



Java 自动内存管理技术

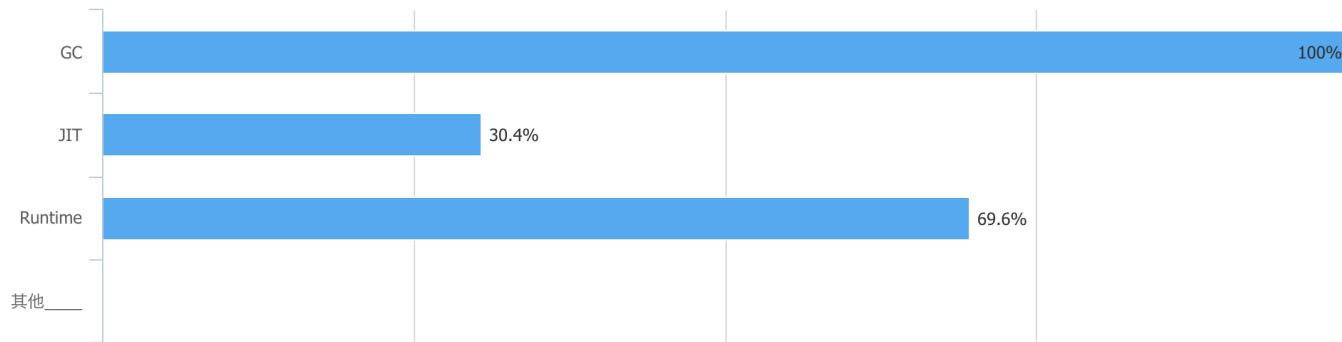
现状和未来

陆传胜 @阿里巴巴



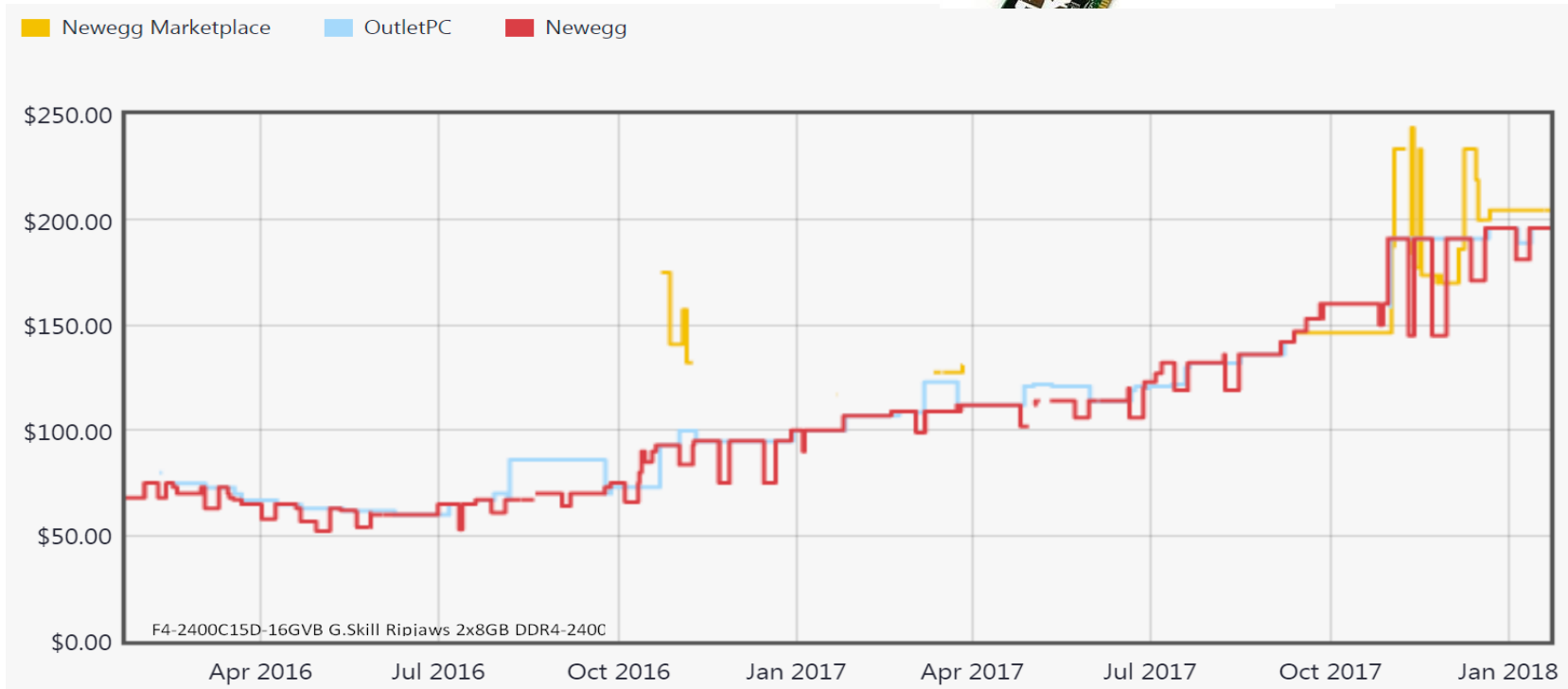
0x1,为什么要讨论内存管理

GreenTea JUG 2018-03

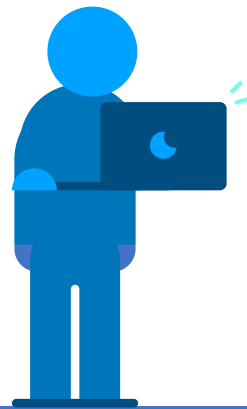


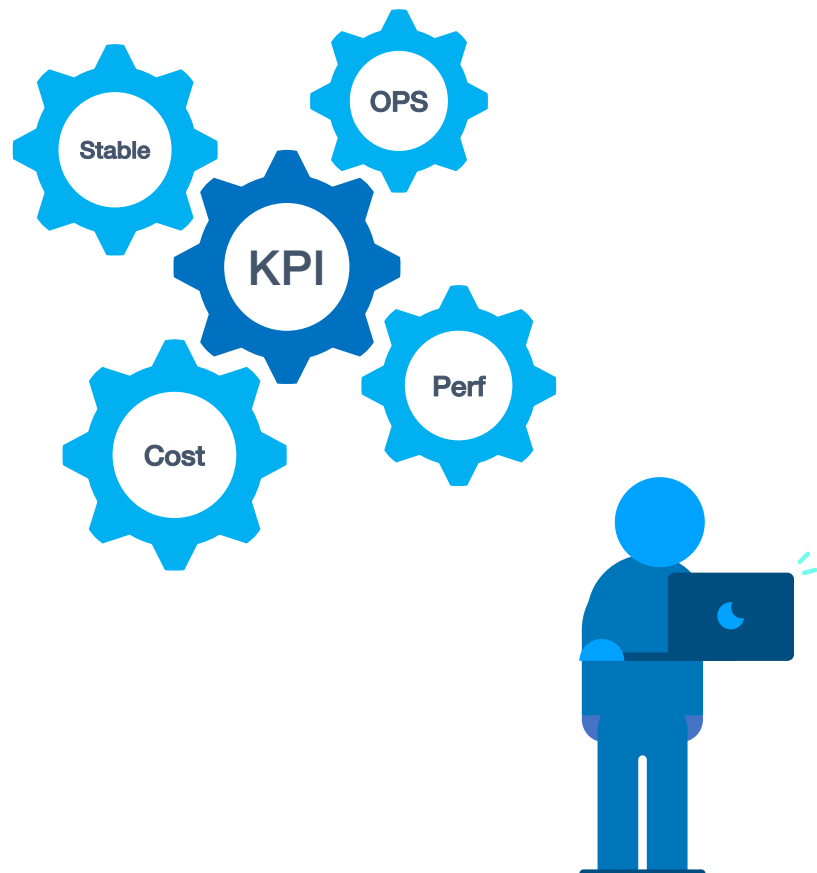
<http://greenteajug.cn/>

