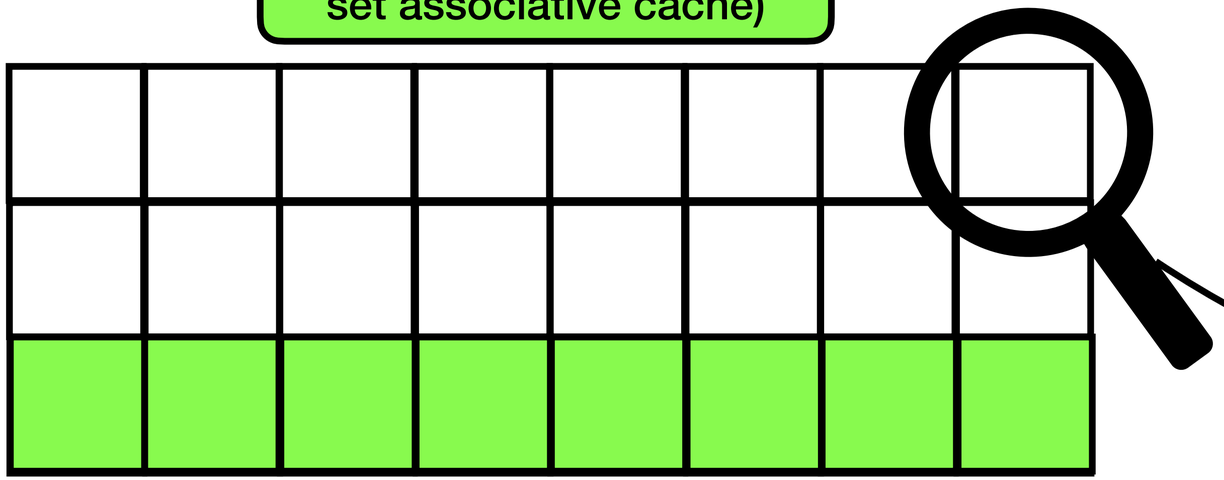
Cache Construction

Outline

- Fundamental cache components
- Cache organization
- Understanding cache misses

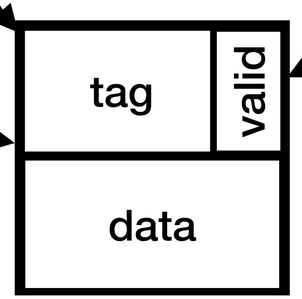
Overview of Basic Cache Terminology

Associativity: describes size of cache sets in a cache (e.g., this is an 8-way set associative cache)



