Java Multithreading

Presented by

Sagar
Java Consultant
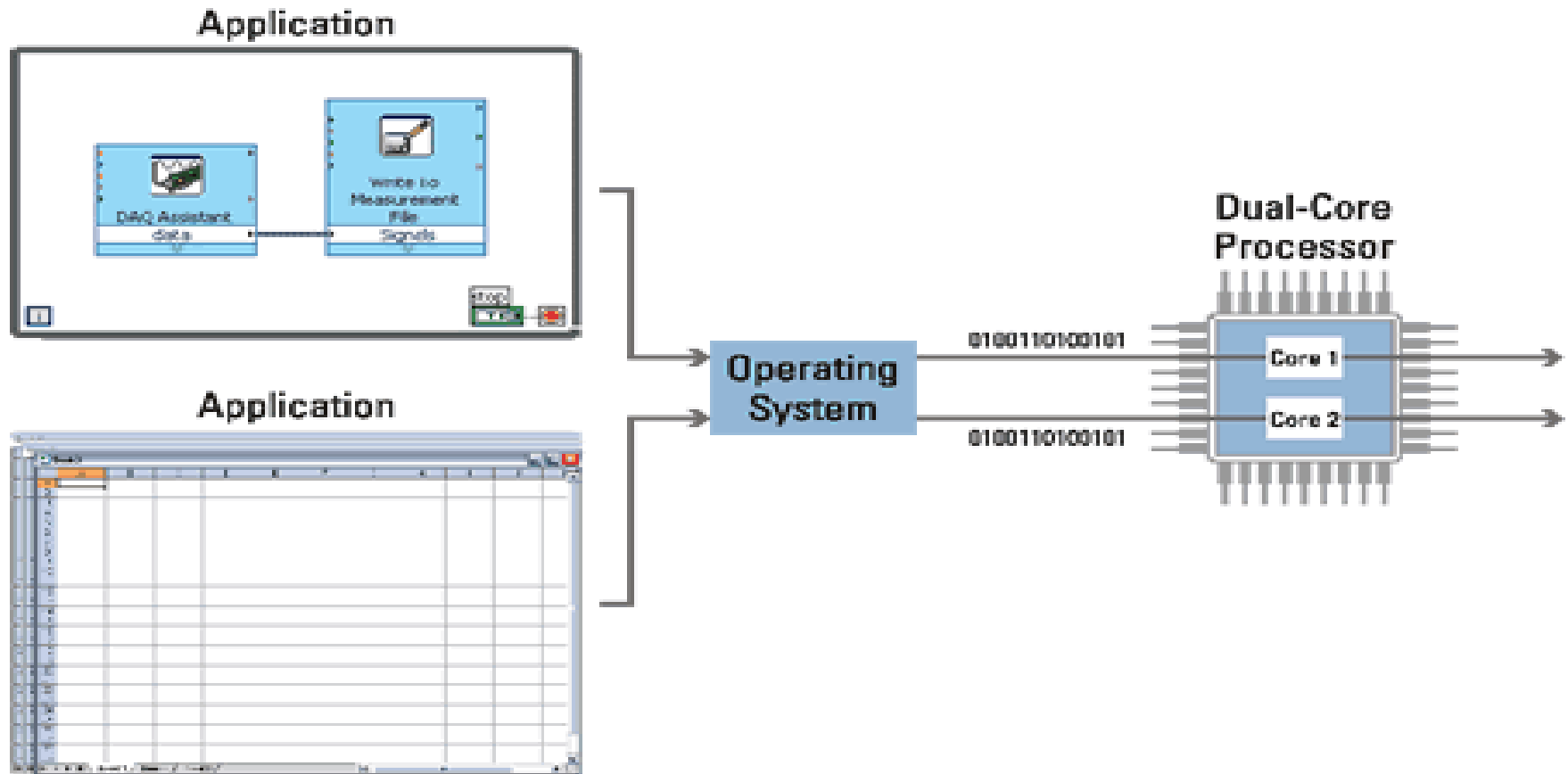
Multitasking

