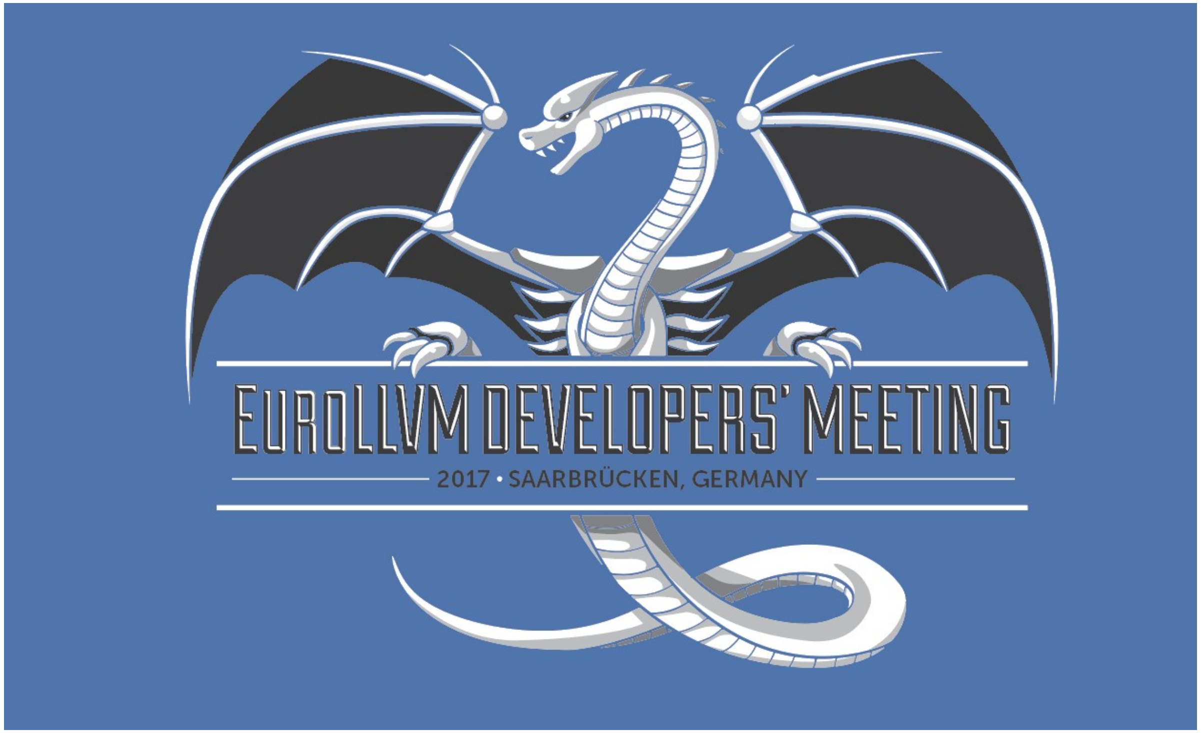
Clank: Java-port of C/C++ Frontend

Sharing the NetBeans Team’s Experience

Petr Kudriavtsev
Vladimir Voskresensky
Oracle

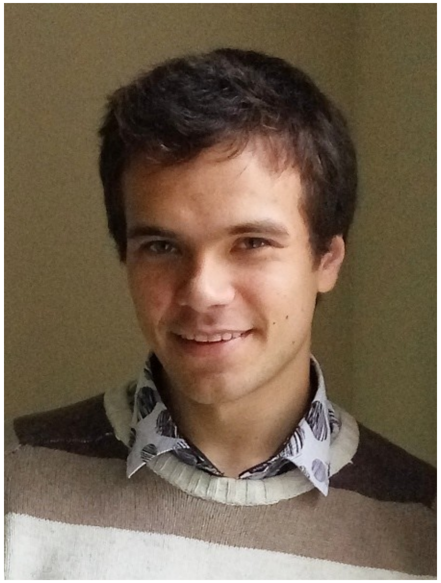


March 27, 2017

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.

Speakers



Petr Kudriavtsev

Vladimir Voskresensky

Agenda

- Why porting?
- Known approaches
- Converter
- Porting C++ and Clang challenges
- Clank Demo

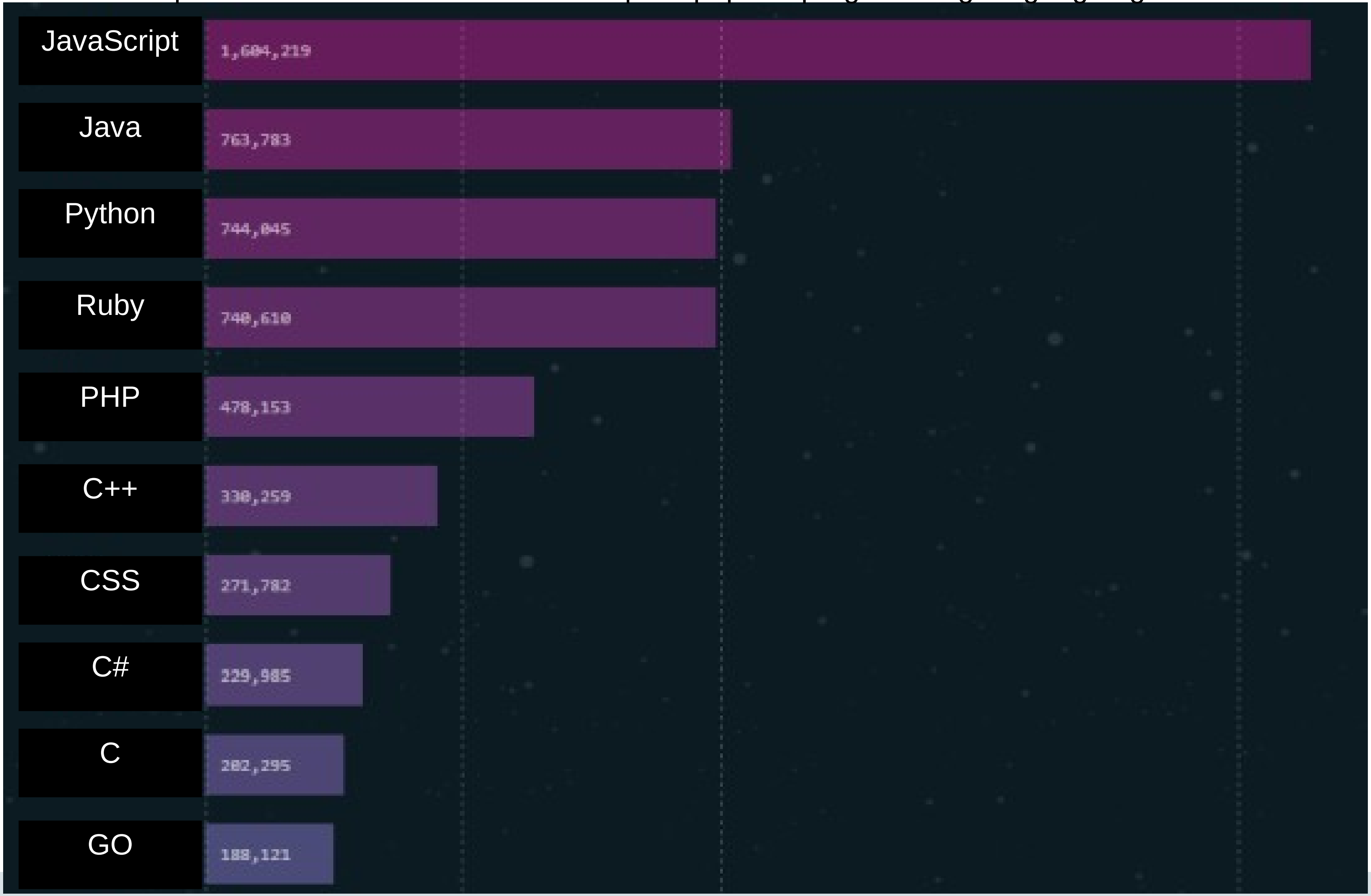
Why not binding?

- Why Emscripten?
  - LLVM IR to JavaScript 'assembler'?
- Why Lucene => CLucene?
  - Java ported to C++?
- Why Hibernate => NHibernate?
  - Java ported to .NET?
- Why people do porting?
  - It's fun!

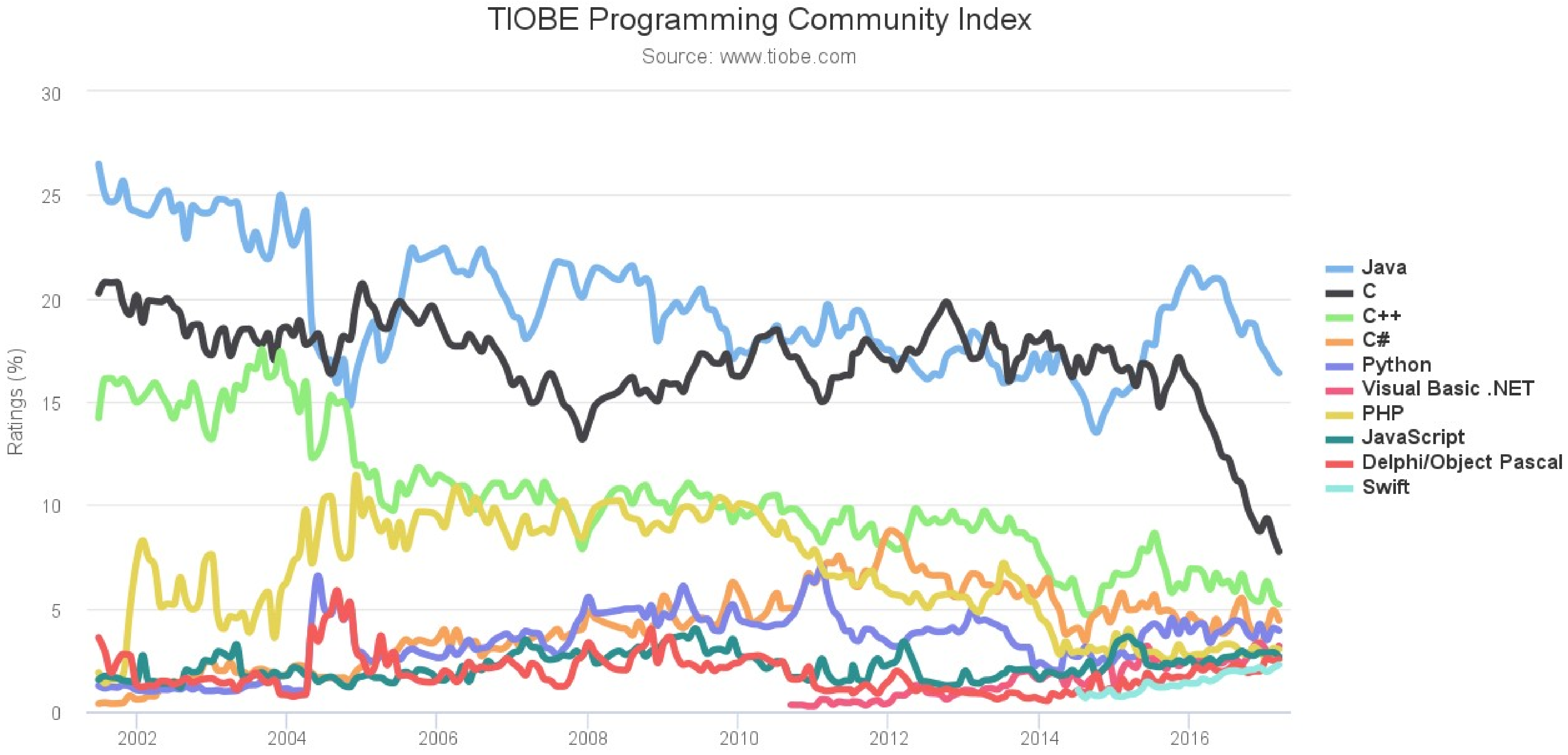
C++ and 2*C == Java

The 10 most popular computer languages on GitHub

https://www.techworm.net/2016/09/top-10-popular-programming-languages-github.html



C++ and C == Java


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
Java
C
C++
C#
Python
Visual Basic .NET
PHP
JavaScript
Delphi/Object Pascal
Swift

What is our favorite C++ Technology?

