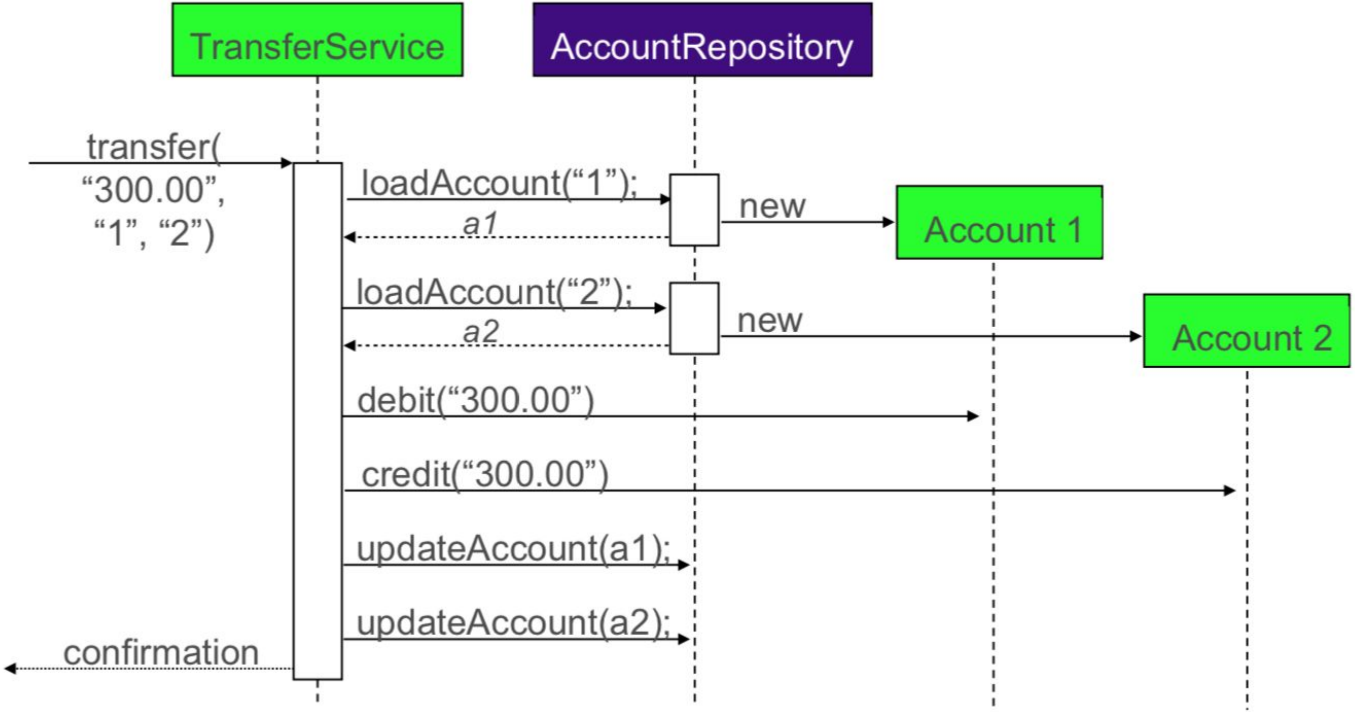
JVM

Introduction

Ex.


TransferService
AccountRepository
transfer("300.00", "1", "2")
loadAccount("1");
a1
new
Account 1
loadAccount("2");
a2
new
Account 2
debit("300.00")
credit("300.00")
updateAccount(a1);
updateAccount(a2);
confirmation

JVM Language(s)

Apart from the Java language, the most common or well-known other JVM languages are:

- Clojure, a modern, dynamic, and functional dialect of the Lisp programming language
- Groovy, a dynamic programming and scripting language
- JRuby, an implementation of Ruby
- Jython, an implementation of Python
- Kotlin, a statically-typed language from JetBrains, the developers of IntelliJ IDEA
- Scala, a statically-typed object-oriented and functional programming language

JVM specification

It is a specification that provides a runtime environment in which Java bytecode can be executed.

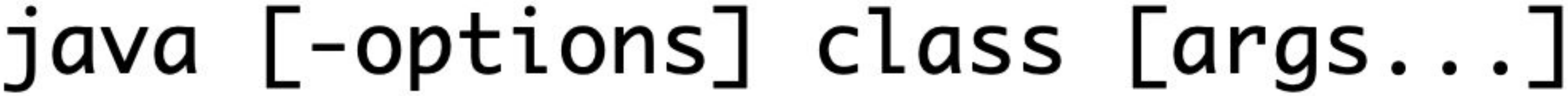
It can also run those programs which are written in other languages and compiled to Java bytecode.

