

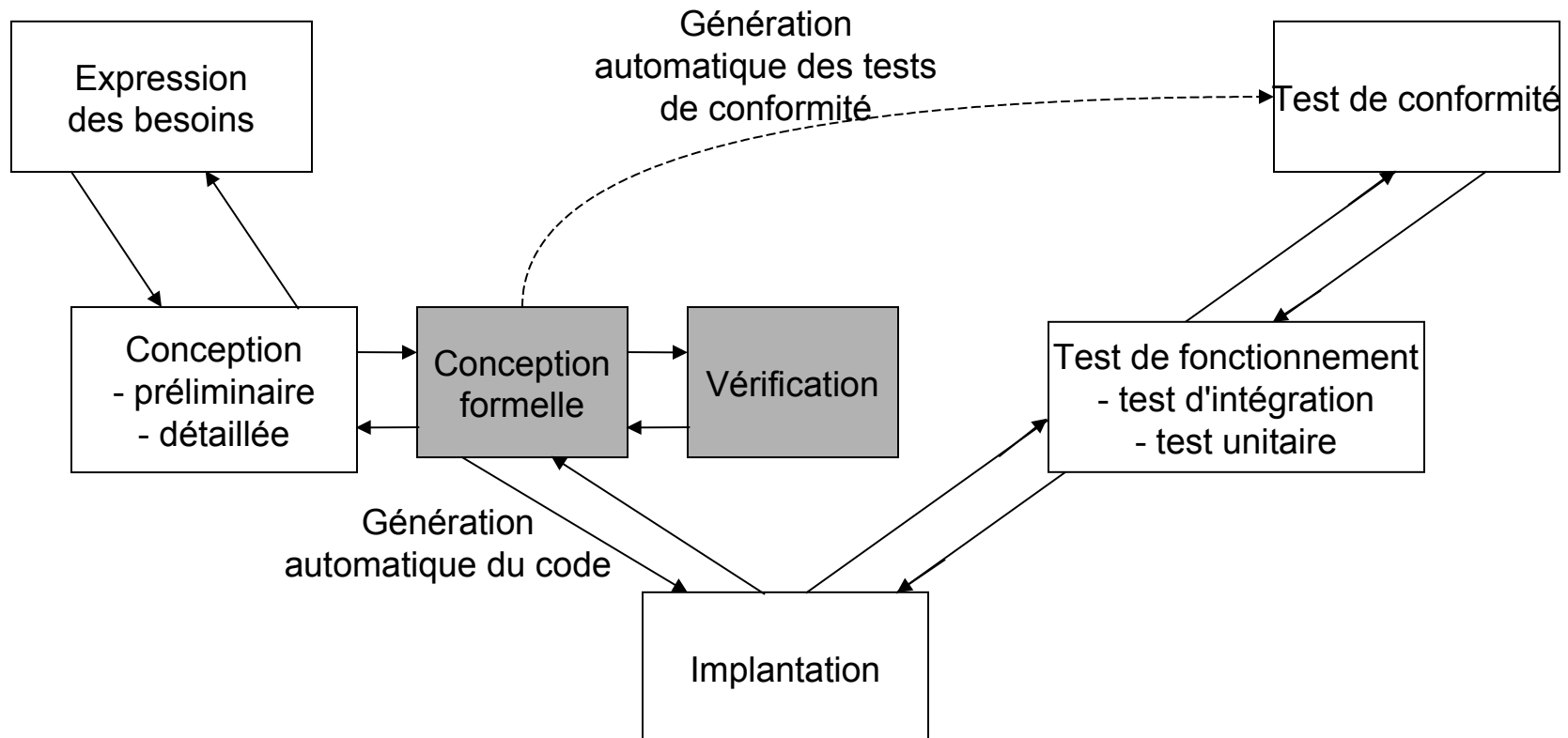
# Introduction au langage SDL

---

Fatiha Zaïdi  
zaidi@lri.fr

# Test de Conformité

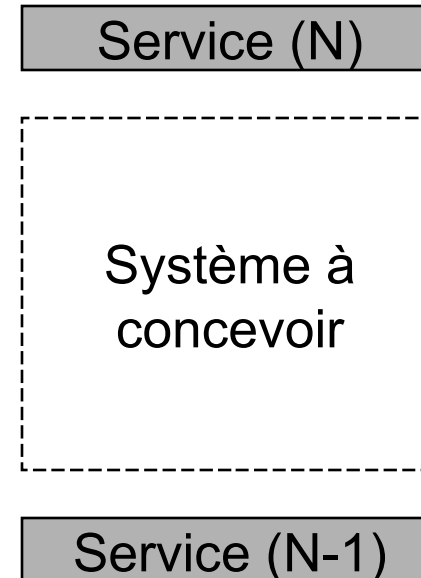
- Le modèle en V avec conception formelle



# Expression des besoins

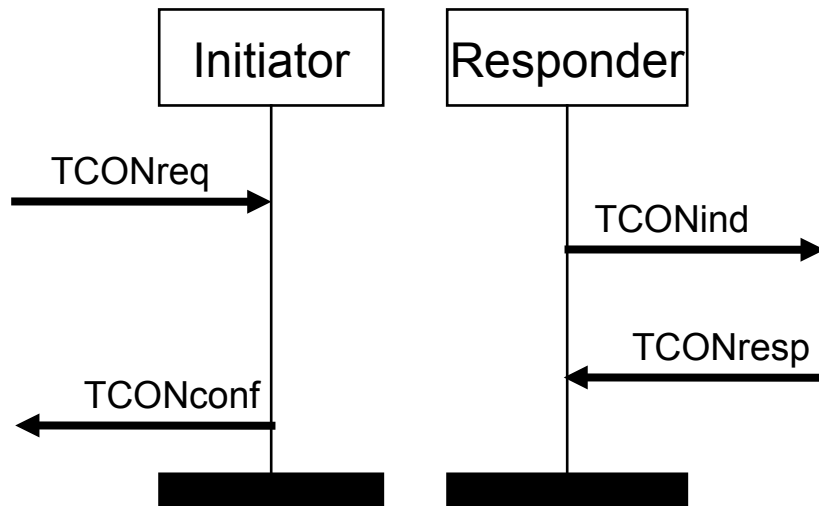
---

- Établissement d'un cahier des charges
  - Besoins de l'utilisateur en termes de services disponibles et de services désirés
  - Spécifier uniquement le comportement externe du logiciel
  - Spécifier les réponses
    - aux événements attendus
    - aux événements indésirables (robustesse)

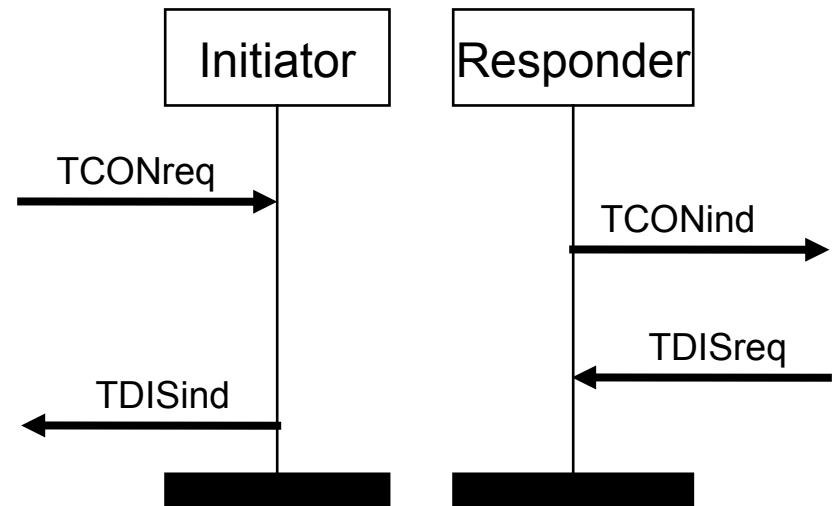


# Expression des besoins

- Description du service sous forme de MSC
  - MSC (Message Sequence Chart) : norme ITU-T Z.120



Connexion acceptée



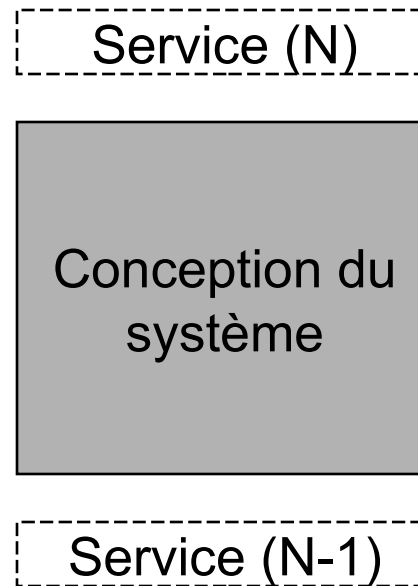
Connexion refusée

# Conception

---

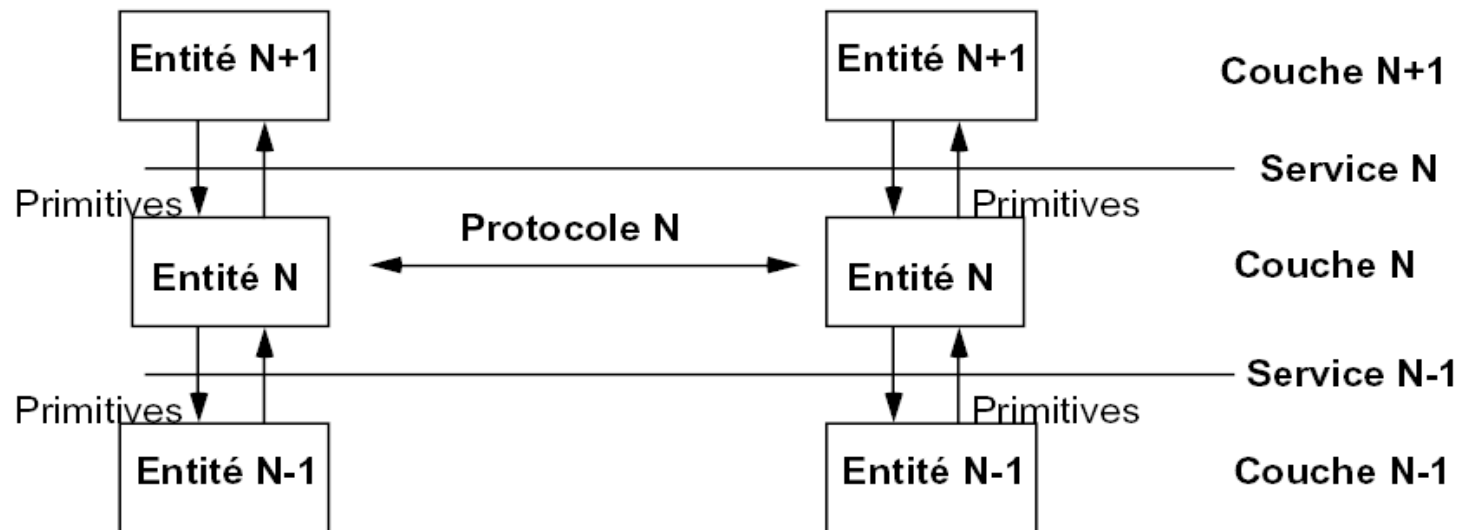
**Conception d'un protocole  
fournissant le service  
demandé en utilisant le service  
disponible**

- Réception et émission de primitives de service
- Réception et émission de PDU
- Gestion de temporisateurs
- Gestion de variables
- etc.



# Conception ...

## Notion de service : modèle en couches



**Le service** est une définition **fonctionnelle** de l'interface entre couches

Liste des primitives avec paramètres

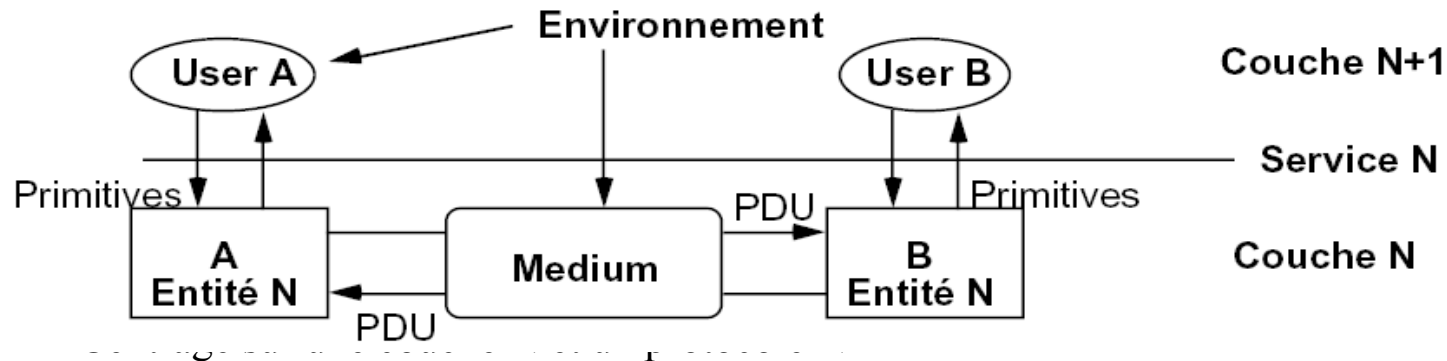
Ordonnancement permis des primitives

6

Le service N est une abstraction des couches de protocole inférieures.

# Conception ...

Notion d'environnement : abstraction des couches supérieures et inférieures



Les couches supérieures sont vues comme des utilisateurs abstraits

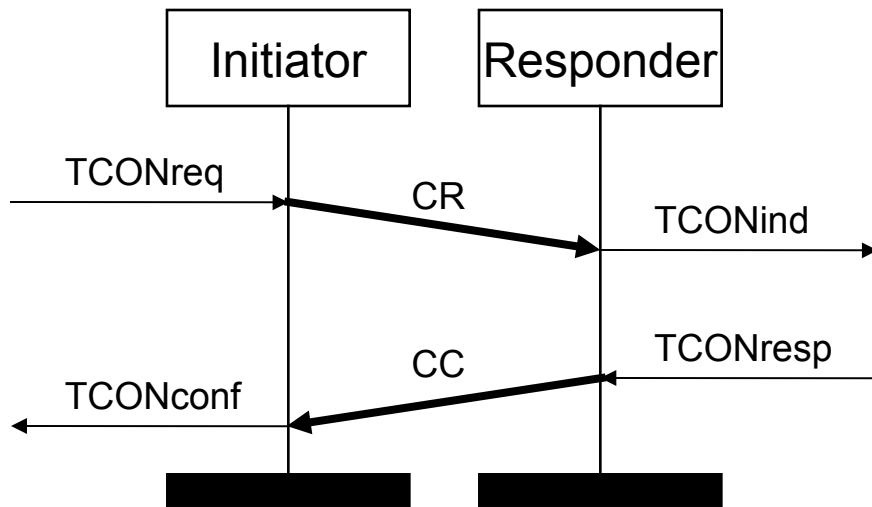
Les couches inférieures sont vues comme un milieu de transmission. Il décrit les hypothèses qui caractérisent ces couches (pertes, doublons possibles, ...)

Le protocole N décrit le fonctionnement des entités N (A et B) en réaction aux primitives de service N et aux PDU N venant de l'entité homologue via le milieu de transmission (médium).

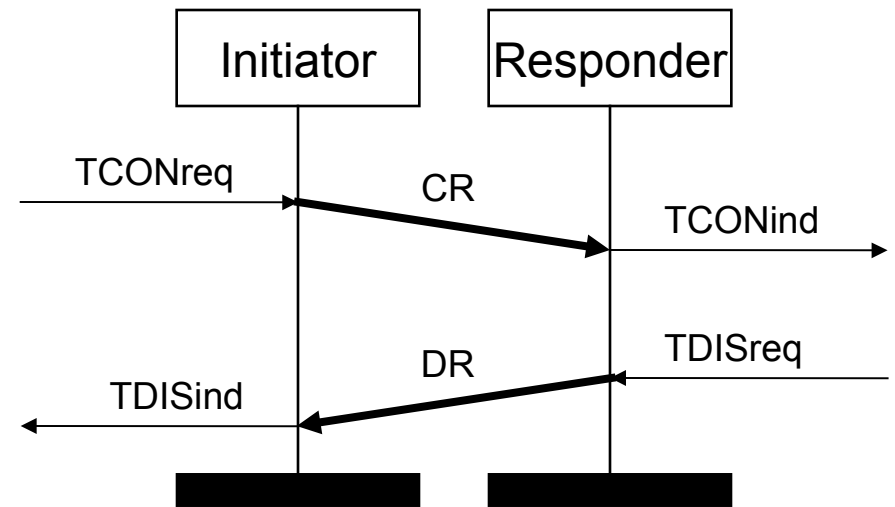
L'environnement d'un protocole est composé des utilisateurs et du milieu de transmission.

# Conception ...

- **Présentation de certains scénarios sous forme de MSC**



Connexion acceptée



Connexion refusée



# Conception formelle

---

- **Complexité des protocoles de communication**
  - **Systemes distribués**
  - **Tailles importantes**
  - **Lignes de transmission non fiables**
  - **Matériels souvent hétérogènes**



**Nécessité d'un formalisme  
de description formelle approprié**

# Test de conformité

- Objectif
  - Vérifier qu'une implantation de protocole est conforme à la spécification

?

