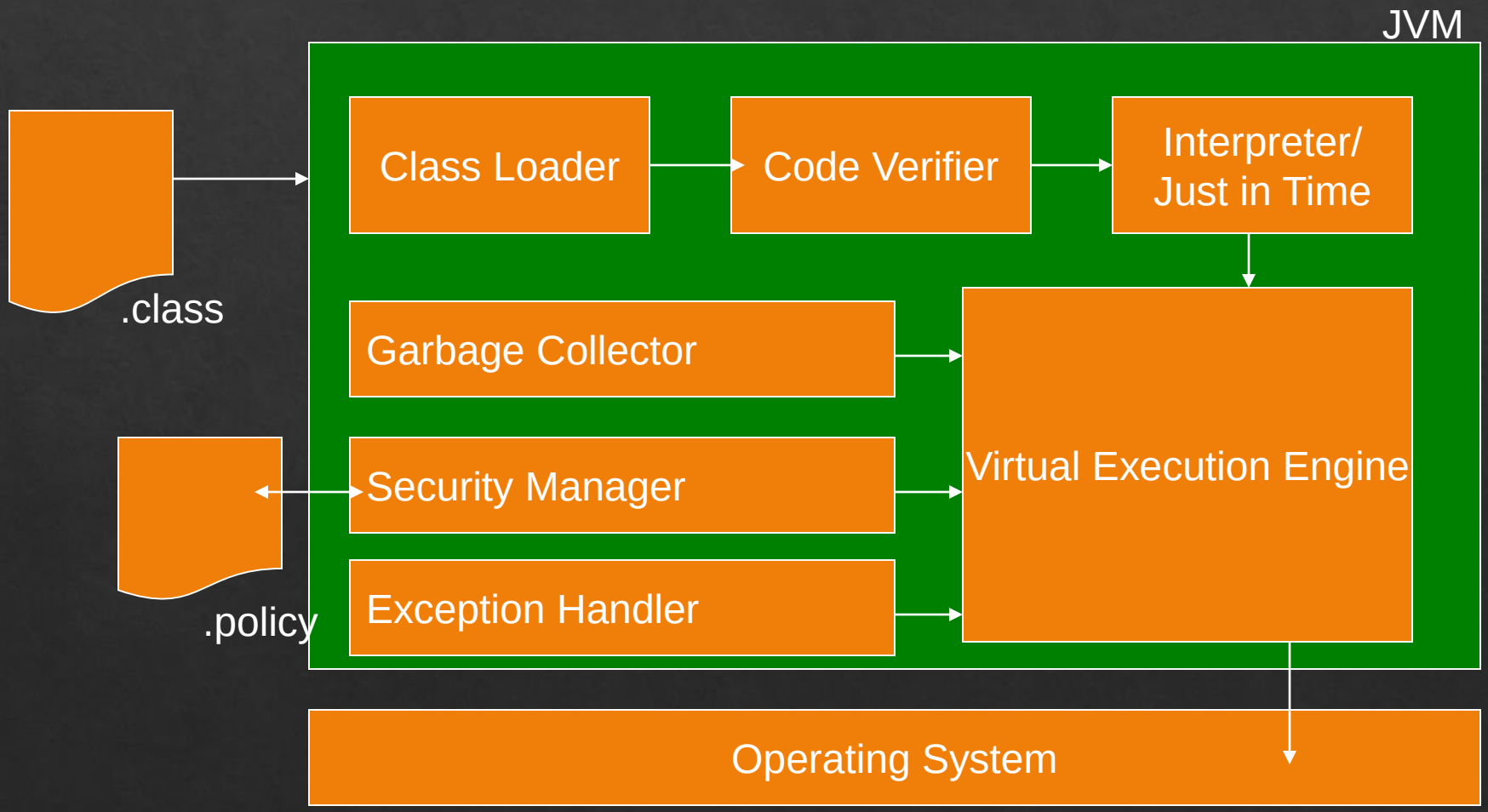
Java Platform

- Java Development Kit
  - To develop Java application
  - Comprises of JRE & set of Java Utilities
  - Also known as Java Standard Edition(Java SE)
- Java Runtime Environment
  - To execute Java Application
  - Comprises of JVM + rt.jar(runtime.jar)
- PATH & CLASSPATH variables
  - Path variable is used for locating java tools and exe's like javac, java
  - Classpath variable is used by JVM to locate the .classes at the time of execution

Java Execution Process



Java Execution Process

- Start of Execution
  - Use Java command to execute a .class file
  - Java command starts the JVM
  - JVM loads the class into Memory
  - Verifies the code
  - Converts the byte code to native code (specific to O/S)
  - Starts execution from the Entry point (main method)

Execution Process

- During Execution
  - Garbage collector deallocates memory for unused objects
  - Security Manager checks for code security against policy file
  - Exception handler manages exceptions
- End of Execution
  - When all active threads(Non-Deamon) are over Application ends
- More on JVM
  - Supports custom class loading
  - Supports several algorithms for GC

GC History

- Garbage Collection (GC) has been around for a while (1960's LISP, Smalltalk, Eiffel, Haskell, ML, Scheme and Modula-3)
- Went mainstream in the 1990's with Java (then C#)
- JVM implementations have improved on GC algorithms/speed over the past decade

