IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

OBJECT ORIENTED PROGRAMMING WITH JAVA

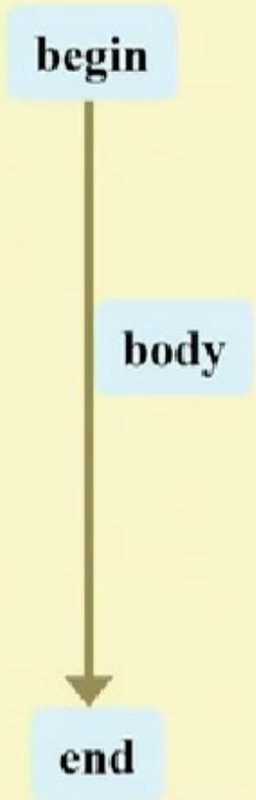
Multithreaded Programming in Java – I

Debasis Samanta
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

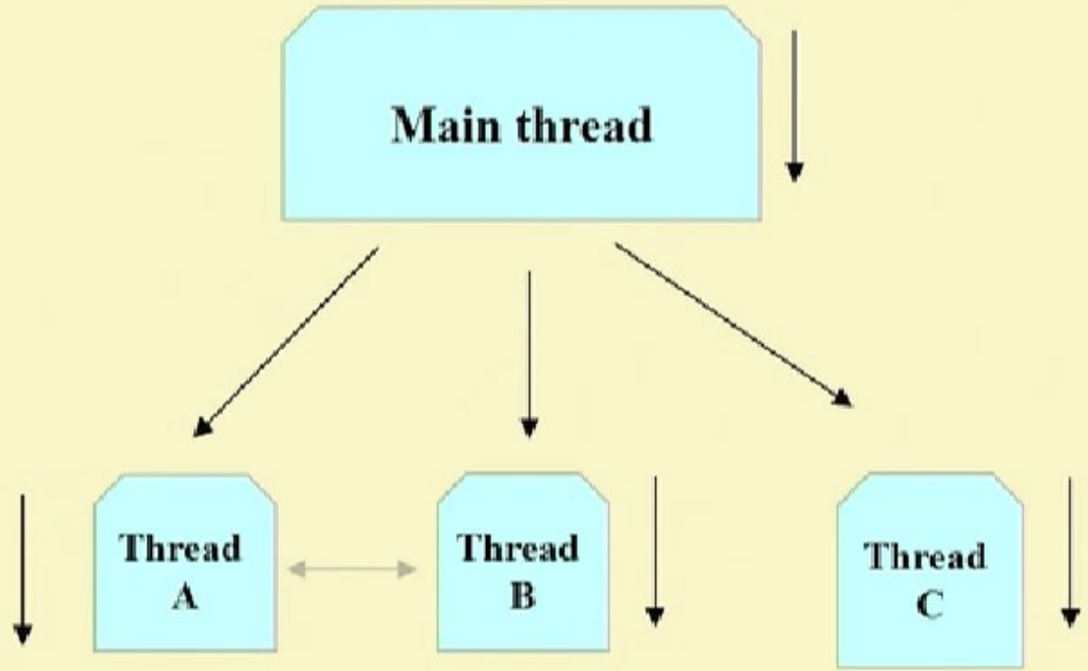
What is Multithreading?

A single threaded program

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

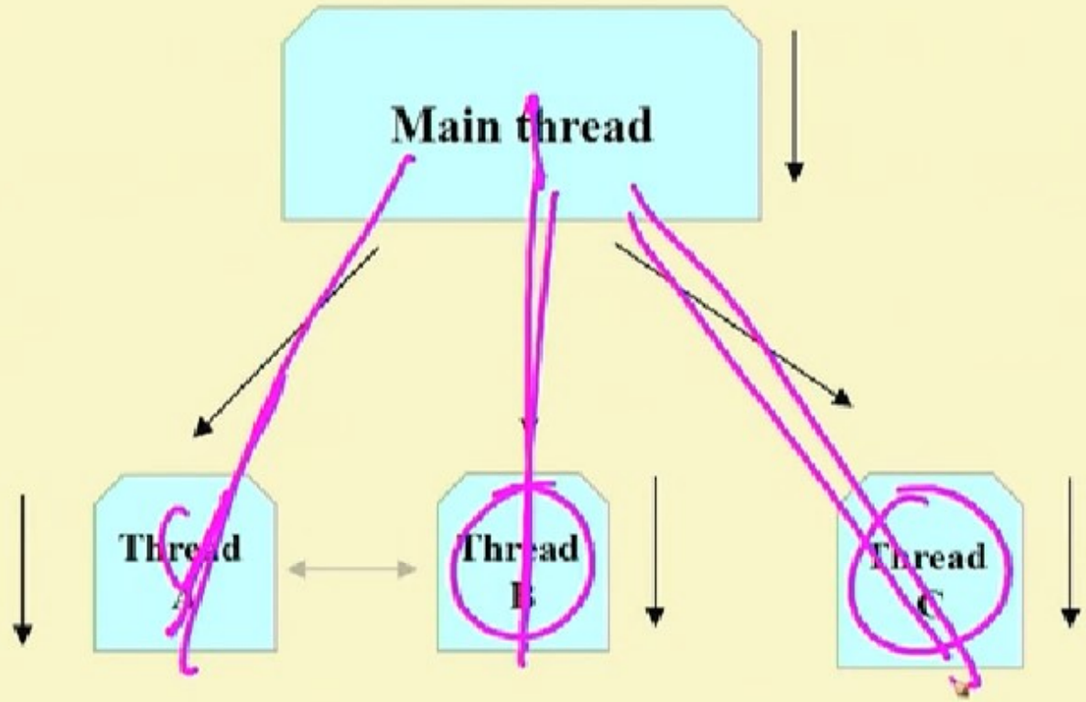


A multithreaded program



Threads may switch or exchange data/ results among them

A multithreaded program



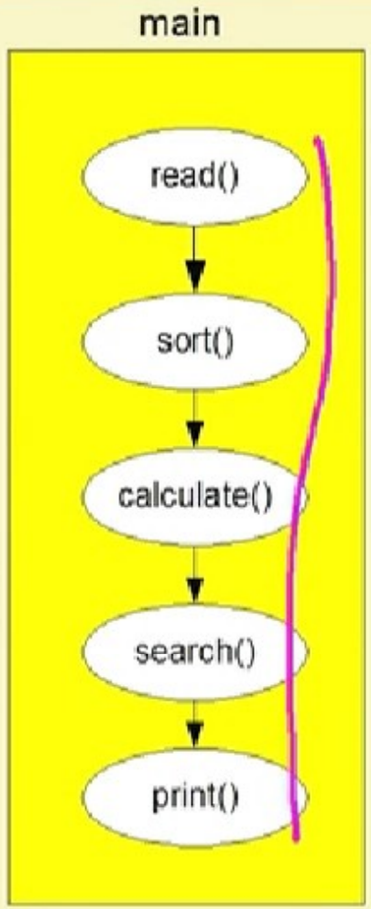
Threads may switch or exchange data/ results among them

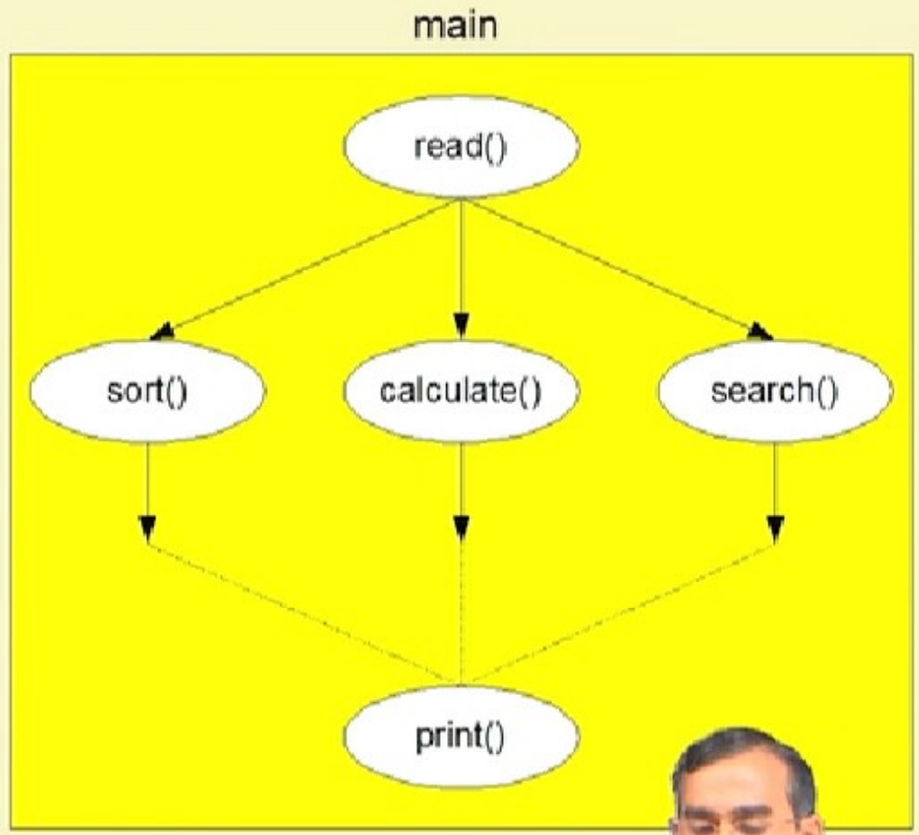
A multithreaded program

```
public class X {
    main () {
        . read()
            { ...};
        . sort()
            {...};
        . calculate()
            {...};
        . search()
            {...};
        . print()
            {...};
}
```

A multithreaded program

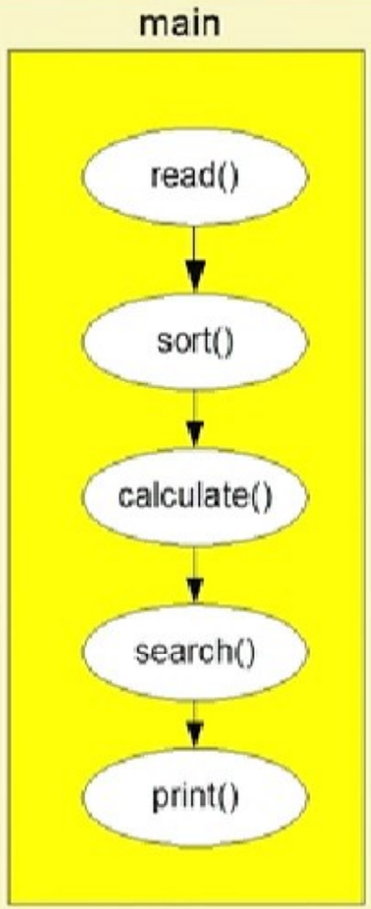
```
public class X {
    main () {
        . read()
            { ...};
        . sort()
            {...};
        . calculate()
            {...};
        . search()
            {...};
        . print()
            {...};
}
```

