Cours 7
Allocation mémoire

Jean-Jacques.Levy@inria.fr

http://jeanjacqueslevy.net

secrétariat de l'enseignement:

Catherine Bensoussan

cb@lix.polytechnique.fr

Laboratoire d'Informatique de l'X

Aile 00, LIX

tel: 34 67

http://w3.edu.polytechnique.fr/informatique

Références

- Modern Compiler Implementation in ML, Andrew Appel,
  http://www.cs.princeton.edu/~appel/modern/java/
- Programming Languages, Concepts and Constructs, Ravi Sethi, 2nd edition, 1997.
  http://cm.bell-labs.com/who/ravi/teddy/
- Theories of Programming Languages, John C. Reynolds, Cambridge University Press, 1998.
  http://www.cs.cmu.edu/afs/cs.cmu.edu/user/jcr/www/
- Uniprocessor Garbage Collection Techniques, Paul R. Wilson, IWMM, 1992.

Plan

1. **Le mutateur et ses racines**
2. **Le collecteur (*Garbage Collector*)**
3. **Compteurs de références**
4. **Marquer et récupérer (*Mark and Sweep*)**
5. **Méthode par recopie (*Stop and Copy*)**
6. **Méthode incrémentale**
7. **Générations**

Allocation mémoire explicite — implicite

- Allocation mémoire fine = problème complexe
- Gros programmes font parfois leur propre allocation mémoire
- $\Rightarrow$ Systèmes incomplets, erreurs complexes

Mieux vaut un système automatique de gestion de la mémoire.
Les causes d'erreurs sont plus localisées.

Types $\Rightarrow$ Récupération automatique de la mémoire

$$\equiv$$

Garbage collection

Durée de vie des valeurs

En PCF, deux sortes de valeurs.

- Les valeurs locales à chaque appel récursif de l'interpréteur utiles pour calculer la valeur courante.

  Nos interpréteurs sont récursifs. Si écrits dans un langage plus proche de la machine (C, assembleur), ils deviennent itératifs, la récursion étant remplacée par la gestion d'une pile. Les variables contenant ces valeurs temporaires sont souvent appelées variables de la pile.

  $$\beta_\ell \quad \langle \texttt{let}\ x = V\ \texttt{in}\ N,\ s\rangle \rightarrow \langle N[x\backslash V],\ s\rangle$$

- Les valeurs survivant à l'exécution du programme. Allouées dans la mémoire (via des locations ℓ), ou encore appelées variables du tas.

$$\textbf{alloc} \quad \langle \texttt{ref}\ V,\ s\rangle \rightarrow \langle \ell,\ s + [\ell = V]\rangle \qquad (\ell \notin \mathit{domain}(s))$$

$$\textbf{alloc}_t \quad \langle [|V_1; V_2; \ldots V_n|],\ s\rangle \rightarrow \langle \ell,\ s + [\ell = [|V_1; V_2; \ldots V_n|]]\rangle \qquad (\ell \notin \mathit{domain}(s))$$

Allocation en mémoire

Les variables de la pile sont allouées et libérées avec les appels récursifs de l'interpréteur.

Les variables du tas ne sont jamais libérées. Il peut ne plus y avoir de place disponible pour allouer de nouvelles valeurs, alors que beaucoup d'entre elles sont devenues inutiles.

Le *garbage collector* **(ou plus simplement GC) récupère l'espace disponible dans le tas.**

Disposition mémoire des valeurs

En PCF, les types indiquent si la valeur est contenue dans le tas. Un peu plus compliqué avec le polymorphisme, mais on y arrive. (disposition statique)

Souvent dans les langages (même typés), on *tague* les valeurs pour savoir si c'est un scalaire, ou une référence vers une valeur figurant dans le tas. (Un bit pour indiquer si pointeur, ou entier). (disposition dynamique)

Allocation en mémoire en PCF

Valeurs

V, V'	$::=$	$\underline{n}$	constante entière
	$\mid$	$\lambda x.M$	abstraction
	$\mid$	ℓ	location
	$\mid$	$()$	valeur vide

