ERICSSON

# MAKING YOUR DEBUGGING EFFORTS COUNT:

## BEST PRACTICES WITH THE CDT DEBUGGER

Marc Khouzam

# ABOUT ME

- Working with CDT Debug since 2007
- CDT project co-lead, lead for Debug component
- Things you don't like about CDT Debug are probably my fault
- You can help get them improved
  - Give feedback
  - Open bugs
  - Contribute

# AGENDA

- Running the debugger
- No more *Printf-debugging*
- Examining debugging data
- Controlling execution
- Multi-thread and beyond
- More advanced topics
- Future plans

And one demo chosen by YOU

# RUNNING THE DEBUGGER

# STAND-ALONE DEBUGGER

› Easy installation through its own package
  – https://eclipse.org/cdt/downloads.php
  – https://wiki.eclipse.org/CDT/StandaloneDebugger

CDT Downloads

https://eclipse.org/cdt/down

**CDT 8.8.1 for Eclipse Mars**

Eclipse package: Eclipse C/C++ IDE for Mars.2.

p2 software repository: http://download.eclipse.org/tools/c

The git repos have been tagged with the CDT_8_8_1 tag. Yc

Archived p2 repos:

- cdt-8.8.1.zip
- cdt-8.8.0.zip
- cdt-8.7.0.zip

CDT 8.8.1 stand-alon

download.eclipse.org/tools/cdt/rel

**CDT 8.8.1 stand-alone RCP debugger**

- cdt-stand-alone-debugger-1.1.1-20160205-0501-linux.gtk.x86.tar.gz (77.5M)
- cdt-stand-alone-debugger-1.1.1-20160205-0501-linux.gtk.x86_64.tar.gz (77.6M)
- cdt-stand-alone-debugger-1.1.1-20160205-0501-win32.win32.x86.zip (78.0M)
- cdt-stand-alone-debugger-1.1.1-20160205-0501-win32.win32.x86_64.zip (78.1M)

CDT also provides its debugger as a stand-alone RCP application which can be downloaded and installed on its own, from: http://download.eclipse.org/tools/cdt/releases/8.8.1/r/rcp

The RCP stand-alone debugger has its own p2 software repository to be used within the RCP itself to upgrade it: http://download.eclipse.org/tools/cdt/releases/8.8.1/r/rcp-repository

# STAND-ALONE DEBUGGER

› Easy launch: ./cdtdebug *-e myBinary*

Eclipse C/C++ Stand-alone Debugger

File Edit Search Run Window Help

Debug

_home_lmckhou_testing_loopfirst [C/C++ Application]
loopfirst [22889] [cores: 0]
Thread #1 [loopfirst] 22889 [core: 0] (Suspended : Step)
main() at loopfirst.cc:10 0x400570
gdb (7.11)

Variables | Breakpoints | Registers | Modules

| Name | Type | Value |
|---|---|---|
| i | int | 42 |
| max | int | 900 |

loopthird.cc | gdb.c | loopfirst.cc

```
2  #include <stdio.h>
3
4  int j = 0;
5
6  int main() {
7      int max = 900;
8      for (int i=0;i<max;i++) {
9          printf("i is %d\n",i);
10         sleep(1);
11     }
12     return 0;
13 }
```

Outline | Console

_home_lmckhou_testing_loopfirst [C/C++ Application] loopfirst

```
i is 36
i is 37
i is 38
i is 39
i is 40
i is 41
i is 42
```

# POST-MORTEM DEBUG

- Examining a core file: Variables, Registers, Memory
- ./cdtdebug -c *coreFile* -e *matchingBinary*

Debug

- DSFTestApp Core [C/C++ Postmortem Debugger]
  - DSFTestApp
    - Thread #1 28327 (Suspended : Container)
      - dup() at DSFTestApp.cpp:203 0x4016fd
      - main() at DSFTestApp.cpp:244 0x4018b1
  - gdb.7.9 (7.9)

Variables | Breakpoints | Modules

| Name | Type | Value |
|---|---|---|
| argc | int | 1 |
| argv | char ** | 0x7fffffffdf18 |
| b | int [5] | 0x7fffffffdcc0 |
| um | wchar_t | 32767 L'翻' |
| array_small | int [4] | 0x7fffffffdcf0 |
| de | int | 0 |

DSFTestApp.cpp

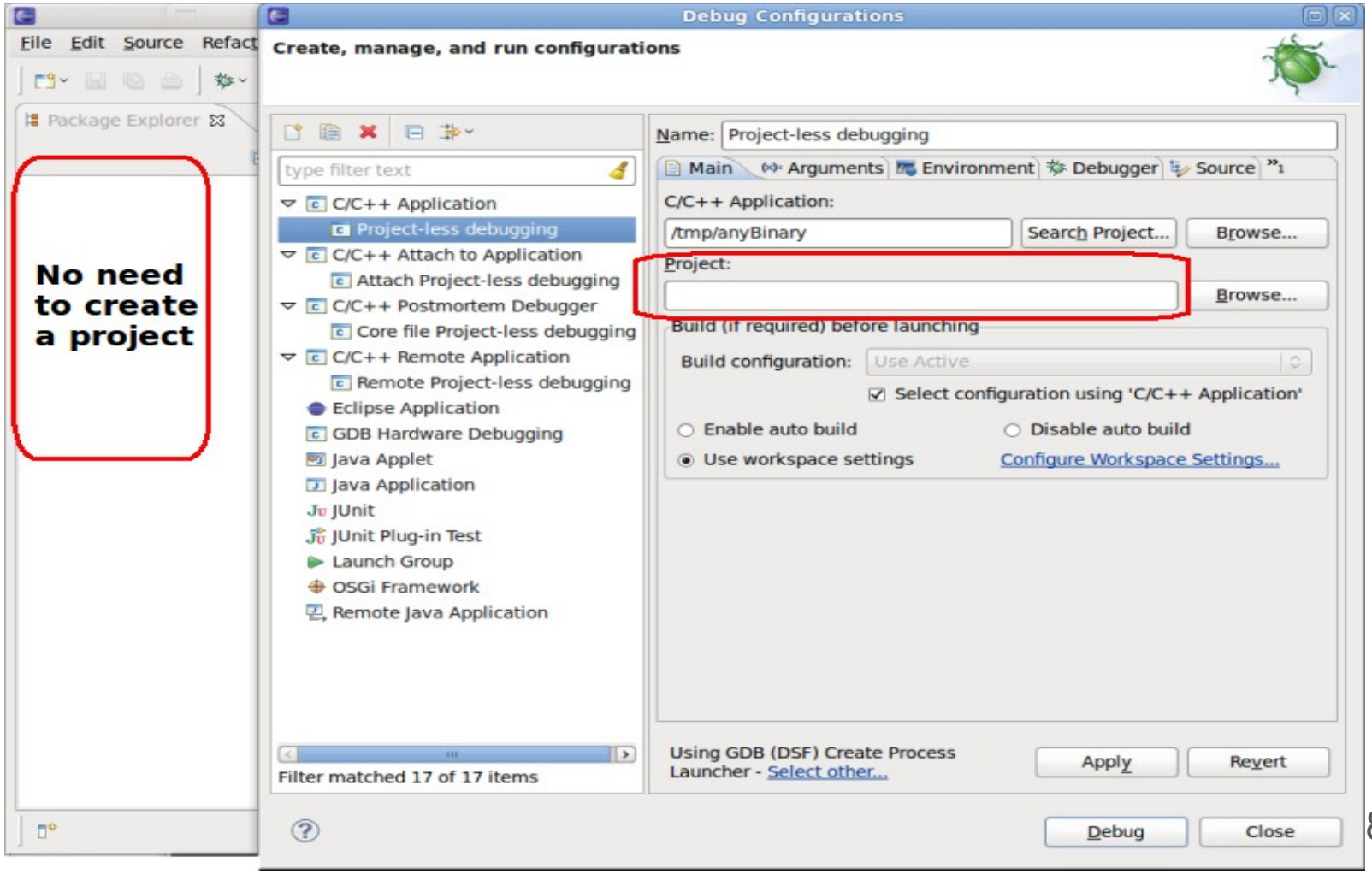
```
        thatIsABigArray[i] = i;
    }

    float myFloat[4];
    myFloat[0] = 4.6;
    myFloat[1] = 3.14;
    myFloat[2] = 6;
    myFloat[3] = (float)1/(float)3;

    dup();

    int x = 1;
    while(x < 5){
//      sleep(1);
```

Console | Memory | Registers

| Name | Value | Descript |
|---|---|---|
| rbp | 0x7fffffffde30 | |
| rsp | 0x7fffffffb1a0 | |
| r8 | 4294967295 | |
| r9 | 0 | |
| r10 | 34 | |
| r11 | 582 | |
| r12 | 4199104 | |
| r13 | 140737488346896 | |

# PROJECT-LESS DEBUG

› Debug any binary!