也许是因为新的投资方式

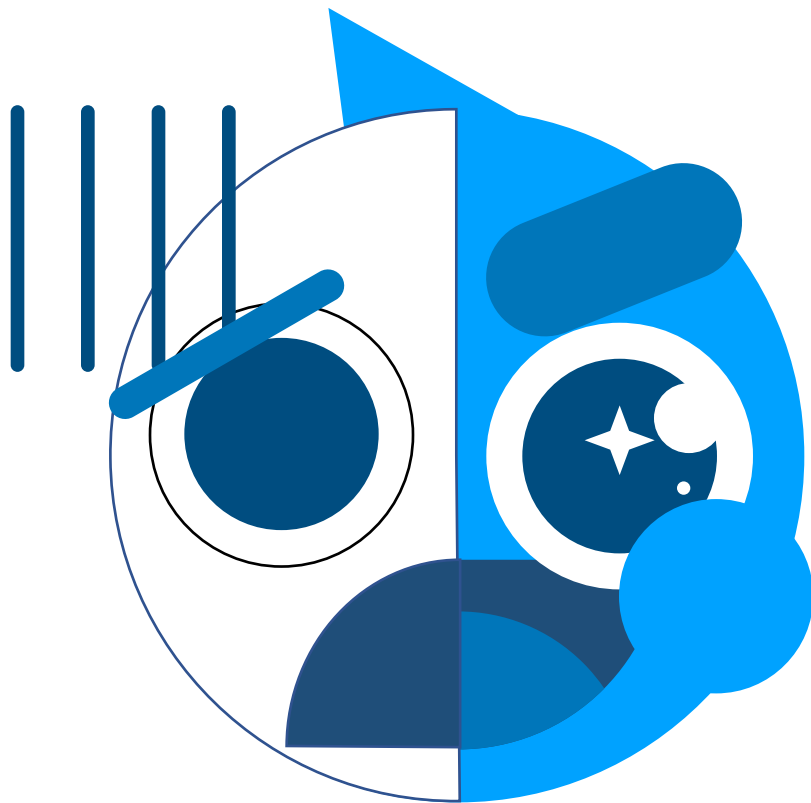


WHO

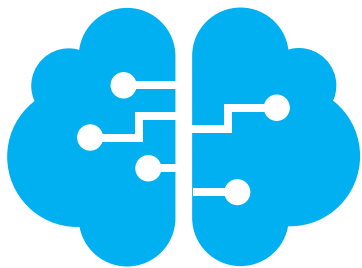




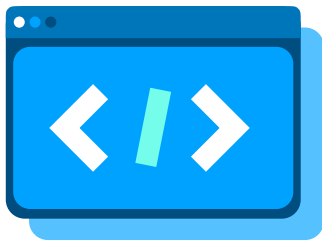
价值和手段



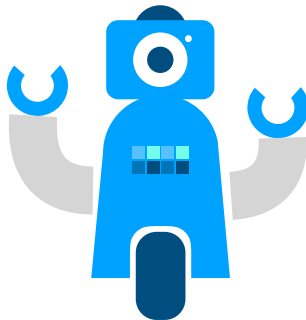
本次视角



互联网行业



工程实践

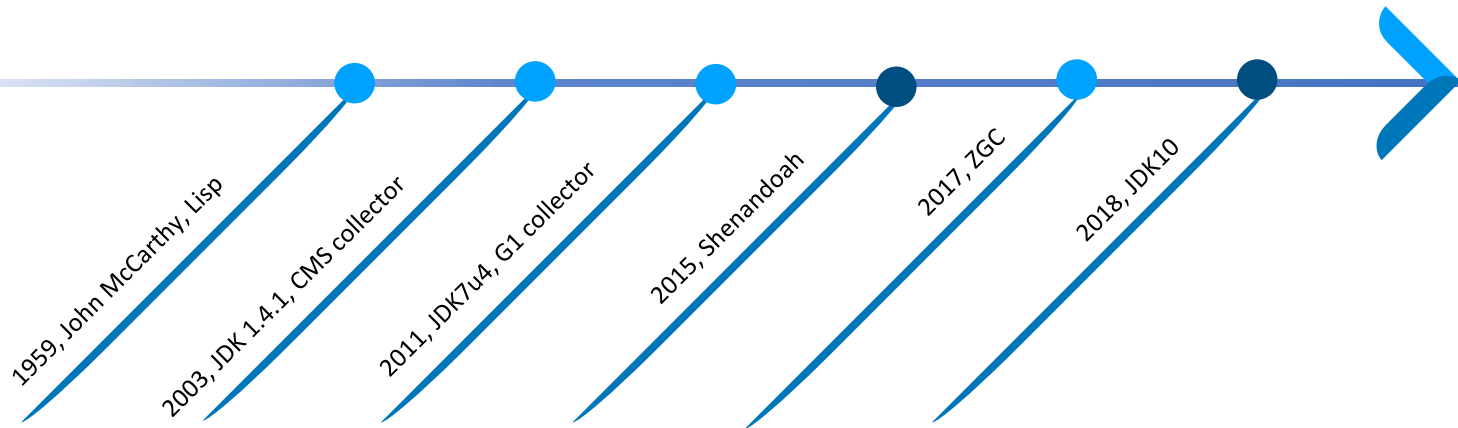


“旁观者”

