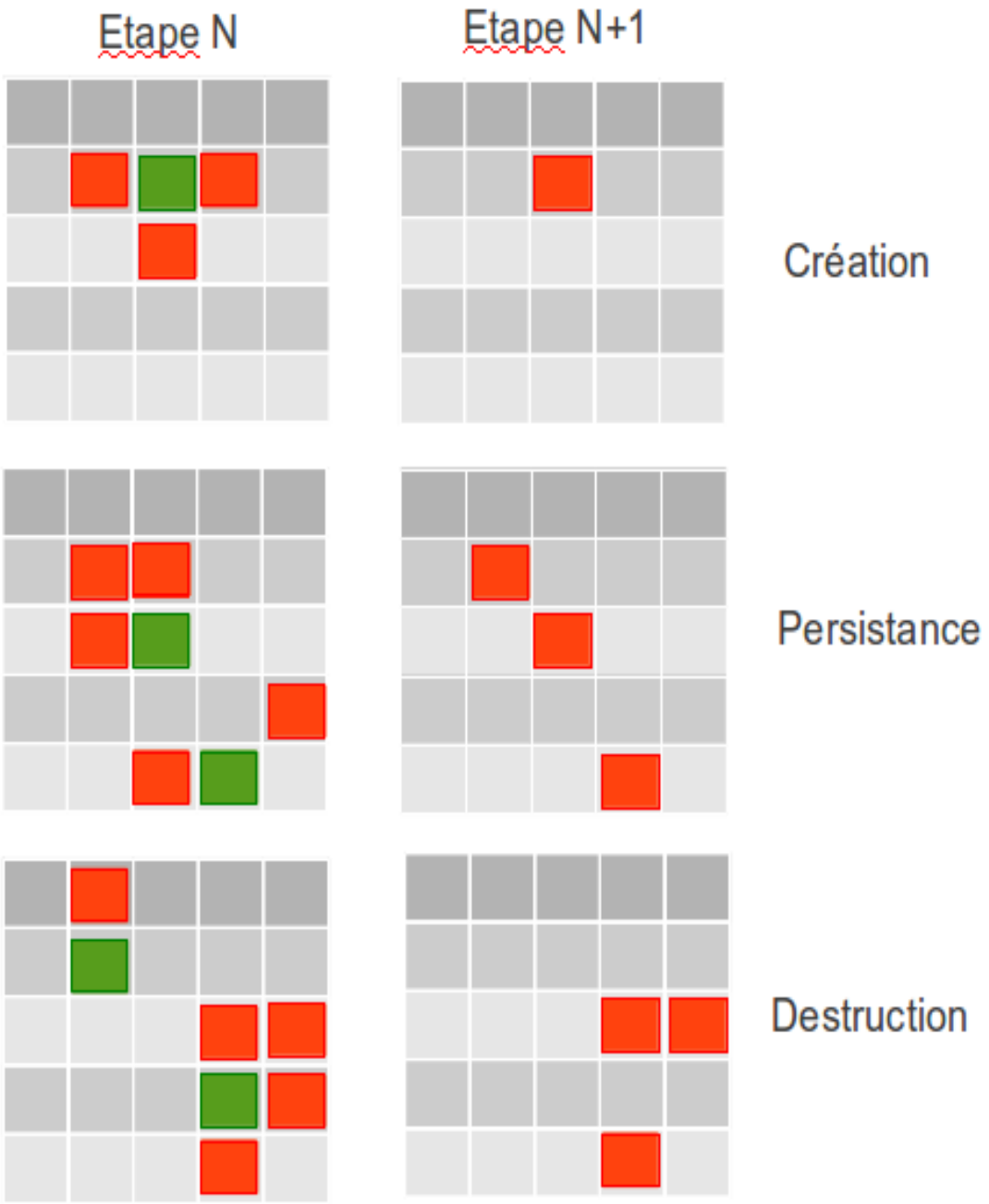
Jeux de la vie

Table des matières

Jeux de la vie

- Grille de taille quelconque

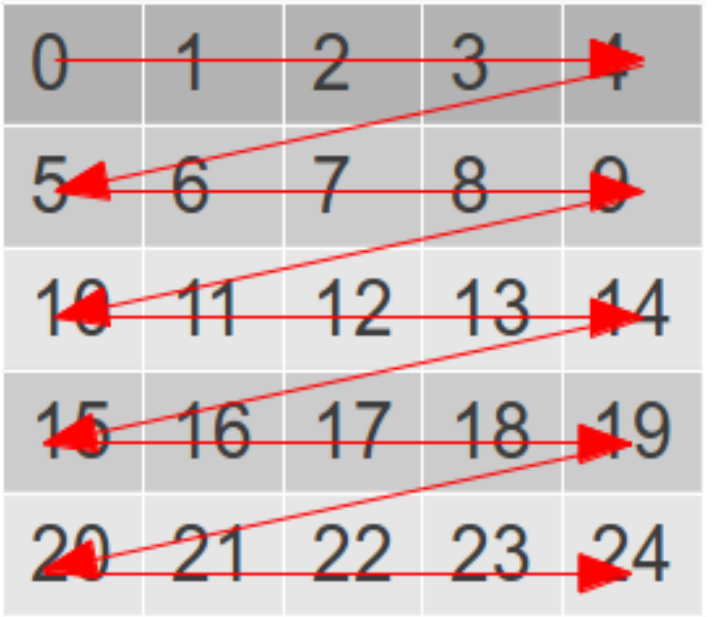
- Calcul du nombre de voisin
  - 3 → création
  - 2 → reste vivante
  - > 3 → surpopulation
  - < 2 → population