Implantation = Spécification

- Méthodologie de test
  - Norme ISO 9646

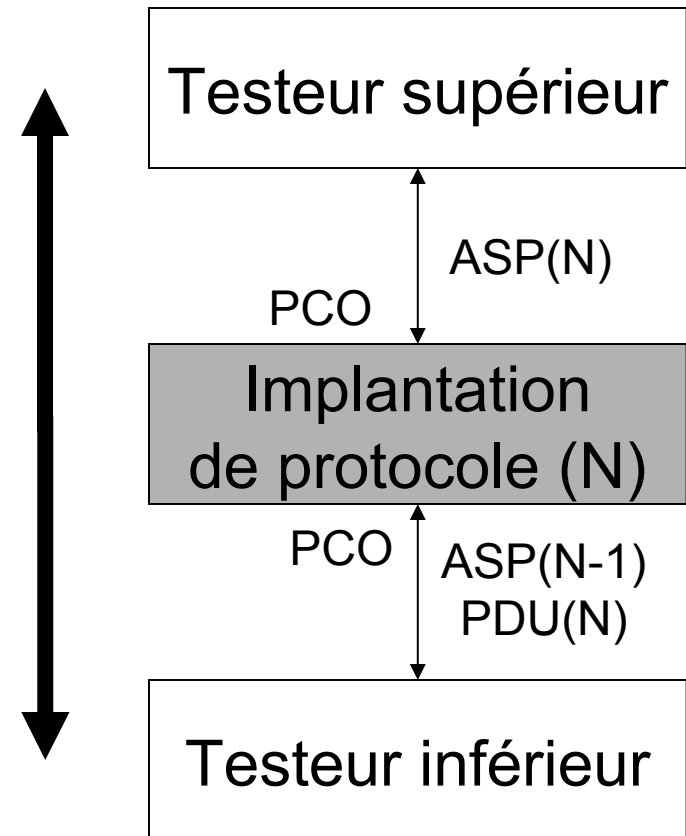
## Deux problèmes fondamentaux

- Architecture de test : environnement matériel et logiciel permettant la communication entre le système de test et l'implantation testée
- Construction des tests : élaboration des échanges de messages à exécuter dans l'architecture de test

# Test de conformité ...

- **Architecture de test**

- Étude de l'implantation sous test
- Interfaces accessibles de l'implantation sous test : les points de contrôle et d'observation (PCO)
- Utilisation d'un système de test, formé d'un testeur supérieur et d'un testeur inférieur communicants
- Le testeur inférieur est souvent distant



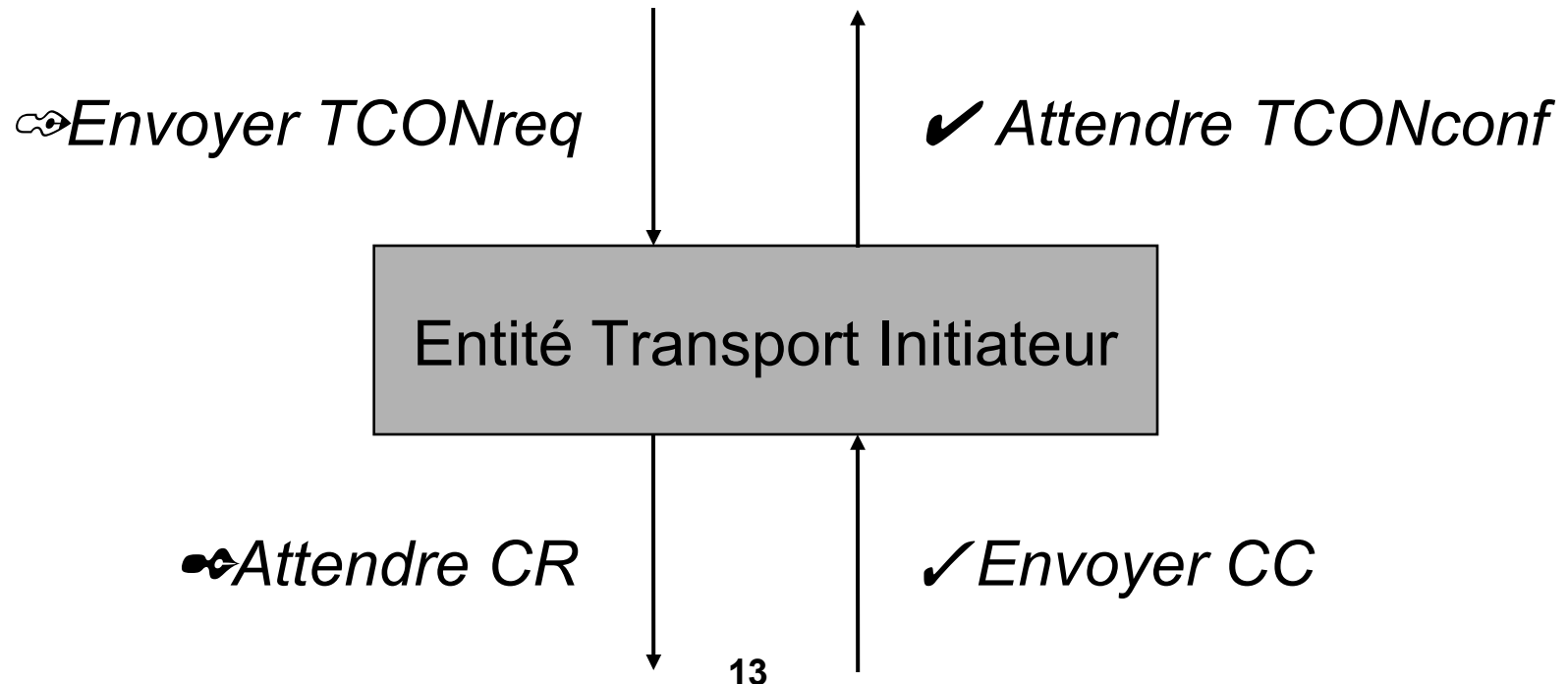
# Construction des tests

---

- **Structuration des tests**
  - Utilisation d'une suite de test, composée de nombreux jeux de test
- **Structure d'un jeu de test**
  - Préambule : placer l'implantation dans un état donné
  - Corps : tester la transition concernée, puis tester l'état d'arrivée après cette transition
  - Postambule : remettre l'implantation dans l'état initial
- **Types d'erreur détectées**
  - Erreurs d'opération
  - Erreurs de transfert

# Construction des tests ...

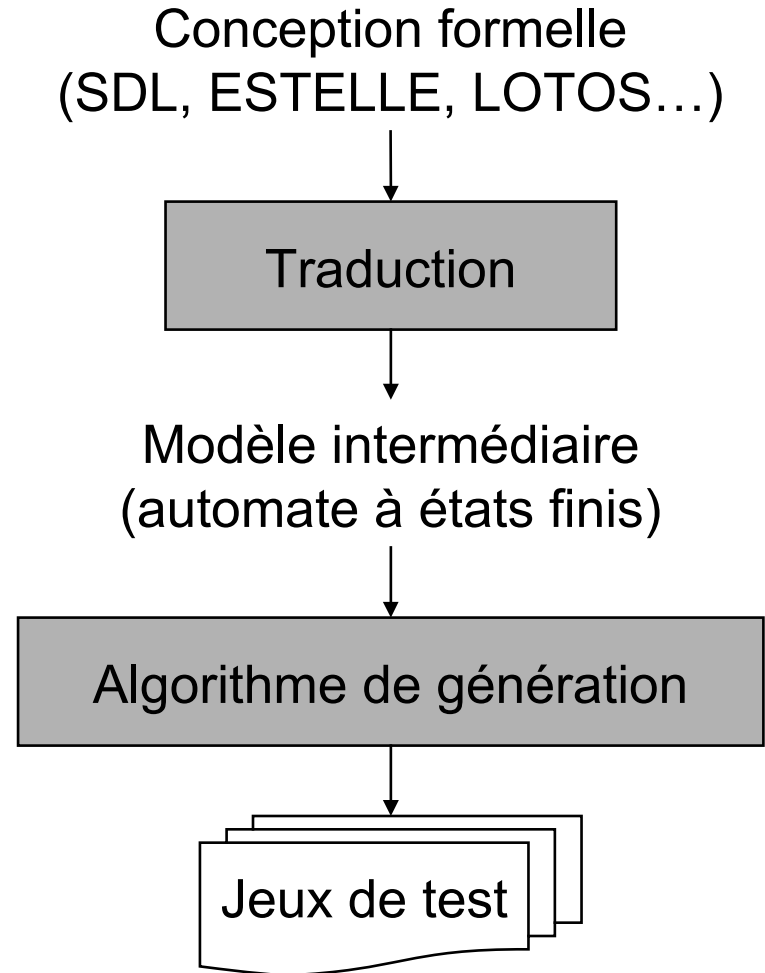
- **Exemple pour le transport OSI**
  - Test de l'entité Initiateur, cas d'une connection acceptée par l'entité Répondeur



# Construction des tests ...

- **Méthodes de construction des tests**

- Écriture manuelle : par un expert du domaine
- Génération automatique ou semi-automatique à partir de la conception formelle



# Spécification des tests

---

- **Verdict d'un test**
  - PASS
  - FAIL
  - INCONCLUSIVE
- **La langage utilisé pour les tests : TTCN**
  - Norme ISO 9646-3 : Tree and Tabular Combined Notation
  - Un très gros langage
  - TTCN contient ASN.1

# Spécification des tests ...

- Exemple pour le transport OSI

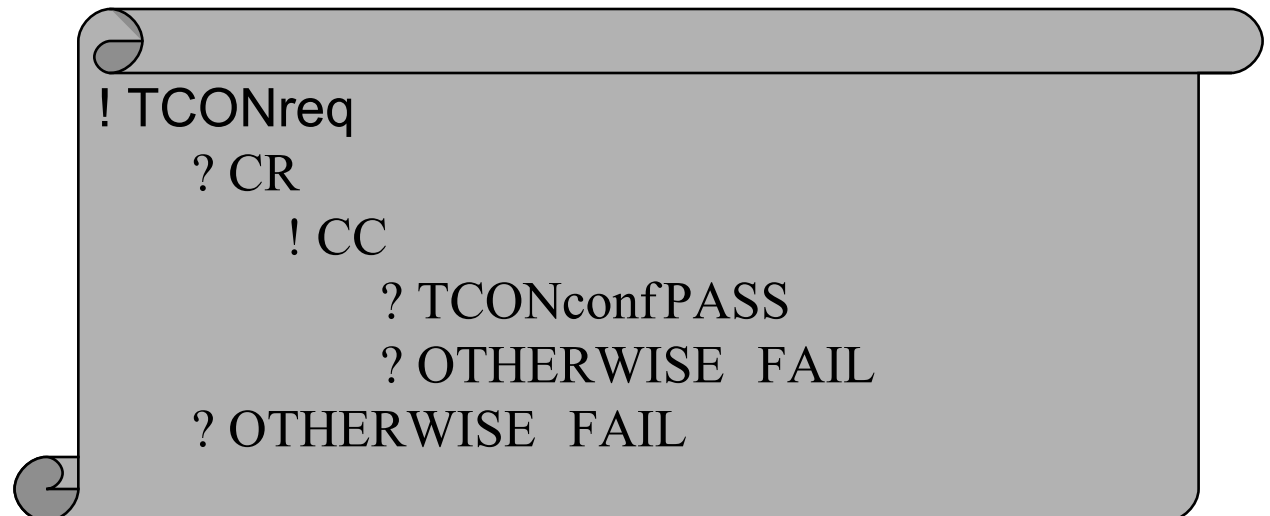
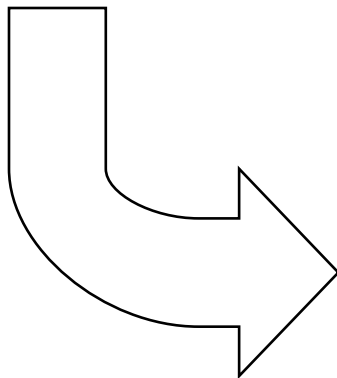
⇒ *Envoyer TCONreq*

• *Attendre CR*

✓ *Envoyer CC*

✓ *Attendre TCONconf*

Jeu de test TTCN





# Le langage SDL

---

# Une technique de description formelle

---

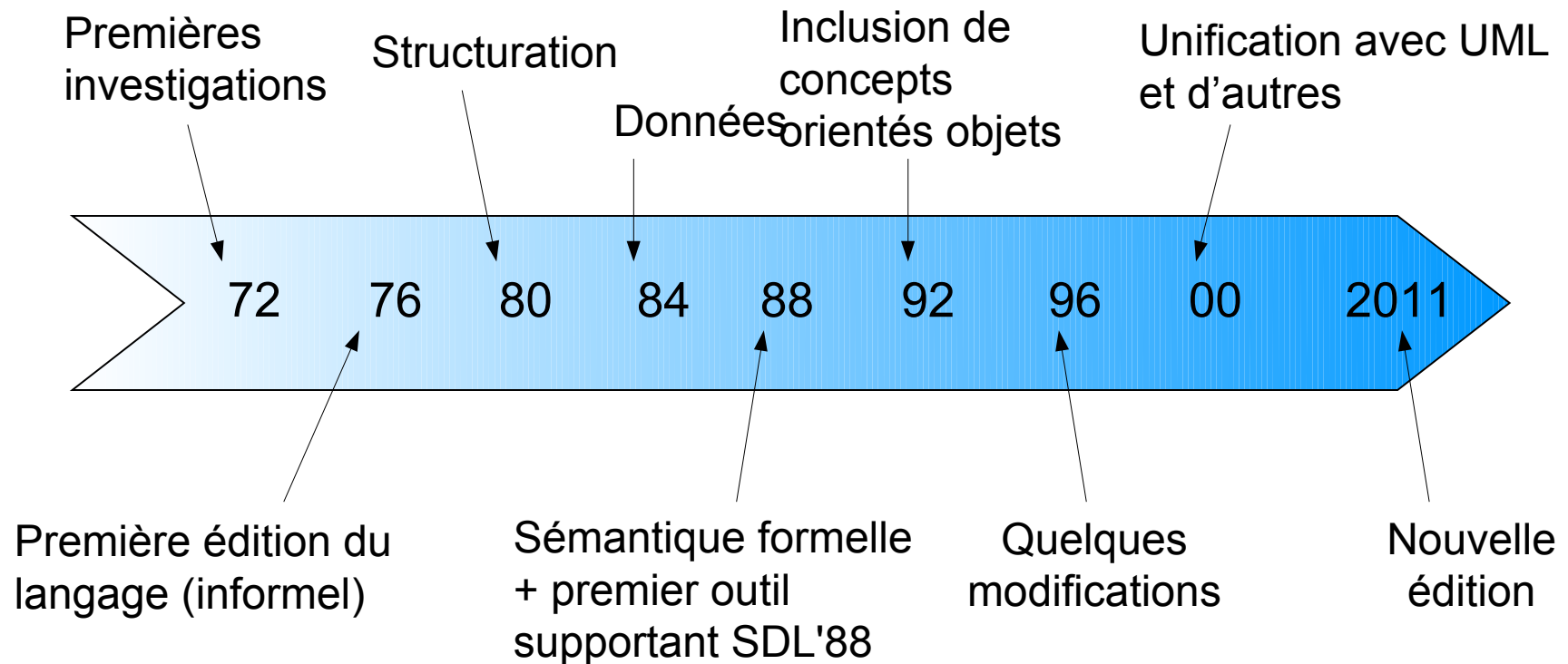
- SDL : Specification and Description Language □
  - spécification des **propriétés fonctionnelles** du système en fonction de son environnement
  - description des **systèmes distribués** composés de processus qui s'exécutent en parallèle et qui communiquent de façon **asynchrone** par messages
- D'autres langages et techniques
  - algèbres de processus
  - machines à états finis
  - réseaux de Petri
  - logiques temporelles...

# Objectifs du langage SDL

---

- Spécifier, décrire sans ambiguïté des systèmes de télécommunication
- Représenter les propriétés fonctionnelles d'un système
  - **Propriétés structurelles** : architecture du système, décomposition sous forme de blocs fonctionnels interconnectés
  - **Propriétés comportementales** : réactions du système aux stimuli provenant de son environnement
- L'architecture et le comportement sont deux caractéristiques indépendantes

# Un peu d'histoire



**Recommandation de l'ITU Z.100**

# Intérêts d'un langage formel

---

- Spécifications claires, précises et concises
- Possibilité d'analyser la **correction** et la **complétude** des spécifications
- Possibilité de tester la **conformité** d'une implantation par rapport à sa spécification
- Possibilité d'utiliser des **outils** pour développer, maintenir, vérifier, valider, analyser et simuler des spécifications

# Intérêts de spécifier

---

- Spécification à différents niveaux d'abstraction
  - donner une vue d'ensemble d'un système complexe
  - reporter les choix d'implantations
  - ne pas contraindre l'implantation par des détails inutiles au niveau des spécifications
- Raffinement des spécifications jusqu'à l'implantation
  - spécifications de plus en plus précises jusqu'à pouvoir être automatiquement traduites dans un langage de programmation

# Message Sequence Chart

---

- Recommandation Z.120 de l'ITU
  - « une msc doit fournir un langage de trace pour la spécification et la description du comportement de la communication entre les composants du système et leur environnement par l'échange de messages »