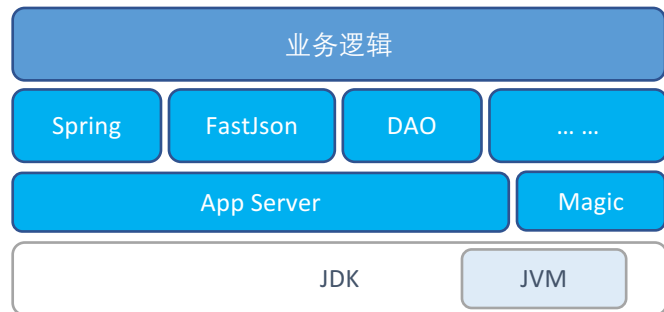
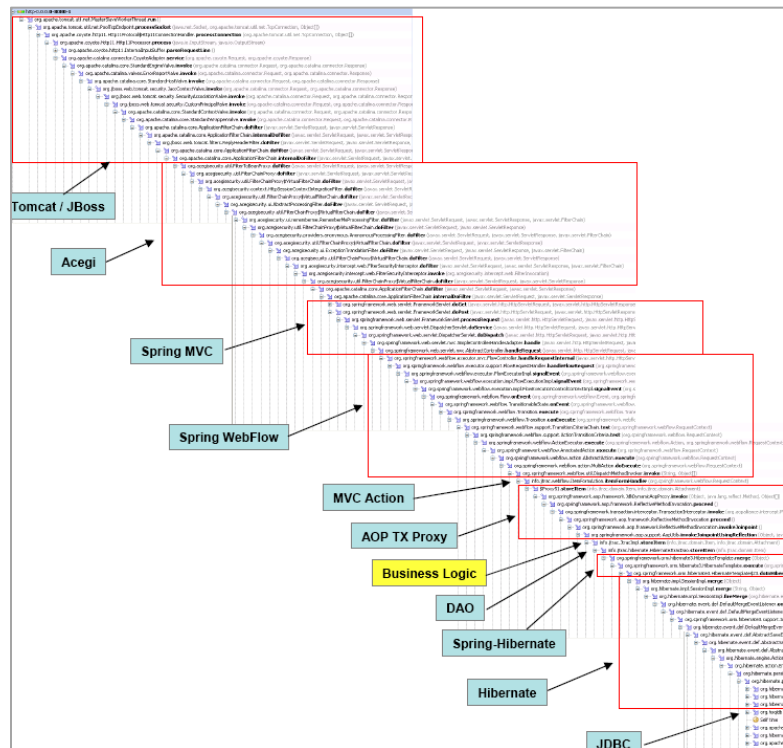
内存管理的元素



蓦然回首



现代Java系统的趋势



The Central Repository

Date index last refreshed: 2018-04-04 01:38:34 UTC

Total number of artifacts indexed (GAV): 2,689,975

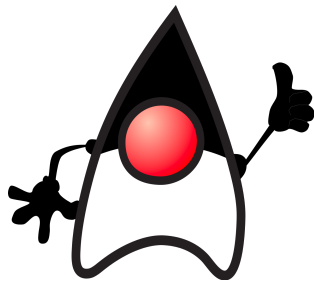
Total number of unique artifacts indexed (GA): 227,321

Current size of repository on disk: 0 MB

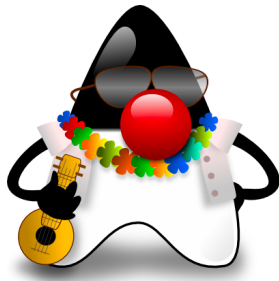
对内存管理的挑战

- 复杂的对象生命周期模式
- 难以预料的内存使用方式
- 多线程与内存
- 可预测性和易用性
- 性能

互联网公司的选择



- JDK自带的回收器
- 可以胜任90%的工作



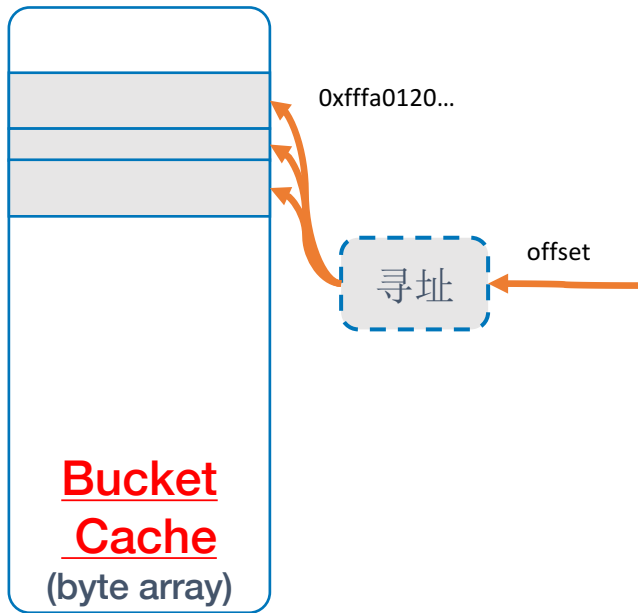
- 深度定制现有实现
- 阿里



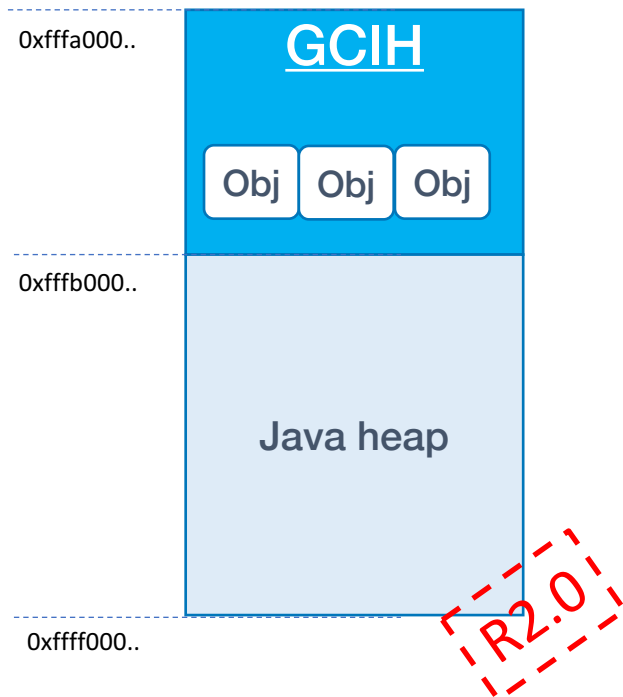
- 商业版垃圾回收器
- C4回收器

0x2,最好的内存管理是不用管理

一个尝试: Hbase Bucket Cache



另一个尝试: GCIH (GC Invisible Heap)



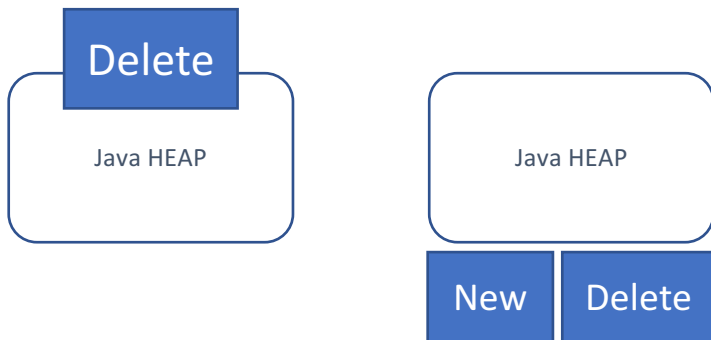
破罐子破摔.....

- 干脆什么都不要做了
- 场景
 - 短小的批处理任务
 - 测试任务...
- Epsilon GC
 - <http://openjdk.java.net/jeps/318>



如果您能接受的话....

