# Lecture 8

# Java SE – Multithreading

presentation

**Java Programming – Software App Development**

**Cristian Toma**

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro

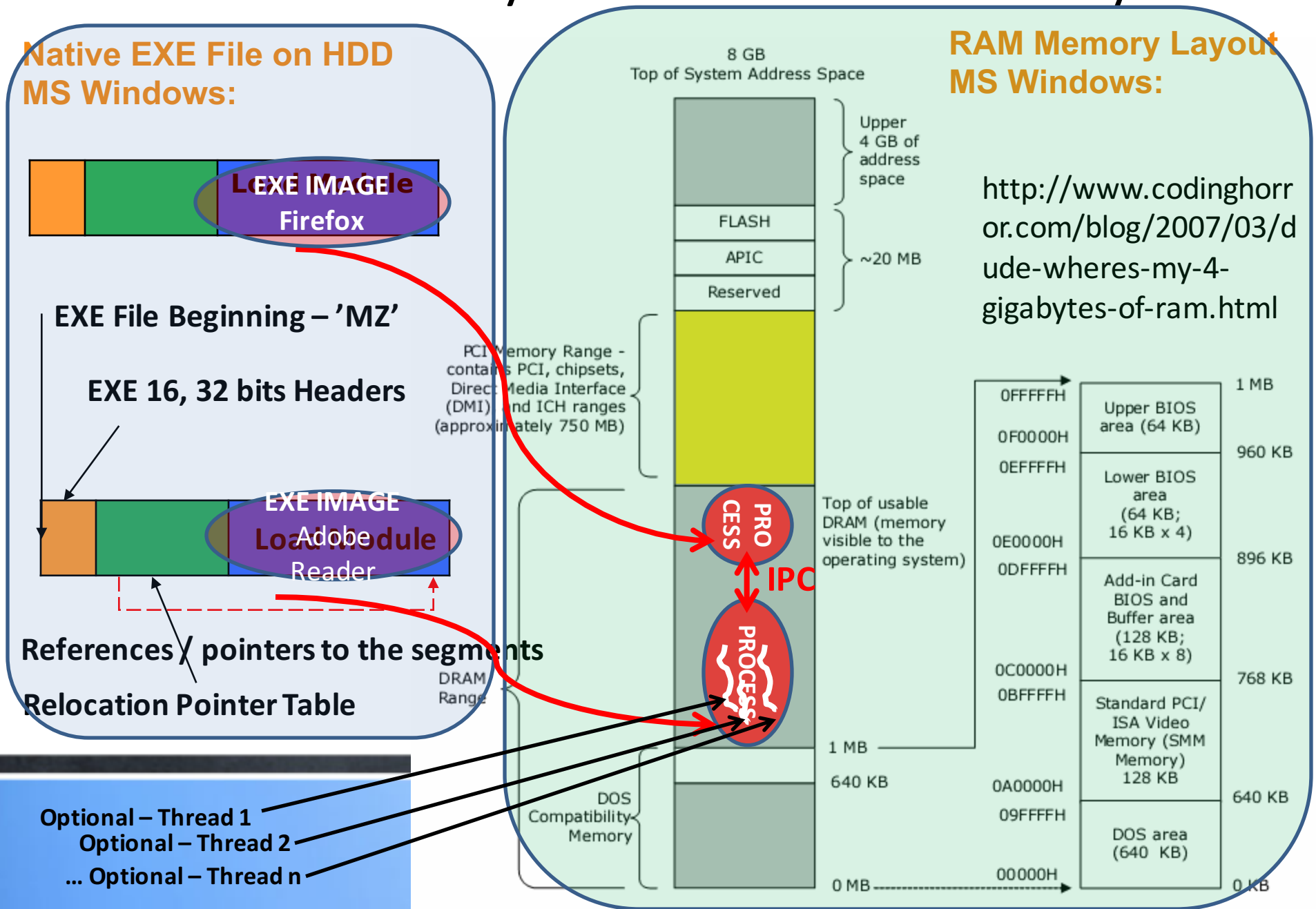
# **Cristian Toma** – Business Card

# **Agenda** for Lecture 8 – JSE Multi-threading

Multi-threading models, features, atomic operations, JVM and OS threads

# **Multi-threading** vs. Multi-process

# 1.1 Summary of MS Windows Memory



# 1.2 Summary of Processes & IPC in Linux

http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

https://computing.llnl.gov/tutorials/pthreads/ http://www.advancedlinuxprogramming.com/alp-folder/

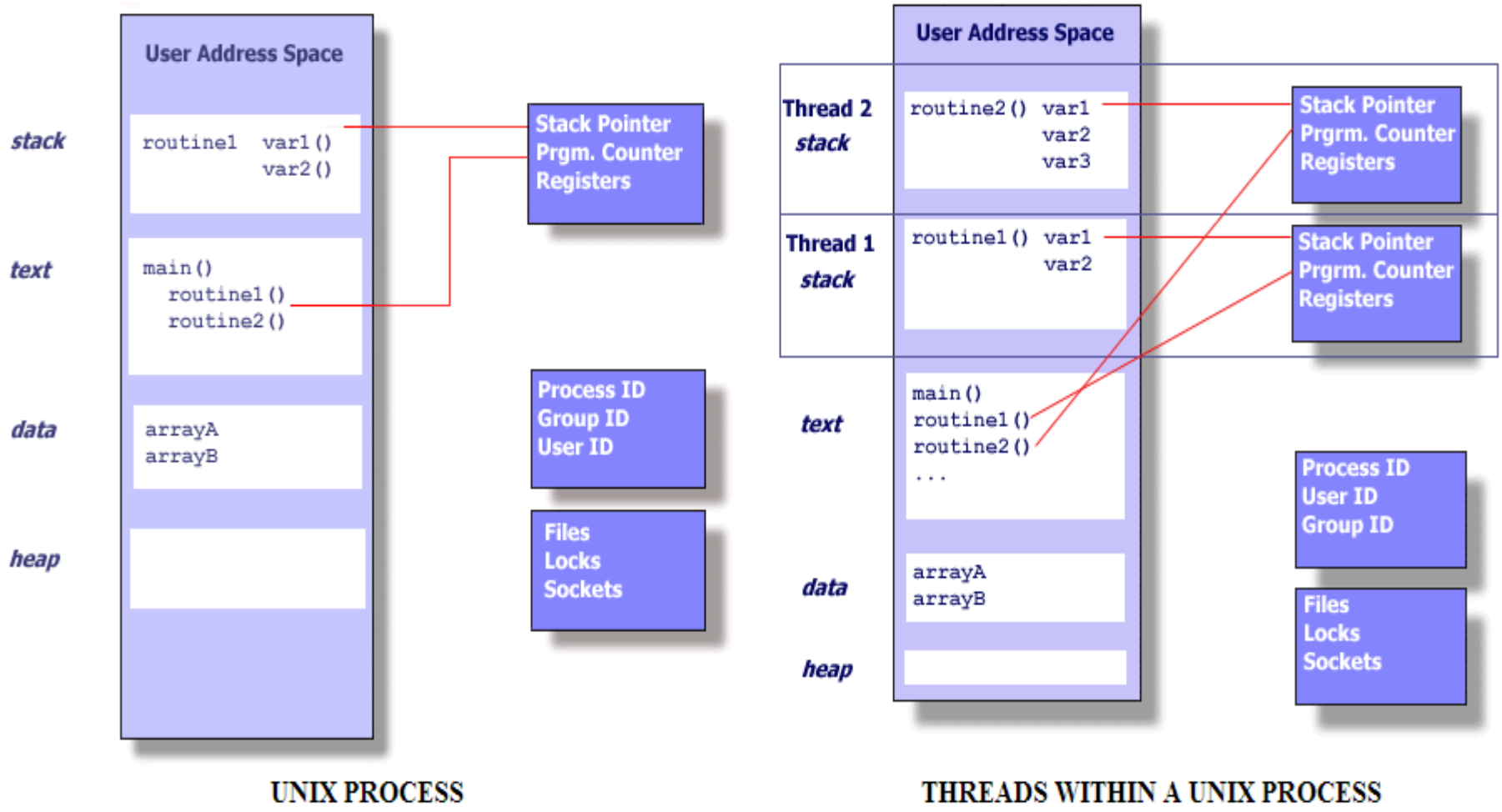
Before understanding a thread, one first needs to understand a UNIX process. A process is created by the operating system, and requires a fair amount of "overhead".

