Introducing LLDB for Linux on Arm and AArch64

Omair Javaid

Linaro
connect
Budapest 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Agenda

- Brief introduction and history behind LLDB
- Status of LLDB on Linux and Android
- Linaro’s contributions in LLDB
- An overview of LLDB Architecture
- More detail on working and architecture
- Overview of basic functionality and commands
- Overview of expressions and data formatters.

LLDB - Introduction

- Debugger component of LLVM project.
- A modular, high-performance source-level debugger written in C++
- Re-uses LLVM/Clang code JIT/IR for expression parsing, disassembly etc.
- Provides a C++ Debugger API which can be used by various clients
- Very permissive BSD-like license.

LLDB - Introduction (cont…)

- **Supported Host Platforms**
  - OS X, Linux/Android, FreeBSD, NetBSD, and Windows
- **Supported Target Architectures**
  - i386/x86_64, Arm/AArch64, MIPS/MIPS64, IBM s390
- **Supported Languages**
  - Fully support C, C++ and Objective-C while SWIFT and GoLang (under development)

LLDB - History

- Apple stopped shipping GDB after version 6.3.5
- Last GPLv2 based GDB release.
- Announced LLDB at WWDC2010.
- Extensively under development for various use-cases.
- Active code contributors are Apple and Google.
- Default debugger for OSX, Xcode IDE, Android Studio.
- Also adopted by FreeBSD as their default debugger.

LLDB - Linaro’s contributions

- **LLDB port for ARMv7 architecture.**
  - Register Context and Basic Debugging support.
  - Support for SysV ARM/Linux ABI.
  - Support for Arm Hardware Watchpoints
  - Support for Arm Single Byte-Selection Watchpoints.
  - Various bug fixes and improvements.
  - Support for Arm Hardware Breakpoints **(NEW)**

LLDB - Linaro's contributions (cont…)

- **LLDB port for ARMv8 architecture.**
  - Support for SysV AArch64/Linux ABI.
  - Support for AArch64 Hardware Watchpoints
  - Various bug fixes and improvements.
  - Support for AArch64 Hardware Breakpoints **(NEW)**
- **Maintenance and Testing**
  - Buildbot development and maintenance
  - Tester bring up and validation
  - Buildbot Failure Triage

LLDB - Features Status ARM/AArch64

- LLDB features stable on ARM and AArch64
- For more details visit: http://lldb.llvm.org/status.html

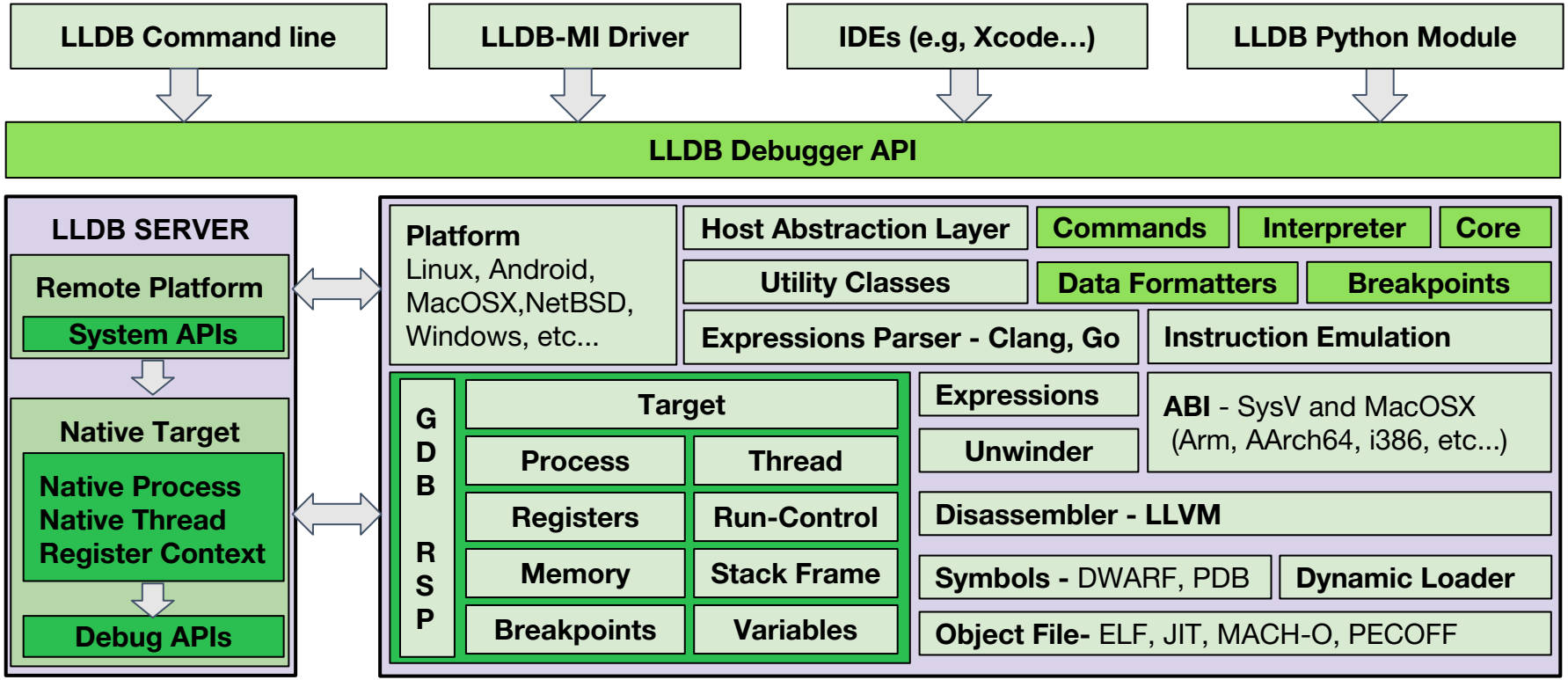
	Process control -- attach -- continue -- exec -- fork -- launch -- status	**Thread Control** -- step-in -- step-out -- step-over -- inspection	-- Breakpoints **-- HW Breakpoints** -- HW Watchpoints -- DWARF Symbols -- ELF Obj File -- C++11 Support -- Backtracing	-- Register Inspection -- Memory Inspection -- Expression Evaluation -- JIT Expressions -- Disassembly -- Remote Debugging
ARM	OK	OK	OK	OK
AArch64	OK	OK	OK	OK

