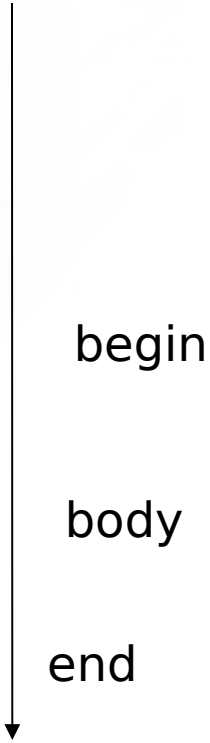
Laboratório de Pesquisa em Redes e Multimídia

THREADS

Laboratório - Java

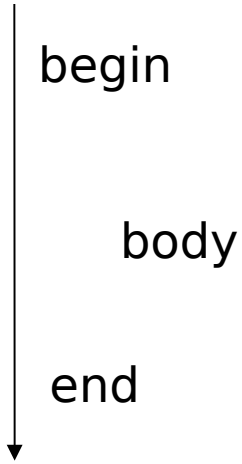
Universidade Federal do Espírito Santo
Departamento de Informática

```
class ABC
{
....
    public void main(..)
    {
    ...
    ..
    }
}
```

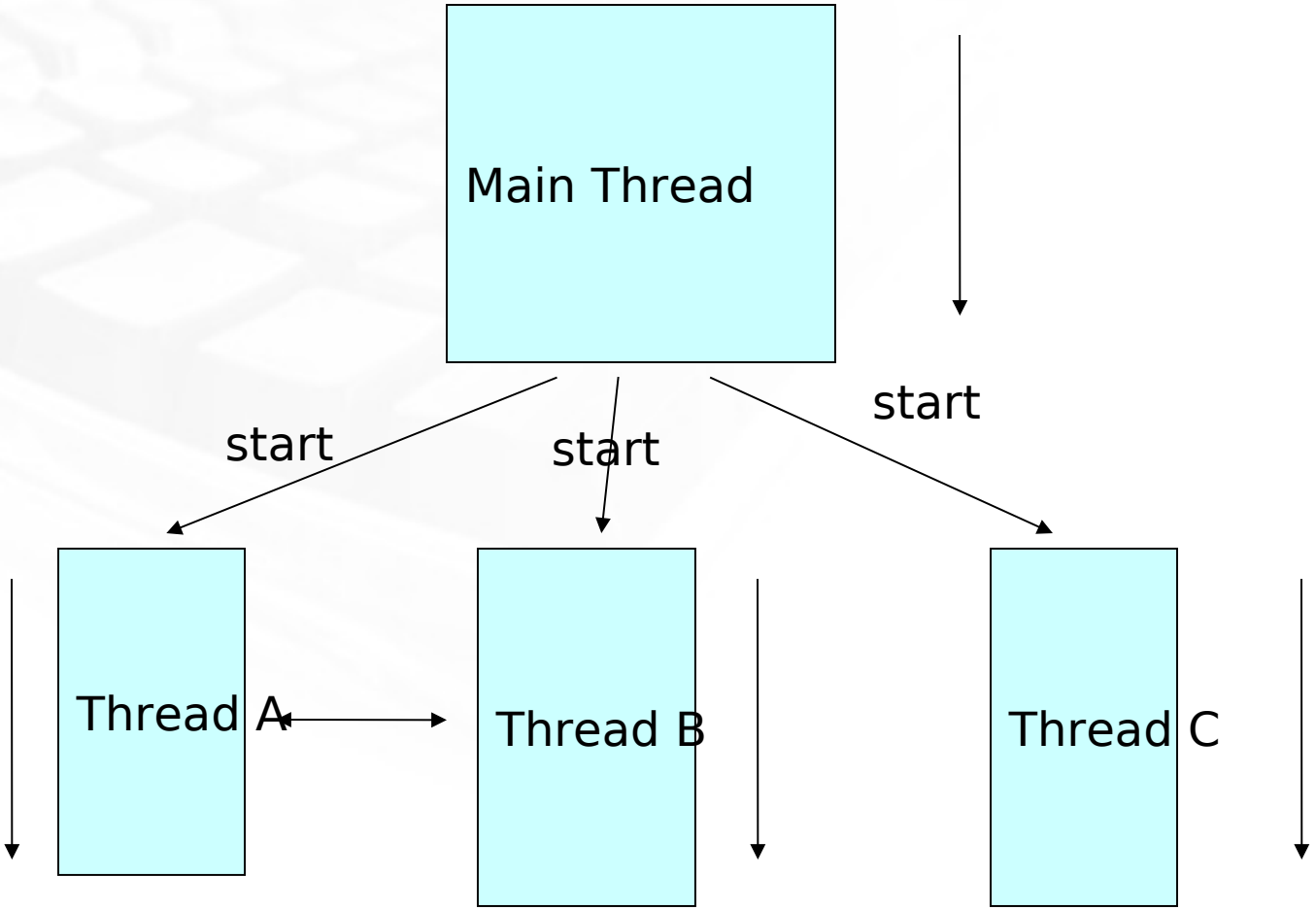


A single threaded program

```
class ABC
{
....
    public void main(..)
    {
    ...
    ..
    }
}
```



A Multithreaded Program



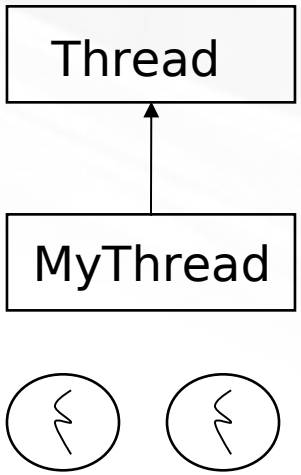
Java Threads

- No Linux, Java™ threads são mapeadas em threads de sistema (kernel)
- Java oferece suporte built-in a threads
  - Inter-Thread Communication/ Synchronization
  - Thread Scheduling
- Java Garbage Collector é uma low-priority thread

Mecanismos de Criação

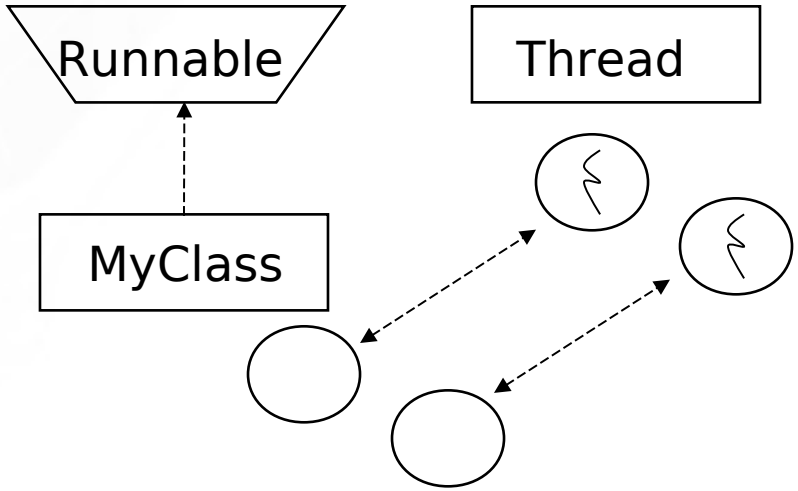
a) Criar uma classe que *extends* a *Thread class*

b) Criar uma classe que implementa a interface *Runnable*



(objects are threads)

[a]



(objects with run() body)

