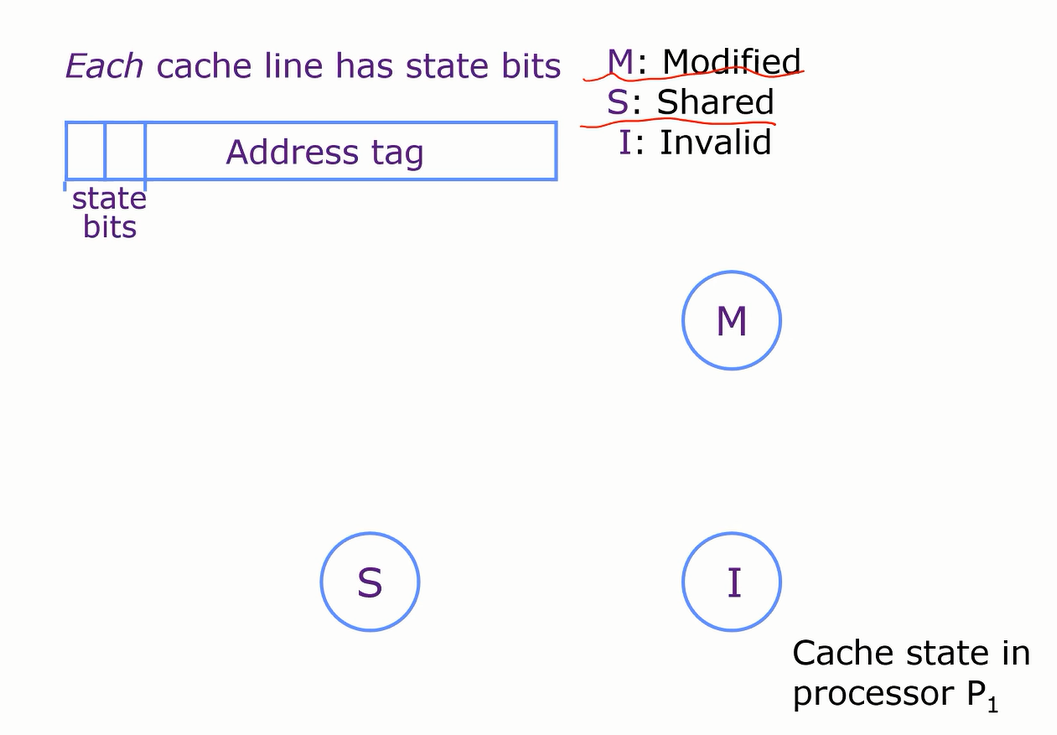
缓存一致性

基于监听的协议：每个核监听其他核是否更改自己私有Cache包含的数据，一旦发现更新就要无效这个数据或者更新。需要通过总线连接所有的核，并且只要有写回就要对所有核广播，很麻烦，消耗大。多核处理器刚出现时使用（核较少）

每一时刻只有一个核作为master，在总线上发送信息，其他核作为slave，监听总线的信息自己有没有，如果有就需要无效自己的数据并记录最新的数据在哪（或者直接做更改）