GC Benefits/Cost

- Benefits
  - Increased reliability
  - Decoupling of memory mgmt from program design
  - Less time spent debugging memory errors
  - Dangling points/memory leaks do not occur (Note: Java programs do NOT have memory leaks; “unintentional object retention” is more accurate)
- Costs
  - Length of GC pause
  - CPU/memory utilization

GC Options

- JDK 1.3 included 3 GC strategies
- JDK 1.4 includes 6 GC strategies and over a dozen command line options
- Application type will demand strategy(1.5):
  - Real-time – short and bounded pauses
  - Enterprise – may tolerate longer or less predictable pauses in favor of higher throughput

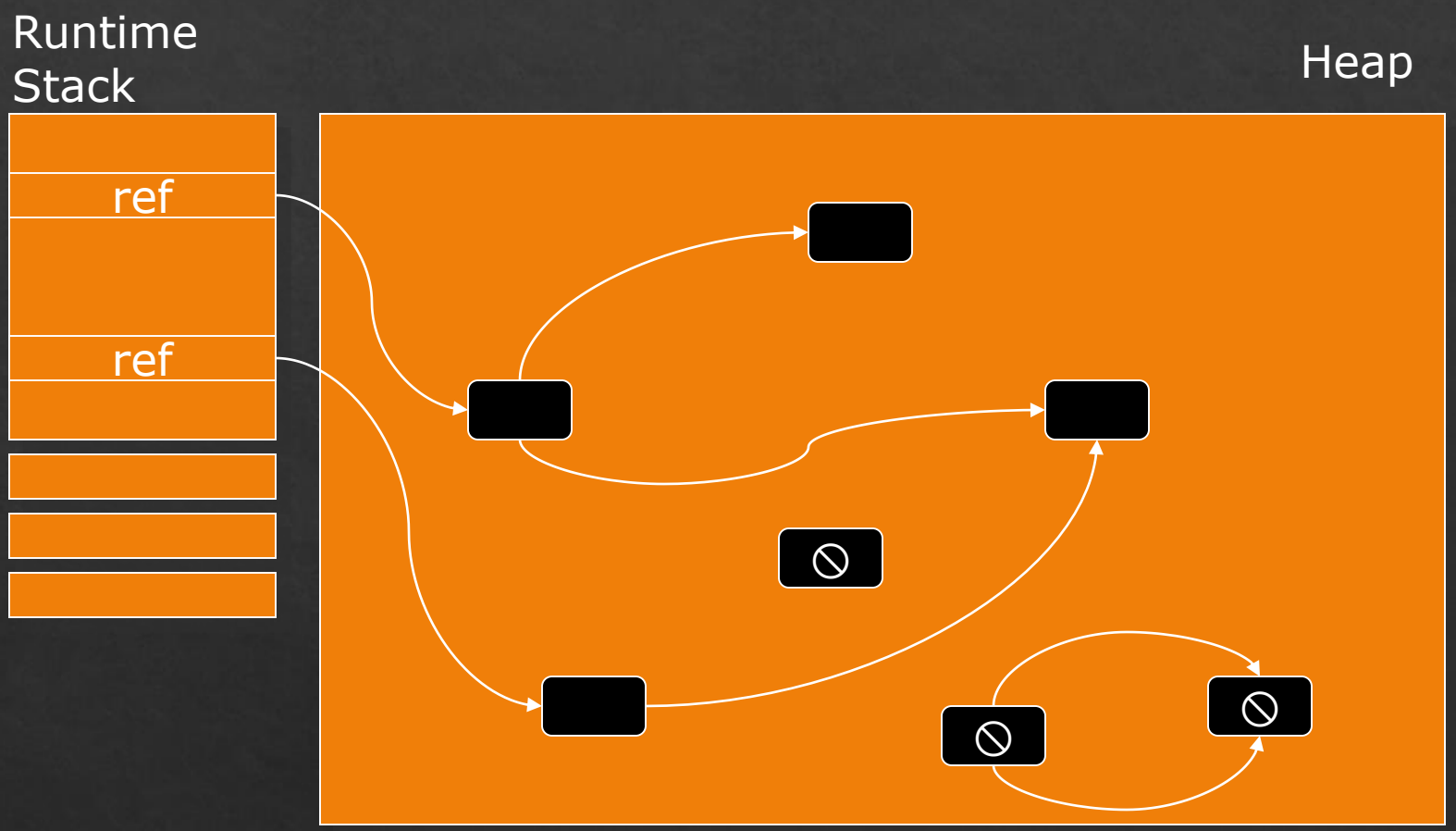
GC Phases

- GC has two main phases:
  - Detection
  - Reclamation
- Steps are either distinct:
  - Mark-Sweep, Mark-Compact
- … or interleaved
  - Copying

Reachability

- Roots – reference to object in a static variable or local variable on an active stack frame
- Root Objects – directly reachable from roots
- Live Objects – all objects transitively reachable from roots
- Garbage Objects – all other objects

