Java 垃圾回收

#从入门到吊打面试官

内容100%正确

100%够过面试

Java 之前的垃圾回收

C　malloc()/ralloc()/ calloc()/ free()

C++　new/destructors

Java 的垃圾回收

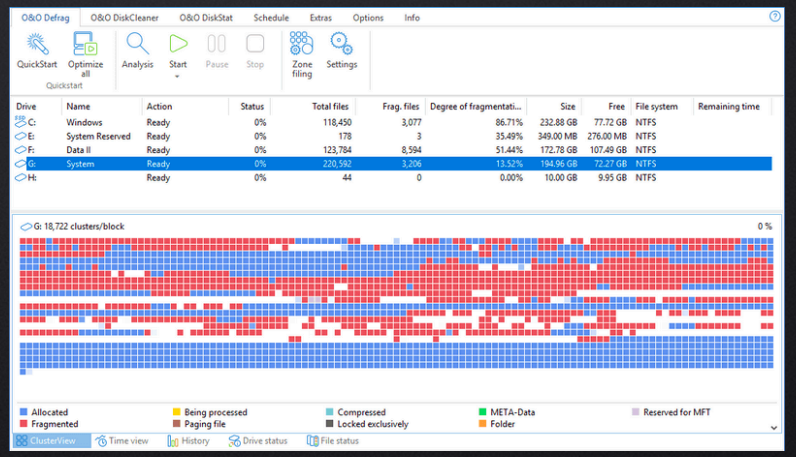
何为垃圾	**Heap** 中不再被引用的对象（Object） while(true) { new Student(); }
基本原理	移除不再被使用的对象（Object）
基本假设	大多数对象的生命周期都很短
两行代码	System.gc(); java.lang.OutOfMemoryError

