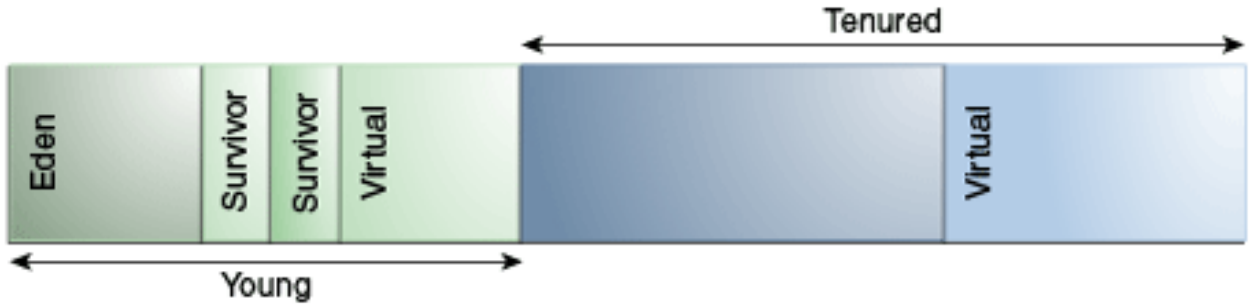
Garbage Collection

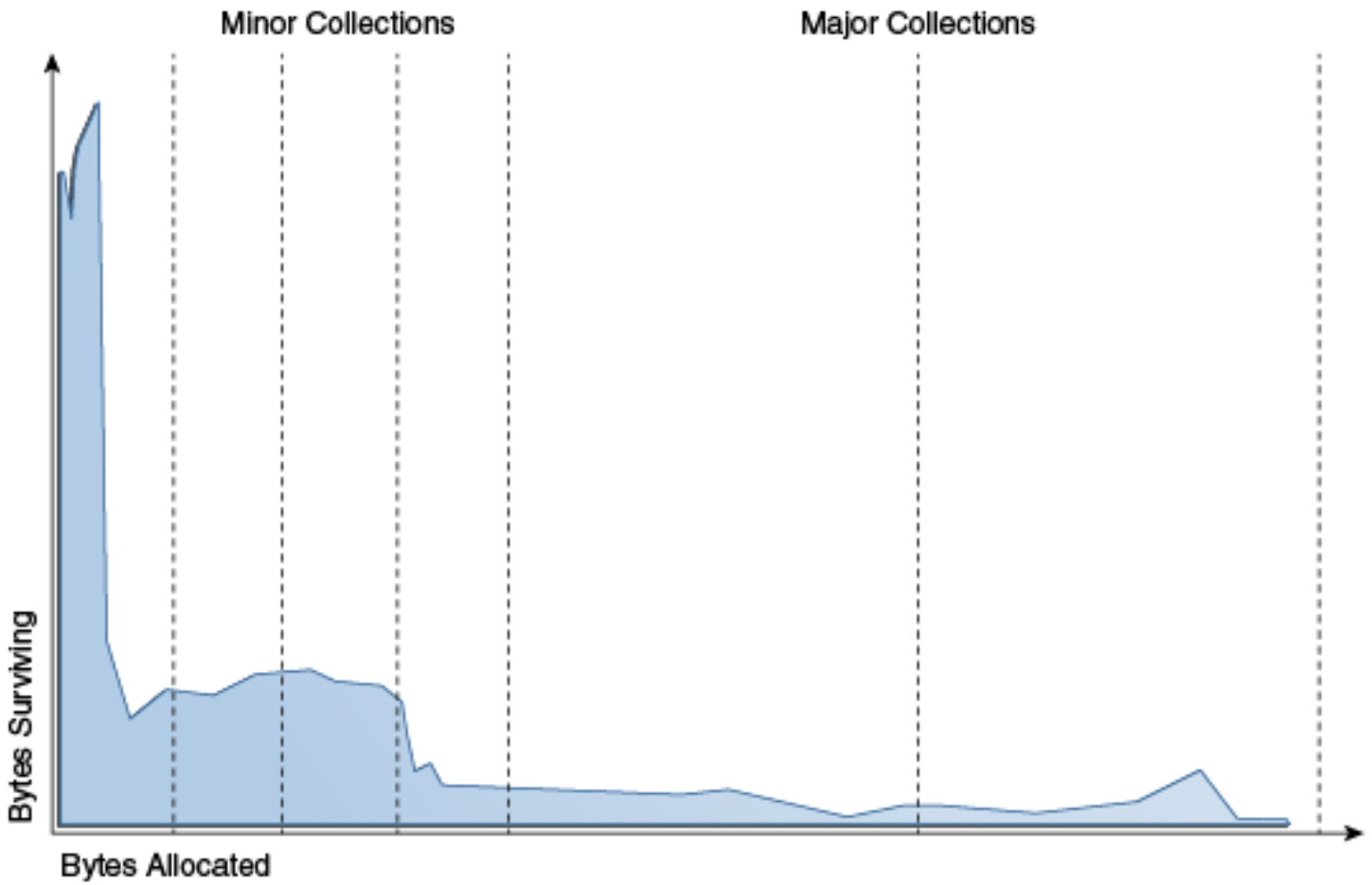
- Young generation GC
  - Copy
  - PS Scavenge
  - ParNew
  - G1 Young Generation
- Old generation GC
  - MarkSweepCompact
  - PS MarkSweep
  - ConcurrentMarkSweep
  - G1 Mixed Generation