https://docs.oracle.com/javase/specs/jvms/se7/html/

JVM Implementation

https://en.wikipedia.org/wiki/List_of_Java_virtual_machines

JVM instance



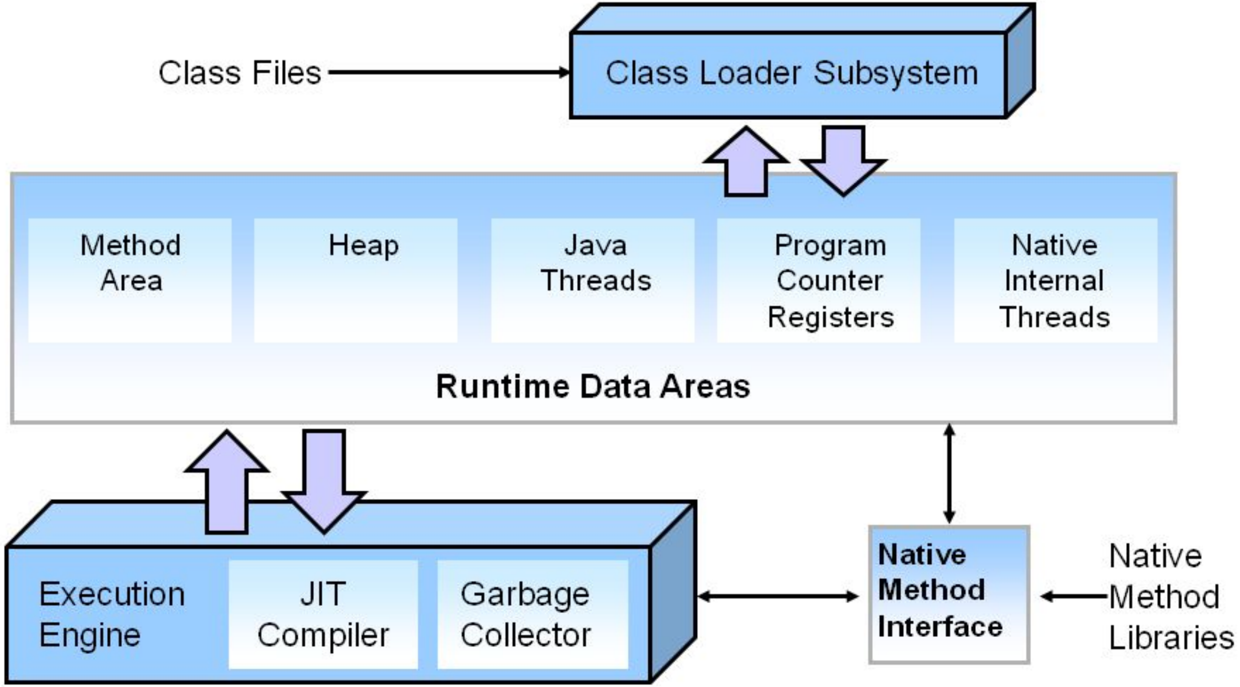
```
java [-options] class [args...]
```

JVM

The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

HotSpot JVM: Architecture



Class Files

Class Loader Subsystem

Method Area

Heap

Java Threads

Program Counter Registers

Native Internal Threads

Runtime Data Areas

Execution Engine

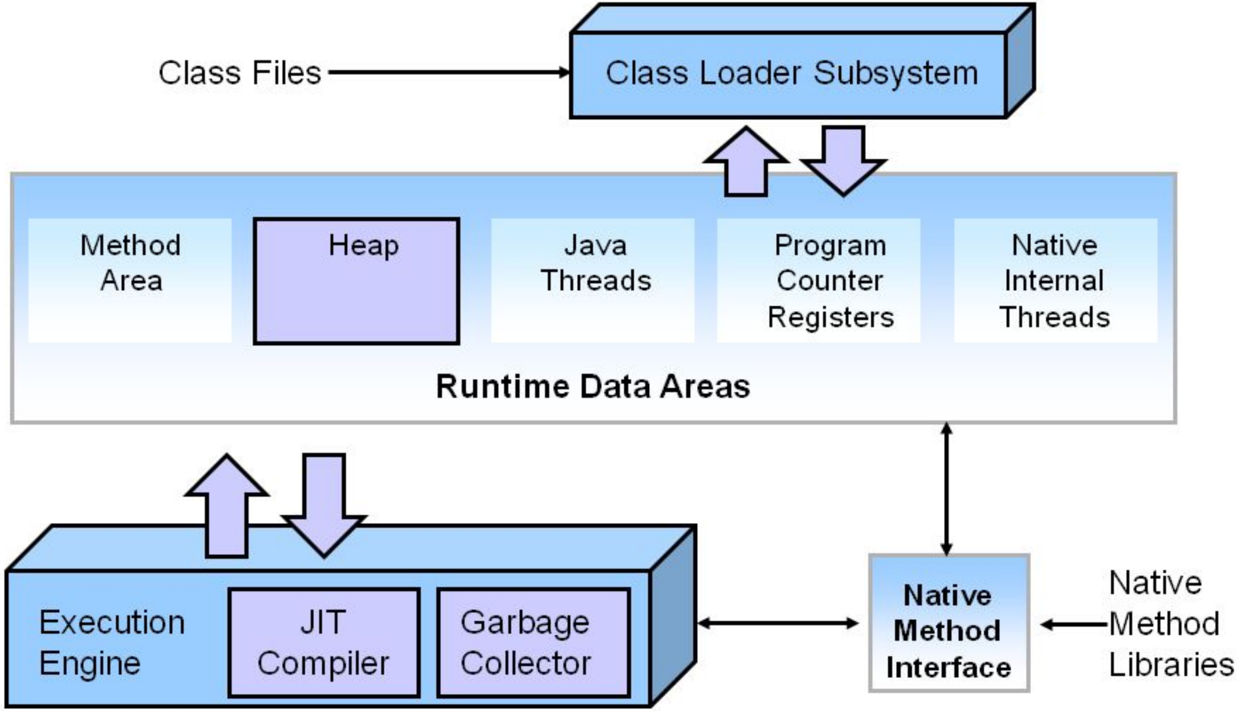
JIT Compiler

Garbage Collector

Native Method Interface

Native Method Libraries

Key HotSpot JVM Components

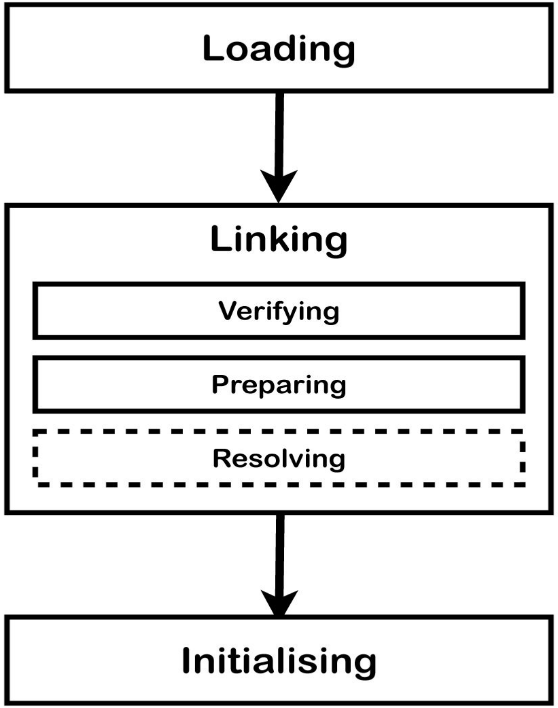


Class Loader

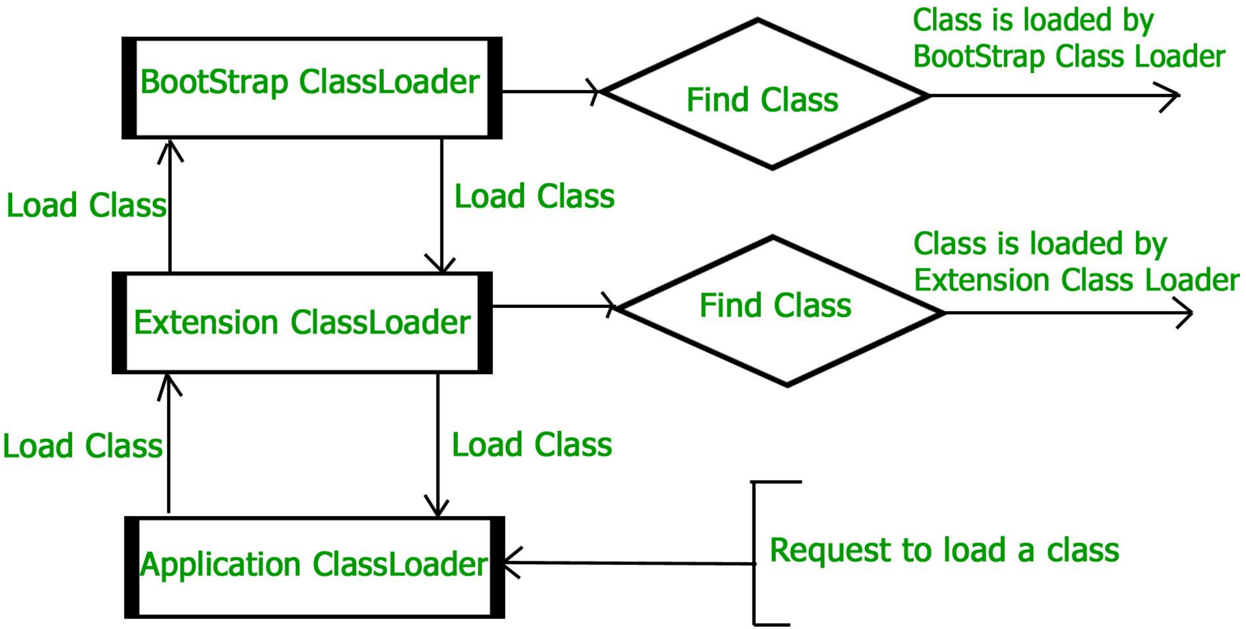
It is mainly responsible for three activities.

- Loading
- Linking
- Initialization

Class Loader



Class Loader



JVM memory : Method area

Method area : In method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables.

There is only one method area per JVM, and it is a shared resource.

Metaspace (JDK 1.8)

JVM memory : **Heap area**

Heap area : Information of all objects is stored in heap area.

There is also one Heap Area per JVM.

It is also a shared resource.

JVM memory : **Stack area**

Stack area : For every thread, JVM create one run-time stack which is stored here.

Every block of this stack is called activation record/stack frame which store methods calls.

All local variables of that method are stored in their corresponding frame.

After a thread terminate, it’s run-time stack will be destroyed by JVM.

It is not a shared resource.

JVM memory : **PC Registers**

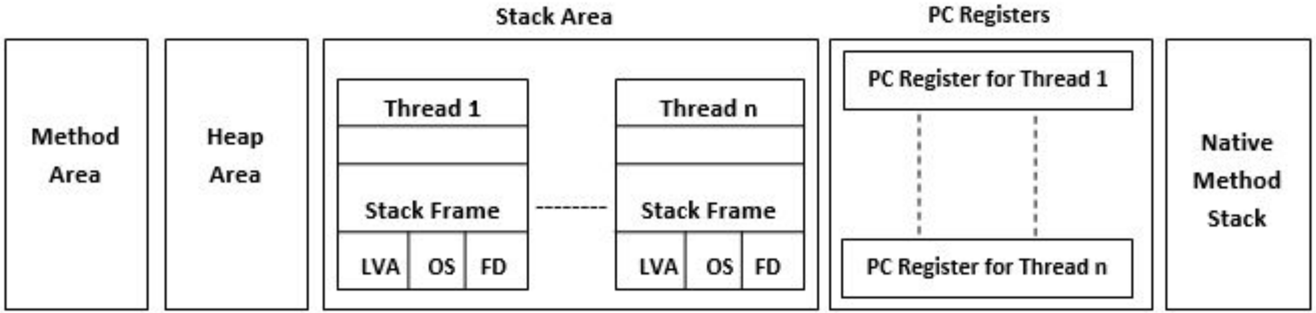
PC Registers : Store address of current execution instruction of a thread.

Obviously each thread has separate PC Registers.

JVM memory : **Native method stacks**

Native method stacks : For every thread, separate native stack is created.

It stores native method information.


Stack Area
PC Registers
Method Area
Heap Area
Thread 1
Stack Frame
LVA
OS
FD
Thread n
Stack Frame
LVA
OS
FD
PC Register for Thread 1
PC Register for Thread n
Native Method Stack

Execution Engine

Execution engine execute the *.class* (bytecode).

It reads the byte-code line by line, use data and information present in various memory areas and execute instructions.

Execution Engine Interpreter

A JVM interpreter pretty much converts each byte-code instruction to corresponding native instruction by looking up a predefined JVM-instruction to machine instruction mapping.

It directly executes the bytecode and does not perform any optimization.

Execution Engine JIT

To improve performance, JIT compilers interact with the JVM at runtime and compile appropriate bytecode sequences into native machine code.

Typically, JIT compiler takes a block of code (not one statement at a time as interpreter), optimize the code and then translate it to optimized machine code.

Execution Engine GC

It destroy unreferenced objects.

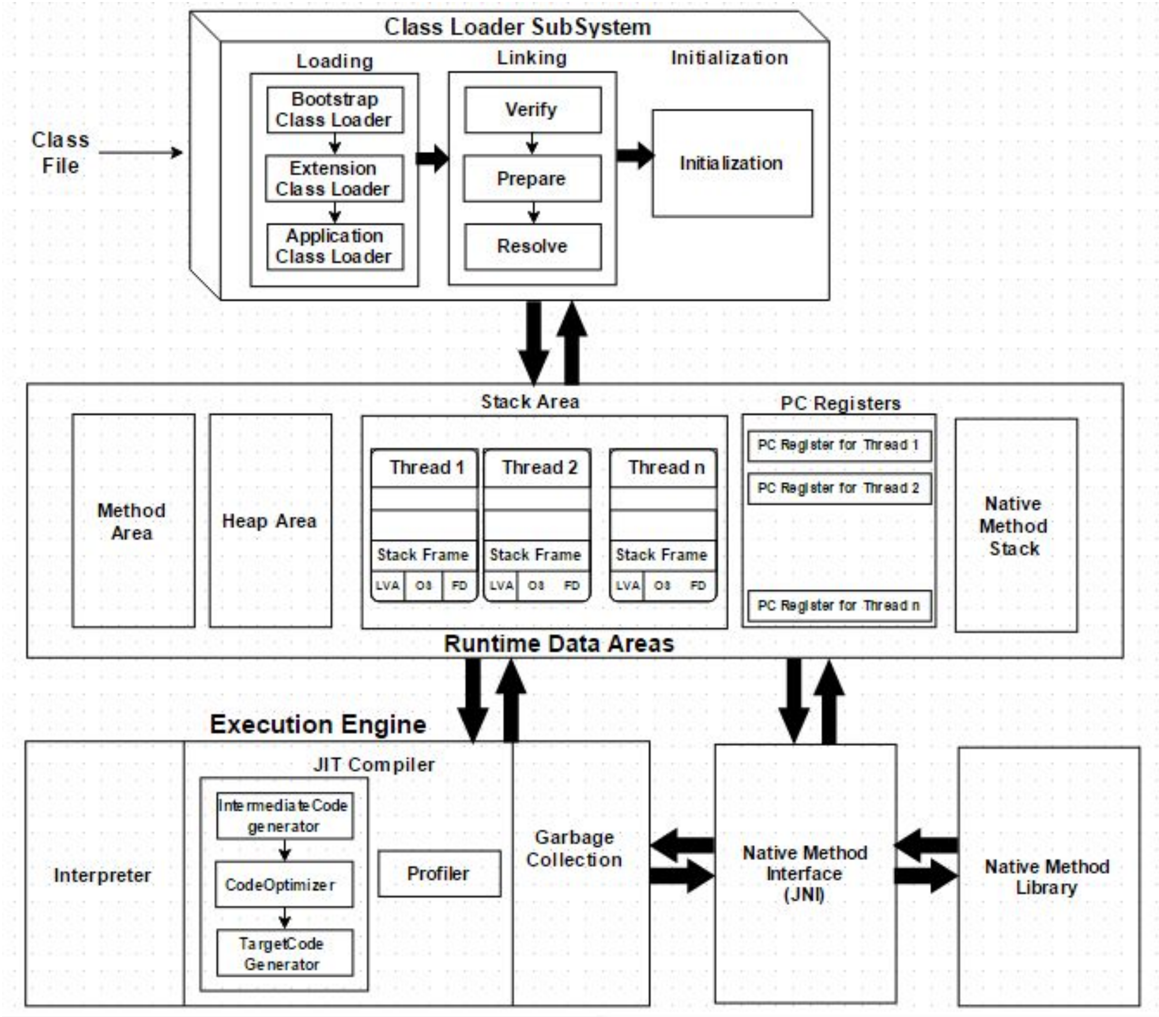
JNI

It is an interface which interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution.

It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.

Native Method Libraries

It is a collection of the Native Libraries(C, C++) which are required by the Execution Engine.


Class Loader SubSystem
Class File
Loading
Bootstrap Class Loader
Extension Class Loader
Application Class Loader
Linking
Verify
Prepare
Resolve
Initialization
Initialization
Method Area
Heap Area
Stack Area
Thread 1
Thread 2
Thread n
Stack Frame
LVA
OS
FD
PC Registers
PC Register for Thread 1
PC Register for Thread 2
PC Register for Thread n
Native Method Stack
Runtime Data Areas
Execution Engine
Interpreter
JIT Compiler
Intermediate Code generator
CodeOptimizer
TargetCode Generator
Profiler
Garbage Collection
Native Method Interface (JNI)
Native Method Library

JRE

JRE bundles the following components –

1. DLL files used by the Java HotSpot Client Virtual Machine.
2. DLL files used by the Java HotSpot Server Virtual Machine.
3. Code libraries, property settings, and resource files used by the Java runtime environment. e.g. rt.jar and charsets.jar.
4. Java extension files such as localedata.jar.
5. Contains files used for security management. These include the security policy (java.policy) and security properties (java.security) files.
6. Jar files containing support classes for applets.
7. Contains TrueType font files for use by the platform.

JRE

JREs can be downloaded as part of JDKs or you can download them separately.

JREs are platform dependent.

It means that based on the type of machine (OS and architecture), you will have to select the JRE bundle to import and install.

For example, you cannot install a 64-bit JRE distribution on 32-bit machine. Similarly, JRE distribution for *Windows* will not work in *Linux*; and vice-versa.

