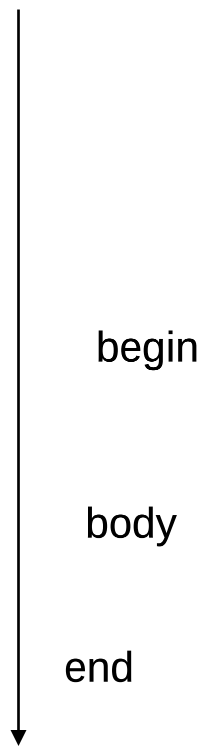
Multithreaded Programming in Java

Agenda

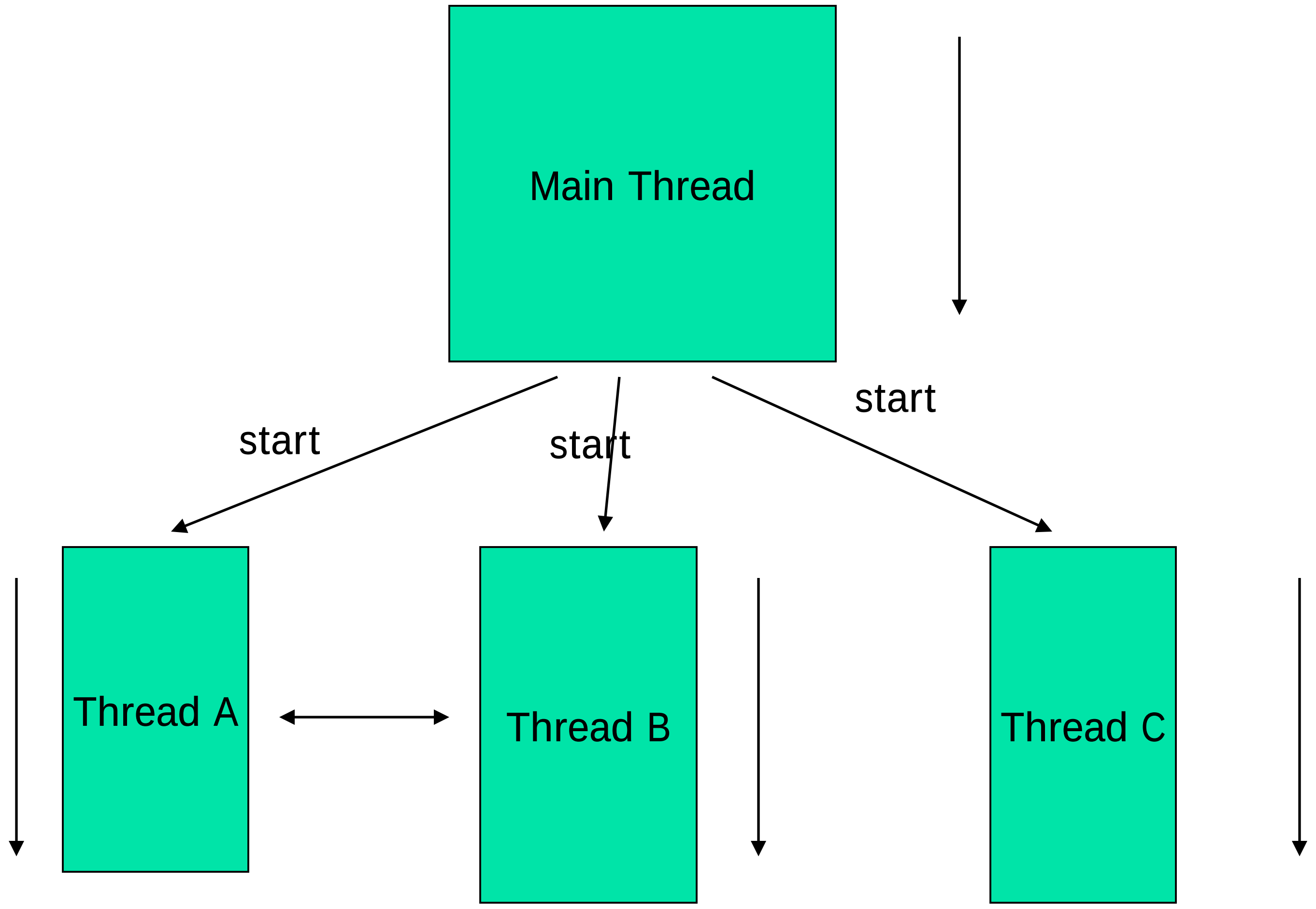
- Introduction
- Thread Applications
- Defining Threads
- Java Threads and States
- Examples

A single threaded program

```
class ABC
{
....
     public void main(..)
     {
     ...
     ..
     }
}
```

