ECE 462
Object-Oriented Programming using C++ and Java

Lecture 14

Yung-Hsiang Lu
yunglu@purdue.edu

- VirtualBase.cc
- VirtualBaseCopyConstruct.cc (modified with deep copy and destructor)
- VirtualBaseAssign.cc (modified)
- The demonstration is conducted in Windows using cygwin so the executable files have .exe extensions.

Java Applet

- programs transmitted through web browsers and executed in the clients
- security restrictions:
  - cannot access clients’ file systems
  - cannot communicate through Internet
  - cannot execute any program in clients
- invoked by HTML using <applet code=xxx.class width=... height=...></applet>

Watch.html

load Watch.html in a web browser or appletviewer Watch.html

```html
<html>
<head>
<title>Watch Applet</title>
</head>
<body>
<h1>Java Watch</h1>
<applet code="Watch.class" height="50" width="345">
This program requires a Java-enabled browser.
</applet>
</body>
</html>
```

```
// Watch.java (from Teach Yourself Java 2 in 21 Days)
import java.awt.*;
import java.util.*;
public class Watch extends javax.swing.JApplet {
   private String lastTime = "";
   private int height = 0;
   private int width = 0;
   public void init() {          // no main function
      setBackground(Color.white);
      height = Integer.valueOf(getParameter("height")).intValue();
      width = Integer.valueOf(getParameter("width")).intValue();
   }
   public void paint(Graphics screen) {
      Graphics2D screen2D = (Graphics2D)screen;
      Font type = new Font("Monospaced", Font.BOLD, 20);
      screen2D.setFont(type);
```

create Watch.class by running javac Watch.java

```java
        GregorianCalendar day = new GregorianCalendar();
        String time = day.getTime().toString();
        screen2D.setColor(Color.white);
        screen2D.fillRect(0, 0, width, height);  // erase the previous time
        screen2D.setColor(Color.black);
        screen2D.drawString(time, 5, 25);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            // do nothing
        }
        lastTime = time;
        repaint();
    }
}
```

Procedure to Create Applet

- write HTML page with <applet code="XXX.class" height="YYY" width="ZZZ">
- create public class XXX extends JApplet
- remove main and JFrame object
- initialize in the **init** function
- do not call setSize, setTitle, setVisible(true)
- four special functions in Applet
  - init: initialize the Applet, assign values to attributes, always needed
  - start: called when the Applet is visible on a browser
  - stop: execute when invisible (minimized, blocked ...), to conserve the resources of the client
  - destroy: not needed in most Applets

Calculator

```html
<!--- HTML comment: from Core Java 2 Volume 1-Fundamentals --->
<!--- http://horstmann.com/corejava.html, Further Information,
    Download Code --->
<html>
  <head><title>A Calculator</title></head>
  <body>
    <p>Here is a calculator, just in case you can't find yours.</p>
    <applet code="CalculatorApplet.class" width="180" height="180">
    </applet>
  </body>
</html>
```

```java
/**
   @version 1.31 2004-05-07
   @author Cay Horstmann
*/
// CalculatorApplet.java
import java.awt.*;
import javax.swing.*;
public class CalculatorApplet extends JApplet
{
   public void init()
   {
      CalculatorPanel panel = new CalculatorPanel();
      getContentPane().add(panel);
   }
}
```

```
/**
   @version 1.31 2004-05-07
   @author Cay Horstmann
*/
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/**
   A panel with calculator buttons and
     a result display.
*/
class CalculatorPanel extends JPanel
{
   public CalculatorPanel()
   {
      setLayout(new BorderLayout());
      result = 0;
```

```
lastCommand = "=";
start = true;
 // add the display
display = new JLabel("0");
add(display,
BorderLayout.NORTH);
ActionListener insert = new
InsertAction();
ActionListener command = new
CommandAction();
// add the buttons in a 4 x 4 grid
panel = new JPanel();
panel.setLayout(new
GridLayout(4, 4));
addButton("7", insert);
addButton("8", insert);
addButton("9", insert);
addButton("/", command);
```

```
    addButton("4", insert);
    addButton("5", insert);
    addButton("6", insert);
    addButton("*", command);
    addButton("1", insert);
    addButton("2", insert);
    addButton("3", insert);
    addButton("-", command);
    addButton("0", insert);
    addButton(".", insert);
    addButton("=", command);
    addButton("+", command);
    add(panel,
   BorderLayout.CENTER);
}
```

```
/**       Adds a button to the center
   panel.
    @param label the button label
    @param listener the button
   listener    */
private void addButton(String label,
   ActionListener listener)
{
    JButton button = new
   JButton(label);
   button.addActionListener(listener);
    panel.add(button);
}
/**       This action inserts the button
   action string to the
    end of the display text.    */
private class InsertAction implements
   ActionListener
{
```

```
public void
actionPerformed(ActionEvent
  event)
  {
     String input =
  event.getActionCommand();
     if (start)
     {
        display.setText("");
        start = false;
     }
    display.setText(display.getText()
  + input);
  }
}
/**      This action executes the
  command that the button
  action string denotes.    */
```

```
private class CommandAction
implements ActionListener
{
 public void
 actionPerformed(ActionEvent
 event)
 {
    String command =
 event.getActionCommand();
    if (start)
    {
       if (command.equals("-"))
       {
      display.setText(command);
          start = false;
       }
       else
          lastCommand = command;
```

```
        }
      else
      {
   calculate(Double.parseDouble(dis
   play.getText()));
        lastCommand = command;
        start = true;
      }
    }
  }
  /**
    Carries out the pending
   calculation.
   @param x the value to be
   accumulated with the prior result.
  */
  public void calculate(double x)
  {
    if (lastCommand.equals("+"))
    result += x;
     else if (lastCommand.equals("-"))
    result -= x;
     else if (lastCommand.equals("*"))
    result *= x;
     else if (lastCommand.equals("/"))
    result /= x;
     else if (lastCommand.equals("="))
    result = x;
     display.setText("" + result);
  }
  private JLabel display;
  private JPanel panel;
  private double result;
  private String lastCommand;
  private boolean start;
}
```

