Memfault

Employing Coredumps to Debug Your Embedded Devices

Eric Johnson

Eric Johnson

Firmware Solutions Engineer, Memfault

- Contributor of humble bug fixes to BLE, Kernel, Shell subsystems of Zephyr
- Previously: Walgreens Health, Athos, Acuity Brands, Lexmark
- Can find my thoughts and content on Memfault's Interrupt blog (interrupt.memfault.com)

Memfault

Agenda

1. Coredumps Overview
2. Coredumps with Zephyr
3. Zephyr Coredump Demo
4. Coredumps with Memfault
5. Memfault Coredump Demo

Poll #1

Do you use coredumps to debug crashes?

A. Yes, regularly

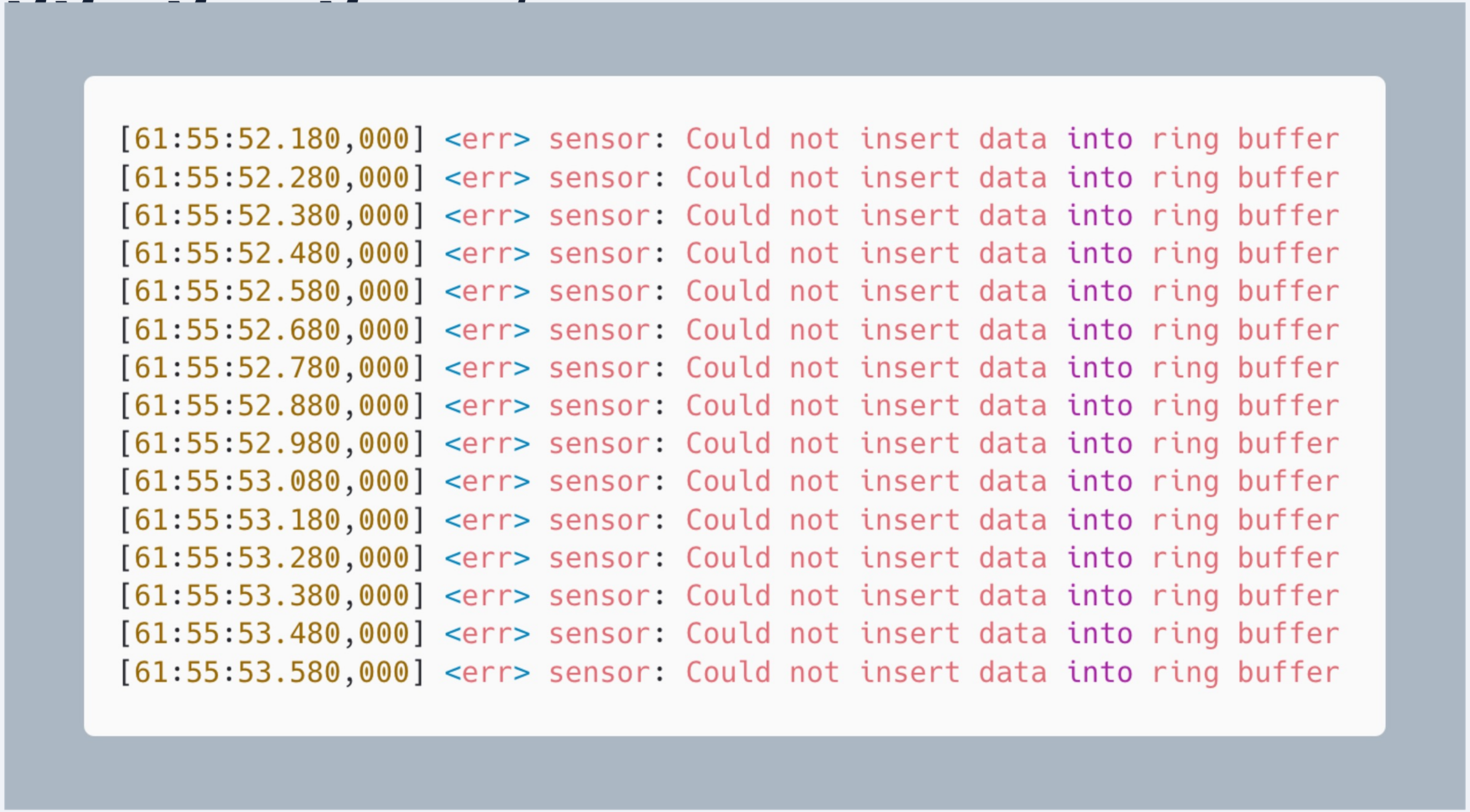
B. Yes, a handful of times

C. No, workflow not established

D. No, what is a coredump?

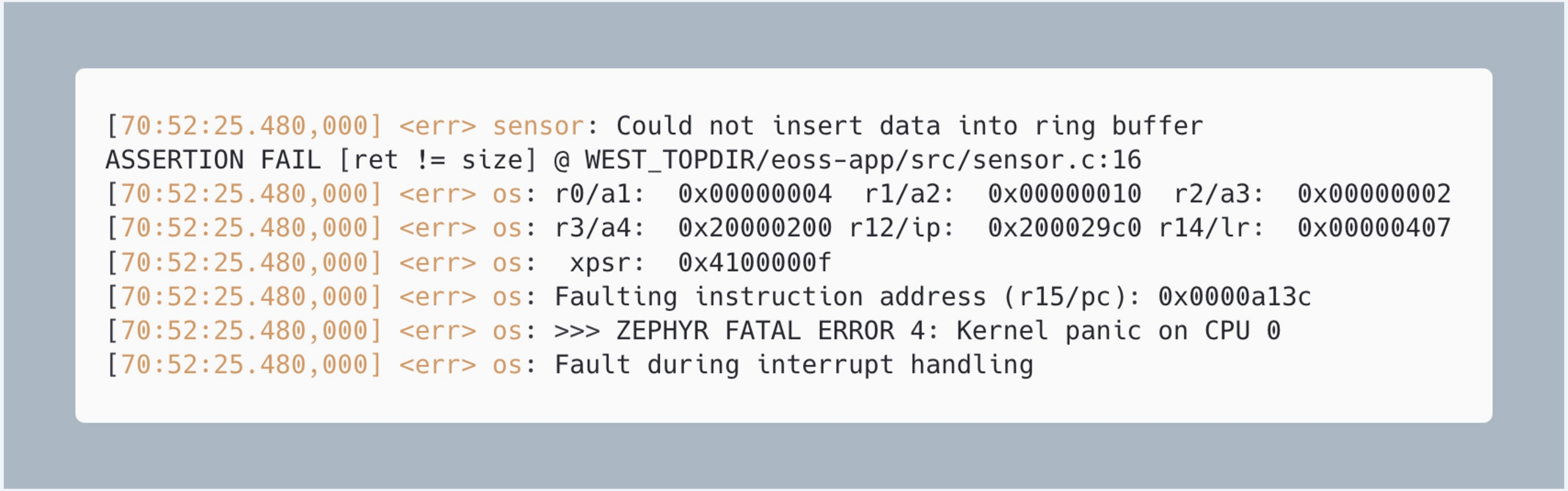
Congratulations! Your device is deployed!

