
black hat
MIDDLE EAST AND AFRICA
REVERSING MCU WITH FIRMWARE EMULATION
15 – 17 NOVEMBER 2022
RIYADH FRONT EXHIBITION CENTRE
SAUDI ARABIA
MuChen SU
NGUYEN Anh Quynh
KaiJern Lau
Zheng YU
STRATEGIC SPONSORS
GOLD SPONSORS
CO-ORGANISED BY
SPIRE
IBM
Infoblox
Honeywell
informatech
SAUDI FEDERATION FOR CYBERSECURITY, PROGRAMMING & DRONES

About xwings and Quynh

Qiling Framework

Cross platform and multi architecture advanced binary emulation framework

- https://qiling.io
- Lead Developer
- Founder

Badge Maker

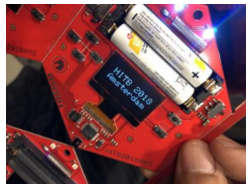
Electronic fan boy, making toys from hacker to hacker

- Reversing Binary
- Reversing IoT Devices
- Part Time CtF player

Some Recent Talk (Partial)

- 2019, DEFCON USA, Qiling Framework Preview
- 2019, Zeronights, Qiling Framework to Public
- 2020, Nullcon GOA, Building Reversing Tools with Qiling
- 2020, HITB AMS, Building Reversing Tools with Qiling
- 2020, HITB Singapore, Training, How to Hack IoT with Qiling
- 2020, HITB UAE, Training, Lightweight Binary Analyzer
- 2020, Blackhat USA, Building IoT Fuzzer with Qiing
- 2020, Blackhat Singapore, Lightweight Binary Analyzer
- 2020, Blackhat Europe, Deep Dive Into Obfuscated Binary

Badge Designer for Hacking Conferences





Dr. NGUYEN Anh Quynh

Founder:
- Capstone Engine
- Unicorn Engine
- Keystone Engine

Co-Founder:
- Qiling Framework

Papers:
- Usenix LISA, 12/2007
- Usenix LISA, 12/2006
- Usenix Annual Technical Conference 2007 (Usenix), 12/2007
- ACM Press, 03/2007
- IEEE CS Press, 07/2006

Conferences:
- Blackhat/Defcon: 2007, 2010, 2013, 2014, 2015, 2016, 2020
- etc.

What Is Qiling Framework

Open-source Security Frameworks Since 2013

Overview

Architecture

Loader

OS

Posix

macOS®

EDL

MBR

Debugger

qdb

GHIDRA

Extensions

coverage

idaplugin

report

sanitizers

windows sdk

Instrumentation

External Hardware Emulation

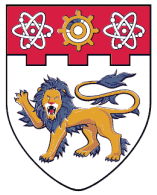
Qiling Framework

Unicorn Engine

SentinelOne

Qualcomm

腾讯
Tencent



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Carbon Black.

