# Femto-Containers

## Lightweight DevOps-style Virtual Machines on RIOT

Koen Zandberg

TRiBE
inTeRnet BEyond the usual

RIOT Summit 2021

# Overview

- How to maintain your deployment
- Virtual Machine Solutions
- The Linux solution: eBPF
- Femto-Containers
- Example: thread counter
- Limitations and Conclusion

# How to maintain your deployment?

- Deploying IoT nodes at scale challenging.

- How about maintaining them in the field?

# Current Issues

One of the devices in the field shows odd behaviour, can we debug this?

A customer needs modified behaviour on the deployed nodes.

A third party wants to run code on our devices.

# Categories of Solutions

- Traditional solution: firmware updates
  - Simple, but has downsides
  - Maintaining and deploying another firmware version is costly
- Alternative solutions? Modular updates
  - Dynamic linking
  - Virtual Machines
  - ...

# Overview

- How to maintain your deployment
- Virtual Machine Solutions
- The Linux solution: eBPF
- Femto-Containers
- Example: thread counter
- Limitations and Conclusion

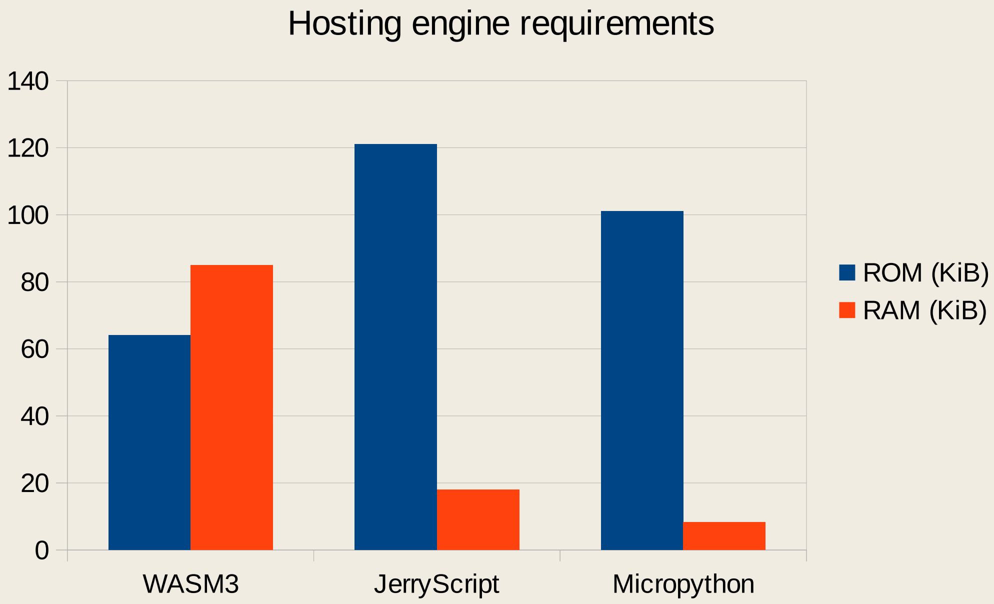
# Virtual Machine Solutions

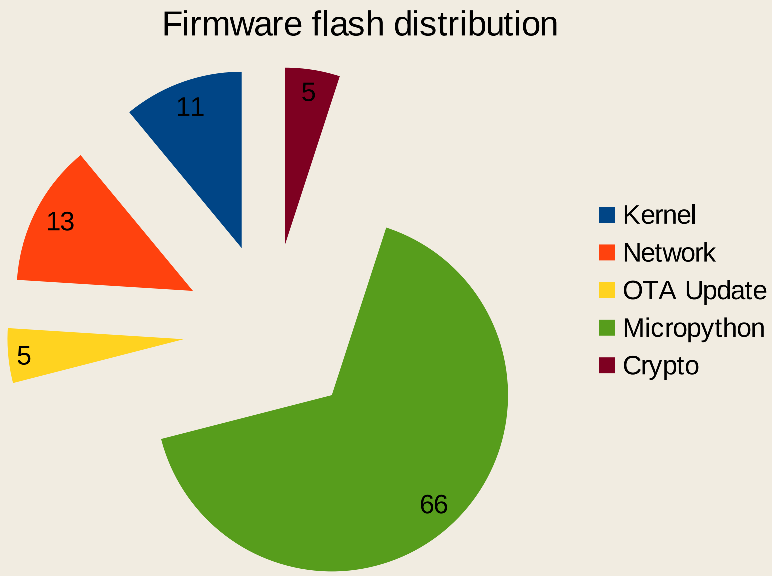
- Python
- Javascript
- WebAssembly
- MicroEJ

