Problem Solving and Object-Oriented Programming

CS 18000
Sunil Prabhakar
Department of Computer Science
Purdue University

Objectives

In this module we will study:

- The difference between hardware and software
- Problem solving with computers
- Programming languages; Java
- Fundamentals of Object-Oriented Programming
    - classes and objects

This Course

- We will study how computers can be used to solve certain problems
  - Identify how to represent the problem so that we can use computers to solve them
  - Design a solution for the problem
  - Convert the solution to a program (in Java)
- We will learn several aspects that are common to most programming languages
  - and also several details specific to Java

The Art of Programming

- Computers are not inherently intelligent.
  - They have a very small number of simple operations available
  - They do not “understand” what they are doing -- they simply follow (like a mindless automaton) the instructions given to them
  - But, they are very fast, tireless, and perfectly obedient
- All the “magic” is in the program
  - How to represent real world concepts in the bits of a program?
  - How to use the simple instructions to achieve a high-level task such as playing chess?

Programming is ...

- Not unlike writing a symphony
  - But with perfect players to perform it!
- A highly creative exercise
  - How to create solutions to complex problems using a set of simple building blocks
- Can initially be painful
  - not unlike finger exercises — persevere!
- Highly rewarding and useful
  - Internet, Apps, Facebook, Amazon, EMR, space flight, climate modeling and prediction, simulations of phenomena, Hubble, Pacemakers, Computer games, telemedicine, Watson, ….
- Essential for many modern sciences

Working with computers

- Computers aren't smart, but they are *perfectly dumb* :
  - all errors are due to *your (mis)instructions!*

"I've sorted it out, the computer had put your National Insurance number in the tax due column."

