

WebAssembly

Here Be Dragons



JF Bastien
Google

@jfbastien



Dan Gohman
Mozilla

@sunfishcode

Presentation given at the [2015 LLVM Developer's Meeting](#) on October 29th in San Jose, California.

¶ JF narrates:

WebAssembly is a tale of four browser vendors, seeking new languages and capabilities while staying fast, secure and portable. The old JavaScript wizard still has many spells under its belt, but it seeks a companion on its quest to reach VM utopia. WebAssembly is that companion.

¶ Dan intones:

In this quest, mad alchemist Dan and jester JF will detail their exploration of LLVM-land. You'll get to witness firsthand their exploration of ISEL and MI, hear of their wondrous encounter with MC, and gasp at the Spell of Restructuring wherein SSA+CFG is transmuted into regs+AST. Will our adventurers conquer the Target and capture the virtual ISA?

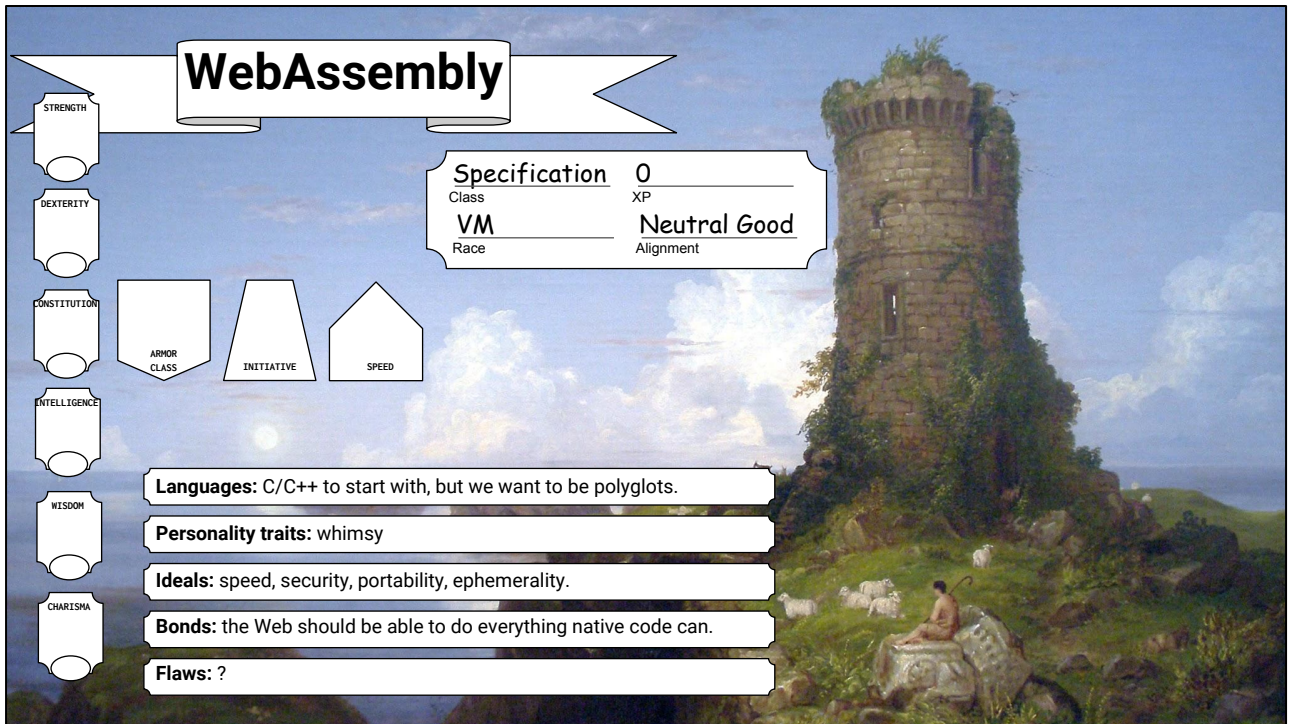
Join us in this exciting tale to which *you* are the hero!

