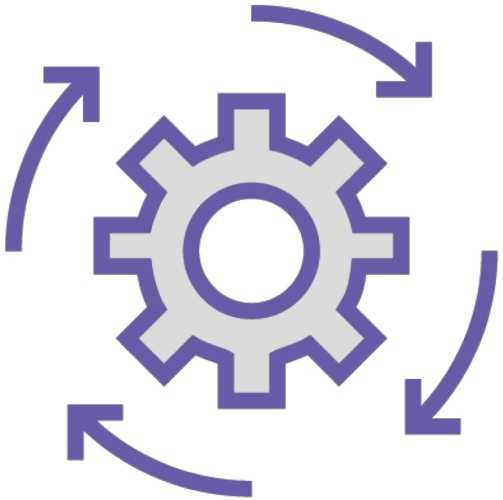
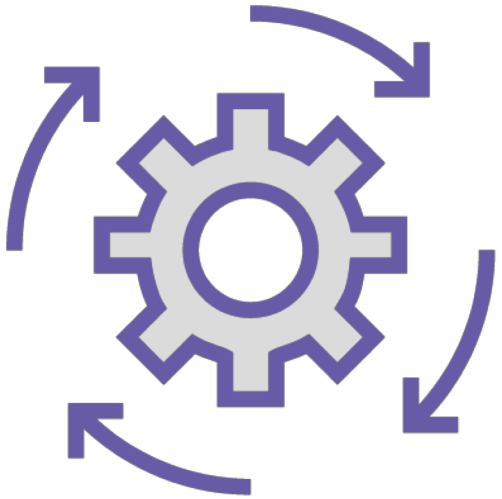
Language Types