What is our favorite C++ Technology?

No religious wars!
Let's share Clang Technology

Add One More Thread Holding Developers Together

Clang Technology evaluation

- Native Clang library requirements without functional regressions:
  - Full access to the strength of technology
  - All Java-aware platforms
  - Safety
  - Debug
  - Performance of native clang
  - JNI/JNA Bridging overhead
  - Upgrade to new Clang release

Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG …

Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG …
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits

Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG …
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!

Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG …
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet…

Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet...
- Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower

Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG …
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet...
- Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
- JNI/JNA Bridging overhead
  - Need to expose whole AST API

Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet...
- Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
- JNI/JNA Bridging overhead
  - Need to expose whole AST API
- Upgrade to new Clang release

Clang Technology evaluation (JNI/JNA prototyping)

- ✗ Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG …
- ✗ All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- ✗ Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- ✗ Debug
  - We hadn't have Mixed-dev in NetBeans yet...
- ✗ Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
- ✗ JNI/JNA Bridging overhead
  - Need to expose whole AST API
- ✓ Upgrade to new Clang release

Conclusion: Clang doesn't bring any extra value?

Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG …
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet…
- Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
- JNI/JNA Bridging overhead
  - Need to expose whole AST API
- Upgrade to new Clang release

Wait! Let's try Clang in Java!

Clang Technology evaluation (JNI/JNA prototyping)

- ✓ Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG …
- ✓ All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- ✓ Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- ✓ Debug
  - We hadn't have Mixed-dev in NetBeans yet...
- ✗ Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
- ✓ JNI/JNA Bridging overhead
  - Need to expose whole AST API
- ✗ Upgrade to new Clang release

Wait! Let's try Clang in Java!

Agenda

- Why porting?
- **Known approaches**
- Converter
- Porting C++ and Clang challenges
- Clank Demo

Manual

- Inspired by ...

Manual

LLVM IR Based

LLVM IR Based

- Inspired by Emscripten

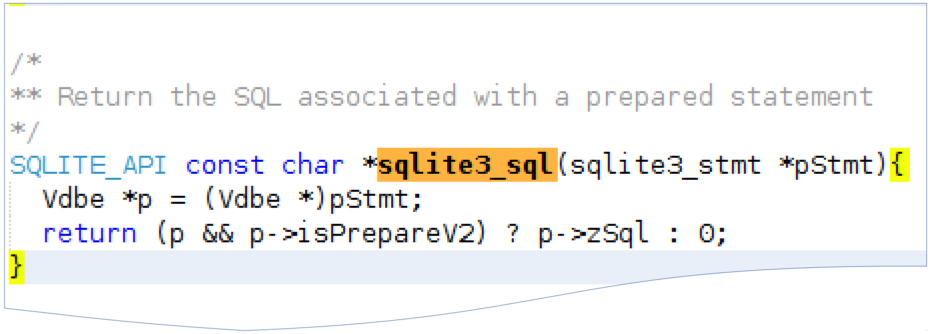
LLVM IR Based

- Inspired by Emscripten
- Transform LLVM IR to Java Bytecode

LLVM IR Based

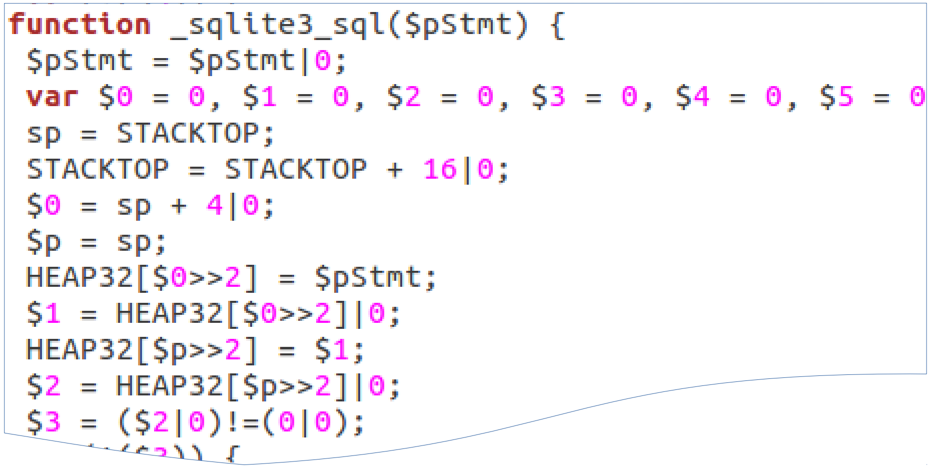
- Inspired by Emscripten
- Transform LLVM IR to Java Bytecode
- Assembler Level Output
  - Difficult to understand
  - Difficult to debug by client

C++



```
/*
** Return the SQL associated with a prepared statement
*/
SQLITE_API const char *sqlite3_sql(sqlite3_stmt *pStmt){
  Vdbe *p = (Vdbe *)pStmt;
  return (p && p->isPrepareV2) ? p->zSql : 0;
}
```

JavaScript



```
function _sqlite3_sql($pStmt) {
 $pStmt = $pStmt|0;
 var $0 = 0, $1 = 0, $2 = 0, $3 = 0, $4 = 0, $5 = 0
 sp = STACKTOP;
 STACKTOP = STACKTOP + 16|0;
 $0 = sp + 4|0;
 $p = sp;
 HEAP32[$0>>2] = $pStmt;
 $1 = HEAP32[$0>>2]|0;
 HEAP32[$p>>2] = $1;
 $2 = HEAP32[$p>>2]|0;
 $3 = ($2|0)!=(0|0);
```

LLVM IR Based

- Inspired by Emscripten
- Transform LLVM IR to Java Bytecode
- Assembler Level Output
  - Difficult to understand
  - Difficult to debug by client
- Java AST* APIs are needed to be generated from C-like IR back to Java Classes/methods

Existing C++ to Java Converters

Existing C++ to Java Converters

Low Accuracy on C++11 Codebases

Existing C++ to Java Converters

Low Accuracy on C++11 Codebases

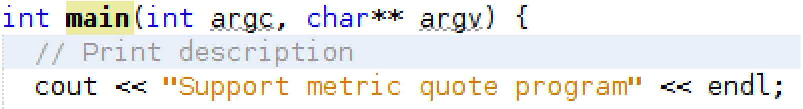
Clang Based

- Inspired by ast-print

Clang Based

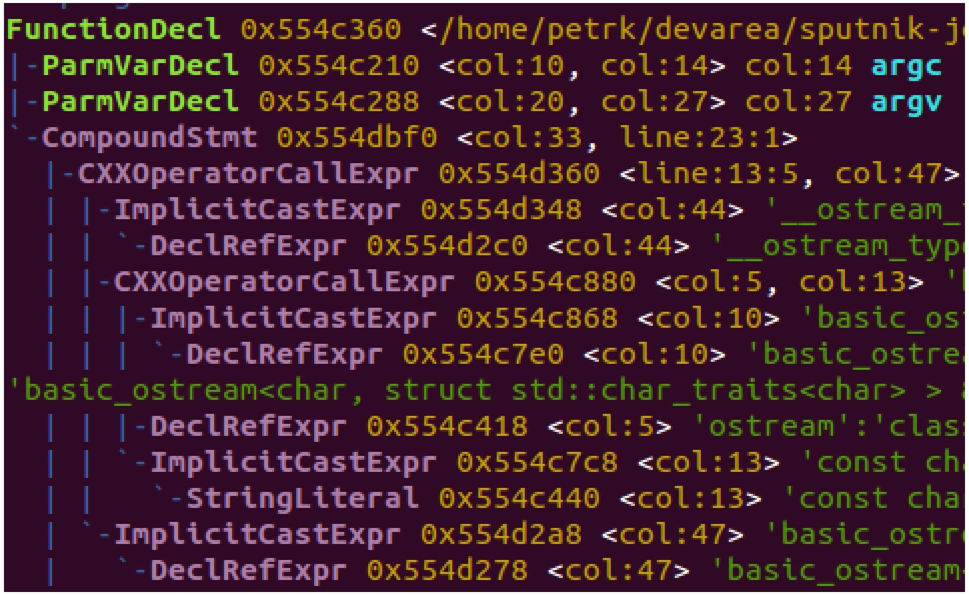
- Inspired by ast-print
  - Clang: C++ Source to Clang-AST

C++:



```
int main(int argc, char** argv) {
  // Print description
  cout << "Support metric quote program" << endl;
```

AST:

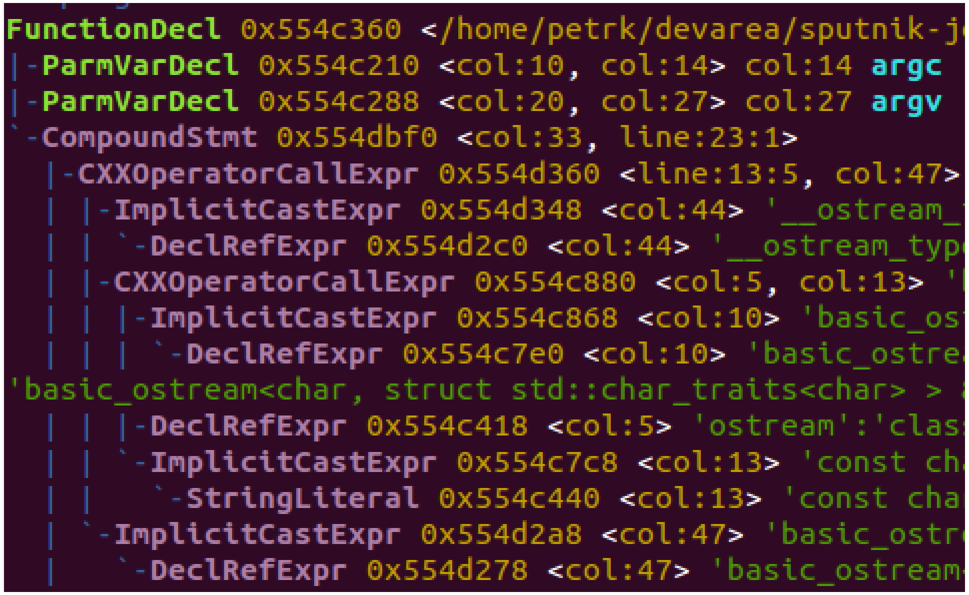


```
FunctionDecl 0x554c360 </home/petrk/devarea/sputnik-j
|-ParmVarDecl 0x554c210 <col:10, col:14> col:14 argc
|-ParmVarDecl 0x554c288 <col:20, col:27> col:27 argv
`-CompoundStmt 0x554dbf0 <col:33, line:23:1>
  |-CXXOperatorCallExpr 0x554d360 <line:13:5, col:47>
  | |-ImplicitCastExpr 0x554d348 <col:44> '__ostream_
  | | `-DeclRefExpr 0x554d2c0 <col:44> '__ostream_typ
  | |-CXXOperatorCallExpr 0x554c880 <col:5, col:13> '
  | | |-ImplicitCastExpr 0x554c868 <col:10> 'basic_os
  | | | `-DeclRefExpr 0x554c7e0 <col:10> 'basic_ostre
'basic_ostream<char, struct std::char_traits<char> >
  | | |-DeclRefExpr 0x554c418 <col:5> 'ostream':'clas
  | | `-ImplicitCastExpr 0x554c7c8 <col:13> 'const ch
  | |   `-StringLiteral 0x554c440 <col:13> 'const cha
  | `-ImplicitCastExpr 0x554d2a8 <col:47> 'basic_ostr
  |   `-DeclRefExpr 0x554d278 <col:47> 'basic_ostream
```

