Hello London!

C++ (& Delphi) in 12.1 Athens

David Millington, C++ Product Manager
david.millington@embarcadero.com

embarcadero®

Safe Harbour Statement

Any plans discussed represent our intentions as of this date, and our development plans and priorities are subject to change, due to competitive factors, availability of resources and other matters common to all independent software vendors.

Accordingly, we can't offer any commitments or other forms of assurance that we will ultimately release any or all of the described products on the schedule or in the order described, or at all.

These general indications of development schedules or "product roadmaps" should not be interpreted or construed as any form of a commitment, and our customers' rights to upgrades, updates, enhancements and other maintenance releases will be set forth only in the applicable software license agreement.

IMPORTANT: Features are not committed until completed and Generally Available (GA) released

What’s new in C++

and know lots of Delphi people watching

So: cover what’s new, but also

what’s cool in C++ that *everyone* can use?

Goal: lots more live IDE use, than slides :)

Clang Upgrade in C++Builder 12.1

A new foundation for C++Builder and RAD Studio

- New version of Clang
- Revision of the entire toolchain
- Focus on *platform conventions*
- Focus on *doing it right*
- Focus on *quality*
  - STL, linker etc: a must-use

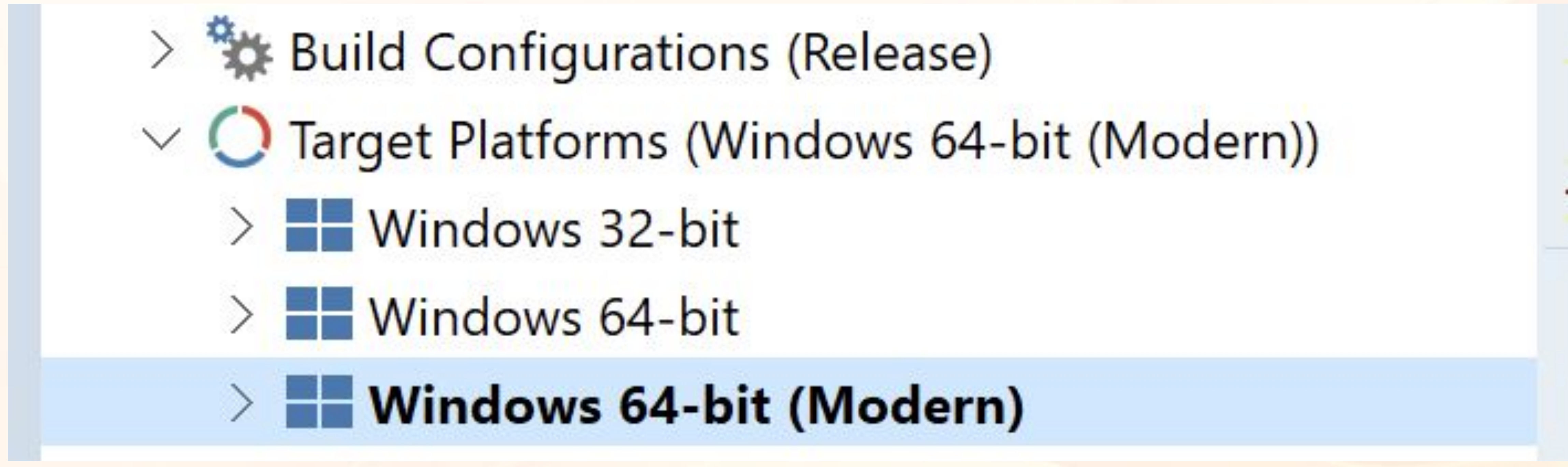
The value of C++Builder with a high quality toolchain for modern C++

