SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Soutenance de projet
04 Juin 2019

LIRE POUR ECRIRE

Master 1 Informatique - SAR
Systemes et Applications Répartis

Etudiants :
- Nathan MAURICE
- Clément THERRY
- Smail AIDER

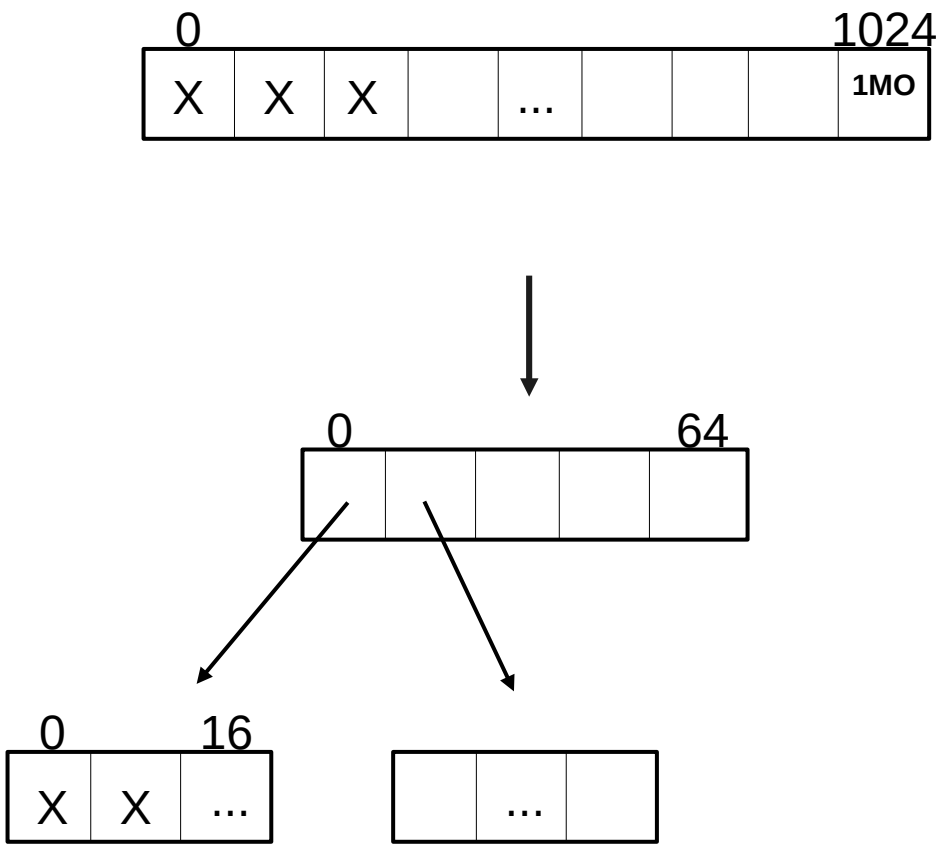
Encadrant :
- Julien SOPENA

PLAN

I. Présentation
   i. I/O – Page cache
   ii. Problématique
II. Objectifs
III. Benchmarks
      i. Ecritures séquenctielles vs espacées
      ii. Page cache et détection de vides
IV. Réalisation - Patch
   i. Algorithmes
   ii. Résultats
V. Conclusion

