- What are the tradeoffs with the Approximate Counter in the book when updating the global counter more or less frequently?

The approximate counter method, utilizes a local counter for each core on the CPU and a single global counter, when a thread wants to increment the counter it does so on its local counter, and periodically the counter's value gets transferred to the global counter by obtaining its lock and incrementing it by the local counters value. How often this is determined is based off of a threshold S. The smaller S is, the less scalable the counter actually is, . The bigger S is the more scalable counter will be, but the further off the global might be from the actual count.

- Why does having locks on both the head and tail of the concurrent queue have a performance advantage over having one big lock for the whole thing?

For a concurrent queue the goal of having two locks, one on the head and one on the tail is to enable the concurrency of the enqueue and dequeue operations. In the common case the enqueue routine will only access the tail lock and the dequeue will only access the head lock.

- Why doesn't the concurrent hash table in the reading need any locks in any of the `Hash_` functions?

For the Hash Table, instead of using locks in the hash functions or around the whole structure, we can instead use a lock per hash bucket(each represented by a list), enabling many concurrent operations.

- In the concurrent queue in the book, speculate on why the dummy node is required.

In the book we are told the dummy node enables the separations of the head and tail operations, and within the code in the initialization of the queue they create a node first using that as a basis to build the queue from, this makes me think that utilizing this node helps with handling special cases that can occur in multiple threads concurrently using the queue, when the queue is in an empty state and the first element is being enqueued or dequeued.