LLDB - Arm/AArch64 Test Devices

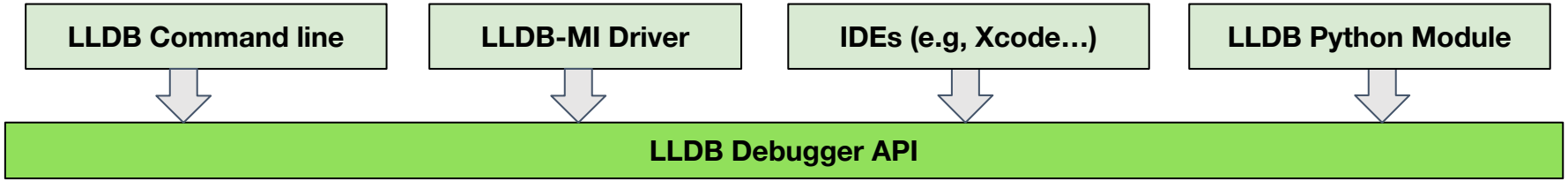
- Buildbots contributed by google test various Android devices
  - Link: lab.llvm.org:8011/builders/lldb-x86_64-ubuntu-14.04-android
- Linaro keeps track of LLDB on Linux platform for ARM and AArch64 targets.
  - Buildbot is still not public but we plan to do so.

Platform/Architecture	**ARMv7**	**ARMv8 - AArch64**	**ARMv8 - AArch32**
Android M	Nexus 5	Nexus 5x	Nexus 5x
Ubuntu Linux - Xenial	Raspberry Pi2	Pine64	Pine64