Lab 8: Creating a Java Applet

Dual-Purpose Program

- single program for both application and applet
- in the applet (SlideShow) class
  - set inApplet to be true
  - create a constructor, set inApplet to false, call init
  - in init, if inApplet is true, call getParameter
- in public static void main:
  - create a JFrame
  - create an object from the applet (SlideShow) class
  - add the object, pack, setSize, and setVisible
- application: call main
- applet: call init

Parallel Programming

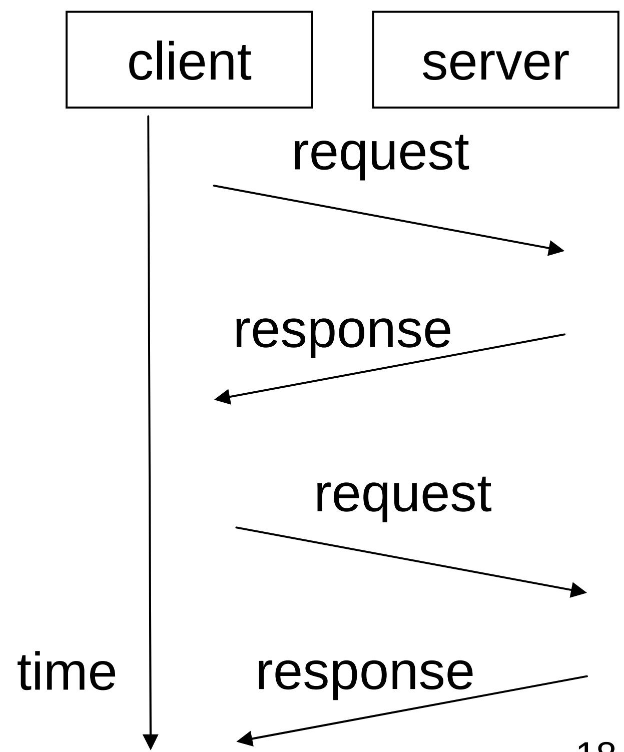
- Sequential programs are no longer sufficient to meet today's needs.
- A web server must handle many viewers.
- A eCommerce server must process many transactions simultaneously.
- Most computers shipped today have multiple cores.
- National Science Foundation gives Purdue ECE $0.8M to teach the concept of parallelism in computer courses.
- Companies need engineers that understand parallelism.
- OOP is a natural approach for parallelism. Objects are **independent** and **active**.