UNIVERSITY OF GEORGIA

Noroff
School of technology and digital media

KASPERSKY lab

IBM Watson

IBM华生实验室

accenture

IIJ
Internet Initiative Japan

TALOS
CISCO

CookieYes

AFRL-RI-RS-TM-2020-001
Version 4 of 4

EDGE OF THE ART IN VULNERABILITY RESEARCH VERSION 4 OF 4

TWO SIX LABS

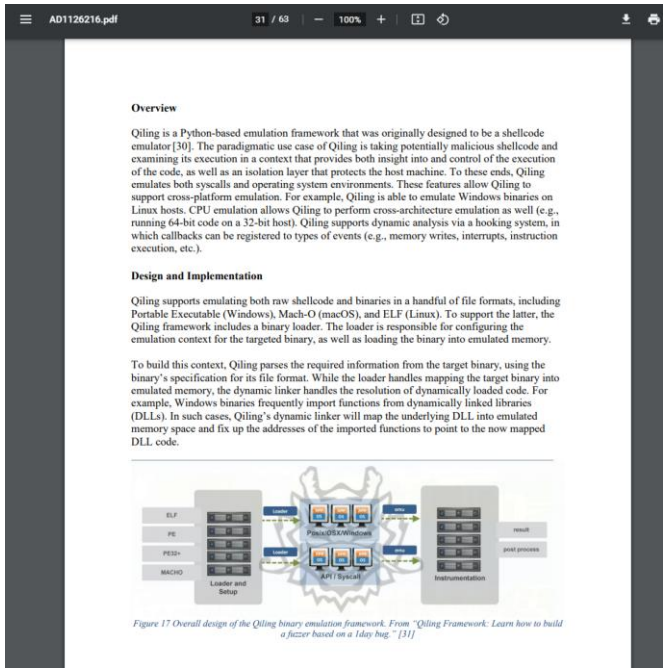
MARCH 2021

TECHNICAL MEMORANDUM

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY INFORMATION DIRECTORATE


AD1126216.pdf 31 / 63 100%

Overview

Qiling is a Python-based emulation framework that was originally designed to be a shellcode emulator [30]. The paradigmatic use case of Qiling is taking potentially malicious shellcode and examining its execution in a context that provides both insight into and control of the execution of the code, as well as an isolation layer that protects the host machine. To these ends, Qiling emulates both syscalls and operating system environments. These features allow Qiling to support cross-platform emulation. For example, Qiling is able to emulate Windows binaries on Linux hosts. CPU emulation allows Qiling to perform cross-architecture emulation as well (e.g., running 64-bit code on a 32-bit host). Qiling supports dynamic analysis via a hooking system, in which callbacks can be registered to types of events (e.g., memory writes, interrupts, instruction execution, etc.).

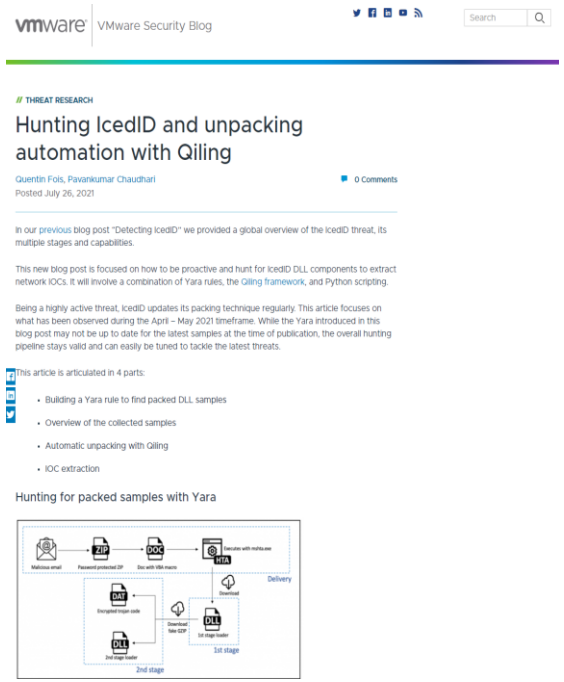
Design and Implementation

Qiling supports emulating both raw shellcode and binaries in a handful of file formats, including Portable Executable (Windows), Mach-O (macOS), and ELF (Linux). To support the latter, the Qiling framework includes a binary loader. The loader is responsible for configuring the emulation context for the targeted binary, as well as loading the binary into emulated memory.

To build this context, Qiling parses the required information from the target binary, using the binary's specification for its file format. While the loader handles mapping the target binary into emulated memory, the dynamic linker handles the resolution of dynamically loaded code. For example, Windows binaries frequently import functions from dynamically linked libraries (DLLs). In such cases, Qiling's dynamic linker will map the underlying DLL into emulated memory space and fix up the addresses of the imported functions to point to the now mapped DLL code.

Figure 17 Overall design of the Qiling binary emulation framework. From "Qiling Framework: Learn how to build a fuzzer based on a 1day bug." [31]

URL: https://apps.dtic.mil/sti/pdfs/AD1126216.pdf


vmware VMware Security Blog

// THREAT RESEARCH

Hunting IcedID and unpacking automation with Qiling

Quentin Fois, Pavankumar Chaudhari
Posted July 26, 2021
0 Comments

In our previous blog post "Detecting IcedID" we provided a global overview of the IcedID threat, its multiple stages and capabilities.

