UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

NGÔN NGỮ LẬP TRÌNH JAVA

Chương 5
LẬP TRÌNH ĐA LUỒNG

GVGD: ThS. Lê Thanh Trọng

NỘI DUNG

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

- Mục tiêu bài học
- Giới thiệu về Đa luồng trong Java
- Lớp thread
- Giao diện Runnable
- Các phương thức được hỗ trợ
- Sự ưu tiên trong đa luồng
- Daemon Threads
- Đồng bộ hóa
- Thread pool
- Tóm tắt bài học

Mục tiêu bài học

- Giới thiệu về các khái niệm về tiến trình, luồng và khái niệm lập trình đa luồng với Java
- Sử dụng lớp Thread hoặc giao diện Runnable do Java cung cấp để lập trình đa luồng
- Từ đó sinh viên nắm được giải pháp thực hiện đa luồng trong khi lập trình bằng Java kết hợp với các phương thức được hỗ trợ như join, sleep của lớp Thread
- Giới thiệu về các khái niệm liên quan đến lập trình đa luồng như kiểm soát sự ưu tiên khi lập công việc giữa các threads, Daemon Threads, việc đồng bộ hóa giữa các thread khi truy cập chung tài nguyên
- Cơ chế thread pool trong Java để quản lý và sử dụng các thread hiệu quả hơn

CÂU HỎI THẢO LUẬN

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

1. Process là gì? Thread là gì? So sánh process và thread. Lập trình multi thread là gì? Cho ví dụ?
2. Thread có các trạng thái nào? Vẽ sơ đồ vòng đời gồm các trạng thái của thread.
3. Các giao diện/lớp nào liên quan đến lập trình multi thread? Nêu các thuộc tính, phương thức của giao diện/ lớp đó. Các cách tạo các đối tượng thread trong java?
4. Độ ưu tiên của thread là gì? Ý nghĩa? Thread trong java có các độ ưu tiên nào?
5. Đồng bộ hóa là gì? Cho ví dụ? Cách hiện thực tính năng đồng bộ hóa trong java.

Giới Thiệu

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Hệ điều hành đa nhiệm cổ điển:

- Đơn vị cơ bản sử dụng CPU là tiến trình (process)
- Tiến trình là đoạn chương trình độc lập đã được nạp vào bộ nhớ
- Mỗi tiến trình thi hành một ứng dụng riêng.
- Mỗi tiến trình có một không gian địa chỉ và một không gian trạng thái riêng
- Các tiến trình liên lạc với nhau thông qua HĐH, tập tin, mạng

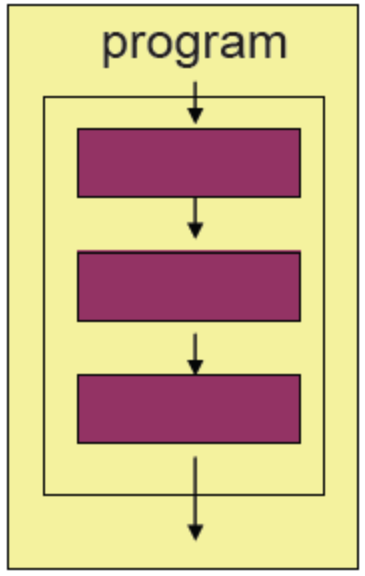
Giới Thiệu

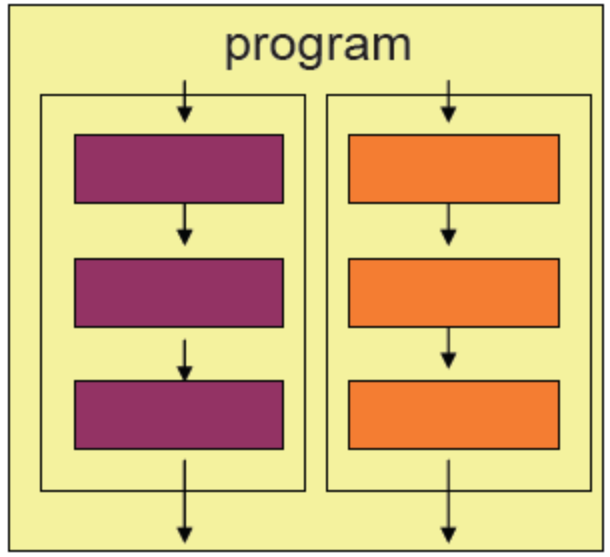
UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

- **Hệ điều hành đa nhiệm hiện đại, hỗ trợ luồng:**
  - Đơn vị cơ bản sử dụng CPU là luồng (thread).
  - Luồng một đoạn các câu lệnh được thi hành.
  - Mỗi tiến trình có một không gian địa chỉ và nhiều luồng điều khiển.
  - Mỗi luồng có bộ đếm chương trình, trạng thái các thanh ghi và ngăn xếp riêng.
  - Luồng của một tiến trình có thể chia sẻ nhau không gian địa chỉ : Biến toàn cục, tập tin, chương trình con, hiệu báo, . . .
  - Luồng chia sẻ thời gian sử dụng CPU => Luồng cũng có các trạng thái: Sẵn sàng (ready), Đang chạy (running), Nghẽn(Block) như quá trình.
  - Luồng cung cấp cơ chế tính toán song song trong các ứng dụng.

