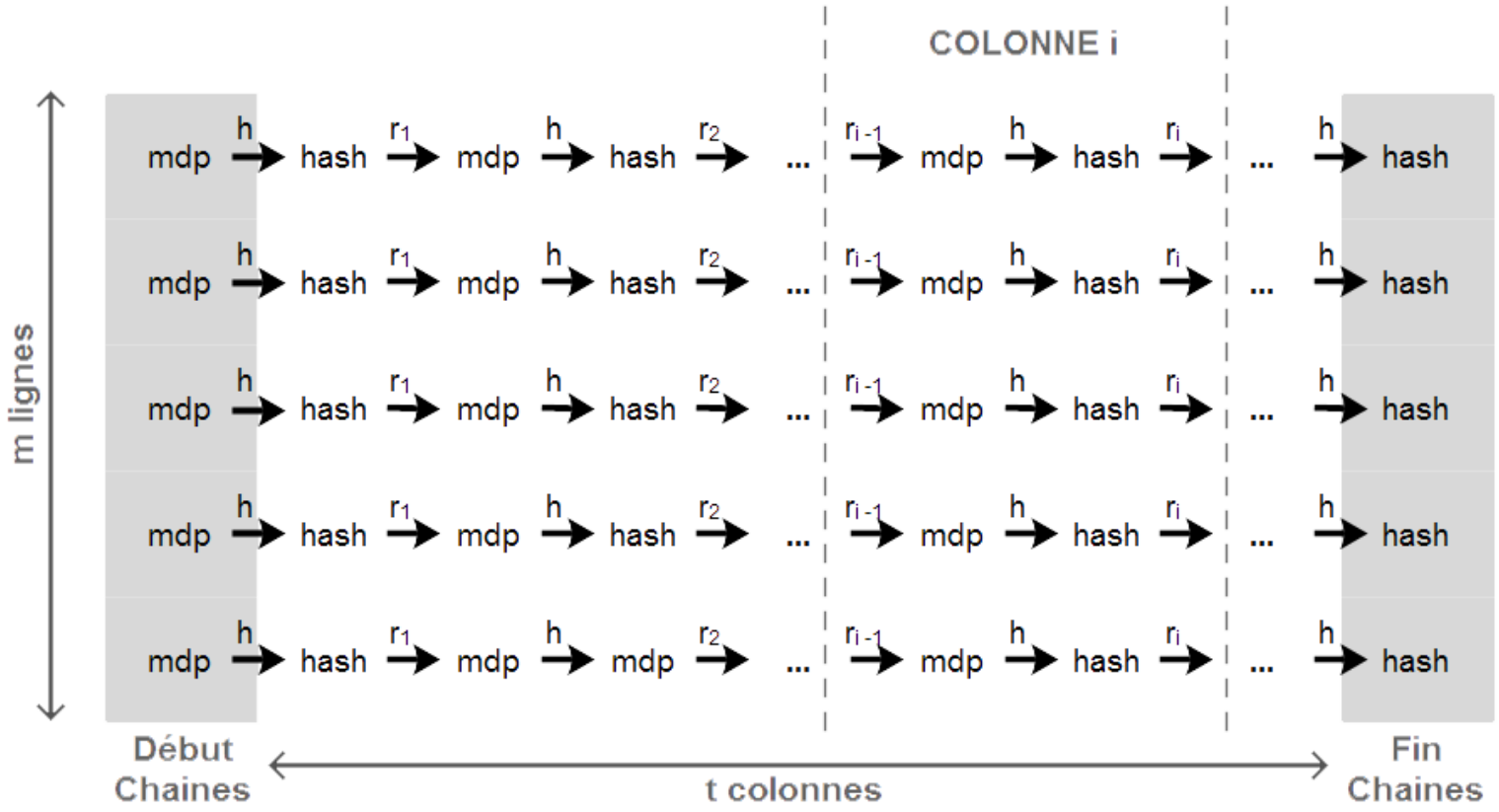
OS OBJECTIF SÉCURITÉ
Architecte de la sécurité informatique

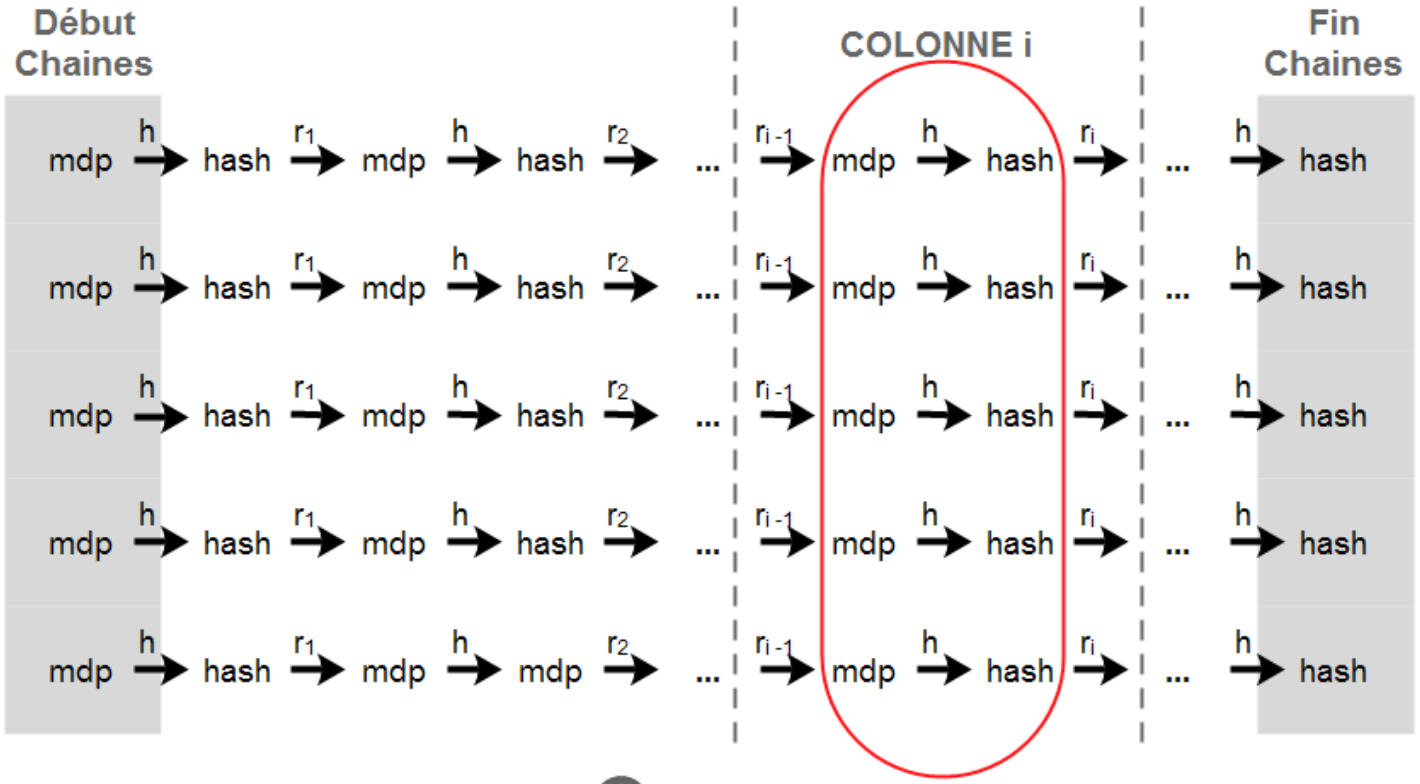
Limites des tables Rainbow et dépassement par l'utilisation de méthodes probabilistes optimisées

Résumé

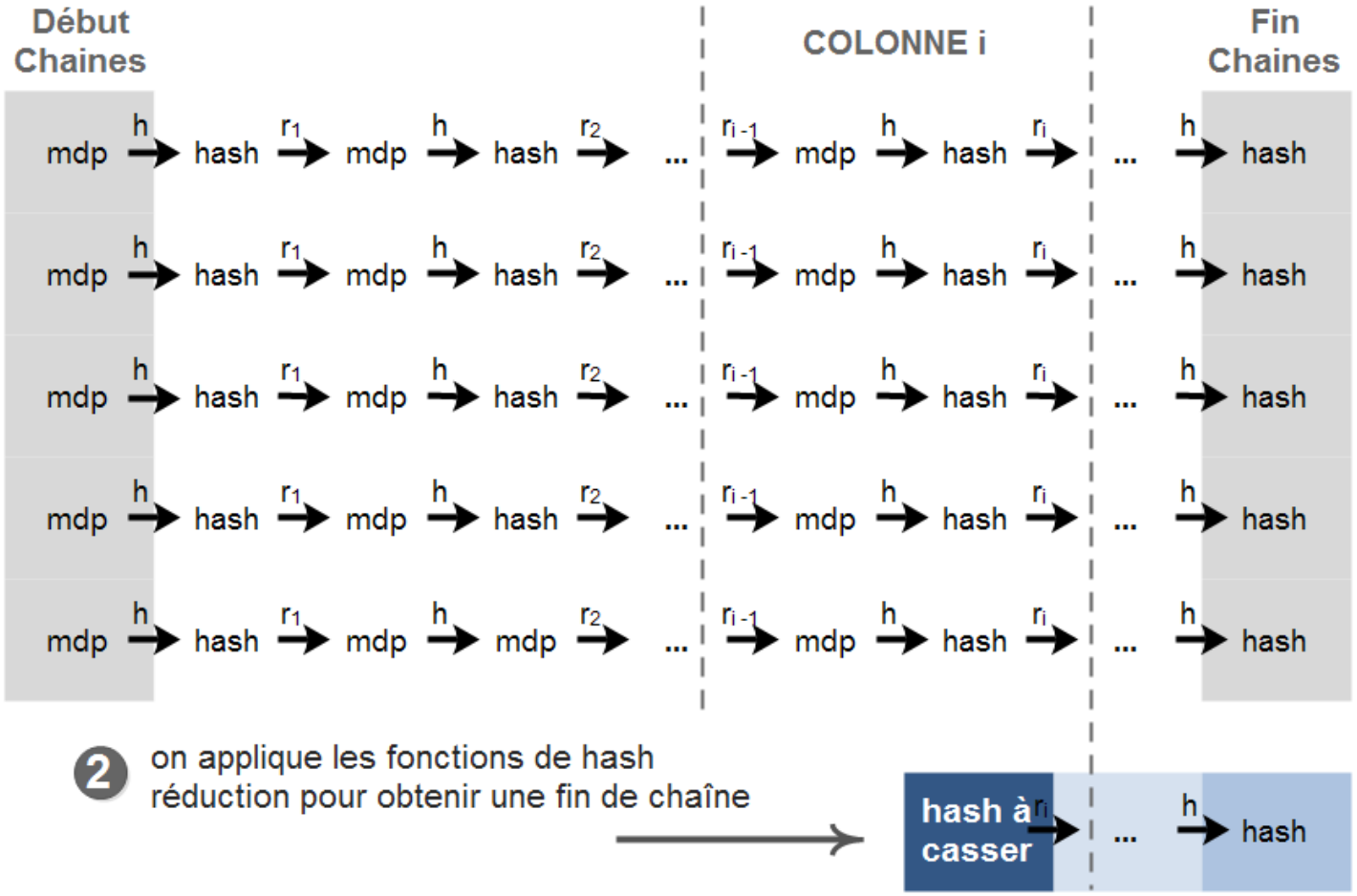
- Fonctionnement des tables Rainbow
- Limites des tables actuelles
- Méthodes probabilistes
- Optimisation
- Comparaison et conclusion

