

# From Algorithms to Architecture

*...a lightning introduction to computer architecture*

# Implementing Algorithms

- Now have a methodology for going from problem to program
- Next develop a mental model of a device that might actually execute our algorithm, i.e. a computer!

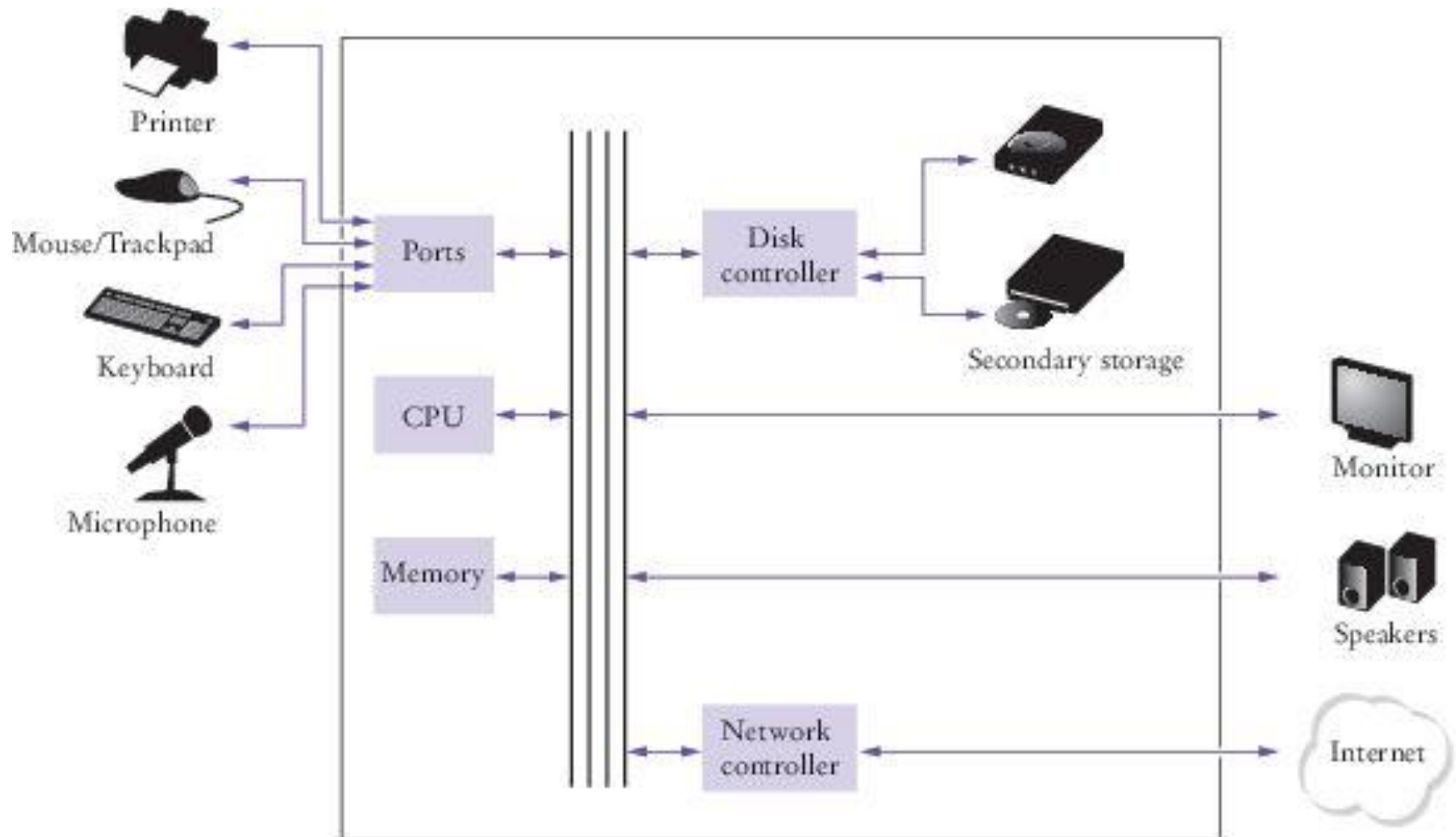
# Computer Programs

- **Computers** are programmed to perform many different tasks.
- Computers execute very basic instructions in rapid succession.
- A **computer program** is a sequence of instructions and decisions.
- **Programming** is the act of designing and implementing computer programs.
- The physical computer and peripheral devices are collectively called the **hardware**.
- The programs the computer executes are called the **software**.

# The Anatomy of a Computer

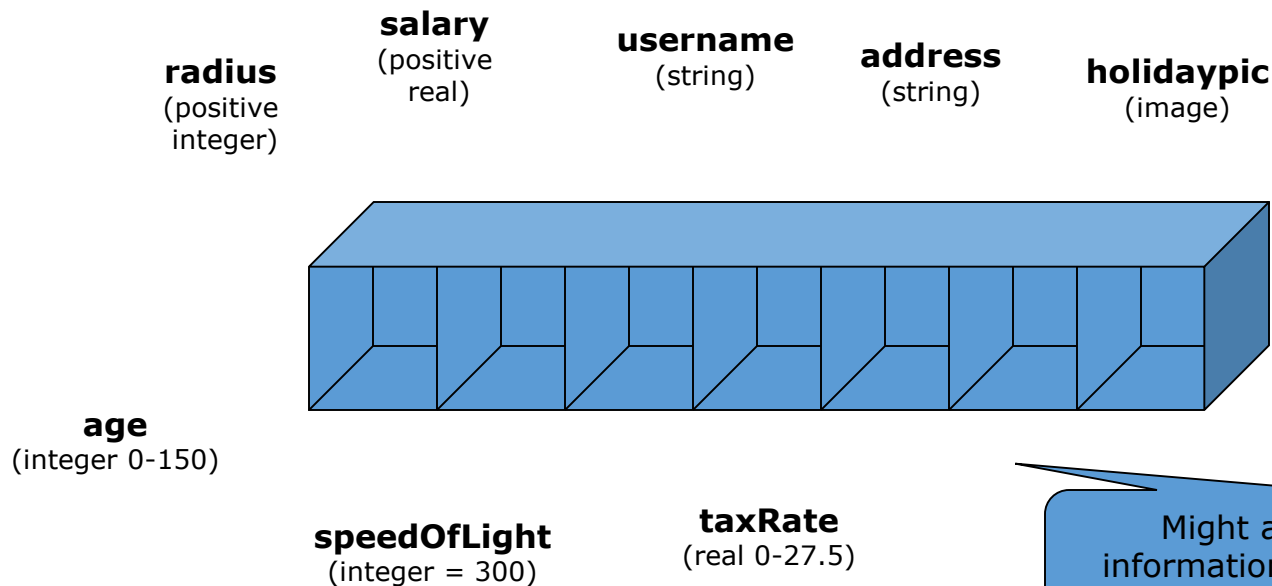
- Central processing unit (CPU) performs
  - Program control
  - Data processing
- Storage
  - Memory (Primary storage)
  - Secondary storage
- Peripherals
  - To interact with human users
- Networks

