ACCU 2021
VIRTUAL EVENT

Bloomberg
Engineering

undo

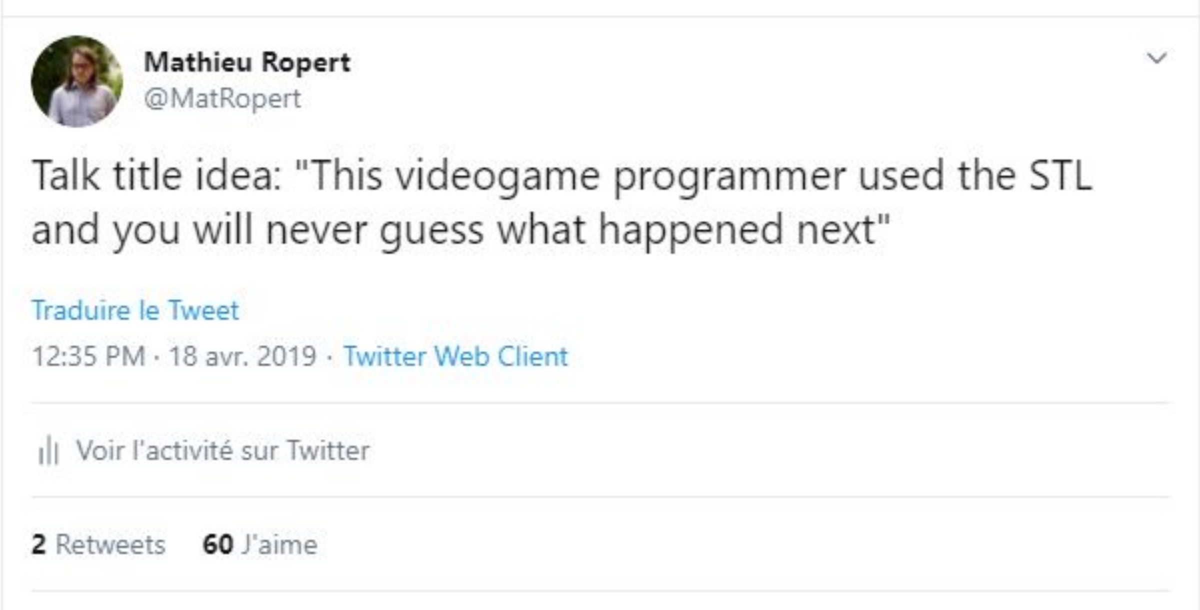
mosaic
CONSULTANTS TO FINANCIAL SERVICES

This Videogame Developer Used the STL and You'll Never Guess What Happened Next

Mathieu Ropert

This Videogame Programmer Used the STL

(and You Will Never Guess What Happened Next)



Mathieu Ropert
@MatRopert

Talk title idea: "This videogame programmer used the STL and you will never guess what happened next"

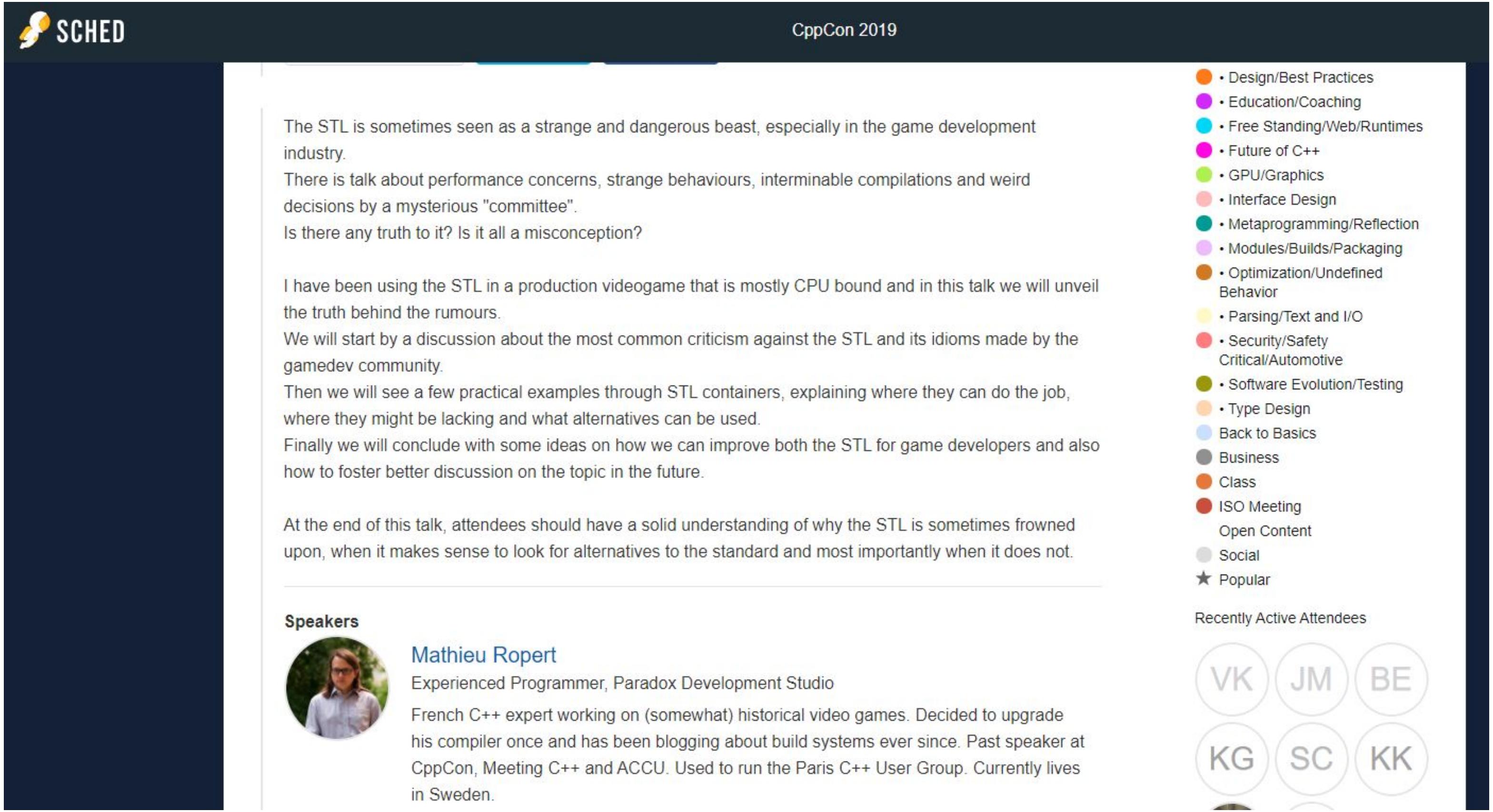
Traduire le Tweet

12:35 PM · 18 avr. 2019 · Twitter Web Client

Voir l'activité sur Twitter

2 Retweets **60** J'aime

STL and videogames



The STL is sometimes seen as a strange and dangerous beast, especially in the game development industry.
There is talk about performance concerns, strange behaviours, interminable compilations and weird decisions by a mysterious "committee".
Is there any truth to it? Is it all a misconception?

I have been using the STL in a production videogame that is mostly CPU bound and in this talk we will unveil the truth behind the rumours.
We will start by a discussion about the most common criticism against the STL and its idioms made by the gamedev community.
Then we will see a few practical examples through STL containers, explaining where they can do the job, where they might be lacking and what alternatives can be used.
Finally we will conclude with some ideas on how we can improve both the STL for game developers and also how to foster better discussion on the topic in the future.

At the end of this talk, attendees should have a solid understanding of why the STL is sometimes frowned upon, when it makes sense to look for alternatives to the standard and most importantly when it does not.

