



ADVANCED C/C++
DEBUGGING (CDT PROJECT)
TRACING (LINUX TOOLS PROJECT)

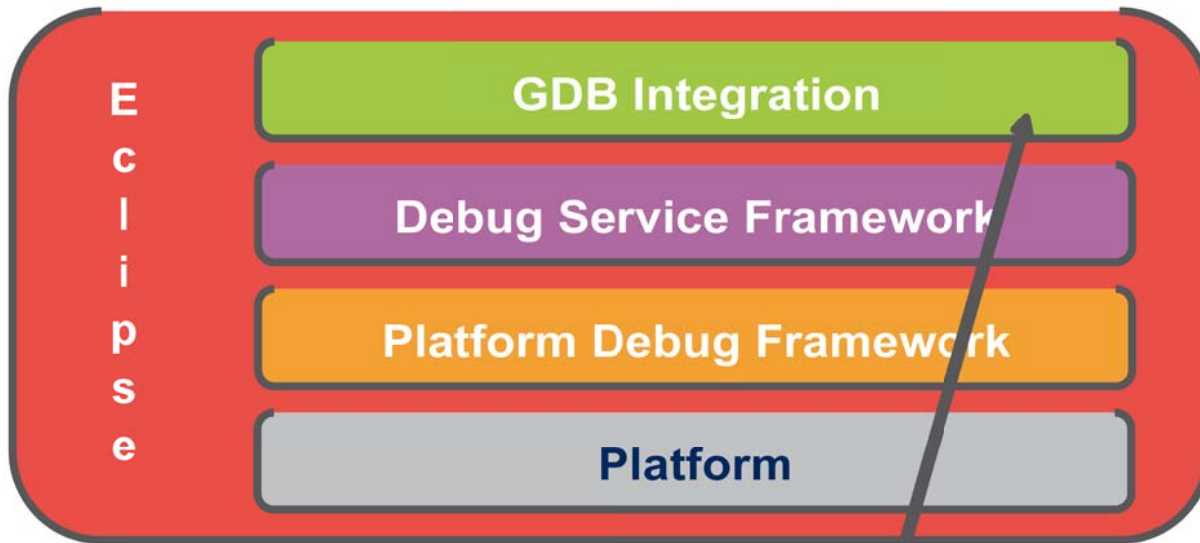
DOMINIQUE DOT TOUPIN AT
ERICSSON DOT COM



