Smith College
Computer Science

# Multithreading in Java

CSC352 - Week #3

Dominique Thiébaut
dthiebaut@smith.edu

# Reference

- http://www.science.smith.edu/dftwiki/index.php/CSC352_Synchronization_and_Java_Threads

# MultiThreading in Java



- Problems when sharing data
- Critical sections
- Semaphores
- Locks
- Synchronized sections
- Definitions
- Lab/Exercises

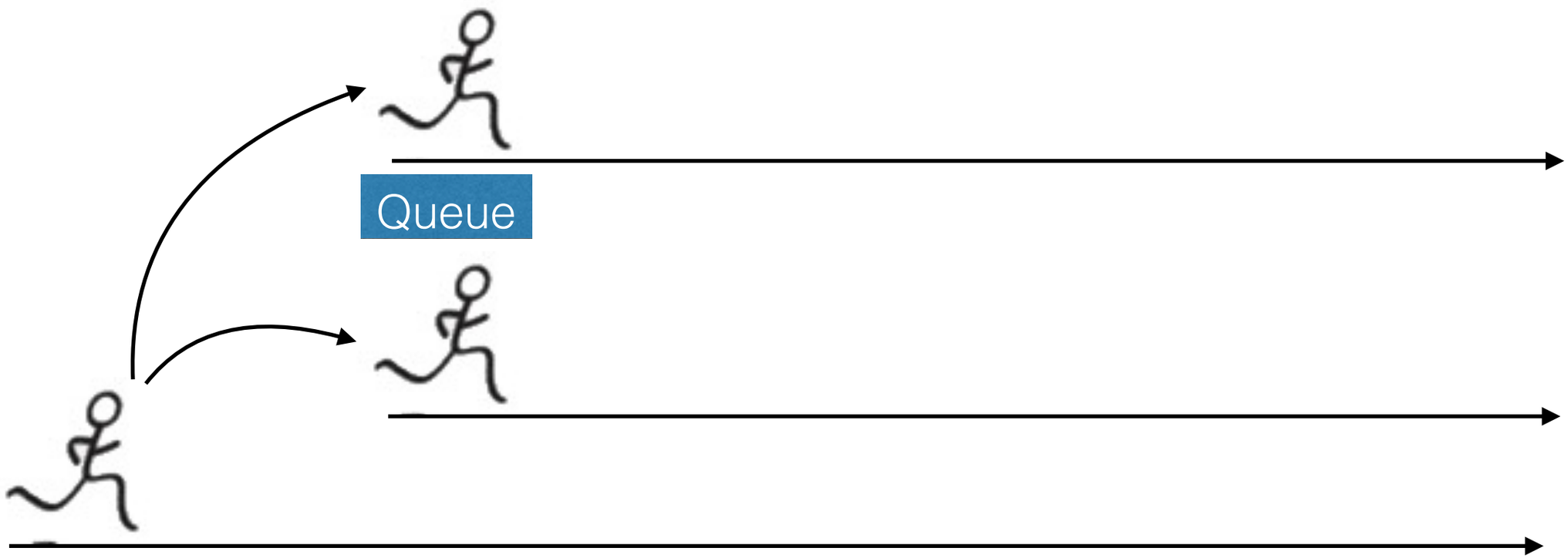
https://javantura.com/java-logo-background-png/

