Cache Me Maybe

Using Caches to Improve Performance in Production Code!

MICHELLE FAE D'SOUZA

Cppcon
The C++ Conference

2025

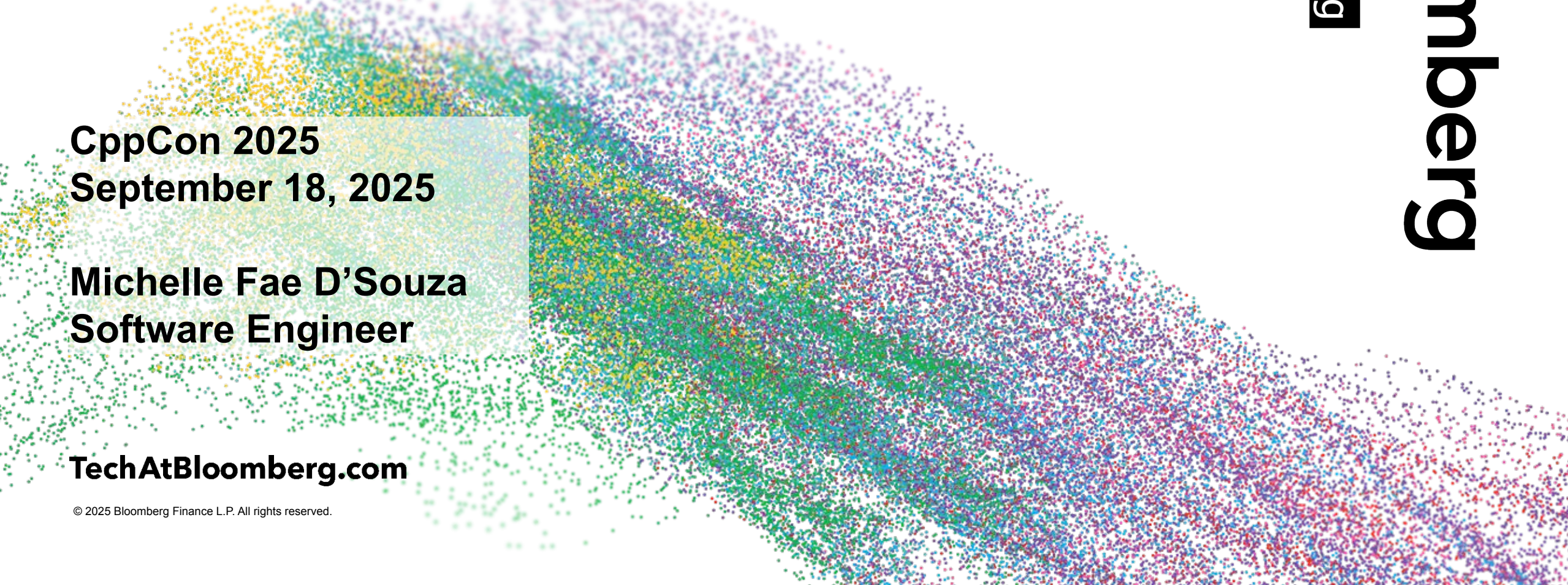
September 13 - 19

Cache Me Maybe:

Using Caches to improve Performance in Production Code!

Bloomberg

Engineering



CppCon 2025
September 18, 2025

Michelle Fae D'Souza
Software Engineer

TechAtBloomberg.com

Inspired by a real 10x speedup

Chris Cotter commented on May 24, 2024 Member • • •

Do we even need to maintain the unordered_map at all? #8270 runs 10x faster by only using the vector.

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Basic ~~CompSci Facts~~
TRUST ISSUES

For any specific operation

Time complexity of a vector ≥ Time complexity of an unordered map

TechAtBloomberg.com

Bloomberg

Engineering

✨DetectiveD’Souza

🕵️ Undercover Detective

👥 No case is ever cracked alone

Have something to share?
I’d love to hear at the end (just for the sake of time)!

TechAtBloomberg.com

Back on the Case

Chris Cotter commented on May 24, 2024 Member • • •

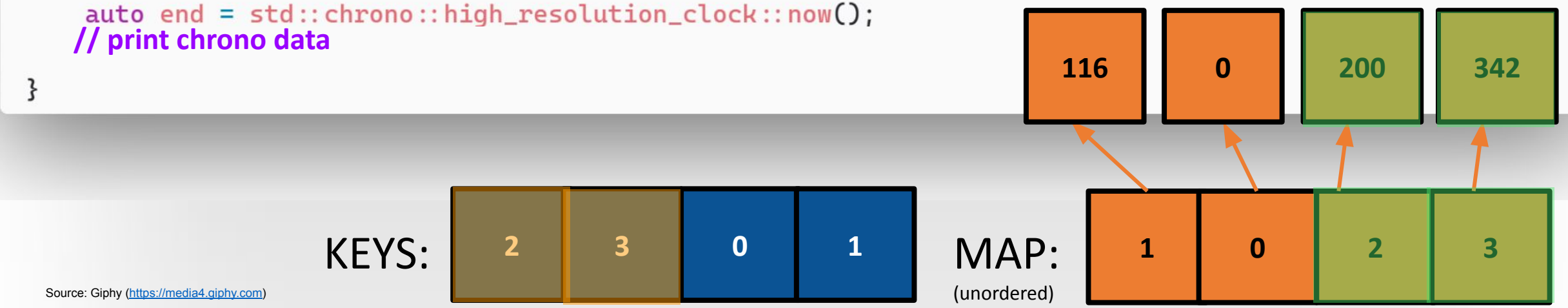
Do we even need to maintain the unordered_map at all? #8270 runs 10x faster by only using the vector.

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

The Case of Catching the Missing Complexity (Simplified)

```
constexpr size_t N = 1'000'000; // number of elements in map


 /* Access each value and sum it up */
void benchmark_unordered_map(const std::unordered_map<i
    volatile int sum = 0;
    auto start = std::chrono::high_resolution_clock::no
    for (size_t i = 0; i < LOOKUPS; ++i) {
        sum += map.at(keys[i]);
    }
    auto end = std::chrono::high_resolution_clock::now();
    // print chrono data
}
```

O(K)
K is the number of keys



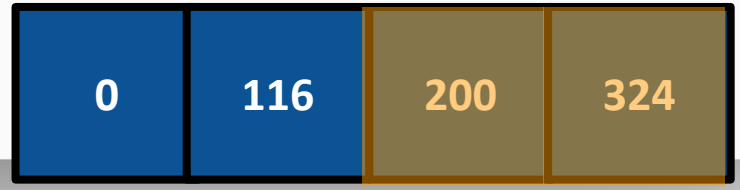
Source: Giphy (https://media4.giphy.com)

The Case of Catching the Missing Complexity (Simplified)

```
constexpr size_t N = 1'000'000; // number of elements in map


/* Access each value and sum it up */
void benchmark_vector(const std::vector<int>& vec, cons
    volatile int sum = 0;
    auto start = std::chrono::high_resolution_clock::no
    for (size_t i = 0; i < LOOKUPS; ++i) {
        sum += vec[keys[i]];
    }
    auto end = std::chrono::high_resolution_clock::now();
    // print chrono data
}
```

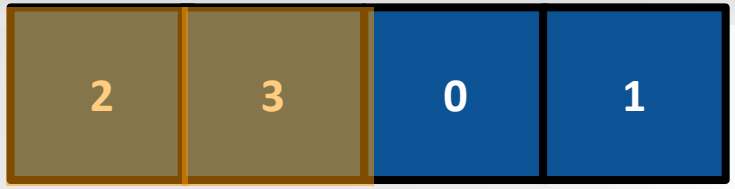
O(K)
K is the number of keys

VEC:



P.S. In reality, this was done with a vector of pairs. The code above is simplified.

KEYS:



Source: Giphy (https://media4.giphy.com)

The Vector Wins!

```
[vector] Elapsed: 23.9168 ms
[unordered_map] Elapsed: 361.732 ms

[vector] Elapsed: 21.505 ms
[unordered_map] Elapsed: 353.587 ms

[vector] Elapsed: 22.4668 ms
[unordered_map] Elapsed: 338.601 ms
```

93.58% Speedup

15.53x faster!

Bloomberg

Engineering

Gathering Intel (and not the x86 kind)

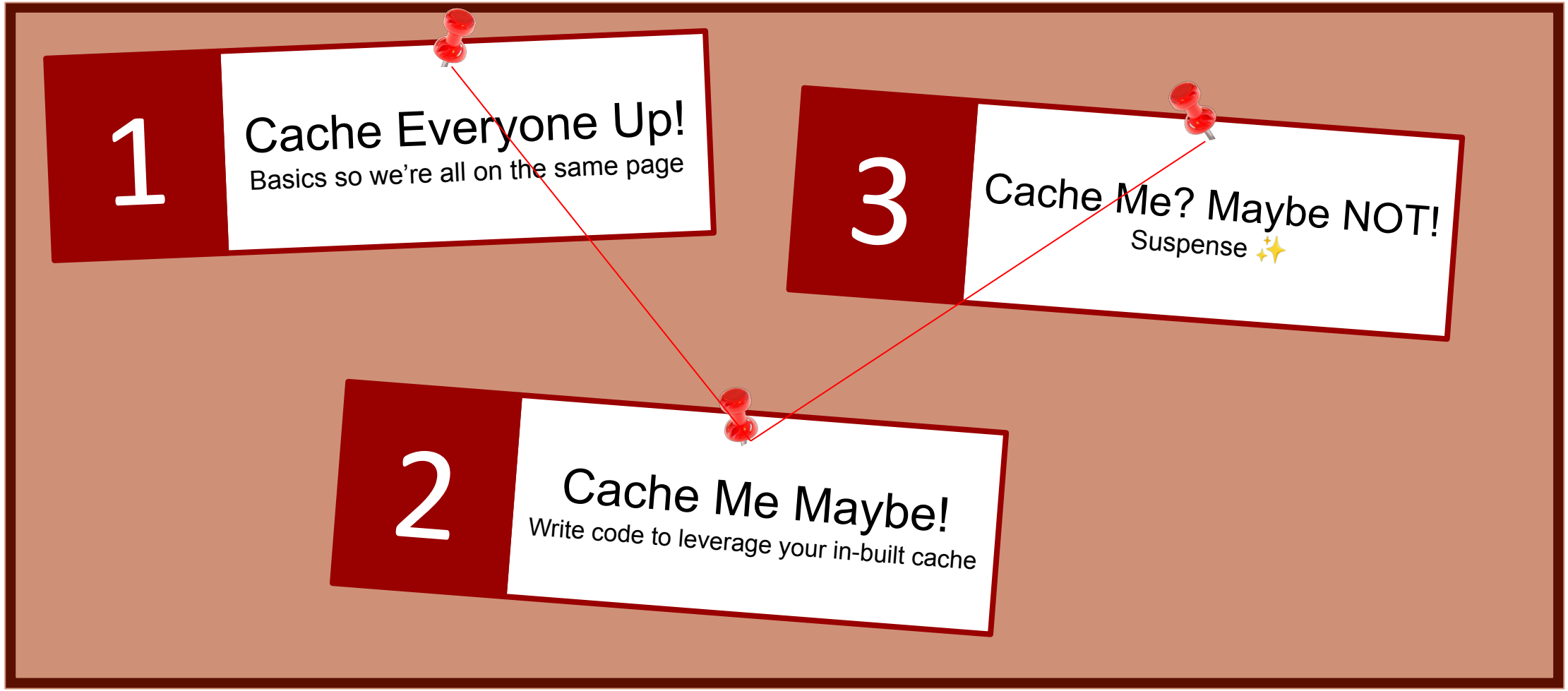
Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Code Sleuths Assemble! Can YOU Crack the Case?

- Hashing takes time
- Fewer machine instructions
- No pointer chasing in the vector!
- Vector has contiguous memory!

How many detectives in the room knew this already?

~~Agenda~~ The Case File: What We'll Be Investigating Today



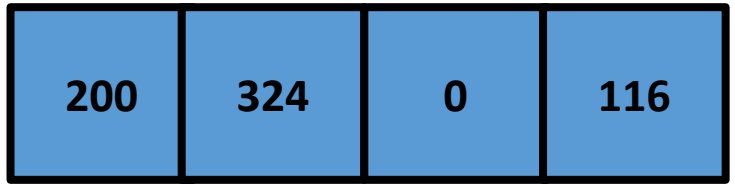
P.S. From first timers to pros, this case holds a new twist for almost every detective.

Engineering

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

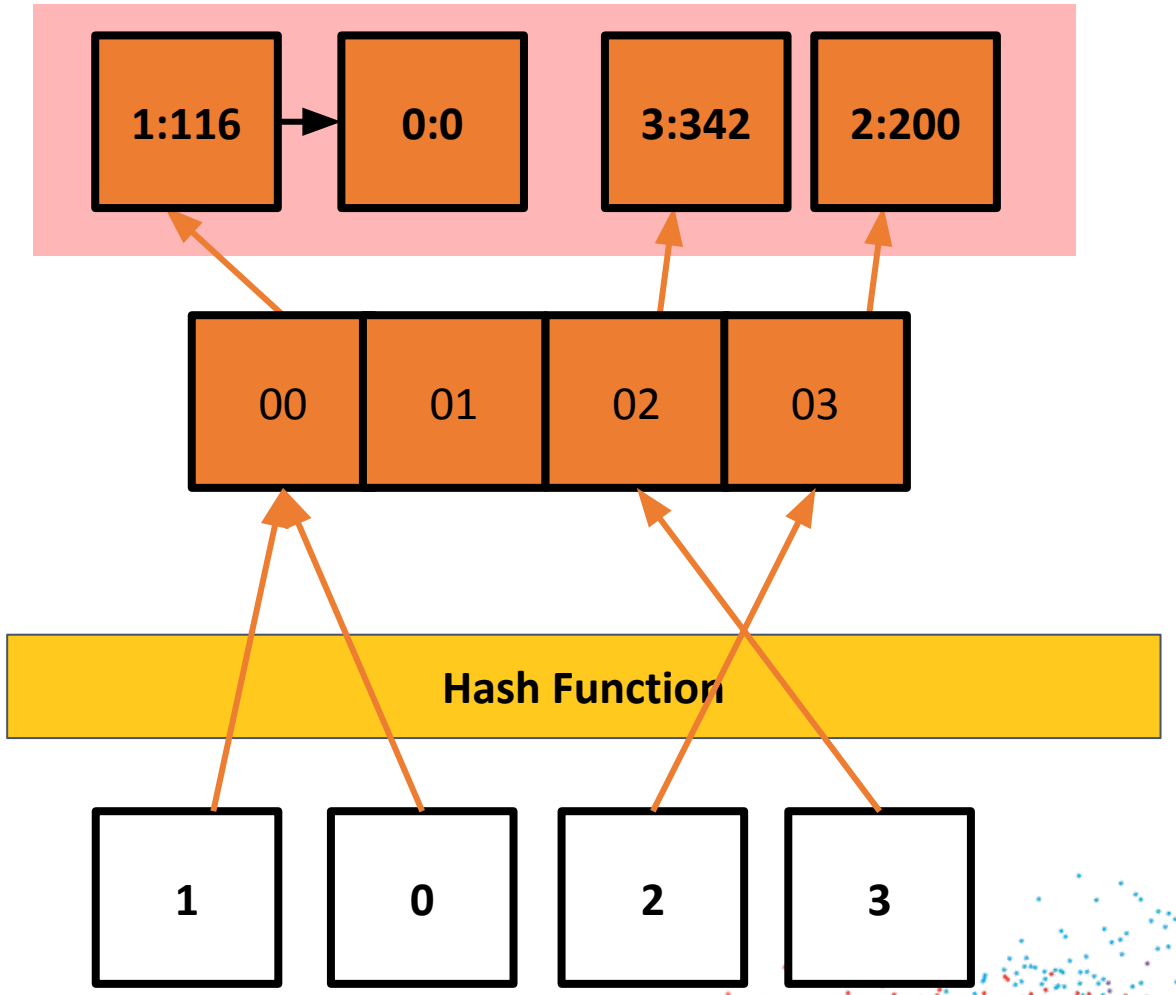
Why the vector was faster than the unordered map

Contiguous memory!



Why the vector was faster than the unordered map

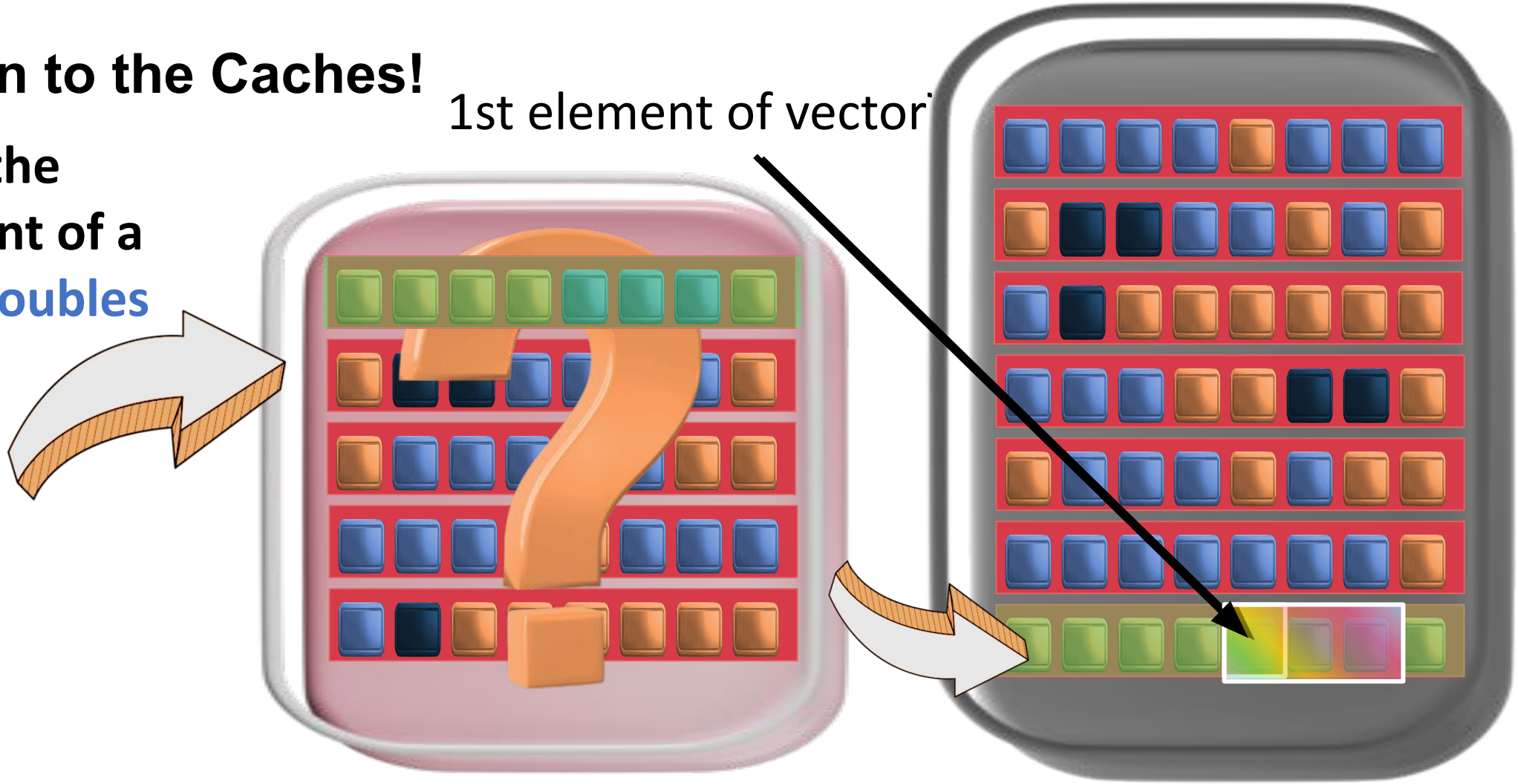
NOT Contiguous memory! **But why does that matter?**



Drill Down to the Caches!