Processes contain information about program resources & program execution state, including:

- *Process ID, process group ID, user ID, and group ID;*
- *Environment;*
- *Working directory;*
- *Program instructions;*
- *Registers;*
- *Stack;*
- *Heap;*
- *File descriptors;*
- *Signal actions;*
- *Shared libraries;*
- *Inter-process communication tools (such as message queues, pipes, semaphores, or shared memory)*

# 1.2 Summary of Multi-threading in C vs. Java

## Multi-threading vs. Multi-process development in UNIX/Linux:




https://computing.llnl.gov/tutorials/pthreads/

# 1.2 Summary of Multi-threading in C vs. Java

## Threads Features:

Thread operations include thread creation, termination, synchronization (joins, blocking), scheduling, data management and process interaction.

A thread does not maintain a list of created threads, nor does it know the thread that created it.

All threads within a process share the same address space.

Threads in the same process share:

- Process instructions
- Most data
- open files (descriptors)
- signals and signal handlers
- current working directory
- User and group id

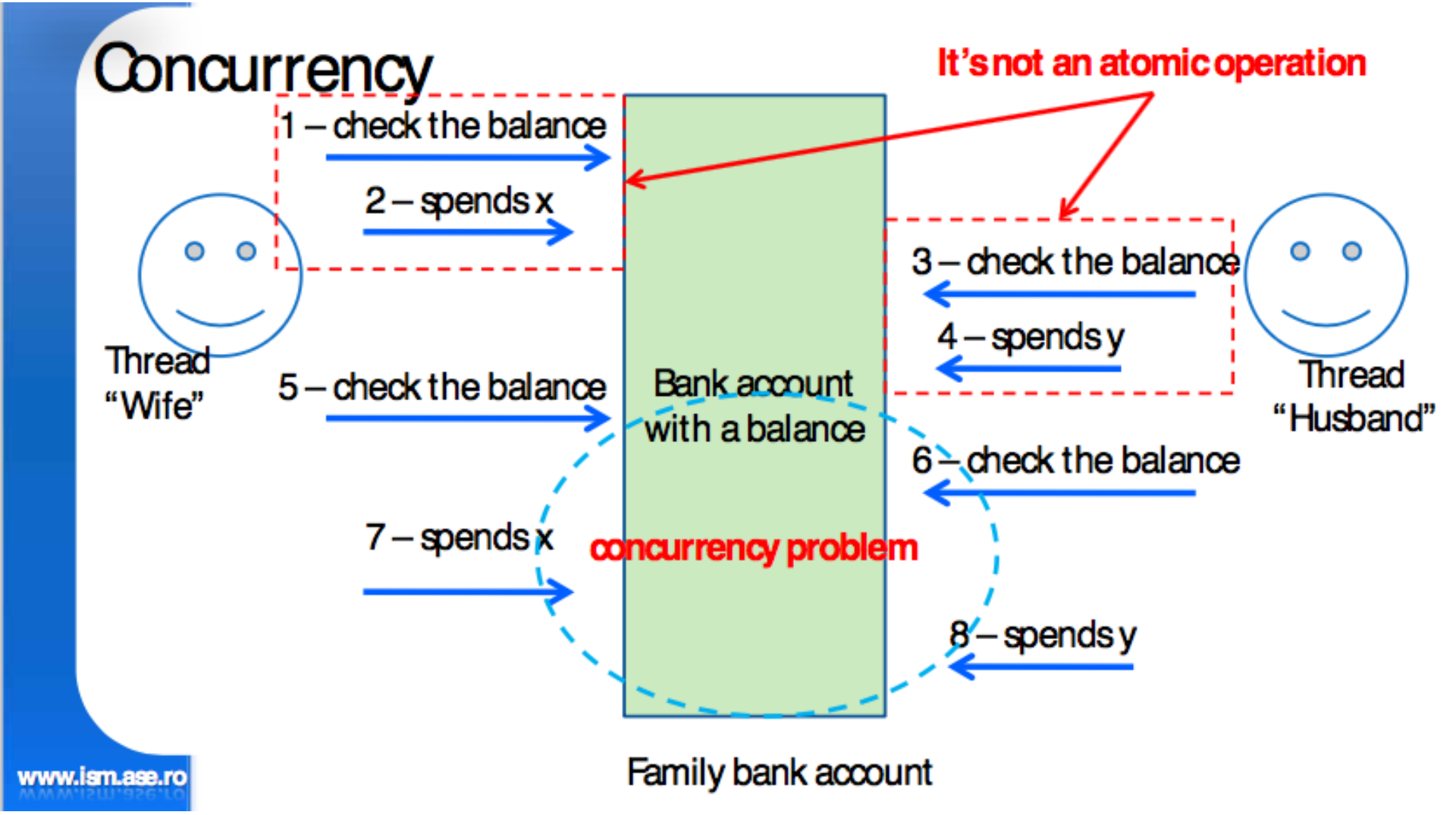
Each thread has a unique:

- Thread ID
- set of registers, stack pointer
- stack for local variables, return addresses
- signal mask
- priority
- Return value: errno

pthread functions return "0" if OK.