# Quick Review

## Manager-Worker Paradigm

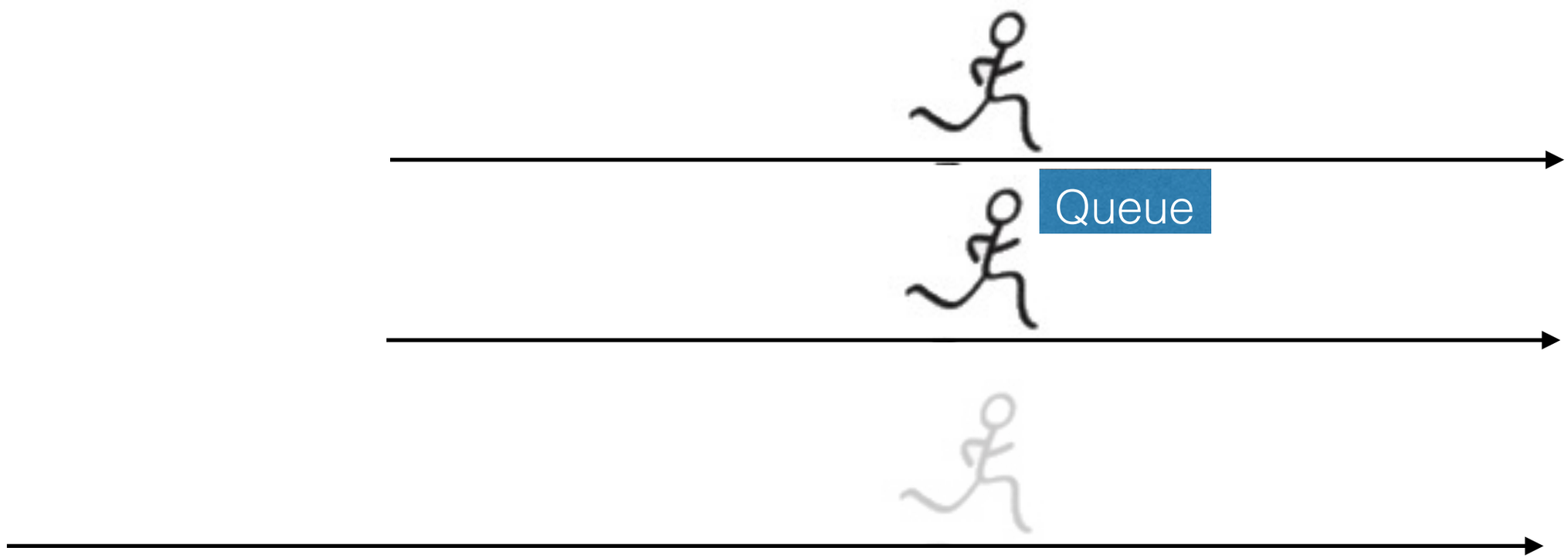
# Quick Review

## Manager-Worker Paradigm



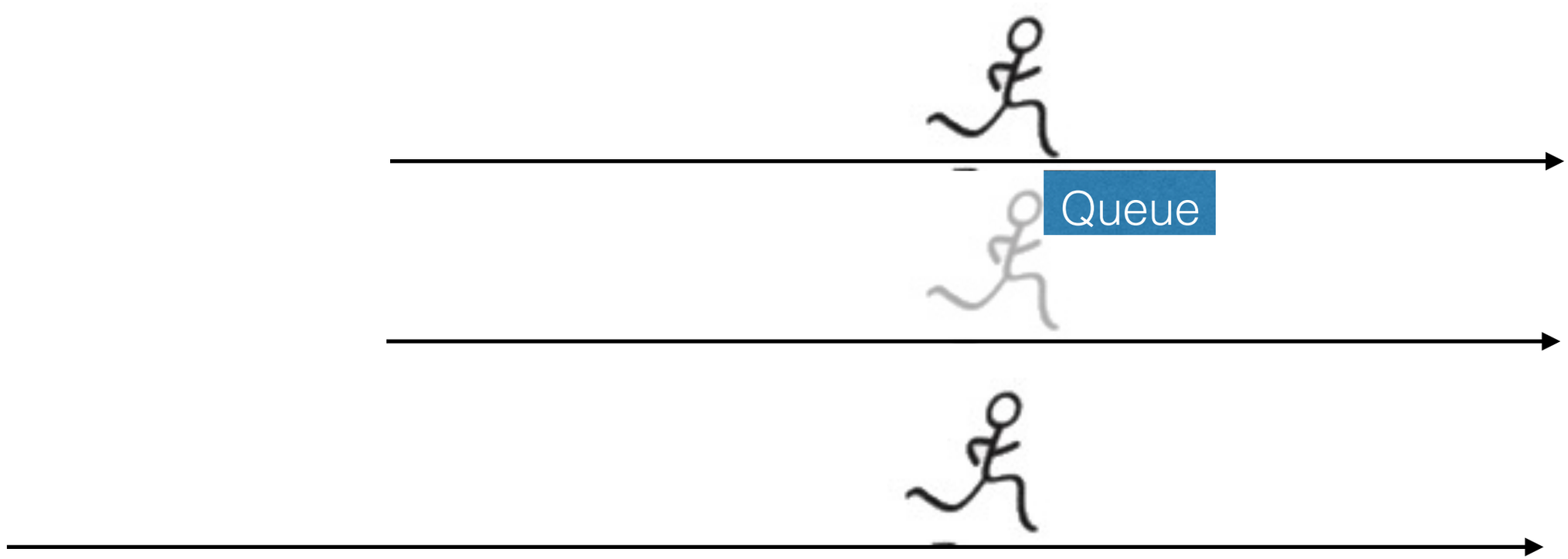
# Quick Review

## Manager-Worker Paradigm



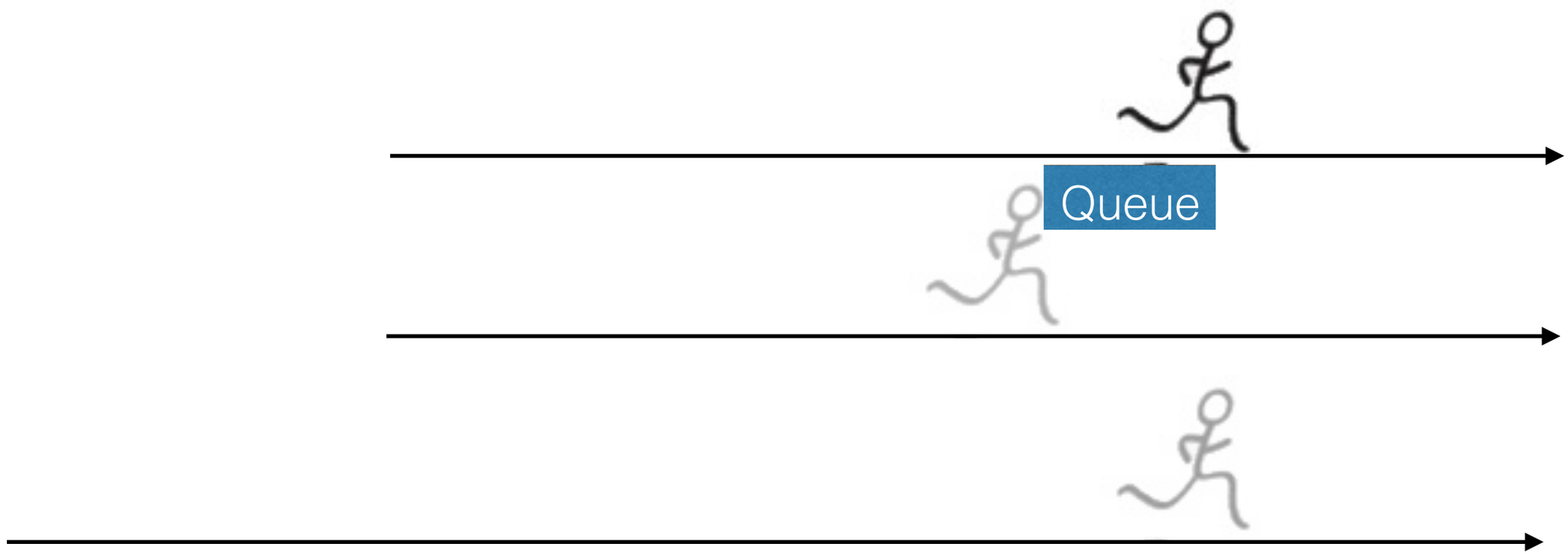
# Quick Review

## Manager-Worker Paradigm



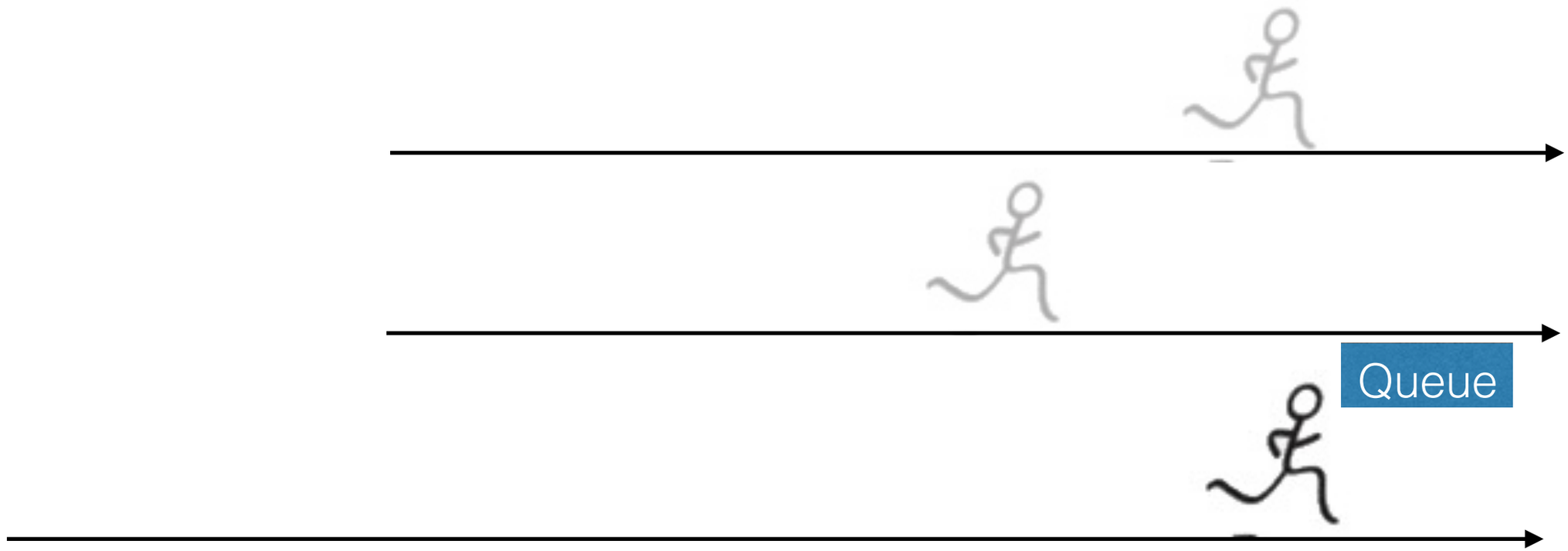
# Quick Review

## Manager-Worker Paradigm



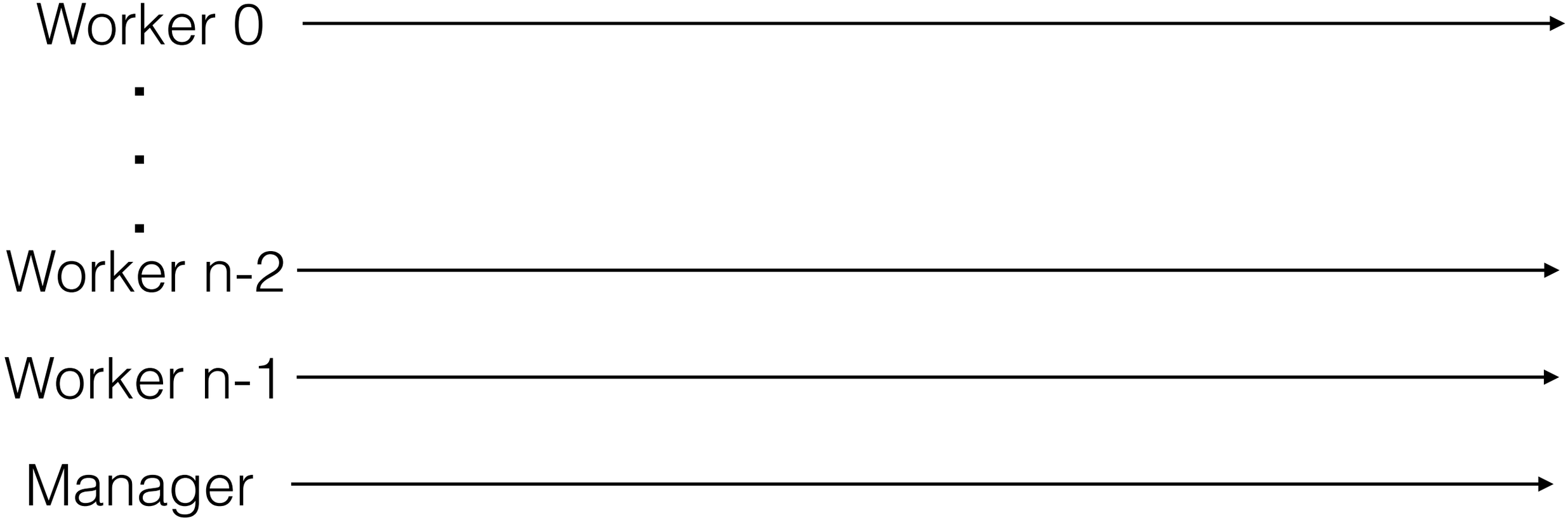
# Quick Review

## Manager-Worker Paradigm



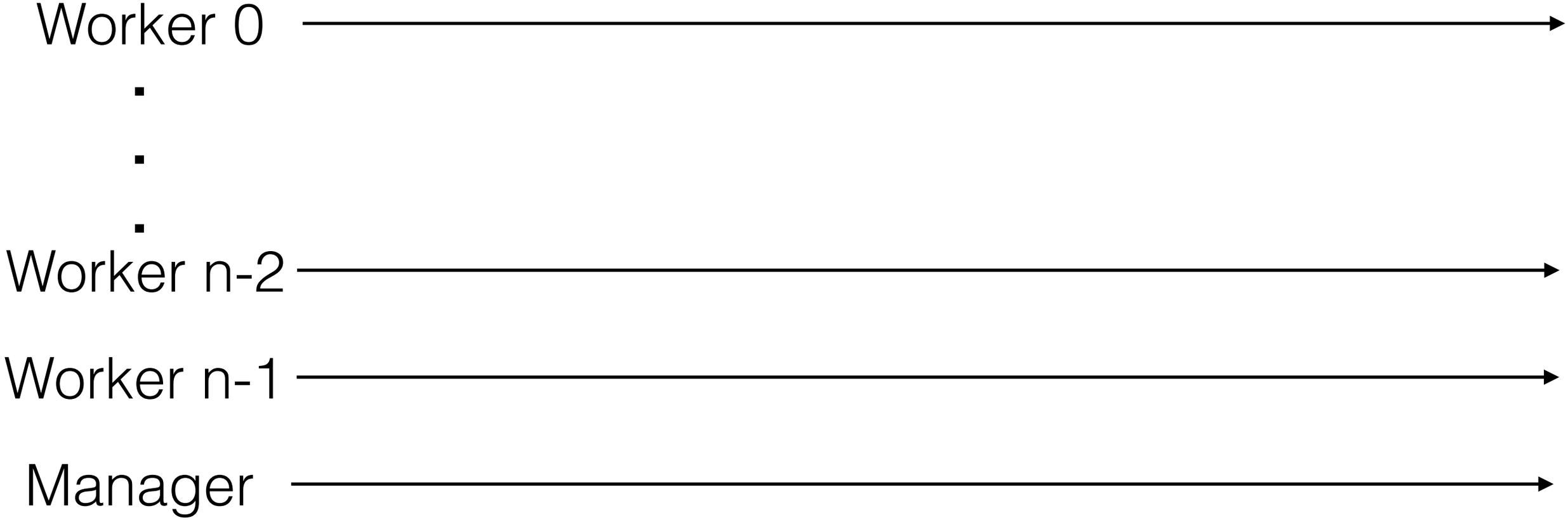
Different Communication Patters

# Quick Review (2)



Different Communication Patters

# Quick Review (2)



# Threads in Java

```
