Hardware and its Habits

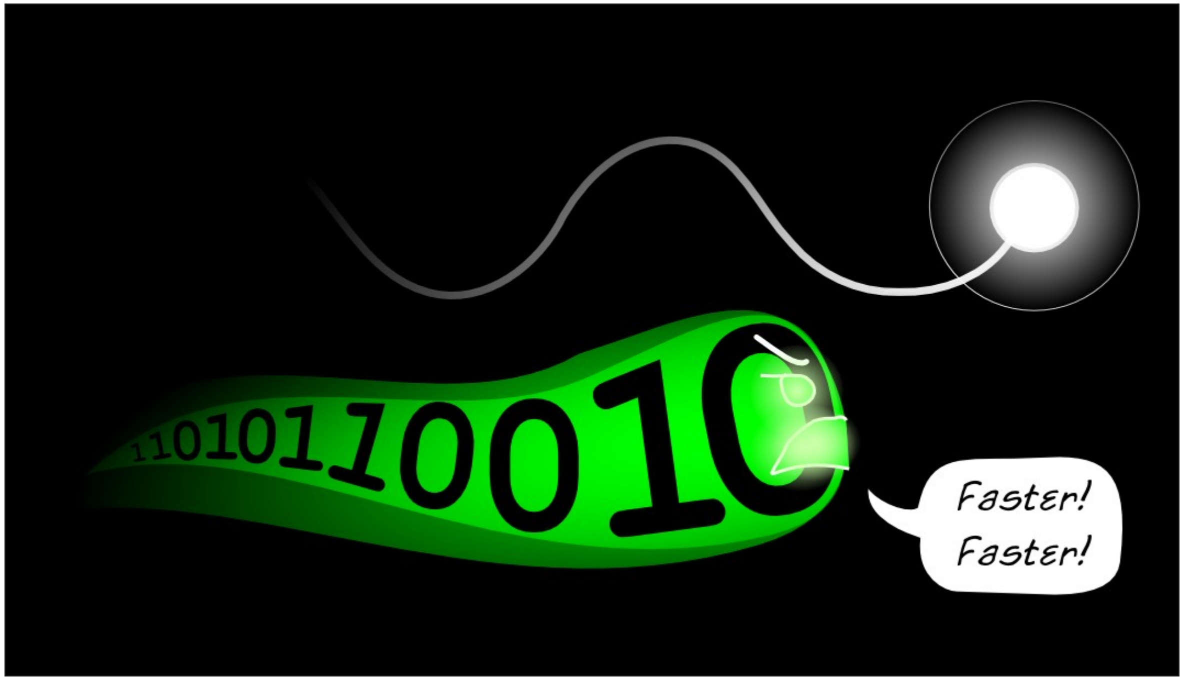
Paul E. McKenney
IBM Linux Technology Center

http://www.rdrop.com/users/paulmck
https://www.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html

Premature Abstraction Is The Root of All Evil!!!

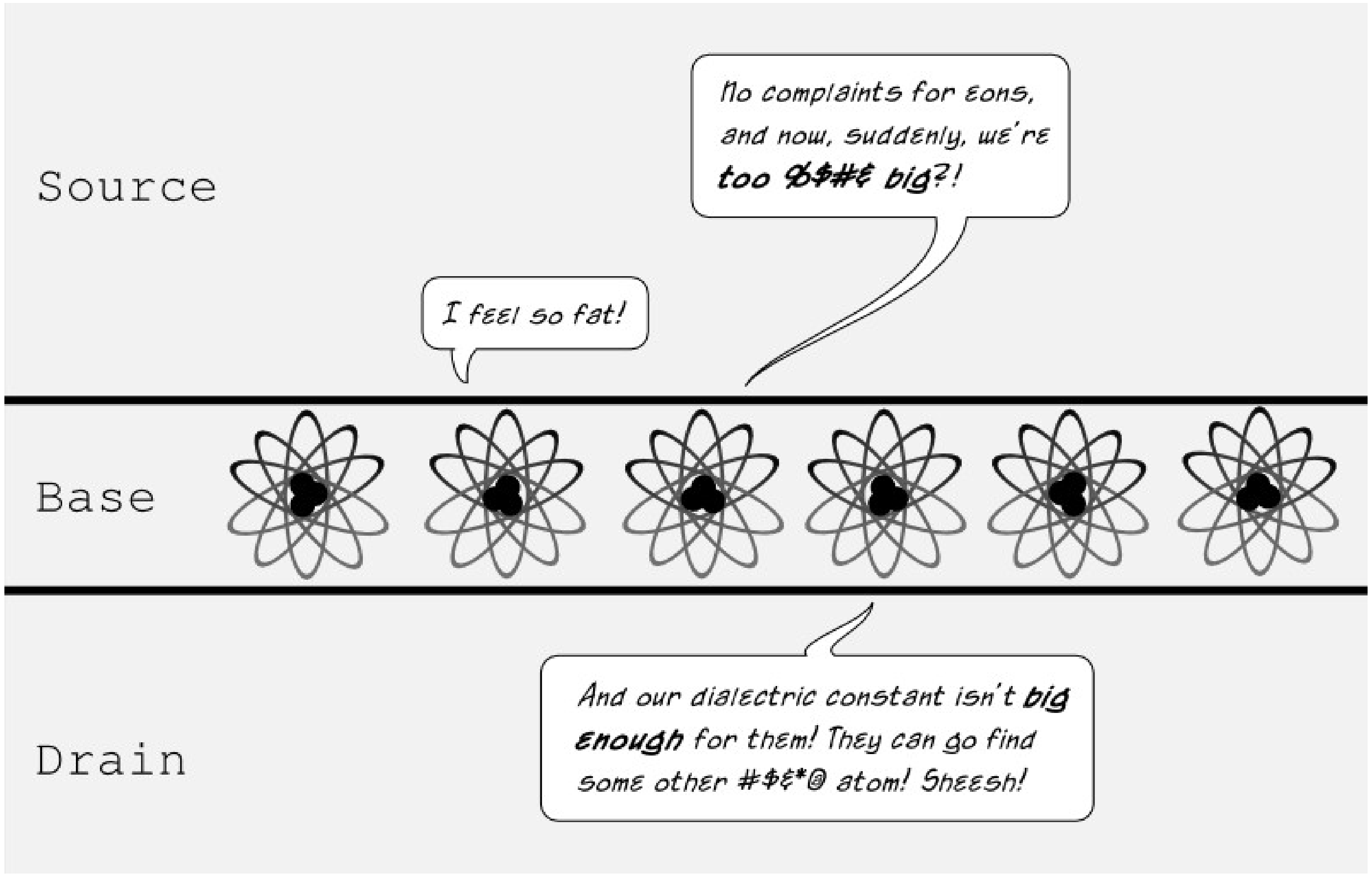
Premature Abstraction

Abstract away the finite speed of light???

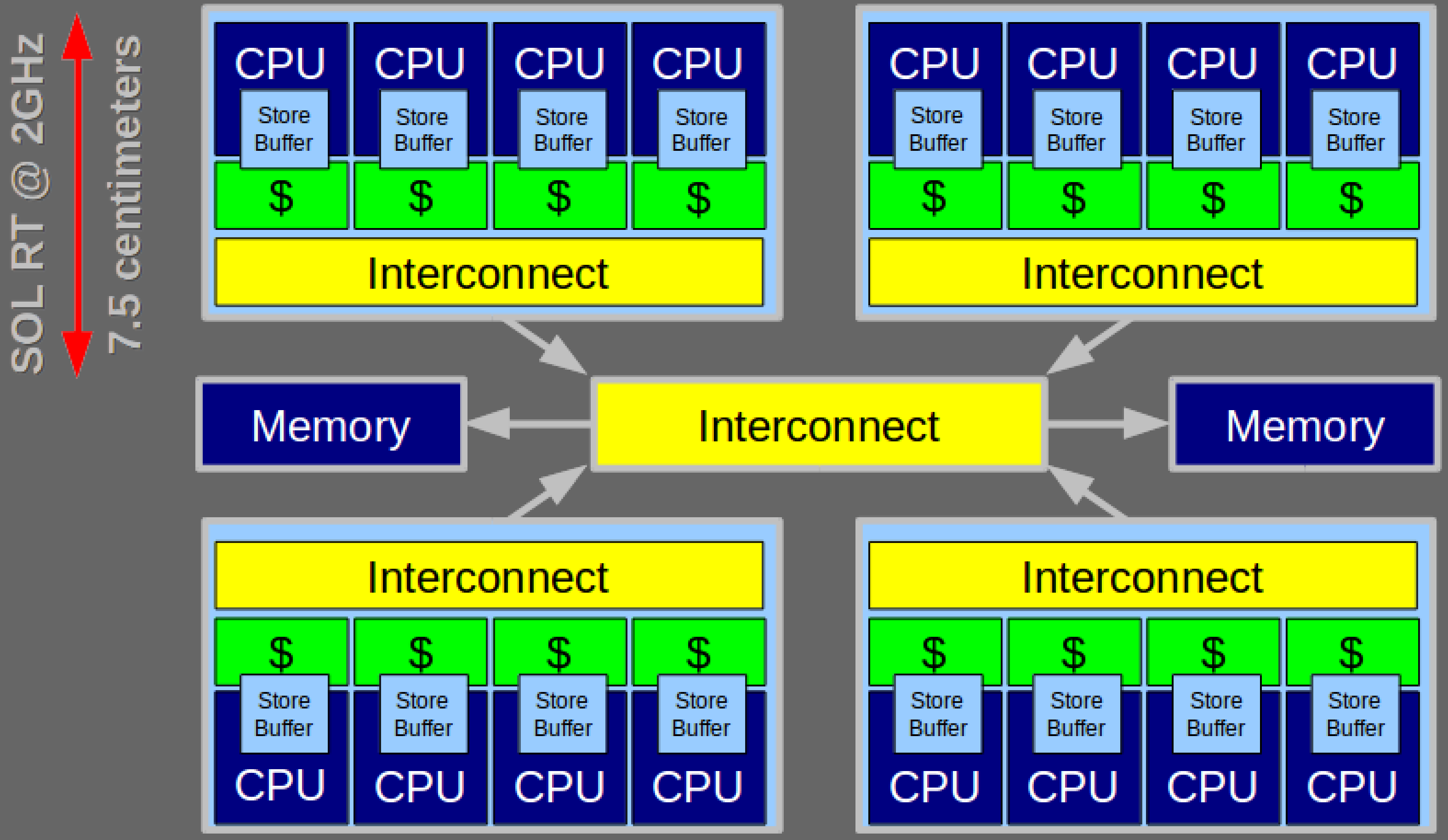


Premature Abstraction

Abstract away the atomic nature of matter???

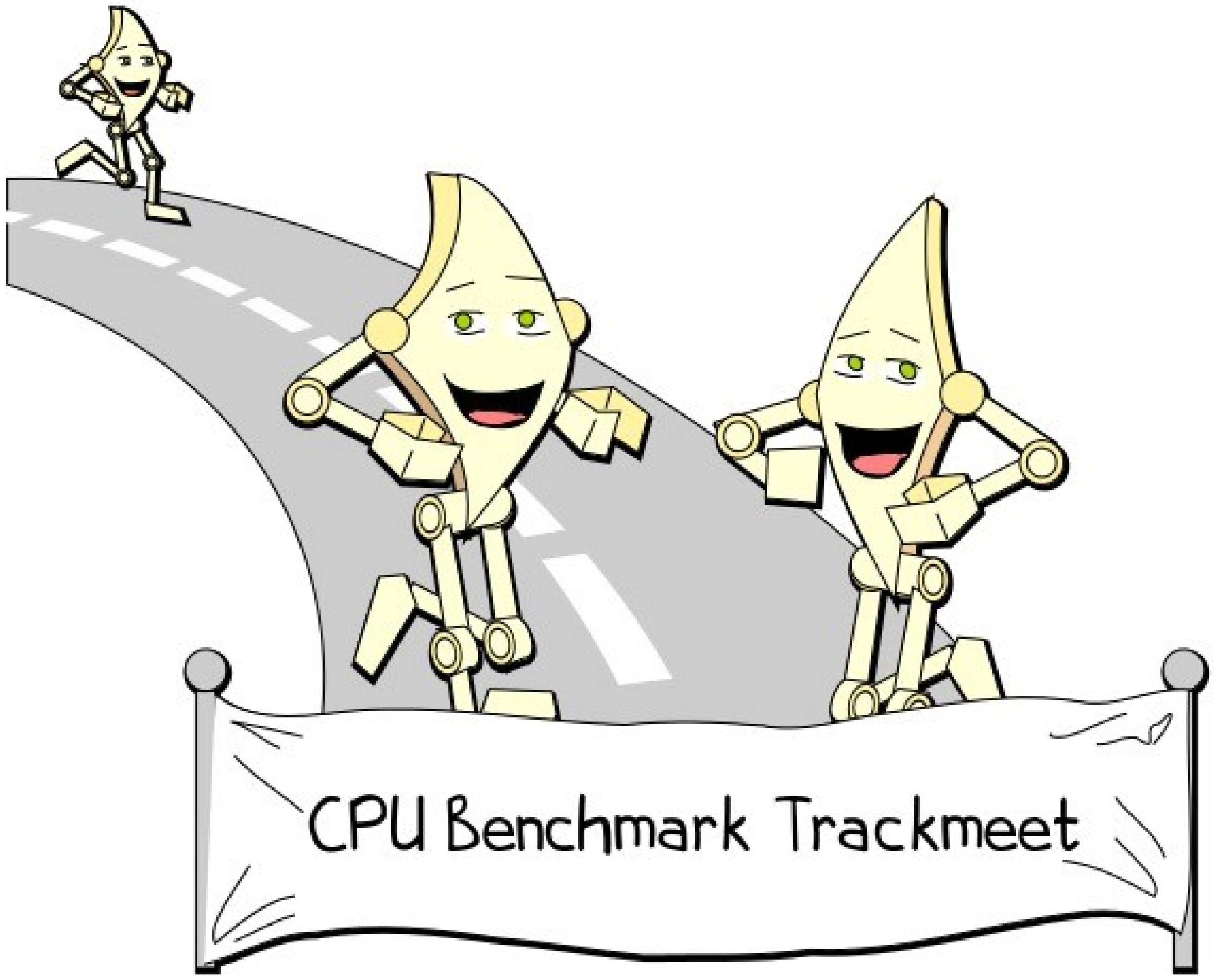

Source
Base
Drain
No complaints for eons, and now, suddenly, we're too %$#€ big?!
I feel so fat!
And our dialectric constant isn't big enough for them! They can go find some other #$€*@ atom! Sheesh!