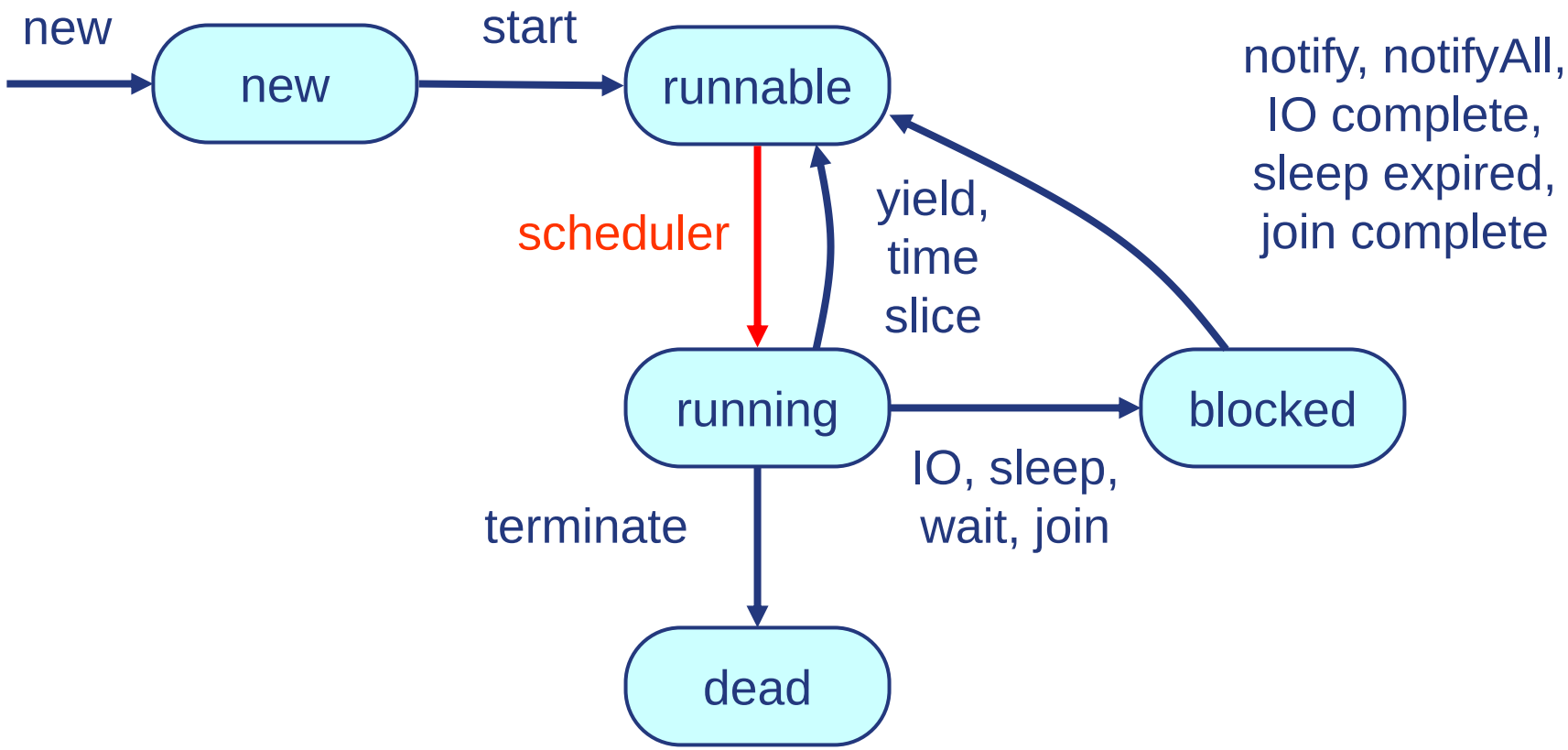
Giới Thiệu

- Luồng là mạch thi hành độc lập của một tác vụ trong chương trình
- Một chương trình có nhiều luồng thực hiện cùng lúc gọi là đa luồng





