

*Soutenance de projet*  
*04 Juin 2019*

*LIRE POUR ECRIRE*

*Master 1 Informatique - SAR*  
*Systemes et Applications Répartis*

*Etudiants :*

- Nathan MAURICE
- Clément TERRY
- Smail AIDER

*Encadrant :*

- Julien SOPENA

# PLAN

## *I. Présentation*

- i. I/O – Page cache
- ii. Problématique

## *II. Objectifs*

## *III. Benchmarks*

- i. Ecritures séquentielles vs espacées
- ii. Page cache et détection de vides

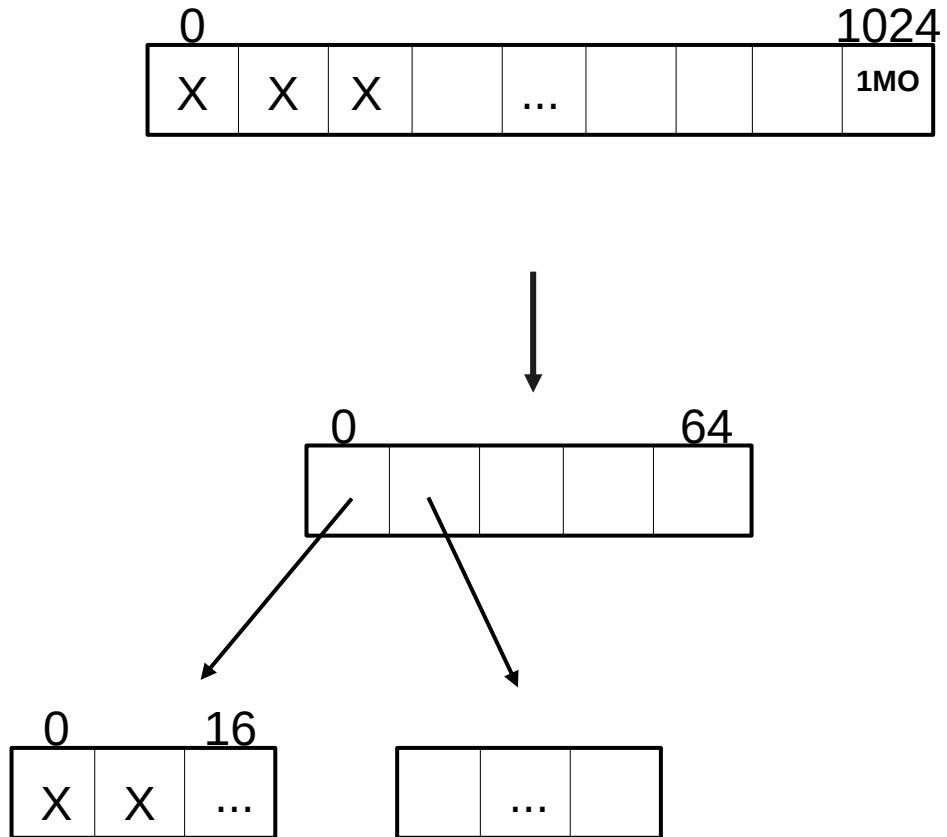
## *IV. Réalisation - Patch*

- i. Algorithmes
- ii. Résultats

## *V. Conclusion*

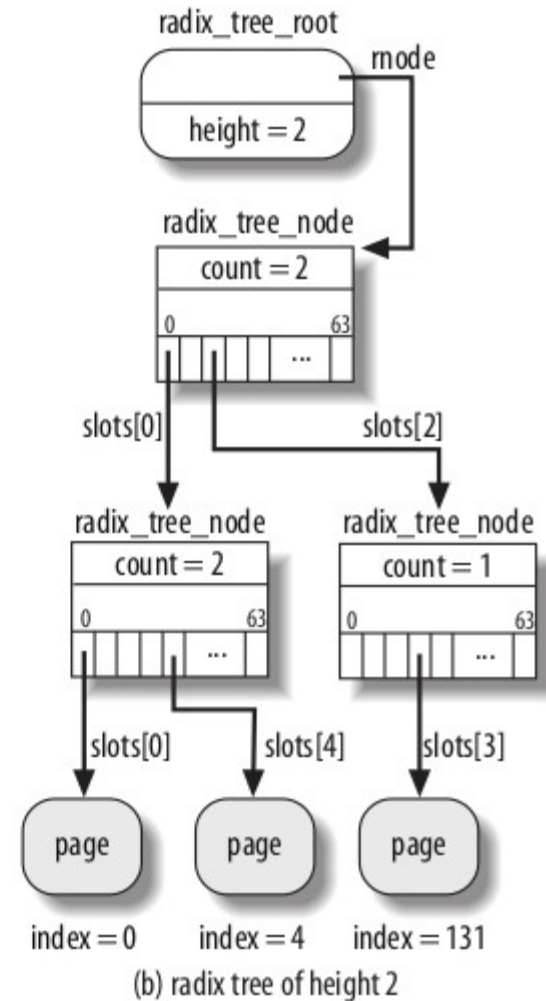
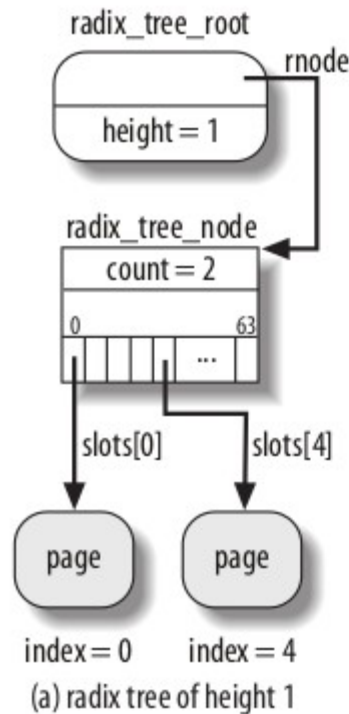
# I/O – Page Cache

