Thread Synchronization

By

Uday Kumar

Synchronization

- Synchronization is the capability of control the access of multiple threads to any shared resource.
- It is better in case we want only one thread can access the shared resource at a time.
- **Why use Synchronization?**

  The synchronization is mainly used to
  - To prevent thread interference.
  - To prevent consistency problem

Contd..

- To synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called **monitors**.

- Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

- Mutual Exclusive
  - Synchronized method.
  - Synchronized block.
  - static synchronization.
- Cooperation (Inter-thread communication)

Mutual Exclusive

- Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:
  - by synchronized method
  - by synchronized block
  - by static synchronization

Understanding the concept of Lock

- Synchronization is built around an internal entity known as the lock or monitor.
- Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.
- From Java 5 the package java.util.concurrent.locks contains several lock implementations

Synchronized method

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the method returns

Syntax:

```
Void synchronized method()
{
// statements to be synchronized
}
```

Synchronized block

- Synchronized block can be used to perform synchronization on any specific resource of the method.
- Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
- If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method

Syntax:

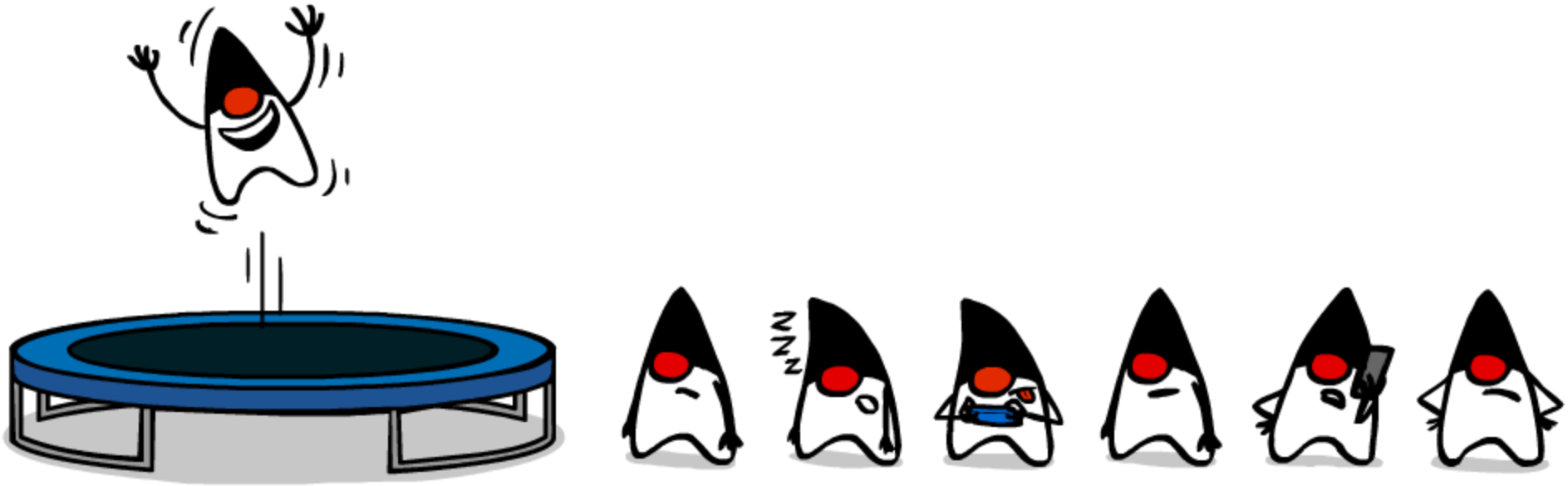
```
synchronized(object)
{
// statements to be synchronized
}
```

Static synchronization

- If you make any static method as synchronized, the lock will be on the class not on object.

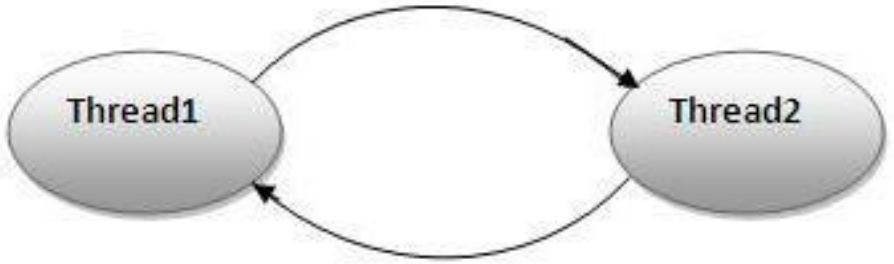
Syntax:

```
Static synchronized void method()
{
// statements to be synchronized
}
```