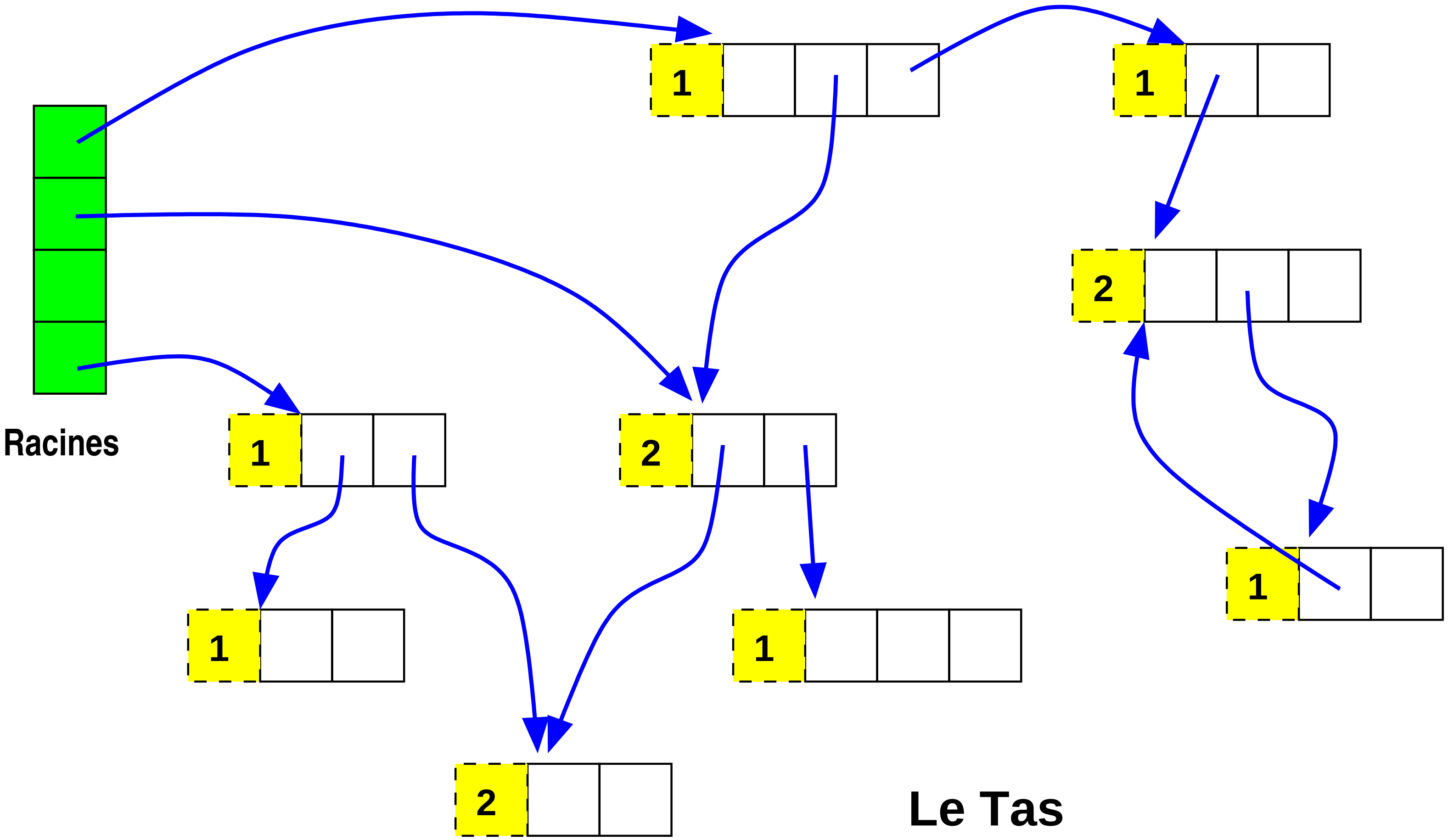
Un tableau `tas` représente le tas, une location ℓ est un indice (tagué) dans `tas`. Si on a aussi les listes ou les tableaux, il faut mettre une indication de taille dans le premier mot réservé pour une cellule. La liste des blocs libres (*free-list*) permet de retrouver les blocs libres, si on ne fonctionne pas par recopie.

