考虑一个系统包含 4 个字节寻址的处理器。每个处理器有一个私有的 L1 Cache，大小为 256 bytes，采用直接映射（direct-mapped）和写回策略（write-back），block size 是 64 bytes。该系统采用 MESI 协议保证 cache coherence。
内存访问的地址范围是 0x50000000 - 0x5FFFFFFF。假设所有的内存访问地址在对应的 cache block 中的 offset 都是 0，多处理器之间采用广播的方式进行通信。
假设宇宙射线导致该系统的某个 cache line 中的 MESI 协议状态发生了变化，导致了 cache 的不一致。下图给出了系统初始状态（已经发生了宇宙射线导致的变化）：

初始状态

Cache 0			Cache 1		
	Tag	MESI State		Tag	MESI State
Set 0	0x5FFFFF	M	Set 0	0x522222	I
Set 1	0x5FFFFF	E	Set 1	0x510000	S
Set 2	0x5FFFFF	S	Set 2	0x5FFFFF	S
Set 3	0x5FFFFF	I	Set 3	0x533333	S
Cache 2			Cache 3		
	Tag	MESI State		Tag	MESI State
Set 0	0x5F111F	M	Set 0	0x5FF000	E
Set 1	0x511100	E	Set 1	0x511100	S
Set 2	0x5FFFFF	S	Set 2	0x5FFFF0	I
Set 3	0x533333	S	Set 3	0x533333	I

内存地址的组成：Tag+SetIndex+BlockOffset。
在直接映射缓存中，内存地址的 Set Index 和 Tag 共同作用来决定缓存中存储的是哪个内存块。

（a）哪个 cache line 的状态因为宇宙射线导致了变化，产生了不一致？
Cache 2, Set 1 应该是 S 状态。或者 Cache 3, Set 1 应该是 I 状态

所以只看Set Index 和 Tag相同的就行

（b）在初始状态基础上，下面的程序执行顺序，是否会导致错误的执行结果（即读到的数据不正确）？

主要看宇宙射线改变的部分

高位
set0: 00
set1: 40
set2: 80
set3: C0

order	Processor 0	Processor 1	Processor 2	Processor 3
1			ld 0x51110040	
2	st 0x5FFFFF40			
3				st 0x51110040
4		ld 0x5FFFFF80		
5		ld 0x51110040		
6		ld 0x5FFFFF40		

不会引起错误。第 3 条指令会访问 processor 3 的 set 1，将其变为 M 状态，而 processor 2 的 set1 会变为 I 状态。这样就回到了一致状态。

（c）在初始状态基础上，下面的程序执行顺序，是否会导致错误的执行结果（即读到的数据不正确）？

order	Processor 0	Processor 1	Processor 2	Processor 3
1				ld 0x51110040
2	ld 0x5FFFFF00			
3			ld 0x51234540	
4	st 0x5FFFFF40			
5				ld 0x51234540
6	ld 0x5FFFFF00			

会，当宇宙射线导致了 Cache 3 从 I 变为 S，第 1 条指令会读到错误数据。

因为原先 I 状态下的数据是脏数据

（d）在初始状态基础上，怎么通过最少的内存访问指令，使得系统最终状态变成下图所示（假设宇宙射线不会再发生了），请写出内存访问顺序，具体操作（ld/st），访问的内存地址以及由哪个处理器发起。

Cache 0			Cache 1		
	Tag	MESI State		Tag	MESI State
Set 0	0x5FFFFF	M	Set 0	0x5FF000	I
Set 1	0x5FFFFF	E	Set 1	0x510000	S
Set 2	0x5FFFFF	S	Set 2	0x5FFFFF	S
Set 3	0x5FFFFF	E	Set 3	0x533333	I
Cache 2			Cache 3		
	Tag	MESI State		Tag	MESI State
Set 0	0x5F111F	M	Set 0	0x5FF000	M
Set 1	0x511100	E	Set 1	0x511100	S
Set 2	0x5FFFFF	S	Set 2	0x5FFFF0	I
Set 3	0x533333	I	Set 3	0x533333	I

最小的指令序列为：

（1）st 0x533333C0 //processor 0

（2）ld 0x5FFFFFC0 //processor 0 ？

（3）ld 0x5FF00000 //processor 1

（4）st 0x5FF00000 //processor 3

只要保证（1）（2）之间的顺序，（3）（4）之间的顺序，其他指令顺序也可以。