LLDB Architecture



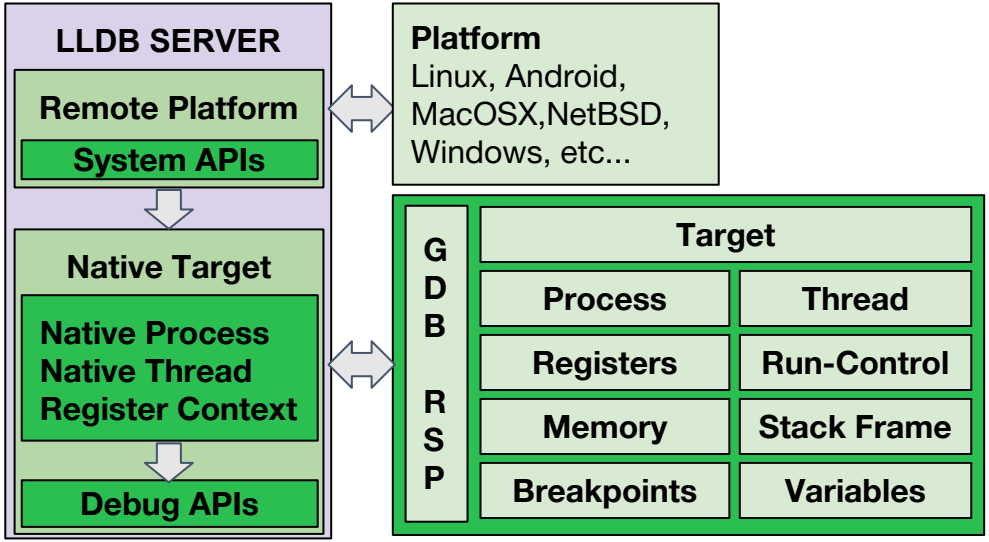
LLDB Debugger API



- A C++ shared library with object-oriented interface.
- **LLDB.framework** on MacOS X and **lldb.so** on linux.
- Used by
  - **lldb -** The LLDB Debugger command line
  - **lldb-mi** and **lldbmi2 -** Machine Interface (MI) drivers
  - **XCode** and **Android Studio -** IDEs with graphical front-ends.
  - **lldb Python module -** LLDB API exposed through python script bindings using SWIG.

lldb-server - Platform Mode

- File transfer to remote target.
- Spawns gdbserver on target.
- Runs shell commands on target.
- List remote processes.
- Allows single or multiple parallel debug sessions.



lldb-server - Platform Mode (Single Session)

- Spawn lldb-server on remote-android target in platform mode for single session.
- Select remote-android platform and connect.
- Try connecting another lldb remote-android platform session.

```
bullhead:/data/local/tmp/lldb/arm64 $
/lldb-server platform --listen *:5432
```

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb
(lldb) platform select remote-android
  Platform: remote-android
 Connected: no
(lldb) platform connect connect://localhost:5432
  Platform: remote-android
    Triple: aarch64-*-linux-android
OS Version: 25.0.0 (3.10.73-gfe160e5)
    Kernel: #1 SMP PREEMPT Wed Dec 7 20:26:32 UTC 2016
  Hostname: localhost
 Connected: yes
WorkingDir: /data/local/tmp/lldb/arm64
(lldb)
```

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb
(lldb) platform select remote-android
  Platform: remote-android
 Connected: no
(lldb) platform connect connect://localhost:5432
error: failed to get reply to handshake packet
(lldb)
```

lldb-server - Platform Mode (Multiple Sessions)

Select remote-android platform and connect.

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb
(lldb) platform select remote-android
  Platform: remote-android
 Connected: no
(lldb) platform connect connect://localhost:5432
  Platform: remote-android
    Triple: aarch64-*-linux-android
OS Version: 25.0.0 (3.10.73-gfe160e5)
    Kernel: #1 SMP PREEMPT Wed Dec 7 20:26:32 UTC 2016
  Hostname: localhost
 Connected: yes
WorkingDir: /data/local/tmp/lldb/arm64
(lldb)
```

Connect another remote-android platform session.

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb
(lldb) platform select remote-android
  Platform: remote-android
 Connected: no
(lldb) platform connect connect://localhost:5432
  Platform: remote-android
    Triple: aarch64-*-linux-android
OS Version: 25.0.0 (3.10.73-gfe160e5)
    Kernel: #1 SMP PREEMPT Wed Dec 7 20:26:32 UTC 2016
  Hostname: localhost
 Connected: yes
WorkingDir: /data/local/tmp/lldb/arm64
(lldb)
```

lldb-server platform -- server mode

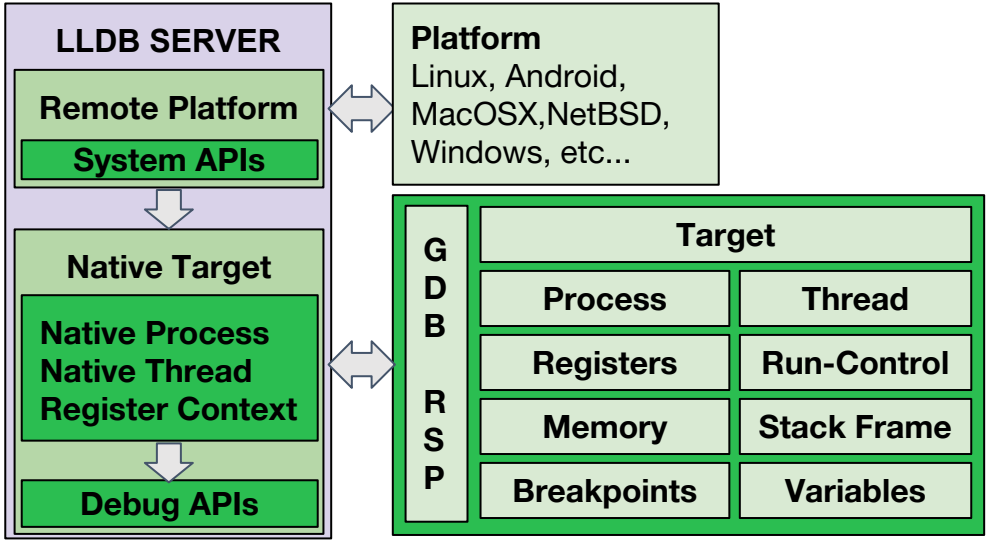
```
bullhead:/data/local/tmp/lldb/arm64 $
/lldb-server platform --listen *:5432 --server
Connection established.
```

lldb-server multiple sessions connected

```
bullhead:/data/local/tmp/lldb/arm64 $
/lldb-server platform --listen *:5432 --server
Connection established.
Connection established.
```

lldb-server - GDB Server Mode

- Run as a gdb-remote-stub.
- Communicates with LLDB over RSP protocol.
- On Mac OSX and iOS, the remote-gdb functionality is in **debugserver** binary.