Compiled

Interpreted



Compiled

Transforms program code into binary instructions

Compiled binaries target specific CPU architectures

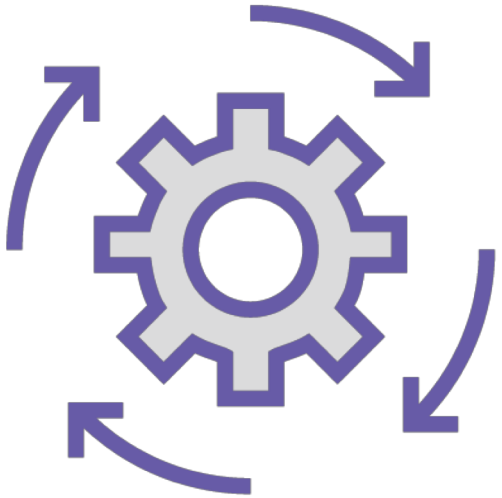
Interpreted

Translates each line of code into binary instructions as the line is executed

Is portable

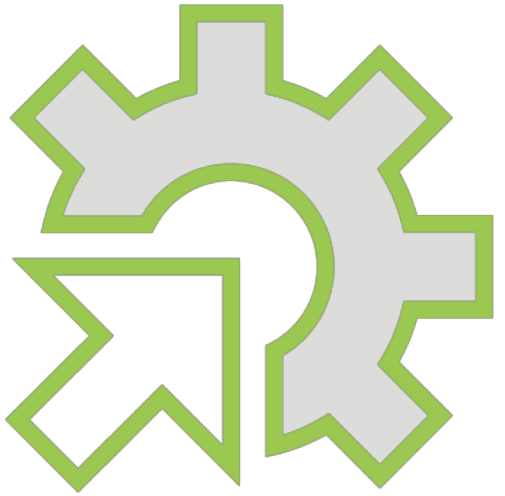
Is not as fast as compiled languages

Language Types



Compiled

Interpreted



Intermediate

Intermediate Language Benefits

Compiler can perform type checking and optimizations

Bytecode is compiled only once, and is close to machine code

Maintains portability

Just-In-Time Compiler

Introduced in 1999

Identifies hot methods (“hotspots”) and compiles them

Overview

JIT Compilation Modes

JIT Tuning Flags

Garbage Collection Intro

Measuring Garbage Collection Performance

Garbage Collection Tuning

JIT Compilation Modes

JIT Compiler Types

Compiler 1 / Client Compiler

Compiler 2 / Server Compiler

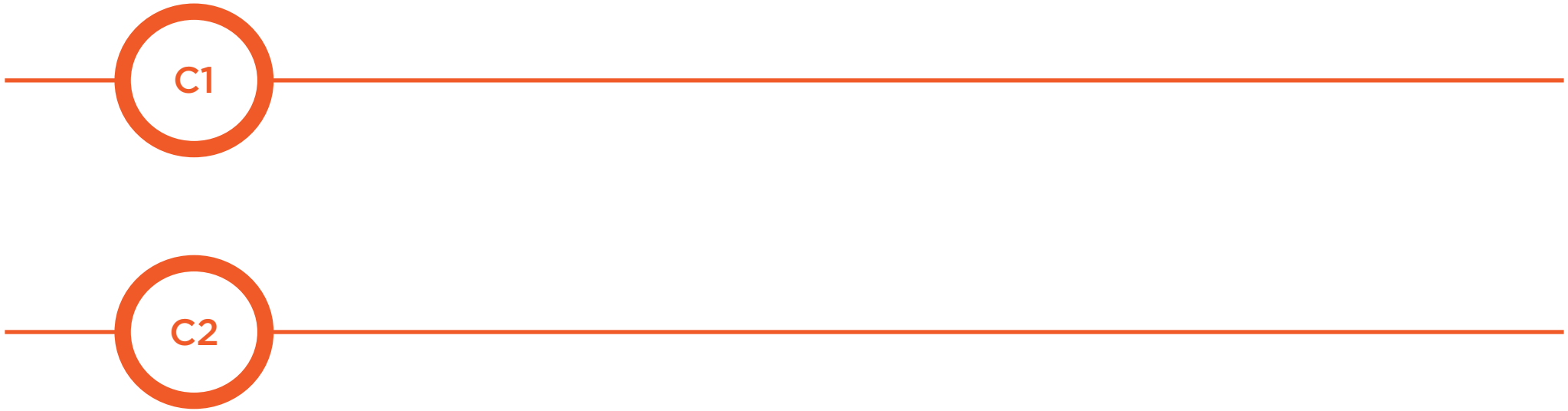
JIT Compiler Types

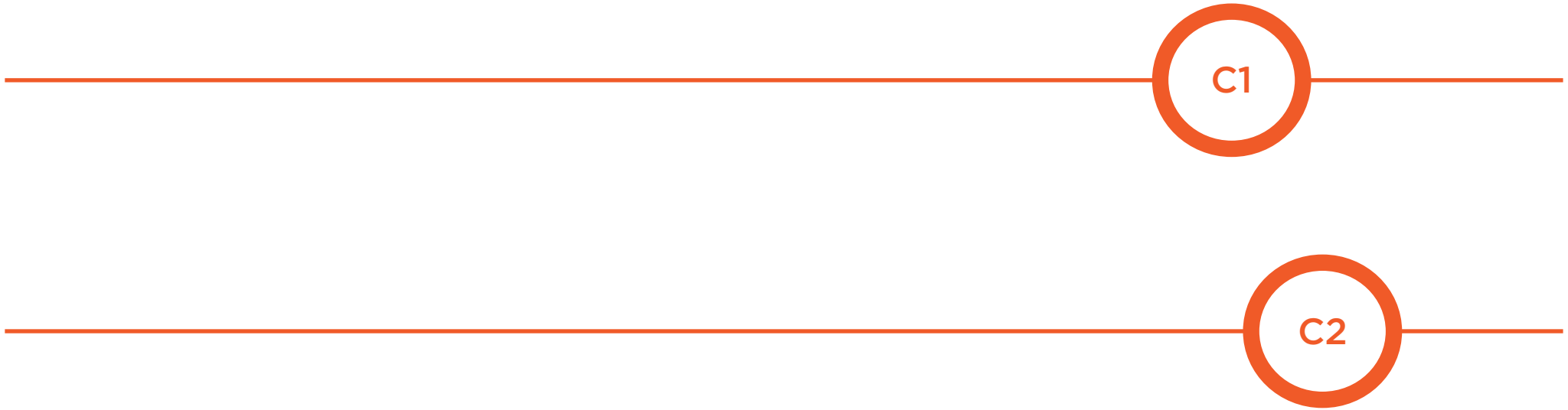
Compiler 1 / Client Compiler

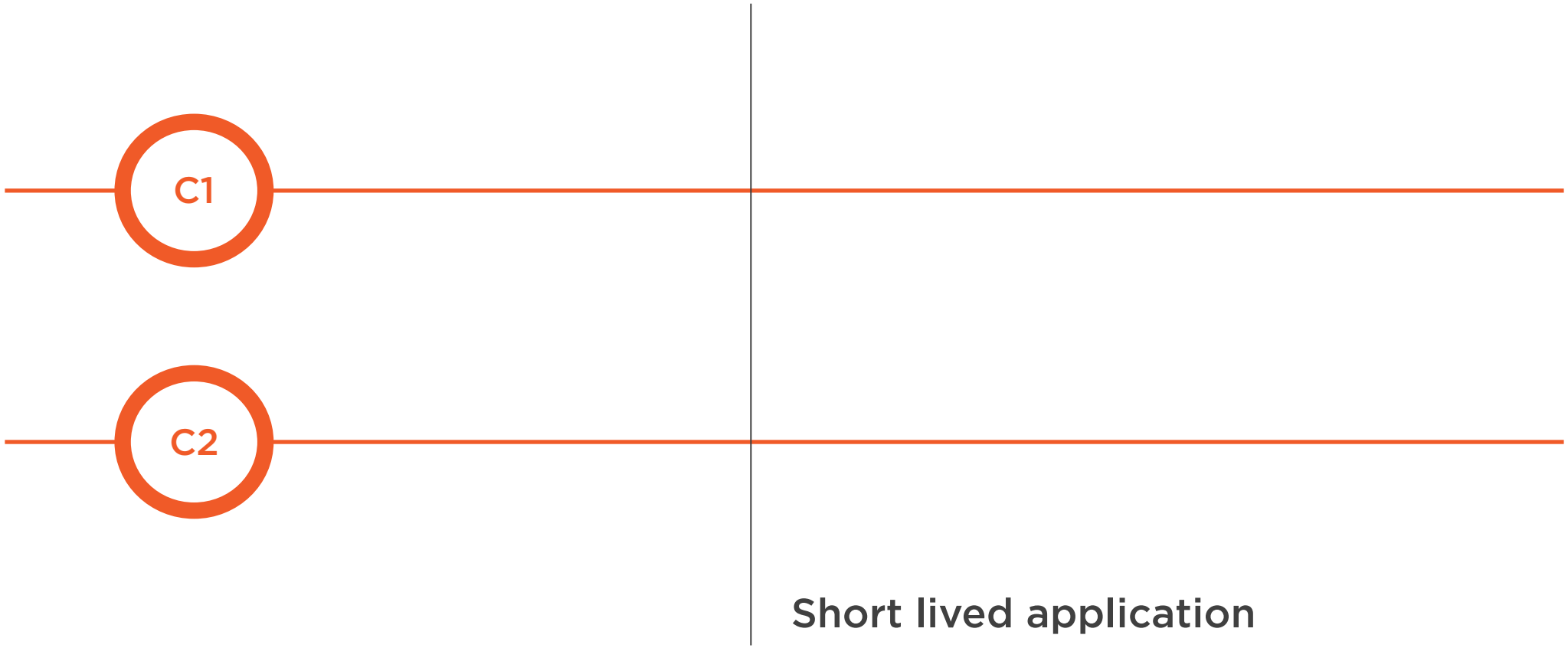
Optimized for startup performance

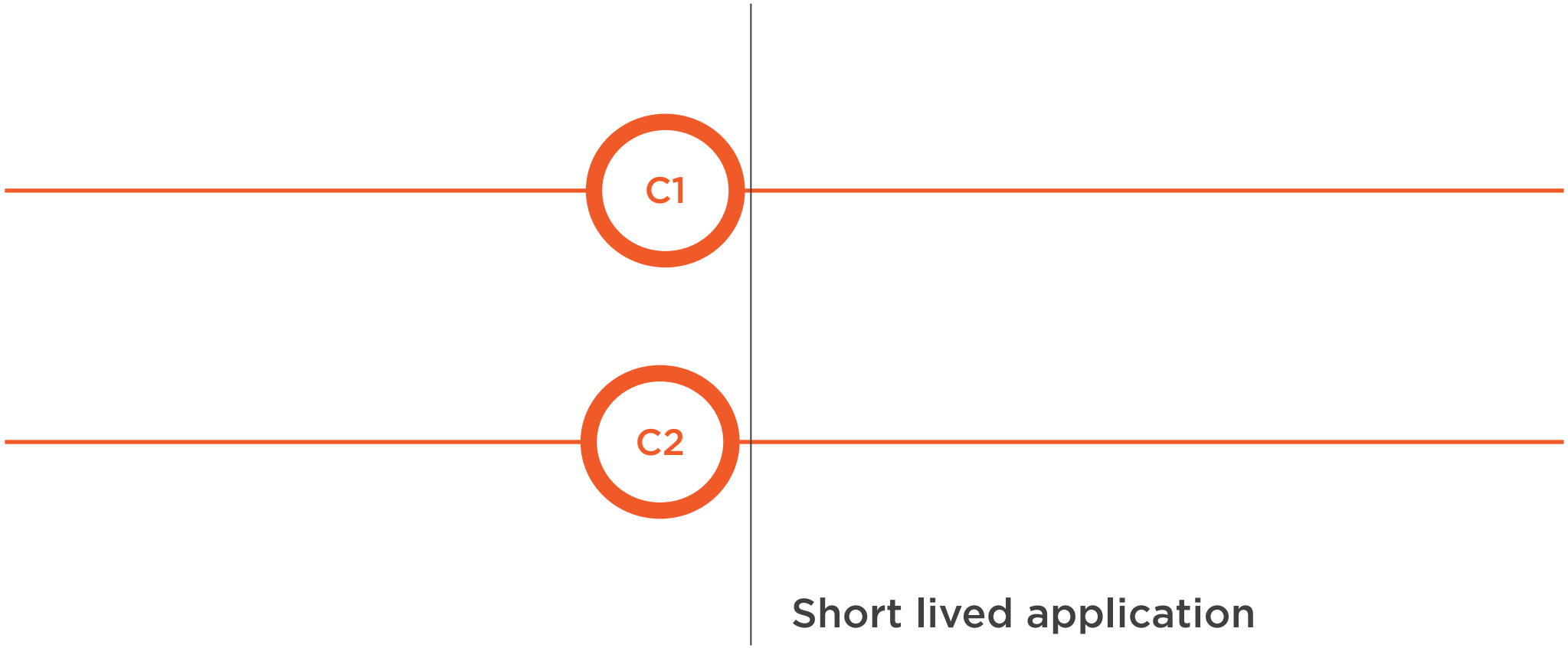
Compiler 2 / Server Compiler

Waits and profiles so that it can apply more aggressive optimizations


C1
C2


C1
C2


C1
C2
Short lived application


C1
C2
Short lived application

Tiered Compilation

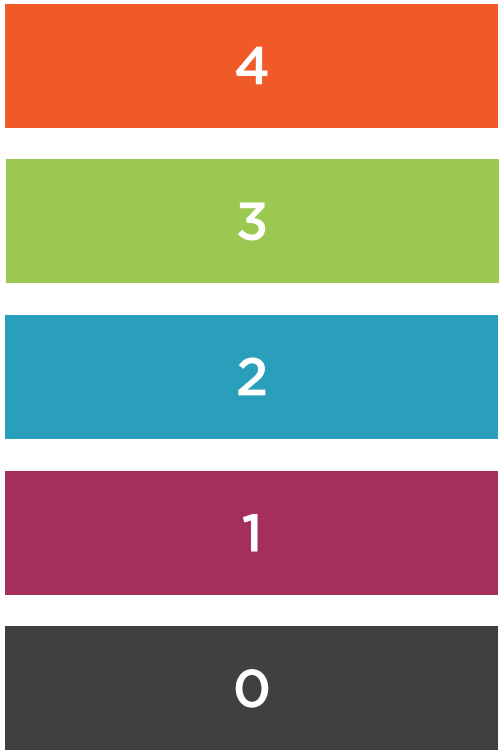
Just-In-Time Compiler

Introduced in Java 7

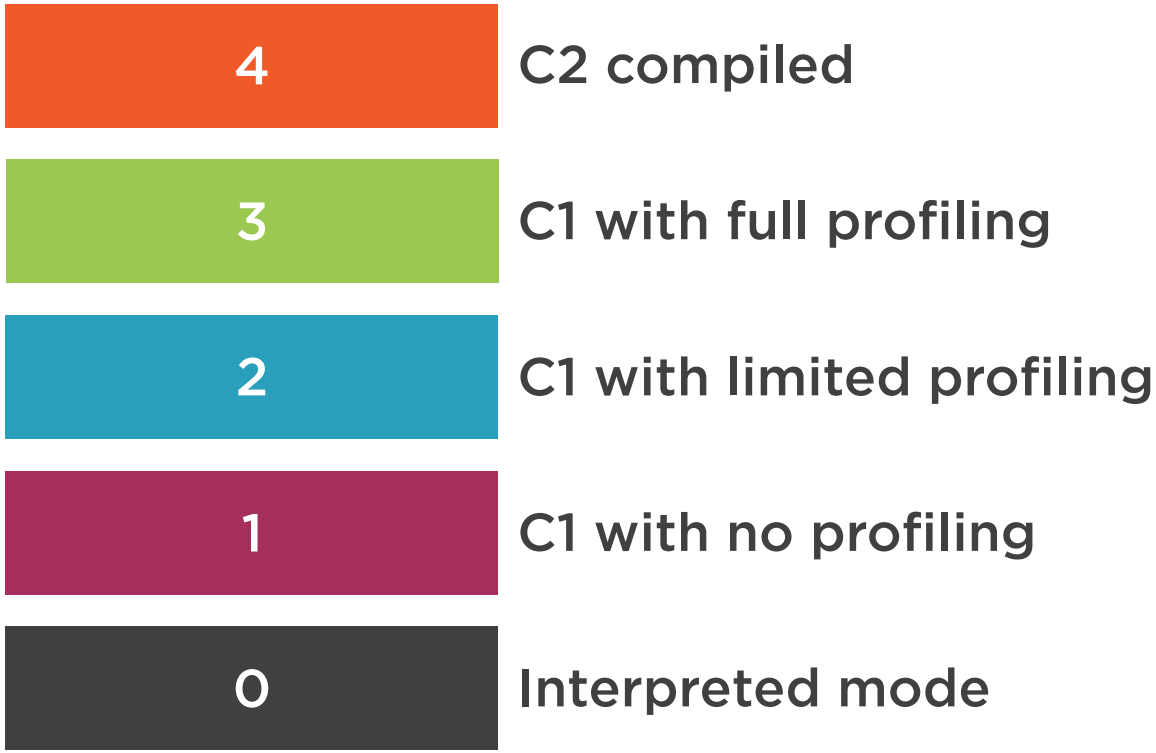
Hot methods are first compiled with C1, and then C2 as they get hotter