- 我们给Java来添加
 - delete 操作！
- 半自动管理内存
 - GC + Delete
 - New + Delete



BridgedHeap

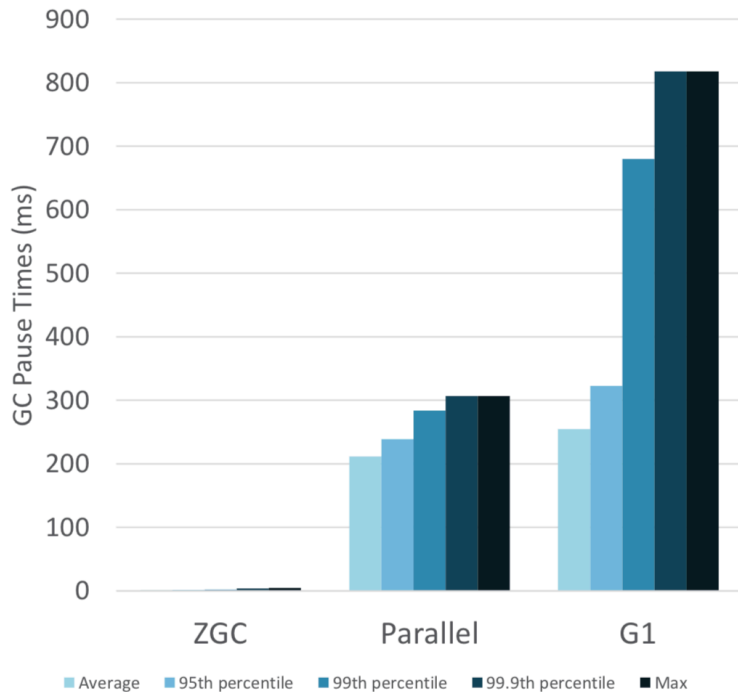
<https://github.com/luchsh/bridgedHeap>

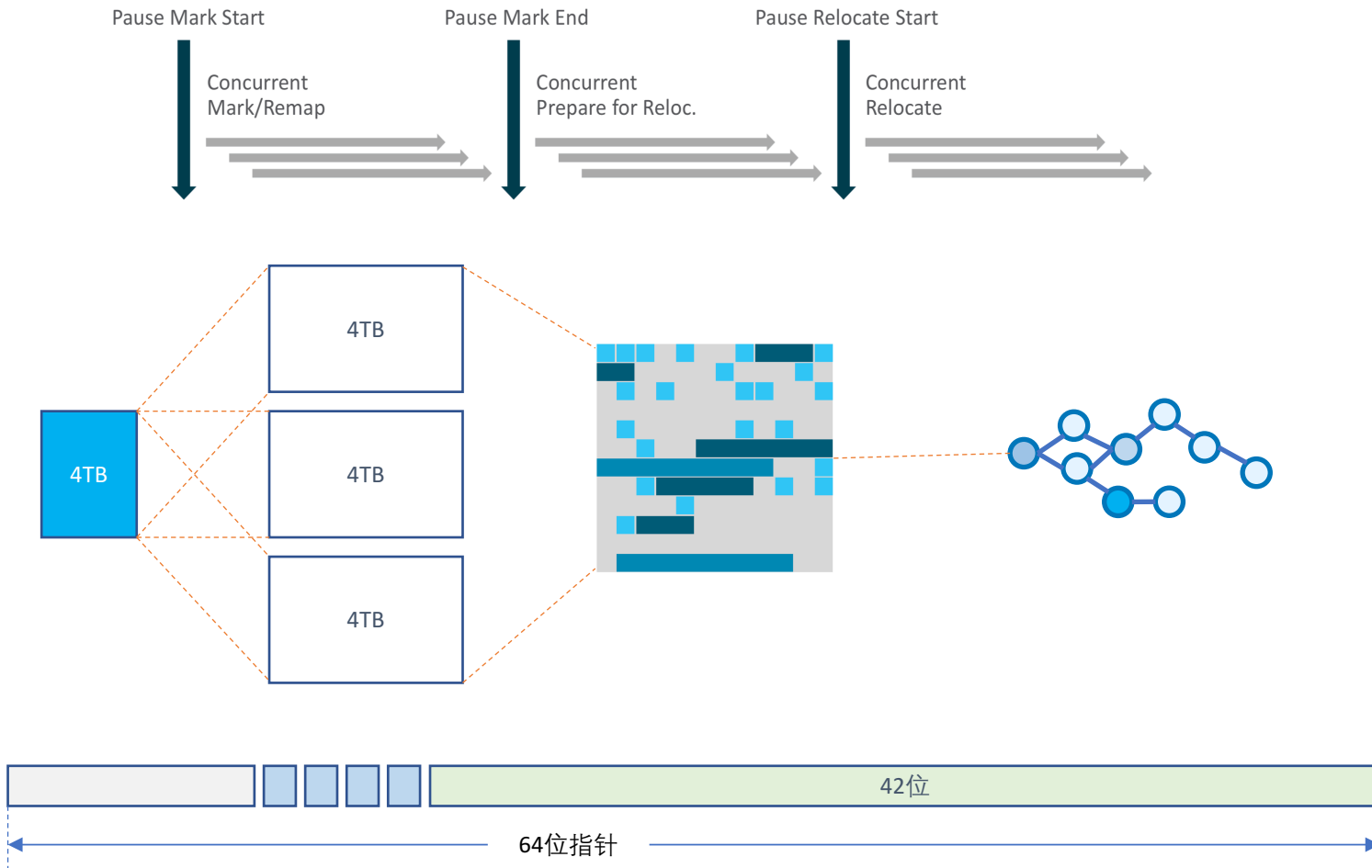


0x3, 降低内存管理的开销

RI的动向

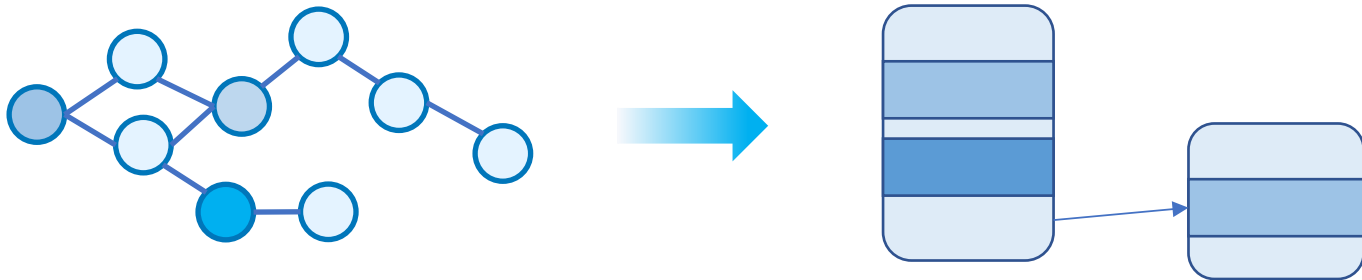
- ZGC
 - 固定时间暂停
 - 停顿时间大大缓解
 - 但是throughput有下降
- 可能是未来RI GC工作的基础
- Load barrier
- 15% throughput损失





