# ITSE322 Modern Programming Language: Advanced Java

## Multithreaded Programming using Java Threads

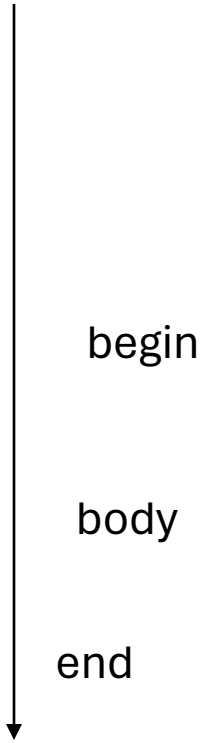
## Lecture 5

# Learning Objectives

- Understand the concept of multithreading
- Create programs with multi-threads
- Accessing Shared Resources
  - Synchronisation
- Understand Advanced Topics:
  - Concurrency Models: master/worker, pipeline, peer processing
  - Multithreading Vs multiprocessing

# Java programs are single threaded

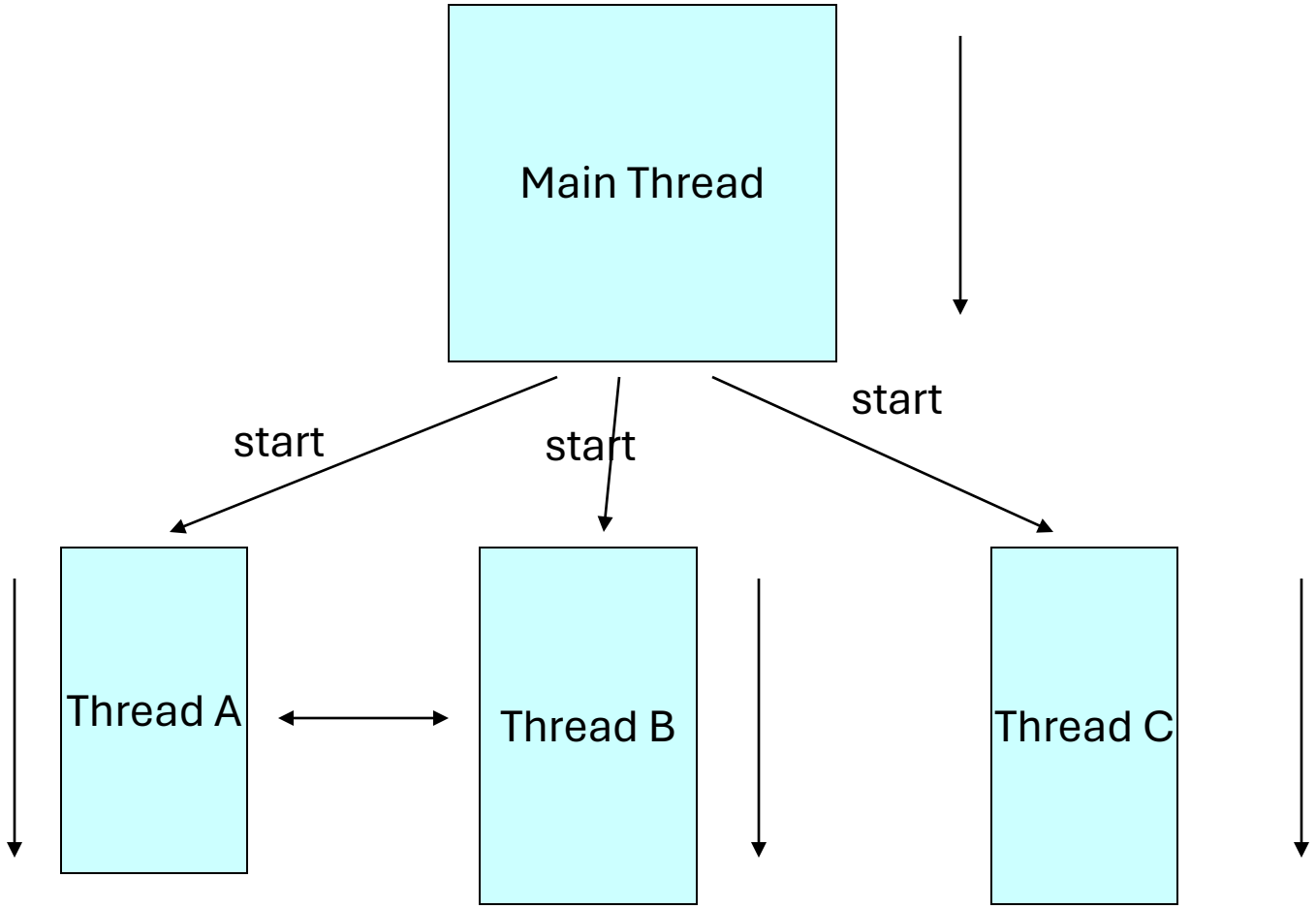
```
class ABC
{
....
    public void main(..)
    {
    ...
    ..
    }
}
```



# What is a Thread?

- A piece of code that runs in concurrent with other threads.
- They are essential for performing background tasks like file downloads, data processing, or network communication without blocking the main program flow.
- Threads allow for better resource utilization by allowing different parts of a program to work independently without waiting for each other.
- They are essential for implementing concurrent data structures and synchronization mechanisms to ensure thread safety and avoid data inconsistency.

