马士兵教育

G1

GC tuning

马士兵

http://mashibing.com

Revision

- 2019年8月2日 – 马士兵

预备知识

- 年轻代
  - Eden
  - Survivor
- 老年代
  - Tenured

problem of the old GCs

- Serial GC
- Parallel GC
- PN + CMS

操作必须扫描整个老年代，不适合大内存
YO都是独立的内存块，大小必须提前确定

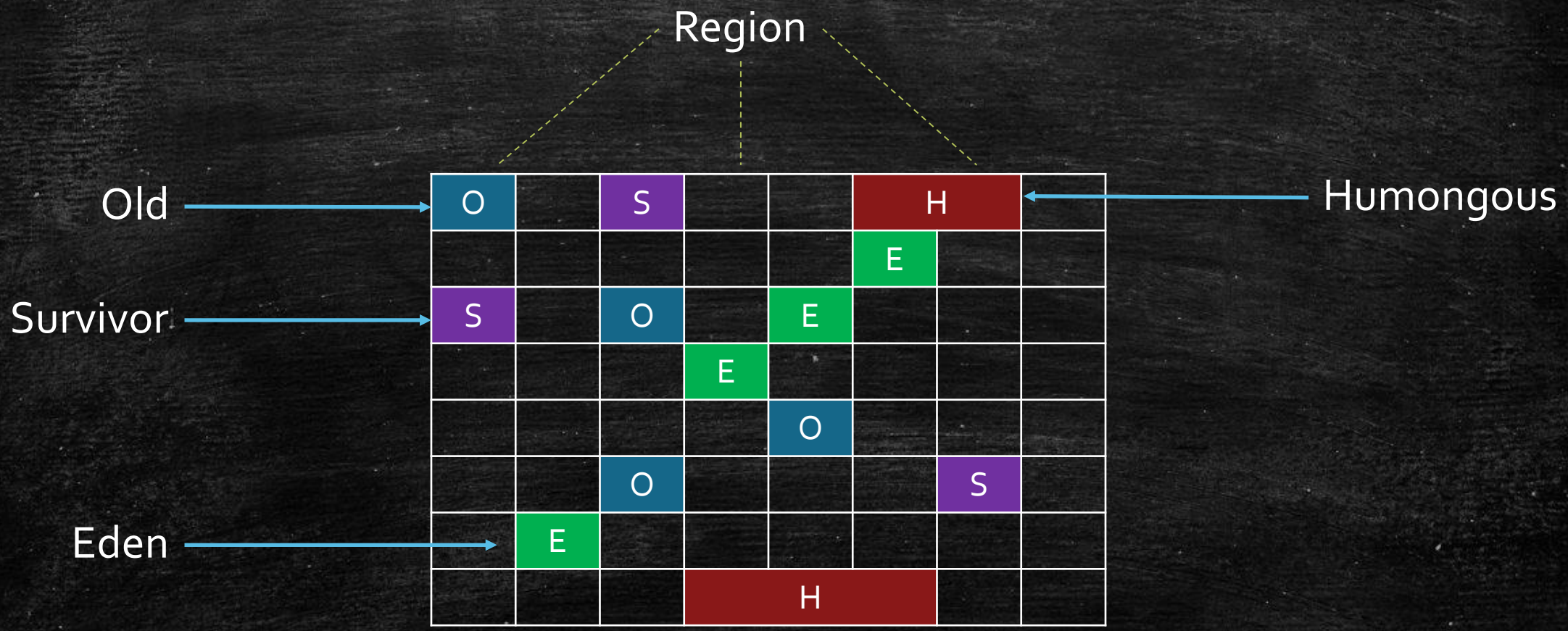
G1

- https://www.oracle.com/technical-resources/articles/java/g1gc.html
- The Garbage First Garbage Collector (G1 GC) is the low-pause, server-style generational garbage collector for Java HotSpot VM. The G1 GC uses concurrent and parallel phases to achieve its target pause time and to maintain good throughput. When G1 GC determines that a garbage collection is necessary, it collects the regions with the least live data first (garbage first).
- G1是一种服务端应用使用的垃圾收集器，目标是用在**多核、大内存**的机器上，它在大多数情况下可以实现指定的GC暂停时间，同时还能保持较高的吞吐量