FULL OPEN SOURCE SOLUTION

- › ECLIPSE CDT
- › ECLIPSE LINUX TOOLS
- › GDB
- › LTTNG
- › LINUX

CDT DSF-GDB



Open source applicable to all target OS

Open source applicable to all target OS

Host

MI protocol

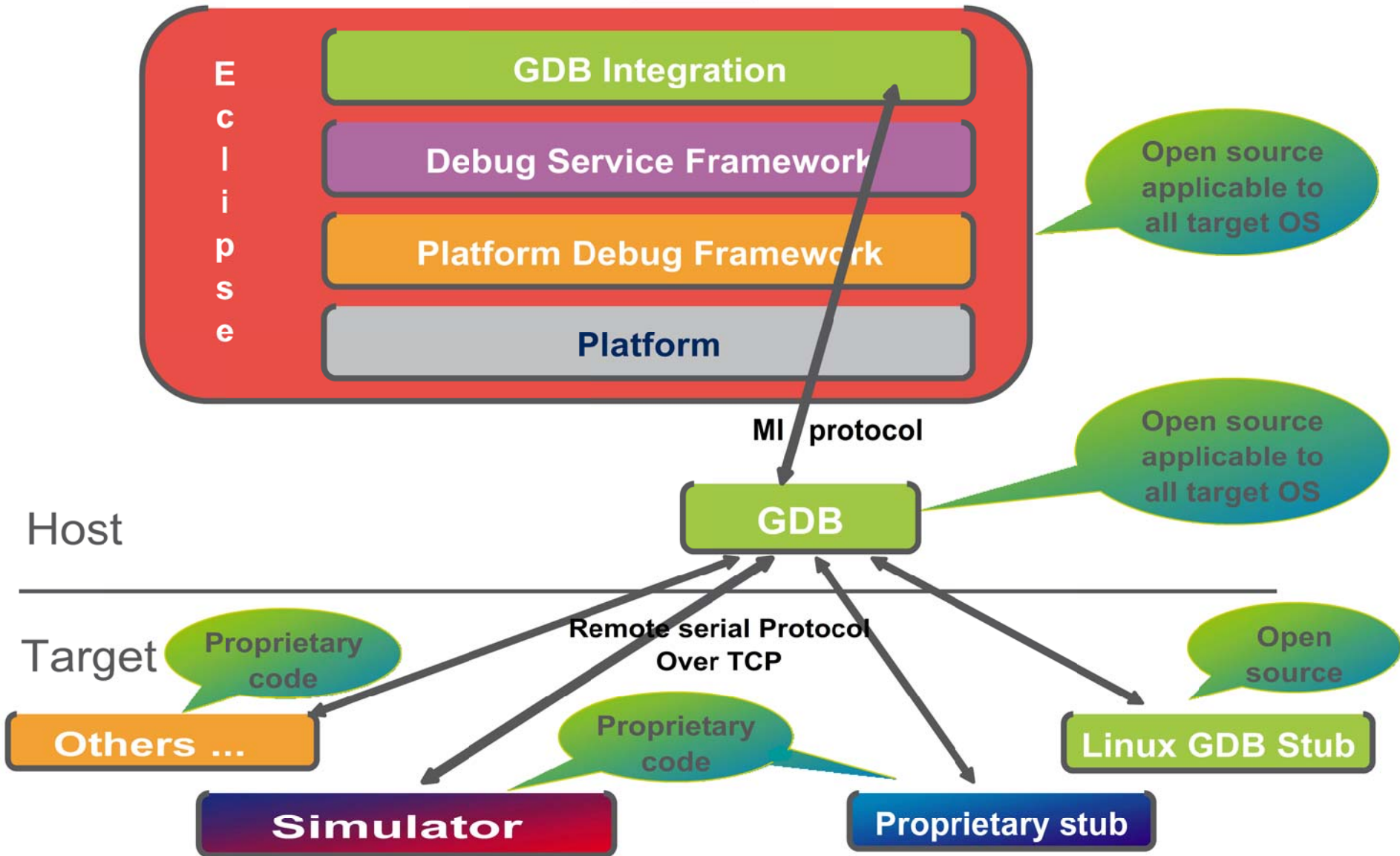


Target

Remote serial Protocol Over TCP

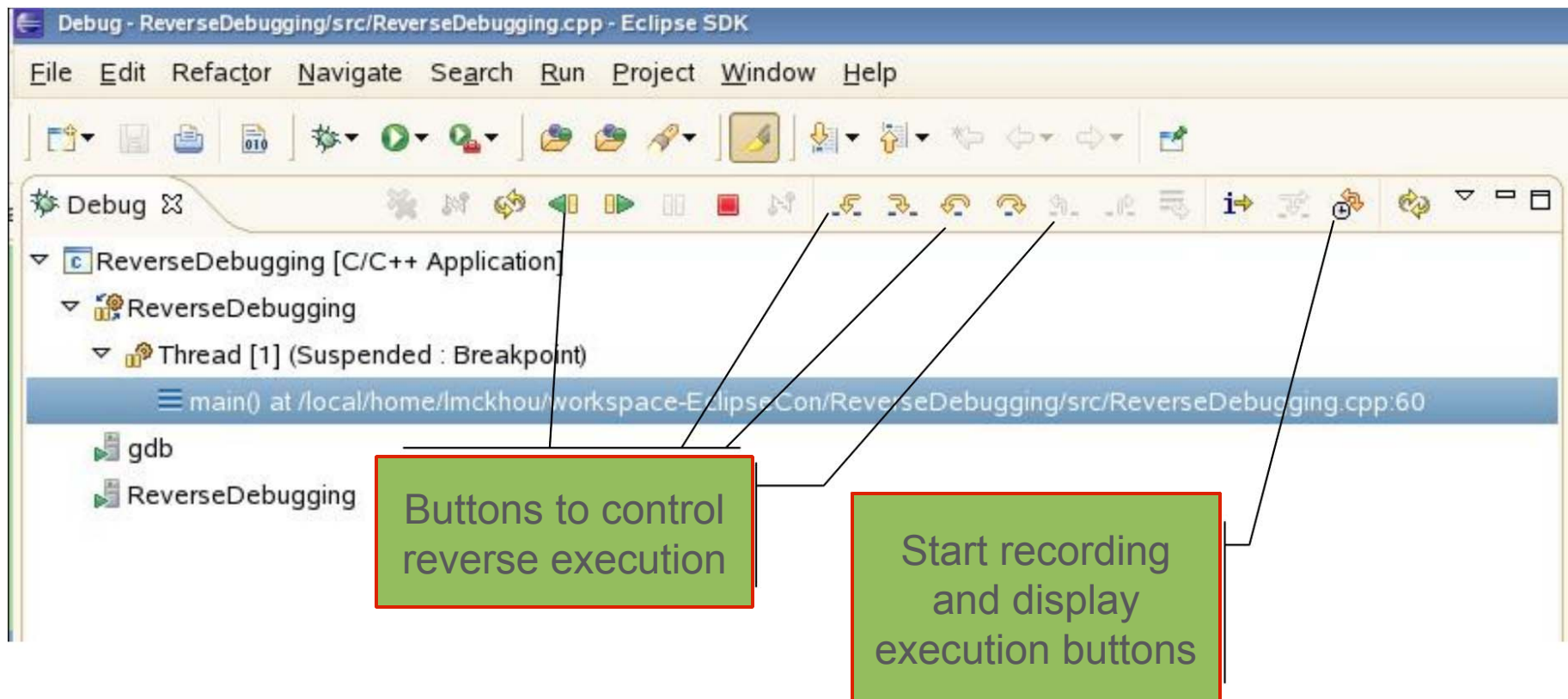


Open source



REVERSE DEBUGGING

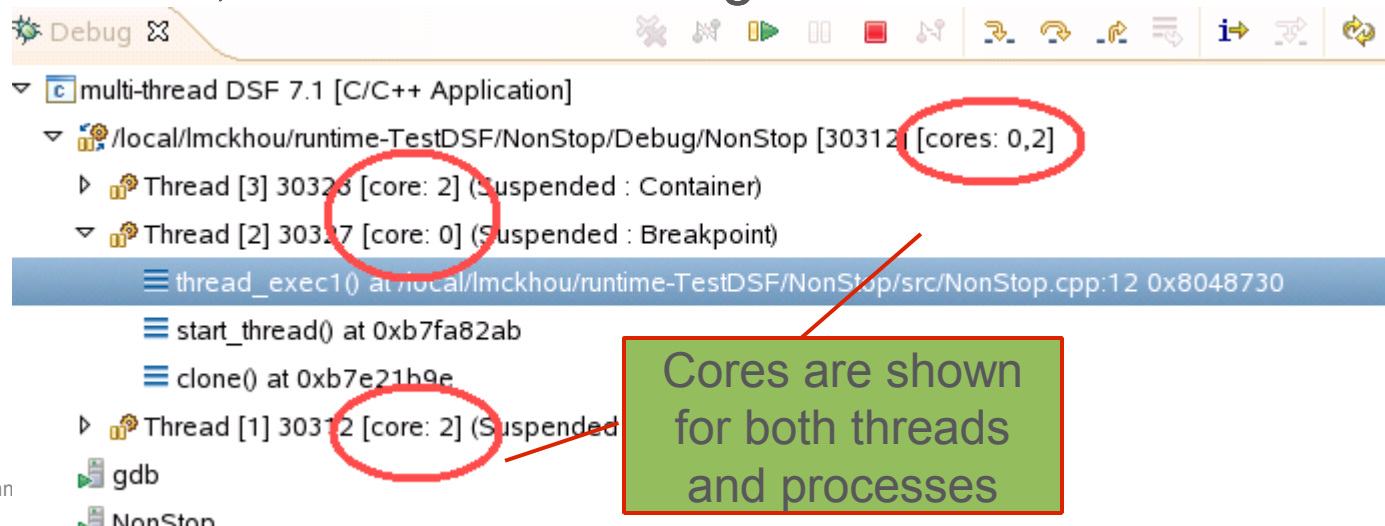
- › Allows to undo register and memory changes so as to move the execution backwards
- › Uses recording and playback



