ECEN 5623

Performance Tuning and Debugging

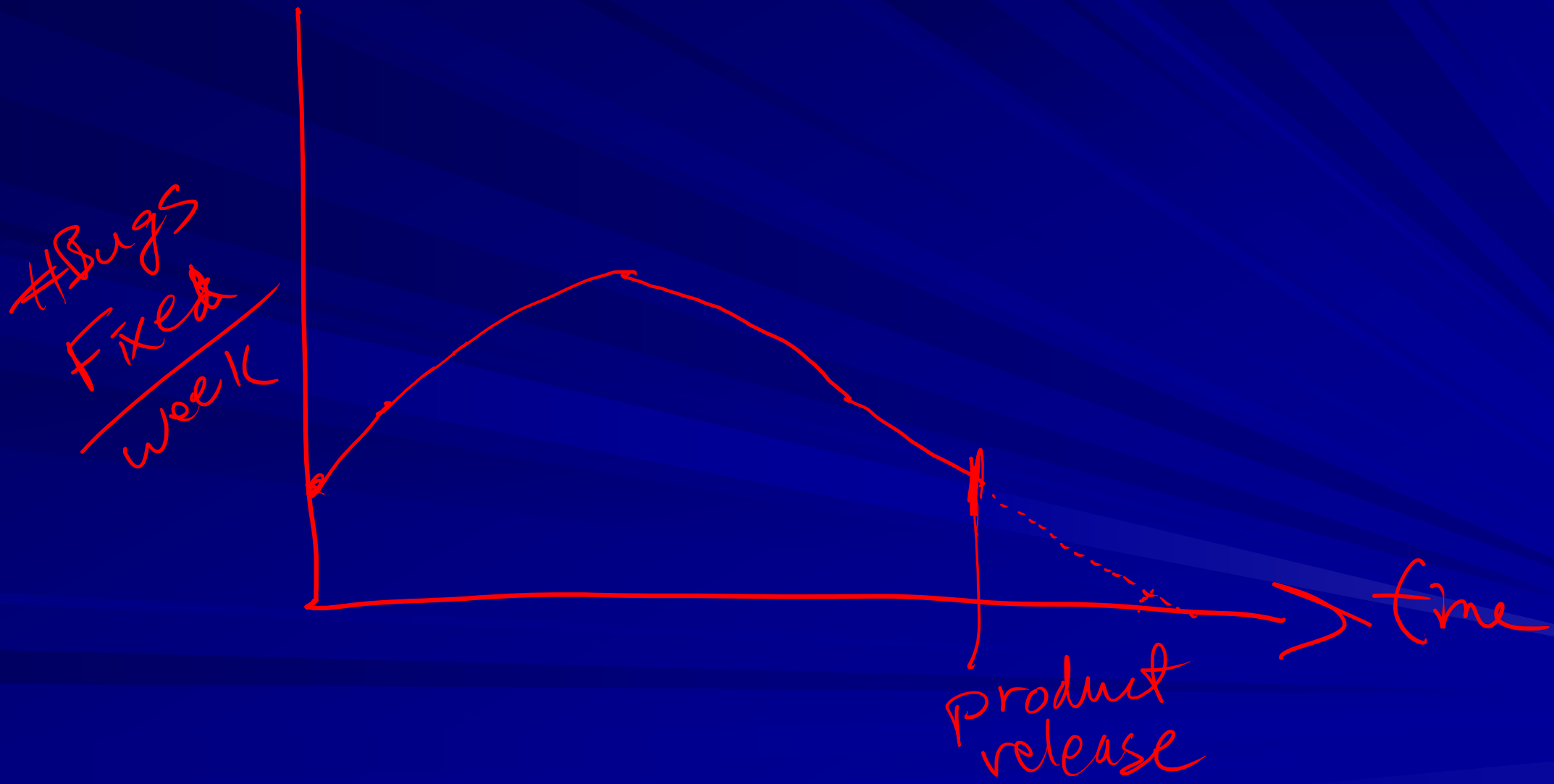
Be Boulder.

University of Colorado Boulder

RT Embedded System Performance/Debug

- Performance/Debug Concepts
  - Software-In-Circuit (SWIC) and Hardware-In-Circuit (HWIC) Methods
  - Single-Step Debugging
  - Full-Speed Tracing
    - Execution and Memory Reference Tracing
    - Service/Thread Scheduler and RTOS Event Tracing
    - Statistical Performance Analysis
  - Test-Vector/Code Coverage Verification
- Common Tools Used
  - ICE (In-Circuit Emulator) – In-Cache and SoC Issues!
  - Logic Analyzer (External Memory Trace, Trace-Port Acquisition, I/O)
  - JTAG Debugger (BDM, COP, OCD, OnCE)
  - Built-In Trace-Port and/or Internal Analyzer – E.g. Xilinx Chip-Scope
  - OS Event Trace Tools – E.g. WindView, Linux Trace Toolkit
  - SWIC/HWIC Tools – E.g. CodeTest, HP Caliper, Intel VTune
  - Built-In Performance Monitor – E.g. Intel XScale/Pentium PMU
- Analysis Methods
  - Source Code Correlation, Thread/Event Gantt Charts
  - Histograms for Hot-Spot Analysis

Progress of Bug elimination



RT Embedded System Performance/Debug

- Performance/Debug Concepts
  - Software-In-Circuit (SWIC) and Hardware-In-Circuit (HWIC) Methods
  - Single-Step Debugging
  - Full-Speed Tracing
    - Execution and Memory Reference Tracing
    - Service/Thread Scheduler and RTOS Event Tracing
    - Statistical Performance Analysis
  - Test-Vector/Code Coverage Verification
- Common Tools Used
  - ICE (In-Circuit Emulator) – In-Cache and SoC Issues!
  - Logic Analyzer (External Memory Trace, Trace-Port Acquisition, I/O)
  - JTAG Debugger (BDM, COP, OCD, OnCE)
  - Built-In Trace-Port and/or Internal Analyzer – E.g. Xilinx Chip-Scope
  - OS Event Trace Tools – E.g. WindView, Linux Trace Toolkit
  - SWIC/HWIC Tools – E.g. CodeTest, HP Caliper, Intel VTune
  - Built-In Performance Monitor – E.g. Intel XScale/Pentium PMU
- Analysis Methods
  - Source Code Correlation, Thread/Event Gantt Charts
  - Histograms for Hot-Spot Analysis

