Comment compter les moutons
sans perdre la mémoire

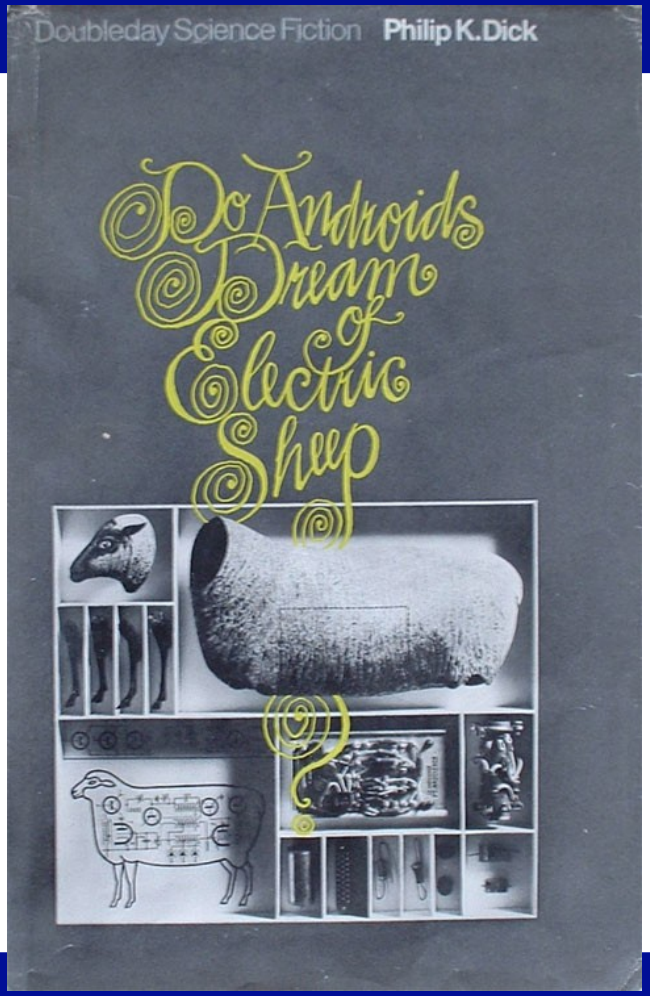
Image de catalyststuff sur Freepik

Benoît Masson

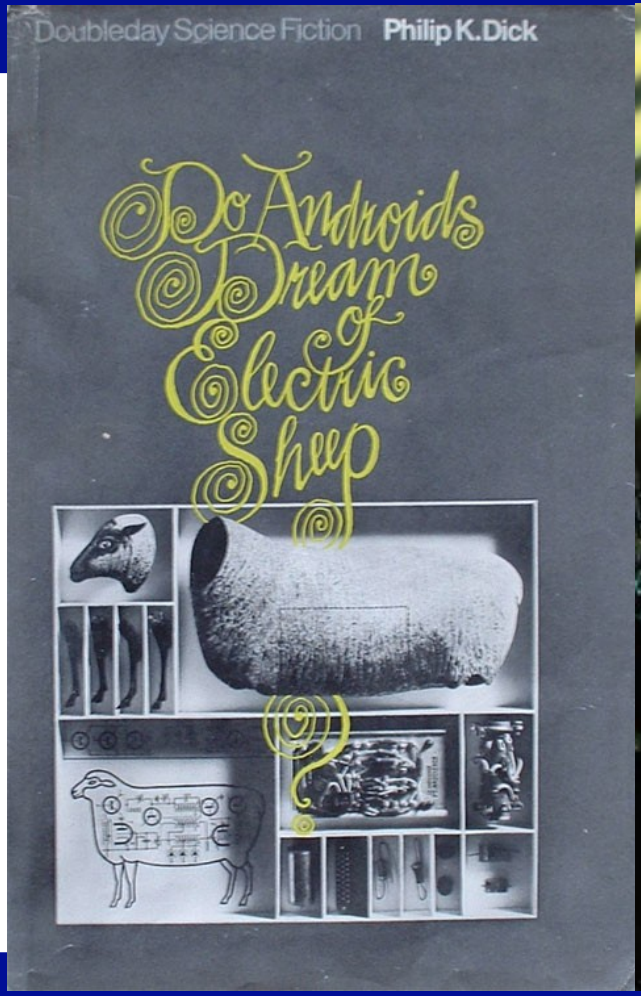
OVHcloud

Breizh C@mp La conférence à l'Ouest 2025 - 27 juin

Do Androids Dream of Electric Sheep?



Do Androids Dream of Electric Sheep?



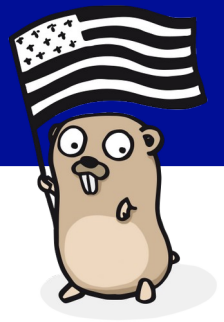
Comment compter les moutons sans perdre la mémoire

Benoît Masson

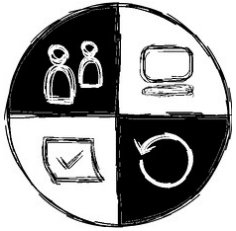
OVHcloud

BREIZHCAMP 2049

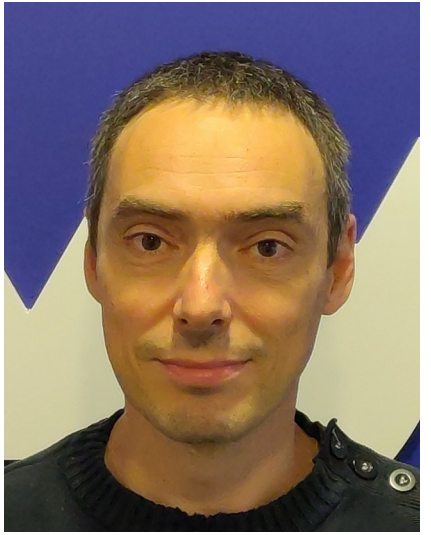
Générique



- Développeur Go depuis 2015
- Software Craftsman



@OVHcloud Rennes
Noms de Domaines depuis 2020



benoitmasson

Comptons les moutons

Image de catalyststuff sur Freepik

Décompte simple (voire naïf)

Décompte simple (voire naïf)

+ performant
- mémoire non bornée
- pas de résultats intermédiaires

Décompte itératif

Décompte itératif

- performant
- mémoire bornée
- visualisation en temps réel (*stream*)

Comptons les moutons (distincts)

