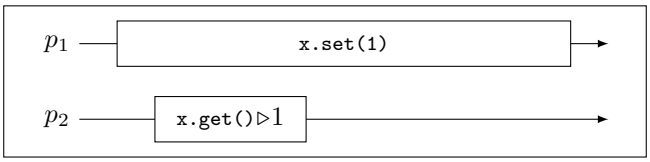
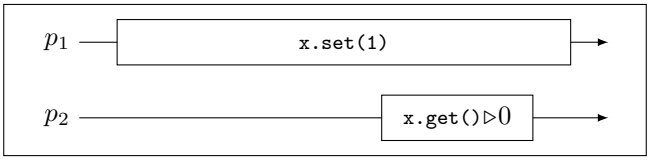
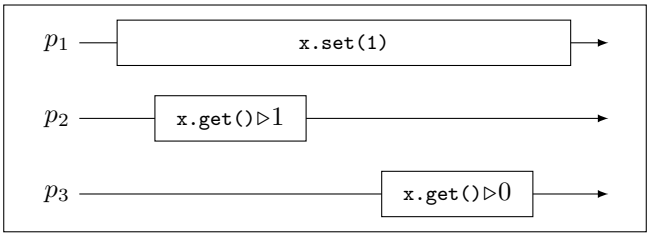
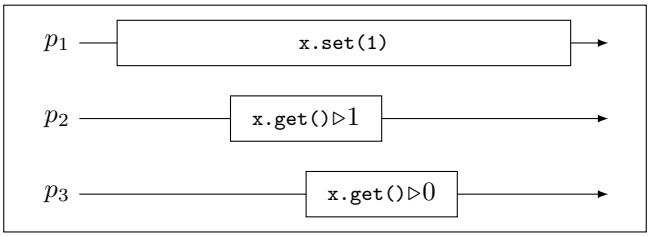
Programmation multi-cœurs — TD 3
Modèles de mémoire

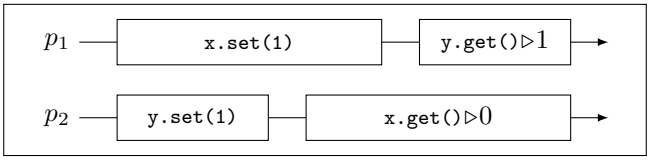
Exercice 1. Quelles histoires sont linéarisables ?

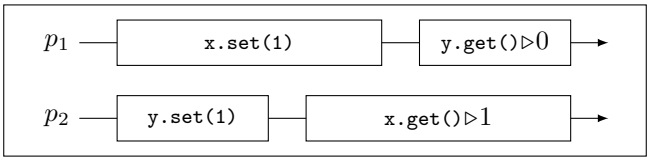












Exercice 2. Quelles sont les sorties possibles si les trois codes suivants s'exécutent en parallèle et les assertions sont vérifiées :

1. Si `y` et `z` sont déclarés **volatile** ?
2. Si `x` est déclaré **volatile** ?
3. Si seulement `y` est déclaré **volatile** ?

```
/******* Thread 1 *******/
x = 1;
y = 1;
```

```
/******* Thread 2 *******/
assert(y == 1);
z = 1;
```

```
/******* Thread 3 *******/
assert(z == 1);
system.out.println(x);
```

Exercice 3. Quelles sont les sorties possibles si les deux codes suivants s'exécutent en parallèle :

1. Si x et y sont des variables locales ?
2. Si x et y sont des variables partagées ?
3. Si x et y sont déclarées **volatile** ?

```
/************** Thread 1 **************/
int a = 0;
x = 1;
while (y == 0){
  a = a + 1;
}
system.out.println(a);
```

```
/************** Thread 2 **************/
int b = 0;
y = 1;
while (x == 0){
  b = b + 1;
}
system.out.println(b);
```

Exercice 4. *Le compteur*. On propose l'implémentation suivante d'un compteur.

```
class Compteur {

  private int value = 0;

  public int read() {
    return value;
  }

  public synchronized void increment() {
    ++value;
  }
}
```

1. Expliquez pourquoi ce compteur n'est pas linéarisable.
2. Corrigez le programme.

Exercice 5. *Le singleton*. On rappelle le code séquentiel du patron de conception "Singleton".

```
class Single {

  private static Single instance = null;

  private Singleton() {}

  public static Single getInstance() {
    if (instance == null)
      instance = new Single();
    return instance;
  }
}
```

1. Donnez une exécution avec deux threads où deux instances sont créées.
2. Corrigez le programme. Il faudra réduire l'usage des verrous au maximum.