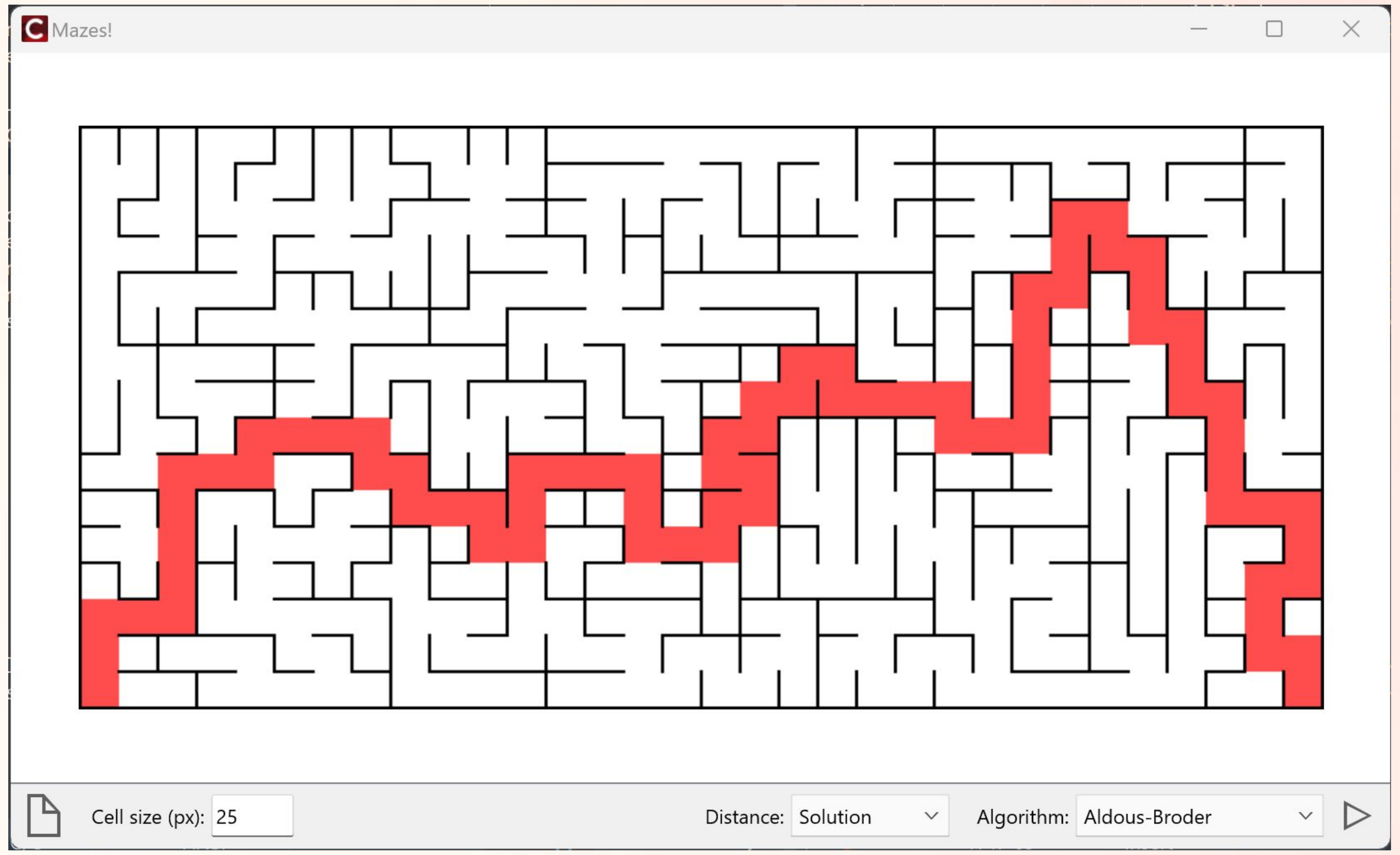
Technical Info of the new Clang Toolchain

- General goal: follow Windows platform standards
  - Leads to COFF, UCRT, PDB, etc
- Uses the Itanium C++ ABI
- COFF object file format, PDB debug format
  - Not link-compatible with a VC++ COFF object file, different C++ ABI (goal: *source compatible*, handle any C++ code)
- C RTL: Uses the Windows component: Universal C Runtime (UCRT)
- C++ RTL, with enormous amount of exception handling work - 700+ new tests + more
- STL: LLVM's libc++
- Linker: LLVM's lld (used for Chrome)
- 64-bit binaries (even in IDE) – enormous memory space

Clang Toolchain: VCL and FMX apps

- VCL applications
- FMX applications
- DLLs, LIBs, console apps, etc
  - Can be 'pure C++' for these if you wish
- In 12.0, Delphi packages (components) are statically linked – but much faster linker to make up for it
  - Ie the Delphi default! Never had a fast enough linker to do it before
  - These features, like dynamic package linking and CMake support, **coming soon**


Build Configurations (Release)
Target Platforms (Windows 64-bit (Modern))
Windows 32-bit
Windows 64-bit
Windows 64-bit (Modern)



Cell size (px): 25
Distance: Solution
Algorithm: Aldous-Broder

FMX 'Mazes' app

- See GA disclaimer

Clang Upgrade: What’s Coming

- Dynamic packages (using BPLs) – coming soon
  - You can still use DLLs etc fine, this is only packages
  - Making packages with C++ source, similarly coming soon

- CMake support – also coming soon

- Parallel compilation with --jobs: coming, but use TwineCompile now (and in future too, it’s good)

- See GA disclaimer

Clang Upgrade: Why It Matters to You!

Quality

- Linker problems? (Eg memory issues) -> *entirely new, large memory linker*
- STL problems? (Using 3rd party C++ code or modern C++ code?) -> *Much better STL!*
- Compiler stability (ICEs) etc -> *More stable!*

Compatibility

- Want to use other tools? (Eg other debuggers) -> *Generates PDB!*
- Use C++ code from anywhere -> *Much better compatibility!*

Modernisation

- Clang 15, much more recent.

Performance

- Linker about 4x faster

Our foundation for C++'s future – here for you now.

Feb 2024: Behind the Build for 12.1:
youtube.com/watch?v=Ps5pW5uhmMw

Clang Upgrade: Key Differences

We've minimized differences as much as possible

- C RTL: it no longer uses the old C RTL, it uses the standard Windows C RTL
  - Functions coming from Turbo C don't exist any more
  - All of them have standard C replacements
  - We were worried before release but have seen very, very little code using them in the wild. And they're easy to fix. Eg, **`random()`** is replaced by **`rand() % x`**.
  - Occasionally MS-isms, eg swapped parameter order of some obscure functions. Caught by the compiler. Easy to fix.
- Memory manager: uses UCRT MM, so can't replace with FastMM
  - We like FastMM, but this is a result of using the standard Windows C runtime
- STL
  - This is a good difference! *So much works*
  - Parallel algorithms are not available in this version

Clang Upgrade: Key Differences

Some differences that are *really good*

- New linker. Enough said.
- Compiler, linker etc are all 64-bit EXEs
  - Huge memory space
  - This includes when compiling in the IDE!
- New STL. It works very well.
- C++ RTL: massive work on compatibility and exception handling

- In Aug 2023 (last year!) we had more C++ and EH tests passing than our then-shipping toolchain, and it's only got better since then.

