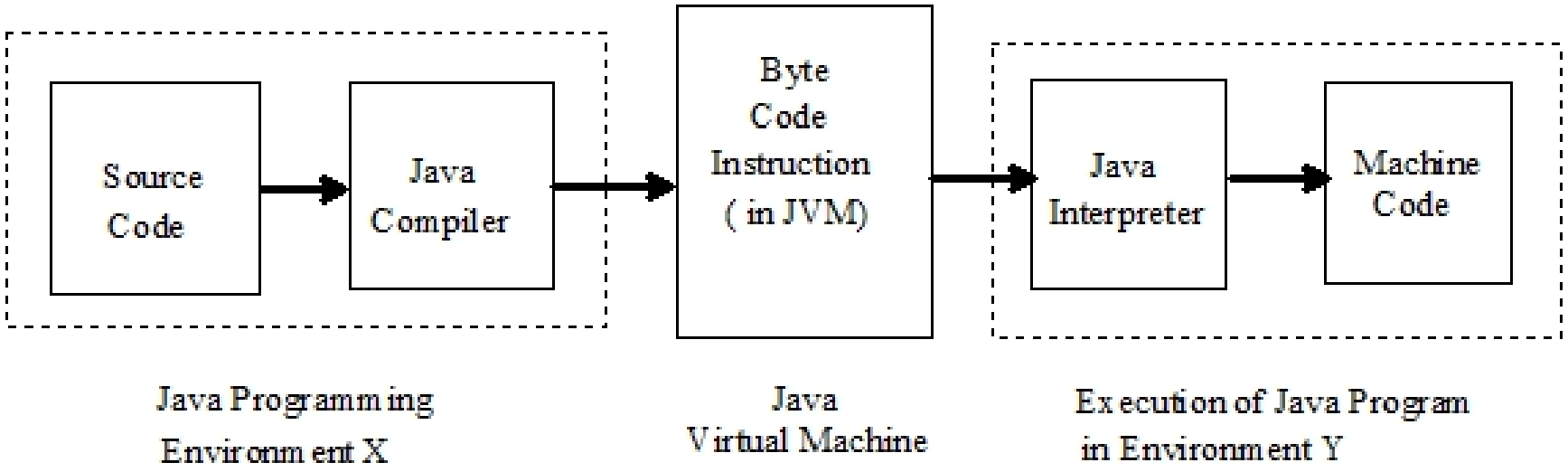
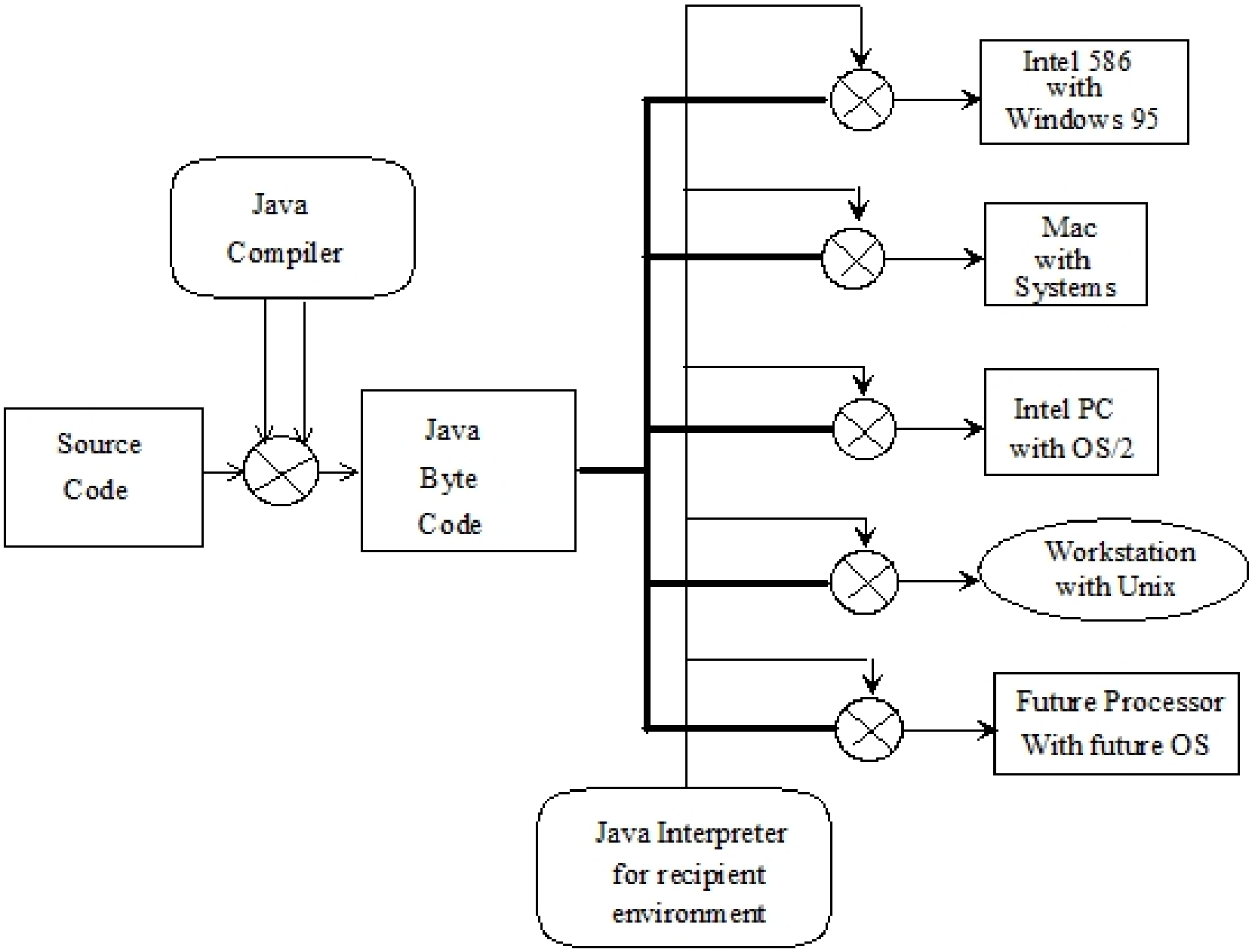
