

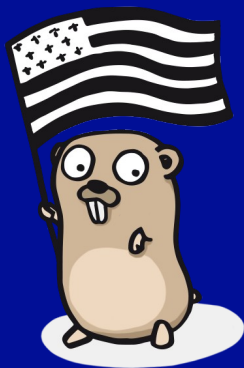
# G.I. Go

## Générateurs & Itérateurs en Go

*Comment compter les Gophers  
sans perdre la mémoire*



*Image de ChatGPT*



Benoît Masson

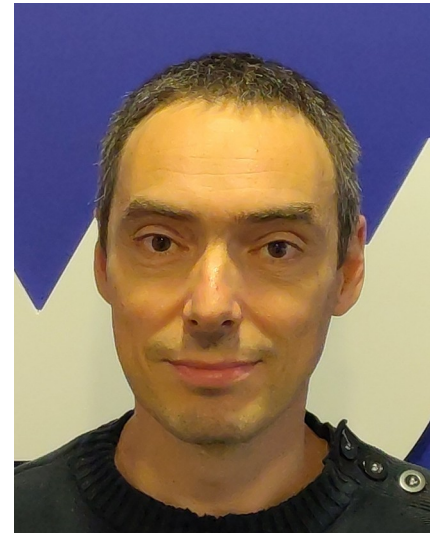


19 novembre 2024

# Générique

- Développeur Go depuis 2015
- Software Craftsman

@OVHcloud Rennes  
Noms de Domaines depuis 2020



benoitmasson

# Mais pourquoi ?

- Uniformisation des pratiques observées
  - `archive/tar.Reader.Next`
  - `bufio.Reader.ReadByte`
  - `bufio.Scanner.Scan`
  - `container/ring.Ring.Do`
  - `database/sql.Rows`
  - `expvar.Do`
  - `flag.Visit`
  - `go/token.FileSet.Iterate`
  - `path/filepath.Walk`
  - `runtime.Frames.Next`
  - `sync.Map.Range`
  - ...

# Mais pourquoi ?

- Uniformisation des pratiques observées

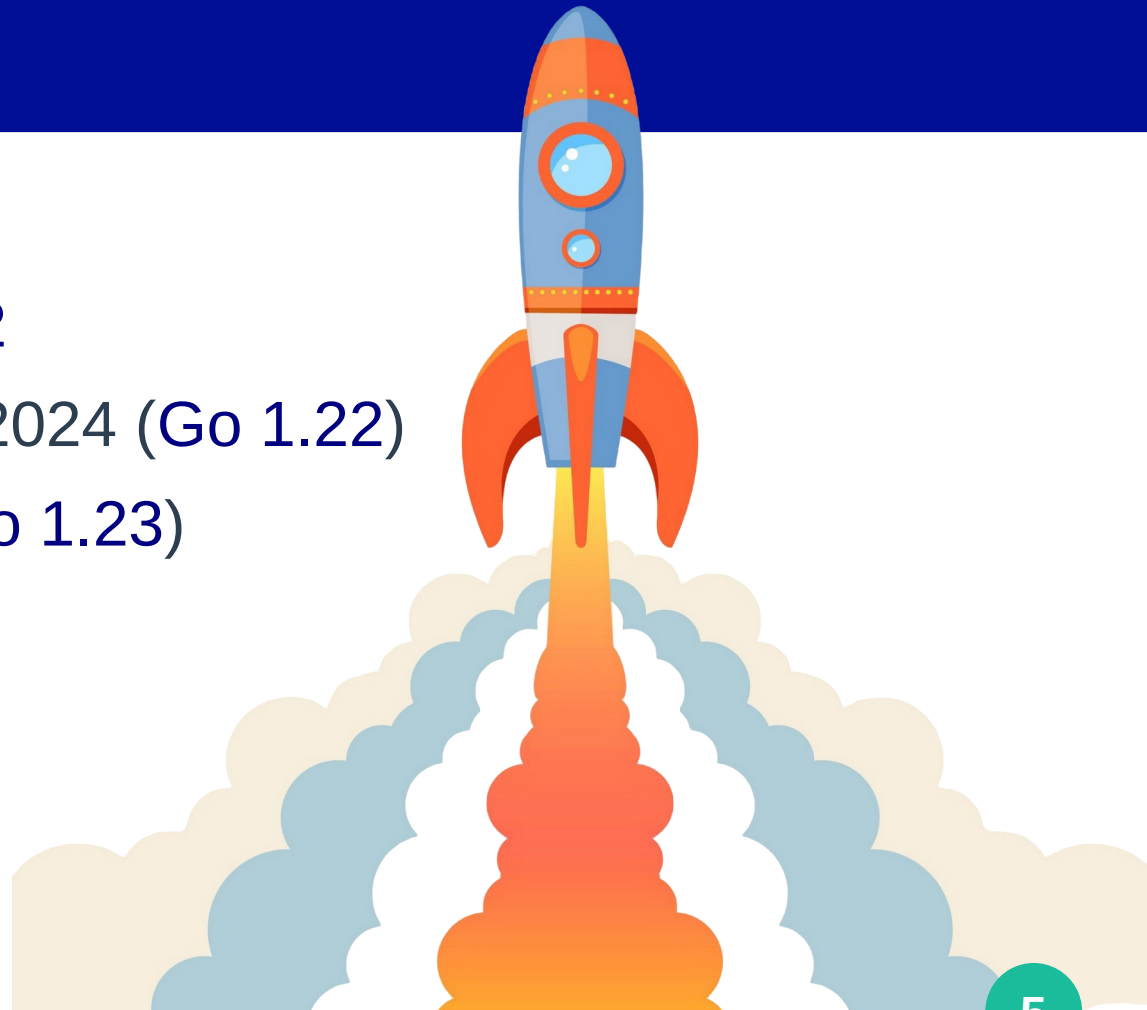
- `archive/tar.Reader.Read`
- `bufio.Reader.Read`
- `bufio.Scanner.Scan`
- `container/ring.Ring`
- `database/sql.Row`
- `expvar.Do`
- `flag.Visit`
- `io/token.FileSet.Iterate`
- `path/filepath.Walk`
- `runtime.Frames.Next`
- `sync.Map.Range`



- S'appuie sur les génériques, disponibles depuis 2022 (Go 1.18)