## ➤ Radix Tree



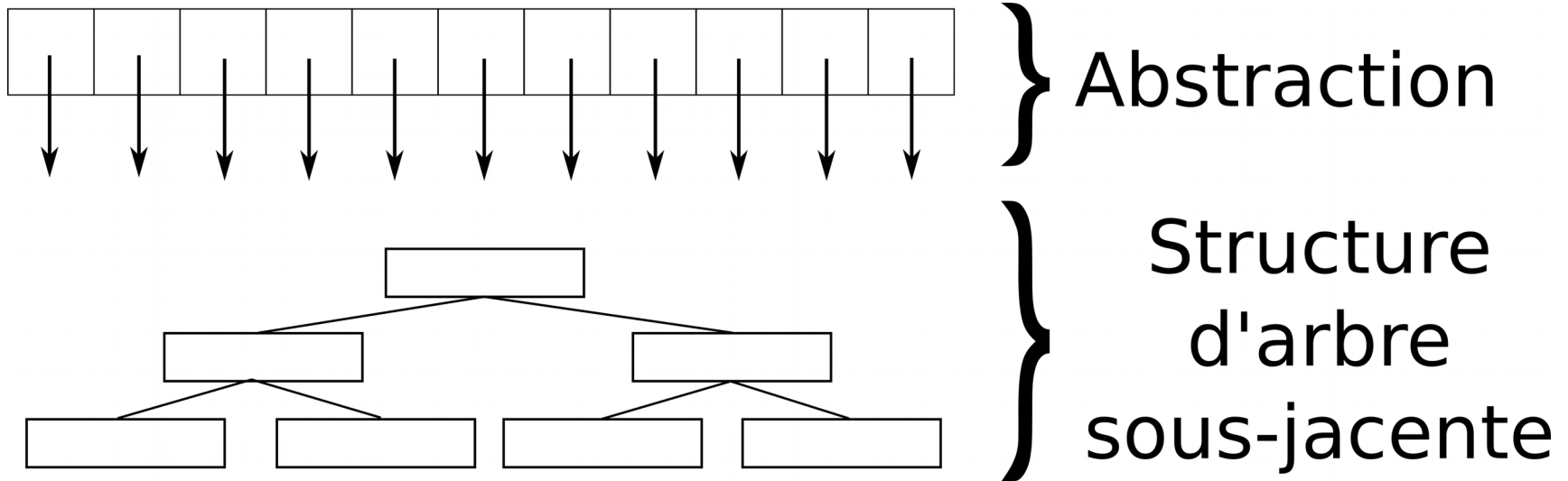
# I/O – Page Cache

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



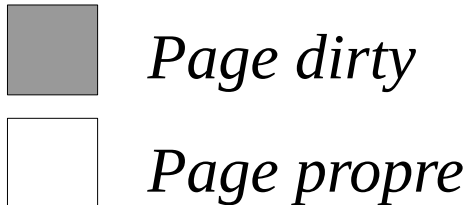
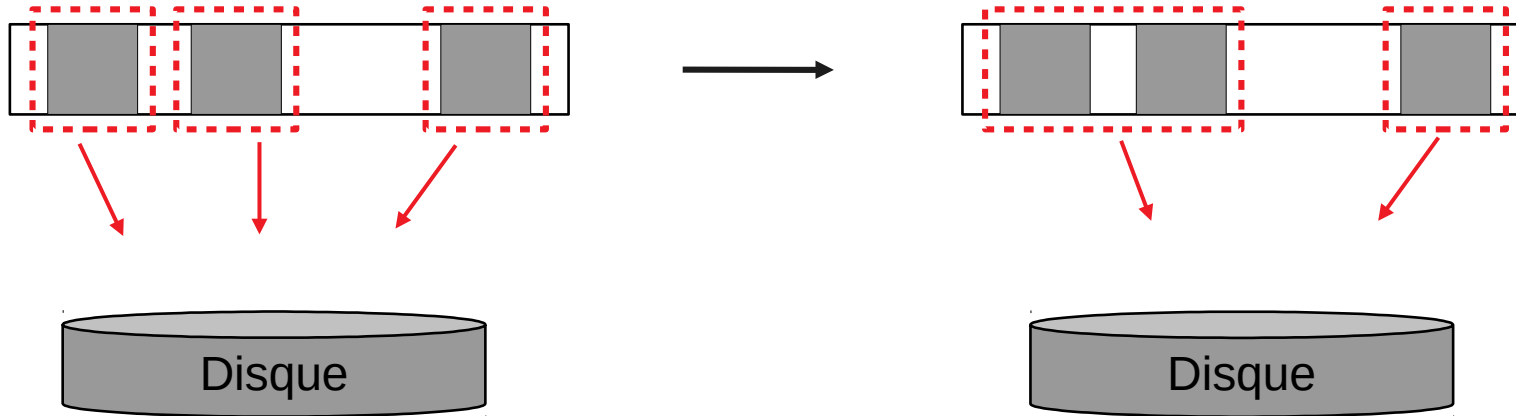
# I/O – Page Cache

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



# Problématique

- Regrouper les blocs dirty non adjacents !



