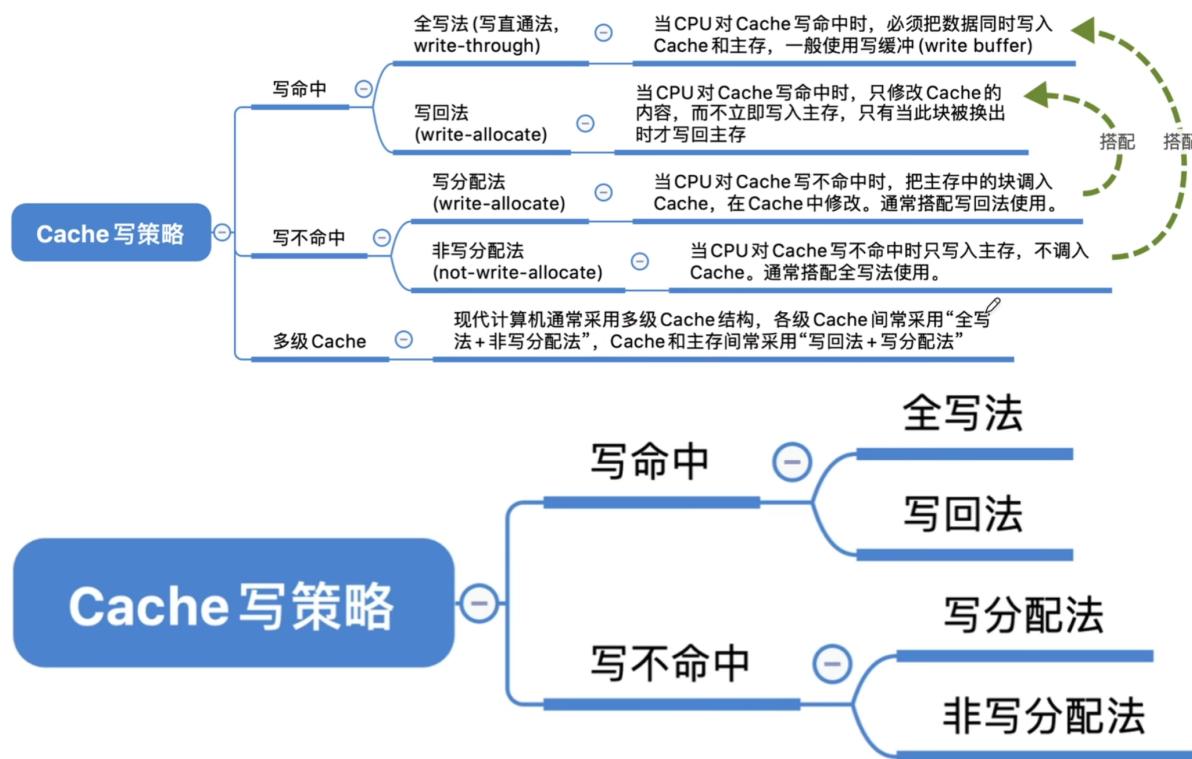


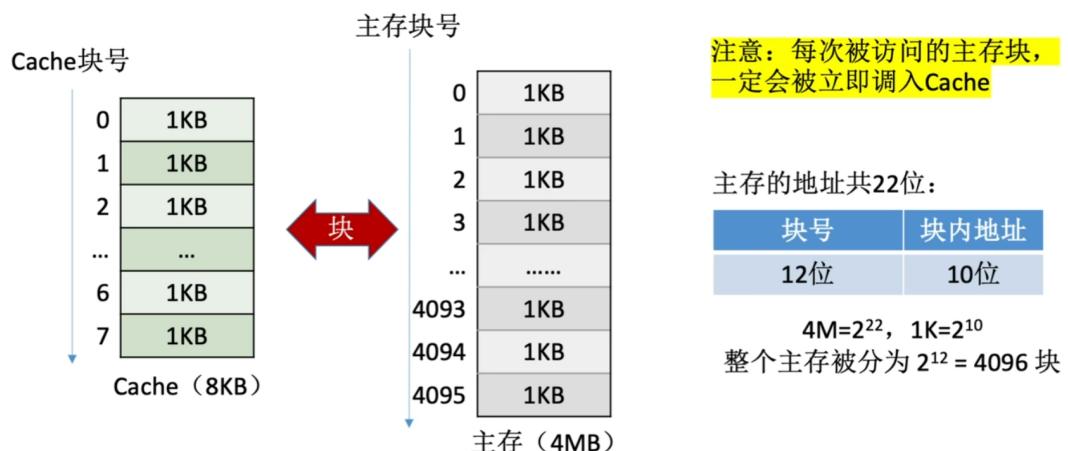
# Cache的一致性问题



为何不讨论读命中、读不命中的情况？

读操作不会导致Cache和主存的数据不一致

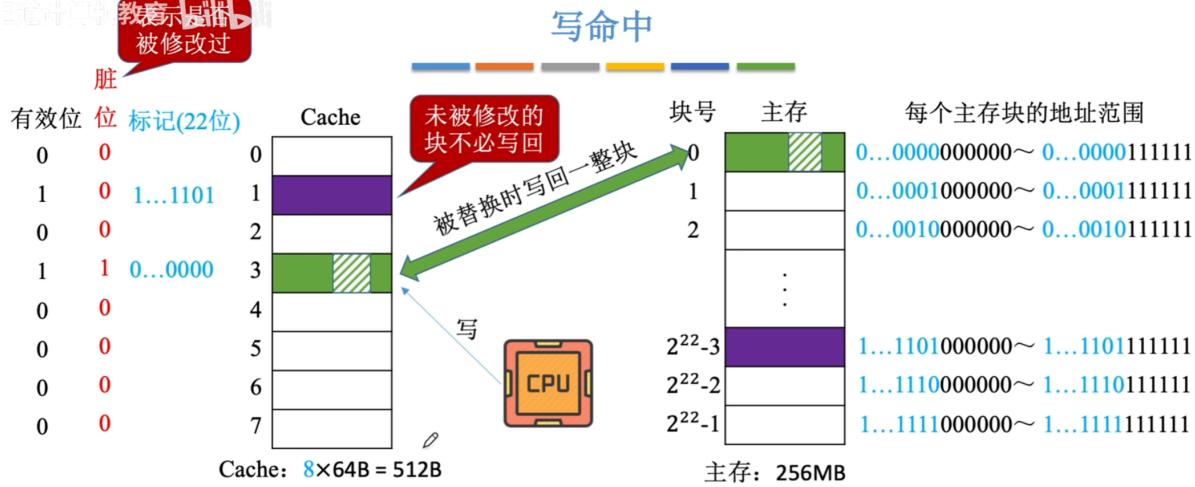
## 有待解决的问题



- 如何区分 Cache 与 主存 的数据块对应关系?
  - Cache 很小, 主存很大。如果Cache满了怎么办?
  - CPU修改了Cache中的数据副本, 如何确保主存中数据母本的一致性?
- Cache 和 主存 的映射方式  
— 替换算法  
— Cache 写策略