# Historique

- Discuté depuis octobre 2022
- Expérimenté depuis février 2024 (Go 1.22)
- Généralisé en août 2024 (Go 1.23)



# Comment ça marche



# Comment ça marche



- itérateur « push » = **générateur**
  - 1 ou 2 valeurs
- `yield`, `break`
- **itérateur** « pull »
- test unitaire

# Bibliothèque standard

## slices

- `All([]E) iter.Seq2[int, E]`
- `Values([]E) iter.Seq[E]`
- `Collect(iter.Seq[E]) []E`
- `AppendSeq([]E, iter.Seq[E]) []E`
- `Backward([]E) iter.Seq2[int, E]`
- `Sorted(iter.Seq[E]) []E`
- `SortedFunc(iter.Seq[E], func(E, E) int) []E`
- `SortedStableFunc(iter.Seq[E], func(E, E) int) []E`
- `Repeat([]E, int) []E`
- `Chunk([]E, int) iter.Seq([]E)`

## maps

- `All(map[K]V) iter.Seq2[K, V]`
- `Keys(map[K]V) iter.Seq[K]`
- `Values(map[K]V) iter.Seq[V]`
- `Collect(iter.Seq2[K, V]) map[K, V]`
- `Insert(map[K, V], iter.Seq2[K, V])`



# Critiques

Lire par exemple : `go-evolves-in-the-wrong-direction`

- Deux façons d'itérer sur les anciennes bibliothèques
- Signature de l'itérateur restrictif, pas adapté à toutes les situations
- Augmentation de la complexité implicite du *range*
- Vérification manuelle des erreurs à chaque itération

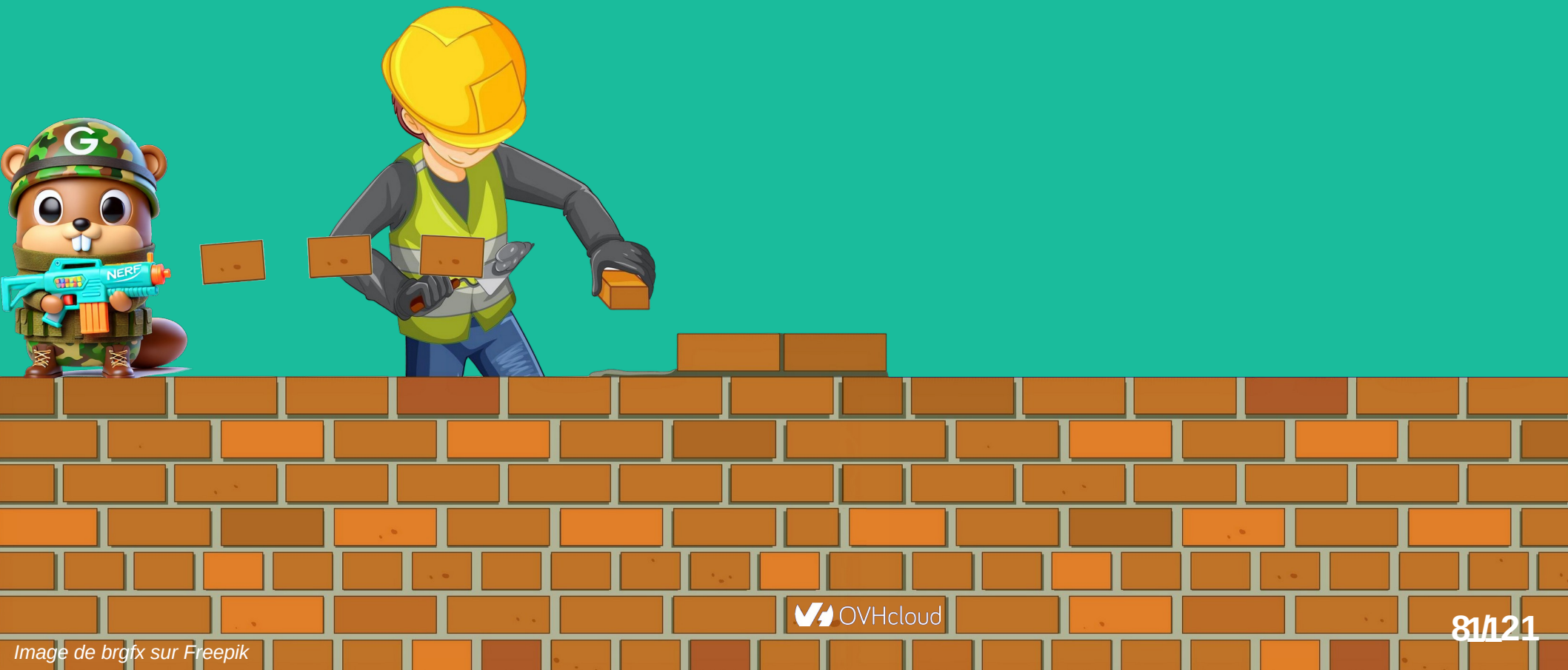
# Critiques

Lire par exemple : `go-evolves-in-the-wrong-direction`

- Deux façons d'itérer sur les anciennes bibliothèques
- Signature de l'itérateur restrictif, pas adapté à toutes les situations
- Augmentation de la complexité implicite du *range*
- Vérification manuelle des erreurs à chaque itération

Mon avis : pas un vrai problème, la complexité étant déportée dans une fonction isolée, l'usage est **simple** et **clair**

# Cas d'usage



# Parcourir un fichier mot à mot



# Parcourir un fichier mot à mot



- performance (rapidité, mémoire)
- lisibilité

# Requêtes paginées

```
func Words(g io.PaginatedGetter) iter.Seq[string] {
    return func(yield func(string) bool) {
        byteBuf := make([]byte, bufSize)
        wordBuf := bytes.Buffer{}
        var cursor string
        var results []string
        for {
            n, err := r.Read(byteBuf)
            results, cursor, err := g.getPage(cursor)
            if err != nil {
                if errors.Is(err, io.EOF) {
                    return
                }
                panic(fmt.Errorf("failed to get page: %w", err))
            }

            for _, result := range results {
                b := byteBuf[i]
                if !unicode.IsSpace(rune(b)) {
                    wordBuf.WriteByte(b)
                    continue
                }
                // b is a space
                if wordBuf.Len() == 0 {
                    // ignore empty words
                    continue
                }
                if !yield(result) {
                    return
                }
                wordBuf.Reset()
            }
        }
    }
}
```

