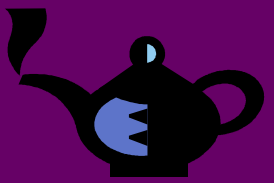
T-Java
Threads



Java Fundamentals 6

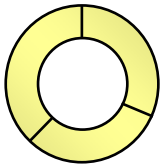
Threads

- Thread States
- A Thread's Life
- Sleep()
- Thread Safe
- Thread Synchronization
- wait() and notify()

Threads

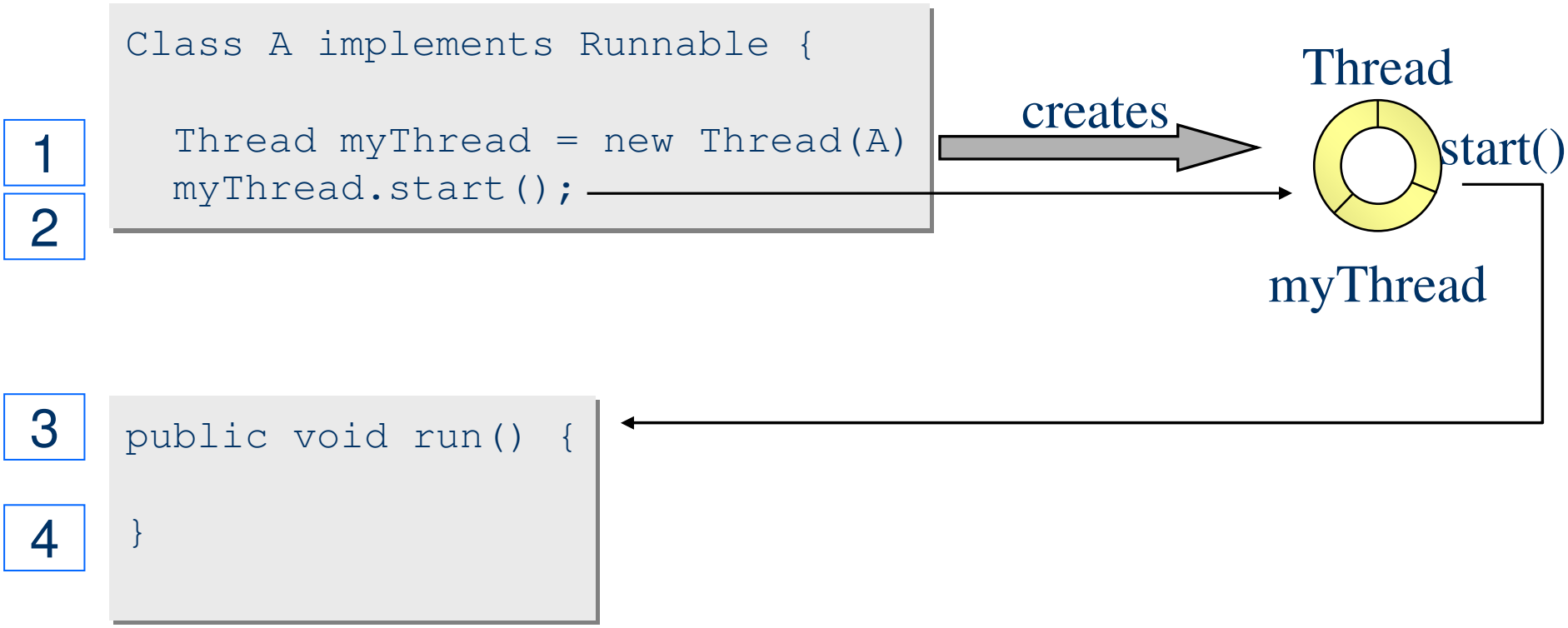
- A *thread of control* (thread) is a flow of control within a program
- Java programs may use multiple threads
- Different threads can access instance variables but not local variables

java.lang.Thread



- A thread object represents a real thread in the Java interpreter
- The thread object is a handle for controlling the real thread
- The thread object start() method executes a run() method in a particular specified object

