Low Level Tracing for Latency Analysis

From Baremetal to Hardware Tracing Blocks

Suchakrapani Datt Sharma & Thomas Bertauld

Oct 12, 2016

École Polytechnique de Montréal

Laboratoire DORSAL

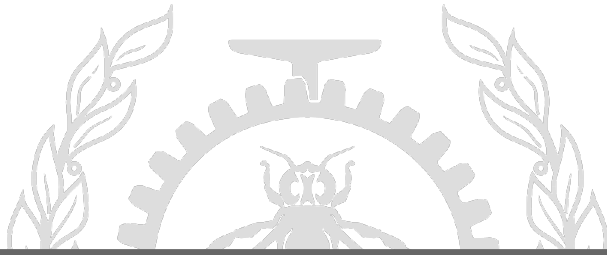
whoami

Suchakra

- PhD student, Computer Engineering

  (Prof Michel Dagenais)

  DORSAL Lab, École Polytechnique de Montréal - UdeM
- Works on debugging, tracing and trace aggregation (LTTng, eBPF), hardware tracing and VMs.
- Loves poutine, samosas and bikes



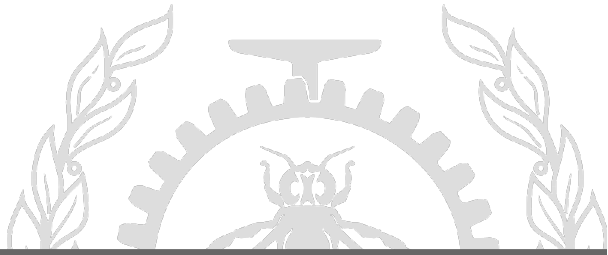
whoami

Thomas

- MSc student, Computer Engineering

  (Prof Michel Dagenais)

  DORSAL Lab, École Polytechnique de Montréal - UdeM
- Worked on embedded systems tracing, baremetal systems, trace analysis and now in financial-tech domain
- Loves computer games



Agenda

Latency

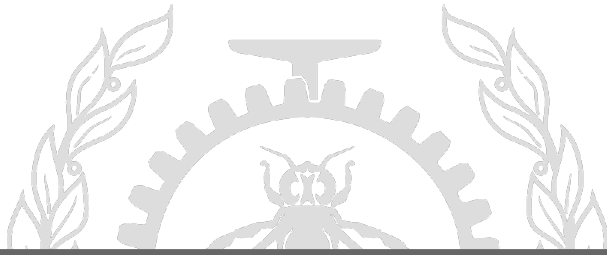
- Introduction
- Tools and techniques

Hardware Tracing

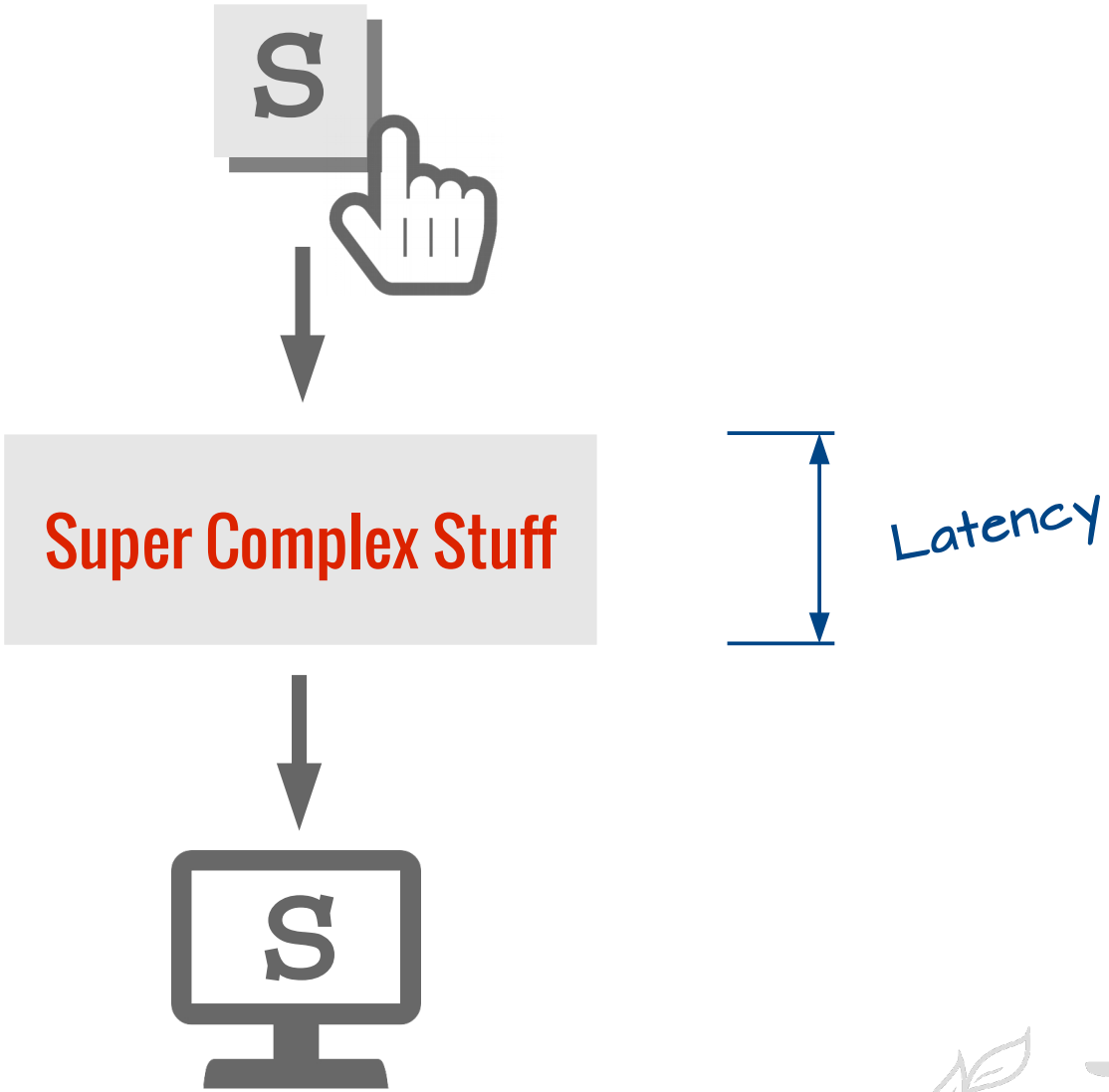
- Intel Processor Trace
- ARM CoreSight
- Hardware trace based analysis

