Building a faster expression evaluator for LLDB

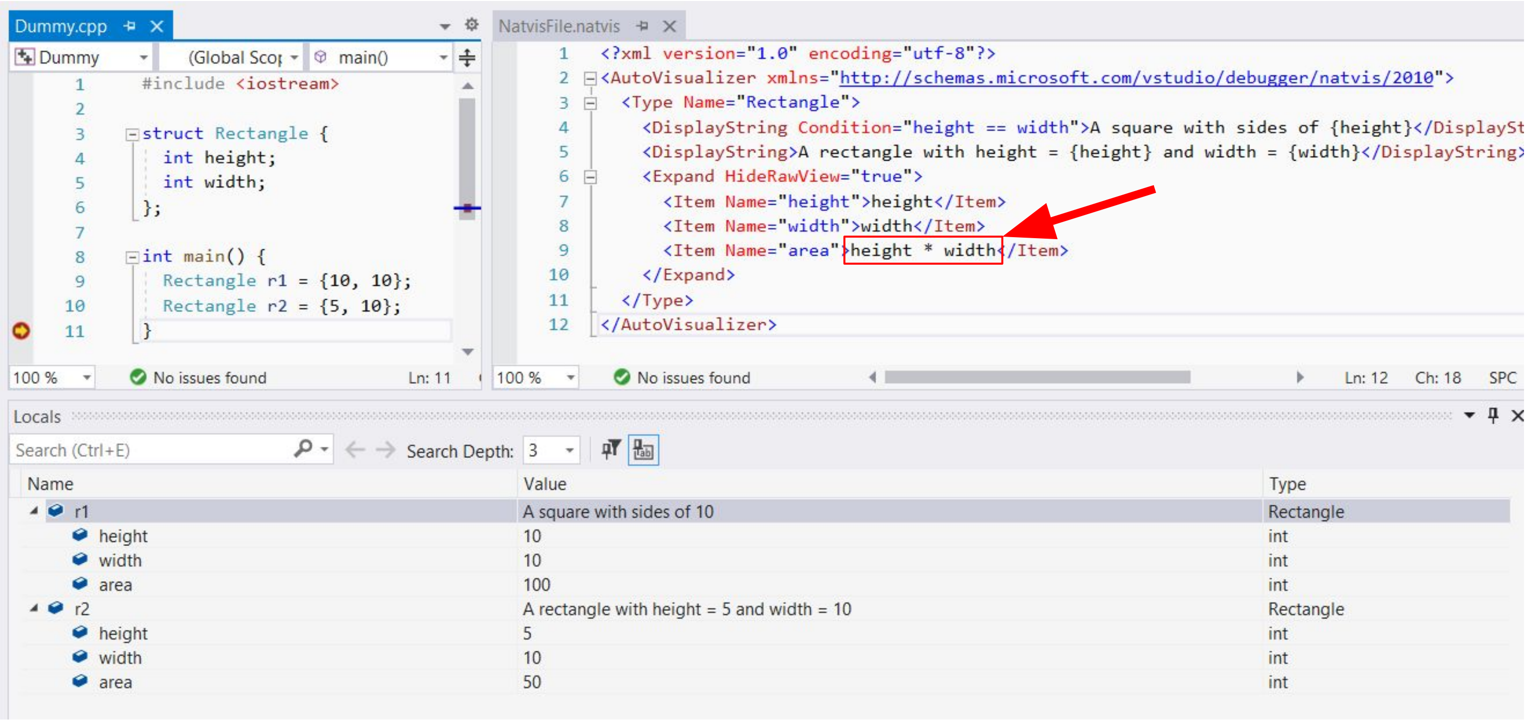
Andy Yankovsky, Google @werat

LLVM Developers' Meeting 2021

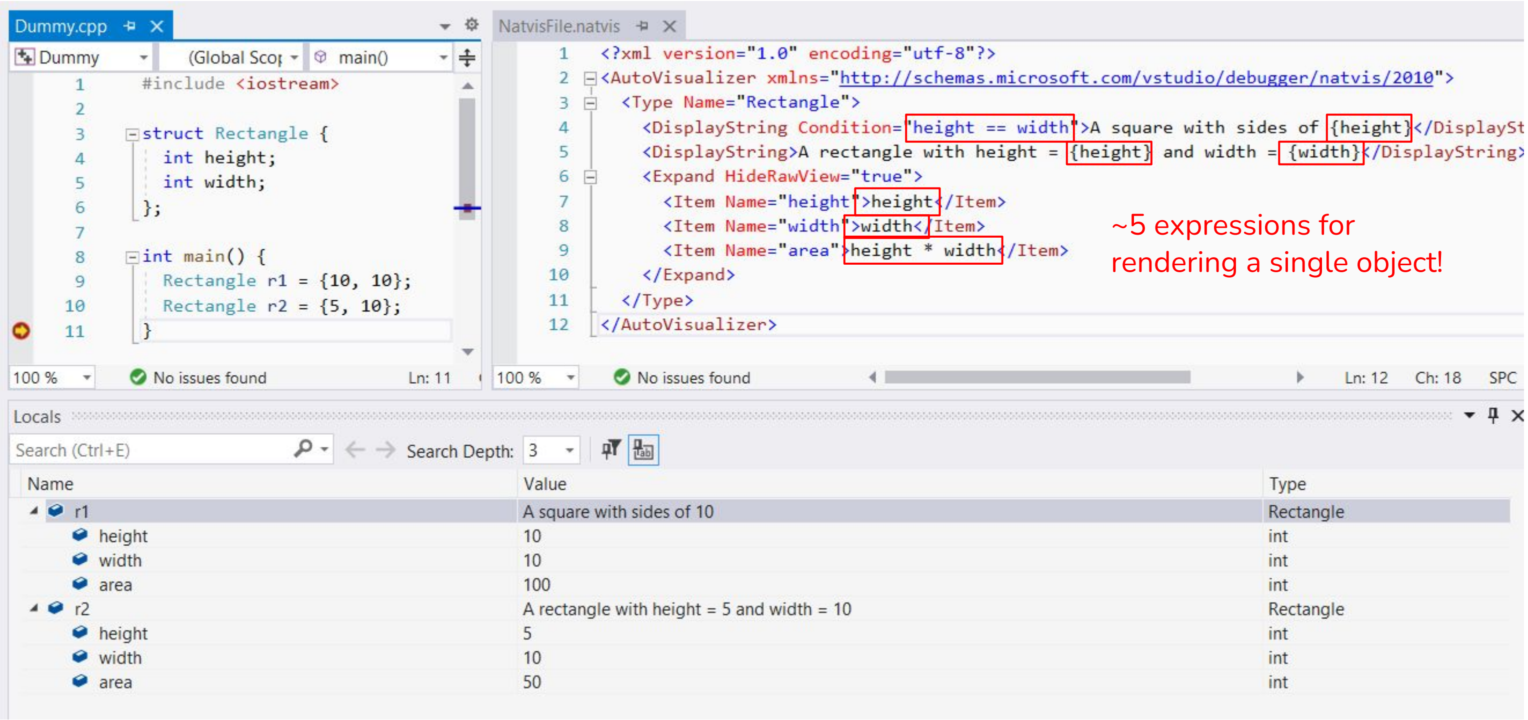
Stadia for Visual Studio debugger

- Visual Studio extension implementing typical developer workflows
  - https://github.com/googlestadia/vsi-lldb
- Build & Run & Debug the game in the Cloud™
- Debugger is based on LLDB...
- ... and supports NatVis!

