CSE 411: DISTRIBUTED COMPUTER SYSTEMS

OS Support for Building Distributed Applications

Multithreaded Programming using Java Threads

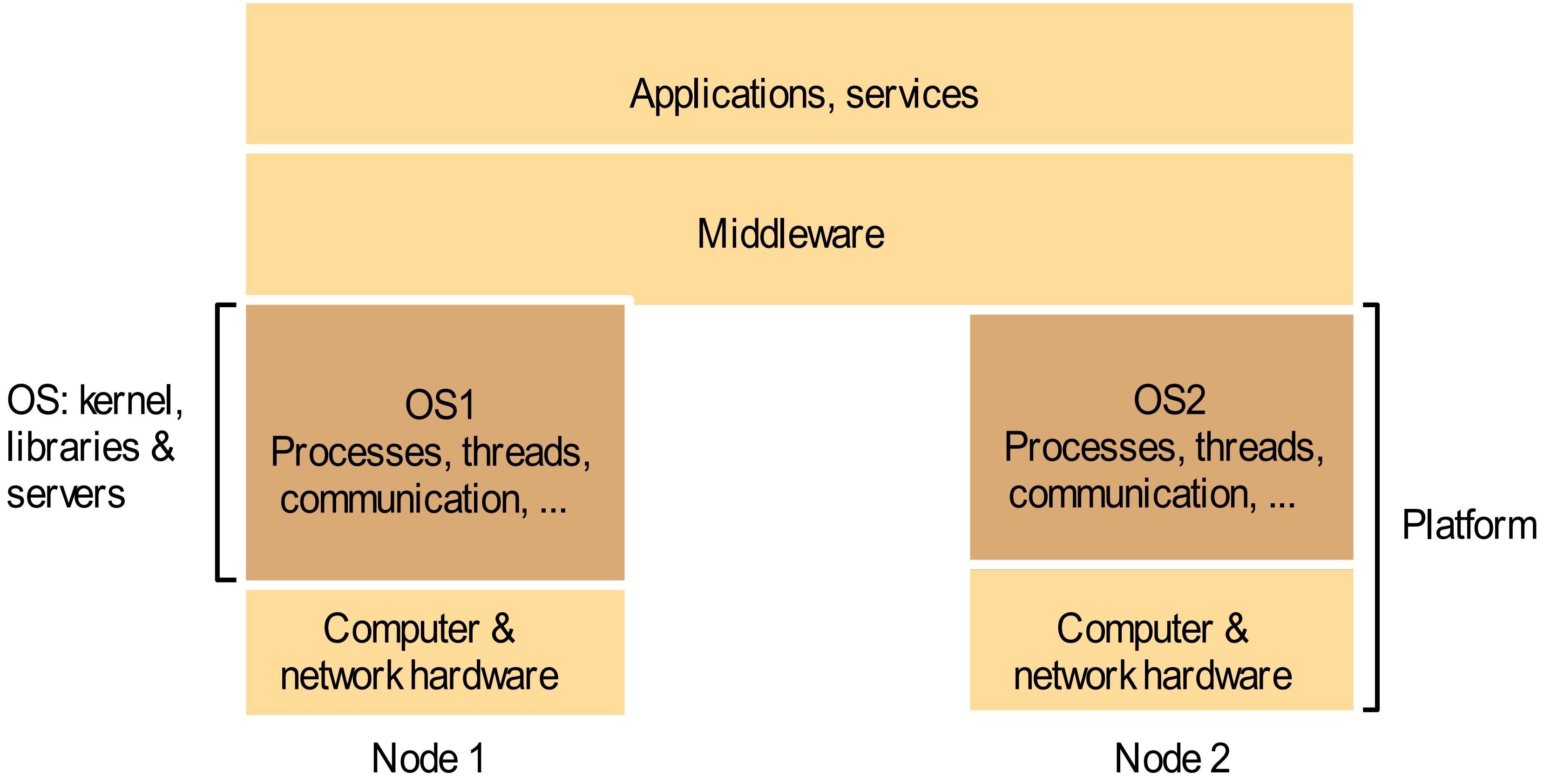
Outline

- Introduction
- Thread Applications
- Defining Threads
- Architecture of Multithreaded servers
- Threads Synchronization
- Summary

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


Applications, services
Middleware
OS: kernel, libraries & servers
OS1 Processes, threads, communication, ...
OS2 Processes, threads, communication, ...
Platform
Computer & network hardware
Computer & network hardware
Node 1
Node 2

- 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, ssh.