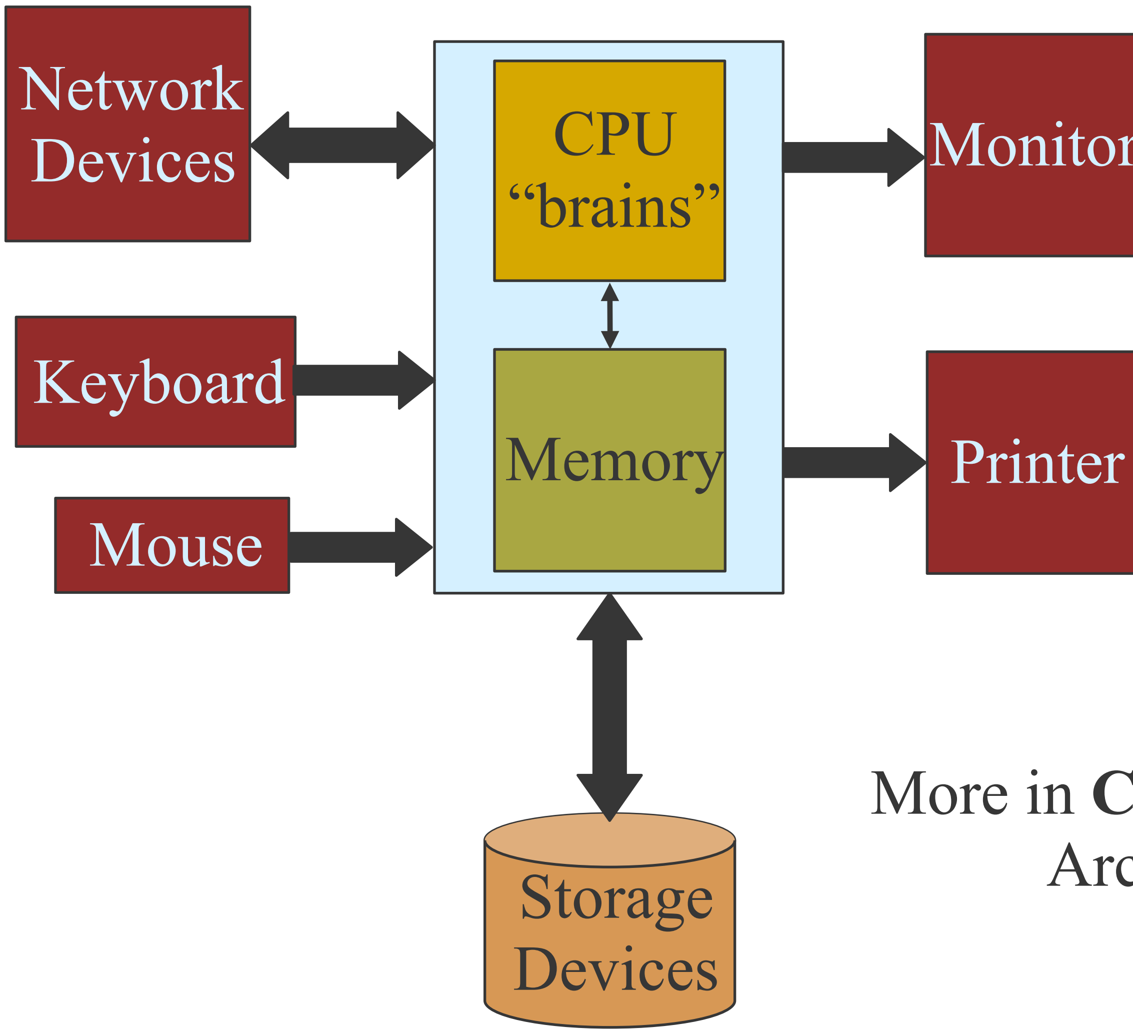
Programming Languages

- Programming languages provide a means to communicate our instructions to a simpler "mind" -- we need to learn to break complex tasks into simpler sub-tasks.
- We need to understand how to use only the operations available to achieve our goals
- We need to understand how simple bits can be used to represent complex concepts such as videos, images, web pages, gene expression data, particle collider outputs, global climate models, ...

Computer Systems

- There are two main components of a computer:
  - Hardware
    - The physical device including the IC chips, hard disks, displays, mice, etc.
    - Generally stuff that you can touch.
  - Software
    - The information stored on the computer
    - Includes programs and data
    - Stored in binary (0s and 1s)

Computer Architecture (simplified)



More in **CS250**: Computer Architecture.

Software

- The electronic components only store binary: 0s and 1s (voltage levels or magnetization)
- Two types of information
  - Instructions(programs) -- executed by the CPU
  - Data -- manipulated by CPU
- These are stored in memory
- The software provides a means to access and control the hardware
- This is done through a very important piece of software called the Operating System
- The OS is always running. More in **CS252** and **CS354**

