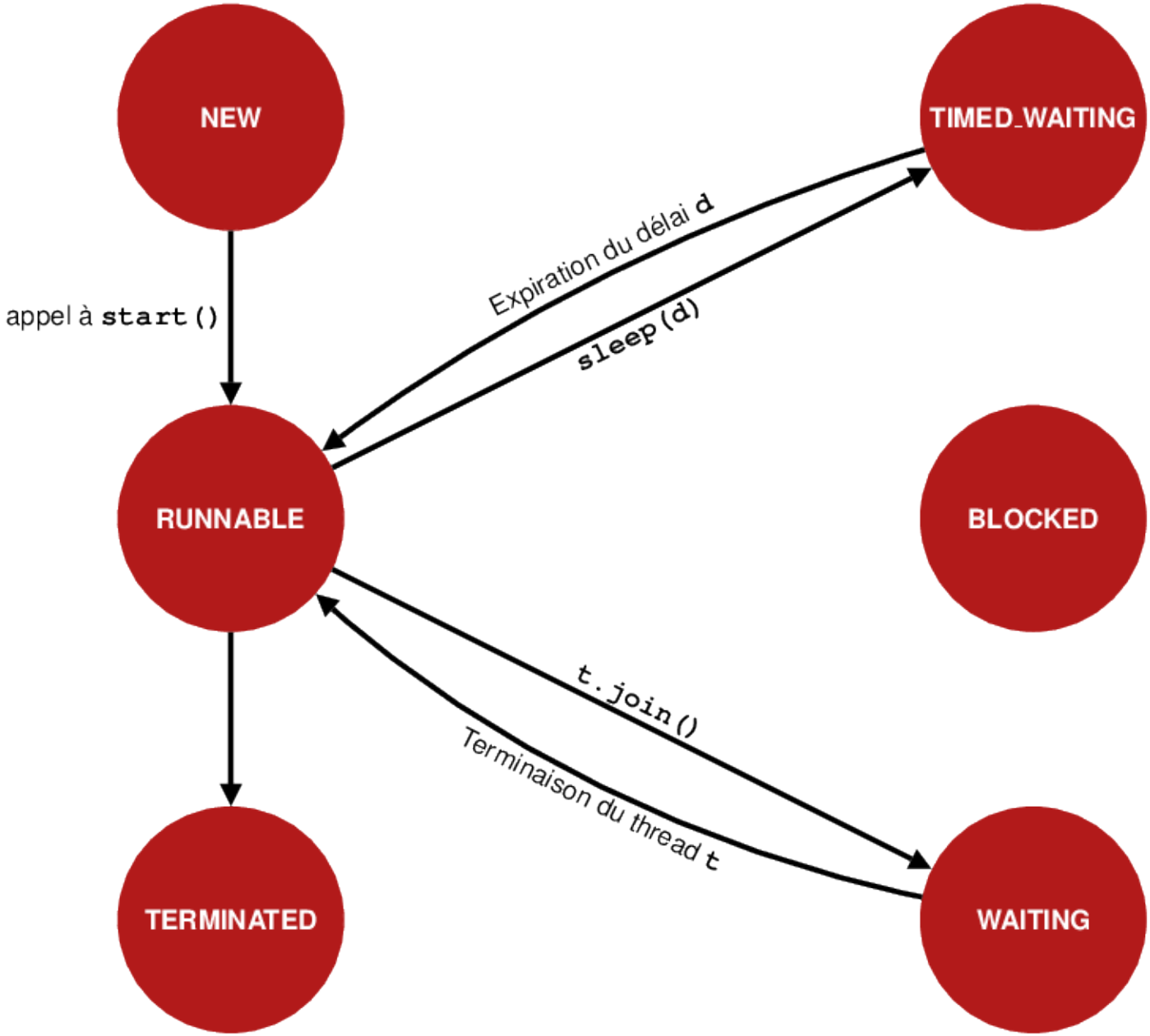
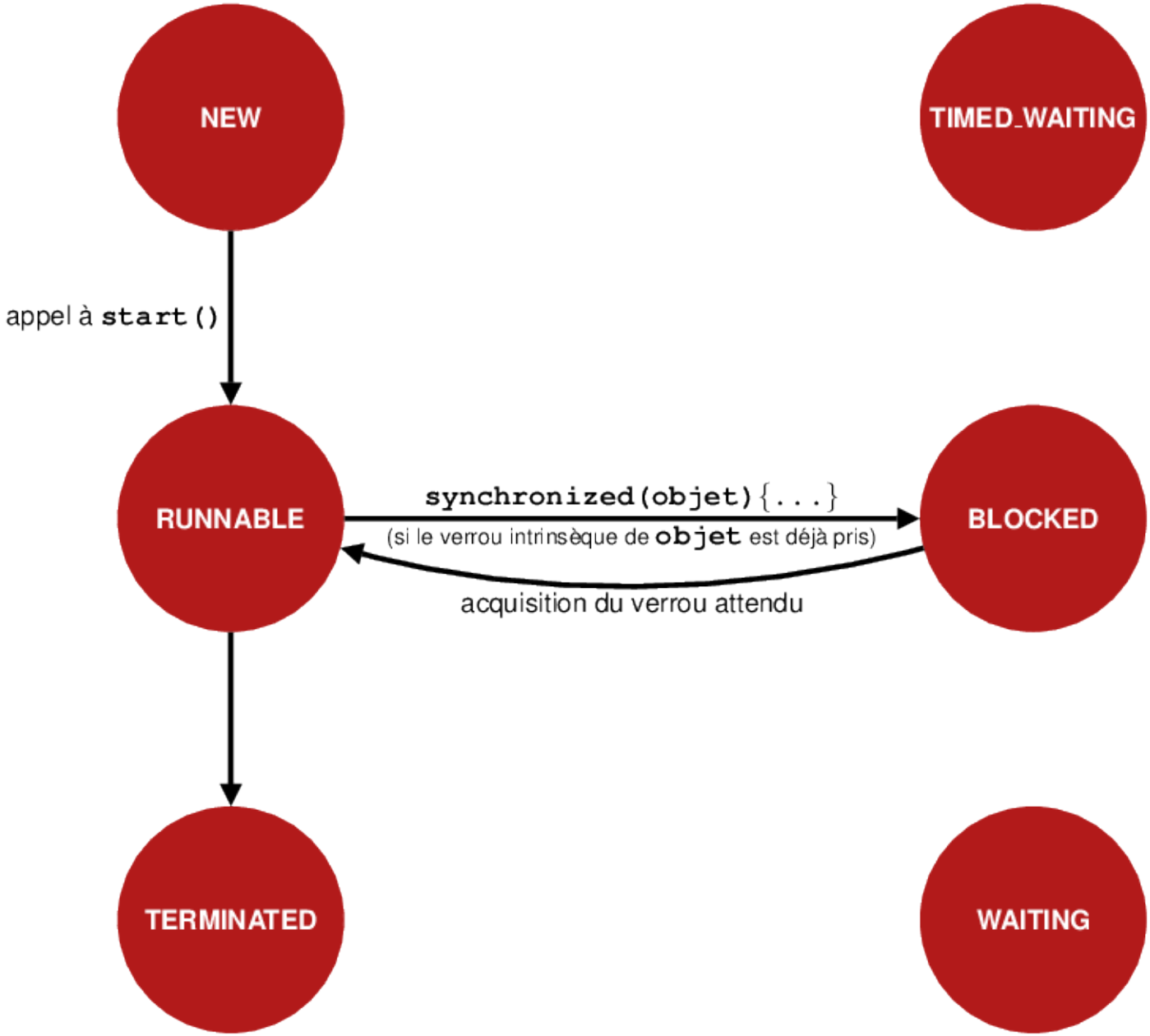