Presenters:

- [Dan Gohman](#), Mozilla.
- [JF Bastien](#), Google.

Note to the reader: we use the following markers throughout the speaker notes:

- ▶▶ denotes when we click to animate inside slides.
- ¶ denotes when we change who's talking.



What is WebAssembly?

¶ **Dan: Let's meet our hero!**

As one does for anything new, let's build a [character sheet!](#) ►►

- **Character name:** WebAssembly
- **Level:** 0 (working on building a [minimum viable product](#)).
- **Class:** Specification.
- **Race:** ISA.
- **Alignment:** Neutral Good.
- **Experience points:** start at 0, we'll get some experience as we get closer to launch.
- **Background:**
 - Born from the union of [Emscripten/asm.js](#) and [PNaCl](#);
 - Cousin of JavaScript;
 - Godparents [Chromium](#), [Mozilla](#), [Edge](#), [WebKit](#);
 - Designed on [Github](#).
 - Born under [W3C Community Group](#).

►► Basic attributes:

- **Strength:** leveraging the power of the Web.
- **Dexterity:** portability.
- **Constitution:** designed for security-in-depth from the start.
- **Intelligence:** very low! WebAssembly is a low-level language. It's up to the

- compiler on the developer's machine to optimize, a WebAssembly VM only does basic things such as instruction selection and register allocation.

¶ **JF goes for the remaining `__attribute__((*))`!**


- **Wisdom:** this is a bit self serving... but we're on stage giving a keynote at the LLVM developer's conference, so some people must think this is a wise idea.
- **Charisma:** it's *of the Web*, the internet loves it.

► Others:

- **Armor class:** Internet-proof. Security is our thing. Learn more on security from [JF's CppCon talk](#).
- **Initiative:** the announcement [seems to have taken folks by surprise](#) (more than we expected!). Either we rolled high or had a large base modifier.
- **Speed:** near-native. PNaCl and asm.js have shown code could go pretty fast, often on par with native, and we think WebAssembly can surpass either.
- ► **Languages:** C/C++ to start with, but we want to be polyglots.
- **Personality traits:** whimsy.
- **Ideals:** speed, security, portability, ephemerality. Note the last 3 are the Web's ideals, WebAssembly adds speed.
- **Bonds:** the Web should be able to do everything native code can.
- **Flaws:** it would be a bit silly to design WebAssembly with purposeful flaws. We'll see as it matures.

Bring the latest and greatest C++14 to the web, portably and efficiently (see [goals](#)). WebAssembly also wants to adventure in the [non-Web parts of the world](#). It should also work on servers and tiny IoT devices.

Call to action: what can *you* do to be the hero?



Hardcore developer details

- ❖ Inventory check
- ❖ Cantrips

¶ JF con't.

Let's dive into the details.

¶ **Dan:** WebAssembly's first surprise encounter, and with a real experienced C++ programmer. "I know how these things go, you take away my pointers! Engarde!"

¶ JF takes action.