lldb-server - GDB Server Mode

Start lldb-server in gdbserver mode

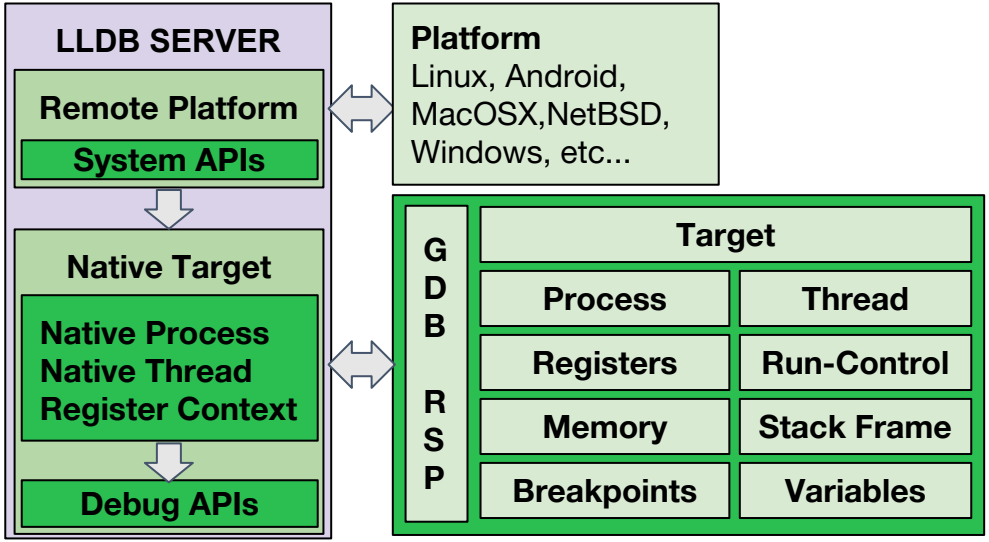
```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb-server gdbserver *:5432
lldb-server-local_buildConnection established.
Launched '/home/omair/work/lldb-dev/hwbreak/x86_64.out' as process 26892...
```

Connect to gdb remote stub and start a debugging session

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb /home/omair/work/lldb-dev/hwbreak/x86_64.out
(lldb) target create "/home/omair/work/lldb-dev/hwbreak/x86_64.out"
Current executable set to '/home/omair/work/lldb-dev/hwbreak/x86_64.out' (x86_64).
(lldb) process connect connect://localhost:5432
(lldb) b main
Breakpoint 1: where = x86_64.out`main + 8 at test.c:14, address = 0x000000000040054b
(lldb) run
Process 26892 launched: '/home/omair/work/lldb-dev/hwbreak/x86_64.out' (x86_64)
Process 26892 stopped
* thread #1, name = 'x86_64.out', stop reason = breakpoint 1.1
    frame #0: 0x000000000040054b x86_64.out`main at test.c:14
   11
   12   int main()
   13   {
-> 14     int temp = 0;
   15
   16     printf("Read Op: Value of temp is %i",temp);
   17
(lldb)
```

lldb - Debuggee (or inferior) context on host system

- **Platform**
  - Local or remote platform functions.
- **Process GDB Remote Client**
  - Let lldb host talks to lldb-server running in gdb remote mode.
  - Performs debug operations via GDB Remote Serial Protocol (RSP)
- **Target**
  - Implements debugee (inferior) management functions like
    - Process Control
    - Thread Control
    - Registers and Memory
    - Stack Frames and Variables
    - Breakpoints and Watchpoints
    - Module loading



lldb - *target create* **and** *process launch*

 # Select target platform
platform select <name>

 # Connect to platform
platform connect <url>

 # Load file and symbols
target create <file-name>

 # Set breakpoint at main
breakpoint set -b main

 # Start remote inferior
process launch

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb
(lldb) platform select remote-android
  Platform: remote-android
 Connected: no
(lldb) platform connect connect://localhost:5432
  Platform: remote-android
    Triple: aarch64-*-linux-android
OS Version: 25.0.0 (3.10.73-gfe160e5)
    Kernel: #1 SMP PREEMPT Wed Dec 7 20:26:32 UTC 2016
  Hostname: localhost
 Connected: yes
WorkingDir: /data/local/tmp/lldb/arm64
(lldb) target create /home/omair/work/lldb-dev/hwbreak/arm64.out
Current executable set to '/home/omair/work/lldb-dev/hwbreak/arm64.out' (aarch64).
(lldb) breakpoint set -b main
Breakpoint 1: where = arm64.out`main + 8 at test.c:14, address = 0x00000000000006dc
(lldb) process launch
Process 17232 launched: '/home/omair/work/lldb-dev/hwbreak/arm64.out' (aarch64)
Process 17232 stopped
* thread #1, name = 'arm64.out', stop reason = breakpoint 1.1
    frame #0: 0x000000558e4a06dc arm64.out`main at test.c:14
   11
   12   int main()
   13   {
-> 14     int temp = 0;
   15
   16     printf("Read Op: Value of temp is %i",temp);
   17
(lldb)
```

