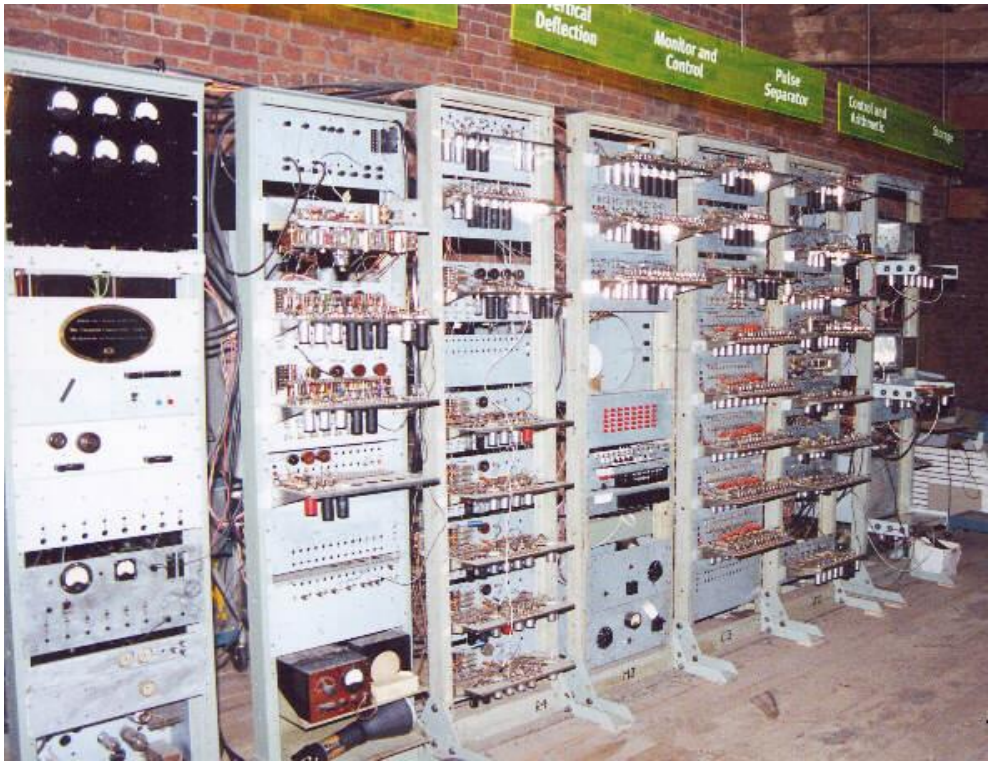
What is a computer?

Early "computers" were actually humans!

The term ‘computing machine’ was used increasingly from the 1920s for machines that did the work of a human computer

What did early computers look like?

One of the earliest stored-program computer: Manchester 'Baby' (first ran on 21 June 1948)



A reconstruction of the world's first computer

Monash's First Computer: The Ferranti Sirius, 1962

A modern Supercomputer

Fugaku Supercomputer

World's Fastest SuperComputer

(2021)

What is a computer?

Computer systems

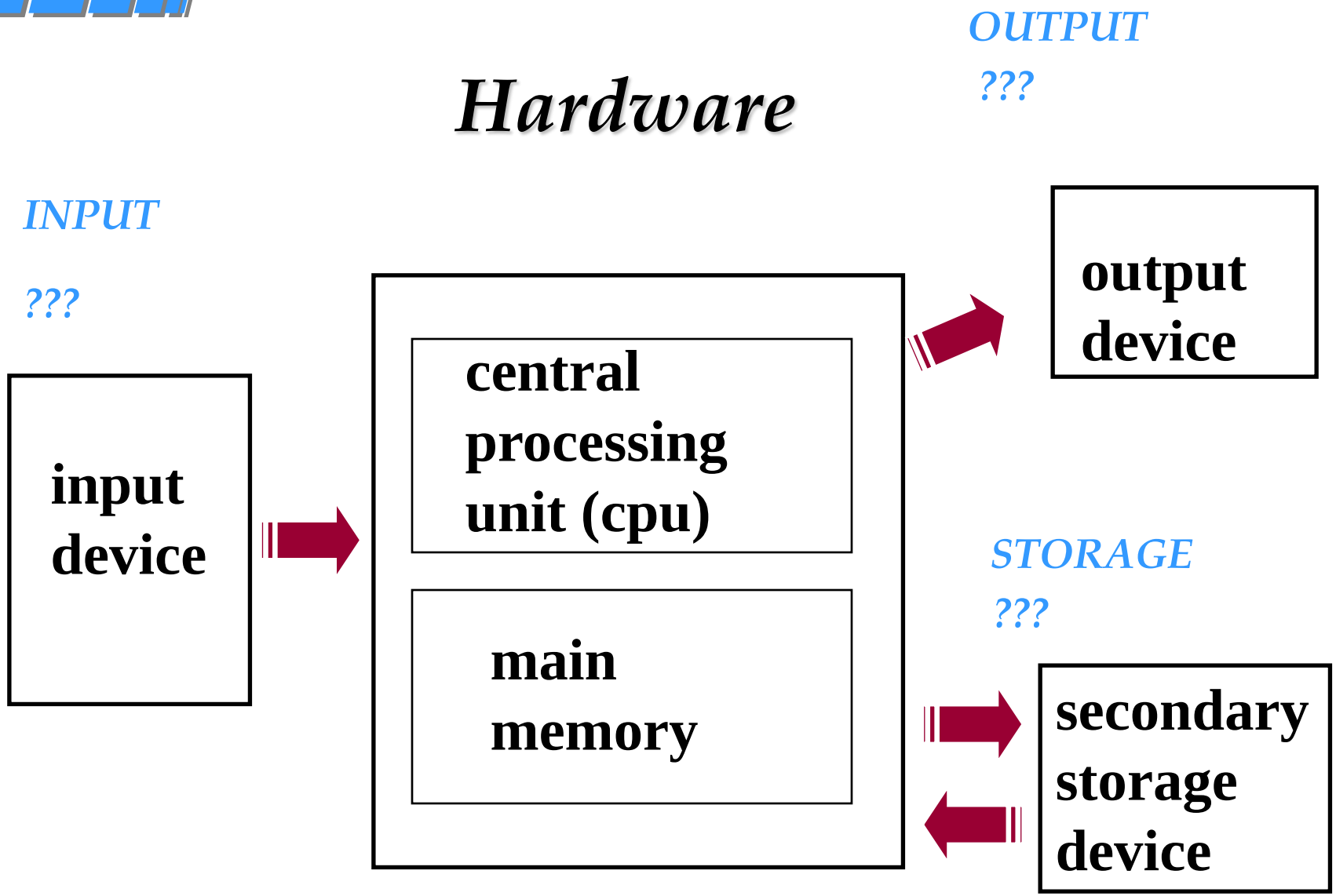
A computer system has these components:

- Hardware
- Software
- Procedures
- Data
- People

- *Hardware* is the machine itself.
- *Software* tells the hardware what to do.
- *Procedures* tell people how to use the system.

A computer processes *data* and produces *information* useful to people.

Hardware


OUTPUT
???
INPUT
???
output device
central processing unit (cpu)
input device
main memory
STORAGE
???
secondary storage device

Hardware

INPUT

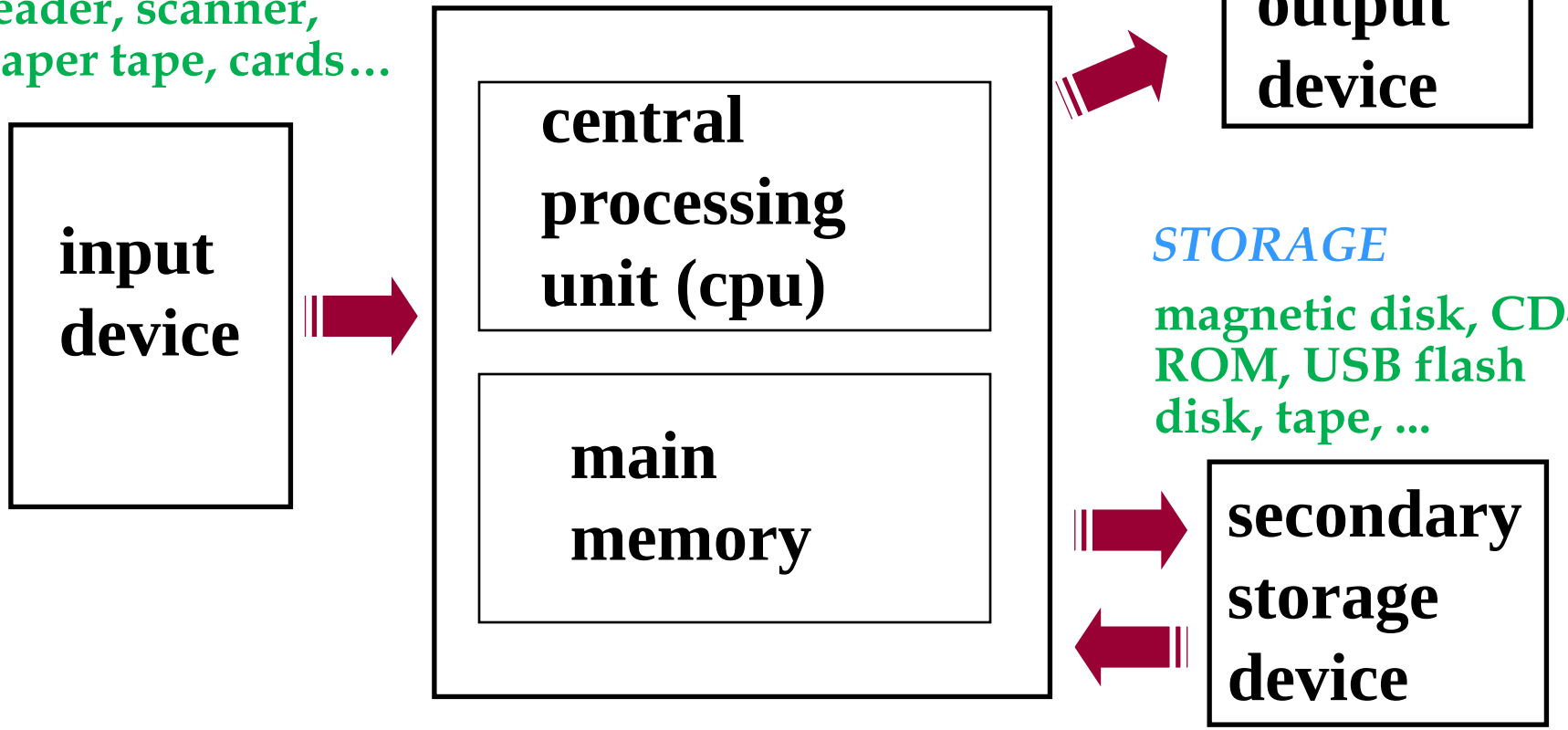
keyboard, mouse, joystick, bar code reader, scanner, paper tape, cards…