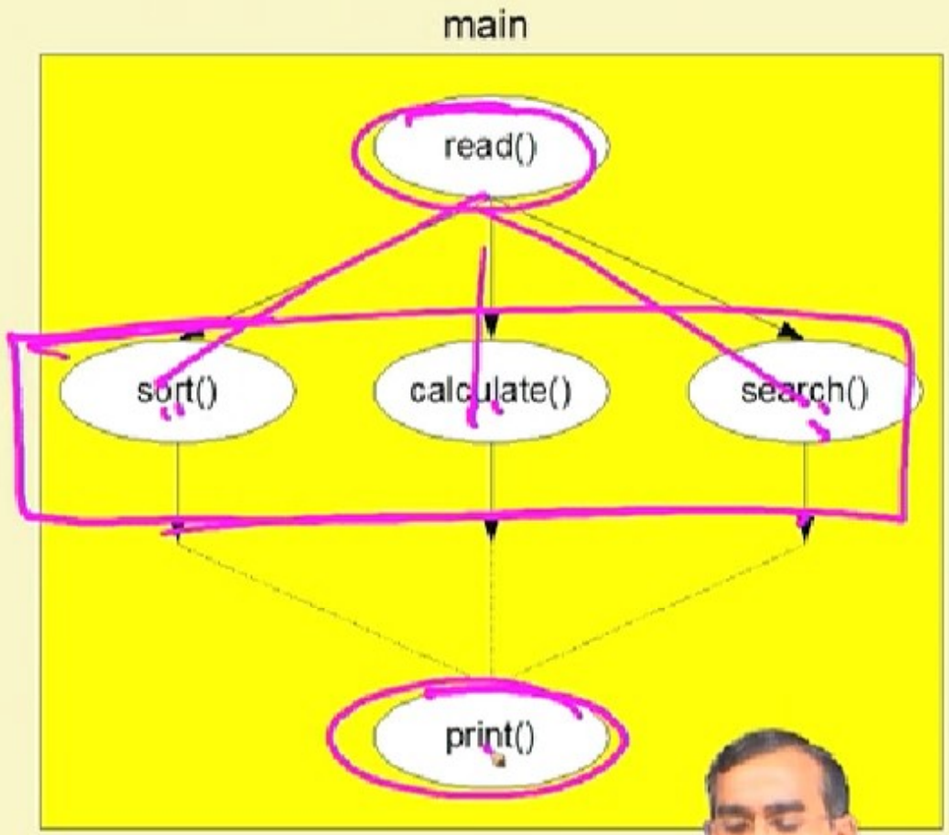




A multithreaded program

```
public class X {
    main () {
        . read()
            { ...};
        . sort()
            {...};
        . calculate()
            {...};
        . search()
            {...};
        . print()
            {...};
}
```





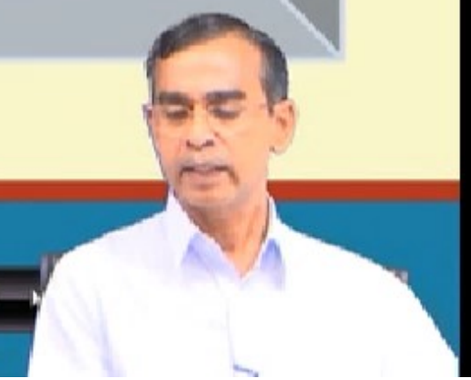
The concept

Multiple tasks in computer

- Draw and display images on screen
- Check keyboard and mouse input
- Send and receive data on network
- Read and write files to disk
- Perform useful computation (editor, browser, game)

How does computer do everything at once?

- Multitasking
- Multiprocessing



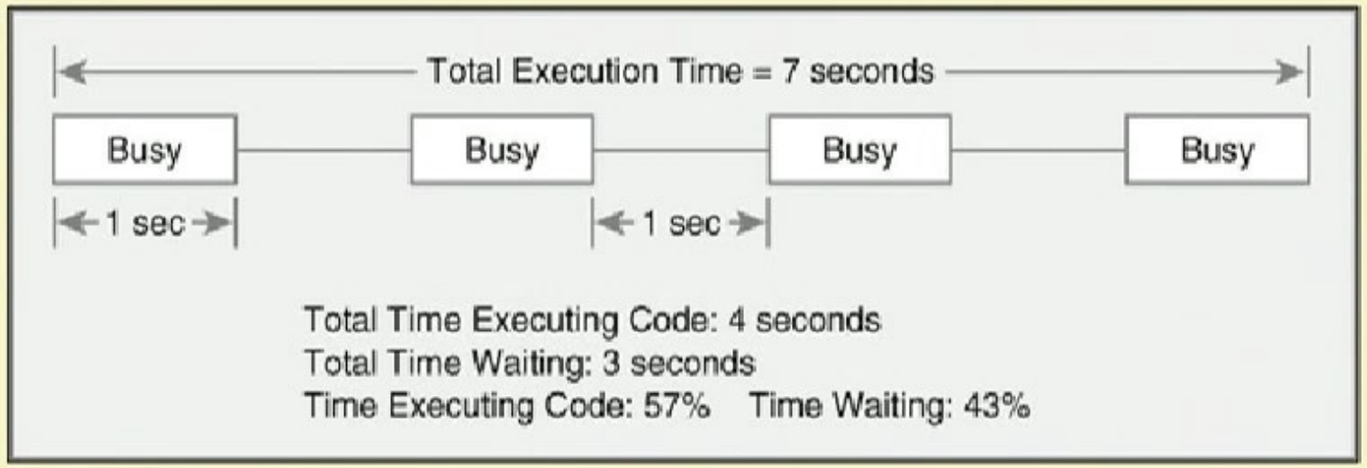
Multitasking (time-sharing)

- Approach
  - Computer does some work on a task
  - Computer then quickly switch to next task
  - Tasks managed by operating system (scheduler)

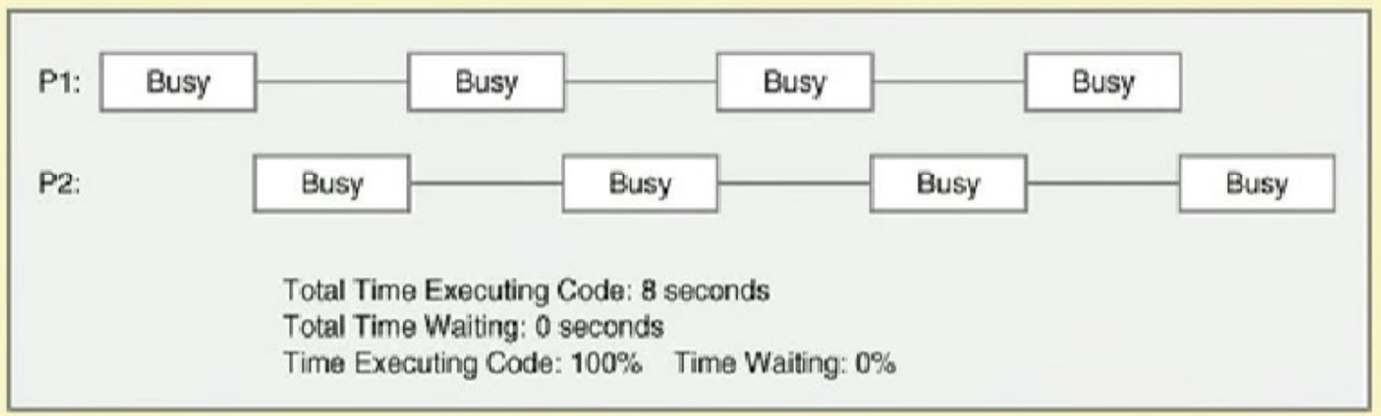
- Computer seems to work on tasks concurrently
- Can improve performance by reducing waiting

