This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2020-7322C

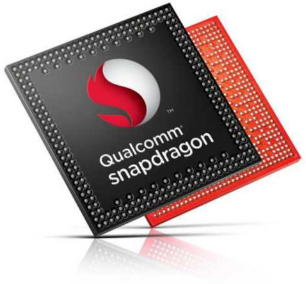
# HALucinator: Firmware Re-hosting Through Abstraction Layer Emulation

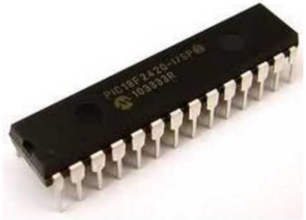
**Abraham Clements***, **Eric Gustafson***, Tobias Scharnowski, Paul Grosen, David Fritz,
Christopher Kruegel, Giovanni Vigna, Saurabh Bagchi, and Mathias Payer

*Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525, SAND2020-XXXX*

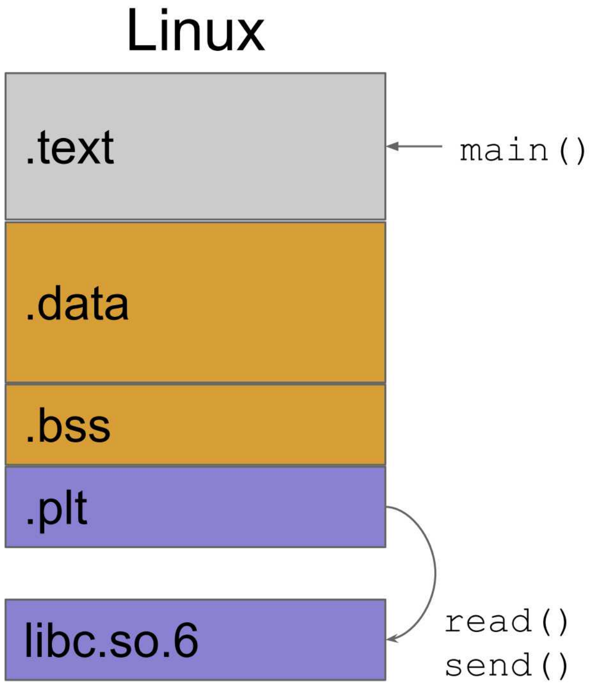
# IoT and Operational Technology

# Device Internals

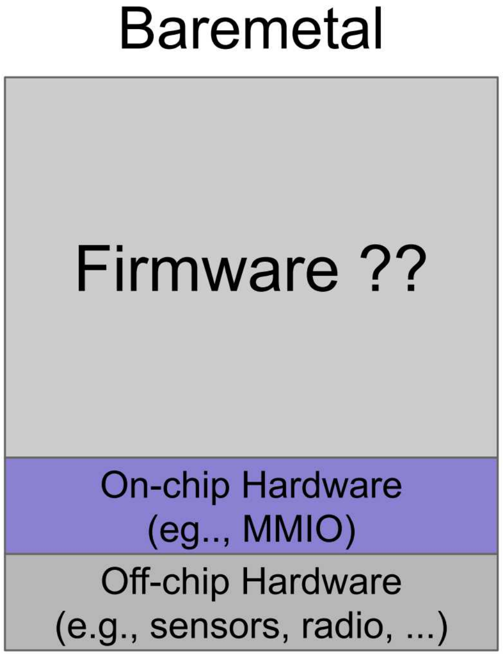




# Many run Baremetal Firmware



Kernel abstractions used for hardware interactions



Raw hardware access

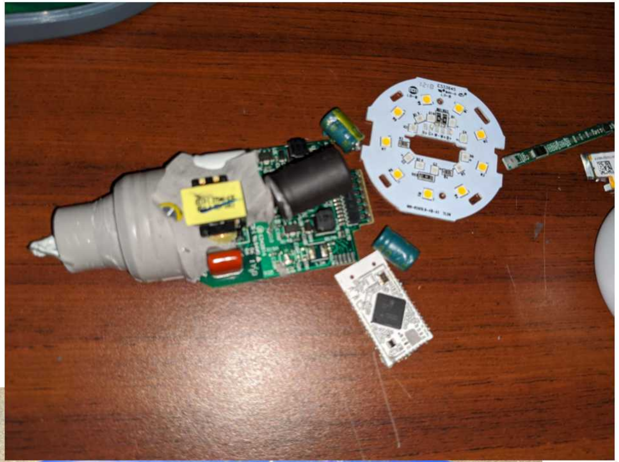
# Dynamic Analysis

## Hardware

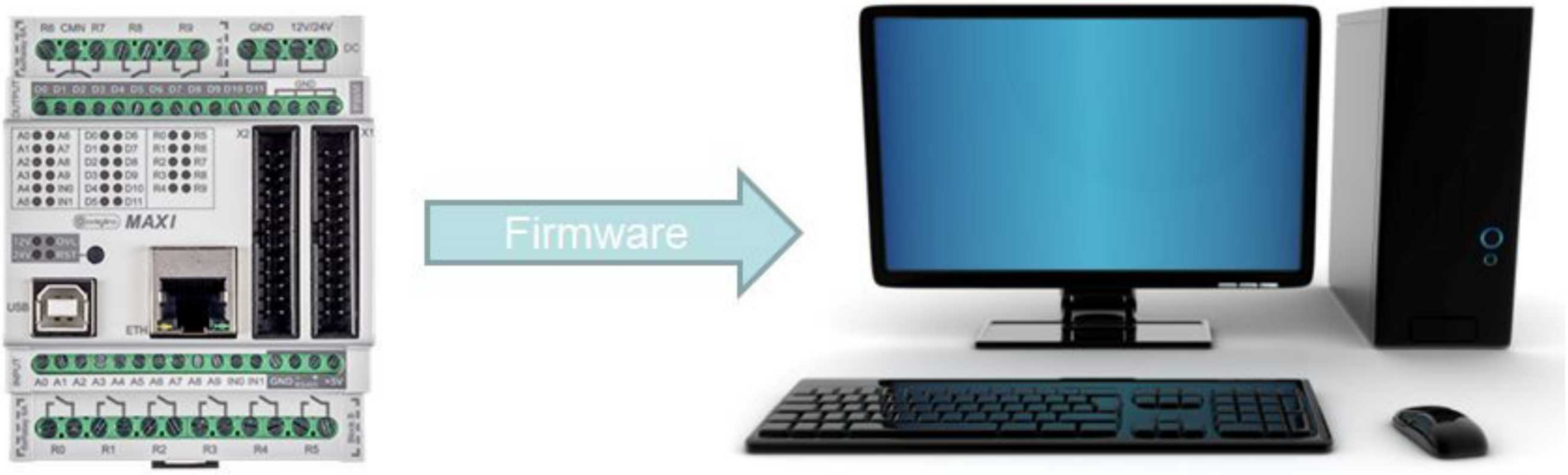
- Expensive ($10,000)
- Brittle, easily bricked
- No parallelism

## Opaque

- Debug ports should be disabled
- If present, very limited

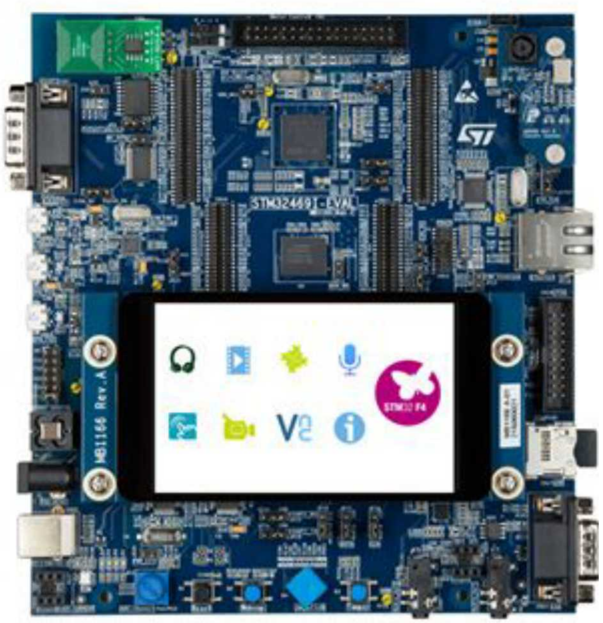


# Re-hosting to the Rescue?



**Goal: Enable firmware testing without requiring its specialized hardware**

# Re-hosting Challenges



**On chip**

**CPU**

AES Accelerator

Hash

Coprocessor

Timers

Counters

Flash Controller

Clock Config

IAP

DMA

**Off chip**

Ethernet

SD-MMC

GPIO

Camera

LCD

Touch Screen

Wireless

EEPROM

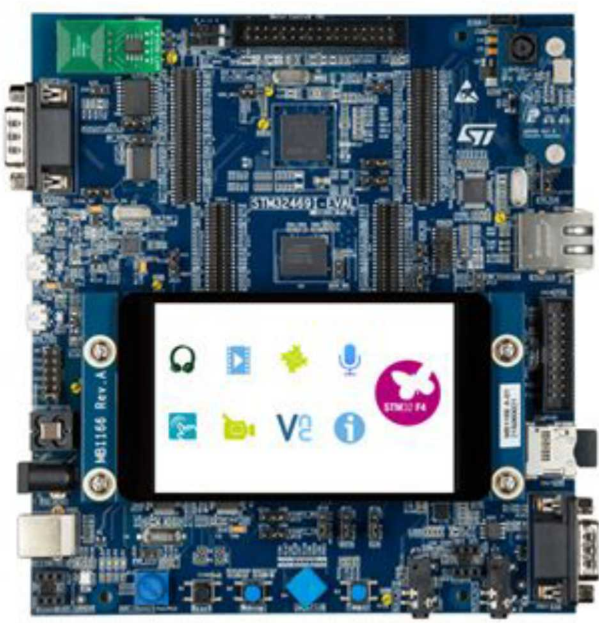
Serial

CAN

Analog IO

USB

# Re-hosting Challenges



| On chip | Off chip |
|---|---|
| **CPU** | Ethernet |
| AES Accelerator | SD-MMC |
| | Analog IO |
| | USB |