Set as default in Java 8

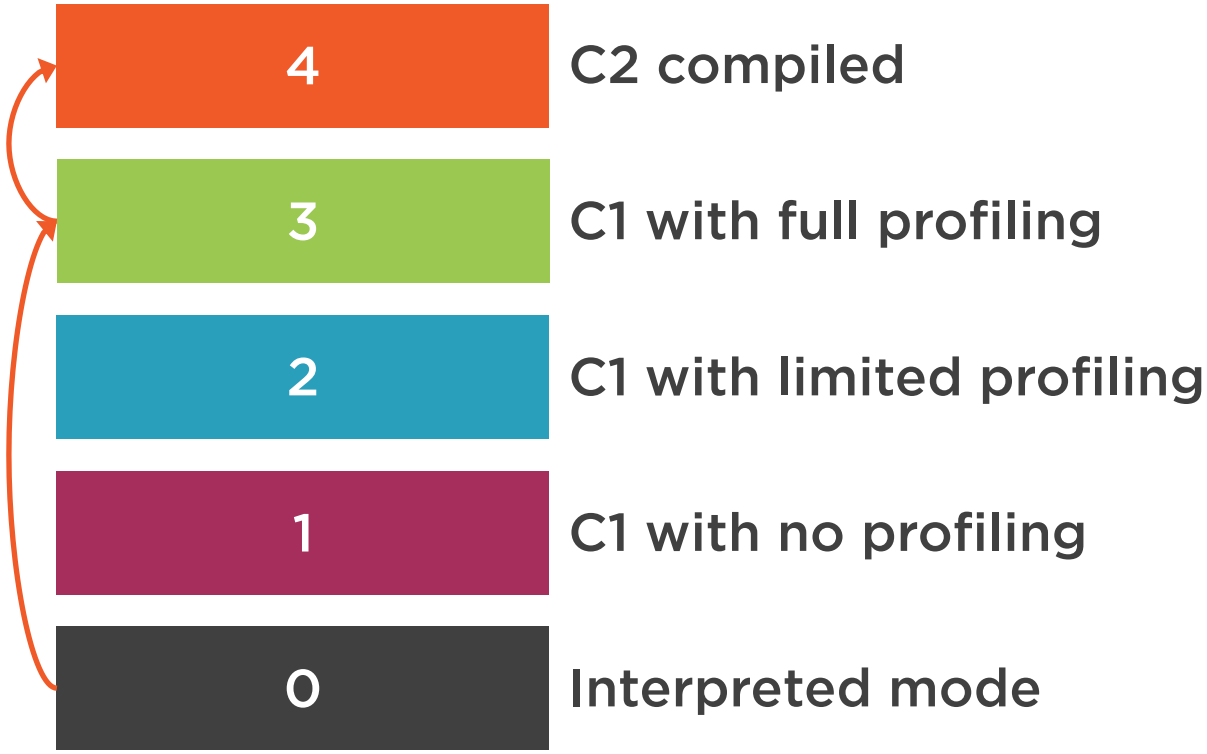
Tiered Compilation Levels



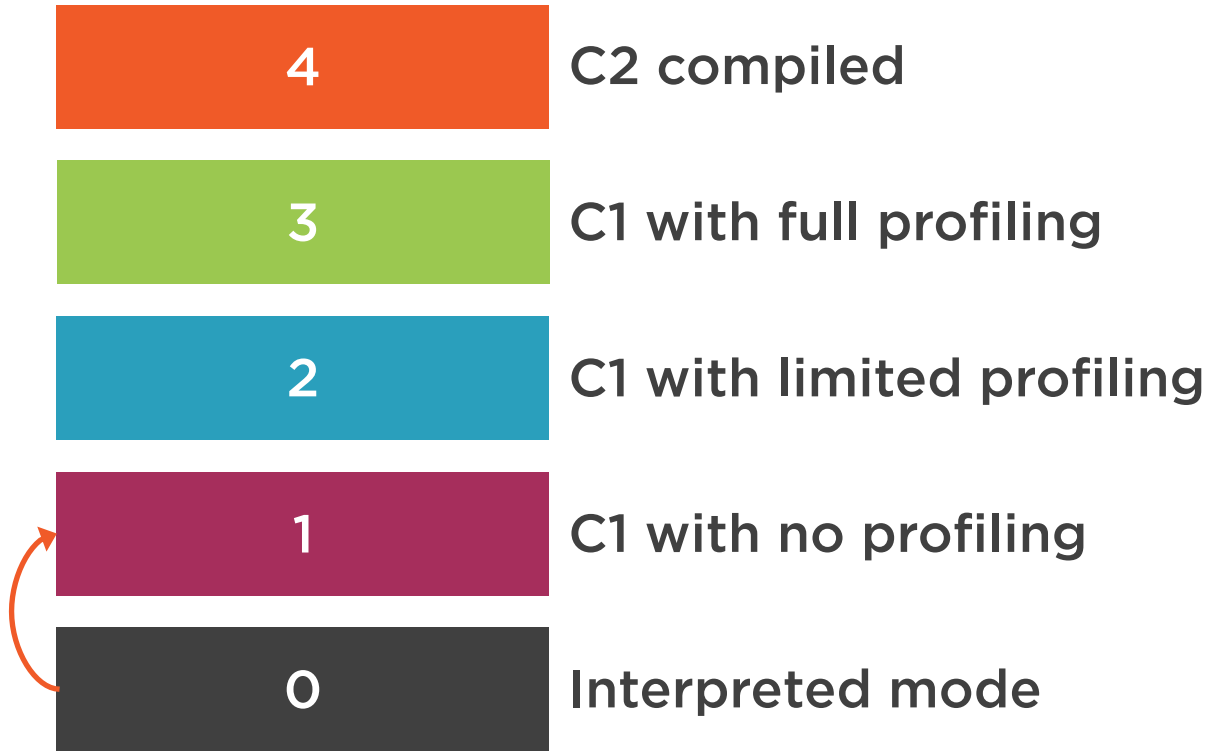
Tiered Compilation Levels



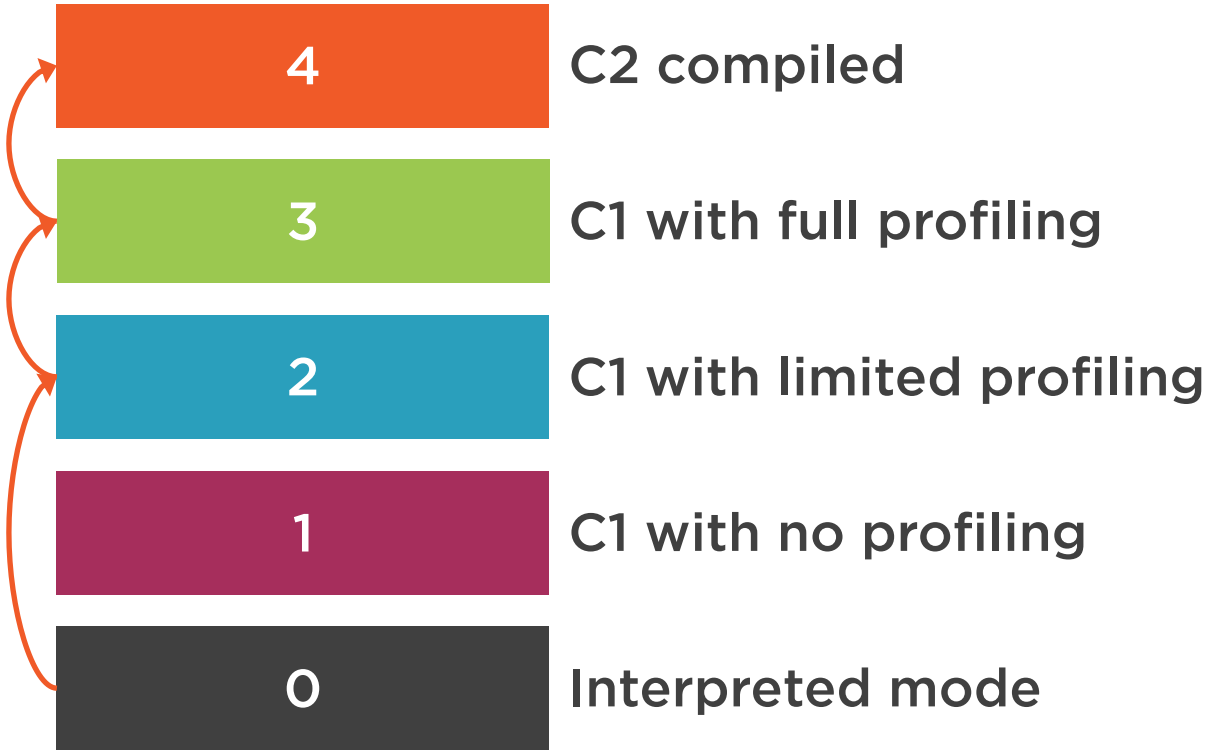
Tiered Compilation Levels



Tiered Compilation Levels



Tiered Compilation Levels



Choosing Your Compilation Mode

Start with **Tiered Compilation**

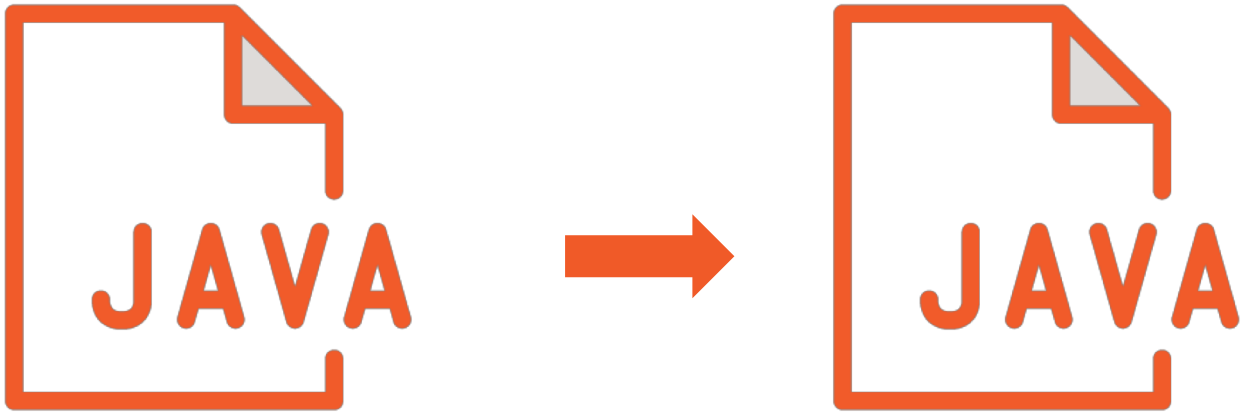
Test with **C1** or

Test with **C2**

Choosing Your Compilation Mode

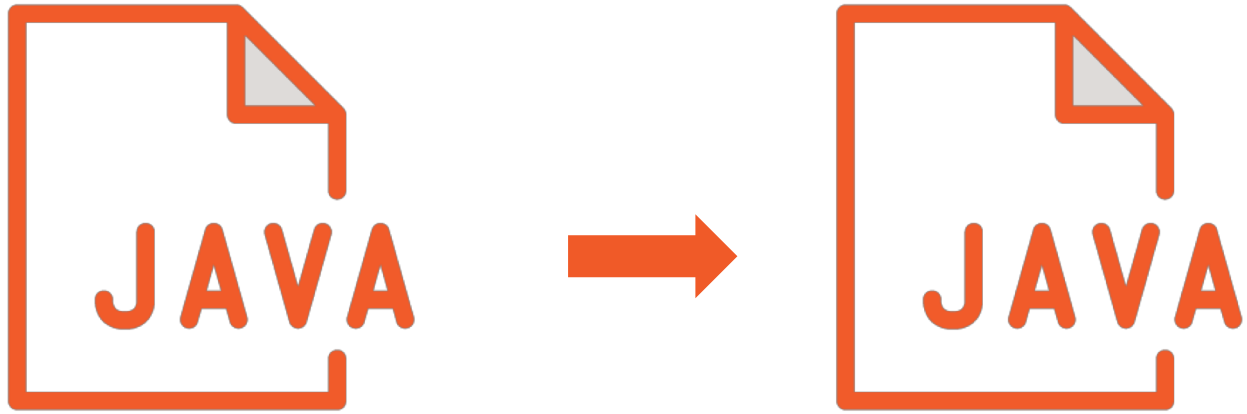
Start with **Tiered Compilation**

Test with **C2**



Java 7

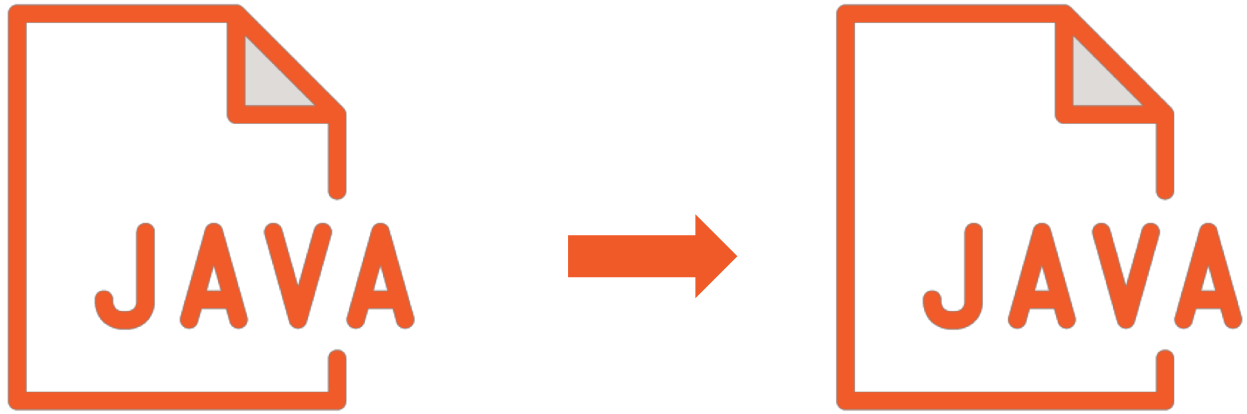
Java 8



Java 7

Java 8

Default: server mode



Java 7

Default: server mode

Java 8

Default: tiered mode

Java 7

Default: server mode

Java 8

Default: tiered mode

Java 8

Set: server mode

JIT Tuning Flags

```
java -server
```

Setting the Compilation Mode in Java 7

```
java -client
```

Setting the Compilation Mode in Java 7

```
java -XX:TieredStopAtLevel=1
```

Setting the Client Compilation Mode in Java 8

Available levels

- **Level 0: Interpreted mode**
- **Level 1: C1 compiled with no profiling**

```
java -XX:-TieredCompilation
```

Setting the Server Compilation Mode in Java 8

Hot Method Definition

`i`: invocation counter

The number of times a method has been called

`b`: backedge counter

The number of times any loops in a method have branched back

Hot Method Definition – Tiered Compilation

```
i > Tier3InvocationThreshold  ||

(i > Tier3MinInvocationThreshold && i + b >
      Tier3CompileThreshold)
```

Hot Method Definition – Tiered Compilation

```
i > 200 ||
(i > 100 && i + b > 2000)
```

Hot Method Definition – Tiered Compilation

```
i > 5000 ||
(i > 600 && i + b > 15000)
```

```
java -XX:Tier4InvocationThreshold=4000 -
XX:Tier4CompileThreshold=10000
```

Lowering the Compilation Thresholds in Tiered Mode

Hot Method Definition – Server Compilation

```
i + b > CompileThreshold
```

Hot Method Definition – Server Compilation

```
i + b > 10000
```

```
java -XX:-TieredCompilation -XX:CompileThreshold=8000
```