lldb - Target and Process commands

Target

- **Create / Delete**
- **List and Select**
- **Symbols**
- **Variables**

Process

- **Connect -** to remote debug.
- **Launch -** current target executable
- **Attach/Detach -** to a process
- **Continue/Interrupt/Kill -** current target process.
- **Load/Unload -** shared library

lldb - Thread, Frame, Memory & Registers commands

Thread

- **Stepping**
  - **Step-in, step-out** and **step-over**
  - **step-inst** and **step-inst-over**
- **Select -** Select current thread.
- **Backtrace -** Thread call stacks.
- **Continue -** Continue one or all threads.
- **info/list -** Information about threads in current process.

Frame, Memory and Registers

- **Frame**
  - **Variable**
  - **Select**
- **Memory**
  - **Read**
  - **Write**
- **Registers**
  - **register read <reg-name>**
  - **register write <reg-name> <data>**

lldb - Break/Watchpoints & Disassembly commands

breakpoint/watchpoint

- **set <options>**
  - Set a breakpoint based on options.
- **clear** <line-no or filename>
- **command <add, delete or list>**
  - Run command or script when breakpoint is hit.
- **delete**
  - All or specified breakpoint is deleted.
- **enable/disable**
  - All or specified breakpoint.
- **read/write**
  - Read from or write breakpoints to file.

disassemble

- **(-f) --frame**
  - from the start of the current frame
- **(-m) --mixed**
  - mixed source and assembly display.
- **(-n) --name <function-name>**
  - entire contents of the given function.
- **(-p) --pc**
  - around the current pc.
- **(-a) --address <address>**
  - function containing address.
- **--start-address / --end-address**

lldb - *command alias <alias> <command> <args>*

Some Built-in Aliases

File <file>

- target create <file>

b main

- breakpoint set -b main

run

- process launch

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb
(lldb) platform select remote-android
  Platform: remote-android
 Connected: no
(lldb) platform connect connect://localhost:5432
  Platform: remote-android
    Triple: aarch64-*-linux-android
OS Version: 25.0.0 (3.10.73-gfe160e5)
    Kernel: #1 SMP PREEMPT Wed Dec 7 20:26:32 UTC 2016
  Hostname: localhost
 Connected: yes
WorkingDir: /data/local/tmp/lldb/arm64
(lldb) file /home/omair/work/lldb-dev/hwbreak/arm64.out
Current executable set to '/home/omair/work/lldb-dev/hwbreak/arm64.out' (aarch64).
(lldb) b main
Breakpoint 1: where = arm64.out`main + 8 at test.c:14, address = 0x00000000000006dc
(lldb) run
Process 17286 launched: '/home/omair/work/lldb-dev/hwbreak/arm64.out' (aarch64)
Process 17286 stopped
* thread #1, name = 'arm64.out', stop reason = breakpoint 1.1
    frame #0: 0x00000055863896dc arm64.out`main at test.c:14
   11
   12   int main()
   13   {
-> 14     int temp = 0;
   15
   16     printf("Read Op: Value of temp is %i",temp);
   17
(lldb)
```

lldb - Using .lldbinit file

.lldbinit file on top right of this slide is running at startup:

- Platform select and connect
- Load object file and symbols
- Set breakpoint at main
- Start inferior process
- Run to main.

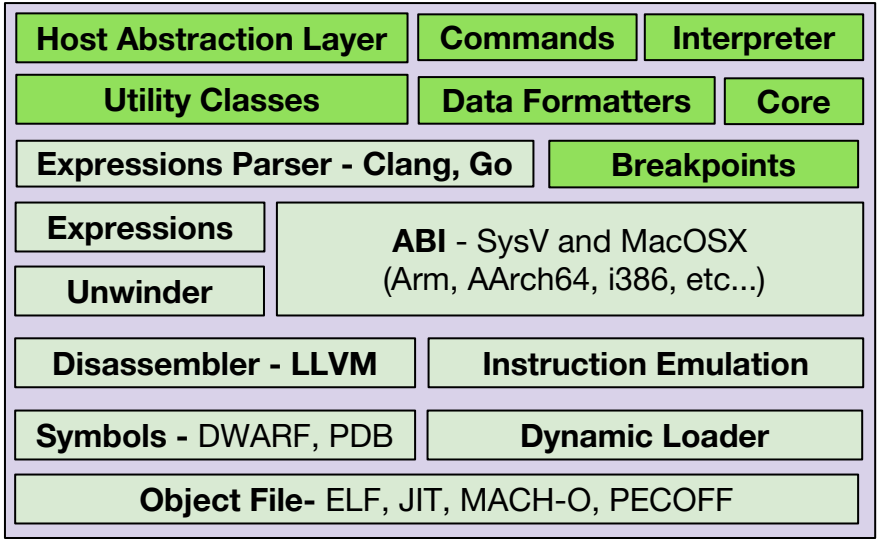
LLDB command-line output on bottom right shows lldb state after we start it with .lldbinit file in user's home directory.