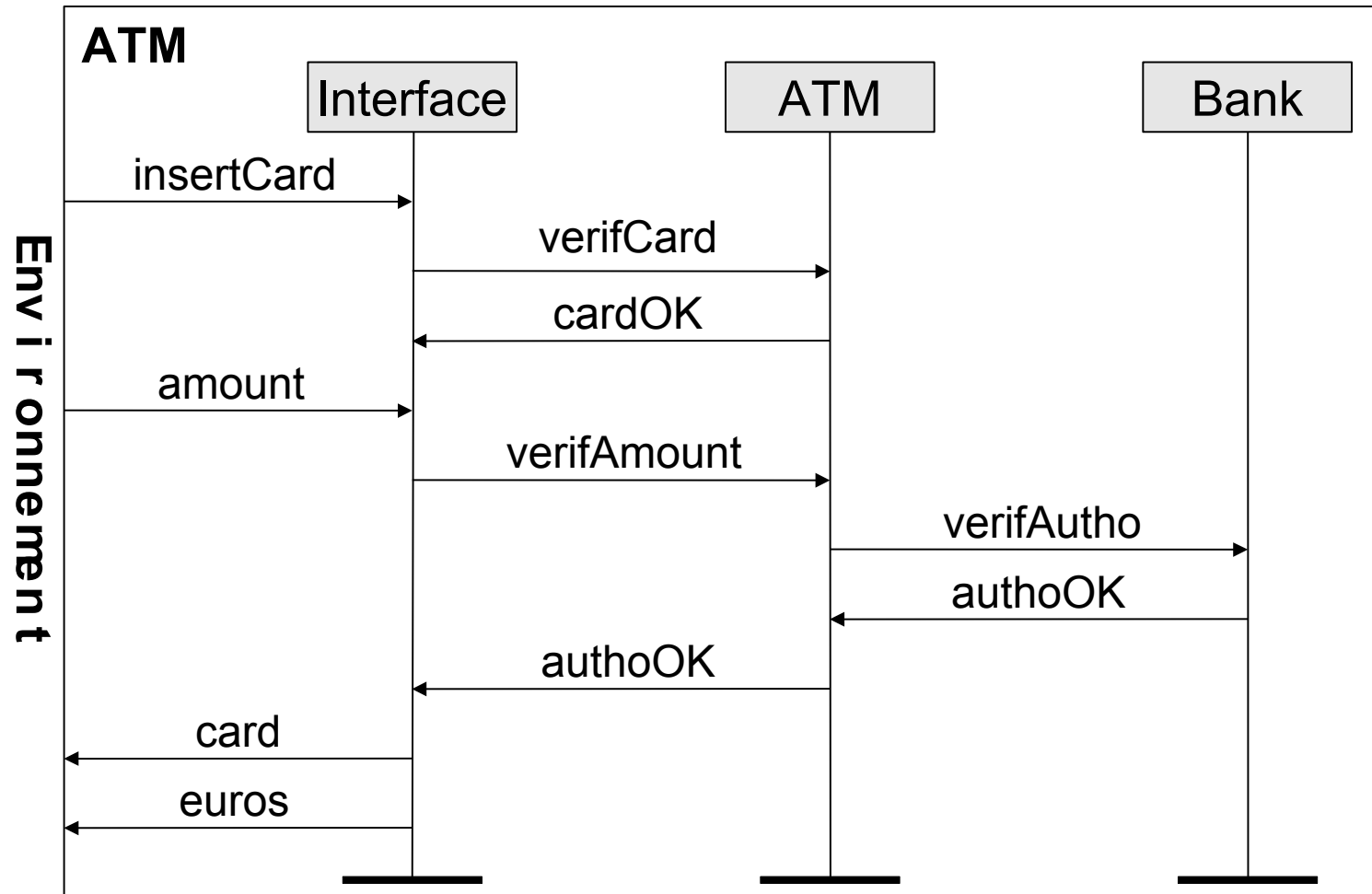
# Message Sequence Charts

---

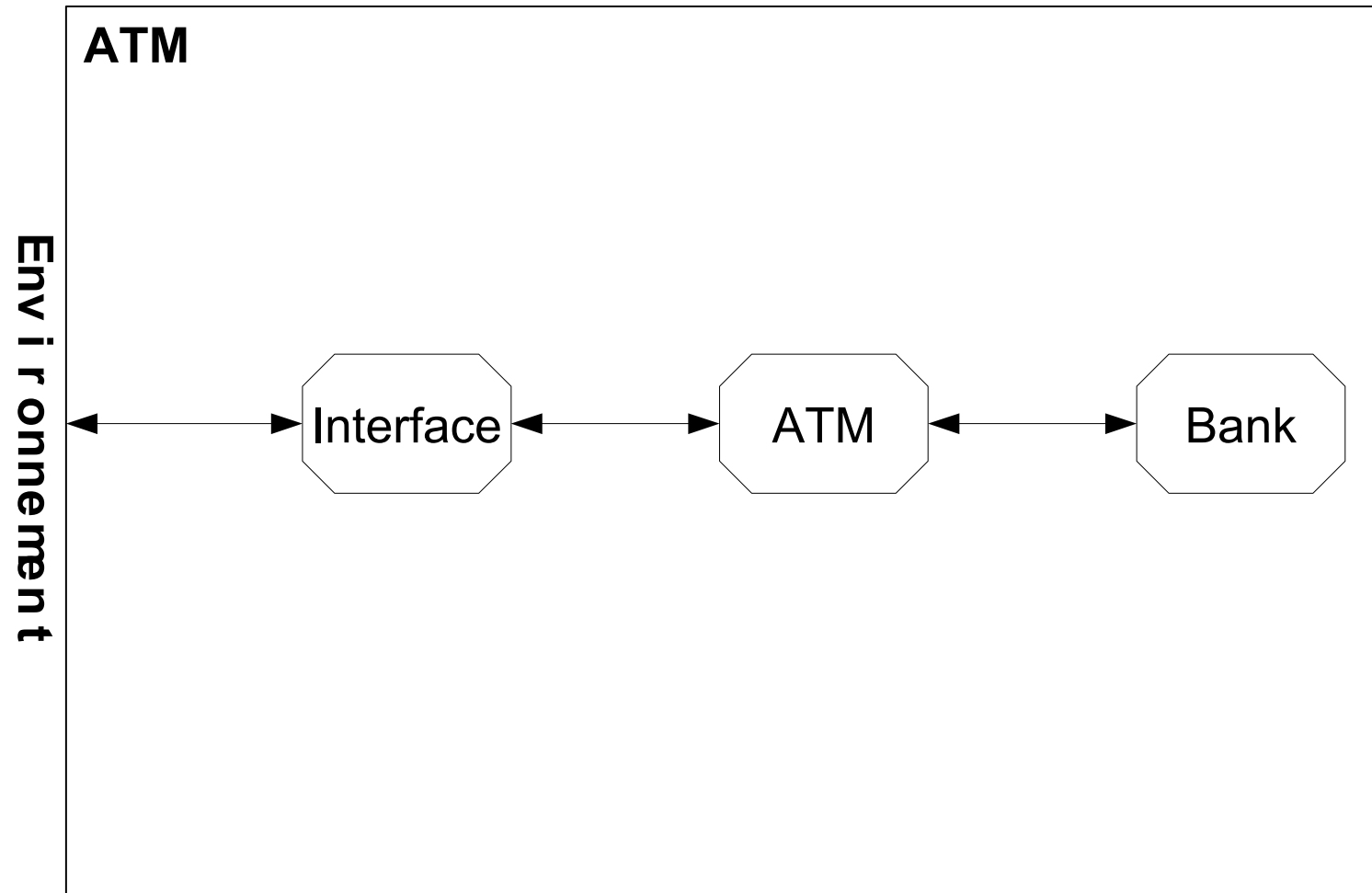
- Description des échanges de messages entre les processus du système et entre le système et son environnement
- Description haut niveau de scénarios d'utilisation, spécifie seulement un ordre entre les messages, pas d'affectation de valeurs et d'opérations. On utilise SDL et on peut contrôler avec les MSCs
- Abstraction du comportement interne de chacun des processus



# Exemple de MSC

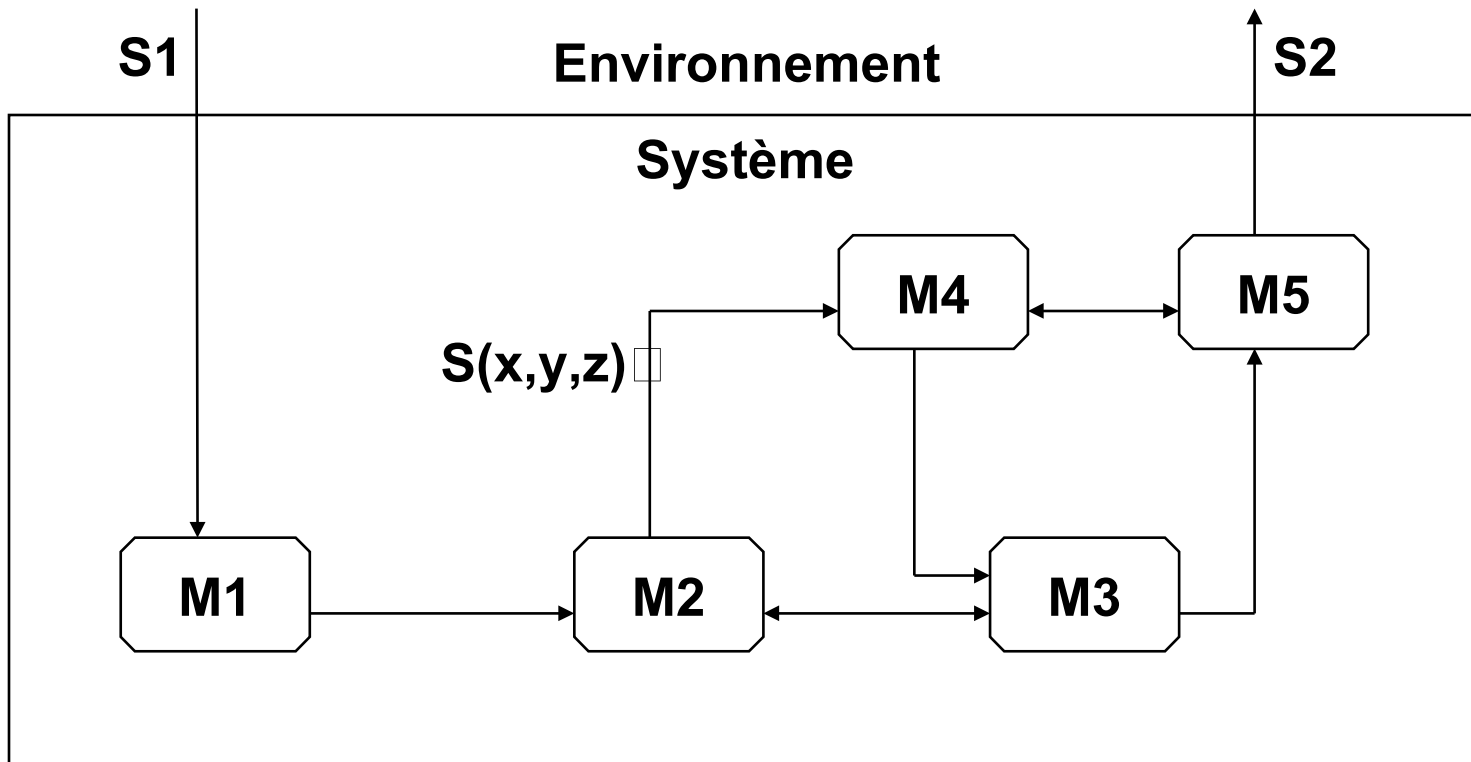


# Systeme SDL correspondant

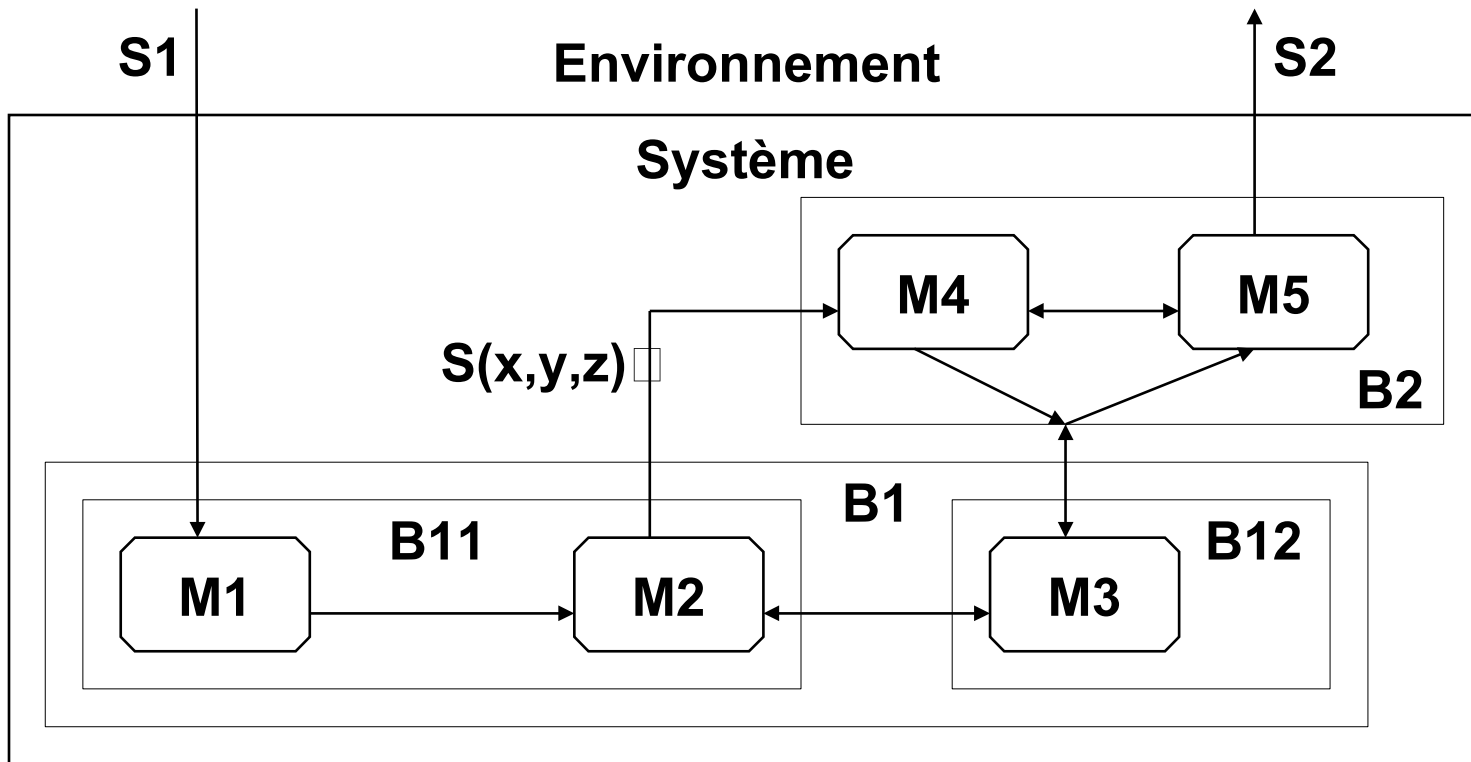


# Comportement d'un système SDL

---



# Architecture d'un système SDL



# Spécification d'un système avec SDL

---

- Trois aspects
  - **Structure** : décomposition hiérarchique en blocs et en processus reliés par des canaux de communication
  - **Comportement** : machines à états finis et échange de messages
  - **Données** : opérations sur les structures de données associées aux interactions

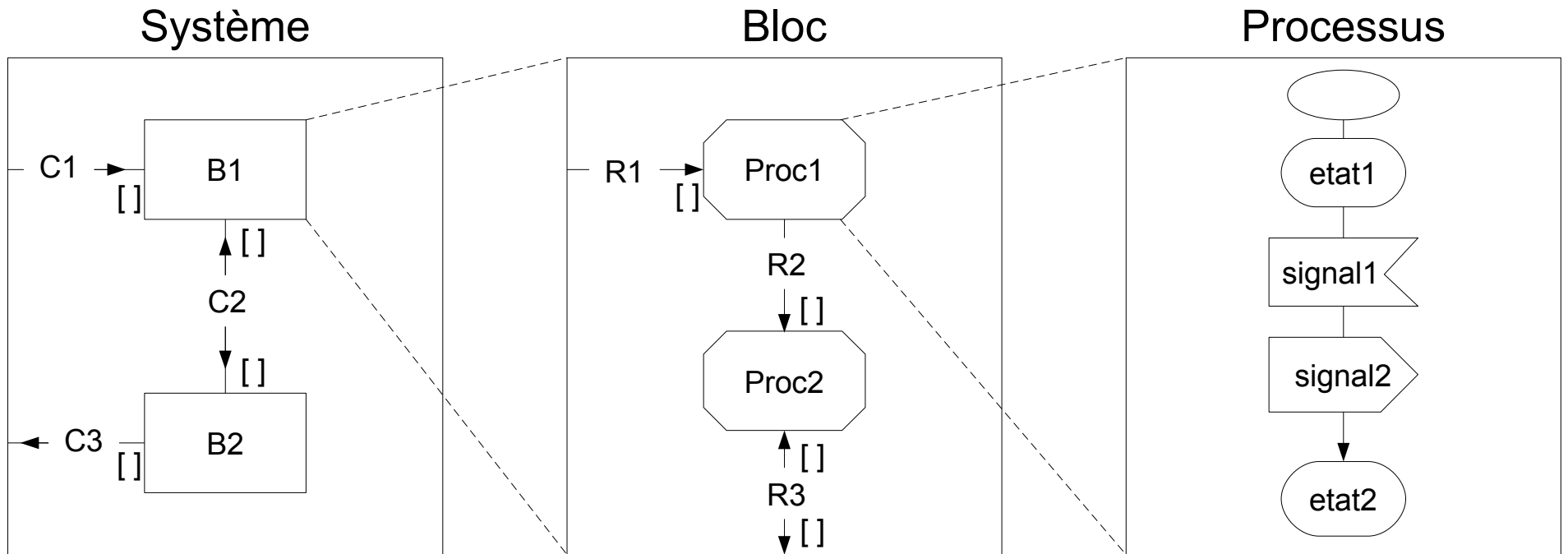
→ trois modèles sémantiques

# Les modèles sémantiques de SDL

---

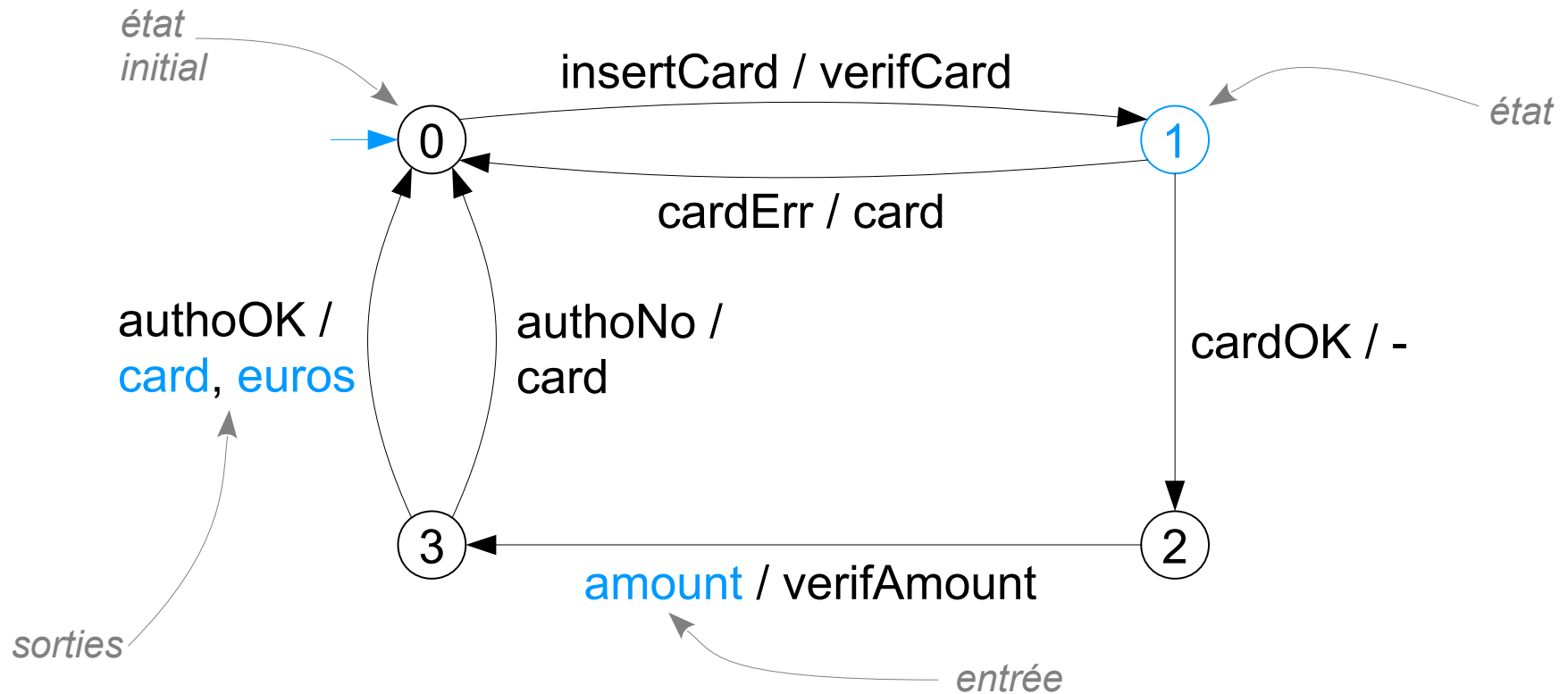
- Architecture
  - Décomposition en entités structurelles interconnectées : système, bloc, sous-bloc, processus, reliés par des canaux de communication
  - **Organisation arborescente** suivant l'approche de conception descendante
- Comportement
  - Ensemble de processus communiquant de manière asynchrone par échange de messages et par variables partagées
  - **Machines à états finis étendues**
- Structures de données
  - **Types abstraits algébriques**