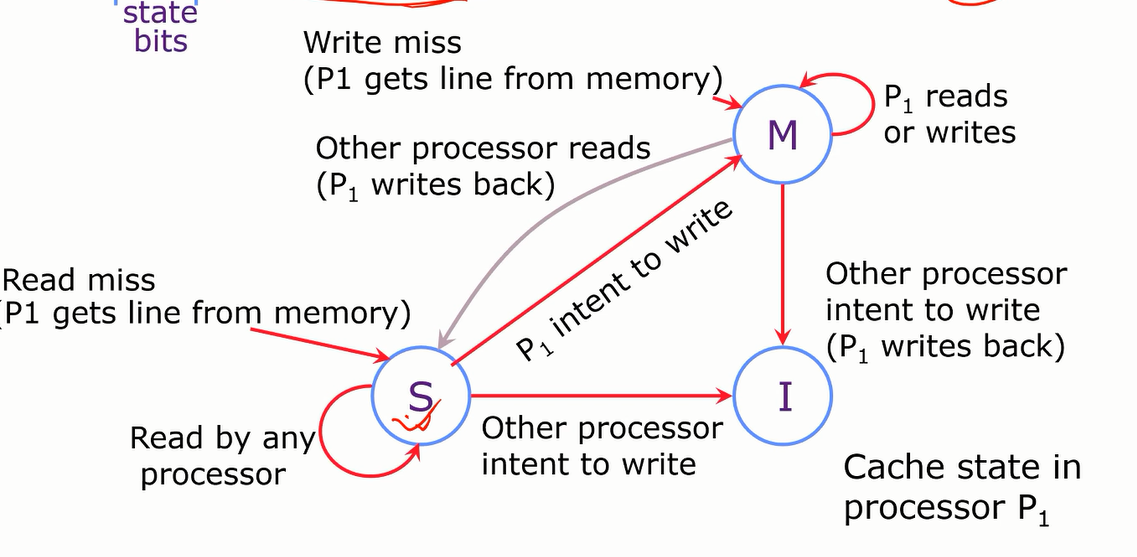
状态机，有3个状态

S:shared，在其他核中也有同样的备份

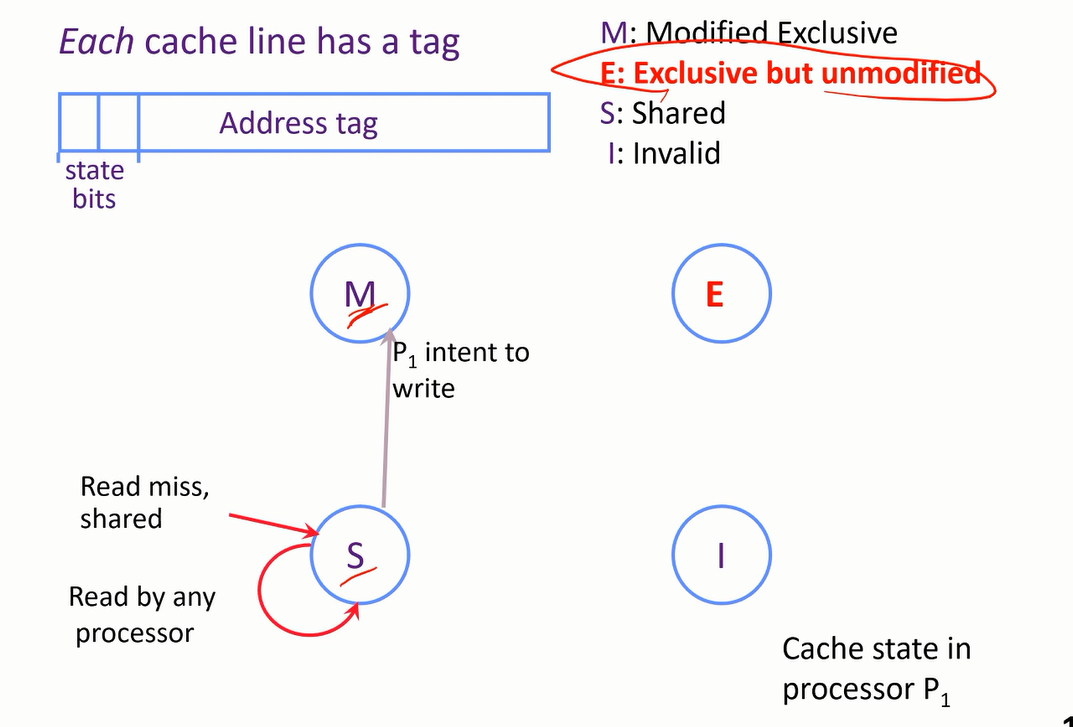
M:modified，自己核对数据进行了更改

