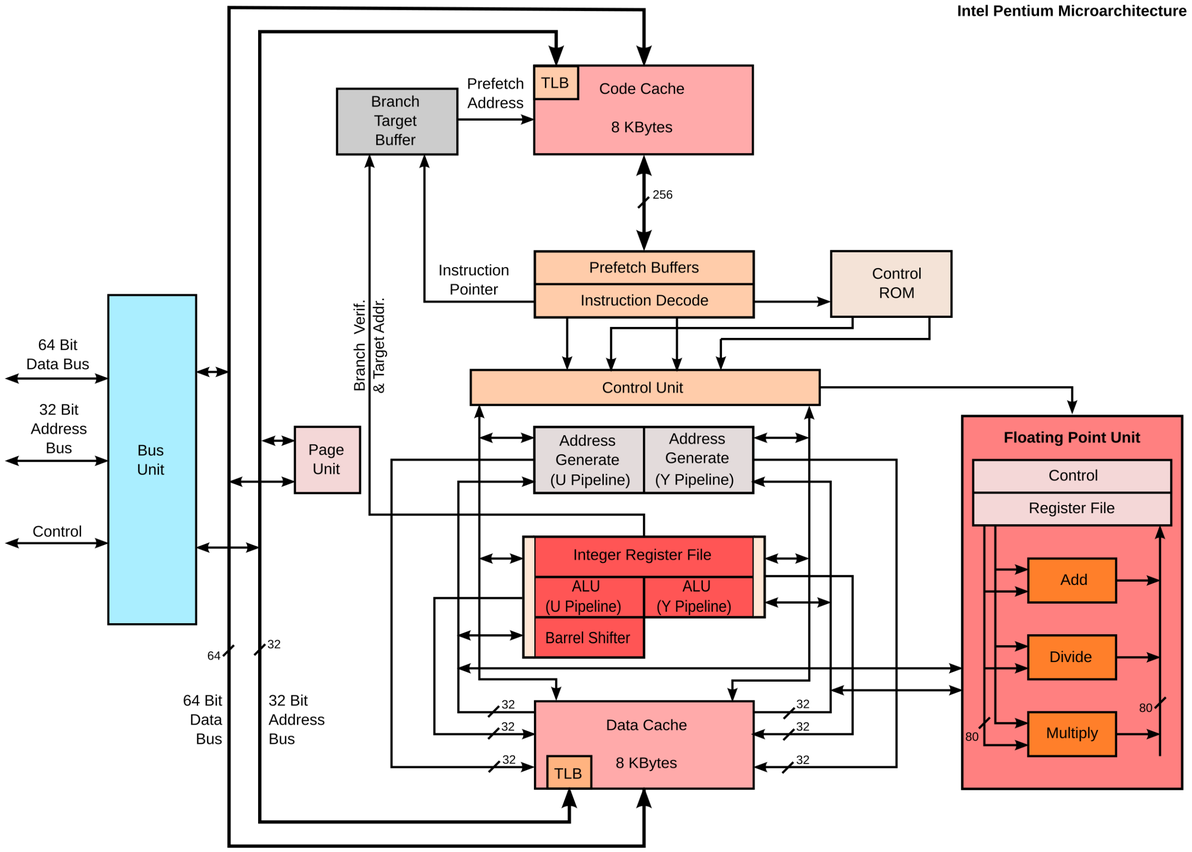
Cache Construction, continued!