Tag: unique identifier for the data in a cache block

Valid bit: indicates if block currently holds legitimate data



Set: the possible blocks to which some data may be stored within the cache

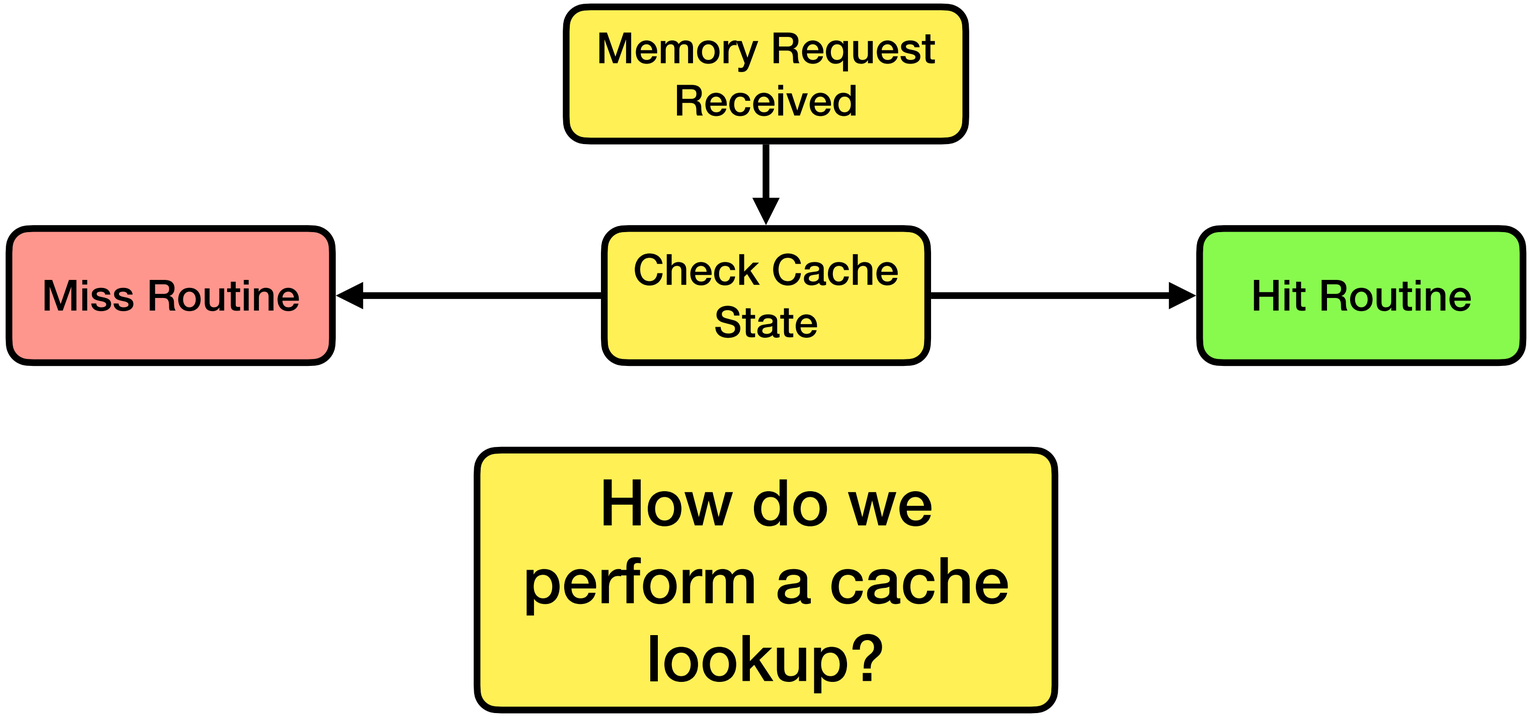
Cache: the collection of *cache blocks* that buffers/stores commonly reused items

Cache Block: the minimum unit of information that can be present/not present in a cache (typically data is the size of a "word" ➡️ 64 bytes in modern processors)

Problems to Address in Cache Construction

- How do we decide where to place data in the cache?
- What control information do we need to track to implement the cache?
- How do we decide what data to evict?
- How do we maintain consistent data?

Constructing a Cache State Machine (Part 1)

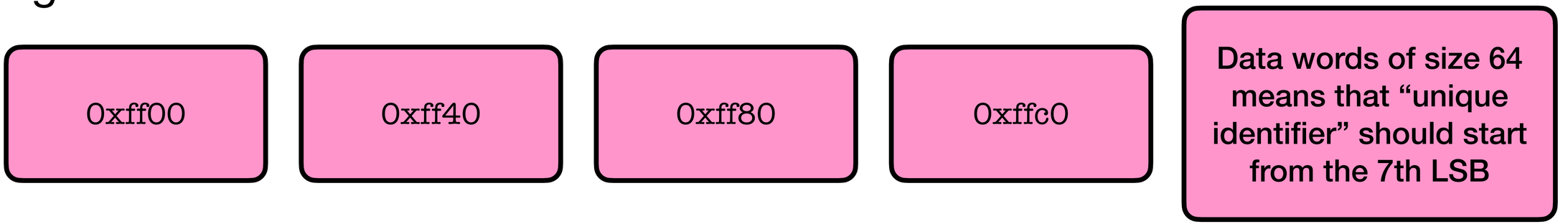


Chat with your neighbor(s)!