I:invalid，其他核对这个数据进行了更改

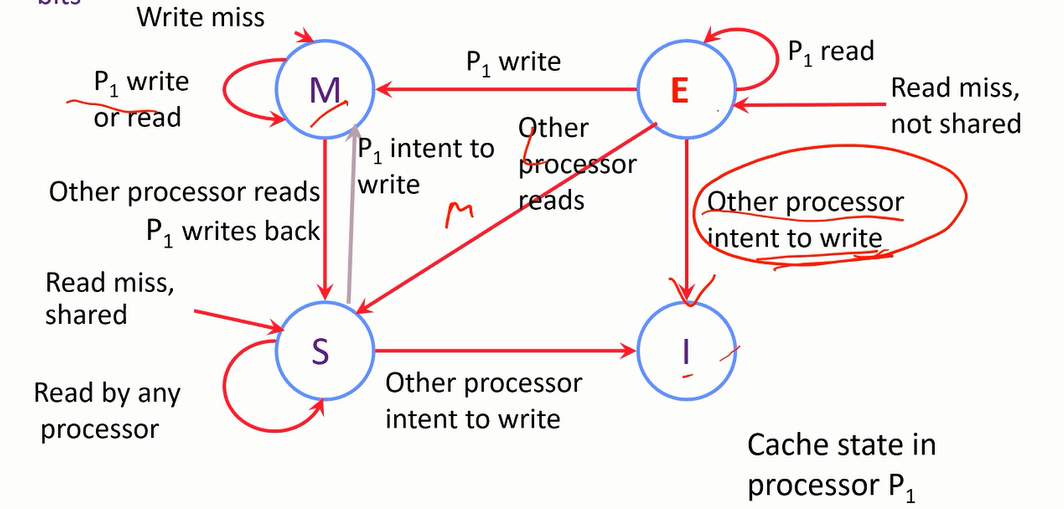
MSI



如果是独享某个数据就不需要广播了，所以引入E:exclusive but not modified

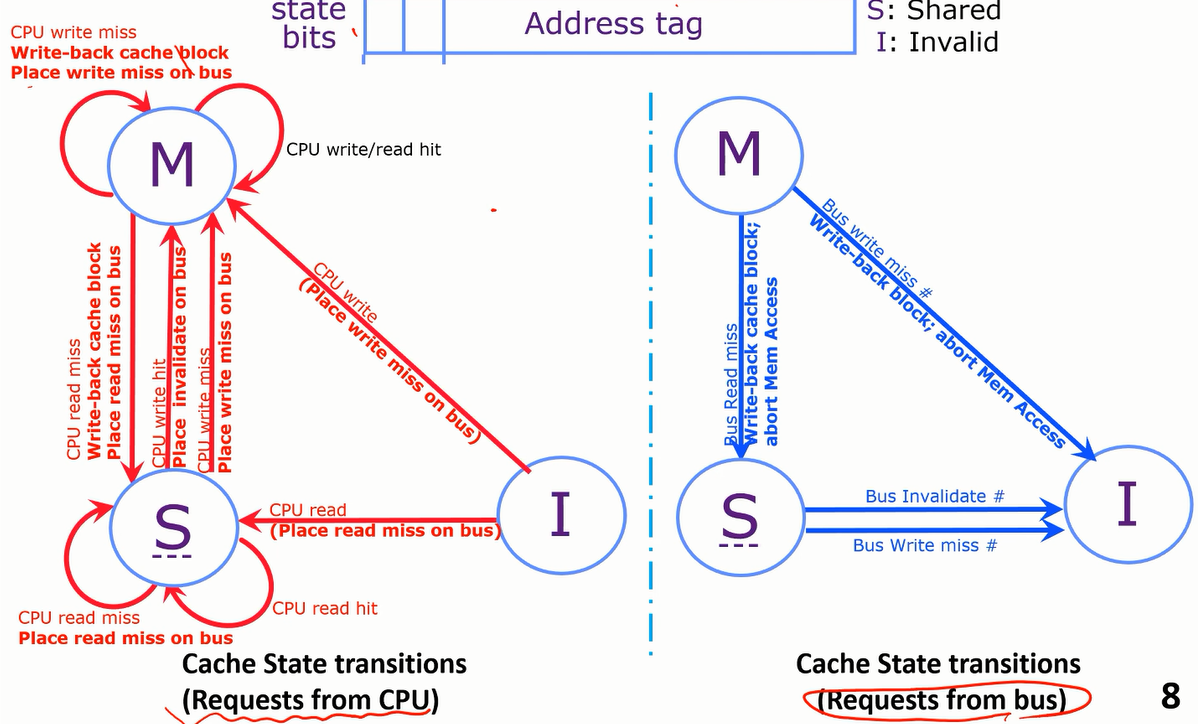