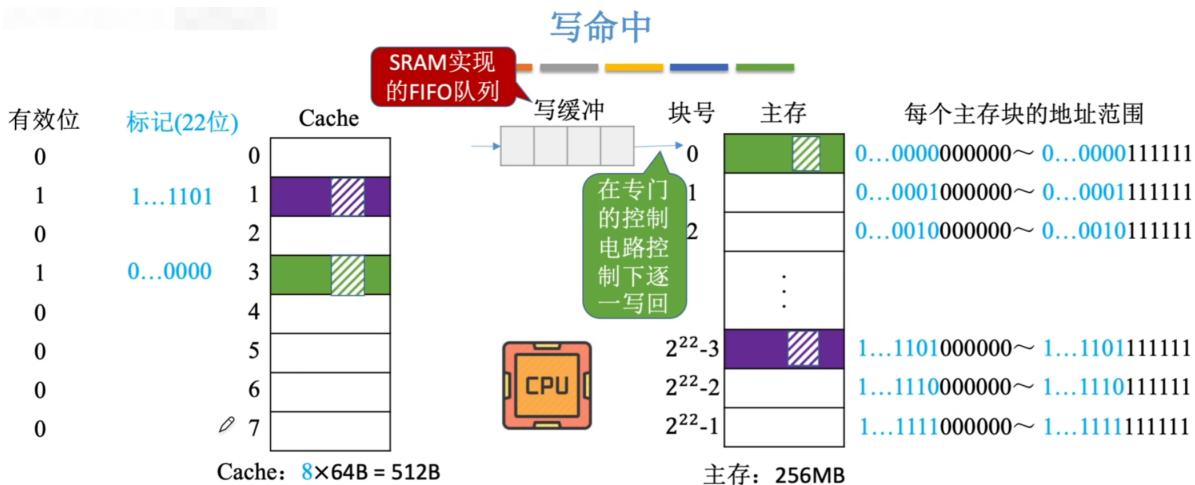
## 写命中

## 写回法



写回法(write-back) —— 当CPU对Cache写命中时，只修改Cache的内容，而不立即写入主存，只有当此块被换出时才写回主存  
减少了访存次数，但存在数据不一致的隐患。

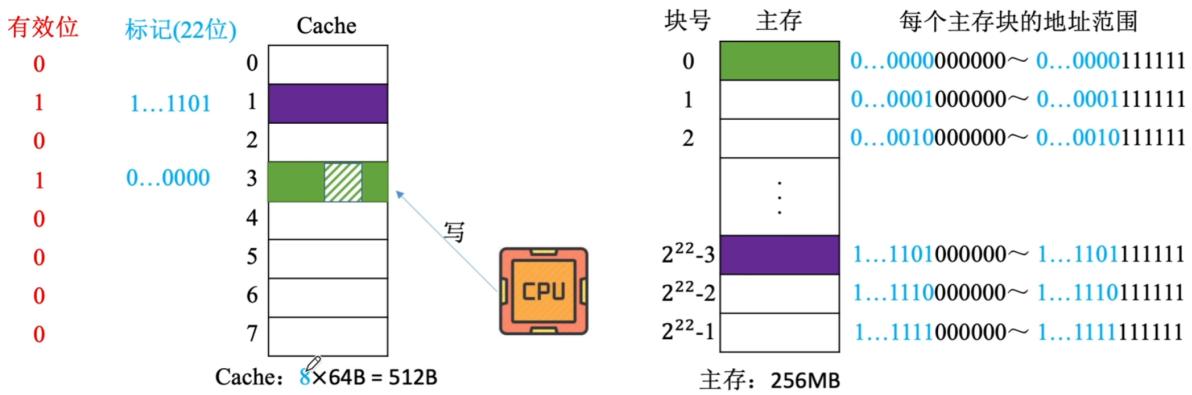
## 全写法



全写法(write-through) —— 当CPU对Cache写命中时，必须把数据同时写入Cache和主存，一般使用写缓冲(write buffer)  
使用写缓冲，CPU写的速度很快，若写操作不频繁，则效果很好。若写操作很频繁，可能会因为写缓冲饱和而发生阻塞