```
1  # Select platform
2  platform select remote-android
3
4  # Connect to platform
5  platform connect connect://localhost:5432
6
7  # Create debuggee target with executable
8  file /home/omair/work/lldb-dev/hwbreak/arm64.out
9
10 # Set breakpoint at main function
11 b main
12
13 # Launch debuggee process
14 run
```

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb
Process 17782 stopped
* thread #1, name = 'arm64.out', stop reason = breakpoint 1.1
    frame #0: 0x000000558151d6dc arm64.out`main at test.c:14
   11
   12   int main()
   13   {
-> 14     int temp = 0;
   15
   16     printf("Read Op: Value of temp is %i",temp);
   17
(lldb)
```

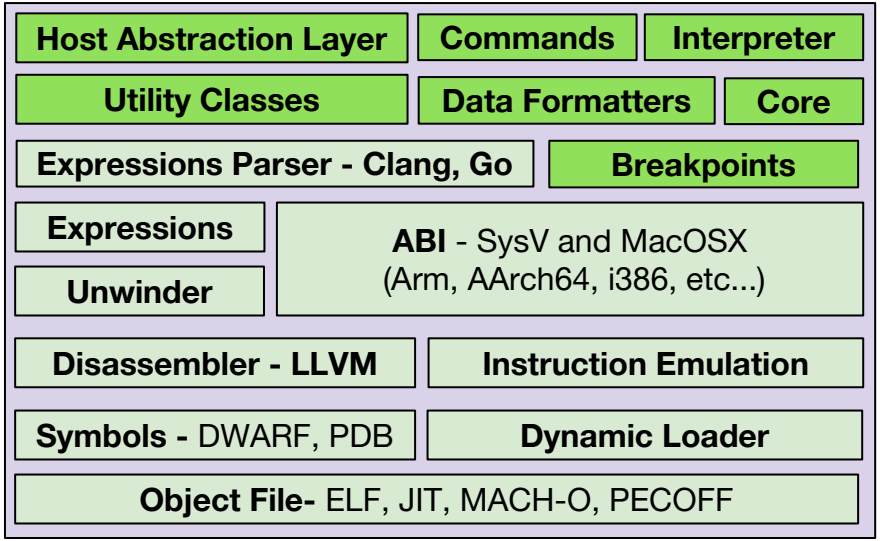
LLDB - Brief description of components

- **ABI**
  - Executes JITTed function calls in process being debugged.
  - Call frame calculation based on target calling convention.
- **Unwinder**
  - Implements stack unwinding features.
- **Instruction Emulation**
  - Helps with stack frame constructions.
  - Helps single step by emulating execution paths.
- **Expression Parser**
  - A modified DWARF expression parser
  - Clang front-end latest c++ support



LLDB - Brief description of components

- **Disassembler**
  - LLVM disassembler
- **Dynamic Loader**
  - Helps detect dynamically loaded modules
- **Object File**
  - Handles object files management.
- **Symbols**
  - Handles debug symbols mangement
- **Host Abstraction Layer**
- **Core and Utility Classes**
- **Interpreter and Commands**
- **Data Formatters**
- **Breakpoint Utilities**



