

## Processes

- One of the most important concepts in all modern operating systems
- A process is a **container** (or execution environment) for a program (software) in execution.
- Any software is executed with one or more processes.
  - Java VM (JVM), MS Word, Excel, PPT, Firefox, iTunes, Google Chrome, etc.

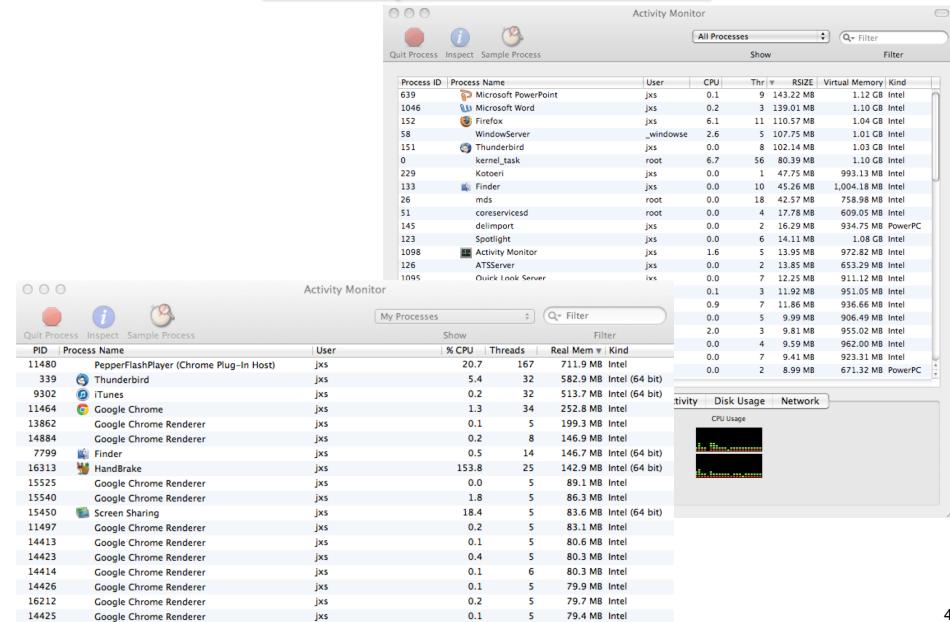
1

## Concurrency with Threads

- All modern OSes support **multi-tasking**.
  - Doing **multiple things with multiple processes on a single CPU** (more precisely, a single CPU core)
    - e.g., writing a document with MS Word while playing a music with iTunes and downloading files with a Web browser
- At any moment, a single CPU core can execute a single program (process).
- **Pseudo concurrency/parallelism**: An **illusion** for human eyes as if multiple processes are executed at the same time.
  - OS periodically assigns one process to another to the CPU core.
    - every several tens of milliseconds to several hundreds of milliseconds
    - A CPU core is multiplexed among processes; processes are not executed in a completely parallel manner.