Who Cares About Laws of Physics?

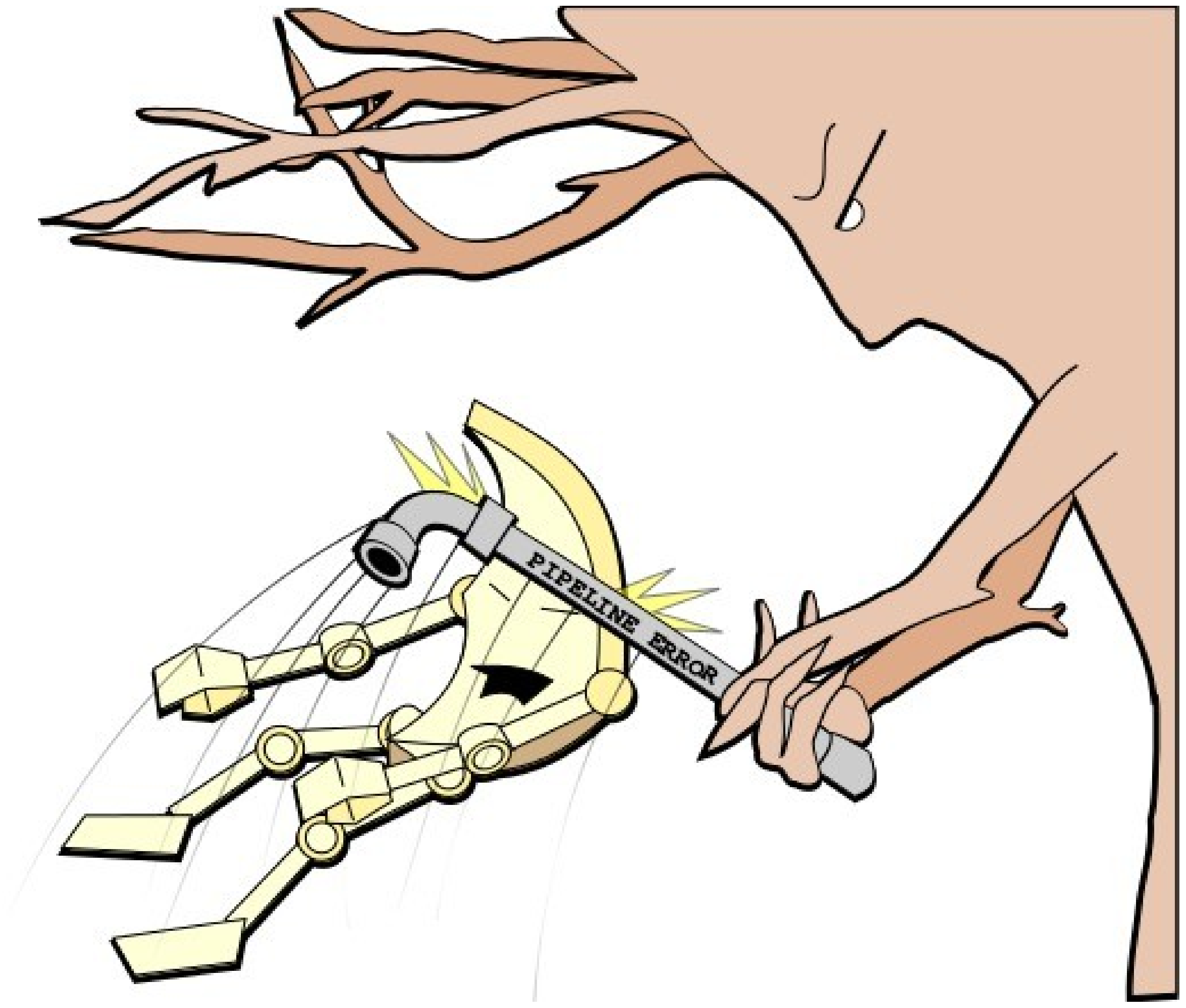


Electrons move at 0.03C to 0.3C in transistors and, so lots of waiting. 3D???

