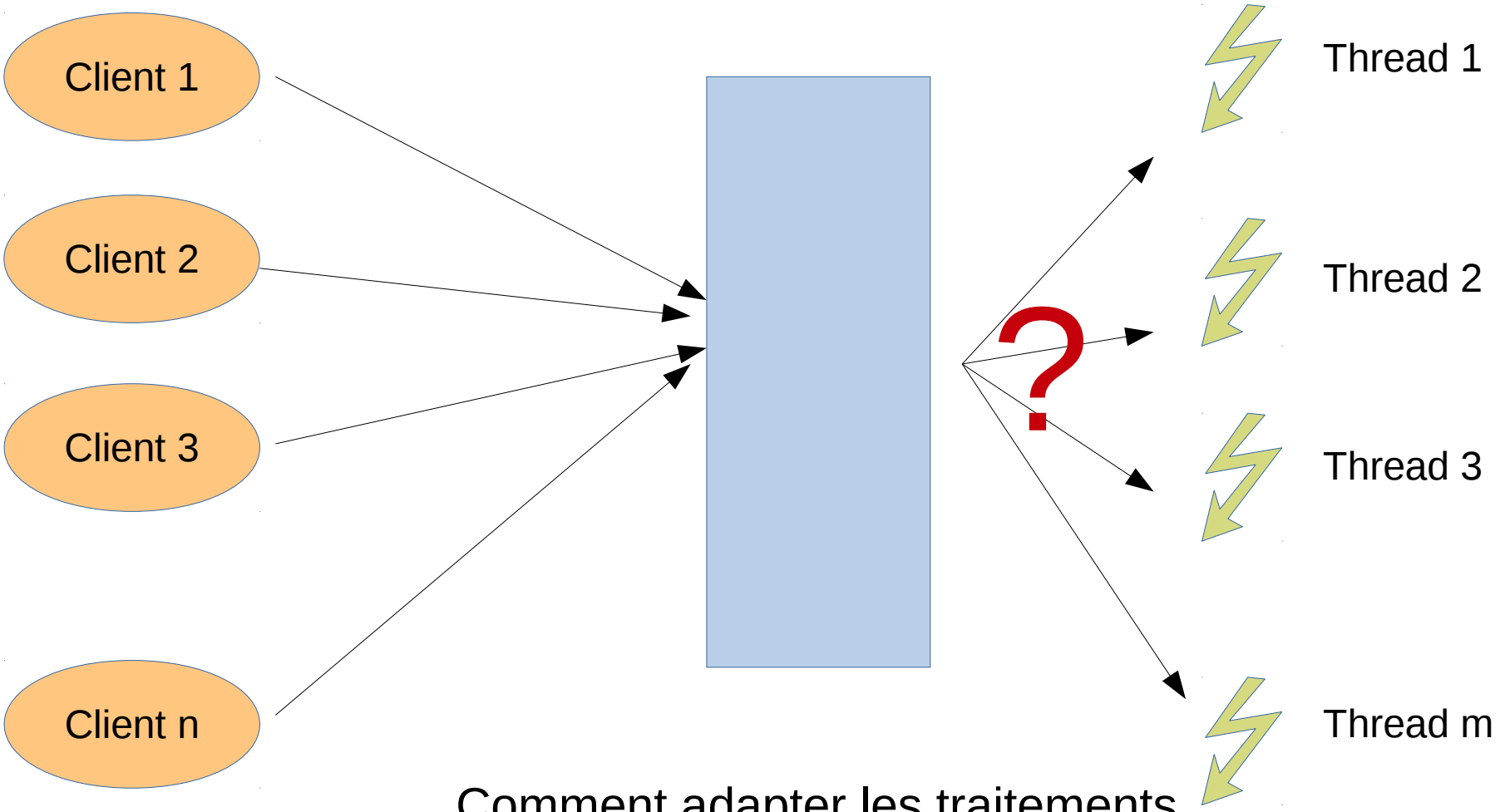
Concurrence producteur/consommateur

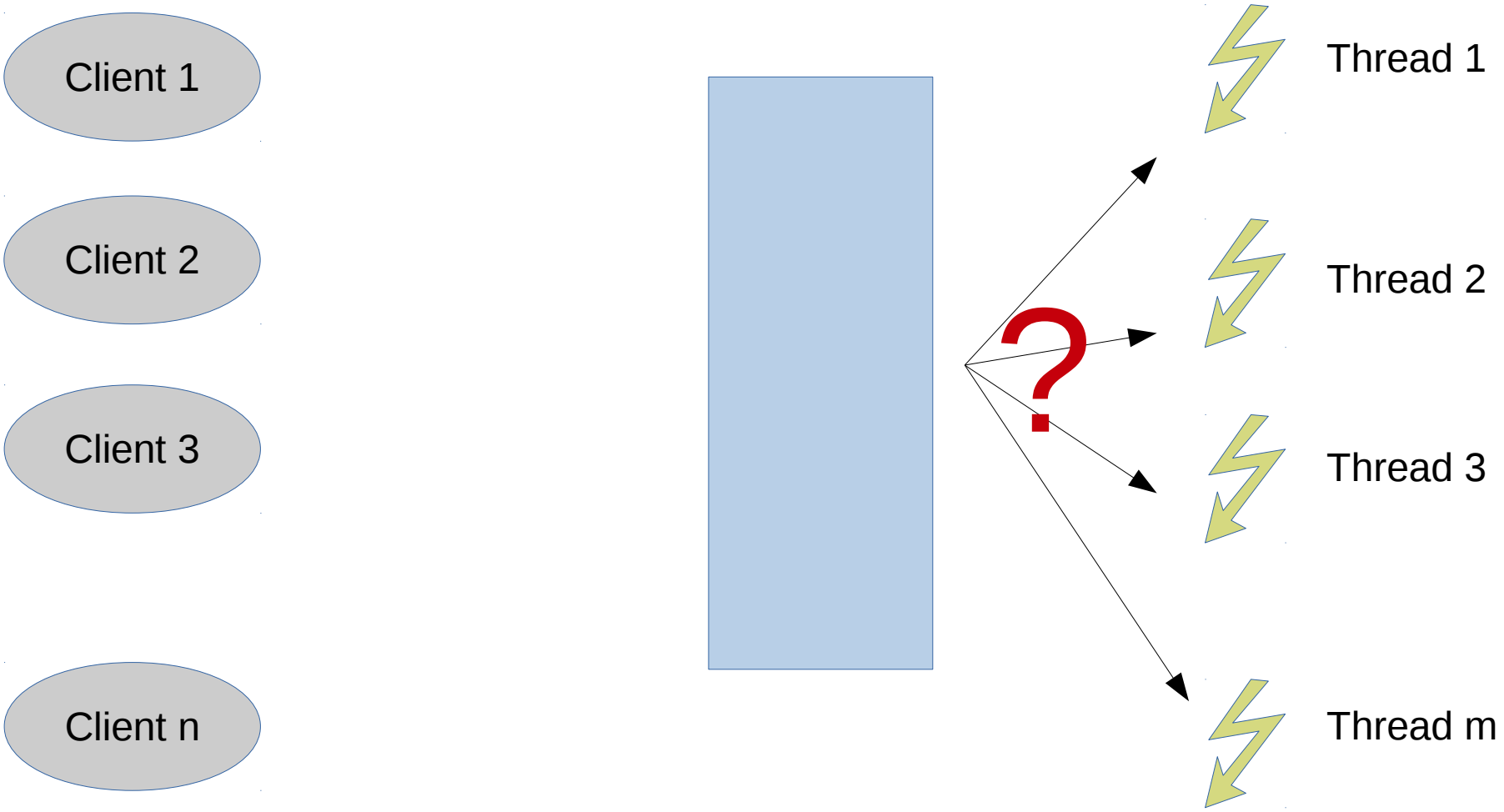
Rémi Forax