Demo: Clang Upgrade

Examples?

A great deal of bcc64x is that *it works.*

That’s hard to demo.

What about the other benefits?
What can Delphi folk get out of it?
What can C++ folk get out of it (beyond, already 100x better)?

Let’s do some code optimisation!
I’ll show you something cool, but technically *unsupported* :)

Example Code Optimisation with Clang

The Clang compiler is very good at optimisation.

Data is generally FP or integer! (At its core. Strings are a set of integers.)

This code processes a large array of floating point data. Mimics:

- **Image processing. Scientific data processing. Engineering calculations? Payroll processing (if you have a lot.)**
- **Not intended to be real-world**
- **Is intended to demonstrate possibilities *just by recompiling***
  - No major code modification
- **Measuring just by how many gigabytes of data we can process per second**
  - Not a ‘real’ measurement, but indicative

Example Code Optimisation with Clang

Take some Delphi or C++ code (I prepared both)

Rewrite it in C++, or recompile it in C++, using the new Clang. What happens?

Then do some more advanced (but not very advanced) things.
I am ***not an expert at what I am showing you*** **and there will be much more you can do.**

Delphi -> C++, multiple ways:

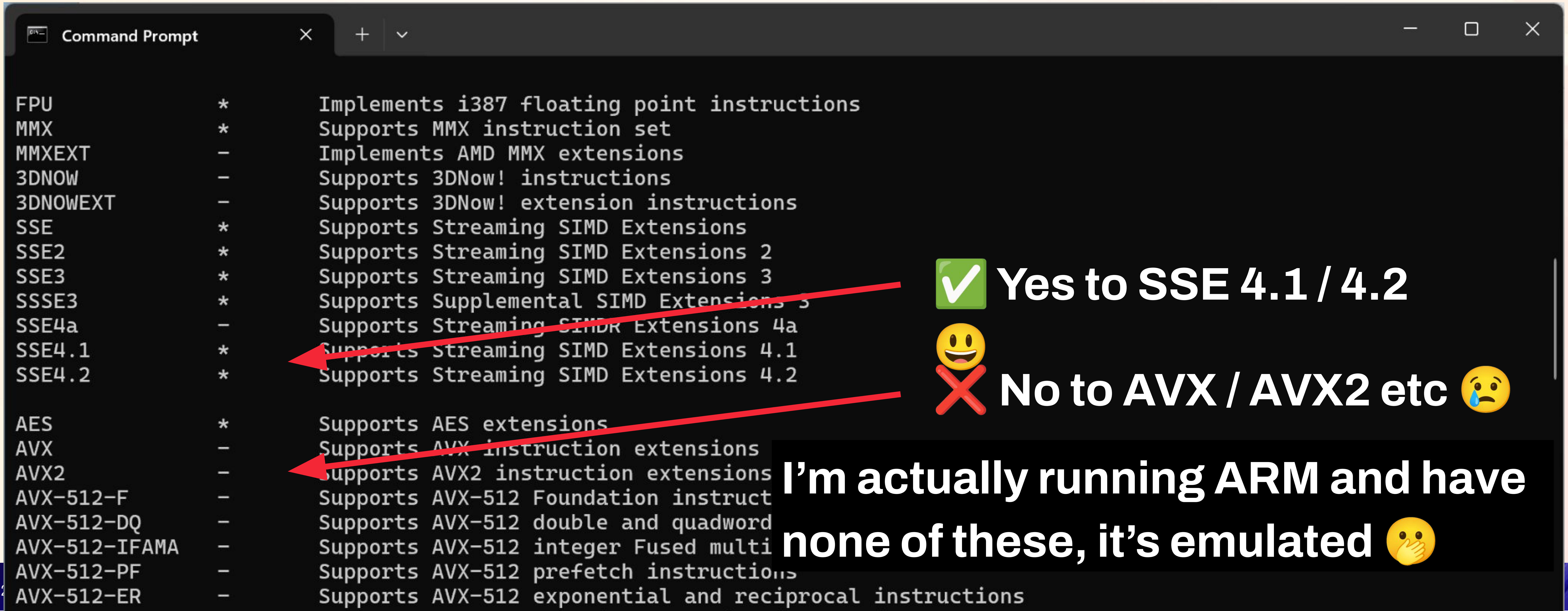
- **Delphi abstract class / interface, implemented in C++**
  - https://github.com/Embarcadero/CodeRage2016/tree/master/David Millington - Mixing Delphi and C++ plus Youtube plus blog
- **Here, simple, just calling a DLL**