Algorithme linéaire

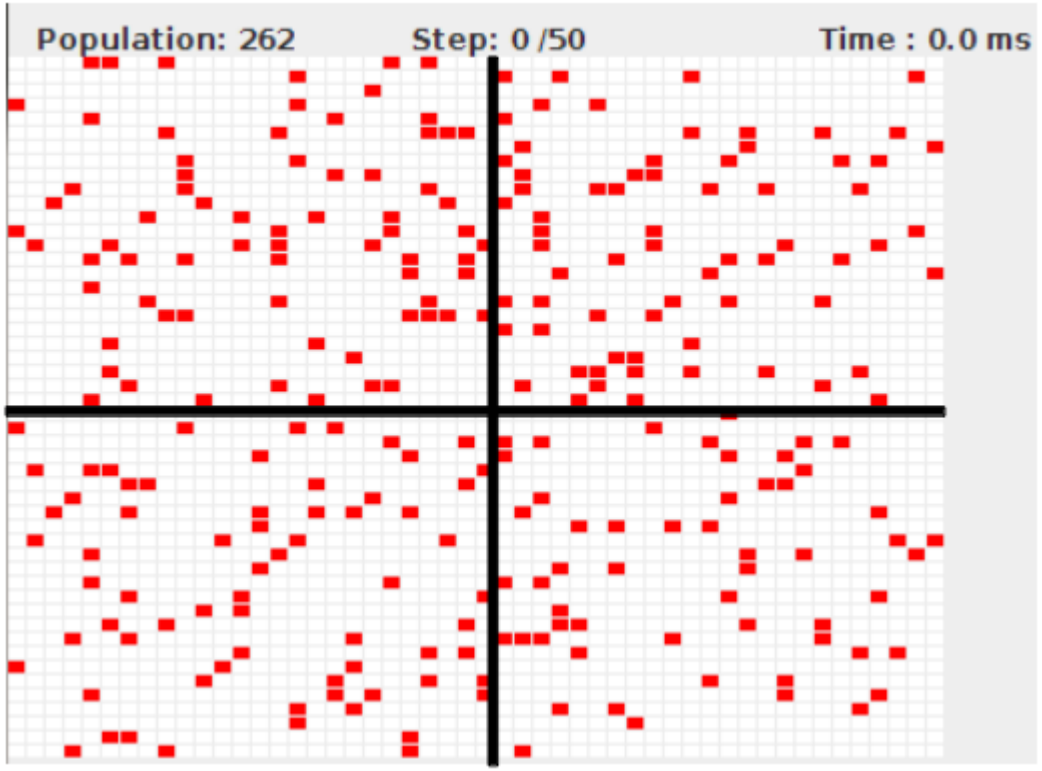
- Algorithme simple

- Efficace pour un grand nombre de case

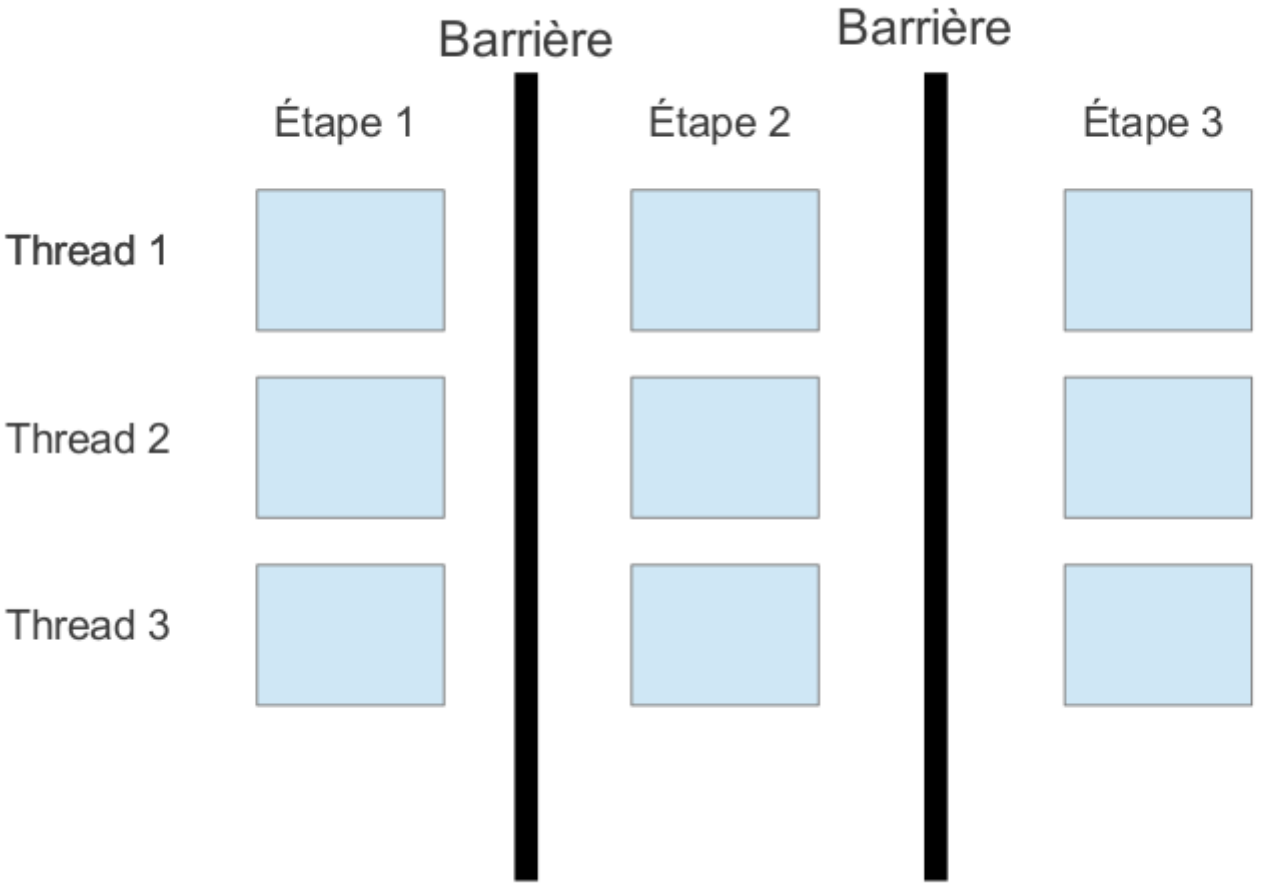


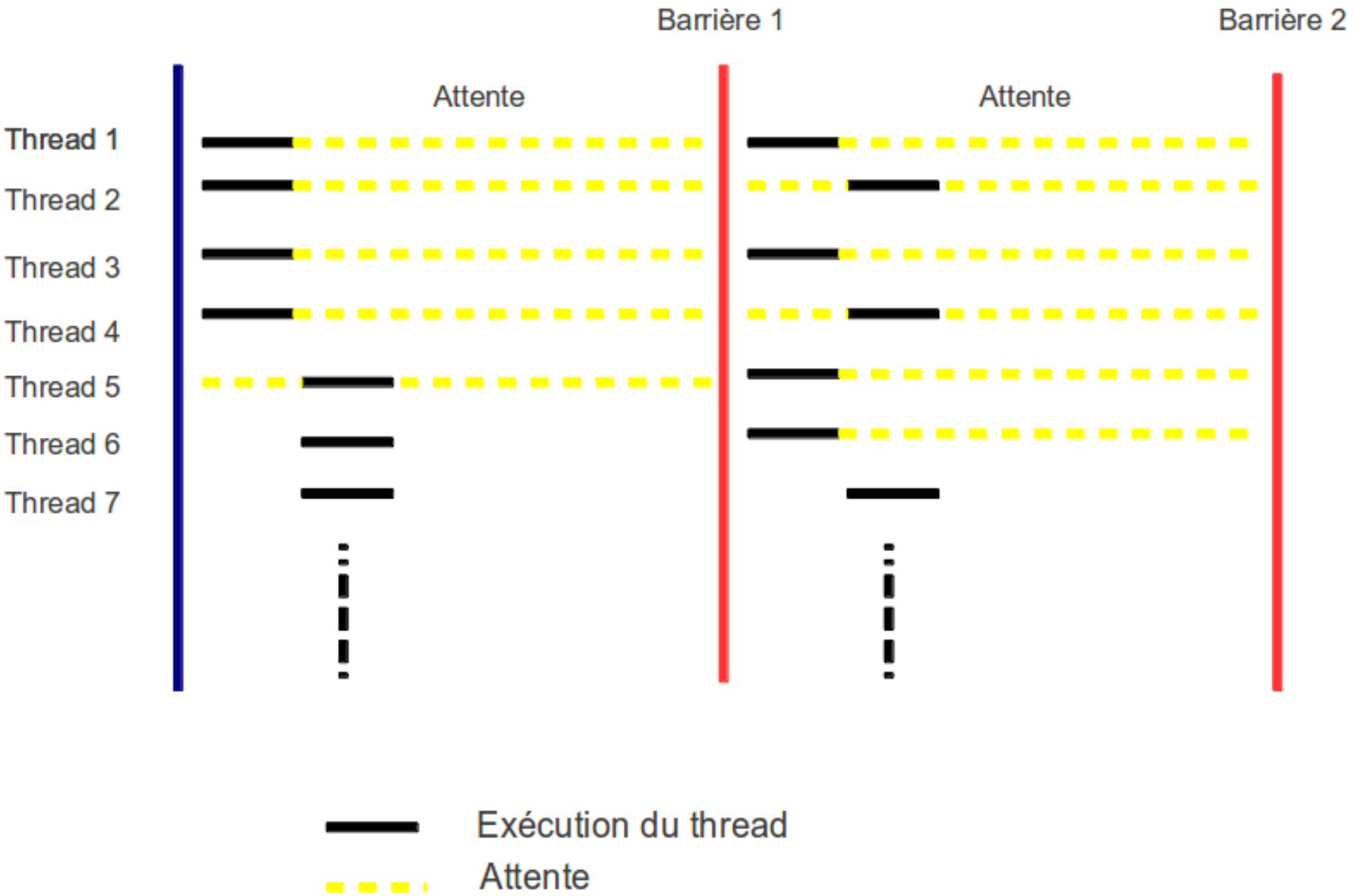
Algorithme « threadé »

- Découpage de la zone de jeux en 4.
- Création de 4 threads
- Synchronisé avec des barrières



Les barrières




Barrière 1
Barrière 2
Attente
Attente
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Exécution du thread
Attente

Les barrières

- Utilisation de CyclicBarrier.
  - java.util.concurrent
- Initialisation au nombre de case + 1
- Nécessite **nb case + 1** → **await()** pour s'ouvrir

Un thread par case

- Temps de création des barrières très lent.
- Nécessite d'initialiser la barrière à NBCase + 1.