Executing multiple programs simultaneously.
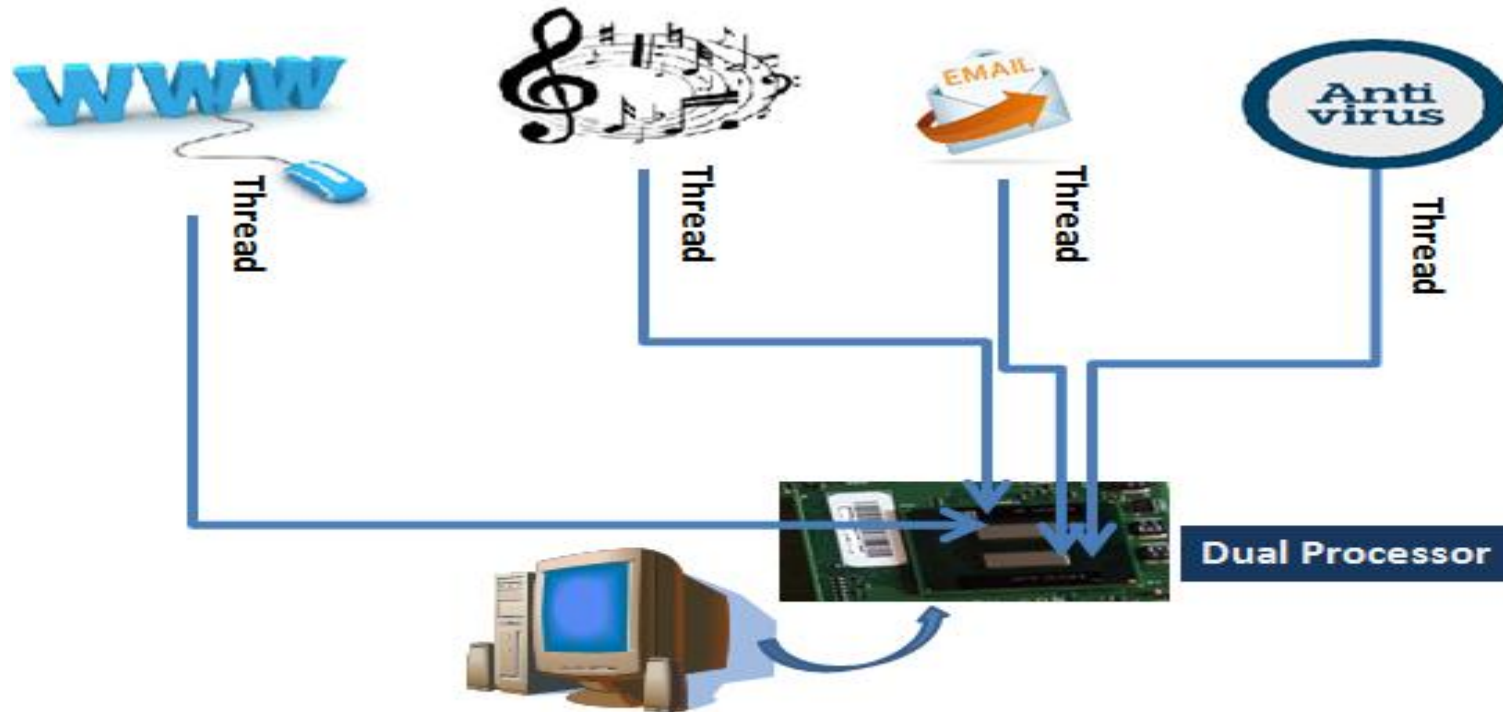
1. Process based   **2. Thread based**