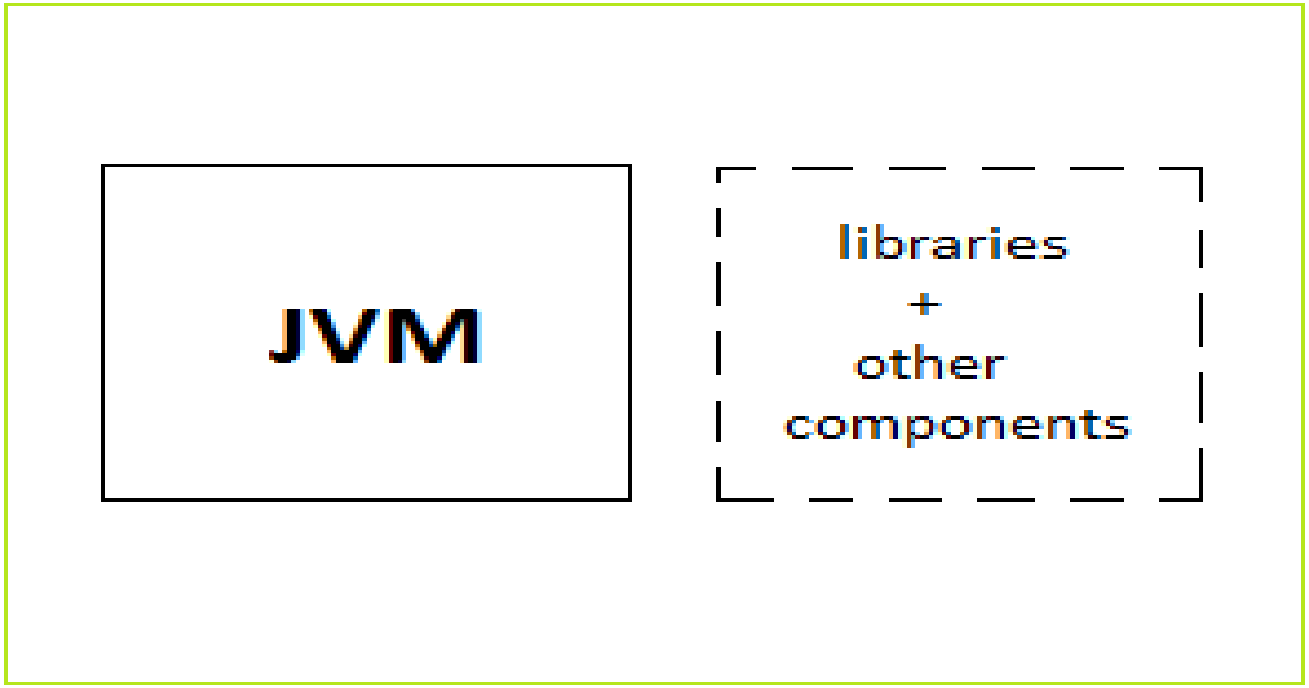
Java Execution