ERICSSON

# NO MORE PRINTF-DEBUGGING

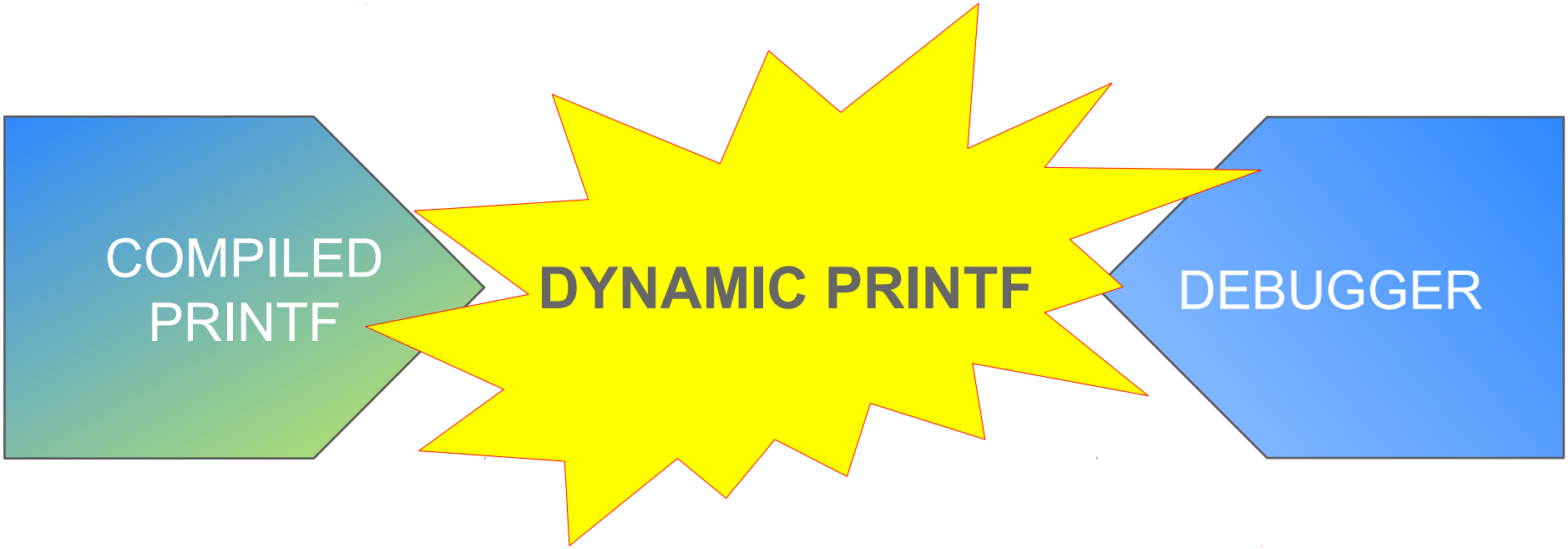
# PRINTF DEBUGGING

- Still much too popular
  - Comfortable, familiar, easy

- Costly efficiency limitations
  - Expensive debug cycle
    1. Recompiling
    2. Redeploying to target
    3. Repeating steps to reproduce issue
  - Info provided is fixed per debug cycle
  - ➔ Multiple such debug cycles

# DYNAMIC-PRINTF

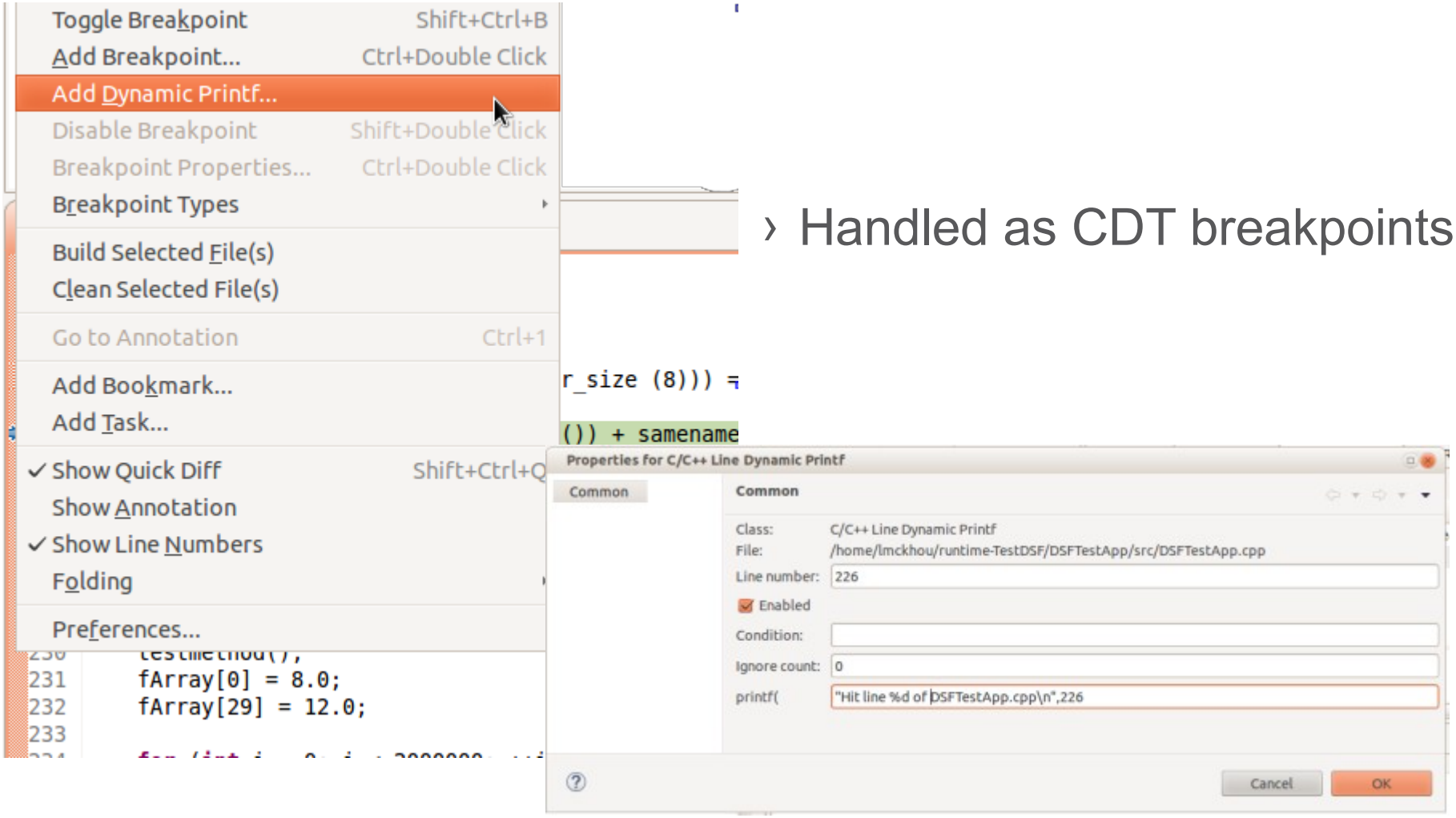
› Familiarity meets flexibility and efficiency!



# DYNAMIC-PRINTF

- Printf dynamically inserted by debugger in executing program
- Prints in same location as compiled-printfs
- Same syntax as compiled-printf
- No recompiling! No redeploying!