Multitasking can improve performance

Single task →

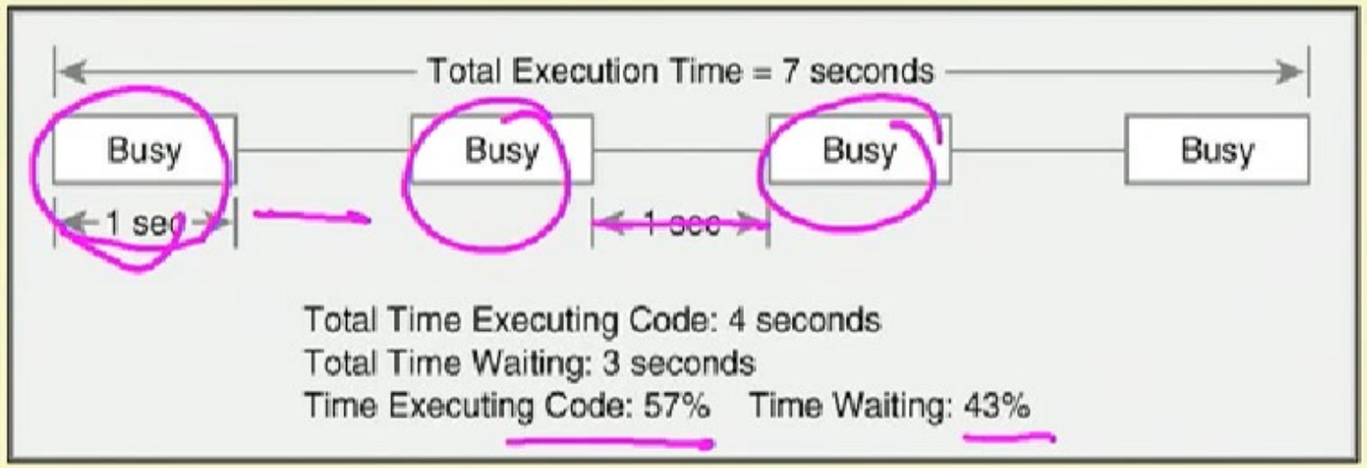


Two tasks →

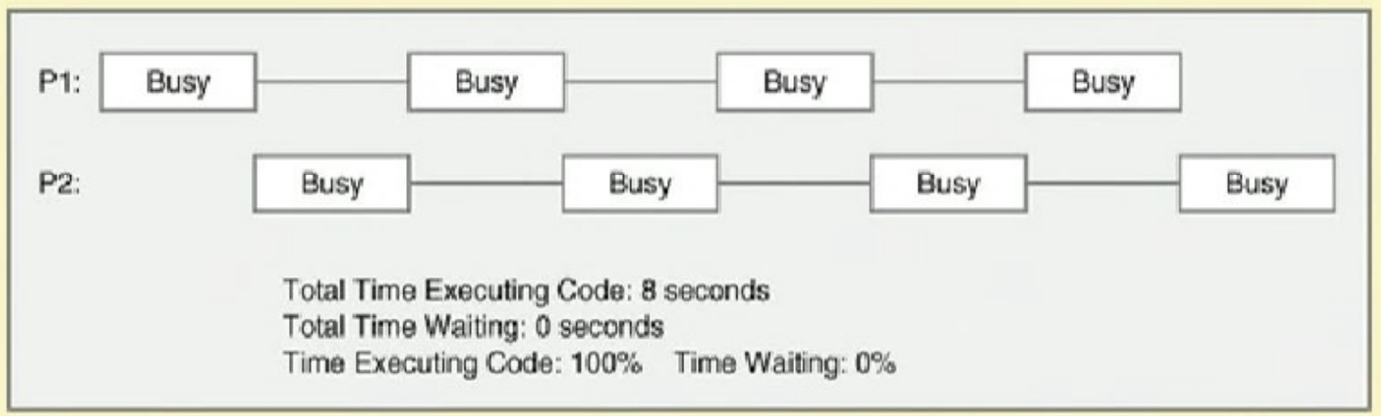


Multitasking can improve performance

Single task →

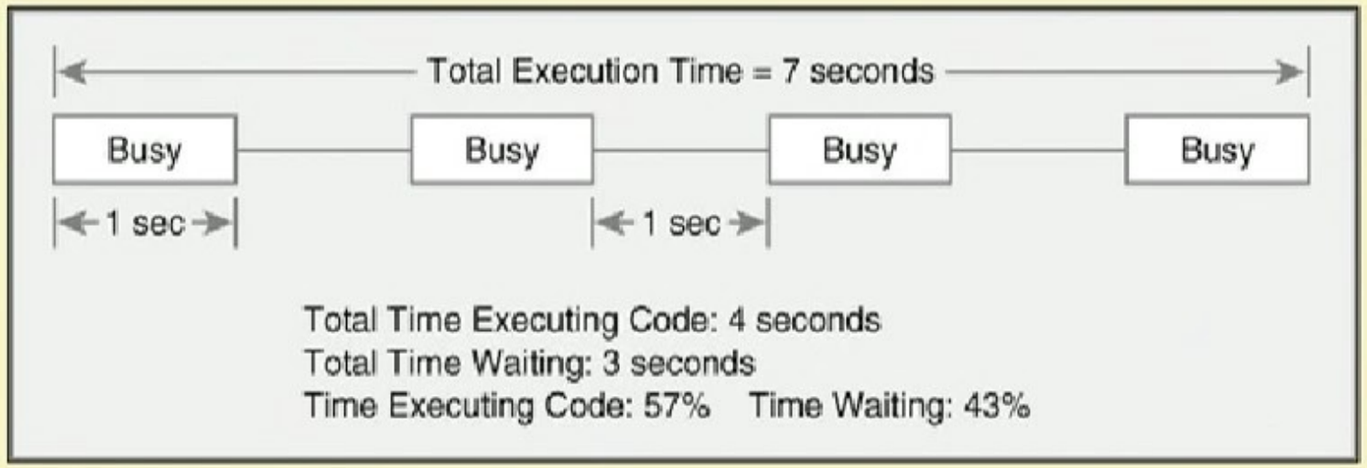


Two tasks →

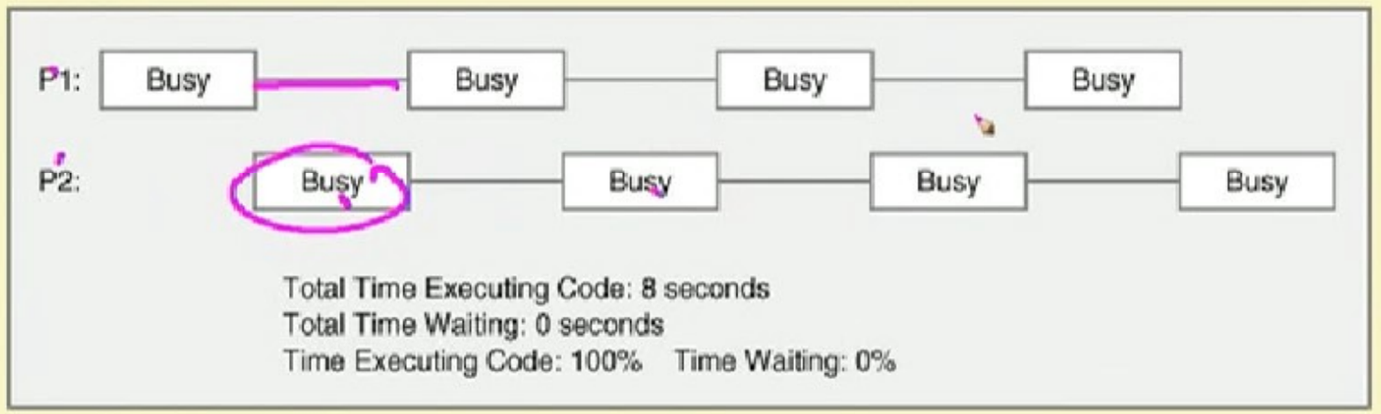


Multitasking can improve performance

Single task →



Two tasks →



Multiprocessing (multi-threading)

- Multiple processing units (multiprocessor).
- Computer works on several tasks in parallel.
- Performance can be improved.

Dual-core AMD Athlon X2

32 processor Pentium Xeon

4096 processor Cray X1

Perform multiple tasks using...

Process

- Definition – executable program loaded in memory
- Has own address space : Variables and data structures (in memory)
- Each process may execute a different program
- Communicate via operating system, files, network
- May contain multiple threads

Thread

- Definition – sequentially executed stream of instructions
- Shares address space with other threads
- Has own execution context : Program counter, call stack (local variables)
- Communicate via shared access to data
- Multiple threads in process execute same program
- Also, known as "*lightweight process*"

Why Multithreading?

Motivation for multithreading

1. Captures logical structure of problem
   - May have concurrent interacting components
   - Can handle each component using separate thread
   - Simplifies programming for problem

Example

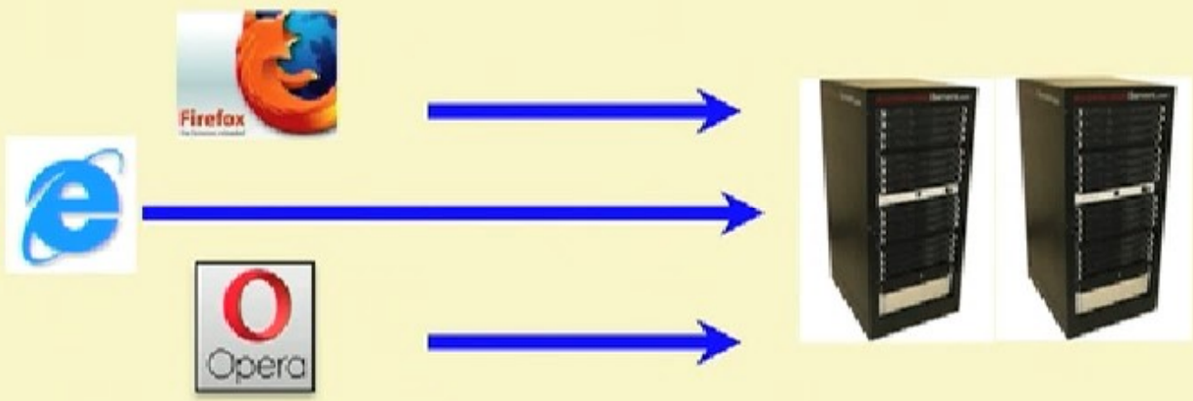
Web server uses threads to handle …

Multiple simultaneous web browser requests

Motivation for multithreading

2. Better utilize hardware resources
   - When a thread is delayed, compute other threads
   - Given extra hardware, compute threads in parallel
   - Reduce overall execution time

Example

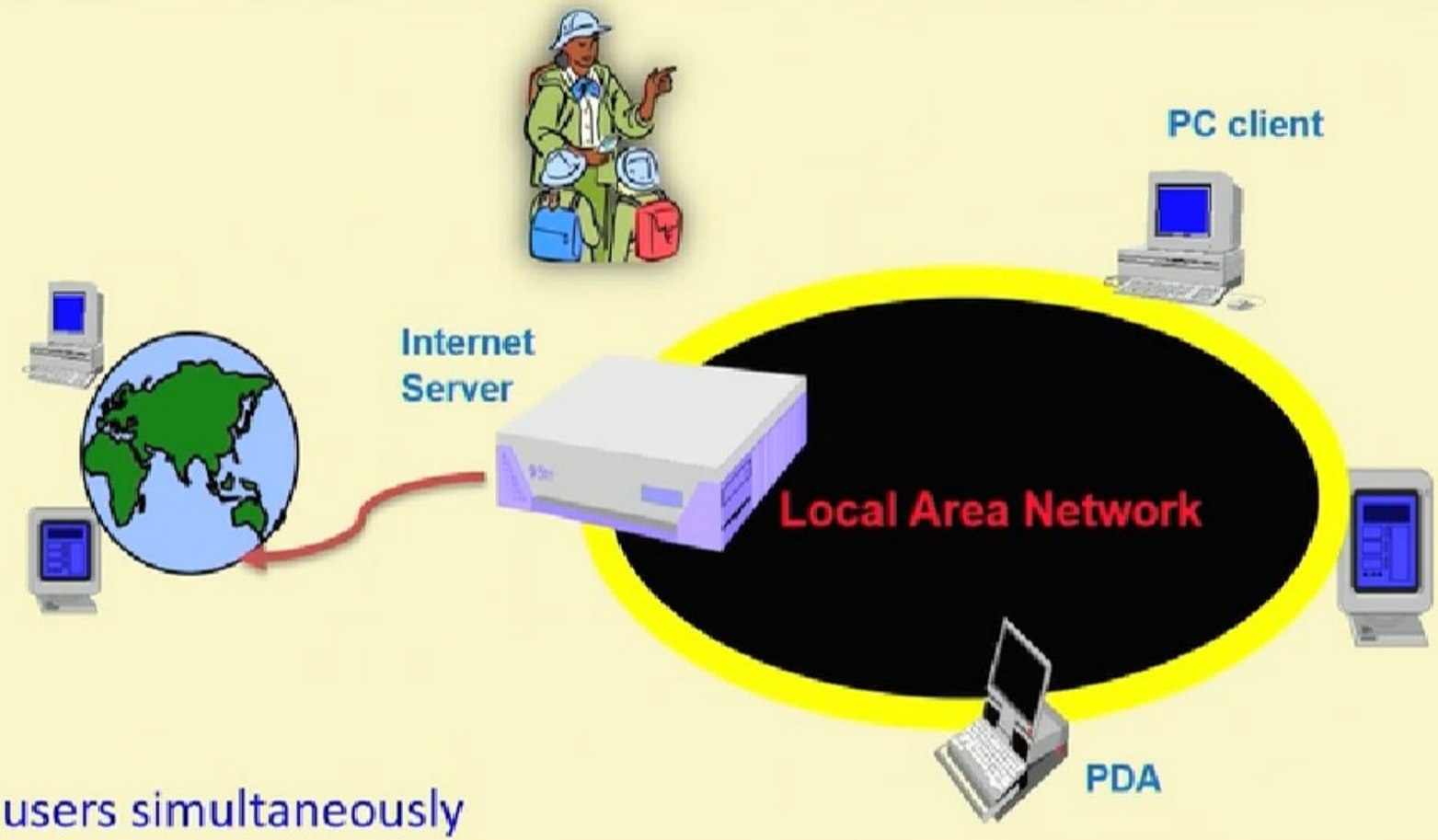


Motivation for multithreading

2. Better utilize hardware resources
   - When a thread is delayed, compute other threads
   - Given extra hardware, compute threads in parallel
   - Reduce overall execution time