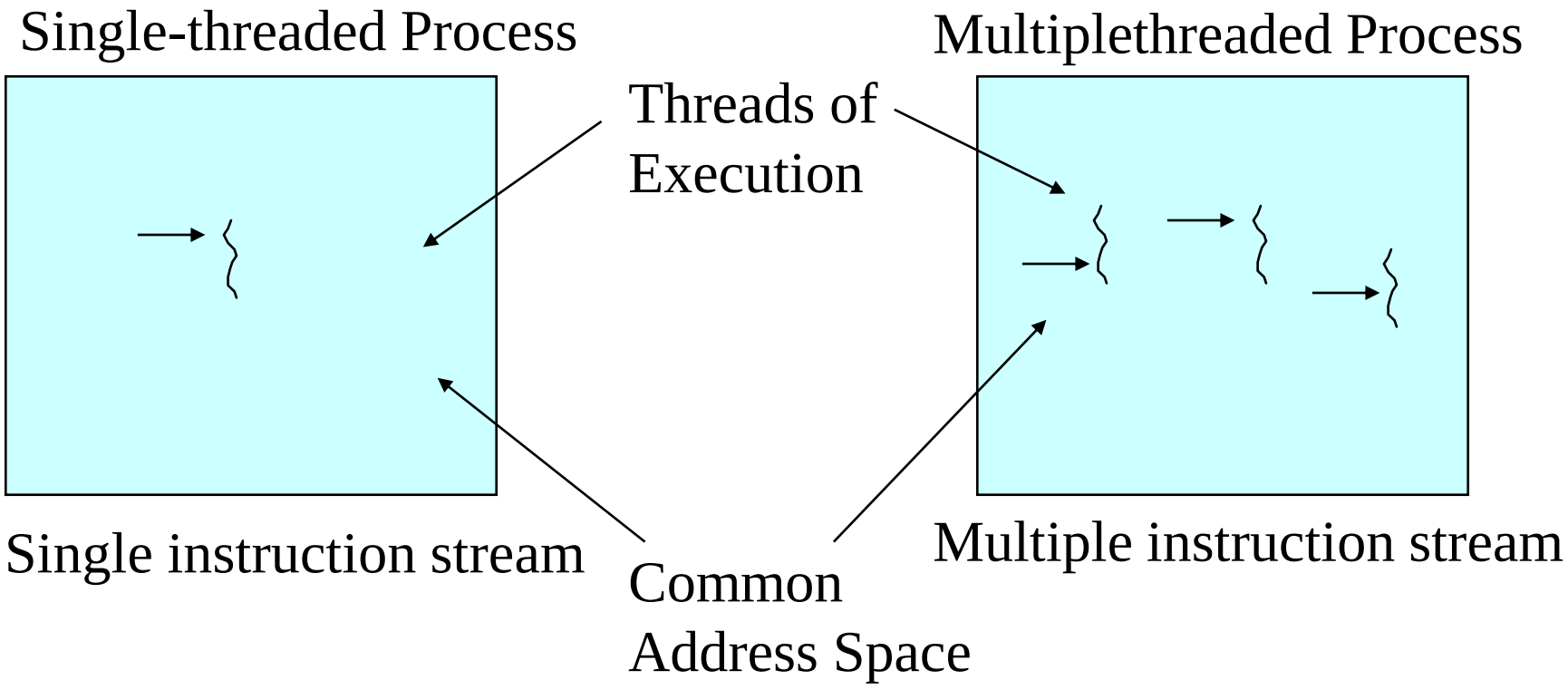
# We can write Multithreaded Programs



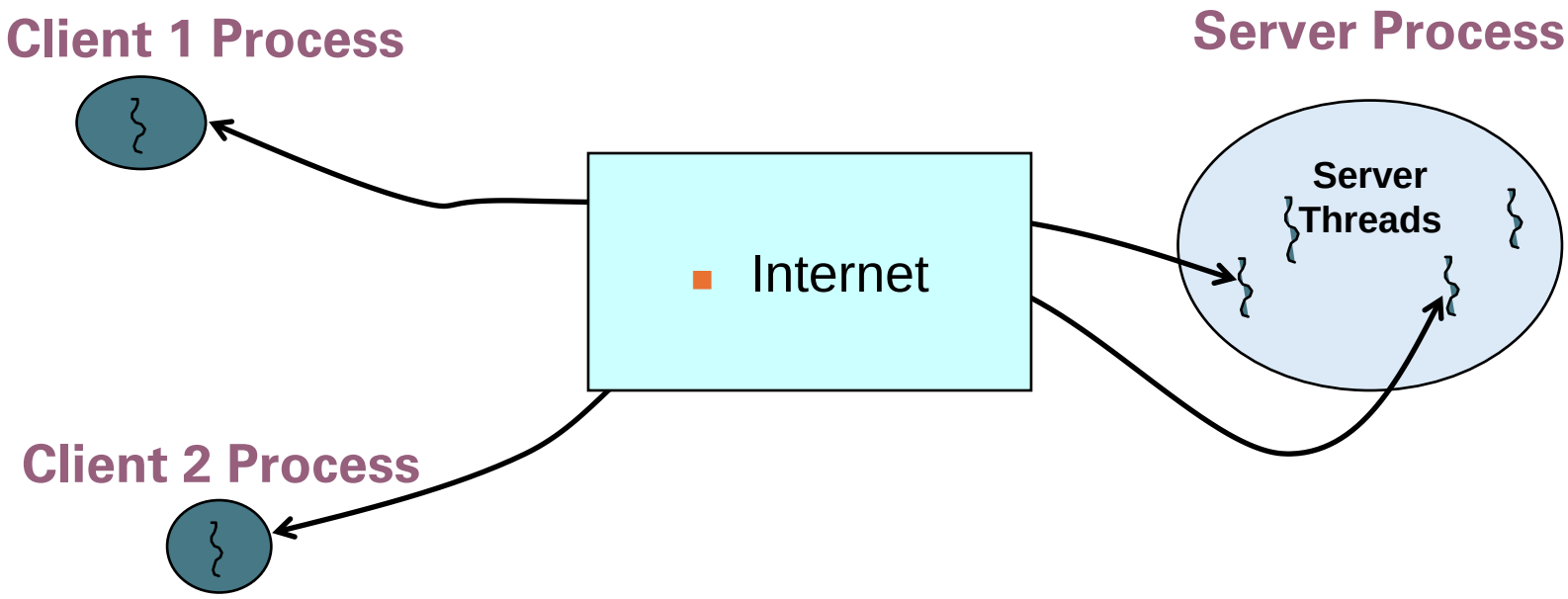

Threads may switch or exchange data/results

# Single and Multithreaded Processes

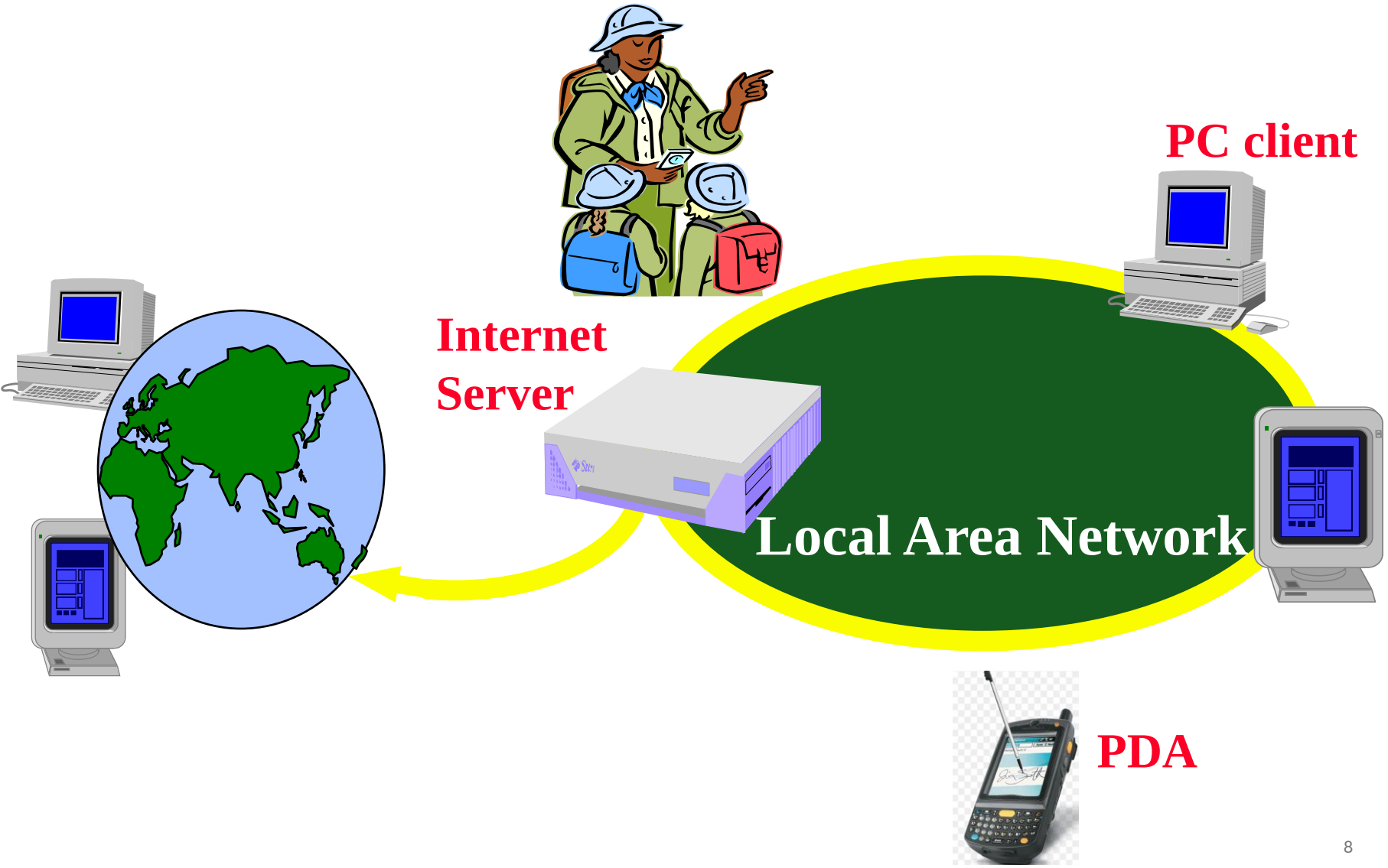
threads are light-weight processes within a process



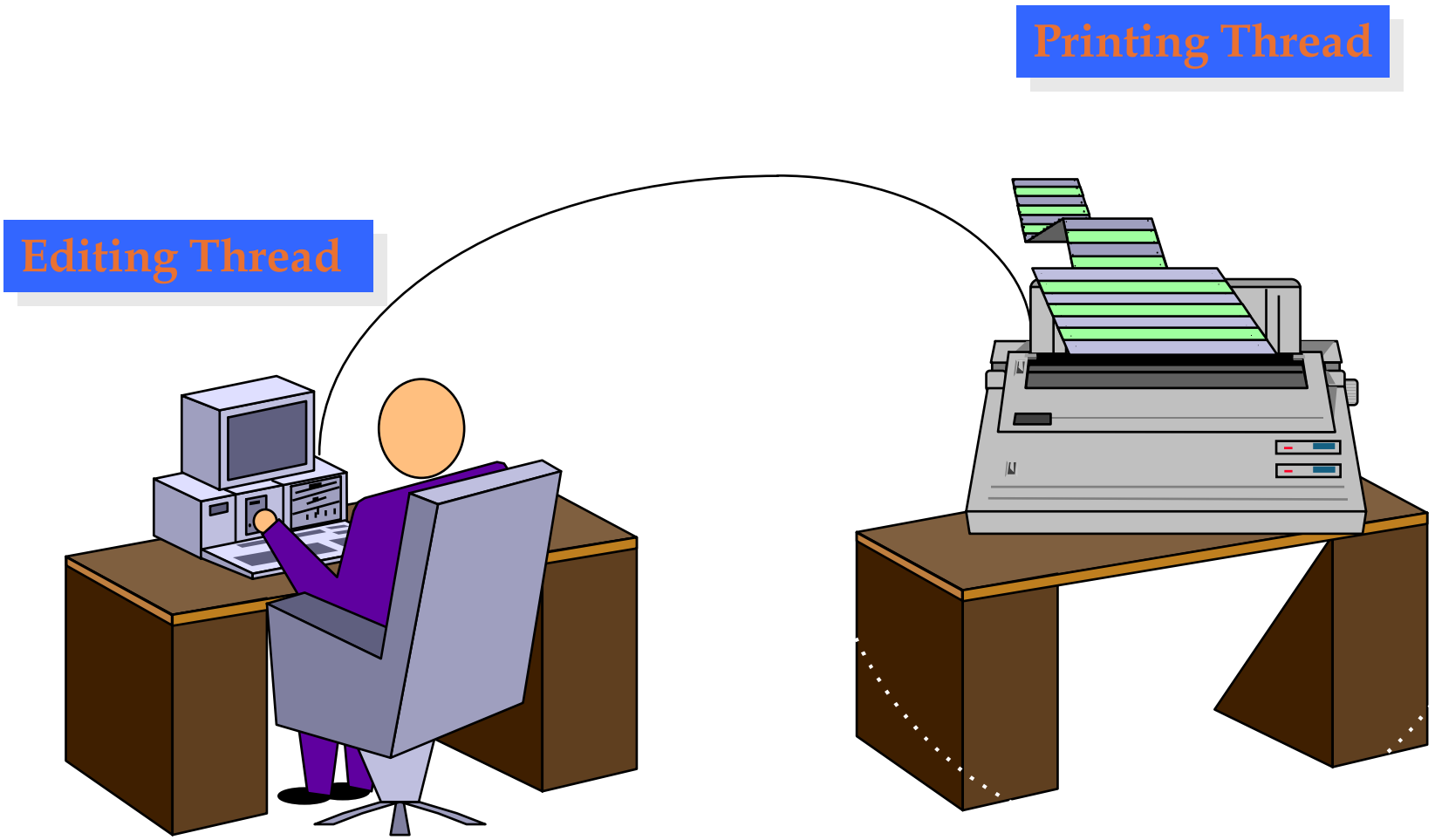
# Multithreaded Server: For Serving Multiple Clients Concurrently



