# [Langage noyau]

pptPlex Section Divider

The slides after this divider will be grouped into a section and given the label you type above.  Feel free to move this slide to any position in the deck.

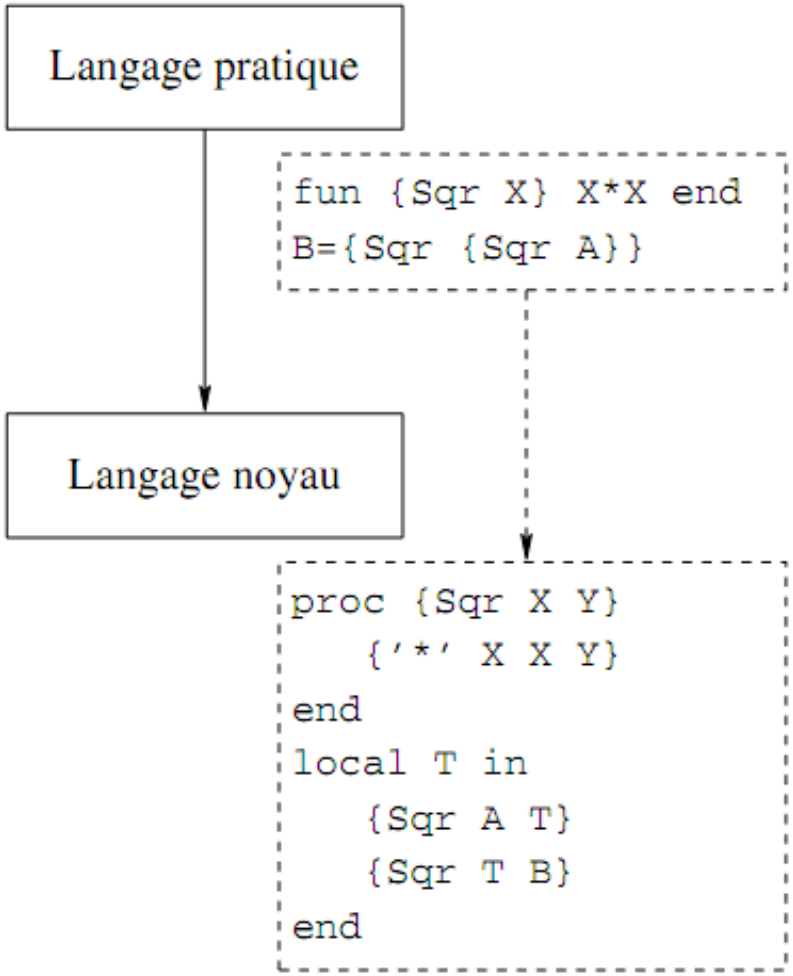
# But de cette section

- Illustrer concrètement les idées précédentes
  - Concepts de programmation
  - Paradigme
  - Sémantique
- Etapes
  - Définition d'un langage simple (langage « noyau »)
  - Définition de sa sémantique opérationnelle
  - Exemples de raisonnement en utilisant sa sémantique
  - Montrer comment certains concepts de programmation permettent d'en obtenir d'autres

# Pourquoi un langage simple ?

- Un langage pratique
  - Beaucoup de concepts
  - Des incontournables, d'autres pour le confort du programmeur
  - Certains concepts exprimables par d'autres plus fondamentaux
- Langage noyau
  - Simple car que des concepts fondamentaux
  - Simplifie le raisonnement sur la sémantique

# Exemple




- *Fournit des abstractions utiles pour le programmeur*
- *Peut être étendu avec des abstractions linguistiques*
- *Contient un petit nombre de concepts, qui sont intuitifs*
- *Est facile à comprendre et à raisonner pour le programmeur*
- *Est défini par une sémantique formelle (opérationnelle, axiomatique ou dénotationnelle)*

**Figure 2.4** L'approche du langage noyau pour la sémantique.

# Langage noyau : les instructions

| ⟨s⟩ ::= | |
|---|---|
| **skip** | Instruction vide |
| ⟨s⟩$_1$ ⟨s⟩$_2$ | Séquence d'instructions |
| **local** ⟨x⟩ **in** ⟨s⟩ **end** | Déclaration de variable |
| ⟨x⟩$_1$=⟨x⟩$_2$ | Lien variable-variable |
| ⟨x⟩=⟨v⟩ | Création de valeur |
| **if** ⟨x⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end** | Instruction conditionnelle |
| **case** ⟨x⟩ **of** ⟨pattern⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end** | Correspondance de formes |
| {⟨x⟩ ⟨y⟩$_1$ ... ⟨y⟩$_n$} | Application de procédure |

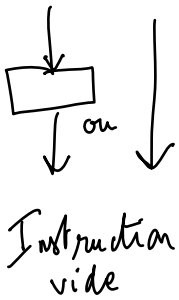
**Tableau 2.1** Le langage noyau déclaratif.