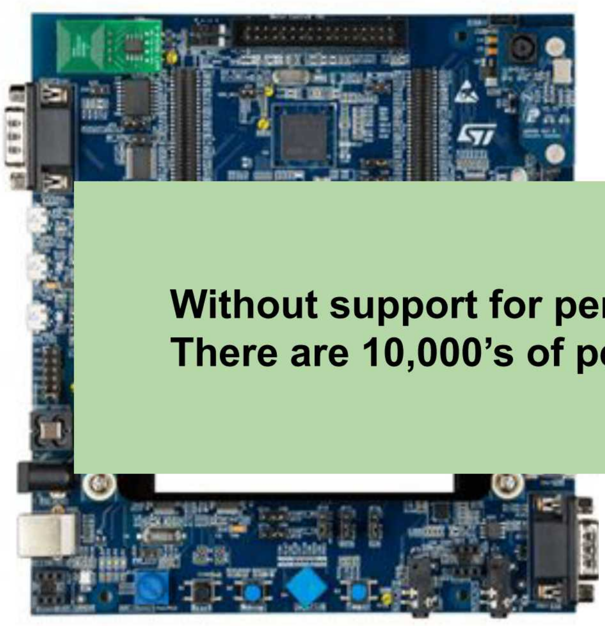
Mouser Lists

44,520 Microcontrollers

3,502 Datasheets

26 Manufactures

# Re-hosting Challenges

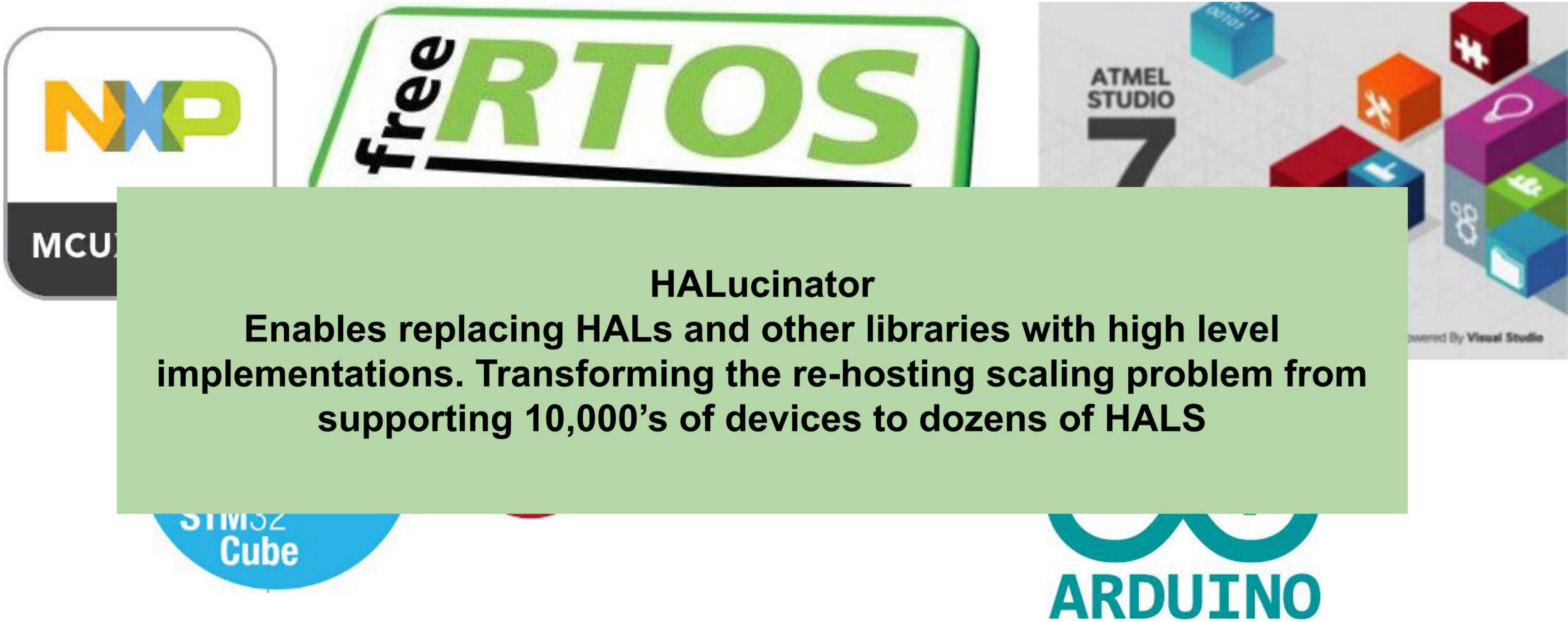


| On chip | Off chip |
|---|---|
| **CPU** | Ethernet |
| AES Accelerator | SD-MMC |
| | Analog IO |
| | USB |

**Without support for peripherals baremetal firmware will not run!**
**There are 10,000’s of peripherals and combinations there of!**

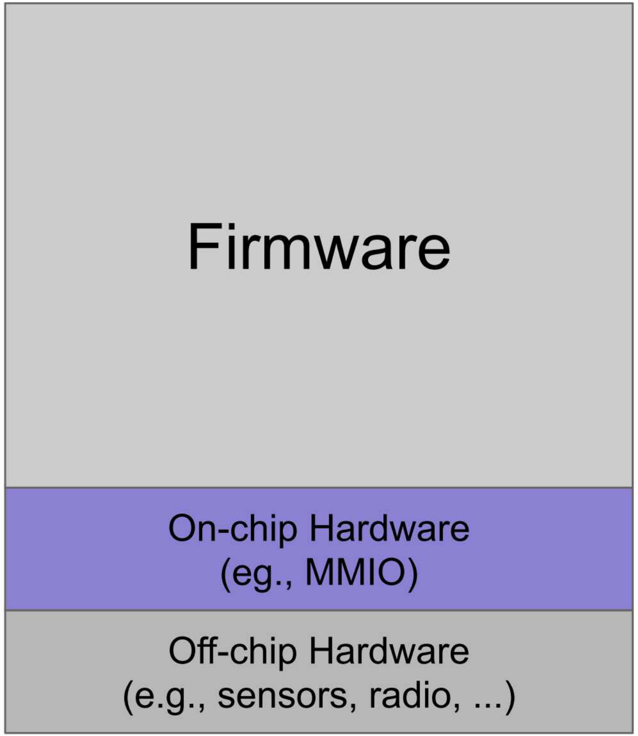
# Hardware Abstraction Libraries

# Hardware Abstraction Libraries



**HALucinator**
**Enables replacing HALs and other libraries with high level implementations. Transforming the re-hosting scaling problem from supporting 10,000’s of devices to dozens of HALS**

# The Modern Firmware Stack

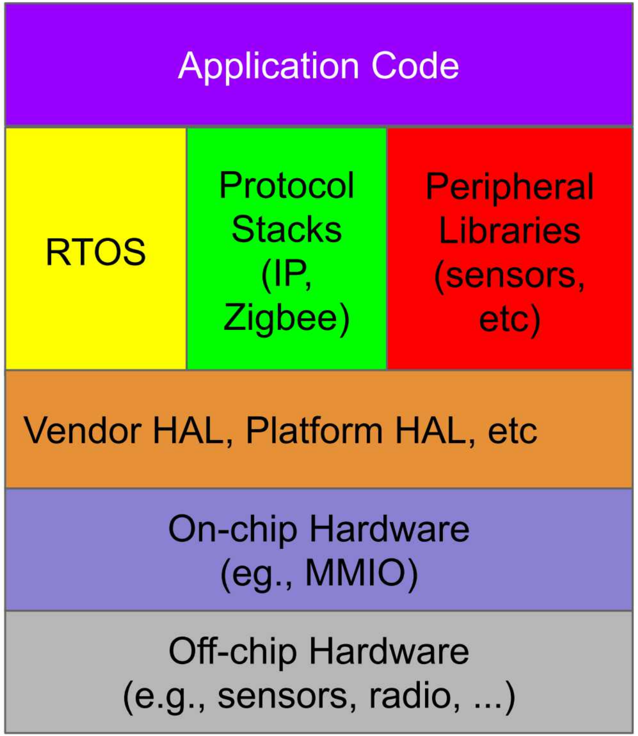


Firmware

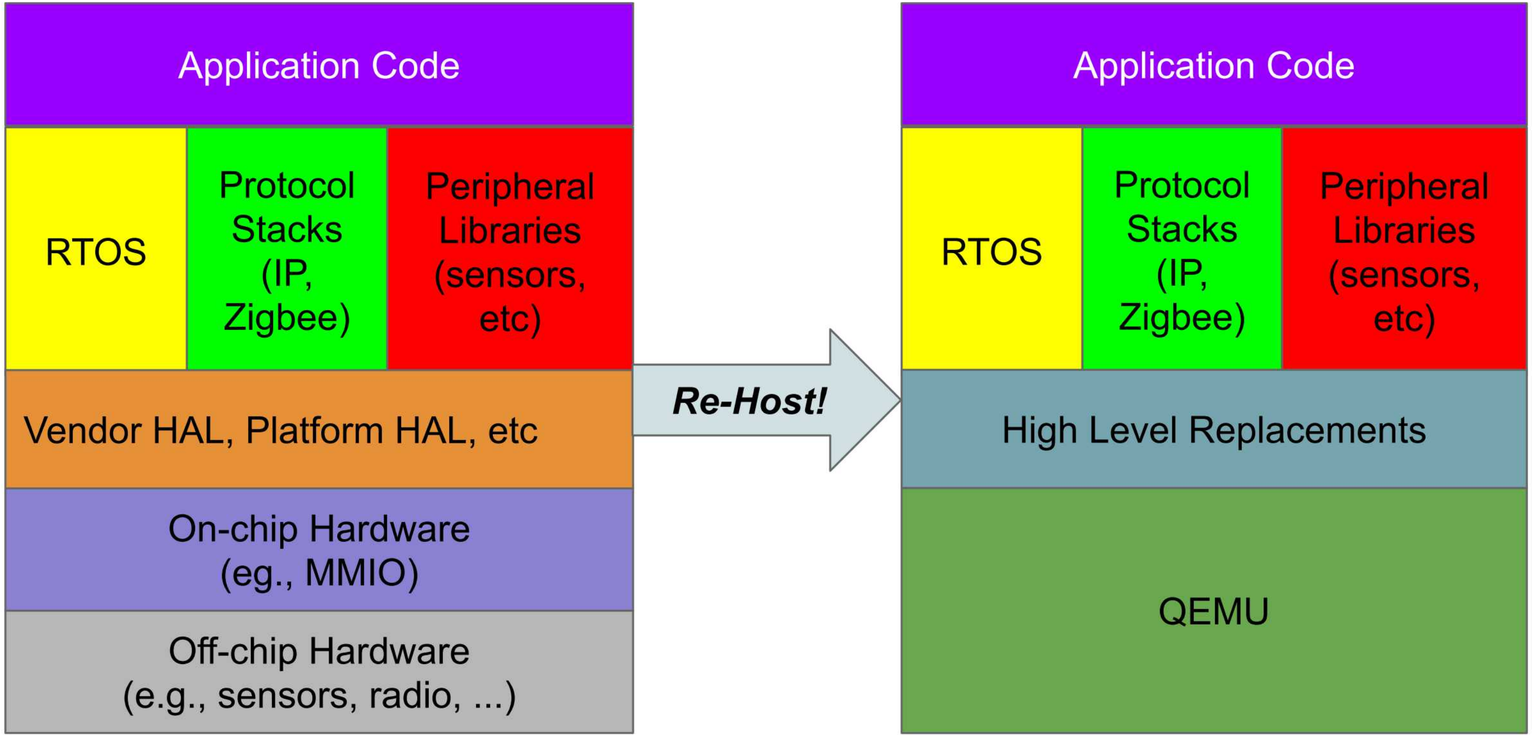
On-chip Hardware
(eg., MMIO)

