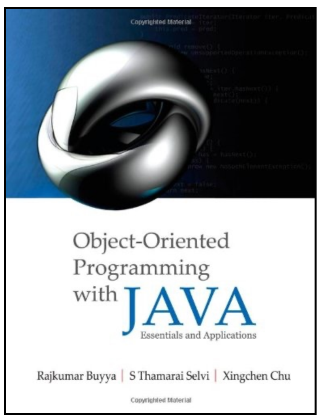
OS Support for Building Distributed Applications: Multithreaded Programming using Java Threads



Dr. Rajkumar Buyya

Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory

School of Computing and Information Systems

The University of Melbourne, Australia

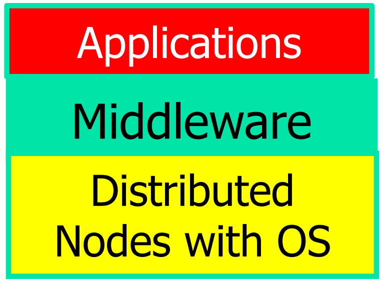
http://www.buyya.com

Outline

- Introduction to Middleware
- Thread Applications
- Defining Threads
- Java Threads and States
- Architecture of Multithreaded servers
- Threads Synchronization
- Summary

Introduction

- Middleware is a layer of software (system) between Applications and Operating System (OS) powering the nodes of a distributed system.
- The OS facilitates:
  - Encapsulation and protection of resources inside servers;
  - Invocation of mechanisms required to access those resources including concurrent access/processing.

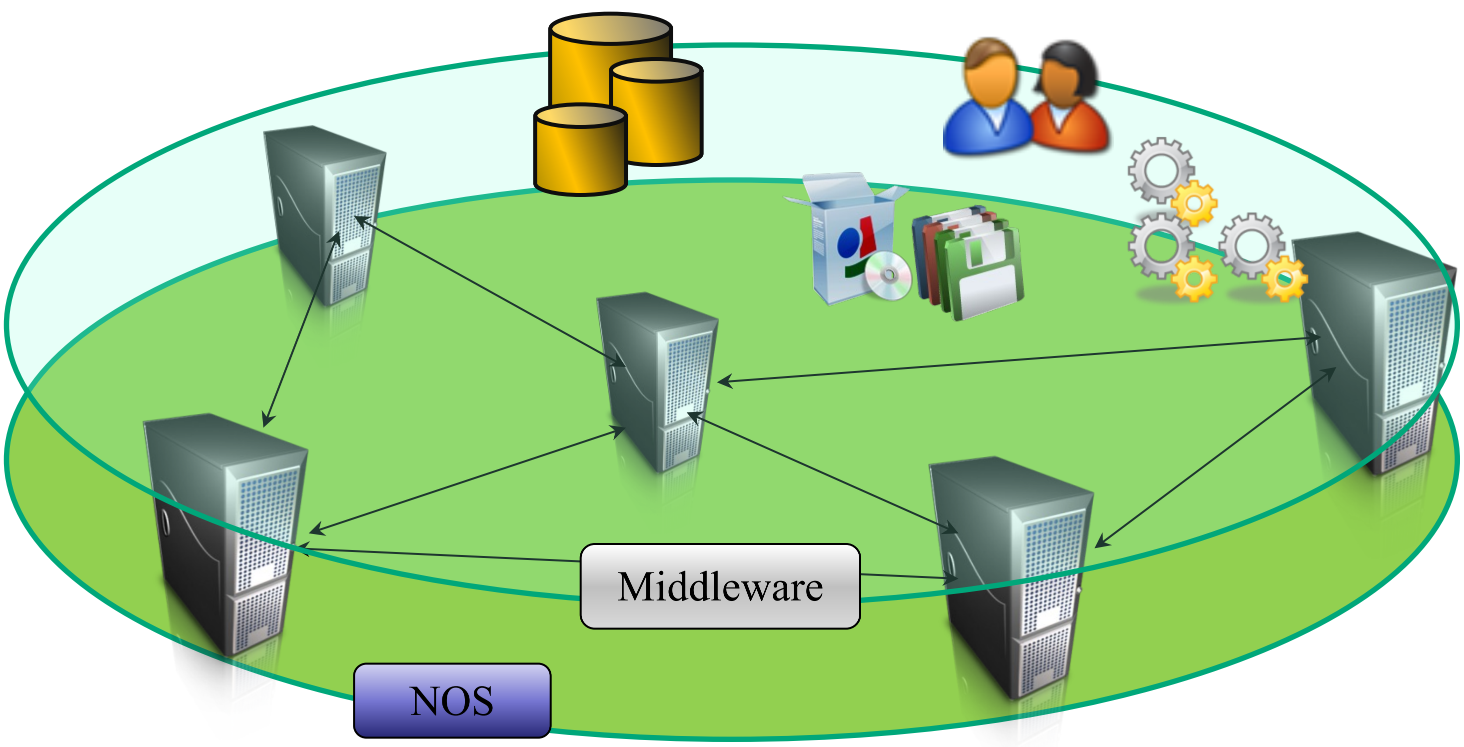


Middleware and Network Operating System (NOS)

- Many DOS (Distributed OS) have been investigated, but there are none in general/wide use. But NOS are in wide use for various reasons both technical and non-technical.
  - Users have much invested in their application software; they will not adopt a new OS that will not run their applications.
  - Users tend to prefer to have a degree of autonomy of their machines, even in a closely knit organisation.
- A combination of middleware and NOSs provides an acceptable balance between the requirement of autonomy and network transparency.
  - NOS allows users to run their favorite word processor.
  - Middleware enables users to take advantage of services that become available in their distributed systems.

Introducing a middleware

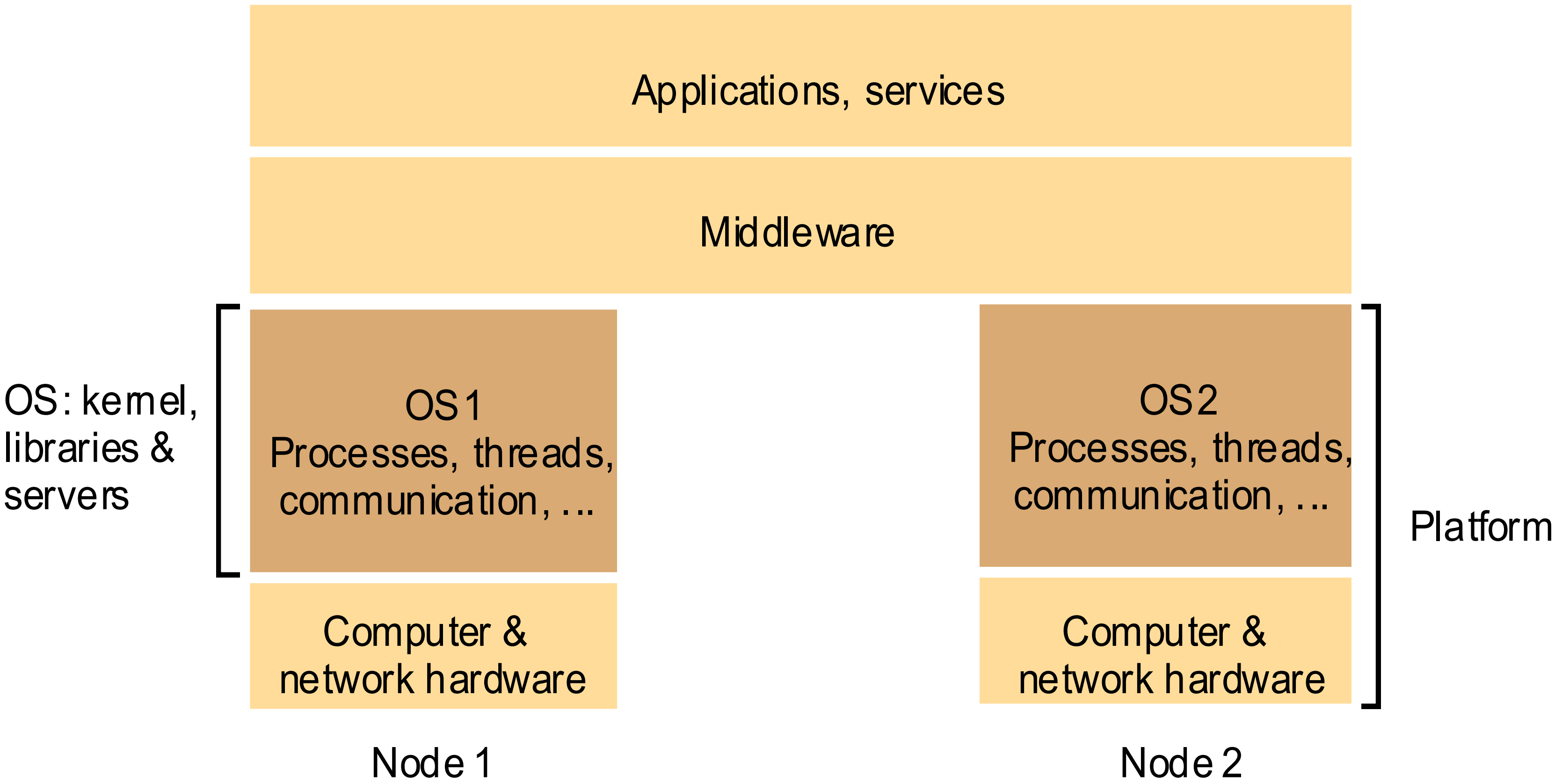
- Building Distributed Systems
  - DOS or NOS are not enough to build a DS!
  - NOS are a good starting point but ….
  - … we need an additional layer "gluing" all together