A Thread's Life



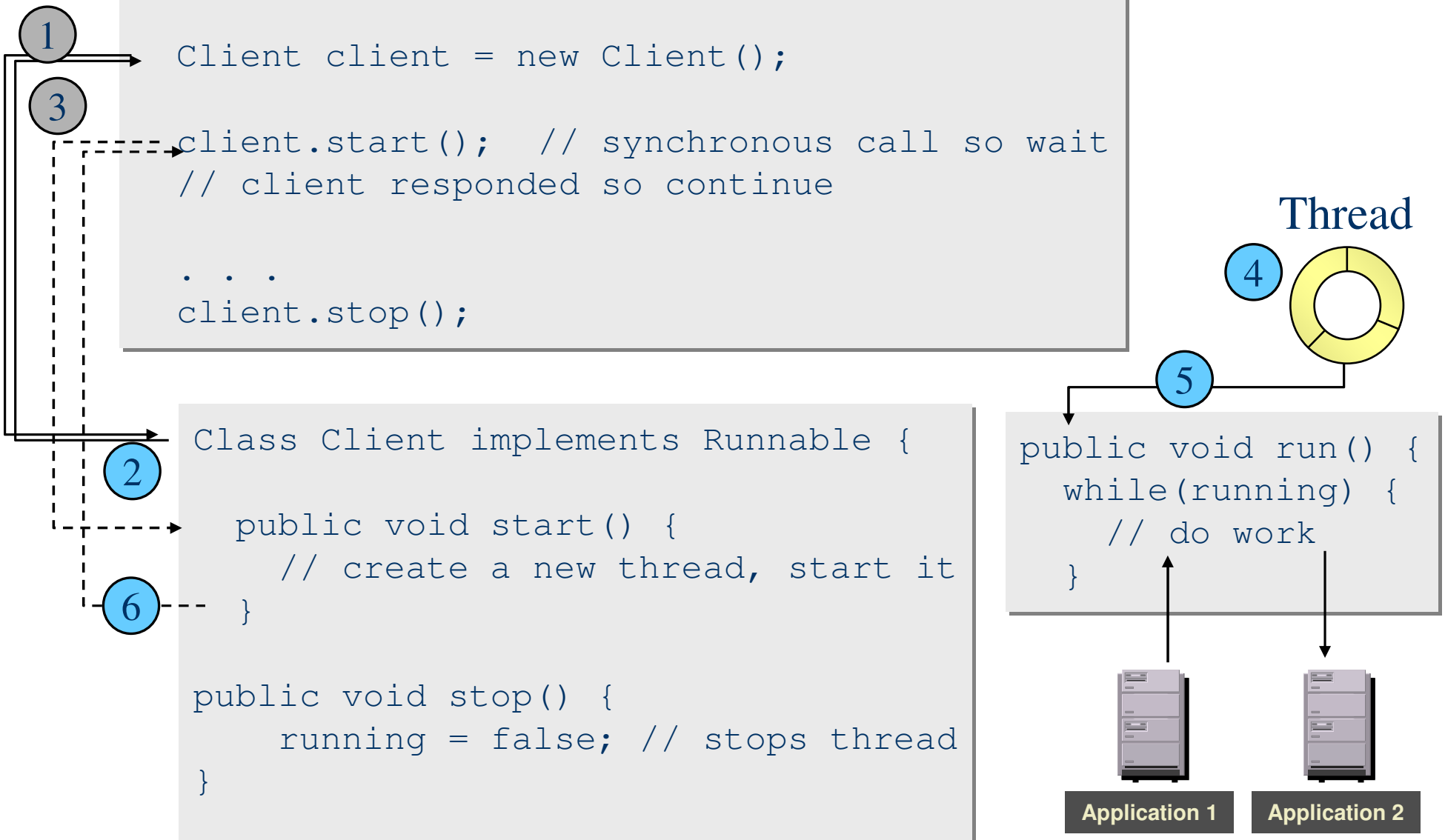
- On return from the run() method, the thread myThread terminates
- If the run() method never returns, the thread lives on even after the application that created it has finished (i.e., main() returns)