Young generation: Most of the newly created objects are located here. Since most objects soon become unreachable, many objects are created in the young generation, then disappear. When objects disappear from this area, we say a "**minor GC**" has occurred.

Old generation: The objects that did not become unreachable and survived from the young generation are copied here. It is generally larger than the young generation. As it is bigger in size, the GC occurs less frequently than in the young generation. When objects disappear from the old generation, we say a "**major GC**" (or a "**full GC**") has occurred.



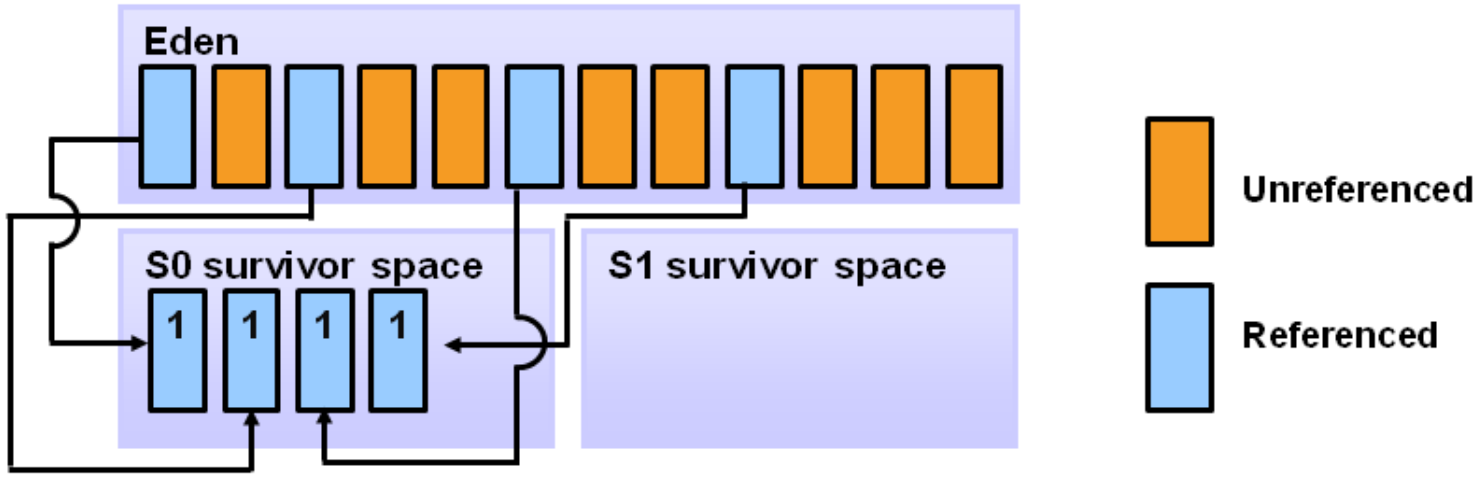
Allocated/Surviving



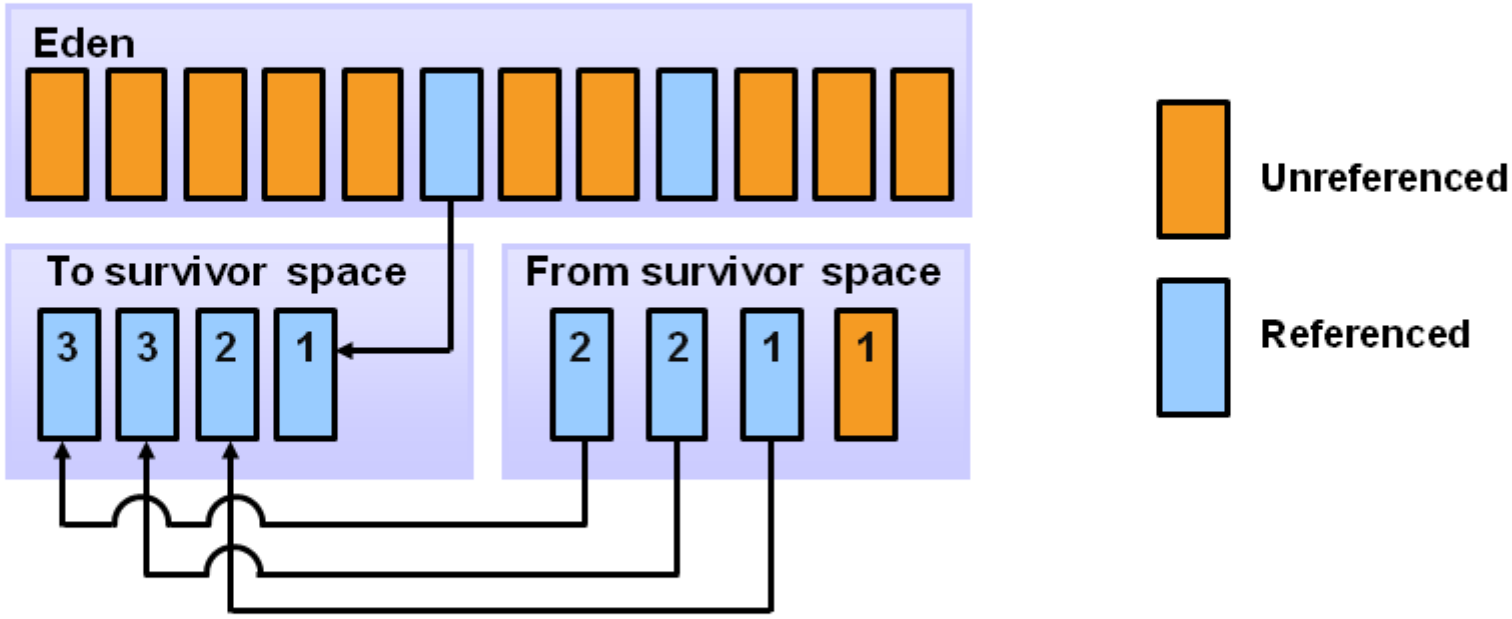
https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/generations.html