# Scan SQL

Source : [github.com/achille-rousseau/sqlrange](https://github.com/achille-rousseau/sqlrange)

```
for p, err := range sqlrange.Query[Point](db, `select x, y from points`) {  
    if err != nil {  
        ...  
    }  
    ...  
}
```

# Comptons les gophers





# Comptons les gophers (distincts)



# Décompte exact

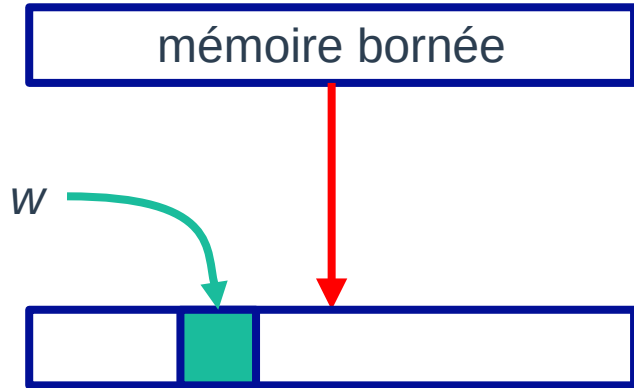


# Décompte exact

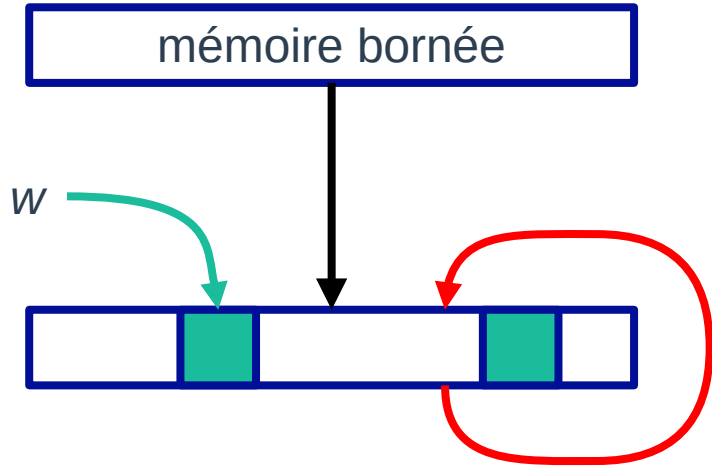


- performant
- mémoire non bornée

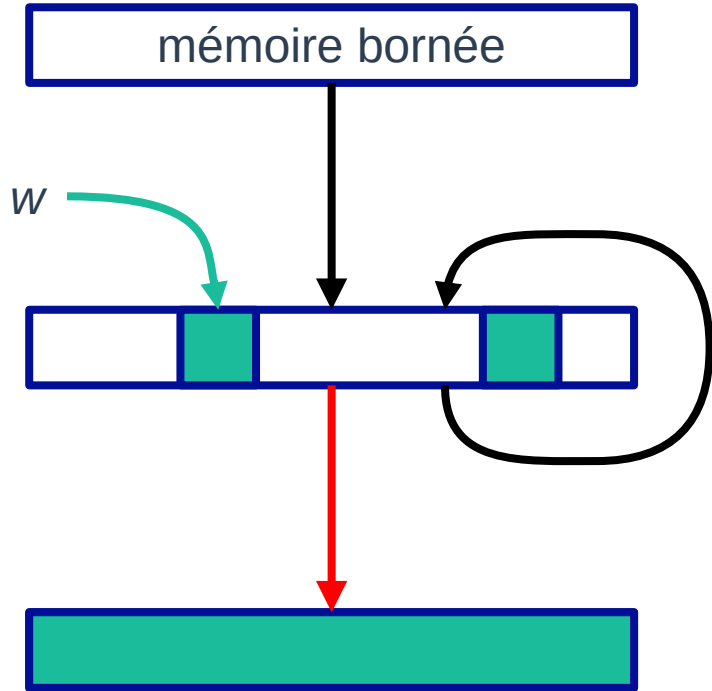