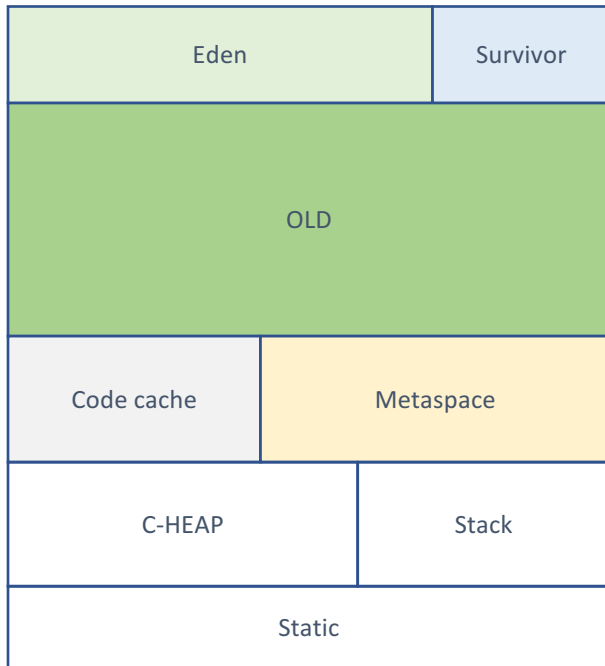
对象和类型之上

- Java 语言的局限性限制了GC的优化
- 虽然有Project Valhalla



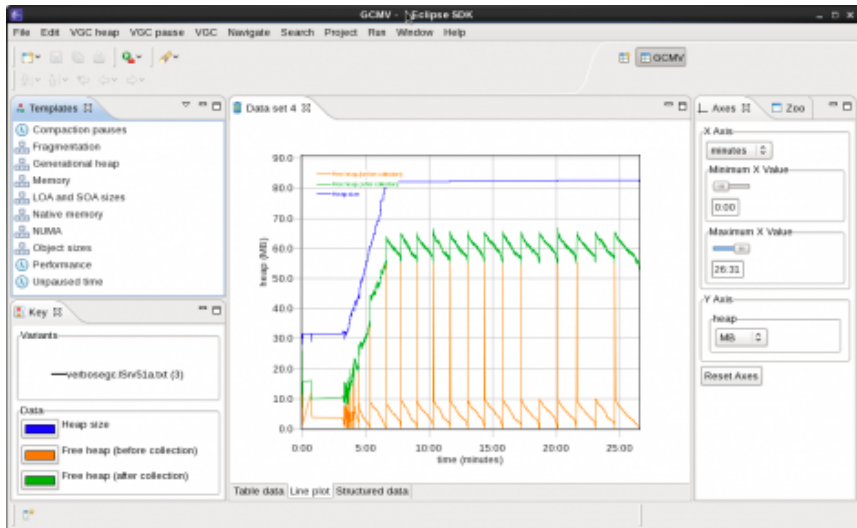
对于应用行为的观察

- 观察内存
 - 不只有heap
 - 分布
 - 声明周期
 - 大小
 - 分配
 - 速率
 - 比例



工欲善其事，必先profiling

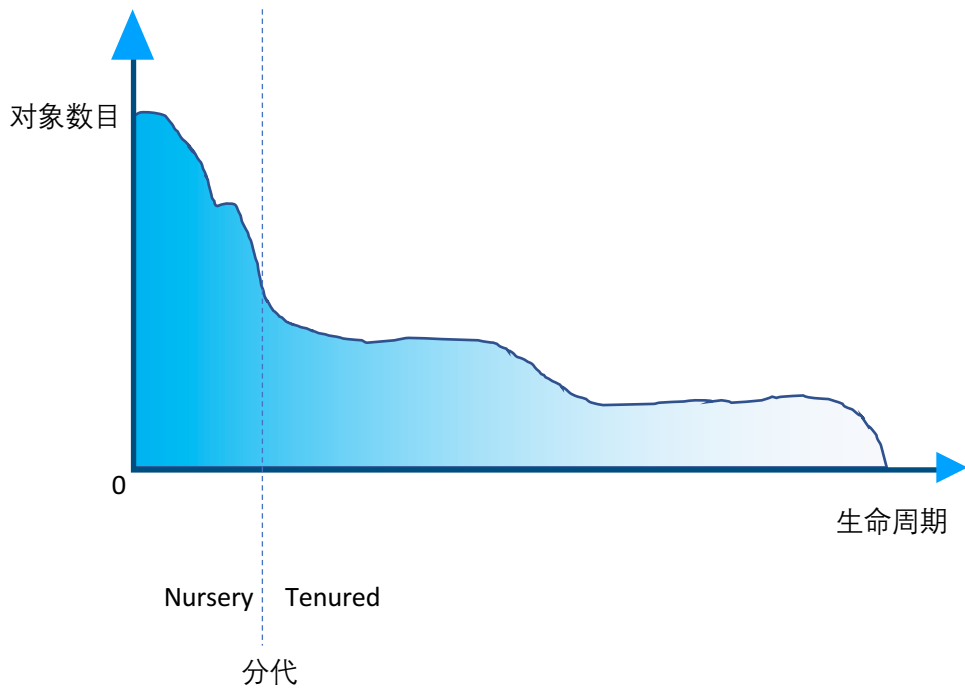
- 宏观
 - GC Log, 大部分的APM工具都是基于此
- 微观
 - Allocation site
 - 全生命周期计数
 - 死亡节奏
 - 对象移动
 - 对象图变更



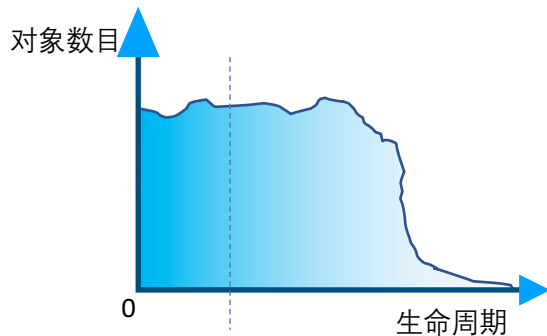
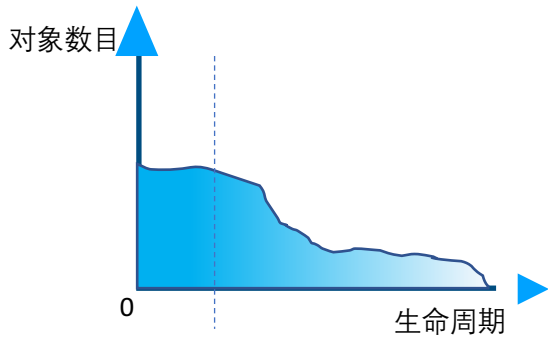
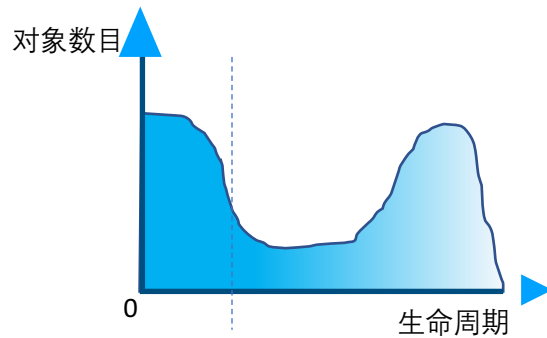
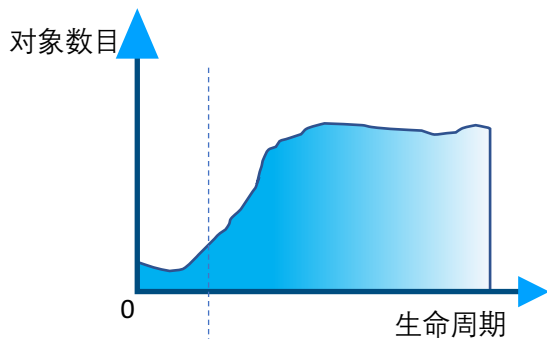
更细粒度的分配行为