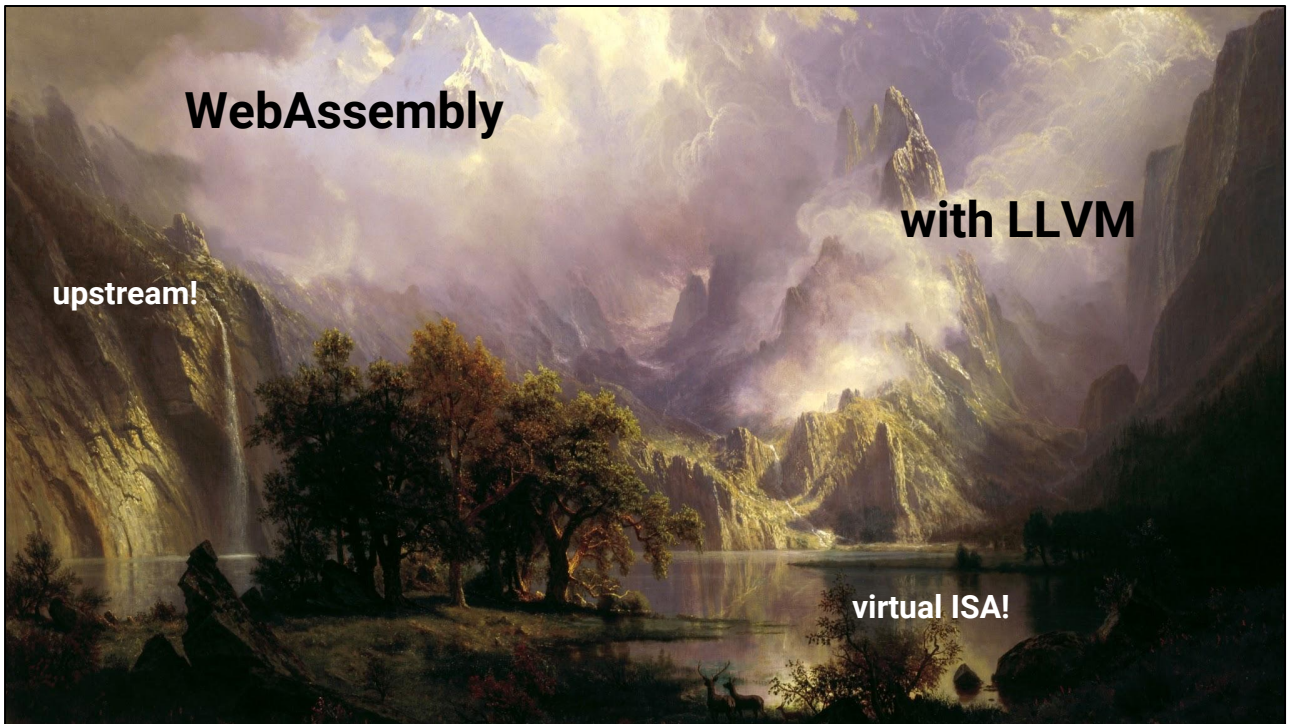
- ▶ What's in our current bag of holdings? Let's do an inventory check:
 - "linear memory" is virtual memory. Cache locality.
 - Pointers for serious.
 - Function pointers too, in a way.
 - Even pointer to member functions if you're into that type of thing.
 - vtables.
 - APIs on the Web, based on SDK + helper libraries that "just work":
 - OpenGL → WebGL (GLES3 will soon be in Firefox and Chrome).
 - SDL → WebAudio.
 - C++/POSIX filesystems: whatever the web has... It's still improving in that area.
 - Networking: WebSocket / WebRTC.

¶ Dan seizes the conch shell!

- ▶ Expected to quickly learn the following cantrips:

- [Threads/atomics/shared memory.](#)
- [SIMD.](#)
- [Zero-cost EH.](#)
- [Dynamic linking.](#)

This will round out a fairly complete C++ platform.



But how do you do this?

Learned from Emscripten / PNaCl mistakes.

WebAssembly is developed [straight in upstream LLVM](#) under `lib/Target/WebAssembly`, mistakes and design changes included.

We're fixing other parts of the LLVM codebase too, making life easier for other virtual ISAs.

¶ JF jumps in!

Virtual ISA: what's that? We talk about it a lot like it a big mythical beast.

- Compile on the developer's machine, then lower to different ISAs on the user's machine. There's a second compiler that runs once we know what the architecture is. In browsers it'll share code with the JavaScript engine. There are 10 implementations of this in progress, 3 of the prototypes use LLVM!
- Browser as an OS (or in general "embedder": WebAssembly [isn't just a browser thing](#)).
- We must make certain assumptions about the target for code layout and performance:
 - 8-bit bytes.
 - Two's complement integers.
 - IEEE 754-2008 32-bit and 64-bit floating point.
 - Developers choose either 32-bit or 64-bit pointers.
 - Little-endian machine.