[b]

a)Estendendo a classe Thread

- Cria subclasse estendo a classe Thread e sobrescrevendo o método run()

```
class PrintNameThread extends Thread {
    PrintNameThread(String name) {
        super(name);
    }

    // Override the run() method of the Thread class.
    // This method gets executed when start() method is invoked.
    public void run() {
        System.out.println("run() method of the " +
                           this.getName() + " thread is called" );
        for (int i = 0; i < 10; i++) {
            System.out.print(this.getName());
        }
    }
}
```

a)Estendendo a classe Thread

```
public class ExtendThreadClassTest0 {

    public static void main(String args[]) {
      // Create object of a class that is subclass of Thread class
      System.out.println("Creating PrintNameThread object instance..");
      PrintNameThread pnt1 = new PrintNameThread("A");
      // Start the thread execution by invoking start()
      System.out.println("Calling start() method of " + pnt1.getName());
      pnt1.start();

      System.out.println("Creating PrintNameThread object instance..");
      PrintNameThread pnt2 = new PrintNameThread("B");
      System.out.println("Calling start() method of " + pnt2.getName());
      pnt2.start();

      System.out.println("Creating PrintNameThread object instance..");
      PrintNameThread pnt3 = new PrintNameThread("C");
      System.out.println("Calling start() method of " + pnt3.getName());
      pnt3.start();
    }
}
```

b)Implementando *Runnable*

- Cria-se uma classe que implementa a interface *Runnable*, implementando igualmente o método *run()*

```
class PrintNameRunnable implements Runnable {
    String name;
    PrintNameRunnable(String name) {
        this.name = name;
    }

    // Implementation of the run() defined in the
    // Runnable interface.
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.print(name);
        }
    }
}
```

b)Implementando *Runnable*

```
public class RunnableThreadTest1 {
  public static void main(String args[]) {
      PrintNameRunnable pnt1 = new PrintNameRunnable("A");
      Thread t1 = new Thread(pnt1);
      t1.start();

      PrintNameRunnable pnt2 = new PrintNameRunnable("B");
      Thread t2 = new Thread(pnt2);
      t2.start();

      PrintNameRunnable pnt3 = new PrintNameRunnable("C");
      Thread t3 = new Thread(pnt3);
      t3.start();

  }
}
```

Shared Resources

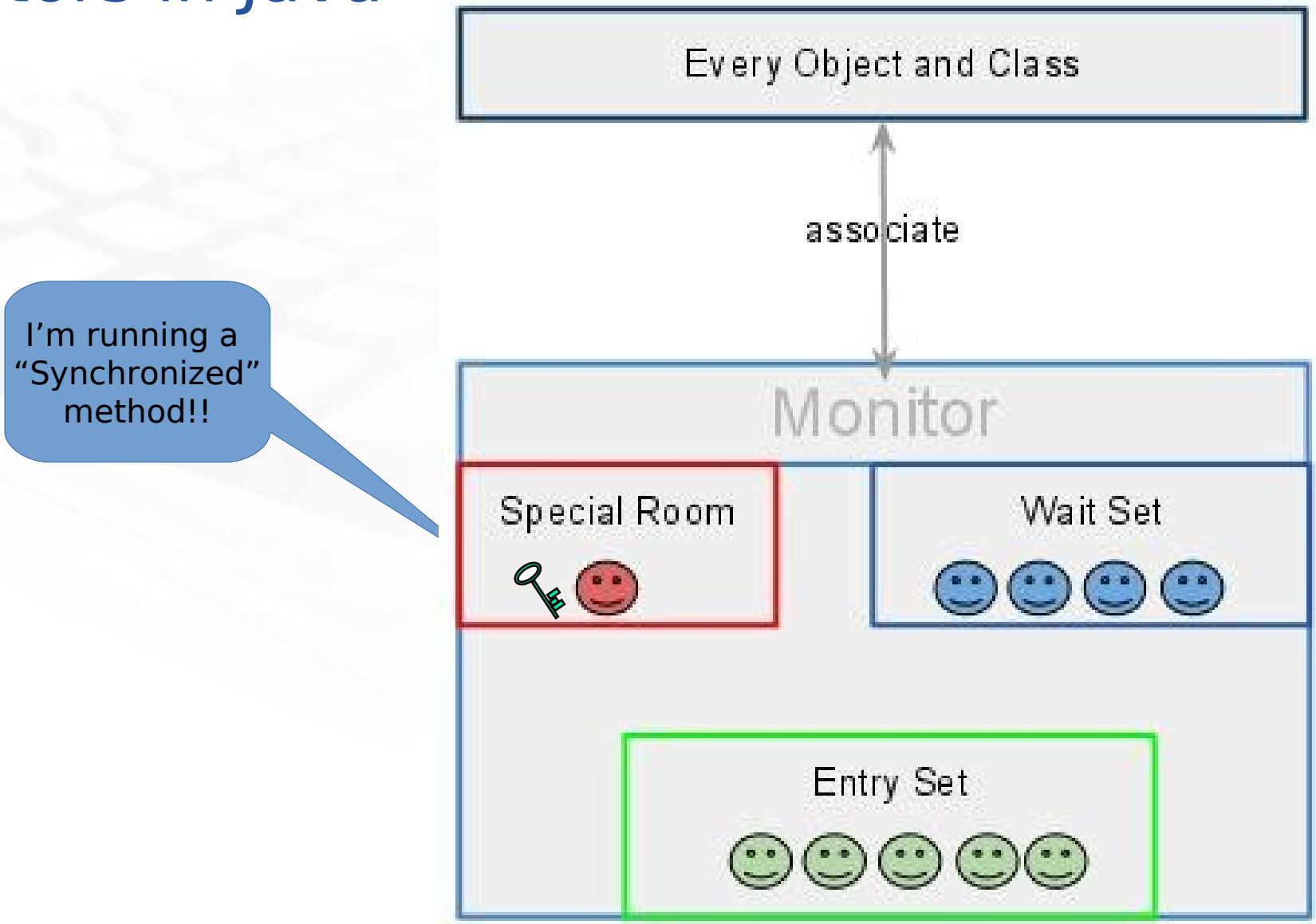
If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.

This can be prevented by synchronising access to the data.