MULTI CORE-PROCESS-CONTEXT

[HTTP://WIKI.ECLIPSE.ORG/CDT/MULTICOREDEBUGWORKINGGROUP](http://wiki.eclipse.org/CDT/MulticoreDebugWorkingGroup)

- › More execution in parallel in many processes
- › Debugging related processes at the same time
- › Dynamically attach and detach from processes
- › Follow child process created with a fork, exec,
- › Global Breakpoint, many processes can execute the same code, auto attached to the process only when the breakpoint is hit, also usefull for short lived-process
- › Core awareness, threads are running on which cores



CDT/designs/MultiCoreDebug - Eclipsepedia - Mozilla Firefox

File Edit View History Bookmarks Favorites Tools Help

http://wiki.eclipse.org/CDT/designs/MultiCoreDebug

CDT/designs/MultiCoreDebug - Eclips...

2.4) View where users can see the cores and what is running on each core, in many cases users will use core affinity to put thread on specific core for execution locality. The threads don't move much between cores.

2.5) Define what are the process/threads for an application, attach to all of them;

2.6) Select one core and stop it or put a breakpoint on all the threads running on it; or a global breakpoint for the first process or thread hitting the breakpoint;

2.7) Continue/step for all thread running on a core;

2.8) Understand the interaction between process/threads of different cores, e.g. signal breakpoint;

2.9) A few use cases/features are almost the same as the one in PTP (process set, operations on set, etc.)
<http://www.eclipse.org/ptp/documentation/2.1/org.eclipse.ptp.help/html/06parDebugging.html>

■ **3) Systems with multiple cores (On Chip Debugging)**

3.1) Hardware bring-up debugging
User attaches a debugger to a multi-core system for purpose of hardware configuration, such as configuring registers, IO devices, etc. (no OS, no symbol data, no executable image)

3.2) No-OS debugging
User attaches to a multi-core system and downloads simple executable image to each core and debugs them.

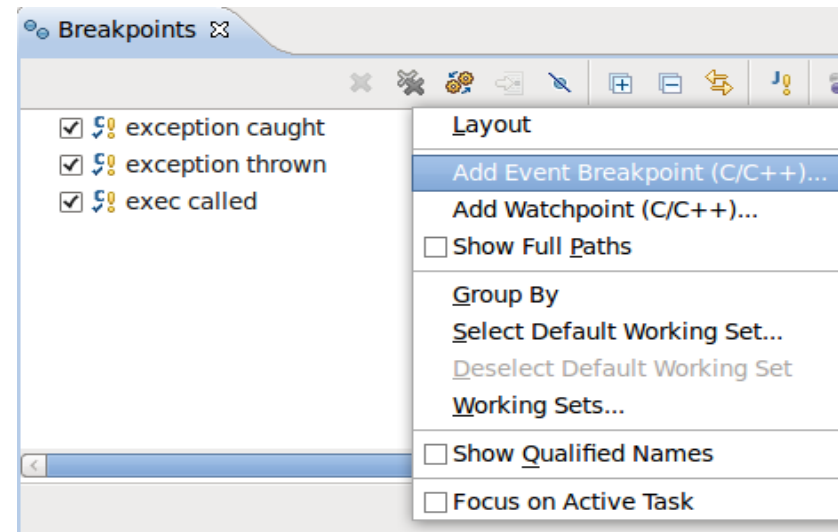
3.3) Debugging with OS-awareness
User attaches to a multi-core system, downloads an OS image to each, starts debugging the entire OS on each board, or specific tasks/processes running on specific cores.

3.4) Synchronized run control operations
Some multi-core debuggers allow for performing synchronized operations on multiple cores. Allowing the debugger or the hardware to operate on multiple cores minimizes the skid, or the time lag, in when the operations are carried out on the different cores. There needs to be dedicated UI and APIs to exercise these debugger features.

Done

SPECIAL BREAKPOINTS

- › Conditional Breakpoint
 - Stop only if the condition is true.
 - C assert condition, break when assertion is false
- › Data Breakpoint or Watchpoint
 - Stop whenever the value of an expression change
 - Don't have to predict where this may happen
 - Can be a complex expression or just a single variable
- › Program event breakpoint
 - Stop when a special event occurs
 - Throwing/catching C++ exception,
 - unhandled exception
 - call to exec, fork, syscal



- Display and editing of complex objects like Lists, Maps, Vectors

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,
Default:{...}
Decimal:{...}
Hex:{...}
Binary:{...}
Octal:{...}

Complex structures shown in an intuitive way and editable

- Some programs have a deep interaction with OS resources DSF-GDB can show: process groups, file descriptors, internet-domain sockets, shared memory segments, semaphore, message queues, loaded kernel modules, etc.