Java
Compiler
Source
Code
Java
Byte
Code
Intel 586
with
Windows 95
Mac
with
Systems
Intel PC
with OS/2
Workstation
with Unix
Future Processor
With future OS
Java Interpreter
for recipient
environment

JRE

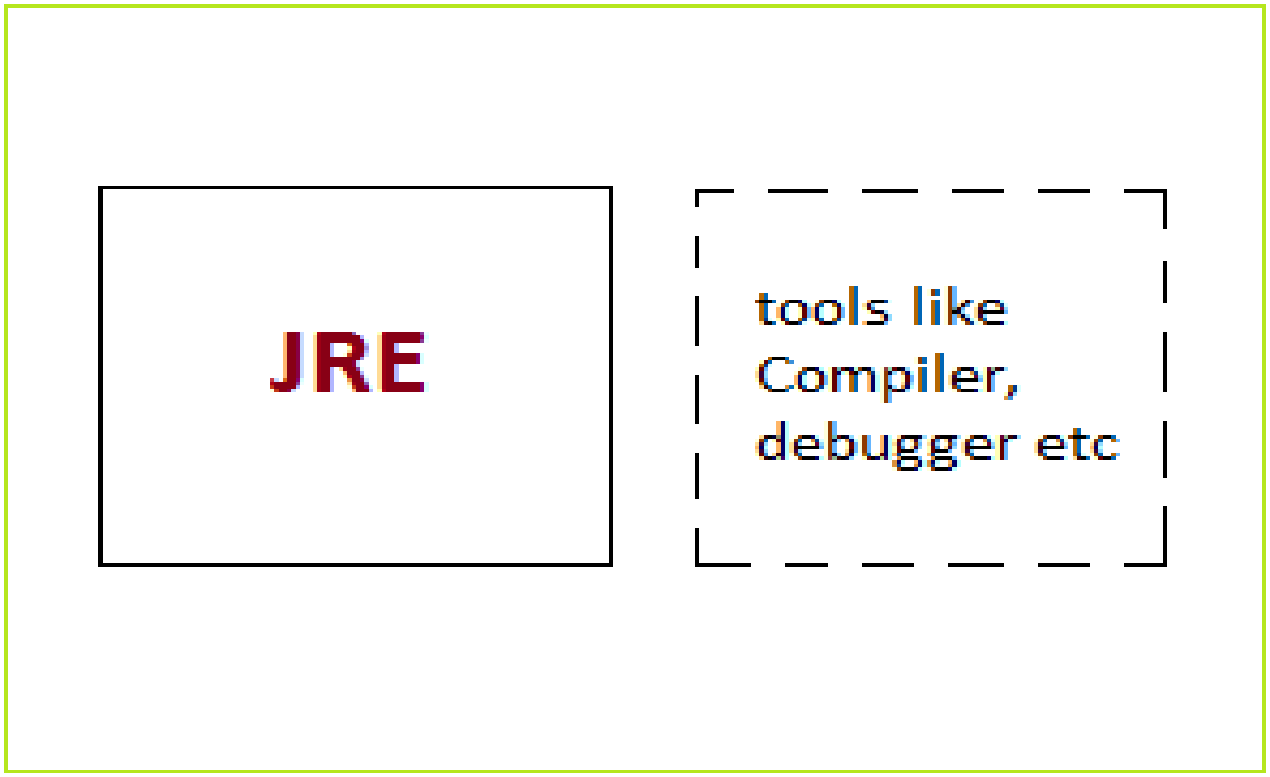
- **JRE** : The Java Runtime Environment (JRE)
- libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language.
- JRE does not contain tools and utilities such as compilers or debuggers for developing applets and applications.
-



