Fermetures

Vincent Archambault-B
IFT 2035 - Université de Montréal

Pour obtenir le code source de ce document

- https://github.com/archambaultv/IFT2035-UdeM
- vincent.archambault-bouffard@umontreal.ca

Fonction d’ordre supérieur

- Une fonction peut recevoir une fonction en argument
- Une fonction peut retourner une fonction

Fonction d’ordre supérieur

```
map :: (a -> b) -> [a] -> [b]
map f [] = []
map f (x : xs) = f x : map f xs

plusOne :: [Int] -> [Int]
plusOne x = map (+ 1) x
```

Fonction d’ordre supérieur

```
foldl :: (b -> a -> b) -> b -> [a] -> b
foldl f acc [] = acc
foldl f acc (x : xs) = foldl f (f acc x) xs

sum :: [Int] -> Int
sum x = foldl (+) 0 x
```

Environnement de fermeture

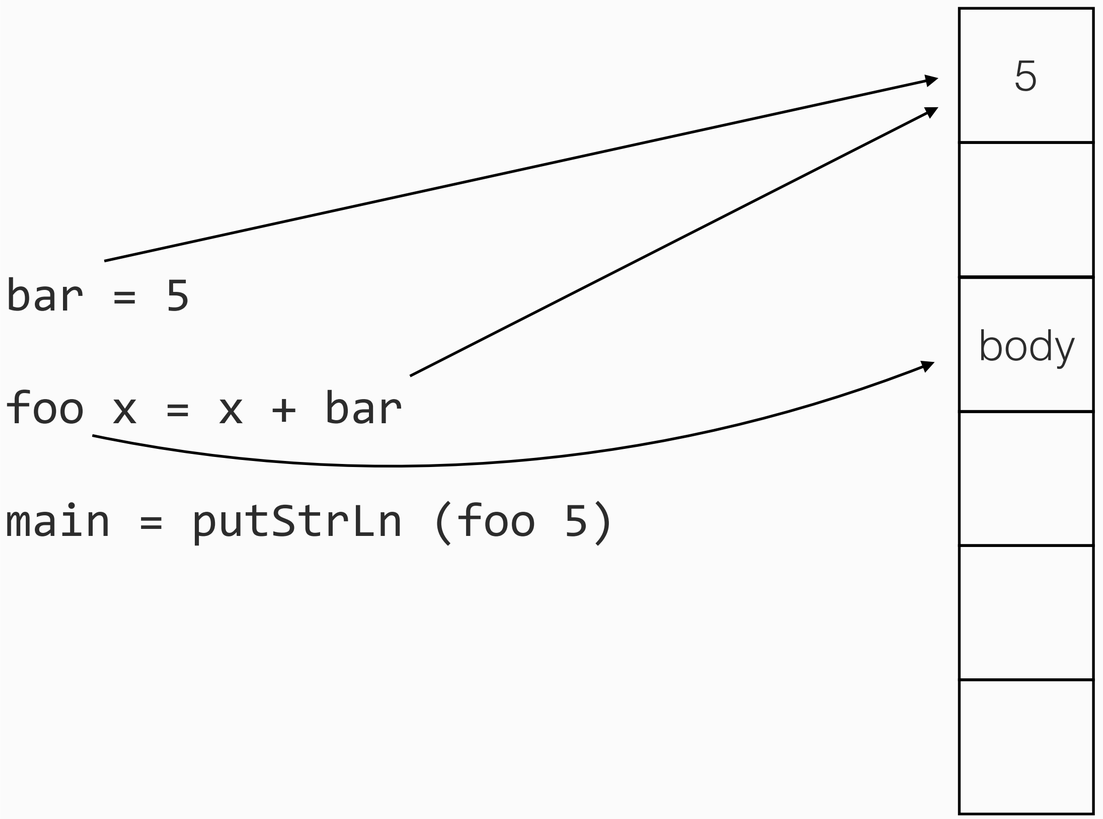
Comment foo se rappelle de la valeur de bar ?

```
bar = 5

foo x = x + bar

main = putStrLn (foo 5)
```