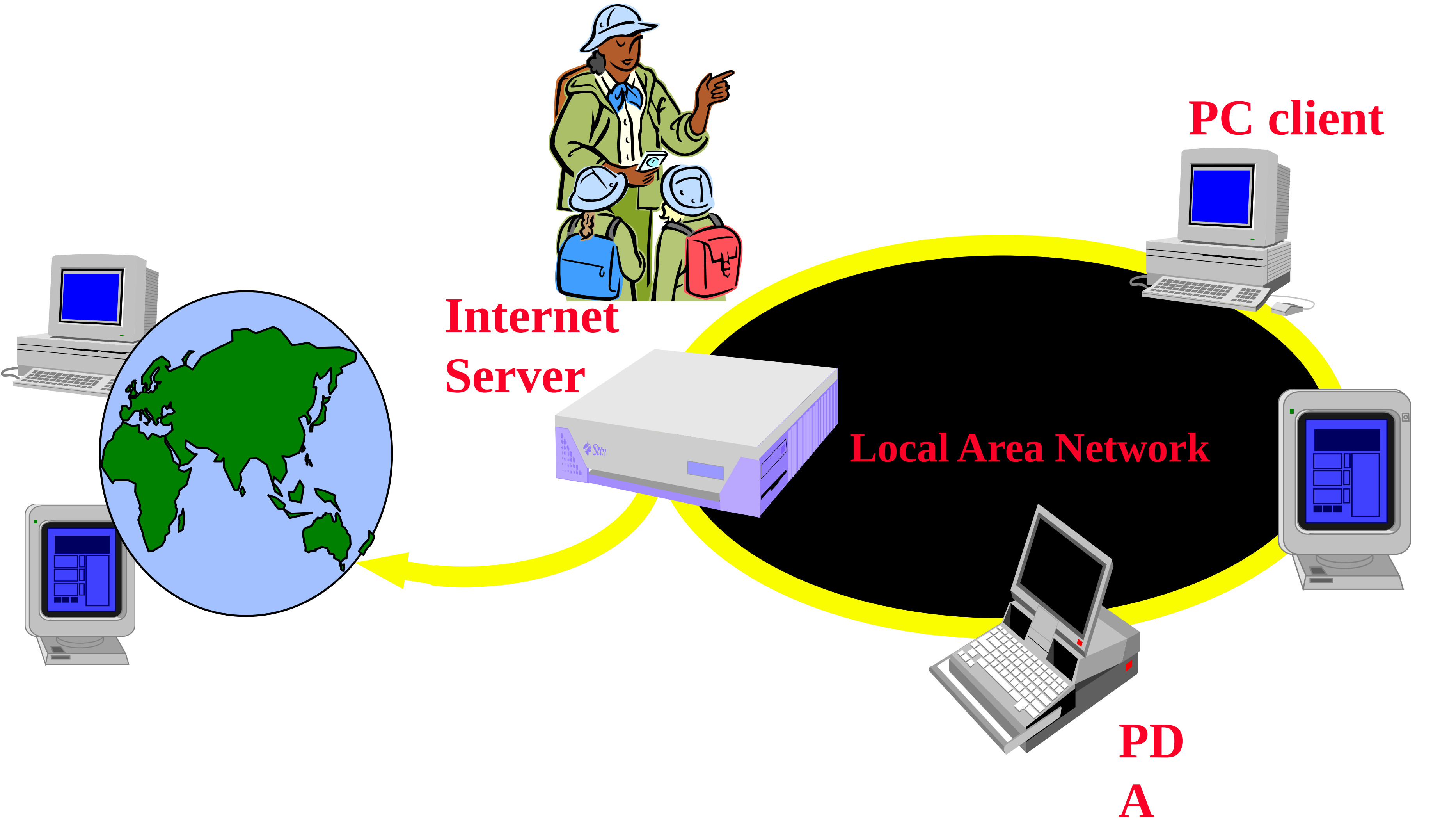


A Multithreaded Program

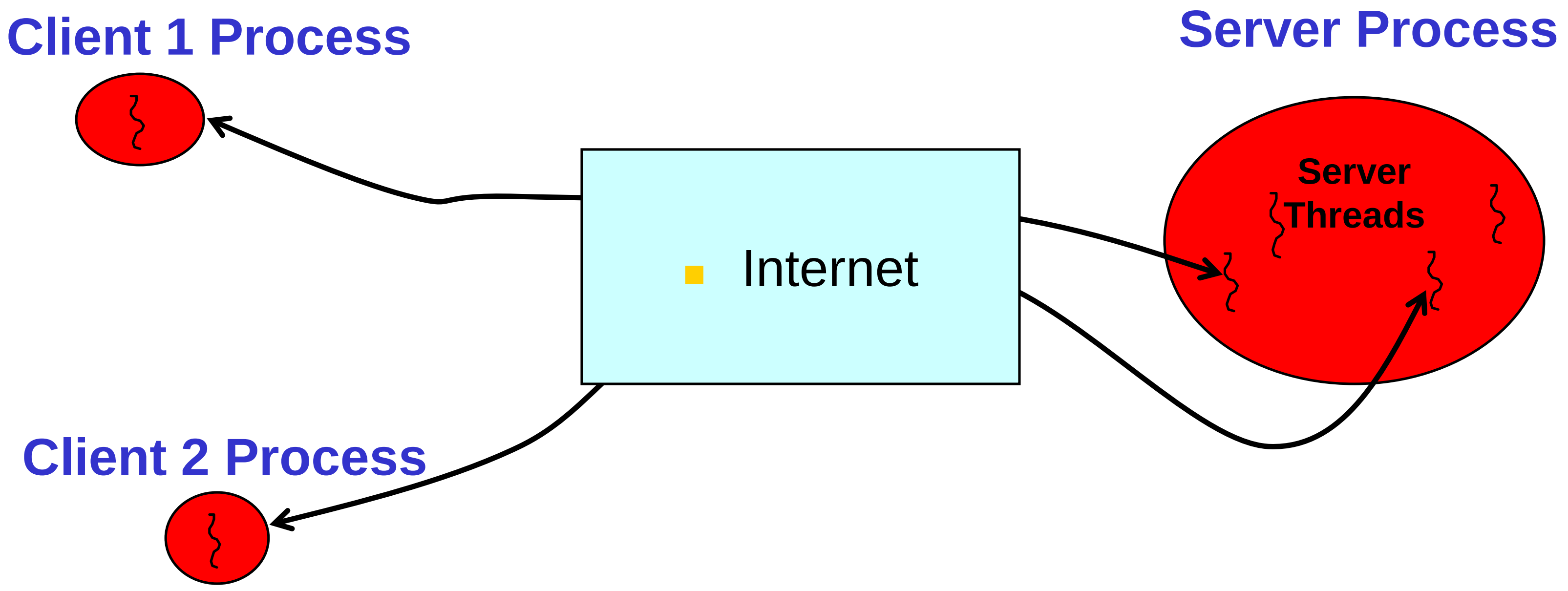


Threads may switch or exchange data/results

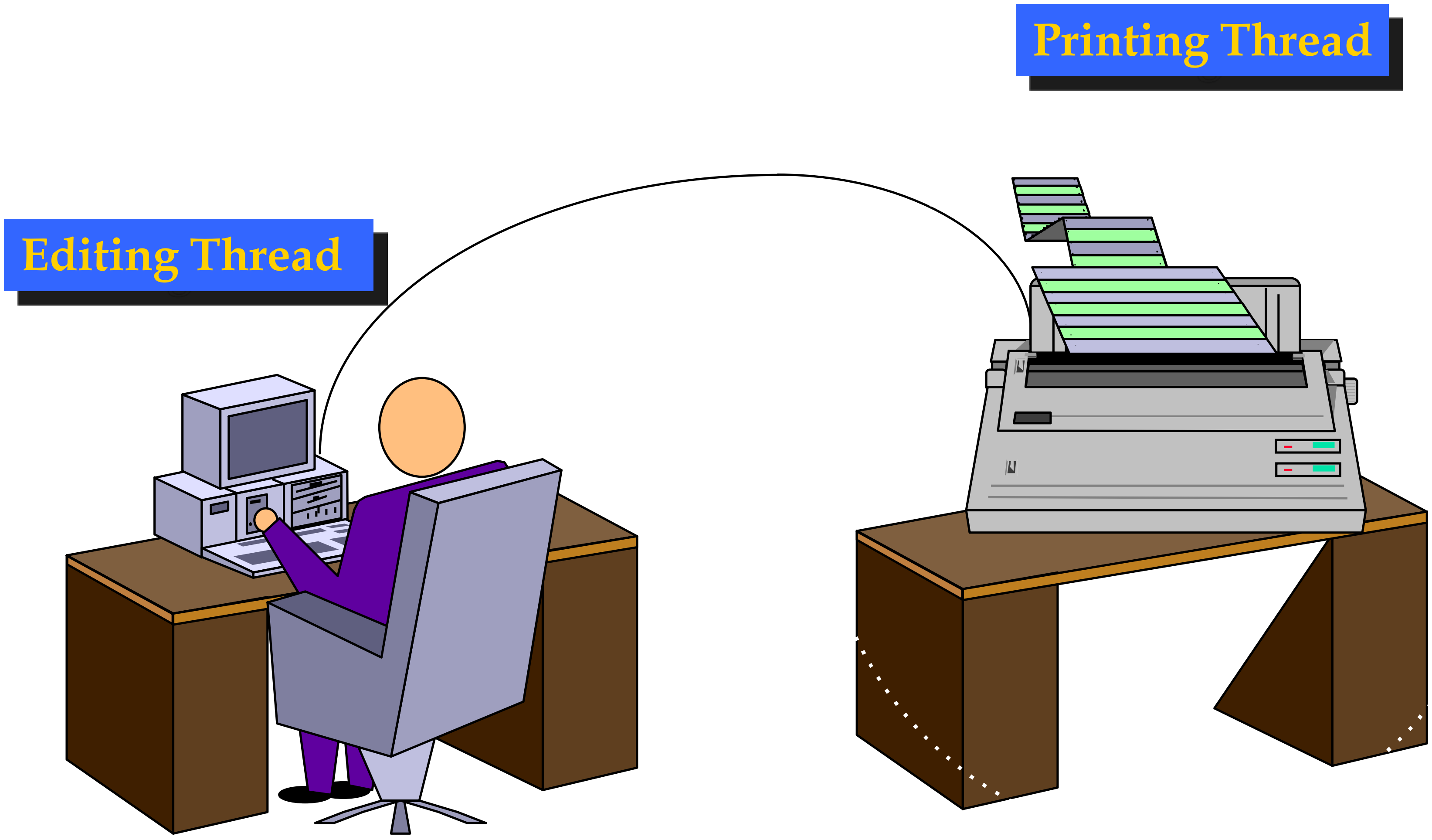
Web/Internet Applications: Serving Many Users Simultaneously