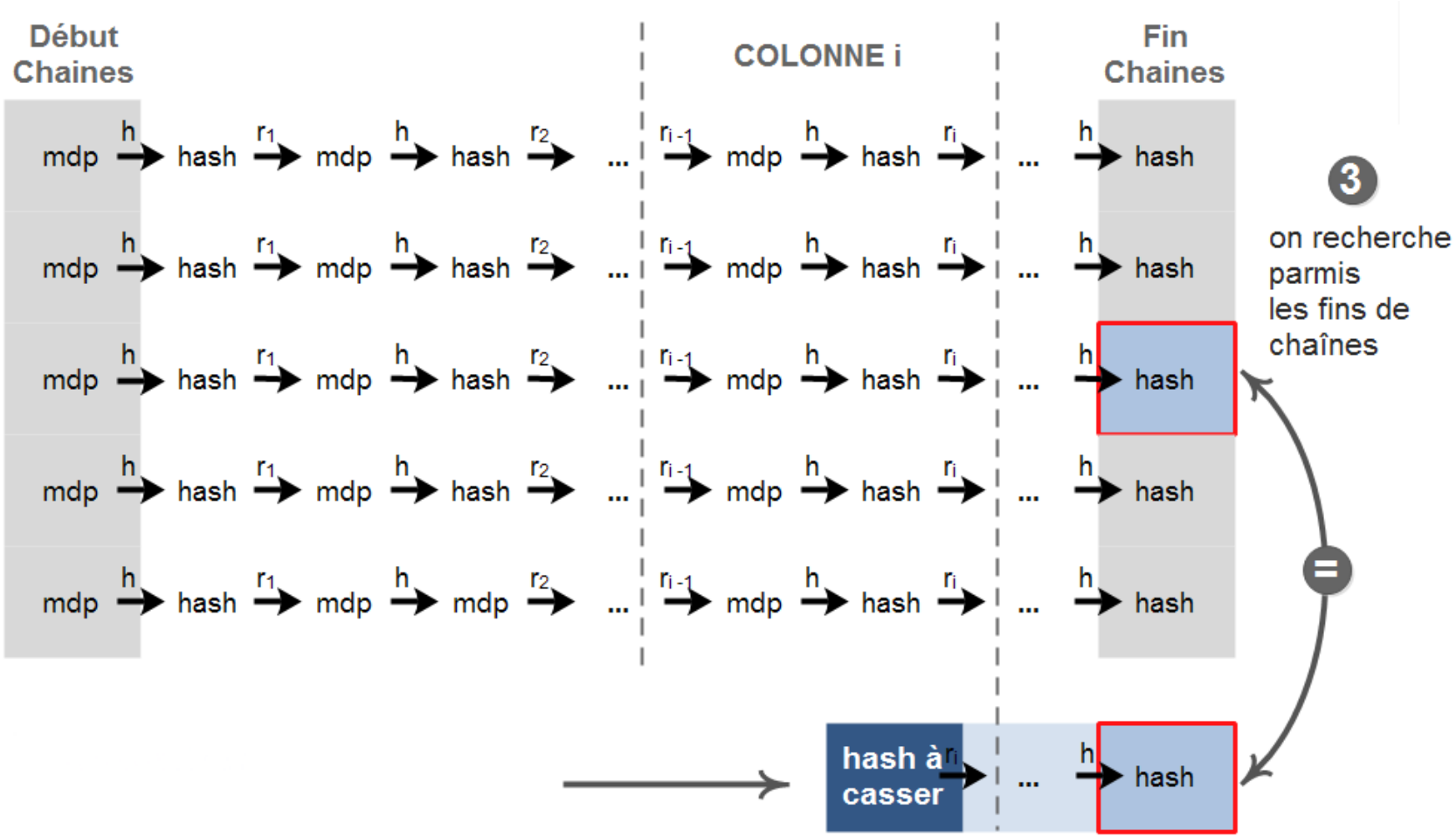
Fonctionnement des tables Rainbow

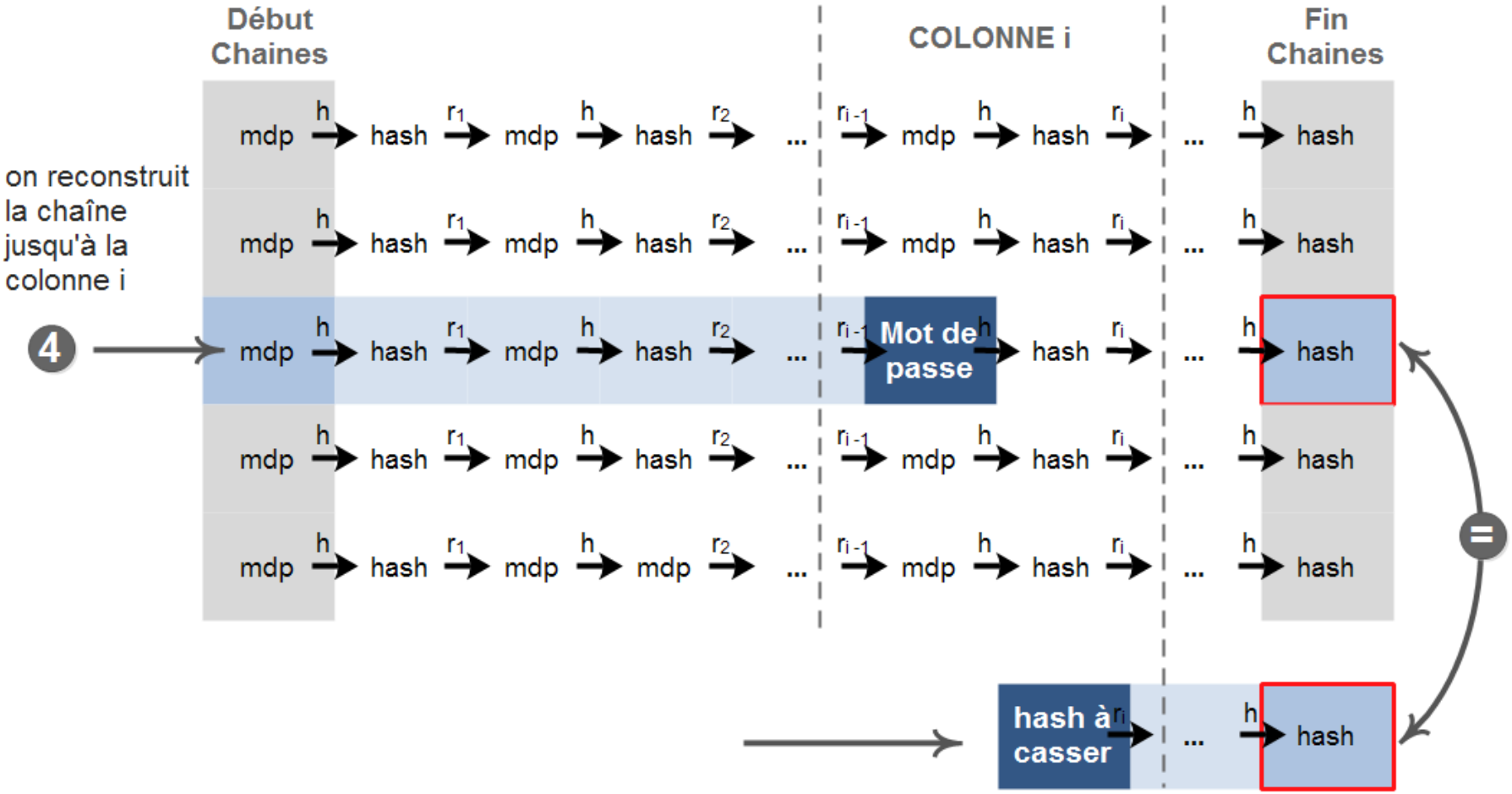

COLONNE i
m lignes
mdp
h
hash
r1
r2
ri -1
ri
...
Début
Chaines
t colonnes
Fin
Chaines



1 on suppose que le mot de passe recherché est contenu dans la colonne i


Début Chaines
COLONNE i
Fin Chaines
mdp
h
hash
r1
r2
ri -1
ri
...
2
on applique les fonctions de hash
réduction pour obtenir une fin de chaîne
hash à casser


Début Chaines
COLONNE i
Fin Chaines
mdp
h
hash
r1
r2
ri -1
ri
...
3
on recherche parmis les fins de chaînes
=
hash à casser


Début Chaines
COLONNE i
Fin Chaines
on reconstruit la chaîne jusqu'à la colonne i
4
mdp
h
hash
r1
r2
ri -1
ri
...
Mot de passe
hash à casser
=

Limites des tables actuelles

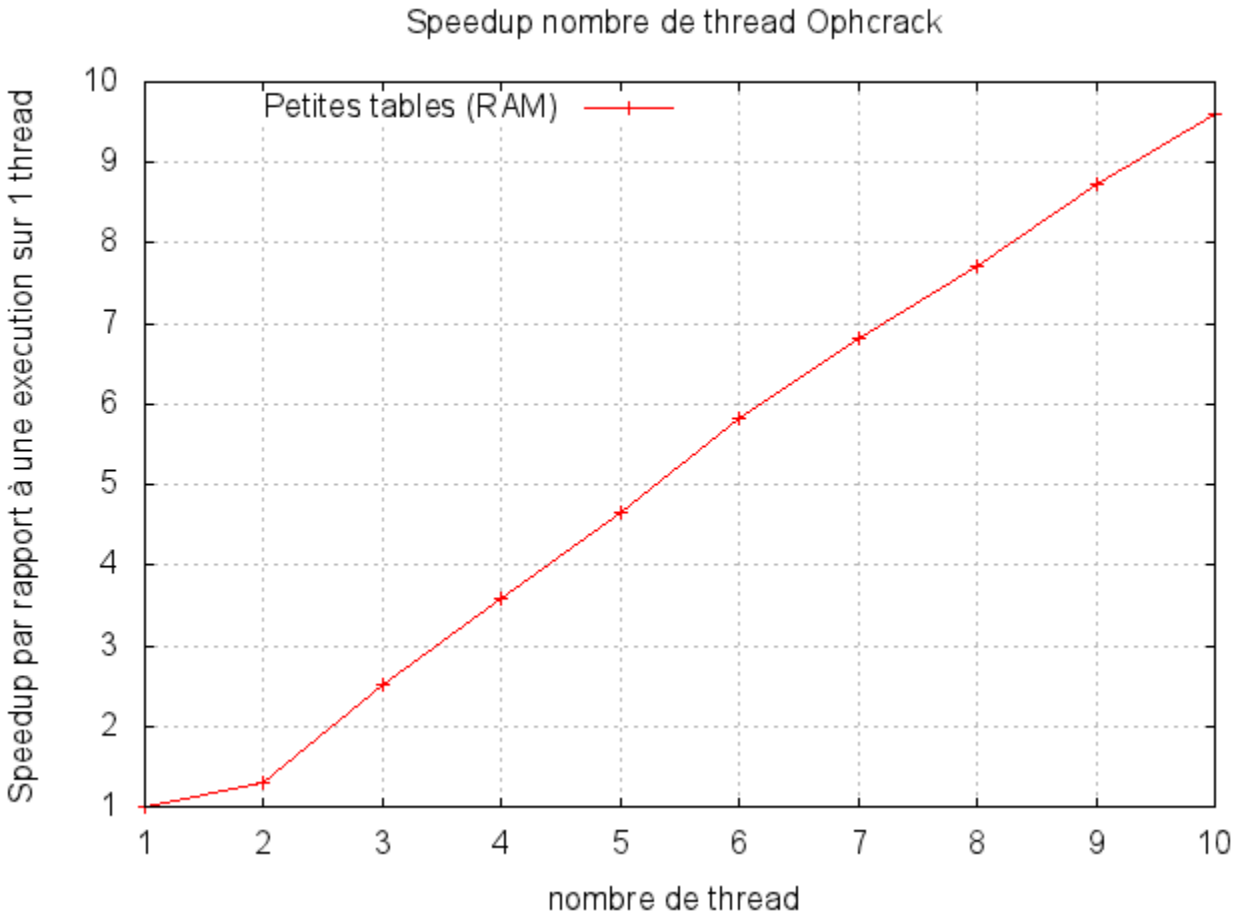
Cas des tables du logiciel Ophcrack

Tables «classiques»:

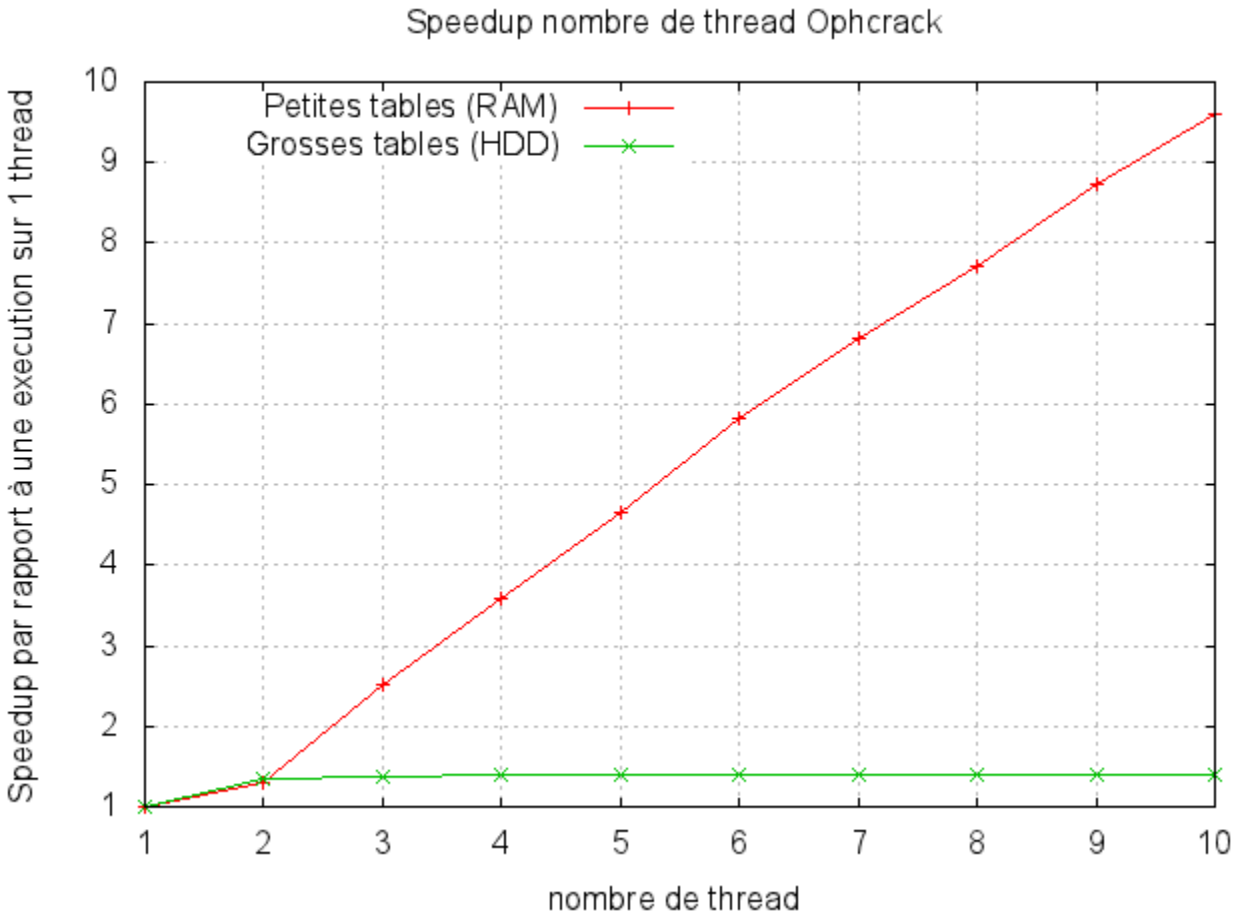
- Mot de passe de longueur 8
- Alphabet d’une centaine de caractères
- 99 % de taux de succès

- 6 mois de calcul pour trois cartes HD6990
- 2 To

Influence des accès disque



Influence des accès disque



Pistes de réflexions

- Puissance de calcul non utilisée lors du cassage
- Comment considérer des mots de passe de taille plus grande?

«est-il possible d'obtenir 99% succès sur des mots de passe réels de 8 caractères avec des tables plus petites?»

Méthodes probabilistes

Problème: définir une fonction de réduction qui permettent efficacement d'adresser les mots de passe les plus probables.

Méthodes probabilistes

On se base sur des ensembles de mots de passe réels lors des phases d'apprentissage et d'évaluation.

RockYou, Yahoo, Myspace, …

Méthode 1: les patterns

Un pattern est constitué d'une suite de blocs de caractères appartenant à une même catégorie.

Quatre catégories de caractères:

-lettres minuscules: '**L**'

-lettre majuscules: '**U**'

-chiffres: '**D**'

-caractères spéciaux: '**S**'

Passw0rd!

Passw0rd!

U_1 L_4 D_1 L_2 S_1

Sélection des patterns

Au terme de la phase d'apprentissage, pour chaque pattern i:

- P_i la probabilité du pattern i
- S_i la taille de l'espace engendré par le pattern i
- $E_i = P_i/S_i$ efficacité du pattern i

Quels patterns retenir pour maximiser le succès total tout en respectant une contrainte de taille?

Sélection des patterns

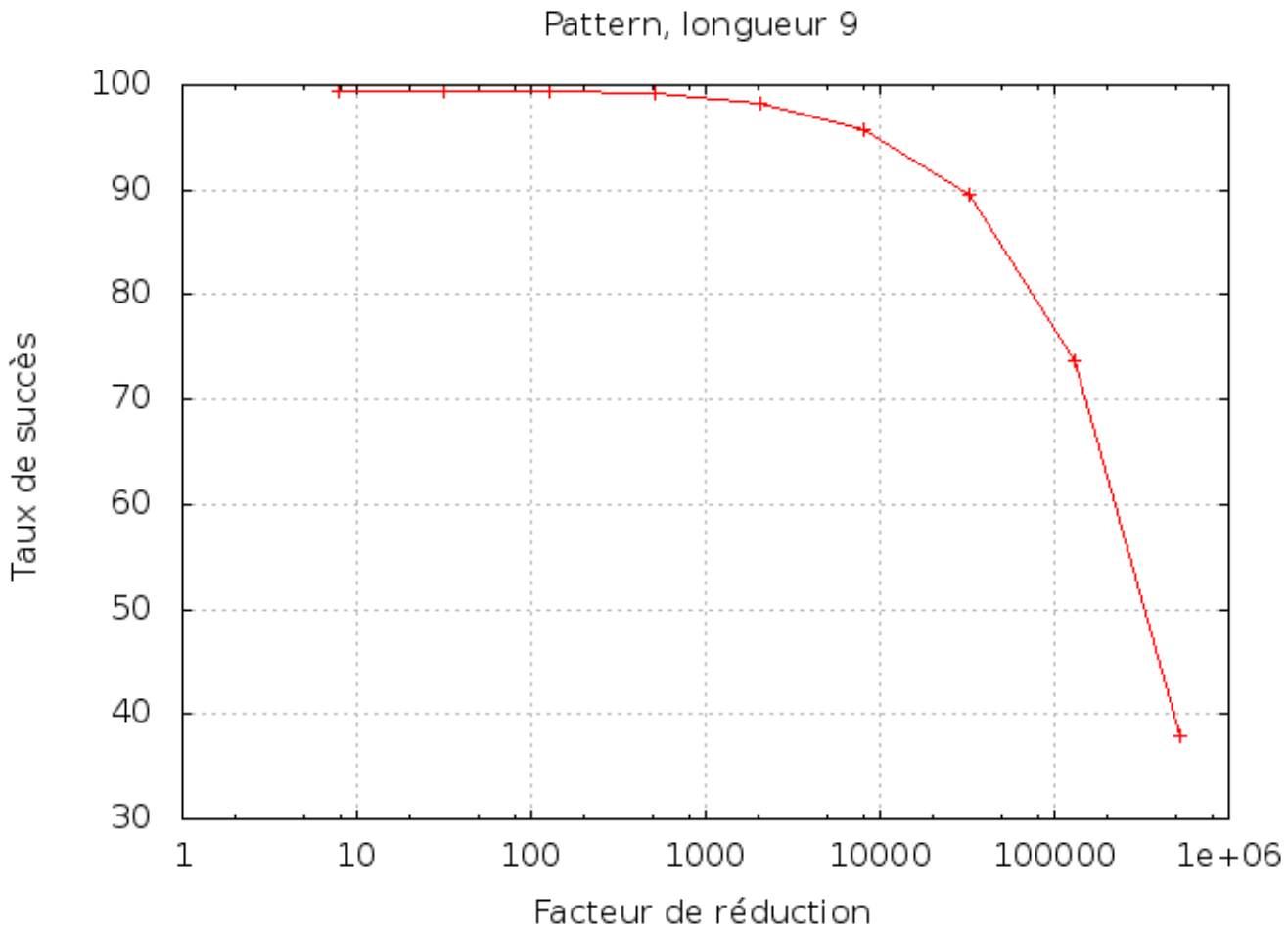
Au terme de la phase d'apprentissage, pour chaque pattern i:

- P_i la probabilité du pattern i
- S_i la taille de l'espace engendré par le pattern i
- $E_i = P_i/S_i$ efficacité du pattern i

Quels patterns retenir pour maximiser le succès total tout en respectant une contrainte de taille?

Algorithme du sac à dos: heuristique de l'algo glouton

Quelques résultats



Méthode 2: Modèle de Markov

«*Les utilisateurs utilisent des mots de passe phonétiquement proche de leur langue*»

Cette propriété peut-être exploitée par l'utilisation de chaînes de Markov.

Couramment utilisées dans le domaine de cassage de mots de passe.