- ...and [more](#).
- How do we stay sane? Develop an extensive [test suite](#).
- What other virtual ISAs are in or proposed for LLVM?
 - NVPTX
 - BPF
 - AMDGPU
 - CppBackend
 - SPIR-V
 - HSAIL

Undefined behavior, implementation defined, unspecified, etc. [WHAT DOES IT MEAN?](#)
C++ developers all know of UB, they're afraid that it'll snatch their children in the night and that demons will fly out of their nose.

- Clang and LLVM already eliminate a lot of the leeway C++ gives compilers.
- Progressive refining of undefined behavior.
- WebAssembly nails most of the rest down.
 - Dividing by zero traps.
 - Out of bounds accesses trap.
 - A malicious program can't harm the user directly (developers should still be careful with the data their WebAssembly modules contains).
- Some still remains, for example threads can still race but offer limited local nondeterminism.

Key takeaway: Developers can really think of this as an ISA.

It turns out LLVM IR is a compiler IR.

Compiler Internals: Instruction Selection

```
defn ADD : BinaryInt<add>;
defn SUB : BinaryInt<sub>;
defn MUL : BinaryInt<mul>;
defn DIV_S : BinaryInt<sdiv>;
defn DIV_U : BinaryInt<udiv>;
defn REM_S : BinaryInt<rem>;
defn REM_U : BinaryInt<urem>;
defn AND : BinaryInt<and>;
defn OR : BinaryInt<or>;
defn XOR : BinaryInt<xor>;
defn SHL : BinaryInt<shl>;
defn SHR_U : BinaryInt<shr>;
defn SHR_S : BinaryInt<shra>;

defn EQ : ComparisonInt<SETEQ>;
defn NE : ComparisonInt<SETNE>;
defn LT_S : ComparisonInt<SETLT>;
defn LE_S : ComparisonInt<SETLE>;
defn LT_U : ComparisonInt<SETULT>;
defn LE_U : ComparisonInt<SETULE>;
defn GT_S : ComparisonInt<SETGT>;
defn GE_S : ComparisonInt<SETGE>;
defn GT_U : ComparisonInt<SETGTO>;
defn GE_U : ComparisonInt<SETGTE>;

defn CLZ : UnaryInt<clz>;
defn CTZ : UnaryInt<cttz>;
defn POPCNT : UnaryInt<ctpop>;

def : Pat<(clz_zero_undef I32:$src), (CLZ_I32 I32:$src)>;
def : Pat<(clz_zero_undef I64:$src), (CLZ_I64 I64:$src)>;
def : Pat<(cttz_zero_undef I32:$src), (CTZ_I32 I32:$src)>;
def : Pat<(cttz_zero_undef I64:$src), (CTZ_I64 I64:$src)>;
```

```
def LOAD_I32 : I<(outs I32:$dst), (ins I32:$addr),
  [(set I32:$dst, (load I32:$addr))]>;
def LOAD_I64 : I<(outs I64:$dst), (ins I32:$addr),
  [(set I64:$dst, (load I32:$addr))]>;
def LOAD_F32 : I<(outs F32:$dst), (ins I32:$addr),
  [(set F32:$dst, (load I32:$addr))]>;
def LOAD_F64 : I<(outs F64:$dst), (ins I32:$addr),
  [(set F64:$dst, (load I32:$addr))]>;

def LOAD8_S_I32 : I<(outs I32:$dst), (ins I32:$addr),
  [(set I32:$dst, (sextloadi8 I32:$addr))]>;
def LOAD8_U_I32 : I<(outs I32:$dst), (ins I32:$addr),
  [(set I32:$dst, (zextloadi8 I32:$addr))]>;
def LOAD16_S_I32 : I<(outs I32:$dst), (ins I32:$addr),
