State of Clang
This talk in brief

• Clang as a compiler
• Applications of Clang libraries
• Clang C++ and future directions
Clang Goals and Ambitions

• C Language Family (C/C++/ObjC) Front-end Technology
  ▪ Parser + AST Generation Libraries
  ▪ Code generation through LLVM
  ▪ Infrastructure for source level tools

• Tools built from the libraries:
  ▪ GCC compatible compiler
  ▪ Static Analyzer
  ▪ Chris’ crazy automatic code review and correction tool?
  ▪ Your feature here :-)
Two Clang Releases

• Apple Xcode 3.2: “Apple Clang 1.0”
  ▪ Production quality C and ObjC support for Darwin X86
  ▪ Branched from mainline ~May’09

• LLVM 2.6: “Clang 2.6”
  ▪ New warnings, code generation improvements, many bug fixes
  ▪ The FreeBSD kernel and 99% of user space builds and works with clang!
  ▪ Branched from mainline ~Sep’09
User Visible Features over Mainline GCC

• Language feature support:
  ▪ Full Objective-C 2 and Apple “Blocks” Support
  ▪ Generic vector extensions (from OpenCL)
  ▪ Feature checking macros: #if __has_builtin(__builtin_unreachable)

• Better compile-time performance
• Better diagnostics

http://clang.llvm.org/docs/LanguageExtensions.html
Frontend Performance

• C and C++ are hostile to fast compile times:
  ▪ Many phases of translation: trigraphs, escaped newlines, macros
  ▪ Textual #include of (large) headers
  ▪ File system abuse searching for header files

• Compiler users and tools both want fast compiles:
  ▪ We compare debug “-O0 -g” compile times vs GCC 4.2
• Much faster PCH generation and syntax checking!
• Time pretty evenly split between phases

http://clang.llvm.org/performance.html
176.gcc Compile Time Breakdown

- Builtin assembler could be ~18% win!
- Heavily irgen and codegen (llvm) bound
  - Much left to do!

http://clang.llvm.org/performance.html
Error and Warning Improvements

• “Diagnostics” are error and warning reports
  ▪ Compiler detects something is wrong
  ▪ Tries to guess why and explain it

• GCC diagnostics are often not helpful
  ▪ Confusing, poorly worded, not precise

http://clang.llvm.org/diagnostics.html
Range and Location Information

GCC 4.2

$ gcc t.c
t.c: In function ‘foo’:
t.c:8: error: invalid operands to binary + (have ‘int’ and ‘struct A’)  

Clang

$ clang t.c
t.c:8:36: error: invalid operands to binary expression ('int' and 'struct A')
X = X + func(X ? ((SomeA.F + 40) + SomeA) / 42 + SomeA.F : Ptr->F);
~~~~~~~~~~~~~~ ^ ~~~~~

http://clang.llvm.org/diagnostics.html
Better Diagnosis of the Problem

GCC 4.2

$ gcc t.c
t.c:2: error: expected ‘=’, ‘,’ or ‘;’ before ‘P’

Clang

$ clang t.c
t.c:2:1: error: unknown type name 'foo_t'
foo_t P = 42;
^

http://clang.llvm.org/diagnostics.html
Macro Expansion Information

GCC 4.2

$ gcc t.c
  t.c: In function ‘foo’:
  t.c:9: error: invalid operands to binary > (have ‘int’ and ‘struct A’)

Clang

$ clang t.c
  t.c:9:7: error: invalid operands to binary expression ('int' and 'struct A')
    X = MAX(X, *Ptr);
       ^~~~~~~~~~~
  t.c:2:24: note: instantiated from:
#define MAX(A, B) ((A) > (B) ? (A) : (B))
       ^~~~

http://clang.llvm.org/diagnostics.html
Other Great Refinements

$ clang t.c

$ clang t.c

• Diagnostics tell you which -W flag controls them
• Fixit hints for obvious fixes (and -fixit mode that applies them)
• Diagnostics really are color coded on the command line

http://clang.llvm.org/diagnostics.html
Summary: Clang as a Compiler

• Clang is a great compiler to use:
  ▪ Ridiculously fast
  ▪ Great “user interface”
  ▪ Useful language extensions

• Not quite done yet:
  ▪ Missing warnings: e.g. 64-bit portability warnings
  ▪ Support for every crazy target triple
  ▪ Fully featured cross compiler support
  ▪ C++!

• Clang is a super hackable compiler front-end, come help!
Clang Applications
OpenCL

- Language and framework for general purpose use of GPUs and CPUs
- Use Clang and LLVM to JIT compile “C” code
Clang Static Analyzer

- Standalone tool for finding bugs by analyzing source code
- Find deeper bugs than compiler warnings
- Memory leaks, logic errors, API violations, many others

http://clang-analyzer.llvm.org
What we showed you last year ....

$ scan-build <build command>
What we showed you last year...