JRE - Java Runtime Environment

JDK

- The JDK also called Java Development Kit is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications



JDK - Java Development Kit

Understand

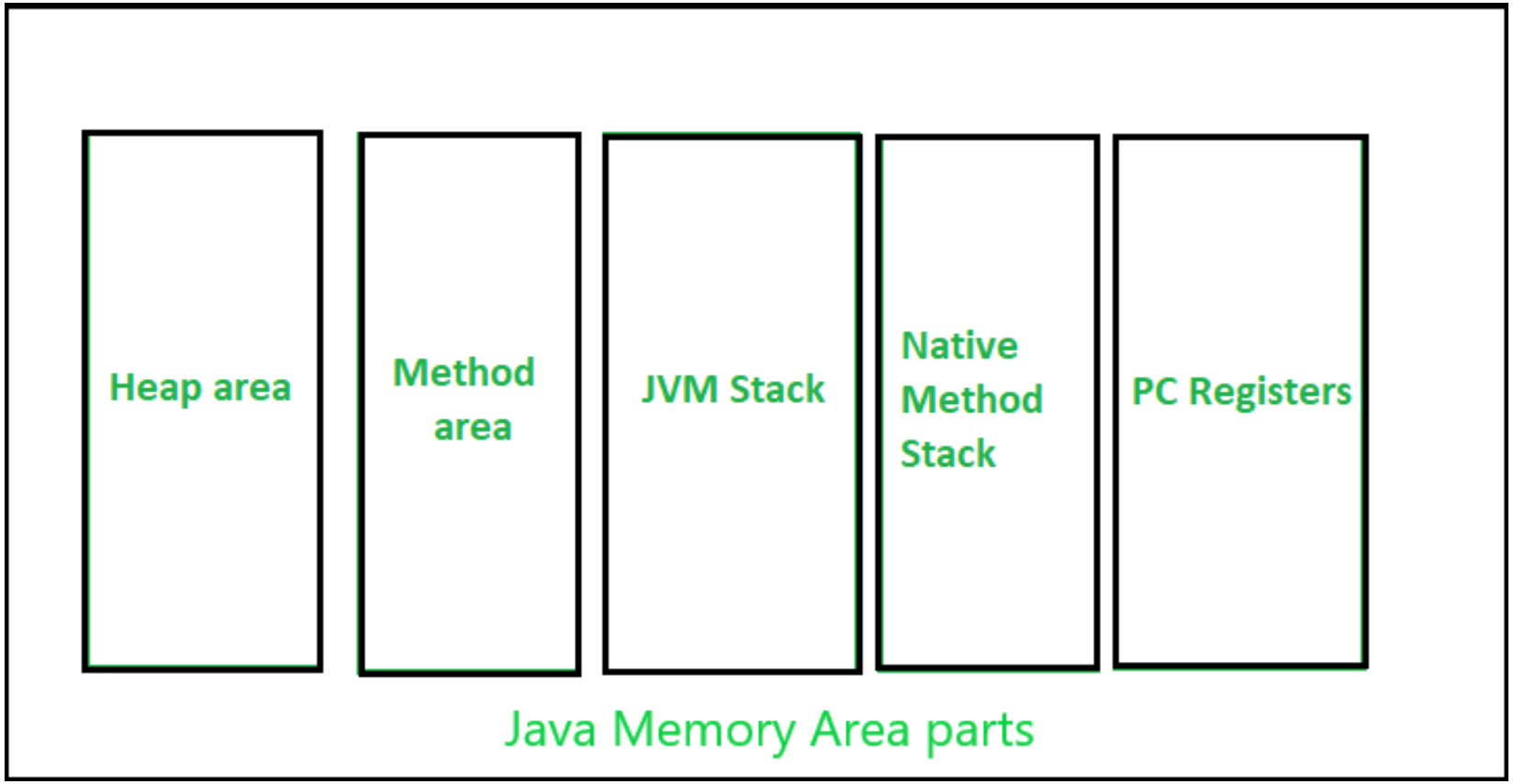
- Unlike other programming language, the JVM and to be specific Garbage Collector has the role of managing memory allocation so that the programmer needs not to.
- Whereas in other programming languages such as C the programmer has direct access to the memory who allocates memory in his code, thereby creating a lot of scope for leaks.

