

7. Tables de hachage



Le nom et la chose



Parfois sans le savoir, on fait toujours la distinction entre **une chose** et **le nom** qu'on lui donne.

Nom, prénom → Personne

Adresse mémoire → Structure de données

Adresse IP → Processus

...

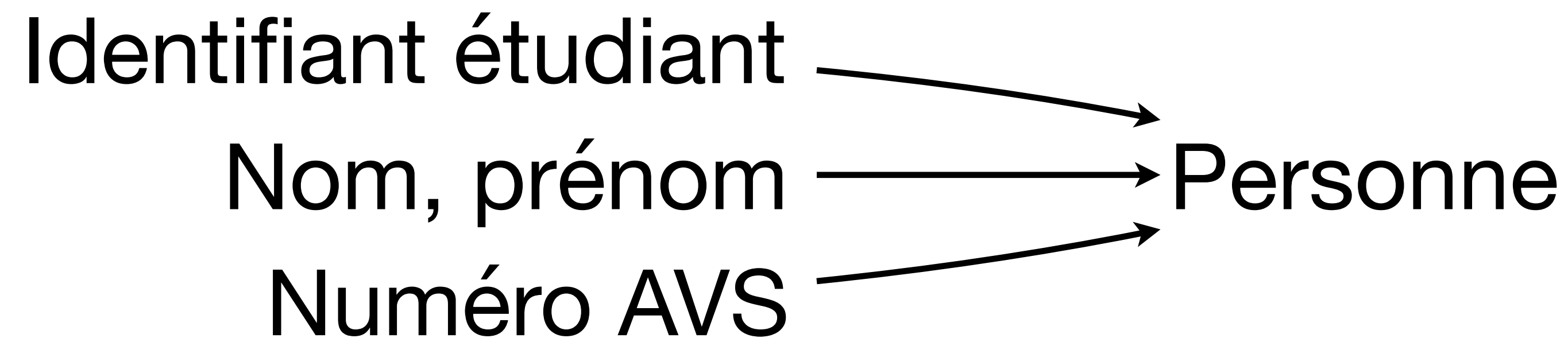
Cette relation dépend du contexte dans lequel ils sont utilisés

Le nom et la chose

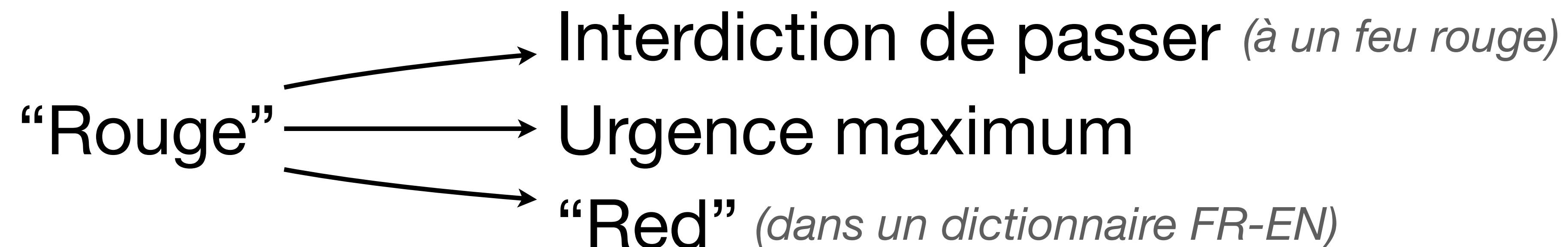


En fonction du contexte...

...une chose peut avoir différents noms



...un nom peut correspondre à différentes choses



Le nom et la chose



Dans un système informatique, un “nom” peut être, par exemple,

- Une adresse mémoire
- Une variable
- Un identifiant globalement unique
- Tout autre objet de notre choix !

Le **tableau associatif** permet d’associer un nom arbitraire à tout objet.

On appelle ce nom la **clé** de l’objet, qu’on appelle la **valeur**.



Le **tableau associatif** permet d'associer un nom arbitraire à tout objet.

On appelle ce nom la **clé** de l'objet, qu'on appelle la **valeur**.

Il offre aussi les opérations de

- **Insertion** (insert/put) - Ajoute une nouvelle association clé-valeur
- **Recherche** (find/search/get) - Retourne la valeur associée à la clé donnée
- **Suppression** (erase/delete) - Retire une association clé-valeur

Mises en oeuvre possibles



**Tableau
trié**

Insérer

Rechercher

Supprimer

Mises en oeuvre possibles



**Tableau
trié**

**Tableau
non trié**

Insérer

$O(n)$

Rechercher

$O(\log_2(n))$

Supprimer

$O(n)$

Mises en oeuvre possibles



	Tableau trié	Tableau non trié	Liste triée
Insérer	$O(n)$	$O(1)$	
Rechercher	$O(\log_2(n))$	$O(n)$	
Supprimer	$O(n)$	$O(n)$	

Mises en oeuvre possibles



	Tableau trié	Tableau non trié	Liste triée	Liste non triée
Insérer	$O(n)$	$O(1)$	$O(n)$	
Rechercher	$O(\log_2(n))$	$O(n)$	$O(n)$	
Supprimer	$O(n)$	$O(n)$	$O(n)$	

Mises en oeuvre possibles



	Tableau trié	Tableau non trié	Liste triée	Liste non triée	Arbre
Insérer	$O(n)$	$O(1)$	$O(n)$	$O(1)$	
Rechercher	$O(\log_2(n))$	$O(n)$	$O(n)$	$O(n)$	
Supprimer	$O(n)$	$O(n)$	$O(n)$	$O(n)$	

Mises en oeuvre possibles



	Tableau trié	Tableau non trié	Liste triée	Liste non triée	Arbre	Table de hachage
Insérer	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(\log_2(n))$	$O(1)$
Rechercher	$O(\log_2(n))$	$O(n)$	$O(n)$	$O(n)$	$O(\log_2(n))$	$O(1)$
Supprimer	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(\log_2(n))$	$O(1)$

Principe du hachage



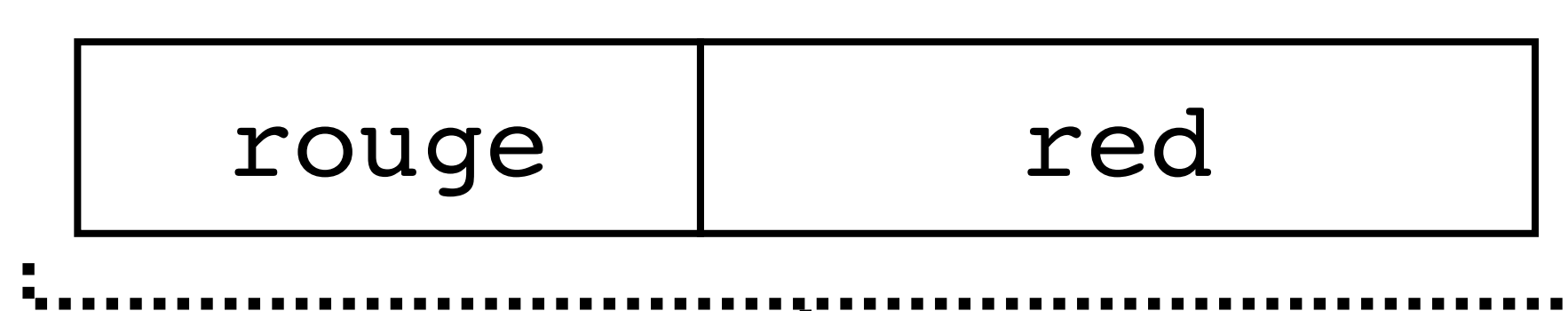
Pour chaque entrée, créer une paire contenant la clé et sa valeur