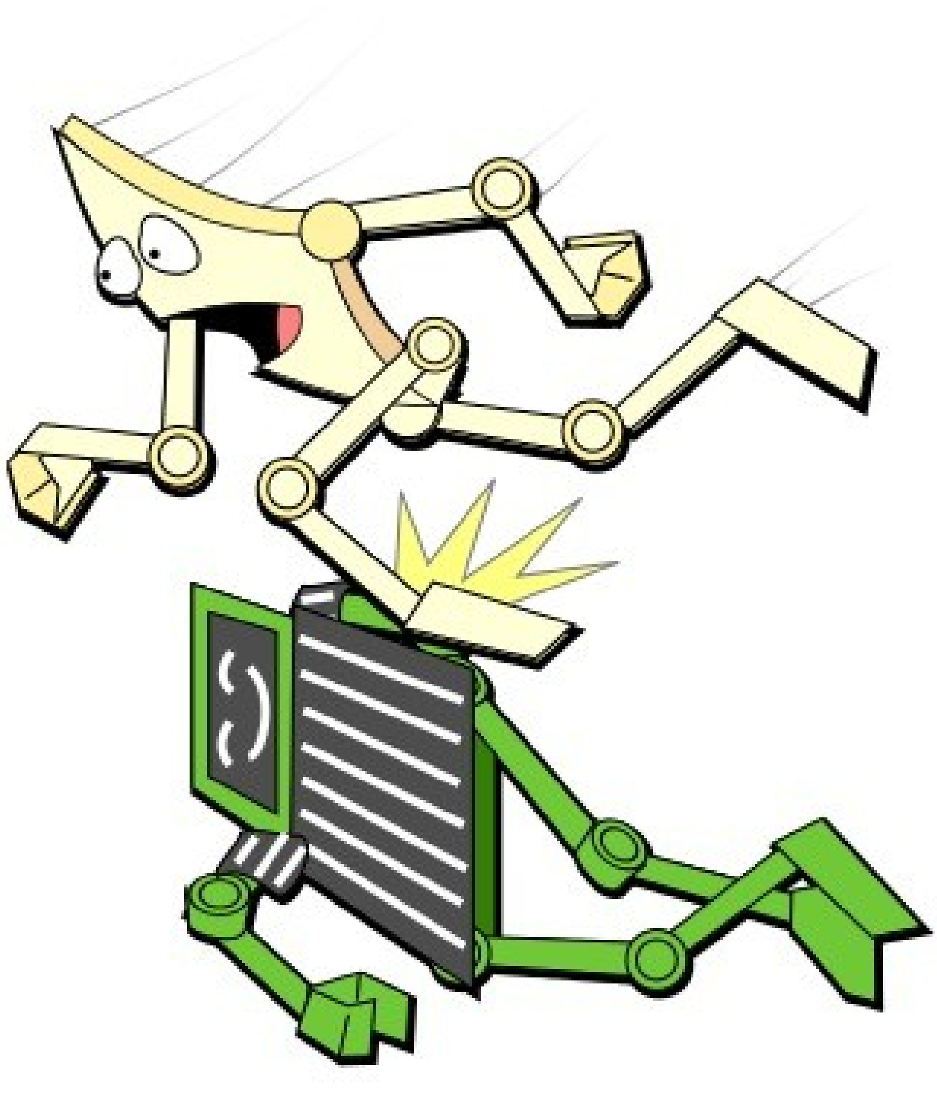
CPUs at Their Best



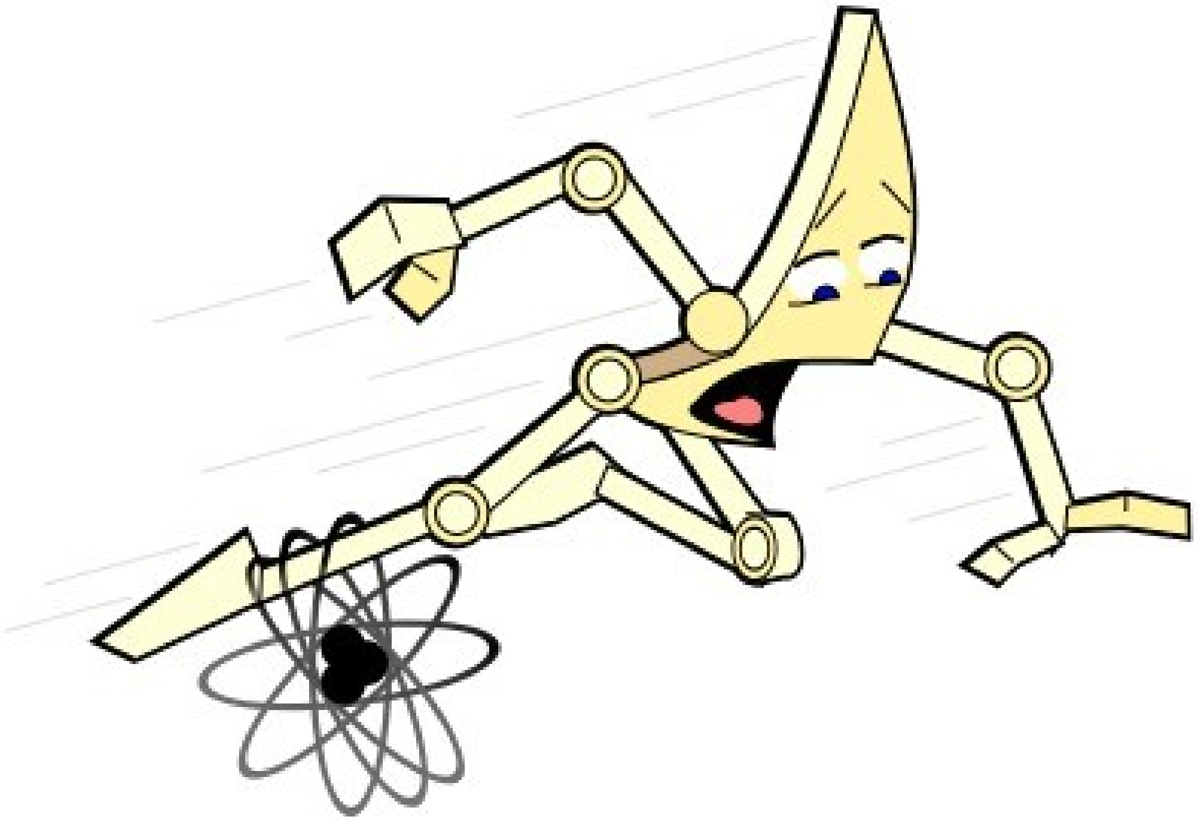
Mispredicted Branch



Memory Reference

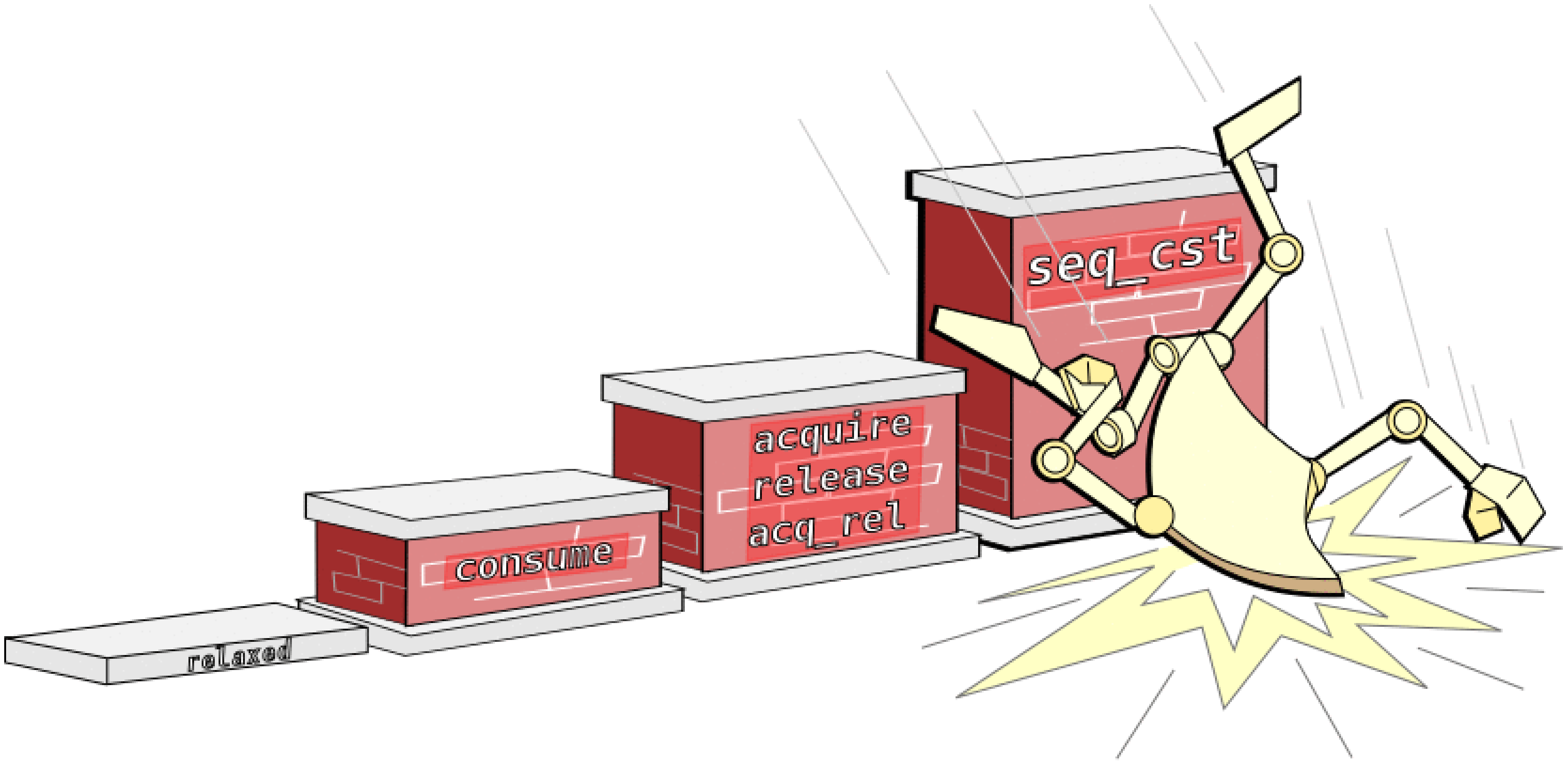


RMW Atomic Instruction



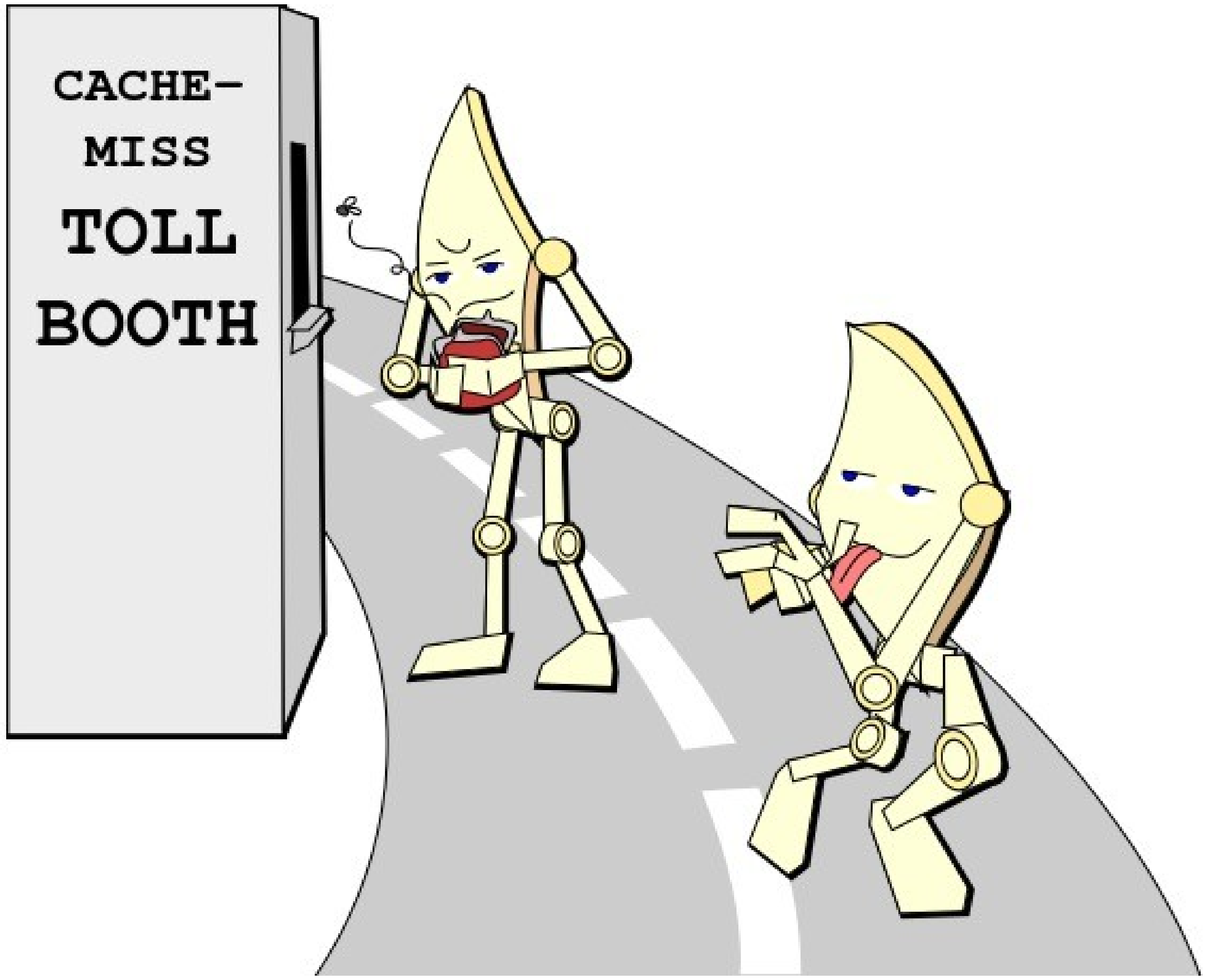
Memory Barrier (AKA Fence)