Using Multiple Threads

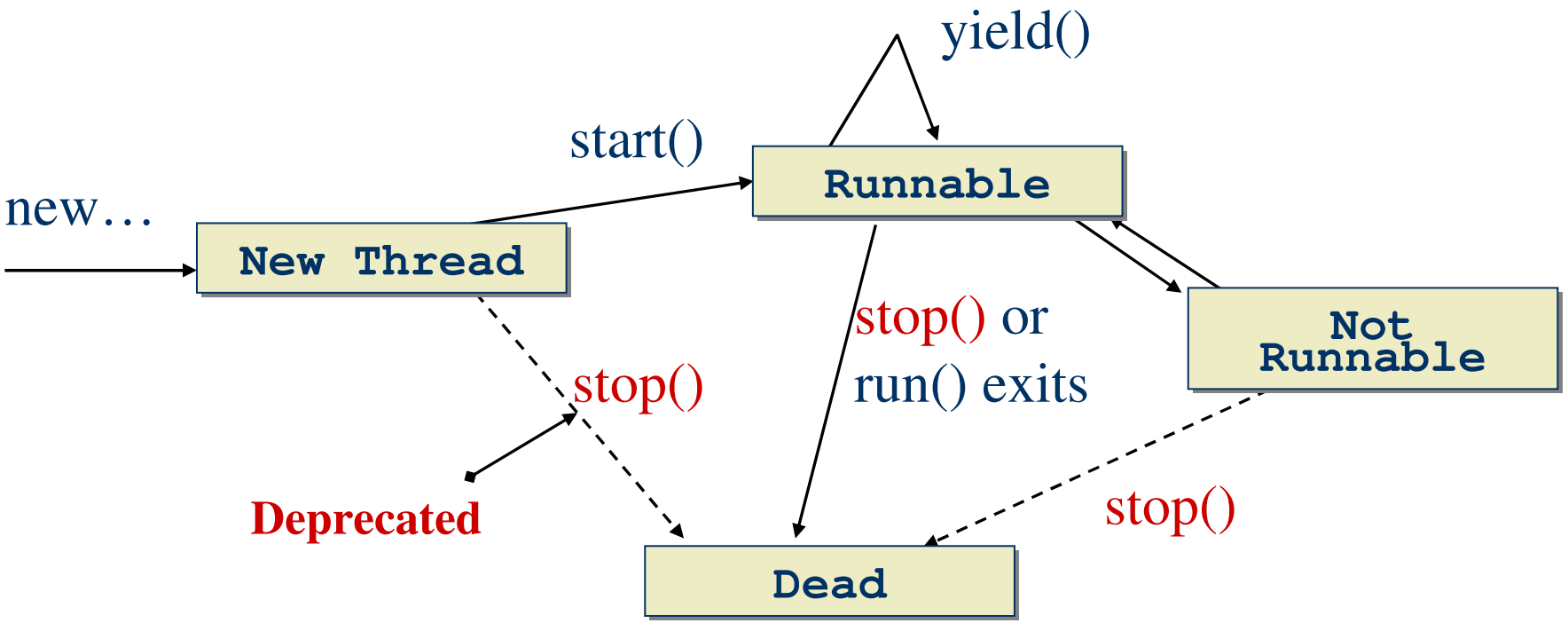
```
Class FlowManager {

  Client client = new Client();

client.start();  // synchronous call so wait
  // client responded so continue

  . . .
  client.stop();
```

```
Class Client implements Runnable {

  public void start() {
    // create a new thread, start it
  }

public void stop() {
    running = false; // stops thread
}
```

```
public void run() {
  while(running) {
    // do work
  }
```



Thread States



- **start()** causes this thread to begin execution; the Java Virtual Machine calls the run() method of this thread.
- **yield()** causes the currently executing thread object to temporarily pause and allow other threads to execute