Vòng Đời Của Thread



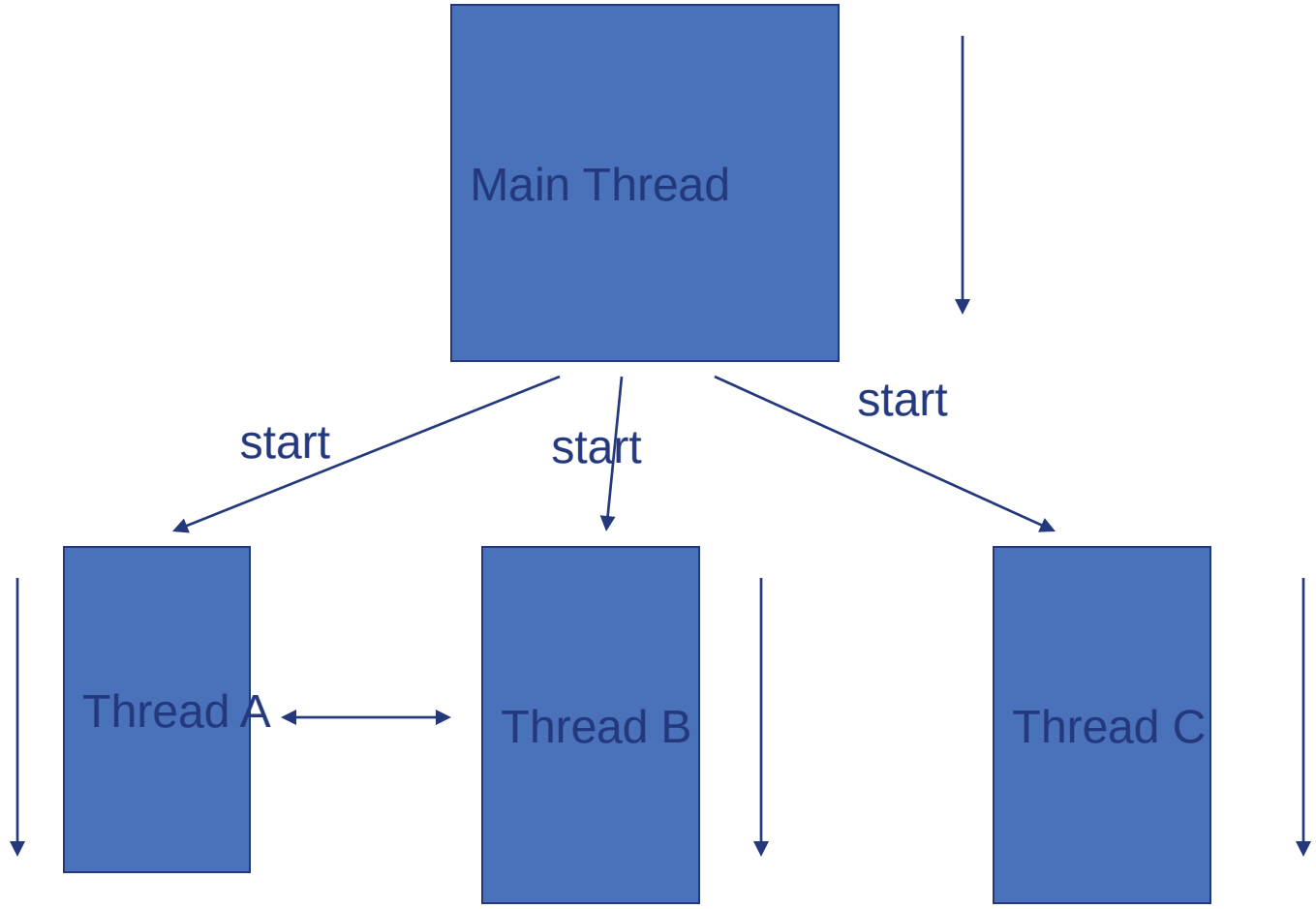
Chương trình đơn luồng

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

Đa luồng

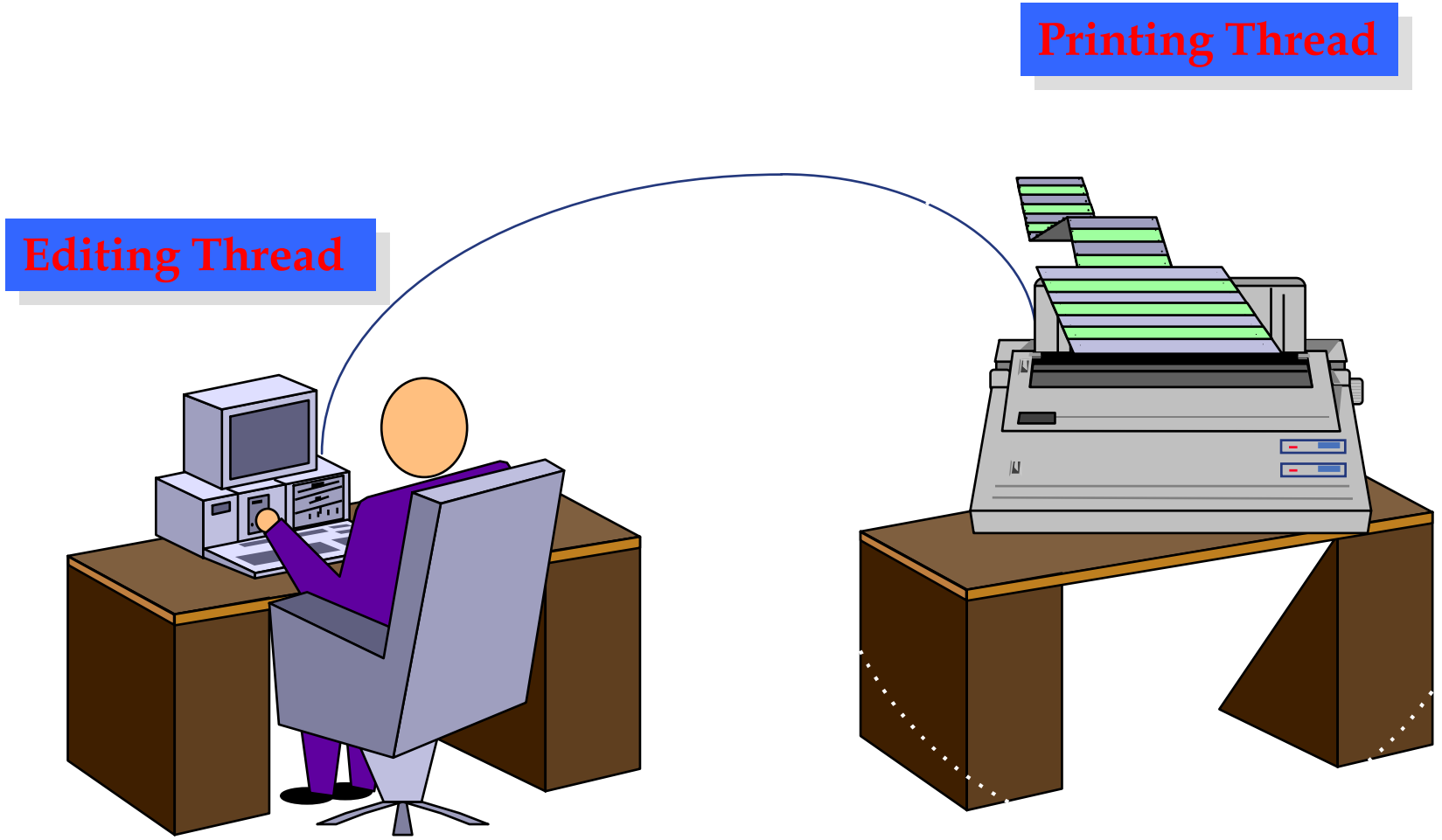
- Là khả năng làm việc với nhiều thread
- Đa luồng chuyên sử dụng cho việc thực thi nhiều công việc đồng thời
- Đa luồng giảm thời gian rỗi của hệ thống đến mức thấp nhất.