Building Distributed Systems

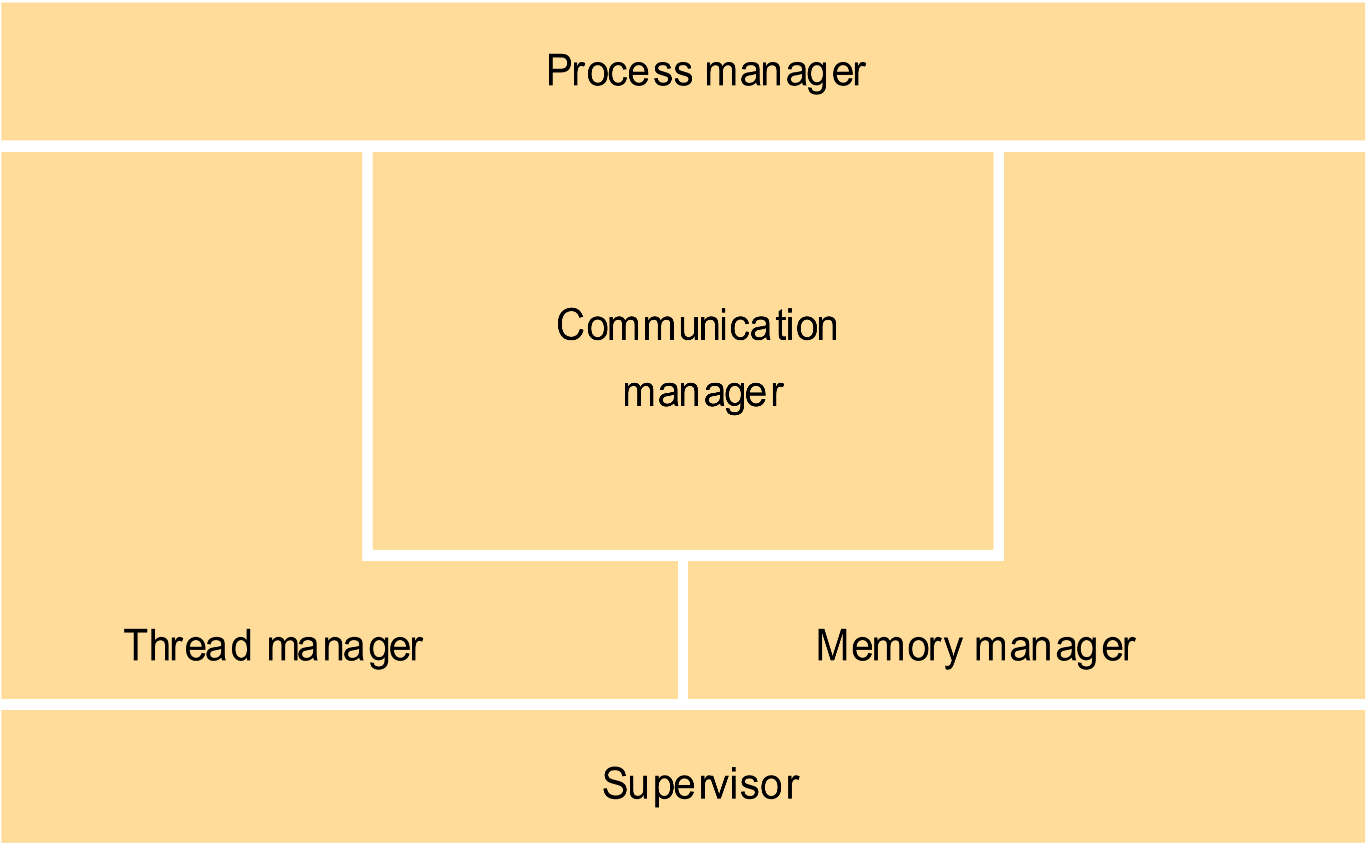
- Middleware
  - High-level features for DS
    - Communication
    - Management
    - Application specific
  - Uniform layer where to build DS services
  - Runtime environment of applications
- Operating System
  - Low / medium level (core) features
    - Process / threads management
    - Local hardware (CPU, disk, memory)
    - Security (users, groups, domain, ACLs)
    - Basic networking

Operating system layers and Middleware



- Unix and Windows are two examples of Network Operating Systems – have a networking capability built into them and so can be used to access remote resources using basic services such as rlogin, telnet.

Core OS components and functionality

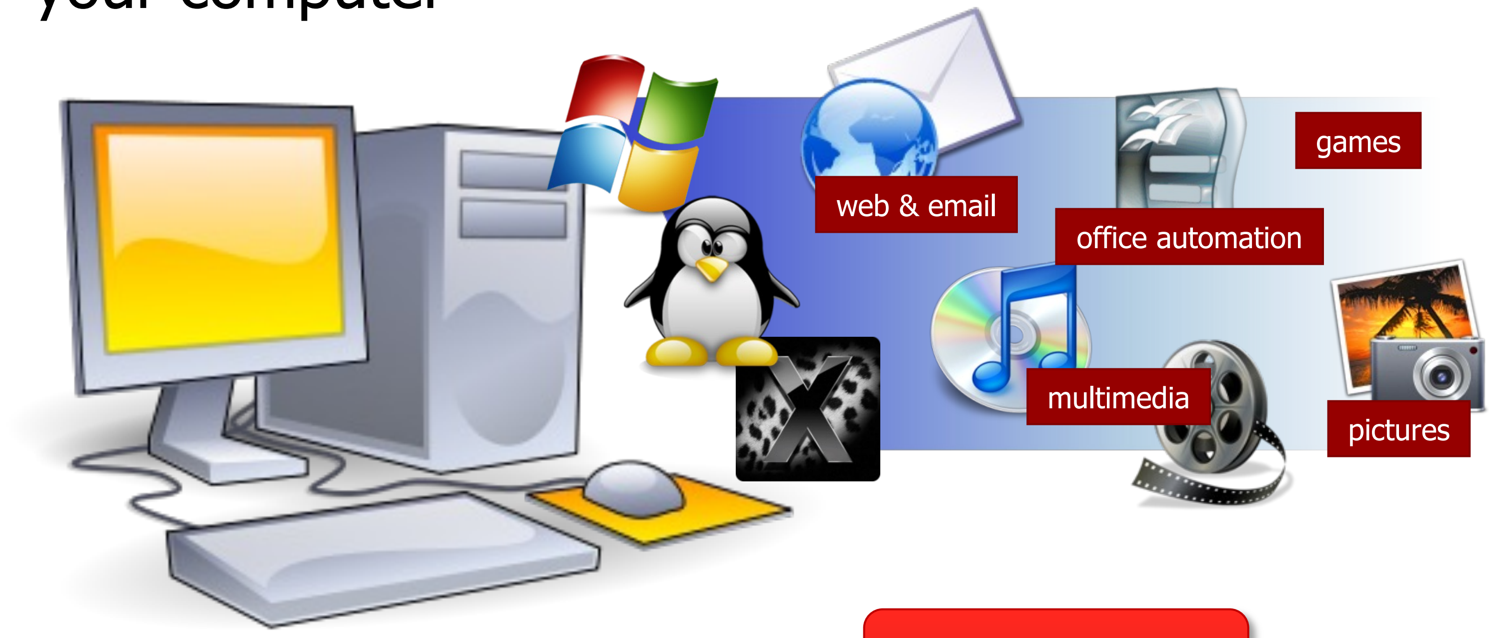


Threaded Applications

- Modern Applications and Systems
  - Operating System Level
    - Multitasking: multiple applications running at once
  - Application Level
    - Multithreading: multiple operations performed at the same time within an application.
  - Bottom Line:
    - Illusion of concurrency