# Schematic Diagram of a Computer



# Computer Memory

- Set of memory locations
  - To identify easily, number or name them
  - To minimise bugs, restrict type of content

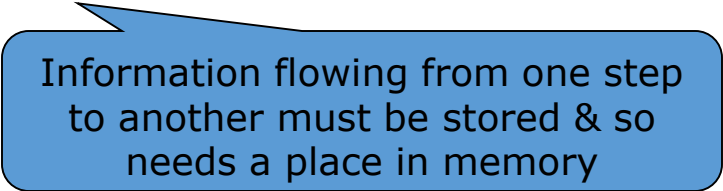


Might also specify whether information can be changed or not i.e. is constant/variable

# Area-Circumference Problem

## To find area & circumference of circle...

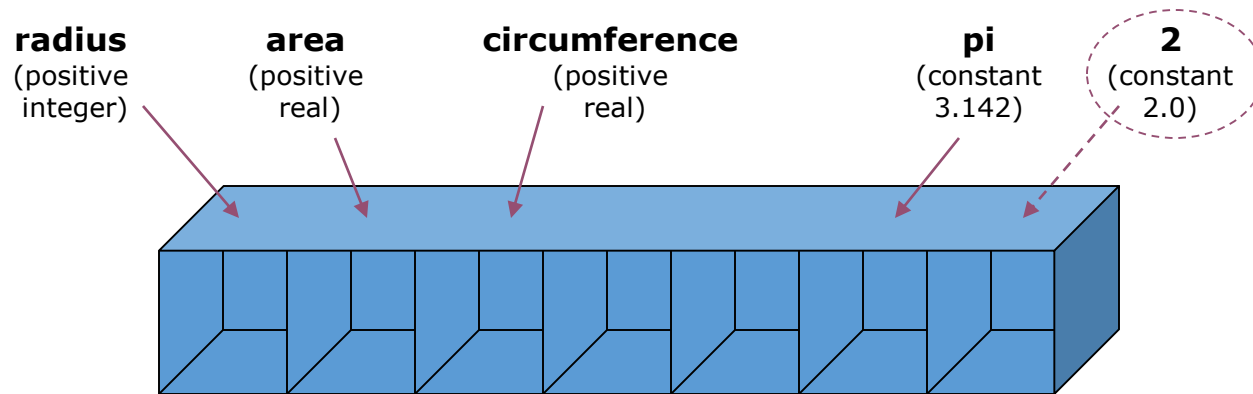
1. Print welcome message
2. Ask for & get radius from user
3. Compute area as  $\pi \cdot \text{radius} \cdot \text{radius}$
4. Compute circumference as  $2 \cdot \pi \cdot \text{radius}$
5. Report area, circumference & radius



Information flowing from one step to another must be stored & so needs a place in memory

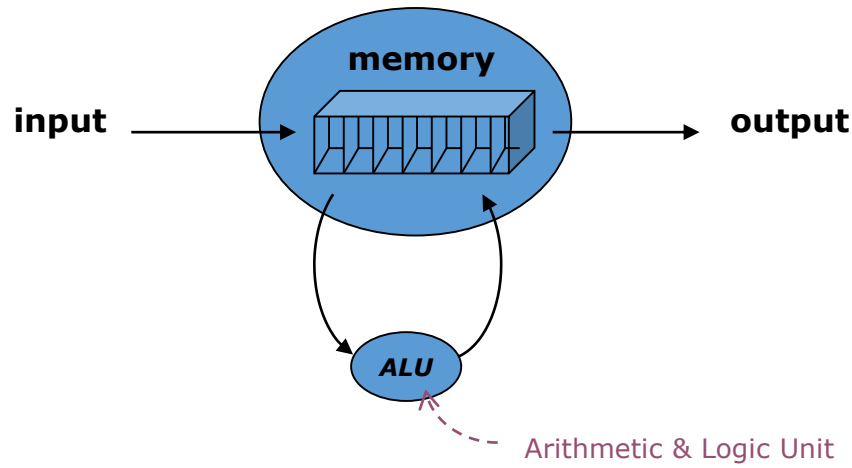
# Algorithm

- Envisage area/circumference algorithm in terms of computer memory model



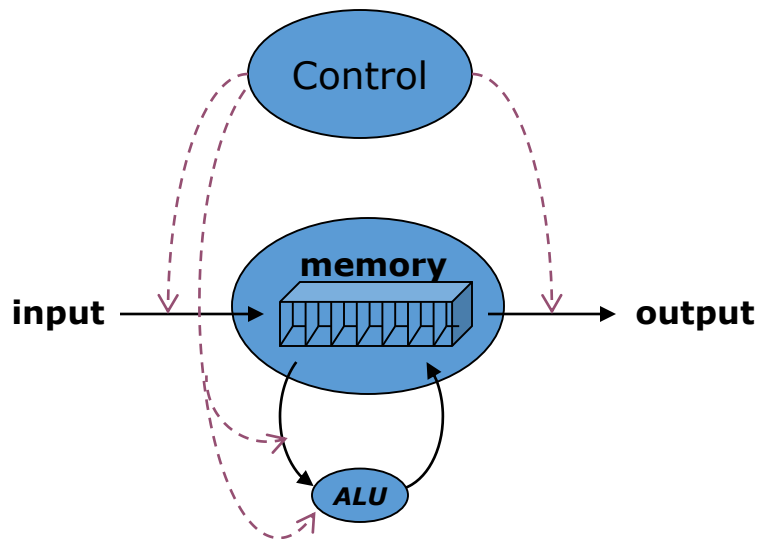
# Data flow

- Only three sorts of instructions
  - **input** - from external world to memory
  - **output** - from memory to world
  - **assignment** - from memory to memory



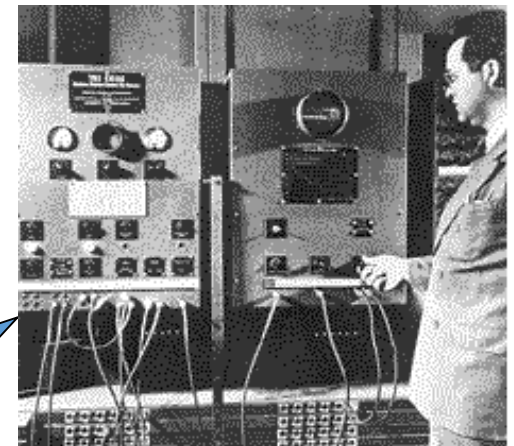
# Control flow (1)