Custom object visualizers (NatVis) #1



Custom object visualizers (NatVis) #2



Custom object visualizers (NatVis) #3

clang/utils/ClangVisualizers/clang.natvis

The whole visualizer is 84 lines >_<

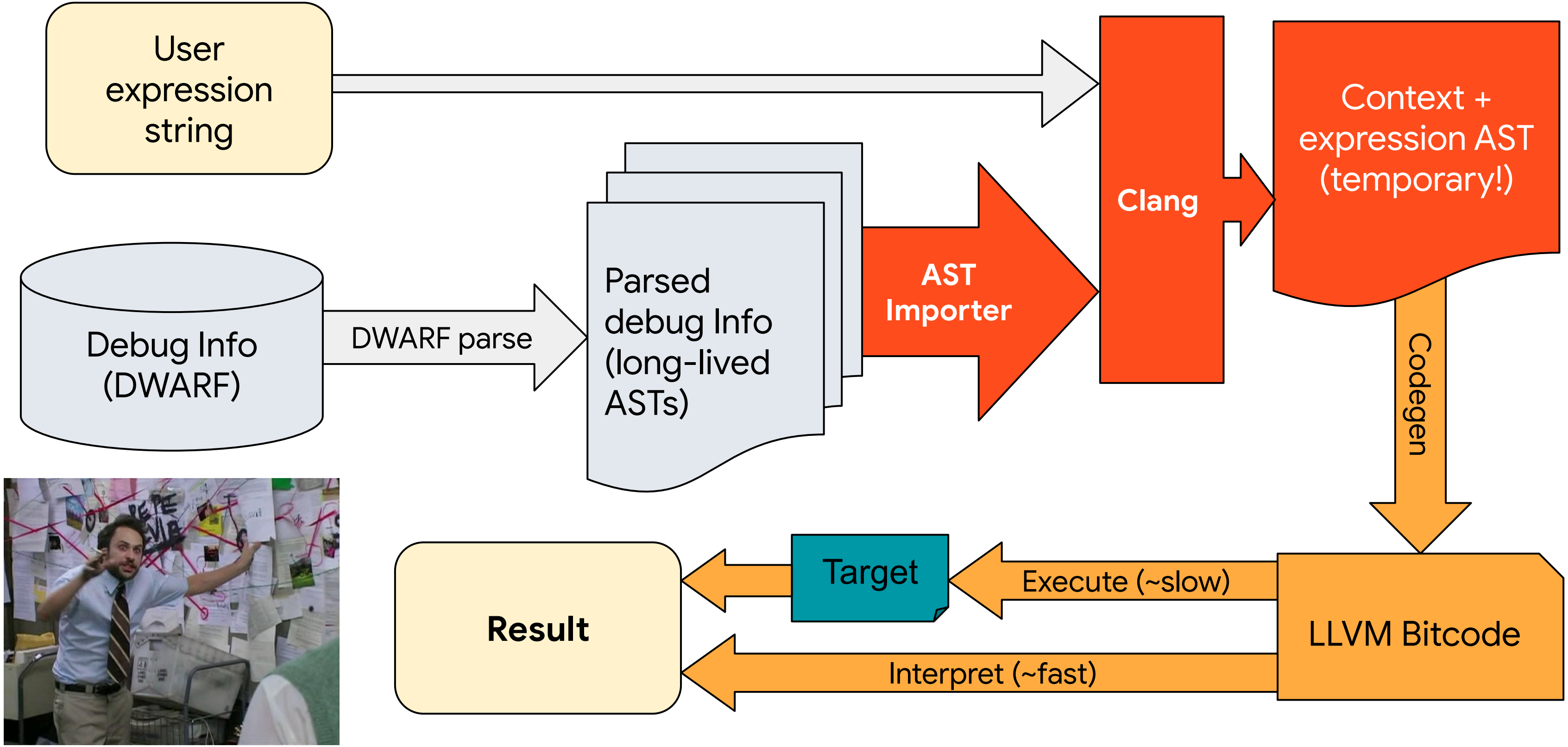
```
<Type Name="clang::Type">
  <!-- To visualize clang::Types, we need to look at TypeBits.TC to determine the actual
       type subclass and manually dispatch accordingly (Visual Studio can't identify the real type
       because clang::Type has no virtual members hence no RTTI).

       Views:
         "cmn": Visualization that is common to all clang::Type subclasses
         "poly": Visualization that is specific to the actual clang::Type subclass. The subtype-specific
                 <DisplayString> is typically as C++-like as possible (like in dump()) with <Expand>
                 containing all the gory details.
         "cpp": Only occasionally used when we need to distinguish between an ordinary view and a C++-like view.
  -->
  <DisplayString IncludeView="cmn" Condition="TypeBits.TC==clang::LocInfoType::LocInfo">LocInfoType</DisplayString>
  <DisplayString IncludeView="cmn">{(clang::Type::TypeClass)TypeBits.TC, en}Type</DisplayString>
  <!-- Dispatch to visualizers for the actual Type subclass -->
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::Builtin" IncludeView="poly">{*(clang::BuiltinType *)this}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::Pointer" IncludeView="poly">{*(clang::PointerType *)this}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::LValueReference" IncludeView="poly">{*(clang::LValueReferenceType *)this}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::RValueReference" IncludeView="poly">{*(clang::RValueReferenceType *)this}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::ConstantArray" IncludeView="poly">{(clang::ConstantArrayType *)this,na}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::ConstantArray" IncludeView="left">{(clang::ConstantArrayType *)this,view(left)na}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::ConstantArray" IncludeView="right">{(clang::ConstantArrayType *)this,view(right)na}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::IncompleteArray" IncludeView="poly">{(clang::IncompleteArrayType *)this,na}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::IncompleteArray" IncludeView="left">{(clang::IncompleteArrayType *)this,view(left)na}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::IncompleteArray" IncludeView="right">{(clang::IncompleteArrayType *)this,view(right)na}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::Typedef" IncludeView="poly">{(clang::TypedefType *)this,na}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::Typedef" IncludeView="cpp">{(clang::TypedefType *)this,view(cpp)na}</DisplayString>
  <DisplayString Condition="TypeBits.TC==clang::Type::TypeClass::Attributed" IncludeView="poly">{*(clang::AttributedType *)this}</DisplayString>
```