Accessing the first element of a vector of doubles

Cache Miss



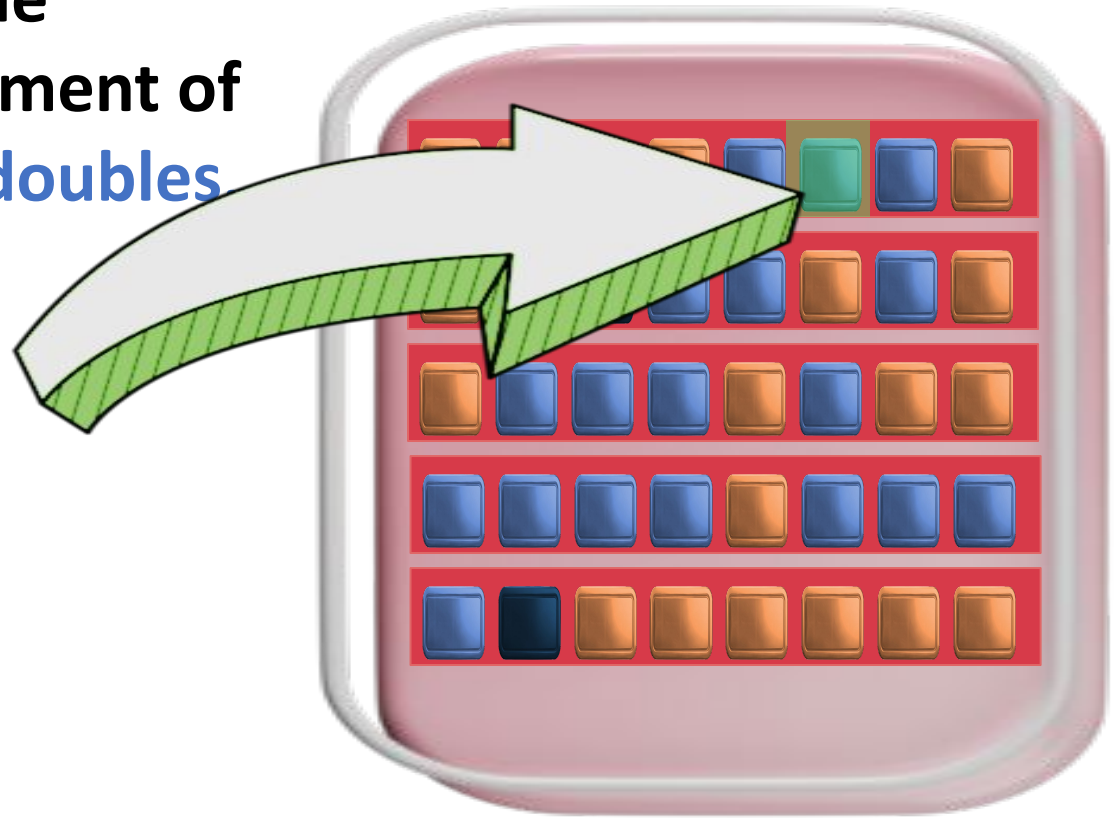
Cache line is typically 64 Bytes in x86, 32 Bytes in ARM

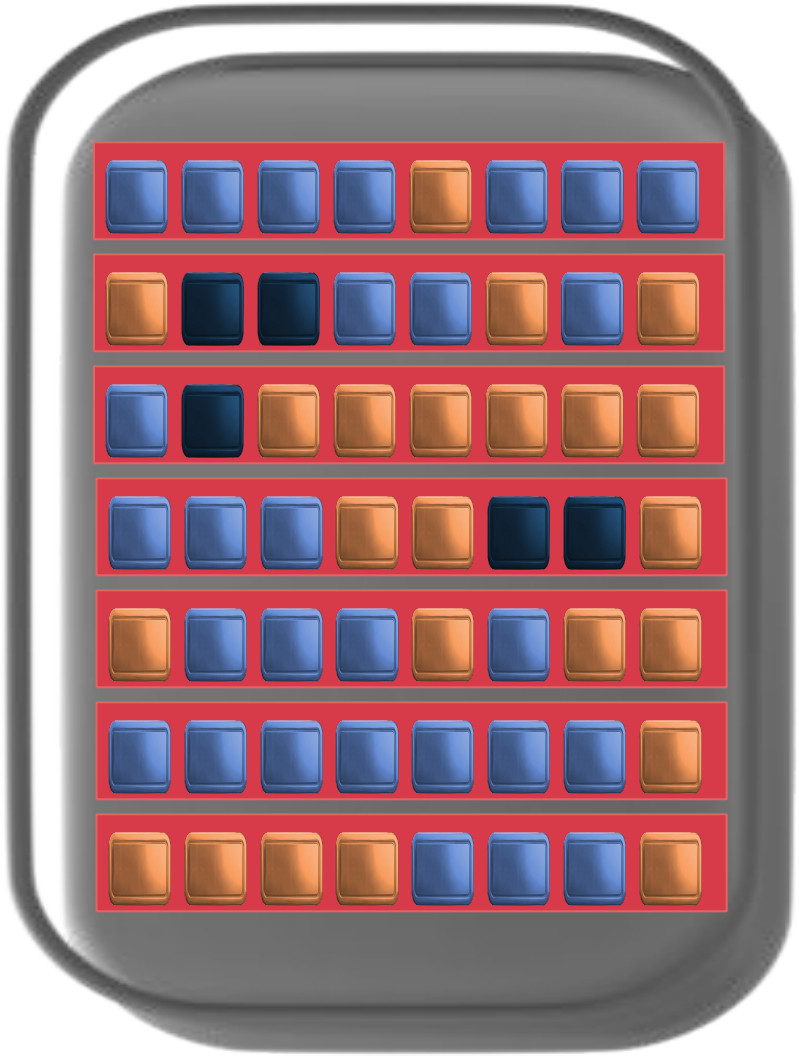
Source: Vecteezy (www.vecteezy.com)

Drill Down to the Caches!

Accessing the
SECOND element of
a vector of doubles

Cache hit





Cache

Main memory

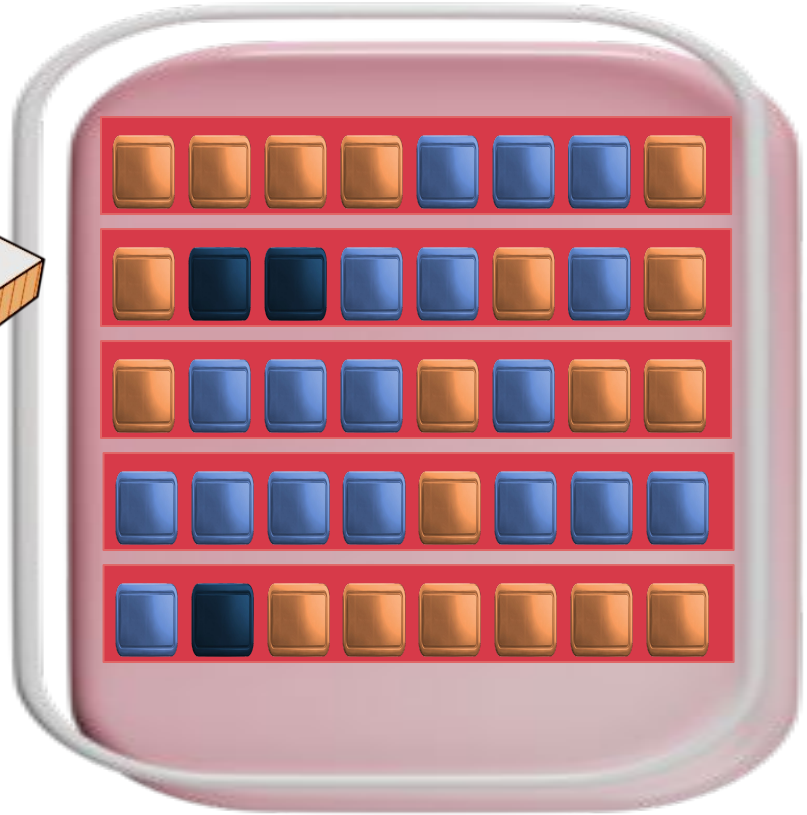
TechAtBloomberg.com

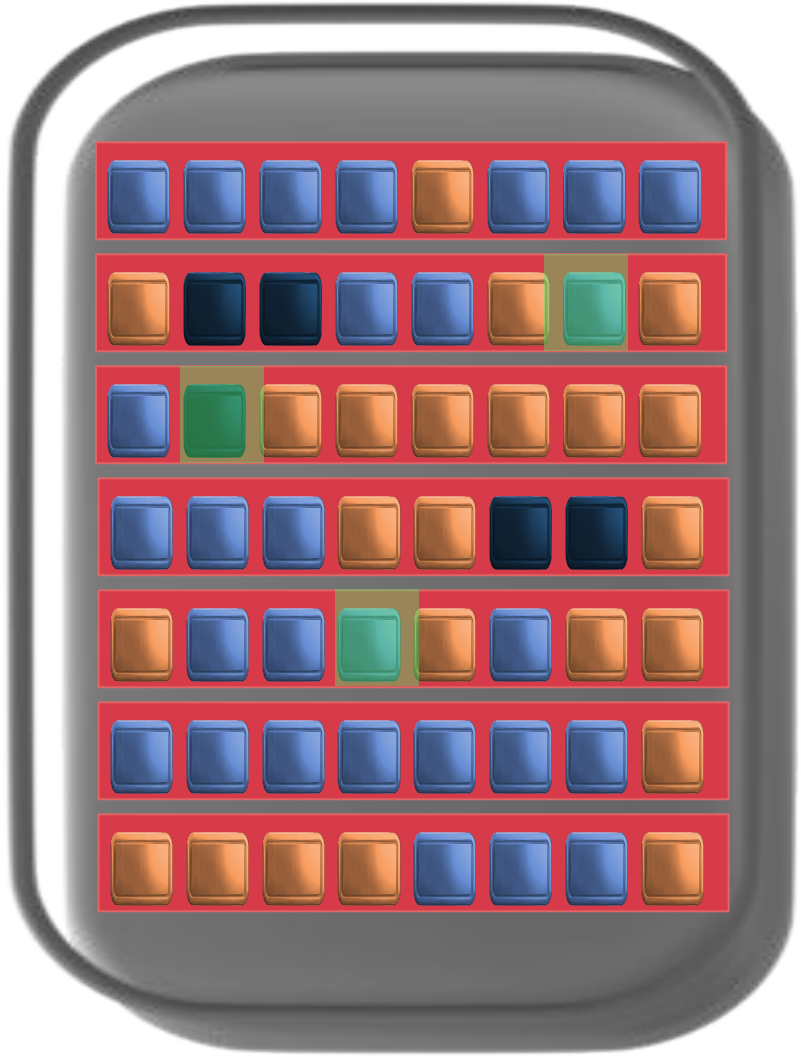
Source: Vecteezy (www.vecteezy.com)

Drill Down to the Caches!

Accessing the SECOND element of an unordered map.

Cache Miss





Cache

Main memory

TechAtBloomberg.com

Source: Vecteezy (www.vecteezy.com)

So what if there is a cache miss?

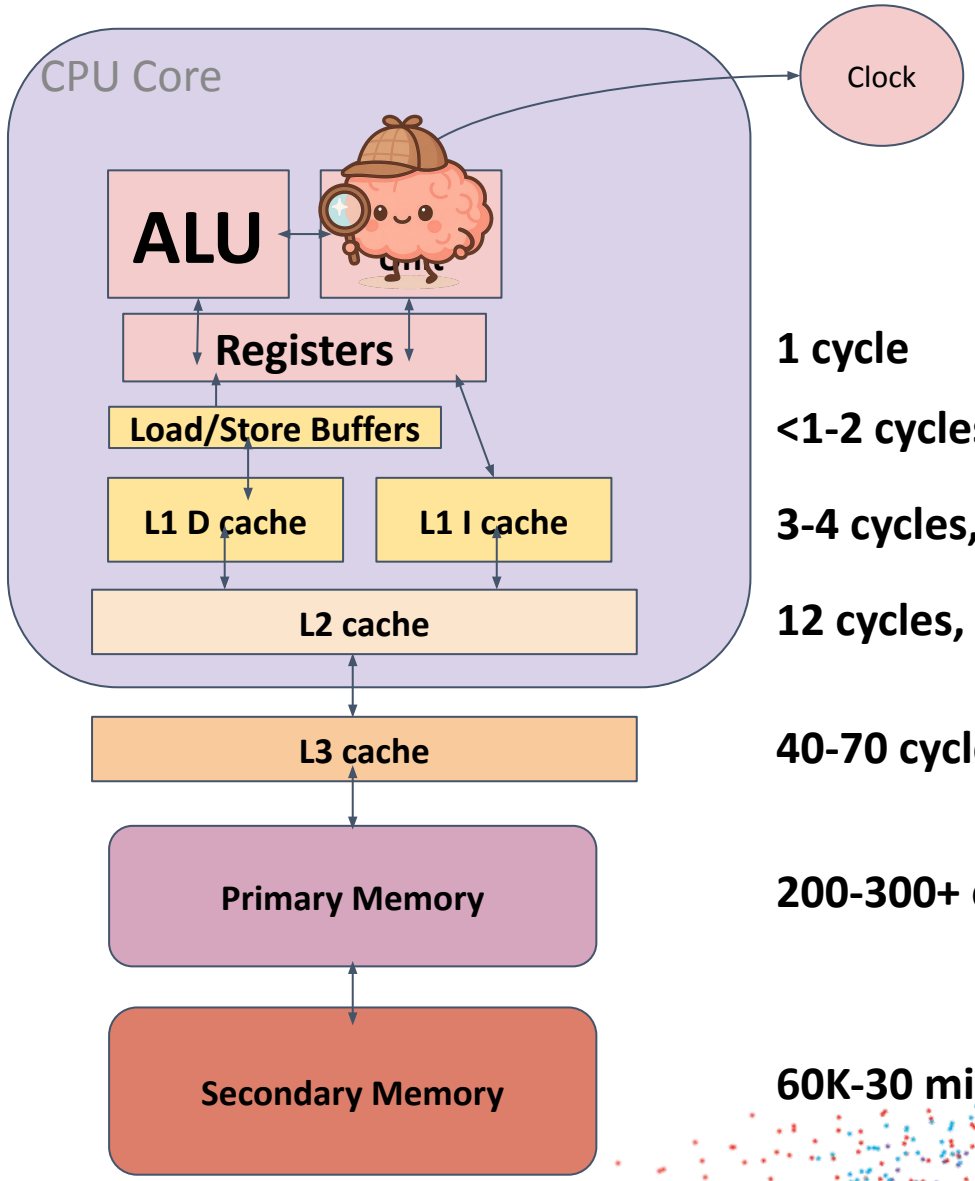
TechAtBloomberg.com

Bloomberg

Engineering

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Caches 101 - Contemporary CPUs



1 cycle

<1-2 cycles, 0.3 ns

3-4 cycles, 1 ns

12 cycles, 3-4 ns

40-70 cycles, 10-20 ns

200-300+ cycles, 60-120ns

60K-30 mil cycles,0.2-10ms

Slower access time

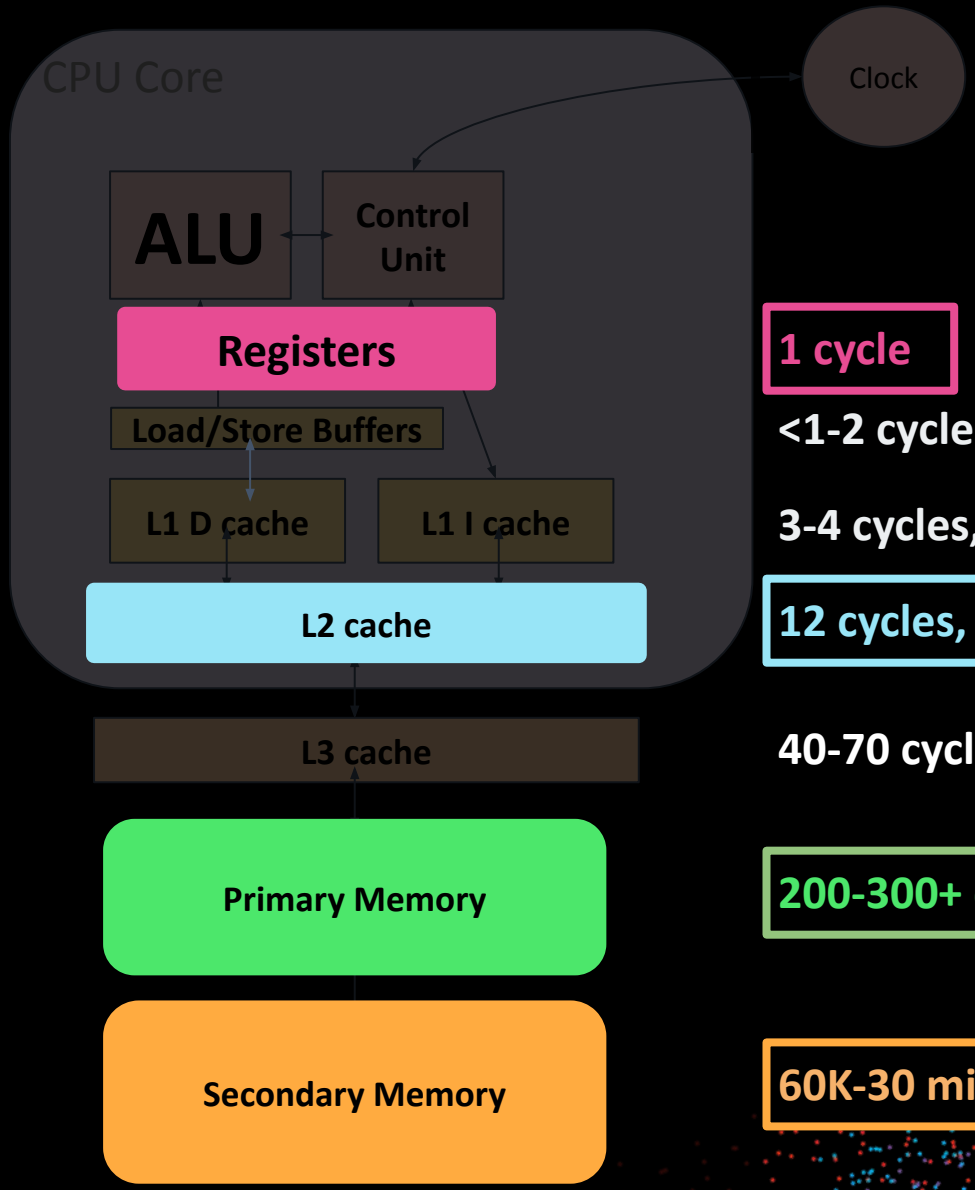
More storage space

(because less $ per KB)

* Diagram leaves out TLB, MMR, branch predictor, etc.
* There are also different kinds of architectures!