# Organisation Hiérarchique



# Machine à états finis

- Interface de l'ATM





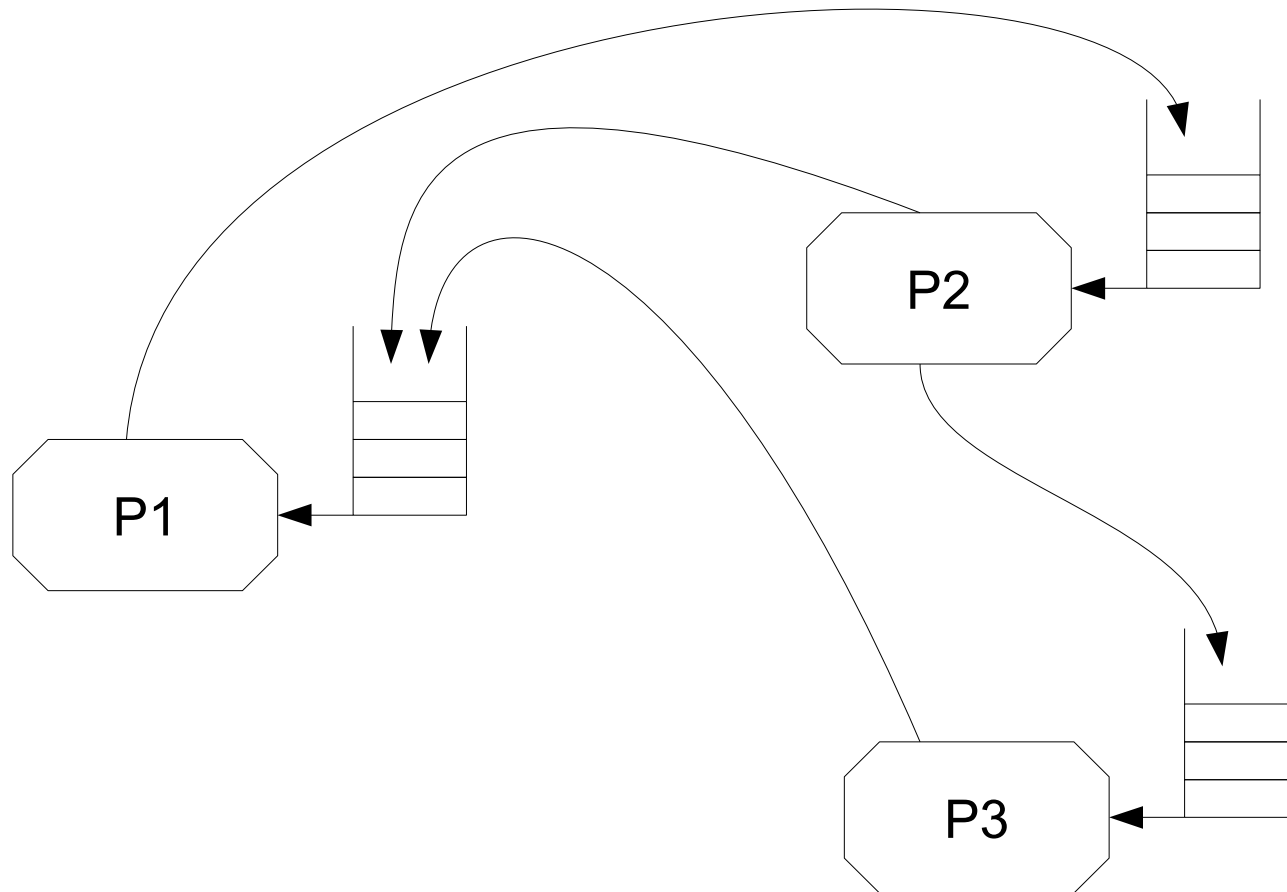
# Machine à états finis

---

- Ensemble d'états reliés par des transitions
- À la réception d'un signal, une transition est franchie, un ou plusieurs signaux sont émis, et la machine passe dans un nouvel état
- Communications **asynchrones** : une file d'attente **FIFO** est associée à chaque machine (à chaque processus)

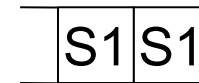
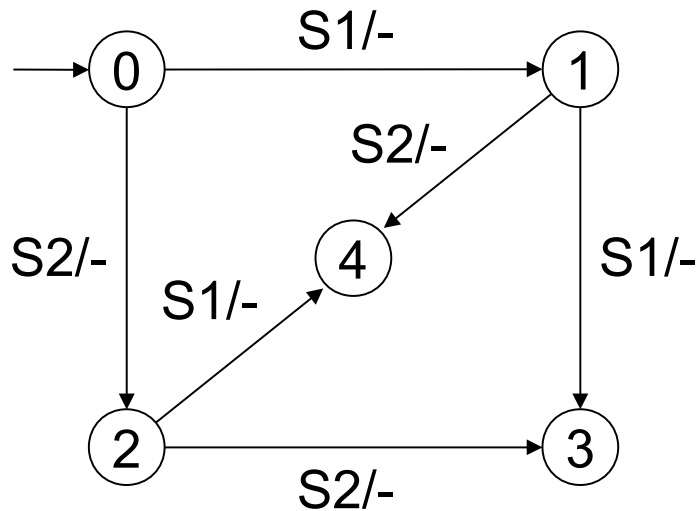
# Communication asynchrone FIFO

---

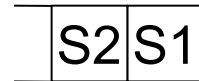


# Communication asynchrone FIFO

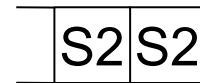
- Changement d'état en fonction du signal reçu



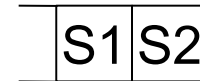
$0 \rightarrow 1 \rightarrow 3$



$0 \rightarrow 1 \rightarrow 4$



$0 \rightarrow 2 \rightarrow 3$



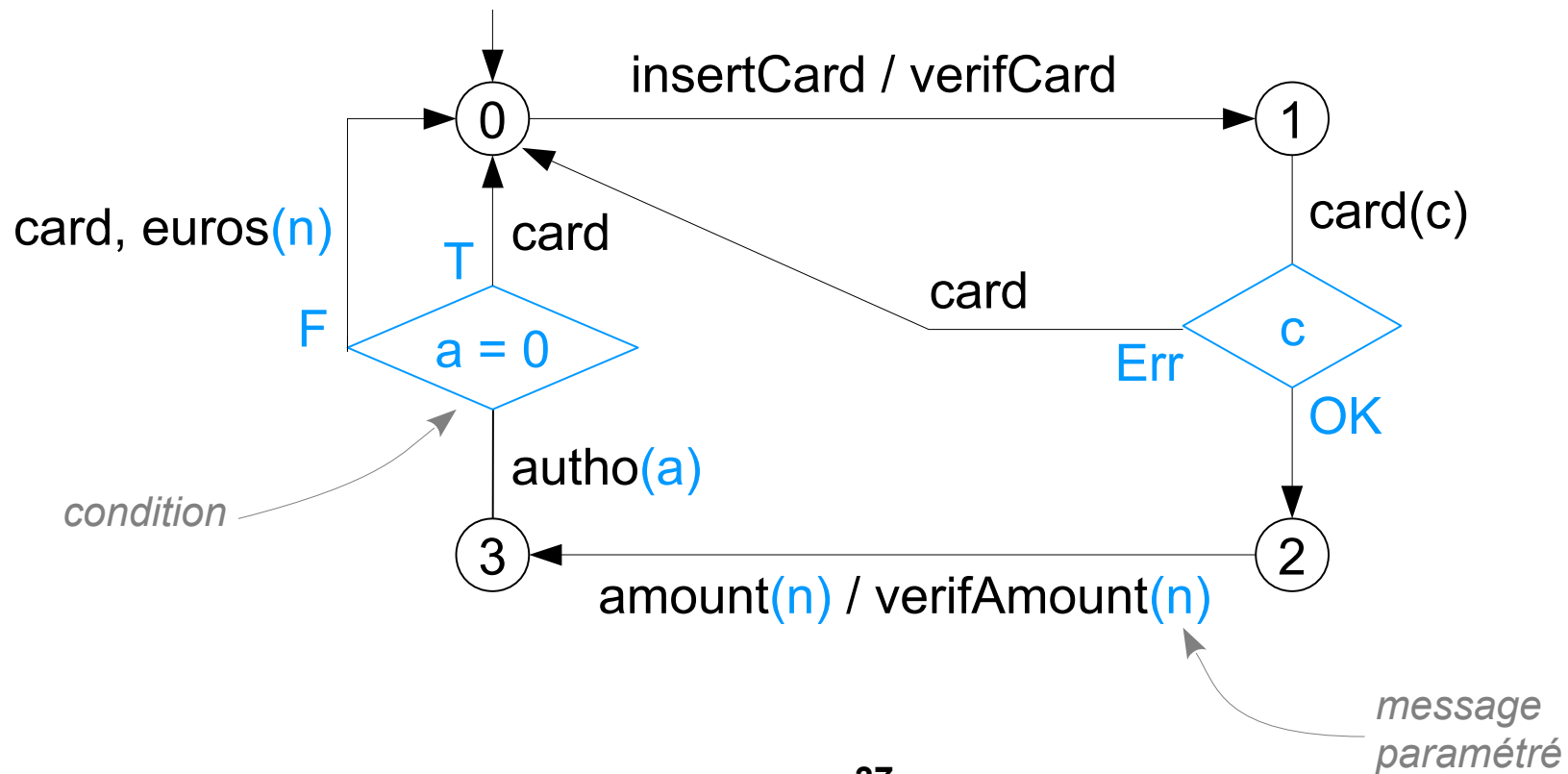
$0 \rightarrow 2 \rightarrow 4$

# Machine à états finis étendue

---

- Extension aux données
  - Ensemble de **variables locales** à chaque processus (mémoire locale) permettant de stocker des valeurs, de conditionner le passage d'une transition...
  - **Messages paramétrés** par des variables : envoi et réception de valeurs
- Réduction considérable du nombre d'états et de transitions (syntaxiquement)
- Espace d'états réels potentiellement infini

# Machine à états finis étendue



# Processus SDL

---

- Extension du modèle de machine à états finis
  - **File d'attente** : pas nécessairement FIFO. Quatre types de stimuli possibles, et une transition peut être déclenchée sur file vide.
  - **Actions** dans une transition : envoi de signaux et lecture de variables locales, mais aussi
    - exécution de tâches (affectation ou texte informel) □
    - appel de procédures
    - création dynamique de processus
    - armement et désarmement de temporisateurs
  - **Durée d'une transition** : pas nécessairement nulle

# Processus SDL

---

- Extension du modèle de machine à états finis
  - Transition d'**initialisation** vers l'état initial
  - **Timeout** d'un temporisateur modélisé par un signal spécial, non prioritaire
  - Plusieurs niveaux de visibilité des variables
    - **Variables locales** : visibles uniquement à l'intérieur d'un processus
    - **Variables révélées** : visibles par tous les processus du même bloc
    - **Variables exportées** : visibles par les processus de tous les blocs

# Syntaxes concrètes de SDL

---

- Deux syntaxes
  - **SDL/GR** : représentation **graphique** à l'aide de symboles normalisés
  - **SDL/PR** : représentation **textuelle** analogue à un langage de programmation
- Avantages
  - SDL/GR montre très clairement la structure d'un système et son flot de contrôle
  - SDL/PR est un moyen d'échange entre différents outils SDL



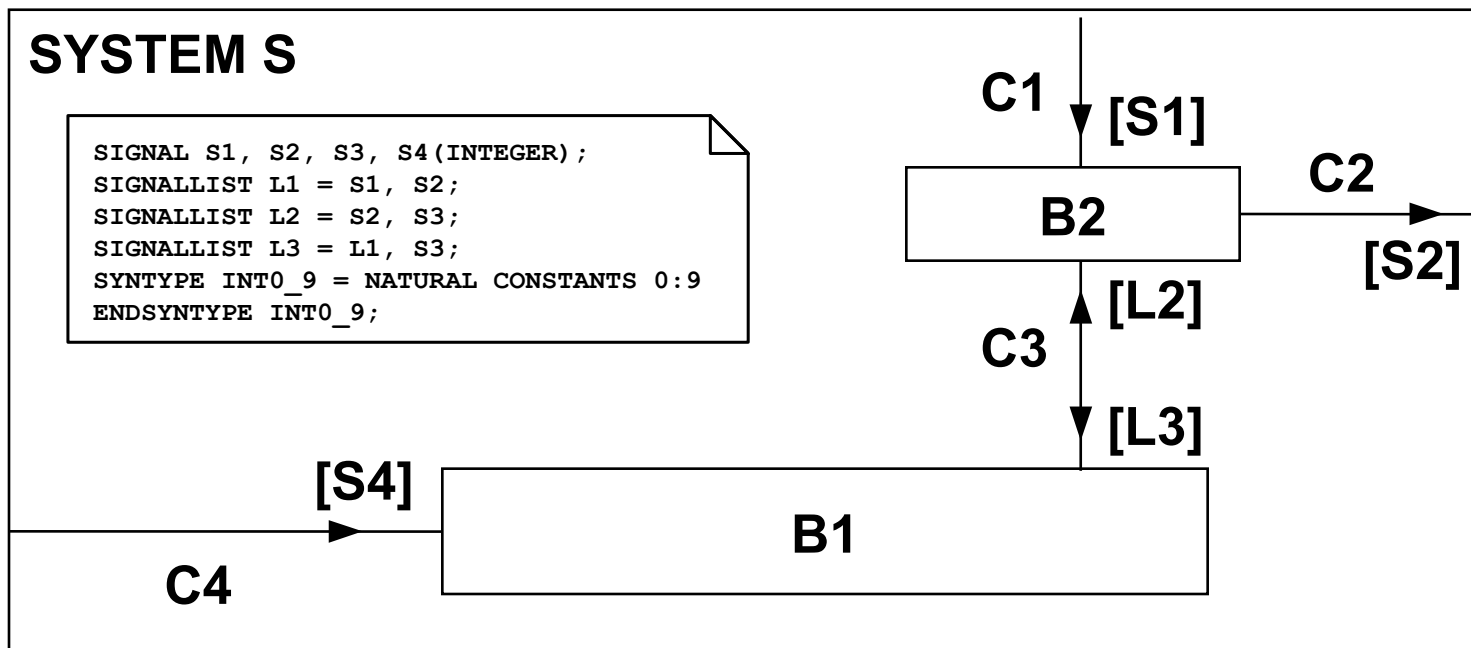
# Système SDL

---

- Spécification d'un système (premier niveau)
  - Nom du système
  - Définition des signaux, précision du type des paramètres □
  - Définition de listes de signaux pour faciliter l'écriture de la spécification
  - Définition des types de données visibles dans toute la spécification
  - Définition des blocs composant le système
  - Définition des canaux connectant les blocs du système entre eux et à l'environnement, avec la spécification de l'ensemble des signaux transportés

# Systeme SDL : exemple

- Représentation graphique



Les blocs B1 et B2 sont référencés

# Systeme SDL : exemple

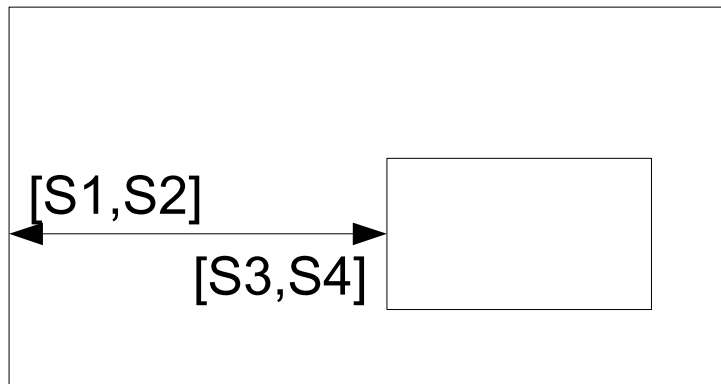
- Représentation textuelle

```
SYSTEM S;  
  
  SIGNAL S1, S2, S3, S4(INTEGER);  
  SIGNALLIST L1 = S1, S2;  
  SIGNALLIST L2 = S2, S3;  
  SIGNALLIST L3 = L1, S3;  
  SYNTYPE INTO_9 = NATURAL  
    CONSTANTS 0:9  
  ENDSYNTYPE INTO_9;  
  
  BLOCK B1 REFERENCED;  
  BLOCK B2 REFERENCED;
```

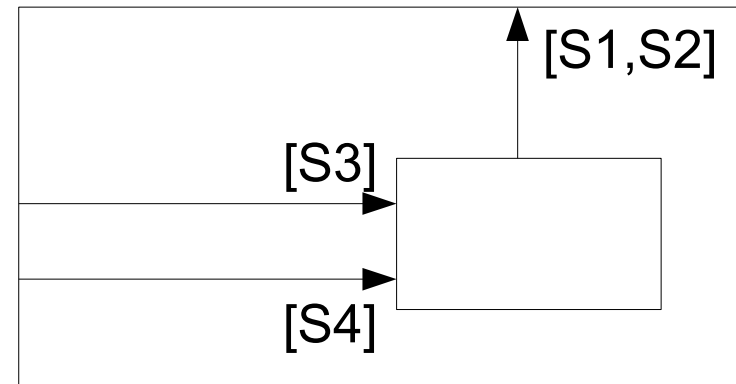
```
  CHANNEL C1  
    FROM ENV TO B2 WITH S1;  
  CHANNEL C2  
    FROM B2 TO ENV WITH S2;  
  CHANNEL C3  
    FROM B1 TO B2 WITH L2;  
    FROM B2 TO B1 WITH L3;  
  CHANNEL C4  
    FROM ENV TO B1 WITH S4;  
  
ENDSYSTEM S;
```

# Systeme SDL

- Interface(s) avec l'environnement
  - un processus par extrémité de canal
  - environnement = plusieurs processus concurrents



S3 et S4 séquentiels



S3 et S4 concurrents

# Signaux et listes de signaux

---

- Définition d'un signal
  - Nom du signal
  - Liste de sortes : liste des types des paramètres
- SDL/PR

```
SIGNAL S1, S2, S3;  
SIGNAL S4 (INTEGER);  
SIGNAL S5 (sort1, sort2);
```

- SDL/GR

```
SIGNAL S1, S2, S3;  
SIGNAL S4 (INTEGER);  
SIGNAL S5 (sort1, sort2);
```

# Signaux et listes de signaux

---

- Définition d'une liste de signaux
  - Nom de la liste
  - Liste des signaux et/ou listes de signaux

