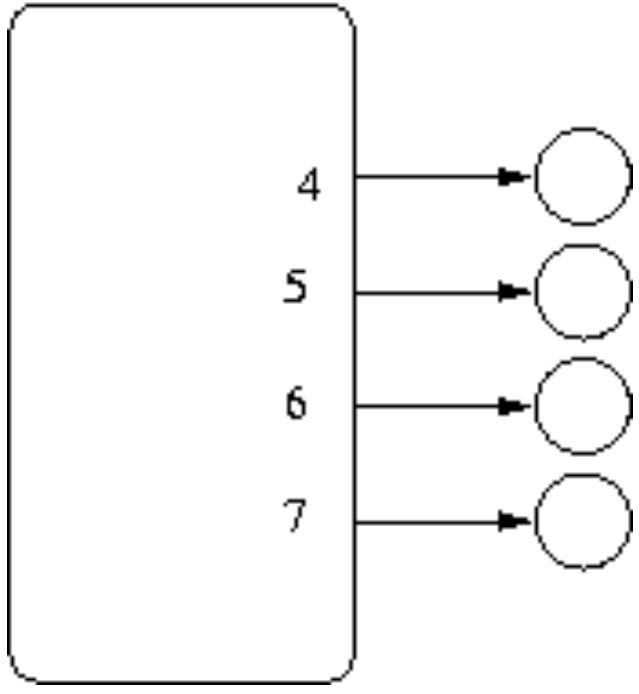
PAR permet d'exécuter des processus en concurrence.

- *exemple* :
  PAR
    clavier(de.clavier)
    filtre(de.clavier,vers.ecran)
    ecran(vers.ecran)
- les processus échangent des valeurs par canaux.
- les canaux sont unidirectionnels et point à point.
- les canaux ne peuvent etre lus et écrits dans le même processus
- les PAR peuvent être imbriqués arbitrairement
- les variables écrites par un processus ne peuvent être lues (et écrites) par les processus de la construction.
- PAR
    SEQ
      mice:=42
      c!42
    c? mice - - illégal

Processus construits: Concurrence et réplications

On peut construire des tableaux de processus en les connectant par des canaux également tabulés.

- *exemple* :
  PAR i=4 FOR 4
    user[i] ! message
- *se lit* :
  PAR
    user[4] ! message
    user[5] ! message
    user[6] ! message
    user[7] ! message



Librairie utilitaires : course.lib

”course.lib” propose des fonctions de formatage et d’impression. *Visiter dans l’arbre du compilateur kroc, le répertoire* `modules/course/libsrc`.

```
–* Write a string to a channel.
– outputs [@code s] in fieldwidth [@code field] to [@code out].
– @param s String
– @param field Field width to right-justify in
– @param out Channel to write to
PROC out.string (VAL []BYTE s, VAL INT field, CHAN BYTE
out!)
  VAL INT length IS SIZE s:
  SEQ
    out.repeat (’ ’,field - length, out!)
    SEQ i = 0 FOR length
      out ! s[i]
:
```

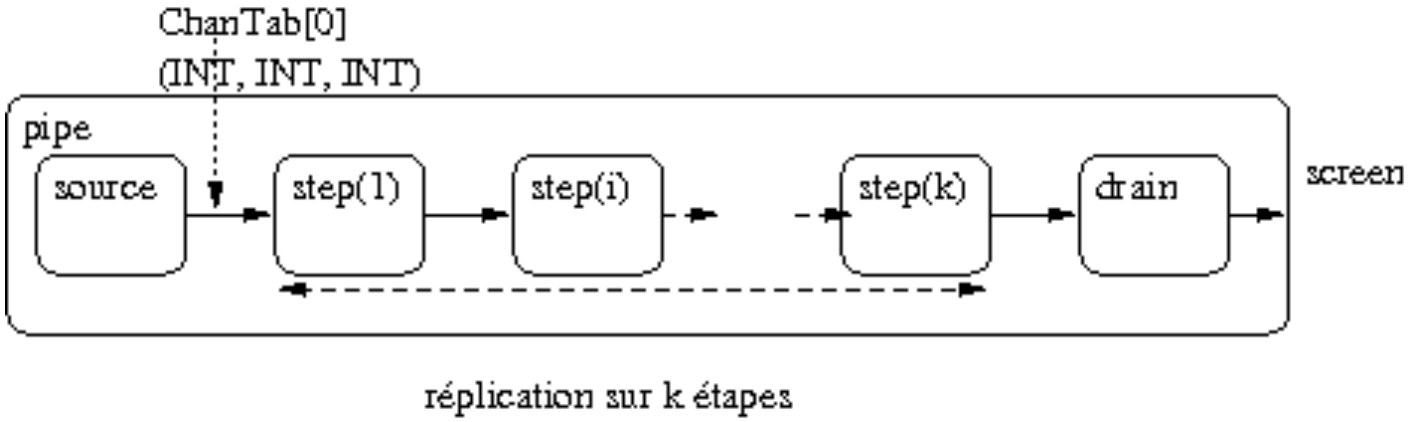
Impressions formatées

```
#USE ”course.lib”
PROC Sys(CHAN OF BYTE
in,out,err)
  VAL [] BYTE string IS ”un
message fort”:
  SEQ
    out.string(string,0,out)
    out ! ’*n’
    out.string(string,40,out)
    out ! ’*n’
:
– resultats:
un message fort
                    un message fort
```

```
#USE ”course.lib”
PROC Sys(CHAN OF BYTE
in,out,err)
  VAL [] BYTE string IS ”un
message fort”:
  INT n:
  SEQ
    n := SIZE string
    out.number(n,0,out)
    out ! ’*n’
    out.number(n,10,out)
    out ! ’*n’
: – resultat :
– 15
–         15
```

Exemple: un pipeline polynomial

On désire effectuer une somme de puissances sucessives d'une variable x. L'architecture proposée se présente donc ainsi:



La source produit des entiers, le drain imprime une chaine présentant les résultats.
$\Sigma_{i=1}^{k} x^i.a_i$ (on pose $a_i = 1$ et l'on saurait simplifier le calcul)

Un pipeline polynomial: ossature

```
PROC Sigma(CHAN OF BYTE kbd, screen, err)
  VAL INT K IS 4: - - degre du polynôme
  VAL INT N IS 6: - - valeurs calculées
  PROC Source(CHAN OF INT out)
  :
  PROC Step(CHAN OF INT in,out)
  :
  PROC Drain(CHAN OF INT in, CHAN OF BYTE screen)
  :
  [K+1] CHAN OF INT TabChan:
  PAR
    Source(TabChan[0])
    PAR i=0 FOR K
      Step(TabChan[i],TabChan[i+1])
    Drain(TabChan[K], screen)
:
```

Un pipeline polynomial: processus générique

```
PROC Step(CHAN OF INT in,out)
  INT xin, pin, sin, pout, sout:
  SEQ i=0 FOR N
    SEQ
      in?xin - - variable x
      in?pin - - puissance entrante
      in?sin - - somme partielle
      pout := xin * pin
      sout := sin + pout
      out! xin - - variable inchangée
      out! pout - - puissance sortante
      out! sout - - somme sortante
:
```

Un pipeline polynomial: processus source

```
PROC Source(CHAN OF INT out)
  SEQ i=0 FOR N
    SEQ
      out! i – x
      out! 1 – p
      out! 1 – s
:
```

Un pipeline polynomial: processus drain

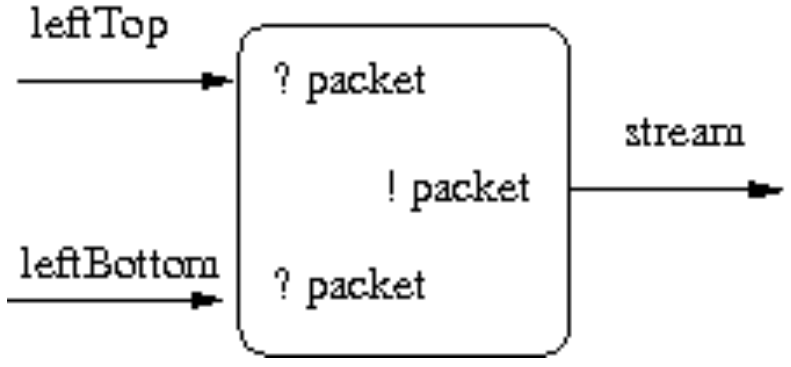
```
PROC Drain(CHAN OF INT in, CHAN OF BYTE screen)
  INT xin, pin, sin, p:
  SEQ
    screen!'*n'
    SEQ i=0 FOR N
      SEQ
        in?xin
        in?pin
        in?sin
        out.number(xin,8,screen)
        out.number(pin,8,screen)
        out.number(sin,8,screen)
:
```

Processus construits: Alternatives

ALT permet de laisser l'environnement décider quelle opération d'entrée prête sera prise dans un ensemble d'entrées possibles.

exemple d'un multiplexeur:

```
ALT
  leftTop ? packet
    stream ! packet
  leftBottom ? packet
    stream ! packet
```



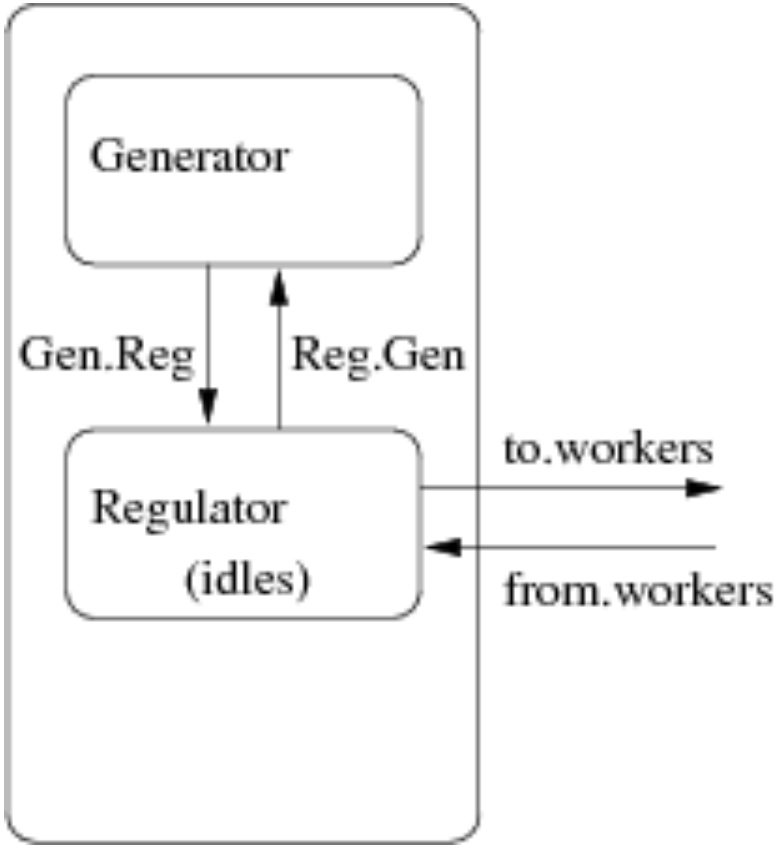
$\rightarrow$ Si une entrée est prête, elle est prise, si 2 entrées sont prêtes, une est tirée au hasard. Sinon l'appel bloque.

Alternatives gardées

Lors d'une alternative, on peut valider ou invalider une entrée en la conditionnant par une *garde* suivie de & . Les gardes déterminent un sous-ensemble des entrées où une opération alternative peut être opérée.

exemple d'un régulateur:

```
ALT
 from.worker ? result
    SEQ
      RegGen ! result
      idles:=idles+1
 (idles>=1) & Gen;Reg ? packet
    SEQ
      to.workers ! packet
      idles:=idles-1
```



Extension des clauses alternatives

Si il s'avère qu'aucune entrée n'est nécessaire dans le cas d'une garde particulière, on peut remplacer l'entrée par un processus vide SKIP.

exemple :

```
ALT
  in.data ? data
    out.data ! data
  dimanche & SKIP
    out.data ! no.data
```

On peut avoir 0, 1 ou 2 cas prenables..

- dimanche est faux et l'entrée est vide: on bloque.
- c'est dimanche, et pas de données, on sort no.data
- ce n'est pas dimanche, mais il y a une donnée, on la passe.
- c'est dimanche et il y a une donnée, alors no.data, ou data sortent au hasard

Réplication des alternatives

Les alternatives peuvent être répliquée pour par exemple ’scruter’ un tableau de canaux entrants.

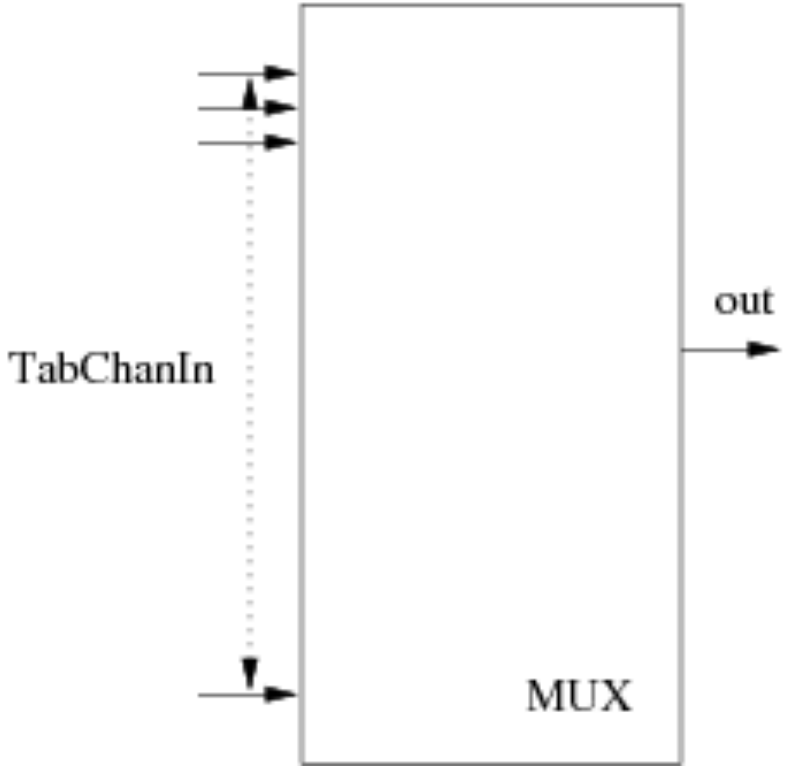
exemple :

```
[NChan]CHAN OF BYTE TabChanIn
ALT i=0 FOR NChan
  TabChanIn[i] ? data
    out ! data
```

On peut aussi les imbriquer les unes dans les autres:

exemple :

```
ALT
  ALT i=0 FOR NChan
    TabChanIn[i] ? ..
```



Ordre de priorité des alternatives

Cet ordre peut être forcé entrainant un examen séquentiel strict des entrées faisables:

exemple 1: priorité

```
PRI ALT
  exceptions ? event
    urgent(event)
  normaux ? event
    lent (event)
```

exemple 2: scrutation

```
WHILE pasFini
  PRI ALT
    exceptions ? event
      urgent(event)
    travailler & SKIP
      SOccuper(pasFini)
```

Types simples

BOOL	booléen (FALSE ou TRUE)
BYTE	entiers positifs entre 0 et 255
INT	entiers signés de la plateforme
INT16	entiers signés entre $(-1).2^{15} - 1$ et 2^{15}
	respectivement INT32 et INT64
REAL32	IEEE 754 réel (s=1 e=8 m=23)
REAL64	IEEE 754 réel (s=1 e=11 m=52)

Les *types nommés* peuvent être renommés pour forcer une sémantique:

```
DATA TYPE MONEY IS INT64:
MONEY a, b:
INT64 c:
```

```
a := a + b - - OK
a := c - - Non!
```

Litéraux

Formulation des litéraux:

123	entier en décimal
#11	entier en hexadécimal
’a’	caractère codé en un BYTE
TRUE	constante BOOL

Dans les expressions, il est utile de préciser le type associé à une constante derrière cette constante:

123 (BYTE)	entier 8 bits
’x’ (INT)	entier représentant ’x’
42.0 (REAL32)	32 bits
42.0 (REAL64)	64 bits
42 (MONEY)	type nommé

Tableaux

Ce sont des agglomération d'éléments de même type.

- [5] INT : désigne un tableau de 5 entiers énumérés de 0 à 4.
- les tableaux peuvent intervenir dans les définitions de types et utiliser des types nommés.
- [3][5] INT : désigne un tableau de 3 tableaux de 5 entiers énumérés de 0 à 4.
- DATA TYPE VECTOR10 IS [10] INT : définit VECTOR10 comme un tableau de 10 entiers.
- VAL INT DIM IS 3:
  DATA TYPE MATRIX2 IS [*DIM*][*DIM*] REAL32 : définit une matrice 3x3 de flottants.

Record

On peut aggréger des éléments de différents types simples ou nommés en structurant des Records:
exemple:

```
DATA TYPE Complex
   RECORD
      REAL32 real,image:
:
ou:
DATA TYPE MoreComplex
   RECORD
      REAL32 real,image:
      MONEY itsGaz:
      [2]INT couple:
:
```

Records litéraux, records compacts

- En listant des litéraux, et en forçant leur interprétation par des types nommés, on peut construire des initialiseurs complexes.

exemple:
$[[2,3](COMPLEX),10000(MONEY),[1,2](COUPLE)](MoreComplex$

- Il est parfois intéressant de demander de compacter les nombres pour minimiser la place occupée (ou le temps de communication). Ceci s'effectue dans l'ordre de déclaration des champs, et les données peuvent ne pas être portable par défaut d'alignement.

exemple:
PACKED RECORD
  BYTE b:
  INT16 i:

Variables et valeurs

- On déclare une variable en nommant son type, et son nom:
  INT i:
  MONEY mini,max:
  [4][5] INT table2:
- On utilise une variable en la nommant, simplement ou dans une expression.
- Les dimensions des tableaux doivent être calculable à la compilation (pas de variables),
- [9]*INTclocks*:
  [8][9][10]*INTdata*:
- *clocks*[1] : second élément de la table
  data[0] : premier élément de la table multidim [9][10]
  data[*i*][0] : premier élément de la i-*ieme* table [10] entiers

Exemple de collection indexée

```
#USE ”course.lib”

PROC main(CHAN OF BYTE k,s,e)
  VAL INT N IS 100:
  DATA TYPE T10 IS [10]INT:
  [N]INT table:
  PROC Fill()
    SEQ i=0 FOR N
      table[i]:=i
  :
  PROC Copy(VAL INT k, [10]INT   dest)
    dest := [table FROM k FOR 10]
  :
```

Exemple de collection indexée

```
[10]INT IaTab:
INT sum:
SEQ
  Fill()
  SEQ i=0 FOR 10
    SEQ
      Copy(i,IaTab)
      sum := 0
      SEQ j=0 FOR 10
        sum := sum + IaTab[j]
      out.number(sum,10,s)
      s ! '*n'
:
```

Communications tamponnées

On peut définir les canaux comme des tableaux ou des records et effectuer des communications par lot de données, si les types sont compatibles:

```
[10]INT laTab,tab2:
INT sum:
CHAN OF [10]INT c10,d10:
SEQ
  Fill()
  SEQ i=0 FOR 10
    SEQ
      Copy(i,laTab)
      sum := 0
      SEQ j=0 FOR 10
        sum := sum + laTab[j]
      out.number(sum,10,s)
      s ! '*n'
  PAR
    c10 ! laTab
    c10 ? tab2
:
```

Adressage des champs de records

Les champs de records s'adressent simplement en indexage symbolique, à la manière d'un tableau, mais en indiquant le nom du champ.

```
DATA TYPE cplx
  RECORD
    INT r,i:
:
PROC Test()
  cplx z1,z2:
  INT t:
  SEQ
    t:= z1 [r]
    z1 [r]:= z2 [i]
    z2 [i] :=t
    t:= z2 [r]
    z2 [r]:= z1 [i]
    z1 [i] :=t
:
```

Part II

Algorithmes Distribués

Systèmes Distribués et Réseaux, Master1, 2018

Algorithmes distribués

Bernard Pottier
Université de Brest
http://wsn.univ-brest.fr/pottier

Simulations synchrones en Occam

Relation du modèle théorique avec le codage de simulations:

- Etat du processus (state): ensemble des variables locales modifiées à chaque étape.
- Etat initial du processus (start): variables locales avant la premiere étape.
- Messages entrant et sortant (mesg) : tableaux de variables qui alimentent les liens ou sont affectés par les liens, en parallèle.
- Fonction de transition (trans): inclue dans les programme du noeud qui se décompose en deux phases de avec des communications sortantes (M) et entrantes (N), et le changement d'état (C).
- Alphabet de messages : ensembles décrivant les données circulant sur les liens que l'on associe à un protocole.

Protocoles pour les messages en Occam

```
Protocole
PROTOCOL  diam.proto
  CASE
    tab ; TableIdVal
    null ; BYTE
    int; INT
:
```

```
Ecrire
    PAR i=0 FOR N
      out[i]  ! int ; tabOut[ i]
```

```
Déclarer :
PROC Node([N] CHAN OF  diam.proto in,out)
   [N] INT   tabInt,tabOut:
   BYTE isNull:
   SEQ
      SEQ i=0 FOR N
        tabOut [i] := i
      - -
```

```
Lire
    PAR i=0 FOR N
      in [i] ? CASE
        int ; tabIn[i]
          SKIP
        null ; isNull
          SKIP
```

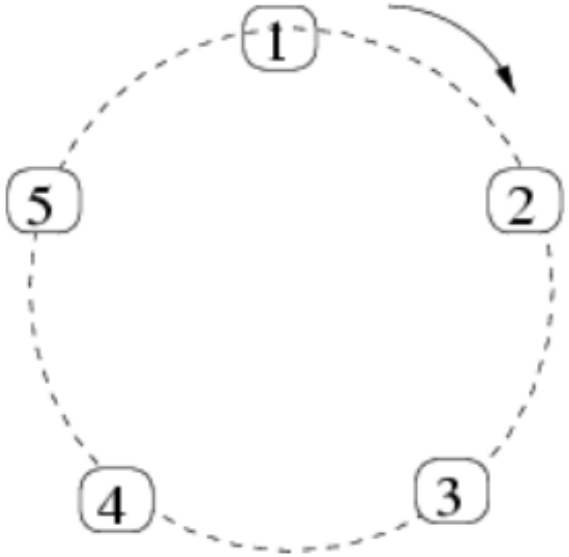
Exemple d'usage, protocoles en Occam

- Plusieurs types de données utilisés à différents moments d'un algorithme (exemple du diamètre avec des tables et des entiers)
- Messages null (BYTE) servant à la synchronisation.

