Turing College 图灵课堂 | 致敬大师 致敬未来的你

为Java开疆拓土的ZGC深度剖析

King老师

- 十余年Java行业经验
- 先后在58、招行等公司工作
- 主导互联网金融项目的核心架构开发

课程内容

1、ZGC中JVM内存布局及设计
2、ZGC如何做到10ms的暂停
3、ZGC基于指针着色的并发标记算法
4、ZGC基于指针着色的并发转移算法
5、ZGC整体流程分析及性能对比实战
6、大厂项目ZGC的技术运用与挑战
7、并发转移算法关键技术之读屏障
8、ZGC的触发时机与性能优化

腾讯课堂-图灵课堂

01月26日 晚上20:00

课程安排

JVM最新技术	
章节	章节名称
1	为Java开疆拓土的ZGC深度剖析
2	让Java性能提升的JIT深度剖析
3	GraalVM:云原生时代的Java虚拟机

上课说明：

课程前置知识：JVM性能调优专题

1、课程中简单代码不会手写，一般只会手写核心代码（提高课程效率）

2、一个知识点如果大部分同学明白，不会重复讲解，未明白的同学请看视频、笔记、请教同学或加老师QQ

3、以上为本次的章节安排，不是课时安排，如果一章内容在一次课内未讲完，则会进行顺延

讲师介绍

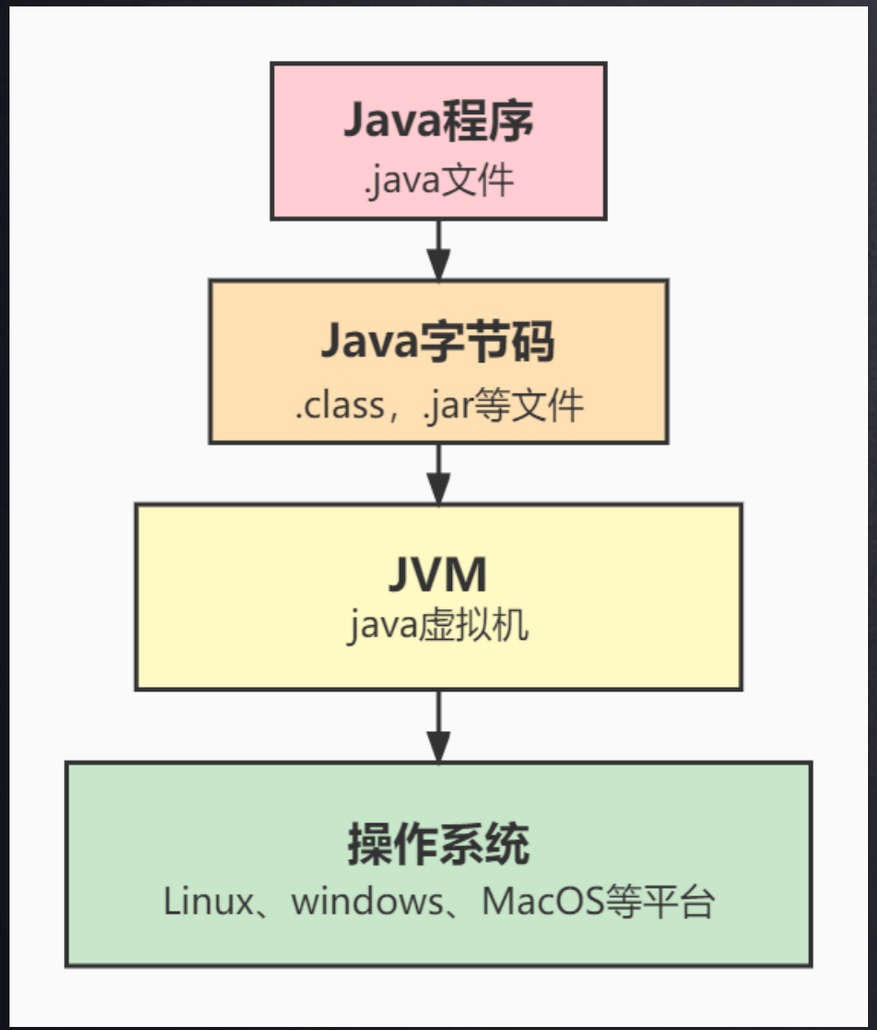
King老师

十三年Java行业经验。

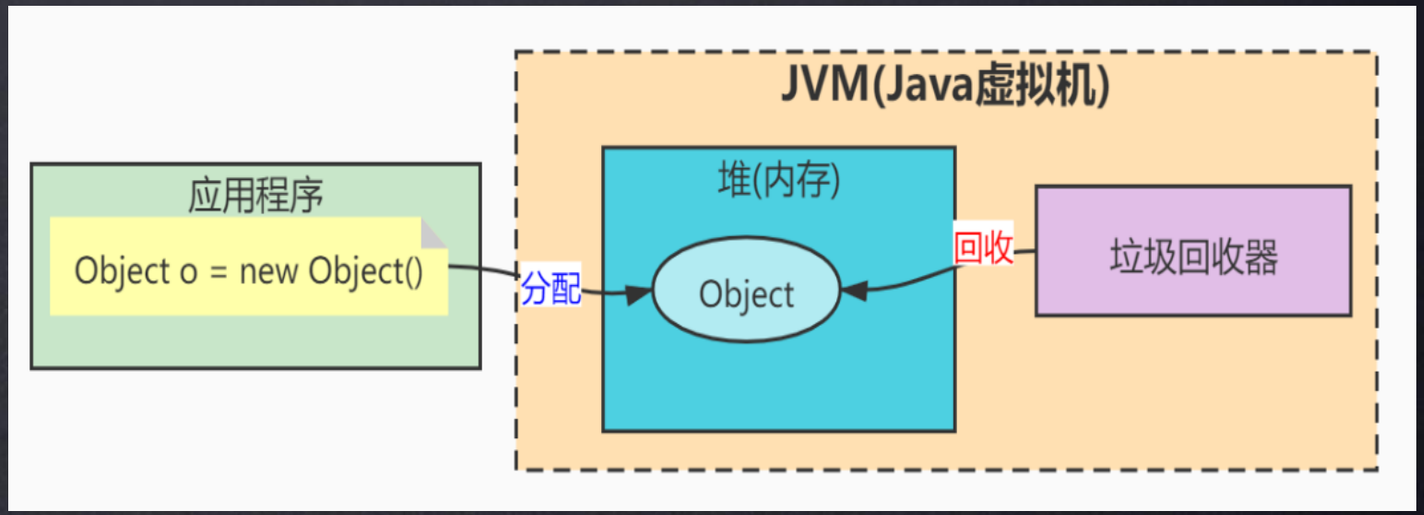
- 资深架构师
- 资深讲师

此处本应有100行介绍，(地方小装不下)