Logging is great, until…



```
[61:55:52.180,000] <err> sensor: Could not insert data into ring buffer
[61:55:52.280,000] <err> sensor: Could not insert data into ring buffer
[61:55:52.380,000] <err> sensor: Could not insert data into ring buffer
[61:55:52.480,000] <err> sensor: Could not insert data into ring buffer
[61:55:52.580,000] <err> sensor: Could not insert data into ring buffer
[61:55:52.680,000] <err> sensor: Could not insert data into ring buffer
[61:55:52.780,000] <err> sensor: Could not insert data into ring buffer
[61:55:52.880,000] <err> sensor: Could not insert data into ring buffer
[61:55:52.980,000] <err> sensor: Could not insert data into ring buffer
[61:55:53.080,000] <err> sensor: Could not insert data into ring buffer
[61:55:53.180,000] <err> sensor: Could not insert data into ring buffer
[61:55:53.280,000] <err> sensor: Could not insert data into ring buffer
[61:55:53.380,000] <err> sensor: Could not insert data into ring buffer
[61:55:53.480,000] <err> sensor: Could not insert data into ring buffer
[61:55:53.580,000] <err> sensor: Could not insert data into ring buffer
```

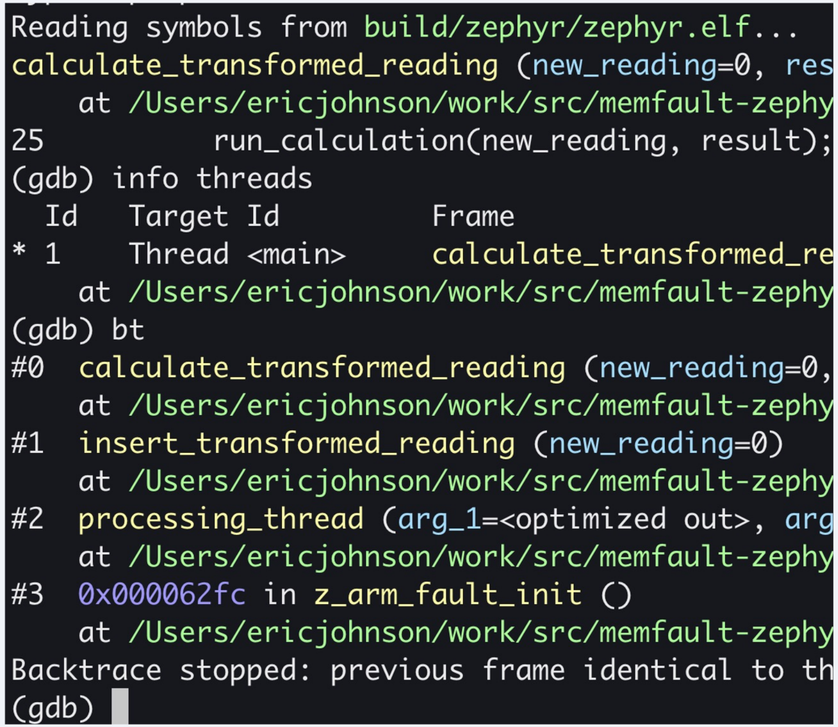
Panics are great, but only show 1 frame



```
[70:52:25.480,000] <err> sensor: Could not insert data into ring buffer
ASSERTION FAIL [ret != size] @ WEST_TOPDIR/eoss-app/src/sensor.c:16
[70:52:25.480,000] <err> os: r0/a1:  0x00000004  r1/a2:  0x00000010  r2/a3:  0x00000002
[70:52:25.480,000] <err> os: r3/a4:  0x20000200 r12/ip:  0x200029c0 r14/lr:  0x00000407
[70:52:25.480,000] <err> os:  xpsr:  0x4100000f
[70:52:25.480,000] <err> os: Faulting instruction address (r15/pc): 0x0000a13c
[70:52:25.480,000] <err> os: >>> ZEPHYR FATAL ERROR 4: Kernel panic on CPU 0
[70:52:25.480,000] <err> os: Fault during interrupt handling
```