Présentation au SSTIC 2011 dans le cadre tables Rainbow

On considère les ordres de Markov suivants:

- **Ordre 0**: la probabilité d'un caractère est indépendante des caractères précédents.
- **Ordre 1**: la probabilité d'un caractère dépend du caractère précédent.
- **Ordre 2**: la probabilité d'un caractère dépend des deux caractères précédents.

improba = -10 . log(proba)

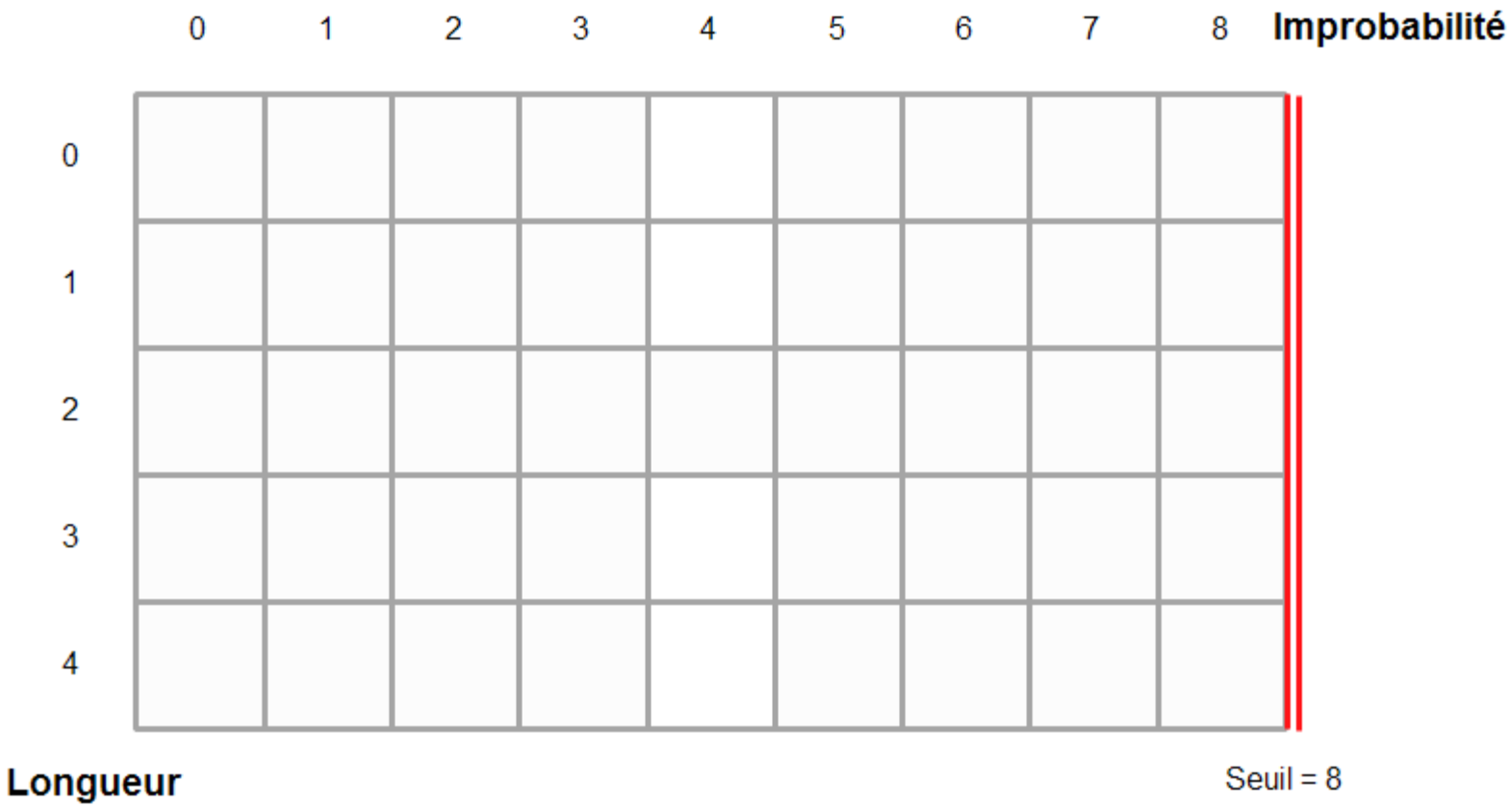
Quand utiliser le modèle de Markov?

Conjointement aux patterns pour réduire les blocks de lettres.

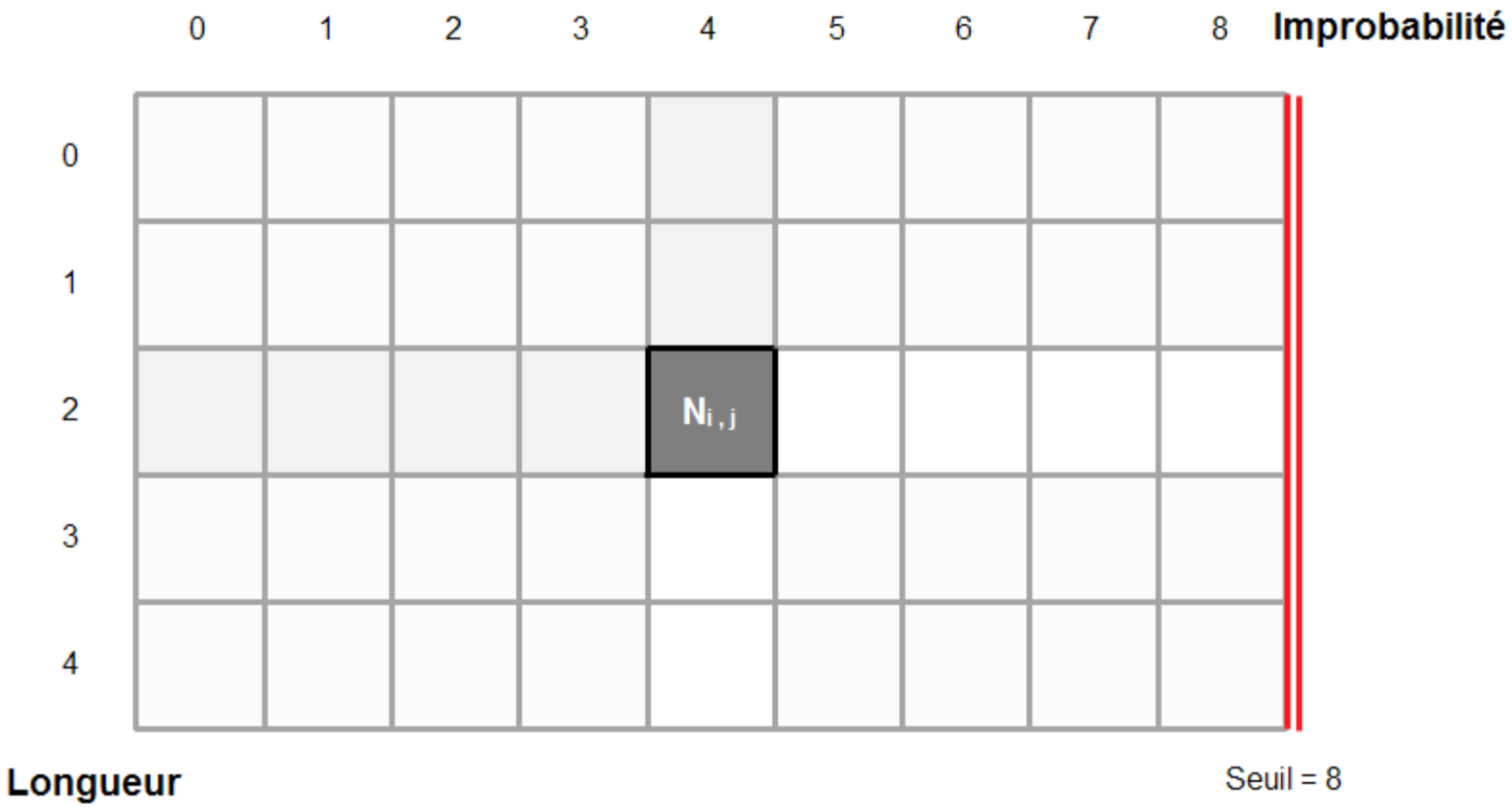
Cependant il n’est pas intéressant d’utiliser Markov systématiquement.

Algorithme du sac à dos à choix multiples.