This new blog post is focused on how to be proactive and hunt for IcedID DLL components to extract network IOCs. It will involve a combination of Yara rules, the Qiling framework, and Python scripting.

Being a highly active threat, IcedID updates its packing technique regularly. This article focuses on what has been observed during the April – May 2021 timeframe. While the Yara introduced in this blog post may not be up to date for the latest samples at the time of publication, the overall hunting pipeline stays valid and can easily be tuned to tackle the latest threats.

This article is articulated in 4 parts:

- Building a Yara rule to find packed DLL samples
- Overview of the collected samples
- Automatic unpacking with Qiling
- IOC extraction

Hunting for packed samples with Yara

Automatic unpacking with Qiling

As shown in the Yara creation section, the outer packed layer of the 1st stage DLL is full of opaque predicates and other obfuscation artifacts. While one could reverse the packer code and come up with a 100% static approach to unpack the inner layer of the DLL, this would take a non-negligeable amount of time and it would not be resilient to small packer change.

Previous manual experimentation showed that we were able to extract a clean version of the unpacked sample by using a debugger and placing a breakpoint on CreateThread. When the executed DLL triggers the breakpoint, we could then use PeSieve tool to extract the unpacked PE from memory.

However, this required too much manual interaction and did not scale well, so we found a middle ground by using Qiling.

Qiling is a lightweight emulator framework, easy to instrument, with a ready-to-use PE loader and Python bindings. It has already been introduced through multiple talks and blogpost.

Running a packed sample through Qiling is as easy as a few lines of python:

```
from qiling import *

# initialize emulator (x86_64 windows)
ql = Qiling(["./structButtonMemory.jpg"], "/qiling/examples/rootfs/x8664_windows")

# start emulation
ql.run(begin=0x180007FDF)
```

Figure 14 Executing IcedID 1st stage DLL via Qiling

Qiling output is rather verbose by default. One can see multiple calls to interesting windows API functions that can be used to have a better understanding of the execution flow and how the sample unpacks itself.

The output can be split into 4 parts:

1). Loader initialization

```
>>> from qiling import *
>>> ql = Qiling(["./structButtonMemory.jpg"], "/qiling/examples/rootfs/x8664_windows")
[=] Initiate stack address at 0x7fffffffe000
[=] Loading ./structButtonMemory.jpg to 0x180000000
[=] PE entry point at 0x180008c37
[=] TEB addr is 0x6000030
[=] PEB addr is 0x60000b0
[=] Loading /qiling/examples/rootfs/x8664_windows/Windows/System32/ntdll.dll to 0x7ffff0000000
[!] Warnings while loading /qiling/examples/rootfs/x8664_windows/Windows/System32/ntdll.dll:
[!]   - SizeOfHeaders is smaller than AddressOfEntryPoint: this file cannot run under Windows 8.
[!]   - AddressOfEntryPoint lies outside the sections' boundaries. AddressOfEntryPoint: 0x0
[=] Done with loading /qiling/examples/rootfs/x8664_windows/Windows/System32/ntdll.dll
[=] Loading /qiling/examples/rootfs/x8664_windows/Windows/System32/kernel32.dll to 0x7ffff01e1000
[=] Done with loading /qiling/examples/rootfs/x8664_windows/Windows/System32/kernel32.dll
[=] Loading /qiling/examples/rootfs/x8664_windows/Windows/System32/shlwapi.dll to 0x7ffff8292000
[=] Done with loading /qiling/examples/rootfs/x8664_windows/Windows/System32/shlwapi.dll
[=] Loading /qiling/examples/rootfs/x8664_windows/Windows/System32/user32.dll to 0x7ffff82e4000
[=] Done with loading /qiling/examples/rootfs/x8664_windows/Windows/System32/user32.dll
[=] Loading /qiling/examples/rootfs/x8664_windows/Windows/System32/gdi32.dll to 0x7ffff0475000
[=] Done with loading /qiling/examples/rootfs/x8664_windows/Windows/System32/gdi32.dll
```

