缓存一致性协议(MESI)理解

原创 专注写bug 于 2021-08-26 18:06:14 发布 阅读量965 收藏 10 点赞数 3 版权

分类专栏： 并发编程 文章标签： java

文章目录

前言

在之前博客中，说到 多线程 共同 写操作 更改 同一个共享变量 时，会导致数据运算的不正确性。

为了解决这个问题，分别有了两种不同的解决策略。

- 总线锁。
- 缓存一致性协议

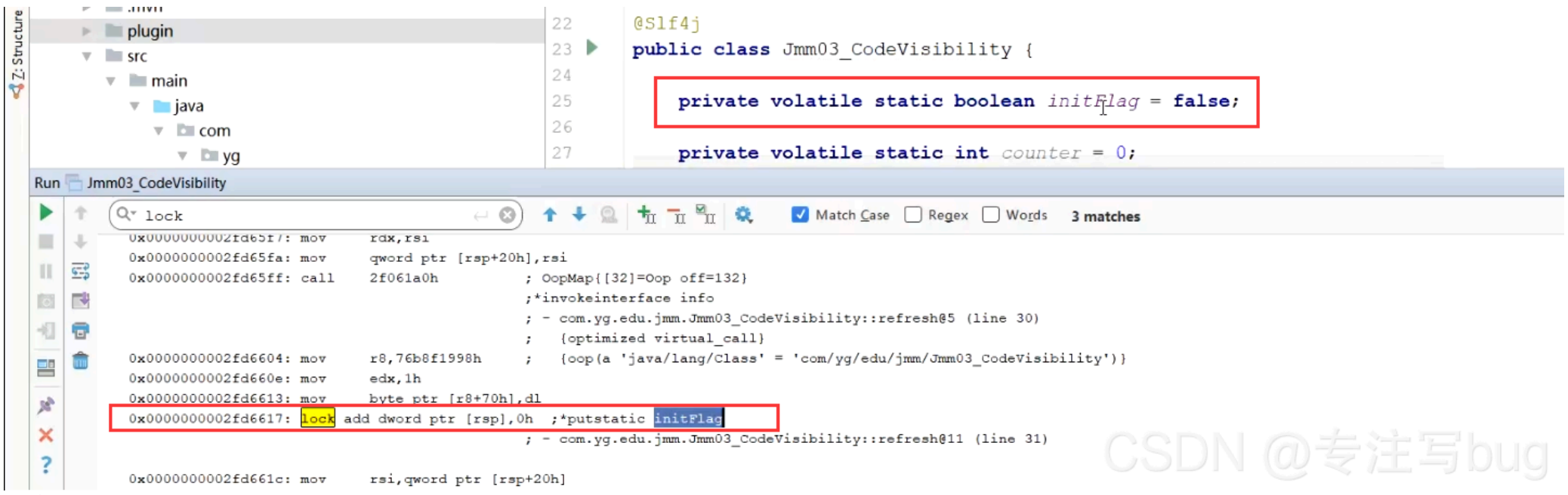
> 汇编中的 lock 指令，会触发 硬件缓存锁定机制 。
> 即：总线锁、 缓存一致性 协议

总线锁

早期为了解决这种情况下，数据出现并发问题。提出了 总线锁 的说法。

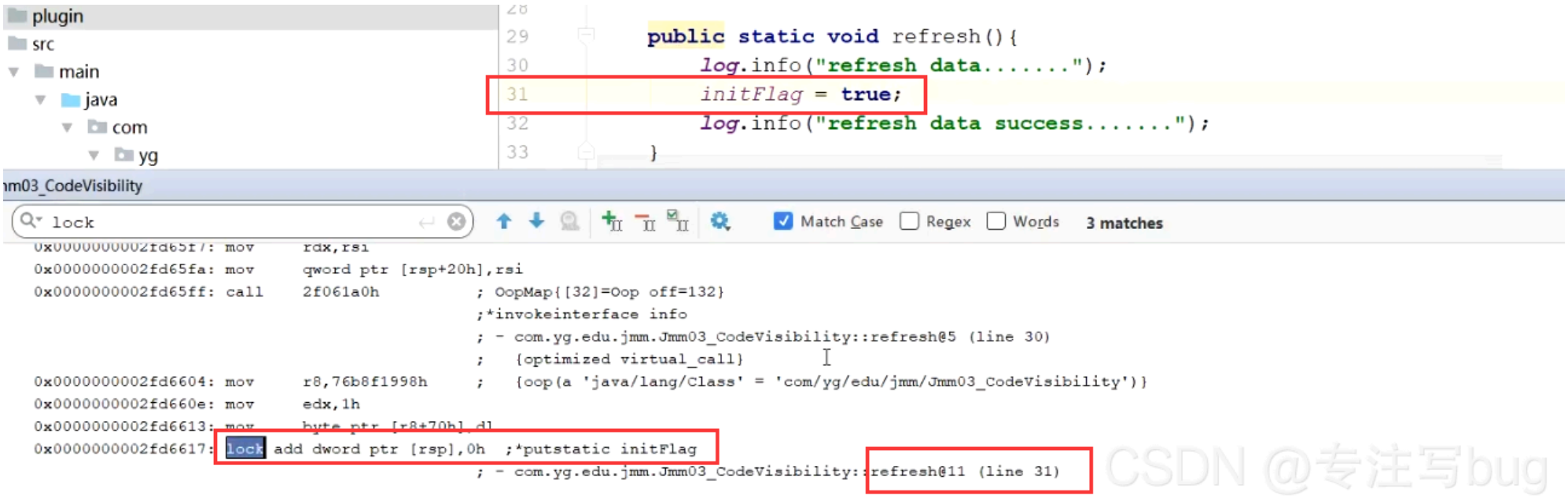
比如查看被 volatile 修饰的共享变量的 汇编指令 ， 如下所示：

> 查看Java代码的 汇编指令 ， 参考文章：查看Java代码执行的汇编语言



【发现：】

在对 volatile 修饰的 共享变量 做变更操作时，进行增加 lock 前缀。



查看 IA-32架构开发手册 ， 得到下列信息：

指令	指令描述	对应用程序有用?	禁止应用程序使用?
LOCK(前缀)	总线锁	是	否

这里的 总线锁 是什么呢?