LLDB - **unique_ptr** variables with data formatters

Without deleter

- frame variable nup
- frame variable iup
- frame variable sup

With deleter

- frame variable ndp
- frame variable idp
- frame variable sdp

```
* thread #1, name = 'unique.out', stop reason = breakpoint 1.1
    frame #0: 0x0000000000400baf unique.out`main at main.cpp:21
   18      std::unique_ptr<std::string, Deleter> sdp(new std::string("baz"),
   19                                                Deleter{3, 4});
   20
-> 21      return 0; // Set break point at this line.
   22    }
(lldb) frame variable nup
(std::unique_ptr<char, std::default_delete<char> >) nup = nullptr {}
(lldb) frame variable iup
(std::unique_ptr<int, std::default_delete<int> >) iup = 0x616c20 {
  object = 123
}
(lldb) frame variable sup
(std::unique_ptr<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::def
sup = 0x616c40 {
  object = "foobar"
}
(lldb) frame variable ndp
(std::unique_ptr<char, Deleter>) ndp = nullptr {
  object = <parent is NULL>

  deleter = (a = 0, b = 0)
}
(lldb) frame variable idp
(std::unique_ptr<int, Deleter>) idp = 0x616c70 {
  object = 456
  deleter = (a = 1, b = 2)
}
(lldb) frame variable sdp
(std::unique_ptr<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, Deleter>
  object = "baz"
  deleter = (a = 3, b = 4)
}
```

LLDB - **shared_ptr** & **weak_ptr** with data formatters

--ptr-depth 1

- Dereference pointer once

std::shared_ptr

- frame variable **nsp**
- frame variable **isp**
- frame variable **ssp**

std::weak_ptr

- frame variable **nwp**
- frame variable **iwp**
- frame variable **swp**

```
Process 13041 launched: '/home/omair/work/bud17/formatters/smart.out' (x86_64)
Process 13041 stopped
* thread #1, name = 'smart.out', stop reason = breakpoint 1.1
    frame #0: 0x0000000000400cc7 smart.out`main at main.cpp:15
   12       std::weak_ptr<int> iwp = isp;
   13       std::weak_ptr<std::string> swp = ssp;
   14
-> 15       nsp.reset(); // Set break point at this line.
   16       isp.reset();
   17       ssp.reset();
   18
(lldb) frame variable --ptr-depth 1 nsp
(std::shared_ptr<char>) nsp = nullptr {
  _M_ptr = 0x0000000000000000
}
(lldb) frame variable --ptr-depth 1 isp
(std::shared_ptr<int>) isp = 123 {
  _M_ptr = 0x0000000000617c20 {
    *_M_ptr = 123
  }
}
(lldb) frame variable --ptr-depth 1 ssp
(std::shared_ptr<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >)
  _M_ptr = "\x80|a"... {}
}
(lldb) frame variable --ptr-depth 1 iwp
(std::weak_ptr<int>) iwp = 123 {
  _M_ptr = 0x0000000000617c20 {
    *_M_ptr = 123
  }
}
(lldb) frame variable --ptr-depth 1 swp
(std::weak_ptr<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >) s
  _M_ptr = "\x80|a"... {}
}
```

LLDB - **std::map** variables with data formatters

std::map<std::string, int>

- frame variable **si**

std::map<int, std::string>

- frame variable **is**

std::map<std::string, std::string>

- frame variable **ss**

```
Process 13892 launched: '/home/omair/work/bud17/formatters/map.out' (x86_64)
Process 13892 stopped
* thread #1, name = 'map.out', stop reason = breakpoint 1.1
    frame #0: 0x0000000000401363 map.out`main at main.cpp:29
   26       ss["gatto"] = "cat";
   27       ss["a Mac.."] = "..is always a Mac!";
   28
-> 29       return 0;// Set break point at this line.
   30   }
(lldb) frame variable si
(std::map<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, 
<std::pair<const std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<cha
  [0] = (first = "one", second = 1)
  [1] = (first = "three", second = 3)
  [2] = (first = "two", second = 2)
  [3] = (first = "zero", second = 0)
}
(lldb) frame variable is
(std::map<int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char
, std::allocator<char> > > >) is = size=4 {
  [0] = (first = 1, second = "is")
  [1] = (first = 2, second = "smart")
  [2] = (first = 3, second = "!!!")
  [3] = (first = 85, second = "goofy")
}
(lldb) frame variable ss
(std::map<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, 
tring<char, std::char_traits<char>, std::allocator<char> > >, std::allocator<std::pair<const
ar, std::char_traits<char>, std::allocator<char> > > >) ss = size=4 {
  [0] = (first = "a Mac..", second = "..is always a Mac!")
  [1] = (first = "casa", second = "house")
  [2] = (first = "ciao", second = "hello")
  [3] = (first = "gatto", second = "cat")
}
(lldb)
```

LLDB - Evaluate **expressions** using clang parser

- Compile “hello world” code
- Launch it using LLDB.
- Stop in main before return.
- Now our expressions will have a context to work with.
- Let's try some example expressions.

```
1 #include <iostream>
2
3 int main()
4 {
5   std::cout << "Hello World!" << std::endl;
6
7   return 0;
8 }
```

