

Multithreading using Java

Dr. Ferdin Joe John Joseph

KAMNOETVIDYA

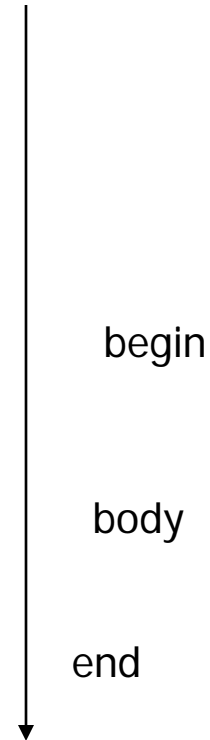
CIENCE ACADEM

Agenda

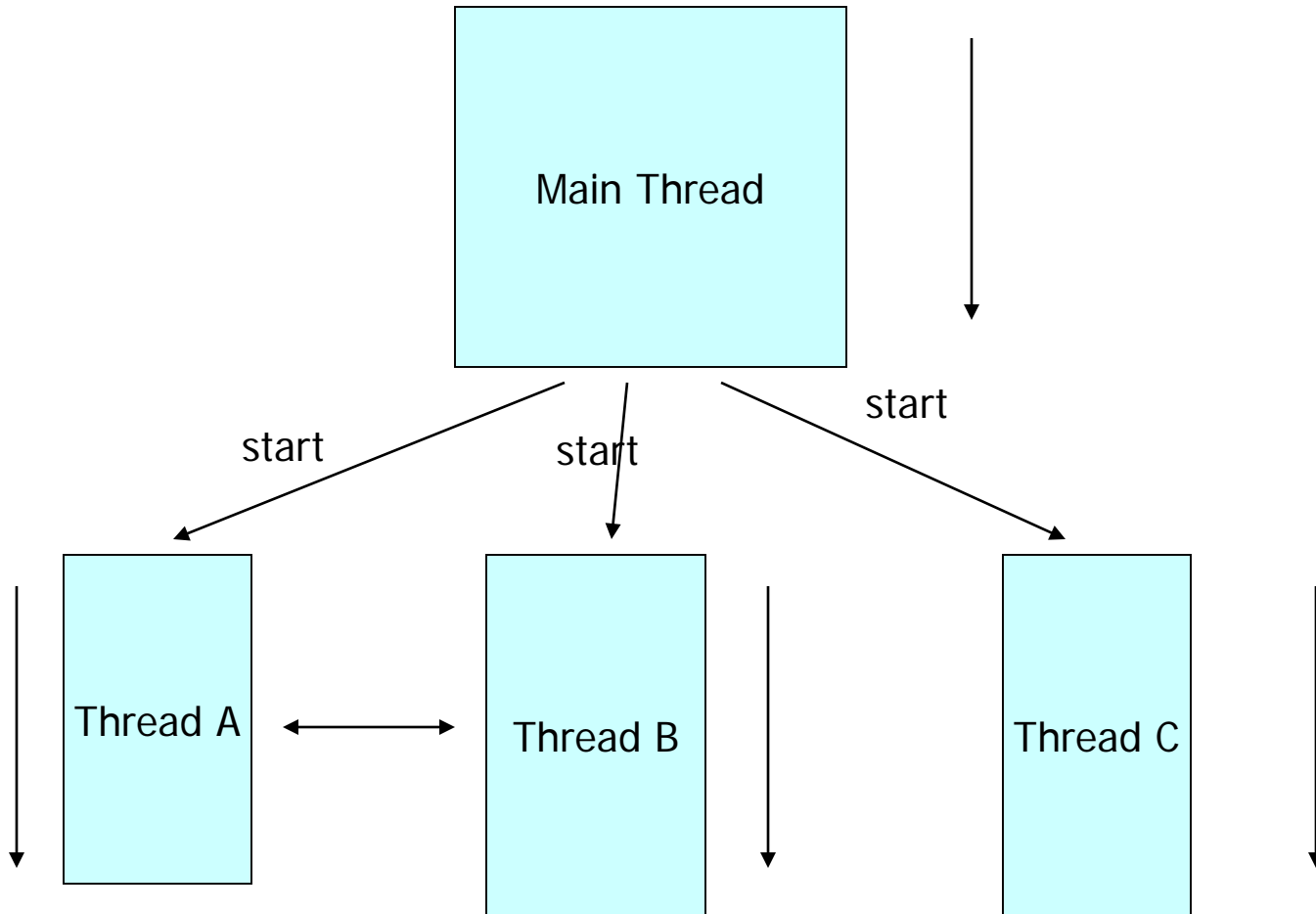
- Introduction
- Thread Applications
- Defining Threads
- Java Threads and States
 - Priorities
- Accessing Shared Resources
 - Synchronisation
- Assignment 1:
 - Multi-Threaded Math Server
- Advanced Issues:
 - Concurrency Models: master/worker, pipeline, peer processing
 - Multithreading Vs multiprocessing