在 IA-32架构开发手册 中对 lock总线锁 有以下描述：

在修改内存操作时，使用 LOCK 前缀去调用加锁的读-修改-写操作(原子的)。这种机制用于多处理器系统中处理器之间进行可靠的通讯，具体描述如下：

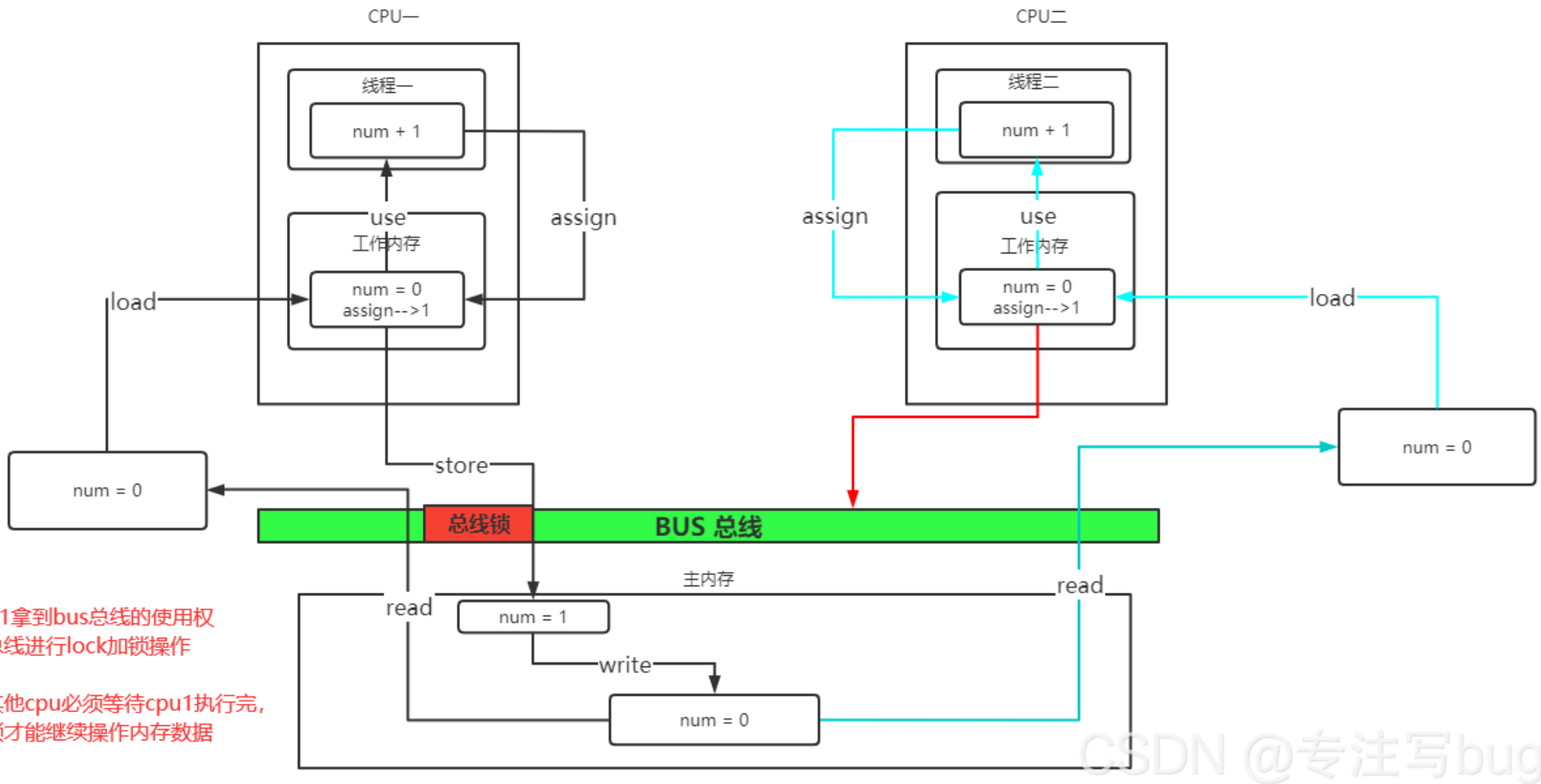
在 Pentium 和早期的 IA-32 处理器中，LOCK 前缀会使处理器执行当前指令时产生一个 LOCK#信号，这总是引起显式总线锁定出现。

在 Pentium 4、Intel Xeon 和 P6 系列处理器中，加锁操作是由高速缓存锁或总线锁来处理。如果内存访问有高速缓存且只影响一个单独的高速缓存线，那么操作中就会调用高速缓存锁，而系统总线和系统内存中的实际内存区域不会被锁定。同时，这条总线上的其它 Pentium 4、Intel Xeon 或者 P6 系列处理器就回写所有的已修改数据并使它们的高速缓存失效，以保证系统内存的一致性。如果内存访问没有高速缓存且/或它跨越了高速缓存线的边界，那么这个处理器就会产生 LOCK#信号，并在锁定操作期间不会响应总线控制请求。

