Programmation fonctionnelle avec OCaml

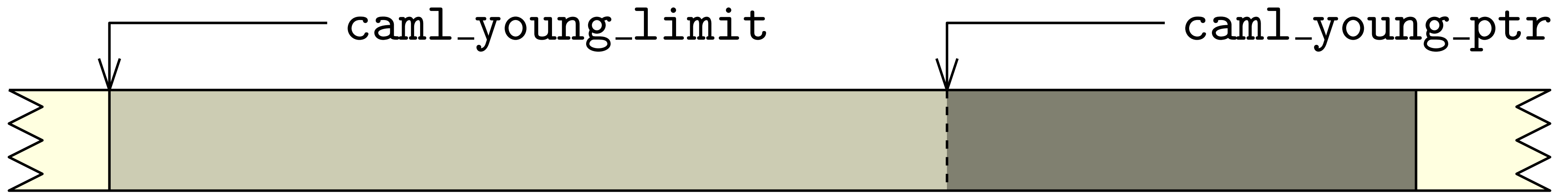
6ème et dernière séance, 30 avril 2014

Les entrailles d'OCaml

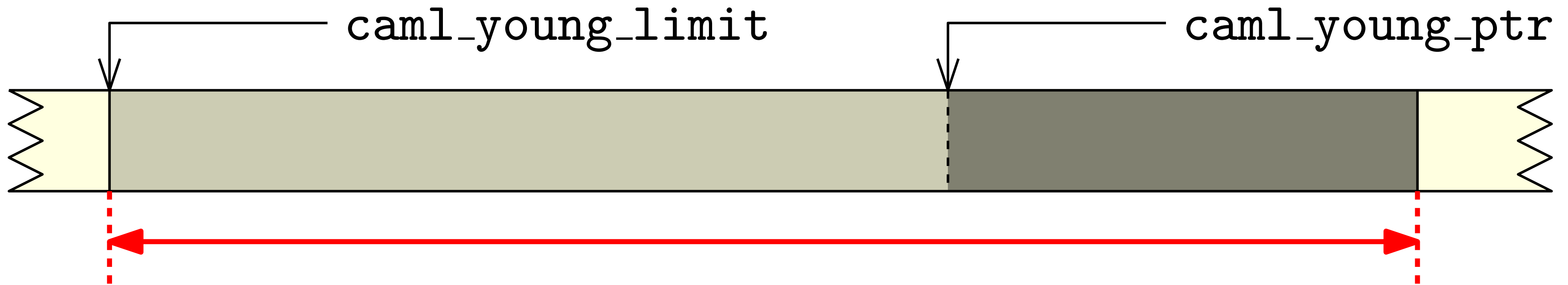
samuel.hornus@inria.fr

http://www.loria.fr/~shornus/ocaml/

Le petit tas – *The minor heap*

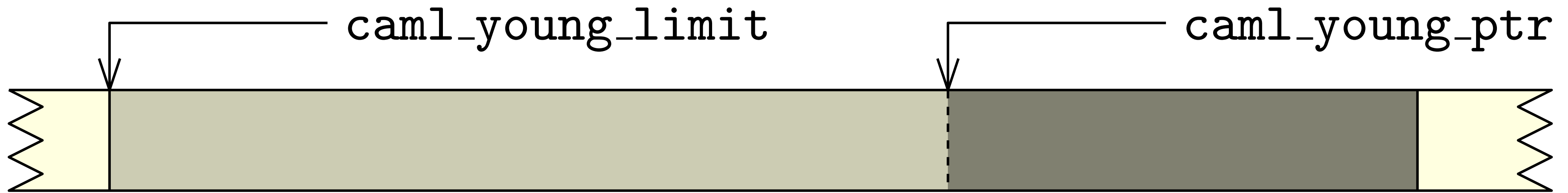


Le petit tas – *The minor heap*



C'est une zone continue de mémoire (typiquement 256 KiB)

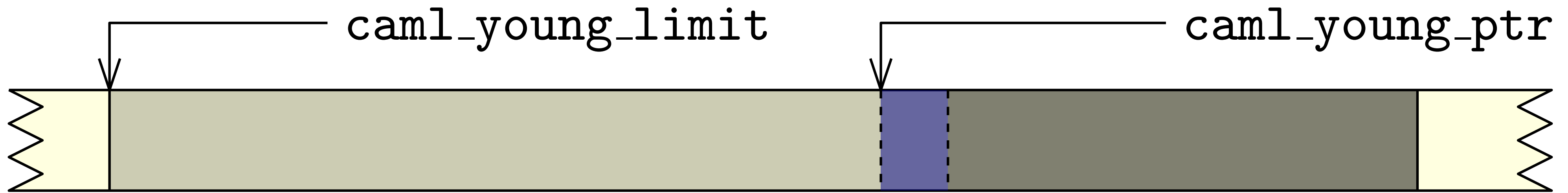
Le petit tas – *The minor heap*



C'est une zone continue de mémoire (typiquement 256 KiB)

comprenant une partie libre et une partie allouée.

Le petit tas – *The minor heap*

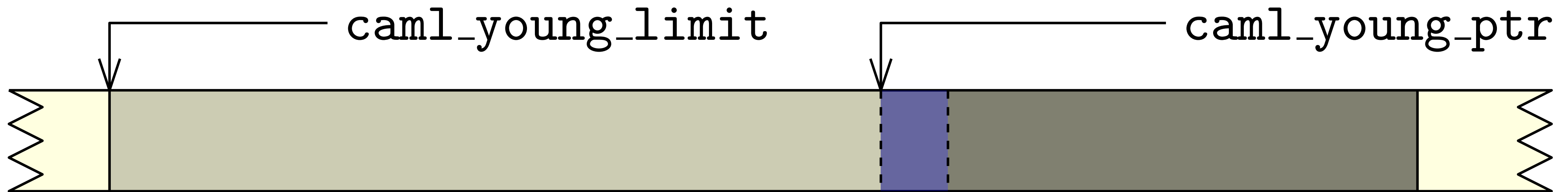


C'est une zone continue de mémoire (typiquement 256 KiB)

comprenant une partie libre et une partie allouée.