Programs

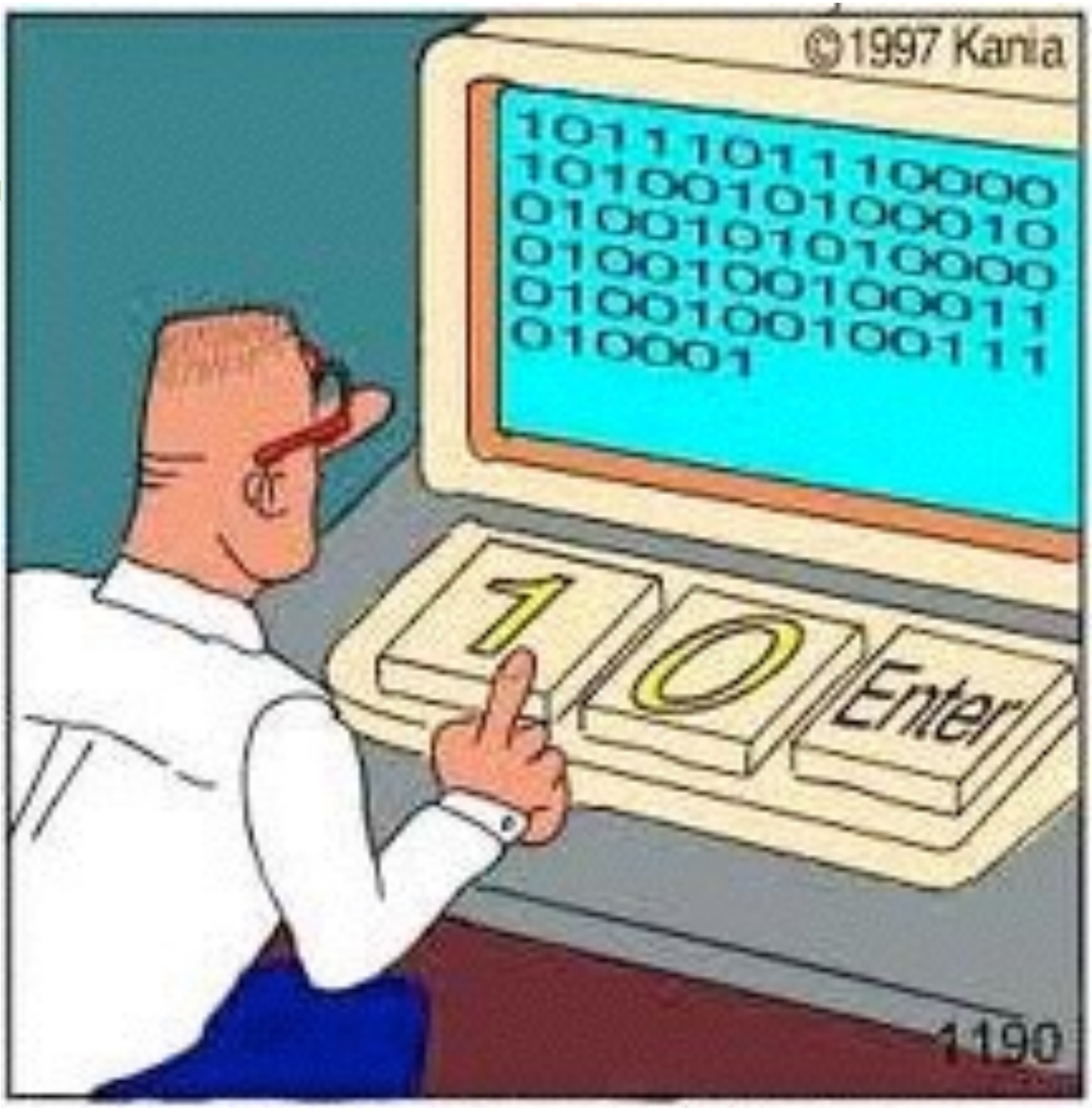
- A program is simply a set of instructions to the CPU to perform one of its operations
  - Arithmetic, Logic, Tests, Jumps, …
- A program typically takes input data, e.g.,
  - input keywords to a browser
  - mouse clicks as input to the operating system
- It also produces output, e.g.,
  - the display of search results in a browser
  - launching a program
- The program is stored in memory. It is read and executed by the CPU.

Machine Language

- A computer only runs programs that are specified in its own machine language (ML)
- For example, for the 8085 microprocessor:

  11000011 1000010100100000

  Instruction Address
- Also called binary or executable code.
- This instruction tells the CPU to pick its next instruction from memory location 133200.
- The ML is specific to the CPU, e.g. Pentium, 386, PowerPC G3, G4, …
- An executable program written for one CPU will not run on another CPU -- i.e. it is not portable.



Real programmers code in binary.

Not really!

Assembly language

- Machine language codes are not easy to remember
- Assembly language uses mnemonics and symbols to ease programming, e.g.,

  JMP L2

- A special program called an assembler must be used to convert the assembly code to machine code
- The assembly code is also hardware-specific.
- Eases programming but still requires one to think in terms of low-level steps taken by the CPU.
- Humans think at a higher level.

High-Level Languages (HLLs)

- Allow programmers to work with constructs that are closer to human language.
  - E.g. Java, C, C++, Basic, Fortran, COBOL, Lisp, …
- Need a special purpose program to convert the high-level program to machine language.
- This program is called a compiler.
- Can write programs in many different HLLs for the same CPU.
- Need a compiler for each language and CPU (OS).
- Efficient conversion is still an issue. More in **CS35200** Compilers
  - still use Machine Language for critical tasks

High-Level Languages (cont.)

- Since the language is not specific to the hardware, HLL programs are more portable
  - Some hardware, OS issues limit portability
- All we need is the program and a compiler for that language on the given hardware platform
  - E.g., a C compiler for Mac OS X
- Thus we can write a program once in a HLL and compile it to run on various platforms, e.g.,Firefox

Source Code

- A program written in a machine language is called an executable, or a binary.
  - It is (typically) not portable.
- A program written in a HLL is often called source code.
- Given an executable, it is difficult to recover the source code (not impossible).
- Thus, companies release only the executables.
- This makes it hard for someone else to replicate the software and also to modify it (maybe even to trust it completely)
- Open-Source is an alternative approach.