■ JVM：Java虚拟机

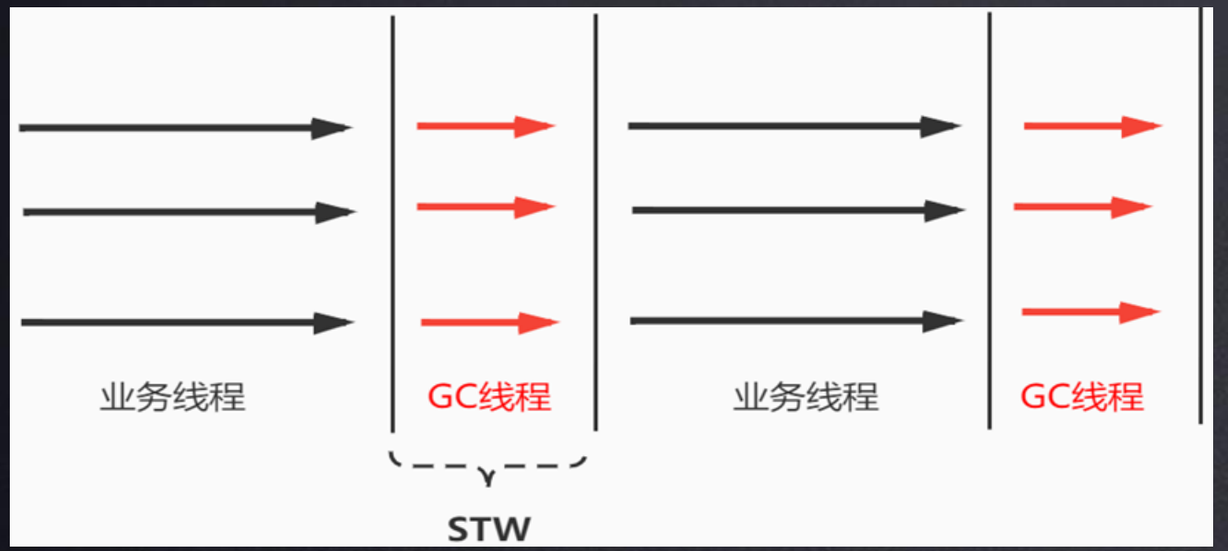


■ JVM核心功能：自动化的垃圾回收机制



垃圾回收中的STW

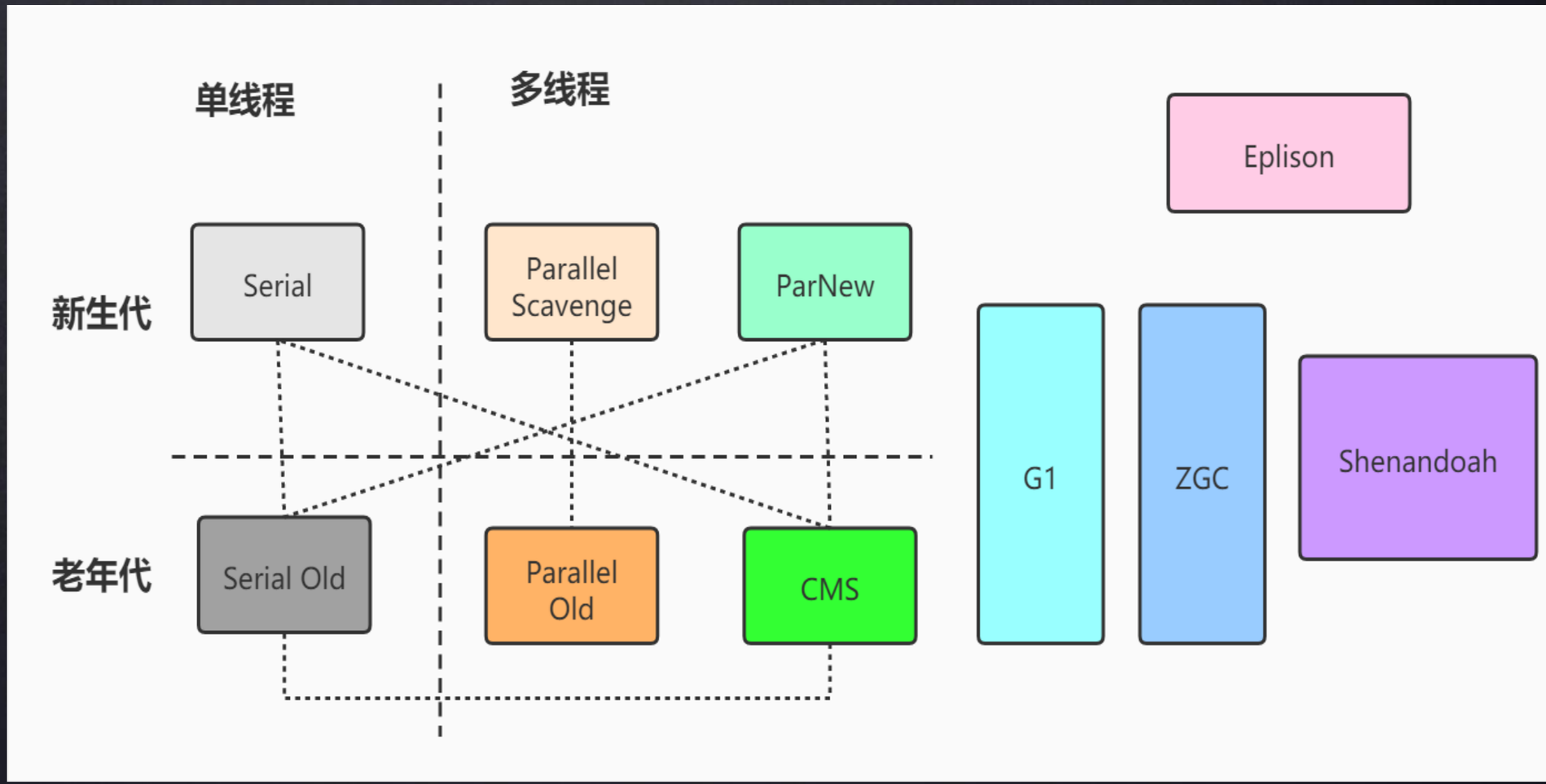
- STW：Stop The World



- Java生态圈已经足够强大，但还不够！
  - 手机系统(Android) 显示卡顿（google主导）
  - 证券交易系统实时性要求（抢C++市场）
  - 大数据平台(HBase集群性能)（58、腾讯、阿里等公司）

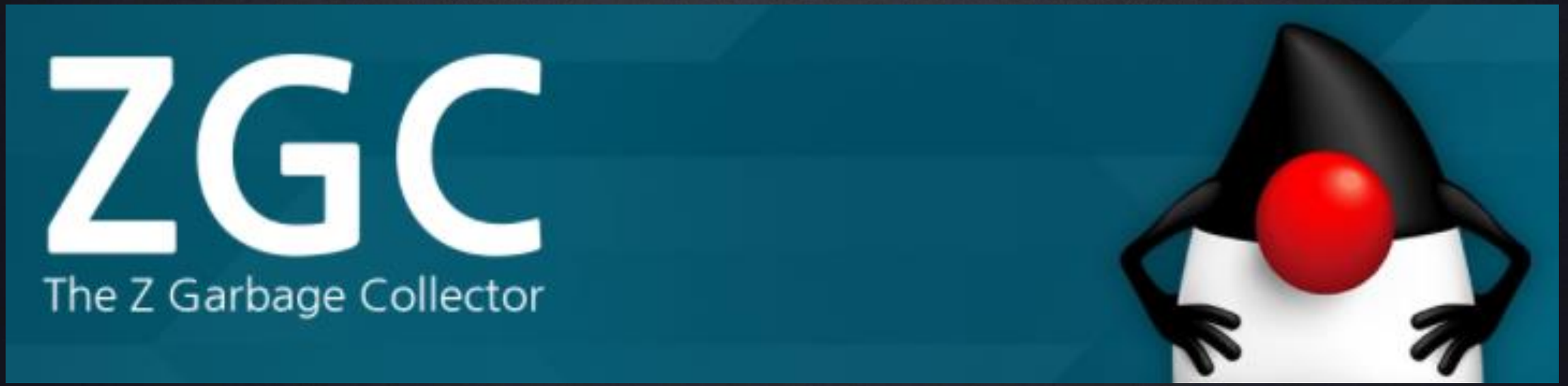
垃圾回收器的发展

- 单线程
  - Serial
  - SerialOld
- 多线程
  - ParallelScavenge
  - ParallelOld
- 多线程+并发
  - CMS
  - G1
  - Shenadndoah
  - ZGC(STW控制在1ms)