- (And others)

# Virtual Machine Solutions

Downsides: bulky to add for simple applications [1]



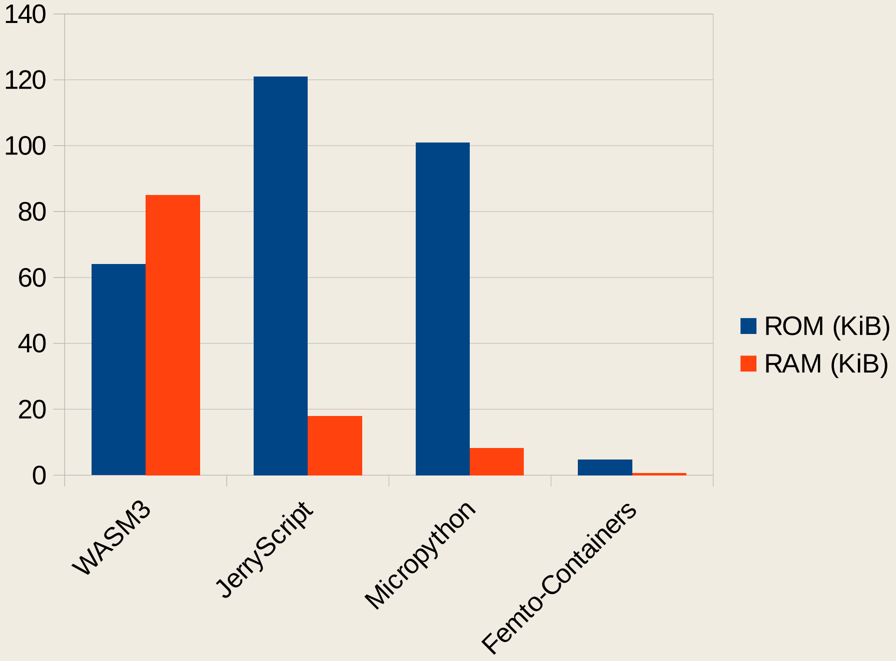


Measured on the nRF52840dk, Hosting engine only (RIOT 2021.4 release)

[1] K. Zandberg, E. Baccelli. *Femto-Containers: DevOps on Microcontrollers with Lightweight Virtualization & Isolation for IoT Software Modules*. ArXiv, June 2021.

# Sneak Peek

- Femto-containers:
  - Much smaller VMs!
  - Based on eBPF
  - Hosting engine:
    - 4.7 KiB ROM
    - 664 B of RAM



Measured [1] on the nRF52840dk (RIOT 2021.4 release)

[1] K. Zandberg, E. Baccelli. *Femto-Containers: DevOps on Microcontrollers with Lightweight Virtualization & Isolation for IoT Software Modules*. ArXiv, June 2021.

# Overview

- How to maintain your deployment
- Virtual Machine Solutions
- The Linux solution: eBPF
- Femto-Containers
- Example: thread counter
- Limitations and Conclusion