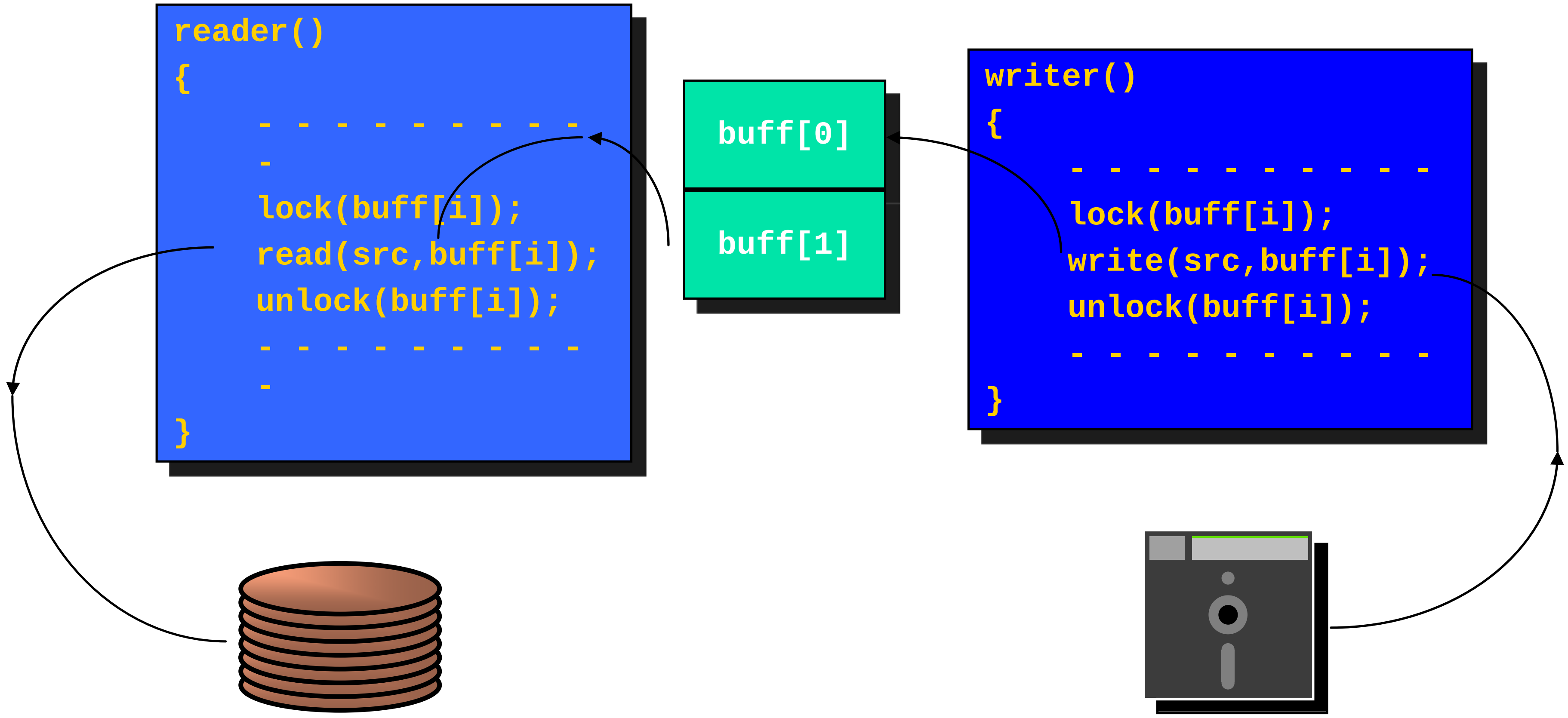
Multithreaded Server: For Serving Multiple Clients Concurrently



Modern Applications need Threads (ex1): Editing and Printing documents in background.



Multithreaded/Parallel File Copy

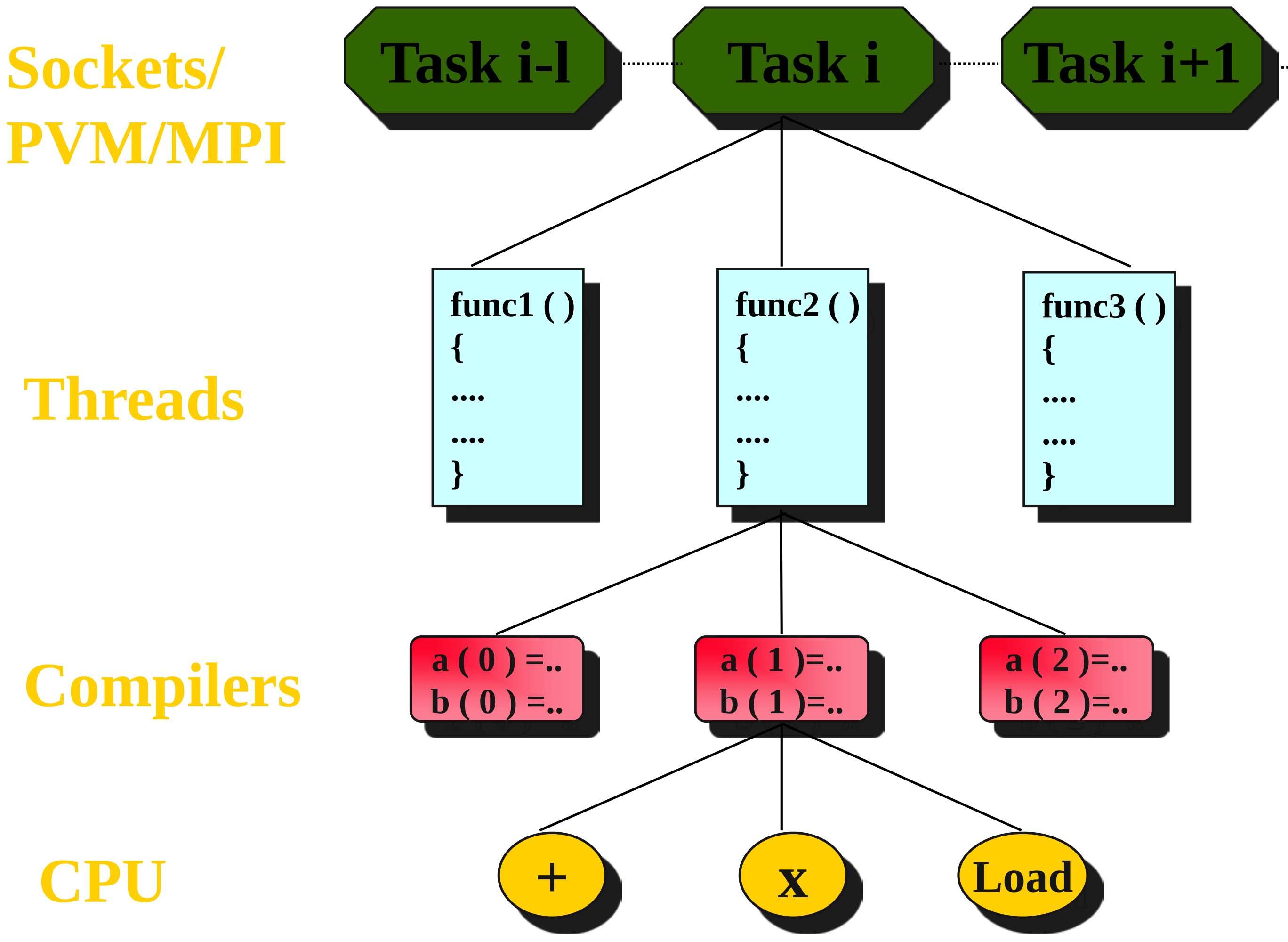


```
reader()
{
      - - - - - - - - - -
      -
      lock(buff[i]);
      read(src,buff[i]);
      unlock(buff[i]);
      - - - - - - - - - -
      -
}
```

```
writer()
{
      - - - - - - - - - -
      lock(buff[i]);
      write(src,buff[i]);
      unlock(buff[i]);
      - - - - - - - - - -
}
```