ZGC介绍

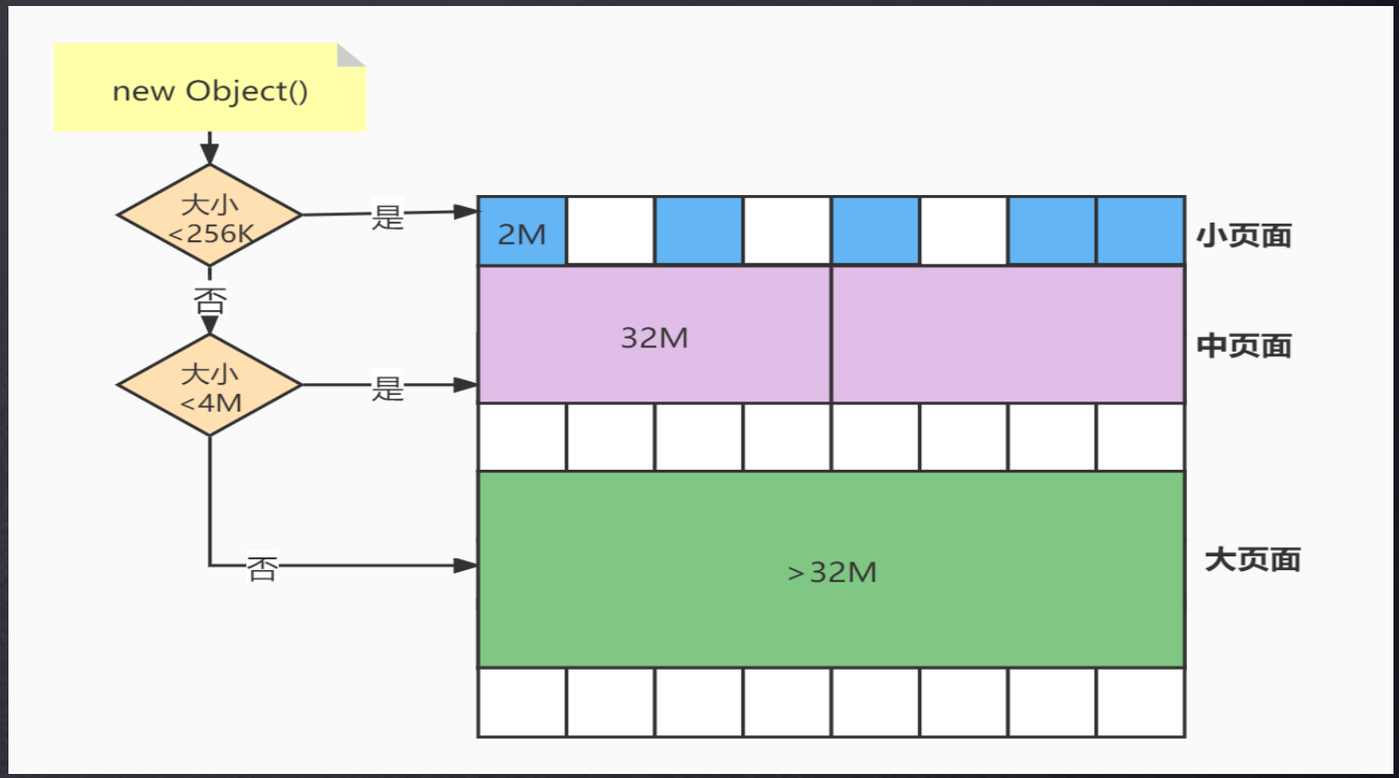
- JDK11中推出的一款低延迟垃圾回收器
- 支持16TB级别的堆
- 停顿时间(STW)不超过1ms，且不会随着堆的大小增加而增加



- ZGC出现的目标：卷其他语言！

ZGC中的内存布局

- 堆空间分页模型（无分代）
  - 小页面
  - 中页面
  - 大页面



- 为什么这么设计？

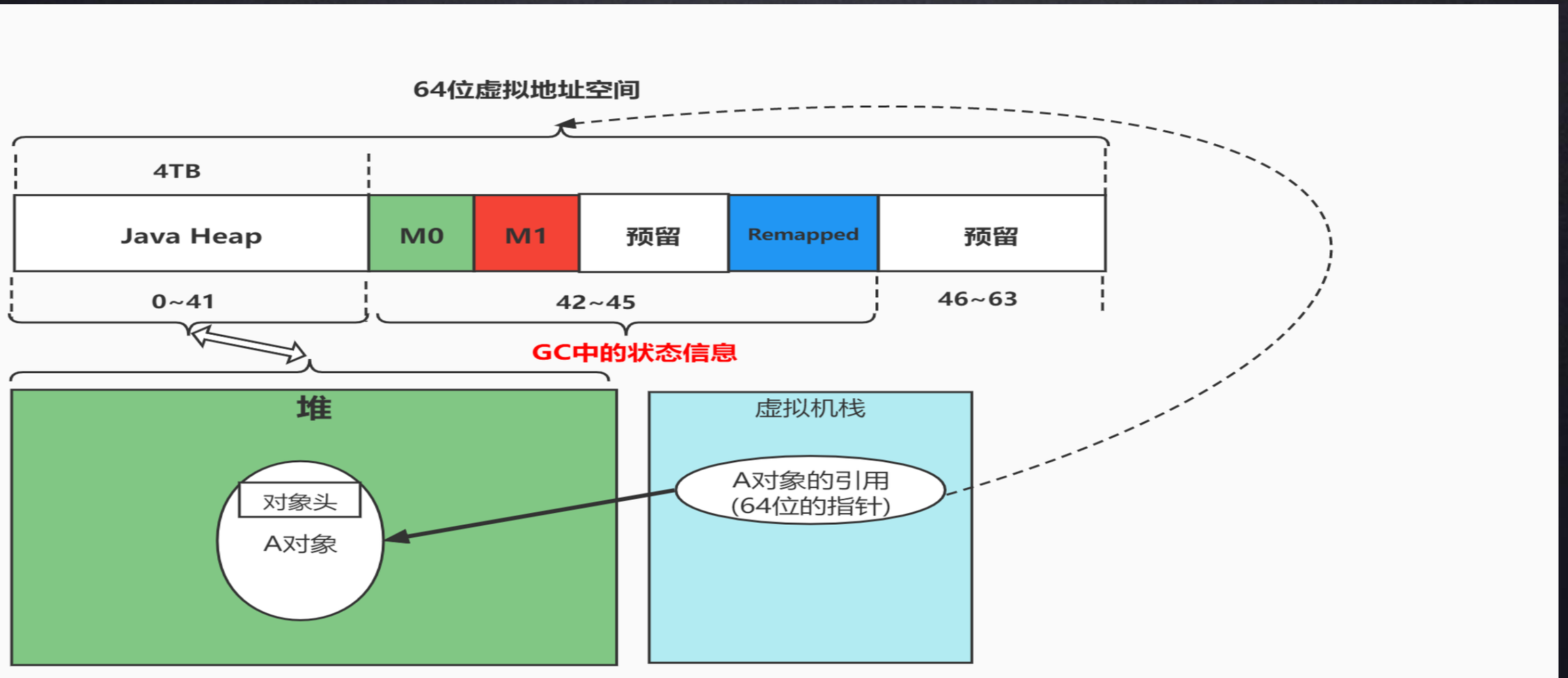
Linux Kernel 2.6引入的标准大页（huge page）

- ZGC支持NUMA(了解即可)

Non-Uniform Memory Access(非统一内存访问)