Figure 15 Qiling sample initialization logs

URL: https://blogs.vmware.com/security/2021/07/hunting-icedid-and-unpacking-automation-with-qiling.html

Similarity

User Mode Emulation

qemu-usermode

- The TOOL
- Limited OS Support, Very Limited
- No Multi OS Support
- No Instrumentation
- **Syscall Forwarding**

usercorn

- Very good project !
- It's a Framework !
- Mostly *nix based only
- Limited OS Support (No Windows)
- Go and Lua is not hacker's friendly
- **Syscall Forwarding**

Binee

- Very good project too
- Only X86 (32 and 64)
- Limited OS Support
- Only PE Files
- Just a tool, we don't need a tool
- Again, is GO

WINE

- Limited ARCH Support
- Limited OS Support, only Windows
- Not Sandbox Designed
- No Instrumentation

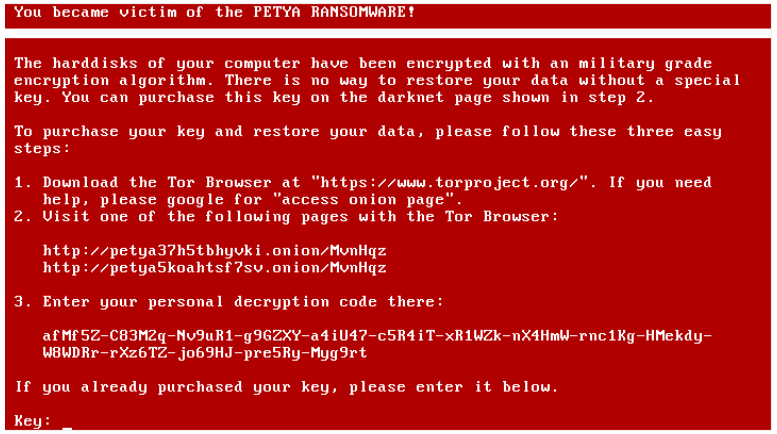
Speakeasy

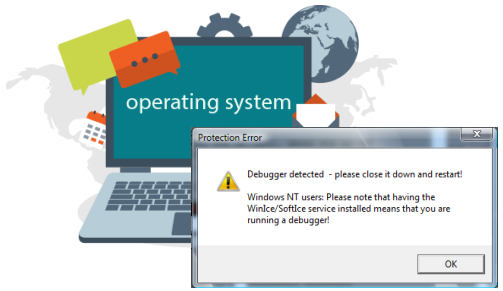
- Very good project too
- X86 32 and 64
- PE files and Driver
- Limited OS Support
- Only Windows

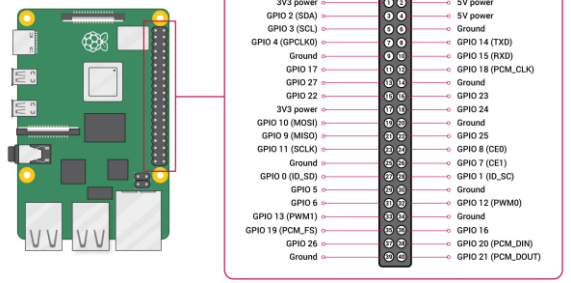
Zelos

- Very good project !
- It's a Framework !
- Linux based only (No Windows)
- Incomplete support for Linux multi arch