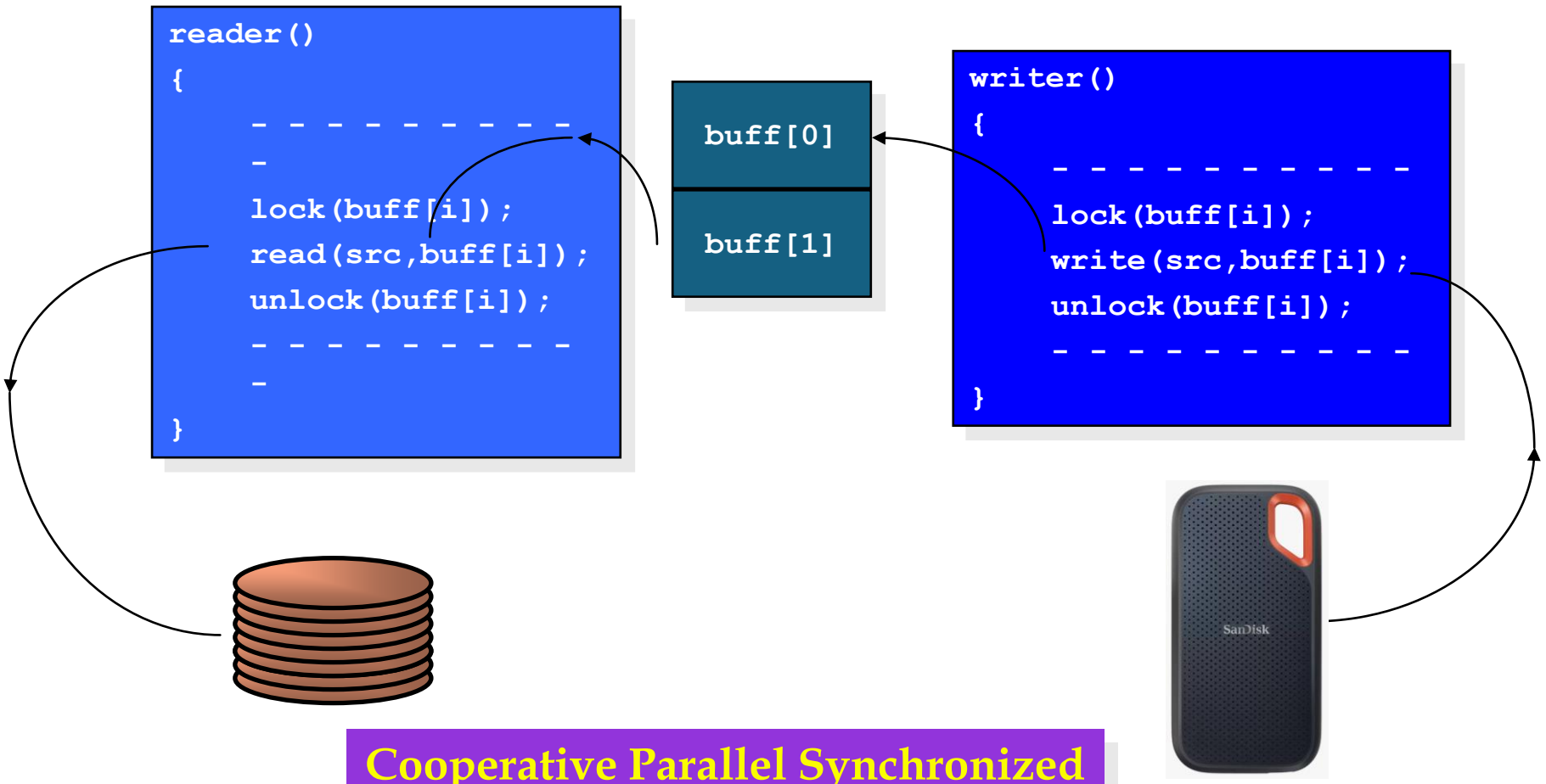
# Web Applications:
# Serving Many Users Simultaneously



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



# Multithreaded/Parallel File Copy




**Cooperative Parallel Synchronized Threads**

# Benefits of Threads

1. **Concurrency**: Threads enable concurrent execution of multiple tasks, allowing programs to perform multiple operations simultaneously.
2. **Responsiveness**: By using threads, programs can remain responsive even while performing time-consuming tasks in the background.
3. **Efficiency**: Threads allow programs to make efficient use of system resources, such as CPU cores, by executing tasks concurrently.
4. **Parallelism**: Threads enable parallel processing, where multiple threads can execute different parts of a program in parallel, potentially speeding up execution.
5. **Asynchronous Operations**: Threads are useful for handling asynchronous operations, such as downloading files or making network requests, without blocking the main program flow.

# Benefits of Threads

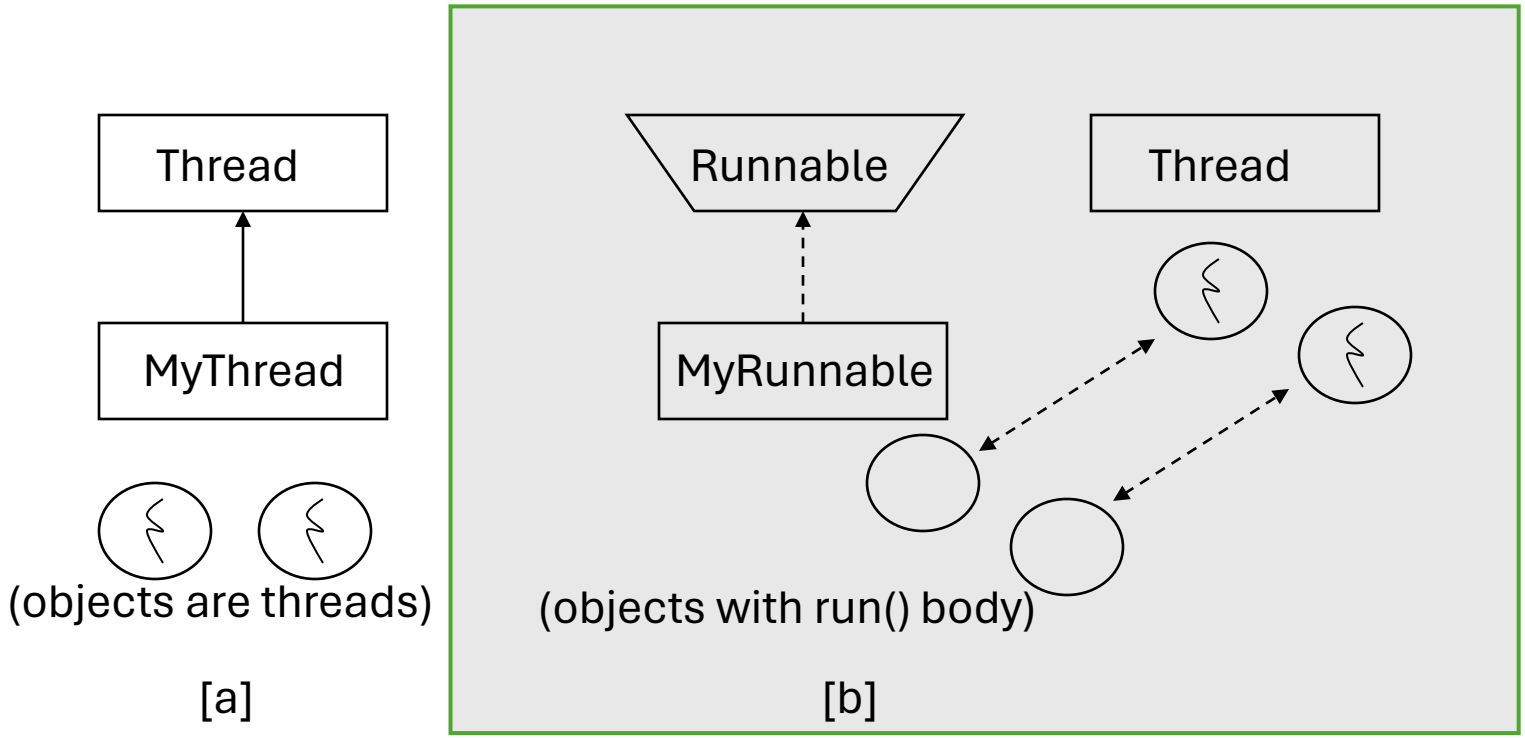
6. User Interface: Threads are essential in graphical user interfaces (GUIs) to keep the interface responsive while performing background tasks.
7. Server Applications: Threads are valuable in server applications to handle multiple client requests concurrently, ensuring efficient resource utilization.
8. Background Processing: Threads are used for running background tasks, such as data processing or periodic maintenance, without impacting the main execution flow.
9. Multitasking: Threads allow programs to perform multiple tasks at the same time, such as processing input while generating output or handling multiple events concurrently.
10. Resource Sharing: Threads facilitate sharing resources, such as data structures or files, between different parts of a program, enabling efficient collaboration and

# 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.

# Two Ways to Create Threads

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



# 1st method: Extending Thread class

- Create a class by extending Thread class and override run() method:

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

- Create a thread:

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

- Start Execution of threads:

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

- Create and Execute:

```
new MyThread().start();
```

# An example

```
class MyThread extends Thread {
      public void run() {
          for(int i=1 ; i<11; i++)
            System.out.println(" this thread is running ... ");
      }
}

class ThreadTest {
      public static void main(String [] args  ) {
          MyThread thread1 = new MyThread();
           thread1.start();
      }
}
```

# 2nd method: Threads by implementing Runnable interface

- Create a class that implements the interface Runnable and override run() method:

```
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 );
```