MESI

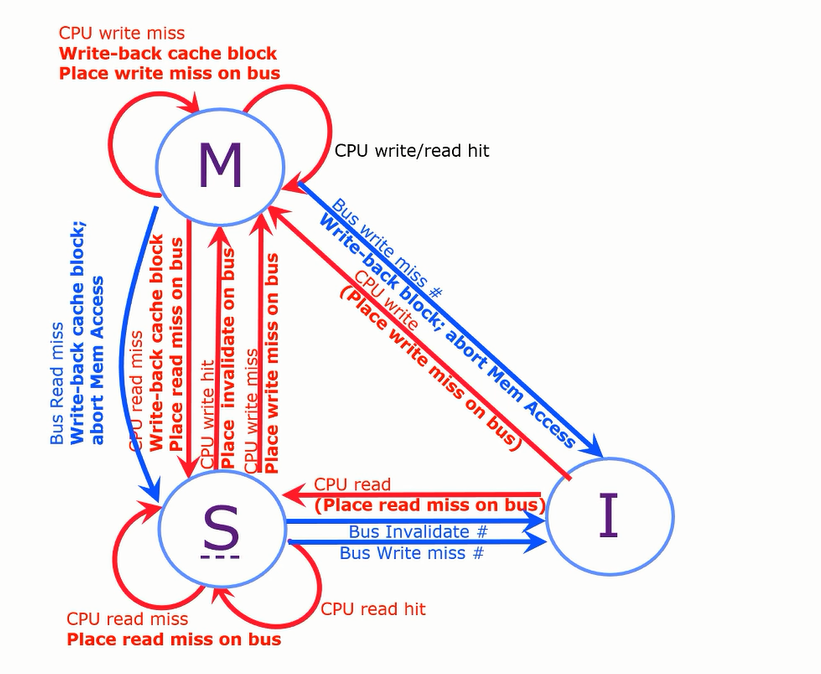


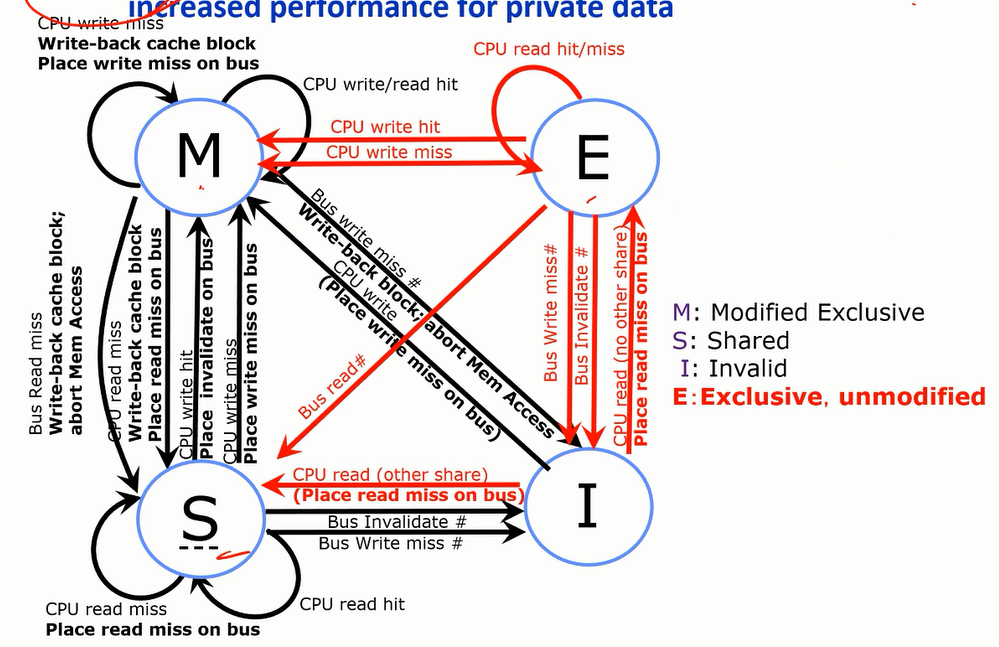
Cache line被替换写回(dirty=1)时modified状态就变成了shared