3

## Example Processes

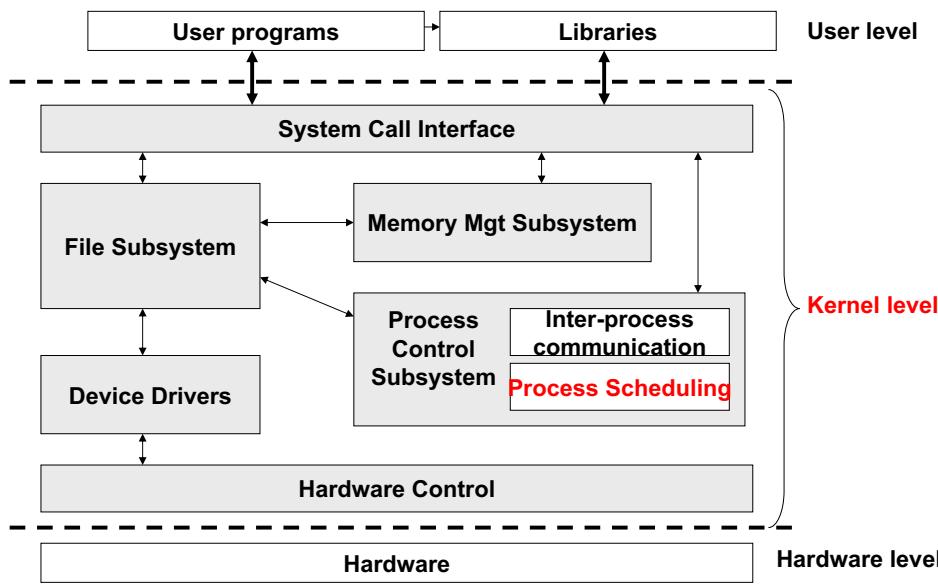


The screenshot shows the Activity Monitor application on a Mac OS X desktop. The main window displays a table of running processes, each with a small icon, process ID (PID), name, user, CPU usage, threads, real memory, virtual memory, and kind. Below this table are tabs for 'CPU Usage' and 'Network' showing system-wide resource consumption.

Activity Monitor						
All Processes						
Process ID	Process Name	User	CPU	Thr	RSIZE	Virtual Memory
639	Microsoft PowerPoint	jxs	0.1	9	143.22 MB	1.12 GB Intel
1046	Microsoft Word	jxs	0.2	3	139.01 MB	1.10 GB Intel
152	Firefox	jxs	6.1	11	110.57 MB	1.04 GB Intel
58	WindowServer	_windowse	2.6	5	107.75 MB	1.01 GB Intel
151	Thunderbird	jxs	0.0	8	102.14 MB	1.03 GB Intel
0	kernel_task	root	6.7	56	80.39 MB	1.10 GB Intel
229	Koteroi	jxs	0.0	1	47.75 MB	993.13 MB Intel
133	Finder	jxs	0.0	10	45.26 MB	1,041.18 MB Intel
26	root	root	0.0	18	45.26 MB	758.00 MB Intel
31	coreservicesd	root	0.0	4	17.79 MB	609.05 MB Intel
145	delimport	jxs	0.0	2	16.29 MB	934.75 MB PowerPC
123	Spotlight	jxs	0.0	6	14.11 MB	1.08 GB Intel
1098	Activity Monitor	jxs	1.6	5	13.95 MB	372.82 MB Intel
126	ATSServer	jxs	0.0	2	13.85 MB	653.29 MB Intel
1095	Quark Look Server	iws	0.0	7	12.25 MB	911.12 MB Intel
			0.1	3	11.92 MB	951.05 MB Intel
			0.9	7	11.86 MB	936.66 MB Intel
			0.0	5	9.99 MB	906.49 MB Intel
			2.0	3	9.81 MB	955.02 MB Intel
			0.0	4	9.59 MB	962.00 MB Intel
			0.0	7	9.41 MB	923.31 MB Intel
			0.0	2	8.99 MB	671.32 MB PowerPC