Multitasking Applications

Multitasking Applications

Multithreading On a Dual Processor Desktop System

Note: Multithreading can also be possible on single processor systems

Multitasking

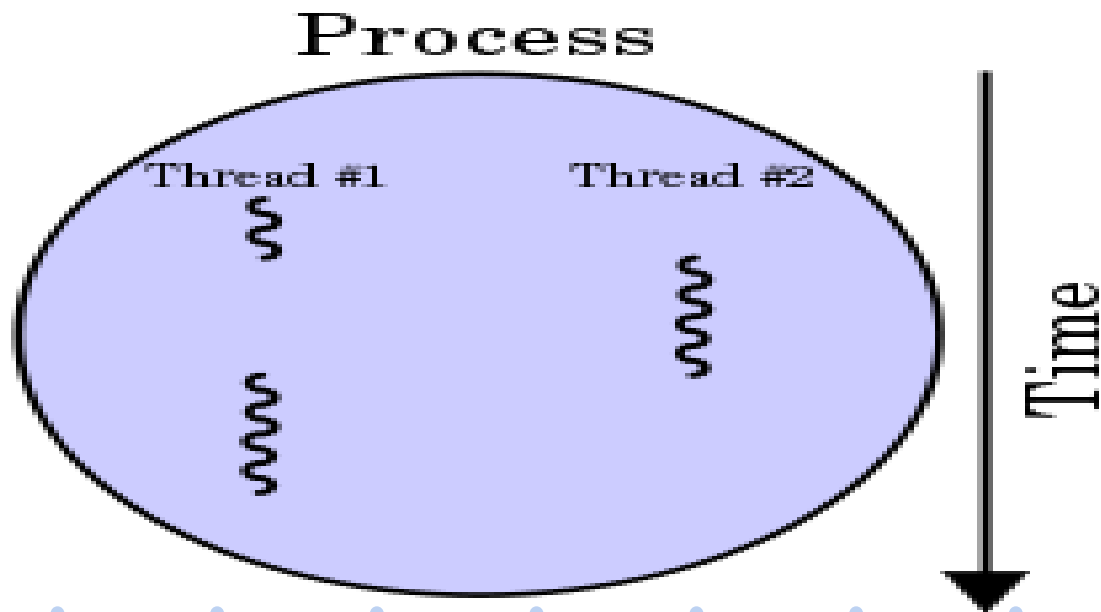
1   2   3   5

Multi Threading

Thread

The smallest unit of processing

Ex: Few lines of code – computing 3 account's interest

Multi Tasking vs Multi Threading

Multitasking and Multithreading

- **Multitasking:**
  - refers to a computer's ability to perform multiple jobs concurrently
  - more than one program are running concurrently, e.g., UNIX
- **Multithreading:**
  - A thread is a single sequence of execution within a program
  - refers to multiple threads of control within a single program
  - each program can run multiple threads of control within it, e.g., Web Browser
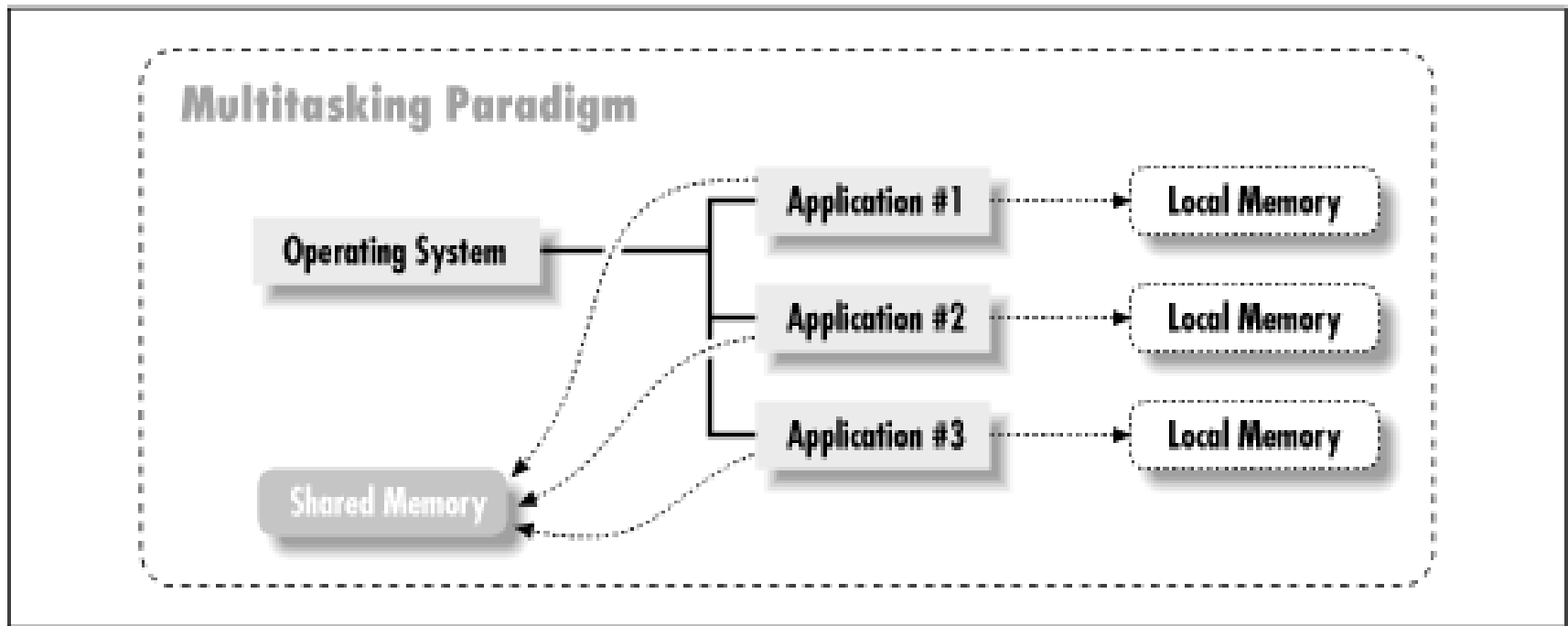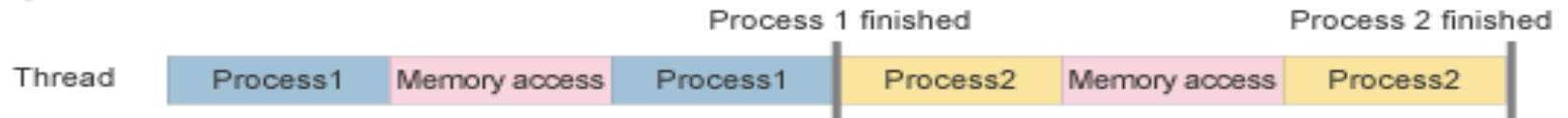
2