The sleep() method

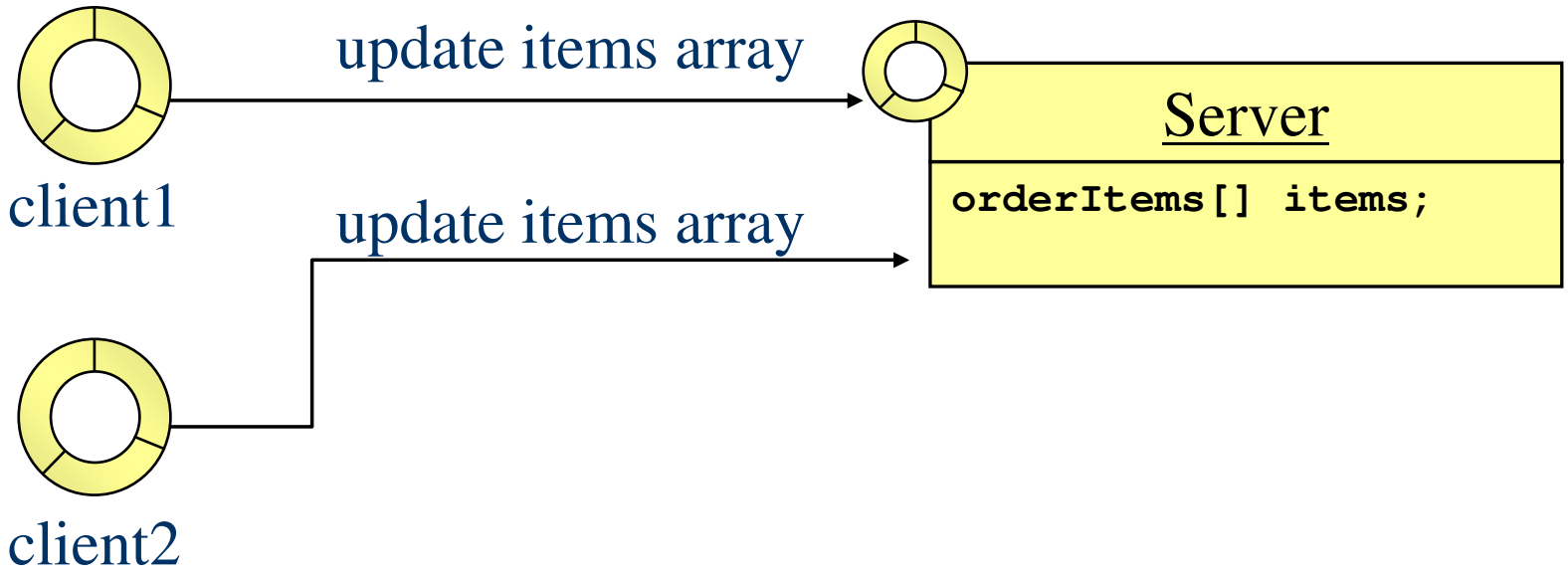
- The sleep() method can force a thread to be idle for a period of time

```
long delay = 10000; // milliseconds

try {
  // sleep() is a static method
  Thread.sleep(delay);
}
catch(InterruptedException e){
  // someone woke us up prematurely
}
```

- Some other thread can throw the InterruptedException to cause the sleeping thread to resume immediately

Thread Safe

- Thread safe programming is necessary when data can be modified by more than one thread at a time
- If multiple clients can change shared server data, a thread-safe design is needed