Threaded Applications

- Modern Systems
  - Multiple applications run concurrently!
  - This means that… there are multiple processes on your computer



Multitasking

A single threaded program

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

begin

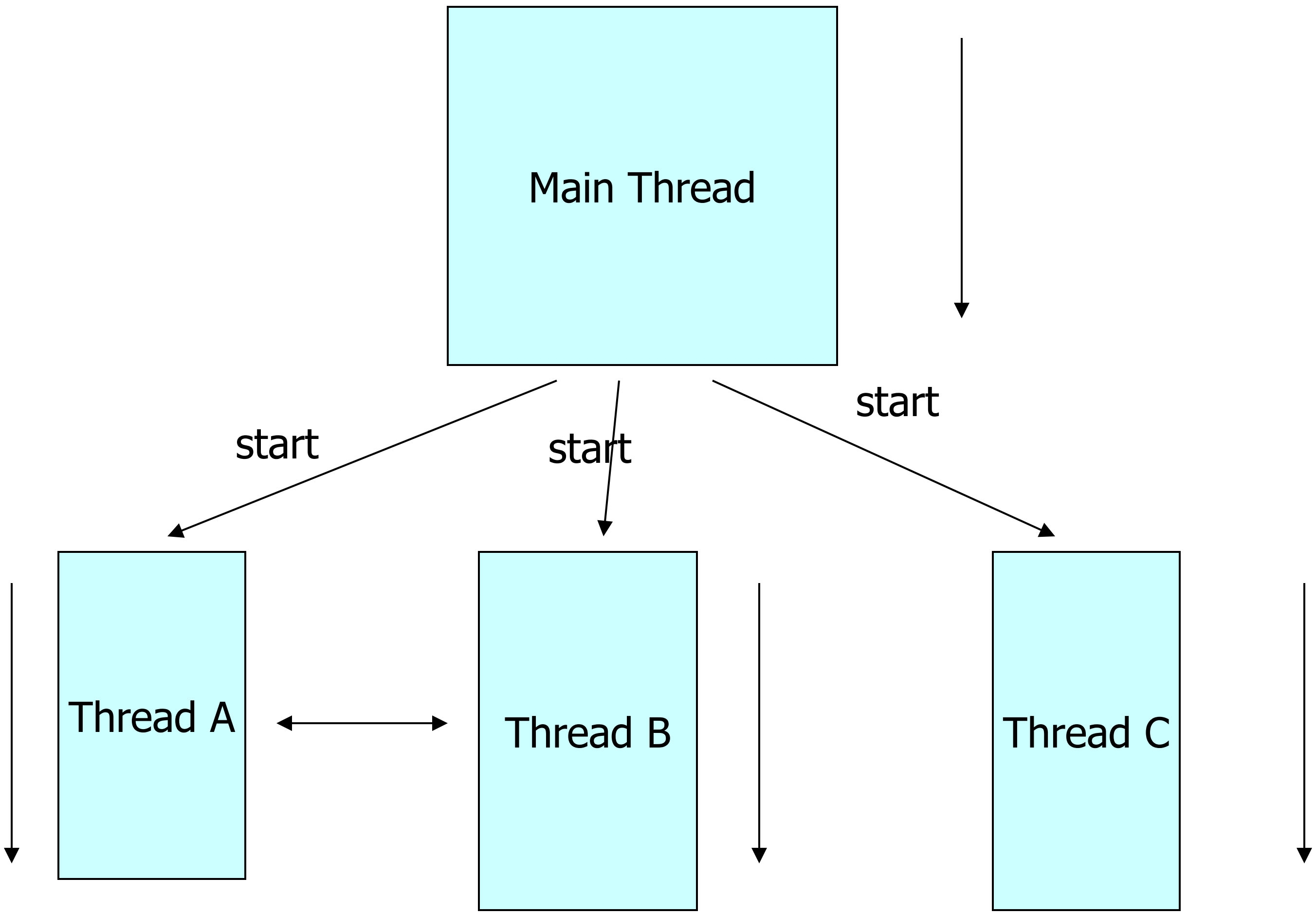
body

end

Threaded Applications

- Modern Systems
  - Applications perform many tasks at once!
  - This means that… there are multiple <u>threads</u> within a single process.