- 复杂的分配方式导致难以profile
- TLAB分配的，特别是JIT生成的代码
- 需要采集的信息导致海量数据
 - Stack
 - Age
 - Object move

对象生命周期与分代假说



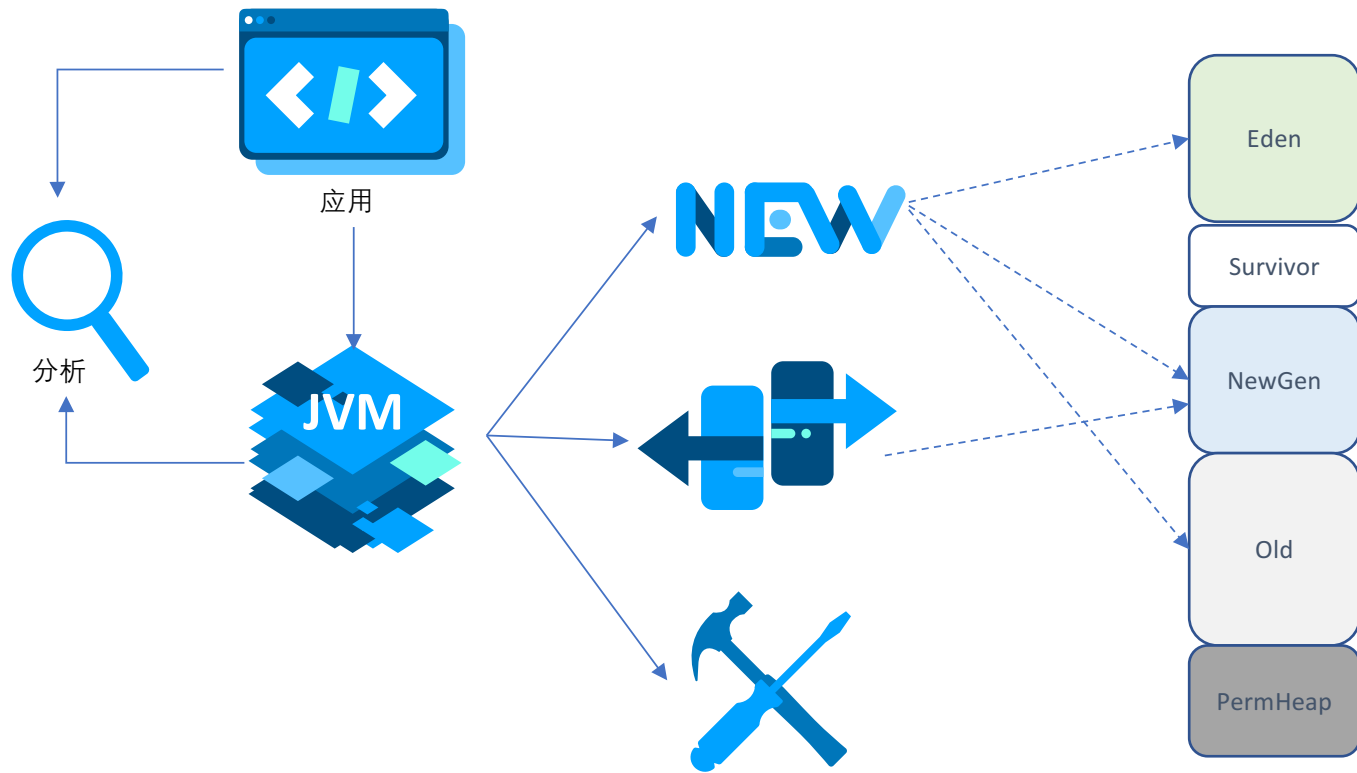
分代假说的例外



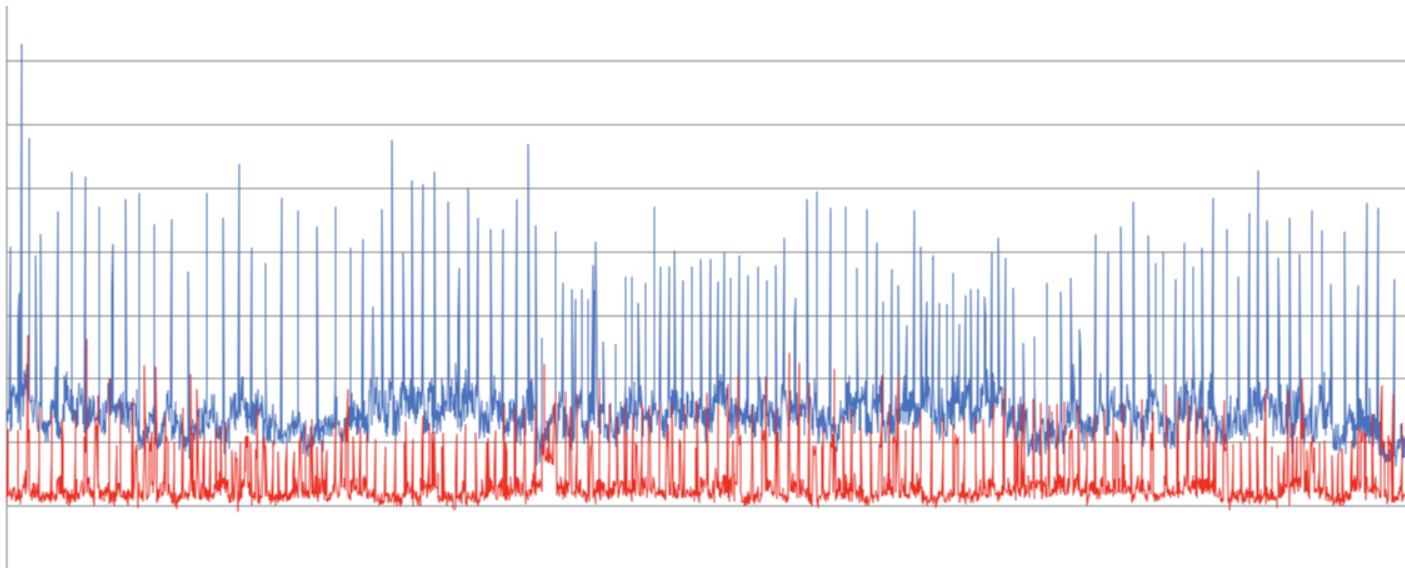
我们的一些探索：

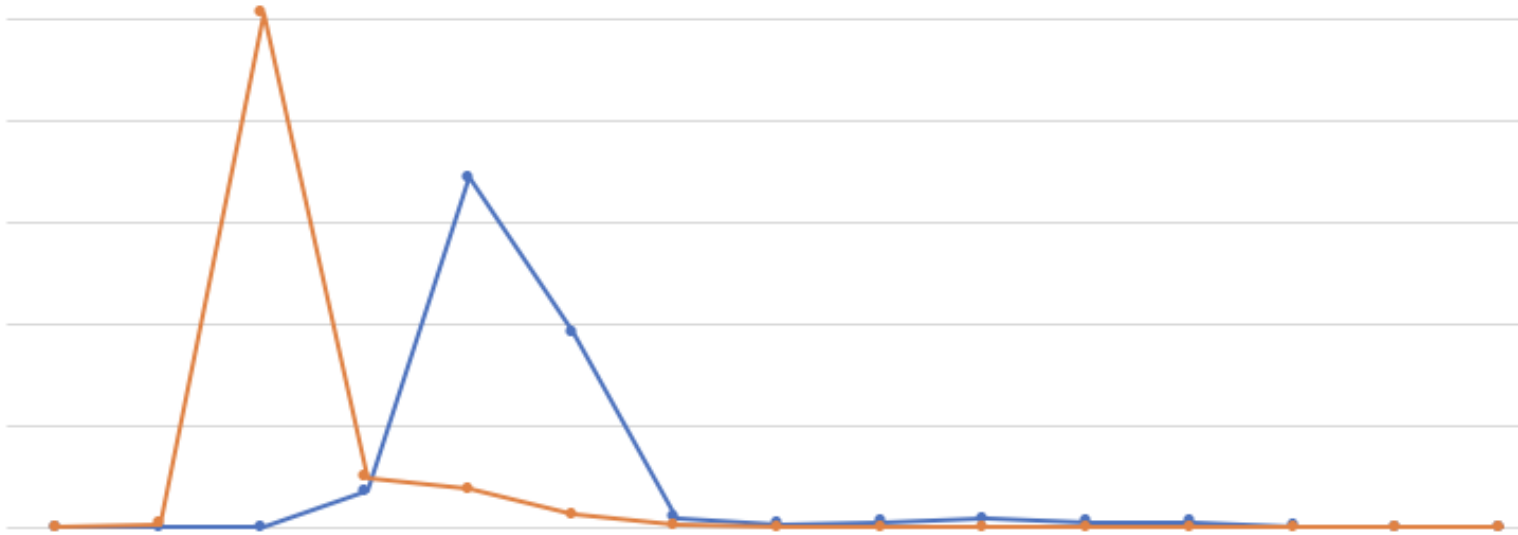
- 一个现象
 - 无序的内存分布影响Gc效率
- VM和应用的结合
- 一个解法
 - 局部有序的对象分布
 - 不同的对象分布区域区域
 - 按照区域的方式进行回收
- 没有throughput损失！