# Décompte approximatif - Algorithme



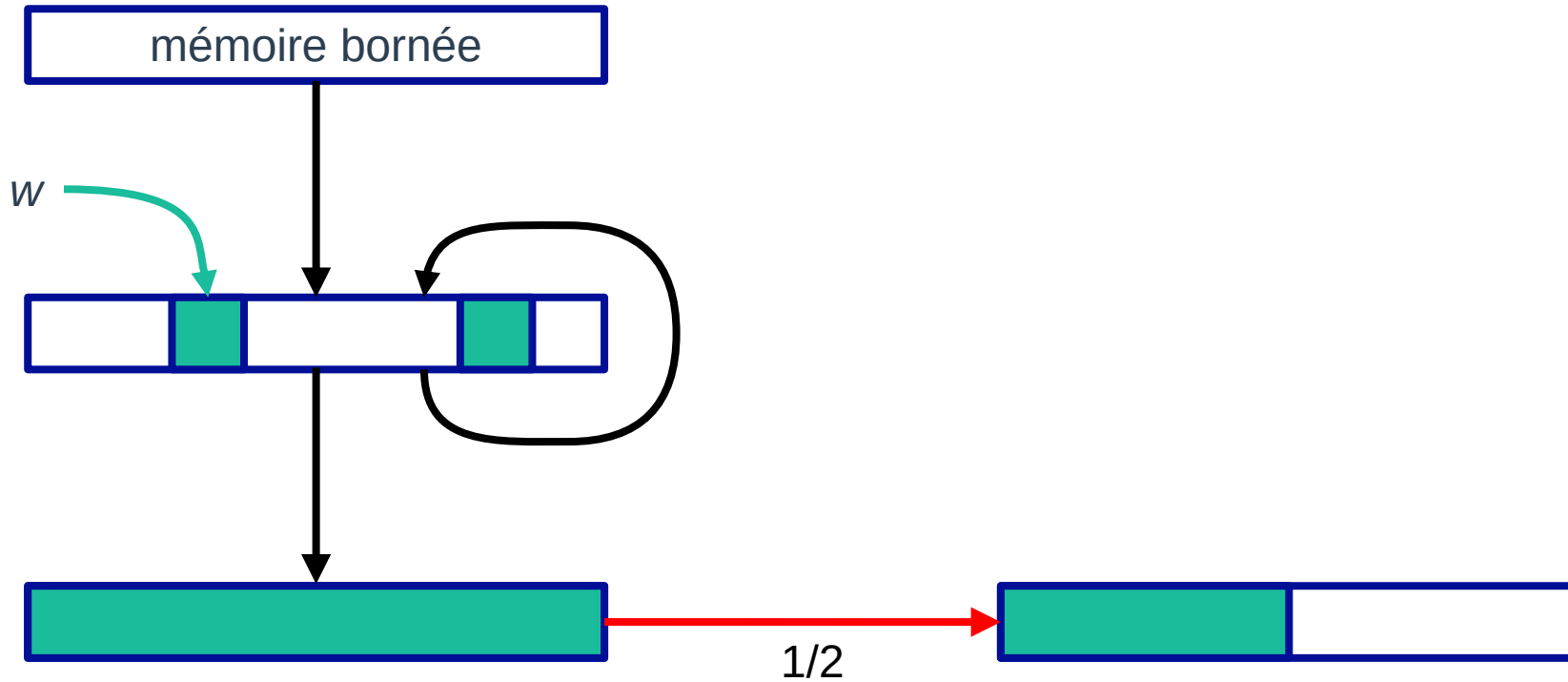
# Décompte approximatif - Algorithme



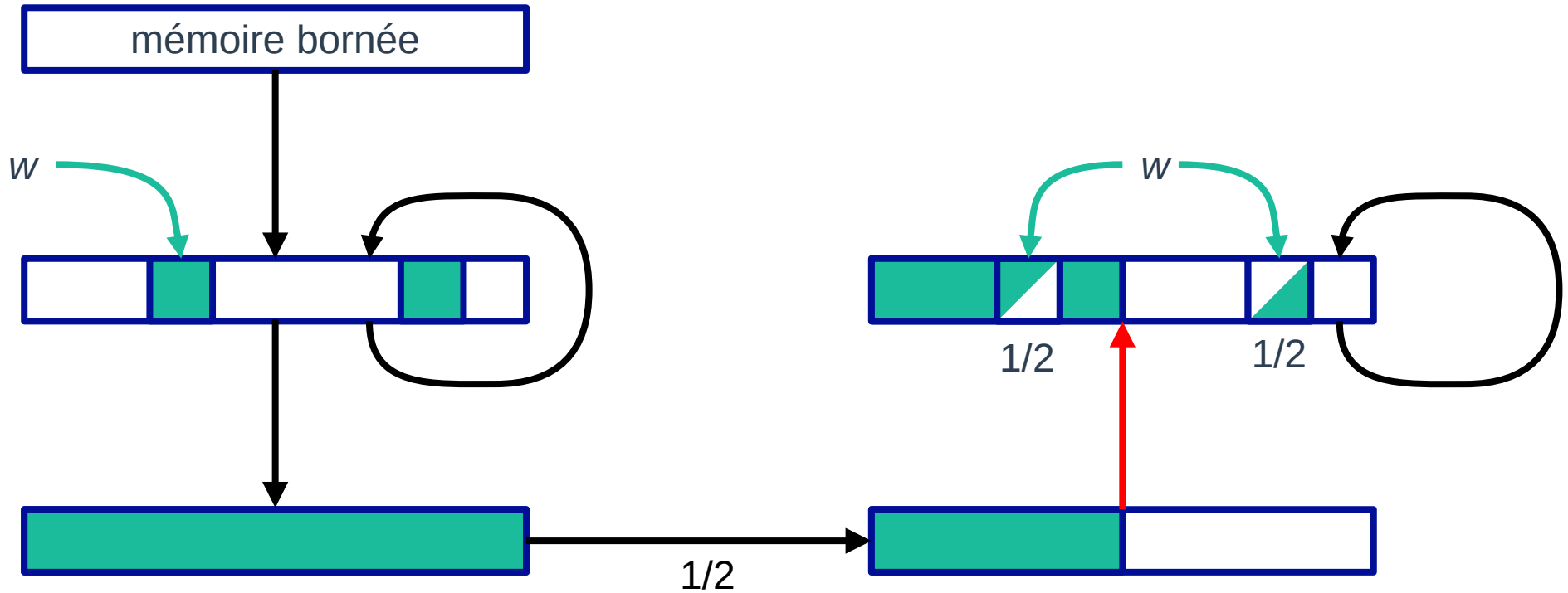
# Décompte approximatif - Algorithme



# Décompte approximatif - Algorithme

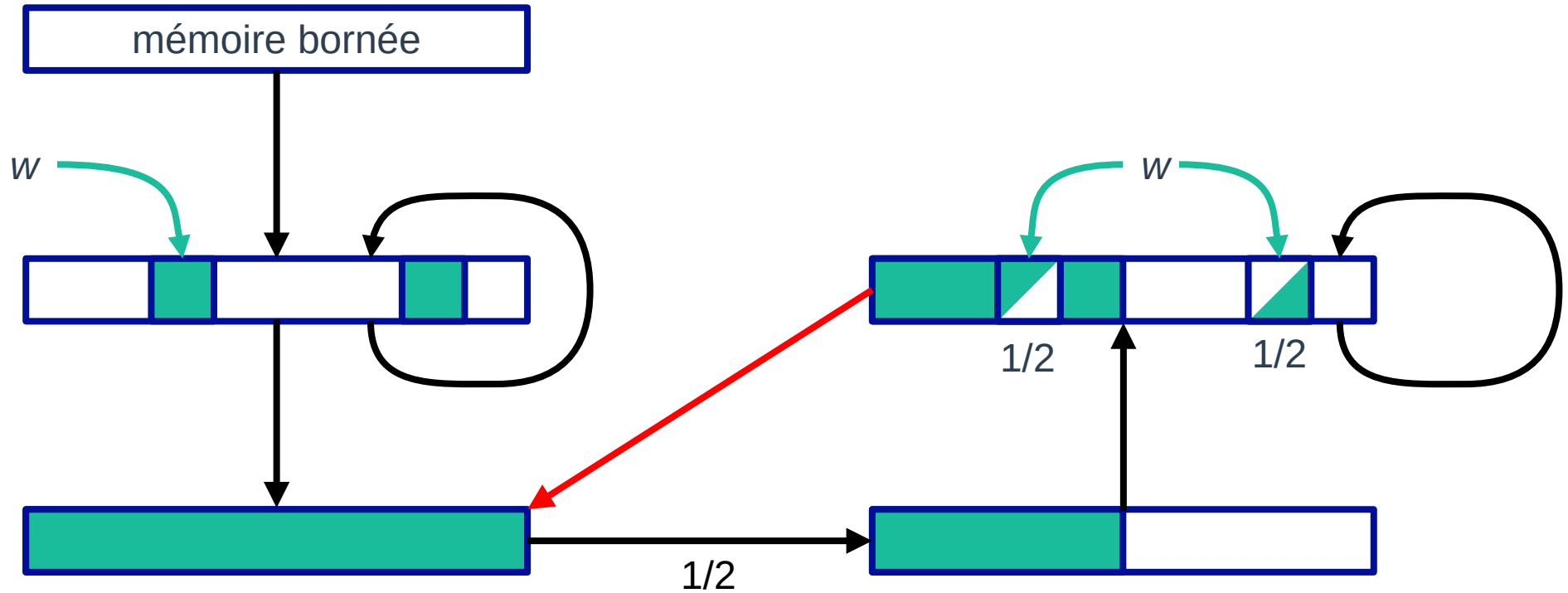


# Décompte approximatif - Algorithme

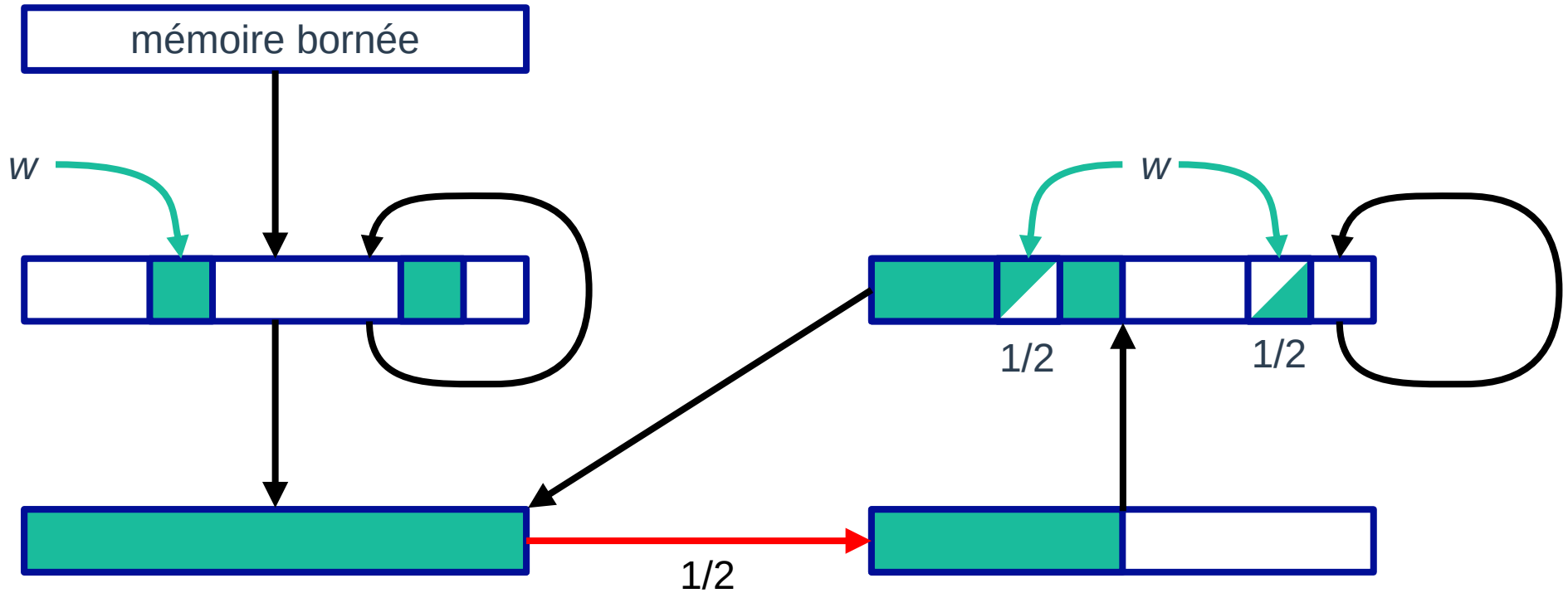




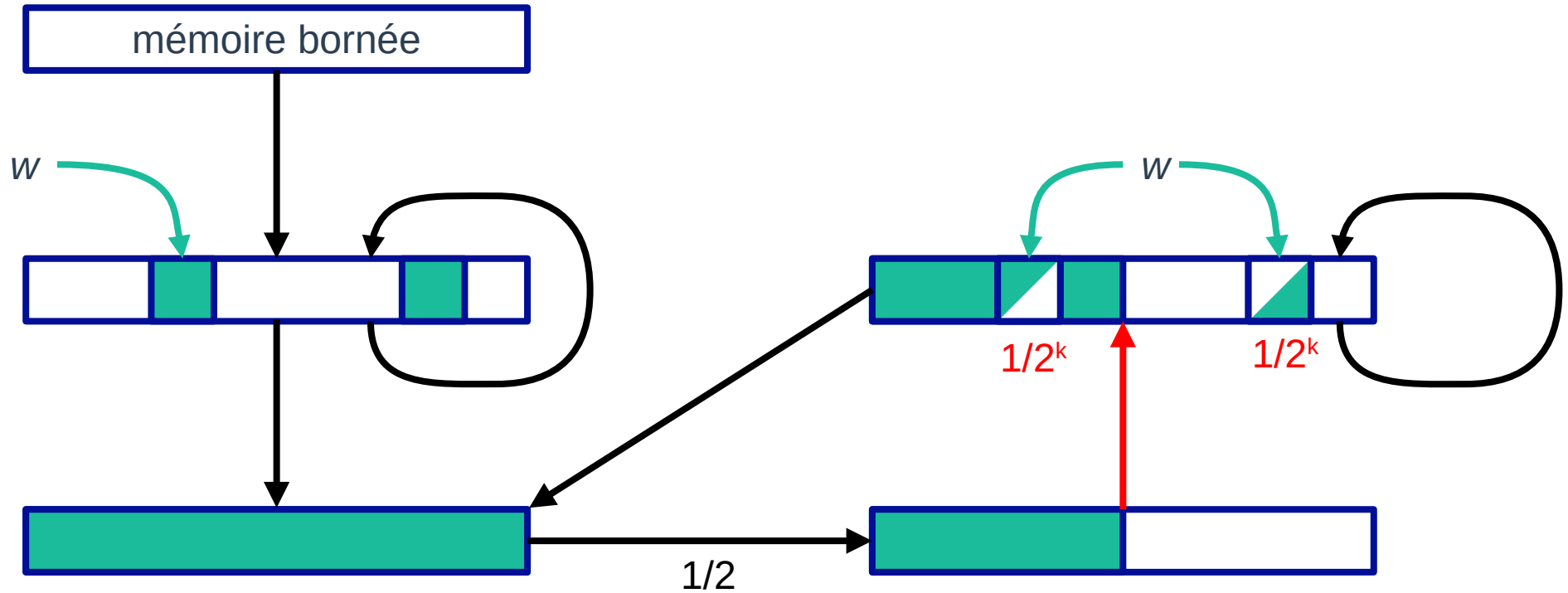
# Décompte approximatif - Algorithme



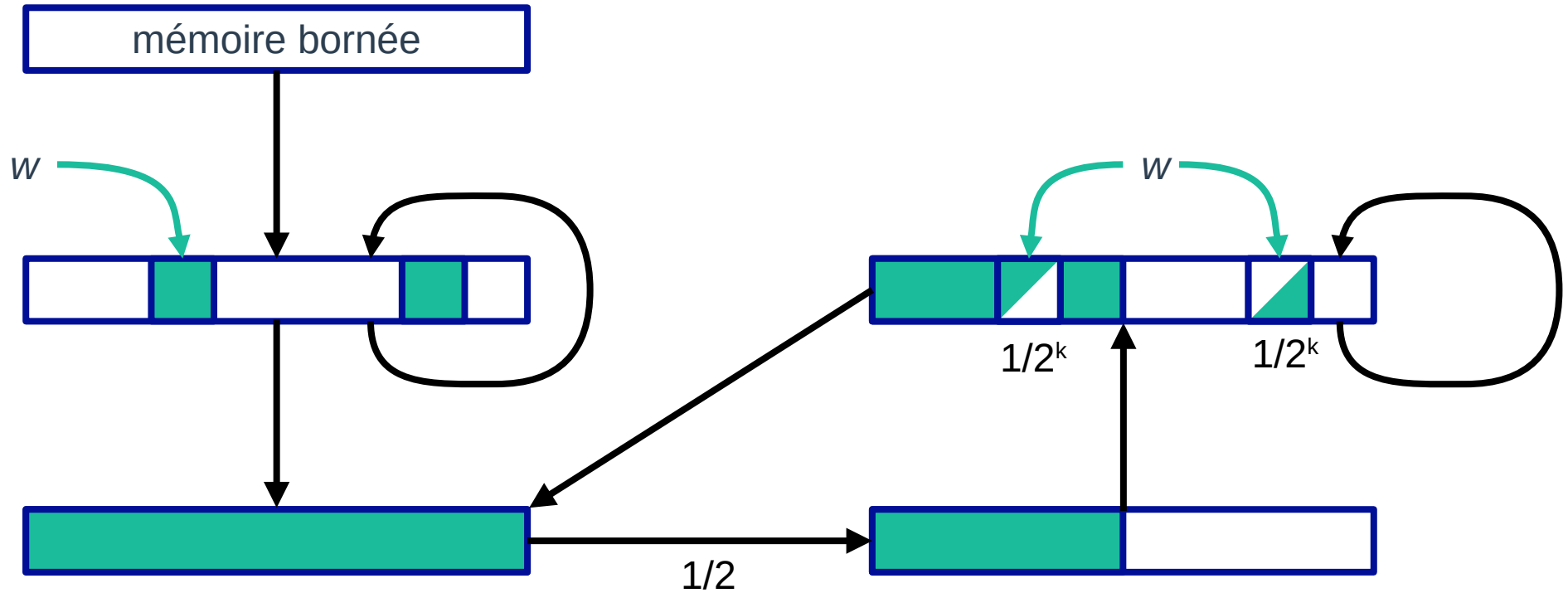
# Décompte approximatif - Algorithme



# Décompte approximatif - Algorithme

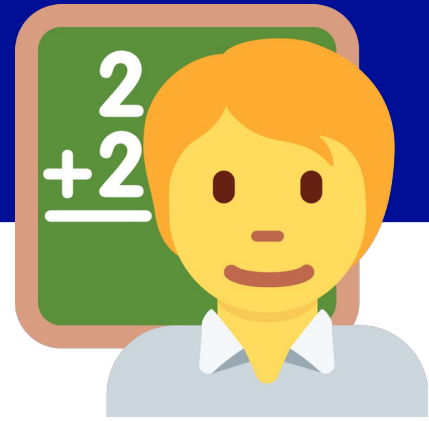