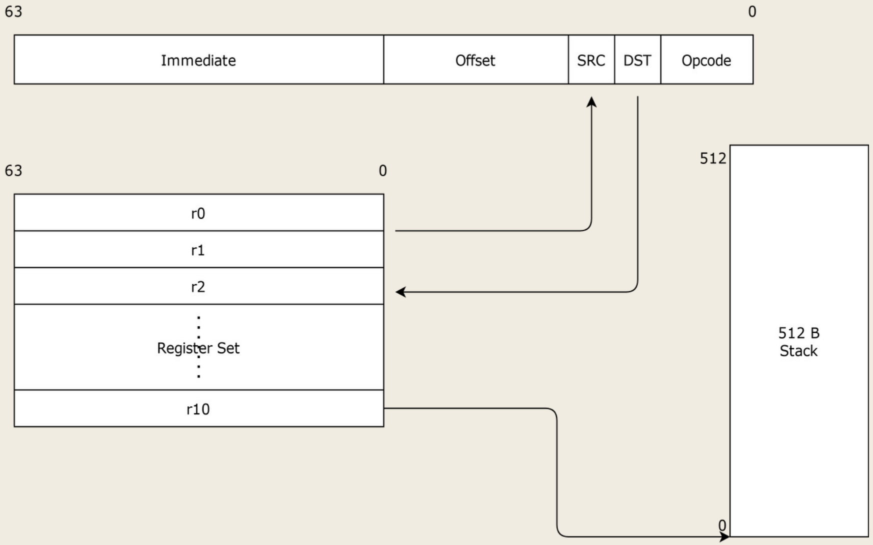
# The Linux solution: eBPF

Event-driven in-kernel sandbox:

- Tracing
- Profiling
- Monitoring
- Network Protocol parsers

# The Linux solution: eBPF

- in-kernel Virtual Machine:
  - 64 bit RISC architecture
  - Register based
  - 512 byte stack
- Allows for verification of loaded applications:
  - Application must halt



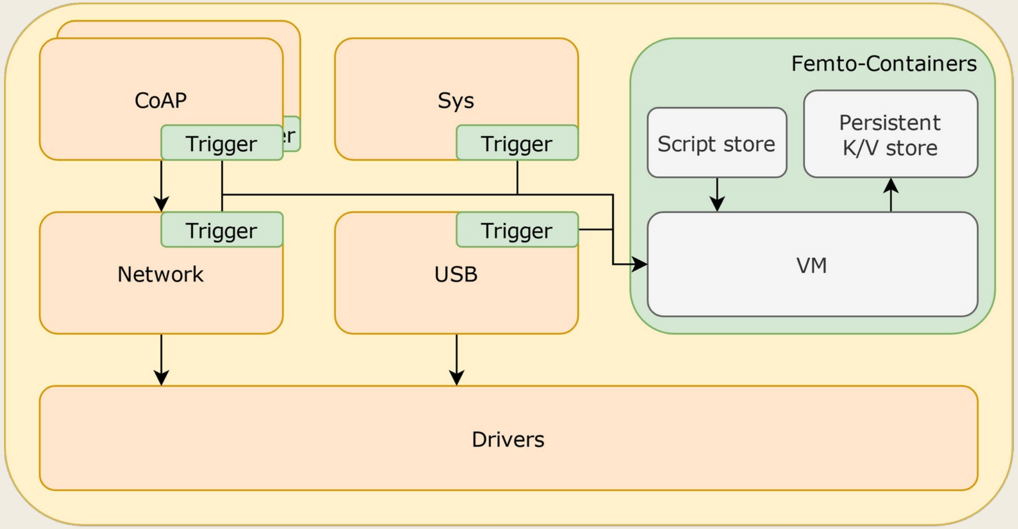
# Overview

- How to maintain your deployment
- Virtual Machine Solutions
- The Linux solution: eBPF
- Femto-Containers
- Example: thread counter
- Limitations and Conclusion

# Femto-Containers

- Simple virtual machine
- Hardware independent
- Short-lived, Event driven

- Integration with RIOT
- Based on Linux eBPF
- Minimal footprint



# Femto-Containers

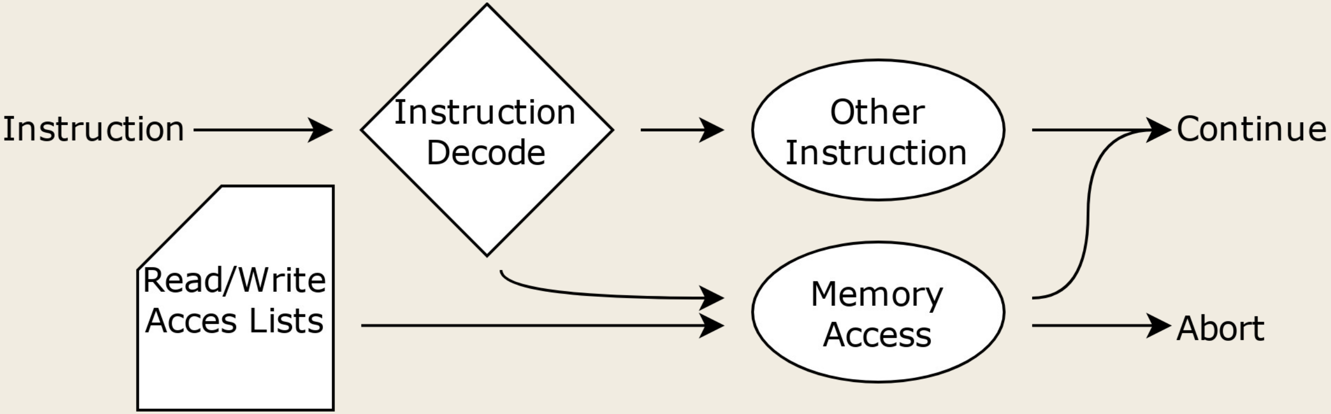
## Why eBPF?