Programming Model

- Multi-threading is one of the most popular, but not the only available programming model.
- Other models include
  - client-server
  - parallel loop, do-all
  - message passing
  - producer-consumer
  - pipeline
  - branch and join

Client Server

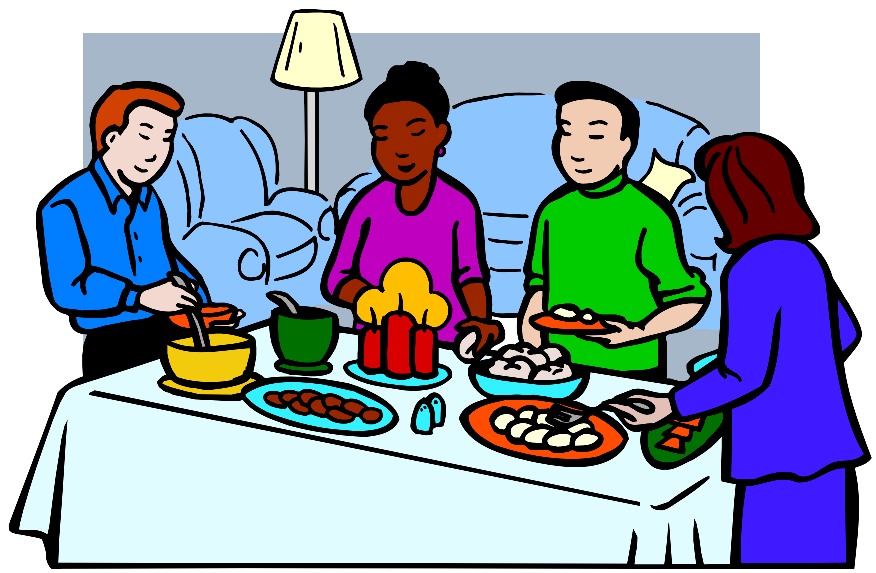
- web browser (client) —— web server
- client sends request, server responds
- The client and the server may (in fact, very likely) be mulit-thread individually.
- synchronization: ask and wait



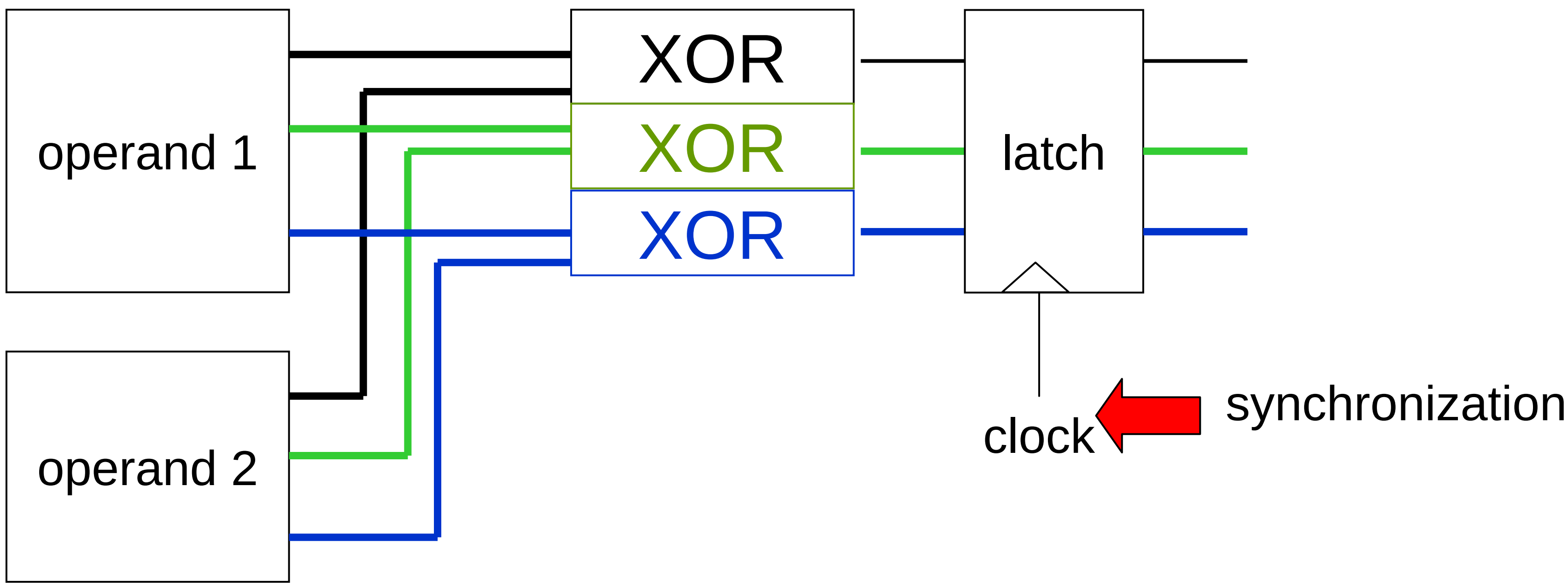
Pipeline

stage 1 → buffer → stage 2 → buffer → stage 3

- instruction-level parallelism
- factory assembly line
- buffet line



Hardware Parallelism (3-bit XOR)



(Adders need to propagate carry signals.)