- SDL/PR

```
SIGNALLIST L1 = S1, S4;  
SIGNALLIST L2 = L1, S2;
```

- SDL/GR

```
SIGNALLIST L1 = S1, S4;  
SIGNALLIST L2 = L1, S2;
```

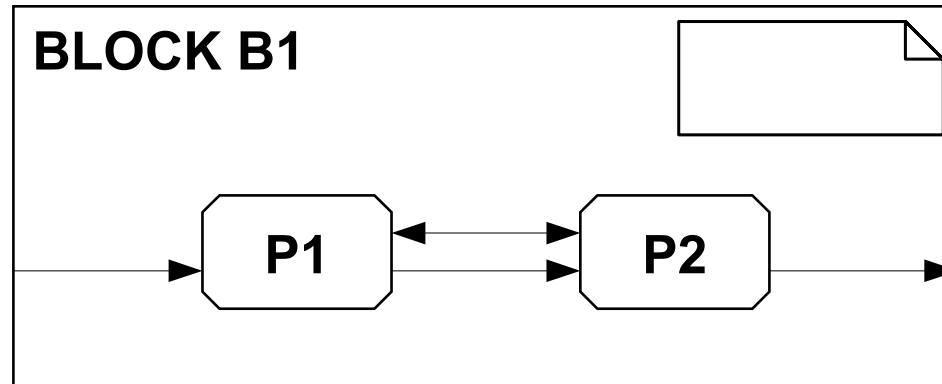
# Les types de blocs

- Bloc terminal : bloc composé exclusivement de processus

- SDL/PR

```
BLOCK B1;  
    < définitions de processus >  
ENDBLOCK B1;
```

- SDL/GR



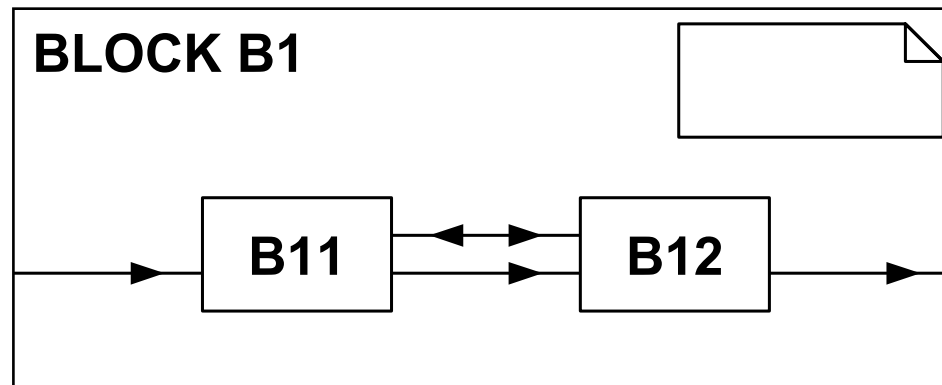
# Les types de blocs

- Bloc non terminal ou sous-structure de bloc : bloc composé exclusivement de blocs

- SDL/PR

```
BLOCK B1;  
  SUBSTRUCTURE B1;  
    < définitions de blocs >  
  ENDSUBSTRUCTURE B1;  
ENDBLOCK B1;
```

- SDL/GR





# Canaux

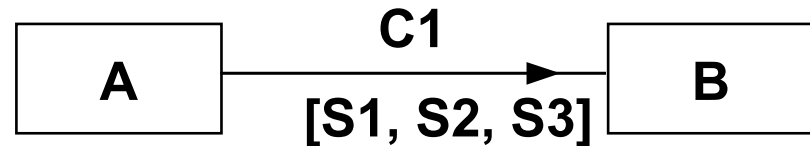
---

- Spécification d'un canal (*channel*)
  - Nom du canal
  - Directions possibles des signaux dans le canal
    - canal unidirectionnel : signaux transmis dans un seul sens
    - canal bidirectionnel : signaux transmis dans les deux sens
  - Pour chaque sens, spécification des blocs extrémités ou de l'environnement (ENV), ainsi que de la liste des signaux transportés

# Canaux

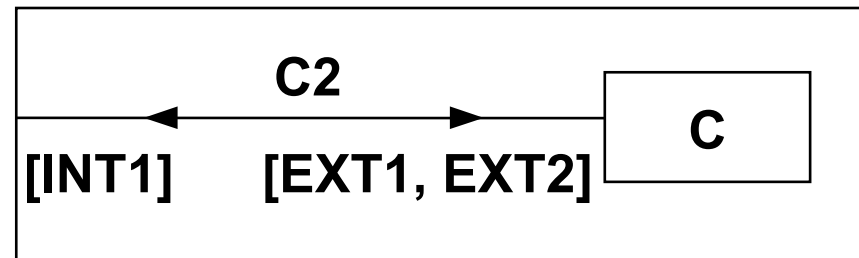
- Canal unidirectionnel entre deux blocs

```
CHANNEL C1
  FROM A TO B WITH S1, S2, S3;
ENDCHANNEL C1;
```



- Canal bidirectionnel entre un bloc et l'environnement

```
CHANNEL C2
  FROM ENV TO C WITH EXT1, EXT2;
  FROM C TO ENV WITH INT1;
ENDCHANNEL C2;
```

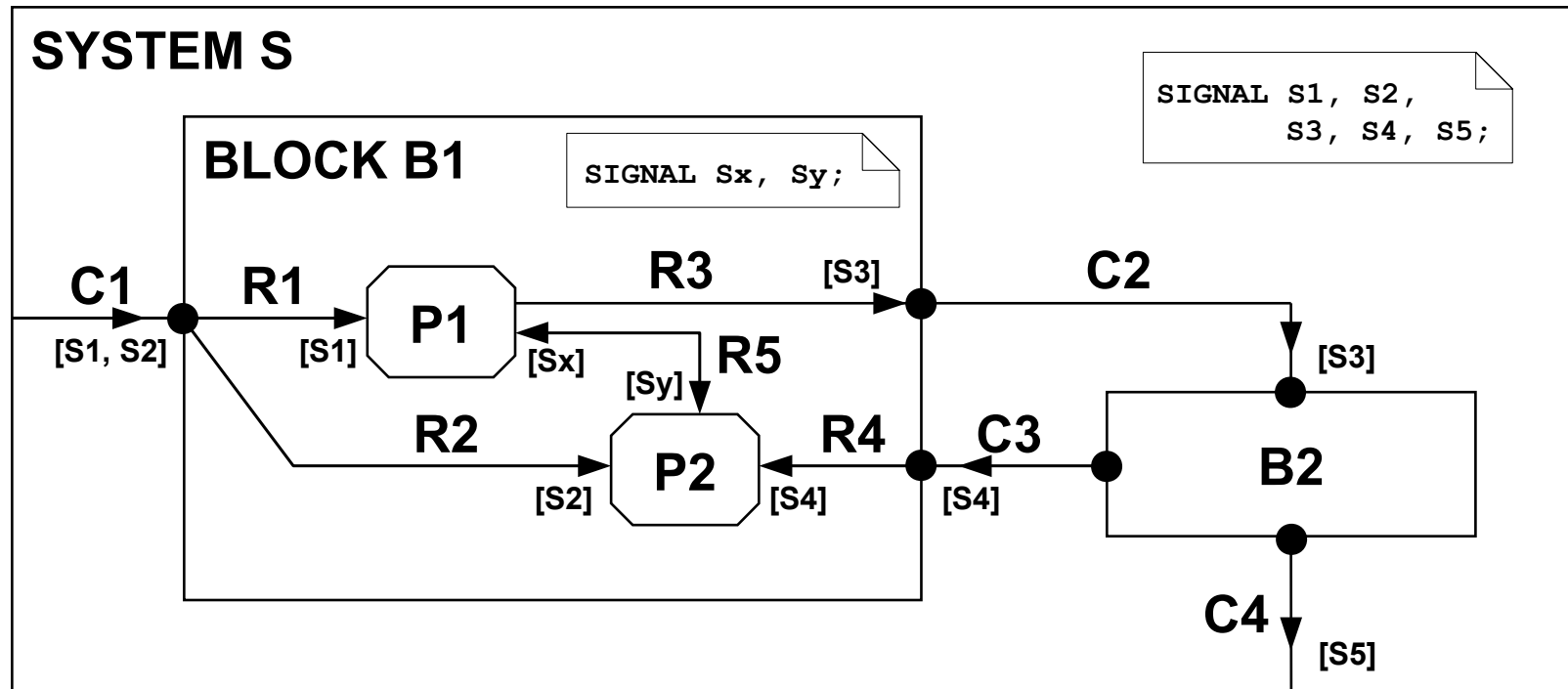


# Bloc terminal

---

- Spécification d'un bloc terminal
  - Nom du bloc
  - Définition des signaux, précision du type des paramètres
  - Définition de listes de signaux
  - Définition des types de données utilisables par tous les processus à l'intérieur du bloc
  - Définition des processus qui décrivent le comportement du bloc
  - Définition de l'acheminement des signaux connectant les processus du bloc entre eux et avec l'environnement, avec la spécification de l'ensemble des signaux transportés
  - Définition des connexions entre les canaux externes au bloc et les acheminements de signaux internes au bloc

# Bloc terminal



# Acheminement de signal (*signal route*)

- Acheminement de signal bidirectionnel entre deux processus

CHANNEL R5

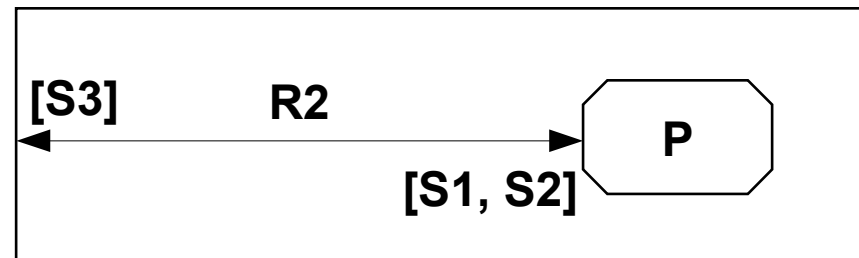
FROM P1 TO P2 WITH Sy;  
FROM P2 TO P1 WITH Sx;



- Acheminement de signal bidirectionnel entre un processus et l'environnement

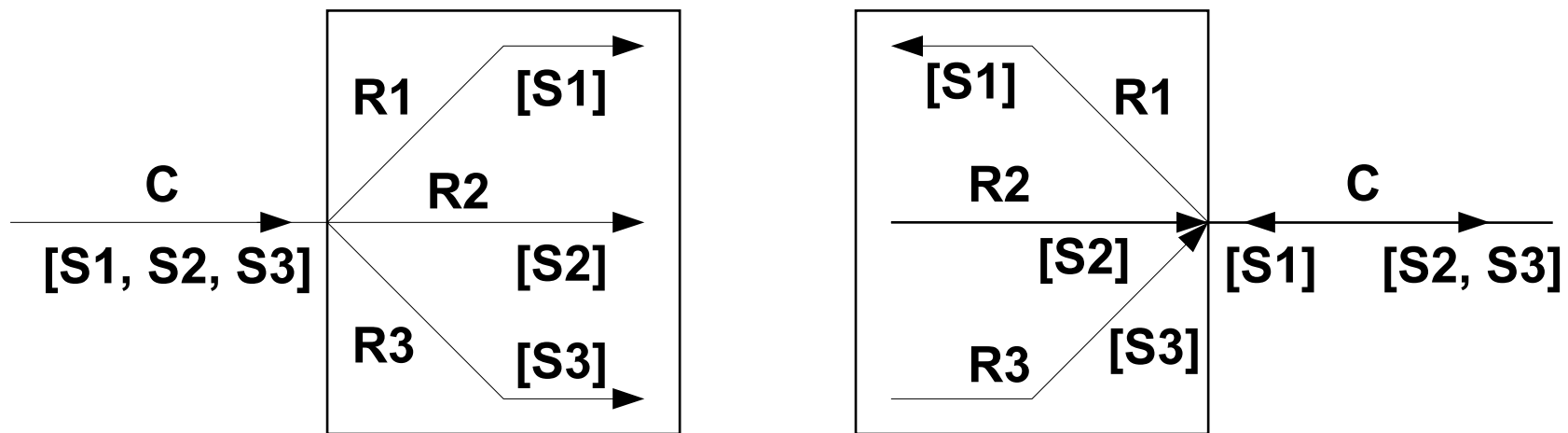
CHANNEL R2

FROM ENV TO P WITH S1, S2;  
FROM P TO ENV WITH S3;



# Connexions entre canaux et acheminements de signaux

CONNECT C AND R1, R2, R3;

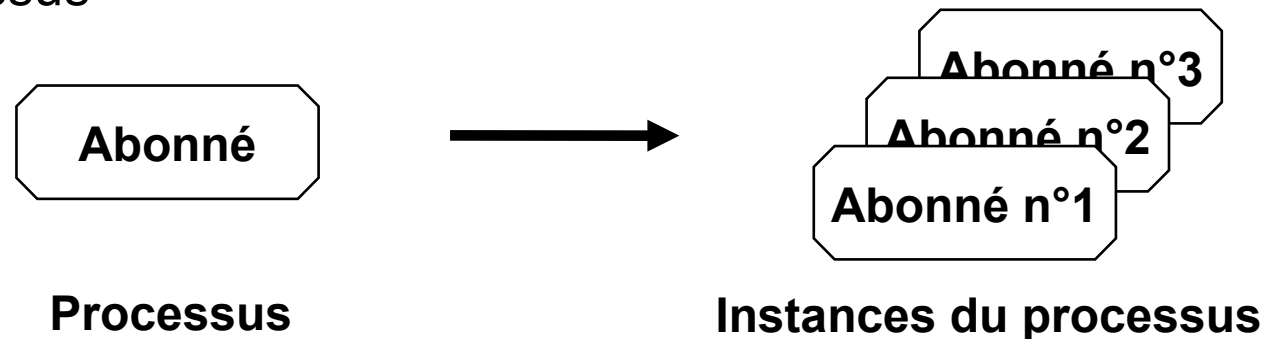


$$S_C = S_{R1} \cup S_{R2} \cup S_{R3}$$

# Processus SDL

---

- Machine à états finis étendue
- Définition d'un processus = spécification d'un type de processus
  - Plusieurs instances d'un processus peuvent être créées et exister simultanément
  - Des instances d'un processus peuvent exister à la création du système, ou peuvent être créées suite à une requête d'un autre processus



# Processus SDL

---

- Spécification d'un processus SDL
  - **Nom** du processus
  - Couple de nombres entiers  $(n,m)$  (par défaut  $(1,\infty)$ )
    - $n$  : nombre d'instances créées à l'initialisation du système
    - $m$  : nombre maximum d'instances simultanées du processus
  - Paramètres formels (informations nécessaires à la création du processus)
  - Définition des **types de données**
  - Déclaration des **variables**
  - Définition des **temporisateurs**
  - Définition des procédures et des macros
  - **Corps** du processus : graphe de la machine à états finis étendue



# Déclaration de variables

---

- Mot-clé : DCL

```
DCL A BOOLEAN;  
    B NATURAL;  
    P PID;  
  
DCL C BOOLEAN := TRUE;  
DCL N INTEGER := 0;
```

# Définition de temporisateurs

---

- Mot-clé : `TIMER`

```
TIMER T1, T2 := 5;
```

- T1 temporisateur de durée indéterminée (de 0 à l'infini)
  - T2 temporisateur de durée 5
- Attribution d'une durée pour T1 au déclenchement

# Corps d'un processus

---

- Transition d'initialisation vers l'état initial
- Ensemble d'états et de transitions
- Pour chaque état
  - Nom de l'état
  - Ensemble des stimuli possibles dans cet état
  - Transition effectuée à la réception de chaque stimulus
    - ensemble d'actions (TASK, OUTPUT, CREATE REQUEST, PROCEDURE CALL, DECISION, SET, RESET)□
    - terminateur de transition (NEXTSTATE, JOIN, STOP)□

# Corps d'un processus (suite) □

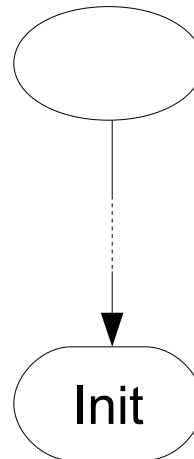
---

- Transition d'initialisation vers l'état initial **START**

- SDL/PR

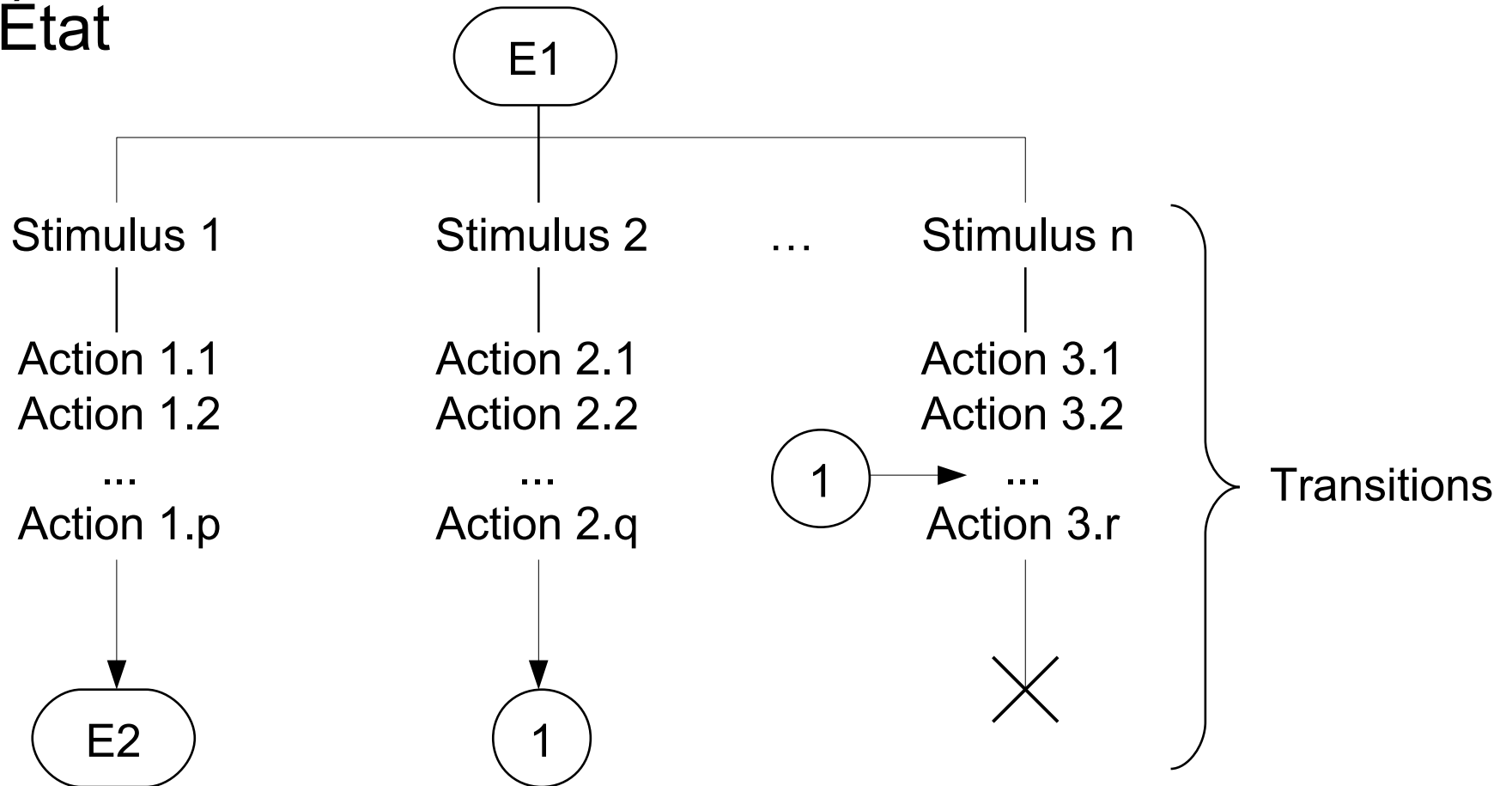
```
START;  
    < Séquence d'actions >  
NEXTSTATE Init;
```