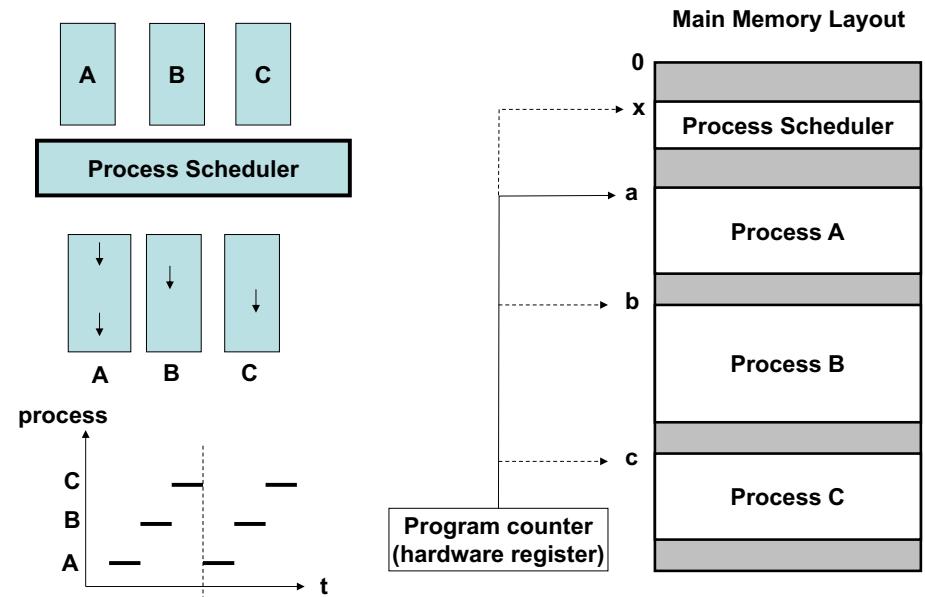
2

# An Architectural View of an OS



5

## Process Scheduling



6

## Why Threads?

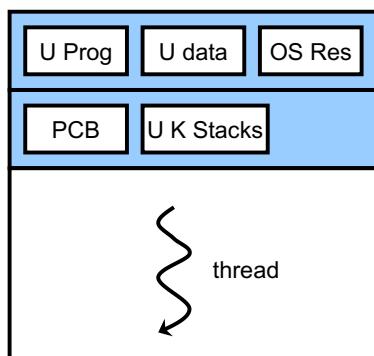
- **Good old days:** Process-based, **coarse-grained concurrency** was enough
  - e.g., Editing a document while downloading files
- **Now:** **Fine-grained concurrency** is required.
  - Each program is required to execute **in-program concurrent tasks**.
    - Doing different things at the same time.
    - Doing the same or a similar thing for multiple clients at the same time.
  - Each program is required to do different things at the same time.
    - Word
      - Displaying text and images
      - Responding to keystrokes and mouse inputs from the user
      - Downloading fancy document templates from MS web site
      - Performing spelling and grammar checking in the background
    - Web browser
      - Displaying text and images
      - Responding to keystrokes/mouse inputs
      - Checking and downloading software updates
      - Multiple tabs
    - iTunes
      - Playing music
      - Downloading music and its metadata (album's cover img, song titles, lyrics...)

- # of threads
  - Kernel tasks: 121
  - Dropbox: 103
  - Firefox: 74
  - Thunderbird: 57
  - iTunes: 30
  - MS PPT: 21
- Each program is required to do the same or a similar thing for multiple clients at the same time.
- Web server
  - Accepts and parses an HTTP request
  - Finds a target file
  - Makes an HTTP message (header, payload, etc.)
  - Returns a target file with the HTTP message
- Assign **threads** to in-program concurrent tasks

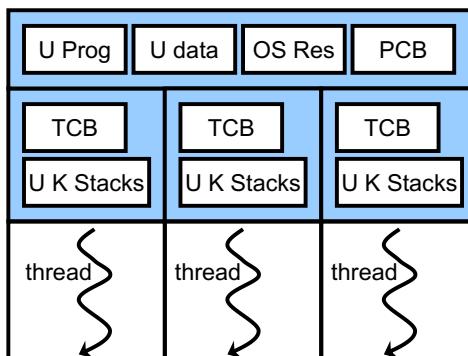
9

10

- Process-creation is heavyweight.
  - Time-consuming and resource intensive.
    - Creating a process is 30 times slower than creating a thread.
    - Process switching is 5 times slower than thread switching.



Single-threaded (traditional) process



Multi-threaded process

- **Fine-grained pseudo parallelism:** An **illusion** for human eyes as if multiple threads are executed on a process at the same time.
  - OS periodically assigns one thread to another to the CPU core.
    - A CPU core is multiplexed among processes and threads; processes/threads are not executed in a completely parallel manner on a single CPU core.

11

12

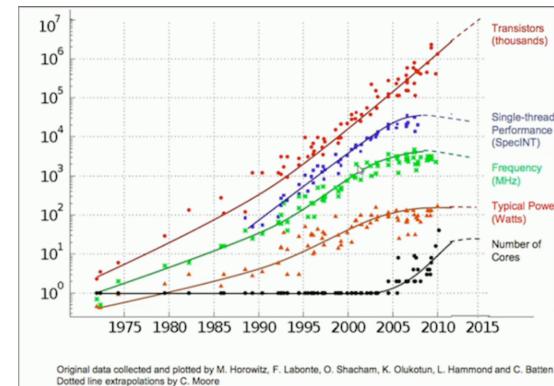
## Summary: Why Threads?