Algorithms

- Humans tend to think of programs at a higher level than HLL -- more in terms of algorithms.
- An algorithm is a well-defined, finite set of steps that solves a given problem
  - E.g., the rules for multiplying two numbers
- Algorithms are sometimes described using flow charts, or pseudo-code.
- This avoids the details of a particular HLL's syntax rules.

HLL Paradigms

- Procedural
  - A program is composed of packets of code called procedures, and variables. A procedure is at full liberty to operate on data that it can access. E.g., C, Pascal, COBOL, Fortran
- Object-Oriented
  - Programs are composed of Objects, each of a specific class with well defined methods. Data and programs are tightly coupled -- better design. E.g., Java, Objective-C, C#, (C++?)
- Functional
  - Programs are composed of functions. E.g., Lisp
- More in **CS45600** Programming Languages.

Java

- We will use Java as a representative language.
- Java is based upon C++ (which, in turn, is based on C).
- Unlike C++, which is really a hybrid language, Java is purely Object-Oriented.
  - This results in significant advantages.
- Most HLL programs are compiled to run on a single platform.
- Java programs can run on multiple platforms after compilation -- i.e., its compiled format is platform-independent.
- This design choice comes from its history.

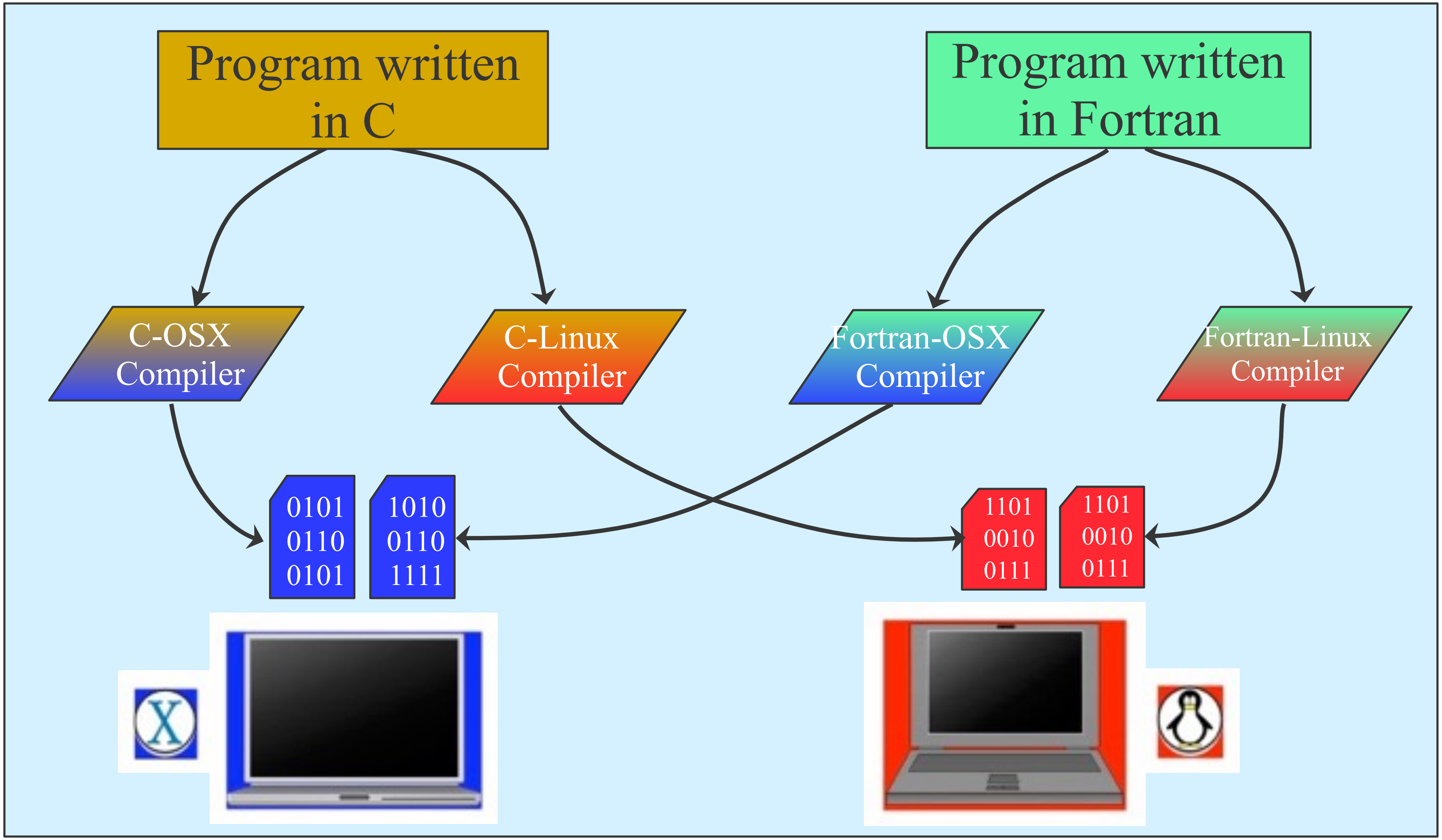
History of Java

- Java was developed by J. Gosling at Sun Microsystems in 1991 for programming home appliances (variety of hardware platforms).
- With the advent of the WWW (1994), Java's potential for making web pages more interesting and useful was recognized. Java began to be added to web pages (as applets) that could run on any computer (where the browser was running).
- Since then it has been more widely accepted and used as a general-purpose programming language, partly due to
  - its platform-independence, and
  - it is a truly OO language (unlike C++)
- Now belongs to Oracle following the purchase of Sun Microsystems.