Multitasking Environment

Data within processes is seperated; separate stack for local variables and data area for objects
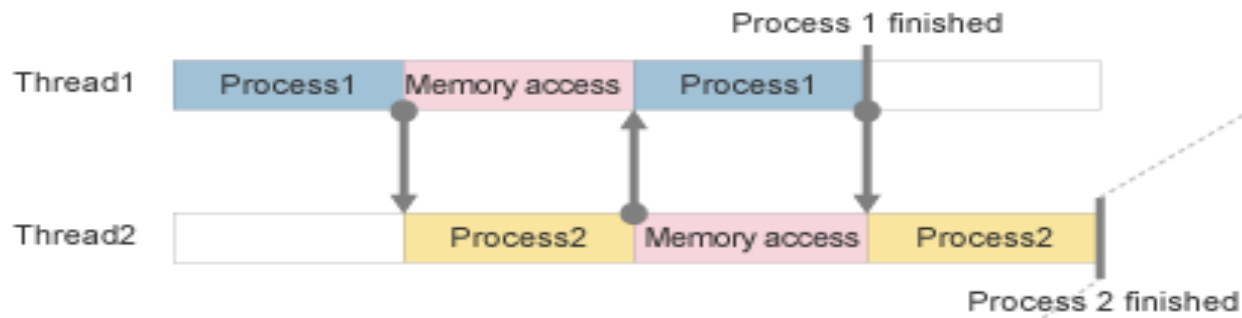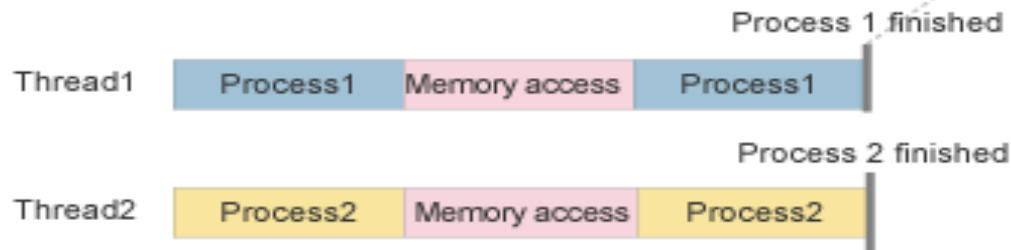