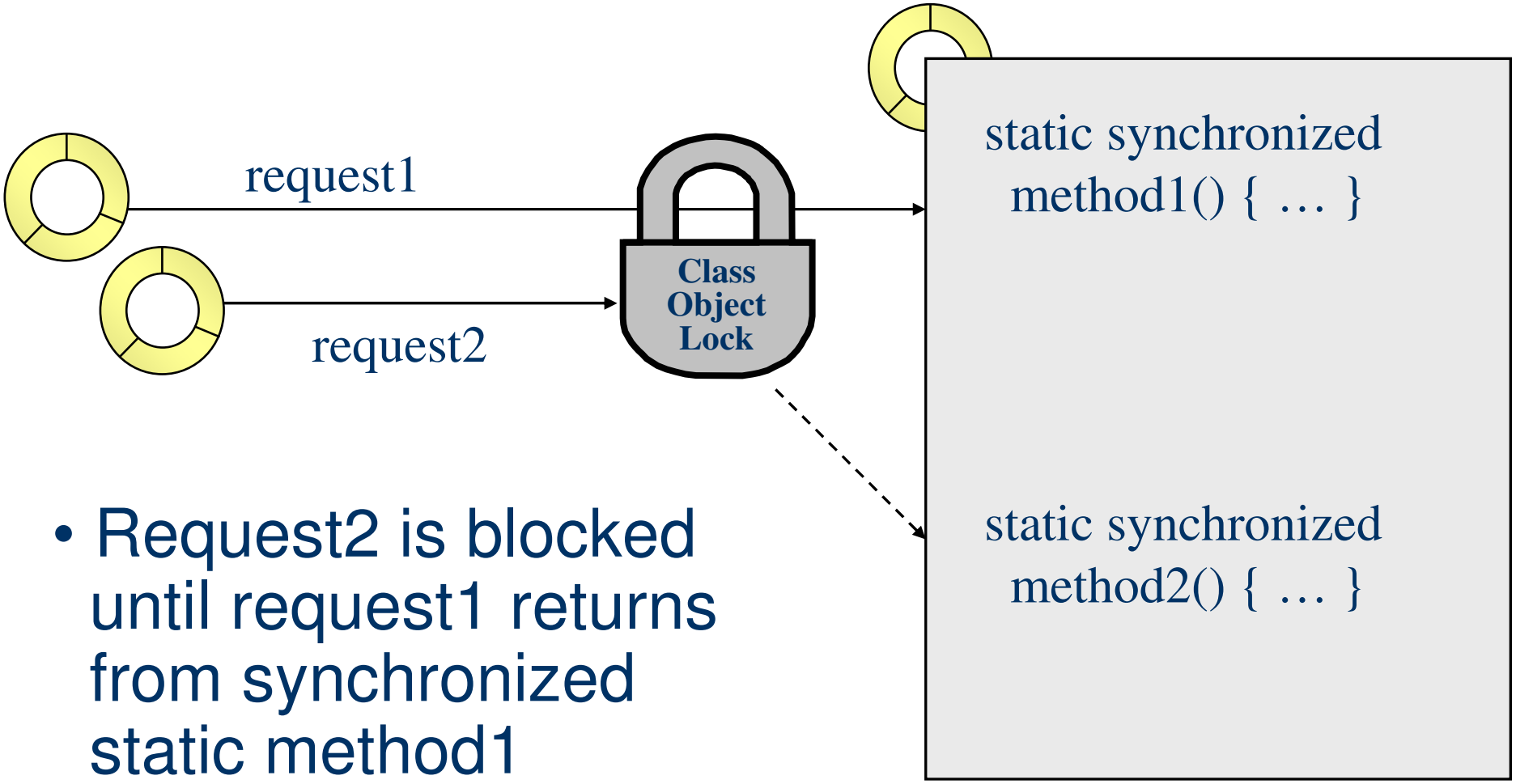
Serializing access to Methods

- In Java, every class and every instance of a class has a lock associated with it
- The *synchronized* keyword identifies places where a thread must acquire the lock before proceeding

```
Class A {

 static synchronized void validate() {
 }
 static synchronized void restore() {
 }
 synchronized void update() {
 }
 synchronized void replace() {
 }
```