Young generation GC algorithm

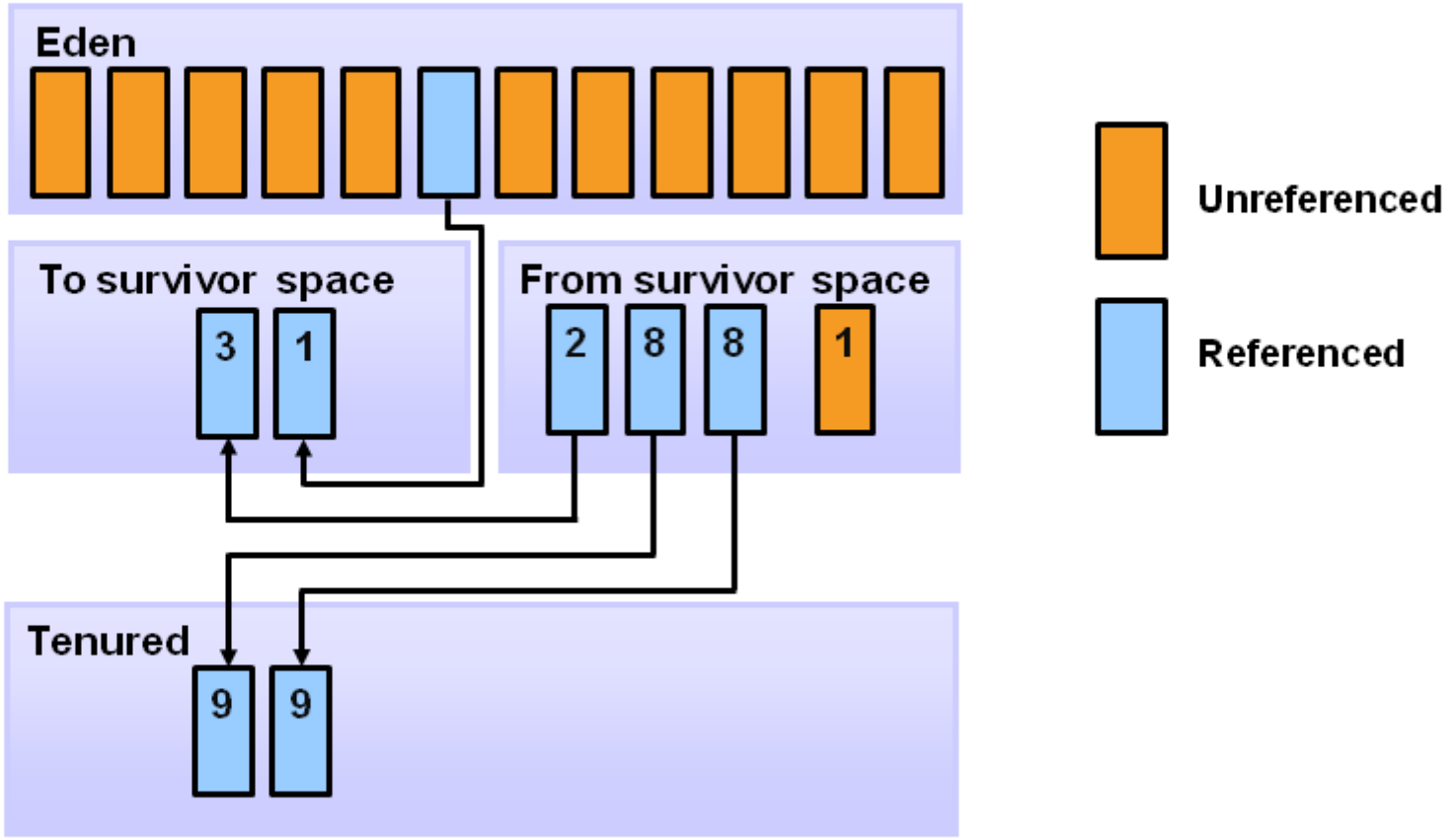
1. Eden → S0



2. Eden → S1, S0 → S1



9. Eden → S0, S1 → S0, S1



Young generation collectors

Copy (enabled with -XX:+UseSerialGC)

the serial copy collector, uses one thread to copy surviving objects from Eden to Survivor spaces and between Survivor spaces until it decides they've been there long enough, at which point it copies them into the old generation.

PS Scavenge (enabled with -XX:+UseParallelGC)

the parallel scavenge collector, like the Copy collector, but uses multiple threads in parallel and has some knowledge of how the old generation is collected (essentially written to work with the serial and PS old gen collectors).

ParNew (enabled with -XX:+UseParNewGC)

the parallel copy collector, like the Copy collector, but uses multiple threads in parallel and has an internal 'callback' that allows an old generation collector to operate on the objects it collects (really written to work with the concurrent collector).

G1 Young Generation (enabled with -XX:+UseG1GC)

the garbage first collector, uses the 'Garbage First' algorithm which splits up the heap into lots of smaller spaces, but these are still separated into Eden and Survivor spaces in the young generation for G1.

Old generation collectors

MarkSweepCompact (enabled with -XX:+UseSerialGC)

the serial mark-sweep collector, the daddy of them all, uses a serial (one thread) full mark-sweep garbage collection algorithm, with optional compaction.

PS MarkSweep (enabled with -XX:+UseParallelOldGC)