特点

- 并发收集
- 压缩空闲空间不会延长GC的暂停时间；
- 更易预测的GC暂停时间；
- 适用不需要实现很高的吞吐量的场景

G1



http://mashibing.com

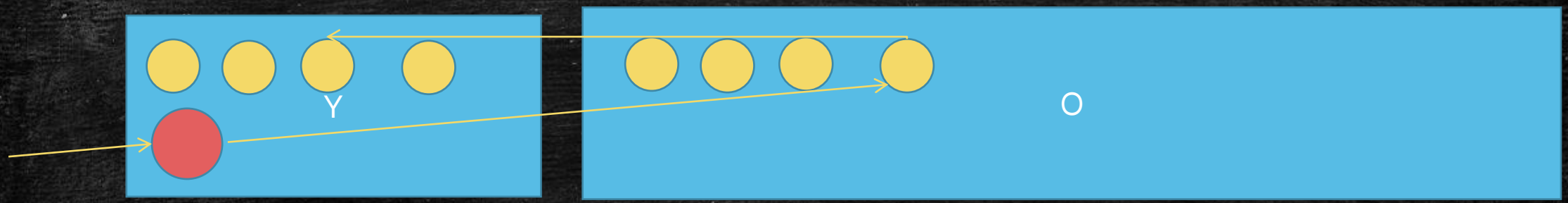
每个分区都可能是年轻代也可能是老年代，但是在同一时刻只能属于某个代。

年轻代、幸存区、老年代这些概念还存在，成为逻辑上的概念，这样方便复用之前分代框架的逻辑。在物理上不需要连续，则带来了额外的好处——有的分区内垃圾对象特别多，有的分区内垃圾对象很少，G1会优先回收垃圾对象特别多的分区，这样可以花费较少的时间来回收这些分区的垃圾，这也就是G1名字的由来，即首先收集垃圾最多的分区。

新生代其实并不是适用于这种算法的，依然是在新生代满了的时候，对整个新生代进行回收——整个新生代中的对象，要么被回收、要么晋升，至于新生代也采取分区机制的原因，则是因为这样跟老年代的策略统一，方便调整代的大小。

G1还是一种带压缩的收集器，在回收老年代的分区时，是将存活的对象从一个分区拷贝到另一个可用分区，这个拷贝的过程就实现了局部的压缩。每个分区的大小从1M到32M不等，但是都是2的冥次方。