Speakers

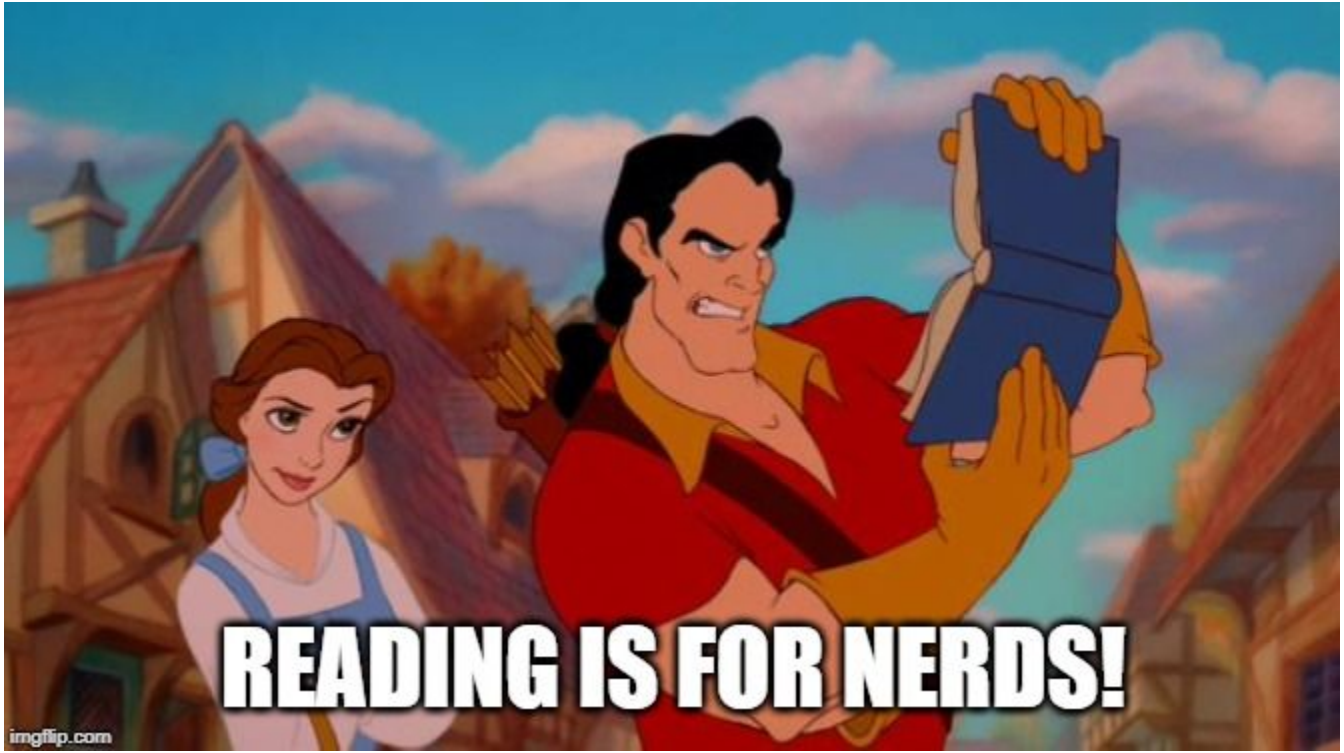
Mathieu Ropert
Experienced Programmer, Paradox Development Studio

French C++ expert working on (somewhat) historical video games. Decided to upgrade his compiler once and has been blogging about build systems ever since. Past speaker at CppCon, Meeting C++ and ACCU. Used to run the Paris C++ User Group. Currently lives in Sweden.

- ● • Design/Best Practices
- ● • Education/Coaching
- ● • Free Standing/Web/Runtimes
- ● • Future of C++
- ● • GPU/Graphics
- ● • Interface Design
- ● • Metaprogramming/Reflection
- ● • Modules/Builds/Packaging
- ● • Optimization/Undefined Behavior
- ● • Parsing/Text and I/O
- ● • Security/Safety Critical/Automotive
- ● • Software Evolution/Testing
- ● • Type Design
- ● Back to Basics
- ● Business
- ● Class
- ● ISO Meeting
- Open Content
- ● Social
- ★ Popular

Recently Active Attendees

VK JM BE
KG SC KK

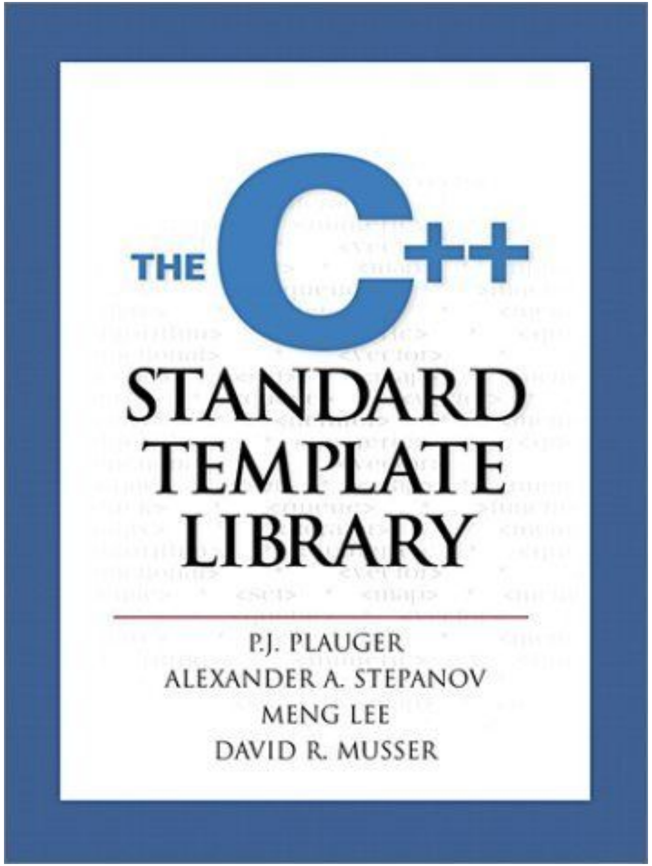

READING IS FOR NERDS!
imgflip.com

STL in videogames considered harmful

Picture unrelated

This Videogame Programmer Used the STL

(and You Will Never Guess What Happened Next)



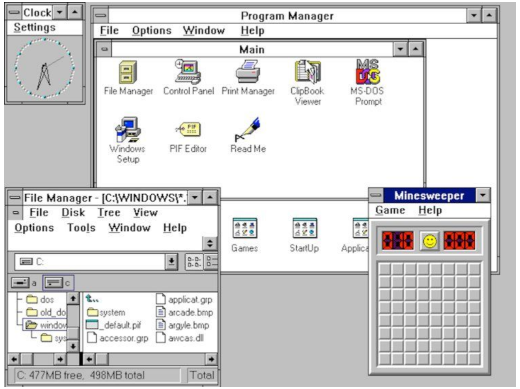
THE C++

STANDARD TEMPLATE LIBRARY

P.J. PLAUGER
ALEXANDER A. STEPANOV
MENG LEE
DAVID R. MUSSER

Standard Template Library

- Proposed in 1993 by Alex Stepanov
- Adopted in 1994
- Offers a set of generic containers and algorithms for C++



ARTS FESTIVALS 'Times' guide to the best of British

EDWARD LEIGH How the Tories can win on Europe

SNEEZE SEASON Help is at hand for hay fever victims

30P

THE TIMES

TUESDAY MAY 3 1994

Triumphant Mandela 'free at last'

De Klerk concedes defeat then is told by successor: 'This is your victory, too'

Shots and dancers greet South Africa's new dawn

Top jockey critically ill after fall

Tory disarray over MP's 'sack ministers' call

Darby gives United title

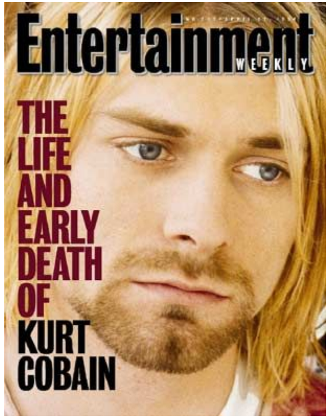
Knight flies from Spain to face £7m raid charge

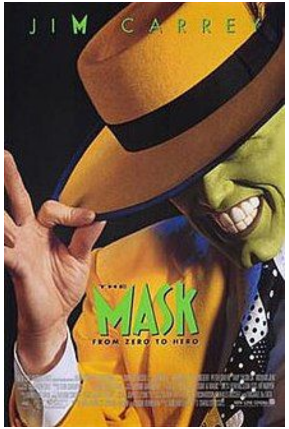
Hendry wins

Platform blast

YOU TAKES YOUR CHOICE. YOU SAVES YOUR MONEY.