Multitasking Paradigm

Operating System

Application #1 — Local Memory

Application #2 — Local Memory

Application #3 — Local Memory

Shared Memory

Multitasking with Threads

Single thread timeline

Process 1 finished

Process 2 finished

| Thread | Process1 | Memory access | Process1 | Process2 | Memory access | Process2 |

Multi thread timeline (SPARC64™ VI) : VMT

Process 1 finished

| Thread1 | Process1 | Memory access | Process1 | |

| Thread2 | | Process2 | Memory access | Process2 |

Process 2 finished

Multi thread timeline (SPARC64™ VII) : SMT

Process 1 finished

| Thread1 | Process1 | Memory access | Process1 |

Process 2 finished

| Thread2 | Process2 | Memory access | Process2 |

Multitasking with Threads

Thread - similar to process

Multiple thread running within a single instance of JVM – similar to multiple processes within an OS
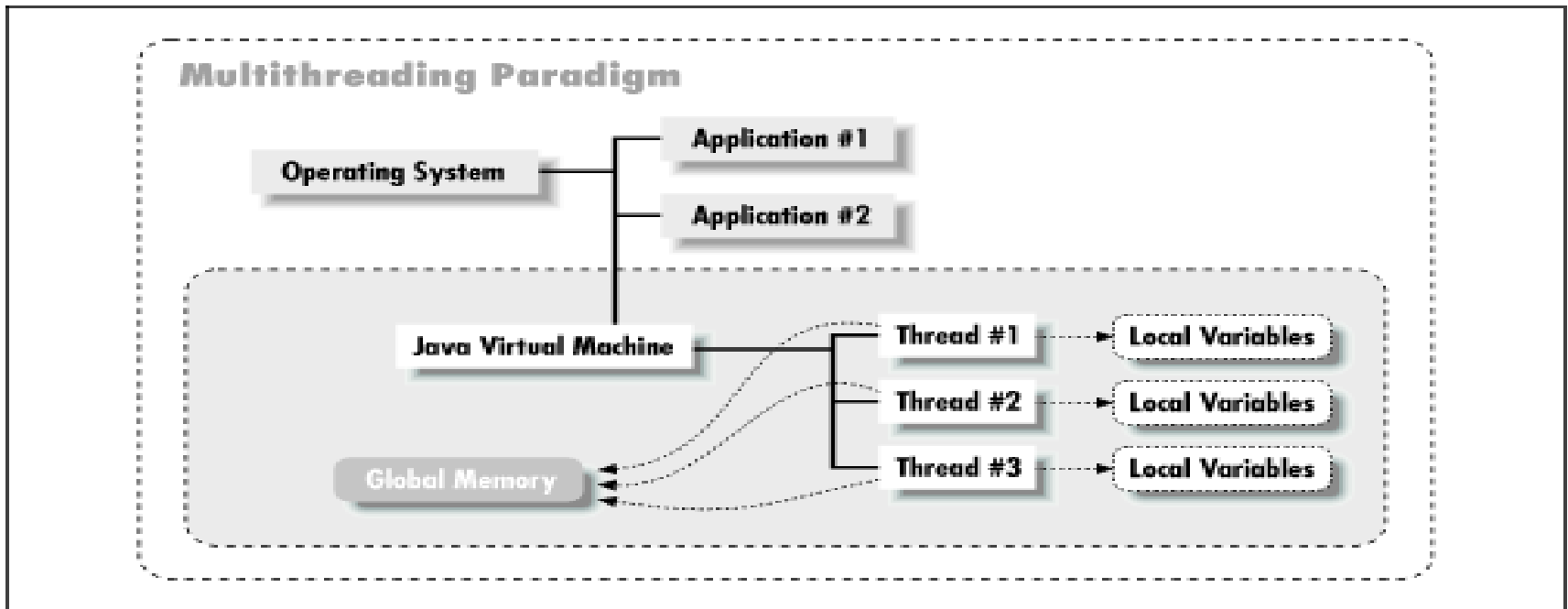
Thread Life Cycle

Implementing Threads

Threads are developed by :

      First, by subclassingthe **Thread** class

      Second, by implementing the **Runnable** interface

      Override the public void run() method

      Place the functionality in the run()method

How a thread starts?

public void start()
☐Creates a new thread. The thread will be in runnablestate

public void run()
☐The new thread begins its life

Thread Life Cycle

```java
public class BytePrinter extends Thread {
    public void run() {
        for (intbnum= -128; bnum< 128; bnum++) {
            System.out.println(bnum);
        }
    }
}
```

Multithreading Issues

- A single-threaded program performs one task at a time

- With multithreading ,two or more threads may coexist and possibly try to use the same resource simultaneously.