Why is Multiple Thread Important?

- The industry needs thousands of programmers.
- Today’s new machines have multiple cores and true thread-level parallelism is here.
- Multiple threads allow a program to respond to different requests quickly.
- You have been using multi-thread programs for years: web browser and most GUI programs.
- one of the most popular programming models for parallelism / concurrency
- supported by many languages and operating systems
- It demonstrates many important concepts.

What is a “thread”?

- A thread is (almost) an independent program except
  - different threads created in the same program can access shared variables / objects
  - smaller overhead compared with processes in context switch
  - once a thread starts running, it runs independently until completion or synchronization
- context switch = the processor changes which thread to execute

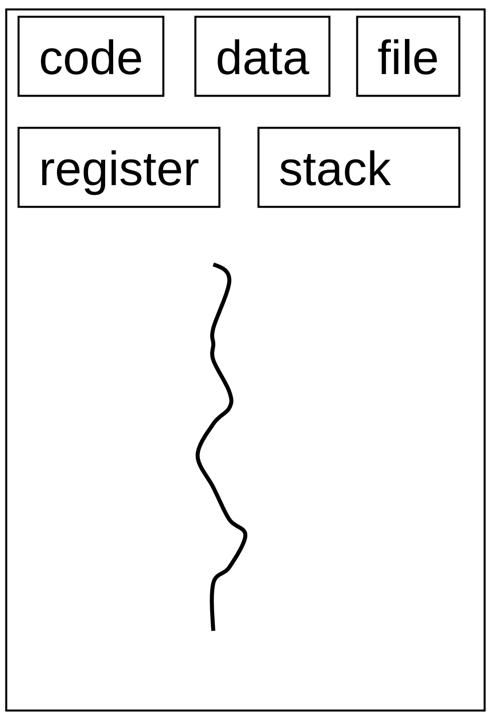
Advantages of Multi-Thread

- Small overhead in creating a thread, compared with processes, example in C

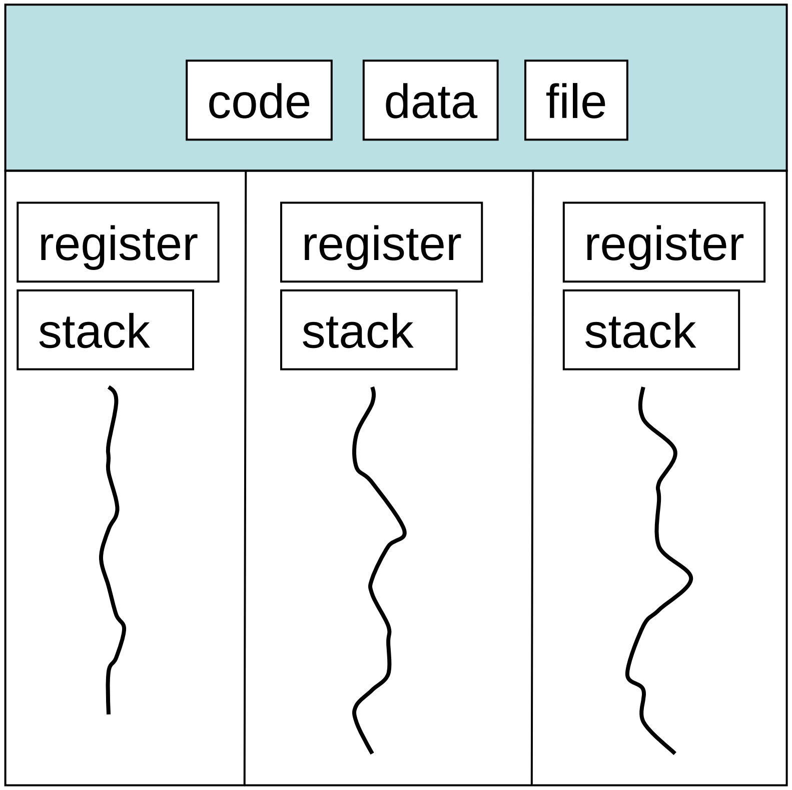
spawn process	NT	spawnl	12.0 ms
	Linux	fork	6.0 ms
spawn thread	NT	pthread_create	0.9 ms
	Linux	pthread_create	0.3 ms

- Threads in the same program share data.
- Context switch is faster since data are shared.
- Some languages (such as Java) provide built-in support for multiple threads.