JDK

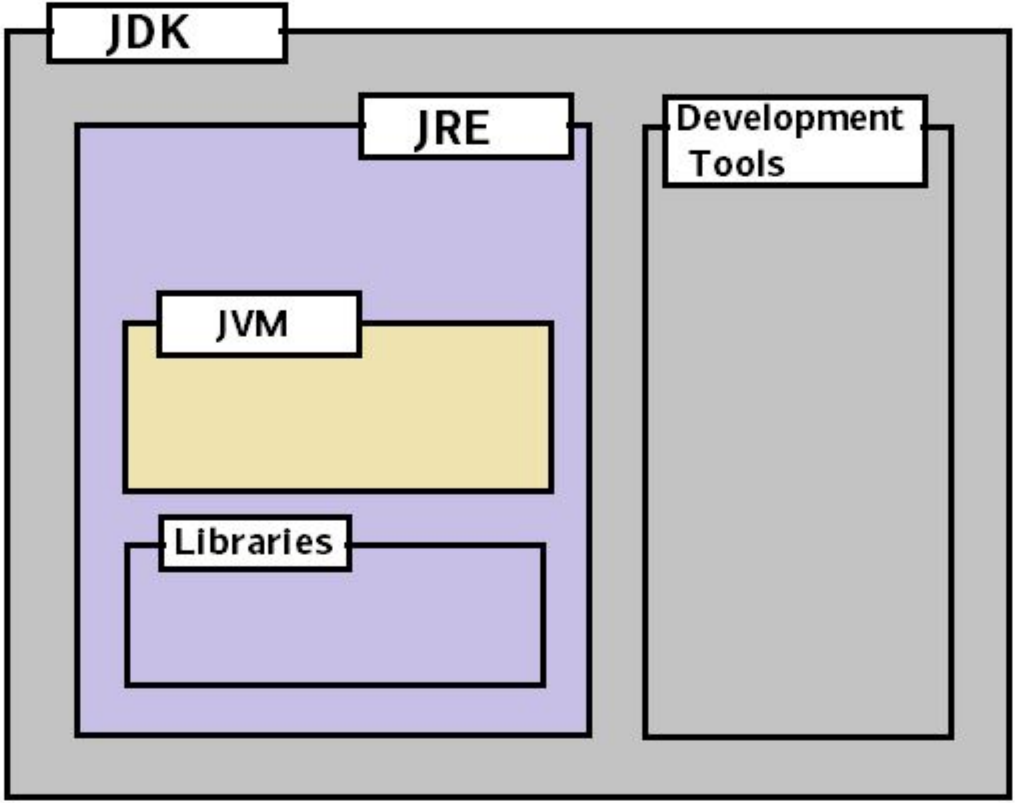
Few important components shipped with JDKs are as follows:

- **apt** – the annotation-processing tool
- **extcheck** – a utility that detects JAR file conflicts
- **javadoc** – the documentation generator, which automatically generates documentation from source code comments
- **jar** – the archiver, which packages related class libraries into a single JAR file. This tool also helps manage JAR files
- **jarsigner** – the jar signing and verification tool
- **javap** – the class file disassembler
- **javaws** – the Java Web Start launcher for JNLP applications
- **JConsole** – Java Monitoring and Management Console
- **jhat** – Java Heap Analysis Tool
- **jrunscript** – Java command-line script shell
- **jstack** – utility that prints Java stack traces of Java threads
- **keytool** – tool for manipulating the keystore
- **policytool** – the policy creation and management tool
- **xjc** – Part of the Java API for XML Binding (JAXB) API. It accepts an XML schema and generates

JDK, JRE & JVM

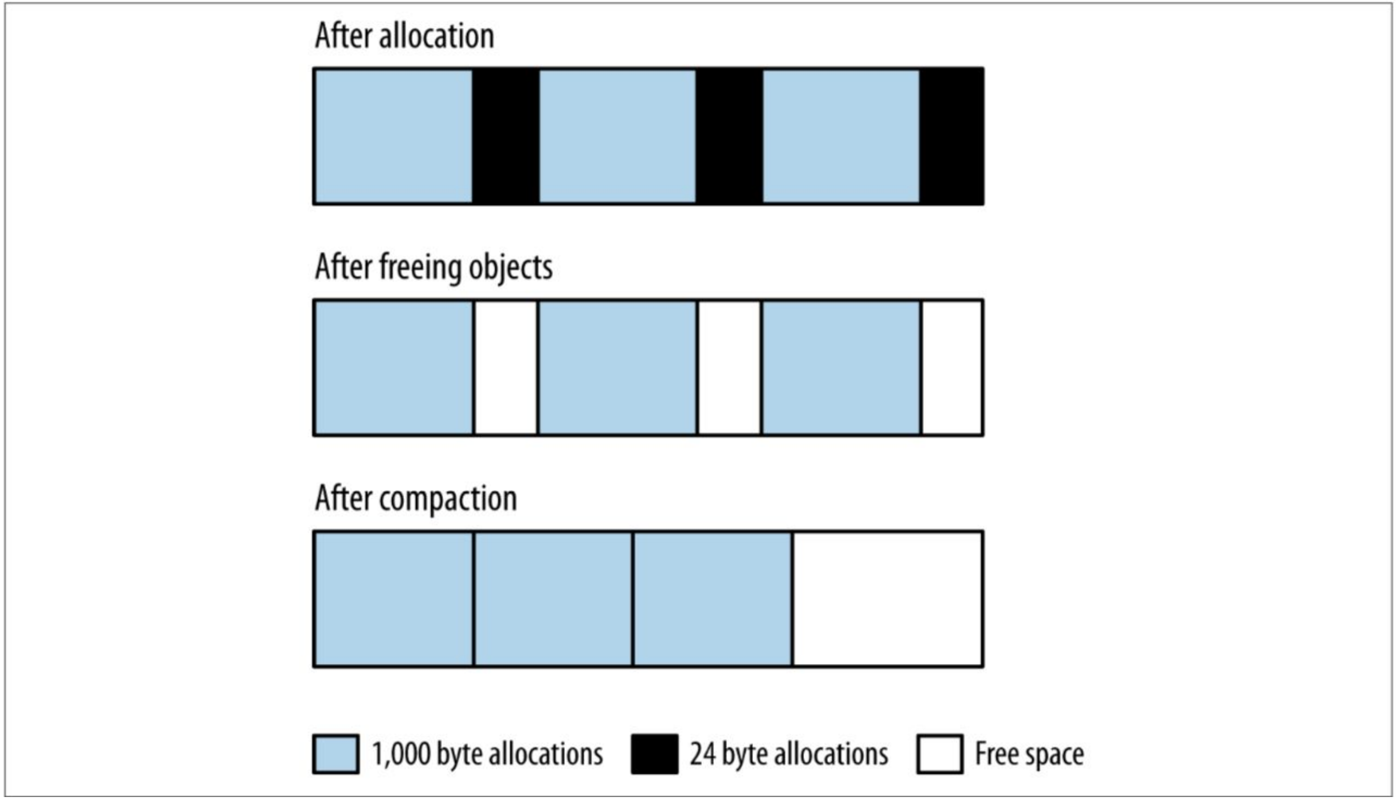
JRE = JVM + libraries to run Java application.

JDK = JRE + tools to develop Java Application.



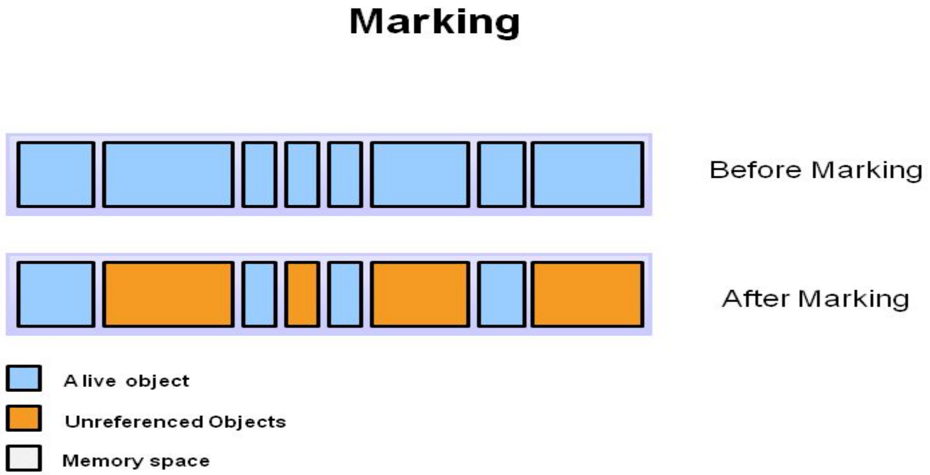
JVM - GC

Idealized GC heap during collection

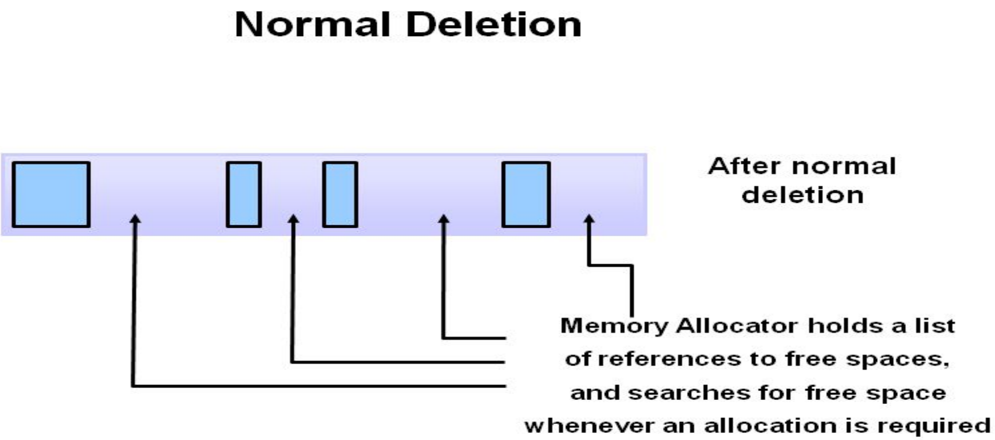


"Mark and Sweep" process

GC - step-1


Marking
Before Marking
After Marking
A live object
Unreferenced Objects
Memory space

GC - step-2



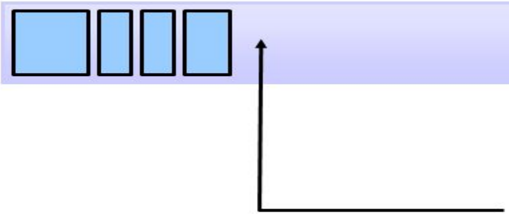
Normal Deletion

After normal deletion

Memory Allocator holds a list of references to free spaces, and searches for free space whenever an allocation is required

GC - step-2a

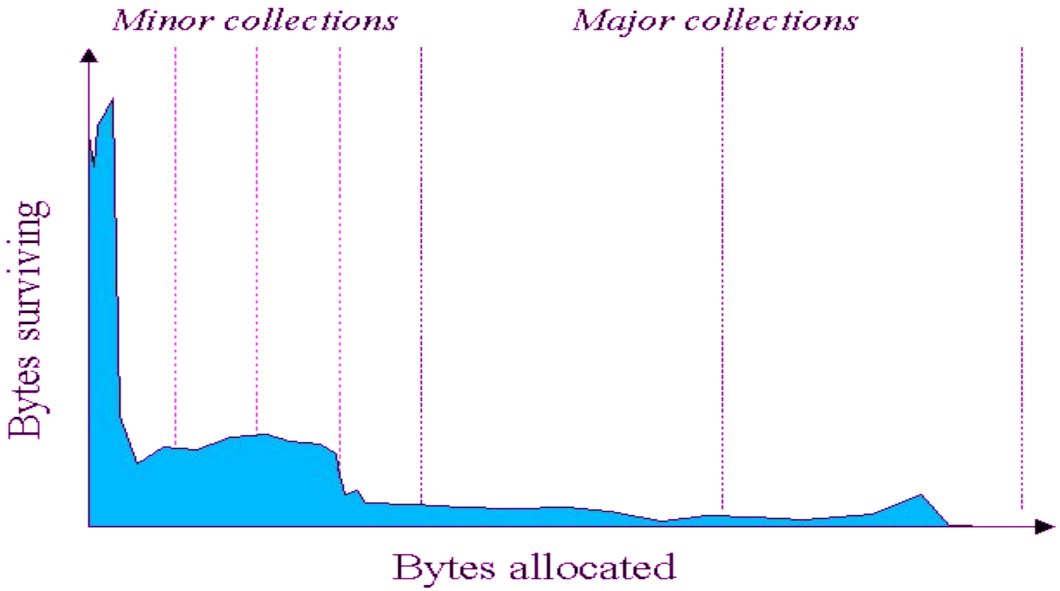
Deletion with **Compacting**



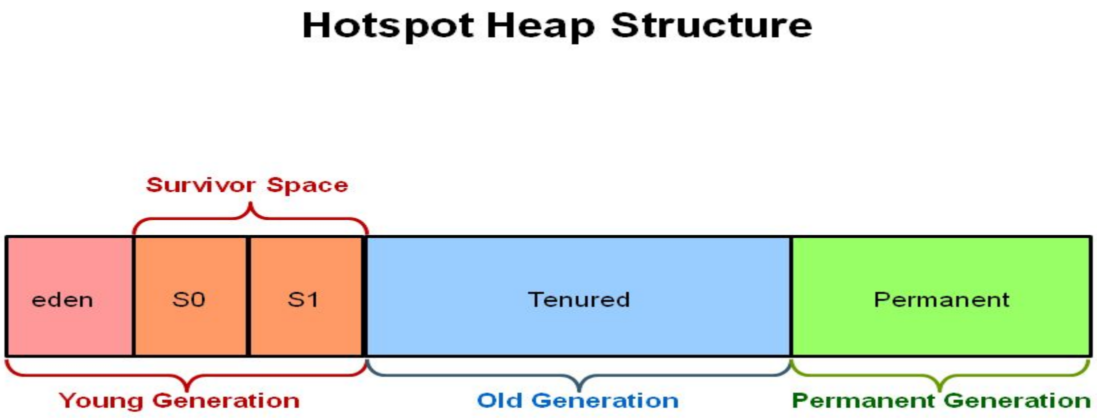
After normal Deletion with compacting

Memory Allocator holds the reference to the beginning of free space, and allocated memory sequentially then on.