Reachability Example



GC Algorithms

- Two basic approaches:
  - Reference Counting – keep a count of references to an object; when a count of zero, object is garbage
  - Tracing – trace out the graph of objects starting from roots and mark; when complete, unmarked objects are garbage

Reference Counting

- An early GC strategy
- Advantages:
  - can be run in small chunks of time
- Disadvantages:
  - does not detect cycles
  - overhead in incrementing/decrementing counters
- Reference Counting is currently out-of favor

Tracing

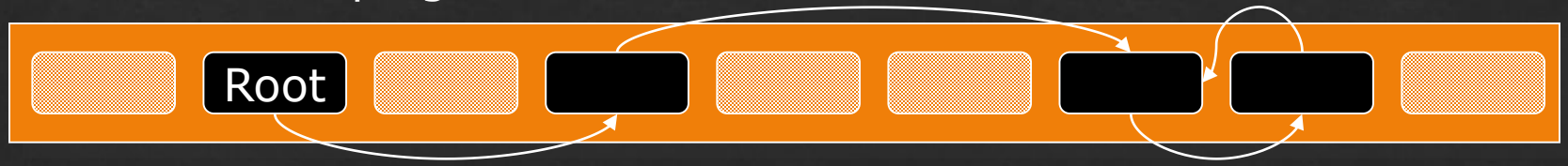
- Basic tracing algorithm is known as *mark and sweep*
  - *mark phase* – GC traverses reference tree, marking objects
  - *sweep phase* – umarked objects are finalized/freed

Mark-Sweep Example

1. mark-sweep start

2. end of marking

3. end of sweeping



Mark-Sweep

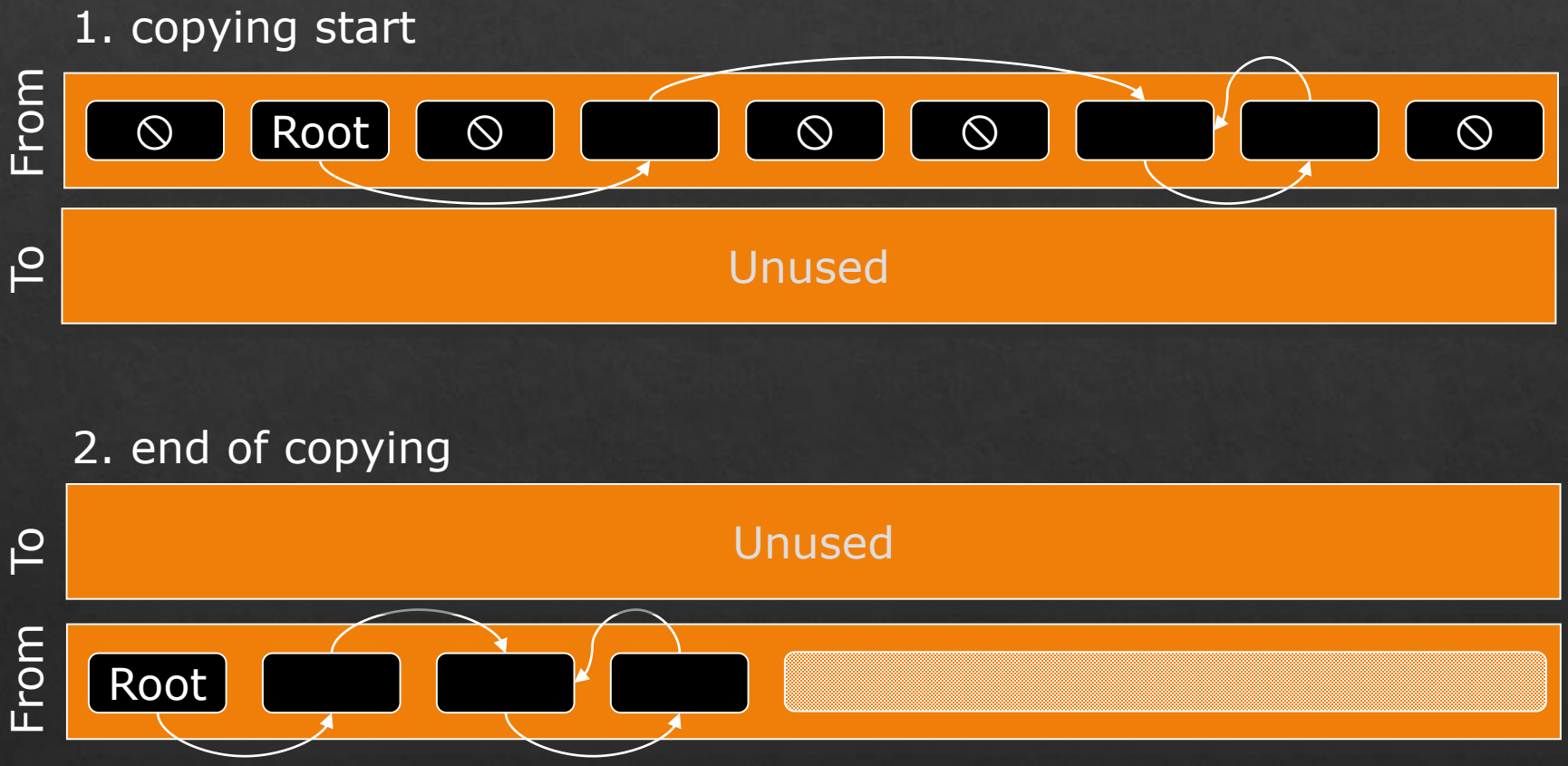
- Problem is fragmentation of memory
- Two strategies include:
  - Mark-Compact Collector – After mark, moves live objects to a contiguous area in the heap
  - Copy Collector – moves live objects to a new area