- Responsiveness/availability
  - Threads allow a program to continue running even if a part of it is blocked for I/O or is performing a long operation.
- Resource sharing
  - Threads share the memory and other resources of a process that they belong to.
- Efficiency
  - Threads are more lightweight than processes.
    - Process creation is expensive.
    - Switching processes is heavyweight.

13

## Other Viewpoints to Threads

- CPU speed does not increase any more as it did in the past...
  - Physical material barrier to increase clock speed and # of transistors in a CPU
  - Heat (and cooling) and power consumption problems



14

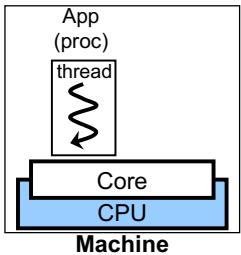
- Each CPU increases its density of cores, not its speed.
  - Intel Core i7: up to 4.0 GHz and up to 8 cores
  - Intel Xeon: up to 4.0 GHz and up to 8 cores
  - Multiple relatively-slower CPU cores for mobile devices to reduce power consumption
    - Amazon Kindle Fire (\$50): Quad-core 1.3 GHz
    - Apple iPhone 8/10: 6 cores
- Significant energy consumption and CO<sub>2</sub> emission by data centers
  - Amazon (450K+ servers), Facebook, Google (500K+ servers), Microsoft, Salesforce.com and Yahoo!
  - DCs consumed approx. 1.5% of global electricity usage ('10).
  - Google paid \$2M/mo for electricity bills ('07).
  - The Info. and Comm. Technology (ICT) industry produced 2% of global CO<sub>2</sub> emission ('07).
    - On par with the aviation industry.
  - IDCs were responsible for 23% of the ICT industry's emission

15

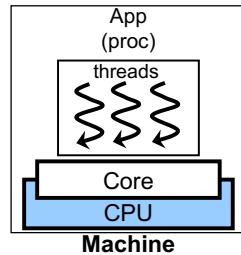
- GPUs are designed with a massive # of cores.
  - NVIDIA Tesla K80: 4992 cores
- You are expected to increase your app's performance by increasing its concurrency.
- Ultimate goals of multi-threaded applications:  
**Responsiveness and performance improvement**

16

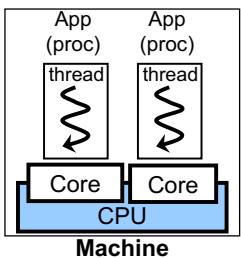
# Terminology: Single- or Multi-threaded?



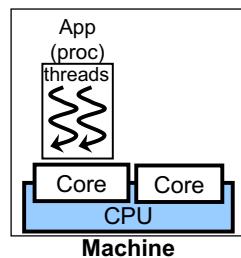
Single-threaded (sequential) app



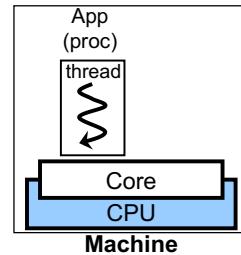
Multi-threaded app (concurrent)



Single-threaded (sequential) app

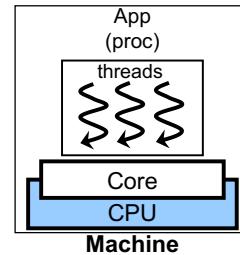


Multi-threaded app (concurrent)



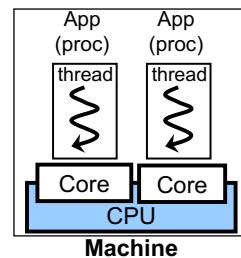
Single-threaded (sequential) app

Executed sequentially



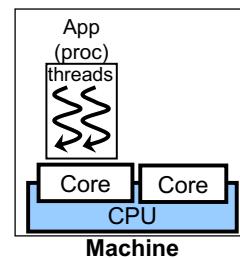
Multi-threaded app (concurrent)

Executed concurrently



Single-threaded (sequential) app

Executed in parallel. Each proc is executed sequentially.