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Caches 101 - Contemporary CPUs



1 cycle

<1-2 cycles, 0.3 ns

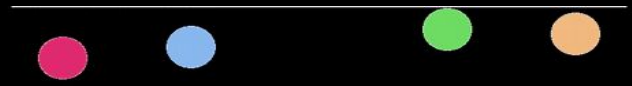
3-4 cycles, 1 ns

12 cycles, 3-4 ns

40-70 cycles, 10-20 ns

200-300+ cycles, 60-120ns

60K-30 mil cycles,0.2-10ms



In this investigation, the cache is sacred.

Don't miss it.

NOT CONTIGUOUS

CONTIGUOUS

★ WANTED ~~DEAD~~ OR ~~ALIVE~~ ★

std::map
std::unordered_map
std::list
std::forward_list
std::set
std:unordered_set
std::multiset
std::unordered_multiset
std::multimap
std::unordered_multimap
boost::unordered_set

std::vector
std::array
C-style arrays (e.g int[])
std::valarray
std::string
std::flat_map (C++26)?
std::flat_set (C++26)?
absl::flat_hash_set
boost::flat_set
folly::F14FastSet

(C++ 26?, https://wg21.link/p0429)

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Agenda – The Case File: What We'll Be Investigating Today

2

Cache Me Maybe!

Write code to leverage your in-built cache

Clues to Cache Friendliness

Maybe you've already heard *whispers* about these techniques.

Trust No One.

Let's verify your understanding!

Clues to Cache Friendliness

Compact structures help fit more things into the cache.

Due to false sharing, it may not always leads to fewer cache misses though.

```
struct Bad {
    char a;
    double b;
    char c;
};
```

Total size = **24 bytes**

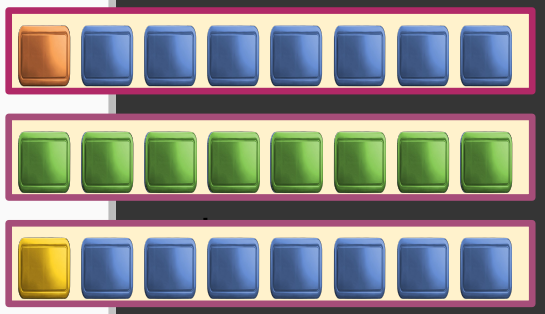


Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Clues to Cache Friendliness

Compact structures help fit more things into the cache.

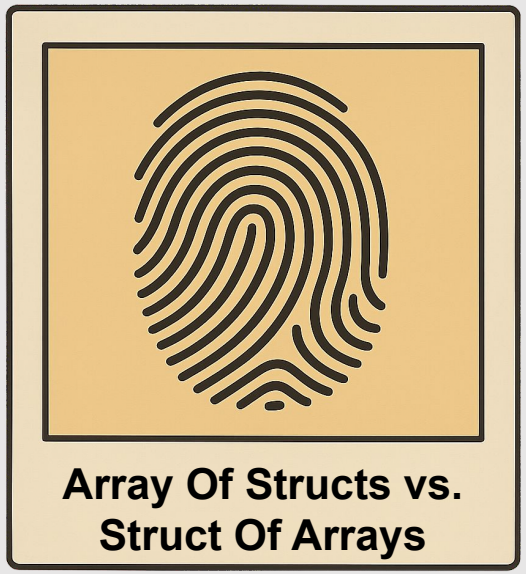
Due to false sharing, it may not always leads to fewer cache misses though.

```
struct Good {
    char a;
    char c;
    double b;
};
```

Total size = **16 bytes**

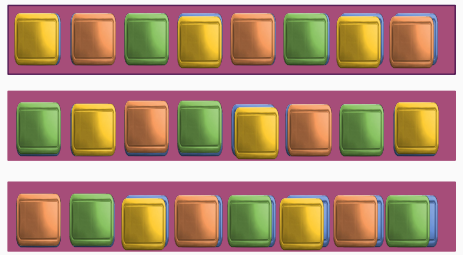
Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Clues to Cache Friendliness



Design your code based on how you access your data.

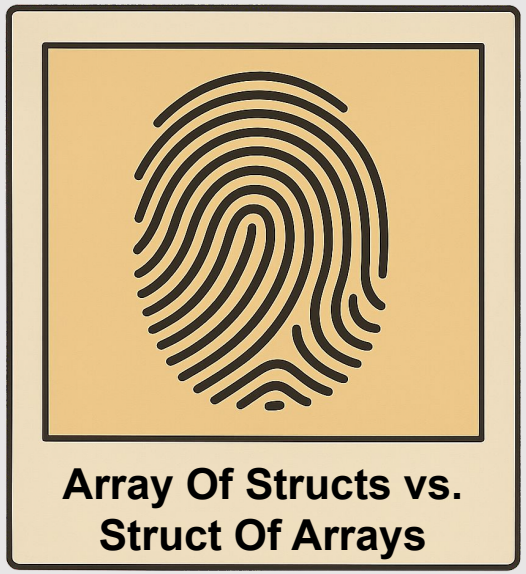
```
struct Detective {
    double age;
    double numberOfSolvedCases;
    double classificationLevel;
};

std::vector<Detective> detectives;
```



Your code: Loops through **numberOfSolvedCases** to print them out!

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Clues to Cache Friendliness



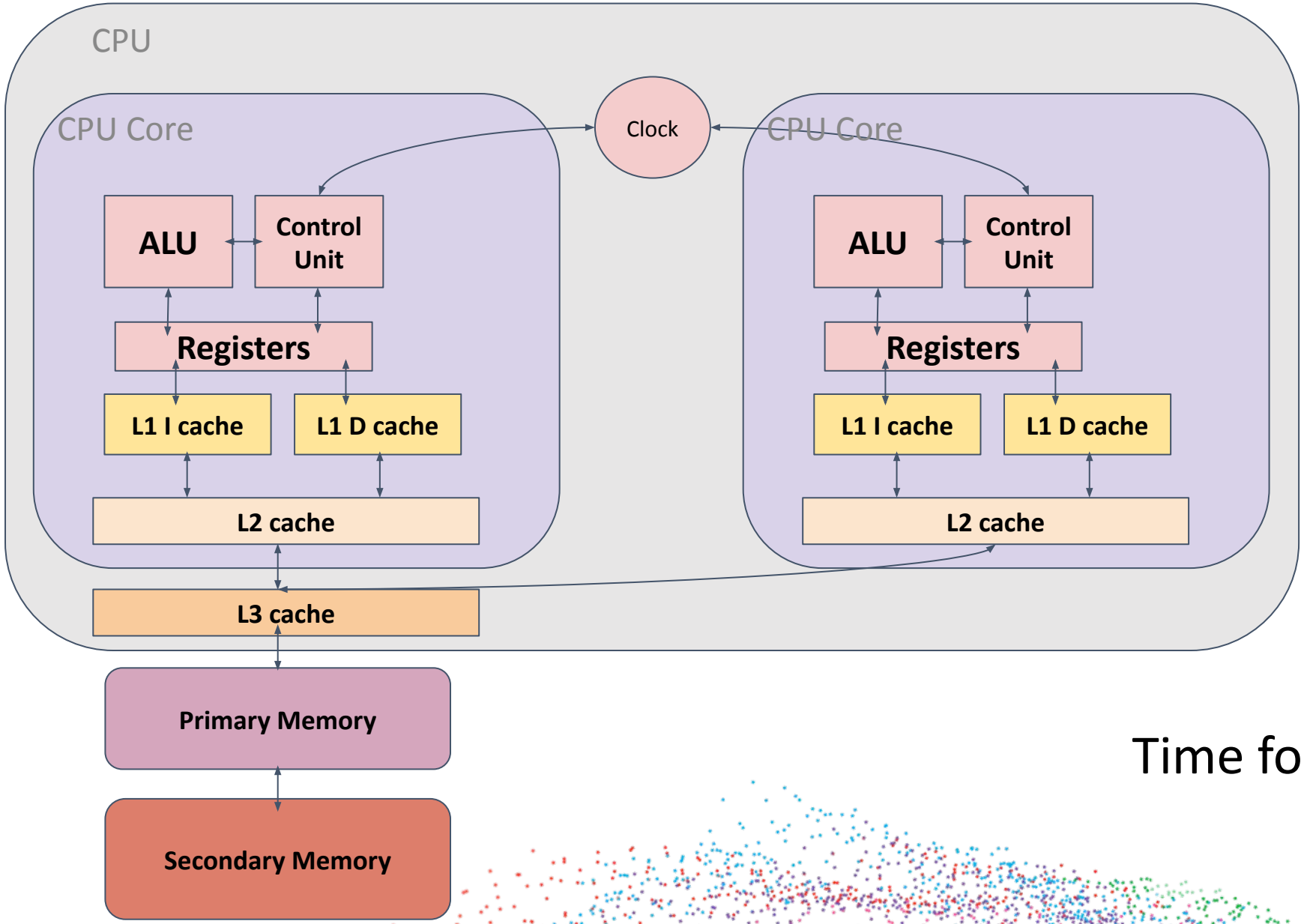
Design your code based on how you access your data.

```
struct Detectives {
    std::vector<double> age;
    std::vector<double> numberOfSolvedCases;
    std::vector<double> classificationLevel;
};
```

Fits more in!
Less Cache Misses*
* - Unless False Sharing happens

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

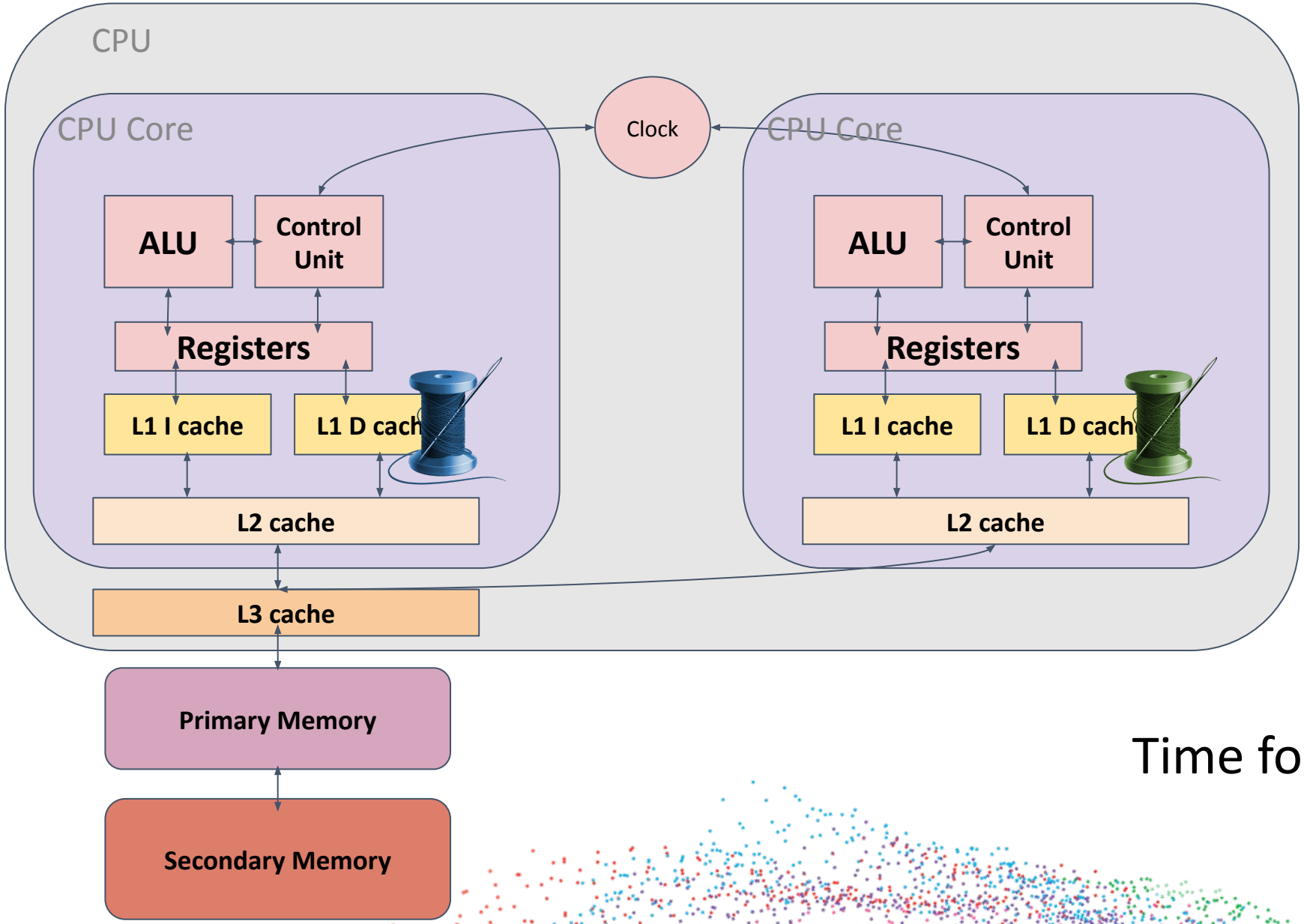
First Rule of being a Detective? Trust No One…



Time for false sharing!

Source: pngarts (https://www.pngarts.com)

Multiple Threads on Multiple Cores

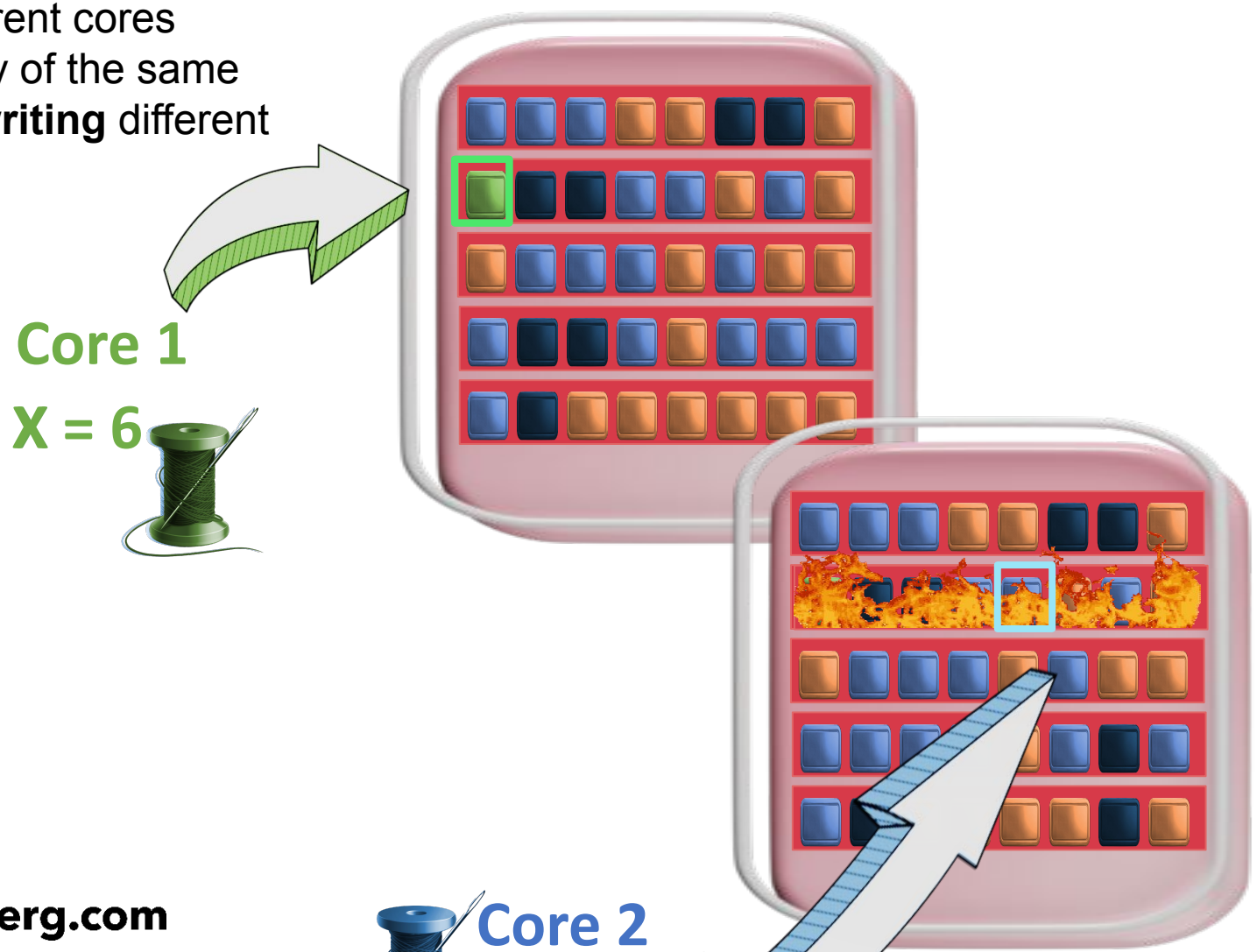


Time for false sharing!

