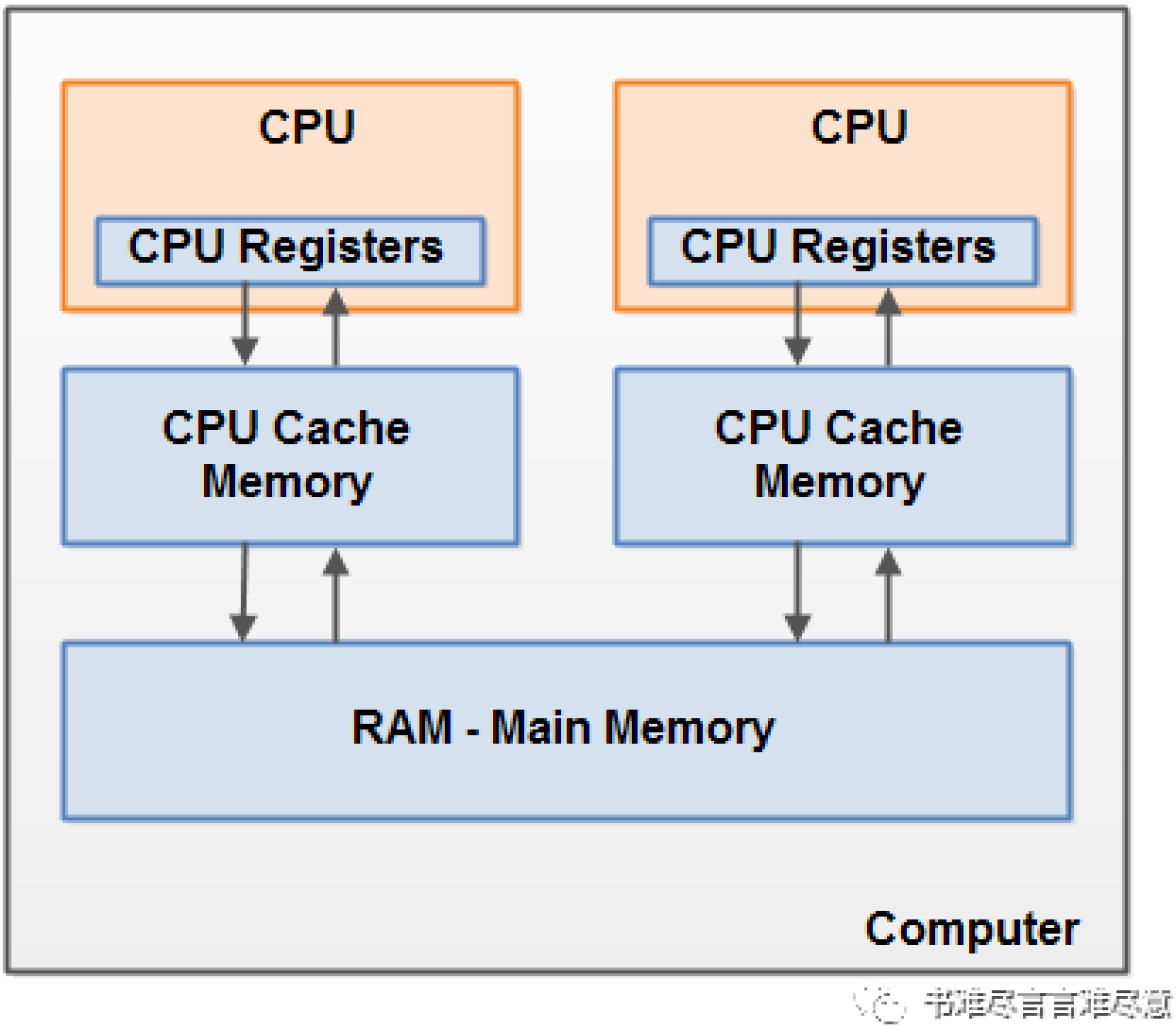
带你了解缓存一致性协议 MESI

1 CPU Cache 结构



CPU 在执行指令的时候需要从 memory 中获取指令和需要的数据，但是 CPU 的速度要比 memory 快很多，这就导致了 CPU 大部分时间都不是在做运算上而是用在了和 memory 进行数据的 I/O 过程中，这样性能是很低的。这就导致了 CPU cache 的产生，CPU 将数据从 memory 读取到 cache 中，在 cache 中对数据进行读写的速度是很快的，这样就提高了性能。CPU 执行运算时不可能需要某一个数据就去读取一次，这样就增加了 I/O 的频率，导致性能低下。所以会一次性读取一块内存的数据存放到 cache 中， 这个块称为 “cache line” ， cache line 的大小是固定的， 通常是 2 的 N 次方，在 16 ~ 256 byte 不等。当某个数据首次被 CPU 访问时， cache 中不存在，这称为 “cache miss” （或 “startup” 或 “warmup” cache miss)。这意味着 CPU 必须要去 memory 中读取该数据，这时候 CPU 必须等待（stalled)。当 cache 装满后，后续的 cache miss 需要置换 cache 中现有的数据，这样的 cache miss 被称为 “capacity miss” 。如果随后又访问了被替换的 cache line ，这时的 cache miss 被称为 “associativity miss” 。