Use “Synchronized” method:

```
public synchronized void update()
{
   …
}
```

Monitors in Java

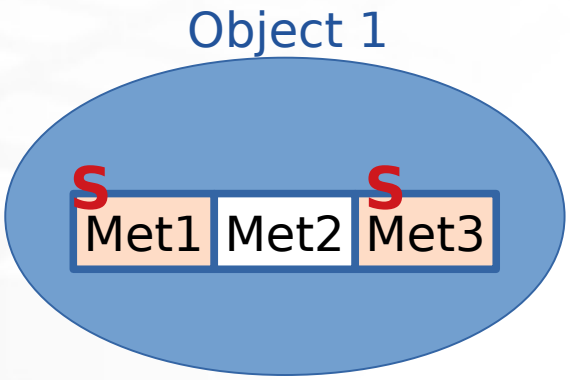


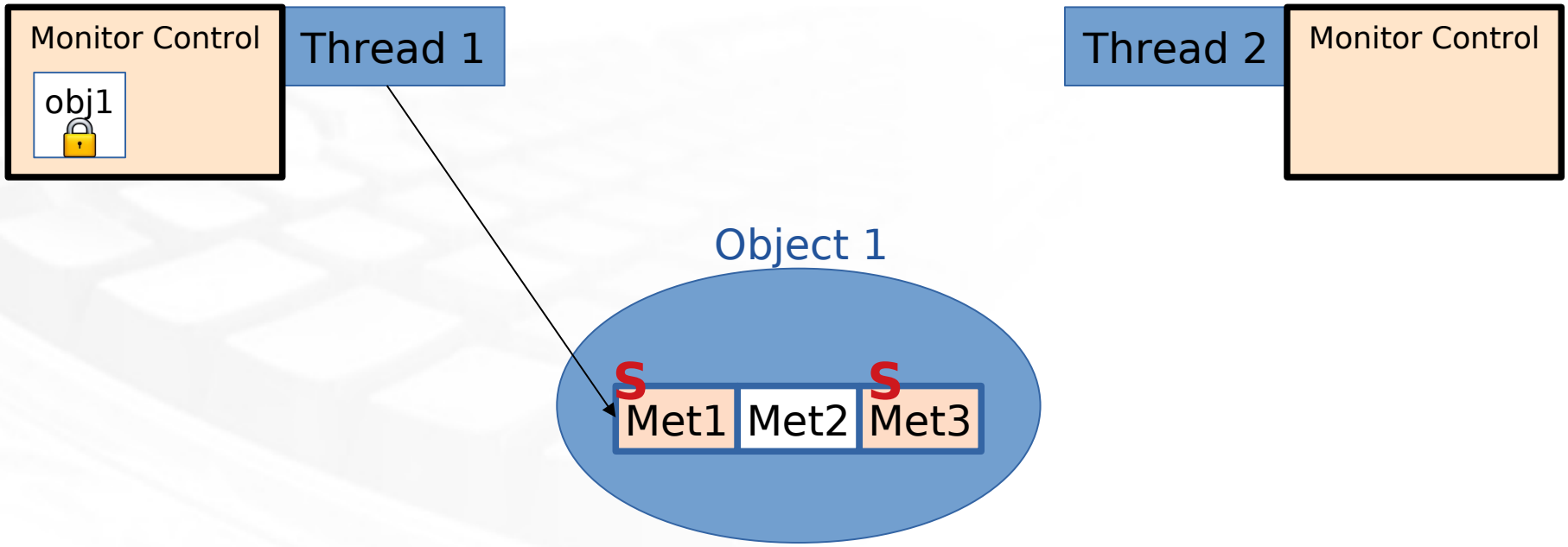
Monitor Control

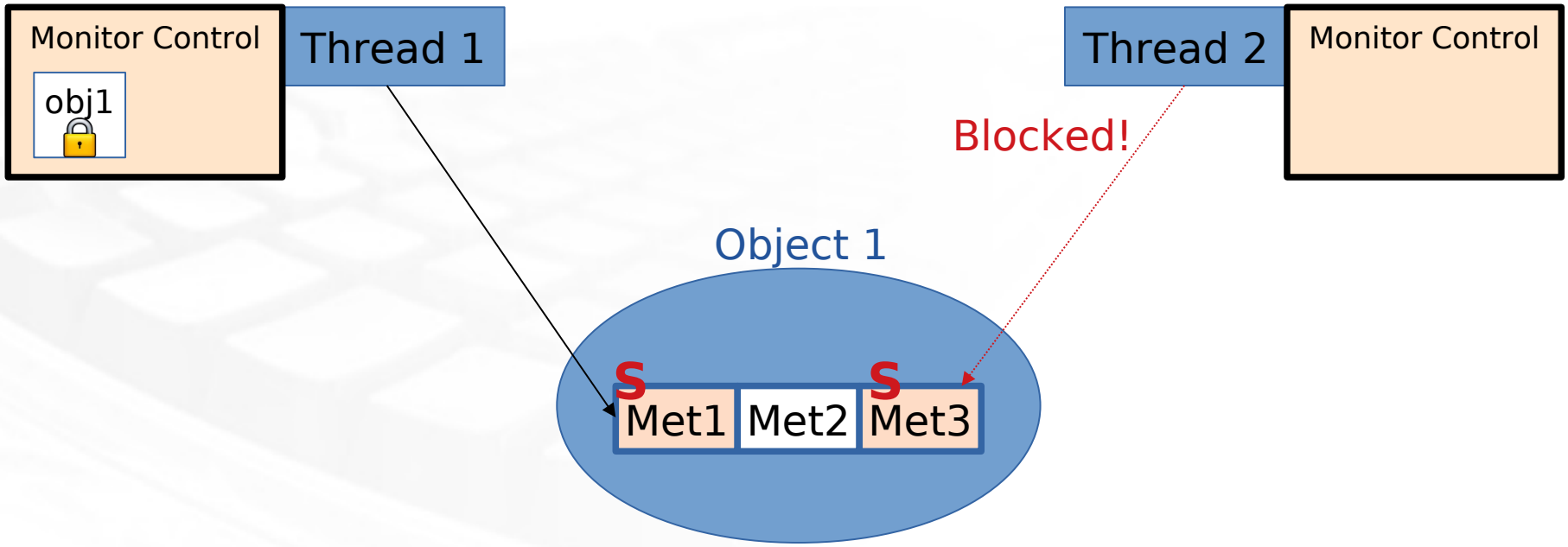
Thread 1

Thread 2

Monitor Control




Monitor Control
obj1
Thread 1
Thread 2
Monitor Control
Object 1
S
S
Met1
Met2
Met3


Monitor Control
obj1
Thread 1
Thread 2
Monitor Control
Blocked!
Object 1
S
S
Met1
Met2
Met3

Monitor (shared object access):

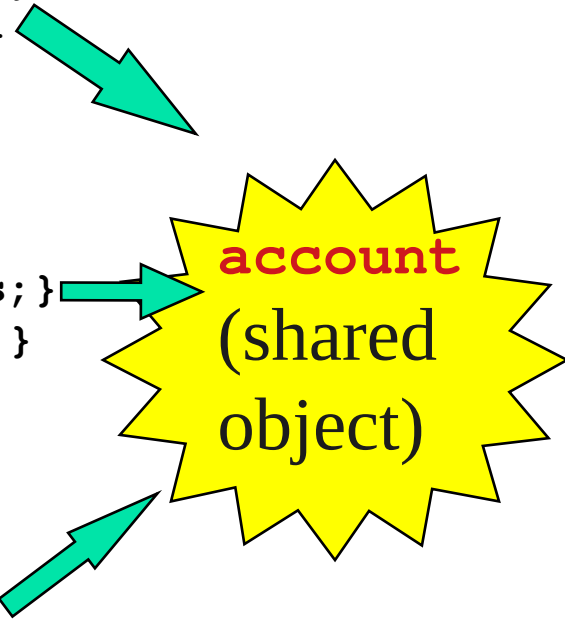
```
class Account {    // the 'monitor'
    int balance;

    // if 'synchronized' is removed, the outcome is unpredictable
     public synchronized void deposit(int deposit_amount){
        // METHOD BODY : balance += deposit_amount;
      }

    public synchronized void withdraw(int deposit_amount){   //
        //METHOD BODY: balance -= deposit_amount;
      }

    public synchronized void enquire( ) {
       // METHOD BODY: display balance… should it be synch.?
     }
}
```