RT Embedded System Performance/Debug

- Performance/Debug Concepts
  - Software-In-Circuit (SWIC) and Hardware-In-Circuit (HWIC) Methods
  - Single-Step Debugging
  - Full-Speed Tracing
    - Execution and Memory Reference Tracing
    - Service/Thread Scheduler and RTOS Event Tracing
    - Statistical Performance Analysis
  - Test-Vector/Code Coverage Verification
- Common Tools Used
  - ICE (In-Circuit Emulator) – In-Cache and SoC Issues!
  - Logic Analyzer (External Memory Trace, Trace-Port Acquisition, I/O)
  - JTAG Debugger (BDM, COP, OCD, OnCE)
  - Built-In Trace-Port and/or Internal Analyzer – E.g. Xilinx Chip-Scope
  - OS Event Trace Tools – E.g. WindView, Linux Trace Toolkit
  - SWIC/HWIC Tools – E.g. CodeTest, HP Caliper, Intel VTune
  - Built-In Performance Monitor – E.g. Intel XScale/Pentium PMU
- Analysis Methods
  - Source Code Correlation, Thread/Event Gantt Charts
  - Histograms for Hot-Spot Analysis

SWIC vs. HWIC or Combination?

- SWIC
  - From a Firmware/Software Viewpoint, more Intrusive
    - Code is Modified at Source-Level or Machine Code Inserted with Binary Annotation to Trace Events
    - Changes Timing and Memory Reference Stream
    - However, Can Be Minimal and In Many Cases Insignificant
    - Simple Memory-Port or Trace Buffer Writes Inserted
      - Events Encoded Into Single 32-bit/64-bit/Cache-Line Write
      - Each Event Recorded May Only Require a Single CPU Cycle
    - E.g. WindView, Linux Trace Toolkit, Ftrace, SystemViewer, System Tap.
- HWIC
  - From a Firmware/Software Viewpoint, no Intrusion!
  - Impacts HW Design Significantly – Must Be Built-In and Requires HW Resources
    - E.g. IBM 4xx Series, ARM 9, 11, 12, and Tensilica Trace-Ports
    - E.g. Xilinx Virtex-II Chip-Scope
    - E.g. JTAG BDM TAP
- SWIC/HWIC Combined
  - E.g. Logic-Analyzer Event Trace
    - Binary Annotation to Write Events to Memory Port
    - Acquisition by External Logic-Analyzer

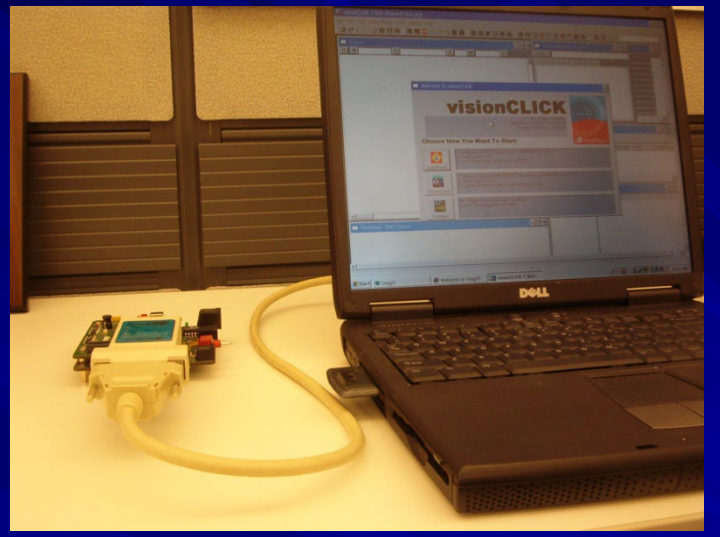
HWIC Single-Step Debugging

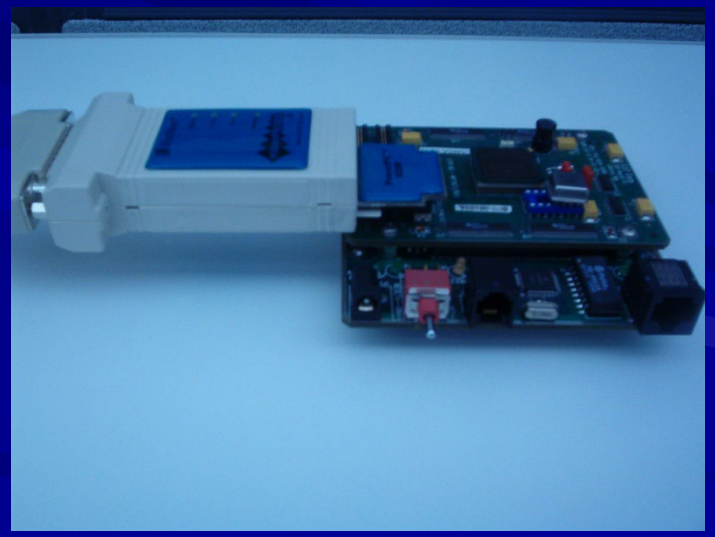
- JTAG TAP – IEEE 1149.1
  - Derived from Original JTAG Boundary Scan Chain
    - Used for Board/Device Factory Verification
    - TDI/TDO Shift Register Interface for Test Data In/Out
      - Verify Bits Shifted Out of Chain for Given Bit Pattern Shifted In
      - BSDL – an HDL use to Specify Scan Chain
  - TAP Added For Firmware (Boot-Strapping) Debug
  - Replaces or Enhances PROM Monitor
  - Obviates “Burn and Learn” and/or PROM Monitor Boot-Strapping

- CPU Clocked/Controlled By External JTAG – HW Single-Step
- Many Variants and Extensions of JTAG TAP
  - BDM (Background Debug Mode), OnCE (On-Chip Emulator), OCD (On-Chip Debugger)
  - Variants/Extensions Provide HW-Supported Features
    - Internal Break-points (HW raised Debug EXC on PC-Addr Match)
    - Debug SRAM or Instruction Cache for JTAG controlled Debug Handler
    - Hot-Debug
- Read “Zen of BDM” for History and Background

JTAG TAP Tools

- JTAG Probe
  - Host Interface
    - Parallel, Serial, USB, Ethernet Port Connection to Host
    - Translation of Debugger Commands to JTAG TDI/TDO and JTAG Commands for TAP Execution
  - Debugger Tool
    - Traditional Single-Step Debugger Interface – E.g. DDD/GDB
    - Extensions for Core Specific Registers and Features
  - JTAG Personality Module
    - Conforms to Specific JTAG Pin-out (10, 16, 20 pin)
    - Interface to JTAG Standard Signals: TDI, TDO, TRST, TMS, TCK