当 CPU 写数据的时候，需要保证该数据在多个 CPU cache 之间的一致性。在写之前必须先让其他 CPU cache 中的该数据失效，之后才可以安全的写数据。如果这个数据已经存在于要执行写指令的 CPU cache 中，但是是 read only 的，这个过程被称为 "write miss" ,一旦其他 CPU cache 中的该数据都失效后，该 CPU 可以不断的写或读其 cache 中的数据。如果另外某一个 CPU 尝试访问该数据，会形成一次 cache miss ，这时称为 “communication miss” 。因为这通常是多个 CPU 之间使用缓存通信造成的。

2. 缓存一致性协议

缓存一致性协议用于管理多个 CPU cache 之间数据的一致性，这些协议十分复杂，在这里我们仅讨论 MESI 协议的四种状态。

M : modified
E : exclusive
S : shared
I : invalid

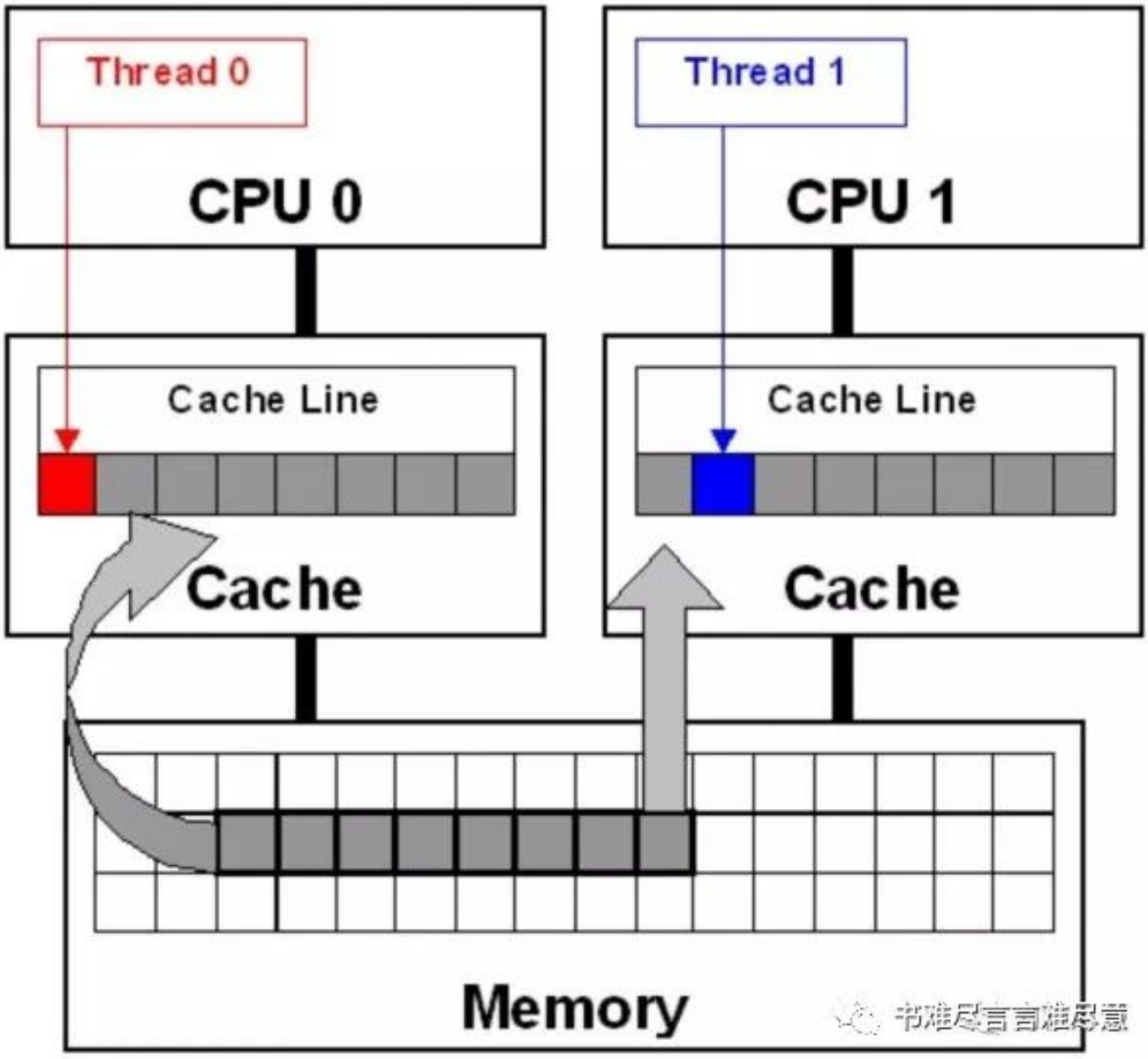
协议在每一个 cache line 中维护一个两位的状态 “tag” ，这个 “tag” 在 cache line 的物理地址或者数据后。