rouge	red
-------	-----

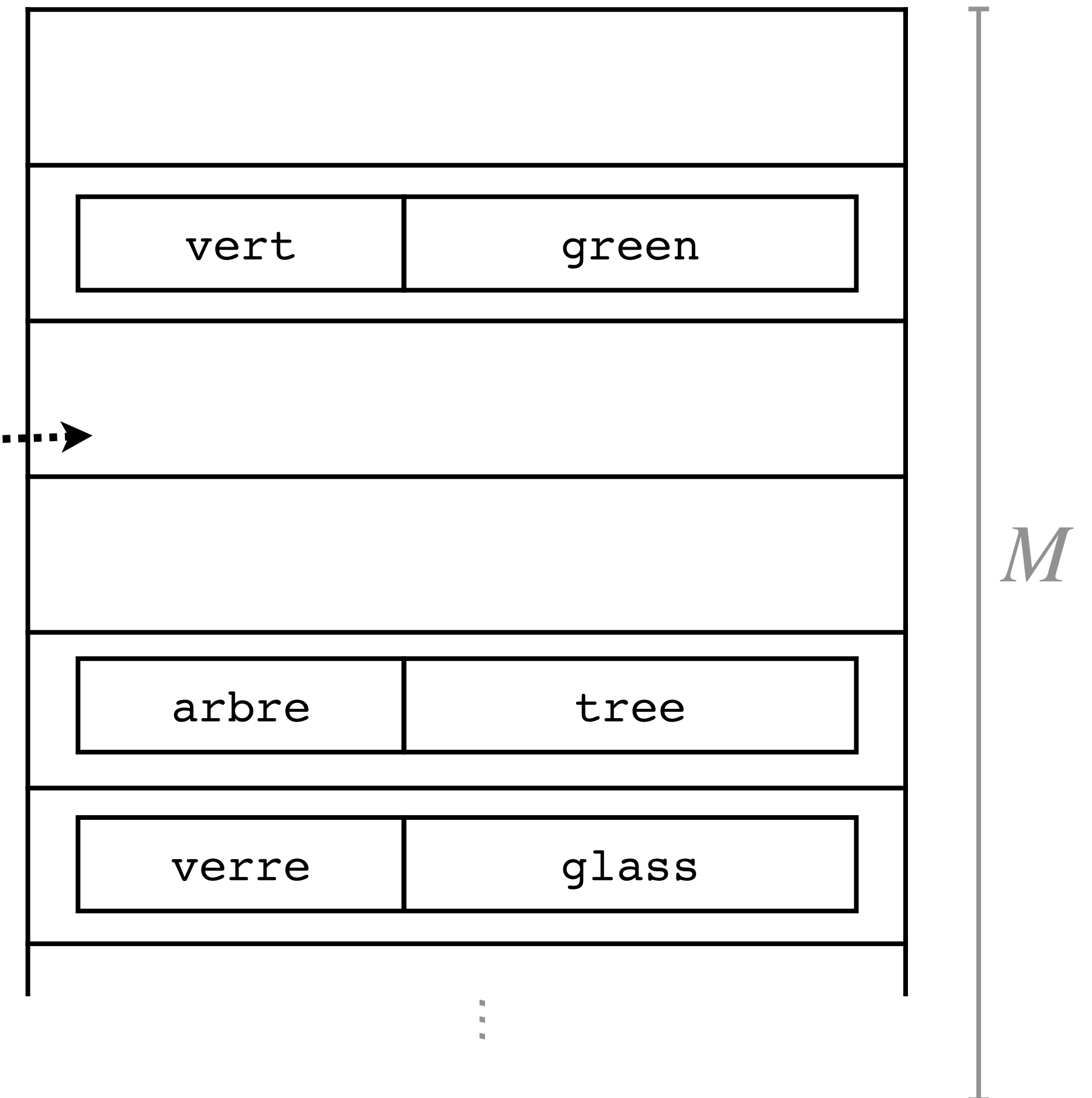
Principe du hachage



Pour chaque entrée, créer une paire contenant la clé et sa valeur



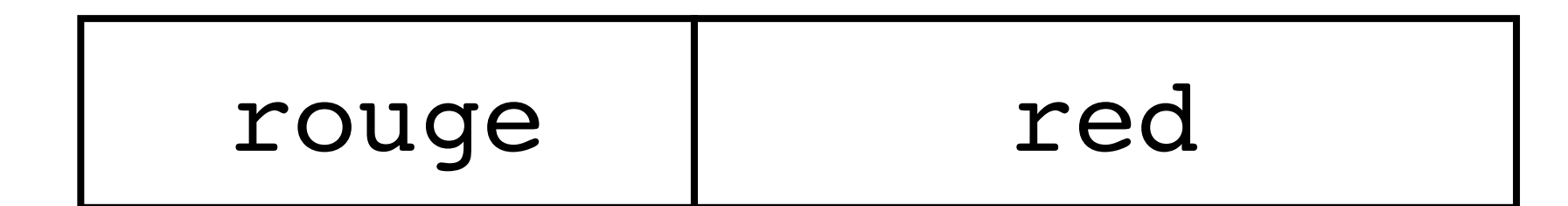
Stocker cette paire dans un tableau de taille M



Principe du hachage



Pour chaque entrée, créer une paire contenant la clé et sa valeur



$$h(\text{"rouge"}) = 2$$

Stocker cette paire dans un tableau de taille M

À quelle position ?