  [(set I32:$dst, (sextloadi16 I32:$addr))]>;
def LOAD16_U_I32 : I<(outs I32:$dst), (ins I32:$addr),
  [(set I32:$dst, (zextloadi16 I32:$addr))]>;
def LOAD8_S_I64 : I<(outs I64:$dst), (ins I32:$addr),
  [(set I64:$dst, (sextloadi8 I32:$addr))]>;
def LOAD8_U_I64 : I<(outs I64:$dst), (ins I32:$addr),
  [(set I64:$dst, (zextloadi8 I32:$addr))]>;
def LOAD16_S_I64 : I<(outs I64:$dst), (ins I32:$addr),
  [(set I64:$dst, (sextloadi16 I32:$addr))]>;
def LOAD16_U_I64 : I<(outs I64:$dst), (ins I32:$addr),
  [(set I64:$dst, (zextloadi16 I32:$addr))]>;
def LOAD32_S_I64 : I<(outs I64:$dst), (ins I32:$addr),
  [(set I64:$dst, (sextloadi32 I32:$addr))]>;
def LOAD32_U_I64 : I<(outs I64:$dst), (ins I32:$addr),
  [(set I64:$dst, (zextloadi32 I32:$addr))]>;

def : Pat<(i32 (extloadi8 I32:$addr)), (LOAD8_U_I32 $addr)>;
def : Pat<(i32 (extloadi16 I32:$addr)), (LOAD16_U_I32 $addr)>;
def : Pat<(i64 (extloadi8 I32:$addr)), (LOAD8_U_I64 $addr)>;
def : Pat<(i64 (extloadi16 I32:$addr)), (LOAD16_U_I64 $addr)>;
def : Pat<(i64 (extloadi32 I32:$addr)), (LOAD32_U_I64 $addr)>;
```

```
defn ADD : BinaryFP<fadd>;
defn SUB : BinaryFP<fsub>;
defn MUL : BinaryFP<fmul>;
defn DIV : BinaryFP<fdiv>;
defn SQRT : UnaryFP<fsqrt>;

defn ABS : UnaryFP<fabs>;
defn NEG : UnaryFP<fneg>;
defn COPYSIGN : BinaryFP<fcopysign>;

defn CEIL : UnaryFP<fceil>;
defn FLOOR : UnaryFP<ffloor>;
defn TRUNC : UnaryFP<ftrunc>;
defn NEAREST : UnaryFP<fnearbyint>;

def : Pat<(frint F32:$src), (NEAREST_F32 F32:$src)>;
def : Pat<(frint F64:$src), (NEAREST_F64 F64:$src)>;

defn EQ : ComparisonFP<SETEQ>;
defn NE : ComparisonFP<SETNE>;
defn LT : ComparisonFP<SETOLT>;
defn LE : ComparisonFP<SETOLE>;
defn GT : ComparisonFP<SETOGT>;
defn GE : ComparisonFP<SETOGE>;

def : Pat<(seteq F32:$lhs, F32:$rhs), (EQ_F32 F32:$lhs, F32:$rhs)>;
def : Pat<(seteq F64:$lhs, F64:$rhs), (EQ_F64 F64:$lhs, F64:$rhs)>;
def : Pat<(setlt F32:$lhs, F32:$rhs), (LT_F32 F32:$lhs, F32:$rhs)>;
def : Pat<(setle F32:$lhs, F32:$rhs), (LE_F32 F32:$lhs, F32:$rhs)>;
def : Pat<(setgt F32:$lhs, F32:$rhs), (GT_F32 F32:$lhs, F32:$rhs)>;
def : Pat<(setge F32:$lhs, F32:$rhs), (GE_F32 F32:$lhs, F32:$rhs)>;
def : Pat<(seteq F64:$lhs, F64:$rhs), (EQ_F64 F64:$lhs, F64:$rhs)>;
def : Pat<(setle F64:$lhs, F64:$rhs), (LE_F64 F64:$lhs, F64:$rhs)>;
def : Pat<(setlt F64:$lhs, F64:$rhs), (LT_F64 F64:$lhs, F64:$rhs)>;
def : Pat<(setle F64:$lhs, F64:$rhs), (LE_F64 F64:$lhs, F64:$rhs)>;
def : Pat<(setgt F64:$lhs, F64:$rhs), (GT_F64 F64:$lhs, F64:$rhs)>;
def : Pat<(setge F64:$lhs, F64:$rhs), (GE_F64 F64:$lhs, F64:$rhs)>;
```

All the same: up to instruction selection, pretty much everything is the same as for the rest of LLVM's targets.

