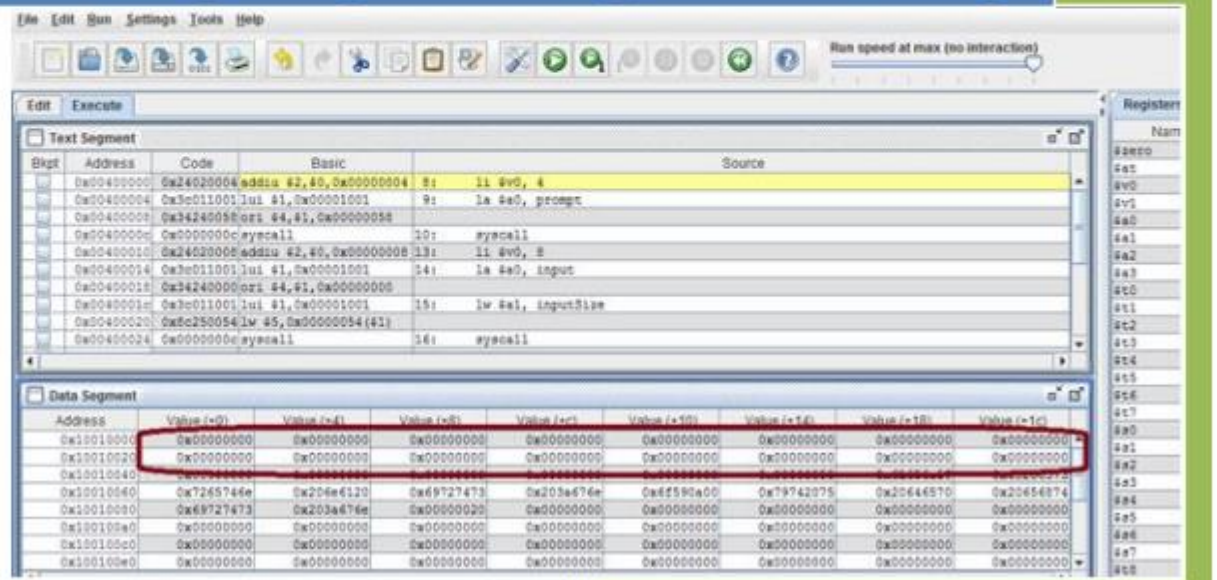
Компилиране и изпълнение на JAVA програми

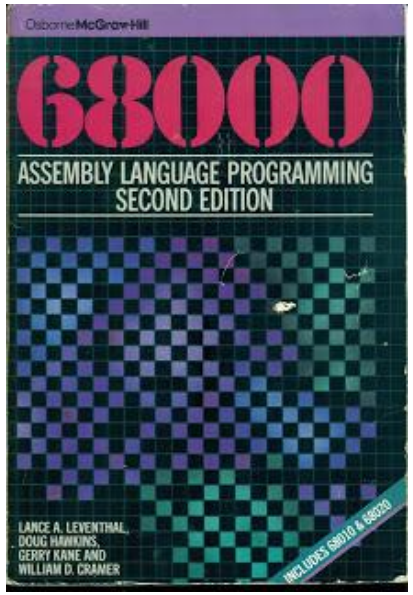
доц. д-р Мария Маринова

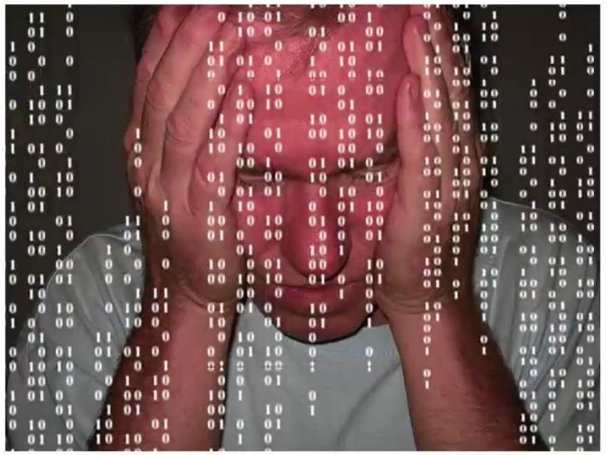
Машинен код

- Процесорът изпълнява група **инструкции** на всеки такт
- Програмата -> множество инструкции
- Инстрикцията -> различна последователност от **0** и **1**
- Машинният език <-> машиния код
- Например: 000000 00001 00010 00110 00000 100000

Introduction to MIPS Assembly Language Programming







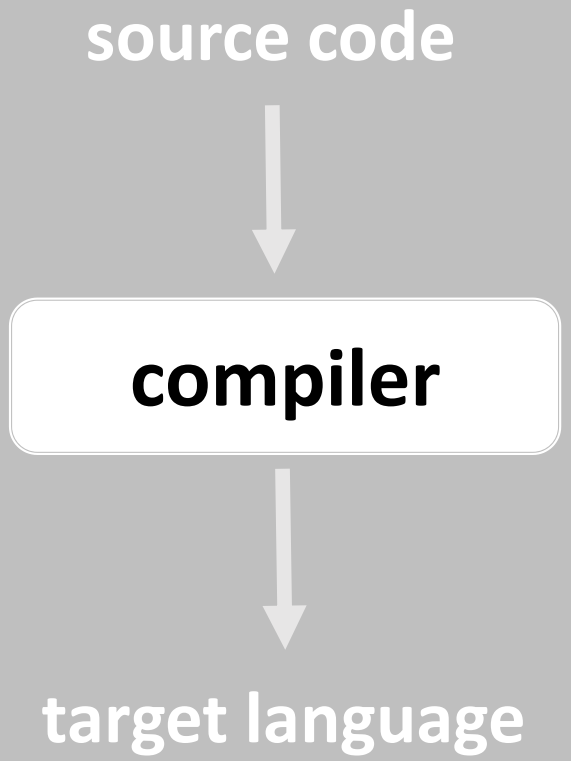
Програмни езици от ниско ниво

- Машинен език & асемблерен език
- Използват програмиране от ниско ниво, например адресиране на памет
- Писането на такива програми е изключително трудно и програмиста трябва да е запознат с процесорната архитектура, за да напише дадена програма;

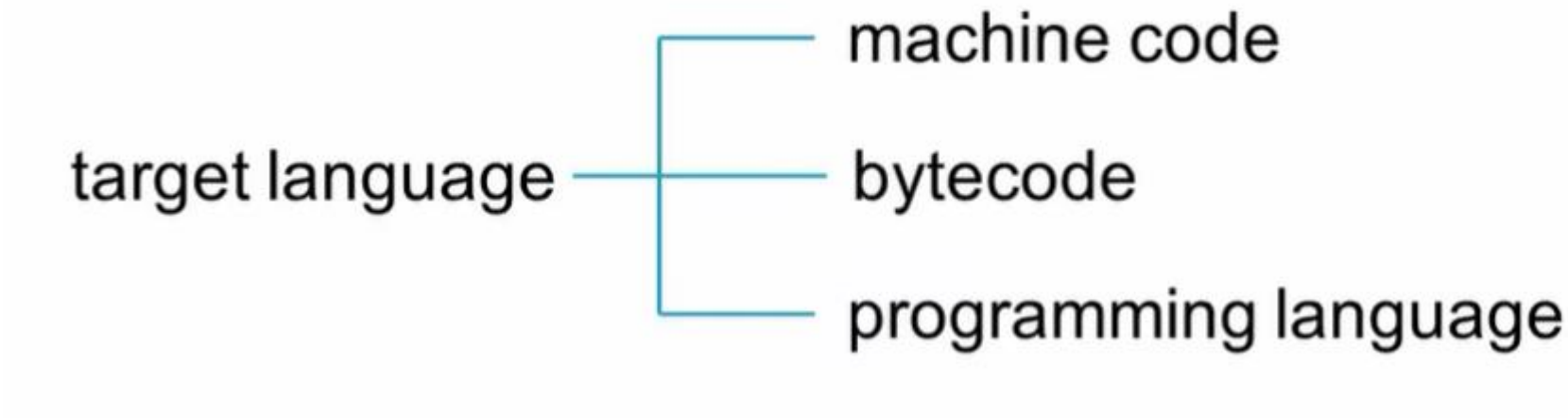
Програмни езици от високо ниво

- C/C ++, Java, C#
- използват english-like думи, математически обозначения
- Езиците от ниско ниво се използват при програмиране на електоника, вградени схеми и в ОСи;
- source code