A Multithreaded Program

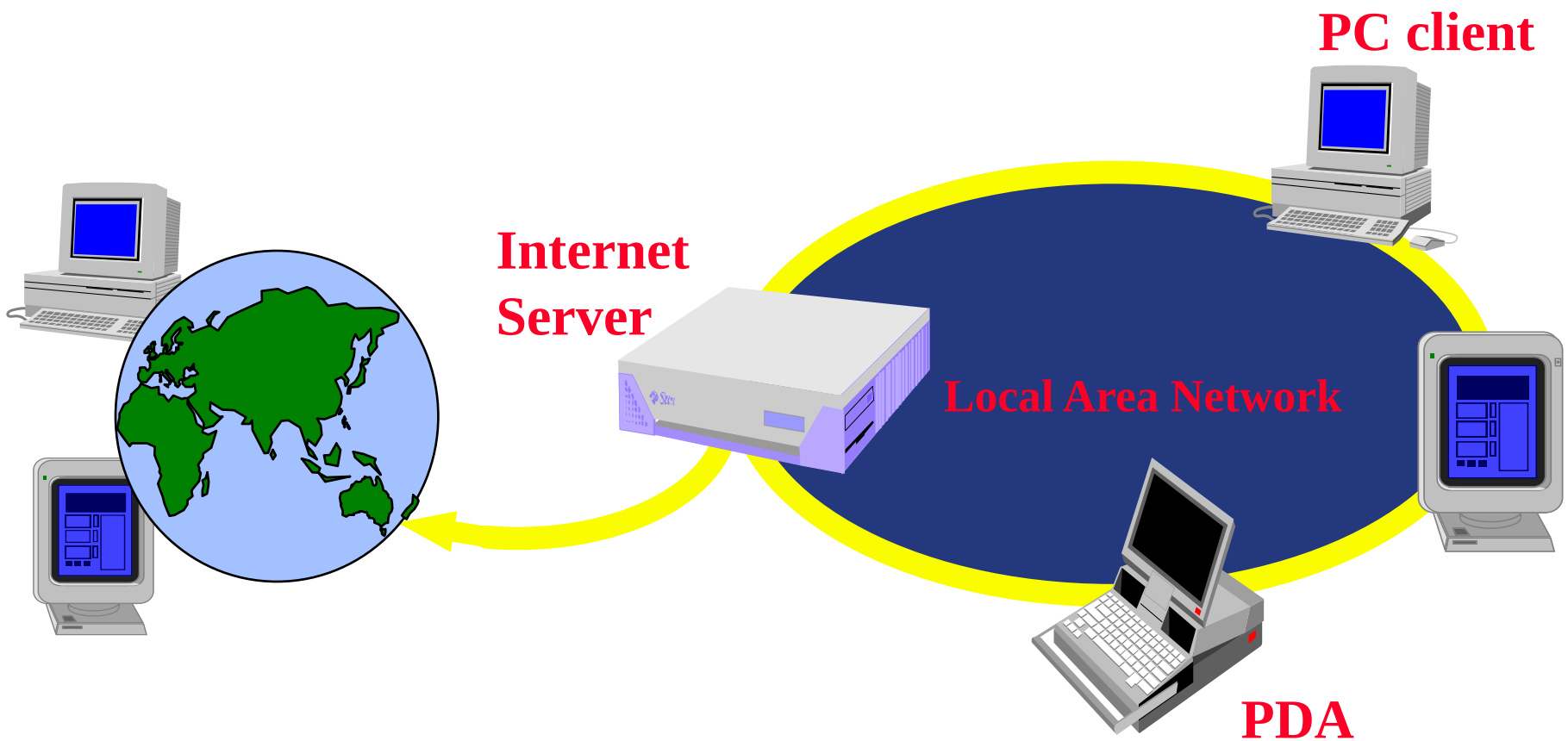


Các thread có thể chuyển đổi dữ liệu với nhau

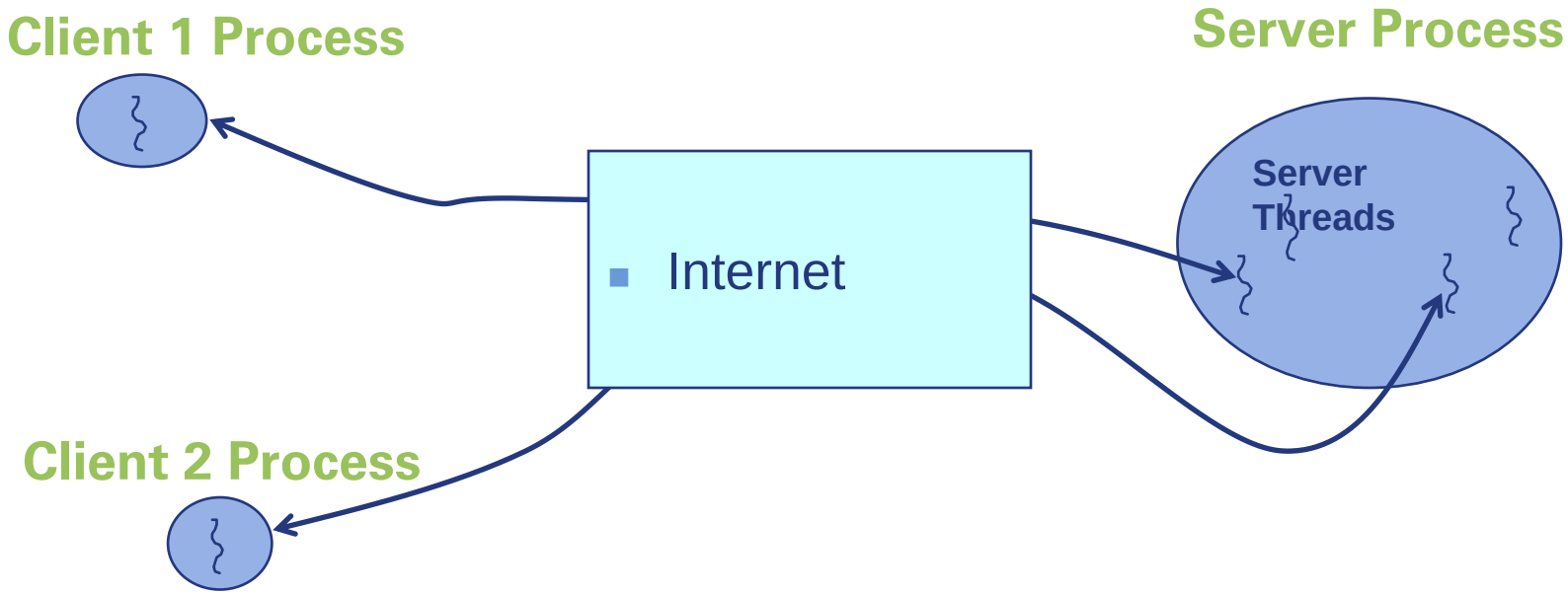
Ứng Dụng Multithreading



Web/Internet Applications



Multithreaded Server



Tạo và quản lý luồng

- Khi chương trình Java thực thi hàm main(): luồng main được thực thi. Luồng này được tạo ra một cách tự động, tại đây :
  - Các luồng con sẽ được tạo ra từ đó
  - Nó là luồng cuối cùng kết thúc việc thực hiện. Nếu luồng chính ngừng thực thi, chương trình bị chấm dứt
- Luồng có thể được tạo ra bằng 2 cách:
  - Dẫn xuất từ lớp Thread
  - Hiện thực giao diện Runnable.

Lập trình đa tuyến với Java

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

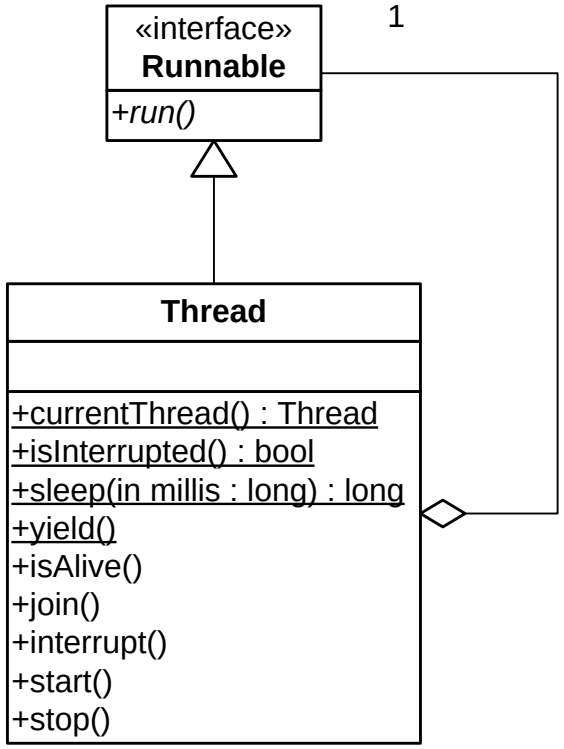
- Cách thực hiện
  - Sử dụng lớp java.lang.Thread

    public class Thread extends Object { … }
  - Sử dụng giao diện java.lang.Runnable

    public interface Runnable {

        public void run(); // work ⇒ thread

    }



Lớp java.lang.Thread

- Luồng trong java là một đối tượng của lớp **java.lang.Thread**
- Một chương trình cài đặt luồng bằng cách tạo ra các lớp con của lớp Thread.
- Lớp **Thread có 3 phương thức cơ bản:**
  - *public static synchronized void start() :*
    - Chuẩn bị mọi thứ cần thiết để thực hiện luồng.
  - *public void run():*
    - Chứa mã lệnh thực hiện công việc thực sự của luồng.
    - run() được gọi một cách tự động bởi start().
  - *public void stop() : kết thúc một luồng.*
  - Luồng kết thúc khi:
    - Hoặc tất cả các lệnh trong run() đã được thực thi.
    - Hoặc phương thức stop() của luồng được gọi.

