Compte bancaire avec un verrou timbré (1/2)

```
public class CompteBancaire {
  private volatile long épargne;
  private final StampedLock verrou = new StampedLock();

  public CompteBancaire(long épargne) {
    this.épargne = épargne;
  }

  public void déposer(long montant) {
    long timbre = verrou.writeLock();
    try {
      épargne += montant;
    } finally {
      verrou.unlockWrite(timbre);
    }
  }
```

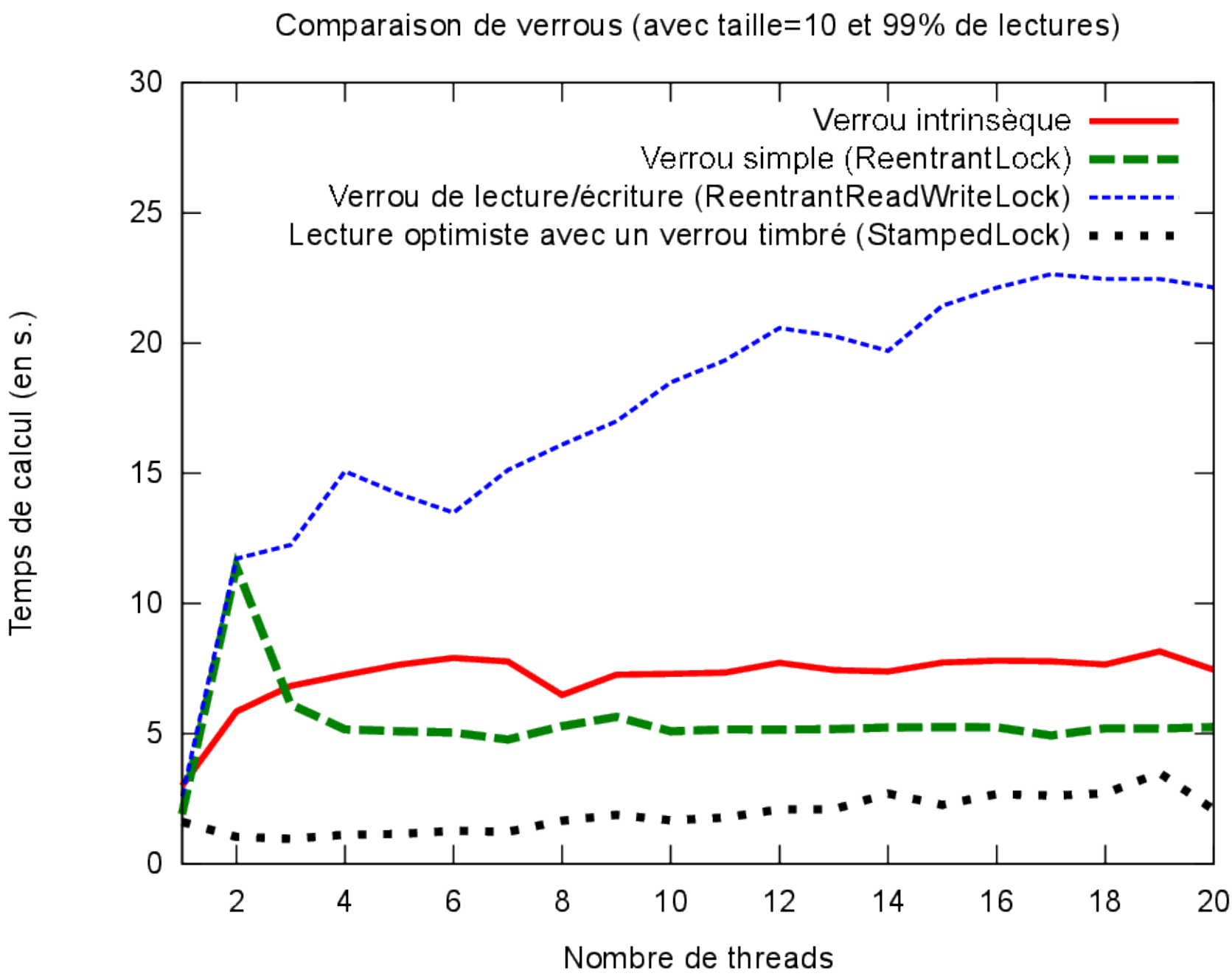
Compte bancaire avec un verrou timbré (2/2)

```
  public void retirer(long montant) {
    long timbre = verrou.writeLock();
    try {
      épargne -= montant;
    } finally {
      verrou.unlockWrite(timbre);
    }
  }
  public long solde() {
    long timbre = verrou.readLock();
    try {
      return épargne;
    } finally {
      verrou.unlockRead(timbre);
    }
  }
}
```