Memory Ordering Not Created Equal

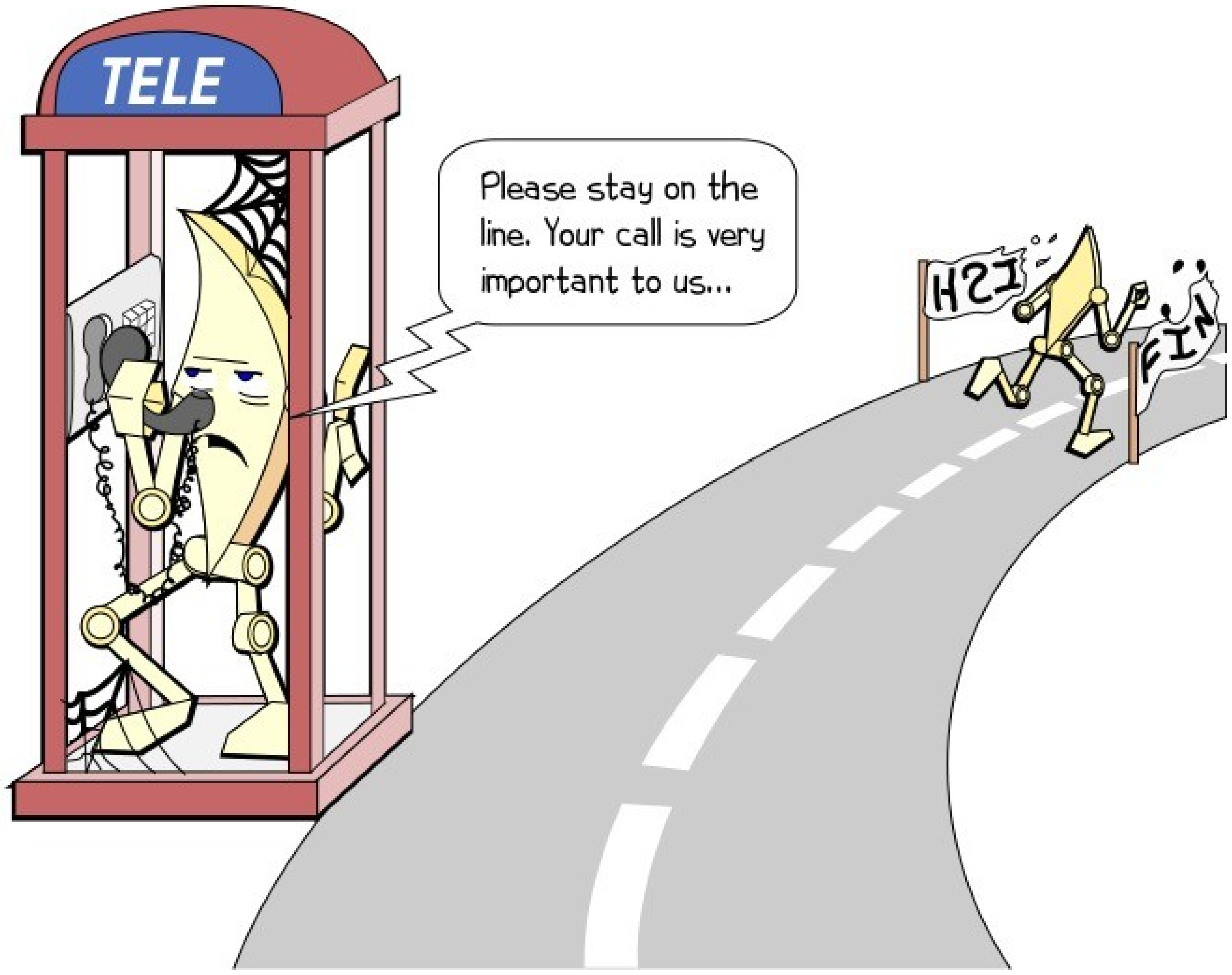


Exact relation depends strongly on hardware architecture and on compiler

Cache Miss

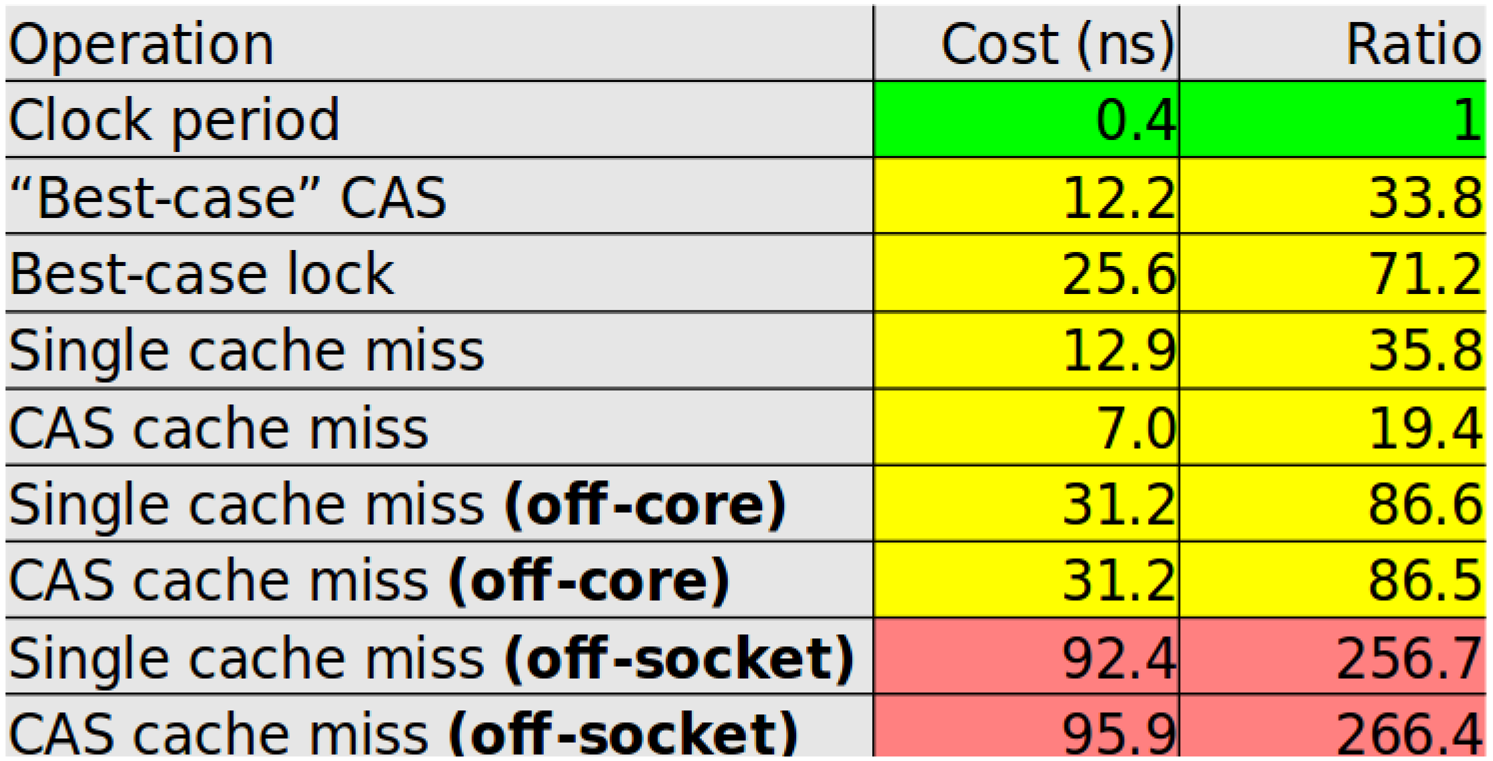


I/O Operation


TELE
Please stay on the line. Your call is very important to us...
HSI
FIN

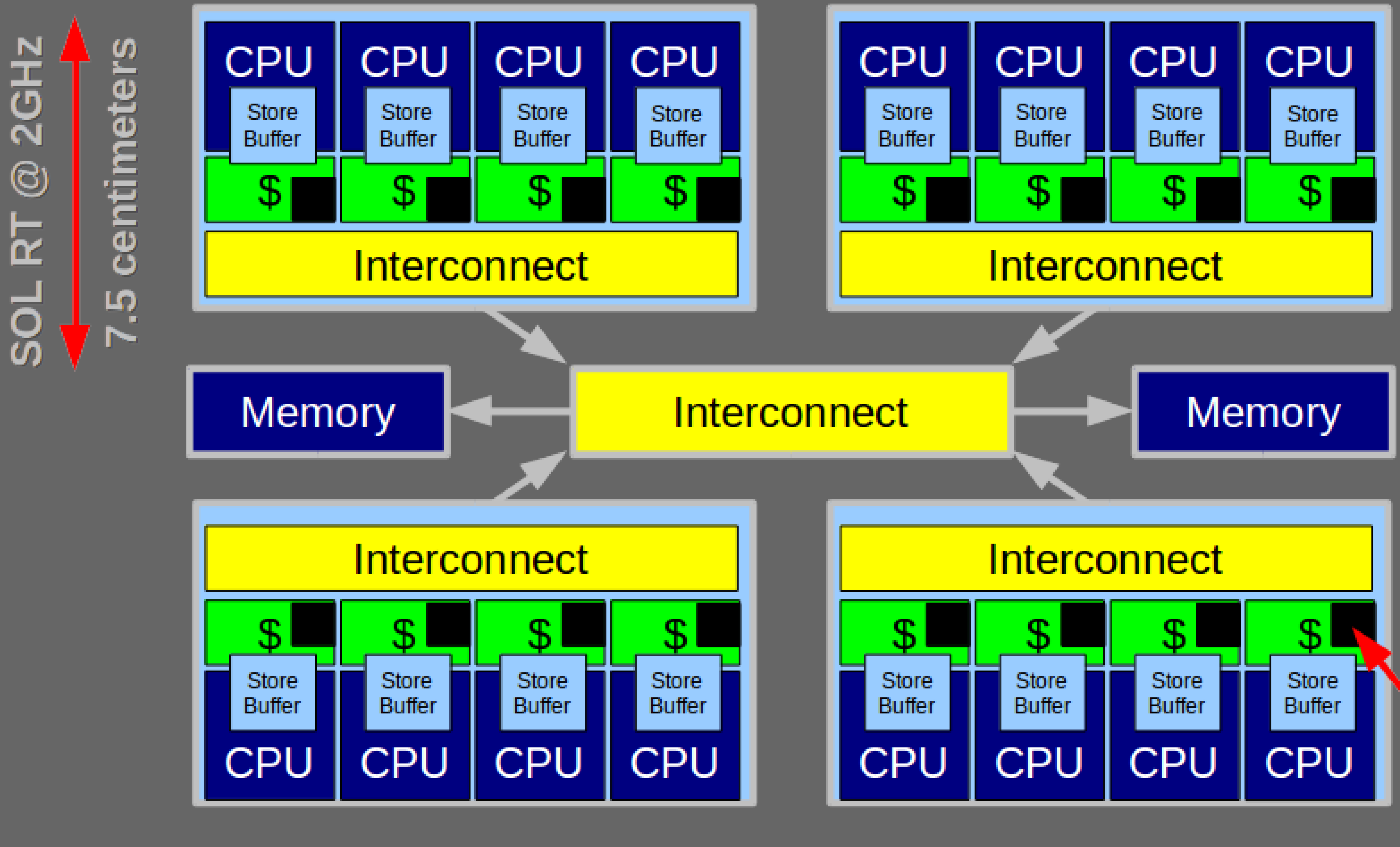
Costs of Operations

16-CPU 2.8GHz Intel X5550 (Nehalem) System



Operation	Cost (ns)	Ratio
Clock period	0.4	1
“Best-case” CAS	12.2	33.8
Best-case lock	25.6	71.2
Single cache miss	12.9	35.8
CAS cache miss	7.0	19.4
Single cache miss **(off-core)**	31.2	86.6
CAS cache miss **(off-core)**	31.2	86.5
Single cache miss **(off-socket)**	92.4	256.7
CAS cache miss **(off-socket)**	95.9	266.4

Loophole in Laws of Physics



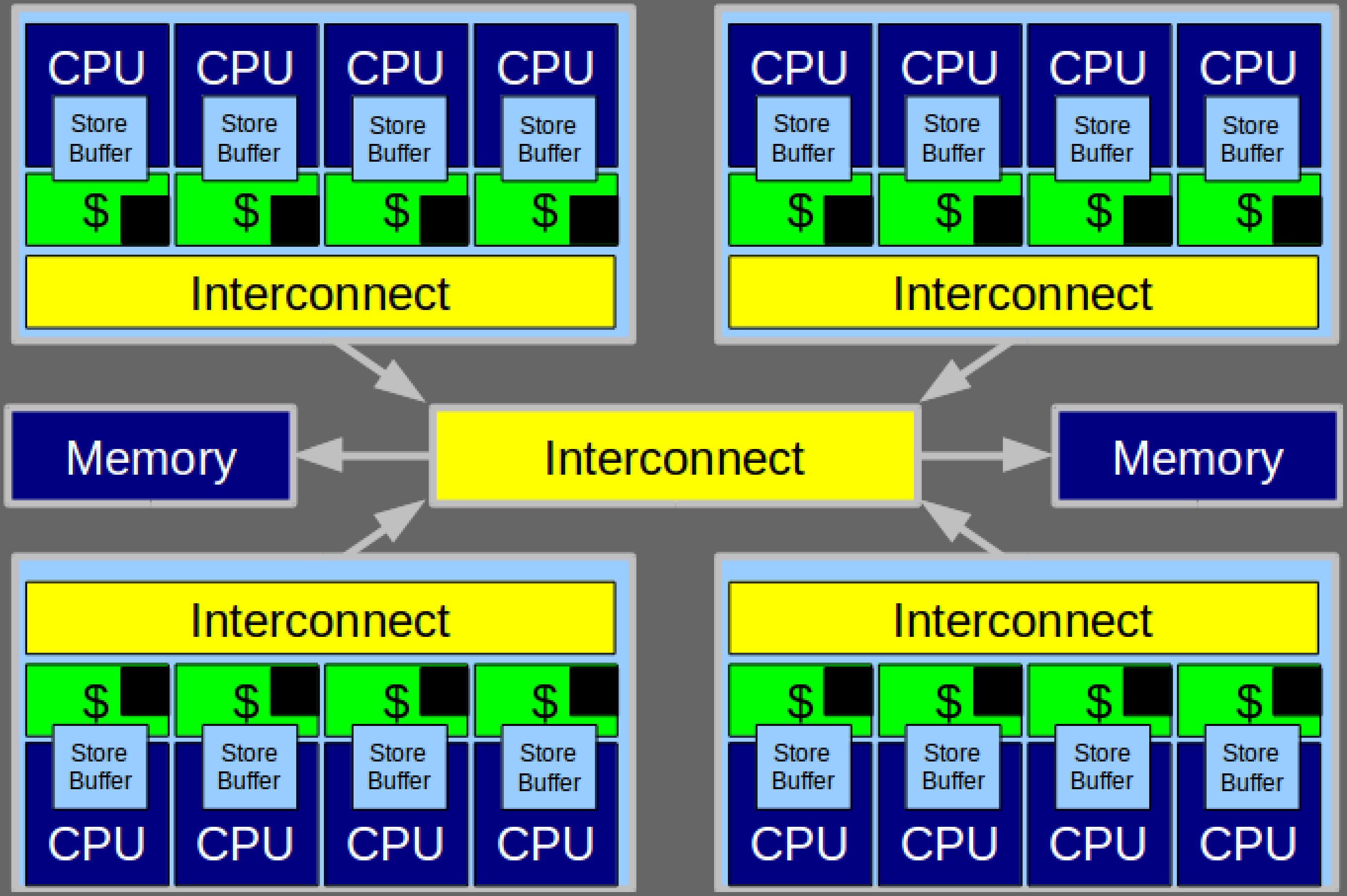
Read-only data remains replicated in all caches

In Real Estate:

Location, location, location!!!

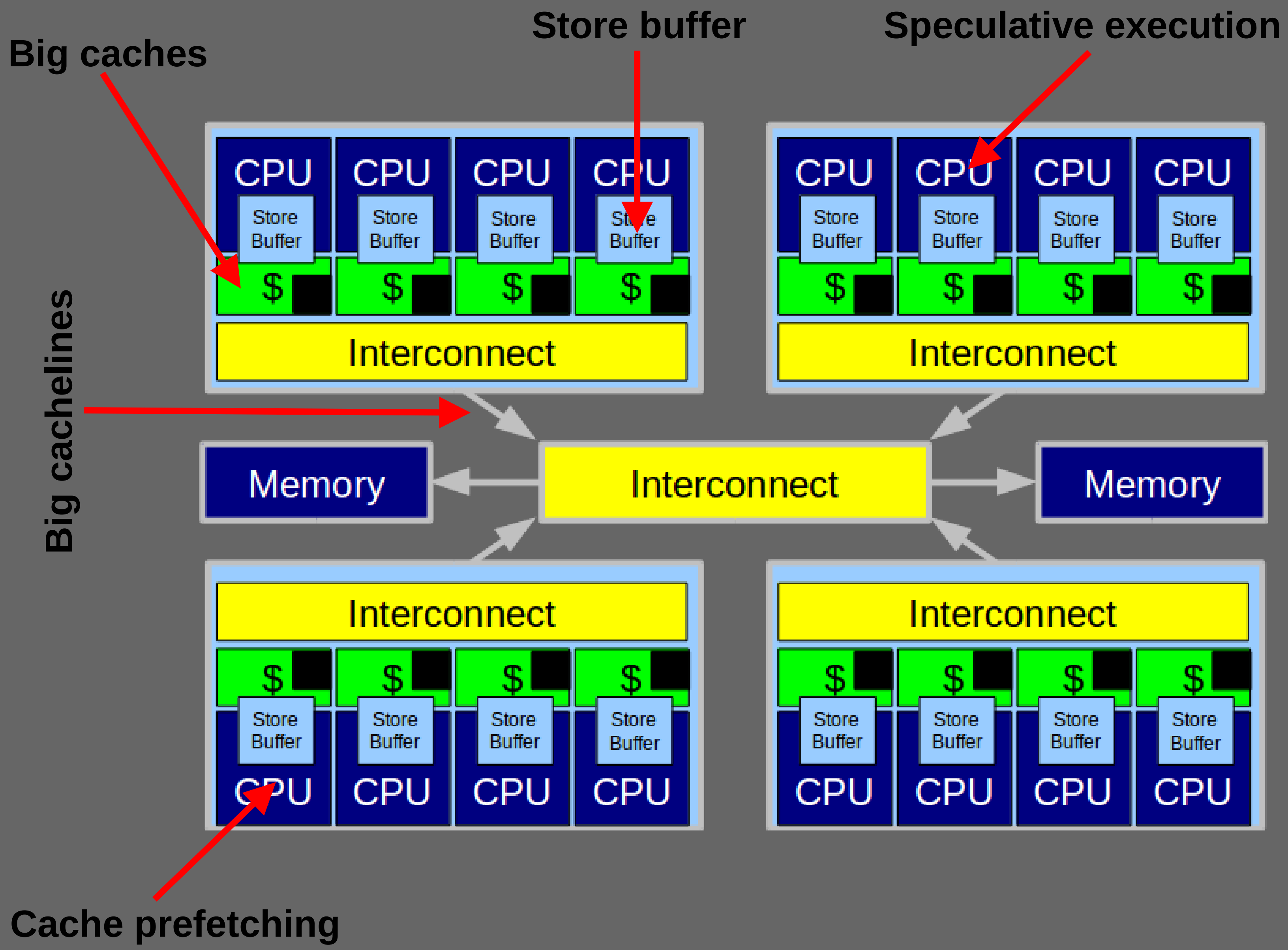
In Parallel Programming:

Locality, locality, locality!!!

