Indépendance entre parties de programmes

L'ensemble de lecture R_P (le « read set ») d'une partie P d'un programme est l'ensemble des variables lues par cette partie.

L'ensemble d'écriture W_P (le « write set ») d'une partie P d'un programme est l'ensemble des variables modifiées par cette partie.

Deux parties P_1 et P_2 d'un programme sont dites **indépendantes** si

- $W_{\mathrm{P}_1} \cap W_{\mathrm{P}_2} = \emptyset$

  $\rightsquigarrow$ P_1 et P_2 n'écrivent dans aucune variable commune

- $W_{\mathrm{P}_1} \cap \mathrm{R}_{\mathrm{P}_2} = \emptyset$

  $\rightsquigarrow$ les variables lues par P_2 ne sont pas modifiées par P_1

- $W_{\mathrm{P}_2} \cap \mathrm{R}_{\mathrm{P}_1} = \emptyset$

  $\rightsquigarrow$ les variables lues par P_1 ne sont pas modifiées par P_2

Ce que ça veut dire en pratique

Une **instruction atomique** est une suite d'actions telle que

① « *Les variables lues ou modifiées par cette instruction ne peuvent pas être modifiées par le reste du programme au cours de son exécution.* »

Donc *l'effet de l'instruction atomique sur l'état global du programme* est le même que les autres processus poursuivent leur exécution ou non.

② « *Les modifications effectuées par cette instruction ne sont visibles par le reste du programme qu'à la fin de son exécution.* »

Donc *l'effet de l'exécution sur les autres processus* est le même que si les autres processus sont suspendus pendant l'exécution de l'instruction atomique.

Ainsi, comme annoncé, tout se passe « comme si » les autres processus sont suspendus ! c'est-à-dire que l'instruction semble s'exécuter de façon exclusive.

Élimination d'un scenario d'exécution

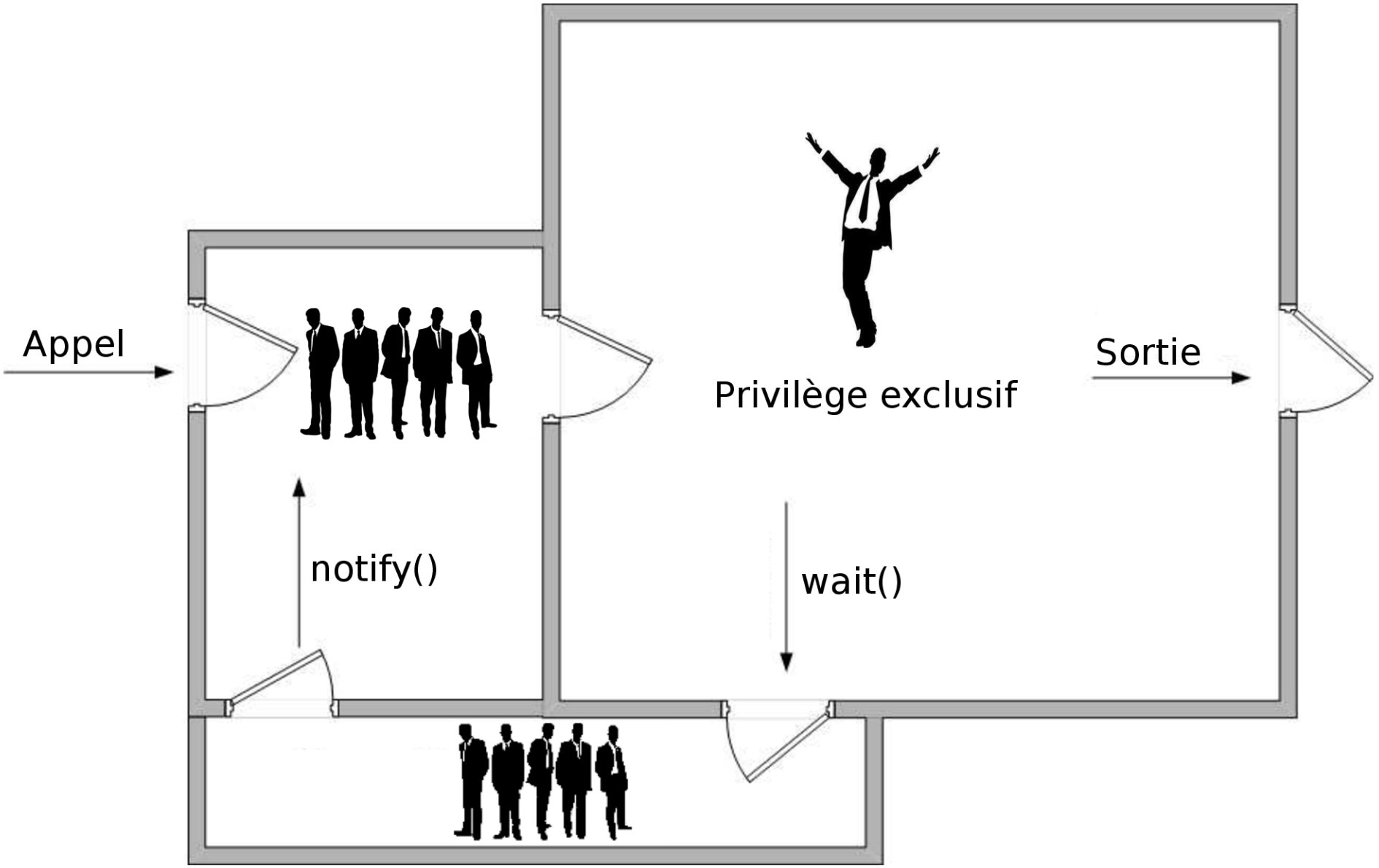
Le rôle des opérations de **synchronisation** dans un programme est d'exclure certains scenarios d'exécution indésirables. On distingue généralement deux types d'opérations de synchronisation :

① **Exclusion mutuelle** : il s'agit former des séquences d'actions, appelées *sections critiques*, de sorte qu'à chaque instant **au plus un** processus exécute le code d'une section critique.
Les verrous sont les outils les plus simples pour assurer cela.

