Hacking Sema for clang-repl

Stefan Gränitz // LLVM Meetup Berlin // 24 April 2024

Hacking Sema? for clang-repl?

Stefan Gränitz // LLVM Meetup Berlin // 24 April 2024

What is clang-repl?

Incremental C++ prompt in upstream LLVM

```
➜ git clone https://github.com/llvm/llvm-project
➜ cd llvm-project
➜ git switch release/18.x
➜ cmake -Bbuild -Sllvm -GNinja -DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD=host
➜ ninja -C build clang-repl
```

What is clang-repl?

Incremental C++ prompt in upstream LLVM

```
➜ git clone https://github.com/llvm/llvm-project
➜ cd llvm-project
➜ git switch release/18.x
➜ cmake -Bbuild -Sllvm -GNinja -DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD=host
➜ ninja -C build clang-repl
➜ build/bin/clang-repl
clang-repl> int a = 1;
clang-repl> int b = a + 1;
clang-repl> extern "C" int printf(const char*,...);
clang-repl> printf("%d\n", b);
2
```

What is clang-repl?

Incremental C++ prompt in upstream LLVM

```
➜ git clone https://github.com/llvm/llvm-project
➜ cd llvm-project
➜ git switch release/18.x
➜ cmake -Bbuild -Sllvm -GNinja -DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD=host
➜ ninja -C build clang-repl
➜ build/bin/clang-repl
clang-repl> int a = 1;
clang-repl> int b = a + 1;
clang-repl> extern "C" int printf(const char*,...);
clang-repl> printf("%d\n", b);
2
clang-repl> b
Not implement yet.
```

What means incremental?

```
➜ build/bin/clang-repl -Xcc -Xclang -Xcc -ast-dump
clang-repl> int a = 1;
TranslationUnitDecl 0x7fcb08032c88 prev 0x7fcb0880dd70 <<invalid sloc>> <invalid sloc>
`-VarDecl 0x7fcb08032d08 <input_line_1:1:1, col:9> col:5 a 'int' cinit
  `-IntegerLiteral 0x7fcb08032d70 <col:9> 'int' 1
```

What means incremental?

```
➜ build/bin/clang-repl -Xcc -Xclang -Xcc -ast-dump
clang-repl> int a = 1;
TranslationUnitDecl 0x7fcb08032c88 prev 0x7fcb0880dd70 <<invalid sloc>> <invalid sloc>
`-VarDecl 0x7fcb08032d08 <input_line_1:1:1, col:9> col:5 a 'int' cinit
  `-IntegerLiteral 0x7fcb08032d70 <col:9> 'int' 1

clang-repl> int b = 1 + a;
TranslationUnitDecl 0x7fcb08032df0 prev 0x7fcb08032c88 <<invalid sloc>> <invalid sloc>
`-VarDecl 0x7fcb08032e70 <input_line_2:1:1, col:13> col:5 b 'int' cinit
  `-BinaryOperator 0x7fcb08032f30 <col:9, col:13> 'int' '+'
    |-IntegerLiteral 0x7fcb08032ed8 <col:9> 'int' 1
    `-ImplicitCastExpr 0x7fcb08032f18 <col:13> 'int' <LValueToRValue>
      `-DeclRefExpr 0x7fcb08032ef8 <col:13> 'int' lvalue Var 0x7fcb08032d08 'a' 'int'
```

Clang static compilation pipeline

Every compile unit has its own track

$$\begin{tikzcd}
\text{Lex} \arrow[r, shift left] & \text{Parse} \arrow[l, shift left] \arrow[r, shift left] & \text{Sema} \arrow[l, shift left] \arrow[r] & \text{Codegen} \arrow[r] & \text{Opt} \arrow[r] & \text{Backend} \arrow[dr] & \\
\text{Lex} \arrow[r, shift left] & \text{Parse} \arrow[l, shift left] \arrow[r, shift left] & \text{Sema} \arrow[l, shift left] \arrow[r] & \text{Codegen} \arrow[r] & \text{Opt} \arrow[r] & \text{Backend} \arrow[r] & \text{Linker} \\
 & & & & & \text{...} \arrow[ur] &
\end{tikzcd}$$

Clang incremental mode

Frontend state is continuous

$$\begin{tikzcd}
\text{Lex} \arrow[dr, Leftrightarrow] & & & & & \text{Opt} \arrow[r] & \text{Backend} \arrow[dr] & \\
\text{Lex} \arrow[r, Leftrightarrow] & \text{Parse} \arrow[r, rightleftarrows] & \text{Sema} \arrow[r] & \text{Codegen} \arrow[r] & \text{Orc} \arrow[ur] \arrow[r] \arrow[dr] & \text{Opt} \arrow[r] & \text{Backend} \arrow[r] & \text{JITLink} \\
\text{...} \arrow[ur, Leftrightarrow] & & & & & \text{...} & \text{...} \arrow[ur] &
\end{tikzcd}$$

What means REPL?

read-evaluate-PRINT loop

Value printing needs a runtime

```
➜ git clone https://github.com/llvm/llvm-project
➜ cd llvm-project
➜ git switch release/18.x
➜ cmake -Bbuild -Sllvm -GNinja -DCMAKE_BUILD_TYPE=... -DLLVM_TARGETS_TO_BUILD=host
➜ ninja -C build clang-repl
➜ build/bin/clang-repl
clang-repl> int a = 1;
clang-repl> int b = a + 1;
clang-repl> extern "C" int printf(const char*,...);
clang-repl> printf("%d\n", b);
2
clang-repl> b
Not implement yet.
```

