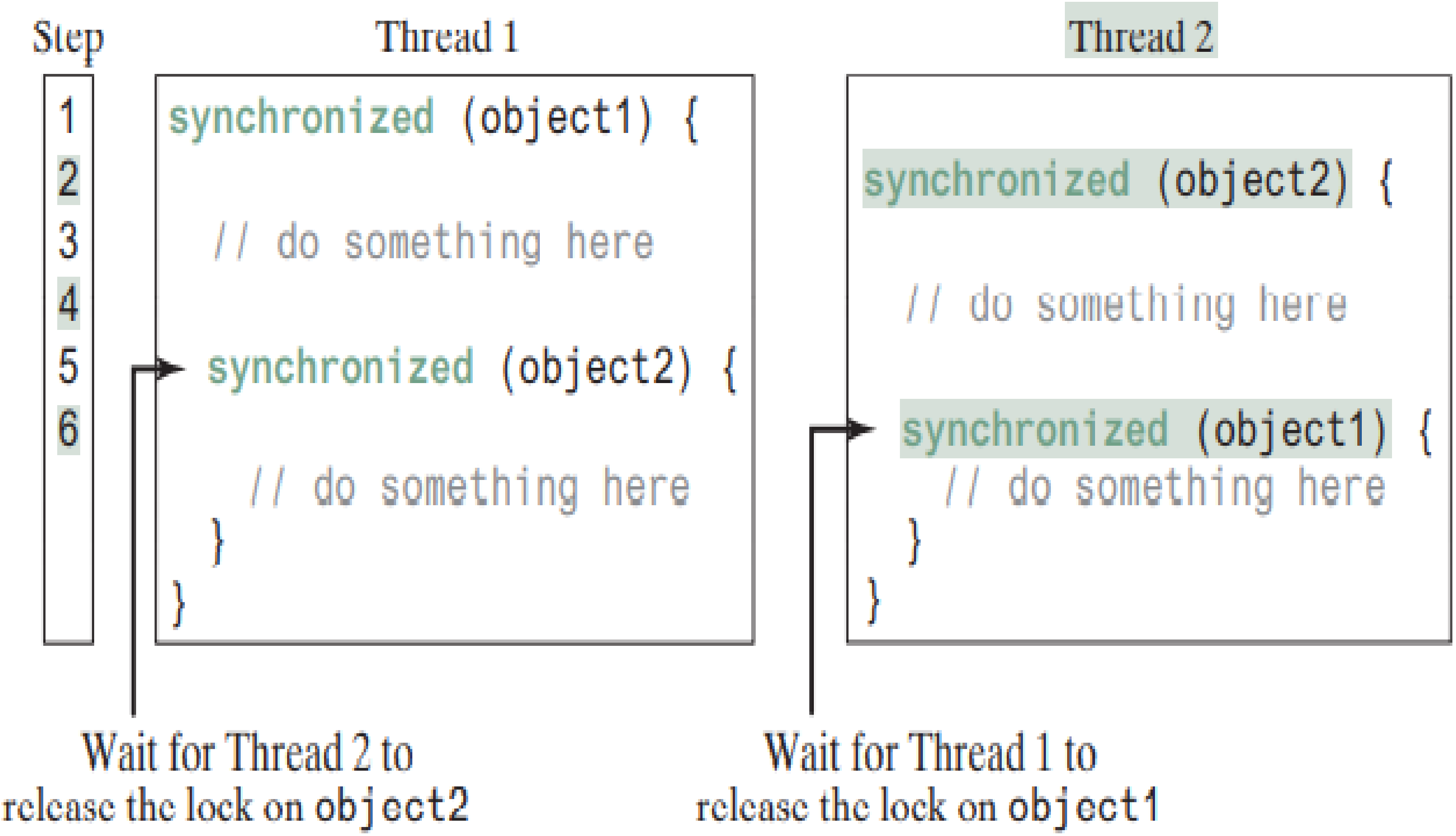
Deadlock

Deadlock

- **Deadlock** is a programming situation where two or more threads are blocked forever, this situation arises with at least two threads and two or more resources.
- A deadlock has the following characteristics:
  - It is two threads, each waiting for a lock from the other.
  - It is not detected or avoided.
  - Deadlock can be avoided by:
    - Deciding on the order to obtain locks
    - Adhering to this order throughout
    - Releasing locks in reverse order