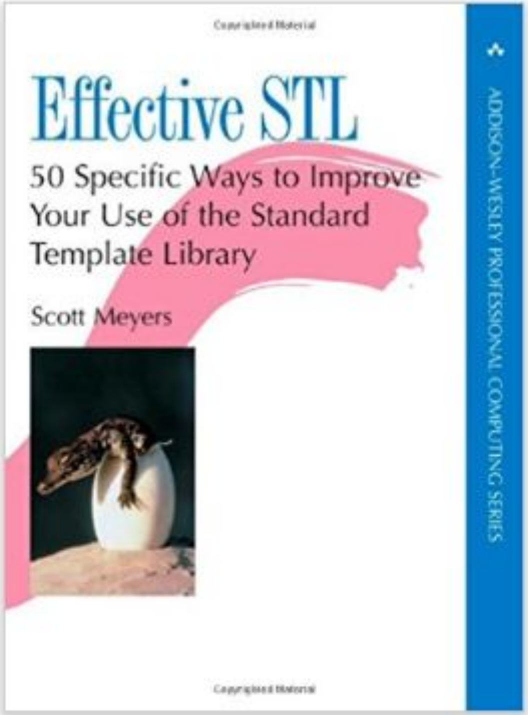
Where Talk is Cheaper





Lillehammer'94

1994



Copyrighted Material

Effective STL

50 Specific Ways to Improve Your Use of the Standard Template Library

Scott Meyers

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Copyrighted Material

Province of Value Queries
Province of Property Queries
Province of 2-Ranges Properties
Glorious County of Algos on Sets
Lonely Islands
Lands of Queries
Peninsula of Raw Memory
Territory of Movers
Value Searchers
Province of Research
Relative Value Searchers
Island of Structure Changers
Land of Value Modifiers
Secret Runes
Lands of Permutations
Fluent {C++}
fluentcpp.com
The World of C++ STL Algorithms
Fluent {C++}

“We don’t use the STL here”

Anonymous videogame programmer, 2019

Hello!

*I am **Mathieu Ropert***

I’m a Tech Lead at Paradox Development Studio where I make Hearts of Iron IV, Stellaris and more.

You can reach me at:

- ✉ mro@puchiko.net
- @MatRopert
- https://mropert.github.io

About this talk

- The case against STL
- STL containers in practice
- Frequently sought-after alternatives
- Performance & maintenance

About not this talk

- Allocators
- Exceptions
- Build times

1 Is the STL so bad?

Or where the criticism is coming from

Common complaints

- “STL is unfamiliar”
- “STL is not supported on platform X”
- “STL is bloated”
- “STL performance isn’t that great”

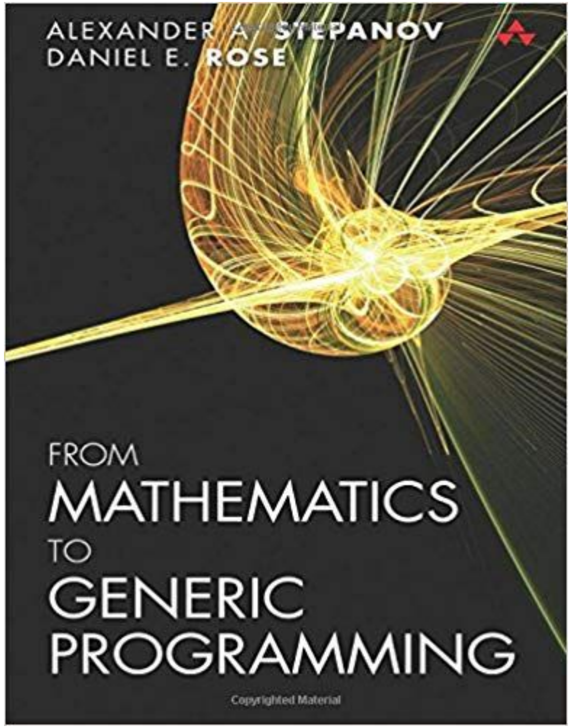
STL familiarity

- STL been around for 25 years
- Popular C++ libraries adopted the same idioms (Boost, Abseil, Intel TBB...)
- Resources teaching Containers, Iterators and Algorithms are plenty

STL familiarity

- Stepanov’s approach on decoupling containers and algorithms is based on sound research

- We might need to study and teach the principles better in schools



STL availability

- Major vendors should provide a reasonably good implementation of the STL
- As any software, they may have bugs or caveats
- Keep up with updates, report issues

STL availability

- Vendors that won't care about STL probably won't care about C++ in general
- Chances are they will have broken standard support or subpar optimizations
- Consider using open source alternatives

STL bloat

- Standard additions may feel unnecessary or unwanted

- Vendor implementations may look over-complicated for what they are trying to achieve

STL bloat?

- STL, like C++, is designed for general purpose usage
- C++ design principles dictate that unused features should not be added to the cost
- Not always possible in practice, as the cost of multiple policies grows quite fast

STL bloat?

- Vendor implementations may include additional debug features to help developers
- There is a build flag somewhere to turn them off
- Debug checks are not incompatible with optimizations

The quest for performance

- Games need to run within a timebox
- Worst case scenarios and unpredictable latency matter a lot
- Common wisdom recommends low level languages for better control over performance

The quest for performance

- STL comes with some degree of abstraction
  - Templates
  - Iterators
  - Debug / checked iterators
  - Proxy iterators

- Requires a good optimizer to yield performance

STL performance

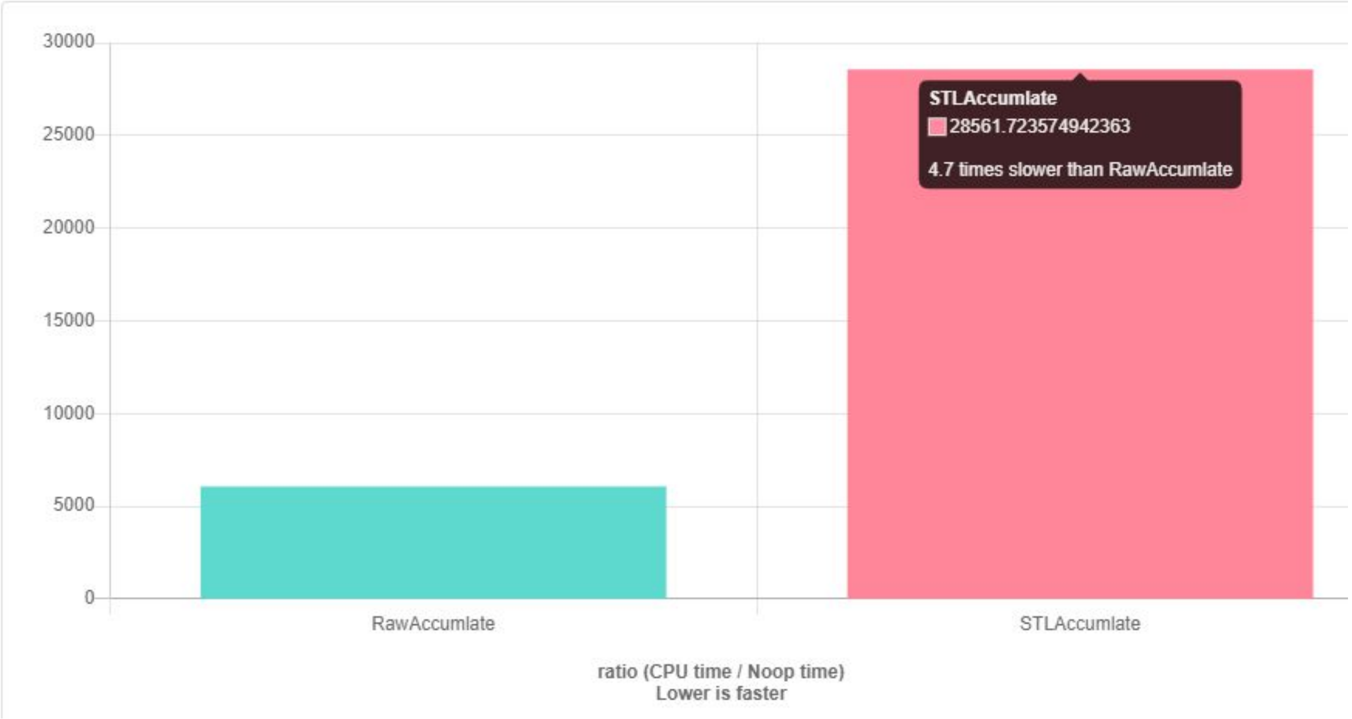
```
static void RawAccumlate(benchmark::State& state) {
    const auto v = generate_values<int>(10000);
    for (auto _ : state) {
        const int* p = v.data();
        const int sz = v.size();
        int sum = 0;
        for (int i = 0; i < sz; ++i )
            sum += p[i];

        benchmark::DoNotOptimize(sum);
    }
}
BENCHMARK(RawAccumlate);
```