OUTPUT

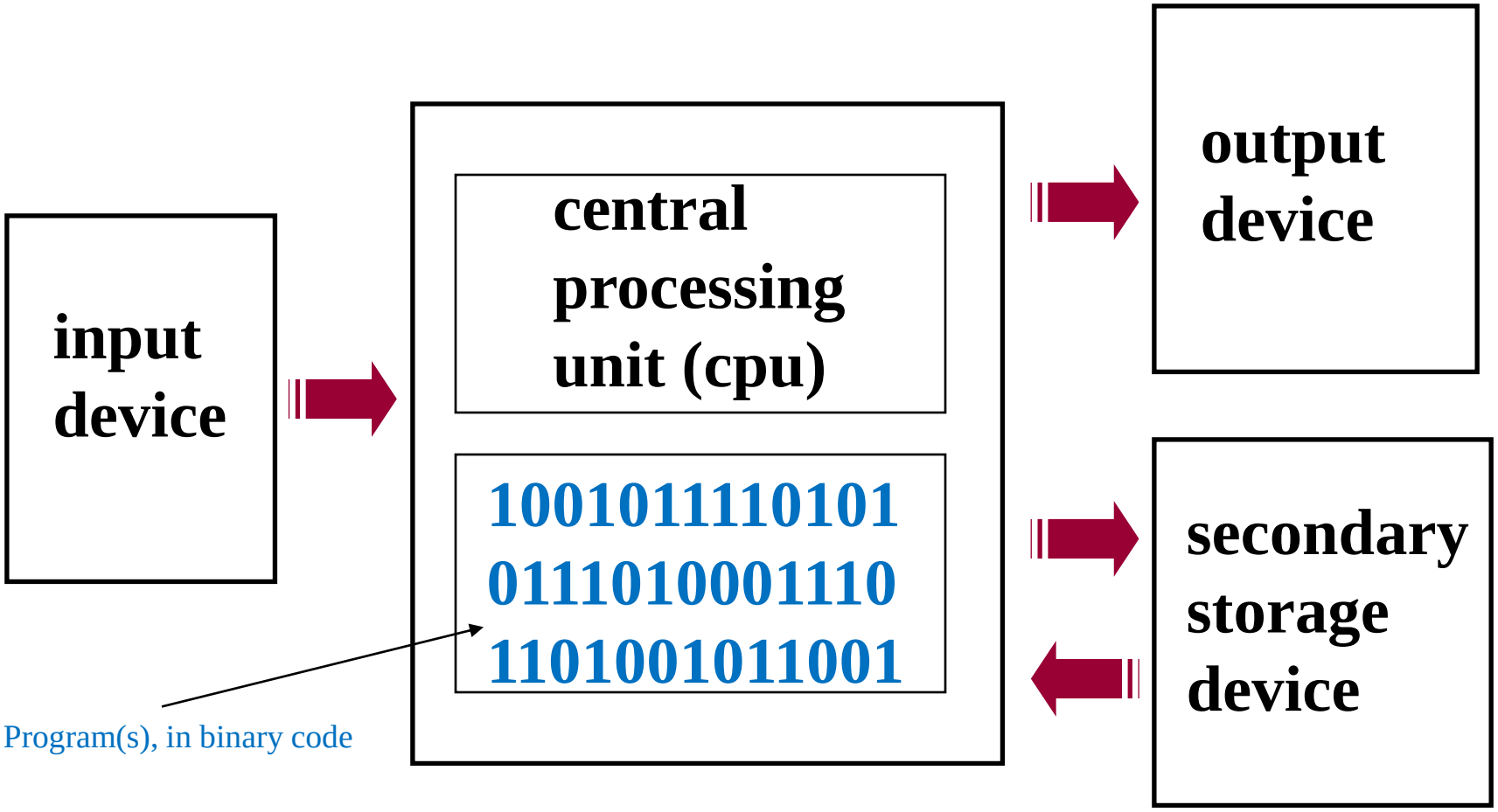
screen, printer, plotter, speech synthesizer, paper tape...

STORAGE

magnetic disk, CD-ROM, USB flash disk, tape, ...



Hardware


input device
central processing unit (cpu)
1001011110101
0111010001110
1101001011001
output device
secondary storage device
Program(s), in binary code

Software

Two broad classifications:

System software — computer programs to control the computer and provide a user interface (e.g. operating system, windowing system, file management software)

Application software — computer programs to perform specific tasks (e.g. word processor, database management system, web browser, your programs)

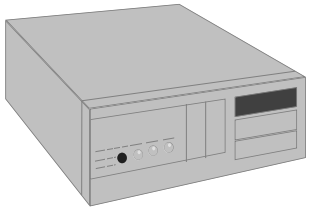
What is a computer program?

- a set of ***instructions*** to be obeyed (or executed) in order to solve a problem
- typically ***data*** plus ***processing***
- purpose is to convert *data* into *information*
- particularly useful for large amounts of data and complicated processing