- Control mechanism implements algorithm
  - sets up the data paths in appropriate sequence

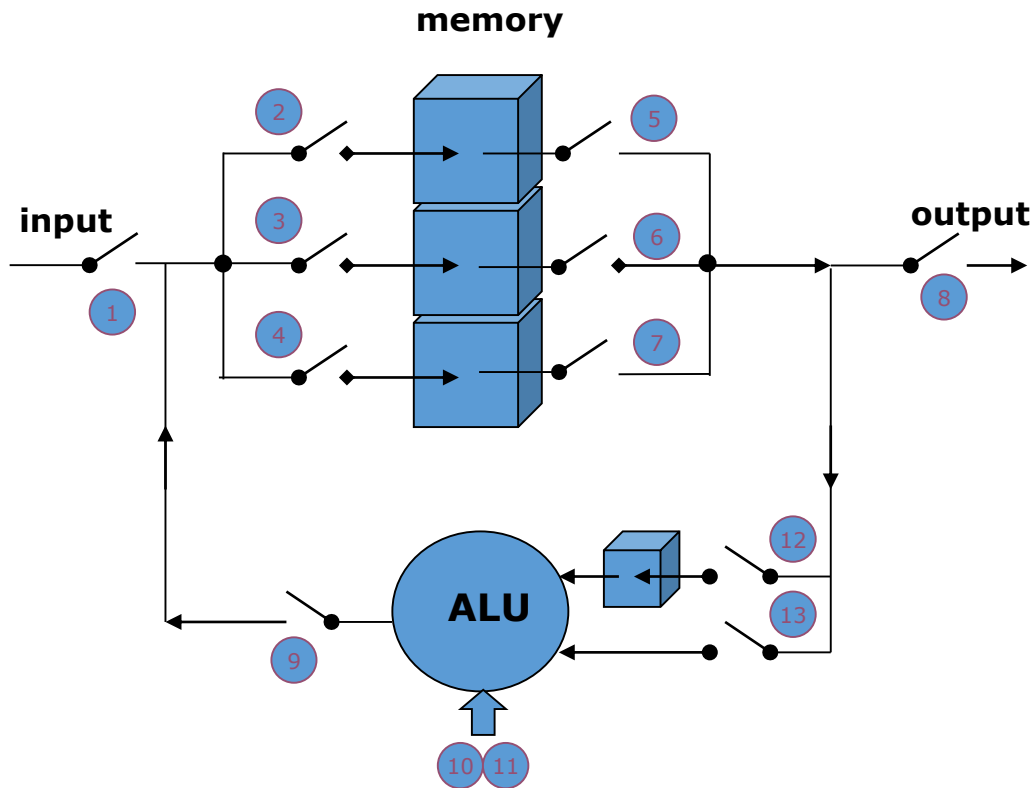


First computing devices had control mechanisms that were hardwired (fixed) or at best “pluggable” e.g ENIAC

Changing the function of the machine required literally changing its wiring!



# How it works...

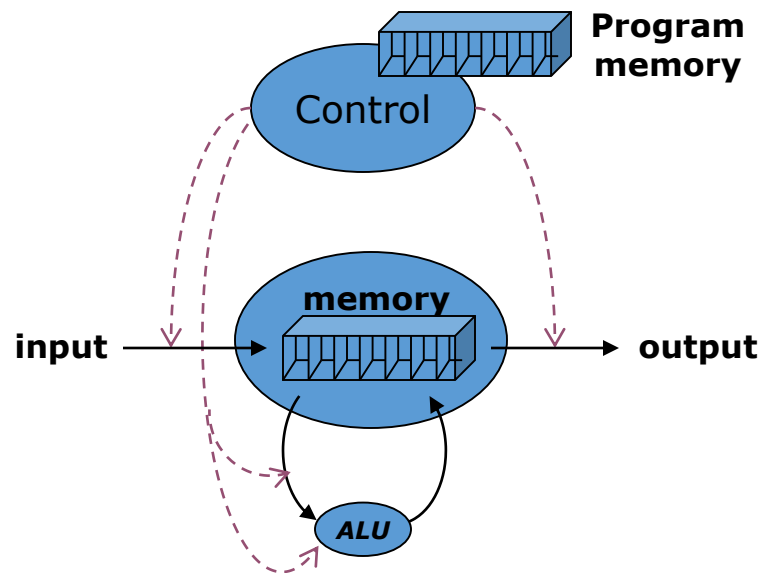


- Switches control data flow into and out of memory.
- Sequence of switch settings determines function

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0  | 0  | 1  | 0  |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0  | 1  | 0  | 1  |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0  | 0  | 0  | 0  |

# Control flow (2)

- Recognising
  - only three forms of control sequence required (*sequence, decision & repetition*)
  - & instructions can be encoded in memory like data
- allows general control mechanism to be implemented

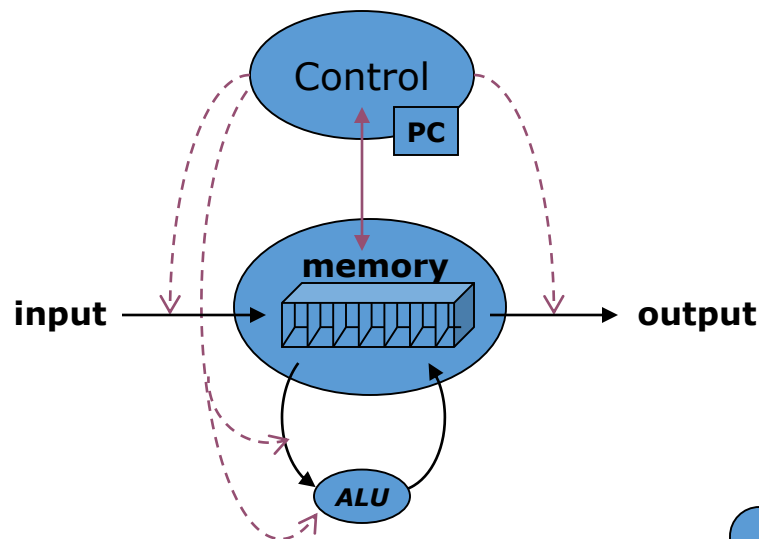


Instructions & hence the machine's function can be changed quickly & easily.

Limitation: may be out of program memory, yet have free data memory or vice versa!

# Control flow (3)

- Finally
  - realise that program & data can share same memory



Memory now holds both program & data

Fetch  
Execute  
cycle

Instructions stored in sequentially numbered locations in memory.