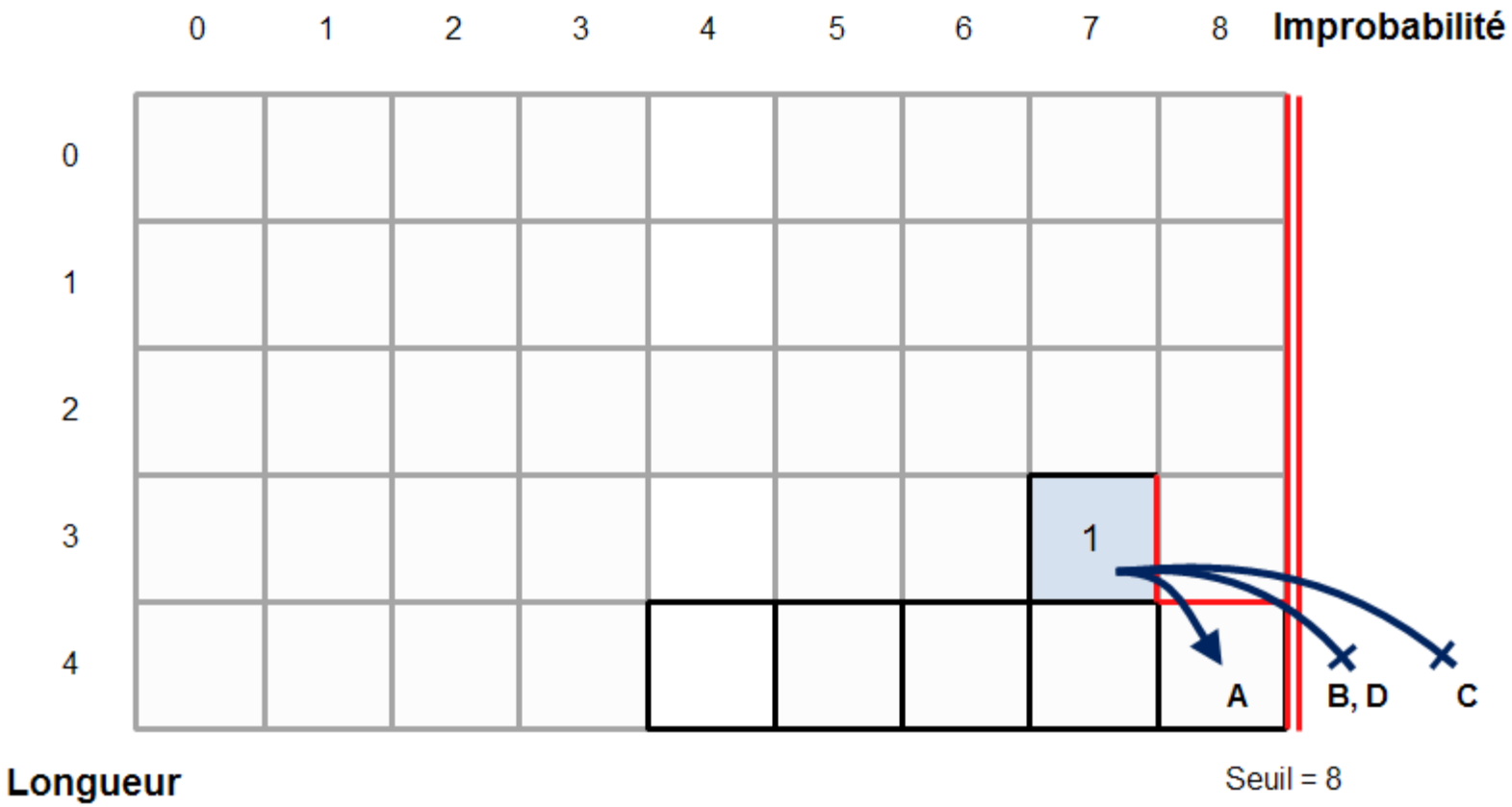
Indexation Phase 1



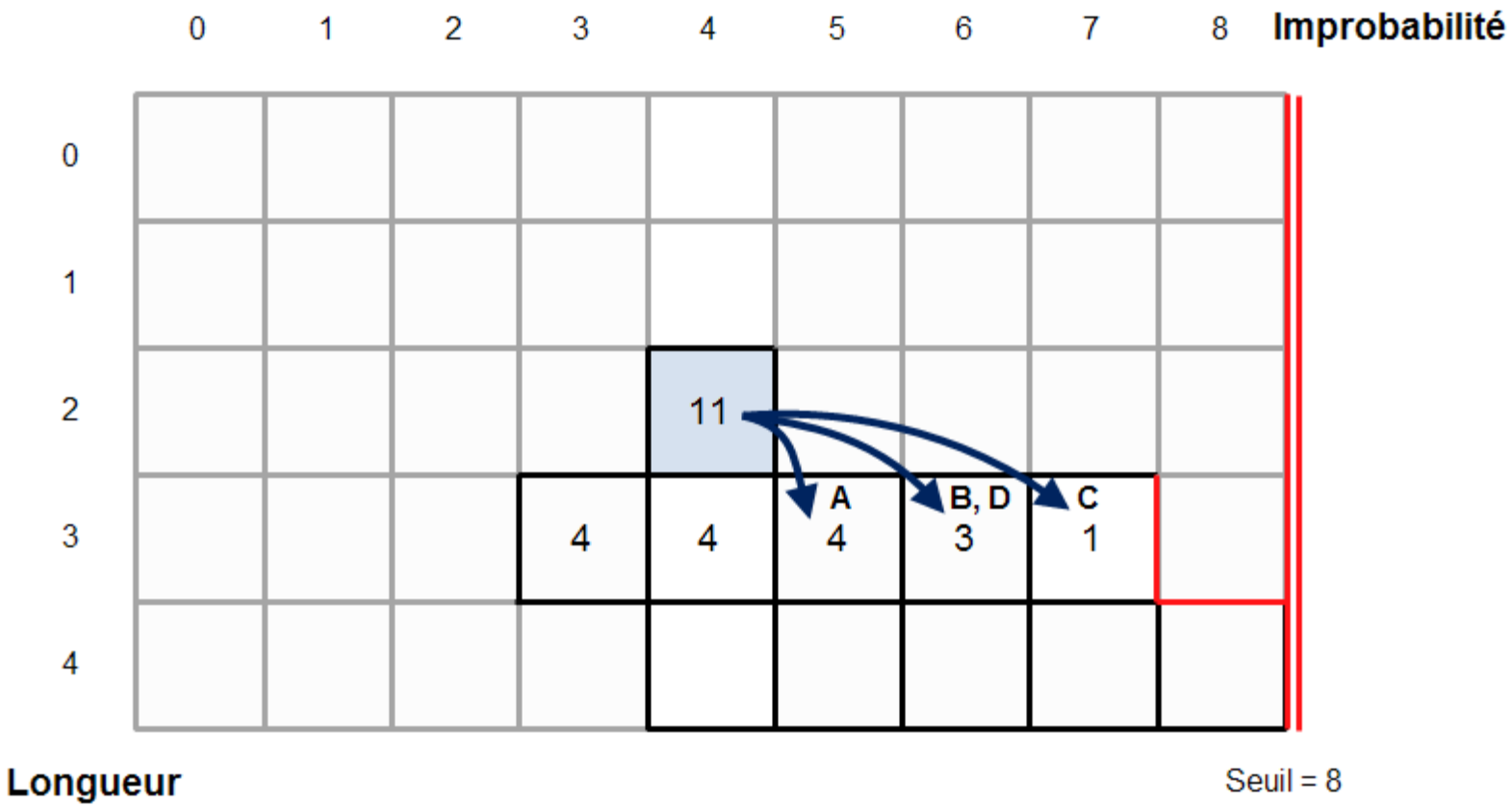
Alphabet = {A, B, C, D} improbabilités respectives: {1, 2, 3, 2}



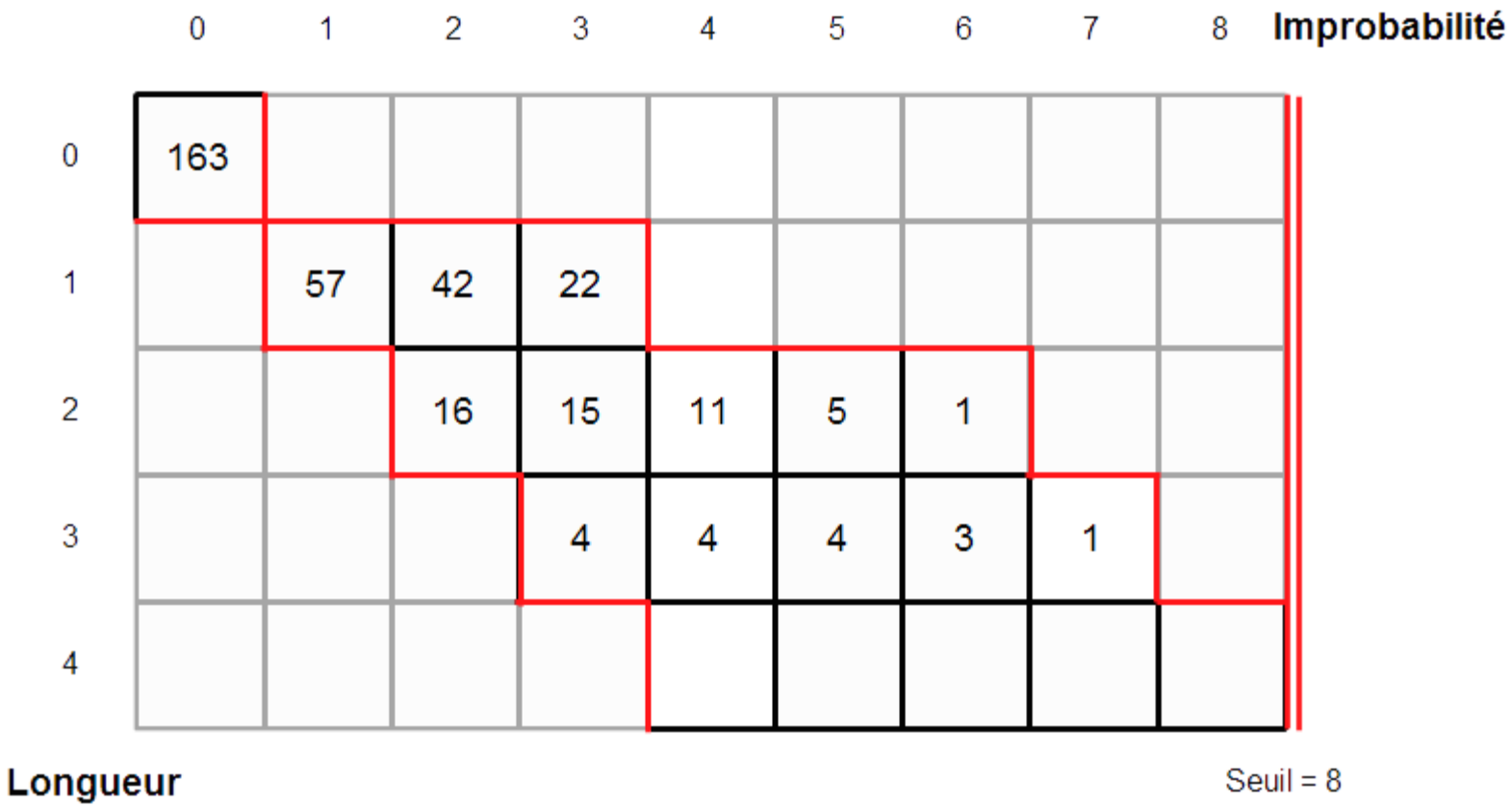
$N_{i,j}$: nombre de combinaisons satisfaisant une improbabilité *j* sur les *i* premiers caractères.