Off-chip Hardware
(e.g., sensors, radio, ...)

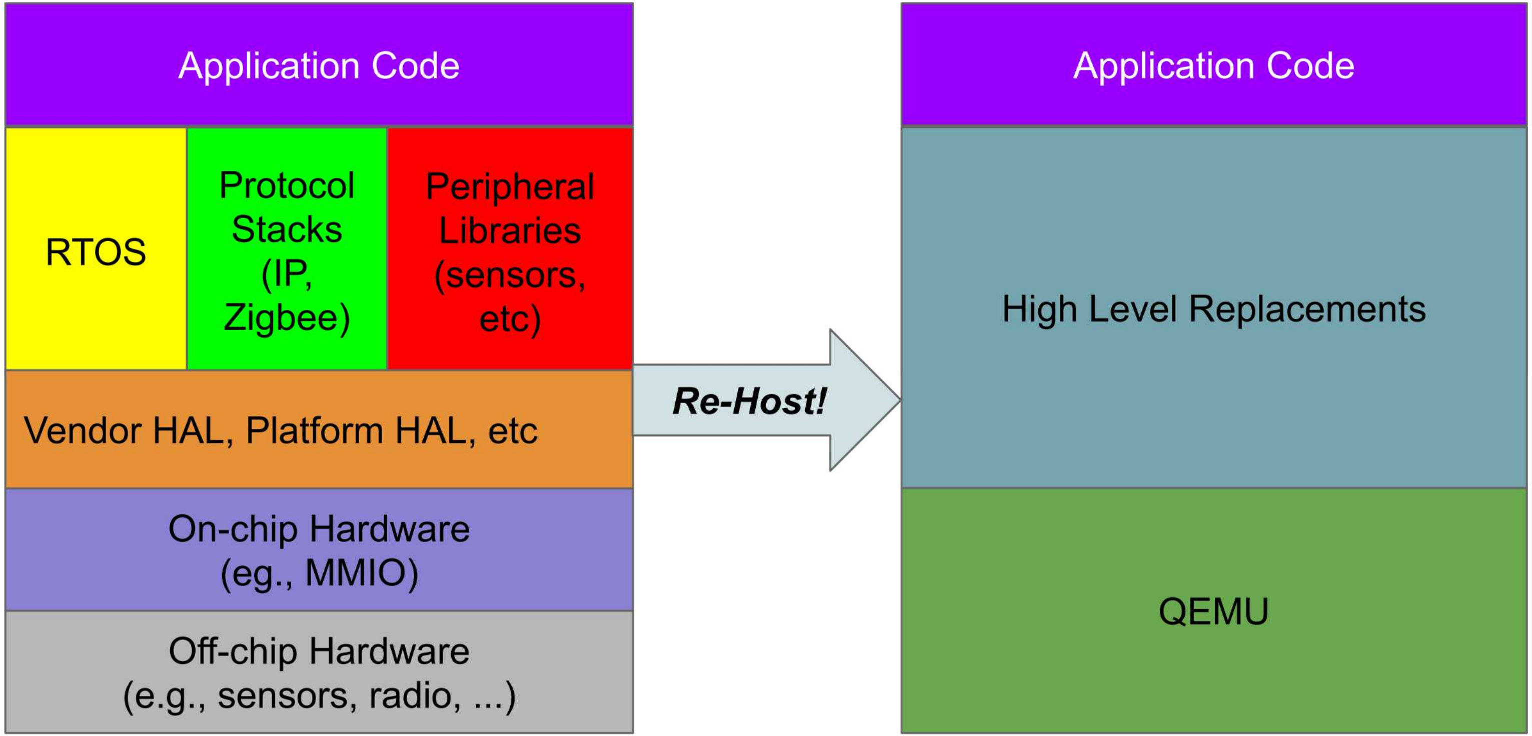
# High Level Emulation


Application Code
RTOS
Protocol Stacks (IP, Zigbee)
Peripheral Libraries (sensors, etc)
Vendor HAL, Platform HAL, etc
On-chip Hardware (eg., MMIO)
Off-chip Hardware (e.g., sensors, radio, ...)

# High Level Emulation


Application Code
RTOS
Protocol Stacks (IP, Zigbee)
Peripheral Libraries (sensors, etc)
Vendor HAL, Platform HAL, etc
On-chip Hardware (eg., MMIO)
Off-chip Hardware (e.g., sensors, radio, ...)
Re-Host!
Application Code
RTOS
Protocol Stacks (IP, Zigbee)
Peripheral Libraries (sensors, etc)
High Level Replacements
QEMU

# High Level Emulation



Application Code
RTOS
Protocol Stacks (IP, Zigbee)
Peripheral Libraries (sensors, etc)
Vendor HAL, Platform HAL, etc
On-chip Hardware (eg., MMIO)
Off-chip Hardware (e.g., sensors, radio, ...)
Re-Host!
Application Code
High Level Replacements
QEMU

# HALucinator enables

Introspection – What is the firmware doing?

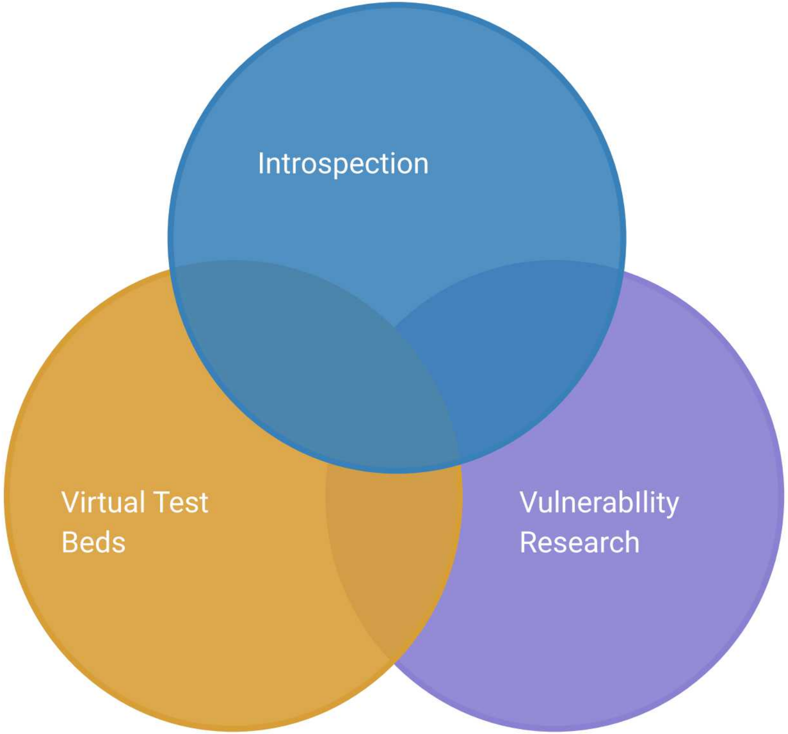
- Debugging/system testing
- Determine effects of malware on firmware
- Experiment with firmware in controlled environment

Vulnerability Research – Is the system vulnerable?