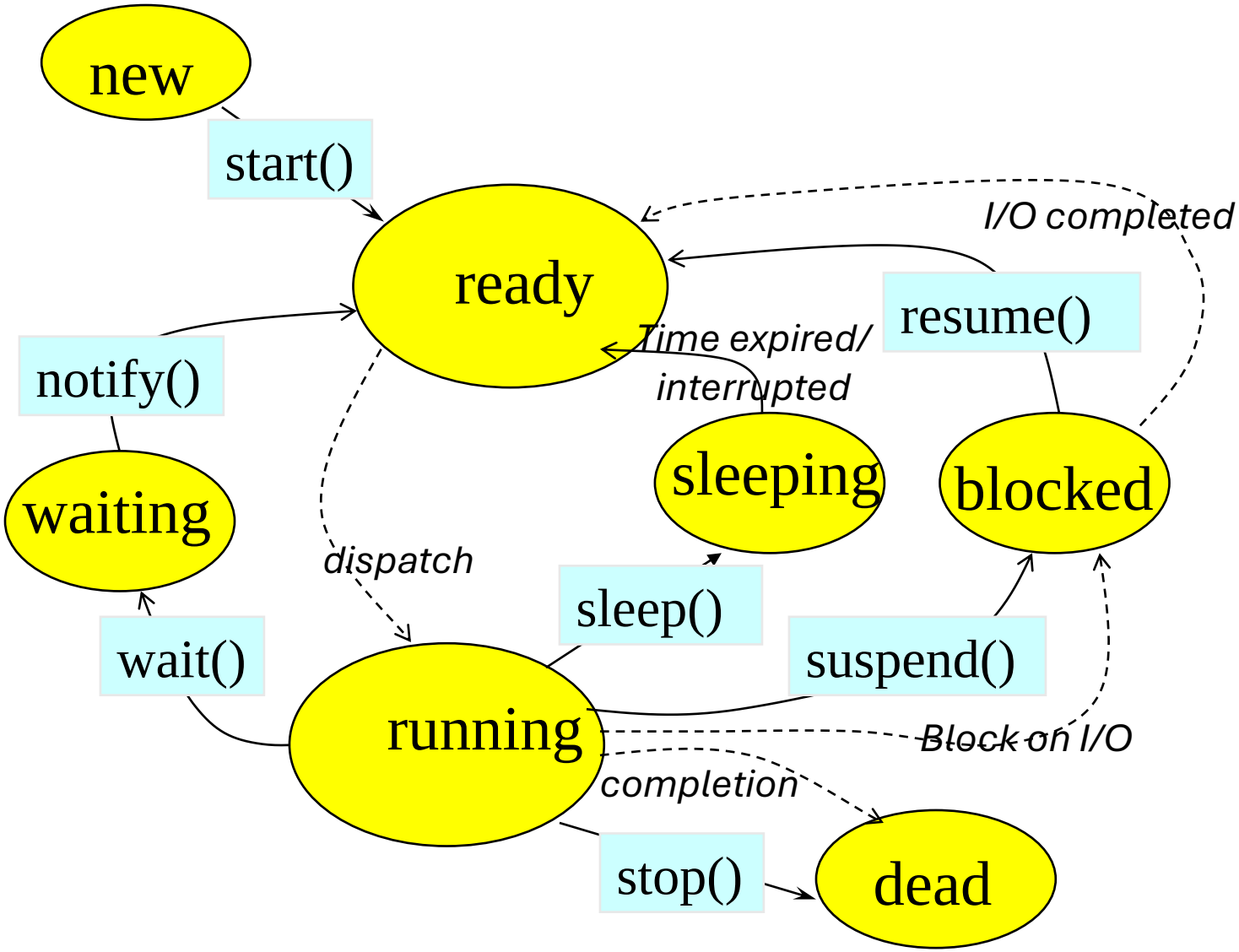
- Start Execution:

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

# An example

```
class MyRunnable implements Runnable  {
      public void run() {
            for(int i=1 ; i<11; i++)
             System.out.println(" this thread is running ... ");
      }
}

class ThreadTest {
      public static void main(String [] args  ) {
            MyRuinnable runnable1 = new MyRunnable();
            Thread thread2 = new Thread(runnable1);
             thread2.start();
      }
}
```

# Life Cycle of Thread



# Example

- Write a program that creates 3 threads

# Three threads example

```
class MyThread1 extends Thread
{
    public void run()
     {
        for(int i=1;i<=5;i++)
         {   System.out.println("\t From Thread1: i= "+i);            }
          System.out.println("Exit from A");
     }
}
class MyThread2 extends Thread
{     public void run()
     {
        for(int j=1;j<=5;j++)
         {
            System.out.println("\t From Thread1: j= "+j);
         }
          System.out.println("Exit from B");
     }
}
```

```
public class MyThread3 extends Thread
{
    public void run()
     {
        for(int k=1;k<=5;k++)
         {
            System.out.println("\t From Thread3: k= "+k);
         }
          System.out.println("Exit from C");
     }
}
public class ThreadTest
{       public static void main(String args[])
       {   MyThread1 tr1 =  new MyThread1();
           MyThread1 tr2 =  new MyThread2();
           MyThread1 tr3 =  new MyThread3();
            tr1.start();
            tr2.start();
            tr3.start();
}
}
```

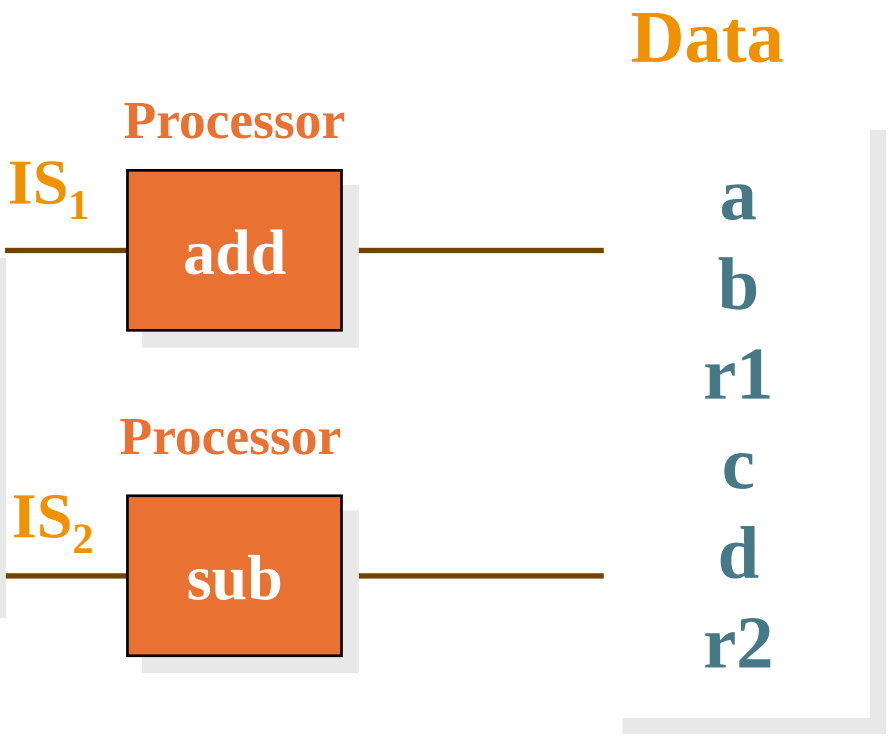
# Run 1

# Run2

# 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);
```

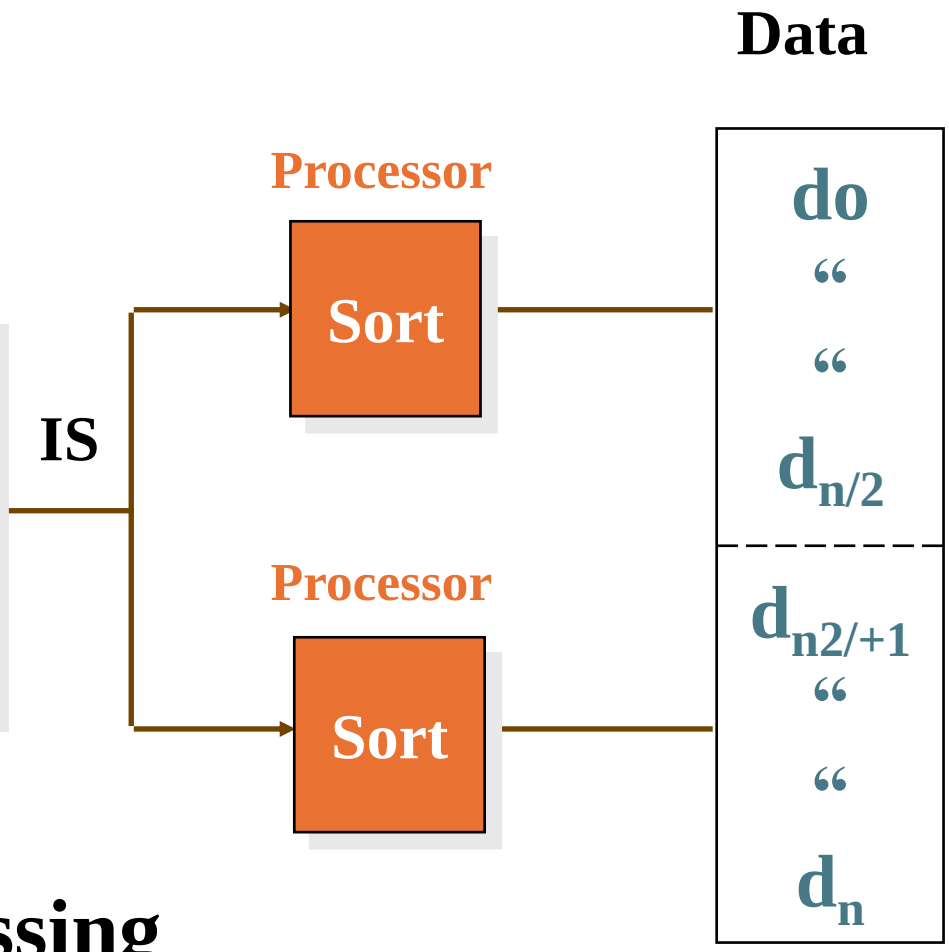



**MISD and MIMD Processing**

# 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);
```