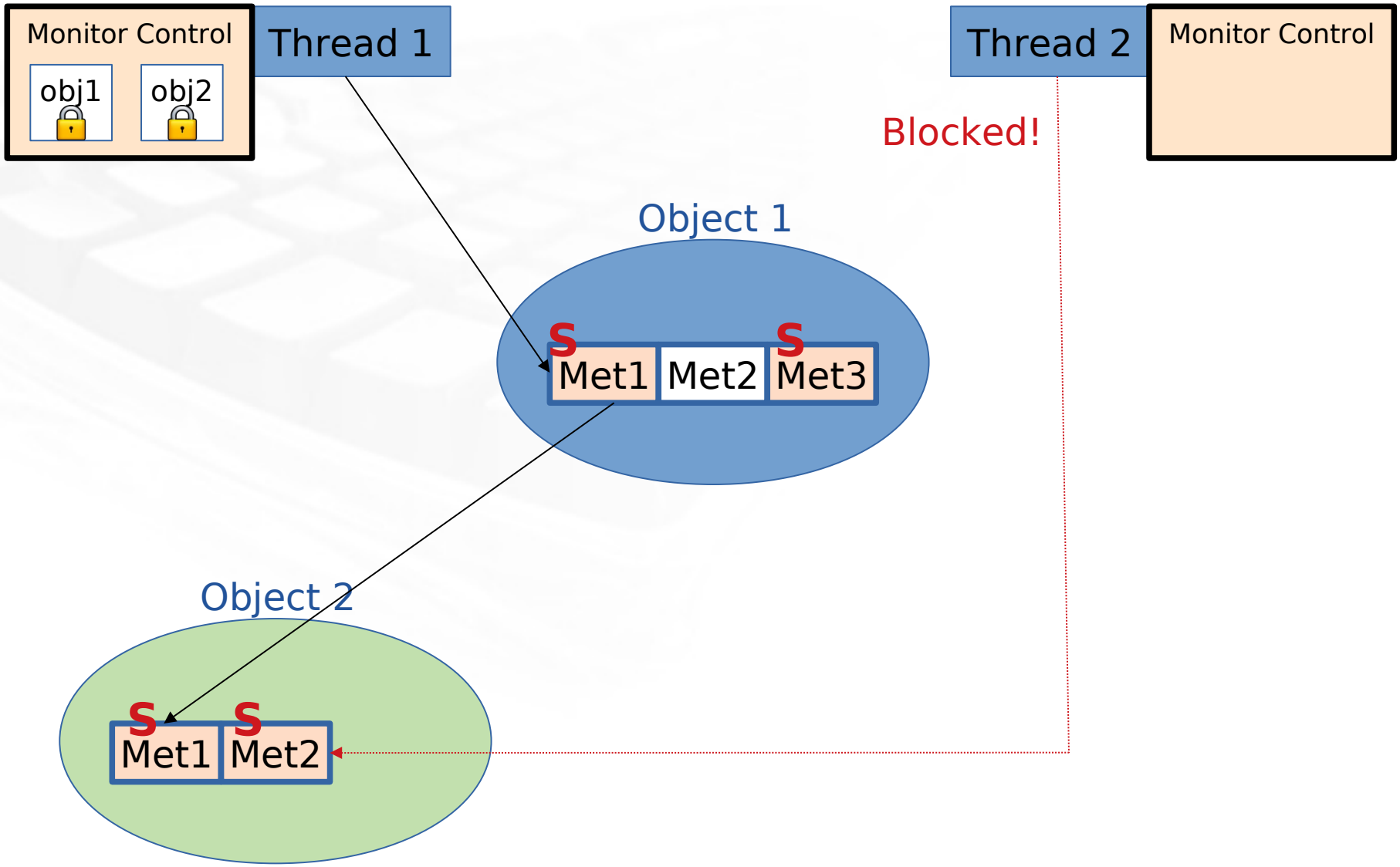
3 Threads sharing the same object

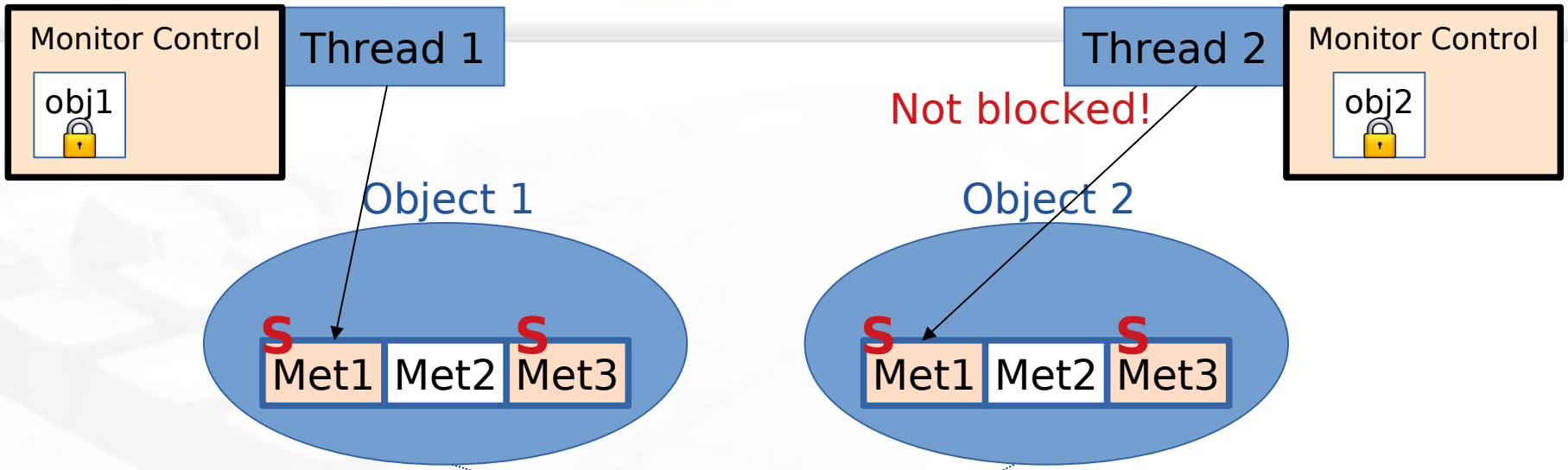
```
class InternetBankingSystem {
  public static void main(String [] args  ) {
    Account accountObject = new Account ();
    Thread t1 = new Thread(new MyThread(accountObject));
    Thread t2 = new Thread(new YourThread(accountObject));
    Thread t3 = new Thread(new HerThread(accountObject));
             t1.start();
             t2.start();
             t3.start();
           // DO some other operation
    } // end main()
}
```

Shared account object between 3 threads

```
class MyThread implements Runnable  {
 Account account;
        public MyThread (Account s) {  account = s;}
        public void run() { account.deposit(...); }
} // end class MyThread

class YourThread implements Runnable  {
 Account account;
        public YourThread (Account s) { account = s;}
        public void run() { account.withdraw(...); }

} // end class YourThread

class HerThread implements Runnable  {
 Account account;
        public HerThread (Account s) { account = s; }
        public void run() {account.enquire(); }
} // end class HerThread
```



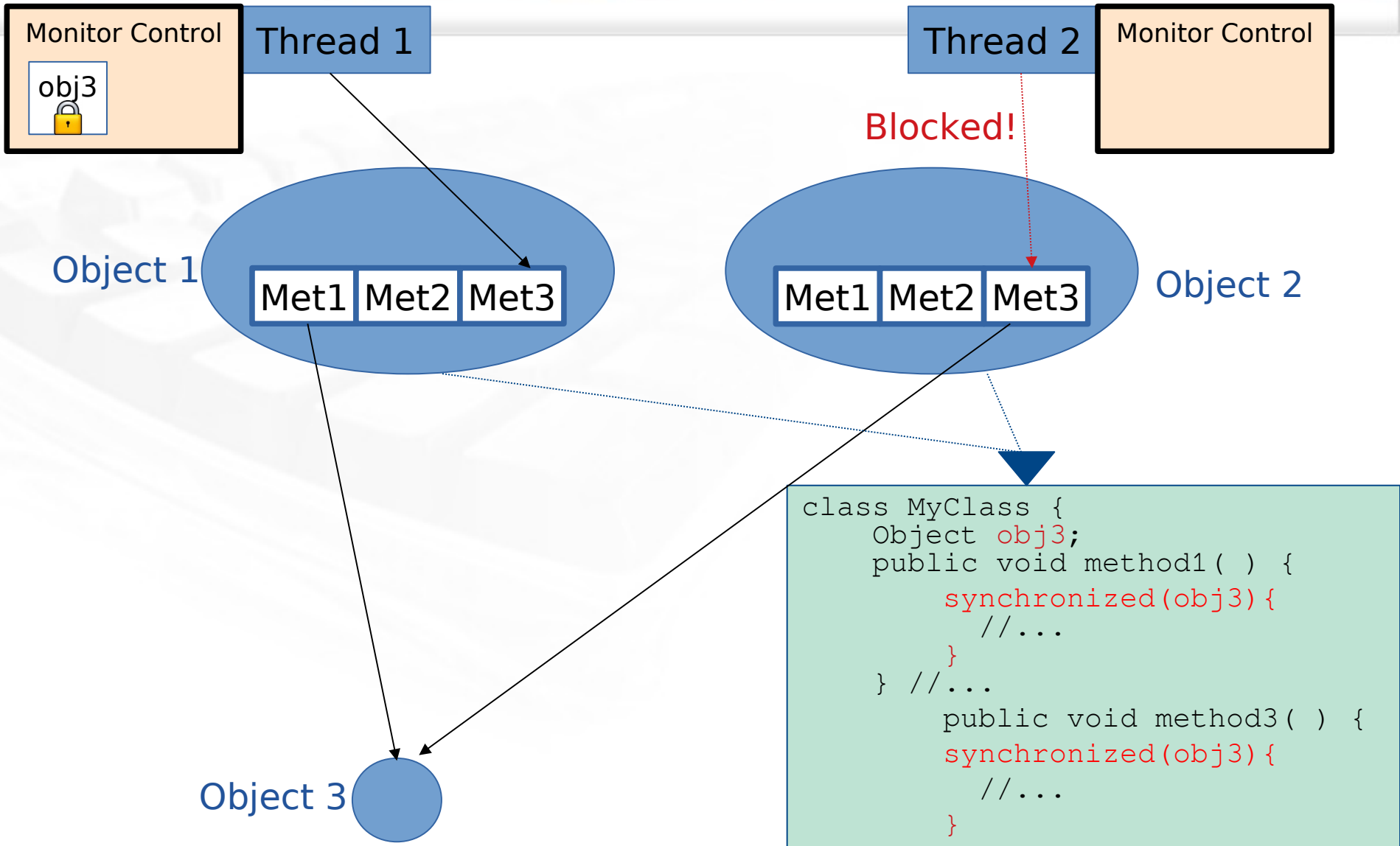

Monitor Control
obj1
obj2
Thread 1
Thread 2
Monitor Control
Blocked!
Object 1
S
Met1
Met2
S
Met3
Object 2
S
Met1
S
Met2