Why is it called “Thread”?

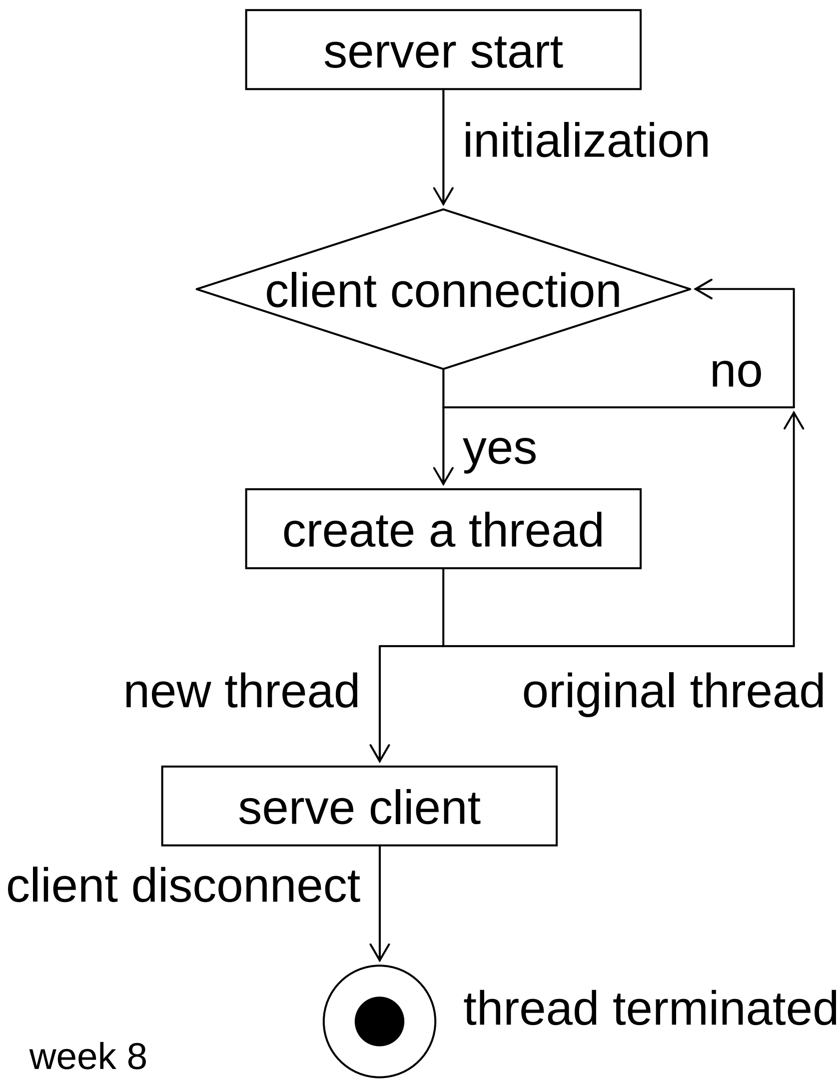


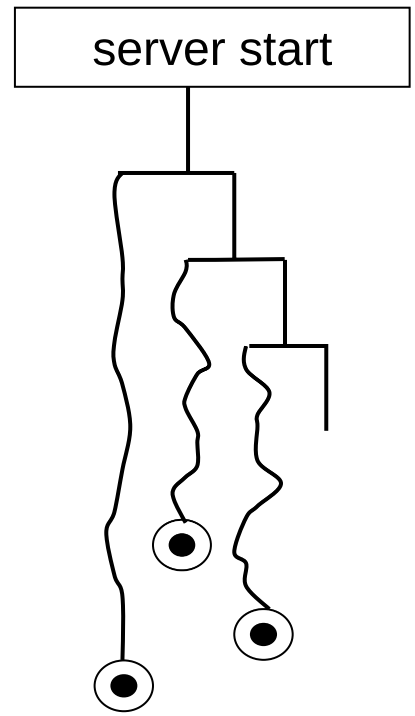
single thread



multiple threads

A Web Server with Multiple Threads





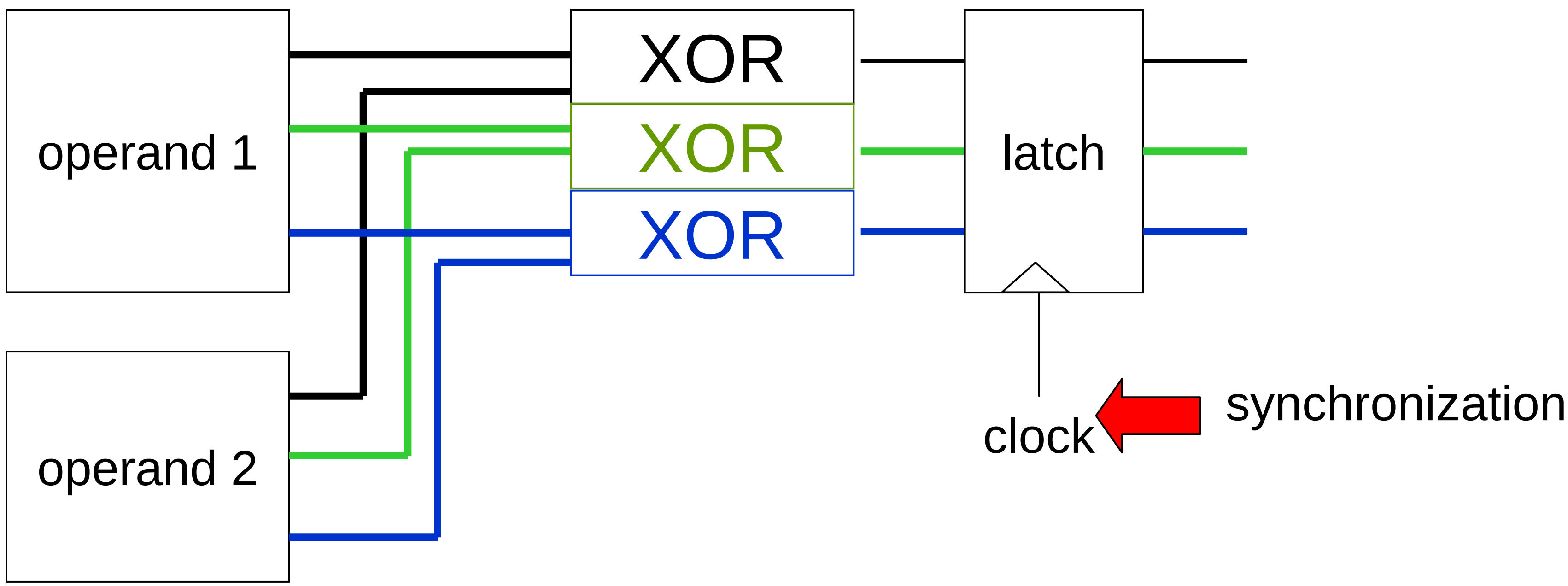
Different shapes of the threads indicate different operations, such as different files, different users, different transaction amounts …

Problems of Multi-Thread