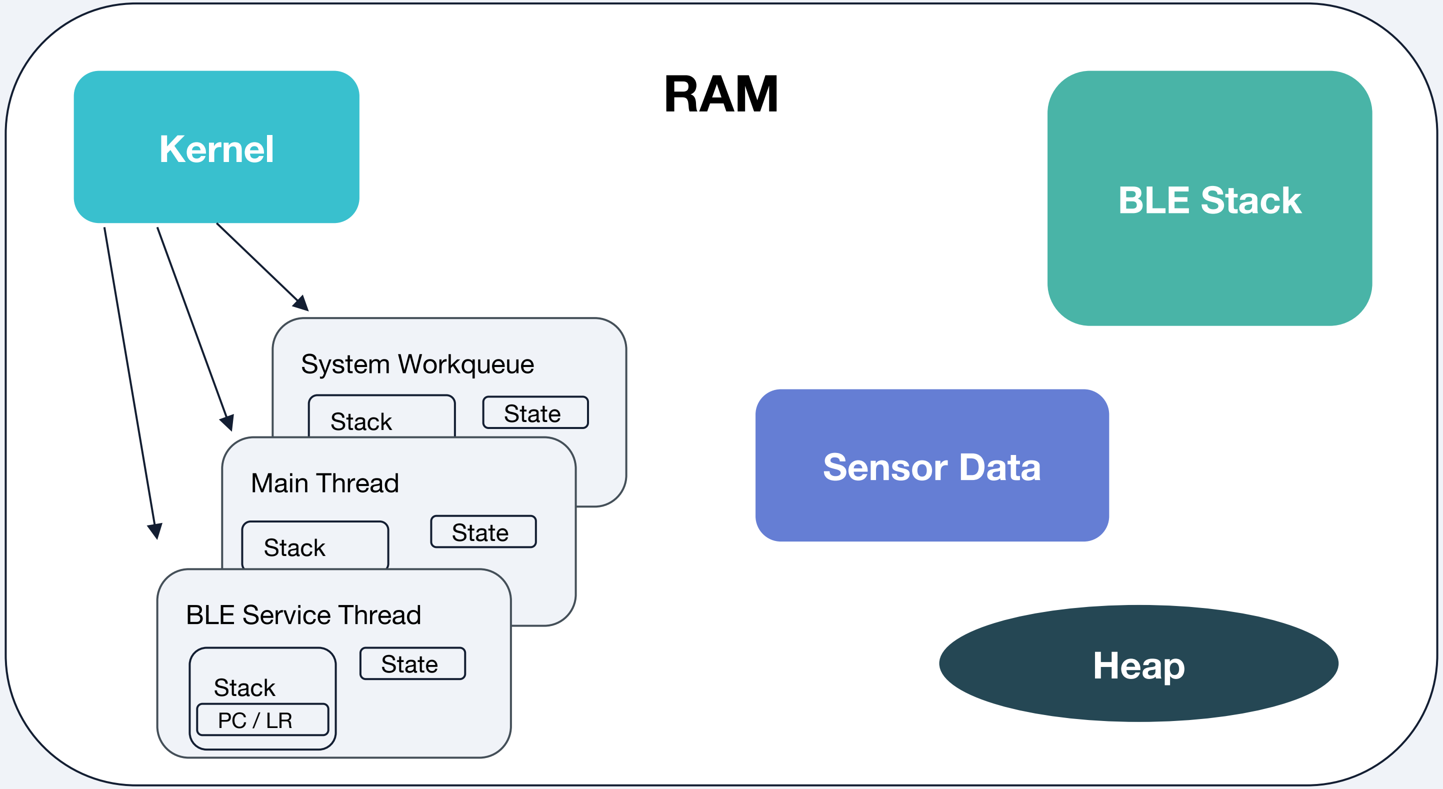
Coredumps

- Triggered by faults, kernel panics, asserts
- Captures registers and memory to allow for later analysis
- Data can be streamed out immediately or stored in non-volatile memory



```
Reading symbols from build/zephyr/zephyr.elf...
calculate_transformed_reading (new_reading=0, res
    at /Users/ericjohnson/work/src/memfault-zephy
25            run_calculation(new_reading, result);
(gdb) info threads
  Id   Target Id         Frame
* 1    Thread <main>     calculate_transformed_re
    at /Users/ericjohnson/work/src/memfault-zephy
(gdb) bt
#0  calculate_transformed_reading (new_reading=0,
    at /Users/ericjohnson/work/src/memfault-zephy