**SIMD Processing**

# 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(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");
    }
}
```

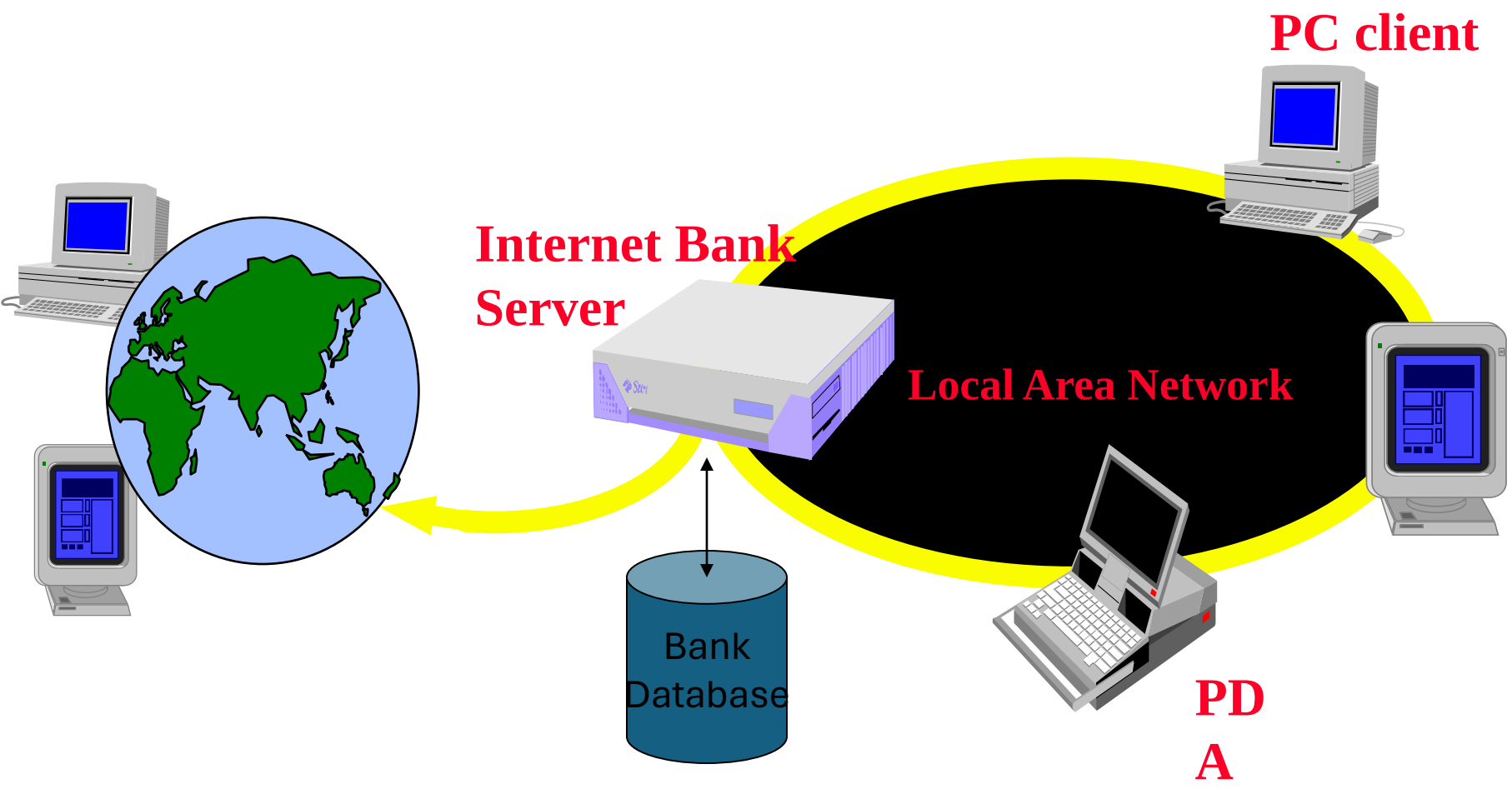
# Thread Priority Example

```
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");
      }
}
```

# 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.
  - Can the following operations be done at the same time on the same account?
    - Deposit()
    - Withdraw()
    - Enquire()

# Online Bank: Serving Many Customers and Operations



# Shared Resources

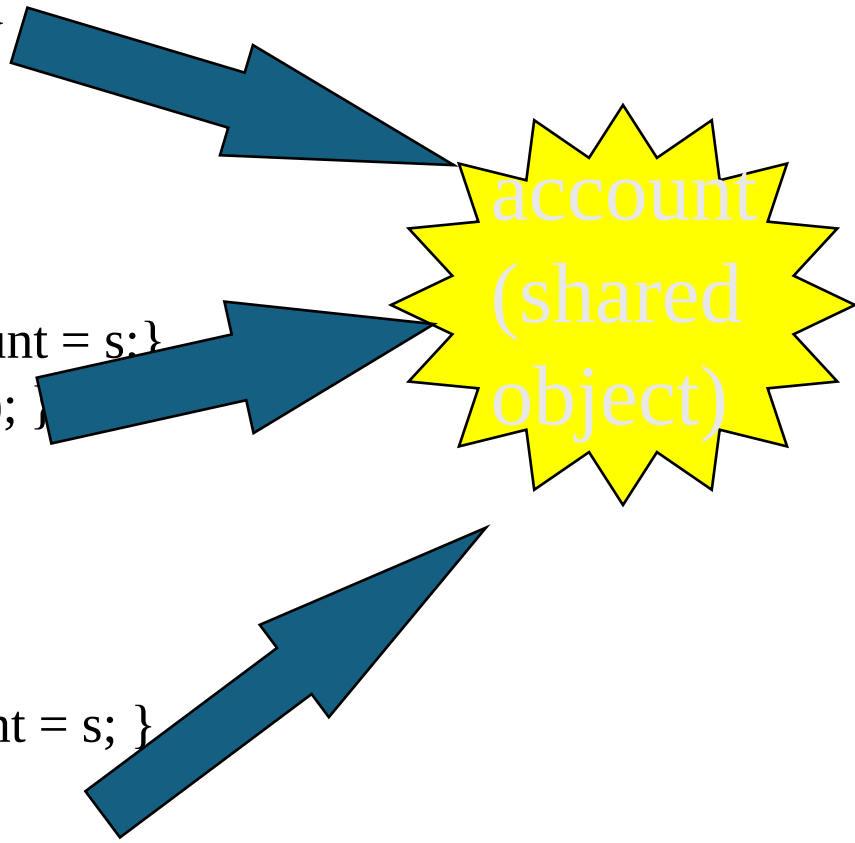
- If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.
- This can be prevented by synchronising access to the data.
- Use “Synchronized” method:
  - public synchronized void update()
  - {
    - …
  - }

# 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()
}
```

# Shared account object between 3 threads

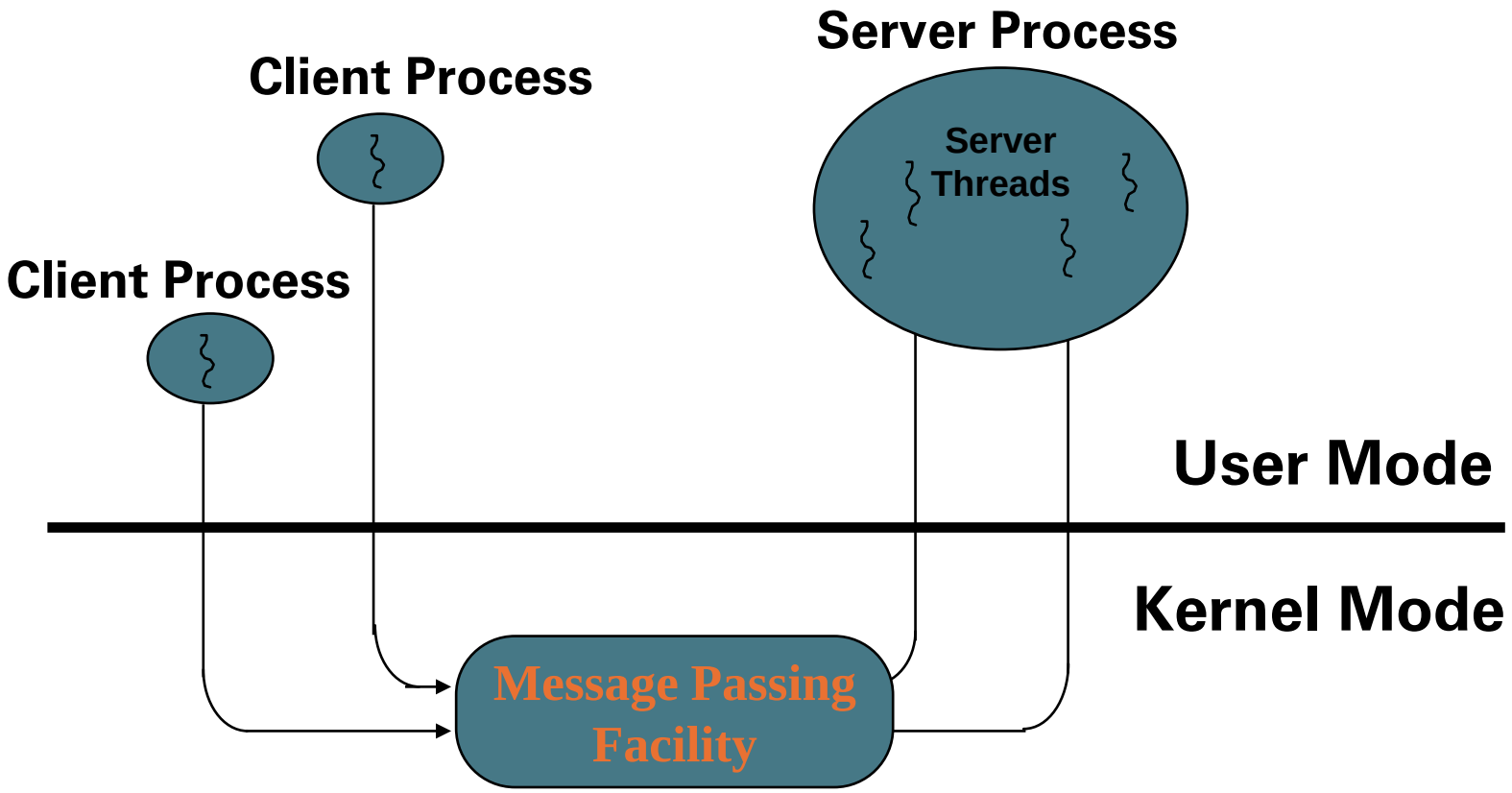
```
class MyThread implements Runnable  {
 Account account;
        public MyThread (Account s) {  account = s;}
        public void run() { account.deposit(); }
} // end class MyThread

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
```