La définition de ces types est globale à un réseau, les messages circulant de proche en proche.

Ils sont tous interprétables en utilisant le protocole.

Algo abstrait: leader sur un anneau



initialement:

- uid := ID
- out_i := uid
- $state_i$:= unknown

fonction $trans_i$:

- v:= N[0]
- send := null
  si $v > uid$ send := v
  si v=uid $state_i$:= leader

- chaque noeud a un UID
- le nombre de processus est inconnu

l'algorithme finit en n étapes et n^2 messages. n est le diamêtre.
Un et un seul processus s'élit *leader*

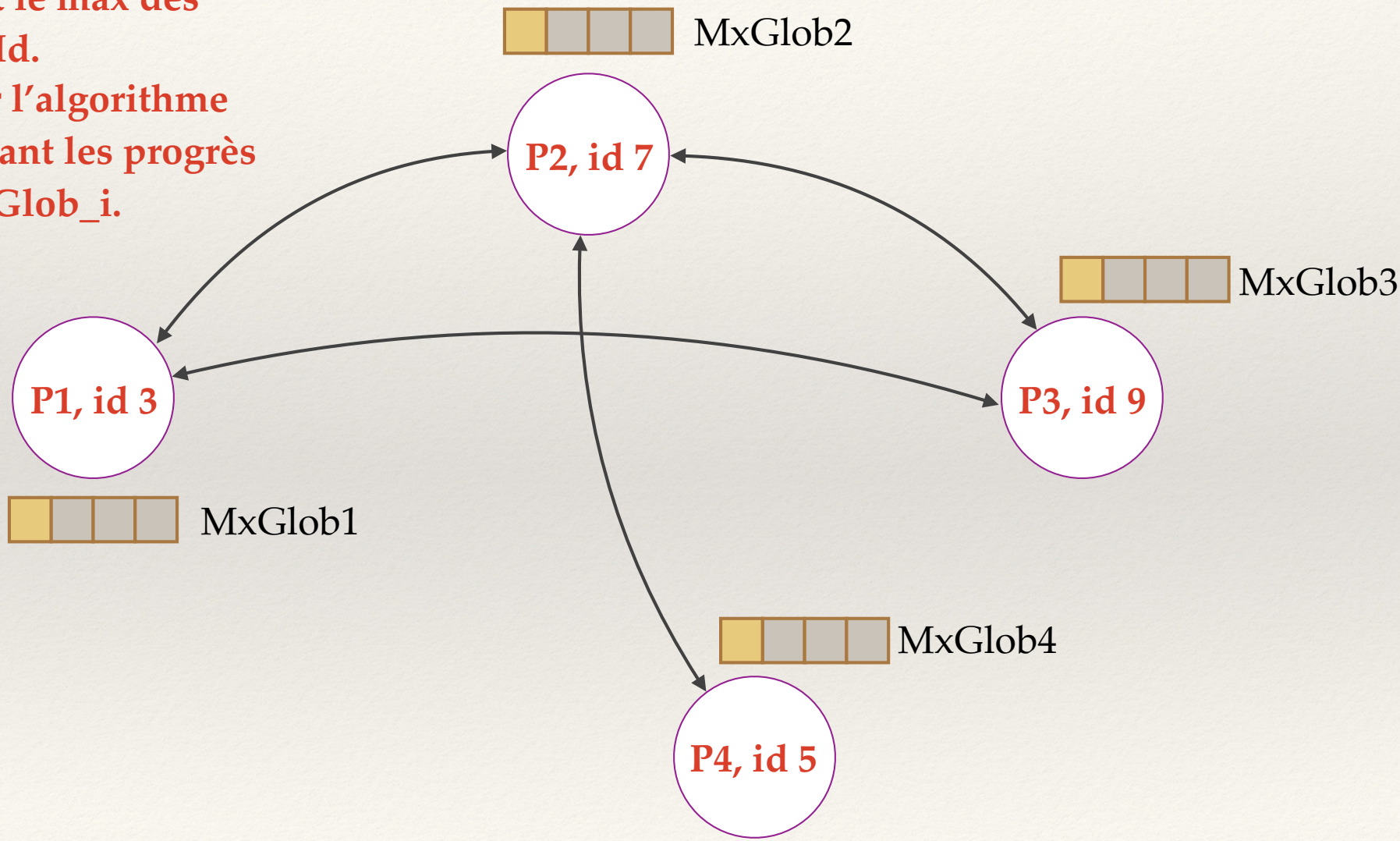
Algo simulé : leader sur un anneau

Description sur un anneau directionnel en Occam:

- N=5 Processus Pi exécutent Node() dans un PAR
- Ils ont 1 canal entrant et 1 canal sortant
- initialement, MaxLocal := MaxGlobal := Identité,
- On écrit MaxGlobal sur le lien sortant
- On boucle N=5 fois:
  - communication parallele (PAR) sur lien in et out
  - comparer MaxGlobal avec valeur entrante et mise à jour
  - Ecrire MaxGlobal (ou null) sur le lien sortant

Algo informel : maximum distribué

On veut le max des entiers Id.
Rejouer l'algorithme en figurant les progrès des MxGlob_i.



Leader sur un réseau quelconque

Principe:

- La valeur discriminante est l'Identificateur (Id)
- On doit calculer le maximum MaxGlobal des Ids
- Le propriétaire de MaxGlobal est unique et est le leader sur le réseau

Note:

Si le réseau est non-connexe, alors il y aura plusieurs leaders

Algo de maximum distribué, simulation

Principe:

- chaque nœud a une variable locale MaxLoc et un avis sur le maximum global MxGlobal (initialement MaxLoc=Id)
- On exécute Nb tours, en plaçant MxGlobal dans les tampons sortant vers les voisins
- Après chaque communication, on compare MxGlobal aux valeurs entrantes. On le met à jour si une entrée est plus grande.

Application:
En *MaxNodes -1* étapes, tout le monde connait le maximum global

Algorithme de Diamètre - Stratégie

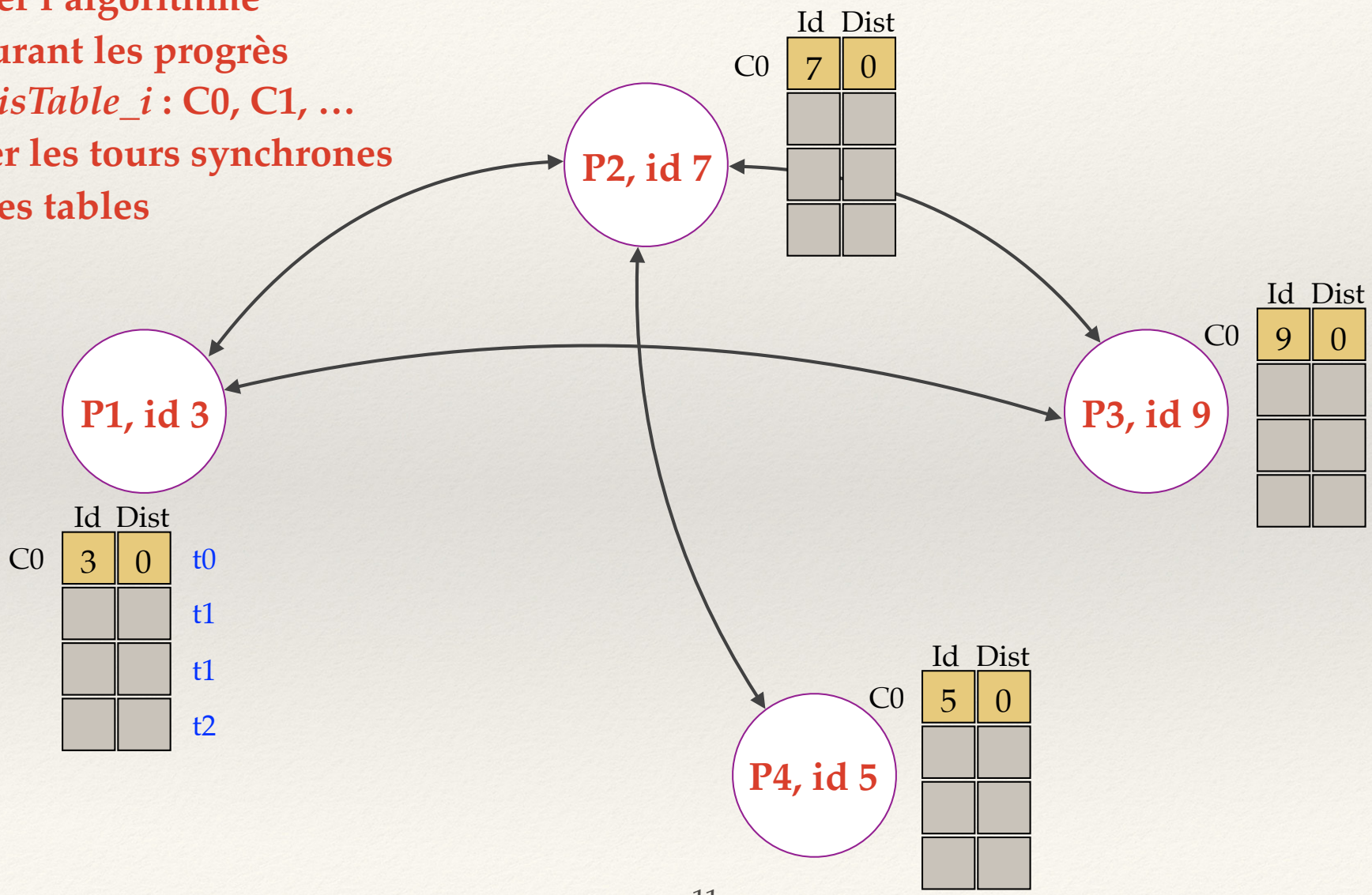
Chaque Node a une table T où il enregistre les Id+Distance des autres nœuds.
Les messages sont des files d'enregistrements (FE) Id+Distance
On démarre en plaçant son enregistrement dans T, et on tourne N fois
A chaque tour, on regarde si un nouveau nœud s'est fait connaitre
Si c'est le cas, on l'enregistre, localement dans T, en incrémentant la Distance, et on l'ajoute dans une FE pour le tour suivant.

En N étapes tout le monde connait tout le monde, et la plus grande Distance relative.

Un max global permet de connaitre le diamètre.

Réseau R1, calcul de distances relatives

Rejouer l'algorithme en figurant les progrès des *thisTable_i* : C0, C1, … Figurer les tours synchrones dans les tables



Diamètre - structures données

```
DATA TYPE IdentifiedValue
 RECORD
  INT Id:
  INT Val:
:
```

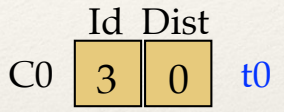
```
DATA TYPE TableIdVal
 RECORD
  INT limit:
  [MaxNodes] IdentifiedValue tabNodes
:
```

```
Protocole
PROTOCOL diam.proto
 CASE
  tab ; TableIdVal
  null ; BYTE
  int; INT
:
```

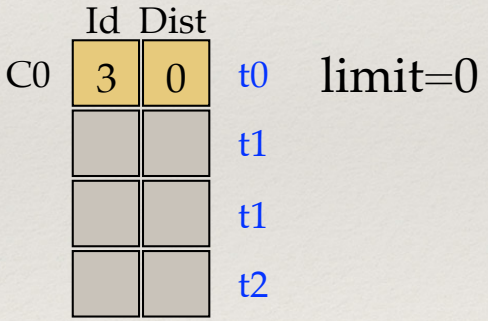
```
PROC AddInTable( TableIdVal thisTable , IdentifiedValue idVal)
 INT limit:
 SEQ
  limit:= thisTable[ limit]
  thisTable[ tabNodes][limit] :=idVal
  thisTable [limit] := limit+1
:
```

Diamètre Phase 1- initialization données

```
PROC  InitMyVal(IdentifiedValue  myVal,INT  aValue)
  SEQ
    myVal[Id] :=identity
    myVal[Val] :=aValue
:
```

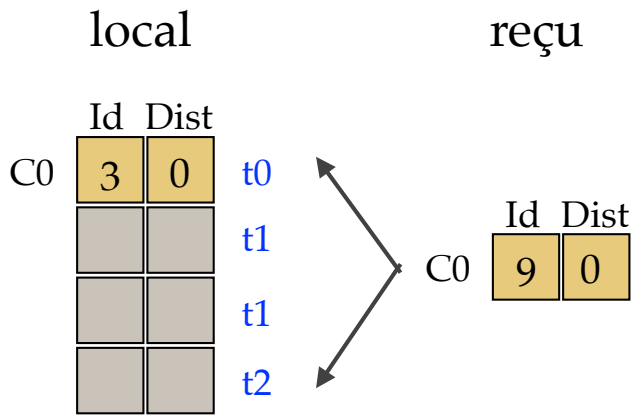



```
PROC  InitMyTable(TableIdVal  thisTable)
  SEQ
    thisTable[limit] :=0
    AddInTable(thisTable,myVal)
:
```



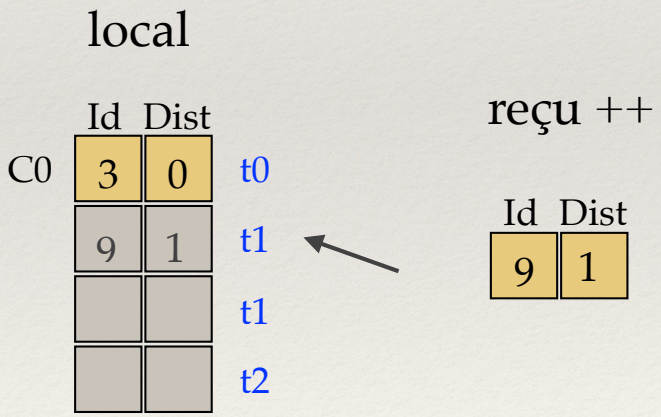
Diamètre - Lookup connaissances

```
PROC LookInTable(TableIdVal thisTable,IdentifiedValue idVal,BOOL found)
 INT limit:
 SEQ
  found:=FALSE
  SEQ index= 0 FOR thisTable[ limit]
   IF
    thisTable[tabNodes][index][Id] =idVal[Id]
      SEQ
       idVal:= thisTable[tabNodes][index]
       found:=TRUE
    TRUE
      SKIP
:
```



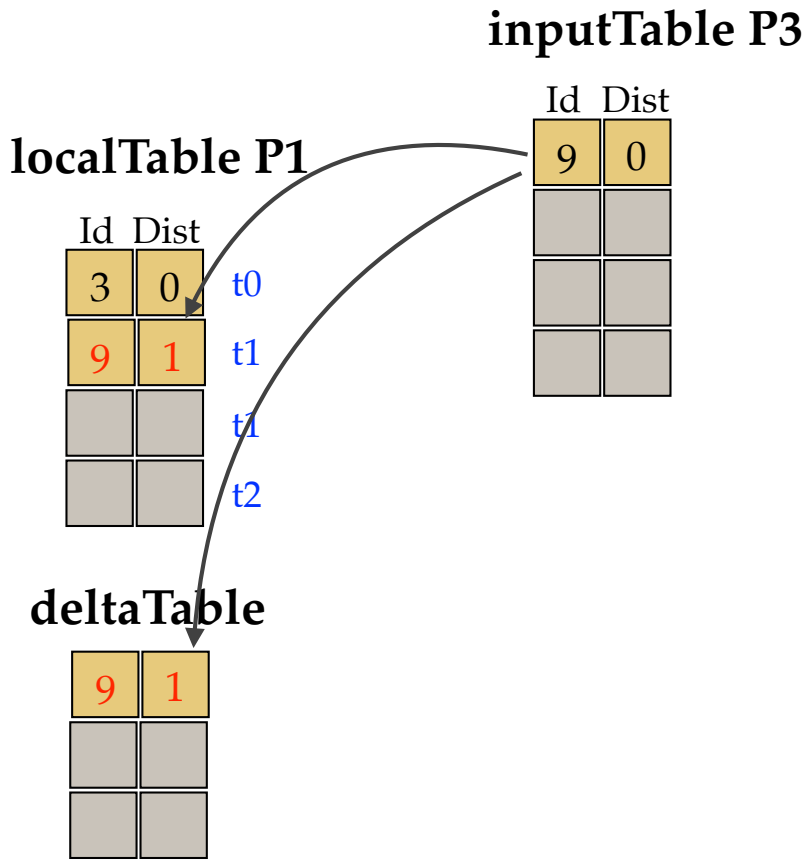
Diamètre - Insertion table connu

```
PROC AddInTable(TableIdVal thisTable, IdentifiedValue idVal)
  INT limit:
  SEQ
    limit := thisTable[limit]
    thisTable[tabNodes][limit] := idVal
    thisTable[limit] := limit +1
:
```



Diamètre - Intégration table P3

```
PROC MergeTable(TableIdVal localTable, TableIdVal inputTable, TableIdVal deltaTable)
  INT limit:
  BOOL found:
  IdentifiedValue idVal:
  SEQ
    SEQ i=0 FOR inputTable[limit]
      SEQ
        idVal := inputTable[tabNodes] [i]
        LookInTable(localTable,idVal, found)
        IF
          NOT found
            SEQ
              idVal [Val ] := idVal [Val ] + 1
              AddInTable(localTable, idVal)
              AddInTable(deltaTable, idVal)
          TRUE
            SKIP
:
```



Diamètre - Boucle synchrone (a)

```
PROC Circulate(INT ivar)
 TableIdVal deltaTable:
 SEQ
  InitMyVal(myVal, ivar)
  InitMyTable(myTable)
  SEQ i=0 FOR SIZE out
   outMessages[i] := myTable
  SEQ tours =0 FOR MaxNodes −1
   SEQ
    PAR
     PAR i=0 FOR SIZE in
      in [ i] ? CASE
       tab ; inMessages[i]
         SKIP
     PAR i=0 FOR SIZE out
      out [ i] ! tab ; outMessages[i]:
```

Initialisation

Messages sortants

Boucle synchrone

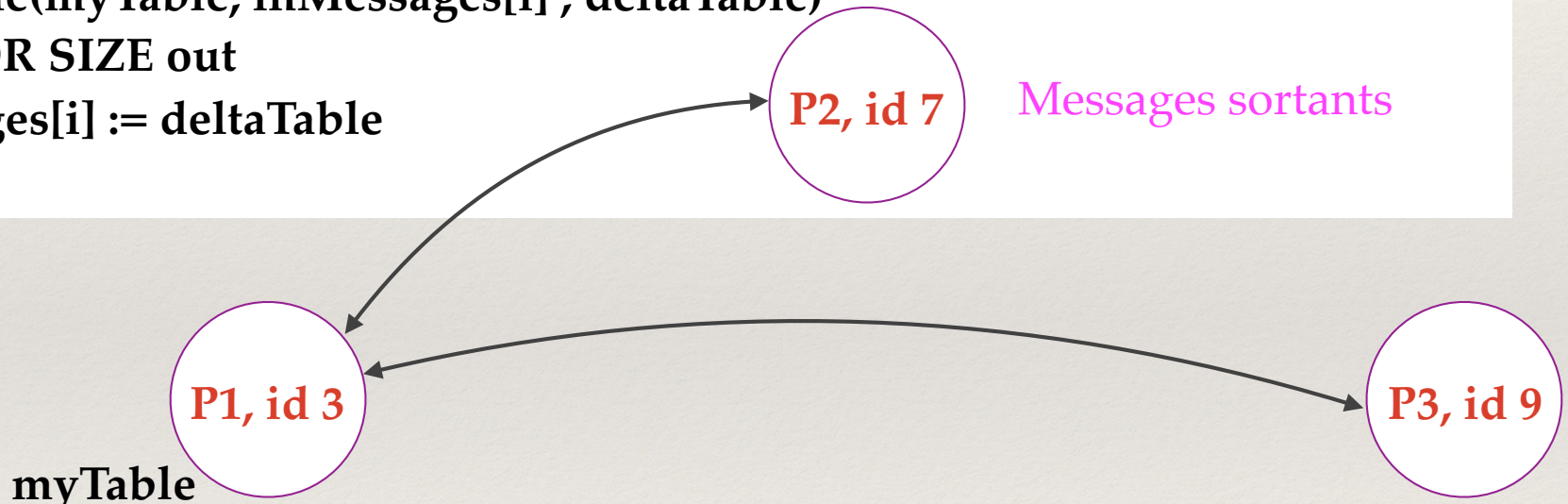
Communications

Diamètre - Boucle synchrone (b)

```
PROC Circulate(INT ivar)
…
      InitMyTable(deltaTable)
      SEQ i=0 FOR SIZE in
        MergeTable(myTable, inMessages[i] , deltaTable)
      SEQ i=0 FOR SIZE out
        outMessages[i] := deltaTable
:
```

Changement d'état

Messages sortants



myTable

Id	Dist	
3	0	t0
9	1	t1-P3
7	1	t1-P2
		t2

deltaTable

9	1	t1-P3
7	1	t1-P2

Suite Diamètre : Phase 2

Calculs des maxima locaux

Balayage local de myTable, colonne Dist

Calculs du maximum des maxima locaux

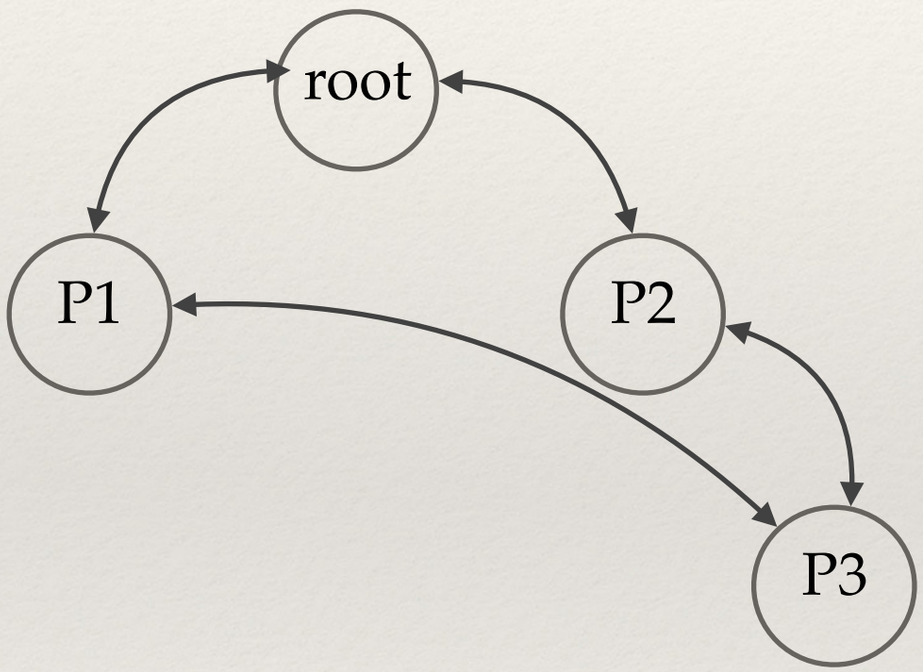
Procédure de maximum distribué appliqué au maximum local -> **Diamètre**

Parcours en largeur d'abord – BFS

- *Breadth First Search* en anglais.
- Réseau quelconque, on traite un cas connexe.
- Noeud de départ dénommé *root*
- L'algorithme construit un arbre permettant de joindre tous les nœuds en un temps le plus court possible
- Les routes sont fixées dans des variables d'état des nœuds intermédiaires.