A single threaded program

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



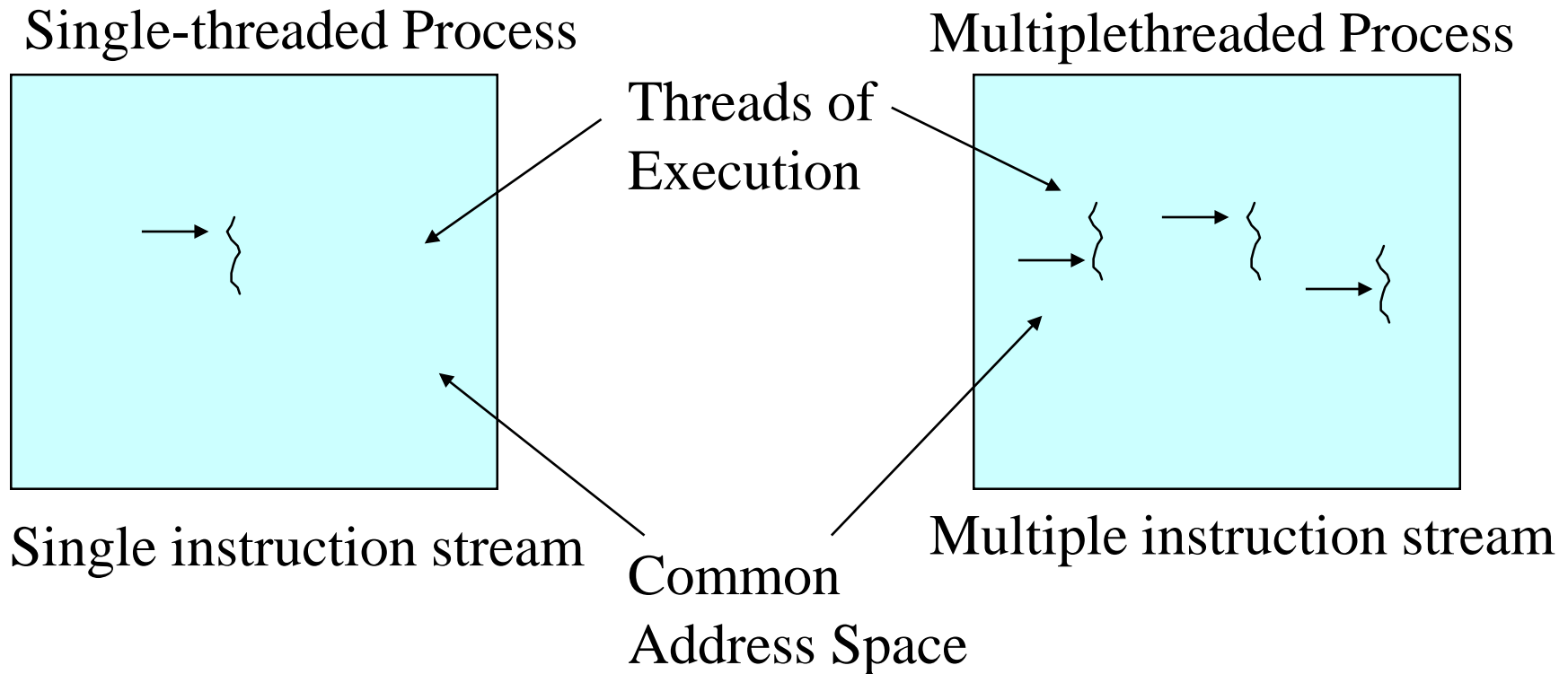