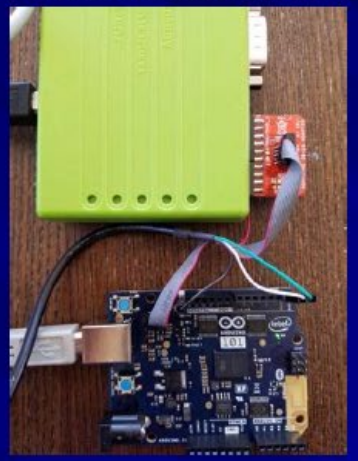




Modern JTAG & USB ICDI Tools

- Most JTAG interfaces are on-chip, USB host interface
  - Rather than a USB to JTAG Bridge Device
  - E.g. Jetson – JetPack or JTAG
- TIVA – 10-pin header
  - IEEE Standard
  - Or USB to JTAG Bridge and Monitor (ICDI – In Circuit Debug Interface)
- Intel Curie/Quark IoT

Zephyr with Flyswatter



Jetson with Lauterbach JTAG

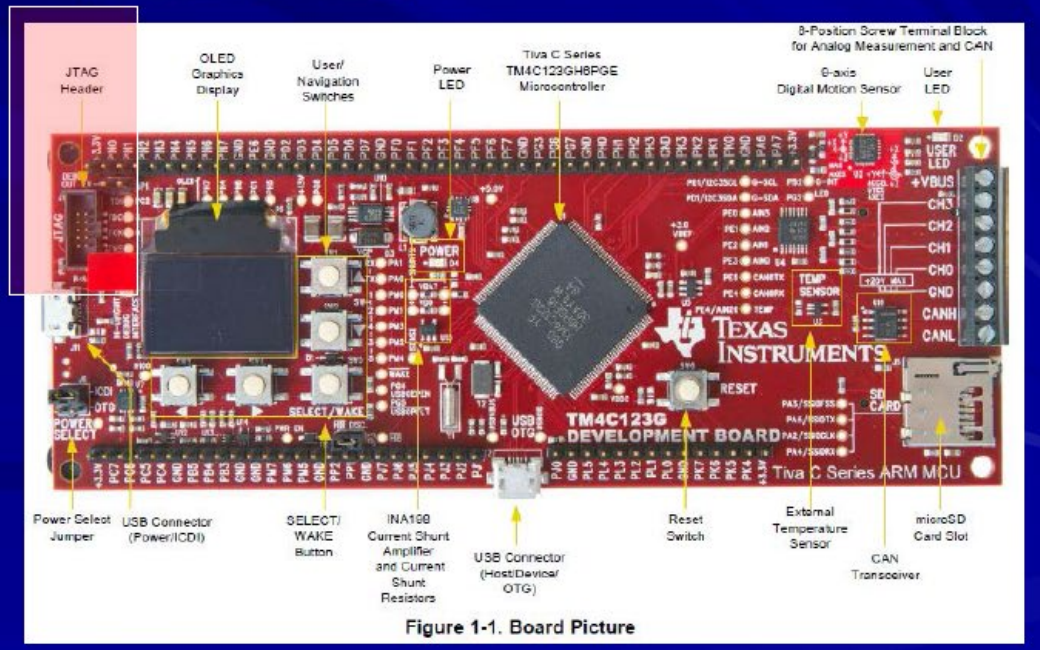


Figure 1-1. Board Picture

SWIC Single-Step Debugging

- System-Level Debug
  - Single Step Debug of Entire RTOS and All Threads
  - Requires Non-Interrupt I/O Interface
    - Polled Serial
    - Polled Ethernet (e.g. VxWorks END Driver)
  - Similar to JTAG Debugging, But CPU Running Full-Speed, All Firmware is in Single-Step Mode
  - Debug Exception Used to Break After Each Line of Code
  - Alternative to JTAG for Debugging Firmware
- Service/Thread/Task-Level Debug
  - Debugger Attached to Single Service (Task or Thread)
  - Other Services Run Normally
  - Debug EXC Raised for Each Line of Code in Attached Task
  - Similar to Unix/Windows Application Debug

HWIC Probe Full-Speed Tracing

- ICE Traditionally Provides Full-Speed Core Trace
  - In Addition to HWIC Single-Step Debug (May Be JTAG)
  - Historically Provided with a CPU Bond-Out Interface
  - Snoops Every CPU ASIC External Pin I/O
    - External Interrupts
    - External Memory References
    - Dedicated I/O Pins (E.g. GPIO)
  - Provides DSP for Snooping and Buffering of Trace Data
  - May Shadow CPU Execution and State with Similar CPU Core in an Emulator
- Value of ICE Diminished By SoC and On-Chip Cache
  - ICE Can't See On-Chip Memory or Cache Accesses
  - ICE Can't See Internal Bus or Multi-Core SoC (Signals Not External)
- Trace-Port Alternative for SoC and On-Chip Cache Systems

HWIC Built-In Full-Speed Trace

- Trace-Port
  - At Core Clock Rate
  - Program-Counter for Each Core
  - Data Address References for Each Core or Internal Bus
  - Typically Encoded to Reduce Pin Count
    - E.g. XScale Branch Trace 8 Bit Encoding
    - ARM Branch Trace
- Internal Logic Analyzer
  - High Speed Low Voltage Probing is Difficult
  - Pin Count for External Probing Interfaces is a Problem
  - Alternatively Build a Logic Analyzer Into SoC
  - Dump ILA Buffer via Firmware or JTAG
  - E.g. Xilinx Chip-Scope

What is the most productive approach for debugging RTES?