Clang Based

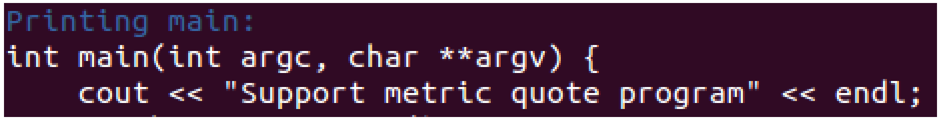
- Inspired by ast-print
  - Clang: C++ Source to Clang-AST
  - ast-print: Clang-AST to C++ source

AST:



```
FunctionDecl 0x554c360 </home/petrk/devarea/sputnik-j
|-ParmVarDecl 0x554c210 <col:10, col:14> col:14 argc
|-ParmVarDecl 0x554c288 <col:20, col:27> col:27 argv
`-CompoundStmt 0x554dbf0 <col:33, line:23:1>
  |-CXXOperatorCallExpr 0x554d360 <line:13:5, col:47>
  | |-ImplicitCastExpr 0x554d348 <col:44> '__ostream_
  | | `-DeclRefExpr 0x554d2c0 <col:44> '__ostream_typ
  | |-CXXOperatorCallExpr 0x554c880 <col:5, col:13> '
  | | |-ImplicitCastExpr 0x554c868 <col:10> 'basic_os
  | | | `-DeclRefExpr 0x554c7e0 <col:10> 'basic_ostre
'basic_ostream<char, struct std::char_traits<char> > 
  | | |-DeclRefExpr 0x554c418 <col:5> 'ostream':'clas
  | | `-ImplicitCastExpr 0x554c7c8 <col:13> 'const ch
  | |   `-StringLiteral 0x554c440 <col:13> 'const cha
  | `-ImplicitCastExpr 0x554d2a8 <col:47> 'basic_ostr
  |   `-DeclRefExpr 0x554d278 <col:47> 'basic_ostream
```

C++:

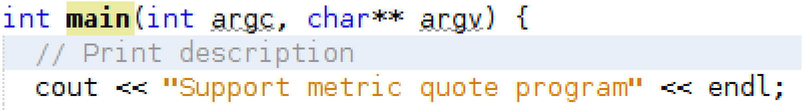


```
Printing main:
int main(int argc, char **argv) {
    cout << "Support metric quote program" << endl;
```

Clang Based

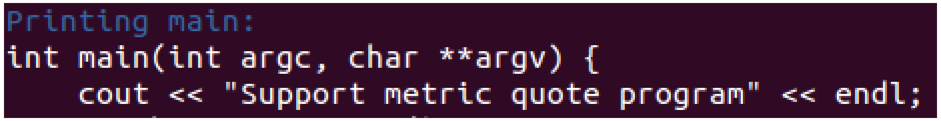
- Inspired by ast-print
  - Clang: C++ Source to Clang-AST
  - ast-print: Clang-AST to C++ source
- Comments are missed

C++:



```
int main(int argc, char** argv) {
  // Print description
  cout << "Support metric quote program" << endl;
```

Printed C++:



```
Printing main:
int main(int argc, char **argv) {
    cout << "Support metric quote program" << endl;
```

Clang Based

- Inspired by ast-print
  - Clang: C++ Source to Clang-AST
  - ast-print: Clang-AST to C++ source
- Comments are missed
- But looks very promising!

Clang Based

- Inspired by ast-print
  - Clang: C++ Source to Clang-AST
  - ast-print: Clang-AST to C++ source
- Comments are missed
- But looks very promising!

Convert whole Clang-AST
to Java Source!

Agenda

- Why porting?
- Known approaches
- **Converter**
- Porting C++ and Clang challenges
- Clank Demo

Prototype Converter

- Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy

Prototype Converter

- Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy
- Let's try to port!

Prototype Converter

- Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy
- Let's try to port!
  - And I'm going on vacation

Prototype Converter

- Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy
- Let's try to port!
  - After 2 weeks...

Prototype Converter

- Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy
- Let's try to port!
  - After 2 weeks...

Prototype Converter

- Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy
- Let's try to port!
  - After 2 weeks...

Team conclusion: Don't bother us with your crazy dreams!
It is still manual!

Need a Plan…

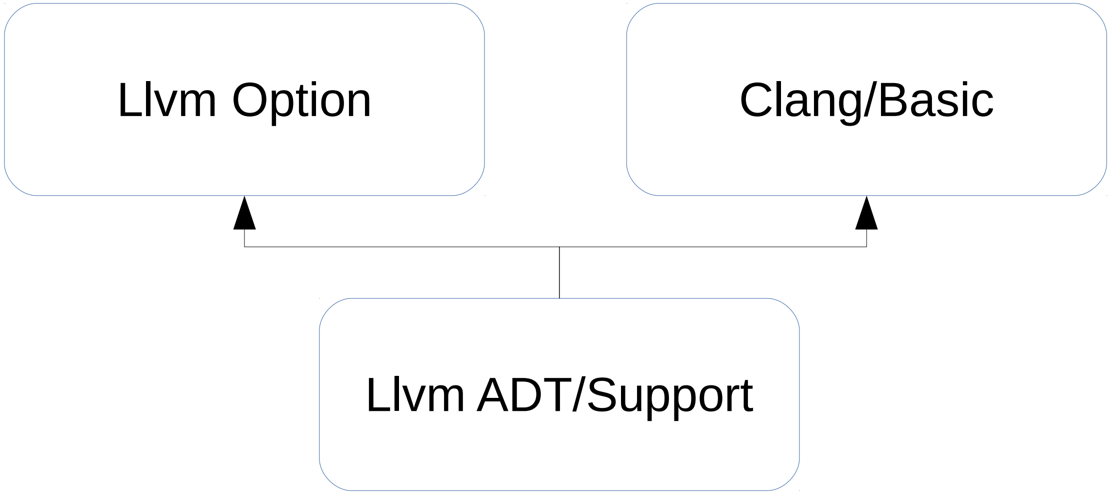
Need a Plan…

- Bottom up approach
  - for API

Llvm ADT/Support

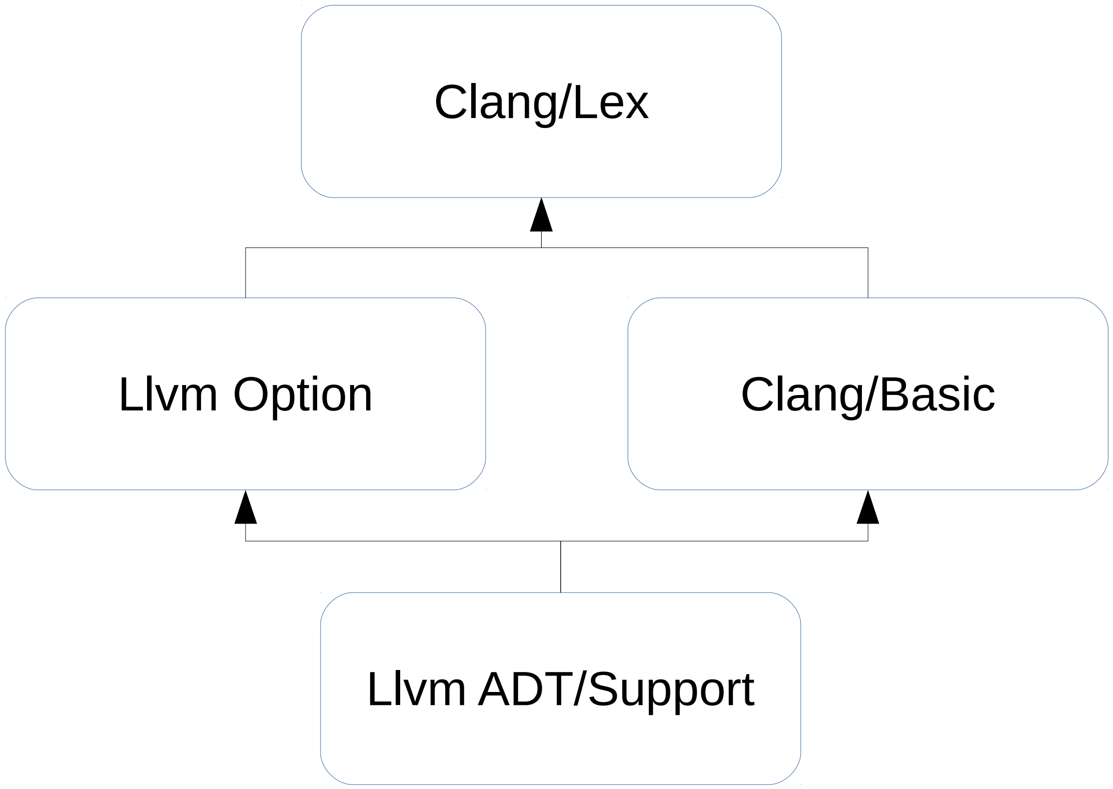
Need a Plan...

- Bottom up approach
  - for API



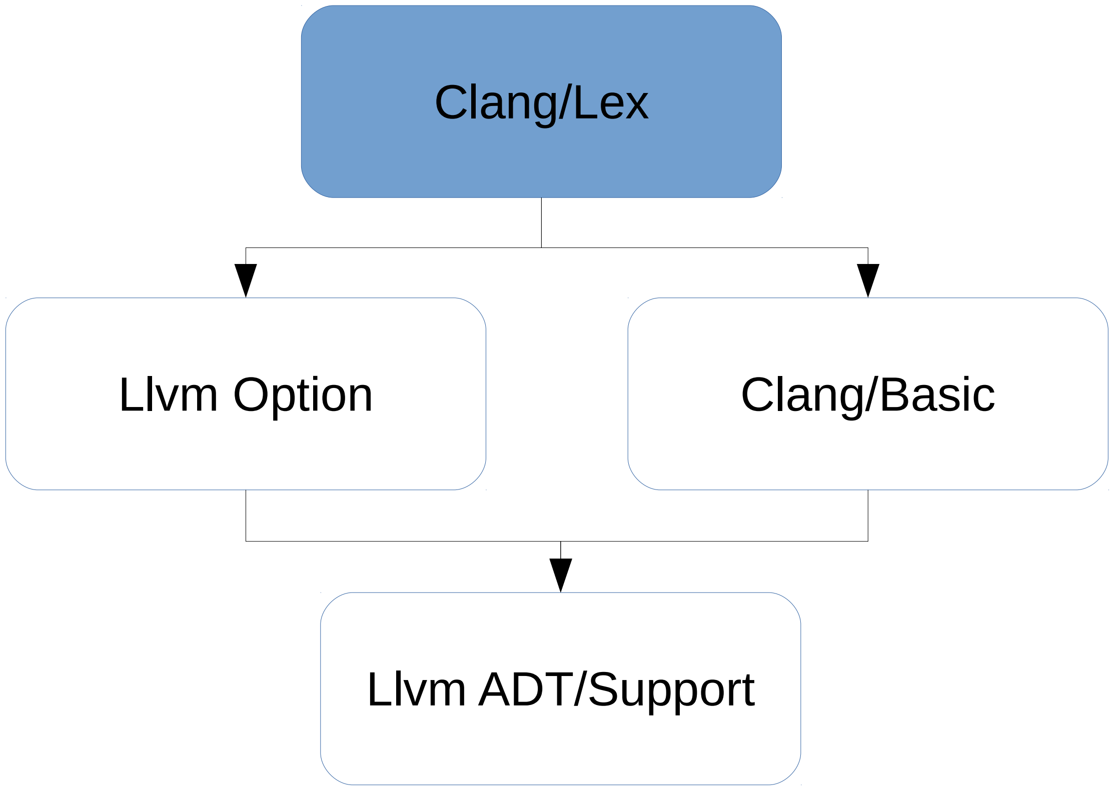
Need a Plan...