A Multithreaded Program



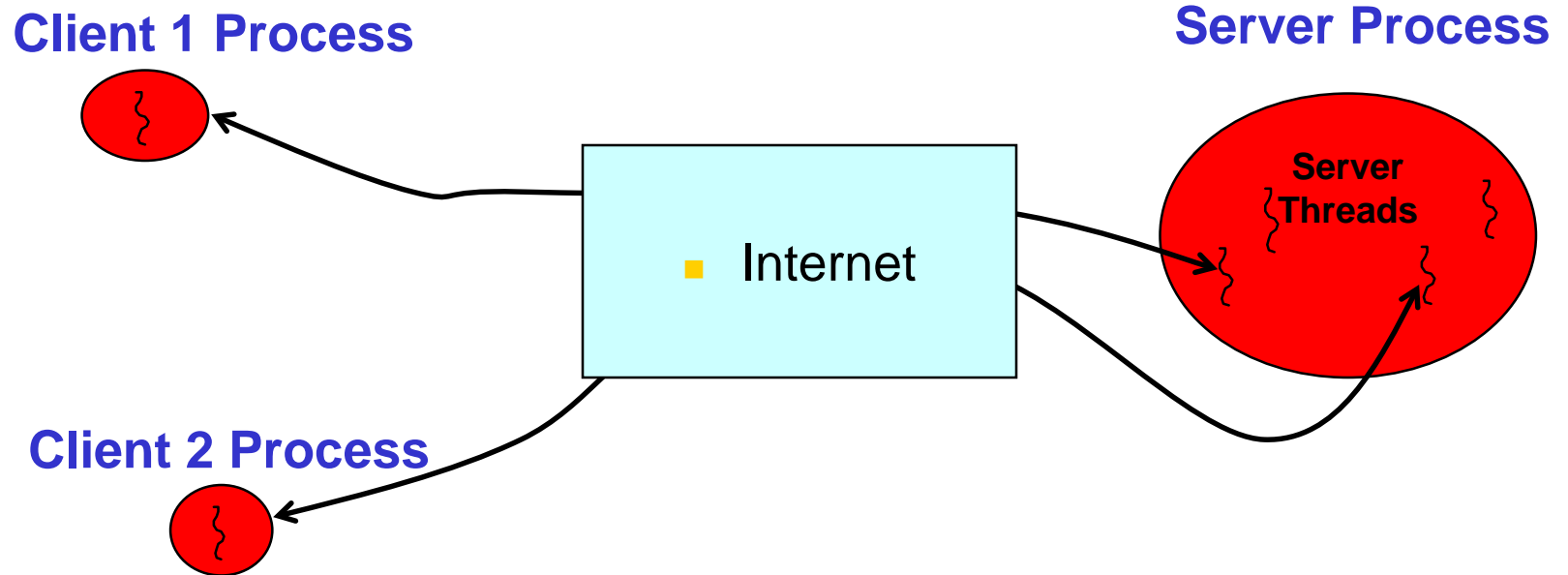
Threads may switch or exchange data/results