Thread Synchronization

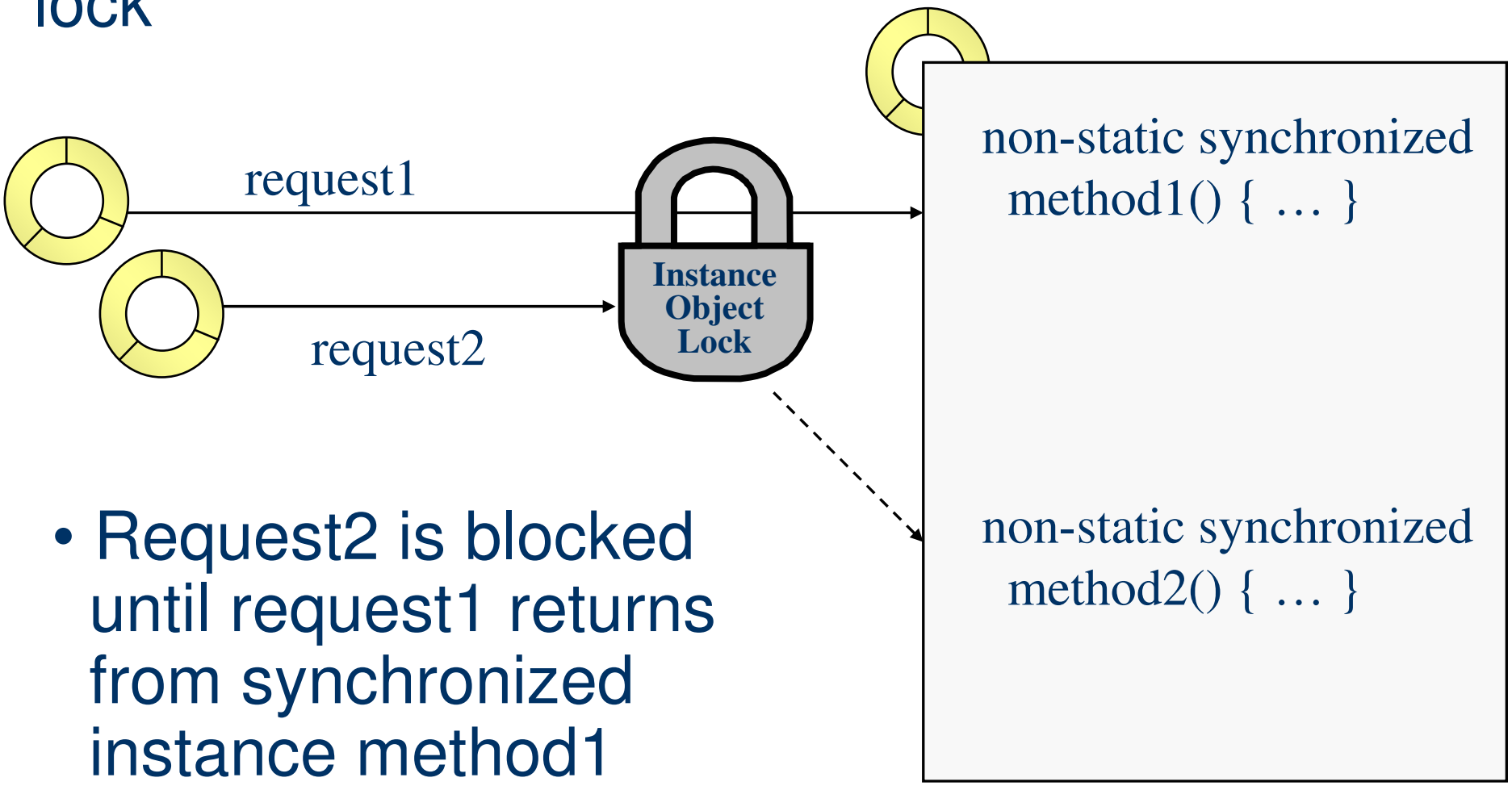
- All **static** synchronized methods in a class use the same class object lock