```bash
scan-build <build command>
```

---

### Bug Types and Reports

<table>
<thead>
<tr>
<th>Bug Type</th>
<th>Quantity</th>
<th>Display?</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Bugs</td>
<td>43</td>
<td>✔️</td>
</tr>
<tr>
<td>Dead store</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dead assignment</td>
<td>26</td>
<td>✔️</td>
</tr>
<tr>
<td>Dead increment</td>
<td>2</td>
<td>✔️</td>
</tr>
<tr>
<td>Dead nested assignment</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>Logic errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Null dereference</td>
<td>11</td>
<td>✔️</td>
</tr>
<tr>
<td>Undefined or garbage result</td>
<td>2</td>
<td>✔️</td>
</tr>
<tr>
<td>Memory (Core Foundation/Objective-C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leak of returned object</td>
<td>1</td>
<td>✔️</td>
</tr>
</tbody>
</table>

#### Reports

<table>
<thead>
<tr>
<th>Bug Group</th>
<th>Bug Type</th>
<th>File</th>
<th>Line</th>
<th>Path Length</th>
<th>View Report</th>
<th>Report Bug</th>
<th>Open File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic errors</td>
<td>Undefined or garbage result</td>
<td>audio/SDL_wave.c</td>
<td>563</td>
<td>8</td>
<td><a href="#">View Report</a></td>
<td><a href="#">Report Bug</a></td>
<td><a href="#">Open File</a></td>
</tr>
<tr>
<td>Logic errors</td>
<td>Undefined or garbage result</td>
<td>events/SDL_mouse.c</td>
<td>62</td>
<td>2</td>
<td><a href="#">View Report</a></td>
<td><a href="#">Report Bug</a></td>
<td><a href="#">Open File</a></td>
</tr>
<tr>
<td>Logic errors</td>
<td>Null dereference</td>
<td>video/SDL_surface.c</td>
<td>95</td>
<td>8</td>
<td><a href="#">View Report</a></td>
<td><a href="#">Report Bug</a></td>
<td><a href="#">Open File</a></td>
</tr>
<tr>
<td>Logic errors</td>
<td>Null dereference</td>
<td>joystick/SDL_joystick.c</td>
<td>146</td>
<td>11</td>
<td><a href="#">View Report</a></td>
<td><a href="#">Report Bug</a></td>
<td><a href="#">Open File</a></td>
</tr>
</tbody>
</table>
WARNING: This is the only function protected this way!

```c
if (!current_video) {
    if (SDL_Init((SDL_INIT_VIDEO|SDL_INIT_NOPARACHUTE) != 0)) {
        return(NULL);
    }
    this = video = current_video;
    /* Default to the current width and height */
    if (width == 0) {
        width = video->info.current_w;
    }
    if (height == 0) {
        height = video->info.current_h;
    }
    /* Default to the current video bpp */
}```
Building Blocks for Source-level Technologies

- **libAnalysis** (Static analyzer)
  - Control-flow graphs
  - Path-sensitive analysis engine
  - Now used for some Clang warnings
- **libRewrite** (Syntactic code editing)
  - Used by Fixit-hints
- **libDriver & libFrontend**
- **libIndex**
  - Uses serialized ASTs
  - Cross translation unit symbol resolution
Other Potential Applications

- **Refactoring**
  - Many pieces in place
  - libRewrite (code rewriting)
  - libIndex (cross-translation unit symbol resolution)

- **Documentation generation**

- **Advanced code search**
  - Within a codebase
  - Multiple codebases
    - Across an organization
    - Sourceforge.net

- **Advanced revision browsing**
  - Examine semantic changes
  - What is the impact of this change?

