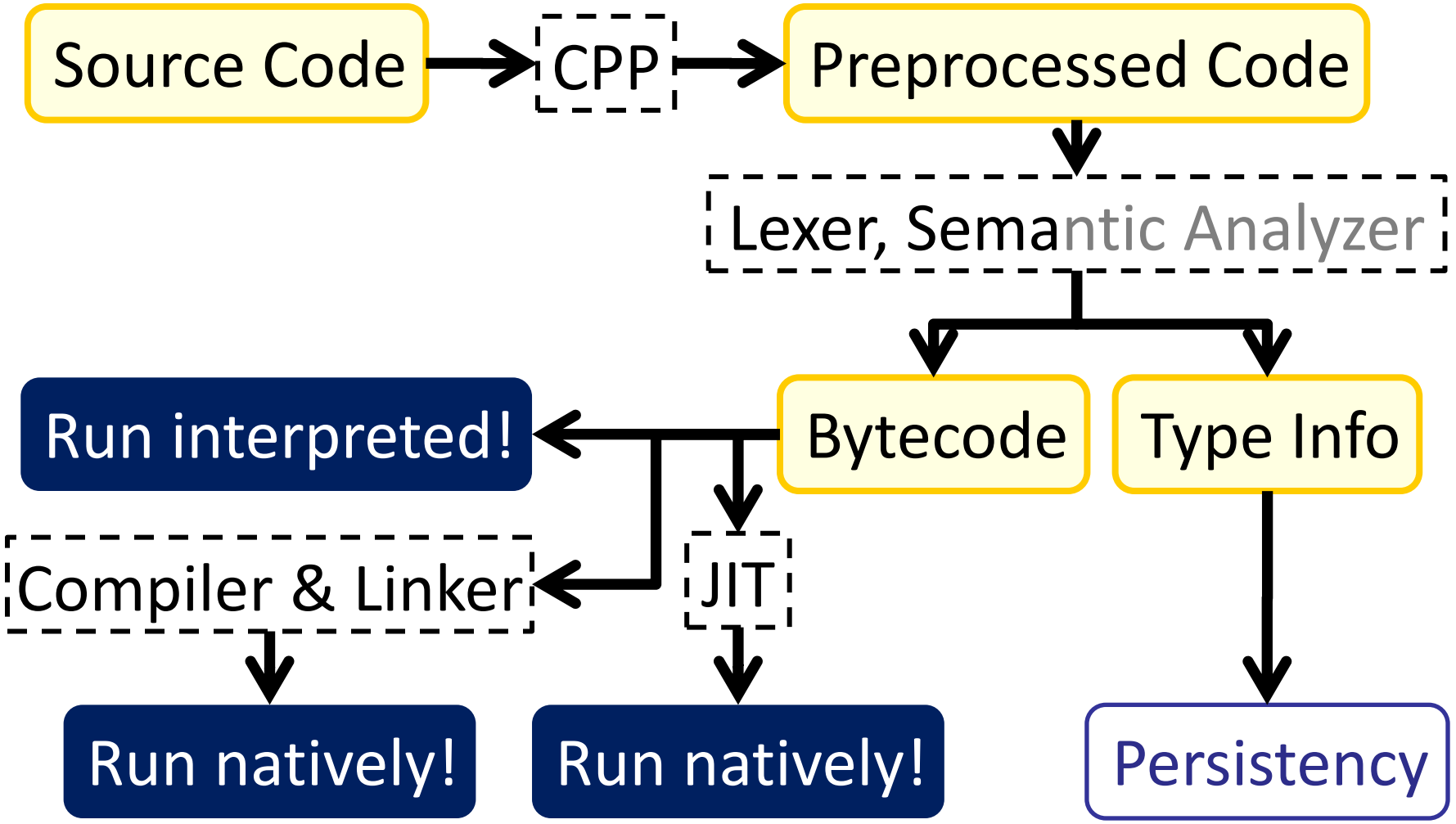
LLVM as Interpreter and Dictionary Generator

Axel Naumann

LCG-AA Meeting

2009-01-21

Life of Code


Source Code
CPP
Preprocessed Code
Lexer, Semantic Analyzer
Bytecode
Type Info
Run interpreted!
Compiler & Linker
JIT
Run natively!
Run natively!
Persistency

Life of Code With Libraries

```
void f() {
   myFunc();
}
```

JIT

see demo!

compile, optimize, link on-the-fly!