Lowering the Compilation Thresholds in Server Mode

- **Methods get compiled earlier**
- **“Warm” methods get compiled**

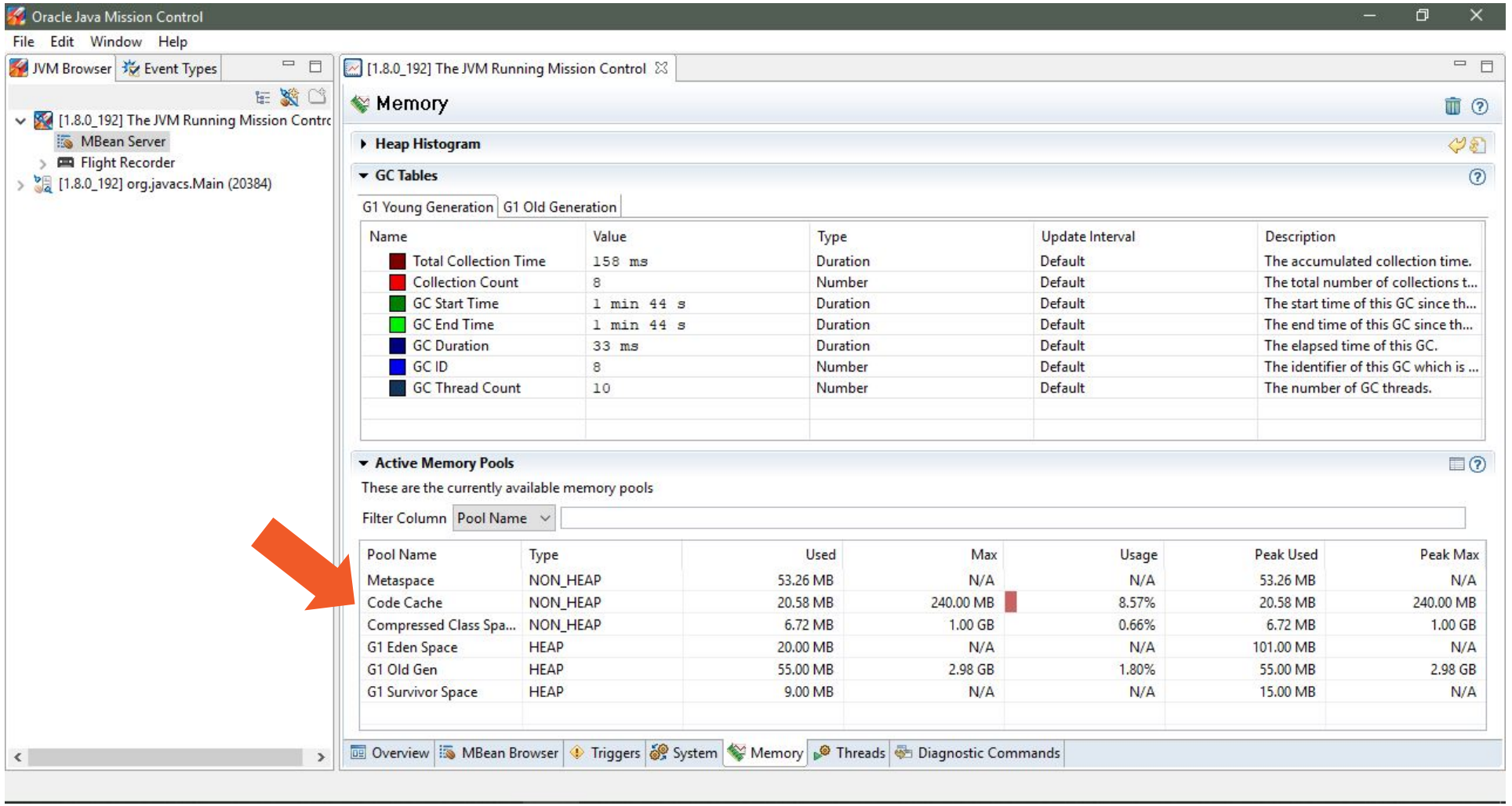
Code Cache

An area of native memory where compiled code is stored for future executions

If the code cache is filled up no more JIT compilation is done!

In Java 8 and up the reserved code cache size is usually sufficient by default

Code Cache Monitoring



```
java -XX:ReservedCodeCacheSize=<N>
```

Setting the Code Cache Size

Java 9 Code Cache Changes

JVM internal code

Profiled (lightly optimized) code

Non-profiled (fully optimized) code

Java 9 Code Cache Changes

JVM internal code

Profiled (lightly optimized) code

Non-profiled (fully optimized) code

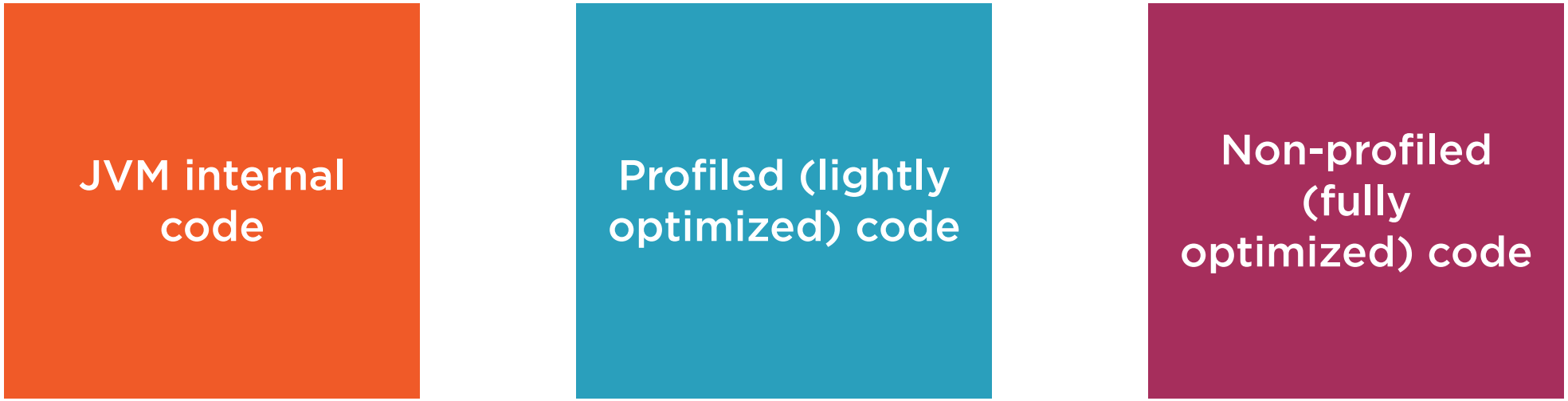
Java 9 Code Cache Changes

JVM internal code

Profiled (lightly optimized) code

Non-profiled (fully optimized) code

Java 9 Code Cache Changes



```
java -XX:ReservedCodeCacheSize=<N>
```

Setting the Code Cache Size in Java 9

Will turn off the segmented code cache feature in Java 9 and above

Introduction to Garbage Collection

Garbage Collection

Mechanism by which the JVM reclaims memory on the behalf of the application when it's no longer needed

Consists of:

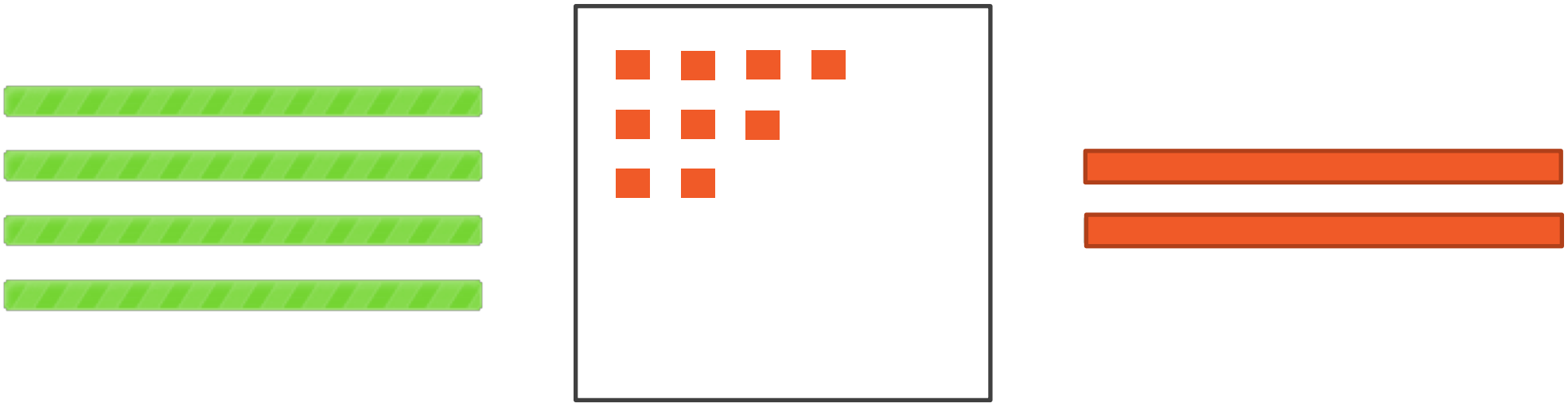
- Finding no longer reachable objects
- Deallocating the memory
- Compacting the heap

GC Pauses

GC threads are used to collect garbage and compact the heap

During this process application threads are typically stopped so that object references aren't lost