# DYNAMIC-PRINTF




› Handled as CDT breakpoints

# EXAMINING DEBUGGING DATA

# ADVANCED DEBUG HOVER

Producer.cpp | Consumer.cpp | Factorial.c | PrettyPrinting.cpp

```
    return NULL;
}

// Send some content to port 10010 at variable intervals of 1 to 4 seconds
void *produce(void *ptr) {
    printf("I am the %s thread\n", (char*)ptr);

    struct sockaddr_in addr;
    int sd;

    if ((sd = socket(PF
        perror("Socket"
        abort();
    }
```

| Expression | Type | Value |
|---|---|---|
| addr | sockaddr_in | {...} |
| sin_family | sa_family_t | 2 |
| sin_port | in_port_t | 6695 |

```
Name : addr
    Details:{sin_family = 2, sin_port = 6695, sin_addr = {s_addr = 16777343}, sin_zer
    Default:{...}
    Decimal:{...}
    Hex:{...}
    Binary:{...}
```

- In-hover expression view
- Detail pane
- User can modify data directly

# PRETTY-PRINTING

› STL classes inspect poorly e.g., Vector, List, Map

Variables | Breakpoints | Expressions | Registers | Modules

| Name | Type | Value |
|---|---|---|
| coll | std::vector<std::vector<int, std::alloca | {...} |
| std::_Vector_base<std::vector< | std::_Vector_base<std::vector<int, std: | {...} |
| _M_impl | std::_Vector_base<std::vector<int, std: | {...} |
| std::allocator<std::vector | std::allocator<std::vector<int, std::allo | {...} |
| _M_start | std::vector<int, std::allocator<int> > * | 0x8055358 |
| _M_finish | std::vector<int, std::allocator<int> > * | 0x8055388 |
| _M_end_of_storage | std::vector<int, std::allocator<int> > * | 0x8055388 |
| str | std::string | {...} |

```
Name : coll
    Details:{<std::_Vector_base<std::vector<int, std::allocator<int> >, std::allocator<std::vect
    Default:{...}
    Decimal:{...}
    Hex:{...}
    Binary:{...}
    Octal:{...}
```

# PRETTY-PRINTING

› Pretty-printers provided with STL library

› Values of elements can even be modified by user!

Variables | Breakpoints | Expressions | Registers | Modules

| Name | Type | Value |
|---|---|---|
| coll | std::vector<std::vector<int, std::alloca | {...} |
| [0] | std::vector<int, std::allocator<int> > | {...} |
| [0] | int | 1 |
| [1] | int | 2 |
| [2] | int | 3 |
| [1] | std::vector<int, std::allocator<int> > | {...} |
| [0] | int | 10 |
| [1] | int | 11 |

```
Name : coll
    Details:std::vector of length 4, capacity 4 = {std::vector of length 3, capacity 3 = {1, 2, 3}
    Default:{...}
    Decimal:{...}
    Hex:{...}
    Binary:{...}
    Octal:{...}
```

# PER-ELEMENT FORMAT

› Ability to set format per element

› Variables, Expressions, Registers views

Breakpoints | Expressions | Variables | Registers

| Expression | Type | Value | Address |
|---|---|---|---|
| * | Group-pattern | 54 unique matches | |
| a | int [2] | 0x7fffffffdbf0 | 0x7fffffffdbf0 |
| aba | int | 100100 (Binary) | 0x7fffffffdcd4 |
| argc | int | 1 | |
| argv | char ** | 0x7fffffffddf8 | |
| array_large | int [111] | 0x7fffffffd6e0 | |
| array_small | int [4] | 0x7fffffffdbd0 | |
| b | int [5] | 0x7fffffffdba0 | |
| bbb | int *[5] | 0x7fffffffdae0 | |
| c | char | 0 '\0' | |
| de | int | 0 (Binary) | |

```
Name : aba
    Details:36
    Default:36
    Decimal:36
    Hex:0x24
    Binary:100100
    Octal:044
```

- Select All Ctrl+A
- Copy Expressions Ctrl+C
- Remove
- Remove All
- Number Format
  - Details
  - Default
  - Decimal
  - Hex
  - • Binary
  - Octal
  - Restore To Preference
- Find... Ctrl+F
- Add Watch Expression...
- Disable
- Enable
- Edit Watch Expression...
- Add Watchpoint (C/C++)...
- Cast To Type...
- Display As Array...
- Watch

# MODIFYING DATA

Memory Browser | Console | Memory

Monitors

- 0xb6516360
- &delay

0xb6516360 : 0xB6516360 <Hex> | New Renderings...

| Address | 0 - 3 | 4 - 7 | 8 - B | C - F |
|---|---|---|---|---|
| B6516360 | 01000000 | 05000000 | 00000042 | 0200271A |
| B6516370 | 7F000001 | 00000000 | 00000000 | 002E0B6B |
| B6516380 | 00000000 | 00000000 | 886451B6 | 6E69FBB7 |
| B6516390 | 46900408 | 02000000 | 02000000 | 02000000 |
| B65163A0 | 446451B6 | 00000000 | 00000000 | 00000000 |
| B65163B0 | 00000000 | 00000000 | 706B51B6 | 00000000 |
| B65163C0 | 00000000 | 00000000 | 00000000 | 00000000 |
| B65163D0 | 00000000 | 00000000 | 00000000 | 00000000 |

Variables

| Name | Type | Value |
|---|---|---|
| ptr | void * | 0x8049046 |
| msg | char | 0 '\000' |
| i | int | 12345 |
| addr | sockaddr_in | {...} |
| sd | int | 5 |
| delay | int | 1 |

› Modifying data during execution:

- Memory view
- Variables view
- Registers view
- Expressions view
- Hover

# RETURN VALUE DISPLAY

› Return value shown after step-return

Debug

test [C/C++ Application]
test [942] [cores: 1]
Thread [1] 942 [core: 1] (Suspended : Step)
foo() at test.cc:6 0x4004db
main() at test.cc:13 0x400500
gdb

When at line 2, pressing step-return will trigger showing the return value

Registers | Variables | Breakpoints | Expressions

| Name | Type | Value | Location |
|---|---|---|---|
| computeValue() returned | int | 84 | |
| i | int | 28 | 0x7fffffffe53c |

```
Name : computeValue() returned
    Details:84
    Default:84
    Decimal:84
    Hex:0x54
    Binary:1010100
    Octal:0124
```