Pour les fermetures $(\lambda x.M)[\rho]$, deux stratégies possibles:

- leur partie environnement est considérée comme une liste de PCF allouée dans le tas.
- on fait un traitement particulier sur l'environnement, et il faudra ne pas oublier de le parcourir quand on trace les éléments accessibles à partir d'une valeur.

Compteurs de références

Chaque cellule mémoire du tas a un compteur de références.



Compteurs de références – bis

- le compteur de référence est mis à jour à chaque affectation.
- Si le compteur d'une cellule devient zéro. On libère la cellule, et on met à jour les compteurs des cellules pointées par la cellule libérée.

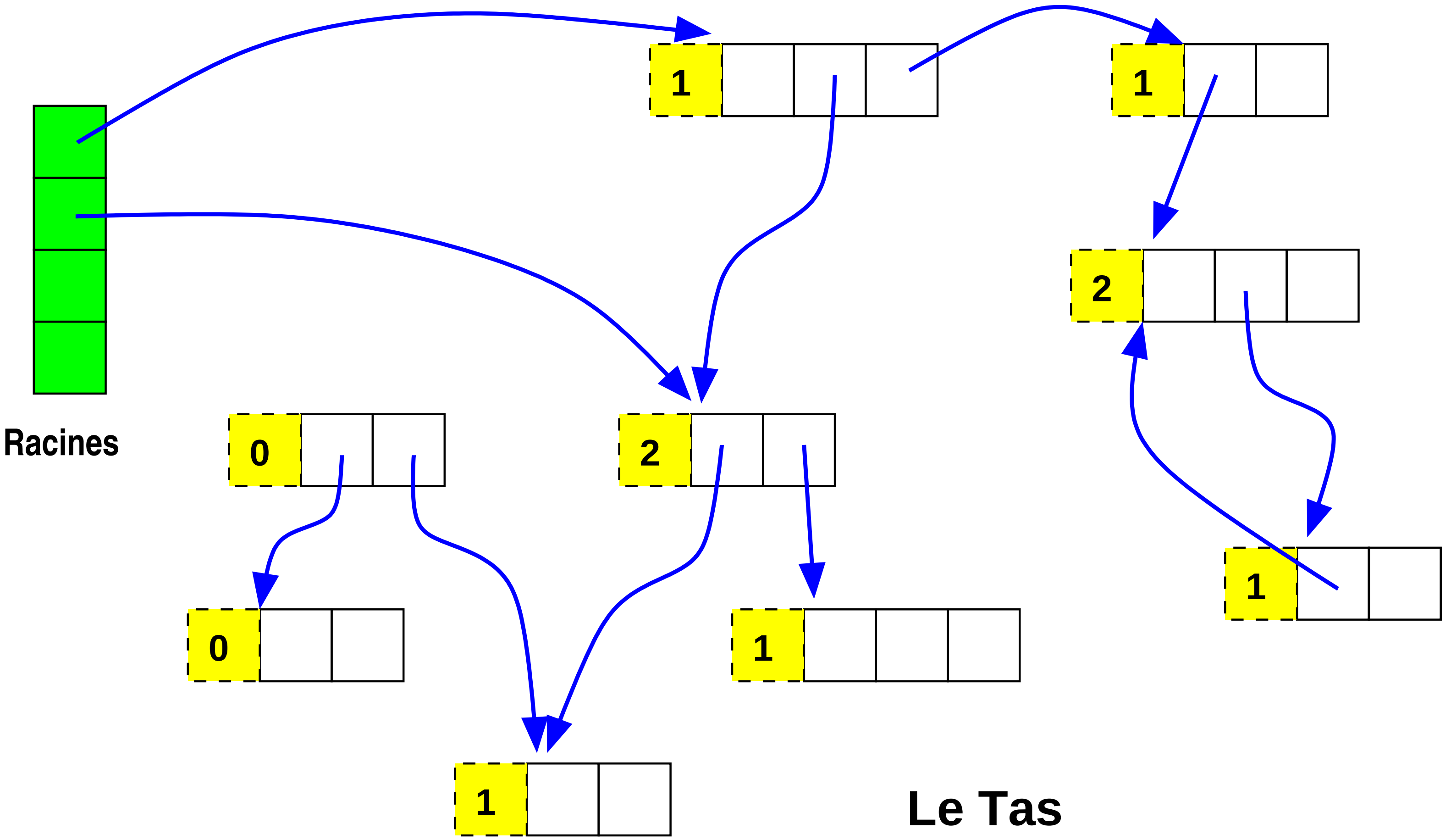
Avantages

- méthode incrémentale. Pas de blocage. Bien pour temps-réel.