A. SWIC
B. HWIC
C. Combination HWIC and SWIC
D. PrintK and read the source

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

SWIC Full-Speed Event Tracing

- Insertion of Code at Source or Binary Machine Code Level to Mark Firmware or Hardware Events
  - E.g. WindView, Linux Trace Toolkit, CodeTest
  - Typically Used to Trace Firmware/Software Events
    - Scheduler Context Switches
    - RTOS Mechanism Events
      - Message Enqueue/Dequeue
      - Interrupts and ISR Execution
      - Semaphore Gives and Takes
      - Exceptions
  - Requires Extensive Target Trace Buffer (Mega Bytes)
  - Event Encoding Must Be Efficient
    - Single Cycle Write of Word or Cache-Line to Memory
    - Single Instruction Sampler Insertion (Minimize Increase in Code Size)
  - Dump Trace Buffer and Post Process with Analysis Tool

Statistical Profile SWIC/HWIC

- E.g. Intel PMU for Pentium and XScale
- Built-In Event Counters and/or Event Trace Buffer
- Lower Overhead than SWIC
- Not Fully HWIC, Typically Requires Interrupt Servicing and Register Configuration
- Event Counter Detectors Are Programmable
  - I-Cache Misses
  - D-Cache Misses
  - Pipeline Stalls
  - Internal Bus Arbitration Cycles
- Counters are Registers
- Detectors are Built-In Logic for Each Event Source
- Counts Can Be Traced
  - Cycle-Based Count Trace – Every N Core Cycles
  - Event-Based Count Trace – E.g. Every N Cache Misses

Code Coverage Verification

- Some Verification Programs Require Measurable Code Test Metrics
  - Path Coverage
  - Statement Coverage
  - MCDC (Multiple Condition Decision Coverage)
    - C Code Short Circuit Branch Logic
      - If(expression_A() && expression_B())
      - When expression_A() is FALSE, expression_B() NOT Executed!
  - Hard Real-Time High Reliability Systems May Require All of the Above for Firmware/Software Verification
- Can Be Supported by HWIC
  - Trace-Port or Tracing of Execution out of External Memory with Cache Disabled
  - Requires Trace Acquisition with Deep Buffering
- Can Be Supported by SWIC
  - Trace Buffer and Encoded Tokens Trace all Branch Events
  - More Intrusive, but Often Sufficient for Functional Tests

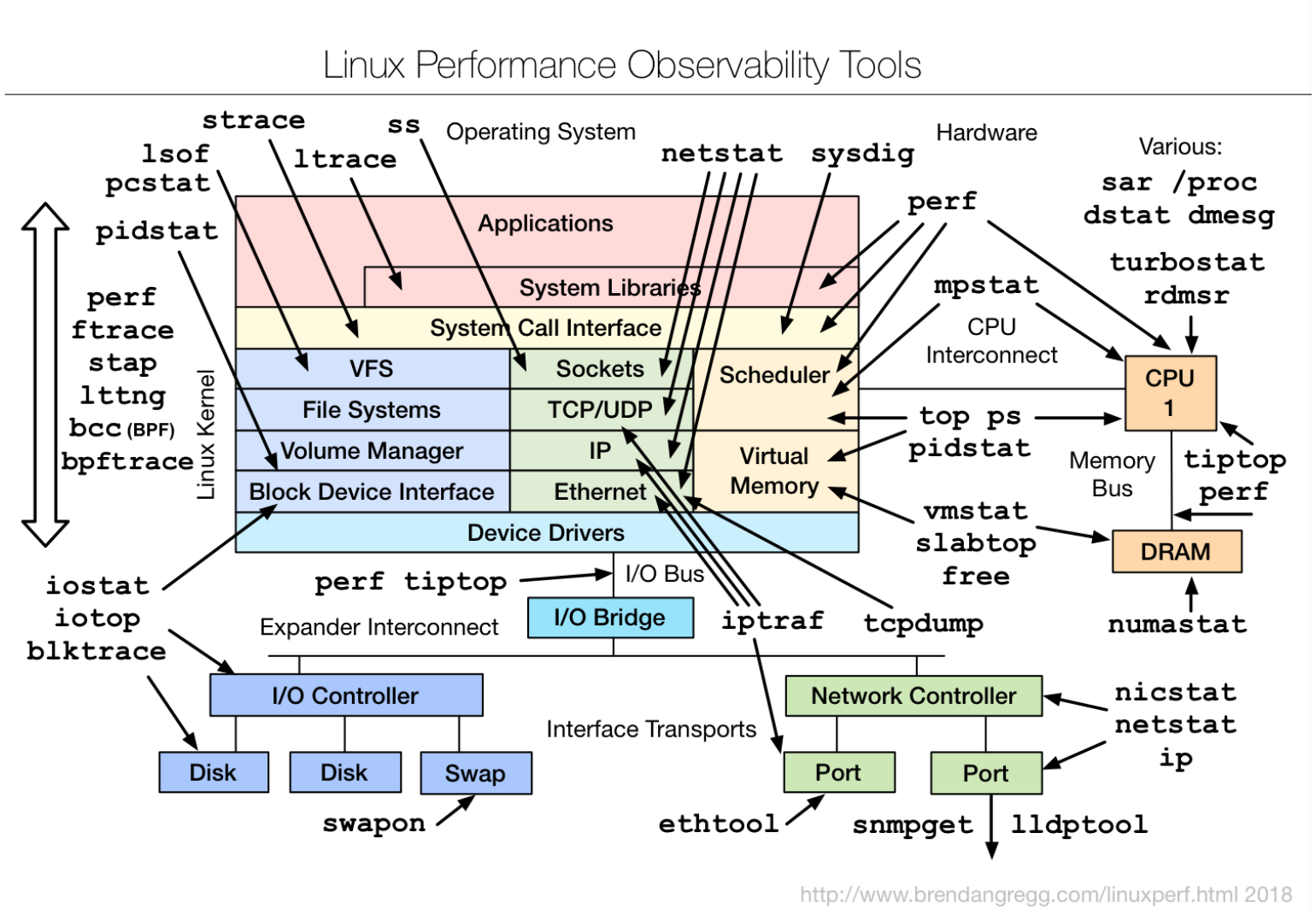
Analysis

- Full Trace
  - E.g. Agilent SCC
  - Source Code Correlation
    - Step Through Code as Executed After the Fact
    - Shows Code Execution Thread as Executed at Full Speed
  - Data Reference Trace
    - Shows Addresses Read/Written at Full Speed
    - Cache Misses Can Show Up on External Memory Trace
    - In-Cache or Register Accesses Only Known if Provided on a Trace-Port
  - Event Trace
    - Provides RTOS or Application Event Trace with Minimal Intrusion
    - Effectively Full Speed if Intrusion is Insignificant (< 1% Resources)
- Statistical
  - E.g. Intel VTune Tool
  - Hot-Spots Noted by Address/Symbol
    - E.g. Functions with Most D-Cache Misses
    - Functions Where Most Cycle Time is Spent
  - Must Run Steady State Code for Long Periods for Validity