I状态再读取相同数据时会Cache miss就会进入S状态(或write miss到了M)



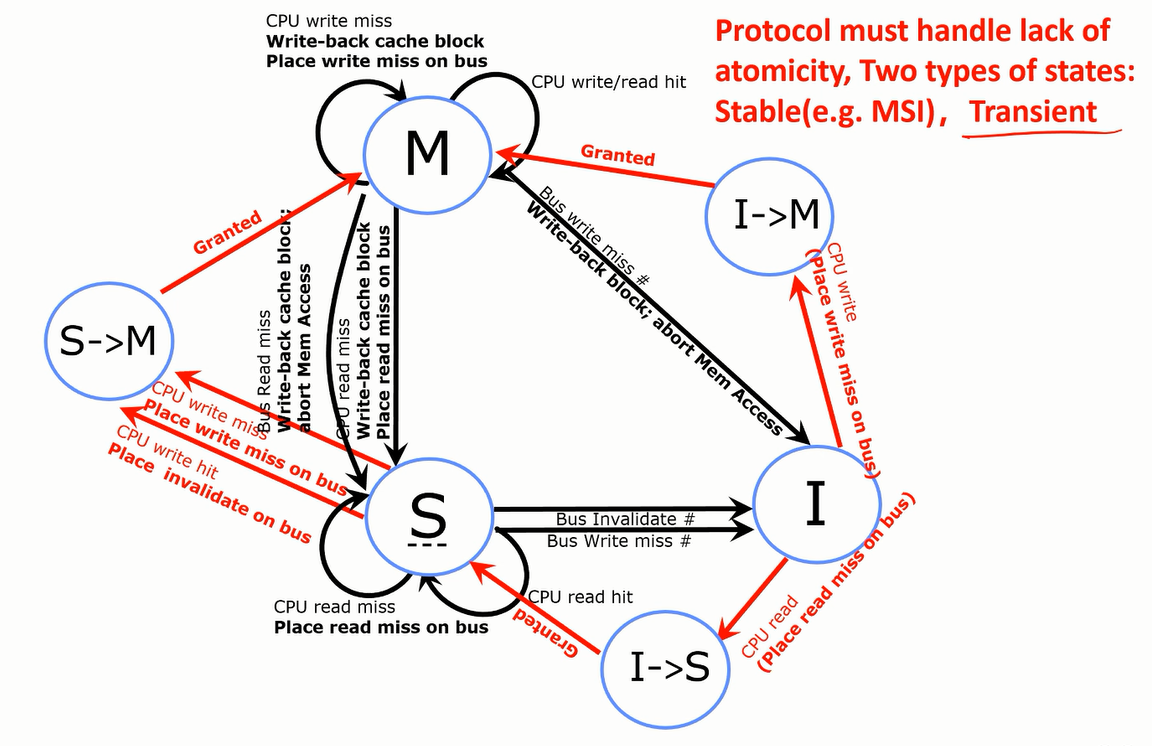
左图是当前核进行操作时相应的状态转换，右图是核监听到总线上的广播做出的状态转换。注意，当在M状态出现其他核cache miss时，不论read还是write都要先write back。read就不用说了，需要从主存中读取，write miss时也需要先读更新后的cache line，因为一个line有多个block，两个write不一定更改的是同一个block的数据，而cache的读写都是对cache line的





MESI 增加了exclusive以免在独享情况下还要广播，浪费bus带宽

但是有时候其实不需要频繁地写回memory，一个核在store一个核在load，两个核都是share，此时load的核被踢出，这并不需要写回memory，因为store的那个核才是生产者。所以加入一个O:owner，表示是核的store状态 延伸出MOSI MOESI