- **Language bindings (Scripting)**

- **Incremental Parsing**

- **Intelligent Code Formatting**

- **C++ Interpreter**

- Many others!
C++ Is a Large, Complex Language

- Classes
  - Derived classes
  - Multiple, virtual inheritance
  - Constructors, destructors
  - Virtual functions
  - Friends
- Namespaces
  - Argument-Dependent Lookup
  - Using directives
  - Using declarations
- Overload resolution
  - Operator overloading
  - User-defined conversions
- Templates
  - Class & function templates
  - Partial specialization
  - Template instantiation
  - Member templates
  - Template argument deduction/SFINAE

http://clang.llvm.org/cxx_status.html
C++ Is a Large, Complex Language

- Classes
  - Derived classes
  - Multiple, virtual inheritance
  - Constructors, destructors
  - Virtual functions
  - Friends
- Namespaces
  - Argument-Dependent Lookup
  - Using directives
  - Using declarations

- Overload resolution
  - Operator overloading
  - User-defined conversions
  - Templates
  - Class & function templates
  - Partial specialization
  - Template instantiation
  - Member templates
  - Template argument deduction/SFINAE

http://clang.llvm.org/cxx_status.html
Clang C++ Parsing: In the Real World

• Clang can parse real C++ code:
  ▪ C++ Standard Library headers (GNU libstdc++ 4.2)
  ▪ XNU Kernel
  ▪ 105/140 QtCore headers
  ▪ Various LLVM headers
• LLVM IR generation is starting to crawl:

```cpp
#include <string>
#include "llvm/Support/raw_ostream.h"
int main() {
  std::string Hello = "Hello";
  llvm::outs() << (Hello + ", " + "World!") << '\n';
}
```
Clang C++ Parsing: In the Real World

• Clang can parse real C++ code:
  ▪ C++ Standard Library headers (GNU libstdc++ 4.2)
  ▪ XNU Kernel
  ▪ 105/140 QtCore headers
  ▪ Various LLVM headers
• LLVM IR generation is starting to crawl:

```cpp
#include <string>
#include "llvm/Support/raw_ostream.h"
int main() {
  std::string Hello = "Hello";
  llvm::outs() << (Hello + ", " + "World!") << ‘\n’;
}
```
Clang C++ Parsing: In the Real World

- Clang can parse real C++ code:
  - C++ Standard Library headers (GNU libstdc++ 4.2)
  - XNU Kernel
  - 105/140 QtCore headers
  - Various LLVM headers
- LLVM IR generation is starting to crawl:

```cpp
#include <string>
#include "llvm/Support/raw_ostream.h"
int main() {
    std::string Hello = "Hello";
    llvm::outs() << (Hello + ", " + "World!") << '\n';
}
```
Performance: Parsing libstdc++ Headers

Parsing Time (64-bit)

- GCC 4.2: 0.503s
- Clang: 0.419s

Memory Consumed (64-bit)

- GCC 4.2: 55649kB
- Clang: 24576kB
Better Diagnostics Now: Ambiguities

• GCC 4.2:
  
  virtual-ambig.cpp:9: error: invalid covariant return type for ‘virtual TeachingAssistant* TeachingAssistant::Clone() const’
  virtual-ambig.cpp:3: error: overriding ‘virtual Person* Person::Clone() const’