Mark-Compact Collector Example

1. end of marking

2. end of compacting

Copy Collector Example



Performance Characteristics

- Mark-Compact collection is roughly proportional to the size of the heap
- Copy collection time is roughly proportional to the number of live objects

Copying

- Advantages:
  - Very fast…if live object count is low
  - No fragmentation
  - Fast allocation
- Disadvantages:
  - Doubles space requirements – not practical for large heaps

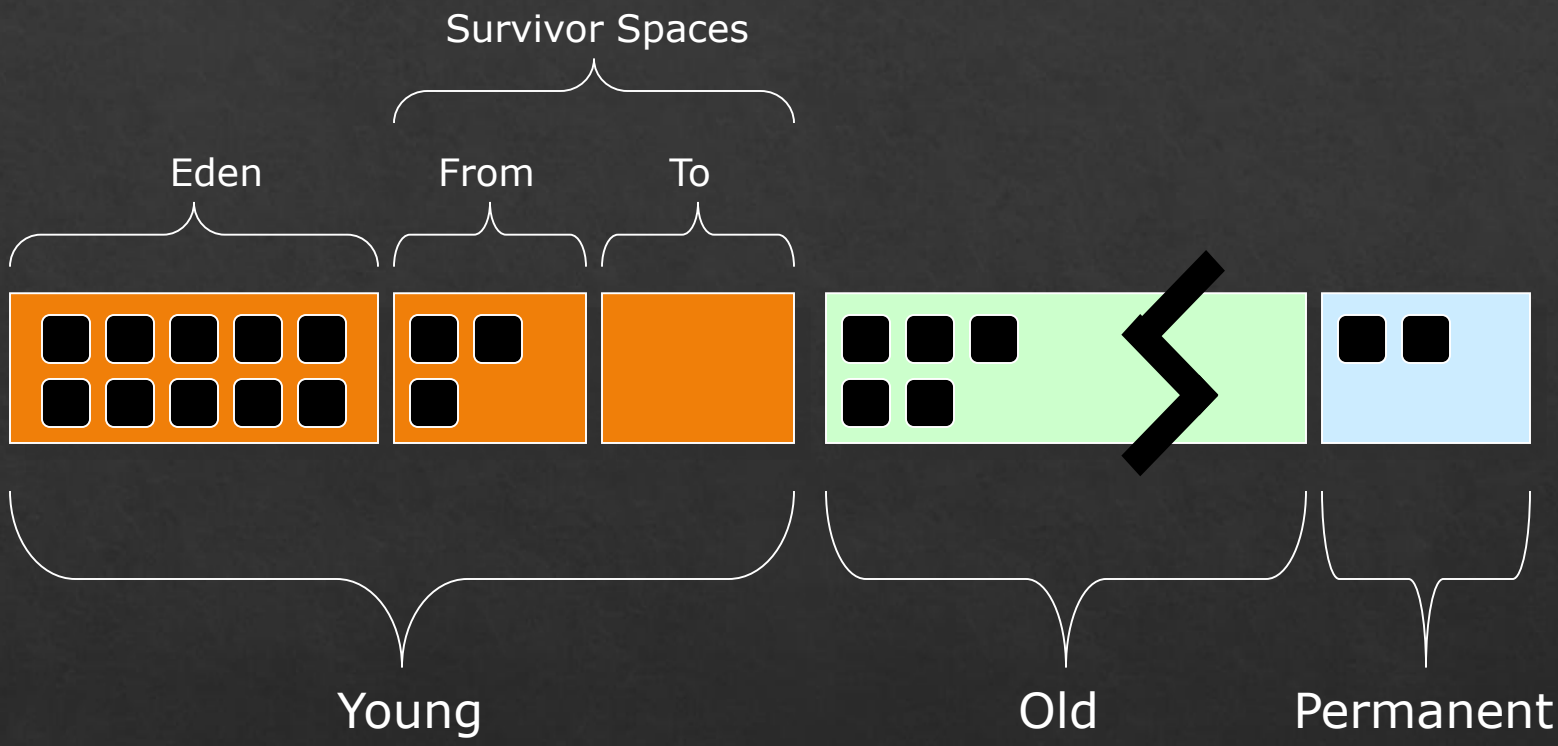
Observations

- Two very interesting observations:
  - Most allocated objects will die young
  - Few references from older to younger objects will exist