Stop-The-World Pauses

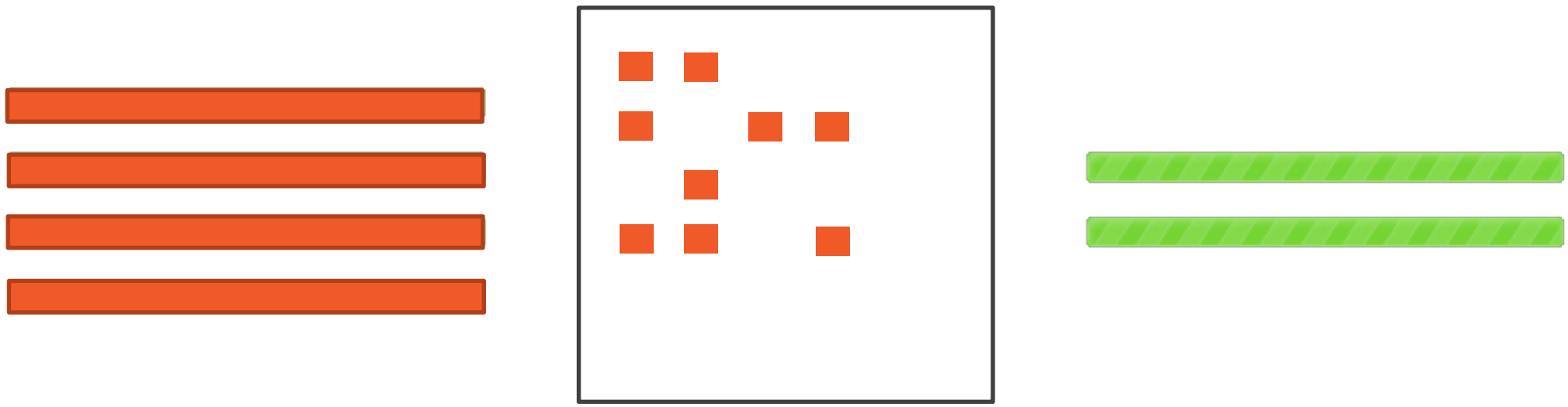


Application threads

Objects in heap

GC threads

Stop-The-World Pauses

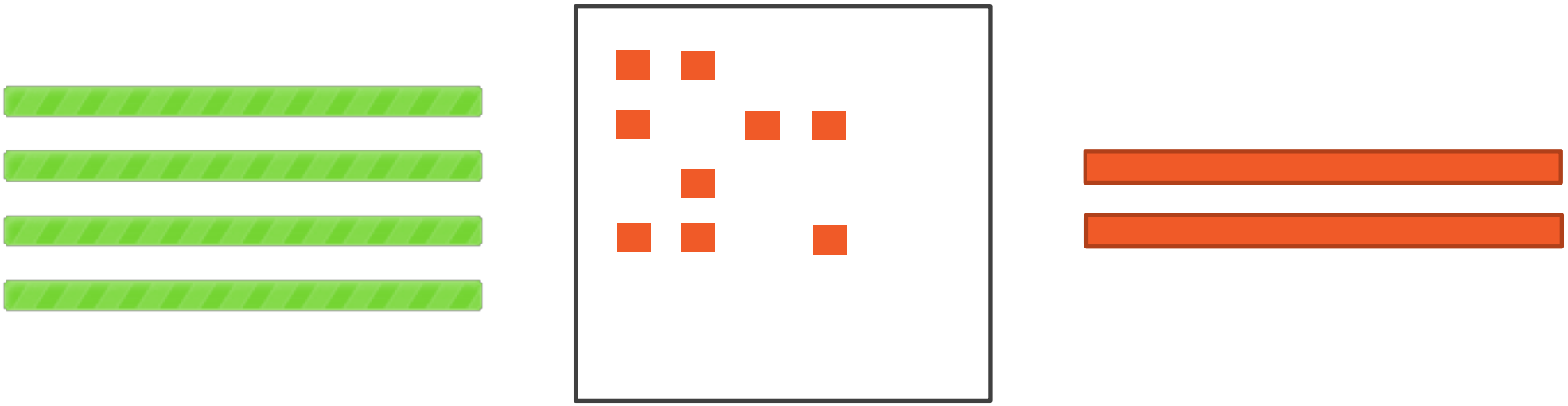


Application threads

Objects in heap

GC threads

Stop-The-World Pauses



Application threads

Objects in heap

GC threads

OpenJDK Garbage Collectors

Serial

Parallel

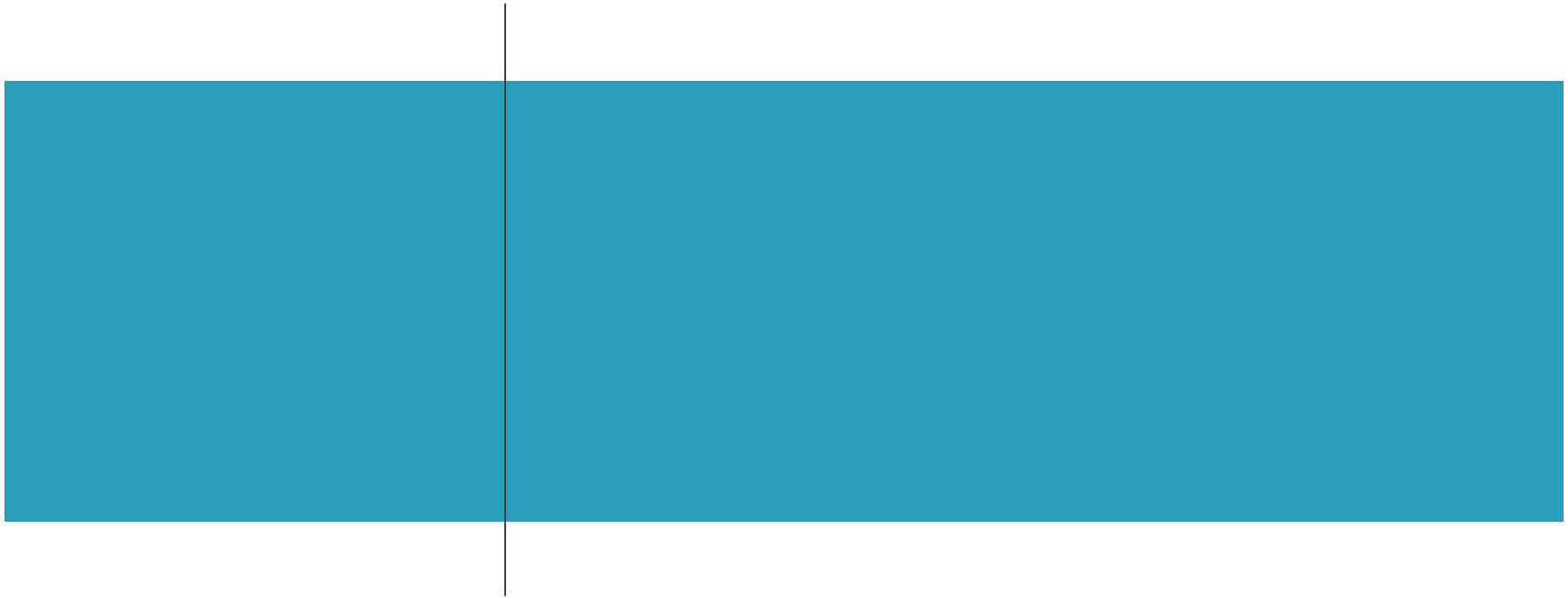
Concurrent Mark Sweep

Garbage First

Generational Garbage Collection

Heap

Generational Garbage Collection



Heap

Generational Garbage Collection

Young Generation

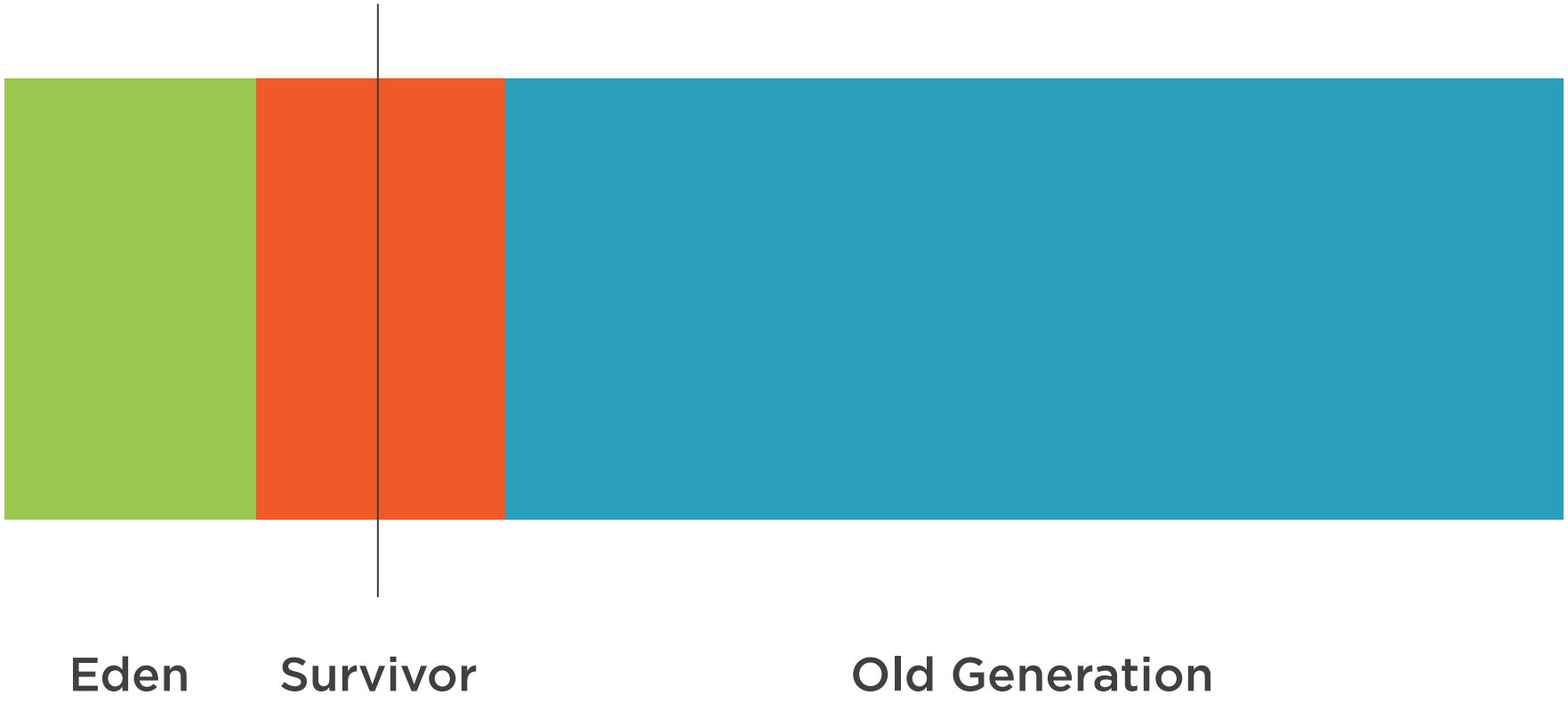
Old Generation

Generational Garbage Collection

Generational Garbage Collection

Eden **Survivor** **Old Generation**

Generational Garbage Collection



Generational Garbage Collection

Eden S0 S1 Old Generation

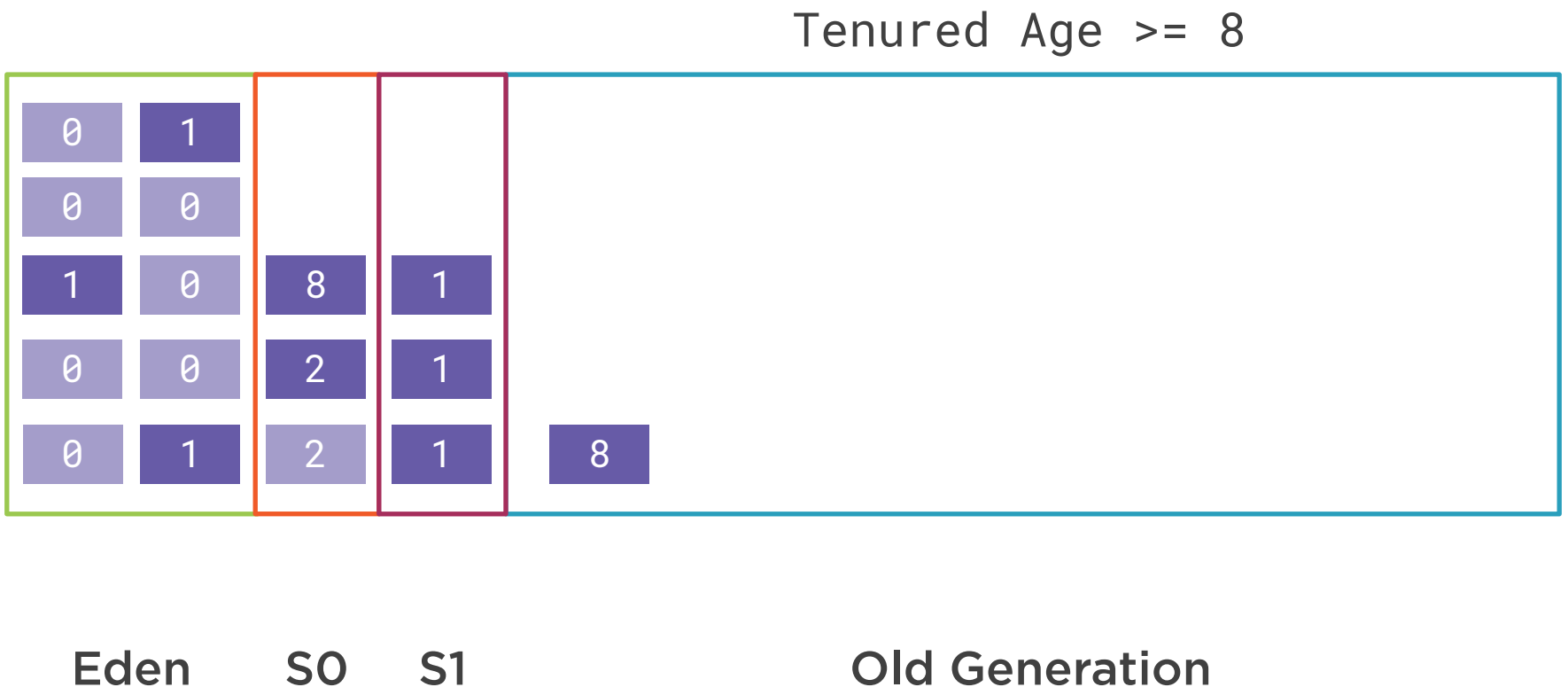
Generational Garbage Collection

Generational Garbage Collection

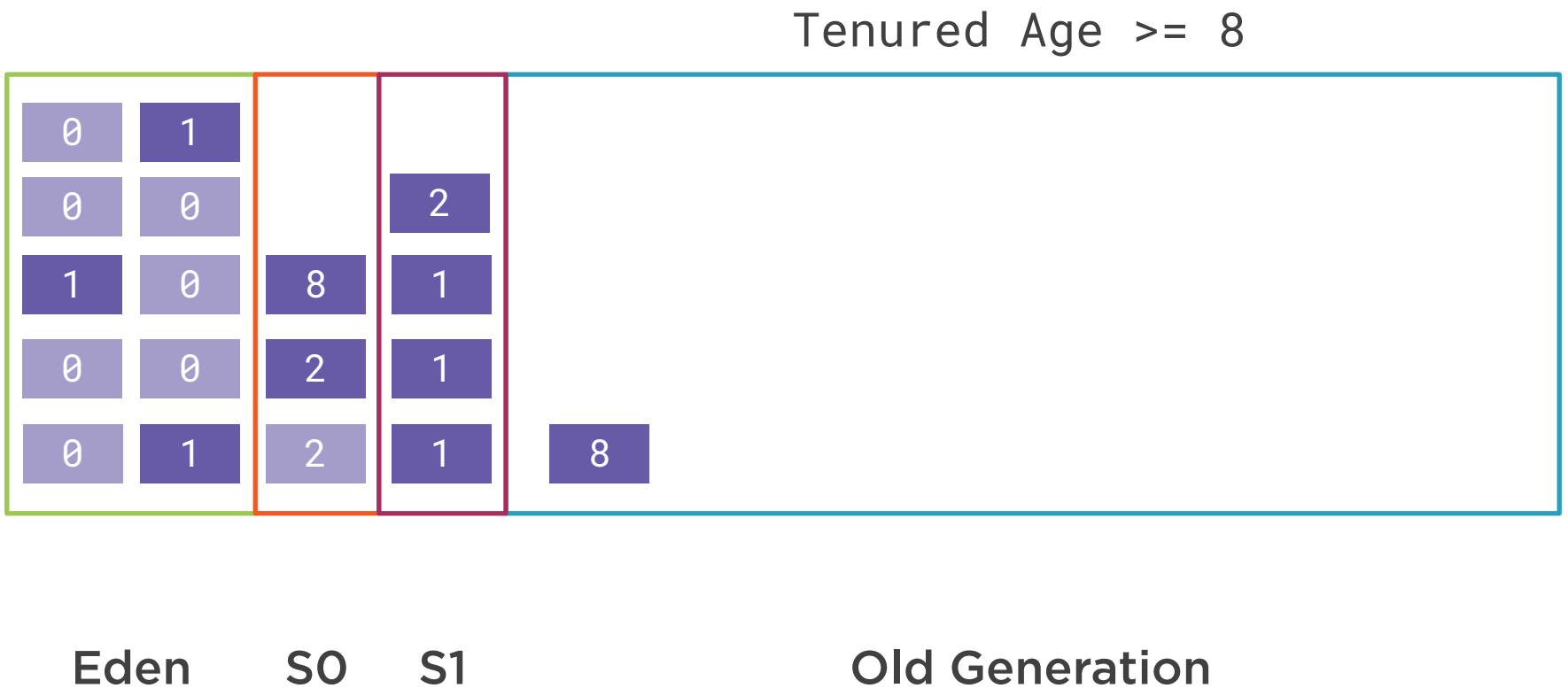
Generational Garbage Collection

Generational Garbage Collection

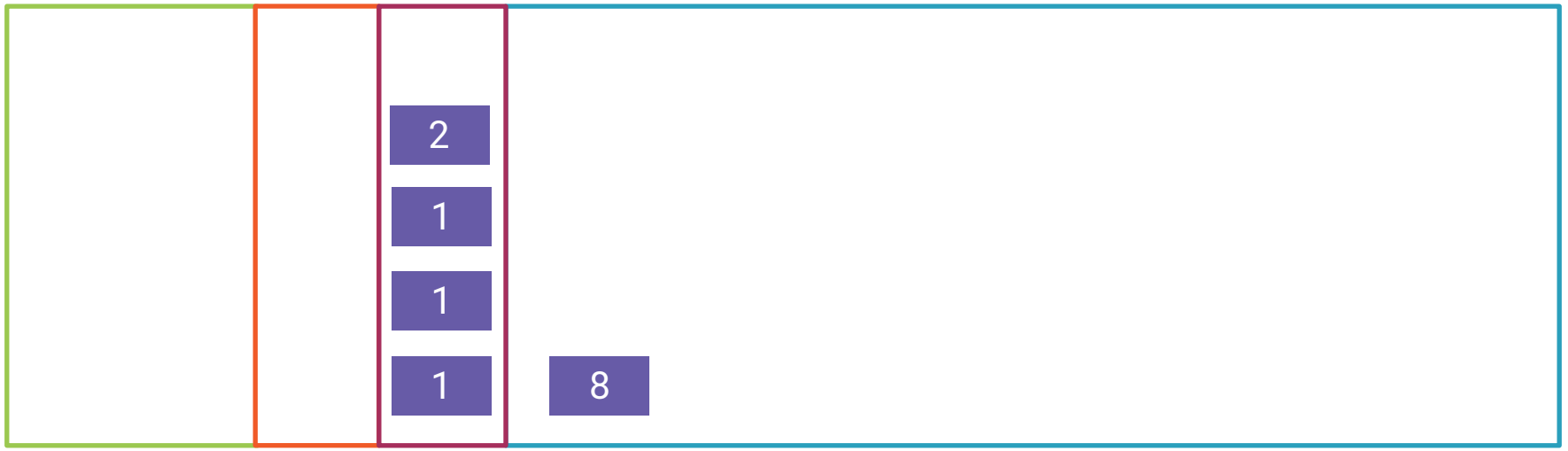
Generational Garbage Collection



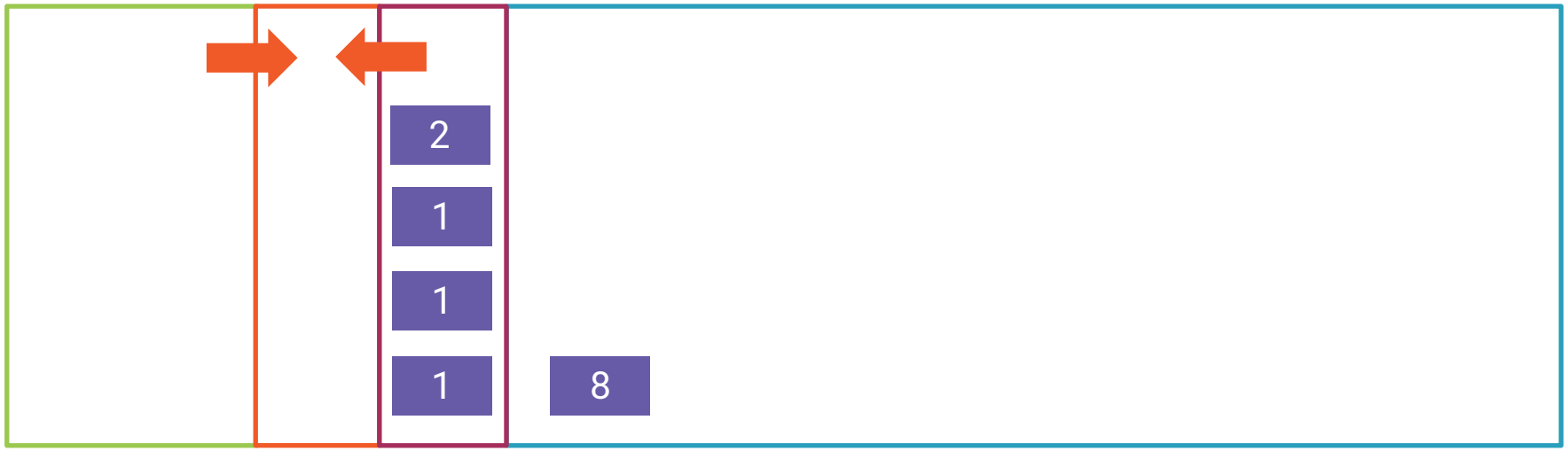
Generational Garbage Collection



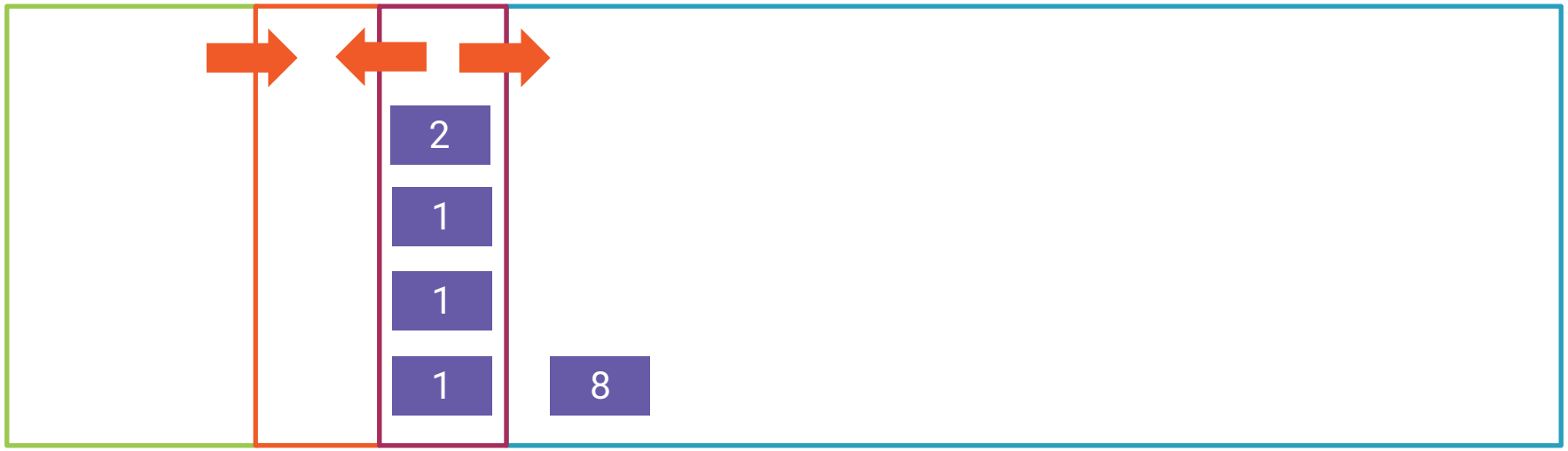
Generational Garbage Collection



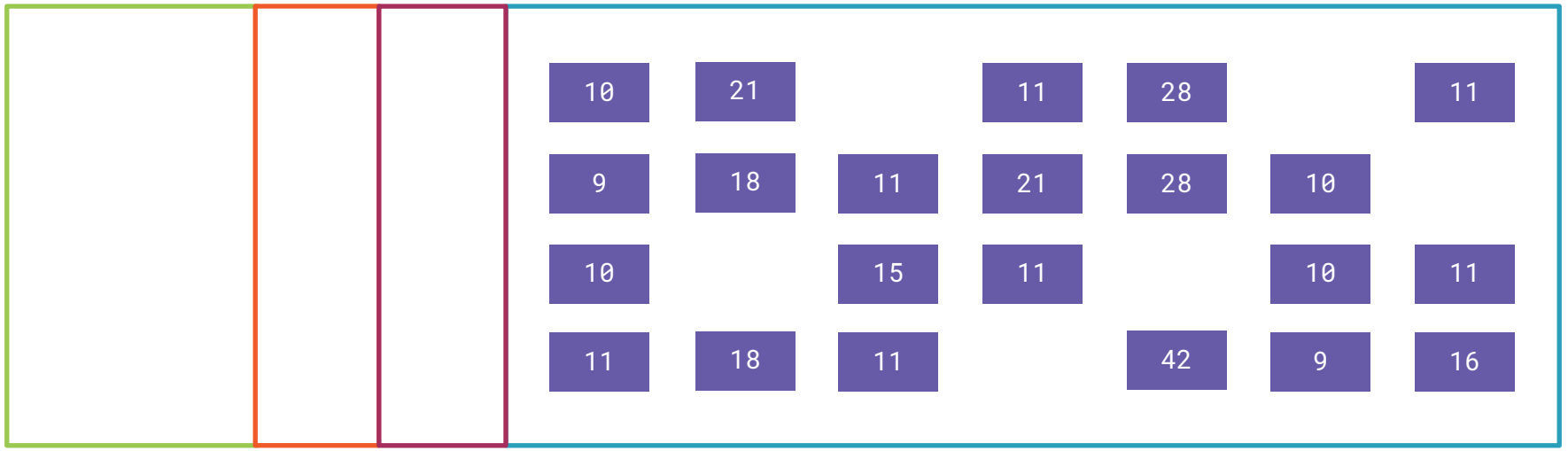
Generational Garbage Collection



Generational Garbage Collection



Generational Garbage Collection



Basic Collectors

- Stop all application threads
- Mark unreachable objects
- Free the memory
- Compact the heap
- Resume application threads

Advanced Collectors

Scan for unreachable objects while the application threads are still running

Only pause threads to free the memory and compact the heap

Known as concurrent collectors, mostly-concurrent collectors, low-pause collectors

Choosing a Garbage Collector

Factors in Choosing a Garbage Collector

Heap size / Amount of live data

Number and speed of available processors

Pause time requirements

Serial Collector

Uses one thread to process the heap

To enable: `-XX:+UseSerialGC`

Use when:

- Only 1 CPU/vCPU available and no pause time requirements

- Multiple small JVMs on a single machine (more than the number of CPUs)

- Small live data set (up to 100MB)

Parallel Collector

Uses multiple threads to process the heap

Fully stops all application threads which can result in long pause times but higher throughput

Use when:

- You have a batch application

To enable in Java 9↑: `-XX:+UseParallelGC`

Default in Java 7 and 8

CMS Collector

Can trace reachable objects and cleanup unreachable objects while application threads are running