RSM 指令(从 SMM 返回)恢复处理器(从一个场境堆中)到系统管理态(SMM)中断之前的状态。

这说的是什么意思呢？下面采取图形解释：

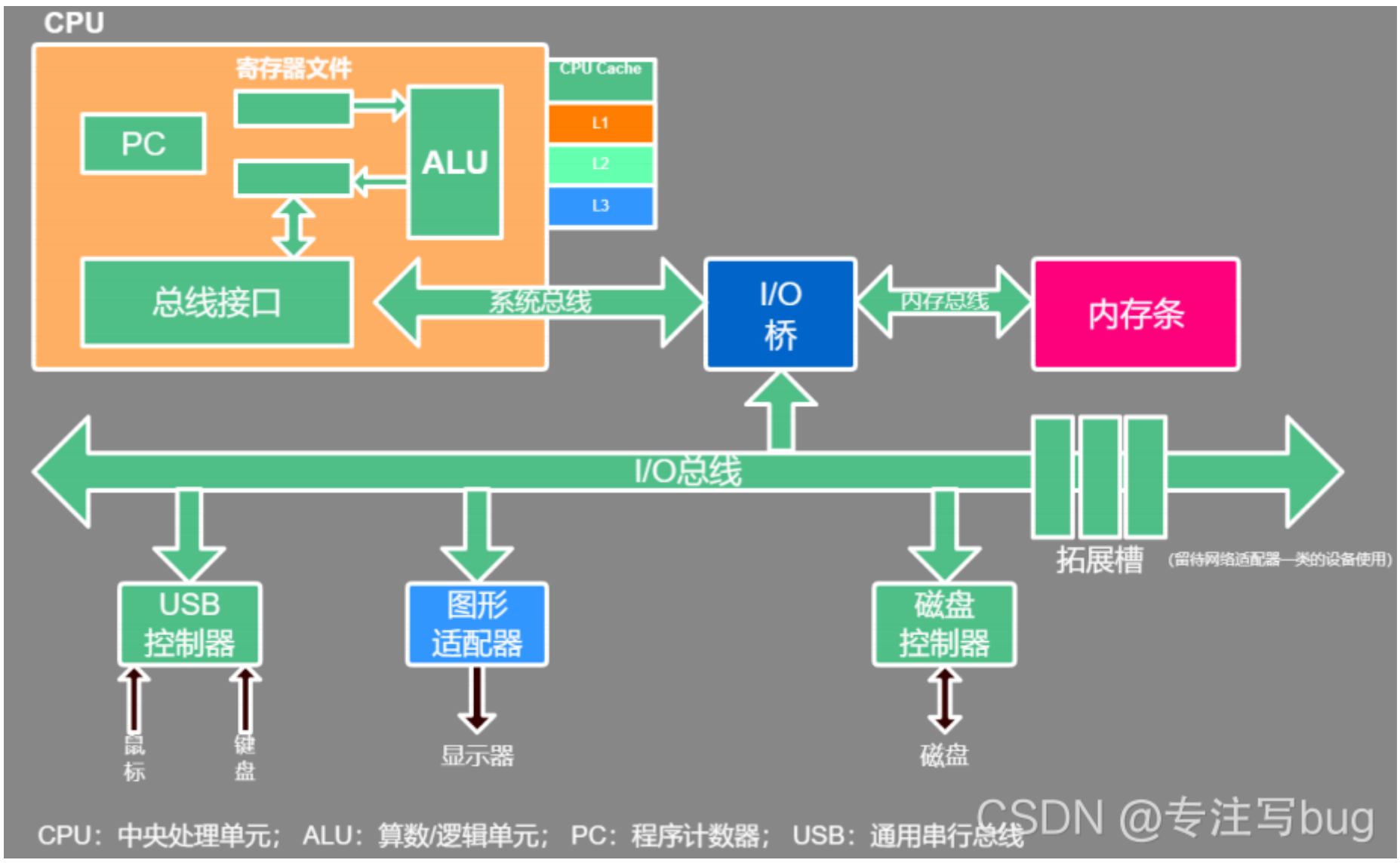
> 早起使用 总线锁 保证缓存数据的一致性。



> 此时，cpu拿到 总线锁 后，相对其他cpu而言，整体就是一个 单核设备 。
> 效率会很低!

这里的 总线 是什么意思呢?

众所周知，在计算机中，CPU和内存条并未集成在一块，他们之间通过一条 系统总线 的渠道进行数据交互。



> 其中， 系统总线 和 内存总线 就是所谓的 BUS总线 。

【注意：】当然这种 总线锁 是以前的处理操作，现在的程序执行，当存在汇编指令 lock 标识，会使用 缓存一致性协议 定义的方式去解决问题。

缓存一致性协议

为了解决上面 lock总线锁 ， 带来的 当cpu抢占成功总线时，其他cpu只能等待其执行结束才能继续抢占 ， 这种 互斥问题 。提出了 缓存一致性协议 的解决方案。