Pour allouer n mots : `<C>  caml_young_ptr -= 8*n  </C>`

Le petit tas – *The minor heap*



C'est une zone continue de mémoire (typiquement 256 KiB)

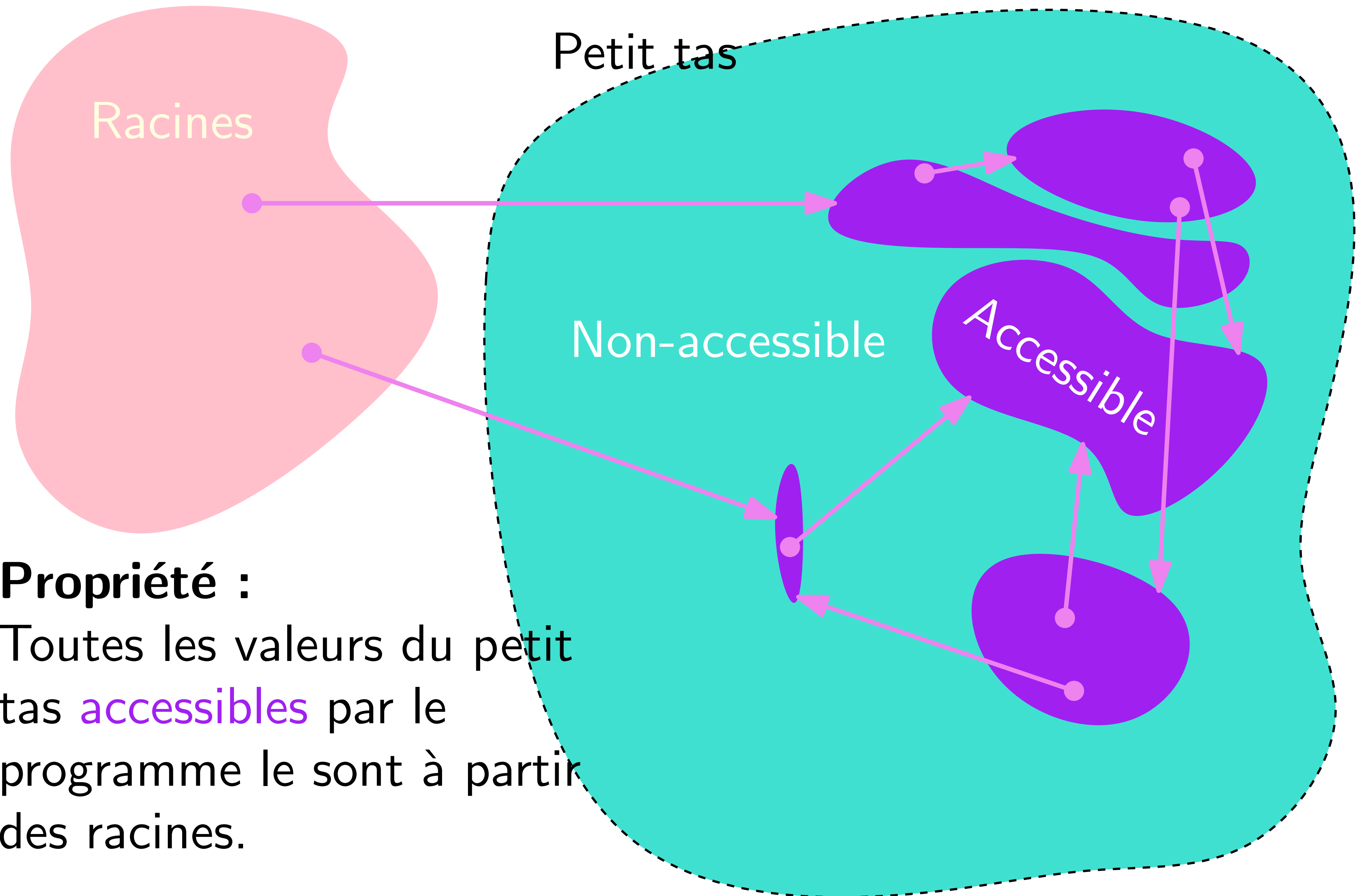
comprenant une partie libre et une partie allouée.

Pour allouer n mots : `<C> caml_young_ptr -= 8*n </C>`

Quand `caml_young_ptr` atteint `caml_young_limit`, le ramasse-miette effectue un « petit balayage » *(minor collection)*.