GC threads may compete with application threads for CPU

To enable: `-XX:+UseConcMarkSweepGC`

Has been superseded by G1GC and is deprecated in Java 9↑

G1GC Collector

Default collector in Java 9 and up

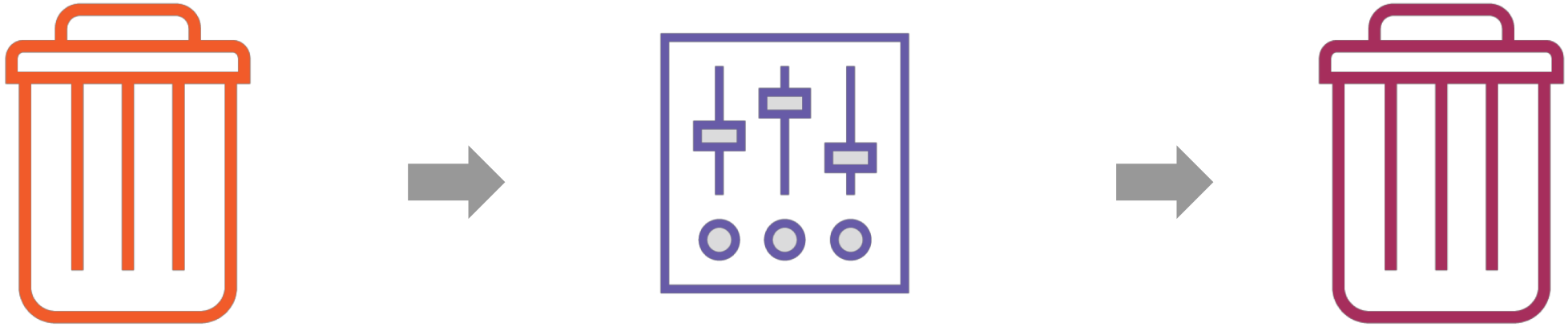
Designed for multiprocessor machines with large heaps. Tries to achieve the best balance between throughput and latency

GC threads mark unreachable objects concurrently while application threads are running

Use when: you have an interactive application with low pause time requirements

To enable in Java 7/8: `-XX:+UseG1GC`

GC Tuning Process



Select a collector

Tune heap and collector settings

Try a different collector

Shenandoah Collector

Takes the next step beyond G1GC

Can mark unreachable objects and move reachable objects concurrently while application threads are running

Can produce 10X reduction in pause times with only a 10% throughput decrease

Measuring Garbage Collector Performance

GC Logging

GC Log info:

- When collections were run

- How long they lasted

- What were the pause times

- How much memory was reclaimed

- How many objects were promoted to the old generation

- Etc.

```
java -XX:+PrintGCDetails -XX:+PrintGCDateStamps -
Xloggc:<file-path>
```

Turning on GC Logging in Java 7 / 8

```
java -XX:+PrintGCDetails -XX:+PrintGCDateStamps -
Xloggc:<file-path>
```

Turning on GC Logging in Java 7 / 8

```
java -XX:+PrintGCDetails -XX:+PrintGCDateStamps -
Xloggc:<file-path>
```

Turning on GC Logging in Java 7 / 8

```
java -XX:+PrintGCDetails -XX:+PrintGCDateStamps -
Xloggc:<file-path>
```

Turning on GC Logging in Java 7 / 8

```
java -XX:+PrintGCDetails -XX:+PrintGCDateStamps -
Xloggc:<file-path> -XX:+UseGCLogFileRotation -
XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=10M
```

Turning on GC Logging with File Rotation in Java 7 / 8

```
java -X:log:gc
```

Turning on GC Logging in Java 9 and Up

```
java -X:log:gc
```

Turning on GC Logging in Java 9 and Up

```
java -X:log:gc:file=<file-path>
```

Turning on GC Logging in Java 9 and Up

```
java -X:log:gc:file=<file-path>:filecount=10,filesize=10M
```

Turning on GC Logging in Java 9 and Up

```
java -X:log:gc*:file=<file-path>:filecount=10,filesize=10M
```

Turning on GC Logging in Java 9 and Up

```
jcmd <pid> VM.log what=gc output=<file-path>
```

Dynamically Turning on GC Logging in Java 9 and Up

```
[150.732s][info][gc,start  ] GC(74) Pause Young (Normal) (G1 Evacuation Pause)
[150.732s][info][gc,task   ] GC(74) Using 4 workers of 4 for evacuation
[150.739s][info][gc,heap   ] GC(74) Eden regions: 316->0(316)
[150.739s][info][gc,heap   ] GC(74) Survivor regions: 3->3(40)
[150.739s][info][gc,heap   ] GC(74) Old regions: 2007->2007
[150.739s][info][gc,heap   ] GC(74) Humongous regions: 4->4
[150.739s][info][gc        ] GC(74) Pause Young (Normal) (G1 Evacuation Pause) 2329M->2013M(2596M) 6.898ms
[150.739s][info][gc,cpu    ] GC(74) User=0.06s Sys=0.00s Real=0.01s
```

```
[150.732s][info][gc,start  ] GC(74) Pause Young (Normal) (G1
Evacuation Pause)

[150.732s][info][gc,task   ] GC(74) Using 4 workers of 4 for
evacuation

[150.739s][info][gc,heap   ] GC(74) Eden regions: 316->0(316)

[150.739s][info][gc,heap   ] GC(74) Survivor regions: 3->3(40)

[150.739s][info][gc,heap   ] GC(74) Old regions: 2007->2007

[150.739s][info][gc,heap   ] GC(74) Humongous regions: 4->4

[150.739s][info][gc        ] GC(74) Pause Young (Normal) (G1
Evacuation Pause) 2329M->2013M(2596M) 6.898ms

[150.739s][info][gc,cpu    ] GC(74) User=0.06s Sys=0.00s
Real=0.01s
```

```
[150.732s][info][gc,start  ] GC(74) Pause Young (Normal) (G1
Evacuation Pause)

[150.732s][info][gc,task   ] GC(74) Using 4 workers of 4 for
evacuation

[150.739s][info][gc,heap   ] GC(74) Eden regions: 316->0(316)

[150.739s][info][gc,heap   ] GC(74) Survivor regions: 3->3(40)

[150.739s][info][gc,heap   ] GC(74) Old regions: 2007->2007

[150.739s][info][gc,heap   ] GC(74) Humongous regions: 4->4

[150.739s][info][gc        ] GC(74) Pause Young (Normal) (G1
Evacuation Pause) 2329M->2013M(2596M) 6.898ms

[150.739s][info][gc,cpu    ] GC(74) User=0.06s Sys=0.00s
Real=0.01s
```

```
[150.732s][info][gc,start  ] GC(74) Pause Young (Normal) (G1
Evacuation Pause)

[150.732s][info][gc,task   ] GC(74) Using 4 workers of 4 for
evacuation

[150.739s][info][gc,heap   ] GC(74) Eden regions: 316->0(316)

[150.739s][info][gc,heap   ] GC(74) Survivor regions: 3->3(40)

[150.739s][info][gc,heap   ] GC(74) Old regions: 2007->2007

[150.739s][info][gc,heap   ] GC(74) Humongous regions: 4->4

[150.739s][info][gc        ] GC(74) Pause Young (Normal) (G1
Evacuation Pause) 2329M->2013M(2596M) 6.898ms

[150.739s][info][gc,cpu    ] GC(74) User=0.06s Sys=0.00s
Real=0.01s
```

JEP 158: Unified JVM Logging

Introduce a common logging system for all components of the JVM

Analyzing the GC Log File

Manual inspection

3rd party tools

Analyzing the GC Log File

Manual inspection

3rd party tools

GCEasy.io

GCViewer

Demo

GCEasy.io

Garbage Collection Tuning

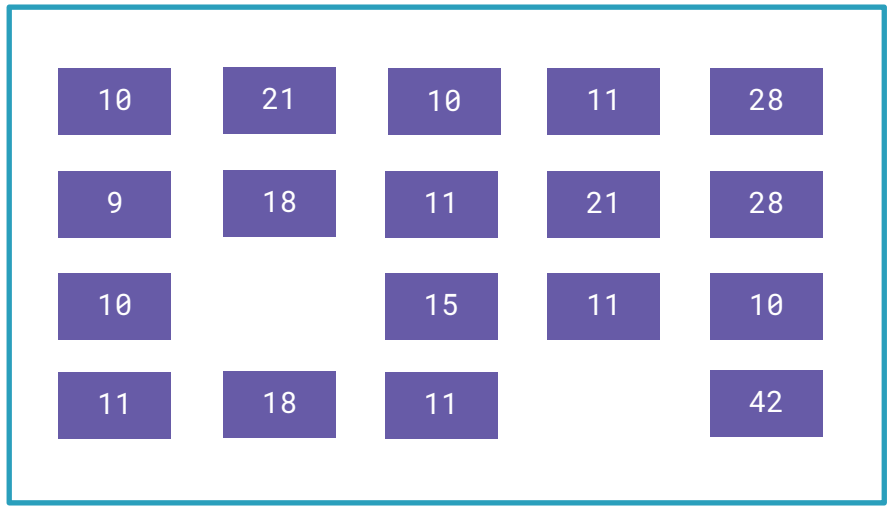
Tuning the Size of the Heap

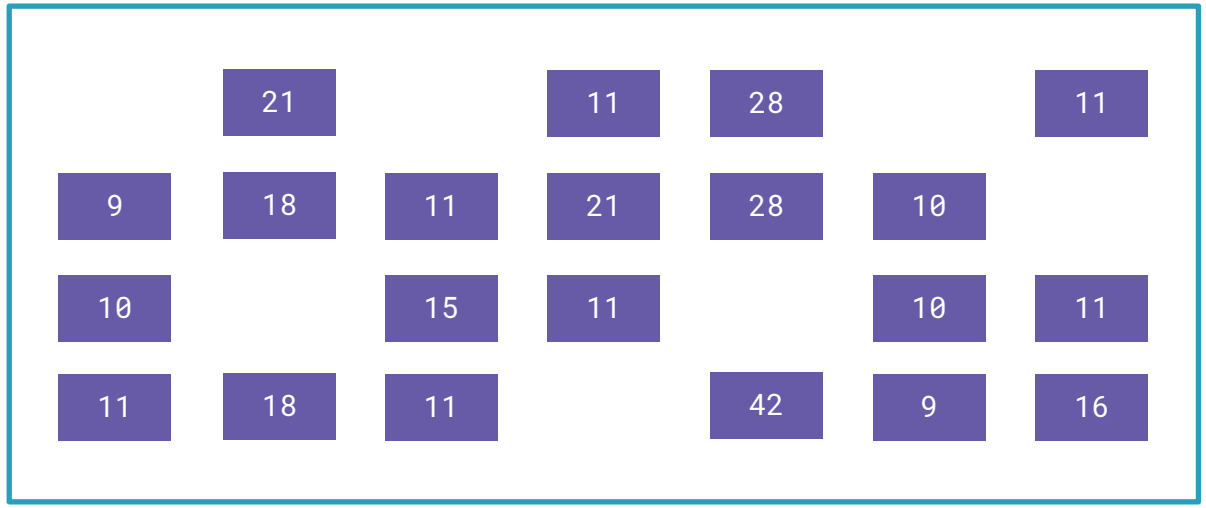
If the heap is too small: there'll be too many GCs in order to free memory and application throughput would suffer

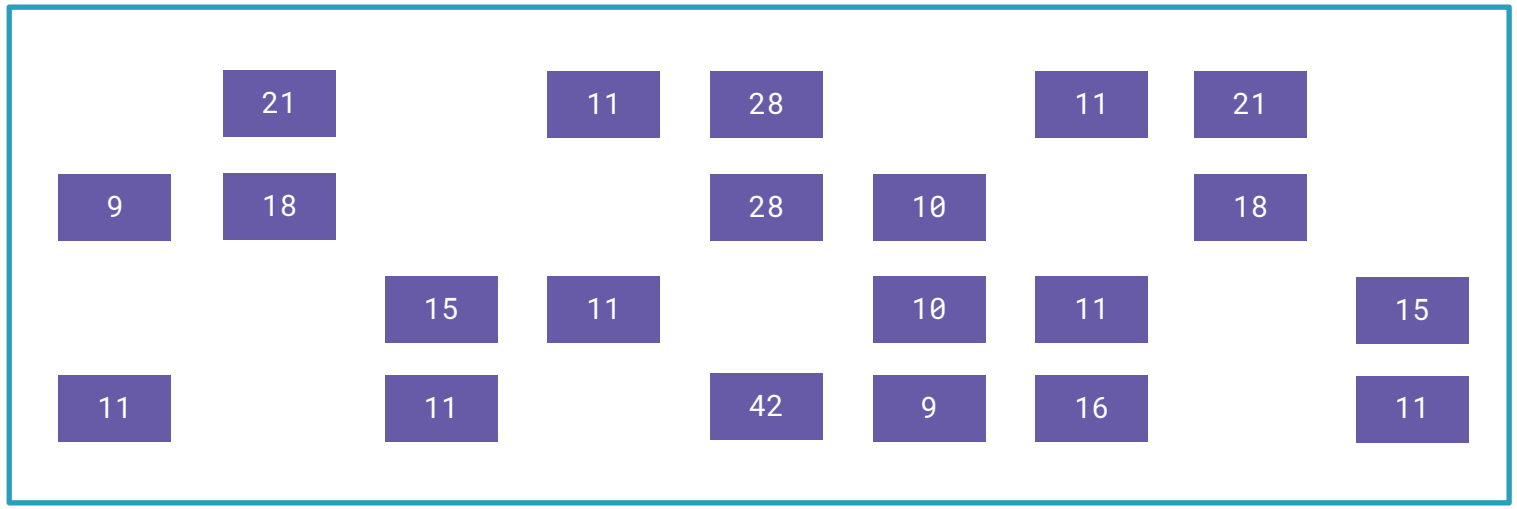
If the heap is too large: GC pauses would take too long and response time metrics would be affected