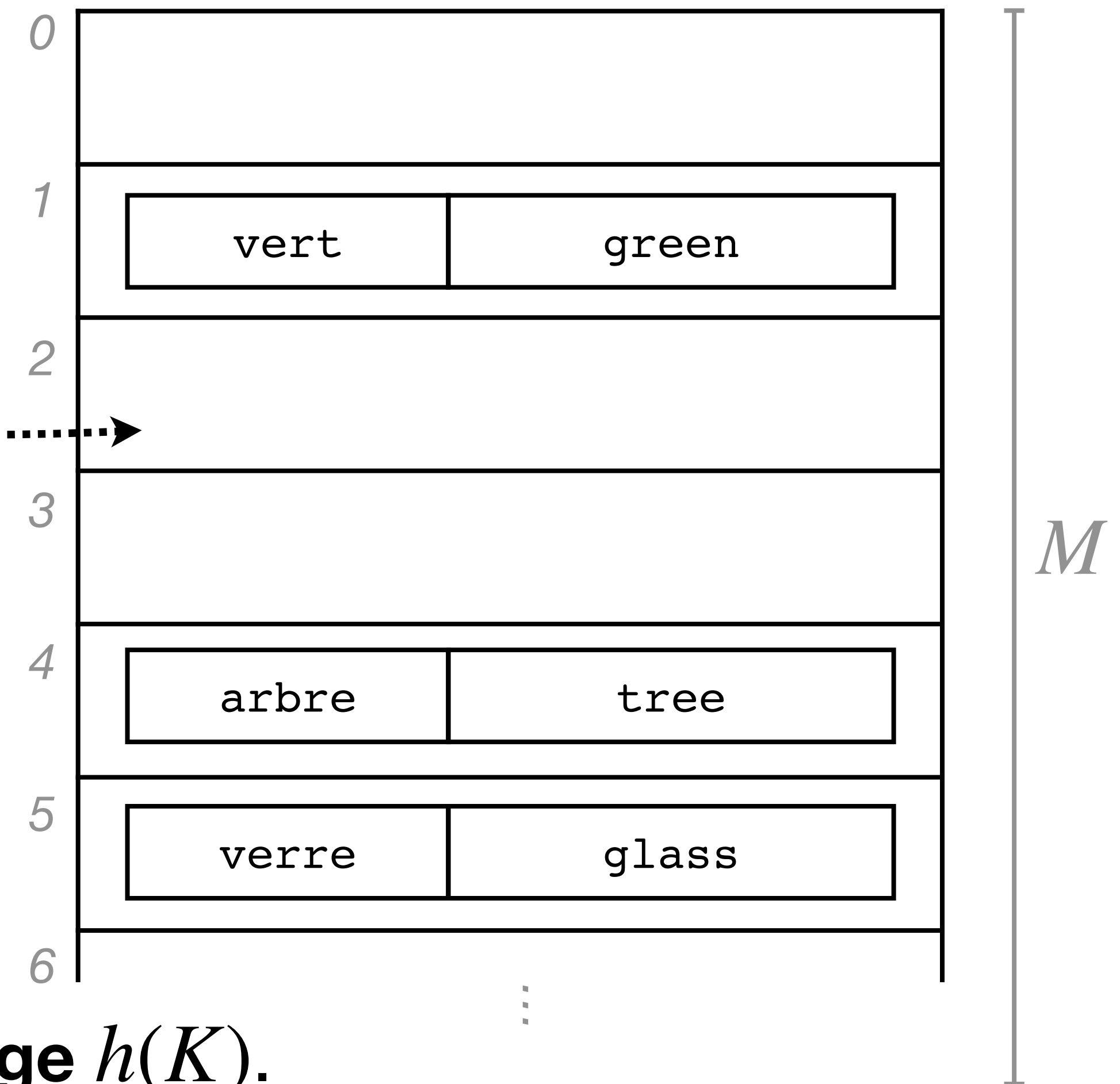
La clé K est **hachée** par une **fonction de hachage**

$$h : \mathcal{K} \mapsto \{0, 1, \dots, M - 1\}$$

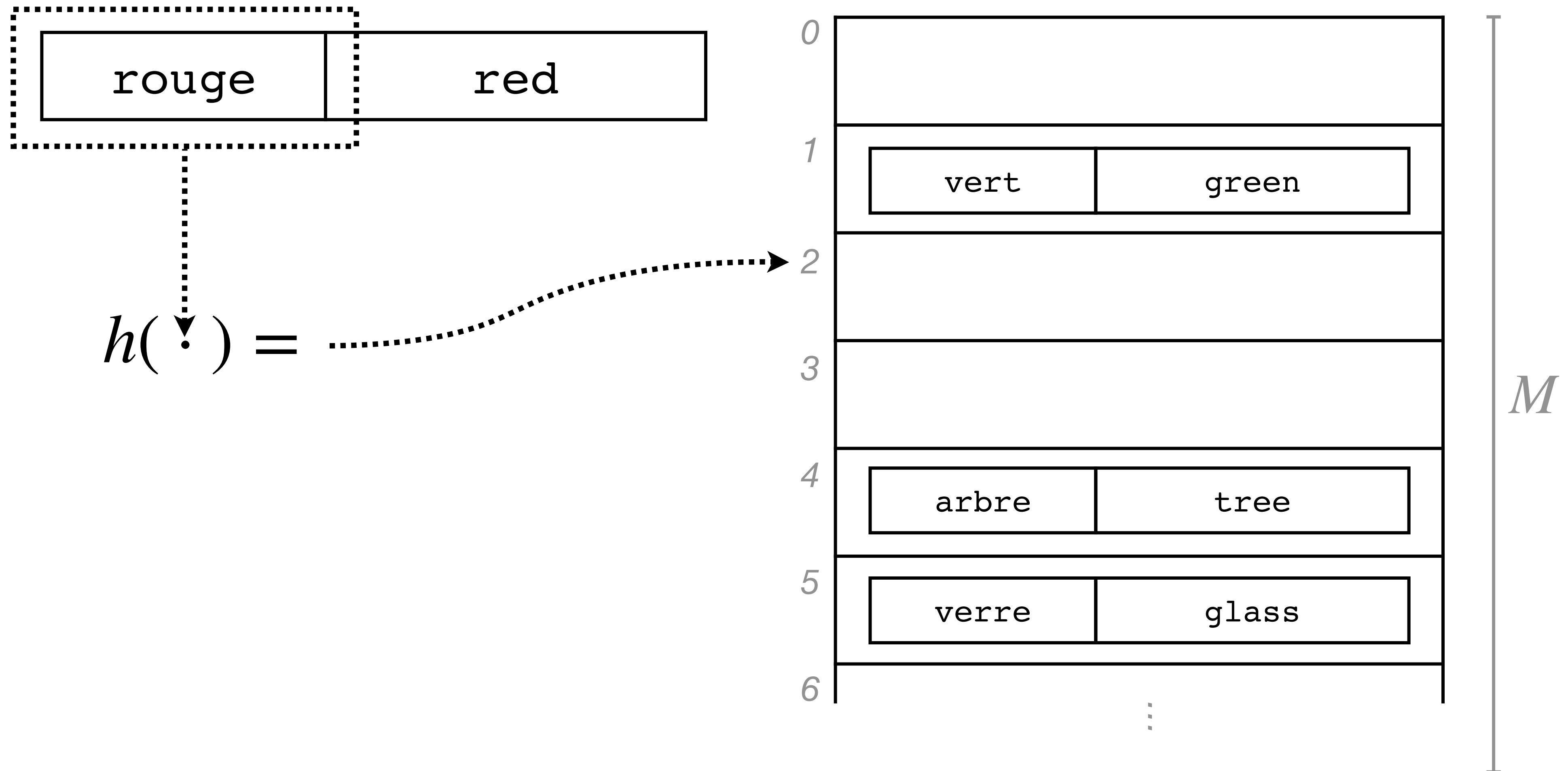
avec $\mathcal{K}$ l'ensemble des clés possibles

“pour chaque clé K , $h(K)$ est un entier entre 0 et $M - 1$.”