GC 的3个方法

Mark　从heap的根节点开始遍历整个heap，标记还“活着”的Object

Delete/Sweep　删除heap中的Object

Compating　碎片整理

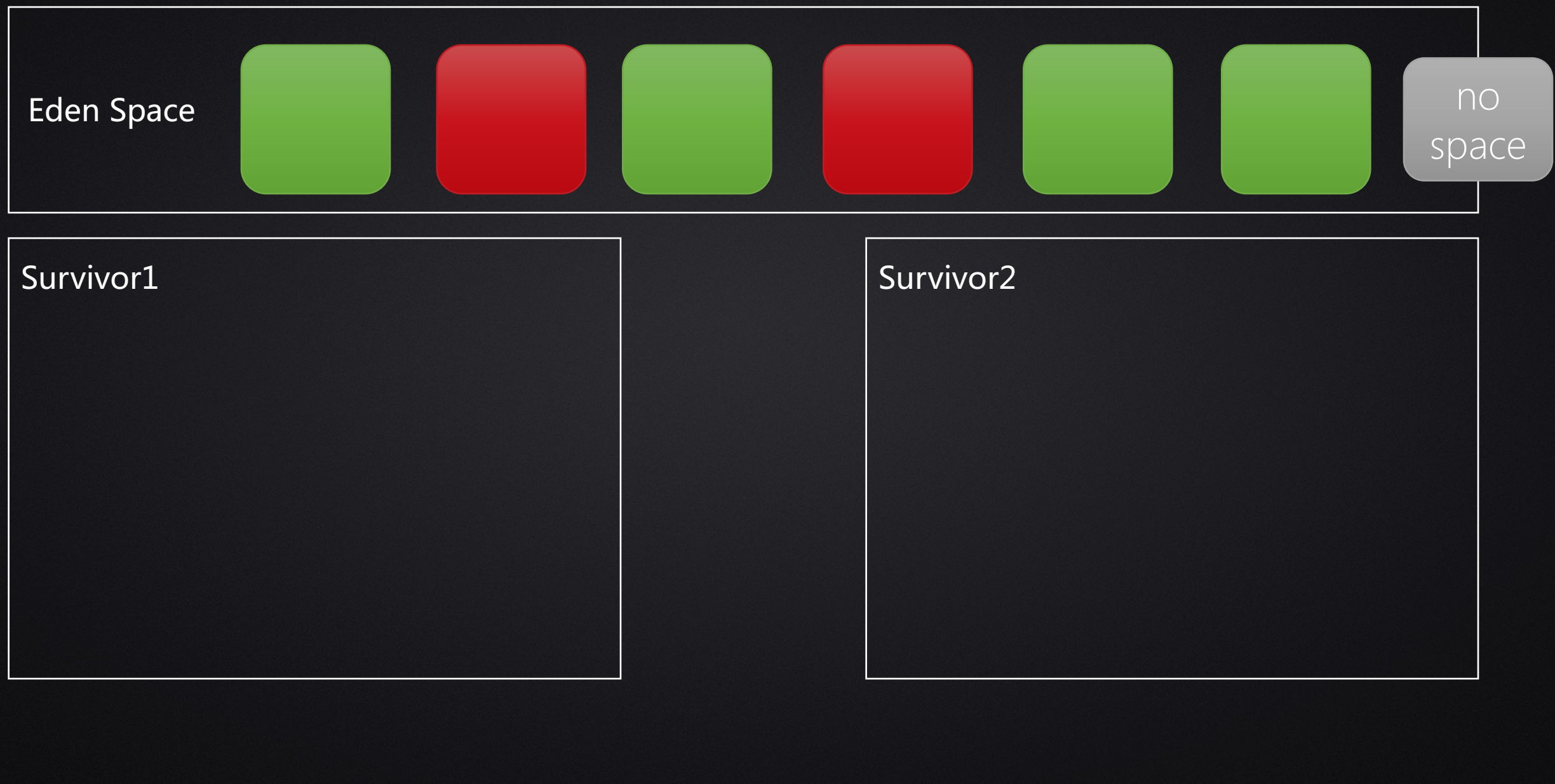


Generational collectors（分代收集器）