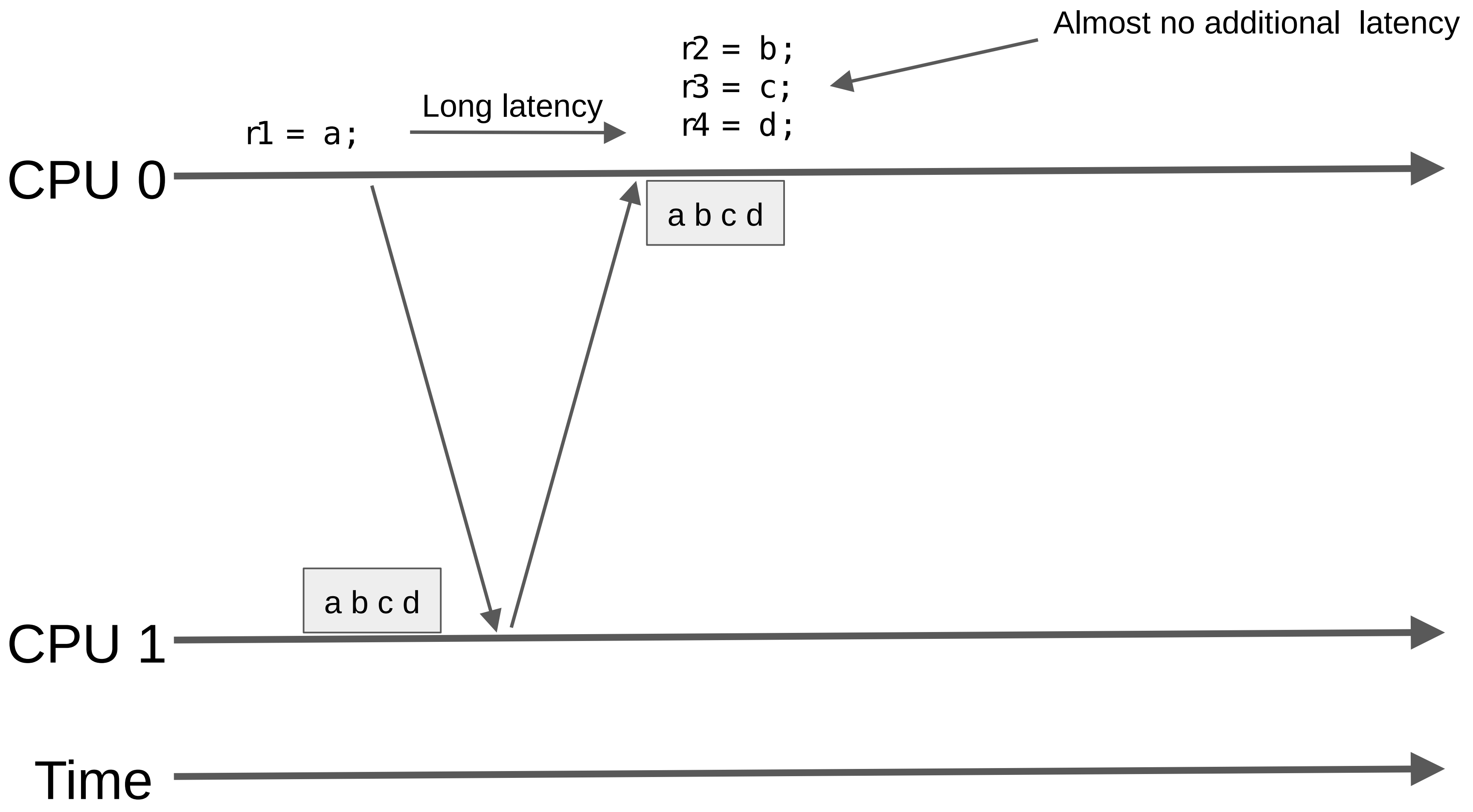


But Can’t The Hardware Help???

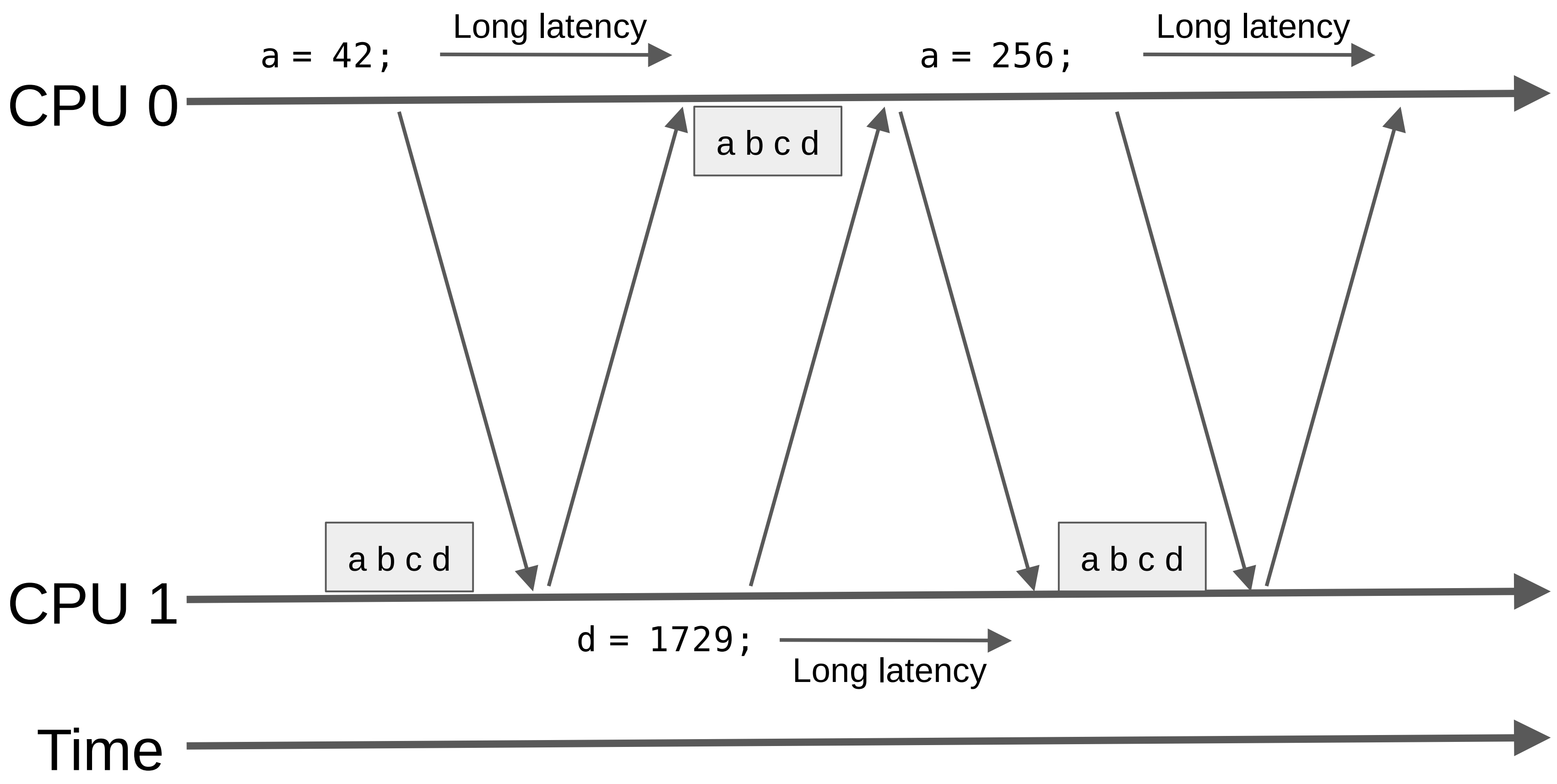
Hardware Locality Optimizations



Big Cachelines: The Good

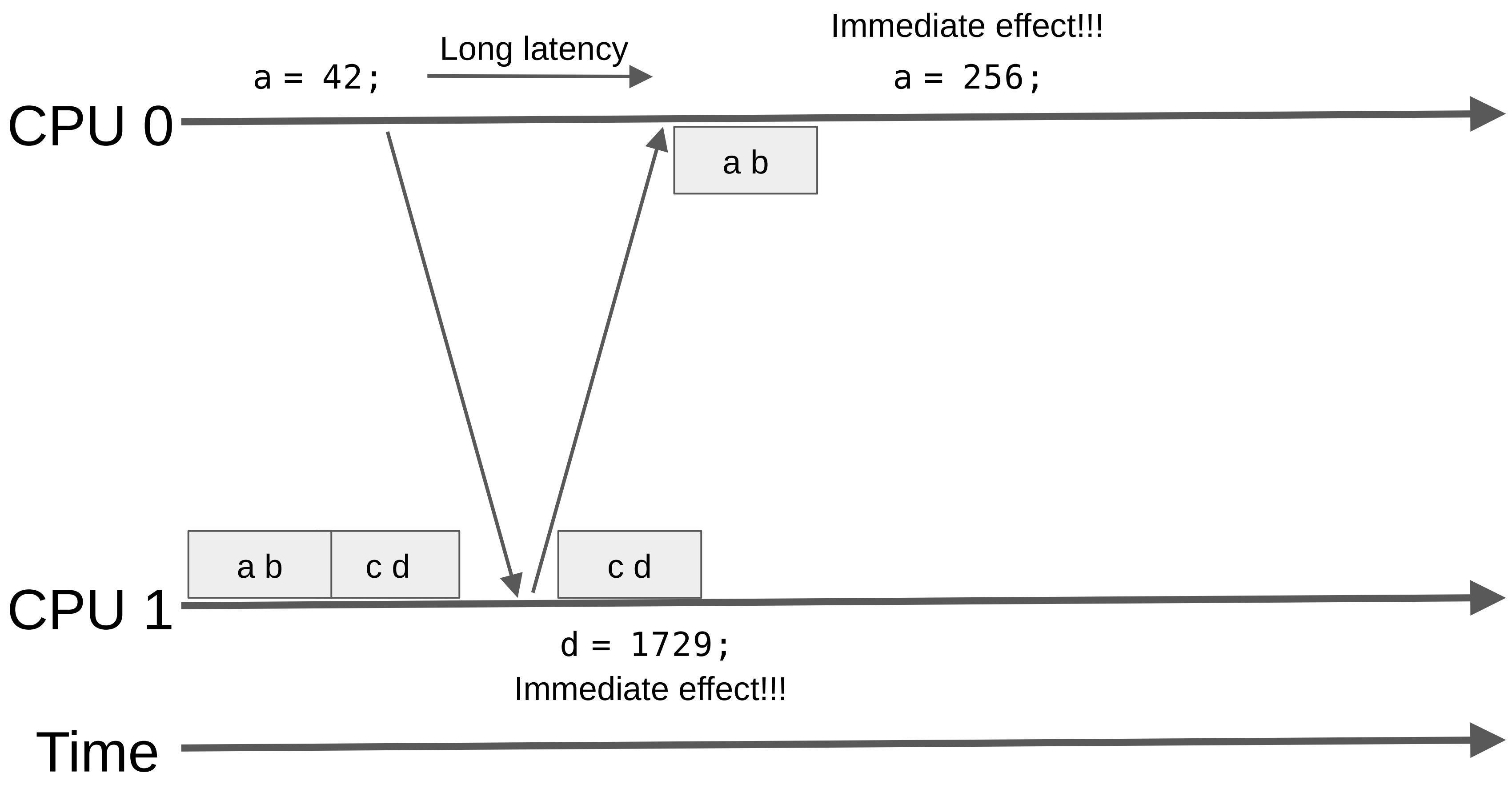


Big Cachelines: The Bad and Ugly

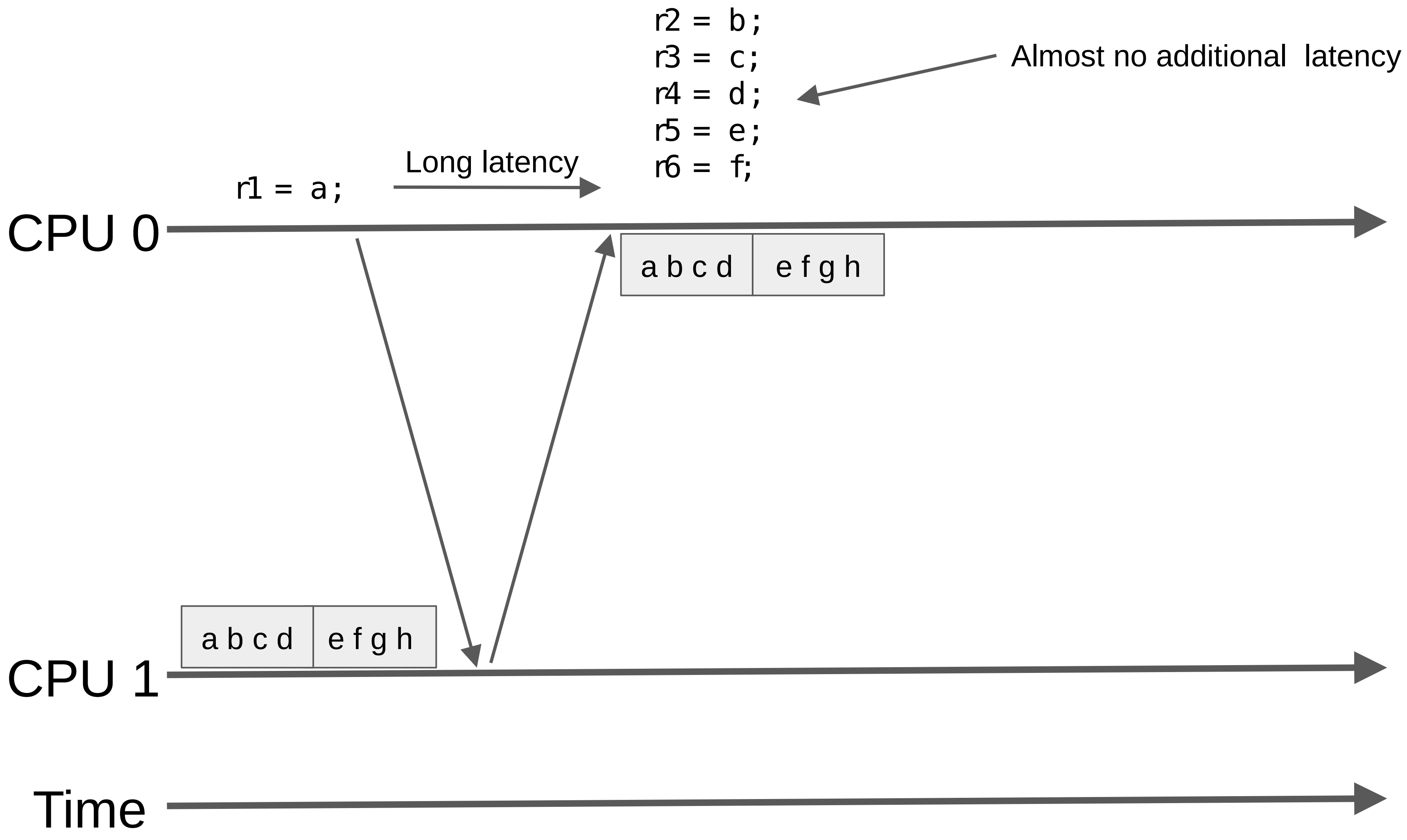


False sharing!!!