Expression evaluation in LLDB

- lldb::SBValue::GetValueForExpressionPath()
  - Expands nested expressions like a->b[0].c[1]->d
  - Very fast – ~0.1 ms per expression
- lldb::SBFrame::EvaluateExpression() // also exists for SBTarget and SBValue
  - Can handle almost any valid C++
  - Can be quite slow – ~50-100 ms per expression

... Can we make it faster?

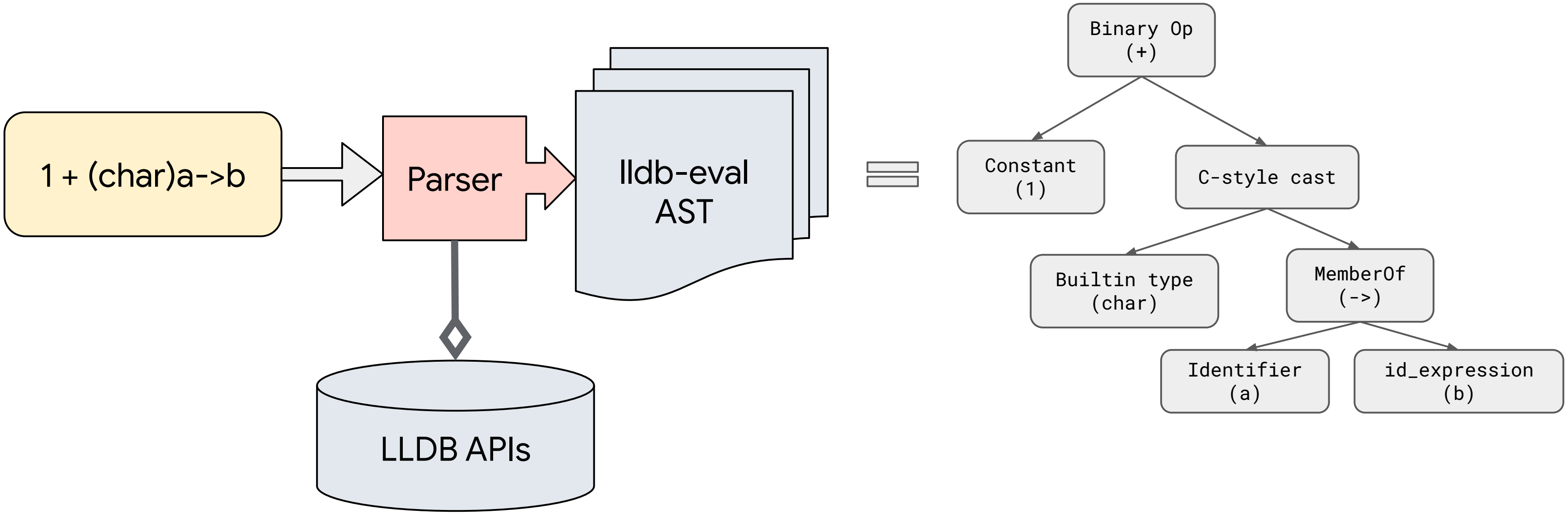
Expression evaluation in LLDB



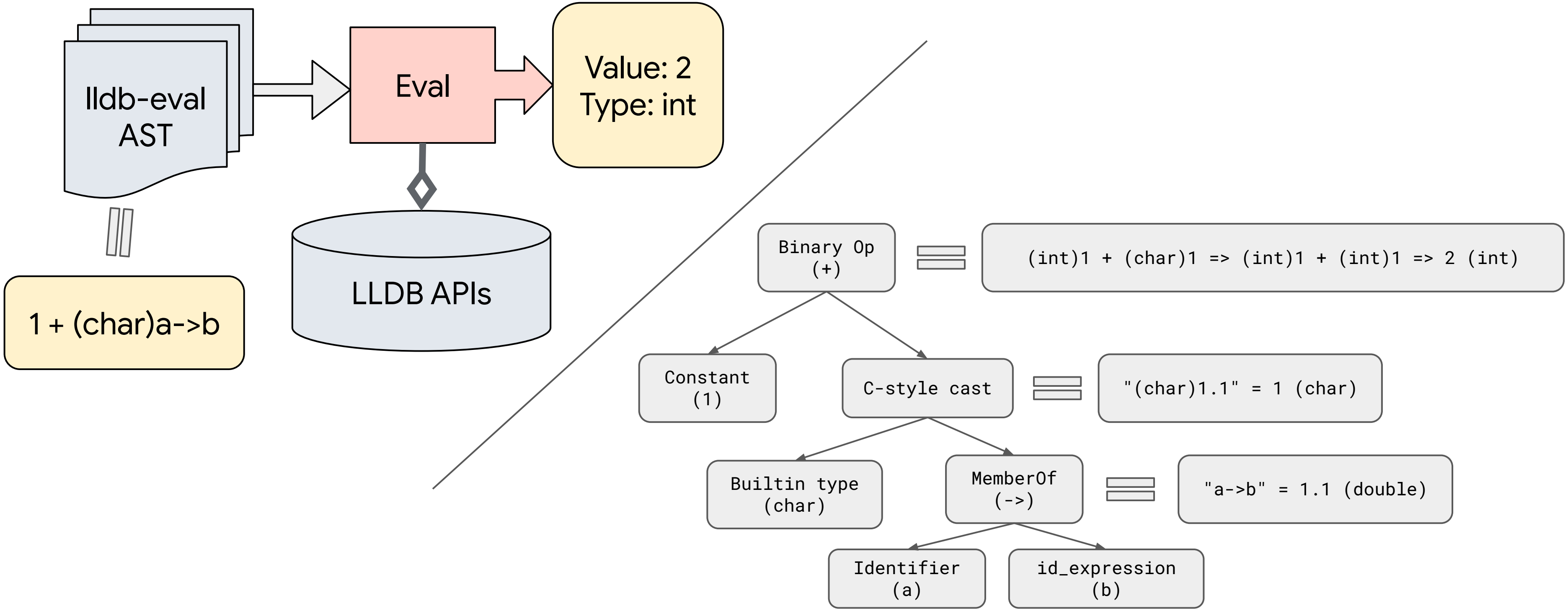
We need to go faster – lldb-eval