Threaded Applications

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

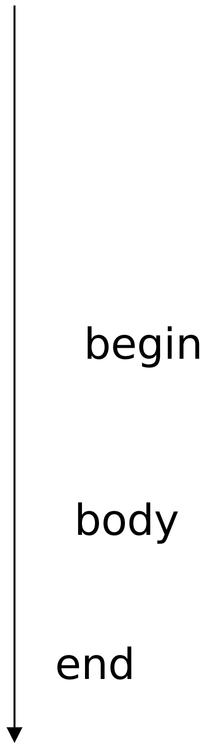
Threaded Applications

- 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(..)
    {
    …
    ..
    }
}
```



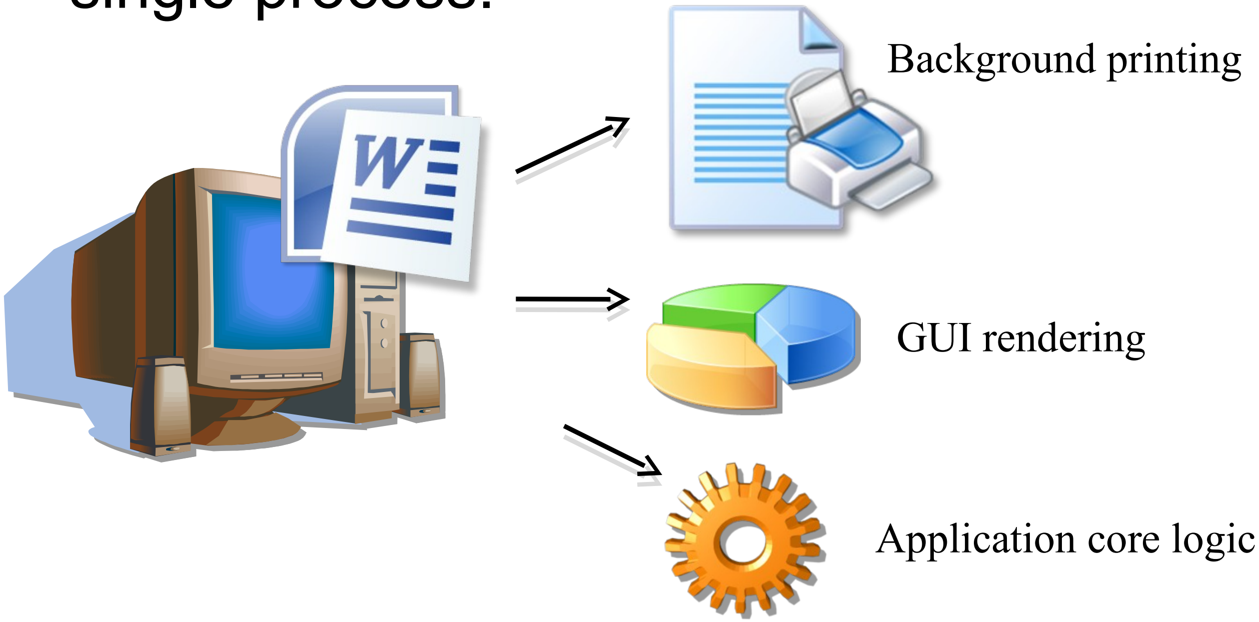
begin

body

end

Threaded Applications

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

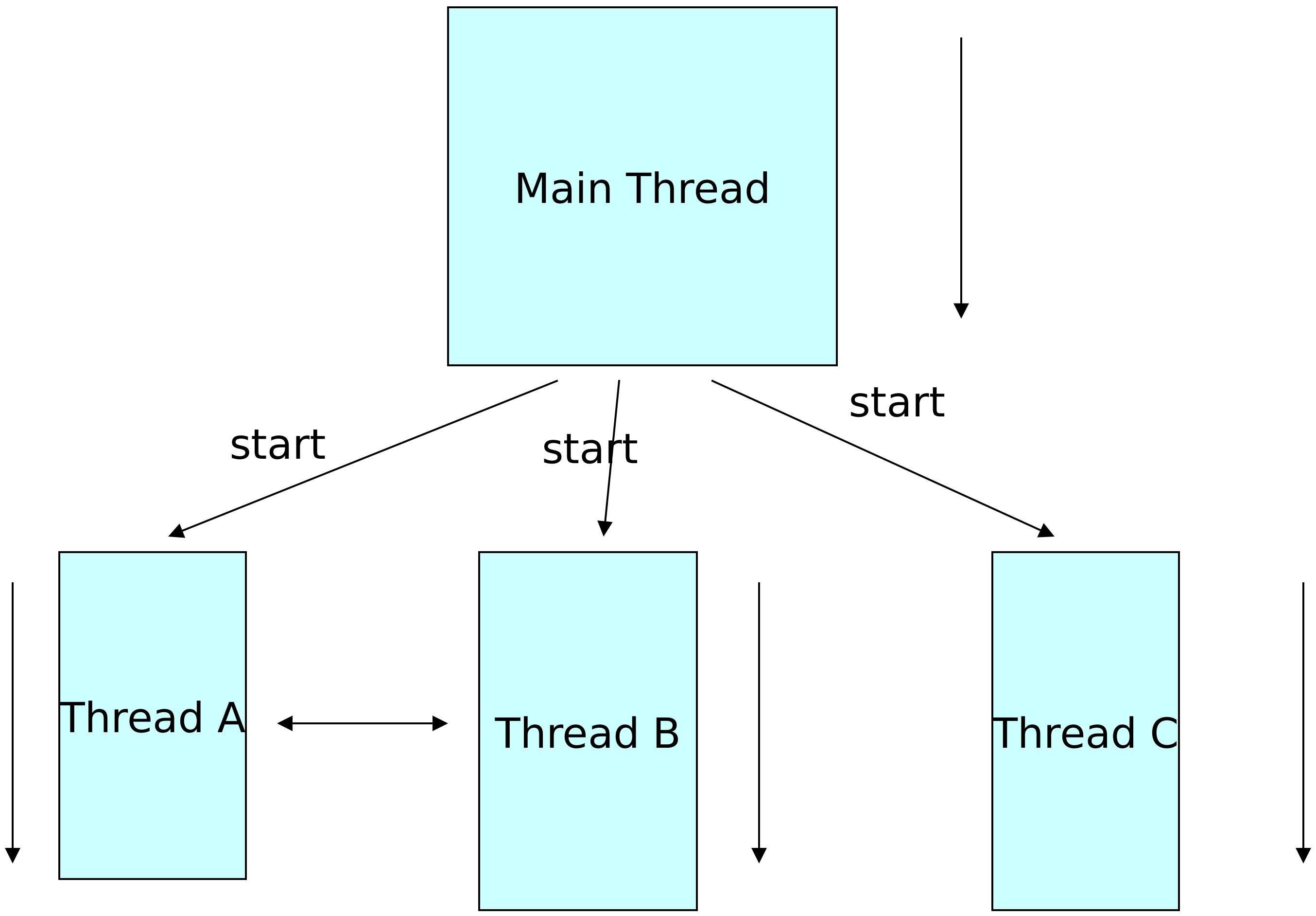


Background printing

GUI rendering

Application core logic