缓存一致性协议 多种多样(MESI、MSI等)，其中最常用的就是 MESI缓存一致性协议 。

接下来重点说明什么是 MESI 。

MESI 并不是指一个协议名称的缩写，而是对应几种状态类型，他们的各个状态信息含义如下所示：

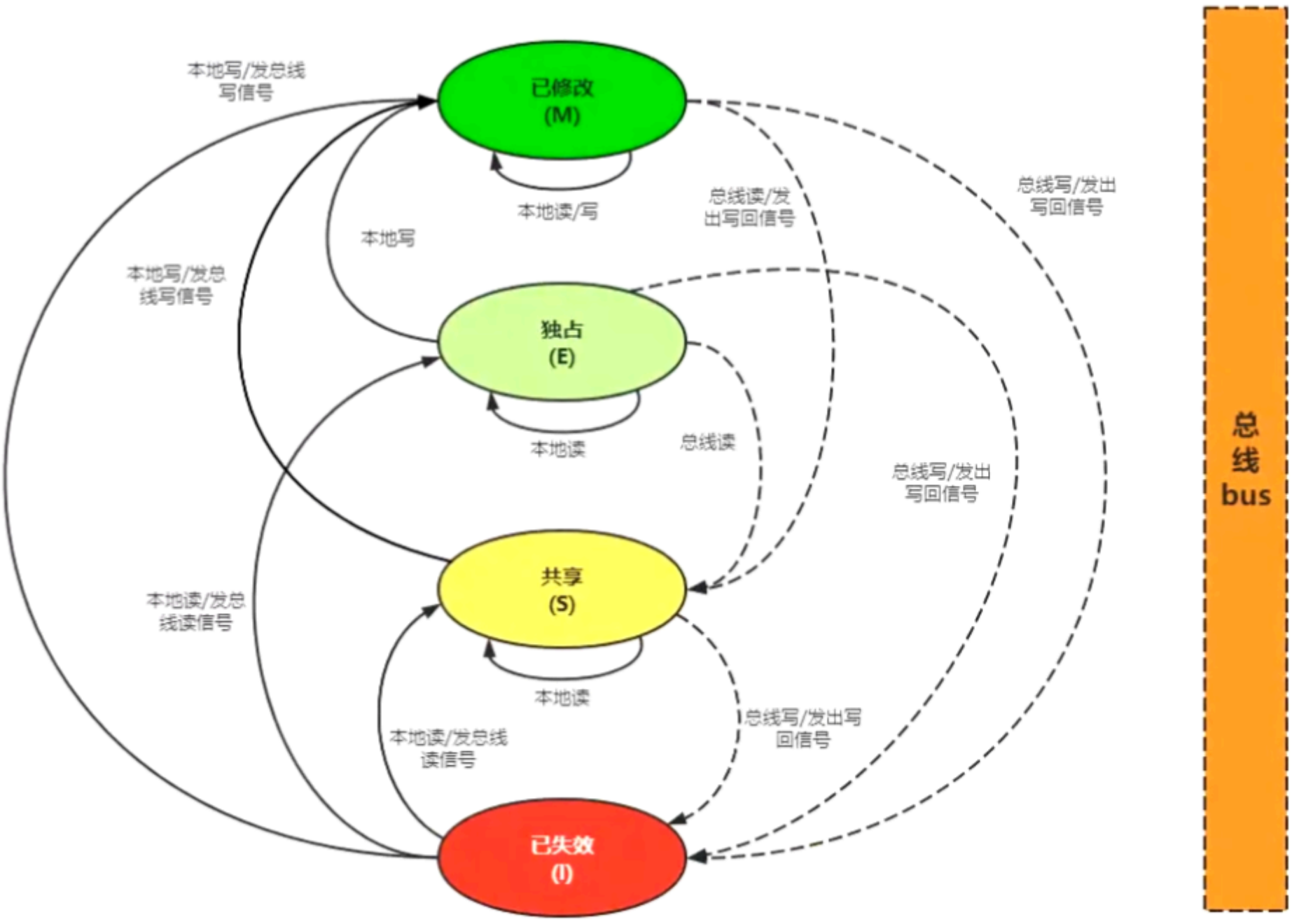
状态	描述	监听任务
M 修改 Modified	该Cache Line有效， 数据被修改。 和内存中的数据不一致。 数据只存在于本Cache中。	缓存行必须时刻监听 所有 试图 读 该缓存行相对就主内存的操作。 这种操作必须在缓存将该 缓存行 写回 主存， 并将状态更改为 S(共享)状态 之前 被 延迟执行 。
E 独享、互斥 Exclusive	该 Cache Line 有效， 数据和内存中的数据一致 数据只存在于本 Cache 中。	缓存行必须监听 其他 缓存 读 主存中该缓存行的操作。 一旦存在这种操作，该缓存行需要更改为 S(共享)状态 。
S 共享 Shared	该 Cache Line 有效， 数据和内存中的数据一致， 数据存在于 很多 Cache 中。	缓存行必须监听 其他 缓存 使该缓存行无效或着独享该缓存行的请求， 并将该缓存行变更为 I （无效）状态 。
I 无效 Invalid	该 Cache Line 无效	