NON-STOP

- › Debugging a process by stopping its execution might cause the program to change its behavior drastically, or perhaps fail, even when the code itself is correct.
 - Troubleshooting in the lab
 - Chasing a race condition
 - Debugging problems happening only under heavy load
 - Investigating user interface issues

- › Non-Stop allows to stop and examine one or more thread in the debugger while other threads continue to execute freely

DYNAMIC TRACEPOINTS

- › Tracepoint collects user-specified info and continues execution without stopping any thread, essential for live sites
- › Dynamic i.e. inserted with a jump (in process) or a trap
- › Data collection can be conditional to a user specified expression
- › Tracepoint actions:
 - collect state trace data e.g. timestamp, and program data e.g. variables, register
 - evaluate expressions , e.g. modify trace variables
 - step (similar to breakpoint step) and collect data in each step
- › A trace experiment can be stopped after the n'th hit
- › Static tracepoint (LTTng UST) can be stored in the debug tracepoint buffer
- › Debug tracepoint are good when no static tracepoint are available and for small quantity of data

DYNAMIC TRACEPOINTS VISUALIZATION

Intuitive display using debugger views

The screenshot displays the Eclipse IDE interface for debugging a C/C++ application. The main window shows the source code of `NonStop.cpp` with a breakpoint set at line 44. The `Variables` view shows the state of the program at the breakpoint, with the value of `i` highlighted as 3. The `Breakpoints` view shows the active breakpoint at `NonStop.cpp [line: 44] [condition: i < 10]`. The `Trace Control` panel shows that tracing is currently not active.

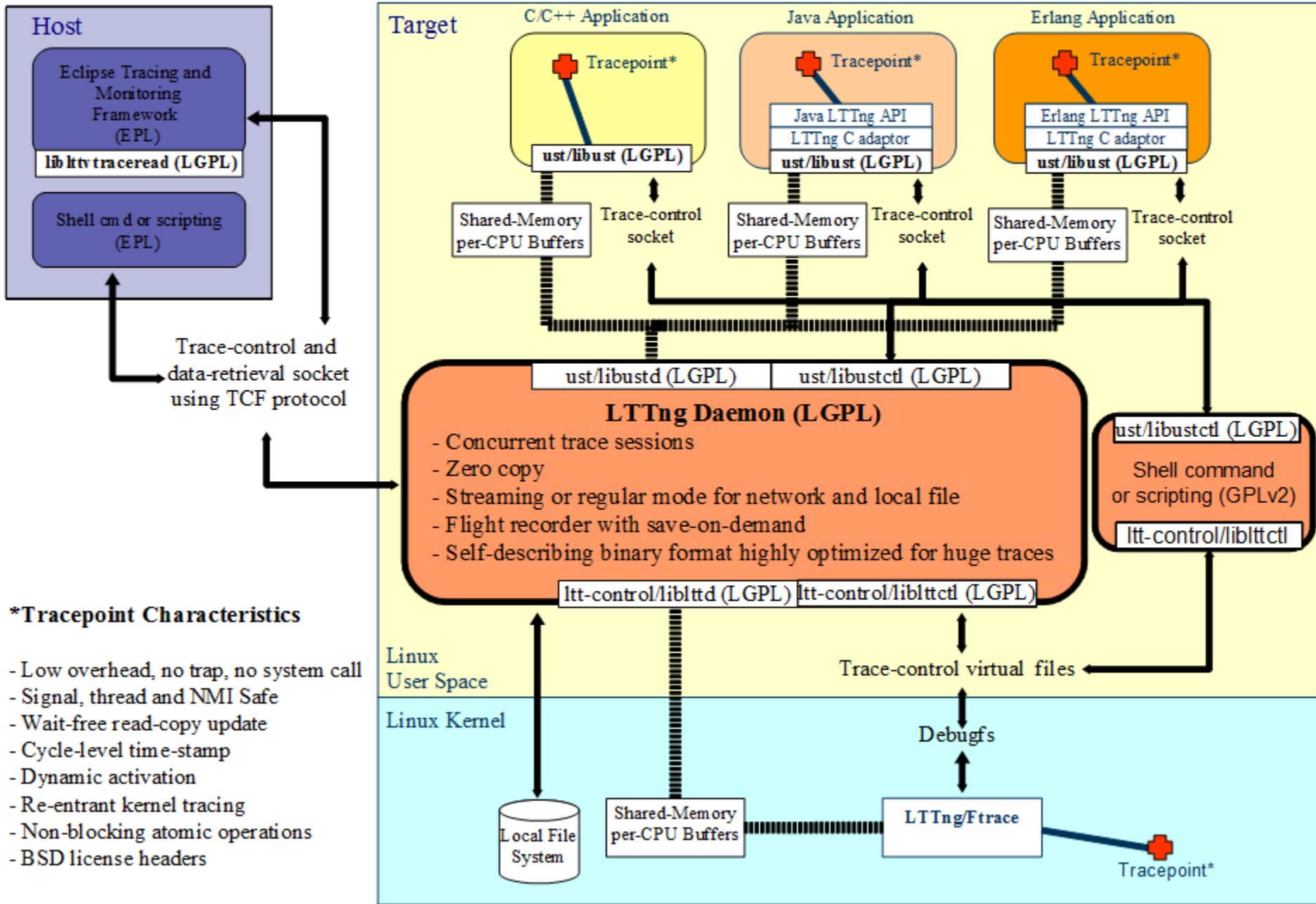
Name	Type	Value
i	int	3
thread3	pthread_t	3066370928
thread	unsigned long [30]	0xbfcc88ac
message3	char *	0x8048821 "Thread 3"
iret3	int	0
thread2	pthread_t	3074763632
message2	char *	0x8048818 "Thread 2"
iret2	int	0

```
40  iret2 = pthread_create( &thread2, NULL, thread_exec1, (void*) message2);
41  iret3 = pthread_create( &thread3, NULL, thread_exec2, (void*) message3);
42
43  for (int i=0;i<15;i++) {
44  printf("another loop\n");
45  sleep(1);
46  pthread_create( &thread[i], NULL, thread_exec3, (void*) &i);
47  }
48  /* Wait till threads are complete before main continues. Unless we
49  /* wait we run the risk of executing an exit which will terminate
50  /* the process and all threads before the threads have completed. */
51
52  pthread_join(thread2, NULL);
53  printf("Thread 2 returns: %d\n", iret2);
54
55  pthread_join(thread3, NULL);
56  printf("Thread 3 returns: %d\n", iret3);
57
58  return 0;
59 }
```