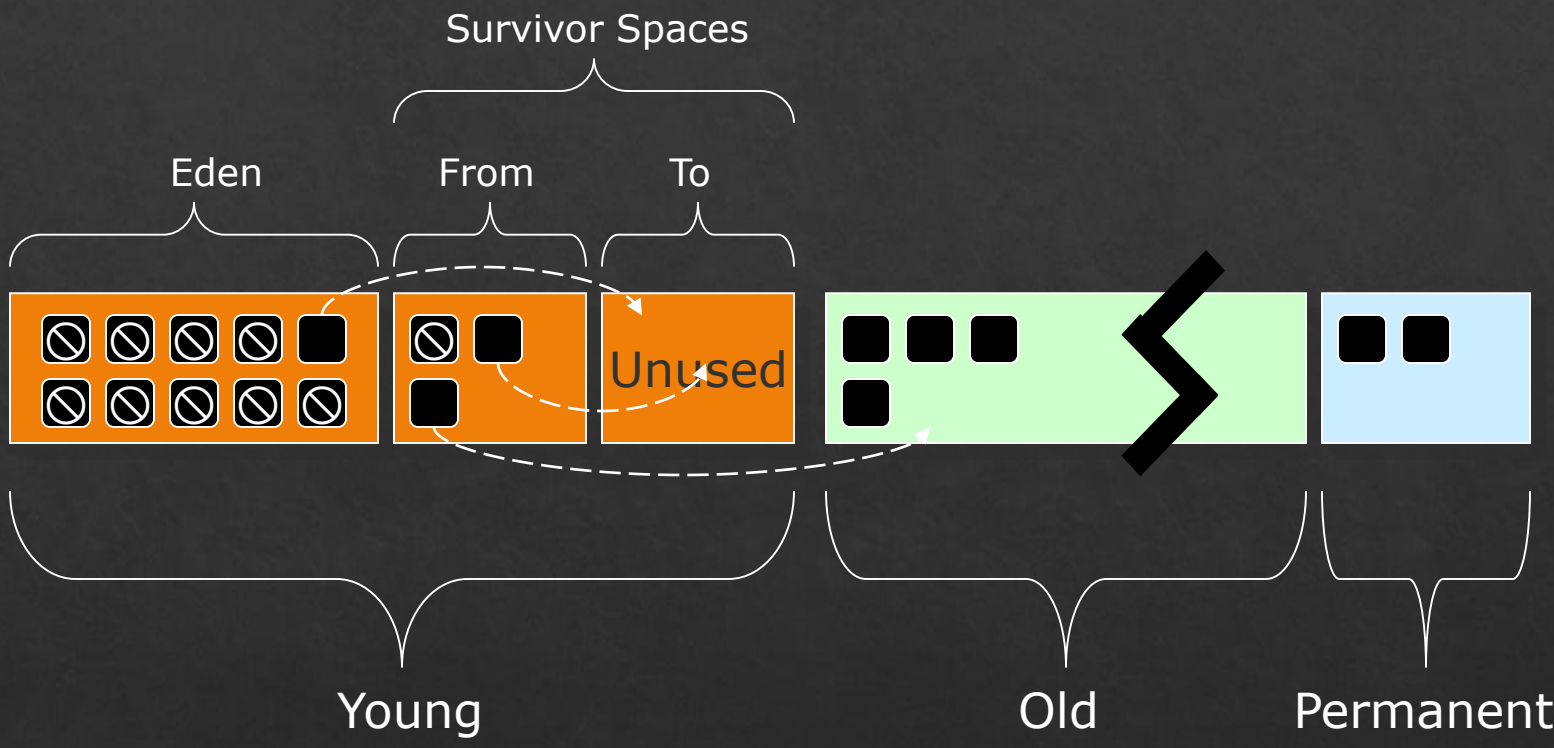
Generations

- Heap is split into generations, one young and one old
  - Young generation – all new objects are created here. Majority of GC activity takes place here and is usually fast (Minor GC).
  - Old generation – long lived objects are promoted (or tenured) to the old generation. Fewer GC's occur here, but when they do, it can be lengthy (Major GC).