RuntimeInterfaceBuilder

```
clang-repl> int a = 1;
clang-repl> int b = a + 1;
clang-repl> b;
TranslationUnitDecl 0x7fcb0901d008 prev 0x7fcb08032df0 <>
`-TopLevelStmtDecl 0x7fcb0901d070 <input_line_3:1:1>








        `-DeclRefExpr 0x7fcb0901d0c8 <col:1> 'int' lvalue Var 0x7fcb08032e70 'b' 'int'
```

RuntimeInterfaceBuilder

Triggered by missing semicolon on trailing statements

```
clang-repl> int a = 1;
clang-repl> int b = a + 1;
clang-repl> b
TranslationUnitDecl 0x7fcb0901d008 prev 0x7fcb08032df0 <>
`-TopLevelStmtDecl 0x7fcb0901d070 <input_line_3:1:1>
  `-CallExpr 0x7fcb0901d3e8 <col:1> 'void'
    |-ImplicitCastExpr 0x7fcb0901d3d0 <> 'void (*)(void *, void *, void *, unsigned long long)'
    | `-DeclRefExpr 0x7fcb0901d378 <> 'void (void *, void *, void *, unsigned long long)'
                    lvalue Function 0x7fcb08031860 '__clang_Interpreter_SetValueNoAlloc'
    |-CStyleCastExpr 0x7fcb0901d220 <> 'void *' <IntegralToPointer>
    | `-IntegerLiteral 0x7fcb0901d1e8 <> 'unsigned long long' 140509973608656
    |-CStyleCastExpr 0x7fcb0901d280 <> 'void *' <IntegralToPointer>
    | `-IntegerLiteral 0x7fcb0901d248 <> 'unsigned long long' 140509973608720
    |-CStyleCastExpr 0x7fcb0901d2e0 <> 'void *' <IntegralToPointer>
    | `-IntegerLiteral 0x7fcb0901d2a8 <> 'unsigned long long' 140510006259984
    `-CStyleCastExpr 0x7fcb0901d350 <col:1> 'unsigned long long' <IntegralCast>
      `-ImplicitCastExpr 0x7fcb0901d320 <col:1> 'int' <LValueToRValue>
        `-DeclRefExpr 0x7fcb0901d0c8 <col:1> 'int' lvalue Var 0x7fcb08032e70 'b' 'int'
```

RuntimeInterfaceBuilder

Triggered by missing semicolon on trailing statements

```
clang-repl> int a = 1;
clang-repl> int b = a + 1;
clang-repl> b
TranslationUnitDecl 0x7fcb0901d008 prev 0x7fcb08032df0 <>
`-TopLevelStmtDecl 0x7fcb0901d070 <input_line_3:1:1>
  `-CallExpr 0x7fcb0901d3e8 <col:1> 'void'
    |-ImplicitCastExpr 0x7fcb0901d3d0 <> 'void (*)(void *, void *, void *, unsigned long long)'
    | `-DeclRefExpr 0x7fcb0901d378 <> 'void (void *, void *, void *, unsigned long long)'
                     lvalue Function 0x7fcb08031860 '__clang_Interpreter_SetValueNoAlloc'
    |-CStyleCastExpr 0x7fcb0901d220 <> 'void *' <IntegralToPointer>
    | `-IntegerLiteral 0x7fcb0901d1e8 <> 'unsigned long long' 140509973608656
    |-CStyleCastExpr 0x7fcb0901d280 <> 'void *' <IntegralToPointer>
    | `-IntegerLiteral 0x7fcb0901d248 <> 'unsigned long long' 140509973608720
    |-CStyleCastExpr 0x7fcb0901d2e0 <> 'void *' <IntegralToPointer>
    | `-IntegerLiteral 0x7fcb0901d2a8 <> 'unsigned long long' 140510006259984
    `-CStyleCastExpr 0x7fcb0901d350 <col:1> 'unsigned long long' <IntegralCast>
      `-ImplicitCastExpr 0x7fcb0901d320 <col:1> 'int' <LValueToRValue>
        `-DeclRefExpr 0x7fcb0901d0c8 <col:1> 'int' lvalue Var 0x7fcb08032e70 'b' 'int'
```

} "Magic"

Not soo different from printf()

```
clang-repl> extern "C" int printf(const char*, ...);
TranslationUnitDecl 0x7fcb0901d430 prev 0x7fcb0901d008 <>
`-LinkageSpecDecl 0x7fcb0901d4b8 <input_line_4:1:1, col:38> col:8 C
  `-FunctionDecl 0x7fcb0901d600 <col:12, col:38> col:16 printf 'int (const char *, ...)'
    |-ParmVarDecl 0x7fcb0901d520 <col:23, col:33> col:34 'const char *'
    |-BuiltinAttr 0x7fcb0901d6b0 <> Implicit 964
    `-FormatAttr 0x7fcb0901d708 <col:16> Implicit printf 1 2

clang-repl> printf("b=%d\n", b);
TranslationUnitDecl 0x7fcb0901d748 prev 0x7fcb0901d430 <>
`-TopLevelStmtDecl 0x7fcb0901d7f8 <input_line_5:1:1, col:19> col:1
  `-CallExpr 0x7fcb0901d958 <col:1, col:19> 'int'
    |-ImplicitCastExpr 0x7fcb0901d940 <col:1> 'int (*)(const char *, ...)' <FunctionToPointerDecay>
    | `-DeclRefExpr 0x7fcb0901d8f0 <col:1> 'int (const char *, ...)'
                    lvalue Function 0x7fcb0901d600 'printf'
    |-ImplicitCastExpr 0x7fcb0901d988 <col:8> 'const char *' <ArrayToPointerDecay>
    | `-StringLiteral 0x7fcb0901d8b0 <col:8> 'const char[6]' lvalue "b=%d\n"
    `-ImplicitCastExpr 0x7fcb0901d9a0 <col:18> 'int' <LValueToRValue>
      `-DeclRefExpr 0x7fcb0901d8d0 <col:18> 'int' lvalue Var 0x7fcb08032e70 'b' 'int'
```