- Bottom up approach
  - for API



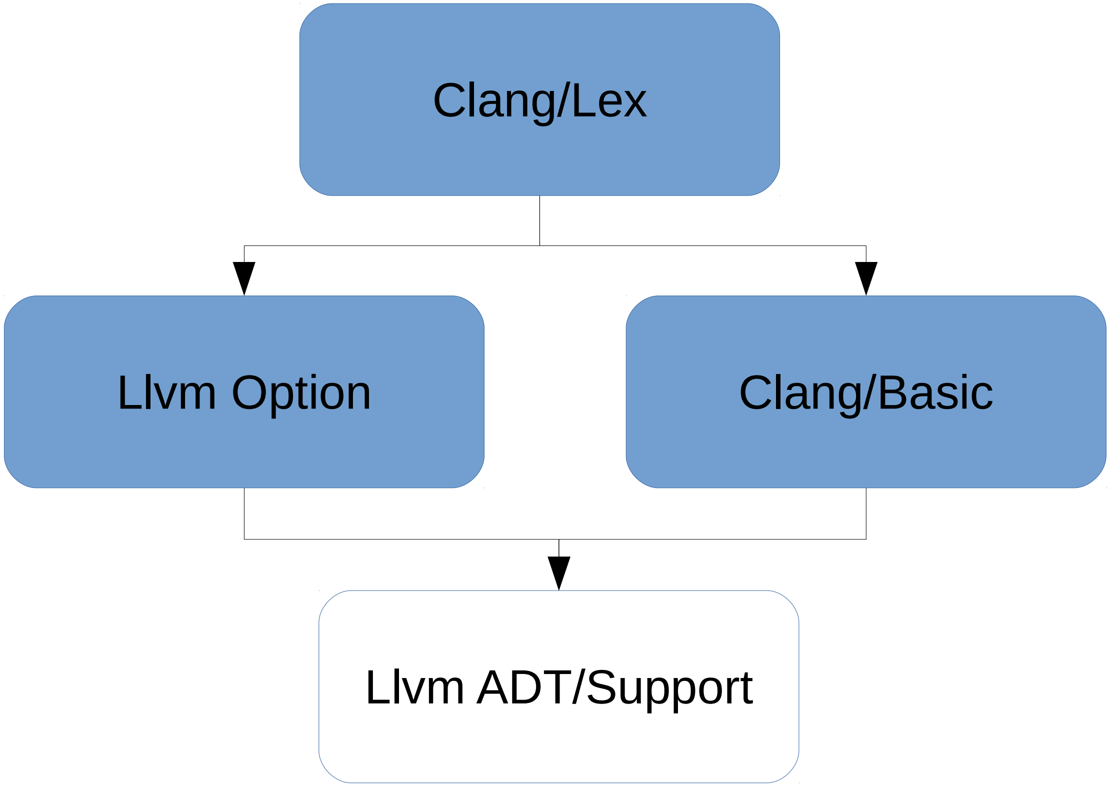
Need a Plan...

- Followed by Top down approach
  - for implementations



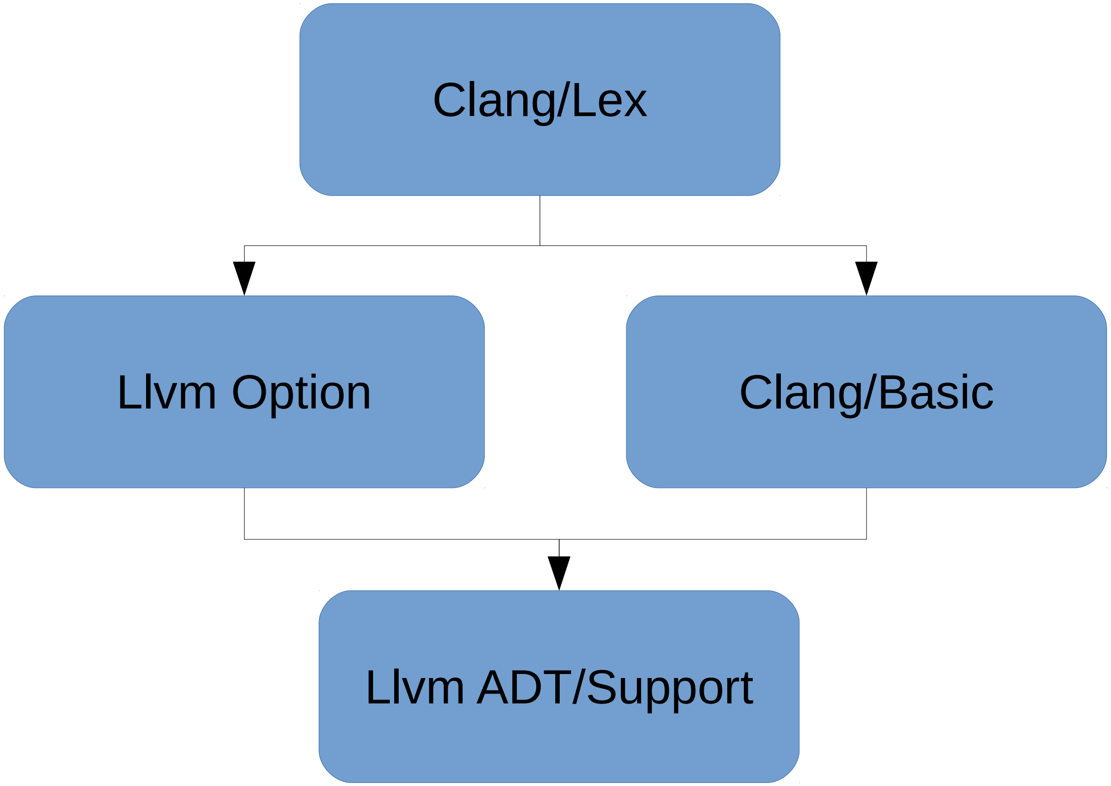
Need a Plan...

- Followed by Top down approach
  - for implementations



Need a Plan...

- Followed by Top down approach
  - for implementations



Need a Plan...

- Bottom up approach
  - Generate APIs without bodies
- Followed by Top down approach
  - Generate bodies starting from clients
    - Let's try Lex module
    - To build infrastructure
    - To evaluate ported Preprocessor
  - Adjusting APIs when better learn Clang/LLVM
    - Easy, fast, because bodies are absent
    - Add Java's “LibC++” and ADT/Support on demand
- Use existing Clang tests to check semantic
- Annotate Java code to get help from IDE
- Release within NetBeans C++ support

Same Time at Different World...

- Use Clang technology to parse C++
- Walk Clang AST to print Java code

During Short Nights...

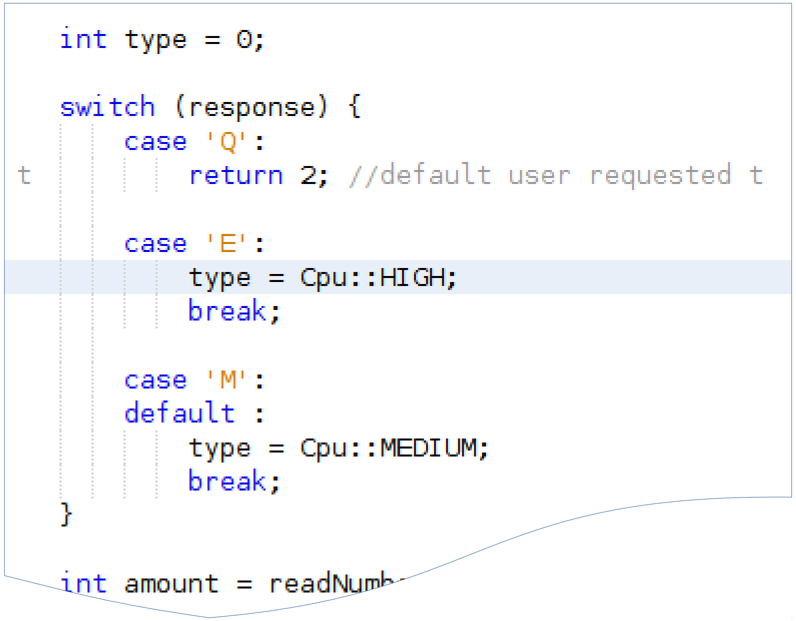
- Use Clang technology to parse C++
- Walk Clang AST to print Java code
- 2 weeks to prototype JConvert
  - Port sample C++ project to Java
  - Keep semantic
  - Keep code as close as possible
  - Keep comments

And Long Weekends...

- Use Clang technology to parse C++
- Walk Clang AST to print Java code
- 2 weeks to prototype JConvert
  - Port sample C++ project to Java
  - Keep semantic
  - Keep code as close as possible
  - Keep comments
- Demo

JConvert 0.0.1

- C++ Quote vs Java Quote snippets



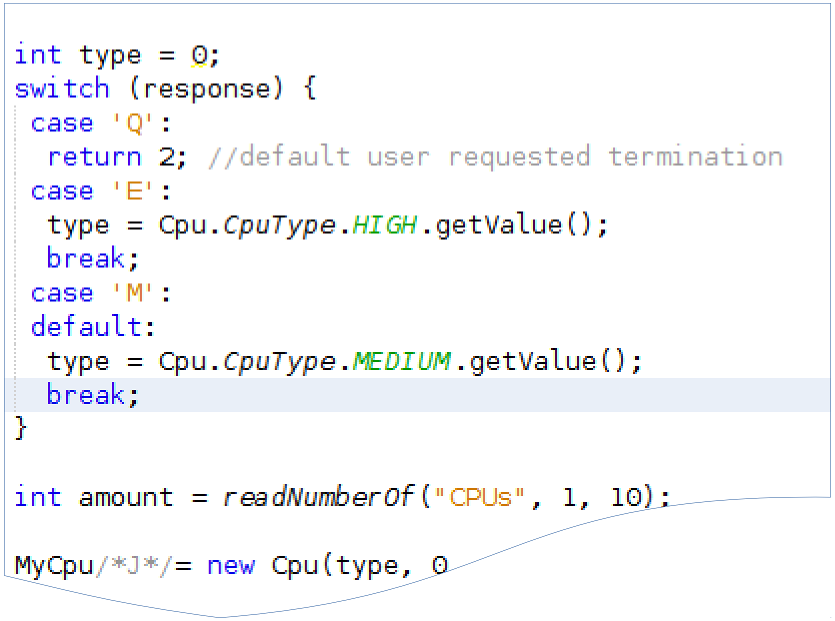
```
   int type = 0;

   switch (response) {
       case 'Q':
t          return 2; //default user requested t

       case 'E':
           type = Cpu::HIGH;
           break;

       case 'M':
       default :
           type = Cpu::MEDIUM;
           break;
   }

   int amount = readNumb
```



```
int type = 0;
switch (response) {
 case 'Q':
  return 2; //default user requested termination
 case 'E':
  type = Cpu.CpuType.HIGH.getValue();
  break;
 case 'M':
 default:
  type = Cpu.CpuType.MEDIUM.getValue();
  break;
}

int amount = readNumberOf("CPUs", 1, 10);

MyCpu/*J*/= new Cpu(type, 0
```

It works for sample C++ project!