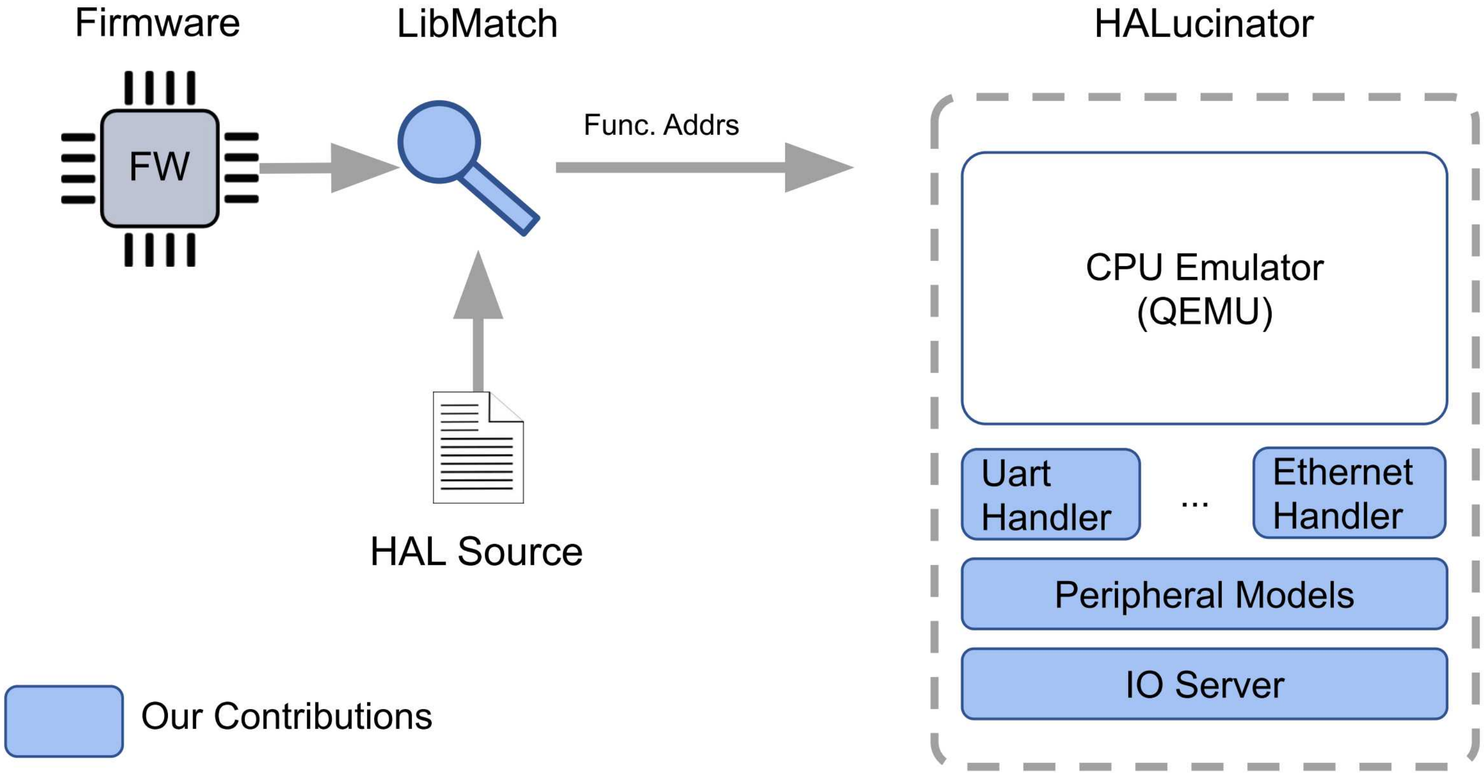
- Identify insecure interfaces
- Find memory corruption errors
- Fuzzing

Virtual Testbeds – How do vulnerabilities impact connected systems?

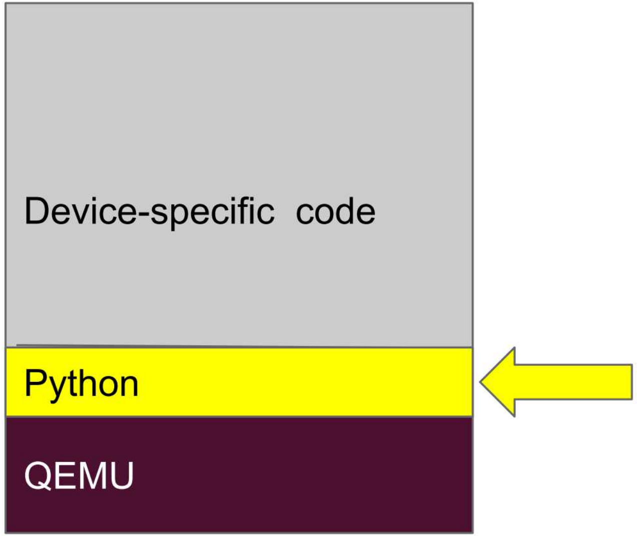
- System of systems modeling
- Firmware in the loop testing
- Software only testbeds of embedded systems



# HALucinator implementation

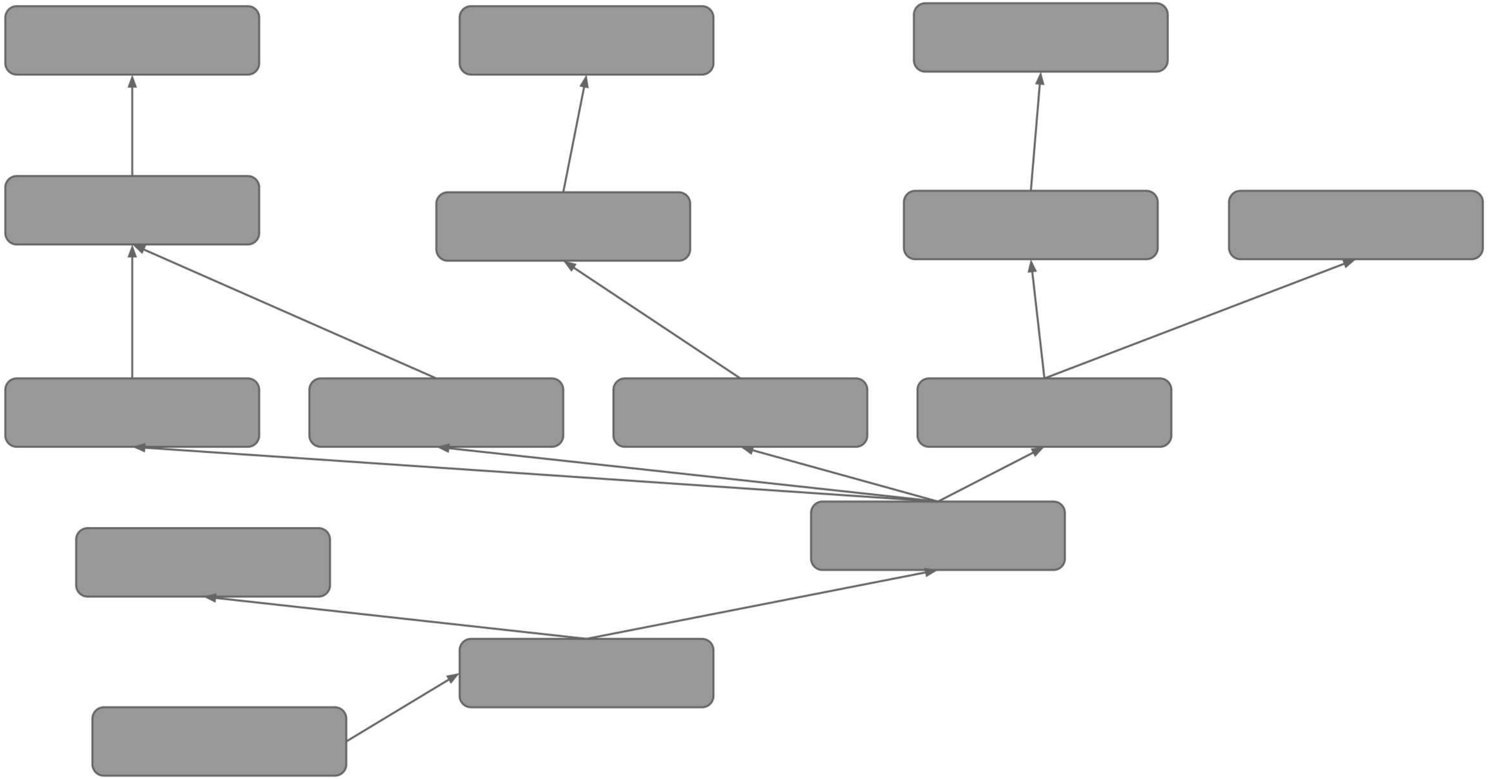


# Handler Example

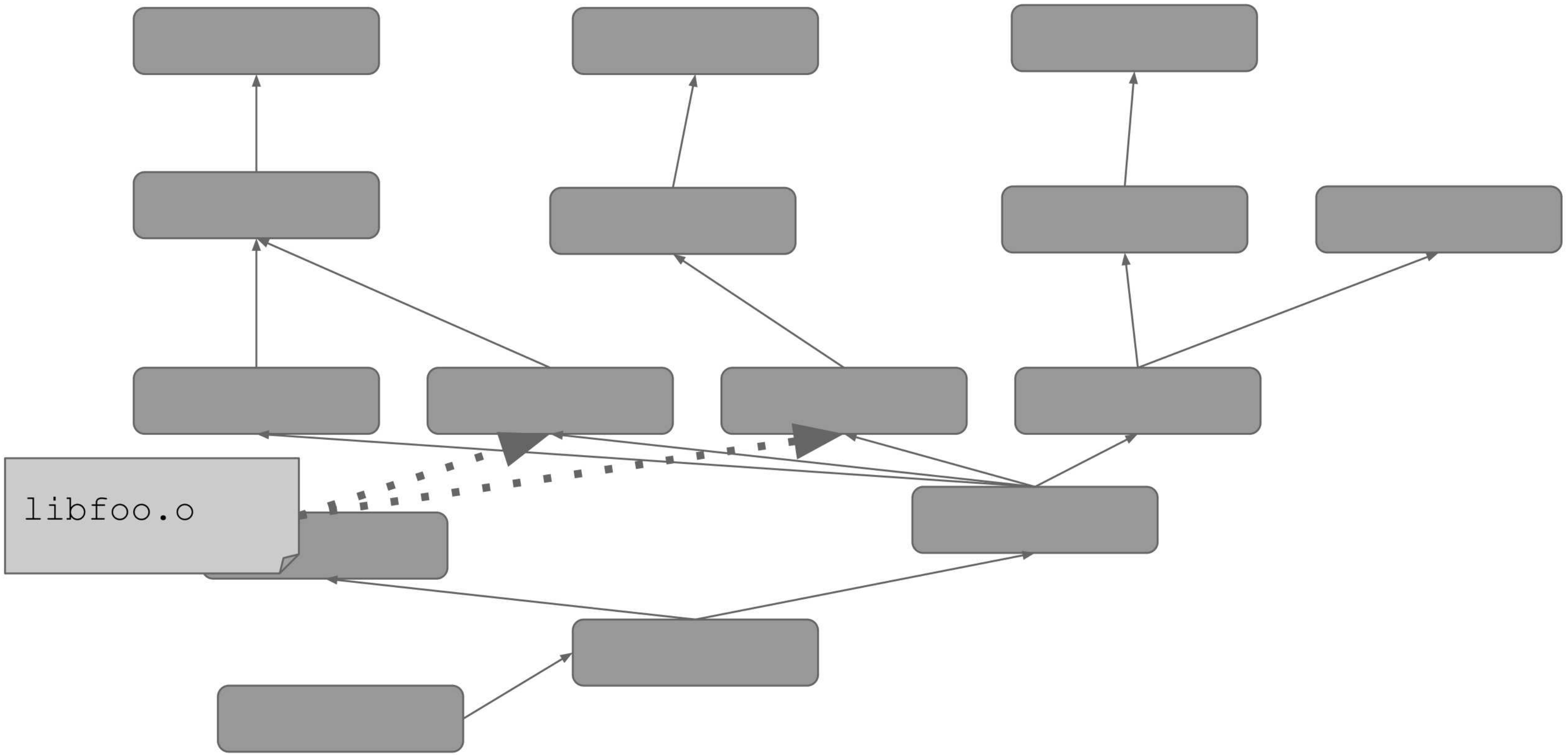


```
def i2c_read_buf(uc):
    # i2c_read_buf(char* buf, int len);
    buf = uc.regs.r1 # arg 0: The buffer
    l = uc.regs.r2   # arg 1: Buffer length
    assert(buf != 0)  # Crash on bad arguments
    assert(len > 0)
    data = I2CModel.rx('i2c', 0, len) # Get the data
                                      # from the virtual bus
    uc.mem[buf] = data                # Store it in the emulator
```