Cooperative Parallel Synchronized Threads

Levels of Parallelism



<u>**Code-Granularity**</u>

Code Item

Large grain (task level)
Program

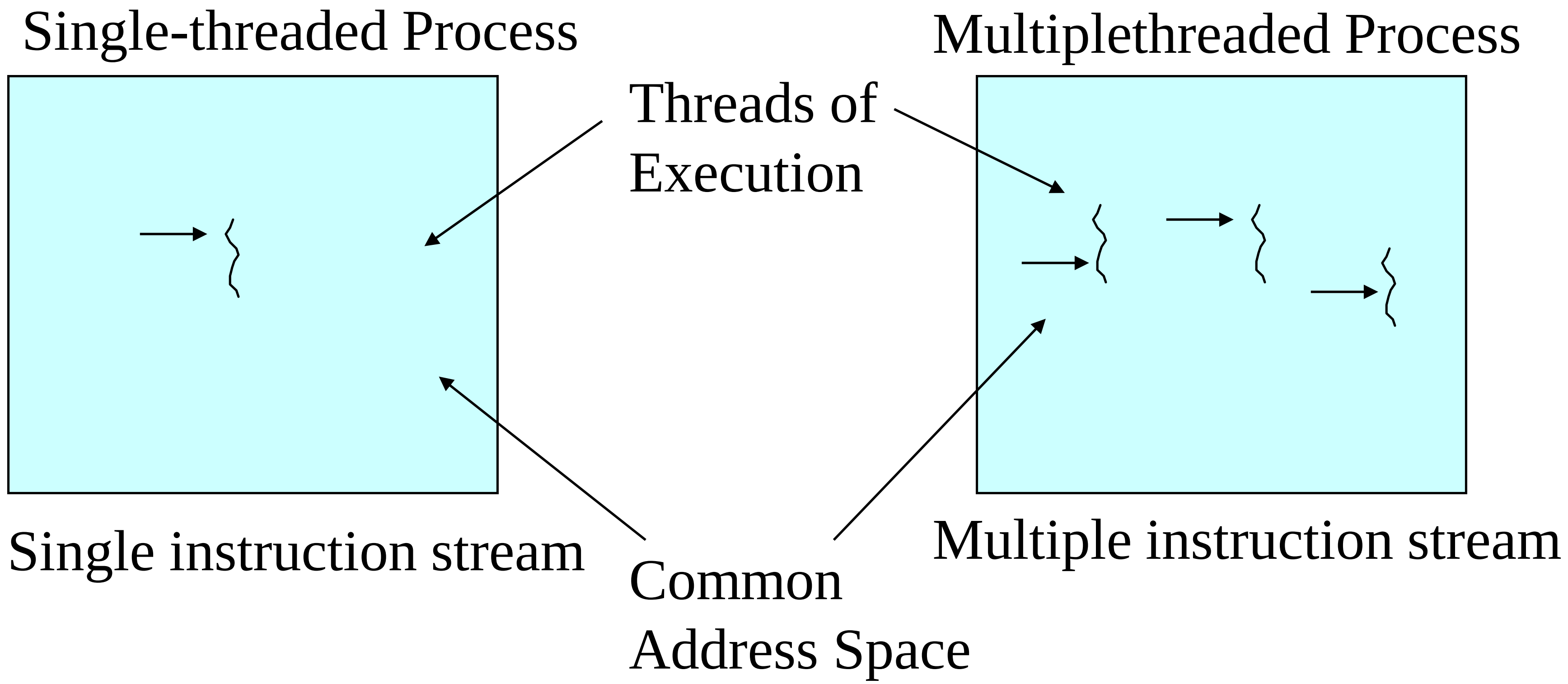
Medium grain (control level)
Function (thread)

Fine grain (data level)
Loop (Compiler)

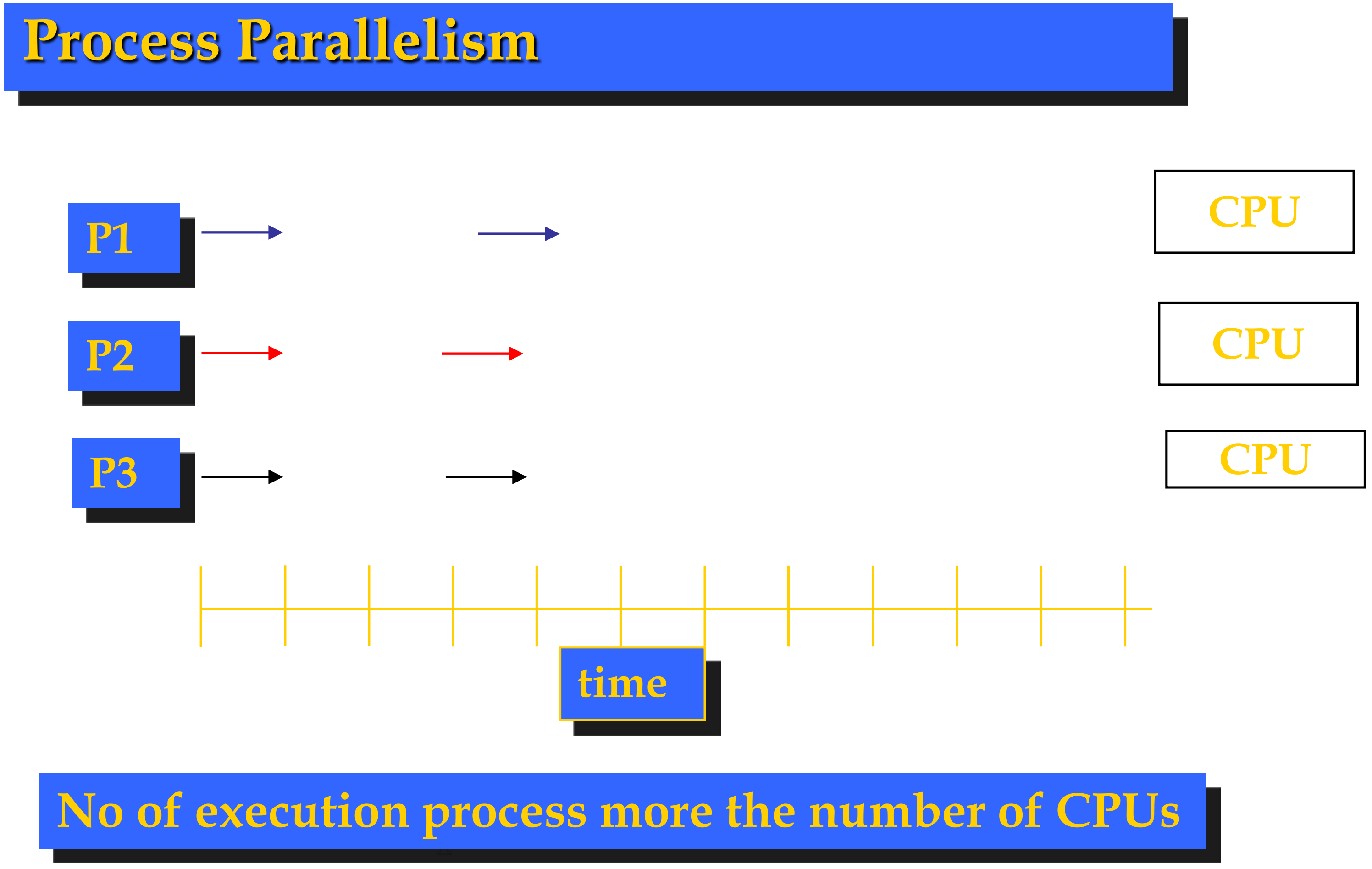
Very fine grain (multiple issue)
With hardware