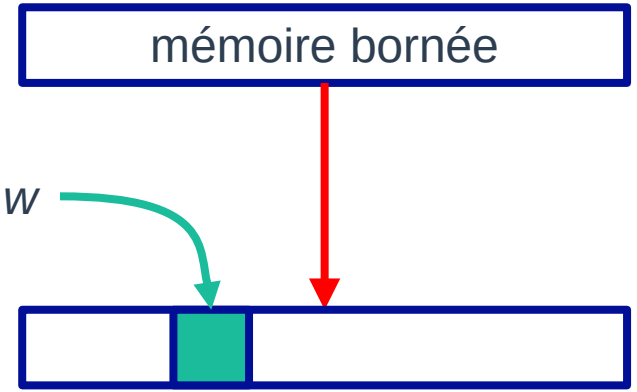
Image de catalyststuff sur Freepik

Décompte exact

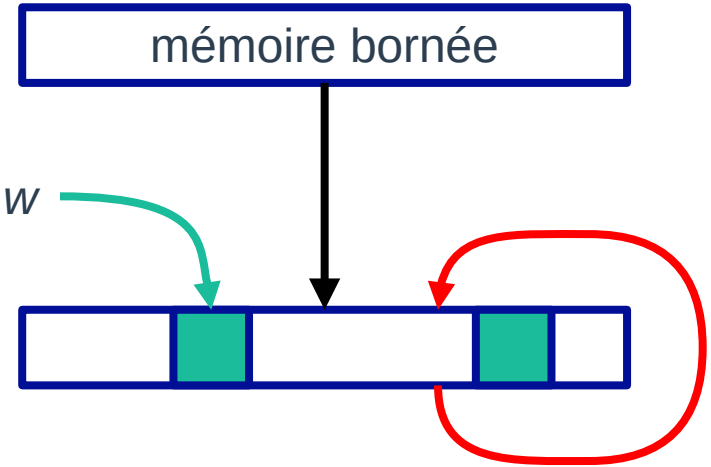
Décompte exact

+ performant
- mémoire non bornée

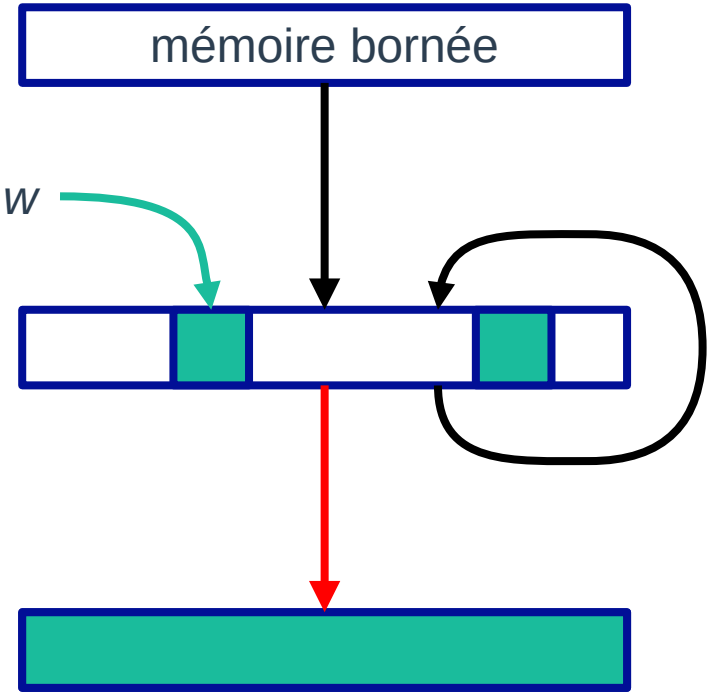
Décompte approximatif - Algorithme



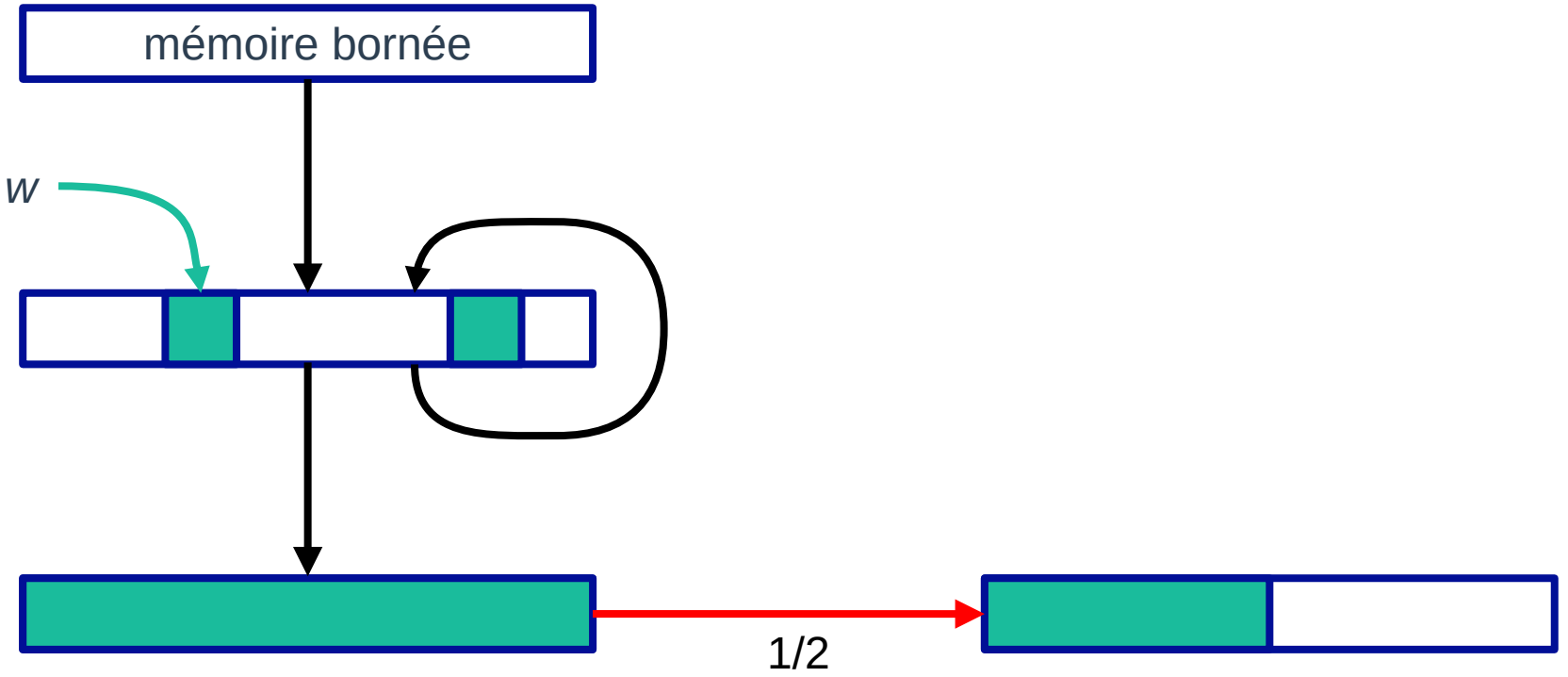
Décompte approximatif - Algorithme



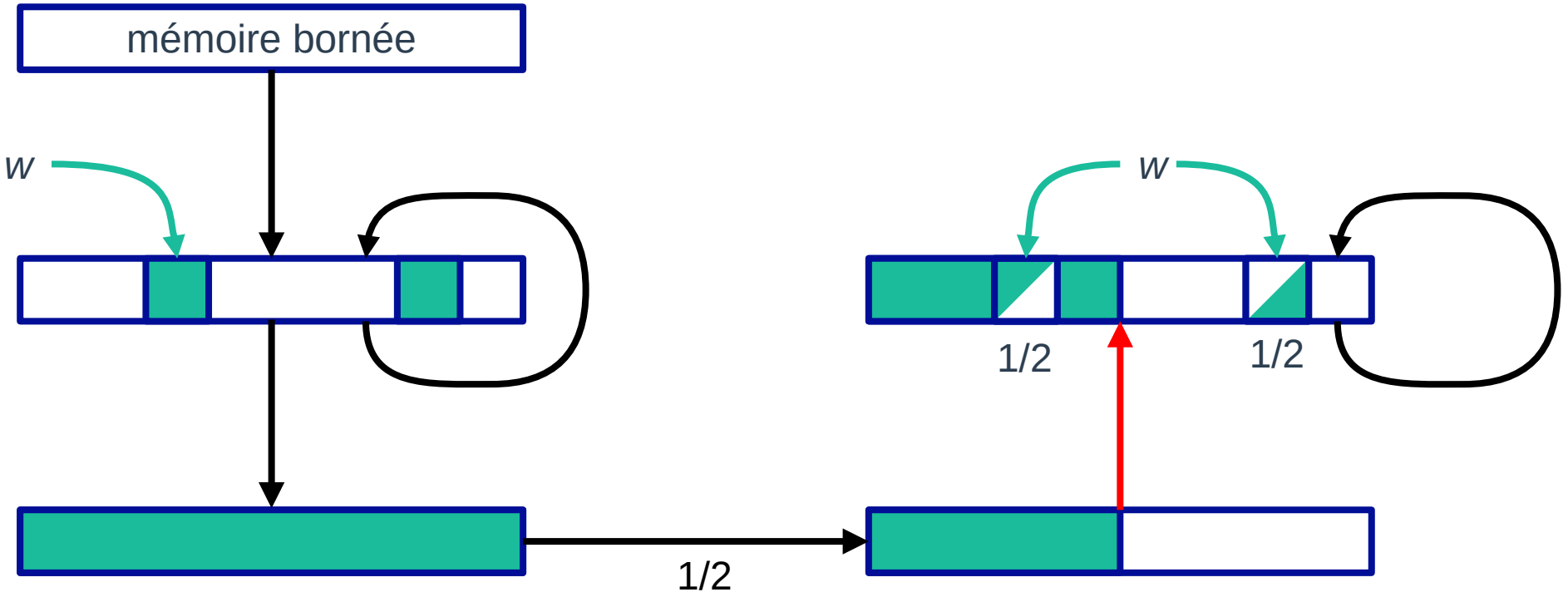
Décompte approximatif - Algorithme



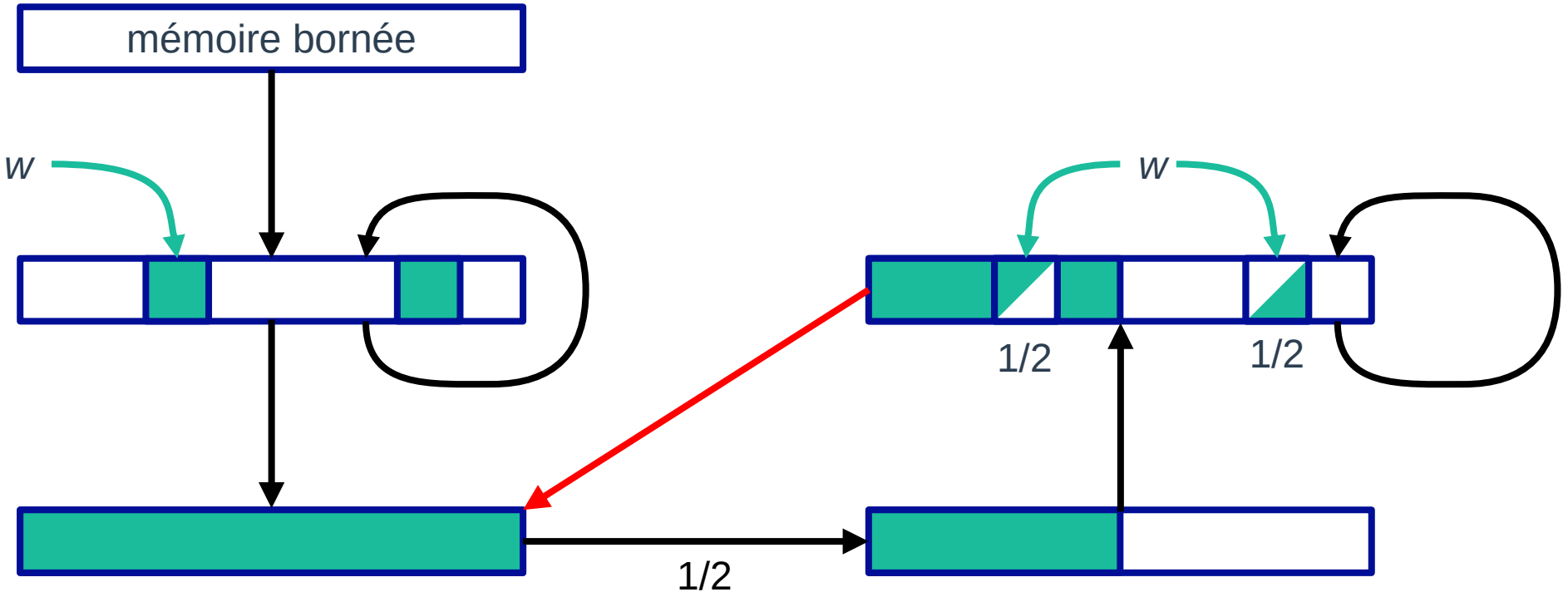
Décompte approximatif - Algorithme



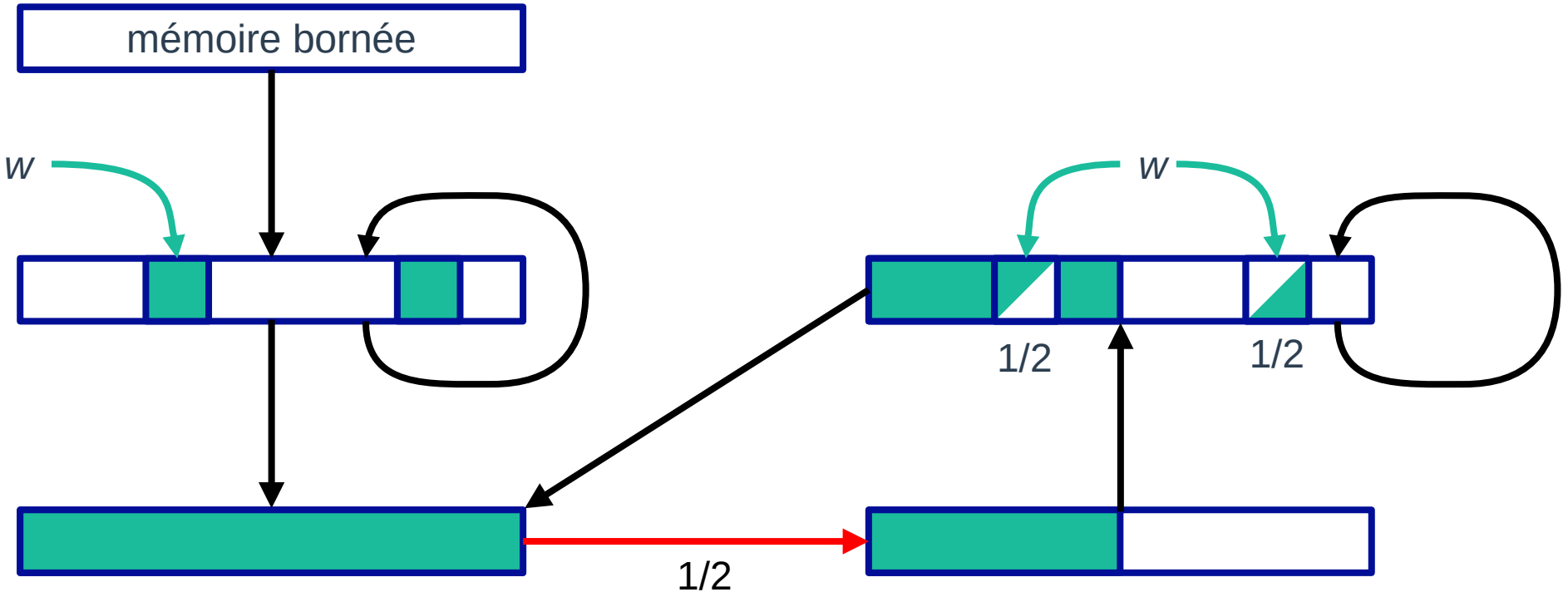
Décompte approximatif - Algorithme



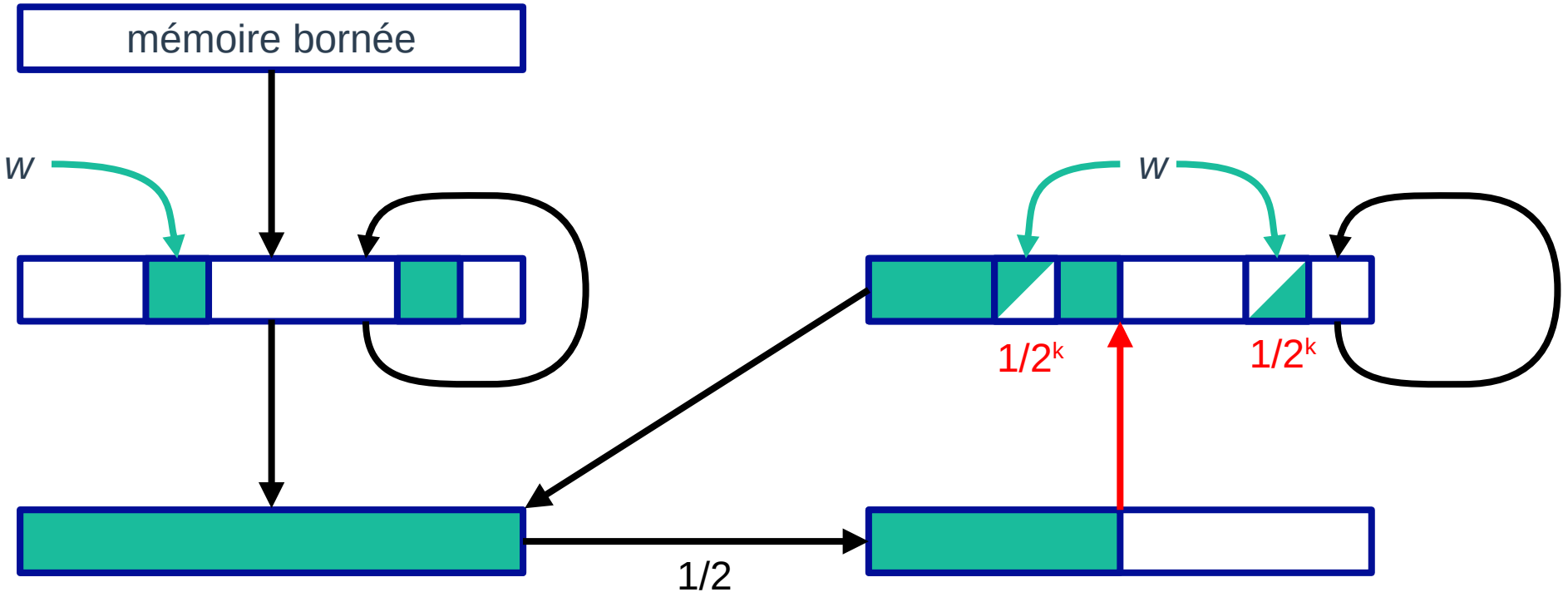
Décompte approximatif - Algorithme



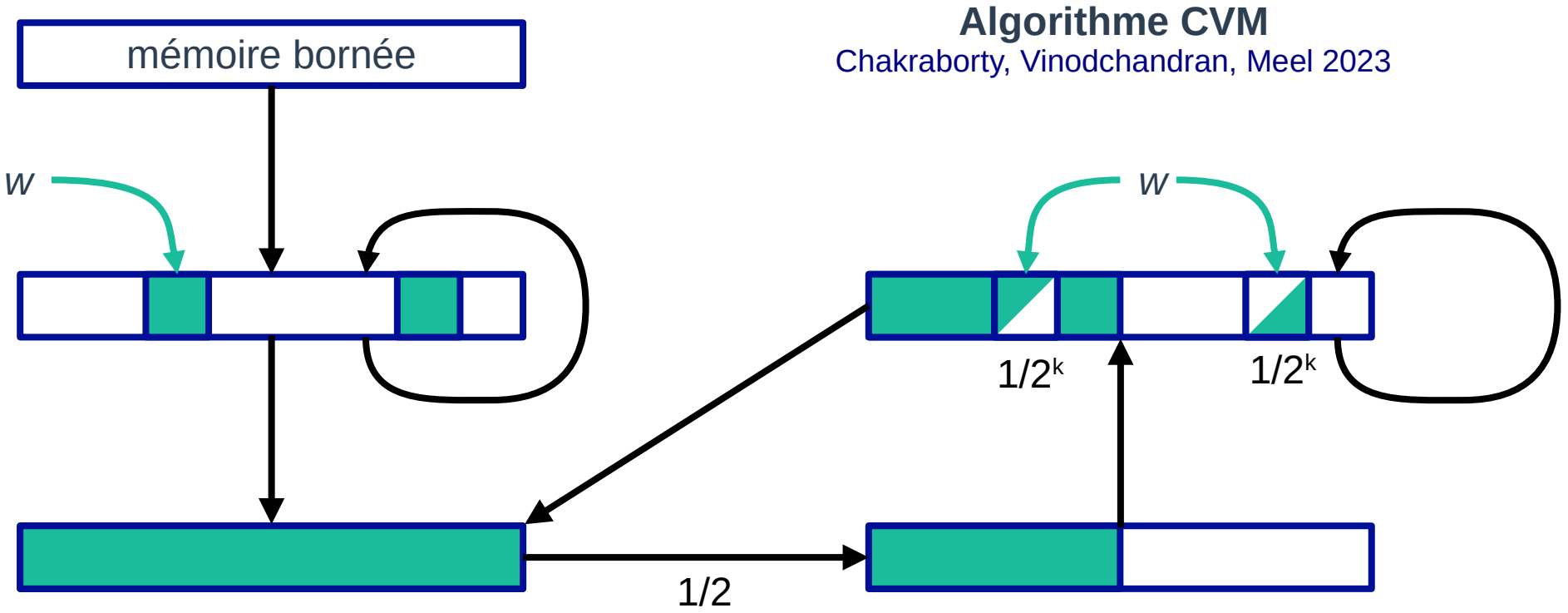