Nombre maximum de thread

Test sur 40 000 cases

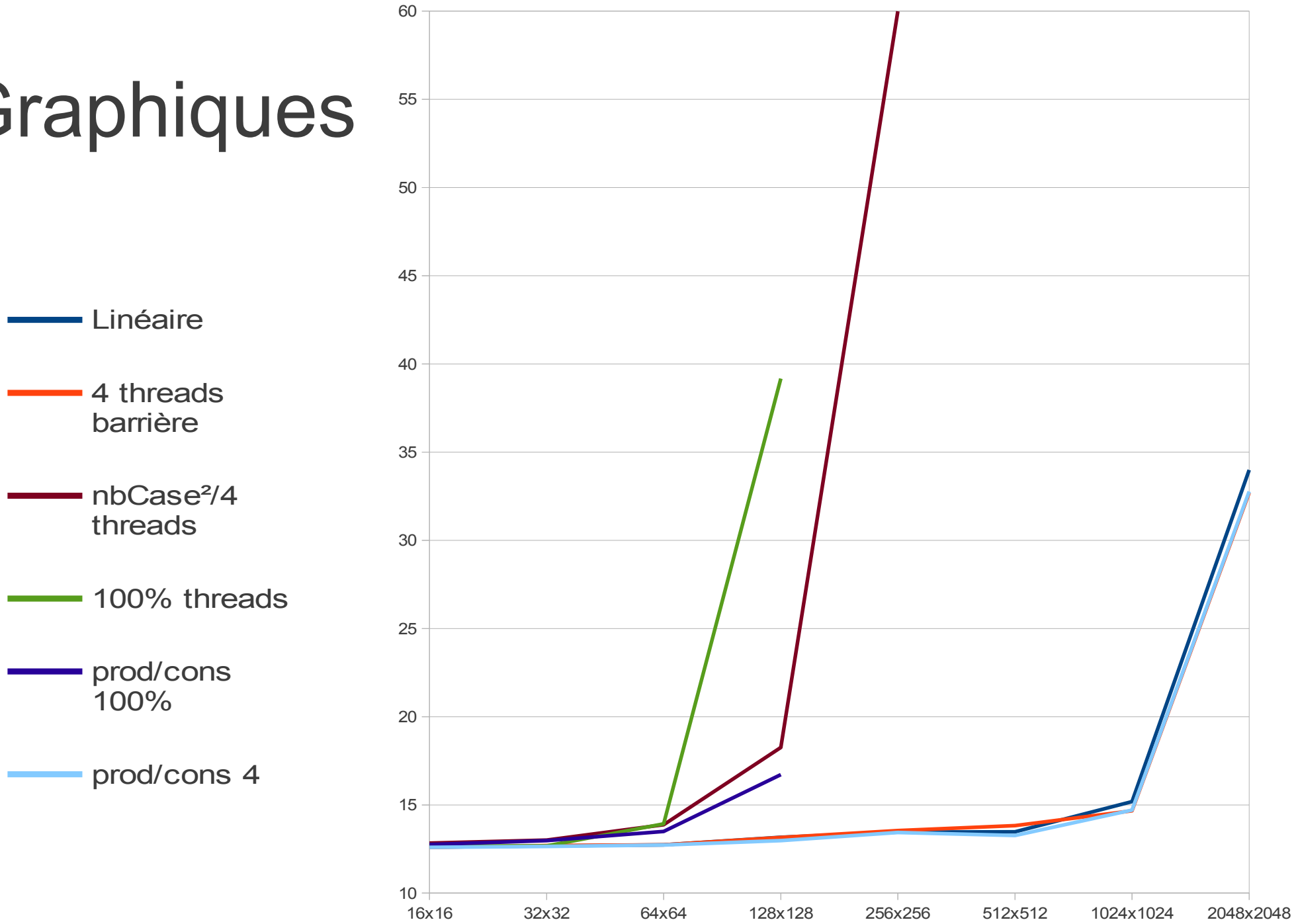
33 050 theads au maximum

```
Exception in thread "AWT-EventQueue-0" java.lang.OutOfMemoryError:
      unable to create new native thread
at java.lang.Thread.start0(Native Method)
at java.lang.Thread.start(Thread.java:640)
at java.awt.EventQueue.initDispatchThread(EventQueue.java:878)
at java.awt.EventDispatchThread.run(EventDispatchThread.java:153)
```

Producteur/consomateur

- Principe
  - Lecture séquentiel de la grille (consommateur)
  - Un thread par case.
  - Calcul de la prochaine valeur déléguée aux threads (producteurs).

Graphiques


Linéaire
4 threads barrière
nbCase²/4 threads
100% threads
prod/cons 100%
prod/cons 4
60
55
50
45
40
35
30
25
20
15
10
16x16
32x32
64x64
128x128
256x256
512x512
1024x1024
2048x2048

Barrière en FSP

```
BARRIERE = (open -> BARRIERE[0]),
BARRIERE[c:0..NBCase] = (
                          when (c < NBCase) await -> BARRIERE[c+1]
                        | when (c == NBCase) open -> BARRIERE[0]).

WORKER = (open -> doWork -> await -> WORKER).

||JEUX = (BARRIERE || [p:0..NBThread-1]:WORKER)
         /{open/[k:0..NBThread-1].open,
         [l:0..NBThread-1].await/await}.
```

No deadlock.

Conclusion

- Les barrières sont lourdes
- Un grand nombre de thread n'améliore pas la rapidité des calculs
  - Temps de création important
  - Algorithme de synchronisation lourd