Le petit balayage – *The minor collection*

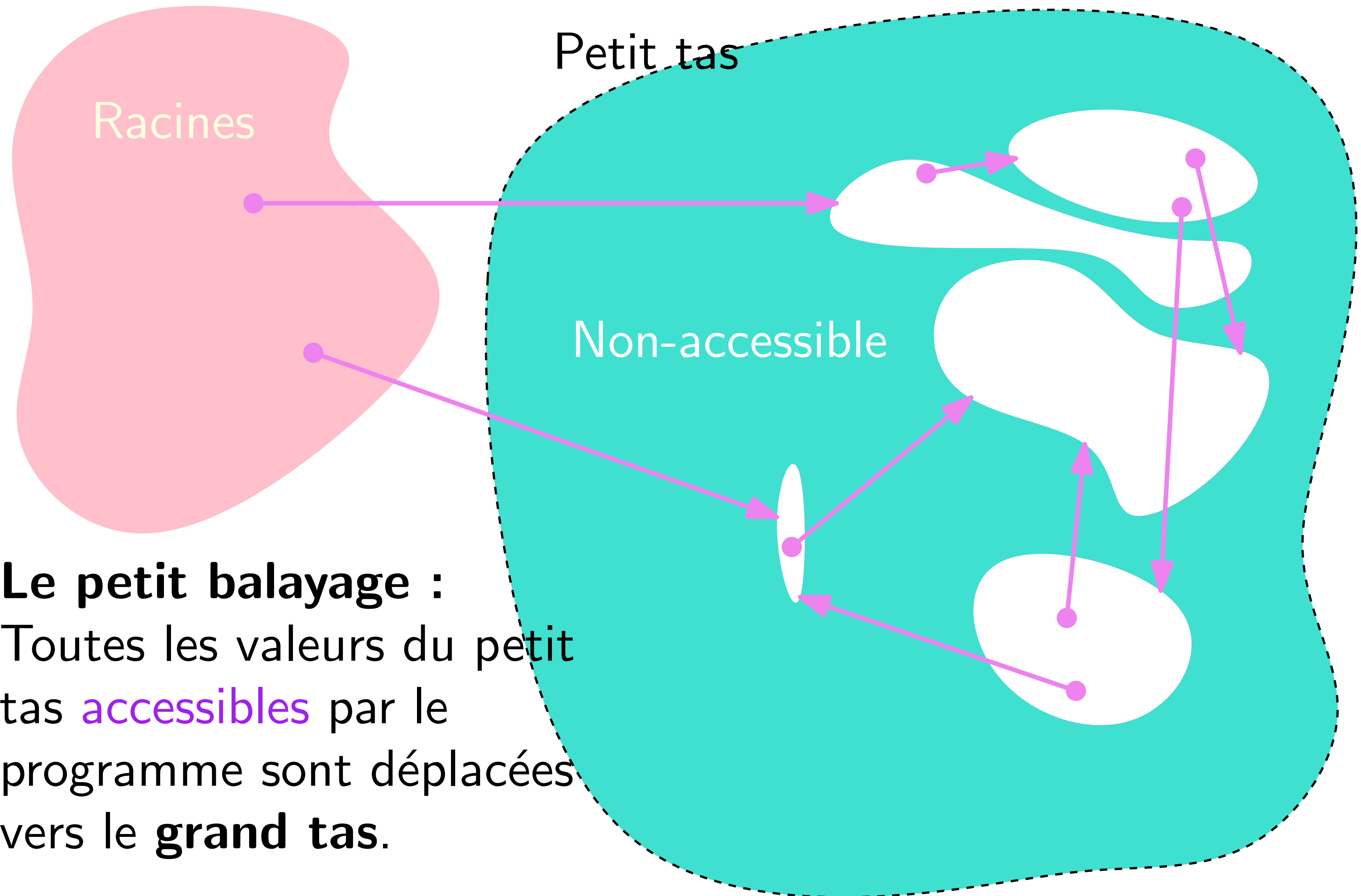
Les *racines* sont des pointeurs vers le petit tas.



Propriété :
Toutes les valeurs du petit tas accessibles par le programme le sont à partir des racines.

Le petit balayage – *The minor collection*

Les *racines* sont des pointeurs vers le petit tas.



Le petit balayage – *The minor collection*

Les *racines* sont des pointeurs vers le petit tas.

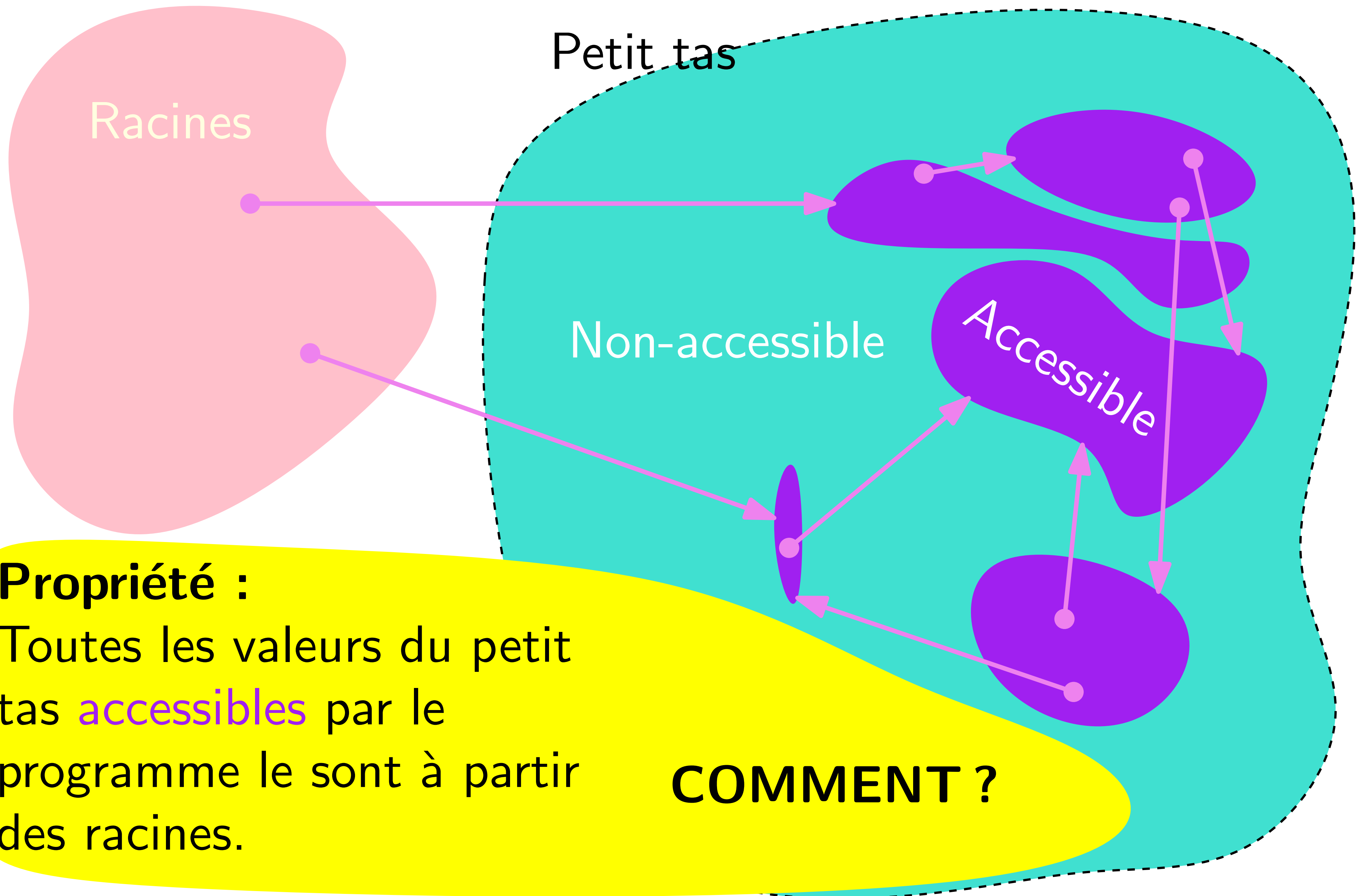
Petit tas

```
<C>
caml_young_ptr = caml_young_end
</C>
```

Le petit balayage :
Puis le petit tas est vidé, et les racines sont mises à jour.