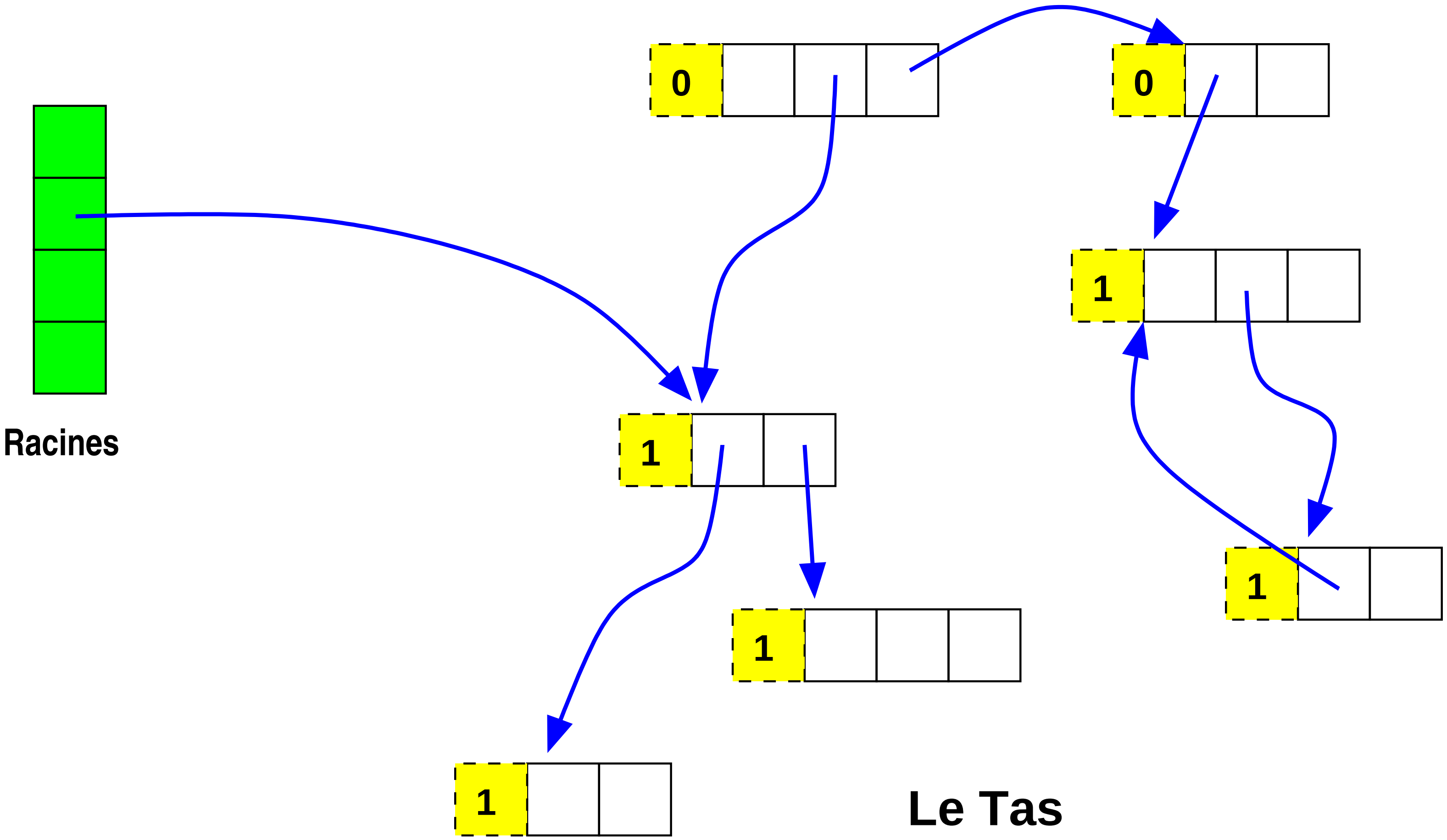
Inconvénients

- ne récupère pas les cycles.
- coût induit sur toutes les opérations de manipulation des références.