The major concepts in Java Memory Management :

- JVM Memory Structure
- Working of Garbage Collector
-

Java Memory Structure:

- JVM defines various run time data area which are used during execution of a program.
- Some of the areas are created by the JVM whereas some are created by the threads that are used in a program.
- However, the memory area created by JVM is destroyed only when the JVM exits.
- The data areas of thread are created during instantiation and destroyed when the thread exits.
-


Heap area
Method area
JVM Stack
Native Method Stack
PC Registers
Java Memory Area parts

- **Heap :**
- It is a shared runtime data area and stores the actual object in a memory. It is instantiated during the virtual machine startup.
- This memory is allocated for all class instances and array. Heap can be of fixed or dynamic size depending upon the system's configuration.
- JVM provides the user control to initialize or vary the size of heap as per the requirement.
- When a new keyword is used, object is assigned a space in heap, but the reference of the same exists onto the stack.
- There exists one and only one heap for a running JVM process.
- Scanner sc = new Scanner(System.in);
- The above statement creates the object of Scanner class which gets allocated to heap whereas the reference 'sc' gets pushed to the stack.
- **Note:** Garbage collection in heap area is mandatory.

- **Method Area:**
- It is a logical part of the heap area and is created on virtual machine startup.
- This memory is allocated for class structures, method data and constructor field data, and also for interfaces or special method used in class.
- Heap can be of fixed or dynamic size depending upon the system's configuration.
- Can be of a fixed size or expanded as required by the computation. Needs not to be contiguous.
- **Note:** Though method area is logically a part of heap, it may or may not be garbage collected even if garbage collection is compulsory in heap area.
-