Un serveur Web



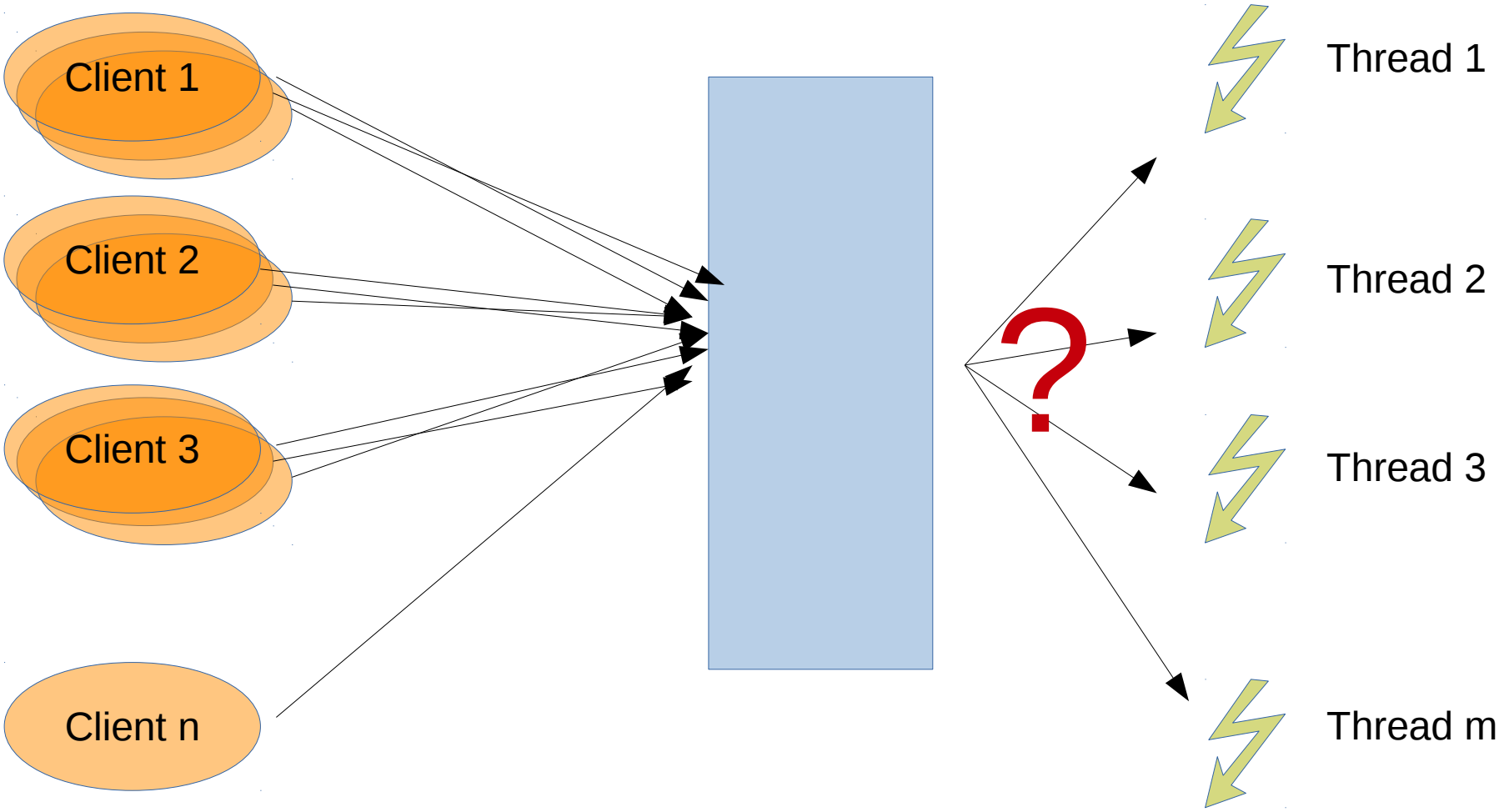
Comment adapter les traitements
si il y a plus de clients que de threads,
ou moins de client que de thread ?

Si aucun client ?



Les threads doivent être en attente !

Si trop de clients ?