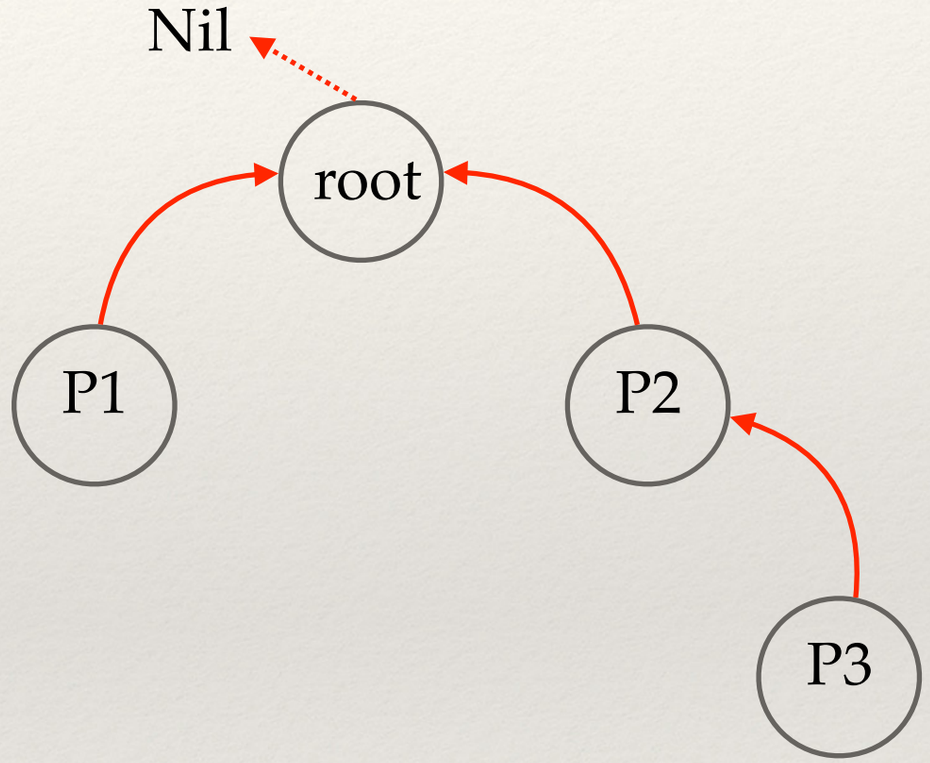
Relation entre réseau et arbre

Réseau R3



Réseau bi-directionnel, racine *root*

Arbre produit par BFS



Arbre directionnel, remontant,
chaque nœud a repéré un index de canal
menant à la racine: *P3* a gardé *P2, pas P1*

BFS synchrone, état initial

- Etat des nœuds:
  - variable booléenne *marked,*
    - *vrai* pour *root*.
    - *faux* ailleurs
- variable entière *parent* : *nil (-1)* partout
- messages initiaux sortants
  - *send* = *search* pour *root*
  - *send* = *null* ailleurs
- (les messages entrants seront dans *received [i]*)

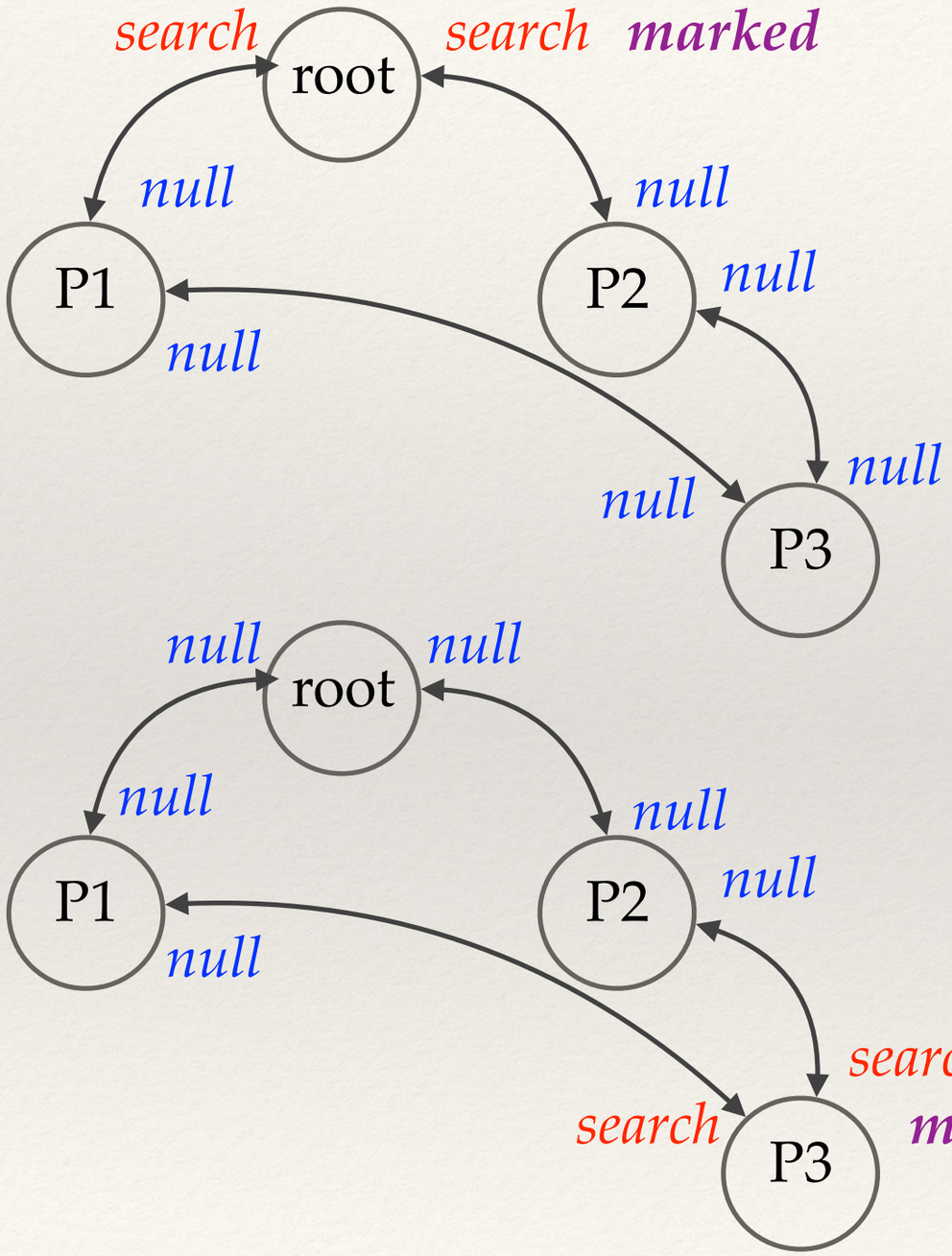
BFS synchrone, boucle

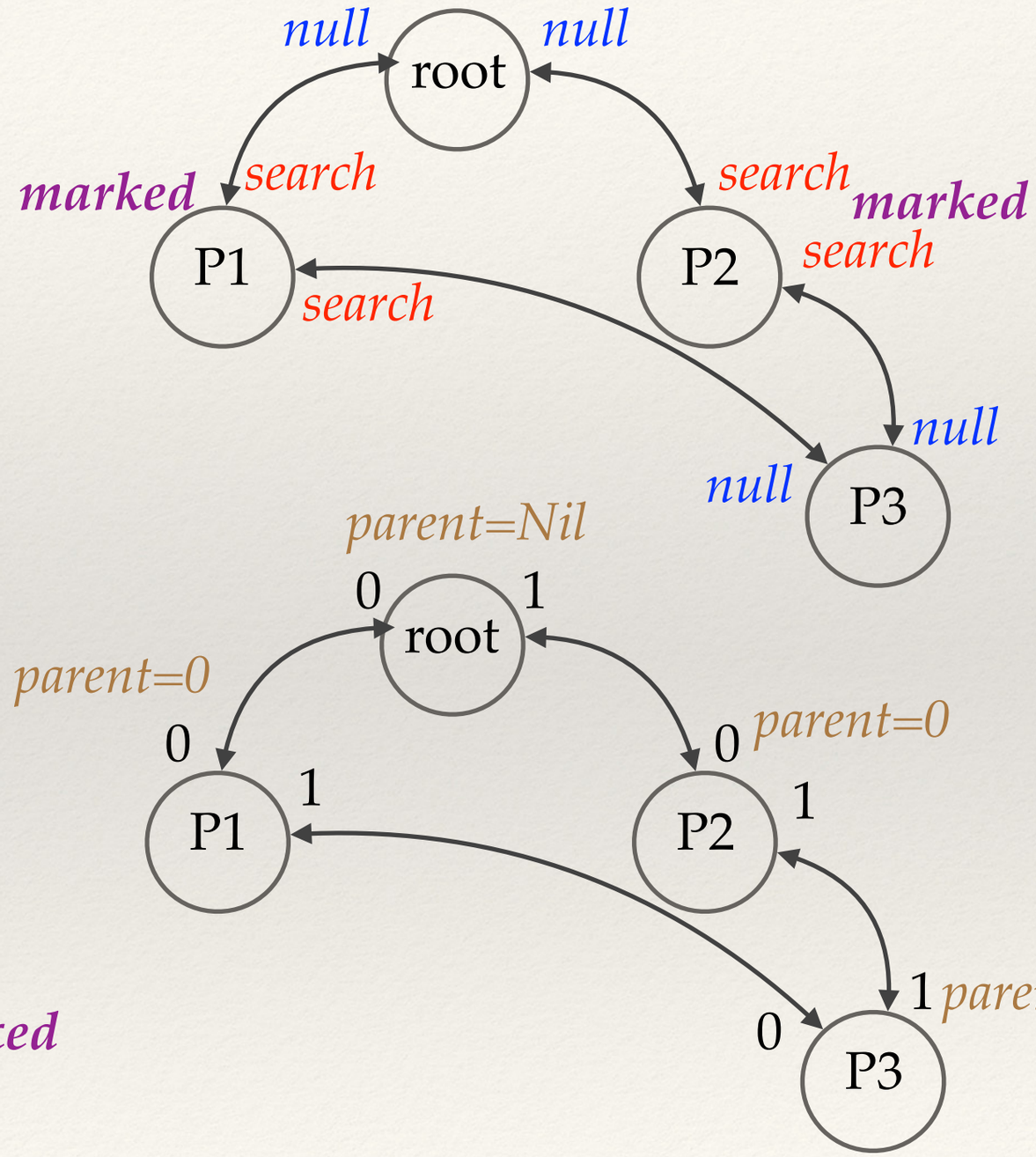
- Répéter N fois:
  - communiquer avec les voisins
  - changement d'état et messages sortants:
    - si *marked* alors *send := null*
    - si *not marked*
      - si *received[i] = search* alors
        - *parent := i, marked := vrai, send := search*
        - sinon *send := null*

BFS synchrone, état final

- Après N tours, (N = MaxNodes -1, ou N = Diamètre)
  - tous les nœuds ont basculé *marked* à *vrai*
  - tous les nœuds ont un index dans *parent*
- Un arbre est constitué qui permet de remonter des informations vers la racine dans un calcul ultérieur.

Exemple d'exécution





Résultat du BFS

- Dans un réseau connexe, chaque nœud connait son *parent*. Il peut utiliser cet index pour des calculs ultérieurs transformant le réseau en arbre *remontant* vers *root*.
- Des calculs descendants peuvent être menés au vol en ajoutant des attributs aux messages *search*:
  - sommes, maxima, minima, sur les arcs descendants
  - distance relative des Pi à *root*, inférée du tour de réception.
- **Pi ayant reçu un message *search* sur un canal *parent* peut renvoyer le message *beParent* au tour suivant, afin de notifier son choix.**
- **Un nœud recevant ce message gardera la position de ce fils dans une file.**

Applications et optimisations du BFS

- évaluation de profondeurs, en remontant la distance à *root*.
- Plusieurs BFS peuvent être démarrés en concurrence, permettant d'obtenir les profondeurs des arbres de recouvrement en tout point (alternative pour calculer le diamètre).

Code : constantes,variables, initialisation

```
VAL INT Nil IS -1:
VAL INT Null IS 0:
VAL INT Search IS 1:

PROTOCOL diam.proto IS INT:

PROC Node([] CHAN OF diam.proto in,out, VAL INT id, CHAN OF BYTE toMux)

  [MaxFanOut] INT inBuf,outBuf:
  INT parent,send:
  BOOL marked,root:
  SEQ
    root := id = 0
    marked := root
    parent := Nil
    IF
      marked
        send := Search
      TRUE
        send := Null
```

— initialisation

Code : sorties et communications

```
PROC Node([] CHAN OF diam.proto in,out, VAL INT id, CHAN OF BYTE toMux)

  [MaxFanOut] INT inBuf,outBuf:
  INT parent,send:
  BOOL marked,root:
  SEQ
    - - raccourci

  SEQ i=0 FOR SIZE out
    outBuf[i] := send
  SEQ tours =0 FOR MaxNodes - 1
    SEQ
      PAR
        PAR i=0 FOR SIZE in
          in[i] ? inBuf[i]
        PAR i=0 FOR SIZE out
          out[i] ! outBuf[i]
```

— buffer sortant

— bouclage

— communication

Code : gestion de l'état

```
PROC Node([] CHAN OF diam.proto in,out, VAL INT id, CHAN OF BYTE toMux)
  [MaxFanOut] INT inBuf,outBuf:
  INT parent,send:
  BOOL marked,root:
  SEQ
    - - raccourci
      IF
        marked
          send := Null
      TRUE
        SEQ
          send := Null
          SEQ i=0 FOR SIZE in
            IF
              inBuf[i] = Search
                SEQ
                  marked := TRUE
                  parent := i
                  send := Search
              TRUE
                SKIP
SEQ i=0 FOR SIZE in
  outBuf[i] := send
```

— **propagation**

Code : statut et résultat

```
PROC Node([] CHAN OF diam.proto in,out, VAL INT id, CHAN OF BYTE toMux)
  [MaxFanOut] INT inBuf,outBuf:
  INT parent,send:
  BOOL marked,root:
  SEQ
    - - raccourci
    out.number(parent,4,toMux)
    toMux ! '*n'
:
 ./PaP
   5         0
   0        -1
   1         0
   2         0
   3         1
   4         0
```

— trace

— root

BFS : usage de l'arbre remontant

```
- - 2 étapes, 2 procédures
PROC Node([] CHAN OF diam.proto in,out, VAL INT id, CHAN OF BYTE toMux)
  [MaxFanOut] INT inBuf,outBuf:
  INT parent,send:
  BOOL marked,root:
   PROC Step1()
    SEQ
      root := id = 0
      marked := root
      parent := Nil
    - -  Proc Node précédent
    :

    PROC Step2()
      INT Max:
      SEQ
        Max:= id + 1
        send := Max
      - - a completer
    :
  SEQ
    Step1()
    Step2()

:$
```

BFS : Step 2, initial et boucle

```
- - 2 étapes, 2 procédures
PROC Node([] CHAN OF diam.proto in,out, VAL INT id, CHAN OF BYTE toMux)
  [MaxFanOut] INT inBuf,outBuf:
  INT parent,send:
  BOOL marked,root:

  PROC Step2()
    INT Max:
    SEQ
      Max:= id + 1 - - pour pas de confusion avec Null
      send := Max
      SEQ i=0 FOR SIZE out
        outBuf[i] := send
      SEQ tours =0 FOR MaxNodes - 1
        SEQ
          PAR
            PAR i=0 FOR SIZE in
              in[i] ? inBuf[i]
            PAR i=0 FOR SIZE out
              out[i] ! outBuf[i]
    - -  raccourci

:
```

BFS : Step 2, gestion de l'etat

```
PROC Node([] CHAN OF diam.proto in,out, VAL INT id, CHAN OF BYTE toMux)
  [MaxFanOut] INT inBuf,outBuf:
  INT parent,send:
  BOOL marked,root:

  PROC Step2()
    INT Max:
    SEQ
    - -  raccourci
         SEQ i=0 FOR SIZE in
           IF
             inBuf[i] > Null  - -  starter Null
               IF
                 inBuf[i] > Max
                   Max := inBuf[i] - - mise a jour Max local
                 TRUE
                   SKIP
             TRUE
               SKIP
         SEQ i=0 FOR SIZE out
           IF
             i=parent - - on passe le Max seulement au parent
               outBuf[i] := Max
             TRUE
               outBuf[i] := Null
```

Etat t+1

BFS : Step 2, statut final

```
PROC Node([] CHAN OF diam.proto in,out, VAL INT id, CHAN OF BYTE toMux)
  [MaxFanOut] INT inBuf,outBuf:
  INT parent,send:
  BOOL marked,root:

  PROC Step2()
    INT Max:
    SEQ
    - -  raccourci
      out.number(Max,4,toMux)
      toMux ! '*n'
  :

Root a le maximum calcule sur l'arbre.
./PaP
   4        6
   0        6
   1        4
   2        6
   3        5
   5        6
```

BFS : arbre descendant (TD/TP)

- Adjonction d'un message *beParent*
- Adjonction d'une table de fils
- Procédure AddFils permettant d'ajouter un fils

```
DATA TYPE TabDeFils
  RECORD
    INT limit:
    [MaxFanOut] INT IndexFilsTab:
:
```

à la première réception de *search*

- on élit un *parent* et un seul
- on expédie *beParent* à ce *parent*, et *search* ailleurs

au tour qui suit le passage de search

- on cherche les *beParent*
- on enfile les index des canaux dans la table de fils

Chaque noeud sait si il a des fils ou pas, et quels sont les canaux concernés
On a constitué un arbre descendant

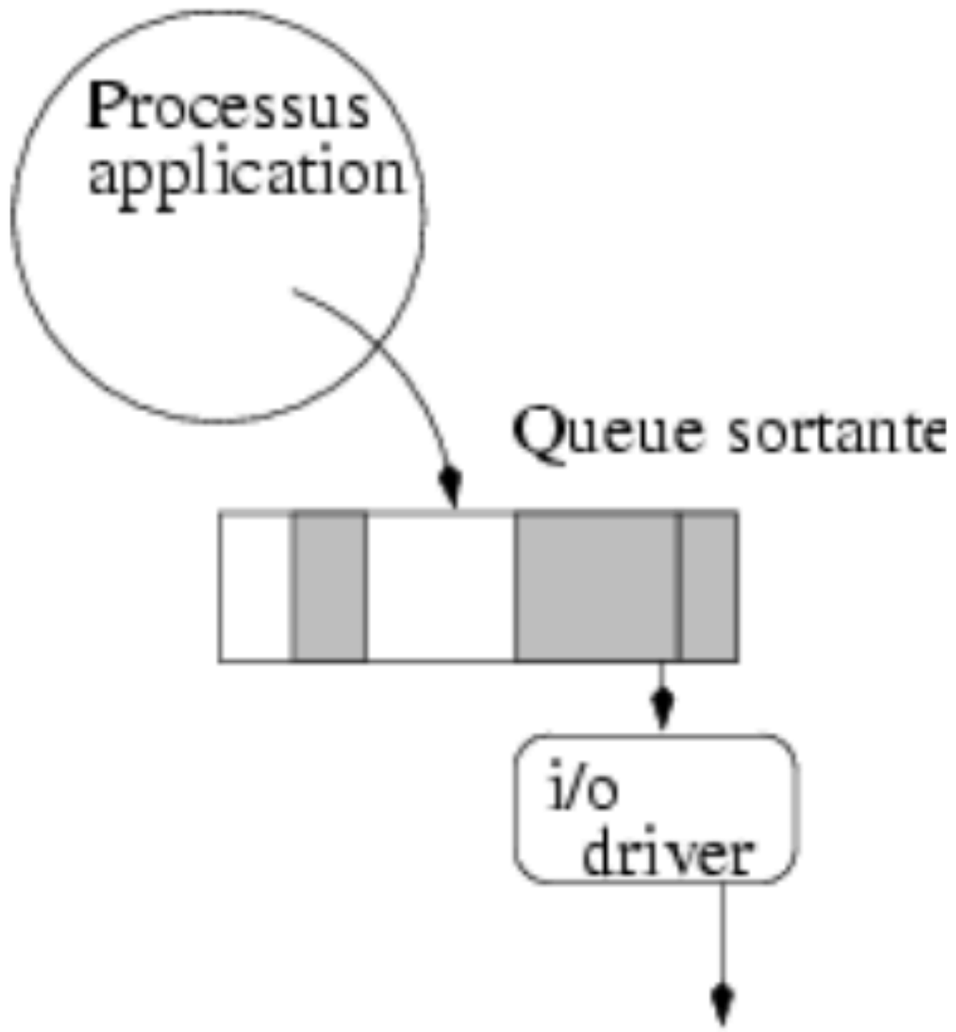
Modèle asynchrone

- Idée de base : découpler les partenaires d'un algorithme
- Application principale : systèmes client serveurs où le client va enfiler des requêtes en surveillant les réponses
- Fondement technique : les queues établies dans le réseau, chez l'émetteur, les routeurs, les récepteurs
- Nombreux messages en transits, dispersés, dont il faut surveiller le statut.