- ✔ 512 Byte stack
- ✔ Limited instruction set
- ✔ Secure by design

- ✘ 64 bit architecture

# Femto-Containers: Isolation

- Sandboxed from the host
  - Pre-flight checks
  - Memory access guards

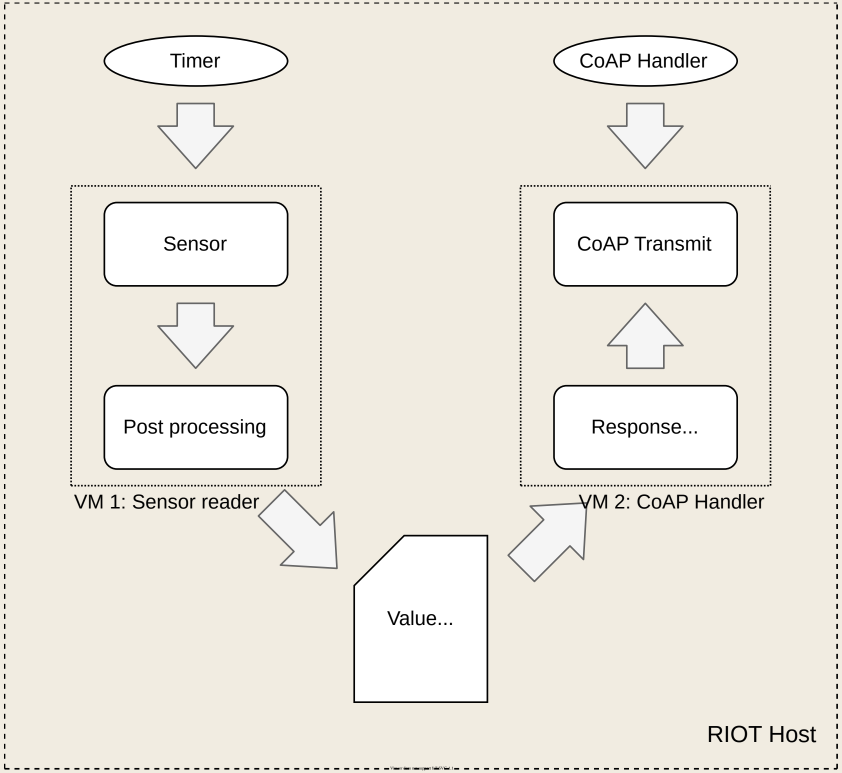


# Femto-Containers: Events

Event triggered:

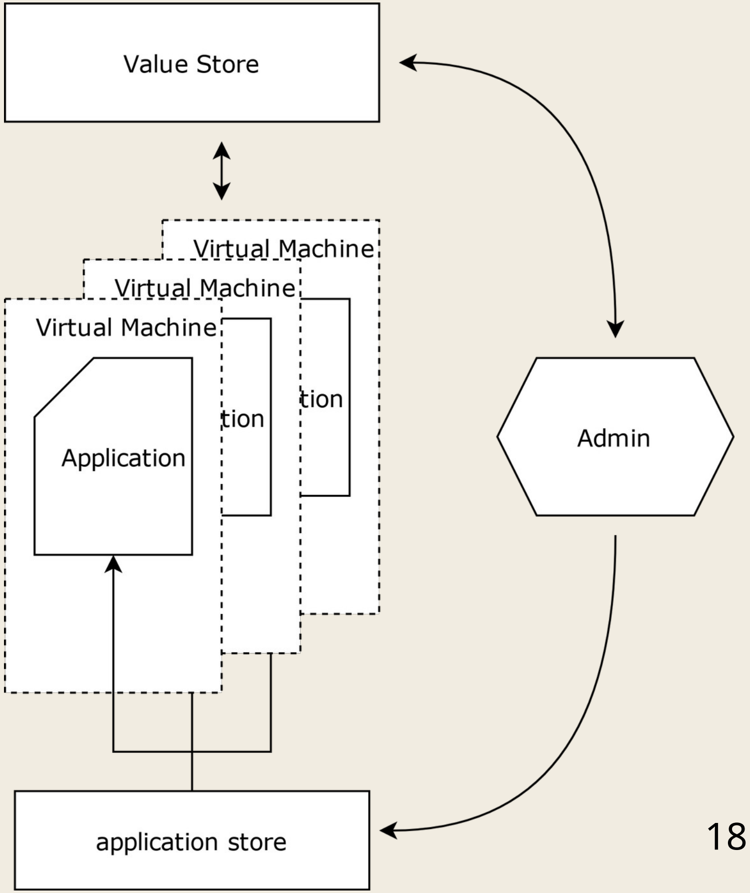
- Network
- USB
- System events
- Timers

Adding hooks is cheap



# Femto-Containers: OS Interaction

- Context and return value
  - Packet and Allow/Reject
- Bindings
  - Calls to OS, e.g. saul_read
- Value store
  - Store simple values



# Femto-Containers: Caveats

- Slow down
  - Virtual machine overhead
- Instruction set limitations
  - No indirect jumps
- Security and isolation
  - Basic security measures only
  - No formal verification (yet)

| | App size | Startup time | Run time |
|---:|---:|---:|---:|
| Native C | 74 B | - | 27 µs |
| WASM3 | 322 B | 17 096 µs | 980 µs |
| Femto-Containers | 456 B | 1 µs | 2133 µs |
| JerryScript | 593 B | 5589 µs | 14 726 µs |
| MicroPython | 497 B | 21 907 µs | 16 325 µs |

Fletcher32 startup time and run time [1]

[1] K. Zandberg, E. Baccelli. *Femto-Containers: DevOps on Microcontrollers with Lightweight Virtualization & Isolation for IoT Software Modules*. ArXiv, June 2021.

# Overview

- How to maintain your deployment
- Virtual Machine Solutions
- The Linux solution: eBPF
- Femto-Containers
- Example: thread counter
- Conclusion

# Example

- Thread counter
  - Maintain thread run counters
  - Hooks into the scheduler
  - Store counters in the value store