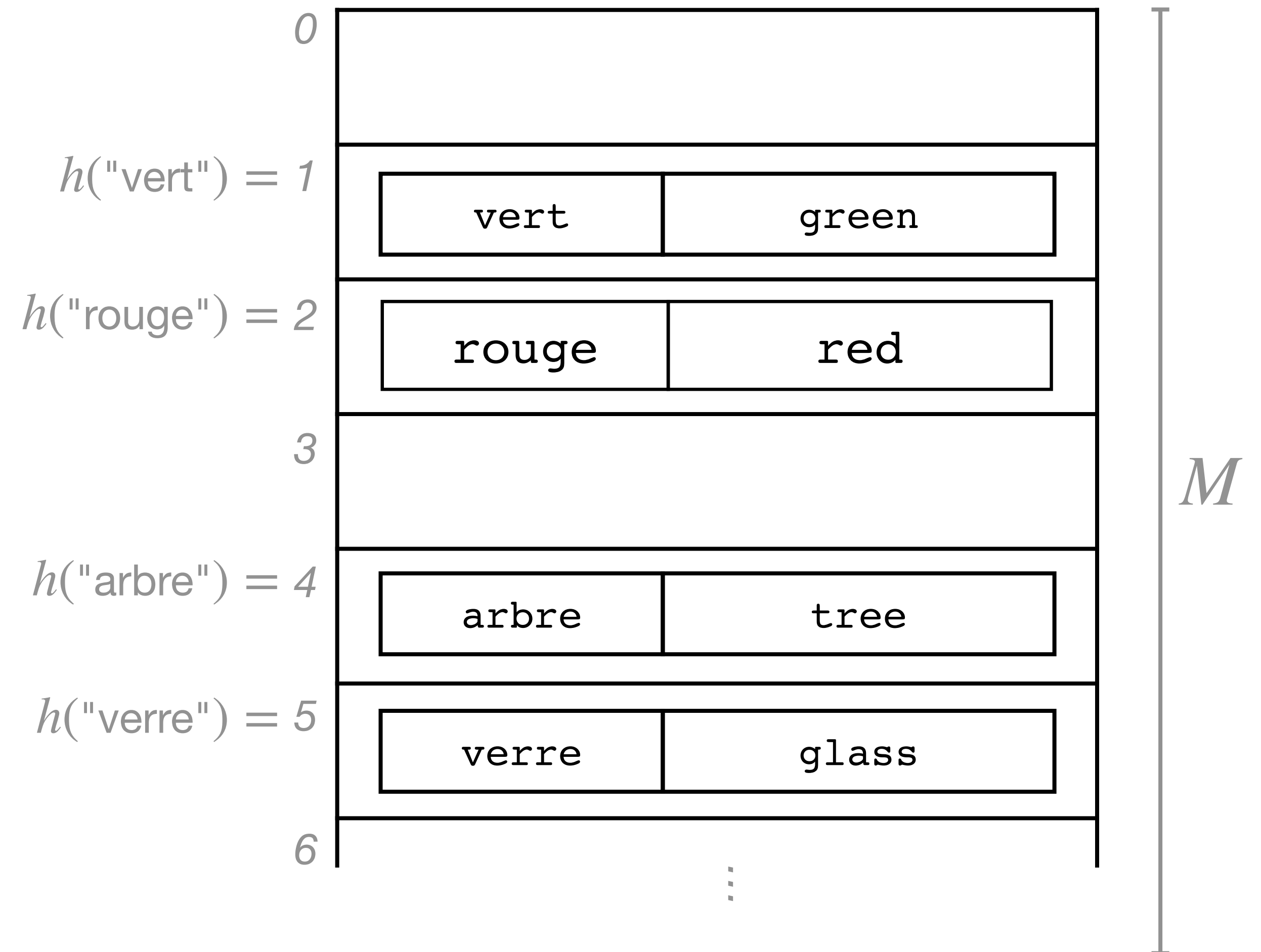
La paire (K, V) est donc placée à l'adresse de hachage $h(K)$.



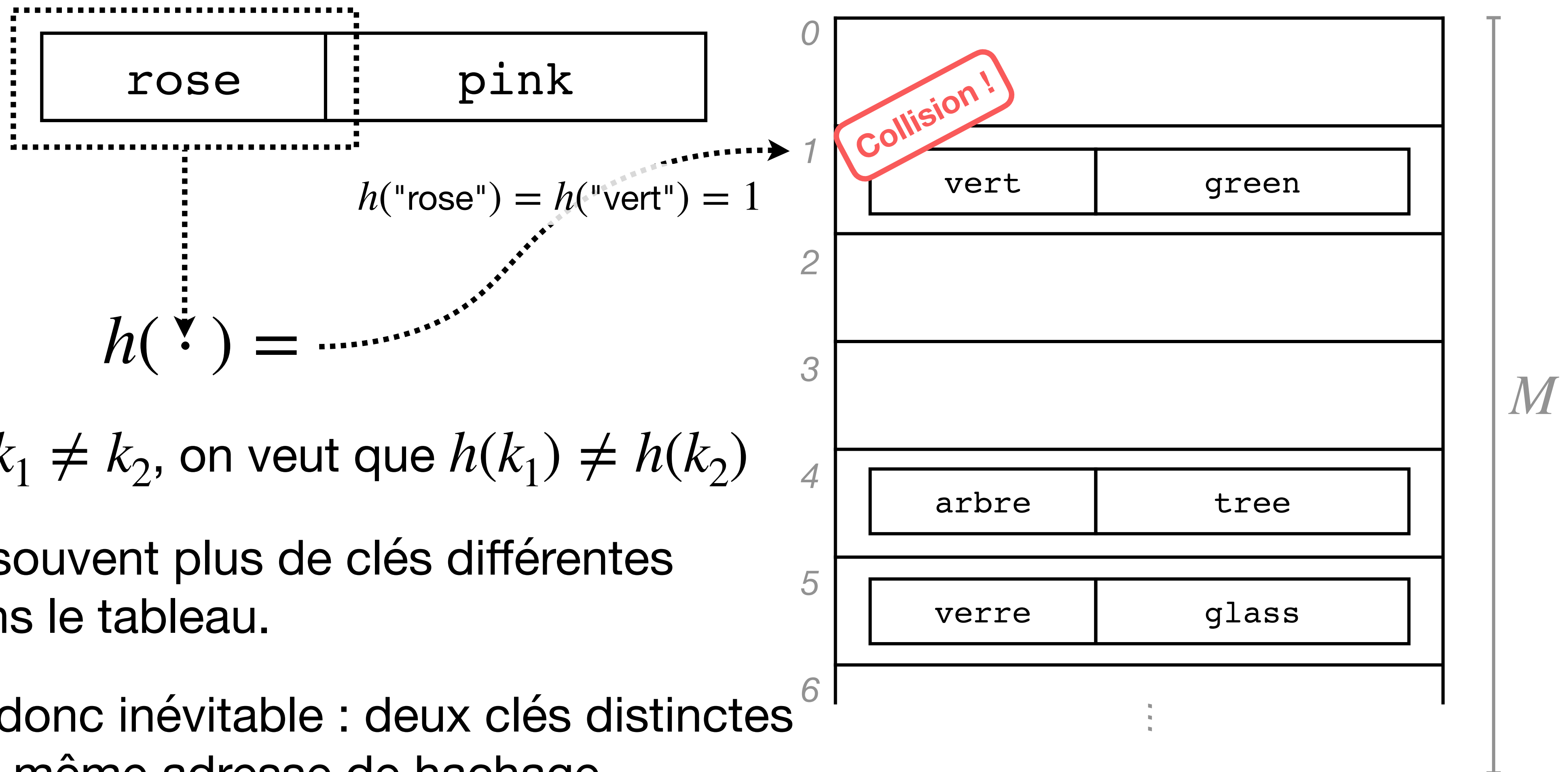
Principe du hachage



Principe du hachage



Principe du hachage



Idéalement, pour $k_1 \neq k_2$, on veut que $h(k_1) \neq h(k_2)$

Cependant, il y a souvent plus de clés différentes que de places dans le tableau.

Une **collision** est donc inévitable : deux clés distinctes peuvent donner la même adresse de hachage.