Baremetal Tracing

- Heterogeneous system challenges
- Low level traces with **`barectf`**

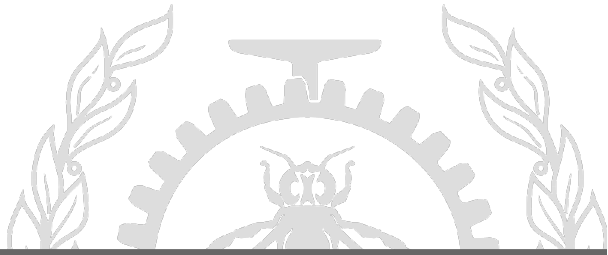


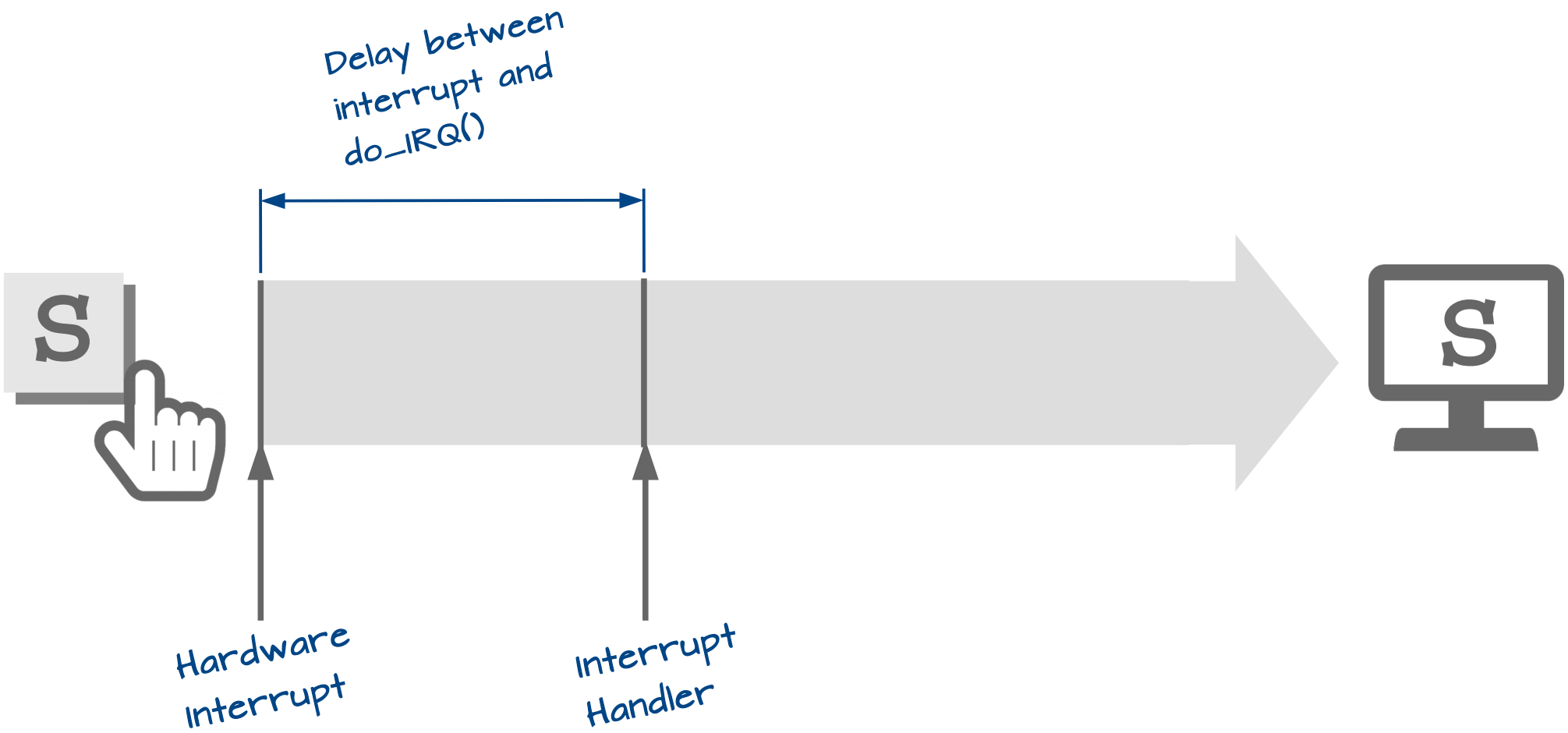
Latency

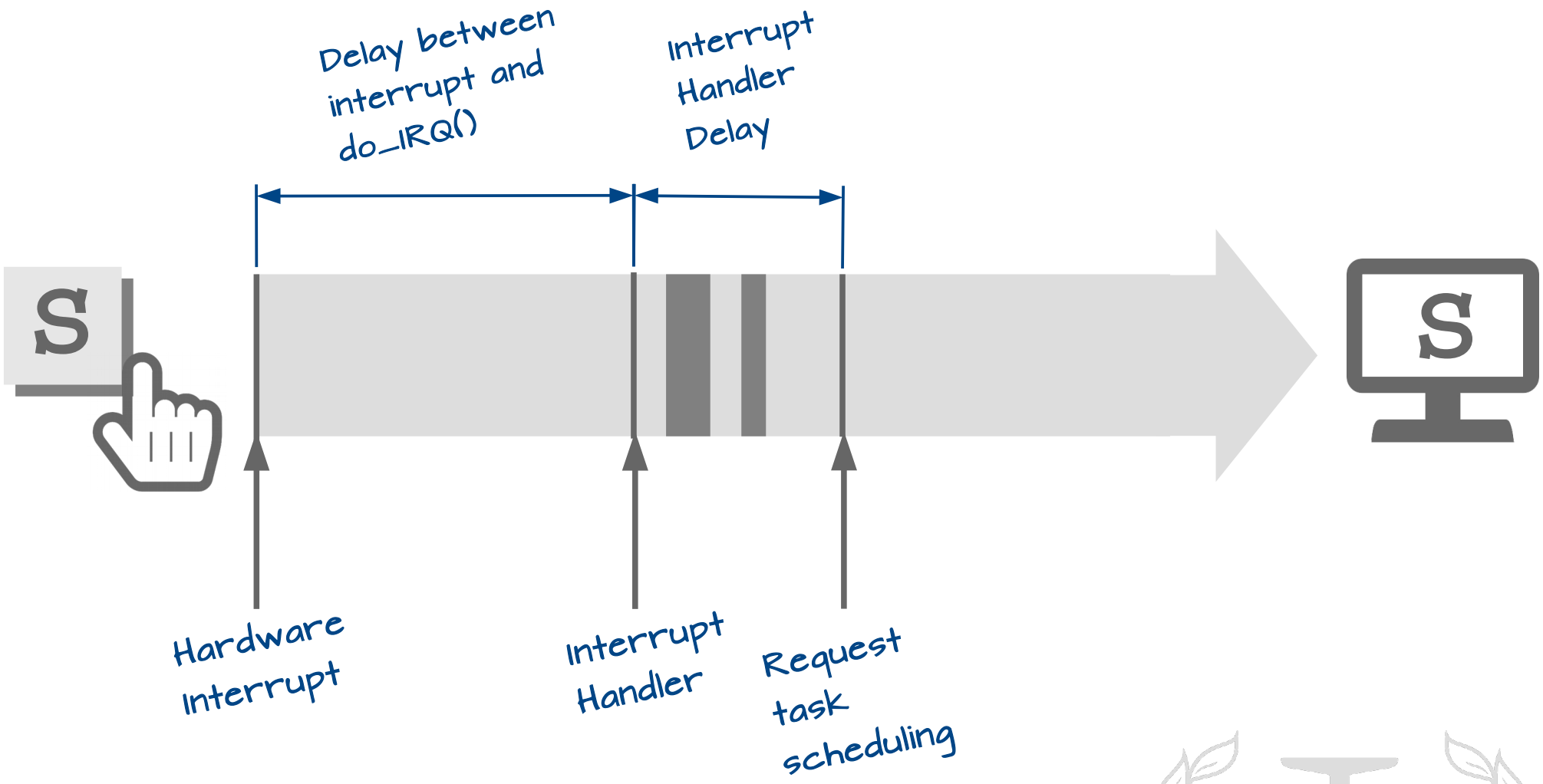

S
Super Complex Stuff
Latency
S

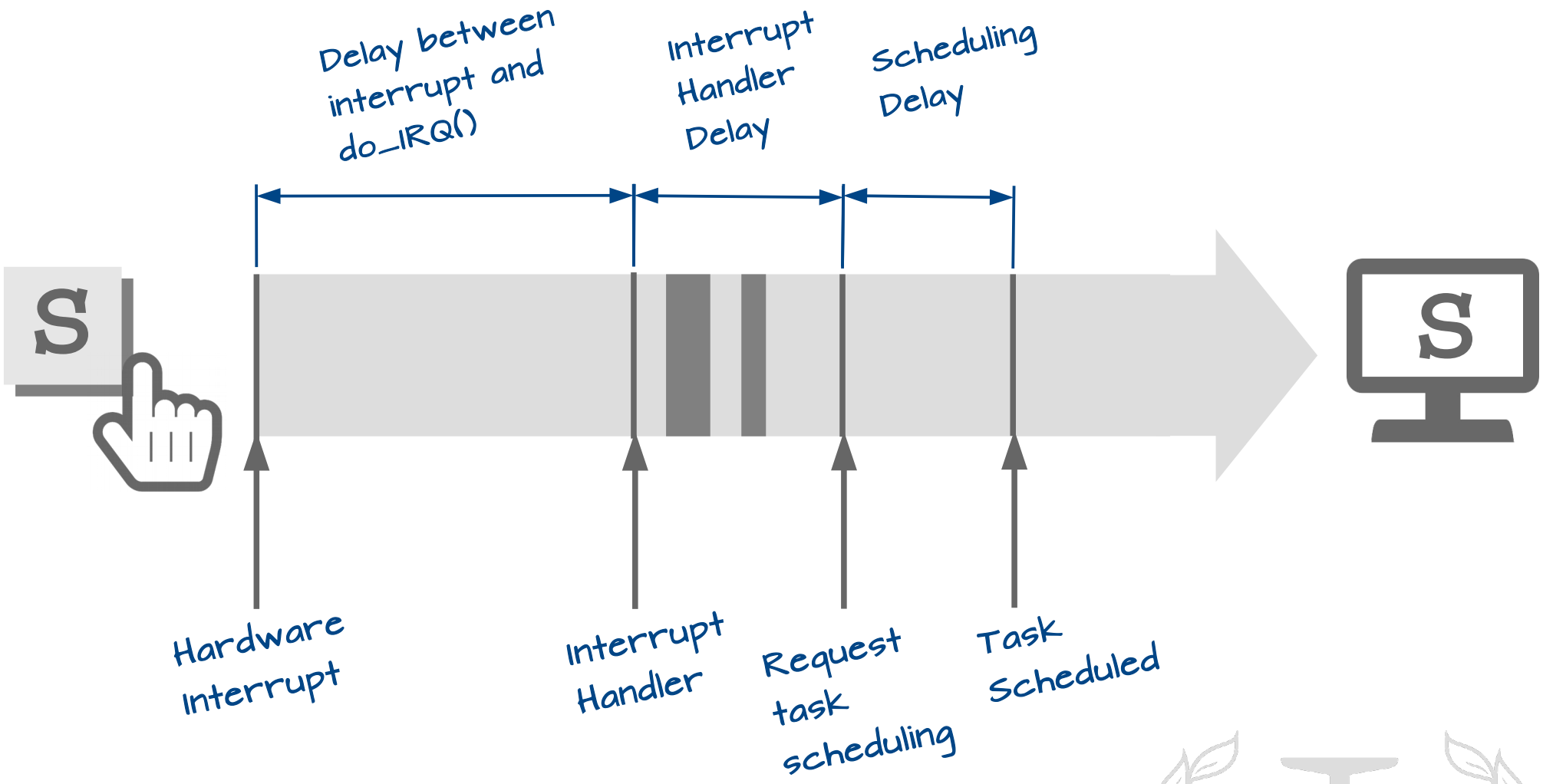

S
S

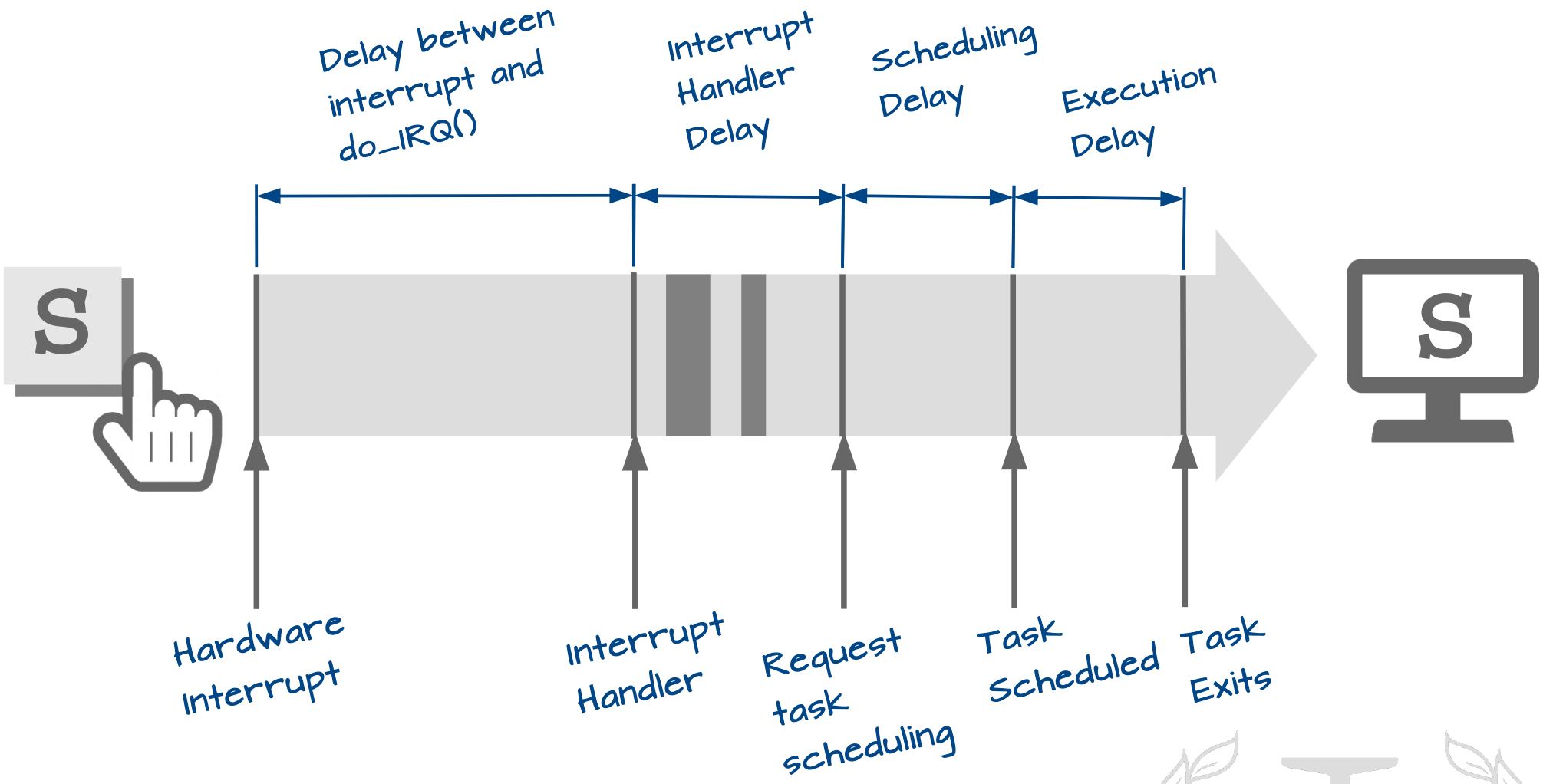

S
S
Hardware
Interrupt




Delay between interrupt and do_IRQ()
S
S
Hardware Interrupt
Interrupt Handler


Delay between interrupt and do_IRQ()
Interrupt Handler Delay
S
S
Hardware Interrupt
Interrupt Handler
Request task scheduling

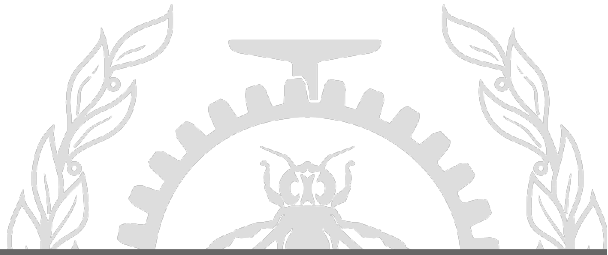

Delay between interrupt and do_IRQ()
Interrupt Handler Delay
Scheduling Delay
S
S
Hardware Interrupt
Interrupt Handler
Request task scheduling
Task Scheduled


Delay between interrupt and do_IRQ()
Interrupt Handler Delay
Scheduling Delay
Execution Delay
S
S
Hardware Interrupt
Interrupt Handler
Request task scheduling
Task Scheduled
Task Exits

Latency

Challenges in Measurement

- May be non-deterministic, sporadic and may required long term continuous monitoring
- Measuring techniques may add latency to critical path
- Modern SoCs - multiple processors, multiple architectures
- Sampling techniques have limited viability



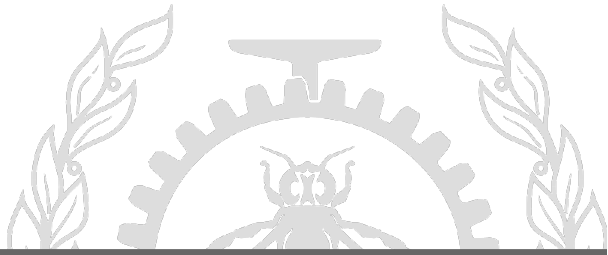
Ftrace

Ftrace

Interesting Events

```
do_IRQ()
events/irq/irq_handler_entry/enable
events/sched/sched_waking/enable
events/sched/sched_wakeup/enable
events/sched/sched_switch/enable
sys_exit
```

```
do_IRQ
irq_handler_entry
sched_waking
sched_wakeup
sched_switch
sys_exit
```

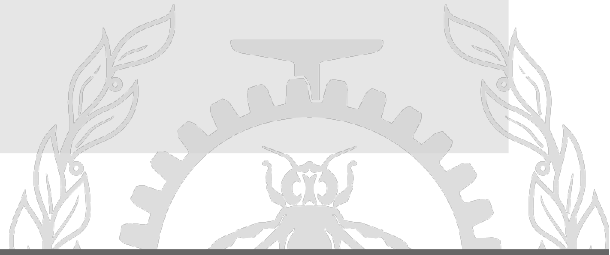


Ftrace

`irqsoff/preemptirqsoff` Tracer

```
# irqsoff latency trace v1.1.5 on 4.5.0
# --------------------------------------------------------------------
# latency: 336 us, #4/4, CPU#1 | (M:desktop VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: gnome-shell-4858 (uid:1000 nice:0 policy:0 rt_prio:0)
#    -----------------
```

```
gnome-sh-4858    1d...   336us : gen8_emit_request <-__i915_add_request
gnome-sh-4858    1d...   337us : trace_hardirqs_on <-__i915_add_request
gnome-sh-4858    1d...   341us : <stack trace>
 => gen8_emit_request
 => __i915_add_request
 => i915_gem_execbuffer_retire_commands
 => intel_execlists_submission
 => i915_gem_do_execbuffer.isra.18
 => i915_gem_execbuffer2
 => drm_ioctl
 => vfs_ioctl
 => do_vfs_ioctl
 => SyS_ioctl
 => entry_SYSCALL_64_fastpath
```

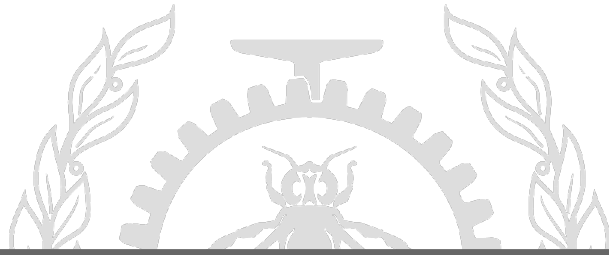


Ftrace

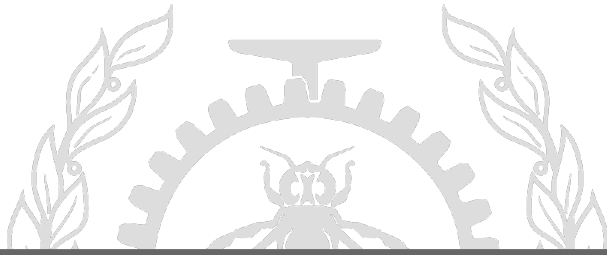
wakeup/wakeup-rt Tracer

```
# wakeup latency trace v1.1.5 on 4.5.0
# --------------------------------------------------------------------
# latency: 10 us, #4/4, CPU#1 | (M:desktop VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: kworker/1:1H-2312 (uid:0 nice:-20 policy:0 rt_prio:0)
#    -----------------
```

```
  <idle>-0  1dNs.    1us :       0:120:R   + [001]  2312:100:R kworker/1:1H
  <idle>-0  1dNs.    2us : ttwu_do_activate.constprop.70 <-try_to_wake_up
  <idle>-0  1d...   10us : __schedule <-schedule
  <idle>-0  1d...   10us :       0:120:R ==> [001]  2312:100:R kworker/1:1H
```



Other Tools

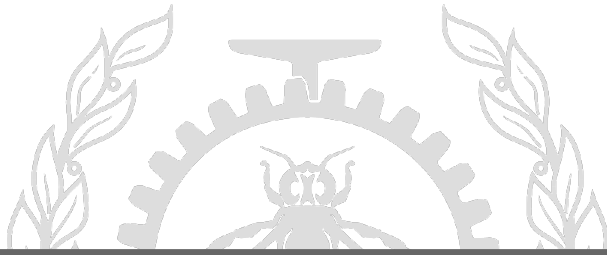


Other Tools

Latency Tracker

- Many Analyses, single module
  - Wakeup Latency
  - Off-CPU latency
  - Syscall Latency
  - I/O Request Latency
- Views in Ftrace or LTTng

eBPF, LatencyTop



Other Tools

Perf `sched`

```
# perf sched record sleep 0.01
# perf sched latency
```

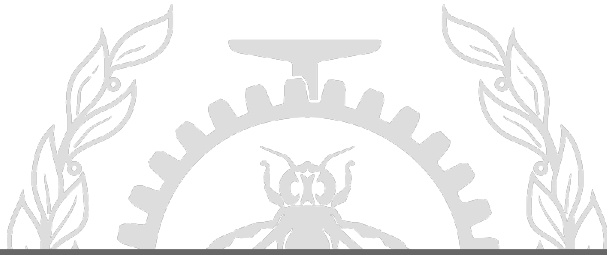
```
 -----------------------------------------------------------------------------------------------------------------
  Task                  |   Runtime ms  | Switches | Average delay ms | Maximum delay ms | Maximum delay at       |
 -----------------------------------------------------------------------------------------------------------------
  kworker/u8:2:10930    |      0.161 ms |        1 | avg:    0.140 ms | max:    0.140 ms | max at: 136693.341916 s
  kworker/2:1:11382     |      0.058 ms |        3 | avg:    0.043 ms | max:    0.075 ms | max at: 136693.376069 s
  rcu_sched:7           |      0.395 ms |        8 | avg:    0.041 ms | max:    0.155 ms | max at: 136693.283139 s
  rcuos/2:25            |      0.183 ms |        4 | avg:    0.039 ms | max:    0.146 ms | max at: 136693.295139 s
  kworker/3:0:11129     |      0.052 ms |        2 | avg:    0.023 ms | max:    0.028 ms | max at: 136693.280035 s
  kworker/0:1:11383     |      0.061 ms |        3 | avg:    0.020 ms | max:    0.029 ms | max at: 136693.283007 s
  soffice.bin:29258     |      0.132 ms |        1 | avg:    0.012 ms | max:    0.012 ms | max at: 136693.298755 s
  nautilus:6071         |      0.048 ms |        1 | avg:    0.012 ms | max:    0.012 ms | max at: 136693.310336 s
  chrome:(3)            |      0.508 ms |        4 | avg:    0.011 ms | max:    0.012 ms | max at: 136693.316332 s
  SCTP timer:3265       |      0.324 ms |       11 | avg:    0.011 ms | max:    0.013 ms | max at: 136693.374016 s
  gmain:(2)             |      0.186 ms |        2 | avg:    0.010 ms | max:    0.011 ms | max at: 136693.278787 s
  sleep:11404           |      1.012 ms |        2 | avg:    0.009 ms | max:    0.012 ms | max at: 136693.375922 s
  kworker/u8:3:11157    |      0.012 ms |        1 | avg:    0.008 ms | max:    0.008 ms | max at: 136693.341930 s
  perf:11403            |      0.656 ms |        1 | avg:    0.007 ms | max:    0.007 ms | max at: 136693.376070 s
  rcuos/0:9             |      0.034 ms |        2 | avg:    0.006 ms | max:    0.009 ms | max at: 136693.277007 s
  irq/28-iwlwifi:695    |      0.000 ms |       21 | avg:    0.005 ms | max:    0.007 ms | max at: 136693.348238 s
  ips-monitor:671       |      0.020 ms |        1 | avg:    0.004 ms | max:    0.004 ms | max at: 136693.375995 s
  rcuos/3:32            |      0.015 ms |        2 | avg:    0.004 ms | max:    0.004 ms | max at: 136693.283155 s
  migration/3:28        |      0.000 ms |        1 | avg:    0.002 ms | max:    0.002 ms | max at: 136693.275081 s
 -----------------------------------------------------------------------------------------------------------------
  TOTAL:                |      3.857 ms |       71 |
 ---------------------------------------------------
```

Other Tools

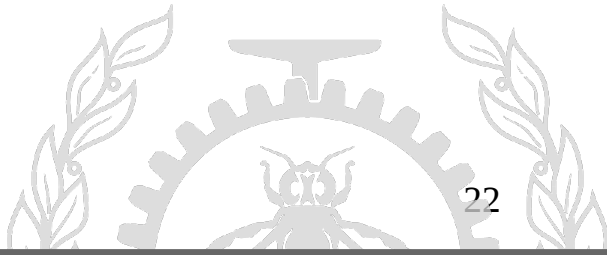
Latency Tracker

- Many Analyses, single module
  - Wakeup Latency
  - Off-CPU latency
  - Syscall Latency
  - I/O Request Latency
- Views in Ftrace or LTTng

eBPF, LatencyTop

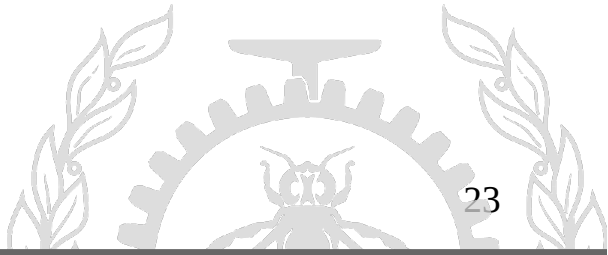


- What happens **before** kernel receives an interrupt?
- Does a tracer affect the test system itself?
- What about regions in kernel that you can't trace?
- Can we observe the system without being in the critical path?