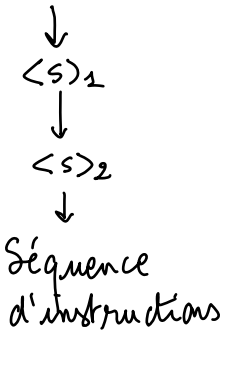
- <s> est à remplacer par une instruction
- <x>, <x>$_1$,<y> … par des variables
- <pattern> par un motif d'enregistrement
- <v> par une valeur

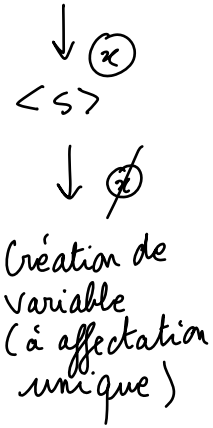
# Langage noyau : les instructions

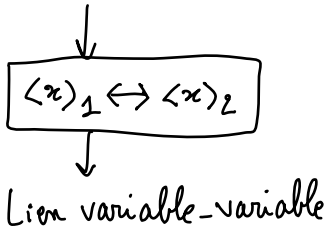
| ⟨s⟩ ::= | |
|---|---|
| **skip** | Instruction vide |
| ⟨s⟩$_1$ ⟨s⟩$_2$ | Séquence d'instructions |
| **local** ⟨x⟩ **in** ⟨s⟩ **end** | Création de variable |
| ⟨x⟩$_1$=⟨x⟩$_2$ | Lien variable-variable |
| ⟨x⟩=⟨v⟩ | Création de valeur |
| **if** ⟨x⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end** | Instruction conditionnelle |
| **case** ⟨x⟩ **of** ⟨pattern⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end** | Correspondance de formes |
| {⟨x⟩ ⟨y⟩$_1$ ... ⟨y⟩$_n$} | Application de procédure |

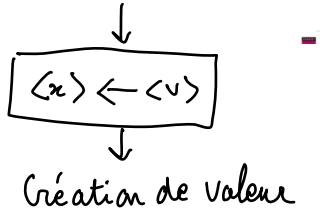
**Tableau 2.1** Le langage noyau déclaratif.