Les clients doivent être en attente

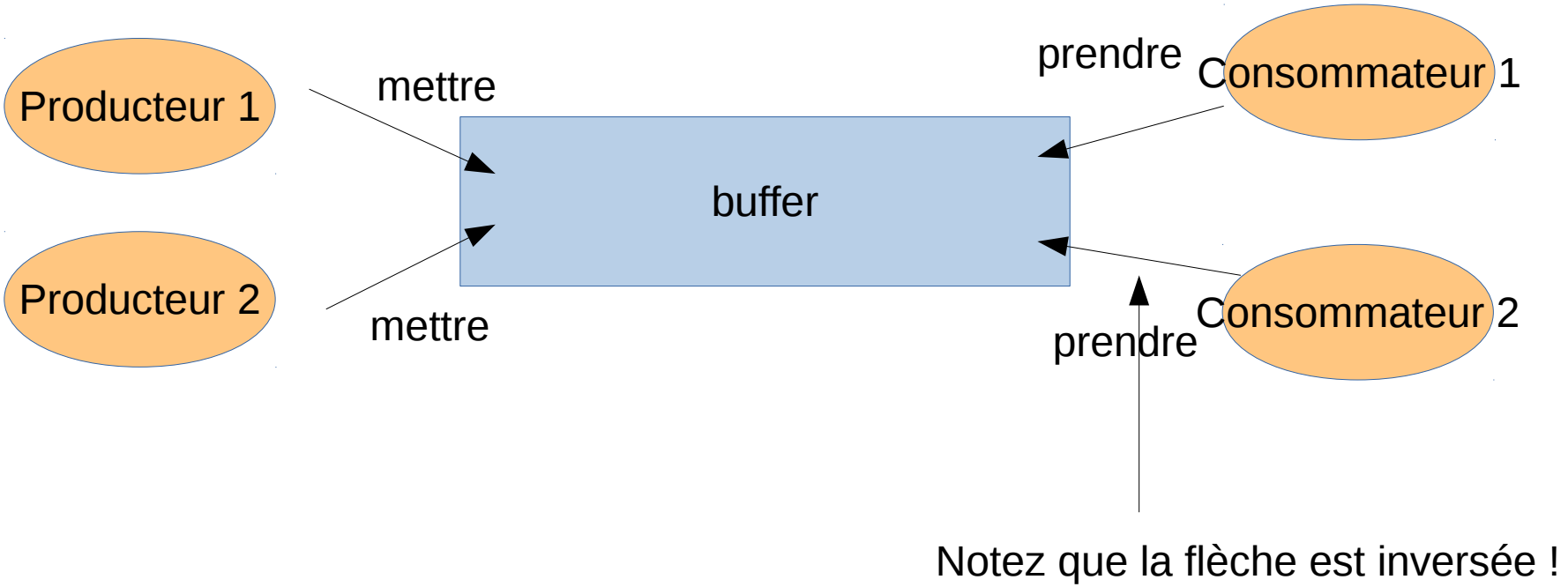
Producteur / Consommateur

Design Pattern permettant

- D'arrêter un producteur si il n'y a pas de consommateur
- D'arrêter un consommateur si il n'y a pas de producteur

Astuce: on utilise un buffer intermédiaire

Producteur / Consommateur



On utilise un buffer intermédiaire,
une file (queue en anglais) dans laquelle
on va mettre des messages et prendre des message