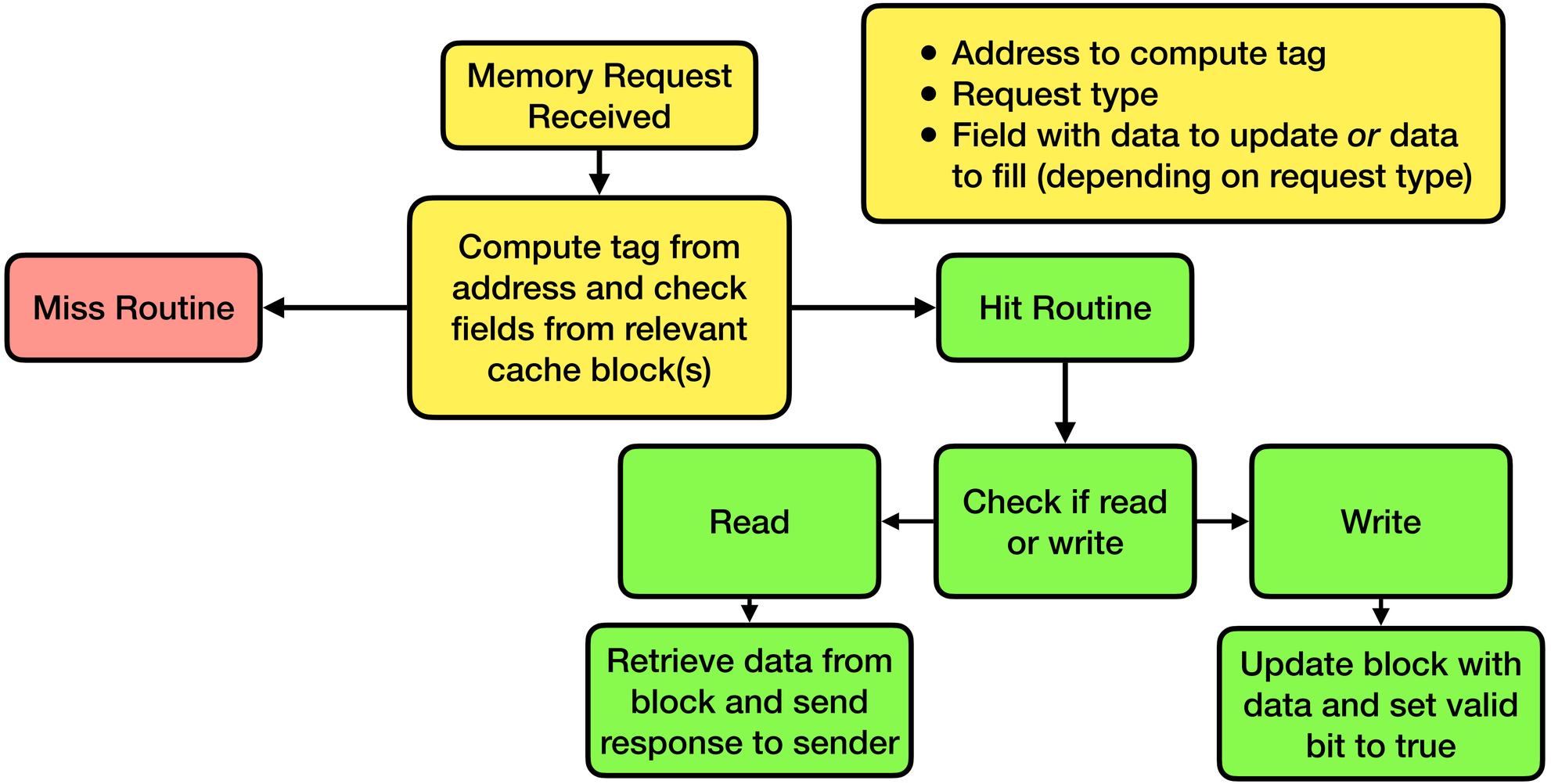
Suppose we want to perform a cache lookup. What information does a cache controller need to be able to find the requested block? Think both in terms of what needs to be communicated in the request and what needs to be stored.

Cache Block Fields

- Valid bit: (for now) used to initialize the state of the data ➡️ before the first access, block is *invalid* (i.e., no data is in the cache block). Valid set to true after the block is filled with relevant information
- Tag: the unique identifier of some data ➡️ typically this refers to the address of that data block!
- Cache block addresses will be aligned to the data word, so the least significant bits will be the same



Constructing a Cache State Machine (Part 2)