# 1.2 Summary of Race Condition

## Race Condition Feature – Family Bank Account:

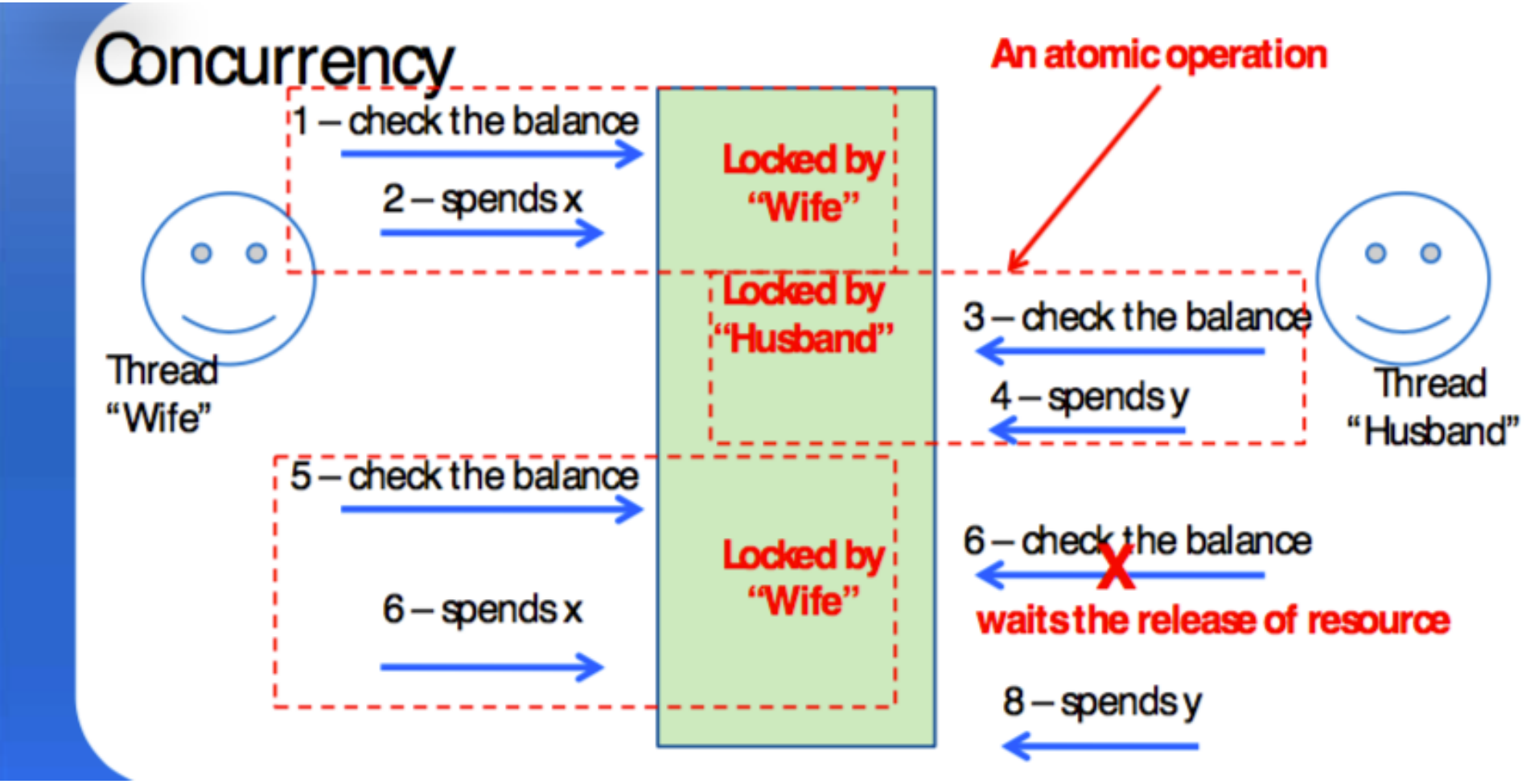





Copyright: Secure Applications Programming – www.ism.ase.ro (Java - Boja)

# 1.2 Summary of Race Condition

## Race Condition Feature – Family Bank Account:




Copyright: Secure Applications Programming – www.ism.ase.ro (Java - Boja)

# 1.2 Summary of Multi-threading in C vs. Java

**Multi-threading in C/C++ with pthread** *(Is "counter++" an atomic operation?)*:

**Without Mutex**

```
1  int counter=0;
2
3  /* Function C */
4  void functionC()
5  {
6
7      counter++
8
9  }
```

**With Mutex**

```
01 /* Note scope of variable and mutex are the same
   */
02 pthread_mutex_t mutex1 =
   PTHREAD_MUTEX_INITIALIZER;
03 int counter=0;
04
05 /* Function C */
06 void functionC()
07 {
08     pthread_mutex_lock( &mutex1 );
09     counter++
10     pthread_mutex_unlock( &mutex1 );
11 }
```

Possible execution sequence

| Thread 1 | Thread 2 | Thread 1 | Thread 2 |
|---|---|---|---|
| counter = 0 | counter = 0 | counter = 0 | counter = 0 |
| counter = 1 | counter = 1 | counter = 1 | Thread 2 locked out.<br>Thread 1 has exclusive use of variable `counter` |
| | | | counter = 2 |

# 1.2 Summary of Multi-threading in C vs. Java

**Multi-threading vs. Multi-process "quite good/imperfect" analogies & terms:**

USE MULTI-THREADING for

- PARALLELISM (e.g. adding 2 matrixes in parallel) and/or;
- CONCURENCY (e.g. handling GUI on one thread & business processing on other thread / even on mono-processor PC, handling multiple HTTP requests from the network)

Mutexes are used to prevent data inconsistencies due to ***race conditions***.
A ***race condition*** often occurs when two or more threads need to perform operations on the same memory area, but the results of computations depends on the order in which these operations are performed.

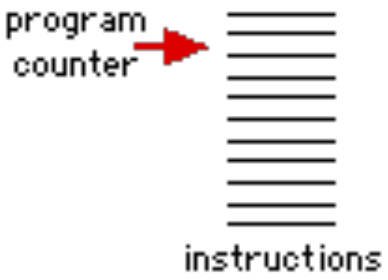
Mutexes are used for serializing shared resources. Anytime a global resource is accessed by more than one thread the resource should have a Mutex associated with it.
One can apply a mutex to protect a segment of memory ("***critical region***") from other threads.
Mutexes can be applied only to threads in a single process and do not work between processes as do semaphores in Linux IPC.

In Java Mutex is quite ⇔ synchronized (also please see java.util.concurrent.*)

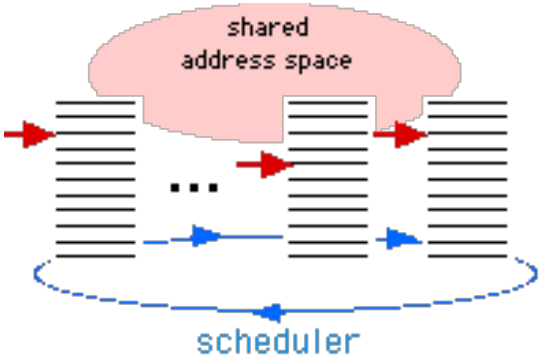
# 1.2 Summary of Multi-threading in C vs. Java

## Multi-threading Models:

http://www-sop.inria.fr/indes/rp/FairThreads/FTJava/documentation/FairThreads.html#One-one-mapping