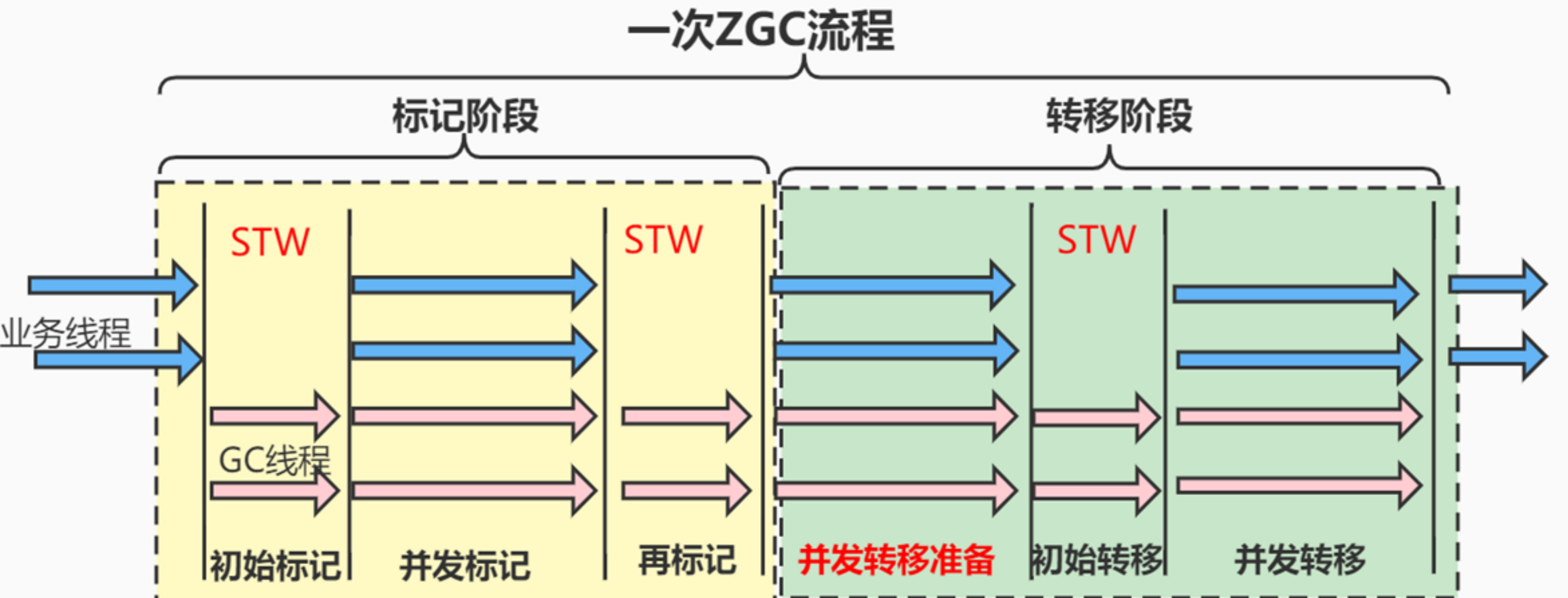
ZGC的核心概念

- 指针着色技术（Color Pointers）
  - ZGC只支持64位系统(使用64位指针)
  - ZGC中低42位表示使用中的堆空间
  - ZGC借几位高位来做GC相关的事情(快速实现垃圾回收中的并发标记、转移和重定位等)



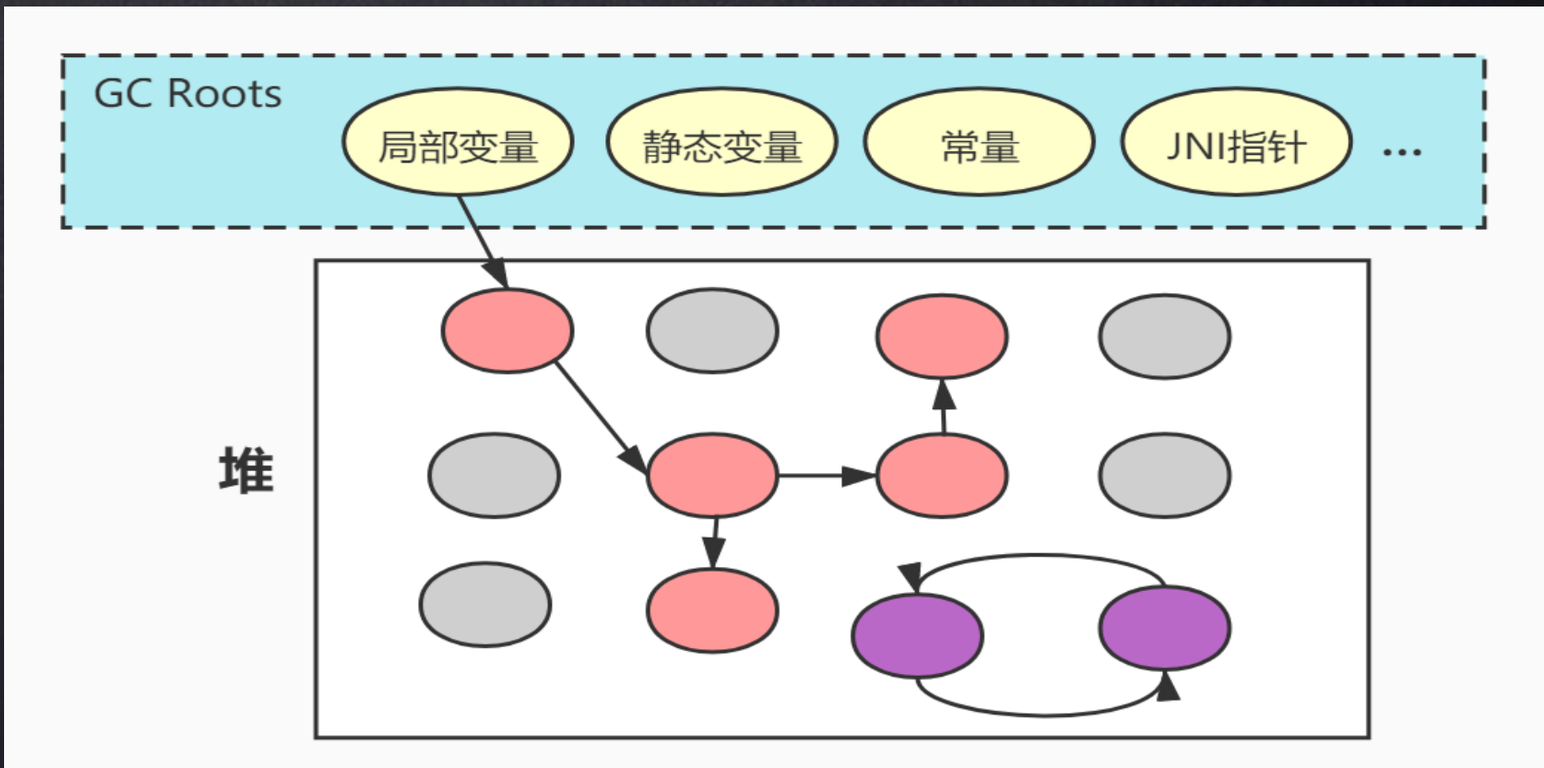
ZGC整体流程

- 一次ZGC流程
  - 标记阶段(标识垃圾)
  - 转移阶段(对象复制或移动)



ZGC中的垃圾标识

- 根可达算法
- 作为GC Roots的对象主要包括下面4种
  - 虚拟机栈（栈帧中的本地变量表）
  - 方法区中类静态变量
  - 方法区中常量
  - 本地方法栈中JNI指针