Le petit balayage – *The minor collection*

Les *racines* sont des pointeurs vers le petit tas.



Représentation d'une valeur OCaml

Toute **valeur** OCaml occupe un mot mémoire (64 bits) :

Si son bit de poids faible est 1, alors c'est un `int` ou un `char` (ou...)

Si 0, c'est un pointeur

- vers un **bloc** dans le petit ou le grand tas
- vers l'extérieur des tas (*eg*, alloué par une librairie C)

Représentation d'une valeur OCaml

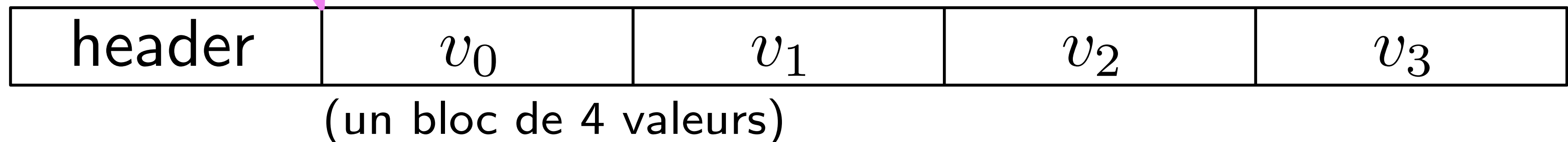
Toute **valeur** OCaml occupe un mot mémoire (64 bits) :

Si son bit de poids faible est 1, alors c'est un `int` ou un `char` (ou...)

Si 0, c'est un pointeur

- vers un **bloc** dans le petit ou le grand tas
- vers l'extérieur des tas (*eg*, alloué par une librairie C)

Un bloc est composé d'un *header* de un mot, suivi de 1 ou plusieurs mots.



(un bloc de 4 valeurs)

Chaque mot du bloc est une valeur OCaml.

Représentation d'une valeur OCaml

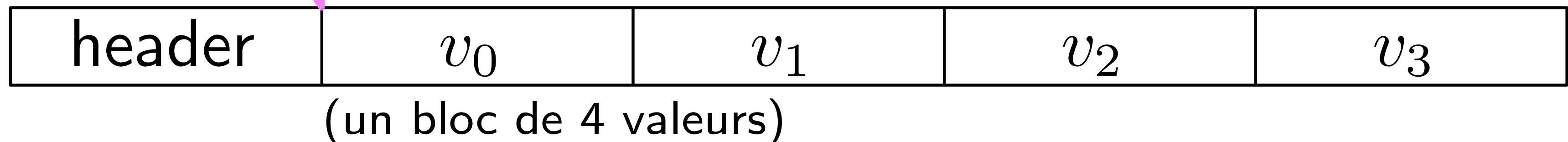
Toute **valeur** OCaml occupe un mot mémoire (64 bits) :

Si son bit de poids faible est 1, alors c'est un `int` ou un `char` (ou...)

Si 0, c'est un pointeur

- vers un **bloc** dans le petit ou le grand tas
- vers l'extérieur des tas (*eg*, alloué par une librairie C)

Un bloc est composé d'un *header* de un mot, suivi de 1 ou plusieurs mots.



(un bloc de 4 valeurs)

Chaque mot du bloc est une valeur OCaml.

Le ramasse-miette peut ainsi parcourir automatiquement l'ensemble des valeurs accessibles.

Les entiers

Un entier n est donc représenté par l'entier $2n+1$.

```
let i = 42
```

Le mot mémoire w_i représentant i contient la représentation binaire de 85 : $w_i = 85$.

Les entiers

Un entier n est donc représenté par l'entier $2n + 1$.

```
let i = 42
```

Le mot mémoire w_i représentant i contient la représentation binaire de 85 : $w_i = 85$.

Par exemple, l'addition de deux entiers i et j se fait par le code assembleur $w_i + w_j - 1$. (`lea -1(%eax, %ebx), %eax` .)

En effet,
$$
\begin{aligned}
w_i + w_j - 1 &= (2i + 1) + (2j + 1) - 1 \\
&= 2(i + j) + 1 \\
&= w_{i+j}
\end{aligned}
$$

Quelques valeurs représentées par n entier

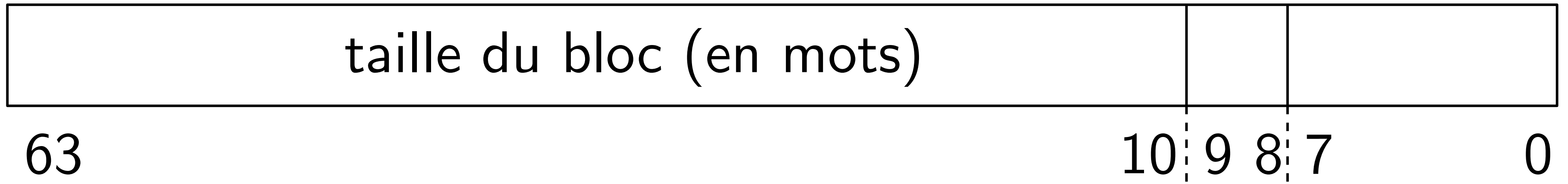
- Les `int` : `n` $\Rightarrow$ 2n+1
- Les `char` : idem
- `()` $\Rightarrow$ 0 $(w_{\texttt{()}} = 1)$
- `false` $\Rightarrow$ 0 $(w_{\texttt{false}} = 1)$
- `true` $\Rightarrow$ 1 $(w_{\texttt{true}} = 3)$
- `[]` $\Rightarrow$ 0 $(w_{\texttt{[]}} = 1)$
- Les constructeurs constants d'un type somme :

```
type truc = Titi | Toto | Tata
```