Single and Multithreaded Processes

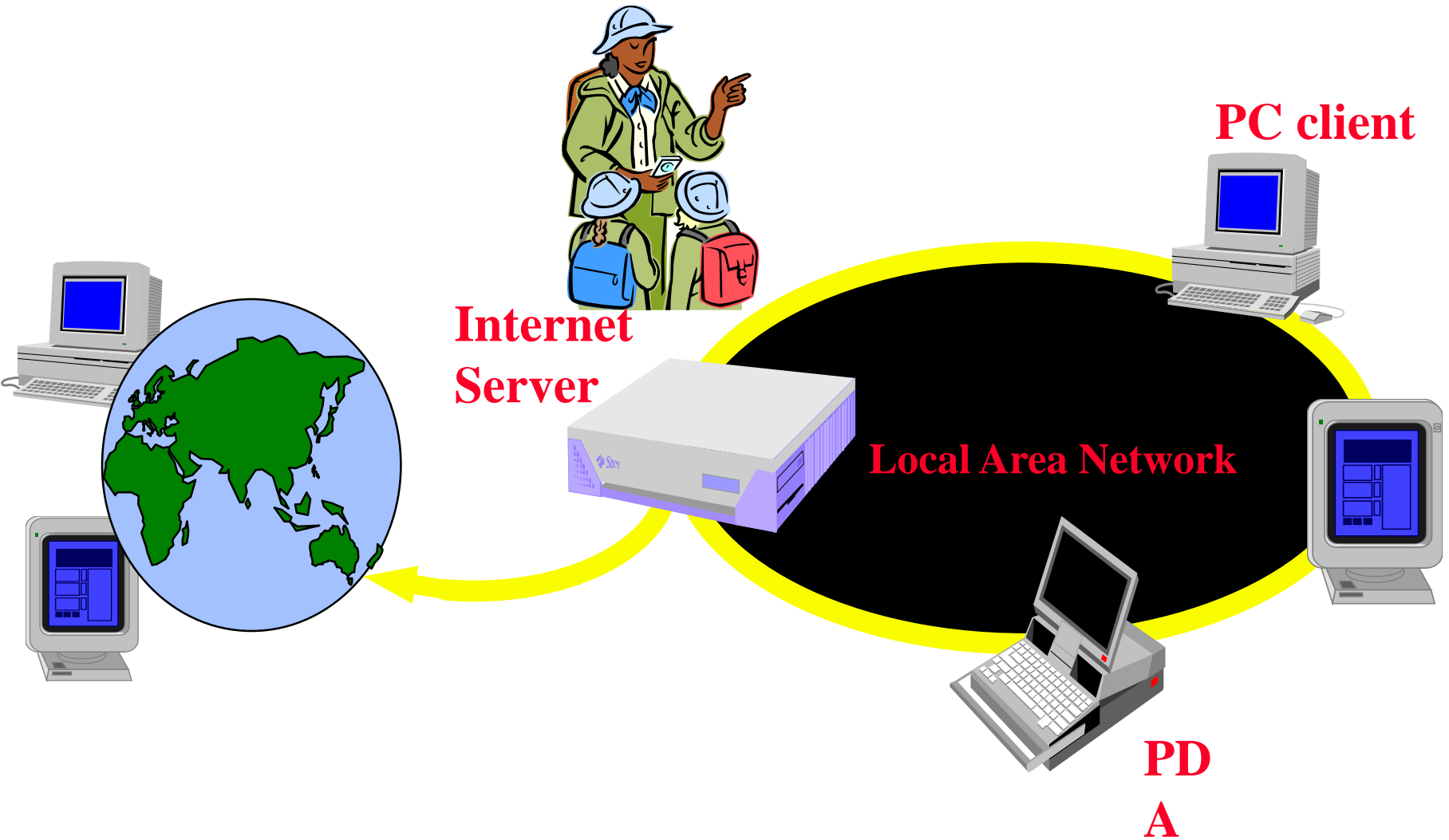
threads are light-weight processes within a process