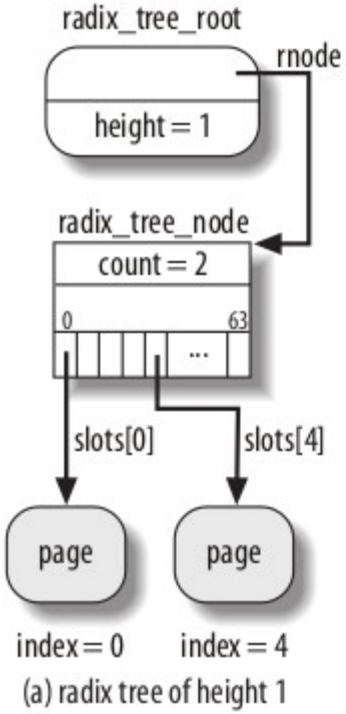
I/O – Page Cache

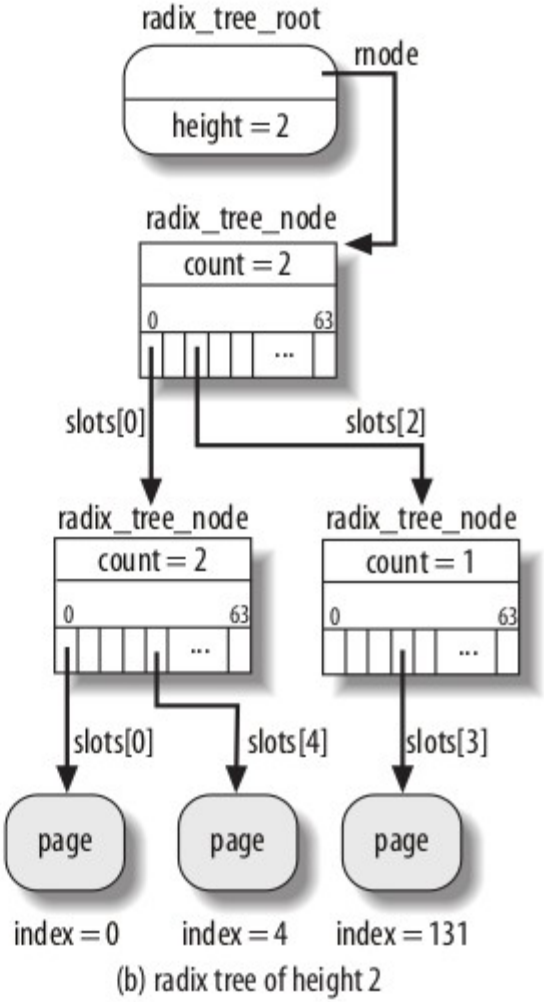
- Radix Tree



I/O – Page Cache

- Radix Tree
  - ✓ Temps de recherche réduit
  - ✓ Eviter l'encombrement mémoire

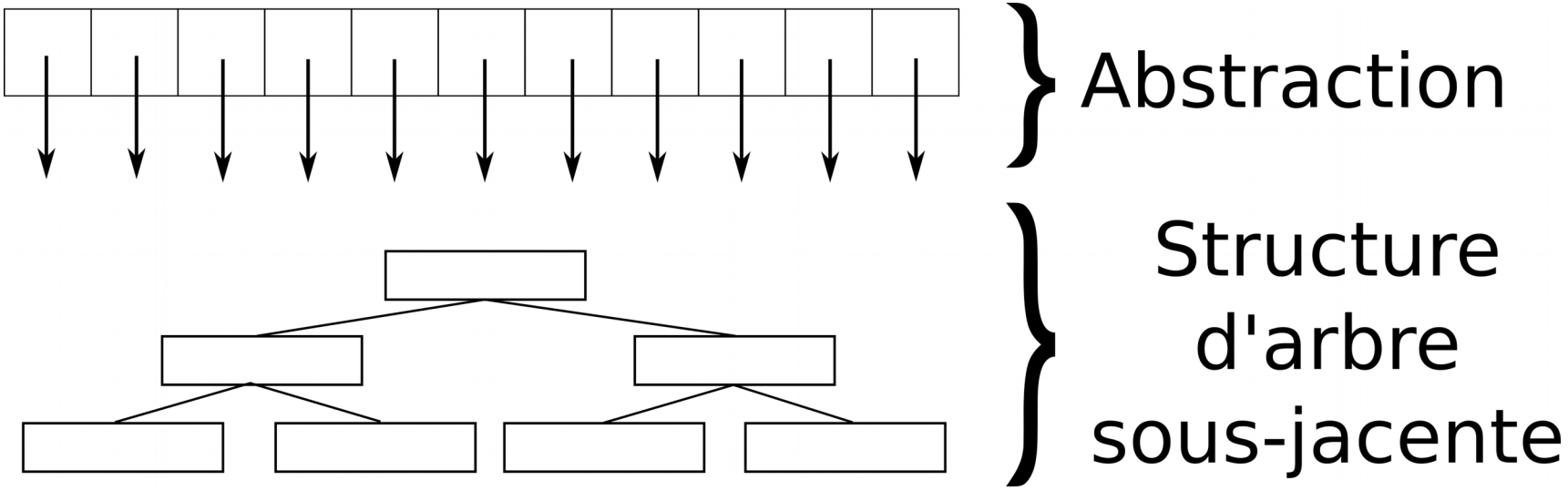




Source : Linux Kernel Development

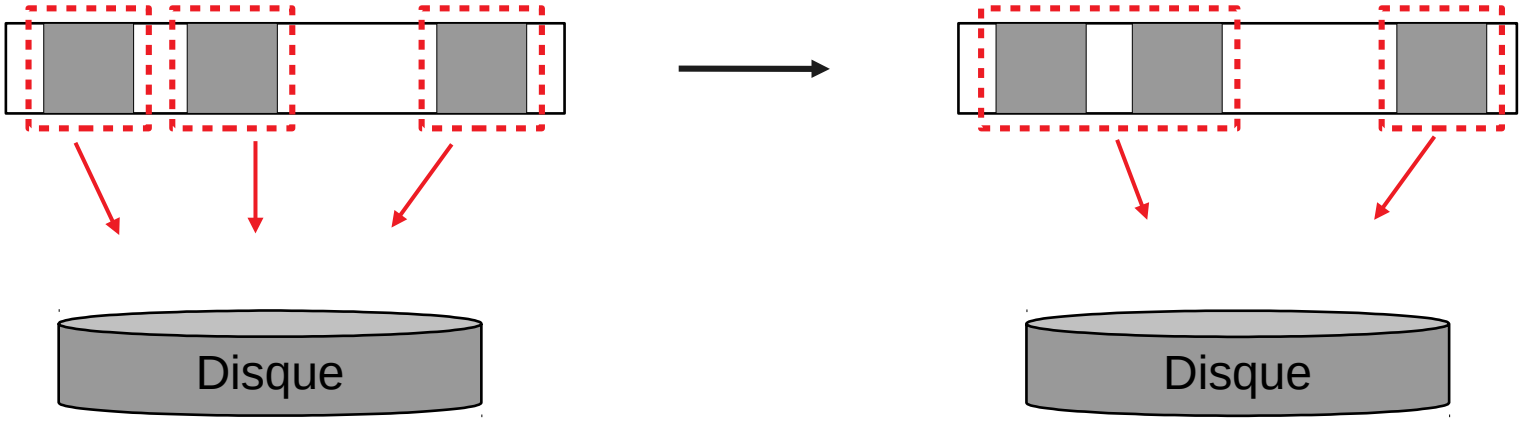
I/O – Page Cache

- Probleme radix-tree
  - Complexe et non instinctif.
- Solution: eXtensible Array – Xarray
  - Linux ≥4.20
  - Meilleure API



Problématique

- Regrouper les blocs dirty non adjacents !



Page dirty

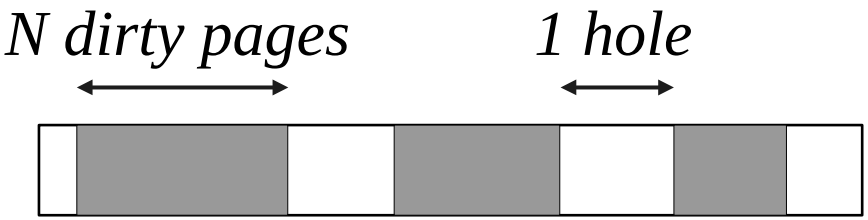
Page propre

Objectifs

- Mesurer l’impact des trous sur les performances.
  - Benchmarks en C
- Implémenter un mécanisme de détection de trous.
- Réaliser un patch pour écrire des trous “propres”.
  - Mesures améliorations