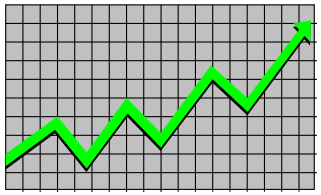
Program

Data

W%312a$R5¢

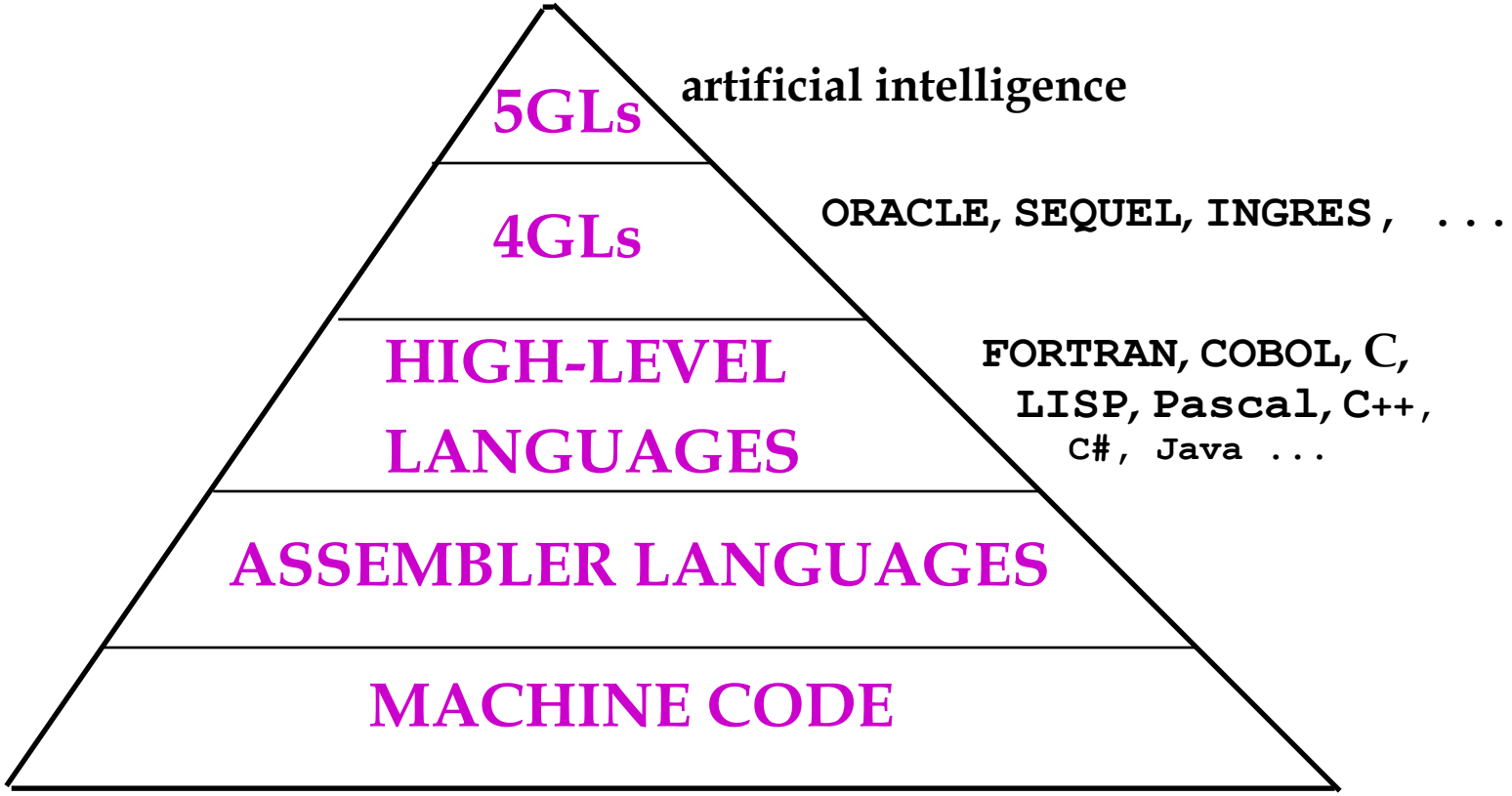


Computer

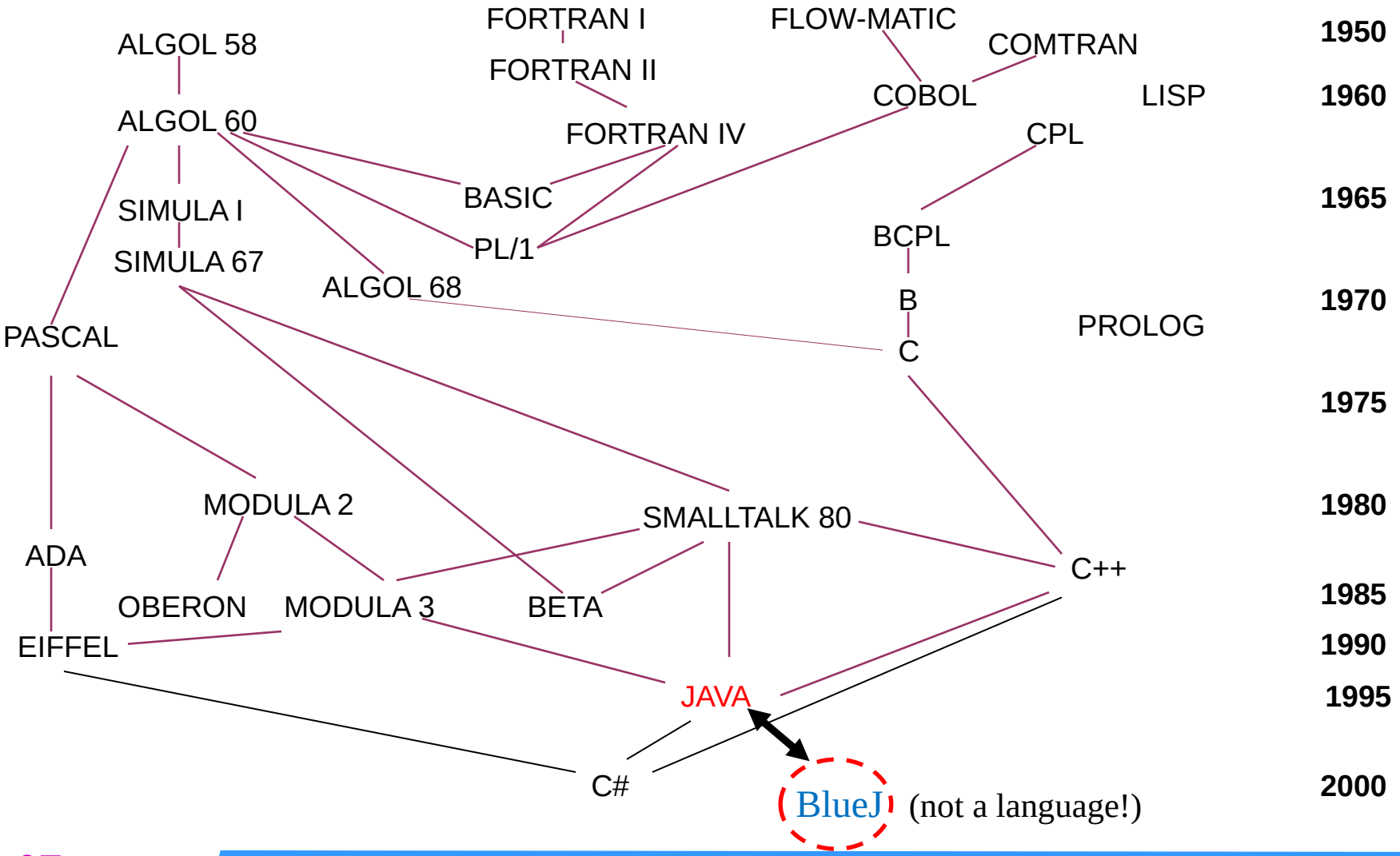


Output

History of programming languages



Language history



Different programming languages

Low-level

Machine code

```
• 000 100C
• 006 300D
• 007 200D
• 100
• 110
```

Assembler

```
LDA VALUE
ADA TOTAL
STA TOTAL
DW1 TOTAL
DW1 VALUE
```

High-level

COBOL	`ADD VALUE TO TOTAL`
FORTRAN	`TOTAL = TOTAL + VALUE`
LISP	`(set! total (+ total value))`
Pascal	`total := total + value`
Java	`total += value;`

Program translation

- a program written in a high-level language (such as Java) must be translated into machine code before the computer can run it (i.e. *execute* the instructions). This is because a computer can really only understand machine code.

