New LLVM C Front-end

Steve Naroff
snaroff@apple.com

Rallying Cry

We, the Apple and LLVM communities, need a great front-end for the next decade!

Why? (motivation for a new front-end)

- GCC's front-end is difficult to work with
  - Learning curve too steep for many developers
  - Implementation and politics limit innovation
- GCC doesn't service the diverse needs of Apple's IDE
  - True in 1990, still true today!
  - "DevKit" was developed to fill the void
- GCC + DevKit results in a "less than great" developer experience
  - Why doesn't my code index properly if it compiles with GCC?
  - Inherently error prone (two preprocessors, lexers, parsers, etc.)

Goals

- Unified parser for C-based languages
  - Language conformance (C99, ObjC, C++)
  - GCC compatibility
  - Expressive diagnostics
- Library based architecture with finely crafted C++ API's
  - Useable and extensible by mere mortals
- Multipurpose
  - Indexing, static analysis, code generation
  - Source to source tools, refactoring
- High performance
  - Low memory footprint, lazy evaluation

Non Goals

- Obsoleting GCC (or llvm-gcc)
  - not pragmatic, we respect GCC’s ubiquity
  - our goals are fundamentally different than GCC
- Support for non-C based languages
  - no plans for Java, Ada, FORTRAN

High Level Architecture

AST->LLVM	FIXME:-)
AST	- Performs semantic analysis, type checking - Builds Abstract Syntax Trees for valid input
Parse	- Hand built recursive descent (C99 now, ObjC/C++ later) - DevKit-inspired "actions" for each production
Lex	- Lexing, preprocessing, and pragma handling (C/C++/ObjC) - Identifier hash table, tokens, macros, literals
Basic	- Source Manager (locations, ranges, buffers, file caching) - Diagnostics, target description, language dialect control
LLVM Support/System	- Casting, CommandLine, MemoryBuffer, MappedFile - FoldingSet, SmallVector, SmallSet, StringMap, SmallString, APInt

Introducing “clang”...

- A simple driver, with (some) GCC compatible options:

```
$ clang implicit-def.c -std=c89
implicit-def.c:6:10: warning: implicit declaration of function 'X'
  return X();
         ^
```

- Driver currently supports multiple -arch options:

```
$ clang -arch ppc -arch linux -fsyntax-only portability.c
portability.c:4:11: note: sizeof(wchar_t) varies between targets,
source is not 'portable'
void *X = L"foo";
          ^
```

- Performance analysis options (-Eonly, -parse-noop, -stats)

```
$ time clang -parse-noop INPUTS/carbon_h.c
real  0m0.204s
user  0m0.138s
sys   0m0.047s
```

GCC diagnostics

```
// test.c
struct A { int X; } someA;
int func(int);

int test1(int intArg) {
 *(someA.X);
 return intArg + func(intArg ? ((someA.X + 40) + someA) / 42 + someA.X : someA.X);
}
```

```
% cc -c test.c

test.c: In function ‘test1’:
test.c:7: error: invalid type argument of ‘unary *’
test.c:8: error: invalid operands to binary +
```

clang “expressive” diagnostics

```
% clang test.c

test.c:7:2: error: indirection requires a pointer operand ('int' invalid)
 *(someA.X);
 ^~~~~~~~~
test.c:8:48: error: invalid operands to binary expression ('int' and 'struct A')
 return intArg + func(intArg ? ((someA.X + 40) + someA) / 42 + someA.X : someA.X);
                                ~~~~~~~~~~~~~~ ^ ~~~~~
```

```
% cc -c test.c

test.c: In function ‘test1’:
test.c:7: error: invalid type argument of ‘unary *’
test.c:8: error: invalid operands to binary +
```

Lazy, fine-grained token evaluation

```
unsigned DiagnosticPrinter::GetTokenLength(SourceLocation Loc)
{
  // Extract relevant info from “Loc”.
  SourceLocation lLoc = SourceMgr.getLogicalLoc(Loc);
  const char *StrData = SourceMgr.getCharacterData(lLoc);
  unsigned FileID = Loc.getFileID();

  // Create a lexer starting at the beginning of this token.
  Lexer TheLexer(SourceMgr.getBuffer(FileID), FileID,
                 *ThePreprocessor,  StrData);

  LexerToken TheTok;
  TheLexer.LexRawToken(TheTok);

  return TheTok.getLength();
}
```