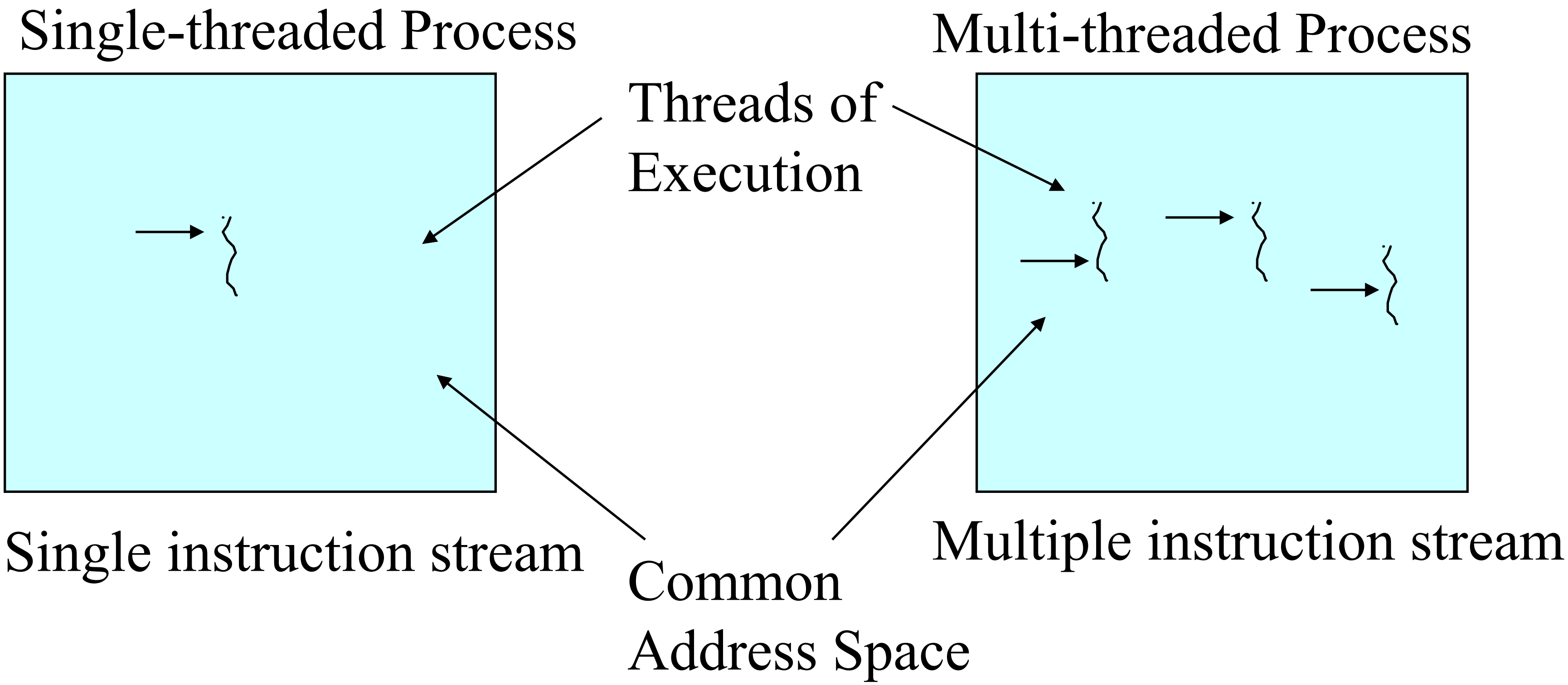
A Multithreaded Program



Threads may switch or exchange data/results

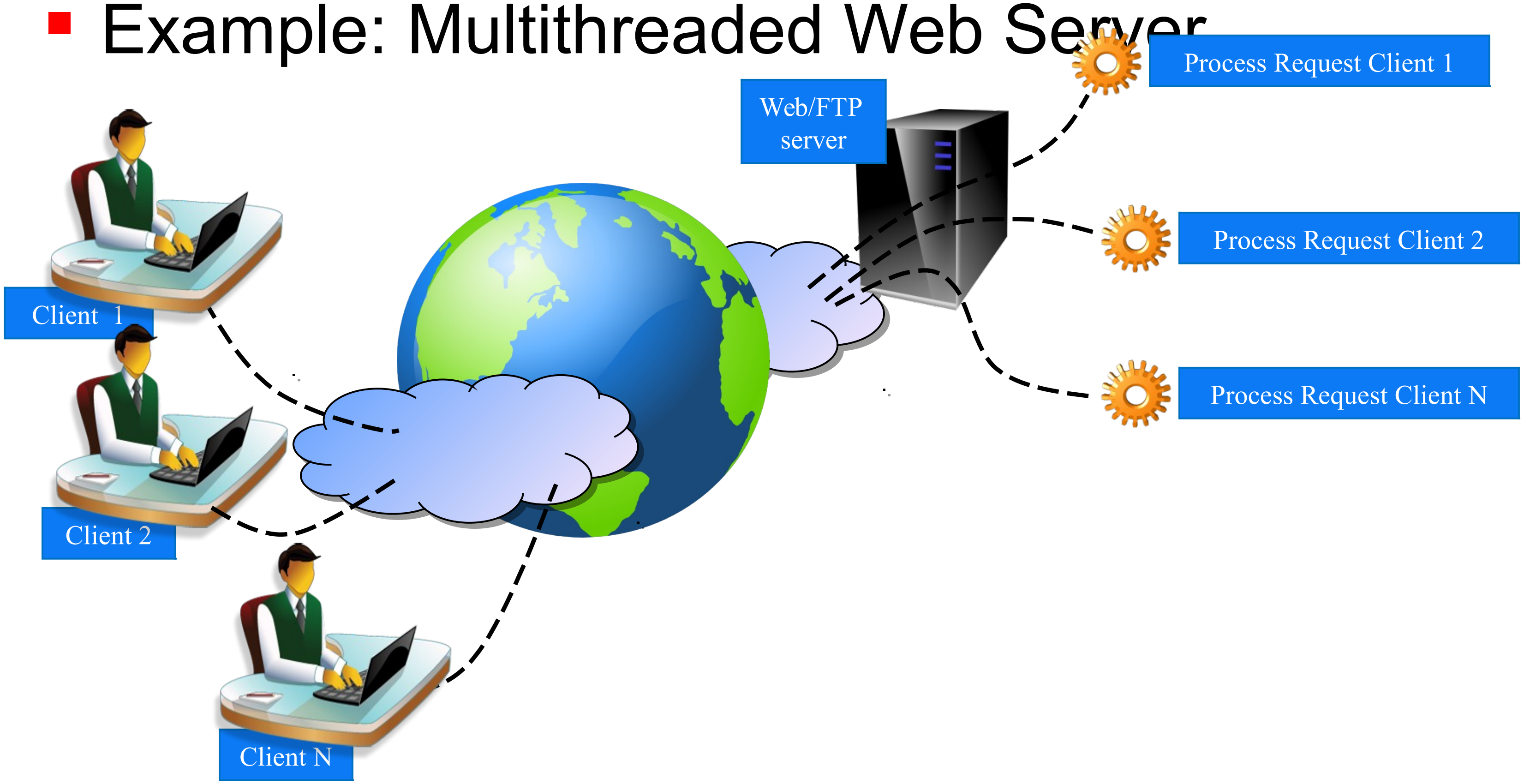
Single and Multithreaded Processes

threads are light-weight processes within a process