Компилиране



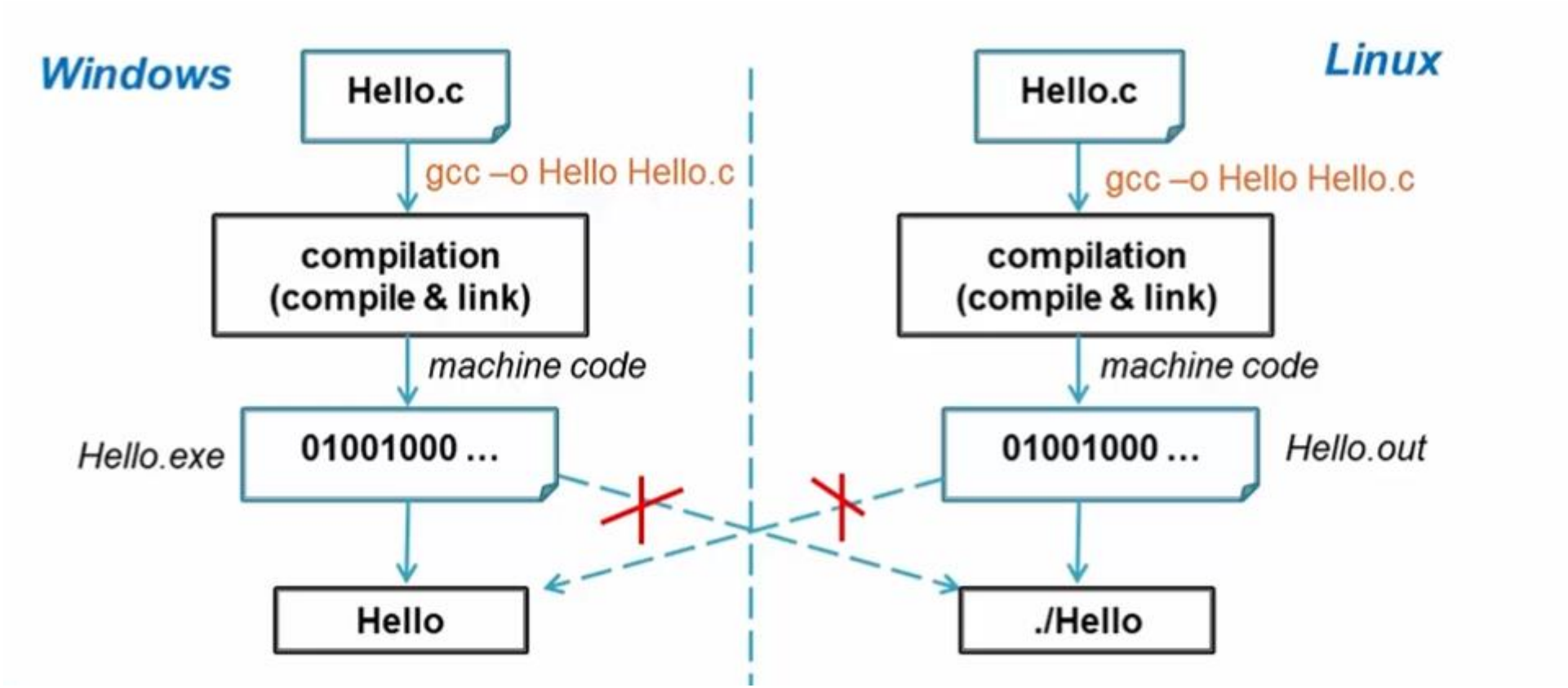
Компилиране



Операции при компилирането

- Проверка на синтаксиса и семантиката на сорс кода
- Оптимизиране за бързо изпълнение
- Генерира машинен код

