We’ve Solved the Critical Section Problem

[This paper is in response to some system design interviews I’ve felt were way too focused on the Critical Section Problem]

How many programmers are thinking about synchronization primitives like mutual exclusion, progress, and bounded weight? A few edge cases aside, like if you’re working on system software, a relatively small portion of our mental bandwidth is concerned with race conditions and deadlocks. These problems have been solved by technologies that are readily available:

1. While it helps to know about operating system concepts like mutex and semaphores when we’re building application software, it isn’t a requirement that we explicitly know about them, our time is better used thinking about abstractions like Simple Queue Service (SQS) or Kafka.

2. While it may help to know about database replication strategies like single-leader, multi-leader, or read-write quorums, our time is better used thinking about abstractions like whether to use a relational databases or object-oriented databases and let the DBMS handle replication.

3. While it might help to know how reverse proxies and load balancers distribute tasks (I have no idea how they work), our time is better used thinking about whether it’s easier to plug in Elastic Load Balancer (ELB) or NGINX.

Less time thinking about critical section solutions allows for more time to focus on our core strengths to build better value for our users.