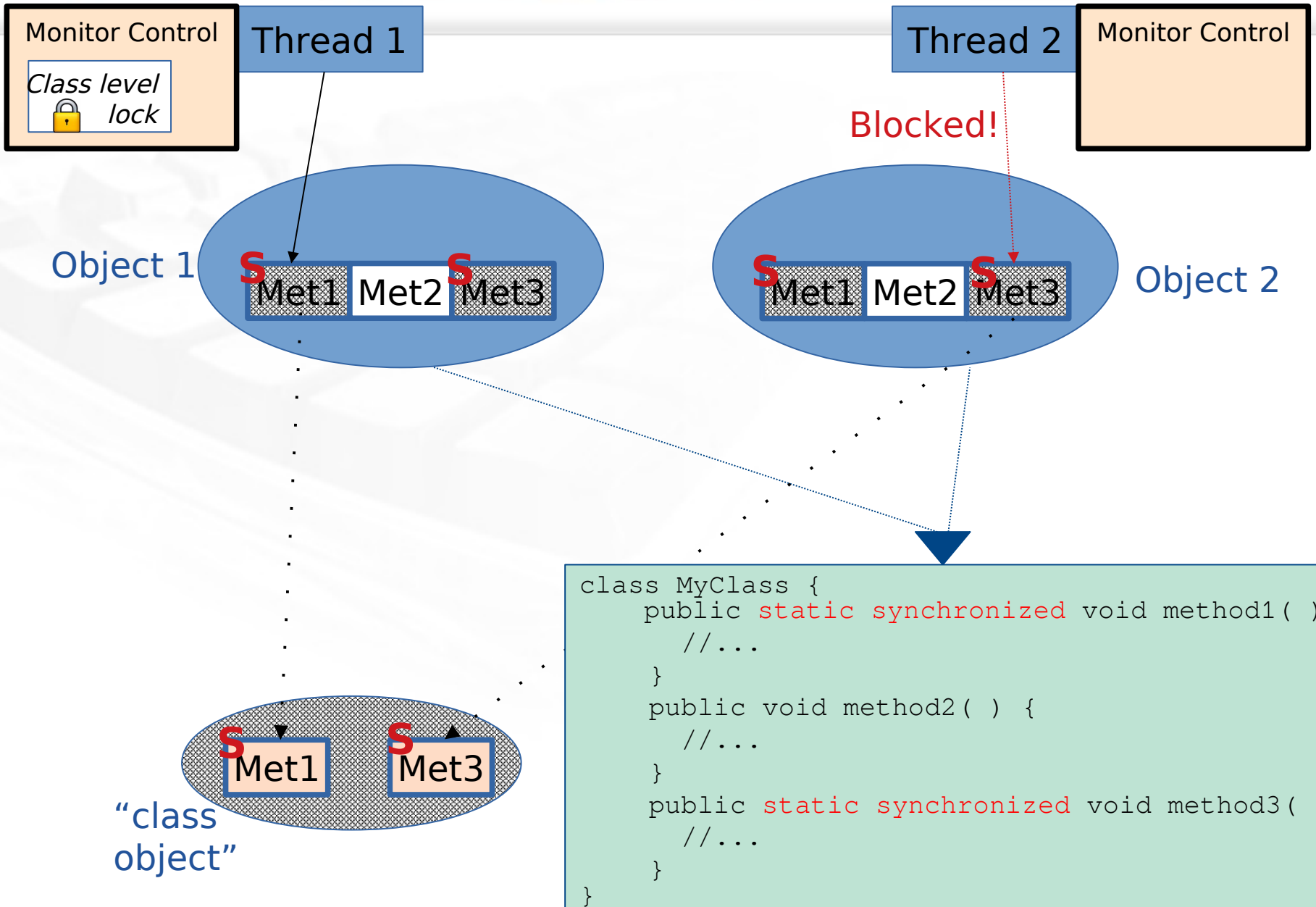

```
class MyMonitor {   // the 'monitor'
    public synchronized void method1( ) {
        for (int i = 0; i < 1000; i++)
            System.out.print("1");
     } //...
     public synchronized void method3( ) {
        for (int i = 0; i < 1000; i++)
            System.out.print("3");
     }
}
```

2 Threads sharing the same object?!?

```
public class TestSync {
   public static void main(String [] args  ) {
     MyMonitor m1= new MyMonitor();
     MyMonitor m2= new MyMonitor(); //NOT THE SAME OBJECT
     MyThread t1 = new MyThread(m1,1);
     MyThread t2 = new MyThread(m2,2);
   }
}

class MyThread extends Thread {
    MyMonitor monitor;
    int method;

    MyThread(MyMonitor monitor, int method) {
        this.monitor=monitor; this.method = method; start();
    }
    public void run() {
       if(method==1) monitor.method1();
       else if (method==2) monitor.method2();
    }
}
```



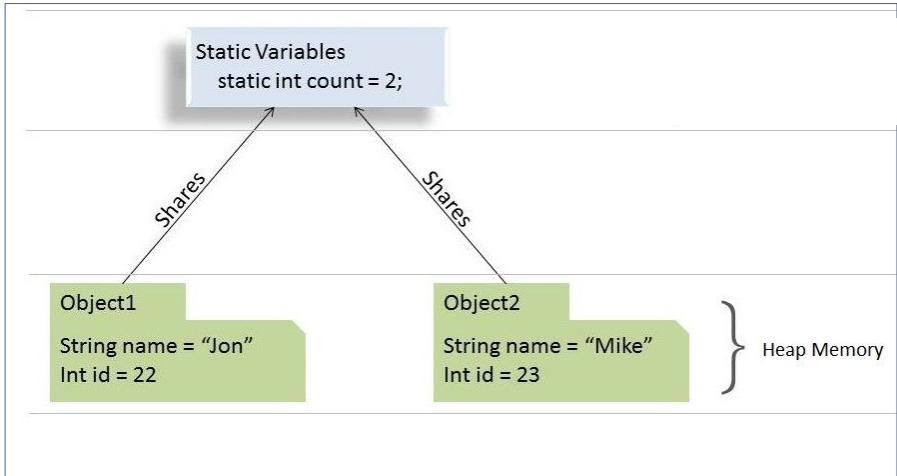

```
class MyClass {
    Object obj3;
    public void method1( ) {
        synchronized(obj3){
          //...
        }
    } //...
        public void method3( ) {
        synchronized(obj3){
          //...
        }
    }
}
```




```
class MyClass {
    public static synchronized void method1( ) {
       //...
     }
     public void method2( ) {
       //...
     }
     public static synchronized void method3( ){
       //...
     }
}
```

2 Threads sharing the same object !?

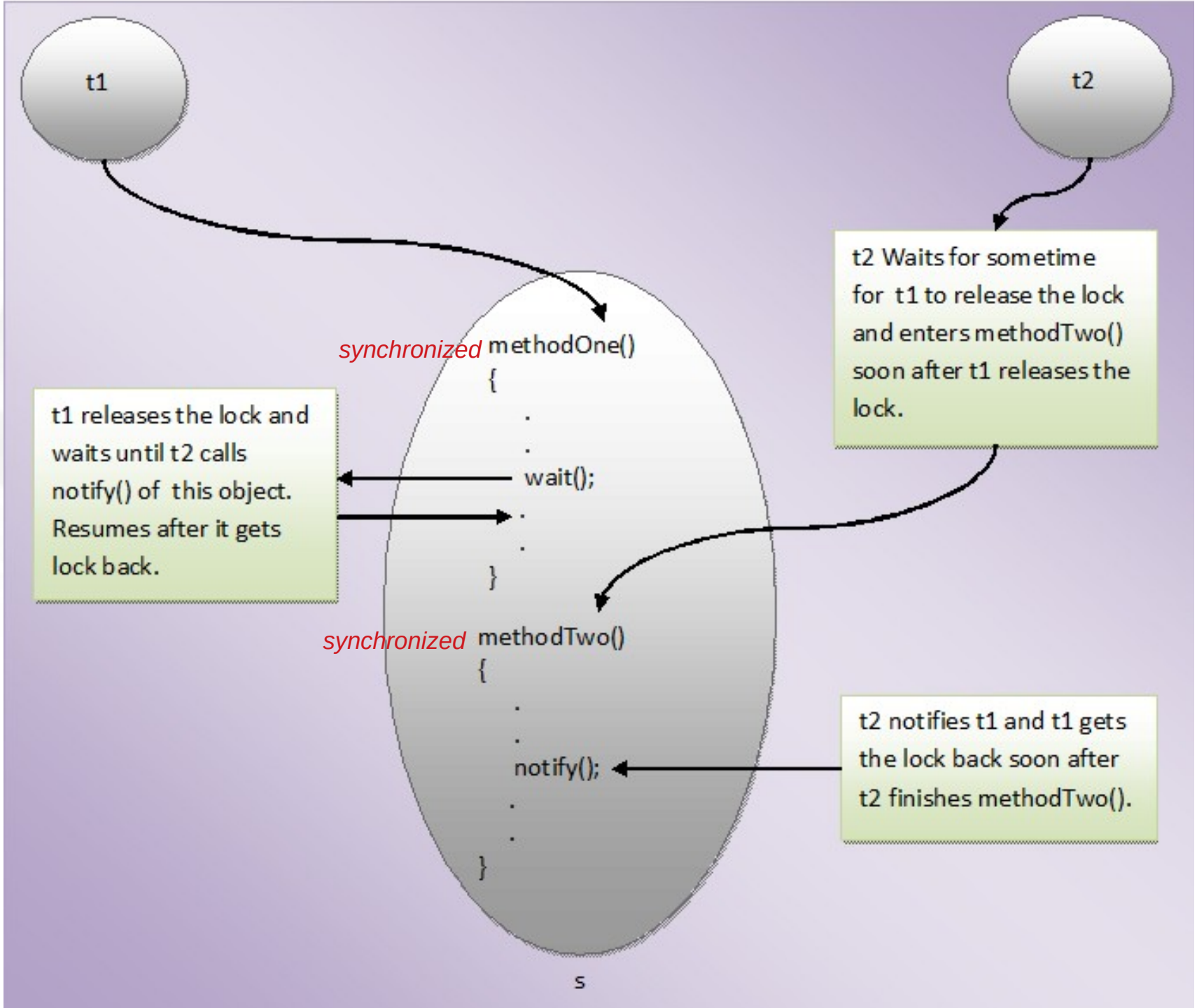
```
class MyMonitor {   // the 'monitor'
    public static synchronized void method1( ) {
       for (int i = 0; i < 1000; i++) {
          System.out.print("1");
       }

     } //...

    public static synchronized void method2( ) {
       for (int i = 0; i < 1000; i++){
          System.out.print("2");
       }
    }
}
```