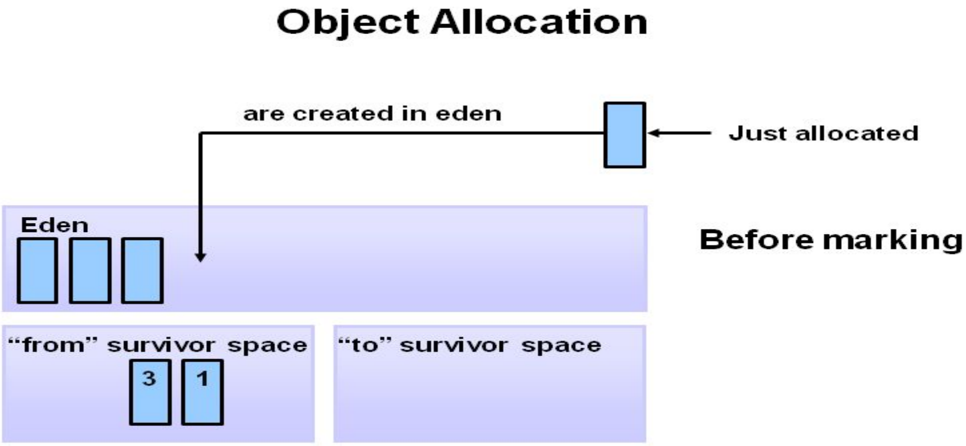
Why Generational Garbage Collection?



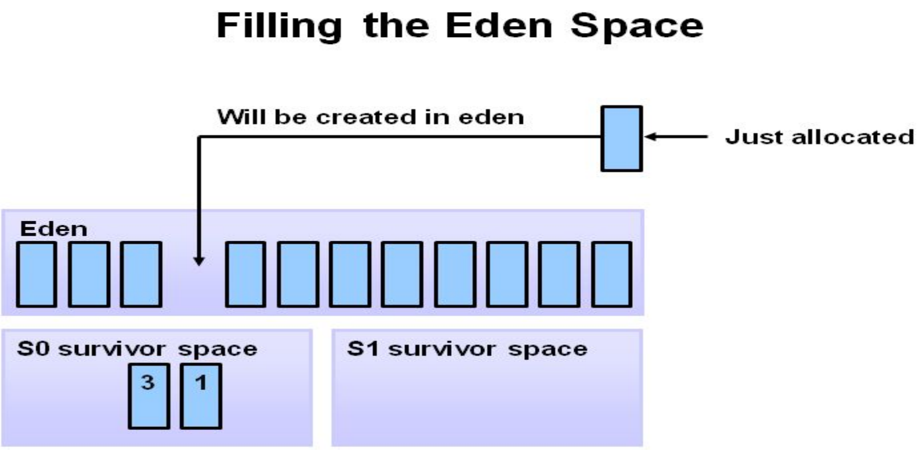
JVM Generations



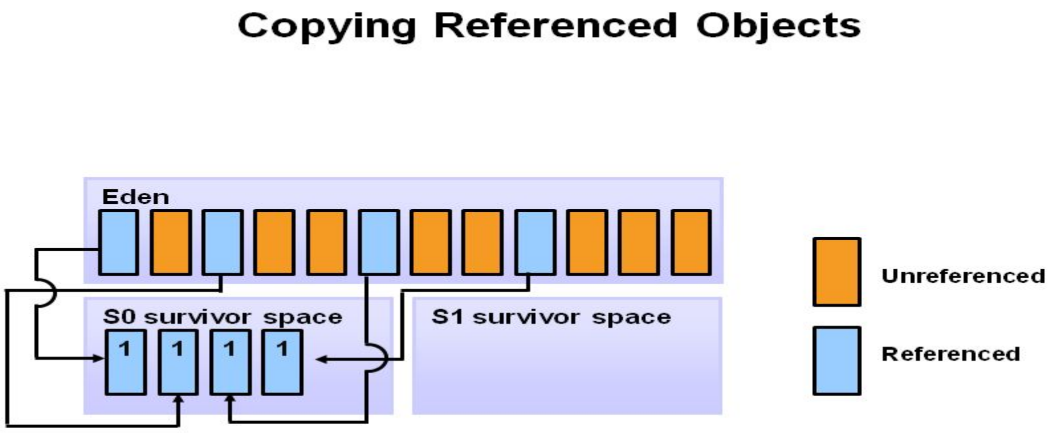
The Generational Garbage Collection Process



The Generational Garbage Collection Process

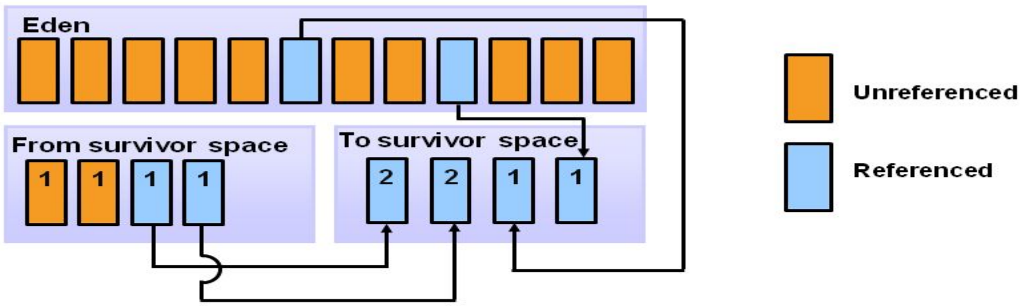


The Generational Garbage Collection Process

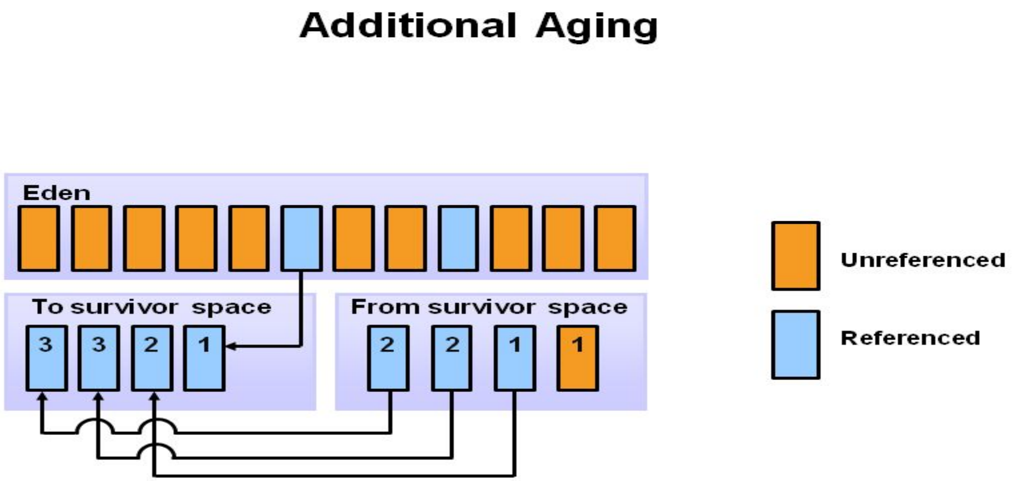


The Generational Garbage Collection Process

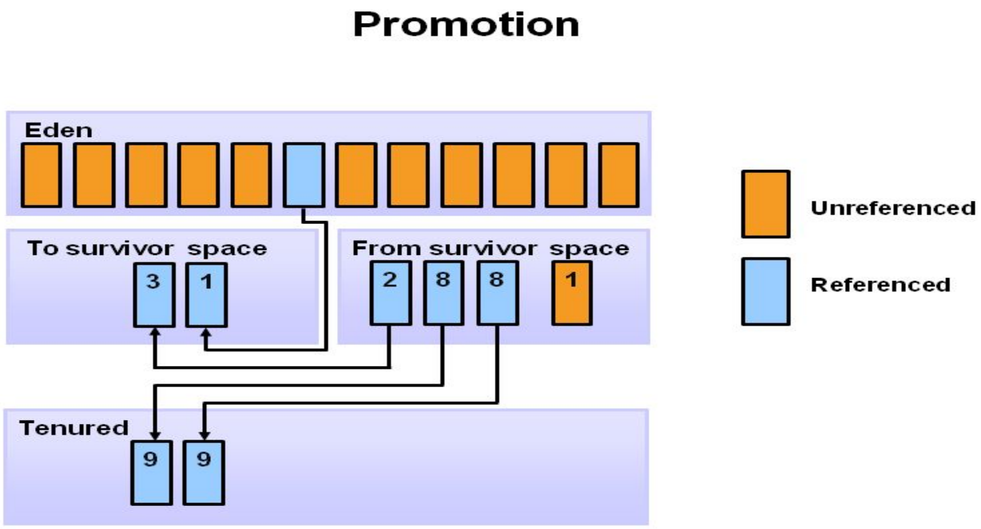
Object Aging



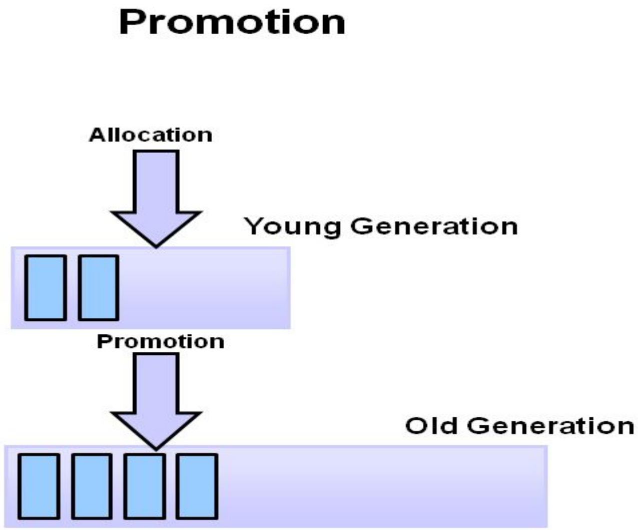
The Generational Garbage Collection Process



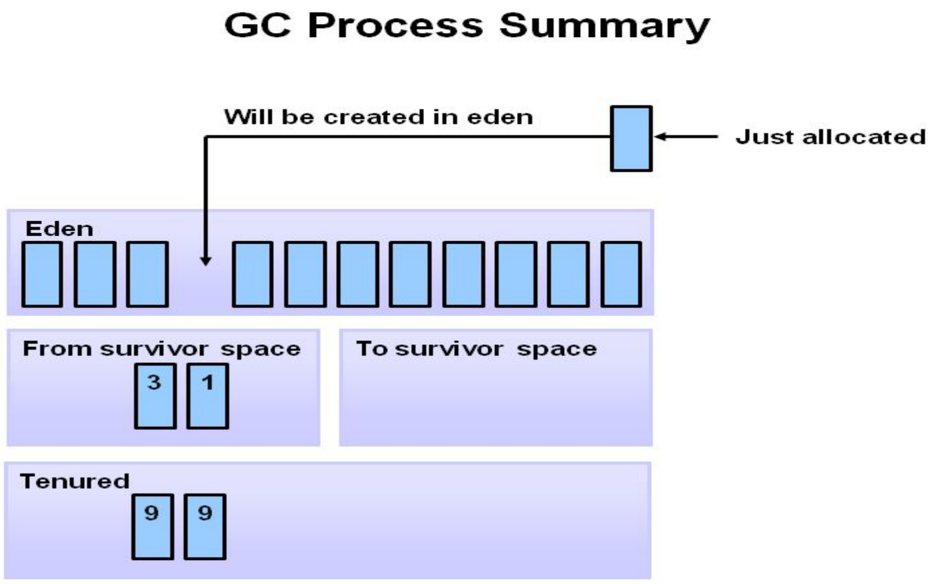
The Generational Garbage Collection Process