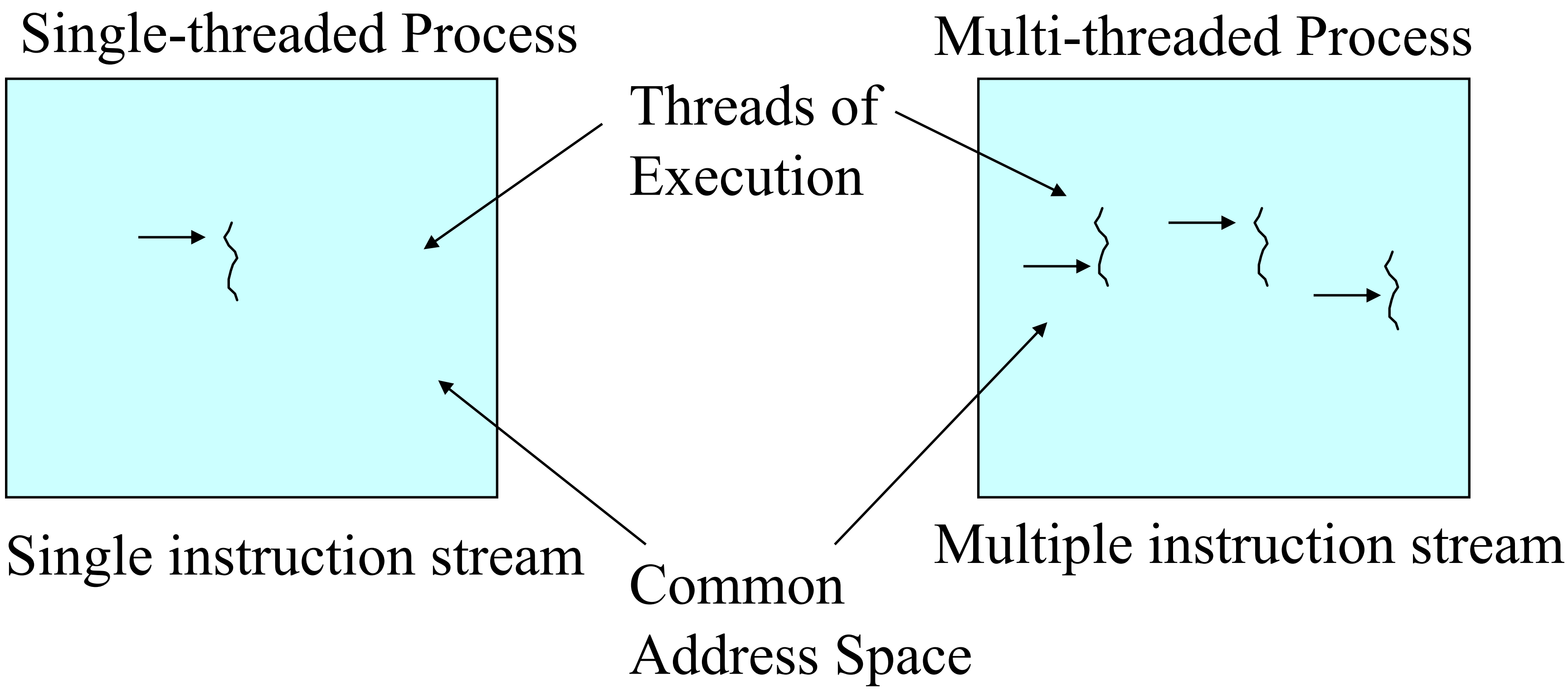
A Multithreaded Program



Threads may switch or exchange data/results

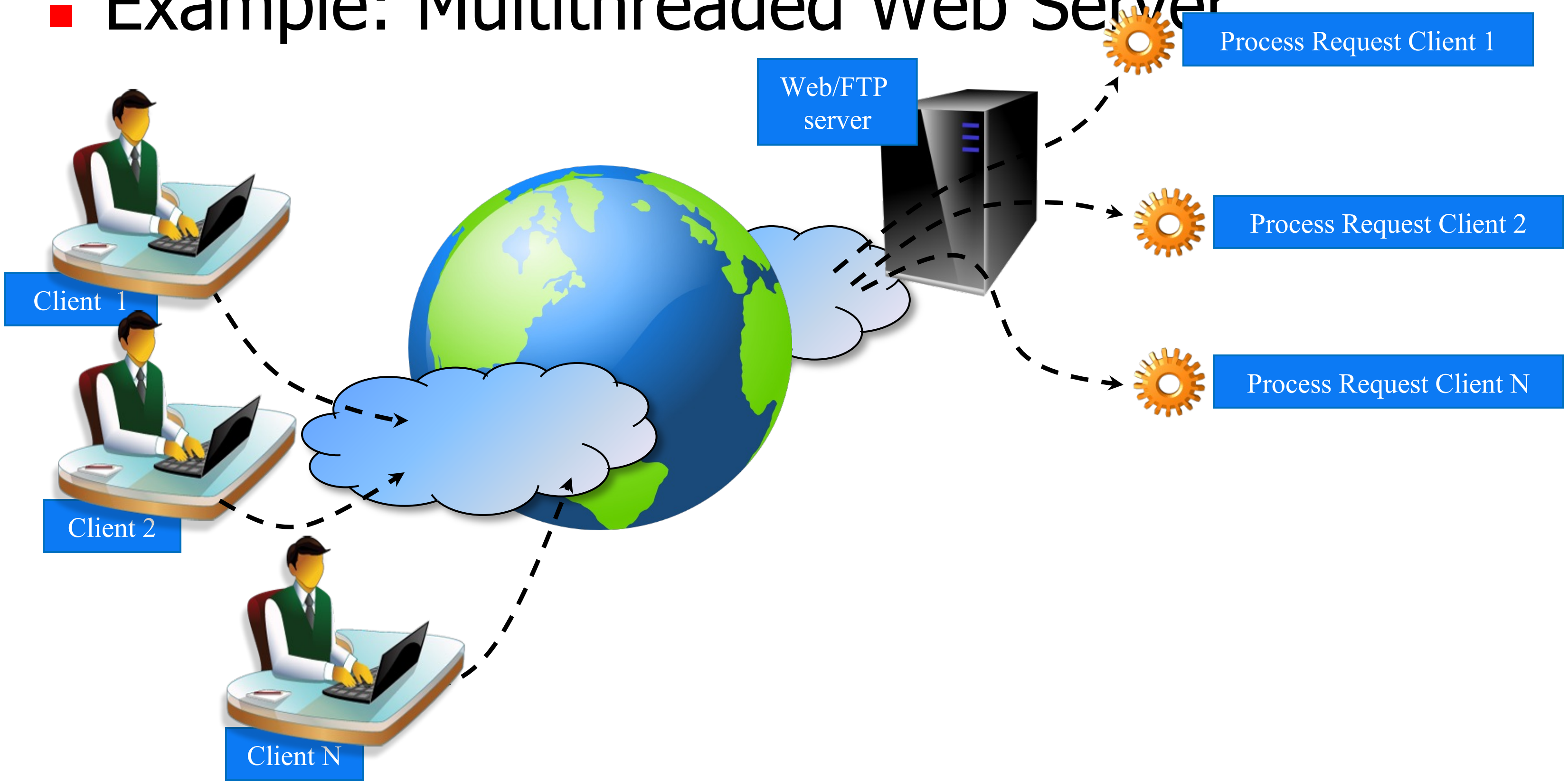
Single and Multithreaded Processes

threads are light-weight processes within a process



Multithreaded Server: For Serving Multiple Clients Concurrently

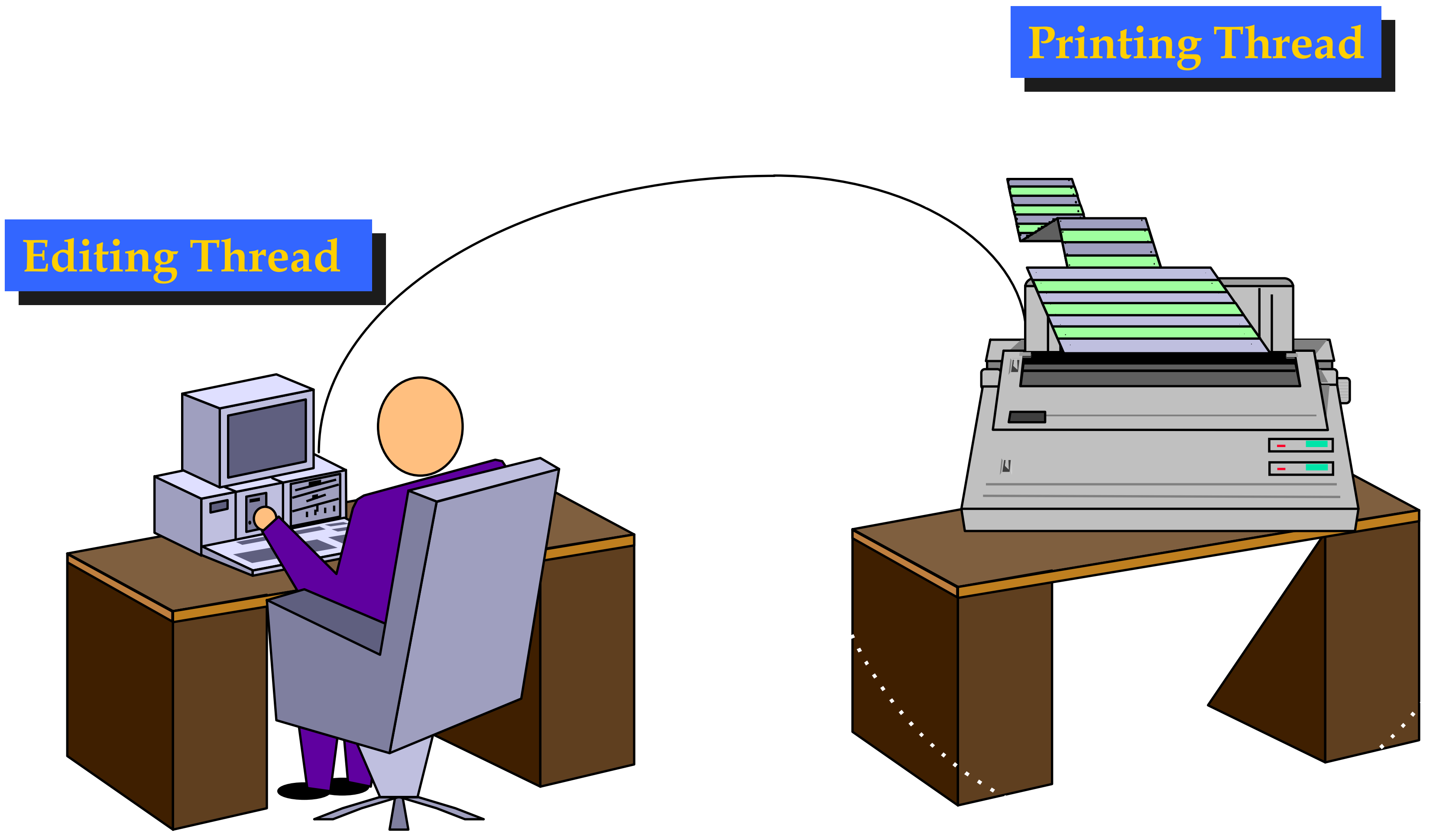
- Modern Applications
  - Example: Multithreaded Web Server