Tạo thread sử dụng lớp Thread

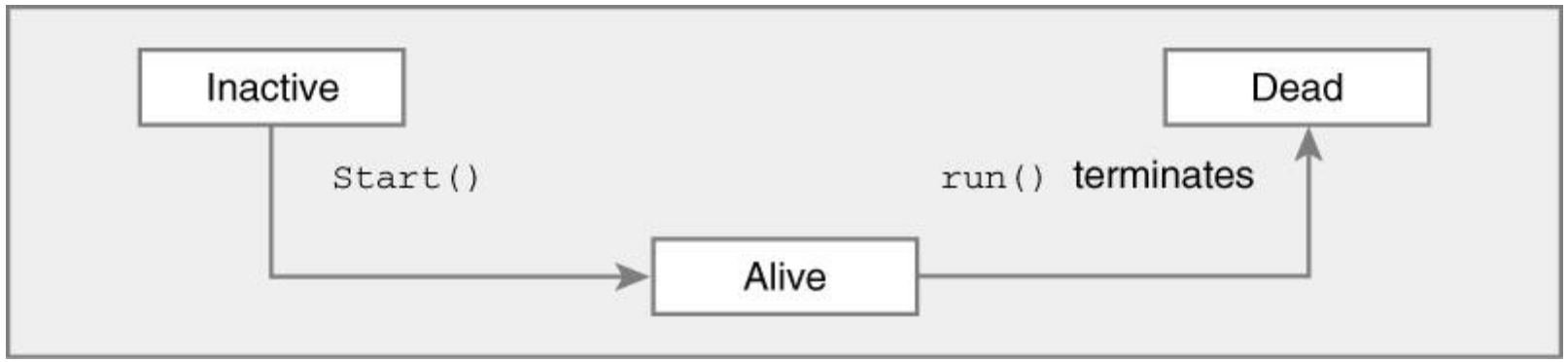
UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

- Cài đặt lớp kế thừa từ lớp Thread và override phương thức run()

```
public class MyThread extends Thread {
 public void run() {
      System.out.println("This code is running in a
thread");
} }
```

Khởi chạy đối tượng Thread

- Cần tạo ra đối tượng Thread và gọi phương thức start để thực hiện công việc.




```
Mythread thread = new Mythread();
thread.start();
System.out.println("This code is outside of the thread");
```

Tạo lớp con của Thread

```
class MultithreadingDemo extends Thread {
    public void run()
    {
        try {
            // Displaying the thread that is running
            System.out.println(
                "Thread " + Thread.currentThread().getId()
                + " is running");
        }
        catch (Exception e) {
            // Throwing an exception
            System.out.println("Exception is caught");
        }
    }
}
```

Tạo ra và chạy n threads

```
// Main Class
public class Multithread {
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i = 0; i < n; i++) {
            MultithreadingDemo object
                = new MultithreadingDemo();
            object.start();
        }
    }
}
```

Kết quả

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Output

```
Thread 15 is running
Thread 14 is running
Thread 16 is running
Thread 12 is running
Thread 11 is running
Thread 13 is running
Thread 18 is running
Thread 17 is running
```

Giao diện Runnable

- java.lang.Runnable là giao diện của Java, cho phép các thể hiện của các lớp cài đặt giao diện này có thể được chạy trong các Thread.
- Tạo 1 lớp hiện thực hóa giao diện Runnable và cài đặt phương thức run:

```
public class Mythread implements Runnable {
	public void run() {
		System.out.println("This code is running in a thread");
	}
}
```

Khởi chạy luồng

- Tạo đối tượng:

  MyThread myObject = new MyThread();

- Tạo thread từ đối tượng:

  Thread thr1 = new Thread(myObject);

- Thi hành thread:

  thr1.start();

```
Runnable obj = new Mythread();
Thread thread = new Thread(obj);
thread.start();
```

Thread class methods

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Sr.No.	Method & Description
1	**public void start()** Khởi chạy luồng
3	**public final void setName(String name)** Thay đổi tên của Thread object. Có thể dùng getName() method để lấy thông tin tên.
4	**public final void setPriority(int priority)** Cài đặt độ ưu tiên của Thread object. Giá trị từ 1 đến 10.
6	**public final void join(long millisec)** Thread chính sẽ gọi PT này đối với các thread thứ cấp. Thread chính khi đó sẽ khóa cho đến khi thread thứ cấp xong công việc (hoặc hết thời gian).
7	**public void interrupt()** Ngắt luồng đang thực thi.
8	**public final boolean isAlive()** Kiểm tra luồng còn đang sống (trang thái nằm giữa từ lúc được gọi start đến khi kết thúc)
9	**public static void sleep(long millisec)** Khóa luồng hiện tại với thời gian xác định trước.

So sánh 2 phương pháp

- Khi tạo 1 lớp bằng cách mở rộng lớp Thread, ta không thể kế thừa thêm các lớp khác.
- Nếu tạo lớp bằng cách hiện thực giao diện Runnable, ta có thể kế thừa thêm các lớp khác, thường dùng khi cài đặt giao diện UI.

Độ Ưu Tiên

- Trong Java, mỗi thread được gán 1 giá trị để chỉ mức độ ưu tiên của thread. Khi thread được tạo ra có độ ưu tiên mặc định (NORM_PRIORITY) sẽ được thi hành theo quy tắc FCFS (First Come First Serve).
- Sử dụng phương thức setPriority() để thay đổi độ ưu tiên của thread:
  - ThreadName.setPriority(intNumber)
    - MIN_PRIORITY = 1
    - NORM_PRIORITY=5
    - MAX_PRIORITY=10

Ví Dụ Thread Priority

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

```
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");
    }
}
```