The land of LLVM is a land of plenty.

C++ vtables and varargs may be slightly different but this isn't be completely novel.

This is mostly easy; it helps when we get to design [our own instruction set](#) :-).

☛ Dan agrees. Rolls a d20; critical hit!

► Put InstrInteger.td, InstMemory.td and InstFloat.td on the screen.

This is all of WebAssembly's integer, load and floating-point operations. Look: it's simple!

SelectionDAG is the way to do isel in LLVM.

Legalization: yes please. And it's what every other target uses, so it's well tested and has lots of optimizations built in.

Compiler Internals: Register Allocation

Reduced code size

Register coloring



Going down a level...

Register allocation: that's what MI is all about.

Minimize the total number of virtual registers.

► Virtual register coloring as hinting for the WebAssembly register allocator.

Pre-coalesce for faster and better code generation in the browser.

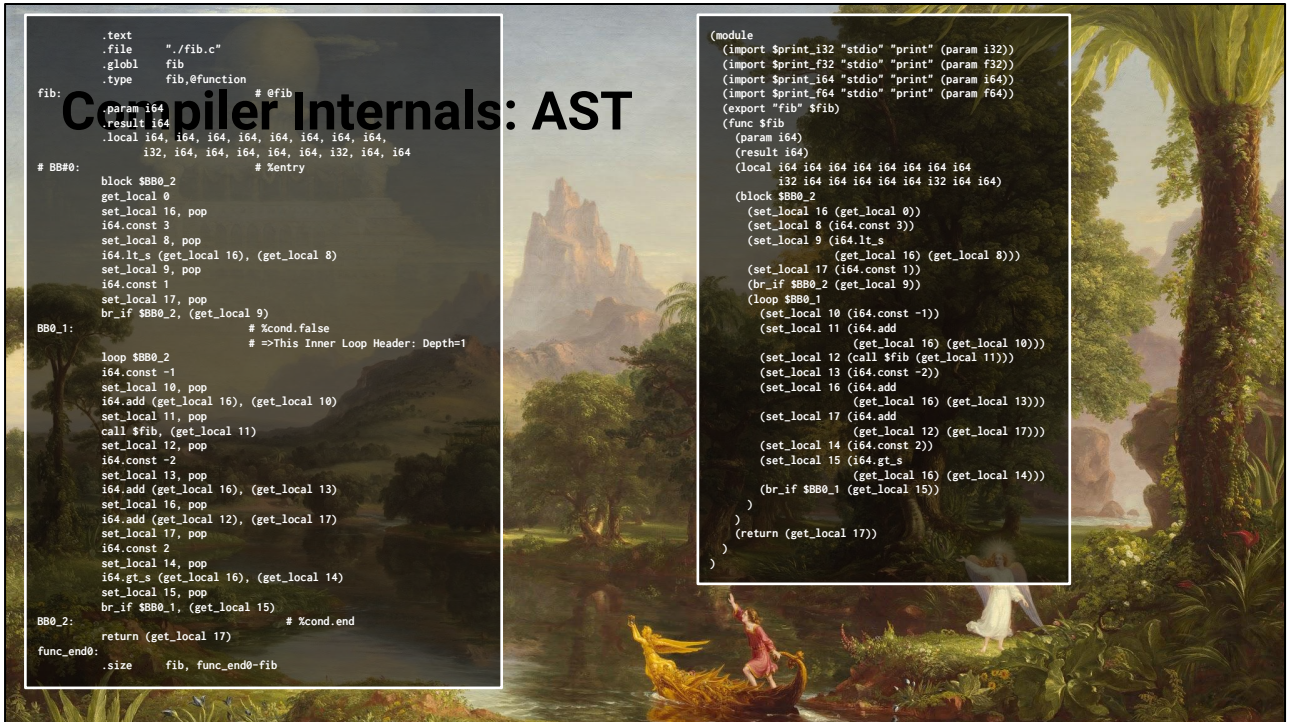
Our adventurer has too many registers in his knapsack. How will he pack them?