# 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

# Écritures séquentielles vs espacées

## ➤ Écritures séquentielles

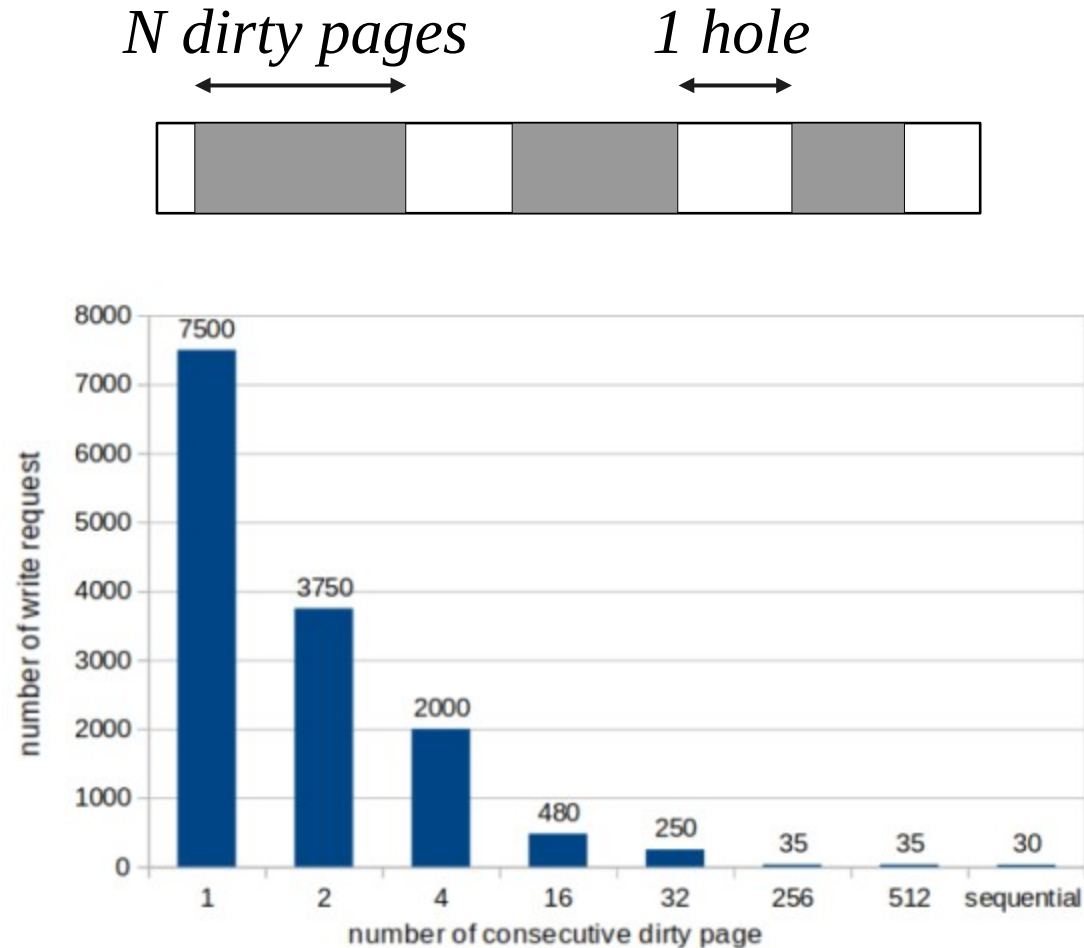
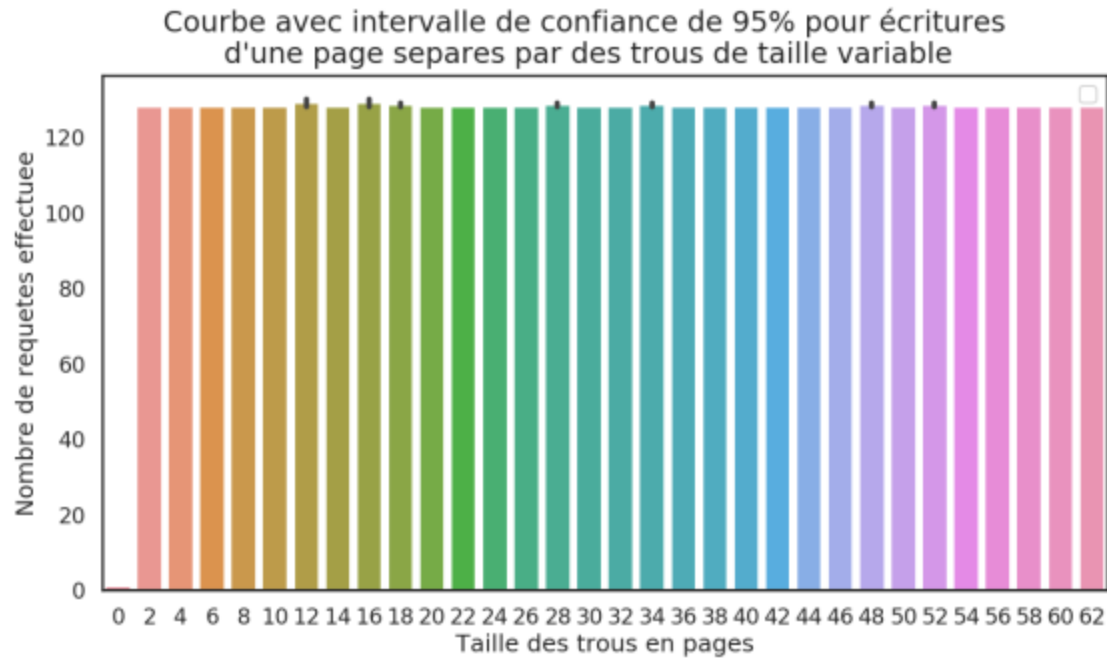
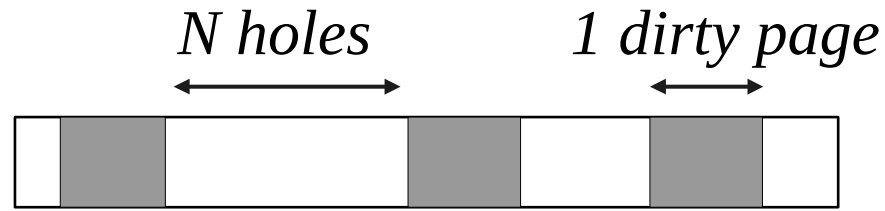


FIGURE 2.2 – Number of write requests(after merges) completed per second for the device according to the number of consecutive dirty page.