The Generational Garbage Collection Process



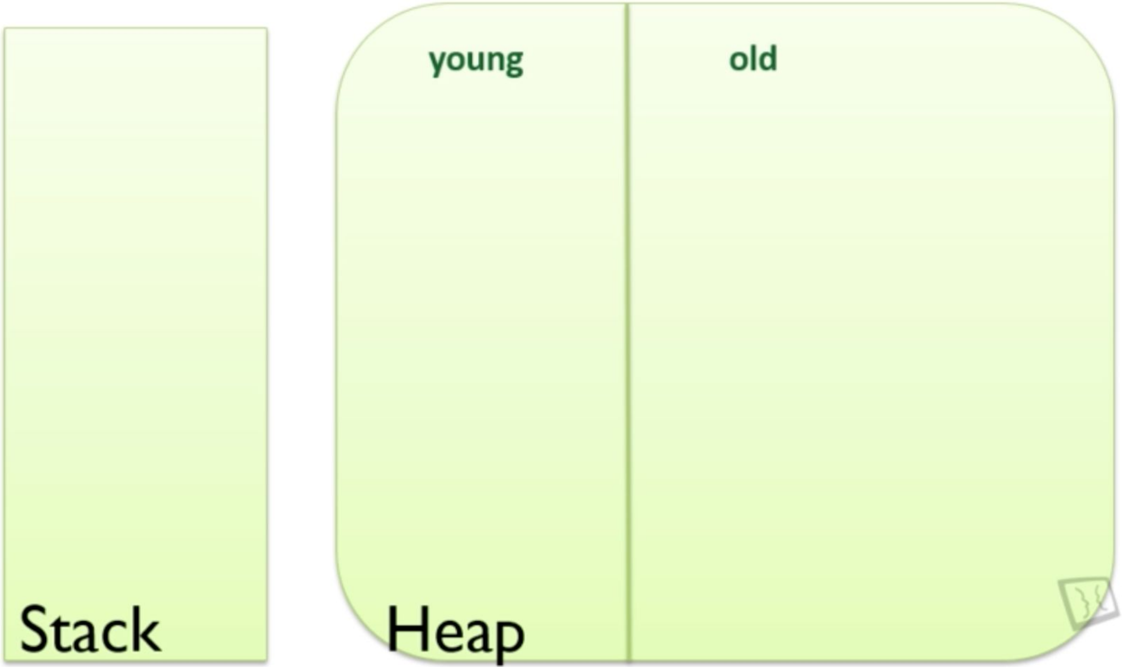
The Generational Garbage Collection Process



Generational Garbage Collection

Most objects don't live for long

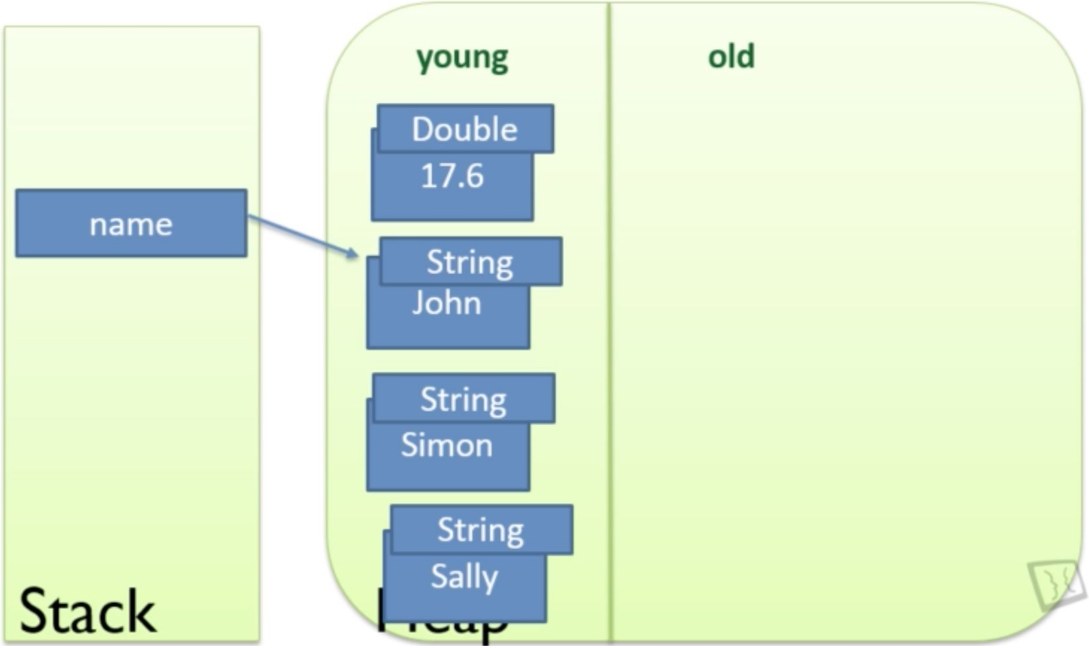
If an object survives it is likely to live forever



Generational Garbage Collection

Most objects don't live for long

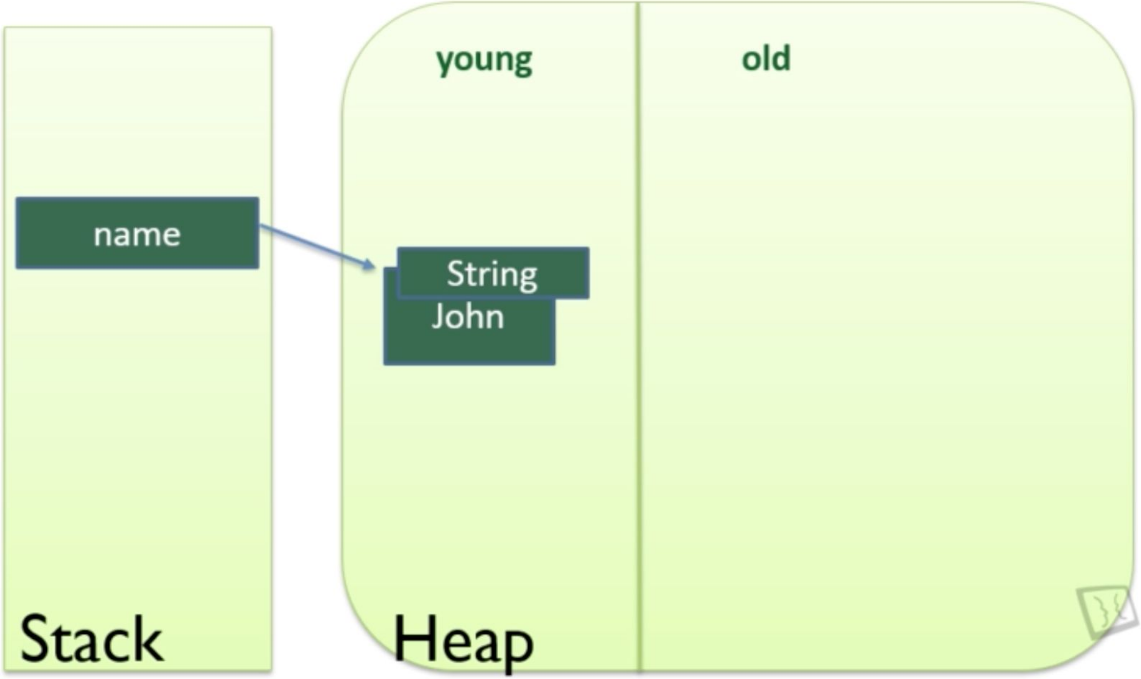
If an object survives it is likely to live forever



Generational Garbage Collection

Most objects don't live for long

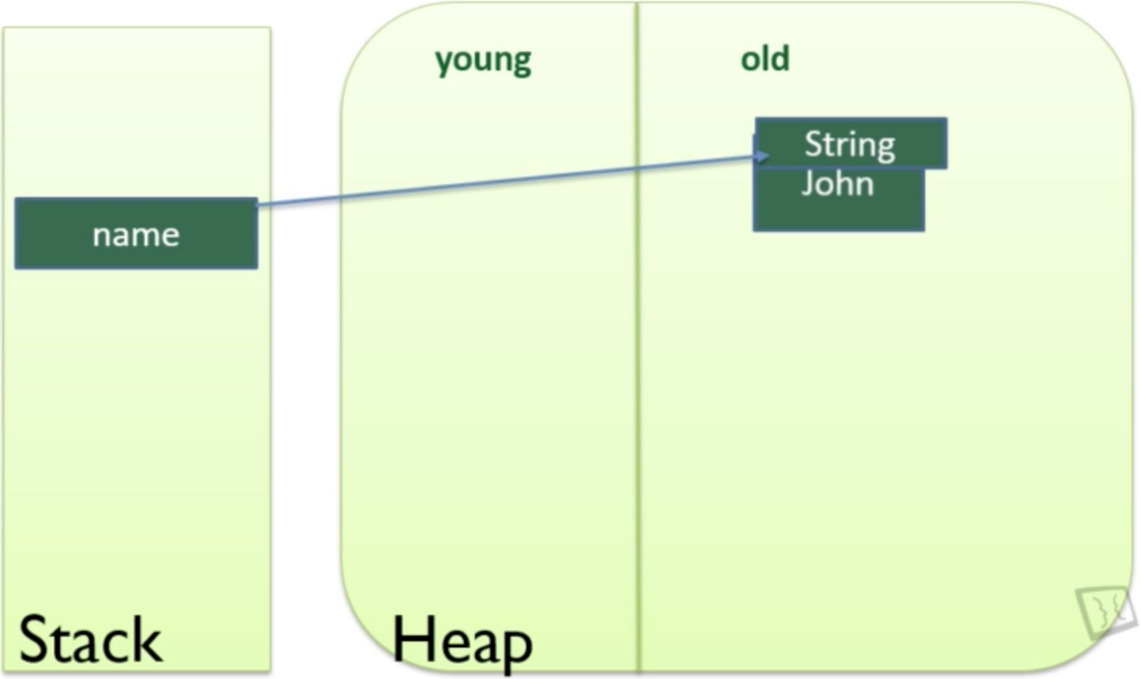
If an object survives it is likely to live forever