Queue émission (first in first out)

Les tampons sortants peuvent être mis en attente pour une variété de raisons:

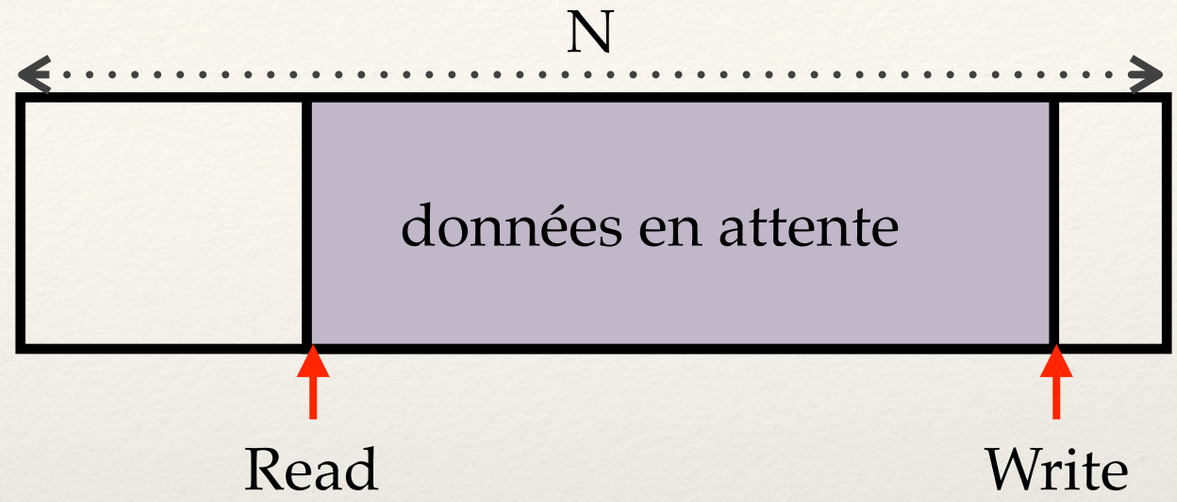
- medium indisponible
- concurrence d'autres processus



Queues: mécanisme

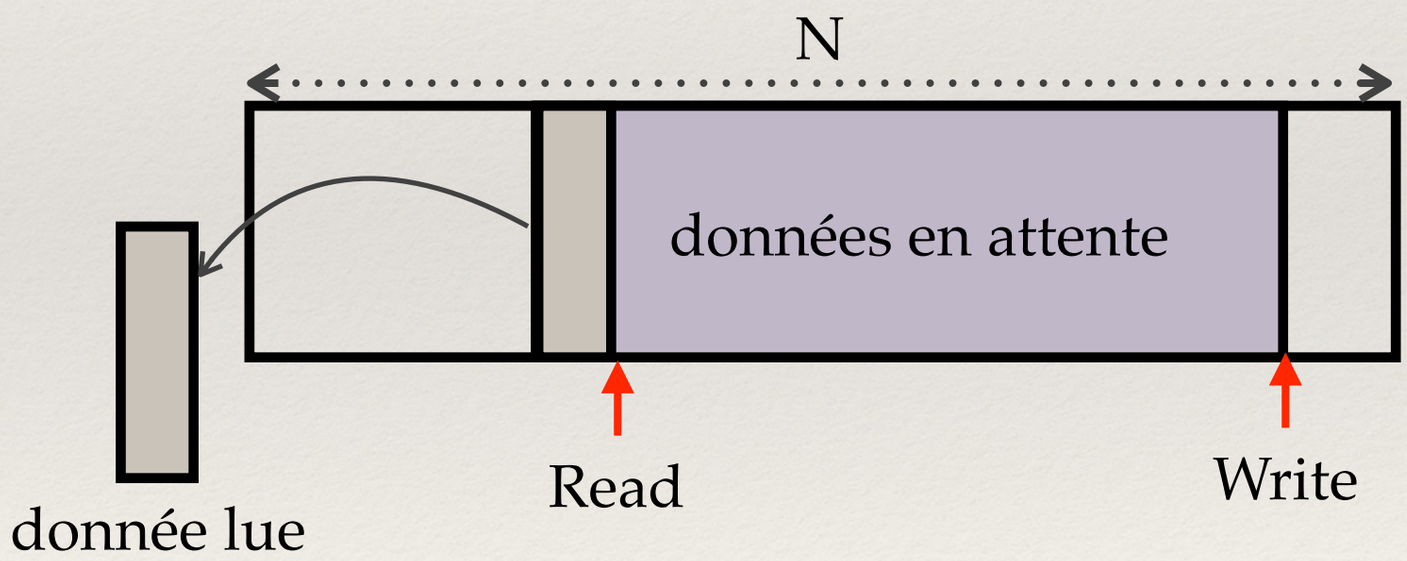
Structure de données:

- Table,
- Read: index de lecture
- Write: Index d'écriture



Gestion des index circulaire:

- modulo sur N
- retour à 0 en bout de table



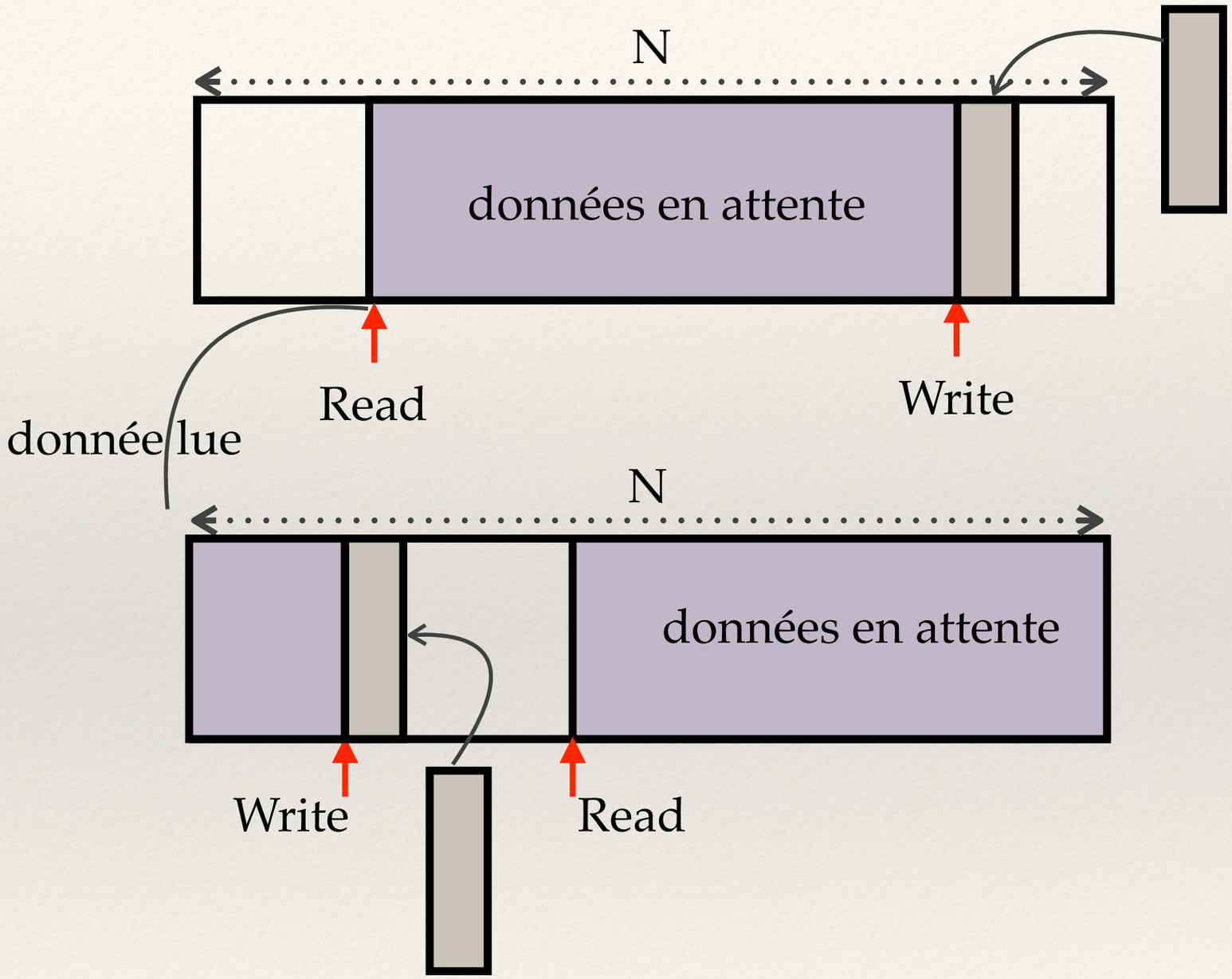
Queues: écriture (put)

Put() :

- Condition : NbElement <N
- Table[Write] := Data
- Write := Increment(Write)
- Return Success

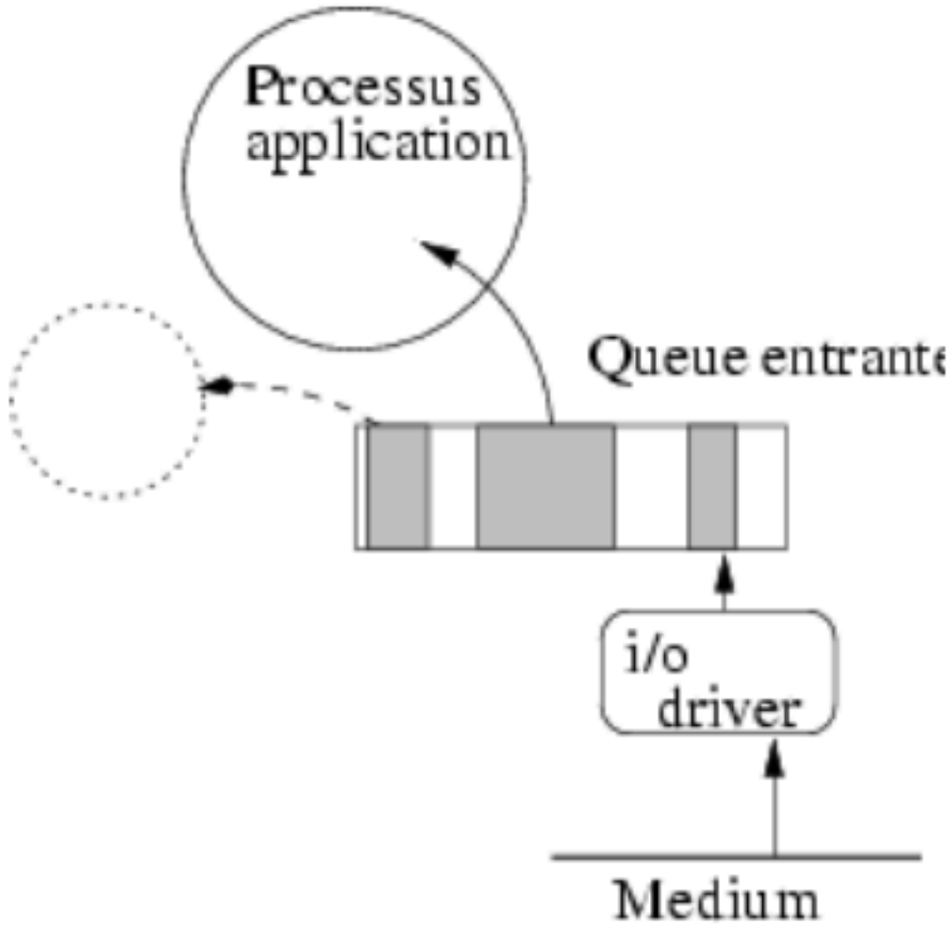
NbElement :

- If Write>Read then Write-Read
- else (N-Read) + Write



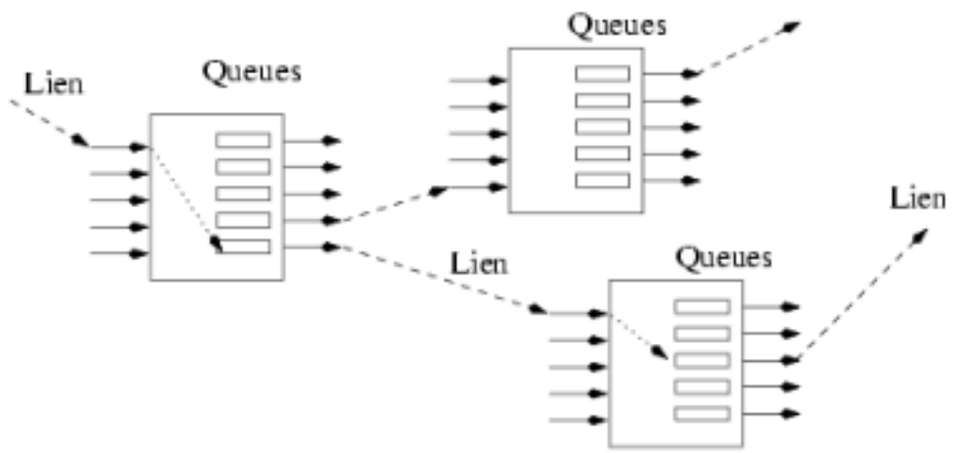
Queue réception

Les tampons entrant permettent d'attendre la disponibilité d'un processus, qui travaille, qui est bloqué, ou préempté par le système au bénéfice d'un autre processus.



Queues en acheminement

Le réseau ne réagit pas différemment en essayant de faire attendre des messages qui s'accumulent en direction d'une ligne. Les queues d'entrées ou de sorties permettent une certaine régulation du flux localement.

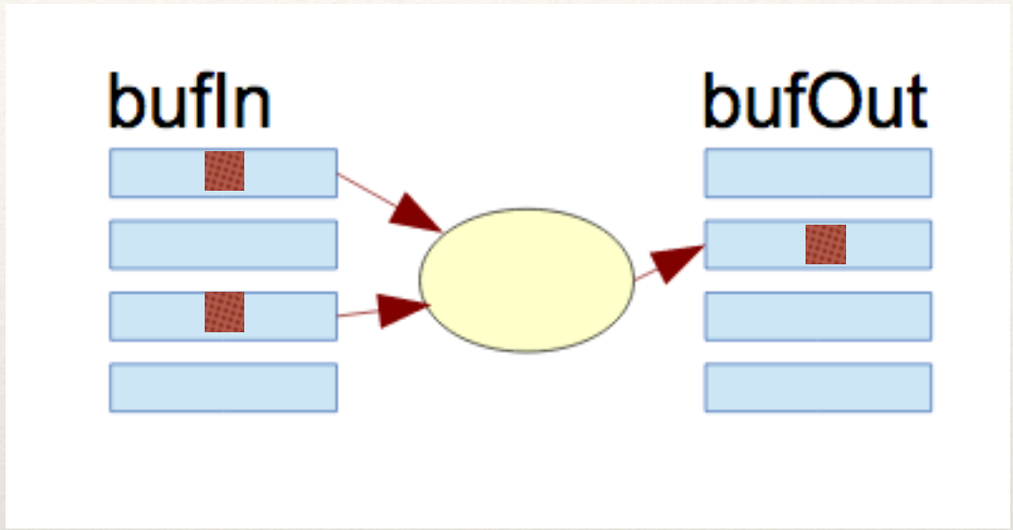


Acheminements sans queues

Cas d'un processus acheminant des messages:

A. Réception de données (bufIn)
B. Ré-émission vers l'aval (bufOut).

Il faut jeter les messages en excès si on n'a pas d'espace de stockage.



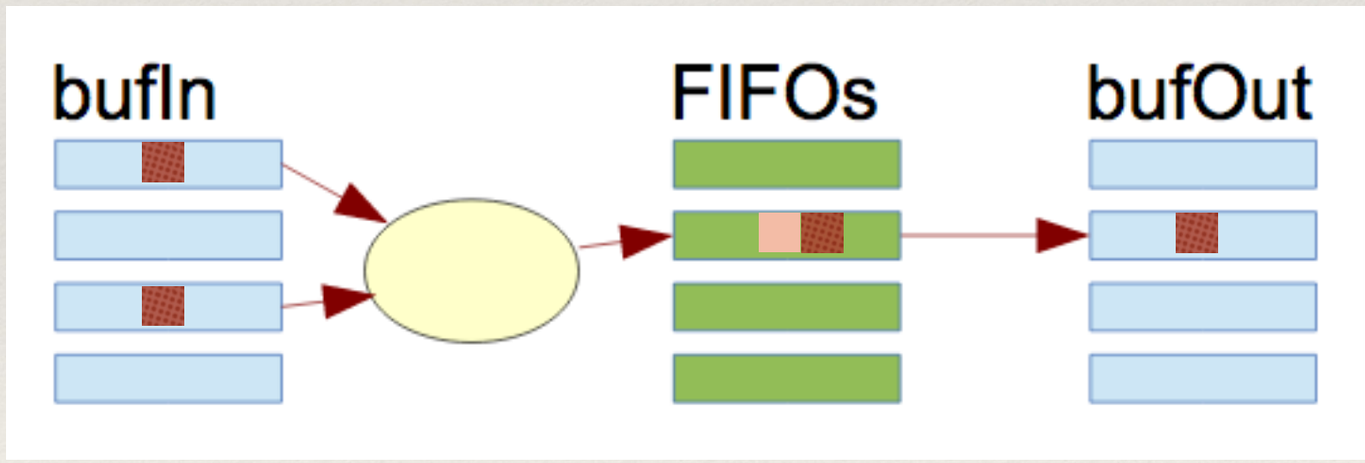
A. 2 messages B. 1 message

Acheminements avec queues

Tables de queues permettent d'éponger les excès de transits

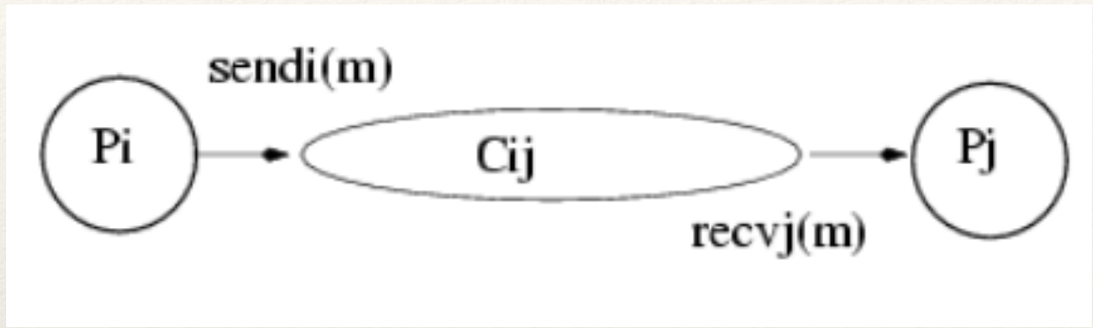
A. Réception de données (*bufIn*)
B. Insertion dans la FIFO de la sortie
C. Balayage des FIFOs et transmission vers l'aval (*bufOut*).

Les pertes peuvent toujours exister, mais les excès temporaires sont absorbés.



A. 2 messages B. 2 messages C. 1 message

Modèle de communication



Pi communique avec Pj:

- Cij est le canal « réseau » de communication directionnel:
  - sendi(m) : Opération d'expédition, bloquant ou non bloquant
  - receivej(m) : Operation de réception, bloquante ou non bloquante
- Les processus sont des automates exécutant des send et des receive
- Le canal a une capacité : plusieurs messages peuvent y être en transit.
- Le canal n'induit pas de contrainte de séquencement
- Des avatars peuvent s'y produire

Avatars de communication

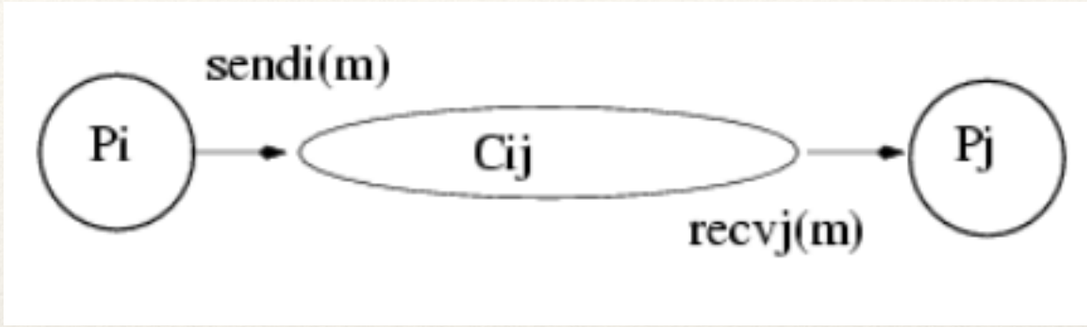
La communication peut être perturbée:

- Queues saturées qui entrainent des pertes de messages
- Perturbation de l'ordre (pertes, changements de routages)
- Agressions délibérées, modifications des contenus.