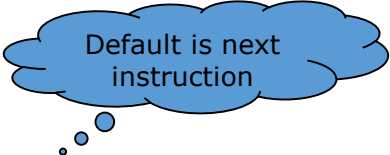
Current position in program held in program counter.

Control fetches current instruction, decodes & executes it (by setting data paths), increments PC, then fetches next instruction, and so on.

# Spaghetti Code?

- “go to” programming...

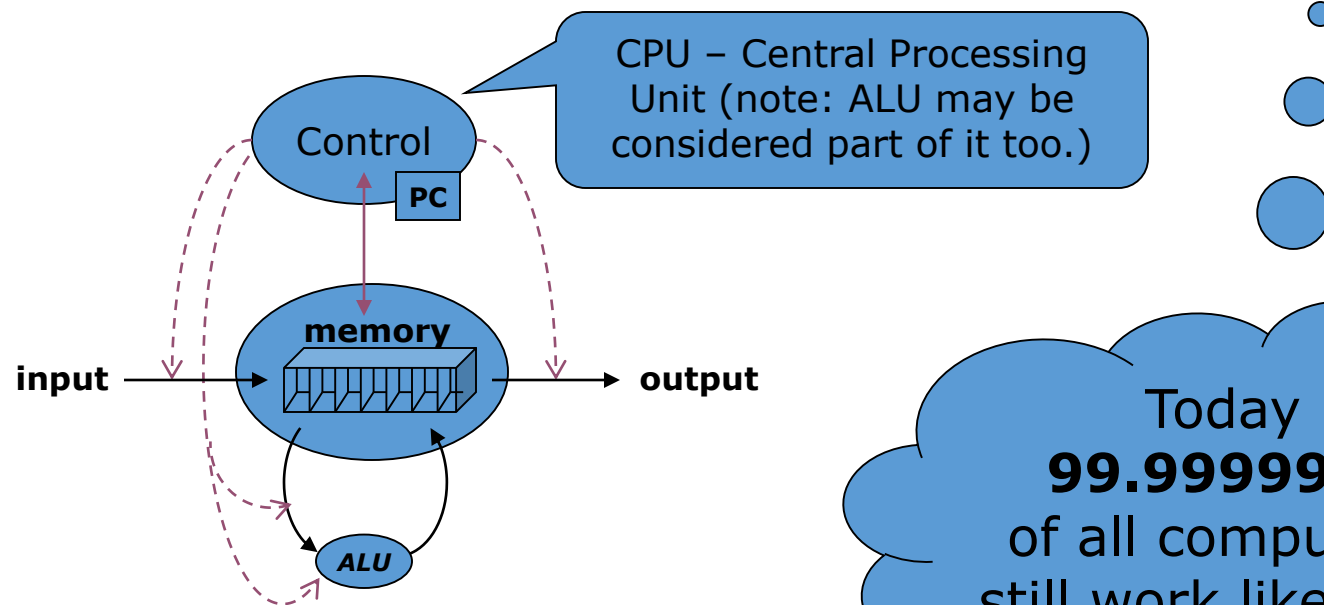
1. Read first number
2. If first number > 10 then go to 4
3. go to 1
4. Read second number
5. If second number <= first number goto 8
6. Print “Well done”
7. go to 10
8. Print “sorry, no good!”
9. go to 4
10. Print “finished.”



Default is next instruction

# Von Neumann Architecture

- Stored-program computer architecture
- Credited to John von Neumann, circa 1946

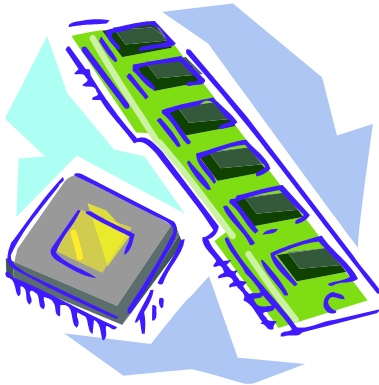


Today  
**99.99999%**  
of all computers  
still work like this!

# Practical Considerations

- Memory crucial to system speed!
- Memory technology

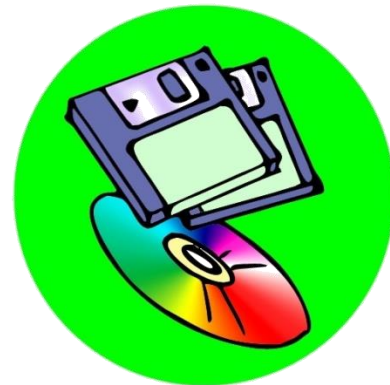
High-speed, volatile,  
expensive, so limited.



Semiconductor  
RAM, ROM

Speed  
differential  
>10,000x

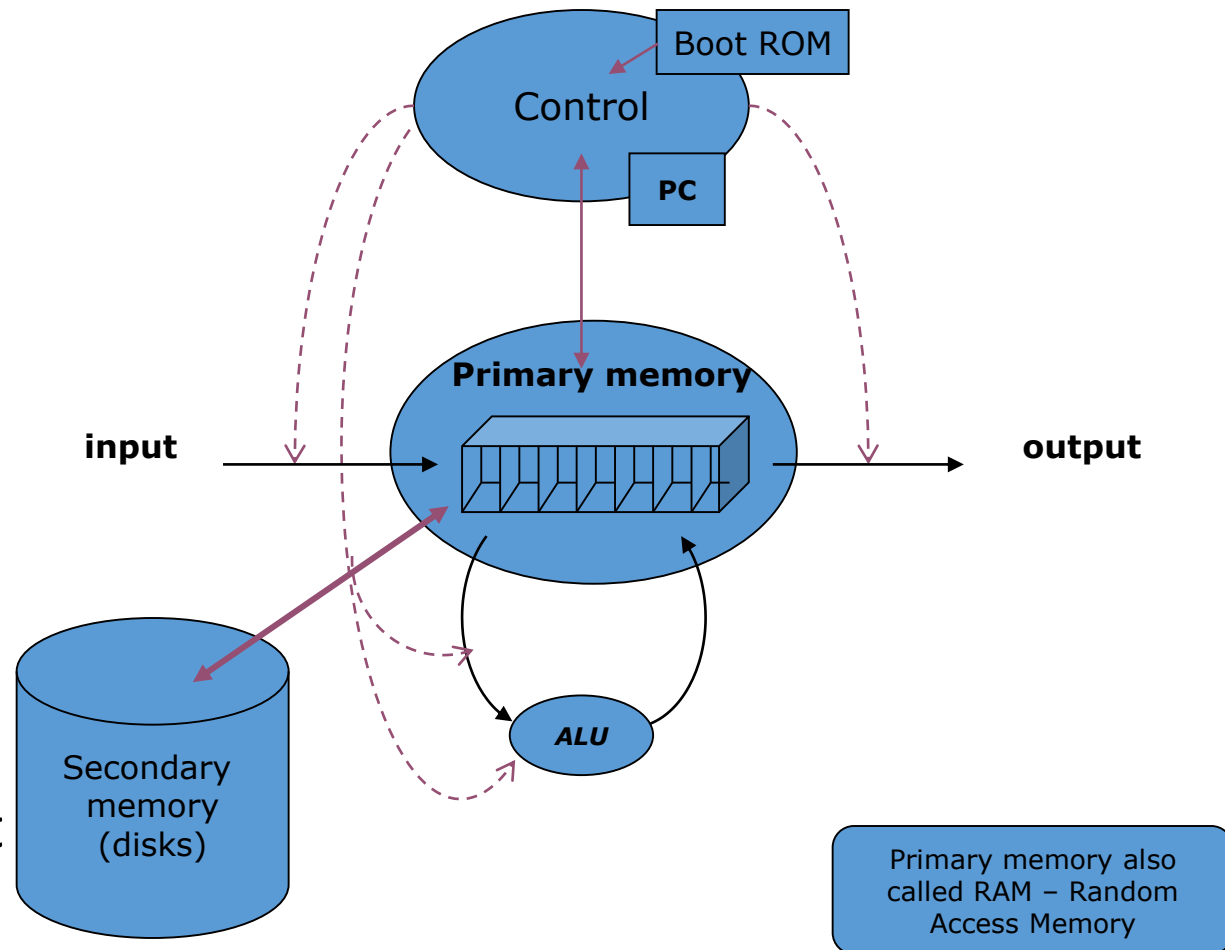
Non-volatile, cheap, so  
plentiful, but slow!