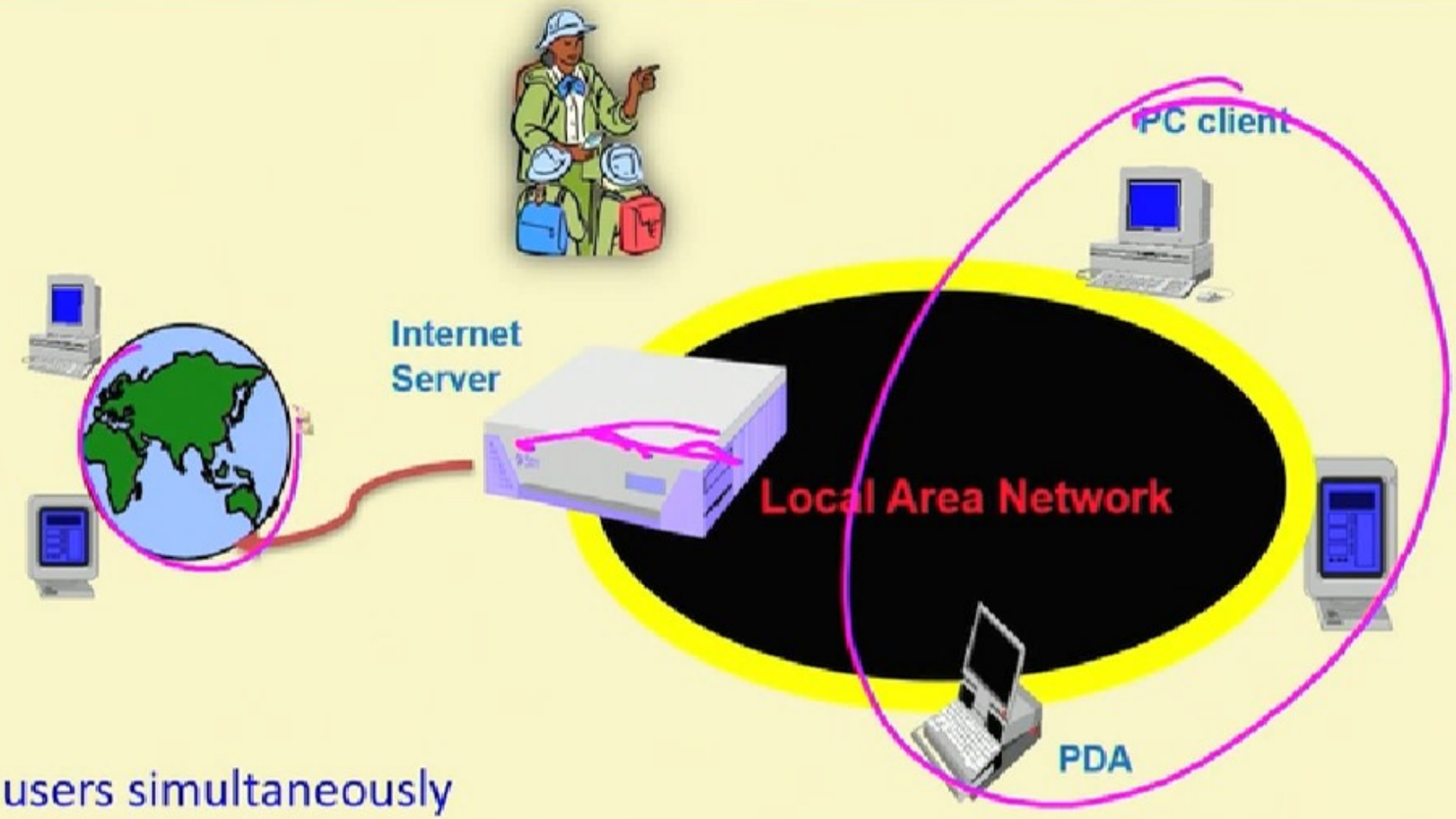
Example

Web/ Internet applications



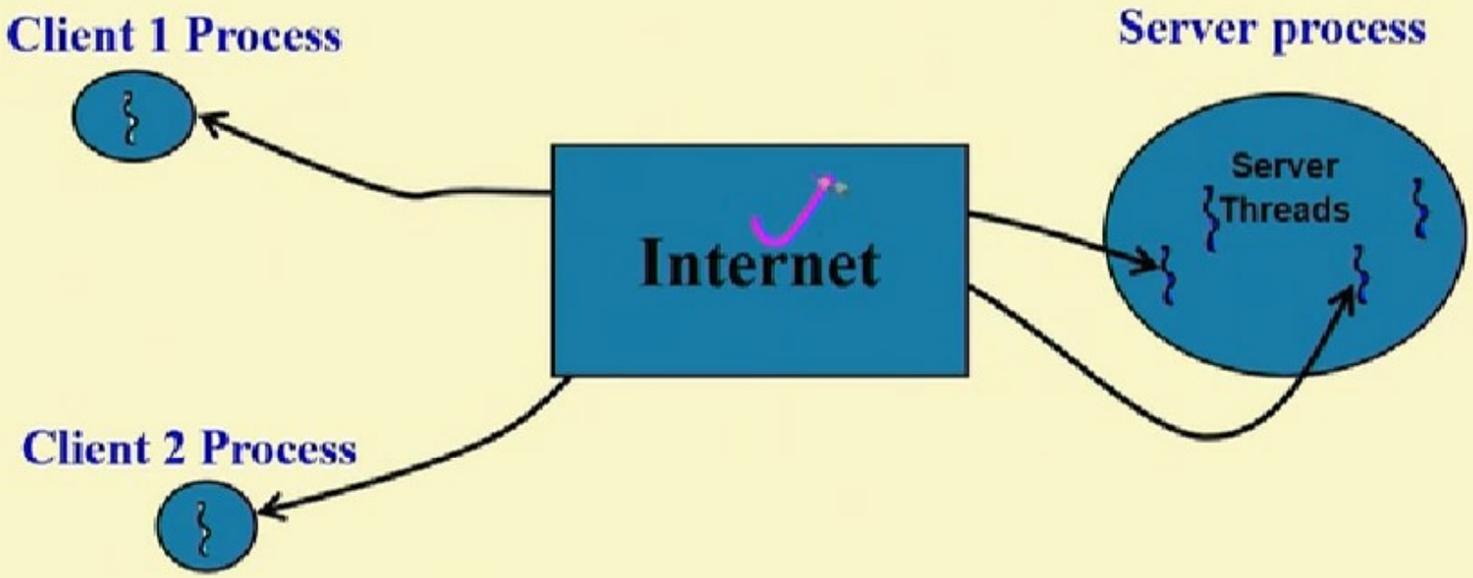
Serving many users simultaneously

Web/ Internet applications



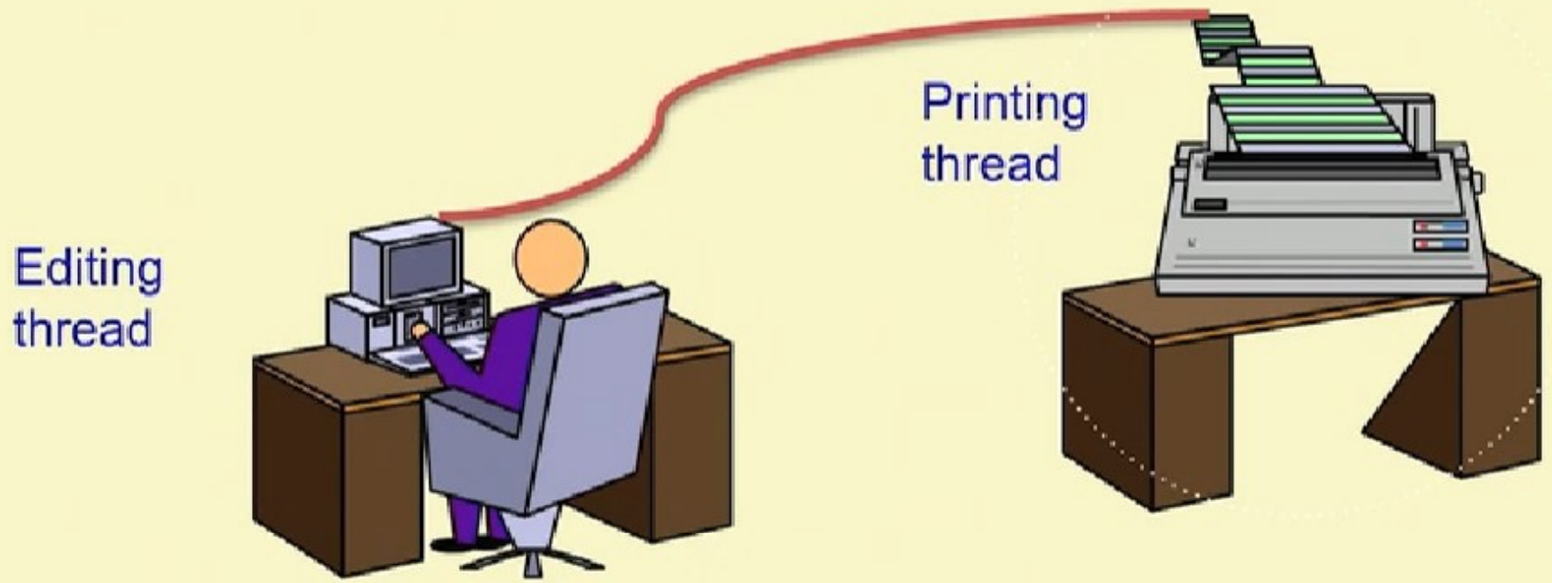
Serving many users simultaneously

Multithreaded server



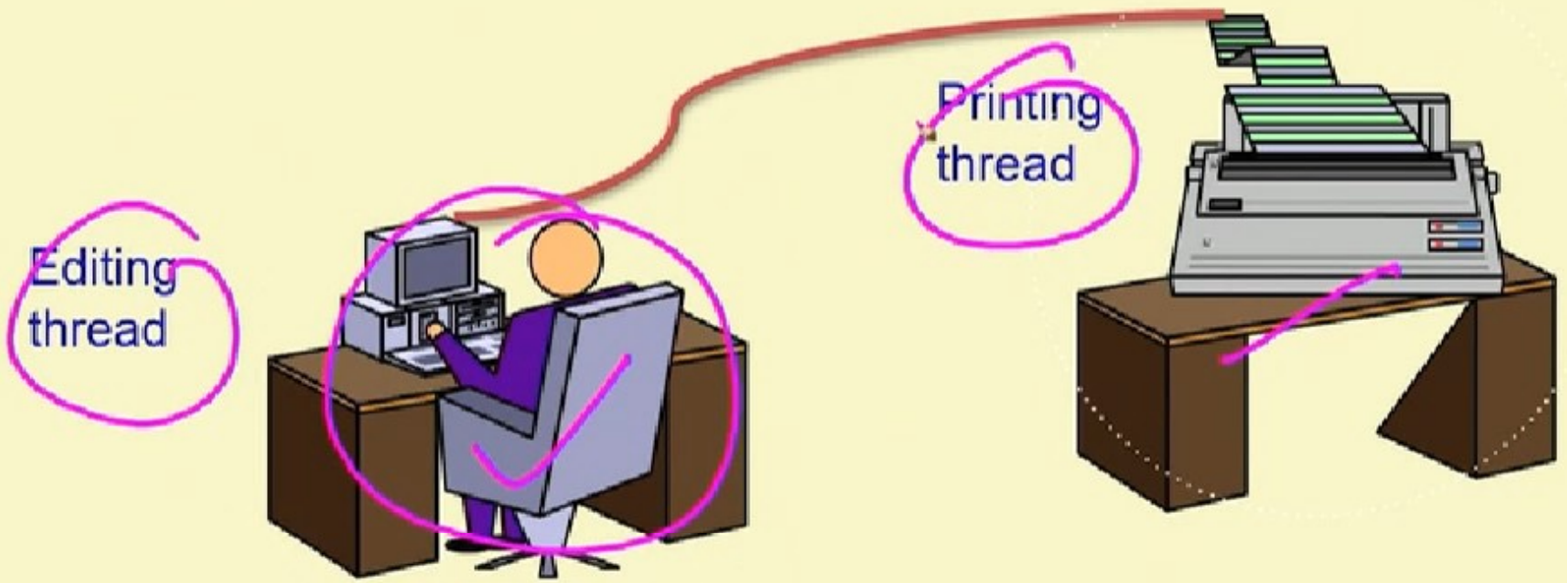
Serving multiple clients concurrently

Modern applications need threads



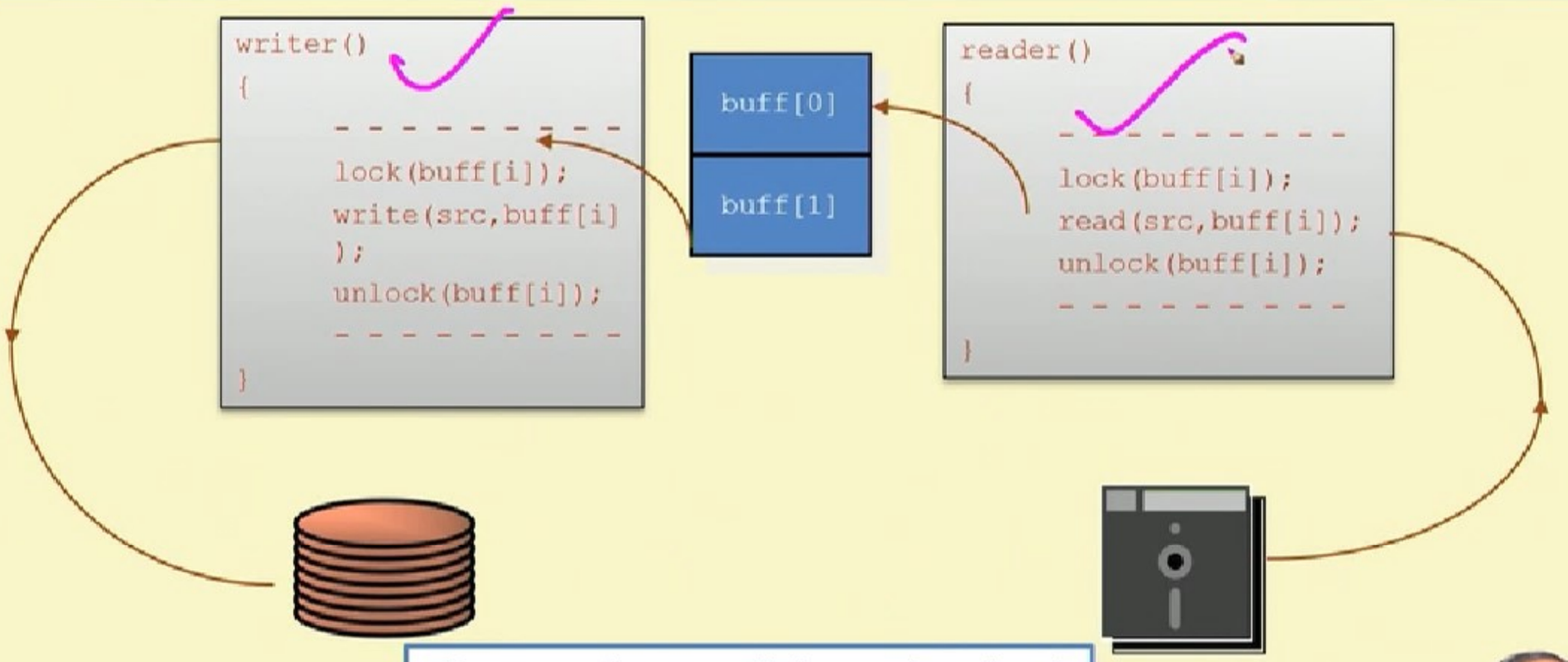
"Editing" and "Printing" documents are in background

Modern applications need threads



"Editing" and "Printing" documents are in background

Multithreaded/ Parallel file copy



Cooperative parallel *synchronized* threads

How Multithreading?