- SDL/GR



# Corps d'un processus

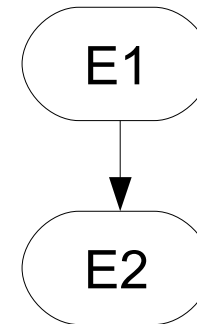
- État



# Termineurs de transition

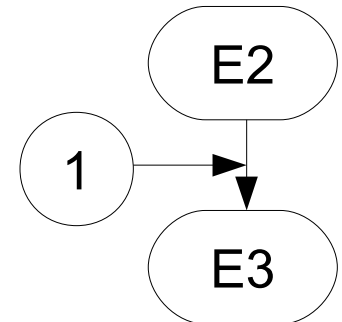
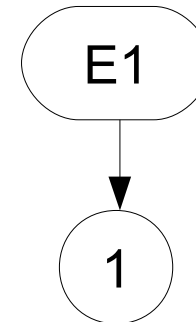
- Arrivée dans un nouvel état

```
STATE E1;  
  < Stimulus >  
  < Actions >  
  NEXTSTATE E2;
```



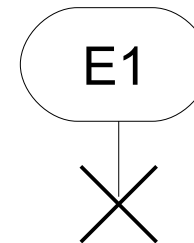
- Débranchement vers un label

```
STATE E1;          STATE E2;  
  < Stimulus >      < Stimulus >  
  < Actions >       1: < Actions >  
  JOIN 1;           NEXTSTATE E3;
```



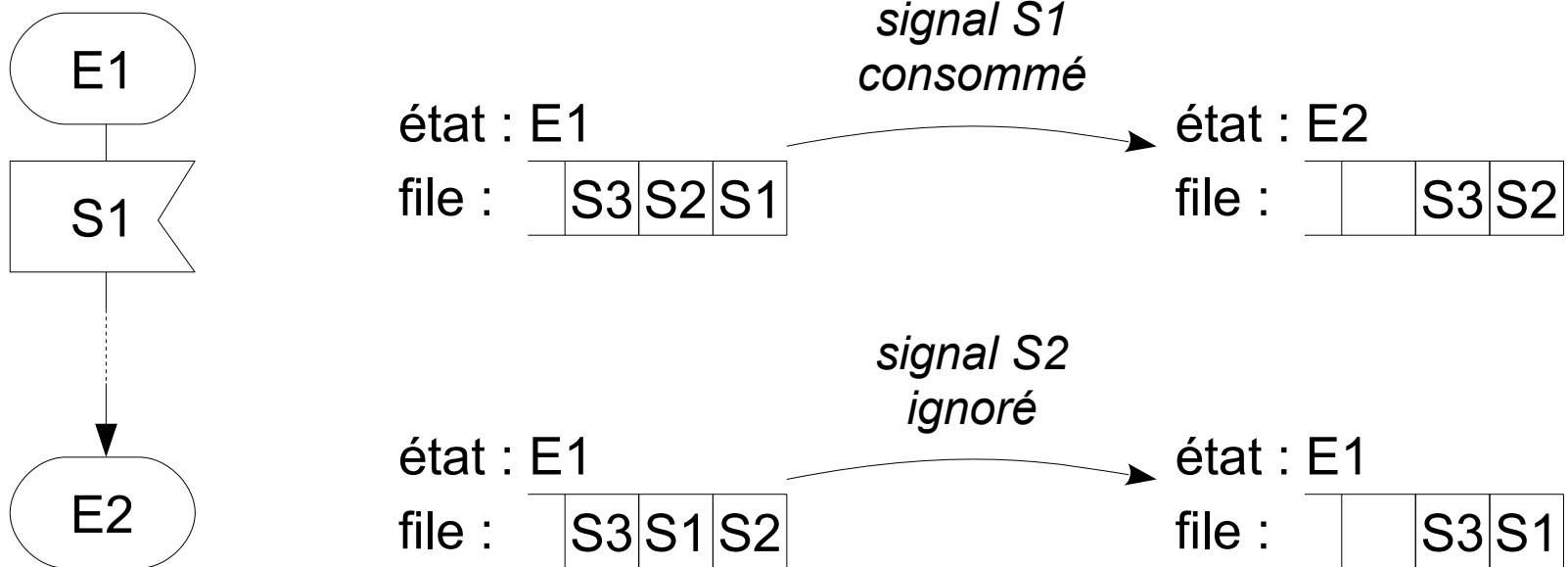
- Terminaison du processus

```
STATE E1;  
  < Stimulus >  
  < Actions >  
  STOP;
```



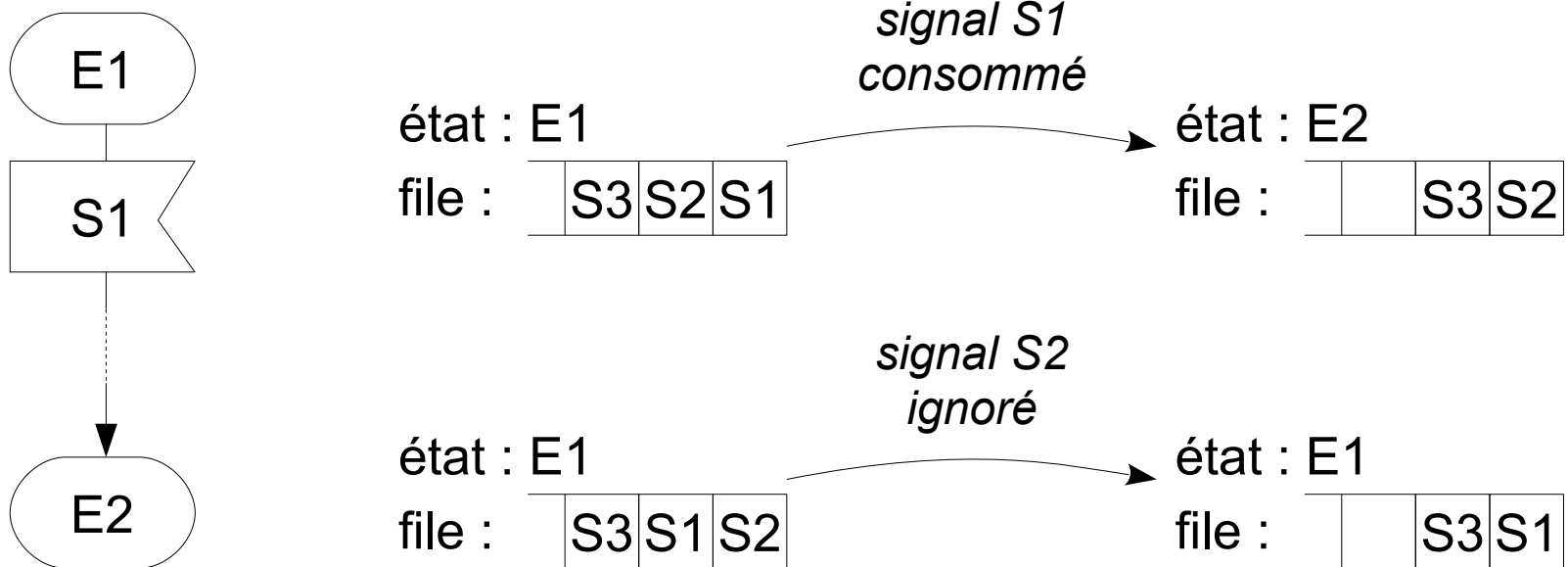
# Types de stimuli

- Réception d'un signal **INPUT**
  - si attendu : lecture du message et exécution de la transition
  - sinon : destruction du message, aucune action



# Types de stimuli

- Réception d'un signal **INPUT**
  - si attendu : lecture du message et exécution de la transition
  - sinon : destruction du message, aucune action

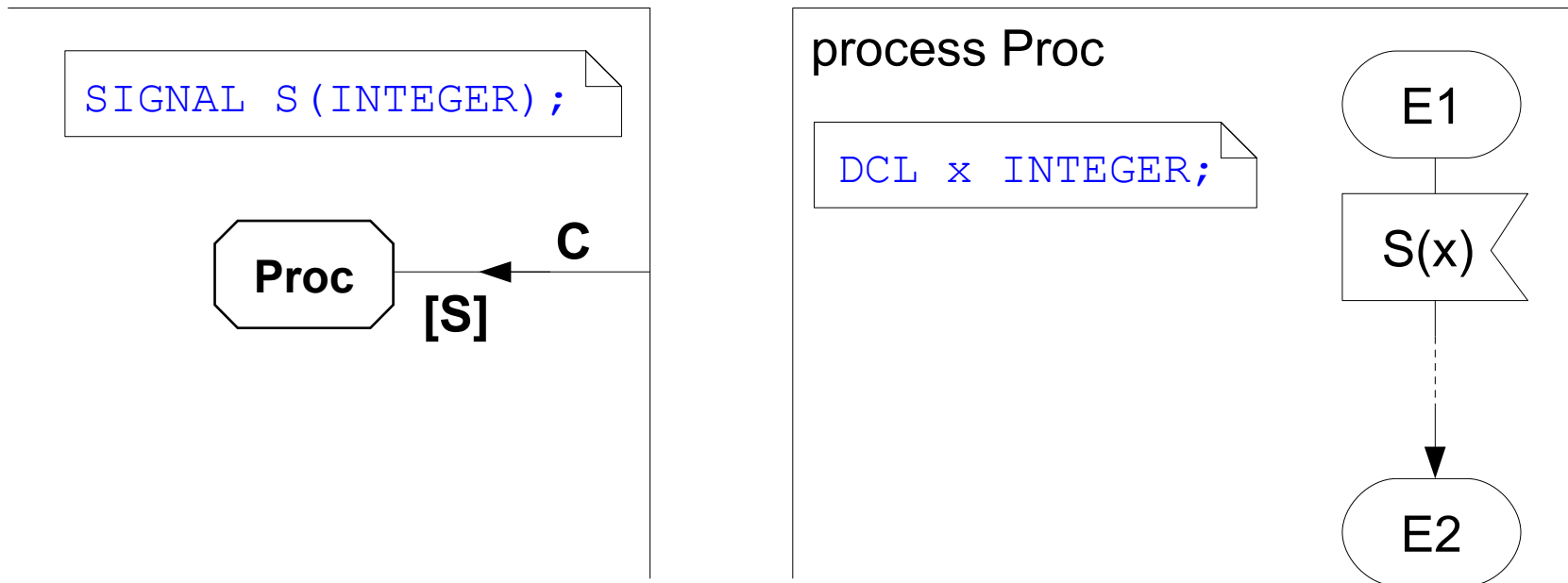




# Types de stimuli

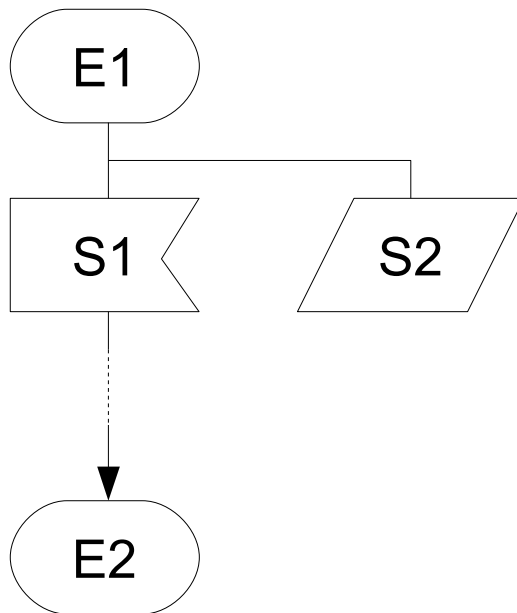
Réception d'un signal `INPUT` portant une valeur

- Valeur stockée dans une variable pour être utilisée ensuite



# Types de stimuli

- Sauvegarde d'un signal *SAVE*
  - reste en tête de la file d'attente, lecture du signal suivant
  - file non FIFO



état : E1

file : 

	S3	S1	S2
--	----	----	----

*signal S1  
consommé*

état : E2

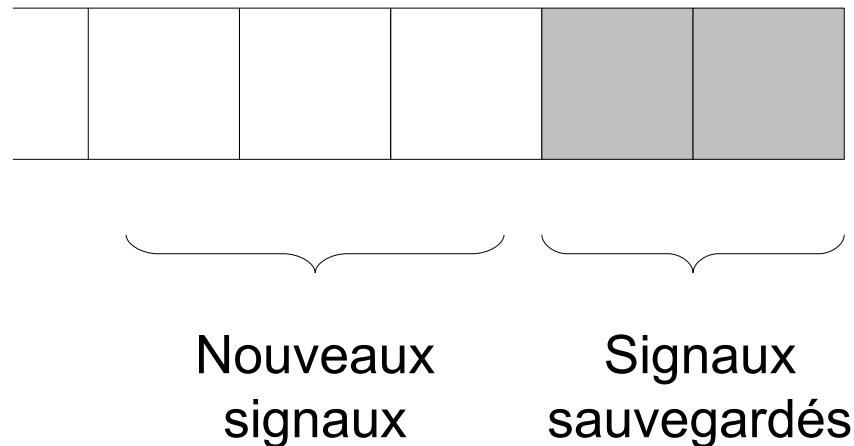
file : 

		S3	S2
--	--	----	----

*signal S2  
sauvegardé*

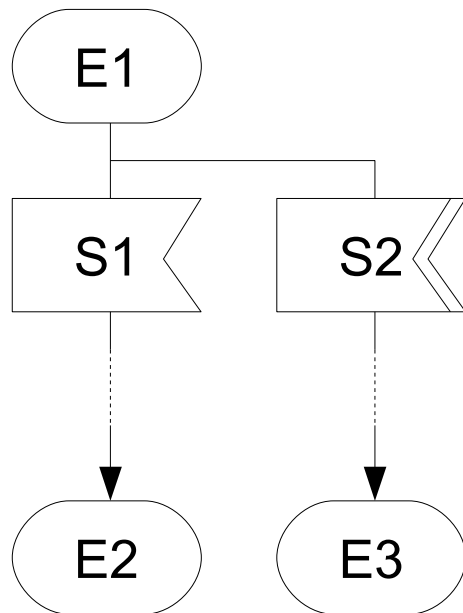
# Types de stimuli

- Sémantique de la sauvegarde
  - file d'attente séparée en deux sous-files
  - file non FIFO



# Types de stimuli

- Réception d'un signal prioritaire
  - si présent dans la file, lu avant les autres messages
  - file non FIFO



état : E1

file : 

	S3	S2	S1
--	----	----	----

état : E3

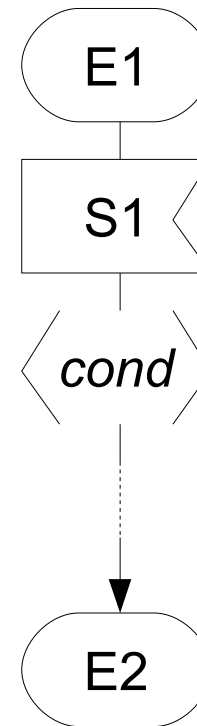
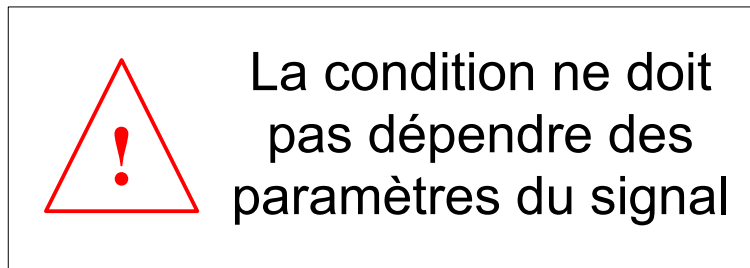
file : 

		S3	S1
--	--	----	----

*signal prioritaire  
S2 consommé*

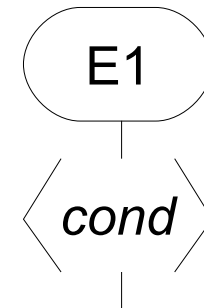
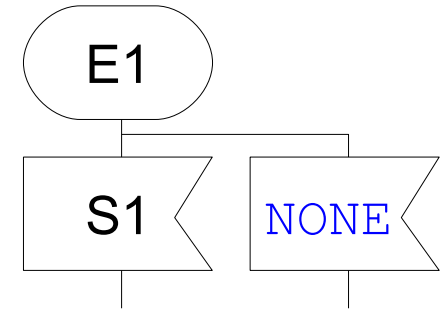
# Types de stimuli

- Réception conditionnée
  - si condition vérifiée, signal consommé
  - sinon, signal sauvegardé



# Types de stimuli

- Transition spontanée
  - franchissable à tout moment
  - non prioritaire sur la réception de signaux
- Signal continu
  - transition déclenchée par une condition

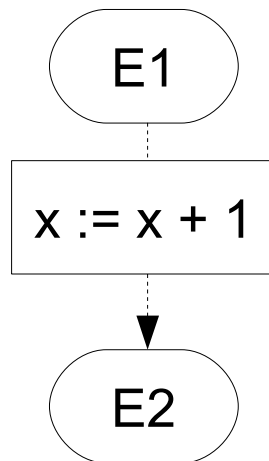


# Actions dans une transition

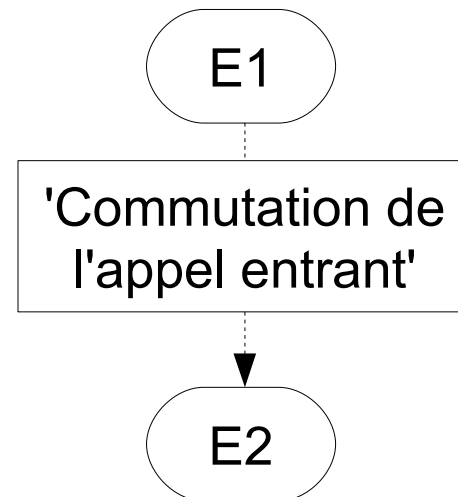
- Tâche **TASK**

- Affectation
- Texte informel

TASK  $X := X + 1;$



TASK 'Commutation de l'appel entrant';