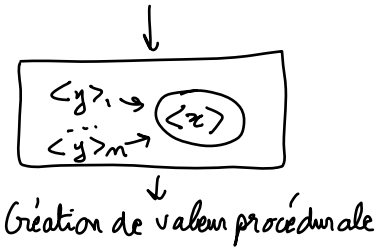


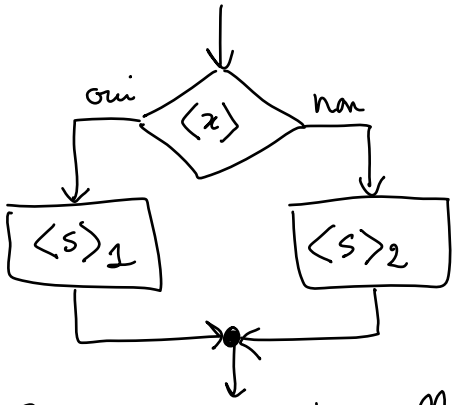



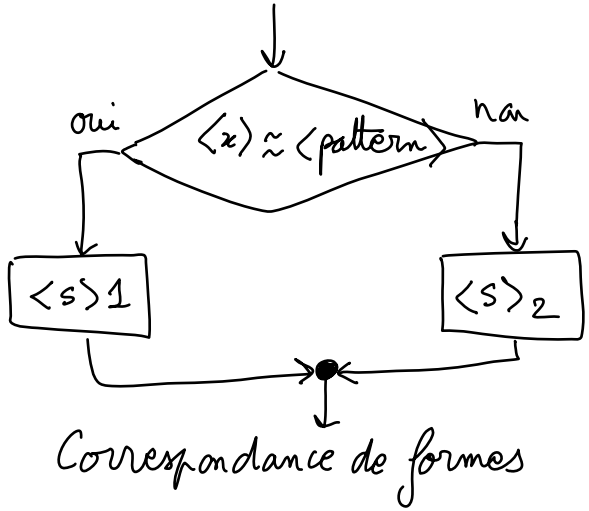



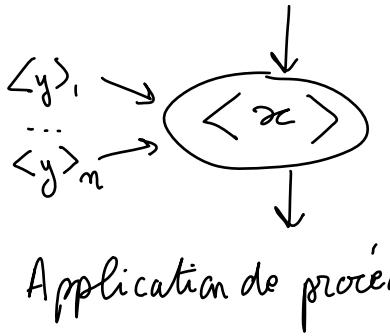



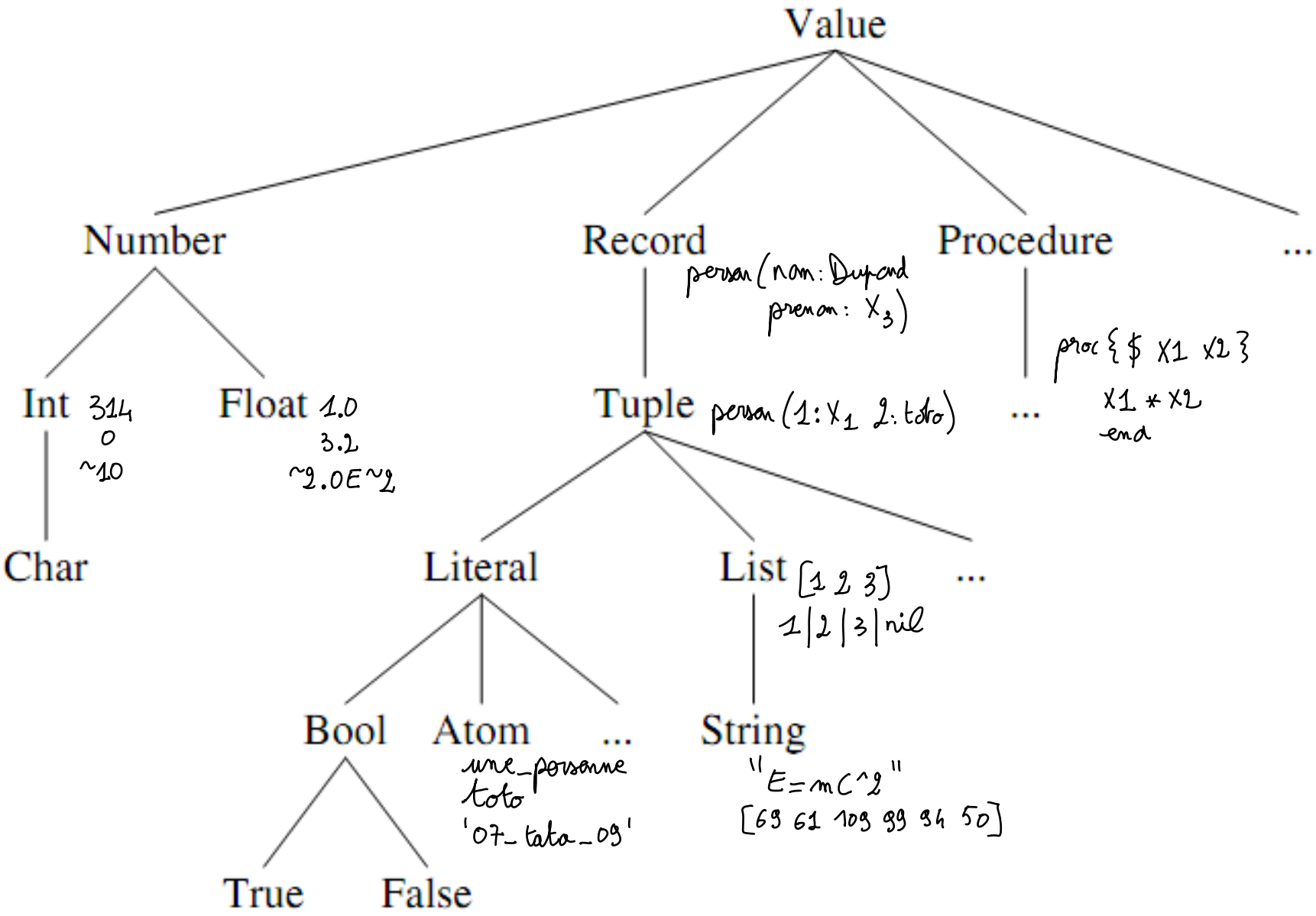



# Langage noyau : les valeurs

| | | |
|---|---|---|
| ⟨v⟩ | : := | ⟨number⟩ \| ⟨record⟩ \| ⟨procedure⟩ |
| ⟨number⟩ | : := | ⟨int⟩ \| ⟨float⟩ |
| ⟨record⟩, ⟨pattern⟩ | : := | ⟨literal⟩ |
| | | \| ⟨literal⟩(⟨feature⟩$_1$ : ⟨x⟩$_1$ $\cdots$ ⟨feature⟩$_n$ : ⟨x⟩$_n$) |
| ⟨procedure⟩ | : := | **proc** { $ ⟨x⟩$_1$ $\cdots$ ⟨x⟩$_n$ } ⟨s⟩ **end** |
| ⟨literal⟩ | : := | ⟨atom⟩ \| ⟨bool⟩ \| ... |
| ⟨feature⟩ | : := | ⟨atom⟩ \| ⟨bool⟩ \| ⟨int⟩ \| ... |
| ⟨bool⟩ | : := | **true** \| **false** |

**Tableau 2.2** Les valeurs dans le langage noyau déclaratif.

# Valeurs


Value
Number
Record
Procedure
...
Int
314
0
~10
Float
1.0
3.2
~2.0E~2
Char
Tuple
Literal
List
[1 2 3]
1|2|3|nil
Bool
Atom
...
String
True
False
"E=mC^2"
[69 61 109 99 94 50]

# Langage noyau : les opérations

| Opération | Description | Type des arguments |
|---|---|---|
| `A==B` | Comparaison d'égalité | Value |
| `A\=B` | Comparaison d'inégalité | Value |
| `{IsProcedure P}` | Test si procédure | Value |
| `A=<B` | Comp. plus petit ou égal | Number ou Atom |
| `A<B` | Comparaison plus petit | Number ou Atom |
| `A>=B` | Comp. plus grand ou égal | Number ou Atom |
| `A>B` | Comparaison plus grand | Number ou Atom |
| `A+B` | Addition | Number |
| `A-B` | Soustraction | Number |
| `A*B` | Multiplication | Number |
| `A div B` | Division (entier) | Int |
| `A mod B` | Modulo | Int |
| `A/B` | Division (flottant) | Float |
| `{Arity R}` | Arité | Record |
| `{Label R}` | Étiquette | Record |
| `R.F` | Sélection de champ | Record |

**Tableau 2.3** Quelques opérations de base.

- Les opérations correspondent à des appels de procédures réécrits plus lisiblement.