# LibMatch

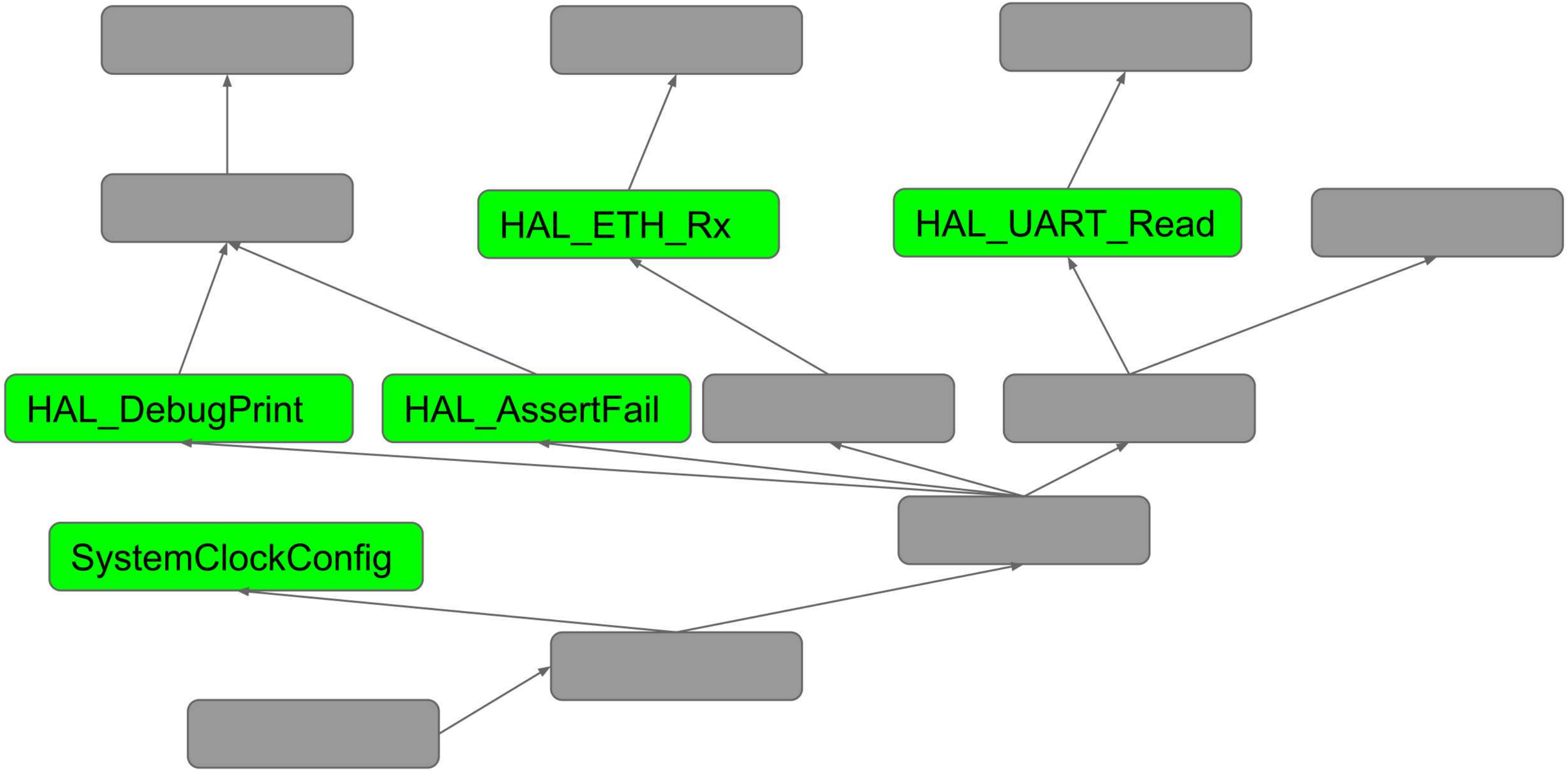


# LibMatch

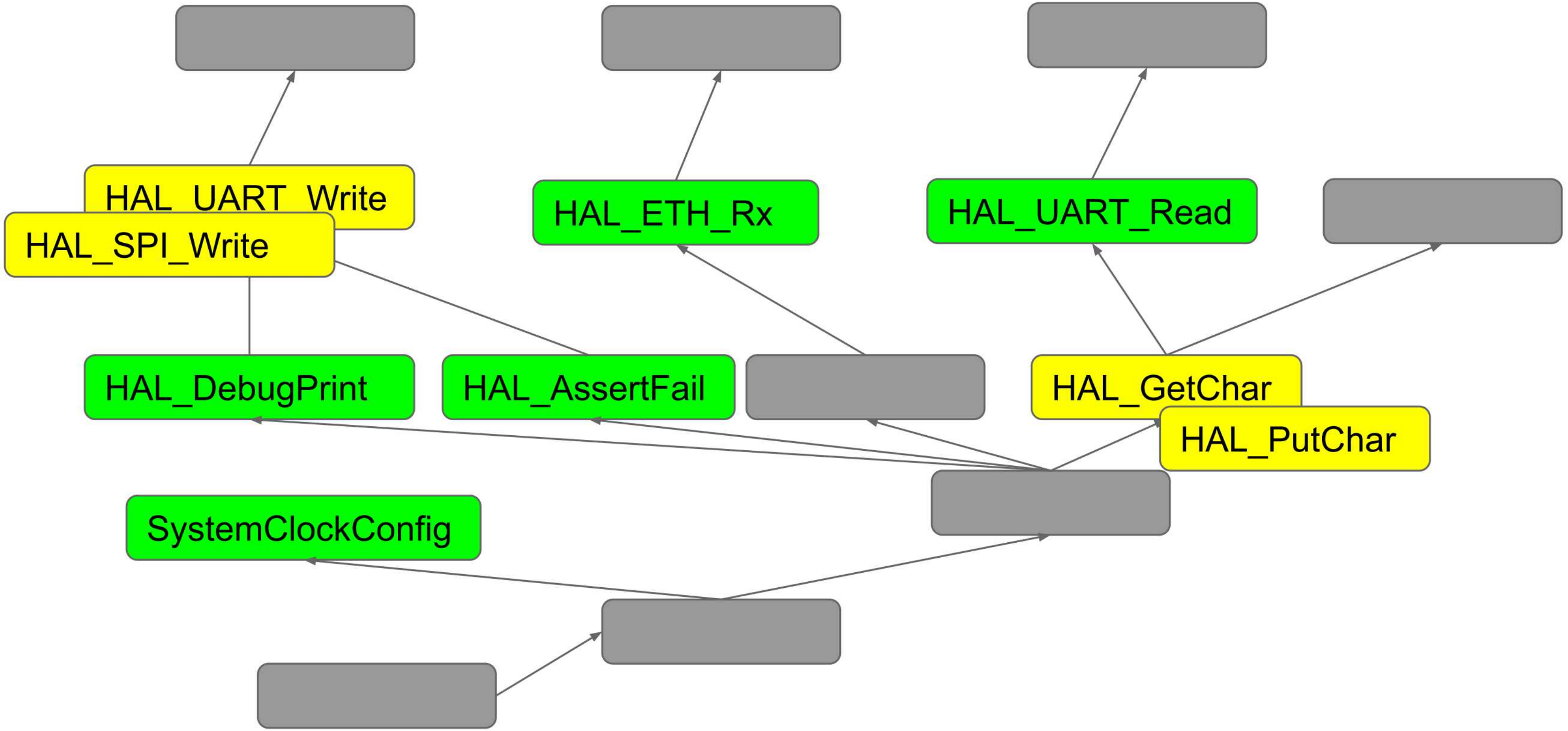


Step 1: Match library content

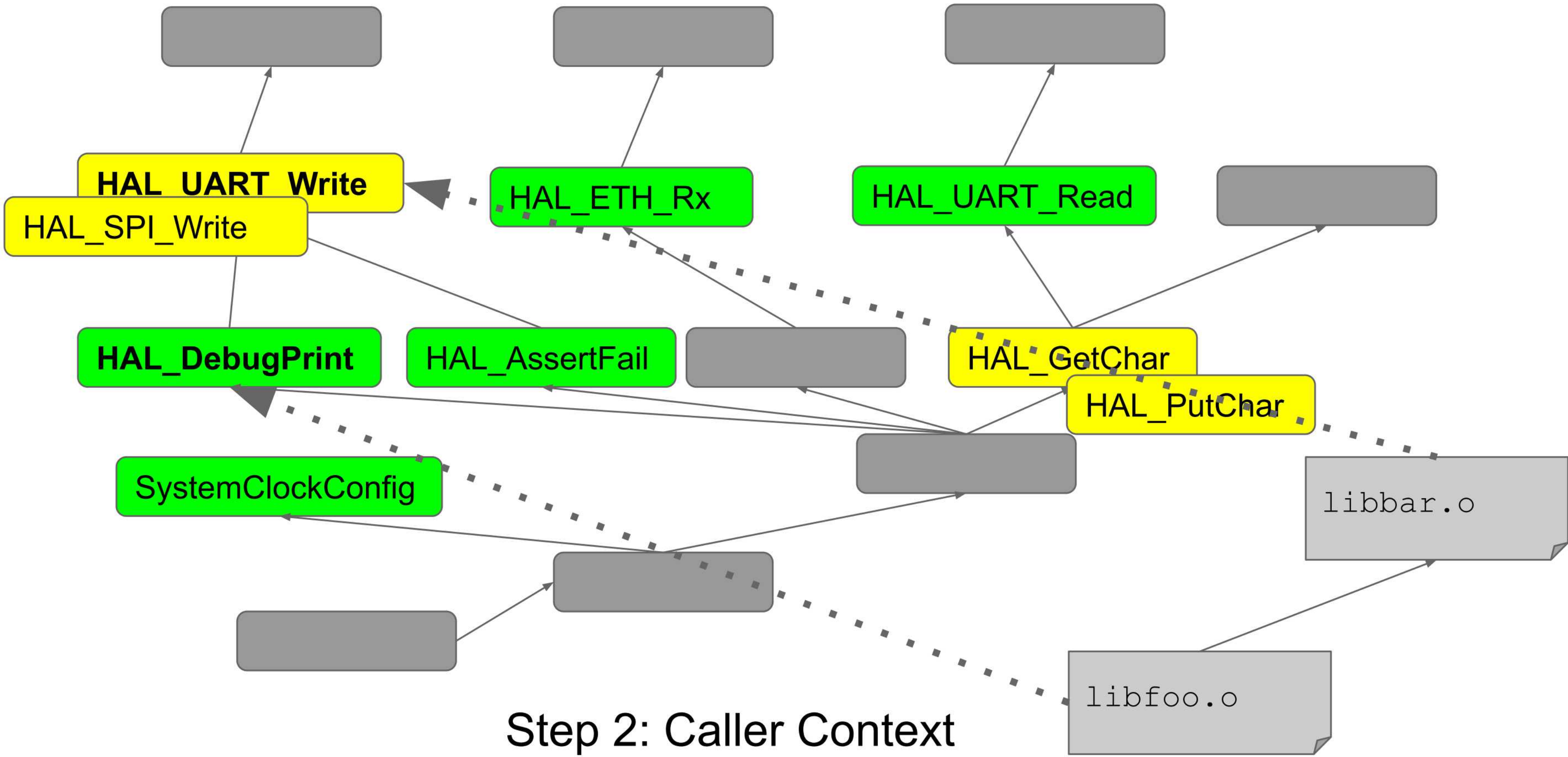
# LibMatch


HAL_ETH_Rx
HAL_UART_Read