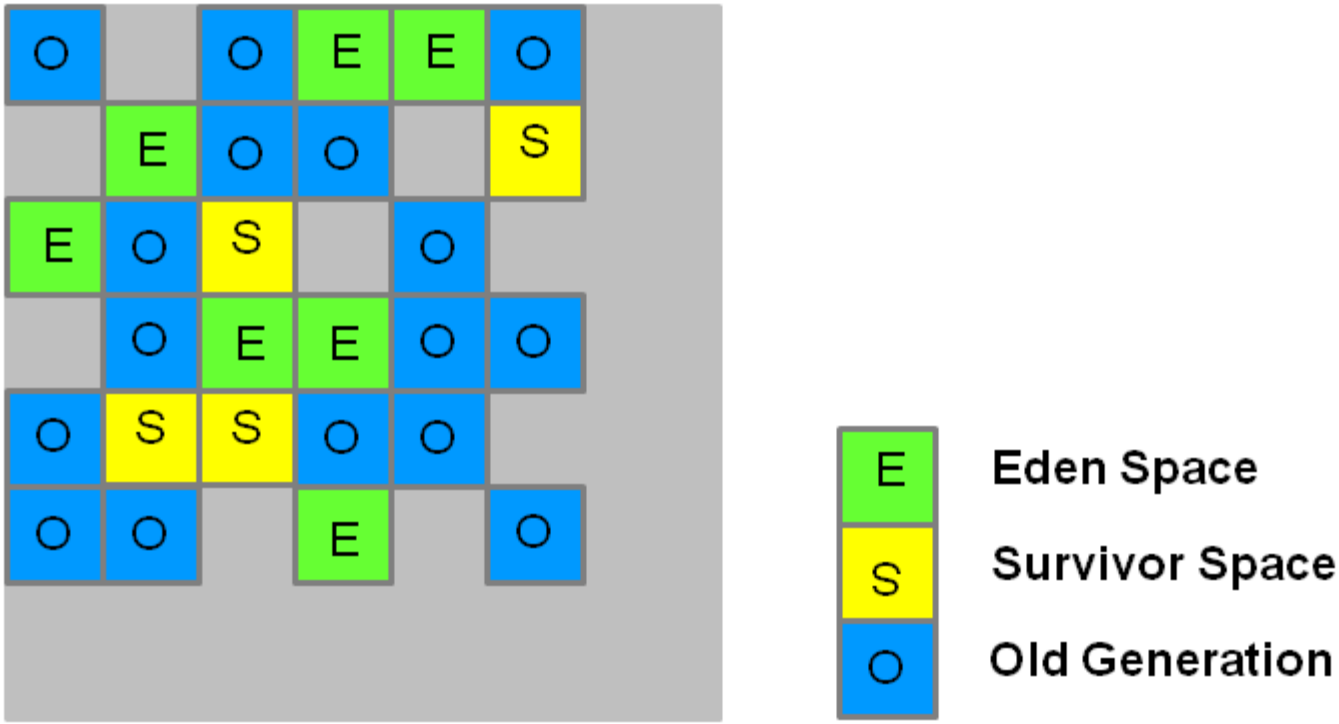
the parallel scavenge mark-sweep collector, parallelised version (i.e. uses multiple threads) of the MarkSweepCompact.

ConcurrentMarkSweep (enabled with -XX:+UseConcMarkSweepGC)

the concurrent collector, a garbage collection algorithm that attempts to do most of the garbage collection work in the background without stopping application threads while it works (there are still phases where it has to stop application threads, but these phases are attempted to be kept to a minimum). Note if the concurrent collector fails to keep up with the garbage, it fails over to the serial MarkSweepCompact collector for (just) the next GC.

G1 Mixed Generation (enabled with -XX:+UseG1GC)

the garbage first collector, uses the 'Garbage First' algorithm which splits up the heap into lots of smaller spaces.



The heap is partitioned into a set of equal-sized heap regions, each a contiguous range of virtual memory. Certain region sets are assigned the same roles (eden, survivor, old) as in the older collectors, but there is not a fixed size for them. This provides greater flexibility in memory usage.

Stop-the-world

All of the garbage collection algorithms except ConcurrentMarkSweep are stop-the-world, i.e. they stop all application threads while they operate - the stop is known as 'pause' time. The ConcurrentMarkSweep tries to do most of it's work in the background and minimize the pause time, but it also has a stop-the-world phase and can fail into the MarkSweepCompact which is fully stop-the-world. (The G1 collector has a concurrent phase but is currently mostly stop-the-world).

Selecting a Collector

If the application has a small data set (up to approximately 100 MB), then

select the serial collector with the option -XX:+UseSerialGC.

If the application will be run on a single processor and there are no pause time requirements, then let the VM select the collector, or select the serial collector with the option -XX:+UseSerialGC.

If (a) peak application performance is the first priority and (b) there are no pause time requirements or pauses of 1 second or longer are acceptable, then let the VM select the collector, or select the parallel collector with -XX: +UseParallelGC.

If response time is more important than overall throughput and garbage collection pauses must be kept shorter than approximately 1 second, then select the concurrent collector with -XX:+UseConcMarkSweepGC or -XX:+UseG1GC.