- What happens **before** kernel receives an interrupt?
- Does a tracer affect the test system itself?
- What about regions in kernel that you can't trace?
- Can we observe the system without being in the critical path?

Precise instruction and data flow straight from processor!



Hardware Tracing

Hardware Tracing 101

1998

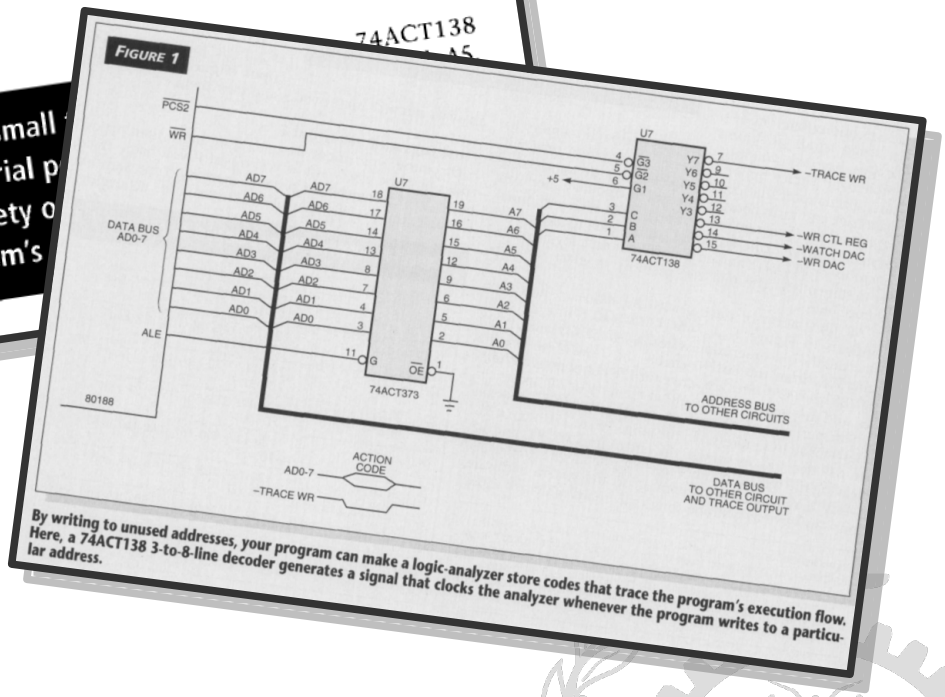
Debugging embedded systems: using hardware tricks to trace program flow

STUART R BALL

A trace buffer is a valuable debugging aid. It provides a history of your program's behavior by recording explanatory "action codes" whenever your program reaches certain key points in its execution (**Reference 1**). In some small µC-based

If your embedded system is too small
provide even a trace buffer or a serial p
debugging aids, you can use a variety o
ware tricks to observe your program's

74ACT138



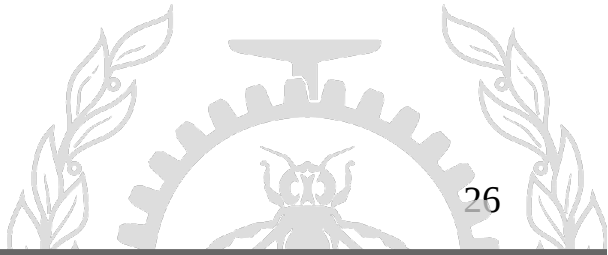
By writing to unused addresses, your program can make a logic-analyzer store codes that trace the program's execution flow. Here, a 74ACT138 3-to-8-line decoder generates a signal that clocks the analyzer whenever the program writes to a particular address.

Courtesy, EDN; Apr 23, 1998; 43, 9; ProQuest pg. 163

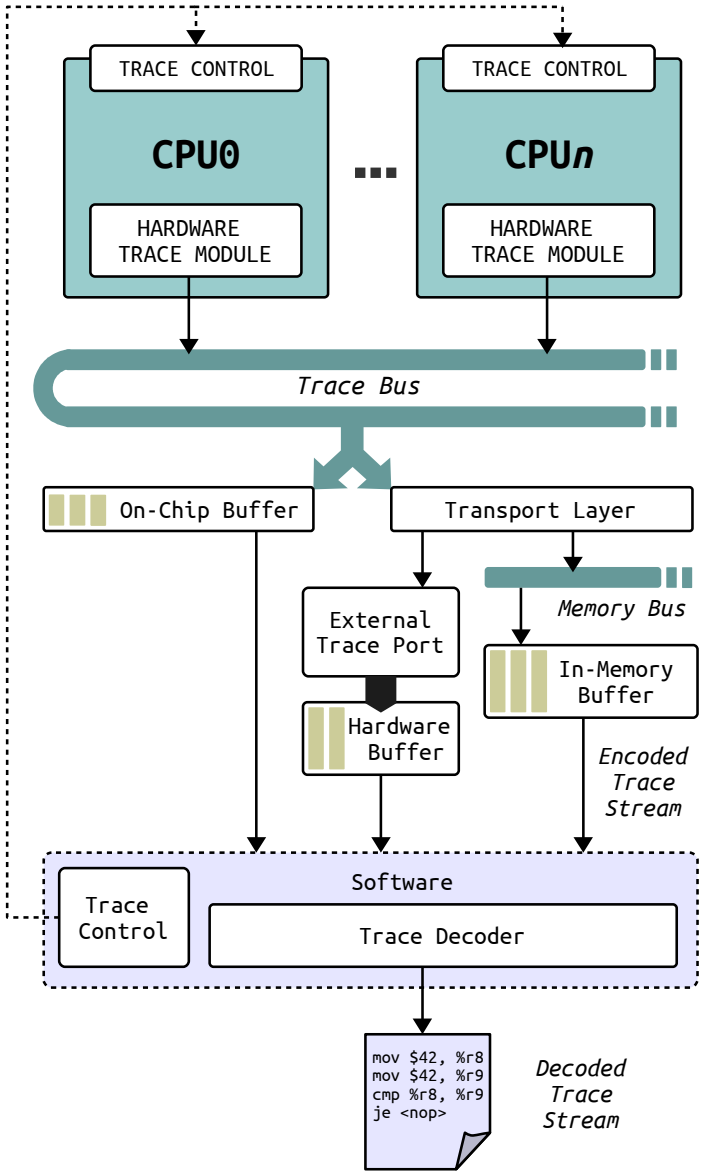
Hardware Tracing 101

Mainstream Adoption

- ARM very early introduced EmbeddedICE
  - Direct access to the data and address bus of the CPU
  - External device controls through TAP (JTAG interfaced)
- ETM/PTM → CoreSight
- Intel starts with LBR support and moves on to BTS [8, 2]
- MIPS PDtrace



Hardware Tracing 101



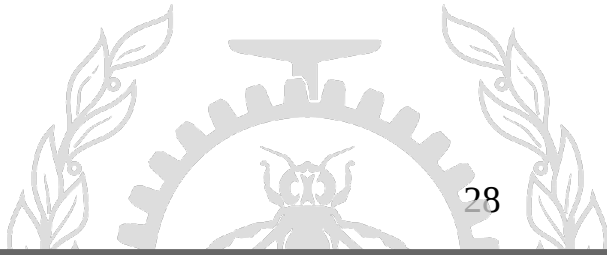
Typical Arrangement

- Trace packets flow from 'processor' to on-chip buffer or external transport
- Traces in range of hundreds of Mbits/s to Gbits/s.
- Being quickly adopted in Linux (Perf/Coresight)
- Decoding performed 'offline' after trace recording
- Can be instruction or data flow

Hardware Tracing 101

Architectures

- ARM CoreSight (Program and Data Flow Trace) [1]
  - Stream trace to external transport or internal buffer
- Intel PT (Program Flow) [2, 3]
- MIPS PDTrace (Program and Data Flow)

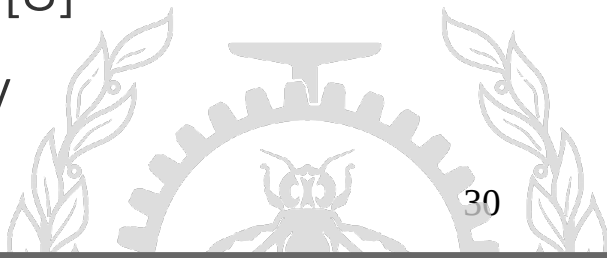


Intel Processor Trace

Background

Intel LBR and BTS

- **LBR** - Last Branch Record
  - Save last *n* number of branches
  - LBR Stack MSRs based - limited for comprehensive analysis
  - BTM messages can be sent on system bus
- **BTS** - Branch Trace Store
  - When enabled, BTM can save data in BTS buffer, generate interrupt when full and save
  - 24 bytes per branch [FROM(64);TO(64);PREDICTED(1/64)]
  - Heavy penalty. Sometimes **40x** overheads! [8]
  - Designed for debugging scenarios primarily



Intel PT

Intel Processor Trace

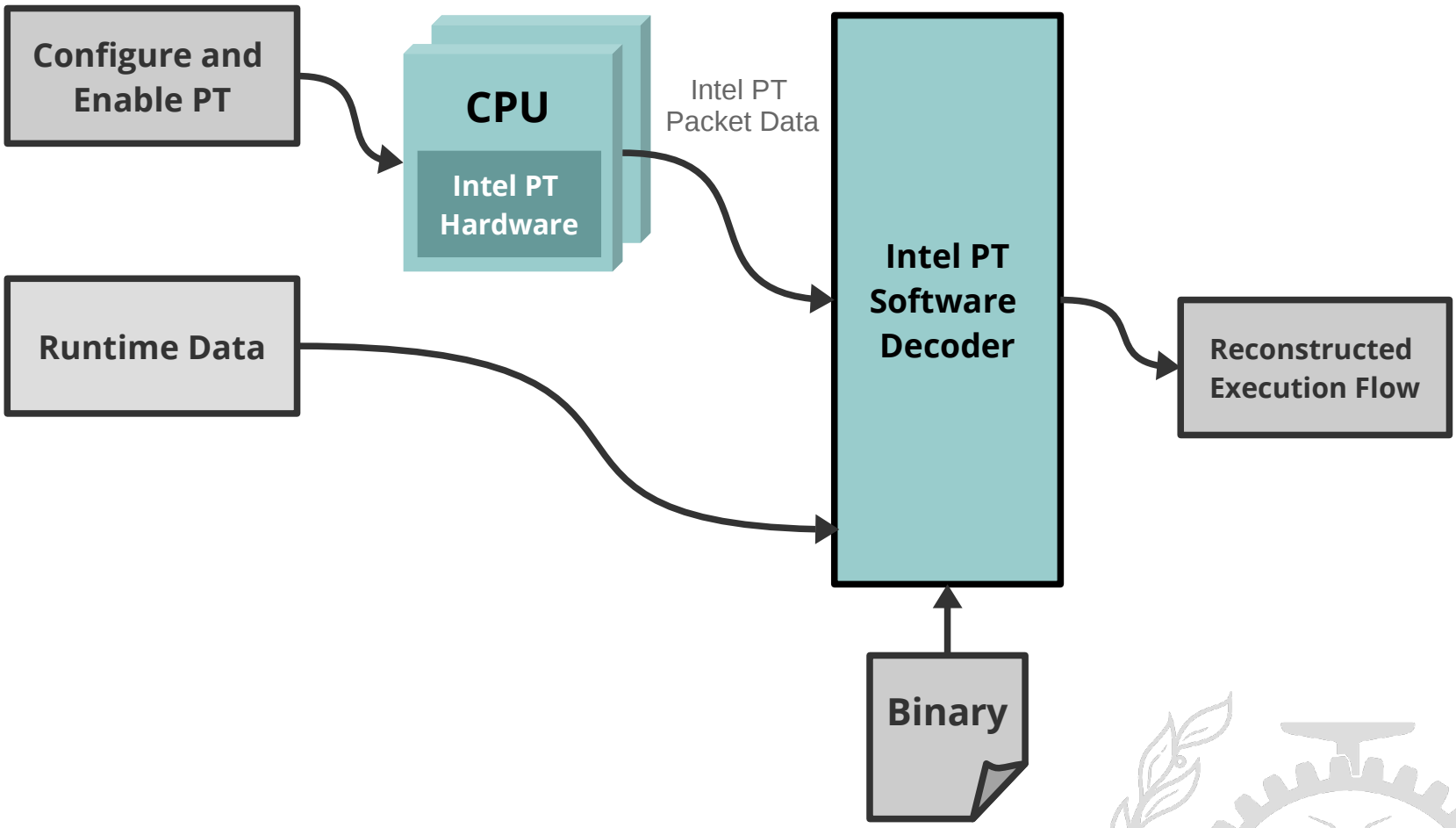
- Control Flow Tracing
  - Record branches to deduce program flow
  - Configure MSRs, setup buffer & generate trace packets
  - Save Packets to buffer or send to 'transport layer'
- Trace Packets Overview
  - **PSB** (Packet Stream Boundary) : Heartbeat, every 4K packets
  - **PIP** (Paging Information Packet) : CR3 change
  - **TSC**, **OVF** (Overflow), **CBR** (Core:Bus) Packets
  - *Control Flow*
    - **TNT** (Taken Not-Taken), except unconditional
    - **TIP** (Target IP) at branches, **FUP**, **MODE**

Intel Processor Trace

- Cycle Accurate Mode
  - **CYC** Packets : Cycle counter data to keep tab on instruction count, IPC, tracking wall-clock time
- Timing
  - **MTC** (Mini Timestamp Counter) : More frequent, based on CTC (crystal clock counter) value (8 bit). Can be frequency adjusted. Used with TSC to get accurate timestamps for less cost. [TSC → TMA → MTC, MTC, MTC → TSC]
  - Decoder finds out accurate time offline through elaborate calculation (Refer Chapter 36, Intel Manual)

Intel PT

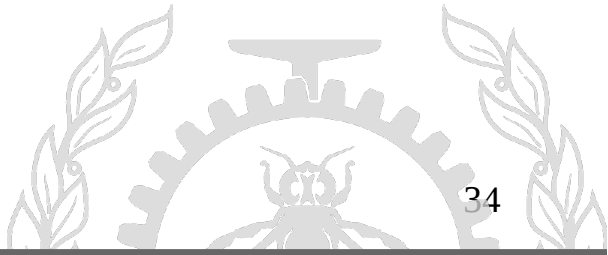
Using Intel PT



Based on, Andi Kleen's Presentation (TracingSummit 2015)

Using Intel PT

- Perf support
  - Perf driver configures and controls PT hardware
  - Generated trace data is dumped to an AUX buffer on top of Perf's buffer
  - Userspace Perf counterpart can decode the perf data
  - Accurate instruction profiling data
- GDB support
  - Accurate reverse debugging.
- Andi Kleen's **simple-pt**