Modèle de canal programmé en Occam

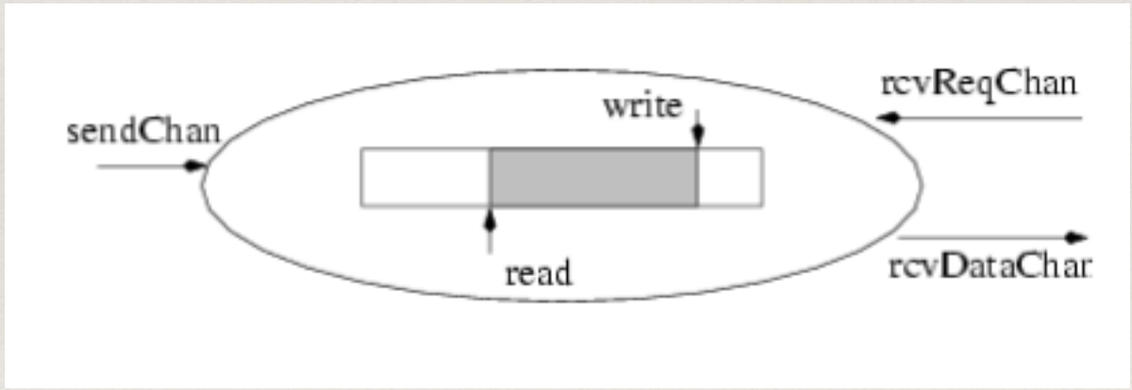
Implantation dans un processus:

- Symétrie et équité pour lecture et écriture
- Non déterminisme



Usage de la construction alternative:

- ALT n'admet que les communications entrantes!
- on contourne le problème avec des lectures en deux temps (deux canaux):
  - Requète (RcvReqChan)
  - Données (rcvDataChan)



FIFO en Occam (Types)

Définition message

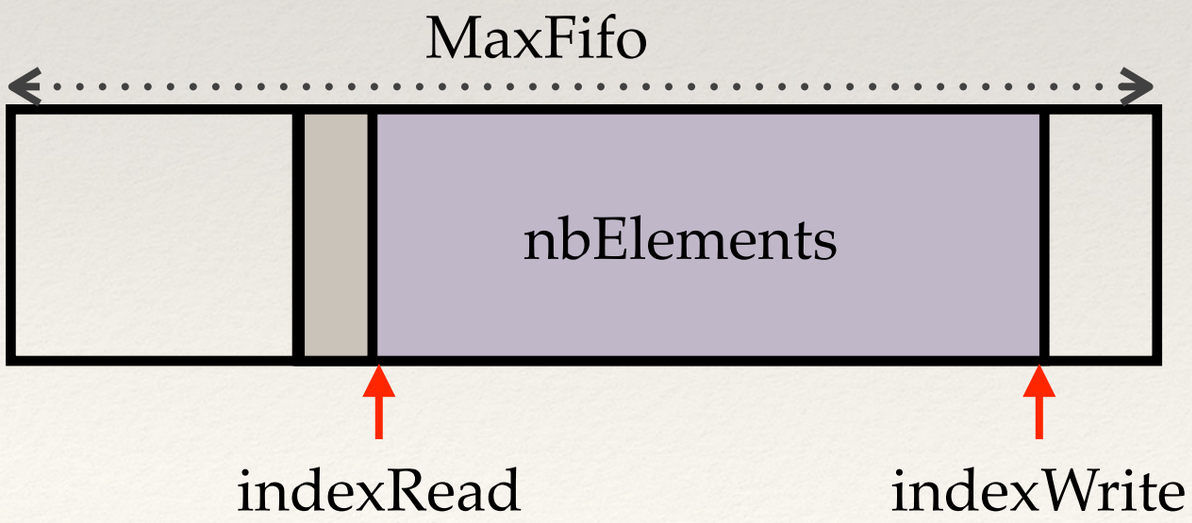
```
VAL INT MaxFifo IS 5:
DATA TYPE Message
  RECORD
    INT nodeId:
    BOOL conditionFound:
:
```

Définition protocole

```
PROTOCOL diam.proto
  CASE
    message; Message
    null; BYTE
:
```

Définition FIFO

```
DATA TYPE ReserveType
  RECORD
    [MaxFifo]Message Fifo:
    INT indexWrite,indexRead:
    INT nbElements:
```



FIFO en Occam (Utilités)

Création message

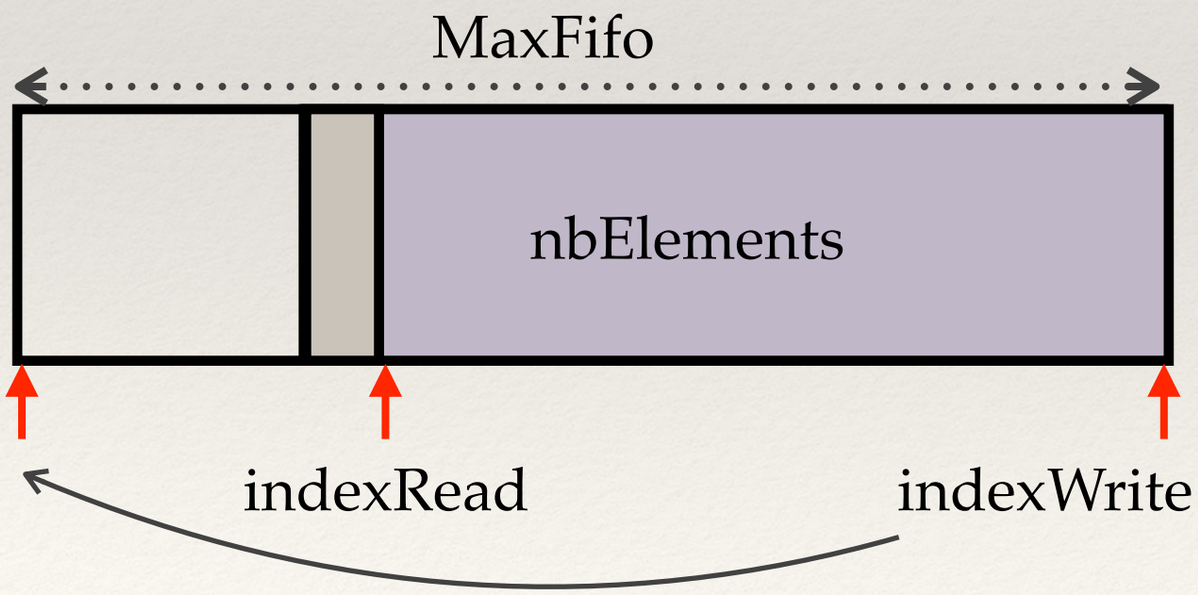
```
PROC InitMessage(Message monMessage, VAL INT Id)
  SEQ
    monMessage[nodeId]:=Id
    monMessage[conditionFound]:= FALSE
:
```

Création Fifo WriteIndex et nbElements

```
PROC InitFifo(ReserveType reserve)
  SEQ
    reserve[nbElements]:=0
    reserve[indexRead]:=0
    reserve[indexWrite]:=-1
:
```

Incrément Index

```
PROC NextIndex(INT currentIndex, nextIndex)
  SEQ
    nextIndex := currentIndex+1
    IF
      nextIndex=MaxFifo
        nextIndex:=0
      TRUE
        SKIP
```



FIFO en Occam (put)

De la place ?

On pousse Write et on écrit

```
PROC PutFifo(ReserveType reserve, Message message,BOOL success)
  INT nextIndex:
  IF
    reserve[nbElements]<MaxFifo
      SEQ
        NextIndex(reserve[indexWrite],nextIndex)
        reserve[Fifo][nextIndex] := message
        reserve[indexWrite] := nextIndex
        reserve[nbElements]:=  reserve[nbElements] +1
        success := TRUE
    TRUE
      success := FALSE
  :
  :
  :
```

FIFO en Occam (get)

FIFO vide ou pas ?

On prepare next Read Index,

- lecture message
- gestion nbElements

Renvoi d'un message

```
PROC GetFifo(ReserveType reserve, Message message,BOOL
success)
  INT nextIndex:
  Message nullMessage:
  IF
    reserve[nbElements]>0
      SEQ
        NextIndex(reserve[indexRead],nextIndex)
        message :=reserve[Fifo][reserve[indexRead]]
        reserve[indexRead] := nextIndex
        reserve[nbElements]:=  reserve[nbElements] -1
        success:= TRUE
    TRUE
      SEQ
        message:=nullMessage
        success := FALSE
:
```

Communication avec FIFO

```
PROC SYS(CHAN OF BYTE in,out,err)
  Message message:
  ReserveType fifo:
  BOOL success:
  INT dest:
  SEQ
    InitFifo(fifo)
```

1

```
SEQ i=0 FOR 10
  SEQ
    InitMessage(message,i)
    PutFifo(fifo, message, success)
    IF
      success
        out ! '+'
      TRUE
        out ! '-'
out!'*n'
```

2

```
SEQ i=0 FOR 10
  SEQ
    GetFifo(fifo, message, success)
    IF
      success
        SEQ
          dest := (INT message[nodeId])
          out.number(dest,0,out)
          out ! '*n'
      TRUE
        SKIP

+++++-----
```

3

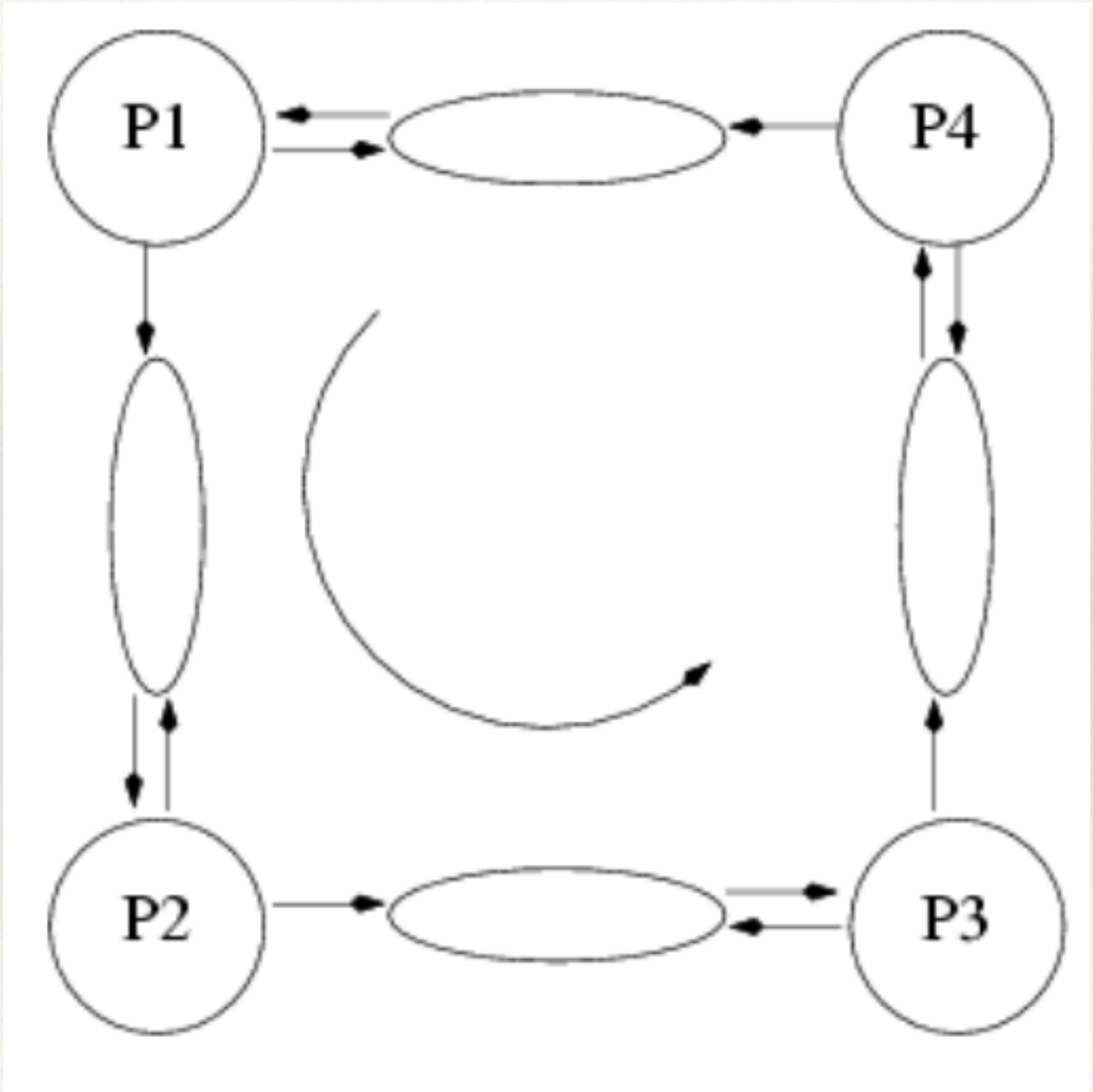
Exemple : anneau asynchrone

Structure

- 4 processus Node
- 4 processus CanalAsync
- 12 canaux Occam bloquants

Comportement

- Node libres d'avancer
- CanalAsync contraints par les FIFO internes



Algorithme : anneau asynchrone

- On assemble quatre processus par des canaux directionnels
- Chaque processus dispose d'un identificateur unique ID qu'il émet sur son canal sortant.
- Lors de la réception d'un identificateur rcvID le processus compare avec le sien.
  - si rcvID $>$ *i* ID, alors on ré-émet rcvID
  - si rcvID $<$ ID, alors on ne fait rien,
  - si rcvID $=$ ID, alors on devient *leader*

Le comportement est différent, les messages pouvant s'accumuler dans les nœuds en nombre arbitraire.

Part III

Réseaux et modélisation

Algorithmes distribués et Réseaux
III. Couches et protocoles

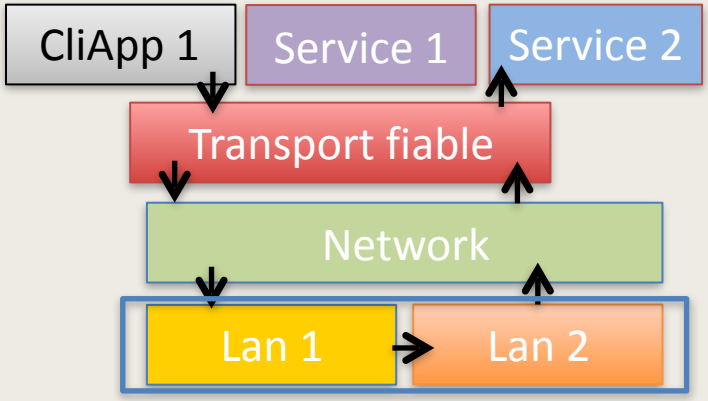
Bernard Pottier

Université de Brest

I. Couches basses 1+2 datagrammes, niveau 3

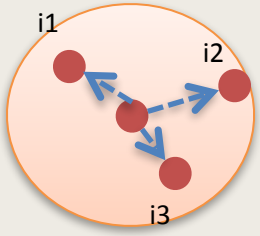
Structure des réseaux

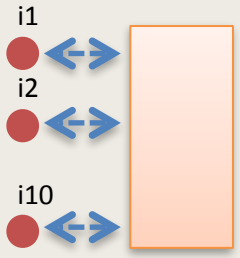
- Usage de « couches » (7)
  - Empilées, correspondantes
  - Évolutives
- Caractérisation par niveaux des services, exemple:
  - Transmission sur un réseau local
  - Adressage sur un réseau général (« internet »)
  - Fiabilité des opérations
  - Usage de services distants



Communication locale

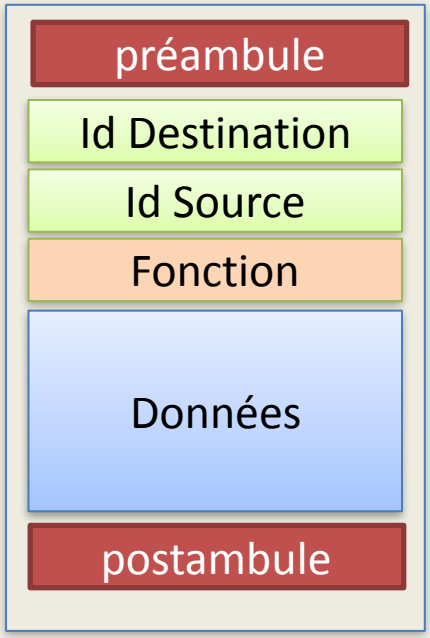
- Fonction de ***communication physique*** autour d'un ***medium partagé:***
  - Par diffusion : radio, cable coaxial
  - Par routage : aiguillage
- Notion d'identité locale
- Notion d'adressage entre nœuds ou groupe de nœuds (diffusion)





Trames de bas niveau

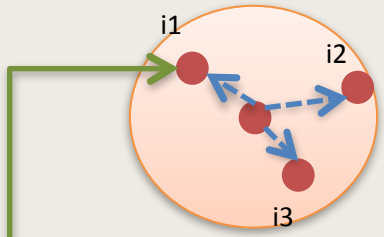
- Identification *locale* de la destination ou d'un groupe destinataire
- Identification *locale* de la source
- Identification de la fonction
- Empaquetage dans une trame adaptée au réseau local
  - Synchronisation et code contrôle



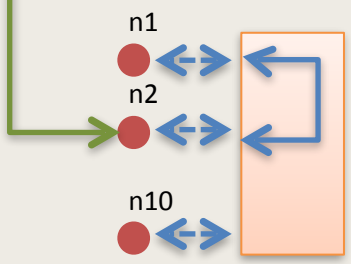
Adressage local

- L'identification *locale* sert aux partenaires d'un medium à échanger des messages
- La transmission src->dest peut concerner 1 ou plusieurs nœuds
- La réception se fait par filtrage des trames entrantes (i1, i2, ..)
- Les réseaux locaux sont reliés par des liaisons locales (i1-n2)

Routages



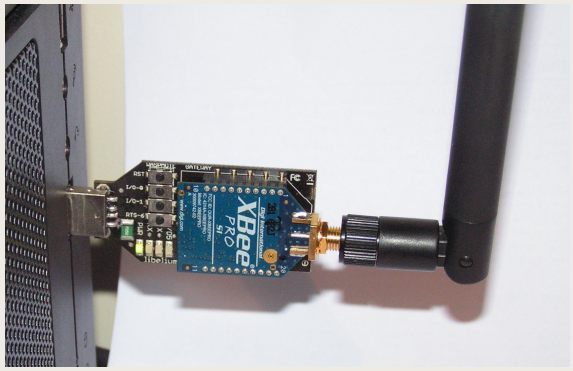
Radio:
1 émetteur, plusieurs récepteurs

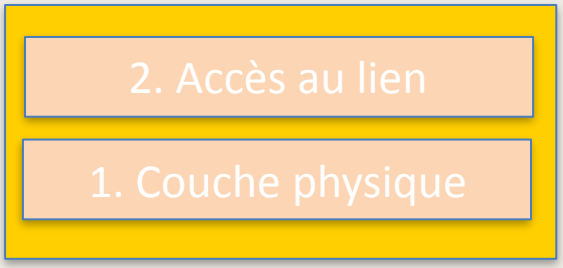


Aiguillage:
1 émetteur, 1 récepteur

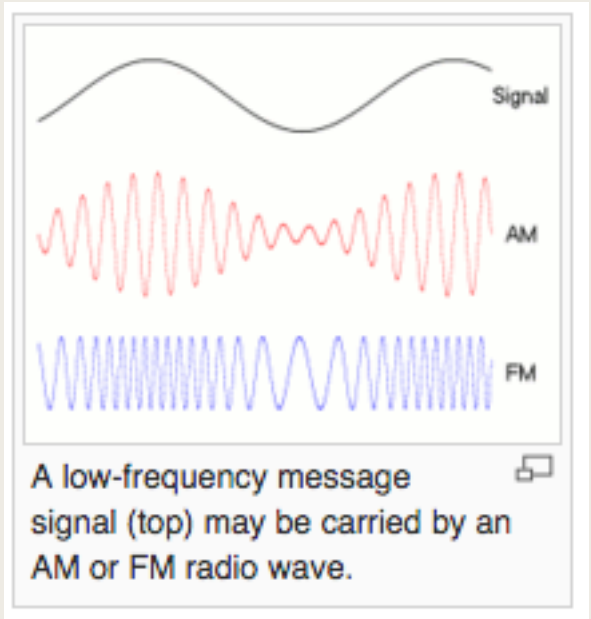
Couche 1: physique

- Systèmes matériels variés
- Modulations, fréquences
- Horloges de transmission
- Trains de bits effectifs et encodages décodages des données
- Interfaces matériels





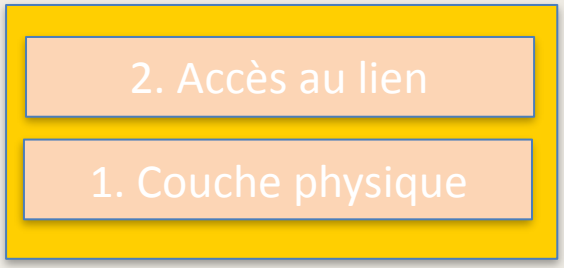
Protocoles de bas niveau



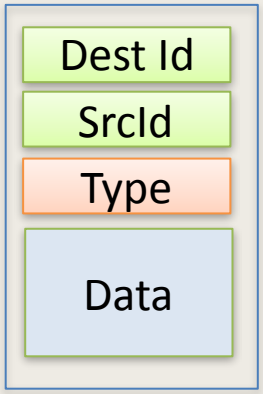
(de wikipedia)

Couche 2: accès au lien

- Cette couche définit les protocoles permettant de transférer les informations entre 2 nœuds locaux.
- Elle définit les adresses, les formats de paquets, la correction d'erreurs.