# Écritures séquentielles vs espacées

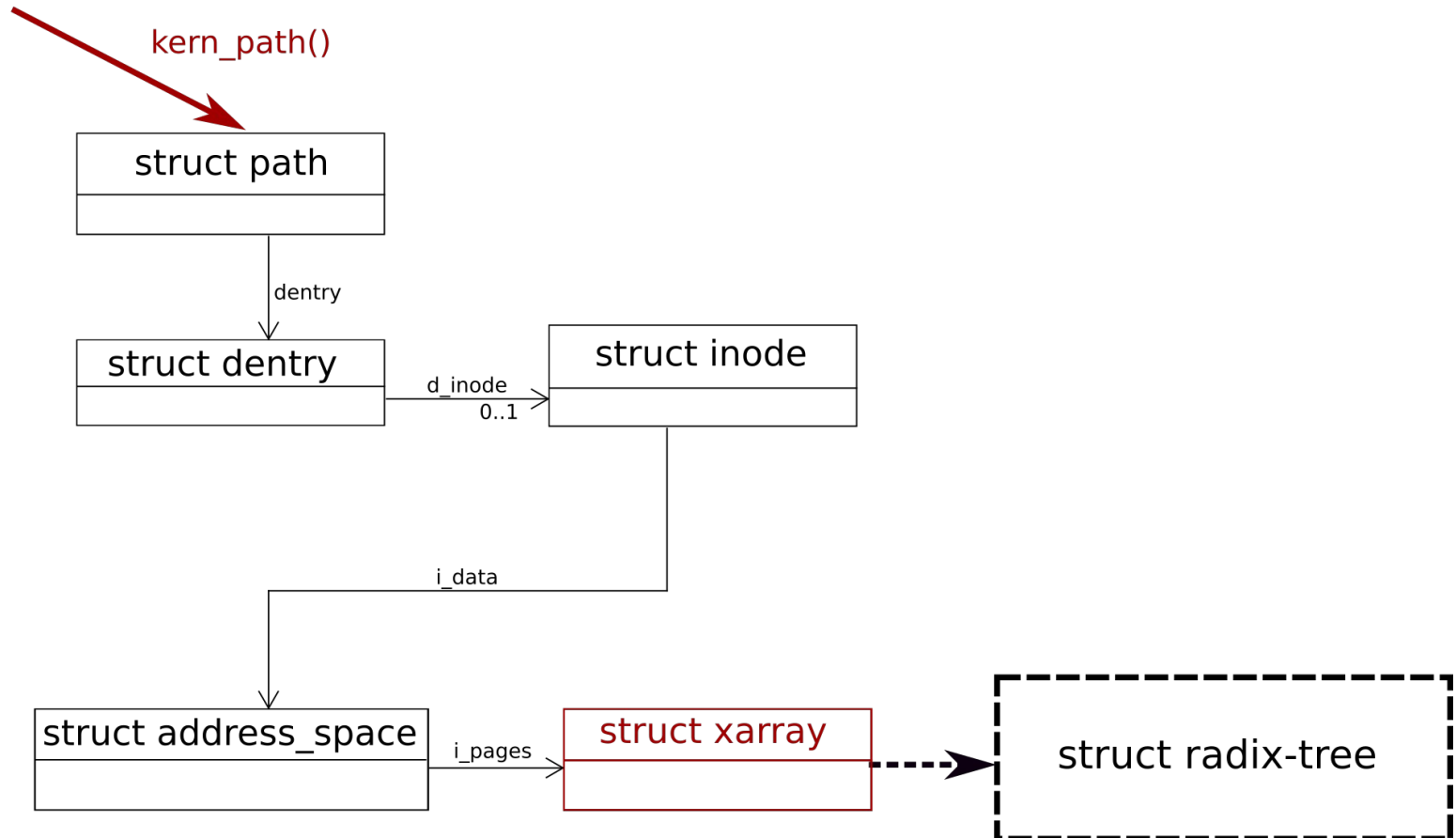
## ➤ Écritures espacées



# Page cache et detection de vides

## ➤ Recuperation page cache

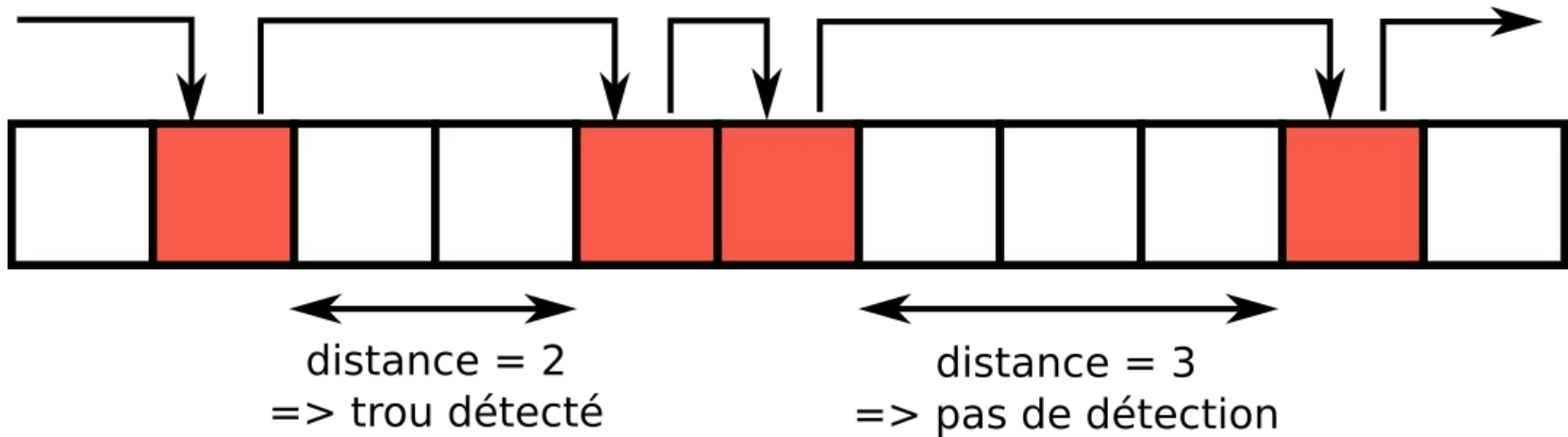
"/home/user/file1.txt"



# Page cache et detection de vides

## Exemple - schéma

Taille trou maximale à détecter = 2



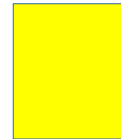
# Réalisation - Patch

- 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 précédemment
    - Plus bas niveau
- On choisit d'utiliser la 1ère méthode