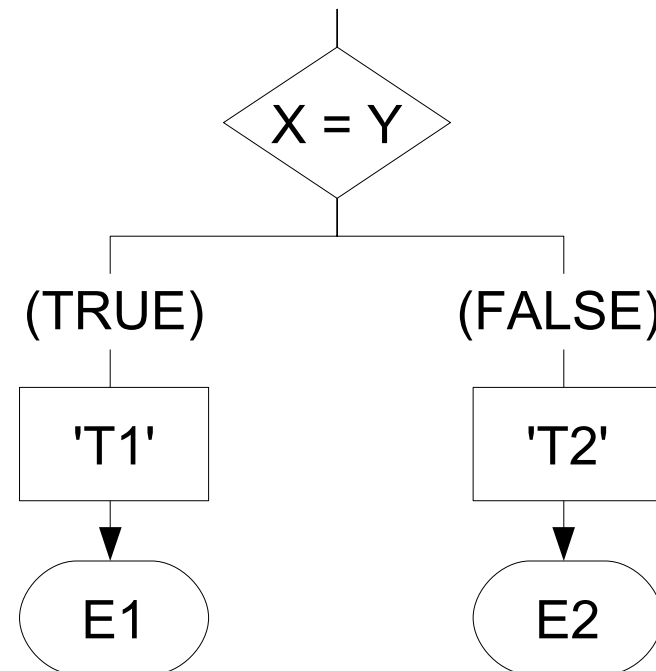


# Actions dans une transition

- Décision

- Question : expression ou texte informel
- Ensemble de réponses : valeurs de l'expression ou texte informel
- Pour chaque réponse, la transition à exécuter

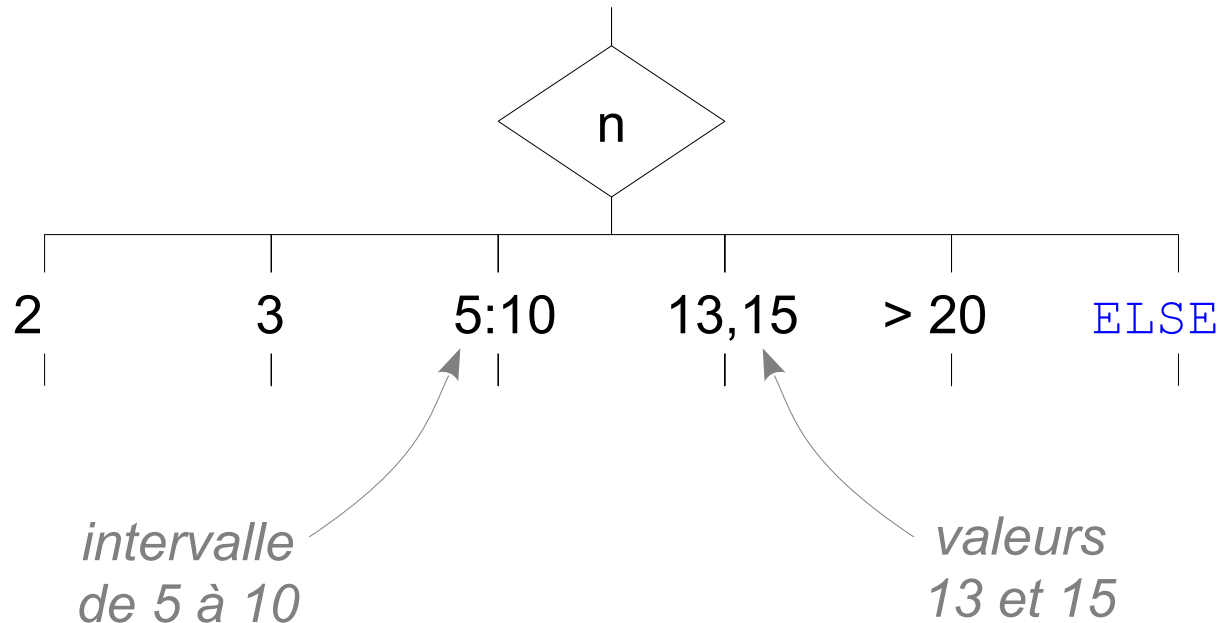
```
DECISION X = Y;  
  (TRUE) : TASK 'T1';  
           NEXTSTATE E1;  
  (FALSE) : TASK 'T2';  
           NEXTSTATE E2;  
ENDDECISION;
```





# Décision

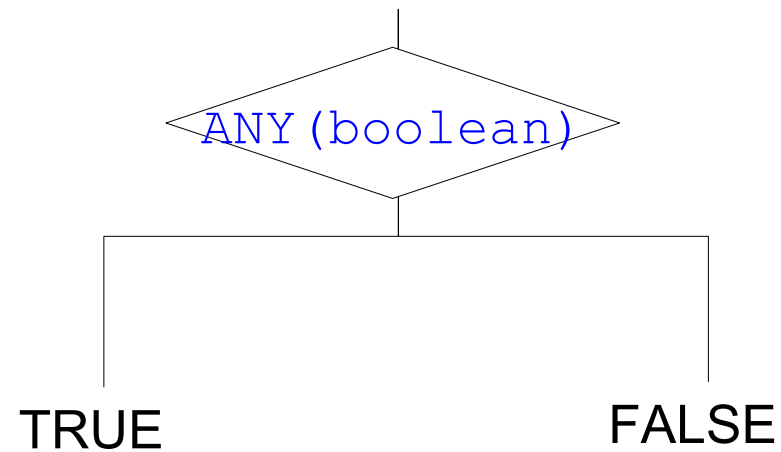
- Utilisation de **ELSE** pour grouper toutes les autres réponses possibles



# Décision

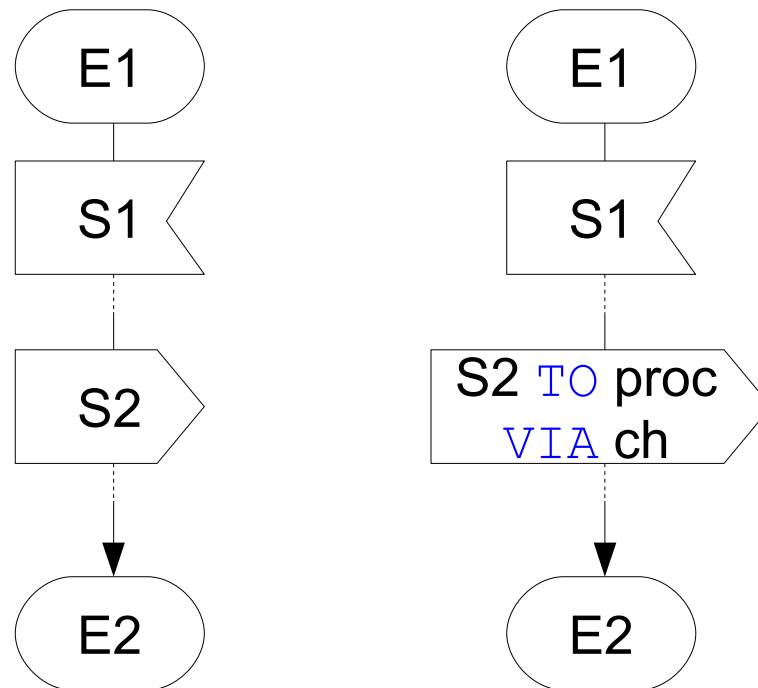
---

- Décision non déterministe



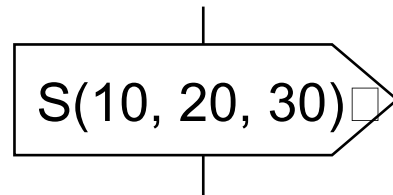
# Actions dans une transition

- Émission d'un signal **OUTPUT**
  - possibilité de spécifier le processus destinataire et le canal à utiliser en cas d'ambiguïté

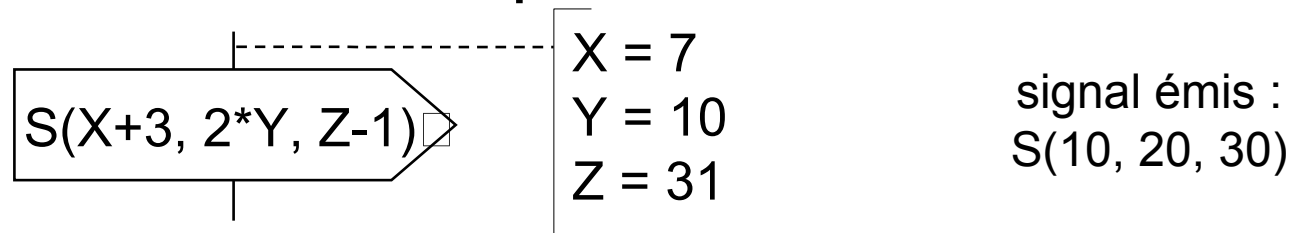


# Exemples d'OUTPUT

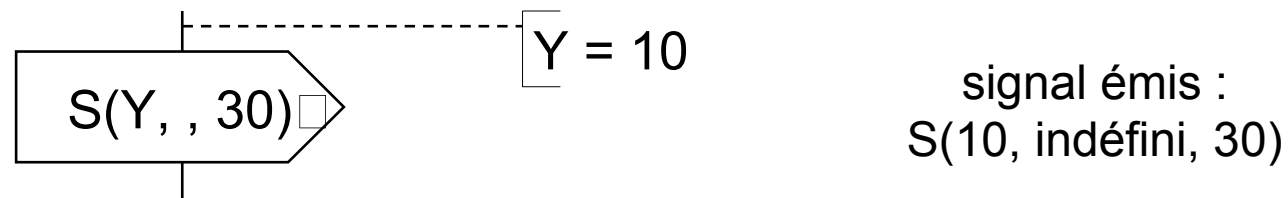
- Signal S avec trois valeurs associées



- Signal S avec trois expressions à évaluer



- Signal S avec une valeur indéterminée



# Expression du temps

- Temporisateur `TIMER`
  - méta-processus émettant un signal lors de son expiration
- Valeur du temps courant absolu : `NOW`
- Armement d'un temporisateur

```
SET (T0);  
SET (NOW + 300, T1);
```

`SET (NOW + 300, T1)` □

`TIMER T1;`

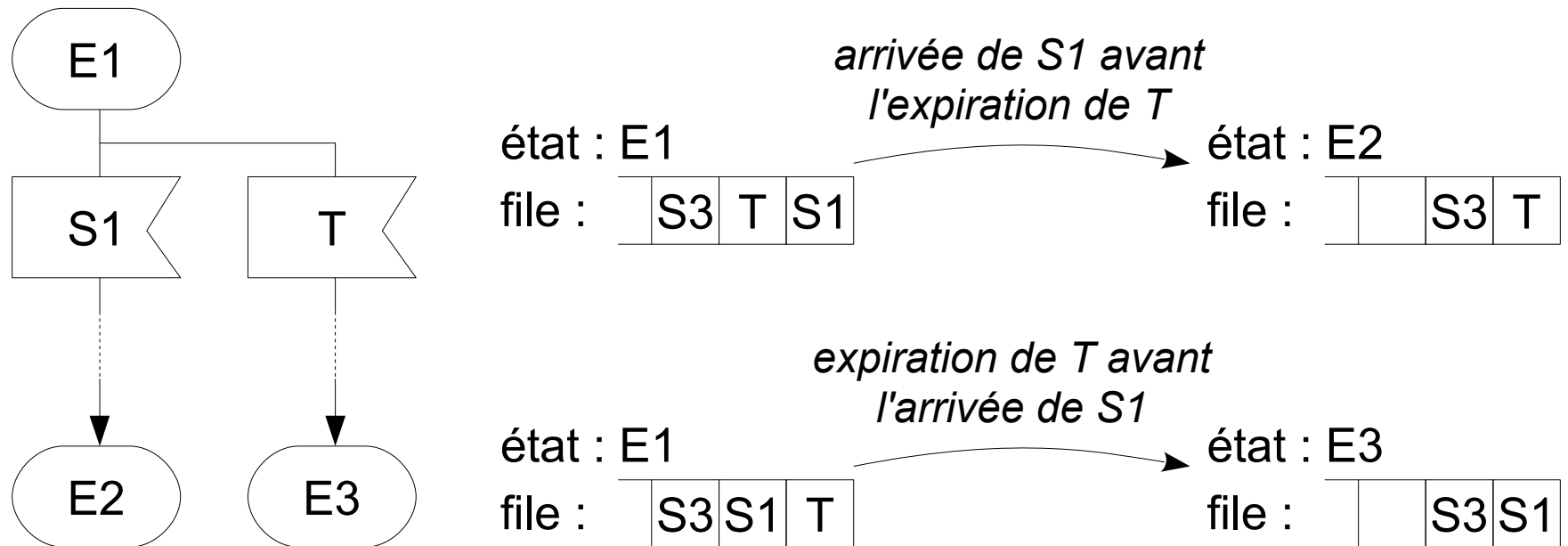
- Désarmement d'un temporisateur

```
RESET (T0);  
RESET (T1, T2, T3);
```

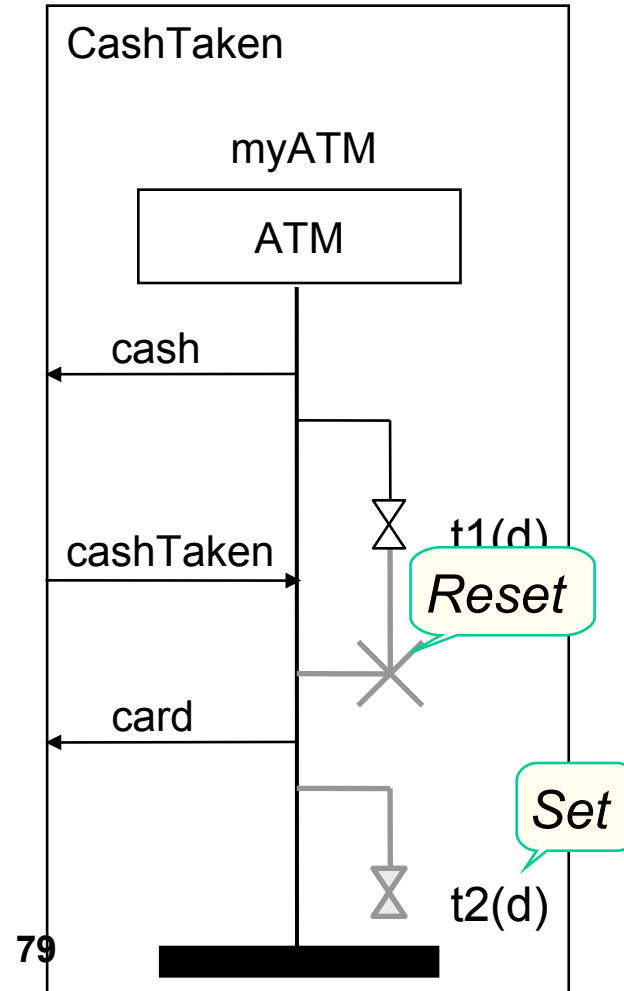
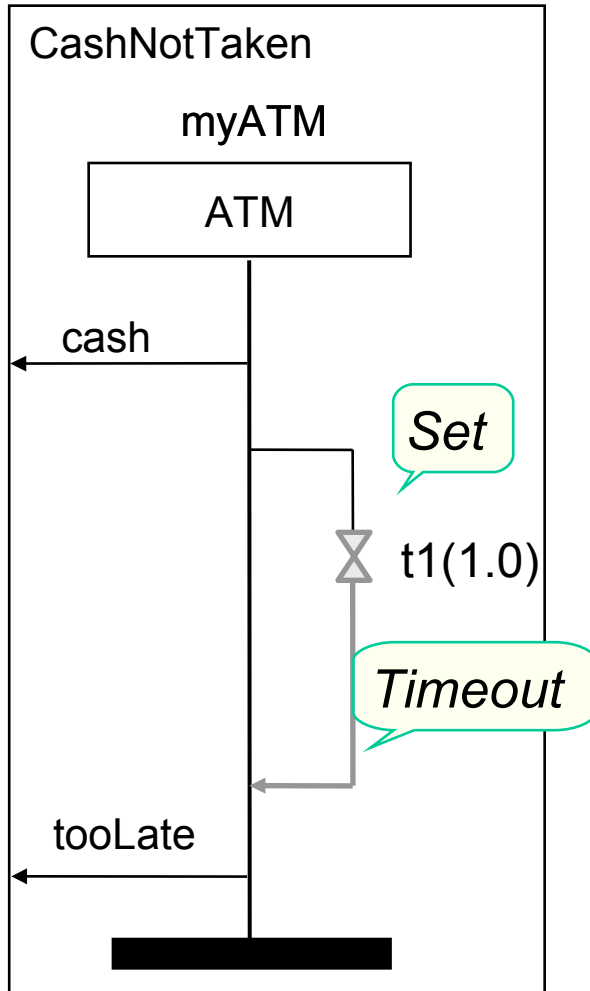
`RESET (T1, T2, T3)` □

# Expiration d'un temporisateur

- Arrivée d'un signal dans la file d'attente du processus
  - Signal non prioritaire