DSFTestApp.cpp | test.cc

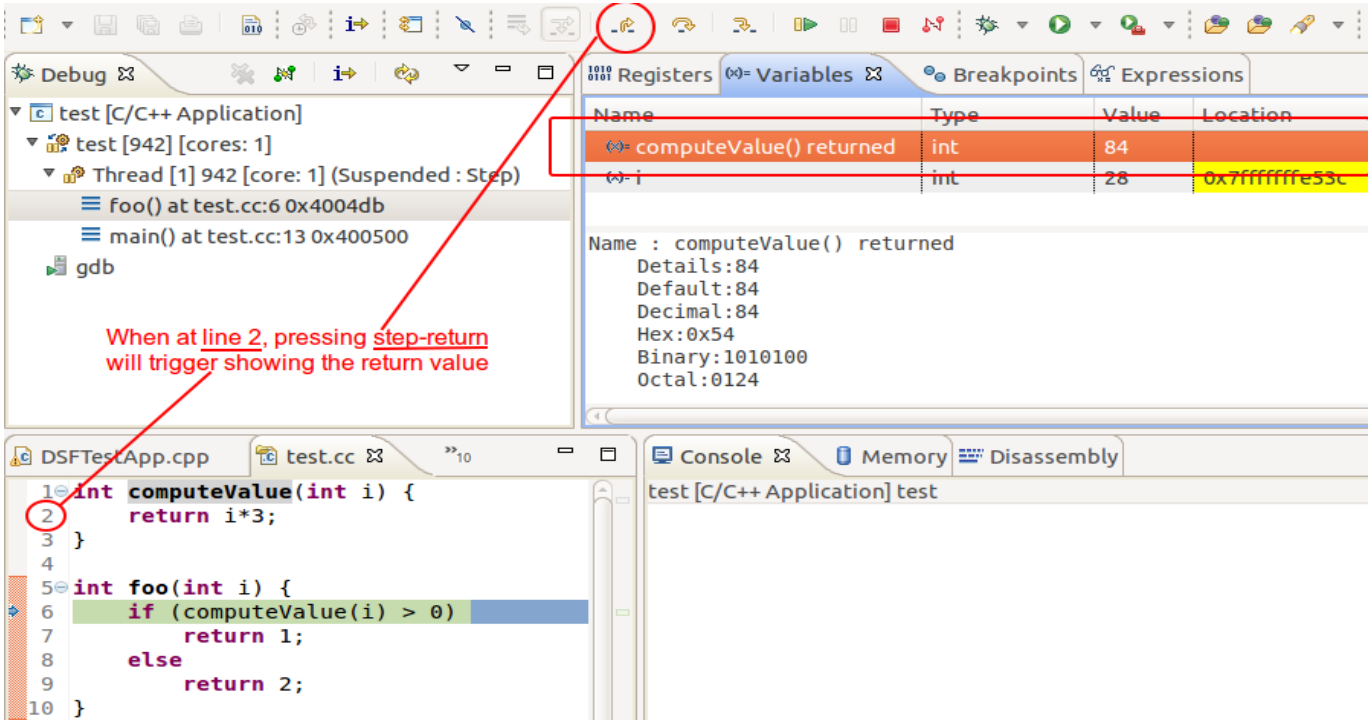
```
1  int computeValue(int i) {
2      return i*3;
3  }
4
5  int foo(int i) {
6      if (computeValue(i) > 0)
7          return 1;
8      else
9          return 2;
10 }
11
12 int main() {
13   foo(28);
14   return 0;
15 }
16
```

Console | Memory | Disassembly

test [C/C++ Application] test

# RETURN VALUE ON STEP-OVER

› Currently return value shown only after step-return




› Plans to show return value after a step-over

› Could be multiple values for a line such as:

– add ( multiply(6,2), divide(9, 3) );

# ENHANCED-EXPRESSIONS

› Shell-like pattern-matching for variables and registers

Expressions

| Expression | Type | Value |
|---|---|---|
| i ; =$eax ; =array[1,6] | Group-pattern | 4 unique matches |
| i | int | 9 |
| $eax | int32_t | -1 |
| array[1] | int | 1 |
| array[6] | int | 6 |
| =$eb* ; =um? | Group-pattern | 5 unique matches |
| $ebp | void * | 0xbfffefe8 |
| $ebx | int32_t | -1073747096 |
| um2 | wchar_t | 129 L'\201' |
| um3 | wchar_t | 255 L'ÿ' |
| um4 | wchar_t | 256 L'Ā' |
| =array[305-308] | Group-pattern | 4 unique matches |
| array[305] | int | 305 |
| array[306] | int | 306 |
| array[307] | int | 307 |
| array[308] | int | 308 |
| * | Group-pattern | 54 unique matches |
| a | int [2] | 0xbfffef08 |
| aba | int | 882090000 |

# ENHANCED-EXPRESSIONS

- Support for pattern-matching and expressions groups
- Provides alphabetical sorting

  - Pattern-matched local variables
    - =v?r – Show all local vars matching pattern
    - =* – Show all local vars alphabetically

  - Array ranges
    - =myarray[30-40] – Show elements 30 to 40
    - =myarray[1-3,20,23-24] – Show elements 1,2,3,20,23,24

# ENHANCED-EXPRESSIONS

- Support for defining expressions and expressions groups
  - Pattern-matched registers
    - =$xmm* – Show all registers starting with xmm
    - =$* – Show all registers
  - Semi-colon-separated groups
    - var1; var2 – Group which children are var1 and var2
    - var1;=* – Show all local vars with var1 being shown first

# ENHANCED-EXPRESSIONS

› Super-set of Variables and of Registers views

Expressions

| Expression | Type | Value |
|---|---|---|
| i ; =$eax ; =array[1,6] | Group-pattern | 4 unique matches |
| i | int | 9 |
| $eax | int32_t | -1 |
| array[1] | int | 1 |
| array[6] | int | 6 |
| =$eb* ; =um? | Group-pattern | 5 unique matches |
| $ebp | void * | 0xbfffefe8 |
| $ebx | int32_t | -1073747096 |
| um2 | wchar_t | 129 L'\201' |
| um3 | wchar_t | 255 L'ÿ' |
| um4 | wchar_t | 256 L'Ā' |
| =array[305-308] | Group-pattern | 4 unique matches |
| array[305] | int | 305 |
| array[306] | int | 306 |
| array[307] | int | 307 |
| array[308] | int | 308 |
| * | Group-pattern | 54 unique matches |
| a | int [2] | 0xbfffef08 |
| aba | int | 882090000 |

# CONTROLLING EXECUTION

# RUN-TO-LINE

› Run-to-line

- Ctrl+R – Execute program until selected code line
- Or right-click on selected line in editor for menu option

DSFTestApp.cpp | Producer.cpp | a.cc

```
253     int* ptr = &i;
254     food* ffPtr = &ff;
255     int aba;
256     bar s = ff;
257     wchar_t um = 0xe4; // ä
258     wchar_t um2 = 0x81; // Invisible.
259     wchar_t um3 = 0xFF; //ÿ
260     wchar_t um4 = 0x100; //Ā
261
262     returnArray(b);
263
264     returnFood(ff);
265
266     int longarray[1000];
267     int *longarraypointer;
268     longarraypointer=&longarray[0];
269     int shortarray[10];
270     shortarray[0] = 260;
271     int *shortarraypointer;
272     shortarraypointer=&shortarray[0];
273
274     int array_small[4] = {0x41424344, 0x45464748}; // Decimal: "1094861636", "116
```

- Step Into Selection Ctrl+F5
- Show IASTNode in DOM View
- Run to Line Ctrl+R
- Move To Line
- Resume At Line
- Add Watch Expression...
- Profile As
- Debug As
- Run As
- Team
- Compare With
- Replace With
- Preferences...
- Input Methods

# STEP-INTO-SELECTION

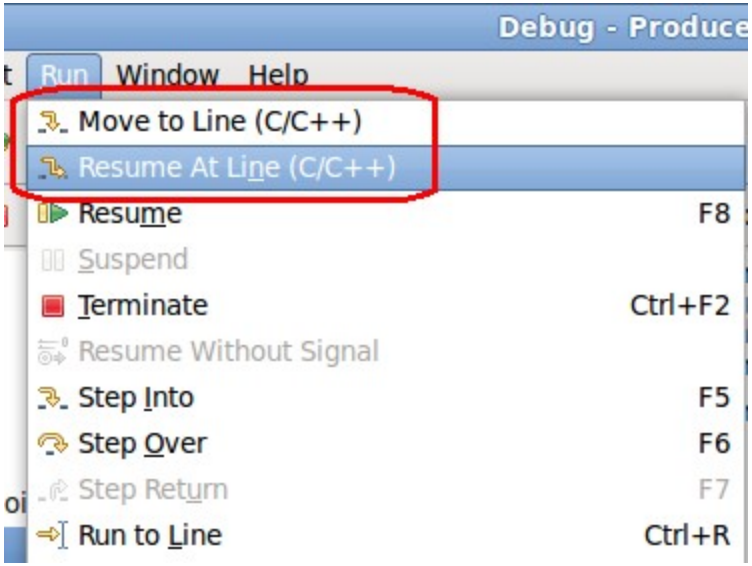
› Ability to specify *which* method to step into

– One step to step into 'substract' instead of 5

```
1295    int result;
1296    int a = 4, b = 5;
1297
1298    add(4,7);
1299    multiply(5,4);
1300
1301    result = subtract( multiply( add( a, b ), 3 ), 5 );
1302
```