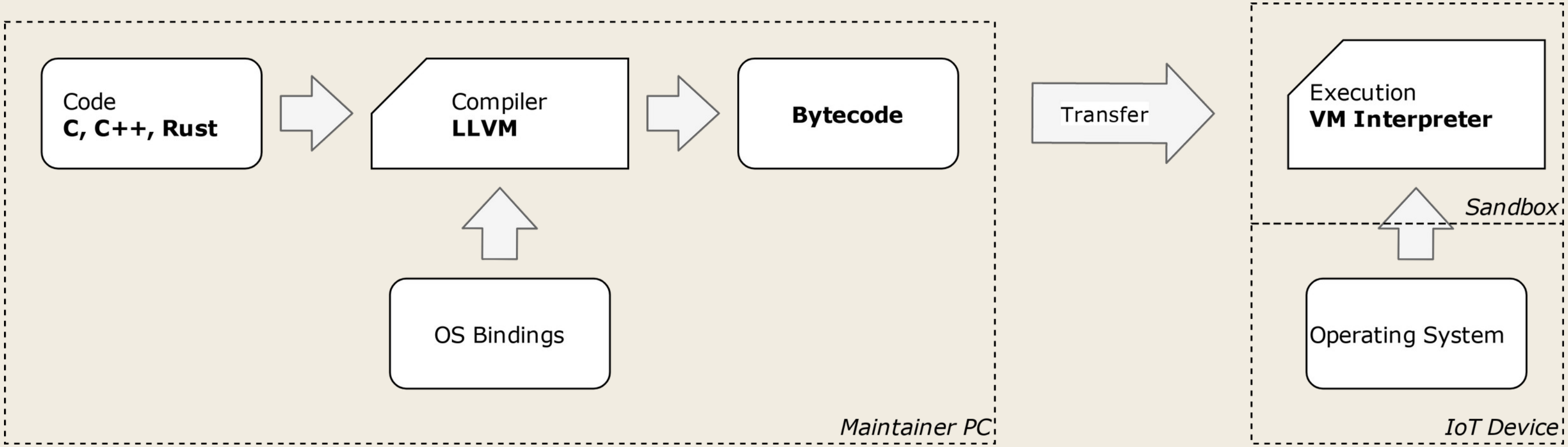
# Workflow

1) Write code

2) Compile

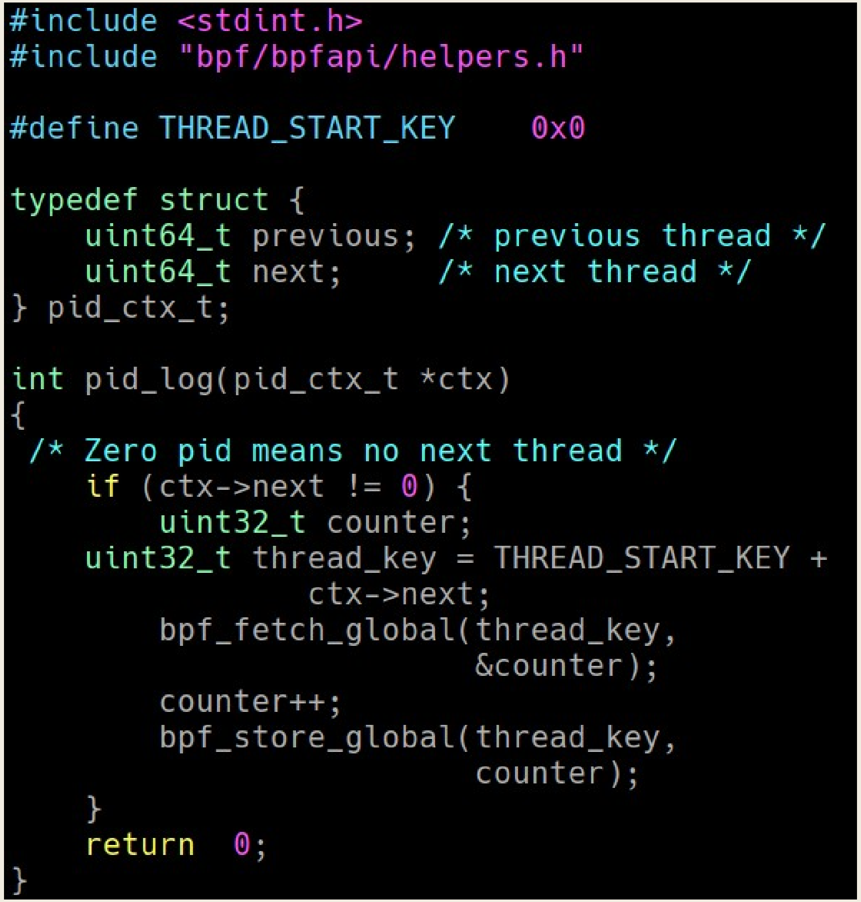
3) Transfer

4) Run



# Workflow

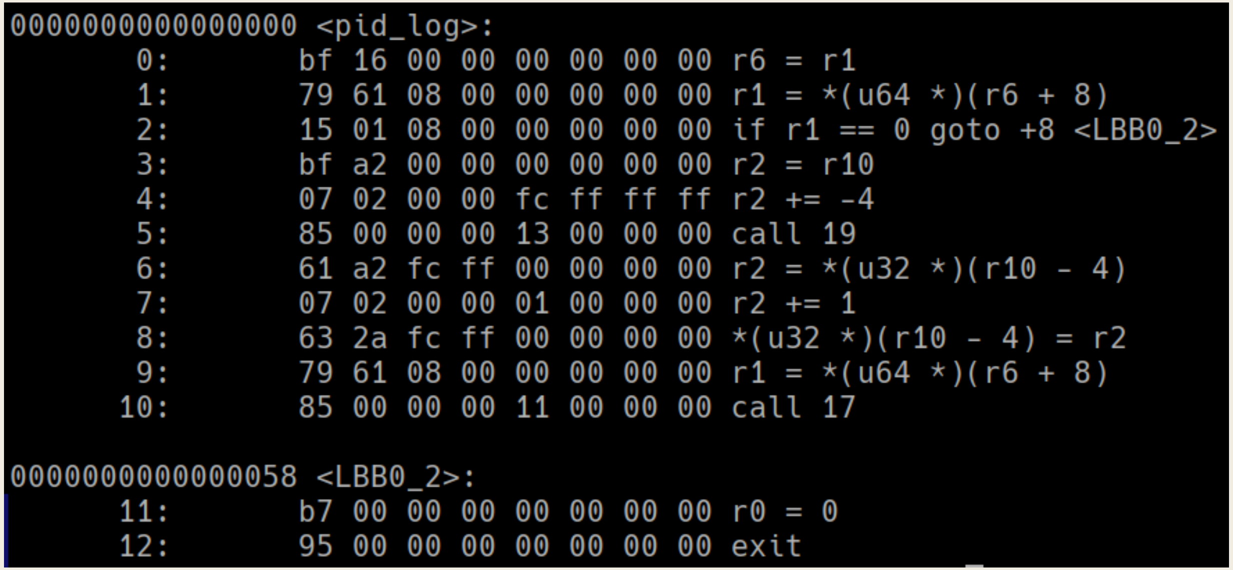
- Write our code
  - C
  - Rust
  - TinyGo?
- Store the run count for each thread



```
#include <stdint.h>
#include "bpf/bpfapi/helpers.h"

#define THREAD_START_KEY    0x0

typedef struct {
    uint64_t previous; /* previous thread */
    uint64_t next;     /* next thread */
} pid_ctx_t;

int pid_log(pid_ctx_t *ctx)
{
 /* Zero pid means no next thread */
    if (ctx->next != 0) {
        uint32_t counter;
    uint32_t thread_key = THREAD_START_KEY +
                ctx->next;
        bpf_fetch_global(thread_key,
                         &counter);
        counter++;
        bpf_store_global(thread_key,
                         counter);
    }
    return  0;
}
```

# Workflow

- Compilation with LLVM
  - eBPF support