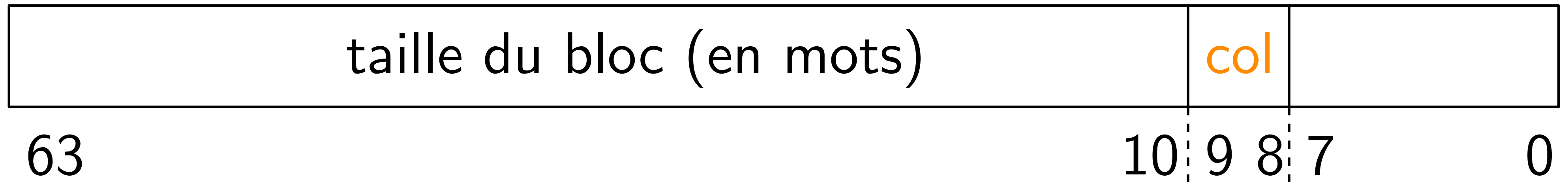
	Titi	Toto	Tata
	0	1	2
	1	3	5

Le header d'un bloc

Le header d'un bloc

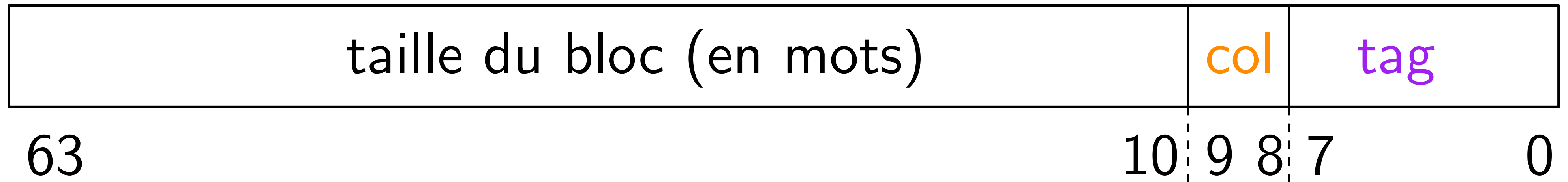


Le header d'un bloc



col = color. Deux bits utilisés par les algorithmes du ramasse-miette.

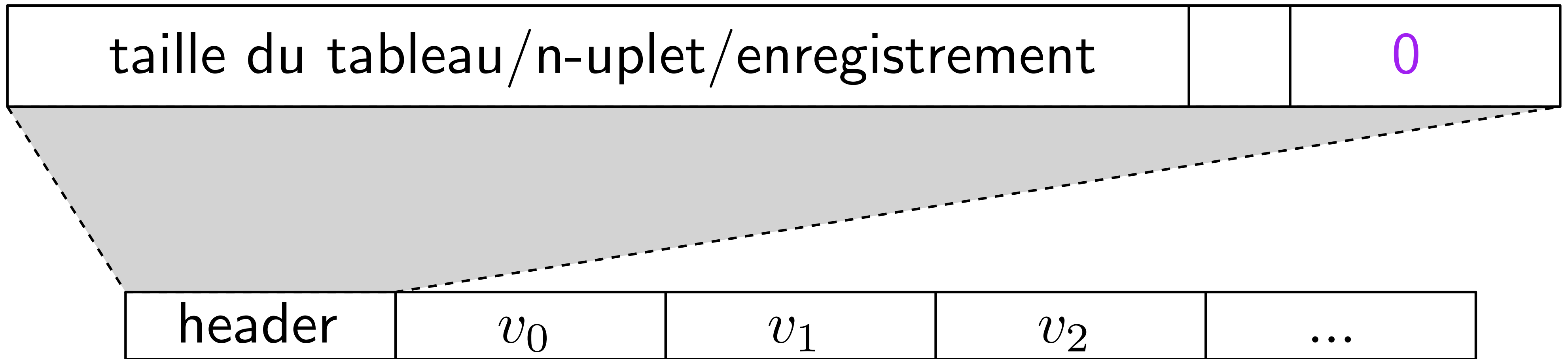
Le header d'un bloc



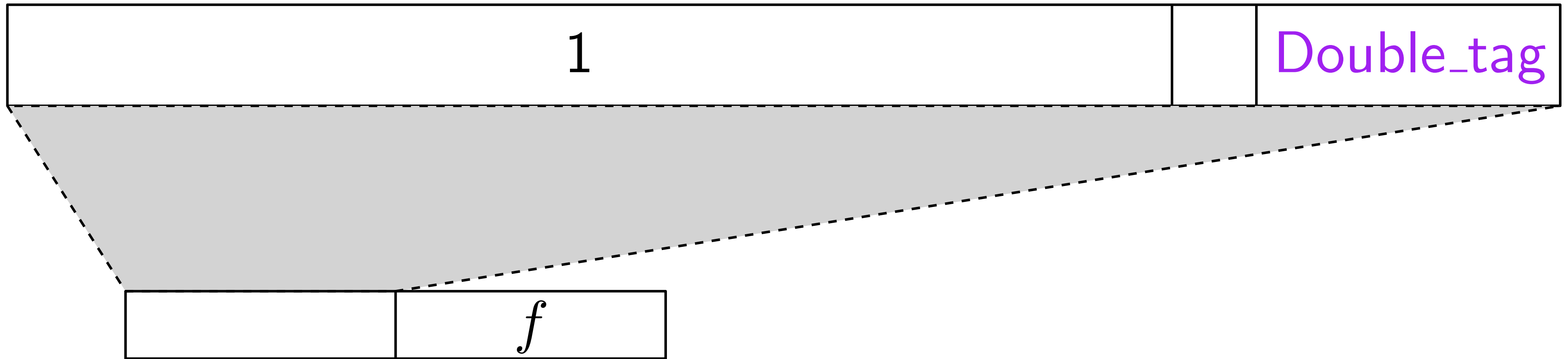
col = color. Deux bits utilisés par les algorithmes du ramasse-miette.

tag : multi-usage, selon le type de donnée que représente le bloc.