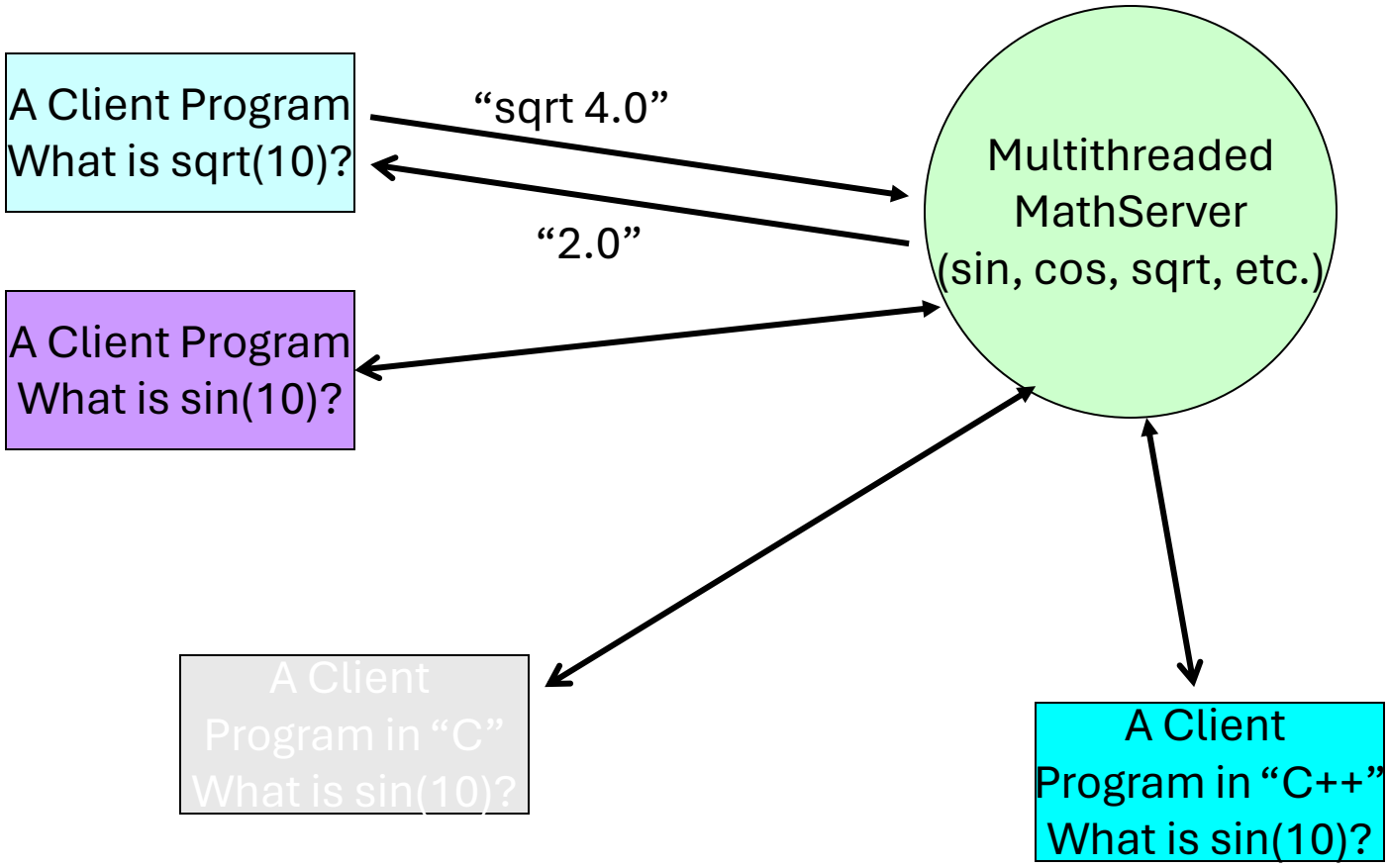
# Monitor (shared object access): serializes operation on shared object

```
class Account {   // the 'monitor'
  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.
       }
}
```