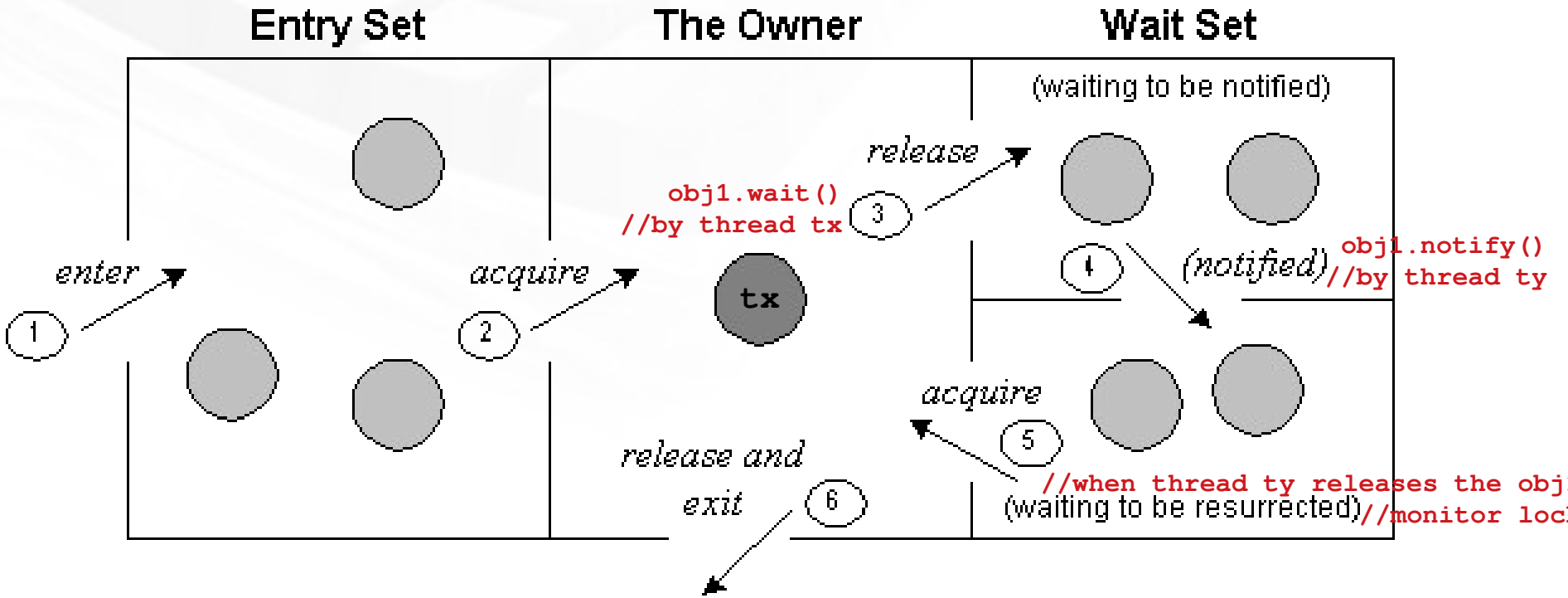
Como bloquear em um monitor esperando por alguma condição?

- O método **Object.wait()** interrompe a Thread atual, ou seja, coloca a mesma para “dormir” até que uma outra Thread use o método “**Object.notify()**” no mesmo objeto para “acordá-la”.
  - A thread que executa wait(), **abre mão do monitor**
  - Esse é o comportamento de uma **“variável de condição”**

```
try {
    wait();  //this.wait()
} catch (InterruptedException e) { }
```


t1
t2
t2 Waits for sometime for t1 to release the lock and enters methodTwo() soon after t1 releases the lock.
synchronized methodOne()
{
.
.
wait();
.
.
}
t1 releases the lock and waits until t2 calls notify() of this object. Resumes after it gets lock back.
synchronized methodTwo()
{
.
.
notify();
.
.
}
t2 notifies t1 and t1 gets the lock back soon after t2 finishes methodTwo().
s

E se várias threads executarem wait() dentro do mesmo monitor?



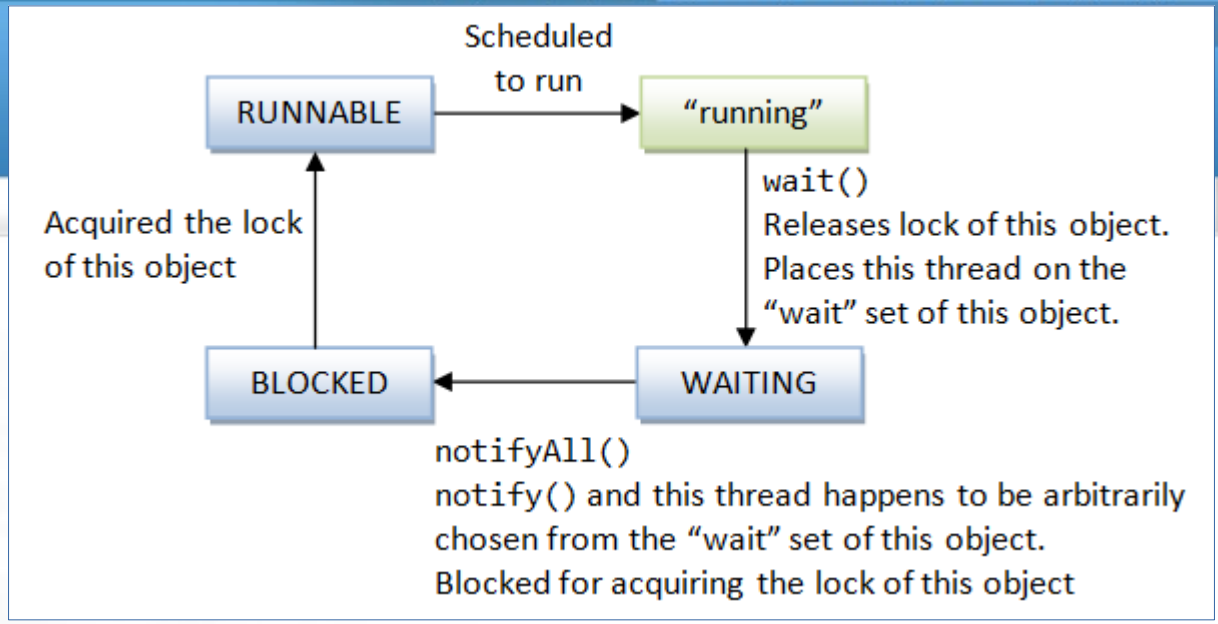
notifyAll()