Option 1 : Se rappeler de l’adresse de bar

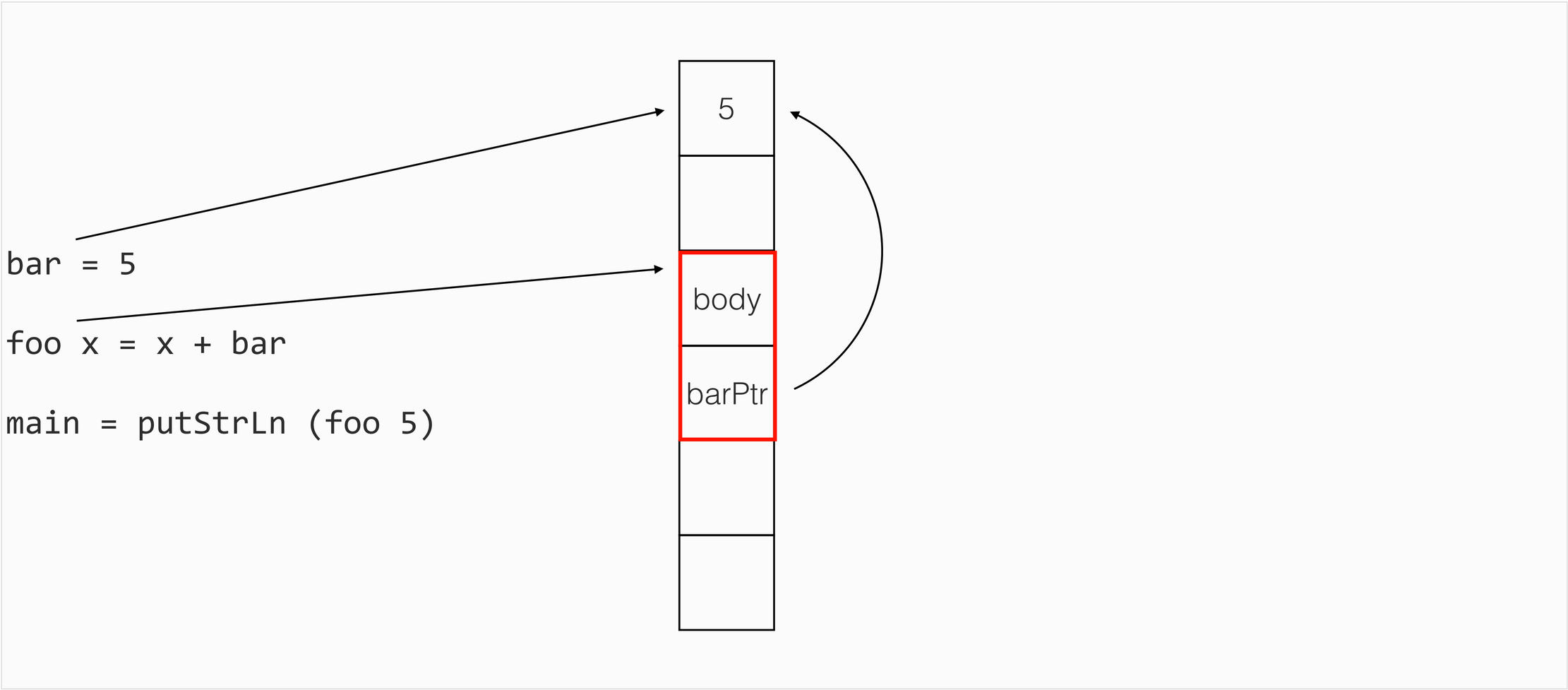


OK avec les identificateurs globaux (top level), car le compilateur connaît leur position mémoire en avance