(and Parallel Programming in General)

- synchronization
- ... synchronization
- ... ... synchronization
- too much synchronization $\Rightarrow$ serialize
- insufficient $\Rightarrow$ corrupt shared data
- often non-deterministic $\Rightarrow$ execute the same program multiple times and get different results

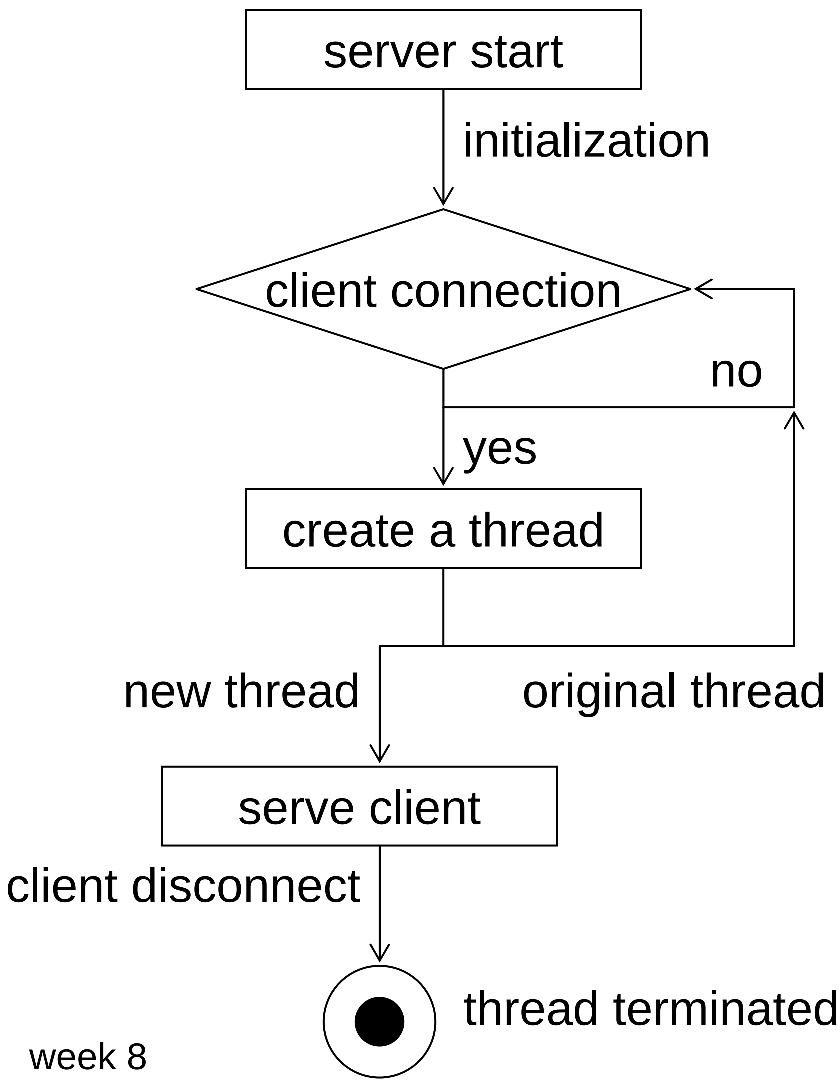
  $\Rightarrow$ very hard to verify correctness and debug