Ideas welcome! How can *you* be the hero?

¶ **JF's turn comes up. He shows his colors (red primary, splashing some green).**

Naming local variables is one of the biggest contributors to code size.

► Use expression trees effectively to reduce redundancy.



Expression trees allow us to reassociate code and use recent values. We go even further, expressing control flow as ASTs instead of CFGs. This moves some of the restructuring to developer's machine instead of being the user's device. We're also thinking that making statement == expression might work out well for our goals.

Why? Encoding size.



Look at those beautiful s-expressions, we could just feed them to a LISP and eval everything and C++ would just execute out of that environment and...

¶ Dan jumps in. WOH THERE!

The JF appears to be confused. ▶▶

The output looks like assembly!

MC.

Syntax is just syntax, these are interchangeable.

Tried bundling. Almost made sense, but awkward.

- It requires many passes be aware of bundling.
- Bundling can't embed things like control flow.

Language Frontends

C++ ABI "v2"?

¶ **JF** Of course, most developers don't want to write assembly by hand, but with just a backend *you shall not pass!*

¶ **Dan** Unless you have a frontend. ▶▶

Using clang's out of the box feature set for most things
Get rid of C++ overhead which other targets are stuck with.

▶▶ C++ ABI v2

There's been discussion about what a C++ ABI "v2" might look like, beyond the "v1" which is present-day Itanium. This is a fun chance to do new ABI design, which isn't something that one often gets a chance to do. Clang has a few of these implemented, and we're using them in WebAssembly now, but several opportunities remain, with some ideas in this thread:

<http://sourcerytools.com/pipermail/cxx-abi-dev/2013-November/002619.html>

As another C++ ABI "v2" feature, Clang can do RTTI type_info objects which aren't unique across a program, but the code is currently ARM64-specific. It'd be nice to implement this functionality for WebAssembly too. See this thread for more info:

<http://sourcerytools.com/pipermail/cxx-abi-dev/2013-November/002623.html>

¶ **JF** stumbles onto the battlefield.

Another notable idea of a C++ ABI “v2” feature; optimizing guard variable synchronization:

<http://sourcerytools.com/pipermail/cxx-abi-dev/2015-May/002848.html>

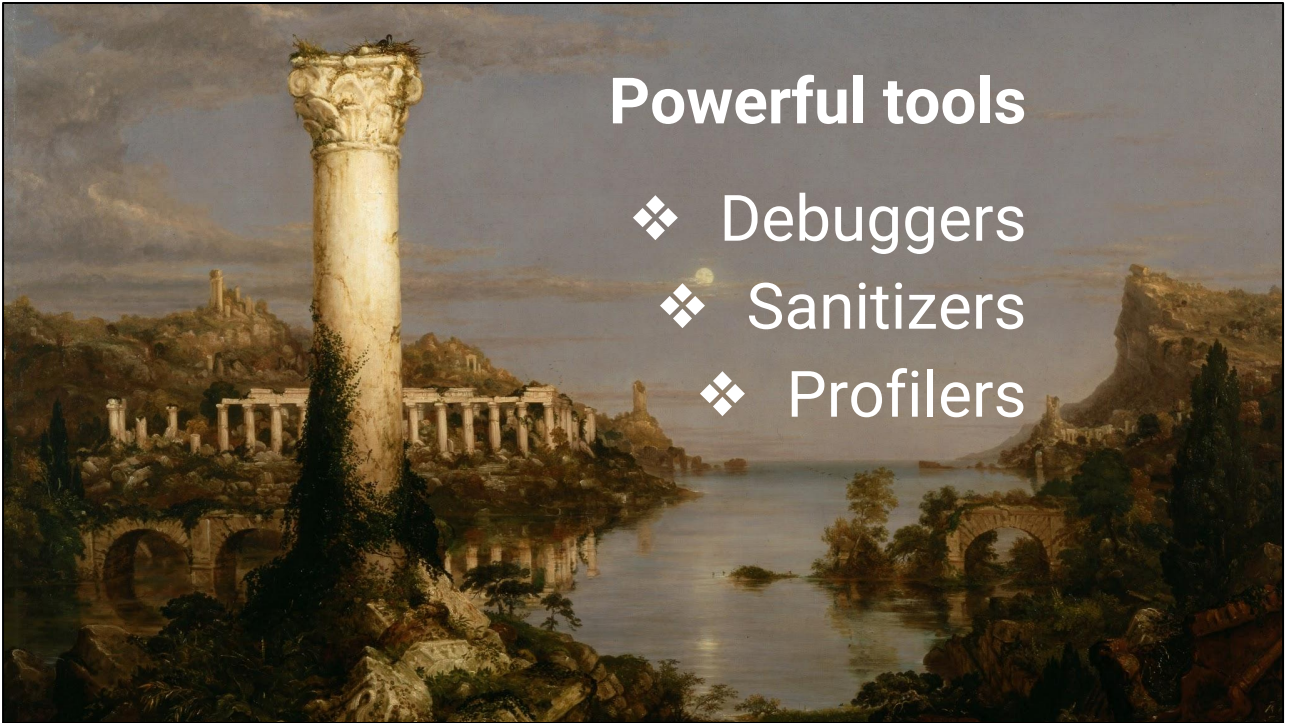
Function pointers, CFI, reduce overhead?

Stronger type checking.

vtables?

We could do more with your help! Neither of us are C++ ABI experts.

This is a battleground for optimizations.



Powerful tools

- ❖ Debuggers
- ❖ Sanitizers
- ❖ Profilers

We're developing LLVM as the first WebAssembly C/C++ compiler. Not to be distasteful at the LLVM developer's conference but we hope other compilers play along as well.

▶ Can do cool tricks with powerful tools, the first we'll build is a toolchain. Choose your own adventure; pick the libm that gives you the performance/precision tradeoff right for your app!

Remember: everything is open source and built in the open. [You can do anything in WebAssembly. Anything at all. The only limit is yourself. Welcome ... to WebAssembly.](#)

¶ Dan tools in. /rimshot

We want a great C++ development story, and that means we're planning for all the tools that modern C++ developers depend on:

- ▶ Debuggers, integrated in the browser and outside of the browser.
- ▶ Sanitizers.
- ▶ Profilers.

Tooling.md ideas.

Oracle of the Future

- ❖ More languages
- ❖ JIT-compilation
- ❖ GC
- ❖ And more!

Reiterate the “soon after” cantrips from slide 3:

- [Threads/atomics/shared memory](#).
- [SIMD](#).
- [Zero-cost EH](#).
- [Dynamic linking](#).

Dan challenges us: this is a good platform but what’s going to take it to the next level?
Do we know what’s out there?

We’ll try to acquire new magical artifacts and achieve true glory: ►►

- Rust
- Go
- C#
- Python
- And anything else you fancy!

¶ **JF peers into the future with his crystal ball. We’ve got this!**

More features, more levelups! How are we going to do these other languages?

- ►► JIT-compilation: WebAssembly is a new ISA that compilers would need to target.
- ►► GC
- Finer-grained memory control.
- Asynchronous signals.

- Many things are possible!



JF Bastien
Google
@jfbastien

Dan Gohman
Mozilla
@sunfishcode

Our adventurer is now back at the tavern, enjoying a good brew... What's the next quest?

We have a few things working, [LLVM can generate toy code and execute it inside a browser](#).

We have many other adventures ahead of us.

¶ **JF and Dan:** Will you [join us](#)?

[Get started!](#)