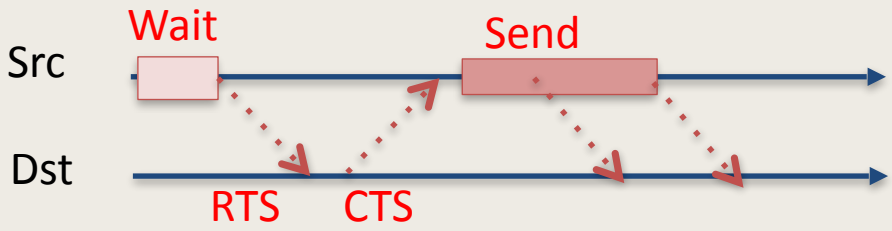


Protocoles de bas niveau



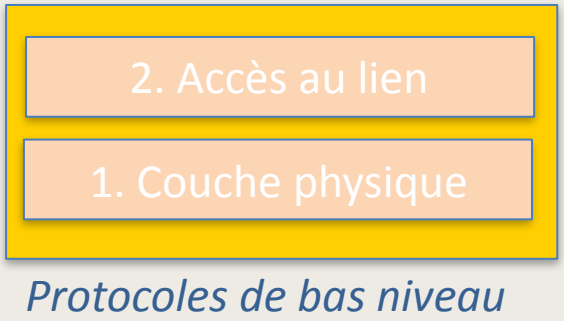
Trame locale

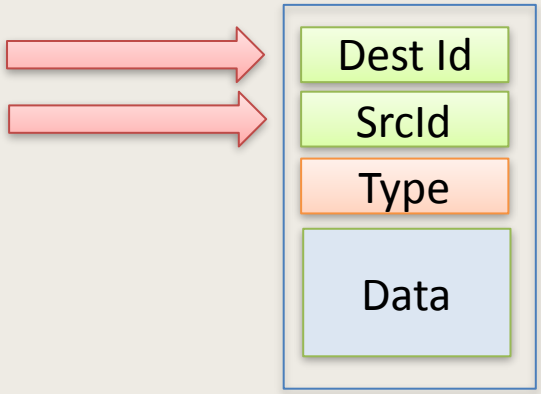
Protocole sur medium partagé



Couche 2: adresses statiques

- Numéros MAC (Medium access control):
  - Uniques, en principe
  - 48 bits (!), 22 pour le constructeur
  - Ethernet, 802.15.4, PPPoE, wifi





Routeur ADSL

```
Réseau :
========
  Adresse MAC Freebox           00:07:CB:B0:77:AE
  Adresse IP                    78.100.14.3
  Mode routeur                  Activé
  Adresse IP privée             192.168.0.100
  Serveur DHCP                  Activé
  Plage d'adresses dynamique    192.168.0.2 - 192.168.0.50

 Attributions dhcp :
 -------------------
  Adresse MAC           Adresse IP
  00:25:4B:92:A2:9C     192.168.0.2
  00:26:08:12:28:F6     192.168.0.4
```

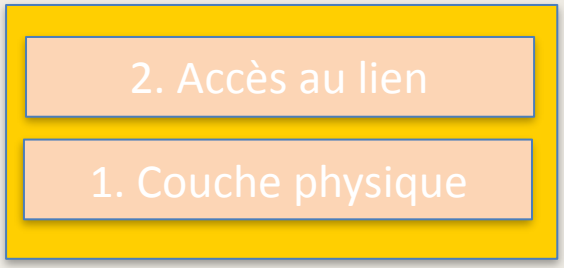
$ ifconfig

```
en0:
ether 0c:4d:e9:a4:91:90
inet 192.168.0.9 netmask 0xffffff00 broadcast 192.168.0.255
```

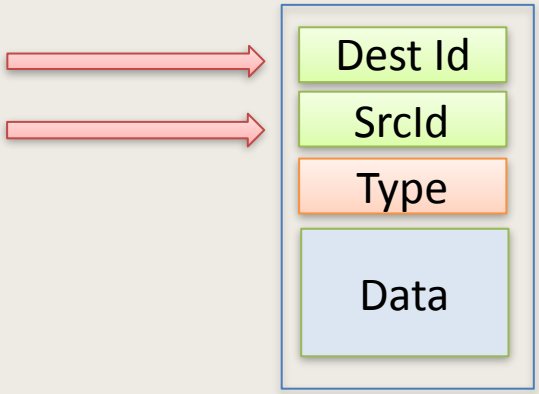
Couche 2: adresses dynamiques

- Acquisition d'adr. dynamique
  - Exemple, LLAP = 8 bits
  - Tirage au sort : adresse Idx
  - Sondage Idx : Enquiry sur Idx
  - Acquité ? Numéro invalide
  - Time out ? Numéro Idx valide acquis

boucle



Protocoles de bas niveau

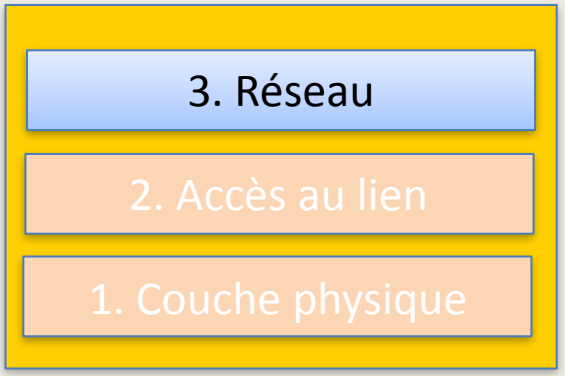


Trame locale

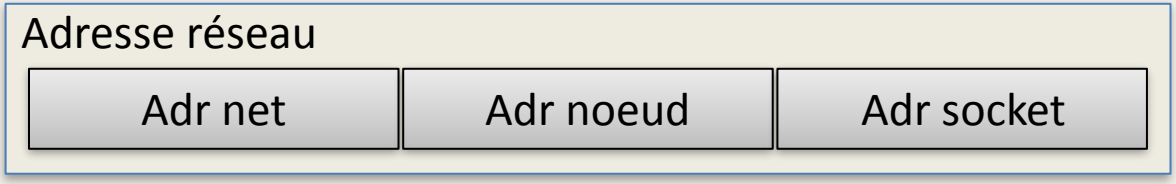
```
nbplkup
                hp LaserJet 4250:SNMP Agent                    65281.4:8
                hp LaserJet 4250:LaserWriter                   65281.4:158
                hp LaserJet 4250:HP LaserJet                   65281.4:157
```

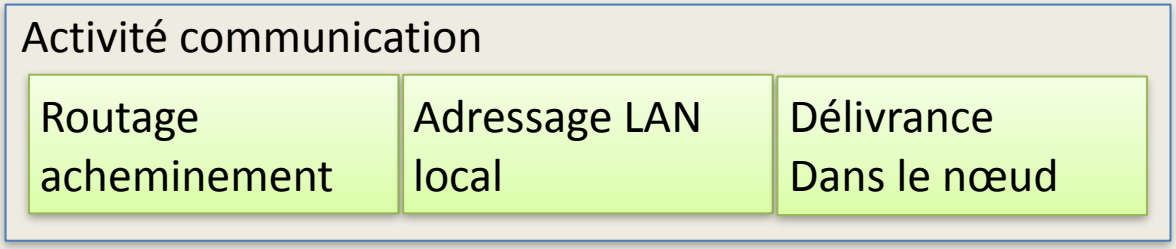
Couche 3: Réseau

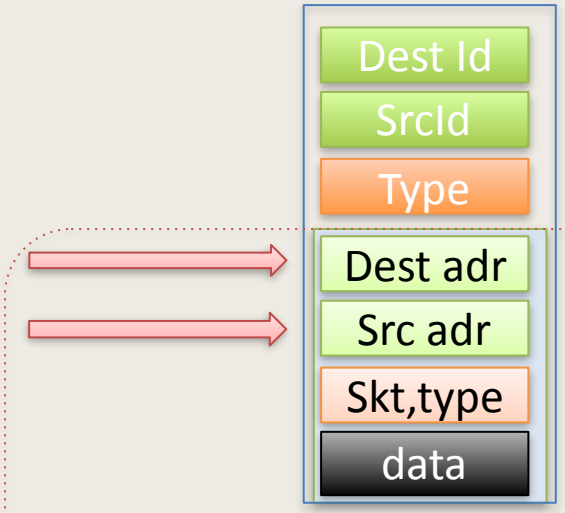
- Adresse logique, absolue
- Datagramme acheminé par 1+2
- Familles: Inet4, Inet6, Atalk,….
- Segmentations en réseau-socket
- Routages entre réseaux



Réseau utilisant 1+2

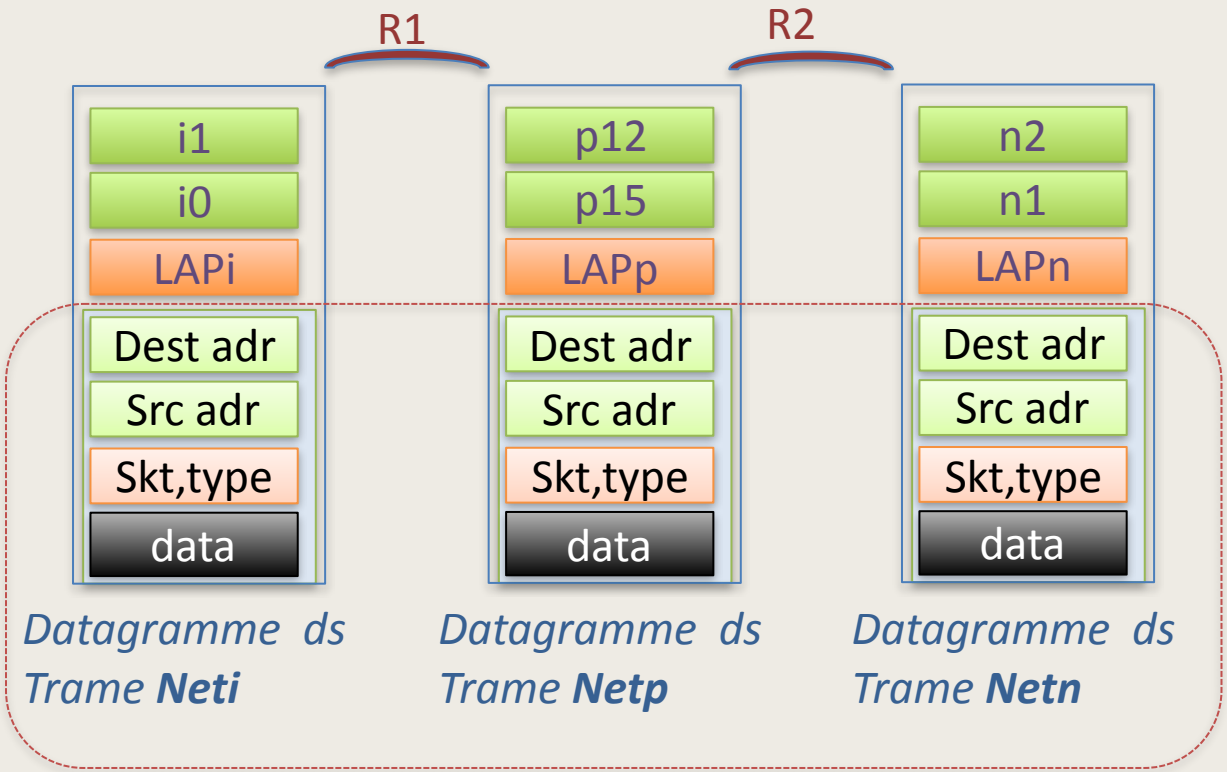


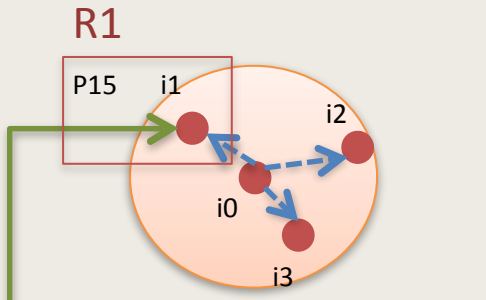




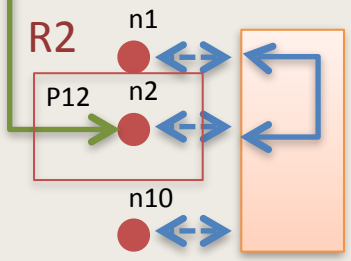
Datagramme ds Trame locale

Exemple: routage i->n





Routeur 1 Neti-Netp



Routeur 2 Netn-Netp

Exemples adresse inet

Adresse réseau

Adr net	Adr noeud	Adr socket

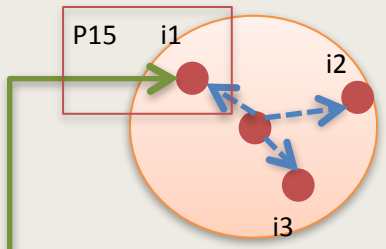
Appletalk

16 bits	8 bits	8 bits

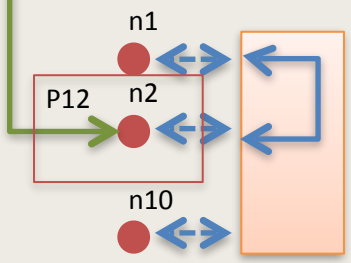
IPV4 = 193.52.16 . 33

X bits	32-X bits	16 bits

Routage réseau acheminement	Adresse locale	délivrance



Routeur 1 Neti-Netp



Routeur 2 Netn-Netp

Caractéristiques protocoles

- ***Non fiable***
- ***Paquets*** de taille contingentée
- Exemples:
  - IP Internet protocol, UDP User datagram.
  - DDP Datagram Delivery protocol

II. Couches « Transport » niveau 4

Fiabilité des opérations

- Les communications sur les réseaux niveau 1+2+3 sont ***non fiables***.
- La **garantie de bonne transmission** repose sur un **acquittement** fourni par le destinataire à la source.
  - Par des ***Transactions*** couplant émission et acquittement. Pas de rôle, symétrie X, Y.
  - Par des ***Flots*** (stream) continu, bidirectionnel, représentant une connexion fiable X <- -> Y

Couche 4 : autres fonctions

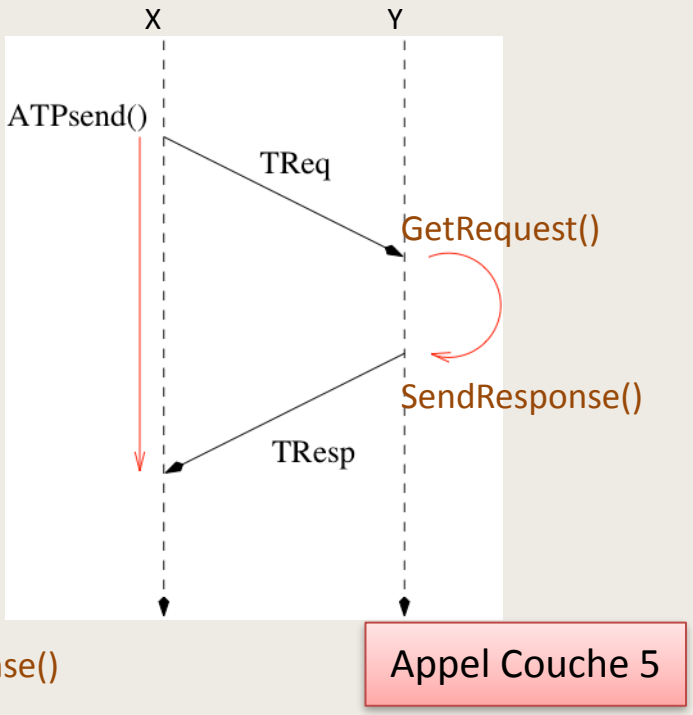
- On range aussi dans la couche 4 les utilités de gestion du réseau couplés directement à la couche 3 (datagrammes).
  - Protocoles de routages **RIP**, **RTMP**, …
  - Protocoles de nommages
  - Protocoles d'application non fiables (imprimantes, ..)

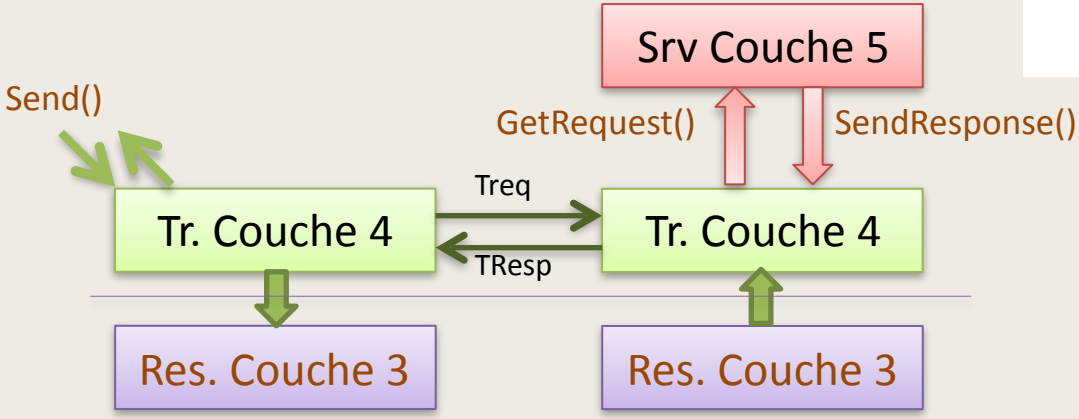
II.1 Transactions

- On nomme ***transaction*** une opération entre deux entités X et Y sur une réseau.
  - ***X émet une requète REQ[tid] vers Y***
  - ***Y expédie une réponse ACK[tid] à X***
  - ***L'échange possède un identificateur tid, numéro permettant de dissocier la transaction d'autres transactions éventuellement concurrentes.***
- A son achèvement, la procédure de ***transaction*** rend un status booléen : succès, ou échec.

Transactions

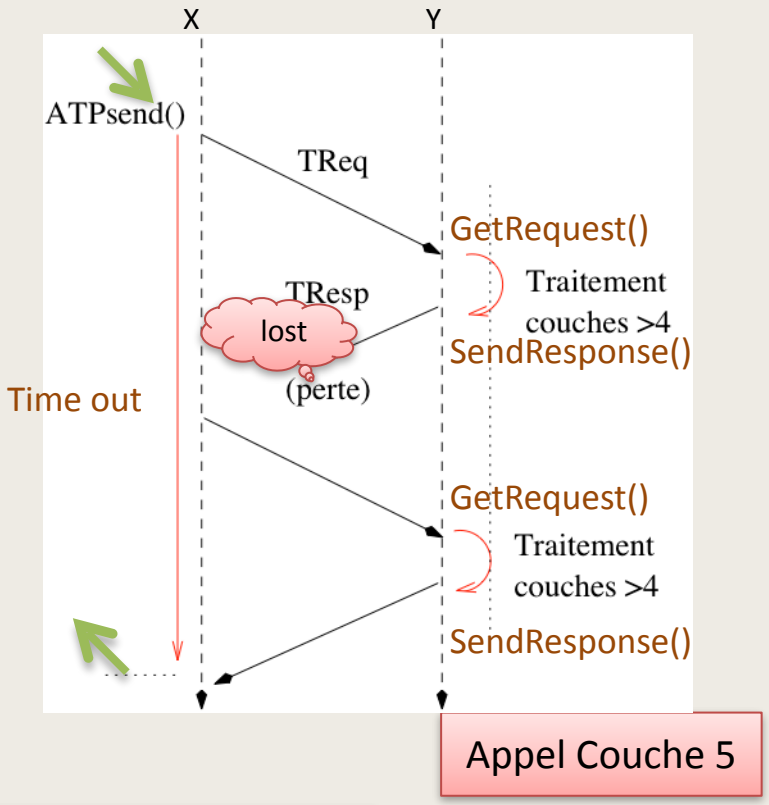
- **Transaction réussie**:
  - Appel procédural initial de l'émetteur Send()
  - Réception et traitement GetRequest(), SendResponse()
  - Fourniture d'un paquet réponse
  - Cloture de la transaction: booléen (Success) et réponse





Transactions (échec 1)

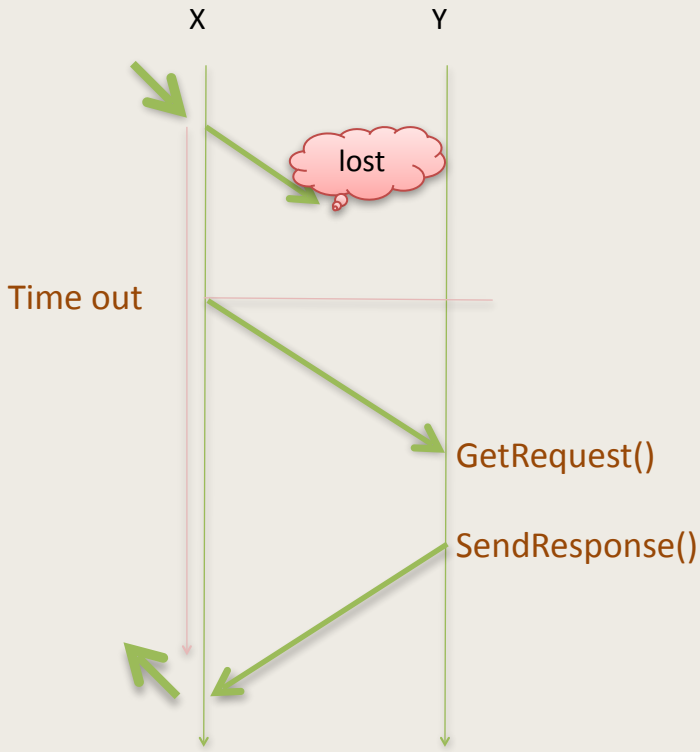
- **Transaction échecs et ré-essais**:
  - Treq reçu et traité
  - SendResponse() , paquet perdu
  - Time out, sans réponse
  - Treq ré-émis, reçu et traité
  - SendResponse() , paquet reçu
  - Cloture de la transaction: booléen (Success) et réponse



Bilan : 2 requètes reçues, **2 traitements** et 1 seule réponse.