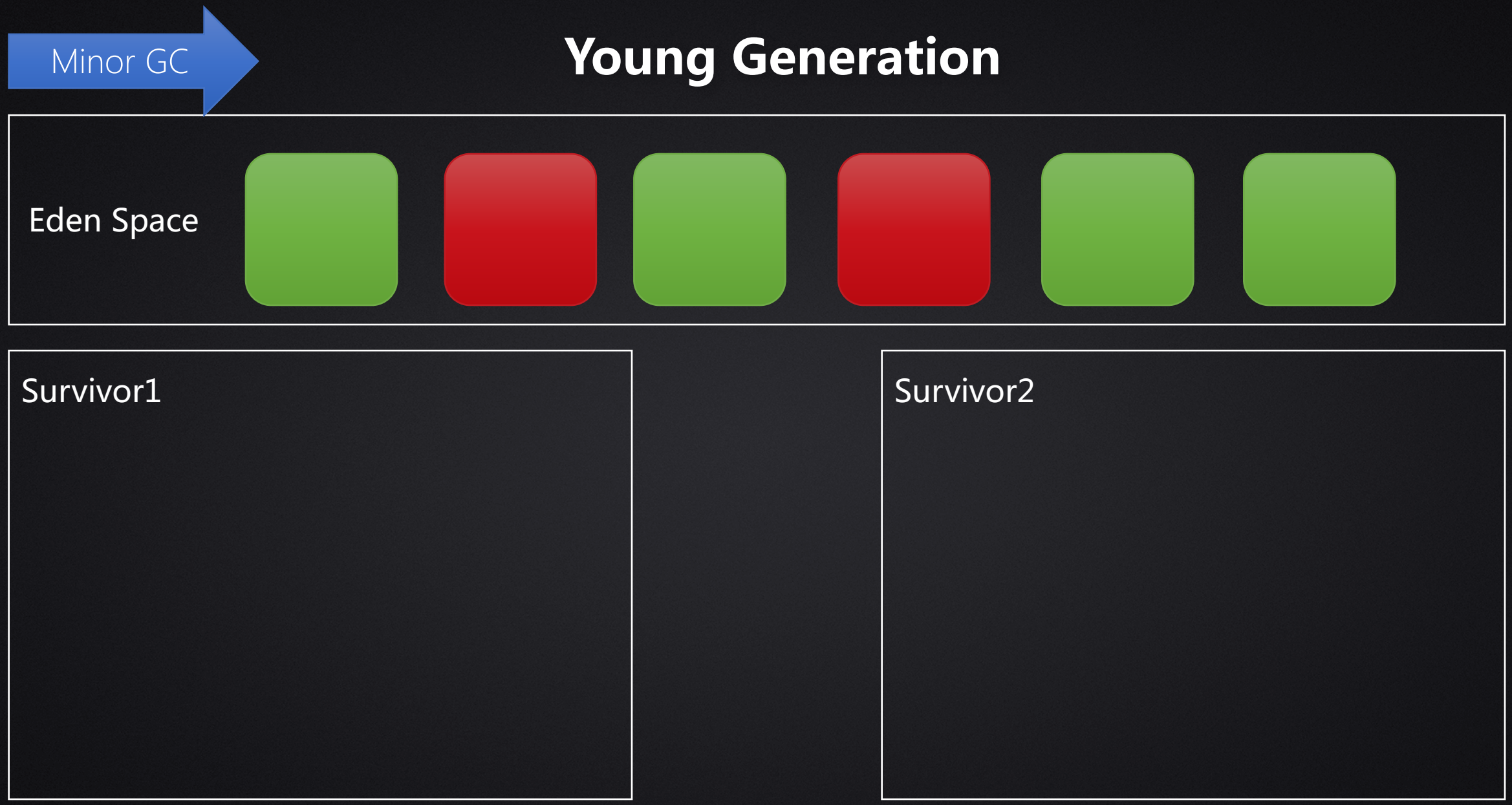
Young Generation

Young Generation

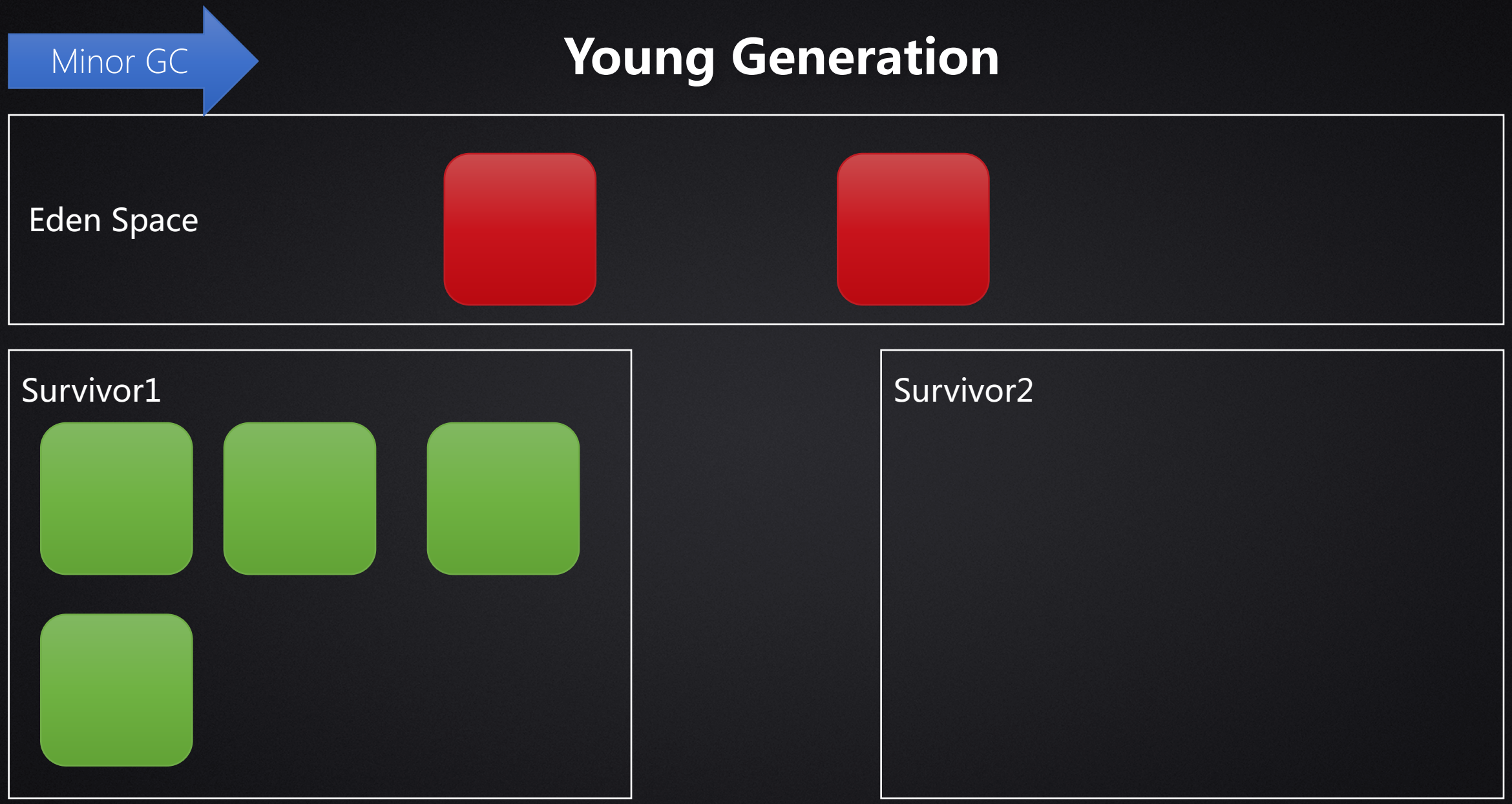
Young Generation



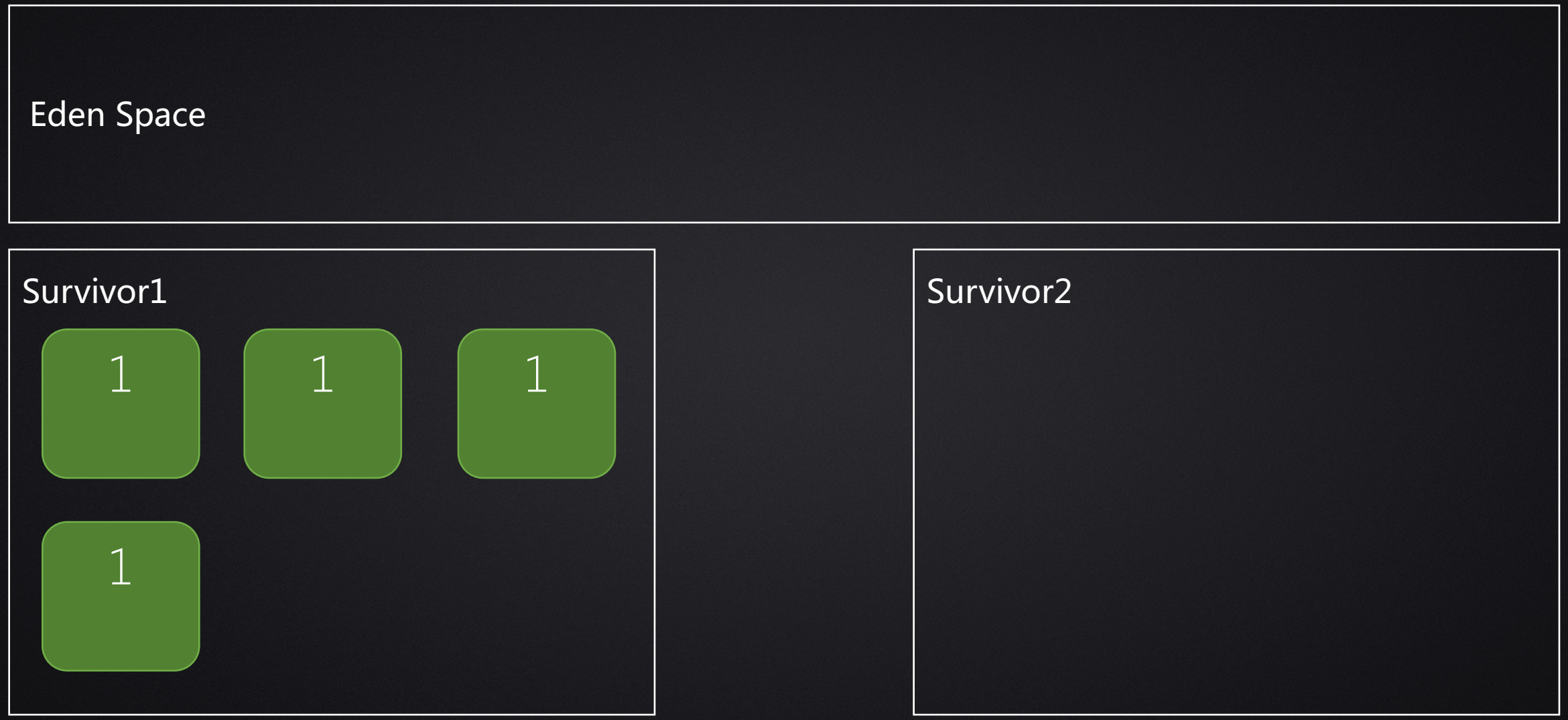
Young Generation