Not soo different from printf()

Except that the type formatter is not given

```
clang-repl> extern "C" int printf(const char*, ...);
TranslationUnitDecl 0x7fcb0901d430 prev 0x7fcb0901d008 <<invalid sloc>> <invalid sloc>
`-LinkageSpecDecl 0x7fcb0901d4b8 <input_line_4:1:1, col:38> col:8 C
  `-FunctionDecl 0x7fcb0901d600 <col:12, col:38> col:16 printf 'int (const char *, ...)'
    |-ParmVarDecl 0x7fcb0901d520 <col:23, col:33> col:34 'const char *'
    |-BuiltinAttr 0x7fcb0901d6b0 <<invalid sloc>> Implicit 964
    `-FormatAttr 0x7fcb0901d708 <col:16> Implicit printf 1 2

clang-repl> printf("b=%d\n", b);
TranslationUnitDecl 0x7fcb0901d748 prev 0x7fcb0901d430 <<invalid sloc>> <invalid sloc>
`-TopLevelStmtDecl 0x7fcb0901d7f8 <input_line_5:1:1, col:19> col:1
  `-CallExpr 0x7fcb0901d958 <col:1, col:19> 'int'
    |-ImplicitCastExpr 0x7fcb0901d940 <col:1> 'int (*)(const char *, ...)' <FunctionToPointerDecay>
    | `-DeclRefExpr 0x7fcb0901d8f0 <col:1> 'int (const char *, ...)'
                   lvalue Function 0x7fcb0901d600 'printf'
    |-ImplicitCastExpr 0x7fcb0901d988 <col:8> 'const char *' <ArrayToPointerDecay>
    | `-StringLiteral 0x7fcb0901d8b0 <col:8> 'const char[6]' lvalue "b=%d\n"
    `-ImplicitCastExpr 0x7fcb0901d9a0 <col:18> 'int' <LValueToRValue>
      `-DeclRefExpr 0x7fcb0901d8d0 <col:18> 'int' lvalue Var 0x7fcb08032e70 'b' 'int'
```

RuntimeInterfaceBuilder injects context

https://github.com/llvm/llvm-project/blob/8ab3caf4d3acef29/clang/lib/Interpreter/Interpreter.cpp#L680

```
const char *const Runtimes = R"(
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, void*);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, float);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, double);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, long double);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, unsigned long long);
    ...                                     ^Interp  ^OutVal  ^Type
)";
```

```
|-CStyleCastExpr 0x7fcb0901d220 <> 'void *' <IntegralToPointer>
| `-IntegerLiteral 0x7fcb0901d1e8 <> 'unsigned long long' 140509973608656
|-CStyleCastExpr 0x7fcb0901d280 <> 'void *' <IntegralToPointer>
| `-IntegerLiteral 0x7fcb0901d248 <> 'unsigned long long' 140509973608720
|-CStyleCastExpr 0x7fcb0901d2e0 <> 'void *' <IntegralToPointer>
| `-IntegerLiteral 0x7fcb0901d2a8 <> 'unsigned long long' 140510006259984
```

RuntimeInterfaceBuilder injects context

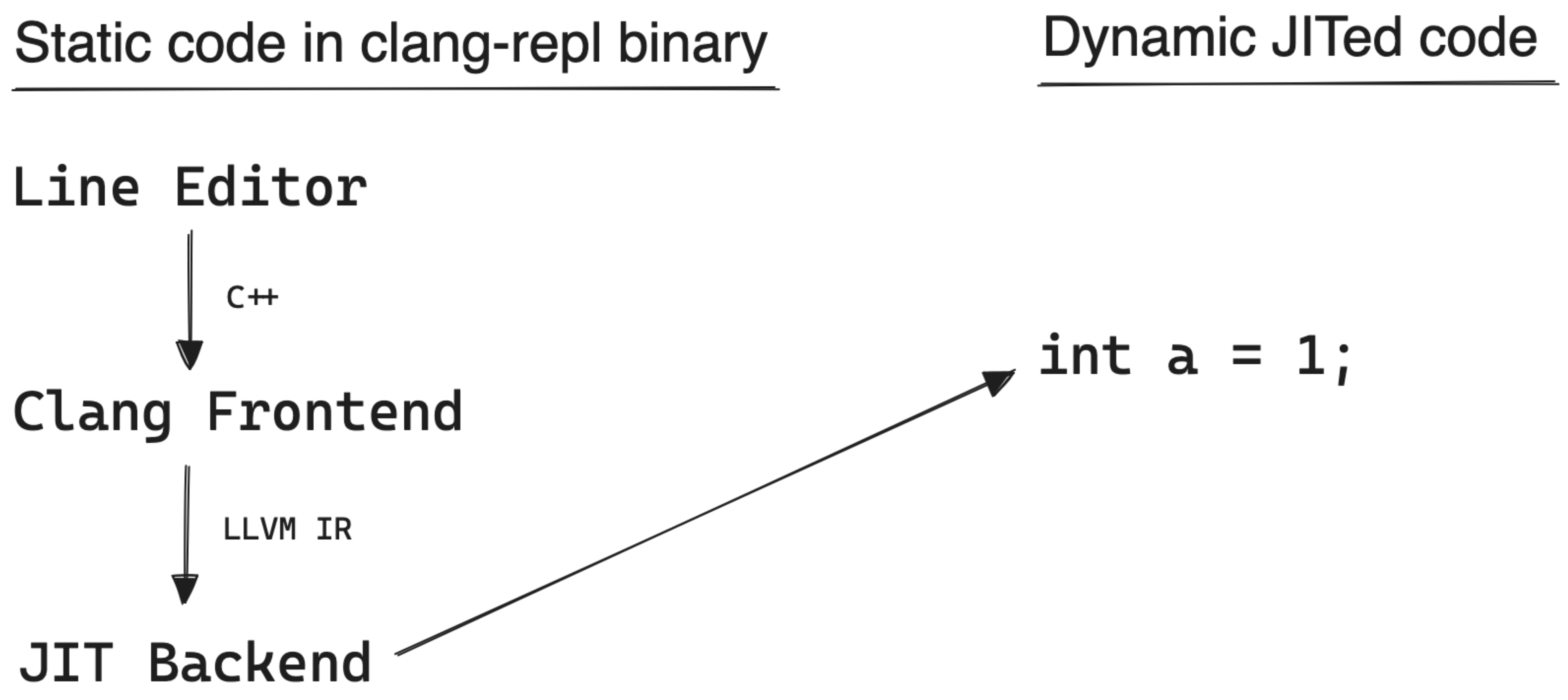
https://github.com/llvm/llvm-project/blob/8ab3caf4d3acef29/clang/lib/Interpreter/Interpreter.cpp#L680

```
const char *const Runtimes = R"(
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, void*);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, float);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, double);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, long double);
    void __clang_Interpreter_SetValueNoAlloc(void*,   void*,   void*, unsigned long long);
    ...                        Mangling determines matching overload! ^^^^^^^^^^^^^^^^^^
)";
```