LLDB - Evaluate **expressions** using clang parser

- Let's try evaluating a few simple expressions.
- Create **integer type** local variables **i** and **j**
- Let see if we can call **getpid**
- Evaluate **getpid() + i + j**
- Evaluate **getpid() + i & j**
- Evaluate **getpid() + i ^ j**
- Evaluate **getpid() + i << j**

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb ~/work/bud17/jit/hello
(lldb) target create "/home/omair/work/bud17/jit/hello"
Current executable set to '/home/omair/work/bud17/jit/hello' (x86_64).
(lldb) breakpoint set -l 7
Breakpoint 1: where = hello`main + 32 at hello.cpp:7, address = 0x000000000040
(lldb) run
Process 15744 launched: '/home/omair/work/bud17/jit/hello' (x86_64)
Hello World!
Process 15744 stopped
* thread #1, name = 'hello', stop reason = breakpoint 1.1
    frame #0: 0x0000000000400866 hello`main at hello.cpp:7
   4    {
   5      std::cout << "Hello World!" << std::endl;
   6
-> 7      return 0;
   8    }
(lldb) expression int $i = 9
(lldb) expression int $j = 3
(lldb) expression (int)getpid()
(int) $0 = 15744
(lldb) expression (int)getpid() + $i + $j
(int) $1 = 15756
(lldb) expression (int)getpid() + $i & $j
(int) $2 = 1
(lldb) expression (int)getpid() + $i ^ $j
(int) $3 = 15754
(lldb) expression (int)getpid() + $i << $j
(int) $4 = 126024
(lldb)
```

LLDB - Evaluate **expressions** using clang parser

- Create context as we did in previous slide.
- Try evaluating a **for** loop expression.

```
omair@omAlien:~/work/bud17$ ../lldb-test/build/host/bin/lldb ./jit/hello
(lldb) target create "./jit/hello"
Current executable set to './jit/hello' (x86_64).
(lldb) breakpoint set -l 7
Breakpoint 1: where = hello`main + 32 at hello.cpp:7, address = 0x0000000000400866
(lldb) run
Process 13579 launched: './jit/hello' (x86_64)
Hello World!
Process 13579 stopped
* thread #1, name = 'hello', stop reason = breakpoint 1.1
    frame #0: 0x0000000000400866 hello`main at hello.cpp:7
   4    {
   5      std::cout << "Hello World!" << std::endl;
   6
-> 7      return 0;
   8    }
(lldb) expression for (int i = 0; i < 10; i++) printf("%d x %d = %d\n",i,i, i * i);
0 x 0 = 0
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
6 x 6 = 36
7 x 7 = 49
8 x 8 = 64
9 x 9 = 81
(lldb)
```

LLDB - Evaluate **expressions** using clang parser

- Create a c++ lambda expression named add.
- Specify return value **auto**
- Evaluate lambda expression on our dummy context.
- Cool we have them working.
- Lets try a more complex lambda with **auto** arguments

```
omair@omAlien:~/work/lldb-test/build/host/bin$ ./lldb ~/work/bud17/jit/hello
(lldb) target create "/home/omair/work/bud17/jit/hello"
Current executable set to '/home/omair/work/bud17/jit/hello' (x86_64).
(lldb) breakpoint set -l 7
Breakpoint 1: where = hello`main + 32 at hello.cpp:7, address = 0x000000000040
(lldb) run
Process 16036 launched: '/home/omair/work/bud17/jit/hello' (x86_64)
Hello World!
Process 16036 stopped
* thread #1, name = 'hello', stop reason = breakpoint 1.1
    frame #0: 0x0000000000400866 hello`main at hello.cpp:7
   4    {
   5      std::cout << "Hello World!" << std::endl;
   6
-> 7      return 0;
   8    }
(lldb) expression auto $add = [](int a, int b) { return a + b; }
(lldb) expression $add(2,3)
(int) $0 = 5
(lldb) expression $add(33,66)
(int) $1 = 99
(lldb) expression auto $add = [](auto a, auto b) { return a + b; }
error: 'auto' not allowed in lambda parameter
error: 'auto' not allowed in lambda parameter
(lldb)
```

Linaro
connect
Budapest 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Conclusion

- Start using LLDB.
- Its more stable than ever before.
- Wanna go an extra mile? Feel free to contribute.
- Topics for future
  - Using LLDB Python API and built-in Python interpreter
  - How-To on more advanced expressions
  - How to use more advanced and Python based data formatters

Linaro
connect
Budapest 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Questions

Email: omair.javaid@linaro.org

IRC: omjavaid @ freenode and oftc

References

- LLDB (Architecture, GDB to LLDB commands, Doxygen)
- LLVM LAB - LLDB Android Buildbot
- Linaro LAB - LLDB Linux Buildbot (Under Development)
- LLDB on FreeBSD

Thank You

#BUD17

For further information: www.linaro.org

BUD17 keynotes and videos on: connect.linaro.org