# LLDB
# Modular
# Debugging
# Infrastructure

# LLDB Session Overview

- Why?
- Introduction
- Integration
- Status
- Future

# Why LLDB?

- Speed
- Efficiency
- Accuracy
- Extensibility
- Reusability

# Speed


- Architected for multi-core
  - Multi-threaded
  - Object oriented
- Leverage performant LLVM classes
  - DenseMap
  - StringMap
    - Constant string pool
      - Type names
      - Function names
      - File paths

# Efficiency

- Minimize memory footprint
  - Lazy and partial parsing
    - Object Files
    - Symbol Files
  - Share memory resources
    - String pool for constant strings
    - Share common object and symbol files
      - Share indexes
      - Share parsed information

# Accuracy

- Improved ability set breakpoints
  - Breakpoints are symbolic
    - File and line
    - Name
      - Fullname
      - Basename
      - C++ Method
      - Selector
      - Regular Expressions
- Improved expression parsing

# Extensibility

lldb

Xcode 4

Python

LLDB.framework

LLDB Core

| Process | Dynamic Loader | Object FIles | Object Containers | Symbols | Disassembly |
|---|---|---|---|---|---|
| Mac OS X | Darwin | Mach-O | Universal | DWARF | LLVM |
| GDB Remote | | ELF | BSD Archive | ObjectFile | |

# Resuability

- Python support
  - Full access to the LLDB API
- Python access
  - lldb
    - “script” command
    - “breakpoint command add” command
  - python

```
(lldb) script
Python Interactive Interpreter. To exit type 'quit()', 'exit()' or Ctrl-D.
>>> import lldb
```

# Introduction

## GDB

```
% gdb a.out
(gdb) break main
Breakpoint 1 at 0x100000f33: file main.c, line 4
(gdb) run
```

## LLDB

```
% lldb a.out
(lldb) breakpoint set --name main
Breakpoint created: 1: name = 'main', locations = 1
(lldb) process launch
```

# Introduction

## GDB

```
(gdb) info args
argc = 1
argv = (const char **) 0x7fff5fbff550
(gdb) info locals
i = 32767
```

## LLDB

```
(lldb) frame variable
argc = 1
argv = 0x00007fff5fbfff68
i = 0
```

# Variables in LLDB

LLDB

```
(lldb) frame variable argv
argv = 0x00007fff5fbffe80
(lldb) frame variable *argv
argv = 0x00007fff5fbfff68
  *argv = 0x00007fff5fbfffa8 "/private/tmp/a.out"
(lldb) frame variable argv[0]
argv[0] = 0x00007fff5fbffef0 "/tmp/a.out"
(lldb) frame variable rect_ptr->bottom_left.x
rect_ptr->bottom_left.x = 1
```

# Expression in LLDB

LLDB

```
(lldb) expression x + y->getCount()
(int) $0 = 2
(lldb) expression pt
(struct point_tag) $1 = {
  (int) x = 2
  (int) y = 3
}
(lldb) expression $1.x
(int) $2 = 2
```

# LLDB Command Syntax

Command Syntax

<noun> <verb> [-options [option-value]] [argument [argument...]]

Uses standard getopt_long() for predictable behavior

(lldb) process launch a.out --stop-at-entry

(lldb) process launch a.out -- --arg0 --arg1

(lldb) process launch a.out -st

Options know which other options they are compatible with

(lldb) process attach --pid 123 --name a.out

# Help Command

(lldb) help frame variable

Show frame variables. All argument and local variables that are in scope will be shown when no arguments are given. If any arguments are specified, they can be names of argument, local, file static and file global variables. Children of aggregate variables can be specified such as 'var->child.x'.

Syntax: frame variable <cmd-options> [<variable-name> [<variable-name> [...]]]

Command Options Usage:

frame variable [-acfglorstyDL] [-d <count>] [-G <variable-name>] [-p <count>] [<variable-name> [<variable-name> [...]]]

-D ( --debug )

Enable verbose debug information.