# Décompte approximatif - Algorithme

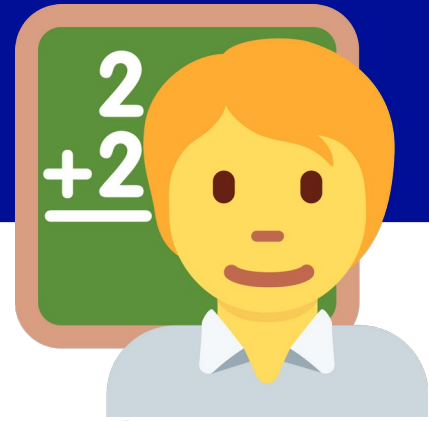


# Un peu de probabilités

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



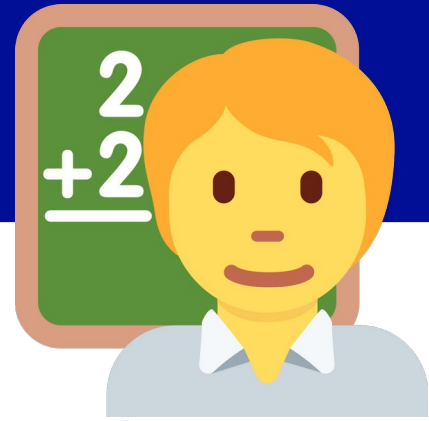
# Un peu de probabilités



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

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

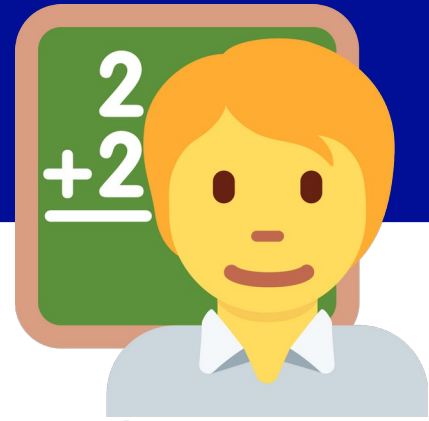
# Un peu de probabilités



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