class MyThread extends Thread{
    public void run(){
        System.out.println( "thread is running..." );

        try {
            Thread.sleep( 1000 ); // 1 sec
        }
        catch (InterruptedException ie) {
            ie.printStackTrace();
        }

        System.out.println( "thread is done..." );
    }
}
```


```
public class SimpleThreadDemo {
    public static void main(String args[]){
        // declare a new thread
        MyThread t1=new MyThread();

        // start a new thread
        t1.start();

        // wait for the thread to be done
        try {
            t1.join();
        }
        catch (InterruptedException ie) {
            ie.printStackTrace();
        }

        // Exit
        System.out.println( "Done!" );

    }
}
```


This time with arguments

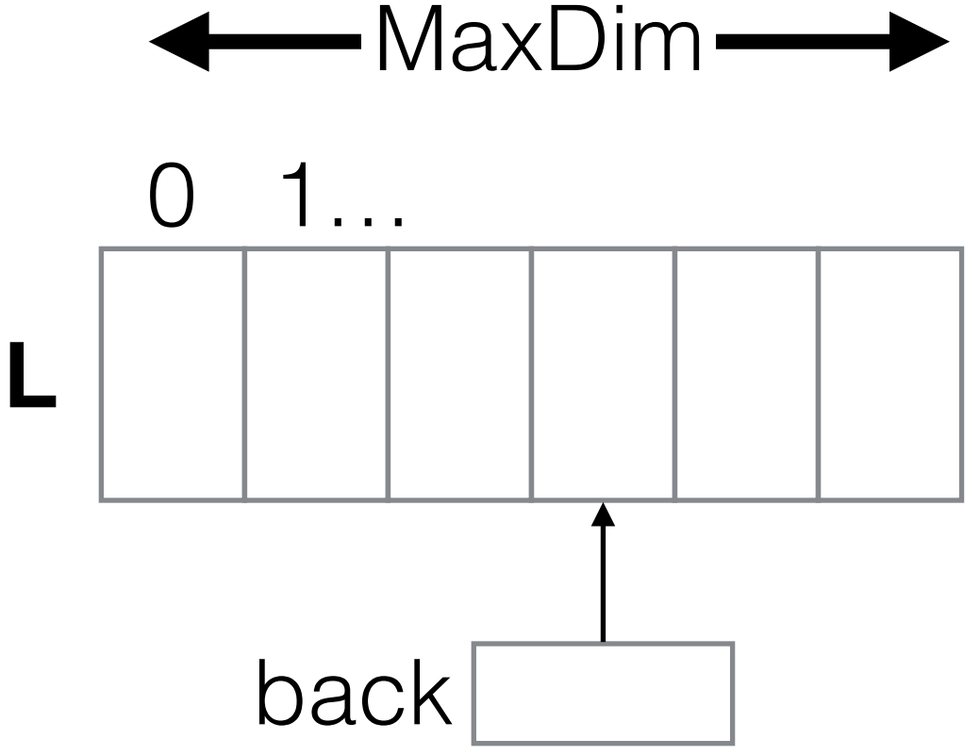
# Threads in Java

```
class MyThread extends Thread{
    int Id = 0;  // The Id of the thread

    public MyThread( int id ){
        Id = id;
    }

    public void run(){
        System.out.println( "thread "+Id+" is running..." );

        try {
            Thread.sleep( 1000 ); // 1 sec
        }
        catch (InterruptedException ie) {
            ie.printStackTrace();
        }

        System.out.println( "thread "+Id+" is done..." );
    }
}
```