# [Sémantique opérationnelle]

pptPlex Section Divider

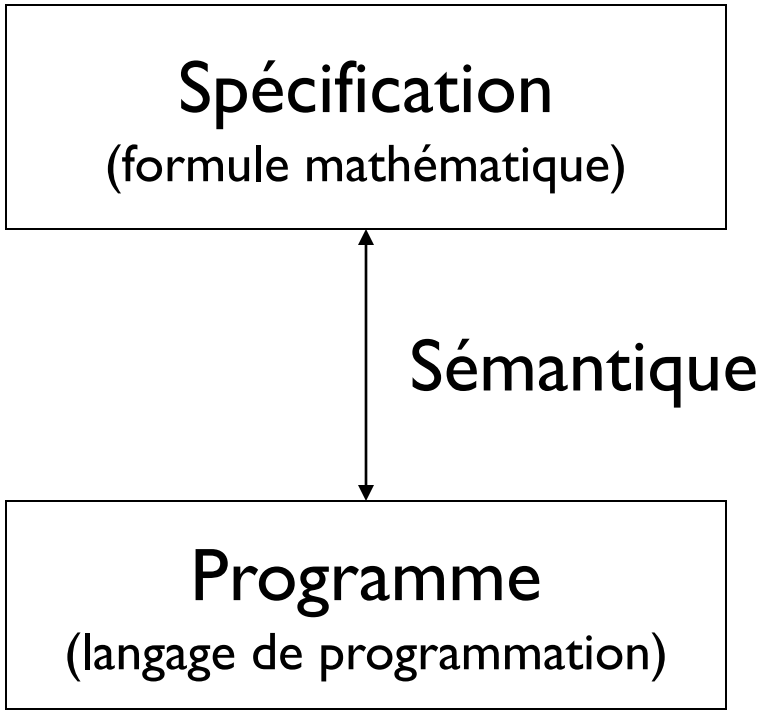
The slides after this divider will be grouped into a section and given the label you type above.  Feel free to move this slide to any position in the deck.

# Votre programme est-il correct?

- ”Un programme est correct (exact) quand il fait ce qu’on veut qu’il fasse”
- Comment vérifier ?
- Deux points
  - **La spécification du programme**
    - une définition du résultat du programme en termes de l’entrée
    - typiquement une fonction ou relation mathématique
  - **La sémantique du langage (opérationnelle)**
    - un modèle précis des opérations du langage de programmation
- On doit prouver que la spécification est satisfaite par le programme, quand il s’exécute selon la sémantique du langage.

# Les trois piliers

- La spécification:
  ce qu'on veut
- Le programme:
  ce qu'on a
- La sémantique permet de faire le lien entre les deux: de prouver que ce qu'on a marche comme on veut!
  ➔ démonstration mathématique



# Induction (récurrence) mathématique

- Programmes récursifs ➔ preuve par induction
  - Un programme récursif est basé sur un ensemble ordonné, comme les entiers et les listes
  - On montre
    - D'abord l'exactitude du programme pour les cas de base
    - Ensuite, si le programme est correct pour un cas donné, alors il est correct pour le cas suivant
- Entiers
  - Cas de base : souvent *0* ou *1*
  - Pour un entier *n* le cas suivant est *n+1*
- Listes
  - Cas de base : nil ou une liste avec un ou plusieurs éléments
  - Pour une liste T le cas suivant est H|T

# Exemple: exactitude de la factorielle

- La spécification de {Fact N} (purement mathématique)

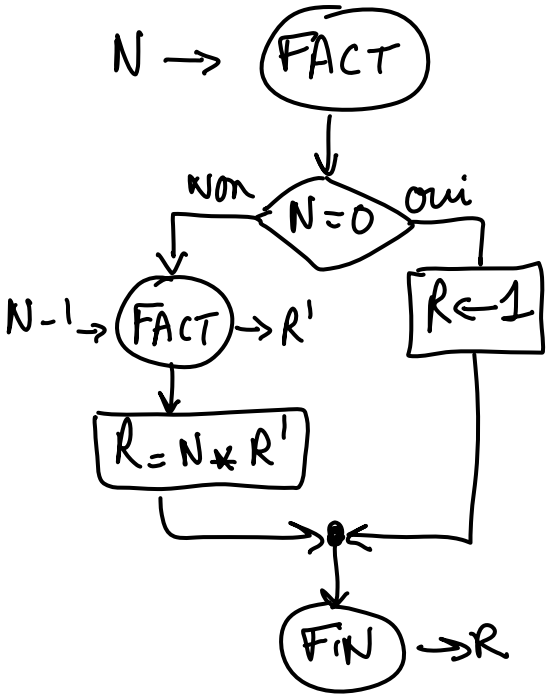
  $0! = 1$

  $n! = n * ((n-1)!)$ si $n>0$

- Le programme (en langage de programmation)

```
fun {Fact N}
   if N==0 then 1 else N*{Fact N-1}
end
end
```

- Où est la sémantique du langage?
  - D'abord, raisonnement intuitif



# Raisonnement pour la factorielle

- Il faut démontrer que l'exécution de {Fact N} donne *n!* pour tout *n*
- Cas de base: *n=0*
  - La spécification montre *0!=1*
  - L'exécution de {Fact 0} avec la sémantique montre {Fact 0}=*1*
- Cas inductif: *(n-1)* $\rightarrow$ *n*
  - L'exécution de {Fact N}, selon la sémantique, montre que {Fact N} = N*{Fact N-1}
  - Avec l'hypothèse de l'induction, on sait que {Fact N-1}=*(n-1)!*
  - Si la multiplication est exacte, on sait donc {Fact N}=N**((n-1)!)*
  - Selon la définition mathématique de la factorielle, on peut déduire {Fact N}=*n!*
- Pour finir la preuve, il faut la sémantique du langage!