Alphabet = {A, B, C, D} improbabilités respectives: {1, 2, 3, 2}

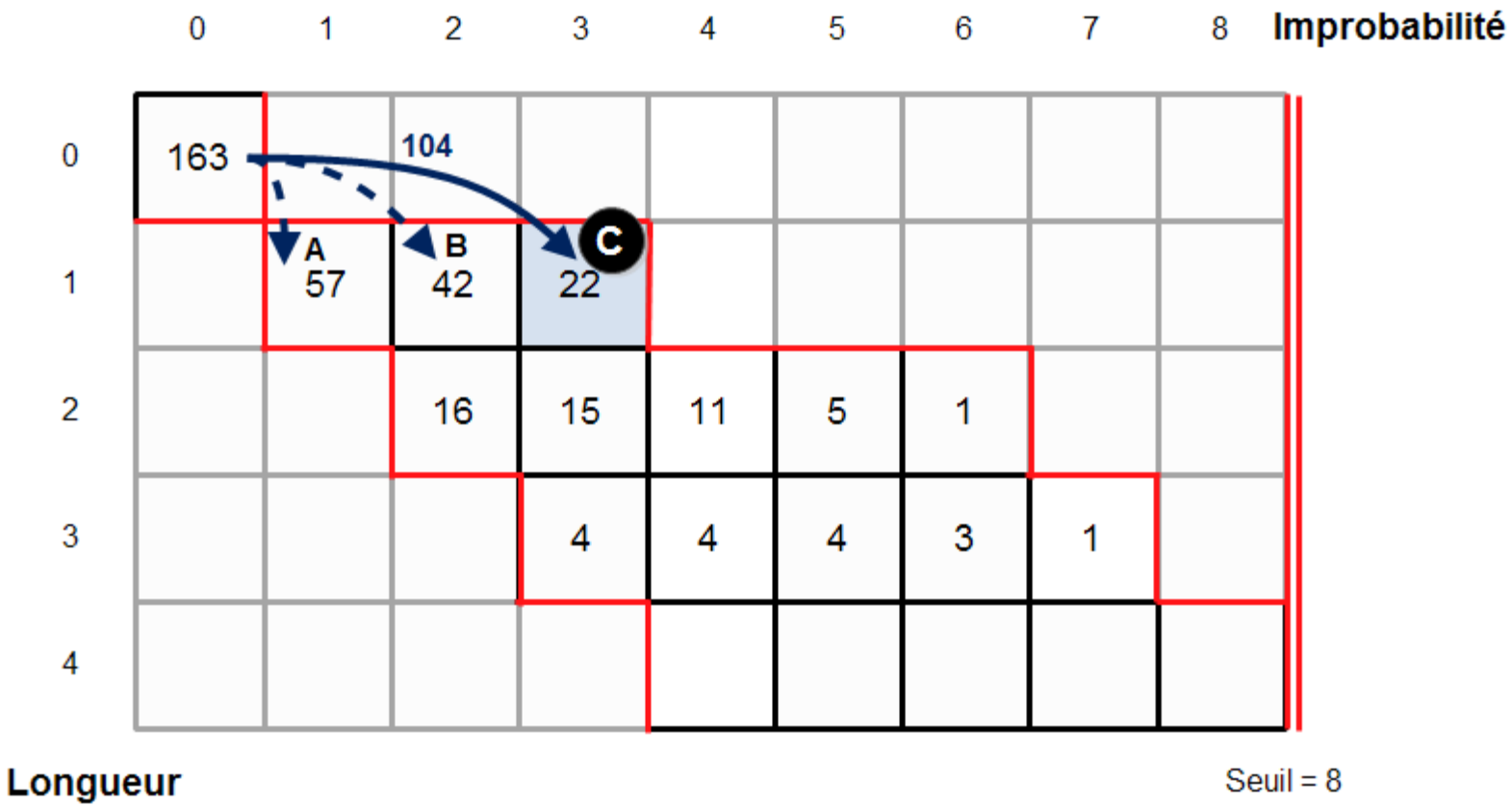


Alphabet = {A, B, C, D} improbabilités respectives: {1, 2, 3, 2}

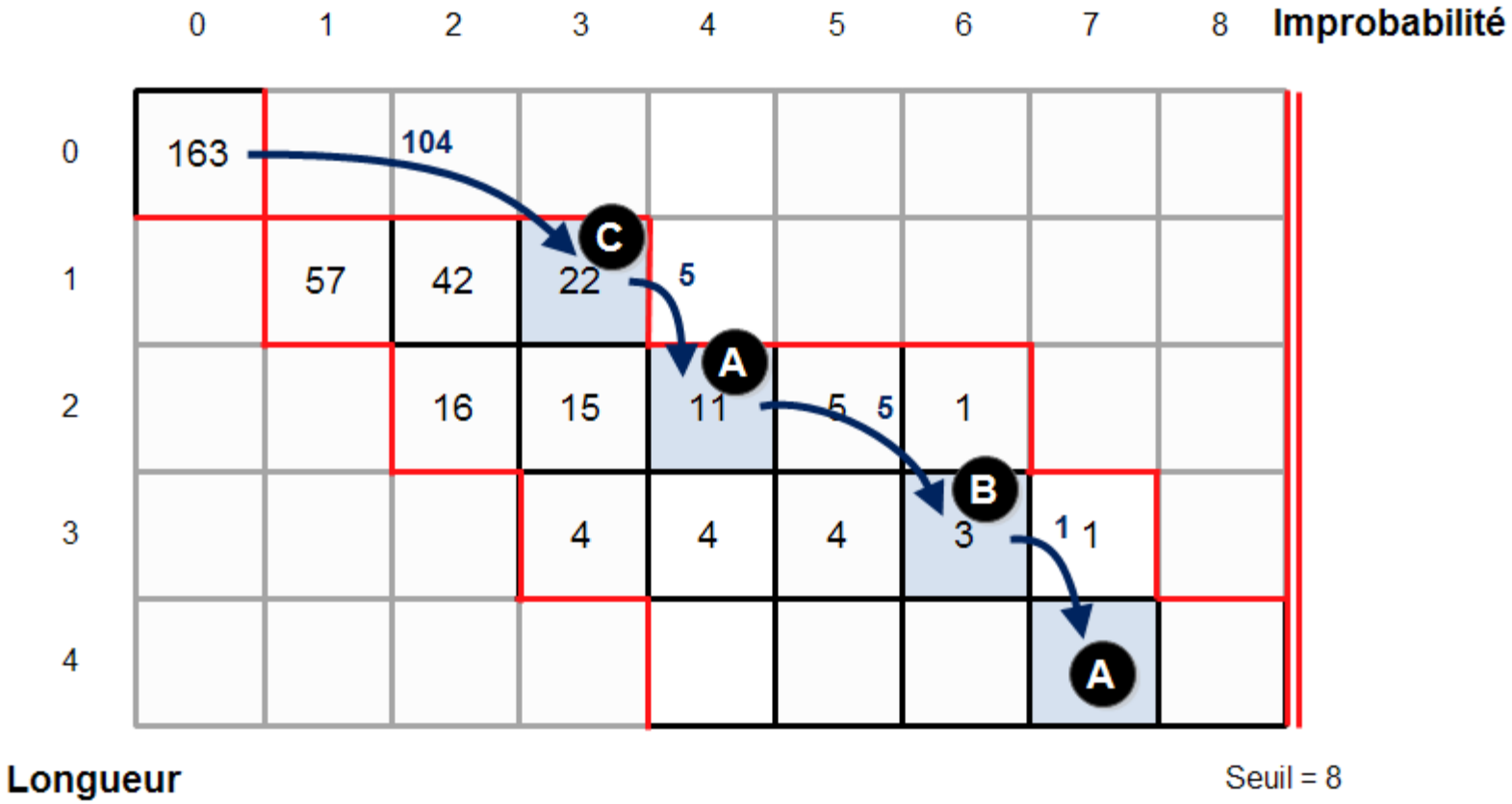


Alphabet = {A, B, C, D} improbabilités respectives: {1, 2, 3, 2}

Indexation Phase 2

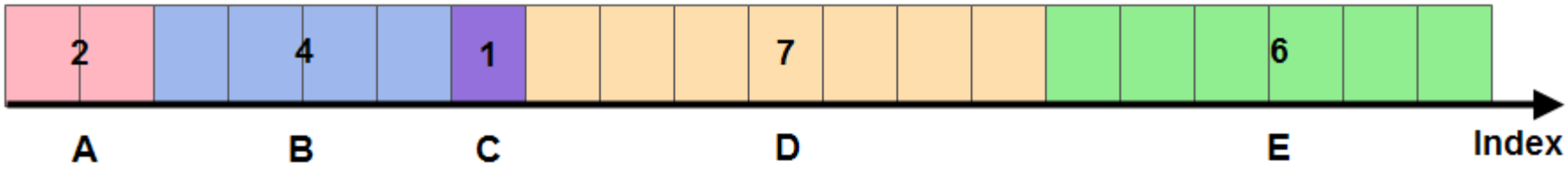


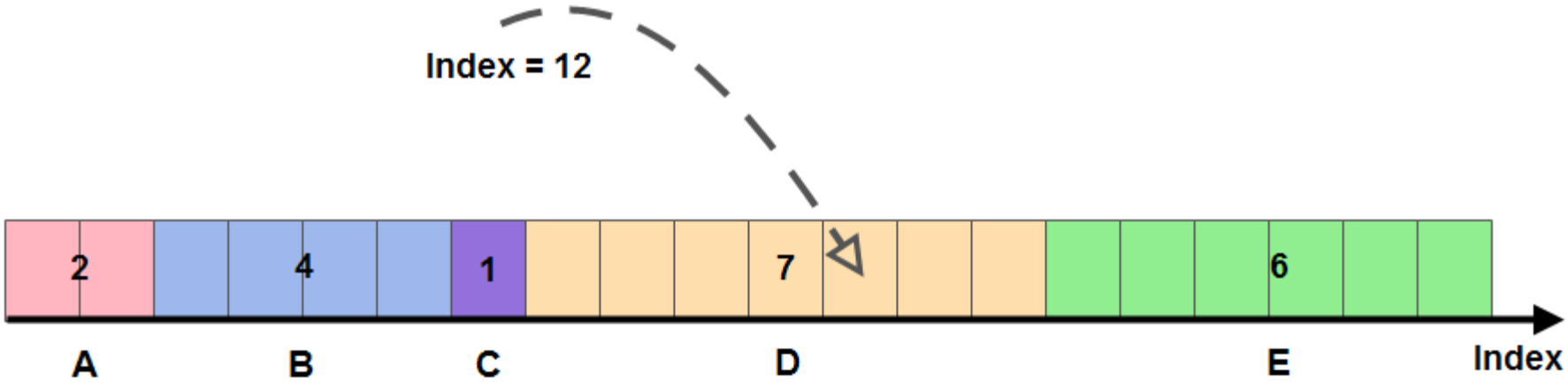
Recherche du mot de passe d'index: **104**



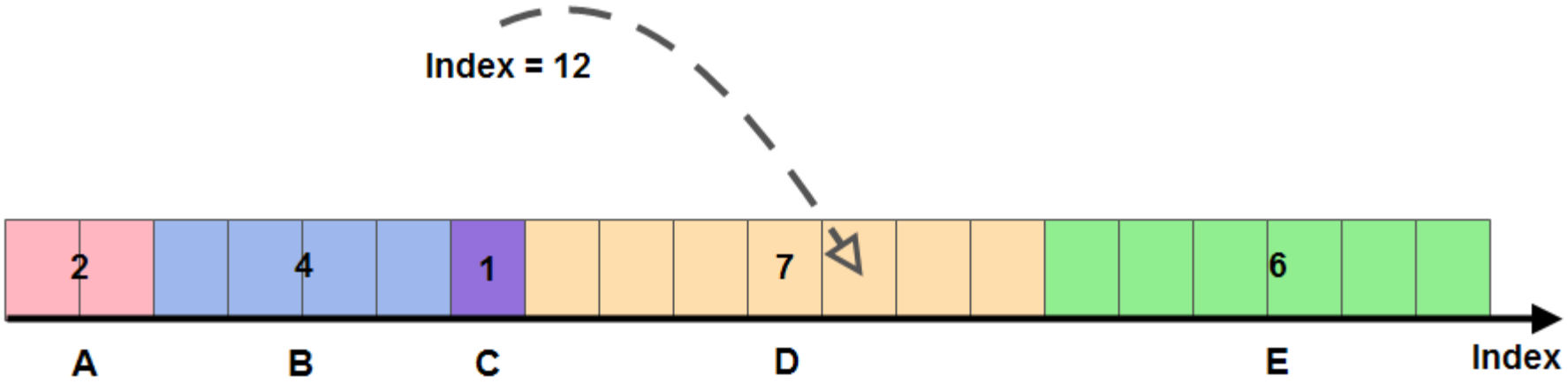
Mot de passe d'index 104: « **CABA** »

Optimisation





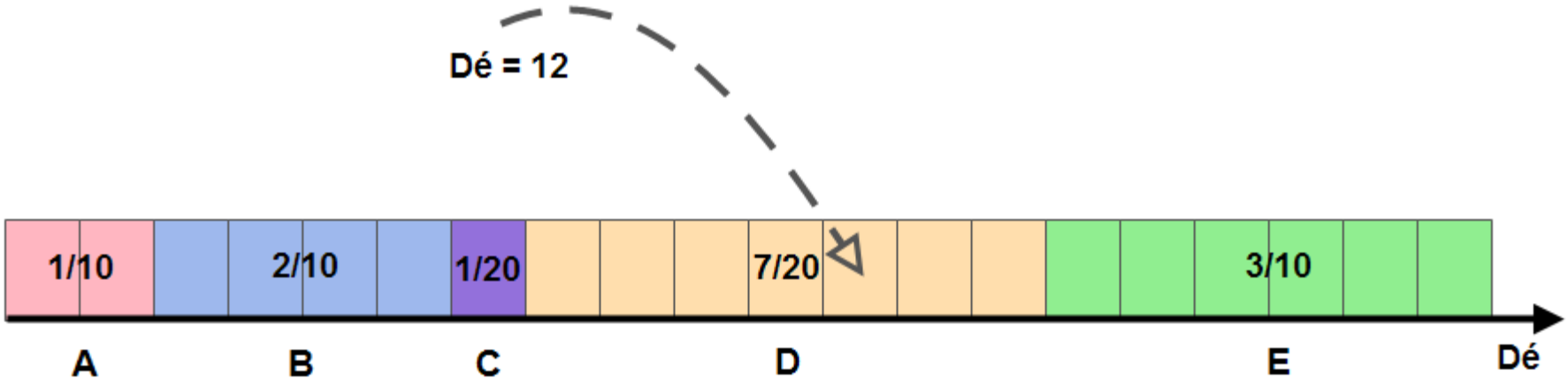

```
for char in alphabet:
    if index <  borne[char]:
        return char
```