Compteurs de références

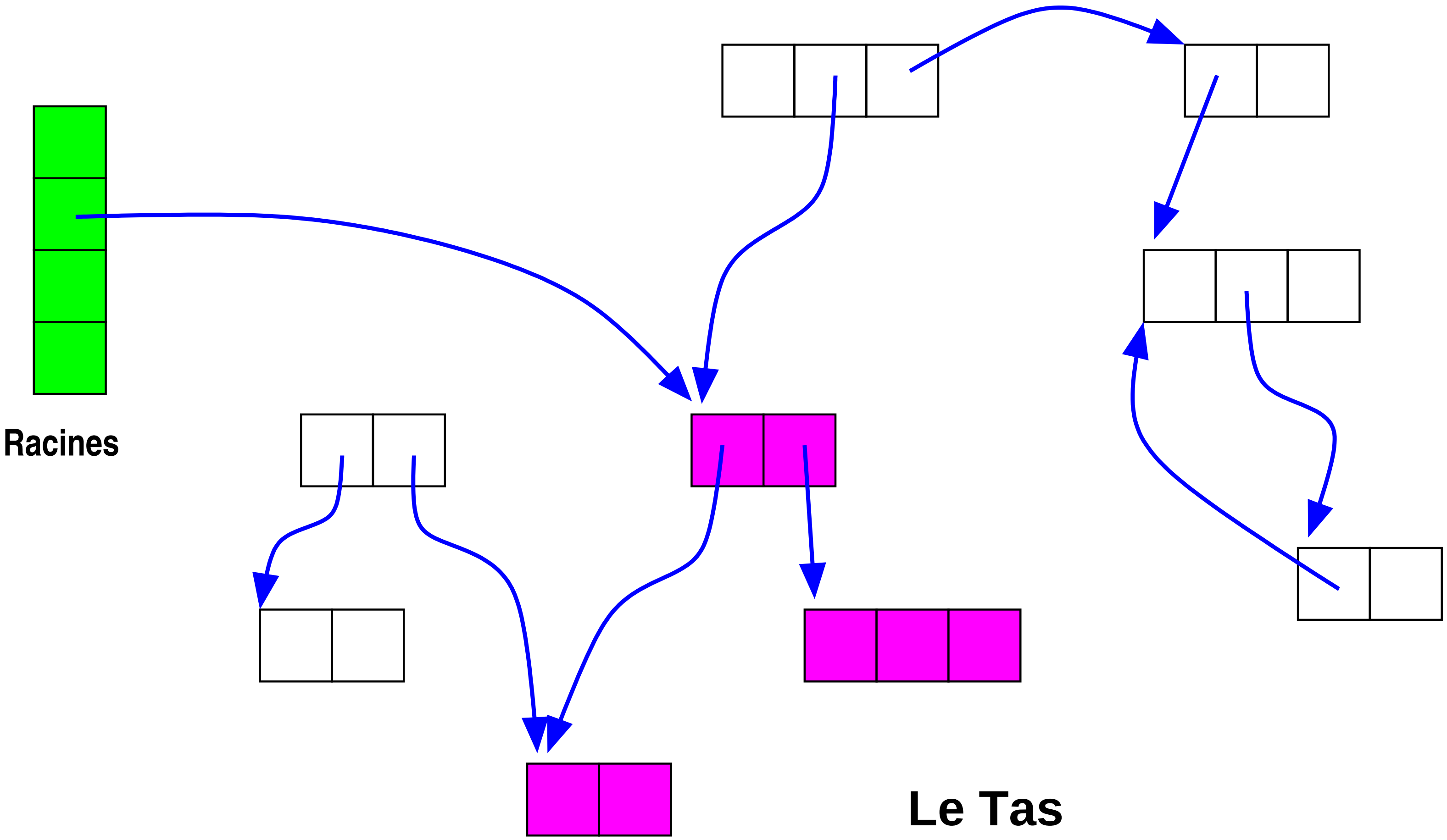


Compteurs de références



Méthodes par traçage

On marque les cellules atteignables depuis les racines. Et on libère les cellules non marquées.