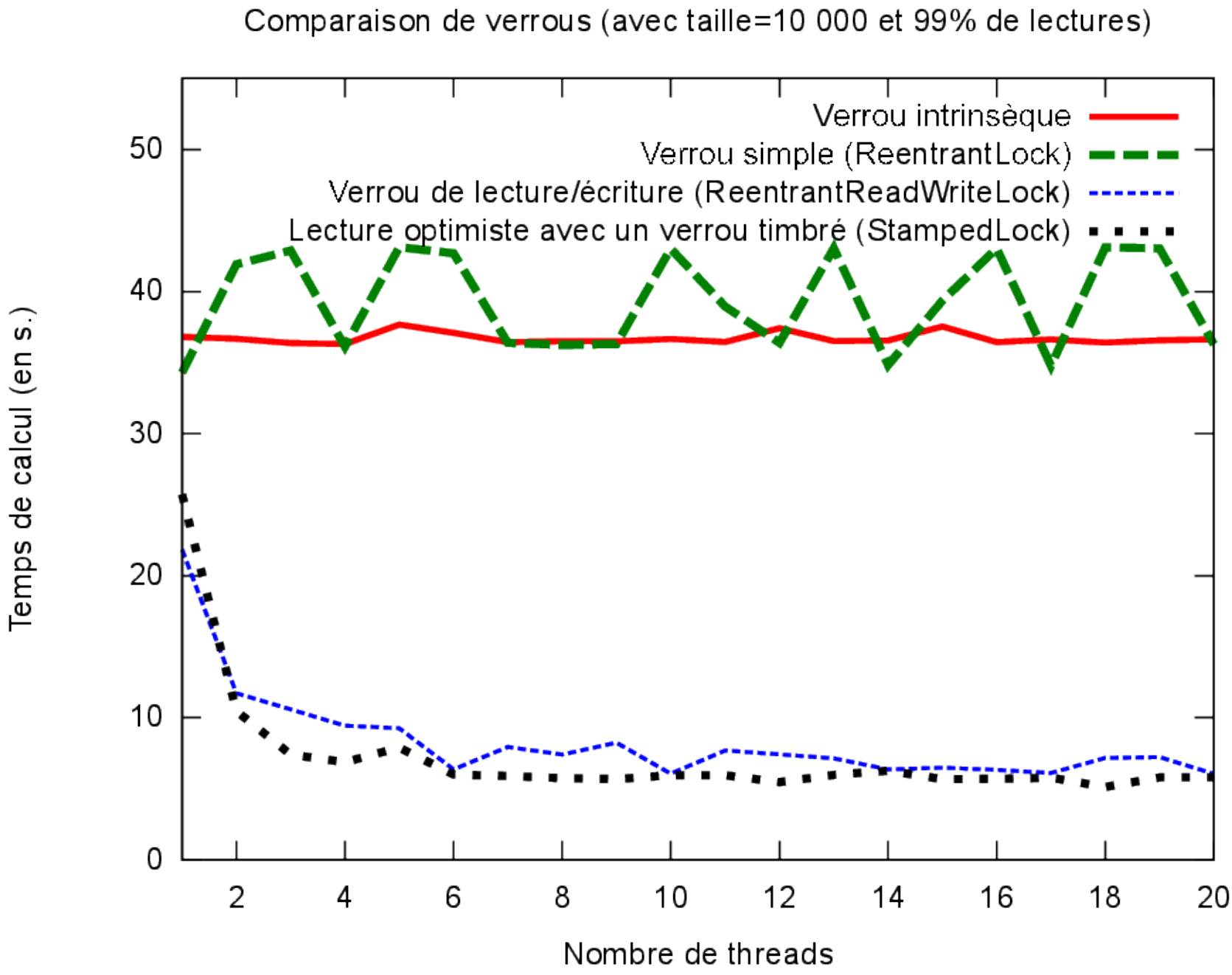
Code typique pour une lecture optimiste

```
public long solde() {
  long timbre = verrou.tryOptimisticRead();
  long copie = épargne;
      // Il serait hasardeux d'exploiter la copie dès à présent!
  if (!verrou.validate(timbre)) {
              // La copie obtenue est potentiellement corrompue !
    timbre = verrou.readLock();
    try {                  // Le verrou est à présent en mode lecture
      copie = épargne;
    } finally {
      verrou.unlockRead(timbre);
    }
  }
  return copie;                          // Cette copie est fiable!
}
```