-G <variable-name> ( --find-global <variable-name> )

# Apropos Command

```
(lldb) apropos thread
The following commands may relate to 'thread':
breakpoint modify -- Modify the options on a breakpoint or set of breakpoints…
breakpoint set    -- Sets a breakpoint or set of breakpoints in the executable.
frame             -- A set of commands for operating on the current thread's…
frame info        -- List information about the currently selected frame in the…
frame select      -- Select a frame by index from within the current thread…
log enable        -- Enable logging for a single log channel.
process continue  -- Continue execution of all threads in the current process.
register          -- A set of commands to access thread registers.
thread            -- A set of commands for operating on one or more…
thread backtrace  -- Show the stack for one or more threads.  If no threads are…
thread continue   -- Continue execution of one or more threads in an active…
thread list       -- Show a summary of all current threads in a process.
thread select     -- Select a thread as the currently active thread.
thread step-in    -- Source level single step in specified thread (current…
thread step-inst  -- Single step one instruction in specified thread (current…
```

# Common Commands

| GDB | LLDB |
| --- | --- |
| (gdb) ^C | (lldb) process interrupt |
| (gdb) signal 2 | (lldb) process signal SIGINT |
| (gdb) info break | (lldb) breakpoint list |
| (gdb) continue | (lldb) process continue |
| (gdb) step | (lldb) thread step-in |
| (gdb) stepi | (lldb) thread step-inst |
| (gdb) next | (lldb) thread step-over |
| (gdb) nexti | (lldb) thread step-over-inst |
| (gdb) finish | (lldb) thread step-out |
| (gdb) info threads | (lldb) thread list |
| (gdb) backtrace | (lldb) thread backtrace |

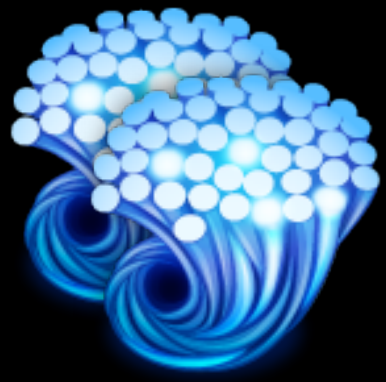
# Common Commands

| GDB | LLDB |
| --- | --- |
| (gdb) ^C | (lldb) pro int |
| (gdb) signal 2 | (lldb) pro s SIGINT |
| (gdb) in br | (lldb) br l |
| (gdb) c | (lldb) c |
| (gdb) s | (lldb) s |
| (gdb) si | (lldb) si |
| (gdb) n | (lldb) n |
| (gdb) ni | (lldb) ni |
| (gdb) f | (lldb) f |
| (gdb) info threads | (lldb) th l |
| (gdb) bt | (lldb) bt |

# Multi-Threaded Debugging

## Better multi-threaded experience

- Per-thread state
- Per-thread runtime control
  - Step, resume or suspend
  - Control actions
  - Control actions are stackable