Mark and Sweep

La phase de marquage se fait par une simple exploration en profondeur d'abord du graphe d'accès à partir des racines. Son résultat se fait en général dans un tableau de bits, 1 bit par mot-mémoire. Pour le marquage, on peut éviter une pile en retournant les pointeurs.

La deuxième phase consiste à balayer séquentiellement le tas en récupérant tous les mots-mémoire non marqués.

Avantages

- **tourne en peu de mémoire**
- **autorise les GC conservatifs (pour C ou C++).**

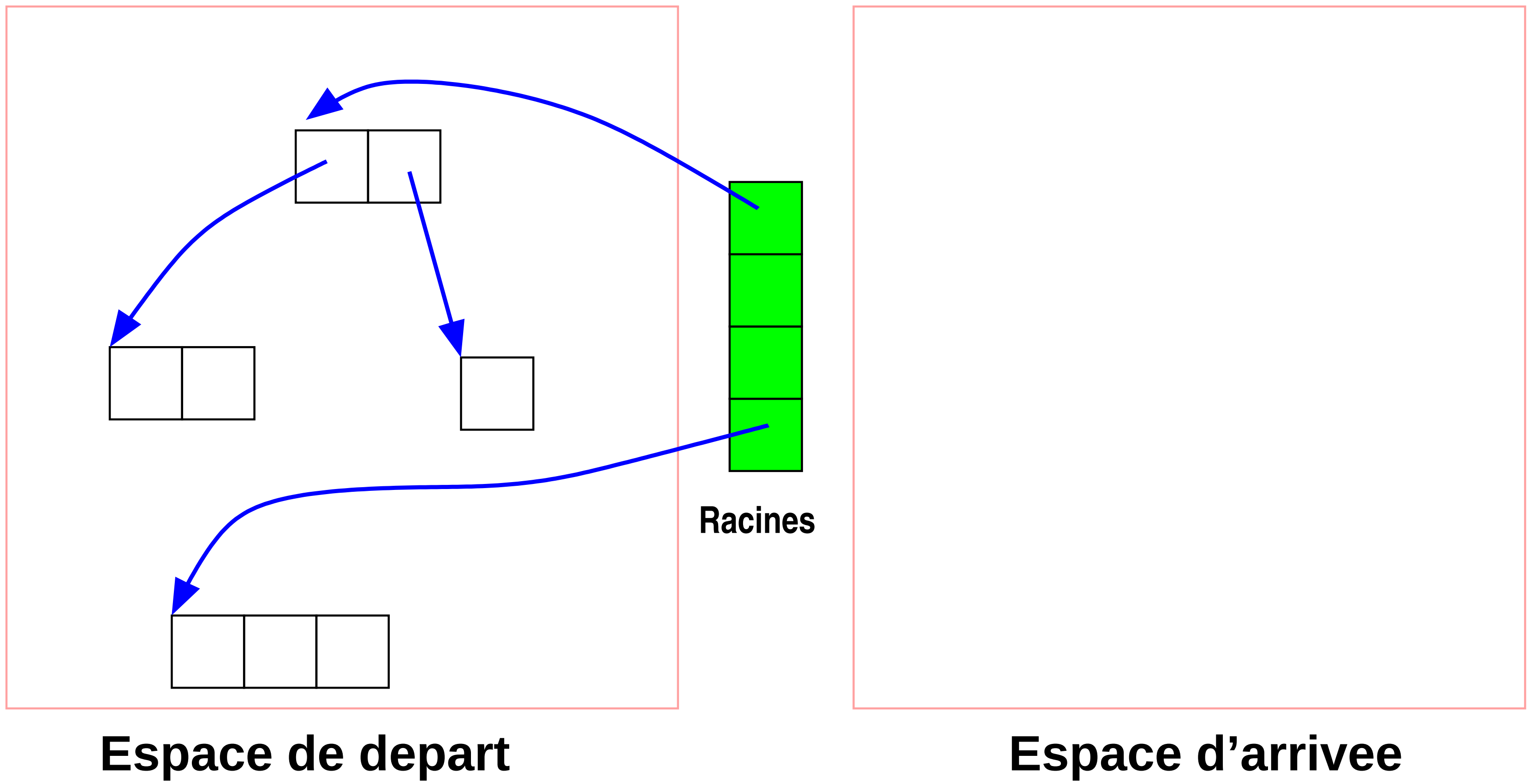
Inconvénients

- **fragmentation**

Coût amorti

$\frac{c_1 N + c_2 T}{T - N}$ **(où T taille du tas, N mots occupés)**

Collecteurs par recopie



Collecteurs par recopie

Le tas est divisé en 2 zones égales: départ (*from-space*) et arrivée (*to-space*) . On alloue des cellules dans la zone de départ à partir d'un index `disponible` indiquant l'emplacement du premier mot-mémoire suivant le dernier mot alloué. Quand `disponible` est au bout de la zone, on recopie les cellules accessibles depuis les racines de la zone départ vers la zone d'arrivée.

Les cellules non-libres seront donc bien compactes. On permute les rôles des deux zones, l'index `disponible` se retrouvant derrière le dernier mot recopié.