TRACING

- › Need to understand what is going on in a system without causing disturbance? → Tracing is for you
- › Compared to logging, tracing typically records lower-level events that occurs much more frequently
- › Tracers are therefore optimized to handle a lot of data while having a small impact on the system
- › Static Tracepoint
 - created by designer before compilation
 - represent wisdom of developers who are most familiar with the code
 - The rest of the world can use static tracepoint to extract a great deal of useful information without having to know the code

LTTng Low-Overhead Tracing Architecture



*Tracepoint Characteristics

- Low overhead, no trap, no system call
- Signal, thread and NMI Safe
- Wait-free read-copy update
- Cycle-level time-stamp
- Dynamic activation
- Re-entrant kernel tracing
- Non-blocking atomic operations
- BSD license headers

LTTNG PERSPECTIVE



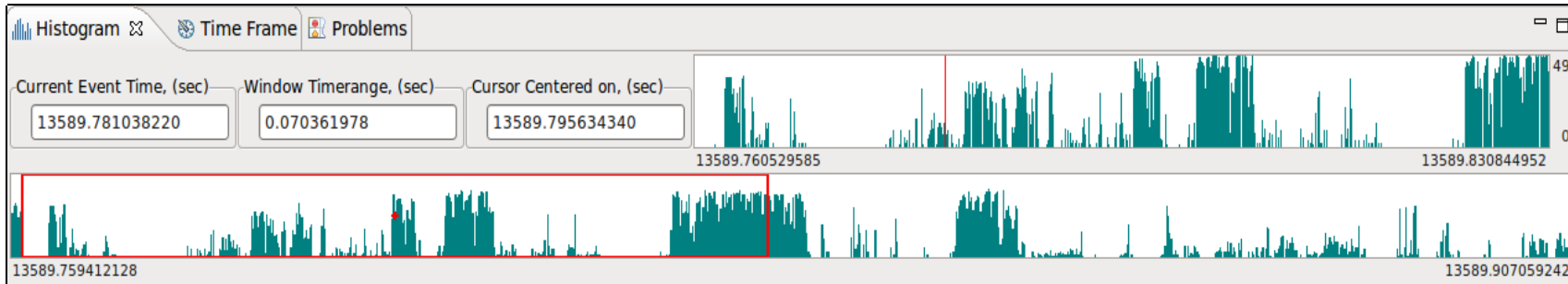
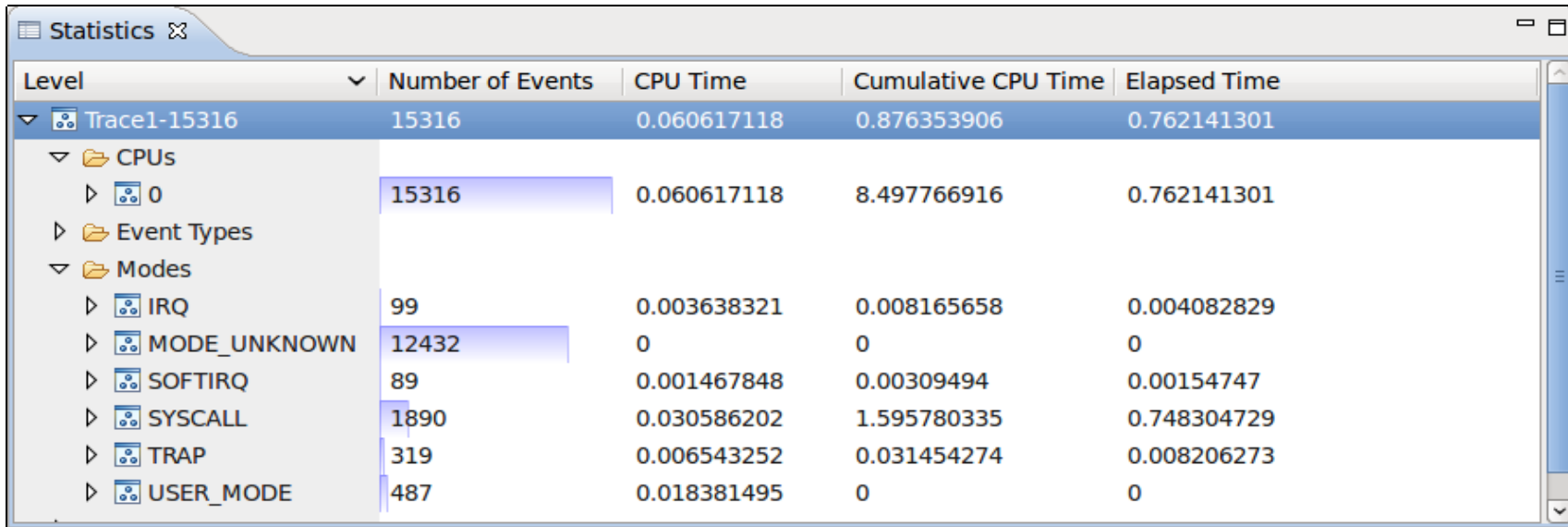
The screenshot displays the LTTng GUI interface with several panels:

- Control Flow:** A table showing process execution details for Trace1-15316.
- Resources:** A timeline view for CPU 0 and various IRQs (1, 239, SOFT_IRQ 1, 6, 9, TRAP 14) over a time scale from 13589:760 to 13589:790.
- Events - MyExperiment1:** A table of kernel events with columns for Timestamp, Source, Type, Reference, and Content.
- Statistics:** A tree view showing the number of events for various categories like CPUs, Event Types, and global_state.
- Histogram:** A bar chart showing event frequency over time, with a time frame from 13589:759412128 to 13589:907059242.