Source: pngarts (https://www.pngarts.com)

False Sharing

Threads on different cores accessing a copy of the same cache line, but **writing** different variables



Source: Are.na (https://www.are.na.com)

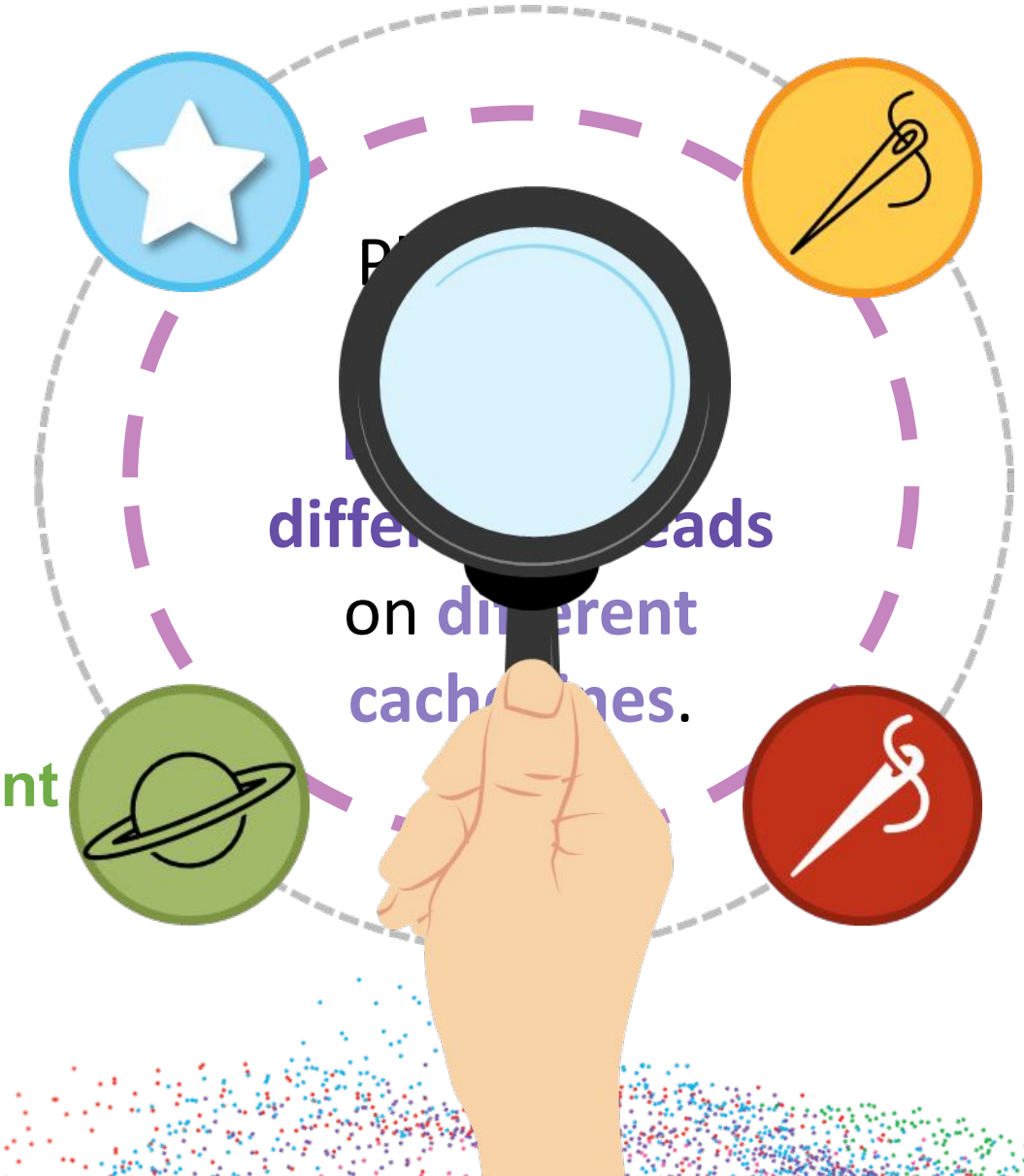
Source: Vecteezy (www.vecteezy.com)

Source: pngarts(https://www.pngarts.com)

How to fix false sharing?

Data Structure Reorganization

Local Variables on the Thread



Padding & Alignment

Thread-Local Storage

TechAtBloomberg.com

Declaration Order Matters … Trust No One…

```
volatile int x = rand();
heavyWork();
int y = x + 5;
```

```
volatile int x = rand();
int y = x + 5;
heavyWork();
```

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Even Declaration Order Matters!

```
void scenarioA() {
    volatile int x = 42;
    doHeavyWork(); // Pollute Cache

    auto start = std::chrono::high_resolution_clock::now();
    int y = x + 1;
    auto end = std::chrono::high_resolution_clock::now();

    // print time taken
    // prevent y to prevent optimization
}
```

A: Declare variable → heavy cache work → access variable

Source: Giphy (https://media4.giphy.com)

Even Declaration Order Matters!

```
void scenarioB() {
    volatile int x = 42;
    auto start = std::chrono::high_resolution_clock::now();
    int y = x + 1;
    auto end = std::chrono::high_resolution_clock::now();

    doHeavyWork(); // Pollute Cache

    // print time taken
    // prevent y to prevent optimization
}
```

B: Declare variable → access variable → heavy cache work

Source: Giphy (https://media4.giphy.com)

Even Declaration Order Matters!

```
int main() {
    scenarioA();
    scenarioB();
    return 0;
}
```

A: Declare variable → heavy cache work → access variable

B: Declare variable → access variable → heavy cache work

Even Declaration Order Matters!

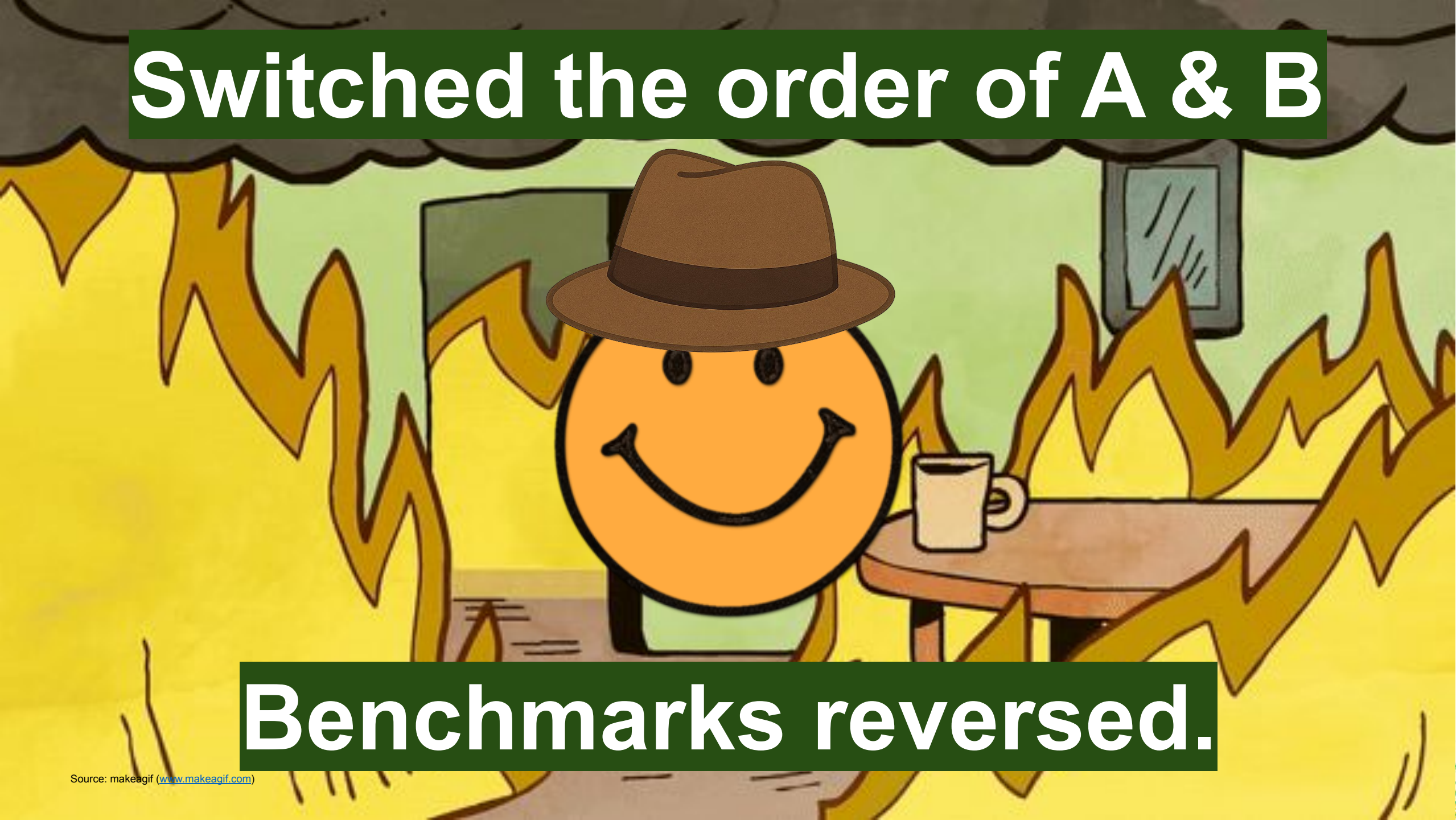
Run	Scenario A Time (ns)	Scenario B Time (ns)	Speedup (A → B)
1	36	28	1.29x
2	48	23	2.09x
3	43	34	1.26x

A: Declare variable → heavy cache work → access variable

B: Declare variable → access variable → heavy cache work

Switched the order of A & B

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.



Source: makeagif (www.makeagif.com)

What Just Happened? Why did A get faster than B when B was run first?

Any guesses?

```
int main() {
    scenarioA();
    scenarioB();
    return 0;
}
```

```
int main() {
    scenarioB();
    scenarioA();
    return 0;
}
```

A: Declare variable → heavy cache work → access variable

B: Declare variable → access variable → heavy cache work

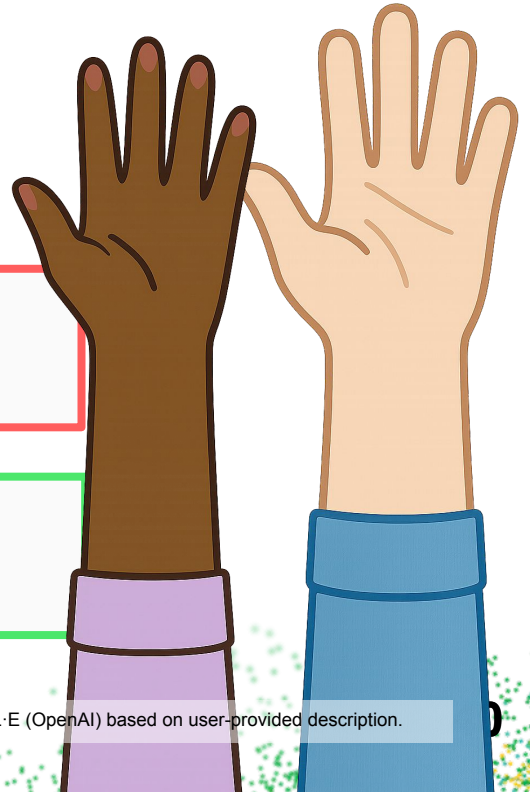
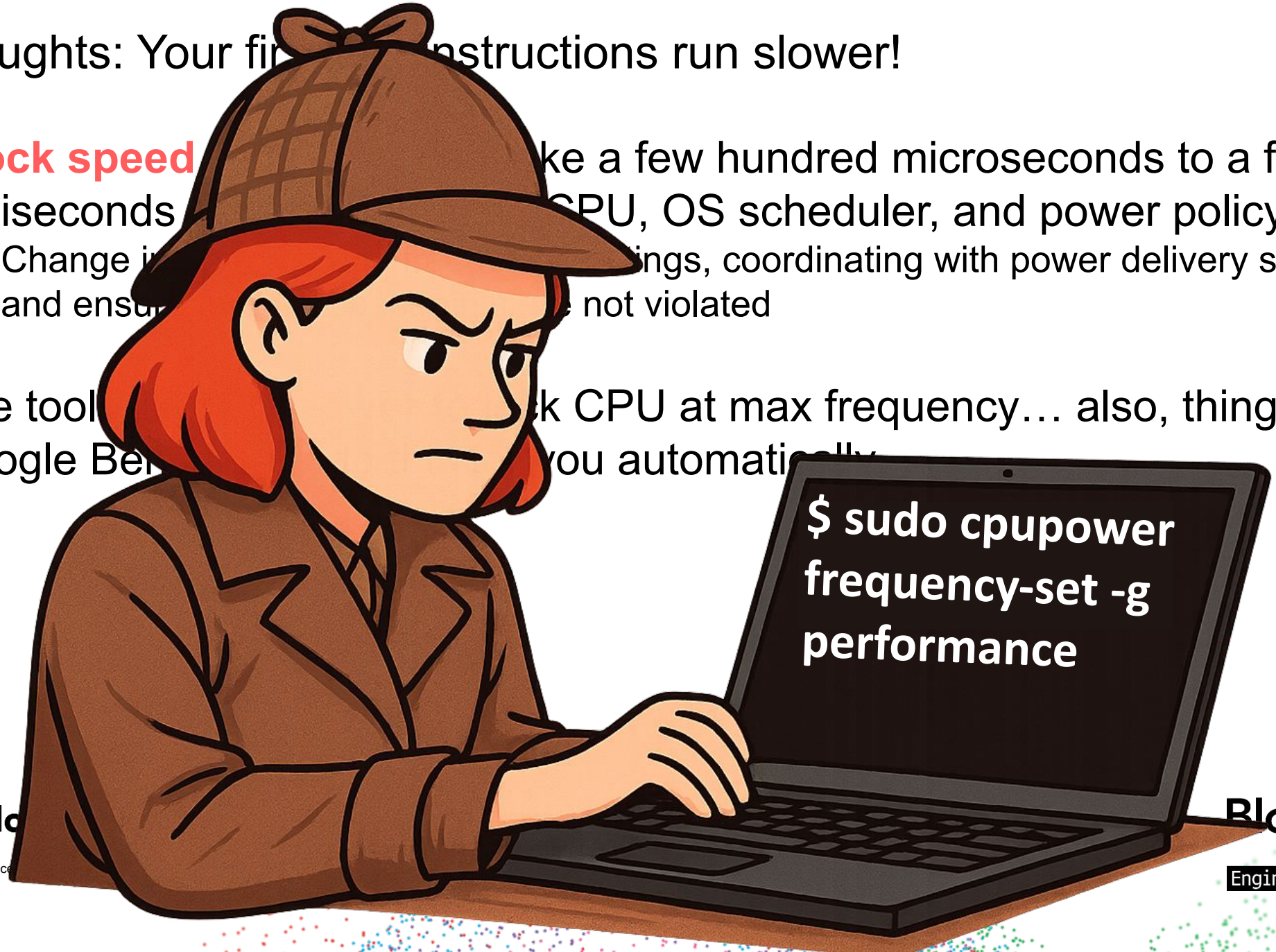


Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

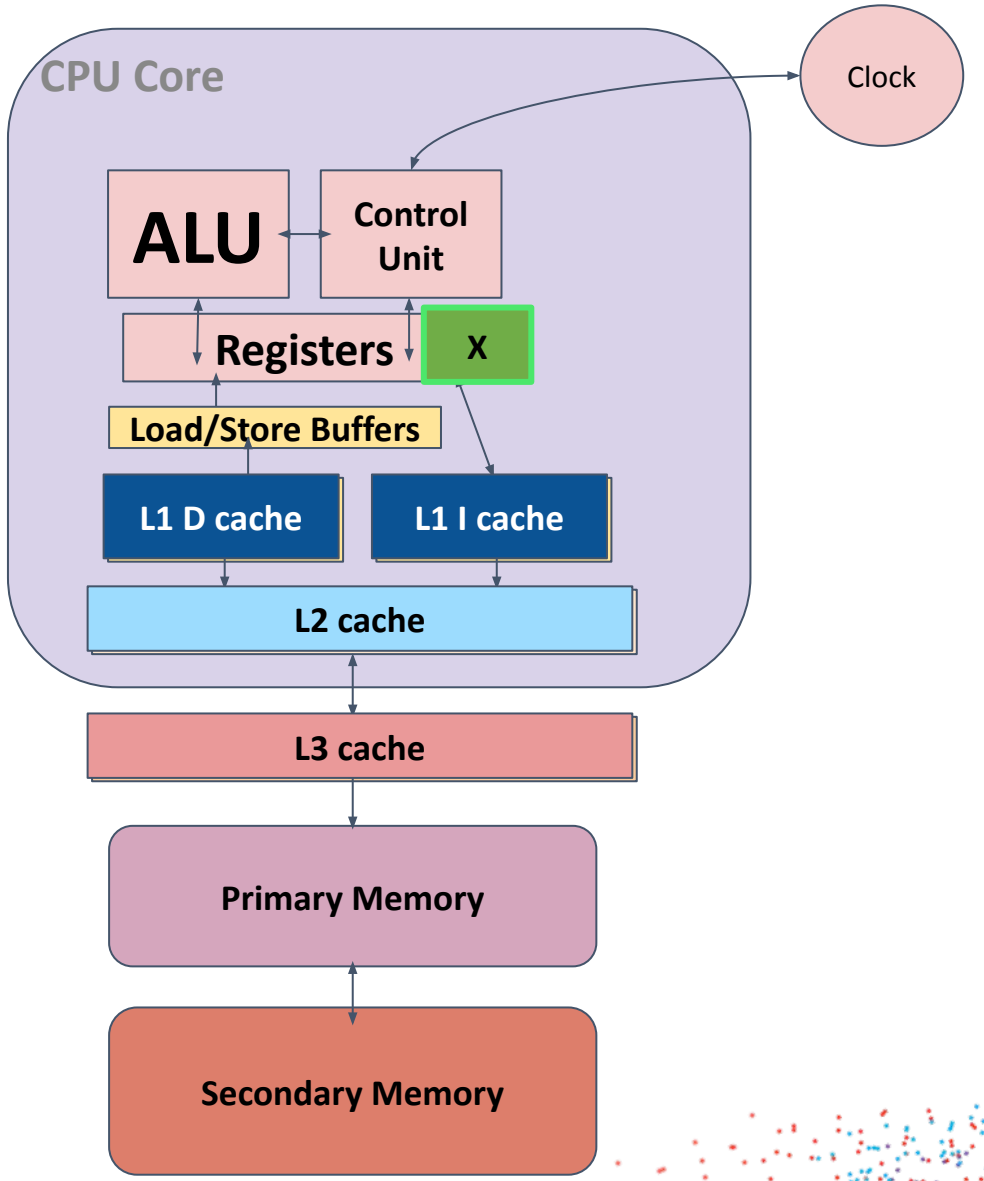
My thoughts: Your fi nstructions run slower!