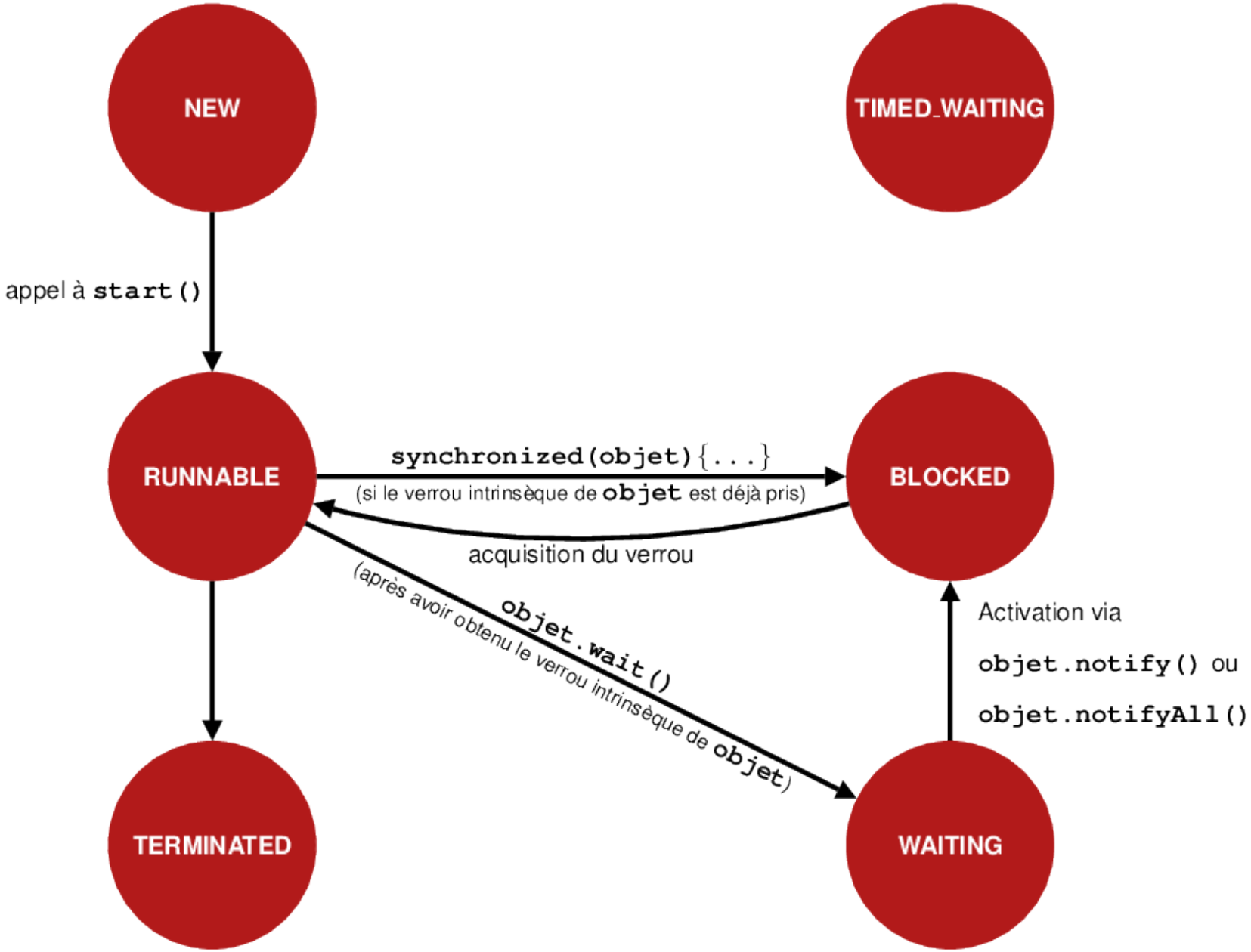
Les six états d'un thread