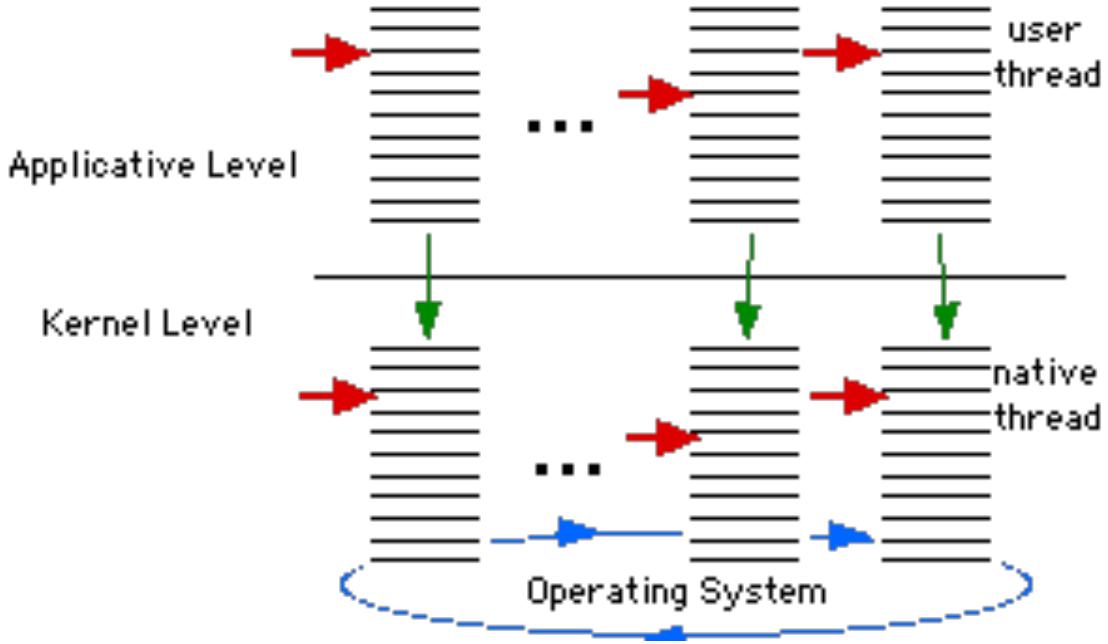



A thread




A scheduler

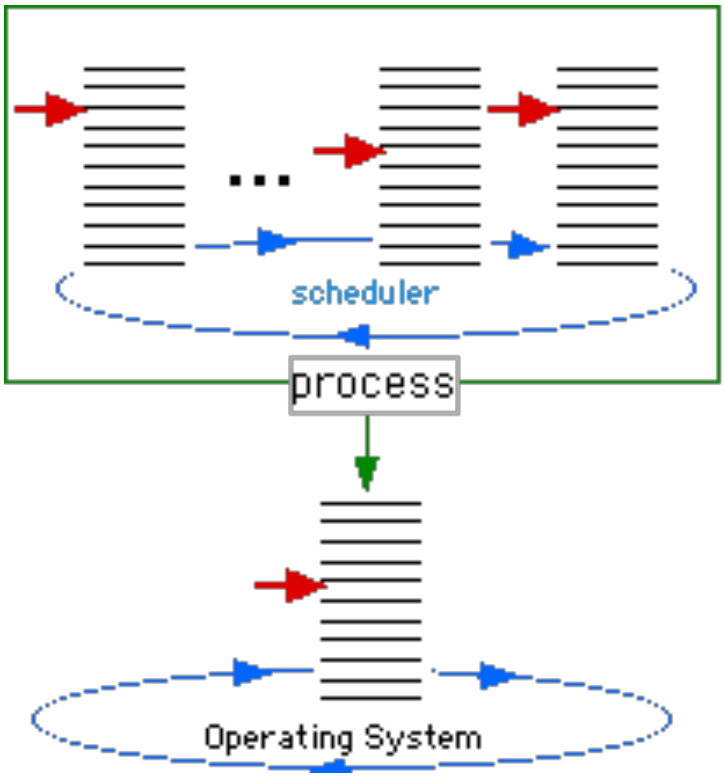
One-to-one Mapping



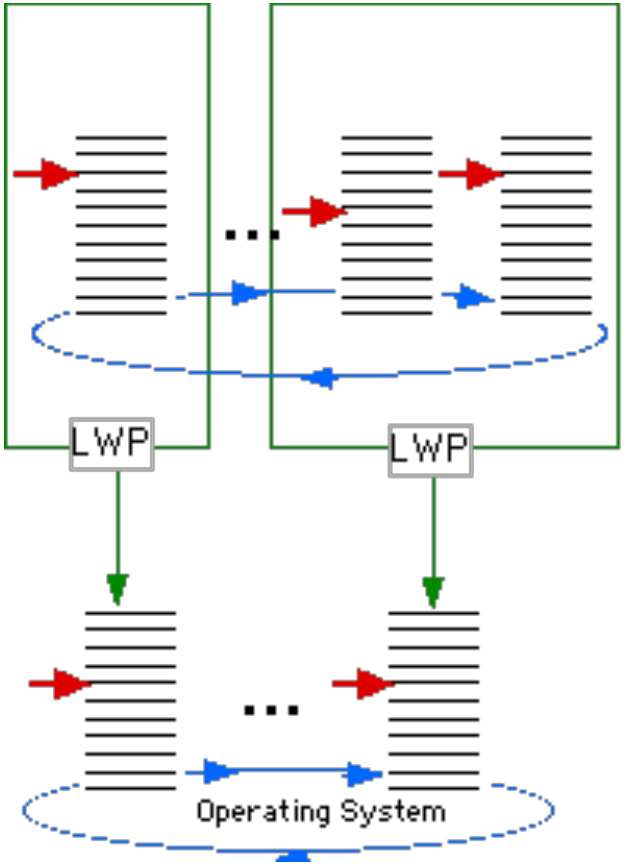
# 1.2 Summary of Multi-threading in C vs. Java

## Multi-threading Models:

http://www-sop.inria.fr/indes/rp/FairThreads/FTJava/documentation/FairThreads.html#One-one-mapping