- **Clock speed** ke a few hundred microseconds to a few milliseconds PU, OS scheduler, and power policy
  - Change i ings, coordinating with power delivery systems, and ensu not violated

- Use tool k CPU at max frequency… also, things like Google Be ou automati



Still no clear winner! The evidence was inconclusive



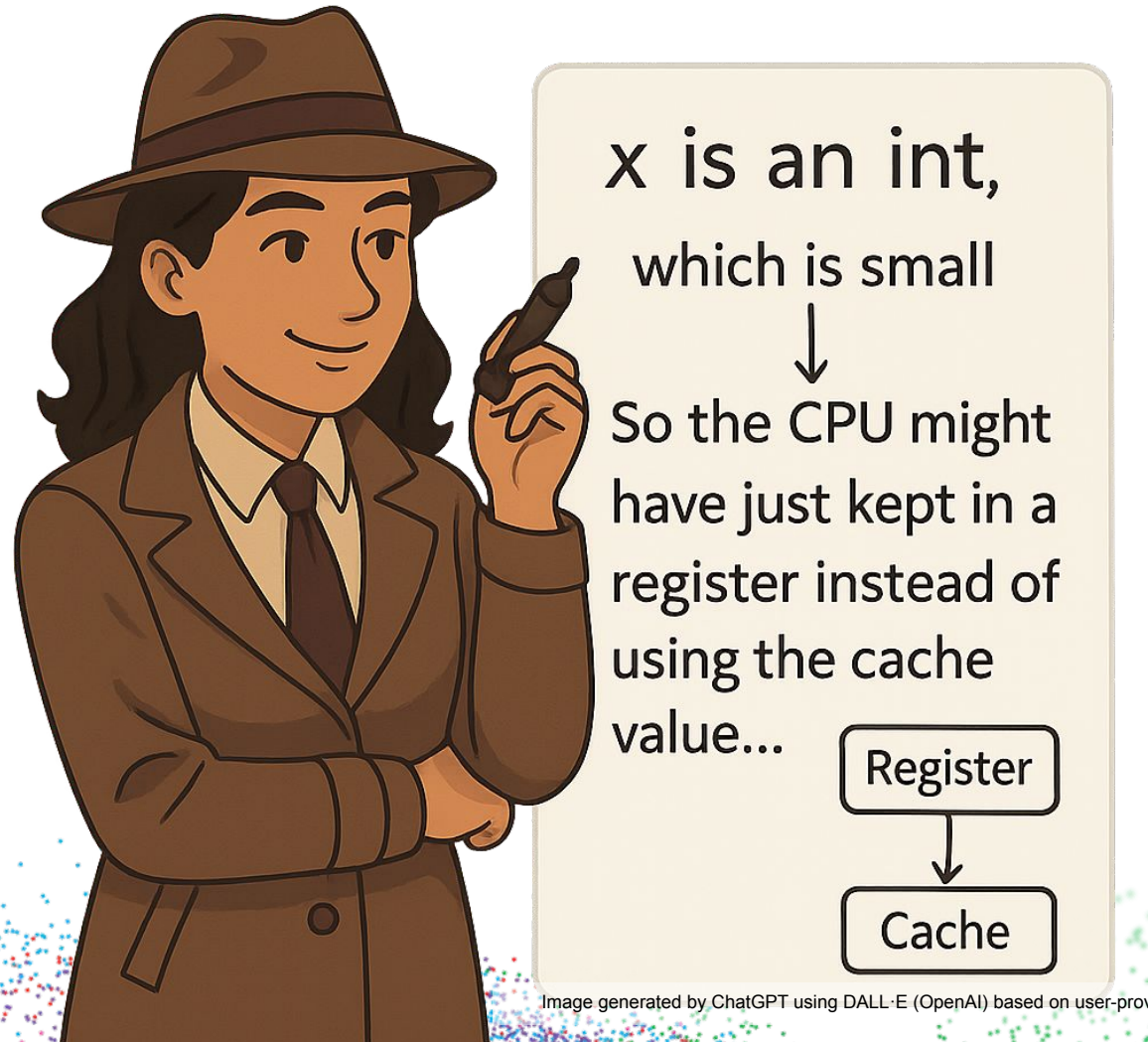
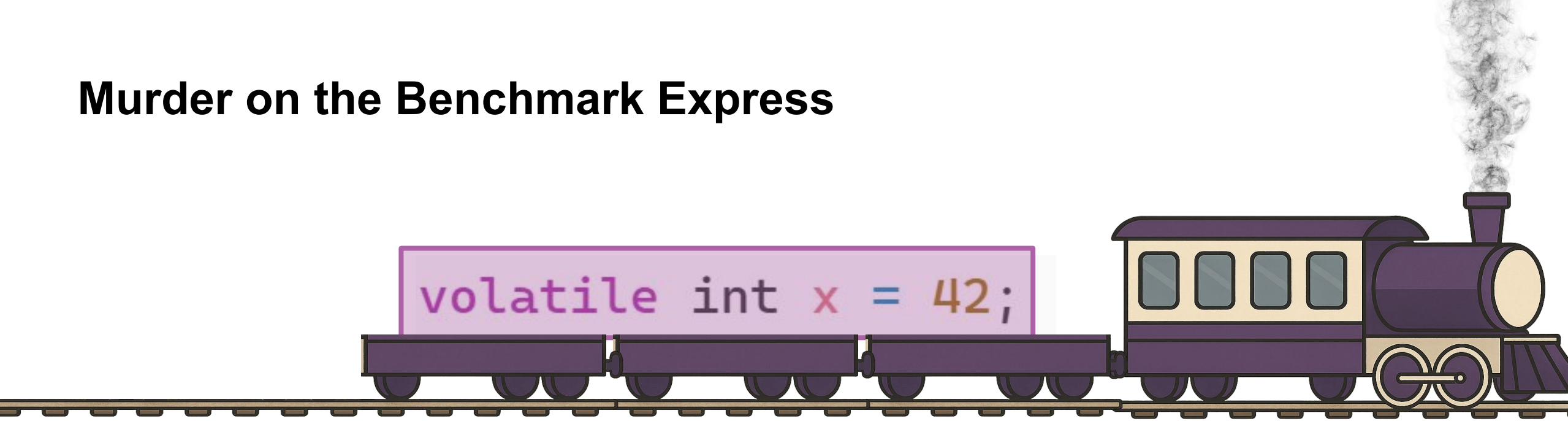


Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Murder on the Benchmark Express



“This variable might be changed or accessed outside the current code's control. My dear compiler, do not optimize access to it.” - **volatile**

It **can interfere with realistic behavior** (e.g. inhibits certain optimizations like keeping value in register without refetching it).

benchmark::DoNotOptimize(x) prevents something from being optimized away by the compiler in a more controlled and performance-safe way.

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Tried Making x BIG

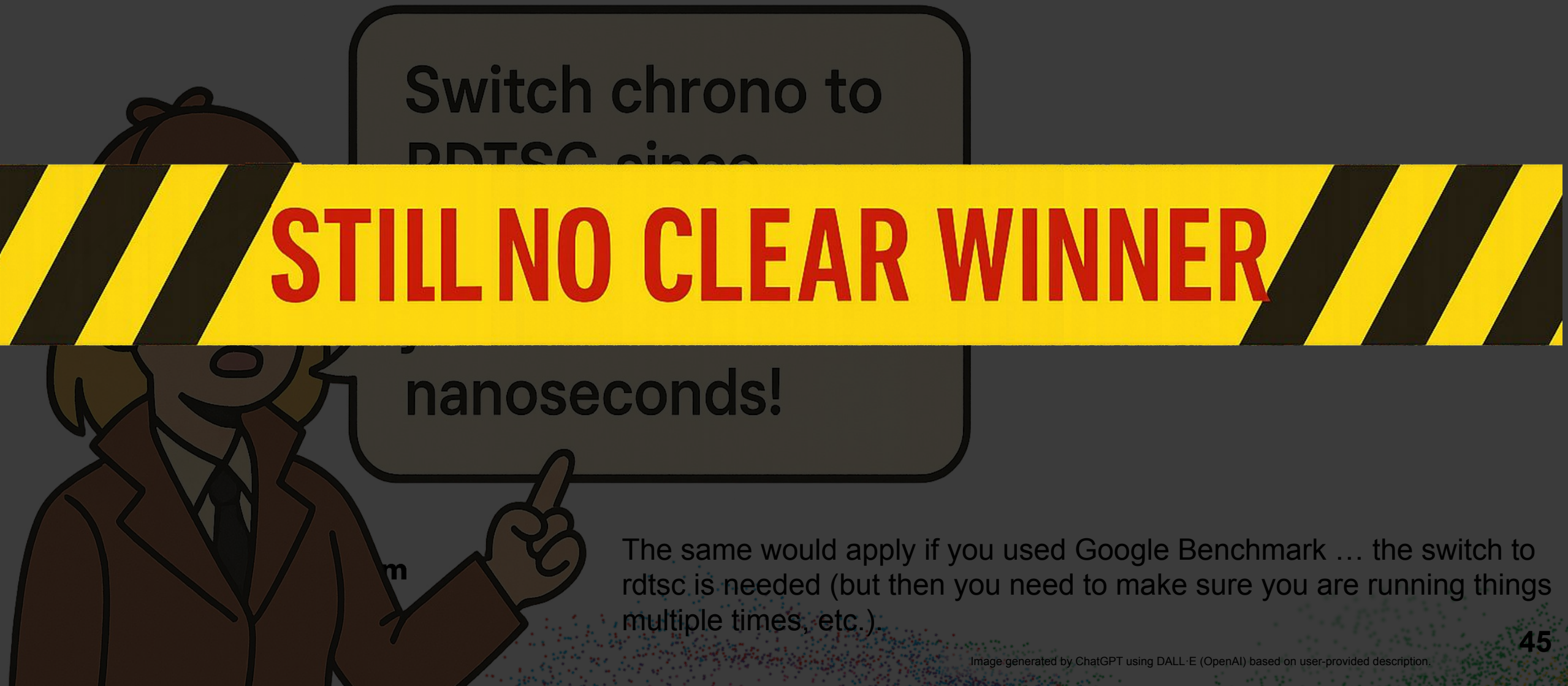
std::vector<size_t> x , with 4,194,304 elements (4 * 1024 * 1024).

But still no clear winner!

TechAtBloomberg.com

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

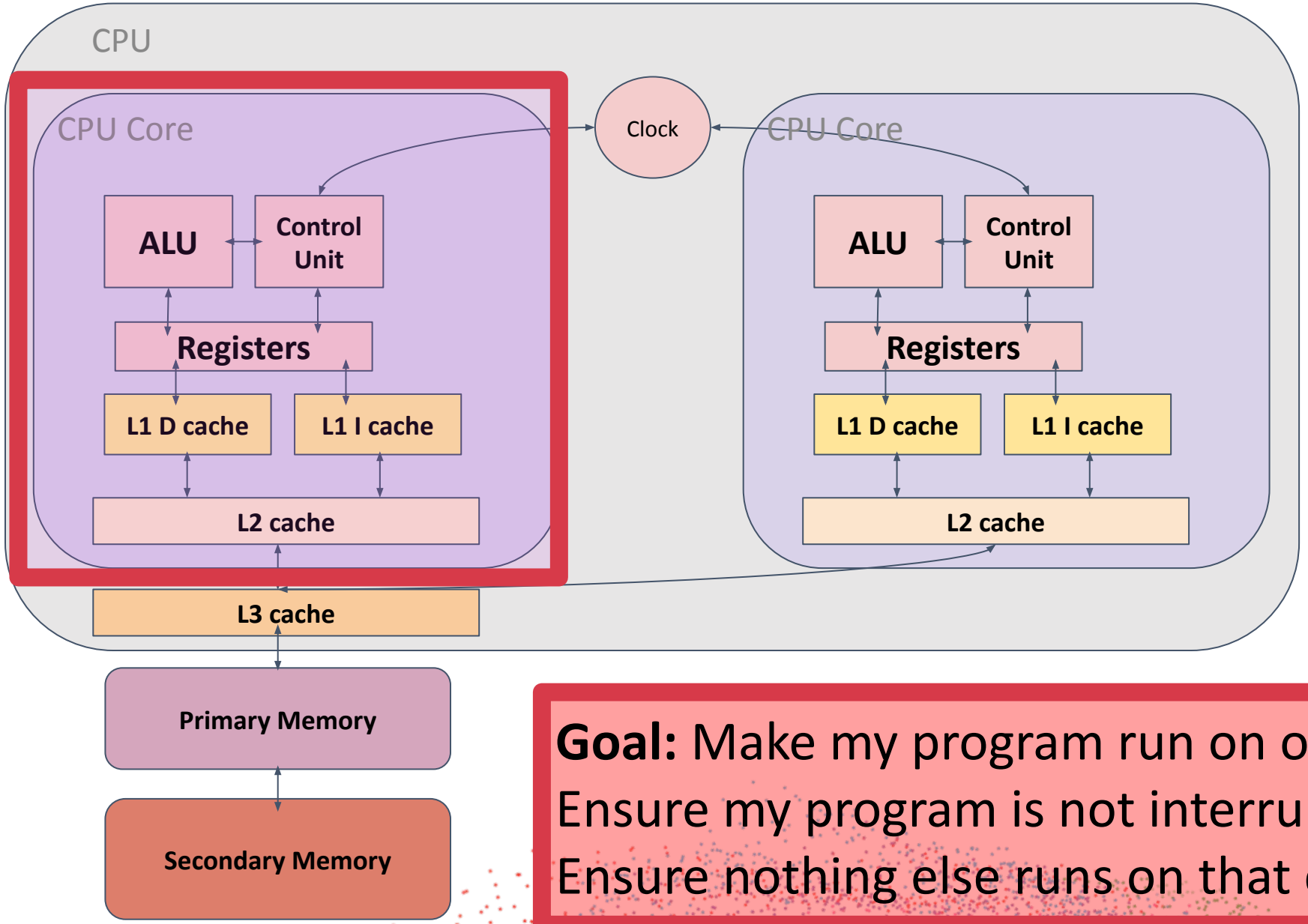
Switch to RDTSC



The same would apply if you used Google Benchmark … the switch to rdtsc is needed (but then you need to make sure you are running things multiple times, etc.).

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Multiple Cores, My Dear Watson!



Goal: Make my program run on only one core.
Ensure my program is not interrupted mid run.
Ensure nothing else runs on that core.

Source: pngarts (https://www.pngarts.com)

Make sure the task is run on only one core and nothing else interrupts it!

BIOS > SMT off

Simultaneous Multi-Threading off prevents multiple threads from running on the same **physical core.**

With SMT off, each physical core exposes one logical core.