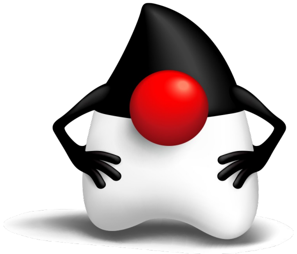
Agenda

- Why porting?
- Known approaches
- Converter
- **Porting C++ and Clang challenges**
- Clank Demo

Clang - Pronunciation: /klaNG/

A loud, resonant metallic sound or series of sounds

- Oxford Dictionary



Clank - Pronunciation: /klaNGk/

A loud, sharp sound or series of sounds, typically made by pieces of metal meeting or being struck together

- Oxford Dictionary

Clank: As close to origin as possible

- Convert Clang components for fully functional Preprocessor
  - Keeps comments
  - Semantically equivalent
  - Passes Clang tests
- Pure Java
  - Modular
  - Java “LibC++”
- Adopted by NetBeans
- The same License as LLVM
  - “Wanted the code to be **used!**” quoting Chris Lattner

“All hope abandon, ye who enter here.”

— Dante Alighieri, The Divine Comedy

C++ in Java Challenges

- Names collisions
  - Non-virtual methods in base and derived classes
    - In Java all methods are virtual
  - 'unsigned int' vs 'int' overloaded methods and constructors
- Diagnostics are not printed
  - Temporary objects lifecycle
- Multiple inheritance
- Compile time preprocessor-conditional code in FileSystem
  - Changed #ifdef/#else/#endif to runtime
- Split by TUs vs Monolithic Java classes
- this+1 and TrailingObjects
- Custom new operators
- **JAVA code Performance**

Clank: All is solvable

- ✓ Names collisions
  - – Non-virtual methods in base and derived classes
    - In Java all methods are virtual
  - – 'unsigned int' vs 'int' overloaded methods and constructors
- ✓ Diagnostics are not printed
  - – Temporary objects lifecycle
- ✓ Multiple inheritance
- ✓ Compile time preprocessor-conditional code in FileSystem
  - – Changed #ifdef/#else/#endif to runtime
- ✓ Split by TUs vs Monolithic Java classes
- ✓ this+1 and TrailingObjects
- ✓ Custom new operators
- ✓ JAVA Clank Preprocessor Performance

Clank: All is solvable

Complete and fast Clank Preprocessor, 1.1 MLoc, integrated into NetBeans

✓ Names collisions
  - Non-virtual methods in base and derived classes
    - In Java all methods are virtual
  - 'unsigned int' vs 'int' overloaded methods and constructors

✓ Diagnostics are not printed
  - Temporary objects lifecycle

✓ Multiple inheritance

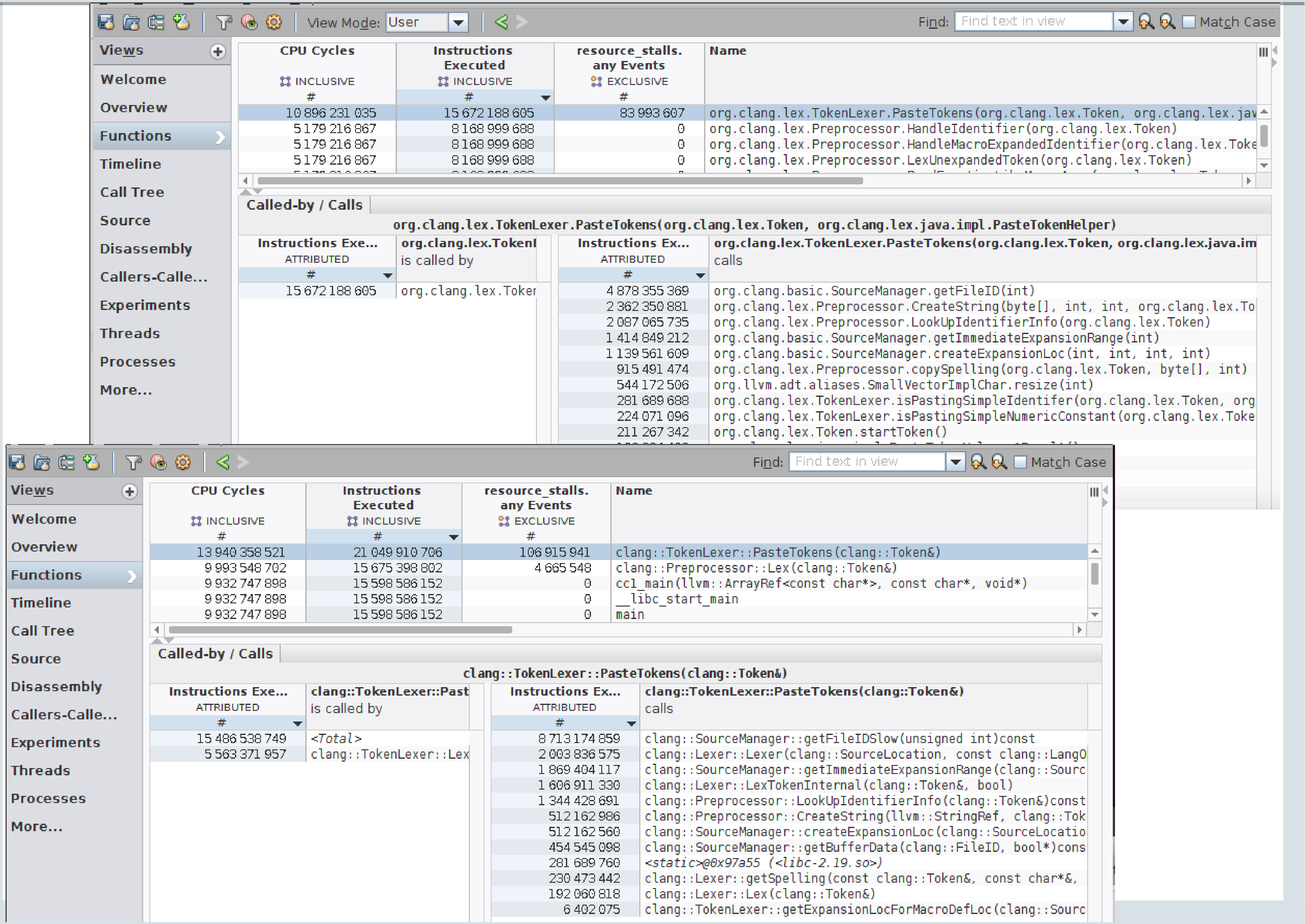
✓ Compile time preprocessor-conditional code in FileSystem
  - Changed #ifdef/#else/#endif to runtime

✓ Split by TUs vs Monolithic Java classes

✓ this+1 and TrailingObjects

✓ Custom new operators

✓ JAVA Clank Preprocessor Performance



CPU Cycles INCLUSIVE #	Instructions Executed INCLUSIVE #	resource_stalls. any Events EXCLUSIVE #	Name
10 896 231 035	15 672 188 605	83 993 607	org.clang.lex.TokenLexer.PasteTokens(org.clang.lex.Token, org.clang.lex.jav
5 179 216 867	8 168 999 688	0	org.clang.lex.Preprocessor.HandleIdentifier(org.clang.lex.Token)
5 179 216 867	8 168 999 688	0	org.clang.lex.Preprocessor.HandleMacroExpandedIdentifier(org.clang.lex.Toke
5 179 216 867	8 168 999 688	0	org.clang.lex.Preprocessor.LexUnexpandedToken(org.clang.lex.Token)

org.clang.lex.TokenLexer.PasteTokens(org.clang.lex.Token, org.clang.lex.java.impl.PasteTokenHelper)

Instructions Exe... ATTRIBUTED #	org.clang.lex.TokenI is called by
15 672 188 605	org.clang.lex.Toker

Instructions Ex... ATTRIBUTED #	org.clang.lex.TokenLexer.PasteTokens(org.clang.lex.Token, org.clang.lex.java.im calls
4 878 355 369	org.clang.basic.SourceManager.getFileID(int)
2 362 350 881	org.clang.lex.Preprocessor.CreateString(byte[], int, int, org.clang.lex.To
2 087 065 735	org.clang.lex.Preprocessor.LookUpIdentifierInfo(org.clang.lex.Token)
1 414 849 212	org.clang.basic.SourceManager.getImmediateExpansionRange(int)
1 139 561 609	org.clang.basic.SourceManager.createExpansionLoc(int, int, int, int)
915 491 474	org.clang.lex.Preprocessor.copySpelling(org.clang.lex.Token, byte[], int)
544 172 506	org.llvm.adt.aliases.SmallVectorImplChar.resize(int)
281 689 688	org.clang.lex.TokenLexer.isPastingSimpleIdentifer(org.clang.lex.Token, org
224 071 096	org.clang.lex.TokenLexer.isPastingSimpleNumericConstant(org.clang.lex.Toke
211 267 342	org.clang.lex.Token.startToken()