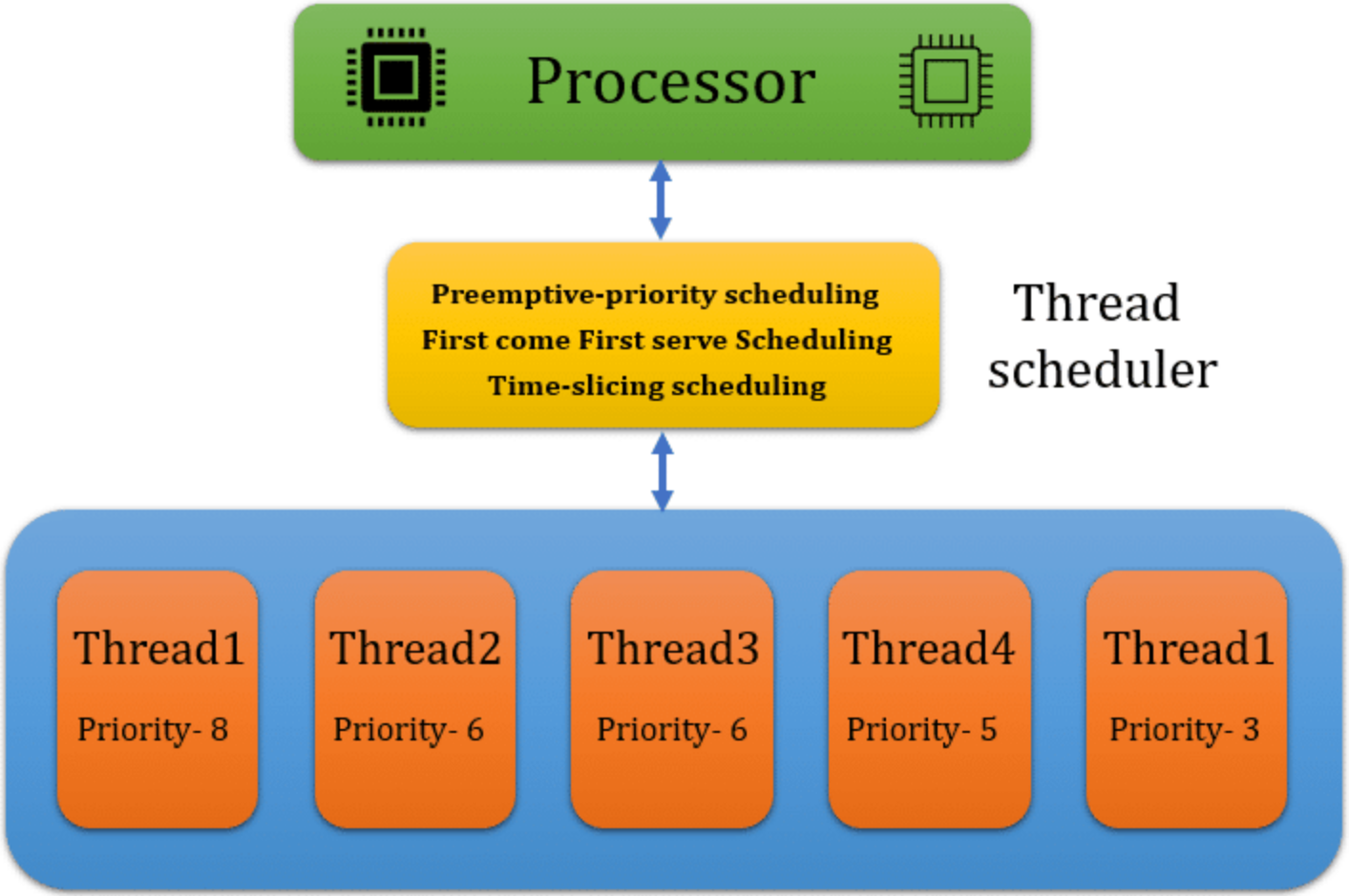
```
class C extends Thread
{
    public void run()
     {
        System.out.println("Thread C started");
        for(int k=1;k<=4;k++)
         {
            System.out.println("\t From ThreadC: k= "+k);
         }
          System.out.println("Exit from C");
    }
}
```

Ví Dụ Thread Priority

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

```
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");
         }
}
```

Java Thread Scheduler

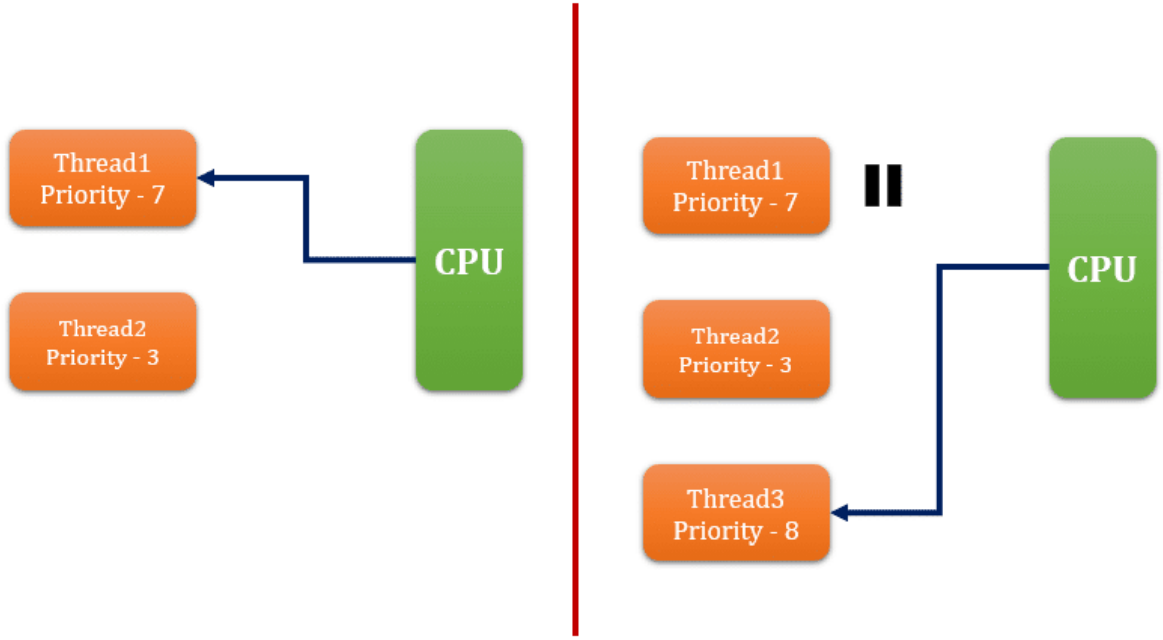


Java Thread Scheduler

- Thread scheduler là một phần của JVM quyết định việc chạy hay chờ của các thread. Nó chỉ chọn các thread để thực thi khi chúng trong trạng thái RUNNABLE.
- 2 nhân tố chính ảnh hưởng khi 1 thread được chọn để chạy:
  - Thread priority: Sự ưu tiên
  - Arrival time : Thời điểm xuất hiện

Preemptive-priority scheduling

- Việc lựa chọn các thread để chạy dựa trên giá trị ưu tiên của thread.
- Giả sử có nhiều threads trong trạng thái RUNNABLE, thread scheduler sẽ chọn thread có priority cao nhất để chạy.



First come First serve Scheduling

- The thread scheduler gán một khoảng thời gian CPU cho thread xuất hiện trước tiên.
- Cơ chế này sẽ thực hiện khi các thread xuất hiện có giá trị ưu tiên bằng nhau.

Daemon Threads

- Có 2 loại threads trong Java là user thread và daemon thread
- Daemon thread có độ ưu tiên thấp và luôn chạy ngầm dưới hệ thống. Thích hợp cho các tác vụ ngầm ví dụ dọn dẹp tài nguyên.
- Để khai báo 1 thread là daemon thread, ta gọi **thread.setDaemon(true)** trước khi gọi start.