当核很多时挂载总线将会是很大的开销，这就需要进行片上网络的设计

基于字典(目录)的协议：

精确记录哪些核具有这个数据的备份，点对点通信

5个状态（记录在字典中）：

U:所有核都没有这个数据

R:多个核共享

W:一个核独享

TR:To R过渡状态

TW:To W过渡状态（并不是马上更改，是一个过程）

Cache本身也要记录MSI 视频42-43 1:50:00左右

避免不同核同时申请时的竞争，如Core0和Core1都store 0xA，需要在目录里加一个队列，先响应一个再响应另一个，后响应的那个申请时不被批准，状态变成I，响应之后再做状态的改变。避免同时读写的混乱。

3-HOP：一个核发生miss，到目录中查询最新数据在哪个核(1 hop)，目录对这个核发起访问，告诉核哪个核需要这个数据(2 hop)，随后核将数据发送给目标核(3 hop)

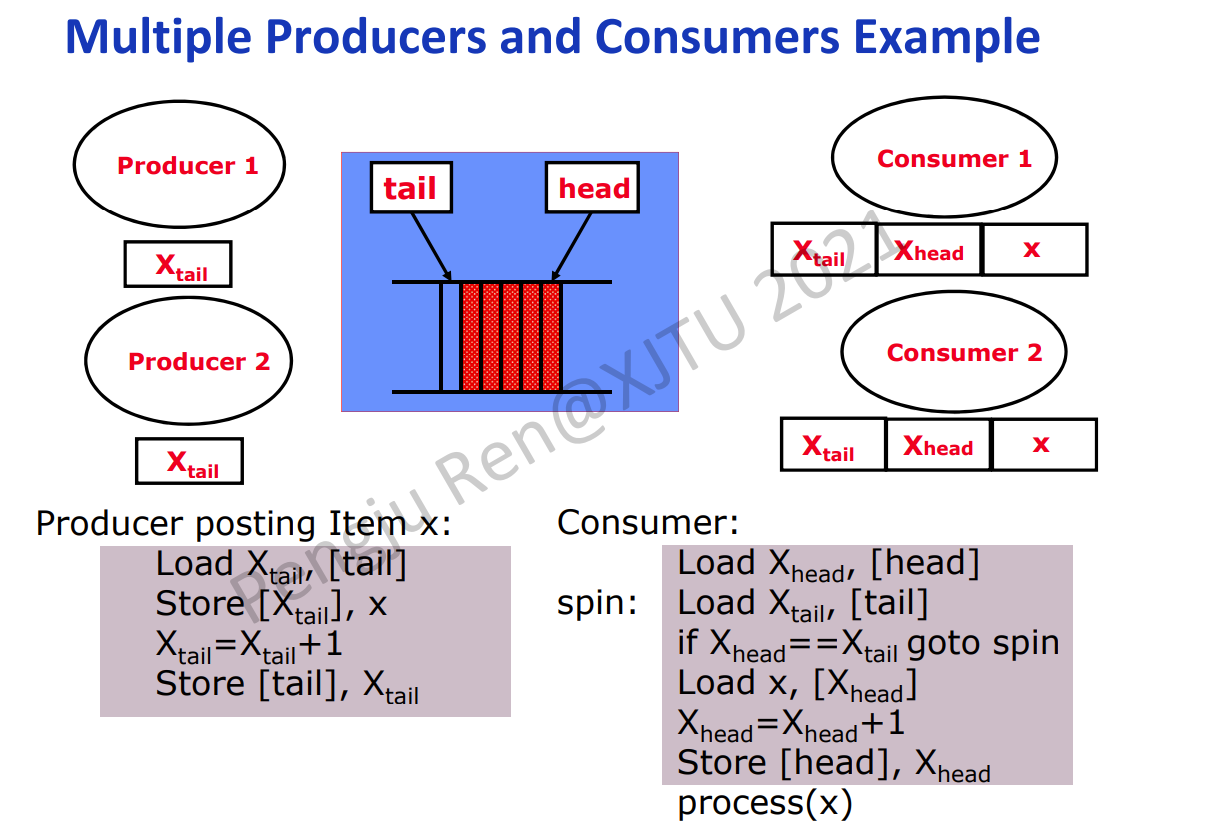