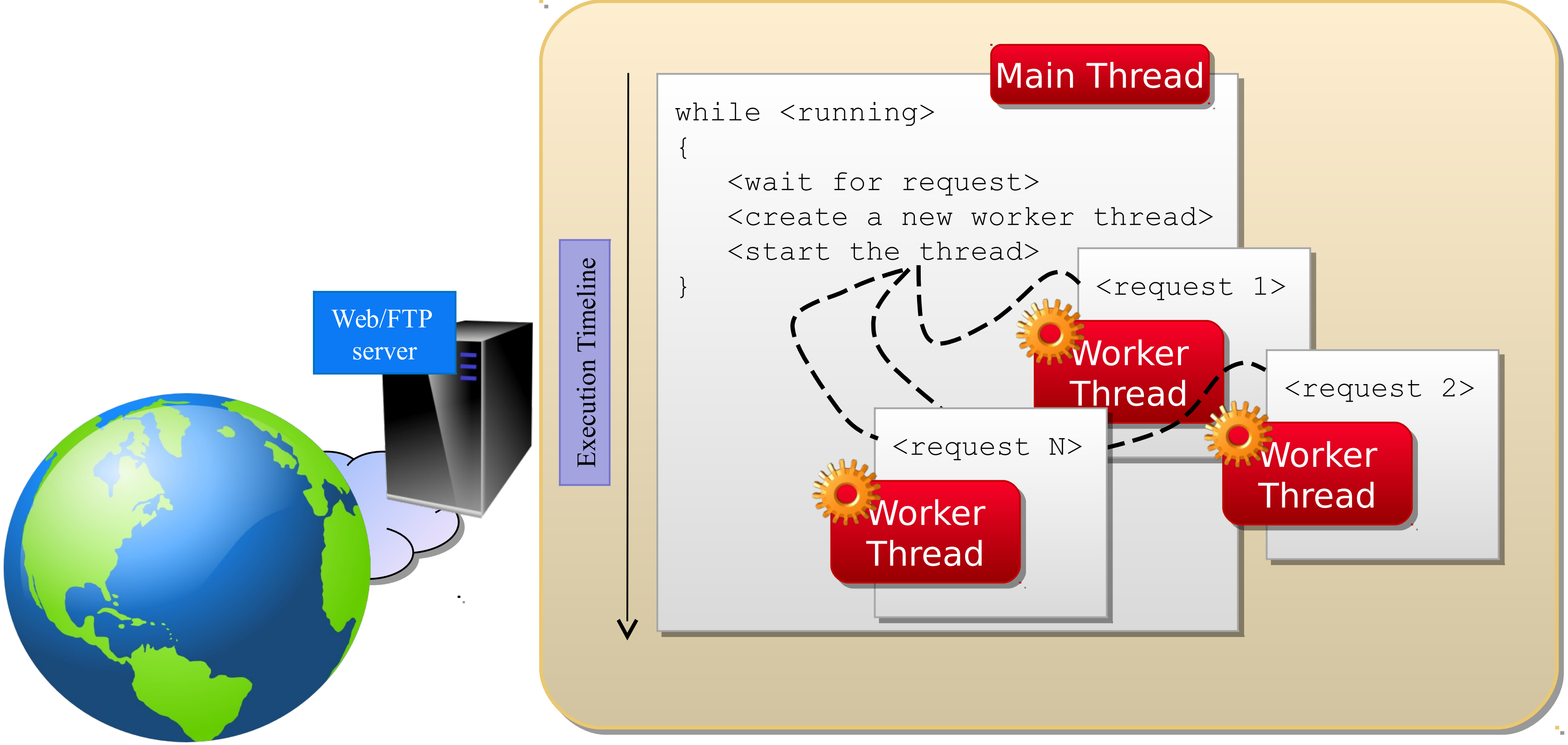
Multithreaded Server: For Serving Multiple Clients Concurrently

- Modern Applications
  - Example: Multithreaded Web Server



Defining Threads

- Example: Web/FTP Server