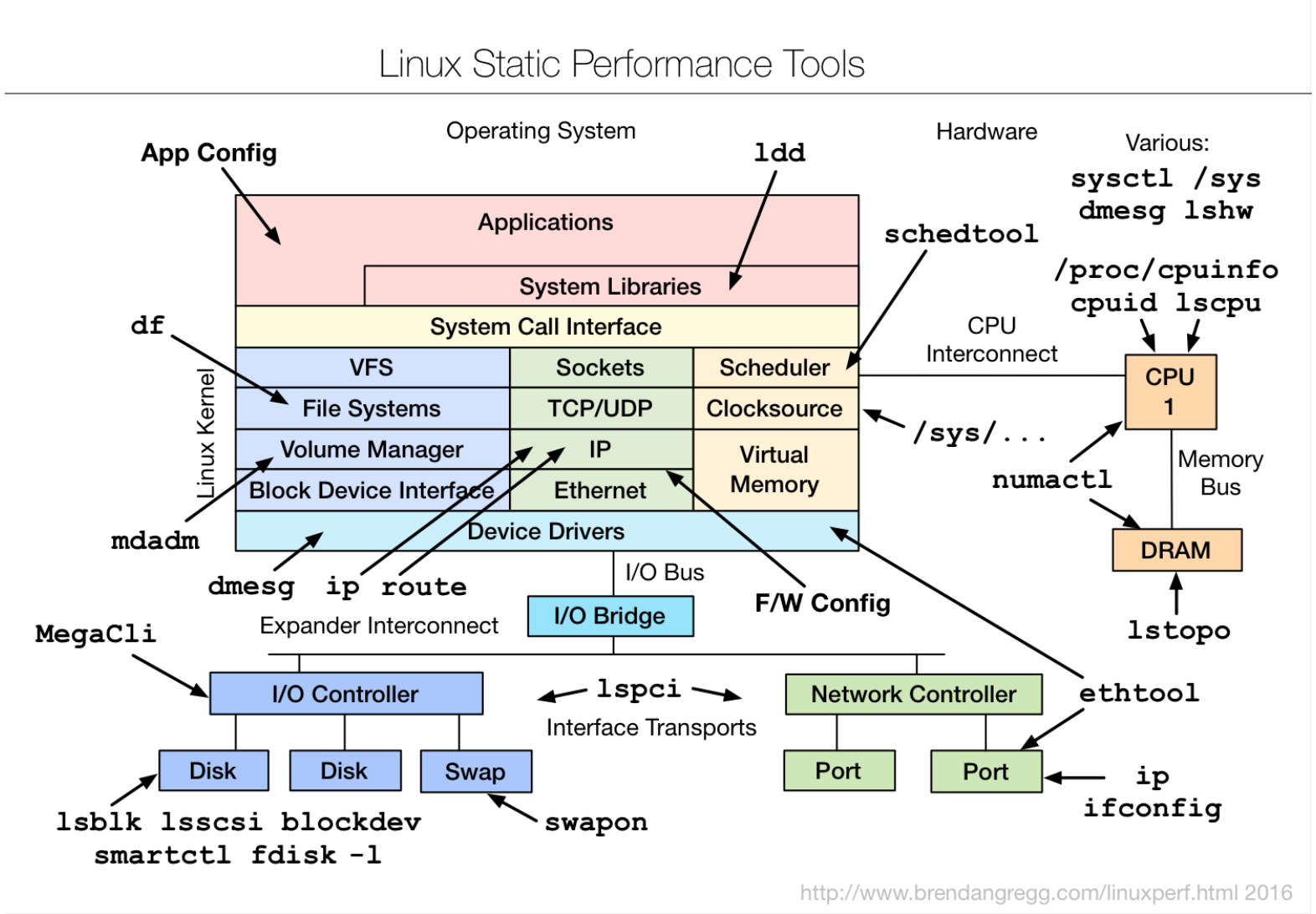
SWIC Tracing and Polling

Linux Methods

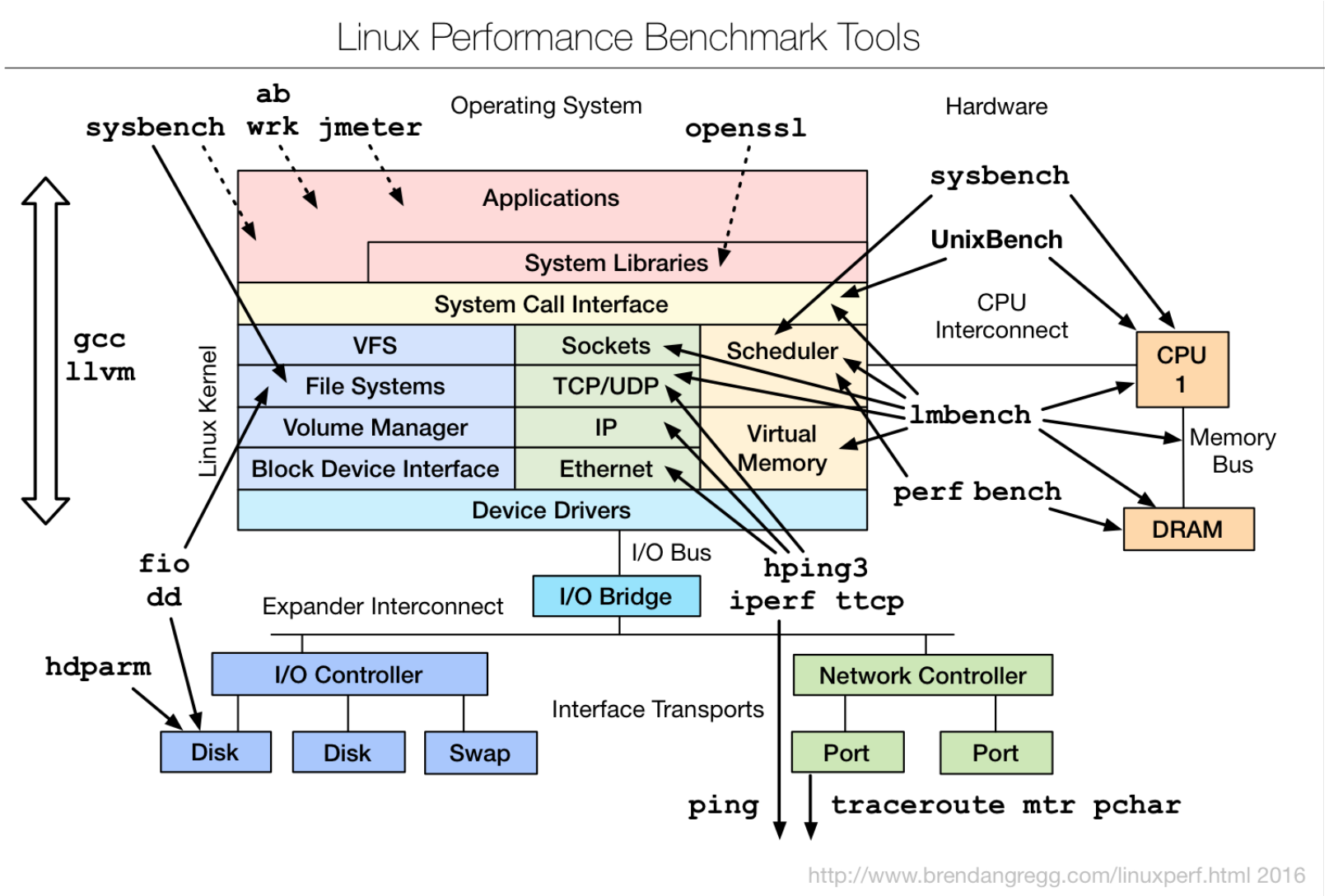
Observability Tools



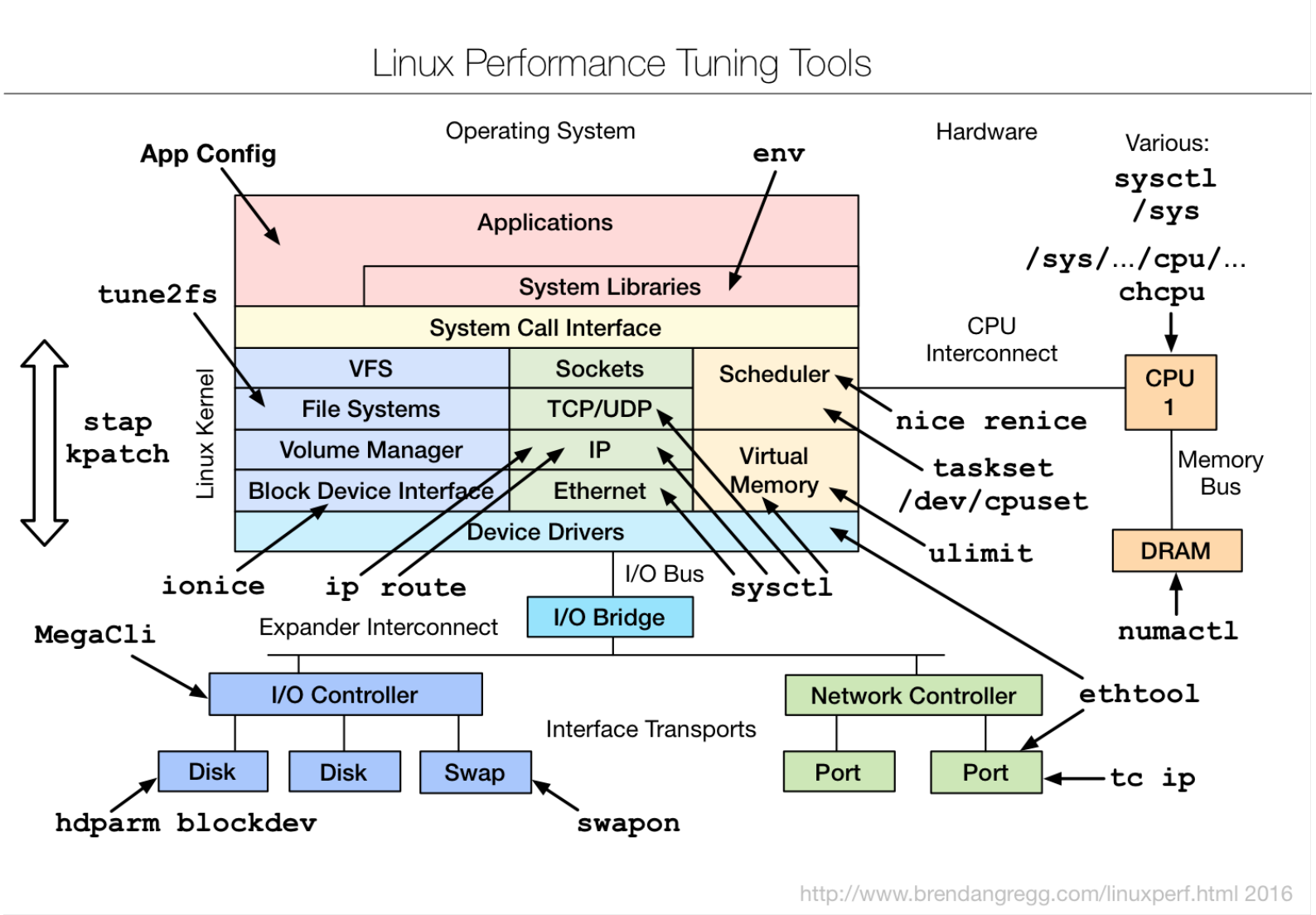
Static Tools



Benchmarking Tools

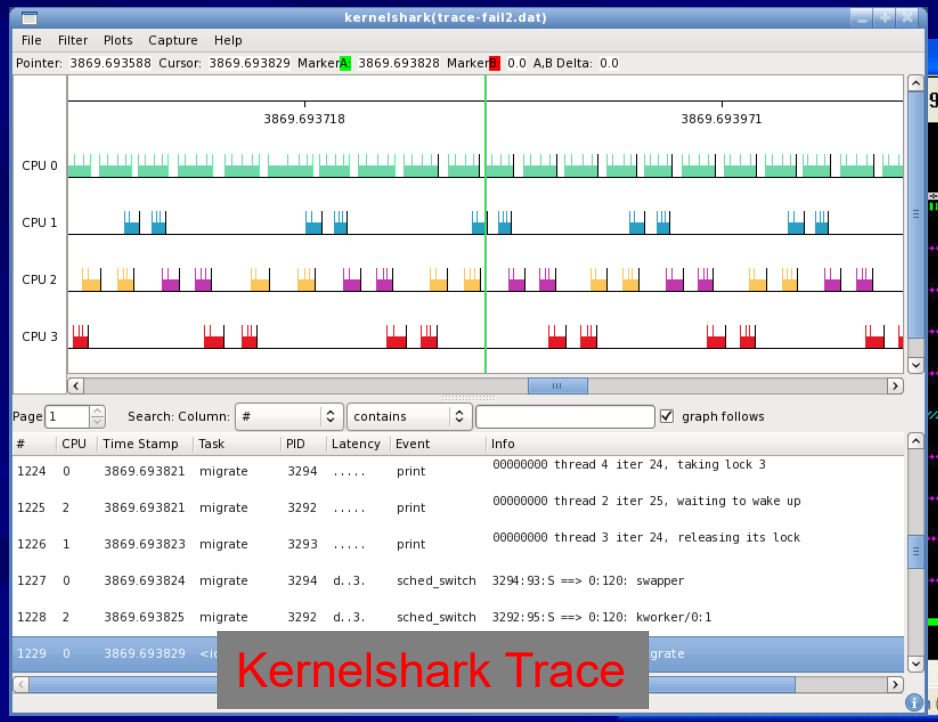