# Réalisation - Patch

## ➤ Algorithme 1ère version

- 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



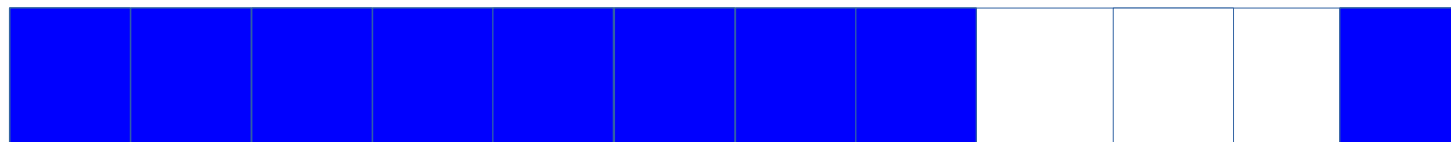
DIRTY



TOWRITE

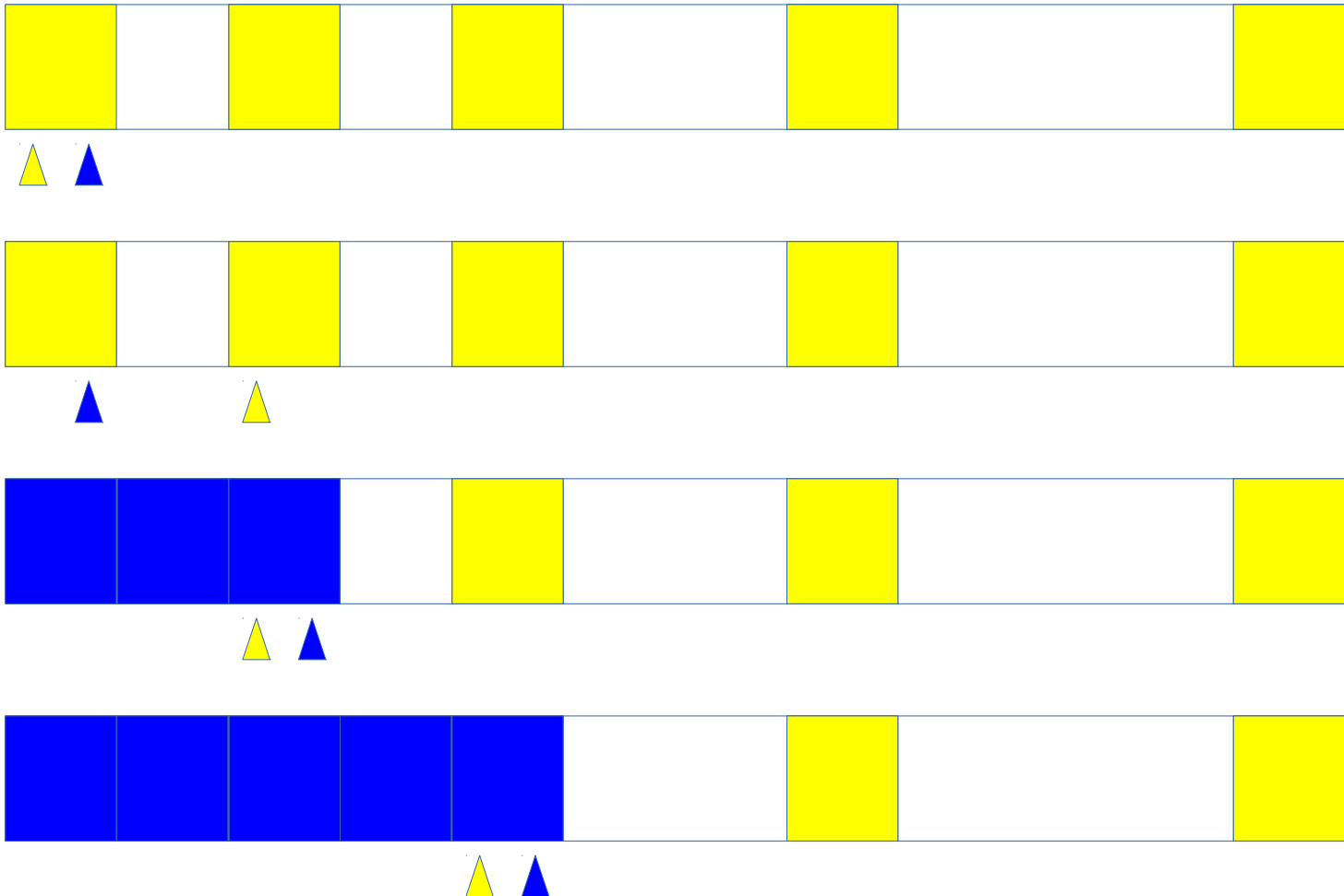


CLEAN



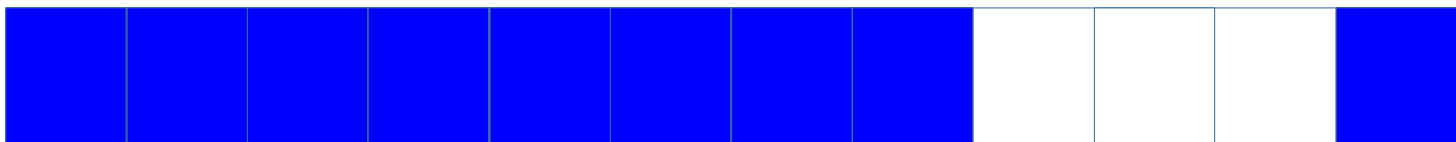
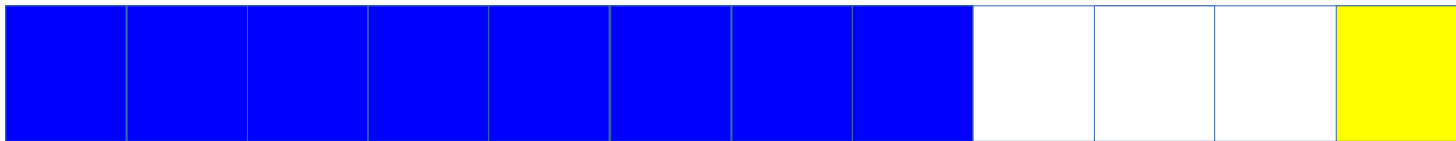
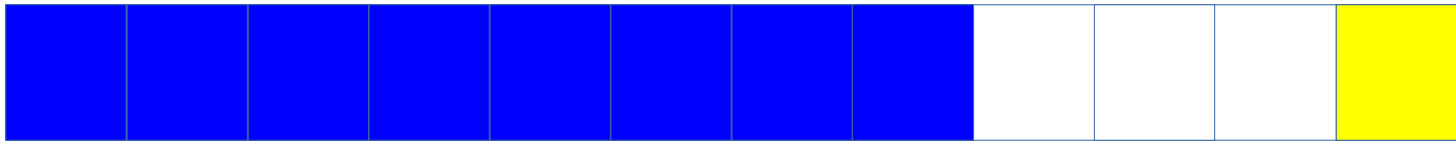
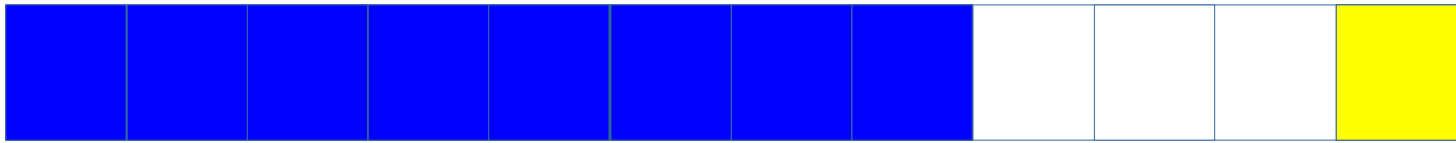
# Réalisation - Patch