interpreted

function stubs

native

OS: linker, ABI, dynamic loader

Library Code
int myFunc()

Scope

Plan: replace CINT by LLVM-based interpreter

Affects

- reading code: CINT's parser
- generating type info: CINT, GCCXML + genreflex
- interpreting: CINT
- compiling: ACLiC

LLVM-Based Interpreter

We want to use LLVM to

- parse C++
- extract type info through C++ API
- interpret adapting existing LLVM bytecode interpreter **`lli`**
- JIT-compile (for speed, to interface libraries)

Build CINT replacement **`cling`** on top by adding:

dictionaries, auto-load, ACLiC,…

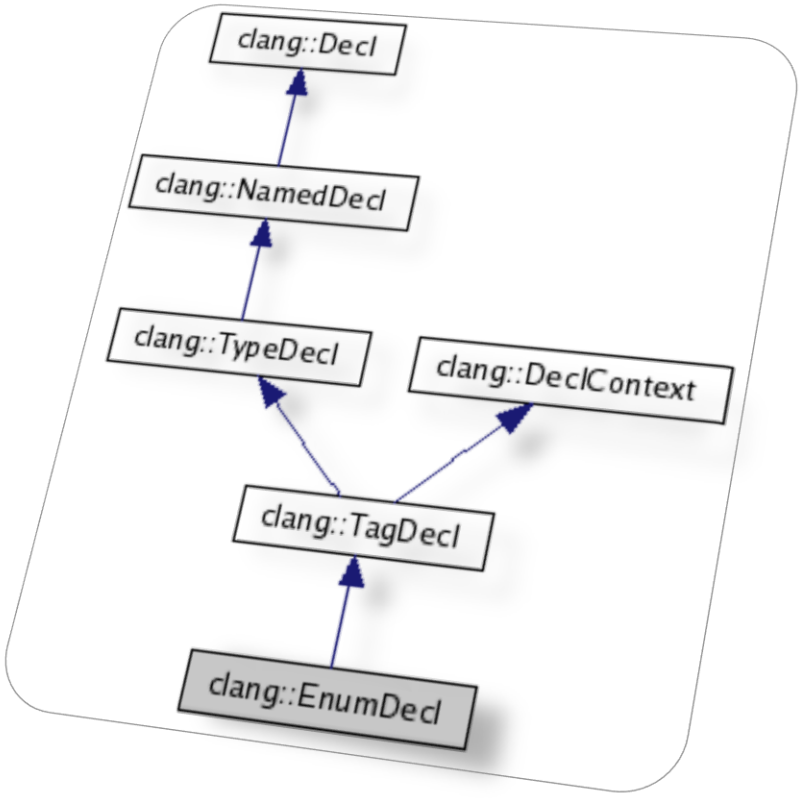
Parser: clang

CINT issues: templates, operators, overloading, lookup, intertwined with execution, file handling, STL and system headers.

clang: C++ API

Targets C++0x

Analyze, extract, generate, manipulate code!



clang & cling

clang's dev plan: "production in 2 years"

- Consistent with cling, will follow development
- Can contribute and influence

clang's dev focus: fast parsing, platform independent, clean, dynamic, good error messages

see demo!

- Consistent with cling

cling & clang: Hot Topics

Dictionaries: persistent reflection, library calls

I/O

Auto-`#include`, auto-dlopen

Auto-declarations

Unloading of code / libraries

Dynamic binding

Multithreading

Prompt

PyROOT

Dictionaries: Reflection

Precompiled header (PCH) support expected 2009:

- dumps headers (AST) to binary file
- reading back corresponds to reading headers
- can selectively read PCHs unique!

Use PCH to store type info! Can use artificial `__attributes__` to store I/O directives etc.

Missing: selective writing of PCH

Dictionaries: Library Calls

Convert string `"myfunc(12)"` into library call

Use just-in-time (JIT) compiler! see demo!

LLVM JIT available on X86, PowerPC, Alpha with Linux (32 / 64 bit), MacOS X, Win32.

If not sufficient: implement JIT or use stubs

ROOT I/O

Challenge:

I/O needs to interface with reflection data

clang does not provide a “safe” query interface but an optimized AST

Use Reflex as API to clang AST

Needs glue between Reflex and ROOT (like Cintex), to be part of ROOT

Auto-#include, Auto-dlopen

Challenge: `TH1* f() {return new TH1F(…);}`

What is `TH1`? `TH1F`? Where is `TH1F::TH1F(…)`?