- the translation is typically done by a process called a *compilation*. There is another similar process called *interpretation* which is also used to translate a program.

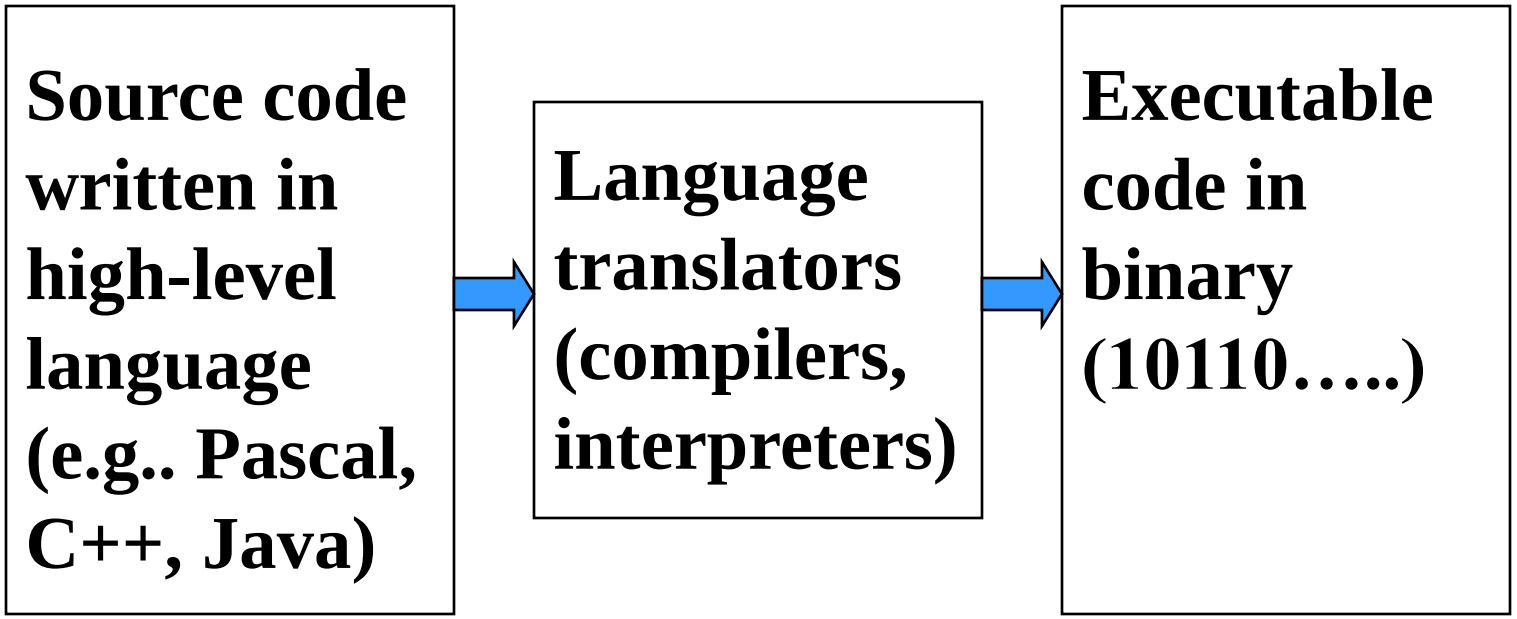
Compiler vs. interpreter

The main difference between a *compiler* and an *interpreter*:

- **a compiler attempts to translate an entire program at once, and gives a list of the problems it found (if any) in the end. This is usually a *batch* process.**

- **an interpreter, on the other hand, translates one instruction at a time and causes that instruction to be executed immediately. This is usually an *interactive* process.**