modified 状态的 cache line 是由于相应的 CPU 进行了写操作，同时也表示该数据不会存在于其他 CPU 的 cache 中。也就说明这个最新的数据目前是被执行写操作的 CPU cache line 独占的。因为如此当前 cache 需要对该数据负责，比如将该数据传递给其他 CPU cache ，或者是将该数据写回到 memory 中。而这些操作都需要在 reuse cache line (置换)之前完成。

exclusive 状态和 modified 状态类似，区别是 CPU 还没有修改 cache line 中的数据。在 exclusive 状态下 CPU 可以不通知其他 CPU 而直接 cache line 进行操作。也就是说 exclusive 状态是该 CPU 对这个数据的独占。此时由于 memory 中和 cache line 中的数据都是最新的，所以不需要对 exclusive 状态的 cache line 执行写回 memory 的操作就可以直接 reuse。

shared 状态的 cache line ，其数据可能在一个或者是多个 CPU cache 中，此时 CPU 不能直接修改 cache line 中的数据，而是需要先通知其他 CPU cache 。和 exclusive 一样， shared 状态的 cache line 对应的 memory 中的数据也是最新的，因此也可以直接 reuse cache line。

invalid 状态时 cache line 是空的。当新的数据要进入 cache 的时候优先选择 invalid 的 cache line ，之所以如此是因为如果选中其他状态的 cache line 会涉及到 cache line 中数据的置换 ， 而之后如果这个被替换的数据被访问到会造成 cache miss ，这样造成很大的开销。