Current Virtual Machine Limitation







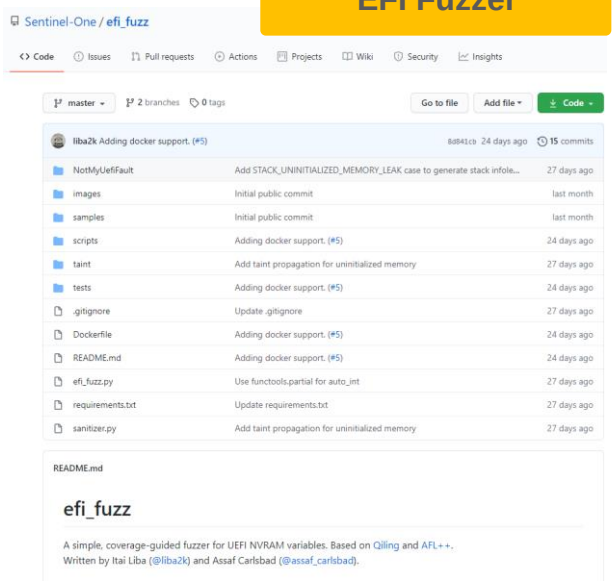
- MBR
- UEFI
- Smart Contract
- GPIO
- Anti-Anti Debug
- Qualcomm EDL

Most modern platform are either limited or NONE emulation or proper analysis tools

The Framework

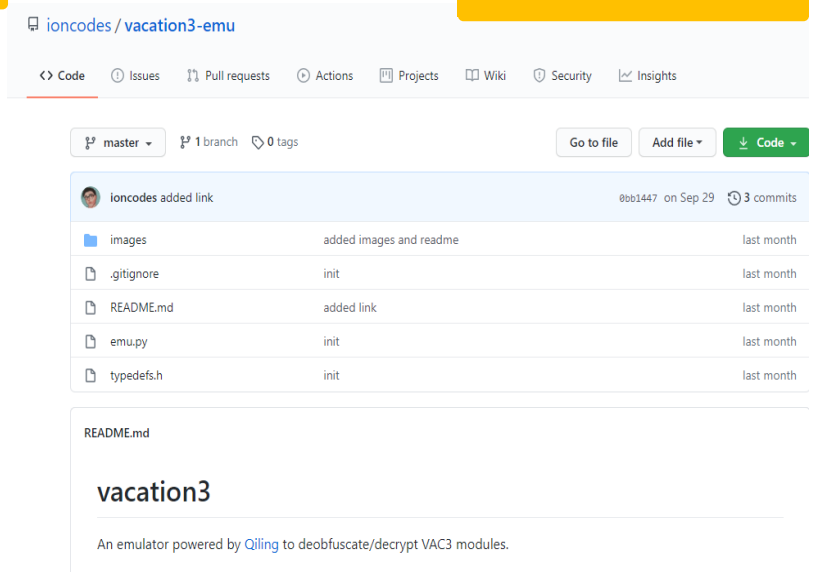
Framework, Not a Tool

EFI Fuzzer



Decoder

VAC3 Emulator



- Binary Fuzzer
- IoT Fuzzer
- Malware Sandbox
- CTF Solver
- IoT Emulator
- MacOS Emulator
- IOS Emulator
- Binary Decrypt

