Concurrency Primitives

Atomic Update

The problem with updates is that there is more than one operation involved: first read, then modify, then store
Concurrency Primitives
Atomic Update

The problem with updates is that there is more than one operation involved: first read, then modify, then store

Another thread may access the shared resource in between the read and store
Concurrent Primitives

Atomic Update

The problem with updates is that there is more than one operation involved: first read, then modify, then store.

Another thread may access the shared resource in between the read and store.

This leads us to another approach to the update race condition by having indivisible *atomic update*.
Concurrency Primitives

Atomic Update

This where the hardware supplies a special instruction to, say, increment an integer as a single atomic operation.
Concurrency Primitives

Atomic Update

This where the hardware supplies a special instruction to, say, increment an integer as a single atomic operation.

This must be in the hardware: the increment instruction must prevent other modifications of that value while it is being incremented.
Concurrency Primitives

Atomic Update

This where the hardware supplies a special instruction to, say, increment an integer as a single atomic operation.

This must be in the hardware: the increment instruction must prevent other modifications of that value while it is being incremented.

The hardware sorts out the sequentialisation in the case of simultaneous (or near-simultaneous) update by different threads.
Concurrency Primitives

Atomic Update

This where the hardware supplies a special instruction to, say, increment an integer as a single atomic operation.

This must be in the hardware: the increment instruction must prevent other modifications of that value while it is being incremented.

The hardware sorts out the sequentialisation in the case of simultaneous (or near-simultaneous) update by different threads.

Note that, depending on the cpu architecture, a single atomic instruction might take possibly hundreds of cpu cycles to execute: atomics are not fast!
Atomics are indeed a reasonable approach, used by many, but they have limitations
Concurrency Primitives
Atomic Update

Atomics are indeed a reasonable approach, used by many, but they have limitations

- Atomic instructions are hard to build in the context of the complexity of caching and so on in modern systems
Atomic Update

Atomics are indeed a reasonable approach, used by many, but they have limitations:

- Atomic instructions are hard to build in the context of the complexity of caching and so on in modern systems.
- You would need an atomic instruction for each kind of update you might want to do.
Concurreny Primitives
Atomic Update

Atomics are indeed a reasonable approach, used by many, but they have limitations

- Atomic instructions are hard to build in the context of the complexity of caching and so on in modern systems
- You would need an atomic instruction for each kind of update you might want to do
- Getting a high-level language compiler to generate code using that instruction will be hard
You do see machine instructions in modern CPUs to do some selection of atomic increment and decrement of integers, add, subtract, logical and, logical or, swap two values in memory, and a couple of conditional tests but usually not much more than that
You do see machine instructions in modern CPUs to do some selection of atomic increment and decrement of integers, add, subtract, logical and, logical or, swap two values in memory, and a couple of conditional tests but usually not much more than that.

Instead, the best approach is to use a more flexible machine instruction that you can build on to make more generic higher-level solutions (see “test and set” and friends, later)
You do see machine instructions in modern CPUs to do some selection of atomic increment and decrement of integers, add, subtract, logical and, logical or, swap two values in memory, and a couple of conditional tests but usually not much more than that.

Instead, the best approach is to use a more flexible machine instruction that you can build on to make more generic higher-level solutions (see “test and set” and friends, later).

Indeed, a lock implementation might be built from these atomic operations.
Concurrency Primitives
Atomic Update

Exercise for hardware geeks: atomic operations often lock an entire cache line, and can stall the CPU for hundreds of clock cycles while the caches synchronise, so they can slow you down more than you think. Read about this

Exercise for hardware geeks: compare the cost of using a lock against the cost of using an atomic update (the answer can depend on the pattern of access)

Exercise. Effective use of atomics involves understanding memory consistency orderings. Read about this

Exercise. Some programming languages offer atomic datatypes, e.g., Java, C++, Rust. These usually just call the machine instruction atomics. Read about this