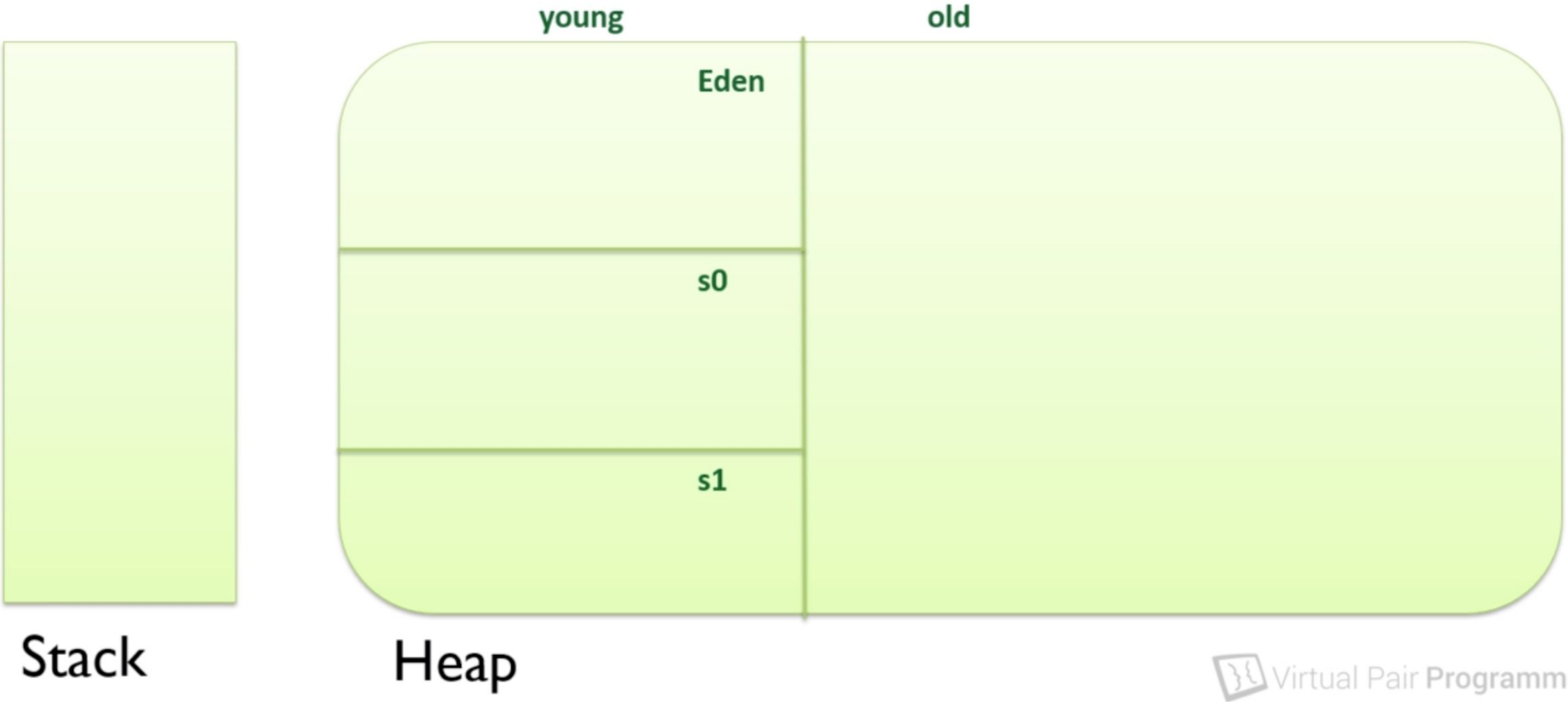
Generational Garbage Collection

Most objects don't live for long

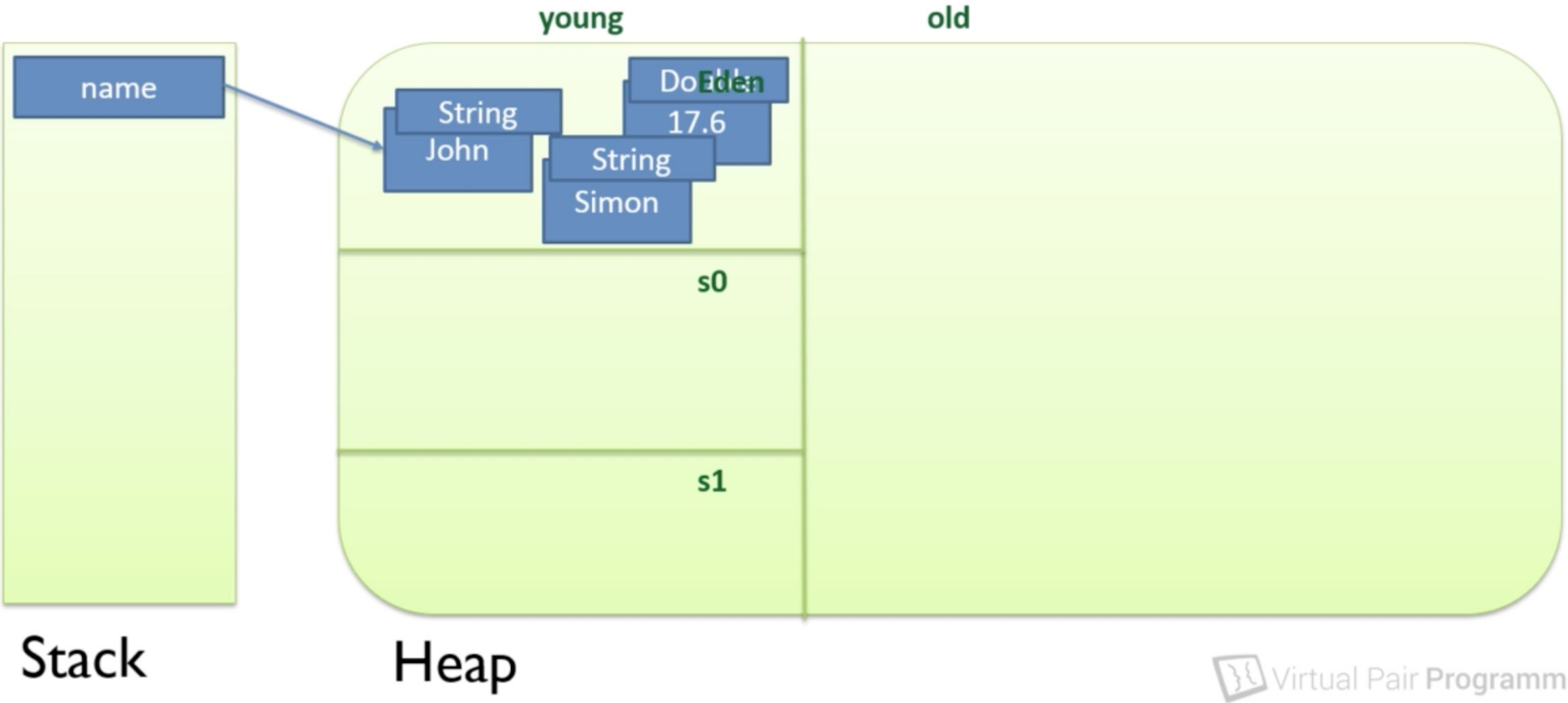
If an object survives it is likely to live forever