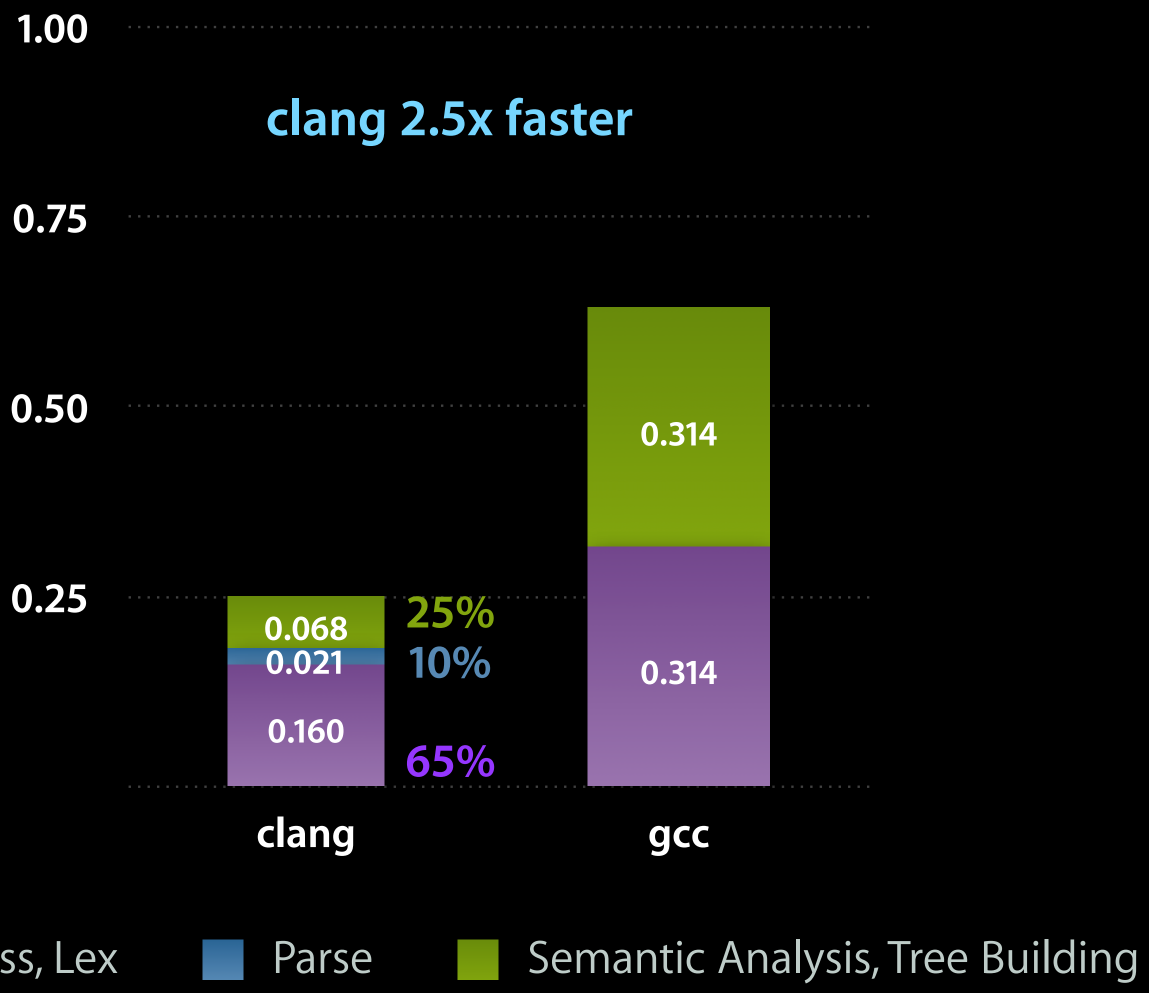
clang performance compiling “carbon.h”

- What is “carbon.h”?
  - Defines the public C API to Mac OS X system services

- How big is it?
  - 558 files
  - 12.3 megabytes!
  - 10,000 function declarations
  - 2000 structure definitions, 8000 fields
  - 3000 enum definitions, 20000 enum constants
  - 5000 typedefs
  - 2000 file scoped variables
  - 6000 macros