性能测试

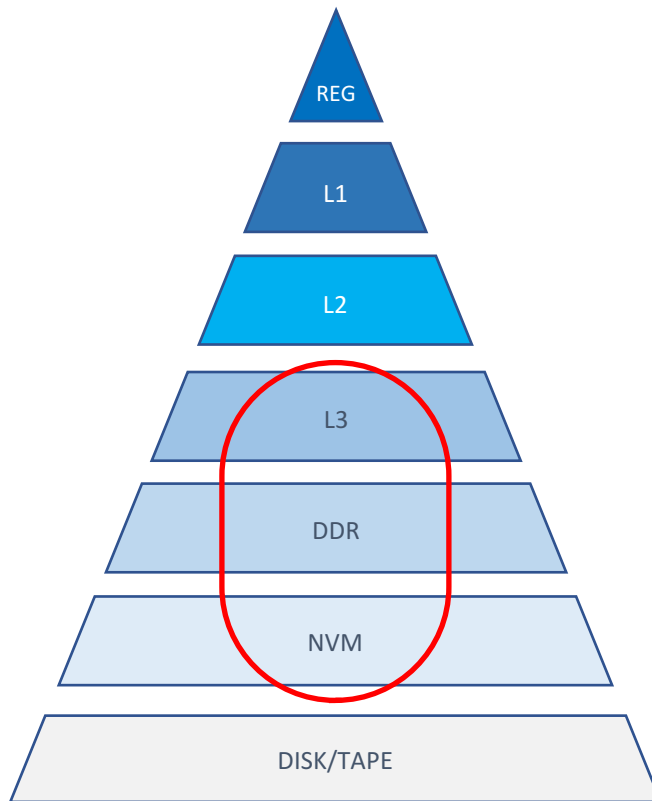




0x4, 榨取更多的好处

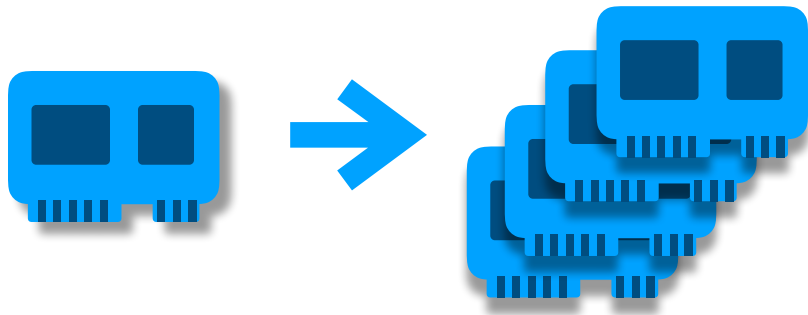
新硬件

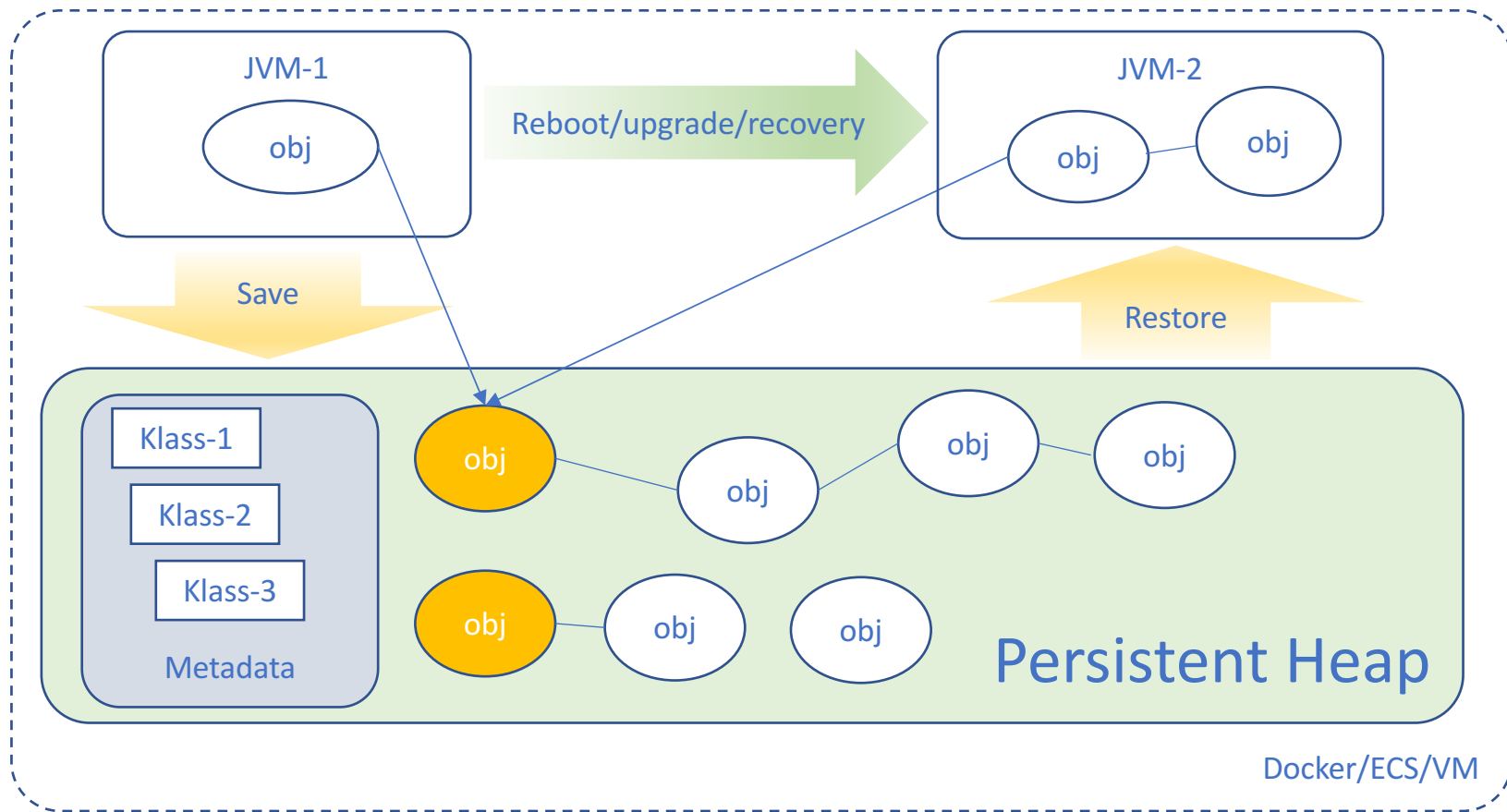
- 潜在的颠覆性
 - 更大
 - 更快
 - 更可控
 - 成本优势



Non-Volatile Memory

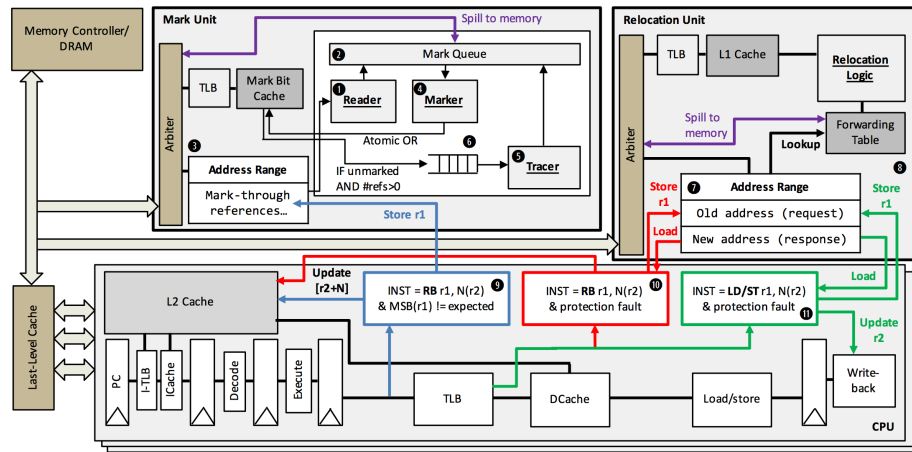
- 数据断电后不会丢掉
- 较快的访问速度
- 巨大的容量