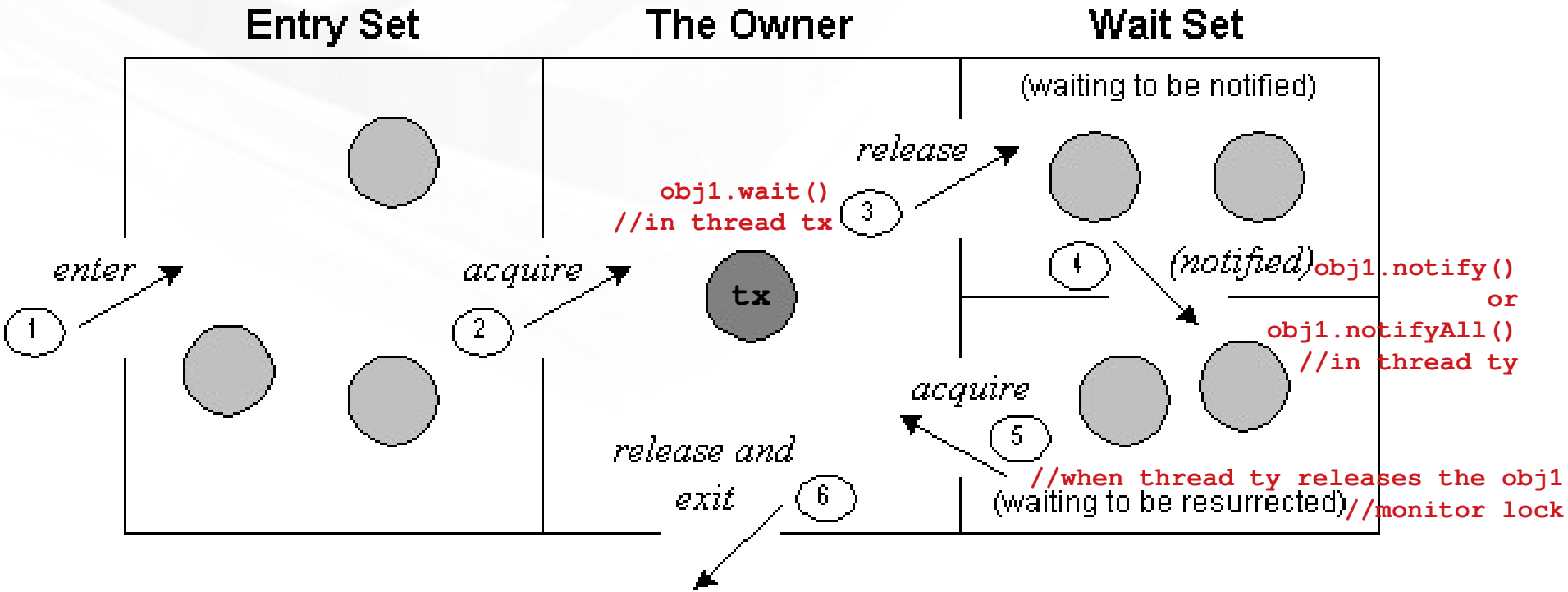
- Como ao executar um notify(), não se é possível saber qual thread será acordada (dentre as que fizeram wait()), uma solução é acordar todas as threads que estavam bloqueadas

Object.notifyAll()

... Nesse caso, a thread deve se bloquear dentro de um loop:

```
while (anyValue == false) { //devo desbloquear mesmo?
            try {
                wait();  //this.wait()
            } catch (InterruptedException e) { }
}
```


Scheduled
to run
RUNNABLE
"running"
wait()
Releases lock of this object.
Places this thread on the
"wait" set of this object.
Acquired the lock
of this object
BLOCKED
WAITING
notifyAll()
notify() and this thread happens to be arbitrarily
chosen from the "wait" set of this object.
Blocked for acquiring the lock of this object


Entry Set
The Owner
Wait Set
(waiting to be notified)
enter
1
acquire
2
tx
obj1.wait()
//in thread tx
release
3
4
(notified)
obj1.notify()
or
obj1.notifyAll()
//in thread ty
acquire
5
release and
exit
6
//when thread ty releases the obj1
(waiting to be resurrected)//monitor lock

Produtor-Consumidor

```
public class ProducerConsumer{

    public static void main(String[] args) {

        MyBuffer b = new MyBuffer();

        Producer p1 = new Producer(b, 1);
        Consumer c1 = new Consumer(b, 1);

        p1.start();
        c1.start();
    }
}
```

Produtor-Consumidor

```
class MyBuffer {
    private int contents;
    private boolean available = false;

    public synchronized int get(int who) {
        while (available == false) {
            try {
                wait();  //this.wait()
            } catch (InterruptedException e) { }
        }
        available = false;
        System.out.format("Consumer %d got: %d%n", who, contents);
        notifyAll(); //in this case, notify() would work… why?!
        return contents;
    }
```

Produtor-Consumidor

```
...
    public synchronized void put(int who, int value) {
        while (available == true) {
            try {
                wait();  //this.wait()
            } catch (InterruptedException e) { }
        }
        contents = value;
        available = true;
        System.out.format("Producer %d put: %d%n", who, contents);
        notifyAll(); //in this case, notify() would work… why?!
    }
}
```

Produtor-Consumidor

```
class Producer extends Thread {

    private MyBuffer buffer;
    private int number;

    public Producer(MyBuffer b, int number) {
        buffer = b;
        this.number = number;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            buffer.put(number, i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
        }
    }
}
```

Produtor-Consumidor

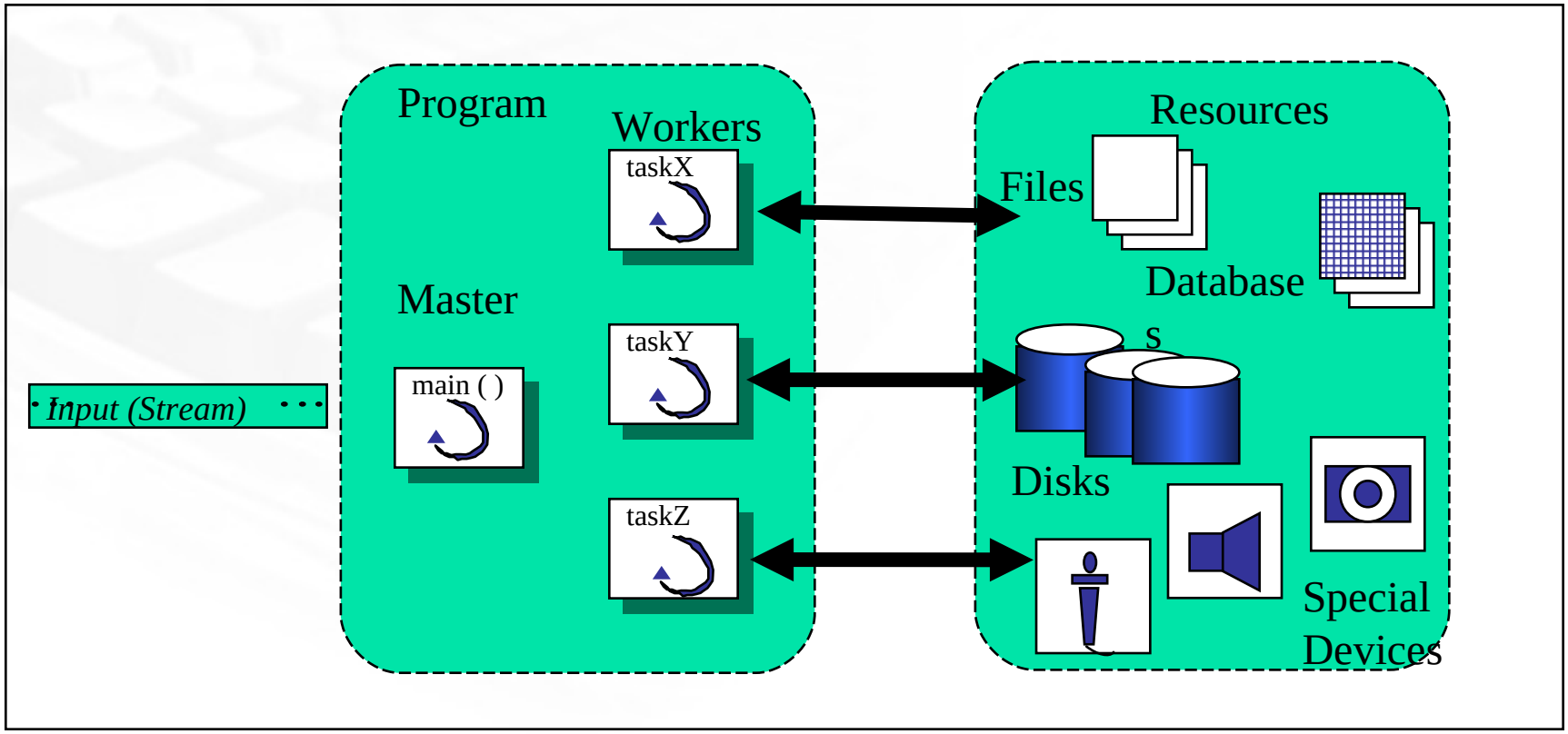
```
class Consumer extends Thread {

    private MyBuffer buffer;
    private int number;

    public Consumer(MyBuffer b, int number) {
        buffer = b;
        this.number = number;
    }

    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = buffer.get(number);
        }
    }
}
```