Single and Multithreaded Processes

<u>threads are light-weight processes within a process</u>

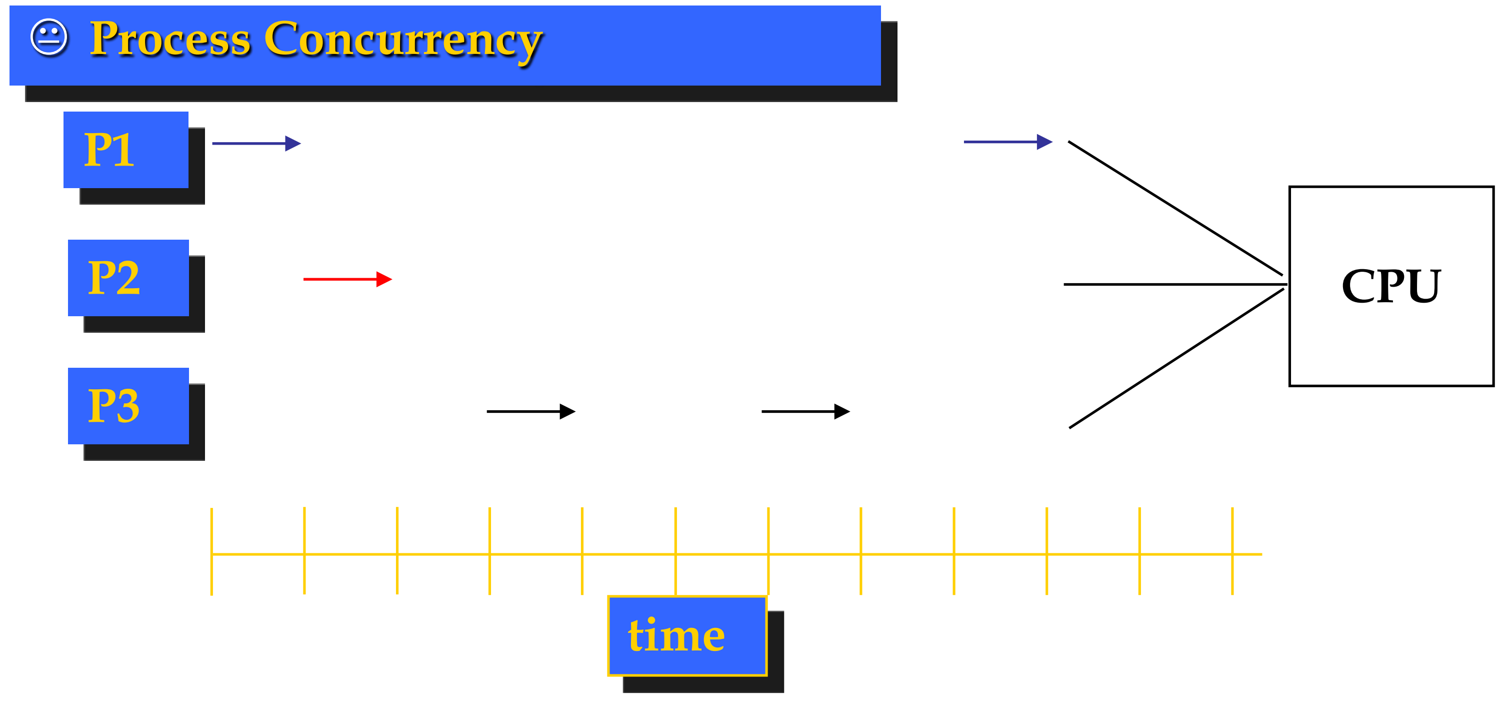


Multithreading - Multiprocessors



Multithreading on Uni-processor

- Concurrency Vs Parallelism



Number of Simultaneous execution units > number of CPUs

What are Threads?

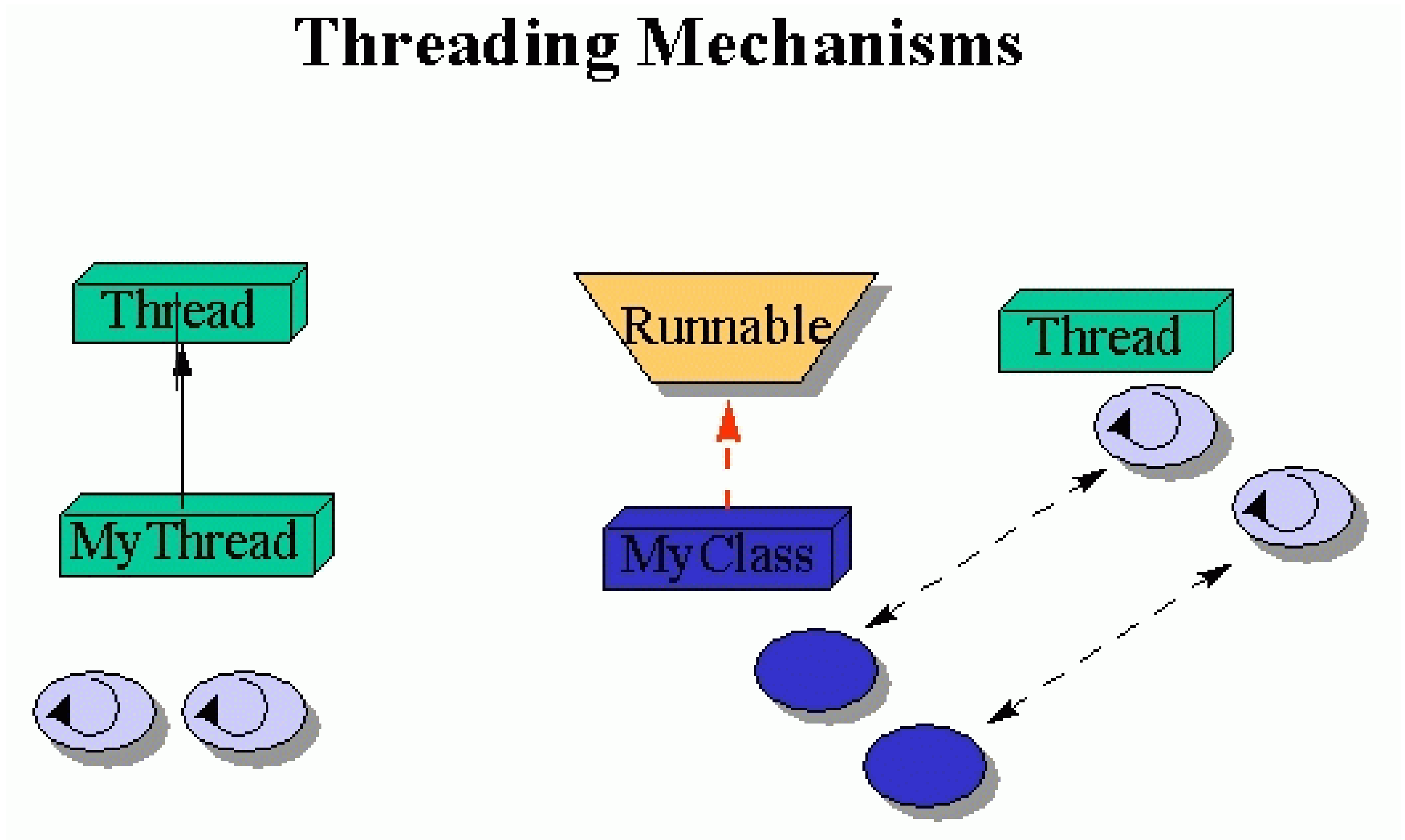
- A piece of code that run in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are being extensively used express concurrency on both single and multiprocessors machines.
- Programming a task having multiple threads of control – Multithreading or Multithreaded Programming.