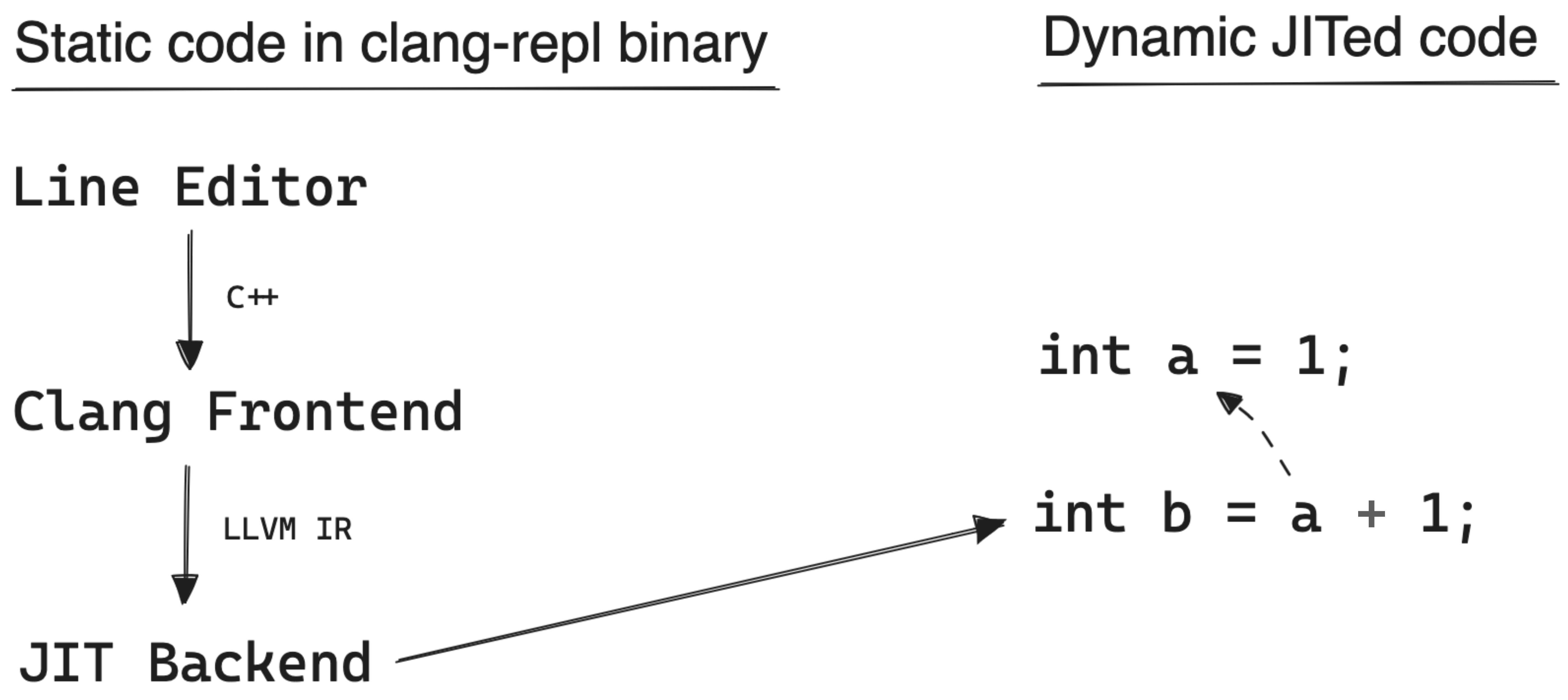
```
|-CStyleCastExpr 0x7fcb0901d220 <> 'void *' <IntegralToPointer>
| `-IntegerLiteral 0x7fcb0901d1e8 <> 'unsigned long long' 140509973608656
|-CStyleCastExpr 0x7fcb0901d280 <> 'void *' <IntegralToPointer>
| `-IntegerLiteral 0x7fcb0901d248 <> 'unsigned long long' 140509973608720
|-CStyleCastExpr 0x7fcb0901d2e0 <> 'void *' <IntegralToPointer>
| `-IntegerLiteral 0x7fcb0901d2a8 <> 'unsigned long long' 140510006259984
```

Not hacky enough? Stay tuned!