# Machine abstraite

- Comment peut-on définir la sémantique d'un langage de programmation?
- La machine abstraite
  - Une construction mathématique qui modélise l'exécution ➔ sémantique opérationnelle
  - Assez générale pour presque tous les langages de programmation
- Avec la machine abstraite, on peut répondre à beaucoup de questions sur l'exécution
  - Prouver l'exactitude des programmes
  - Comprendre l'exécution des programmes compliqués
  - Calculer le temps d'exécution d'un programme

# Concepts de la machine abstraite

- Mémoire à affectation unique $\sigma = \{x_1=10, x_2, x_3=20\}$
  - Variables et leurs valeurs
- Environnement $E = \{X \rightarrow x, Y \rightarrow y\}$
  - Lien entre identificateurs et variables en mémoire
- Instruction sémantique $(\langle s \rangle, E)$
  - Une instruction avec son environnement
- Pile sémantique $ST = [(\langle s \rangle_1, E_1), \ldots, (\langle s \rangle_n, E_n)]$
  - Une pile d'instructions sémantiques
- Exécution $(ST_1, \sigma_1) \rightarrow (ST_2, \sigma_2) \rightarrow (ST_3, \sigma_3) \rightarrow \ldots$
  - Une séquence d'états d'exécution (pile + mémoire)

# L'exécution d'une instruction en langage noyau

- Pour exécuter une instruction, il faut:
  - Un environnement **E** pour faire le lien avec la mémoire
  - Une mémoire σ qui contient les variables et les valeurs
- Instruction sémantique **(<s>, E)**
  - Chaque instruction a son propre environnement
- Etat d'exécution **(*ST*, σ)**
  - Une pile **ST** d'instructions sémantiques avec une mémoire σ
  - La même mémoire σ est partagée par toutes les instructions

# Début de l'exécution

- Etat initial

  **([(<s>,∅)], ∅)**

  - Instruction **<s>** avec environnement vide ∅ (pas d'identificateurs libres): **(<s>,∅)**
  - Pile contient une instruction sémantique: **[(<s>,∅)]**
  - Mémoire vide: ∅ (pas encore de variables)
- A chaque pas
  - Enlevez l'instruction du sommet de la pile
  - Exécutez l'instruction
- Quand la pile est vide, l'exécution s'arrête

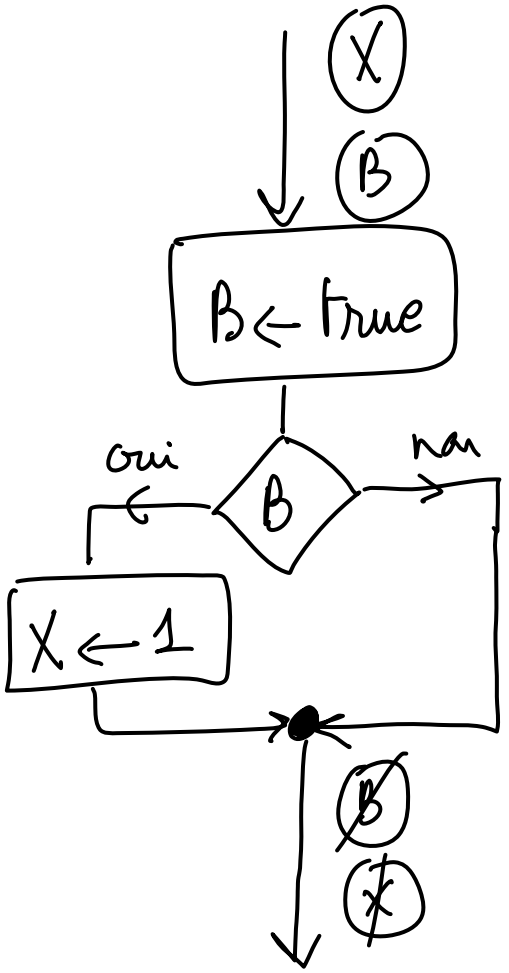
# [Exemple d’une exécution]

pptPlex Section Divider

The slides after this divider will be grouped into a section and given the label you type above.  Feel free to move this slide to any position in the deck.

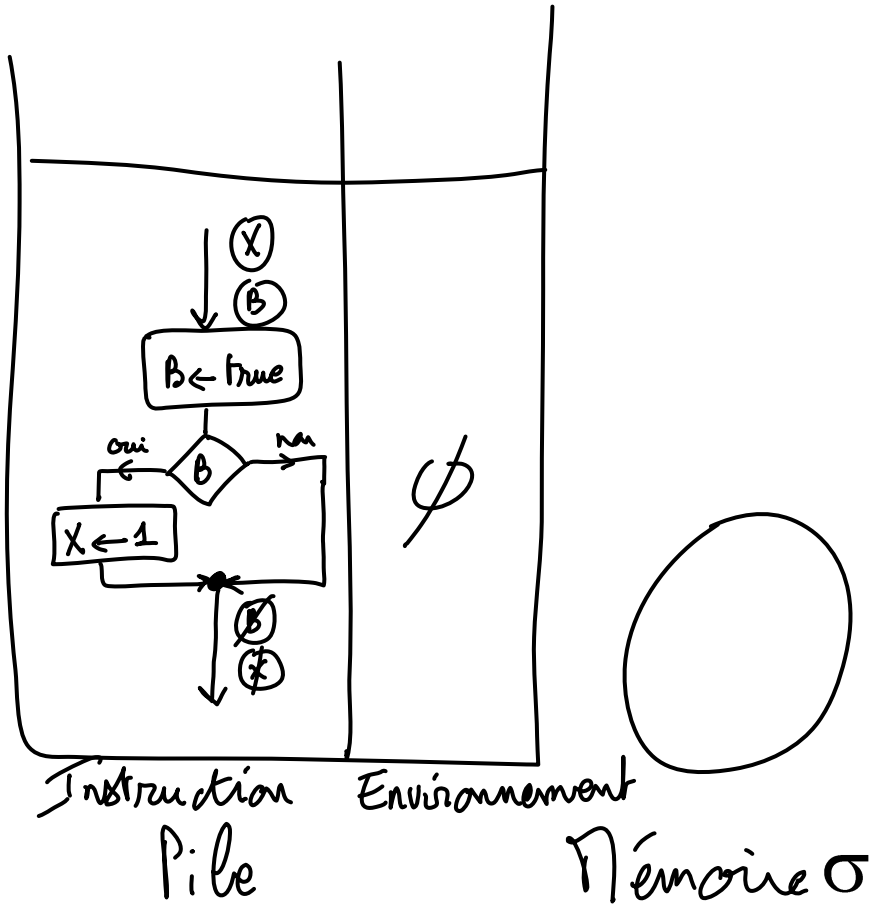
# L'instruction en langage noyau

```
local X in
    local B in
        B=true
        if B
        then X=1
        else skip
        end
    end
end
```