#1  insert_transformed_reading (new_reading=0)
    at /Users/ericjohnson/work/src/memfault-zephy
#2  processing_thread (arg_1=<optimized out>, arg
    at /Users/ericjohnson/work/src/memfault-zephy
#3  0x000062fc in z_arm_fault_init ()
    at /Users/ericjohnson/work/src/memfault-zephy
Backtrace stopped: previous frame identical to th
(gdb) 
```

Coredump Components



Poll #2

What RTOS is your firmware based on?

A. Zephyr

B. FreeRTOS

C. Other RTOS

D. Bare-metal

Agenda

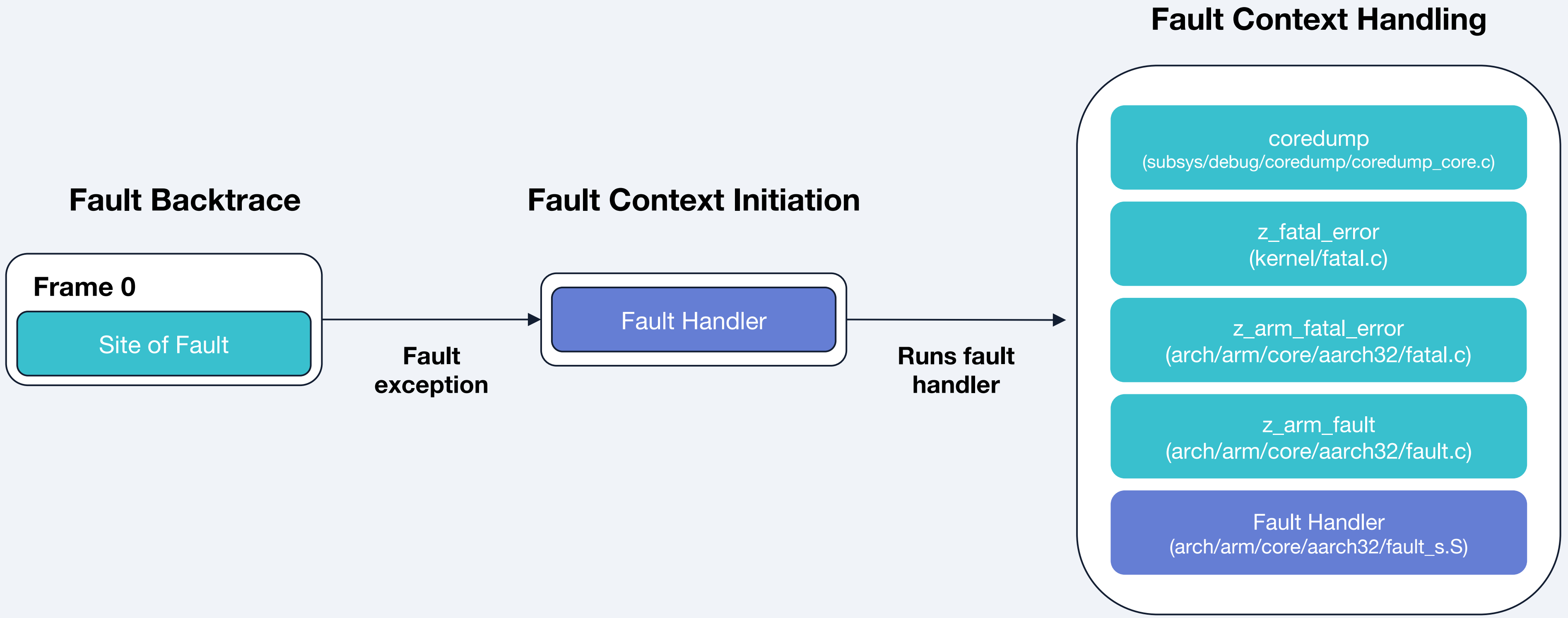
1. Coredumps Overview
2. Coredumps with Zephyr
3. Zephyr Coredump Demo
4. Coredumps with Memfault
5. Memfault Coredump Demo

What causes a coredump?

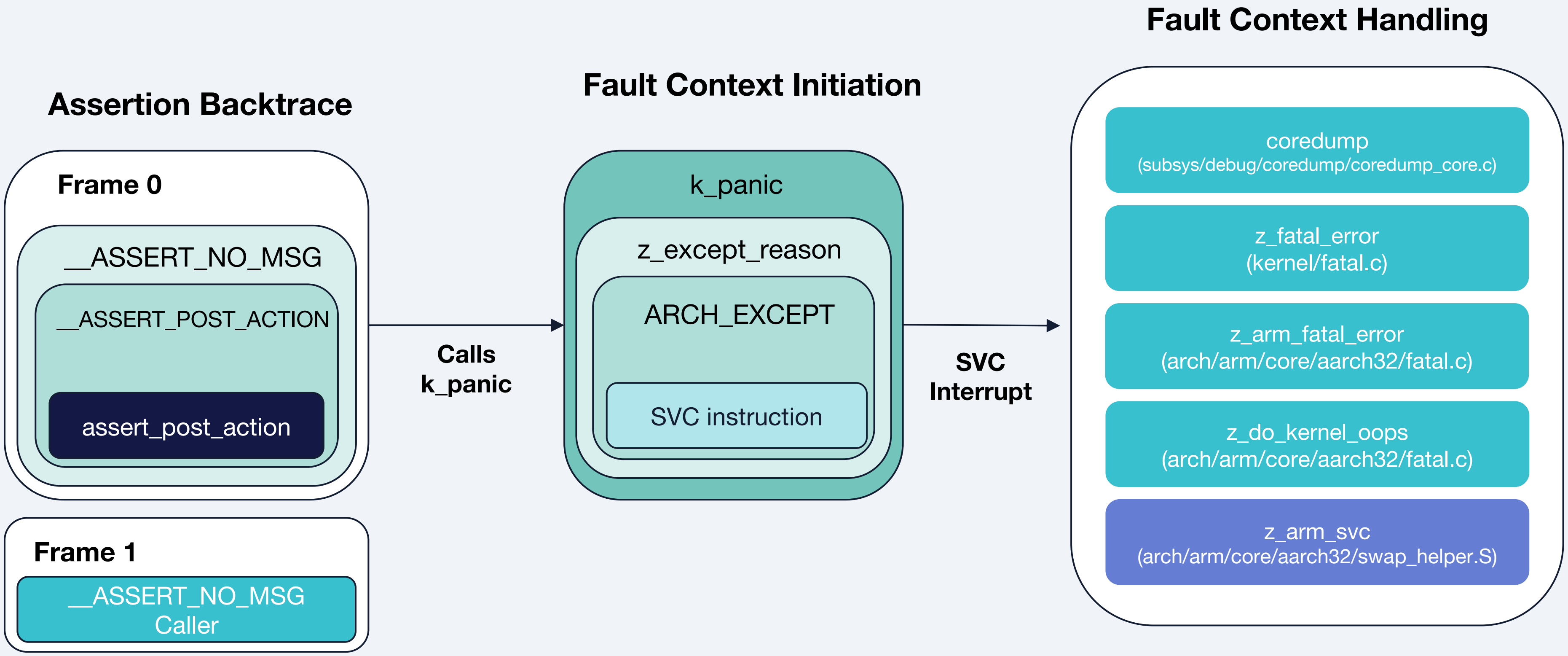
What causes a coredump?