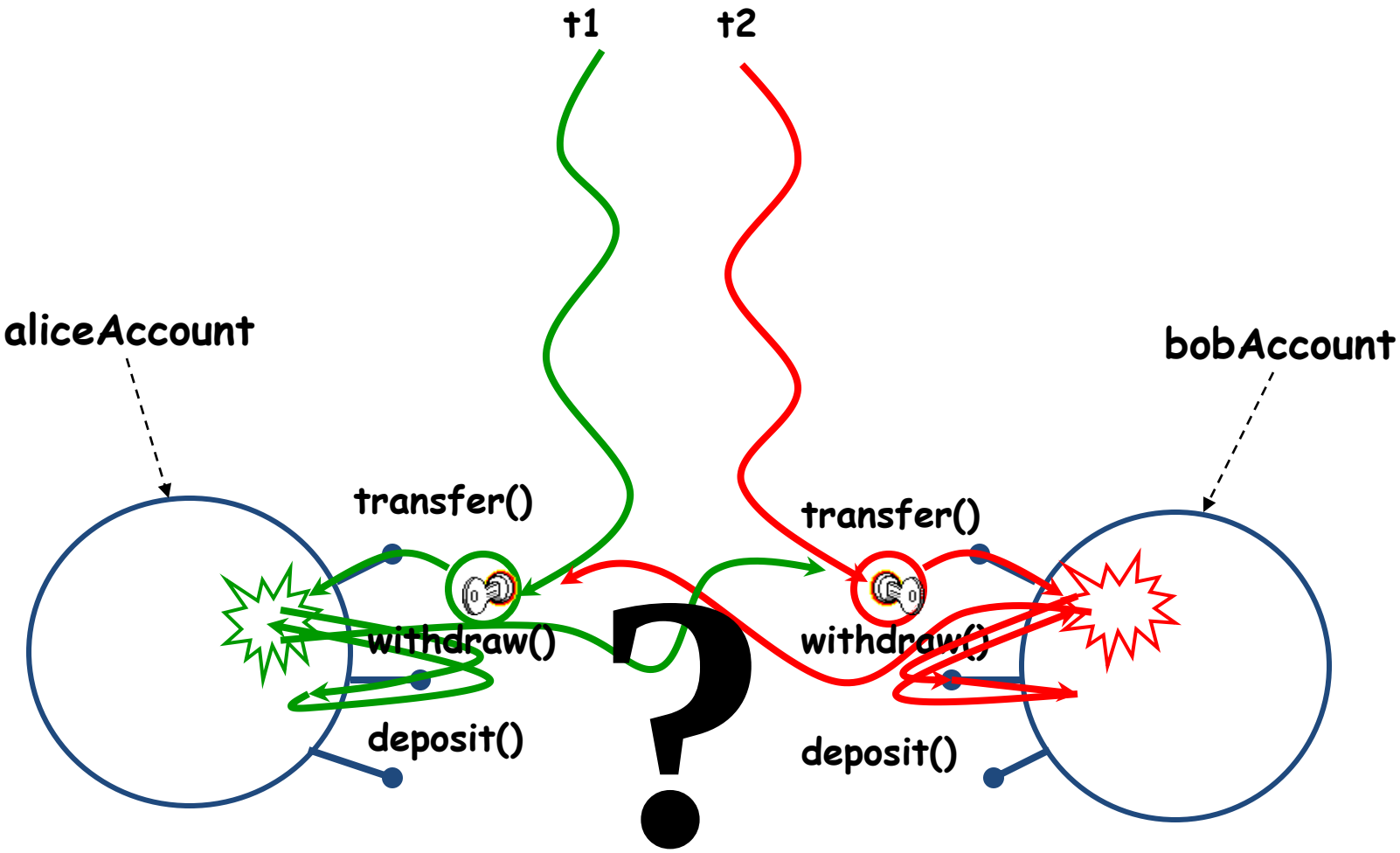


Deadlock

- Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread.
- Since, both threads are waiting for each other to release the lock, the condition is called **deadlock.**



Deadlocks


t1
t2
aliceAccount
bobAccount
transfer()
transfer()
withdraw()
withdraw()
deposit()
deposit()
?

Inter-thread communication

- Cooperation(Inter-thread communication) is all about making synchronized threads communicate with each other.
- Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.
- It is implemented by following methods of Object class:
  - wait()
  - notify()
  - notifyAll()

- **wait() method** causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
- The current thread must own this object's monitor.
- **Syntax:**
  - public final void wait()throws InterruptedException
  - public final void wait(long timeout)throws InterruptedException

- **notify() method-**Wakes up a single thread that is waiting on this object's monitor.
- If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation
  - Syntax: public final void notify()
- **notifyAll()-** Wakes up all threads that are waiting on this object's monitor
  - public final void notifyAll()

Wait(),Yield(),Sleep()

- Wait()

  - The wait() method also sends the current thread into Non-Runnable state like the sleep() method. But, the difference between the two is that in case of 'wait()' the locks are released before going into Non-Runnable state, so that any other thread that is waiting on the object

  can use it (**Unlike the Sleep method - This is the big difference**)

- Yield()