Writeback versus Write-Through

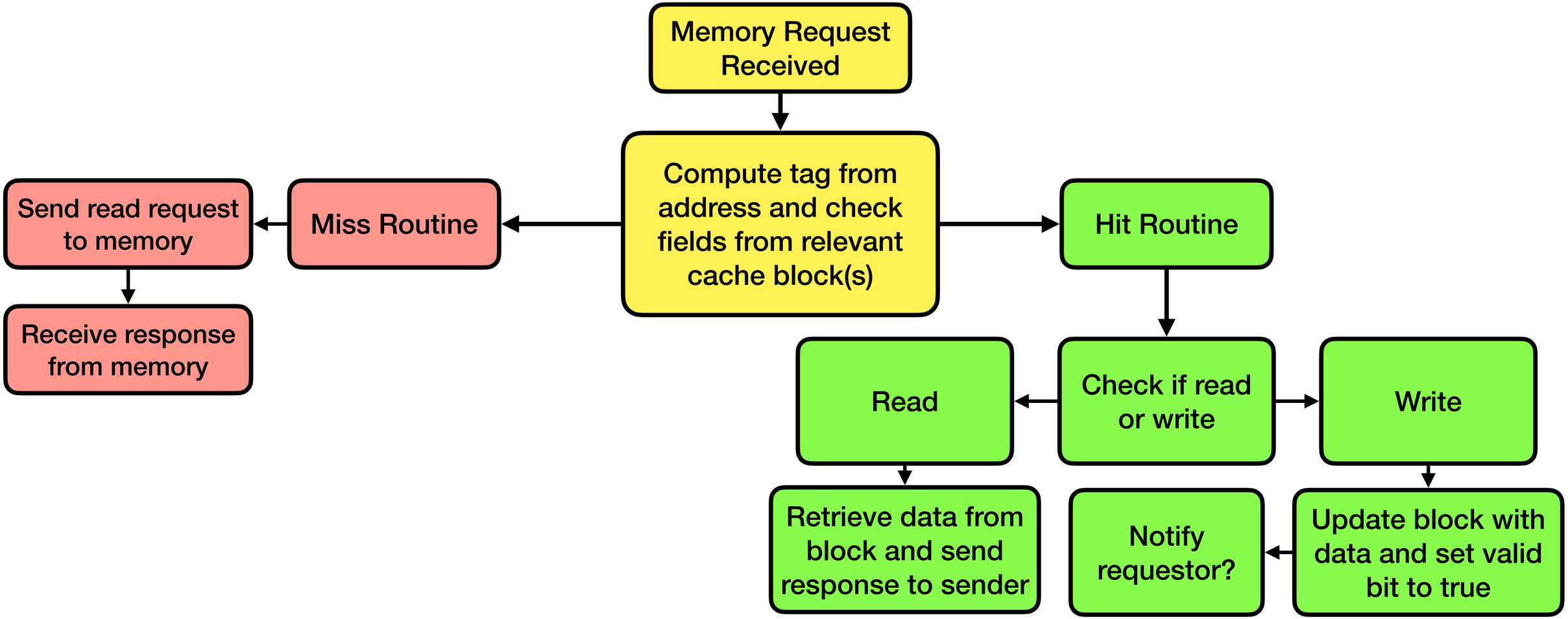
- Writeback: update cache blocks in lower levels of the cache hierarchy only when data is evicted from the cache (i.e., data is *lazily* updated into lower levels of the memory hierarchy)
- Write-Through: every time data is updated in the cache, that update must also propagate to the lower levels of the memory hierarchy

Takeaway: there is no direct consequence on runtime to using a through cache, but it does add traffic!