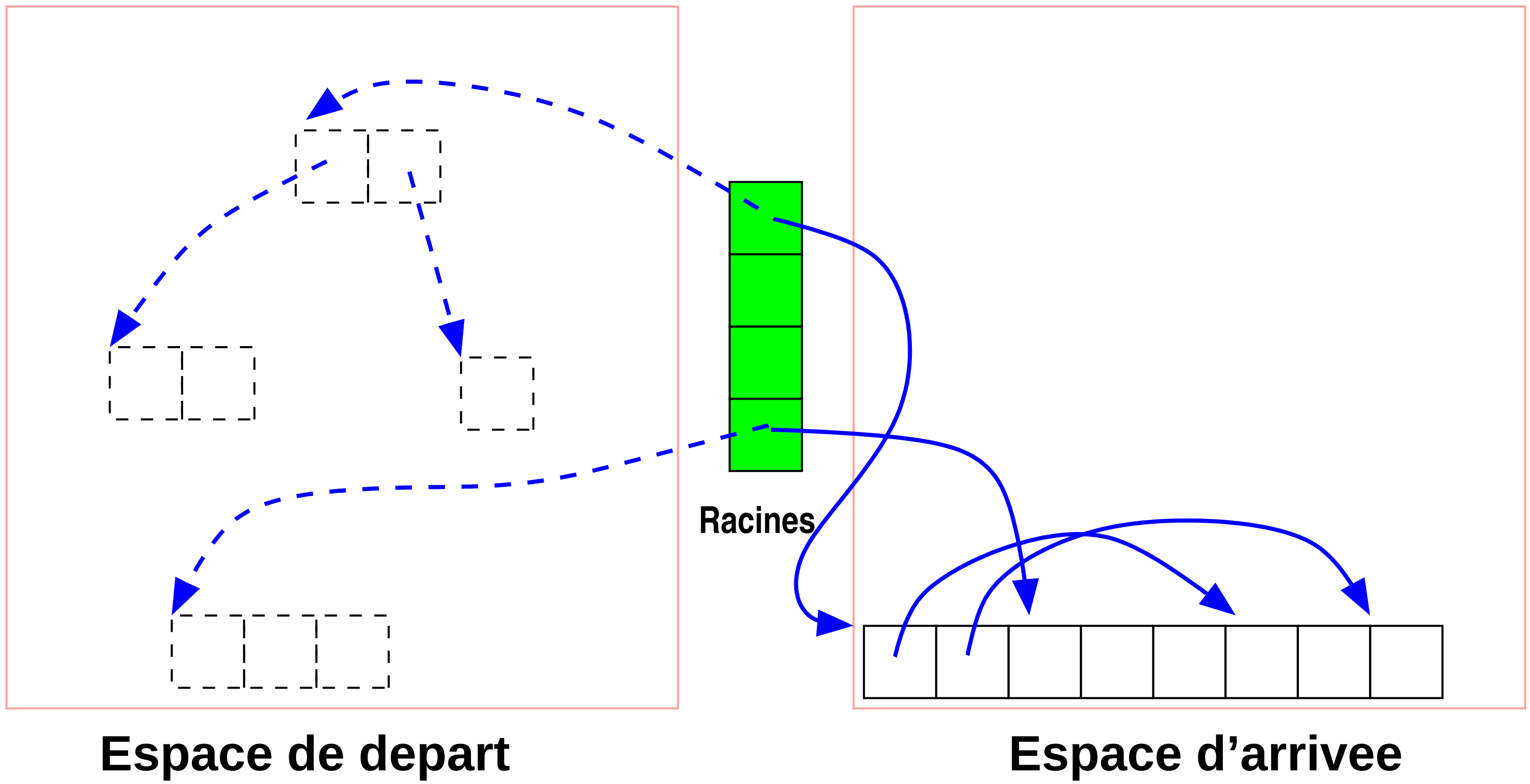
Avantages

- pas de fragmentation

Inconvénients

- il faut disposer de la moitié de la mémoire (en fait on alloue de la mémoire virtuelle, mais cela peut entrainer des défauts de page au moment du GC).

Collecteurs par recopie



Collecteurs par recopie

En fait, il faut mettre à jour les contenus des cellules recopiées pour que les références pointent dans la zone d'arrivée.

Dans l'algorithme de Cheney [70], la recopie se fait par un parcours en largeur du graphe des cellules accessibles à partir des racines.

Quand on recopie une cellule de la zone de départ vers la zone d'arrivée, on laisse un renvoi vers sa nouvelle adresse dans le premier mot de cette cellule dans la zone de départ.

Pour le parcours, on utilise une file dont l'index de début `debut` et l'index de fin `disponible` (en fait le premier mot derrière la file) sont initialement positionnés sur le début de la zone d'arrivée.

On recopie d'abord les mots directement pointés par les racines, que l'on range dans la file.

Tant que la file n'est pas vide, on recopie les mots pointés par le premier élément de la file.

Recopie

Pour copier une cellule, on regarde d'abord si son premier mot pointe vers la zone d'arrivée. Si c'est le cas, rien à faire.

Sinon, on copie la cellule au bout de la file dans la zone d'arrivée. Et on met un pointeur de renvoi dans le 1er mot de son ancienne adresse.

Coût

$\frac{cN}{T/2-N}$ (où T taille du tas, N mots occupés)