HAL_DebugPrint
HAL_AssertFail
SystemClockConfig

# LibMatch

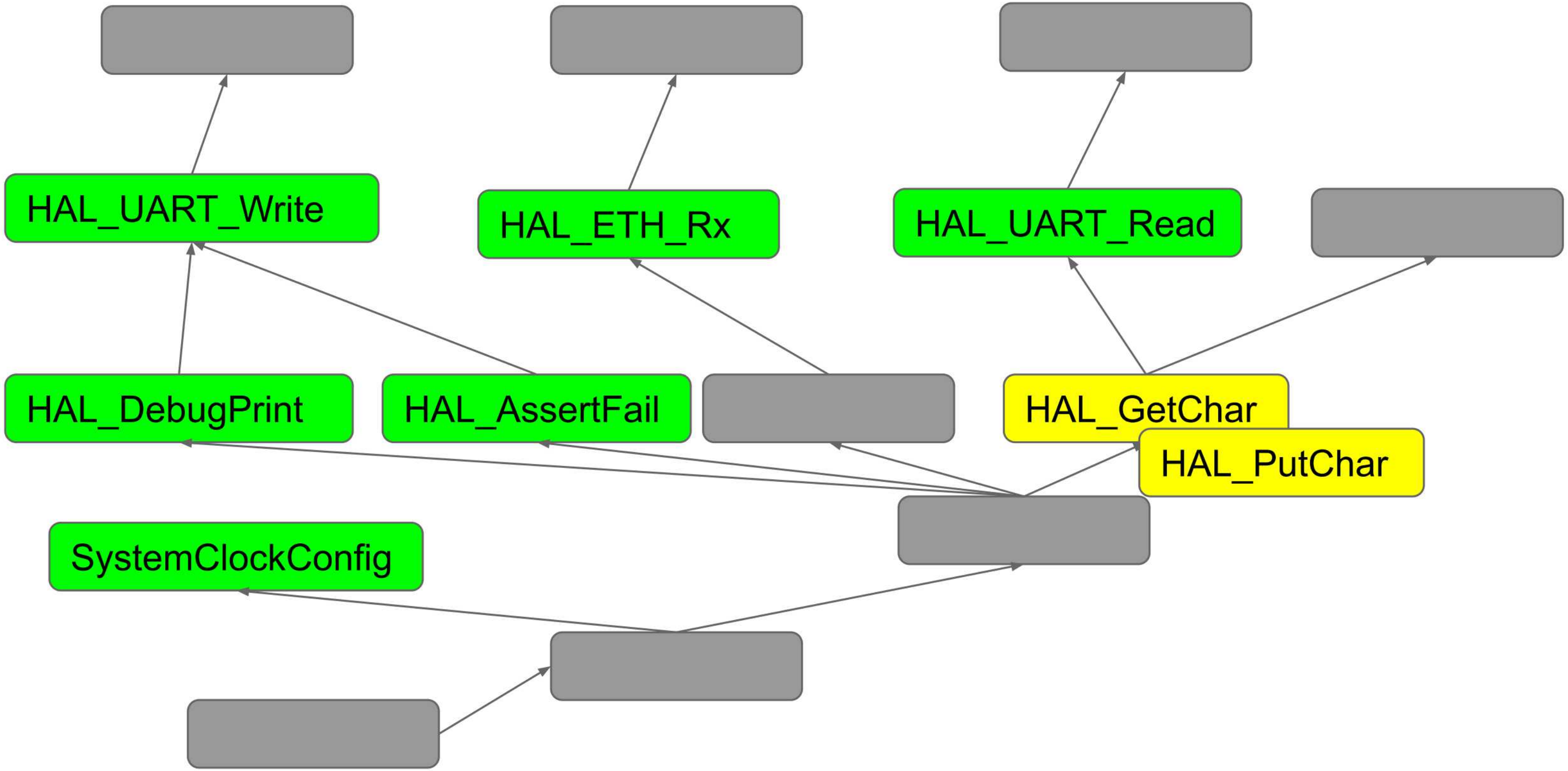

HAL_UART_Write
HAL_SPI_Write
HAL_ETH_Rx
HAL_UART_Read
HAL_DebugPrint
HAL_AssertFail
HAL_GetChar
HAL_PutChar
SystemClockConfig