- A library, (almost) drop-in replacement
  - http://github.com/google/lldb-eval
- Features:
  - Fast – <1 ms per evaluation
  - All basic operations – arithmetic, member access, type casts, etc.
  - Builtin intrinsic functions (e.g. __log2, __findnonnull)
- Limitations:
  - No user function calls (yet)
  - No constant-time evaluation (e.g. Foo<1+2>::bar)

Life of an expression in lldb-eval #1

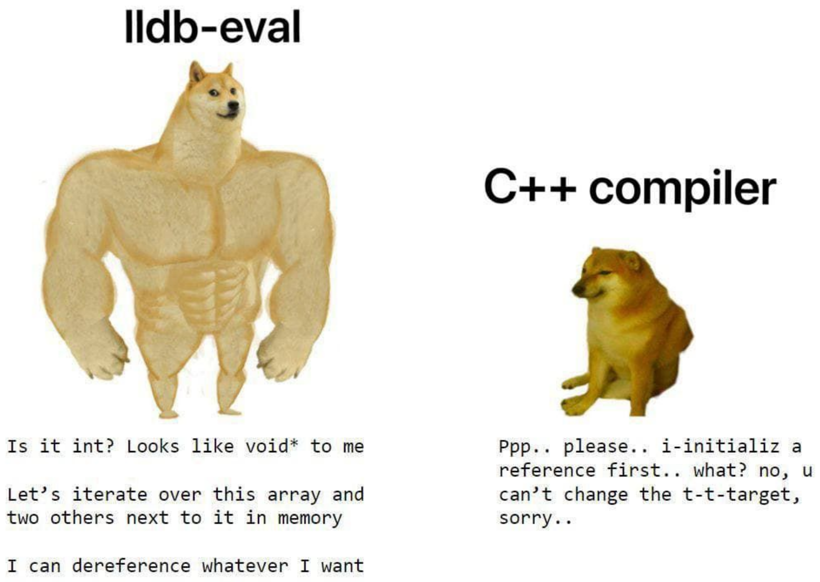


Life of an expression in lldb-eval #2



How come lldb-eval is faster?