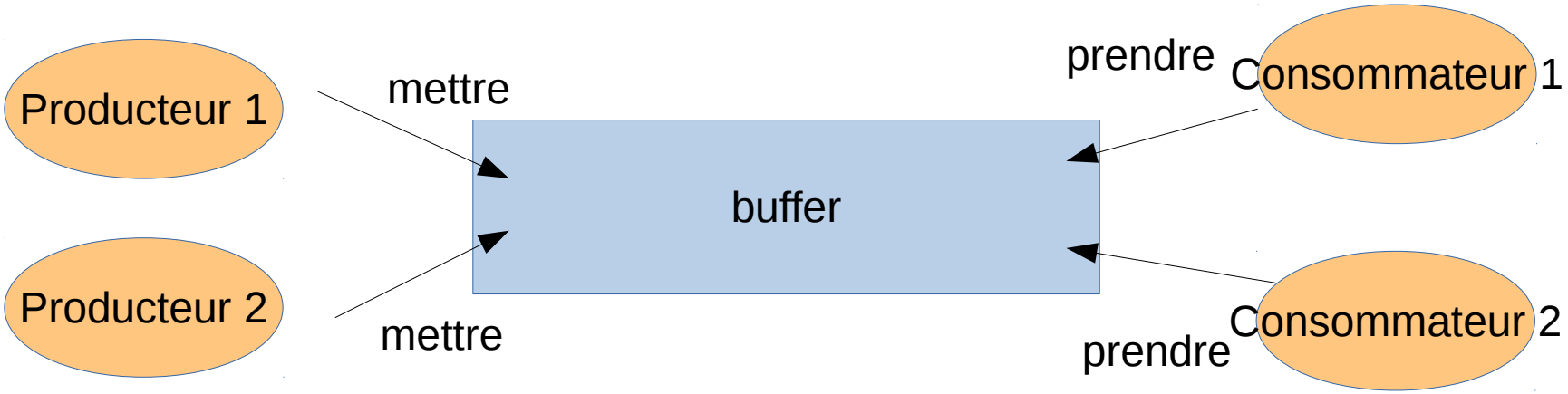
Producteur / Consommateur

Si la file est vide,

- On met le consommateur en attente

Si la file est pleine,

- On met le producteur en attente



Implantation avec wait/notify

```
public class Buffer {
  private final ArrayDeque<Message> buffer =
    new ArrayDeque<>();
  private final int capacity;

  public Buffer(int capacity) {
    this.capacity = capacity;
  }

  public void put(Message message) {
    synchronized(buffer) {
      while(buffer.size() == capacity) {
        buffer.wait();
      }
      buffer.addLast(message);
      ...
    }
  }
  ...
}
```

On peut utiliser buffer comme moniteur !

```
public Message take() {
  synchronized(buffer) {
    while(buffer.size() == 0) {
      buffer.wait();
    }
    ...
    return buffer.removeFirst();
  }
}
```

Implantation avec wait/notify

```
public class Buffer {
  private final ArrayDeque<Message> buffer =
    new ArrayDeque<>();
  private final int capacity;

  public Buffer(int capacity) {
    this.capacity = capacity;
  }

  public void put(Message message) throws IE{
    synchronized(buffer) {
      while(buffer.size() == capacity) {
        buffer.wait();
      }
      buffer.addLast(message);
      buffer.notifyAll();
    }
  }
  ...
}
```

```
public Message take() throws IE {
  synchronized(buffer) {
    while(buffer.size() == 0) {
      buffer.wait();
    }
    buffer.notifyAll();
    return buffer.removeFirst();
  }
}
```

Et avec des locks ?

```
public class Buffer {
  private final ArrayDeque<Message> buffer =
    new ArrayDeque<>();
  private final ReentrantLock lock = new ReentrantLock();
  private final Condition isEmpty = lock.newCondition();
  private final Condition isFull = lock.newCondition();
  private final int capacity;
  ...
  public void put(Message message) throws IE {
    lock.lock();
    try  {
      while(buffer.size() == capacity) {
        isFull.await();
      }
      buffer.addLast(message);
      isEmpty.signalAll();
    } finally {
      lock.unlock();
    }
  }
  ...
}
```

```
public Message take() throws IE {
  lock.lock();
  try  {
    while(buffer.size() == 0) {
      isEmpty.await();
    }
    isFull.signalAll();
    return buffer.removeFirst();
  } finally {
    lock.unlock();
  }
}
```

j.u.c.BlockingQueue

Le buffer des producteurs/consommateurs est déjà implanté en Java

Il existe plusieurs implantations implantant l'interface BlockingQueue

- LinkedBlockingQueue
  - Utilise une liste chainée (attention à fixer la taille)
- ArrayBlockingQueue
  - Utilise un tableau circulaire (comme ArrayDeque)
- SynchronousQueue
  - N'accepte qu'un seul élement -> debug