Décompte approximatif - Algorithme



Décompte approximatif - Algorithme



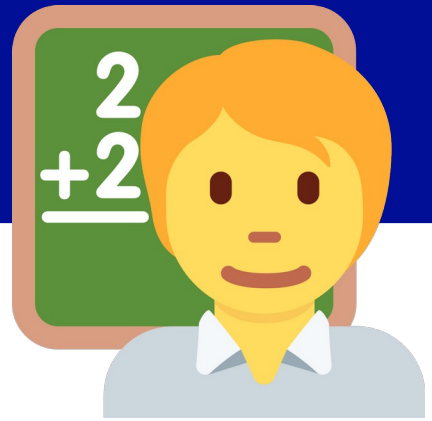
Décompte approximatif - Algorithme



Algorithme CVM

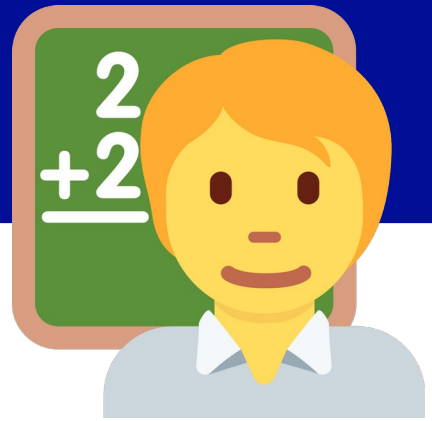
Chakraborty, Vinodchandran, Meel 2023

Un peu de probabilités



Probabilité *P(w)* qu'un mot du texte soit dans la table après **2 rounds** (1 complet et 1 partiel) :

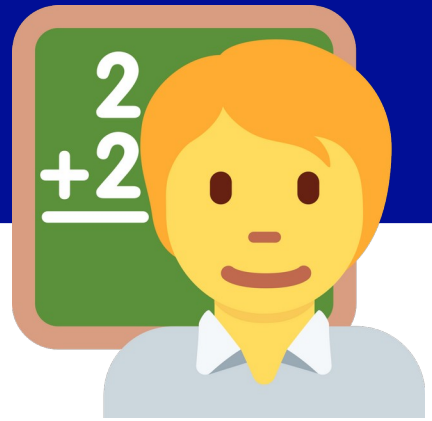
Un peu de probabilités



Probabilité *P(w)* qu'un mot du texte soit dans la table après **2 rounds** (1 complet et 1 partiel) :

- si *w* n'apparaît qu'au premier round, *P(w) = 1/2* (« nettoyage »)

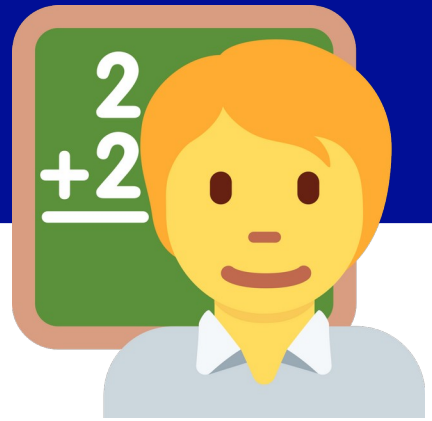
Un peu de probabilités