• Clang:
  
  virtual-ambig.cpp:9:30: error: return type of virtual function 'Clone' is not covariant with the return type of the function it overrides (ambiguous conversion from derived class 'class TeachingAssistant' to base class 'class Person':

  class TeachingAssistant -> class Teacher -> class Person
  class TeachingAssistant -> class Student -> class Person)
  virtual TeachingAssistant *Clone() const;

  virtual-ambig.cpp:3:19: note: overridden virtual function is here
  virtual Person *Clone() const;
Better Diagnostics Now: Ambiguities

- GCC 4.2:
  virtual-ambig.cpp:9: error: invalid covariant return type for ‘virtual TeachingAssistant*
  TeachingAssistant::Clone() const'
  virtual-ambig.cpp:3: error: overriding ‘virtual Person* Person::Clone() const’

- Clang:
  virtual-ambig.cpp:9:30: error: return type of virtual function 'Clone' is not
covariant with the return type of the function it overrides (ambiguous
conversion from derived class 'class TeachingAssistant' to base class
'class Person':

  class TeachingAssistant -> class Teacher -> class Person
  class TeachingAssistant -> class Student -> class Person)

  virtual TeachingAssistant *Clone() const;

  virtual-ambig.cpp:3:19: note: overridden virtual function is here
  virtual Person *Clone() const;
Better Diagnostics Now: Overloading

```cpp
string s = getData();
std::ofstream("file.txt") << s << std::endl;
```

- GCC 4.2:
  ```
os.cpp:8: error: no match for 'operator<<' in

  'std::basic_ofstream<char, std::char_traits<char> >(((const char*)"file.txt"),
  std::operator|(_S_out, _S_trunc)) << s'

  /usr/include/c++/4.2.1/ostream:112: note: candidates are: std::basic_ostream<_CharT,
  _Traits>& std::basic_ostream<_CharT, _Traits>::operator<<(std::basic_ostream<_CharT,
  _Traits>& (*)(std::basic_ostream<_CharT, _Traits>&)) [with _CharT = char, _Traits =
  std::char_traits<char>]
  /usr/include/c++/4.2.1/ostream:121: note:                 std::basic_ostream<_CharT,
  _Traits>& std::basic_ostream<_CharT, _Traits>::operator<<(std::basic_ios<_CharT,
  _Traits>& (*)(std::basic_ios<_CharT, _Traits>&)) [with _CharT = char, _Traits =
  std::char_traits<char>]
  /usr/include/c++/4.2.1/bits/basic_string.h:2410: note: std::basic_ostream<_CharT,
  _Traits>& std::basic_ostream<_CharT, _Traits>::operator<(std::basic_ostream<_CharT,
  _Traits>&, const std::basic_string<_CharT, _Traits, _Alloc>&) [with _CharT = char,
  _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]
  ```
Better Diagnostics Now: Overloading

```cpp
string s = getData();
std::ofstream("file.txt") << s << std::endl;
```

- GCC 4.2:
  os.cpp:8: error: no match for `operator<<' in
`std::basic_ofstream<char, std::char_traits<char>> >((const char*)"file.txt"),
std::operator|(_S_out, _S_trunc)) << s'
/usr/include/c++/4.2.1/ostream:112: note: candidates are: std::basic_ostream<_CharT,_Traits>& std::basic_ostream<_CharT,_Traits>::operator<<(std::basic_ostream<_CharT,_Traits>& (*)(std::basic_ostream<_CharT,_Traits>&&)) [with _CharT = char,_Traits = std::char_traits<char>]
/usr/include/c++/4.2.1/ostream:121: note:                 std::basic_ostream<_CharT,_Traits>& std::basic_ostream<_CharT,_Traits>::operator<<(std::basic_ios<_CharT,_Traits>& (*)(std::basic_ios<_CharT,_Traits>&&)) [with _CharT = char,_Traits = std::char_traits<char>]
/usr/include/c++/4.2.1/bits/basic_string.h:2410: note: std::basic_ostream<_CharT,_Traits>& std::operator<<(std::basic_ostream<_CharT,_Traits>&, const std::basic_string<_CharT,_Traits,_Alloc>&) [with _CharT = char,_Traits = std::char_traits<char>,_Alloc = std::allocator<char>]
```
Better Diagnostics Now: Overloading

- Clang:
  os.cpp:8:15: **error**: invalid operands to binary expression ('std::ofstream' (aka 'class std::basic_ofstream<char, struct std::char_traits<char> >')) and 'string' (aka 'class std::basic_string<char, struct std::char_traits<char>, class std::allocator<char> >'))