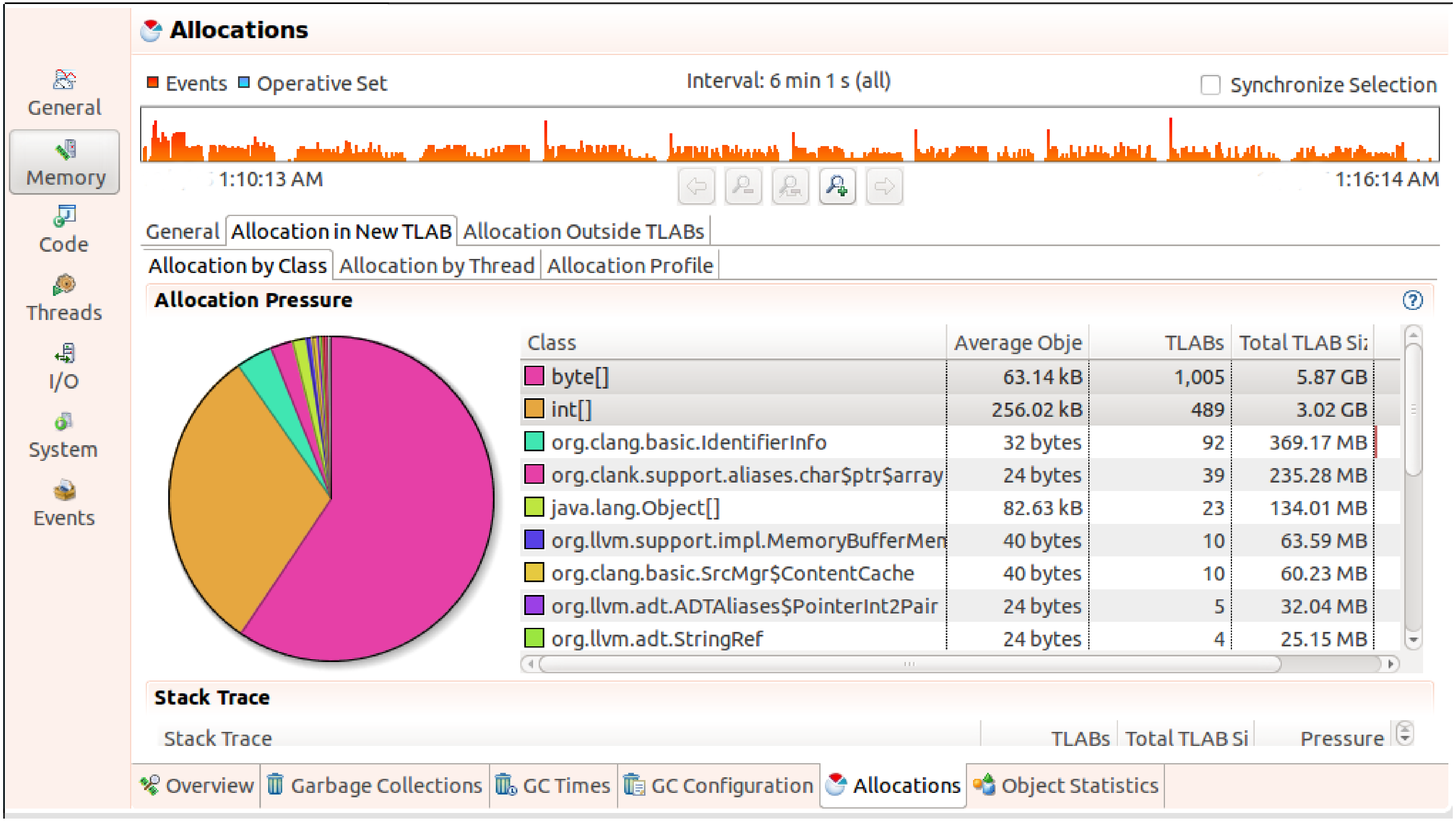
CPU Cycles INCLUSIVE #	Instructions Executed INCLUSIVE #	resource_stalls. any Events EXCLUSIVE #	Name
13 940 358 521	21 049 910 706	106 915 941	clang::TokenLexer::PasteTokens(clang::Token&)
9 993 548 702	15 675 398 802	4 665 548	clang::Preprocessor::Lex(clang::Token&)
9 932 747 898	15 598 586 152	0	cc1_main(llvm::ArrayRef<const char*>, const char*, void*)
9 932 747 898	15 598 586 152	0	__libc_start_main
9 932 747 898	15 598 586 152	0	main

clang::TokenLexer::PasteTokens(clang::Token&)

Instructions Exe... ATTRIBUTED #	clang::TokenLexer::Past is called by
15 486 538 749	*<Total>*
5 563 371 957	clang::TokenLexer::Lex

Instructions Ex... ATTRIBUTED #	clang::TokenLexer::PasteTokens(clang::Token&) calls
8 713 174 859	clang::SourceManager::getFileIDSlow(unsigned int)const
2 003 836 575	clang::Lexer::Lexer(clang::SourceLocation, const clang::LangO
1 869 404 117	clang::SourceManager::getImmediateExpansionRange(clang::Sourc
1 606 911 330	clang::Lexer::LexTokenInternal(clang::Token&, bool)
1 344 428 691	clang::Preprocessor::LookUpIdentifierInfo(clang::Token&)const
512 162 986	clang::Preprocessor::CreateString(llvm::StringRef, clang::Tok
512 162 560	clang::SourceManager::createExpansionLoc(clang::SourceLocatio
454 545 098	clang::SourceManager::getBufferData(clang::FileID, bool*)cons
281 689 760	*<static>@0x97a55 (<libc-2.19.so>)*
230 473 442	clang::Lexer::getSpelling(const clang::Token&, const char*&,
192 060 818	clang::Lexer::Lex(clang::Token&)
6 402 075	clang::TokenLexer::getExpansionLocForMacroDefLoc(clang::Sourc

Clank Memory Profiling



Class	Average Obje	TLABs	Total TLAB Si
🟥 byte[]	63.14 kB	1,005	5.87 GB
🟧 int[]	256.02 kB	489	3.02 GB
🟩 org.clang.basic.IdentifierInfo	32 bytes	92	369.17 MB
🟥 org.clank.support.aliases.char$ptr$array	24 bytes	39	235.28 MB
🟩 java.lang.Object[]	82.63 kB	23	134.01 MB
🟦 org.llvm.support.impl.MemoryBufferMen	40 bytes	10	63.59 MB
🟨 org.clang.basic.SrcMgr$ContentCache	40 bytes	10	60.23 MB
🟪 org.llvm.adt.ADTAliases$PointerInt2Pair	24 bytes	5	32.04 MB
🟩 org.llvm.adt.StringRef	24 bytes	4	25.15 MB

Stack Trace	TLABs	Total TLAB Si	Pressure

Clank: Performance analysis and optimizations in Java code

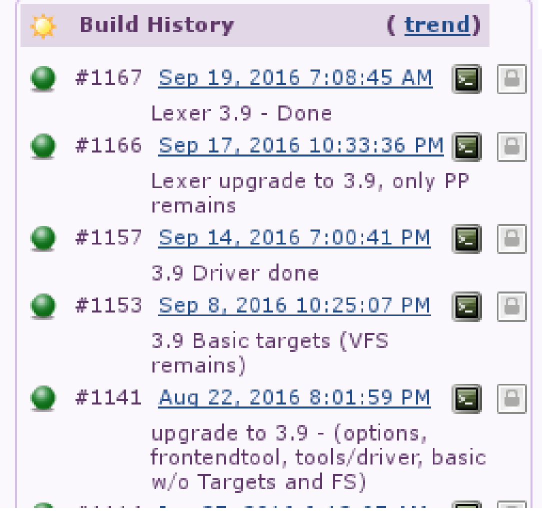
- Use Performance Analyzer to compare with Clang
  - PerfAn profiles Java or C++ using sampling with 2% overhead
  - Compare Instructions and CPU Cycles and do local perf optimizations
- Use Java Flight Recorder to profile memory footprint
- Teach Converter to produce more optimal code
- Use specializations based on parametrized spec files
  - Change template file, all specializations are regenerated
  - Add mapping to generate specializations, regenerate code

Clank: All is solvable

- ✓ Names collisions
  - Non-virtual methods in base and derived classes
    - In Java all methods are virtual
  - 'unsigned int' vs 'int' overloaded methods and constructors
- ✓ Diagnostics are not printed
  - Temporary objects lifecycle
- ✓ Multiple inheritance
- ✓ Compile time preprocessor-conditional code in FileSystem
  - Changed #ifdef/#else/#endif to runtime
- ✓ Split by TUs vs Monolithic Java classes
- ✓ this+1 and TrailingObjects
- ✓ Custom new operators
- ✓ JAVA Clank Preprocessor Performance

Clank: Upgrade to Clang 3.9

- Tooling
  - Analyze diffs
  - Analyze dependencies
  - Detect Changed Entities
  - Prepare TODO actions
  - Process Moved and Renamed actions first
  - Drive upgrade
  - Mark progress
  - Track progress



Build History (trend)

#1167 Sep 19, 2016 7:08:45 AM
Lexer 3.9 - Done

#1166 Sep 17, 2016 10:33:36 PM
Lexer upgrade to 3.9, only PP remains

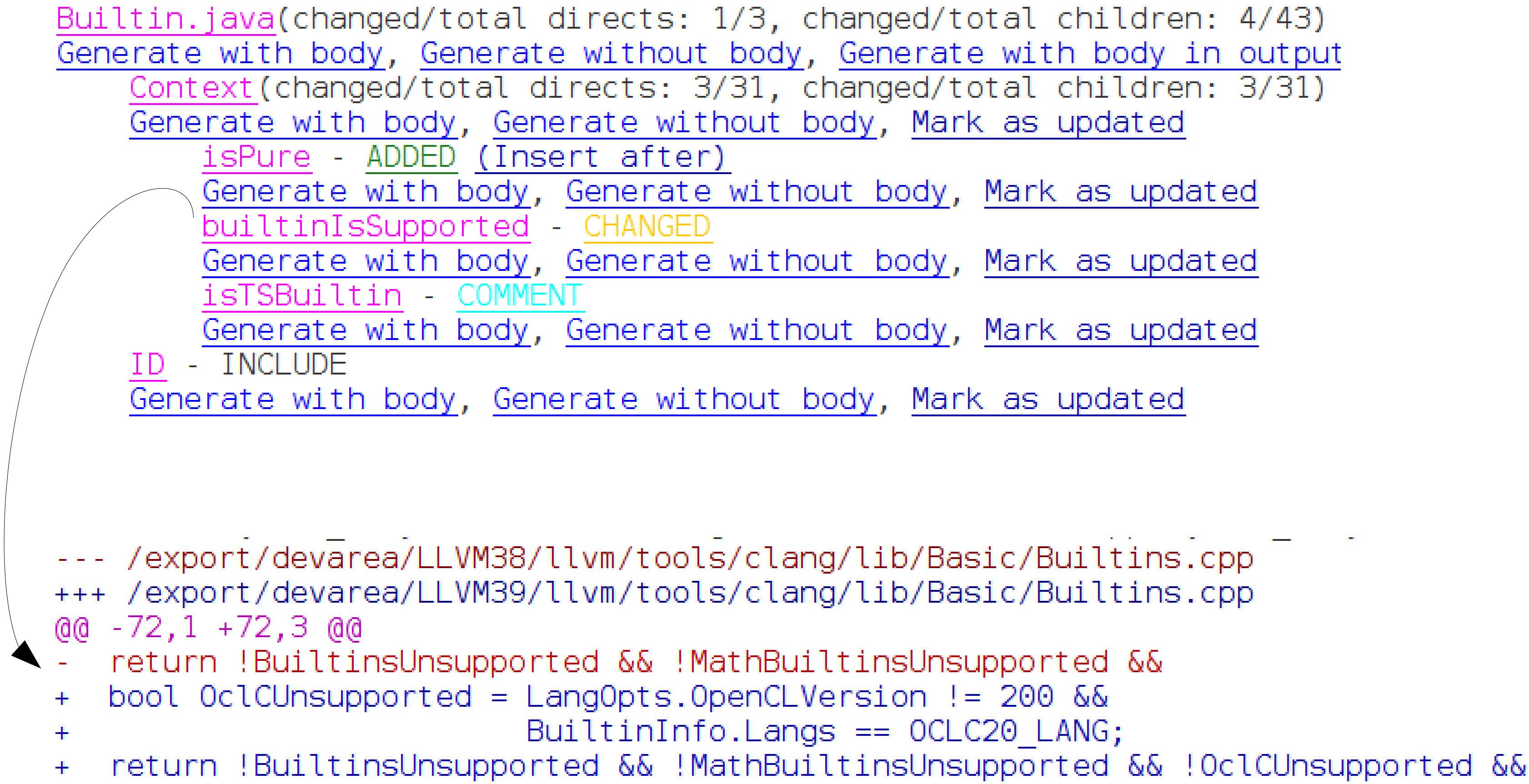
#1157 Sep 14, 2016 7:00:41 PM
3.9 Driver done