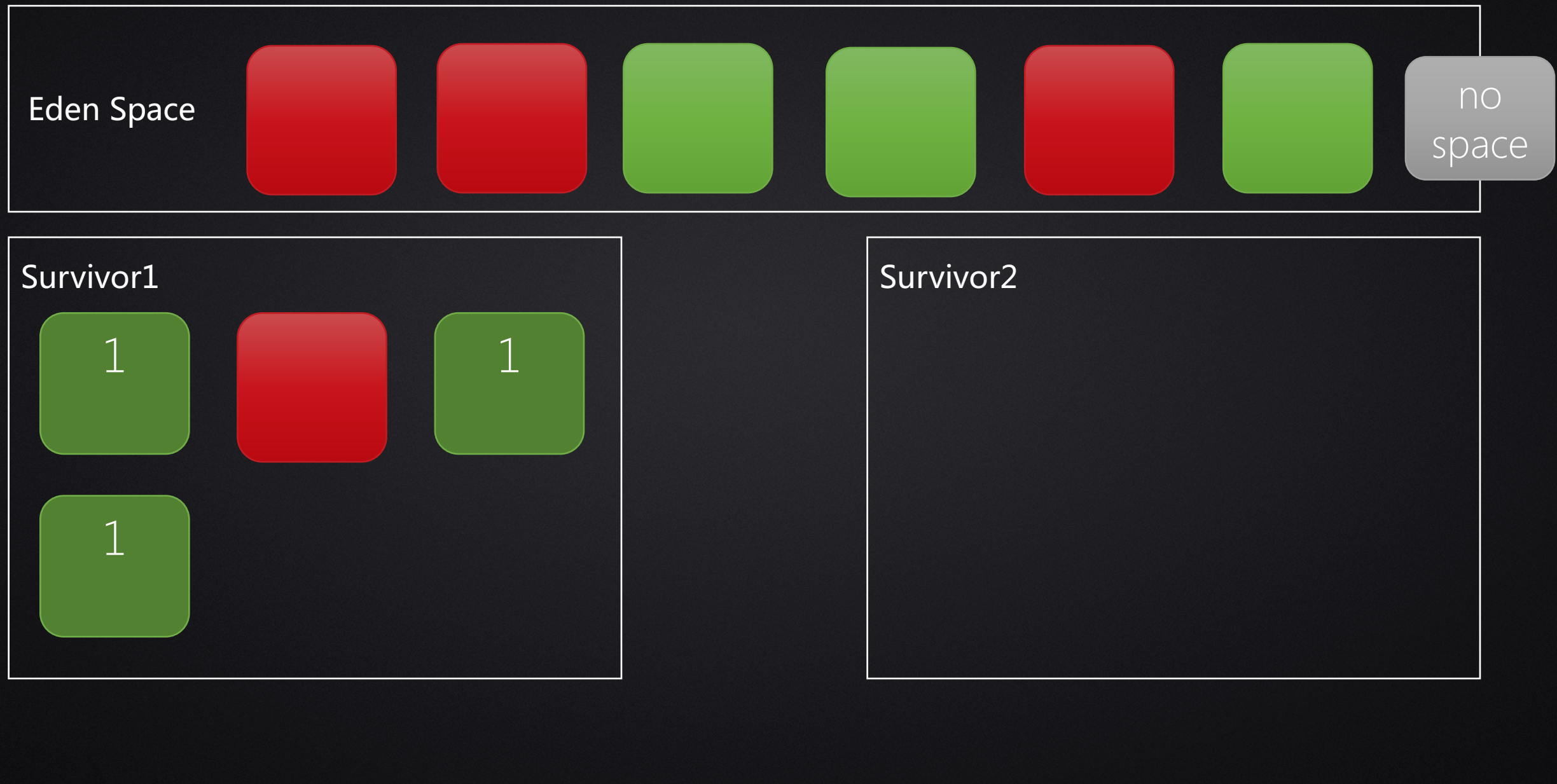
Young Generation



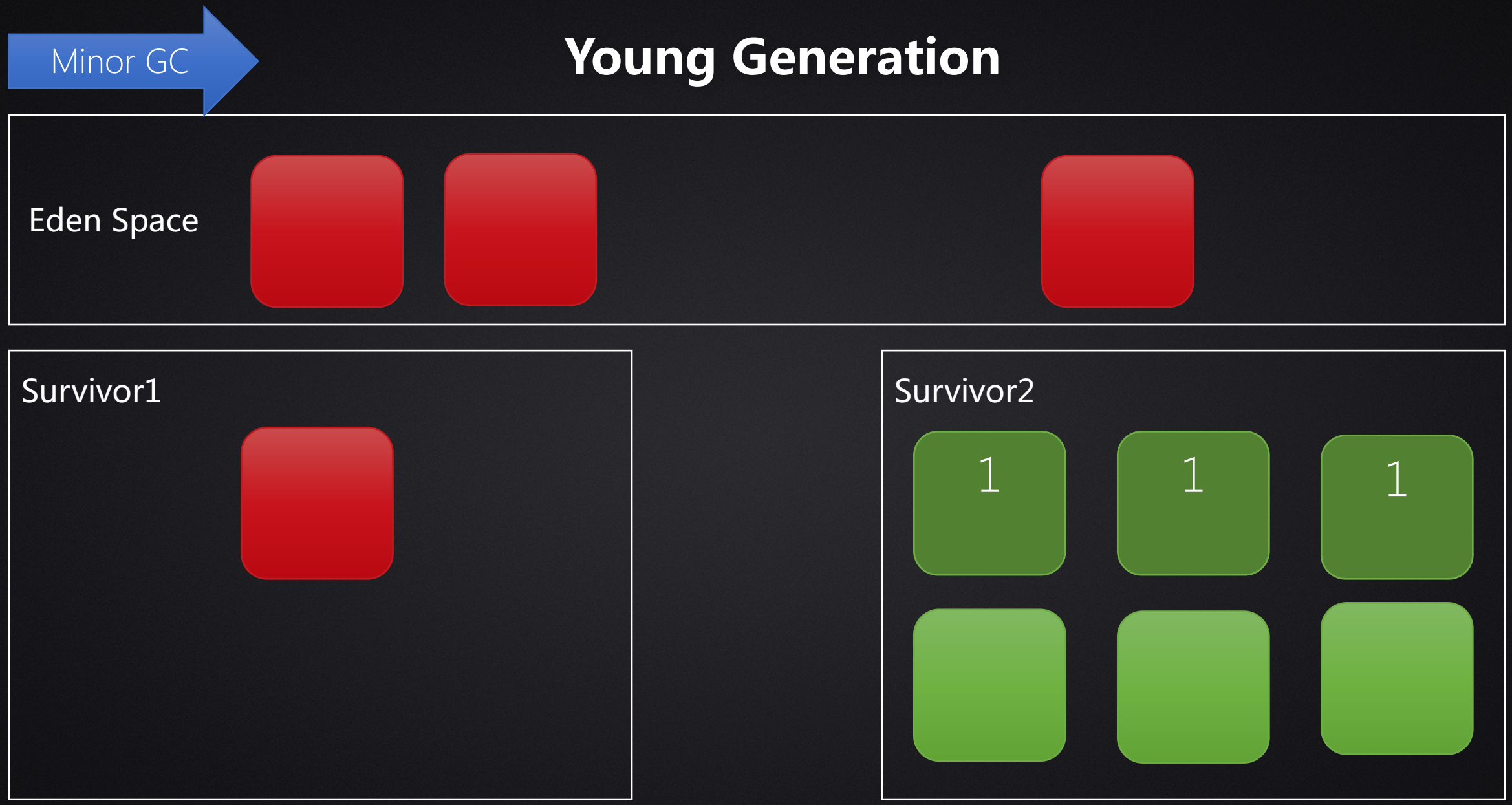
Young Generation



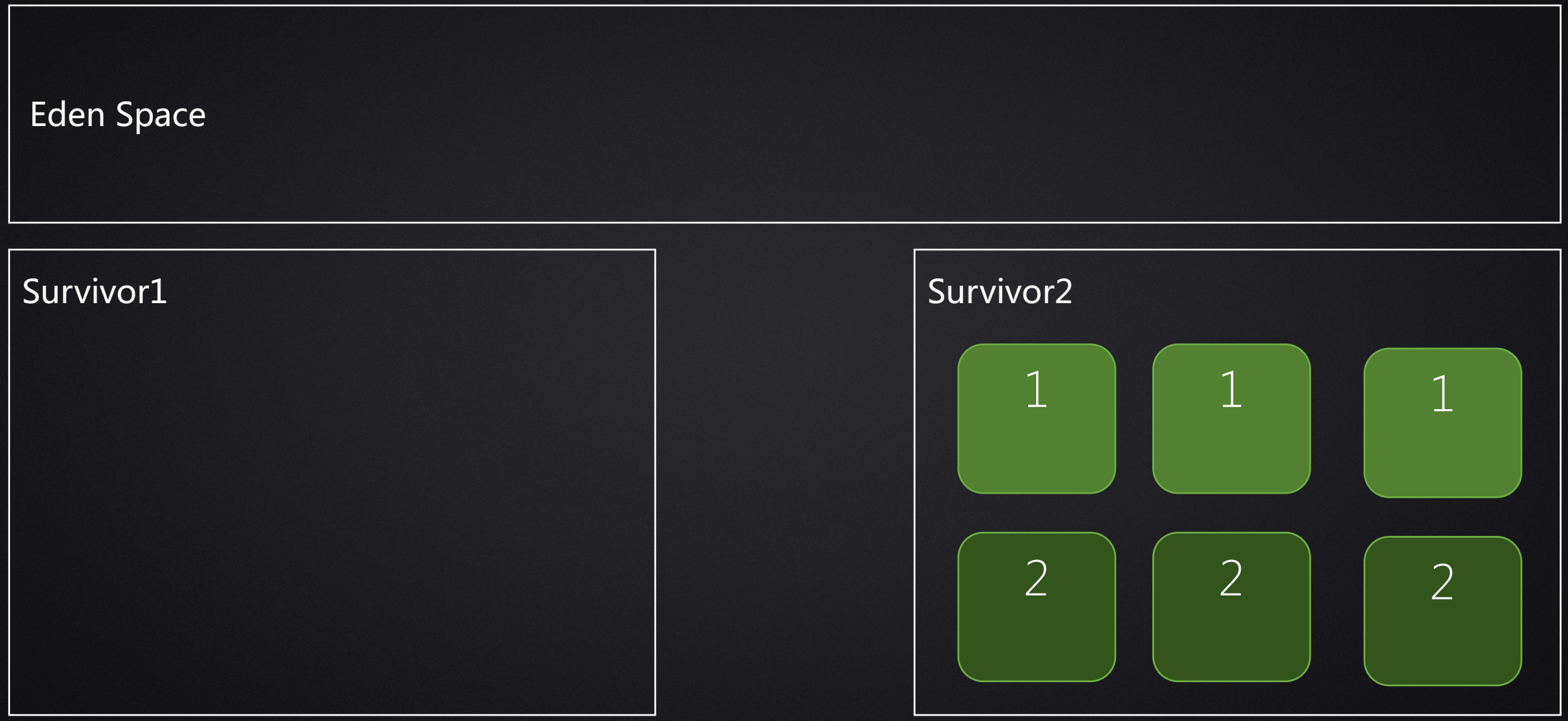
Young Generation