Probabilité *P(w)* qu'un mot du texte soit dans la table après **2 rounds** (1 complet et 1 partiel) :

- si *w* n'apparaît qu'au premier round, *P(w) = 1/2* (« nettoyage »)
- si *w* n'apparaît qu'au second round, *P(w) = 1/2* (ajout)

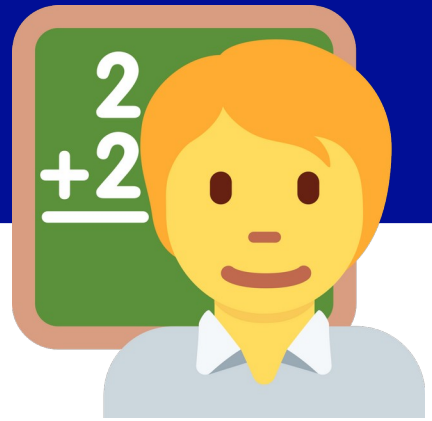
Un peu de probabilités



Probabilité *P(w)* qu'un mot du texte soit dans la table après **2 rounds** (1 complet et 1 partiel) :

- si *w* n'apparaît qu'au premier round, *P(w) = 1/2* (« nettoyage »)
- si *w* n'apparaît qu'au second round, *P(w) = 1/2* (ajout)
- si *w* apparaît dans les 2 rounds, *P(w) = 1/2* (ajout/suppression)

Un peu de probabilités

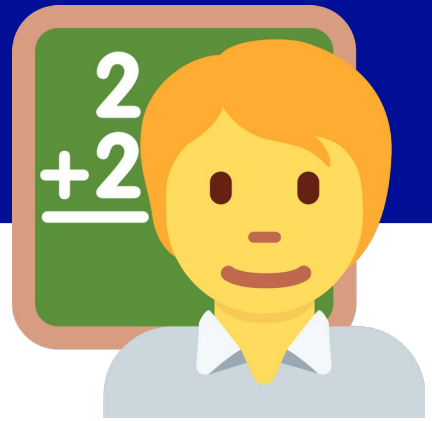


Probabilité *P(w)* qu'un mot du texte soit dans la table après **2 rounds** (1 complet et 1 partiel) :