'a array, tuples, records



`float`



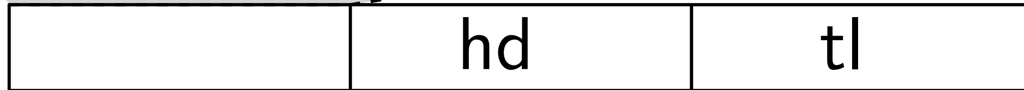
string

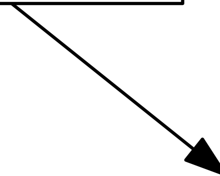
taille en mots (de 64 bits)		String_tag

	8 chars	8 chars	...	padding

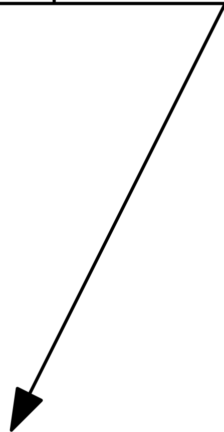
						ω	0x00
					ω	0x00	0x01
				ω	0x00	0x00	0x02
			ω	0x00	0x00	0x00	0x03
		ω	0x00	0x00	0x00	0x00	0x04
	ω	0x00	0x00	0x00	0x00	0x00	0x05
ω	0x00	0x00	0x00	0x00	0x00	0x00	0x06
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x07

`'a list`





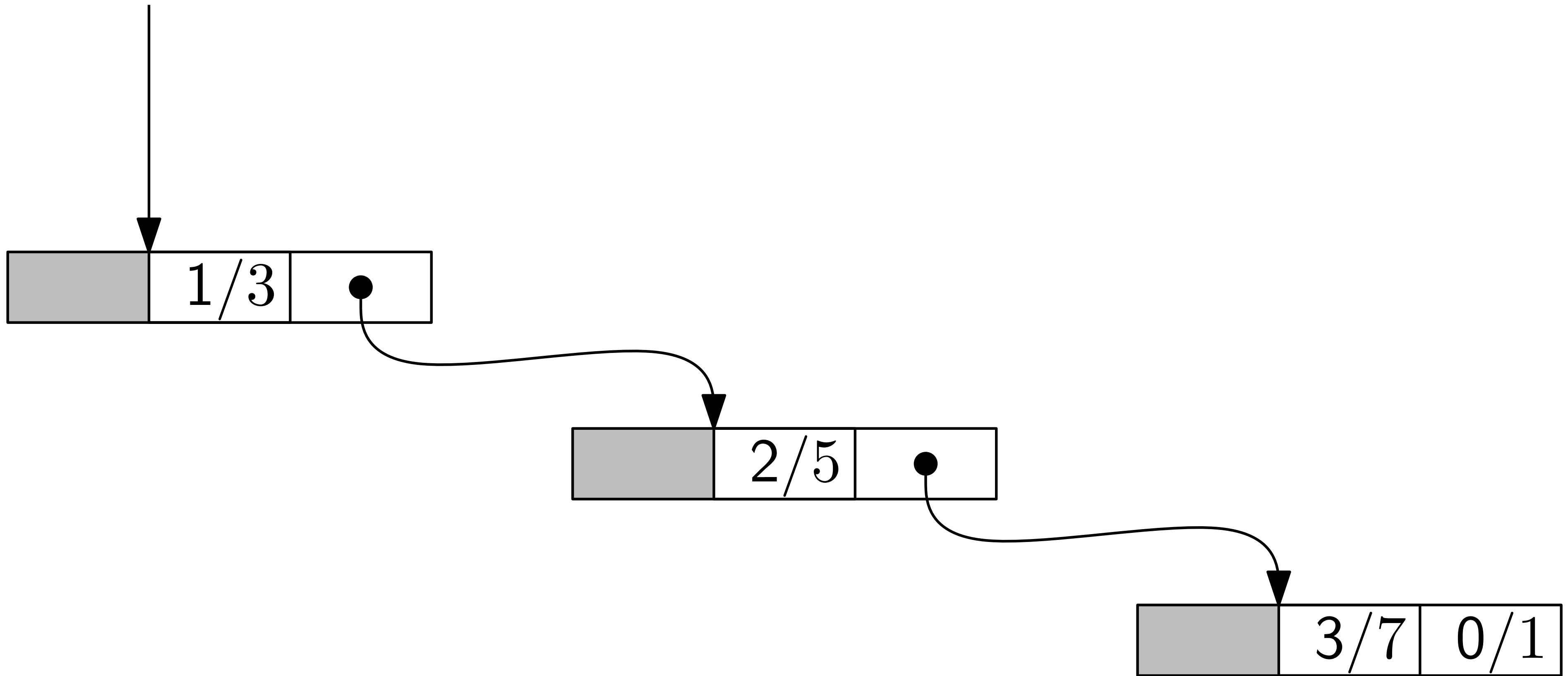
Pointeur vers le
maillon suivant,
ou [] (==0)