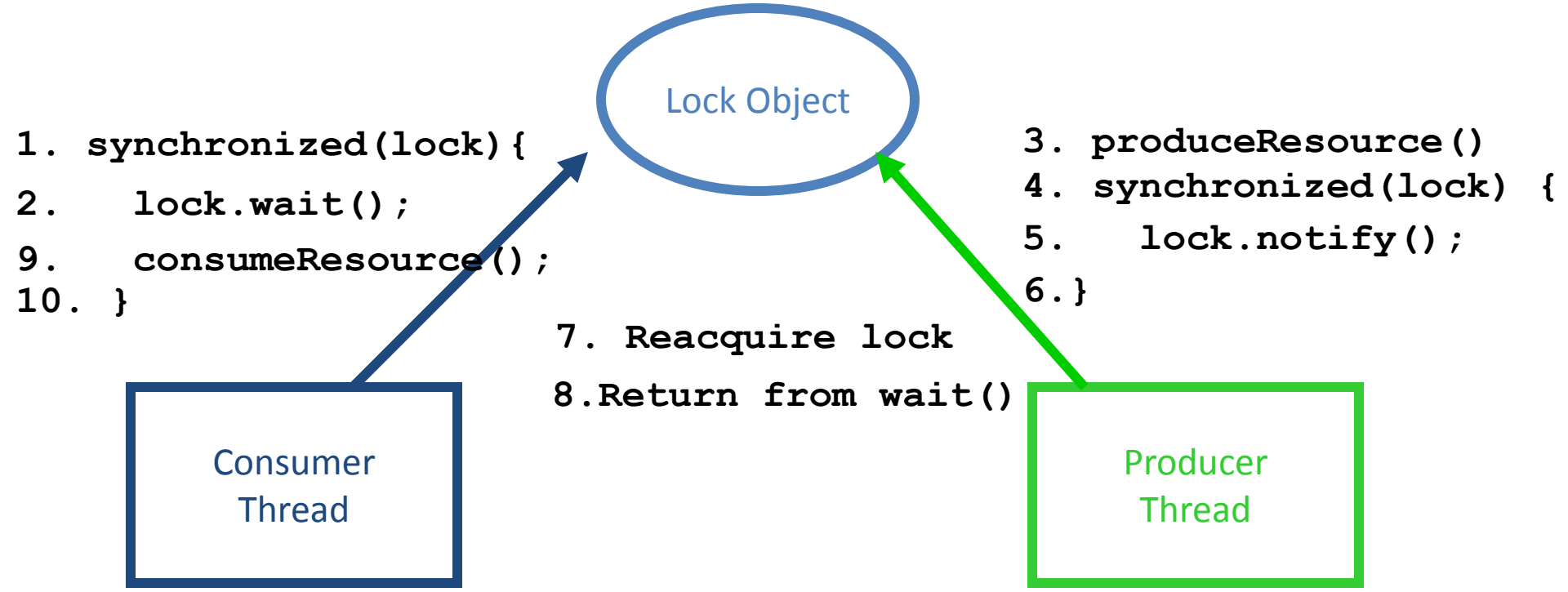
  - yield() allows the current the thread to release its lock from the object and scheduler gives the lock of the object to the other thread with same priority.

- Sleep()

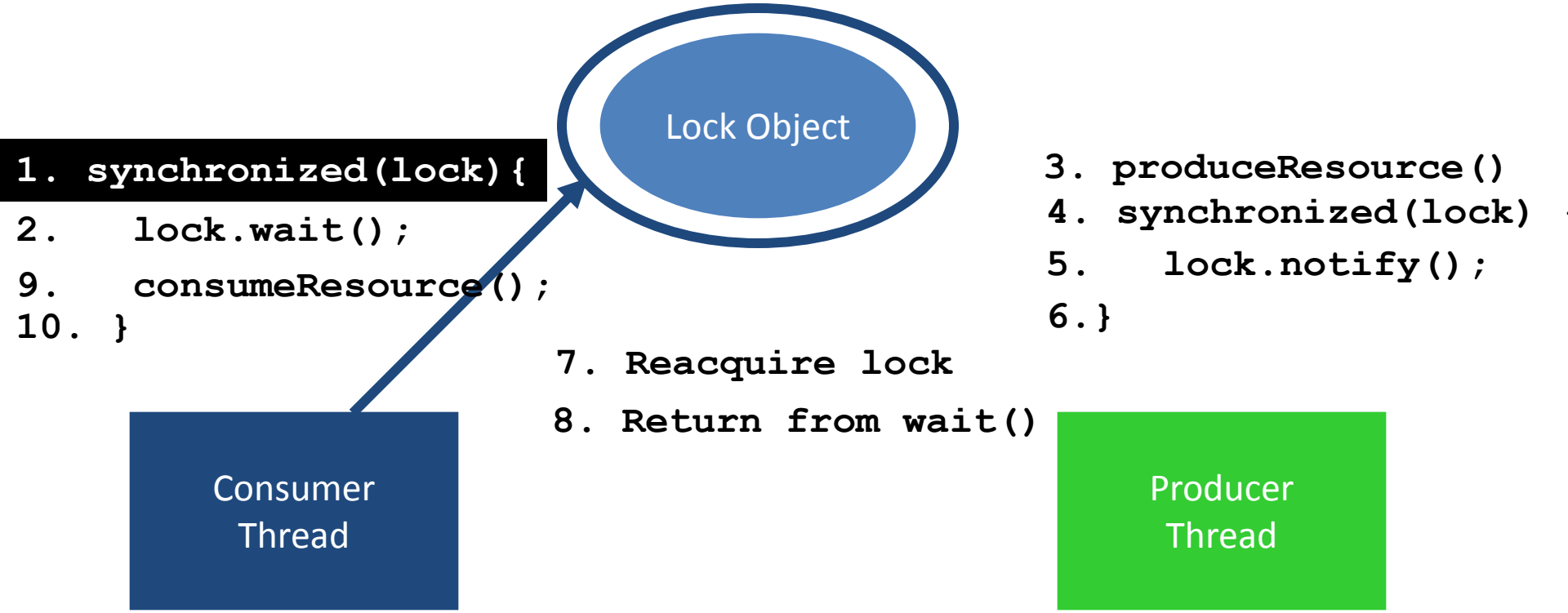
  - Thread.sleep(long milliseconds):

  - The thread's sleep method sends the current thread into Non-Runnable state for the specified amount of time.