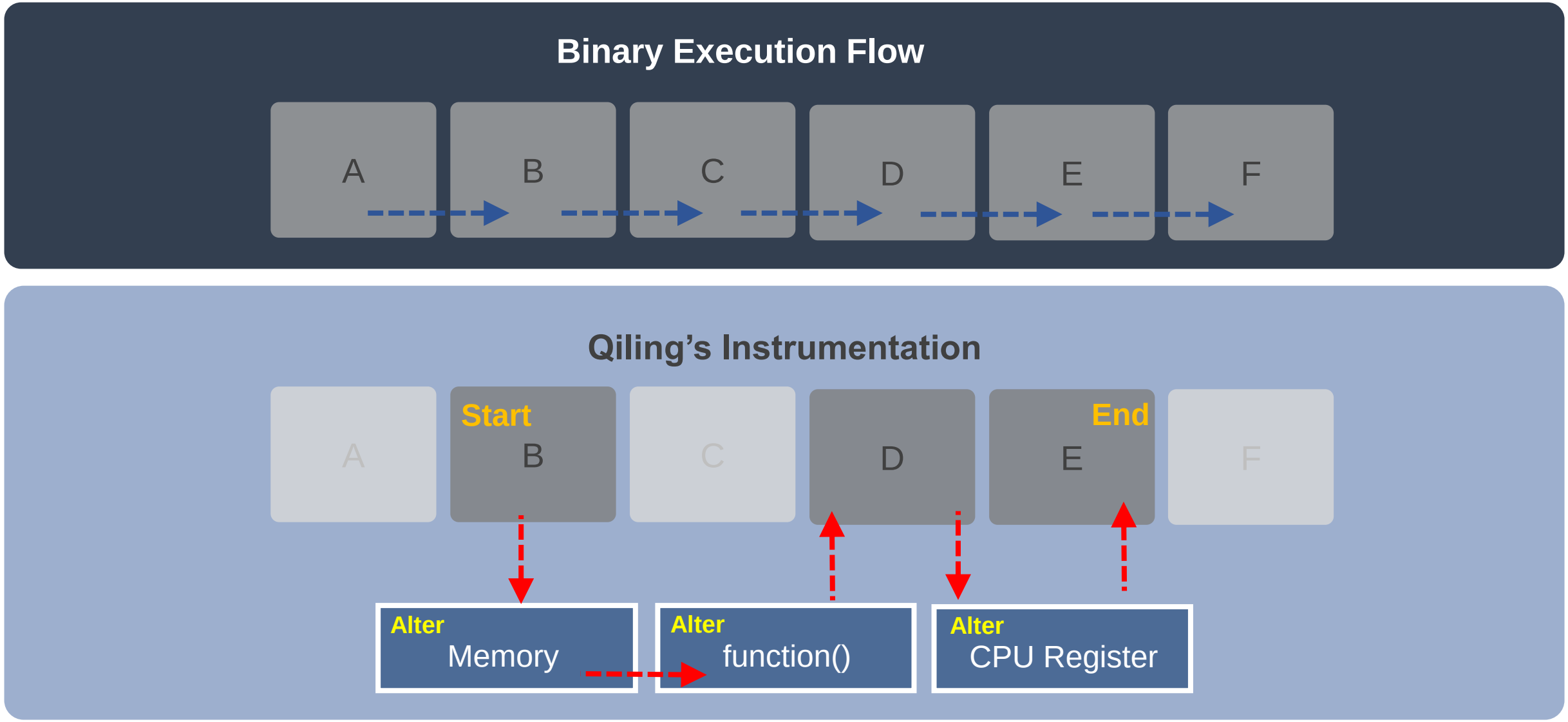
Instrumentation (Qiling's API)