# LibMatch

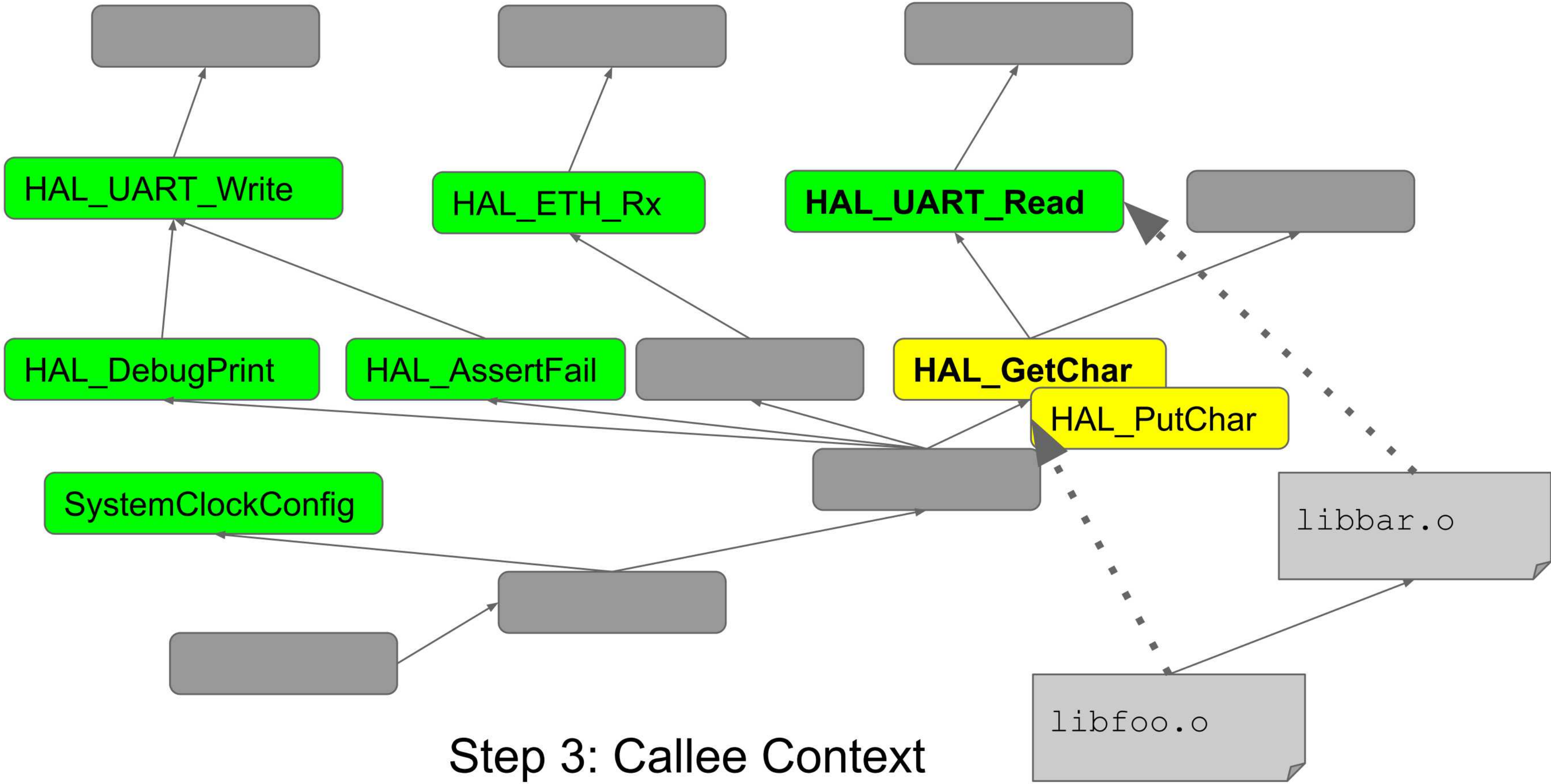


Step 2: Caller Context

# LibMatch

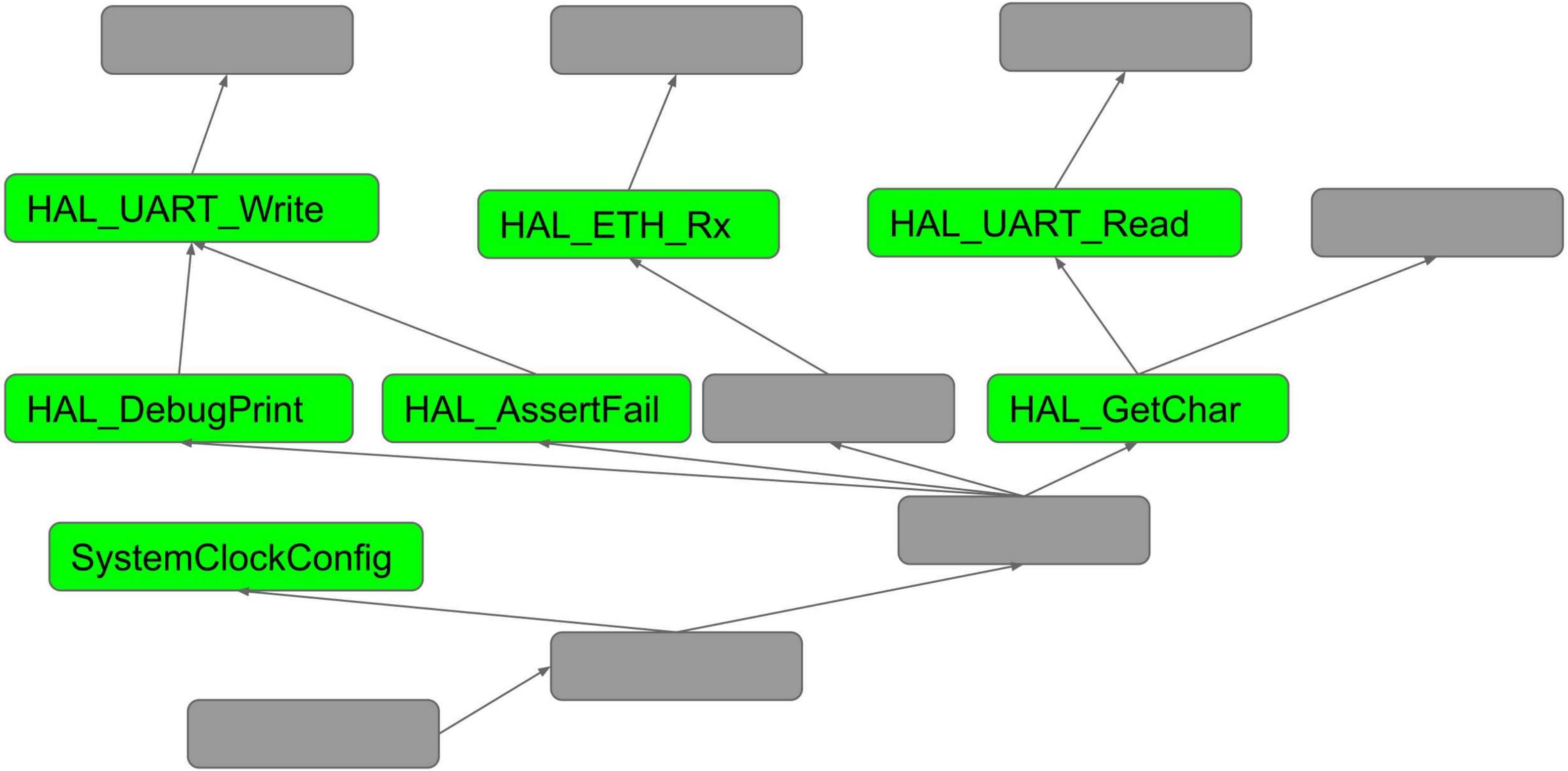

HAL_UART_Write
HAL_ETH_Rx
HAL_UART_Read
HAL_DebugPrint
HAL_AssertFail
HAL_GetChar
HAL_PutChar
SystemClockConfig