Demo: Optimising Math with bcc64x

Checking your CPU's instruction sets

CoreInfo from SysInternals / Microsoft

https://learn.microsoft.com/en-gb/sysinternals/downloads/coreinfo



```
FPU             *       Implements i387 floating point instructions
MMX             *       Supports MMX instruction set
MMXEXT          -       Implements AMD MMX extensions
3DNOW           -       Supports 3DNow! instructions
3DNOWEXT        -       Supports 3DNow! extension instructions
SSE             *       Supports Streaming SIMD Extensions
SSE2            *       Supports Streaming SIMD Extensions 2
SSE3            *       Supports Streaming SIMD Extensions 3
SSSE3           *       Supports Supplemental SIMD Extensions 3
SSE4a           -       Supports Streaming SIMDR Extensions 4a
SSE4.1          *       Supports Streaming SIMD Extensions 4.1
SSE4.2          *       Supports Streaming SIMD Extensions 4.2

AES             *       Supports AES extensions
AVX             -       Supports AVX instruction extensions
AVX2            -       Supports AVX2 instruction extensions
AVX-512-F       -       Supports AVX-512 Foundation instruct
AVX-512-DQ      -       Supports AVX-512 double and quadword
AVX-512-IFAMA   -       Supports AVX-512 integer Fused multi
AVX-512-PF      -       Supports AVX-512 prefetch instructions
AVX-512-ER      -       Supports AVX-512 exponential and reciprocal instructions
```

✅ Yes to SSE 4.1 / 4.2 😃

❌ No to AVX / AVX2 etc 😢

I'm actually running ARM and have none of these, it's emulated 🤭

Demo: Optimising Math with bcc64x

Now we know why AVX2 doesn't work – it's not available on this CPU.

- Original code
- Told it the pointers weren't aliased with __restrict, it could do more optimisation
- Got some nice performance
- We built versions for SSE4 (slow, integer instructions, not useful here)
- We built with AVX2 and it failed completely
  - We realised we can't optimise too far, because some machines don't have some instruction sets
    - ...or can we?

Optimisation via instruction sets

- Everything was looking so good!
- But suppose I want to optimise my code with AVX2
  - *which doesn't work on every machine*
- What do I do? Build three different EXEs/DLLs?
  - SSE2 (safe target)
  - AVX (advanced)
  - AVX2 and AVX-512 (modern computers)

...and ship three different files? `myapp-sse2/avx/avx512.dll`? Load at runtime?

We had something so promising and now it looks like a lot of work.

...or does it?

Optimisation via instruction sets

The ideal solution is:

- One EXE/DLL
- with *multiple different versions of the function* compiled in
- at runtime, *without you doing anything at all,* if you have an AVX2 CPU then the AVX2 version of the function is what's run

So you write your code once.
When compiled, it's optimised for everything.
Runtime dispatch based on your CPU.
It just runs, no matter what hardware your users have.

Back to the demo!

The real C++ way

This code converted Delphi to C++

- Memory allocation is still Delphi
- Initialising the values is still Delphi
- We're accessing an array *by pointer* like it's 1999
- What if we did this all C++ and in a modern style?

Back to the demo!

The real C++ way – what did we see?

- Used a vector
- Reserved memory (to avoid multiple reallocations as add items)
    - Delphi did this too, via SetLength, which actually created all the array elements – reserve reserves the memory but the count of elements remains 0
- Loop to add them, this is ‘embarrassingly parallelizable’ too just like the math
- In the math method, use iterators
    - Iterating over two collections at once has no really elegant solution
    - I iterated over one, and incremented the iterator for the other
    - Note the code assumed the same length, there was no safety checking

The real C++ way – *results*

Time for the entire app – allocate memory, initialise, do math, exit

Original Delphi code:

4 ½ seconds (Win64)