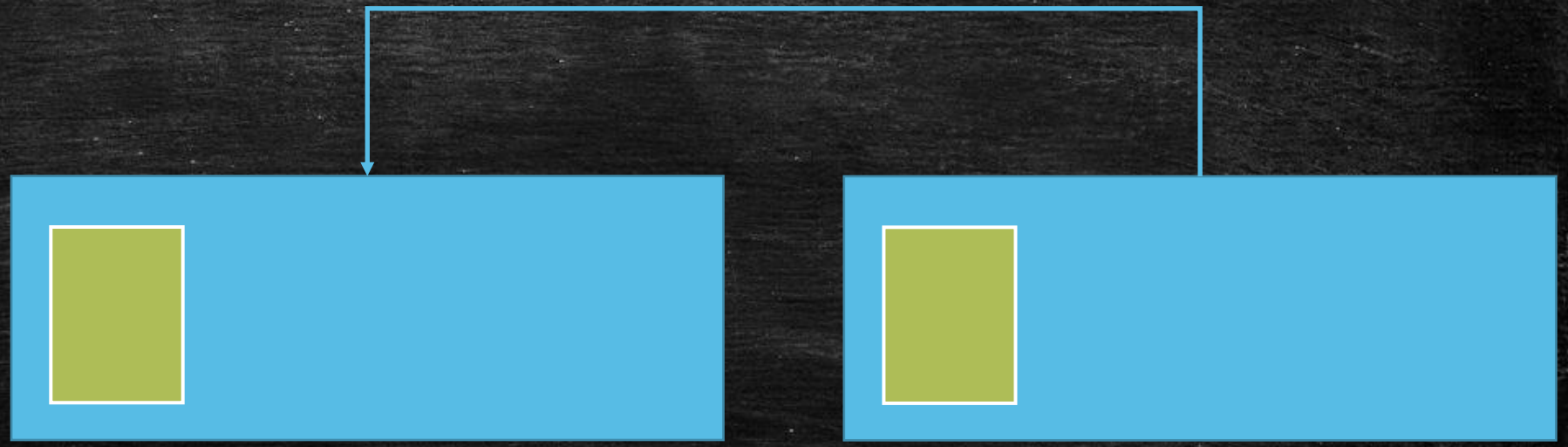
基本概念：card table



基本概念

- CSet = Collection Set
- 一组可被回收的分区的集合。
  在CSet中存活的数据会在GC过程中被移动到另一个可用分区，
  CSet中的分区可以来自Eden空间、survivor空间、或者老年代。
  CSet会占用不到整个堆空间的1%大小。

RSet = RememberedSet

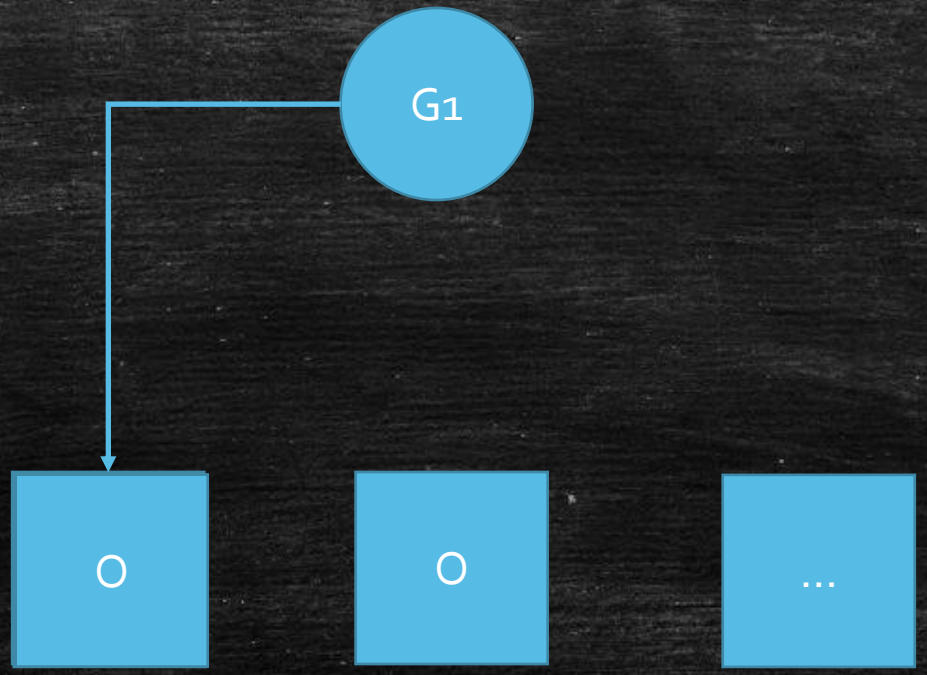


记录了其他Region中的对象到本Region的引用
RSet的价值在于
使得垃圾收集器不需要扫描整个堆找到谁引用了当前分区中的对象，
只需要扫描RSet即可。

阿里的多租户JVM

- 每租户单空间
- session based GC

G1的内存区域不是固定的E或者O



Why G1

- 追求吞吐量
  - 100 cpu
  - 99 app 1 GC
  - 吞吐量 = 99%
- 追求响应时间
  - XX:MaxGCPauseMillis 200
  - 对STW进行控制
- 灵活
  - 分Region回收
  - 优先回收花费时间少、垃圾比例高的Region