Restent à spécifier



Restent donc à spécifier :

- Le choix de la fonction de hachage $h(\cdot)$
- La stratégie de gestion de collisions
- La taille M du tableau

7.1. Gestion des collisions



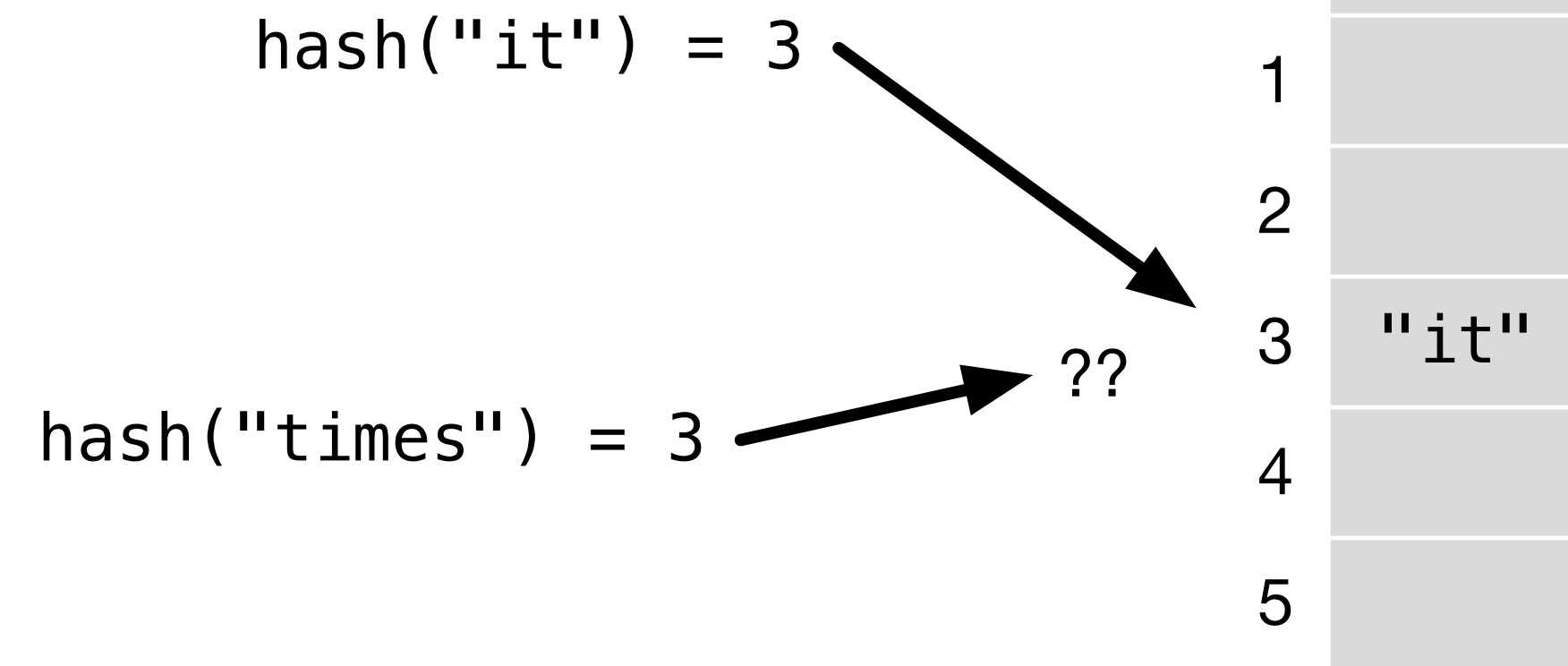
Collisions



Lorsque deux clés distinctes produisent le même hash :

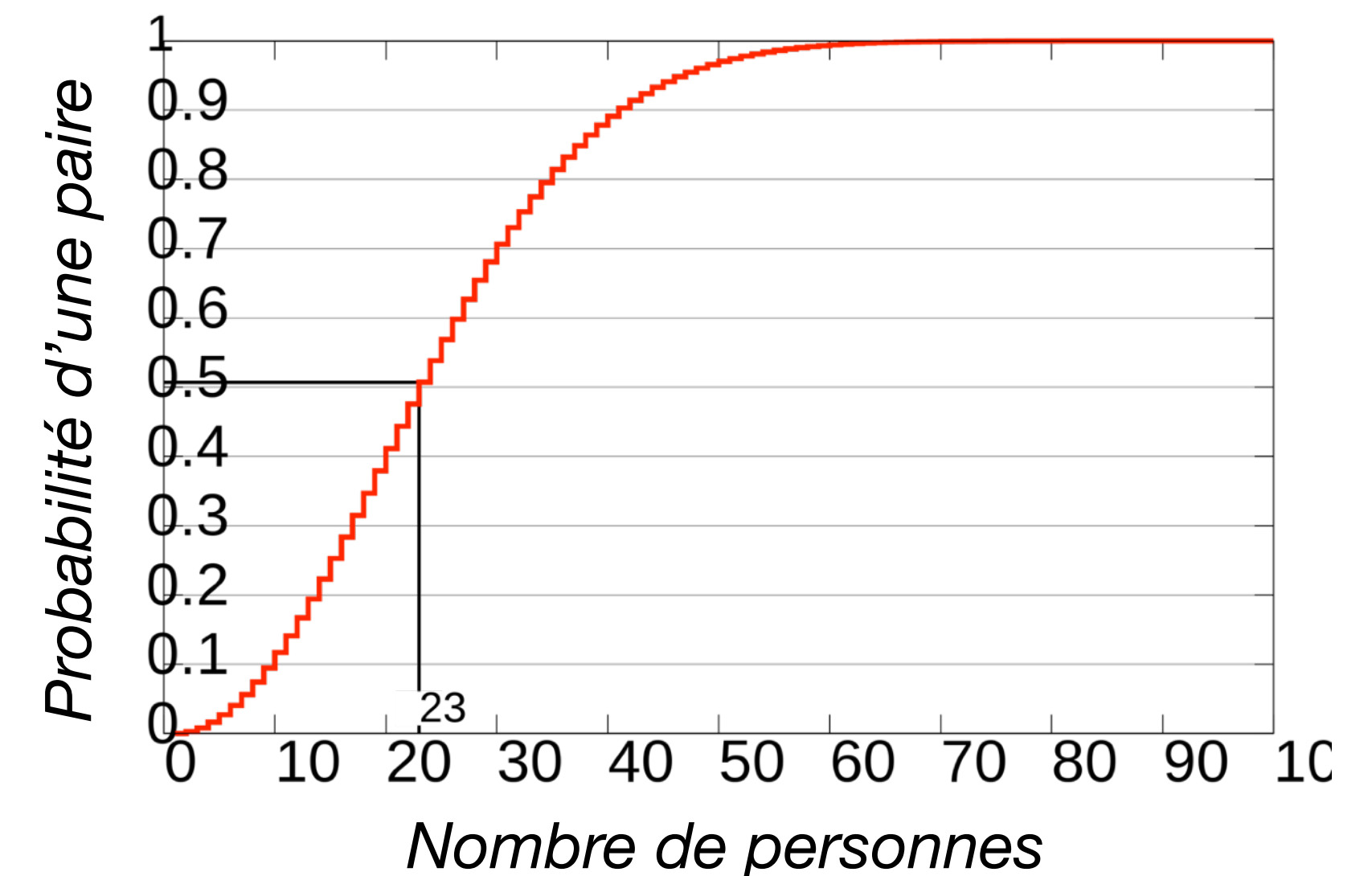
$$k_1 \neq k_2,$$

$$h(k_1) = h(k_2)$$



Même avec une parfaite distribution (uniforme) de la fonction de hachage, les collisions sont très probables (cf. problème des anniversaires).

Il est donc nécessaire de **traiter les collisions efficacement**.