Изпълнение на програмата Hello, зависима от платформата



Зависим код от платформата

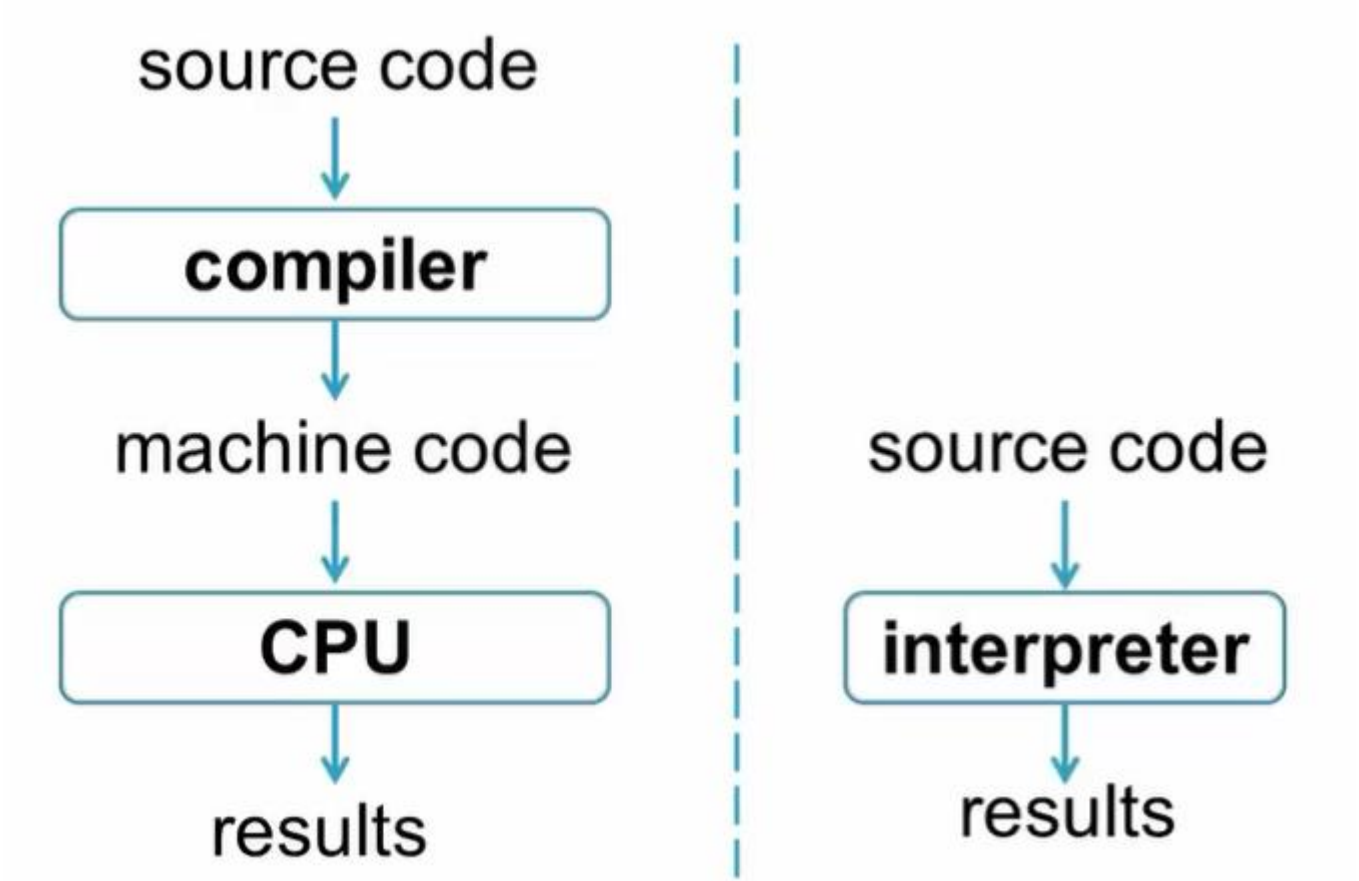
ОС

- изпълним файл – PE(windows), ELF(Linux)
- System calls

Hardware

- Processor – x86 vs ARM instruction sets

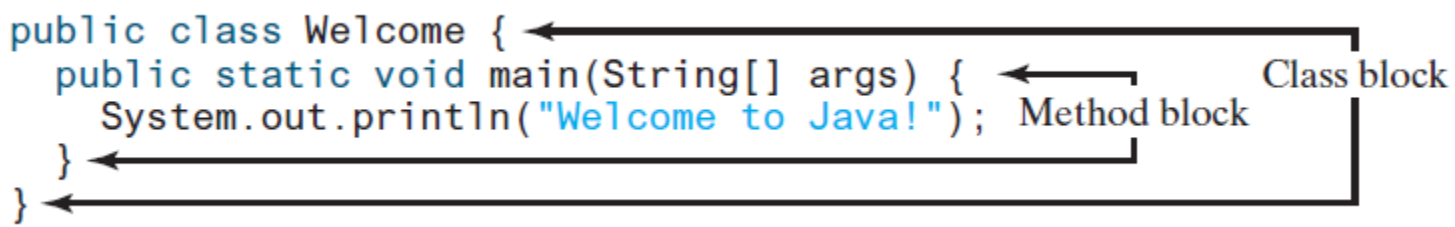
Интерпретатор

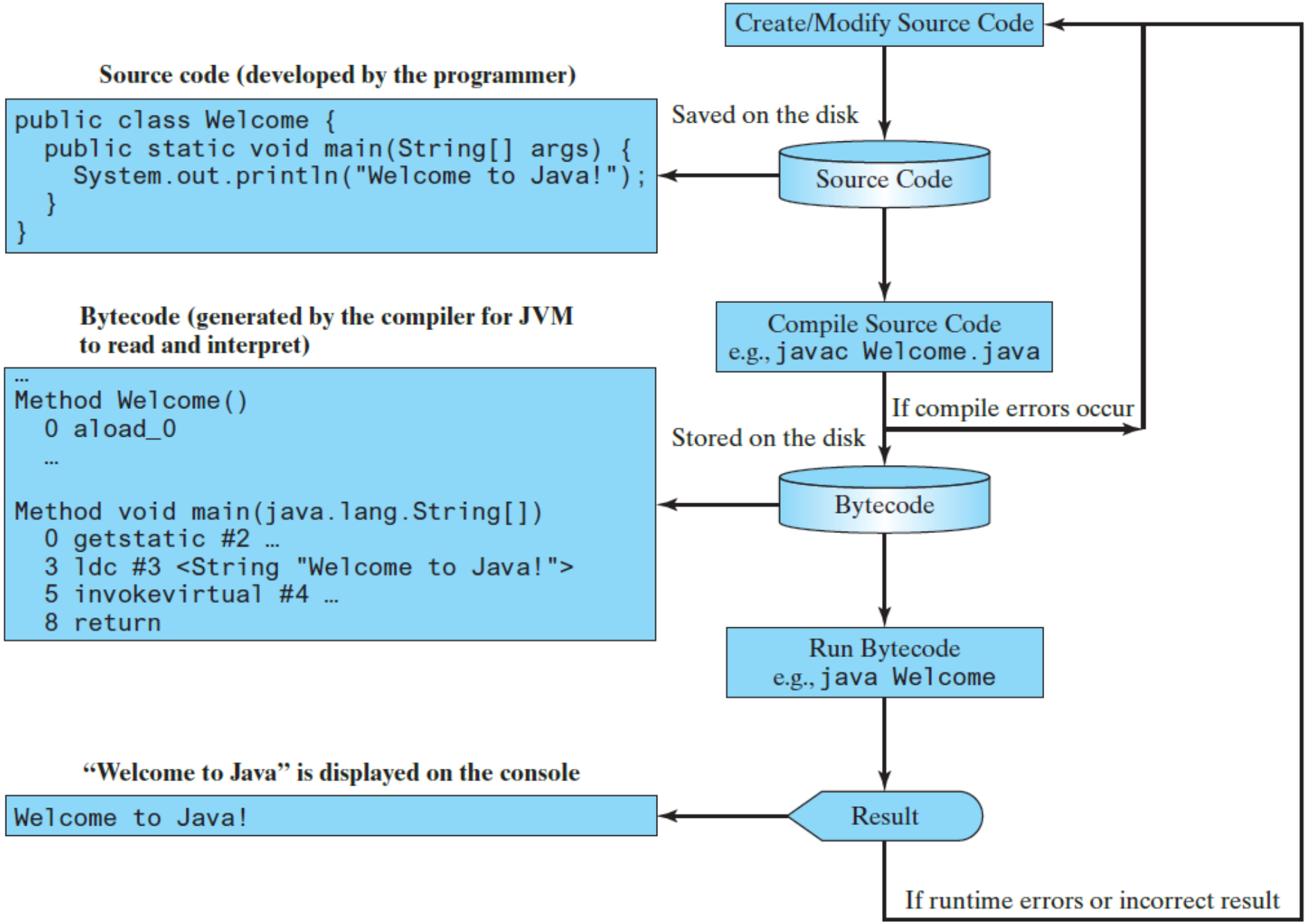


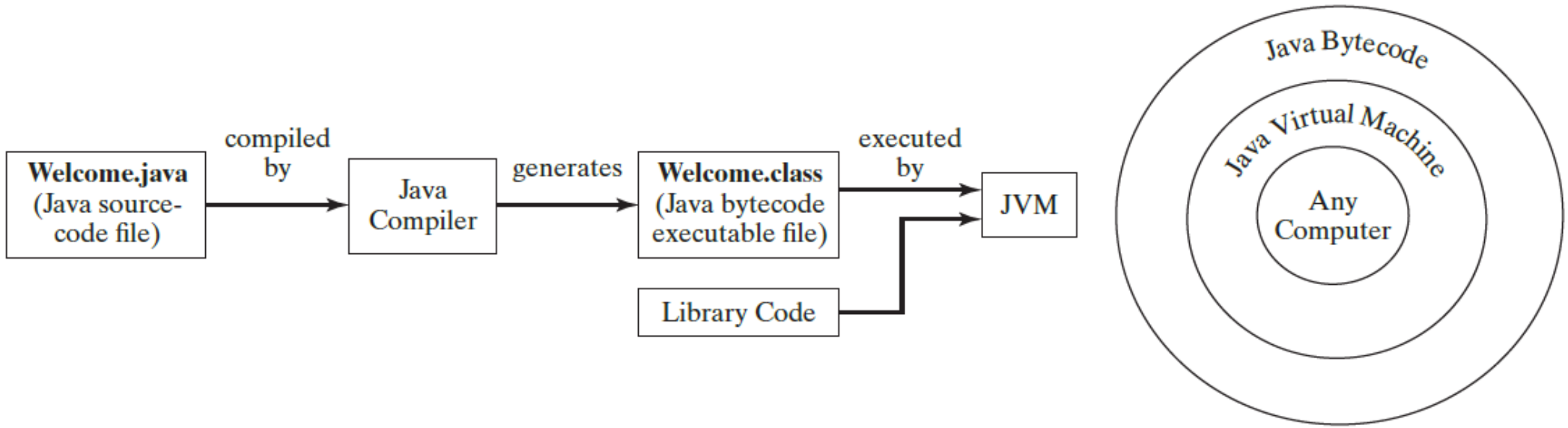
LISTING 1.1 Welcome.java

```
1  public class Welcome {
2    public static void main(String[] args) {
3      // Display message Welcome to Java! on the console
4      System.out.println("Welcome to Java!");
5    }
6  }
```

```
Welcome to Java!
```

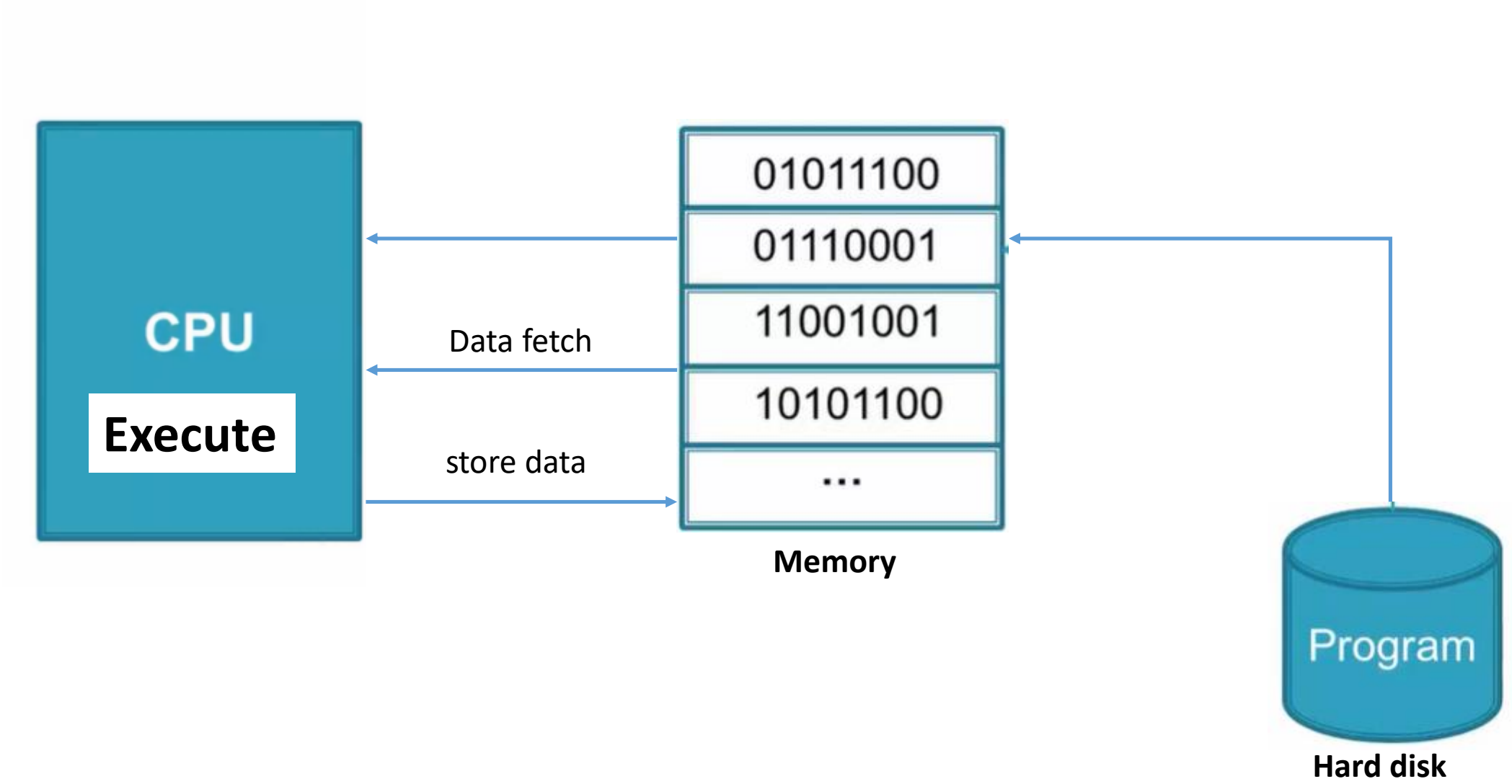



Create/Modify Source Code
Source code (developed by the programmer)
public class Welcome {
public static void main(String[] args) {
System.out.println("Welcome to Java!");
}
}
Saved on the disk
Source Code
Compile Source Code
e.g., javac Welcome.java
If compile errors occur
Bytecode (generated by the compiler for JVM to read and interpret)
...
Method Welcome()
0 aload_0
...
Method void main(java.lang.String[])
0 getstatic #2 ...
3 ldc #3 <String "Welcome to Java!">
5 invokevirtual #4 ...
8 return
Stored on the disk
Bytecode
Run Bytecode
e.g., java Welcome
"Welcome to Java" is displayed on the console
Welcome to Java!
Result
If runtime errors or incorrect result