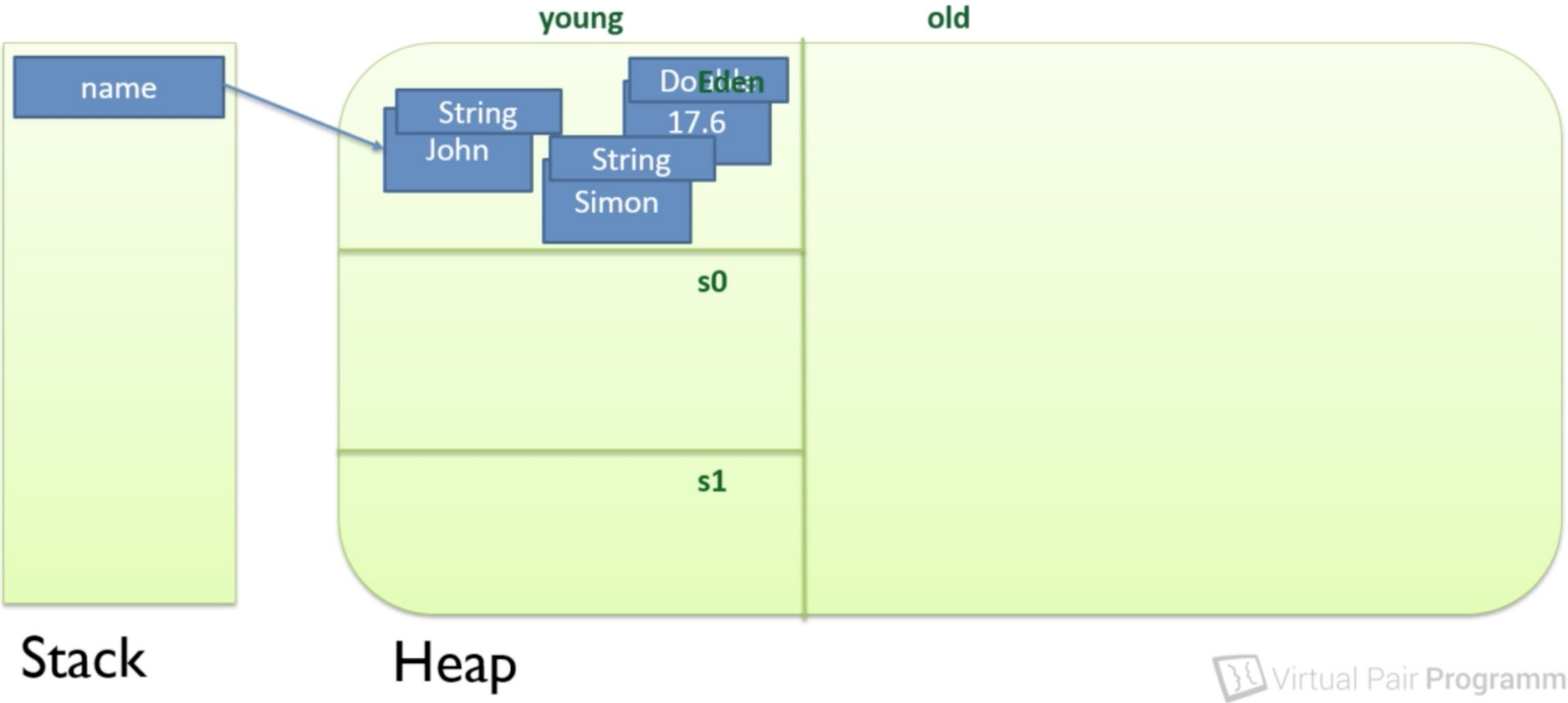
Young generation – survivor spaces



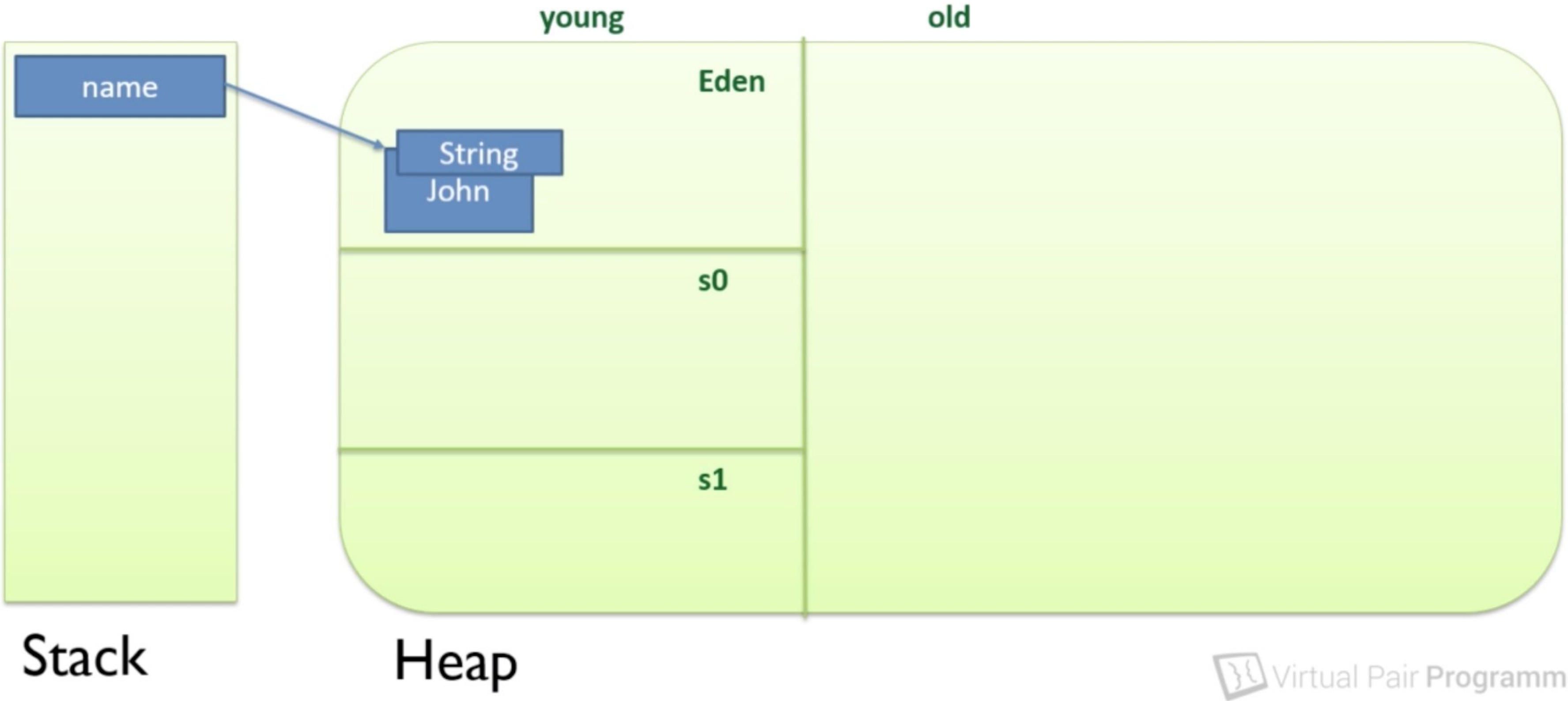
Young generation – survivor spaces



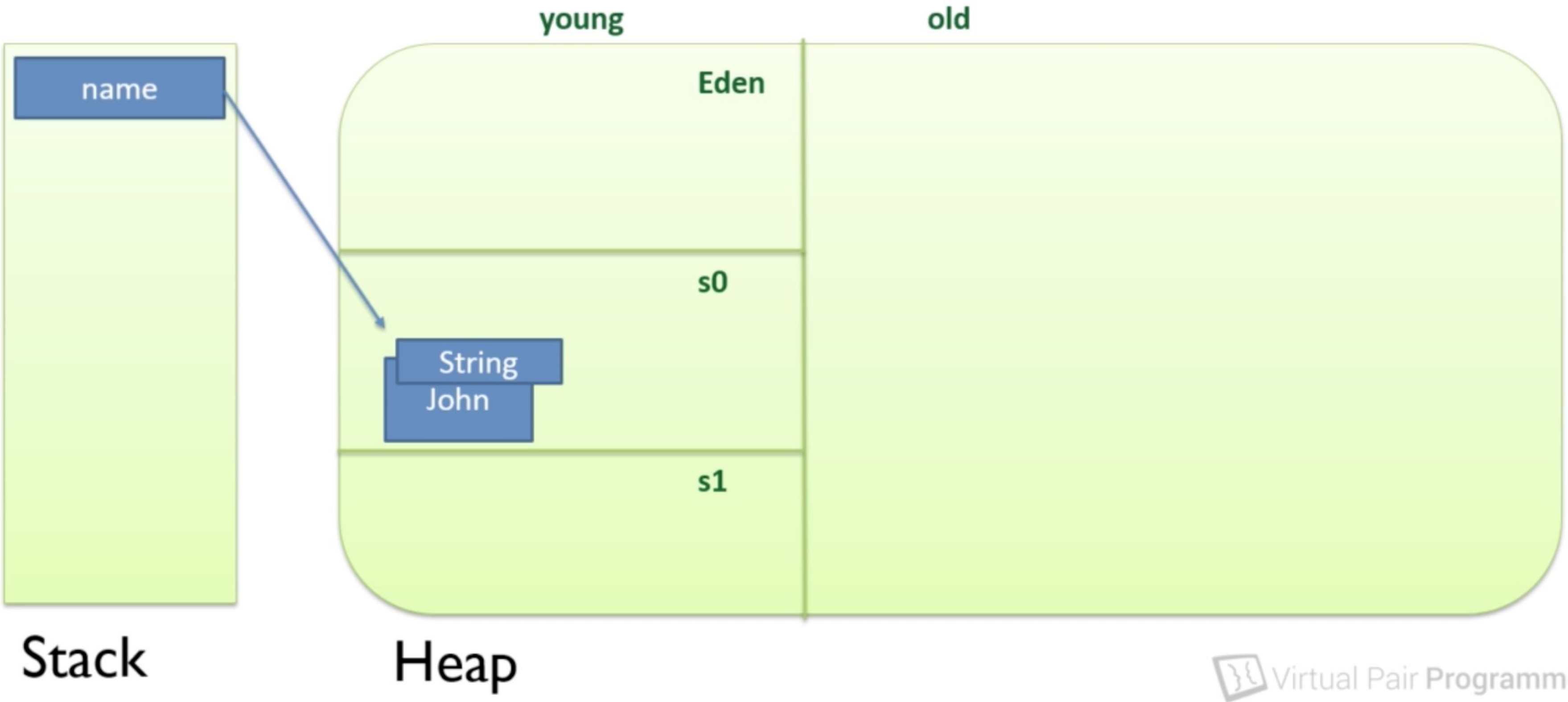
Young generation – survivor spaces



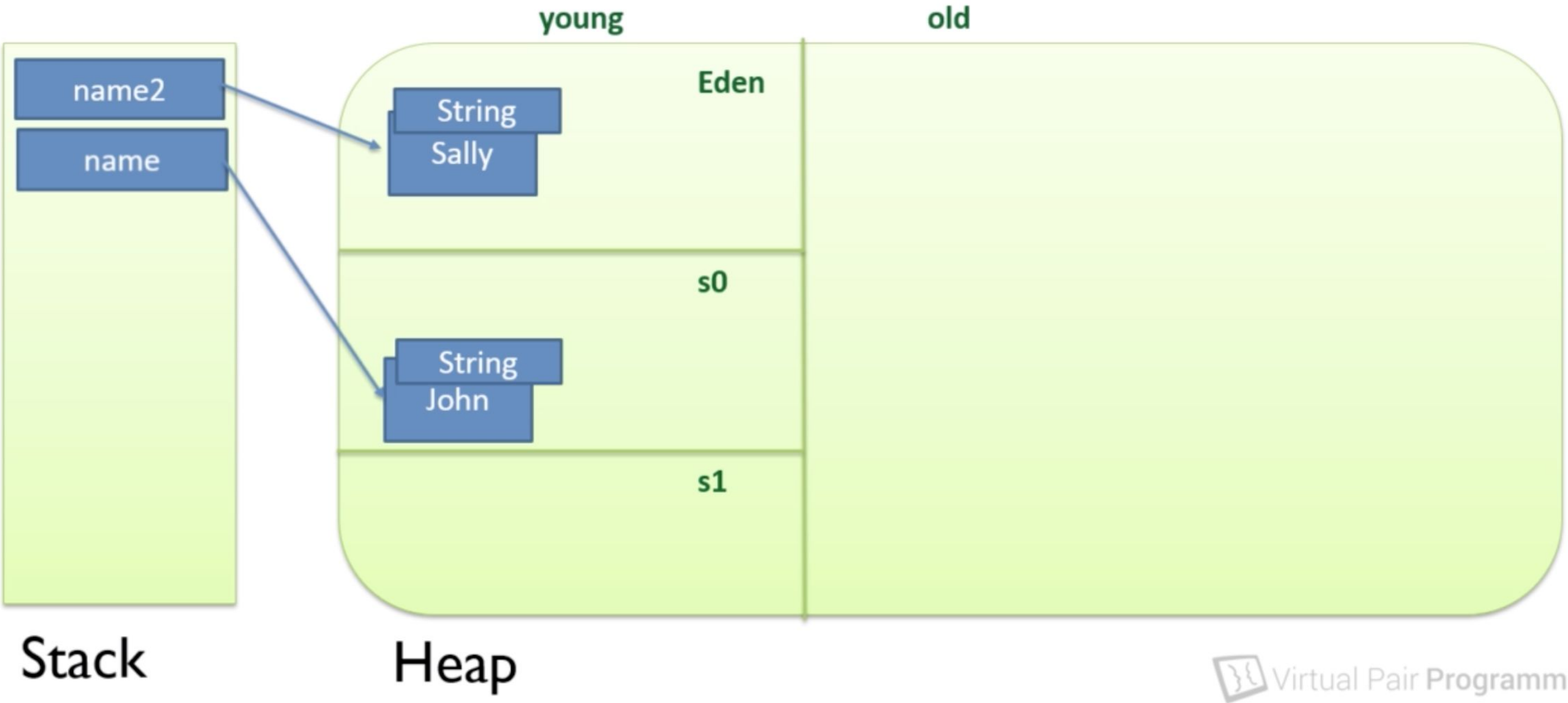
Young generation – survivor spaces



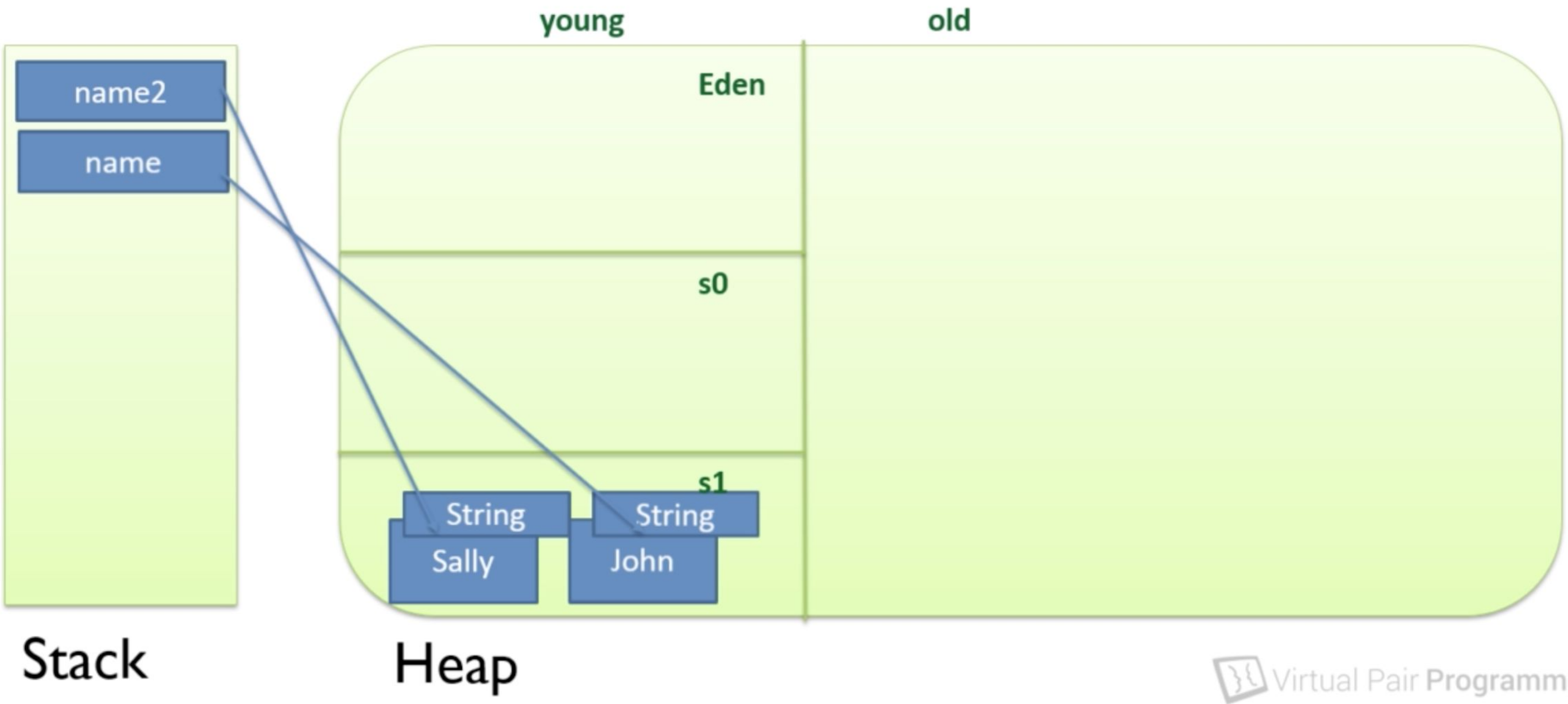
Young generation – survivor spaces



Young generation – survivor spaces

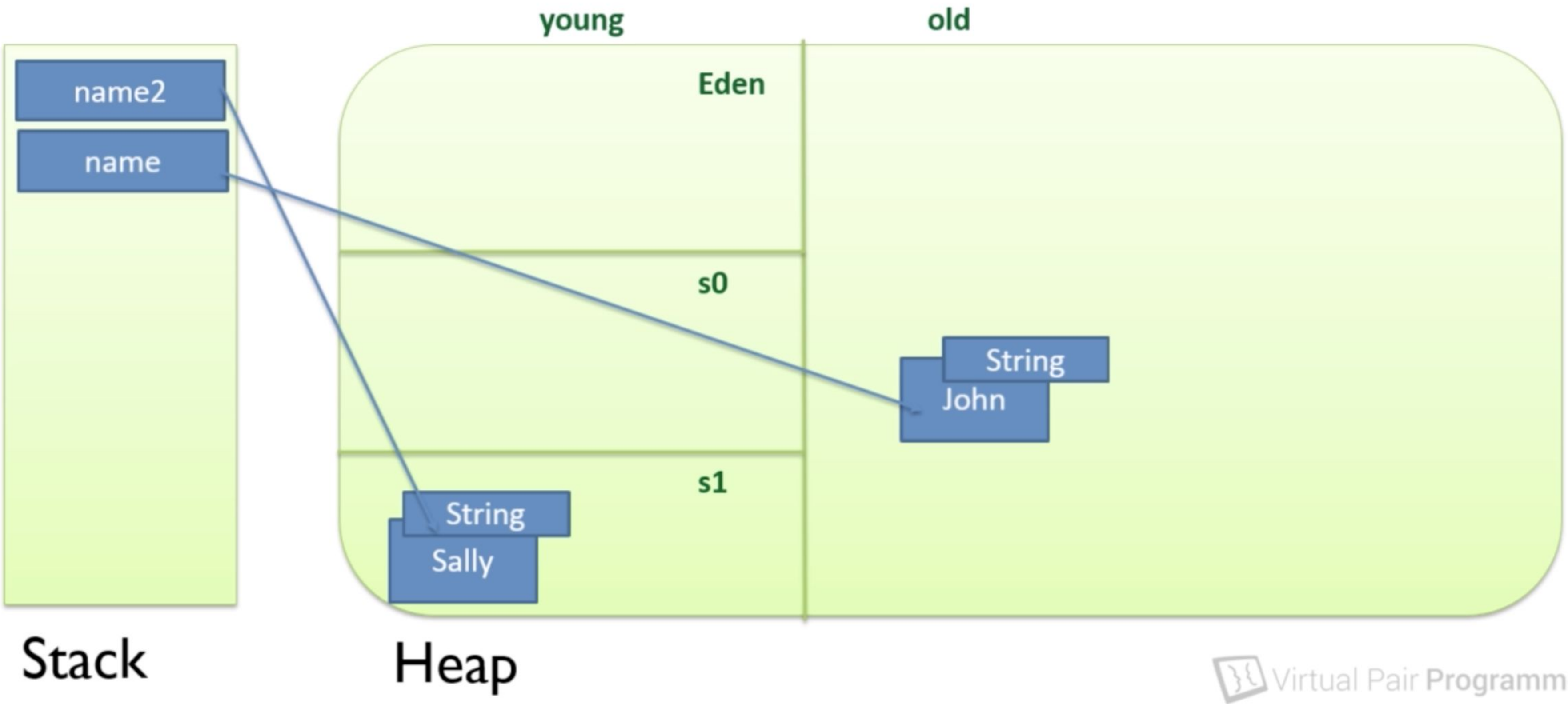


Young generation – survivor spaces

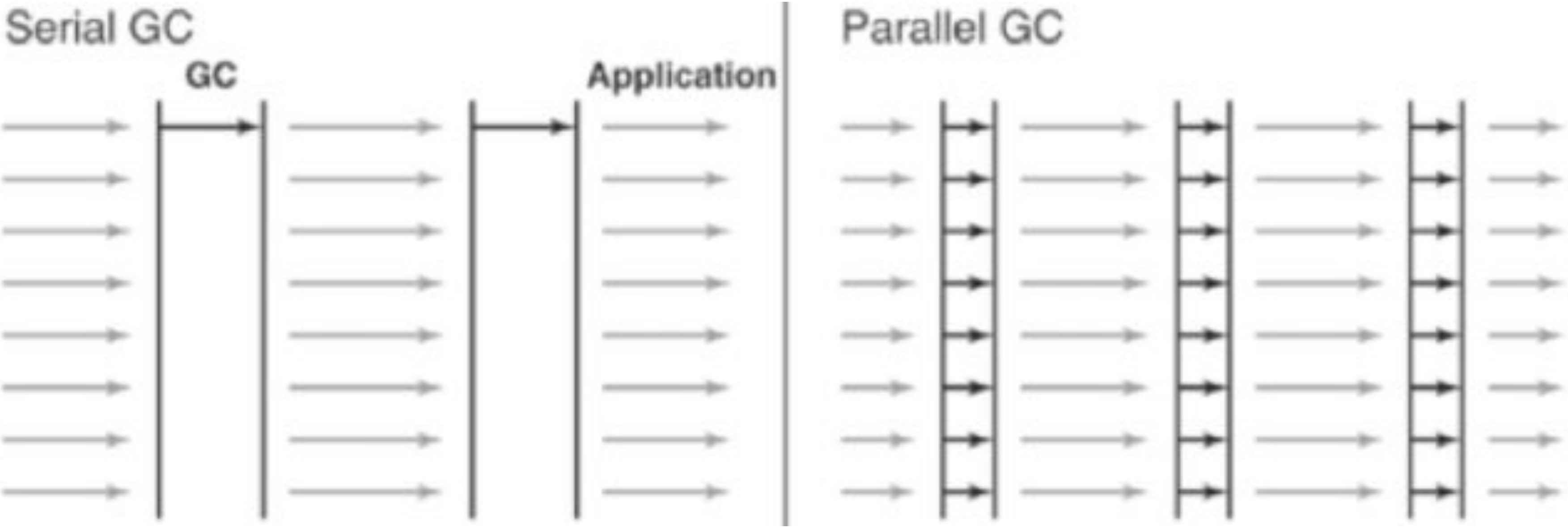


Virtual Pair Programmers

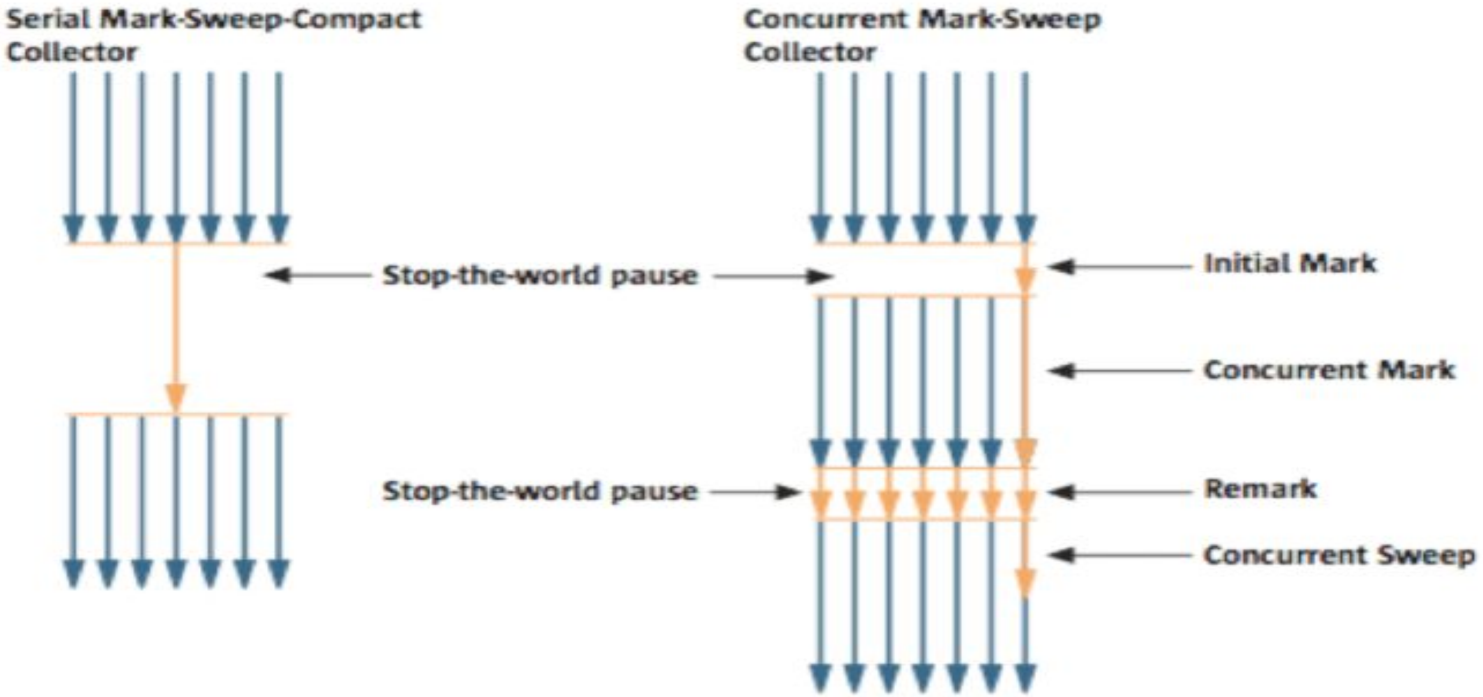