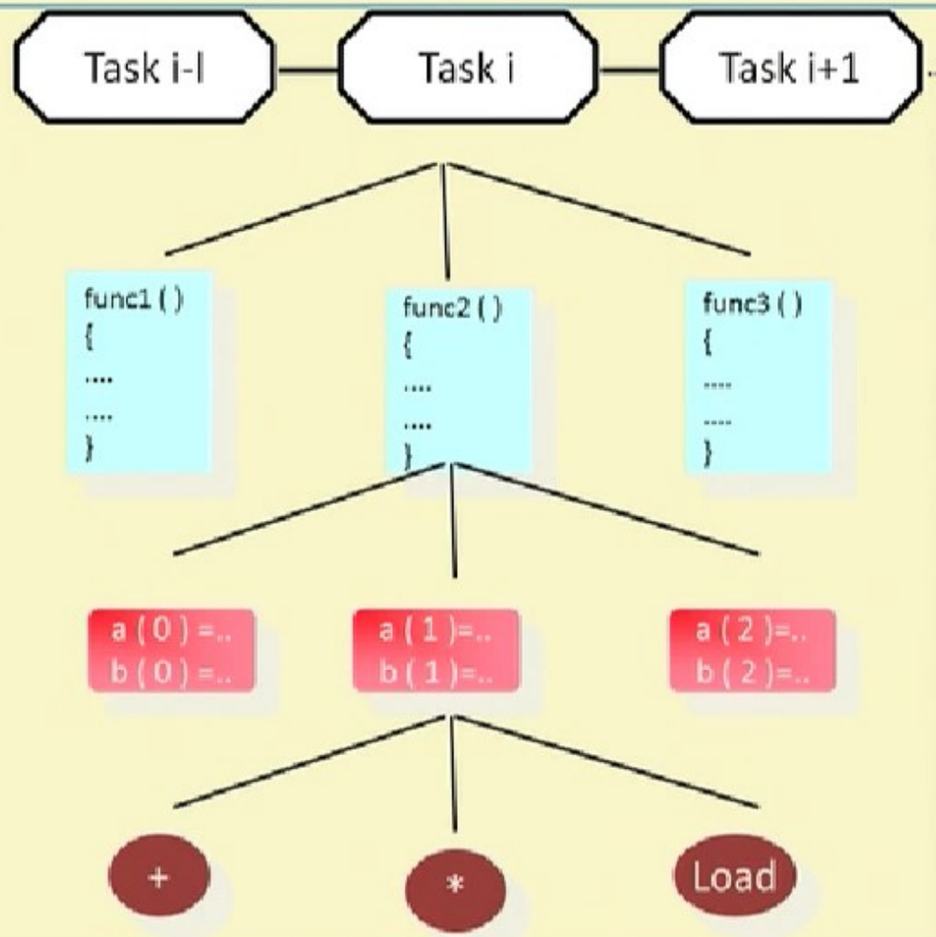
Levels of parallelism

Sockets

Threads

Compilers

CPU



Code-granularity

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

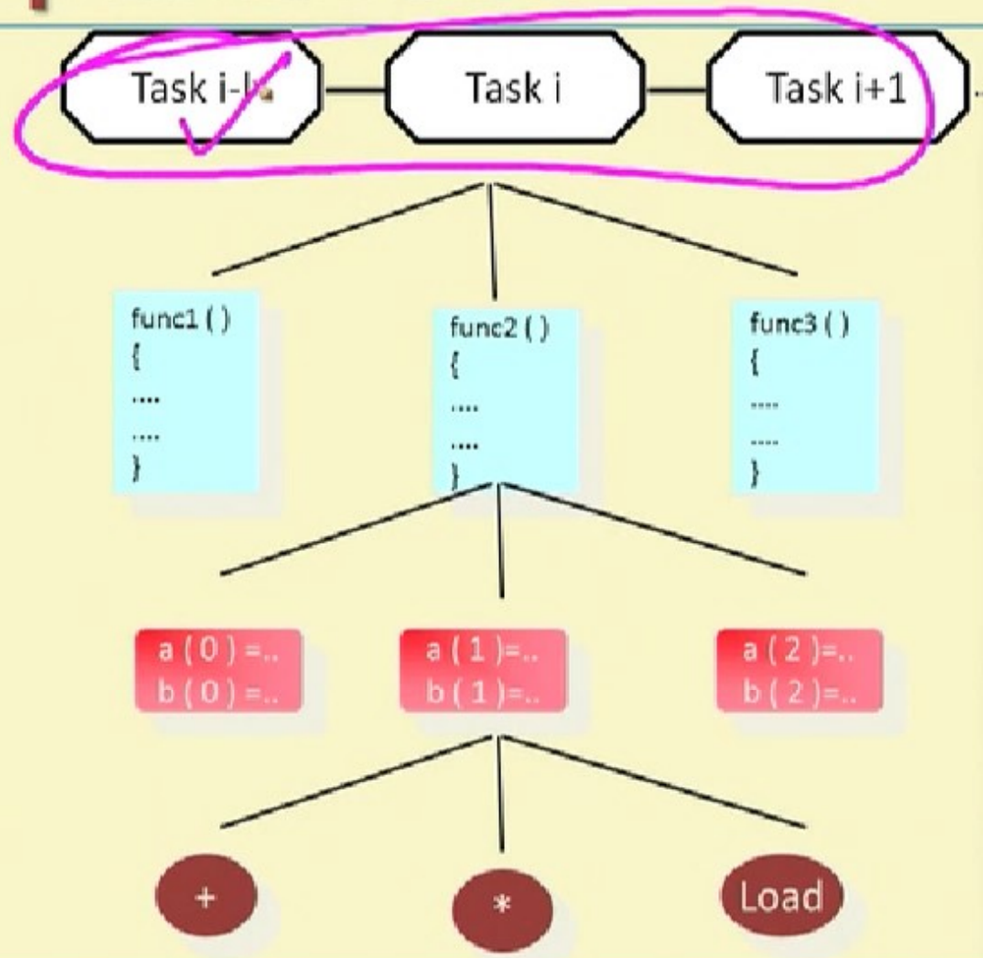
Levels of parallelism

Sockets

Threads

Compilers

CPU



Code-granularity

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

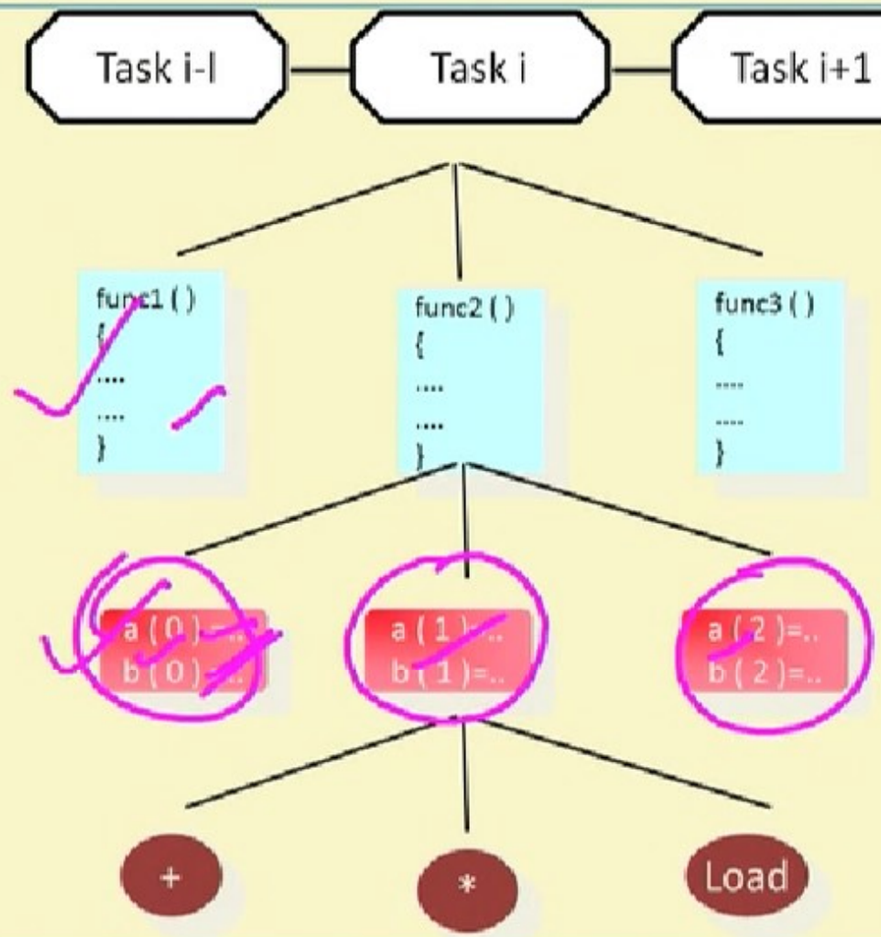
Levels of parallelism

Sockets

Threads

Compilers

CPU



Code-granularity
Code Item
Large grain
(task level)
Program

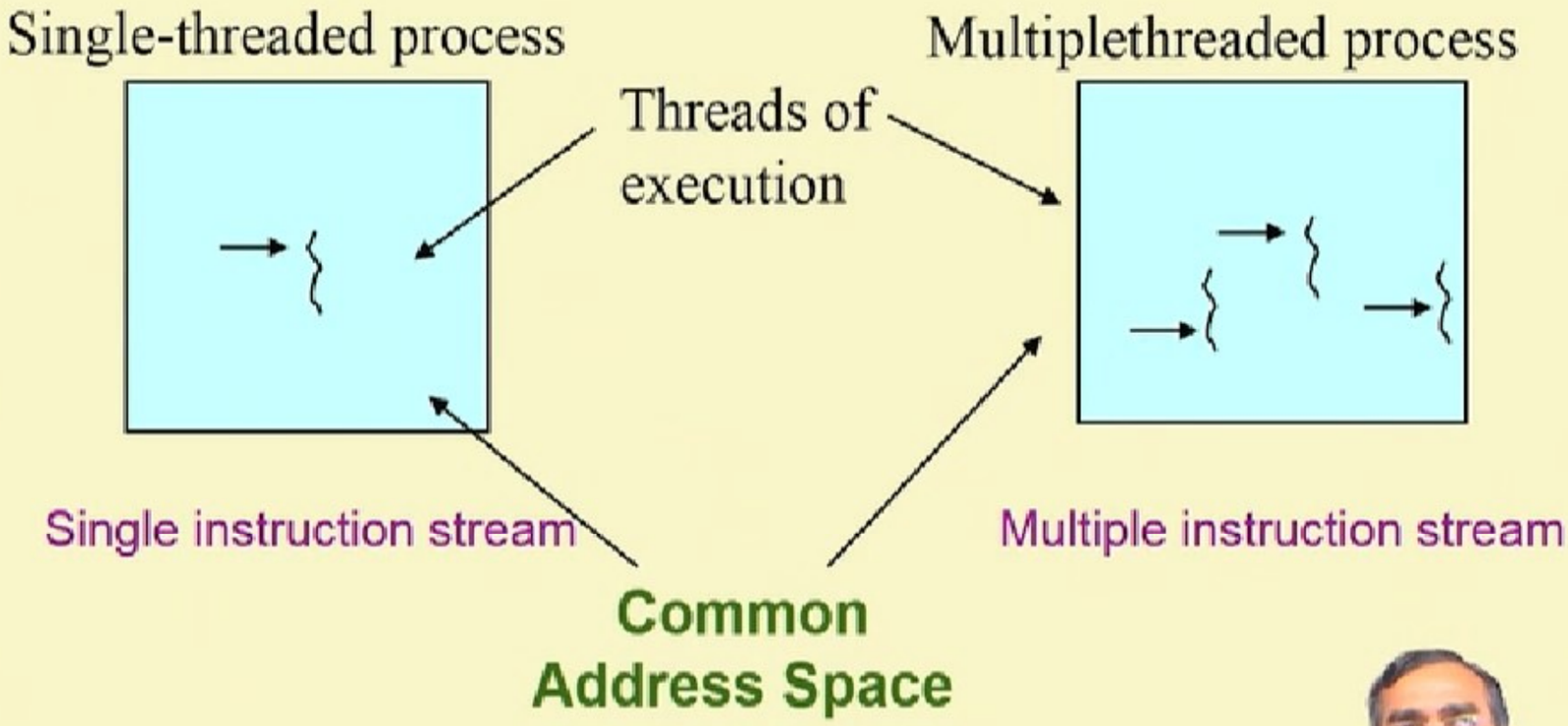
Medium grain
(control level)
Function (thread)

Fine grain
(data level)
Loop (Compiler)

Very fine grain
(multiple issue)
With hardware

Single and multithreaded processes

Threads are light-weight processes within a process



A thread is …

- A piece of code that runs concurrently with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are being extensively used to express concurrency on both single and multiprocessor machines.
- Programming a task having multiple threads of control
  - Multithreading or multithreaded programming.