Java Threads

- Java has built in thread support for Multithreading
- Synchronization
- Thread Scheduling
- Inter-Thread Communication:
  - currentThread start setPriority
  - yield run getPriority
  - sleep stop suspend
  - resume
- Java Garbage Collector is a low-priority thread

Threading Mechanisms...

- Create a class that extends the Thread class
- Create a class that implements the Runnable interface



1st method: Extending Thread class

- Threads are implemented as objects that contains a method called run()

```
class MyThread extends Thread
{
    public void run()
    {
        // thread body of execution
    }
 }
```

- Create a thread:

```
MyThread thr1 = new MyThread();
```

- Start Execution of threads:

```
thr1.start();
```

An example

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

class ThreadEx1 {                           / /  a program that utilizes the thread
        public static void main(String [] args  ) {
             MyThread t = new MyThread();
             // due to extending the Thread class (above)
             //  I can call start(), and this will call
             // run(). start() is a method in class Thread.
             t.start();
       } // end main()
}        // end class ThreadEx1
```

2nd method: Threads by implementing Runnable interface

```
class MyThread implements Runnable
{
  .....
  public void run()
  {
     // thread body of execution
  }
}
```

- Creating Object:

  ```
  MyThread myObject = new MyThread();
  ```

- Creating Thread Object:

  ```
  Thread thr1 = new Thread( myObject );
  ```

- Start Execution:

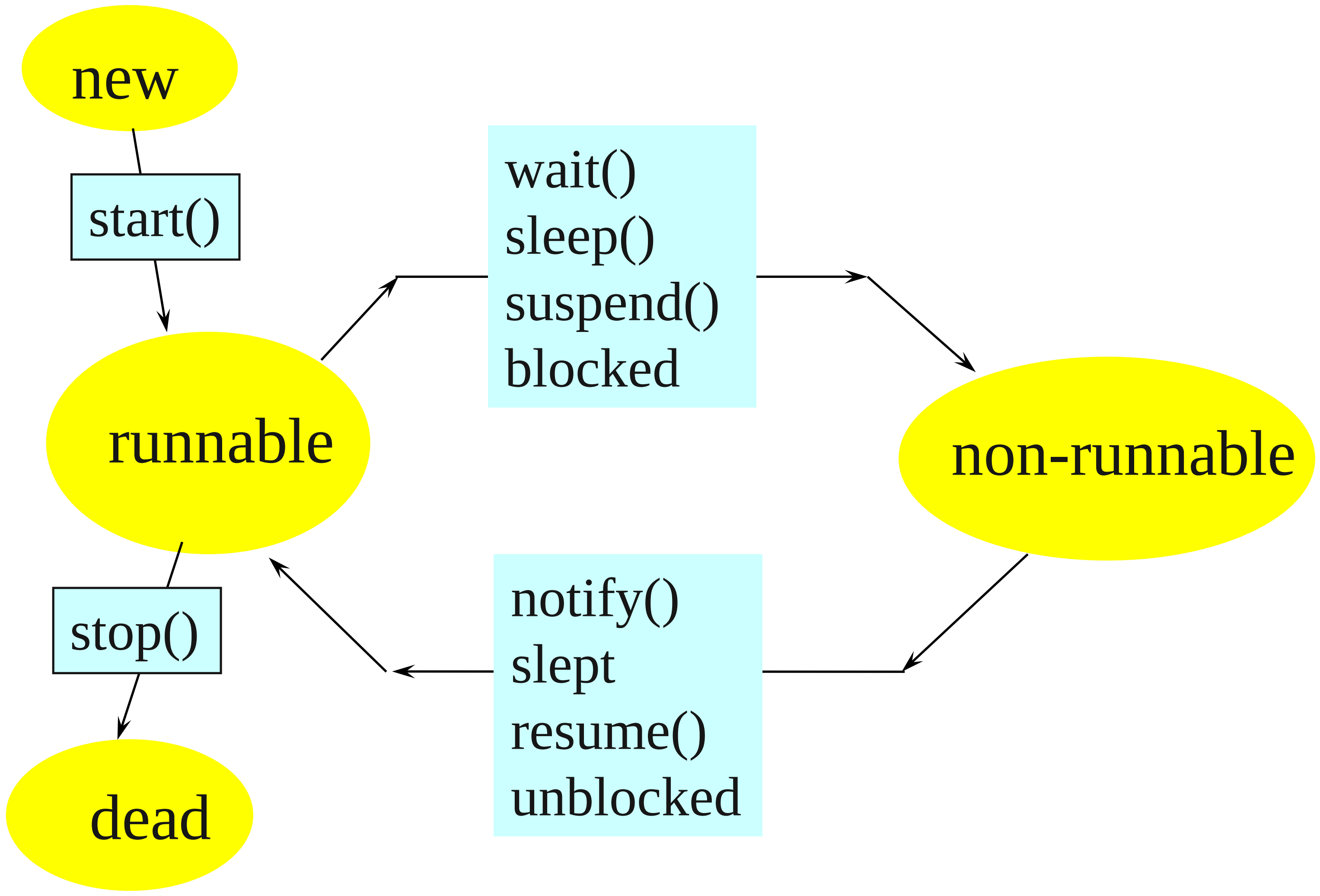
  ```
  thr1.start();
  ```

An example

```
class MyThread implements Runnable  {
        public void run() {
                System.out.println(" this thread is running ... ");
        }
} // end class MyThread

class ThreadEx2 {
        public static void main(String [] args  ) {
                Thread t = new Thread(new MyThread());
                            // due to implementing the Runnable interface
                           // I can call start(), and this will call run().
                t.start();
        } // end main()
}       // end class ThreadEx2
```

Life Cycle of Thread



A Program with Three Java Threads

- Write a program that creates 3 threads

Three threads example

```
class A extends Thread
{
     public void run()
       {
          for(int i=1;i<=5;i++)
            {
                System.out.println("\t From ThreadA: i= "+i);
            }

             System.out.println("Exit from A");
       }

}

class B extends Thread
{
     public void run()
       {

          for(int j=1;j<=5;j++)
            {
                System.out.println("\t From ThreadB: j= "+j);
            }

             System.out.println("Exit from B");
       }

}
```

```
class C extends Thread
{
      public void run()
        {

            for(int k=1;k<=5;k++)
              {
                   System.out.println("\t From ThreadC: k= "+k);
              }

               System.out.println("Exit from C");
        }

}


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

          {
                 new A().start();
                 new B().start();
                 new C().start();