STL performance

```
static void STLAccumlate(benchmark::State& state) {
   const auto v = generate_values<int>(10000);
   for (auto _ : state) {
       auto sum = std::accumulate(begin(v), end(v), 0);
       benchmark::DoNotOptimize(sum);
   }
}
BENCHMARK(STLAccumlate);
```

STL performance



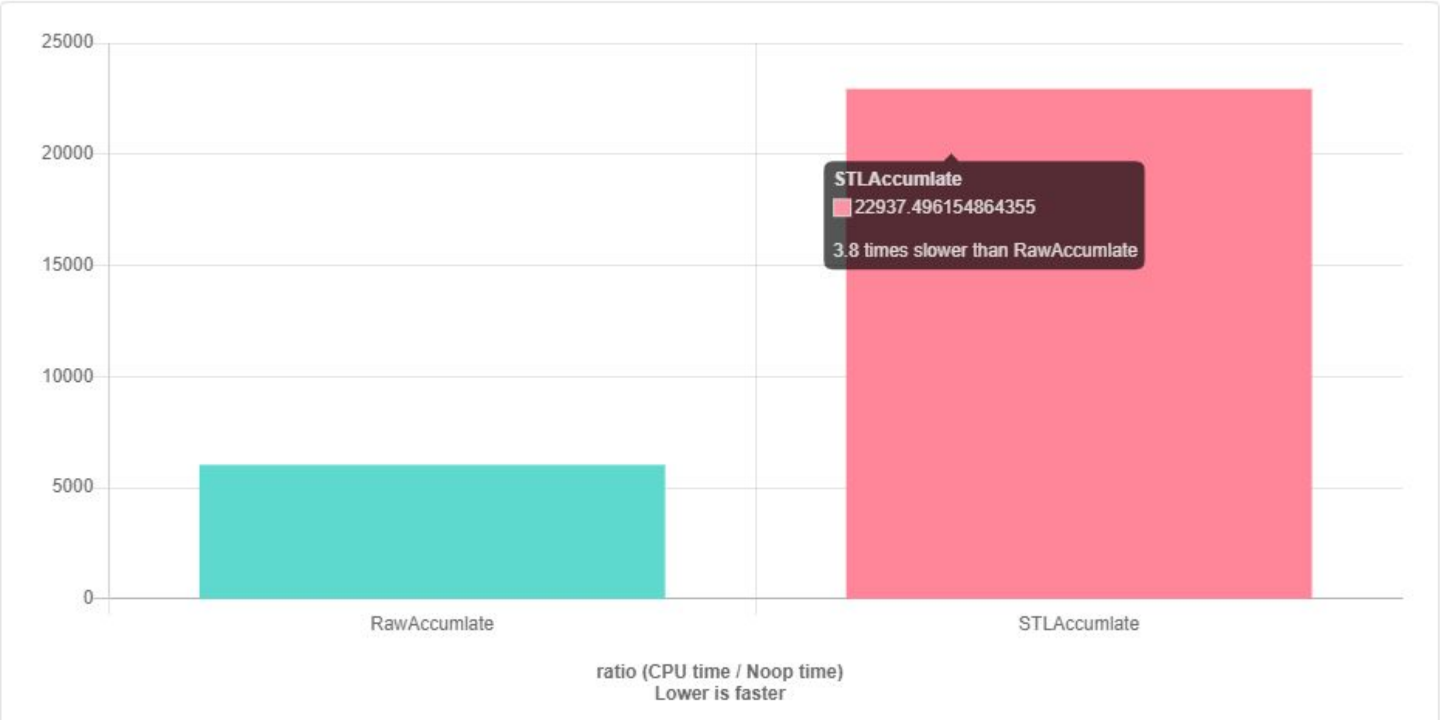
STLAccumlate

🟥 28561.723574942363

4.7 times slower than RawAccumlate

clang / libc++

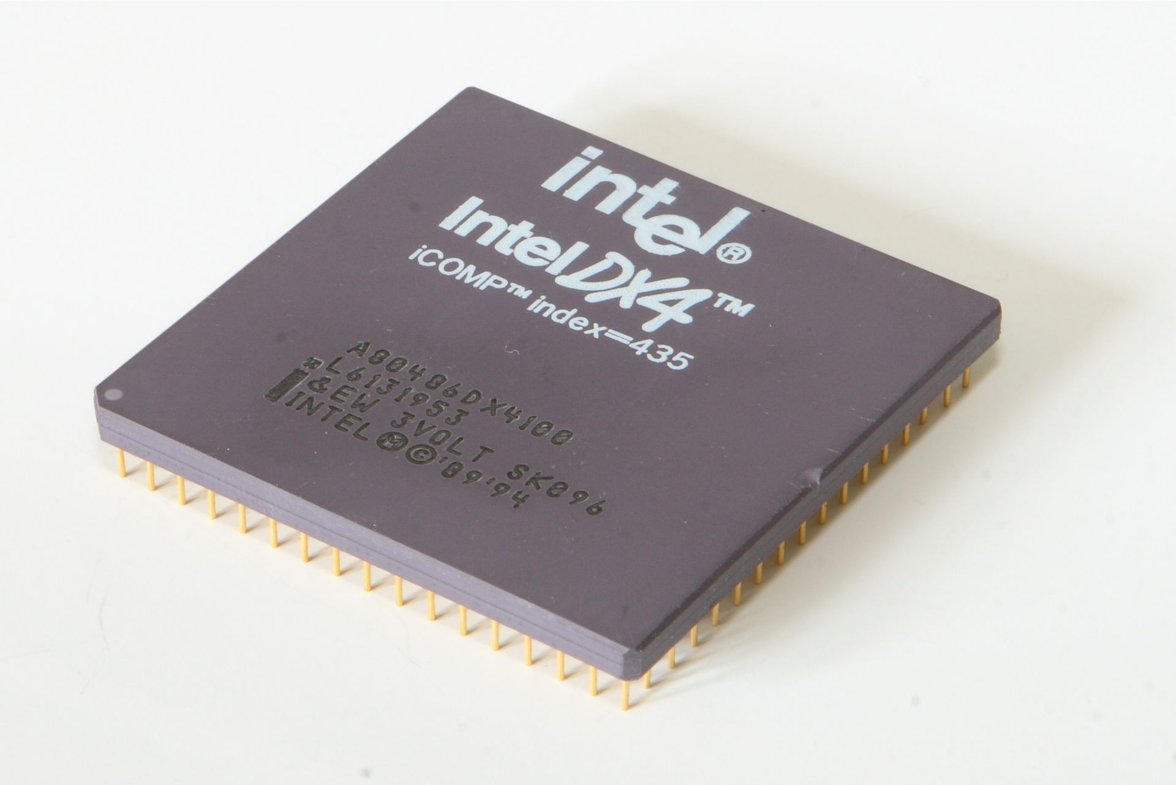
STL performance



clang / libstdc++

> *“That’s why I use C.*
> *C++ has bad performance*
> *without optimization!”*

Anonymous videogame programmer, 2019



Released in 1994 too!