- 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

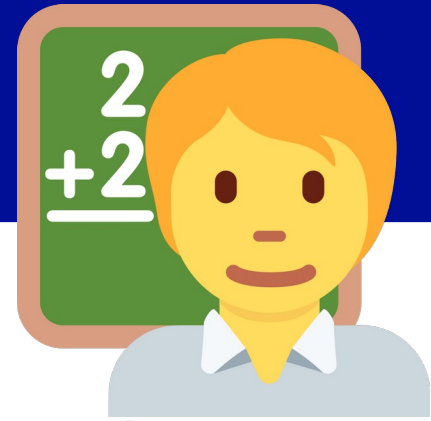


Probabilité  $P(w)$  qu'un mot du texte soit dans la table après **2 rounds** :

- 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

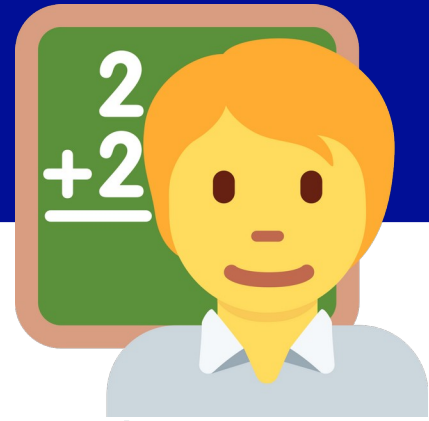


Probabilité  $P(w)$  qu'un mot du texte soit dans la table après **2 rounds** :

- 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) :  **$P(w) = 1/2^k$**

# Un peu de probabilités



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