          }



}
```

Run 1

-
  ```
  [raj@mundroo] threads [1:76] java ThreadTest
       From ThreadA: i= 1
       From ThreadA: i= 2
       From ThreadA: i= 3
       From ThreadA: i= 4
       From ThreadA: i= 5
  Exit from A
       From ThreadC: k= 1
       From ThreadC: k= 2
       From ThreadC: k= 3
       From ThreadC: k= 4
       From ThreadC: k= 5
  Exit from C
       From ThreadB: j= 1
       From ThreadB: j= 2
       From ThreadB: j= 3
       From ThreadB: j= 4
       From ThreadB: j= 5
  Exit from B
  ```

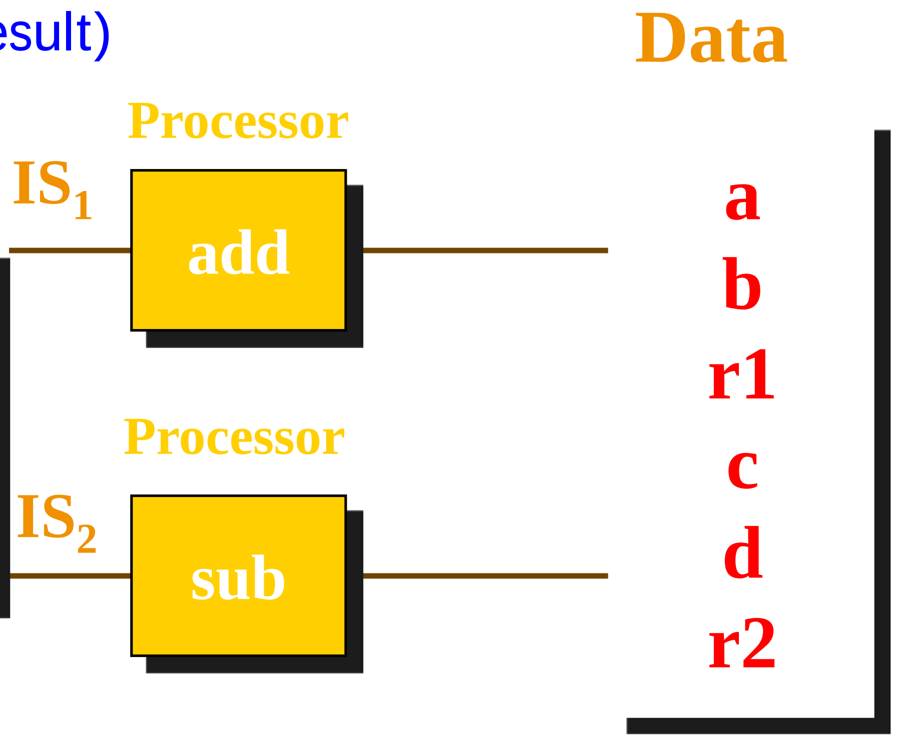
Run2

```
[raj@mundroo] threads [1:77] java ThreadTest
     From ThreadA: i= 1
     From ThreadA: i= 2
     From ThreadA: i= 3
     From ThreadA: i= 4
     From ThreadA: i= 5
     From ThreadC: k= 1
     From ThreadC: k= 2
     From ThreadC: k= 3
     From ThreadC: k= 4
     From ThreadC: k= 5
Exit from C
     From ThreadB: j= 1
     From ThreadB: j= 2
     From ThreadB: j= 3
     From ThreadB: j= 4
     From ThreadB: j= 5
Exit from B
Exit from A
```

Process Parallelism

- int add (int a, int b, int & result)
- // function stuff
- int sub(int a, int b, int & result)
- // function stuff

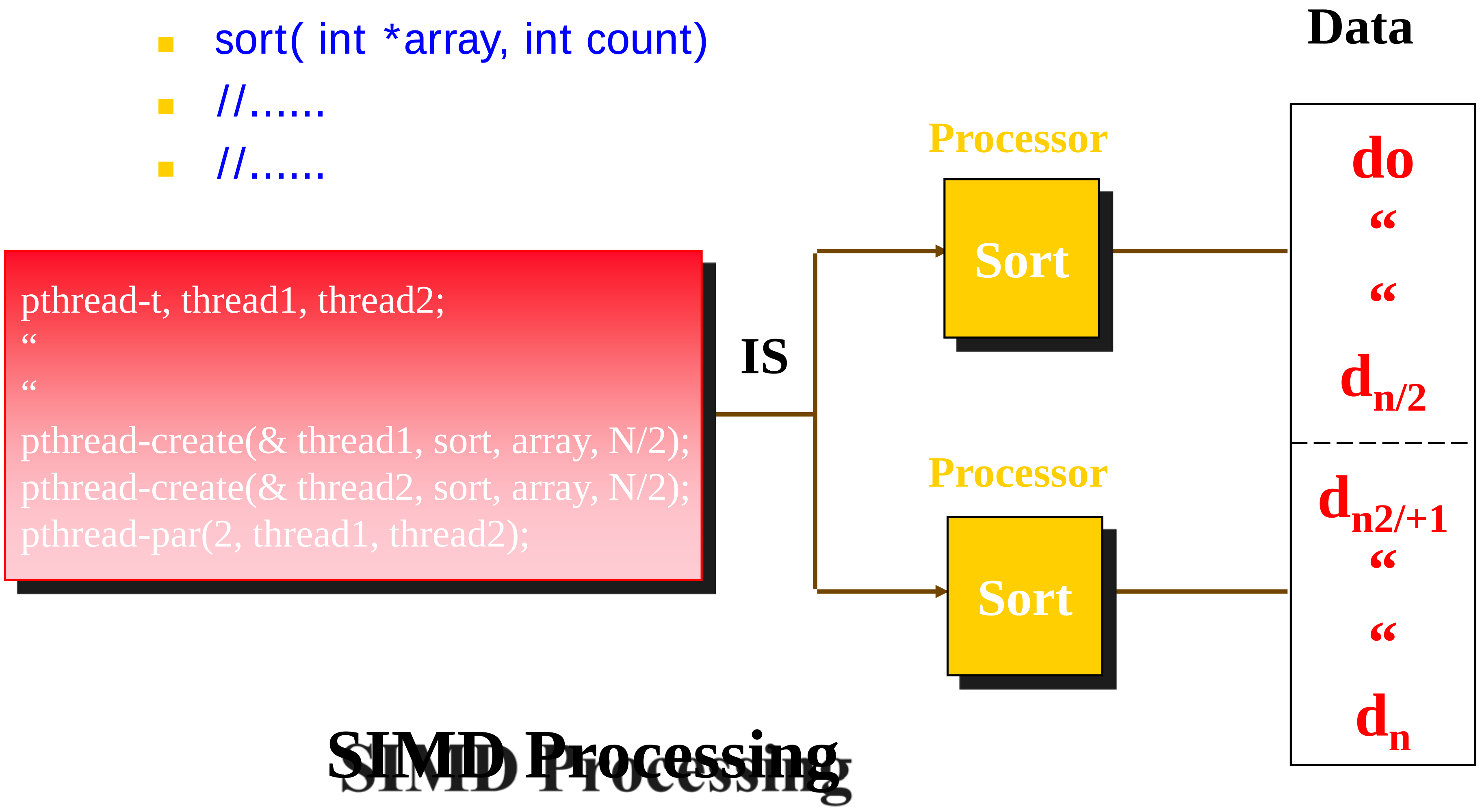
```
pthread t1, t2;
pthread-create(&t1, add, a,b, & r1);
pthread-create(&t2, sub, c,d, & r2);
pthread-par (2, t1, t2);
```



MISD and MIMD Processing

Data Parallelism

- sort(int *array, int count)
- //......
- //......