Daemon Threads

```
public class MainClass {
    public static void main(String args[]) {
        MyThread myThread = new MyThread();
        myThread.setDaemon(true);
        myThread.start();

        // Sleep for five seconds
        try {
            Thread.sleep(5000);
        } catch (InterruptedException ignored) {
        }
        System.out.println("Main thread finished");
    }
}
```

```
class MyThread extends Thread {
    public void run() {
        while (true) {
        System.out.println("Daemon thread is
running");
        try {
                Thread.sleep(500);
            }
        catch (InterruptedException ignored) {
                }
            }
    }
}
```

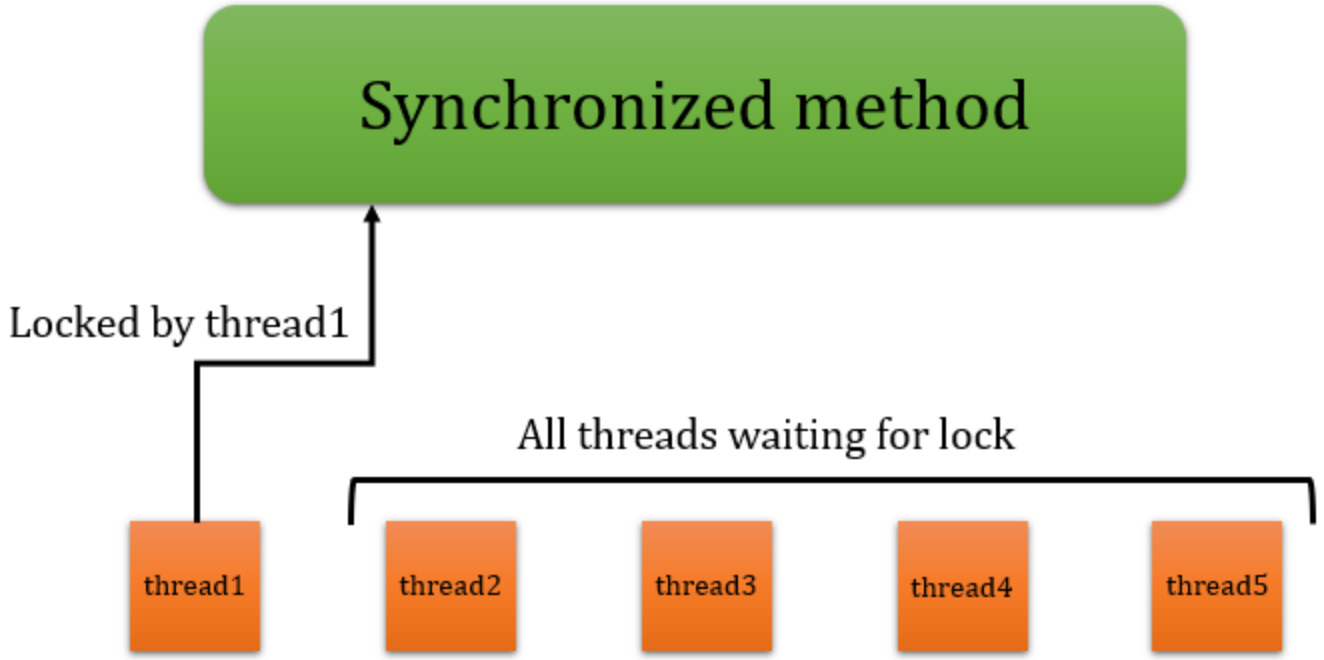
Truy Cập Tài Nguyên Dùng Chung

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

- Các ứng dụng truy cập vào tài nguyên dùng chung cần có cơ chế phối hợp để tránh đụng độ.
  - Máy in (2 công việc in không thể thực hiện cùng lúc)
  - Không thể thực hiện 2 thao tác đồng thời trên một tài khỏan
  - Việc gì sẽ xảy ra nếu vừa thực hiện đồng thời
    - Deposit()
    - Withdraw()

Đồng bộ hóa

Đồng bộ hóa là cơ chế chỉ cho phép 1 tiến trình truy cập vào tài nguyên dùng chung ở mỗi thời điểm.



Synchronized method

- Khai báo phương thức **synchronized** theo cú pháp:
  - **public synchronized int myMethod(int x)**
  - Chỉ duy nhất 1 thread được thực hiện phương thức **synchronized tại 1 thời điểm**
  - Đăng kí truy cập tài nguyên dùng chung :

Method	Mô tả
wait	Được gọi trong khi thi hành phương thức `synchronized`. Thread chuyển sang trạng thái chờ.
`notify`	Thông báo cho thread đang ở trạng thái wait chuyển sang ready
`notifyAll`	Thông báo tất cả thread ở trạng thái wait chuyển sang ready

Ví dụ đồng bộ hóa

```
class demo {
volatile boolean part1done = false;
synchronized void part1() {
        System.out.println("Welcome to UIT");
        part1done = true;
        notify(); // notify the waiting thread
    }
synchronized void part2() {
        // loop to prevent spurious wake-up
        while (!part1done) {
            try {
                System.out.println("Thread t2 waiting");
                wait(); // wait till notify is called
                System.out.println(
                    "Thread t2 running again");
            }
            catch (Exception e) {
                System.out.println(e.getClass());
            }}
        System.out.println("Do visit SEUIT.");
}}
```

```
public class Main {

    public static void main(String[] args){
            demo obj = new demo();

        Thread t1 = new Thread(new Runnable() {
            public void run() { obj.part1(); }
        });

        Thread t2 = new Thread(new Runnable() {
            public void run() { obj.part2(); }
        });

        // Start t2 and then t1
        t2.start();
        t1.start();
    }}
```

Synchronized Block

- Synchronized Block có thể được dùng để đồng bộ hóa 1 phần nhỏ tài nguyên bên trong 1 phương thức nào đó.

Cú pháp:

```
synchronized (object reference expression) {
  //code block
}
```

Khai báo Synchronized Block

```
class Count {
    public void countIncrement() {
        synchronized(this) {
            for (int i = 0; i < 3; i++) {
                System.out.println(i);
                try {
                    Thread.sleep(400);
                }
                catch (Exception e) {
                    System.out.println(e);
                }
            }
        }
    }
}
```

Sử dụng Synchronized Block

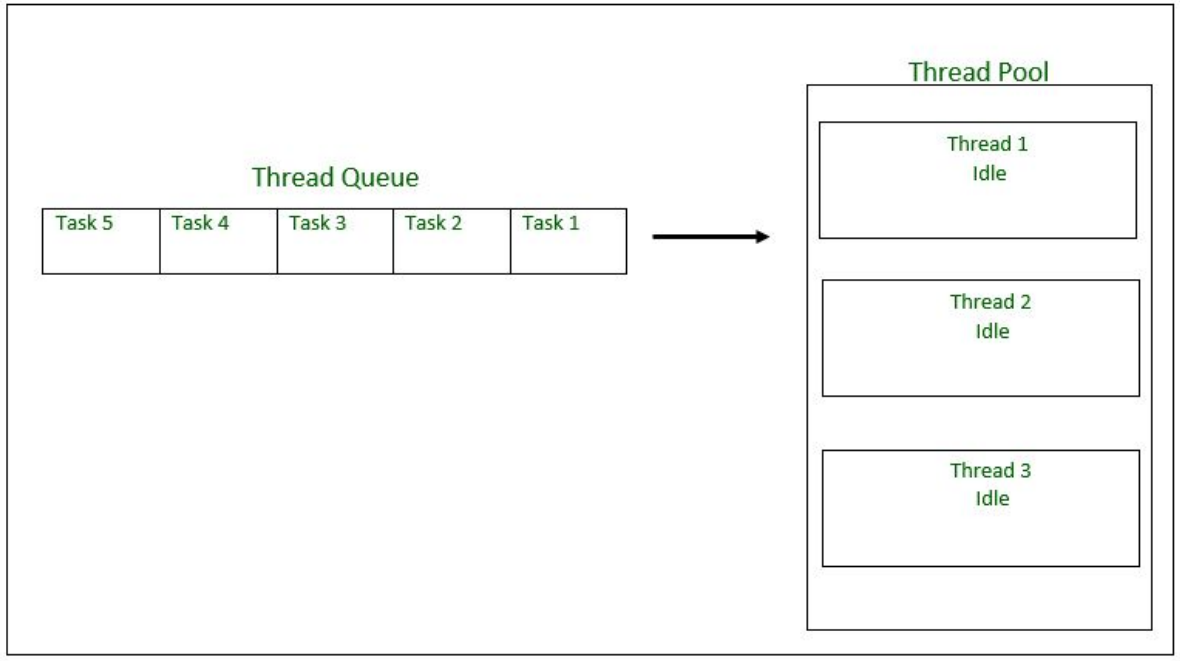
UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

```
class MyThread extends Thread {
      Count count;
      MyThread (Count _count) {
        this.count = _count;
    }

    @Override
    public void run() {
        count.countIncrement();
    }
}
```