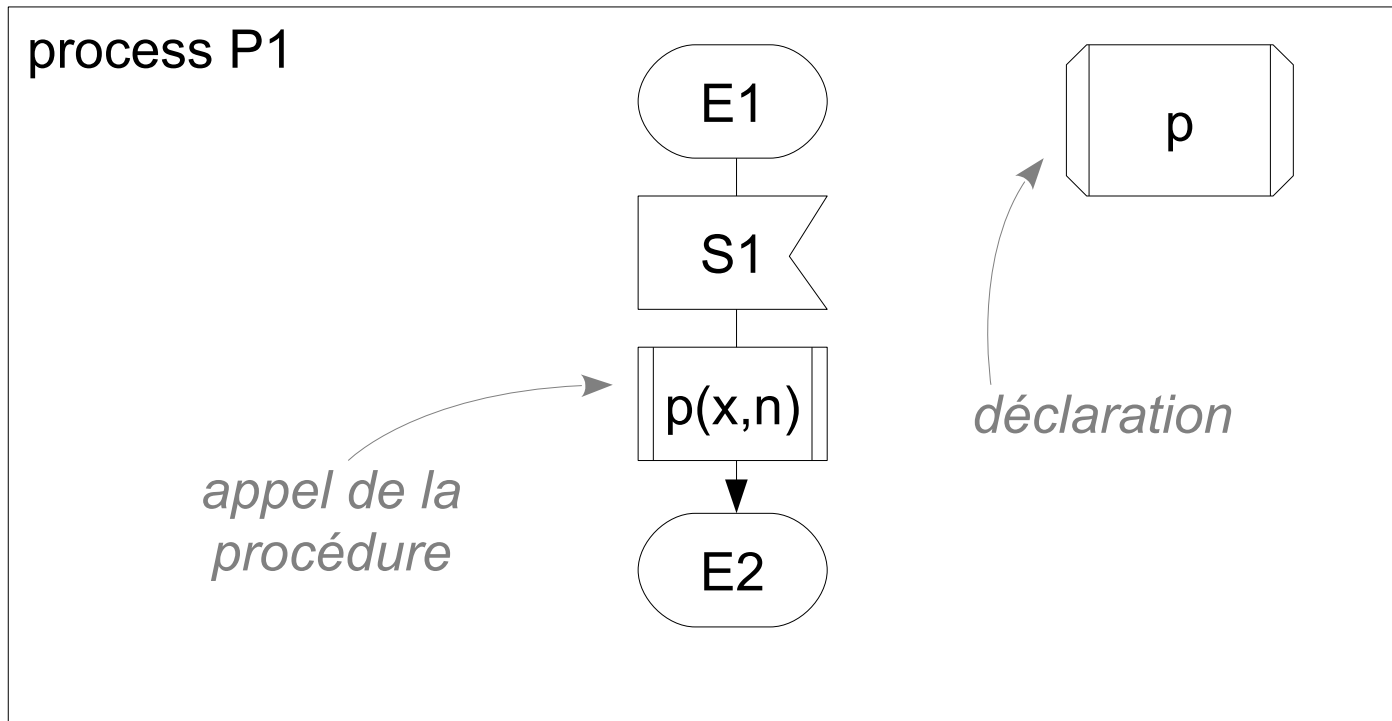


# Temps et MSC



# Procédures

- Morceau de processus remplissant une fonctionnalité



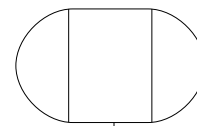


# Paramètres de procédure

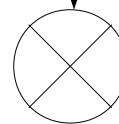
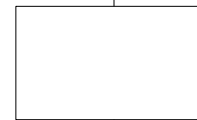
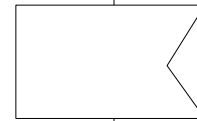
- Paramètres introduits par **FPAR**
  - **IN** : par défaut, appel par valeur
  - **IN/OUT** : appel par référence

procedure p

```
FPAR  
IN/OUT x INTEGER,  
IN n INTEGER ;
```



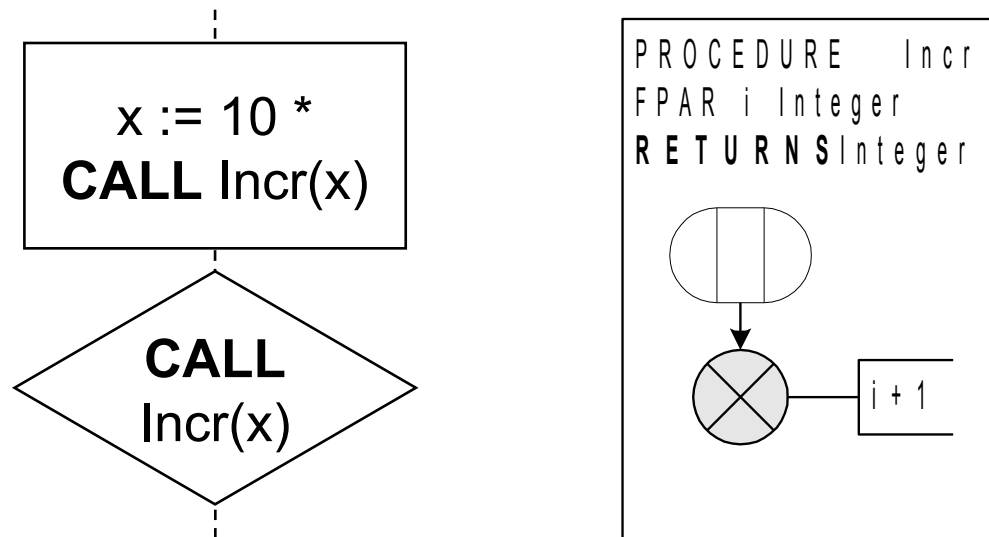
*début*



*fin  
(return)*

# Appel d'une procédure

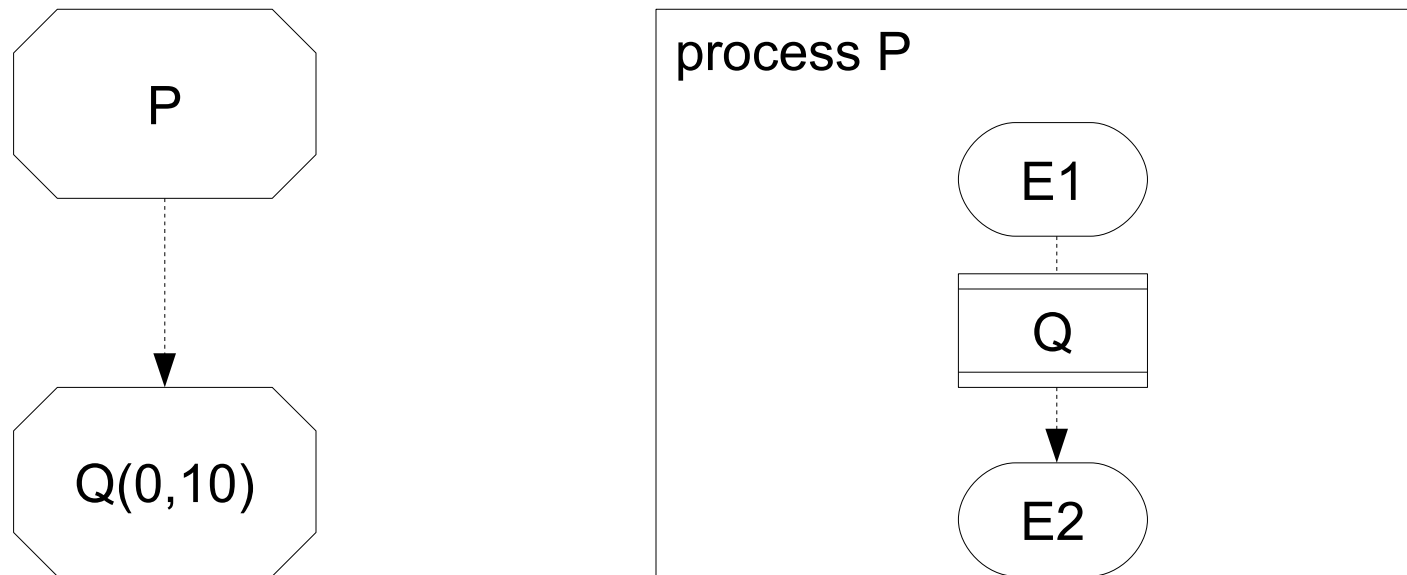
- Appel d'une procédure qui retourne une valeur



- Les procédures peuvent être appelées comme des fonctions dans des expressions
  - Rendent le retour explicite*

# Création dynamique de processus

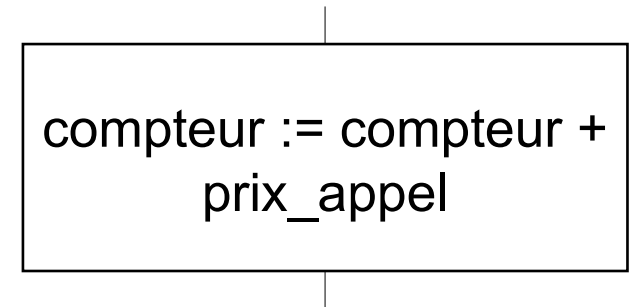
- Création d'une instance de processus par une instance d'un autre processus dans le même bloc



# Actions dans une transition : récapitulatif

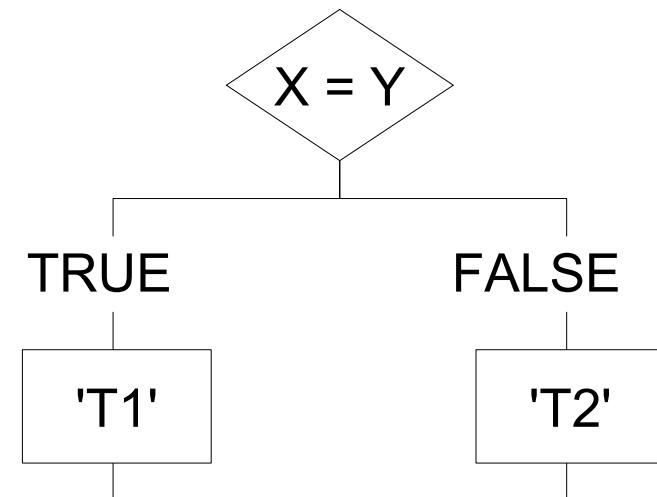
- Tâche : affectation ou texte informel

```
TASK compteur :=  
    compteur + prix_appel;
```



- Décision

```
DECISION X = Y;  
    (TRUE) : TASK  
    'T1';  
    (FALSE) : TASK  
    'T2';  
ENDDECISION;
```

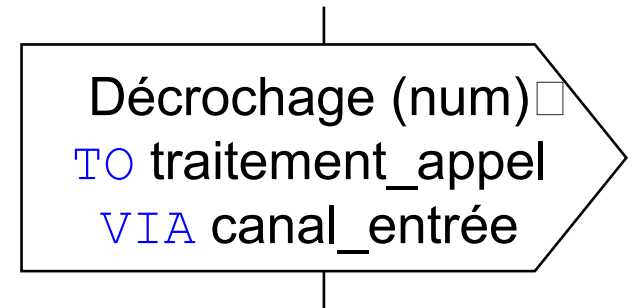


# Actions dans une transition : récapitulatif

---

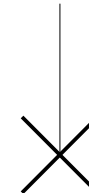
- Émission de signal

```
OUTPUT Décrochage(num)  
  TO traitement_appel  
  VIA canal_entrée
```



- Terminaison d'instance de processus

```
STOP;
```



# Actions dans une transition : récapitulatif

---

- Armement de temporisateur

SET (NOW+80, T1) ;

SET (NOW + 80, T1) □

- Désarmement de temporisateur

RESET (T1) ;

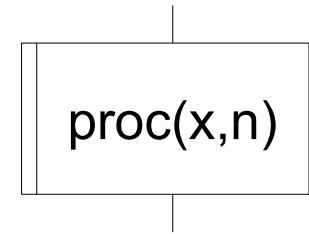
RESET (T1) □

# Actions dans une transition : récapitulatif

---

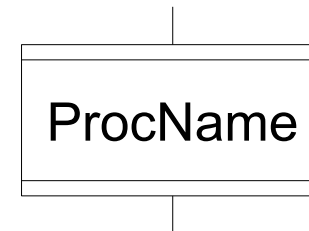
- Appel de procédure

```
CALL proc(x,n) ;
```



- Création de processus

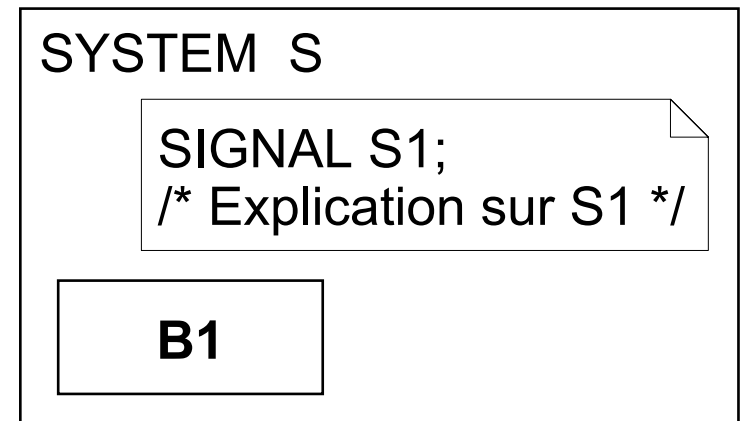
```
CREATE ProcName ;
```



# Commentaires

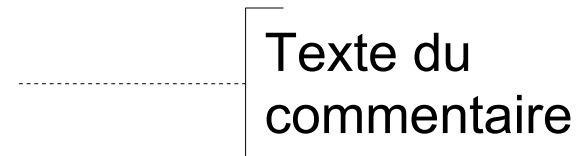
- Commentaire informel

```
SYSTEM S;  
  SIGNAL S1;  
  /* Explication sur S1 */  
  BLOCK B1 REFERENCED;  
ENDSYSTEM S;
```



- Commentaire structuré

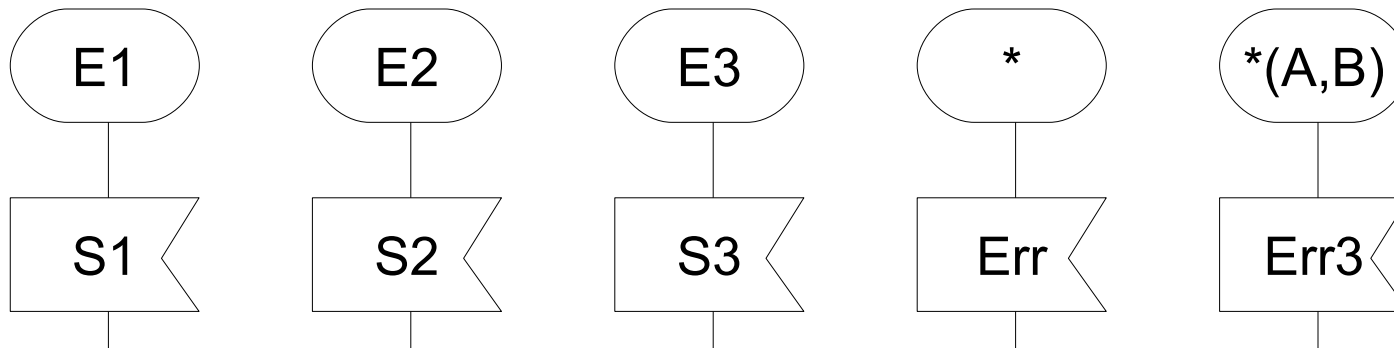
```
COMMENT 'Texte du commentaire'
```





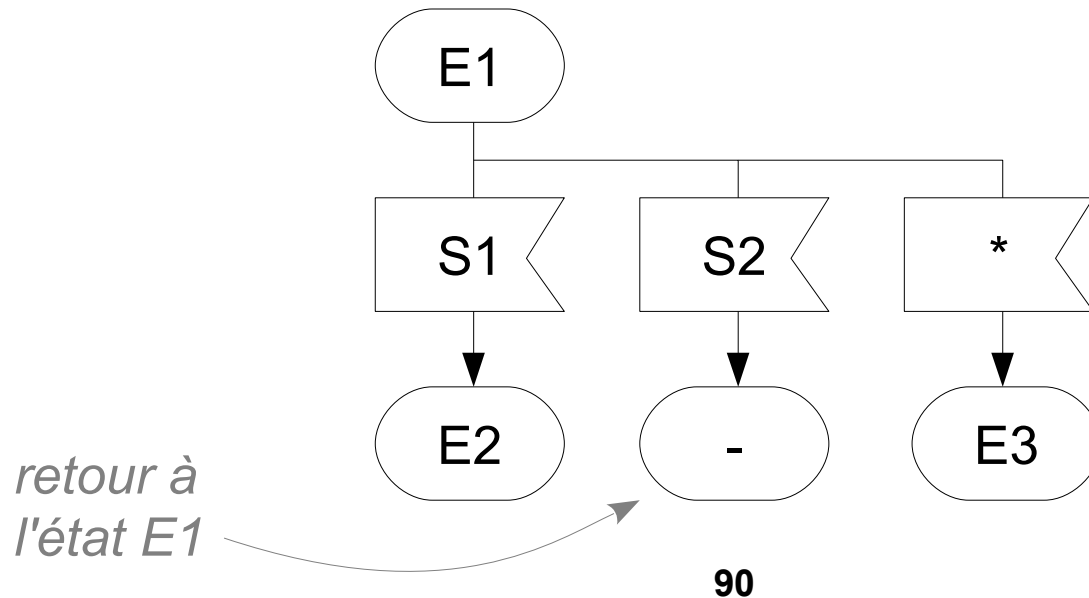
# Facilités d'écriture

- État \*
- transition applicable dans tous les états du processus
- État  $^*(A,B)$ - transition applicable dans tous les états sauf A et B



# Facilités d'écriture

- État -
  - Retour dans le même état
- Input \*
  - réception de tous les signaux autres que ceux spécifiés



# Types de données

---

- Types prédéfinis, avec opérations usuelles
  - `BOOLEAN` : booléens
  - `NATURAL` : entiers naturels
  - `INTEGER` : entiers relatifs
  - `REAL` : réels
  - `CHARACTER` : caractères
  - `CHARSTRING` : chaînes de caractères
  - `TIME` : temps absolu (sous-type de `REAL`)
  - `DURATION` : durée (sous-type de `REAL`)
  - `PID` : identificateur de processus

# Types de données

---

- Tableaux
  - Mot-clé : `ARRAY`
  - Dimension : liste des indices
  - Type des éléments du tableau : sorte
- Ex : Tableau de caractères à deux dimensions

```
NEWTYPE Tab  
    ARRAY (NATURAL, NATURAL, CHARACTER);  
ENDNEWTYPE Tab;
```

# Types de données

---

## Déclaration

```
NEWTYPE Tab  
    ARRAY (NATURAL, CHARACTER);  
ENDNEWTYPE Tab;
```

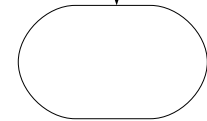
## Initialisation

```
DCL T1 Tab;  
DCL T2 := (. 'a' .);
```

*initialise tous les  
éléments de T2 à 'a'*

## Utilisation

T(2) := 'c'



# Types de données

- Structures

- Mot-clé : STRUCT
- Liste de types de champs

Ex : Structure à deux champs

```
NEWTYPE Srucl1 STRUCT
    Field1 CHARSTRING;
    Field2 Tab;
ENDNEWTYPE Srucl1;
```

## Déclaration

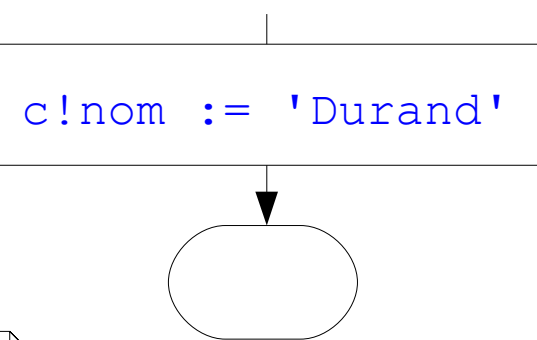
```
NEWTYPE Client STRUCT
    nom CHARSTRING;
    numero NATURAL;
ENDNEWTYPE Client;
```

## Initialisation

```
DCL c Client := (. 'Dupont', 34526 .);
```

## Utilisation

```
c!nom := 'Durand'
```



# Types de données

---

- Types énumérés

```
NEWTYPE WeekDay  
    Litterals mon, tue, wed, thu, fri, sat, sun;  
ENDNEWTYPE WeekDay;
```

- Restriction des valeurs d'un type

```
SYNTYPE Index_T = Natural  
    CONSTANTS 1:12  
ENDSYNTYPE Index_T;
```

```
SYNTYPE WeekEnd = WeekDay  
    DEFAULT sun  
    CONSTANTS sat sun  
ENSYNTYPE WeekEnd;
```

- Constantes

```
SYNONYM pi REAL = 3.14159;
```

# Types de données

---

Type abstrait de données

=

sorte + opérateurs + axiomes

- Sorte : ensemble de valeurs
- Opérateurs : opérations sur ces valeurs
- Axiomes : propriétés définissant le comportement des opérateurs



# Types de données

---

- Exemple : définition d'un type `bit`

```
NEWTYPE bit
  LITERALS un, zero;
  OPERATORS
    inv : bit -> bit ;
  AXIOMS
    inv(zero) == un;
    inv(un) == zero;
ENDNEWTYPE bit;
```