每个Region有多大

- headpRegion.cpp
- 取值
  - 1 2 4 8 16 32
- 手工指定
  - XX:G1HeapRegionSize

```
// Minimum region size; we won't go lower than that.
// We might want to decrease this in the future, to deal with small
// heaps a bit more efficiently.
#define MIN_REGION_SIZE  (      1024 * 1024 )

// Maximum region size; we don't go higher than that. There's a good
// reason for having an upper bound. We don't want regions to get too
// large, otherwise cleanup's effectiveness would decrease as there
// will be fewer opportunities to find totally empty regions after
// marking.
#define MAX_REGION_SIZE  ( 32 * 1024 * 1024 )

// The automatic region size calculation will try to have around this
// many regions in the heap (based on the min heap size).
#define TARGET_REGION_NUMBER          2048
```

RSet 与 赋值的效率

- 由于RSet 的存在，那么每次给对象赋引用的时候，就得做一些额外的操作，
- 指的是在RSet中做一些额外的记录（在GC中被称为写屏障）

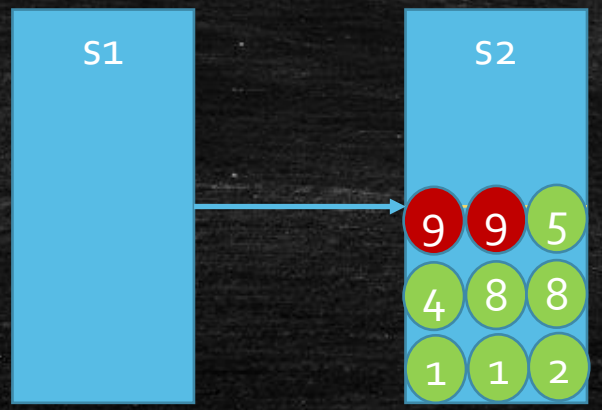
- 这个写屏障 不等于 内存屏障
- No Silver Bullet

新老年代比例

- 5% - 60%
  - 一般不用手工指定
  - 也不要手工指定，因为这是G1预测停顿时间的基准

对象何时进入老年代

- 超过 XX:MaxTenuringThreshold 指定次数（YGC）
  - Parallel Scavenge 15
  - CMS 6
  - G1 15
- 动态年龄
  - s1 - > s2超过50%
  - 把年龄最大的放入O



humongous object

- 超过单个region的50%

GC何时触发

- YGC
  - Eden空间不足
  - 多线程并行执行
- FGC
  - Old空间不足
  - System.gc()