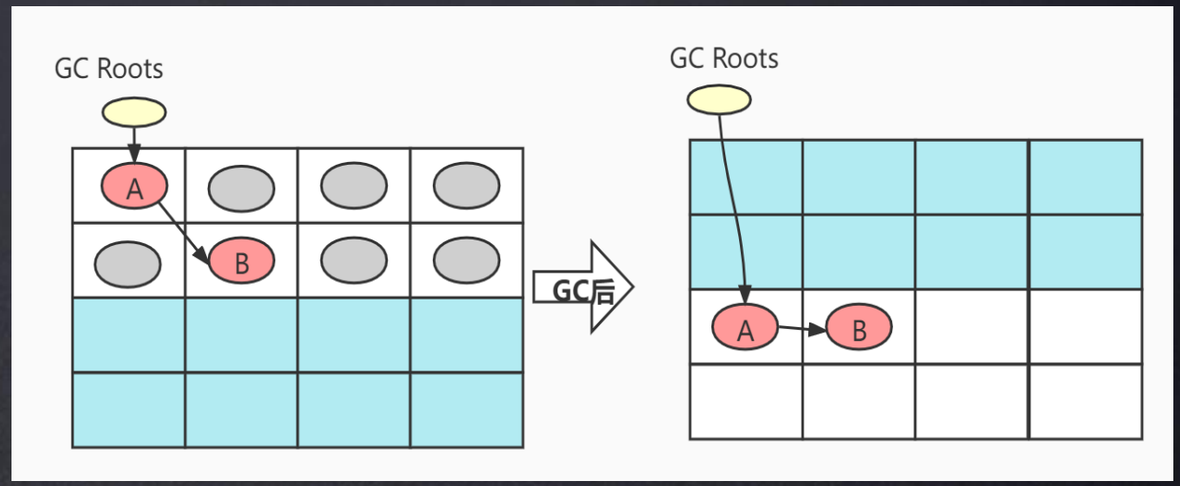


垃圾回收算法

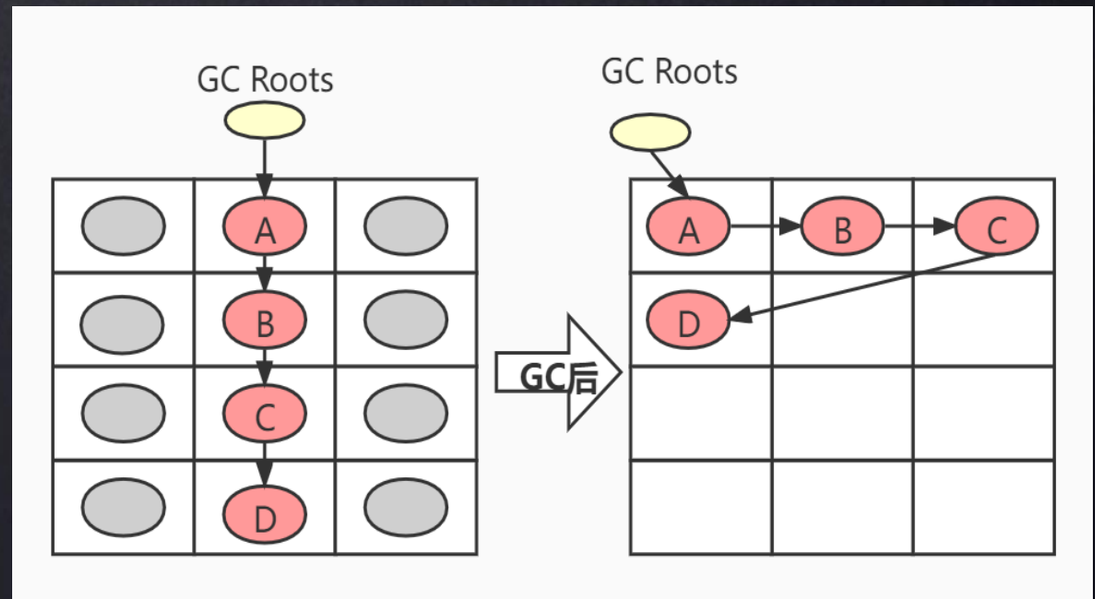
- ZGC中的 “转移”
  - 如果是同一个页面(等同标记整理)
  - 如果是不同页面(等同复制算法)

- **ZGC中垃圾回收算法流程**
  - 标记(mark):从根集合出发，标记活跃对象；此时内存中存在活跃对象和已死亡对象。
  - 转移(relocate)：把活跃对象转移（复制）到新的内存上，原来的内存空间可以回收。
  - 重定位(remap)：因为对象的内存地址发生了变化，所以所有指向对象老地址的指针都要调整到对象新的地址上。

复制算法



标记整理



ZGC基于指针着色的并发标记算法

■ ZGC中标记阶段

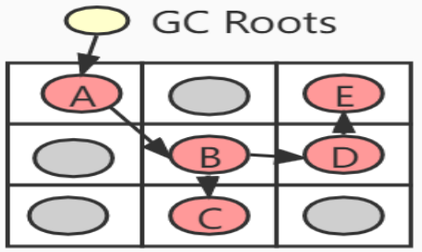
- **初始标记：**从根集合出发，找出根集合直接引用的活跃对象（图中A对象）
- **并发标记：**根据初始标记找到的根对象，使用深度优先遍历对象的成员变量进行标记（图中B,C,D,E对象）
- **再标记：**并发标记需要解决标记过程中引用关系变化导致的漏标记问题

■ 指针着色技术

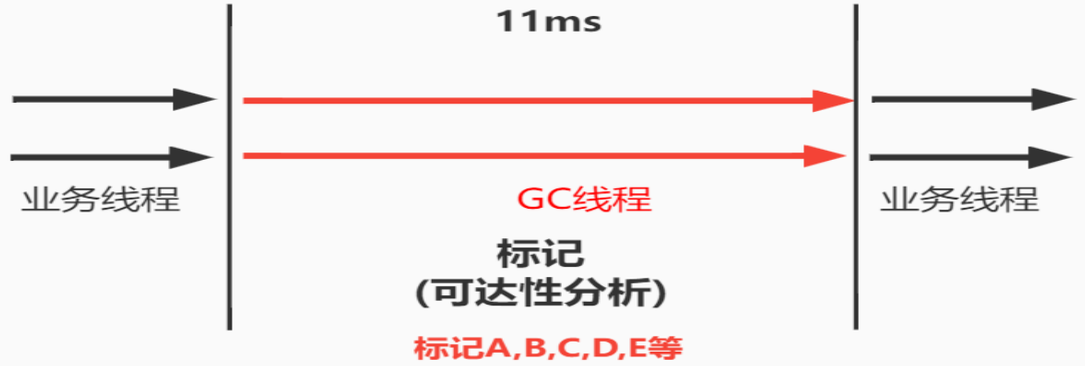
区分相邻两次GC中的标记

- M0(mark-0)
- M1(mark-1)

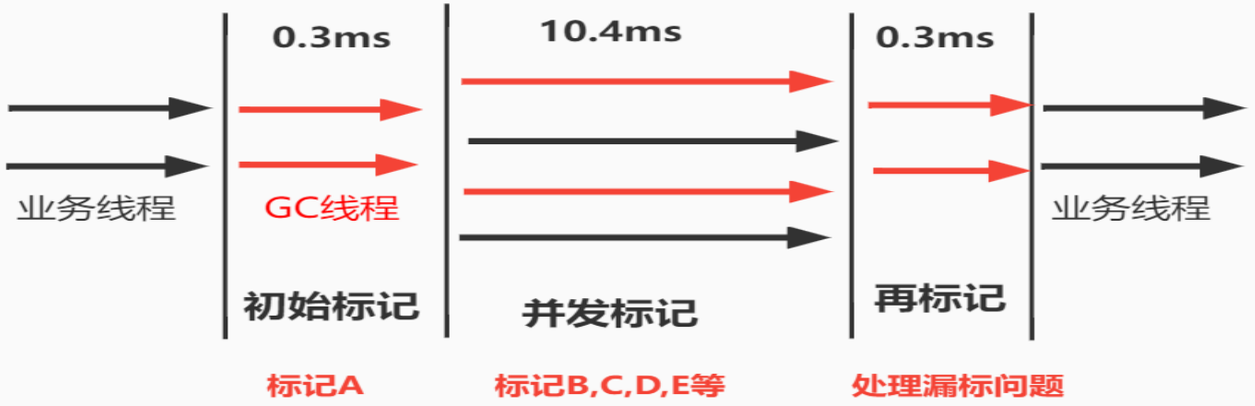
JDK8默认垃圾回收器
标记阶段
(STW 11ms)