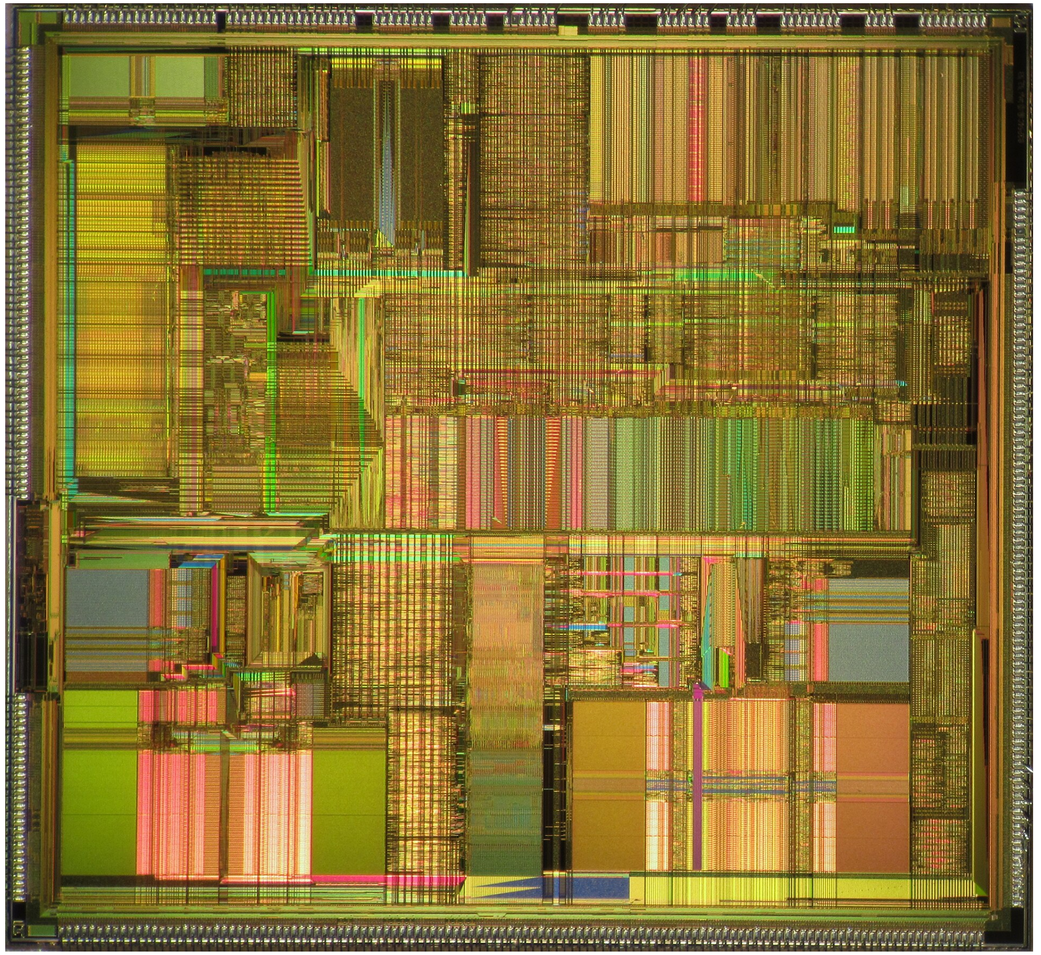


Image credit: https://en.wikipedia.org/wiki/Pentium_(original)

Outline

- Wrapping up the cache protocol state machine
- Overviewing lookups with associativity
- Understanding cache misses

(From Monday) Writeback versus Write-Through

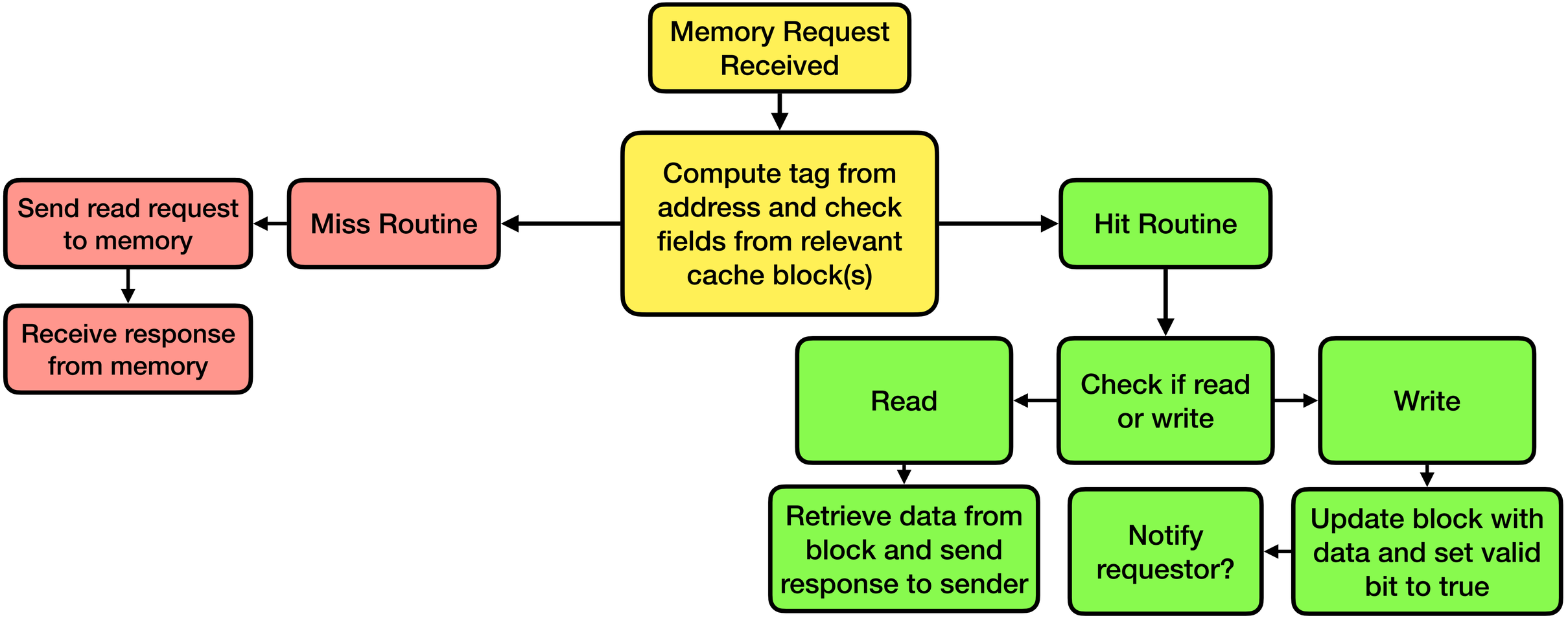
- Writeback: update cache blocks in lower levels of the cache hierarchy only when data is evicted from the cache (i.e., data is *lazily* updated into lower levels of the memory hierarchy)
- Write-Through: every time data is updated in the cache, that update must also propagate to the lower levels of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