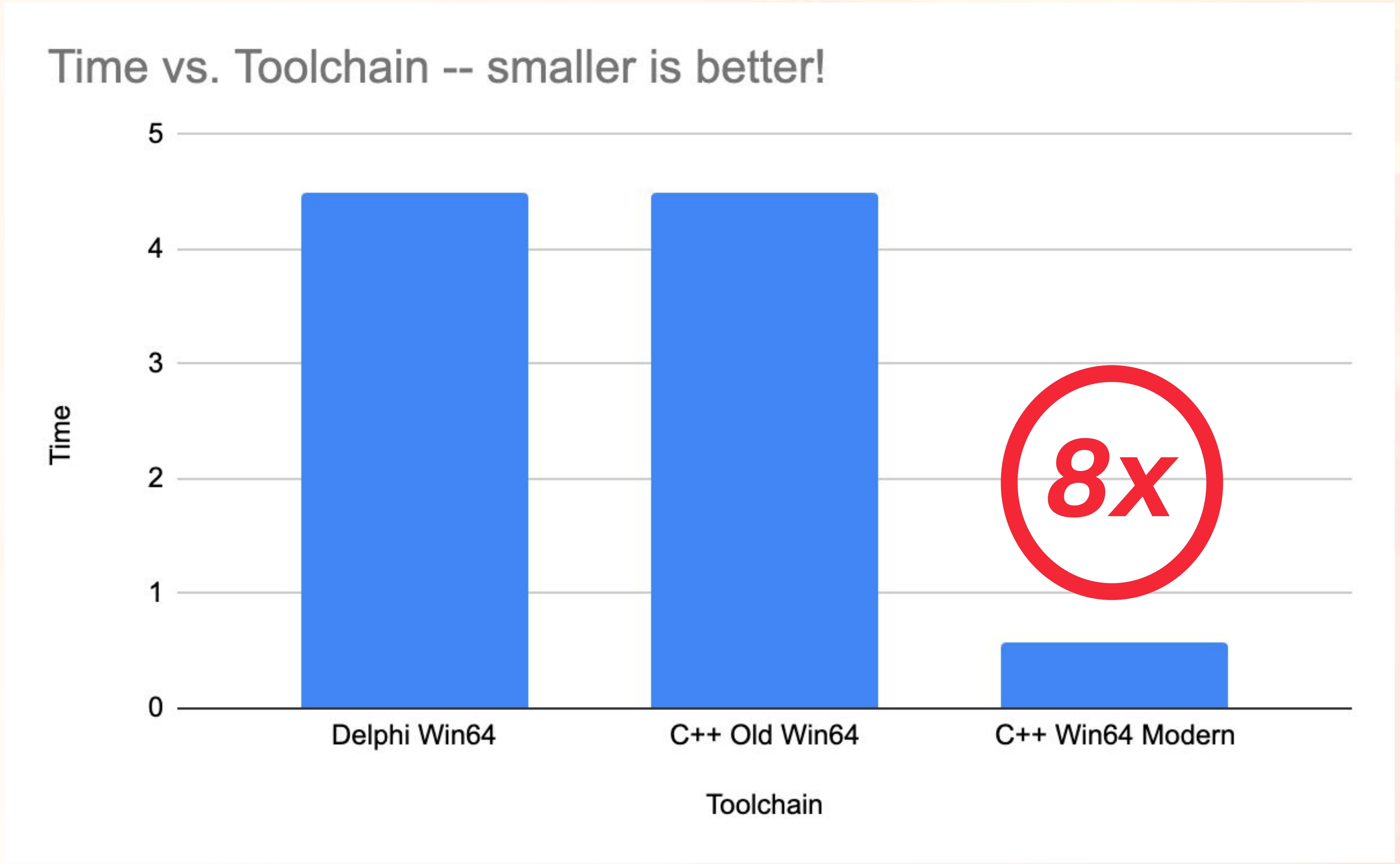
Original C++ code:

4 ½ seconds (old Win64)

New C++ code:

0.57 seconds (new Win64!)

Eight times faster!



What did we see?