Threaded Applications

- Modern Applications
  - Example: Internet Browser + Youtube

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

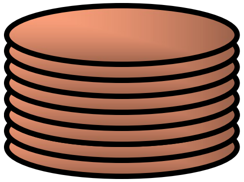


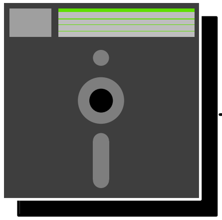
Multithreaded/Parallel File Copy

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

buff[0]
buff[1]

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





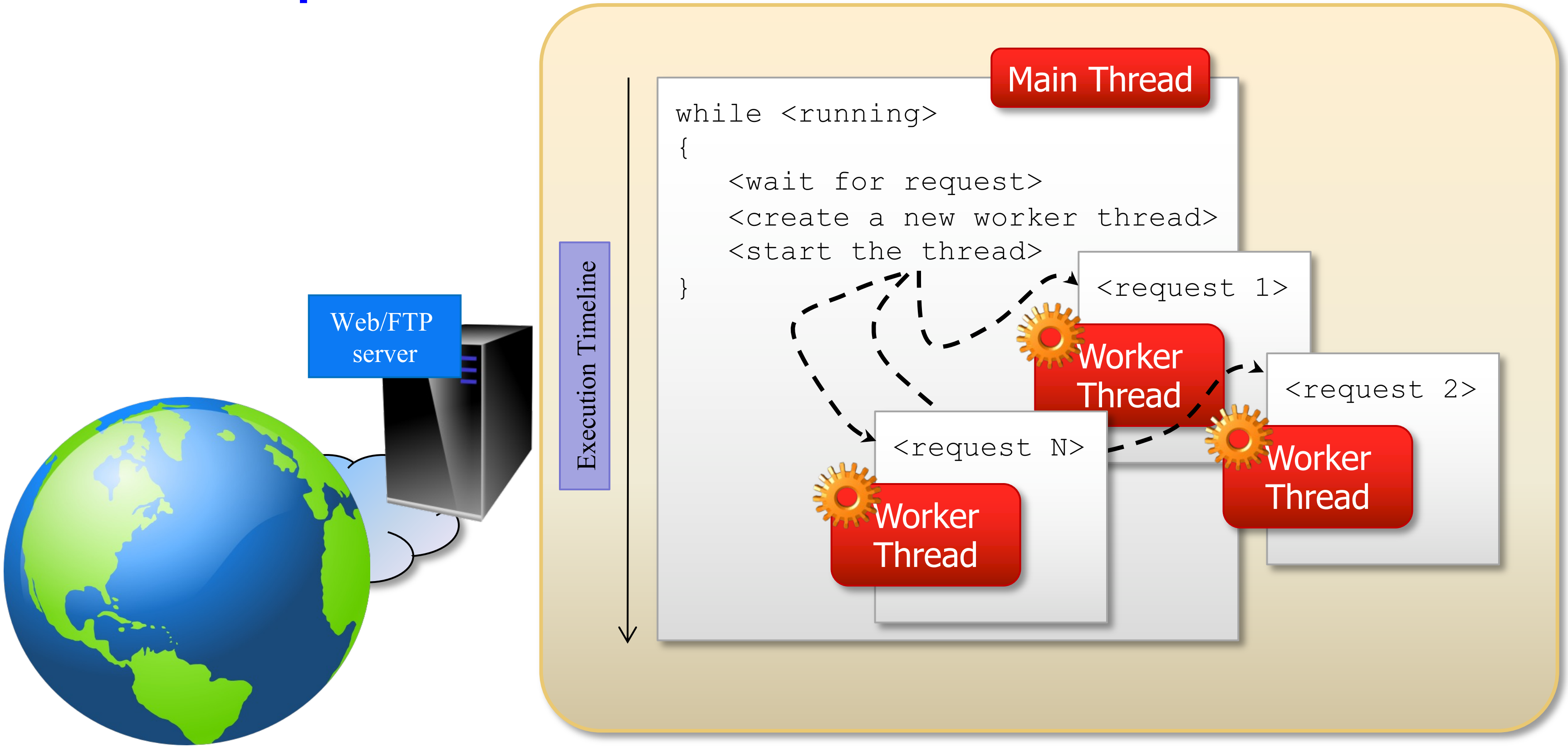
Cooperative Parallel Synchronized Threads

Defining Threads

- Applications – Threads are used to perform:
  - Parallelism and concurrent execution of independent tasks / operations.
  - Implementation of reactive user interfaces.
  - Non blocking I/O operations.
  - Asynchronous behavior.
  - Timer and alarms implementation.

Defining Threads

- Example: Web/FTP Server



Defining Threads

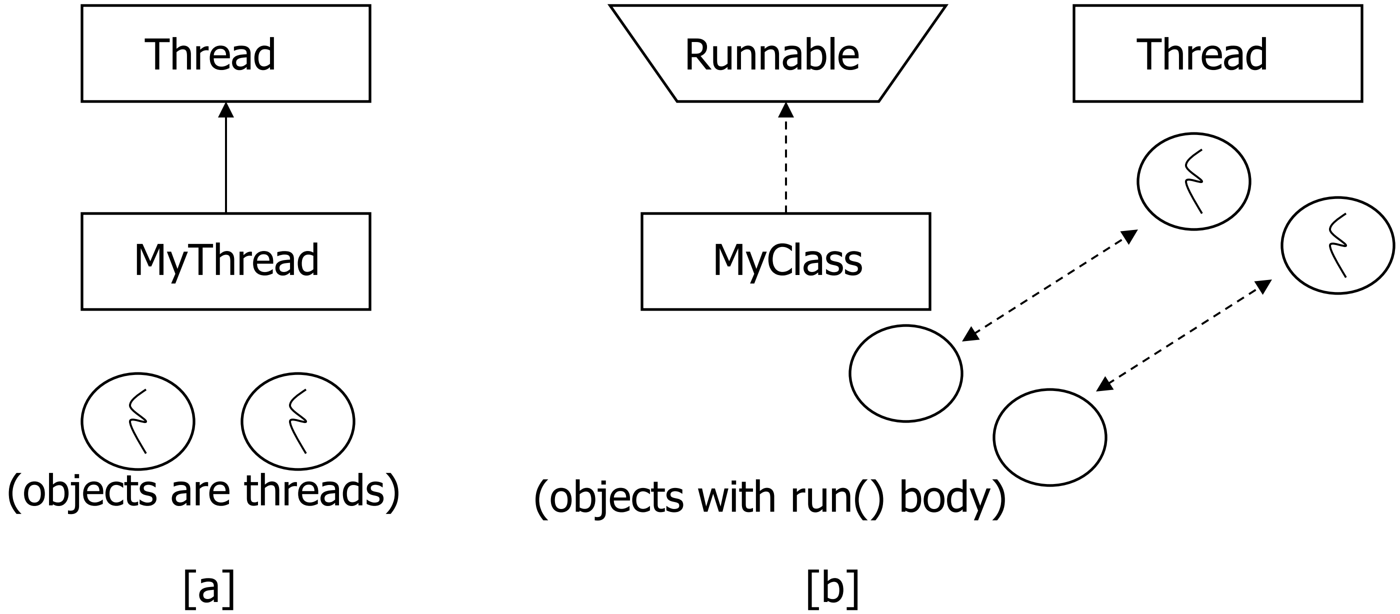
- A Thread is a piece of code that runs in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are used to express concurrency on both single and multiprocessors machines.
- Programming a task having multiple threads of control is called: Multithreading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

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.

Threading Mechanisms...