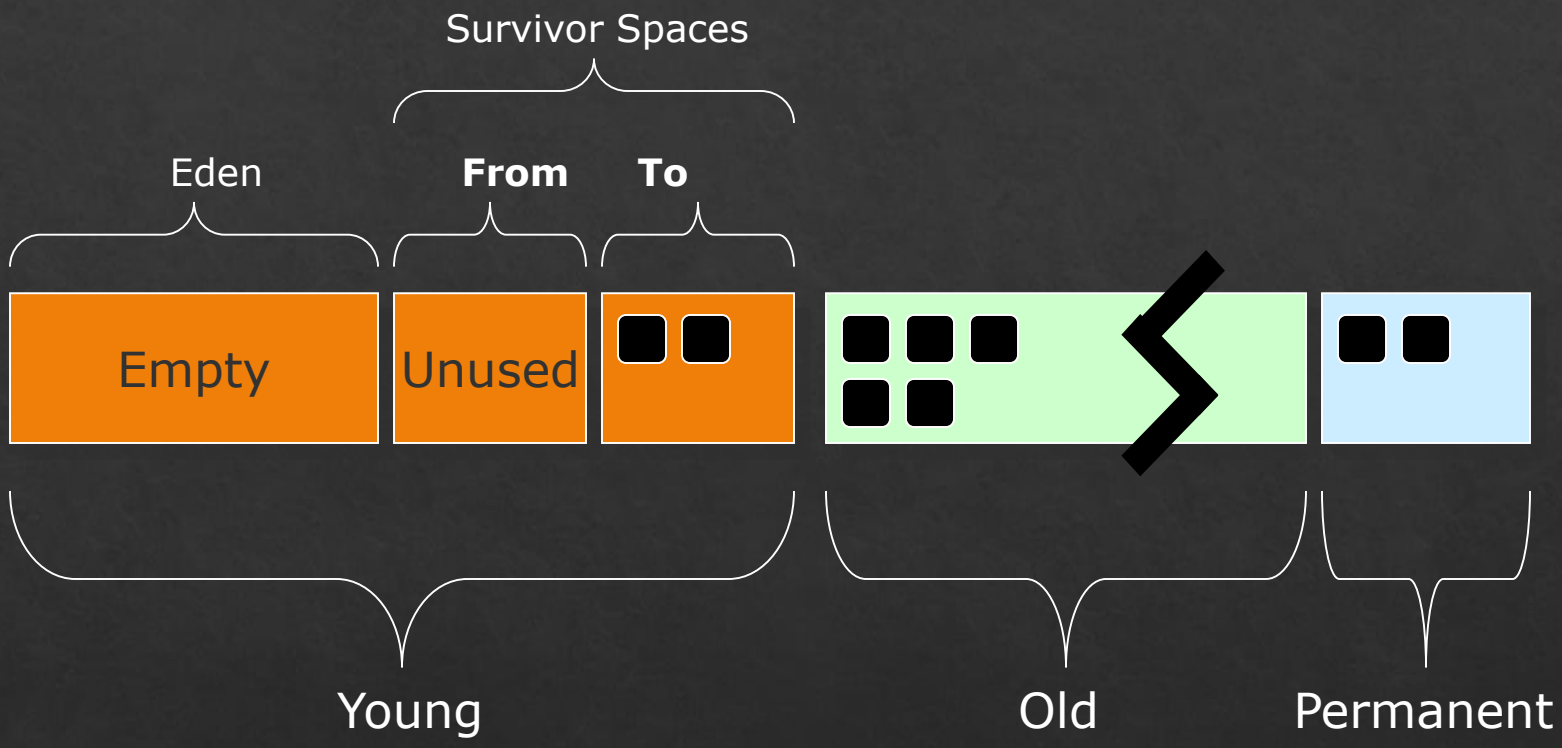
HotSpot Heap Layout



Before Minor GC



After Minor GC



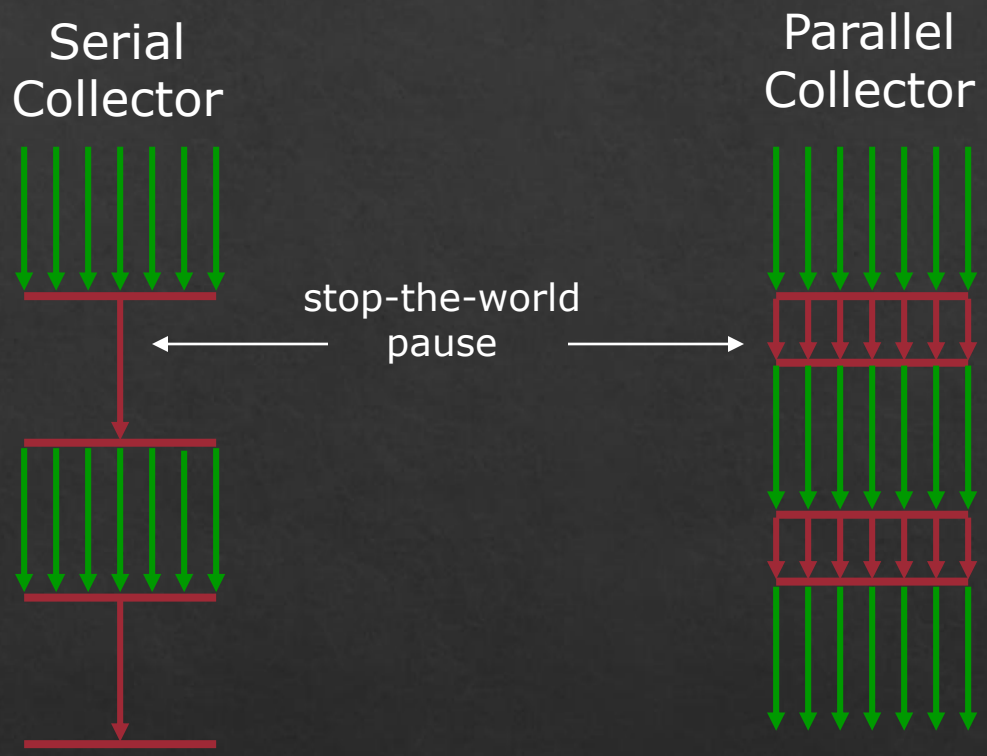
GC Notes

- Algorithm used will vary based on VM type (e.g., client/server)
- Algorithm used varies by generation
- Algorithm defaults change between VM releases