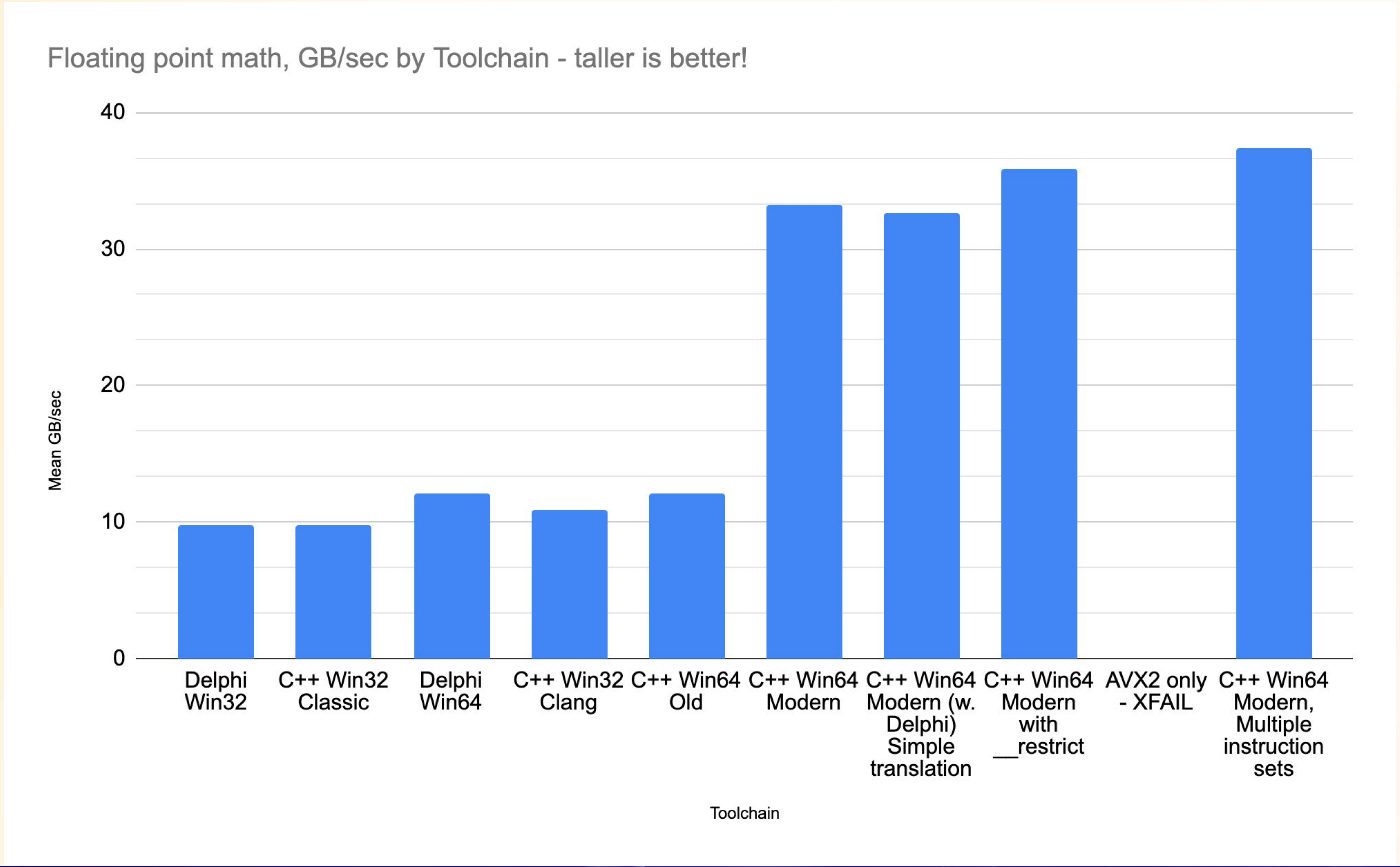
Clang being awesome

What did we see?

- **Started with some basic math code**
- **Big speed bump using new Win64 Modern Clang**
- **Further speed bumps with __restrict for nonaliased pointers, etc**
  - If allocating in C++, could allocate on aligned memory
  - *I expected more performance here, need to look at assembly*
  - See: 'Restrict Qualified Pointers in LLVM' by Hal Finkel
- **Optimising for specific instruction sets, eg SSE2**
  - How this is great but fails when hardware doesn't support those instruction sets
- **Optimising for multiple instruction sets *at the same time***
  - Function version chosen at runtime
  - *Not supported :)* We target SSE2/3 by default
- **Making use of modern C++ – entire app is faster**

What did we see?

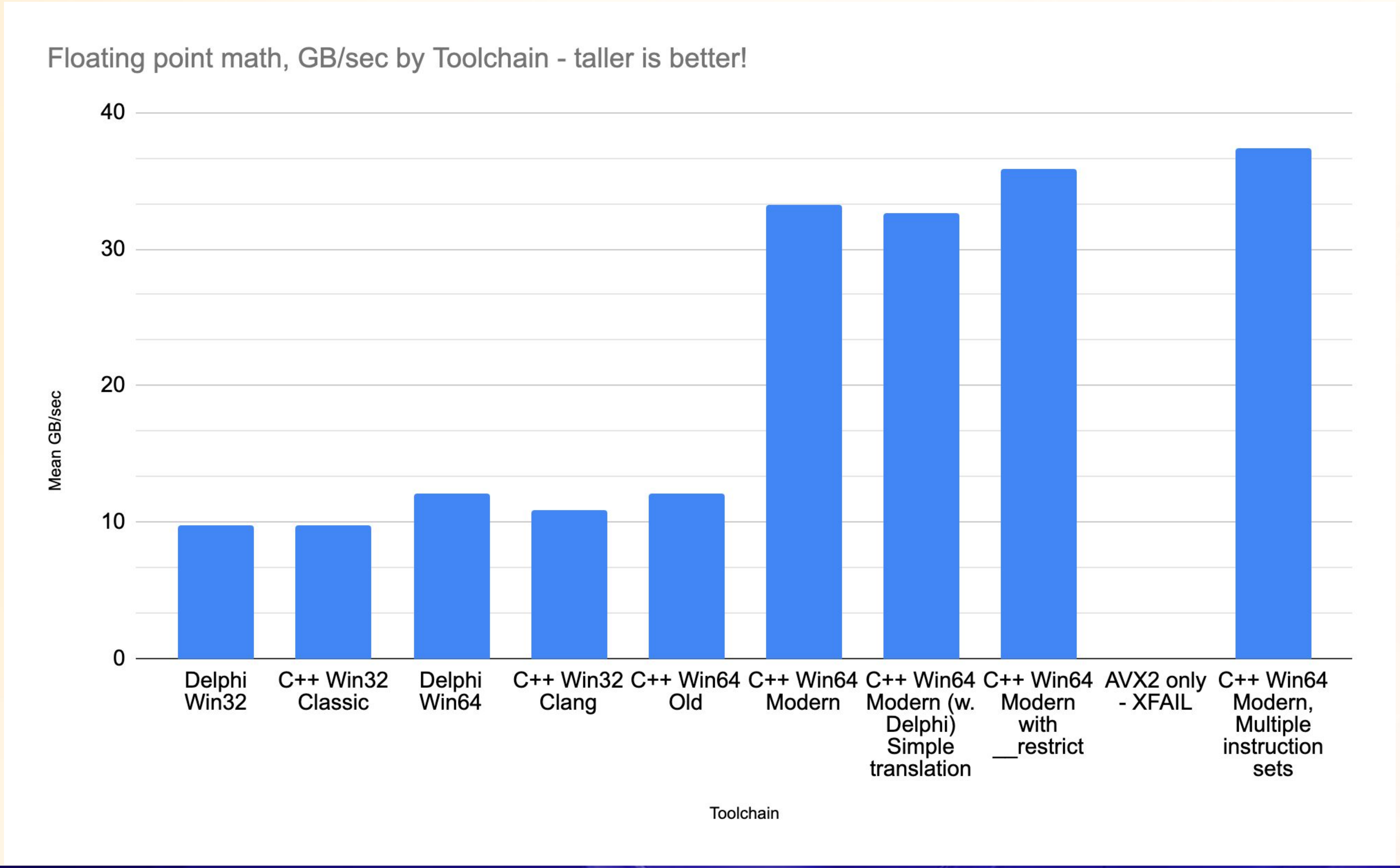
* Measured by running 3 times -> mean. Running on emulated ARM hardware



- Traditional Win64 a little better than Win32
- ***Jump to the Win64 Modern Clang immense!***
- Sanity check: bcc64x similar in C++ app, and Delphi app
- __restrict gives small boost
- (Not shown: C++ with 16-byte alignment, similar)
- Should be able to get (much?) more – *I am not an optimisation expert,* this seems a good demo of possibilities
- **Can build for all instructions sets at once and get best possible per hardware 😍**

What did we see?