Process	pid	ppid	tid	pid	ppid	tid	Trace
sendmail	16867	16867	16857	0	13589	763749454	Trace1-15316
ping	16885	16885	16865	0	13589	763752479	Trace1-15316
ltd	16887	16887	30068	0	13589	763755140	Trace1-15316
UNNAMED	16888	0	0	0	0	000000000	Trace1-15316
ltd	16889	16889	1	0	13589	763758054	Trace1-15316
ltd	16890	16889	16889	0	13589	781038220	Trace1-15316
/usr/local/bin/ltd	16891	16891	30068	0	13589	873239052	Trace1-15316
udev	18054	18054	1	0	13589	763696784	Trace1-15316

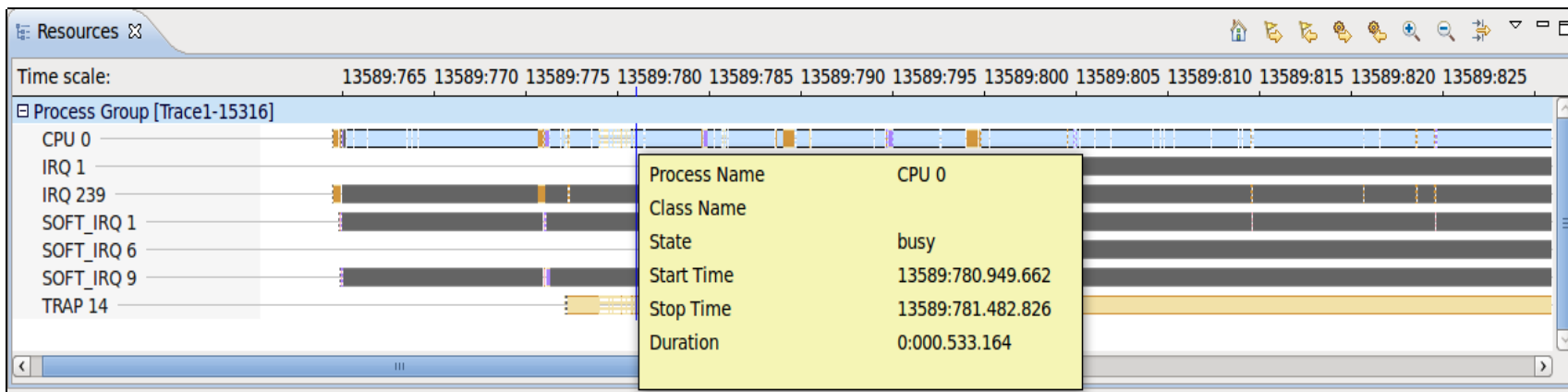
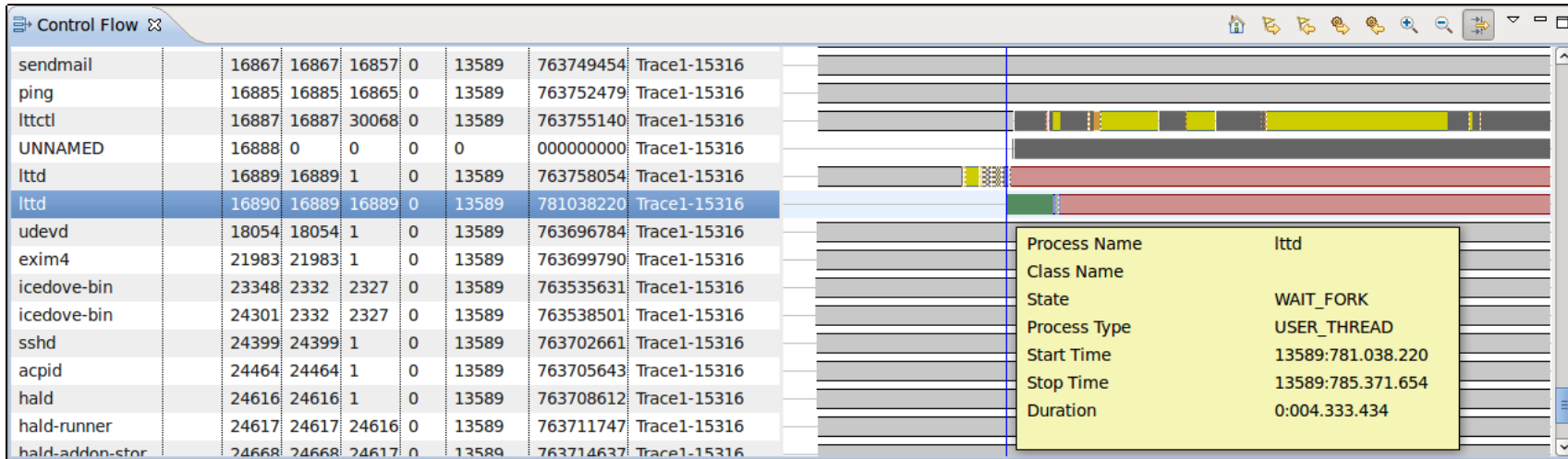
Timestamp	Source	Type	Reference	Content
13589.780940723	Kernel Core	kernel/0/page_fault_entry	Trace1-15316	write_access:0,address:0xb75961ac,ip:0xb770ebe0,trap_id:14
13589.780949662	Kernel Core	kernel/0/page_fault_exit	Trace1-15316	res:512
13589.780959249	Kernel Core	kernel/0/page_fault_entry	Trace1-15316	write_access:0,address:0xb765b220,ip:0xb765b220,trap_id:14
13589.780966470	Kernel Core	kernel/0/page_fault_exit	Trace1-15316	res:512
13589.780974235	Kernel Core	kernel/0/syscall_entry	Trace1-15316	syscall_id:120,ip:0xb765b268
13589.781038220	Kernel Core	kernel/0/process_fork	Trace1-15316	child_pid:16890,child_tgid:16889,parent_pid:16889
13589.781073500	Kernel Core	kernel/0/sched_wakeup_new_task	Trace1-15316	cpu_id:0,state:0,pid:16890
13589.781342500	Kernel Core	kernel/0/syscall_exit	Trace1-15316	ret:16890
13589.781482826	Kernel Core	kernel/0/page_fault_entry	Trace1-15316	write_access:0,address:0xb76df630,ip:0xb76df630,trap_id:14
13589.781499635	Kernel Core	kernel/0/page_fault_exit	Trace1-15316	res:512