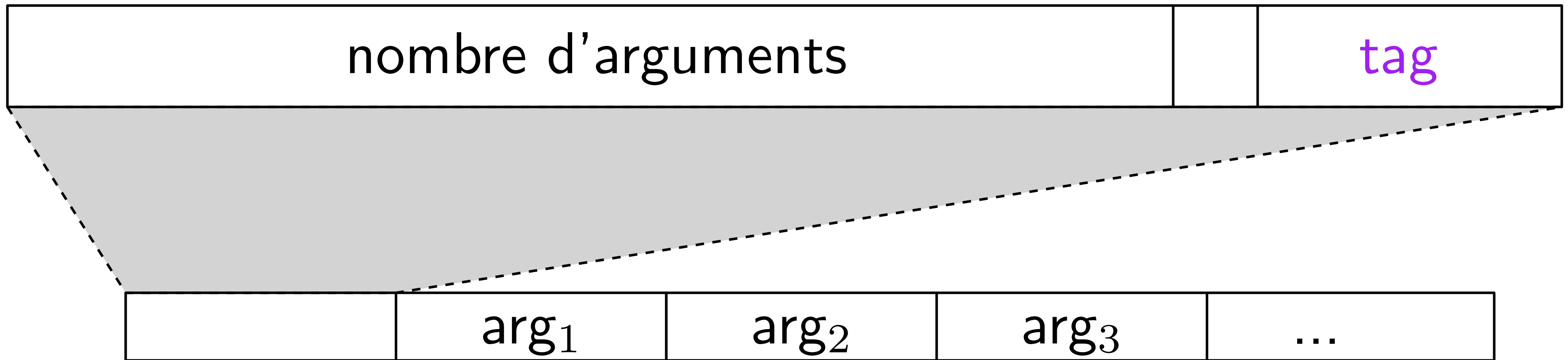
Premier élément de la liste

Liste, exemple

```
let a = [1;2;3]
```



Constructeurs d'un type somme avec argument(s)

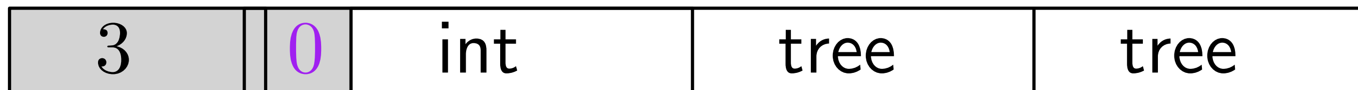


Exemple :

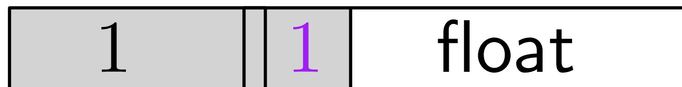
```
type tree =
| Empty
```

$0/1$

```
| Node of int * tree * tree
```




```
| Leaf of float
```

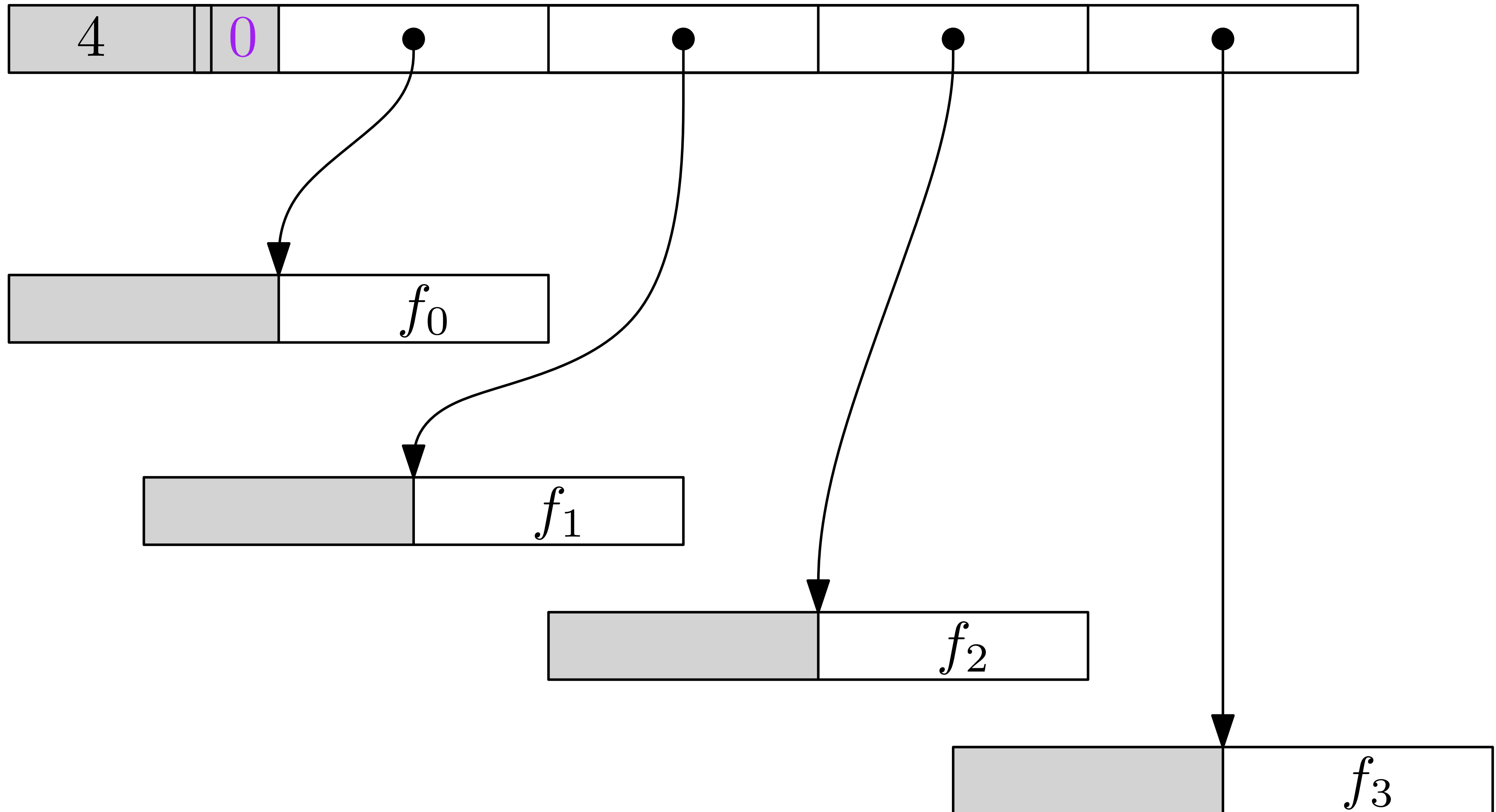


Optimisation d'un cas spécifique

Un enregistrement ne contenant que des `float` ou un `float array` stocke **directement** les `float`s, au lieu de stocker des pointeurs vers des blocs Double_tag.