# MOVE-TO-LINE & RESUME-AT-LINE

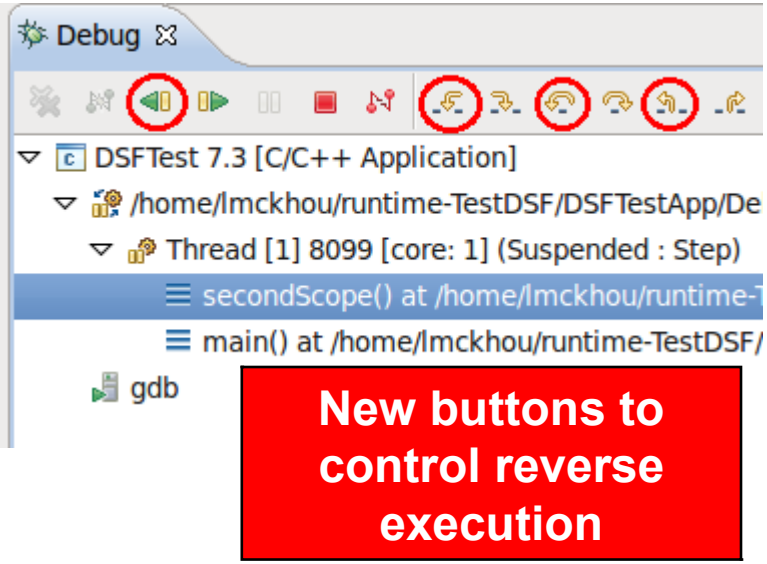
- Move-to-line: set execution line to selected one
- Resume-at-line: move-to-line and automatically resume
- From *Run* menu or *editor right-click* menu



# REVERSE DEBUGGING

## STEP PROGRAM BACKWARDS

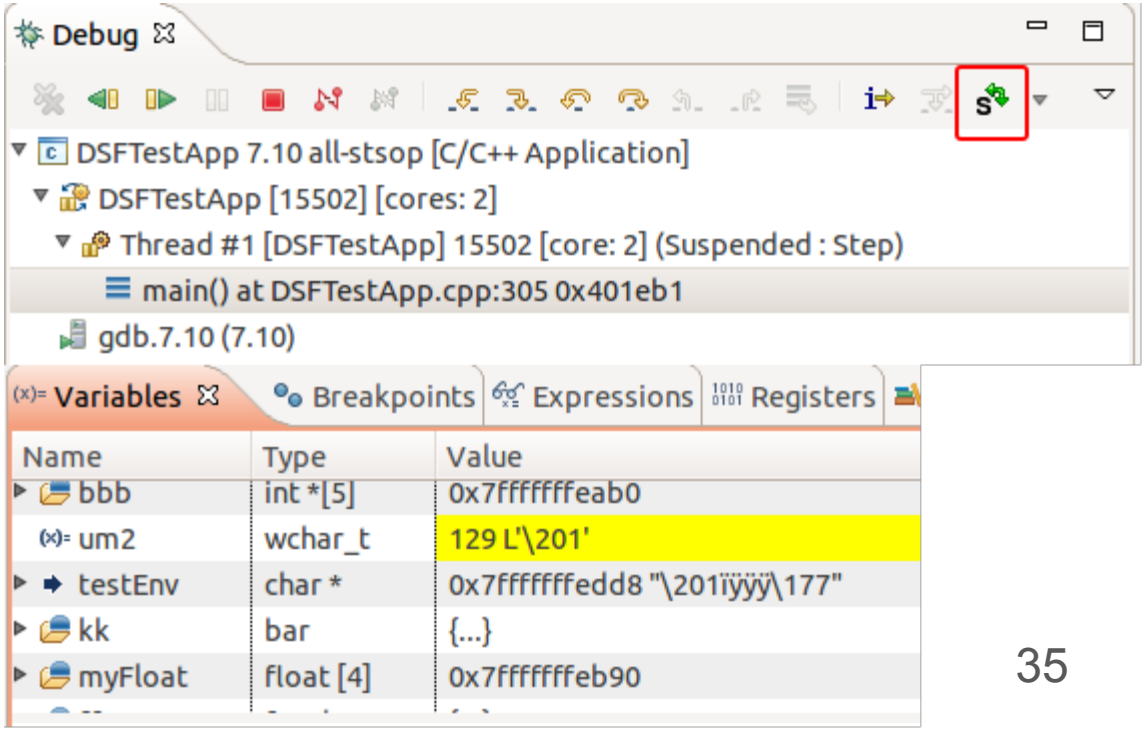
- Recording of program execution
- Replay in reverse
- Allows to examine past execution without restarting it
- Reverse-step, reverse-resume
- Can use breakpoints set in the 'past'