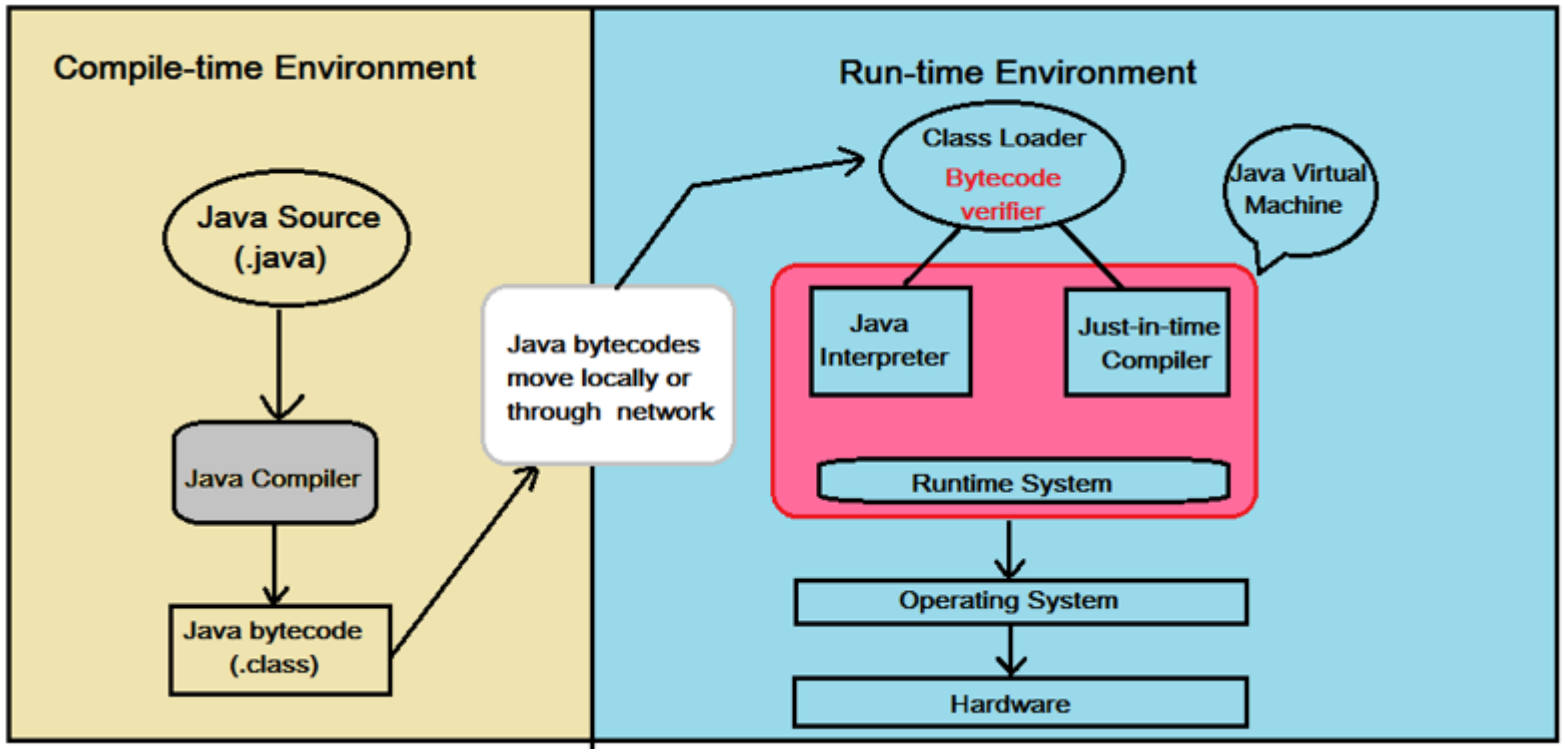
(a) Java source code is translated into bytecode. (b) Java bytecode can be executed on any computer with a Java Virtual Machine.

Fetch-execute life cycle

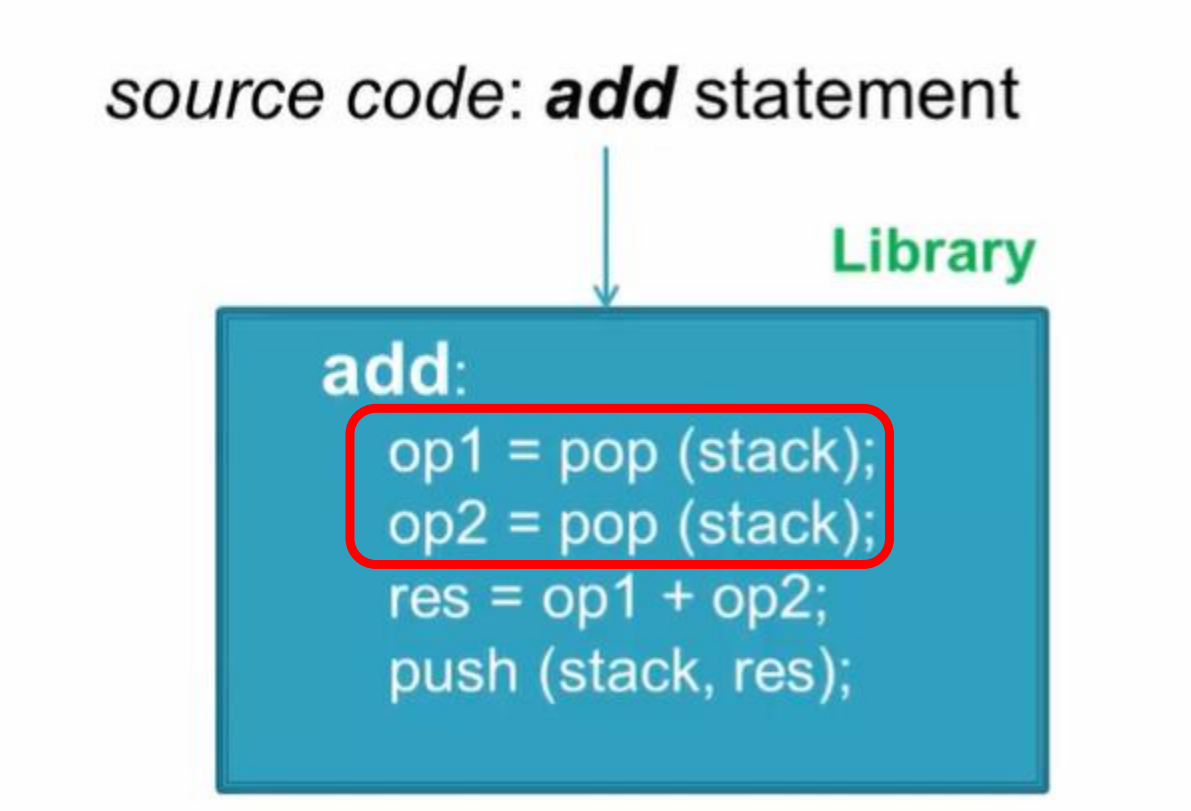


Изпълнение на bytecode от интерпретатора:

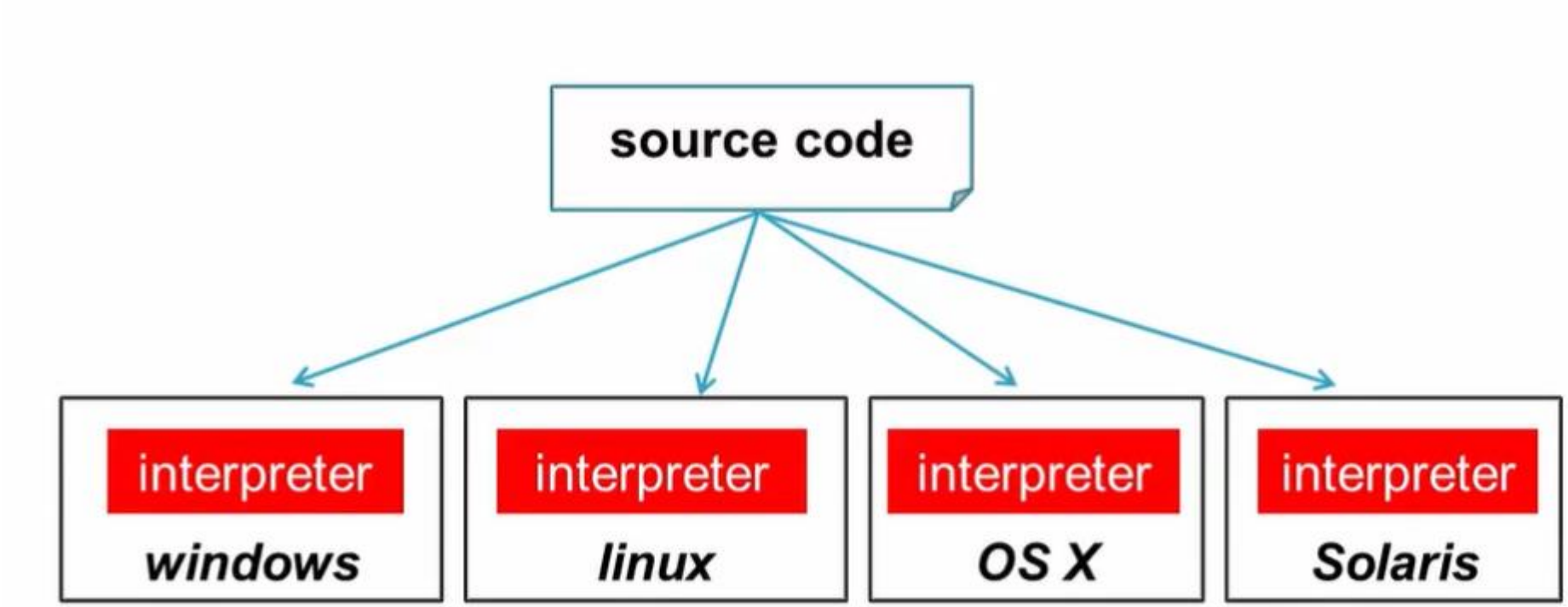
- Извличане на следващото program statement
  - Какво е необходимо, за да се извлече състоянието
- Изпълнява се **прекомпилирания машинен код** в неговите библиотеки