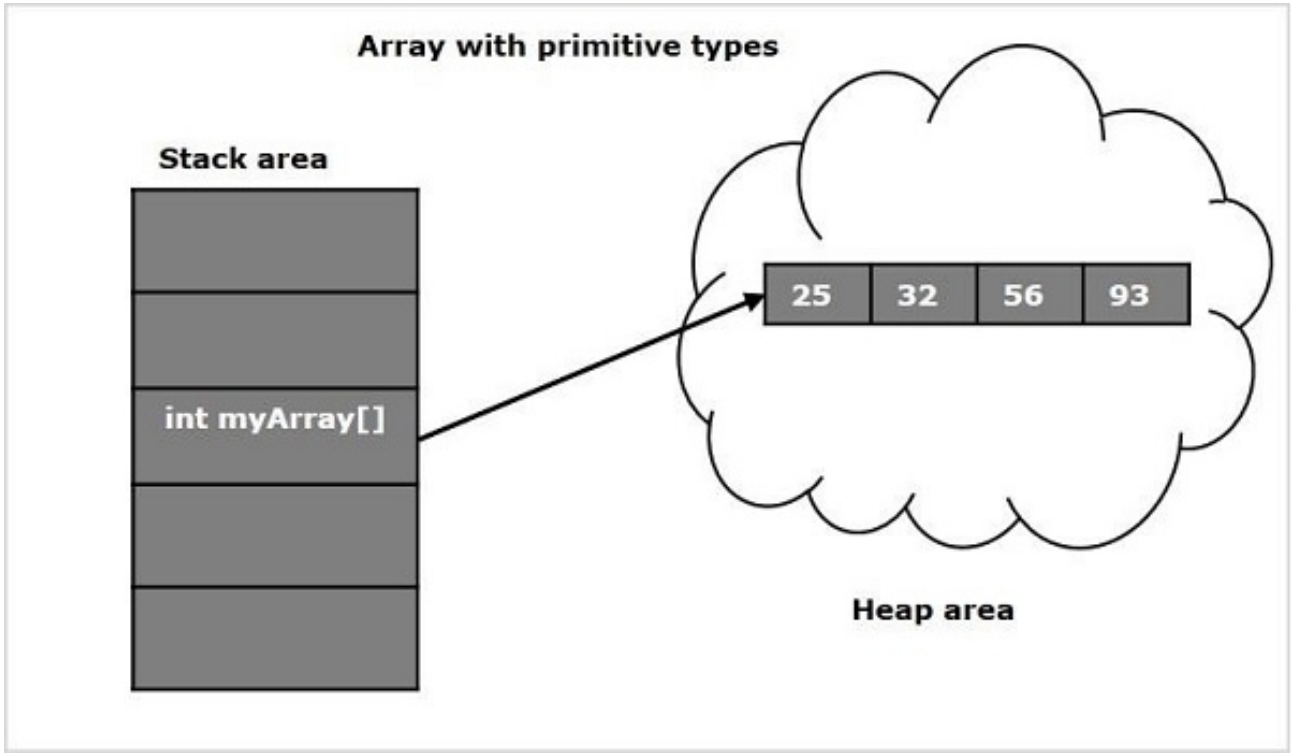
- **JVM Stacks:**
- A stack is created at the same time when a thread is created and is used to store data and partial results which will be needed while returning value for method and performing dynamic linking.
- Stacks can either be of fixed or dynamic size. The size of a stack can be chosen independently when it is created.
- The memory for stack needs not to be contiguous.
-

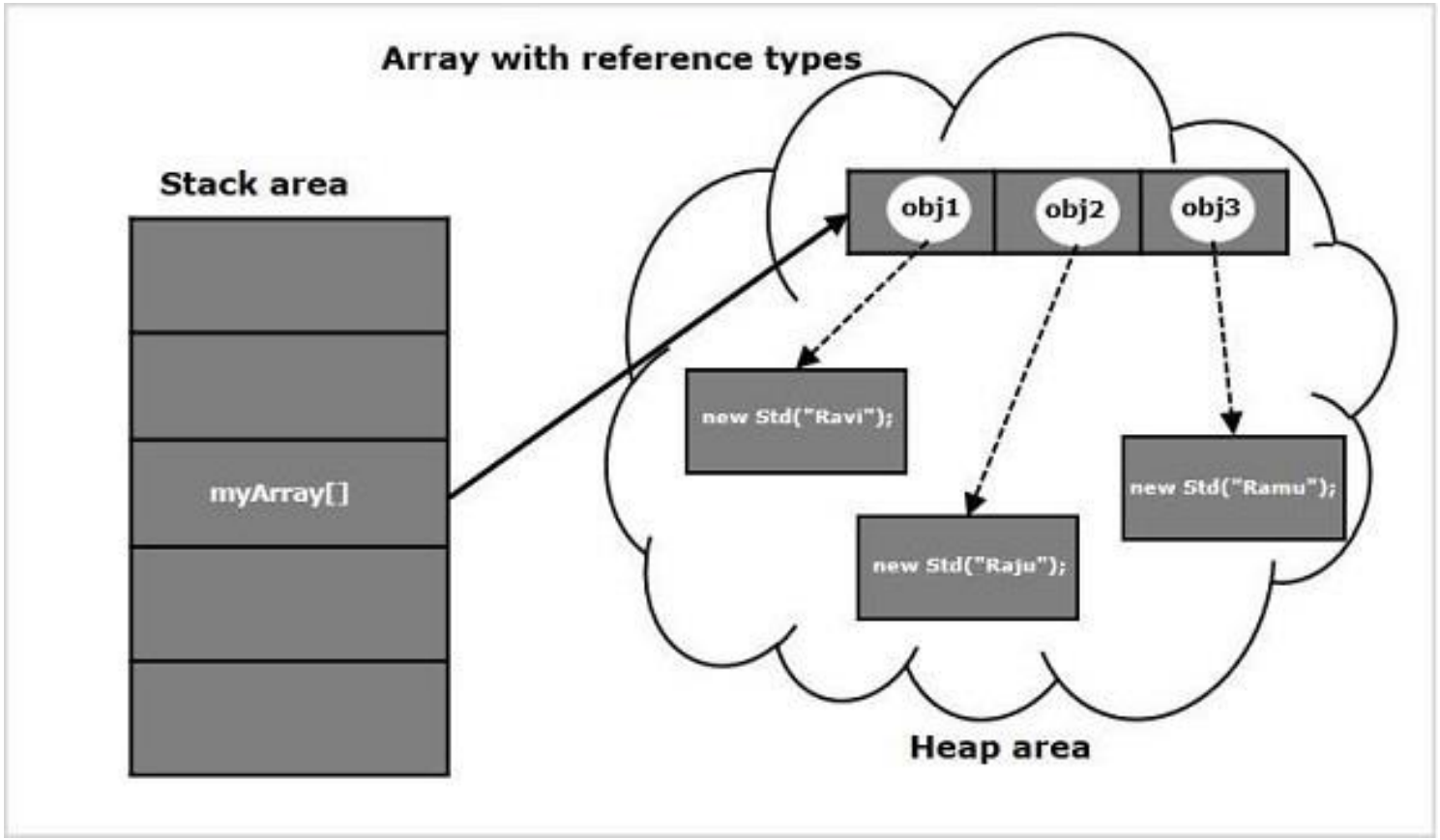
- **Program counter (PC) registers:**
- Each JVM thread which carries out the task of a specific method has a program counter register associated with it.
- The non native method has a PC which stores the address of the available JVM instruction whereas in a native method, the value of program counter is undefined.