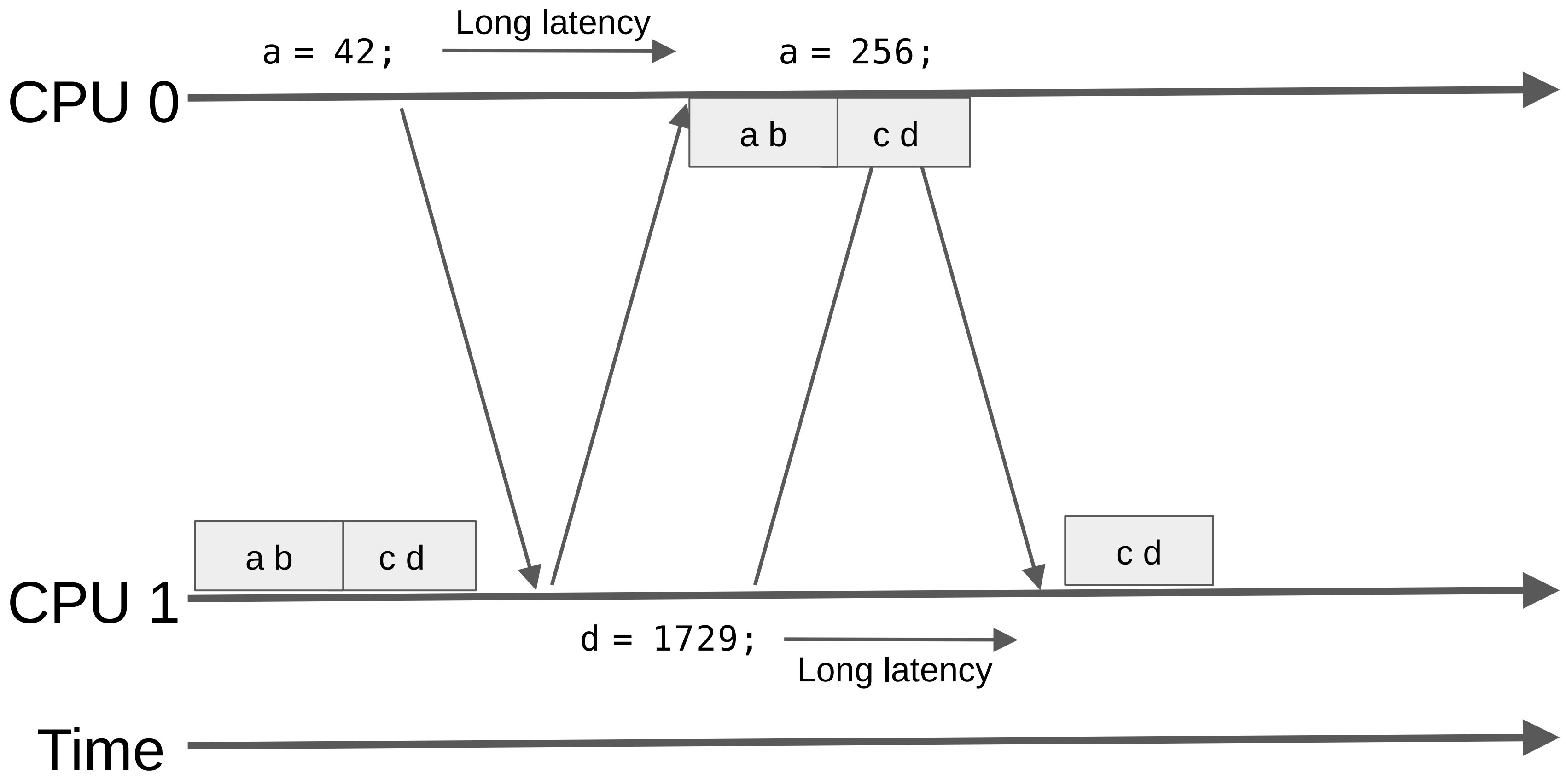
Big Cachelines: Alignment Directives



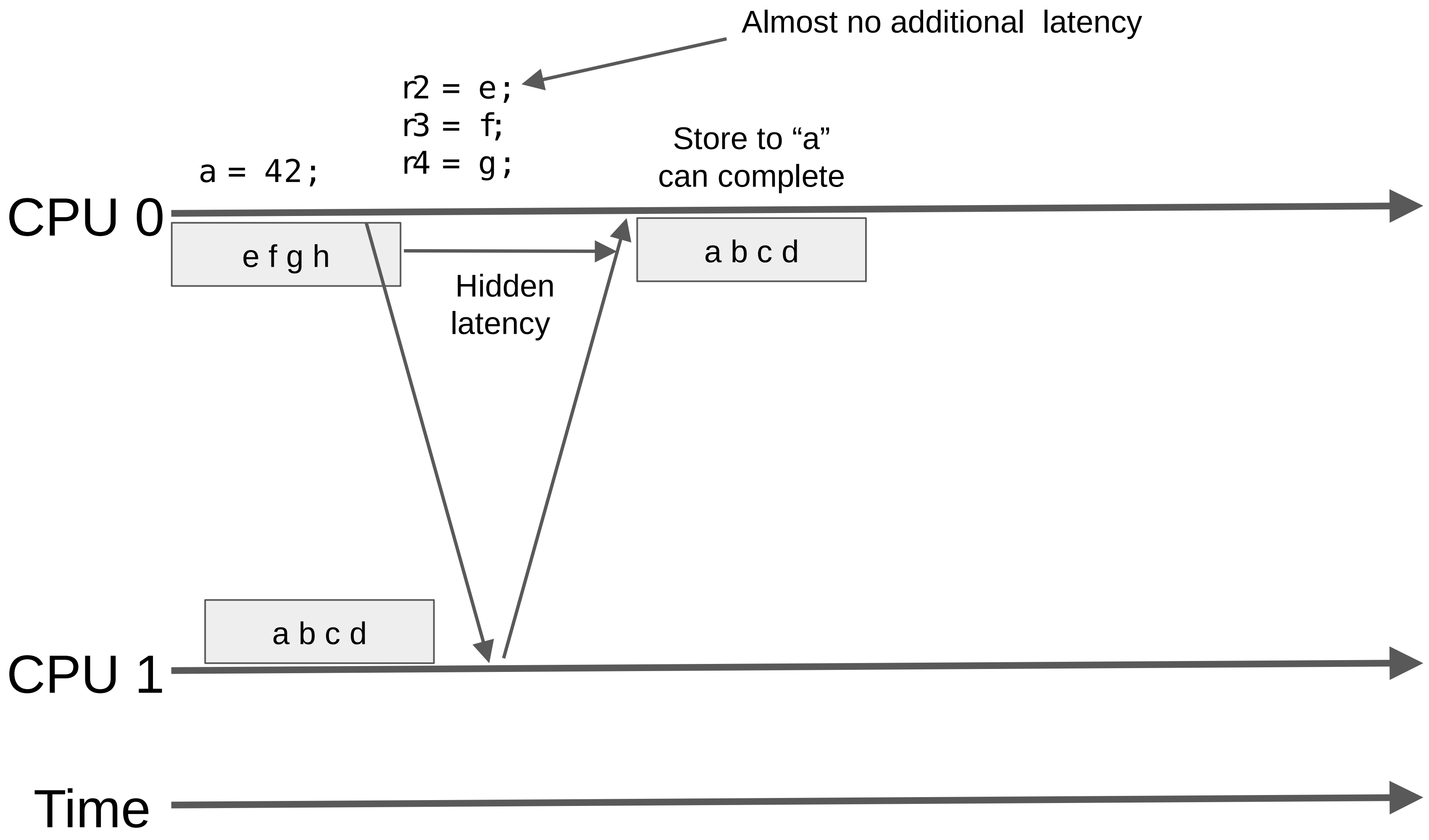
Prefetching Good!!!



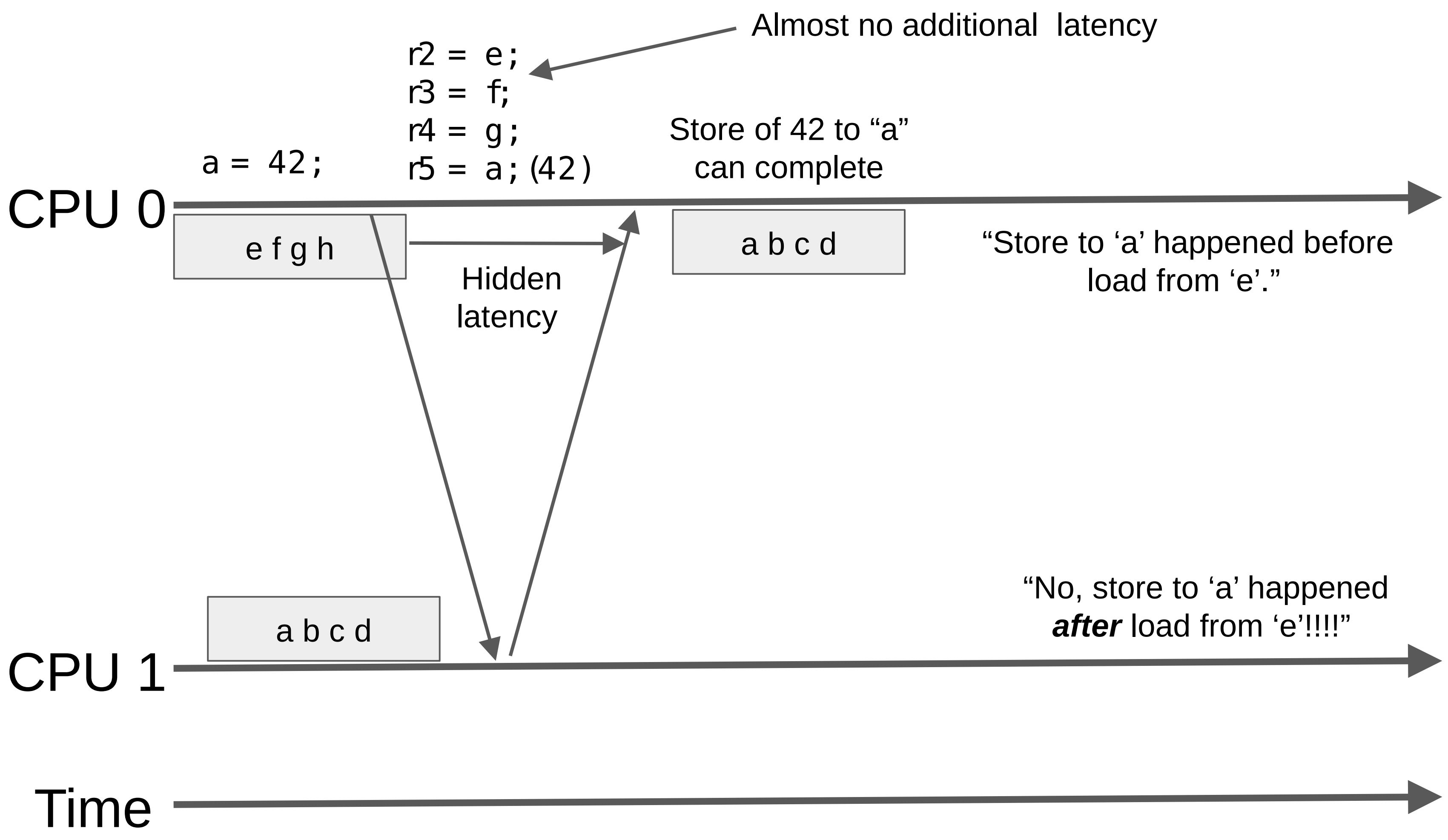
Prefetching: Bad and Ugly!