可以给所有核一个shared cache来存储目录，加快对目录的访问速度

芯片间、处理器间也需要保持缓存一致性，需要灵活选用两个协议

同步性：保证核之间信息交换的正确性

生产者-消费者模型

资源共享互斥模型



1 store [X\_tail],x //将数据写入尾指针

2 store [tail],X\_tail //尾指针后移1

3 Load X\_tail,[tail] //读取尾指针 之后与头指针比较

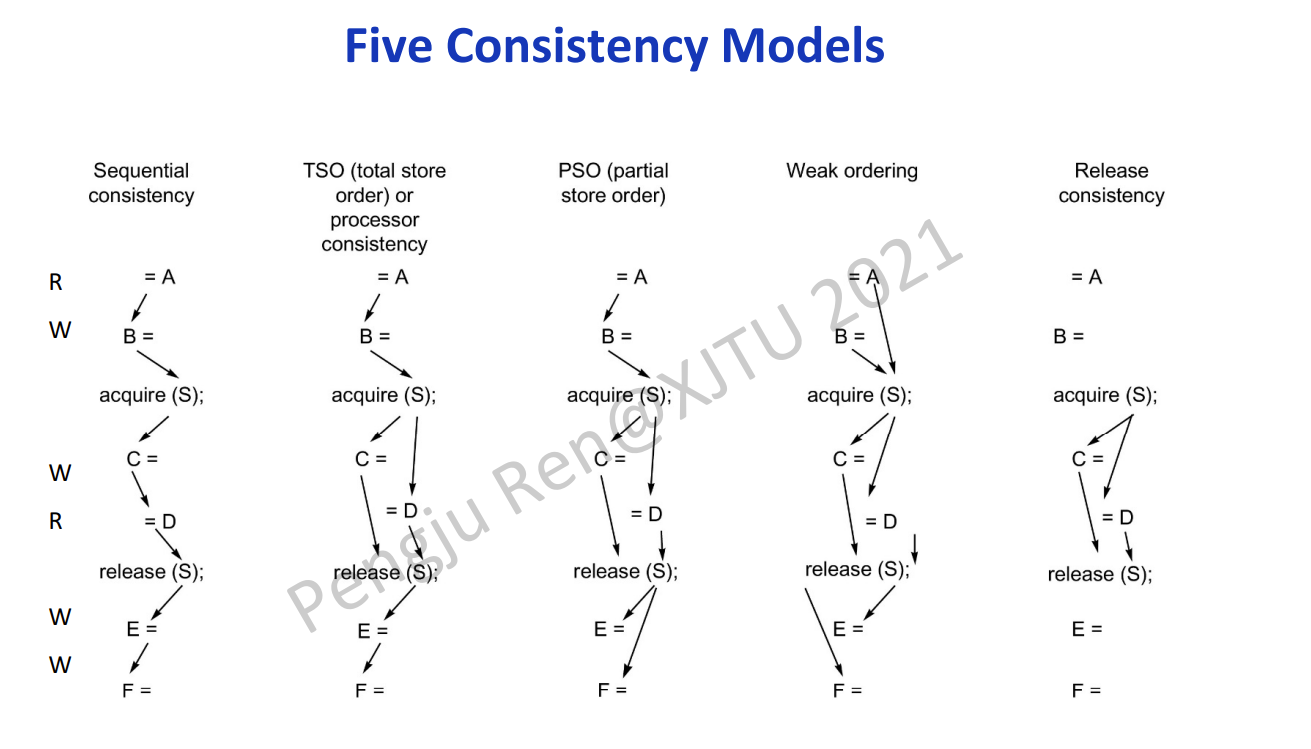
4 store [head],X\_head //头指针后移1

需要保证一定的顺序，但指令乱序且线程交织，指令的顺序不可知

-需要保证1,2;3,4为一组，不可中断分割，1在2前，3在4前。1,2一组称为原子操作，虽然是2条指令但不可分割

严格地保证指令存取的顺序一致性和不会被别的线程打断是很困难的，因此只保证关键步骤的一致，即对公共变量的维护过程的一致

5种一致性(从左到右依次减弱)