- 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() {
                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 );
```

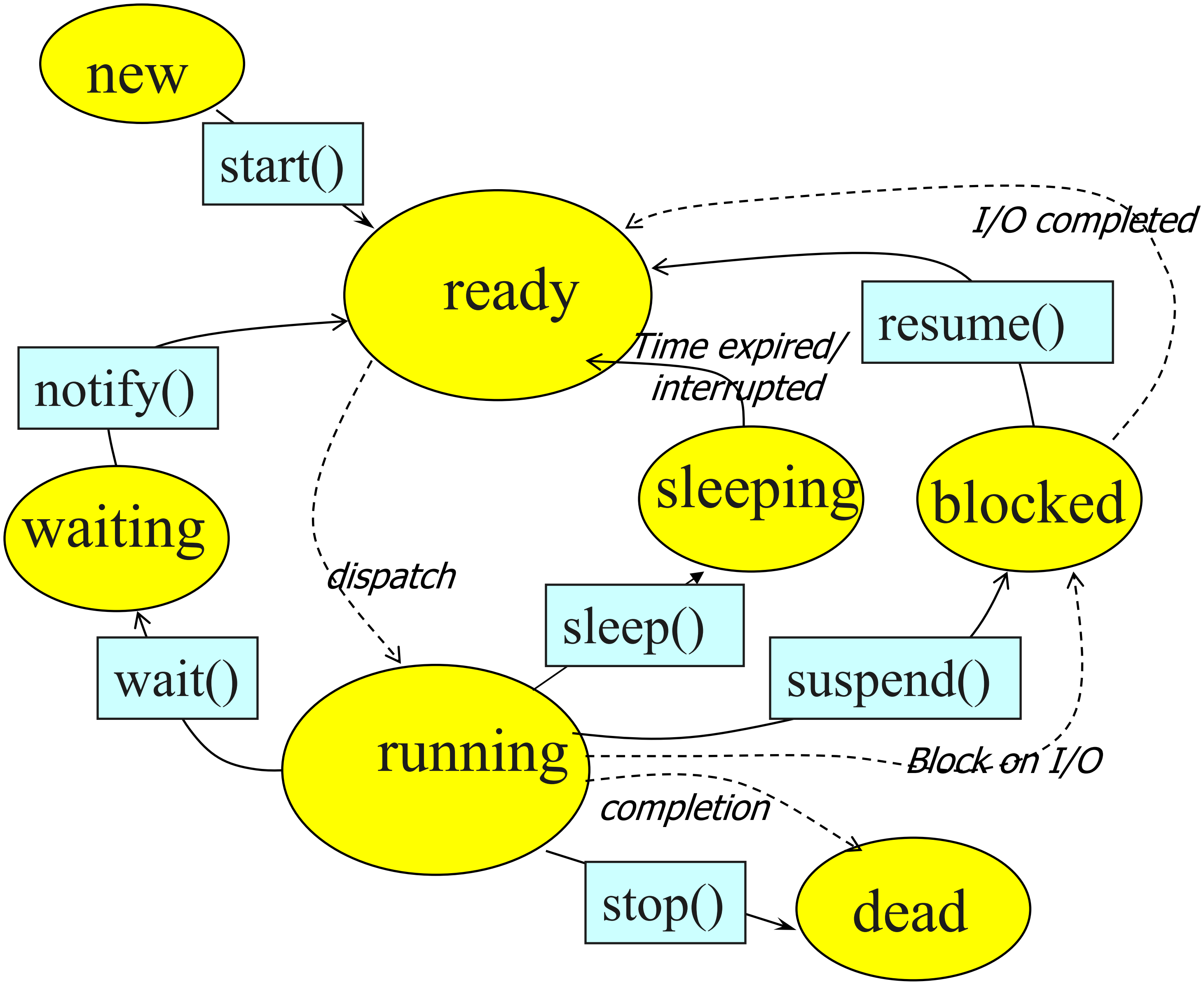
- Start Execution:

```
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



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");
      }
}
```

Three threads example

```
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

Run 2

- [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
```

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

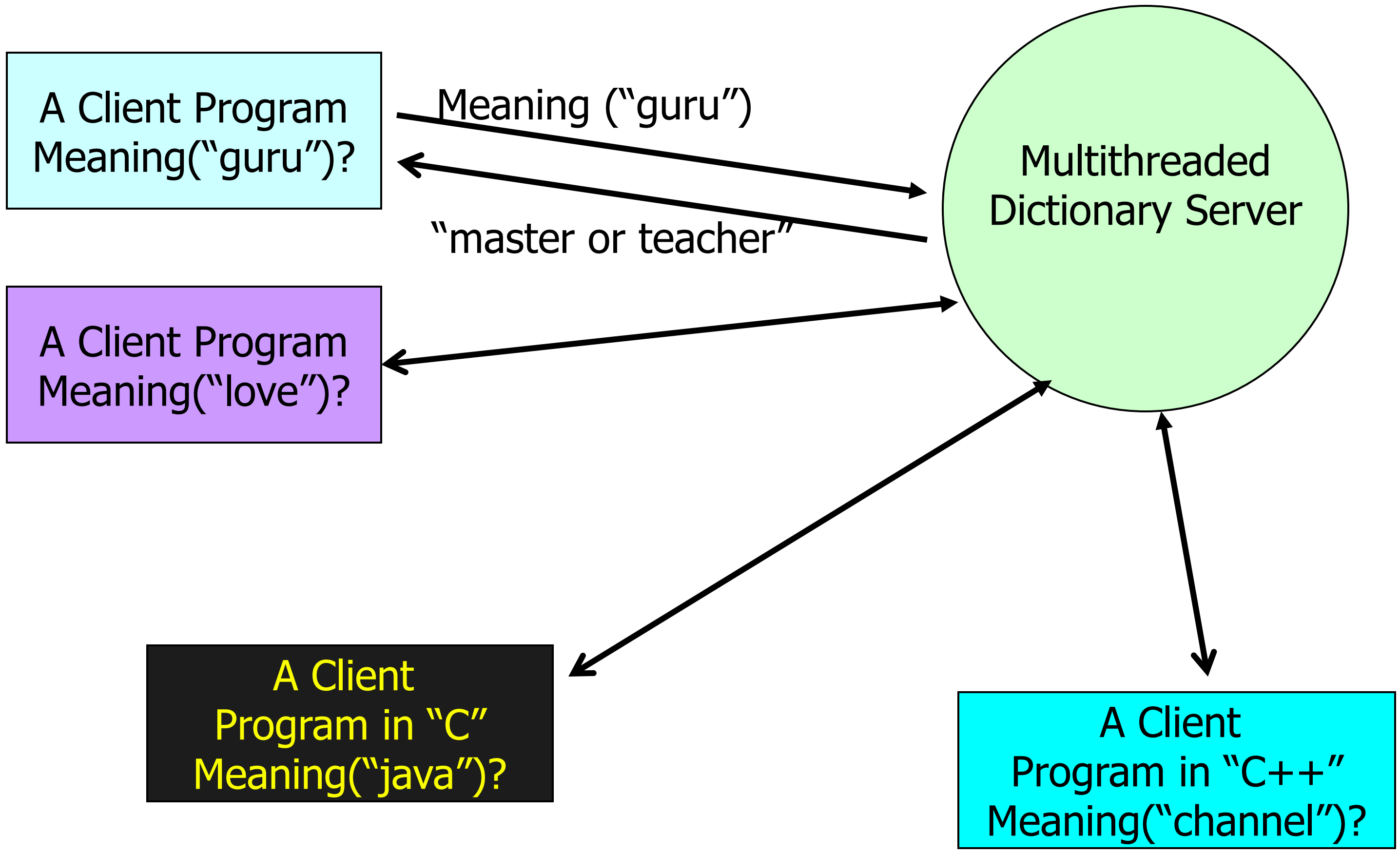
Thread Priority Example