Transactions (échec 2)

- **Transaction échecs et ré-essais**:
  - Treq perdu et ré-émis
  - Time out, sans réponse
  - Treq ré-émis, reçu et traité
  - SendResponse() , paquet reçu
  - Cloture de la transaction: booléen (Success) et réponse

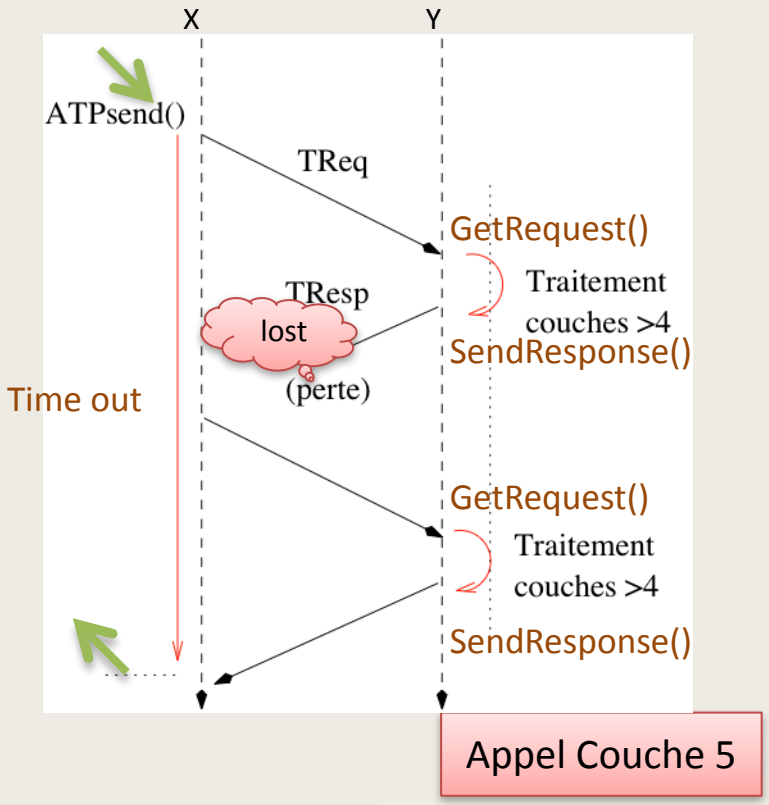


Appel Couche 5

Bilan : 1 requètes reçues, **1 traitement** et 1 seule réponse.

Transactions At Least Once (ALO)

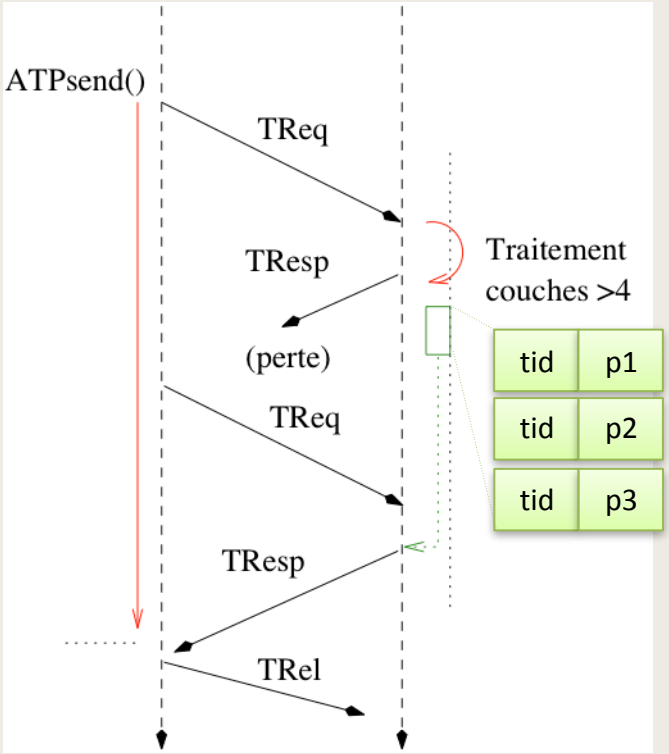
- **Stratégie**
  - Ré-éssais sans gestion des répétitions éventuelles
  - Convient à des situations où l'exécution est sans effet d'état
  - *Exemple : lire l'heure*



Bilan : n requètes reçues, **n traitements**
Et 1 réponse.

Transactions eXactlyOne (XO)

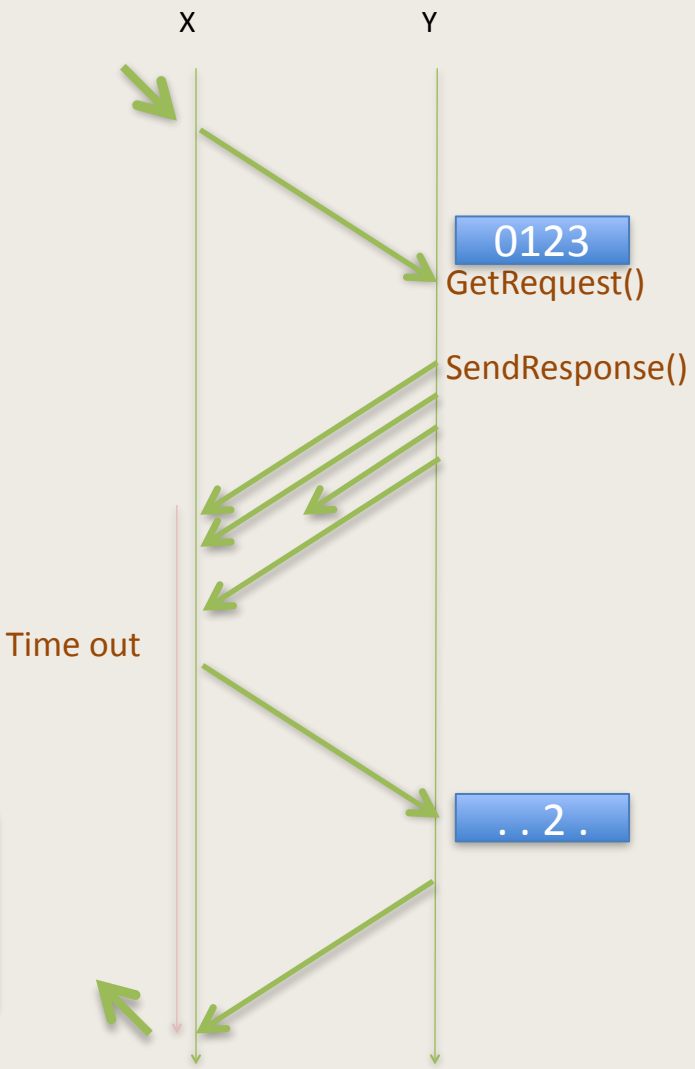
- **Stratégie**
  - Stockage des réponses dans une file, avec le TID
  - A réception, on cherche une réponse déjà traitée
  - Si elle existe on ré-émet sans traiter
  - Un message de libération permet de libérer ma réponse
  - *Exemple: transaction bancaire*



Bilan : n requètes reçues, **1 traitement**
Et 1 réponse.
Mémorisation de paquets et
Transaction plus lourde

Transactions multi-paquets

- **Transaction intégrant une table des paquets attendus**
  - Treq multi paquets
  - Time out,
  - Treq ré-émis, avec bit map attendu
  - Ré-émission des paquets
  - Cloture de la transaction: booléen (Success) et réponse



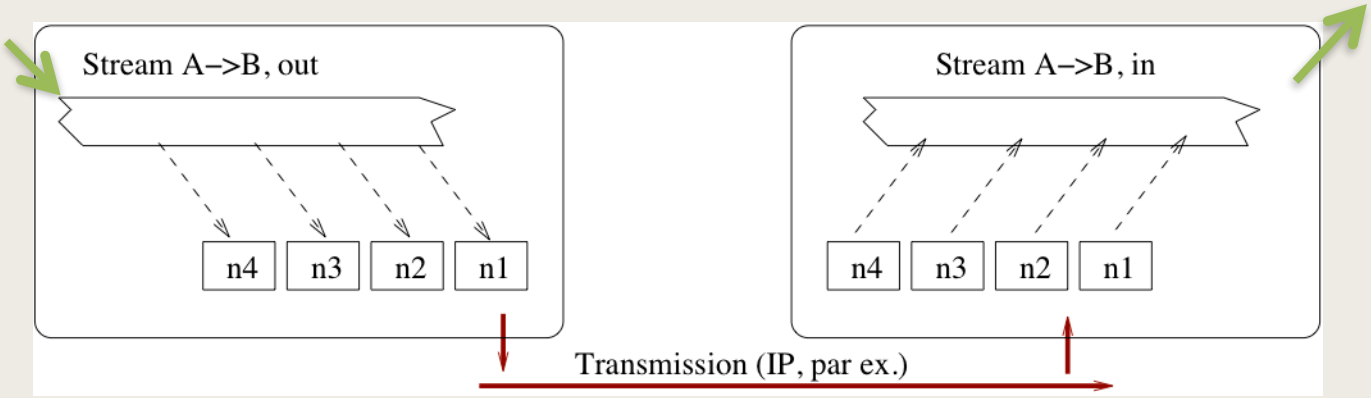
Bilan : 1 requète, **1 traitement**
et plusieurs réponses avec ré-émissions.
Convient aux lectures/écritures de données

II.2 Streams

- On nomme ***Stream, ou flot,*** une *connexion* fiable entre deux entités X et Y sur une réseau.
  - ***La connexion représente un flot de données non structuré (Cf pipe ou stream Unix)***
  - ***Les procédures sont de type read() et write() de lots de données sans contrainte de paquets***
  - ***Le mécanisme interne gère les pertes de paquets en requérant des ré-émissions.***
  - ***Il gère la régulation de flux conjointement avel l'anticipation des transmissions***
- Des implémentations connues sont TCP et ADSP.

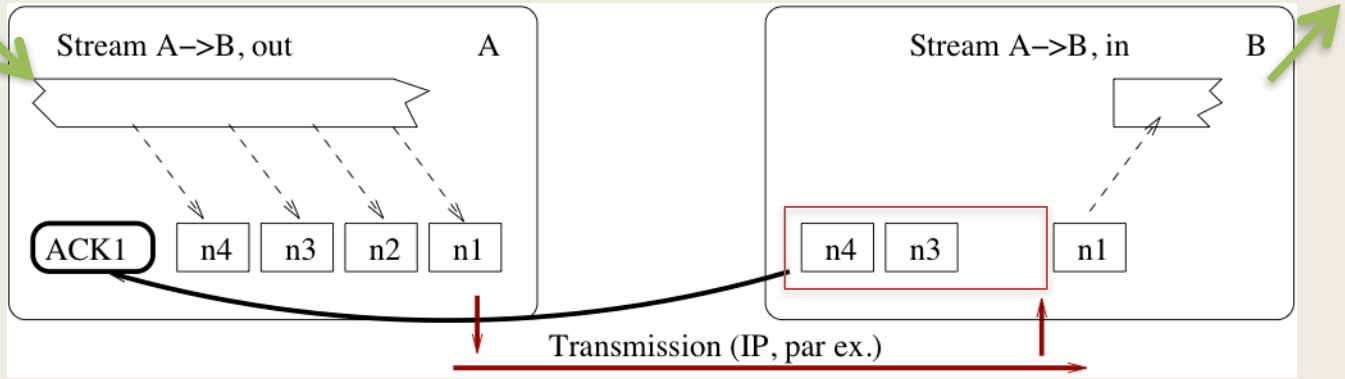
Fractionnement des données

- **La transmission est opérée par *paquets***:
  - L'émetteur fractionne et transmet
  - Le récepteur tamponne les paquets reçus
  - Il peut délivrer la zone fiable au lecteur
  - Les transmissions peuvent avoir lieu *avant la délivrance* : recouvrement des communications et de l'exécution



Récupération des erreurs

- **Acquittements** :
  - La voie de retour permet de notifier l'Identité Id du dernier paquet reçu fiablement.
  - A échéance d'un time out, les paquets manquant sont ré-émis jusque cet Id



Régulation de flux

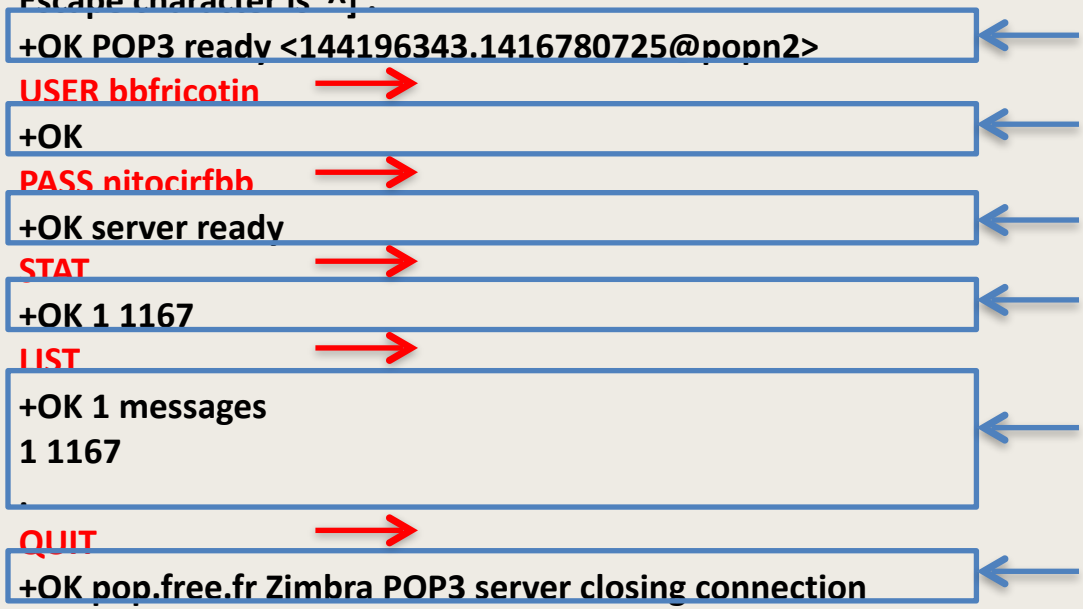
- **Présence d'un compteur *CPT* autorisant des émissions anticipées**:
  - X émet vers Y
    - Si CPT > 0, alors X peut émettre un paquet
    - Il doit alors décrémenter CPT
  - Les acquittements de Y permettent à X de mettre à jour l'Id du dernier paquet reçu de manière fiable
    - Il fait remonter CPT en fonction des progrès de cet Id
- La fenêtre d'émission donnée par CPT permet d'émettre en avance du besoin. C'est une estimation distribuée de la place disponible chez Y.

Comparaison Stream/Transaction

- Stream équivalent au stream Unix: **read()** et **write()**, unifiable avec pipes et fichiers
- Anticipation = recouvrement des communications
- Pas de sémantique, on doit analyser et plaquer les données sur le stream.
- La plupart des applications distribuées sont définies par des protocoles proches des transactions

Exemple sur un stream via telnet

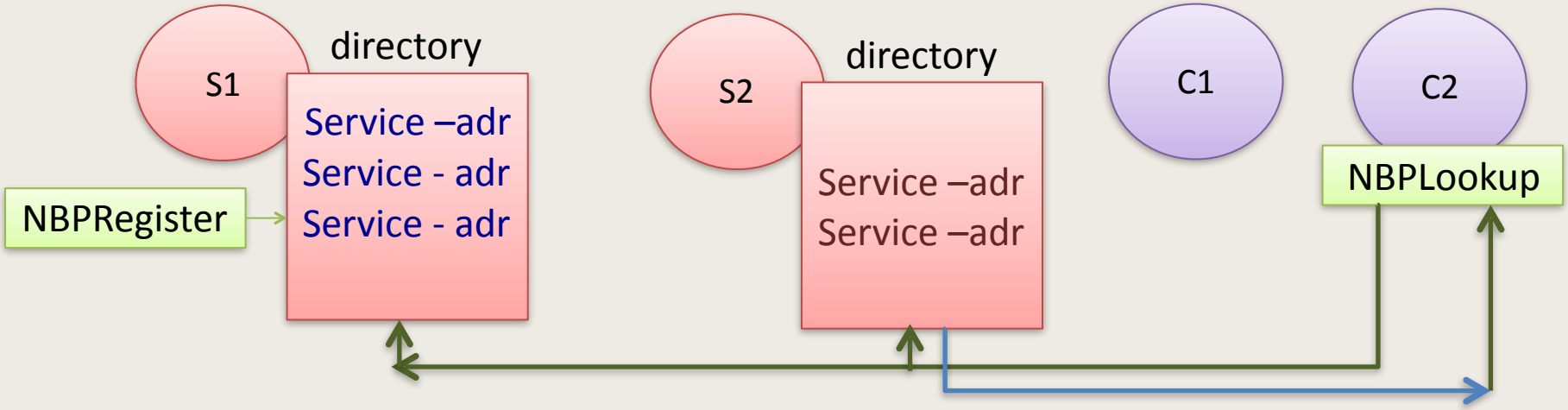
```
bash-3.2$ telnet pop3.free.fr pop3
Trying 212.27.48.3...
Connected to pop.free.fr.
Escape character is '^]'.
+OK POP3 ready <144196343.1416780725@popn2>
USER bbfricotin
+OK
PASS nitocirfbb
+OK server ready
STAT
+OK 1 1167
LIST
+OK 1 messages
1 1167
.
QUIT
+OK pop.free.fr Zimbra POP3 server closing connection
Connection closed by foreign host.
bash-3.2$
```



III. Couches 5, 6, et 7
session – présentation - application

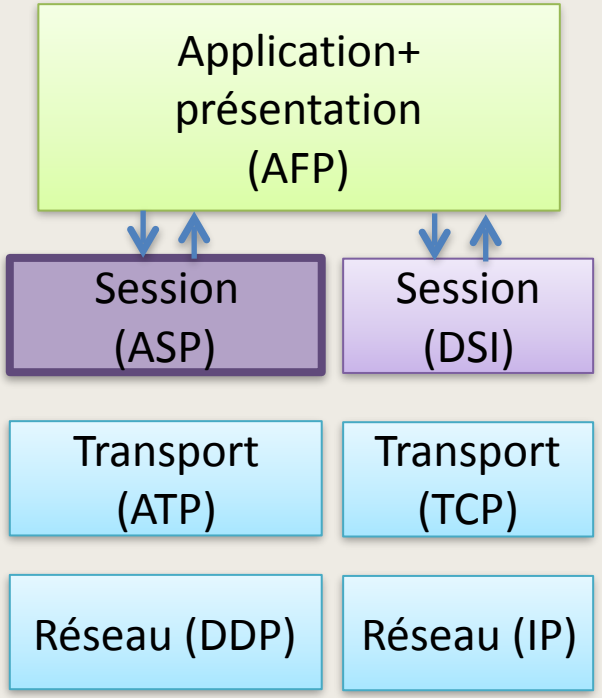
Nommage distribué : NBP

- Ancêtre des protocoles Bonjour, Avahi,… Zeroconf
- Pas d'administration, enregistrement des entités,
- Interrogation distribuée



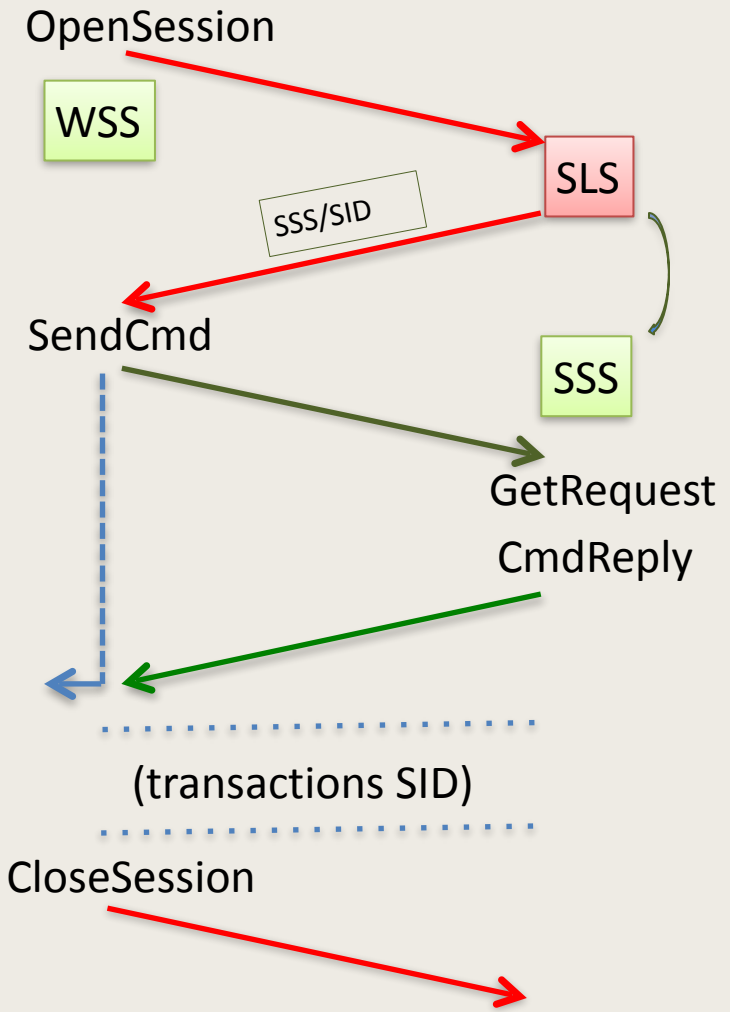
Session

- Une session est une *relation* établie entre deux partenaires sur le réseau.
- Cette relation est dissymétrique, elle recoupe la notion de *client-serveur*
- Elle s'appuie sur la notion de *sockets*, ouverts par un dialogue préalable, et maintenu par un *contrôle de vivacité*.
- *Non supporté* dans la pile TCP, voir plutôt OSI et ASP, ou des Appl. Prog. Int. (API, NetBIOS, RPC)
- Exemple d'usage : Apple Filing Protocol (AFP, porté sur ATP et TCP)



Ouverture de session

- Elles reposent sur 3 sockets
  - SLS, Session Listening Socket,
  - WSS Workstation Session Socket.
  - SSS Server Session Socket,
- Opérations du *client*
  - SendCommand(), GetStatus(),
- Opérations du *serveur*
  - GetSession()
  - GetRequest(), CommandReply(),
  - Attention()
  - Close



Résolution de noms

- La recherche de services est fondée sur des *noms*, permettant de retrouver une *adresse*.
- La résolution peut utiliser plusieurs mécanisme dans un ordre arbitraire

```
# /etc/nsswitch.conf
# `info libc "Name Service Switch"' for information about this file.

passwd:         compat
group:          compat
shadow:         compat

hosts:          files nis mdns4_minimal [NOTFOUND=return] dns
networks:       files nis
automount:      files nis

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis
```

Ordre de résolution du fichier au DNS

Résolution des noms
getaddrinfo()

```
int nameResolver(char *name,char *proto, char *ip)
{
    int sockfd;
    struct addrinfo hints, *servinfo, *p;
    struct sockaddr_in *h;
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    if ( (getaddrinfo( name , proto , &hints , &servinfo)) != 0)
        return 1;
    if ((p = servinfo) != NULL)
    {
        h = (struct sockaddr_in *) p->ai_addr;
        strcpy(ip , inet_ntoa( h->sin_addr ) );
        printf("IP %d\n";ip);
    }
    freeaddrinfo(servinfo); // all done with this structure
    return 0;}
```

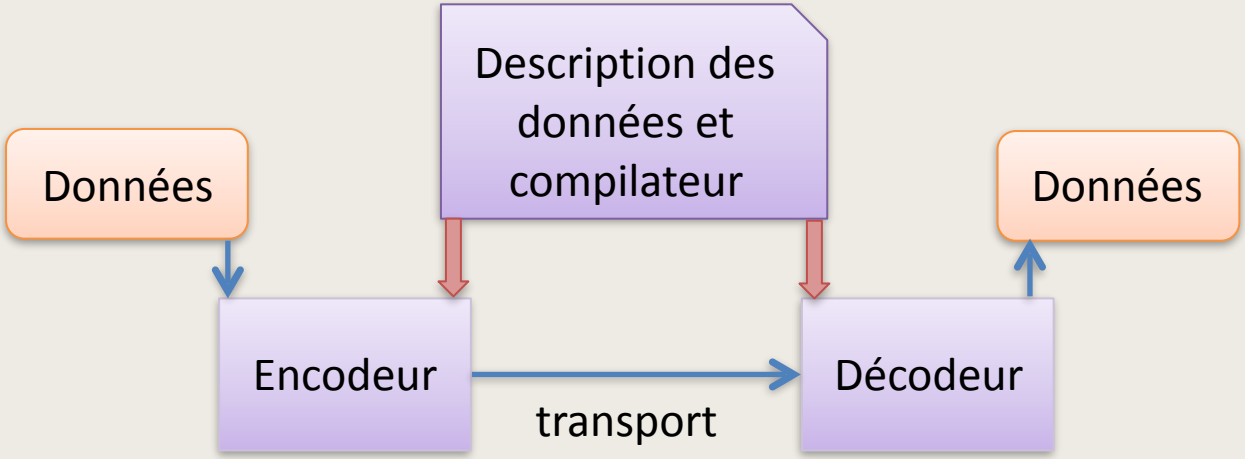
```
 if (argc >1) portName = (argv[1]); else portName = "pop3" ;
 if (argc >2)  name = argv[2]; else name = "www.univ-brest.fr";
 nameResolver(name,portName, ipAddress);
```

Présentation

- Il ne suffit pas de pouvoir échanger des données pour pouvoir communiquer:
  - Les processeurs représentent les données de manière différente (big-endian/little-endian)
  - Les données ont des tailles différentes (16/32/64 bits)
  - Les langages et compilateurs représentent les données de manière différente.
- La couche présentation a la responsabilité de *qualifier strictement les données* et de garantir un *accès correct*.

