SATB算法
是为增量式标记清除垃圾收集器设计的快照标记算法，G1并发的基础就是SATB.
主要应用于垃圾收集的并发标记阶段. SATB可以理解成在GC开始之前对堆内存里的对象做一次快照，此时活的对象就认为是活的，从而形成一个对象图
步骤：

在初始标记的时候生成一个快照图，标记存活对象
在并发标记的时候出现了引用修改会把这些引用的原始值捕获下来，记录在log buffer中
在再次标记阶段扫描SATB，修正SATB的误差
可能存在浮动垃圾，将在下次被收集。

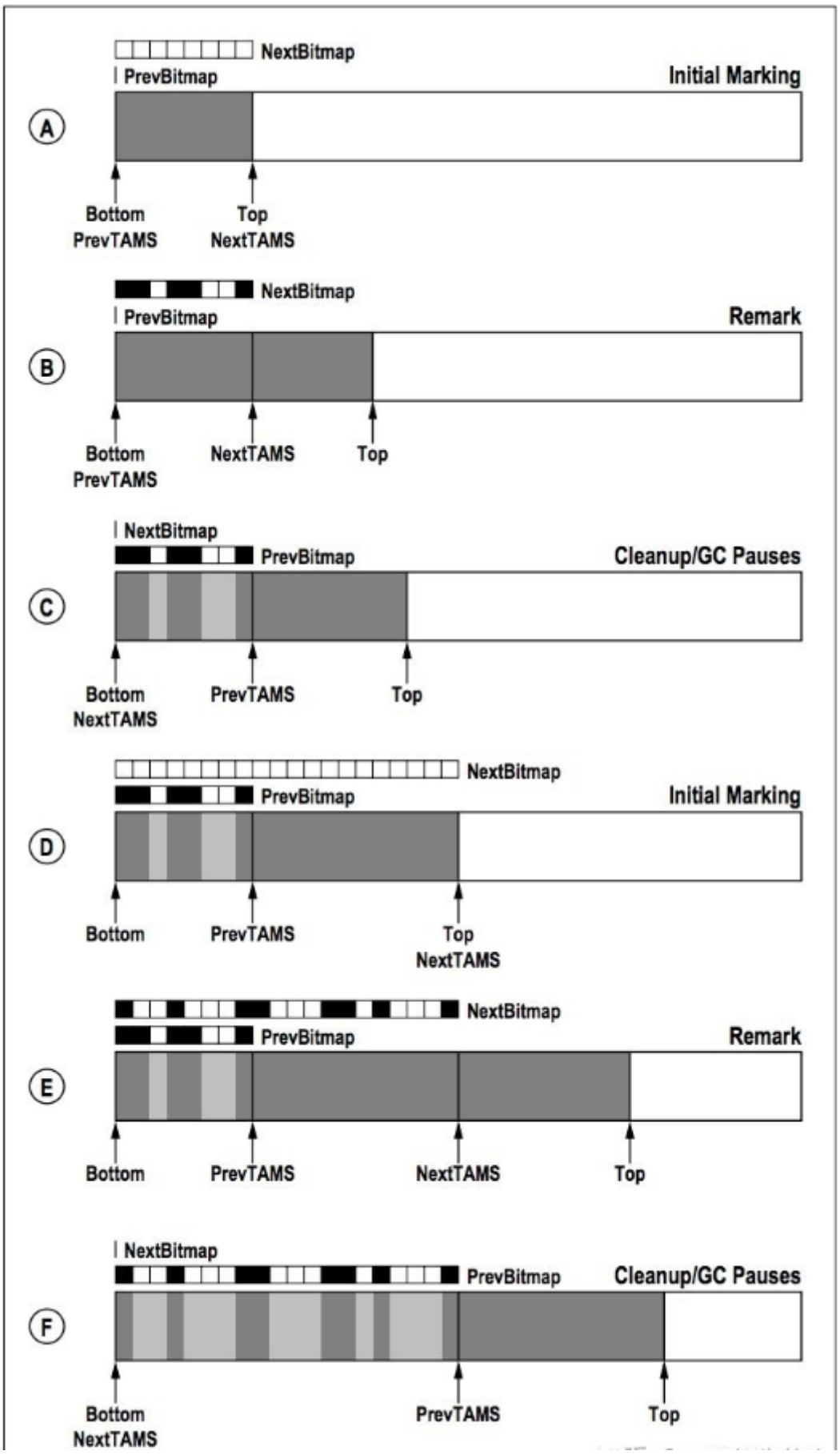
对象漏标
垃圾回收的并发标记阶段，GC线程和应用线程是并发执行的，所以一个对象被标记之后，应用线程可能篡改对象的引用关系，从而造成对象的漏标、误标，其实误标没什么关系，顶多造成浮动垃圾，在下次gc还是可以回收的，但是漏标的后果是致命的，把本应该存活的对象给回收了，从而影响的程序的正确性。

首先需要了解两个概念 三色算法 与 Region结构

三色算法
给每一个对象用颜色进行区分
黑色：自身以及可达对象都已经被标记
灰色：自身被标记，可达对象还未标记
白色：还未被标记
漏标的情况只会发生在白色对象中，且满足以下任意一个条件：

并发标记时，应用线程给一个黑色对象的引用类型字段赋值了白色对象
解决方案: 利用post-write barrier，记录所有新增的引用关系，然后根据这些引用关系为根重新扫描一遍。
并发标记时，应用线程删除所有灰色对象到白色对象的引用

解决方案：利用pre-write barrier，将所有既将被删除的引用关系的旧引用记录下来，最后以这些旧引用为根重新扫描一遍。
Region结构：每一个Region 包含了5个指针，分别是bottom、previous TAMS、next TAMS、top和end，其中previous TAMS、next TAMS是前后两次发生并发标记时的位置. 在prevTAMS和nextTAMS以上的对象就是新分配的


A
NextBitmap
PrevBitmap
Initial Marking
Bottom
PrevTAMS
Top
NextTAMS
B
NextBitmap
PrevBitmap
Remark
Bottom
PrevTAMS
NextTAMS
Top
C
NextBitmap
PrevBitmap
Cleanup/GC Pauses
Bottom
NextTAMS
PrevTAMS
Top
D
NextBitmap
PrevBitmap
Initial Marking
Bottom
PrevTAMS
Top
NextTAMS
E
NextBitmap
PrevBitmap
Remark
Bottom
PrevTAMS
NextTAMS
Top
F
NextBitmap
PrevBitmap
Cleanup/GC Pauses
Bottom
NextTAMS
PrevTAMS
Top

A是初始标记阶段。next TAMS尚未标记任何存活对象，而此时的previous TAMS被初始化为region内存地址起始值，next TAMS被初始化为top。top实际上就是一个region未分配区域和已分配区域的分界点；
B是并发标记之后，进入了再次标记阶段。此时存活对象的扫描已经完成了，因此next bitmap构造好了，代表的是当下状态中region中的内存使用情况。注意的是，此时top已经不再与next TAMS重合了，top和next TAMS之间的就是在前面标记阶段之时，新分配的对象
C代表的是clean up阶段。C和B比起来，next bitmap变成了previous bitmap，而在bitmap中标记为垃圾（也就是白色区域的）的对应的region的区域也被染成了浅灰色。这并不是指垃圾对象已经被清扫了，仅仅是标记出来了。
D代表的是下一个初始标记阶段，该阶段和A类似，next TAMS重新被初始化为top的值；
EF就是BC的重复；