ONLY 1 TASK WILL RUN ON EACH PHYSICAL CORE

THE OS DOESN'T RUN ITS STUFF ON THIS CORE + DOESN'T INTERRUPT ANY TASK ON IT

RUN MY PROGRAM ON THIS CORE

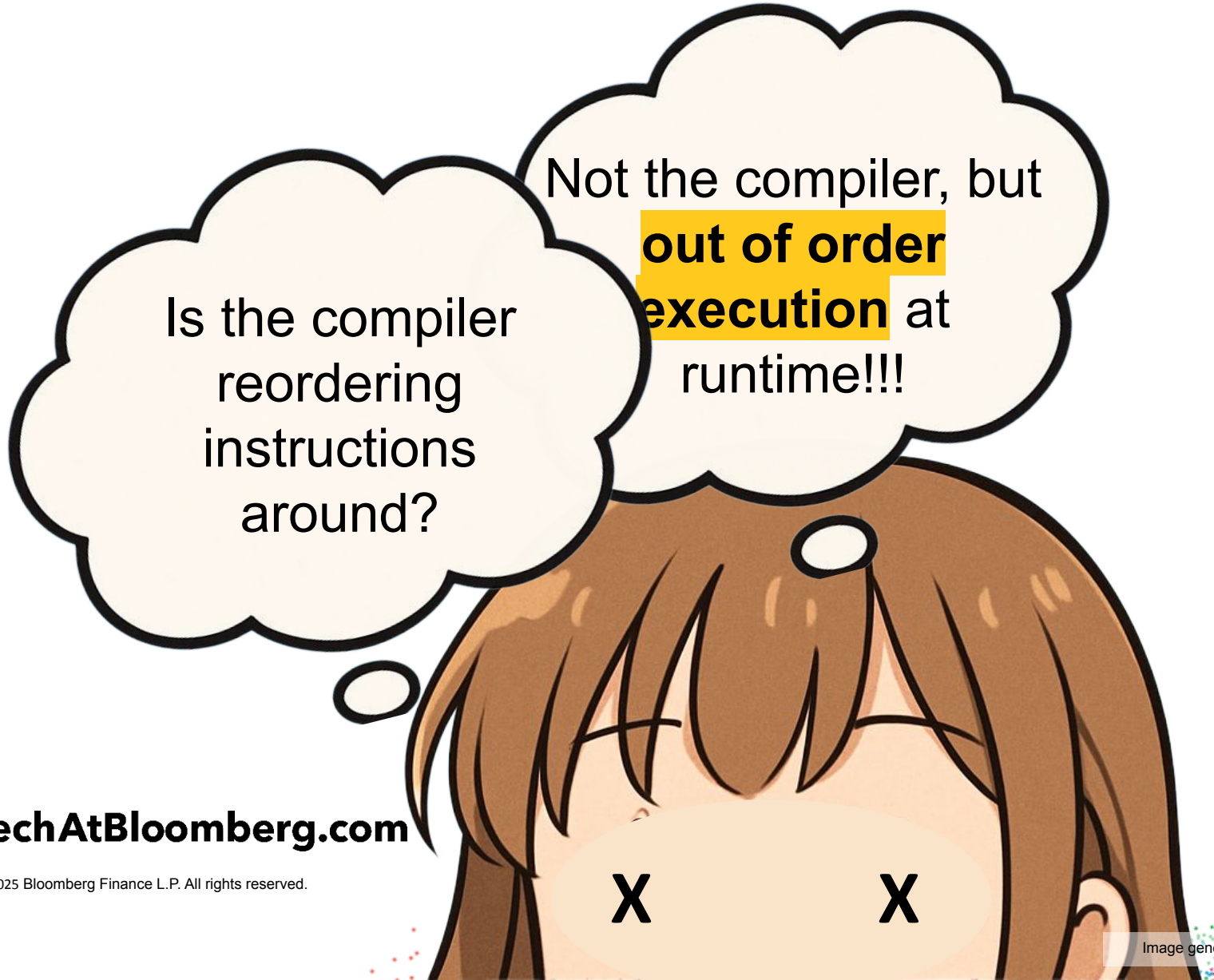
Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Inconsistent!! Even though they aren’t too different!

In terms of cycles!

Run	Avg Cycles A	Avg Cycles B	Winner
1	51.19M	51.84M	**B**
2	58.94M	59.81M	**A**
3	59.24M	59.41M	**A**
4	58.79M	56.81M	**B**

Even Then - No Consistent Results!



TechAtBloomberg.com

Bloomberg

Engineering

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Out of Order Execution…

```
Assign values to vector x;    // FAST
heavyWork();                  // SLOW
Use values in vector x;       // FAST
```

P.s. you can't see this directly in assembly since it is done by the CPU for **independent** instructions.

Driven by the CPU's:

- **Instruction scheduler**
- **Reservation stations**
- **Reorder buffer (ROB)**
- **Dependency tracking logic**

OoO will kick in … depending on the platform

x86/x86-64 (Intel & AMD)
OoO happens!

But not for legacy/older CPUs like Intel 486 & Pentium (original).

PowerPC
OoO happens!

RISC-V
Depends on implementation

MIPS
Not for many embedded MIPS chips (E.g. MIPS32)

ARM
OoO for Cortex-A72, A75, A76, A78, X1, X2; Apple M1/M2 (ARM64); Cortex-A510 (ARMv9)!

Not for embedded microcontrollers, like the Cortex-A53, A55 cpu lines (energy efficient cores)

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

How to test the OoO hypothesis?

```
std::vector<int> x = {1, 2, 3, 4, 5, …}; // With benchmark::DoNotOptimize(x);

heavyWork();

// Use values in vector x
```

```
heavyWork();

std::vector<int> x = {1, 2, 3, 4, 5, …};  // With benchmark::DoNotOptimize(x);

// Use values in vector x
```

Bloomberg

Engineering

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

How to test the OoO hypothesis? Memory fences!

```
std::vector<int> x = {1, 2, 3, 4, 5, …}; // With benchmark::DoNotOptimize(x);
_mm_mfence();
heavyWork();
_mm_mfence();
// Use values in vector x
```

```
heavyWork();
_mm_mfence();
std::vector<int> x = {1, 2, 3, 4, 5, …};  // With benchmark::DoNotOptimize(x);
_mm_mfence();
// Use values in vector x
```

Bloomberg

Engineering

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Consistent results, finally!

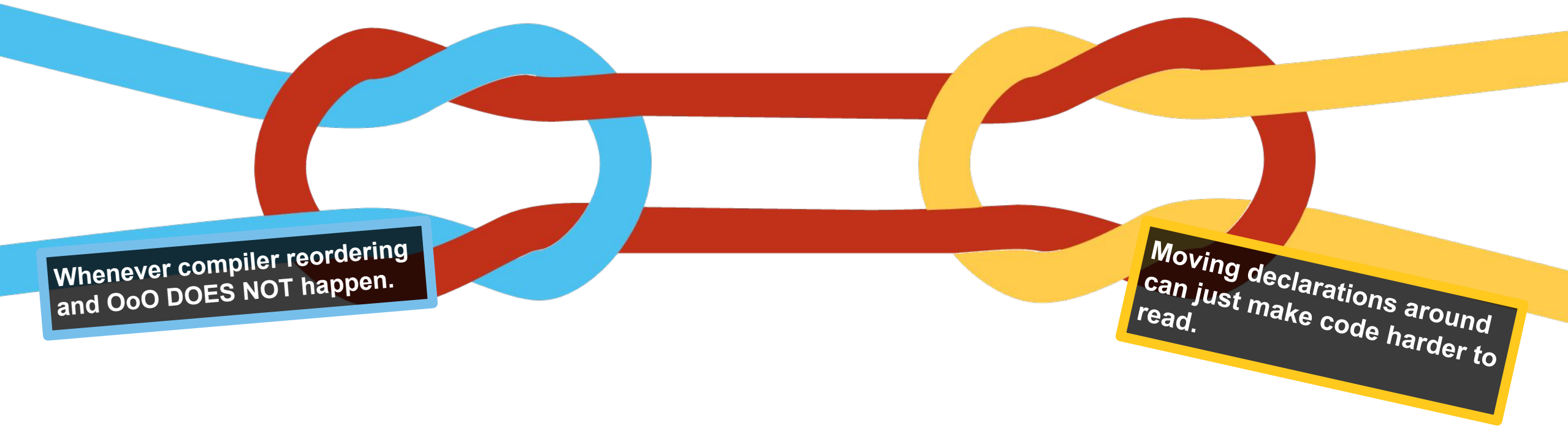
Run	Avg Cycles A	Avg Cycles B	Winner
1	63.74M	59.70M	B
2	65.64M	64.07M	B
3	66.67M	60.84M	B
4	64.09M	62.03M	B

B faster than A

(Irrespective of which is run first)

But more clock cycles than before :’)

Variable Order Matters*



TechAtBloomberg.com

Bloomberg

Engineering

Agenda – The Case File: What We'll Be Investigating Today

TechAtBloomberg.com

Agenda – The Case File: What We'll Be Investigating Today

Cache Me? Maybe NOT!

Assume NO out of order execution.

You read vector A.

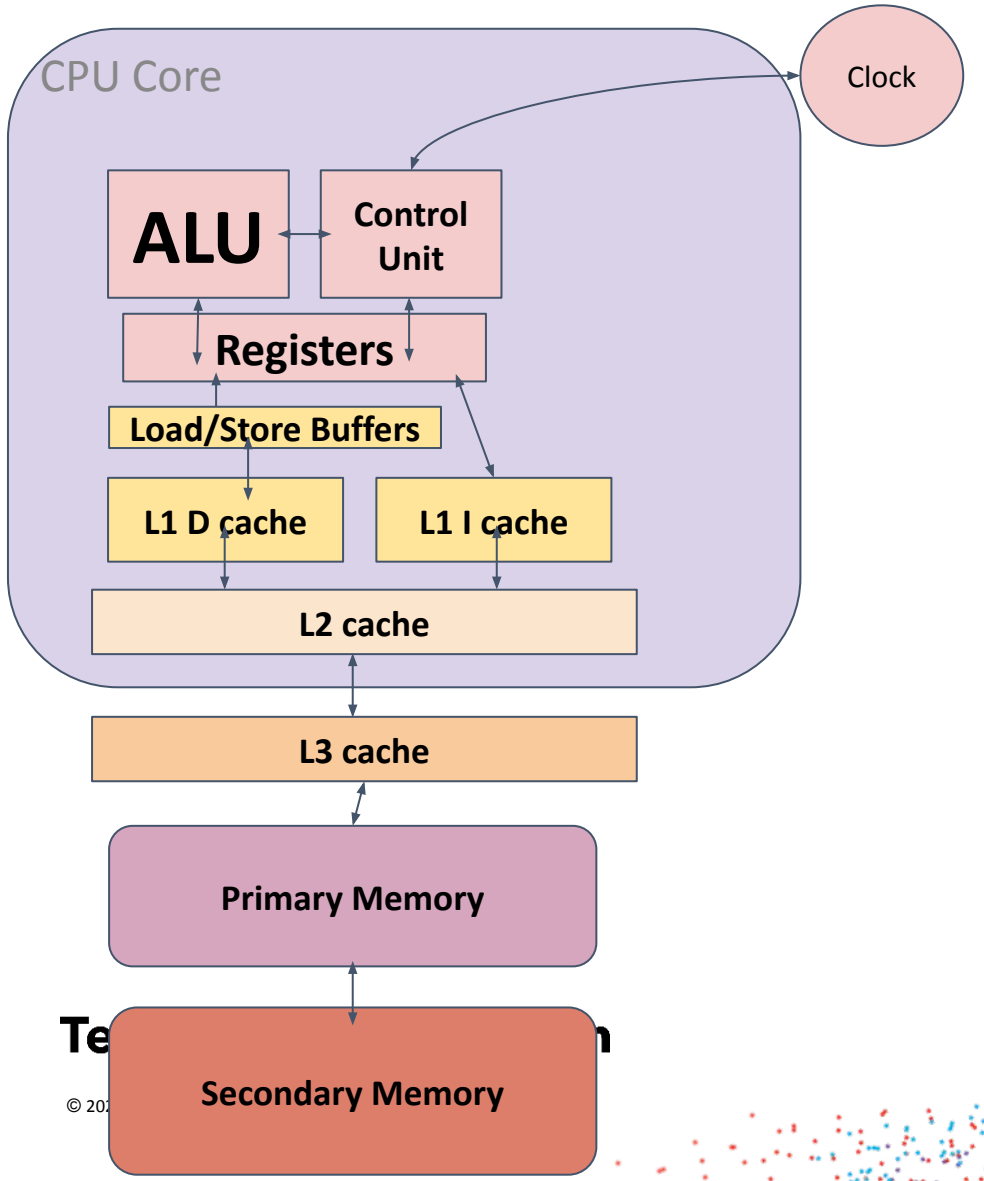
You then write a lot of unrelated data to vector B.

You then read vector A again.

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

P.S. After a sleep of 2 minutes, I did print out the values of A and B so that the compiler doesn't optimize them away.

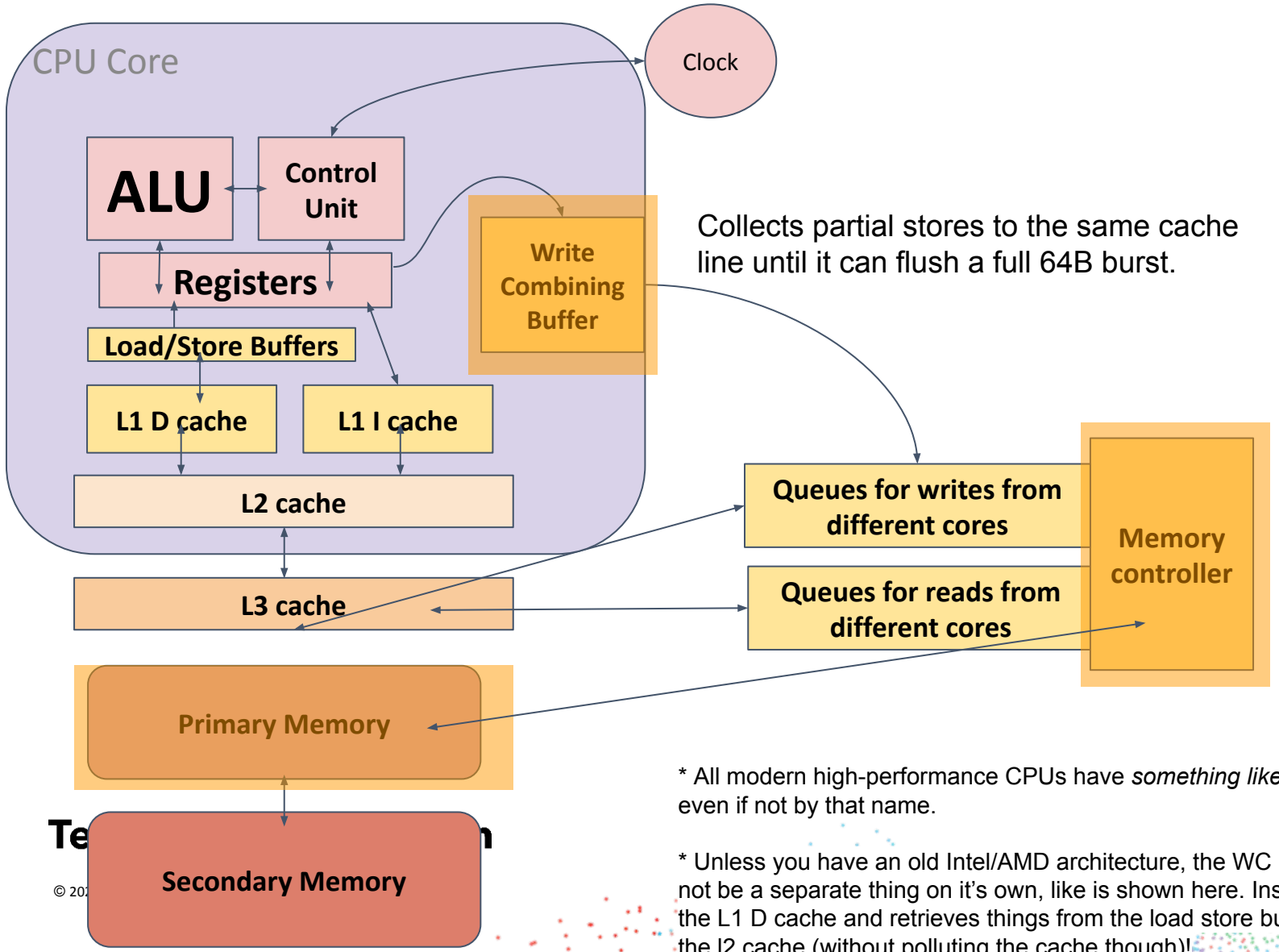
First Rule of being a Detective? Trust No One!



Bloomberg

Engineering

Bypass the Cache!



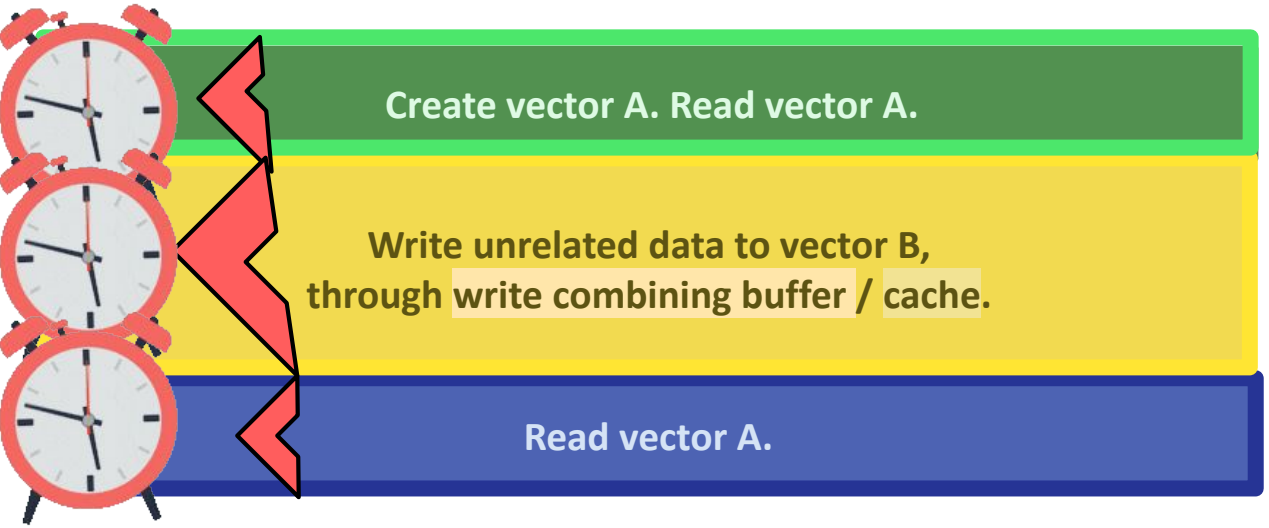
* All modern high-performance CPUs have *something like* a write combining (WC) buffer, even if not by that name.

* Unless you have an old Intel/AMD architecture, the WC Buffer is most likely going to not be a separate thing on it's own, like is shown here. Instead, it is often integrated with the L1 D cache and retrieves things from the load store buffers, and sends them through the l2 cache (without polluting the cache though)!

Bloomberg

Engineering

Experimenting, Since We Trust No One



Source: Giphy (https://media4.giphy.com)

The Fastest in the Lineup?

Overall timing wise, streaming with the write combining buffer is faster!

OoO / SIMD:	63% faster than SIMD write to cache
No OoO / SIMD:	66% faster than SIMD write to cache
OoO / Scalar:	38% faster than write to cache
No OoO / Scalar:	36% faster than write to cache

Experimenting, Since We Trust No One

Create vector A. Read vector A.

Write unrelated data to vector B,
through write combining buffer / cache.