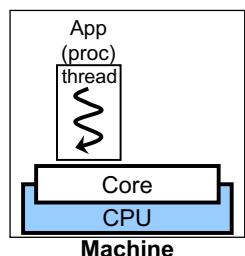
Multi-threaded app (concurrent)

Executed concurrently. Not parallel.

17

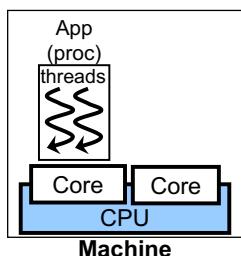
18

# Terminology: Concurrent or Parallel?



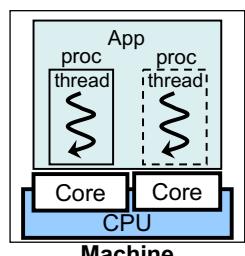
[1] Single-threaded (sequential) app

Not parallel



[2] Multi-threaded app (concurrent)

Not parallel



[3] Multi-threaded app (concurrent)

The terms "concurrent" [2] and "parallel" [3] are not always equivalent.

Parallel

If you think [3] is a special case of [2], you may prefer the term "concurrent."

If you think [2] is a special case of [3], you may prefer the term "parallel."

## In Java...

- The terms "concurrent" and "parallel" are somewhat mixed up in Java.
  - Java 5 (2004) introduced `java.util.concurrent`, which was further enhanced by Java 6 (2006).
    - "Concurrent" collections such as `concurrentHashMap`
    - The Executor framework
      - An extension/abstraction over low-level threads
  - Java 7 (2011)
    - The Fork/Join framework, an extension to the Executor framework
      - The term "parallel" appeared in its documentation, although it was placed in `java.util.concurrent`.
    - "Parallel" garbage collector
  - Java 8 (2014)
    - "Parallel" collection streams, which allows for "parallel" (threaded) operations on collections with lambda expressions
      - e.g. MapReduce-inspired API extension for collections

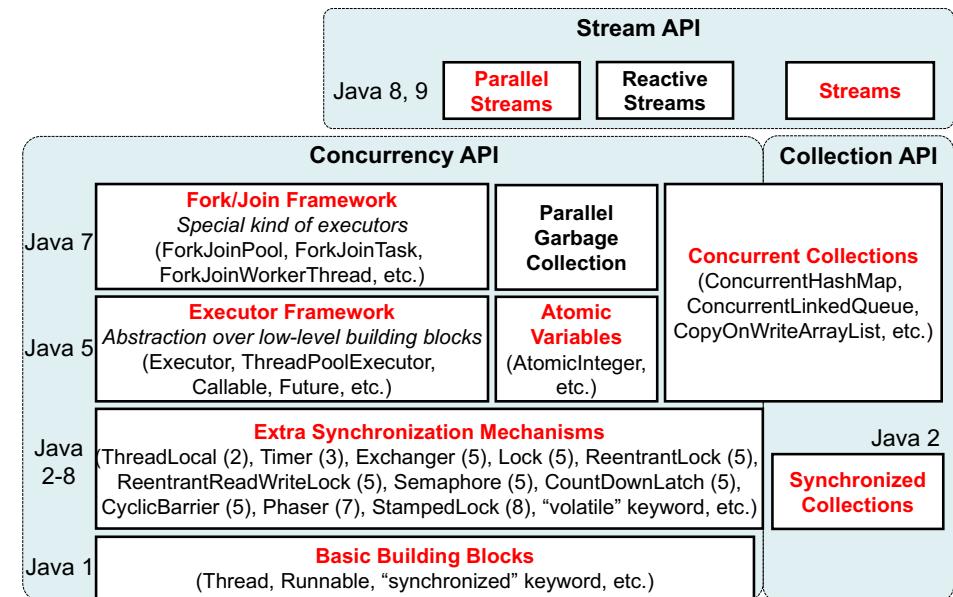
19

20

# Concurrency API in Java

- An example of using a parallel stream.

```
- Integer price = cars.stream()
    .parallel()
    .map((Car car) -> car.getPrice())
    .reduce(0, (result, carPrice) ->{
        if(result==0) return carPrice;
        else if(carPrice < result) return carPrice;
        else return result; } );
```