Performances sur un tableau à 10 élements



Performances sur un tableau à 10 000 éléments



Avec un verrou timbré

```
public boolean retirer(long montant) {
  long timbre = verrou.readLock() ;
  try {
    if (montant > épargne) return false ;
    long nouveauTimbre = verrou.tryConvertToWriteLock(timbre) ;
    if (nouveauTimbre == 0){// La tentative de conversion a échoué
      verrou.unlockRead(timbre) ;
      timbre = verrou.writeLock() ;
      if (montant > épargne) return false ;
    } else {
      timbre = nouveauTimbre ;
    }// timbre est désormais un timbre de verrouillage en écriture
    épargne -= montant ;
  } finally {
    verrou.unlock(timbre) ;     // Timbre d'écriture ou de lecture!
  }
}
```

Recette pour programmer avec un objet atomique

Pour modifier correctement un objet atomique, il suffit en général de

① fabriquer une copie de la valeur courante de l'objet atomique ;

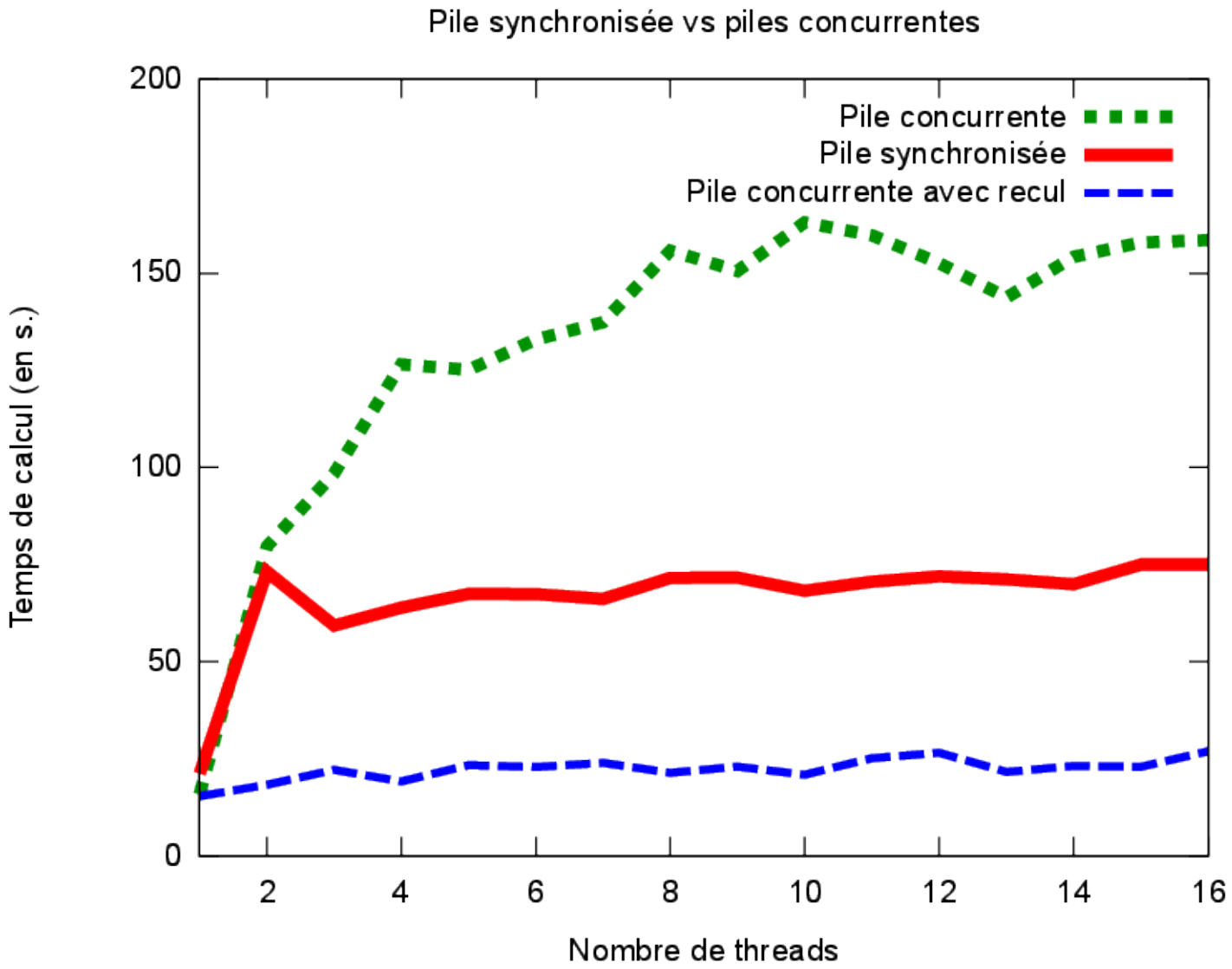
② préparer une modification de l'objet *à partir de la copie obtenue* ;

③ appliquer une mise-à-jour de l'objet atomique conformément à l'étape 2, si sa valeur courante correspond encore à celle copiée à l'étape 1 (et sinon, recommencer).

```
public boolean retirer(long montant) {
  for (;;) {
    int copie = épargne.get();
    if (montant > copie) return false;
    int prochain = copie - montant;
    if (épargne.compareAndSet(copie, prochain)) return true;
  }
}
```