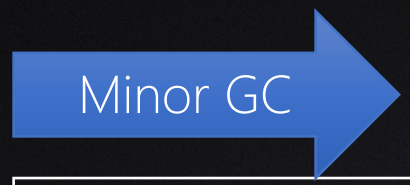
Young Generation



Young Generation



Young Generation



Eden Space

no space

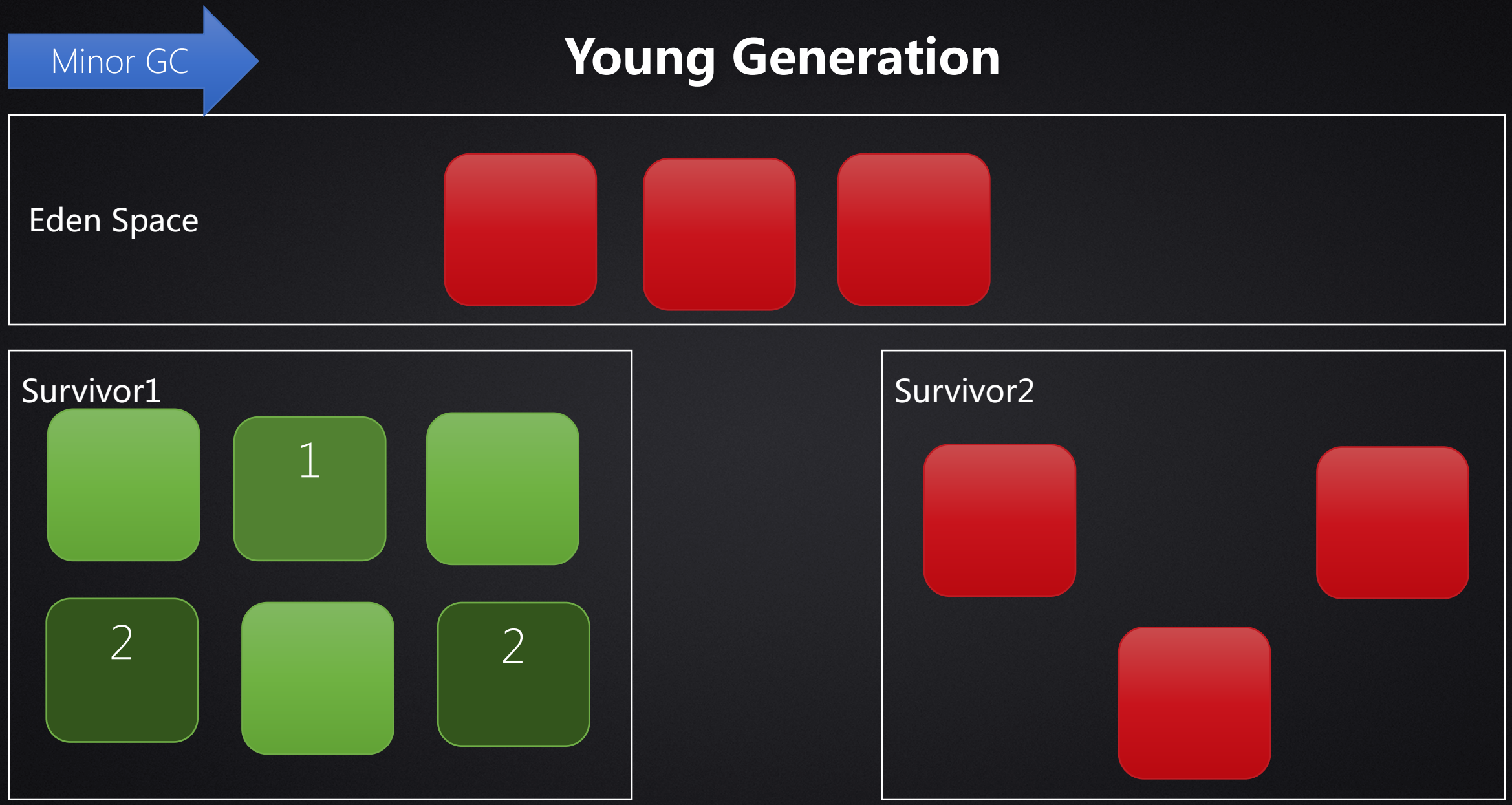
Survivor1

Survivor2

1

2

2


Minor GC
Young Generation
Eden Space
Survivor1
1
2
2
Survivor2

何时进入Old Generation?

Survivor空间不够

-XX:MaxTenuringThreshold存储空间4 bit (二进制1111)(十进制15)

Old Generation何时GC?