- si *w* n'apparaît qu'au premier round, *P(w) = 1/2* (« nettoyage »)
- si *w* n'apparaît qu'au second round, *P(w) = 1/2* (ajout)
- si *w* apparaît dans les 2 rounds, *P(w) = 1/2* (ajout/suppression)

Probabilité générique après *k* rounds (récurrence) : $\boldsymbol{P(w) = 1/2^k}$

Un peu de probabilités



Probabilité *P(w)* qu'un mot du texte soit dans la table après **2 rounds** (1 complet et 1 partiel) :

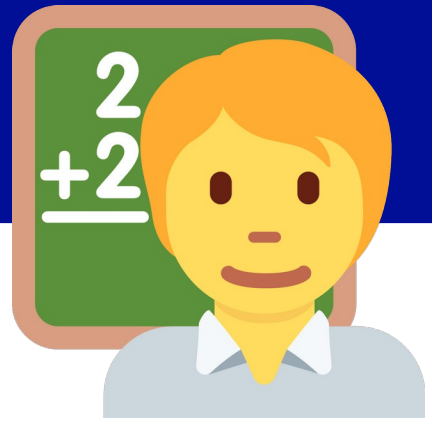
- si *w* n'apparaît qu'au premier round, *P(w) = 1/2* (« nettoyage »)
- si *w* n'apparaît qu'au second round, *P(w) = 1/2* (ajout)
- si *w* apparaît dans les 2 rounds, *P(w) = 1/2* (ajout/suppression)