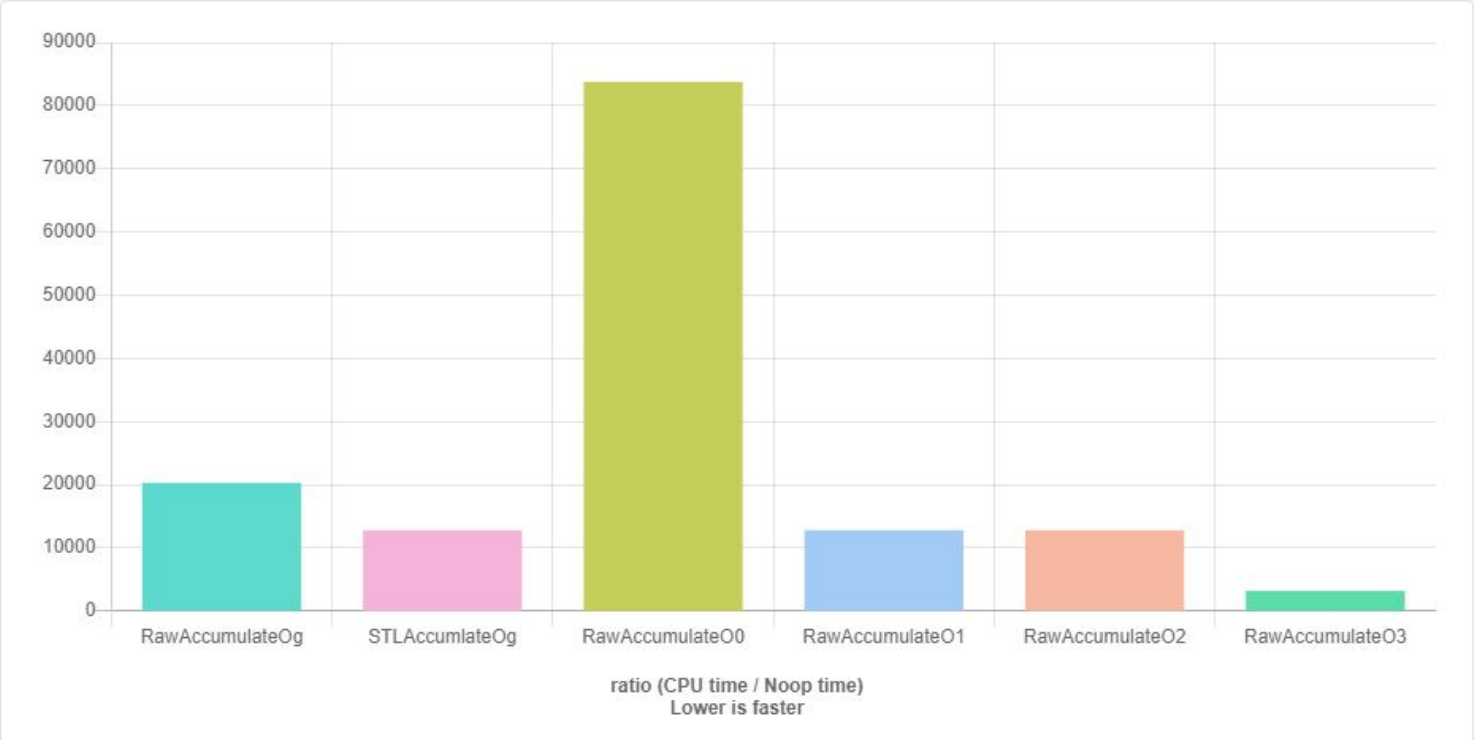
Performance today

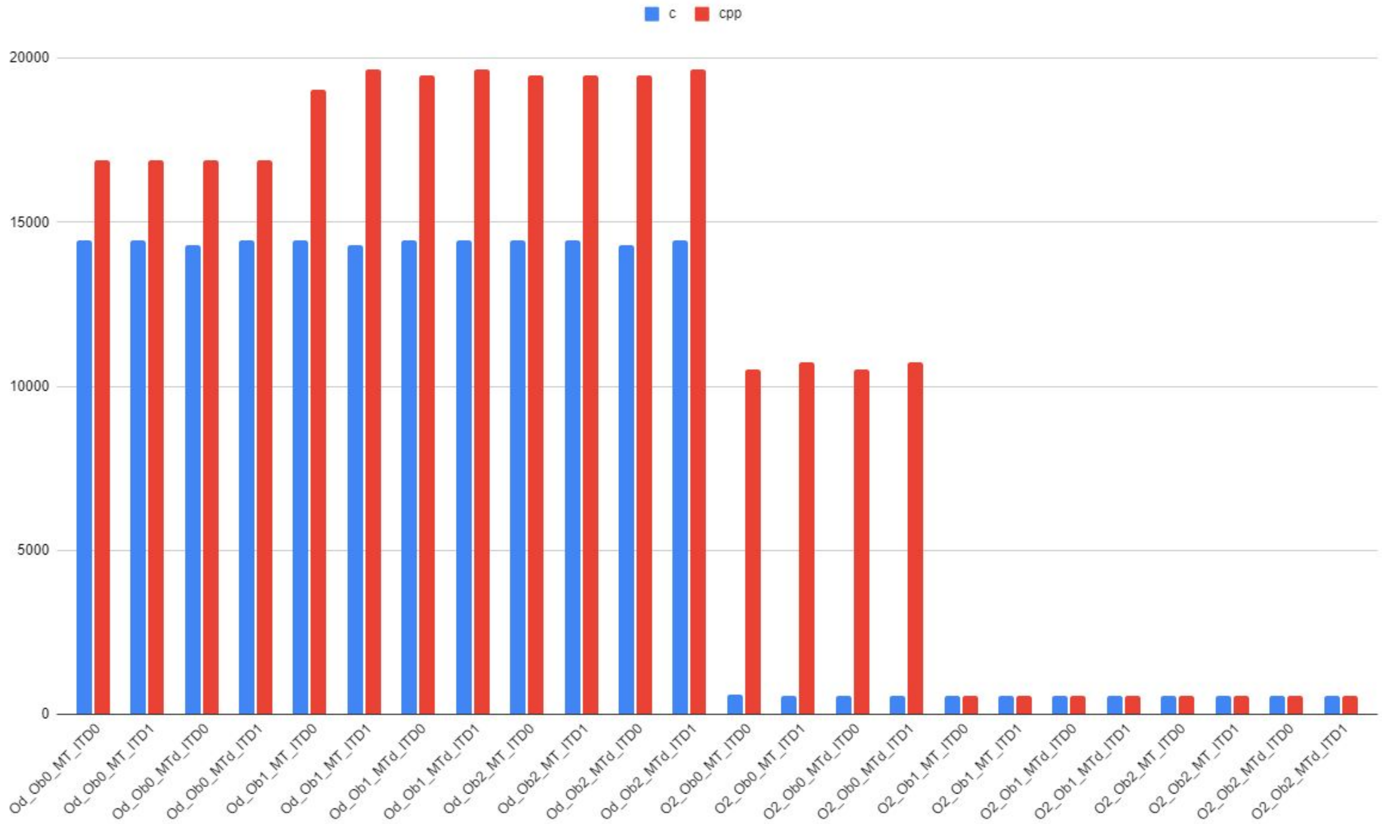
- The 80486 was the last x86 to run instructions sequentially