- RIOT bindings



```
0000000000000000 <pid_log>:
       0:       bf 16 00 00 00 00 00 00 r6 = r1
       1:       79 61 08 00 00 00 00 00 r1 = *(u64 *)(r6 + 8)
       2:       15 01 08 00 00 00 00 00 if r1 == 0 goto +8 <LBB0_2>
       3:       bf a2 00 00 00 00 00 00 r2 = r10
       4:       07 02 00 00 fc ff ff ff r2 += -4
       5:       85 00 00 00 13 00 00 00 call 19
       6:       61 a2 fc ff 00 00 00 00 r2 = *(u32 *)(r10 - 4)
       7:       07 02 00 00 01 00 00 00 r2 += 1
       8:       63 2a fc ff 00 00 00 00 *(u32 *)(r10 - 4) = r2
       9:       79 61 08 00 00 00 00 00 r1 = *(u64 *)(r6 + 8)
      10:       85 00 00 00 11 00 00 00 call 17

0000000000000058 <LBB0_2>:
      11:       b7 00 00 00 00 00 00 00 r0 = 0
      12:       95 00 00 00 00 00 00 00 exit
```

# Workflow

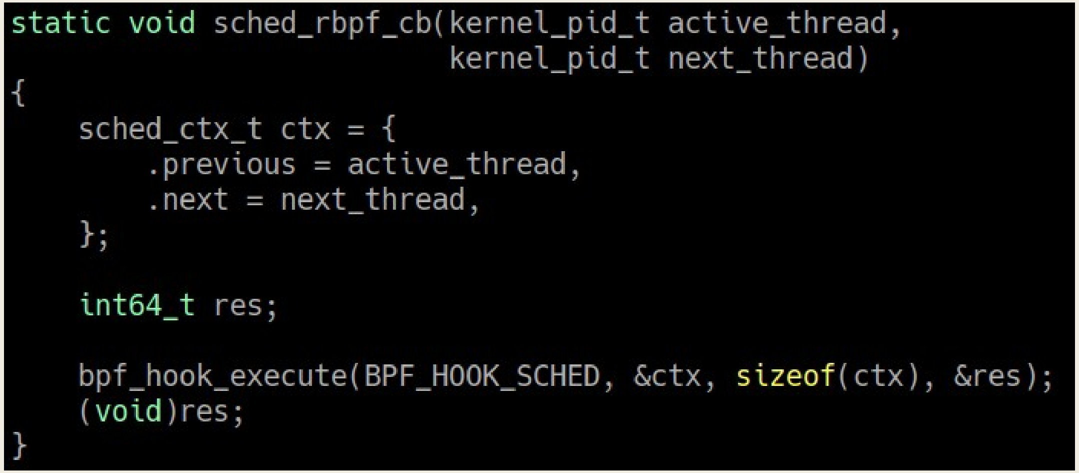
- Transfer the application:
  - CoAP
  - Bluetooth
  - Compile-in
- Independent of Femto-containers