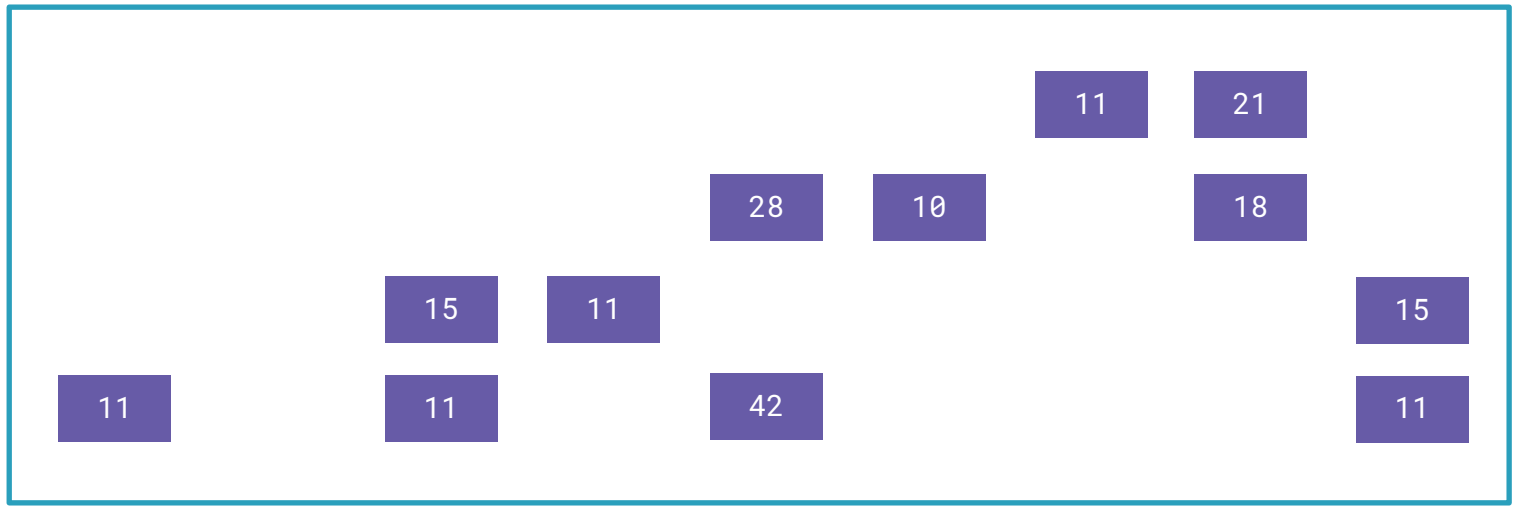
```
java -Xms<i>N</i> -Xmx<i>N</i>
```

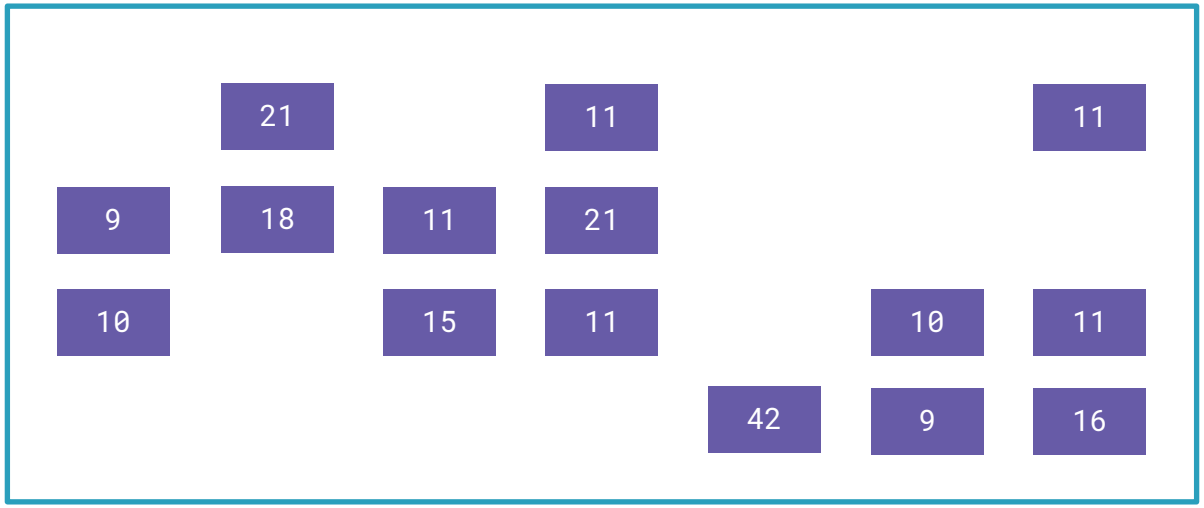
Setting the Heap Size


10
21
10
11
28
9
18
11
21
28
10
15
11
10
11
18
11
42

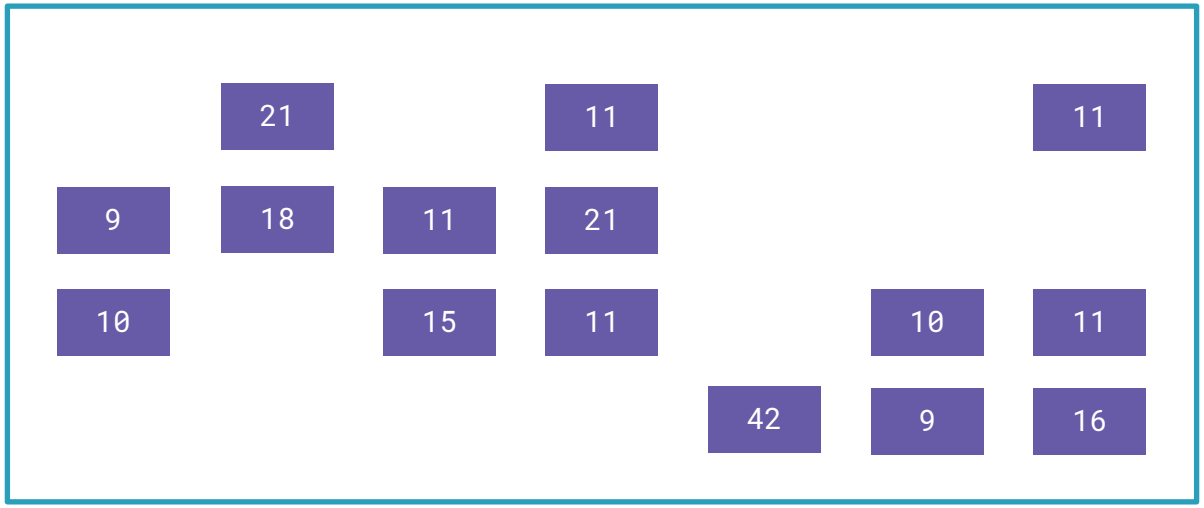

21
11
28
11
9
18
11
21
28
10
10
15
11
10
11
11
18
11
42
9
16


21
11
28
11
21
9
18
28
10
18
15
11
10
11
15
11
11
42
9
16
11


11
21
28
10
18
15
11
15
11
11
42
11


21
11
11
9
18
11
21
10
15
11
10
11
42
9
16

Adaptive Sizing



Turning off Adaptive Sizing

Should do so only if you've finely tuned your application's GC behavior and sizes

Set flag: `-XX:-UseAdaptiveSizePolicy`

`-Xms6g -Xmx6g`

`-XX:NewSize=1g -XX:MaxNewSize=1g`

Max Heap Sizing

Max heap size should not exceed the machine's physical memory

Heap should be about 30% occupied after a full GC

MaxGCPauseMillis

Example: `-XX:MaxGCPauseMillis=350`

Sets a target for the maximum GC pause time

In the Parallel collector:

- Adjusts the size of the young and old generation

- Adjusts the size of the heap

MaxGCPauseMillis

In the G1GC collector:

- Adjusts the size of the young and old generation

- Adjusts the size of the heap

- Starts background processing sooner

- Changes the tenuring threshold

- Adjusts the number of old generation regions processed

G1GC default: `200 ms`

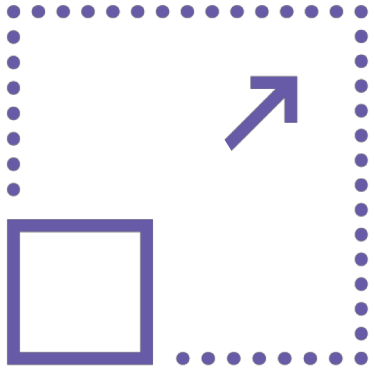
Concurrent Garbage Collection Failures

Since the application threads are still running, they may produce garbage faster than the GC threads can clean them

Known as:

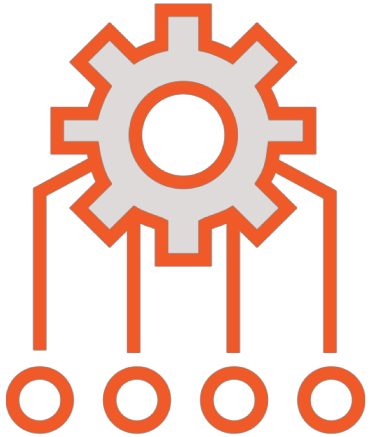
- Concurrent mode failure
- Promotion failure
- Evacuation failure

Solving Concurrent GC Failures

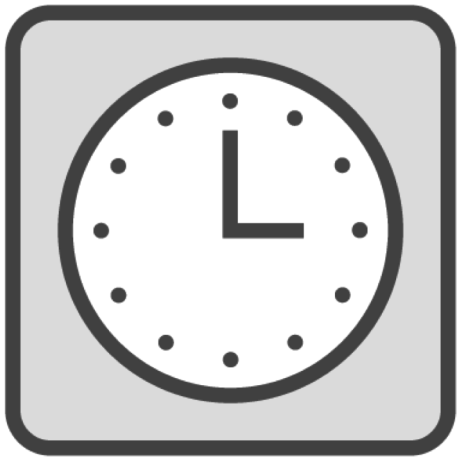


Increase the size of the heap

Start background processing earlier



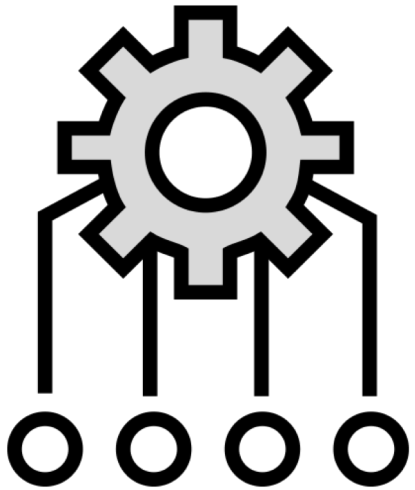
Use more background threads



Reduce the threshold at which the G1 cycle is triggered: -
`XX:InitiatingHeapOccupancyPercent=45`

A GC cycle is triggered when the heap becomes 45% filled

If set too low, GCs will happen too frequently



To increase the number of background threads: `-XX:ConcGCThreads=4`

Default value: `ConcGCThreads = (ParallelGC + 2) / 4`