Multithreading in Java

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

DEBASIS SAMANTA

IIT KHARAGPUR

Java threads

- Java has built in 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.

Java threads

- Java has built in support for multithreading.
  - Synchronization
  - Thread scheduling
  - Inter-thread communication:

currentThread	start	setPriority
yield	run	getPriority
sleep	stop	suspend
resume		

Everything about thread is readily defined in the package *java.lang* and in a class Thread and interface Runnable in it.

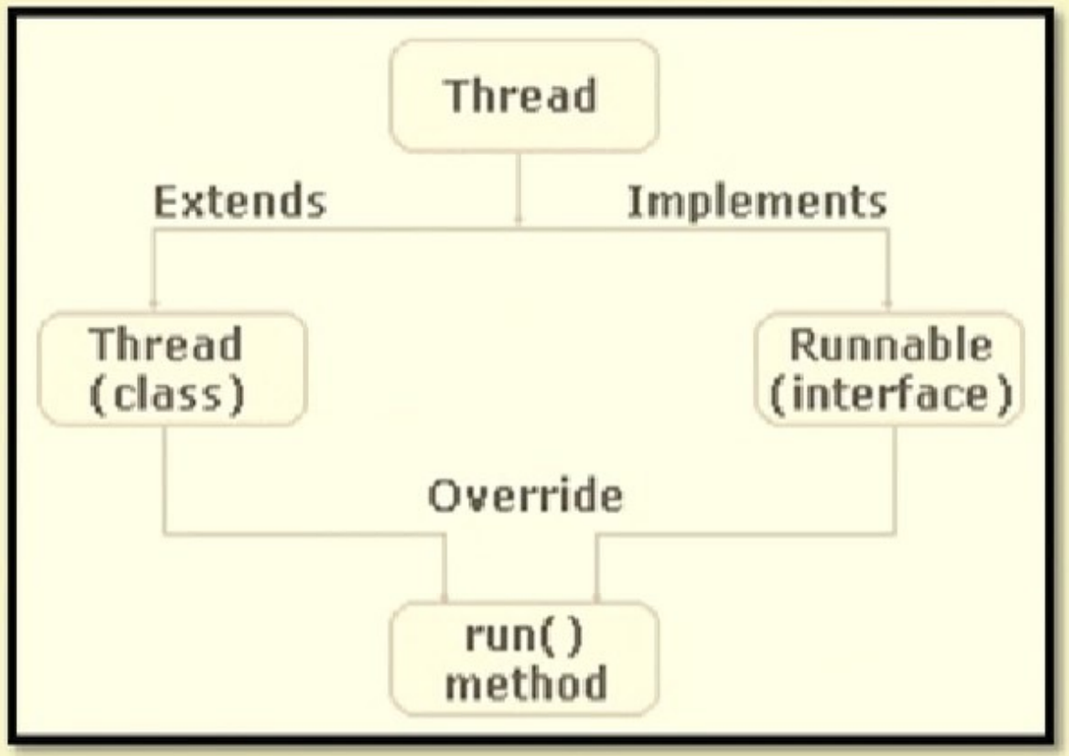
Running a thread in Java

Note:

- Thread starts executing only if start() is called

- Runnable is interface
  - So it can be multiply inherited
  - Required for multithreading in applets

Running a thread in Java



Creating Threads with Thread

Creating threads in Java

There are two ways to create and run a thread:

- Thread class

```
public class Thread extends Object { …
}
```

- Runnable interface

```
public interface Runnable
 {
        public void run( ); //work⇒ thread
 }
```

Thread class

```
public class Thread extends Object implements Runnable {
    public Thread();
    public Thread (String name); //Thread name
    public Thread (Runnable R);  //Thread R.run()
    public Thread (Runnable R, String name);
    public void run();     //if no R, work for thread
    public void start();  //begin thread execution
    ...
}
```

Thread class

```
public class Thread extends Object implements Runnable {
    public Thread();
    public Thread (String name); //Thread name
    public Thread (Runnable R);  //Thread R.run()
    public Thread (Runnable R, String name);
    public void run();     //if no R, work for thread
    public void start();   //begin thread execution
    ...
}
```

More Thread class methods

```
public class Thread extends Object {
        …
    public static Thread currentThread();
    public String getName();
    public void interrupt();
    public boolean isAlive();
    public void join();
    public void setDaemon();
    public void setName();
    public void setPriority();
    public static void sleep();
    public static void yield();
}
```

Creating thread in Java programs

1. Thread class
   - Extend Thread class and override the run method

Example:

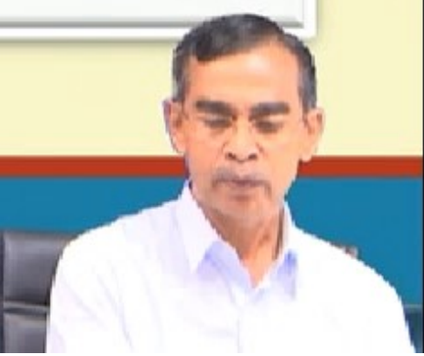
```
public class MyT extends Thread {
    public void run() {
            …                   // work for thread
    }
}
MyT T = new MyT () ;            // create thread
T.start();                      // begin running thread
…                               // thread executing in parallel
```

Creating thread in Java programs

1. Thread class
   - Extend Thread class and override the run method

Example:

```
public class MyT extends Thread {
    public void run() {
        …                    // work for thread
    }
}
MyT T = new MyT () ;         // create thread
T.start();                   // begin running thread
…                            // thread executing in parallel
```



Creating thread : An example

```
class ThreadA extends Thread{
     public void run( ) {
        for(int i = 1; i <= 5; i++) {
           System.out.println("From Thread A with i = "+ -1*i);
        }
        System.out.println("Exiting from Thread A ...");
     }
}
class ThreadB extends Thread {
     public void run( ) {
        for(int j = 1; j <= 5; j++) {
          System.out.println("From Thread B with j= "+2* j);
        }
        System.out.println("Exiting from Thread B ...");
}
```

Creating thread : An example

```
class ThreadA extends Thread
{
     public void run( ) {
        for(int i = 1; i <= 5; i++) {
           System.out.println("From Thread A with i = "+ -1*i);
        }
       System.out.println("Exiting from Thread A ...");
     }
}
class ThreadB extends Thread
{
     public void run( ) {
         for(int j = 1; j <= 5; j++) {
             System.out.println("From Thread B with j = "+2* j);
         }
        System.out.println("Exiting from Thread B ...");     }
```

```
class ThreadC extends Thread
{
     public void run( ) {
         for(int k = 1; k <= 5; k++) {
             System.out.println("From Thread C with k = "+ 2*k-1);
         }
         System.out.println("Exiting from Thread C ...");
     }
}
class MultiThreadClass
{
    public static void main(String args[]) {
         ThreadA a = new ThreadA();
         ThreadB b = new ThreadB();
         ThreadC c = new ThreadC();
         a.start();
         b.start();
         c.start();
         System.out.println("... Multithreading is over ");
    }
}
```

Creating thread : An example

```
class ThreadA extends Thread
{
    public void run( ) {
       for(int i = 1; i <= 5; i++) {
          System.out.println("From Thread A with i = "+ -1*i);
       }
      System.out.println("Exiting from Thread A ...");
    }
}
class ThreadB extends Thread
{
    public void run( ) {
        for(int j = 1; j <= 5; j++) {
            System.out.println("From Thread B with j = "+2* j);
        }
       System.out.println("Exiting from Thread B ...");     }
```

```
class ThreadC extends Thread
{
     public void run( ) {
          for(int k = 1; k <= 5; k++) {
              System.out.println("From Thread C with k = "+ 2*k-1);
          }
          System.out.println("Exiting from Thread C ...");
     }
}
class MultiThreadClass
{
    public static void main(String args[]) {
          ThreadA a = new ThreadA();
          ThreadB b = new ThreadB();
          ThreadC c = new ThreadC();
          a.start();
          b.start();
          c.start();
          System.out.println("... Multithreading is over ");
    }
}
```

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

OBJECT ORIENTED PROGRAMMING WITH JAVA

Multithreaded Programming in Java – II

Debasis Samanta
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Creating Threads with Runnable

Creating thread in Java

2. Runnable interface
   - Create object implementing Runnable interface
   - Pass it to Thread object via Thread constructor

Example

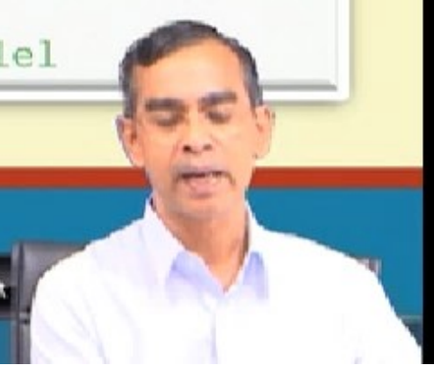
```
public class MyT implements Runnable {
    public void run() {
        …                                   // work for thread
    }
}
Thread T = new Thread(new MyT);             // create thread
T.start();                                  // begin running thread
…                                           // thread executing in parallel
```

Creating thread in Java

2. Runnable interface
   - Create object implementing Runnable interface
   - Pass it to Thread object via Thread constructor

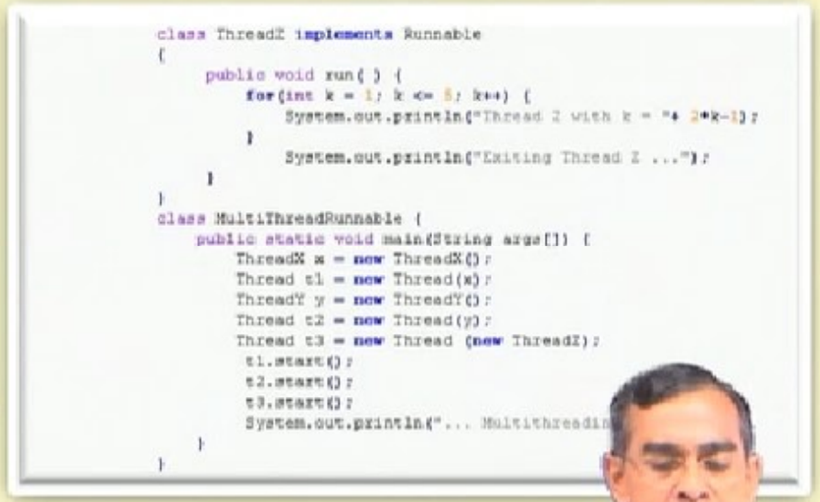
Example

```
public class MyT implements Runnable {
    public void run() {
        …                                   // work for thread
    }
}
Thread T = new Thread(new MyT());           // create thread
T.start();                                  // begin running thread
…                                           // thread executing in parallel
```