# REVERSE DEBUGGING

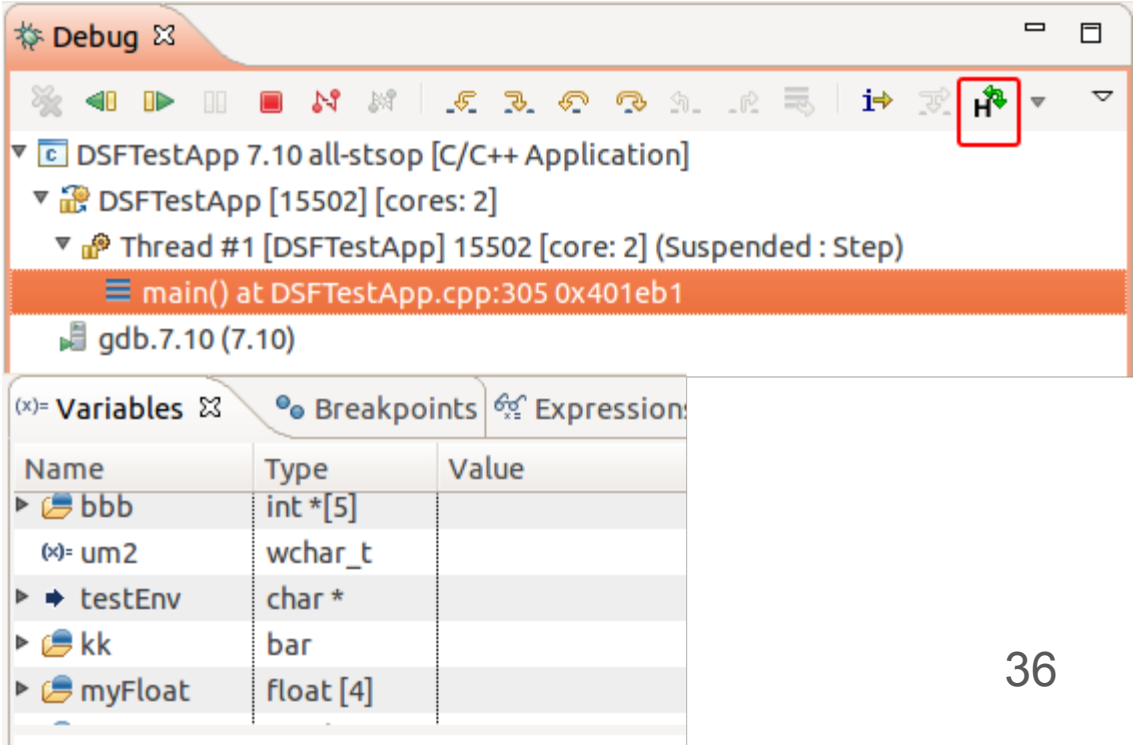
› Software recording

- Code path
- Variables changes
- Register changes
- Memory changes



# REVERSE DEBUGGING

- Hardware recording
  - Code path only
  - Requires Intel(R) processor



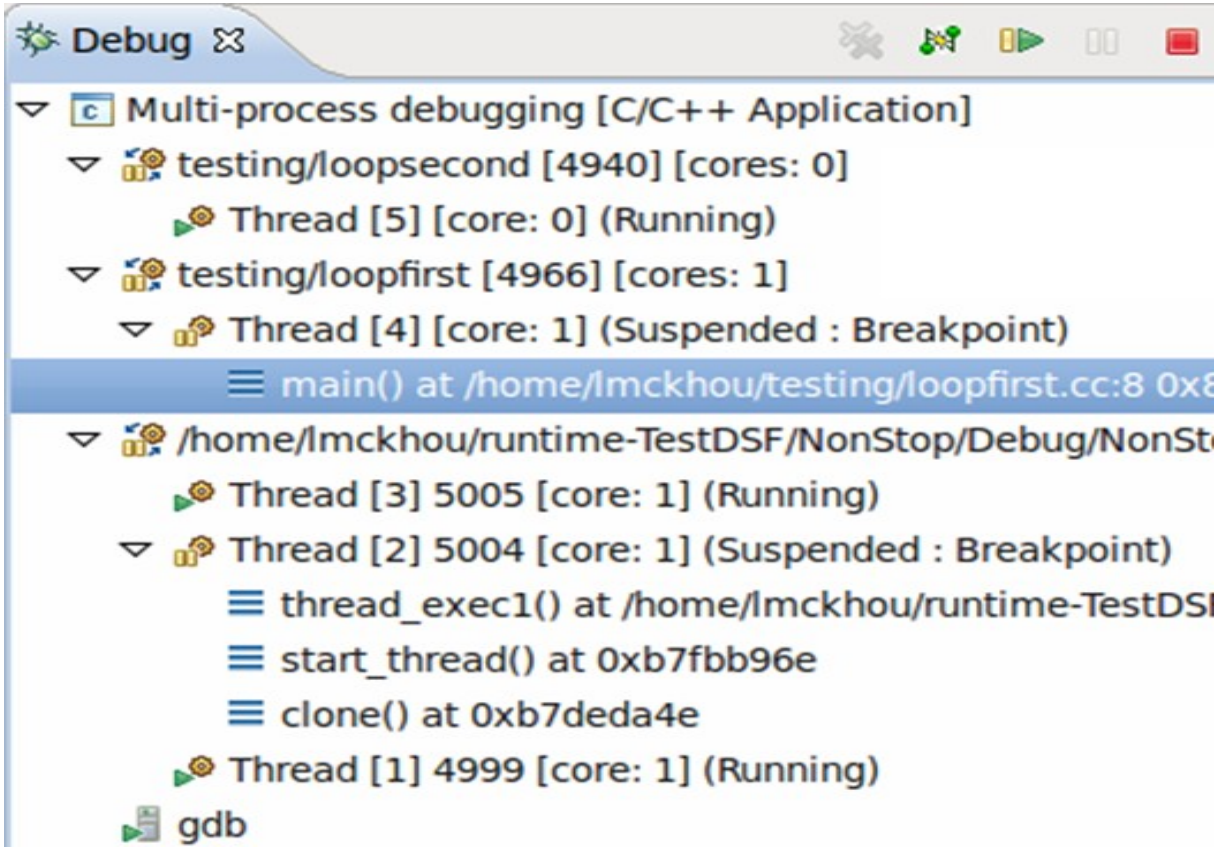
ERICSSON

# MULTI-THREAD AND BEYOND

# NON-STOP DEBUGGING

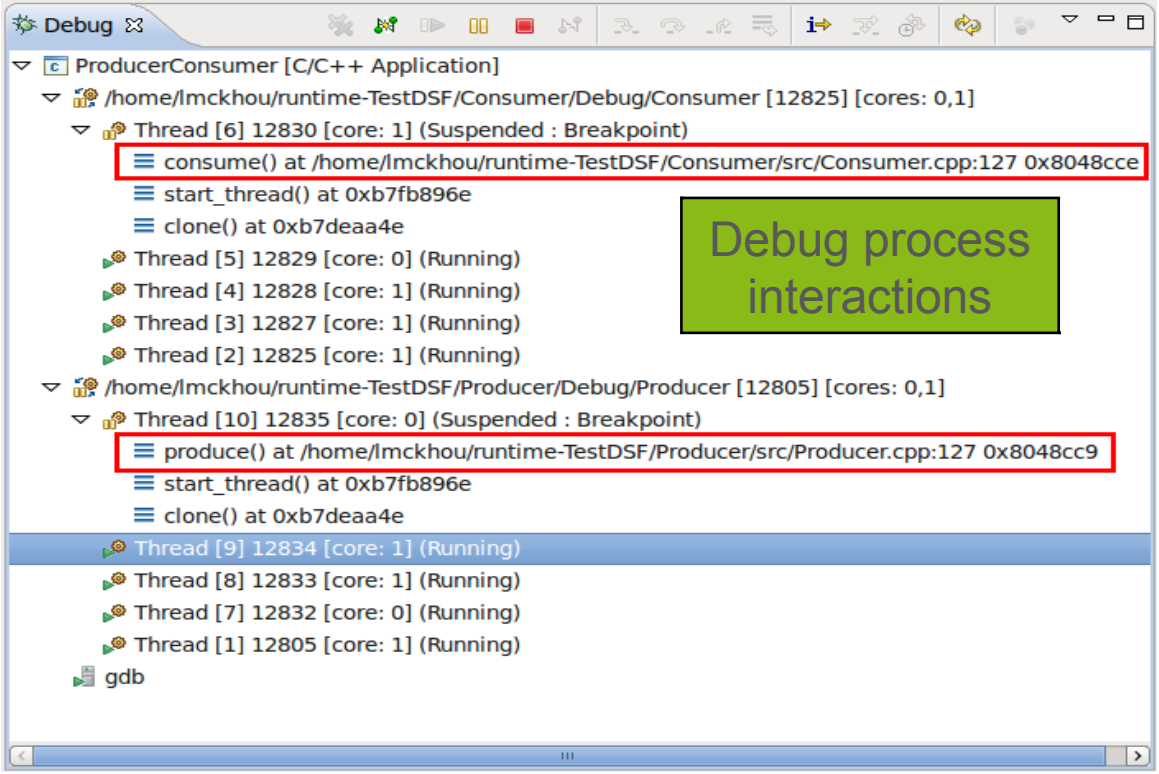
› Program continues execution while suspending some threads