LTTNG HISTOGRAM, STATISTICS

Level	Number of Events	CPU Time	Cumulative CPU Time	Elapsed Time
Trace1-15316	15316	0.060617118	0.876353906	0.762141301
CPUs				
0	15316	0.060617118	8.497766916	0.762141301
Event Types				
Modes				
IRQ	99	0.003638321	0.008165658	0.004082829
MODE_UNKNOWN	12432	0	0	0
SOFTIRQ	89	0.001467848	0.00309494	0.00154747
SYSCALL	1890	0.030586202	1.595780335	0.748304729
TRAP	319	0.006543252	0.031454274	0.008206273
USER_MODE	487	0.018381495	0	0

LTTNG CONTROL FLOW, RESOURCES



UPCOMING FEATURES

> General

- Tracing tool control
- Trace streaming
- Heterogeneous traces
- GDB Tracepoints
- Source lookup
- Performance tuning

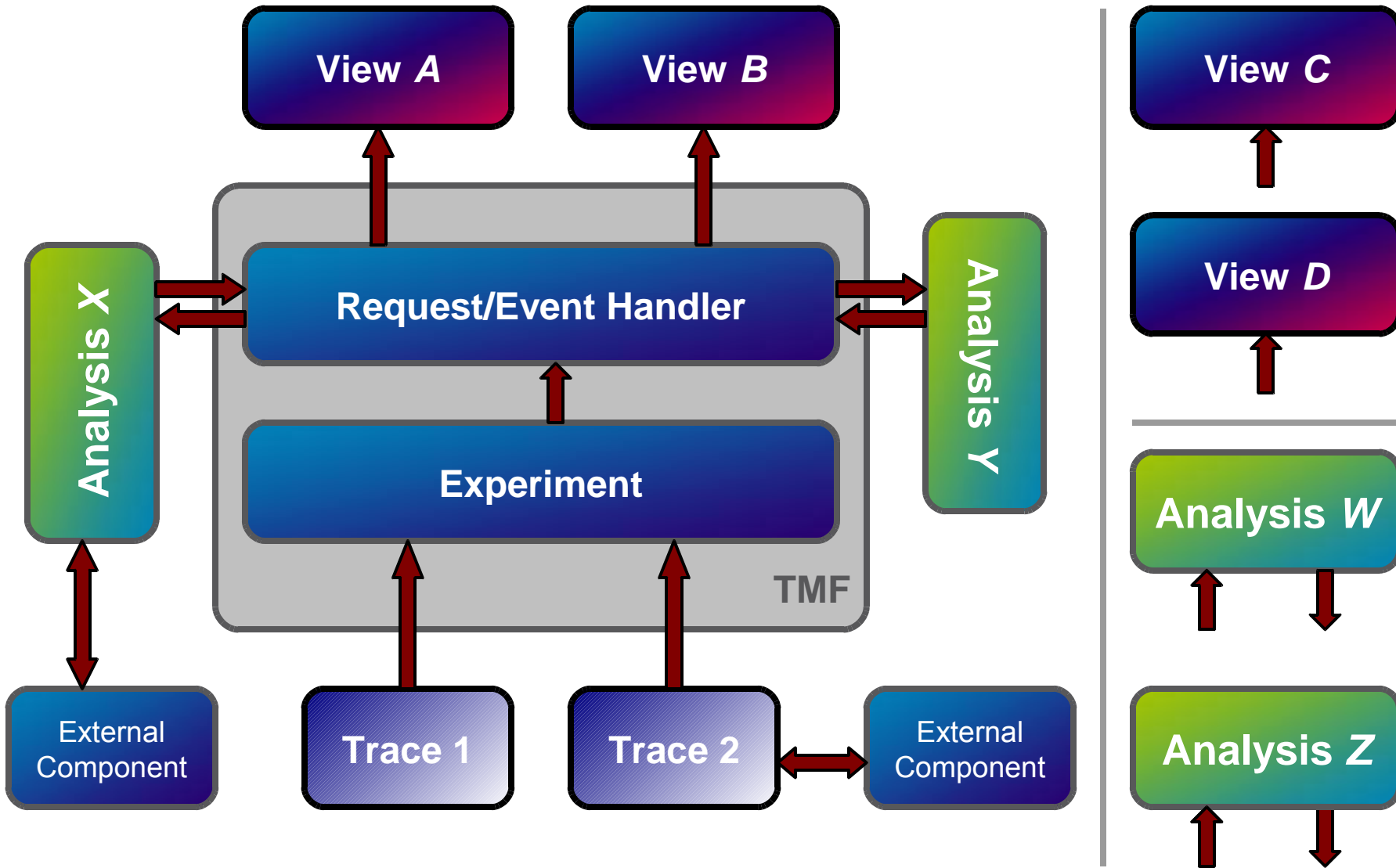
> Analyses

- Time correction (traces synchronization)
 - > Multi-core, multi-level, multi-node
- Timing dependencies (between processes)
- Latency Analysis
- Pattern matching (security e.g. intrusion detection)