# LibMatch

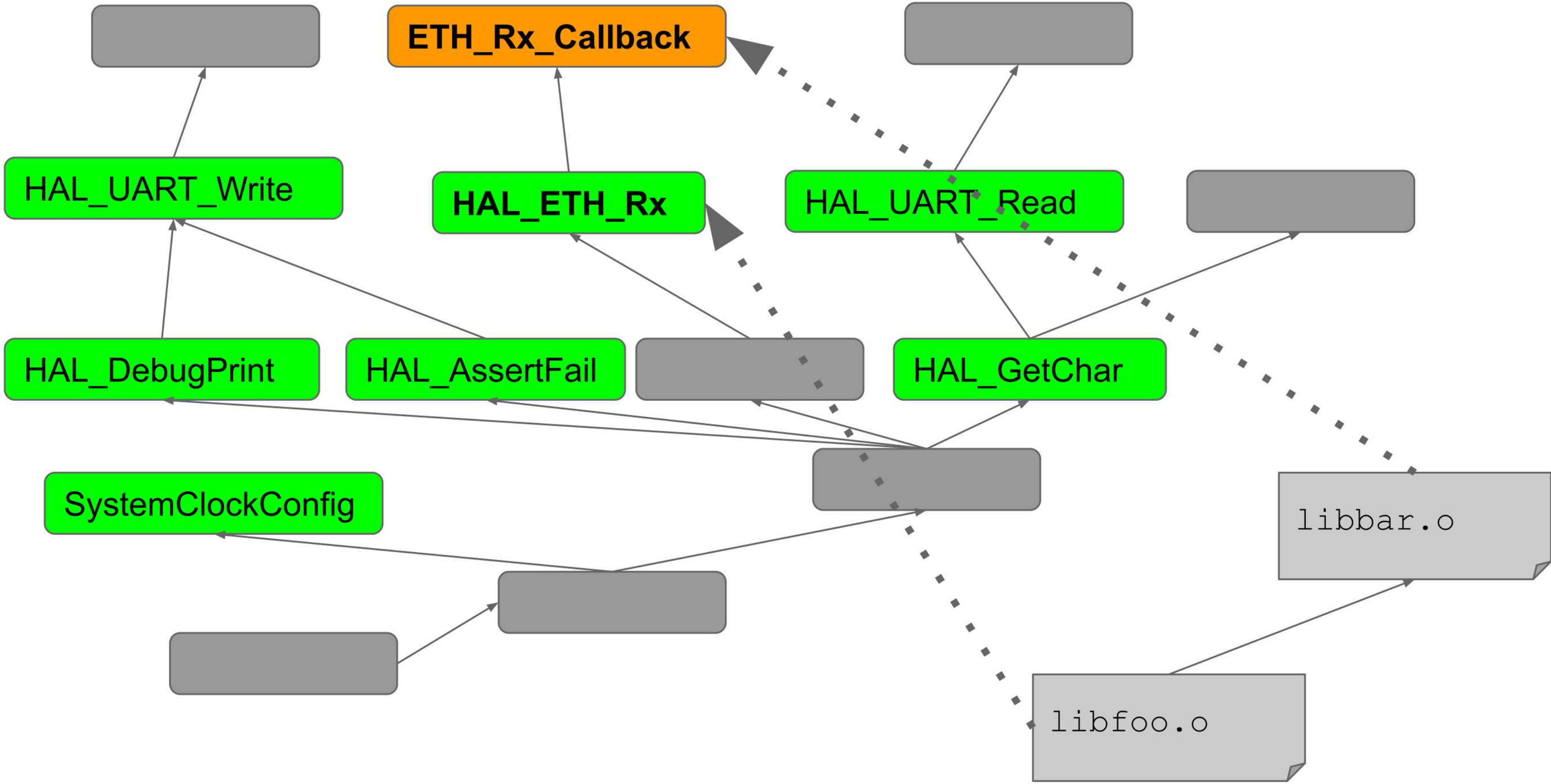


Step 3: Callee Context

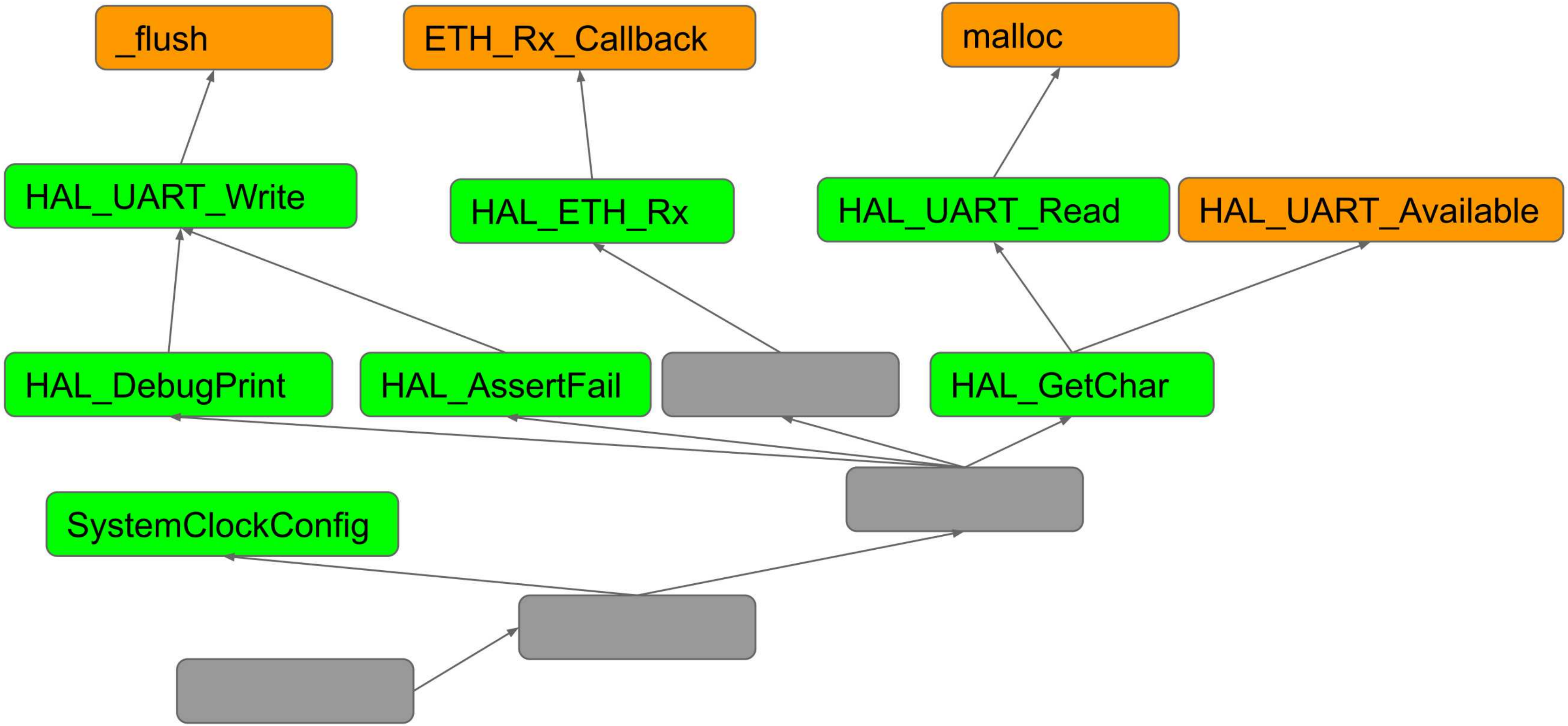
# LibMatch


HAL_UART_Write
HAL_ETH_Rx
HAL_UART_Read
HAL_DebugPrint
HAL_AssertFail
HAL_GetChar
SystemClockConfig

# LibMatch


ETH_Rx_Callback
HAL_UART_Write
HAL_ETH_Rx
HAL_UART_Read
HAL_DebugPrint
HAL_AssertFail
HAL_GetChar
SystemClockConfig
libbar.o
libfoo.o

# LibMatch


_flush
ETH_Rx_Callback
malloc
HAL_UART_Write
HAL_ETH_Rx
HAL_UART_Read
HAL_UART_Available
HAL_DebugPrint
HAL_AssertFail
HAL_GetChar
SystemClockConfig

# hal-fuzz

- Built on AFL-Unicorn
- Program exits when the input is exhausted
- Deterministic timers based on block counts
- Interrupt events also based on block counts
- Crashes detected via Unicorn’s own error detector as well as handler assertions

# 16 Firmware Samples

- ATMEL ASF
  - USART
  - FAT32 on SD-Card
  - **HTTP Server**
  - **6LoWPAN Sender and Receiver**
- STM32Cube
  - UART
  - FAT32 on SD-Card
  - **UDP-Echo Server and Client**
  - **TCP-Echo Server and Client**
  - **PLC**
- NXP -MCUXpresso
  - UART
  - **UDP Echo Server**
  - **TCP Echo Server**
  - **HTTP Server**

# LibMatch Results

| | “Naive” LibMatch (Bindiff) | LibMatch w/ context |
|---|---:|---:|
| Correct | 74.5% | 87.4% |
| Missing | 5.0% | 3.2% |
| Collisions | 18.8% | 8.5% |
| Incorrect | 2.5% | 0.9% |
| External | -- | 9.96% |

**% avg matches across 16 test binaries**

# Ease of Implementation

Three Handler categories:

- **Trivial**: Does nothing / returns a constant
- **Translating**: Collects arguments, interacts with a Model, returns a result
- **Internal Logic**: Needs to re-implement undocumented internal details

# Ease of Implementation

- Over 85% of handlers require little effort
  - 44.5 are “trivial”
  - 42.2 are “translating”

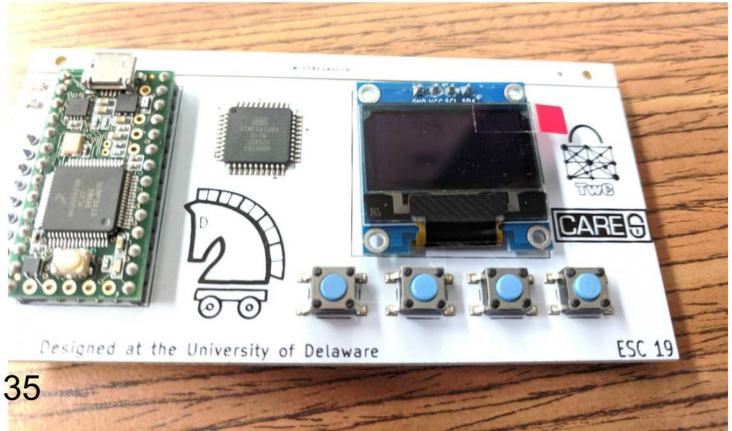
- Remainder: “Internal logic”
  - HAL behavior doesn’t abstract hardware well enough
  - HAL behavior makes assumptions not in the docs (e.g., uses its own heap allocator)

# Fuzzing!

- Hundreds of millions of parallel executions
- Found crashes in ST-PLC, Atmel HTTP server, Atmel 6LowPAN(w/ Contiki)
- Fuzzed HTTP server at two different levels, found crashes in both
- **Discovered CVE-2019-8359 and CVE-2019-9183 in Contiki’s network stack**

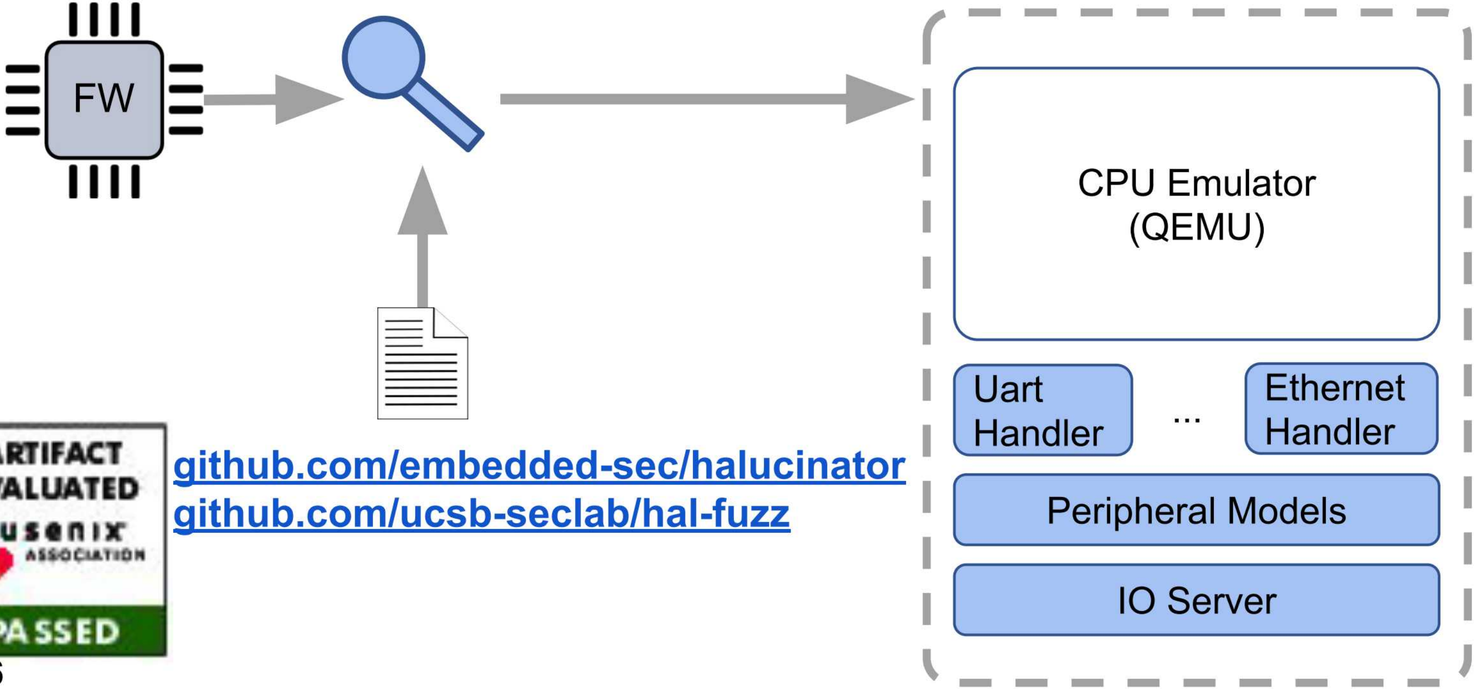
# CSAW ESC 2019 results

- Re-hosted ARM portion of all challenge sets
- Solved 18/19 challenges
- Verified 17/18 solutions w/ just the emulator
- Solved 3 challenges automatically using fuzzing
- Won first place!



# Conclusion

HALucinator eliminates implementing 10,000s of peripherals by using HALs



ARTIFACT EVALUATED usenix ASSOCIATION PASSED

[github.com/embedded-sec/halucinator](https://github.com/embedded-sec/halucinator)
[github.com/ucsb-seclab/hal-fuzz](https://github.com/ucsb-seclab/hal-fuzz)