Intel PT

Hardware Trace Packets (Perf)

```
. ... Intel Processor Trace data: size 8544 bytes
.   00000000:  02 82 02 82 02 82 02 82 02 82 02 82 02 82 PSB
.   00000010:  00 00 00 00 00 00                         PAD
.   00000016:  19 ba 39 4d 7b 89 5e 04                   TSC 0x45e897b4d39ba
.   0000001e:  00 00 00 00 00 00 00 00                   PAD
.   00000026:  02 73 57 64 00 1c 00 00                   TMA CTC 0x6457 FC 0x1c
.   0000002e:  00 00                                     PAD
.   00000030:  02 03 27 00                               CBR 0x27
.   00000034:  02 23                                     PSBEND
.   00000036:  59 8b                                     MTC 0x8b
.   00000038:  59 8c                                     MTC 0x8c
.
.   00000304:  f8                                        TNT TTTTNN (6)
.   00000305:  06 00 00                                  TNT T (1)
.   00000308:  4d e0 3c 6d 9c                            TIP 0x9c6d3ce0
.   0000030d:  1c 00 00                                  TNT TTN (3)
.   00000310:  2d f0 3c                                  TIP 0x3cf0
.   00000313:  06                                        TNT T (1)
.   00000314:  59 2e                                     MTC 0x2e
.   00000316:  94                                        TNT NNTNTN (6)
.   00000317:  a8                                        TNT NTNTNN (6)
.   00000318:  a6                                        TNT NTNNTT (6)
```

Intel PT

Timing

```
. ... Intel Processor Trace data: size 8544 bytes
.  00000000:  02 82 02 82 02 82 02 82 02 82 02 82 02 82 02 82 PSB
.  00000010:  00 00 00 00 00 00                               PAD
.  00000016:  19 ba 39 4d 7b 89 5e 04                         TSC 0x45e897b4d39ba
.  0000001e:  00 00 00 00 00 00 00 00                         PAD
.  00000026:  02 73 57 64 00 1c 00 00                         TMA CTC 0x6457 FC 0x1c
.  0000002e:  00 00                                           PAD
.  00000030:  02 03 27 00                                     CBR 0x27
.  00000034:  02 23                                           PSBEND
.  00000036:  59 8b                                           MTC 0x8b
.  00000038:  59 8c                                           MTC 0x8c
.
.  00000304:  f8                                              TNT TTTTNN (6)
.  00000305:  06 00 00                                        TNT T (1)
.  00000308:  4d e0 3c 6d 9c                                  TIP 0x9c6d3ce0
.  0000030d:  1c 00 00                                        TNT TTN (3)
.  00000310:  2d f0 3c                                        TIP 0x3cf0
.  00000313:  06                                              TNT T (1)
.  00000314:  59 2e                                           MTC 0x2e
.  00000316:  94                                              TNT NNTNTN (6)
.  00000317:  a8                                              TNT NTNTNN (6)
.  00000318:  a6                                              TNT NTNNTT (6)
```

Intel PT

Conditional Branches

```
. ... Intel Processor Trace data: size 8544 bytes
.  00000000:  02 82 02 82 02 82 02 82 02 82 02 82 02 82 PSB
.  00000010:  00 00 00 00 00 00                         PAD
.  00000016:  19 ba 39 4d 7b 89 5e 04                   TSC 0x45e897b4d39ba
.  0000001e:  00 00 00 00 00 00 00 00                   PAD
.  00000026:  02 73 57 64 00 1c 00 00                   TMA CTC 0x6457 FC 0x1c
.  0000002e:  00 00                                     PAD
.  00000030:  02 03 27 00                               CBR 0x27
.  00000034:  02 23                                     PSBEND
.  00000036:  59 8b                                     MTC 0x8b
.  00000038:  59 8c                                     MTC 0x8c
.
.  00000304:  f8                                        TNT TTTTNN (6)
.  00000305:  06 00 00                                  TNT T (1)
.  00000308:  4d e0 3c 6d 9c                            TIP 0x9c6d3ce0
.  0000030d:  1c 00 00                                  TNT TTN (3)
.  00000310:  2d f0 3c                                  TIP 0x3cf0
.  00000313:  06                                        TNT T (1)
.  00000314:  59 2e                                     MTC 0x2e
.  00000316:  94                                        TNT NNTNTN (6)
.  00000317:  a8                                        TNT NTNTNN (6)
.  00000318:  a6                                        TNT NTNNTT (6)
```

Intel PT

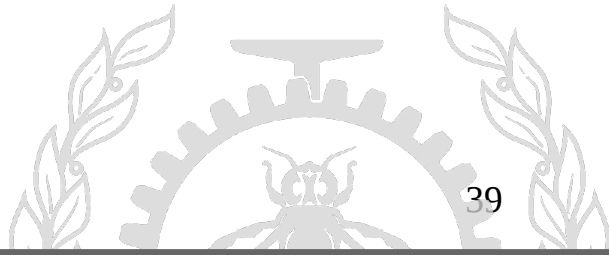
Indirect Branches

```
. ... Intel Processor Trace data: size 8544 bytes
.  00000000:  02 82 02 82 02 82 02 82 02 82 02 82 02 82 PSB
.  00000010:  00 00 00 00 00 00                         PAD
.  00000016:  19 ba 39 4d 7b 89 5e 04                   TSC 0x45e897b4d39ba
.  0000001e:  00 00 00 00 00 00 00 00                   PAD
.  00000026:  02 73 57 64 00 1c 00 00                   TMA CTC 0x6457 FC 0x1c
.  0000002e:  00 00                                     PAD
.  00000030:  02 03 27 00                               CBR 0x27
.  00000034:  02 23                                     PSBEND
.  00000036:  59 8b                                     MTC 0x8b
.  00000038:  59 8c                                     MTC 0x8c
.
.  00000304:  f8                                        TNT TTTTNN (6)
.  00000305:  06 00 00                                  TNT T (1)
.  00000308:  4d e0 3c 6d 9c                            TIP 0x9c6d3ce0
.  0000030d:  1c 00 00                                  TNT TTN (3)
.  00000310:  2d f0 3c                                  TIP 0x3cf0
.  00000313:  06                                        TNT T (1)
.  00000314:  59 2e                                     MTC 0x2e
.  00000316:  94                                        TNT NNTNTN (6)
.  00000317:  a8                                        TNT NTNTNN (6)
.  00000318:  a6                                        TNT NTNNTT (6)
```

Intel PT

Program Control Flow (simple-pt)

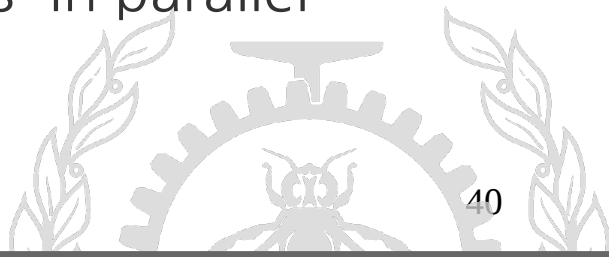
```
826.329    [+0.019]       [+   7]          intmod_ioctl+88
                          [+   7]          intmod_ioctl+148 -> trace_hardirqs_off
                          [+   3]          intmod_ioctl+290 -> trace_hardirqs_on
826.490    [+0.161]       [+   5]              trace_hardirqs_on+27
                          [+   8]          intmod_ioctl+184 -> trace_hardirqs_off
                          [+   2]          intmod_ioctl+193 -> __ndelay
826.503    [+0.013]       [+  10]              __ndelay+38 -> delay_tsc
                          [+  10]                  delay_tsc+24 -> preempt_count_add
826.507    [+0.003]       [+   9]                      preempt_count_add+27
.
.
.
.
.
826.559    [+0.008]       [+   3]                  delay_tsc+157
                          [+   1]          intmod_ioctl+198 -> trace_hardirqs_on
826.573    [+0.014]       [+   7]          intmod_ioctl+118
```



Intel PT Benchmarks

Synthetic Tests

- Test Setup
  - Skylake i5 6600K (3.9Ghz), controlled with simple-pt
  - 2MB trace buffer
  - MTC threshold (TSC update) every 512 cycles (accuracy)
  - PSB packet every 2K bytes (better decoder sync and recovery)
- Wide range of experiments
  - Image processing to arithmetic intensive tests
  - Mainly memory overhead as PT hardware is 'in parallel'
  - Overhead of V8 test



Intel PT Benchmarks

Synthetic Tests

Benchmark	Overhead (%)	Overhead V8 (%)
TailFact	22.91	-
ParseInt	9.65	10.36
Fib	5.86	5.80
RandMatStat	2.58	20.00
Canny_NoOptimize	2.55	-
PiSum	2.47	6.20
Canny_Optimize	2.34	-
Sort	1.05	6.06
RandMatMul	0.83	11.08

Lots of TIP packets

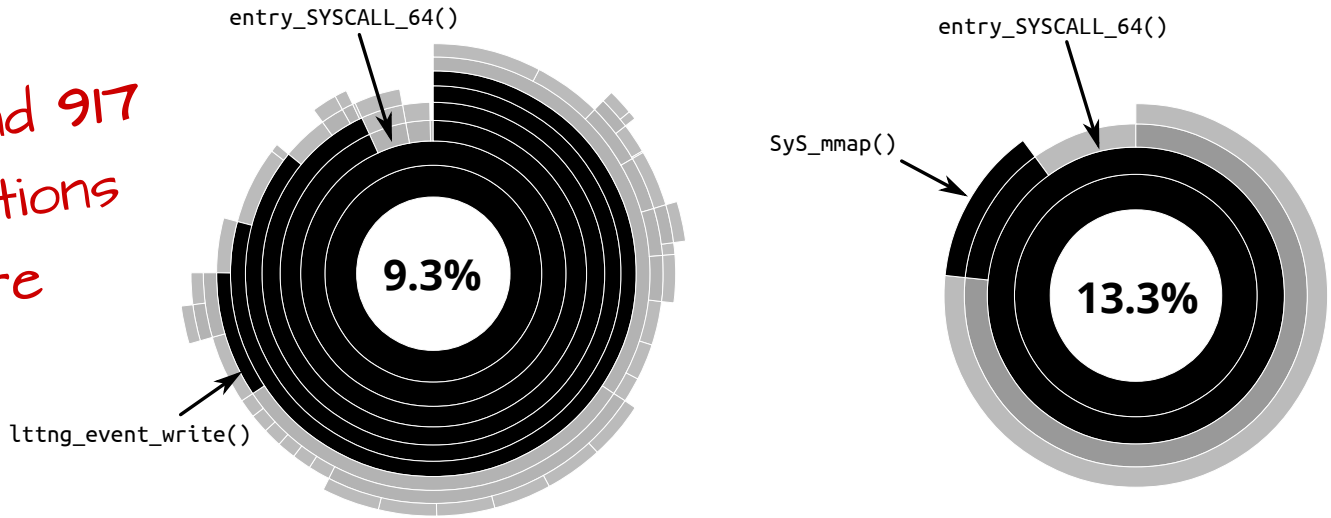
DGEMM using Intel AVX

Intel PT

Tracing the Tracers (Syscall Latency)

- Targeted snapshot of callstacks - `mmap()` example

173 ns and 917 instructions more

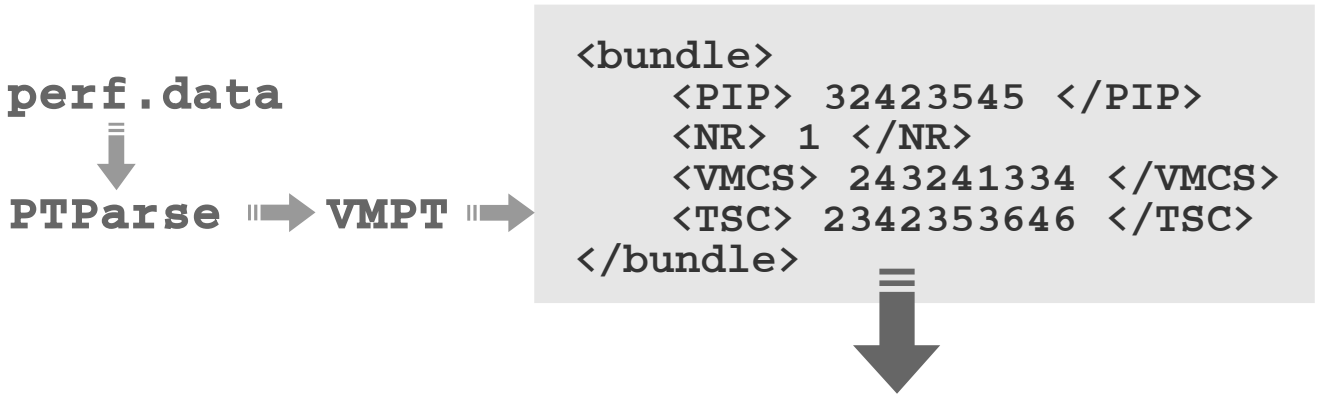


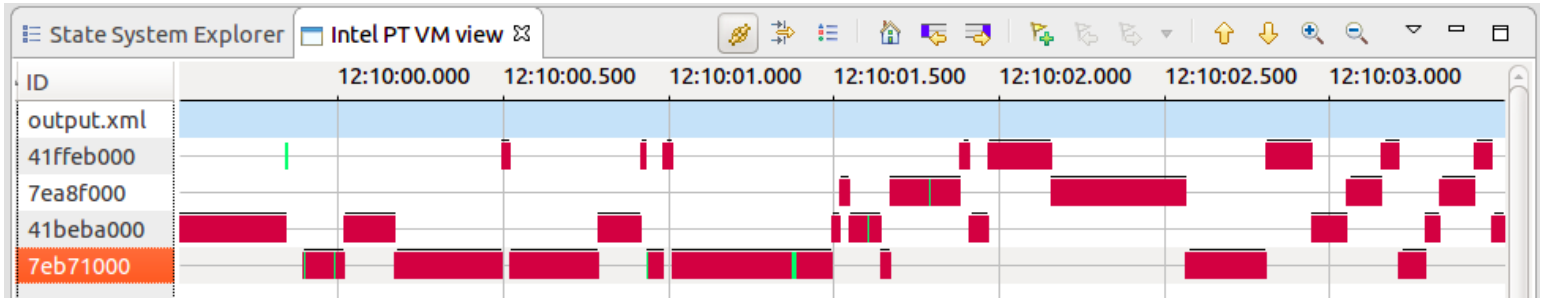
Timing Granularity

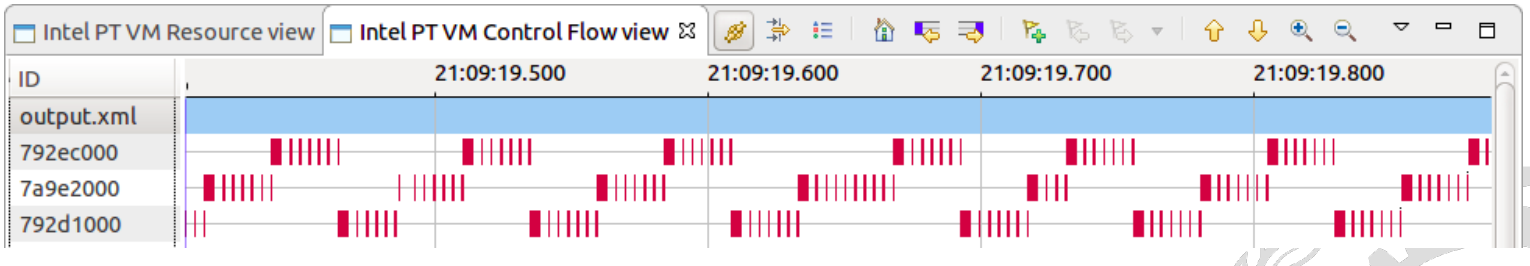
- Can vary from milliseconds (highly coarse) to around 14ns (CYC mode)