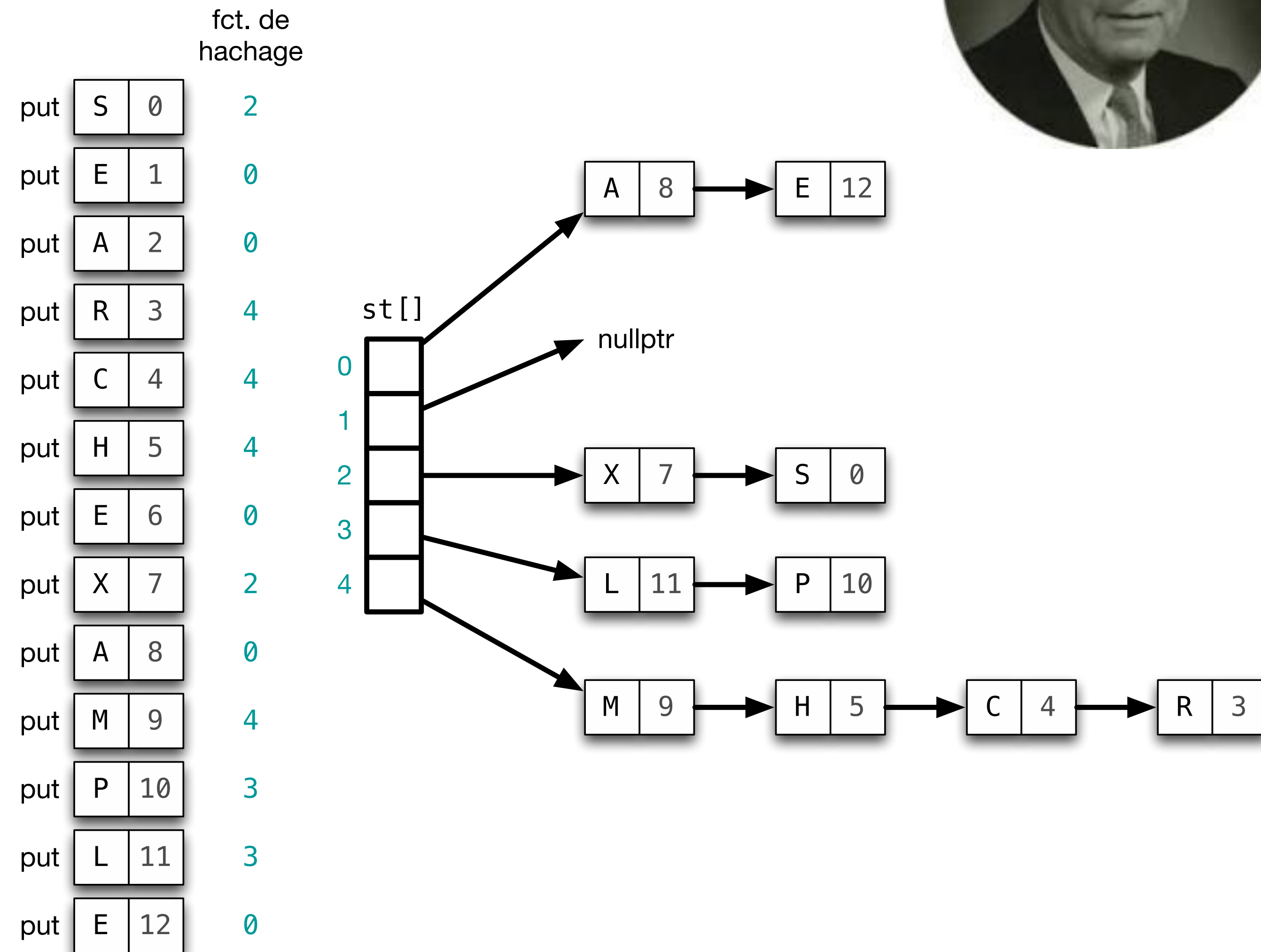
Source : https://en.wikipedia.org/wiki/Birthday_problem



Résolution des collisions par chaînage



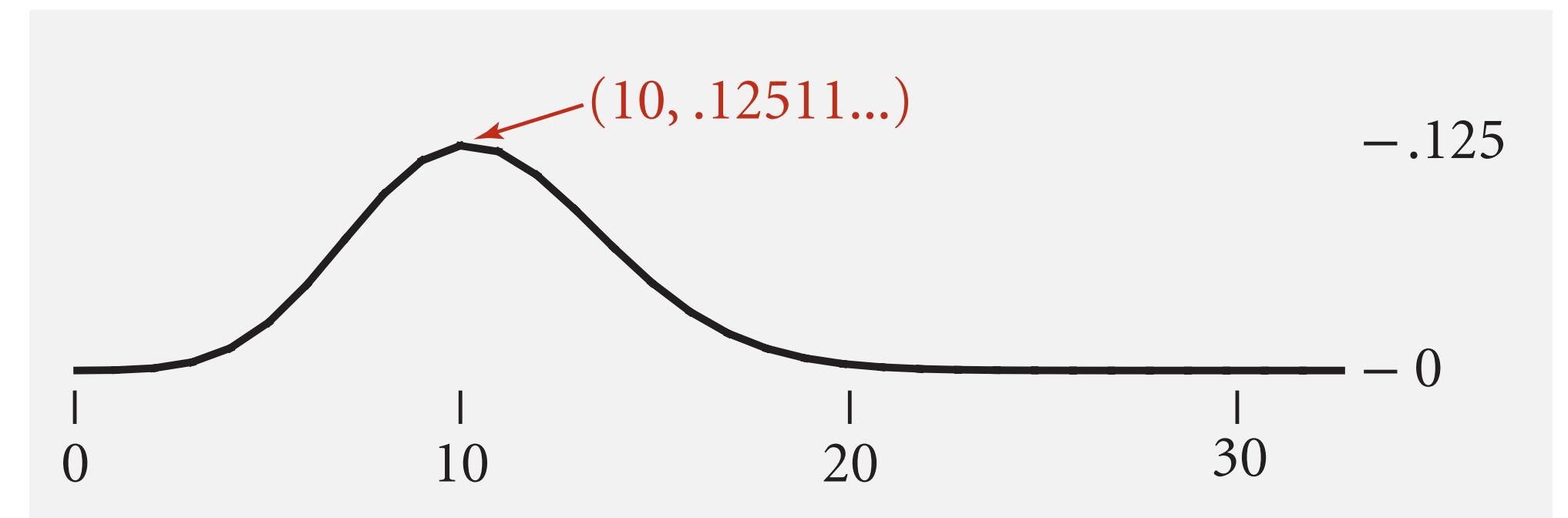
- Inventé par H.P. Luhn, IBM 1953
- Utilise un tableau de M listes simplement chaînées pour stocker N paires clé/valeur, avec $M < N$
- **Insertion** : Insère l'élément au début de la liste d'index $h(k)$, s'il n'y est pas déjà présent.
- **Recherche**: Parcourt la liste d'indice $h(k)$ uniquement
- $\alpha = N/M$ est le taux d'occupation.
Ici $\alpha = 2$ *a.k.a. nombre moyen d'élément par index*



