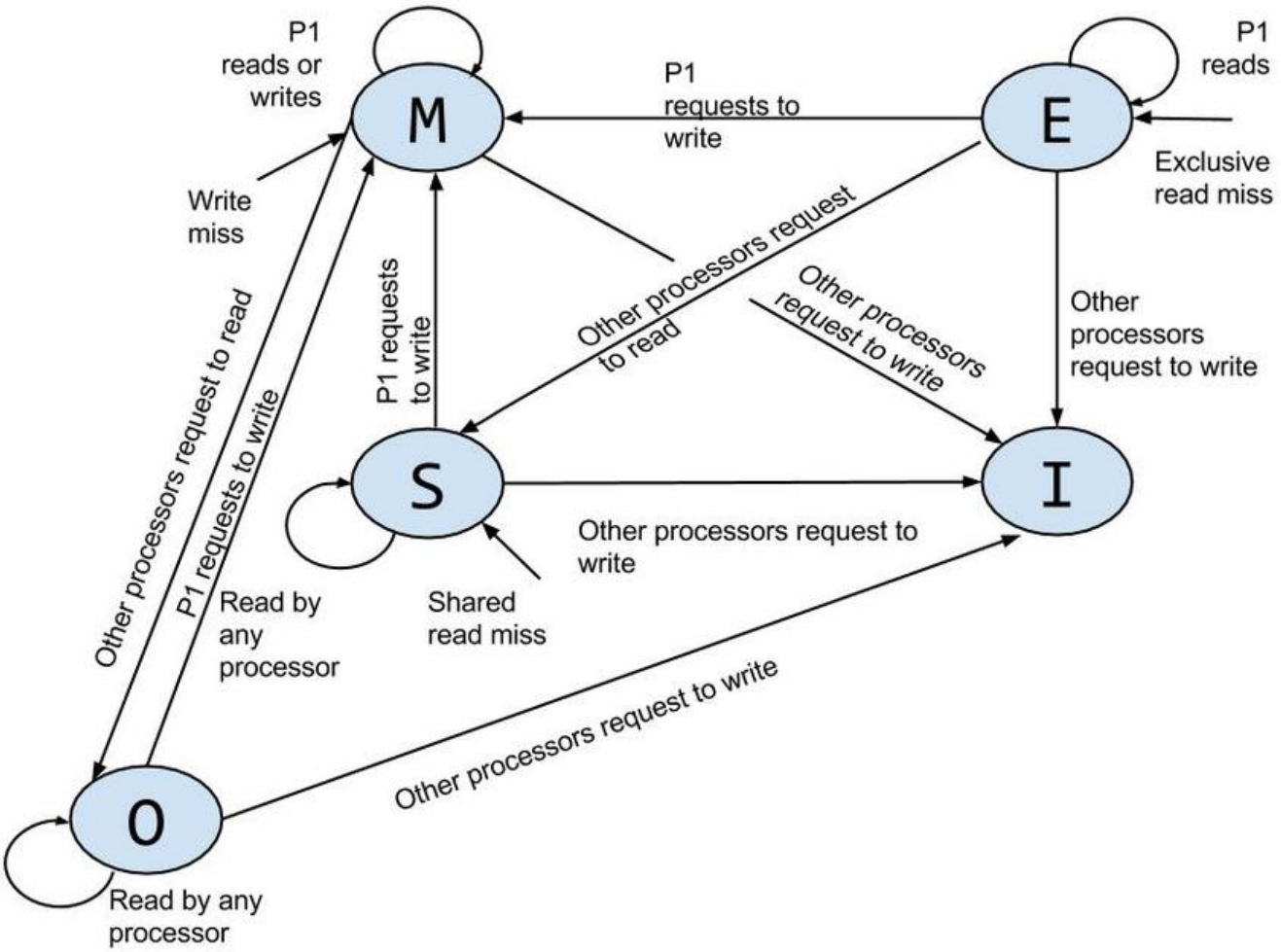
Cache

Many coherence protocols use a subset of the five state MOESI model.