```cpp
std::ofstream("file.txt") << s << std::endl;
```

In file included from os.cpp:1:
In file included from /usr/include/c++/4.2.1/string:53:
/usr/include/c++/4.2.1/bits/basic_string.h:2408:5: **note**: candidate function template specialization [with _CharT = char, _Traits = struct std::char_traits<char>, _Alloc = class std::allocator<char>]
  operator<<(basic_ostream<_CharT, _Traits>& __os, ^

/usr/include/c++/4.2.1/ostream:112:35: **note**: candidate function
  operator<<(__ostream_type& (*__pf)(__ostream_type&)) ^

/usr/include/c++/4.2.1/ostream:121:31: **note**: candidate function
  operator<<(__ios_type& (*__pf)(__ios_type&)) ^
Better Diagnostics Now: Overloading

- Clang:

  os.cpp:8:15: **error:** invalid operands to binary expression (`std::ofstream` (aka 'class std::basic_ofstream<char, struct std::char_traits<char> >') and 'string' (aka 'class std::basic_string<char, struct std::char_traits<char>, class std::allocator<char> >'))

  ```
  std::ofstream("file.txt") << s << std::endl;
  ~~~~~~~~~~~~~~ ^  ~
  ```

  In file included from os.cpp:1:

  In file included from /usr/include/c++/4.2.1/string:53:

  `/usr/include/c++/4.2.1/bits/basic_string.h:2408:5: **note:** candidate function template specialization [with _CharT = char, _Traits = struct std::char_traits<char>, _Alloc = class std::allocator<char>]

  `operator<<(basic_ostream<_CharT, _Traits>& __os,`

  `/usr/include/c++/4.2.1/ostream:112:35: **note:** candidate function

  `operator<<(__ostream_type& (*__pf)(__ostream_type&))`

  `/usr/include/c++/4.2.1/ostream:121:31: **note:** candidate function

  `operator<<(__ios_type& (*__pf)(__ios_type&))`
Better Diagnostics Now: Overloading

• Clang:

    os.cpp:8:15: error: invalid operands to binary expression ('std::ofstream' (aka 'class std::basic_ofstream<char, struct std::char_traits<char> >') and 'string' (aka 'class std::basic_string<char, struct std::char_traits<char>, class std::allocator<char> >'))
        std::ofstream("file.txt") << s << std::endl;
                        ^  ~
        In file included from os.cpp:1:
        In file included from /usr/include/c++/4.2.1/string:53:
        /usr/include/c++/4.2.1/bits/basic_string.h:2408:5: note: candidate function template specialization
                  [with _CharT = char, _Traits = struct std::char_traits<char>, _Alloc = class std::allocator<char>]
                  operator<<(basic_ostream<_CharT, _Traits>& __os, ^
        /usr/include/c++/4.2.1/ostream:112:35: note: candidate function
                  operator<<(__ostream_type& (*__pf)(__ostream_type&)) ^
        /usr/include/c++/4.2.1/ostream:121:31: note: candidate function
                  operator<<(__ios_type& (*__pf)(__ios_type&))
        ~
Better Diagnostics Later: Overloading