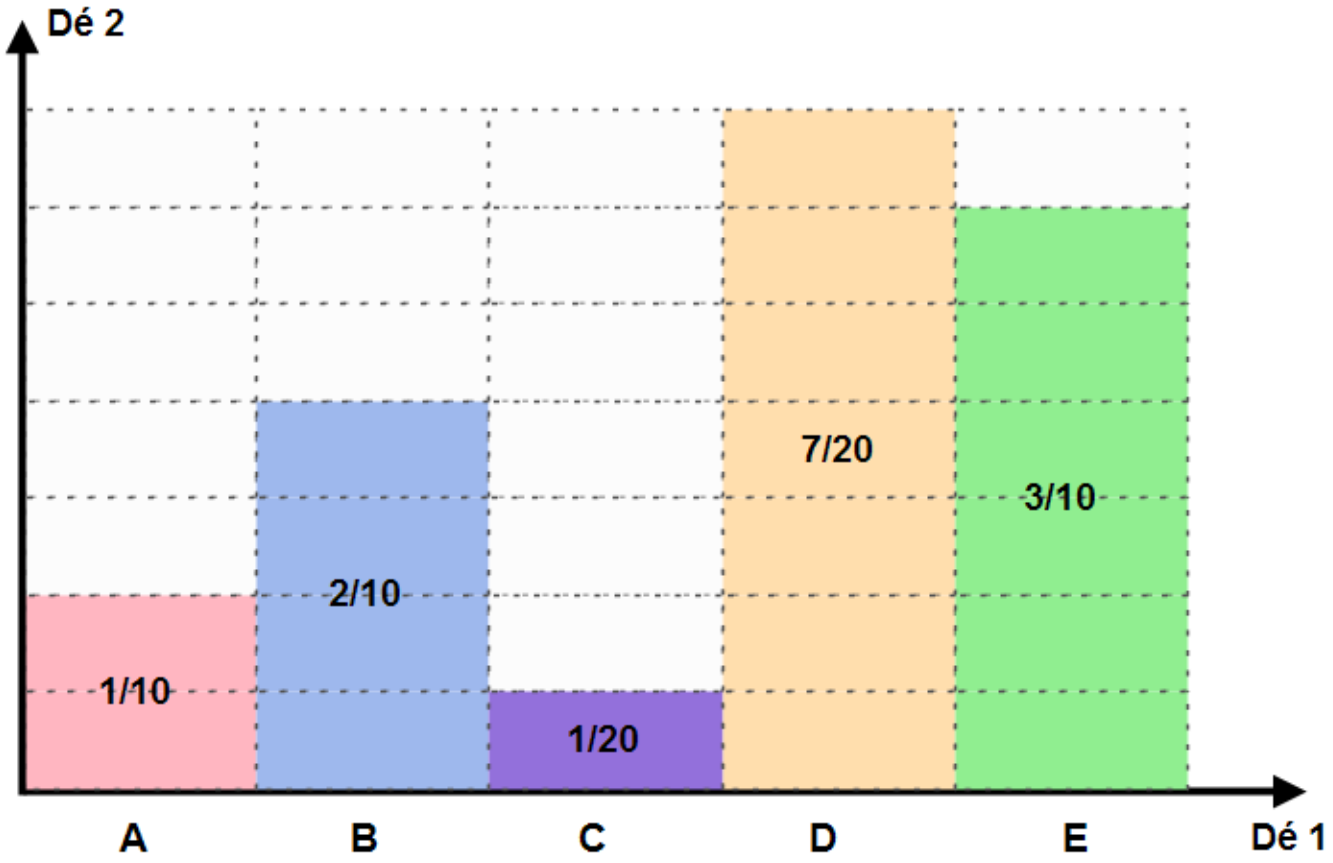

```
for char in alphabet:
    if index < borne[char]:
        return char
```

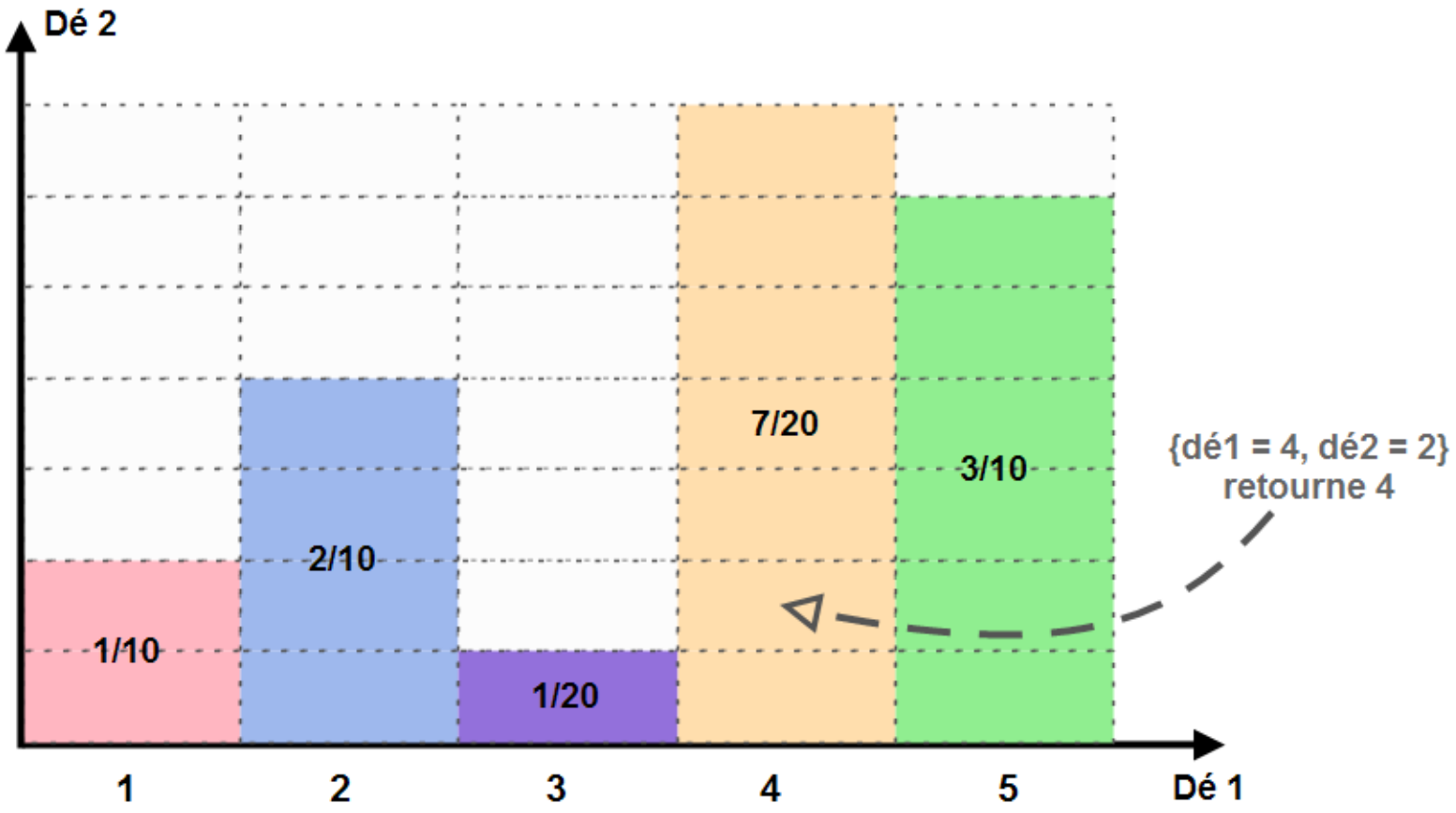
pas adapté à une implémentation GPU

Nouvelle approche: génération d'un dé biaisé à *n* faces

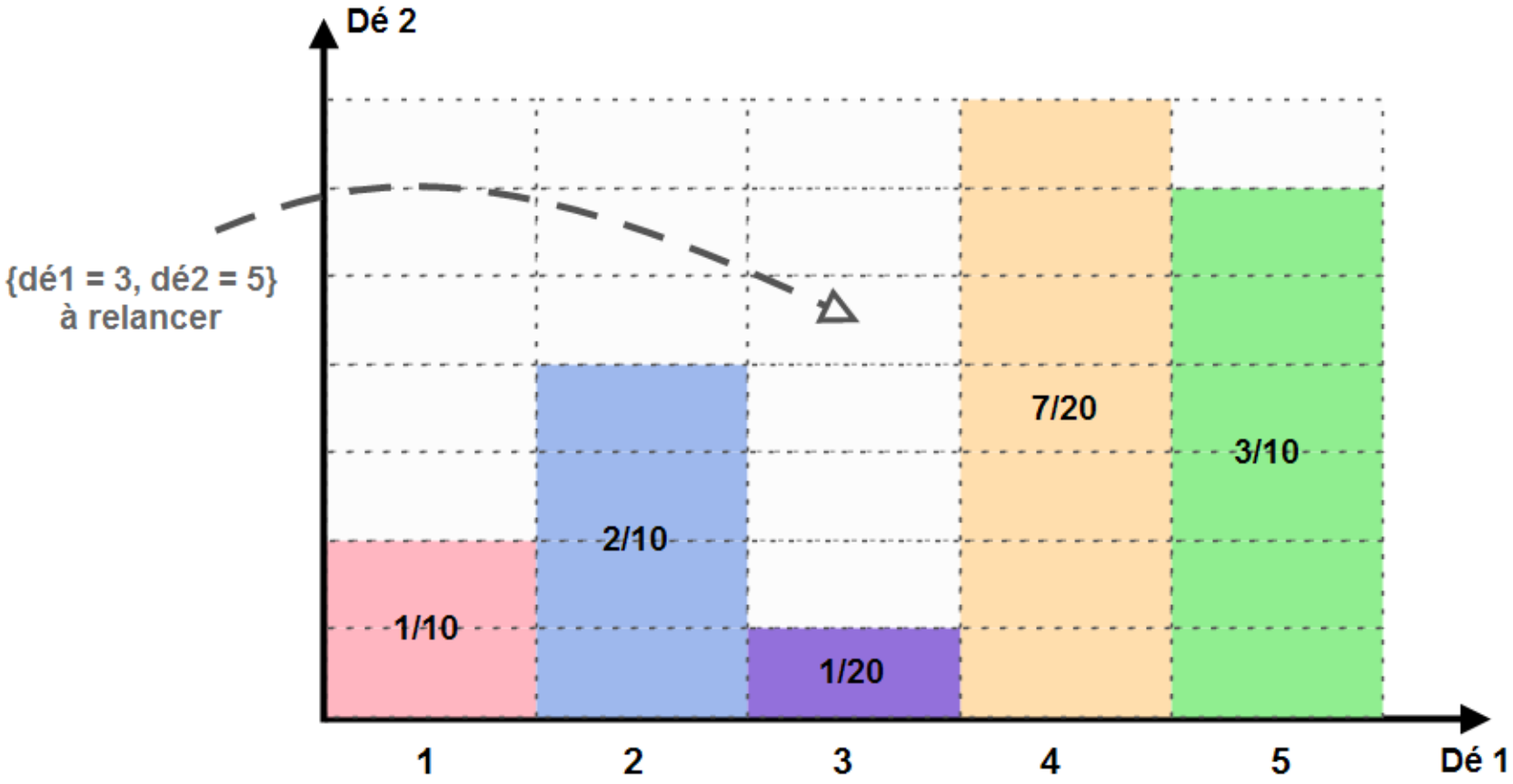



```
for value in interval:
    if dé <  borne[value]:
        return value
```