Produtor-Consumidor Exercício:

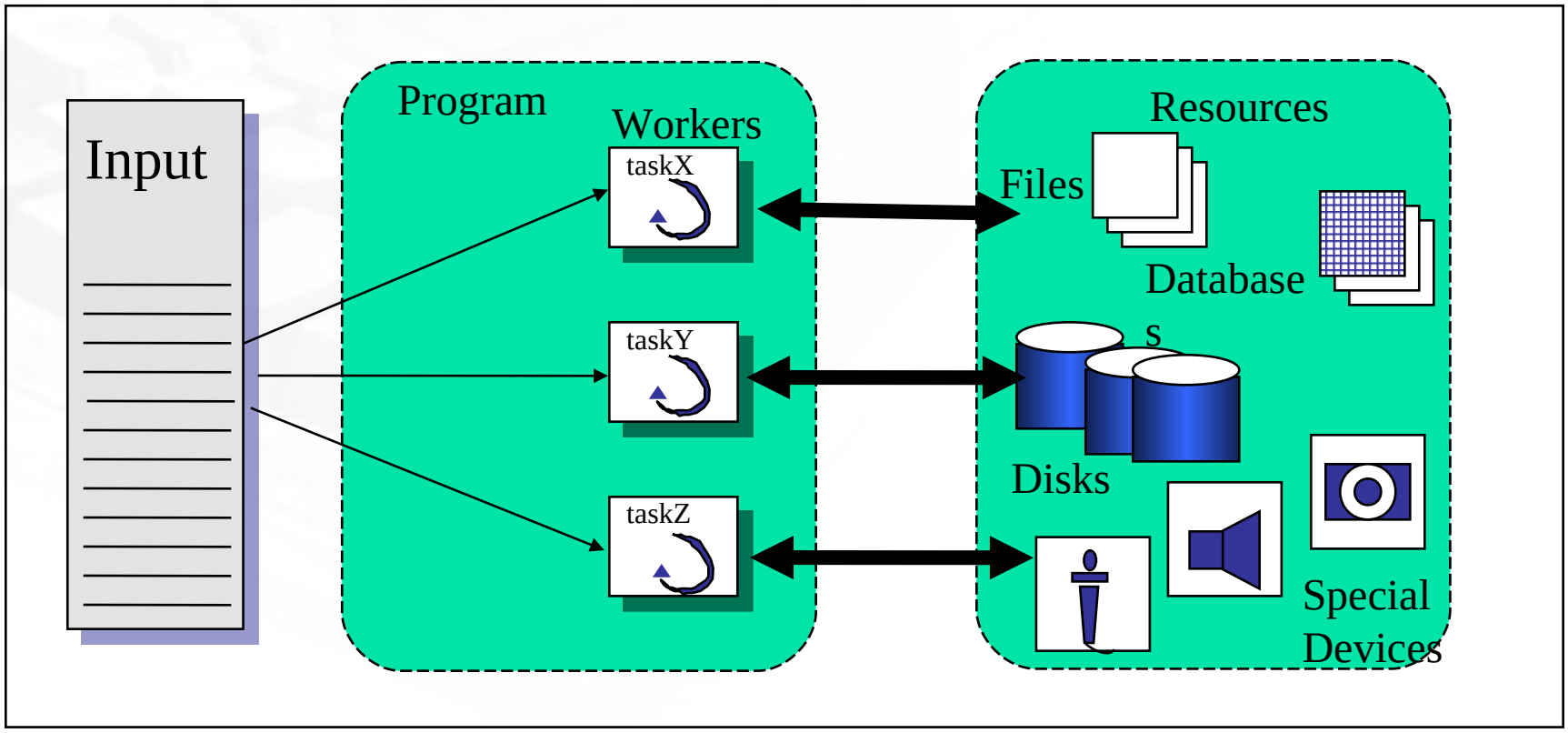
- Faça as seguintes modificações:
  - Use notify() ao invés de notifyAll() no método get()
  - Na thread main, crie vários producers e consumers

Essa solução pode gerar deadlock?

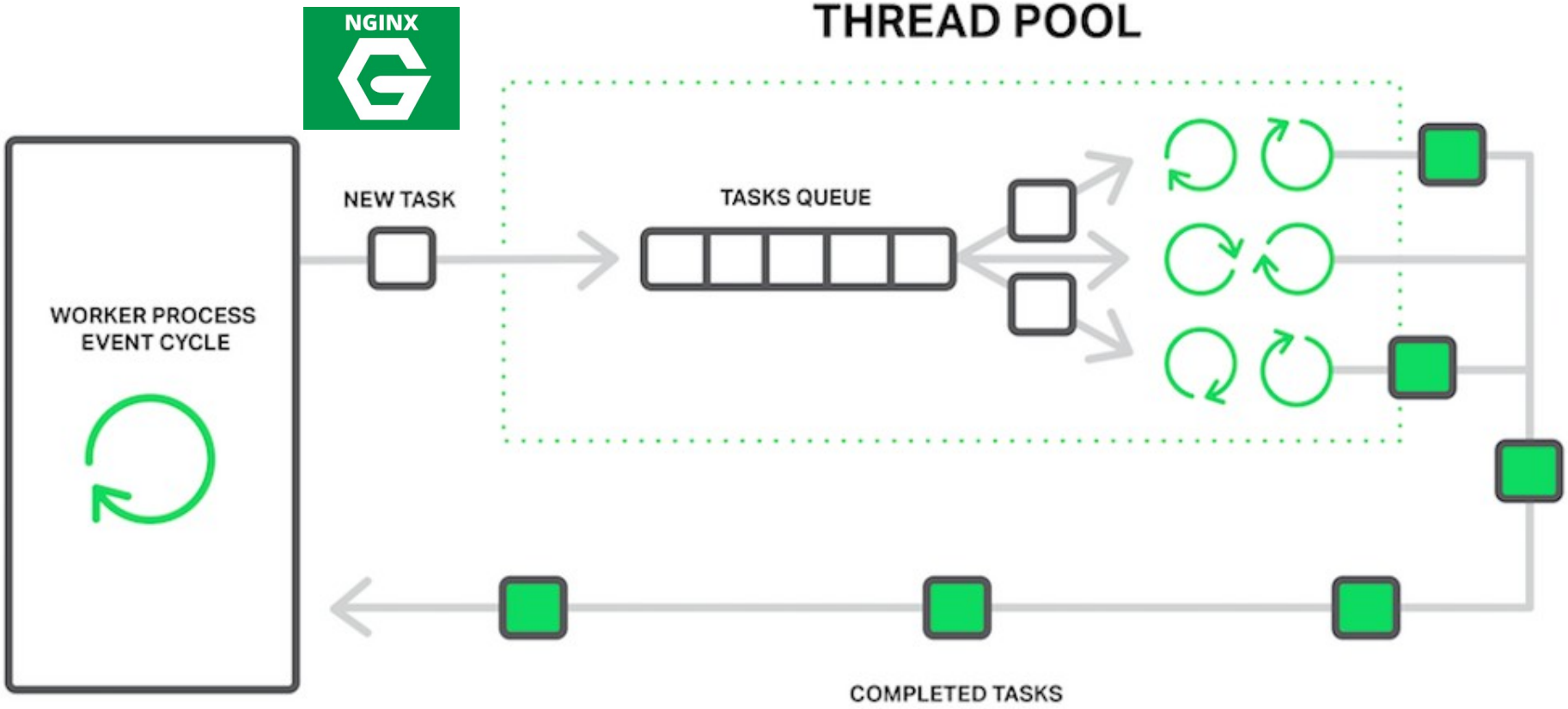
The master/worker model



The peer model



NGINX


NGINX
THREAD POOL
NEW TASK
TASKS QUEUE
WORKER PROCESS
EVENT CYCLE
COMPLETED TASKS