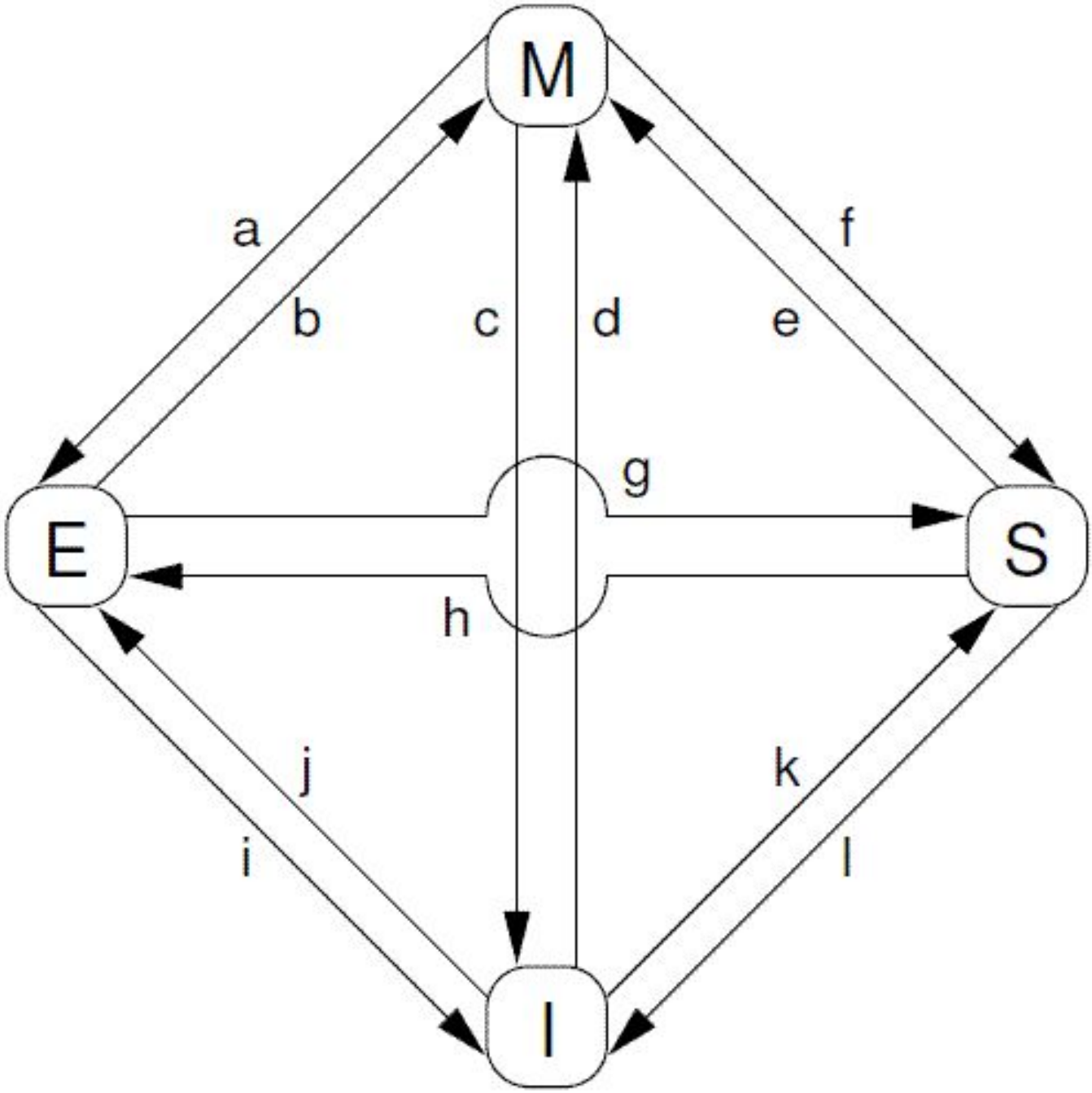
	CPU是否独占数据	cacheline数据	memory数据	直接写数据
modified	是	最新	最新	可以
exclusive	是	最新	最新	可以
shared	否	最新	最新	不可以
invalid	否（无数据）	无数据	最新	无数据

3 MESI 协议消息

1. Read。"read" 消息用来获取指定物理地址上的 cache line 数据。
2. Read Response。该消息携带了 “read” 消息所请求的数据。read response 可能来自于 memory 或者是其他 CPU cache。
3. Invalidate。该消息将其他 CPU cache 中指定的数据设置为失效。该消息携带物理地址，其他 CPU cache 在收到该消息后，必须进行匹配，发现在自己的 cache line 中有该地址的数据，那么就将其从 cahe line 中移除，并响应 Invalidate Acknowledge 回应。