Efficacité du chaînage



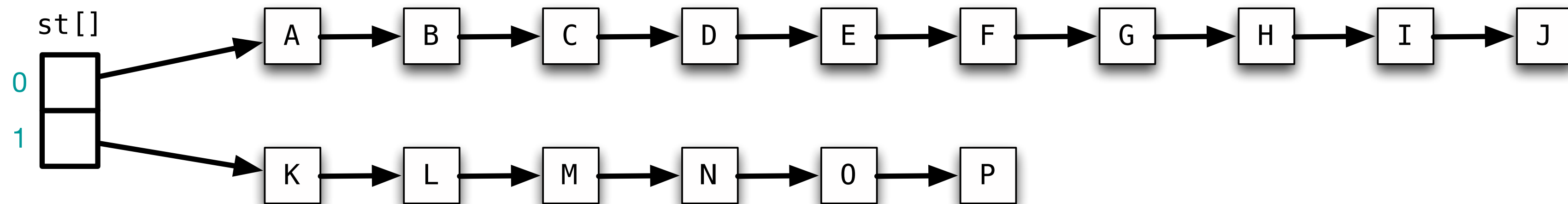
- Avec un hachage uniforme, la distribution des tailles des listes est une distribution binomiale.
- En pratique, la taille des listes - i.e. la complexité des opérations - est proche du taux d'occupation N/M
- M trop grand $\Rightarrow$ trop de listes vides
- M trop petit $\Rightarrow$ listes trop longues
- Choix typique: $M \approx \frac{N}{4} \Rightarrow 4$ éléments par liste en moyenne — complexité constante



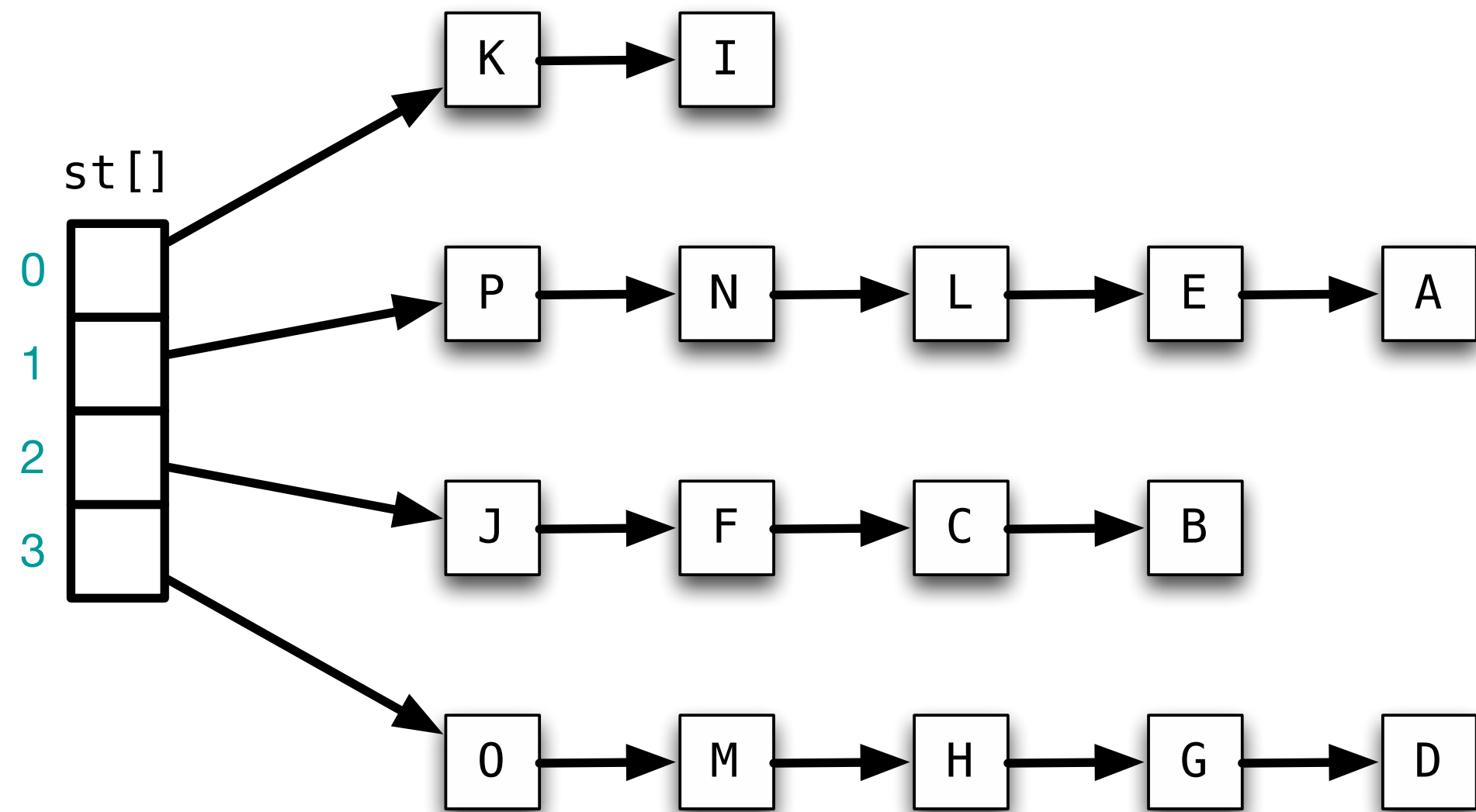
Distribution binomiale de la taille d'une liste lorsque $N = 10^4$, $M = 10^3$ ($\alpha = 10$)



Avant redimensionnement



Après redimensionnement



- Pour garder une longueur moyenne de liste N/M proche du 4 visé, on va
 - doubler M quand $\frac{N}{M} \geq 8$
 - diviser M par deux quand $\frac{N}{M} \leq 2$
- Attention, il faut re-hacher toutes les clés quand on change M



Résolution des collisions par sondage



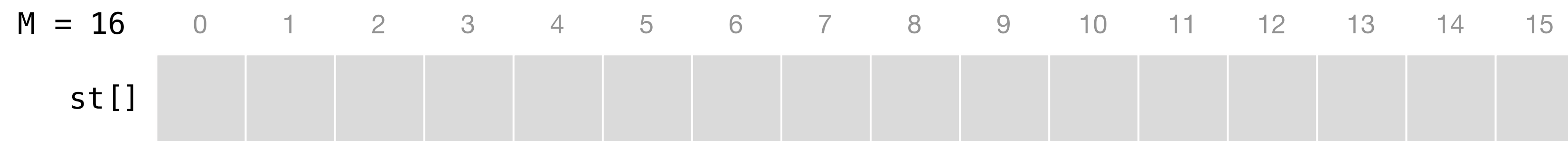
On stocke N paires clé-valeur dans une table de taille $M > N$ en utilisant les emplacements vides dans la table pour la résolution de collisions.

- Variantes:
 - Sondage linéaire (*linear probing*)
 - Double hachage (*double hashing*)
 - Hachage coalescent (*coalesced hashing*)
 - ...