Creating thread : Example

```
class ThreadX implements Runnable
{
     public void run( ) {
        for(int i = 1; i <= 5; i++) {
           System.out.println("Thread X with i = "+ -1*i);
        }
       System.out.println("Exiting Thread X ...");
     }
}
class ThreadY implements Runnable {
     public void run( ) {
         for(int j = 1; j <= 5; j++) {
             System.out.println("Thread Y with j = "+ 2*j);
         }
        System.out.println("Exiting Thread Y ...");
     }
}
```



Creating thread : Example

```
class ThreadX implements Runnable
{
    public void run( ) {
       for(int i = 1; i <= 5; i++) {
          System.out.println("Thread X with i = "+ -1*i);
       }
      System.out.println("Exiting Thread X ...");
    }
}
class ThreadY implements Runnable {
    public void run( ) {
        for(int j = 1; j <= 5; j++) {
            System.out.println("Thread Y with j = "+ 2*j);
        }
       System.out.println("Exiting Thread Y ...");
    }}
```

```
class ThreadZ implements Runnable
{
    public void run( ) {
        for(int k = 1; k <= 5; k++) {
            System.out.println("Thread Z with k = "+ 2*k-1);
        }
            System.out.println("Exiting Thread Z ...");
    }
}
class MultiThreadRunnable {
    public static void main(String args[]) {
        ThreadX x = new ThreadX();
        Thread t1 = new Thread(x);
        ThreadY y = new ThreadY();
        Thread t2 = new Thread(y);
        Thread t3 = new Thread (new ThreadZ);
         t1.start();
         t2.start();
         t3.start();
         System.out.println("... Multithreading is over ");
    }
}
```

Creating thread : Example

```
class ThreadX implements Runnable
{
    public void run( ) {
        for(int i = 1; i <= 5; i++) {
            System.out.println("Thread X with i = "+ -1*i);
        }
        System.out.println("Exiting Thread X ...");
    }
}
class ThreadY implements Runnable {
    public void run( ) {
        for(int j = 1; j <= 5; j++) {
            System.out.println("Thread Y with j = "+ 2*j);
        }
        System.out.println("Exiting Thread Y ...");
    }
```

```
class ThreadZ implements Runnable
{
    public void run( ) {
        for(int k = 1; k <= 5; k++) {
            System.out.println("Thread Z with k = "+ 2*k-1);
        }
            System.out.println("Exiting Thread Z ...");
    }
}
class MultiThreadRunnable {
    public static void main(String args[]) {
        ThreadX x = new ThreadX();
        Thread t1 = new Thread(x);
        ThreadY y = new ThreadY();
        Thread t2 = new Thread(y);
        Thread t3 = new Thread (new ThreadZ);
        t1.start();
        t2.start();
        t3.start();
        System.out.println("... Multithreading is over ");
    }
}
```

Creating thread : Example

```
class ThreadX implements Runnable
{
    public void run( ) {
       for(int i = 1; i <= 5; i++) {
          System.out.println("Thread X with i = "+ -1*i);
       }
      System.out.println("Exiting Thread X ...");
    }
}
class ThreadY implements Runnable {
    public void run( ) {
        for(int j = 1; j <= 5; j++) {
            System.out.println("Thread Y with j = "+ 2*j);
        }
       System.out.println("Exiting Thread Y ...");
    }}
```

```
class ThreadZ implements Runnable
{
    public void run( ) {
        for(int k = 1; k <= 5; k++) {
            System.out.println("Thread Z with k = "+ 2*k-1);
        }
            System.out.println("Exiting Thread Z ...");
    }
}
class MultiThreadRunnable {
    public static void main(String args[]) {
        ThreadX x = new ThreadX();
        Thread t1 = new Thread(x);
        ThreadY y = new ThreadY();
        Thread t2 = new Thread(y);
        Thread t3 = new Thread (new ThreadZ);
        t1.start();
        t2.start();
        t3.start();
        System.out.println("... Multithreading is over ");
    }
}
```

States of a Thread

Threads : Thread states of a thread

Java thread can be in one of these states :

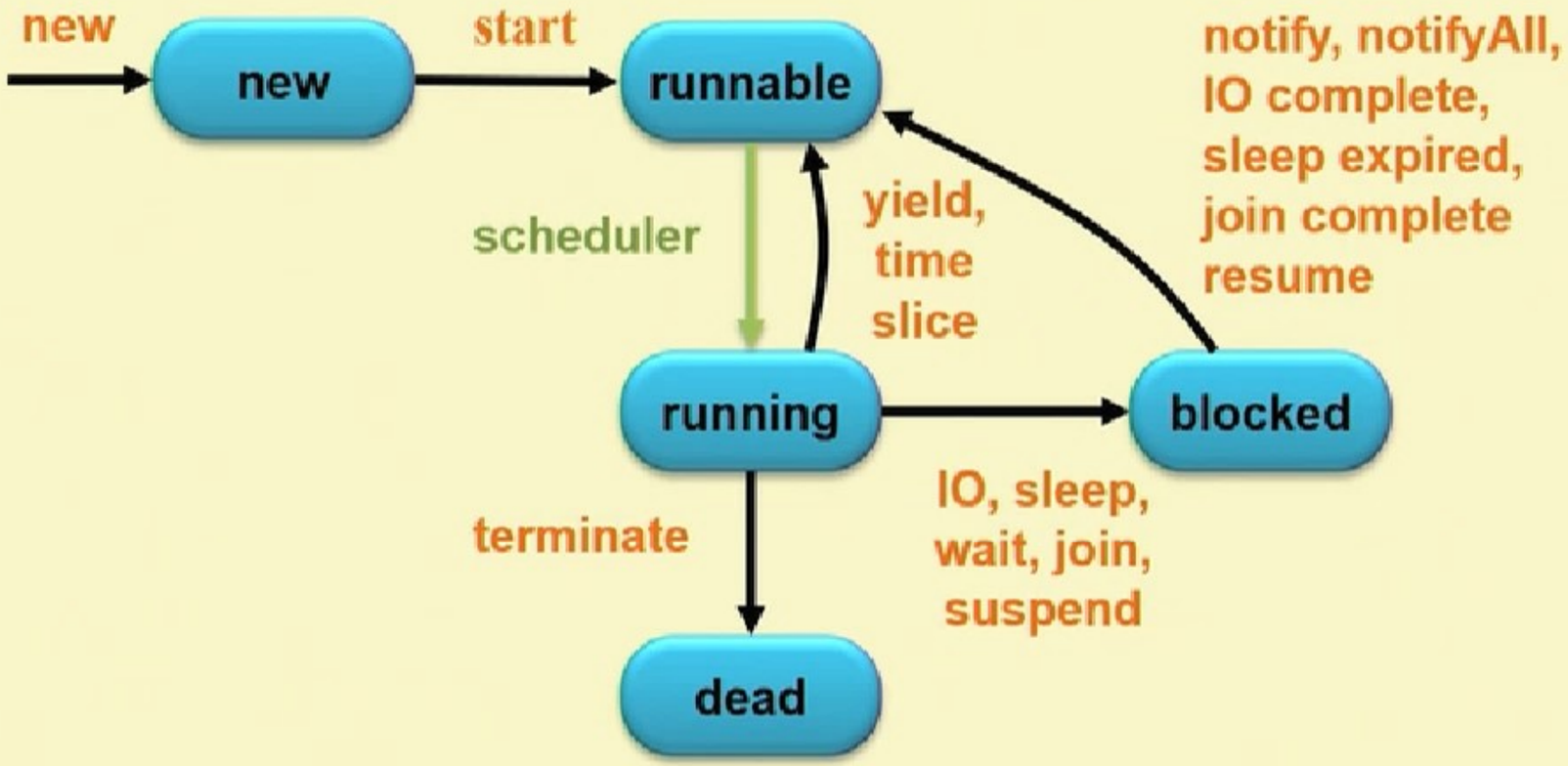
- New – thread allocated & waiting for start()
- Runnable – thread can begin execution
- Running – thread currently executing
- Blocked – thread waiting for event (I/O, etc.)
- Dead – thread finished

Transitions between states caused by

- Invoking methods in class Thread
  - new(), start(), yield(), sleep(), wait(), notify()…
- Other (external) events
  - Scheduler, I/O, returning from run()…

Thread States

State diagram



Thread control methods

start () :→ A newborn thread with this method enter into Runnable state and Java run time create a system thread context and starts it running. This method for a thread object can be called once only

suspend() :→ This method is different from stop( ) method. It takes the thread and causes it to stop running and later on can be restored (by resume())

resume() :→ This method is used to revive a suspended thread. There is no gurantee that the thread will start running right way, since there might be a higher priority thread running already, but, resume() causes the thread to become eligible for running

sleep(int n):→ This method causes the run time to put the current thread to sleep for n milliseconds

yield() :→ This method causes the run time to switch the context from the current thread to the next available runnable thread. This is one way to ensure that the threads at lower priority do not get started

Scheduling of Threads

Threads : Scheduling

Scheduler

- Determines which runnable threads to run.
- Can be based on thread priority.
- Part of OS or Java Virtual Machine (JVM) .

Scheduling policy

- Nonpreemptive (cooperative) scheduling.
- Preemptive scheduling.

Daemon threads

Java threads types

- User
- Daemon
  - Provide general services.
  - Typically never terminate.
  - Call setDaemon() before start().

Program termination

- All user threads finish.
- Daemon threads are terminated by JVM.
- Main program finishes.

Threads : Scheduling

Scheduler

- Determines which runnable threads to run.
- Can be based on thread priority.
- Part of OS or Java Virtual Machine (JVM) .

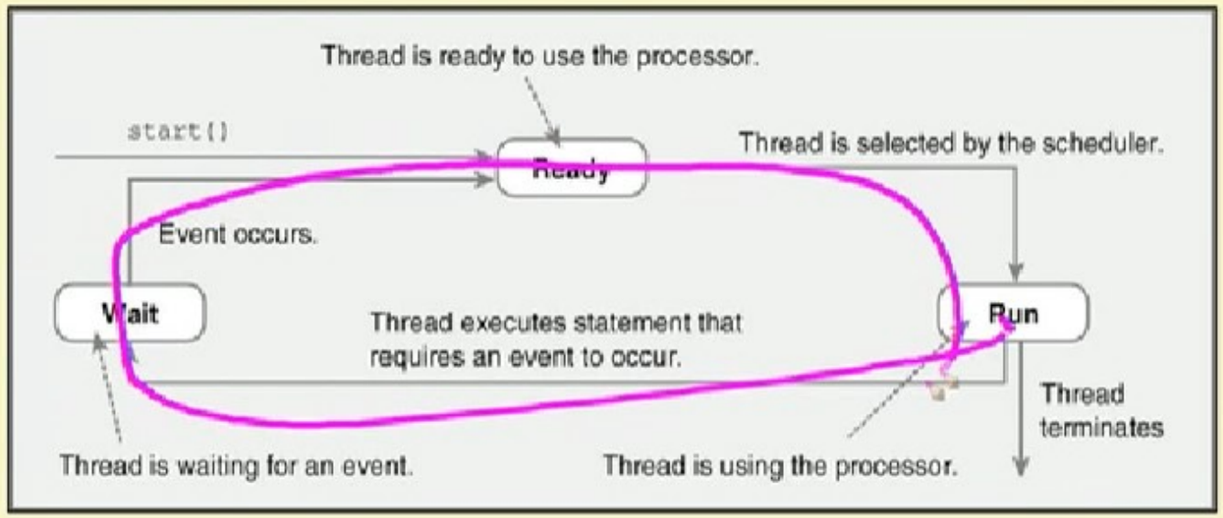
Scheduling policy

- Nonpreemptive (cooperative) scheduling.
- Preemptive scheduling.