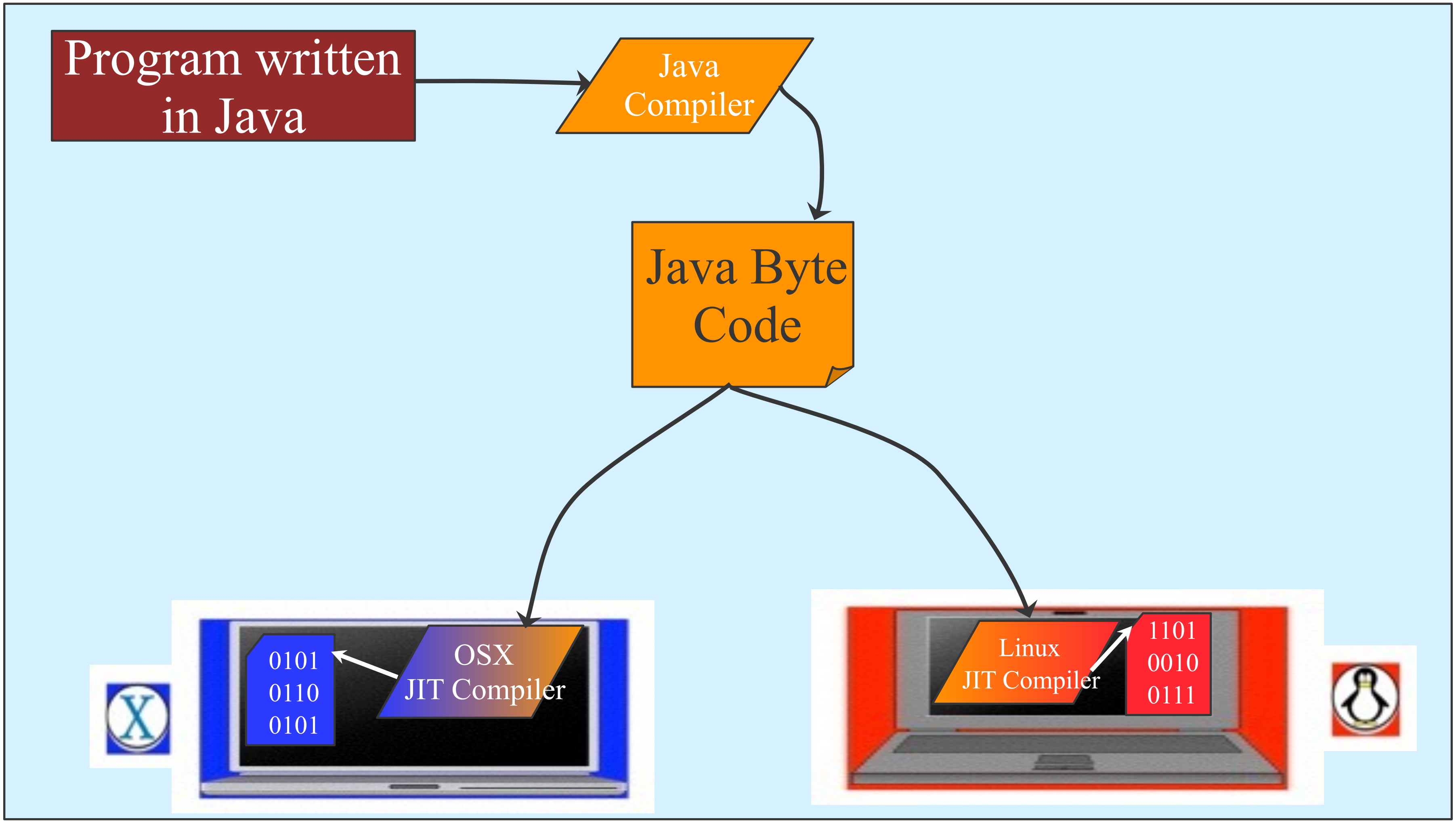
Platform-Independence

- Notion of a “Java Virtual Machine” (JVM)
- Java programs are compiled to run on a virtual machine (just a specification of a machine). This code is called Byte Code
- Each physical machine that runs a Java program (byte code) must “pretend” to be a JVM.
- This is achieved by running a program on the machine that implements the JVM and interprets the byte code to the appropriate machine code.
- This interpreting is done at run-time which can cause a slow down!

Regular Programming Languages



Java



Program written in Java
Java Compiler
Java Byte Code
0101
0110
0101
OSX JIT Compiler
Linux JIT Compiler
1101
0010
0111

Data

- All data are eventually stored in binary.
- In a HLL we treat data as having a type, e.g., integer, character, etc.
- Within a program every piece of data is stored at some location (address) in memory.
- We use identifiers to refer to these locations or to the data itself.
- Program instructions manipulate the various pieces of data accessible to the program.
- An object is a collection of some data along with pieces of code that can manipulate that data.

Key Features of OOP

- Classes and Objects
- Encapsulation
- Inheritance
- Polymorphism (later)
- Dynamic Binding (later)

Classes and Objects

- Object-oriented programs use objects.
- An *object* represents a concept that is part of our problem, such as an Account, Vehicle, or Employee
- Similar objects share characteristics and behavior. We first define these common characteristics for a group of similar objects as a *class*.
- A class defines the type of data that is associated with each object of the class, and also the behavior of each object (methods).