- 低廉的价格





置于硬件的垃圾回收

- 快
 - 直接可控MMU
 - 廉价的barrier
 - 硬件“线程”

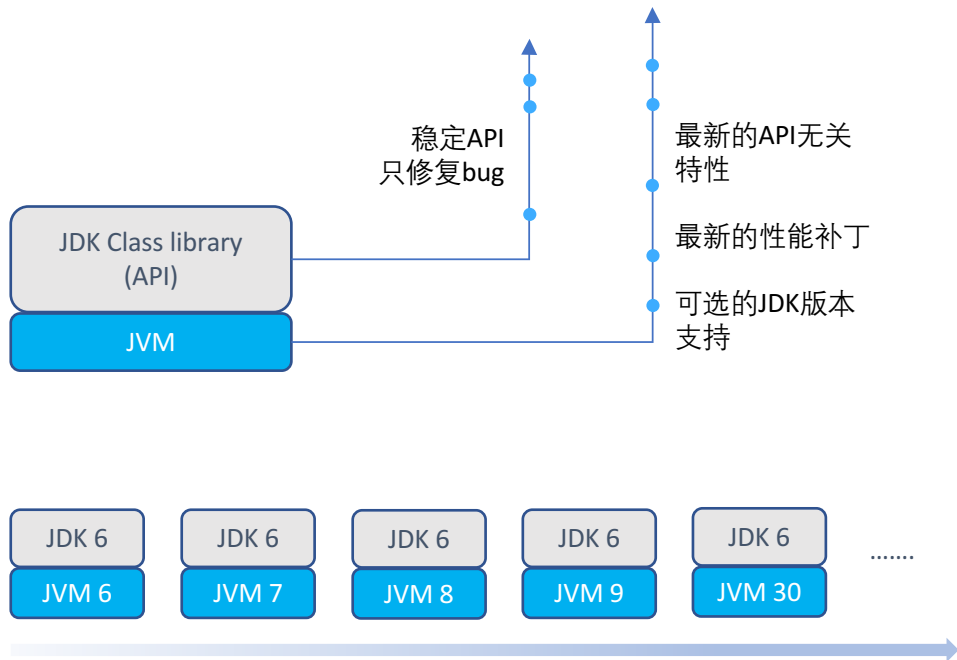


总结

- 短期的现实
 - 还得暂停
 - 并没有通用的银弹
- 可以期待
 - 标准GC接口的确定
 - 大堆无暂停的稳定GC
 - 更多开放的实现
 - Java作为通用平台

BACKUP

JVM-JDK独立发展



云计算

- 围观层面
 - Jvm是一个进程
 - 配合调度
- 宏观层面
 - 统一的平台
 - 互相之间的配合和通信