Program translation



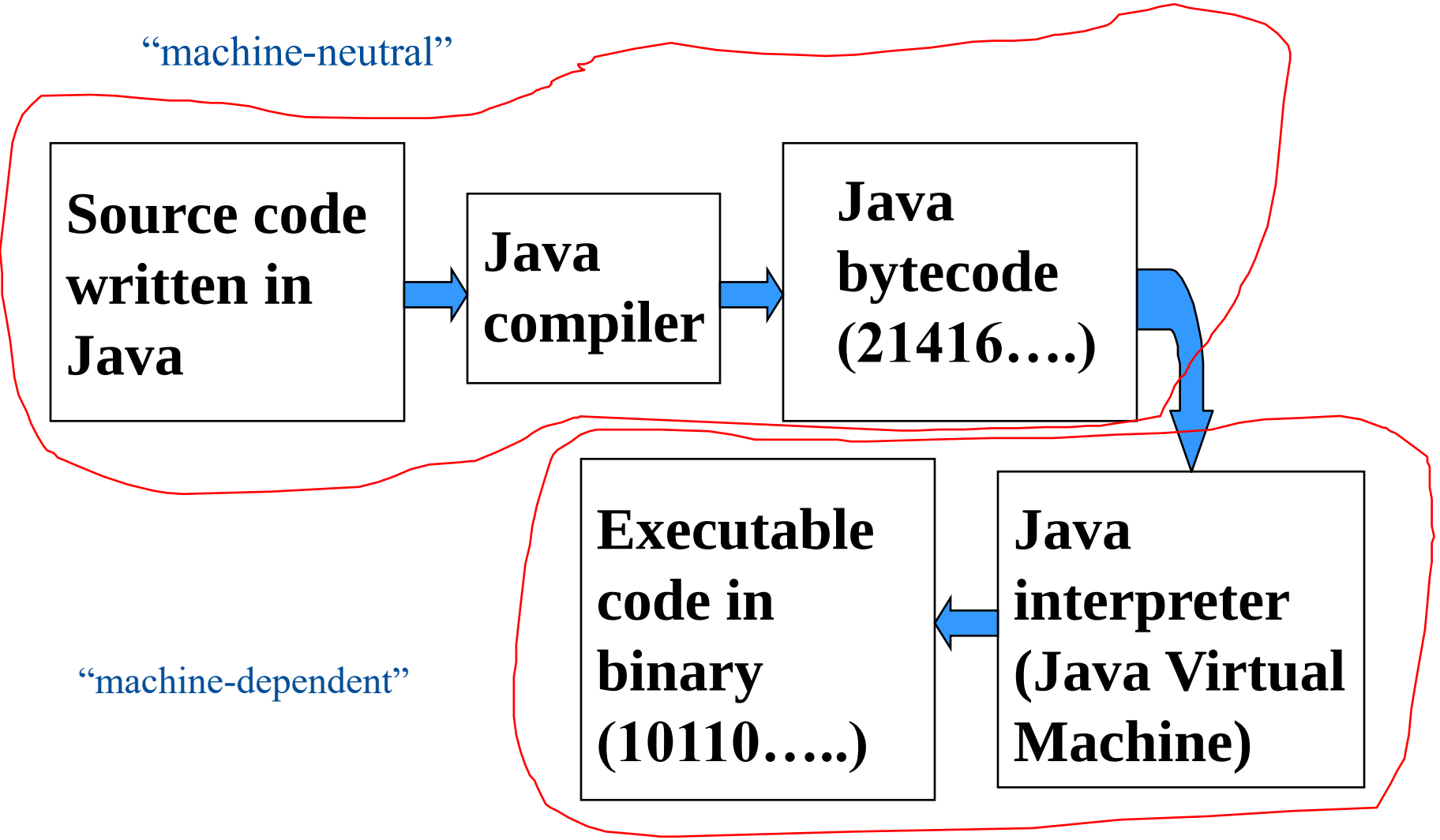
Java program translation

Program translation in Java involves two phases:

1. **Java source code is (typically) *compiled* to something called *bytecodes*.**
2. **The bytecodes are then *interpreted* to machine code by a *Java Virtual Machine* (JVM).**

The Java Virtual Machine is just a piece of software. Each type of machine (e.g. PC, Mac) has its own Java Virtual Machine to convert bytecode to the native machine code.