Ecritures sequentielles vs espacées

- Ecritures séquentielles



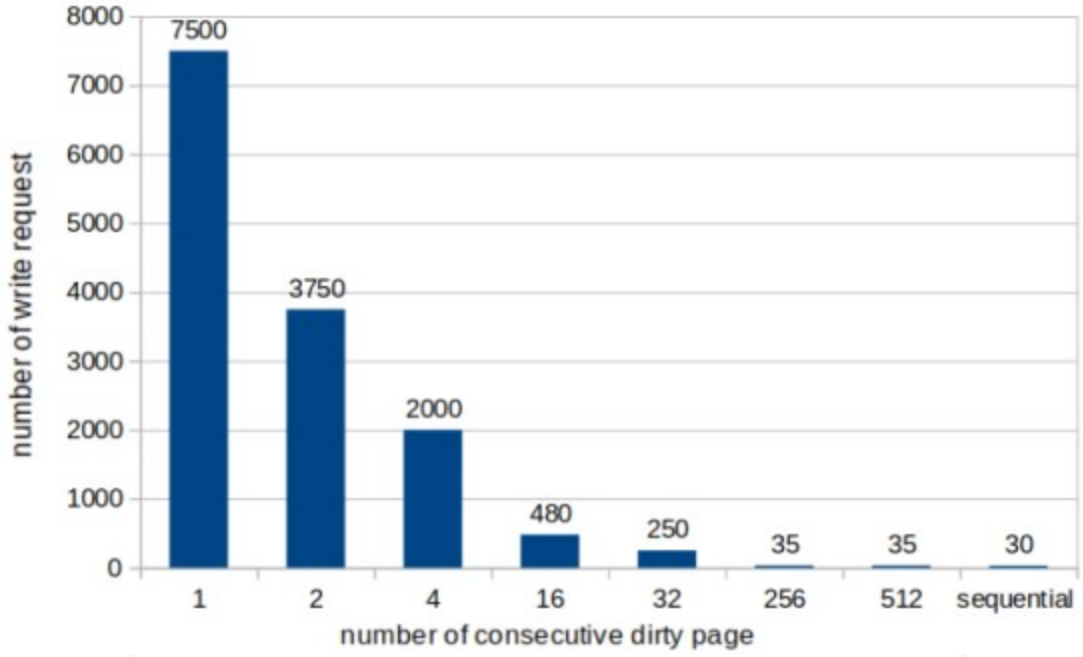
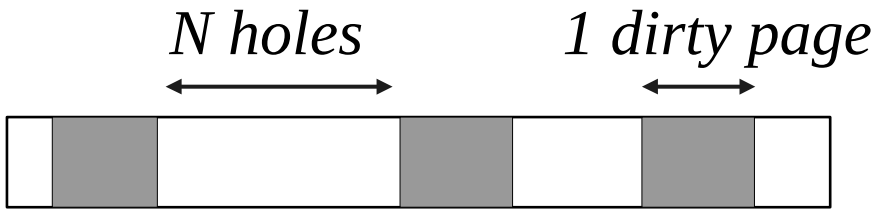
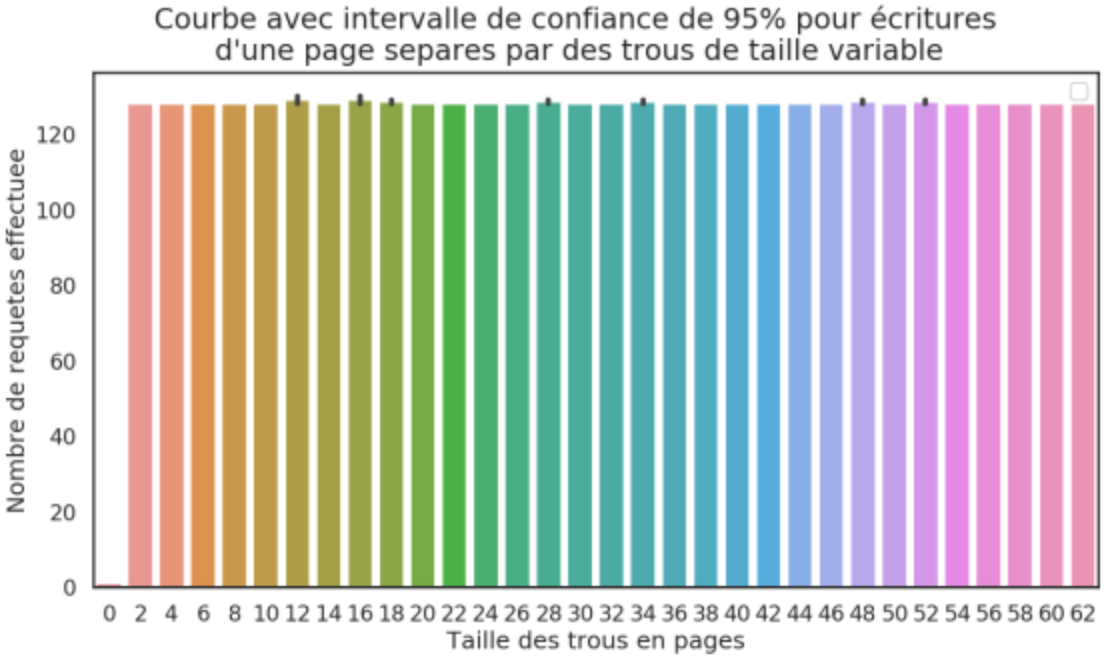


FIGURE 2.2 – Number of write requests(after merges) completed per second for the device according to the number of consecutive dirty page.