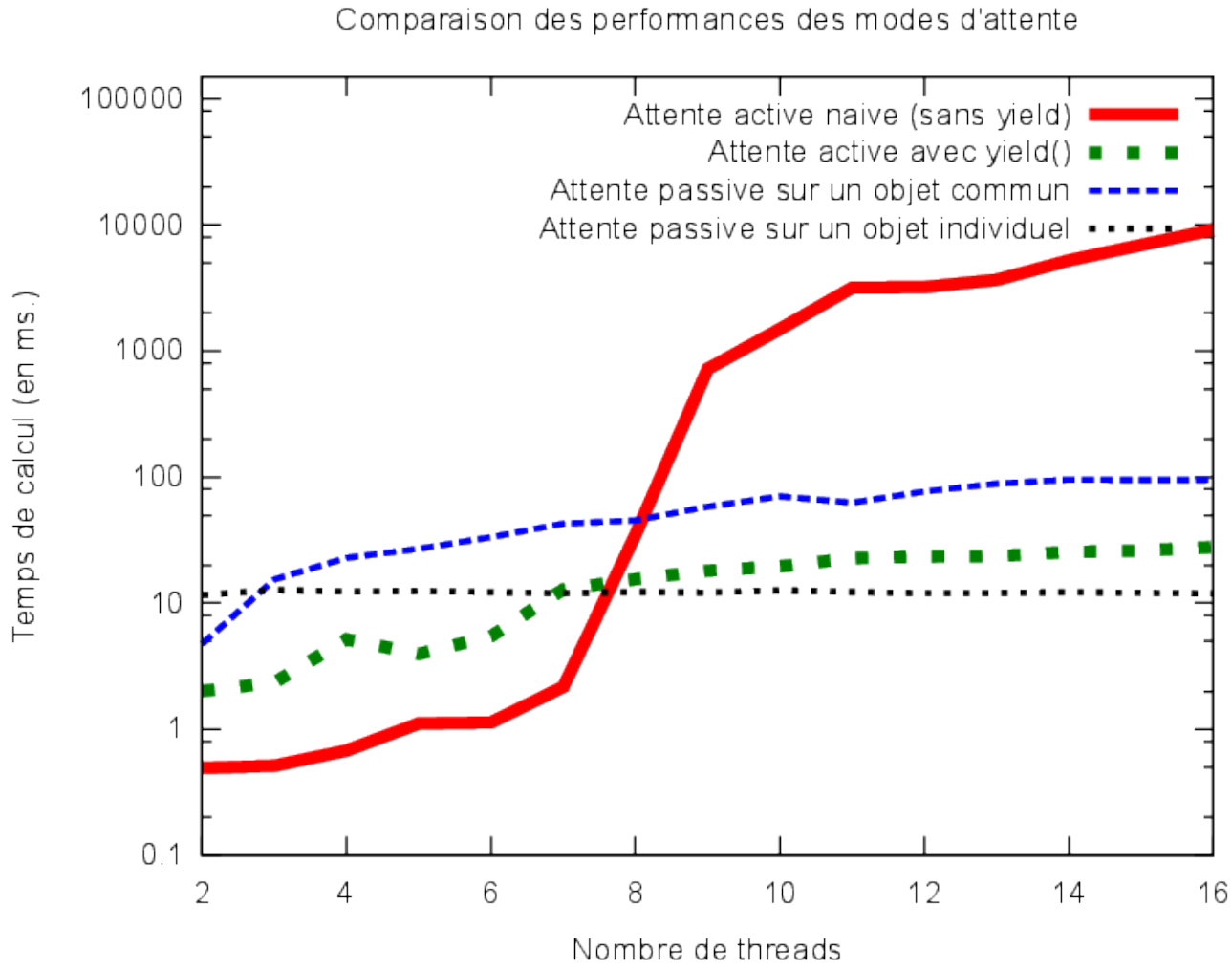
Les six états d'un thread (suite)