Time

2.0 Ghz Intel Core Duo

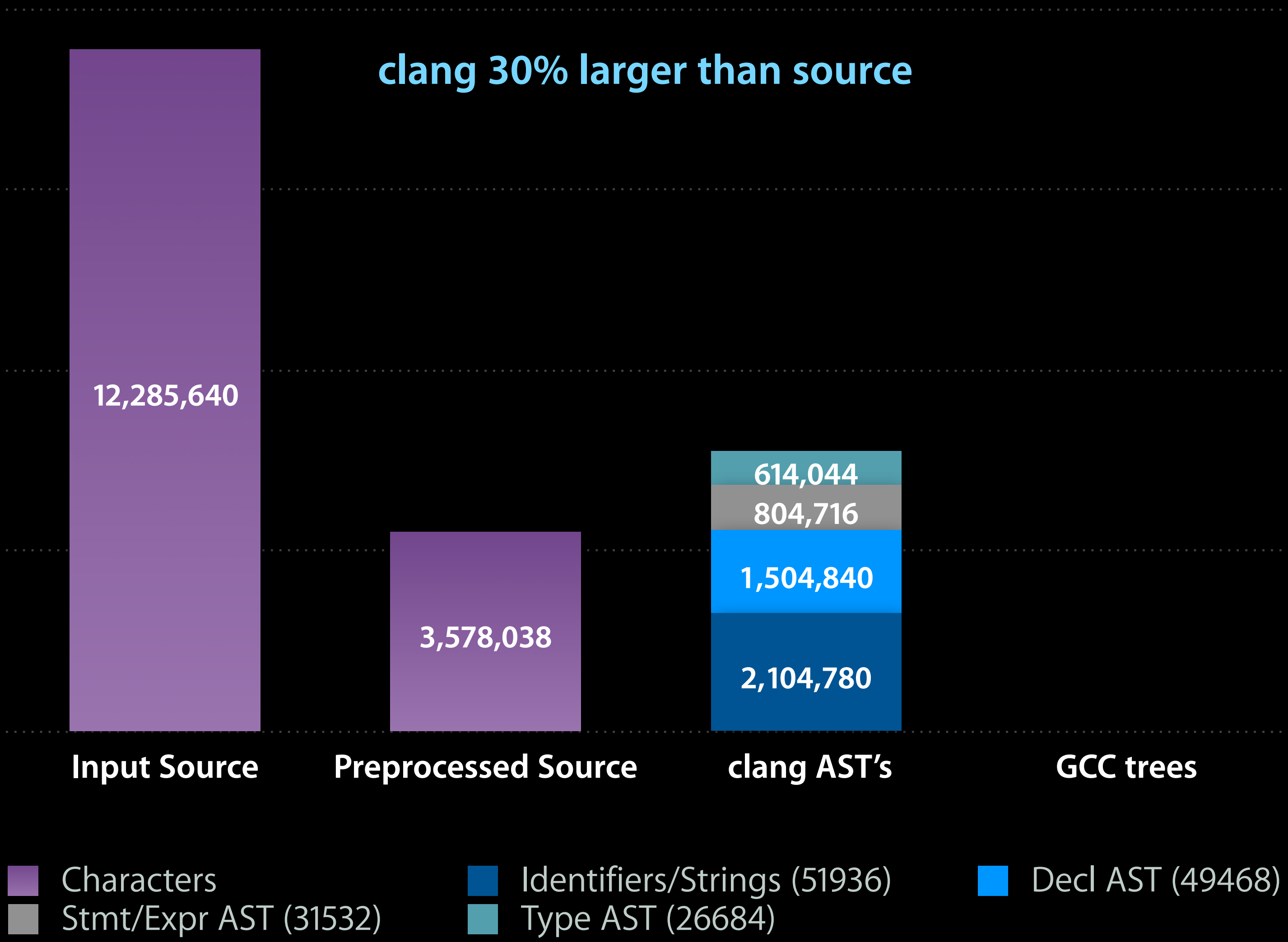


Changing the rules (2.5x good, 10x better:-)

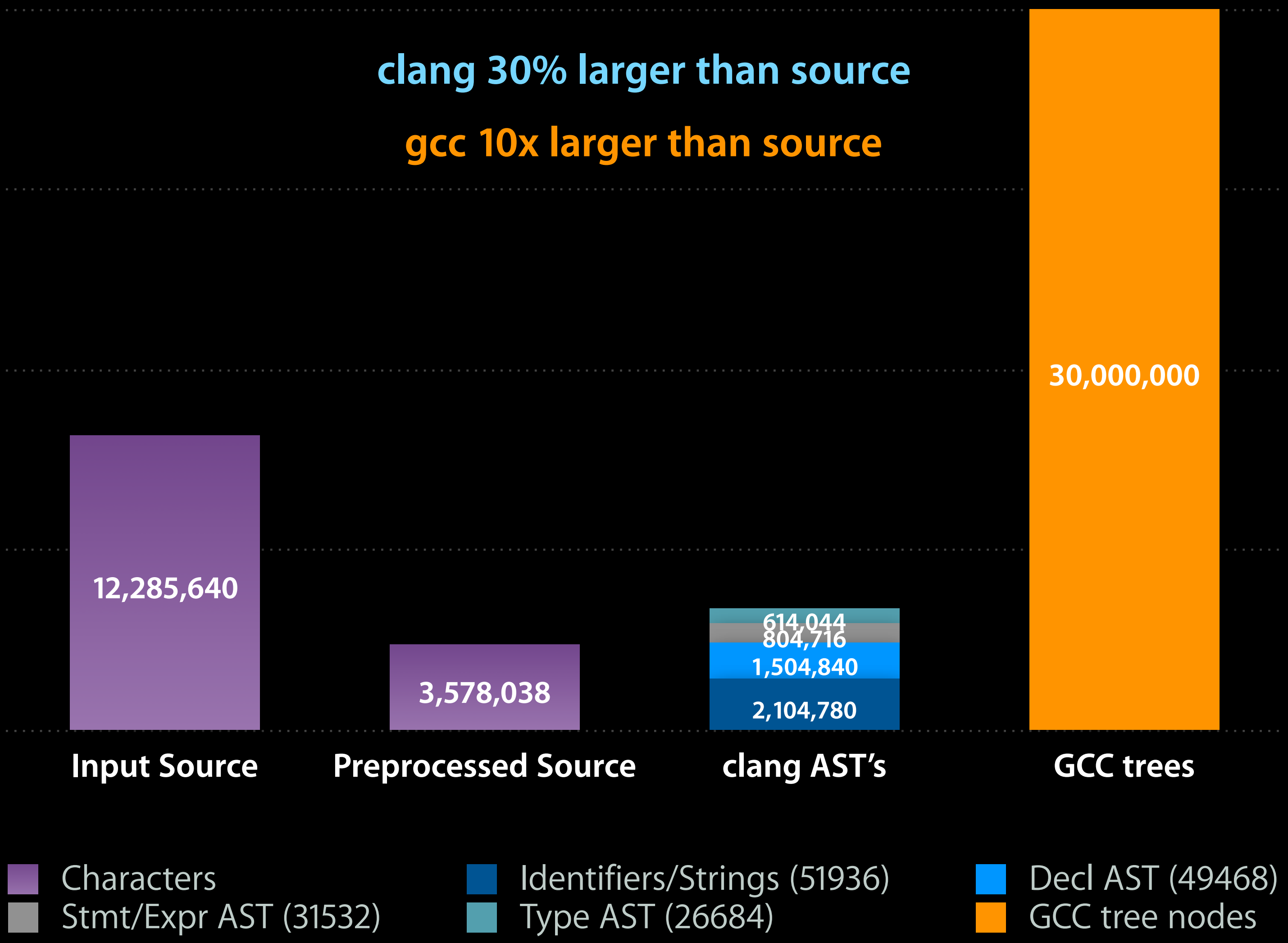
- Whole program ASTs (space efficient and easy to access)
  - Lazily [de]serialize them to/from disk or store in a server/IDE
  - ASTs are built as a side-effect of compiling