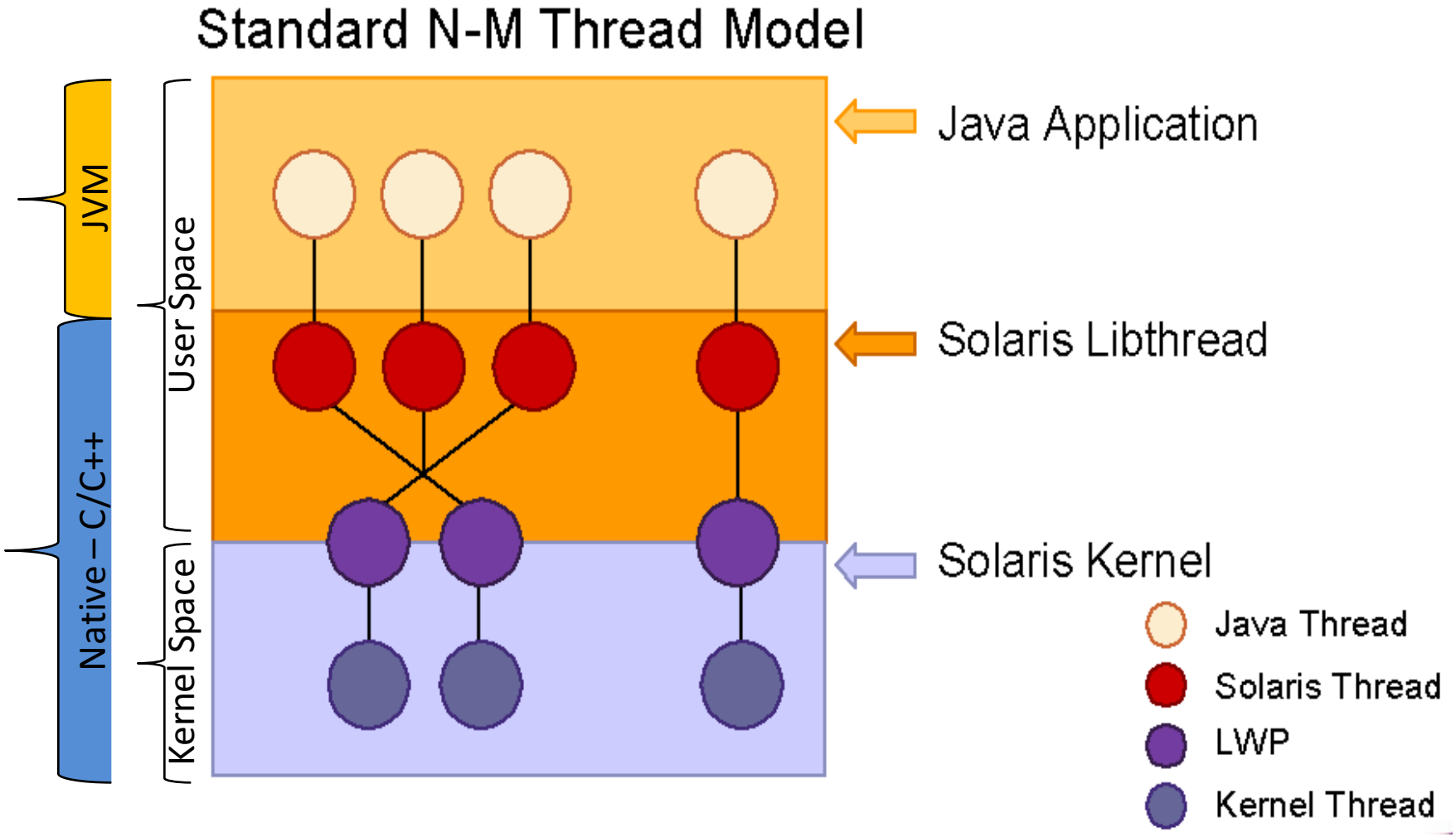

Many-to-one Mapping




Many-to-many Mapping

# 1.2 Summary of Multi-threading in C vs. Java

## JVM Multi-threading Model mapping to OS native threads:

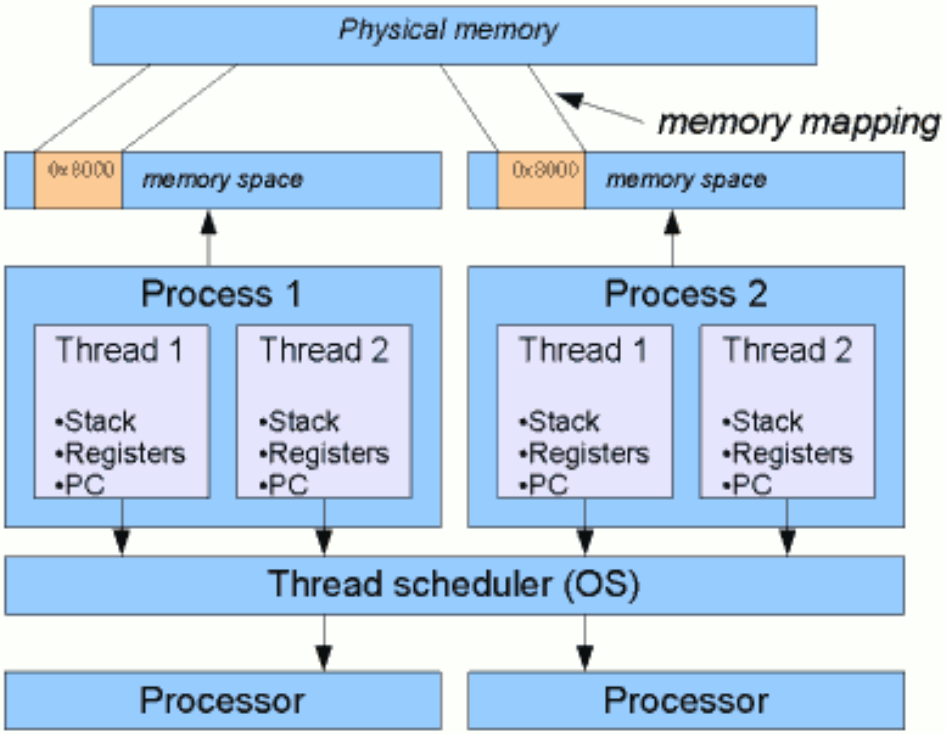


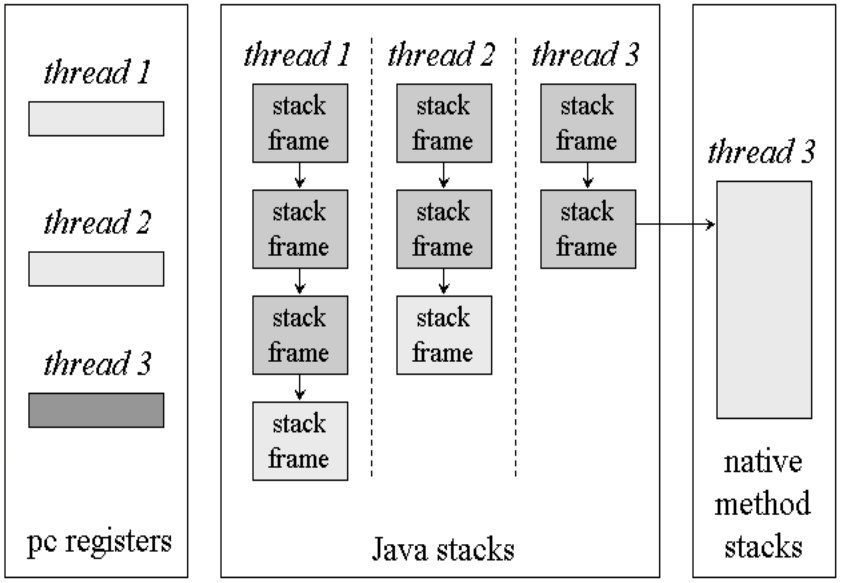

http://192.9.162.55/docs/hotspot/threads/threads.html

# 1.2 Summary of Multi-threading in C vs. Java

## JVM Multi-threading Model mapping to OS native threads:

**http://www.javamex.com/tutorials/threads/how_threads_work.shtml**



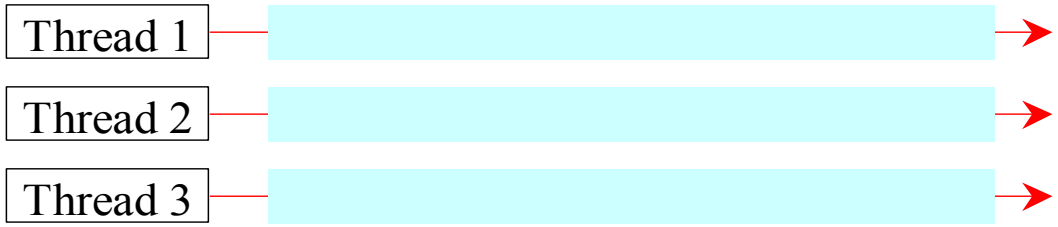




**Inside the Java Virtual Machine, Bill Venners**

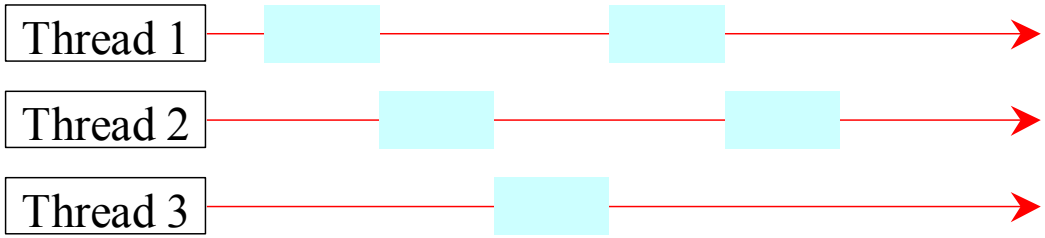
# 1.2 Summary of Multi-threading in Java

## JVM Multi-threading Concept:

Multiple threads on multiple CPUs




Multiple threads sharing a single CPU




- Different processes do not share memory space.
- A thread can execute concurrently with other threads within a single process.
- All threads managed by the JVM share memory space and can communicate with each other.

# 1.3 Threads Issues

## Thread:

- Thread: single sequential flow of control within a program
- Single-threaded program can handle one task at any time.
- Multitasking allows single processor to run several concurrent threads.
- Most modern operating systems support multitasking.

Liang, Introduction to Java Programming, Seventh Edition, (c) 2009 Pearson Education, Inc. All rights reserved. 0136012671

# 1.3 Threads Issues

## Advantages of Multithreading:

- Reactive systems – constantly monitoring
- More responsive to user input – GUI application can interrupt a time-consuming task
- Server can handle multiple clients simultaneously
- Can take advantage of parallel processing

## When Multithreading?:

**Concurrency and/or Parallelism**

Liang, Introduction to Java Programming, Seventh Edition, (c) 2009 Pearson Education, Inc. All rights reserved. 0136012671

# Section Conclusion

Fact: **Multi-threading vs. Multi-process**

In few **samples** it is easy to understand:

- Necessity: Parallelism vs. Concurrency
- Multi-Process vs. Multi-Threading
- Atomic operations (counter++)
- Multi-threading model mapping