- A class is a template for the objects.
- An object is called an *instance (object)* of a class.
- Most programs will have multiple classes and objects.

Banking Example

- In a banking application, there may be numerous accounts.
- There is common behavior (as far as the bank is concerned) for all these accounts
  - Deposit, Check balance, Withdraw, Overdraw? ...
- There are also common types of data of interest
  - Account holder's name(s), SSN, Current balance, …
- Instead of defining these data and behavior for each account separately, we simply define them once -- this is the notion of the Account class.
- Each account will be an instance of this class and will have its own values for the data items, but the same behavior (defined once).

Encapsulation

- To prevent uncontrolled access to read and modify the data of an object, OOP languages restrict what operations (methods) can be applied to each object.
- This restriction of behavior is also called encapsulation -- an important part of OOP.
  - The data and behavior are encapsulated.
- This greatly improves reliability and manageability of code.
- Procedural languages do not have such restrictions -- the onus is on the programmer (less reliable).

Inheritance

- Many applications have concepts that are similar, but not identical
  - e.g., Savings Account, Checking Account
- some data and behavior are shared
  - e.g., name, address, balance, deposit
- Some data and behavior are not shared
  - e.g., interest rate, minimum balance
- Inheritance allows shared data and behavior to be defined once and shared among different classes
  - greater code reliability, easier maintenance