Faults & Assertions!

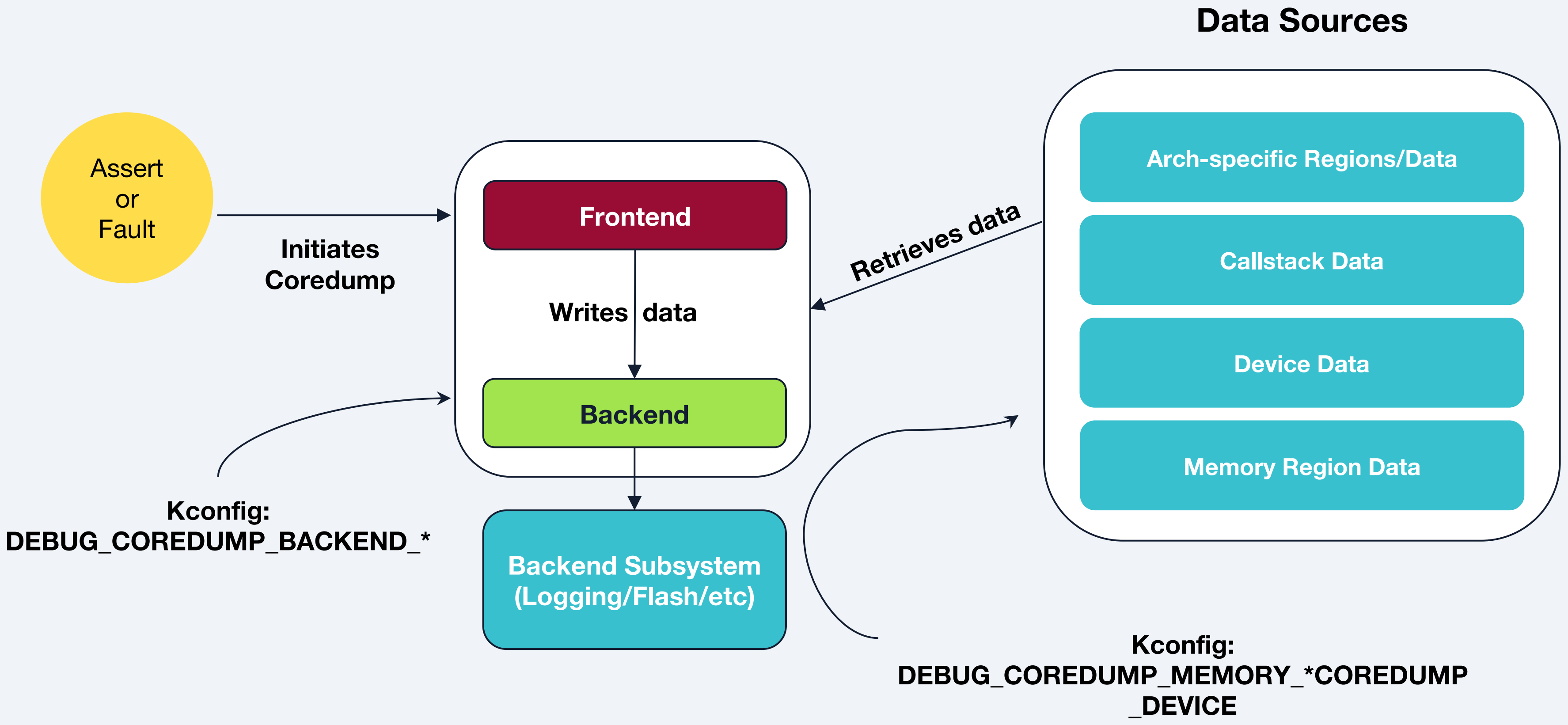
Zephyr Fault Handling Call Graph



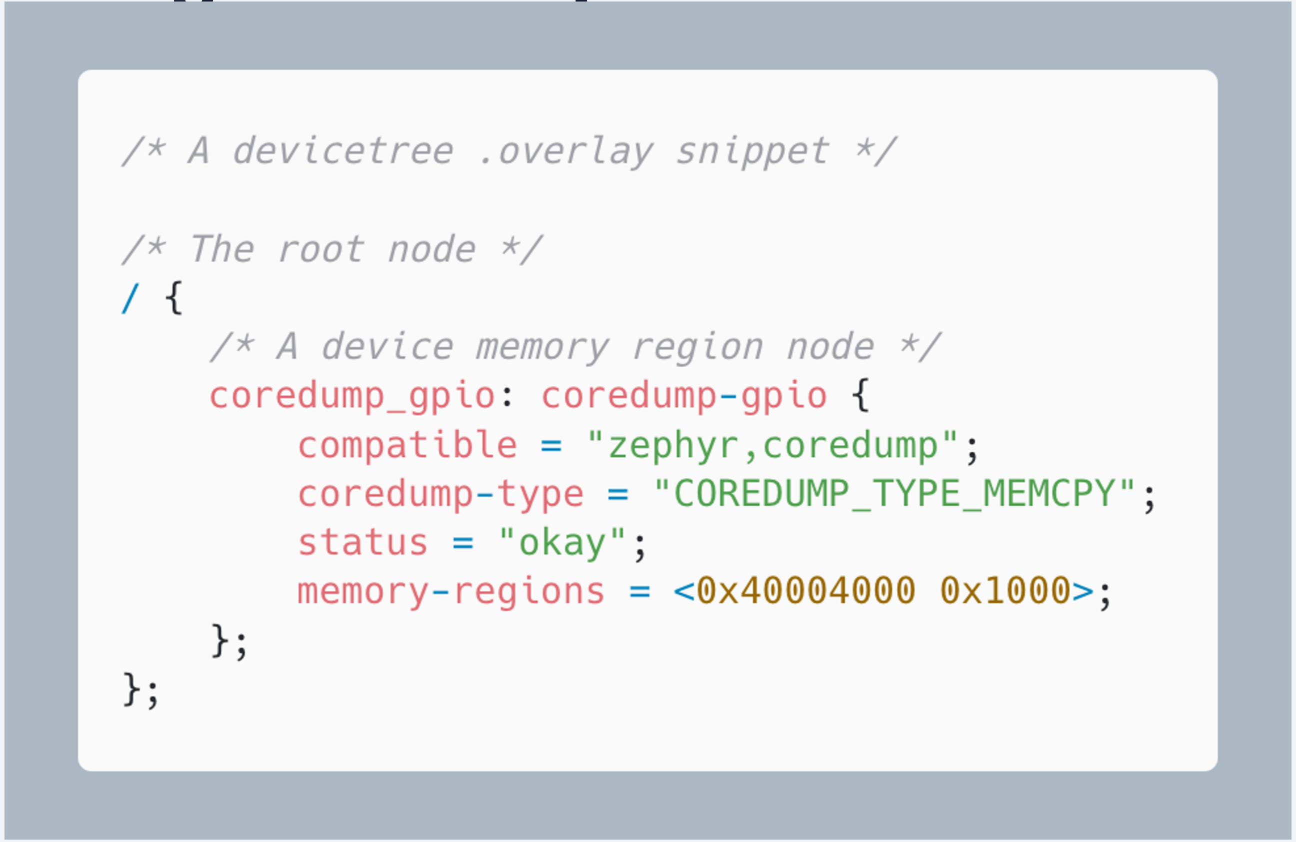
Zephyr Assertion Call Graph



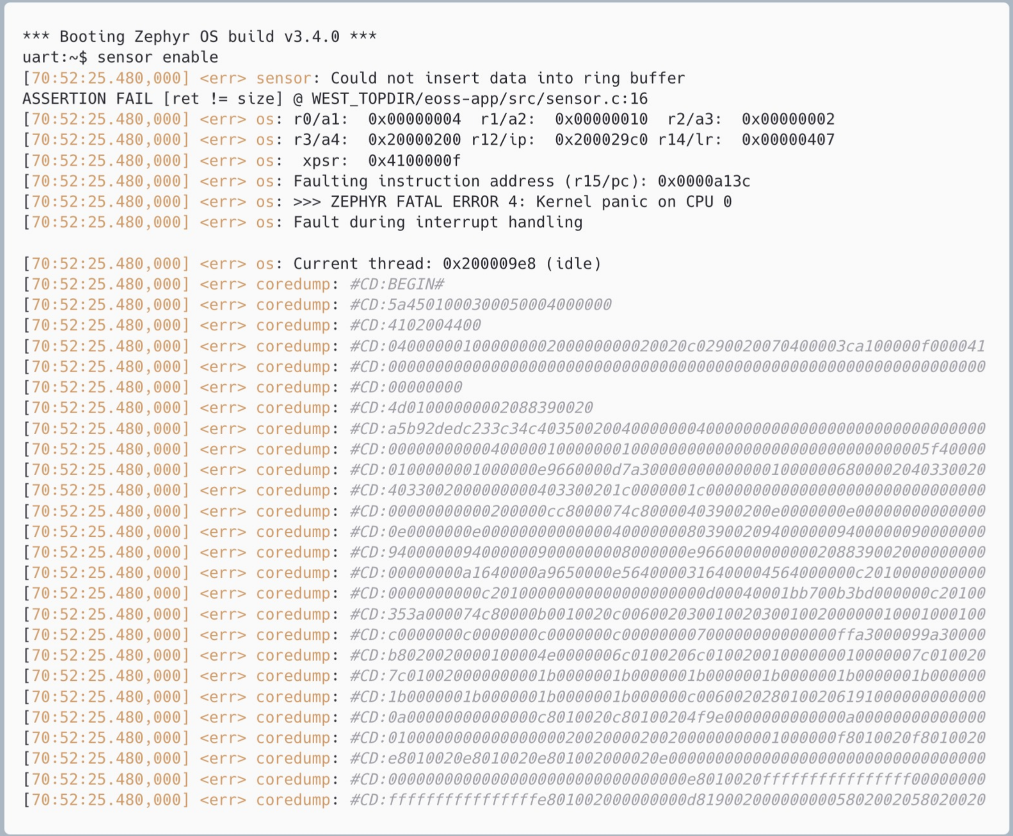
Coredump Subsystem


Data Sources
Assert or Fault
Initiates Coredump
Frontend
Writes data
Backend
Backend Subsystem (Logging/Flash/etc)
Retrieves data
Arch-specific Regions/Data
Callstack Data
Device Data
Memory Region Data
Kconfig: DEBUG_COREDUMP_BACKEND_*
Kconfig: DEBUG_COREDUMP_MEMORY_*COREDUMP_DEVICE