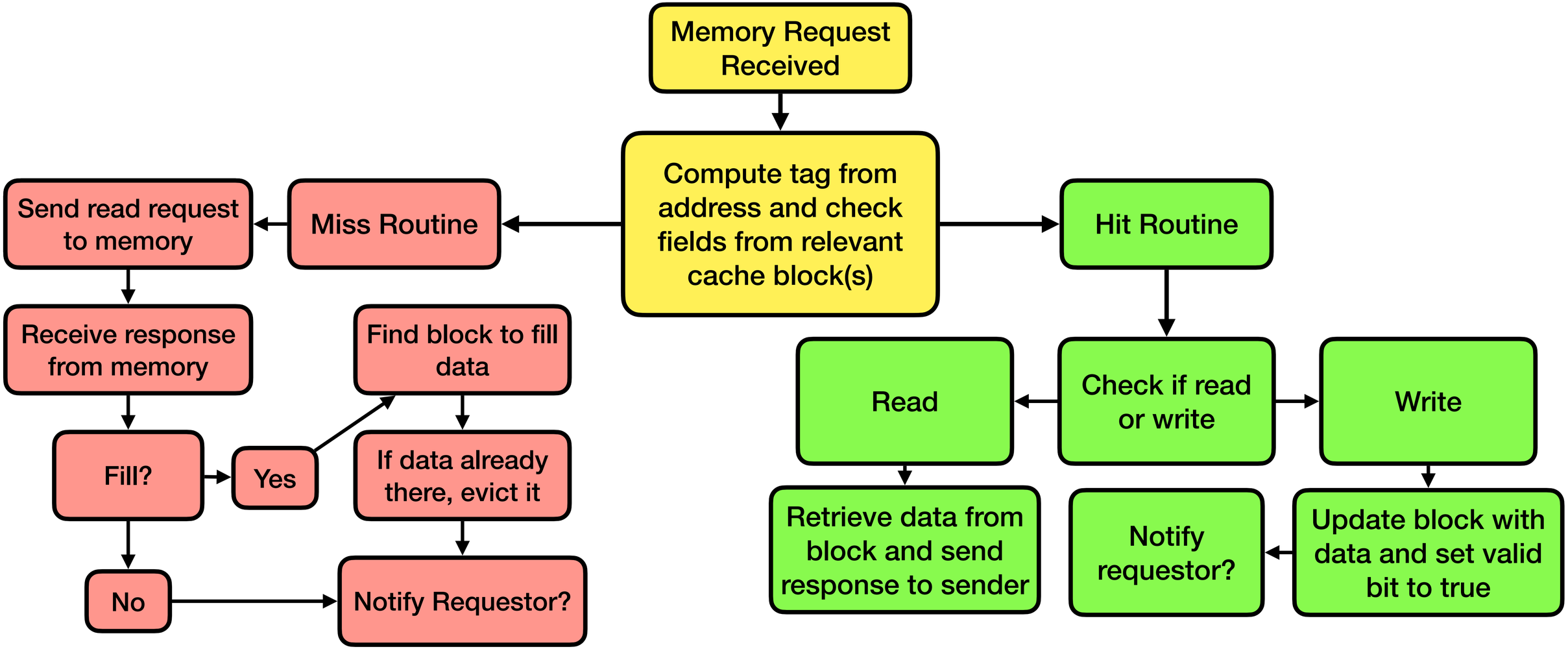
Write Allocate versus No-Write Allocate

- Write Allocate: upon receiving a response from a cache miss, *fill* the missed block with the fetched data from memory
- No-Write Allocate: upon receiving a response from memory, send the data to the processor without filling the cache with the received data

Chat with your neighbor(s)!