**Java API for multi-threading, Java Multi-threading issues – Singleton vs. Immutable Objects**

# Multi-threading in Java & OS Linux

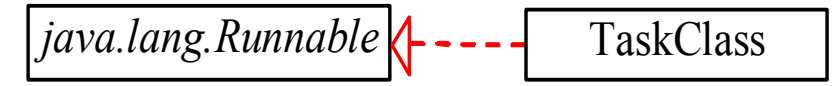
# 2.1 Java Multi-threading API

## Java Multi-threading API

| Option 1 – Java API for Multi-Threading Programming | Option 2 – Java API for Multi-Threading Programming |
|---|---|
| **Defining the classes:** | |
| class MyT extends Thread {<br>  public void run() {…}<br>} | class MyT extends SomeC implements Runnable {<br>  public void run() {…}<br>} |
| **Instantiates the objects** | |
| MyT f = new MyT (); | MyT obf = new MyT();<br>Thread f = new Thread(obf); |
| **Set the thread in 'Runnable' state** | |
| f.start(); | f.start(); |
| **Specific Thread methods calls** | |
| public void run() {<br>  Thread.sleep(…);<br>  String fName = this.getName();<br>  …<br>} | public void run() {<br>  Thread.sleep(…);<br>  Thread t = Thread.currentThread();<br>  String fName = t.getName();<br>  …<br>} |

# 2.1 Java Multi-threading API

## Java Threads API – Creating Tasks & Threads



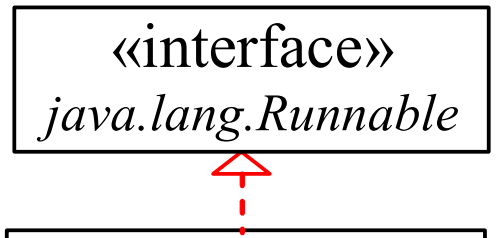

```
// Custom task class
public class TaskClass implements Runnable {
  ...
  public TaskClass(...) {
    ...
  }

  // Implement the run method in Runnable
  public void run() {
    // Tell system how to run custom thread
    ...
  }
  ...
}
```

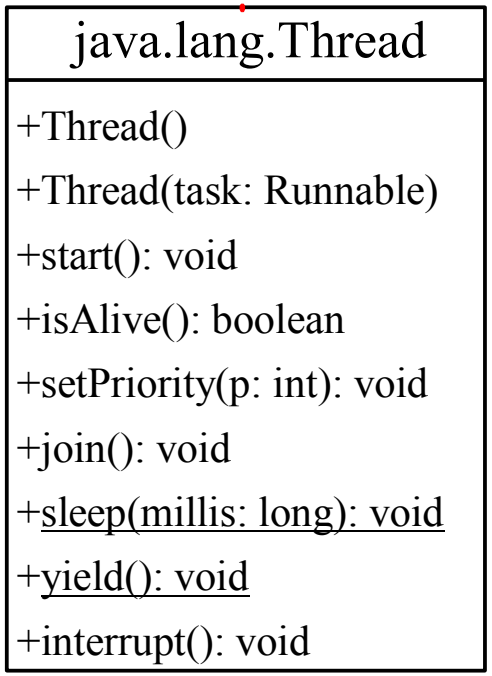
```
// Client class
public class Client {
  ...
  public void someMethod() {
    ...
    // Create an instance of TaskClass
    TaskClass task = new TaskClass(...);

    // Create a thread
    Thread thread = new Thread(task);

    // Start a thread
    thread.start();
    ...
  }
  ...
}
```

Liang, Introduction to Java Programming, Seventh Edition, (c) 2009 Pearson Education, Inc. All rights reserved. 0136012671

# 2.1 Java Multi-threading API

## Java Thread Class




Creates a default thread.

Creates a thread for a specified task.

Starts the thread that causes the run() method to be invoked by the JVM.

Tests whether the thread is currently running.

Sets priority p (ranging from 1 to 10) for this thread.

Waits for this thread to finish.

Puts the runnable object to sleep for a specified time in milliseconds.