堆空间



ZGC标记阶段
(STW 0.6ms)



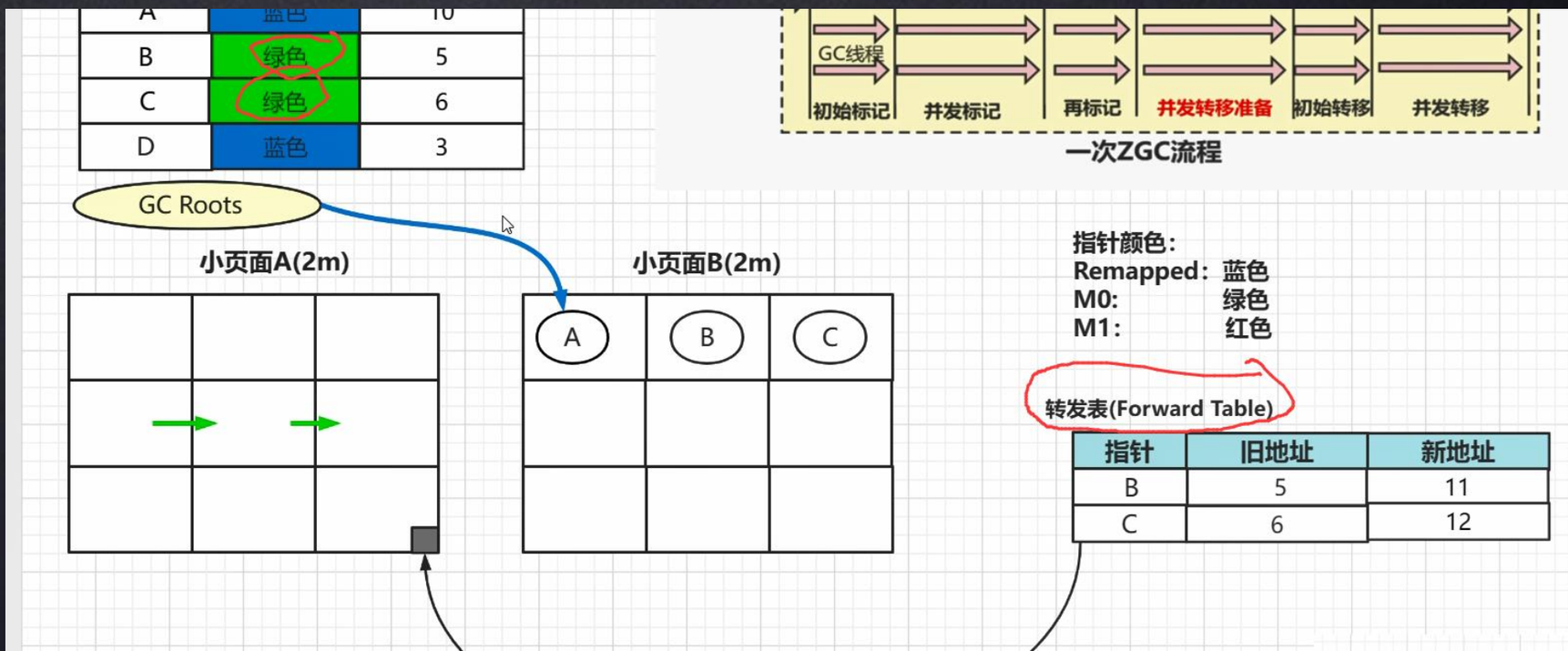
ZGC基于指针着色的并发转移算法

ZGC的转移阶段

- 并发转移准备(分析最有价值GC分页<无STW >)
- 初始转移（转移初始标记的存活对象同时做对象重定位<有STW> ）
- 并发转移（对转移并发标记的存活对象做转移<无STW>）