Multithreaded Server: For Serving Multiple Clients Concurrently



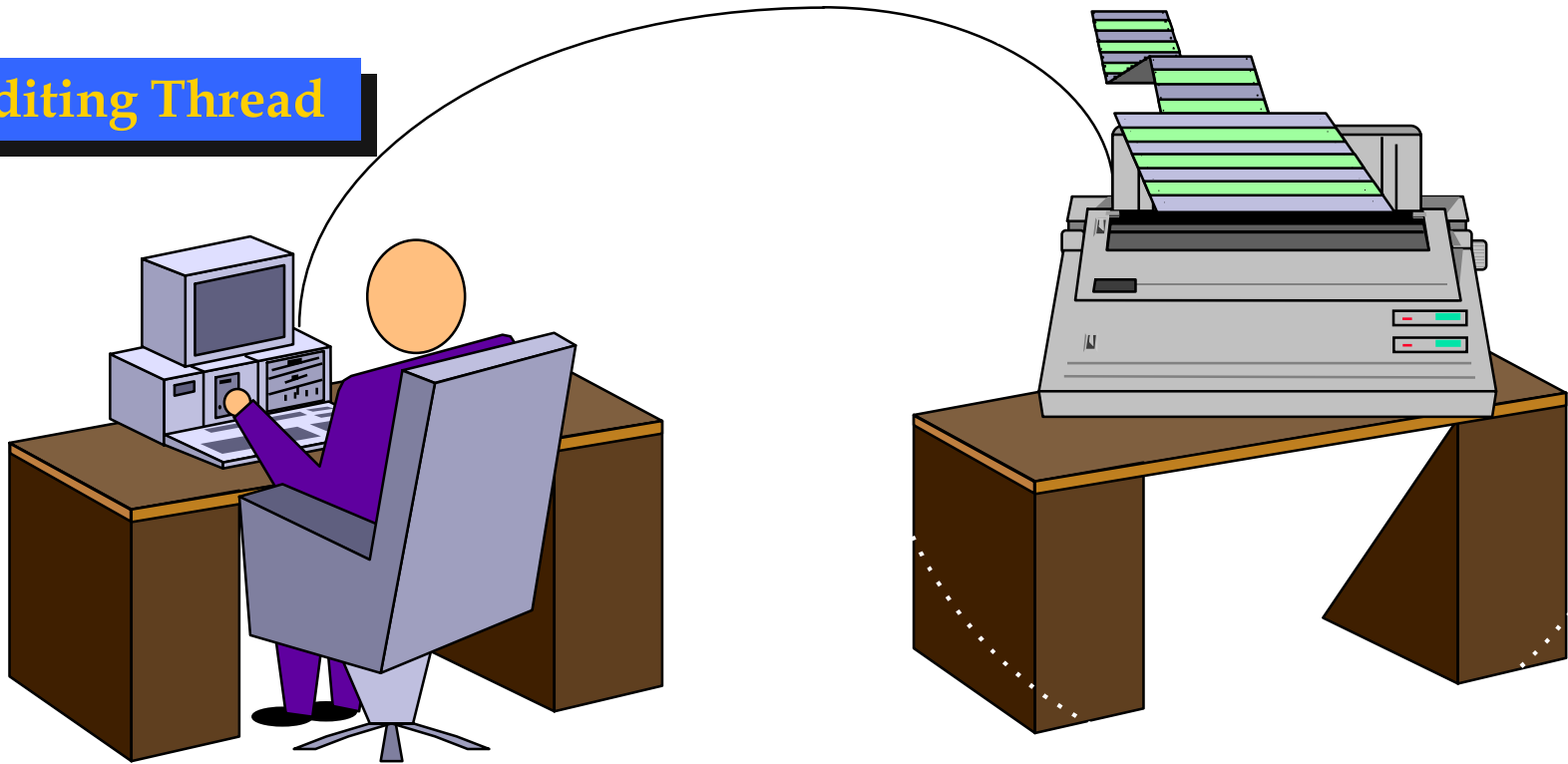
Web/Internet Applications: Serving Many Users Simultaneously