GC à générations

80% à 98% des cellules récemment allouées disparaissent rapidement. Par ailleurs, un petit nombre reste ad vitam eternam.

Un GC par recopie va passer son temps à essayer de récupérer de la place pour les cellules qui disparaissent tout de suite, en recopiant sans arret les cellules qui restent pour toujours.

Dans un GC à générations, on range dans des zones différentes les jeunes et les vieux.

Mais on doit inclure les vieux pointant vers les jeunes dans les racines de la génération des jeunes. Il y en a très peu.

Coût

Typiquement $T/N = 20$ dans la jeune génération. Donc le coût est $\frac{cN}{T/2-N} = c/9$. Génération de 1 MO pour 50 MO de taille totale.

Il y a un coût induit sur toute expression d'affectation de référence, car il faut prévoir le cas où on se retrouve dans la vieille génération pointant sur la jeune génération. Si peu de structures modifiables, ce GC est très bon.

Autres techniques

- **méthodes hybrides (***Lang***)**
- **méthodes incrémentales (***Baker***)**
- **utilisation du hardware, des défauts de page (***Appel, Ellis, Li***)**
- **GC concurrents (***Dijkstra, Lamport, Doligez***)**
- **GC distribués (***Hughes, le Fessant, Shapiro***)**

- **preuves de correction de GC (***Doligez-Gonthier***)**

En TD

- continuer évaluateur, interpréteur, vérificateur de types, synthétiseur de types.
- mettre le GC dans PCF

A la maison et prochaine fois

- Objets et Modules