21

# Java Threads

- Every Java program has at least one *thread of control*.
  - `main()` runs with a thread on a JVM.
    - The “main” thread is *implicitly* created when a JVM starts.
- If you need extra threads in addition to the main thread, you need to *explicitly* create them.
- 4 things to do:
  - Define a class implementing the `java.lang.Runnable` interface
    - `public abstract void run();`
  - Write a threaded/concurrent task in `run()` in the class
  - Instantiate `java.lang.Thread` and associate a `Runnable` object with the thread
  - Start (call `start()` on) the instantiated thread.
    - `run()` is automatically called on the thread.

## Java Threads

23

24

## An Example Code: Creating a Thread

- GreetingRunnable.java

```
- class GreetingRunnable implements Runnable{  
    private String greeting;  
  
    public GreetingRunnable(String aGreeting){  
        greeting = aGreeting; }  
  
    public void run(){  
        for( int i=0; i<10; i++ ){  
            Date now = new Date();  
            System.out.println(now + " " + greeting);  
        } } }
```

- HelloWorldTest.java

```
- main(...){  
    GreetingRunnable runnable1 = new GreetingRunnable("Hello World");  
    Thread thread1 = new Thread(runnable1);  
    thread1.start(); }
```

## Thread.start()

- Creating a Thread object does not mean creating a new thread.

- It is start() that actually creates a thread.

- start()

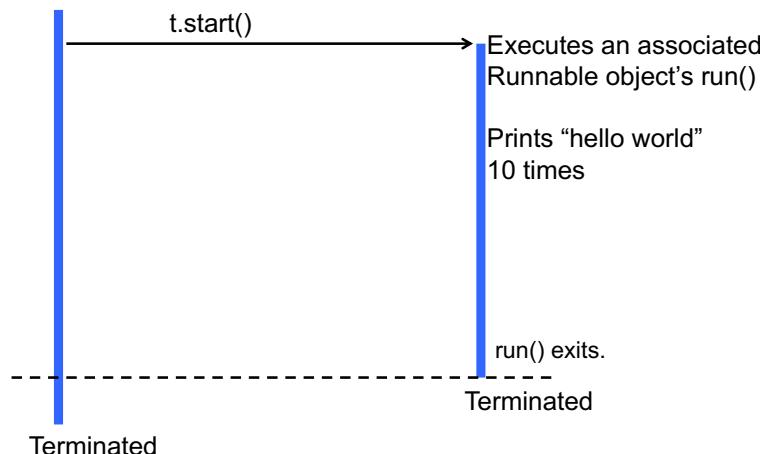
- Allocates memory and initializes a new thread on a JVM.
  - Calls run() of a specified Runnable object.
  - Do not call run() directly, but let start() call run() on behalf of yourself.

25

26

## Program Execution

### Main thread



### • Output:

- Mon Mar 26 15:14:43 EDT 2007 Hello World
- Mon Mar 26 15:14:44 EDT 2007 Hello World
- Mon Mar 26 15:14:45 EDT 2007 Hello World
- Mon Mar 26 15:14:46 EDT 2007 Hello World
- Mon Mar 26 15:14:47 EDT 2007 Hello World
- Mon Mar 26 15:14:48 EDT 2007 Hello World
- Mon Mar 26 15:14:49 EDT 2007 Hello World
- Mon Mar 26 15:14:50 EDT 2007 Hello World
- Mon Mar 26 15:14:51 EDT 2007 Hello World
- Mon Mar 26 15:14:52 EDT 2007 Hello World