Probabilité générique après *k* rounds (récurrence) : $\boldsymbol{P(w) = 1/2^k}$

Nombre *N* de mots distincts du texte :

$$size(mem) \simeq N * P(w)$$

Un peu de probabilités



Probabilité *P(w)* qu'un mot du texte soit dans la table après **2 rounds** (1 complet et 1 partiel) :

- si *w* n'apparaît qu'au premier round, *P(w) = 1/2* (« nettoyage »)
- si *w* n'apparaît qu'au second round, *P(w) = 1/2* (ajout)
- si *w* apparaît dans les 2 rounds, *P(w) = 1/2* (ajout/suppression)

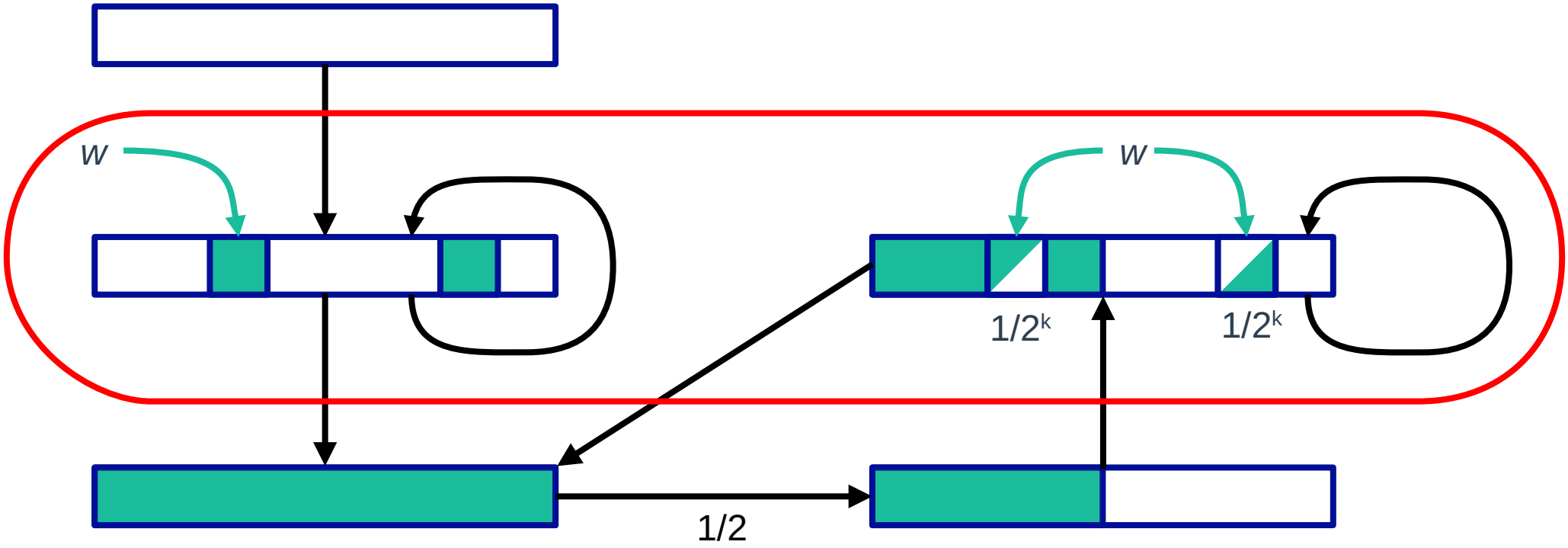
Probabilité générique après *k* rounds (récurrence) : $\boldsymbol{P(w) = 1/2^k}$

Nombre *N* de mots distincts du texte :