› Reduced intrusiveness



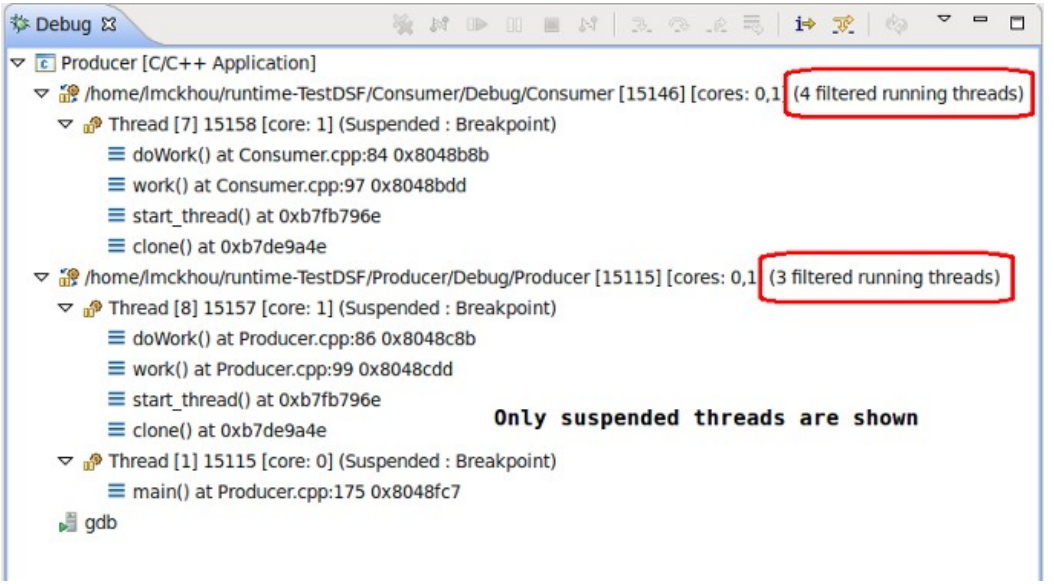
# MULTI-PROCESS DEBUGGING

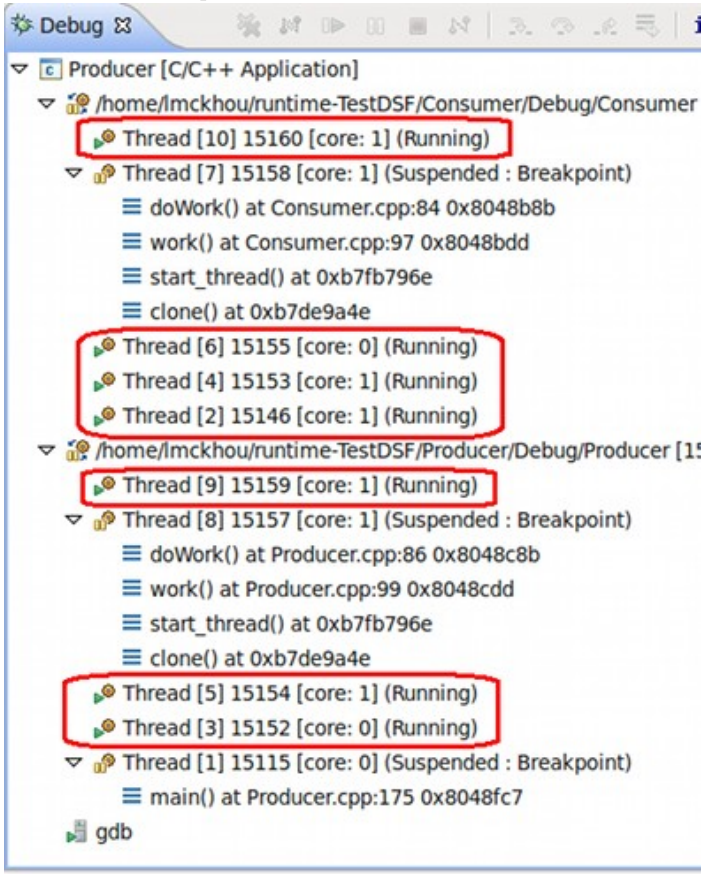
**One gdb controlling many processes**



# FOCUS ON SUSPENDED THREADS

- Crowded display when program has many threads
  - What is really of interest?
  - Threads actively being debugged, i.e., suspended
  - Enable from preferences

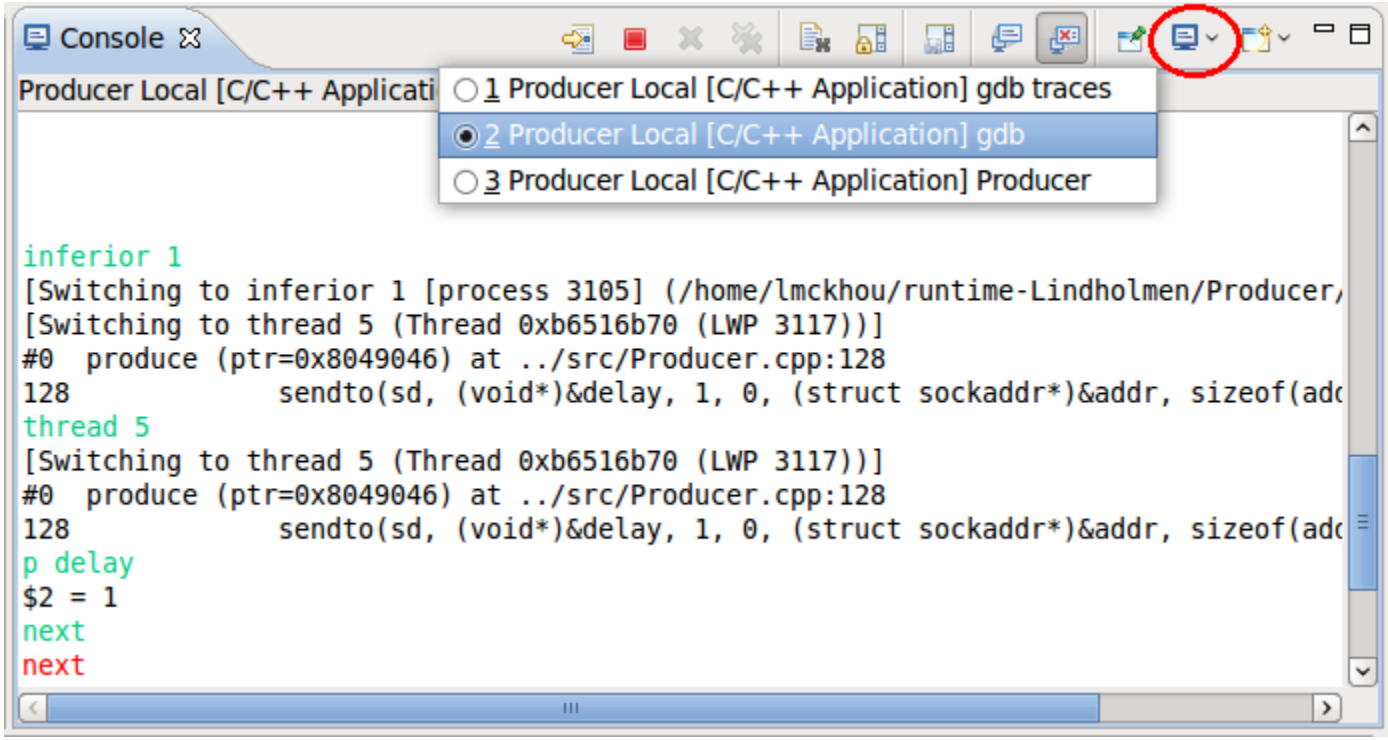


# MORE ADVANCED TOPICS

# GDB AND GDB CONSOLE

- GDB is the brains behind CDT Debug
- Can use gdb command-line from eclipse
- Currently very basic.



# FULL GDB CONSOLE

› Targeting CDT 9.1 and GDB 7.12 (by September 2016)

Debug

PrettyPrinting new console [C/C++ Application]
PrettyPrinting [11859] [cores: 2]
Thread #1 [PrettyPrinting] 11859 [core: 2] (Su
main() at PrettyPrinting.cpp:57 0x400f88
/home/lmckhou/git/palves.gdb/build.console/

DSFTestApp.cpp | PrettyPrinting.cpp

```
56
57      for (int i=0;i<100;i++) {
58          intArray[i] = i;
59      }
60
61      vector<Derived> derivedSet;
62
63      map<int, string> mappedStrings;
```