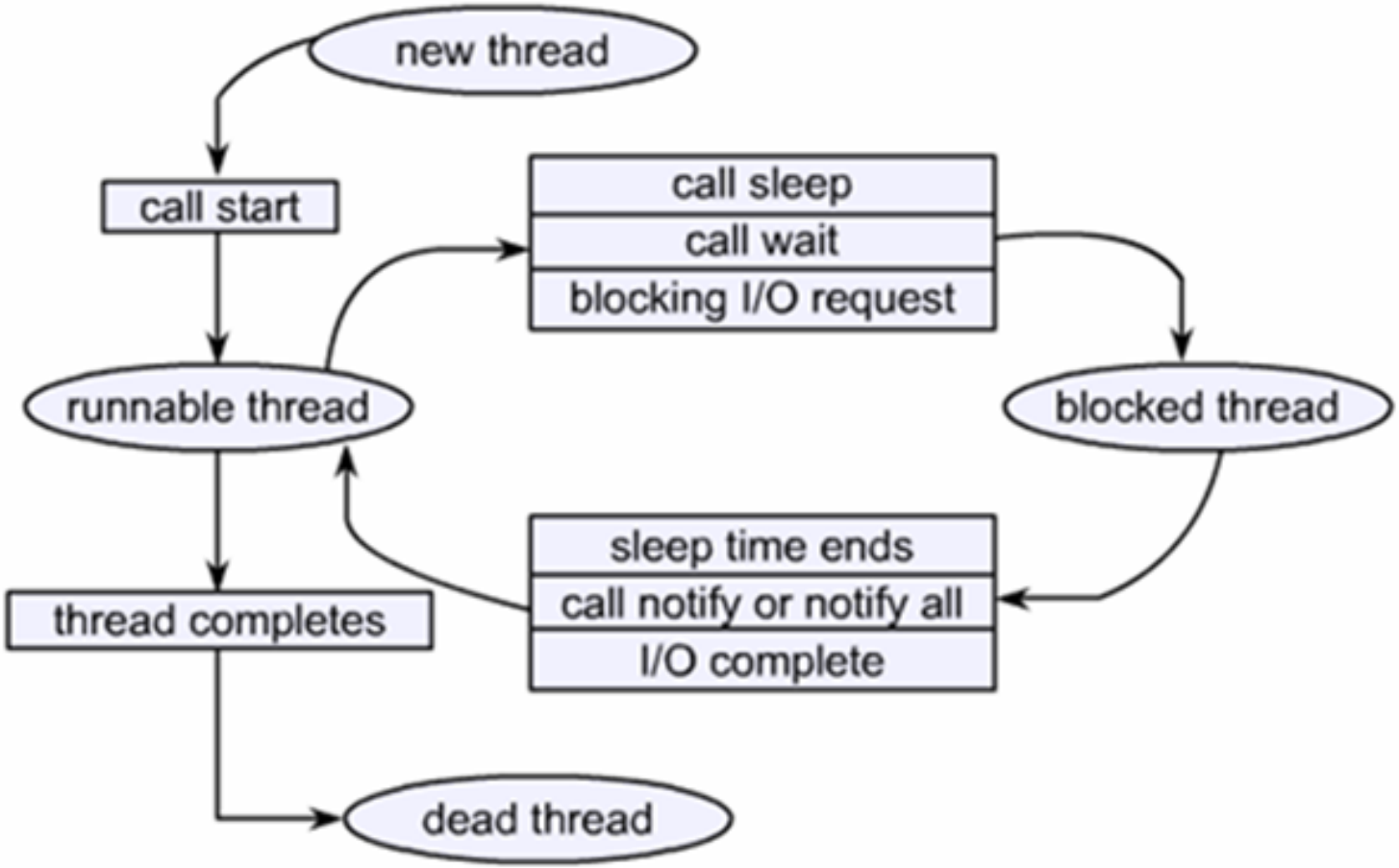
Causes this thread to temporarily pause and allow other threads to execute.

Interrupts this thread.

Liang, Introduction to Java Programming, Seventh Edition, (c) 2009 Pearson Education, Inc. All rights reserved. 0136012671

# 2.1 Java Multi-threading API

## Java Thread States




CISCO Copyright

# 2.2 Is Java Singleton pattern Thread Safe?

```
class ASingleton { // Version 1
    private static ASingleton instance = null;

    private ASingleton() {
    }
    public static ASingleton getInstance() {
        if (instance == null) {
            instance = new ASingleton();
        }
        return instance;
    }
}
```

```
class ASingleton{
// Version 2 - what if we are expecting some parameters
// like DB URL, User, Pass to be passed in the getInstance
private static ASingleton instance = new ASingleton();
 private ASingleton(){
 }
 public static synchronized ASingleton getInstance(){
       return instance;
 }
}
```

# 2.2 Is Java Singleton pattern Thread Safe?

```
class ASingleton { // Version 3 - Now is quite safe
    private static ASingleton instance = null;
    private static Object mutex = new Object();
    private ASingleton(){
    }

    public static ASingleton getInstance() {
        if(instance==null) {
          synchronized (mutex) {
            if(instance == null)
              instance = new ASingleton();
            }
          }
        return instance;
    }
}
```

# 2.2 Is Java Immutable Objects (see previous OOP lecture) – Java Class Pattern Thread Safe?

## Immutable Classes and Objects.

If the contents of an object cannot be changed once the object is created, the object is called an *immutable object* and its class is called an *immutable class*. If you delete the set method in the **Circle** class, the class would be immutable because radius is private and cannot be changed without a set method.

A class with all private data fields and without mutators is not necessarily immutable. For example, the following class **Student** has all private data fields and no mutators, but it is **mutable**.

## What Class is immutable?

*For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.*

Liang, Introduction to Java Programming, Eighth Edition, (c) 2011 Pearson Education, Inc. All rights reserved. 0132130807: http://akyokus.com/courses/Fall2011/ise204/Lectures/Liang8e/10slide.ppt

# 2.2 Is Java Immutable Objects (see previous OOP lecture) – Java Class Pattern Thread Safe?

**Immutable Classes and Objects**.

1) All the fields must be private and preferably final
2) Ensure the class cannot be overridden - make the class final, or use static factories and keep constructors private
3) Fields must be populated from the Constructor/Factory
4) Don't provide any setters for the fields
5) Watch out for collections. Use Collections.unmodifiable*. Also, collections should contain only immutable Objects
6) All the getters must provide immutable objects or use defensive copying
7) Don't provide any methods that change the internal state of the Object

Read more: http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html#ixzz2xrtHsjqL

# 2.2 Is Java Immutable Objects (see previous OOP lecture) – Java Class Pattern Thread Safe?

## Immutable Classes and Objects.

```
public class Student {
  private int id;
  private BirthDate birthDate;

  public Student(int ssn,
      int year, int month, int day) {
    id = ssn;
    birthDate = new BirthDate(year, month, day);
  }

  public int getId() {
    return id;
  }

  public BirthDate getBirthDate() {
    return birthDate;
  }
}
```

```
public class BirthDate {
  private int year;
  private int month;
  private int day;

  public BirthDate(int newYear,
      int newMonth, int newDay) {
    year = newYear;
    month = newMonth;
    day = newDay;
  }

  public void setYear(int newYear) {
    year = newYear;
  }
}
```

```
public Date getDateOfBirth() {
  return new Date(dateOfBirth.getTime());
} //defensive copy instead of original meth.
```

```
public class Test {
  public static void main(String[] args) {
    Student student = new Student(111223333, 1970, 5, 3);
    BirthDate date = student.getBirthDate();
    date.setYear(2010); // Now the student birth year is changed!
  }
}
```

Liang, Introduction to Java Programming, Eighth Edition, (c) 2011 Pearson Education, Inc. All rights reserved. 0132130807: http://akyokus.com/courses/Fall2011/ise204/Lectures/Liang8e/10slide.ppt

# 2.2 Is Java Immutable Objects (see previous OOP lecture) – Java Class Pattern Thread Safe?

## Immutable Classes and Objects – Are THREAD-SAFE?

An object is considered *immutable* if its state cannot change after it is constructed.