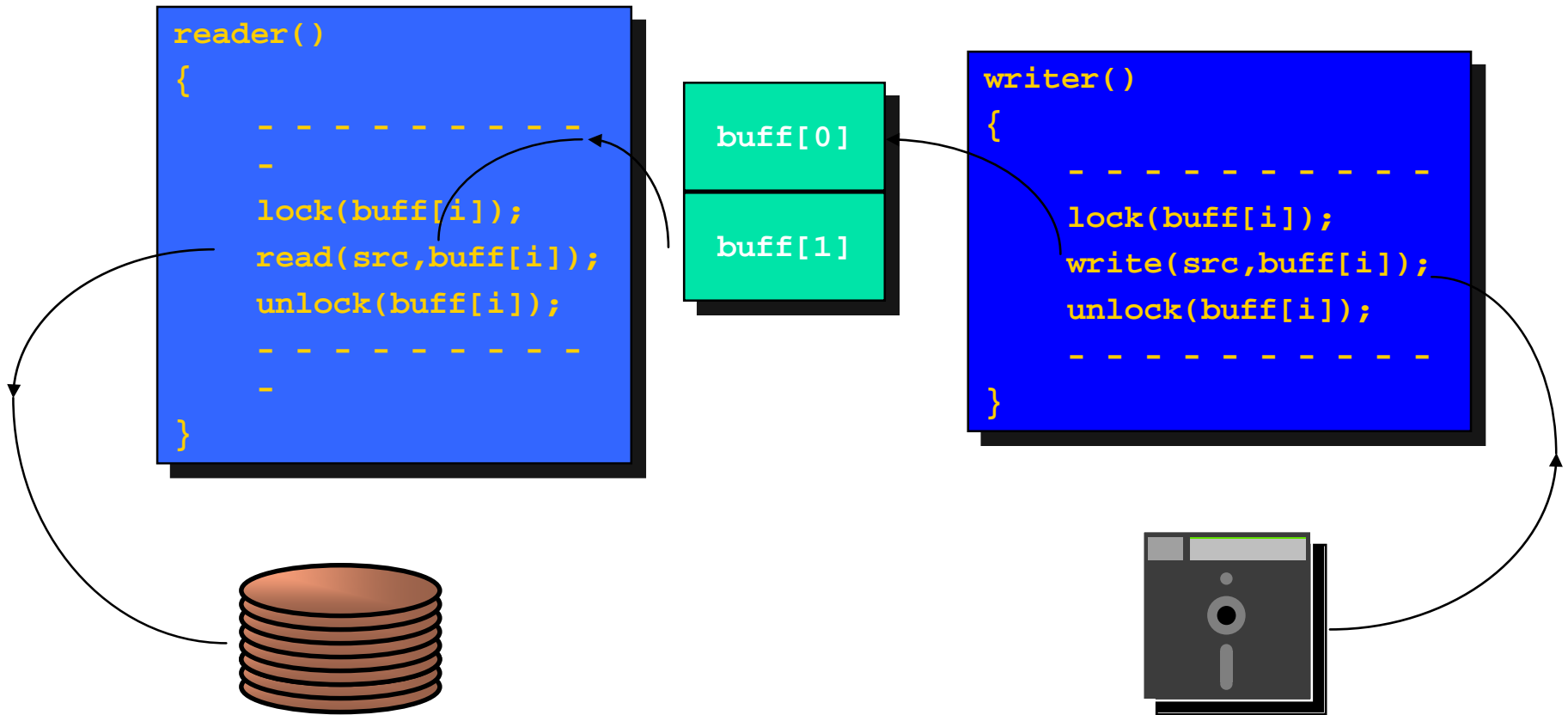
Modern Applications need Threads (ex1): Editing and Printing documents in background.

Editing Thread

Printing Thread



Multithreaded/Parallel File Copy



**Cooperative Parallel Synchronized
Threads**

Levels of Parallelism

Sockets/
PVM/MPI



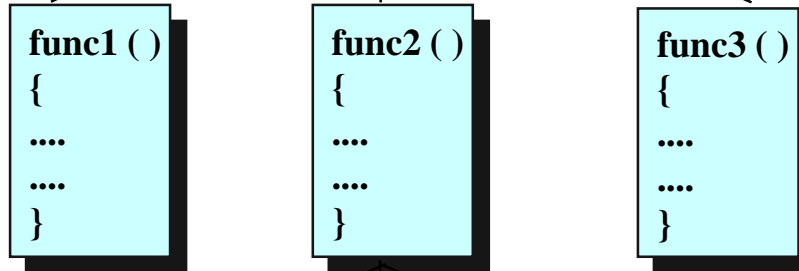
Code-Granularity

Code Item

Large grain
(task level)