Ecritures sequentielles vs espacées

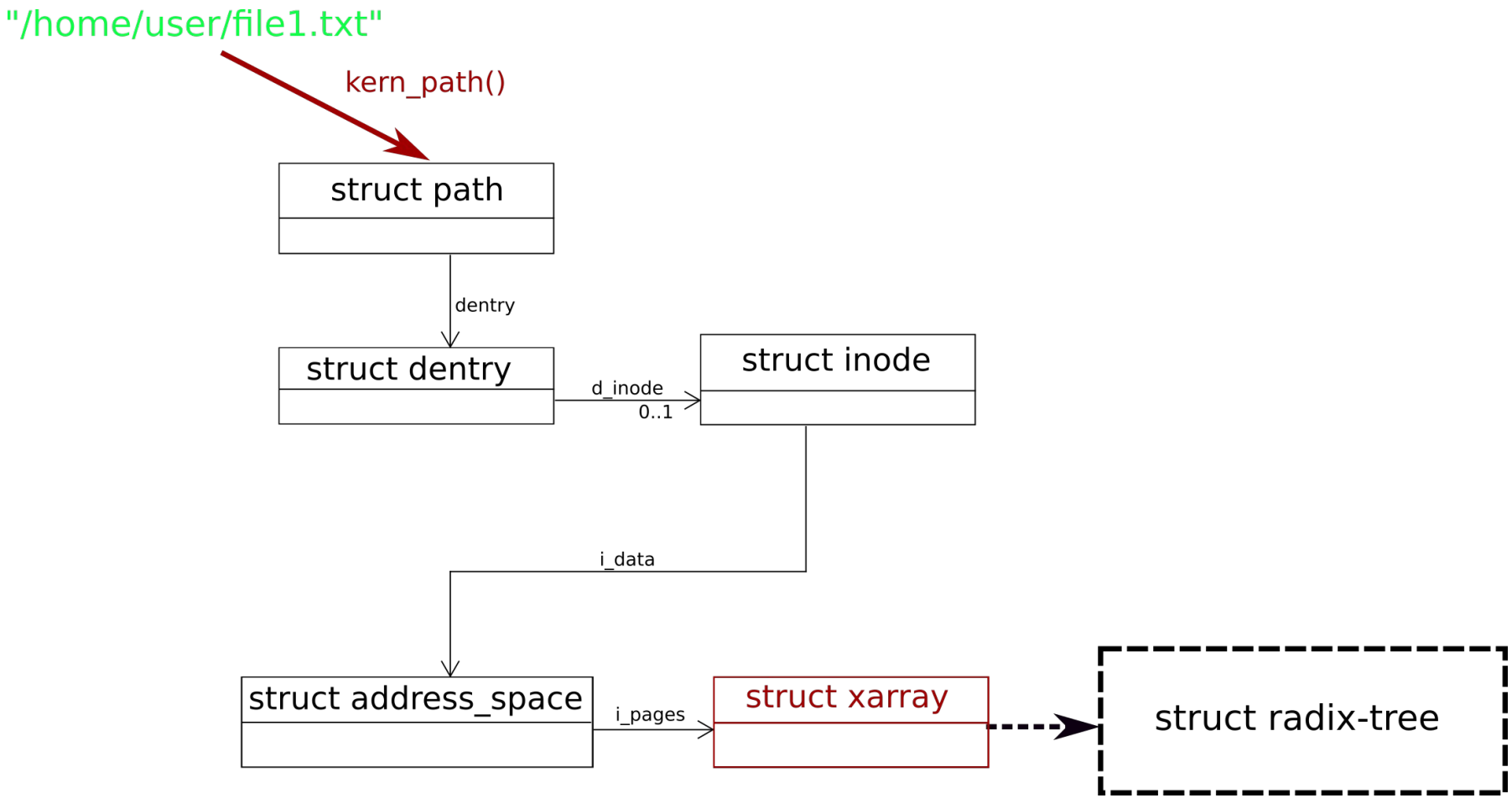
- Ecritures espacées





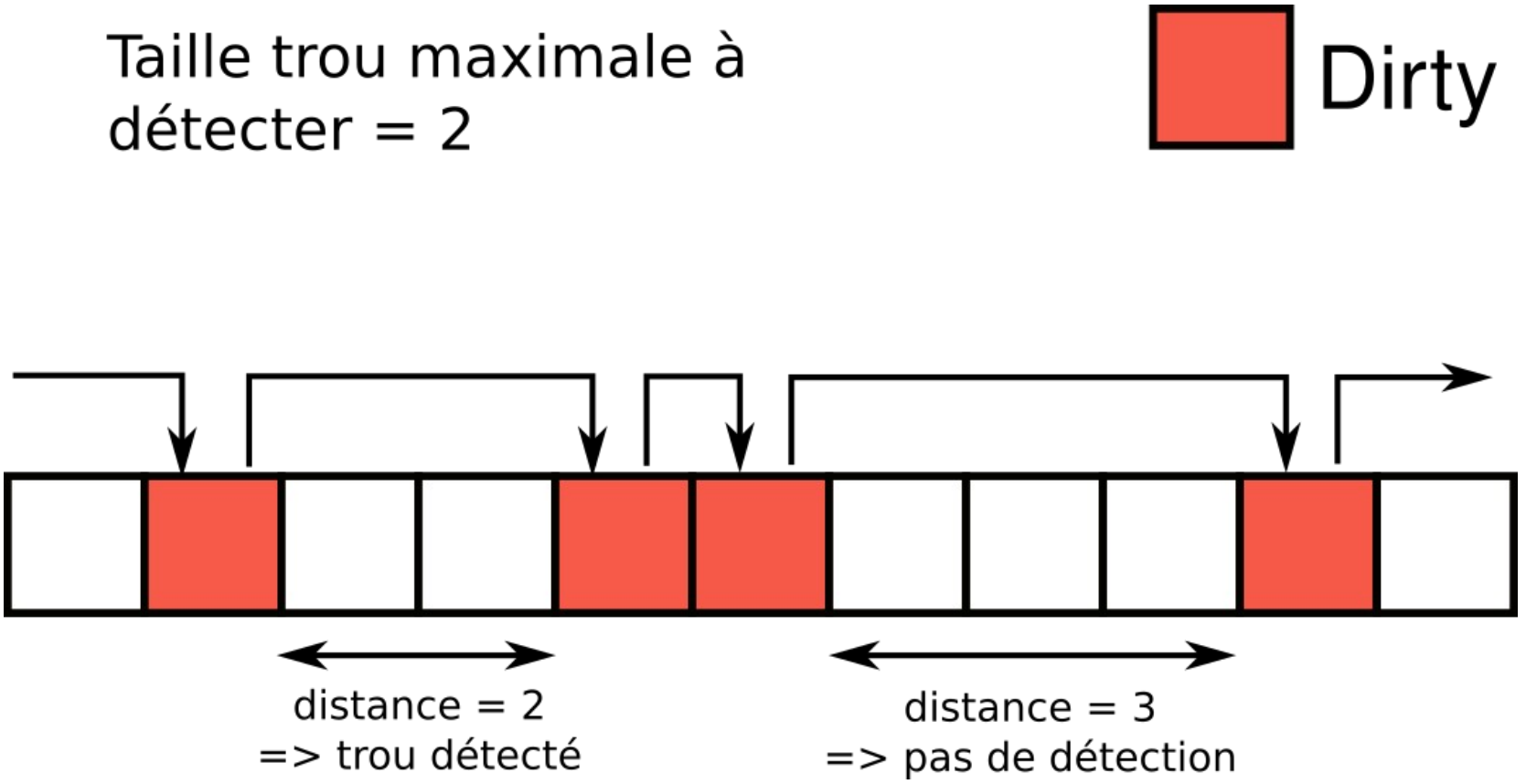
Page cache et detection de vides

- Recuperation page cache



Page cache et detection de vides

- Exemple - schéma

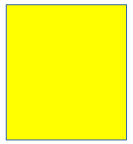


Réalisation - Patch

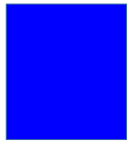
- Deux idées :
  - Utilisation du tag TOWRITE
    - Le noyau tag toutes les pages DIRTY avec le tag TOWRITE
    - On se repose sur le noyau
  - Modification de la politique d’écriture du noyau
    - Changement a l’algorithme vu précedemment
    - Plus bas niveau
- On choisit d’utiliser la 1ère méthode