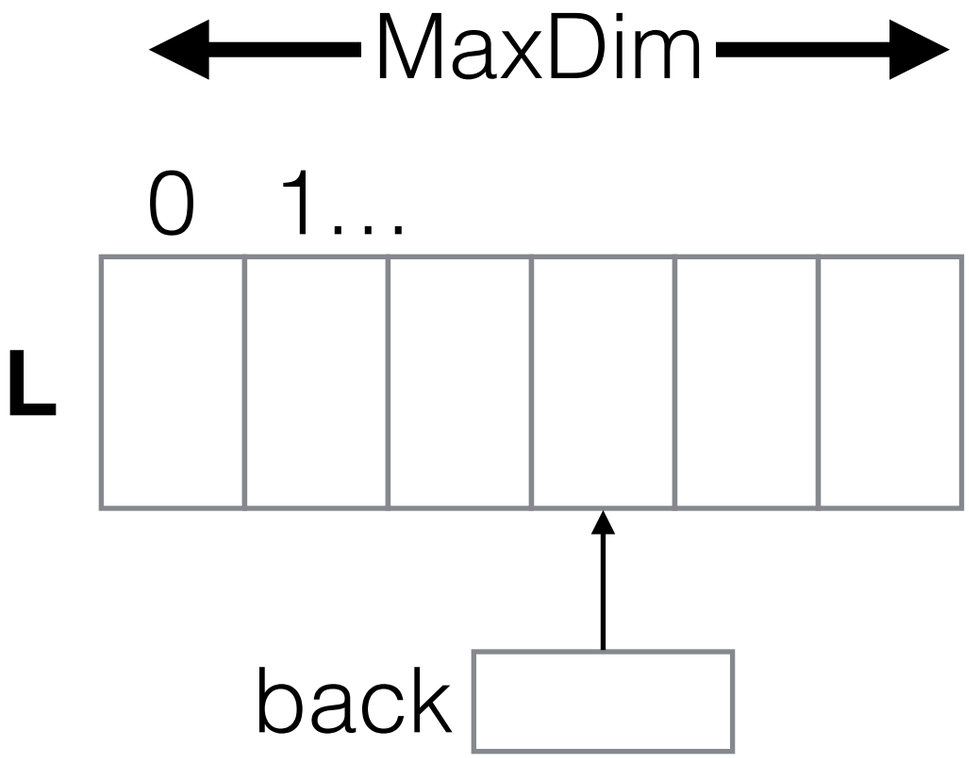
```
public class SimpleThreadDemoWithArgs {
    public static void main(String args[]){

        // declare a new thread with Id 33
        MyThread t1=new MyThread( 33 );

        // start a new thread
        t1.start();

        // wait for the thread to be done
        try {
            t1.join();
        }
        catch (InterruptedException ie) {
            ie.printStackTrace();
        }

        // Exit
        System.out.println( "Done!" );
    }
}
```

# Things can go wrong when sharing data…

# Case Study

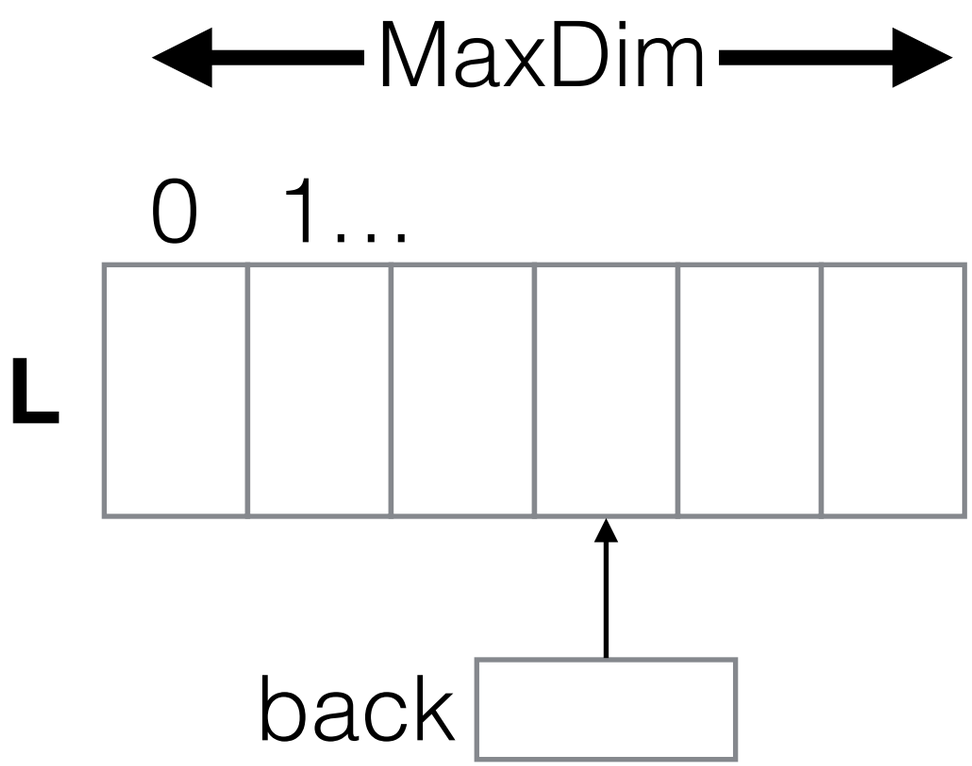


# Case Study




```
void insert( int item ) {

}
```

# Case Study




```
void insert( int item ) {
}
```

```
;; assembly
```

# Class Exercise: Only 1 thread

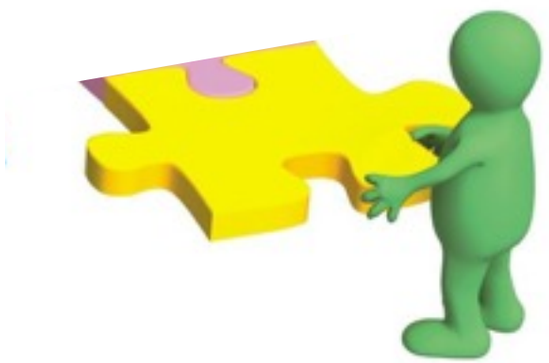


Photo credit: http://www.brocku.ca/blogs/futurestudents/files/2014/10/puzzle-work.jpg

# Class Exercise:
# 2 threads

Photo credit: http://www.brocku.ca/blogs/futurestudents/files/2014/10/puzzle-work.jpg

# Class Exercise:
# 2 threads


Photo credit: http://www.brocku.ca/blogs/futurestudents/files/2014/10/puzzle-work.jpg

Photo credit: http://combiboilersleeds.com/picaso/bad/bad-8.html

# Example of Badly Synchronized Java Code


```
public class UnsynchronizedThreadExample {

    static int sum = 0;

    class PiThreadBad extends Thread {
        private int N;

        public PiThreadBad( int Id, int N ) {
            super( "Thread-"+Id );
            this.N          = N;
        }

        @Override
        public void run() {
            for ( int i=0; i<N; i++ )
                sum ++;
        }
    }
```

```
    public void process( int N ) {
        long startTime = System.currentTimeMillis();
        PiThreadBad t1 = new PiThreadBad( 0, N );
        PiThreadBad t2 = new PiThreadBad( 1, N );

        //--- start two threads ---
        t1.start();
        t2.start();

        //--- wait till they finish ---
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println( "sum = " + sum );
        System.out.println( "Execution time: " +
                (System.currentTimeMillis()-startTime) + " ms" );
    }

    public static void main(String[] args) {
        int N = 100000000;
        UnsynchronizedThreadExample U =
                new UnsynchronizedThreadExample();
        U.process( N );
    }
}
```

http://www.science.smith.edu/dftwiki/index.php/CSC352:_Java_Threads_and_Synchronization_Examples