【疑问】什么是 缓存行？

> 缓存行 就是 缓存存储数据的单元 。64字节。

> MESI 是指4种状态的首字母。每个Cache line有4个状态，可用2个bit表示

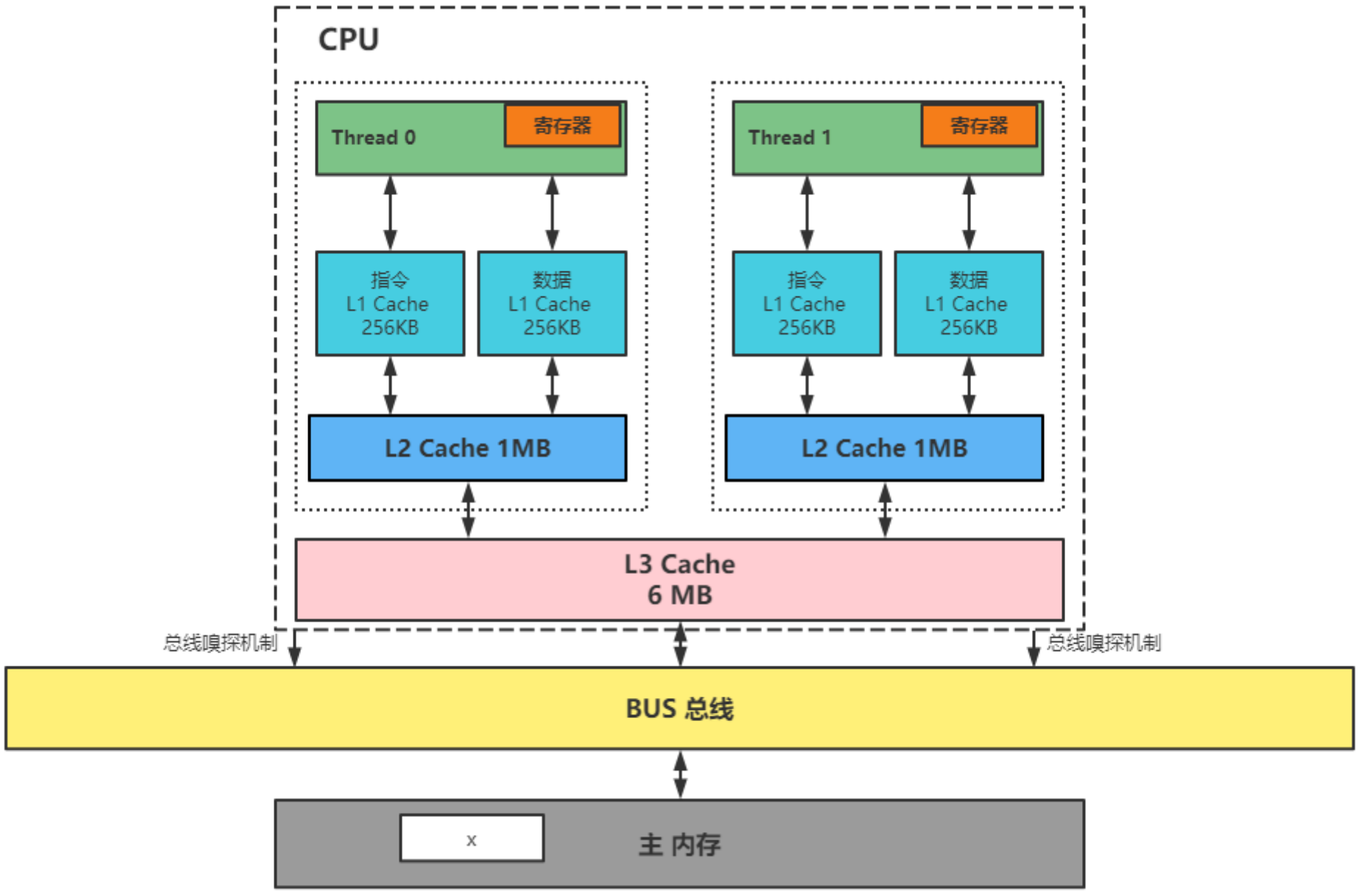
MESI 四种工作状态逻辑图如下所示：



初次看上面的逻辑图，觉得很烧脑，接下来采取案例进行解释和说明。

当一个共享变量被 volatile 修饰时，此时多个线程去对其执行操作，其中数据在缓存中的状态变更如下：

> 假设此时主内存中，存在 volatile 修饰的变量，其名称为 X 。

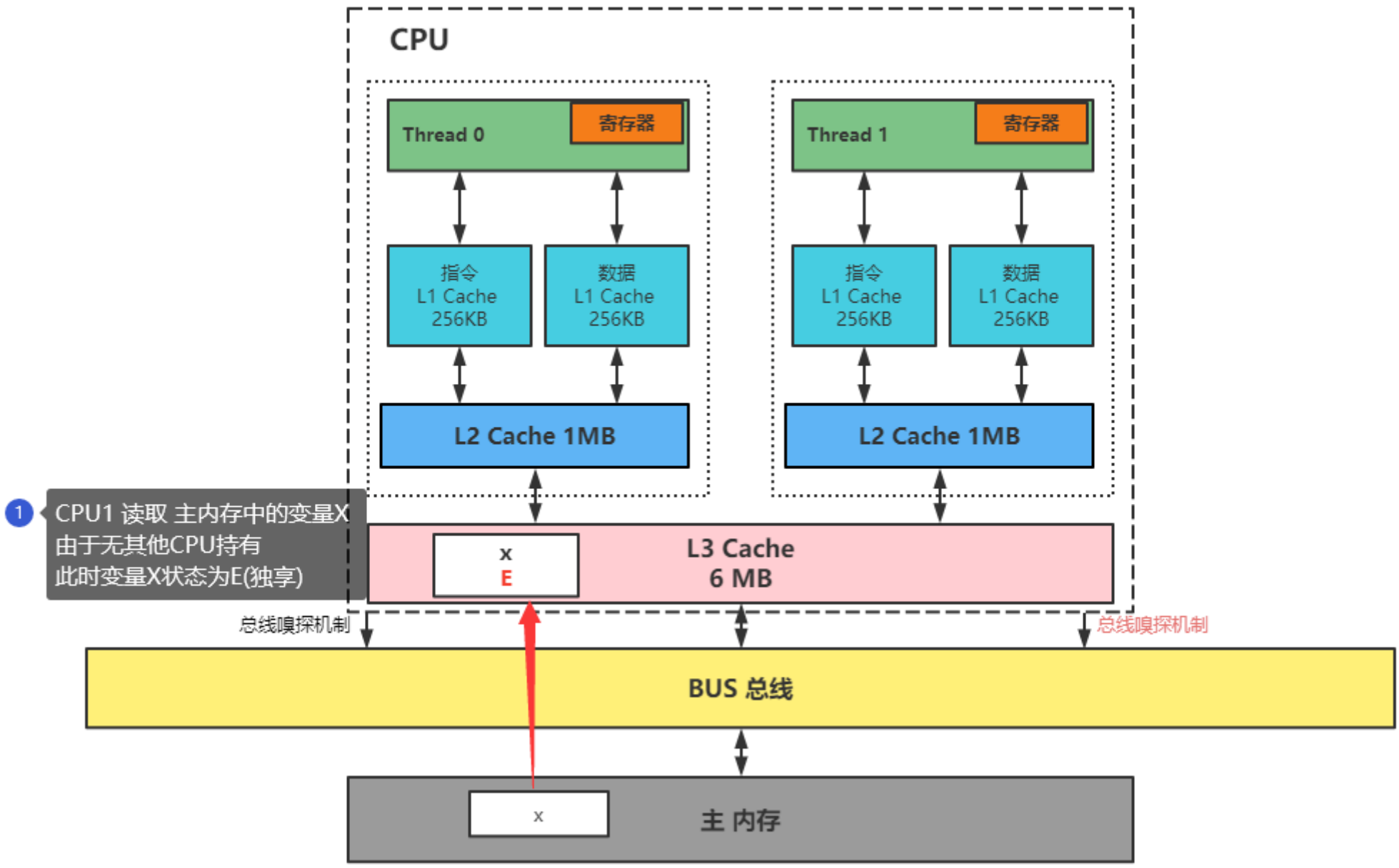


此时 CPU1 开始从 主内存 中加载 共享变量 X 的信息。