```
while <running>
{
   <wait for request>
   <create a new worker thread>
   <start the thread>
}
```

Defining Threads

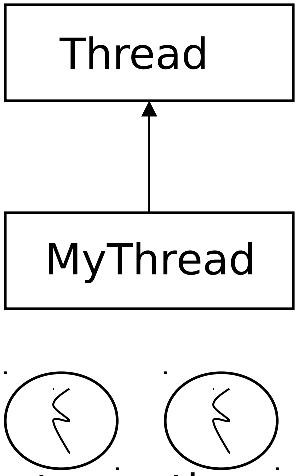
- Summing Up
  - A Thread is a piece of code that runs in concurrent with other threads.
  - Each thread is an ordered sequence of instructions.
  - Threads are used to express concurrency on both single and multiprocessors machines.
  - Programming a task having multiple threads of control – **Multithreading**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

Java Threads

- Java has a built-in support for Multithreading
  - Synchronization
  - Thread Scheduling
  - Inter-Thread Communication
- Java Garbage Collector is a low-priority thread.

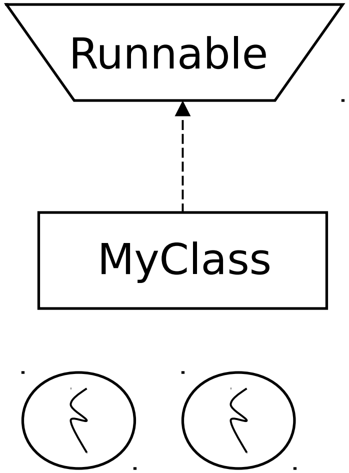
Threading Mechanisms...

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



(objects are threads)

[a]



(objects with run() body)

[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();
       }
}
```