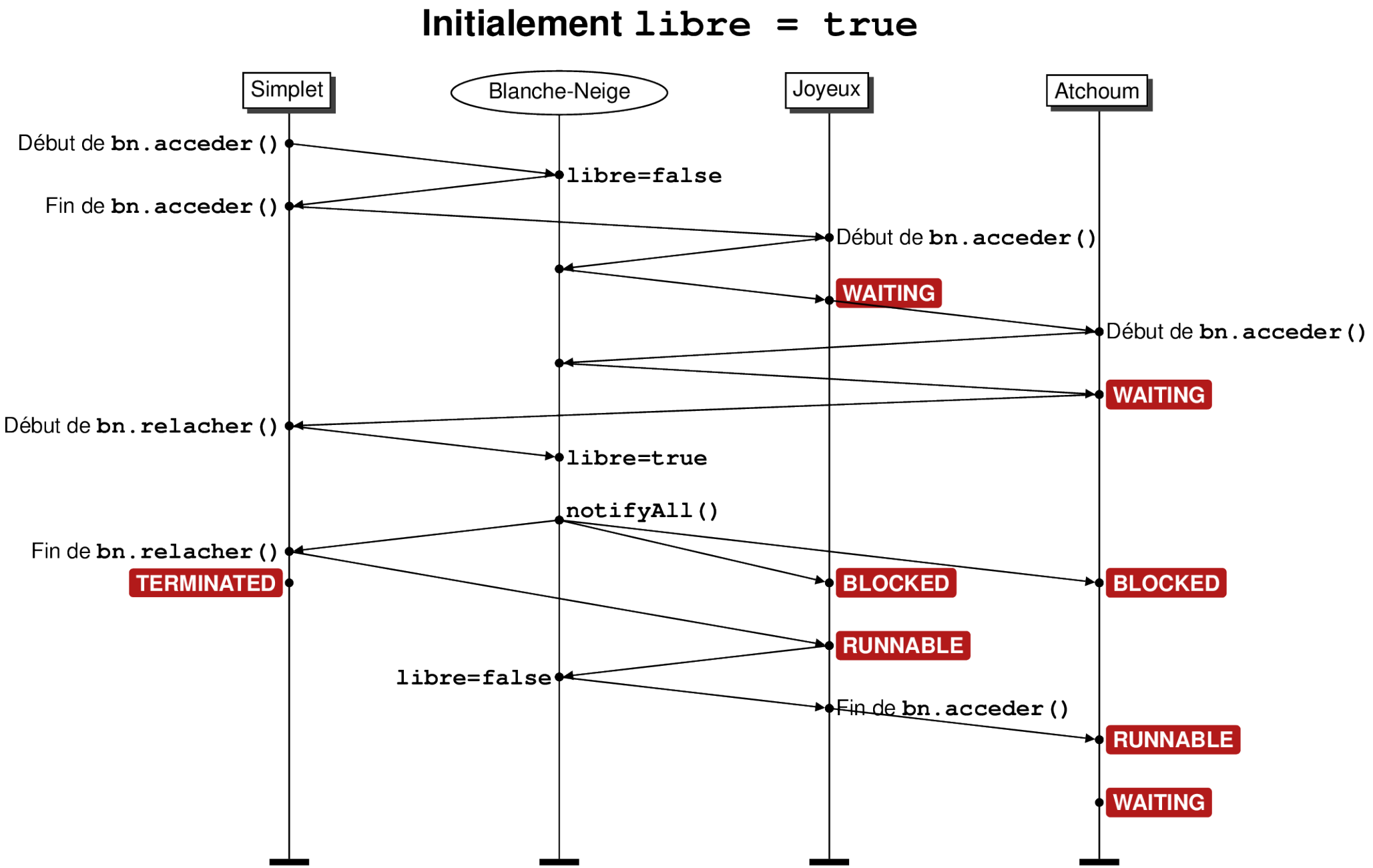
② **Synchronisation conditionnelle** : il s'agit de *retarder* une action d'un processus jusqu'à ce que l'état du programme satisfasse une certaine *condition*.
Les variables de condition sont là pour ça.

Ces deux problématiques sont liées !

Fonctionnement d'un moniteur



Fonctionnement du moniteur Blanche-Neige



Les signaux intempestifs

```
class WakeUp  extends Thread{

  public static void main(String [] args) {
    new WakeUp().start();        // Un seul thread est lancé
  }

  public synchronized void run() {
      try { wait(); }             // Le thread patiente
      catch (InterruptedException ignoree) {};
  }
}
```

Ce programme peut-il terminer ?