Réalisation - Patch

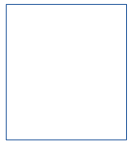
- Algorithme 1ère version



DIRTY

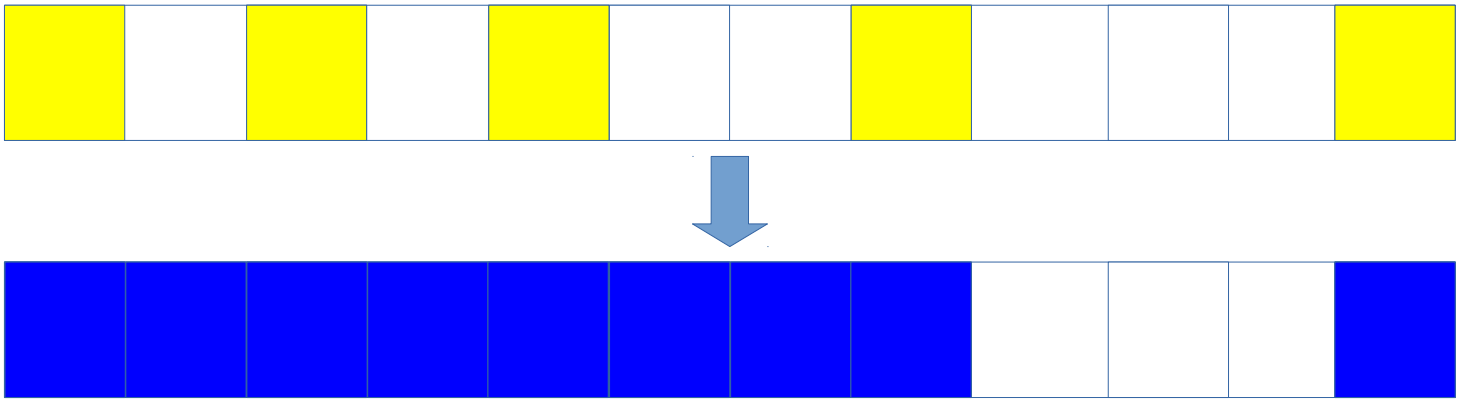


TOWRITE



CLEAN

  - Idée : Utilisé le tag TOWRITE.
  - Utilisation de deux curseurs.
  - Le premier curseur parcours les pages dirty
  - Le deuxième rejoint le premier
  - On laisse ensuite le noyau faire le reste du travail