- Request2 is blocked until request1 returns from synchronized static method1

Thread Synchronization

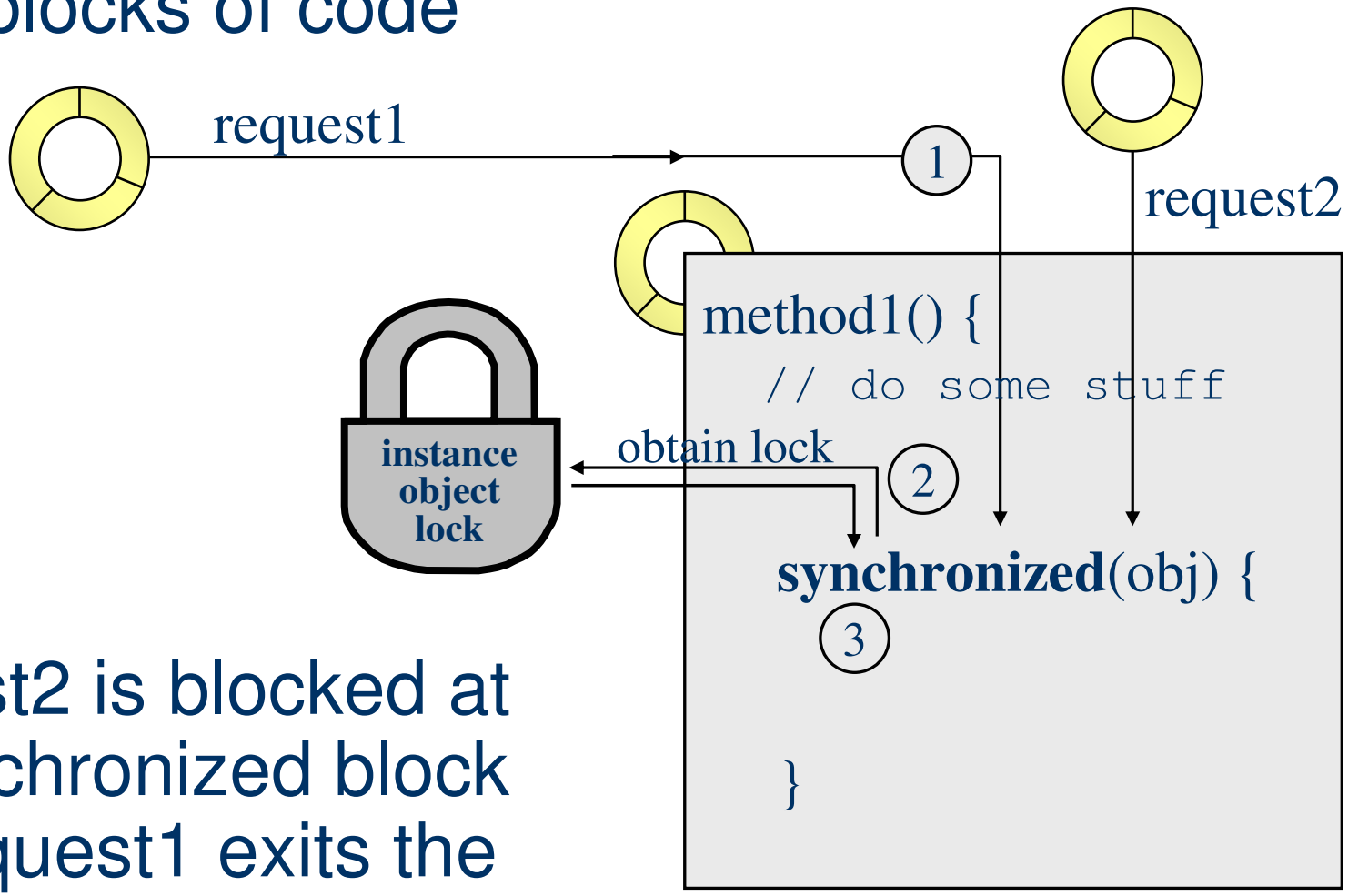
- For each object of a class, all synchronized instance methods use the same instance object lock