Step over main.cpp:12

Breakpoint at bar.cpp:234

Step over bar.cpp:235

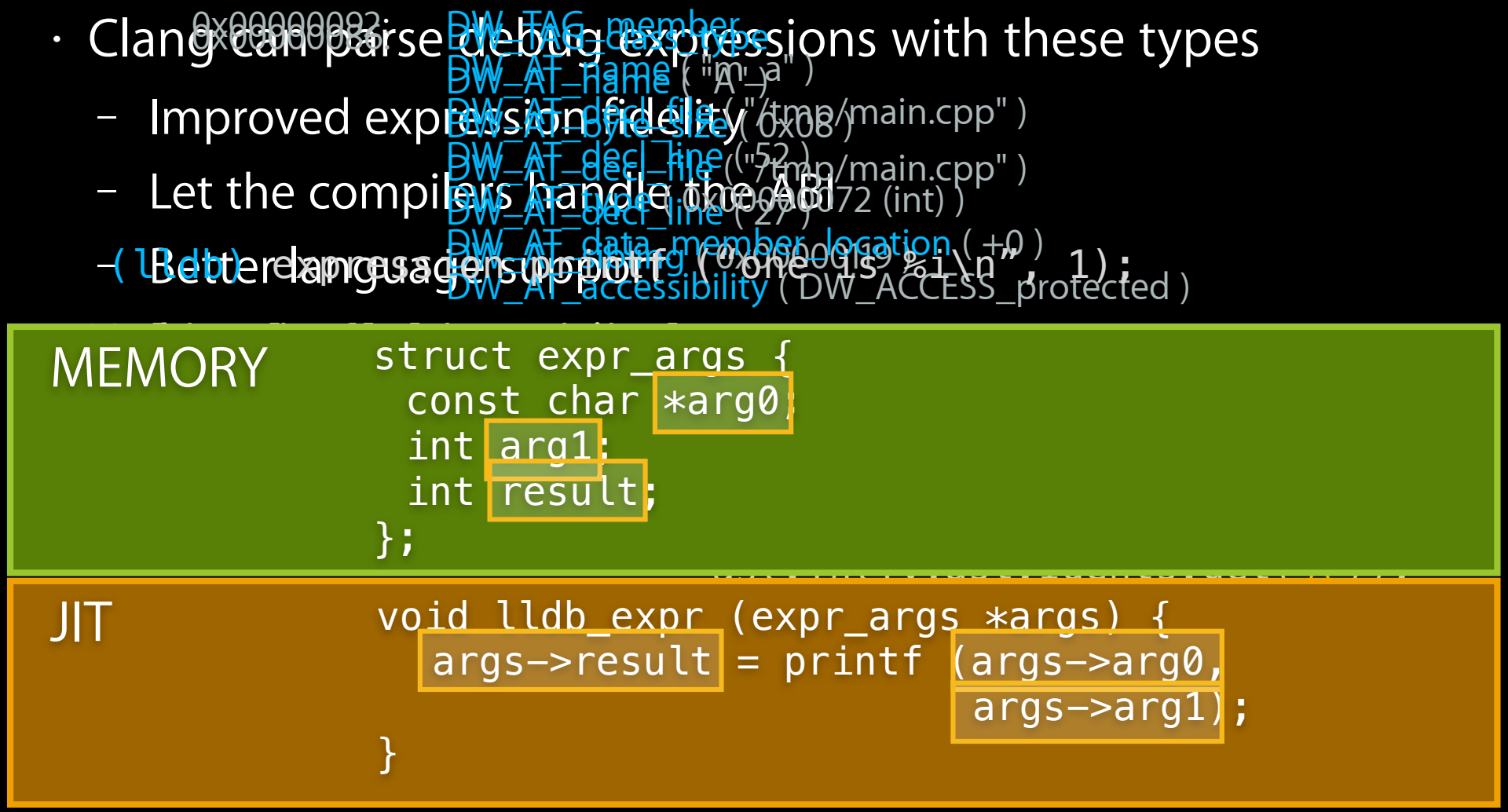
# Clang Integration

## What debuggers typically do…

- Most debuggers invent data structures
  - Functions
  - Types
  - Variables
- Most debuggers have their own expression parser
  - Parsers need to be updated
  - Writing a good C++ expression parser is easy, right???

# Clang Integration

- LLDB creates Clang ASTs from debug information
- Clang can parse debug expressions with these types
  - Improved expression fidelity
  - Let the compilers handle the ABI
  - Better language support



# Clang Expression Parser Benefits

- Multi-line expressions
- Expressions can have local variables
- Expressions can use flow control
- Persistent expression globals

```
(lldb) expression
for (int i=0; i<5; ++i)
printf(“%i\n”);
(lldb) expression int $i = 5
(int) $0 = 5
(lldb) expression ++$i
(int) $1 = 6
0
1
2
3
4
```

# LLDB Status

- Functional for  C, C++ and ObjC debugging
- Supported platforms
  - Mac OS X
  - Linux

| | Breakpoints | Dynamic Loader | Expressions | Stack | Variables | Run |
|---|---|---|---|---|---|---|
| MacOSX | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Linux | ✗ | | ✗ | ✗ | ✗ | ✗ |

# Future Goals and Direction

- Improved Linux support
- Abstract a concurrent programming model
- Have Clang serialize ASTs into object files
  - Compiler grade type and declaration information
  - DWARF for machine code representation
  - ASTs for declarations and types
  - Instantiate templates and inlined functions
  - Semantic breakpoints?
- Expanded disassembler abilities

# Conclusion

- LLDB
  - Modular
  - Scriptable
  - Extensible
  - Reusable
- Open sourced with LLVM and Clang

http://lldb.llvm.org/