- ASTs are the intermediate form to enable many clients:
  - Symbol index, cross reference
  - Source code refactoring
  - Precompiled headers
  - Codegen to LLVM, in process (JIT for -O0?)
  - Debugger: use AST for types, decls, etc, reducing DWARF size
  - Syntax-aware highlighting (not regex'ing source)
  - Who knows what else? Clearly many possibilities...

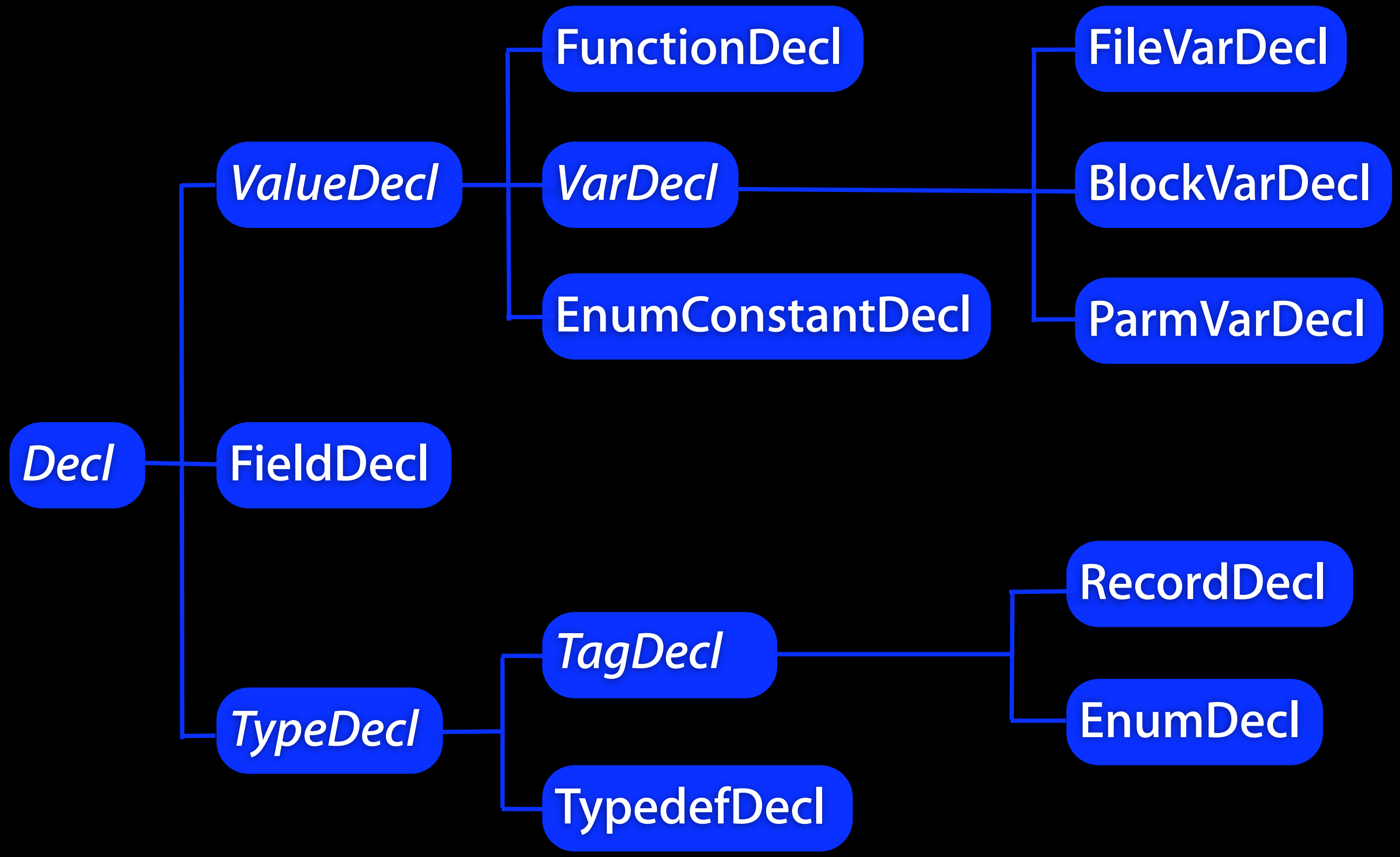
Space


clang 30% larger than source
12,285,640
3,578,038
614,044
804,716
1,504,840
2,104,780
Input Source
Preprocessed Source
clang AST's
GCC trees
Characters
Stmt/Expr AST (31532)
Identifiers/Strings (51936)
Type AST (26684)
Decl AST (49468)