Value printing needs a runtime



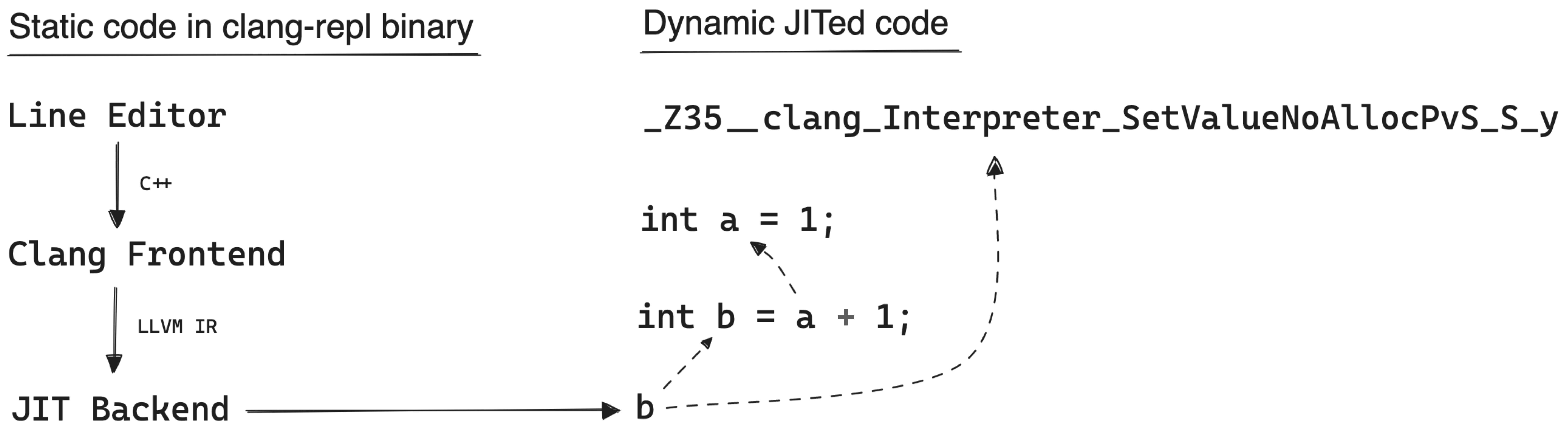
Value printing needs a runtime



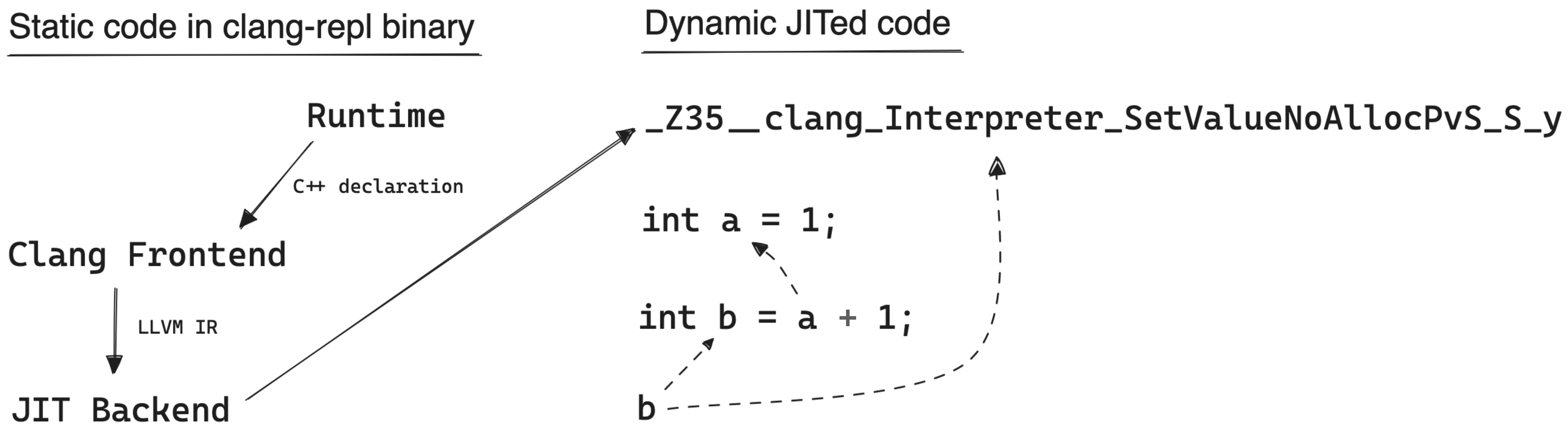
```
int a = 1;

int b = a + 1;
```

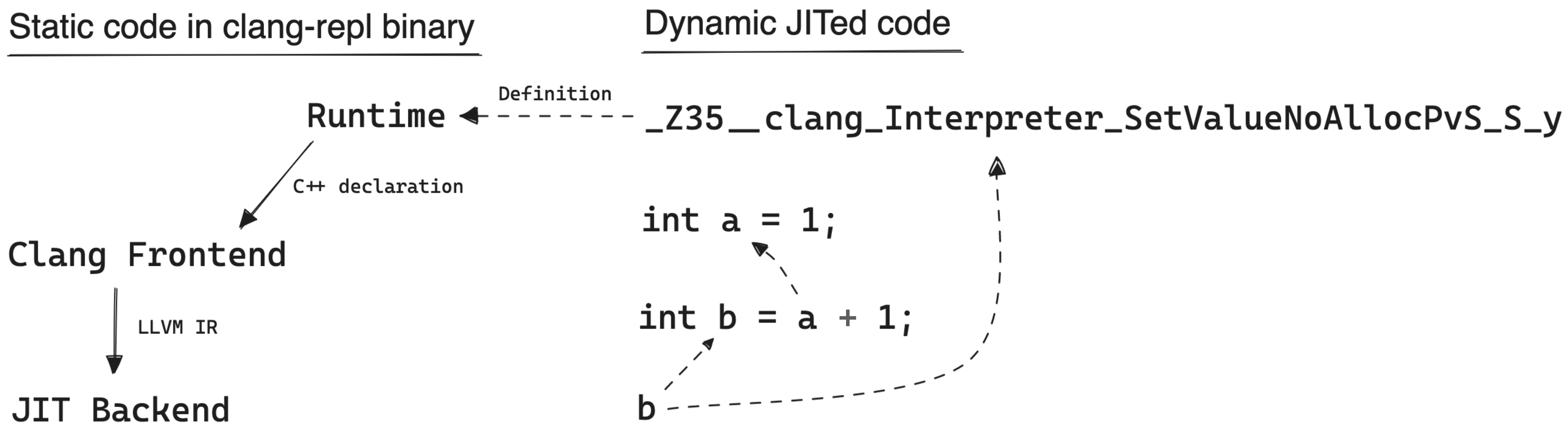
Value printing needs a runtime