MOESI Model

- **M(odified)**: The block may be read or written. The cache has the only valid copy of the block, the cache must respond to requests for the block, and the copy of the block at the memory is potentially stale.
- **O(wned)**: The cache has a read-only copy of the block and must respond to requests for the block. Other caches may have a read-only copy of the block. The copy of the block in the memory is potentially stale.
- **E(xclusive)**: The cache has a read-only copy of the block. No other caches have a valid copy of the block, and the copy of the block in the memory is up-to-date.
- **S(hared)**: The cache has a read-only copy of the block. Other caches may have valid, read-only copies of the block.
- **I(nvalid)**: The cache either does not contain the block or it contains a potentially stale copy that it may not read or write.



State Transition Examples

Consider a system with 4 cores, each having its own cache, and a shared memory. Initially, all caches are in the Invalid state. Assume all the operations are targeted for the same block.

1. **[Invalid]** Initial state:

Core0	Core1	Core2	Core3
Invalid	Invalid	Invalid	Invalid

2. **[Invalid ⇒ Exclusive]** Core0 performs a read request and encounters a read miss

Core0	Core1	Core2	Core3
Exclusive	Invalid	Invalid	Invalid

3. **[Exclusive ⇒ Modified]** Core0 performs a write operation

Core0	Core1	Core2	Core3
Modified	Invalid	Invalid	Invalid

4. **[Modified ⇒ Owned / Invalid ⇒ Shared]** Core1 performs a read request and encounters a read miss

Core0	Core1	Core2	Core3
Owned	Shared	Invalid	Invalid

5. **[Owned ⇒ Invalid / Shared ⇒ Invalid]** Core2 performs a write request and encounters a write miss.

Core0	Core1	Core2	Core3
Invalid	Invalid	Modified	Invalid

6. **[Modified ⇒ Invalid]** Core3 performs a write request and encounters a write miss

Core0	Core1	Core2	Core3
Invalid	Invalid	Invalid	Modified

Cohrence Protocols

> - **Snooping protocol**: A cache controller broadcasts a request message to all other coherence controllers.
> - **Directory protocol**: A cache controller unicasts a request message to the memory controller that is the home for that block. Each memory controller maintains a directory that holds state and sharers of each block in the memory.

> - **Invalidate protocol**: When a core wishes to write to a block, it invalidates the copies in all other caches. If another core wishes to read the block whose copy is invalidated, it has to obtain the block

> from the core that wrote it, thus preserving coherence.
> - **Update protocol**: When a core wishes to write a block, it updates the copies in all other caches to reflect the new value it wrote to the block.

MSHR (Miss Status Handling Registers)

When a miss returns, the processor must know which load or store caused the miss, so that instruction can now go forward; and it must know where in the cache the data should be placed. This information is kept in a set of registers called the **Miss Status Handling Registers (MSHRs)**.