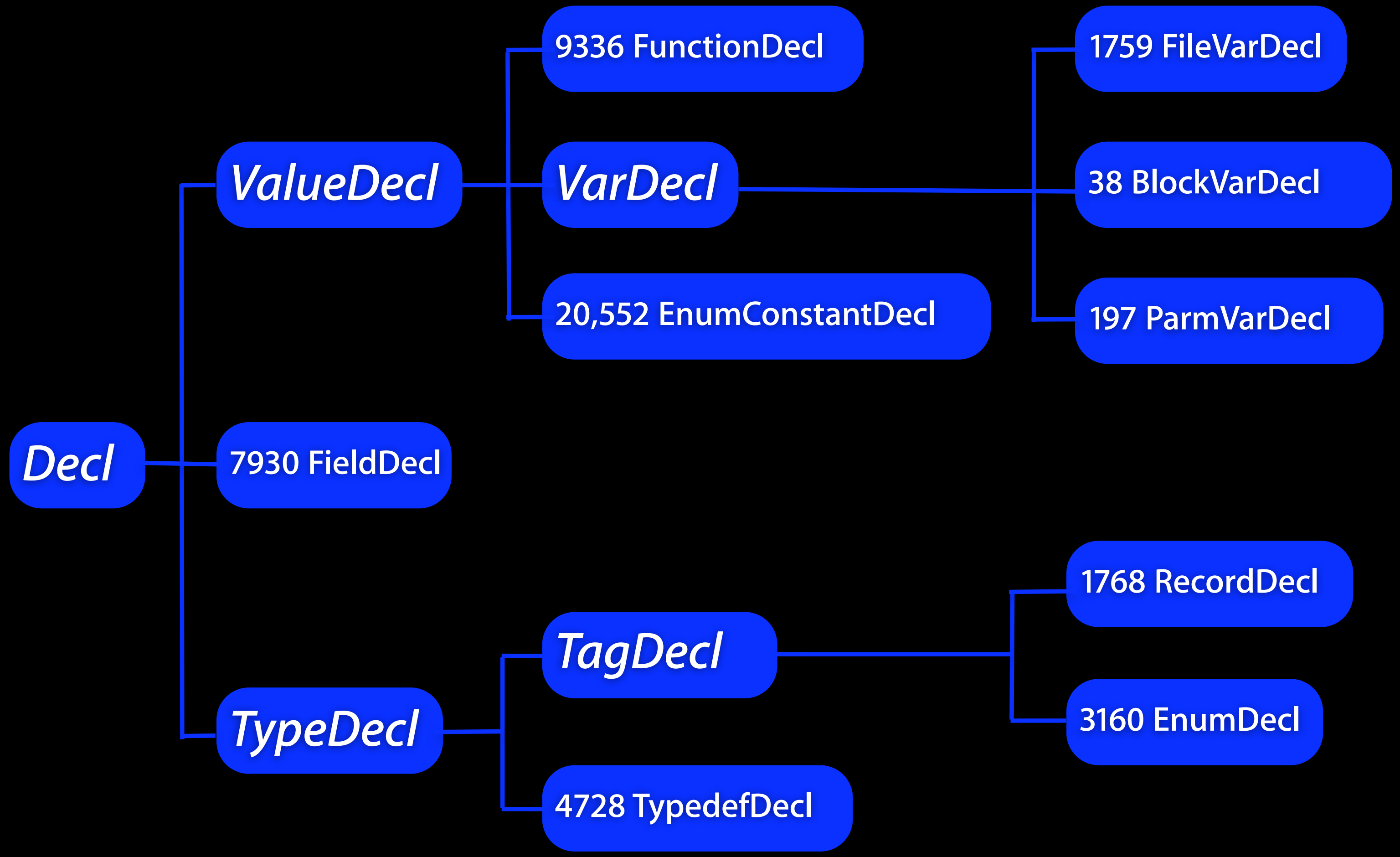
Space


clang 30% larger than source
gcc 10x larger than source
12,285,640
3,578,038
614,044
804,716
1,504,840
2,104,780
30,000,000
Input Source
Preprocessed Source
clang AST's
GCC trees
Characters
Stmt/Expr AST (31532)
Identifiers/Strings (51936)
Type AST (26684)
Decl AST (49468)
GCC tree nodes