Optical & Magnetic  
Disks, tapes & CDROMs

# Memory Hierarchy

- Result of today's technological
- Long-term data & programs stored in secondary storage
- Moved into primary memory (RAM) when needed.
- When machine switched on, no program in memory
- Boot ROM load a program (usually the OS) into RAM and start it running.

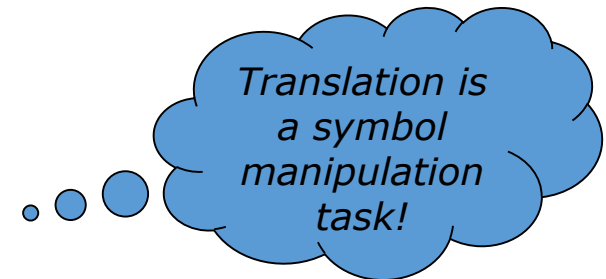


# From problem to execution

- From requirements to algorithm
- From algorithm to program  
(something “understandable” to machine)
- Ultimately need machine code  
(1’s & 0’s representing switch patterns)

*but how do we get it...?*

- *Directly write machine code*
- *Write assembly & translate*
- *Write high-level & translate*



# Language Hierarchy (1)

- Machine code
  - Patterns of 1's & 0's ~ machine instruction
  - Usually restricted, e.g. no real numbers
  - Difficult & error-prone writing by hand!

```
10011000
11000011
01000000
01000110
11111000
10000110
00000011
11111100
:
```

Note: machine dependent, same operation may require different pattern of 1's & 0's on different machines. Computer designer/manufacture defines machine code.

# Language Hierarchy (2)

- Assembly language
  - Each machine instruction has mnemonic
  - Easier to remember & understand  
so slightly less error-prone,  
but still difficult to do even simple things!
  - One-to-one mapping  
so translation to machine code is trivial
  - Use program to do translation (assembler)

```
MOV #5, R1  
ADD R1, R2  
STR @R0  
:
```

Like machine code, assembly  
language is machine dependent  
and low-level

# Language Hierarchy (3)

- High-level language
  - Much more “natural”, e.g. has real numbers!
  - Much easier to understand & write
  - Language standards, so machine independent
  - Translation to machine code is complex  
use program to do translation!

```
input A
Z = A * 3 + 1;
print( "Z=" . Z);
:
```

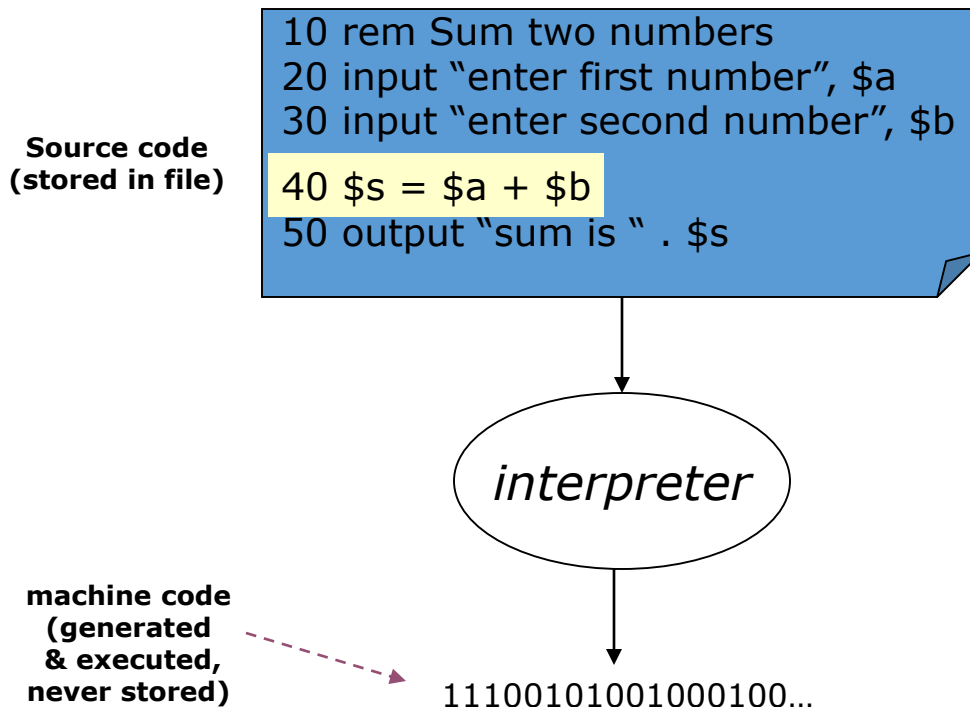
## Two approaches

- Interpret
- Compile

This program must  
itself be able to be  
executed on the  
machine being  
used!