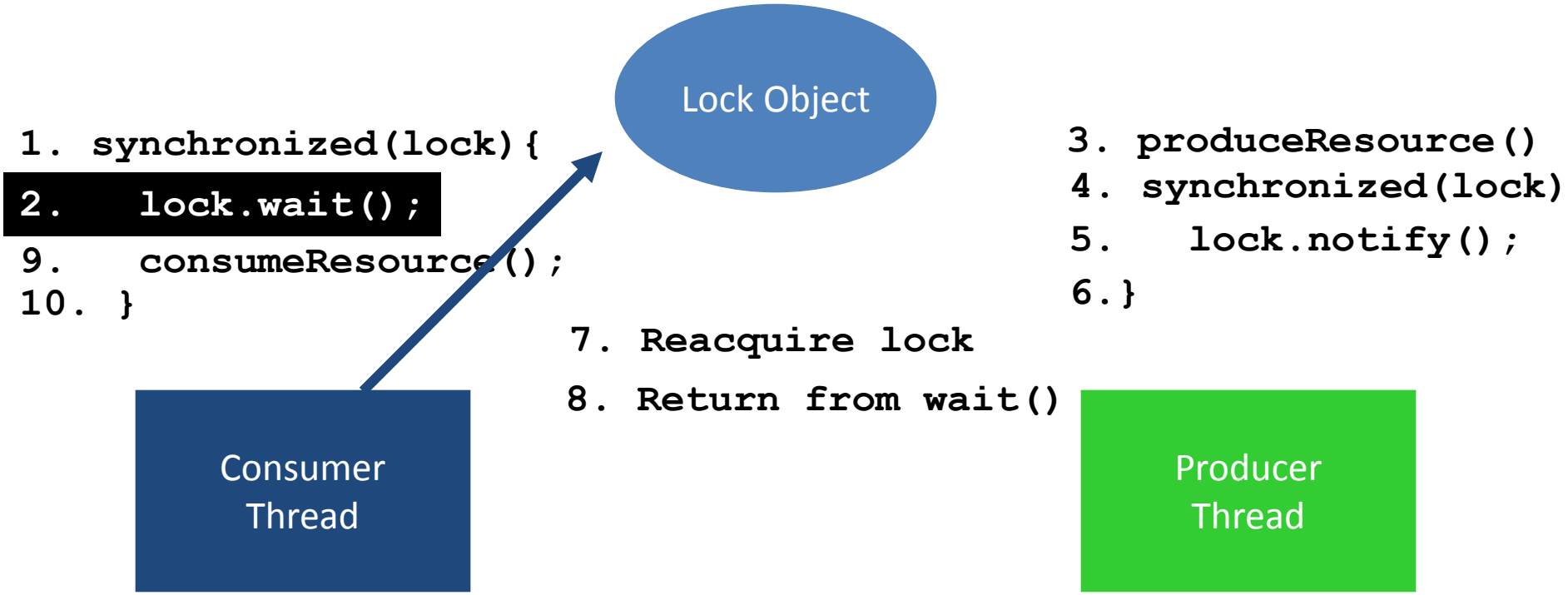
Wait/Notify Sequence



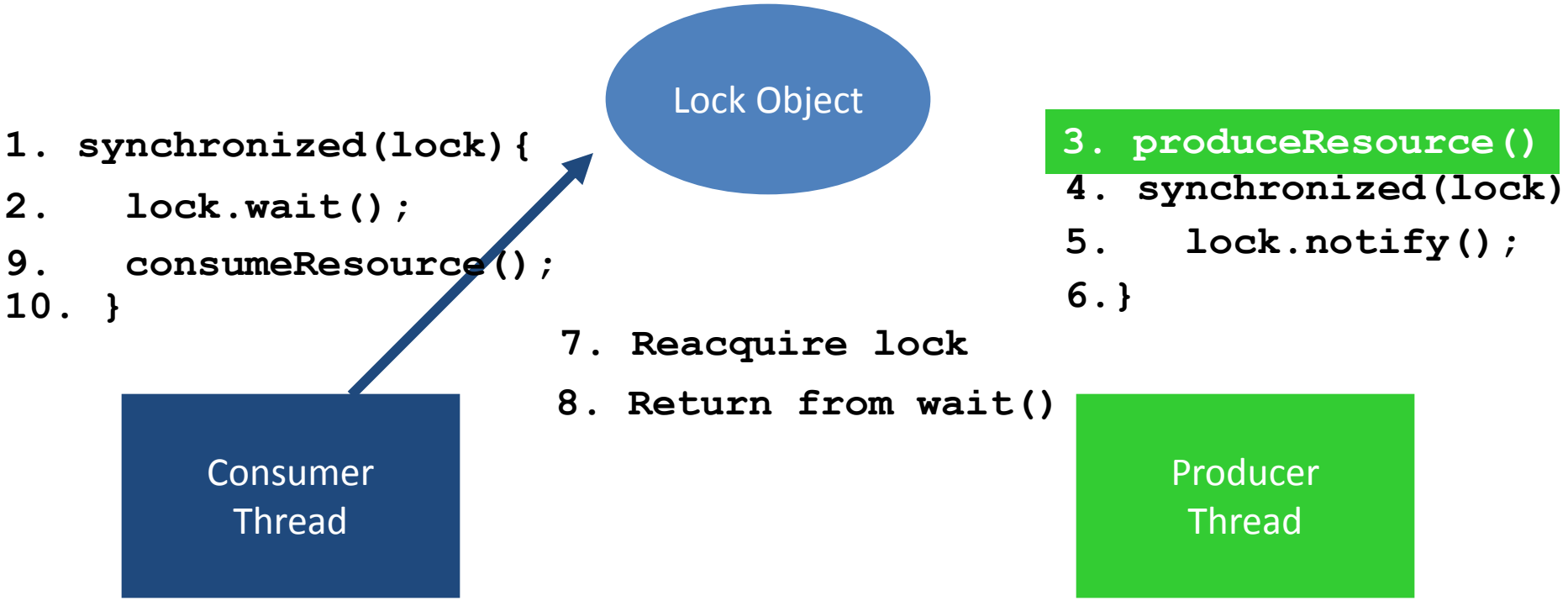
Wait/Notify Sequence



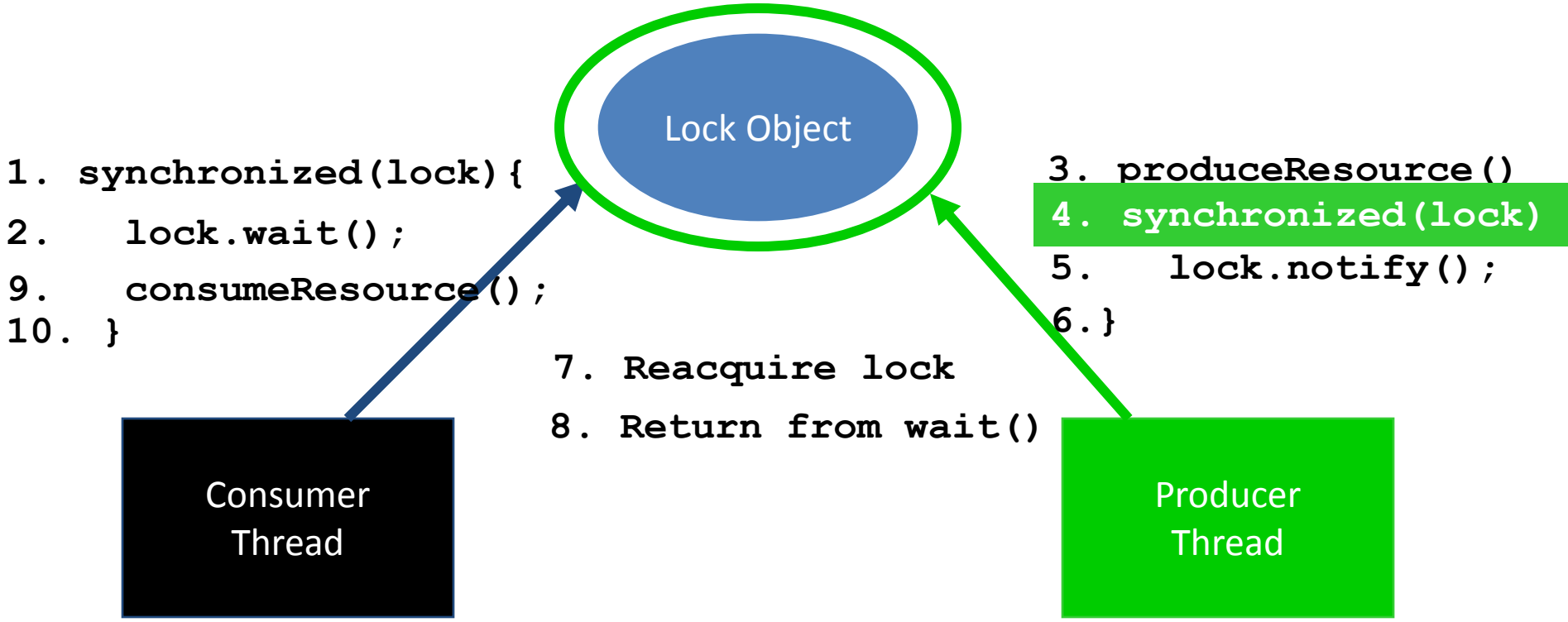
Wait/Notify Sequence



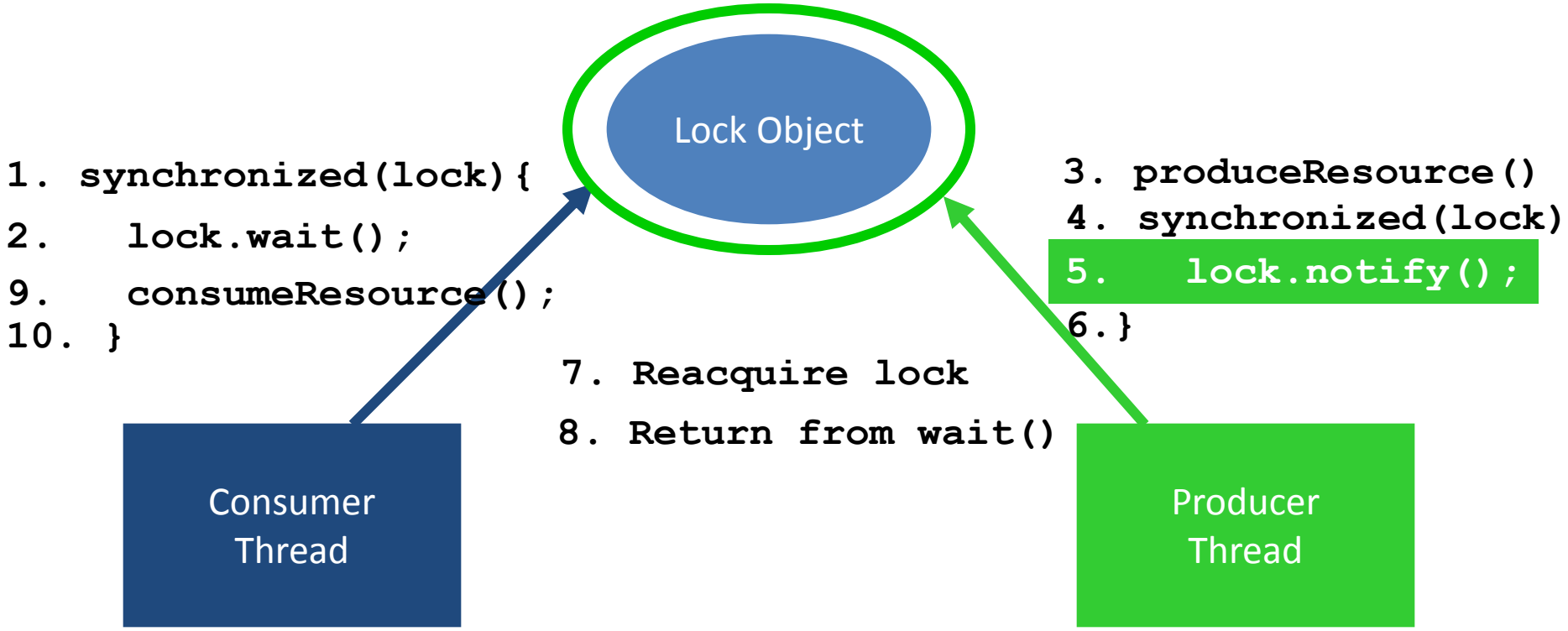
Wait/Notify Sequence



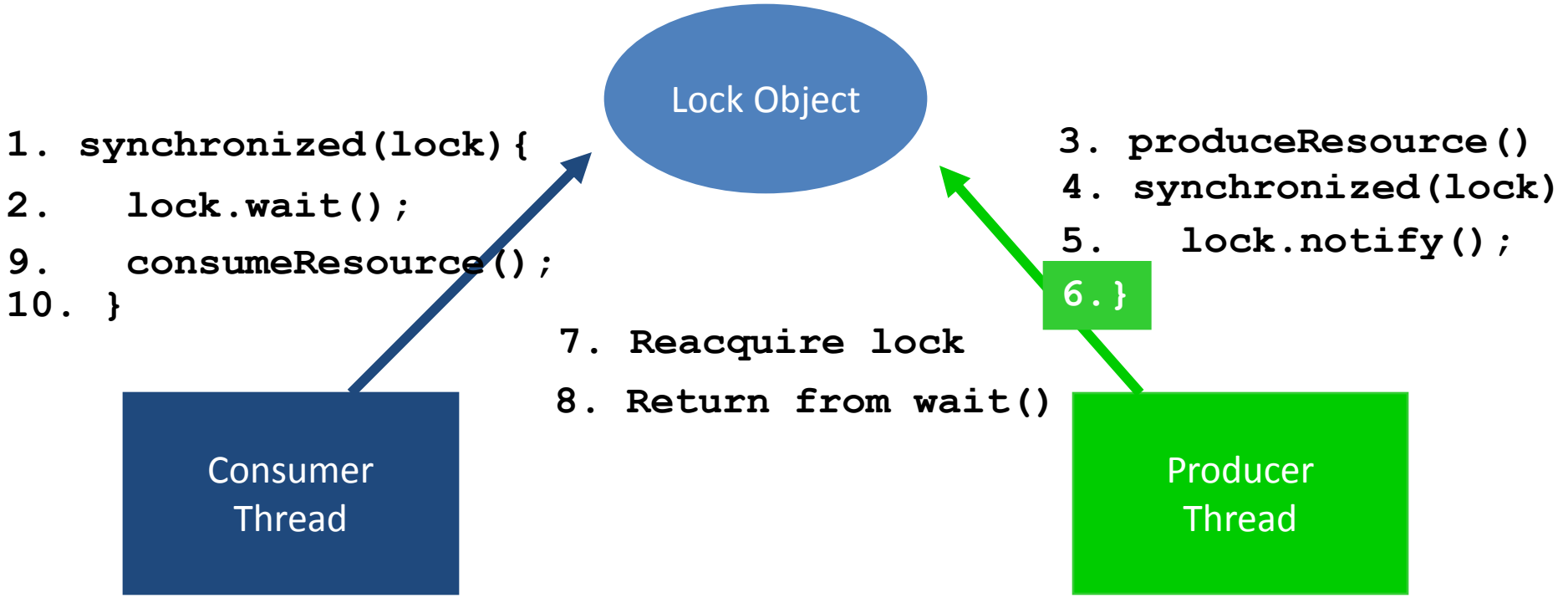