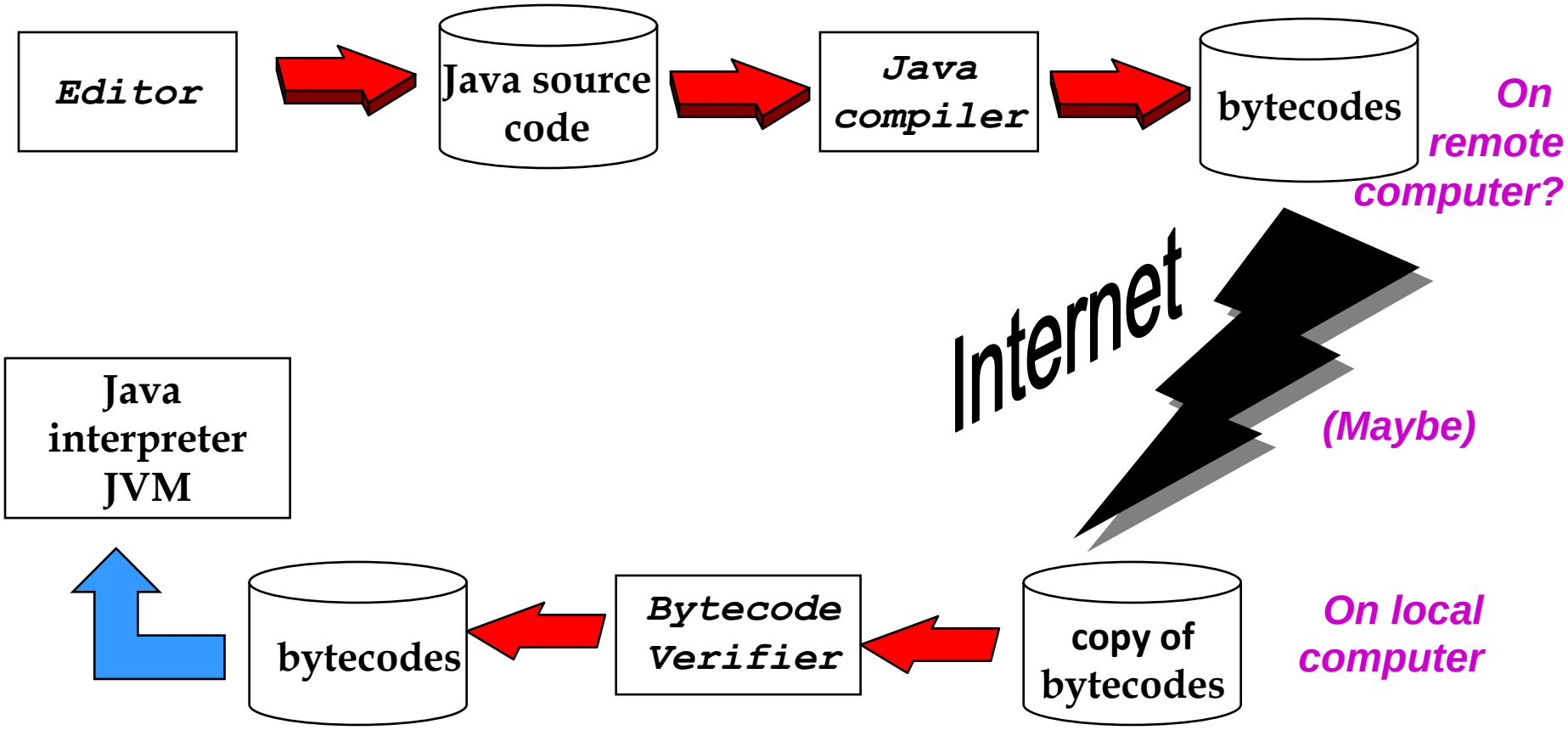
Java program translation



Java: write once, run anywhere!

- Java programs are able to run on any kind of computer when downloaded from the Web.
- With most languages, that would mean downloading source code for the program and having a compiler translate it into the machine code for your machine. The user would have to tell the machine to compile the source code before running the program.
- Java uses *bytecodes* to solve this problem.

Executing a Java program



A computer program (in Java)

```
public class InterestCalculator
{
    private int interestRate;
    private int balance;
    private int interest;

    public InterestCalculator()
    {
        interestRate = 5;
        balance = 1000;
    }

    public void calculateInterest()
    {
        interest = balance * interestRate / 100;
    }
}
```

Java and BlueJ

In FIT9131 this semester we will learn Java using the *BlueJ* development environment. BlueJ was designed for teaching introductory object-oriented programming in Java. It provides:

- a full Java development environment
- visualisation of *classes* and *objects*
- an *editor*
- an *inspector*
- a *debugger*

Both Java and BlueJ are free. You will be given instructions about how to use them in the computer labs and at home.

Programming paradigms

There are several different *styles* of writing computer programs. These are called "*paradigms*". Some common paradigms are *procedural*, *functional*, *logic*, *object-oriented*, and *event-driven*. Different languages typically use different paradigms, to solve different kinds of problems. Some programming languages can use more than one paradigm.

Java uses the *object-oriented* paradigm. This means that the basic building blocks of a program in Java are *objects* and *classes*.

Object-oriented programs

The *object-oriented* (*OO*) programming paradigm uses *objects* to model a situation. A program can be written in a non-object-oriented way, but the object-oriented style is increasingly popular for three main reasons:

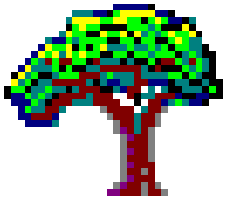
- programs are becoming more and more complex, and the OO style handles this best
- the OO style makes it easy to re-use existing programs or parts of programs
- the OO style makes programs much easier to maintain

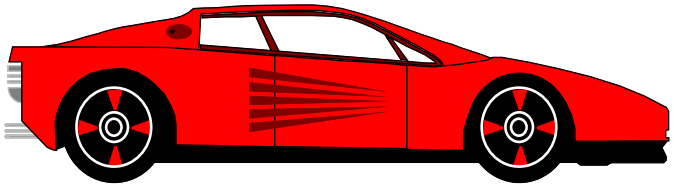
Objects

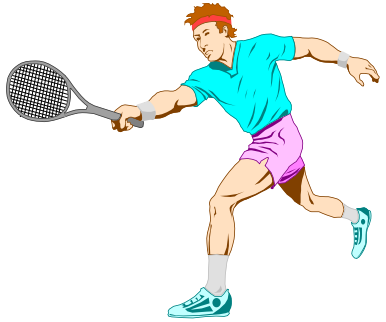
We are all familiar with the idea of an object in everyday life. We are surrounded by them: cars, books, people, houses, cats, etc.

Objects have *attributes*, e.g. colour, size, age, name.

Objects also have *behaviour*. They can *do* things, e.g. grow, breathe, run, stop.







Attributes and Behaviour

e.g.:

	Attributes	Behaviour
Car	Registration number Manufacturer Model Year Colour	Start Stop Turn Accelerate
Cat	Name Breed Colour Age Weight Registration number	Sleep Eat Purr Climb Scratch

Classification

We can classify objects into groups according to their common attributes and behaviour.

Let's say we want to classify objects into cats and non-cats. What are the common attributes and behaviour of something we would classify as a cat? How would you recognise a cat if you saw one?