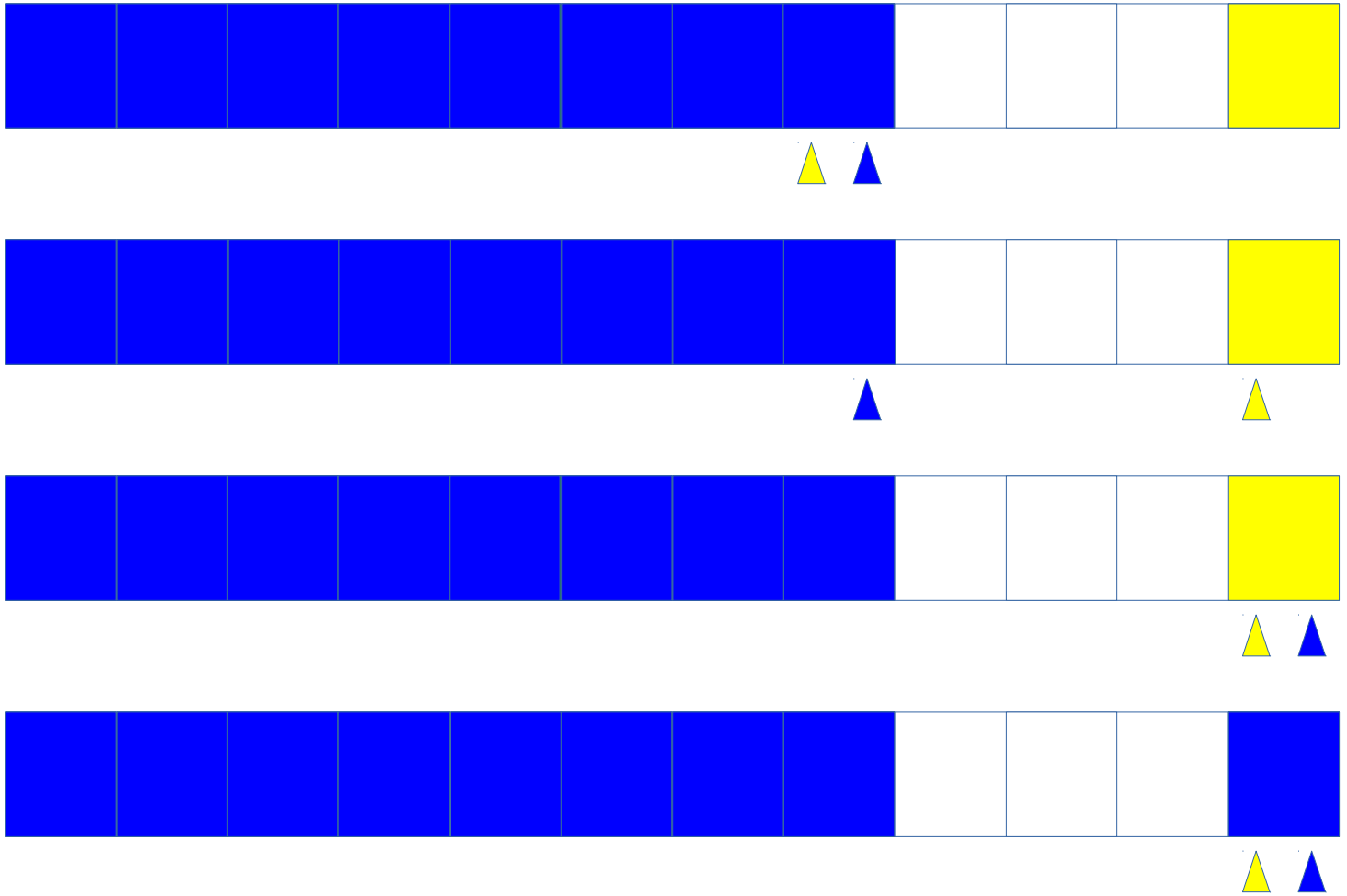


Réalisation - Patch

- Algorithme 1ère version

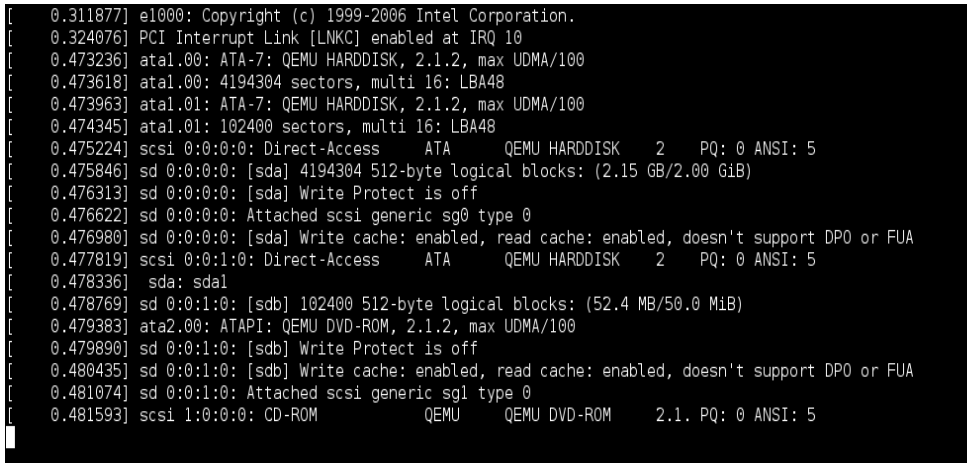
Réalisation - Patch

- Algorithme 1ère version



Réalisation - Patch

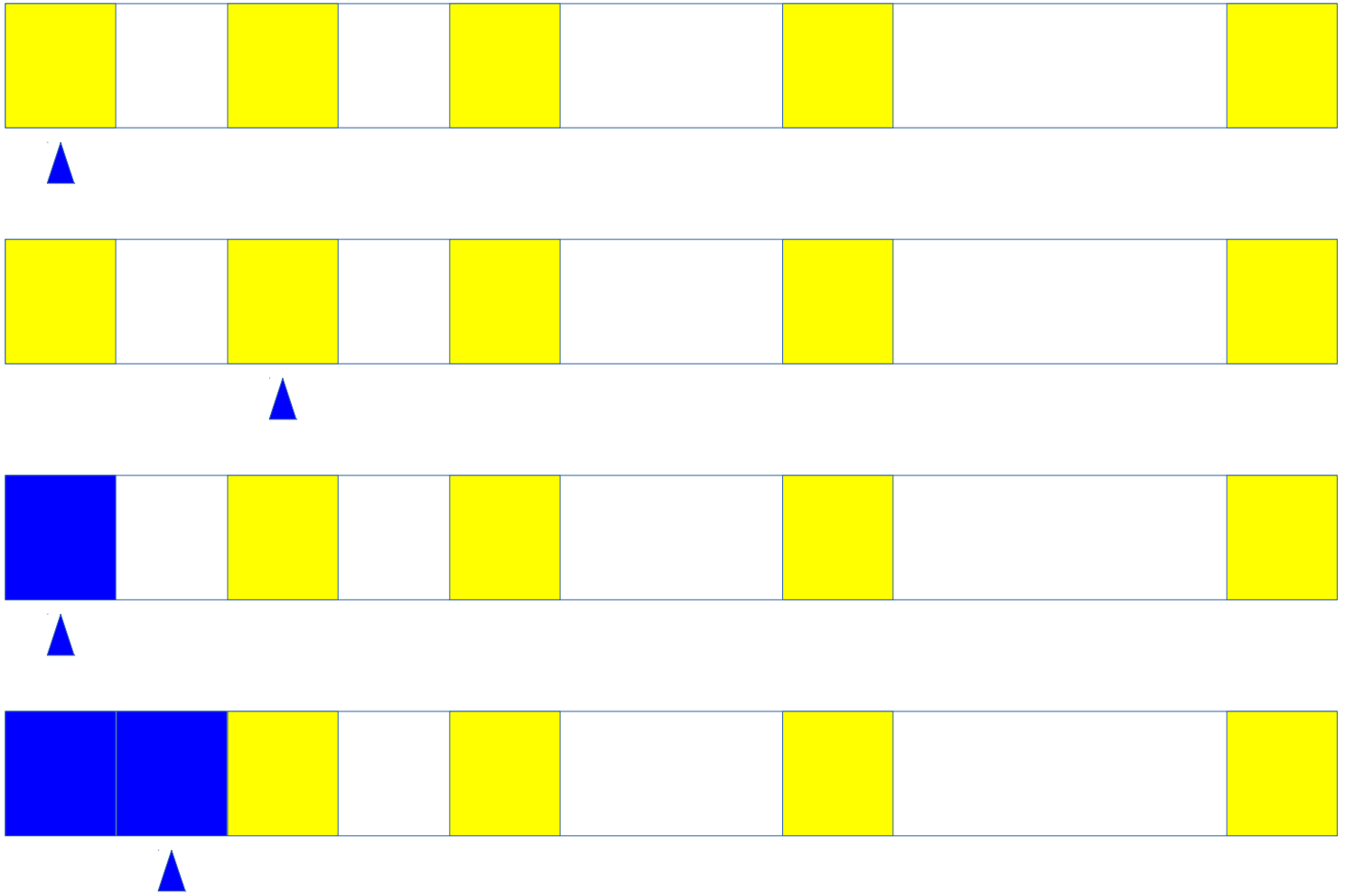
- Bilan 1ère solution :
  - Problème de locks, ne boot pas.



  - Enlever le lock interne ?
    - → l’algo ne fonctionne pas non plus
  - On est obligé de n’utiliser qu’un seul curseur
    - → On adapte la première version