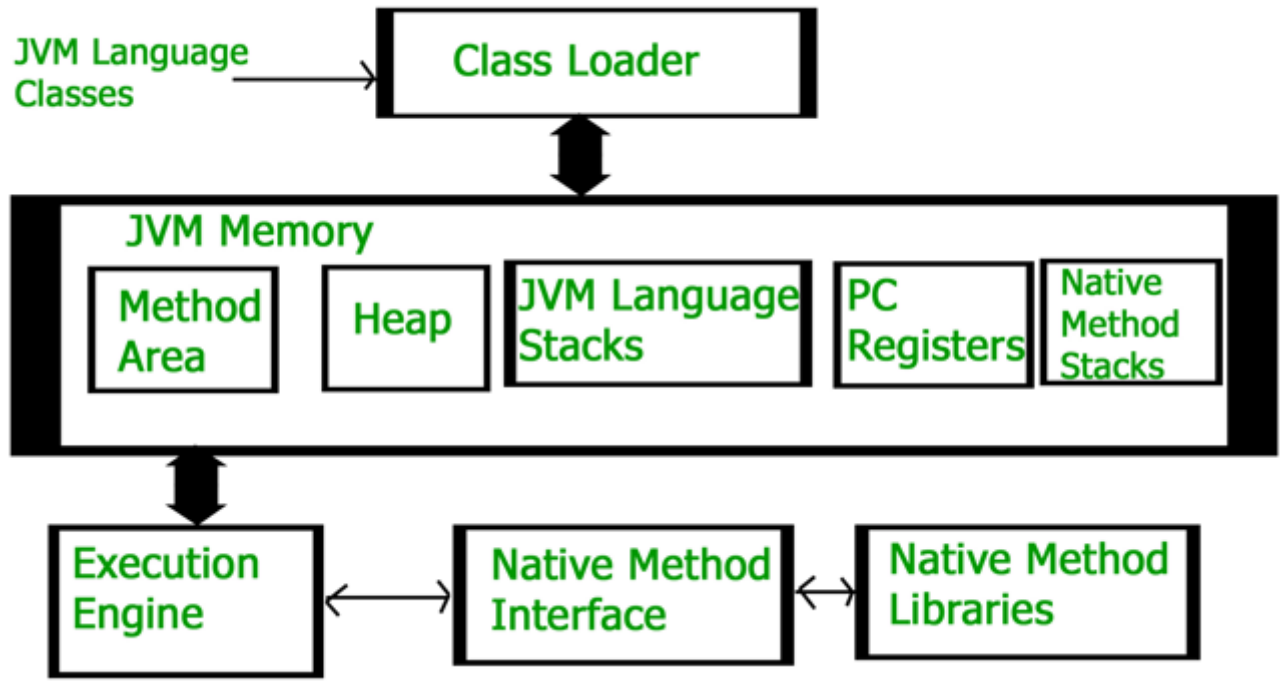


Стъпки на изпълнението:



Java е независима от платформите!




JVM Language
Classes
Class Loader
JVM Memory
Method
Area
Heap
JVM Language
Stacks
PC
Registers
Native
Method
Stacks
Execution
Engine
Native Method
Interface
Native Method
Libraries

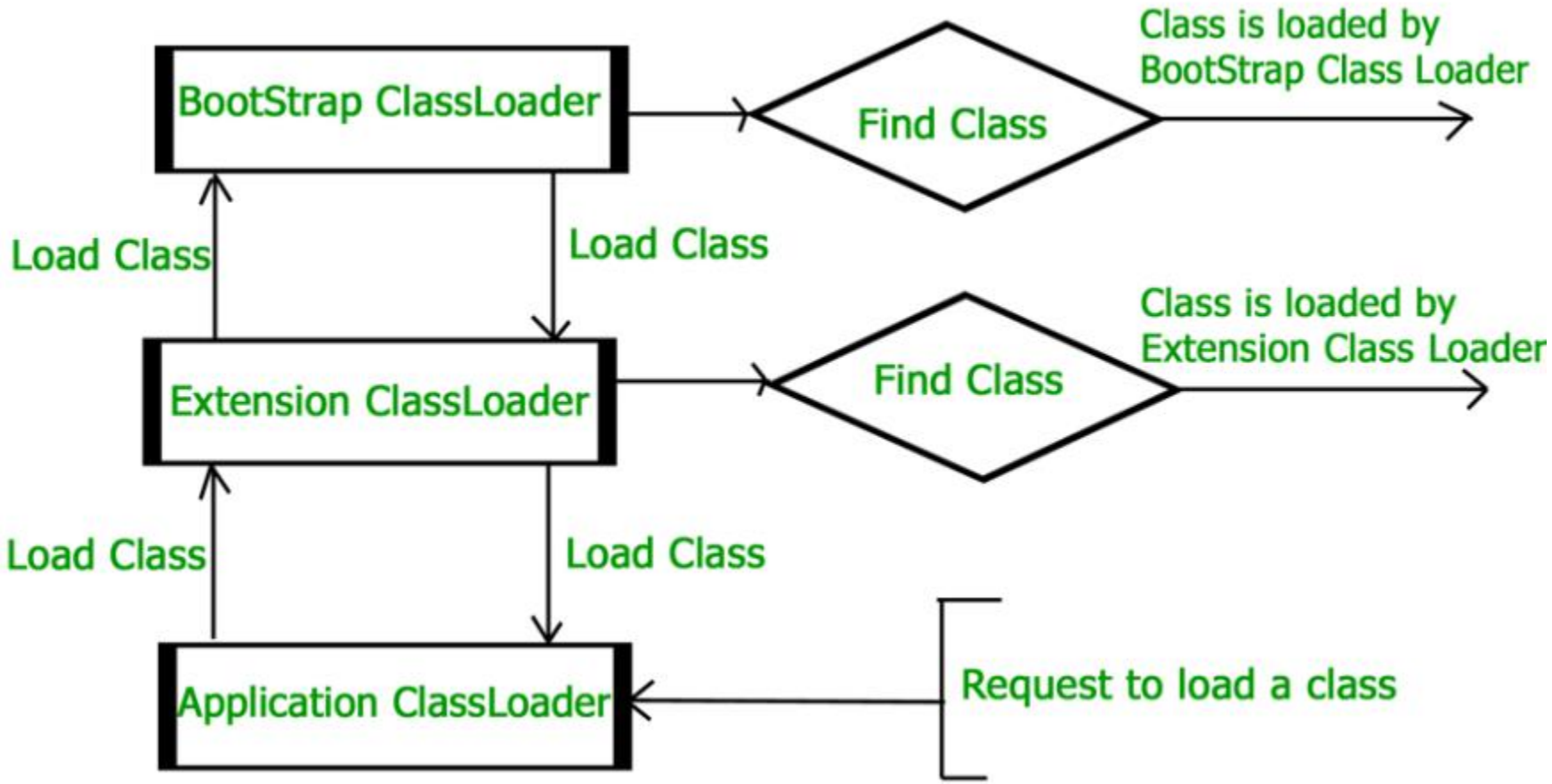
```
// A Java program to demonstrate working of a Class type
// object created by JVM to represent .class file in
// memory.
import java.lang.reflect.Field;
import java.lang.reflect.Method;

// Java code to demonstrate use of Class object
// created by JVM
public class Test
{
        public static void main(String[] args)
        {
                Student s1 = new Student();

                // Getting hold of Class object created
                // by JVM.
                Class c1 = s1.getClass();

                // Printing type of object using c1.
                System.out.println(c1.getName());
```

```
                    // getting all methods in an array
                    Method m[] = c1.getDeclaredMethods();
                    for (Method method : m)

            System.out.println(method.getName());

                    // getting all fields in an array
                    Field f[] = c1.getDeclaredFields();
                    for (Field field : f)

            System.out.println(field.getName());
            }
}

// A sample class whose information is fetched above using
// its Class object.
class Student
{
            private String name;
            private int roll_No;
```