- Modern CPUs execute instructions out of order

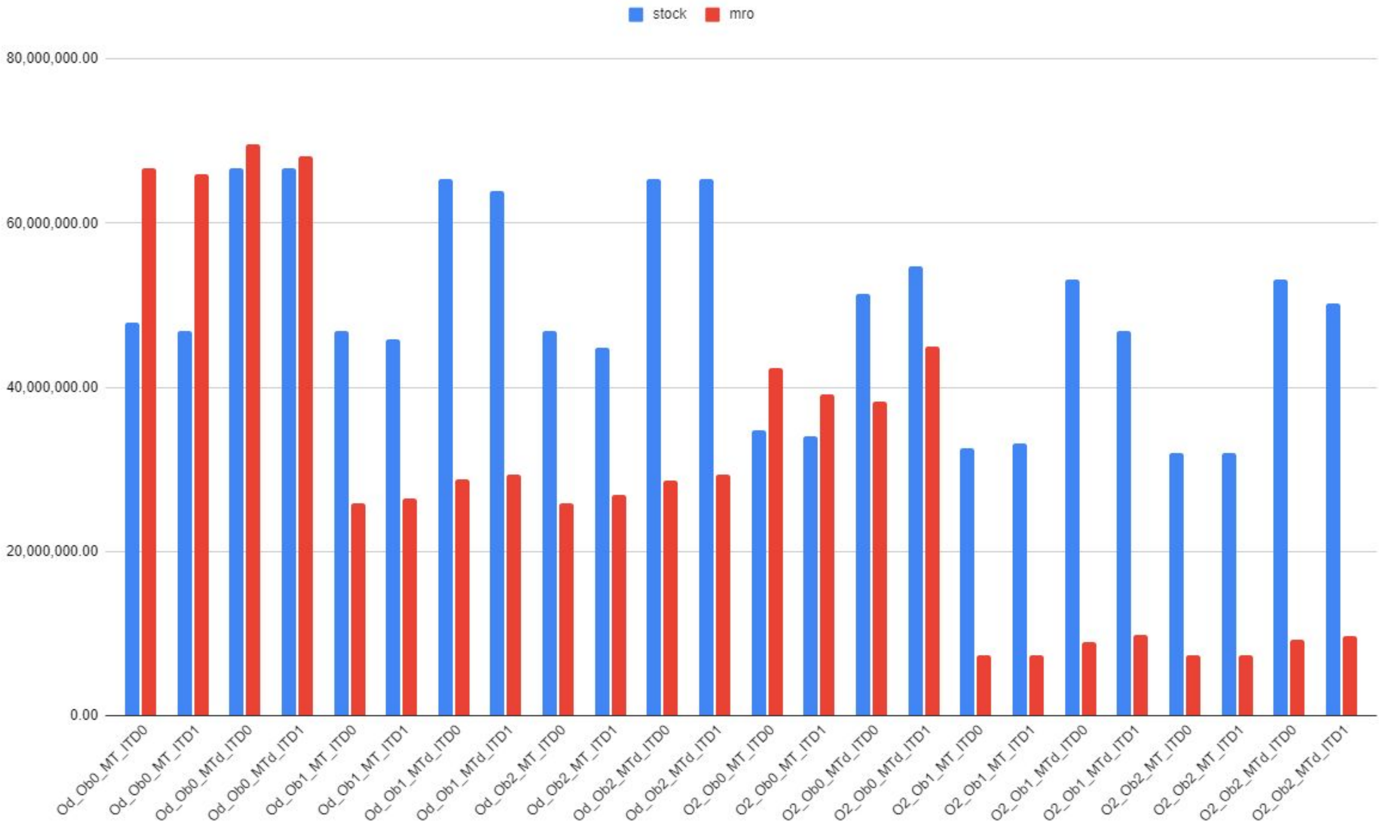
- How does “low level” imperative C fare without optimization today?

Performance in 2019





Accumulate on MSVC, C vs C++



Pathfinder benchmark on MSVC

Performance and debug

- C++ abstractions will be slower than raw C with all optimizations turned off
- Both C and C++ are an order of magnitude slower when you disable optimizations
- Enabling even minimal optimizations yields enormous gains

Performance and debug

- Some vendors offer good or decent support for optimized debug builds (GCC, MSVC)
- There's probably room for improvements
- Know your build flags!