# Début de l'exécution: l'état initial

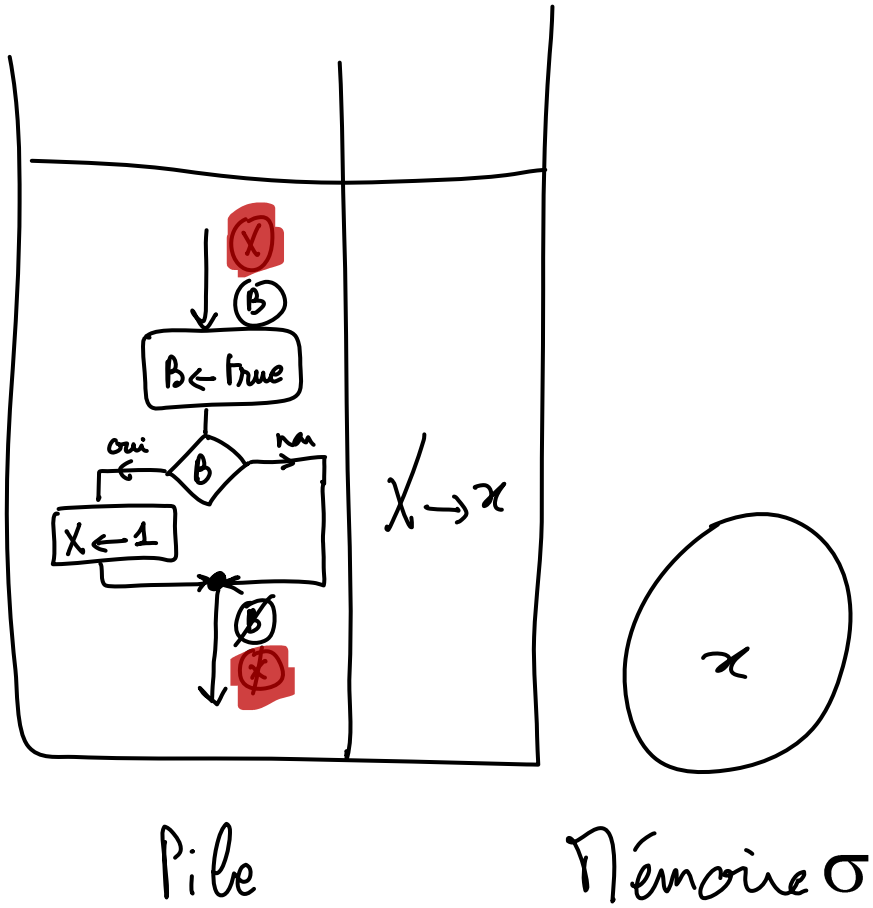
```
([(local X in
      local B in
         B=true
         if B
         then X=1
         else skip
         end
      end
   end, ∅)],
 ∅)
```




Commençons avec une mémoire vide et un environnement vide

# L'instruction "local"

```
([(local B in
      B=true
      if B
      then X=1
      else skip
      end
   end,
   {X → x})],
 {x})
```