Device Region Example

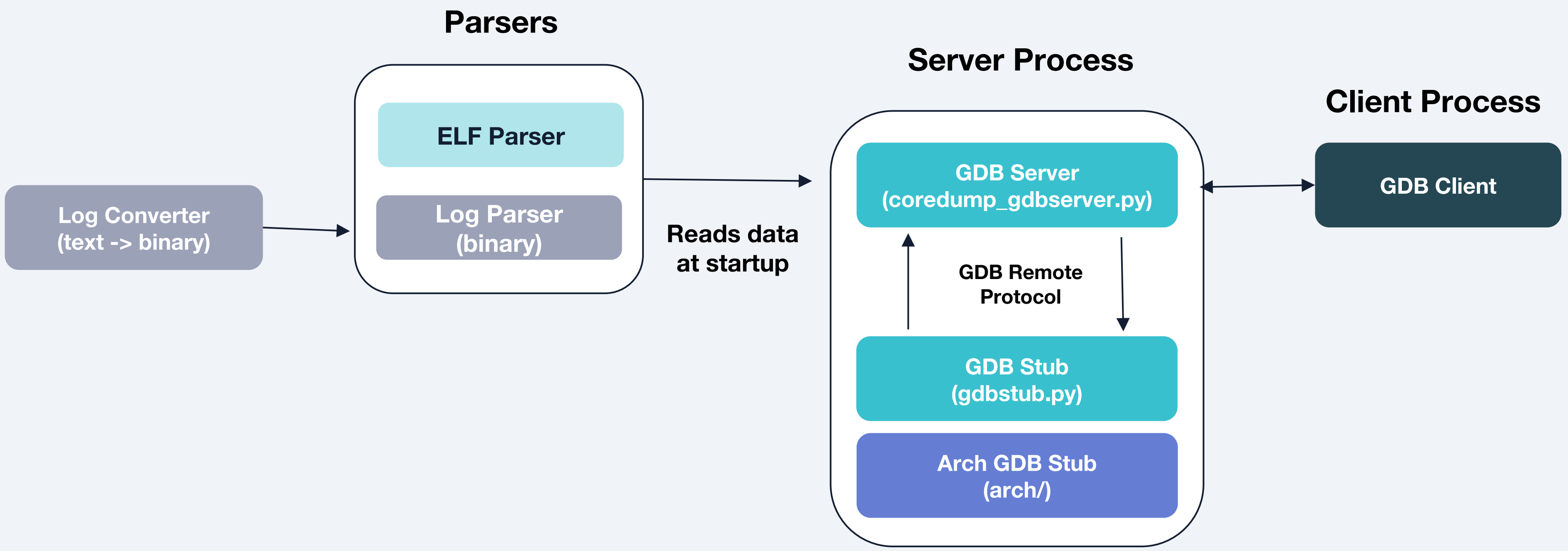


```
/* A devicetree .overlay snippet */

/* The root node */
/ {
    /* A device memory region node */
    coredump_gpio: coredump-gpio {
        compatible = "zephyr,coredump";
        coredump-type = "COREDUMP_TYPE_MEMCPY";
        status = "okay";
        memory-regions = <0x40004000 0x1000>;
    };
};
```



```
*** Booting Zephyr OS build v3.4.0 ***
uart:~$ sensor enable
[70:52:25.480,000] <err> sensor: Could not insert data into ring buffer
ASSERTION FAIL [ret != size] @ WEST_TOPDIR/eoss-app/src/sensor.c:16
[70:52:25.480,000] <err> os: r0/a1:  0x00000004  r1/a2:  0x00000010  r2/a3:  0x00000002
[70:52:25.480,000] <err> os: r3/a4:  0x20000200 r12/ip:  0x200029c0 r14/lr:  0x00000407
[70:52:25.480,000] <err> os:  xpsr:  0x4100000f
[70:52:25.480,000] <err> os: Faulting instruction address (r15/pc): 0x0000a13c
[70:52:25.480,000] <err> os: >>> ZEPHYR FATAL ERROR 4: Kernel panic on CPU 0
[70:52:25.480,000] <err> os: Fault during interrupt handling

[70:52:25.480,000] <err> os: Current thread: 0x200009e8 (idle)
[70:52:25.480,000] <err> coredump: #CD:BEGIN#
[70:52:25.480,000] <err> coredump: #CD:5a4501000300050004000000
[70:52:25.480,000] <err> coredump: #CD:4102004400
[70:52:25.480,000] <err> coredump: #CD:040000001000000002000000000020020c02900200704000003ca100000f000041
[70:52:25.480,000] <err> coredump: #CD:0000000000000000000000000000000000000000000000000000000000000000
[70:52:25.480,000] <err> coredump: #CD:00000000
[70:52:25.480,000] <err> coredump: #CD:4d010000000002088390020
[70:52:25.480,000] <err> coredump: #CD:a5b92dedc233c34c4035002004000000004000000000000000000000000000000
[70:52:25.480,000] <err> coredump: #CD:0000000000040000010000000100000000000000000000000000000000005f40000
[70:52:25.480,000] <err> coredump: #CD:0100000001000000e9660000d7a30000000000000001000000680000204033002 0
[70:52:25.480,000] <err> coredump: #CD:40330020000000000403300201c0000001c00000000000000000000000000000000
[70:52:25.480,000] <err> coredump: #CD:000000000000200000cc8000074c80000403900200e0000000e000000000000000
[70:52:25.480,000] <err> coredump: #CD:0e0000000e00000000000000004000000008039002094000000094000000090000000
[70:52:25.480,000] <err> coredump: #CD:940000009400000090000000008000000e96600000000000020883900200000000
[70:52:25.480,000] <err> coredump: #CD:00000000a1640000a9650000e5640000316400004564000000c2010000000000
[70:52:25.480,000] <err> coredump: #CD:0000000000c20100000000000000000000000d00040001bb700b3bd000000c20100
[70:52:25.480,000] <err> coredump: #CD:353a000074c80000b0010020c006002030010020300100200000000100010000100
[70:52:25.480,000] <err> coredump: #CD:c0000000c0000000c0000000c000000000700000000000000000ffa3000099a30000
[70:52:25.480,000] <err> coredump: #CD:b8020020000100004e0000006c0100206c01002001000000001000000 7c010020
[70:52:25.480,000] <err> coredump: #CD:7c01002000000001b0000001b0000001b0000001b0000001b0000001b000000
[70:52:25.480,000] <err> coredump: #CD:1b0000001b0000001b0000001b000000c00600202801002061910000000000000
[70:52:25.480,000] <err> coredump: #CD:0a000000000000000c8010020c80100204f9e0000000000000000a000000000000000
[70:52:25.480,000] <err> coredump: #CD:010000000000000000020020000200200000000000001000000f8010020f8010020
[70:52:25.480,000] <err> coredump: #CD:e8010020e8010020e801002000020e000000000000000000000000000000000000
[70:52:25.480,000] <err> coredump: #CD:00000000000000000000000000000000e8010020ffffffffffffffff00000000
[70:52:25.480,000] <err> coredump: #CD:ffffffffffffffffe801002000000000d819002000000000580200205802002 0
```

Coredump Host Tools



Scripting Extensions

- GDB can be built with Python extension support
  - Zephyr toolchain defaults to no-py version
  - Use Python with “-py”
- Requires matching system Python install
- Use venv + gdbundle to manage packages