- Colliding over a resource must be prevented

- Because many threads may try to access the same resource, which may yield undesirable result
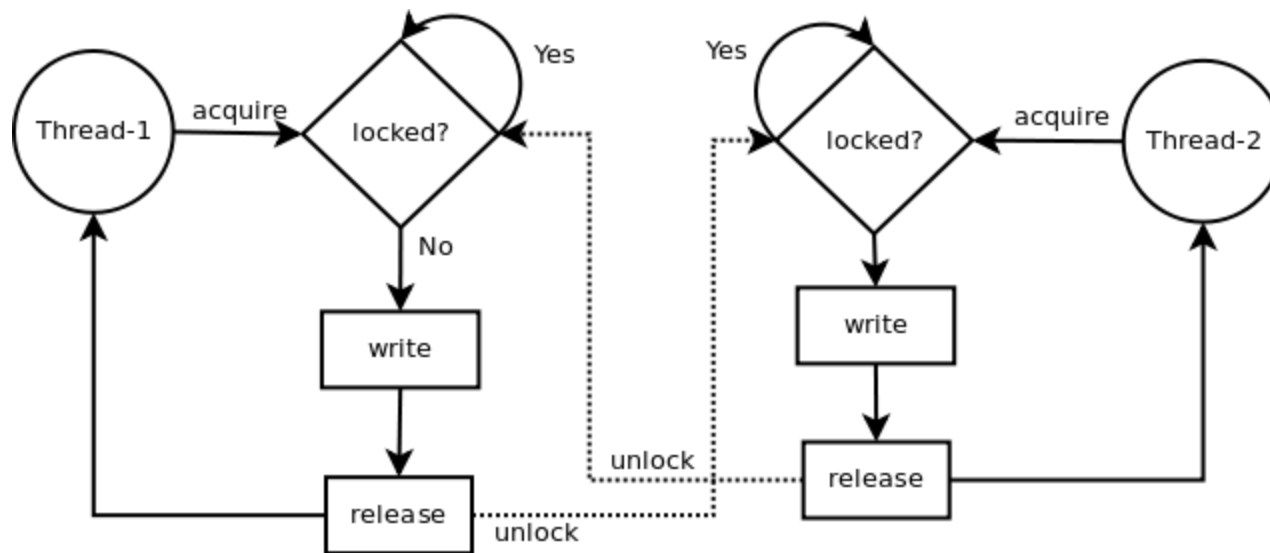
Solution - Synchronization

Thread Synchronization

- **Placing a lock on a resource when one thread is utilizing it.**

- **The first thread that accesses a resource locks it**
- **Other threads can not access that resource until it is unlocked.**

Thread Synchronization

- Methods can be declared as synchronized.

- Only one thread at a time can call a synchronized
- method of a particular object.

- Every thread object has a lock / monitor.

- The object that has synchronizedmethod, is locked

- No other synchronized methods of that object is called until
the first thread completes and releases the lock

Thread Synchronization Example

```java
public class Shared // used by UserThread class methods
        public synchronized int get() {
                forI(int taskNo=1;taskNo<=5;taskNo++) {
                        // code goes here
                }
        }


        public synchronized  void put() {
                for(int taskNo=1;taskNo<=5;taskNo++) {
                        // code goes here
                }
        }

}
```

Thread Synchronization Example

```java
public class MainSynch {
        public static void  main(String args[]) {
                UserThread ut1 = new UserThread();
                UserThread ut2 = new UserThread();
                ut1.start();
                ut2.start();
        }
}
```

Interthread Communication

Allowing "synchronized threads" to communicate with each other. implemented by following methods of Object class:

wait() – Makes the thread to halt until it is notified

notify() - Wakes up a single thread that is waiting on this
**   object**

notifyAll() - Wakes up all threads that are waiting on this
**   object**