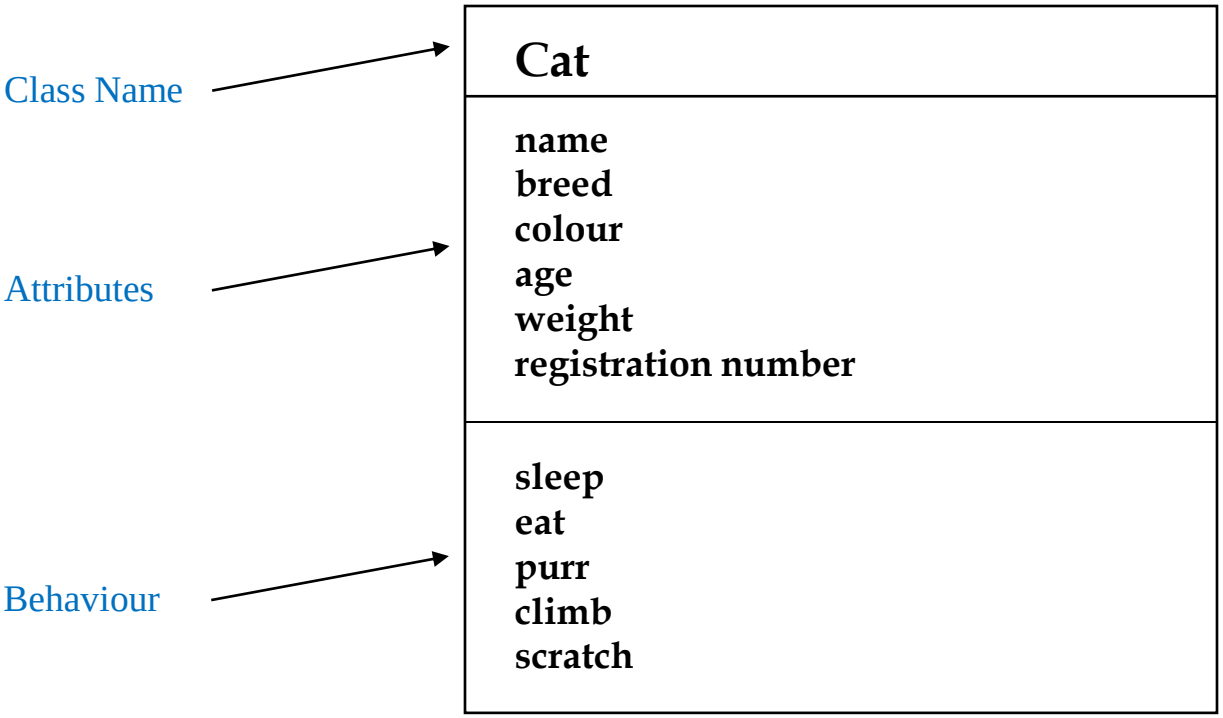






Classes

In a computer program, we use *classes* to describe a group of similar objects. A *class* is a description of what an object would look like if we had one. It is based on common attributes and behaviour of objects.



A Class Diagram

Instances of classes

A class is a *template* or *pattern* from which many objects can be created. It is a description of an object.

Each object created from a class is called an *instance* or an *object* of that class. This process is called *instantiation*.

We can have *multiple instances (objects)* of the same class. Each object has the same attributes and the same behaviour. However, the values of the attributes might be different.

State of an object

The *state* of a particular object is the values of its attributes at any particular moment. For example, a cat might have the following state:

Name: Stimpy

Breed: Burmese

Colour: Lilac

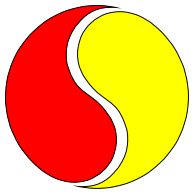
Age: 11 years, 4 months, 8 days

Weight: 3.6 kg

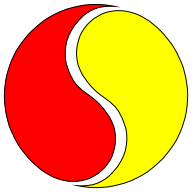
Registration number: 213

Some of these attributes change constantly. Others stay the same for long periods, or forever.

Identity of an object

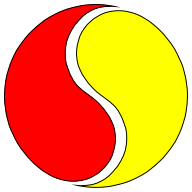


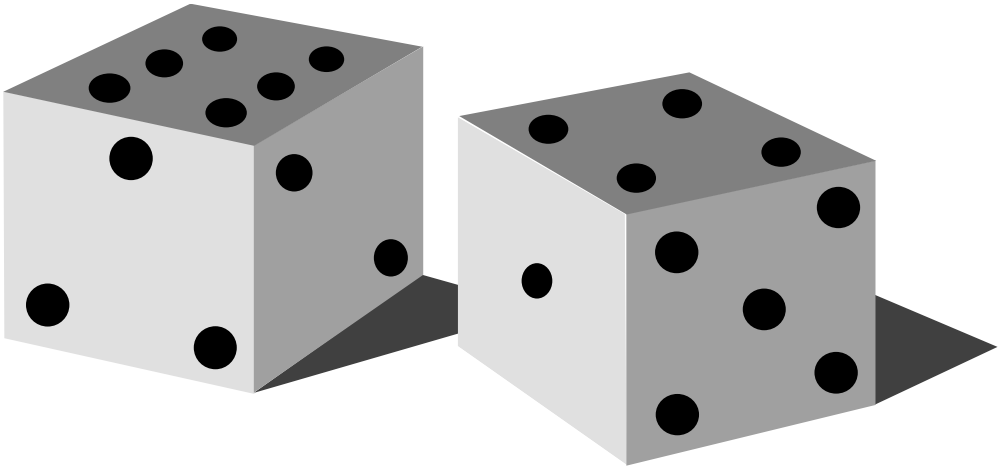
We can have several different objects that have the same attributes and behaviour (ie. the same state) but they are still *different objects*.

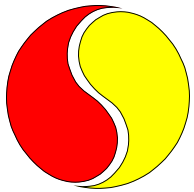


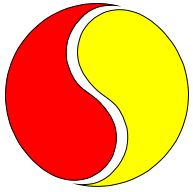
An object has *identity*. We can give it a name to distinguish it from the others (e.g. ball1, ball2).

in other words, ***every object is unique!***









Making something happen

If you have an existing Class, in order to make something happen, you need to:

1. Create an object of that class.

2. Ask that object to do something that it knows how to.

You will experiment with doing this in *BlueJ* during the lab sessions.