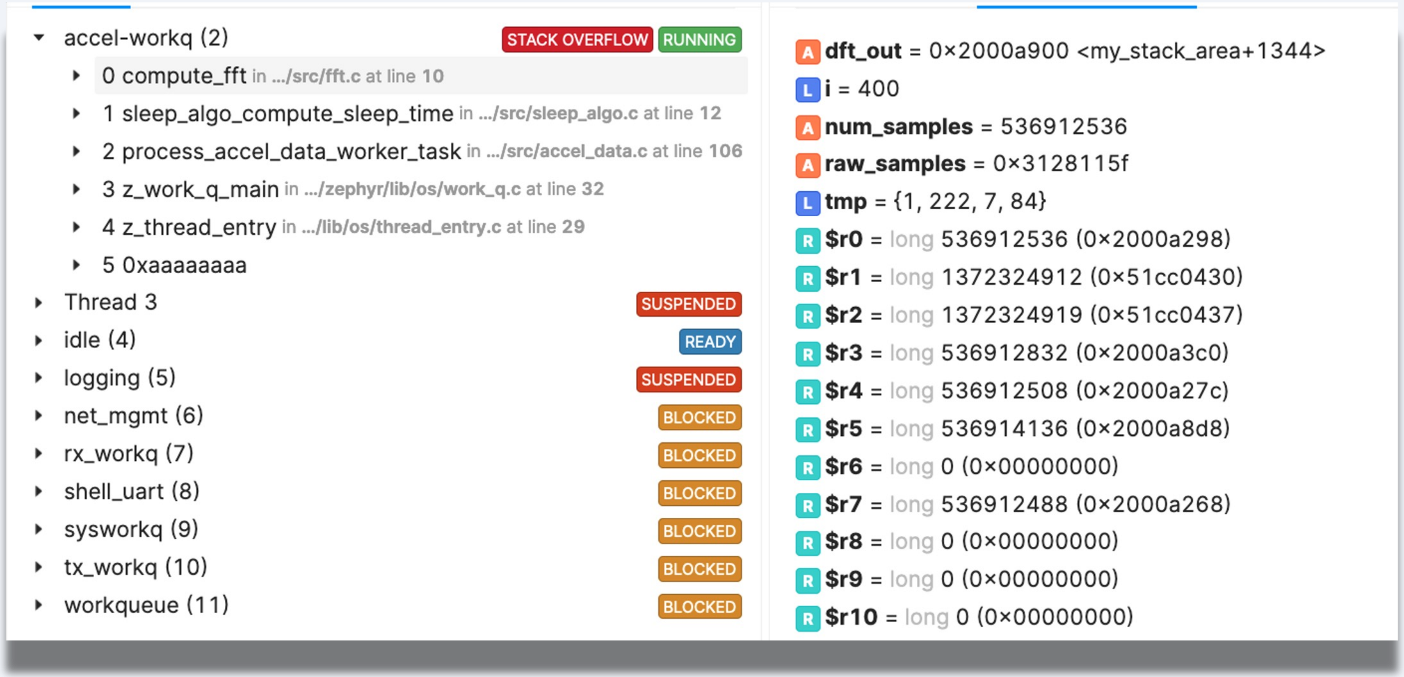
Agenda

1. Coredumps Overview
2. Coredumps with Zephyr
3. Zephyr Coredump Demo
4. Coredumps with Memfault
5. Memfault Coredump Demo

Agenda

1. Coredumps Overview
2. Coredumps with Zephyr
3. Zephyr Coredump Demo
4. Coredumps with Memfault
5. Memfault Coredump Demo

Coredumps with Memfault

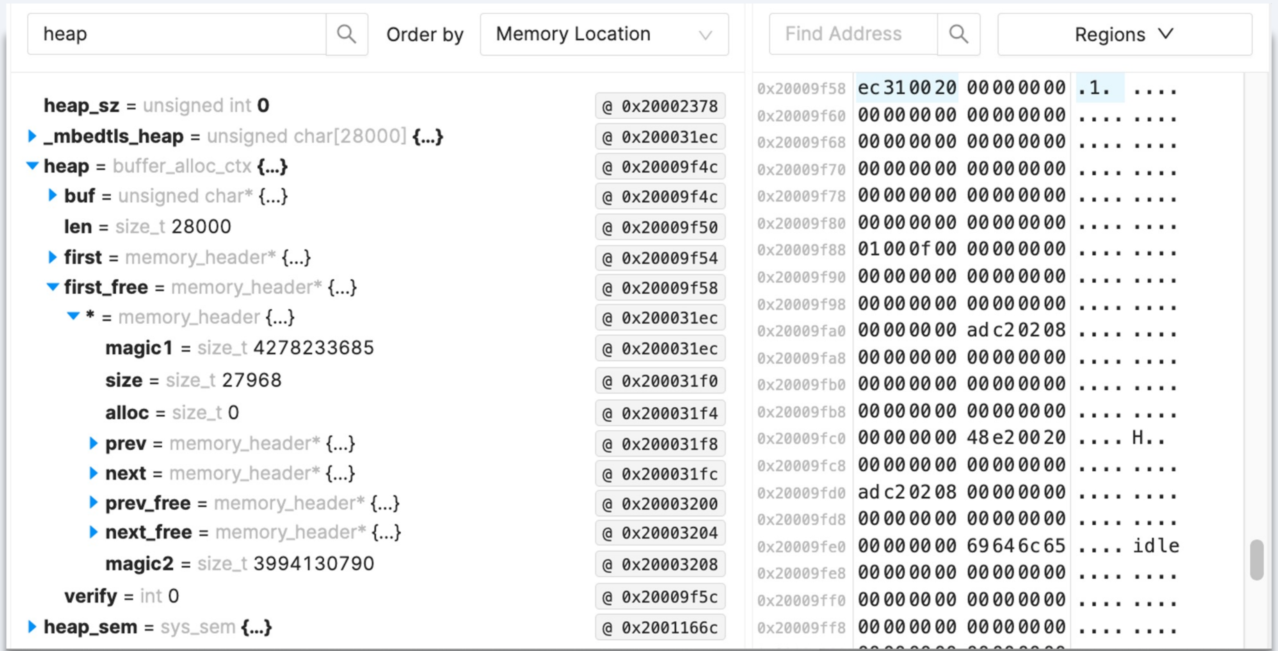


```
▾ accel-workq (2)                                                      STACK OVERFLOW  RUNNING
    ▸ 0 compute_fft in .../src/fft.c at line 10
    ▸ 1 sleep_algo_compute_sleep_time in .../src/sleep_algo.c at line 12
    ▸ 2 process_accel_data_worker_task in .../src/accel_data.c at line 106
    ▸ 3 z_work_q_main in .../zephyr/lib/os/work_q.c at line 32
    ▸ 4 z_thread_entry in .../lib/os/thread_entry.c at line 29
    ▸ 5 0xaaaaaaaa
▸ Thread 3                                                             SUSPENDED
▸ idle (4)                                                             READY
▸ logging (5)                                                          SUSPENDED
▸ net_mgmt (6)                                                         BLOCKED
▸ rx_workq (7)                                                         BLOCKED
▸ shell_uart (8)                                                       BLOCKED
▸ sysworkq (9)                                                         BLOCKED
▸ tx_workq (10)                                                        BLOCKED
▸ workqueue (11)                                                       BLOCKED
```

```
A dft_out = 0×2000a900 <my_stack_area+1344>
L i = 400
A num_samples = 536912536
A raw_samples = 0×3128115f
L tmp = {1, 222, 7, 84}
R $r0 = long 536912536 (0×2000a298)
R $r1 = long 1372324912 (0×51cc0430)
R $r2 = long 1372324919 (0×51cc0437)
R $r3 = long 536912832 (0×2000a3c0)
R $r4 = long 536912508 (0×2000a27c)
R $r5 = long 536914136 (0×2000a8d8)
R $r6 = long 0 (0×00000000)
R $r7 = long 536912488 (0×2000a268)
R $r8 = long 0 (0×00000000)
R $r9 = long 0 (0×00000000)
R $r10 = long 0 (0×00000000)
```