Young generation – survivor spaces



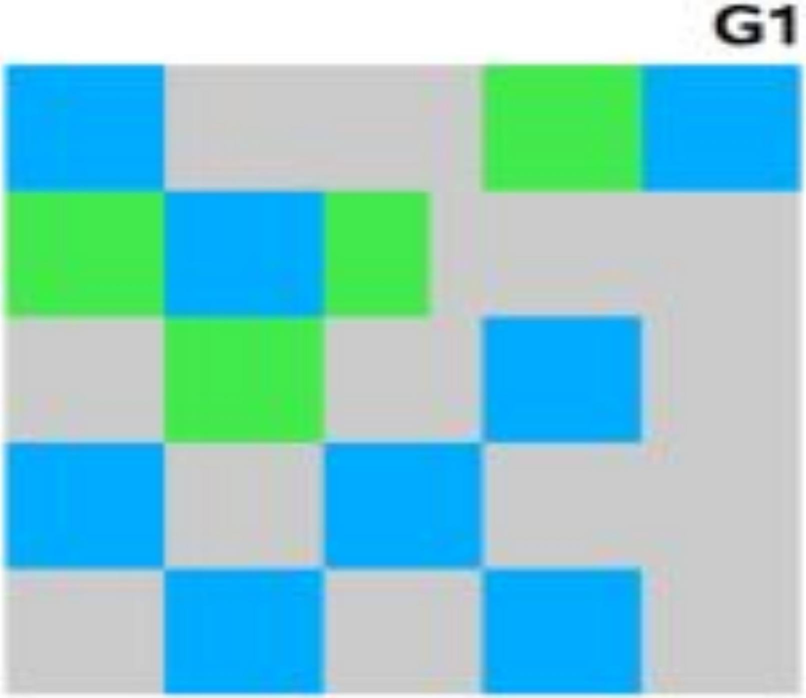
Serial GC & parallel GC

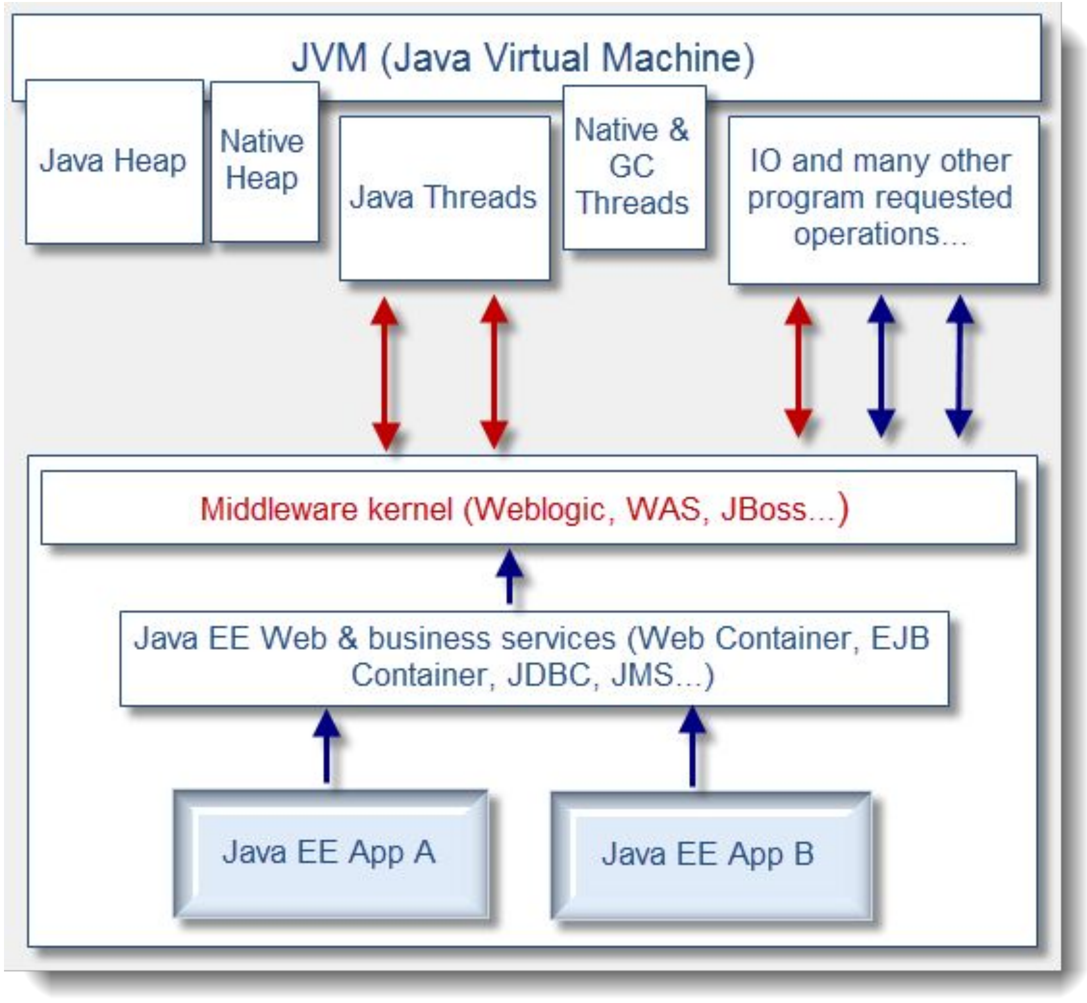


CMS GC


Serial Mark-Sweep-Compact Collector
Concurrent Mark-Sweep Collector
Stop-the-world pause
Initial Mark
Concurrent Mark
Stop-the-world pause
Remark
Concurrent Sweep

G1




JVM (Java Virtual Machine)
Java Heap
Native Heap
Java Threads
Native & GC Threads
IO and many other program requested operations...
Middleware kernel (Weblogic, WAS, JBoss...)
Java EE Web & business services (Web Container, EJB Container, JDBC, JMS...)
Java EE App A
Java EE App B