如何做到并发转移？

- 转发表(类似于HashMap)
- 对象转移和插转发表做原子操作

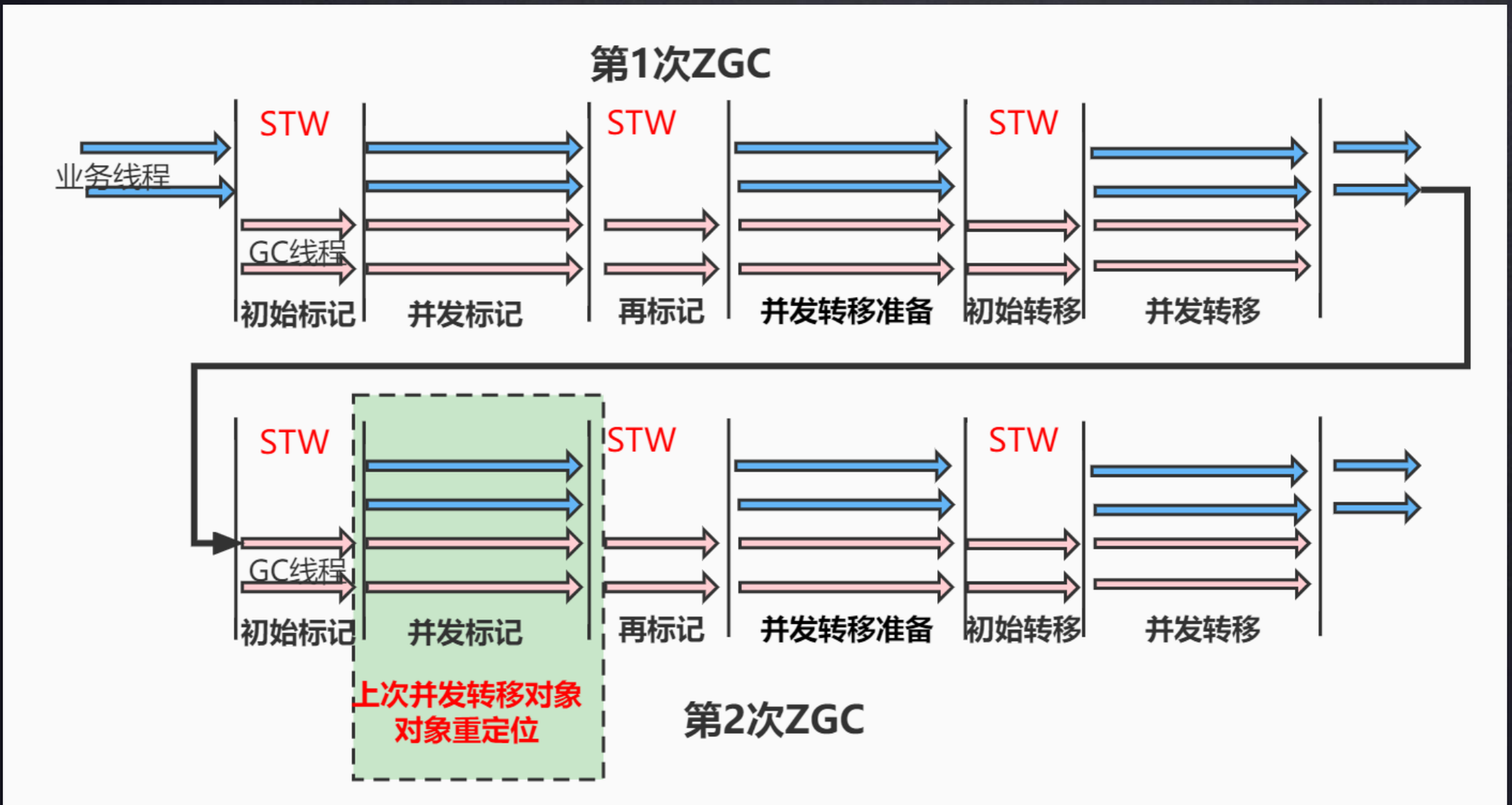


ZGC基于指针着色的重定位算法

■ 并发标记对象的重定位

- 下次GC中的并发标记（同时做上次并发标记对象的重定位）
- 技术上：指针着色中M0和M1区分

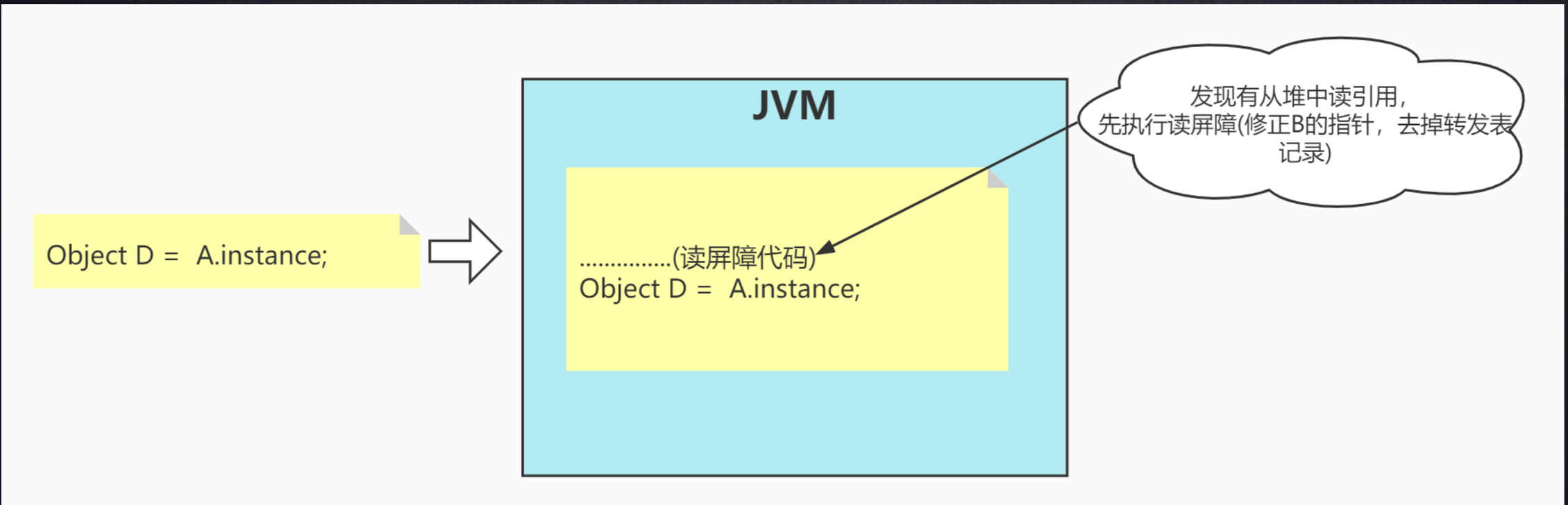
在两次GC中间业务线程
如何访问没有做完重定位的对象？



ZGC中的读屏障

- **涉及对象：**并发转移但还没做对象重定位的对象（着色指针使用M0和M1可以区分）
- **触发时机：**在两次GC之间业务线程访问这样的对象
- **触发操作：**对象重定位+删除转发表记录（两个一起做原子操作）

读屏障是JVM向应用代码插入一小段代码的技术。当应用线程从堆中读取对象引用时，就会执行这段代码。需要注意的是，仅“从堆中读取对象引用”才会触发这段代码



■ 启动参数

-XX:+UseZGC

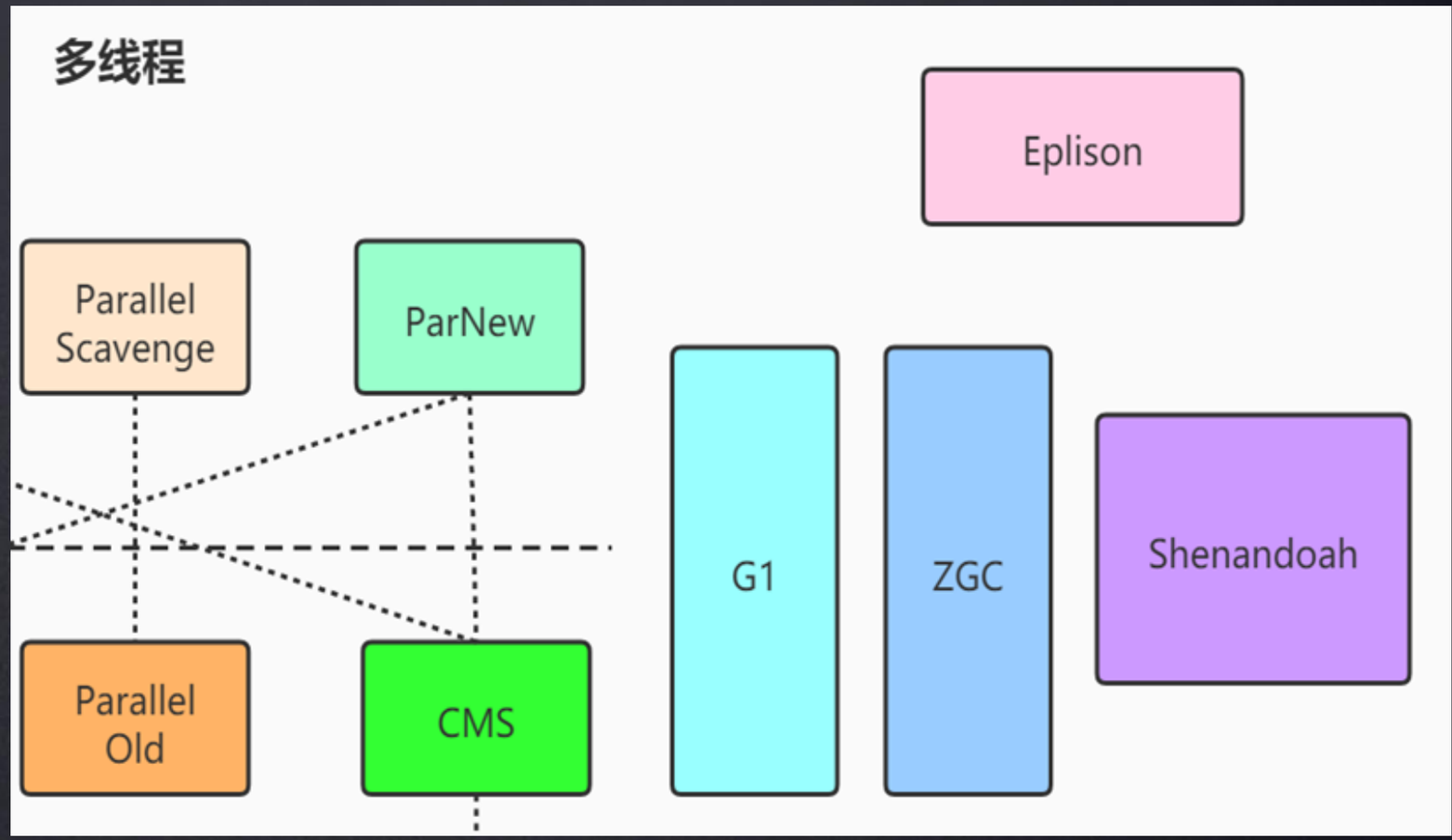
■ ZGC常见触发时机

- 基于固定时间间隔： ZCollectionInterval参数控制
- 基于分配速率的自适应算法(最主要)： ZAllocationSpikeTolerance控制
- 主动触发规则： Zproactive控制
- 启动预热：关键词warmup