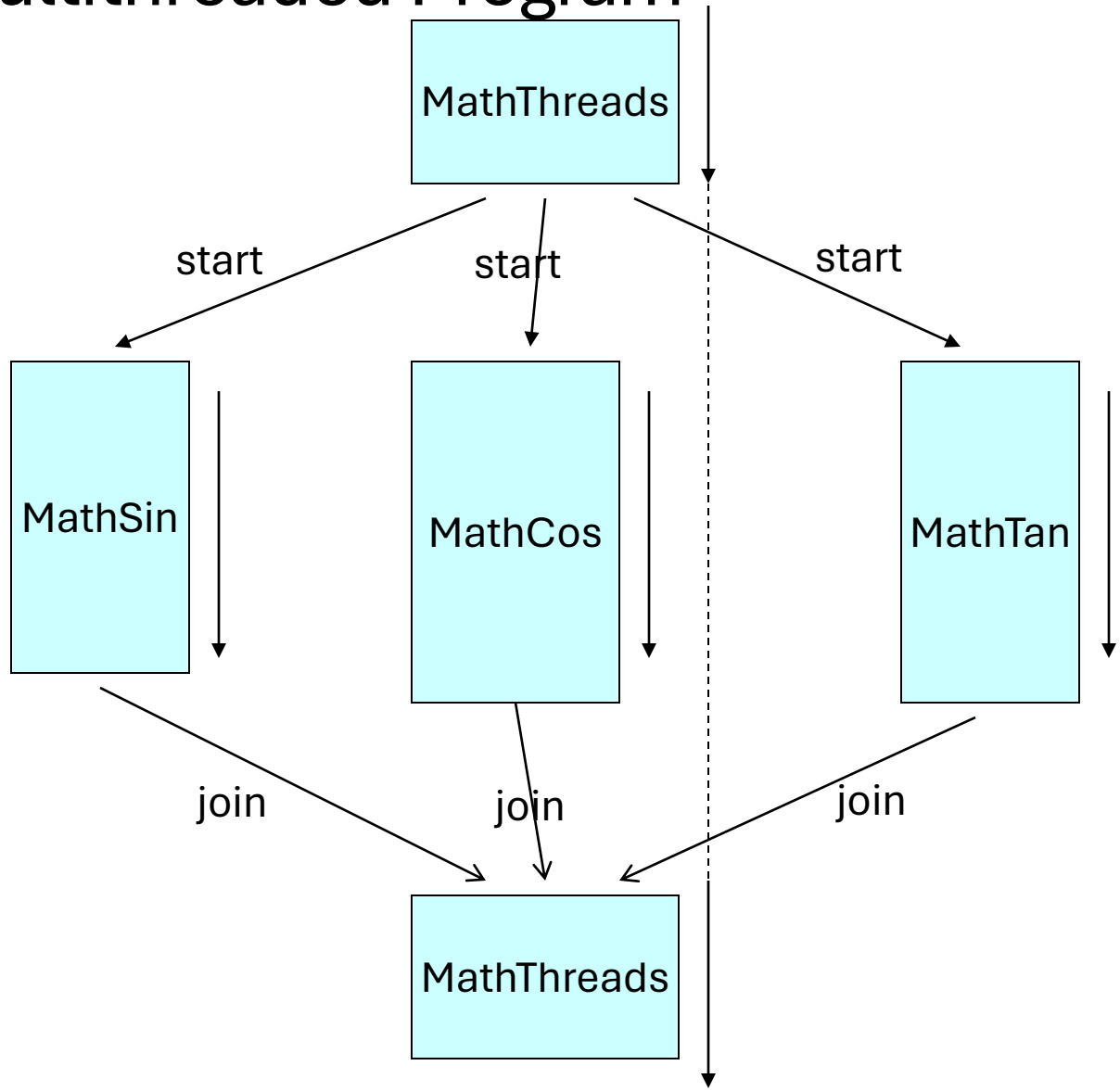
# Multithreaded Server



# Assignment 1: Multithreaded MathServer – Demonstrates the use of Sockets and Threads



# A Multithreaded Program

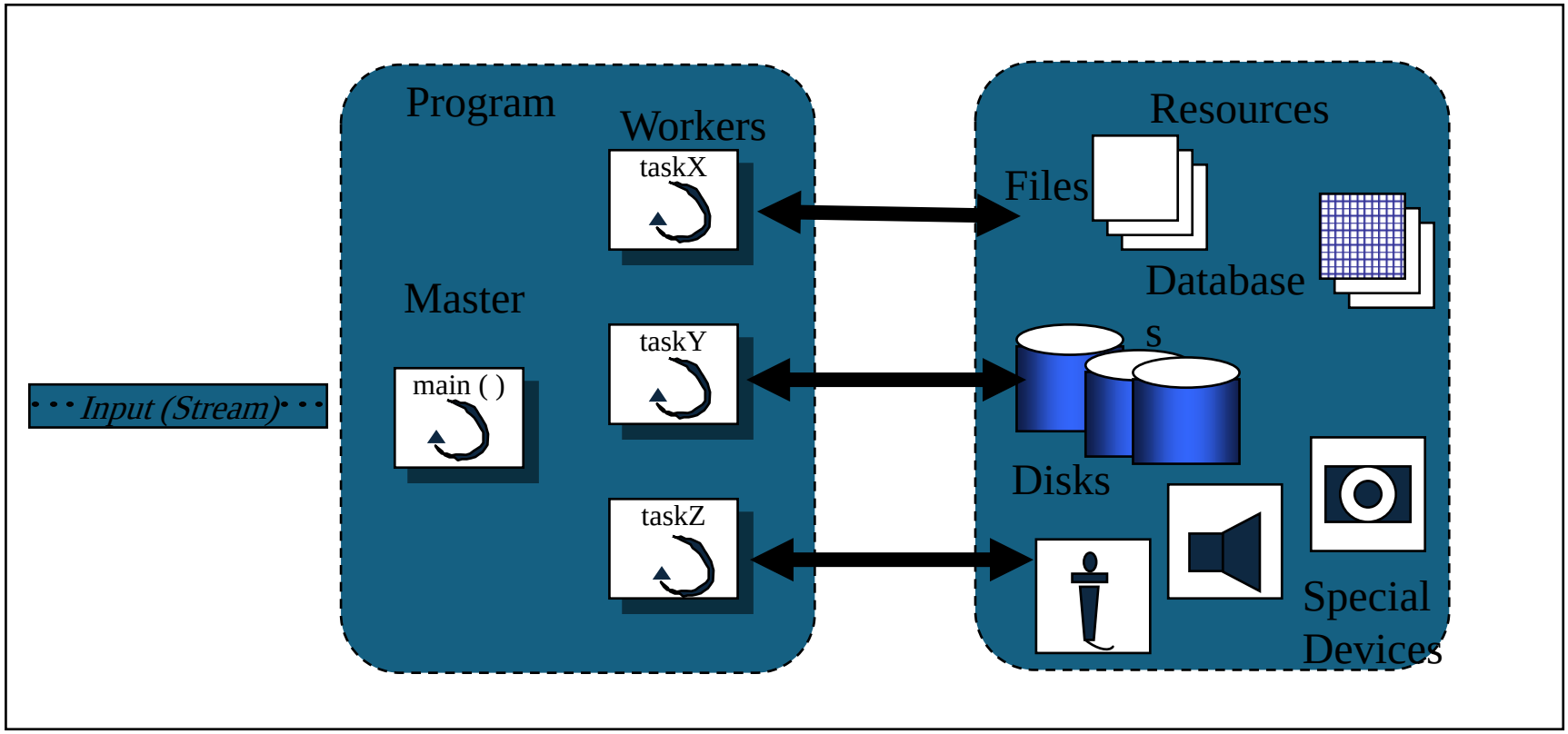


Thread Programming models

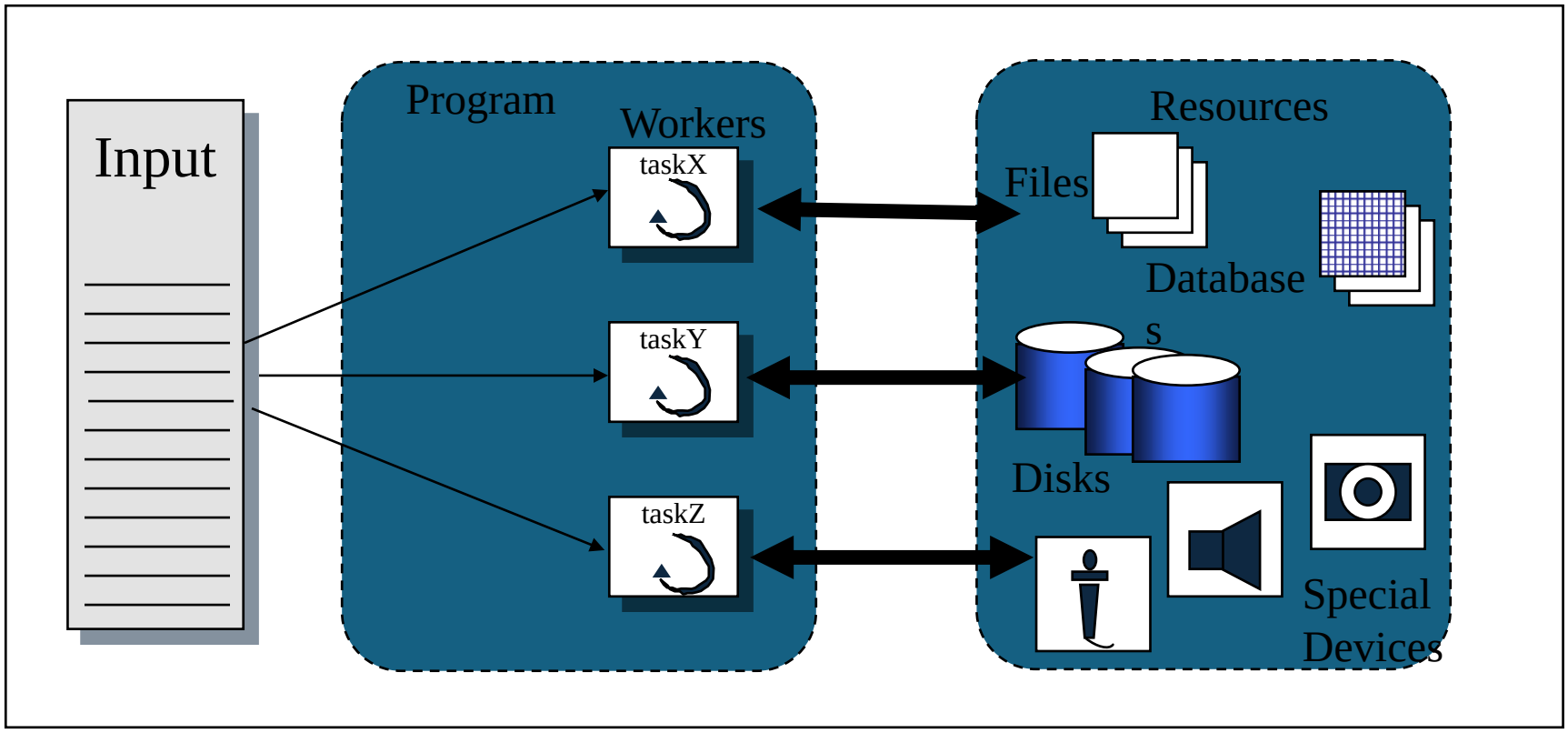
# Thread concurrency/operation models

- The master/worker model
- The peer model
- A thread pipeline

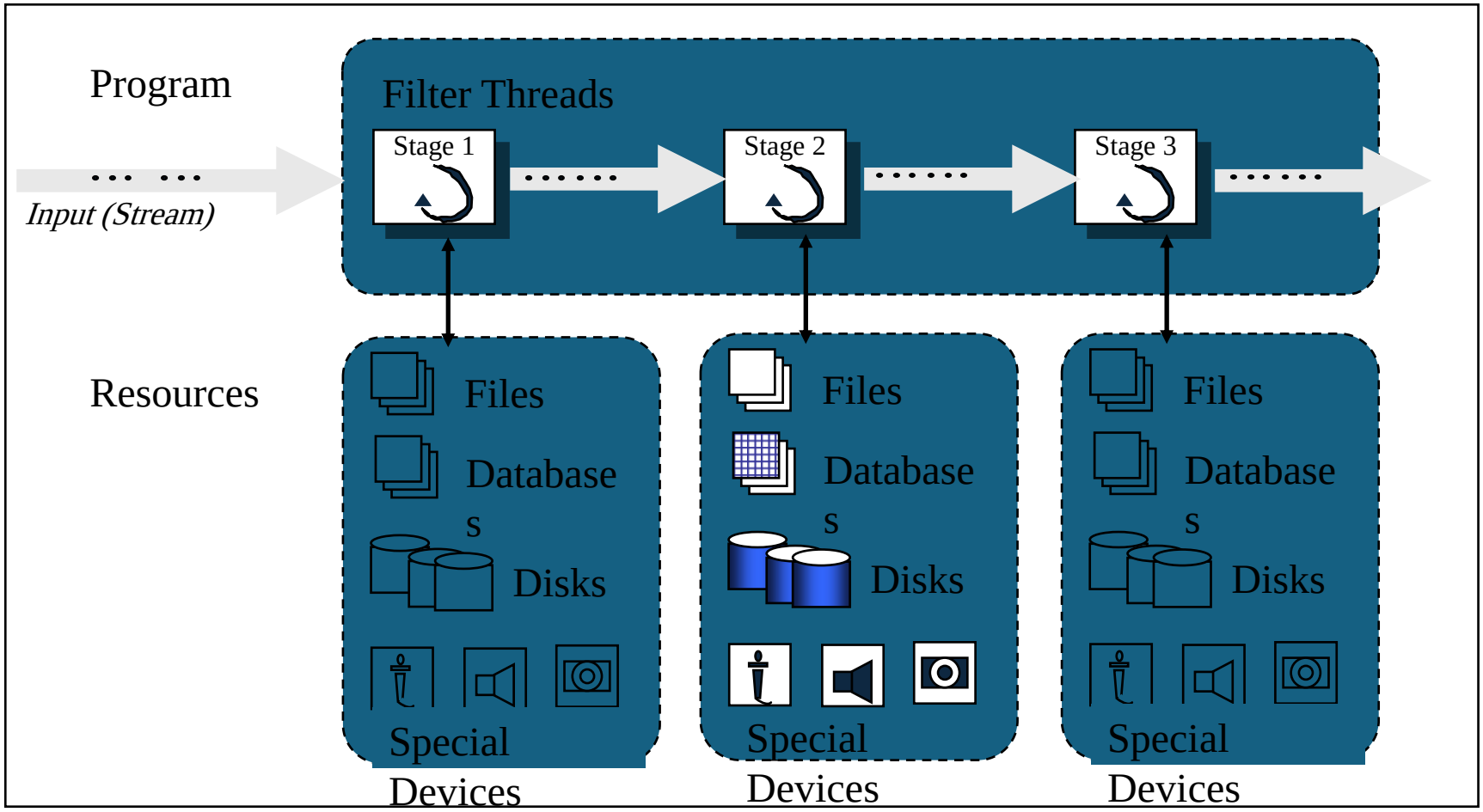
# The master/worker model



# The peer model



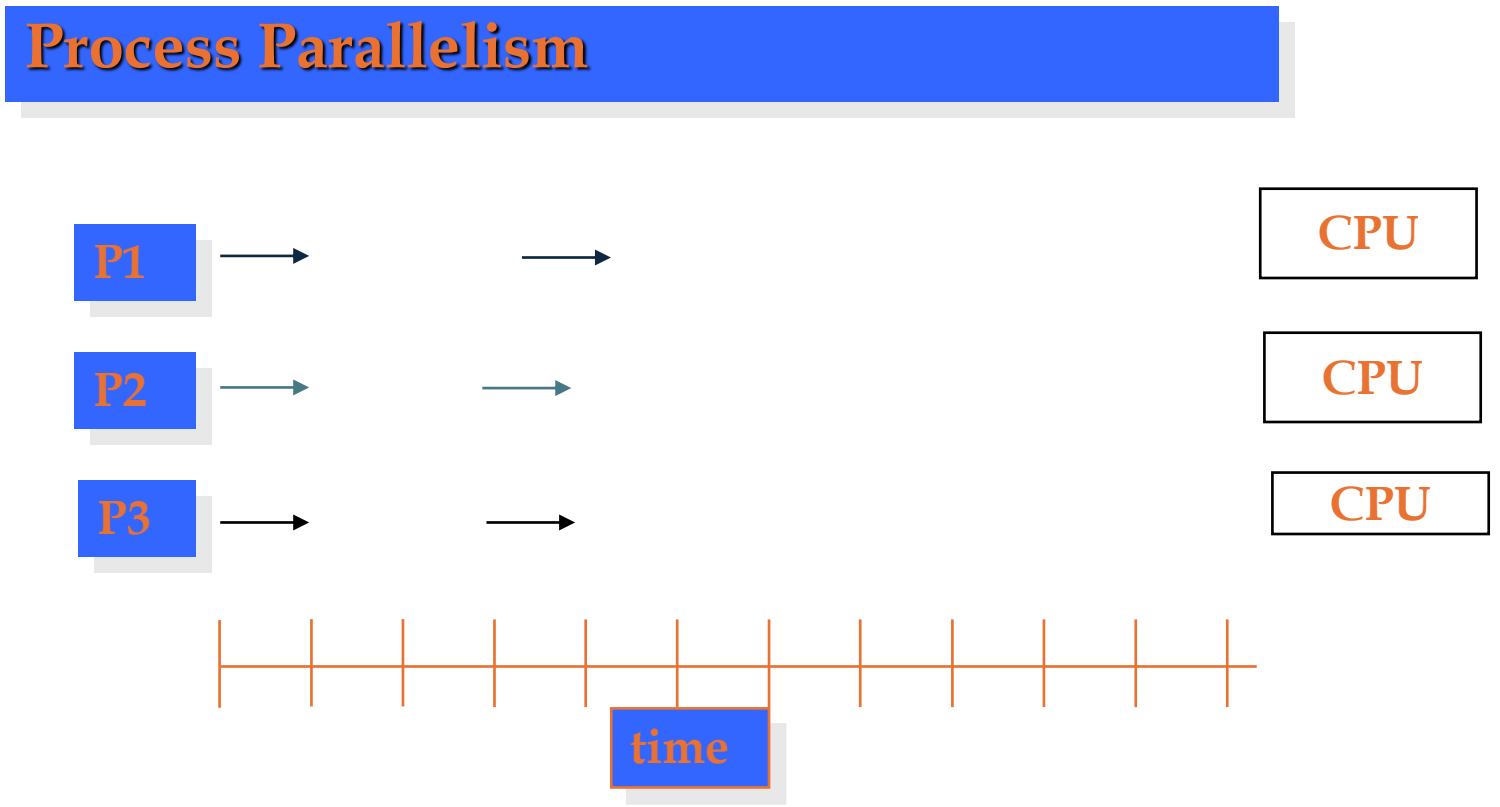
# A thread pipeline



# Multithreading and Multiprocessing Deployment issues

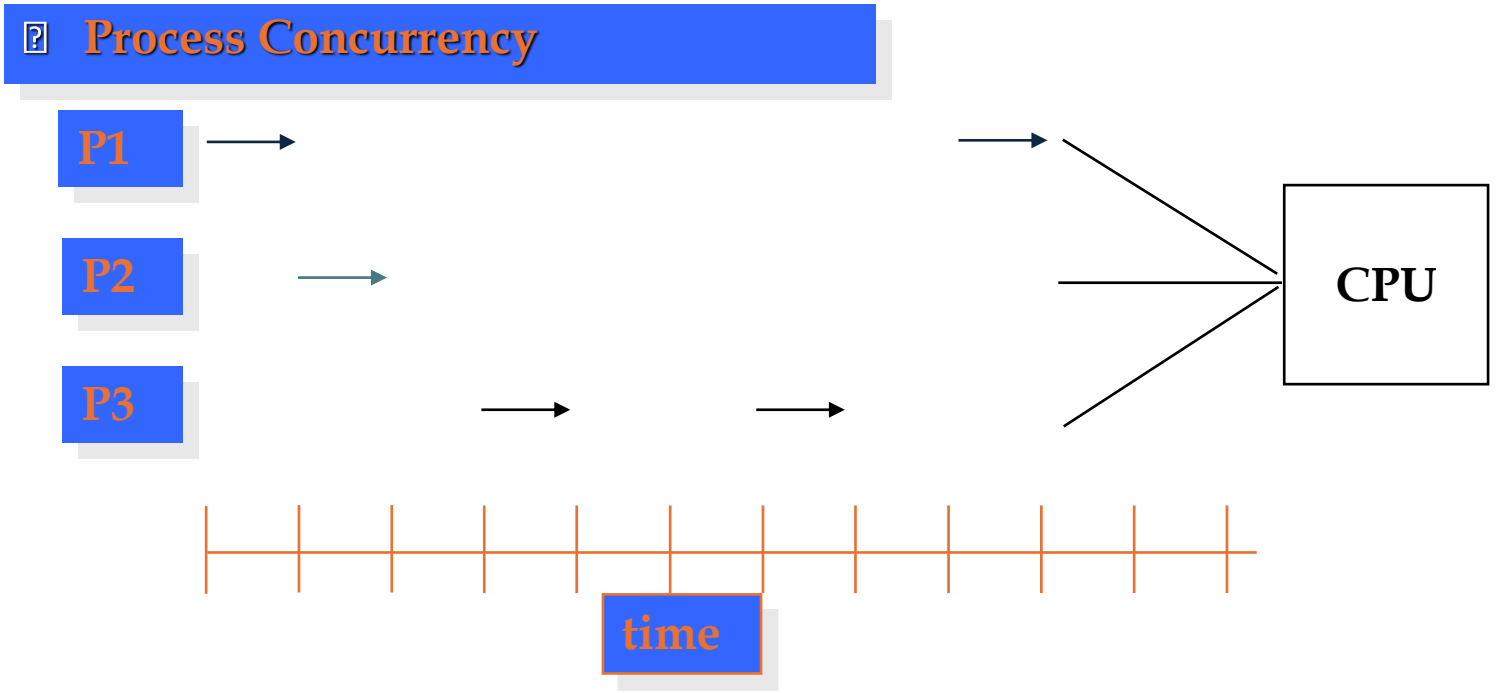
On Shared and distributed memory systems