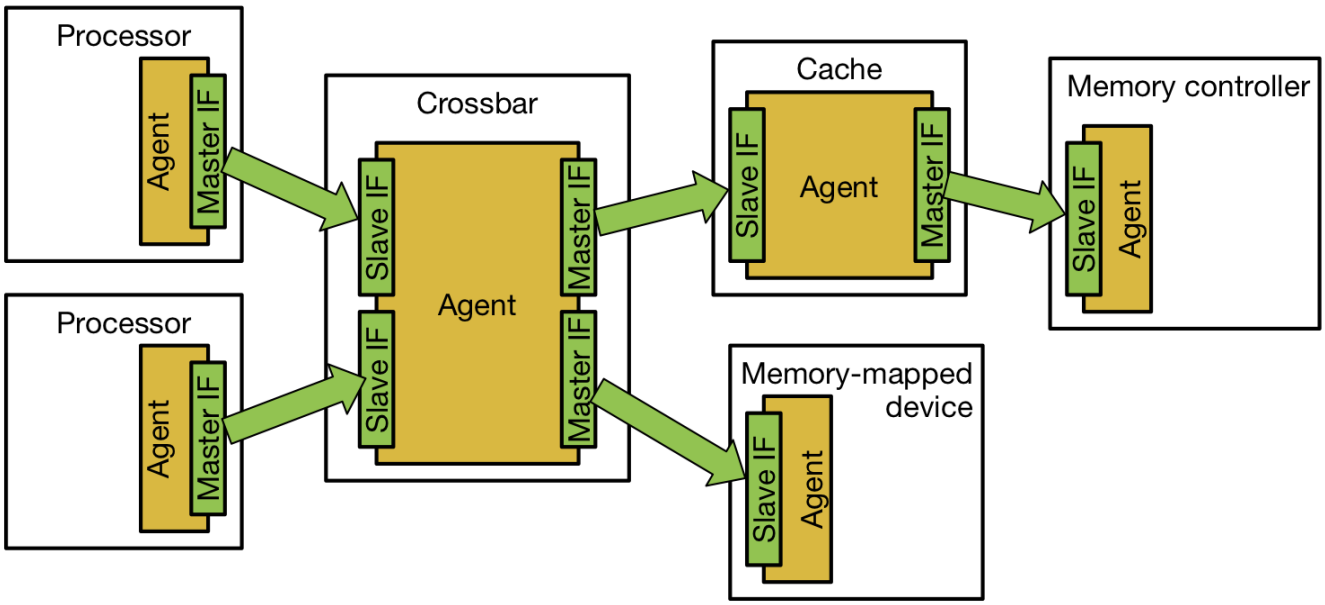
Fields of a single MSHR:

- Valid bit
- Cache block address
- Control/status bits
- Data for each subblock
- For each pending load/store
  - Valid, type, data size, byte in block, destination register or store buffer entry address

If we allow n outstanding misses, there will be n MSHRs. Thus, when a miss occurs, the cache allocates an MSHR for handling