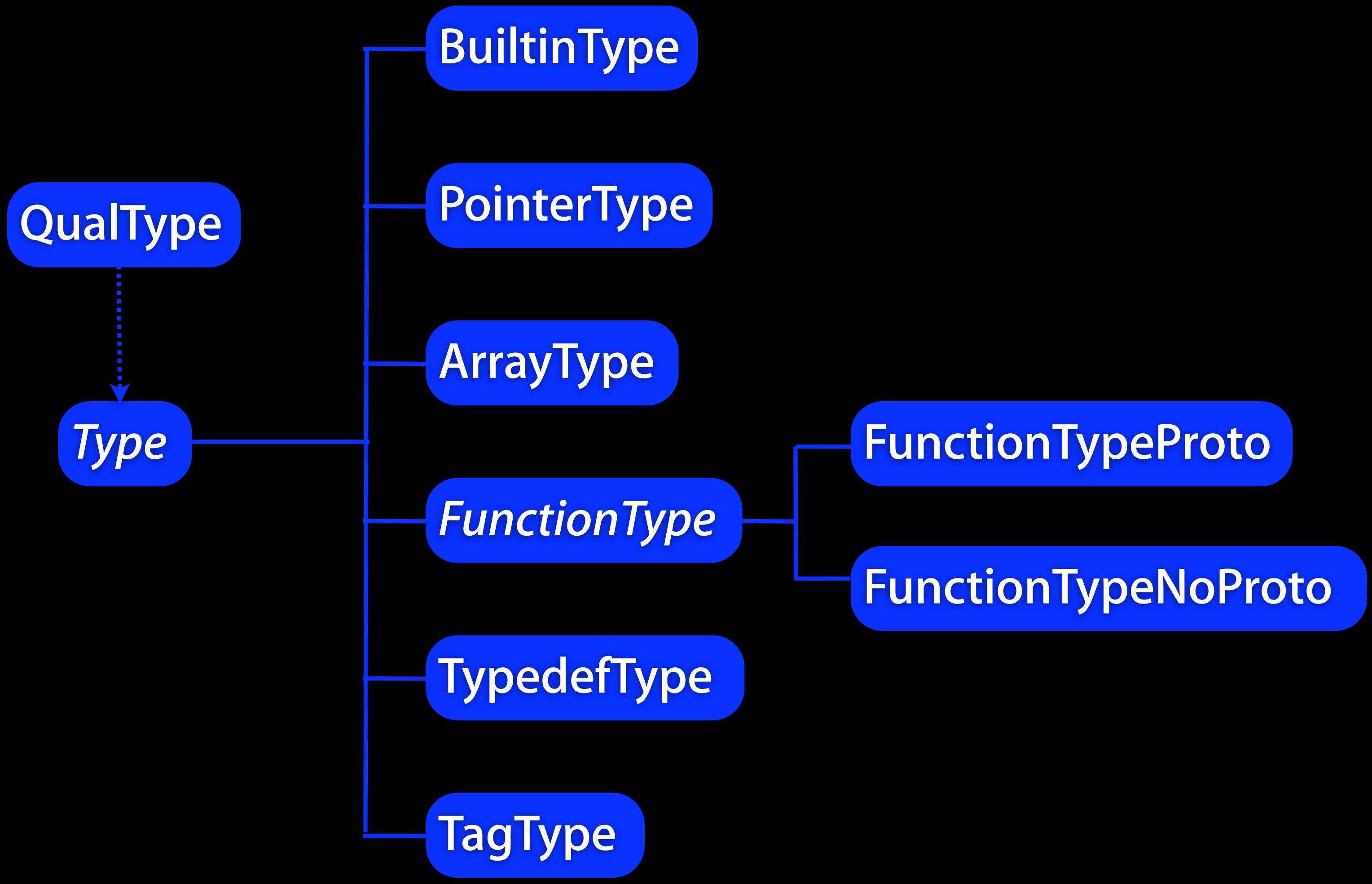
Abstract Syntax Trees - Declarations



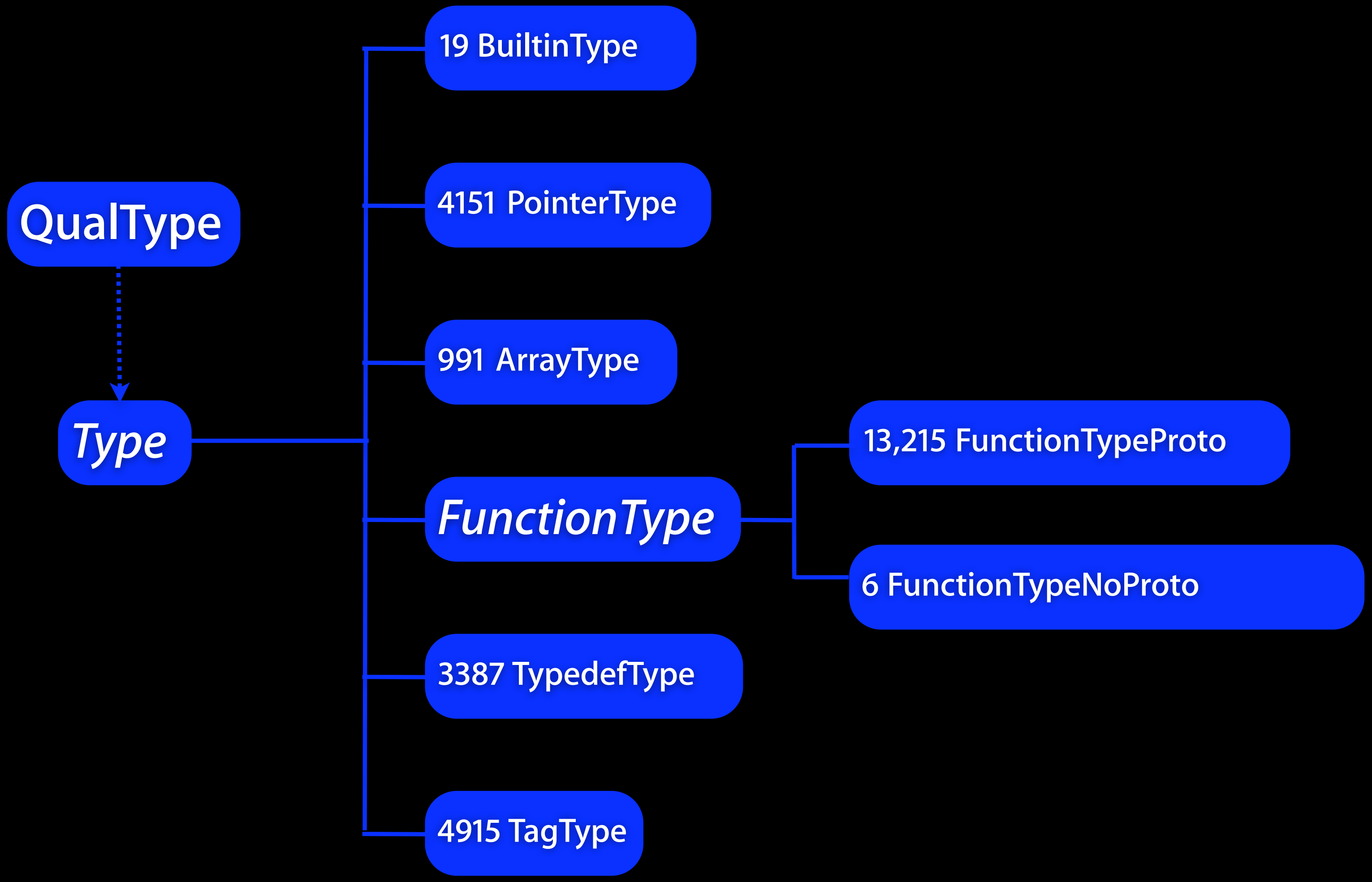
Abstract Syntax Trees - “carbon.h” decls