```
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");
          }
}
```

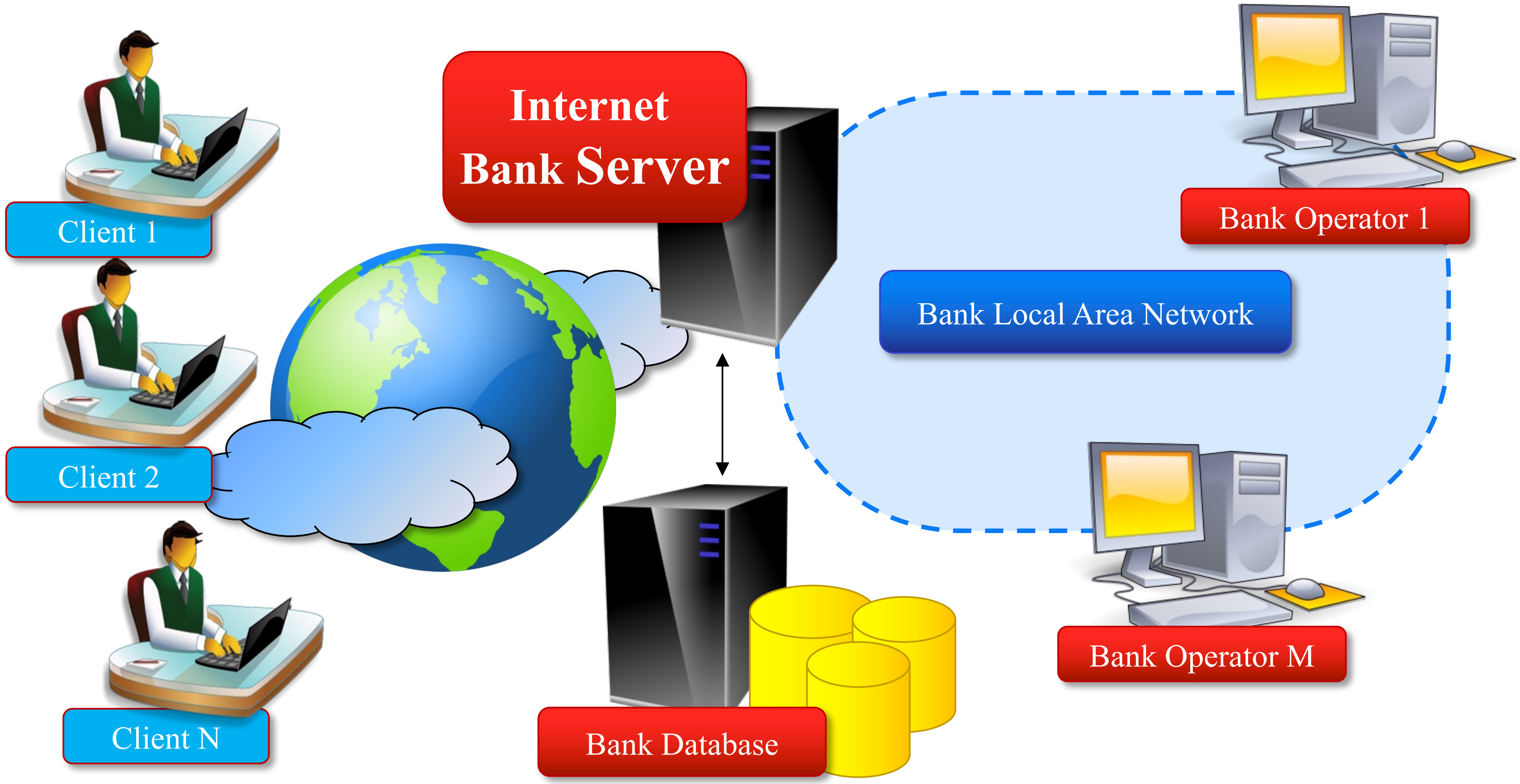
Assignment 1 at a Glance: Multithreaded Dictionary Server – Using Sockets and Threads



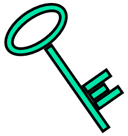
Accessing Shared Resources

- 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



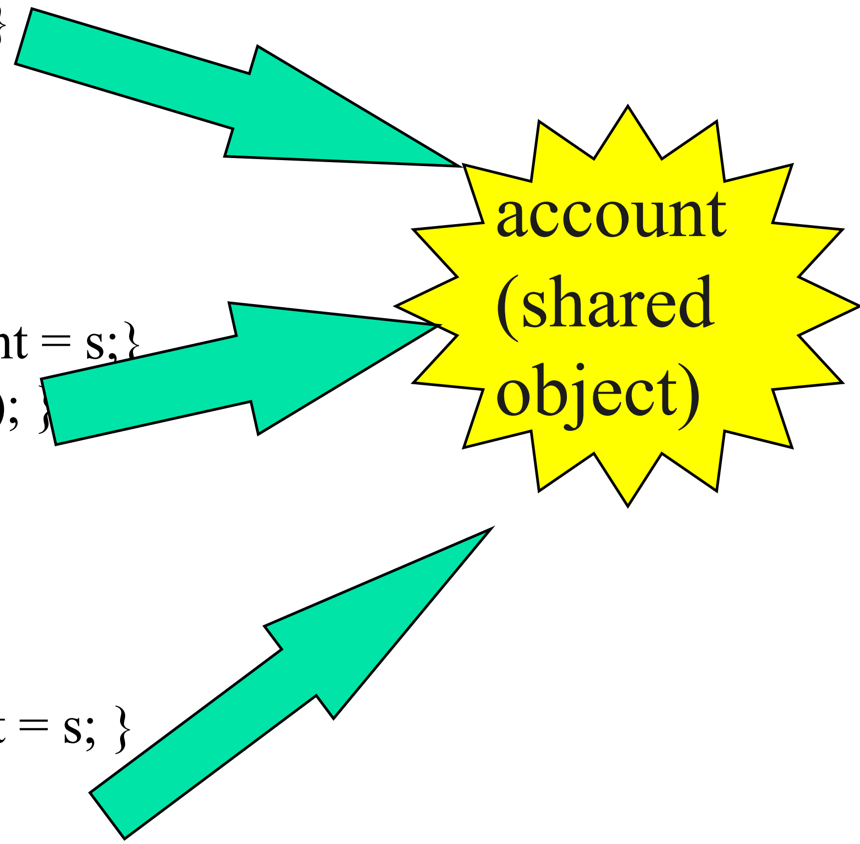
- If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.
- This can be prevented by synchronising access to the data.
- Use “synchronized” method:
  - public synchronized void update()
  - {
    - …
  - }

the driver: 3 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
```



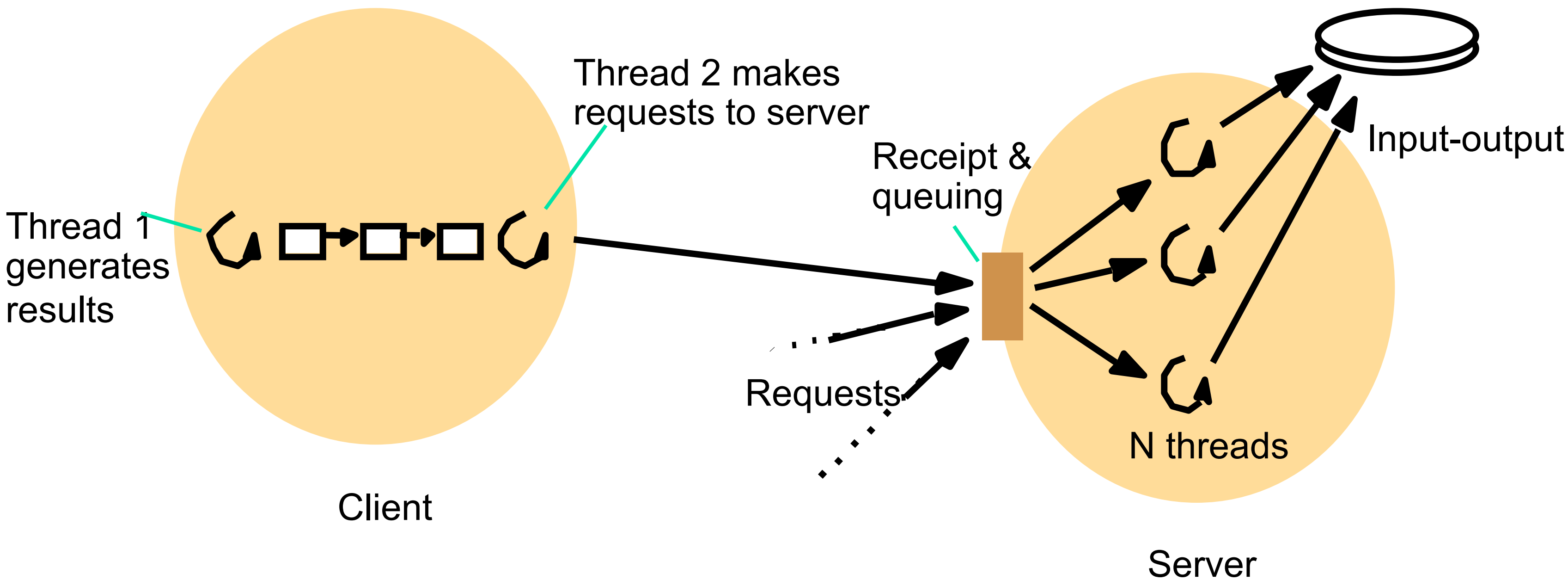
Monitor (shared object access): serializes operation on shared objects

```
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.
     }
}
```