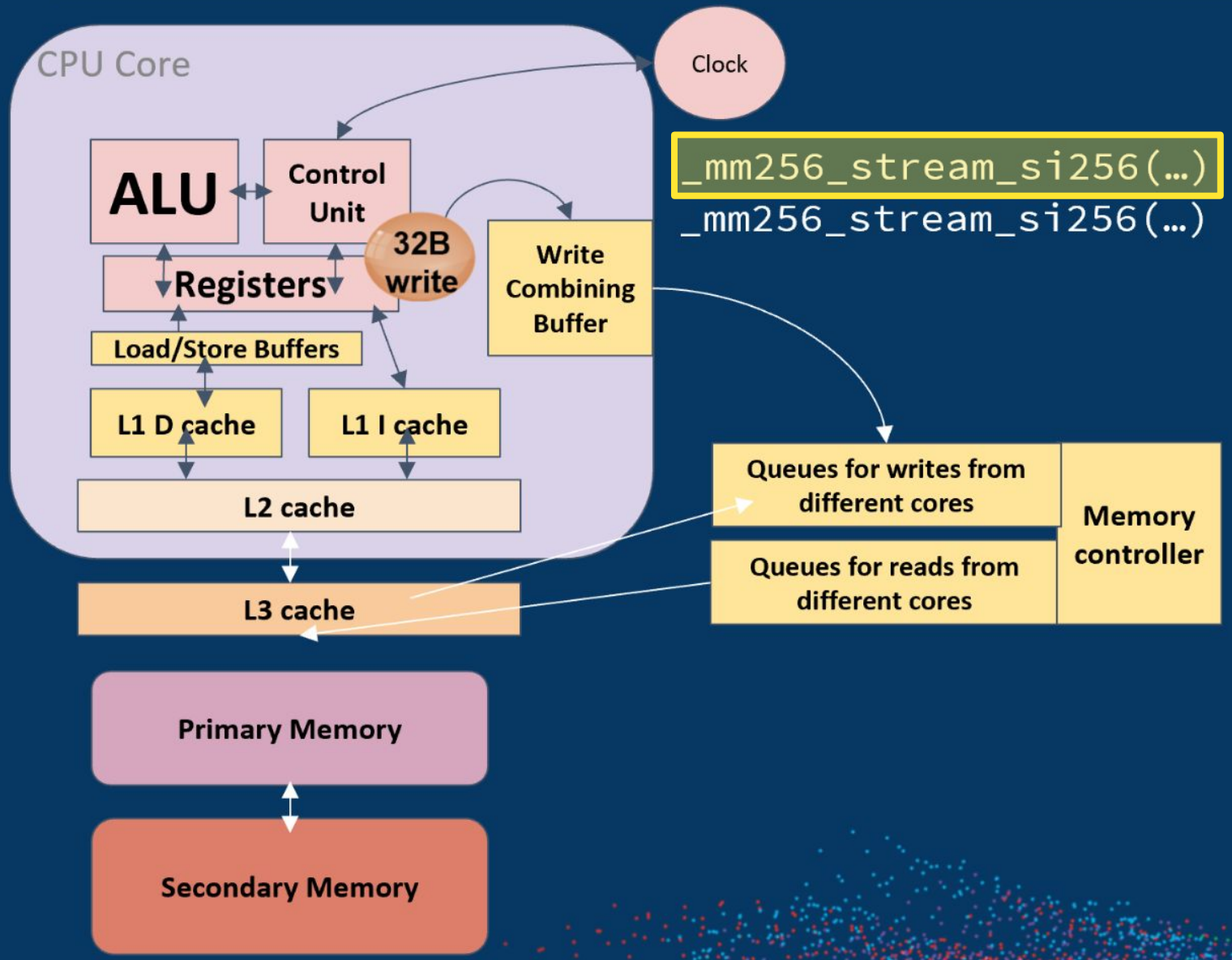
Read vector A.

Main source of the speed up!

Source: Vecteezy (www.vecteezy.com)

Source: Giphy (https://media4.giphy.com)

WC Buffer Case

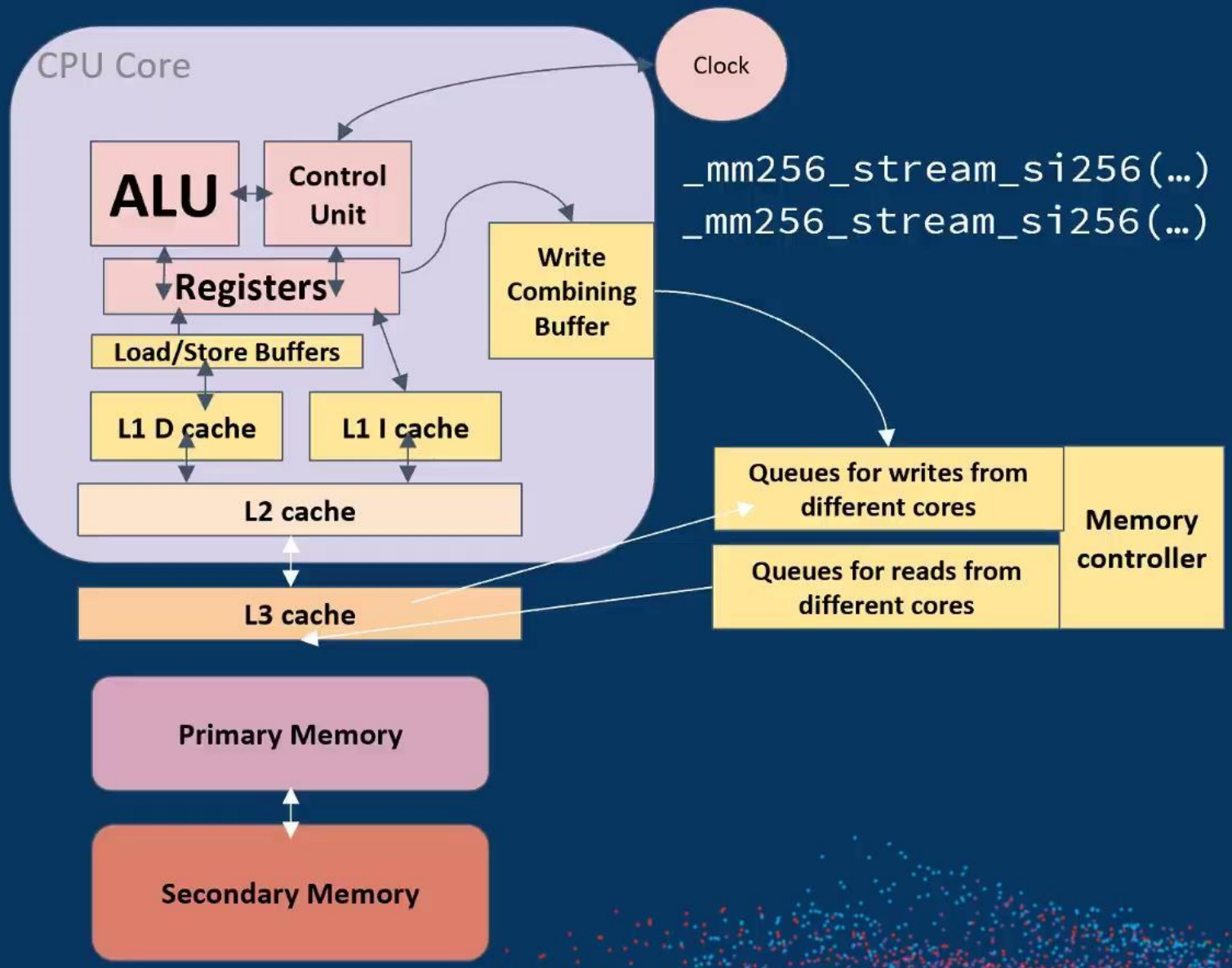



```
_mm256_stream_si256(…)
_mm256_stream_si256(…)
```

I used SIMD Streaming instructions.

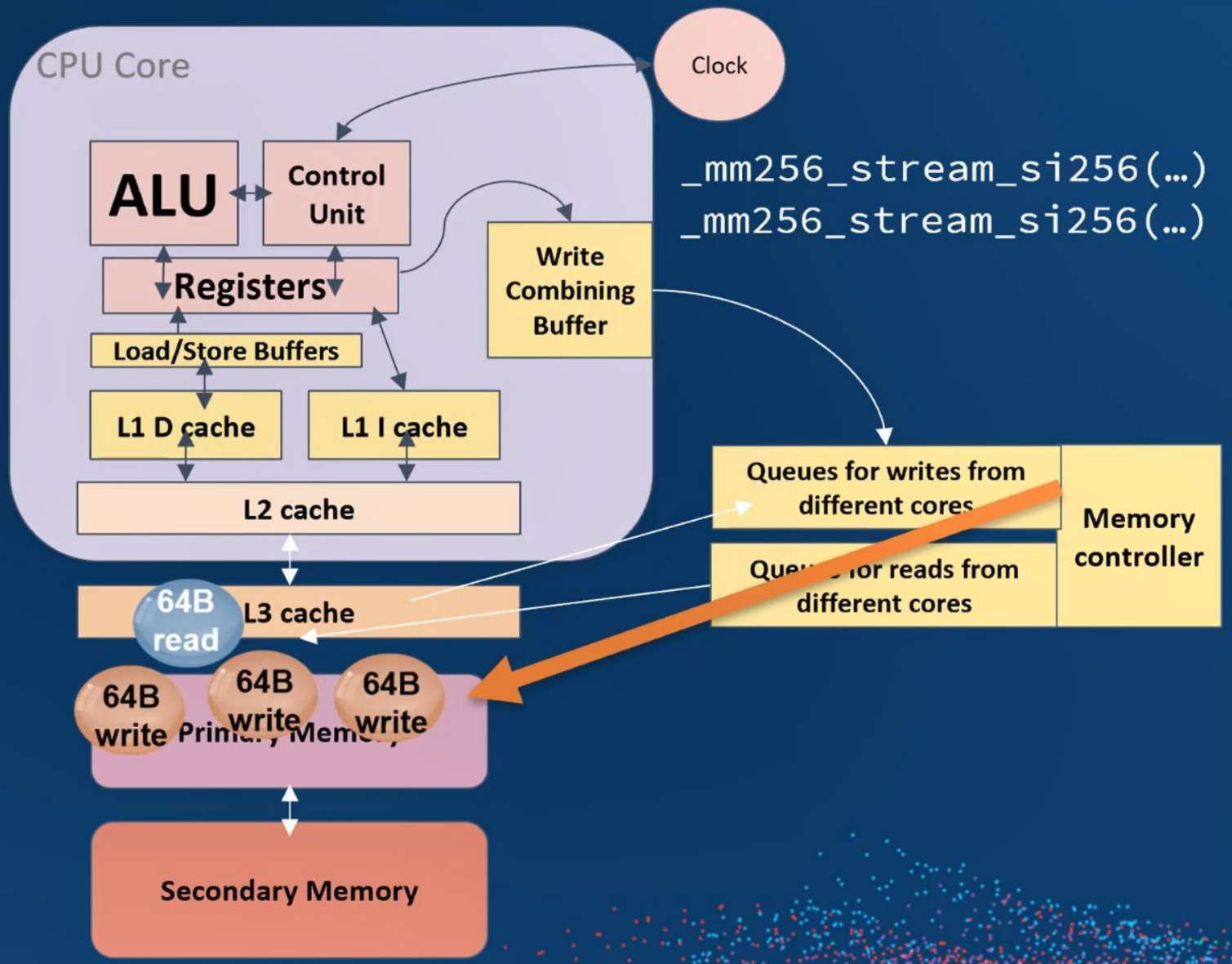
You can use non-SIMD ones too, this would just mean each instruction writes a smaller amount of data to the register.

WC Buffer Case



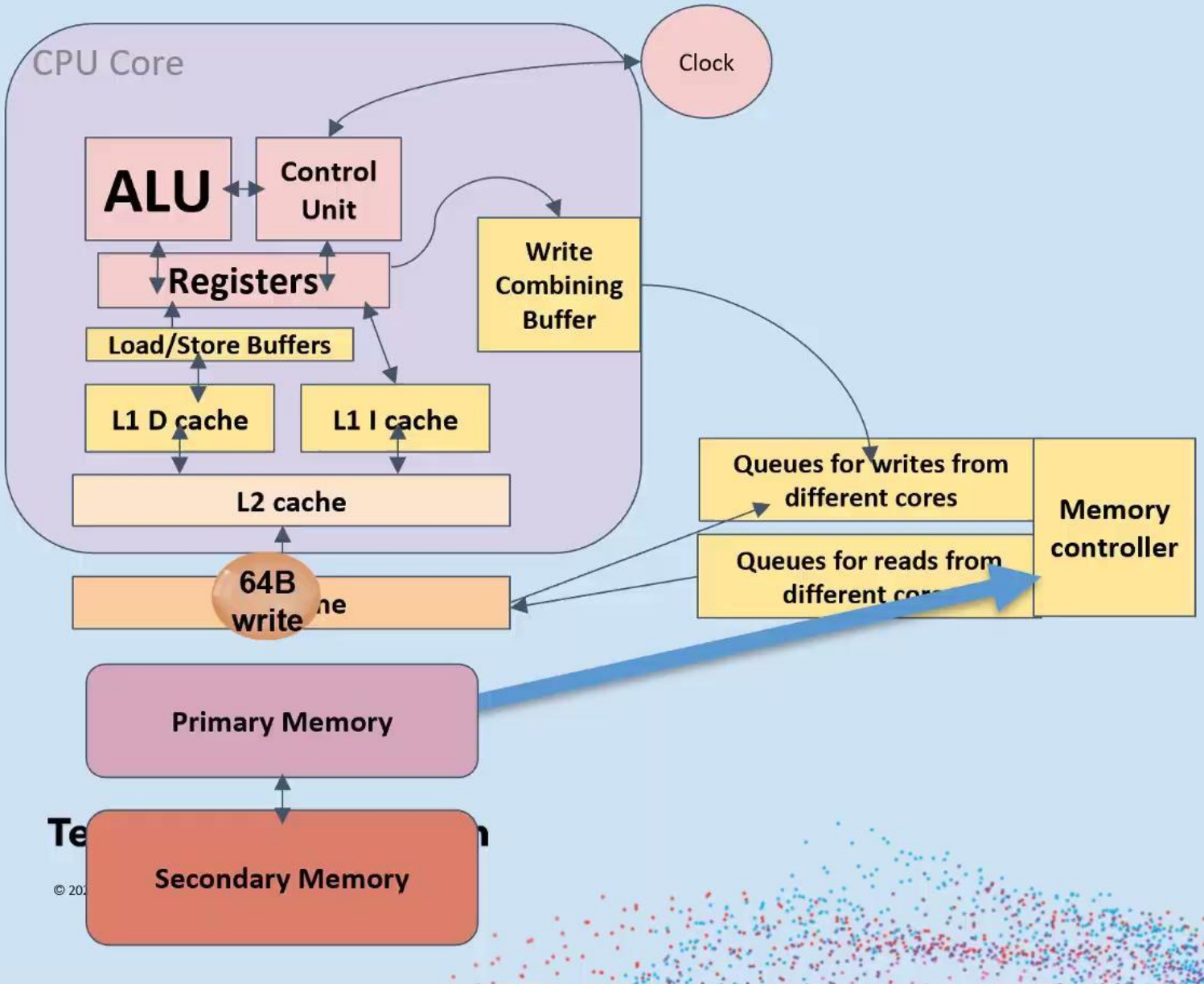
1. Stores go to write combining (WC) buffers.
2. WC buffers flush full 64-B bursts.
3. Memory controller writes these bursts into memory.
4. When the last write has made it into the write combinging buffer, there is a read of memory even though writes may still be in the memory controller and/or WC buffer.

WC Buffer Case



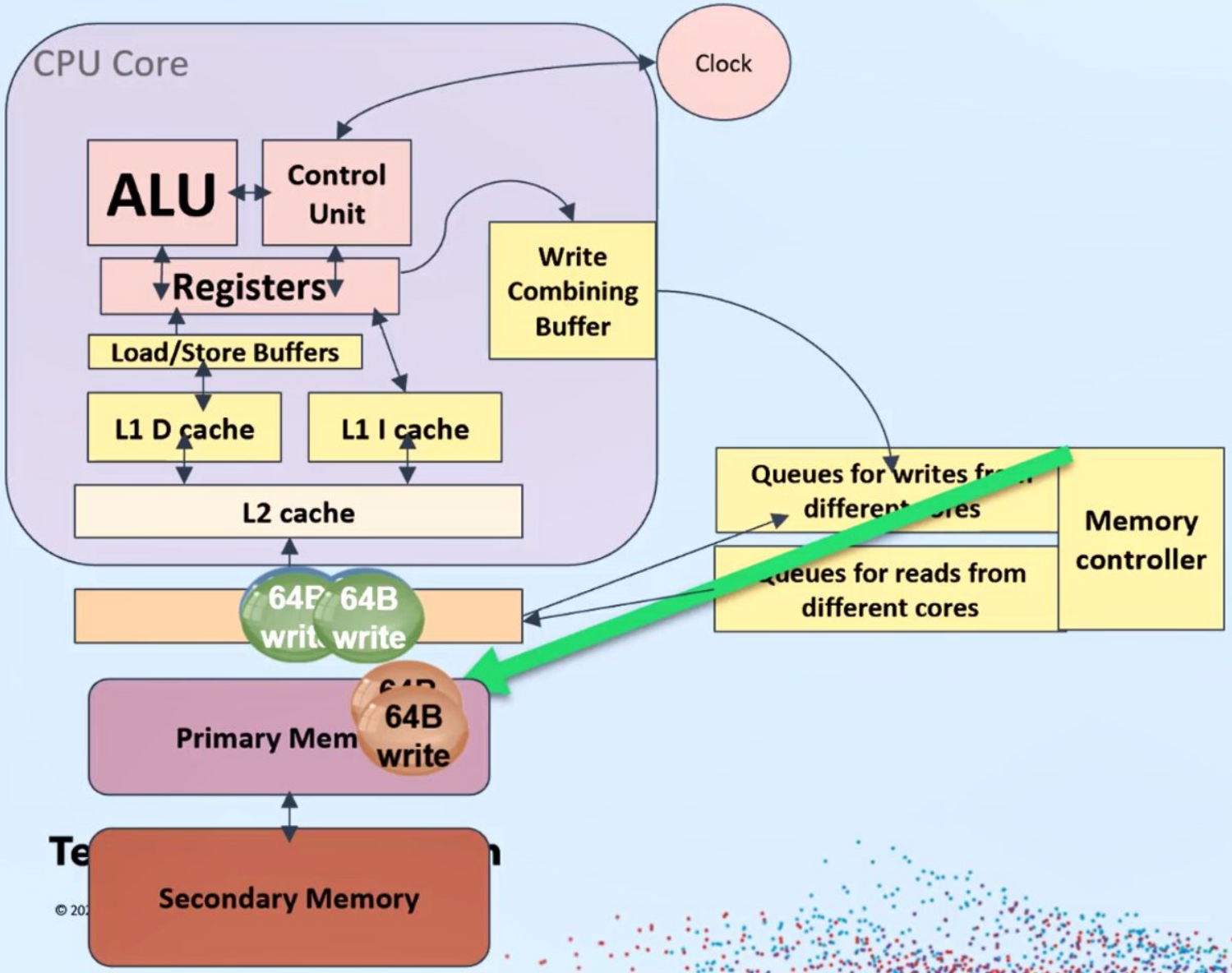
1. Stores go to write combining (WC) buffers.
2. WC buffers flush full 64-B bursts.
3. Memory controller writes these bursts into memory.
4. When the last write has made it into the write combinging buffer, there is a read of memory even though writes may still be in the memory controller and/or WC buffer.