Young Generation Collectors

- Serial Copying Collector
  - All J2SEs (1.4.x default)
  - Stop-the-world
  - Single threaded
- Parallel Copying Collector
  - -XX:+UseParNewGC
  - JS2E 1.4.1+ (1.5.x default)
  - Stop-the-world
  - Multi-threaded
- Parallel Scavenge Collector
  - -XX:UseParallelGC
  - JS2E 1.4.1+
  - Like Parallel Copying Collector
  - Tuned for very large heaps (over 10GB) w/ multiple CPUs
  - Must use Mark-Compact Collector in Old Generation

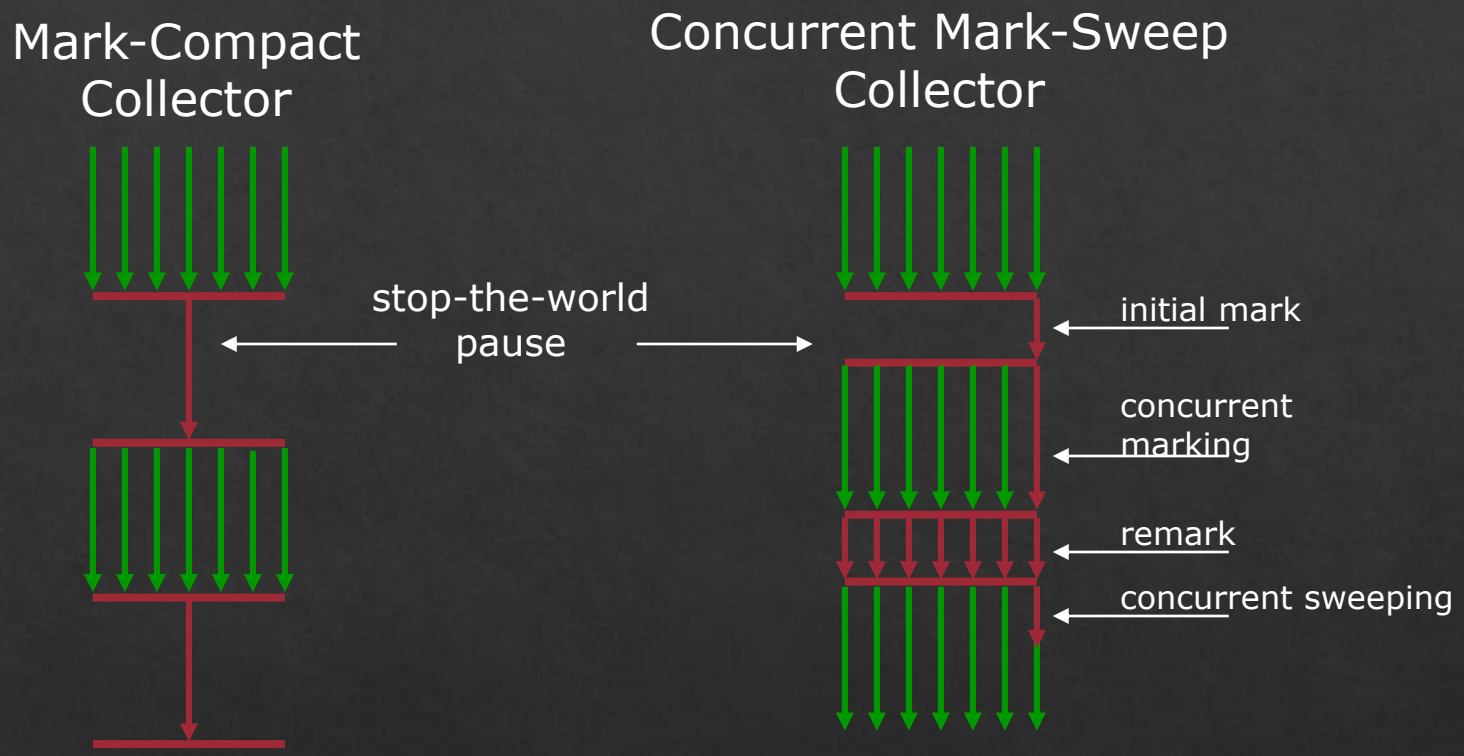
Serial vs. Parallel Collector



Old Generation Collectors

- Mark-Compact Collector
  - All J2SEs (1.4.x default)
  - Stop-the-world
  - Single threaded
- Train (or Incremental) Collector
  - -Xincgc
  - About 10% performance overhead
  - All J2SEs
  - To be replaced by CMS Collector
- Concurrent Mark-Sweep (CMS) Collector
  - -XX:+UseConcMarkSweepGC
  - J2SE 1.4.1+ (1.5.x default (-Xincgc))
  - Mostly concurrent
  - Single threaded

Mark-Compact vs. CMS Collector



JDK 1.5.x Heap Option Summary

Generation	Low Pause Collectors		Throughput Collectors		Heap Sizes
	1 CPU	2+ CPUs	1 CPU	2+ CPUs	