Example



Step

1
2
3
4
5
6

Thread 1

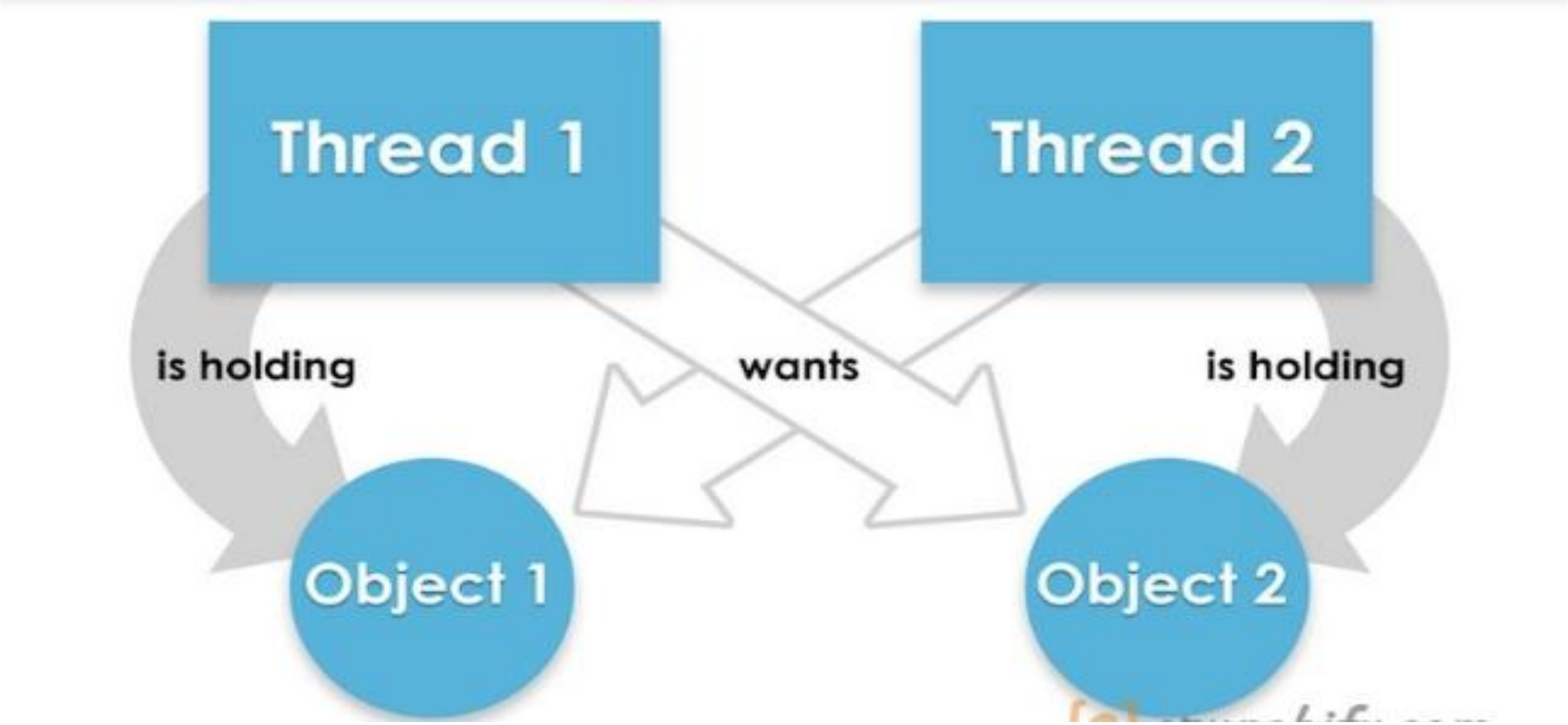
```
synchronized (object1) {

  // do something here

  synchronized (object2) {

    // do something here
  }
}
```

Thread 2

```

synchronized (object2) {

  // do something here

  synchronized (object1) {
    // do something here
  }
}
```

Wait for Thread 2 to release the lock on `object2`

Wait for Thread 1 to release the lock on `object1`



Java technology neither detects nor attempts to avoid this condition. It is the responsibility of the programmer to ensure that a deadlock cannot arise. A general rule of thumb for avoiding a deadlock is: If you have multiple objects that you want to have synchronized access to, make a global decision about the order in which you will obtain those locks, and adhere to that order throughout the program. Release the locks in the reverse order that you obtained them.

Deadlock

```
Object A = new Object();
 Object B = new Object();

Thread 1:
synchronized(A)
{
    synchronized(B)
     {
    }
}
```

```
Thread 2:
synchronized(B)
{
          synchronized(A)
                    {
                     //... }


 }
```

EX-1

```
public class DeadLockDemo {
 public void method()
{
synchronized (String.class) {
 System.out.println("Aquired lock on String.class object");
    synchronized (Integer.class)
    {
    System.out.println("Aquired lock on Integer.class object");
    }
}
}
```

EX-1-contnd

```
public void method2()
{
synchronized (Integer.class)
 {
      System.out.println("Aquired lock on Integer.class object");
    synchronized (String.class)
     {     System.out.println("Aquired lock on String.class object");
    }
 } } }
```

How to Avoid Deadlock in Java?

- Deadlocks cannot be completely resolved. But we can avoid them by following basic rules mentioned below:
- **Avoid Nested Locks**: We must avoid giving locks to multiple threads, this is the main reason for a deadlock condition. It normally happens when you give locks to multiple threads.
- **Avoid Unnecessary Locks**: The locks should be given to the important threads. Giving locks to the unnecessary threads that cause the deadlock condition.
- **Using Thread Join**: A deadlock usually happens when one thread is waiting for the other to finish. In this case, we can use **join** with a maximum time that a thread will take.