View tasks, stack traces, registers, and local variables

Memory Regions in Memfault

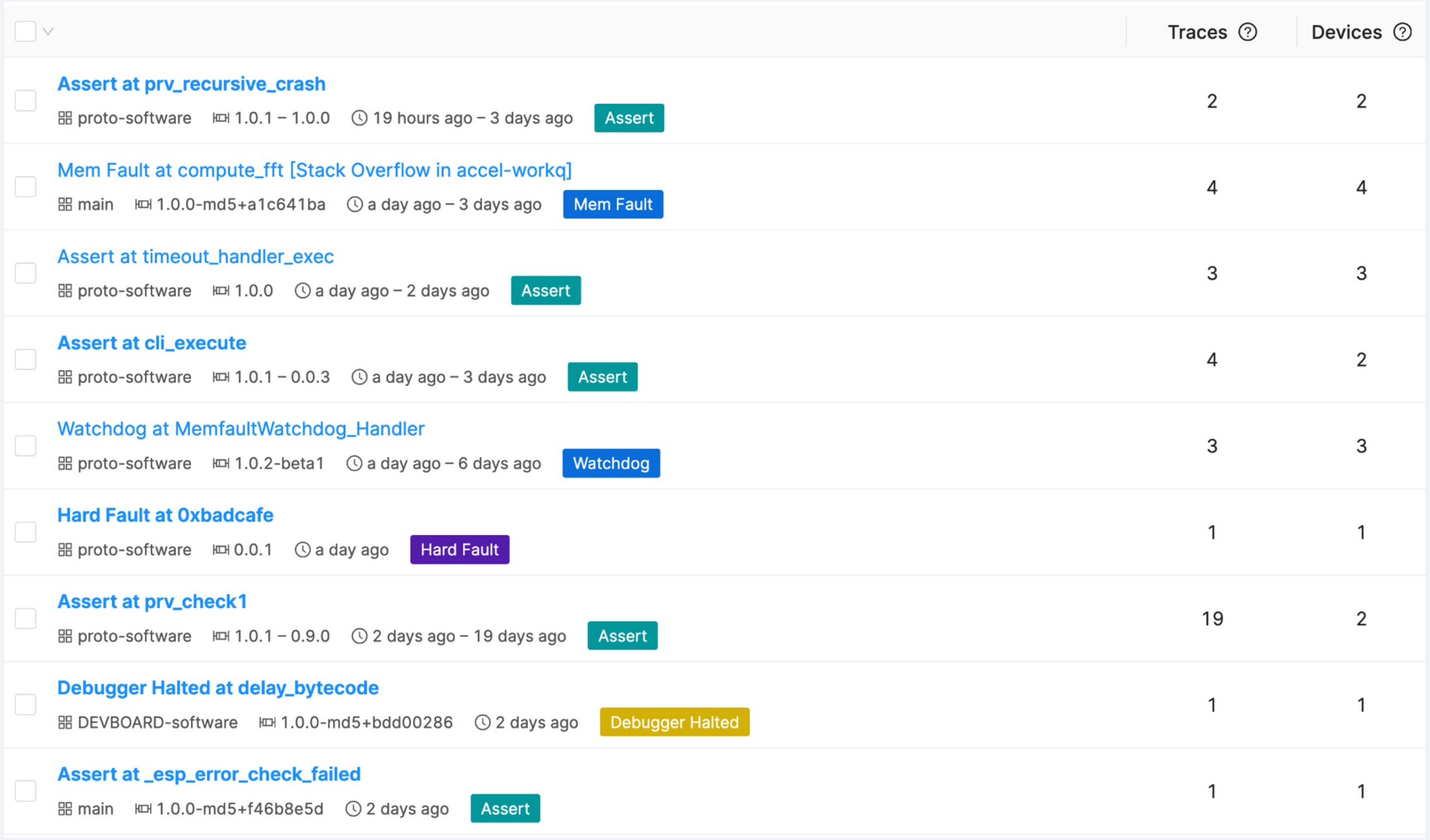


```
heap_sz = unsigned int 0                                @ 0x20002378
▸ _mbedtls_heap = unsigned char[28000] {...}            @ 0x200031ec
▾ heap = buffer_alloc_ctx {...}                         @ 0x20009f4c
  ▸ buf = unsigned char* {...}                          @ 0x20009f4c
    len = size_t 28000                                  @ 0x20009f50
  ▸ first = memory_header* {...}                        @ 0x20009f54
  ▾ first_free = memory_header* {...}                   @ 0x20009f58
    ▾ * = memory_header {...}                           @ 0x200031ec
        magic1 = size_t 4278233685                      @ 0x200031ec
        size = size_t 27968                             @ 0x200031f0
        alloc = size_t 0                                @ 0x200031f4
      ▸ prev = memory_header* {...}                     @ 0x200031f8
      ▸ next = memory_header* {...}                     @ 0x200031fc
      ▸ prev_free = memory_header* {...}                @ 0x20003200
      ▸ next_free = memory_header* {...}                @ 0x20003204
        magic2 = size_t 3994130790                      @ 0x20003208
    verify = int 0                                      @ 0x20009f5c
▸ heap_sem = sys_sem {...}                              @ 0x2001166c
```

```
0x20009f58  ec310020 00000000  .1.  ....
0x20009f60  00000000 00000000  .... ....
0x20009f68  00000000 00000000  .... ....
0x20009f70  00000000 00000000  .... ....
0x20009f78  00000000 00000000  .... ....
0x20009f80  00000000 00000000  .... ....
0x20009f88  01000f00 00000000  .... ....
0x20009f90  00000000 00000000  .... ....
0x20009f98  00000000 00000000  .... ....
0x20009fa0  00000000 adc20208  .... ....
0x20009fa8  00000000 00000000  .... ....
0x20009fb0  00000000 00000000  .... ....
0x20009fb8  00000000 00000000  .... ....
0x20009fc0  00000000 48e20020  .... H..
0x20009fc8  00000000 00000000  .... ....
0x20009fd0  adc20208 00000000  .... ....
0x20009fd8  00000000 00000000  .... ....
0x20009fe0  00000000 69646c65  .... idle
0x20009fe8  00000000 00000000  .... ....
0x20009ff0  00000000 00000000  .... ....
0x20009ff8  00000000 00000000  .... ....
```

View all global variables at time of coredump

Issues



	Traces	Devices
Assert at prv_recursive_crash proto-software · 1.0.1 – 1.0.0 · 19 hours ago – 3 days ago · Assert	2	2
Mem Fault at compute_fft [Stack Overflow in accel-workq] main · 1.0.0-md5+a1c641ba · a day ago – 3 days ago · Mem Fault	4	4
Assert at timeout_handler_exec proto-software · 1.0.0 · a day ago – 2 days ago · Assert	3	3
Assert at cli_execute proto-software · 1.0.1 – 0.0.3 · a day ago – 3 days ago · Assert	4	2
Watchdog at MemfaultWatchdog_Handler proto-software · 1.0.2-beta1 · a day ago – 6 days ago · Watchdog	3	3
Hard Fault at 0xbadcafe proto-software · 0.0.1 · a day ago · Hard Fault	1	1
Assert at prv_check1 proto-software · 1.0.1 – 0.9.0 · 2 days ago – 19 days ago · Assert	19	2
Debugger Halted at delay_bytecode DEVBOARD-software · 1.0.0-md5+bdd00286 · 2 days ago · Debugger Halted	1	1
Assert at _esp_error_check_failed main · 1.0.0-md5+f46b8e5d · 2 days ago · Assert	1	1

Memfault Coredump Features

- Complete RTOS analysis
- Per-thread call stack unwinds
- Compatibility across RTOSes
- Automatic coredump and symbol file association
- Coredump classification, deduplication, and aggregation

Agenda

1. Coredumps Overview
2. Coredumps with Zephyr
3. Zephyr Coredump Demo
4. Coredumps with Memfault
5. Memfault Coredump Demo

Summary

- Coredumps provide advance post-mortem fault and assertion analysis
- Use coredumps early and often
    - Add coredump support to your builds, earlier the better
    - Use coredumps often to identify and thoroughly investigate crashes post-mortem

Thank You!

- Find me at:
  - linkedin.com/in/ejohnso49/
  - Github: ejohnso49
- Read our posts on interrupt.memfault.com
- We’re hiring!

Memfault

Firmware Solutions Engineer, Memfault