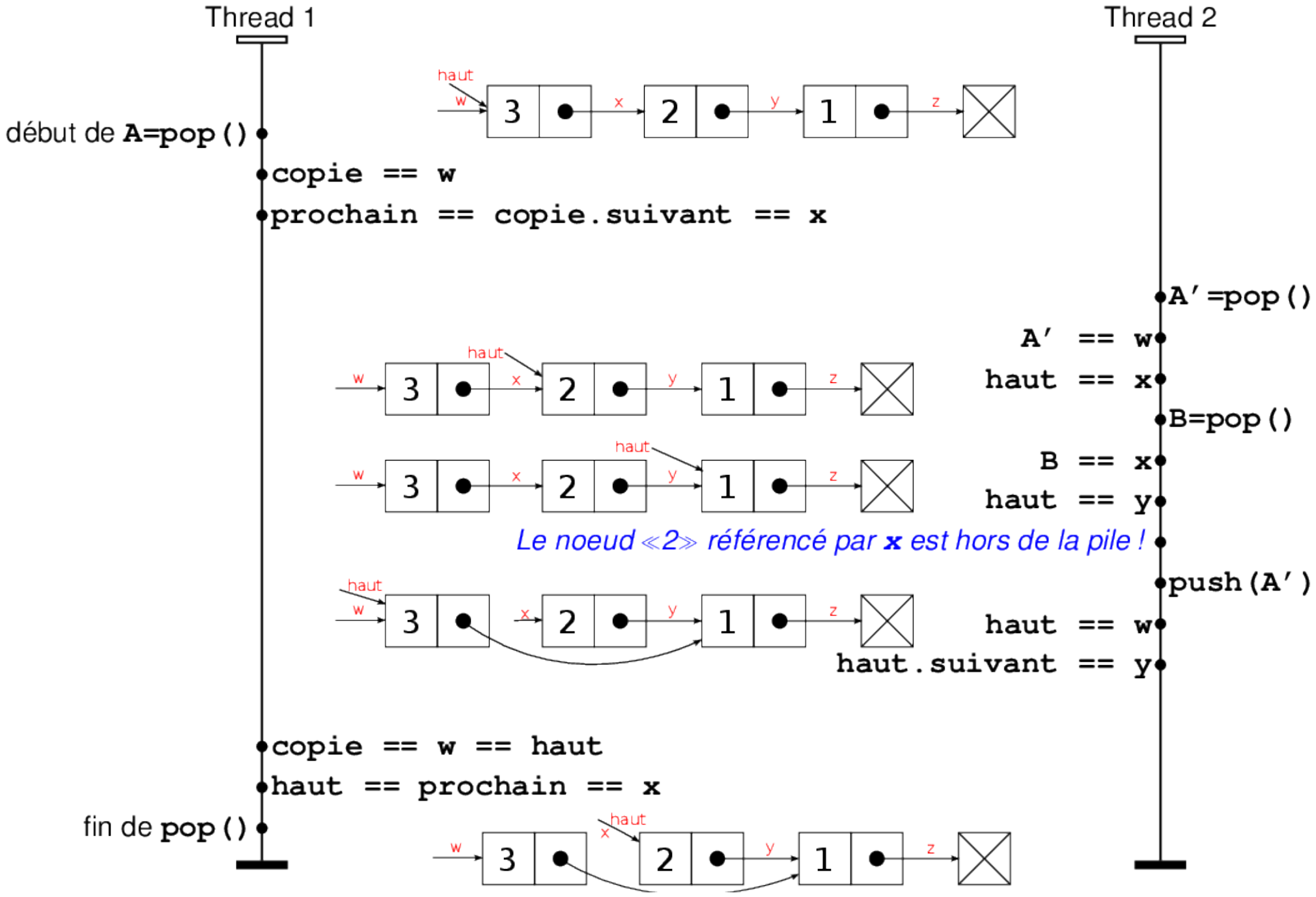
Cette méthode agit elle de manière atomique ?

Performances sur une machine à 8 processeurs



L'ajout d'un recul à la pile concurrente lors d'une contention permet d'obtenir de meilleures performances.

Problème ABA avec cette manipulation de noeuds (en Java)



Ce qu'il faut retenir

La *lecture optimiste* à l'aide d'un ≪ stamped lock ≫ produit assez souvent des résultats meilleurs que ceux observés avec les verrous de lecture-écriture.

Il est souvent souhaitable de *programmer sans utiliser de verrou*. Pour celà il existe des recettes à base d'*objets atomiques* qui reposent essentiellement sur l'instruction **`compareAndSet()`** et une boucle d'itération en cas d'échec.

Pour les données complexes, on utilise une *référence atomique* vers un objet. Les structures obtenues sont souvent aussi performantes que les structures synchronisées.

Néanmoins, il faut veiller à proscrire le *problème ABA*, s'il est susceptible d'apparaître. Les *objets immuables* sont un moyen efficace, mais parfois coûteux, pour s'en prémunir.