## ➤ Algorithme 1ère version



# Réalisation - Patch

## ➤ Algorithme 1ère version



# Réalisation - Patch

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

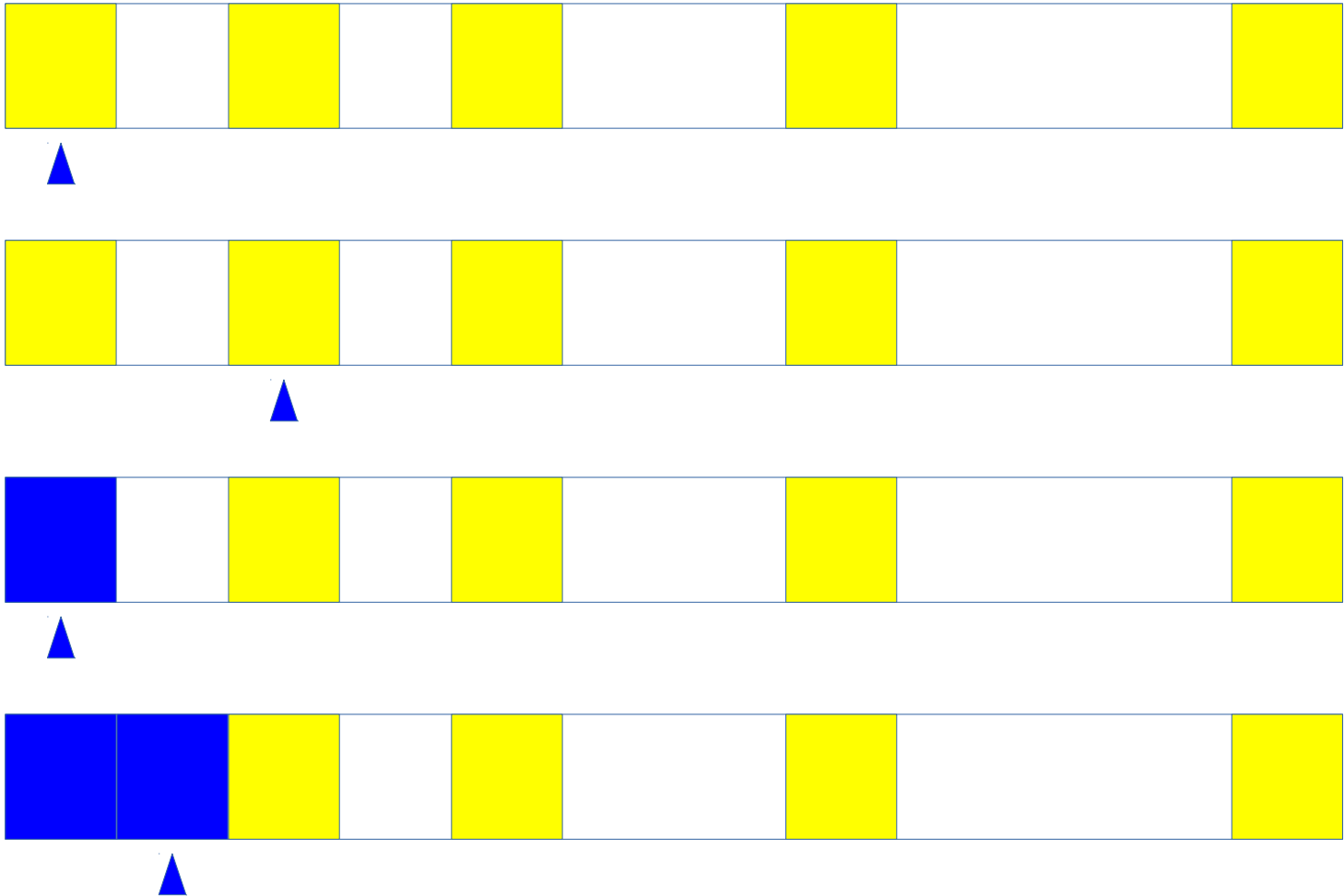
```
[ 0.311877] e1000: Copyright (c) 1999-2006 Intel Corporation.  
[ 0.324076] PCI Interrupt Link [LNKC] enabled at IRQ 10  
[ 0.473236] ata1.00: ATA-7: QEMU HARDDISK, 2.1.2, max UDMA/100  
[ 0.473618] ata1.00: 4194304 sectors, multi 16: LBA48  
[ 0.473963] ata1.01: ATA-7: QEMU HARDDISK, 2.1.2, max UDMA/100  
[ 0.474345] ata1.01: 102400 sectors, multi 16: LBA48  
[ 0.475224] scsi 0:0:0:0: Direct-Access    ATA        QEMU HARDDISK  2    PQ: 0 ANSI: 5  
[ 0.475846] sd 0:0:0:0: [sda] 4194304 512-byte logical blocks: (2.15 GB/2.00 GiB)  
[ 0.476313] sd 0:0:0:0: [sda] Write Protect is off  
[ 0.476622] sd 0:0:0:0: Attached scsi generic sg0 type 0  
[ 0.476980] sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA  
[ 0.477819] scsi 0:0:1:0: Direct-Access    ATA        QEMU HARDDISK  2    PQ: 0 ANSI: 5  
[ 0.478336] sda: sda1  
[ 0.478769] sd 0:0:1:0: [sdb] 102400 512-byte logical blocks: (52.4 MB/50.0 MiB)  
[ 0.479383] ata2.00: ATAPI: QEMU DVD-ROM, 2.1.2, max UDMA/100  
[ 0.479890] sd 0:0:1:0: [sdb] Write Protect is off  
[ 0.480435] sd 0:0:1:0: [sdb] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA  
[ 0.481074] sd 0:0:1:0: Attached scsi generic sgl type 0  
[ 0.481593] scsi 1:0:0:0: CD-ROM          QEMU        QEMU DVD-ROM    2.1. PQ: 0 ANSI: 5
```