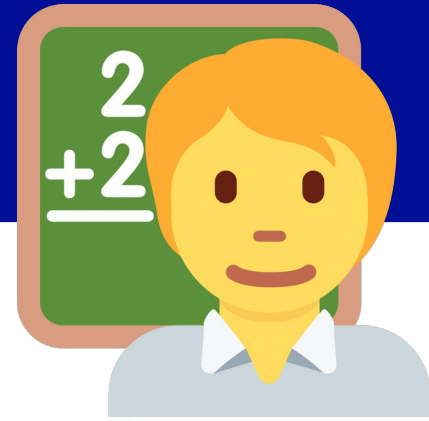
- si  $w$  n'apparaît qu'au premier round,  $P(w) = 1/2$  (« nettoyage »)
- si  $w$  n'apparaît qu'au second round,  $P(w) = 1/2$  (ajout)
- si  $w$  apparaît dans les 2 rounds,  $P(w) = 1/2$  (ajout/suppression)

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

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** :

- 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)

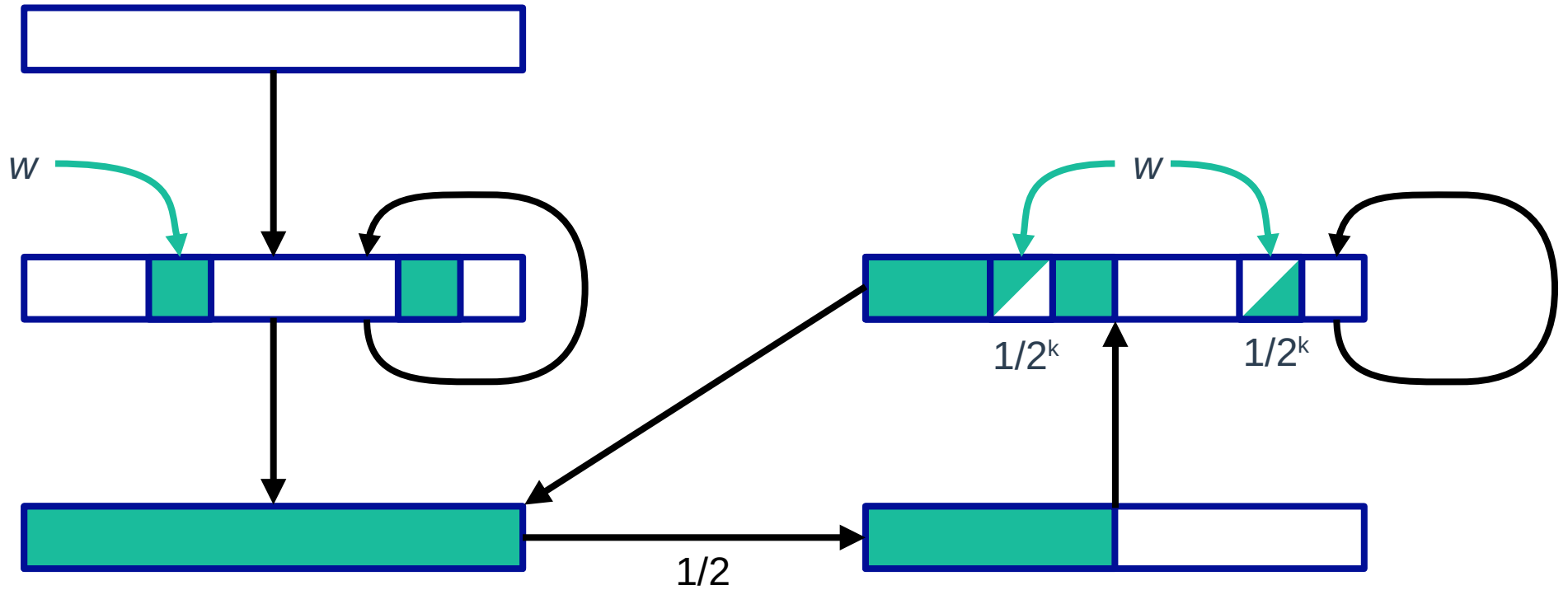
Probabilité générique après  $k$  rounds (récurrence) :  **$P(w) = 1/2^k$**