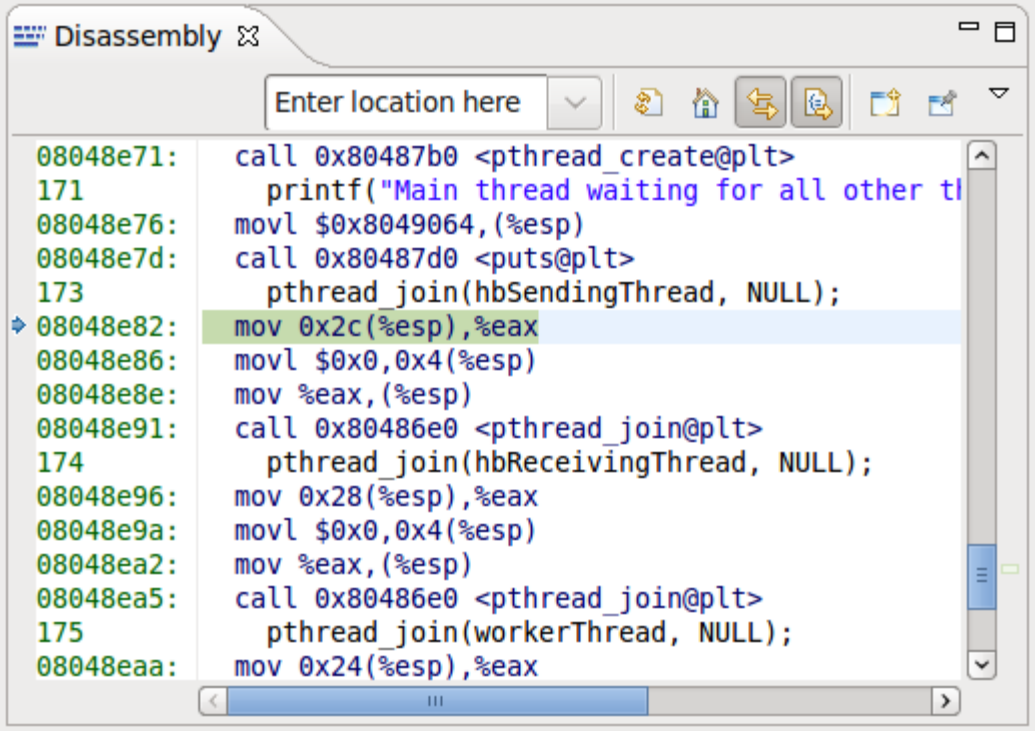
Console

PrettyPrinting new console [C/C++ Application] gdb console

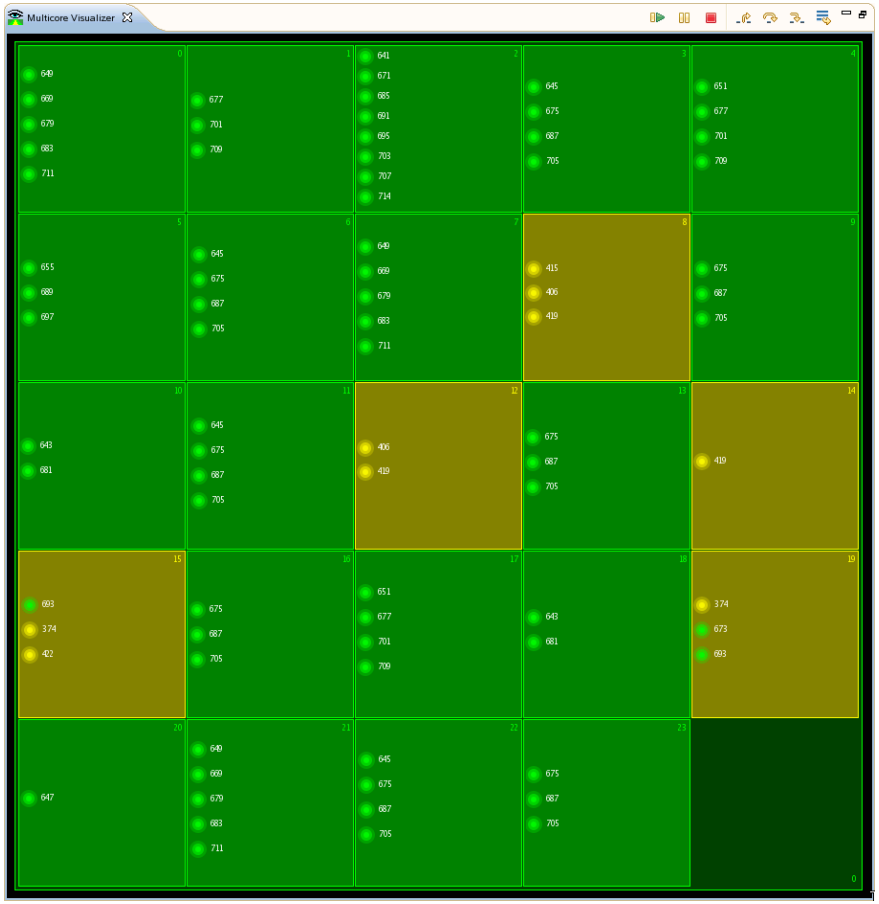
```
Type "apropos word" to search for commands related to "word".
(gdb)

Temporary breakpoint 1, main () at ../src/PrettyPrinting.cpp:57
57              for (int i=0;i<100;i++) {
(gdb) n
58                      intArray[i] = i;
(gdb) n
57              for (int i=0;i<100;i++) {
(gdb) ne
new-console  next         nexti
(gdb)
```

# DISASSEMBLY VIEW

- Shows disassembly of code (optionally with source)
- Supports breakpoints like in editor (and dynamic-printf!)
- Step/resume/suspend from Disassembly view



# MULTICORE VISUALIZER

# FUTURE PLANS

# GLOBAL BREAKPOINTS

› Contribution to Linux Kernel ongoing

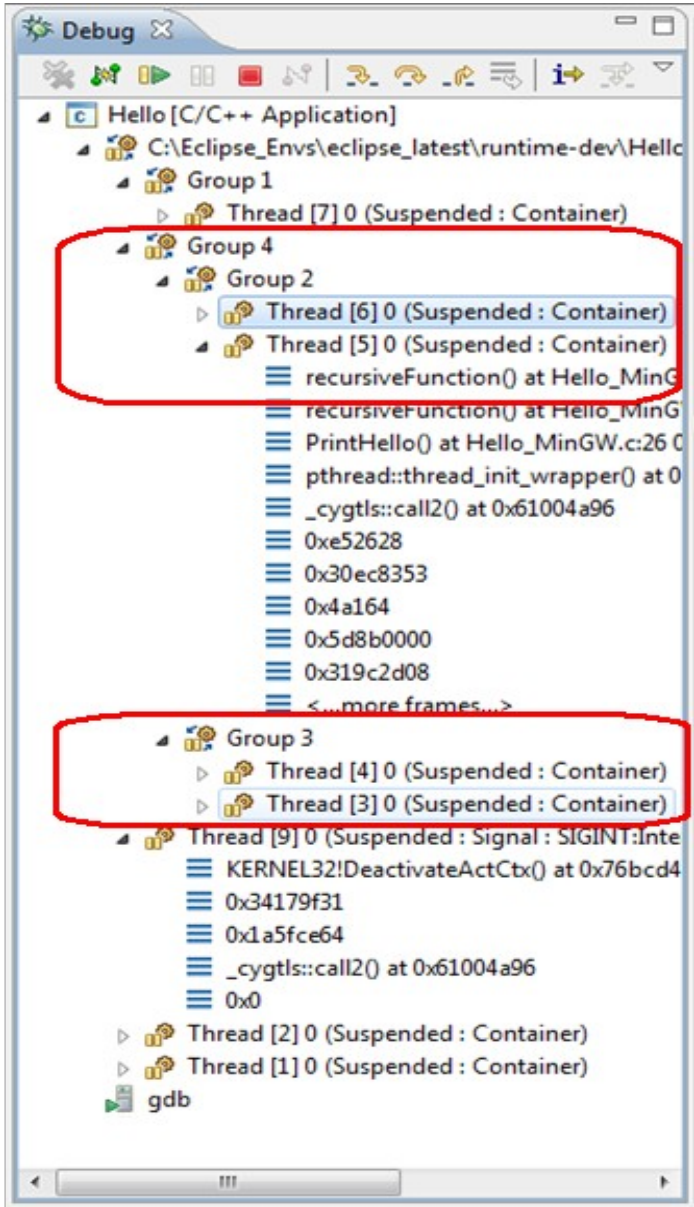
**Applies to every process**

**Auto attach when hit**

**Un-started or short lived process**

# ITSETS

- IT Sets to control groups of elements
- About multicore
  - Step group of threads or processes
  - Set breakpoint on a subset of threads
  - Resume execution on a core or set of cores



# MORE IDEAS

- Improved handling of breakpoints
  - Showing each installed location per breakpoint
  - ...
- Improved Memory view
  - Showing registers and variables
  - …
- Evolving Visualizer
  - Better support when dealing with hundreds of cores
  - ...

# CONCLUSION

# CONCLUSION

- Don't accept *printf-debugging.* This is 2016!
- Debugger will save you time
- Debugging does not have to be difficult
- Help your team improve
  - Lead by example
  - Share knowledge, success stories

eclipsecon
RESTON VIRGINIA 2016

# Evaluate the Sessions

Sign in and vote at **eclipsecon.org**

- 1 0 + 1

# SOME REFERENCES

- CDT Project, http://www.eclipse.org/cdt
- CDT FAQ, http://wiki.eclipse.org/CDT/User/FAQ
- CDT Debug workgroup
  http://wiki.eclipse.org/CDT/MultiCoreDebugWorkingGroup
- CDT Wiki, http://wiki.eclipse.org/CDT

# FINAL Q&A