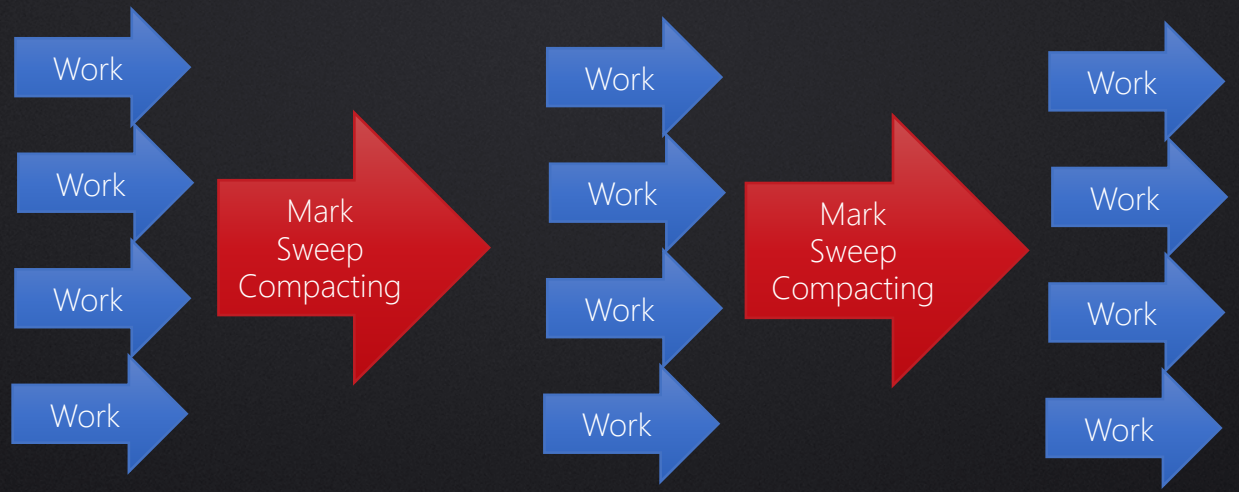
快满时执行Major GC

mark/ sweep/ compacting 整个heap（成本高）

Java 垃圾收集器

Serial Collector

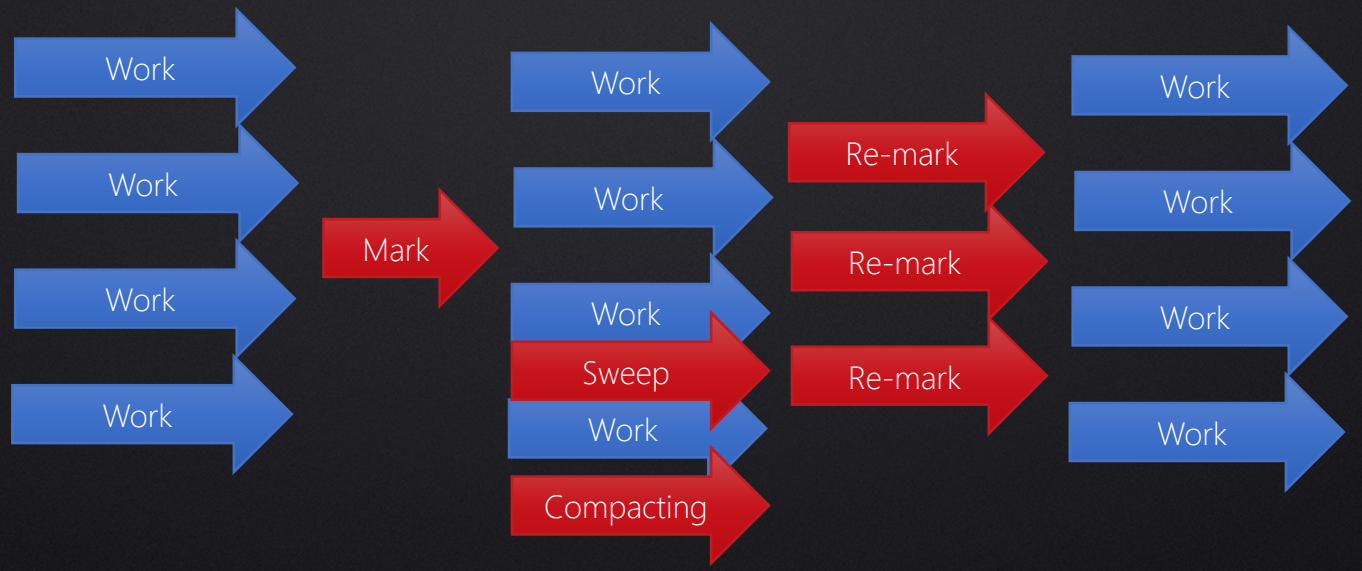
最基础的一种收集器，单线程，适用于非常基础的应用场景



Java 垃圾收集器

Concurrent Collector (CMS)(Concurrent Mark Sweep)

应用程序运行时在后台**同时**运行的一种收集器，不是等待Old Generation即将满了的时候才进行数据收集，而是在后台同时进行。只在mark/ re-mark 的时候"stop-the-world"



Java 垃圾收集器

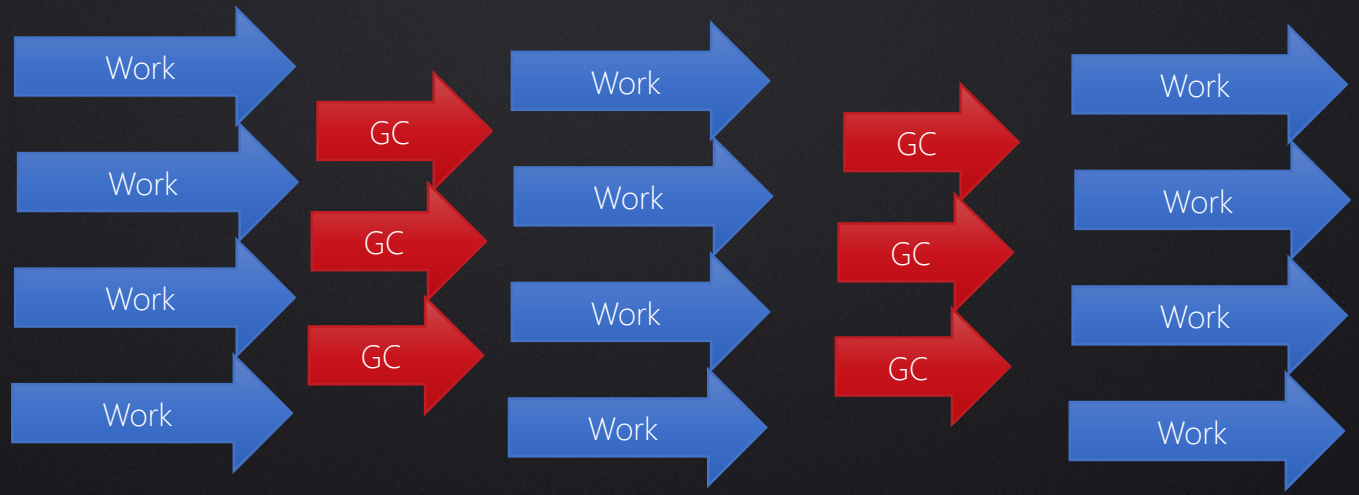
Concurrent Collector 的应用场景

- 足够的内存
- 足够的CPU资源（因为需要提供系统资源给一个常驻后台并不停做清理的线程）
- 较小延迟需求

Java 垃圾收集器

Parallel Collector (并行收集器)

使用CPU多核资源、多线程去处理 mark/ sweep 等操作。仅当heap快满了时才触发，"stop-the-world"



Java 垃圾收集器

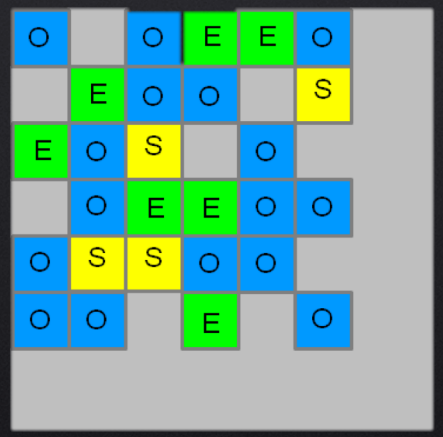
Parallel Collector的应用场景

- 内存有限
- CPU资源有限
- app需要更高的吞吐量并能接受延迟

Java 垃圾收集器

G1 (Garbage-1st)