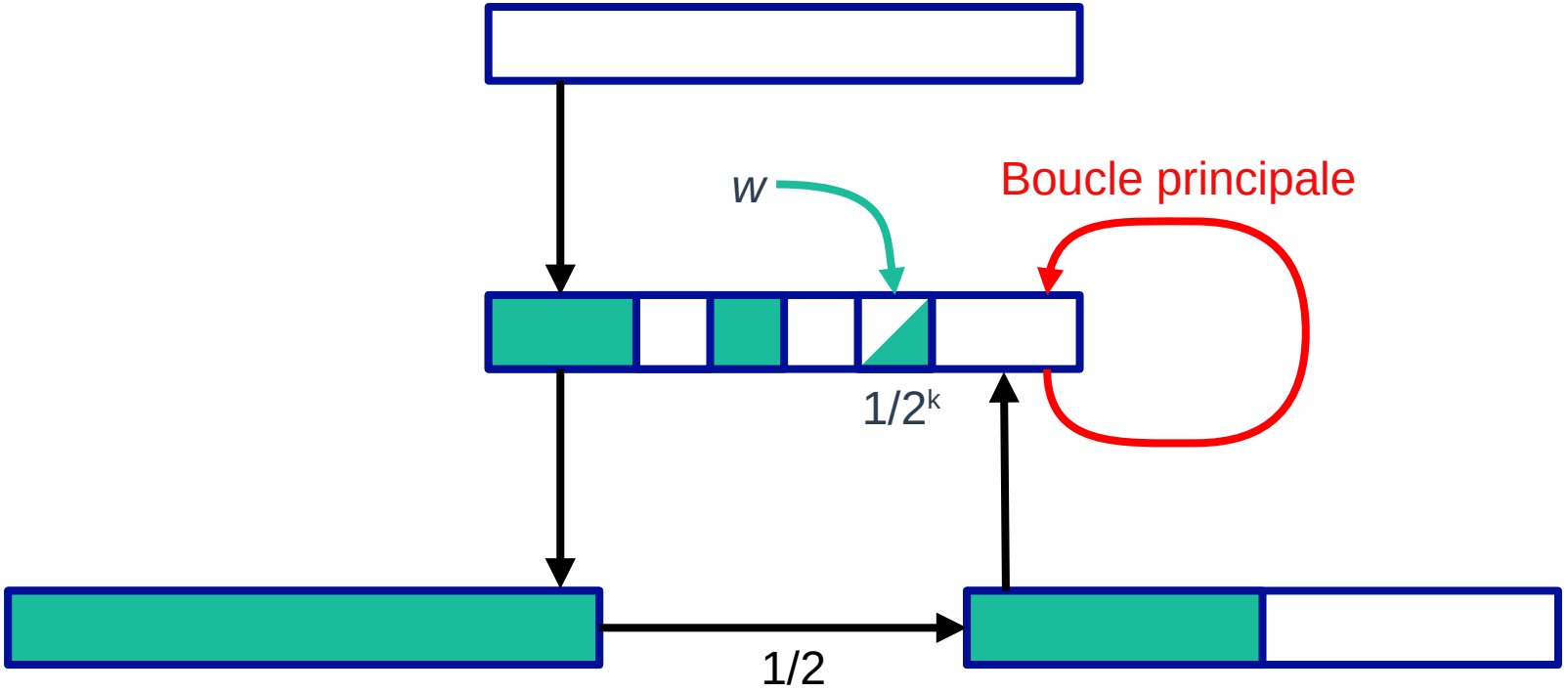
$$size(mem) \simeq N * P(w)$$

$$N \simeq size(mem) / P(w) = \boldsymbol{size(mem) * 2^k}$$

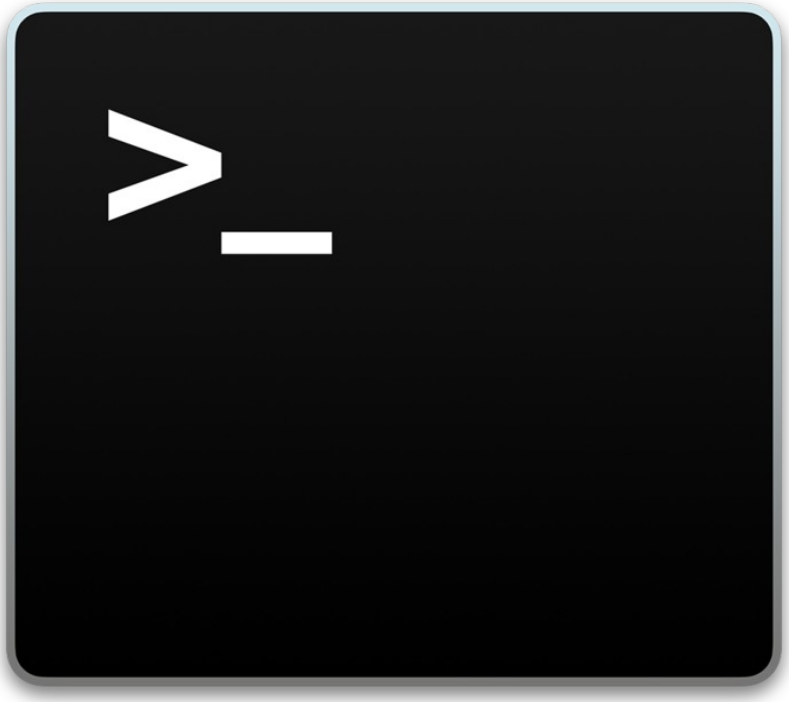
Implémentation



Implémentation



Décompte approximatif



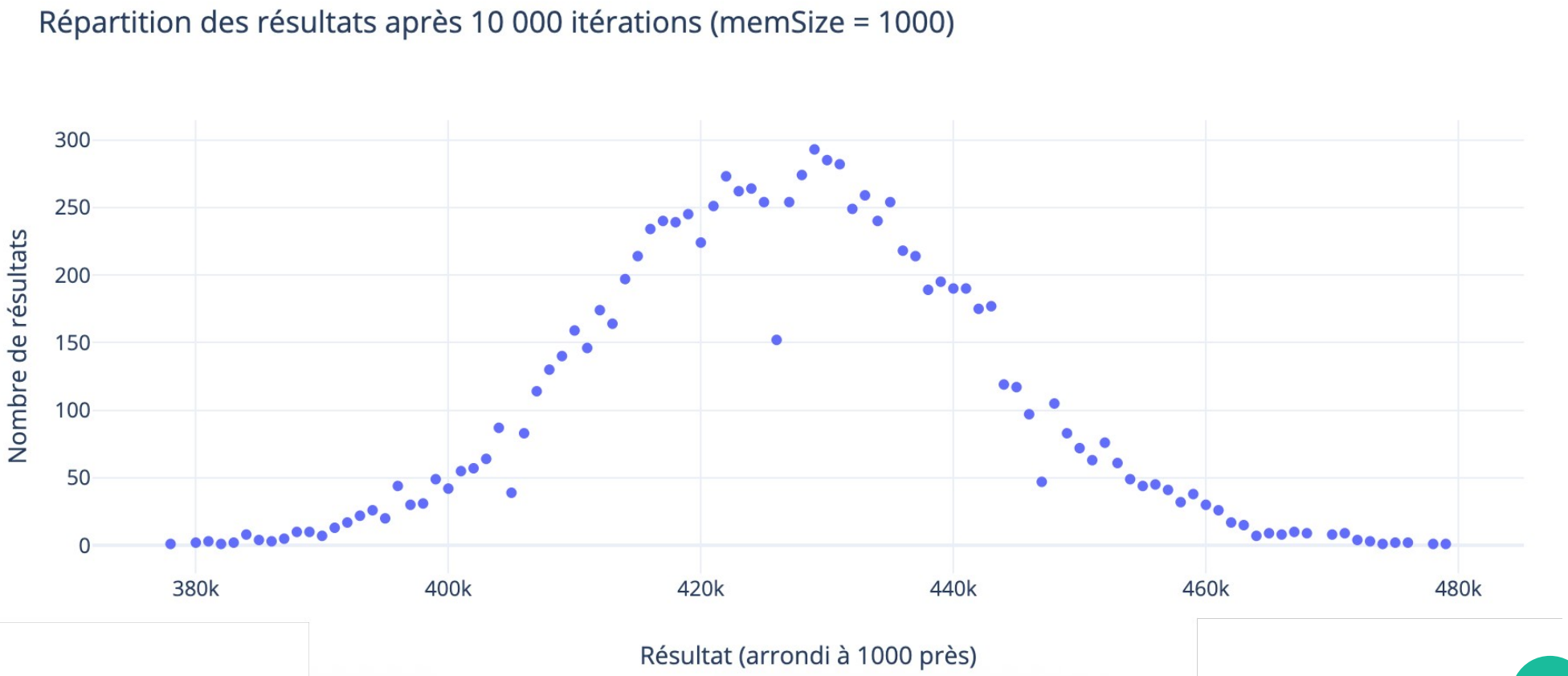
Décompte approximatif

\+ performant

\+ consommation mémoire maîtrisée

= approximation raisonnable

Répartition des résultats



Qualité de l'approximation (pour 427 430 mots distincts)

MemSize = 100

- **moyenne** = 427 717 (0.067% d'erreur)
- **écart-type** = 55 866 (13.06% de la moyenne)

66.7% des résultats ont moins de **13.06%** d'erreur
95% des résultats ont moins de 25.60% d'erreur
99.6% des résultats ont moins de 39.18% d'erreur

MemSize = 1 000

- **moyenne** = 427 576 (0.034% d'erreur)
- **écart-type** = 14 752 (3.45% de la moyenne)

66.7% des résultats ont moins de 3.45% d'erreur
95% des résultats ont moins de **6.76%** d'erreur
99.6% des résultats ont moins de **10.35%** d'erreur

MemSize = 10 000

- **moyenne** = 427 373 (0.013% d'erreur)
- **écart-type** = 5 109 (1.20% de la moyenne)