Intel PT

VM Analysis





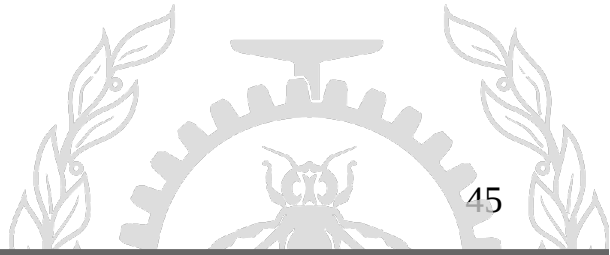


ARM

CoreSight

ARM CoreSight

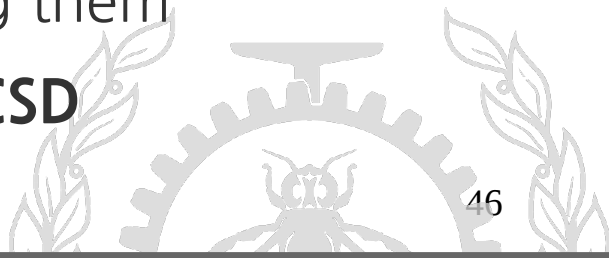
- Collection of trace hardware blocks for ARM
- Advancements from ETMv1, v3 and PFT → **ETMv4** [4]
- Program Flow Trace and Data Trace
  - PE → Trace Router → System Bus → System RAM
  - PE → Trace FIFO → TPIU → External Hardware
  - Can be configured as desired on silicon [9]
- Maturing Linux Kernel and vendor support for multiple chips



ARM CoreSight

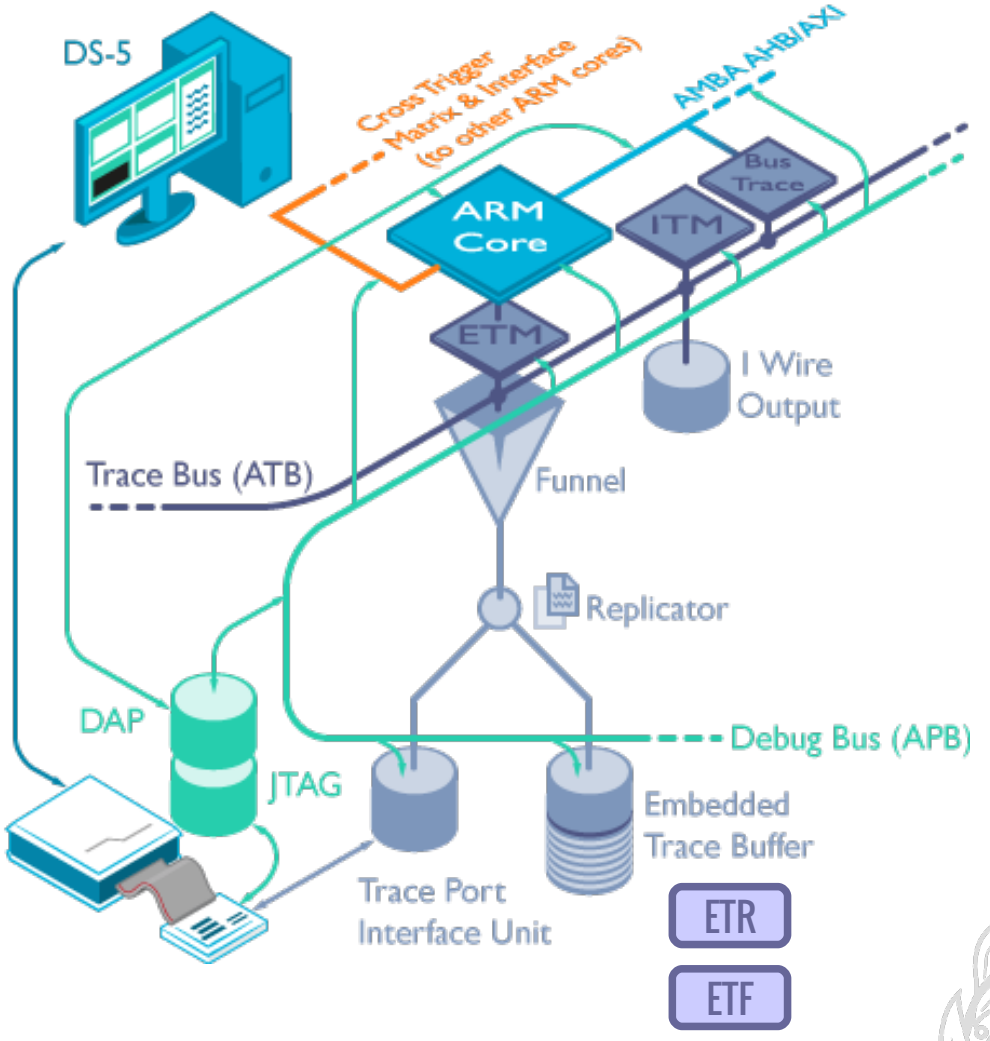
ETMv4

- Major revision [4], highly configurable
  - Insn only for A family. Data+Insn only for M and R family
    - P0 (Insn), P1 and P2 (Data) elements, Other elements
    - **P0 :** Atom elements (E/N), Q elements (cycle count)
    - **P0 :** Branch, Synchronization, Exceptions, TimeStamp, Conditional (C), Result (R), Mispredict etc.
- Decoding same way as libipt (Intel PT decoder lib)
- **Trace Control** with CSAL as in HAT [10] or now, Perf!
  - Expose configuration registers by mmaping them
  - Trace start and stop. **Decoding** with **OpenCSD**



ARM CoreSight

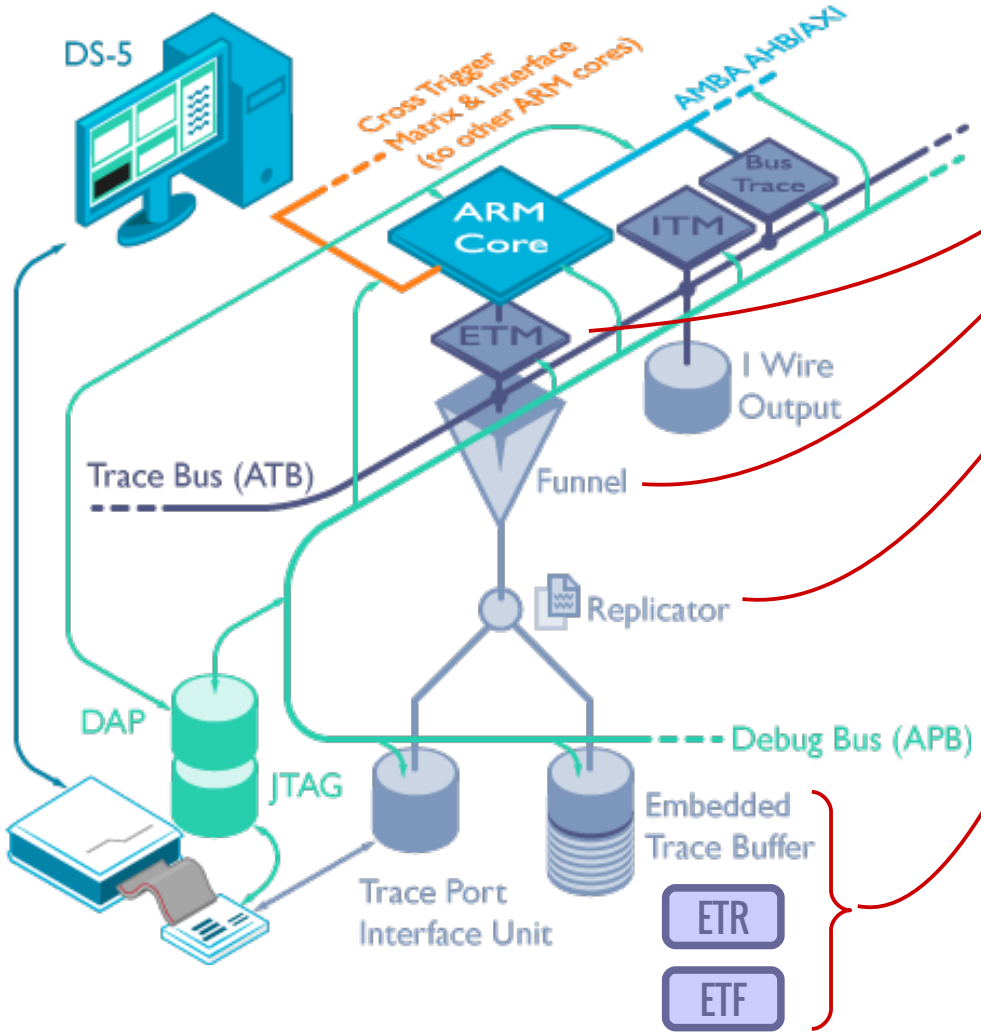
SoC View



Courtesy, ARM

ARM CoreSight

SoC View



Kernel View

Device Tree

CoreSight Driver

Configure Sink/Source

Trace Start/Stop bit

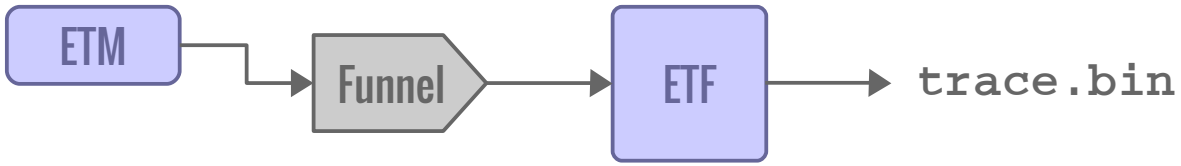
Encoded Trace

Courtesy, ARM

ARM CoreSight

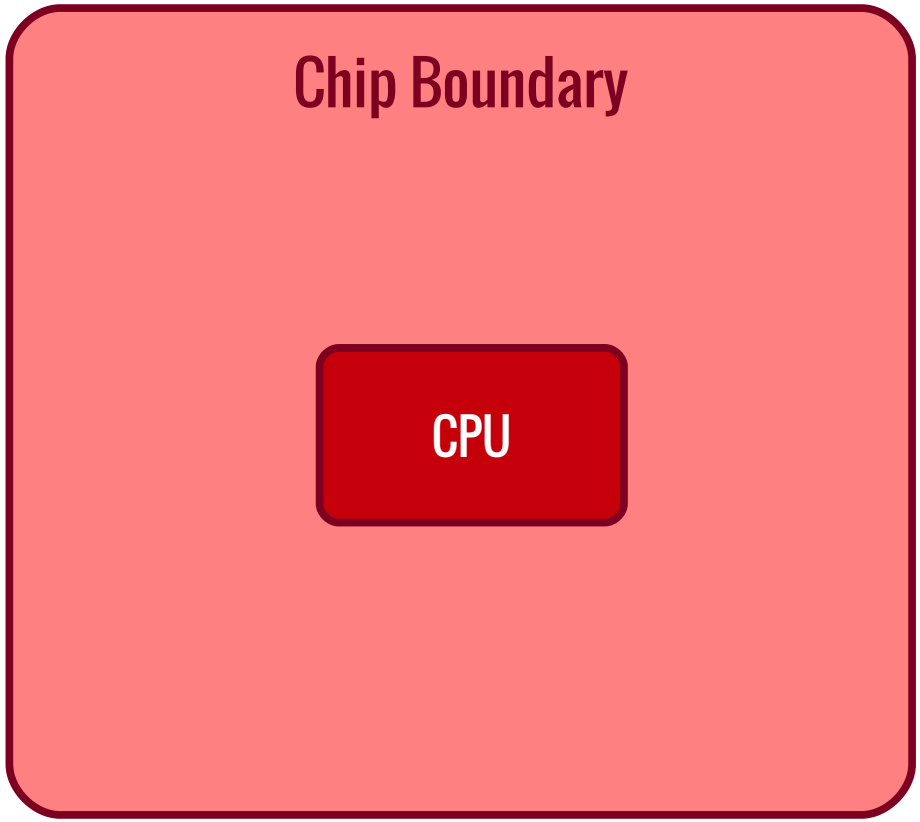
Experiments with Cortex-A53 (ARMv8)

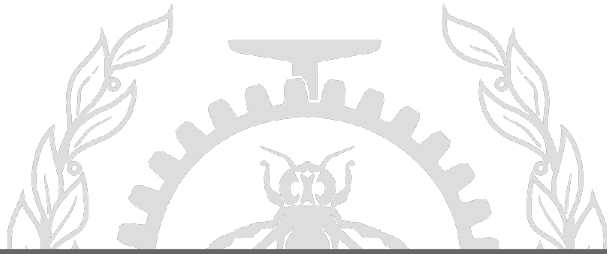
- Qualcomm Snapdragon 410 platform
- Configure ETM as source and ETF as sink with CS driver

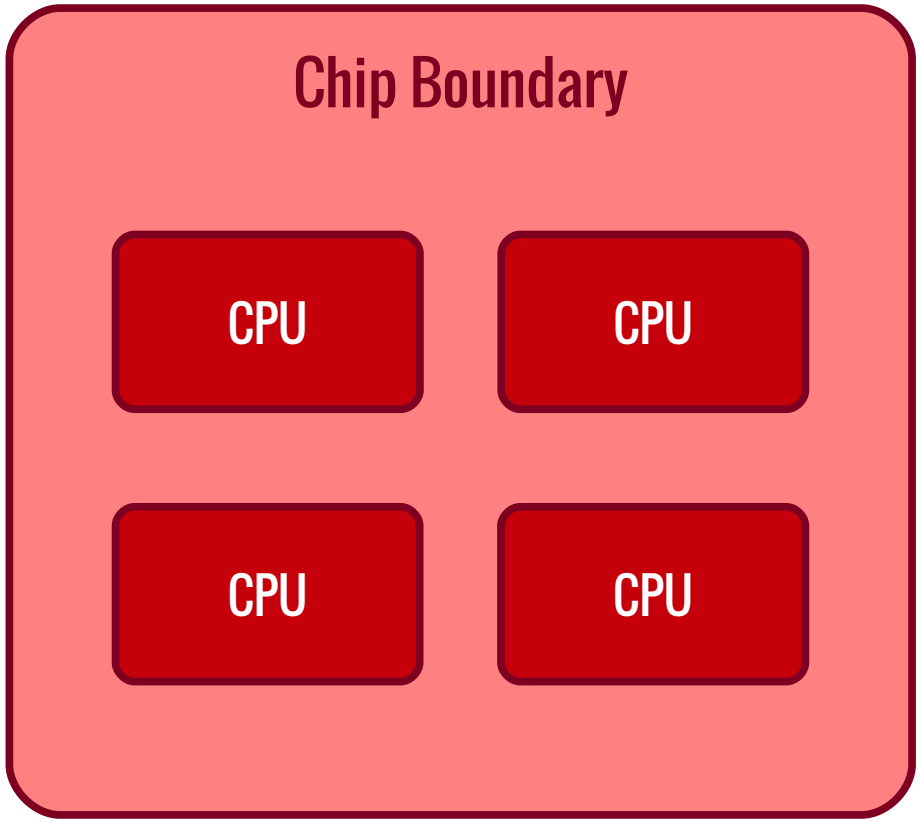


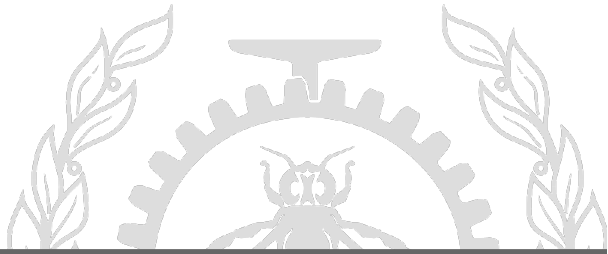
- Linaro's kernel, regular upstream in progress
- Decoding
  - **`ptm2human`** [5] decodes ETMv4 packets but is not as mature as other alternatives. Not recommended
  - **OpenCSD** (*https://github.com/Linaro/OpenCSD/tree/arm-dev*) provides ETMv4 debugging. Perf integration is underway.