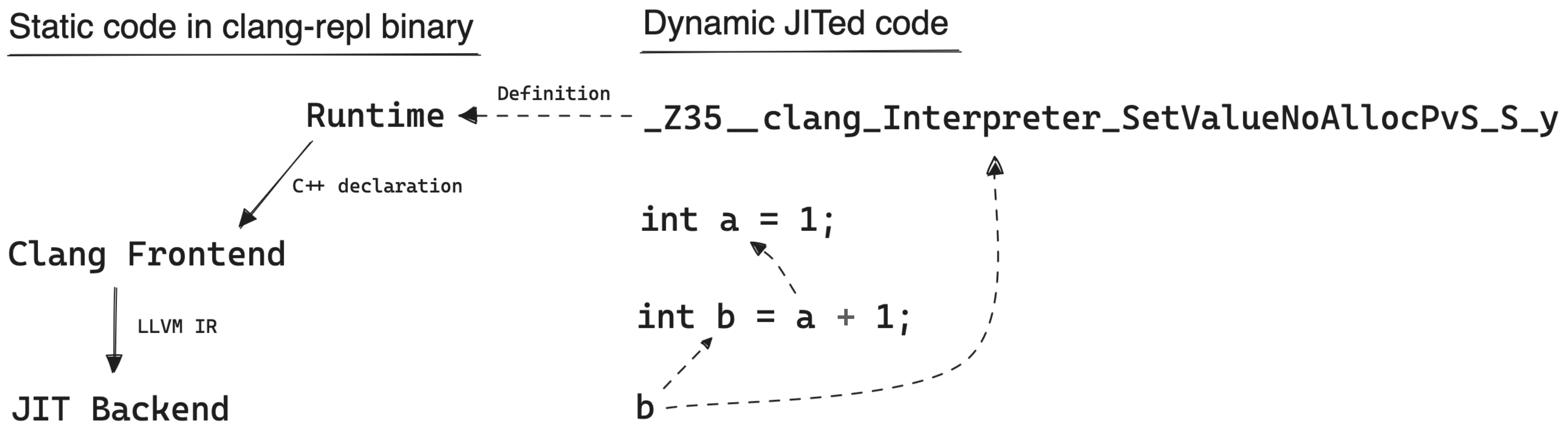
Value printing needs a runtime



Value printing needs a runtime

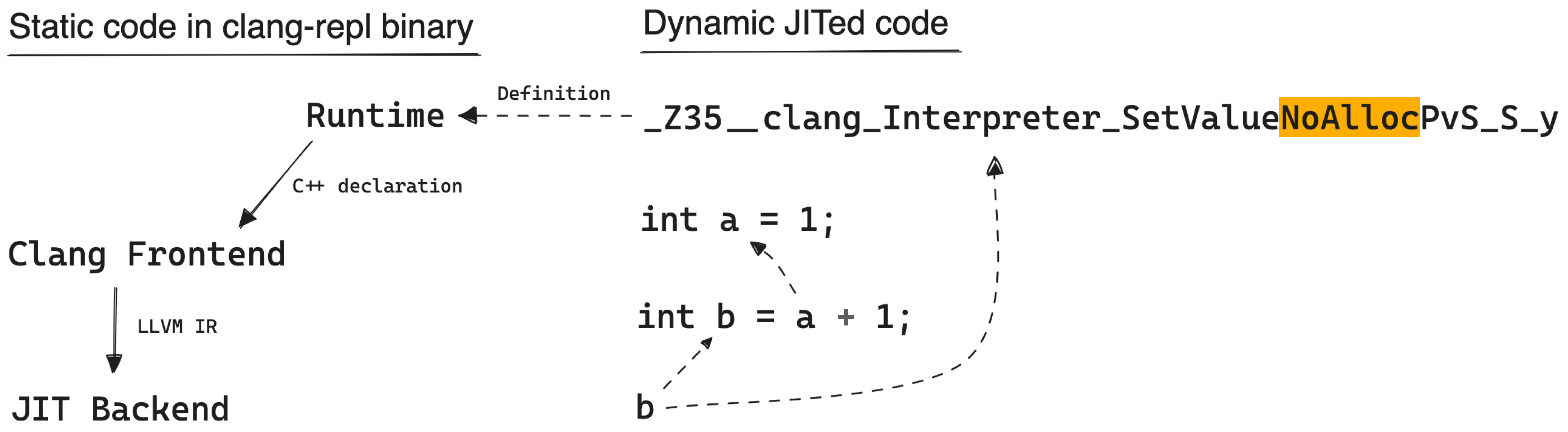


Value printing needs a runtime



🤔 Requires: shared memory and matching CPU arch!