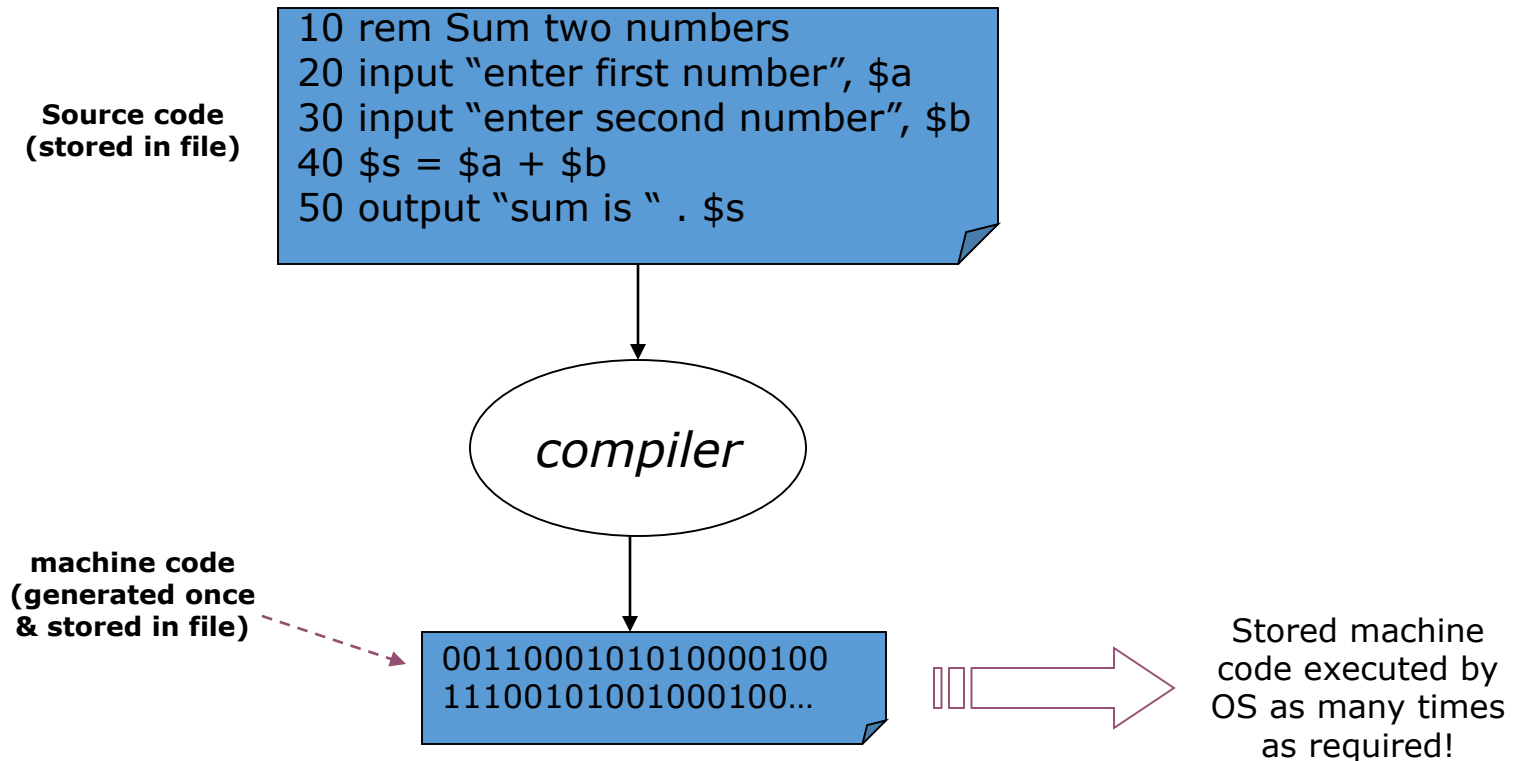
# Interpreters

- *Translate & immediately execute each instruction in turn*



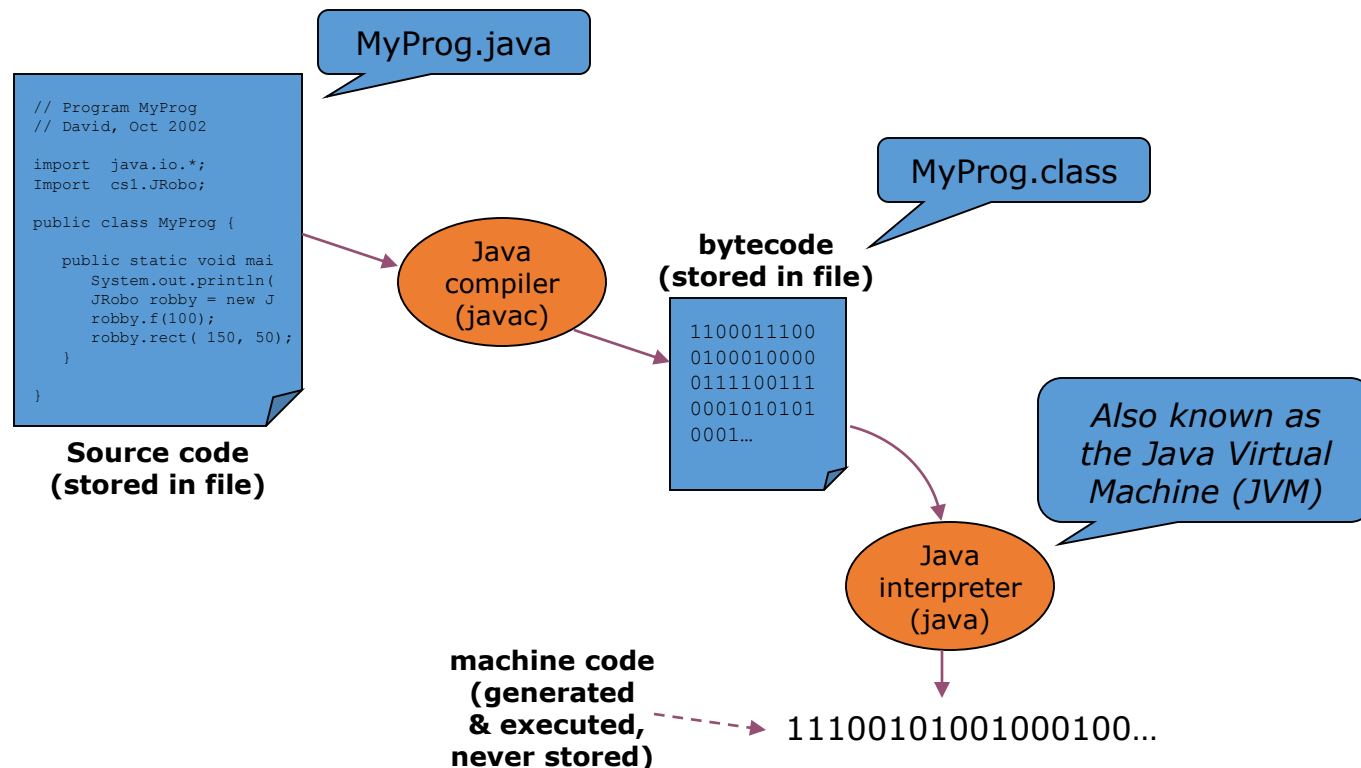
# Compilers

- *Translate all instructions