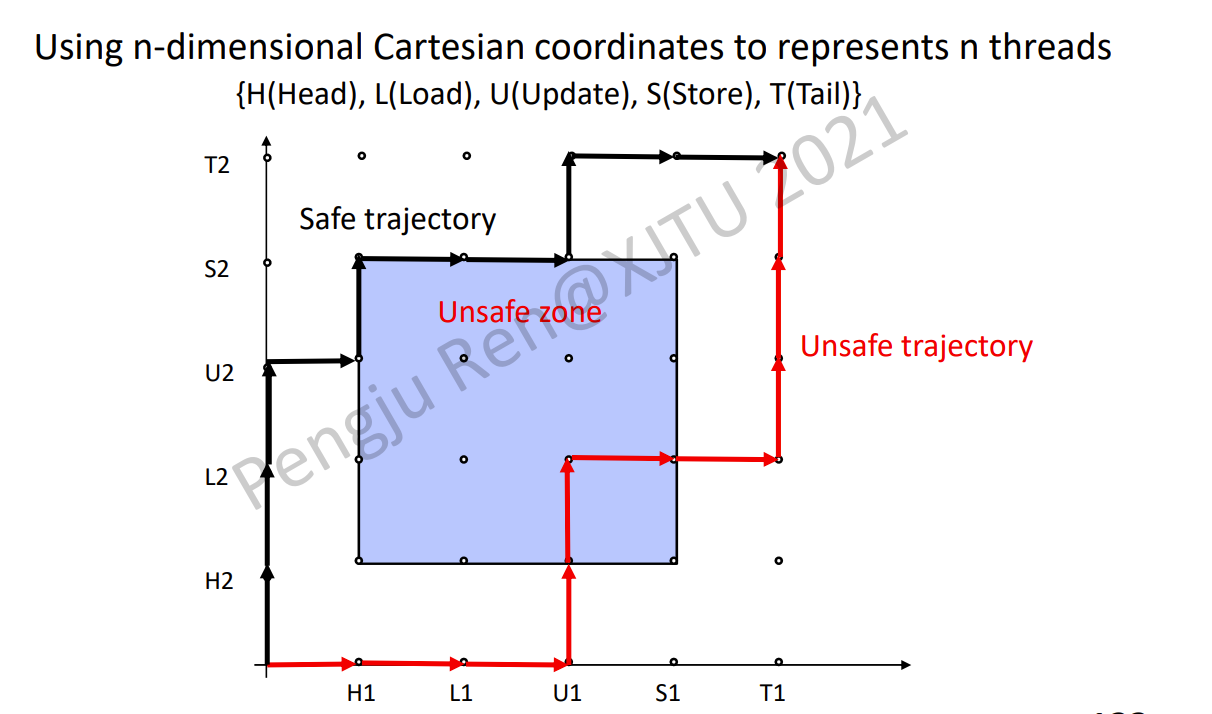
R:load W:store

线程内：采用Fence指令（包括Fence\_wr,Fence\_ww,Fence\_rr,Fence\_rw，即读/写之间Fence）隔开需要保证顺序的存取指令，只有当Fence前的指令执行完毕并提交才会开始后面指令的取指令

线程之间：lr:load-link sc:store-conditional

-Load-Link和Store-Conditional为一组，中间包括的所有指令如果被其他线程指令插入而打断，Store-Conditional就不会执行，并返回到Load-Link重新执行，这样从逻辑上就表现出这部分指令没有被打断

指令调度



L-U-S是一个原子操作(读取公共变量，更新后保存回去)，不允许交织其他线程的这一操作，所以不允许调度线进入不安全区（如红线，执行完U1，没执行S1，就去执行了L2）调度时需要保证不会穿插进不安全区内，如黑线所示。