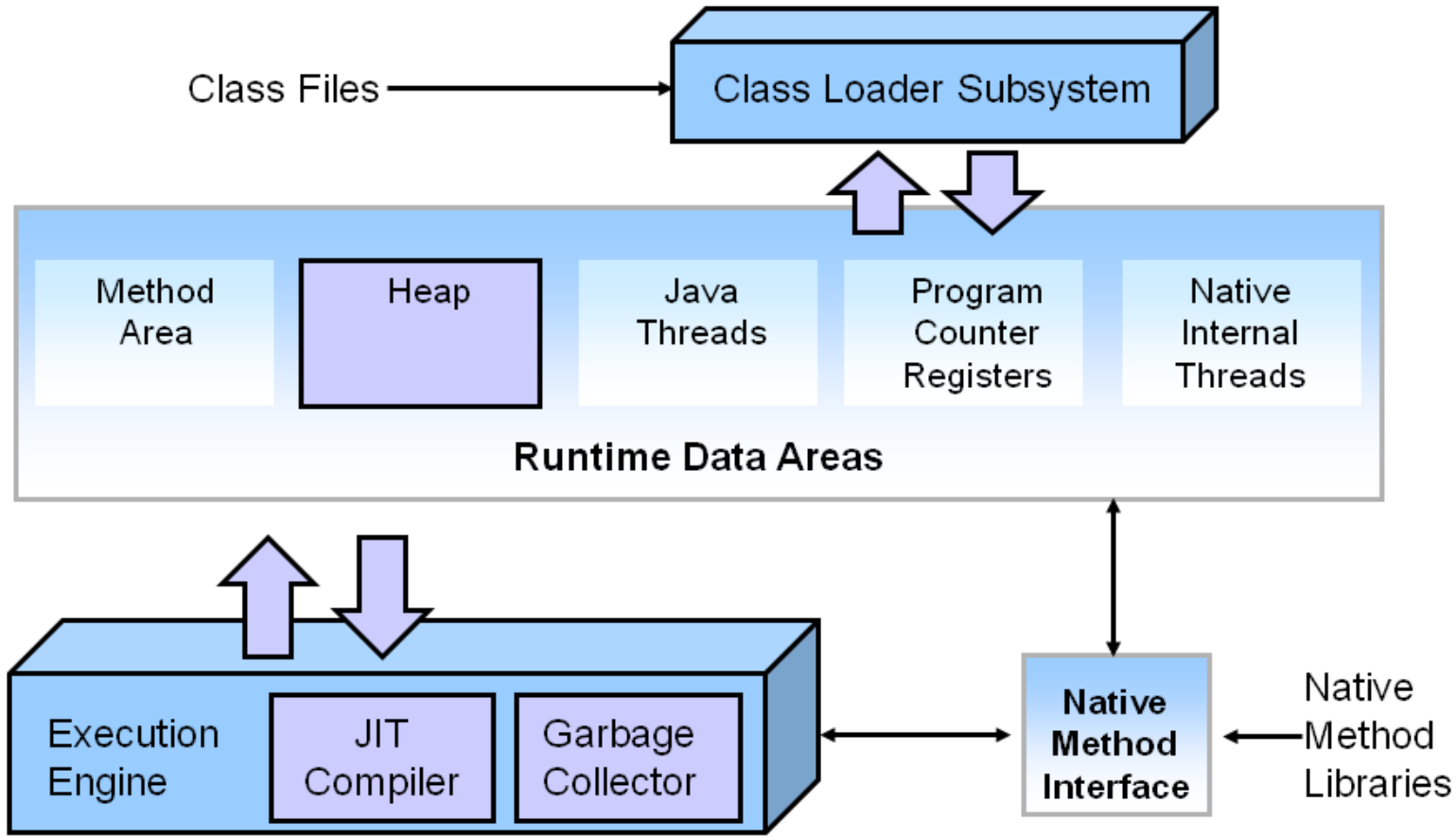
```
        public String getName() { return name; }
        public void setName(String name) { this.name =
name; }
        public int getRoll_no() { return roll_No; }
        public void setRoll_no(int roll_no) {
                this.roll_No = roll_no;
        }
}
```

```
// Java code to demonstrate Class Loader subsystem
public class Test
{
        public static void main(String[] args)
        {
                // String class is loaded by bootstrap loader,
and
                // bootstrap loader is not Java object,
hence null

        System.out.println(String.class.getClassLoader());

                // Test class is loaded by Application loader

        System.out.println(Test.class.getClassLoader());
        }
}
```