# Workflow

Start VM from RIOT

- Our code is compiled-in for simplicity

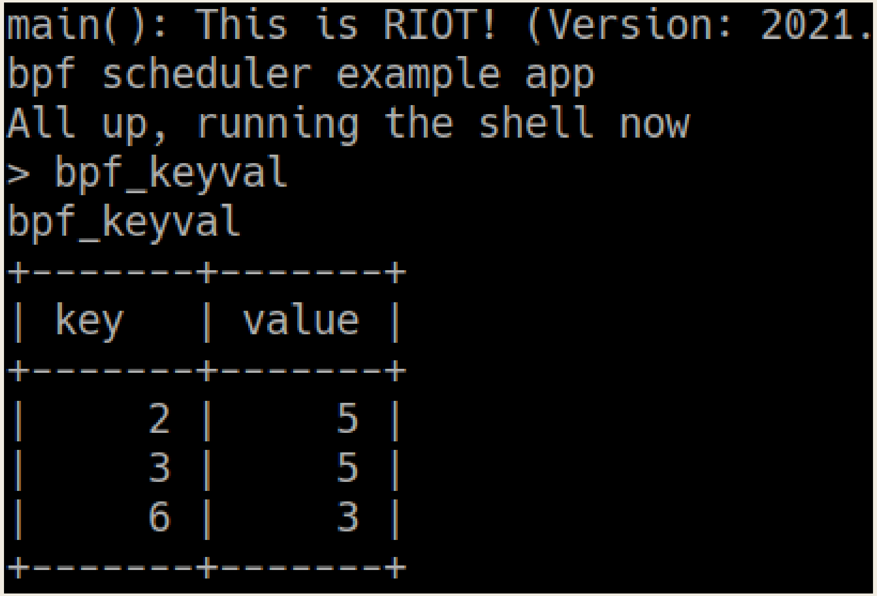
- RIOT executes the VM when switching threads



```
static void sched_rbpf_cb(kernel_pid_t active_thread,
                          kernel_pid_t next_thread)
{
    sched_ctx_t ctx = {
        .previous = active_thread,
        .next = next_thread,
    };

    int64_t res;

    bpf_hook_execute(BPF_HOOK_SCHED, &ctx, sizeof(ctx), &res);
    (void)res;
}
```

# Workflow

- Run the code
- Query the value store counters



```
main(): This is RIOT! (Version: 2021.
bpf scheduler example app
All up, running the shell now
> bpf_keyval
bpf_keyval
+--------+--------+
| key    | value  |
+--------+--------+
|      2 |      5 |
|      3 |      5 |
|      6 |      3 |
+--------+--------+
```

https://github.com/bergzand/RIOT/tree/wip/bpf/examples/rbpf_sched/

# Overview

- How to maintain your deployment
- Virtual Machine Solutions
- The Linux solution: eBPF
- Femto-Containers
- Example: thread counter
- Conclusion

# Conclusions

Rethink the cost of a VM on your RIOT device! Femto-Containers can provide:

- Customized behaviour
- Debugging
- Isolating code

With minimal impact on memory requirements.

# Femto-Containers

Want to know more?

- Example:
  https://github.com/bergzand/RIOT/tree/wip/bpf/examples/rbpf_sched
- Tutorials:
  https://github.com/future-proof-iot/Femto-Container_tutorials
- Preprint:
  K. Zandberg, E. Baccelli. *Femto-Containers: DevOps on Microcontrollers with Lightweight Virtualization & Isolation for IoT Software Modules*. ArXiv, June 2021.
  https://arxiv.org/abs/2106.12553

# Thanks!