4. Invalidate Acknowledge。该消息用做回应 Invalidate 消息。
5. Read Invalidate。该消息中带有物理地址，用来说明想要读取哪一个 cache line 中的数据。这个消息还有 Invalidate 消息的效果。其实该消息是 read + Invalidate 消息的组合，发送该消息后 cache 期望收到一个 read response 消息。
6. Writeback。 该消息带有地址和数据，该消息用在 modified 状态的 cache line 被置换时发出，用来将最新的数据写回 memory 或其他下一级 cache 中。

4 MESI状态图



根据 MESI 协议消息的发送和接收或者是对数据的读写，cache line 的状态会在 modified ， exclusive , shared , invalid 之间进行转换。

1. cache 通过 writeback 将数据回写到 memory 或者下一级 cache 中。这时候状态由 modified 变成了 exclusive 。
2. cpu 直接将数据写入 cache line ，导致状态变为了 modified 。
3. CPU 收到一个 read invalidate 消息，此时 CPU 必须将对应 cache line 设置成 invalid 状态，并且响应一个 read response 消息和 invalidate acknowledge 消息。
4. CPU 需要执行一个原子的 readmodify-write 操作，并且其 cache 中没有缓存数据。这时候 CPU 就会在总线上发送一个 read invalidate 消息来请求数据，并试图独占该数据。CPU 可以通过收到的 read response 消息获取到数据，并等待所有的 invalidate acknowledge 消息，然后将状态设置为 modifie 。
5. CPU需要执行一个原子的readmodify-write操作，并且其local cache中有read only的缓存数据（cacheline处于shared状态），这时候，CPU就会在总线上发送一个invalidate请求其他cpu清空自己的local copy，以便完成其独自霸占对该数据的所有权的梦想。同样的，该cpu必须收集所有其他cpu发来的invalidate acknowledge之后才能更改状态为 modified。
6. 在本cpu独自享受独占数据的时候，其他的cpu发起read请求，希望获取数据，这时候，本cpu必须以其local cacheline的数据回应，并以read response回应之前总线上的read请求。这时候，本cpu失去了独占权，该cacheline状态从Modified状态变成shared状态（有可能也会进行写回的动作）。
7. 这个迁移和f类似，只不过开始cacheline的状态是exclusive，cacheline和memory的数据都是最新的，不存在写回的问题。总线上的操作也是在收到read请求之后，以read response回应。
8. 如果cpu认为自己很快就会启动对处于shared状态的cacheline进行write操作，因此想提前先霸占上该数据。因此，该cpu会发送invalidate敦促其他cpu清空自己的local copy，当收到全部其他cpu的invalidate acknowledge之后，transaction完成，本cpu上对应的cacheline从shared状态切换exclusive状态。还有另外一种方法也可以完成这个状态切换：当所有其他的cpu对其local copy的cacheline进行写回操作，同时将cacheline中的数据设为无效（主要是为了为新的数据腾些地方），这时候，本cpu坐享其成，直接获得了对该数据的独占权。
9. 其他的CPU进行一个原子的read-modify-write操作，但是，数据在本cpu的cacheline中，因此，其他的那个CPU会发送read invalidate，请求对该数据以及独占权。本cpu回送read response”和“invalidate acknowledge”，一方面把数据转移到其他cpu的cache中，另外一方面，清空自己的cacheline。
10. cpu想要进行write的操作但是数据不在local cache中，因此，该cpu首先发送了read invalidate启动了一次总线transaction。在收到read response回应拿到数据，并且收集所有其他cpu发来的invalidate acknowledge之后（确保其他cpu没有local copy），完成整个bus transaction。当write操作完成之后，该cacheline的状态会从Exclusive状态迁移到Modified状态。
11. 本CPU执行读操作，发现local cache没有数据，因此通过read发起一次bus transaction，来自其他的cpu local cache或者memory会通过read response回应，从而将该 cache line 从Invalid状态迁移到shared状态。
12. 当cache line处于shared状态的时候，说明在多个cpu的local cache中存在副本，因此，这些cacheline中的数据都是read only的，一旦其中一个cpu想要执行数据写入的动作，必须先通过invalidate获取该数据的独占权，而其他的CPU会以invalidate acknowledge回应，清空数据并将其cacheline从shared状态修改成invalid状态。