> 变量 X 在汇编指令中，存在 lock 修饰。 CPU1 对其进行读取加载操作时(数据经过BUS总线)，会被 其他CPU 进行 监听 。

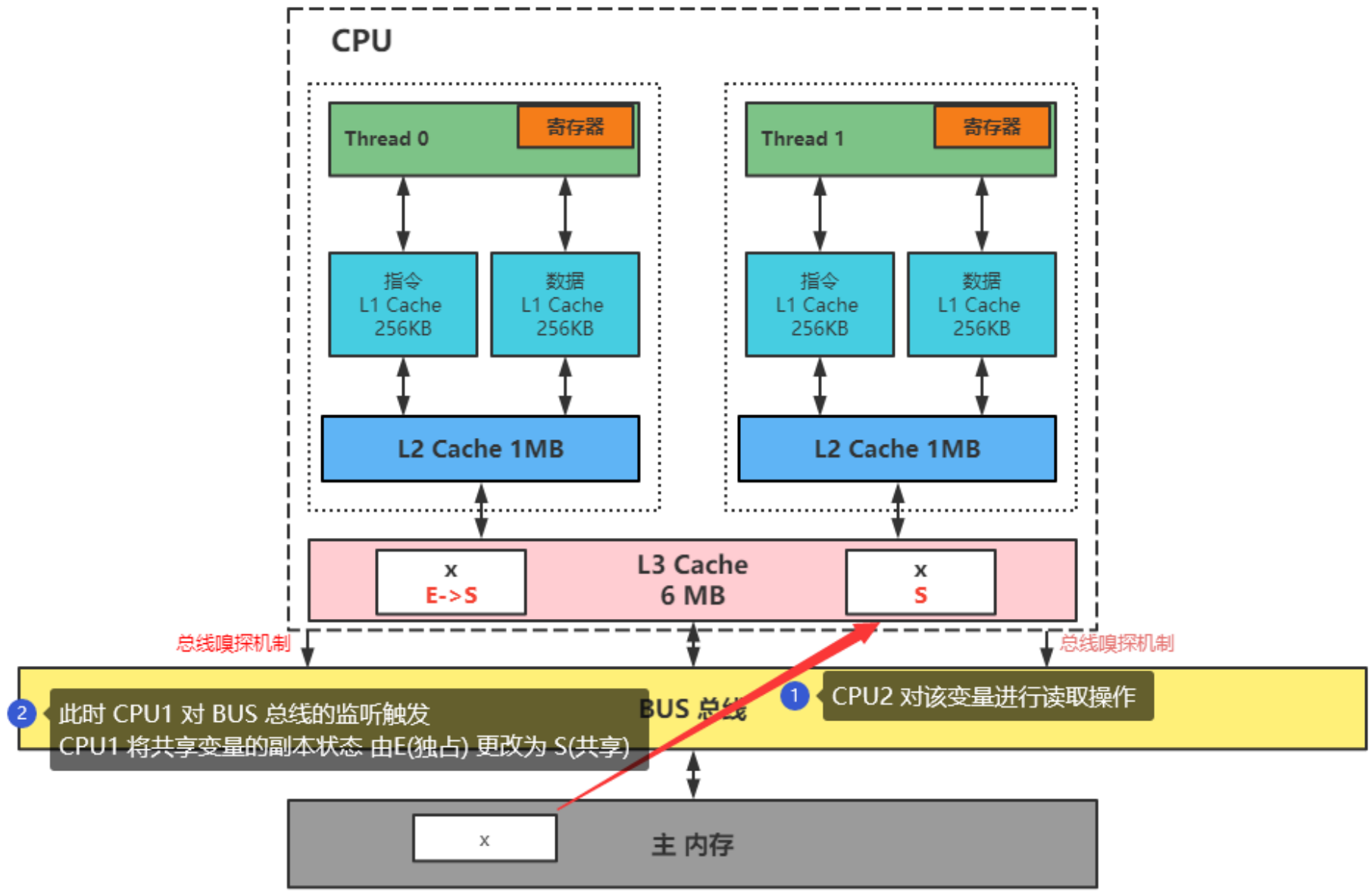
当 CPU1 读取 X 变量 至 高速缓存 中后，此时的变量 只被 CPU1 持有 ， 其他CPU 未持有 。

> 这个时候，会对该变量 X 在 CPU1中的副本 ， 标记一个 状态 E（独占） 。



假设此时 CPU2 也对该变量 X 进行读取操作。

> 此时 变量 X 经过 BUS总线，由于是 CPU2 读取变量操作，那么就会被 CPU1 对 BUS总线 监听到。
> CPU1 监听到其他CPU 读取 到指定的变量 X，那么就会将该变量的状态信息进行更改操作。
> 同时 CPU2 也会将该变量 X 的副本状态信息写为 S 共享状态 。