Cache Case – Read for Ownership!



1. When you have to write to a line, bring it into the cache first (read for ownership). We didn't have to do this for WC Buffer writes!
2. Mark data modified when cache line is written to.
3. When need cache space for new cache line, evict old line.

Cache Case – Read for Ownership!



1. When you have to write to a line, bring it into the cache first (read for ownership). We didn't have to do this for WC Buffer writes!
2. Mark data modified when cache line is written to.
3. When need cache space for new cache line, evict old line.

Experimenting, Since We Trust No One

Create vector A. Read vector A.

Write unrelated data to vector B,
through write combining buffer / cache.

Read vector A.

Source: Giphy (https://media4.giphy.com)

Experimenting, Since We Trust No One

Create vector A. Read vector A.

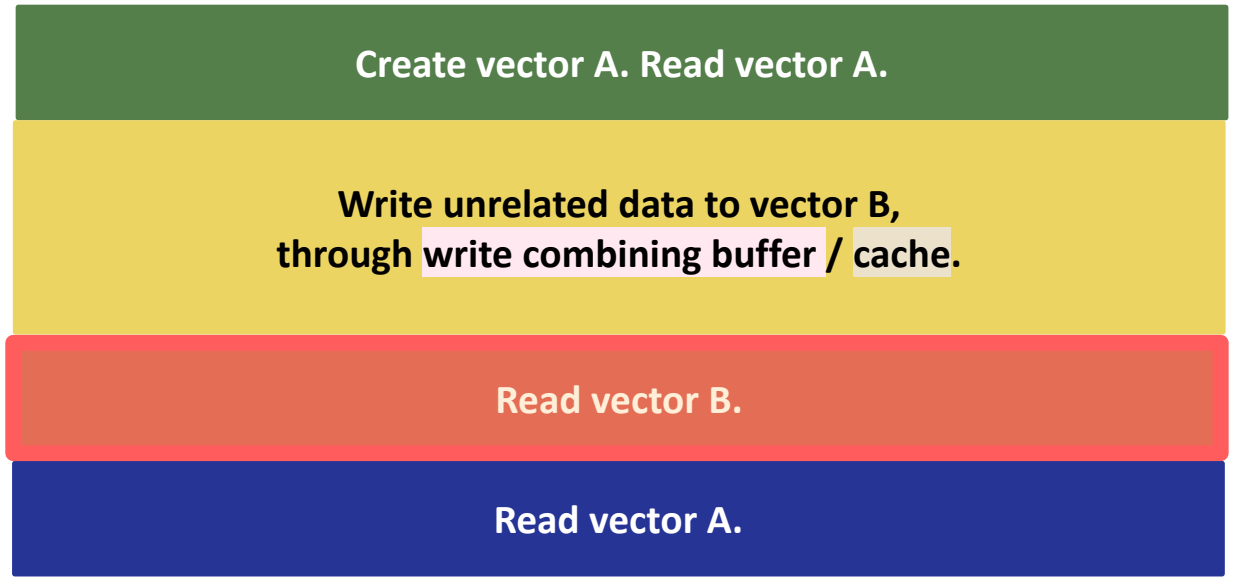
Write unrelated data to vector B,
through write combining buffer / cache.

Read vector A.

What if we read vector B soon after?

Vector B [1, 1, …, 1]

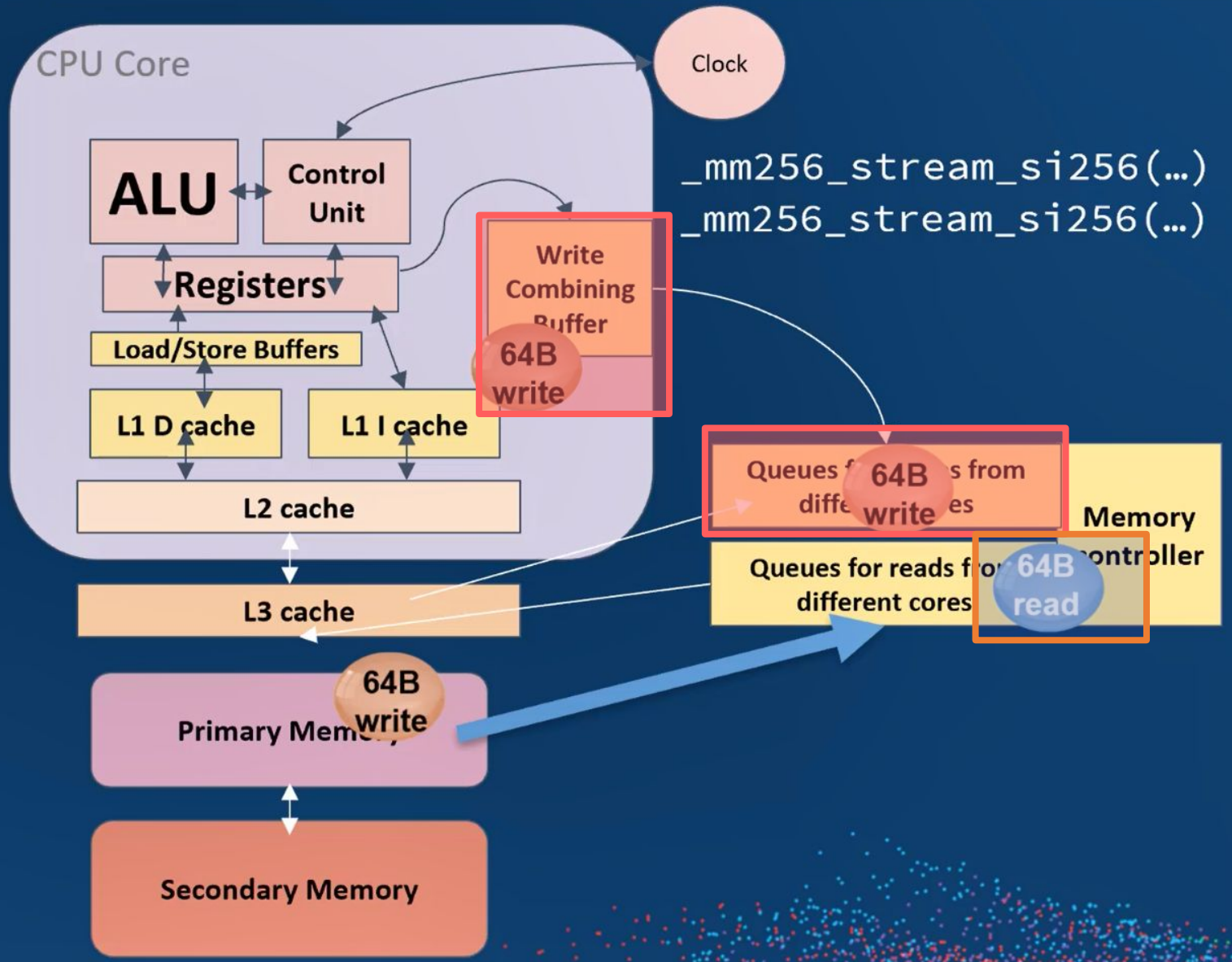
…



[42, 42, …, 42]

Last element of vector B (with WC Buffer)**?**

WC Buffer Case

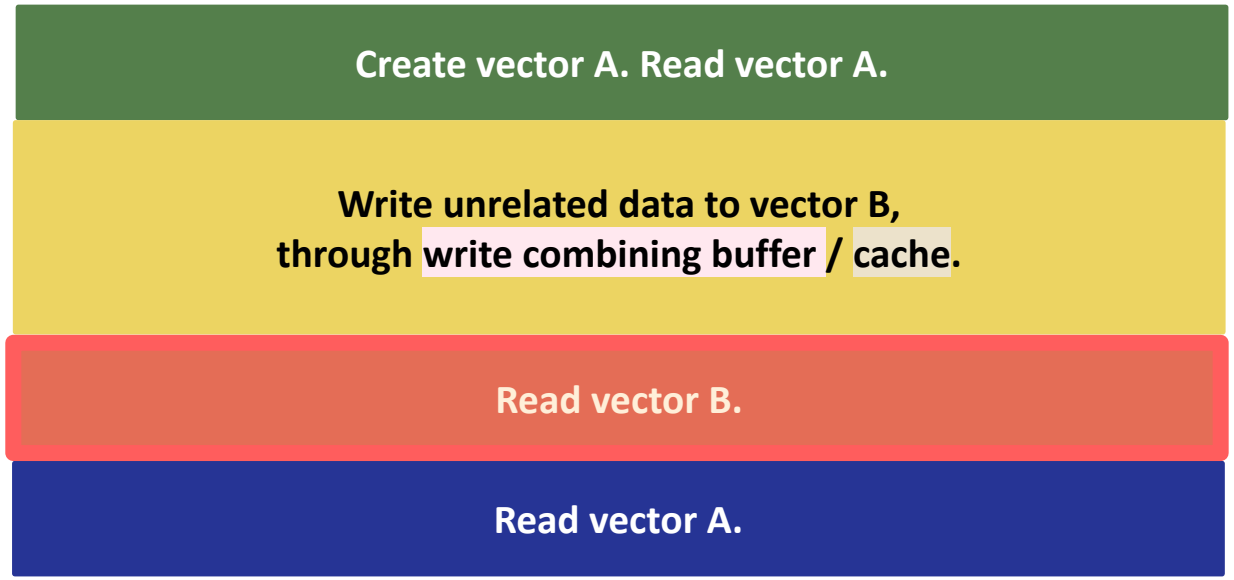


1. Stores go to write combining (WC) buffers.
2. WC buffers flush full 64-B bursts.
3. Memory controller writes these bursts into memory.
4. When the last write has made it into the write combinging buffer, there is a read of memory even though writes may still be in the memory controller and/or WC buffer.

What if we read vector B soon after?

Vector B [1, 1, …, 1]

…



[42, 42, …, 42]

Last element can be either 1 or 42 (with WC Buffer)!

Can have a stale value!
IF running on multiple cores or specific architectures (like ARM)

Source: Vecteezy (www.vecteezy.com)

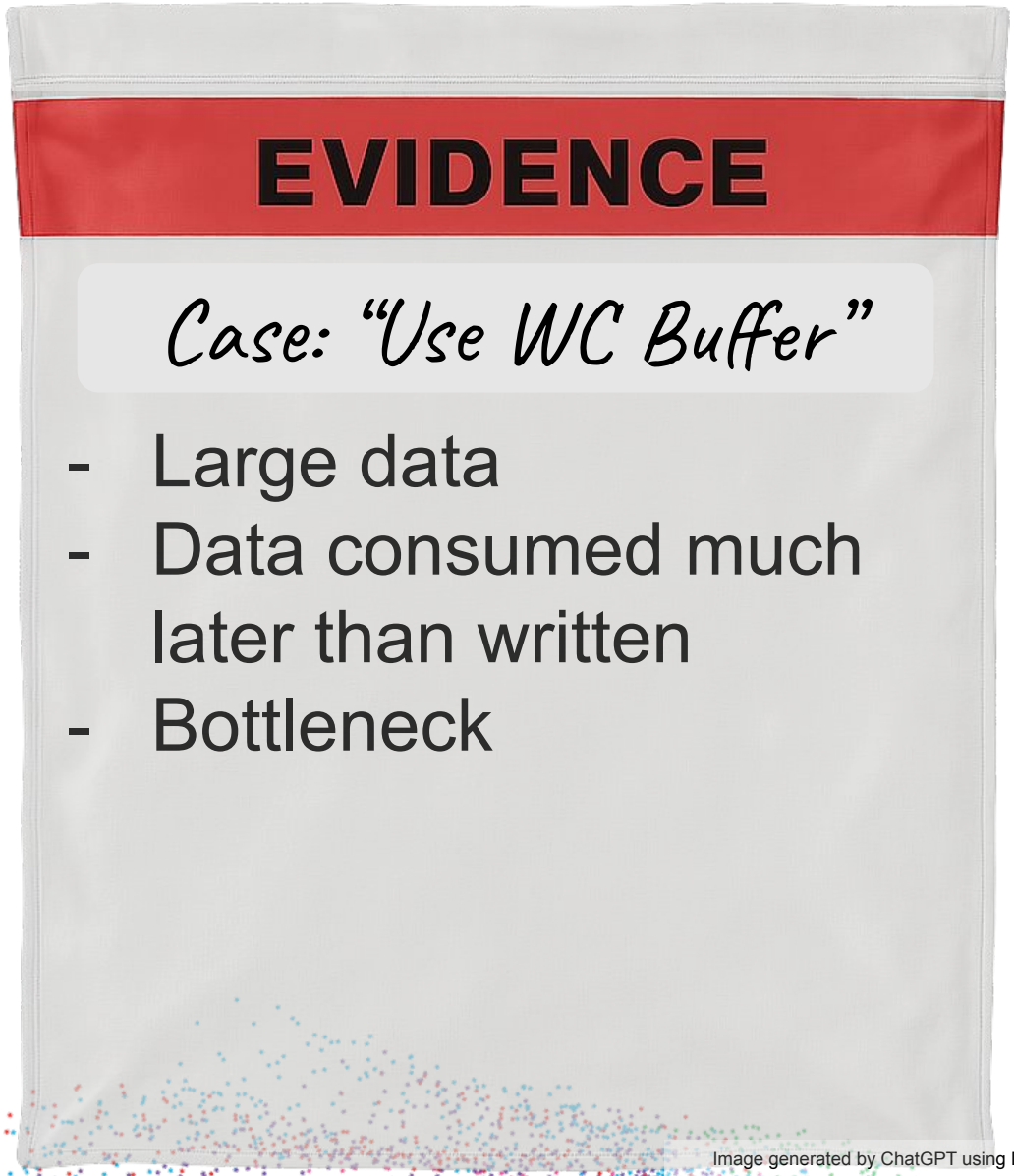
Fixed with a Fence!

OoO / SIMD:	~~63%~~	71%	faster than SIMD write to cache
No OoO / SIMD:	~~66%~~	67%	faster than SIMD write to cache
OoO / Scalar:	~~38%~~	42%	faster than write to cache
No OoO / Scalar:	~~36%~~	40%	faster than write to cache

Ensures all the writes drain before the read! And with fence, performance timings mostly faster (in case where you don't read B)! A fence lets the core batch and drain the WC buffers cleanly, giving bigger, more efficient bursts.

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

When to Use WC Buffer?



If you have a read of the data right after though, caches most likely better than WC buffer.

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.

Tools to Profile the Behavior of your Cache

Tool	Platform	Real-Time	Cache Metrics	Notes
perf	Linux	✅	✅	Most flexible CLI tool
Intel VTune	Linux/Win	✅	✅✅✅	Deepest insights (L1→LLC, TLB)
AMD uProf	Linux/Win	✅	✅	Best for AMD CPUs
Cachegrind	Linux/macOS	❌ (sim)	✅	Great for static analysis
Visual Studio Profiler	Windows	✅	✅ (limited)	Integrated with IDE
Instruments (with xcode)	macOS	✅	⚠️ Limited	Visual, but lacks deep hardware data
LIKWID	Linux	✅	✅	Lightweight and easy

In Summary - You've CACHED it!

C ontiguous memory understanding

A voiding costly cache misses is beneficial

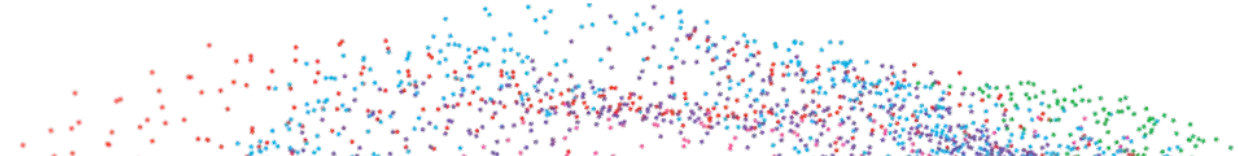
C ache friendly coding patterns

H ardware insights: OoO execution, core isolation, etc.

E quipping yourself with the right tools (RTCPU vs. chrono, etc.)

D on't need to use the cache in certain cases

Please trust other people…
but always verify things yourself to build understanding!



Some other great related talks!

- CppCon 2017: Chandler Carruth “Going Nowhere Faster”
- CppCon 2016: Timur Doumler “Want fast C++? Know your hardware!"
- CppCon 2014: Mike Acton "Data-Oriented Design and C++"
- C++ on Sea 2024: Björn Fahller “C++ Cache Friendly Data + Functional + Ranges = ❤️”
- Meeting C++ 2018: Jonathan Müller “Writing Cache Friendly C++”
  - (also his CppCon 2025 talk with the same title)
- CppCon 2025: Vittorio Romeo “More Speed & Simplicity”

Bloomberg

Engineering

Hope your attention cached this talk ... time to get evicted!

linkedin.com/in/michellefae/

michellefae.github.io/

Special Thanks:

Chris Cotter, Dietmar Kühl, Sherry Sontag, Andrei Alexandrescu, Laura Savino, Jess Winer, Alex Lo, and Bloomberg's C++ Guild for helping me rehearse my talk!

For some of the code used in the talk!

Bloomberg Engineering

Image generated by ChatGPT using DALL·E (OpenAI) based on user-provided description.