Thread Pool

- Một thread pool sử dụng các threads được tạo sẵn để thực thi một loạt các công việc, nhằm giải quyết vấn đề về chi phí về vòng đời threads và truy cập tài nguyên.



Các bước thực hiện

1. Tạo các tác vụ cần thực hiện(Runnable Object)
2. Tạo Executor Pool sử dụng lớp Executors
3. Chuyển công việc vào Executor Pool
4. Tắt Executor Pool

Bước 1: Hiện thực giao diện Runnable

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

```
class Task implements Runnable
{
    private String name;

    public Task(String s)
    {
        name = s;
    }
    public void run()
    {//Do something here }
}
```

Step 2. Tạo đối tượng Executor

- import java.util.concurrent.ExecutorService;
- import java.util.concurrent.Executors;

```
// creates five tasks
Runnable r1 = new Task("task 1");
Runnable r2 = new Task("task 2");
Runnable r3 = new Task("task 3");
Runnable r4 = new Task("task 4");
Runnable r5 = new Task("task 5");

// creates a thread pool with MAX_T no. of
// threads as the fixed pool size(Step 2)
ExecutorService pool = Executors.newFixedThreadPool(MAX_T);
```

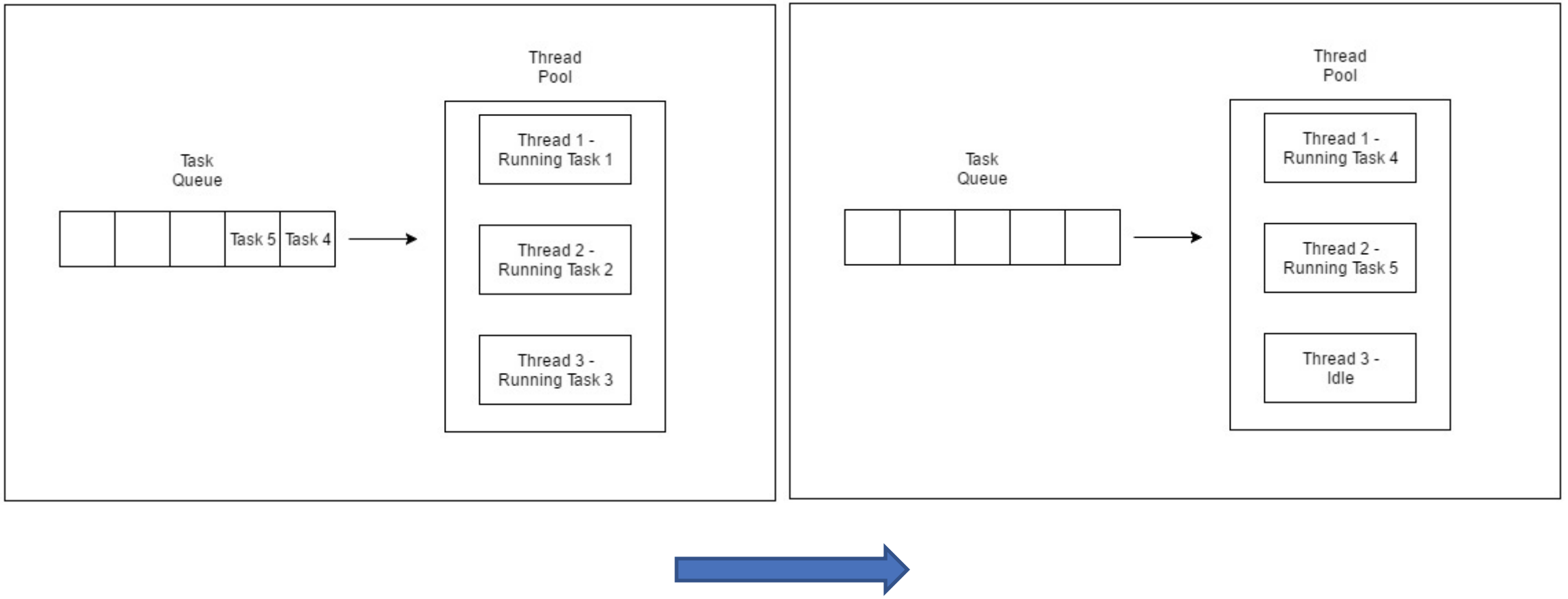
Bước 3 + 4.

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

```
        // passes the Task objects to the pool to
execute (Step 3)
        pool.execute(r1);
        pool.execute(r2);
        pool.execute(r3);
        pool.execute(r4);
        pool.execute(r5);

        // pool shutdown ( Step 4)
        pool.shutdown();
```

Thread pool


Thread
Pool
Thread 1 -
Running Task 1
Thread 2 -
Running Task 2
Thread 3 -
Running Task 3
Task
Queue
Task 5
Task 4
Thread
Pool
Thread 1 -
Running Task 4
Thread 2 -
Running Task 5
Thread 3 -
Idle
Task
Queue

Tóm tắt bài học

- Cơ chế đa luồng trong Java cho phép 1 chương trình thực hiện 1 số công việc đồng thời.
- Để thực hiện đa luồng, ta có thể tạo lớp thực thi giao diện Runnable hoặc mở rộng lớp Thread do java cung cấp.
- Cần lưu ý ưu nhược điểm chính của 2 phương pháp để lựa chọn phù hợp.
- Daemon thread là các luồng chạy ngầm bên dưới hệ thống.
- Ta có thể thay đổi giá trị ưu tiên của một luồng để can thiệp vào thứ tự ưu tiên khởi chạy của chúng trong hệ thống.

Tóm tắt bài học

- Các tài nguyên dùng chung có thể sử dụng synchronized method hoặc block để đồng bộ hóa giữa các luồng.
- Sử dụng thread pool để thực hiện đa luồng hiệu quả hơn đặc biệt khi số lượng công việc cần thực hiện song song là rất lớn