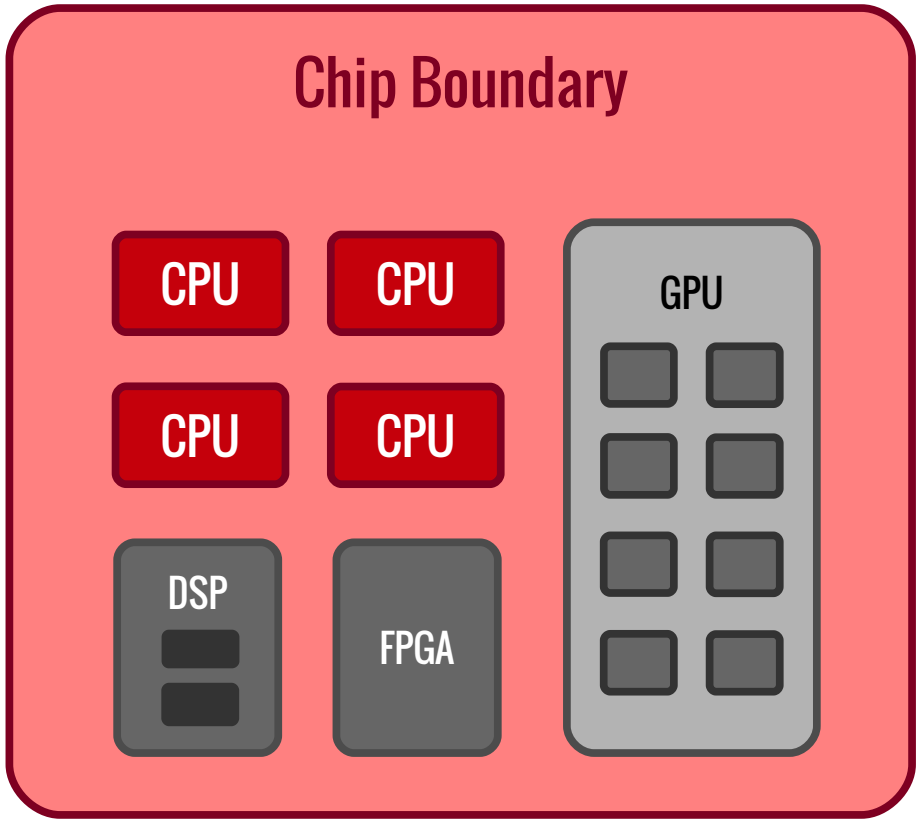
Baremetal Tracing

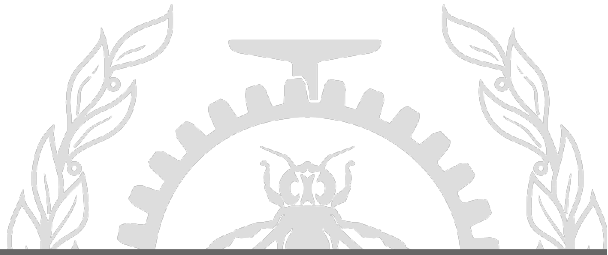

Chip Boundary
CPU




Chip Boundary
CPU
CPU
CPU
CPU



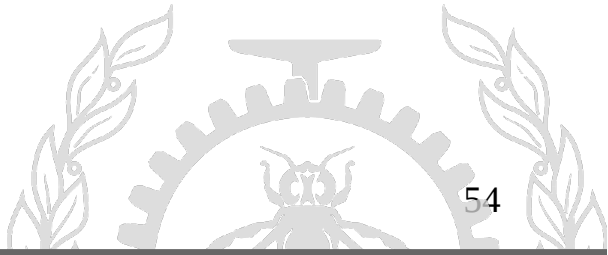

Chip Boundary
CPU
CPU
CPU
CPU
GPU
DSP
FPGA



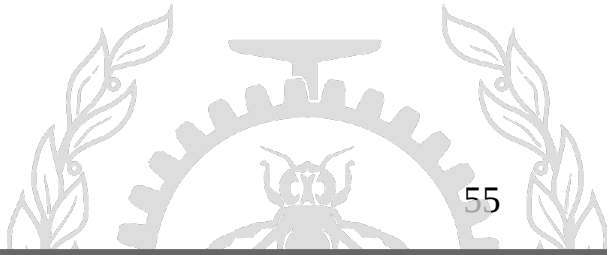
Heterogeneous System

TI Keystone 2

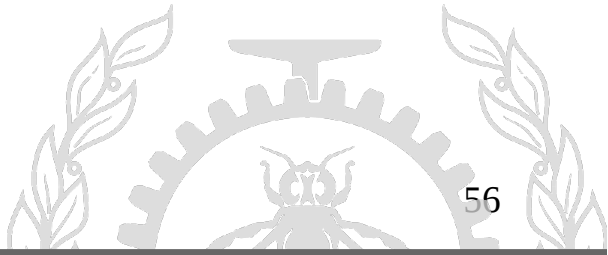
- 66AK2H12 SoC
- 4 x ARM Cortex A15 core
- 8 x C66 CorePacs DSP
  - Real-time micro-kernel - SYS/BIOS
  - Inter-core communication with an API
- 2GB DDR RAM
- 6MB Shared Memory
- All the latest and cool peripheral controllers



- How can we approach tracing such heterogeneous systems?
- Is it possible to gather traces from multiple cores?
- How would we synchronize such traces and view them?



There is a solution…

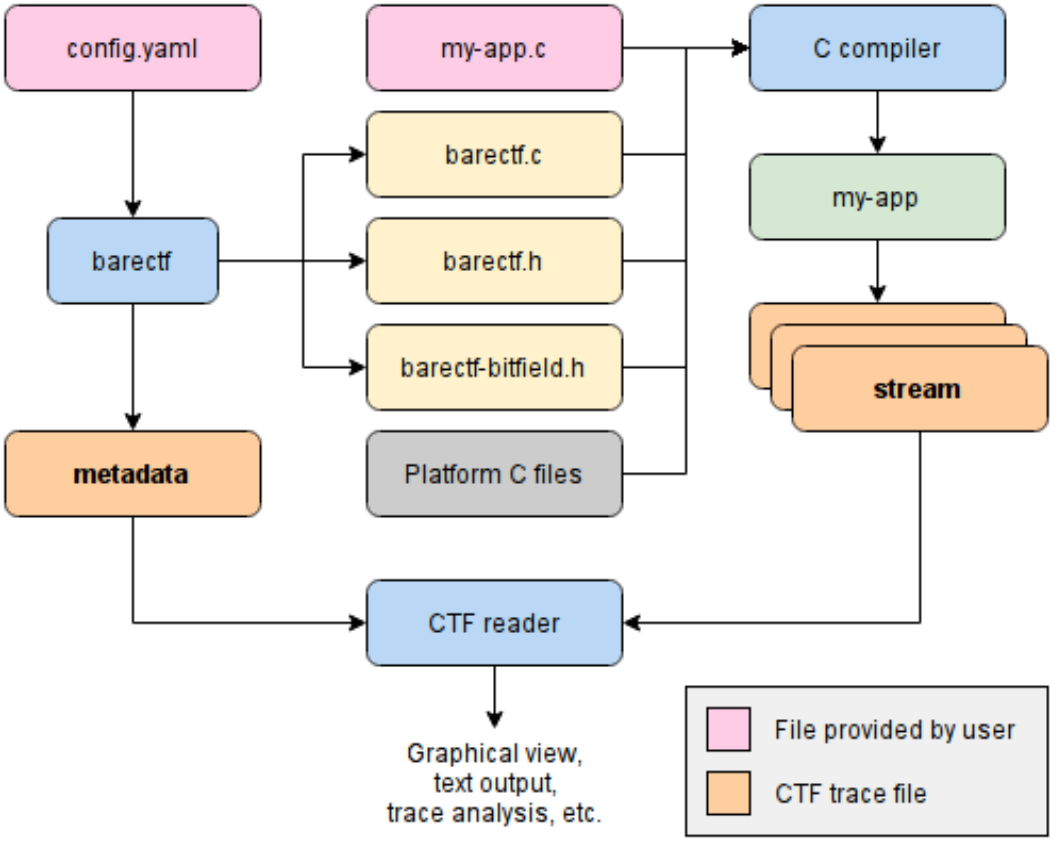


There is a solution…

barectf

BareCTF

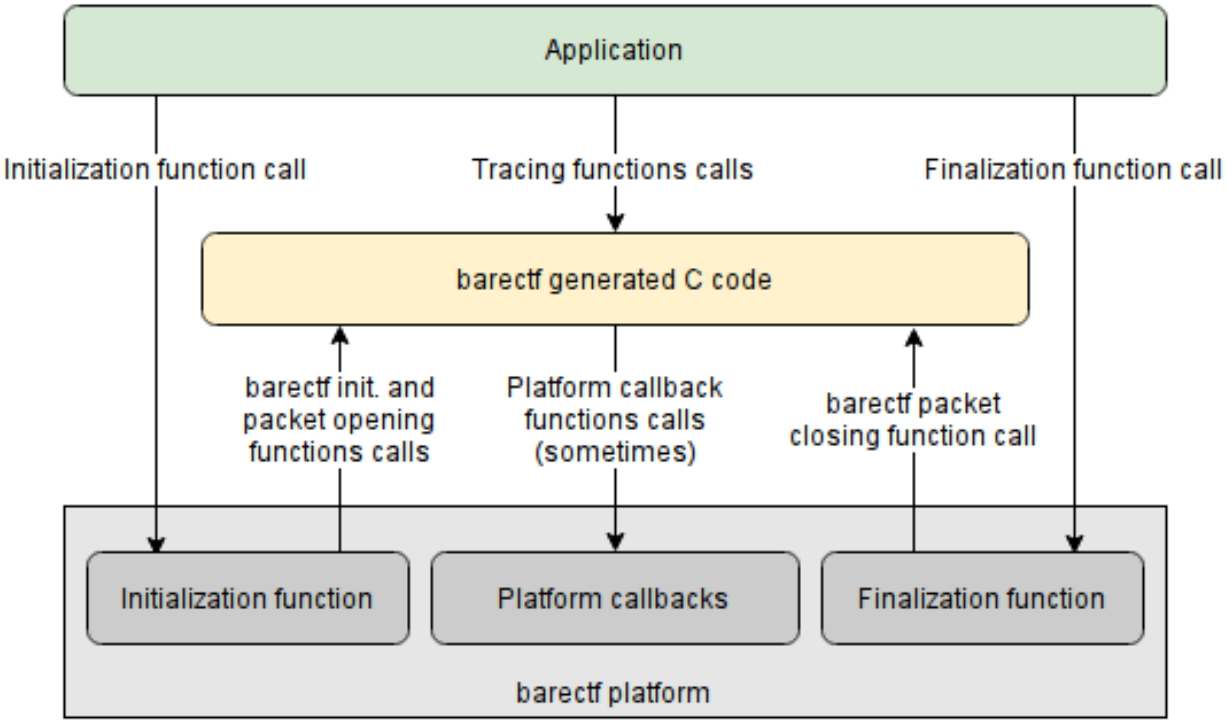
Workflow



Courtesy, Philippe Proulx (EfficiOS) : *http://lttng.org/blog/2015/07/21/barectf-2/*

BareCTF

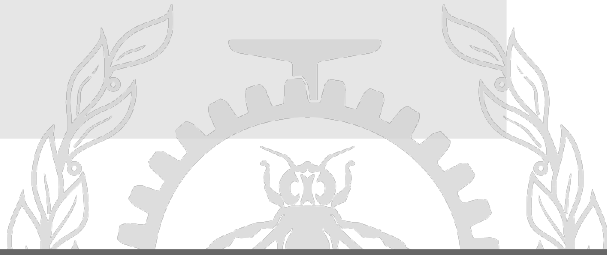
Workflow



Courtesy, Philippe Proulx (EfficiOS) : *http://lttng.org/blog/2015/07/21/barectf-2/*

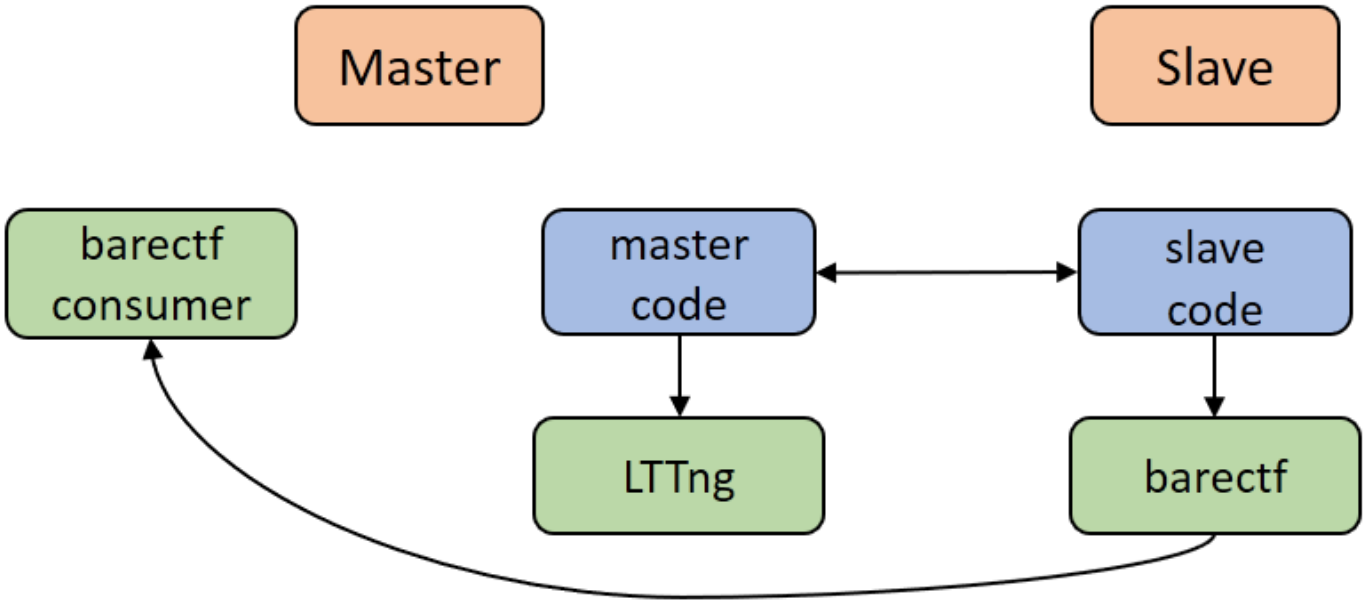
Platform Implementation

```
/* Instantiates the tracing context and initializes every structure
needed to take care of the recorded events and stored packets.*/
int8_t barectf_init(void);

/* Returns the tracing context.*/
barectf_ctx_t *barectf_get_ctx(void);

/* Specifies the way to access 64bits timestamps on the targeted
device.*/
uint64_t barectf_get_clock(void *ctx);

/* Initializes (if needed) the counter used to get timestamps.*/
void barectf_init_clock(void *ctx);

/* Opens a new packet containing recorded events.*/
void barectf_open_packet(void *ctx);

/* Takes care of a full packet. The packet can be sent to the host, put
in another memory location, discarded... */
void barectf_close_packet(void *ctx);

/* Finalizes the tracing session.*/
int8_t barectf_close(void);
```