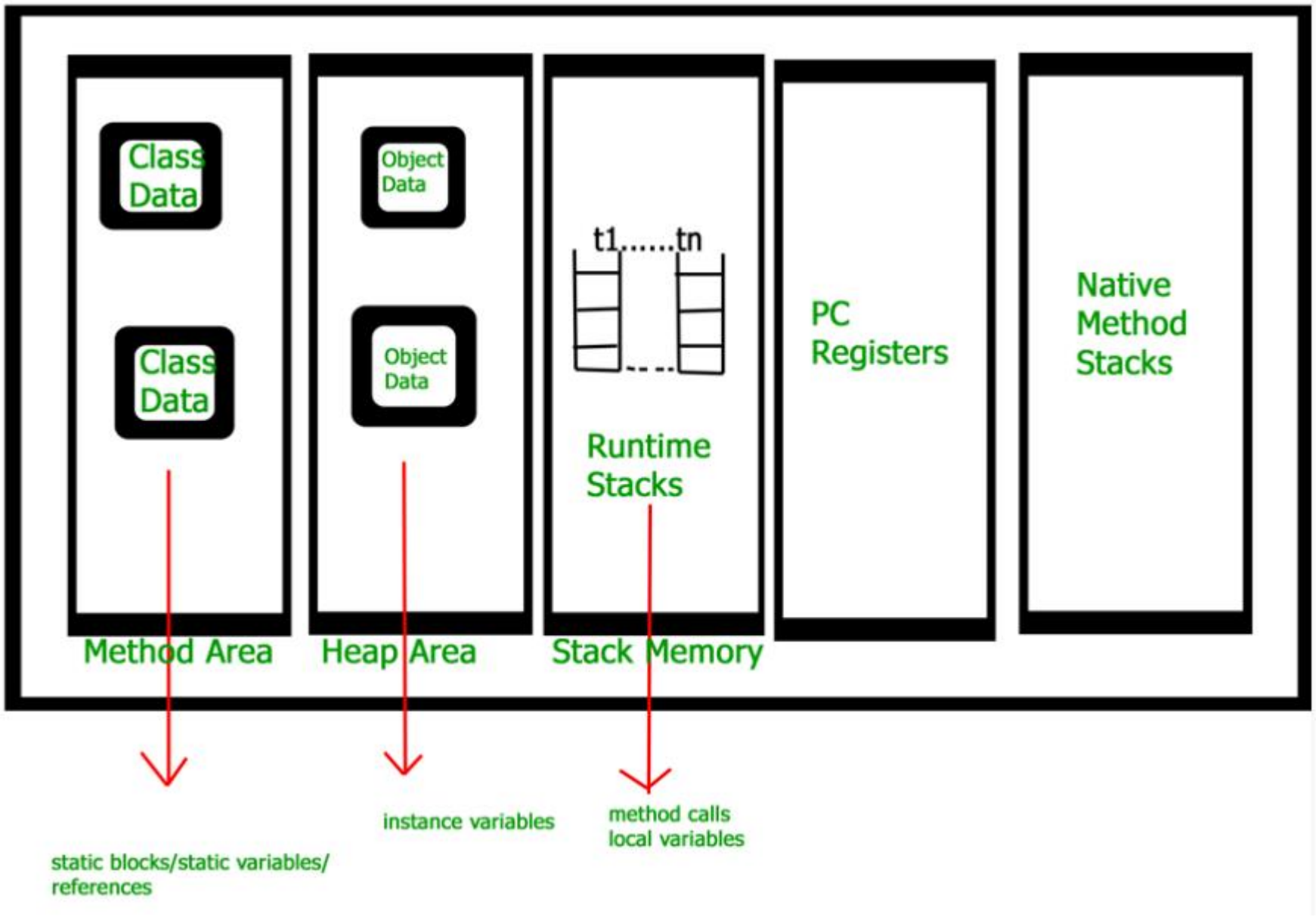
JVM follow Delegation-Hierarchy principle to load classes.



Key HotSpot JVM Components

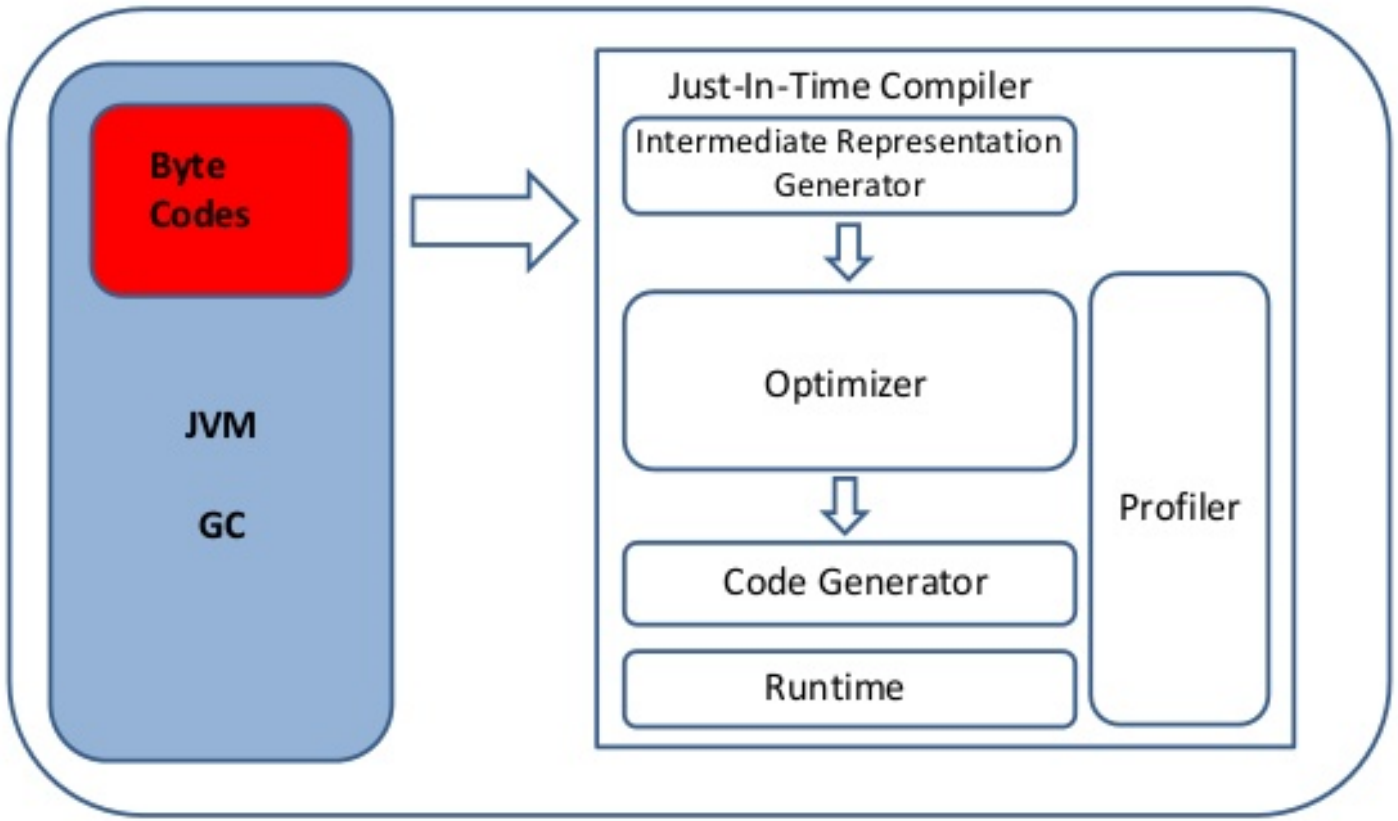


- JVM memory


Class Data
Class Data
Object Data
Object Data
t1.....tn
PC Registers
Native Method Stacks
Runtime Stacks
Method Area
Heap Area
Stack Memory
static blocks/static variables/ references
instance variables
method calls local variables

Just-In-Time (JIT)/Dynamic Compilation :

The Just-In-Time (JIT) compiler is a component of the Java Runtime Environment. It improves the performance of Java applications by compiling bytecodes to native machine code at run time.



Just-In-Time (JIT) Compiler