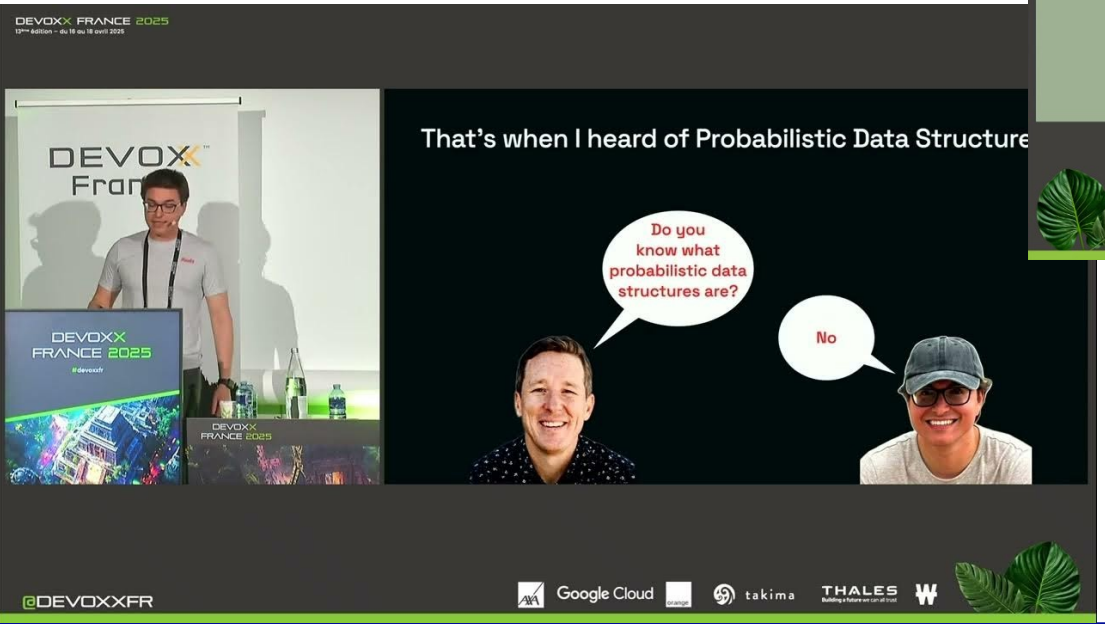
66.7% des résultats ont moins de 1.20% d'erreur
95% des résultats ont moins de 2.34% d'erreur
99.6% des résultats ont moins de **3.59%** d'erreur

Que peut-on faire d'autre ?

Image de catalyststuff sur Freepik

Pour aller plus loin…

Efficient probabilistic data structures
(Raphael de Lio)



Structures de données probabilistes
(Hela Ben Khalfallah)

Structures probabilistes

- Basées sur des **streams** et des **fonctions de hachage** (collisions)

Bloom Filter	Appartenance à un ensemble	• mémoire bornée • recherche rapide • pas de faux négatifs
HyperLogLog	Nombre d'éléments uniques	• mémoire limitée • combinable • peu d'erreurs (≃1%)
Count-min Sketch	Nombre d'occurrences par élément	• mémoire limitée • insertion rapide • surestime légèrement

source : Hela Ben Khalfallah

Intégrations

Redis

Probabilistic data structures

Probabilistic data structures let you answer common questions about your data streams and large datasets much faster. The significant advantages given in terms of efficiency in memory usage and processing speed is a trade-off with absolute accuracy. In Redis 8, in addition to **HyperLogLog**, which allows estimating the cardinality of a set, there are now 5 more probabilistic data structures:

- **Bloom** filter and **Cuckoo** filter—for checking if a given value has already appeared in a data stream
- **Count-min sketch**—for estimating how many times a given value appeared in a data stream
- **Top-k**—for finding the most frequent values in the data stream
- **t-digest**—for querying which fraction of the values in the data stream are smaller/larger than a given value.

Intégrations

- **Redis** : Bloom filter, Hyperloglog, Count-min sketch, Top-k, t-digest
- **PostgreSQL** : Bloom index
- **Spark** : Hyperloglog, Min-hash, Count-min sketch
- **Druid**, **ElasticSearch**, **BigQuery** : t-digest, Hyperloglog

source : Hela Ben Khalfallah

Conclusions

On peut encore faire de l'algo aujourd'hui
(et c'est pas forcément compliqué)

Les probabilités peuvent nous être utiles !

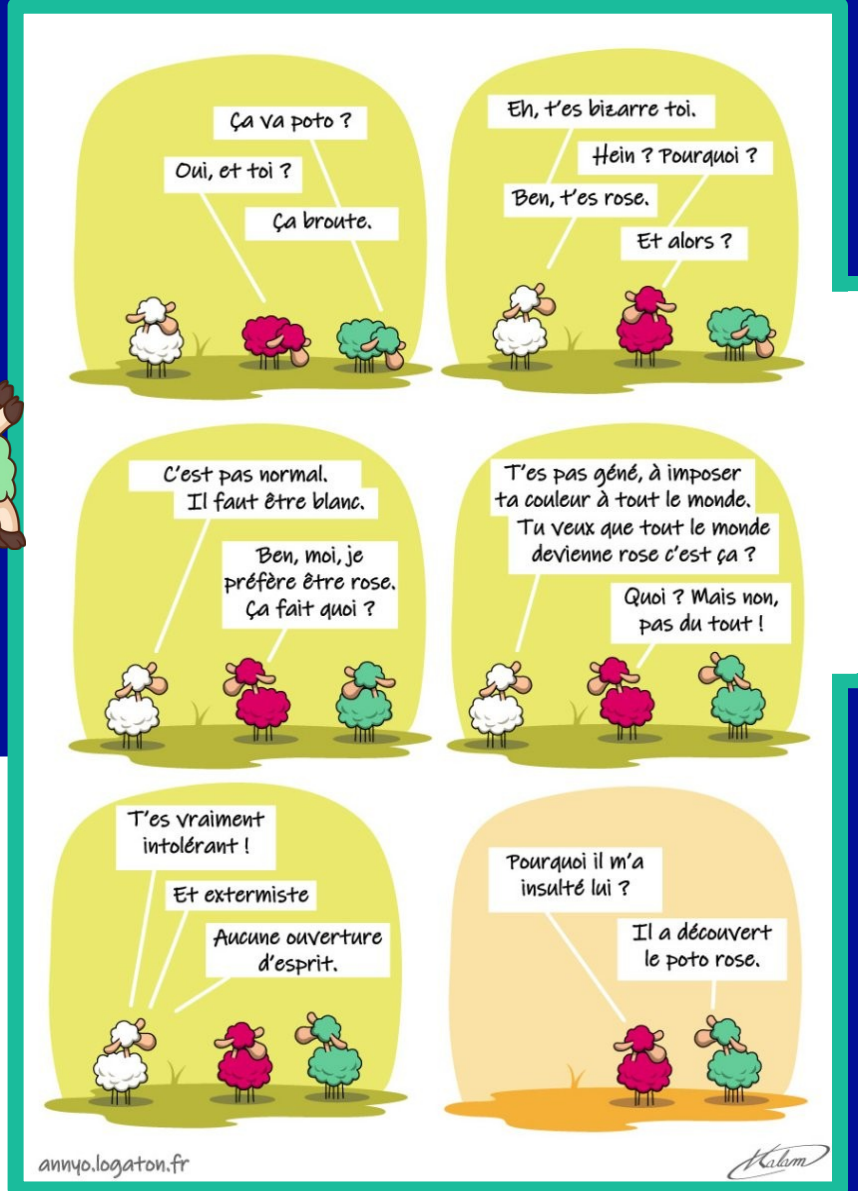
On n'a pas toujours besoin de résultats exacts
économisez des ressources si vous le pouvez

Merci !

Feedbacks

Code & Slides

https://github.com/benoitmasson/gi-go



(Bonus)

Feedbacks

Code & Slides

https://github.com/benoitmasson/gi-go