to machine code, save & execute as needed*

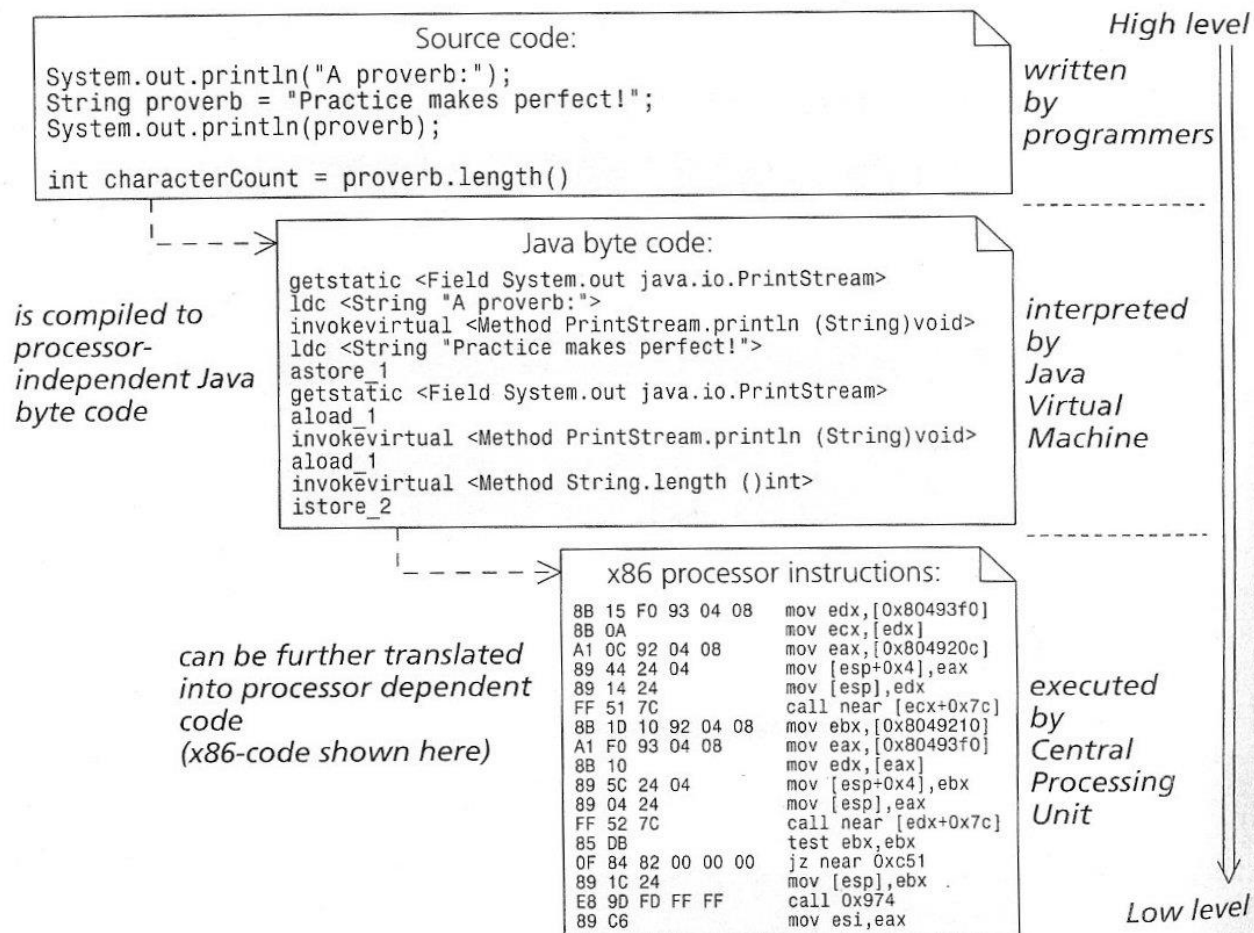


# Java – compile & interpret

- Compile & save to bytecode (*machine independent*)
- Execute by interpreting bytecode