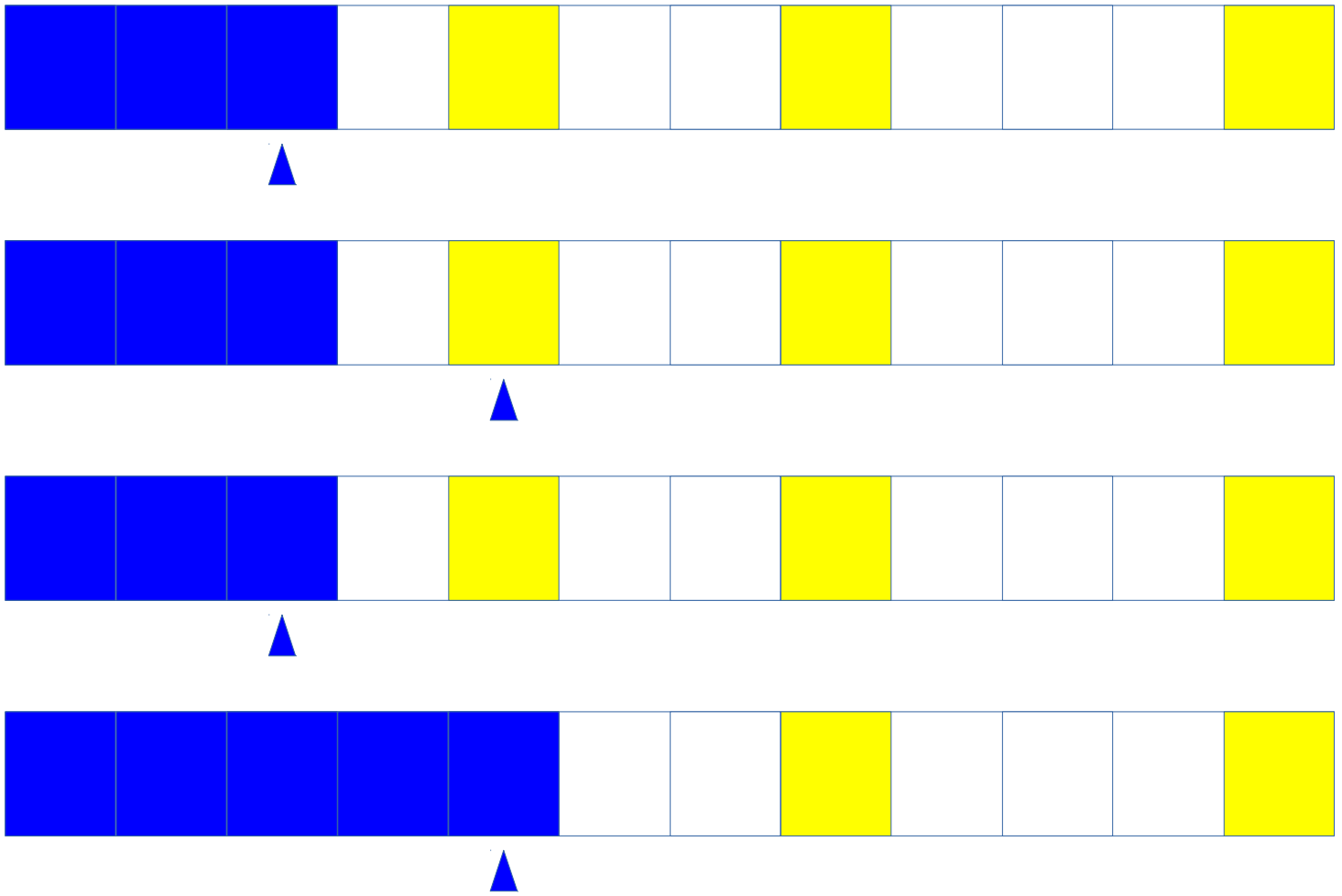
Réalisation - Patch

- Algorithme 2ème version



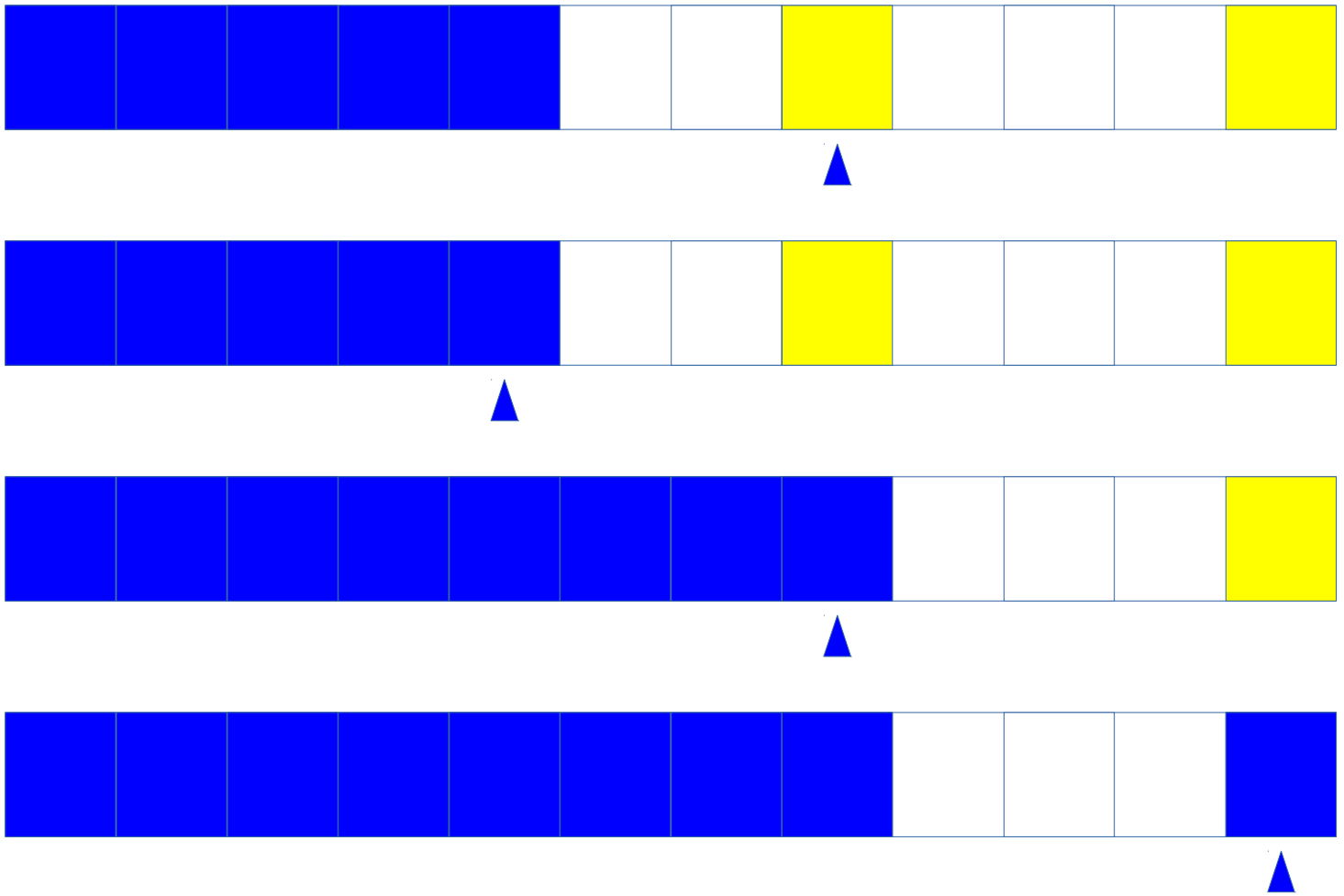
Réalisation - Patch

- Algorithme 2ème version



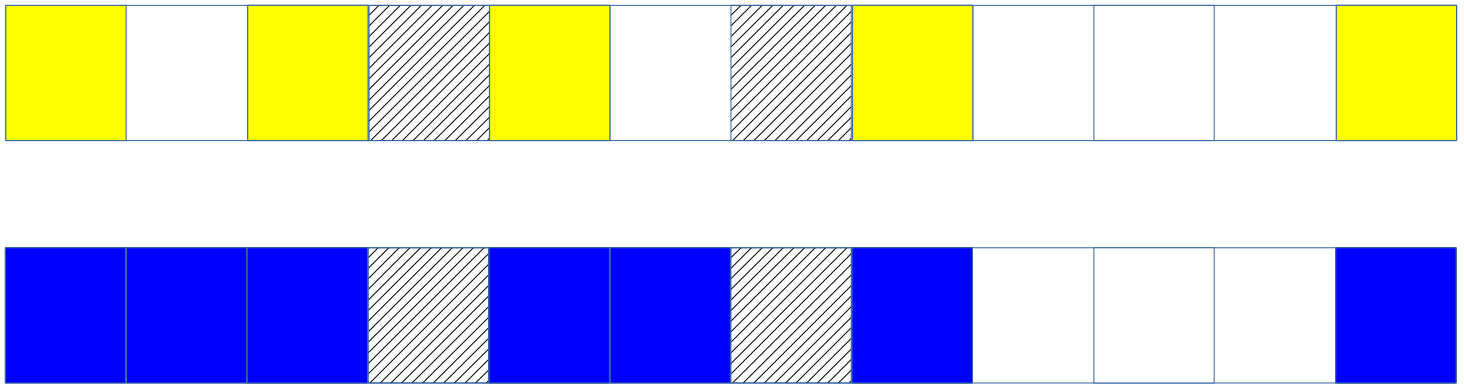
Réalisation - Patch

- Algorithme 2ème version



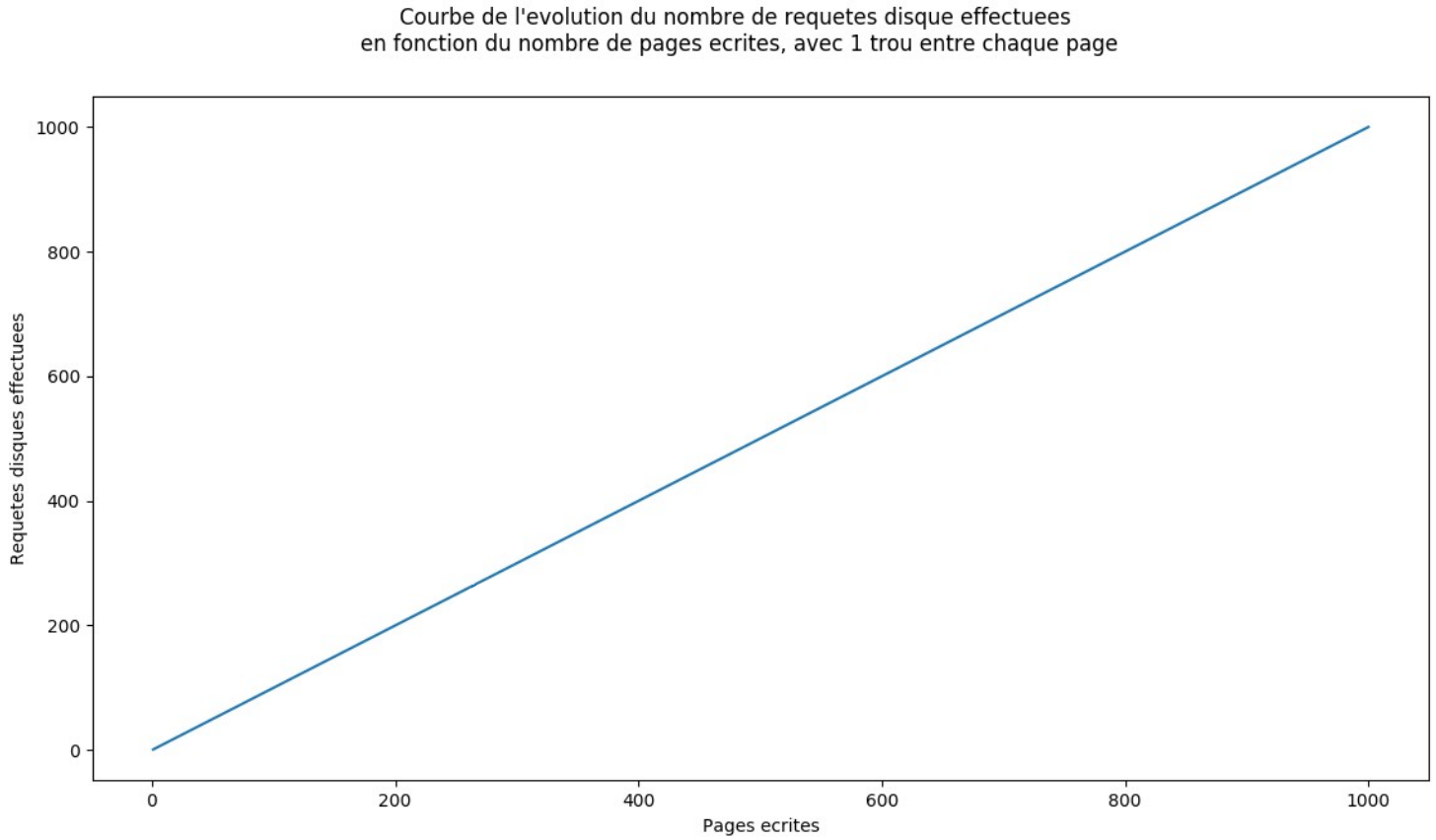
Réalisation - Patch

- Avantages
  - Algorithme simple
- Inconvénient
  - Problème si les pages sont manquantes



Réalisation - Patch

- Résultats



- Ce n’est pas le résultat attendu

Réalisation - Patch

- Conjecture
  - 3 raisons d’écrire une page :
    - Sync
    - Timeout
    - Libération de mémoire
- Nous n’avons tester que le premier point
- Amélioration possible :
    - Correction du patch
    - Lecture des pages qui ne sont pas dans le page cache pour combler les trous.

Conclusion

MERCI DE VOTRE ATTENTION