Threads: Non-preemptive scheduling

Threads continue execution until

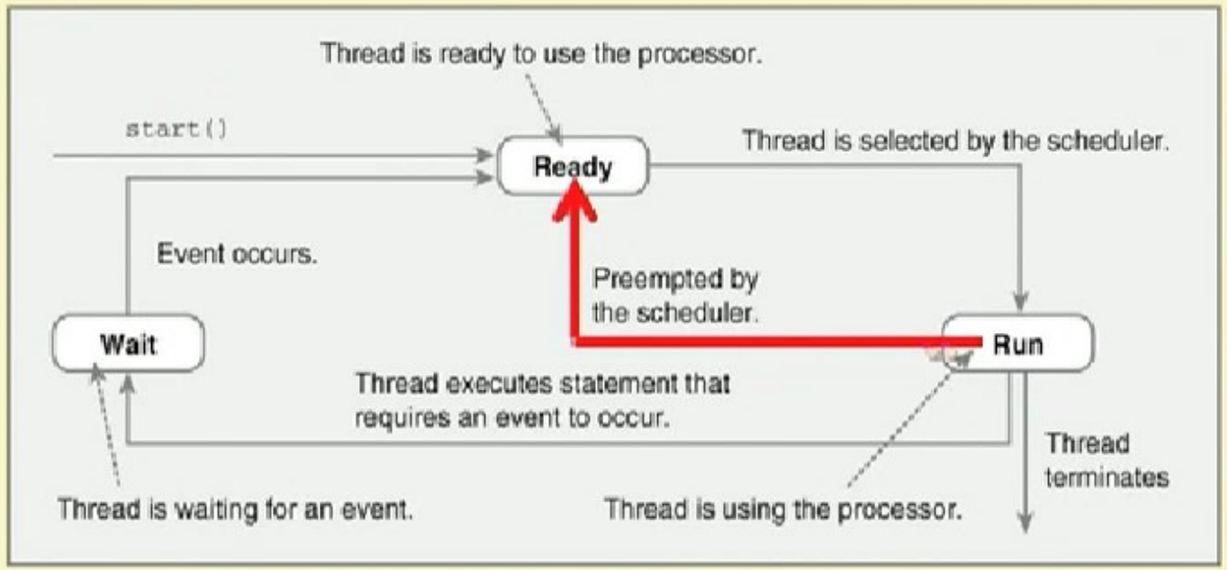
- Thread terminates.
- Executes instruction causing wait (e.g., IO).
- Thread volunteering to stop (invoking yield or sleep).



Threads: Preemptive scheduling

Threads continue execution until

- Same reasons as non-preemptive scheduling.
- Preempted by scheduler.



Java thread : An example

```
public class ThreadExample extends Thread {
    public void run() {
        for (int i = 0; i < 3; i++) {
            try {
                sleep ((int)(Math.random() * 5000));  // 5 secs
            }
             catch (InterruptedException e) {
                 System.out.println (i);
             }
    }
    public static void main(String[ ] args) {
        new ThreadExample().start();
        new ThreadExample().start();
        System.out.println ("Done");
    }
}
```

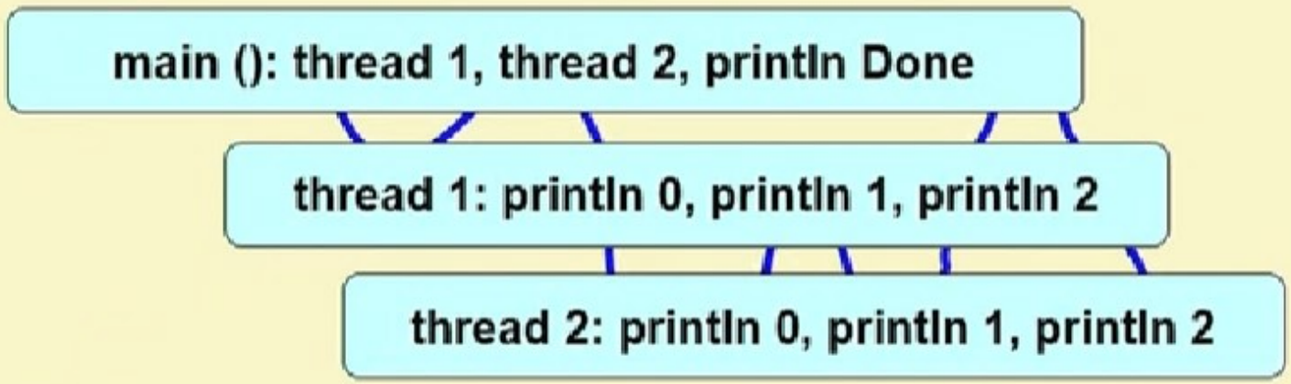
Java thread : An example

```
public class ThreadExample extends Thread {
    public void run() {
        for (int i = 0; i < 3; i++) {
            try {
                sleep ((int)(Math.random() * 5000));  // 5 secs
            }
             catch (InterruptedException e) {
                 System.out.println (i);
             }
    }
    public static void main(String[ ] args) {
        new ThreadExample().start();
        new ThreadExample().start();
        System.out.println ("Done");
    }
}
```

Java Thread Example

Possible outputs

- 0,1,2,0,1,2,Done // thread 1, thread 2, main()
- 0,1,2,Done,0,1,2 // thread 1, main(), thread 2
- Done,0,1,2,0,1,2 // main(), thread 1, thread 2
- 0,0,1,1,2,Done,2 // main() & threads interleaved



Data races

```
public class DataRace extends Thread {
    static int x;
    public void run() {
        for (int i = 0; i < 100000; i++) {
            x = x + 1;
            x = x - 1;
        }
    }
    public static void main(String[] args) {
        x = 0;
        for (int i = 0; i < 100000; i++)
            new DataRace().start();
        System.out.println(x); // x not always 0!
    }
}
```

Data races

```
public class DataRace extends Thread {
    static int x;
    public void run() {
        for (int i = 0; i < 100000; i++) {
            x = x + 1;
            x = x - 1;
        }
    }
    public static void main(String[] args) {
        x = 0;
        for (int i = 0; i < 100000; i++)
            new DataRace().start();
        System.out.println(x); // x not always 0!
    }
}
```

Thread scheduling observations

The order in which threads are selected for execution is indeterminate.	Thread can block indefinitely (starvation).	Thread scheduling may cause data races.	Synchronization
• Depends on scheduler.	• If other threads always execute first.	• Modifying same data from multiple threads. • Result depends on thread execution order.	• Control thread execution order. • Eliminate data races.

Priority of Threads

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 (NORM_PRIORITY) and they are served using FCFS policy.
  - Java allows users to change priority:

    ThreadName.setPriority (int Number)

    - MIN_PRIORITY = 1
    - NORM_PRIORITY = 5
    - MAX_PRIORITY = 10

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 (NORM_PRIORITY) and they are served using FCFS policy.
  - Java allows users to change priority:

    ThreadName.setPriority (int Number)

    - MIN_PRIORITY = 1
    - NORM_PRIORITY = 5
    - MAX_PRIORITY = 10

Thread priority : An 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");
          }

}
```

Thread priority : An 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 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 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 : An 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");
          }
}
```

Thread priority : An example

```
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");
         }
}
```

Thread priority : An example

```
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");
         }
}
```

Thread priority : An example

```
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");
         }
}
```

Thread priority : An example

```
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");
        }
}
```

Thread class : Join

```
public class Test1 {
    static void main(String[] args){
            Thread t1 = new Thread(new R(1));
            Thread t2 = new Thread(new R(2));
            t1.start();
            t2.start();
            try {
                    t1.join();      // waits until t1 has terminated
                    t2.join();      // waits until t2 has terminated
            }
            catch(InterruptedException e){ }
            System.out.println("done");
    }
}
```

Thread class : Join

```
public class Test1 {
    static void main(String[] args){
        Thread t1 = new Thread(new R(1));
        Thread t2 = new Thread(new R(2));
        t1.start();
        t2.start();
        try {
            t1.join();      // waits until t1 has terminated
            t2.join();      // waits until t2 has terminated
        }
        catch(InterruptedException e){ }
        System.out.println("done");
    }
}
```

Synchronization of Threads

Thread synchronization

When two or more processes attempts to access a shared resource, it should be synchronized to avoid conflicts.

Java supports methods to be synchronized.

Following is the syntax by which methods can be made to protect from simultaneous access:

```
synchronized (object) { block of statement(s) }
```

Thread synchronization : An example

```
class Account {
      private int balance;
      public int accountNo;
      void displayBalance( ) {
          System.out.println ( "Account No : " + accountNo
      + "Balance : " + balance );
      }
      synchronized  void deposite (int amount ) {
      // Method to deposit an amount
          balance  = balance + amount;
          System.out.print( amount + " is deposited " );
          displayBalance( ) ;
      }
      synchronized void withdraw (int amount ) {
       // method to withdraw an  amount
          balance = balance - amount;
          System.out.print (amount +  "is withdrawn" );
          displayBalance ( );
      }
}
```

```
// To implement a thread for deposit
class TransactionDeposite  implements
Runnable                {
          Account accountX;
          TransactionDeposite (Account x,
          int amount ) {
 // Constructor to initiate this thread
                          accountX = x;
           this.amount   = amount;
           new Thread (this).start ( );
           }
            public void run( )  {
       accountX.deposite (amount);
           }
}
```

Thread synchronization : An example

```
class Account {
      private int balance;
      public int accountNo;
      void displayBalance( ) {
          System.out.println ( "Account No : " + accountNo
      + "Balance : " + balance );
      }
      synchronized void deposite (int amount ) {
      // Method to deposit an amount
          balance  = balance + amount;
          System.out.print( amount + " is deposited " );
          displayBalance( ) ;
      }
      synchronized void withdraw (int amount ) {
       // method to withdraw an  amount
          balance = balance - amount;
          System.out.print (amount +  "is withdrawn" );
          displayBalance ( );
      }
}
```

```
// To implement a thread for deposit
class TransactionDeposite  implements
Runnable                   {
          Account accountX;
          TransactionDeposite (Account x,
          int amount ) {
 // Constructor to initiate this thread
                          accountX = x;
           this.amount   = amount;
           new Thread (this).start ( );
           }
            public void run( )  {
       accountX.deposite (amount);
           }
}
```

Thread synchronization : An example

```
// To implement a thread for withdraw
class TransactionWithdraw implements
Runnable {
     Account accountY;
     int amount;
     TransactionWithdraw (Account y;  int
amount ) {
          accountY = y ;
          this.amount   = amount;
          new Thread (this).start( );
     }
     public void run ( )   {
          accountY.withdraw (amount);
     }
}
```

```
class Transaction {
          public static void main (String,
     args[ ] ) {
     Account ABC = new Account ( );
     // Create an account
     ABC.balance  = 1000;
     // initialize the account by Rs 1000
     TransactionDeposite t1;
     // A thread for deposite
     TransactionWithdraw t2
     // Another thread for withdarw
     t1 = new TransactionDeposite (ABC ,
     500 );
     t2 = new TransactionWithdraw (ABC,
     900);
      // Two threads are started
     }
}
```

Thread synchronization : Stack example

Example: Stack

```
public class Stack {
    private int top = 0;
    private int[] data = new int [10];
    public void push(int x) {
            data[top] = x;
            top++;
    }
    public int pop() {
            top--;
            return data[top];
    }
}
```

Two threads, one is pushing, the other popping objects

Using blocks

Synchronized blocks

- every object contains a single lock
- lock is taken when synchronized section is entered
- if lock is not available, thread enters a waiting queue
- if lock is returned any (longest waiting?) thread is resumed

```
public void push(int x) {
        synchronized(this){
                data[top] = x;
                top++;
        }
}
```

Instance methods

- Often a method is synchronized on "this":

```
public void push(int x) {synchronized(this) {……}}
```

- Short form:

```
public synchronized void push(int x){……}
```

Question to think...

- In any software, Input-Output is a great concern. How Java facilitates I-O handling?
- What makes Java suitable for network programming?

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

OBJECT ORIENTED PROGRAMMING WITH JAVA

Exception handling: Demonstration - X

Debasis Samanta

Department of Computer Science & Engineering

Indian Institute of Technology Kharagpur

In today's demonstration ...

1. About compile-time error
2. About run-time error
3. Simple Try-Catch block
4. Try with multiple Catch
5. Multiple errors with single catch
6. Finally in try-catch block
7. Exception handling using Throws statement
8. Nested try-catch block