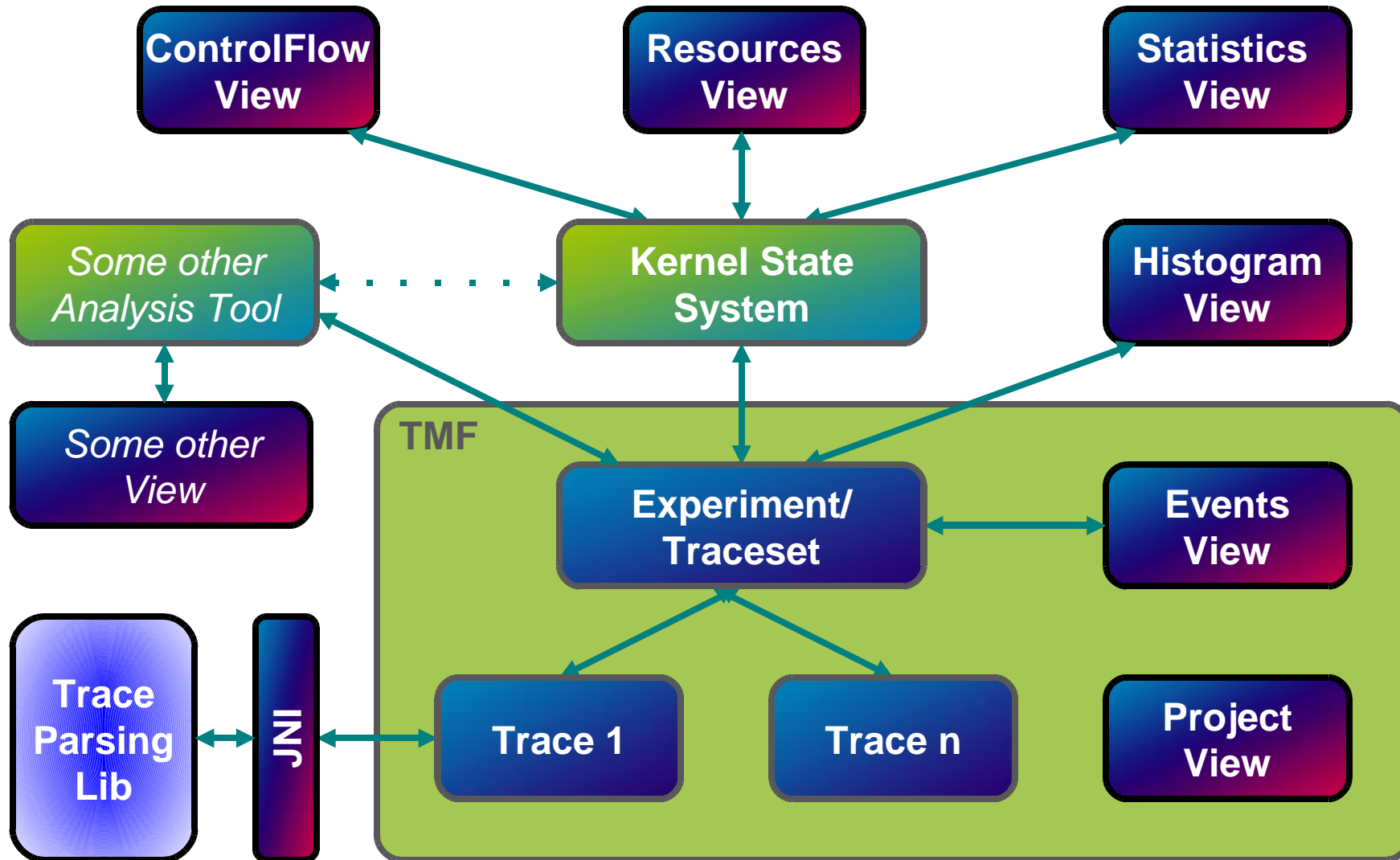
> Other trace format

- Linux User Space Tracing
- Text format
- De-facto standard format Multi-core association, Embedded Linux Forum, Samsung, Ericsson, Mentor Graphic, WindRiver, IBM, Freescale, TI, Nokia-Siemens Network, National Instruments, etc.
www.multicore-association.org/workgroup/tiwg.php
Common Trace Format Requirement:
<http://lwn.net/Articles/408824/>
Common Trace Format Implementation:
<http://lwn.net/Articles/408825/>

TMF - ARCHITECTURE



LTTNG - TMF INTEGRATION



ADDITIONAL ONLINE RESOURCES

- › Eclipse CDT DSF-GDB lead: **marc DOT khouzam AT ericsson DOT com**
cdt-dev@eclipse.org
- › CDT Multi-core debugging <http://wiki.eclipse.org/CDT/designs/MultiCoreDebug>,
<http://wiki.eclipse.org/CDT/MultiCoreDebugWorkingGroup>, <http://wiki.eclipse.org/PinAndClone>
- › <http://gcc.gnu.org/wiki/summit2010>
- › Advanced Tracing Features using GDB and LTTng, Real-time debugging using GDB
Tracepoints, GDB Tracepoints: From Prototype to Production
- › <http://gcc.gnu.org/wiki/HomePage?action=AttachFile&do=get&target=2009-GCC-Summr>
Using Eclipse for Reverse, Multi-Process and Non-Stop Debugging with GDB p.65,
GDB Tracepoints, Redux p.105, Hybrid multi-architecture debugging with GDB p.137
- › <http://www.gccsummit.org/2008/gcc-2008-proceedings.pdf> Non-stop Multi-Threaded
Debugging in GDB p.117
- › Eclipse LTTng plug-in lead: **francois DOT chouinard AT ericsson DOT com**
linuxtools-dev@eclipse.org
- › <http://www.eclipse.org/linuxtools/projectPages/ltnng>
- › <http://www.lttng.org>, <http://lttng.org/content/success-stories>



ADVANCED C/C++
DEBUGGING (CDT PROJECT)
TRACING (LINUX TOOLS PROJECT)

DOMINIQUE DOT TOUPIN AT
ERICSSON DOT COM