```
pthread-t, thread1, thread2;
“
“
pthread-create(& thread1, sort, array, N/2);
pthread-create(& thread2, sort, array, N/2);
pthread-par(2, thread1, thread2);
```

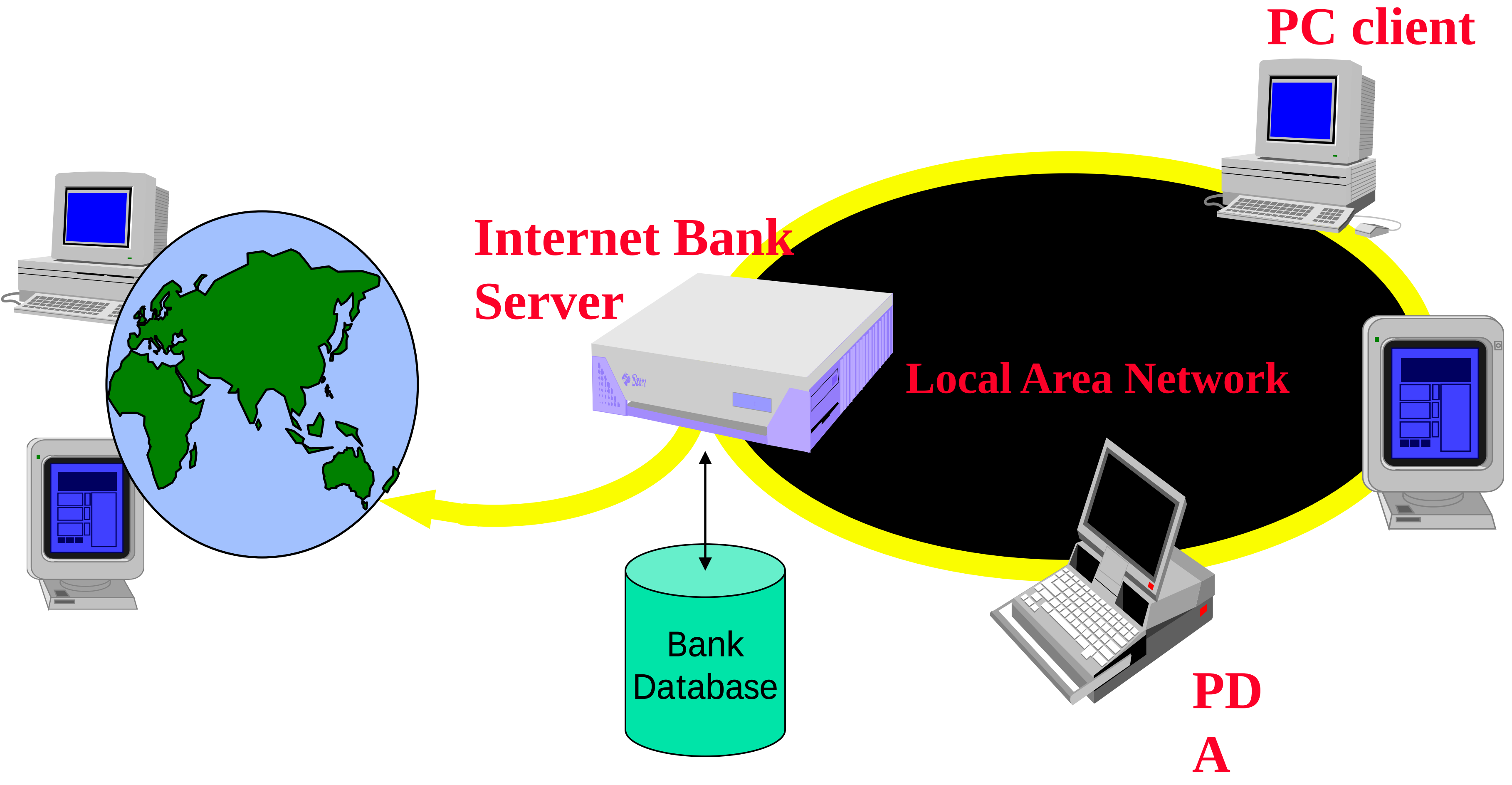
Next Class

- Thread Synchronisation
- Thread Priorities

Accessing Shared Resources

- Applications Access to Shared Resources need to be coordinated.
  - Printer (two person jobs cannot be printed at the same time)
  - Simultaneous operations on your bank account

Online Bank: Serving Many Customers and Operations



Shared Resources

- If one thread tries to read the data and other thread tries to update the same date, it leads to inconsistent state.
- This can be prevented by synchronising access to data.
- In Java: “Synchronized” method:
  - syncronised void update()
  - {
    - ...
  - }

the driver: 3rd Threads sharing the same object

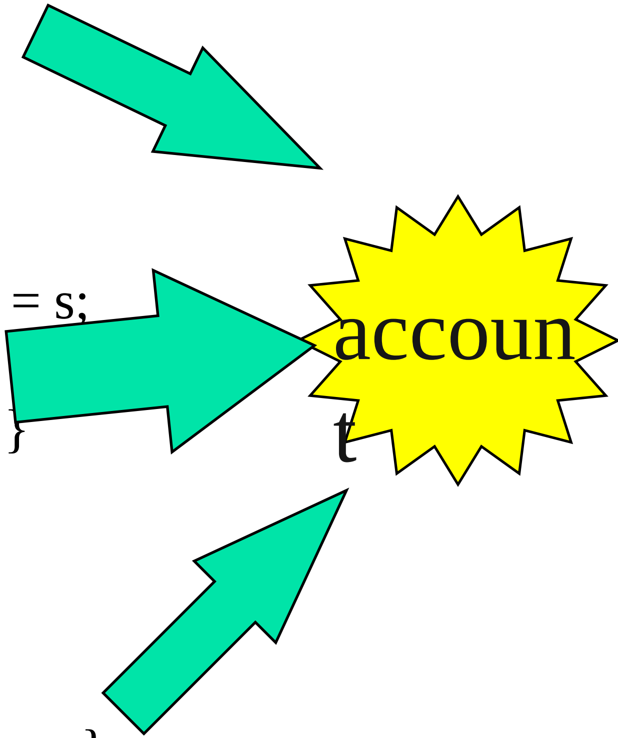
```
class InternetBankingSystem {
        public static void main(String [] args  ) {
            Account accountObject = new Account ();
            Thread t1 = new Thread(new MyThread(accountObject));
            Thread t2 = new Thread(new YourThread(accountObject));
            Thread t3 = new Thread(new HerThread(accountObject));
            t1.start();
            t2.start();
            t3.start();
          // DO some other operation
        } // end main()
}
```

Program with 3 threads and shared object

```
class MyThread implements Runnable  {
 Account account;
        public MyThread (Account s) {  account = s;}
        public void run() { account.deposit(); }
} // end class MyThread

class YourThread implements Runnable  {
 Account account;
        public YourThread (Account s) { account = s;
}
        public void run() { account.withdraw(); }
} // end class YourThread

class HerThread implements Runnable  {
 Account account;
        public HerThread (Account s) { account = s; }
        public void run() {account.enquire(); }
} // end class HerThread
```



Monitor (shared object) example

```
class Account {    // the 'monitor'
// DATA Members
  int balance;

// if 'synchronized' is removed, the outcome is unpredictable
 public synchronized void deposit( ) {
    // METHOD BODY : balance += deposit_amount;
  }

   public synchronized void withdraw( ) {
      // METHOD BODY: balance -= deposit_amount;
   }
   public synchronized void enquire( ) {
      // METHOD BODY: display balance.
   }
}
```

Thread Priority

- In Java, each thread is assigned priority, which affects the order in which it is scheduled for running. The threads so far had same default priority (ORM_PRIORITY) and they are served using FCFS policy.
  - Java allows users to change priority:
    - ThreadName.setPriority(intNumber)
      - MIN_PRIORITY = 1
      - NORM_PRIORITY=5
      - MAX_PRIORITY=10

Thread Priority Example

```
class A extends Thread
{
       public void run()
         {
              System.out.println("Thread A started");

              for(int i=1;i<=4;i++)
                {
                      System.out.println("\t From ThreadA: i= "+i);
                }

                 System.out.println("Exit from A");
         }

}

class B extends Thread
{
       public void run()
         {
              System.out.println("Thread B started");

              for(int j=1;j<=4;j++)
                {
                      System.out.println("\t From ThreadB: j= "+j);
                }

                 System.out.println("Exit from B");
         }

}
```

Thread Priority Example

```
class C extends Thread
{
      public void run()
        {
            System.out.println("Thread C started");

            for(int k=1;k<=4;k++)
              {
                   System.out.println("\t From ThreadC: k= "+k);
              }
              System.out.println("Exit from C");
        }
}
class ThreadPriority
{
         public static void main(String args[])
          {
                   A threadA=new A();
                   B threadB=new B();
                   C threadC=new C();

                 threadC.setPriority(Thread.MAX_PRIORITY);
                 threadB.setPriority(threadA.getPriority()+1);
                 threadA.setPriority(Thread.MIN_PRIORITY);

                 System.out.println("Started Thread A");
                  threadA.start();

                 System.out.println("Started Thread B");
                  threadB.start();

                 System.out.println("Started Thread C");
                  threadC.start();

                  System.out.println("End of main thread");
          }
}
```