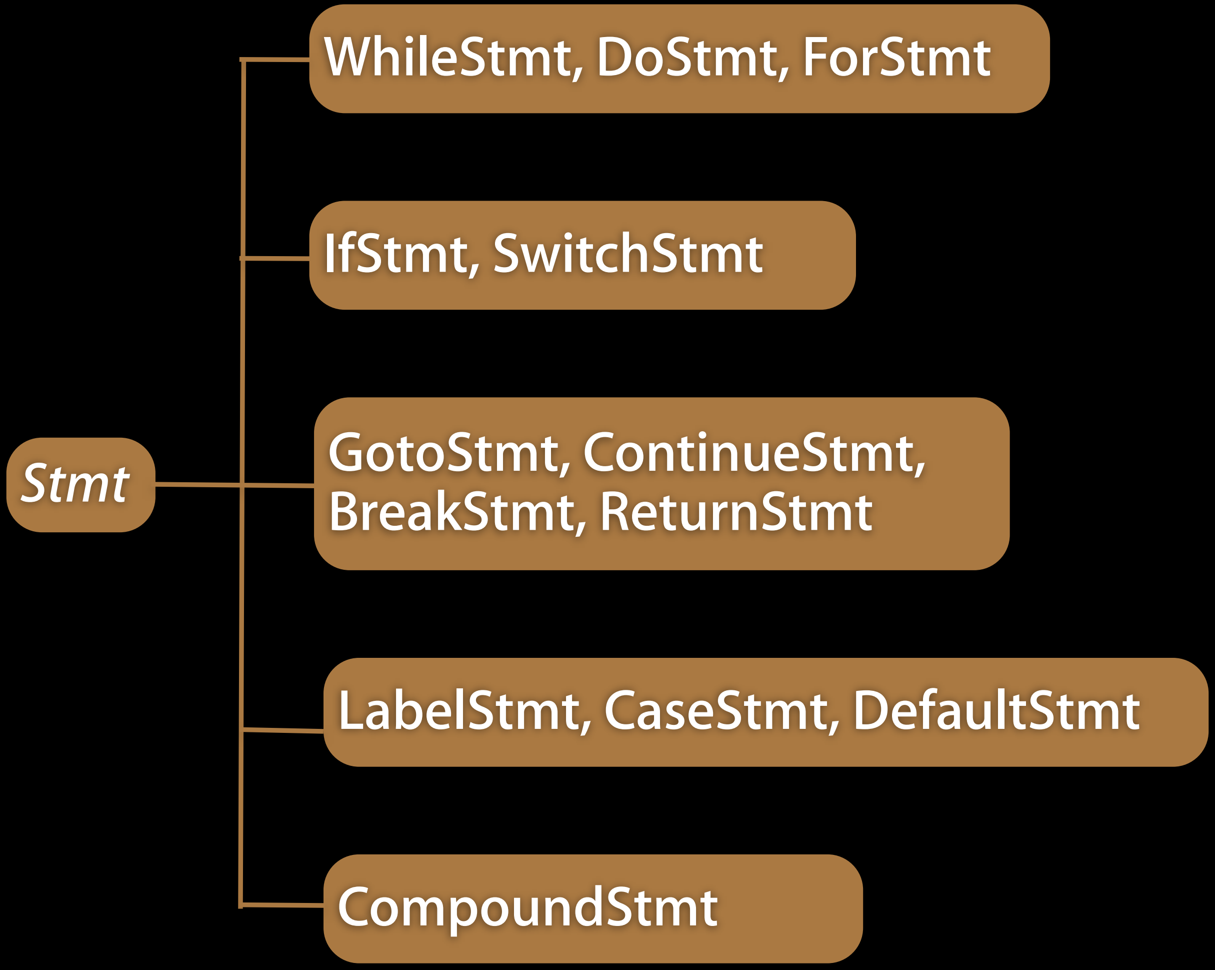
Abstract Syntax Trees - Types



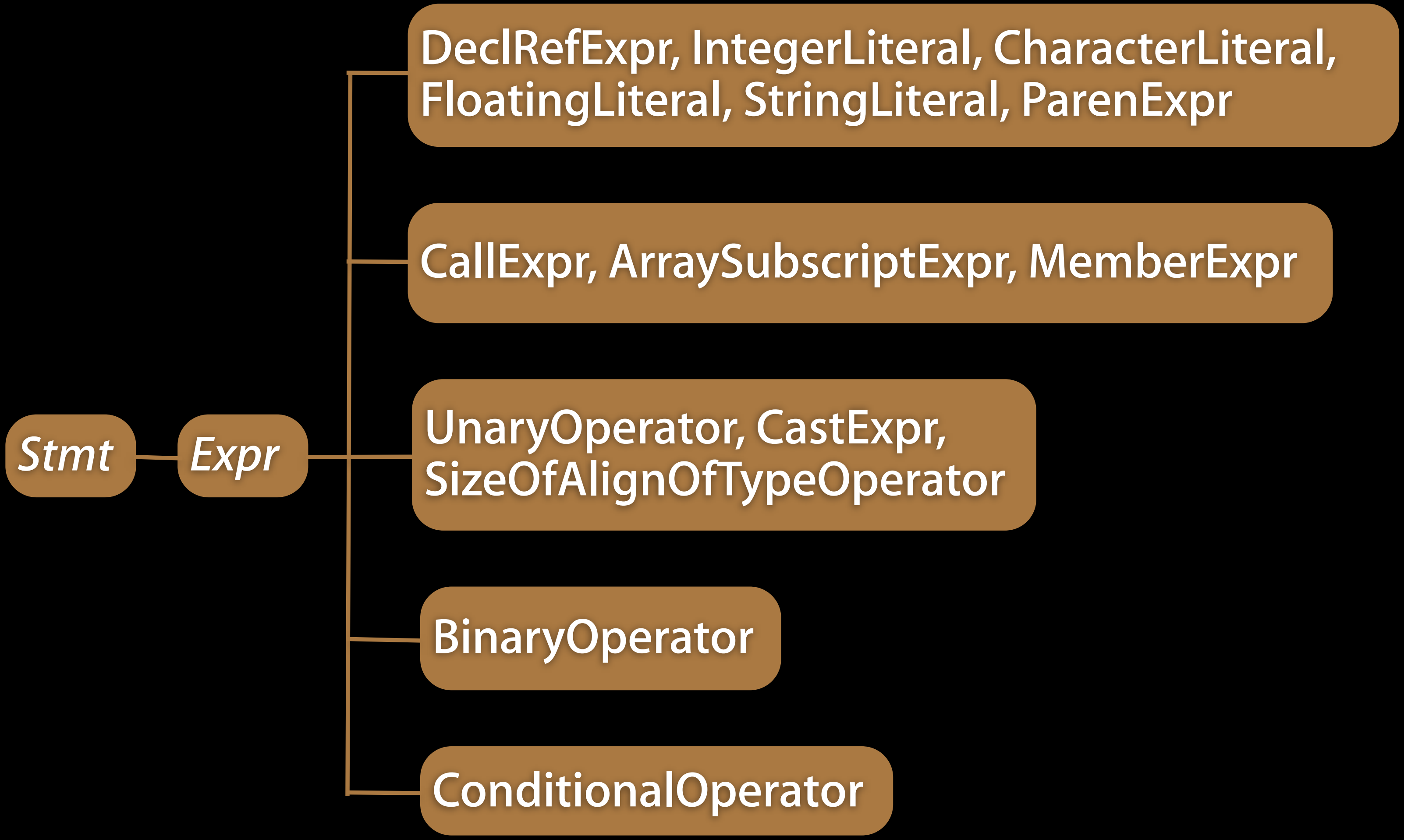
Abstract Syntax Trees - “carbon.h” types



Abstract Syntax Trees - Statements


Stmt
WhileStmt, DoStmt, ForStmt
IfStmt, SwitchStmt
GotoStmt, ContinueStmt, BreakStmt, ReturnStmt
LabelStmt, CaseStmt, DefaultStmt
CompoundStmt

Abstract Syntax Trees - Expressions


Stmt
Expr
DeclRefExpr, IntegerLiteral, CharacterLiteral, FloatingLiteral, StringLiteral, ParenExpr
CallExpr, ArraySubscriptExpr, MemberExpr
UnaryOperator, CastExpr, SizeOfAlignOfTypeOperator
BinaryOperator
ConditionalOperator

Guiding Thoughts

Good artists copy. Great artists steal.

- Pablo Picasso

Guiding Thoughts

There is a pleasure in creating well-written, understandable programs. There is a satisfaction in finding a program structure that tames the complexity of an application. We enjoy seeing our algorithms expressed clearly and persuasively. We also profit from our clearly written programs, for they are much more likely to be correct and maintainable than obscure ones.

- Harbison 1992