# Multithreading - Multiprocessors



# Multithreading on Uni-processor

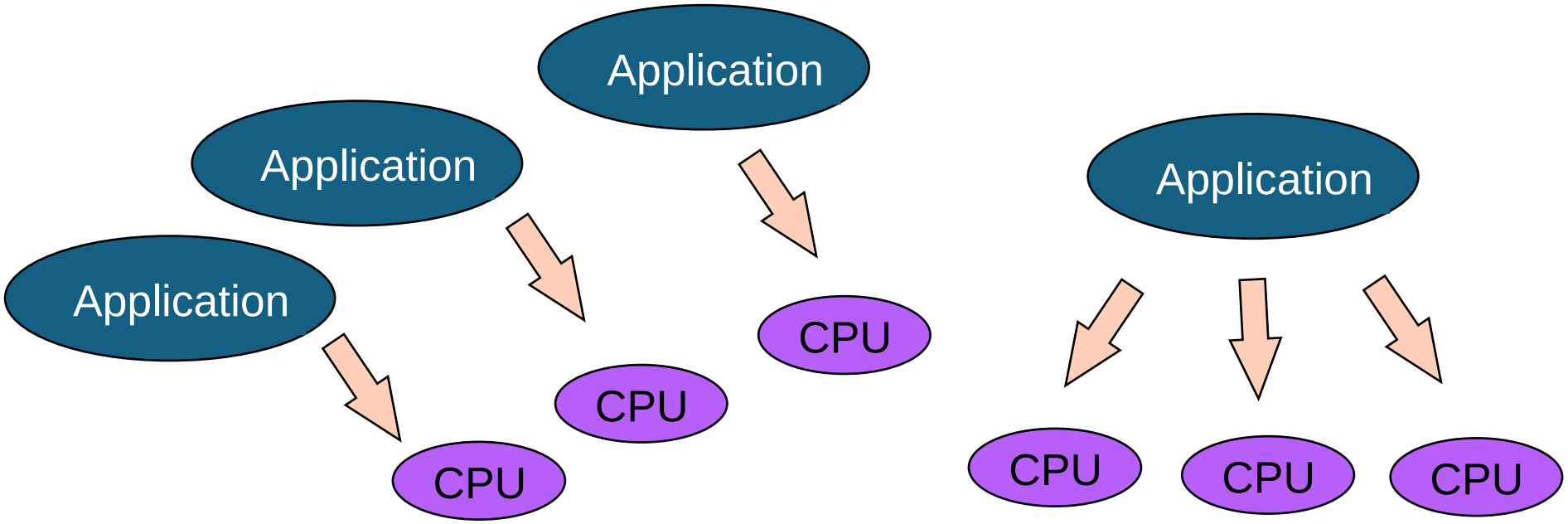
- Concurrency Vs Parallelism




**Number of Simultaneous execution units > number of CPUs**

# Multi-Processing (clusters & grids) and Multi-Threaded Computing

*Threaded Libraries, Multi-threaded I/O*




*Better Response Times in Multiple Application Environments*

*Higher Throughput for Parallelizeable Applications*

# References


- Rajkumar Buyya, Thamarai Selvi, Xingchen Chu, **Mastering OOP with Java**, McGraw Hill (I) Press, New Delhi, India, 2009.
- Sun Java Tutorial – Concurrency:
  - http://java.sun.com/docs/books/tutorial/essential/concurrency/