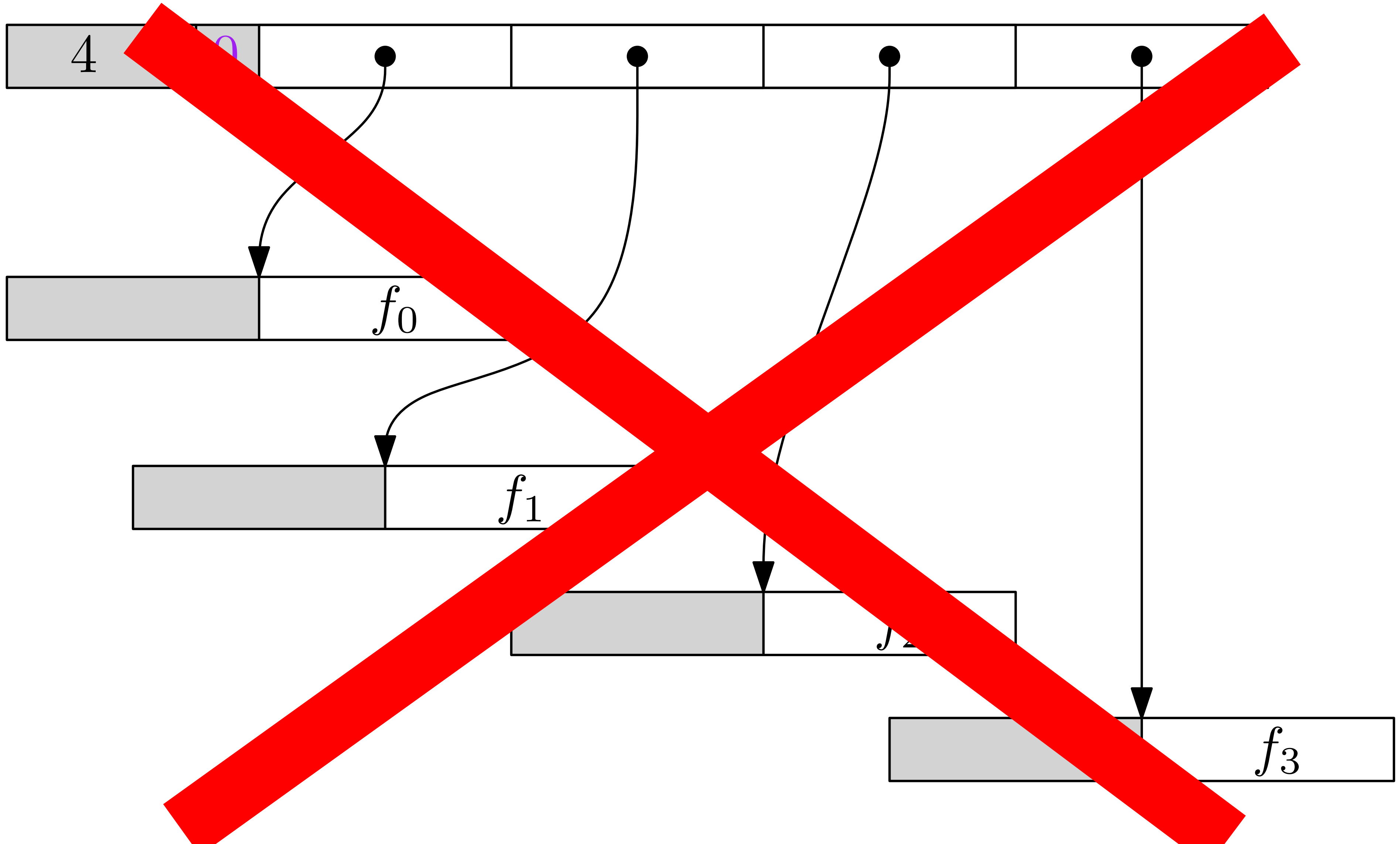
Optimisation d'un cas spécifique

Un enregistrement ne contenant que des `float` ou un `float array` stocke **directement** les `float`s, au lieu de stocker des pointeurs vers des blocs Double_tag.



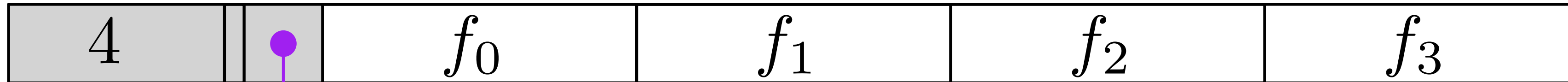
Optimisation d'un cas spécifique

Un enregistrement ne contenant que des `float` ou un `float array` stocke **directement** les `float`s, au lieu de stocker des pointeurs vers des blocs Double_tag.



Optimisation d'un cas spécifique

Un enregistrement ne contenant que des `float` ou un `float array` stocke **directement** les `float`s, au lieu de stocker des pointeurs vers des blocs Double_tag.



Double_array_tag

Autres types de bloc

`Closure_tag`

`Abstract_tag`

`Custom_tag`

Appeller du code C depuis OCaml

Dans l'interface `.mli` :

`val` *nom* `:` *type*   comme d'habitude

Dans la structure `.ml` :

`external` *nom* `:` *type* `= "`*C_function_name*`"`

Pour l'édition de liens :

```
ocaml[c|opt] ... code.o
```

Appeller du code C depuis OCaml

Avec `ocamlbuild`, dans un fichier `myocamlbuild.ml` :

```
open Ocamlbuild_plugin;;
dispatch (function
    | After_rules ->
      pdep ["link"] "linkdep" (fun param -> [param])
    | _ -> ())
```

dans un fichier `_tags` :

```
<*.byte>: linkdep(toto_c.o), custom
<*.native>: linkdep(toto_c.o)
```

Écrire un stub en C

```
#include <caml/mlvalues.h>
CAMLprim value toto(value p1, value p2, value p3)
{
  CAMLparam3(p1,p2,p3);
  const int i = Int_val(p1);
  const double d = Double_val(p2);
  double *out = (double*)p3;
  ...
  CAMLreturn Val_true;
}
```

http ://caml.inria.fr/pub/docs/manual-ocaml/manual033.html