G1中的MixedGC

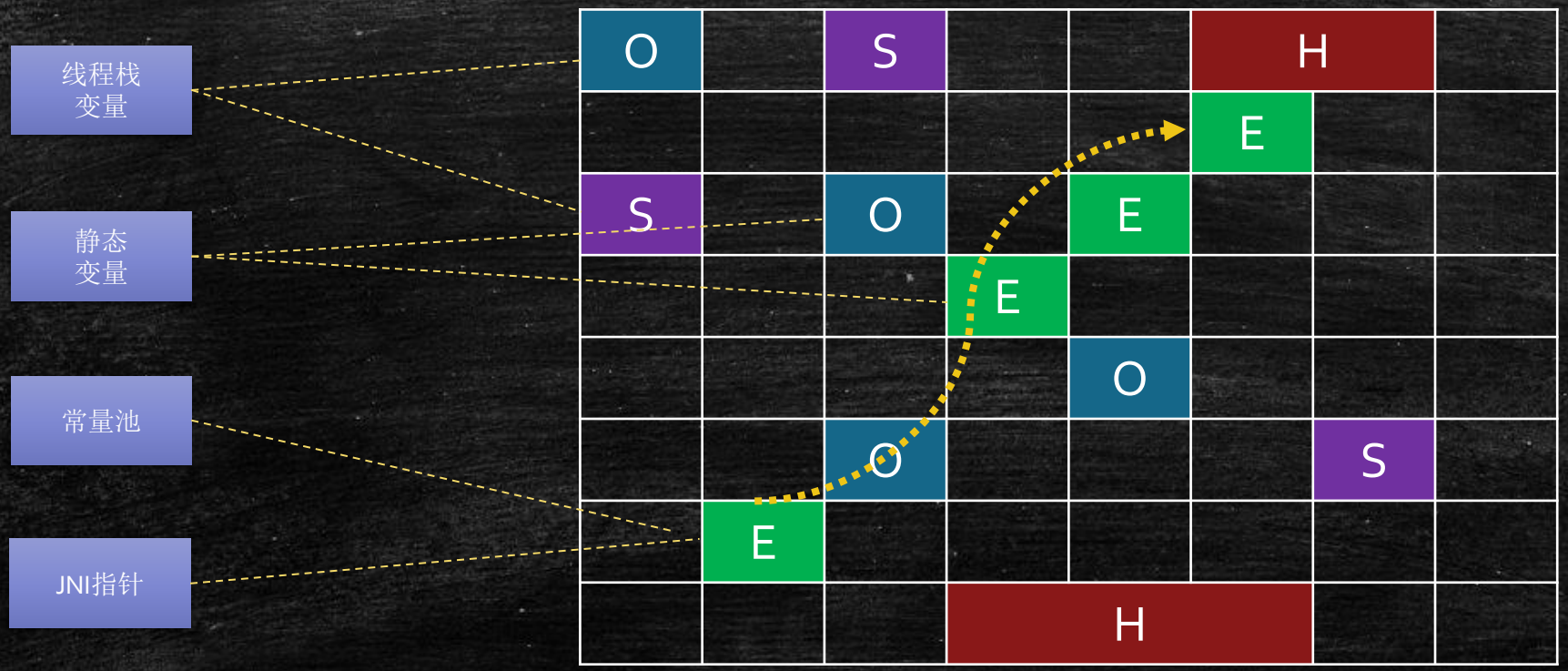
- = CMS
- XX:InitiatingHeapOccupacyPercent
  - 默认值45%
  - 当O超过这个值时，启动MixedGC