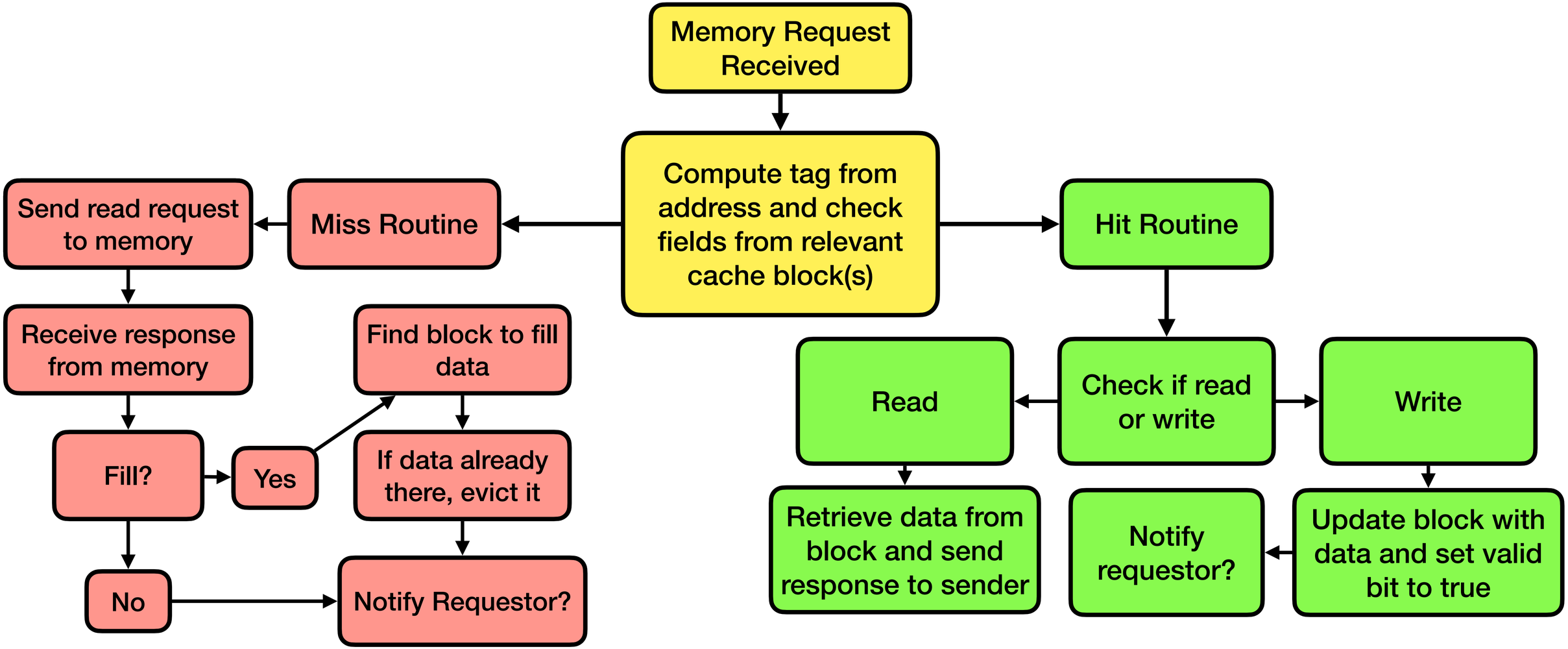
Takeaway: if a memory system implements writeback caching, it may need to support more memory requests to allow for explicit flushing to storage

Takeaway: writeback caching allows for more *asynchronous* behavior, which can allow for faster (but more complicated) cases in the memory system

Constructing a Cache State Machine (Part 3)

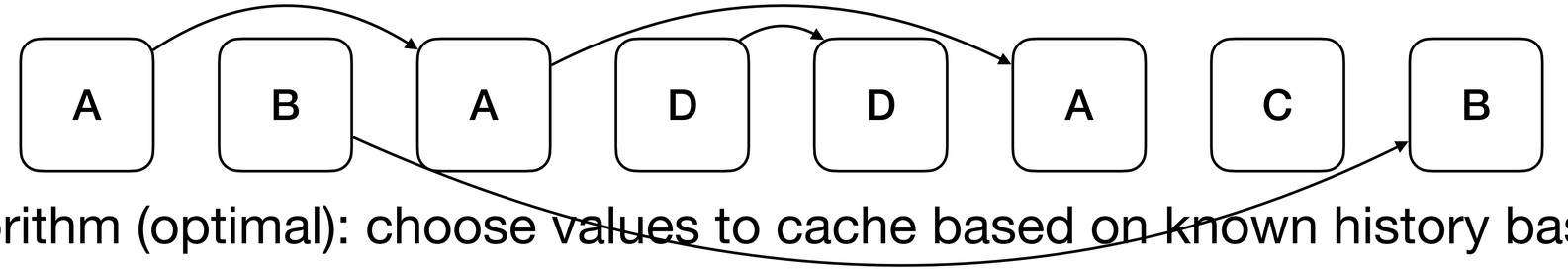


Constructing a Cache State Machine (Part 3)



Cache Replacement Policies

- Find an eviction target from the candidates within a set
- If an invalid block exists within the set, *greedily* fill that block
- If no valid block exists, choose a target to evict that makes sense relative to the application behavior



- Belady's algorithm (optimal): choose values to cache based on known history based on *reuse distance*
- In practice, *least recently used* is a heuristic that works well ➡️ you will explore alternatives in HW2 (written)!

Takeaways

- We construct caches from fundamental components (cache blocks and memory requests)
- Caches have can implement various strategies to more efficiently build memory system behaviors
- (Re)placement policies can help determine the optimal set of values to maintain in the cache