os.cpp:8:15: error: invalid operands to binary << (have ‘std::ostream’ (aka ‘std::basic ofstream< char>’) and ‘std::string’ (aka ‘std::basic string< char>’))
    std::ofstream("file.txt") << s << std::endl;
    ~~~~~~~~~~~~~~~~~~~~~~~~~^  ~
/usr/include/c++/4.2.1/bits/basic_string.h:2410: note: cannot initialize a non-const reference with a temporary of type ‘std::ostream’ (aka ‘basic ofstream< char>’)
    operator<<(basic_ostream<_CharT, _Traits>& __os,
    ~
/usr/include/c++/4.2.1/ostream:112:35: note: cannot initialize parameter with lvalue of type ‘std::string’ (aka ‘std::basic string< char>’)}
    operator<<(__ostream_type& (*__pf)(__ostream_type&))
    ~
/usr/include/c++/4.2.1/ostream:121:31: note: cannot initialize parameter with lvalue of type ‘std::string’ (aka ‘std::basic string< char>’)}
    operator<<(__ios_type& (*__pf)(__ios_type&))
    ~
Better Diagnostics Later: Overloading

```cpp
os.cpp:8:15: error: invalid operands to binary << (have `std::ofstream' (aka `std::basic_ofstream<char>')) and `std::string' (aka `std::basic_string<char>'))
    std::ofstream("file.txt") << s << std::endl;
            ^  ~
/usr/include/c++/4.2.1/bits/basic_string.h:2410: note: cannot initialize a non-const reference with a temporary of type `std::ofstream' (aka `basic_ofstream<char>')</
    operator<<(basic_ostream<_CharT, _Traits>& __os,
            ^
/usr/include/c++/4.2.1/ostream:112:35: note: cannot initialize parameter with lvalue of type `std::string' (aka `std::basic_string<char>')</
    operator<<(__ostream_type& (*__pf)(__ostream_type&))
            ^
/usr/include/c++/4.2.1/ostream:121:31: note: cannot initialize parameter with lvalue of type `std::string' (aka `std::basic_string<char>')</
    operator<<(__ios_type& (*__pf)(__ios_type&))
            ^
```
Better Diagnostics Later: Overloading

os.cpp:8:15: error: invalid operands to binary << (have ‘std::ofstream’ (aka ‘std::basic_ofstream<char>’) and ‘std::string’ (aka ‘std::basic_string<char>’))
    std::ofstream("file.txt") << s << std::endl;

/usr/include/c++/4.2.1/bits/basic_string.h:2410: note: cannot initialize a non-const reference with a temporary of type `std::ofstream’ (aka `basic_ofstream<char>’)
    operator<<(basic_ostream<_CharT, _Traits>& __os,

/usr/include/c++/4.2.1/ostream:112:35: note: cannot initialize parameter with lvalue of type `std::string` (aka `std::basic_string<char>’)
    operator<<(__ostream_type& (*__pf)(__ostream_type&))

/usr/include/c++/4.2.1/ostream:121:31: note: cannot initialize parameter with lvalue of type `std::string` (aka `std::basic_string<char>’)
    operator<<(__ios_type& (*__pf)(__ios_type&))

```
Better Diagnostics Later: Overloading

os.cpp:8:15: **error**: invalid operands to binary << (have `std::ofstream` (aka `std::basic_ofstream<char>`) and `std::string` (aka `std::basic_string<char>`))
  std::ofstream("file.txt") << s << std::endl;
    ^ ~
/usr/include/c++/4.2.1/bits/basic_string.h:2410: **note**: cannot initialize a non-const reference with a temporary of type `std::ofstream` (aka `basic_ofstream<char>`)  
      operator<<(basic_ostream<_CharT, _Traits>& __os, 
                   ~

**clang -fshow-minimal-overload-candidates**  
(the default)
C++ Future

- We don’t know when C++ will be done, but we’re moving **fast**.
- C++ Standard Library:
  - GNU libstdc++ support is critical
  - Apache, Dinkumware, STLport should work
- C++’0x support:
  - Clang C++ is designed with C++’0x in mind
  - C++’98 support comes first
- C++ Static Analysis:
  - Existing analyses should work on C++ code
  - Extend for C++ idioms and abstractions (RAII, iterators)
the end.

- Clang web page: http://clang.llvm.org
- Clang C++ status page: http://clang.llvm.org/cxx_status.html
- Clang Static Analyzer page: http://clang-analyzer.llvm.org
- Clang developer mailing list: cfe-dev@cs.uiuc.edu