Présentation: outils

- Usages de langages de spécification et de compilateurs dédiés: Abstract Syntax Number 1 (ASN1), PROTOCOL chez Occam



Présentation: outils

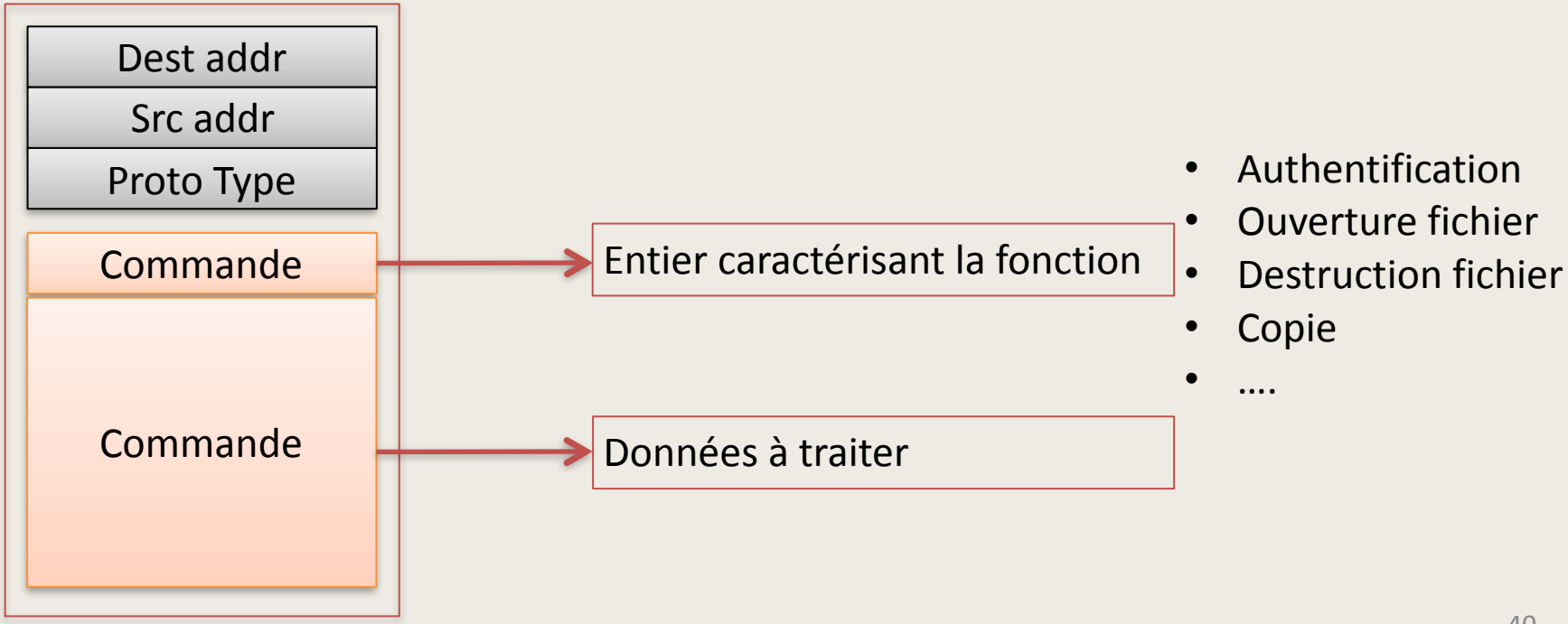
- *A la main :*
  - Spécification des données dans un standard:
    i : int16, b: byte, s8 : string[8], etc…
  - Construction des paquets, et relecture par des procédures *variables -> paquet -> variables*

  - *Htons, ntohs: conversions de short host et network*
  - *Htonl, ntohl: long (32 bits)*

```
NAME
     htonl, htons, ntohl, ntohs --

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <arpa/inet.h>

     uint32_t
     htonl(uint32_t hostlong);

     uint16_t
     htons(uint16_t hostshort);

     uint32_t
     ntohl(uint32_t netlong);

     uint16_t
     ntohs(uint16_t netshort);
```

Application : paquet requète

- *Les paquets sont lus par dessus la couche session : GetRequest(), ou lecture sur un socket SSS.*



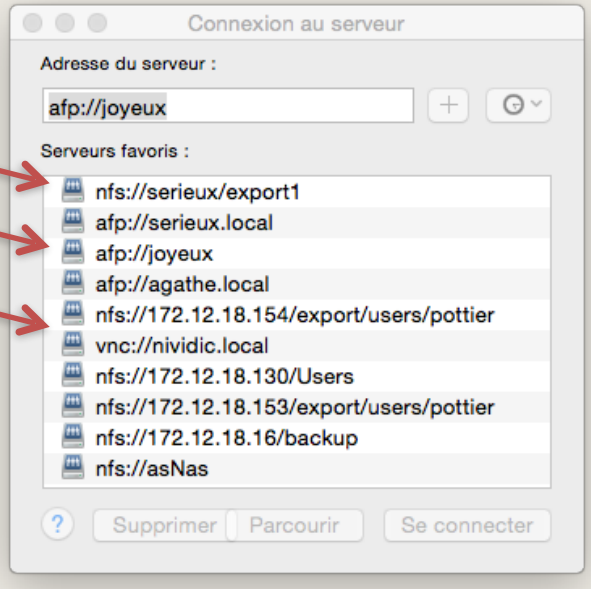
Application : service fichier

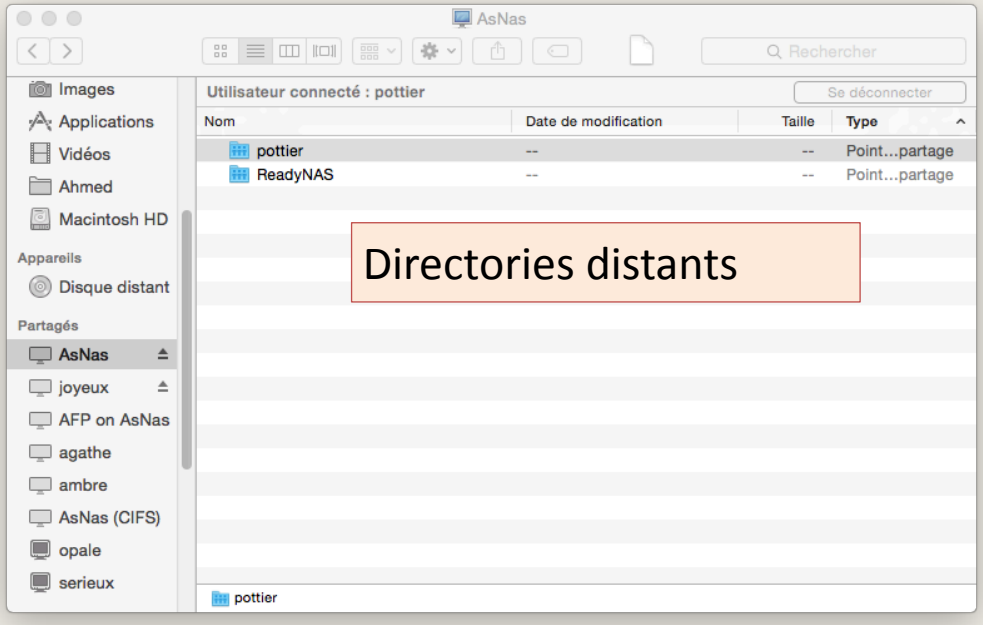
- **NFS** : *Network File System*, interprétation locale du système de fichier et échanges de blocs entre clients et serveurs. (Linux)
- **AFP**: *Appletalk File Protocol*, interprétation distante du système de fichier et *commandes-réponses* entre clients et serveurs. (Apple et Netatalk)
- **SMB**: *Server Message Block*, interprétation distante du système de fichier et API entre clients et serveurs. (Microsoft, Apple, Linux)

Application : interface user

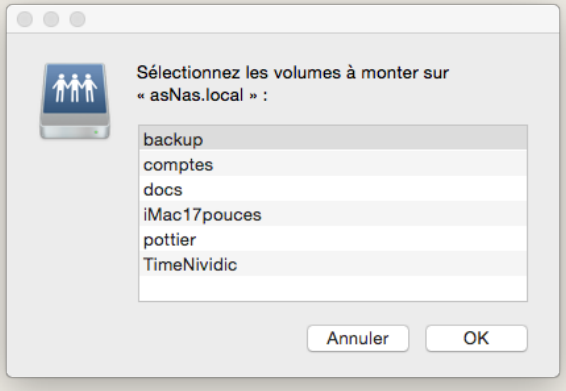
Interface service réseaux MacOS

- Montages NFS
- Montages AFP
- Serveurs d'écrans VNC
- Montage SMB/CIFS

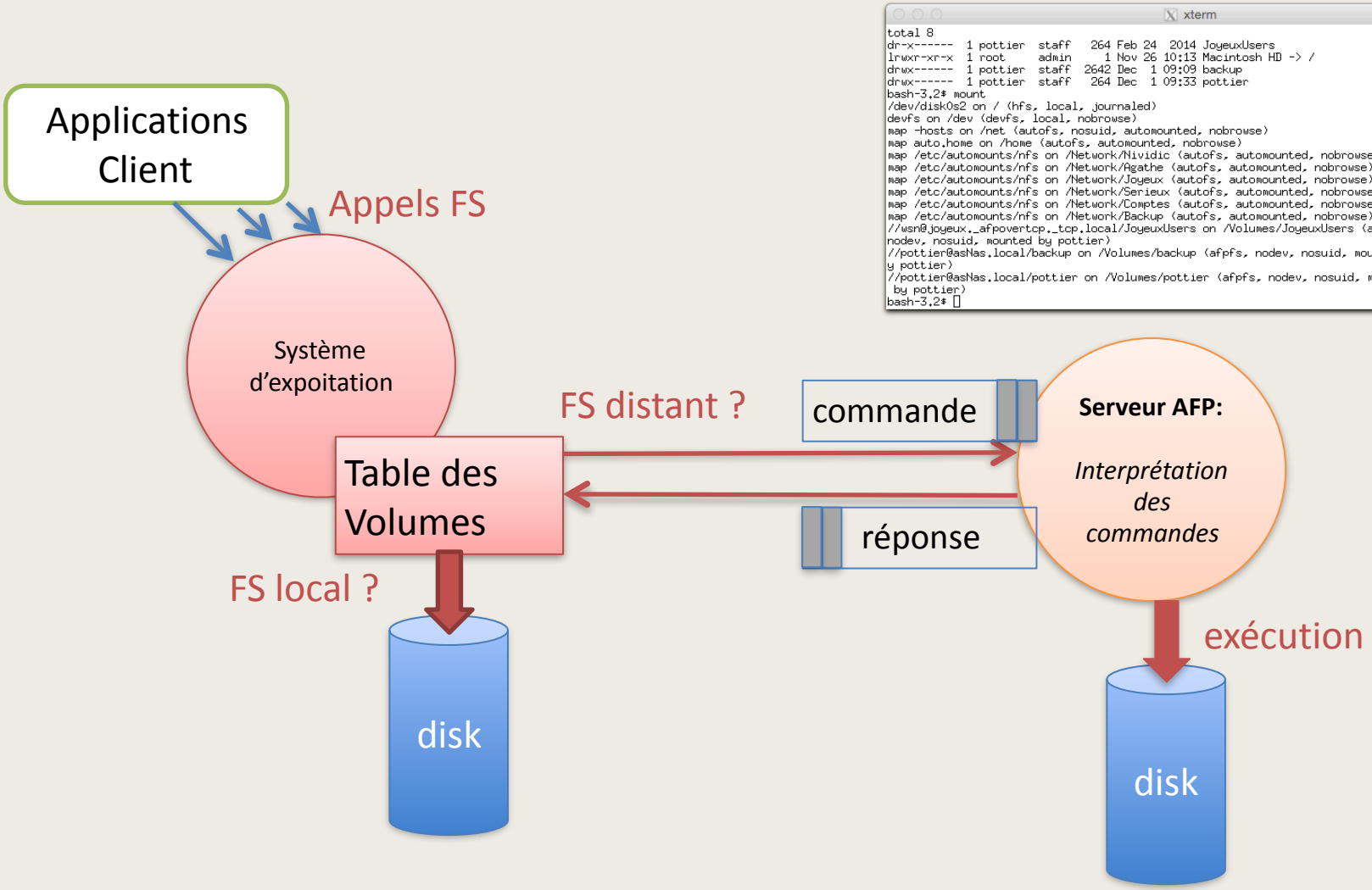




Directories distants

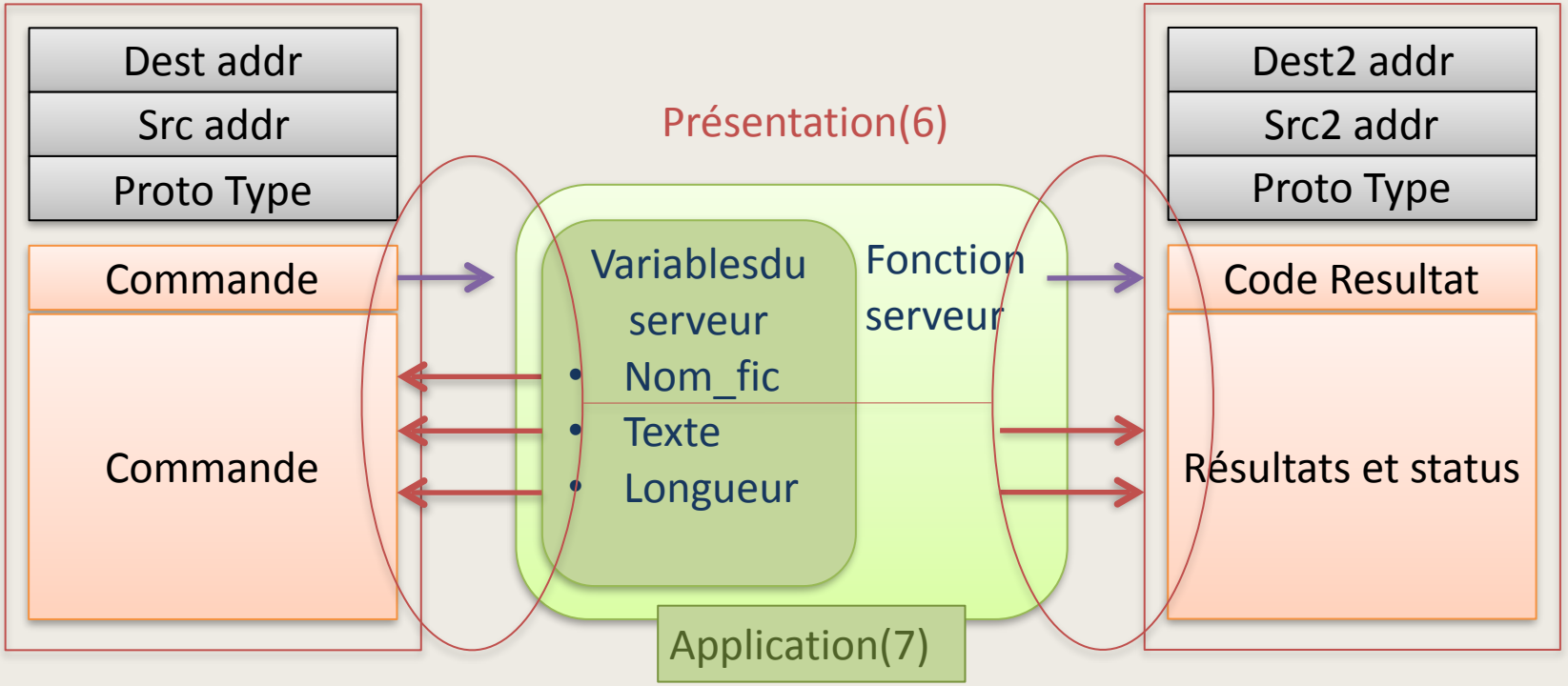


Application : interprétation FS



Application : traitement requète

- *Les paquets sont analysés et convertis en variables du serveur, puis celui ci traite.*
- *Les variables sont paquetées dans la réponse*



Application : branchement AFP

```
AFPCmd *afp_switch = preauth_switch;

AFPCmd postauth_switch[] = {
    NULL, afp_bytelock, afp_closevol, afp_closedir,
    afp_closefork, afp_copyfile, afp_createdir, afp_createfile, /*   0 -   7 */
    afp_delete, afp_enumerate, afp_flush, afp_flushfork,
    afp_null, afp_null, afp_getforkparams, afp_getsrvrinfo,     /*   8 -  15 */
    afp_getsrvrparms, afp_getvolparams, afp_login, afp_logincont,
    afp_logout, afp_mapid, afp_mapname, afp_moveandrename,      /*  16 -  23 */
    afp_openvol, afp_opendir, afp_openfork, afp_read,
    afp_rename, afp_setdirparams, afp_setfilparams, afp_setforkparams,
    /*  24 -  31 */
    afp_setvolparams, afp_write, afp_getfildirparams, afp_setfildirparams,
    afp_changepw, afp_getuserinfo, afp_getsrvrmesg, afp_createid, /*  32 -  39 */
    afp_deleteid, afp_resolveid, afp_exchangefiles, afp_catsearch,
    afp_null, afp_null, afp_null, afp_null,                     /*  40 -  47 */
    afp_opendt, afp_closedt, afp_null, afp_geticon,
    afp_geticoninfo, afp_addappl, afp_rmvappl, afp_getappl,     /*  48 -  55 */
    afp_addcomment, afp_rmvcomment, afp_getcomment, NULL,
    NULL, NULL, NULL, NULL,                                     /*  56 -  63 */
    NULL, NULL, NULL, NULL,
    NULL, NULL, NULL, NULL,                                     /*  64 -  71 */
    NULL, NULL, NULL, NULL,
    NULL, NULL, afp_syncdir, afp_syncfork,      /*  72 -  79 */
    NULL, NULL, NULL, NULL,
```

Contrôle AFP

```
while ((reply = asp_getrequest(asp))) {
    if (reload_request) {
        reload_request = 0;
        load_volumes(child);
    }
    switch (reply) {
    case ASPFUNC_CLOSE :
        afp_asp_close(obj);
        LOG(log_info, logtype_afpd, "done" );
        return;
        break;

    case ASPFUNC_CMD :
        func = (u_char) asp->commands[0];
        if ( afp_switch[ func ] != NULL ) {
            /*
             * The function called from afp_switch is expected to
             * read its parameters out of buf, put its
             * results in replybuf (updating rbuflen), and
             * return an error code.
            */
            asp->datalen = ASP_DATASIZ;
            reply = (*afp_switch[ func ])(obj,
                                          asp->commands, asp->cmdlen,
                                          asp->data, &asp->datalen);
        } else {
            LOG(log_error, logtype_afpd, "bad function %X", func );
            asp->datalen = 0;
            reply = AFPERR_NOOP;
        }
        if ( asp_cmdreply( asp, reply ) < 0 ) {
            LOG(log_error, logtype_afpd, "asp_cmdreply: %s", strerror(errno) );
            afp_asp_die(EXITERR_CLNT);
        }
        break;

    case ASPFUNC_WRITE :
        func = (u_char) asp->commands[0];
```

Passage par le branchement