ZGC实战

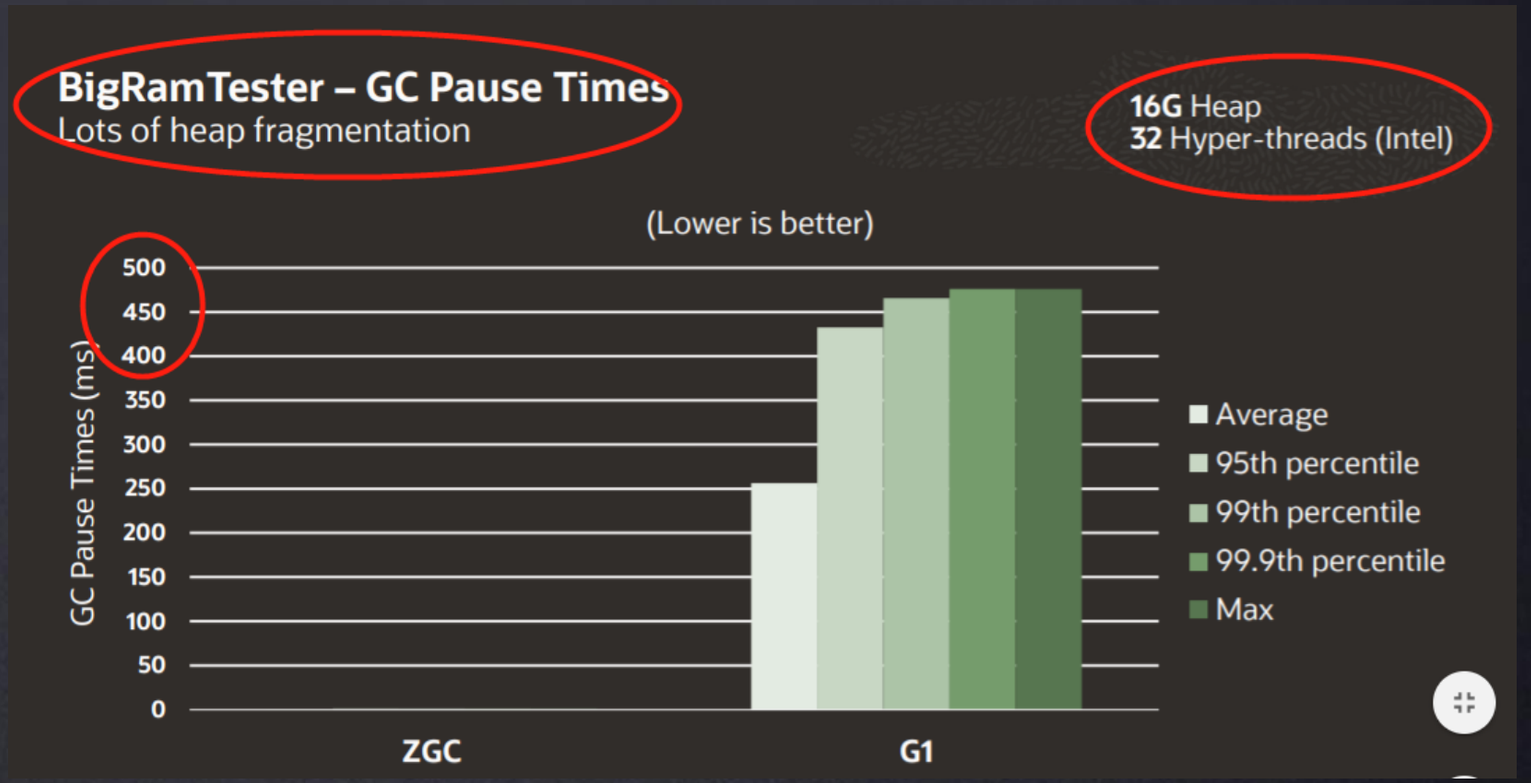
```
[4.491s][info    ][gc          ] GC(42) Pause Young (Normal) (G1 Evacuation Pause) 1033M->1035M(1282M) 30.409ms
[4.491s][info    ][gc,cpu      ] GC(42) User=0.20s Sys=0.06s Real=0.03s
```

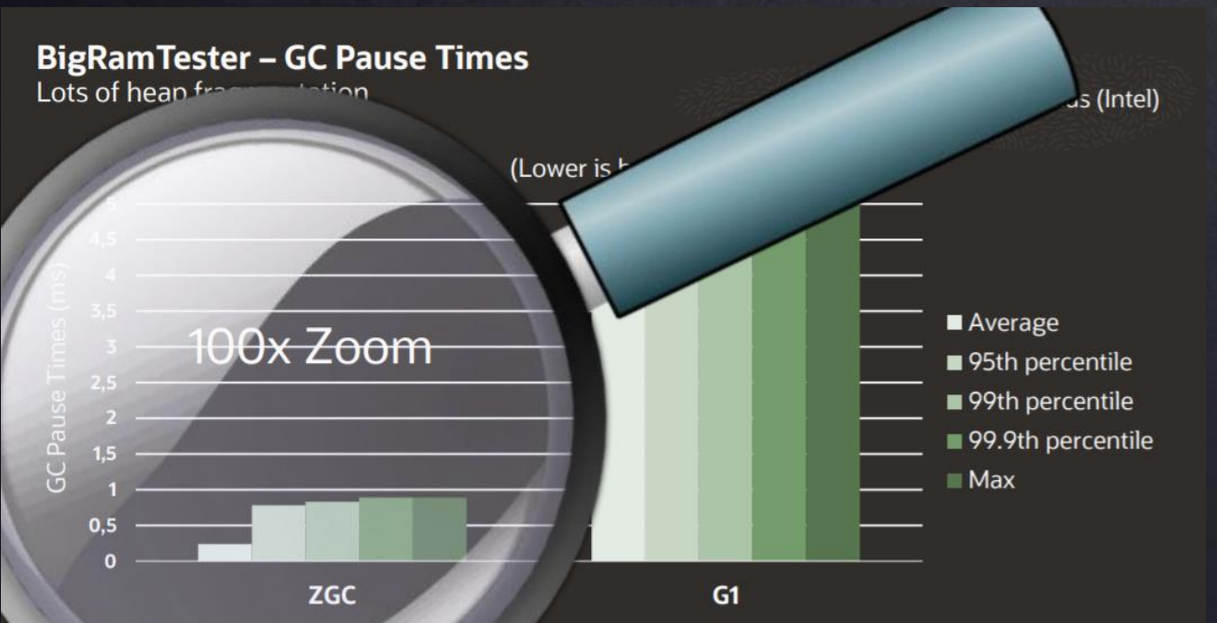
- 案例代码介绍
- 对比PS、G1、ZGC的STW
- ZGC大厂的运用
  - 阿里（借鉴ZGC优化自己JVM）
  - 美团（规则平台等）
  - 58（Hbase平台）
  - 腾讯（在线交互、竞价广告、量化交易等）
  - 华为（毕昇JDK、大数据项目）




```
] GC(77) Pause Young (Allocation Failure) 1351M->1364M(1820M) 197.754ms
```

ZGC与G1测试对比


BigRamTester – GC Pause Times
Lots of heap fragmentation
16G Heap
32 Hyper-threads (Intel)
(Lower is better)
GC Pause Times (ms)
500
450
400
350
300
250
200
150
100
50
0
ZGC
G1
Average
95th percentile
99th percentile
99.9th percentile
Max


BigRamTester – GC Pause Times
100x Zoom
GC Pause Times (ms)
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
ZGC
G1
Average
95th percentile
99th percentile
99.9th percentile
Max