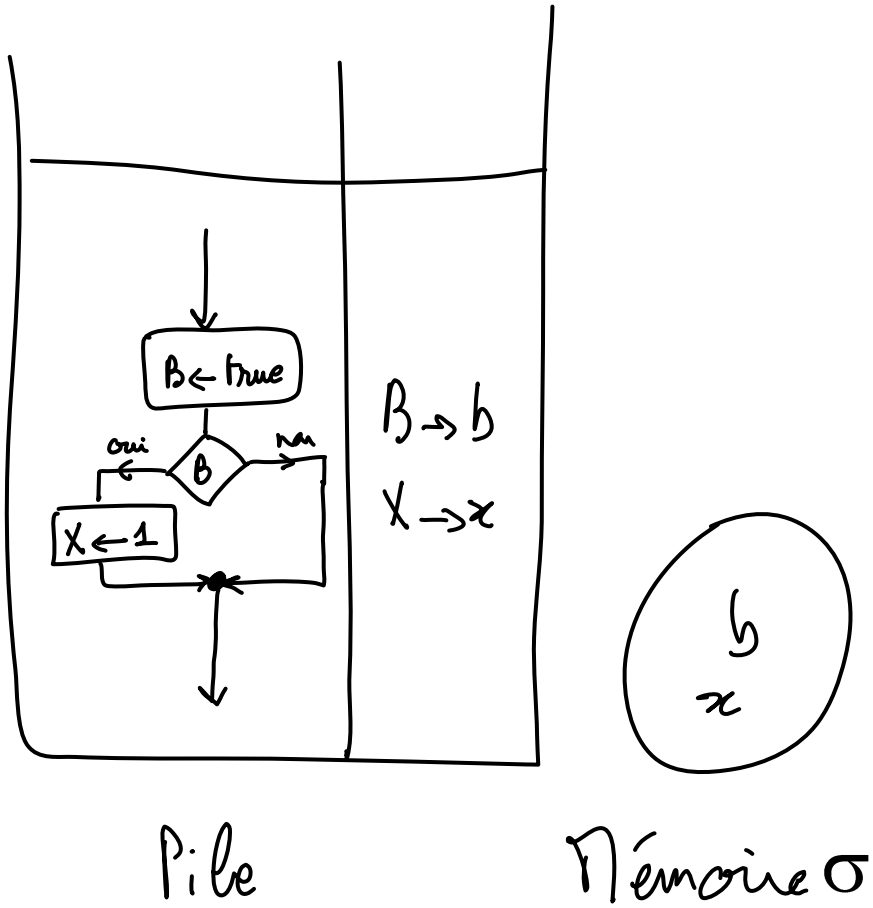



Créez la nouvelle variable *x* dans la mémoire

Continuez avec le nouvel environnement

# Encore un "local"

```
([( (B=true
     if B
     then X=1
     else skip
     end) ,
   {B → b, X → x})],
{b,x})
```

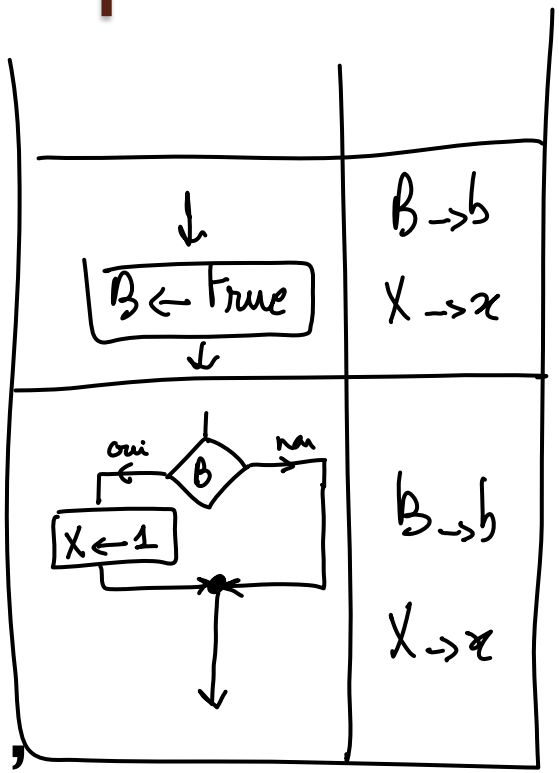



Créez la nouvelle variable *b* dans la mémoire

Continuez avec le nouvel environnement

# Composition séquentielle

```
([(B=true,
     {B → b, X → x}) ,
  (if B
   then X=1
   else skip
   end,
     {B → b, X → x})] ,
 {b,x})
```