Approach: smart error handling

- clang's parser notifies cling: unknown name `"MyClass"`
- cling holds map: type → library (aka rootmap)
- load library, insert "header" (AST) from PCH

Auto-Declarations

Challenge: `h = new TH1F(…);`

h undeclared.

Approach: again smart error handling

1. clang's parser notifies cling: unknown name "h"
2. check that expression is an assignment
3. using C++ 0x `auto`, transform to

```
auto h = new TH1F(…);
```

Unloading of Code / Libraries

Not foreseen by LLVM – compilers don't care!

clang's data structures are highly optimized, no room for unload facility

Need to implement in cling

Updating clang's code view (headers / AST) is manageable

Use Reflex for supporting unloaded names

Need interpreter state with dependency graphs / check pointing. Not trivial, major development

Dynamic Binding

Example: `f.root` contains `myhist`.

```
new TFile("f.root"); myhist->Draw();
```

LLVM: `myhist` unknown! Catch and transform to

```
new TFile("f.root");
interpret("myhist->Draw();");
```

`interpret(…)` makes valid C++

delayed `interpret(…)` will find `myhist` and use it!

Need to make LLVM re-entrant

Multithreading

In this context: one interpreter per thread,

shared reflection data

Issues:

1. LLVM's reflection data not thread safe → Reflex!
2. JIT … not thread safe → make it thread safe or share instance! clang's modular C++ code with few statics helps!

NOT comparable to making CINT thread safe:

Plenty of globals, little modularity,…

PyROOT

"llvm-py provides Python bindings for LLVM. It's goal is to expose enough of LLVM APIs to implement a compiler backend or a VM in pure Python."[1]

No Swig or anything – it's "direct"!

PyROOT could request any C++ types, on-the-fly template instantiation, call into any library given its header, and C++ JIT!

1) http://mdevan.nfshost.com/llvm-py/ [original spelling]

Evaluation Dev Needs

Compared to situation with CINT:

- ***much*** cleaner code base
- production-oriented background
- patching C++ 0x into code from 1991 is no fun
- re-use good code instead of eternally continuing to fix our own limited code

Or to summarize: exciting and promising!

Estimate Dev Needs

Prototype target: two years for ROOT, three for LHC

Estimate approx 2 + ½ developers:

- ½ person @ FNAL:
  ROOT / Reflex; I/O related issues; unloading
- Fellow and me: interpreter state, thread safety / reentrance, dictionaries / PCH / Reflex

Need more details before commitment to dates:
tests of open issues; another discussion with LLVM developers

Summary

Compile a list of ingredients for a C++ interpreter and code parser for reflection extraction:

clang & LLVM is an incredibly good match

Industrial strength code base

Open and positive community

First evaluation and discussion showed matching development goals and no show stoppers

cling is the better GCCXML/genreflex and CINT!

Demo!

Parsers: LLVM-GCC

GCC afraid of external access, mostly due to ©

Richard Stallman, 2000: *"the GNU Project will in all probability not install such changes."*[1]

Plugins project started 2008 outside GCC.

"[…] licensing & legal issues related to plugins are been worked upon."[2] Not a realistic option for us.

E.g. GCCXML: GCC patches,

"gccxml is just dumping what gcc's parser produces."[3]

1) http://gcc.gnu.org/ml/gcc/2000-01/msg00572.html
2) http://gcc.gnu.org/wiki/GCC_Plugins
3) http://www.mail-archive.com/gccxml@gccxml.org/msg00326.html

Prompt

Far from trivial! Example:

```
[cling] float f = 0.2;
[cling] float g;
[cling] g = sin(f);
[cling] g
```

Interspersed declarations, expressions, calls

Challenge: map "growing code" into compiler / linker system

Prompt: Probable Solution

Prompt-namespace:

grows with prompt's declarations:

```
float f;
float g;
```

Prompt-functions:

each line is a function:

```
void f0(){f=0.2;}
void f1(){g=sin(f);}
```

Still some open issues:

expression dump: `[cling] g`

unloading might need interpreter state

Summary Dev Needs

Major development tasks:

- Interpreter state
- Support for unloading
- Making LLVM reentrant, thread safe
- Enhancing PCH to store I/O directives, selective writing of PCH
- Reflex will stay, but needs link to ROOT ("embedded Cintex")