Program

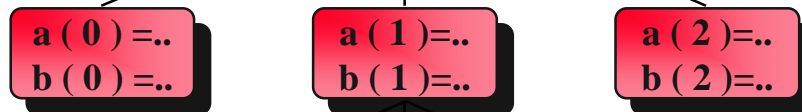
Threads



Medium grain
(control level)

Function (thread)

Compilers



Fine grain
(data level)

Loop (Compiler)

CPU



Very fine grain
(multiple issue)

With hardware

What are Threads?

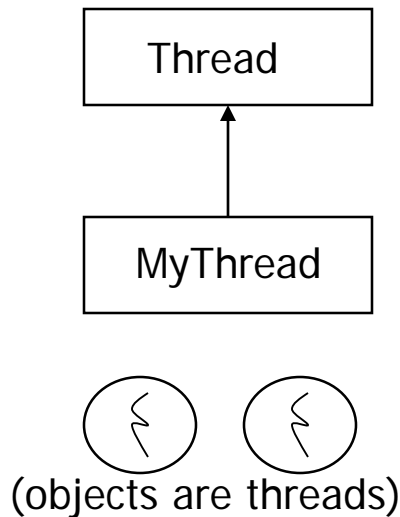
- A piece of code that run in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are being extensively used express concurrency on both single and multiprocessors machines.
- Programming a task having multiple threads of control – Multithreading or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

Java Threads

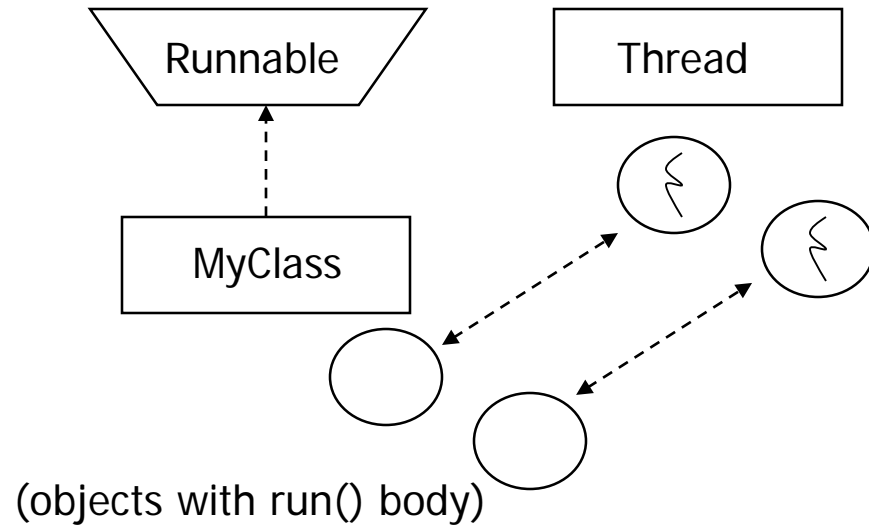
- Java has built in thread support for Multithreading
- Synchronization
- Thread Scheduling
- Inter-Thread Communication:
 - `currentThread` `start` `setPriority`
 - `yield` `run` `getPriority`
 - `sleep` `stop` `suspend`
 - `resume`
- Java Garbage Collector is a low-priority thread.

Threading Mechanisms...