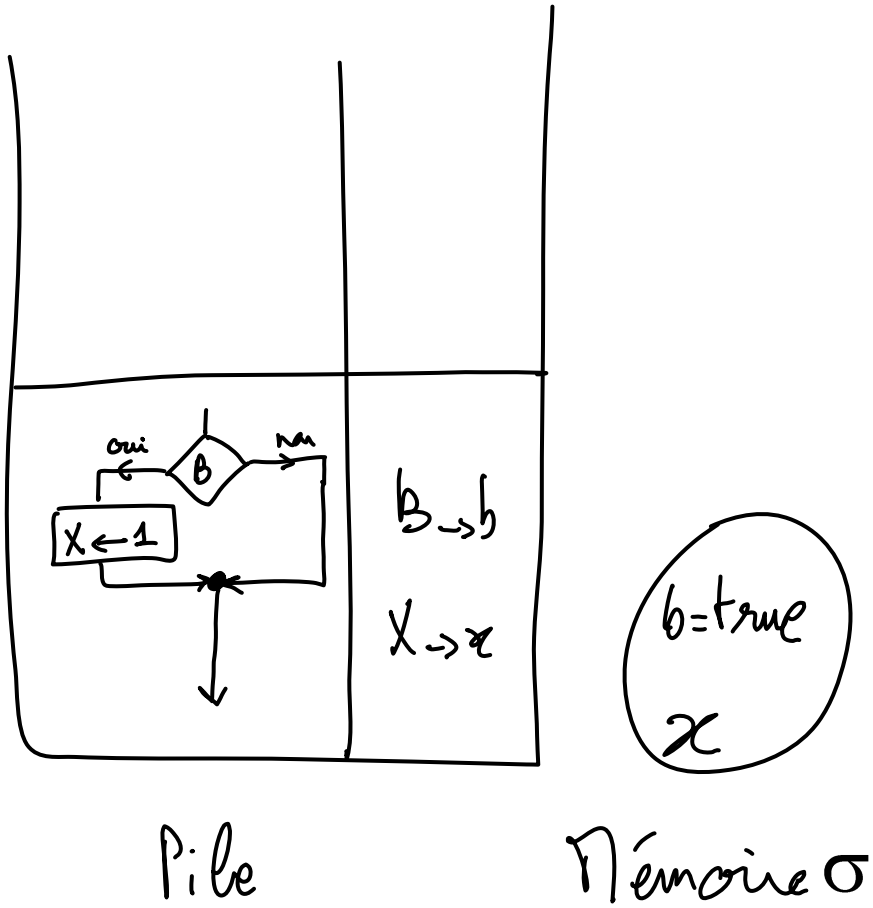



L'instruction composée devient deux instructions

La pile contient maintenant deux instructions sémantiques

# Affectation de B

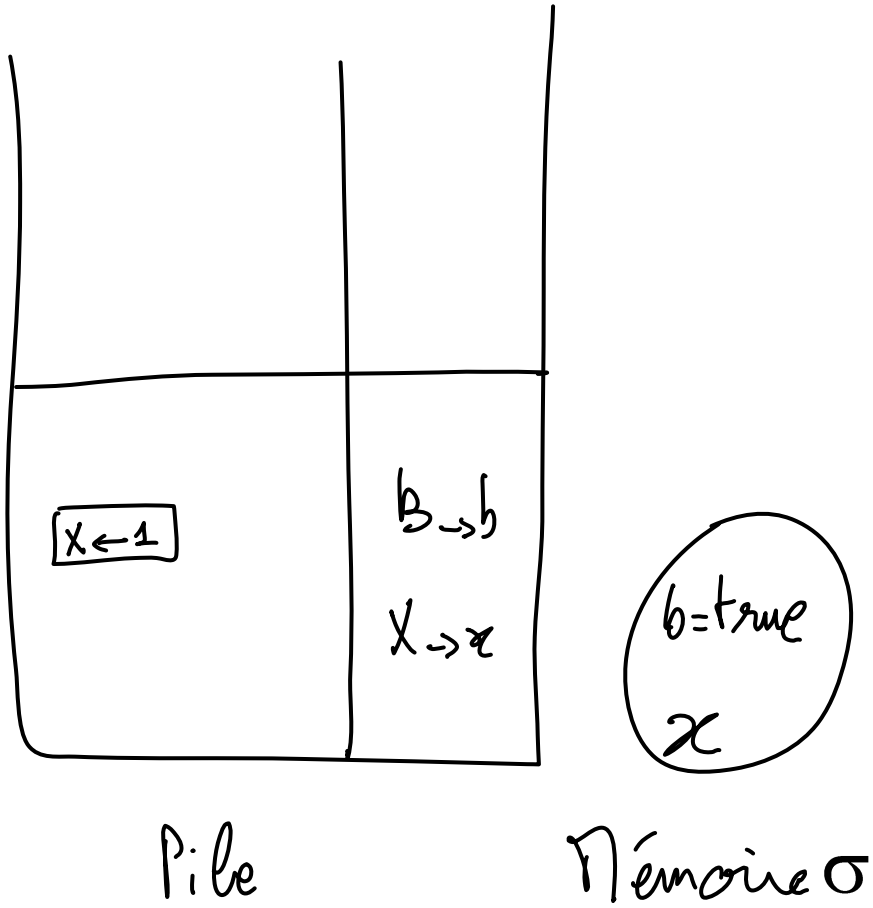
```
([(if B
   then X=1
   else skip
   end,
    {B → b, X → x})],
{b=true, x})
```




Affectez *b* à **true**

# Instruction "if"

$([(X=1,$
$\{B \to b, X \to x\})],$
$\{b=\texttt{true}, x\})$

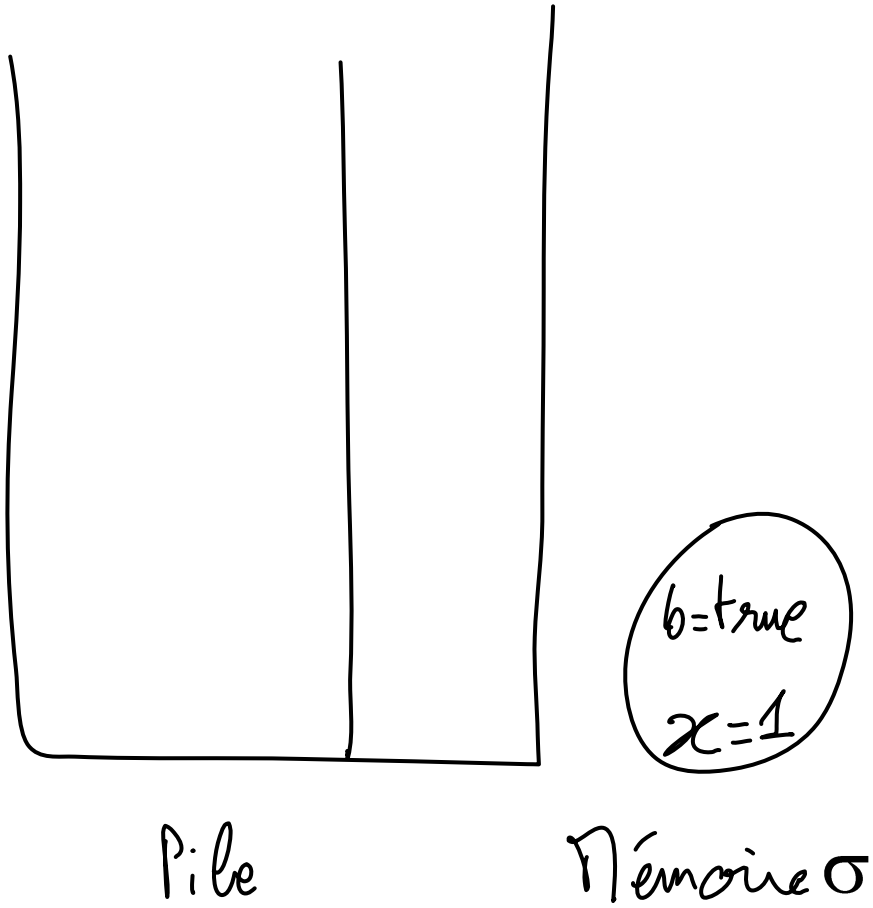



Testez la valeur de B

Continuez avec l'instruction après le **then**

# Affectation de X

([],
 {*b*=`true`, *x*=1})




Affectez *x* à 1

L'exécution termine parce que la pile est vide