Experimentation

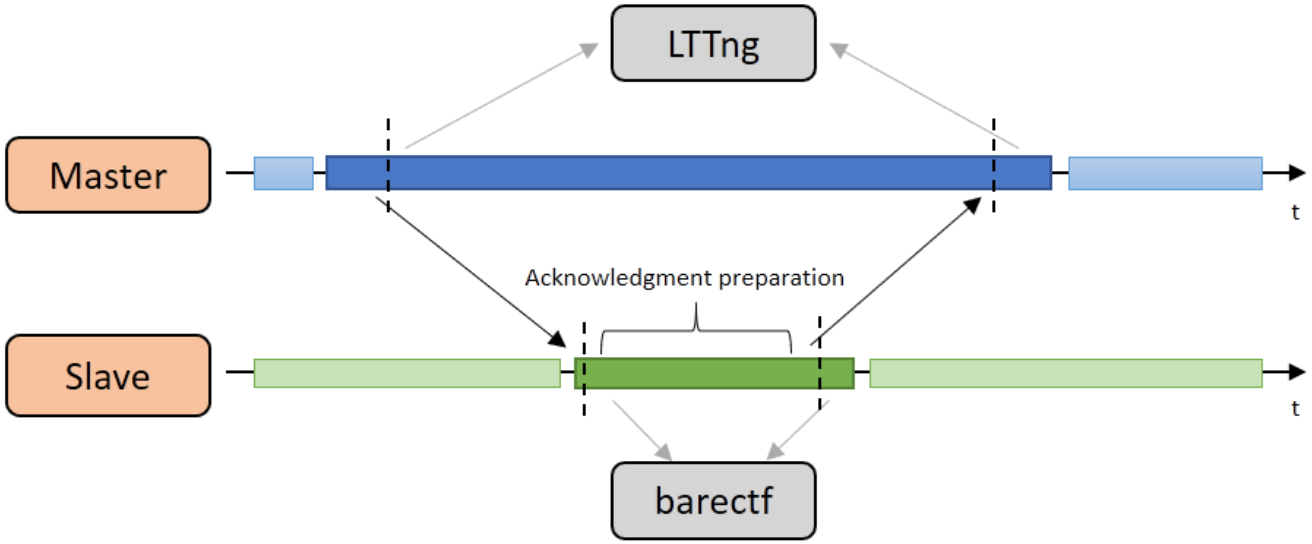
Workflow



Experimentation

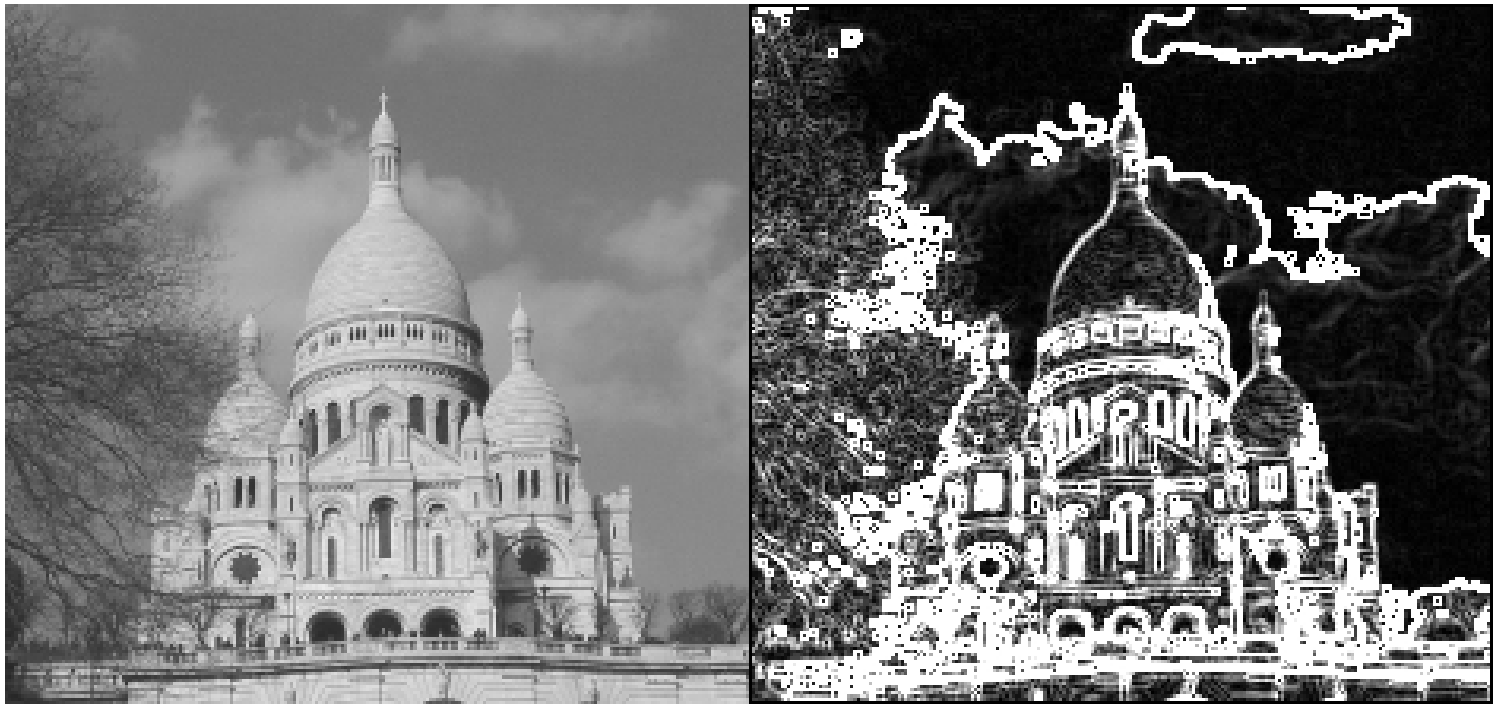
Synchronization

- ✔ *Master* can interrupt a *Slave* processor [MessageQ]
- ■ Shared memory between Master or Slave



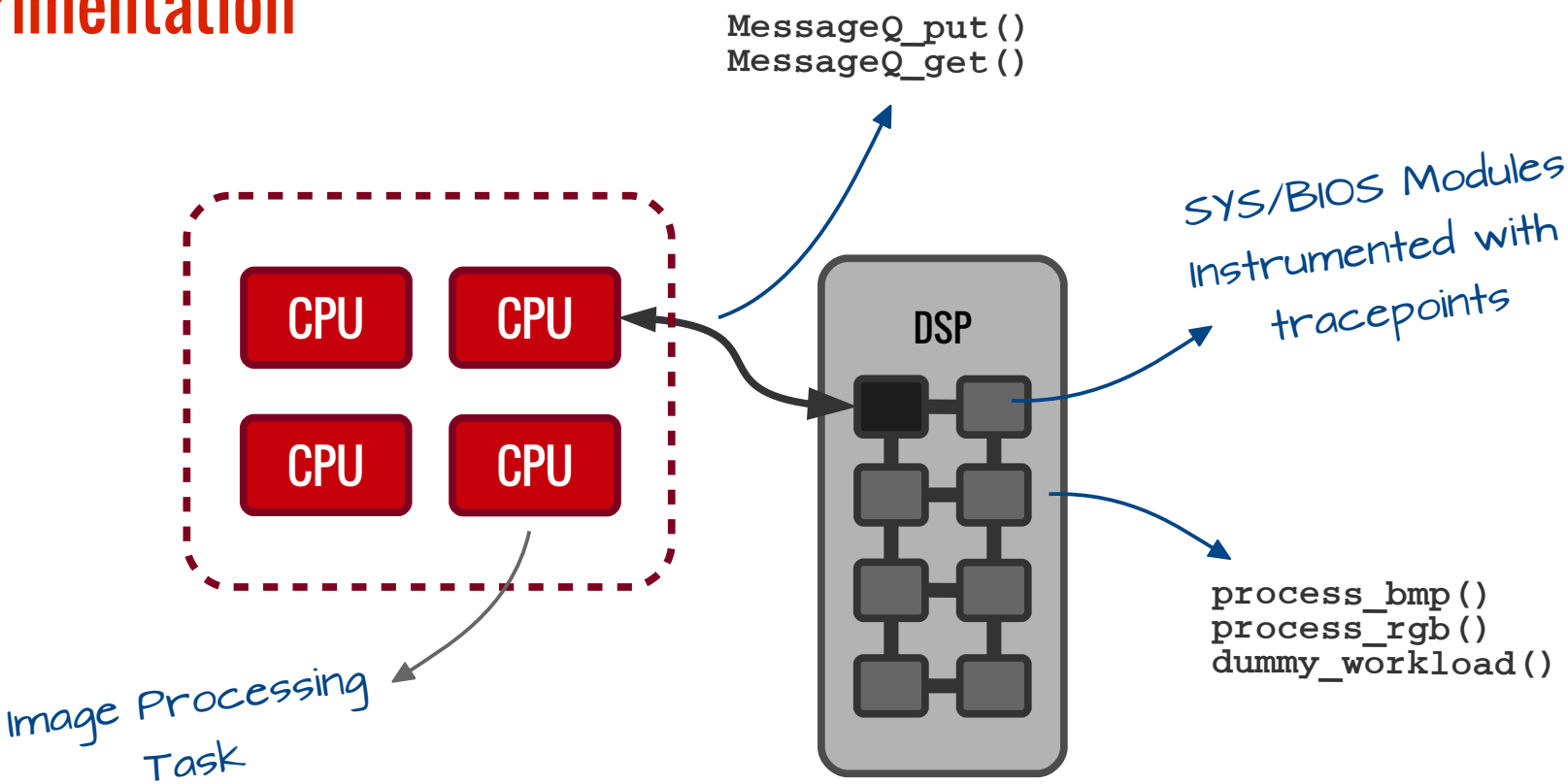
Experimentation

Sobel's Filter for Edge Detection



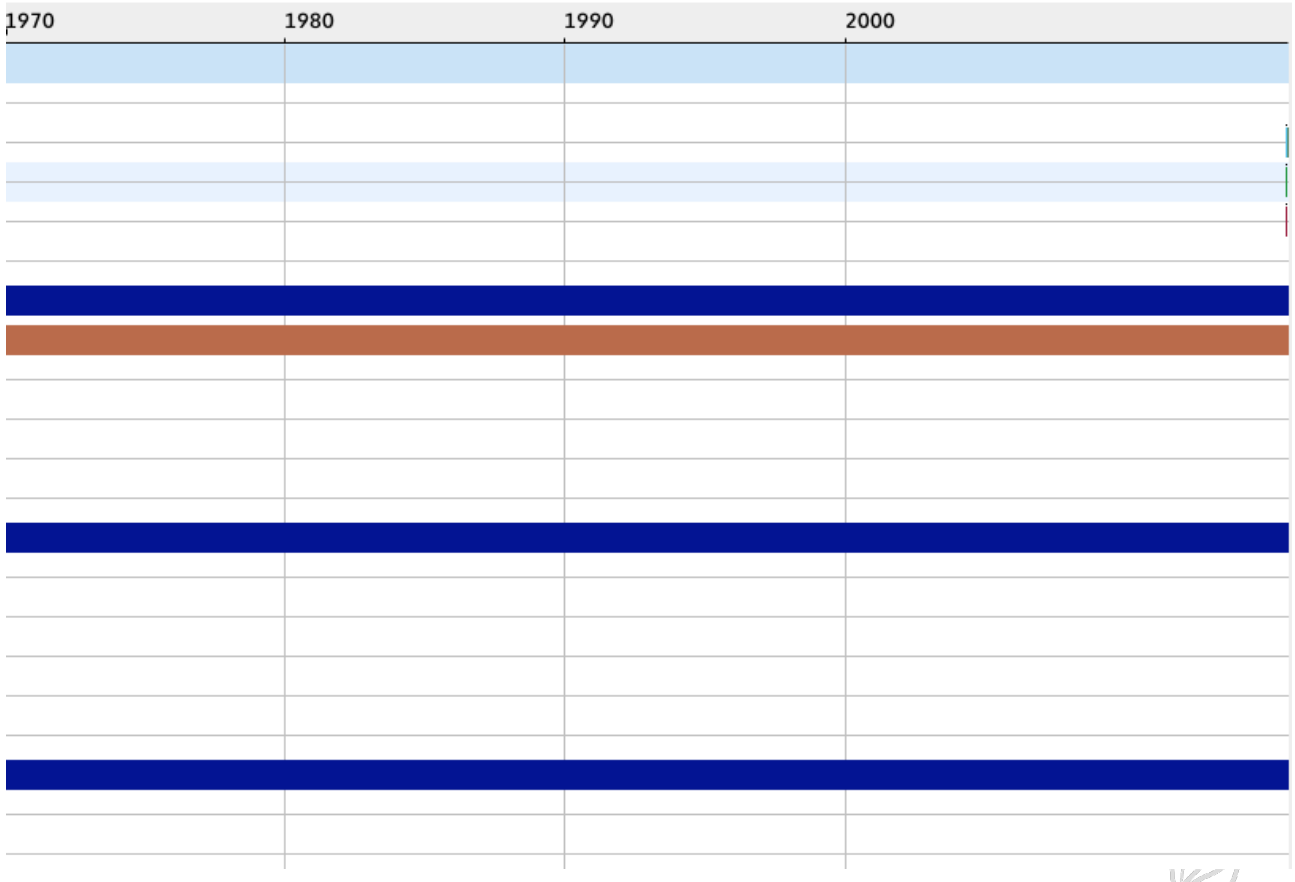
Experimentation

Experimentation



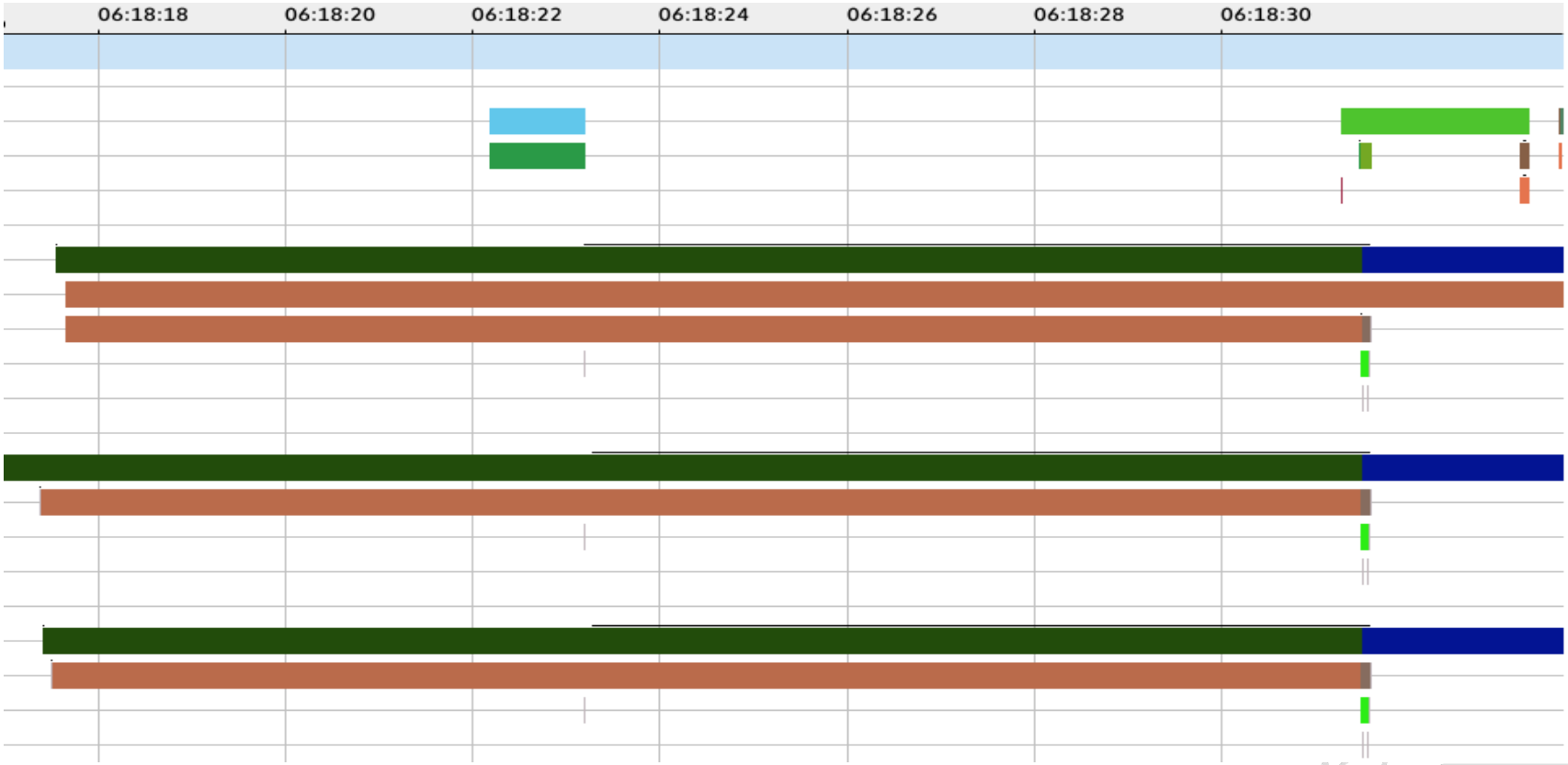
Tracing

Trace View (Non-Synchronized)