#1153 Sep 8, 2016 10:25:07 PM
3.9 Basic targets (VFS remains)

#1141 Aug 22, 2016 8:01:59 PM
upgrade to 3.9 - (options, frontendtool, tools/driver, basic w/o Targets and FS)

Clank: Upgrade to Clang 3.9

- Update view

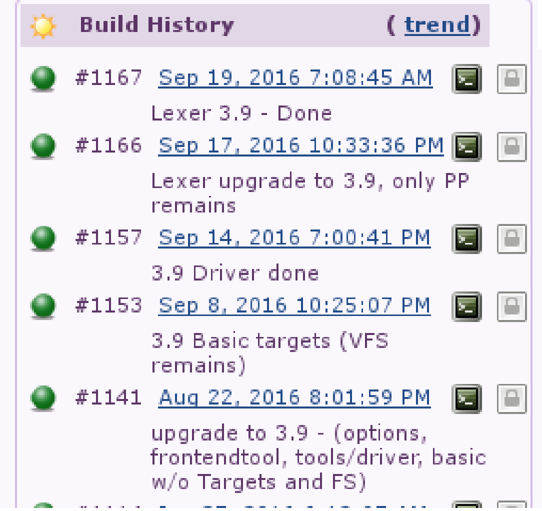


```
Builtin.java(changed/total directs: 1/3, changed/total children: 4/43)
Generate with body, Generate without body, Generate with body in output
    Context(changed/total directs: 3/31, changed/total children: 3/31)
    Generate with body, Generate without body, Mark as updated
        isPure - ADDED (Insert after)
        Generate with body, Generate without body, Mark as updated
        builtinIsSupported - CHANGED
        Generate with body, Generate without body, Mark as updated
        isTSBuiltin - COMMENT
        Generate with body, Generate without body, Mark as updated
    ID - INCLUDE
    Generate with body, Generate without body, Mark as updated
```

```
--- /export/devarea/LLVM38/llvm/tools/clang/lib/Basic/Builtins.cpp
+++ /export/devarea/LLVM39/llvm/tools/clang/lib/Basic/Builtins.cpp
@@ -72,1 +72,3 @@
-   return !BuiltinsUnsupported && !MathBuiltinsUnsupported &&
+   bool OclCUnsupported = LangOpts.OpenCLVersion != 200 &&
+                          BuiltinInfo.Langs == OCLC20_LANG;
+   return !BuiltinsUnsupported && !MathBuiltinsUnsupported && !OclCUnsupported &&
```

Clank: Upgrade to Clang 3.9

- Tooling
  - Analyze diffs
  - Analyze dependencies
  - Detect Changed Entities
  - Prepare TODO actions
  - Process Moved and Renamed actions first
  - Drive upgrade
  - Mark progress
  - Track progress
- 1 person – 4 weeks for 1.1MLoc
- Improve Upgrade Tools based on feedback


Build History (trend)

#1167 Sep 19, 2016 7:08:45 AM
Lexer 3.9 - Done

#1166 Sep 17, 2016 10:33:36 PM
Lexer upgrade to 3.9, only PP remains

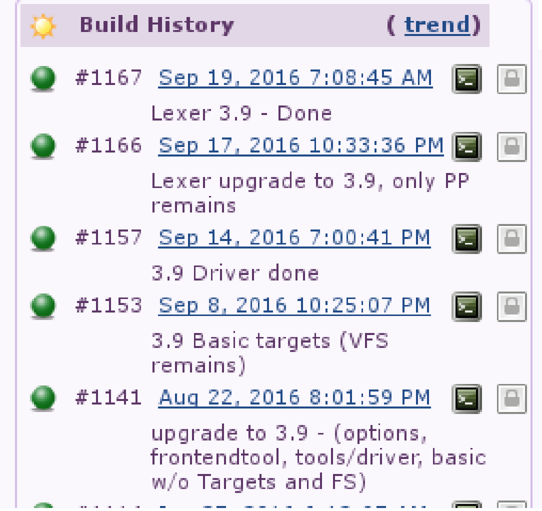
#1157 Sep 14, 2016 7:00:41 PM
3.9 Driver done

#1153 Sep 8, 2016 10:25:07 PM
3.9 Basic targets (VFS remains)

#1141 Aug 22, 2016 8:01:59 PM
upgrade to 3.9 - (options, frontendtool, tools/driver, basic w/o Targets and FS)

Clank: Upgrade to Clang 3.9

- Tooling
  - Analyze diffs
  - Analyze dependencies
  - Detect Changed Entities
  - Prepare TODO actions
  - Process Moved and Renamed actions first
  - Drive upgrade
  - Mark progress
  - Track progress
- 1 person – 4 weeks for 1.1MLoc
- Improve Upgrade Tools based on feedback



#1167 Sep 19, 2016 7:08:45 AM
Lexer 3.9 - Done

#1166 Sep 17, 2016 10:33:36 PM
Lexer upgrade to 3.9, only PP remains

#1157 Sep 14, 2016 7:00:41 PM
3.9 Driver done

#1153 Sep 8, 2016 10:25:07 PM
3.9 Basic targets (VFS remains)

#1141 Aug 22, 2016 8:01:59 PM
upgrade to 3.9 - (options, frontendtool, tools/driver, basic w/o Targets and FS)

Let's move toward complete C++ Frontend!

Agenda

- Why porting?
- Known approaches
- Converter
- Porting C++ and Clang challenges
- **Clank Demo**

Demo

Clank: Modular Structure

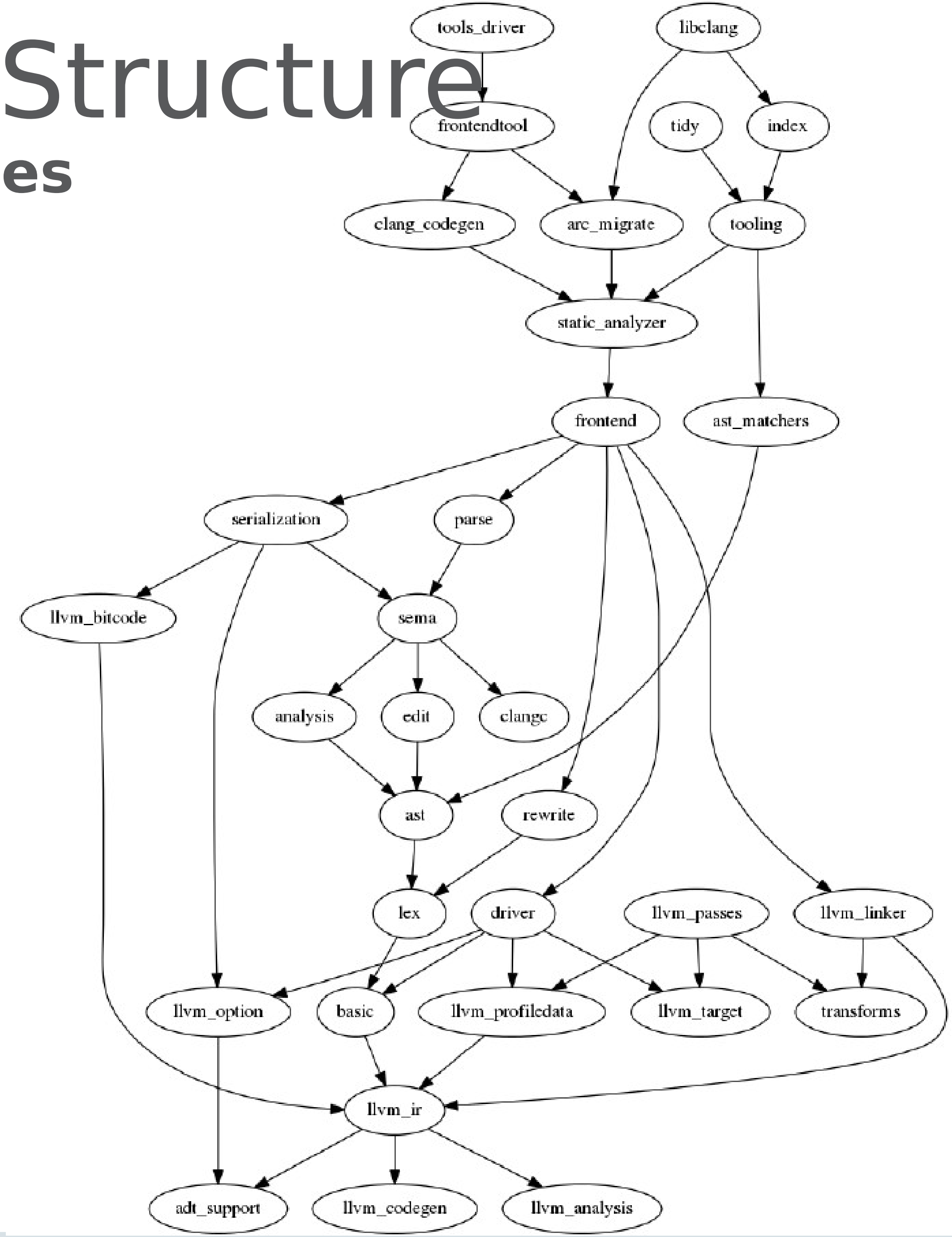
Java “LibC++”

- Memory and Pointers abstraction
- Unsigned types support
- Bit fields support
- STL Templates / Specializations
- I/O
- Function pointers
- @Converted annotation