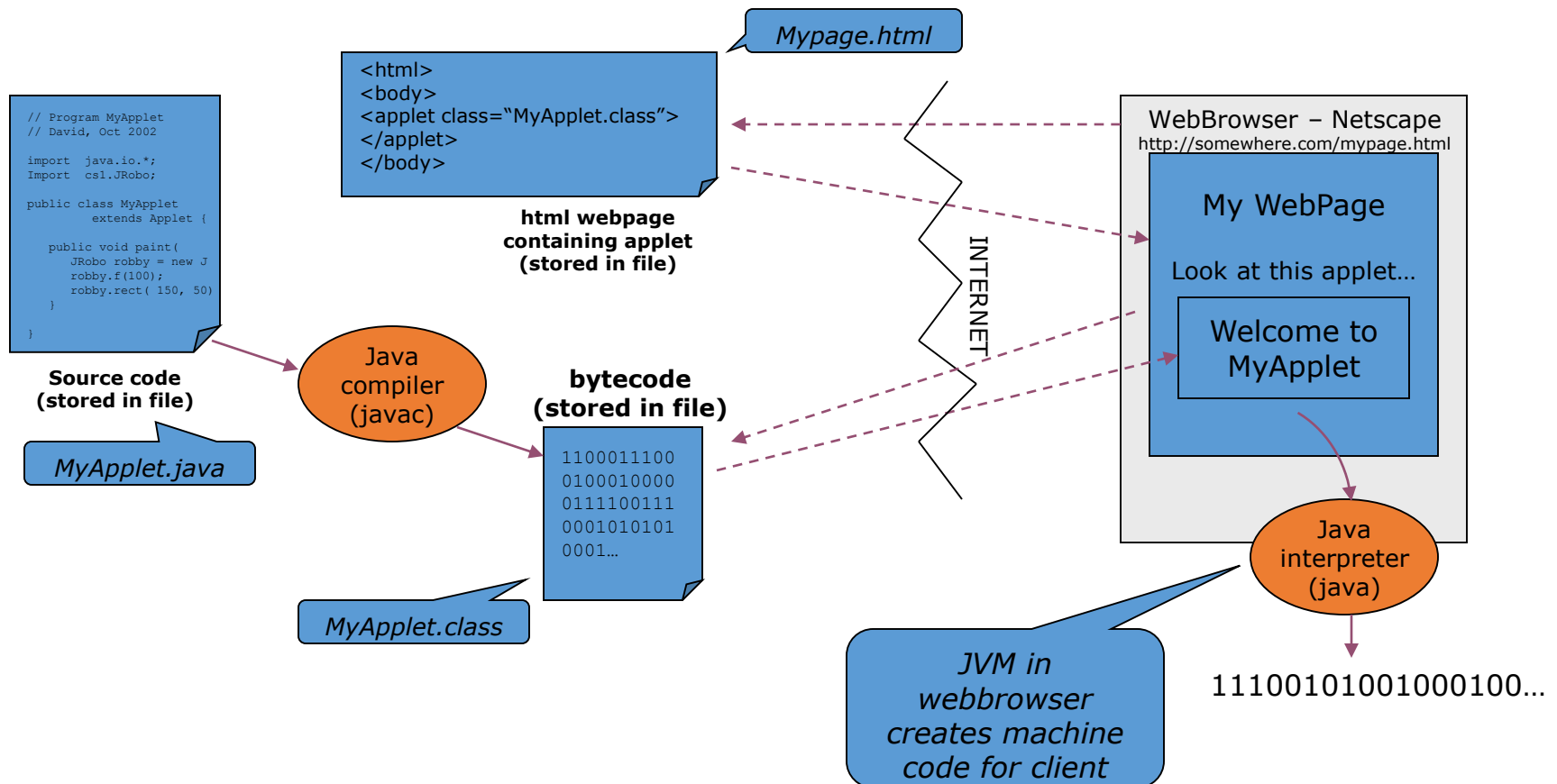


# Java language levels...



# Programs across Internet

- Java Applets – run anywhere safely!



# The Java Programming Language

- Safe
- Portable
- Platform-independent
  - Distributed as instructions for a virtual machine
- Vast set of library packages
- Designed for the Internet

# HelloPrinter.java

```
1  public class HelloPrinter
2  {
3      public static void main(String[] args)
4      {
5          // Display a greeting in the console window
6
7          System.out.println("Hello, World!");
8      }
9  }
```

# Analyzing Your First Program: Class Declaration

- Classes are the fundamental building blocks of Java programs:
- Declaration of a class called `HelloPrinter`  
`public class HelloPrinter`
- The name of the public class must match the name of the file containing the class:
  - Class `HelloPrinter` must be contained in a file named `HelloPrinter.java`

# Analyzing Your First Program: Methods

- Each class contains declarations of methods.
- Each method contains a sequence of instructions.
- A method contains a collection of programming instructions that describe how to carry out a particular task.
- A method is called by specifying the method and its arguments.

# Analyzing Your First Program: `main` Method

- Every Java application contains a class with a `main` method
  - When the application starts, the instructions in the `main` method are executed

- Declaring a `main` method

```
public static void main(String[] args)
{
    . . .
}
```

# Analyzing Your First Program: Statements

- The *body* of the main method contains **statements**.
- Our method has a single statement:  
`System.out.println("Hello, World!");`
- It prints a line of text:  
`Hello, World!`

# Analyzing Your First Program: Method Call

- A method call:

```
System.out.println("Hello, World!");
```

- A method call requires:

1. The method you want to use (in this case, `System.out.println`)
2. Any values the method needs to carry out its task enclosed in parentheses (in this case, `"Hello, World!"`)

- The technical term for such values is arguments

# Analyzing Your First Program: Printing

- You can print numerical values

```
System.out.println(3 + 4);
```

- evaluates the expression  $3 + 4$
- displays the number 7.

- `System.out.println` method prints a string or a number and then starts a new line.

- The sequence of statements

```
System.out.println("Hello");  
System.out.println("World!");
```

- Prints two lines

```
Hello  
World!
```

- There is a second method, `System.out.print`, that you can use to print an item without starting a new line

# Self Check

How would you modify the `HelloPrinter` program to print the word `"Hello"` vertically?

**Answer:**

```
System.out.println("H");  
System.out.println("e");  
System.out.println("l");  
System.out.println("l");  
System.out.println("o");
```

# Self Check

Would the program continue to work if you replaced line 7 with this statement?

```
System.out.println(Hello);
```

**Answer:** No. The compiler would look for an item whose name is `Hello`. You need to enclose `Hello` in quotation marks:

```
System.out.println("Hello");
```

# Self Check

What does the following set of statements print?

```
System.out.print("My lucky number is");  
System.out.println(3 + 4 + 5);
```

**Answer:** The printout is `My lucky number is12`. It would be a good idea to add a space after the `is`.

# Self Check

What do the following statements print?

```
System.out.println("Hello");
```

```
System.out.println("");
```

```
System.out.println("World");
```

**Answer:**

Hello

*a blank line*

World

# Errors

- A compile-time error (syntax error)
  - is a violation of the programming language rules
  - detected by the compiler.

```
System.ou.println("Hello, World!");
```

- A run-time error (logic error)
  - causes a program to perform an action that the programmer did not intend.

```
System.out.println("Hello, Word!");
```

# Errors

- Exception - a type of run-time error
  - Generates an error message from the Java virtual machine
  - This statement  
`System.out.println(1 / 0)`
  - Generates this run-time error message  
`"Division by zero"`

# Self Check

Suppose you omit the `""` characters around `Hello, World!` from the `HelloPrinter.java` program. Is this a compile-time error or a run-time error?

**Answer:** This is a compile-time error. The compiler will complain that it does not know the meanings of the words `Hello` and `World`.

# Self Check

Suppose you change `println` to `println` in the `HelloPrinter.java` program. Is this a compile-time error or a run-time error?

**Answer:** This is a compile-time error. The compiler will complain that `System.out` does not have a method called `println`.

# Self Check

Suppose you change `main` to `hello` in the `HelloPrinter.java` program. Is this a compile-time error or a run-time error?

**Answer:** This is a run-time error. It is perfectly legal to give the name `hello` to a method, so the compiler won't complain. But when the program is run, the virtual machine will look for a `main` method and won't find one.

# Self Check

When you used your computer, you may have experienced a program that "crashed" (quit spontaneously) or "hung" (failed to respond to your input). Is that behavior a compile-time error or a run-time error?

**Answer:** It is a run-time error. After all, the program had been compiled in order for you to run it.

# Self Check

Why can't you test a program for run-time errors when it has compiler errors?

**Answer:** When a program has compiler errors, no class file is produced, and there is nothing to run.