Wait/Notify Sequence



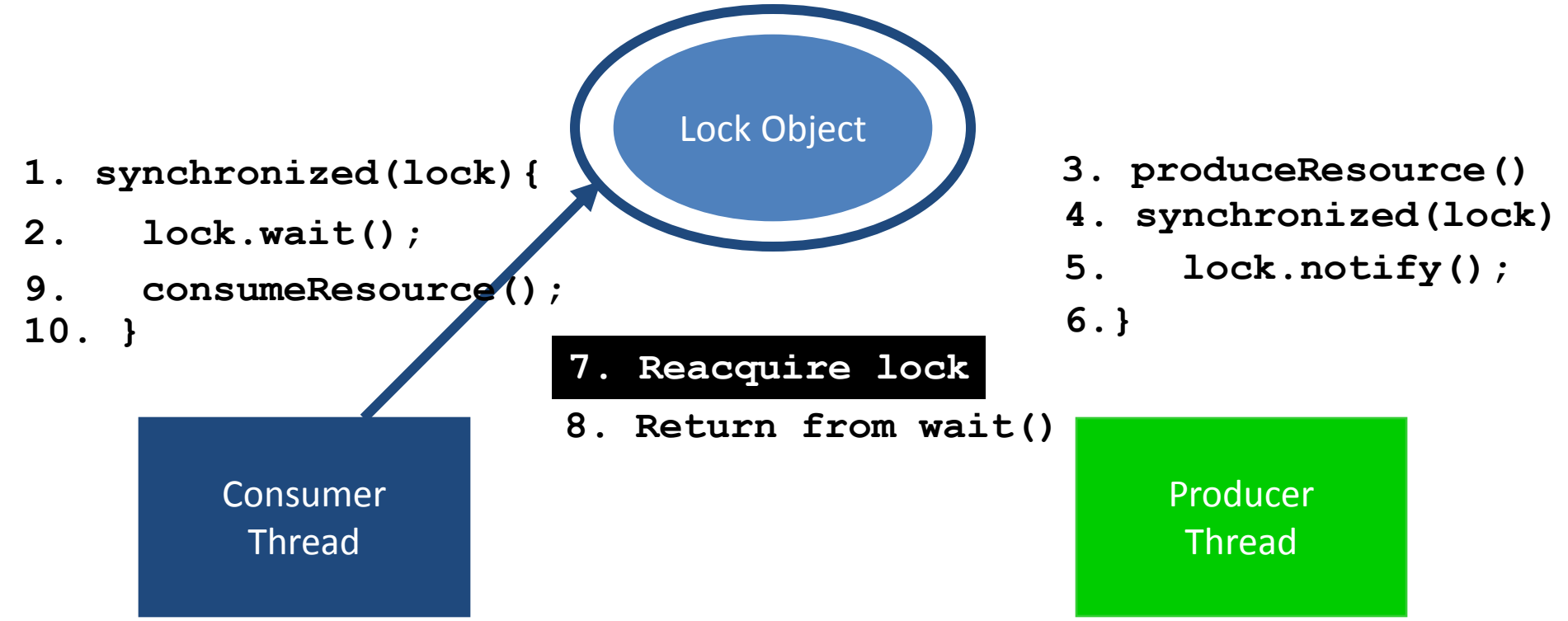
Wait/Notify Sequence



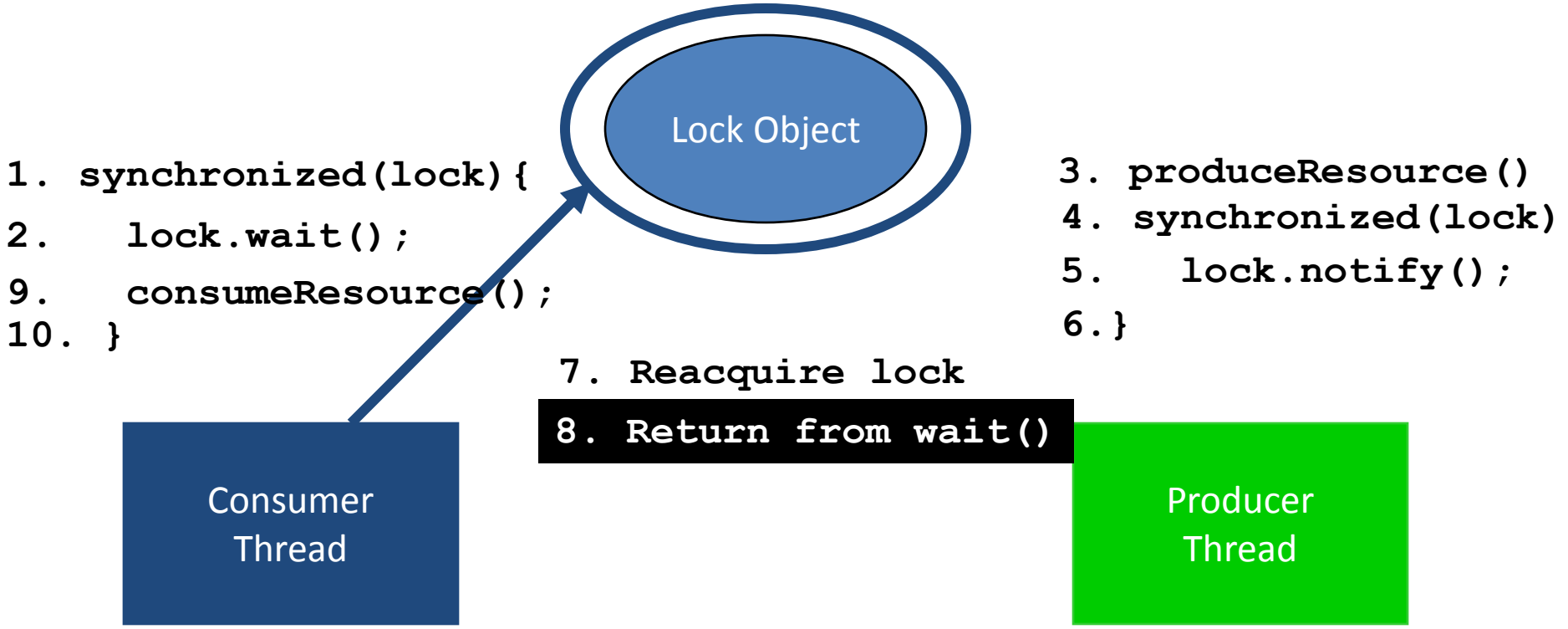
Wait/Notify Sequence



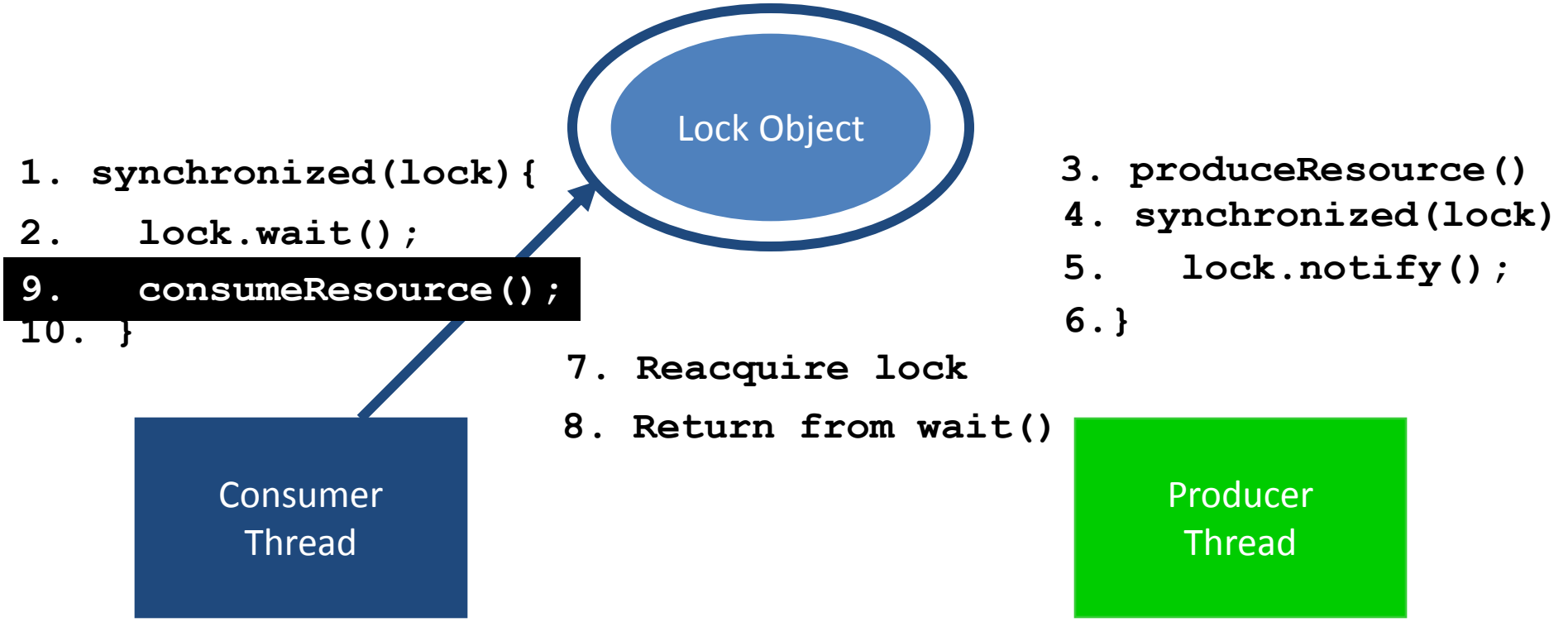
Wait/Notify Sequence



Wait/Notify Sequence



Wait/Notify Sequence



Wait/Notify Sequence


```
1. synchronized(lock){
2.    lock.wait();
9.    consumeResource();
10. }
```

```
3. produceResource()
4. synchronized(lock)
5.    lock.notify();
6.}
```

7. Reacquire lock
8. Return from wait()

Consumer Thread

Producer Thread

Interrupting a Thread

- If any thread is in sleeping or waiting state (i.e. sleep() or wait() is invoked), calling the interrupt() method on the thread, breaks out the sleeping or waiting state throwing InterruptedException.

- If the thread is not in the sleeping or waiting state, calling the interrupt() method performs normal behaviour and doesn't interrupt the thread but sets the interrupt flag to true.

- Let's first see the methods provided by the Thread class for thread interruption

- **The 3 methods provided by the Thread class for interrupting a thread**
  - **public void interrupt()**
  - **public static boolean interrupted()**
  - **public boolean isInterrupted()**

END OF SESSION