## 写不命中

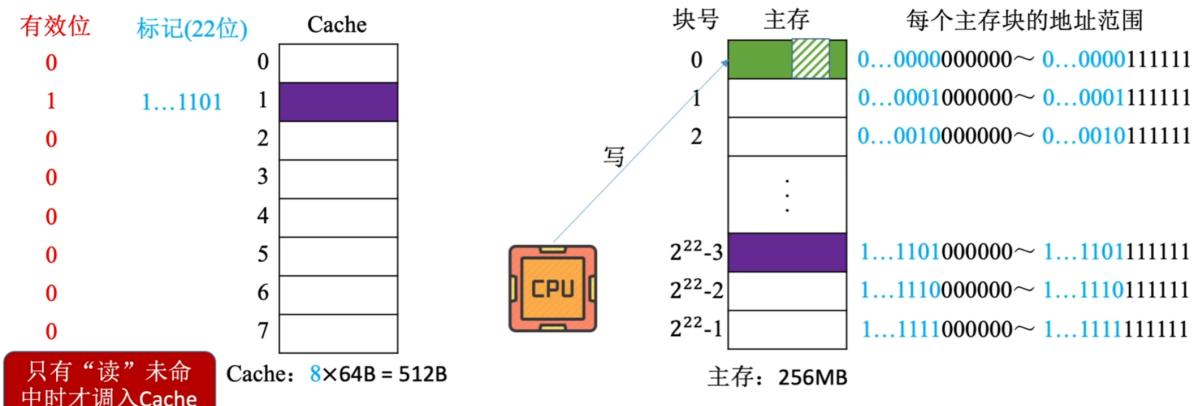
### 写分配法



写分配法(write-allocate)——当CPU对Cache写不命中时，把主存中的块调入Cache，在Cache中修改。通常搭配写回法使用。

写回法(write-back)——当CPU对Cache写命中时，只修改Cache的内容，而不立即写入主存，只有当此块被换出时才写回主存

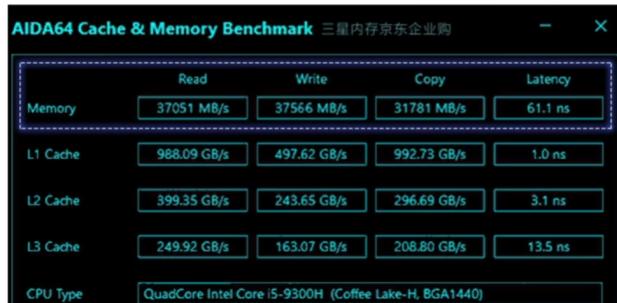
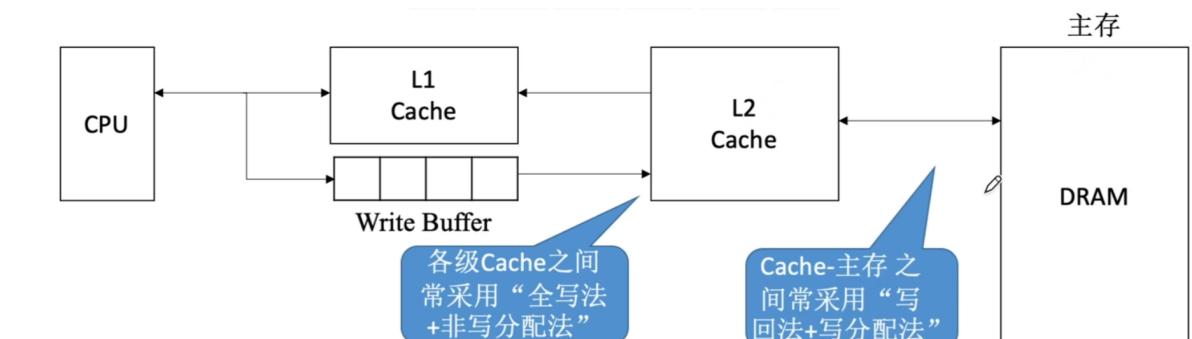
## 非写分配法



非写分配法(not-write-allocate)——当CPU对Cache写不命中时只写入主存，不调入Cache。搭配全写法使用。

全写法(写直通法, write-through)——当CPU对Cache写命中时，必须把数据同时写入Cache和主存，一般使用写缓冲(write buffer)

## 多级Cache



现代计算机常采用多级Cache  
离CPU越近的速度越快，容量越小  
离CPU越远的速度越慢，容量越大