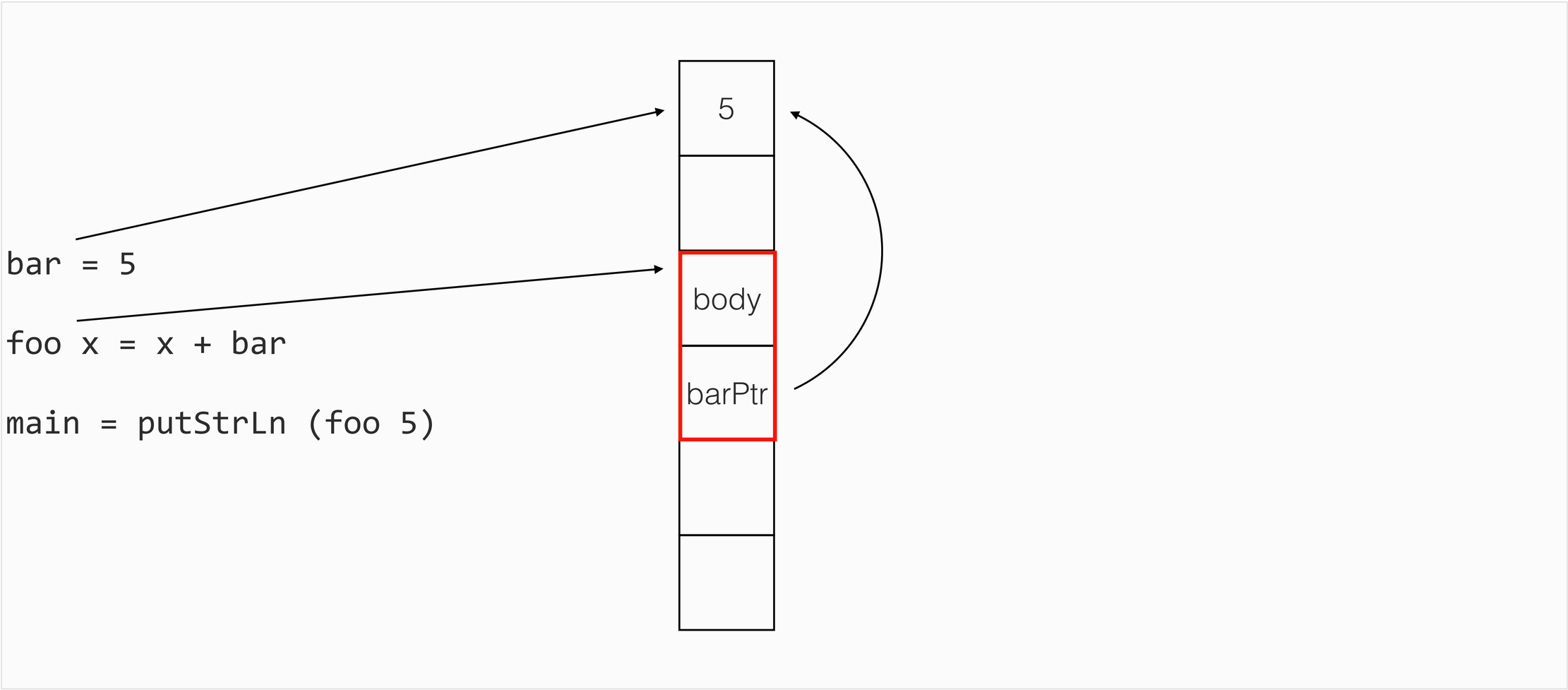
Variables libres

Une variable dans le corps de la fonction qui n’est pas un paramètre ou une définition locale.

Option 2 : Chaque fonction possède son propre environnement pour les variables libres



Option 2 : Les fonctions sont une structure de données

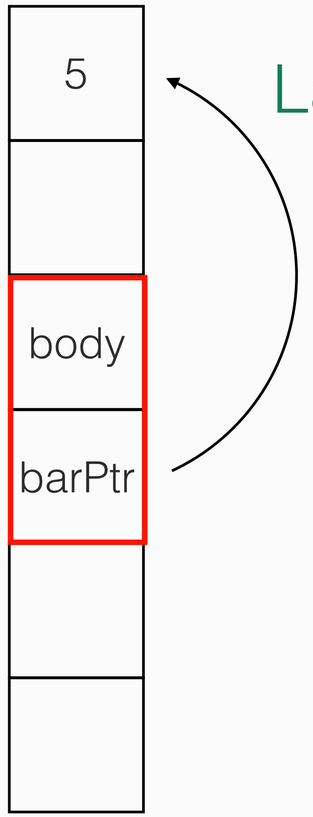


Option 2 : Fonctionne aussi pour les variables locales

```
foo x =
  let bar = 5
  in \x = x + bar

foo2 = foo 2

main = putStrLn (foo2)
```



La position mémoire de bar n'est pas connue à l'avance par le compilateur, car foo peut-être appelée n'importe quand.

Par contre, quand la fonction anonyme est créée la position de bar est connue

Fermeture = Fonction + Environnement

Fermeture = Nom des paramètres + Définition + Environnement

Exemple

Spécialiser une fonction

```haskell
salutation str = \x -> str ++ x

bonjour = salutation "Bonjour "

hello = salutation "Hello "

salut = salutation "Salut "

x = bonjour "Vincent"
y = hello "Vincent"
z = salut "Vincent"
```

Haskell

Exemple

Structure de données

```haskell
makePerson taille age =
  \field -> case field of
              "taille" -> taille
              "age" -> age

paul = makePerson 175 23

age = paul "age"
taille = paul "taille"
```

Haskell

Exemple

Structure de données

```scheme
(define (nouveau-compteur)
  (let ((x 0))
    (lambda () (begin (set! x (+ 1 x))
                      x))))

(define compteur1 (nouveau-compteur))
(define compteur2 (nouveau-compteur))

(display (compteur1)) ; 1
(display (compteur1)) ; 2
(display (compteur2)) ; 1
```

Scheme