* Measured by running 3 times -> mean. Running on emulated ARM hardware



Who benefits?

- Anyone where performance is important
- This was floating point. Try *integer* operations – like string manipulation! Lots of SSE4 and AVX instructions there
    - Funny story while writing the presentation…
- Optimisation is not as simple as turning on a switch, have to write code so the compiler can optimise it
- We did not analyse the assembly at all

Optimisation thoughts

(Did I mention, not an expert on this area?)

Look for *vectorisation* possibilities. 'Embarrassingly parallel'.

- Keep in mind data alignment, avoid data overlap, tell compiler what's safe

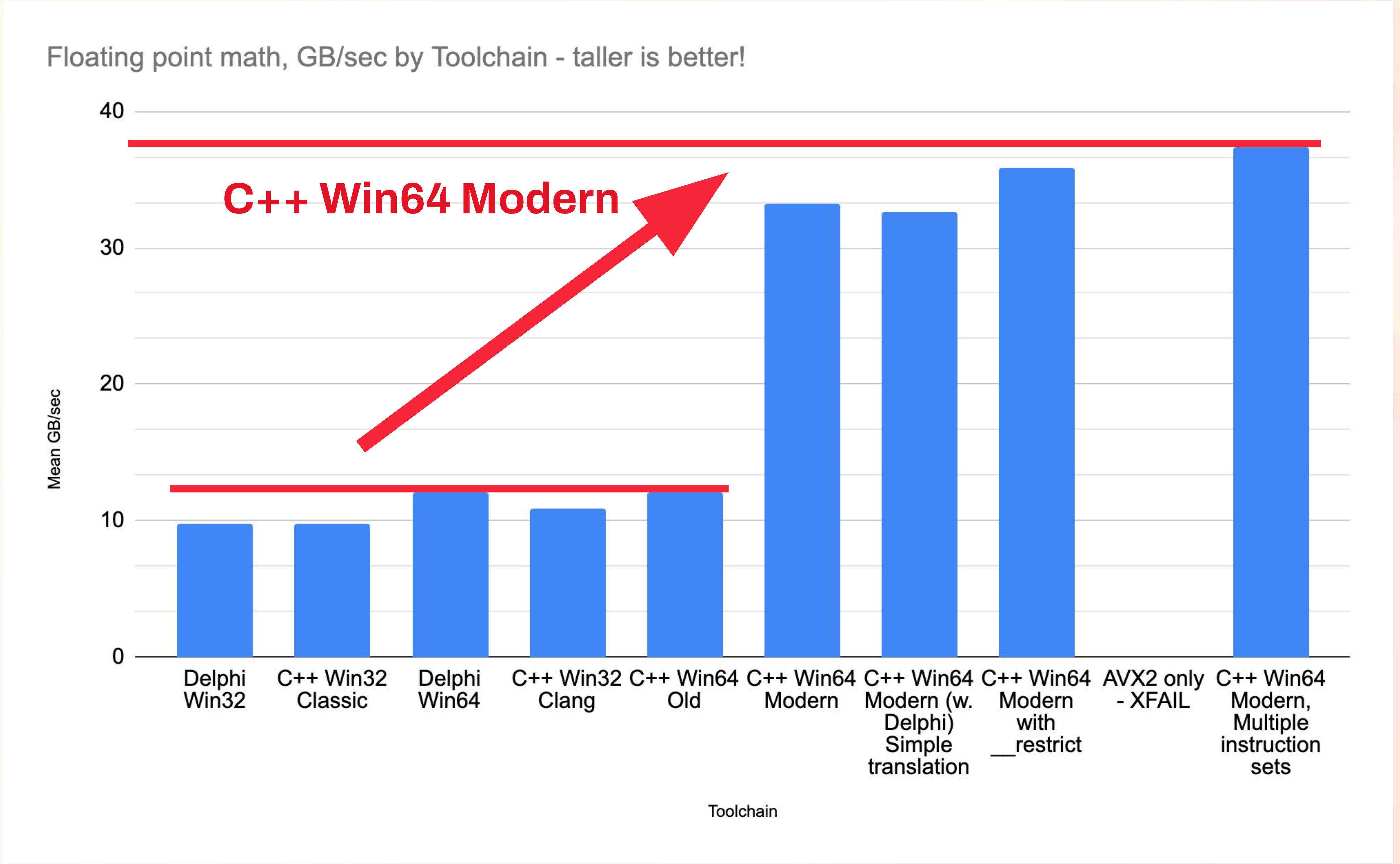
Eg: records of name, DOB, address, age. Each is a *row*

