

Exclusion mutuelle

```

1  class LockOne implements Lock {
2      private boolean[] flag = new boolean[2];
3      // thread-local index, 0 or 1
4      public void lock() {
5          int i = ThreadID.get();
6          int j = 1 - i;
7          flag[i] = true;
8          while (flag[j]) {}          // wait
9      }
10     public void unlock() {
11         int i = ThreadID.get();
12         flag[i] = false;
13     }
14 }

```

```

1  class LockTwo implements Lock {
2      private int victim;
3      public void lock() {
4          int i = ThreadID.get();
5          victim = i;                // let the other
6          while (victim == i) {}    // wait
7      }
8      public void unlock() {}
9  }

```

```

1  class Peterson implements Lock {
2      // thread-local index, 0 or 1
3      private boolean[] flag = new boolean[2];
4      private int victim;
5      public void lock() {
6          int i = ThreadID.get();
7          int j = 1 - i;
8          flag[i] = true;            // I'm interested
9          victim = i;                // you go first
10         while (flag[j] && victim == i) {}; // wait
11     }
12     public void unlock() {
13         int i = ThreadID.get();
14         flag[i] = false;           // I'm not interested
15     }
16 }

```

```

1  class Filter implements Lock {
2      int[] level;
3      int[] victim;
4      public Filter(int n) {
5          level = new int[n];
6          victim = new int[n]; // use 1..n-1
7          for (int i = 0; i < n; i++) {
8              level[i] = 0;
9          }
10     }
11     public void lock() {
12         int me = ThreadID.get();
13         for (int i = 1; i < n; i++) { // attempt level i
14             level[me] = i;
15             victim[i] = me;
16             // spin while conflicts exist
17             while (( $\exists k \neq me$ ) (level[k] >= i && victim[i] == me)) {};
18         }
19     }
20     public void unlock() {
21         int me = ThreadID.get();
22         level[me] = 0;
23     }
24 }

```

```

1  class Bakery implements Lock {
2      boolean[] flag;
3      Label[] label;
4      public Bakery (int n) {
5          flag = new boolean[n];
6          label = new Label[n];
7          for (int i = 0; i < n; i++) {
8              flag[i] = false; label[i] = 0;
9          }
10     }
11     public void lock() {
12         int i = ThreadID.get();
13         flag[i] = true;
14         label[i] = max(label[0], ..., label[n-1]) + 1;
15         while (( $\exists k \neq i$ )(flag[k] && (label[k],k) << (label[i],i))) {};
16     }
17     public void unlock() {
18         flag[ThreadID.get()] = false;
19     }
20 }

```