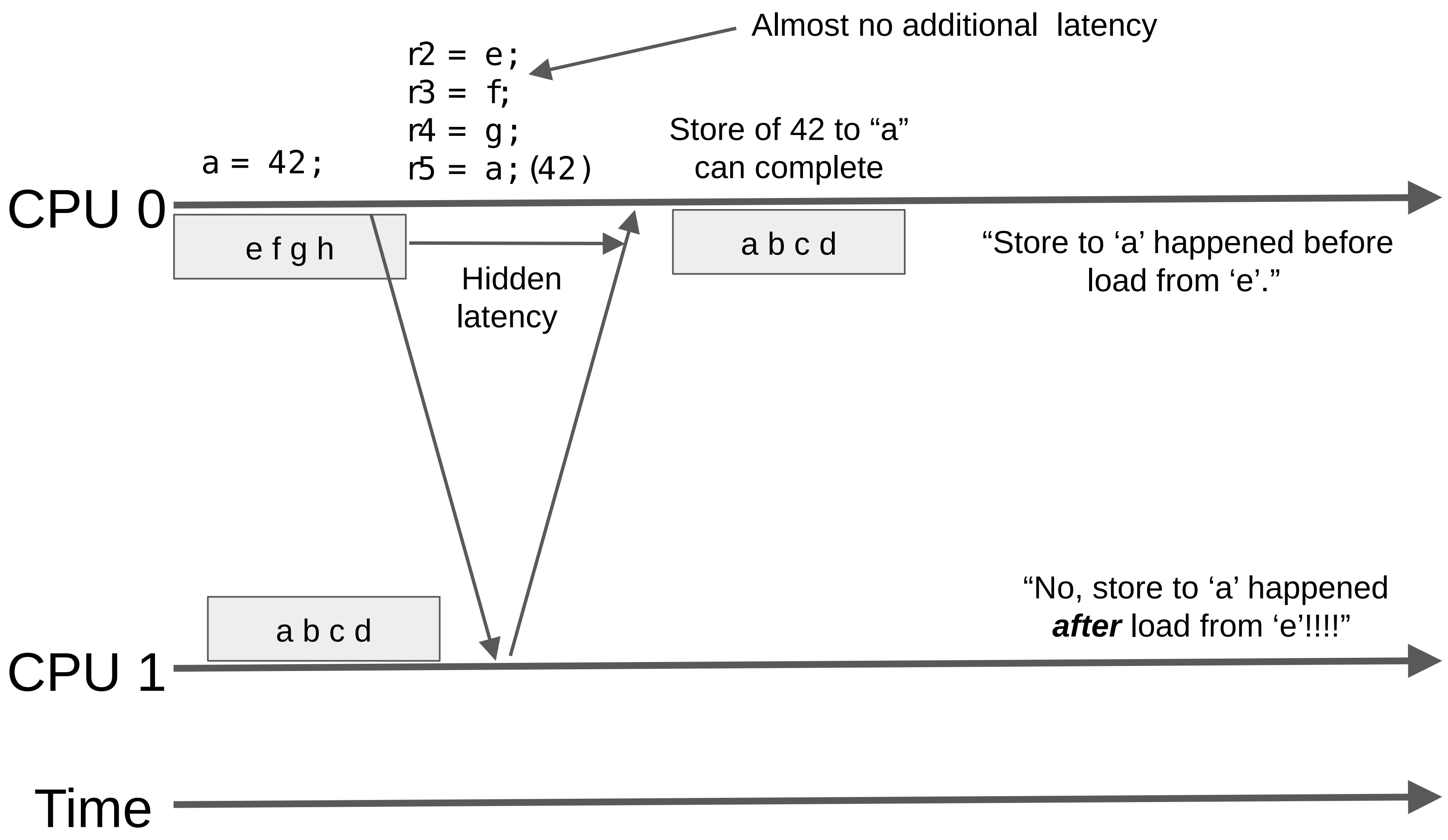
Store Buffers: The Good



Store Buffers: The Bad and Ugly

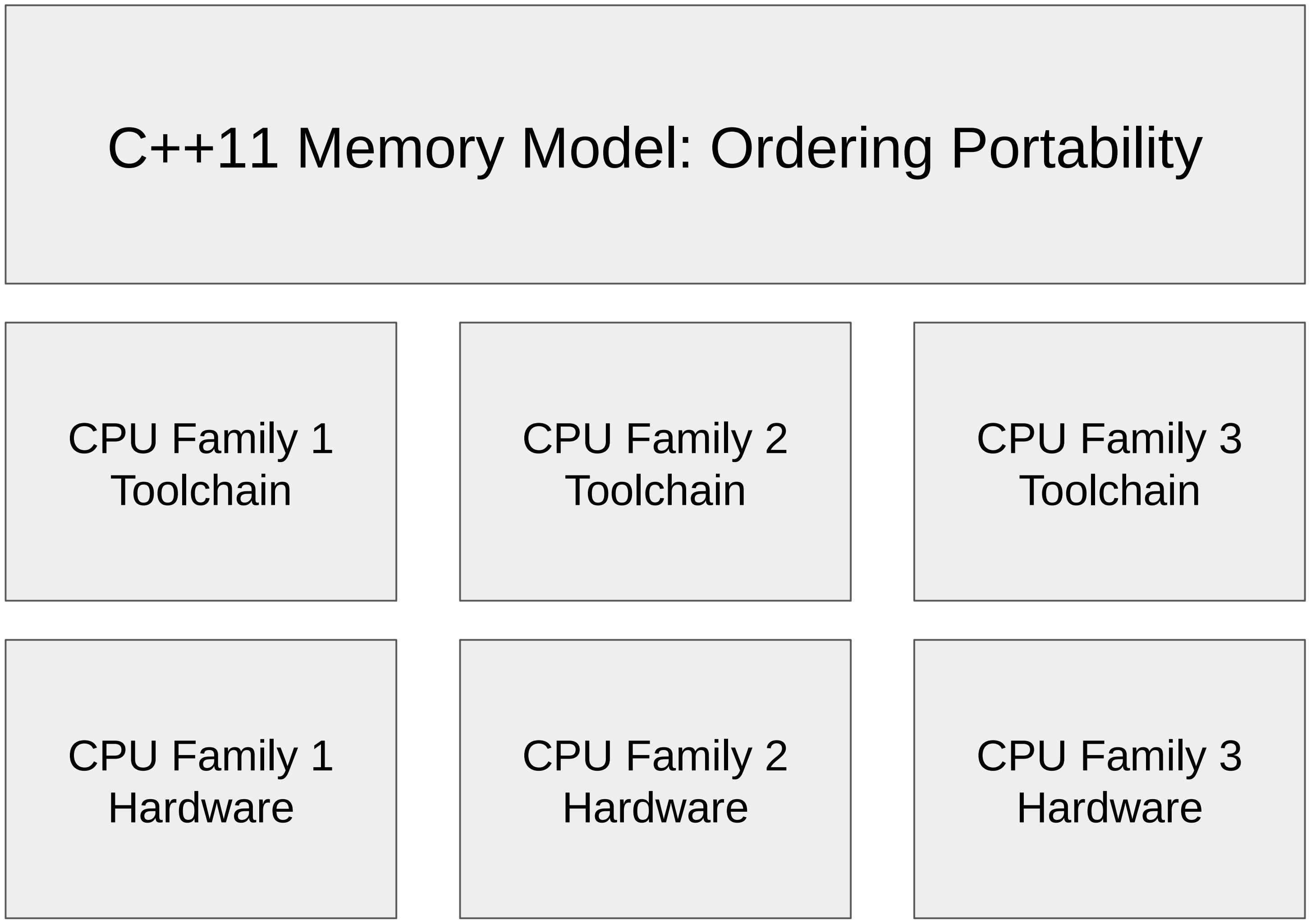

Almost no additional latency
r2 = e;
r3 = f;
r4 = g;
r5 = a; (42)
a = 42;
Store of 42 to “a” can complete
CPU 0
e f g h
a b c d
“Store to ‘a’ happened before load from ‘e’.”
Hidden latency
“No, store to ‘a’ happened ***after*** load from ‘e’!!!!”
a b c d
CPU 1
Time

Store Buffers: The Bad and Ugly


Almost no additional latency
r2 = e;
r3 = f;
r4 = g;
r5 = a; (42)
a = 42;
Store of 42 to "a" can complete
CPU 0
e f g h
a b c d
"Store to 'a' happened before load from 'e'."
Hidden latency
"No, store to 'a' happened ***after*** load from 'e'!!!!"
a b c d
CPU 1
Time

So we use ordering directives where needed!!!

Portable Ordering: Memory Model



Big Caches & Speculative Execution

Big caches:
- Higher throughput when data fits in cache
- Higher latency for cache misses
- Energy inefficiency

Speculative execution:
- Hide latencies and memory misordering
- Higher worst-case latency, energy consumption

Speculative Execution & Ordering

```
data = 42;
flag.store(1, mo_release)
```

```
while (!flag.load(mo_acquire))
    continue;
r1 = data;
```

Maligned Workhorse: Locking

Stupidly simple lock acquisition:

```
while (lk.exchange(memory_order_acquire))
    continue;
```

Lock release:

```
lk.store(0, memory_order_release);
```

Works, but **horrible** high-contention behavior

Maligned Workhorse: Locking

Somewhat less stupidly simple lock acquisition:

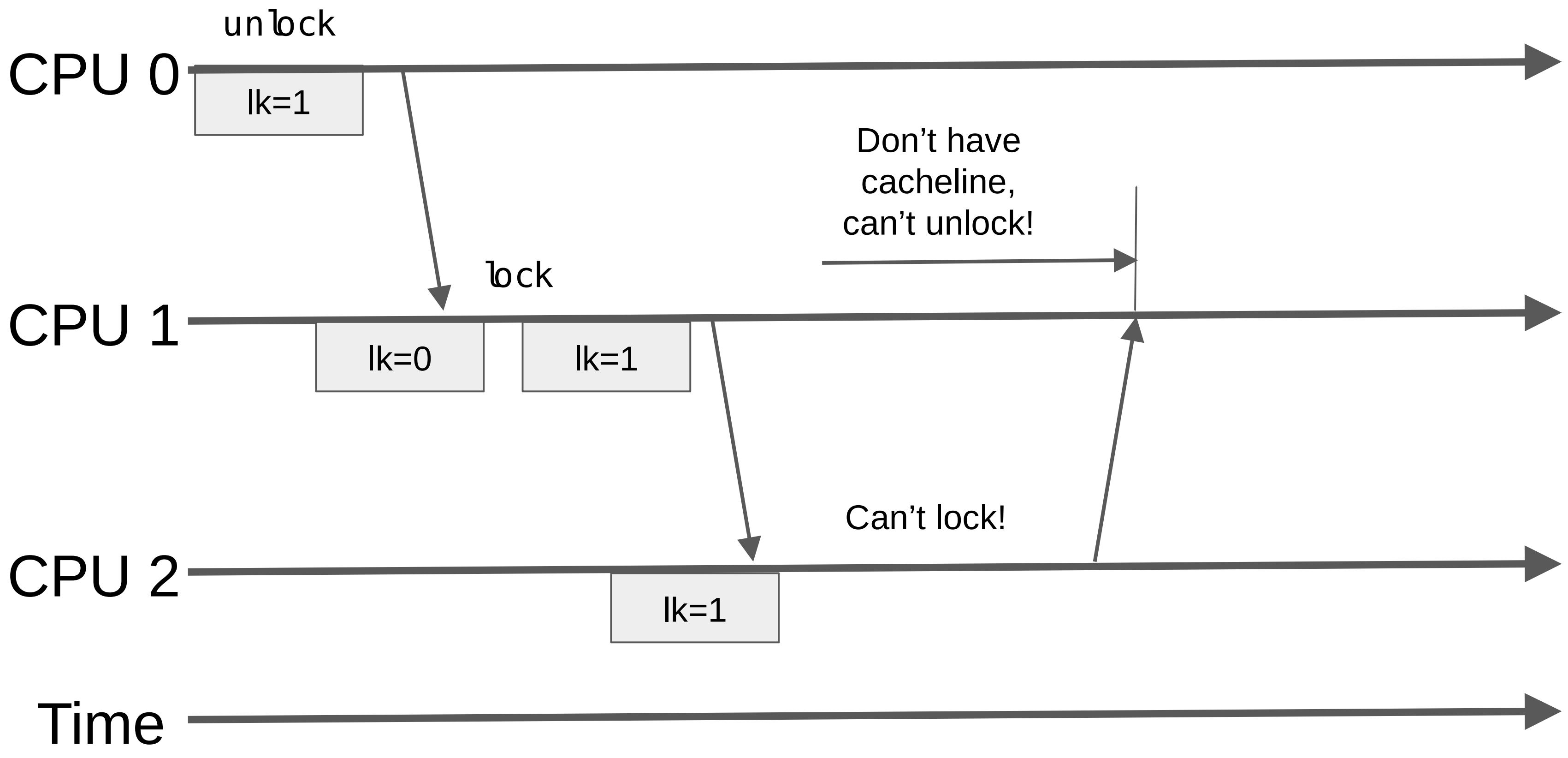
```
for (;;) {
   while (lk.exchange(1, memory_order_acquire))
      continue;
   if (!lk.load(memory_order_acquire))
      break;
}
```