Suppose a cache controller is configured to implement a no-write allocate cache fill policy and a writeback cache update policy. Describe what you would expect to happen if the cache receives a “write” request if the block is in the cache versus not in the cache.

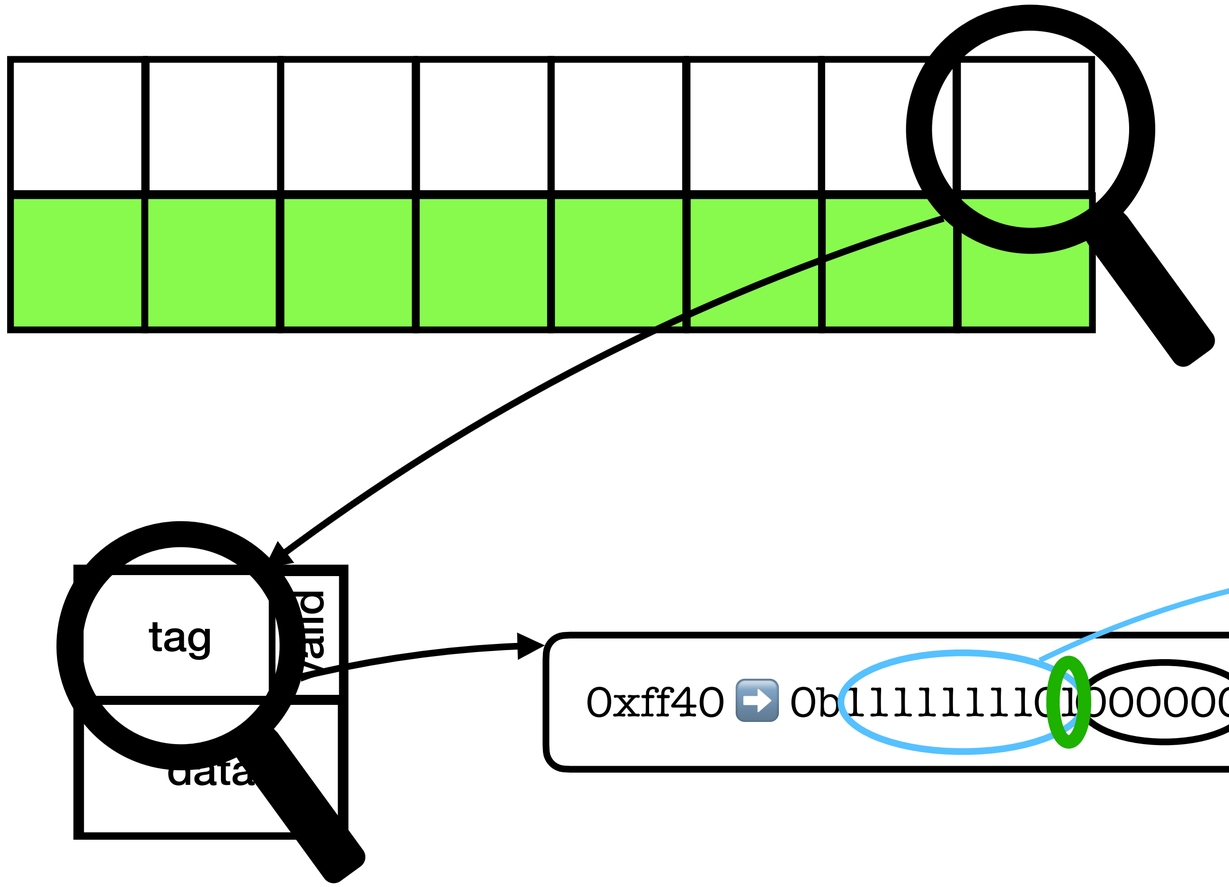
Constructing a Cache State Machine (Part 4)



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?

Indexing Into a Cache



The cache associativity help define how many possible cache blocks the data *could* be located in! So we need to use some bits from the address to identify the set ➡️ two possible sets means one identifying bit

Should the MSBs be the tag?

LSBs are consistent for all cached blocks

Looking Up Data in a Set-Associative Cache

- Recall: an *n-way set associative cache* means that there are *n* cache blocks per set
- This means that the number of locations to search for a cache block are defined by the number of blocks in a set
- A *fully associative cache* is a cache where all blocks belong to the same set (i.e., the data could be anywhere in the cache)
- A *direct mapped cache* is the equivalent to a having exactly one cache block per set
- To find a particular address in the set, compare the computed tag (i.e., the MSBs) to the stored tag for each block within that set ➡️ if valid and tag match, then hit!

Chat with your neighbor(s)!

Thinking about a direct mapped cache versus a fully associative cache, in which cache in the cache hierarchy (e.g., L1, L2, or L3) does it make more sense to deploy each strategy?

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 ➡️ requires additional information in the cache block!