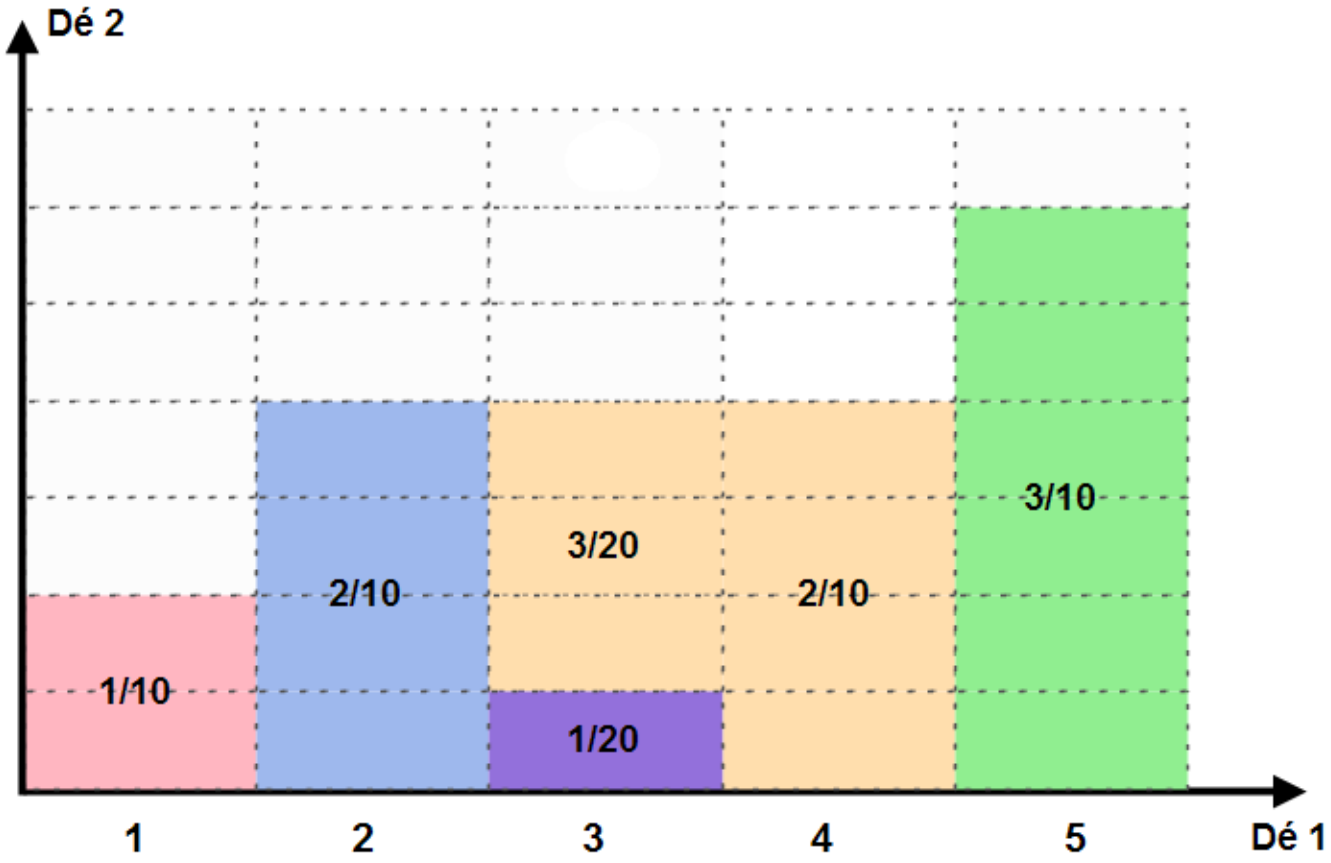

Dé 2
7/20
3/10
2/10
1/10
1/20
A
B
C
D
E
Dé 1




```
while True:
    if de2 < prob[de1]:
        return  de1
    (de1, de2) = nouveau_lancé()
```

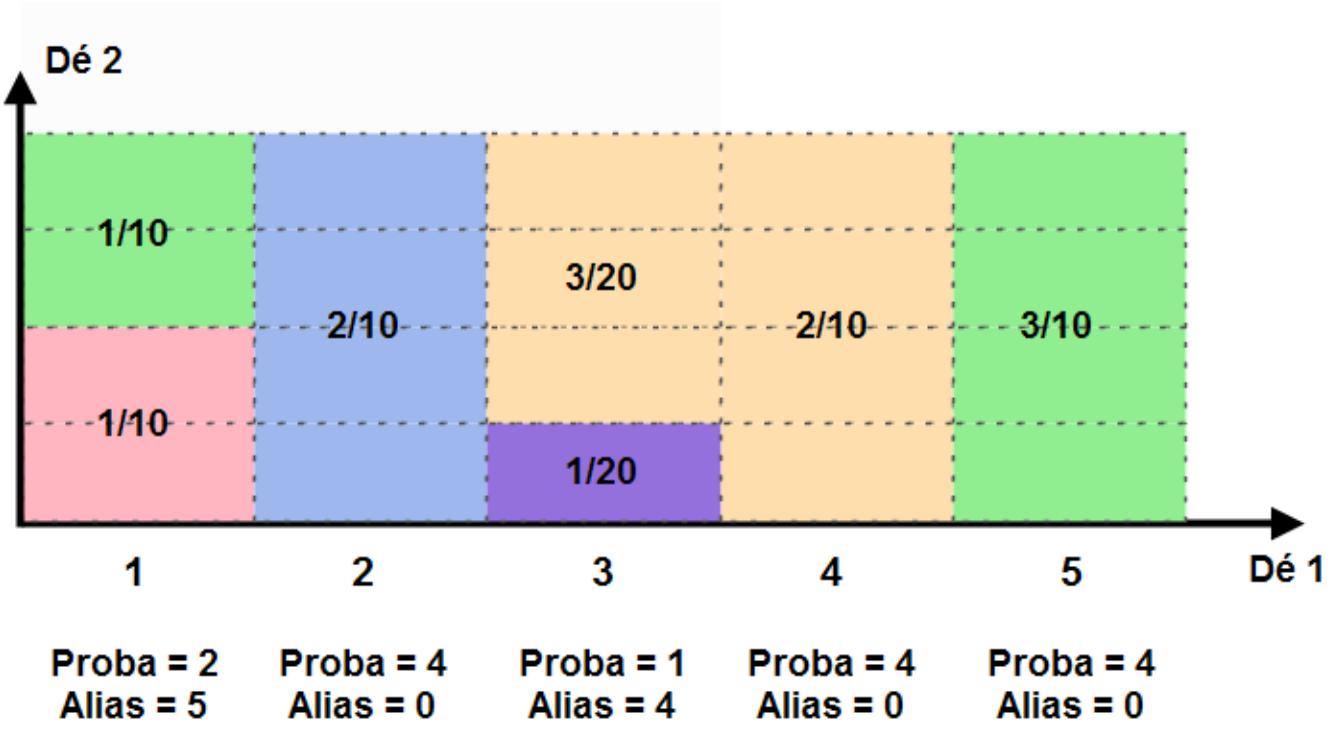



```
while True:
    if de2 < prob[de1]:
        return  de1
    (de1, de2) = nouveau_lancé()
```

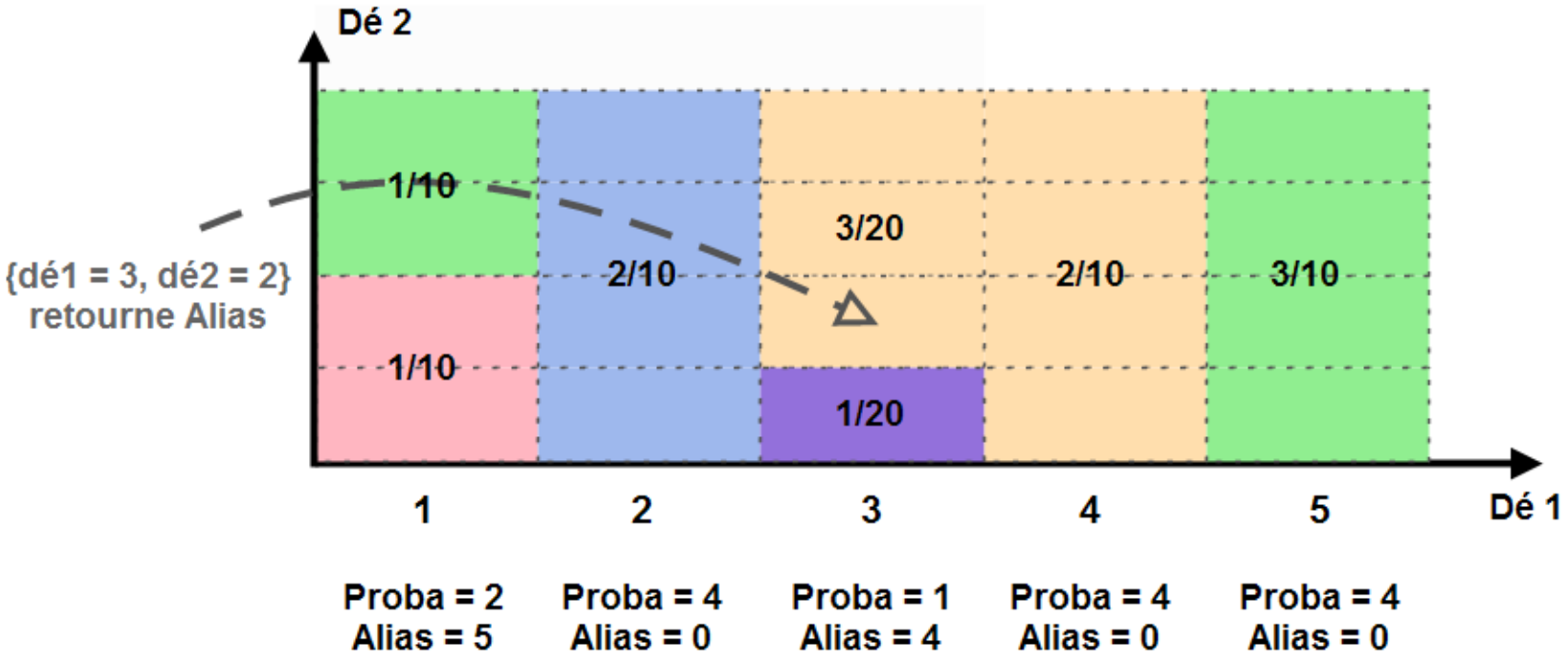

Dé 2
1/10
2/10
3/20
1/20
2/10
3/10
1
2
3
4
5
Dé 1

Deux règles de découpage:

- au plus deux rectangles par colonne
- la colonne *i* débute avec une partie du rectangle *i*



```
if de2 < prob[de1]:
    return de1
else:
    return alias[de1]
```



Cas concrets et résultats

- Tables pour un live CD: 600 Mo
- Tables de 60 Go

	RockYou	Yahoo	MySpace	Online
Tables classiques	70.4%	56.8%	79.0%	47.5%
Live CD	74.8%	62.2%	91.8%	30.6%
60 Go	85.6%	79.2%	96.8%	50.6%

Conclusions

- Les tables Rainbow semblent dans une impasse
- Les méthodes probabilistes décrites permettent de:
  - Réduire la taille des tables
  - Considérer des mots de passe plus longs

Vers un nouvel avenir pour les tables Rainbow?

http://ophcrack.sourceforge.net/

OS OBJECTIF SÉCURITÉ
Architecte de la sécurité informatique

Questions ?