All good as long as we pass primitive L-values!



But then there is code like this

```
➜ build/bin/clang-repl
clang-repl> int *x = new int();
clang-repl> template <class T> struct GuardX { T *x; GuardX(T *x) : x(x) {}; ~GuardX(); };

clang-repl> extern "C" int printf(const char *, ...);
clang-repl> template <class T> GuardX<T>::~GuardX() { delete x; printf("Running dtor\n"); }

clang-repl> (GuardX<int>(x))
```

But then there is code like this

```
➜ build/bin/clang-repl
clang-repl> int *x = new int();
clang-repl> template <class T> struct GuardX { T *x; GuardX(T *x) : x(x) {}; ~GuardX(); };

clang-repl> extern "C" int printf(const char *, ...);
clang-repl> template <class T> GuardX<T>::~GuardX() { delete x; printf("Running dtor\n"); }

clang-repl> (GuardX<int>(x))
Not implement yet.
Running dtor
```

But then there is code like this

```
➜ build/bin/clang-repl
clang-repl> int *x = new int();
clang-repl> template <class T> struct GuardX { T *x; GuardX(T *x) : x(x) {}; ~GuardX(); };

clang-repl> extern "C" int printf(const char *, ...);
clang-repl> template <class T> GuardX<T>::~GuardX() { delete x; printf("Running dtor\n"); }

clang-repl> (GuardX<int>(x))
Not implement yet.
Running dtor
```

👏👏👏

Temporary R-value struct

```
clang-repl> (GuardX(x));
TranslationUnitDecl 0x560b3b604db0 prev 0x560b3b5cfa88 <<invalid sloc>> <invalid sloc>
`-TopLevelStmtDecl 0x560b3b6065e0 <input_line_2:1:1, col:11>
  `-ExprWithCleanups 0x560b3b60a298 <col:1, col:11> 'GuardX'

    `-ParenExpr 0x560b3b608148 <col:1, col:11> 'GuardX'
      `-CXXFunctionalCastExpr 0x560b3b608120 <col:2, col:10> 'GuardX' functional cast to GuardX
        `-CXXBindTemporaryExpr 0x560b3b608100 <col:2, col:10> 'GuardX' (CXXTemporary 0x560b3b608100)
          `-CXXConstructExpr 0x560b3b6080c8 <col:2, col:10> 'GuardX' 'void (int *&)'
            `-DeclRefExpr 0x560b3b6066b0 <col:9> 'int *' lvalue Var 0x560b3b5cfb38 'x' 'int *'
```

RuntimeInterfaceBuilder transformation

```
clang-repl> (GuardX(x))
TranslationUnitDecl 0x560b3b604db0 prev 0x560b3b5cfa88 <<invalid sloc>> <invalid sloc>
`-TopLevelStmtDecl 0x560b3b6065e0 <input_line_2:1:1, col:11>
  `-CXXNewExpr 0x560b3b6087a8 <col:1, col:11> 'GuardX *' global Function 0x560b3b5cbf10 'operator new'
                                              'void *(unsigned long, void *, __clang_Interpreter_NewTag) noexcept'
    |-ParenExpr 0x560b3b608148 <col:1, col:11> 'GuardX'
    | `-CXXFunctionalCastExpr 0x560b3b608120 <col:2, col:10> 'GuardX' functional cast to GuardX
    |   `-CXXBindTemporaryExpr 0x560b3b608100 <col:2, col:10> 'GuardX' (CXXTemporary 0x560b3b608100)
    |     `-CXXConstructExpr 0x560b3b6080c8 <col:2, col:10> 'GuardX' 'void (int *&)'
    |       `-DeclRefExpr 0x560b3b6066b0 <col:9> 'int *' lvalue Var 0x560b3b5cfb38 'x' 'int *'
    |-CallExpr 0x560b3b6083e8 <col:11> 'void *'
    | |-ImplicitCastExpr 0x560b3b6083d0 <<invalid sloc>> 'void *(*)(void *, void *, void *)' <FunctionToPointerDecay>
    | | `-DeclRefExpr 0x560b3b6081f0 <<invalid sloc>> 'void *(void *, void *, void *)'
                                        lvalue Function 0x560b3b5a9b18 '__clang_Interpreter_SetValueWithAlloc'
    | |-CStyleCastExpr 0x560b3b6082b8 <<invalid sloc>> 'void *' <IntegralToPointer>
    | | `-IntegerLiteral 0x560b3b608280 <<invalid sloc>> 'unsigned long long' 94606239928896
    | |-CStyleCastExpr 0x560b3b608318 <<invalid sloc>> 'void *' <IntegralToPointer>
    | | `-IntegerLiteral 0x560b3b6082e0 <<invalid sloc>> 'unsigned long long' 94606239928960
    | `-CStyleCastExpr 0x560b3b608378 <<invalid sloc>> 'void *' <IntegralToPointer>
    |   `-IntegerLiteral 0x560b3b608340 <<invalid sloc>> 'unsigned long long' 94606240805152
    `-CXXConstructExpr 0x560b3b608688 <<invalid sloc>> '__clang_Interpreter_NewTag' 'void (const __clang_Interpreter_NewTag &) noexcept'
      `-DeclRefExpr 0x560b3b608260 <<invalid sloc>> 'struct __clang_Interpreter_NewTag':'__clang_Interpreter_NewTag'
                                      lvalue Var 0x560b3b5c9658 '__ci_newtag'
```