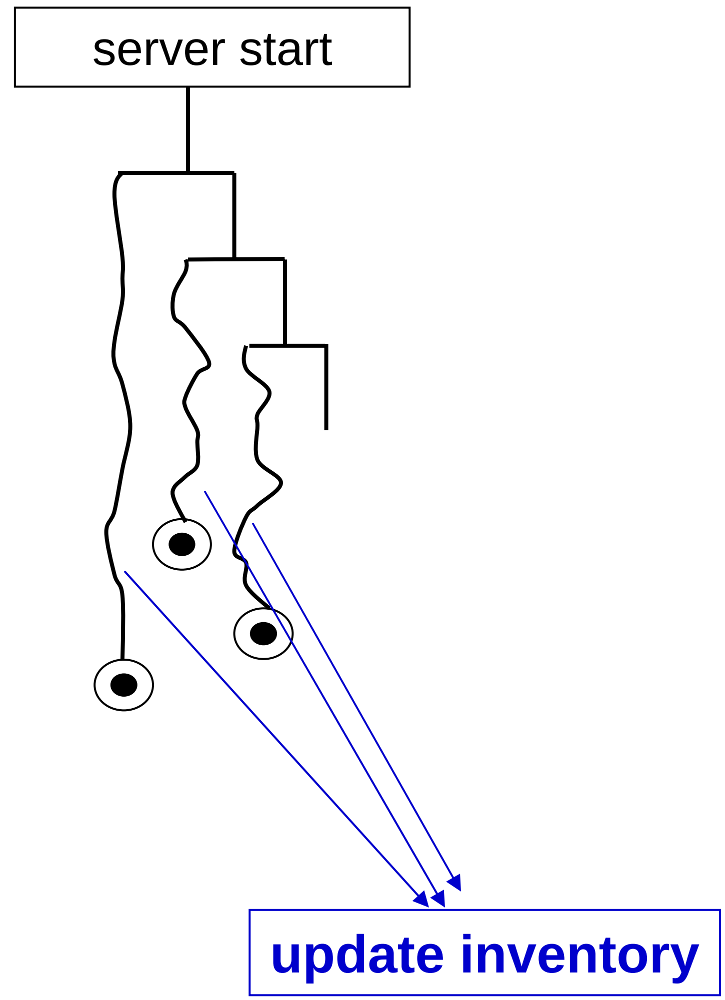
Hardware Parallelism (3-bit XOR)



(Adders need to propagate carry signals.)