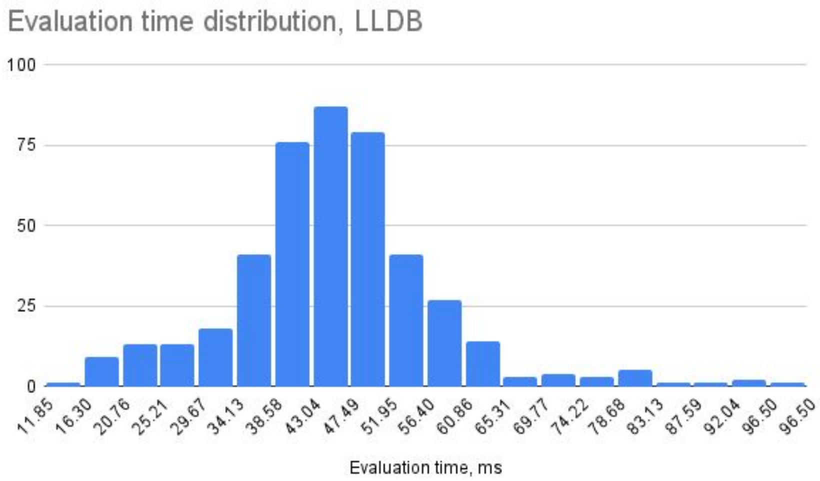
- LLDB uses Clang and Clang is a *compiler* – it needs to resolve everything and the expression AST may end up very large
- lldb-eval tries to be as lazy as possible – requests only the information needed
- lldb-eval is basically a re-implementation of Clang Frontend – specialized, hacky, fast!

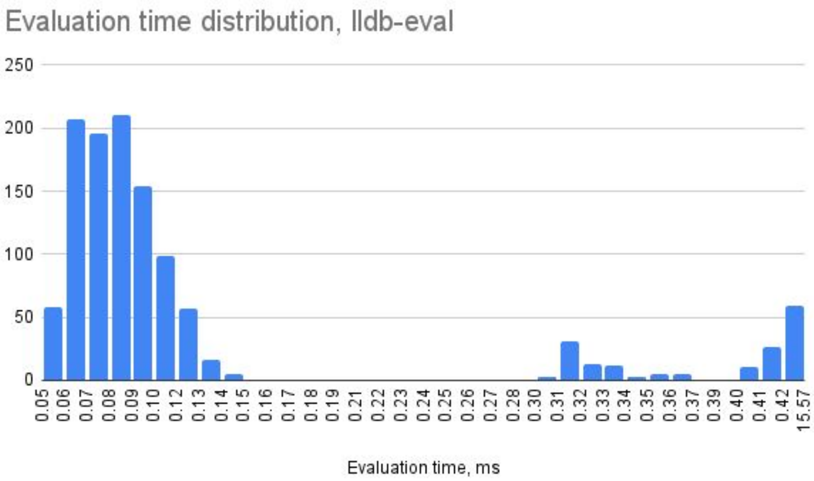


Performance comparison – benchmark setup

- Unreal Engine 4, Infiltrator Demo
  - https://www.youtube.com/watch?v=dO2rM-l-vdQ
- Linux executable
  - Binary – ~150 MB
  - Symbols – ~1800 MB
- Expand an object of type UWorld
  - Triggers evaluation of ~300 expressions

Performance comparison





300 expressions * 50ms ≈ **15 seconds**!

300 expressions * 0.3ms ≈ **90 milliseconds**!

References

- Blazing fast expression evaluation for C++ in LLDB
  - https://werat.dev/blog/blazing-fast-expression-evaluation-for-c-in-lldb/
- R. Isemann “Better C++ debugging using Clang Modules in LLDB”
  - https://www.youtube.com/watch?v=vuNZLlHhy0k