MixedGC的过程

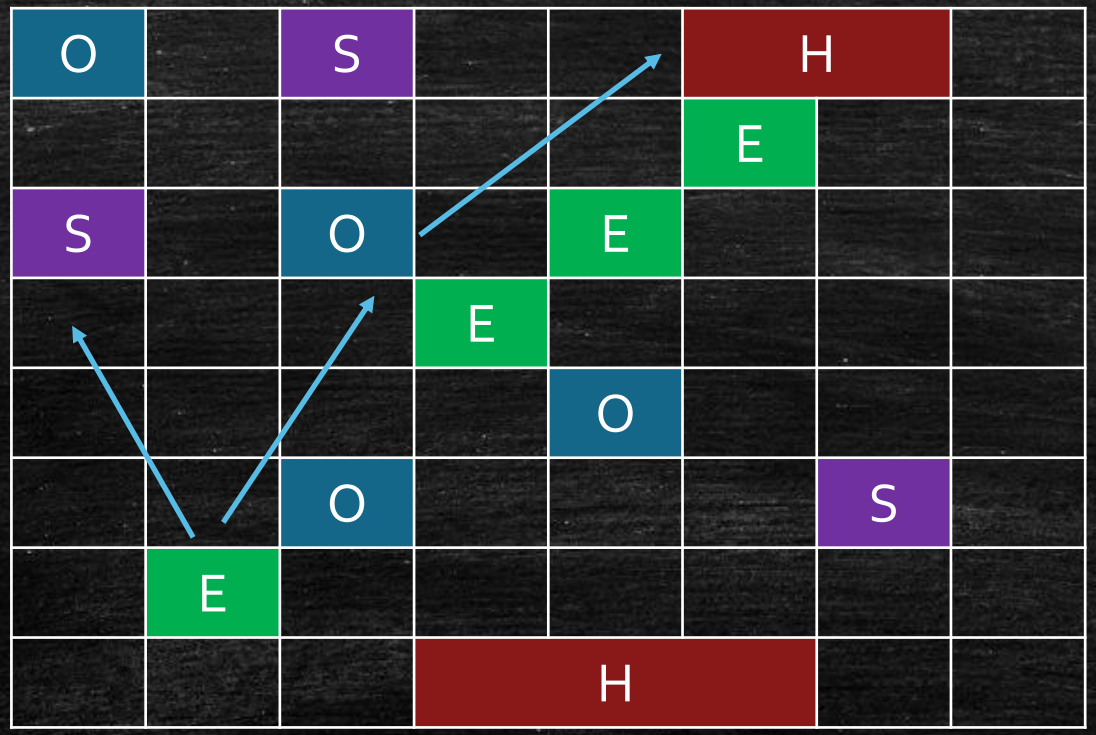
- 初始标记 STW
- 并发标记
- 最终标记 STW （重新标记）
- 筛选回收 STW （并行）