> S 状态：表示当前变量存在 多个副本，分别属于 不同 CPU 操作。
> 一旦需要对该变量副本进行修改时，此时的副本状态为 S，表示 不可直接修改！

【疑问：】为什么 S状态下，不可修改？

> 其他CPU也有该变量副本，必须维护多个CPU副本数据保持一致！

上面的操作还仅仅停留在 读操作层面，如果 多个CPU 需要对数据进行 写操作 呢？

> 将设 CPU1 需要将变量信息变更为 x = 1；但 CPU2 需要将变量信息变更为 x = 2 。

如果此时如果都能执行成功，都能写回 主内存 中，将会导致 主内存中变量信息的不可控 。

> 如果依据条件是 CPU对变量执行的速度，那么就会出现 哪个CPU执行速度快，主内存中的数据就是什么 的现象。

【疑问：】如果多个 CPU 对该变量进行操作，此时如何防止主内存中的数据信息不可控？

> 各个 CPU 会去竞争 对缓存行的加锁 操作。
> 谁先获取到锁，谁就去操作。然后其他CPU不可操作！

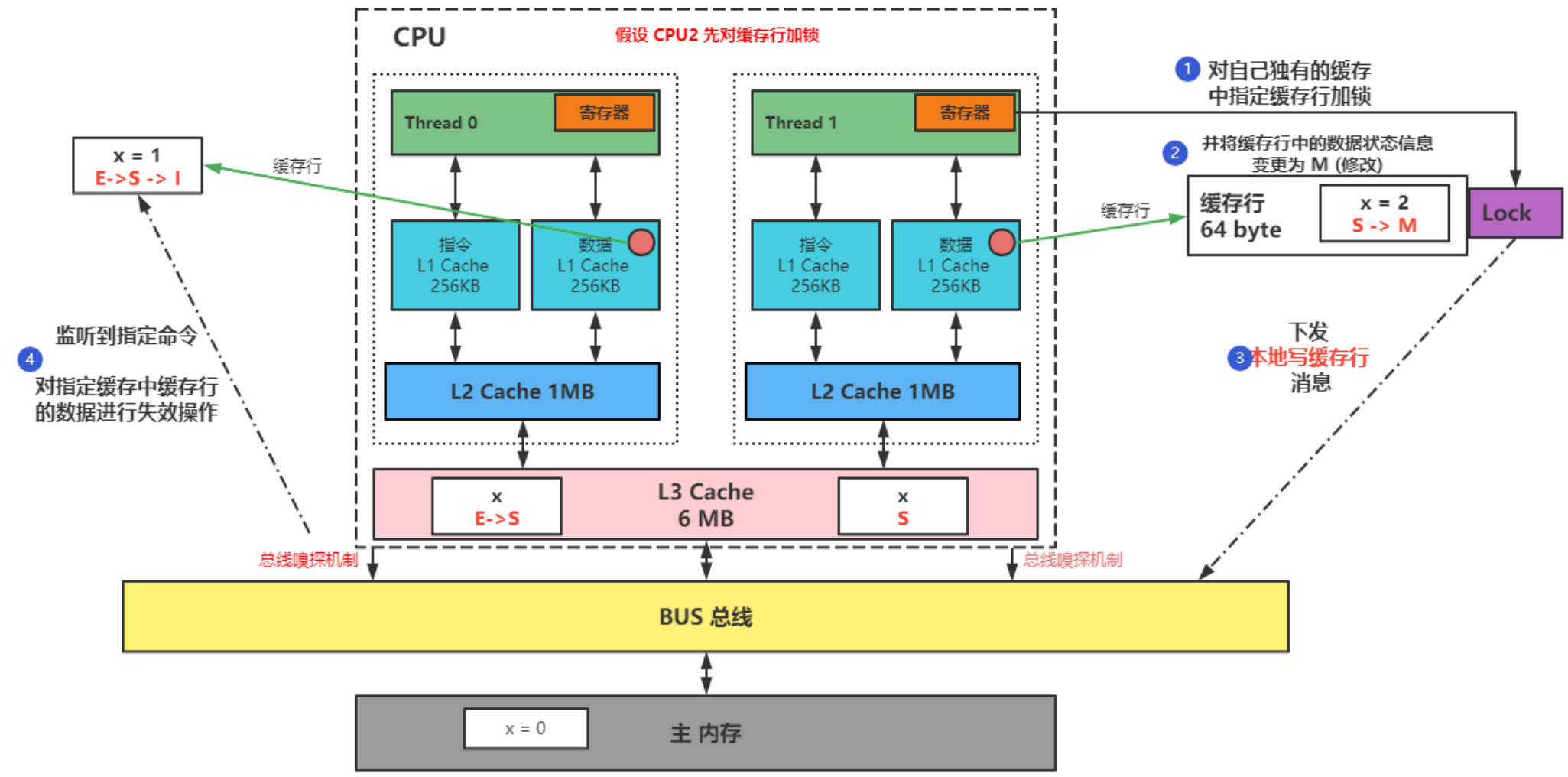
【注意：】但是， L1、L2 是每个CPU各自独占的！ 都是属于 各个CPU私有 ！