- 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



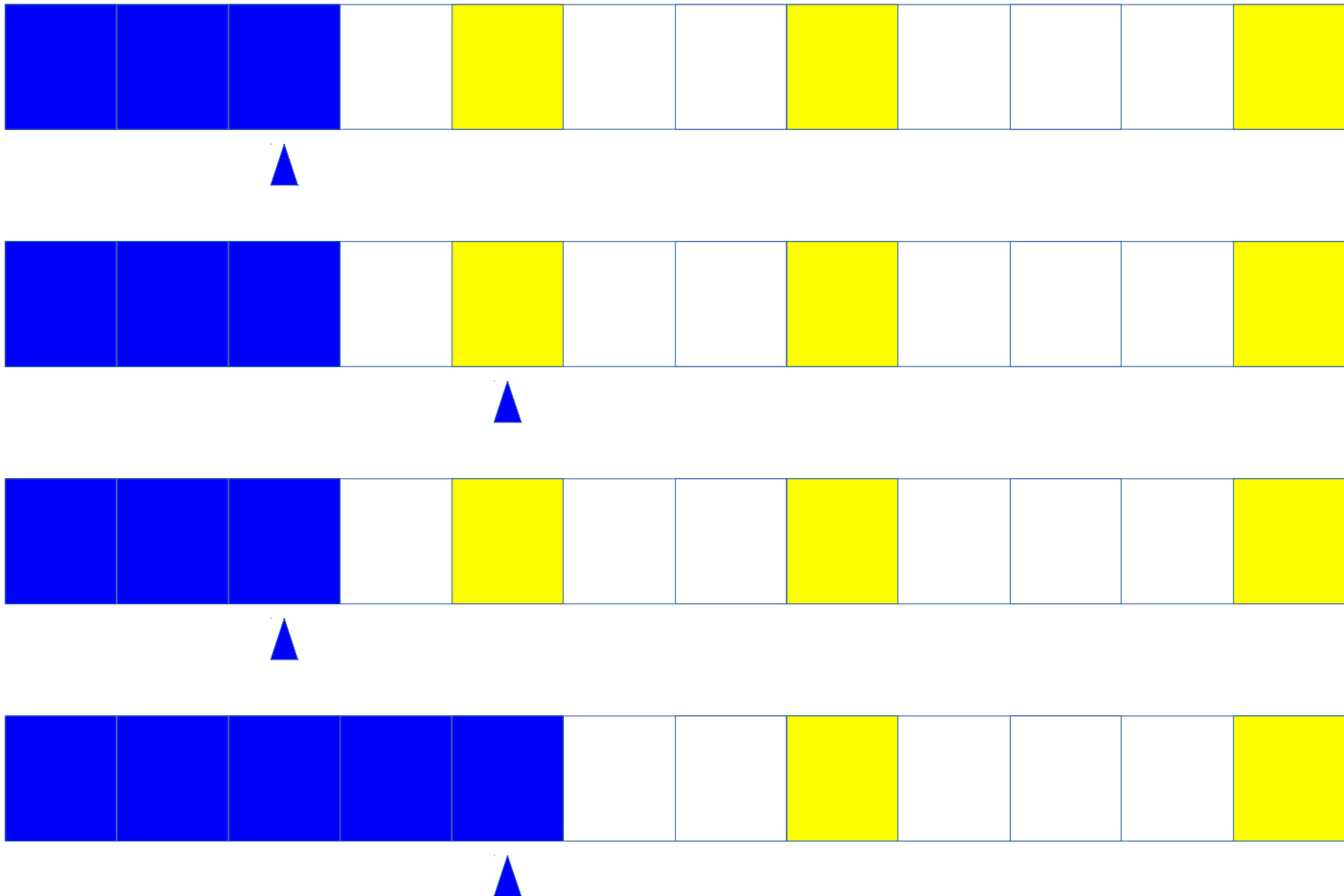
# Réalisation - Patch

## ➤ Algorithme 2ème version



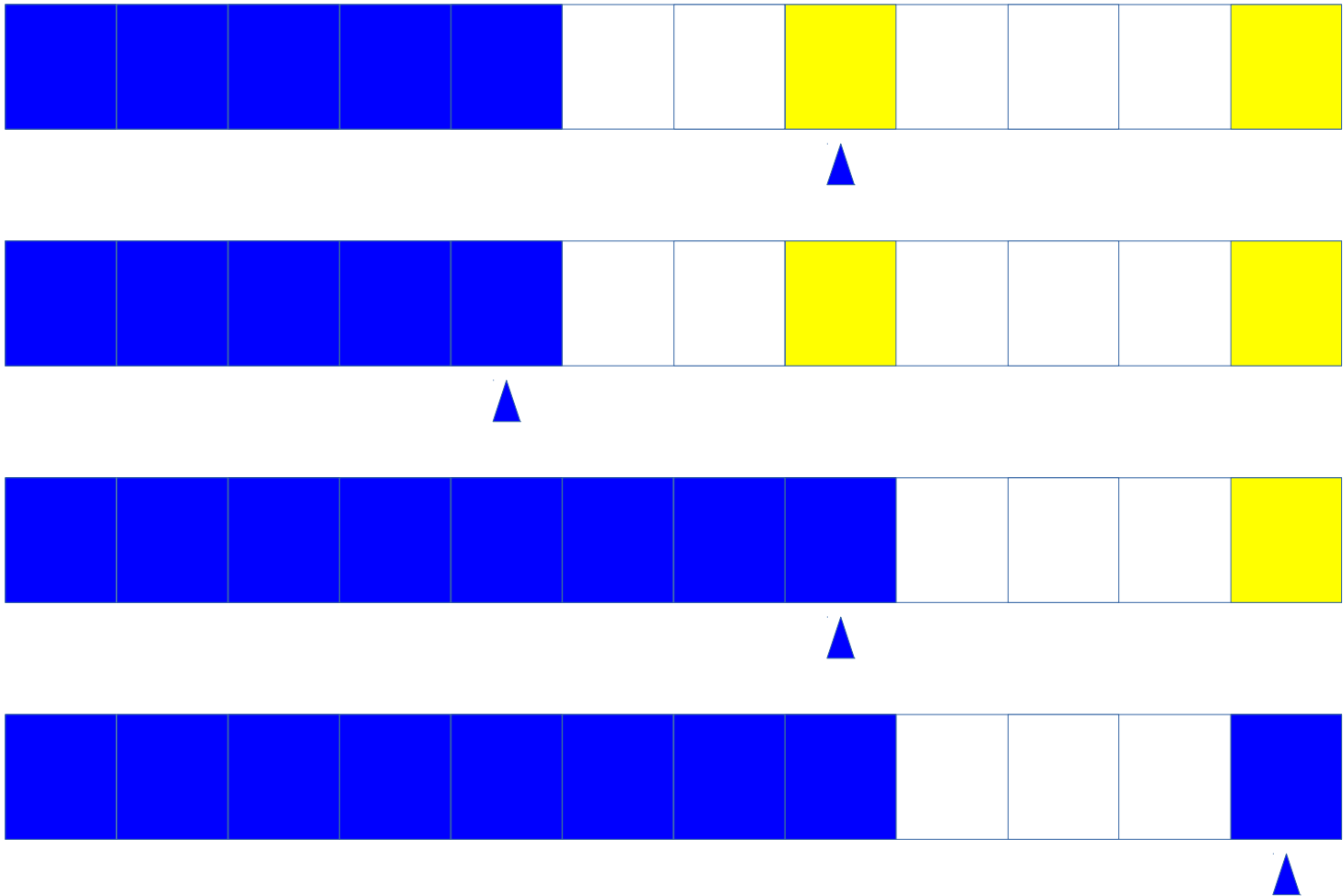
# Réalisation - Patch

## ➤ Algorithme 2ème version



# Réalisation - Patch

## ➤ Algorithme 2ème version



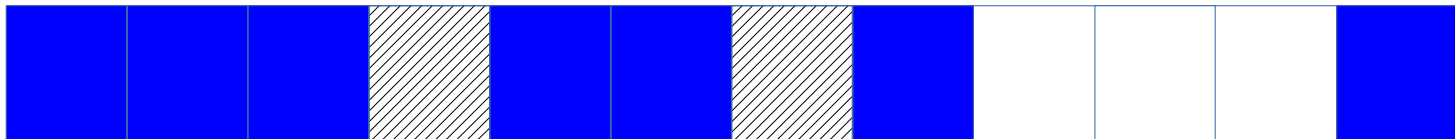
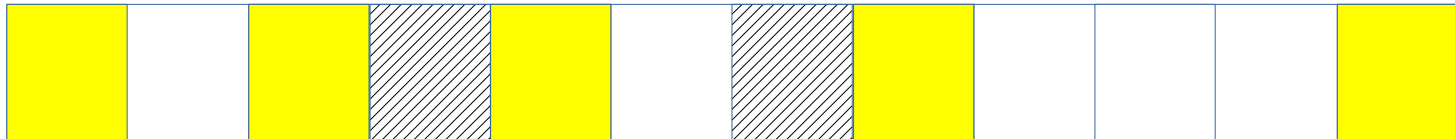