2 STL containers

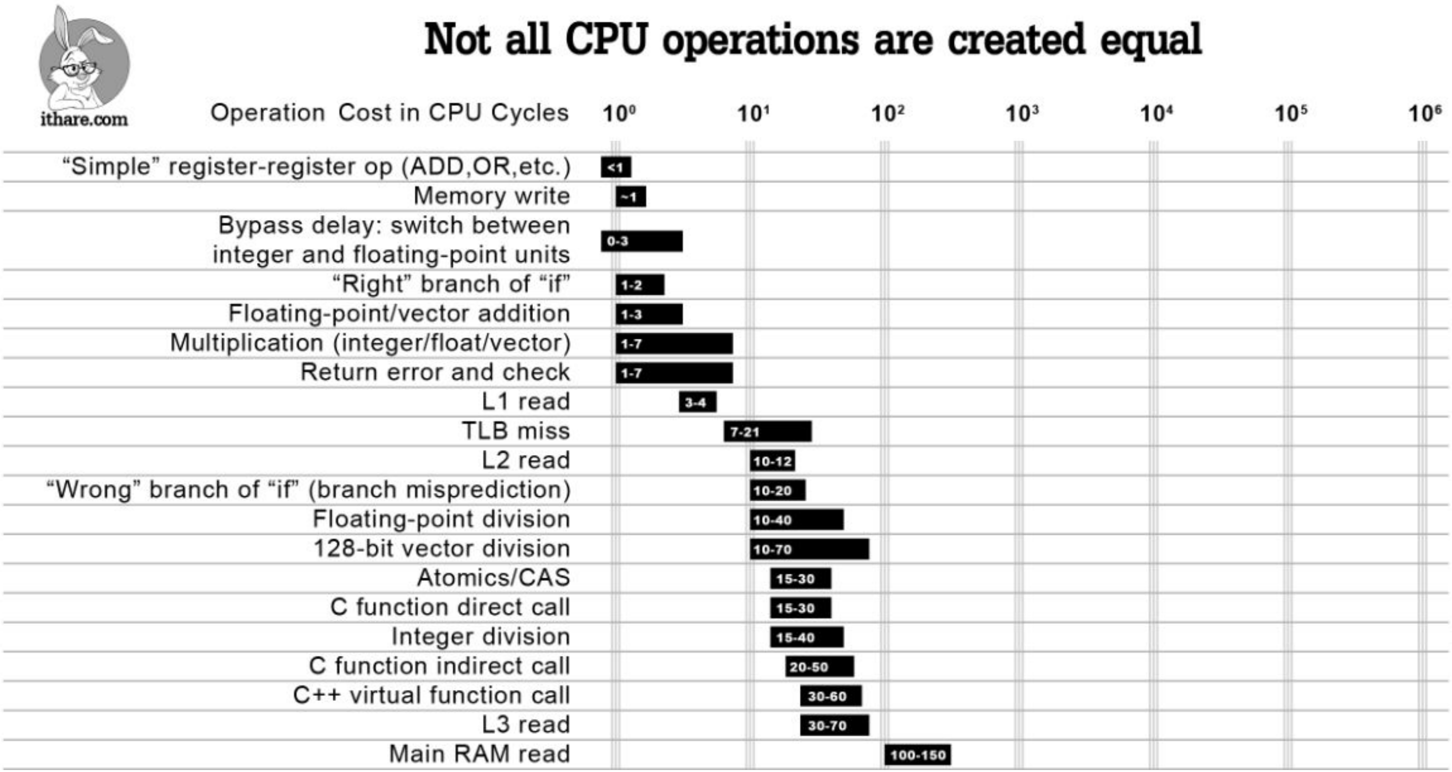
All you need is std::vector

Containers overview

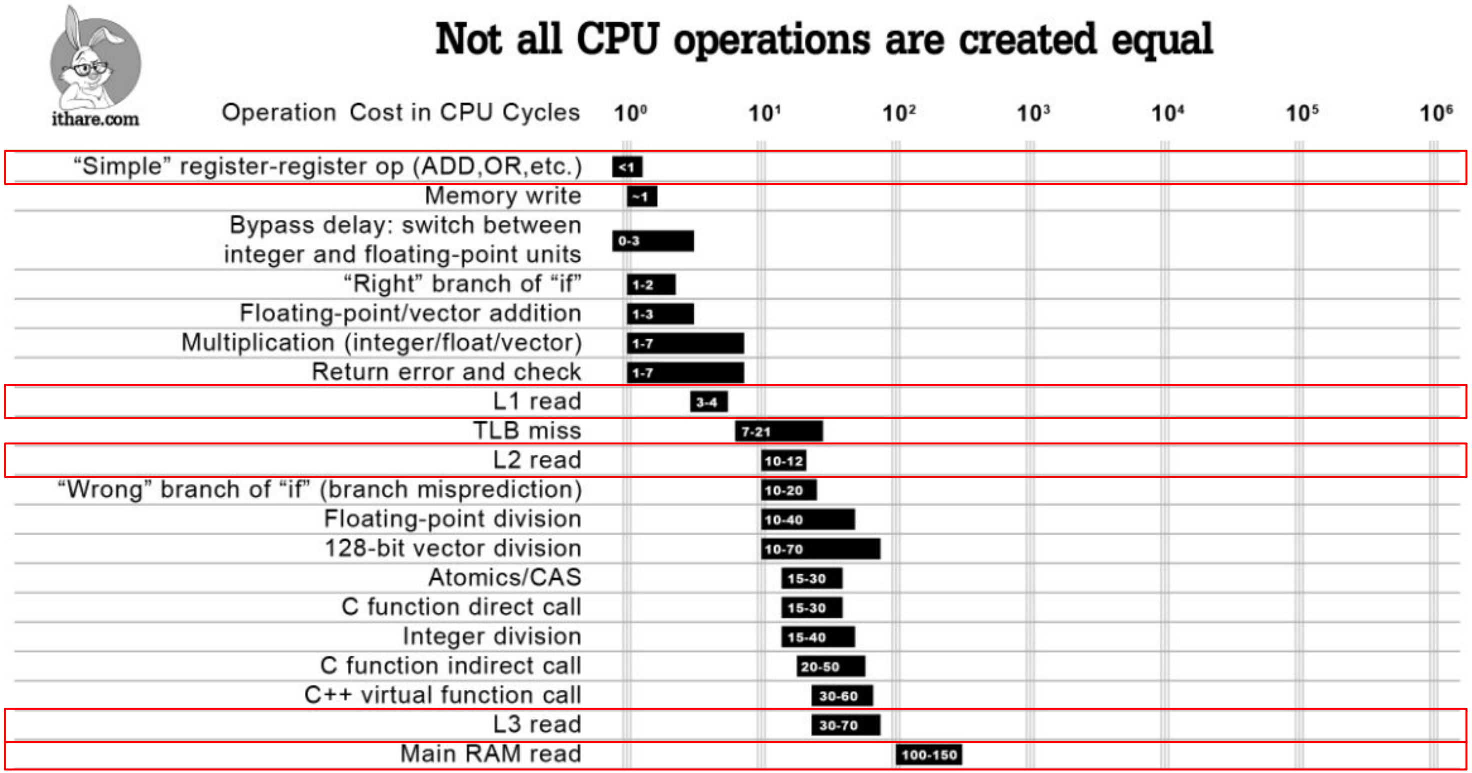
- Most commonly used containers
- Arrays and dynamic arrays
- Ordered associative containers
- Hash tables

std::vector

- Heap-allocated array that can be resized
- Go-to container in the STL
- Cheap to move and random access
- As fast as it gets to iterate over



http://ithare.com/infographics-operation-costs-in-cpu-clock-cycles/



Not all CPU operations are created equal

Operation	Cost in CPU Cycles
"Simple" register-register op (ADD,OR,etc.)	<1
Memory write	~1
Bypass delay: switch between integer and floating-point units	0-3
"Right" branch of "if"	1-2
Floating-point/vector addition	1-3
Multiplication (integer/float/vector)	1-7
Return error and check	1-7
L1 read	3-4
TLB miss	7-21
L2 read	10-12
"Wrong" branch of "if" (branch misprediction)	10-20
Floating-point division	10-40
128-bit vector division	10-70
Atomics/CAS	15-30
C function direct call	15-30
Integer division	15-40
C function indirect call	20-50
C++ virtual function call	30-60
L3 read	30-70
Main RAM read	100-150

http://ithare.com/infographics-operation-costs-in-cpu-clock-cycles/

std::vector

- Modern CPU caching can have a 1–100 impact on performance
- O(n) operations on std::vector can outperform O(log n) on other containers
- Rule of thumb: for small sets, bruteforce search through vector is faster than std::map

std::vector

- For read-intensive associative sets, consider a sorted vector
- Prefer indexes to pointers or iterators for storing long term references

std::vector limitations?

- None!
- std::vector is awesome!
- 😍👍

std::vector limitations

- Growth factor is neither specified nor configurable (most commonly 1.5 or 2)
- Standard specification prohibits small buffer optimization
- std::vector<bool> is a mess

std::array

- Stack-allocated array with fixed size
- C++11 addition
- O(1) random access and cache friendly layout
- O(n) to move, potentially as expensive as copy

std::vector alternatives

- std::vector with small buffer optimization
  - Boost’s boost::small_vector
  - Facebook’s folly::small_vector
  - Google’s absl::InlinedVector
- Avoid heap allocation for small sizes
- May be O(n) on move (and invalidate iterators)

std::array limitations

- Fixed size, not capacity
- Not suitable for dynamic insertion

std::array alternatives

- Fixed capacity vector
  - Boost's boost::static_vector
  - EA's eastl::fixed_vector
  - Facebook's folly::small_vector
- Proposed addition to the standard as P0843
  - WIP name is std::static_vector

std::map and std::set

- Classic sorted associative containers
- O(log n) access, insertion and erase
- Iterators remain valid upon insert and erase
- O(1) move construction

std::map and std::set implementation

- Almost always implemented as a R/B tree
- Data is not stored in a cache-friendly manner
- Lookup time is logarithmic, not constant

> *“STL map and set have terrible performance, don’t use them!”*

Anonymous videogame programmer, 2019

std::map and std::set implementation

- Can we do better?
- Not really…
- … unless we drop some constraints from the standard

std::map and std::set variants

- Drop the sorted requirement
- We get C++11’s std::unordered_set and std::unordered_map
- Average constant time on insert, erase and lookup

> “STL unordered map and set are not using open addressing, don’t use them”