  PC register is capable of storing the return address or a native pointer on some specific platform.
-

JVM memory locations

- JVM has five memory locations namely −
- **Heap** − Runtime storage allocation for objects (reference types).
- **Stack** − Storage for local variables and partial results. A stack contains frames and allocates one for each thread. Once a thread gets completed, this frame also gets destroyed. It also plays roles in method invocation and returns.
- **PC Registers** − Program Counter Registers contains the address of an instruction that JVM is currently executing.
- **Execution Engine** − It has a virtual processor, interpreter to interpret bytecode instructions one by one and a JIT, just in time compiler.
- **Native method stacks** − It contains all the native methods used by the application.
-


Array with primitive types
Stack area
int myArray[]
25
32
56
93
Heap area


Array with reference types
Stack area
myArray[]
obj1
obj2
obj3
new Std("Ravi");
new Std("Raju");
new Std("Ramu");
Heap area

Working of a Garbage Collector:

- JVM triggers this process and as per the JVM garbage collection process is done or else withheld. It reduces the burden of programmer by automatically performing the allocation or deallocation of memory.
- Garbage collection process causes the rest of the processes or threads to be paused and thus is costly in nature. This problem is unacceptable for the client but can be eliminated by applying several garbage collector based algorithms. This process of applying algorithm is often termed as **Garbage Collector tuning** and is important for improving the performance of a program.
- Another solution is the generational garbage collectors that adds an age field to the objects that are assigned a memory. As more and more objects are created, the list of garbage grows thereby increasing the garbage collection time. On the basis of how many clock cycles the objects have survived, objects are grouped and are allocated an ‘age’ accordingly. This way the garbage collection work gets distributed.
- In the current scenario, all garbage collectors are generational, and hence, optimal.
- Knowing how the program and it’s data is stored or organized is essential as it helps when the p

- **Note: System.gc()** and **Runtime.gc()** are the methods which requests for Garbage collection to JVM explicitly but it doesn’t ensures garbage collection as the final decision of garbage collection is of JVM only.