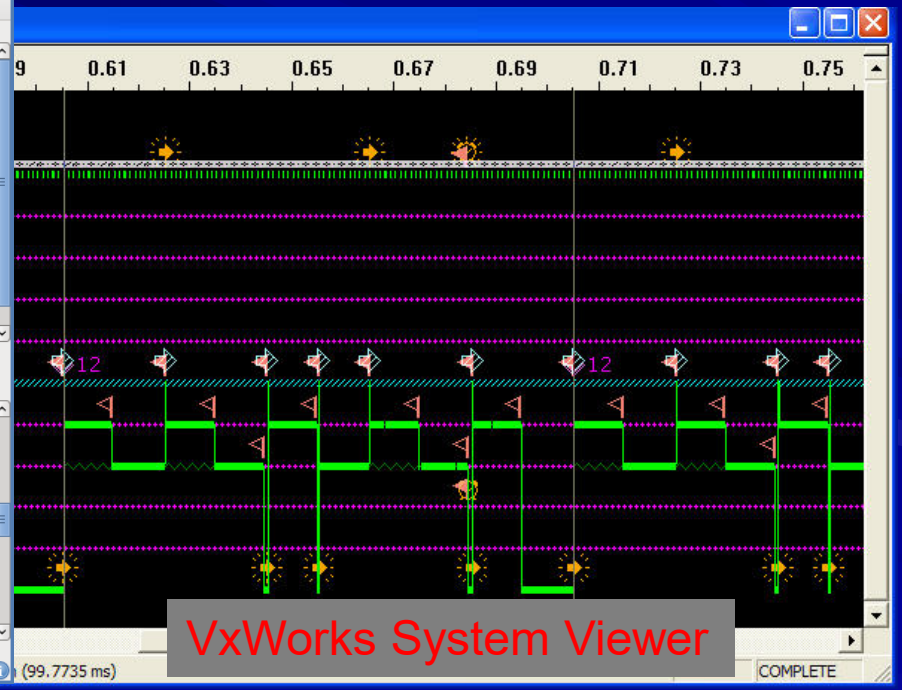
Performance Tuning Tools



Tracing Tools for Linux

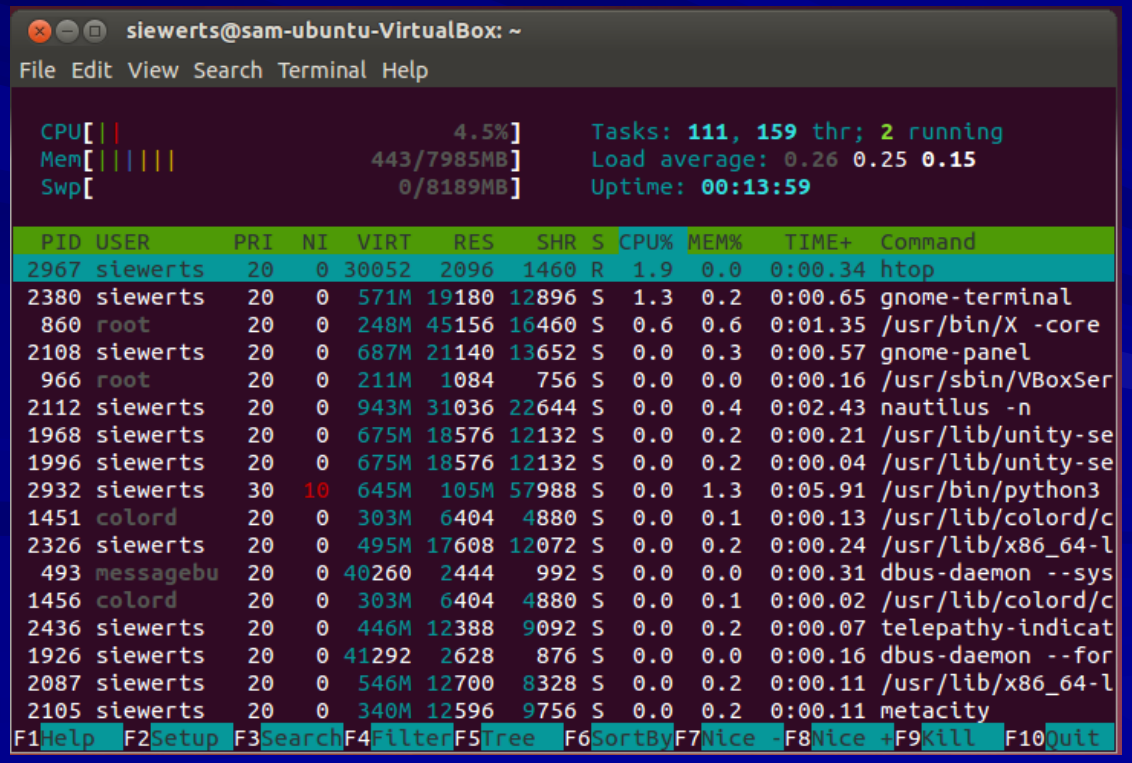
- LTTng – http://lttng.org/
- Systemtap – https://sourceware.org/systemtap/
- Ftrace – http://elinux.org/Ftrace
- Kernelshark – http://rostedt.homelinux.com/kernelshark/, https://lwn.net/Articles/425777/
- Wireshark - https://www.wireshark.org/ (network)
- Syslog - http://linux.die.net/man/5/syslog.conf, http://linux.die.net/man/2/syslog