Anonymous videogame programmer, 2019

std::map and std::set variants

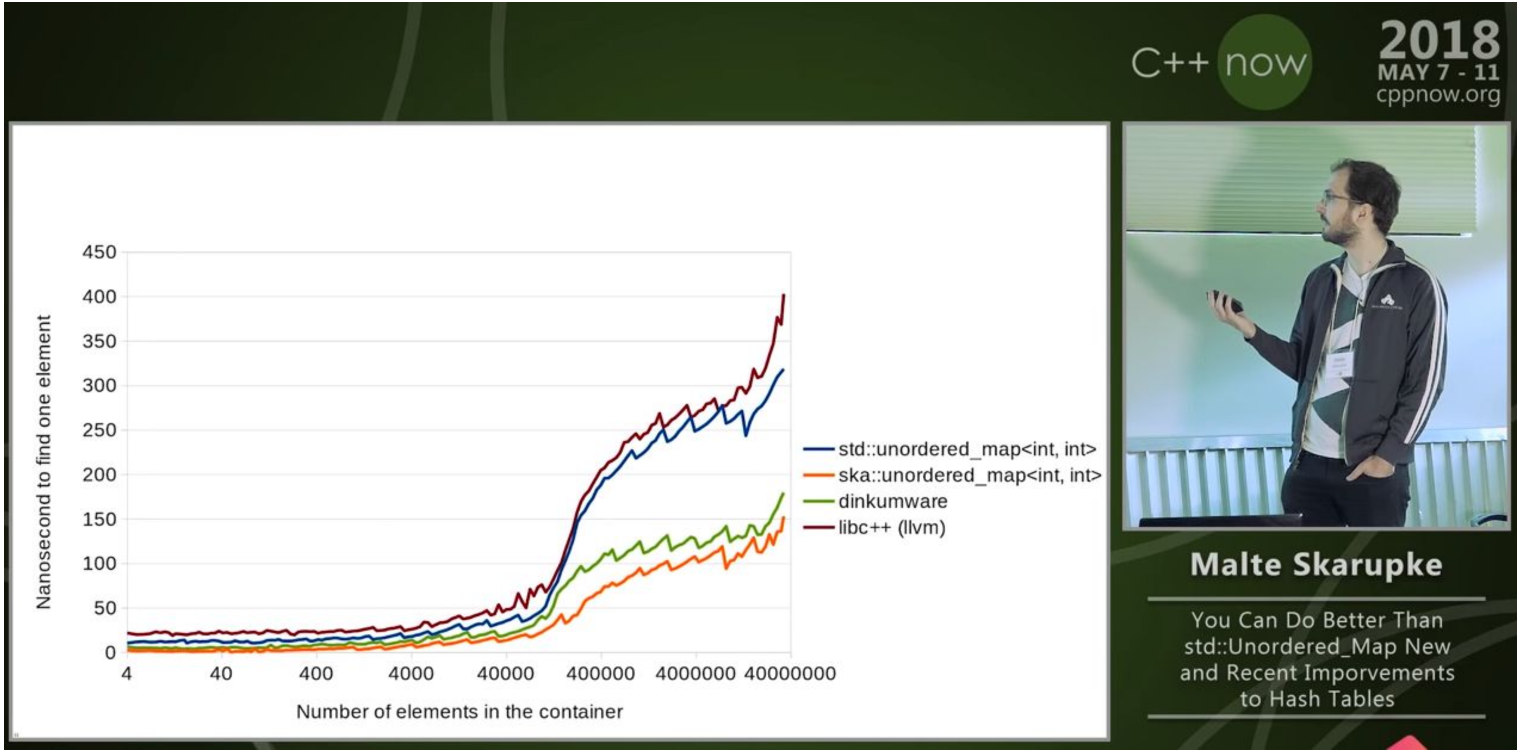
- Open addressing hash tables offer better cache performance

- Incompatible with standard requirements
  - Too high space/time tradeoff
  - Invalidate references even when no rehashing occurs

std::map and std::set variants

- Caching is not the main reason why STL hash tables are slow
- You can get good performance *and* follow the standard…
- As long as implementation doesn’t use modulo

std::map and std::set variants



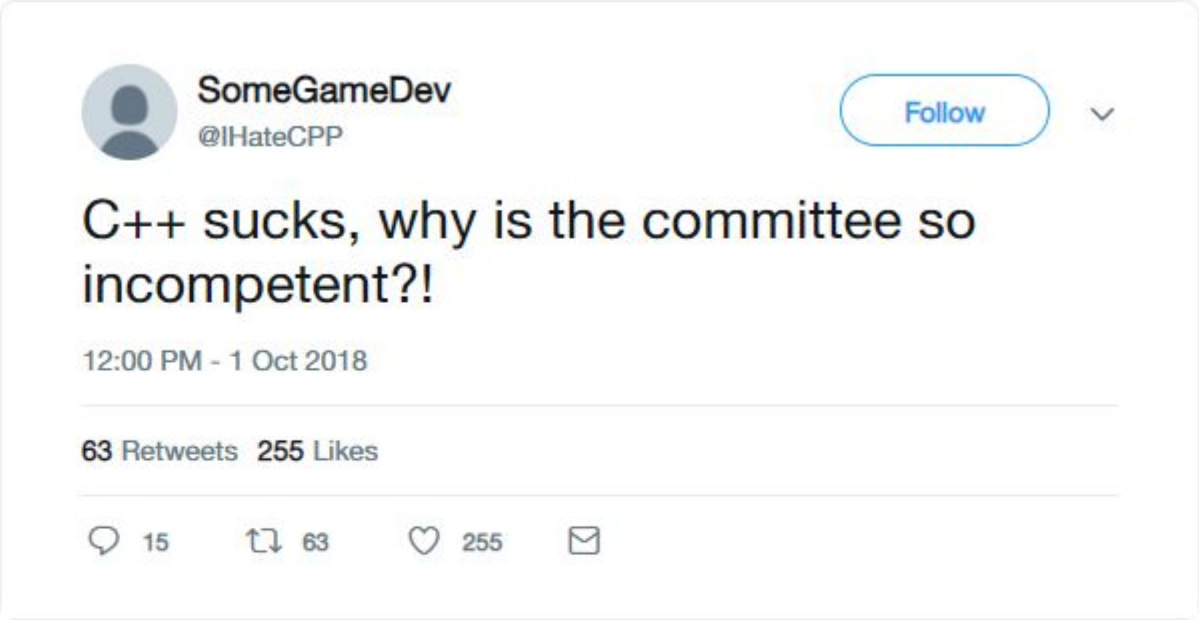
Malte Skarupke

You Can Do Better Than std::Unordered_Map New and Recent Imporvements to Hash Tables

3

The STL and you

How to make things better



SomeGameDev
@IHateCPP

Follow

C++ sucks, why is the committee so incompetent?!

12:00 PM - 1 Oct 2018

63 Retweets 255 Likes

The problem

- The Committee make specifications, not implementation
- C++ is a general purpose language, its defaults have to be sane for the 99%
- Social media rants are not a good way to get a point across

Burden of proof

- Common STL implementations are widely used and tested
- Have feature and performance tests to justify an alternative
- Revisit the comparison from time to time

“Good enough”

- Standard specifications cannot make unsafe assumptions
  - Reference stability
  - Memory overhead
- Target the most common use case
- Specific cases can benefit from specific implementations

“Good enough”

- Corollary: fit-to-purpose alternatives make poor defaults
- Remember the word of Donald Knuth

“The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times”

Sorted maintenance cost

- No code
- Code that comes with your compiler
- 3rd party library
- In-house library

Maintenance cost

- Writing generic containers is hard
- Might look easy at first
- Then one gets into corner cases such as forwarding, constexpr and trivial types
- And then the standard adds another feature…

Tactical choices

- Consider how many people one can spare on STL replacements maintenance

- Pick your battles
  - Better hash map, yes!
  - Rewrite variant or optional, hell no!

Engaging with your peers

- Ranting on Twitter will not make C++ better
- Neither will a talk given only at GDC
- Make your voice heard where the rest of the C++ community is
- Meetups, conferences, ISO study groups

Engaging with your peers

- Progress goes much faster when people collaborate
- The bigger the sample, the better the results
- Don’t be afraid of talking to C++ developers outside of your field

Engaging with your peers

- Challenge your vendor quality of implementation if needed
- Publish your findings
- Provide reusable benchmarks
- Need help packaging? Ask me!

In conclusion

- STL aims to be a good enough default, as long as some optimizations are enabled
- Specific cases may benefit from STL alternatives
- Feedback is needed to improve the experience of all C++ developers

Furthermore, I think your build should be destroyed

Thanks!

Any questions ?

You can reach me at

- ✉ mro@puchiko.net
- @MatRopert
- @mropert
- https://mropert.github.io

References

- *C Is Not a Low-level Language* – David Chisnall, ACM Vol. 16 No. 2 – March-April 2018

- *You Can Do Better than std::unordered_map* – Malte Skarupke, C++Now 2018

- *Fifty shades of debug* – Mathieu Ropert, August 3rd, 2019

References

- *Accumulate Benchmark:* *http://quick-bench.com/Z-PZk-rBkKjhf5OmIcoiwB2Ijdg*
- *MSVC optimization flags benchmark:* *https://github.com/mropert/debug_bench*