that miss, enter the appropriate information about the miss, and tag the memory request with the index of the MSHR. The memory system uses that tag when it returns the data, allowing the cache system to transfer the data and tag information to the appropriate cache block and notify the load or store that generated the miss.

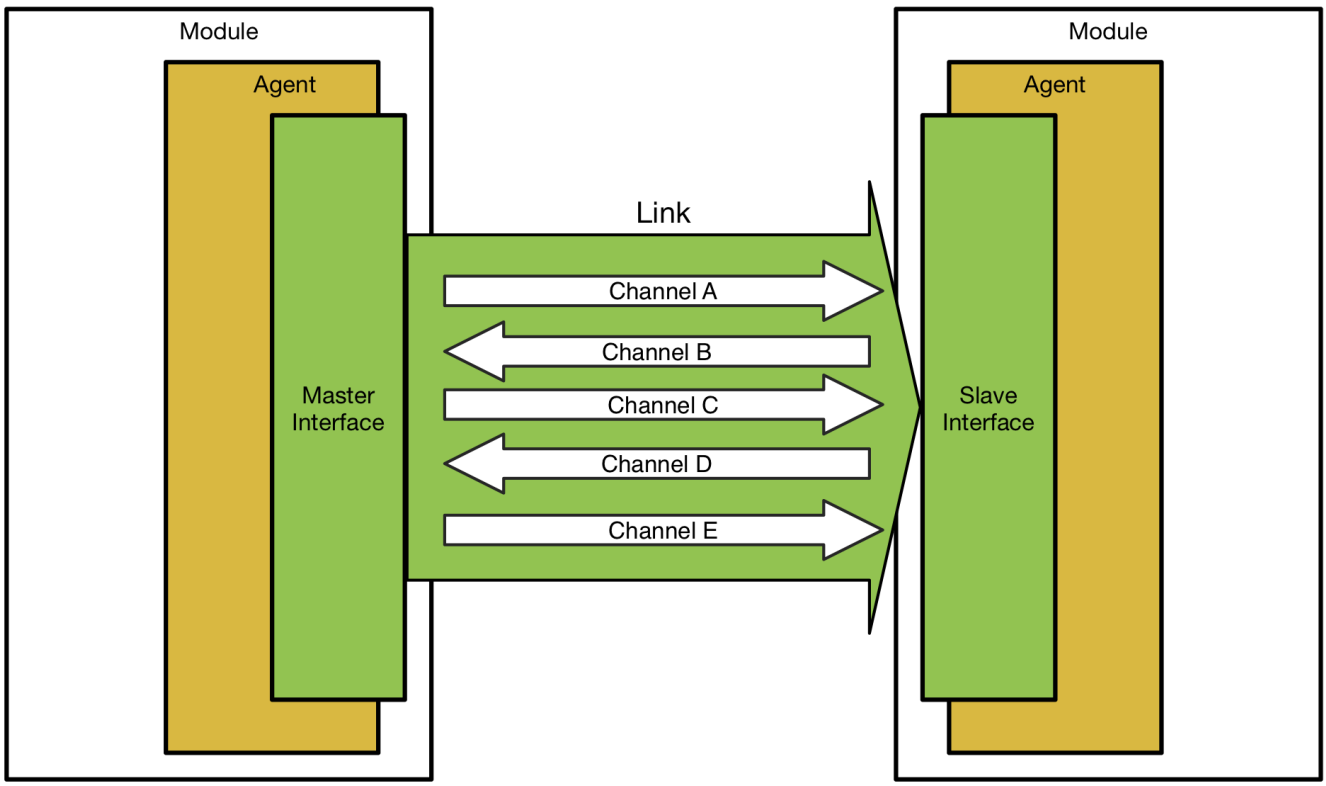
TileLink Protocol (TL-C)