Clang did not emit the destructor yet

... and this is an expression 😬

```
clang-repl> (GuardX(x))
TranslationUnitDecl 0x560b3b604db0 prev 0x560b3b5cfa88 <>
`-TopLevelStmtDecl 0x560b3b6065e0 <input_line_2:1:1, col:11>
  `-CXXNewExpr 0x560b3b6087a8 <col:1, col:11> 'GuardX *' global Function 0x560b3b5cbf10 'operator new'
                                              'void *(unsigned long, void *, __clang_Interpreter_NewTag) noexcept'
    |-ParenExpr 0x560b3b608148 <col:1, col:11> 'GuardX'
    | `-CXXFunctionalCastExpr 0x560b3b608120 <col:2, col:10> 'GuardX' functional cast to GuardX
    |   `-CXXBindTemporaryExpr 0x560b3b608100 <col:2, col:10> 'GuardX' (CXXTemporary 0x560b3b608100)
    |     `-CXXConstructExpr 0x560b3b6080c8 <col:2, col:10> 'GuardX' 'void (int *&)'
    |       `-DeclRefExpr 0x560b3b6066b0 <col:9> 'int *' lvalue Var 0x560b3b5cfb38 'x' 'int *'
    |-CallExpr 0x560b3b6083e8 <col:11> 'void *'
    | |-ImplicitCastExpr 0x560b3b6083d0 <> 'void *(*)(void *, void *, void *)' <FunctionToPointerDecay>
    | | `-DeclRefExpr 0x560b3b6081f0 <> 'void *(void *, void *, void *)'
                                    lvalue Function 0x560b3b5a9b18 '__clang_Interpreter_SetValueWithAlloc'
    | |-CStyleCastExpr 0x560b3b6082b8 <> 'void *' <IntegralToPointer>
    | | `-IntegerLiteral 0x560b3b608280 <> 'unsigned long long' 94606239928896
    | |-CStyleCastExpr 0x560b3b608318 <> 'void *' <IntegralToPointer>
    | | `-IntegerLiteral 0x560b3b6082e0 <> 'unsigned long long' 94606239928960
    | `-CStyleCastExpr 0x560b3b608378 <> 'void *' <IntegralToPointer>
    |   `-IntegerLiteral 0x560b3b608340 <> 'unsigned long long' 94606240805152
    `-CXXConstructExpr 0x560b3b608688 <> '__clang_Interpreter_NewTag' 'void (const __clang_Interpreter_NewTag &) noexcept'
      `-DeclRefExpr 0x560b3b608260 <> 'struct __clang_Interpreter_NewTag':'__clang_Interpreter_NewTag'
                                   lvalue Var 0x560b3b5c9658 '__ci_newtag'
```

A backdoor to the rescue

https://github.com/llvm/llvm-project/blob/8ab3caf4d3acef29/clang/lib/Interpreter/Interpreter.cpp#L726-L732

```
// Force CodeGen to emit destructor
if (auto *RD = Ty->getAsCXXRecordDecl()) {
  auto *Dtor = Sema.LookupDestructor(RD);
  Dtor->addAttr(UsedAttr::CreateImplicit(Ctx));
  Interp.getCompilerInstance()->getASTConsumer().HandleTopLevelDecl(DeclGroupRef(Dtor));
}
```

A backdoor to the rescue

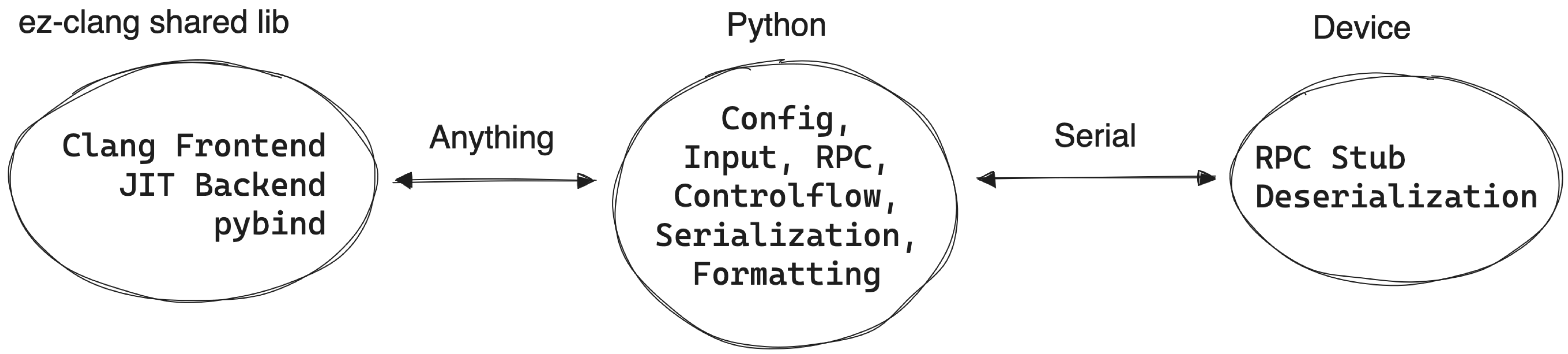
https://github.com/llvm/llvm-project/blob/8ab3caf4d3acef29/clang/lib/Interpreter/Interpreter.cpp#L726-L732

```
// Force CodeGen to emit destructor
if (auto *RD = Ty->getAsCXXRecordDecl()) {
  auto *Dtor = Sema.LookupDestructor(RD);
  Dtor->addAttr(UsedAttr::CreateImplicit(Ctx));
  Interp.getCompilerInstance()->getASTConsumer().HandleTopLevelDecl(DeclGroupRef(Dtor));
}
```

... always keep a backdoor open 😂

Why do I care?

ez-clang needs a runtime too


ez-clang shared lib
Clang Frontend
JIT Backend
pybind
Anything
Python
Config,
Input, RPC,
Controlflow,
Serialization,
Formatting
Serial
Device
RPC Stub
Deserialization