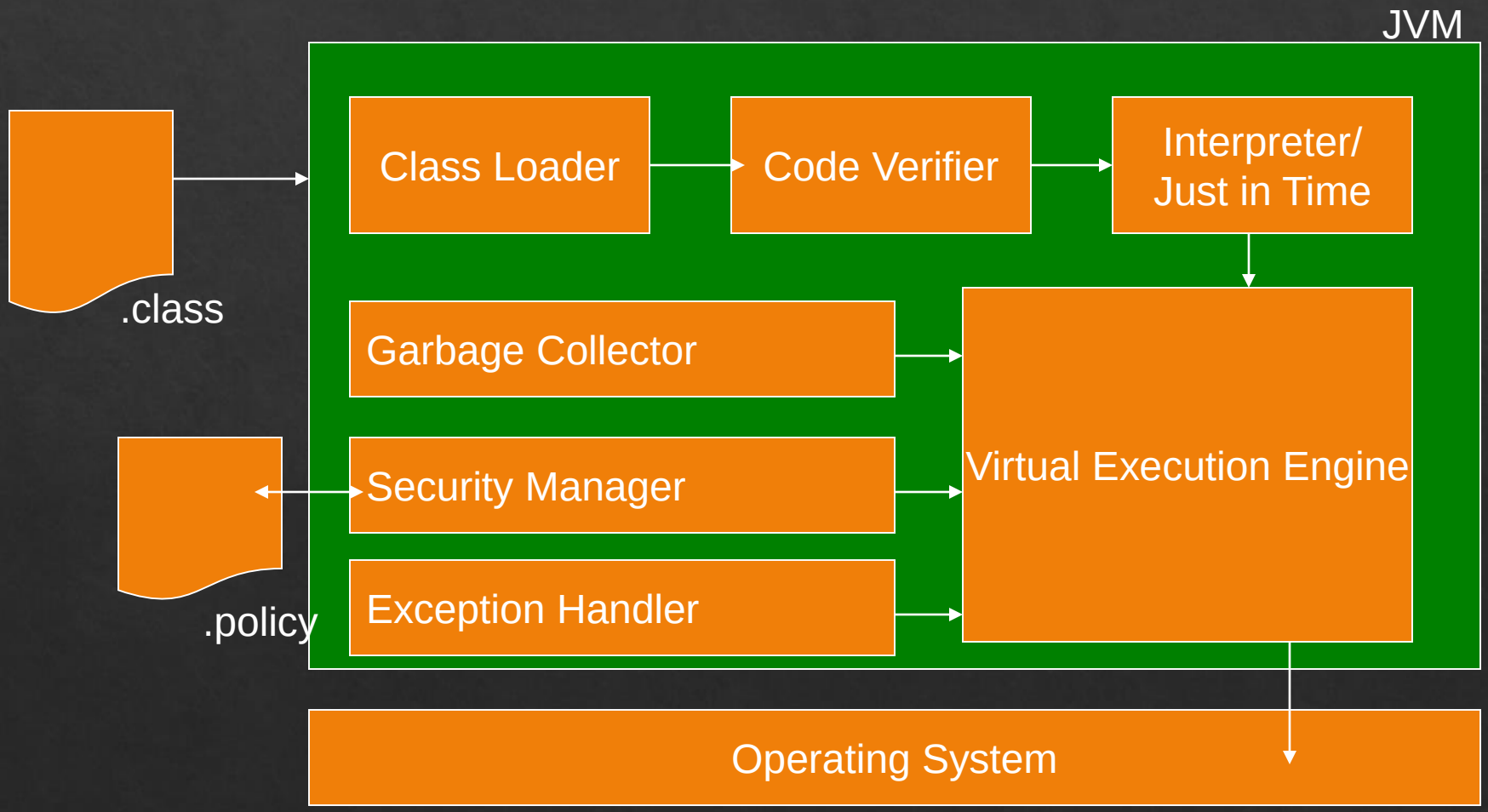
Young	Serial Copying Collector (default)	Parallel Copying Collector
		`-XX:+UseParNewGC`
			Copying Collector (default)	Parallel Scavenge Collector
				`-XX:+UseParallelGC`
					`-XX:NewSize`
					`-XX:MaxNewSize`
					`-XX:SurvivorRatio`

Old	Mark-Compact Collector (default)	Concurrent Collector `-XX:+UseConcMarkSweepGC`	Mark-Compact Collector (default)	Mark-Compact Collector (default)	`-Xms` `-Xmx`
Permanent	Can be turned off with `-Xnoclassgc` (use with care)				-XX:PermSize -XX:MaxPermSize

JVM Switches

- -Xms
  - Mimimum Heap Size
- -Xmx
  - Maximum Heap Size
- -XX:NewRatio
  - Ration between New and Old Generation
- -XX:NewSize
  - New generation size
- -XX:MaxNewSize
- –verbose:gc
- http://java.sun.com/javase/technologies/hotspot/vmoptions.jsp

Java Execution Process



References in Java

- Strong
  - Objects reachable by these references will never be garbage collected.
- Soft
  - Objects reachable by these references will never be garbage collected unless an OutofMemeoryException is going to be thrown.
- Weak
  - Objects reachable by these references will be garbage collected.