Lock release:

```
lk.store(0, memory_order_release);
```

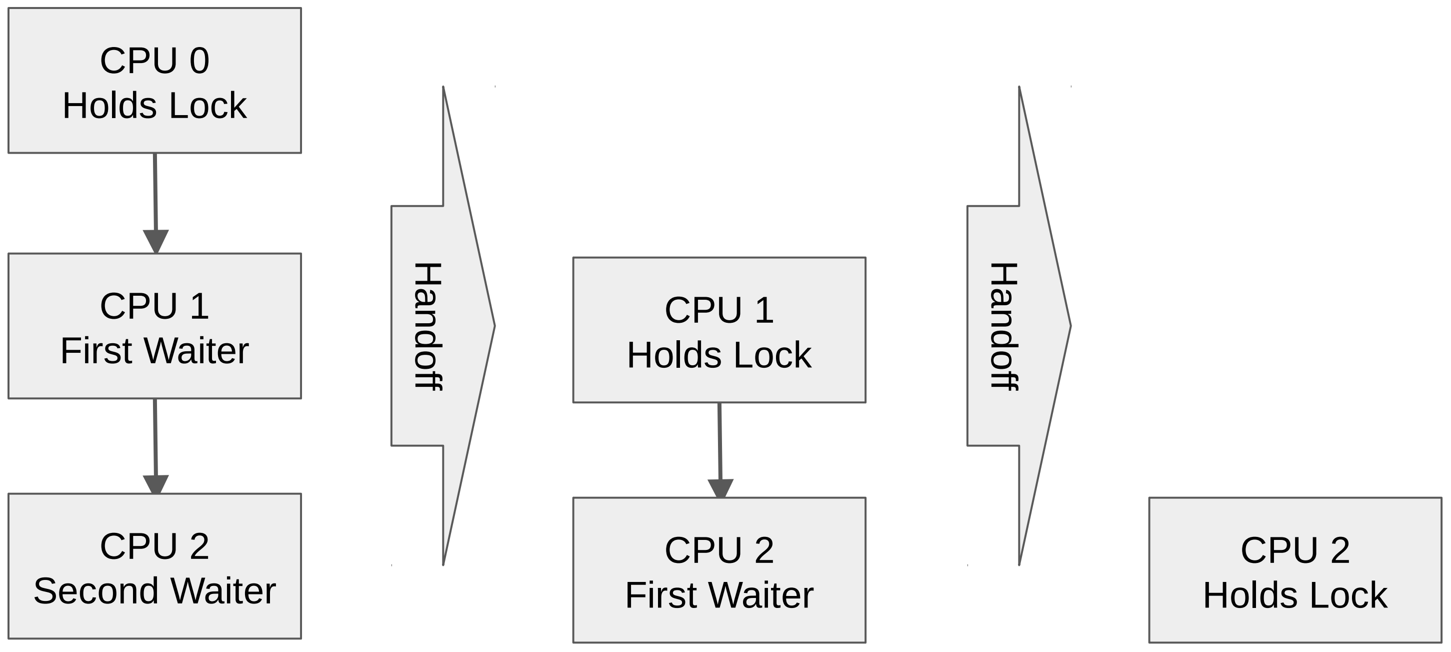
Works, but sub-optimal high-contention behavior

Suboptimal Lock Handoff



Lots of useless bus traffic after unlock time!!!
All CPUs spinning on same location...

Queued Locks



At most two CPUs content for given location

http://www.cs.rochester.edu/~scott/papers/1991_TOCS_synch.pdf

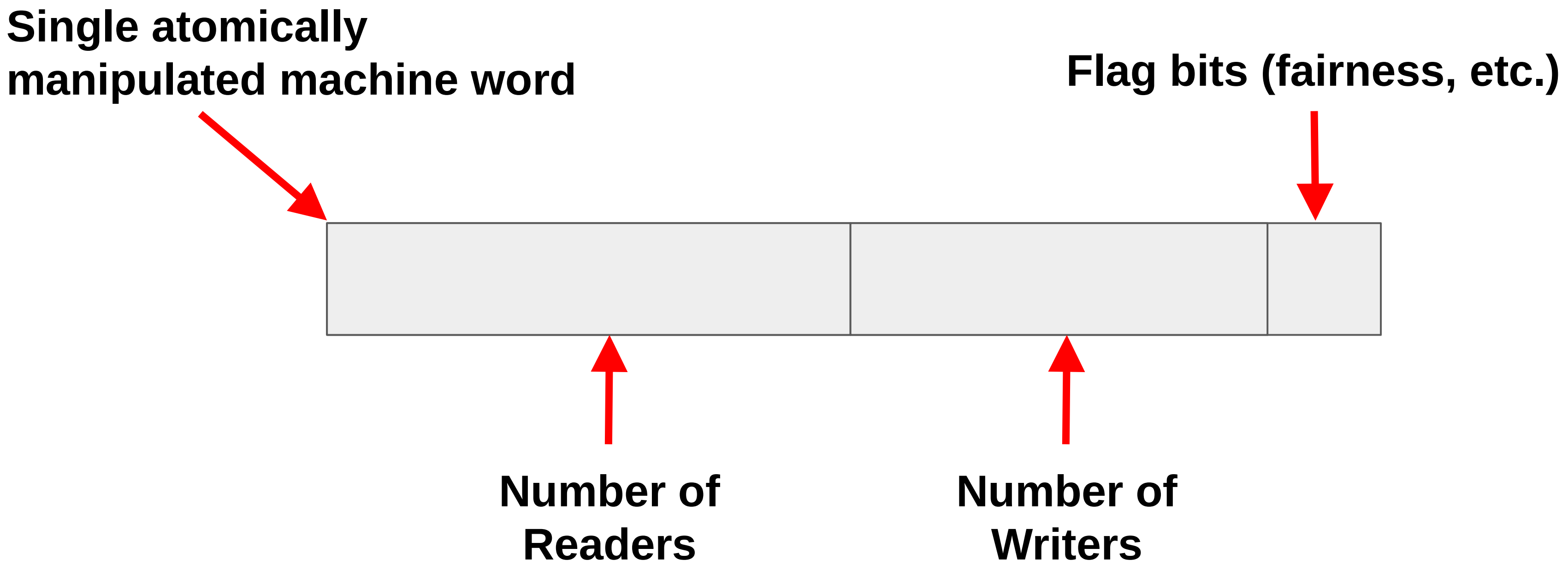
But Even Better...

Maintain low levels of contention!!!

Also, adaptive spin/sleep strategy
(for example, algorithms using futex
 system calls!)

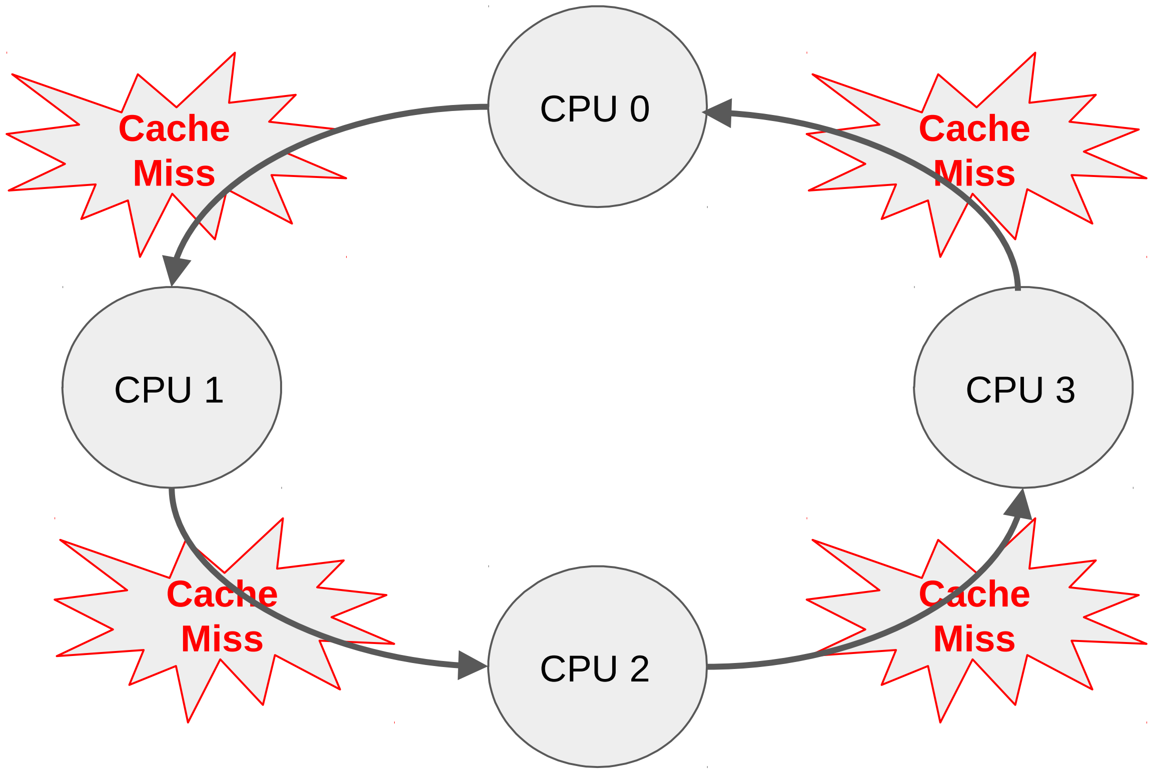
Reader-Writer Locking

Read-side parallelism! What not to like?
Textbook reader-writer lock:


Single atomically manipulated machine word
Flag bits (fairness, etc.)
Number of Readers
Number of Writers

Reader-Writer Locking

Textbook read-only acquisition and release:



Orders of magnitude slowdown from cache misses!!!
Might as well have an exclusive lock unless **huge** critical sections...

Software Must Help Hardware

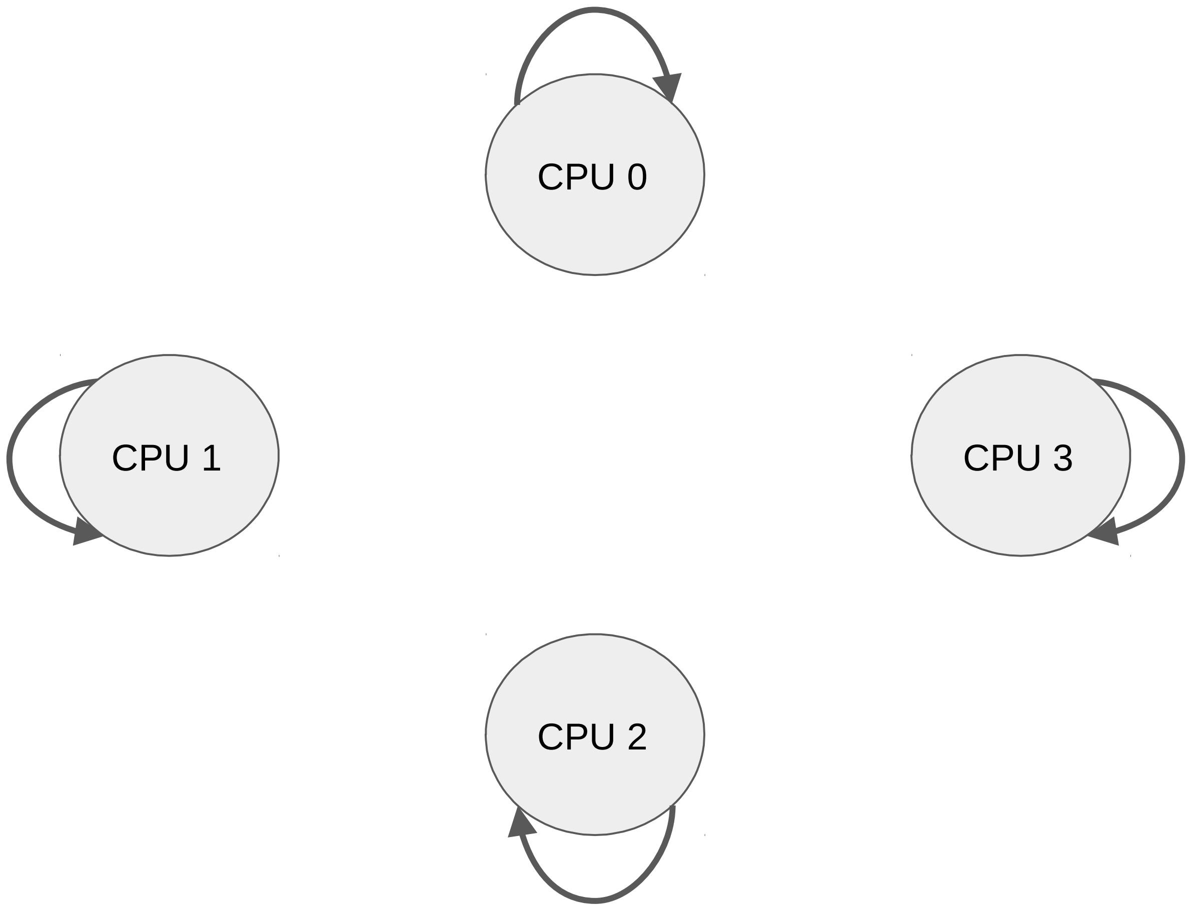
TLS Reader-Writer Lock

One exclusive lock per thread: TLS
Reading threads acquire and release own lock
Writing threads acquire and release **all** locks

Very fast readers, but very slow writers

TLS Reader-Writer Lock

TLS read-only acquisition and release:



Very fast readers!!!
Too bad about those poor writers... Generality!!!

Reader-Writer Locking

And now you understand one motivation for Paul’s RCU and Maged’s Hazard Pointers...

But that is another presentation.

Now for Maged's single-producer single-consumer buffer!!!