Nombre  $N$  de mots distincts du texte :

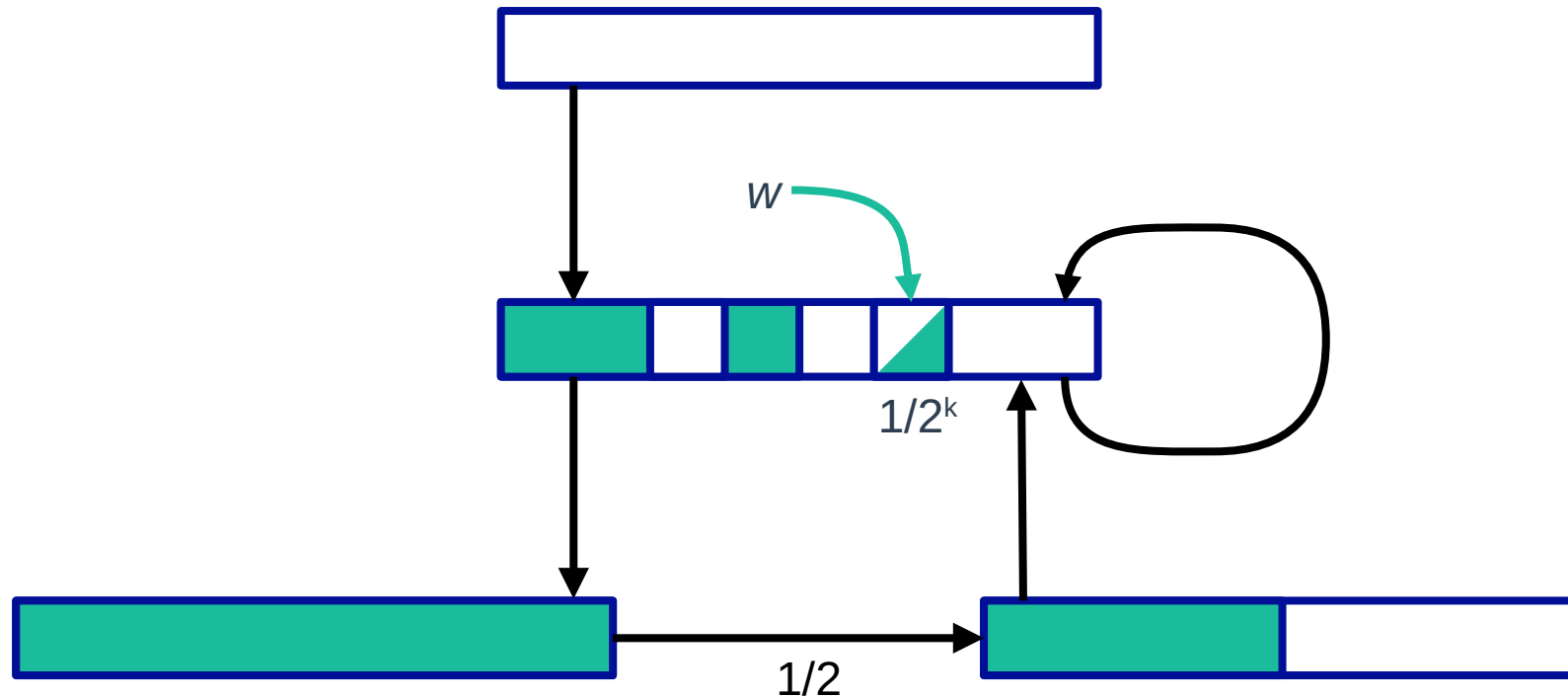
$$size(mem) \simeq N * P(w)$$

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

# Implémentation



# Implémentation



# Décompte approximatif



# Décompte approximatif



- performant
- consommation mémoire maîtrisée
- approximation raisonnable

# Conclusion

Les génériques, c'est magique  
⇒ complexité technique masquée dans la bibliothèque, usage simple

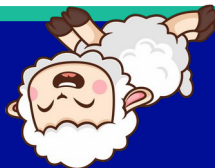
On peut encore faire de l'algo en 2024  
(et c'est pas forcément compliqué)

Autre nouveauté de Go 1.23 pour optimiser la mémoire : unique  
⇒ article de [Valentin Deleplace](#)



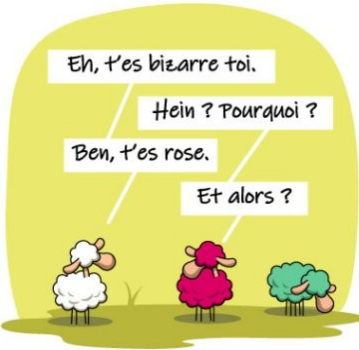
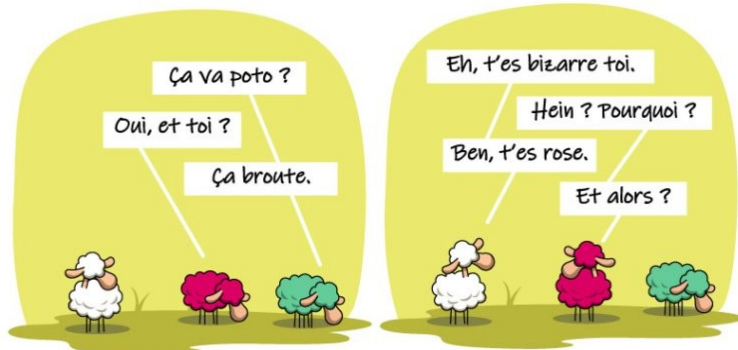


**Merci !**



Code & Slides





(Bonus)



Code & Slides