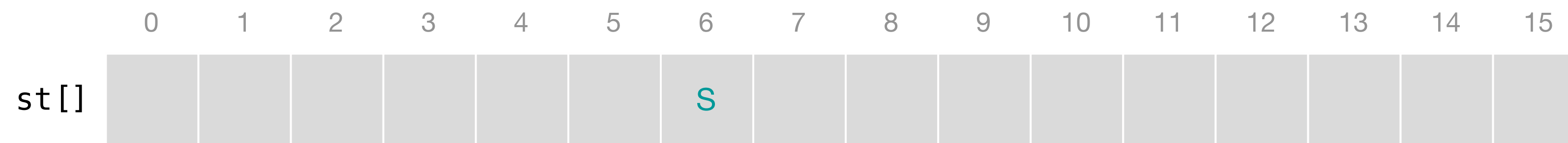
Sondage linéaire



Insérer une clé k : si l'emplacement à l'indice $h(k)$ est occupé, essayer $(h(k) + 1) \bmod M$, $(h(k) + 2) \bmod M$, ... jusqu'à trouver un emplacement vide



insérer S (hash(S) = 6)

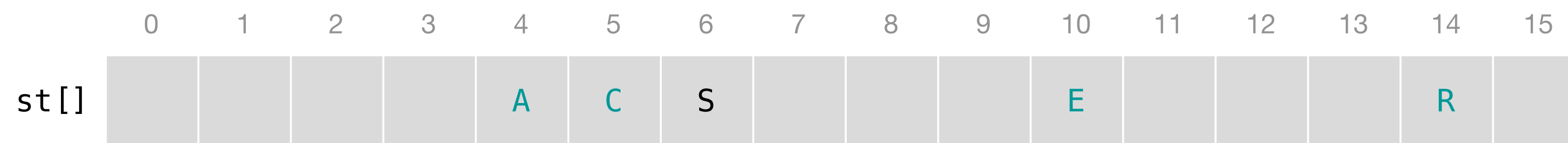


insérer E (hash(E) = 10)

insérer A (hash(A) = 4)

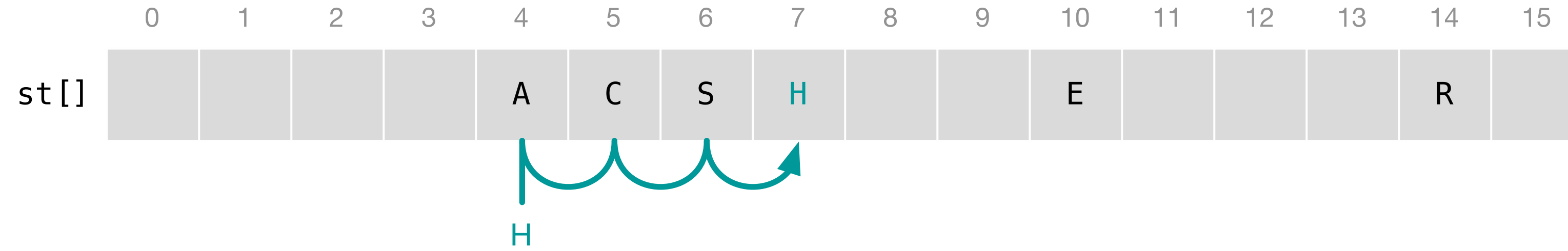
insérer R (hash(R) = 14)

insérer C (hash(C) = 5)





insérer H ($\text{hash}(H) = 4$)

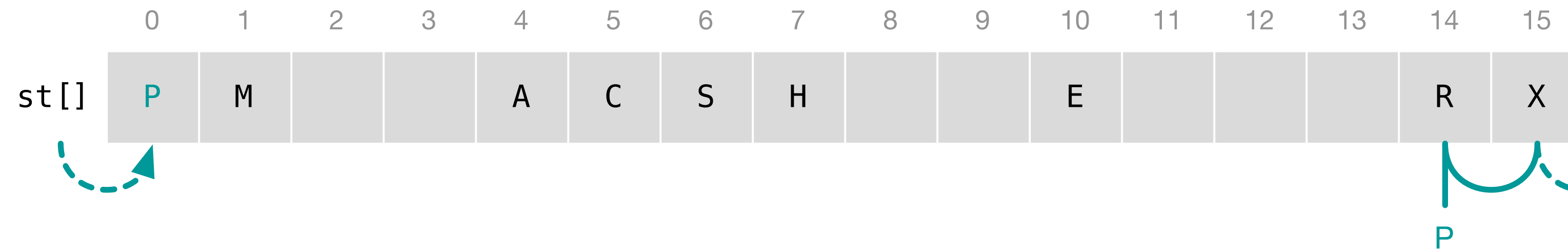


insérer X ($\text{hash}(X) = 15$)

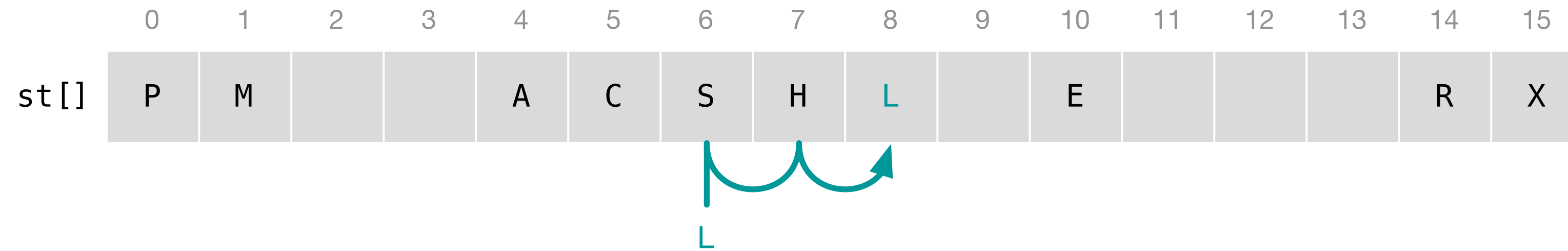
insérer M ($\text{hash}(M) = 1$)



insérer P ($\text{hash}(P) = 14$)



insérer L ($\text{hash}(L) = 6$)



Sondage linéaire



- **Chercher** la clé k : si l'emplacement à l'indice $h(k)$ est occupé mais ne correspond pas à k , essayer $(h(k) + 1) \bmod M$, $(h(k) + 2) \bmod M$, ... jusqu'à trouver soit k soit un emplacement vide

rechercher E (hash(E) = 10)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]	P	M			A	C	S	H	L		E				R	X

↑ succès
E

rechercher L (hash(L) = 6)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]	P	M			A	C	S	H	L		E				R	X

L → succès

rechercher K (hash(K) = 5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]	P	M			A	C	S	H	L		E				R	X

K → échec

Analyse du sondage linéaire



- Donald Knuth montre - en 1962 - que pour une table de hachage avec M positions et $N = \alpha \cdot M$ clés, le nombre de tests à effectuer est de
 - $\approx \frac{1}{2} \left(1 + \frac{1}{1 - \alpha} \right)$ si la recherche aboutit
 - $\approx \frac{1}{2} \left(1 + \frac{1}{(1 - \alpha)^2} \right)$ si la recherche échoue
- Ces valeurs sont grandes quand α approche 1, mais entre 1.5 et 2.5 quand $\alpha < 0.5$
 - M trop grand $\Rightarrow$ trop d'éléments de tableau vides
 - M trop petit $\Rightarrow$ le temps de recherche explose
 - Choix typique: $\alpha = N/M \approx 1/2$

Redimensionner une table de hachage avec sondage linéaire



Pour garder le taux d'occupation $N/M \leq 1/2$

- Doubler la taille du tableau quand $N/M \geq 1/2$
- Réduire le tableau à la moitié quand $N/M \leq 1/8$
- On doit re-hacher toutes les clés quand on change la taille

Avant redimensionnement

M = 8	0	1	2	3	4	5	6	7
st[]		E	S			R	A	

Après redimensionnement

M = 16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]					A		S				E				R	

Supprimer un élément



- On ne peut pas simplement enlever un élément du tableau
- Il faut ré-insérer tous les éléments qui suivent jusqu'au premier emplacement vide

Avant suppression de S

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]	P	M			A	C	S	H	L		E				R	X

Après suppression de S ?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]	P	M			A	C		H	L		E				R	X

ne fonctionne pas, p. ex. si $\text{hash}(H) = 4$