# Output of Code on Previous Slide

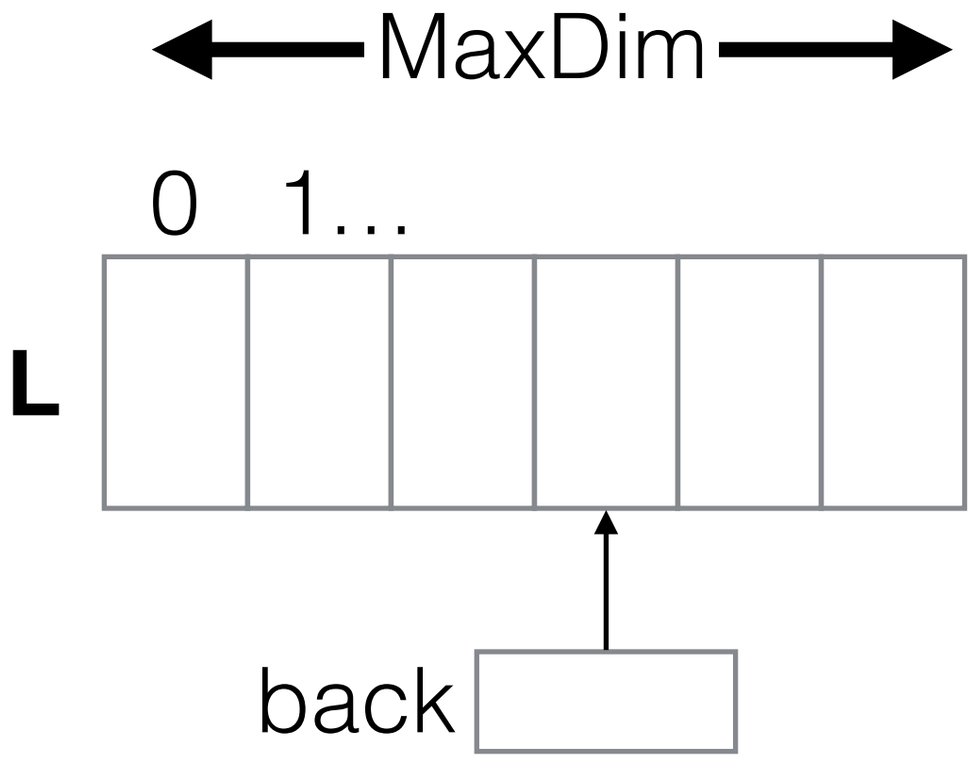
```
aurora:~/handout> java UnsynchronizedThreadExample
sum = 100219659
Execution time: 28 ms
aurora:~/handout> java UnsynchronizedThreadExample
sum = 192885651
Execution time: 25 ms
aurora:~/handout> java UnsynchronizedThreadExample
sum = 100221389
Execution time: 25 ms
aurora:~/handout> java UnsynchronizedThreadExample
sum = 100000000
Execution time: 21 ms
aurora:~/handout> java UnsynchronizedThreadExample
sum = 100220082
Execution time: 28 ms
aurora:~/handout> java UnsynchronizedThreadExample
sum = 173506596
Execution time: 21 ms
aurora:~/handout> java UnsynchronizedThreadExample
sum = 100000000
Execution time: 21 ms
aurora:~/handout> java UnsynchronizedThreadExample
sum = 100000000
Execution time: 21 ms
aurora:~/handout>
```

# That's a

# Possible solution

# #1

# Back to List Example



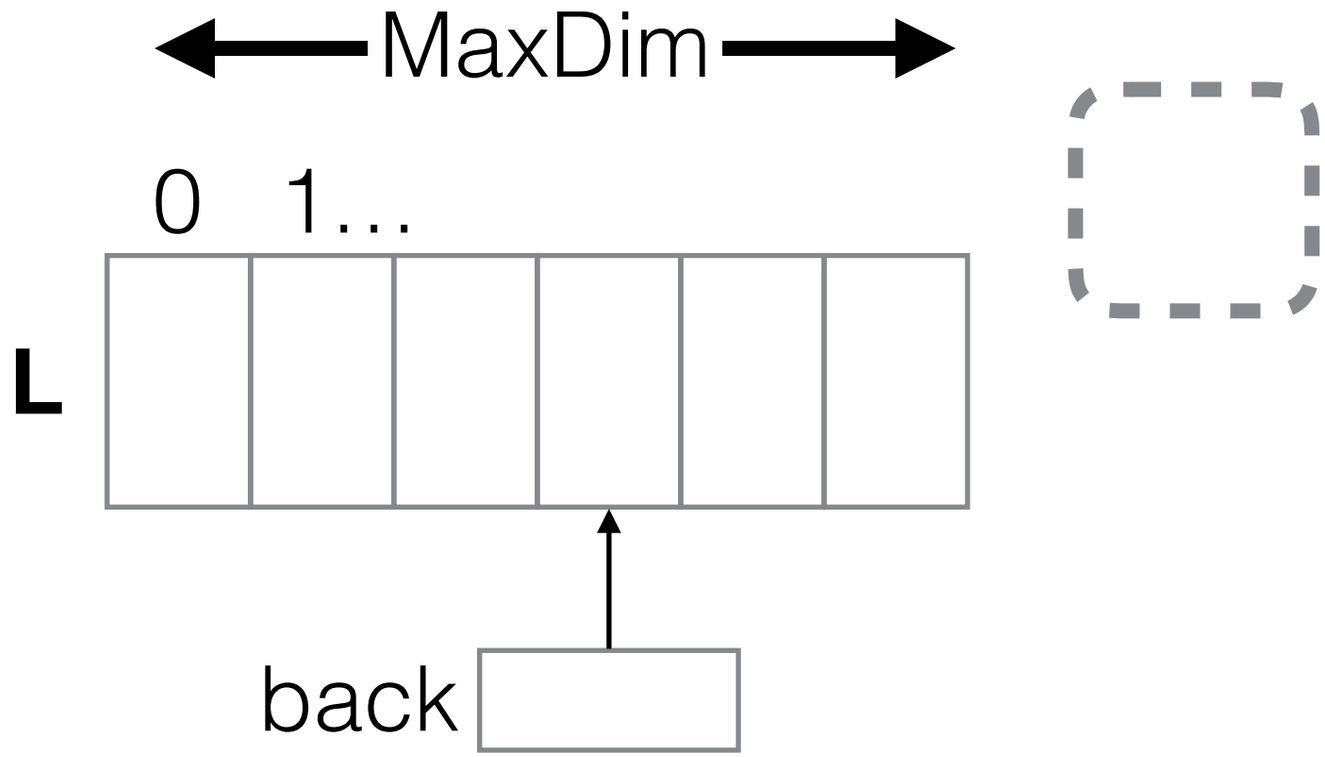

```
void insert( int item ) {



  if (end < MaxDim ) {
       L[end] = item;
       end++;
   }


}
```