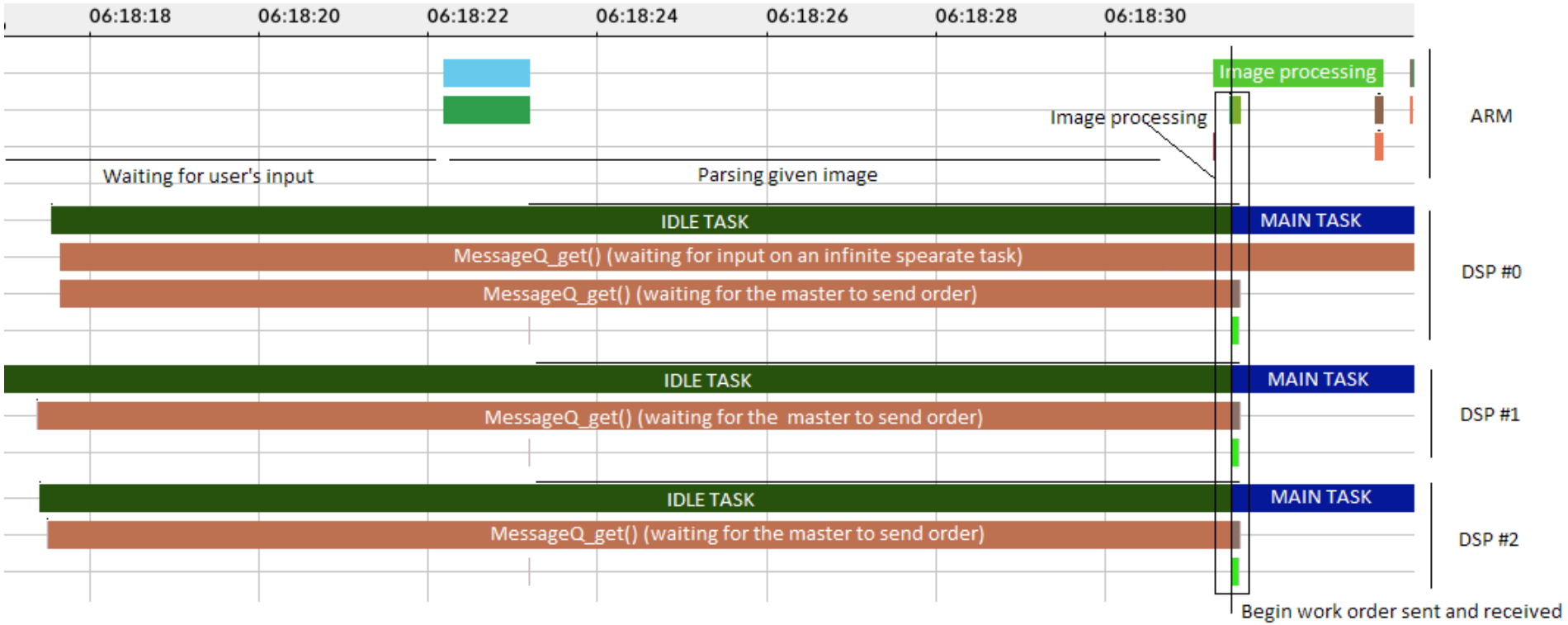
Tracing

Trace View (Synchronized)



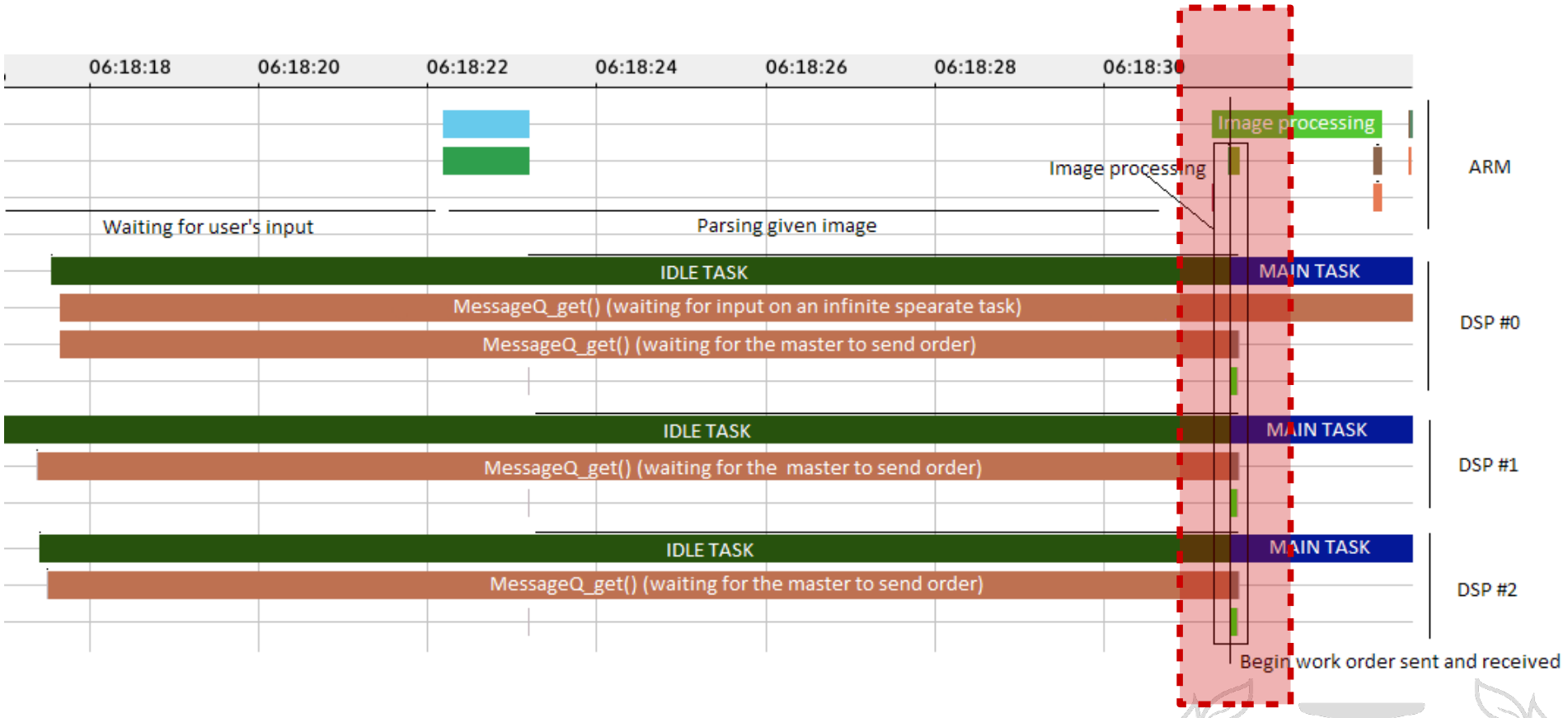
Tracing

Trace View (Synchronized)



Tracing

Trace View (Synchronized)

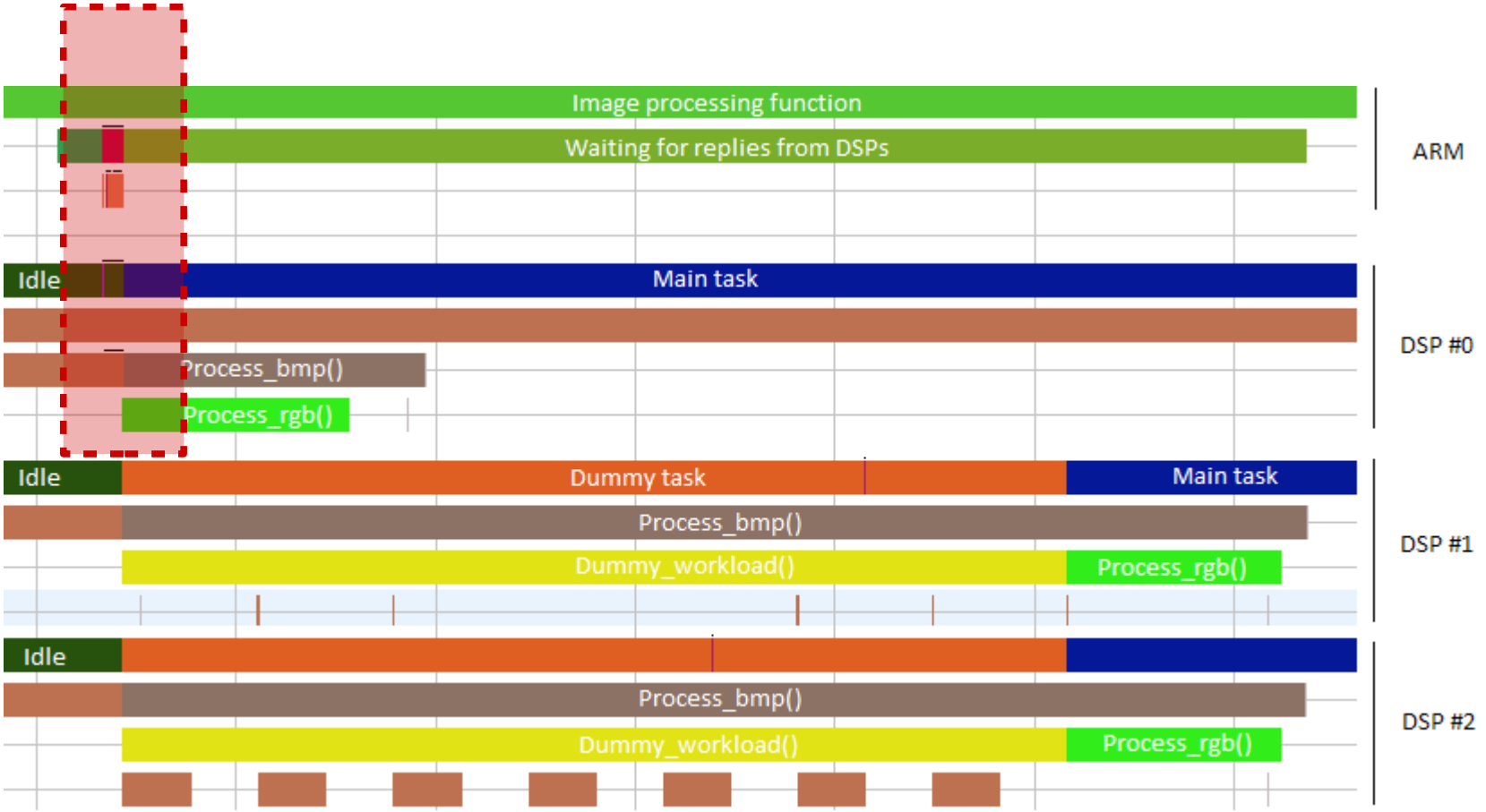


Tracing

Trace View (Synchronized)

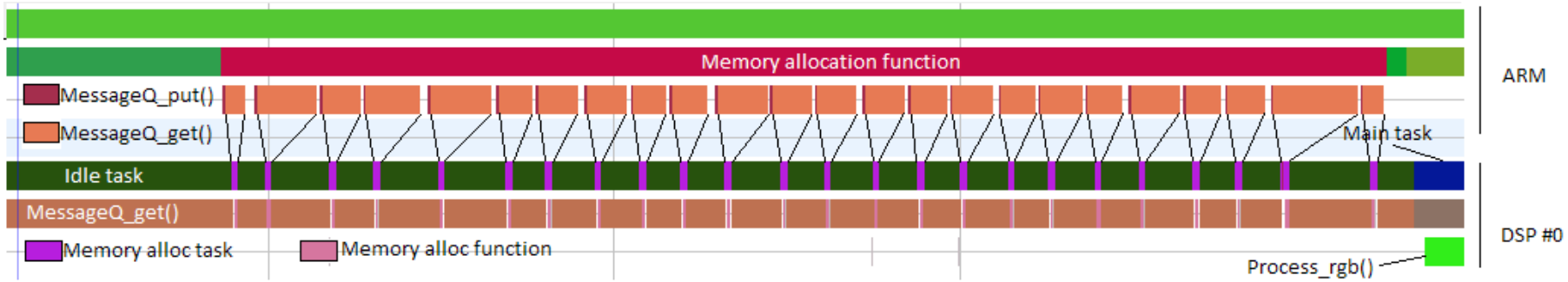
Tracing

Trace View (Synchronized)



Tracing

Trace View (Synchronized)



Tracing

Detecting Unwanted Latency

Detecting Unwanted Latency

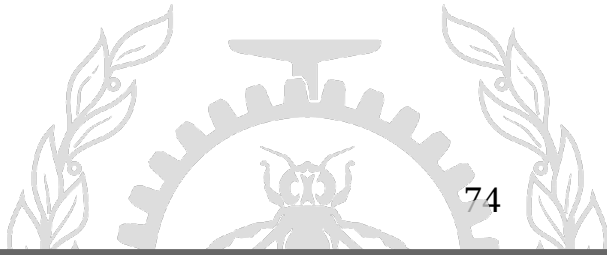
The Future

Hardware Tracing

- Perf is emerging as a standard way to use hardware tracing
- Accurate instruction profiling
- Applications in security, performance, dynamic code analysis
- Trace Viewing integration with TraceCompass

Baremetal Tracing

- Moving towards standardization
  - HSA standards specify Performance Counter APIs
  - Dedicated APIs for trace gathering as well
- Better Trace Viewing and Analysis



References

[1] Debug and Trace for Multicore SoCs, ARM Whitepaper, 2008

[2] Intel 64 and IA-32 Architecture Software Developer Manual

[3] Adding processor Trace Support to Linux [LWN] http://lwn.net/Articles/648154/

[4] ARM® Embedded Trace Macrocell Architecture Specification (ETMv4.0 to ETMv4.2)

[5] *ptm2human* : https://github.com/hwangcc23/ptm2human

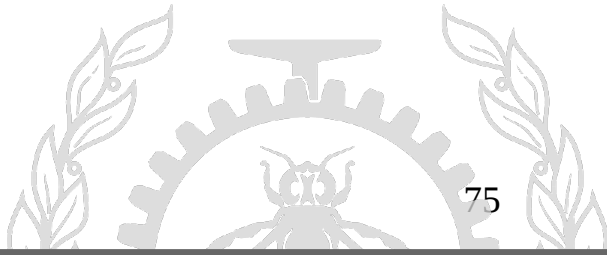
[6] Venkitaraman R, Gupta G, *Static Program Analysis of Embedded Executable Assembly Code*, ACM CASES 2004

[7] Debugging embedded systems : using hardware tricks to trace program flow, EDN, 1998, ProQuest pp163

[8] Exploiting Hardware Advances for Software Testing and Debugging, ICSE'11, ACM

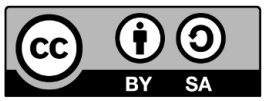
[9] CoreSight Trace Memory Controller Technical Reference Manual

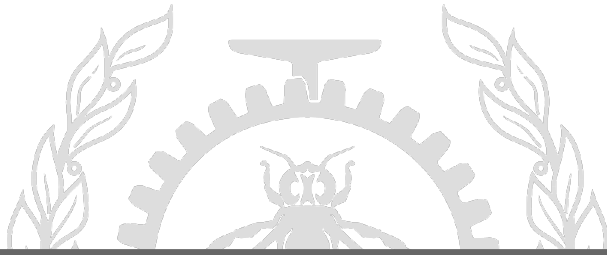
[10] Vittuci C, Larsson A, *HAT, Hardware Assisted Trace: Performance Oriented Trace & Debug System,* ICSSEA 2015.



References

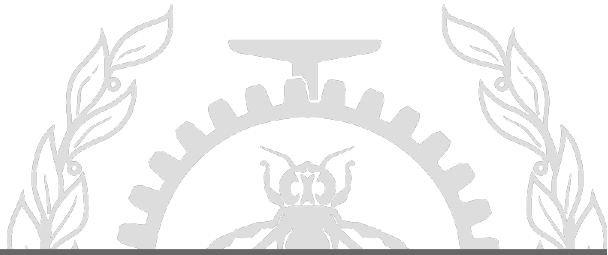
- HSA Foundation Standards : http://www.hsafoundation.com/standards/
- OpenCSD : https://github.com/Linaro/OpenCSD
- CoreSight, Perf and the OpenCSD Library: http://www.linaro.org/blog/core-dump/coresight-perf-and-the-opencsd-library/
- barectf 2: Continuous Bare-Metal Tracing on the Parallella Board : http://lttng.org/blog/2015/07/21/barectf-2/





Acknowledgments

Thanks to EfficiOS, Ericsson Montréal and DORSAL Lab, Polytechnique Montreal for the awesome work on LTTng/UST, TraceCompass and BareCTF. Thanks to DiaMon Workgroup for this opportunity to present.



Questions?

suchakrapani.sharma@polymtl.ca

thomas.bertauld@gmail.com

@tuxology

http://suchakra.in