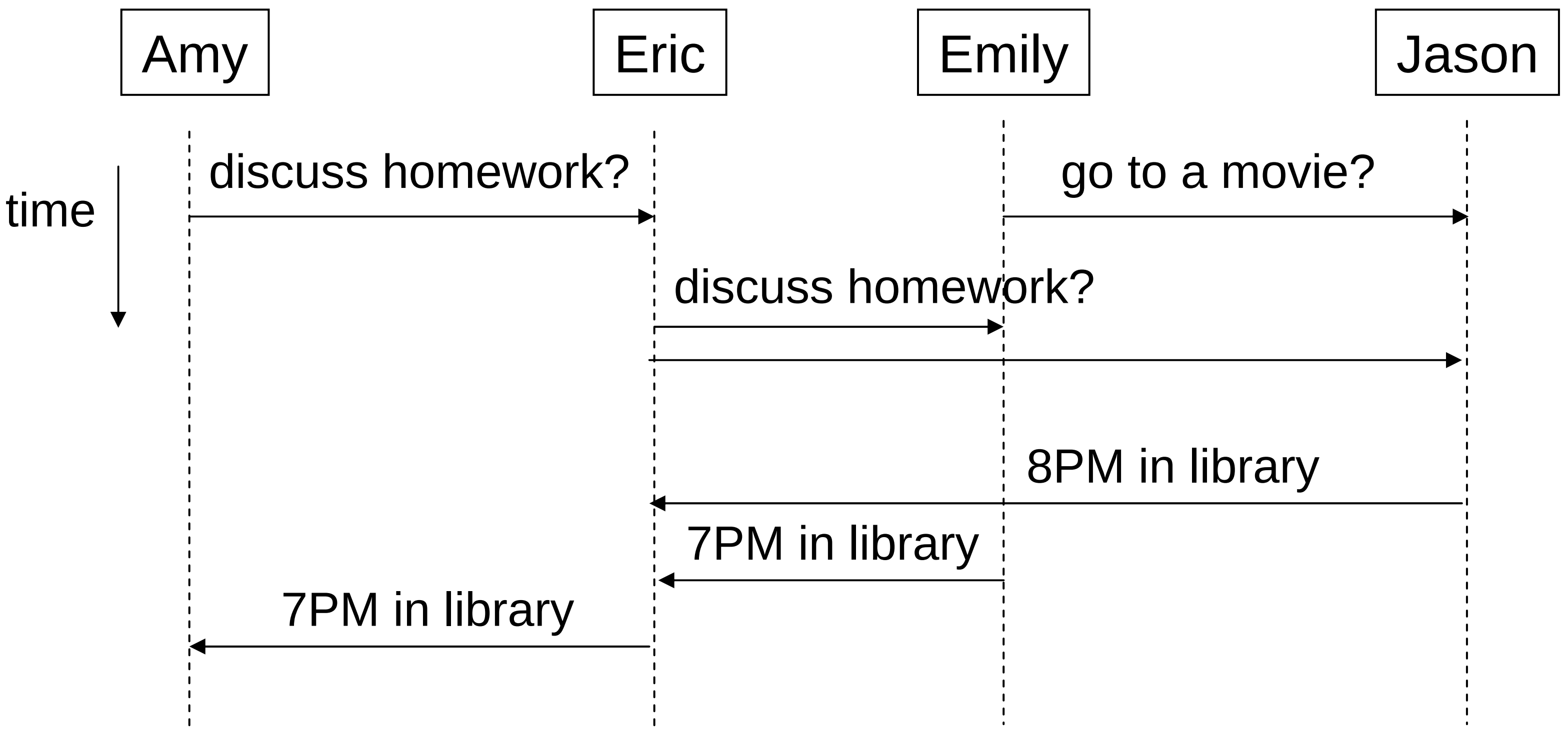
Thread Synchronization





Multithread OOP

Each object can be independent and they can interact simultaneously.



```java
//ThreadsBasic.java
class HelloThread extends Thread {
    String message;
    HelloThread( String message ) { this.message = message; }
    public void run() { System.out.print( message );   }
    public static void main( String[] args )
    {
        HelloThread ht1 = new HelloThread( "Good" );
        HelloThread ht2 = new HelloThread( " morning" );
        HelloThread ht3 = new HelloThread( " to" );
        ht1.start();
        ht2.start();
        ht3.start();
        System.out.println( " you!" );
    }
}
```

Structure of a Multi-Thread Program

```
class Cname extends Thread {
    ...
    public void run () { ... } // must provide the implementation
}
class ... {
    Cname cobj = new Cname(...);
    ...
    cobj.start();
}
```

// Once a thread starts, there is no guarantee which threads will run
// first. In a multiprocessor machine, threads may run in parallel.

```java
//ThreadsBasicWithJoin.java
class HelloThread extends Thread {
    String message;
    HelloThread( String message ) { this.message = message; }
    public void run() { System.out.print( message );   }
    public static void main(String[] args) throws InterruptedException
    {
        HelloThread ht1 = new HelloThread( "Good" );
        HelloThread ht2 = new HelloThread( " morning" );
        HelloThread ht3 = new HelloThread( " to" );
        ht1.start();
        ht2.start();
        ht3.start();
        ht1.join();
        ht2.join();
        ht3.join();
        System.out.println( " you!" );
    }
}
```