# Solution #2

- Create a mechanism for *atomicity*
  - at the processor level
  - between cores

# Class Exercise:
# A List with a Semaphore

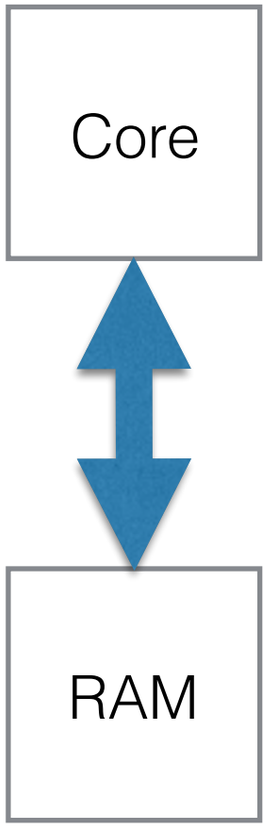
Photo credit: http://www.brocku.ca/blogs/futurestudents/files/2014/10/puzzle-work.jpg


MaxDim
0
1...
L
back

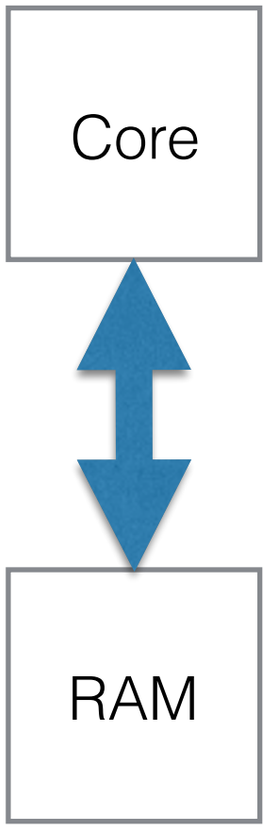
# Locks

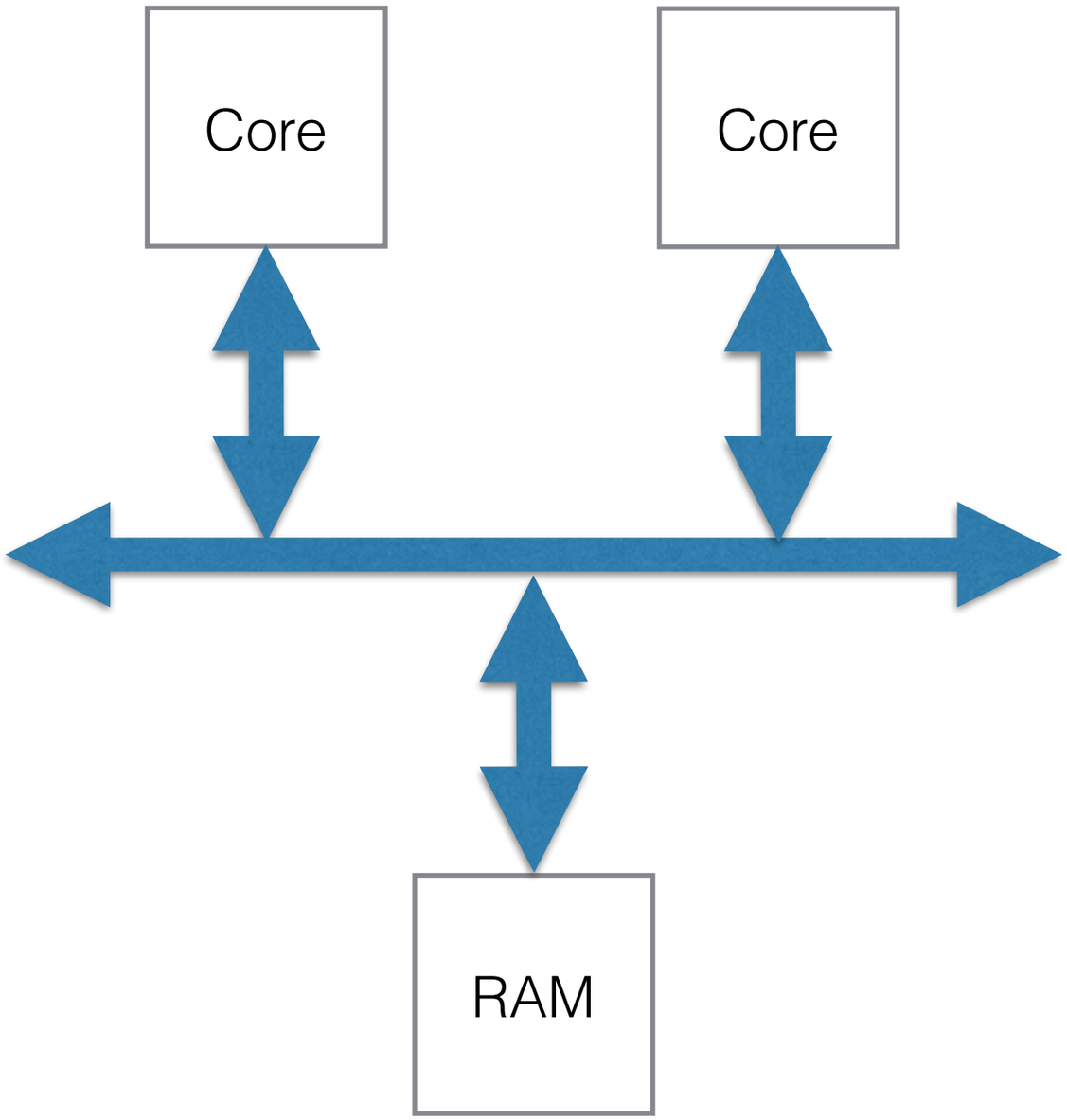
- Lock
- Mutex
- Semaphore
- Monitor
- Test-and-Set
- Compare-and-Swap