初始标记

GC roots



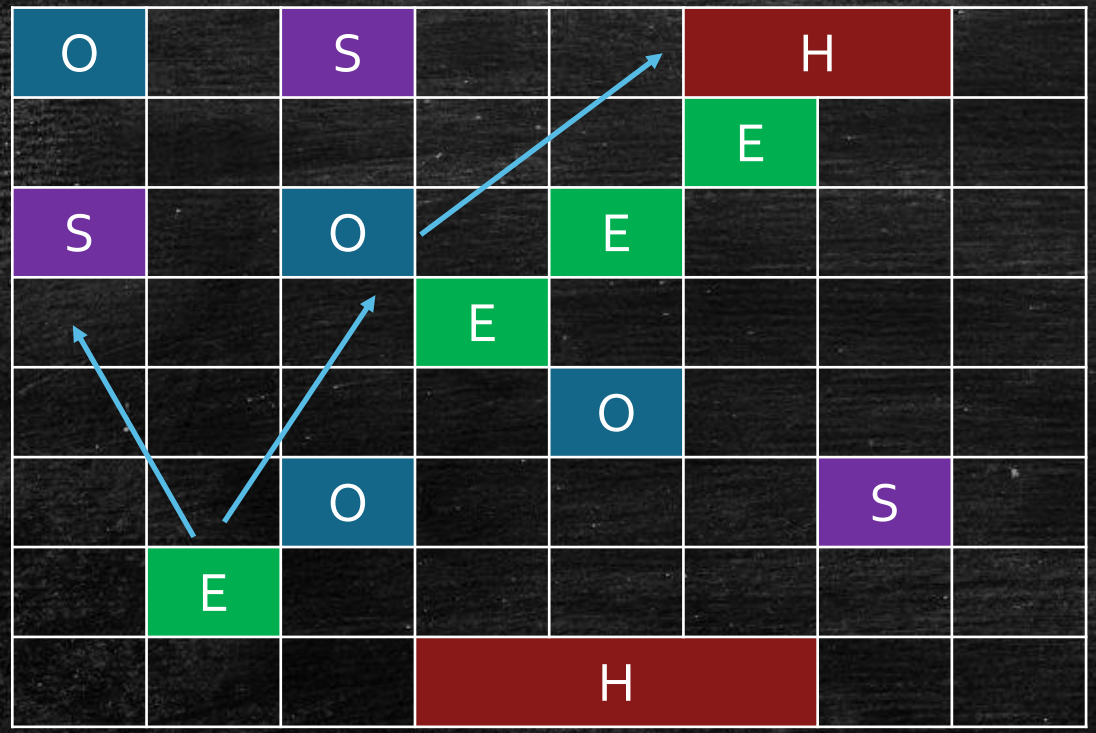
并发标记



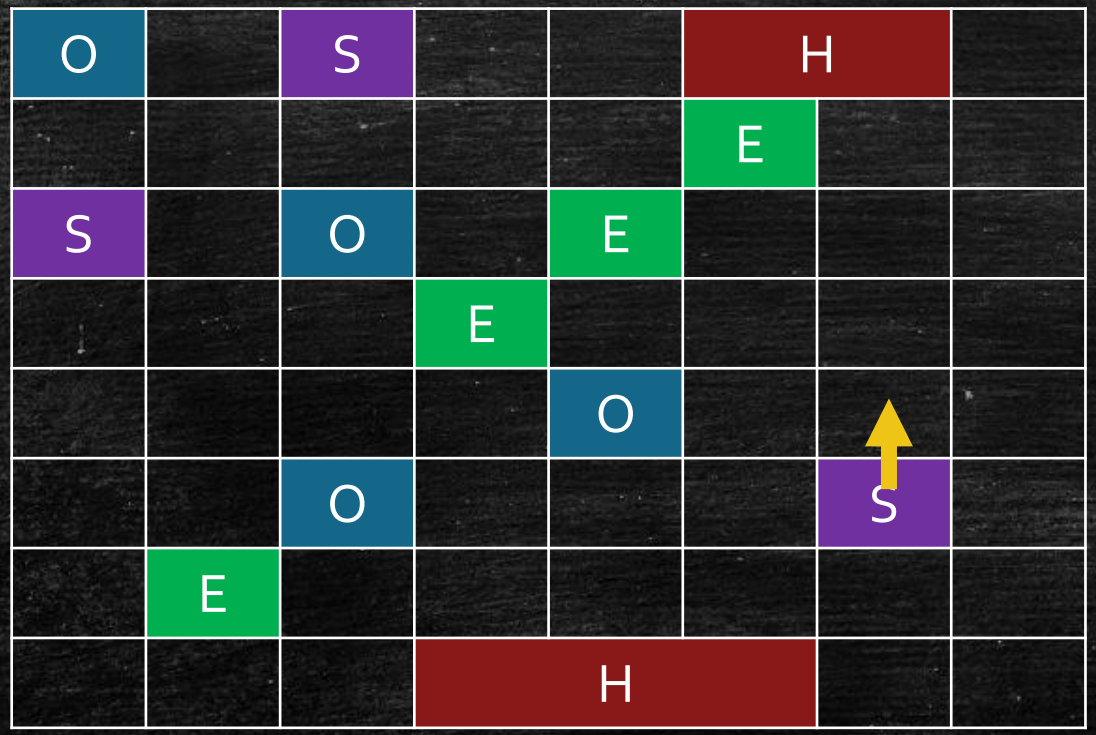
SATB算法：
Snapshot At The Beginning
GC开始时，通过root tracing得到一个Sna
维持并发GC的正确性

如何做到并发GC的正确性：
三色标记算法：
白：对象没有标记，标记阶段结束后，会
灰：对象标记了，但是他的Field还没有标
黑：对象标记了，且他的Field也标记完成

最终标记



并行筛选回收



G1的Full GC

java 10以前是串行FullGC，之后是并行FullGC

参数设定

总结

- what is G1
- why G1
- how G1 works

展望未来

设计目标：

1. 暂停时间不超过10ms
2. 暂停时间不随堆的大小变化而变化
3. 处理内存从数百M到几个T

技术特点：

1. concurrent
2. region-based
3. compacting
4. NUMA-aware
5. colored pointers
6. load barriers

ZGC