Master and Slave



The agent with the master interface can request the agent with the slave interface to perform memory operations, or request permission to transfer and cache copies of data. The agent with the slave interface manages permissions and access to a range of addresses, wherein it performs memory operations on behalf of requests arriving from the master interface.

Channels



Within each network link, the TileLink protocol defines five logically independent channels over which messages can be sent by agents. Channels are directional, in that each passes messages either from master to slave interface or from slave to master interface. The prioritization of messages across channels is **A < B < C < D < E**.

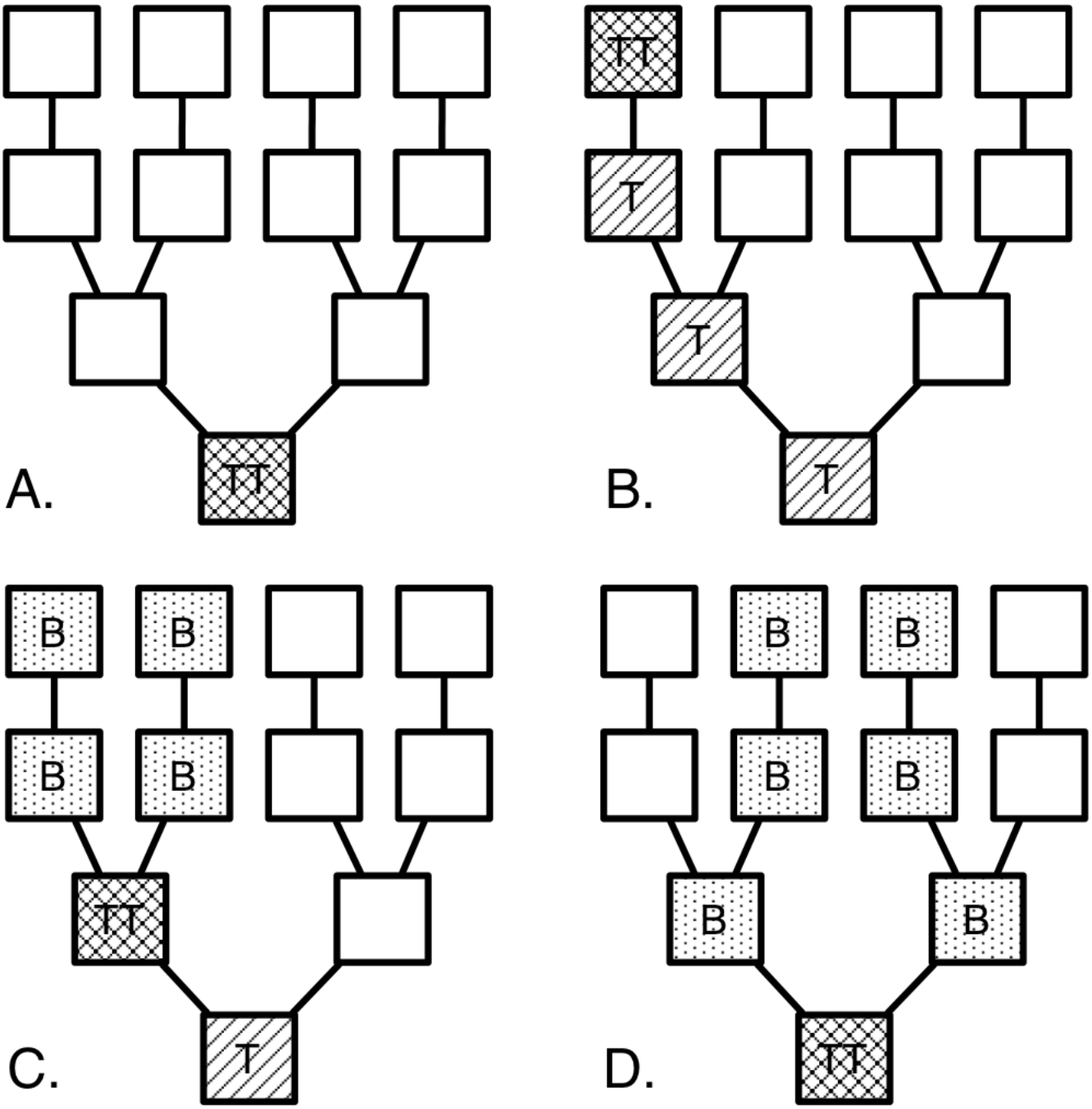
- **Channel A: Master ⇒ Slave**
  - Transmits a request that an operation be performed on a specified address range, accessing or caching the data.
  - A master initiates acquiring permission to read or write a copy of a cache block.
- **Channel B: Slave ⇒ Master**
  - Transmits a request that an operation be performed at an address cached by a master agent, accessing or writing back that cached data.
  - A slave queries or modifies a master's permissions on a cached data block, or forwards a memory access to a master.
- **Channel C: Master ⇒ Slave**
  - Transmits a data or acknowledgment message in response to a request.
  - A master acknowledges a Channel B message, potentially releasing permissions on the block along with any dirty data. Also used to voluntarily write back dirtied cache data.
- **Channel D: Slave ⇒ Master**
  - Transmits a data response or acknowledgement message to the original requestor.

    - A slave provides data or permissions to the original requestor, granting access to the cache block. Also used to acknowledge voluntary writebacks of dirty data.
- **Channel E: Master ⇒ Slave**
    - Transmits a final acknowledgement of a cache block transfer from the original requestor, used for serialization.
    - A master provides final acknowledgment of transaction completion, used by the slave for transaction serialization.

Cache Cohrence

For any given address, there is exactly one path between any given master and the slave that owns that address. When all such paths form a tree with a single slave at the root. The inclusive TileLink coherence protocol requires the tree to grow and shrink in response to memory access operations. Every node in the graph falls into one of four categories describing its position on the tree:

- **Nothing** A node that does not currently cache a copy of the data. Has neither read nor write permissions.
- **Trunk** A node with a cached copy that is on the path between the Tip and the Root. Has neither read nor write permissions on the copy. The copy may be out of date with respect to writes occurring at the Tip.
- **Tip** Has read and write permissions on its copy, which may contain dirty data.
    - **With No Branches** A node with a cached copy that is serving as the point of memory access serialization.
    - **With Branches** A node with a cached copy that is serve as the point of write serialization.
- **Branch** A node with a cached copy that is above the Tip. Has read-only permissions on its copy.



TT is Trunk Tip, **T** is trunk, and **B** is branch.

- A: The root has write and read permission on the only copy.
- B: A single master has write and read permissions on the trunk tip.
- C: Multiple masters have read permissions on branches.
- D: Multiple masters have read permissions on branches, and some branches have been pruned.