27

28

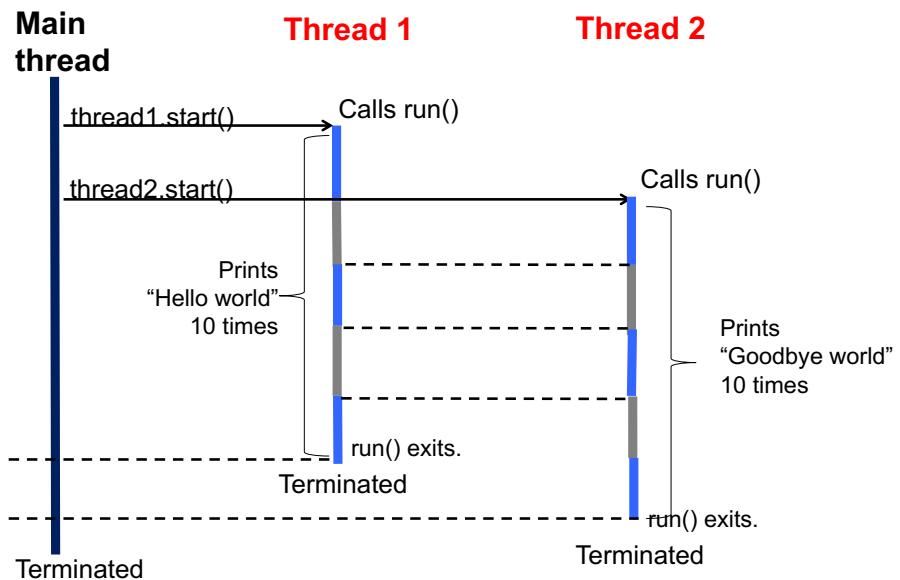
## An Example Code: Creating Threads

- HelloWorldTest2.java and GreetingRunnable.java

```
main(...){  
    GreetingRunnable runnable1 = new GreetingRunnable("Hello World");  
    GreetingRunnable runnable2 = new GreetingRunnable("Goodbye World");  
  
    Thread thread1 = new Thread(runnable1);  
    Thread thread2 = new Thread(runnable2);  
  
    thread1.start();  
    thread2.start();  
}
```

29

## Expected Program Execution



30

## A Possible Output

- Output:
  - Mon Mar 26 15:28:45 EDT 2007 Goodbye World
  - Mon Mar 26 15:28:45 EDT 2007 Hello World
  - Mon Mar 26 15:28:46 EDT 2007 Hello World
  - Mon Mar 26 15:28:46 EDT 2007 Goodbye World
  - Mon Mar 26 15:28:47 EDT 2007 Hello World
  - Mon Mar 26 15:28:47 EDT 2007 Goodbye World
  - Mon Mar 26 15:28:48 EDT 2007 Goodbye World
  - Mon Mar 26 15:28:48 EDT 2007 Hello World
  - Mon Mar 26 15:28:49 EDT 2007 Goodbye World
  - Mon Mar 26 15:28:49 EDT 2007 Hello World
  - Mon Mar 26 15:28:50 EDT 2007 Goodbye World
  - Mon Mar 26 15:28:50 EDT 2007 Hello World
  - Mon Mar 26 15:28:51 EDT 2007 Goodbye World
  - Mon Mar 26 15:28:51 EDT 2007 Hello World
  - Mon Mar 26 15:28:52 EDT 2007 Hello World
  - Mon Mar 26 15:28:52 EDT 2007 Goodbye World
  - Mon Mar 26 15:28:53 EDT 2007 Hello World
- Two message sets (Hello and Goodbye) are not exactly interleaved.

31

## The Order of Thread Execution

- JVM's thread scheduler gives you NO guarantee about the order of thread execution.
- There are always slight variations in the time to execute a threaded task
  - especially when calling OS system calls (typically I/O related system calls)
- Expect that the order of thread execution is somewhat random.

32

## Exercise

- Modify HelloWorldTest2.java;
  - Replace the following 4 lines
    - Thread thread1 = **new** Thread(runnable1);
    - Thread thread2 = **new** Thread(runnable2);
    - thread1.start();
    - thread2.start();
  - with the following 2 lines:
    - runnable1.run();
    - runnable2.run();
- What does the program output?
- Modify HelloWorldTest2.java;
  - Replace the following 4 lines
    - Thread thread1 = **new** Thread(runnable1);
    - Thread thread2 = **new** Thread(runnable2);
    - thread1.start();
    - thread2.start();
  - with the following 2 lines:
    - runnable1.run();
    - runnable2.run();
- Output: “Hello World” in 10 lines, followed by “Good bye” in 10 lines.
  - The program runs **sequentially**, not concurrently.