Maximum reliance on immutable objects is widely accepted as a sound strategy for creating simple, reliable code.

Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state.

Programmers are often reluctant to employ immutable objects, because they worry about the cost of creating a new object as opposed to updating an object in place. The impact of object creation is often overestimated, and can be offset by some of the efficiencies associated with immutable objects. These include decreased overhead due to garbage collection, and the elimination of code needed to protect mutable objects from corruption.

**http://docs.oracle.com/javase/tutorial/essential/concurrency/immutable.html**

# 2.2 What Software Design Pattern is impacted by Multithreading?

**Creational patterns**

| Name | Description |
|---|---|
| Abstract factory | Provide an interface for creating *families* of related or dependent objects without specifying their concrete classes. |
| Builder | Separate the construction of a complex object from its representation allowing the same construction process to create various representations. |
| Factory method | Define an interface for creating a *single* object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses (dependency injection[16]). |
| Lazy initialization | Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed. This pattern appears in the GoF catalog as "virtual proxy", an implementation strategy for the Proxy pattern. |
| Multiton | Ensure a class has only named instances, and provide global point of access to them. |
| Object pool | Avoid expensive acquisition and release of resources by recycling objects that are no longer in use. Can be considered a generalisation of connection pool and thread pool patterns. |
| Prototype | Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype. |
| Resource acquisition is initialization | Ensure that resources are properly released by tying them to the lifespan of suitable objects. |
| Singleton | Ensure a class has only one instance, and provide a global point of access to it. |

http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

# 2.2 What Software Design Pattern is impacted by Multithreading?

**Structural patterns**

| Name | Description |
|---|---|
| Adapter or Wrapper or Translator. | Convert the interface of a class into another interface clients expect. An adapter lets classes work together that could not otherwise because of incompatible interfaces. The enterprise integration pattern equivalent is the translator. |
| Bridge | Decouple an abstraction from its implementation allowing the two to vary independently. |
| Composite | Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. |
| Decorator | Attach additional responsibilities to an object dynamically keeping the same interface. Decorators provide a flexible alternative to subclassing for extending functionality. |
| Facade | Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. |
| Flyweight | Use sharing to support large numbers of similar objects efficiently. |
| Front Controller | The pattern relates to the design of Web applications. It provides a centralized entry point for handling requests. |
| Module | Group several related elements, such as classes, singletons, methods, globally used, into a single conceptual entity. |
| Proxy | Provide a surrogate or placeholder for another object to control access to it. |
| Twin [18] | Twin allows to model multiple inheritance in programming languages that do not support this feature. |

http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

# 2.2 What Software Design Pattern is impacted by Multithreading?

**Behavioral patterns**

| Name | Description |
|---|---|
| Blackboard | Generalized observer, which allows multiple readers and writers. Communicates information system-wide. |
| Chain of responsibility | Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it. |
| Command | Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations. |
| Interpreter | Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language. |
| Iterator | Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation. |
| Mediator | Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently. |
| Memento | Without violating encapsulation, capture and externalize an object's internal state allowing the object to be restored to this state later. |
| Null object | Avoid null references by providing a default object. |
| Observer or Publish/subscribe | Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically. |
| Servant | Define common functionality for a group of classes |
| Specification | Recombinable business logic in a Boolean fashion |
| State | Allow an object to alter its behavior when its internal state changes. The object will appear to change its class. |
| Strategy | Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it. |
| Template method | Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. |
| Visitor | Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates. |

http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

# 2.2 What Software Design Pattern is impacted by Multithreading?

**Concurrency patterns**

| Name | Description |
|---|---|
| Active Object | Decouples method execution from method invocation that reside in their own thread of control. The goal is to introduce concurrency, by using asynchronous method invocation and a scheduler for handling requests. |
| Balking | Only execute an action on an object when the object is in a particular state. |
| Binding properties | Combining multiple observers to force properties in different objects to be synchronized or coordinated in some way.[20] |
| Double-checked locking | Reduce the overhead of acquiring a lock by first testing the locking criterion (the 'lock hint') in an unsafe manner; only if that succeeds does the actual lock proceed. Can be unsafe when implemented in some language/hardware combinations. It can therefore sometimes be considered an anti-pattern. |
| Event-based asynchronous | Addresses problems with the asynchronous pattern that occur in multithreaded programs.[21] |
| Guarded suspension | Manages operations that require both a lock to be acquired and a precondition to be satisfied before the operation can be executed. |
| Join | Join-patterns provides a way to write concurrent, parallel and distributed programs by message passing. Compared to the use of threads and locks, this is a high level programming model. |
| Lock | One thread puts a "lock" on a resource, preventing other threads from accessing or modifying it.[22] |
| Messaging design pattern (MDP) | Allows the interchange of information (i.e. messages) between components and applications. |
| Monitor object | An object whose methods are subject to mutual exclusion, thus preventing multiple objects from erroneously trying to use it at the same time. |
| Reactor | A reactor object provides an asynchronous interface to resources that must be handled synchronously. |
| Read-write lock | Allows concurrent read access to an object, but requires exclusive access for write operations. |
| Scheduler | Explicitly control when threads may execute single-threaded code. |
| Thread pool | A number of threads are created to perform a number of tasks, which are usually organized in a queue. Typically, there are many more tasks than threads. Can be considered a special case of the object pool pattern. |
| Thread-specific storage | Static or "global" memory local to a thread. |

http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

Multi-threading & IPC Summary
for easy sharing

# Section Conclusions

All threads in a program must run the same executable. A child process, on the other hand, may run a different executable by calling an exec function.

An errant thread can harm other threads in the same process because threads share the same virtual memory space and other resources. For instance, a wild memory write through an uninitialized pointer in one thread can corrupt memory visible to another thread.

An errant process, on the other hand, cannot do so because each process has a copy of the program's memory space.

Copying memory for a new process adds an additional performance overhead relative to creating a new thread. However, the copy is performed only when the memory is changed, so the penalty is minimal if the child process only reads memory.

Threads should be used for programs that need fine-grained parallelism. For example, if a problem can be broken into multiple, nearly identical tasks, threads may be a good choice. Processes should be used for programs that need coarser parallelism.

Sharing data among threads is trivial because threads share the same memory. However, great care must be taken to avoid race conditions. Sharing data among processes requires the use of IPC mechanisms. This can be more cumbersome but makes multiple processes less likely to suffer from concurrency bugs.

Share knowledge, Empowering Minds

# Communicate & Exchange Ideas

Questions & Answers!

# But wait…

There’s More!

# Thanks!

Java Programming – Software App Development
End of Lecture 8 – Java SE Multithreading