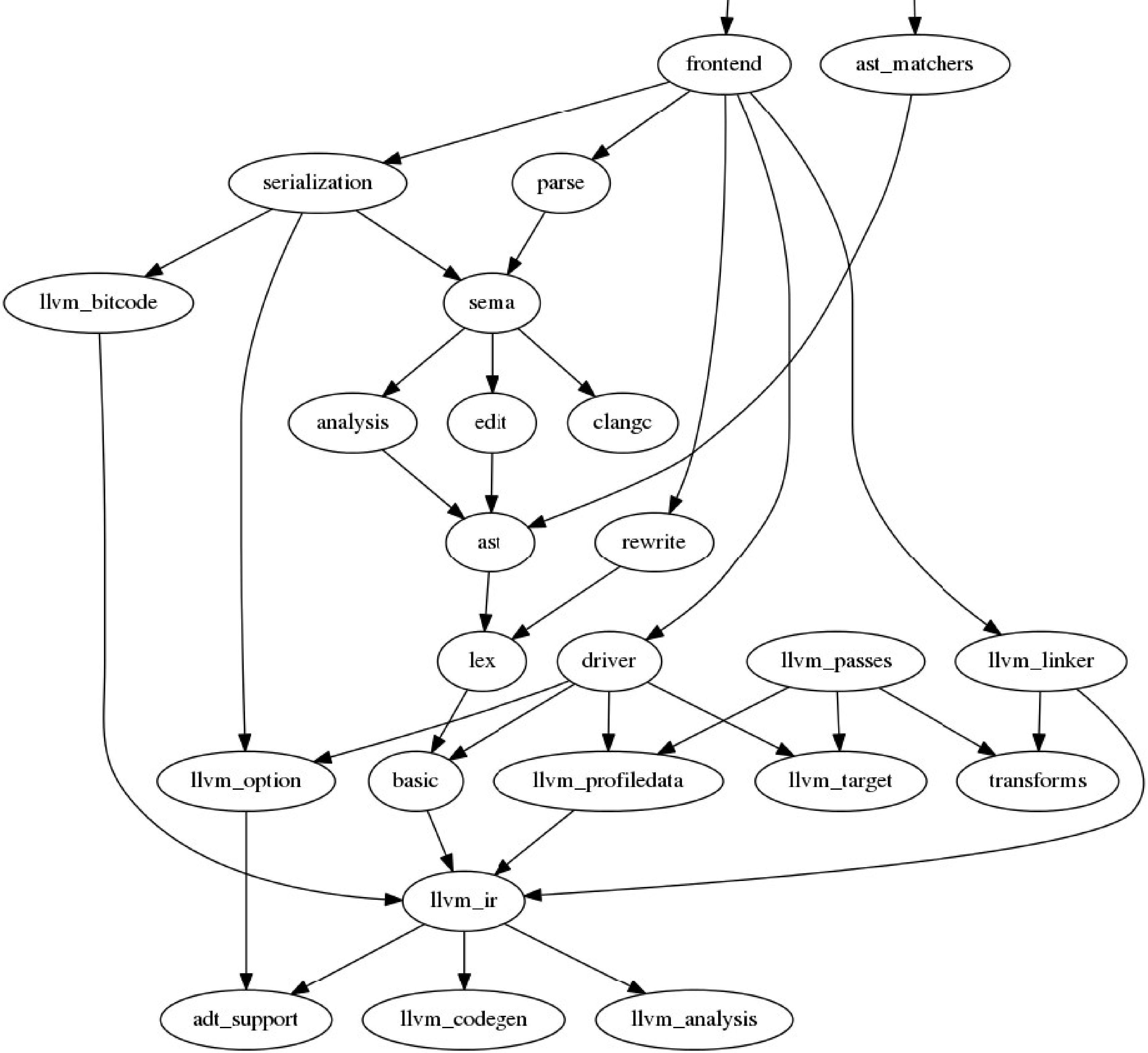
Clank: Modular Structure

Ported LLVM/Clang libraries

- Clang
  - Driver
  - Basic
  - Lex
  - AST
  - Analysis
  - Parse
  - Sema
  - Edit
  - Rewrite
  - Frontend
  - StaticAnalyzer
  - FrontendTool
  - Tools/Driver
- LLVM/Options
- ADT/Support On demand
  - with Templates and Specializations
- Tests: ADT/Support, Lexer, Preprocessor, Parser



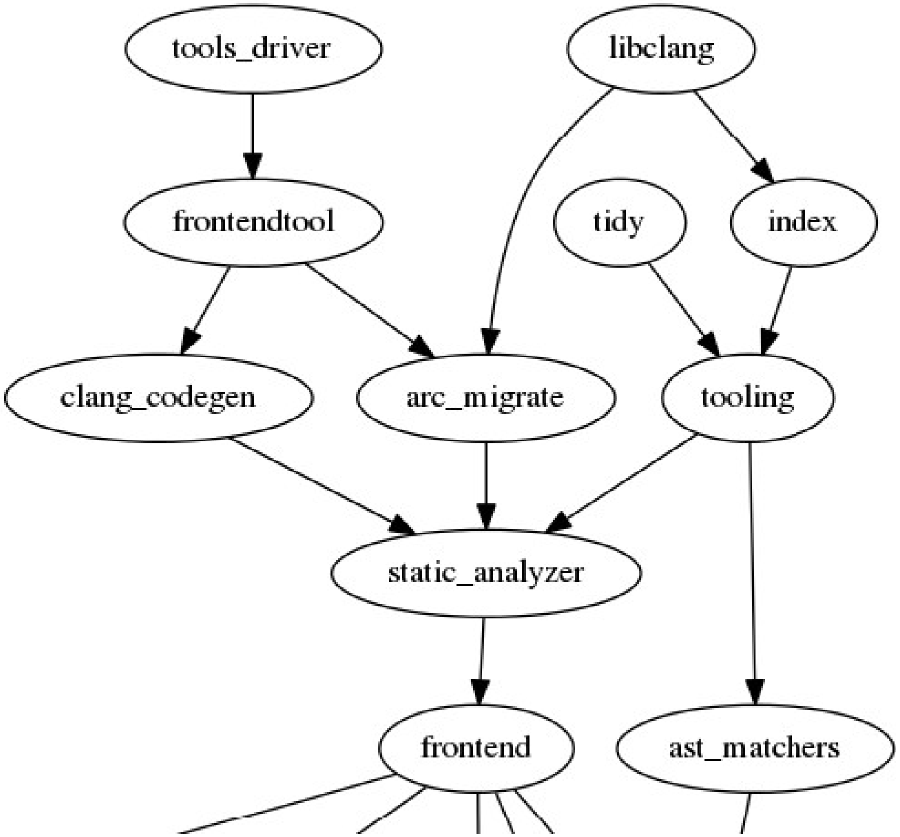
2 Million lines of code


frontend
ast_matchers
serialization
parse
llvm_bitcode
sema
analysis
edit
clangc
ast
rewrite
lex
driver
llvm_passes
llvm_linker
llvm_option
basic
llvm_profiledata
llvm_target
transforms
llvm_ir
adt_support
llvm_codegen
llvm_analysis

Clank: Modular Structure

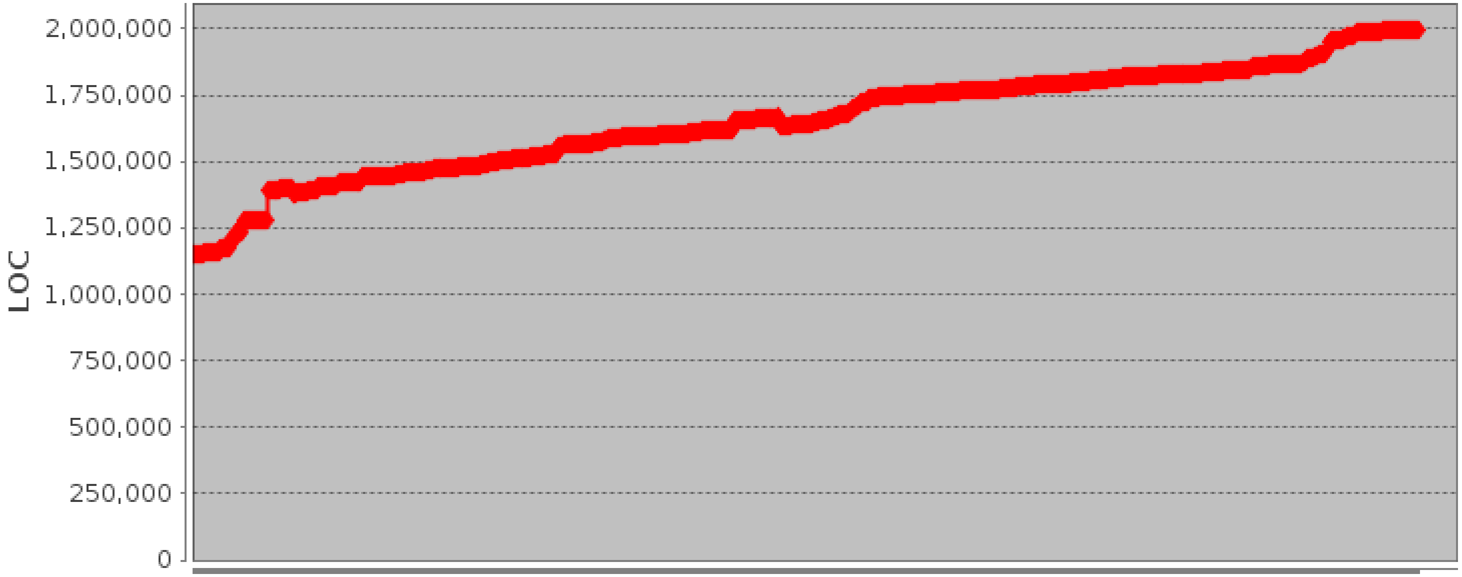
In-progress LLVM/Clang libraries

- Tooling
- ASTMatchers
- Serialization
- LLVM/Bitcode
- LLVM/IR



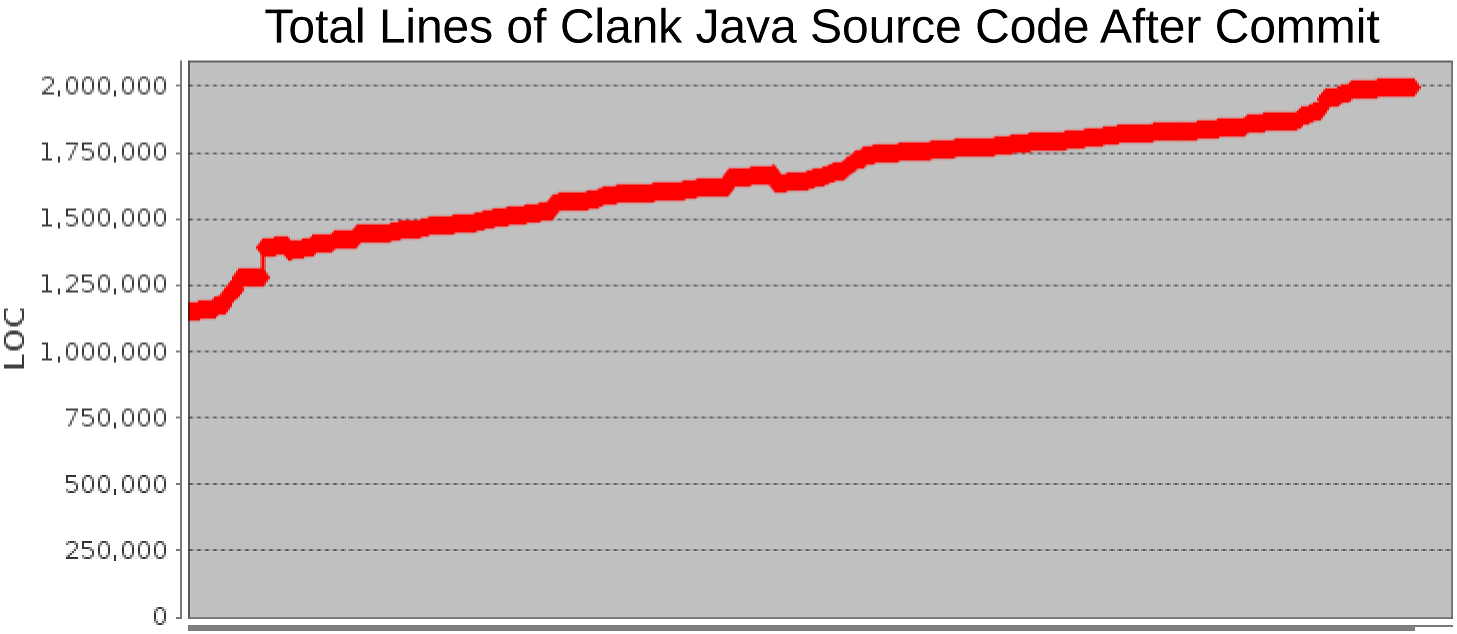
Clank: Porting progress

Total Lines of Clank Java Source Code After Commit



18 November 2016 ... 15 March 2017
2 persons: 4 months with long Russian NY break

Clank: Porting progress



18 November 2016 ... 15 March 2017
2 persons: 4 months with long Russian NY break

Improve Converter based on commits with
“MANUAL” keyword in subject
80% MANUALs are AUTO now

Clank

Thank you!

Clang

C++

Java™

Unite Developers Together