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



[a]



[b]

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() {  
        System.out.println(" this thread is running ... ");  
    }  
}
```

```
class ThreadEx1 {  
    public static void main(String [] args ) {  
        MyThread t = new MyThread();  
        t.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:

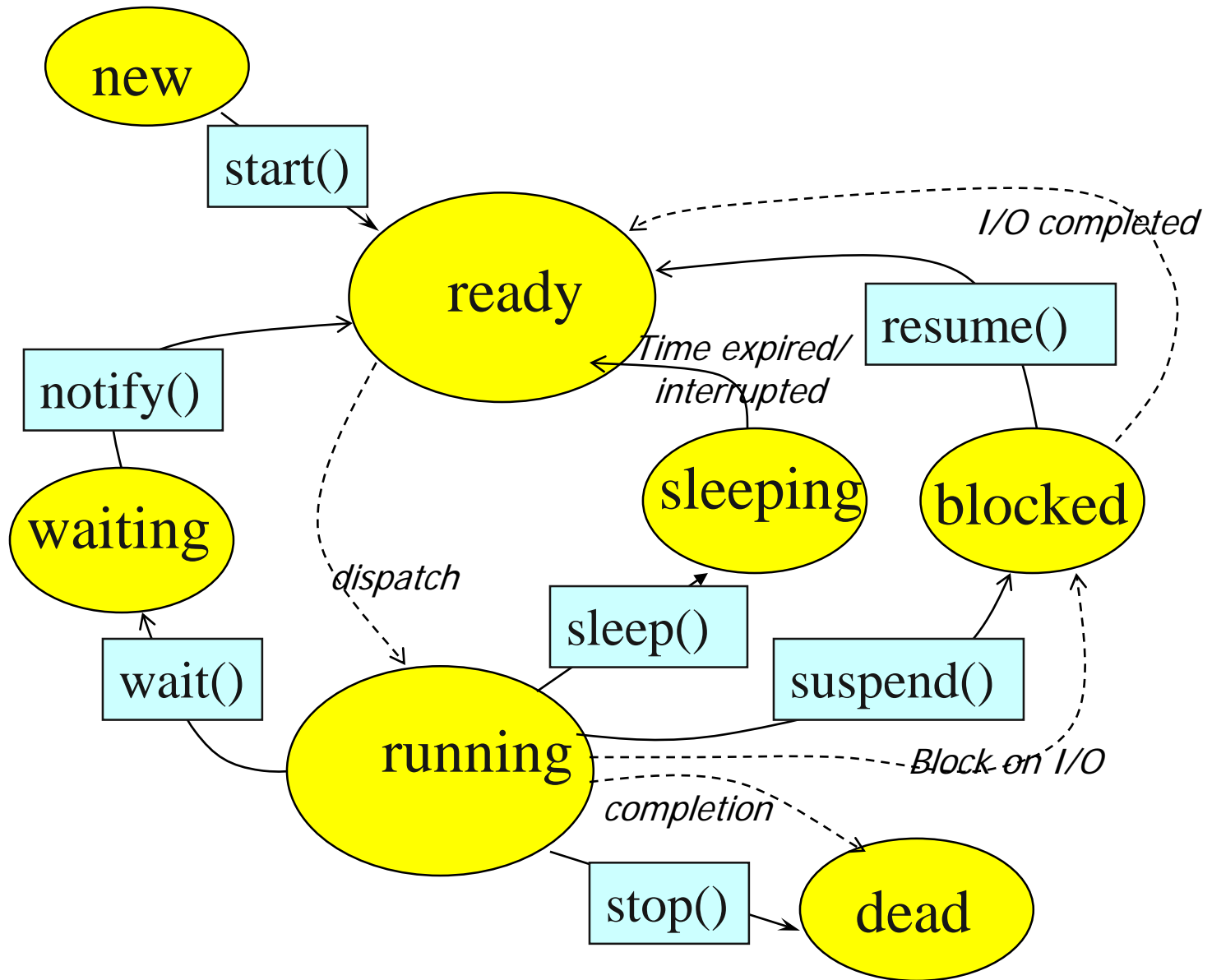
```
thr1.start();
```


An example

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

```
class ThreadEx2 {  
    public static void main(String [] args ) {  
        Thread t = new Thread(new MyThread());  
        t.start();  
    }  
}
```

Life Cycle of Thread



Exercise

- 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[]) throws InterruptedException
■     {
■         new A().start();
■         Thread.sleep(1000);
■         new B().start();
■         Thread.sleep(1000);
■         new C().start();
■     }
■ }
```

Run 1

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

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