- Request2 is blocked until request1 returns from synchronized instance method1

Thread Synchronization

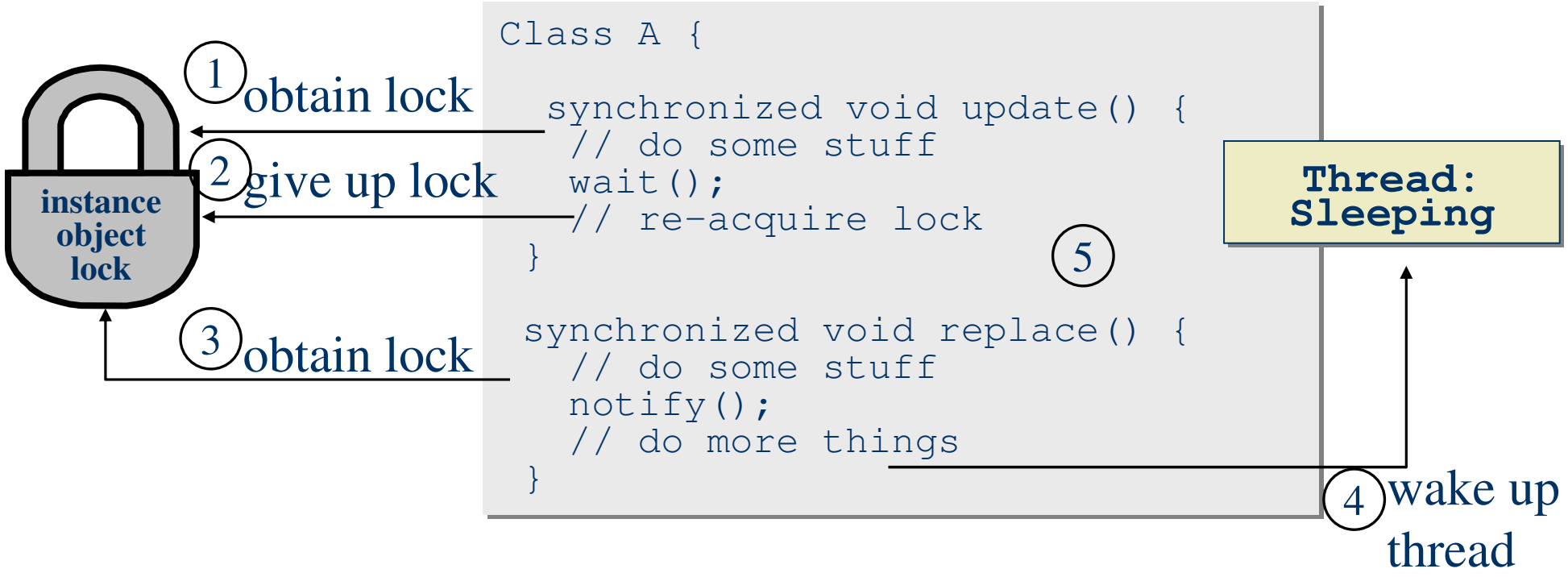
- The *synchronized* keyword can be used to guard arbitrary blocks of code



- Request2 is blocked at the synchronized block until request1 exits the synchronized block

wait() and notify()

- The Object class wait() and notify() methods enable a thread to give up its lock and to wait for another thread to give it back before proceeding

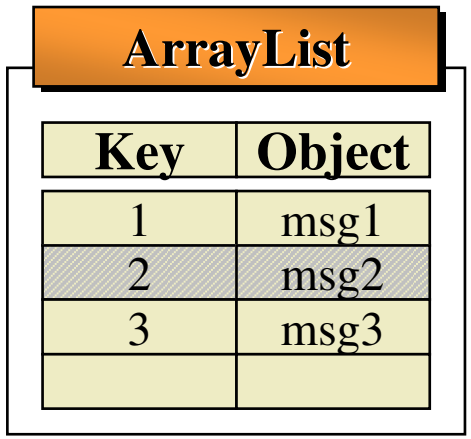


wait() and notify() Example

- Synchronized access to ArrayList elements

```
Class Producer {
  synchronized void putMessage() {
     // do some stuff

     while(noRoomForMessage) {
     wait(); // give up lock
     }
     // re-acquire lock
     messages.add(msg1);
     notify();
  }
```

```
  synchronized void getMessage() {
    // do some stuff
    while(noMessage){
    wait(); // give up lock
    }
    // re-acquire lock
    messages.remove(msg1);
    notify();
    // do more things
  }
```



Navigate to T-Java\Exercises\exp65a…

Summary

- A thread is a flow of control within a program
- You can create threads
- Threads execute the run() method
- *synchronized* keyword identifies places where a thread must acquire the lock before proceeding

Key Terms

Thread
Thread's life
Thread states
sleep
"thread safe"
synchronization
synchronized keyword
wait
notify

- Object locks control synchronization
- Wait() releases the object lock