> 此时针对 L1 、 L2 高速缓存中的 缓存行(64byte) 加锁是无效的！

【疑问：】那如何保证多个 CPU 对数据操作的正确性呢？

> 依旧是在本地高速缓存中 对缓存行 进行 加锁操作 。
> 并且，向外发送一个 本地写缓存行的消息， 通知其他CPU 将其数据信息进行 失效操作 I 。

具体实现如下图所示：



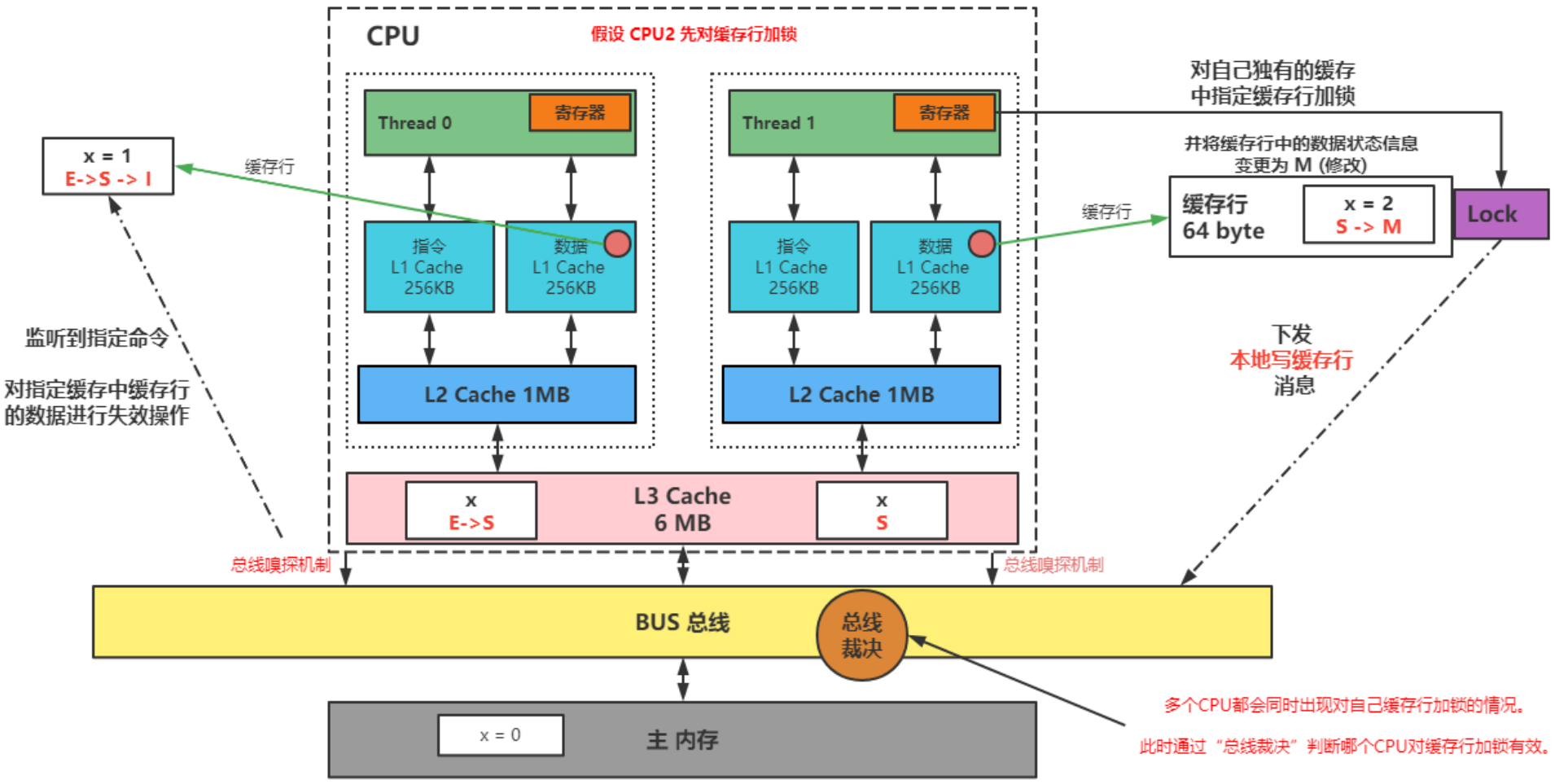
【联想：】但是否会存在 不同CPU 同时给各自缓存行加锁的现象？

> 当CPU对自己的高速缓存中的指定缓存行进行加锁操作时，由于高速缓存属于 各自CPU 独有，其他CPU不可感知，
> 所以 会存在所说的问题 ！

【疑问：】出现这类现象，如何保证让哪个CPU去执行加锁操作呢？

> 不管是哪个CPU对数据进行操作，都需要经过 BUS 总线 这个 通用 的路径。

> 计算机通过 总线裁决 进行判断哪个CPU对缓存行加锁有效！



【疑问：】总线裁决 是如何实现判断的？

> 每个CPU都会有时钟周期数据， 总线裁决 通过 BUS 总线 上具体CPU的时钟信息进行判断！

当数据失效后，让CPU重新从 主内存 中获取 指定的变量信息 。

> 同时也会去 进行缓存行加锁操作 。
> 当进行加锁操作时，由于其他CPU如果还在对其做数据修改操作，则读取的数据依旧会使其失效！

CPU修改 `volatile` 修饰的 共享变量 ， 是通过 对自己的缓存行加锁 操作进行实现的。

> 但是一个缓存行的大小只有64byte。

【疑问：】当出现一个缓存行的大小存放不了一个数据时，此时又该如何保证数据的可见性呢？

> 将 缓存一致性协议 进行 升级操作 ， 更换为 总线锁 ！

注意点

当数据很大时，导致一个 缓存行(64byte) 存放不了时，此时数据就会 跨越多个缓存行 进行 存储 。

volatile修饰的变量信息，不同CPU从主内存加载至高速缓存，都会对该数据所在缓存行加锁，像L1L2都是每个CPU独有的。但因为其他CPU还在对该变量进行修改操作，每次修改之后，他都会向bus总线发送一个通知，其他CPU对总线是存在总线嗅探机制，当有对应的消息监听到，也会让指定CPU将读取到的数据继续失效重读

差不多是只有一个cpu对缓存行加锁成功的意思，在它完成修改前，其他cpu数据失效后即时重新再读取也是失效的（否则不就读到旧数据了），这就保证了其他cpu只能拿到最新的数据，即修改后的数据。

> 如果跨越 多个缓存行 ， 采取 缓存一致性协议 对缓存行进行 加锁操作 ， 此时 加锁操作可能失败！！！

原因：
假设此时有A、B两个缓存行。当 CPU1 对A缓存行进行加锁操作后，需要对B缓存行进行加锁，此时可能在其他CPU中，已经对B缓存行进行了加锁操作。导致 CPU1 对跨 多个缓存行加锁失败!

> 缓存行是一次性获取多个相邻数据，其他变量也有volatile修饰，对其他变量进行了加锁；
> 此时该缓存行中正好存在当前变量。(受其他volatile变量加锁影响)

> 单个缓存行具有原子性。
> 多个缓存行的数据在多线程环境下执行时，可能会被中断!

解决：

> 将缓存一致性协议进行升级处理，更换为 总线锁 。