Les six états d'un thread (fin)



Résultats sur une machine récente, à 8 coeurs (avec plus d'incrémentations)



Tant que le nombre de threads est inférieur au nombre de coeurs, l'attente active naïve est la plus efficace à condition de ne pas insérer l'instruction **`yield()`**.

Ce qu'il faut retenir

Les priorités des threads et la méthode **`yield()`** ne servent a priori à rien, pour commencer.

Les variables susceptibles d'être accédées par plusieurs threads doivent *a priori* être déclarées **`volatile`** par précaution.

Les *verrous* associés aux objets en Java sont un outil fondamental pour écrire un programme correct en Java. La syntaxe de **`synchronized`** assure que chaque verrou pris sera relâché (à la fin du bloc).

`sleep()` permet de faire une pause *un temps déterminé*.

`wait()` permet à un thread *d'attendre sur un objet* jusqu'à ce qu'un autre thread lui lance un *signal*, via un appel à **`notify()`** ou **`notifyAll()`** sur cet objet.

La méthode **`wait()`** nécessite d'acquérir au préalable le verrou intrinsèque de l'objet sur lequel elle est appliquée, à l'aide de **`synchronized`**. *Mais ce verrou est alors relâché !*
Les méthodes **`notify()`** et **`notifyAll()`** nécessitent également au préalable **`synchronized`**, mais *ces méthodes ne relâchent pas le verrou !*