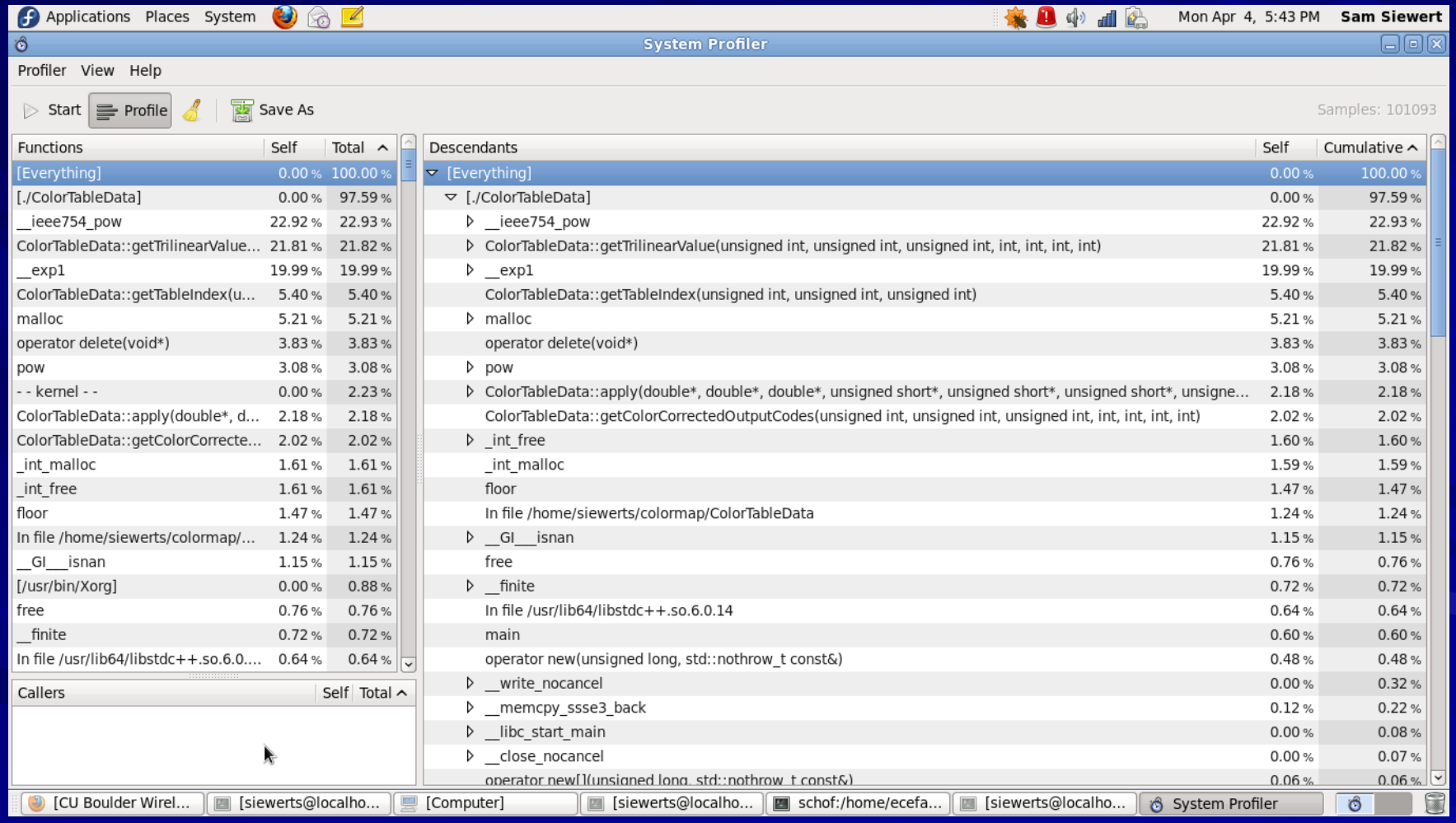
Performance Tests

- Profiling
  - Gprof – Open souce tool [similar to Gcov, but for Profiling]
  - Vtune – Commercial Tool from Intel
  - Logic Analyzer, Agilent SPA (Statistical Performance Analysis)
- Tracing – E.g. Timestamps output to syslog
- Statistics
  - top, htop
  - iostat
  - memstat
- Workloads
  - Iometer
  - stress



Performance - Sysprof

- What is Using CPU on my System
- Rather than Profile of an Application – Sub-System [Service]



Gprof

- Simple –pg compile option
- Run, gprof on gmon.out to get analysis

```
%make
cc -O3 -Wall -pg -msse3 -malign-double -g   -c raidtest.c
raidtest.c: In function 'main':
raidtest.c:99: warning: format '%d' expects type 'int', but argument 2 has type 'long
unsigned int'
raidtest.c:68: warning: unused variable 'aveRate'
raidtest.c:68: warning: unused variable 'totalRate'
raidtest.c:66: warning: unused variable 'rc'
raidtest.c:212: warning: control reaches end of non-void function
cc -O3 -Wall -pg -msse3 -malign-double -g   -c raidlib.c
cc  -O3 -Wall -pg -msse3 -malign-double -g   -o raidtest raidtest.o raidlib.o

%./raidtest
Will default to 1000 iterations
Architecture validation:
sizeof(unsigned long long)=8


RAID Operations Performance Test
Test Done in 453 microsecs for 1000 iterations
2207505.518764 RAID ops computed per second

%ls
Makefile    gmon.out   raidlib.h  raidlib64.c  raidtest    raidtest.o
Makefile64  raidlib.c  raidlib.o  raidlib64.h  raidtest.c  raidtest64
%gprof raidtest gmon.out > raidtest_analysis.txt
```

Gprof Analysis

- 1 million iterations of RAID test XOR and Rebuild

```
RAID Operations Performance Test
Test Done in 206417 microsecs for 1000000 iterations
4844562.221135 RAID ops computed per second
```

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 82.13      1.54     1.54                             main
 15.47      1.83     0.29  2000001   145.38   145.38  xorLBA
  2.67      1.88     0.05  2000001    25.07    25.07  rebuildLBA

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  …

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, …

 total     the average number of milliseconds spent in this
ms/call    function and its descendants per call, …

name       the name of the function.  …
```

Call Graph Profile from Gprof

```
 Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.53% of 1.88 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    1.54    0.34                 main [1]
                0.29    0.00 2000001/2000001     xorLBA [2]
                0.05    0.00 2000001/2000001     rebuildLBA [3]
-----------------------------------------------
                0.29    0.00 2000001/2000001     main [1]
[2]     15.4    0.29    0.00 2000001         xorLBA [2]
-----------------------------------------------
                0.05    0.00 2000001/2000001     main [1]
[3]      2.7    0.05    0.00 2000001         rebuildLBA [3]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children…
     % time     This is the percentage of the `total' time that was spent
                in this function and its children…
     self       This is the total amount of time spent in this function.

     children   This is the total amount of time propagated into this
                function by its children.

     called     This is the number of times the function was called…
```