# Réalisation - Patch

## ➤ Avantages

- Algorithme simple

## ➤ Inconvénient

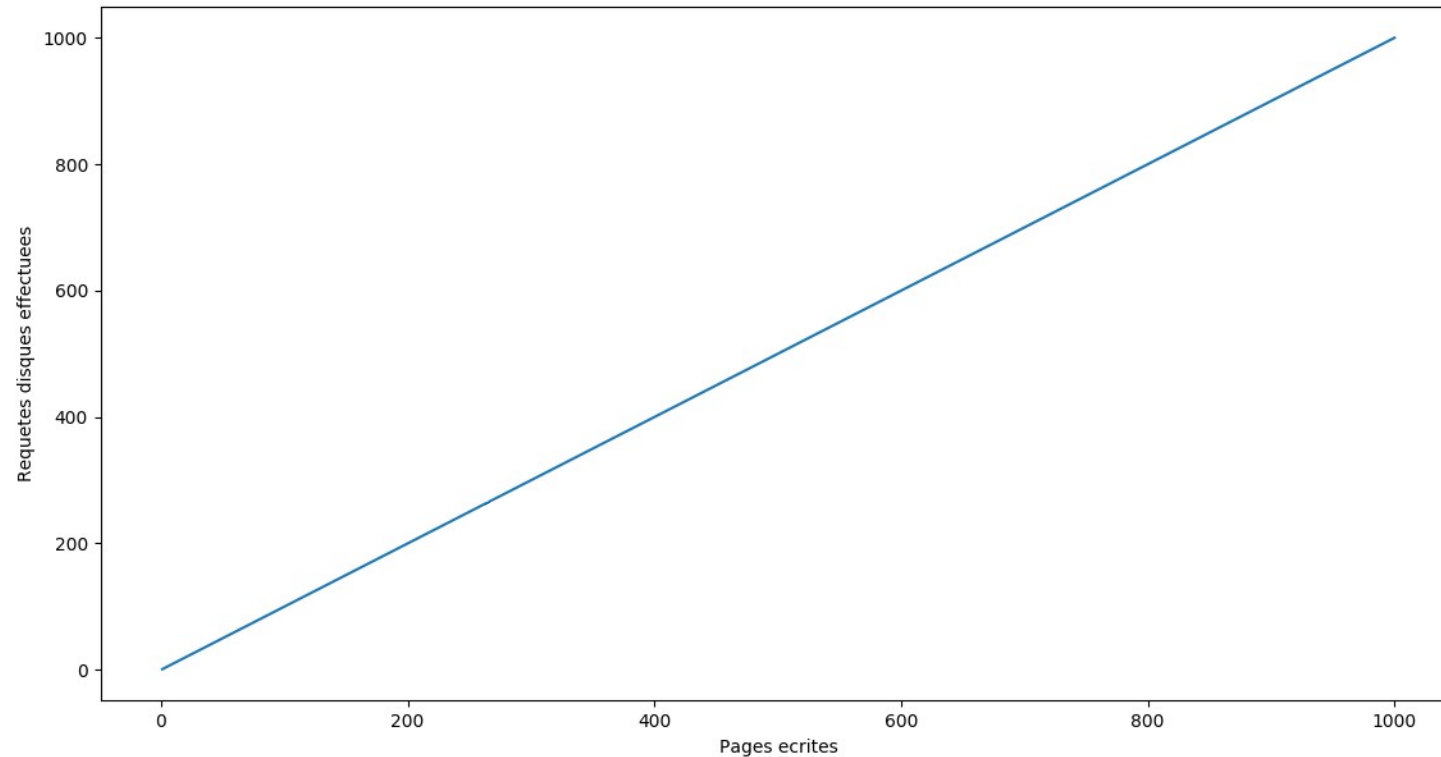
- Problème si les pages sont manquantes



# Réalisation - Patch

## ➤ Résultats

Courbe de l'evolution du nombre de requetes disque effectuees en fonction du nombre de pages ecrites, avec 1 trou entre chaque page



- 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