# How does this work with 1 Core, 2 Cores?



# How does this work with 1 Core, 2 Cores?



# Java Locks

```
import java.util.concurrent.locks.ReentrantLock;

ReentrantLock lock = new ReentrantLock();

void increment() {

    lock.lock();

    try {

        // Put the code that is protected
        // here

    } finally {

        lock.unlock();

    }
}
```

# Fix Parallel Sum Program


```
import java.util.concurrent.locks.ReentrantLock;

public class SynchronizedLockThreadExample {

        static long sum = 0;
        private ReentrantLock lock = new ReentrantLock();

        class PiThreadGood extends Thread {
                private long N;
                public PiThreadGood( int Id, long N ) {
                        super( "Thread-"+Id ); // give a name to the thread
                        this.N          = N;
                }

                @Override
                public void run() {
                   for ( long i=0; i<N; i++ ) {

                        lock.lock();
                        try{
                           sum++;
                        } finally {
                           lock.unlock();
                        }
                   }
                }
        }
```

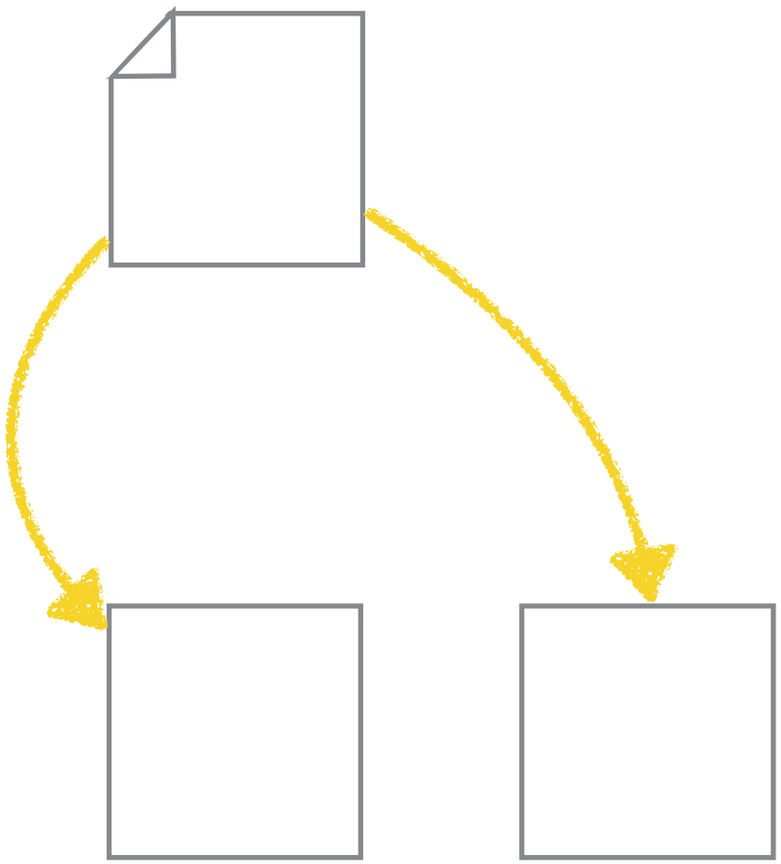
```
    public void process( long N ) {
        long startTime = System.currentTimeMillis();
        PiThreadGood t1 = new PiThreadGood( 0, N );
        PiThreadGood t2 = new PiThreadGood( 1, N );

        //--- start two threads ---
        t1.start();
        t2.start();

        //--- wait till they finish ---
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

                System.out.println( "sum = " + sum );
                System.out.println( "Execution time: " + (System.currentT
            }

    public static void main(String[] args) {
        long N = 1000000000l;
        SynchronizedLockThreadExample S = new SynchronizedL
        S.process( N );
    }

}
```

http://www.science.smith.edu/dftwiki/index.php/CSC352:_Java_Threads_and_Synchronization_Examples

# A Simpler Approach: *synchronized*

# Every Java Object, Every Class…

# Every Java Object, Every Class…



has a lock!

# Option 1

```
public void addNewCustomerMethod( String name ) {
    name = name.trim().toUpper();
    synchronized( this ) {
        myList.add( name );
    }
}
```

# Option 2

```
public synchronized void addNewCustomerMethod( String name ) {
        myList.add( name );
}
```

# Option 3

```
public class MsLunch {
    private long c1 = 0;
    private long c2 = 0;
    private Object lock1 = new Object();
    private Object lock2 = new Object();

    public void inc1() {
        synchronized(lock1) {
            c1++;
        }
    }

    public void inc2() {
        synchronized(lock2) {
            c2++;
        }
    }
}
```

# Revisiting Parallel Sum Program

```
class PiThreadGood extends Thread {
   private long N;                    // the total number of samples/iterations

   public PiThreadGood( int Id, long N ) {
       super( "Thread-"+Id );
       this.N          = N;
   }

   private synchronized void inc() {
       sum++;
   }

   @Override
   public void run() {
       for ( long i=0; i<N; i++ )
           inc();
   }
}
```

# Revisiting Parallel Sum Program

```
class PiThreadGood extends Thread {
   private long N;                    // the total number of samples/iterations

   public PiThreadGood( int Id, long N ) {
       super( "Thread-"+Id );
       this.N          = N;
   }

   private synchronized void inc() {
       sum++;
   }

   @Override
   public void run() {
       for ( long i=0; i<N; i++ )
           inc();
   }
}
```

```
aurora:~/handout> java SynchronizedThreadExample
sum = 2000000000
Execution time: 14172 ms
aurora:~/handout>
```