- You want to calculate, say, average age
- Change to a *column*, so have a set of numbers: process all of them in one go
- The best speedup I ever got in my career was doing this: 260x.
  - Thousands of (slow) virtual method calls iterating over floating point -> a single virtual method call that iterated over a large table of FP

- Look at other libraries like *libsimdpp* for low-level primitives

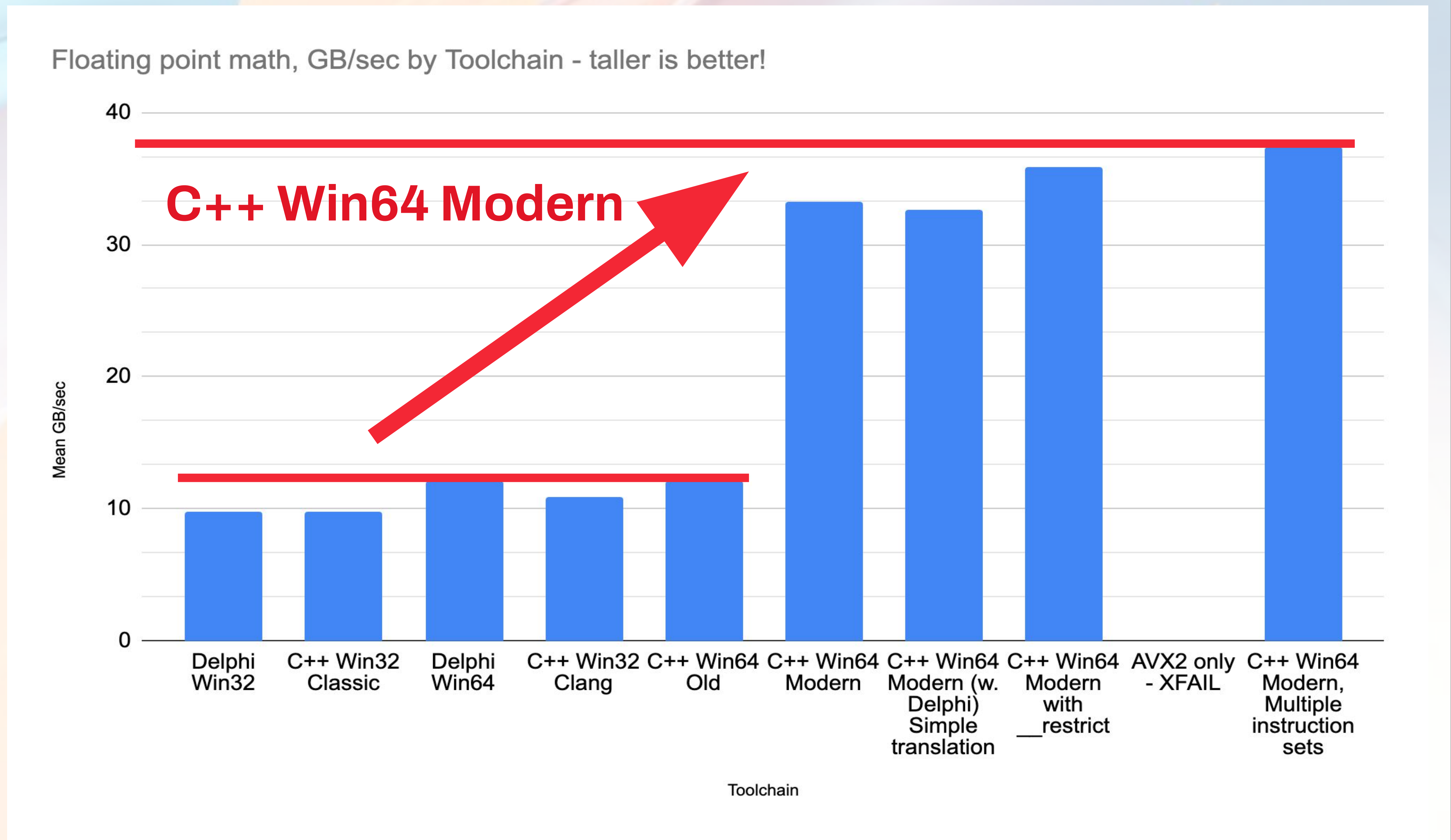
Takeaways

- **New C++ Win64 Modern**
  - Built to 'do it right'
  - Immensely better quality
  - Matches platform standards (eg PDB, COFF)
  - Solves C++ problems re STL, linker, etc
  - Use it today, more coming (eg dynamic packages) soon!
- **Compiled code performance**
  - Immense jump, old to new
  - 4x faster than Delphi & C++ Win32
  - 3x faster than Delphi & C++ Win64 (old)
  - 8x faster when it's all C++
  - **Unsupported**, but try multi-targets :)



New C++ Win64 Modern *does it right*

New linker, STL, huge quality, etc



- ‘Restrict Qualified Pointers in LLVM’ by Hal Finkel: https://llvm.org/devmtg/2017-02-04/Restrict-Qualified-Pointers-in-LLVM.pdf
- CPU dispatch / targets: clickhouse.com/blog/cpu-dispatch-in-clickhouse
- https://github.com/Embarcadero/CodeRage2016/tree/master/David Millington - Mixing Delphi and C++ (remember packages coming)
- https://stackoverflow.com/questions/64580921/c-aligned-new

Live Q & A

david.millington@embarcadero.com