- 把heap切成小的区块放到内存中
- 每一个区块可能被划分为 Eden space, survivor space或 old space
- 当某个区块出现较多垃圾时，该区块会被GC



Java 垃圾收集器

G1 (Garbage-1st)

- G1 即是Concurrent 的，又是parallel 的
  - Parallel: 因为会在某区块快满时进行GC，且处理多个区块时CPU会用到多个内核
  - Concurrent: 只会停掉某一区块，不会stop-the-world
- 更好的利用了heap资源（之前的垃圾收集器都是一整块，G1内存使用切成小块）
- 高效且低延迟

如何让JVM使用某种垃圾收集器

-XX:+UseSerialGC
-XX:+UseParallelGC: Young generation 设置为parallel, Old generation设置为单线程的GC
-XX:+UseParallelOldGC: Young generation 和Old generation都用 parallel
-XX:+UseParNewGC: 只对Young generation设置Parallel
-XX:+UseConcMarkSweepGC: Old Generation 设置为concurrent. Young generation设置为ParNewGC
- -XX:+UseG1GC
1.6 Parallel, 1.7 update4 G1

对JVM的Heap大小进行调整

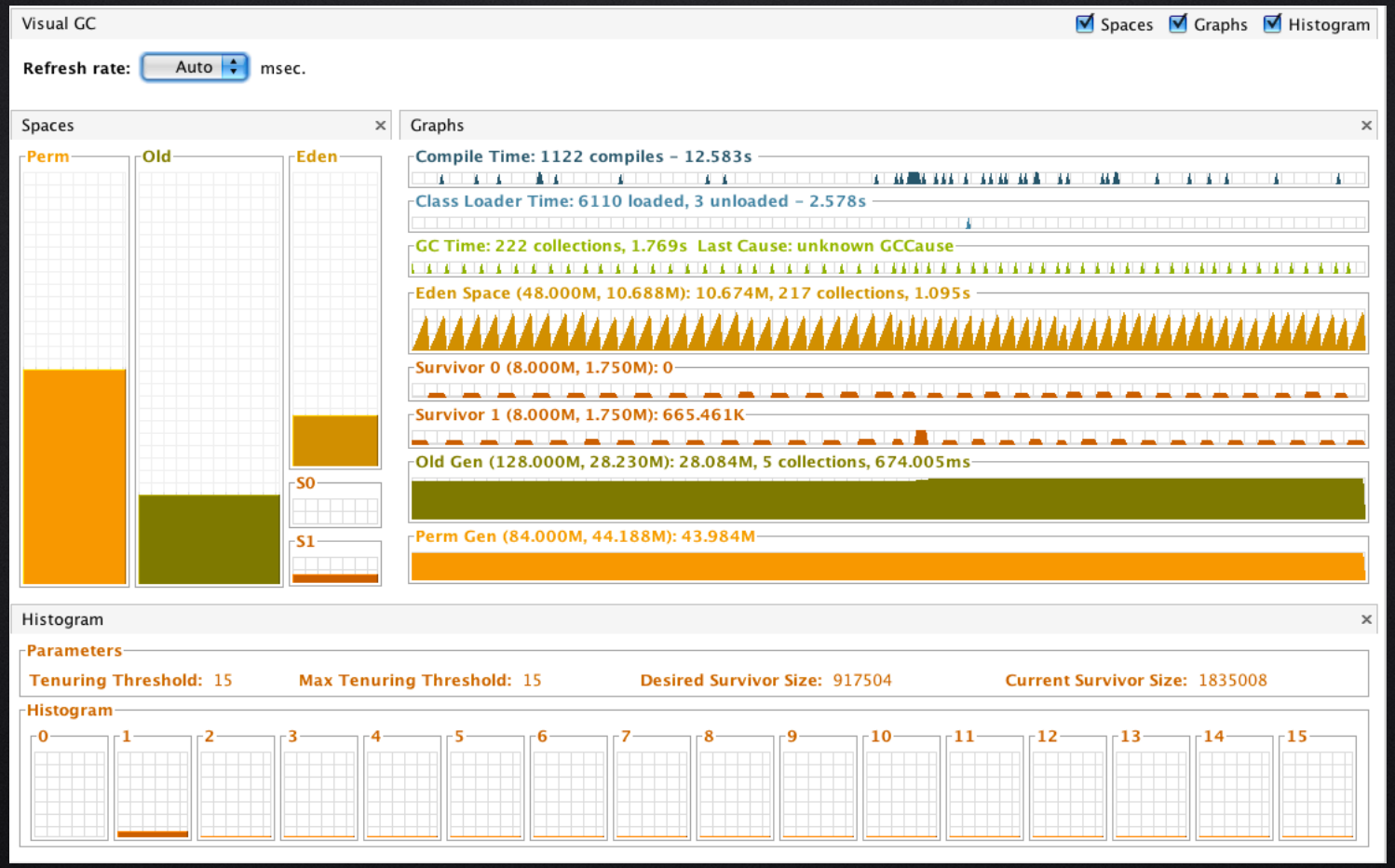
-Xmsvalue (设置heap 的最小值)
-Xmxvalue (设置heap 的最大值) (default:256M)
-XX:NewRatio=ratio (设置为2: Young generation: old generation = 1:2; 1/3 的内存资源分给Young generation; 2/3的内存资源分给Old generation)
-XX:NewSize=size: 设置Eden space 大小
-XX:MaxNewSize=size: 设置Eden space 最大值
-XX:PermSize: 设置Permgen 大小 (method data/ static object....)
-XX:MaxPermSize: 设置Permgen 最大值(def: 64M)

如何打印GC log

-verbose:gc
-XX:+PrintGCDetails
-Xloggc:gc.log

分析GC有没有好用的工具

jvisualvm: gc 可视化插件



还要不要听我讲JVM原理？