Architecture for Multithread Servers

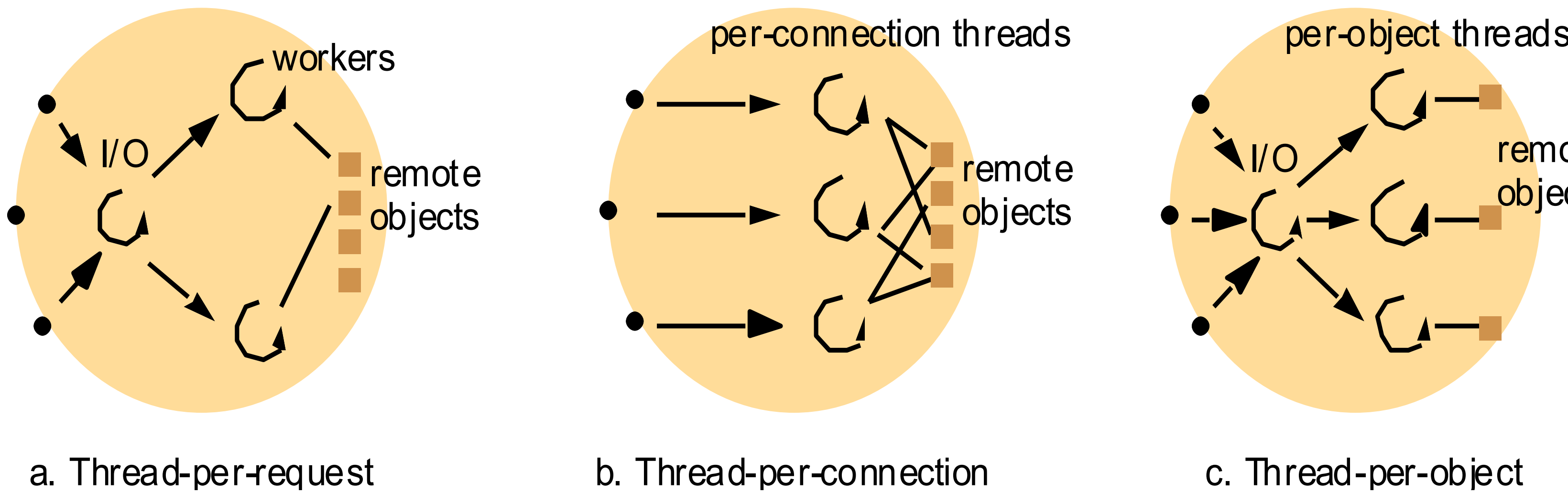
- Multithreading enables servers to maximize their throughput, measured as the number of requests processed per second.
- Threads may need to treat requests with varying priorities:
  - A corporate server could prioritize request processing according to class of customers.
- Architectures:
  - Worker pool
  - Thread-per-request
  - Thread-per-connection
  - Thread-per-object

Client and server with threads (worker-pool architecture)



- In worker-pool architectures, the server creates **a fixed pool of worker threads** to process requests.
- The module "receipt and queuing" receives requests from sockets/ports and places them on a shared request queue for retrieval by the workers.

Alternative server threading architectures

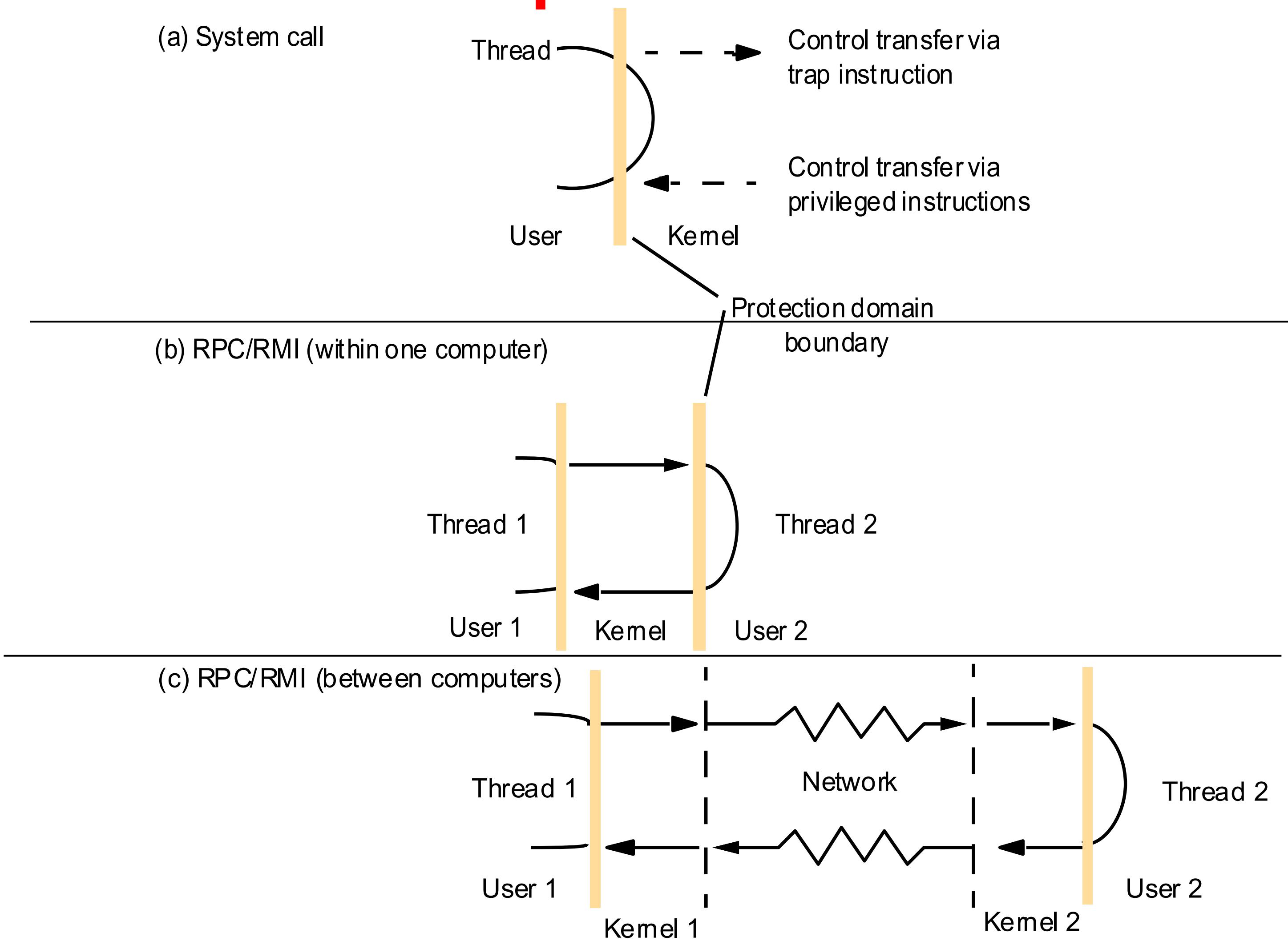


IO Thread creates a new worker thread for each request and worker thread destroys itself after serving the request.

Server associates a Thread with each connection and destroys when client closes the connection.
Client may make many requests over the connection.

Associates Thread with each object. An IO thread receives request and queues them for workers, but this time there is a **per-object queue**.

Invocations between address spaces



Summary

- Operating system provides various types of facilities to support middleware for distributed system:
  - encapsulation, protection, and concurrent access and management of node resources.
- Multithreading enables servers to maximize their throughput, measured as the number of requests processed per second.
- Threads support treating of requests with varying priorities.
- Various types of architectures can be used in concurrent processing:
  - Worker pool
  - Thread-per-request
  - Thread-per-connection
  - Thread-per-object
- Threads need to be synchronized when accessing and manipulating shared resources.
- New OS designs provide flexibility in terms of separating mechanisms from policies.

References

- CDK Book (Text Book)
  - Chapter 7 – "Operating System Support"
- Chapter 14: Multithread Programming
  - R. Buyya, S. Selvi, X. Chu, **"Object Oriented Programming with Java: Essentials and Applications",** McGraw Hill, New Delhi, India, 2009.