2^{nd} 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();
        }
}
```

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

```
        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

```
          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 a 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");
         }
}
```

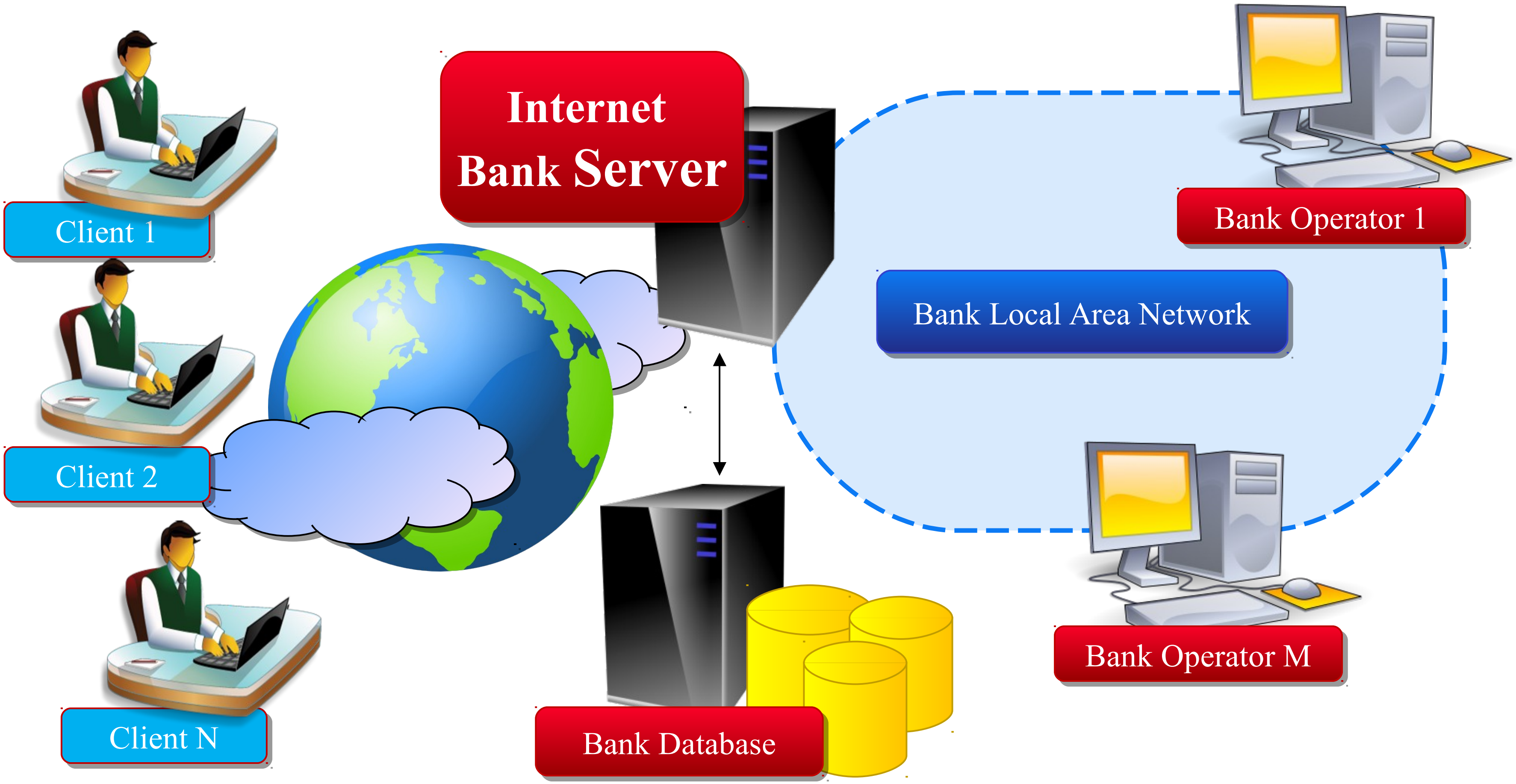
Thread Priority Example

```
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()
  - {
    - …
  - }

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.
      }
}
```

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.
- Threads need to be synchronized when accessing and manipulating shared resources.
- New OS designs provide flexibility in terms of separating mechanisms from policies.