Qiling Framework

CPU Architecture | Loader | OS | Debugger | Extensions

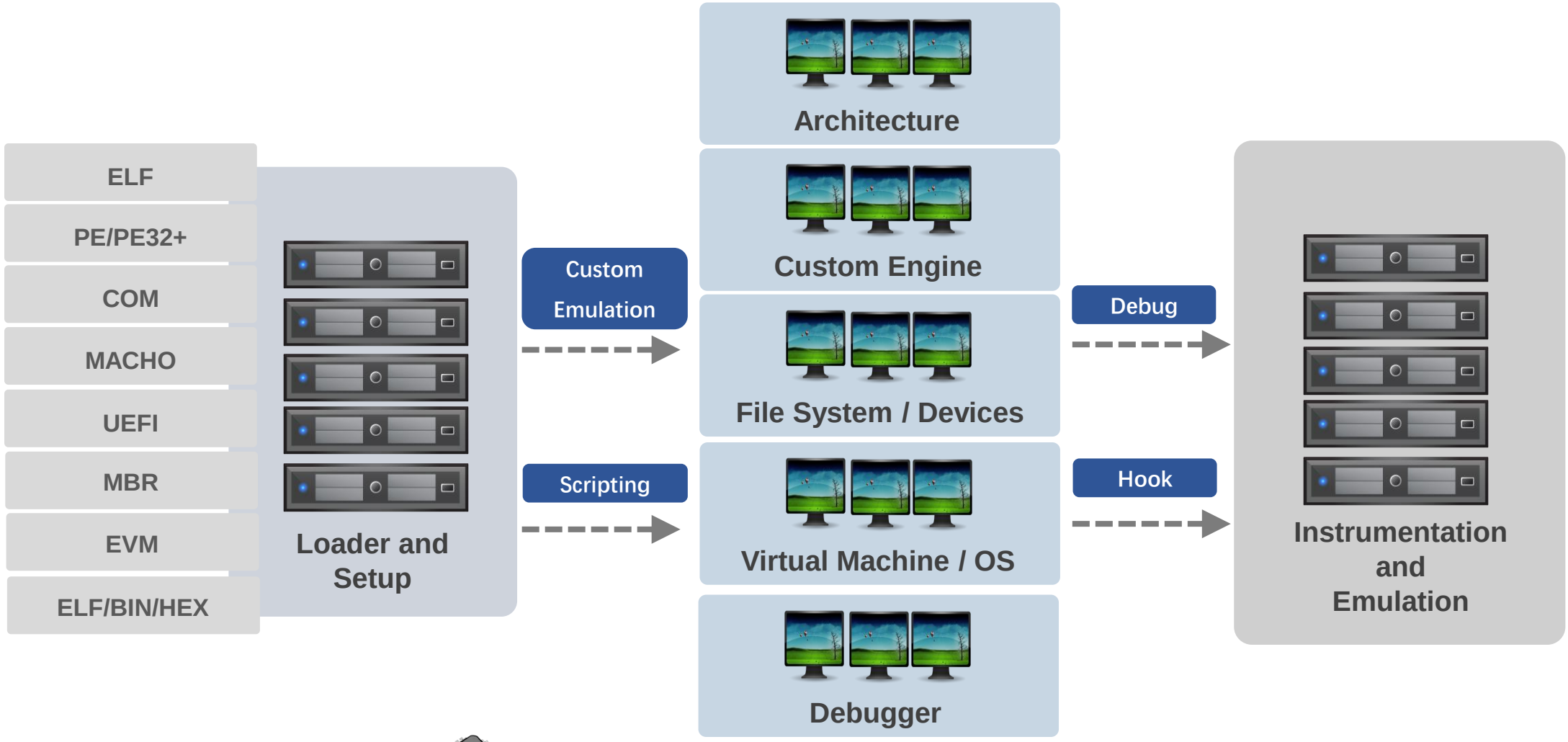
Tools built on top of Qiling Framework

Instrumentation


Binary Execution Flow
A
B
C
D
E
F
Qiling's Instrumentation
A
Start
B
C
D
End
E
F
Alter
Memory
Alter
function()
Alter
CPU Register

Qiling Framework and its Mode

Qiling Framework: The 3 Mode



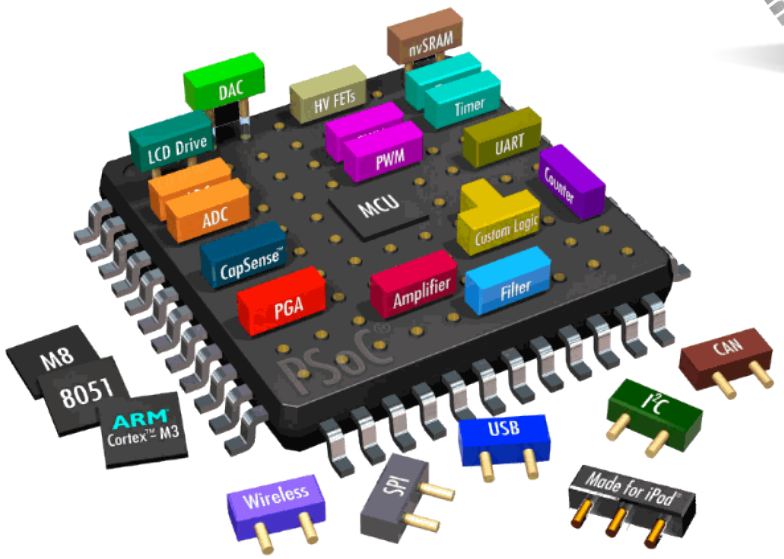
Hardware Mode

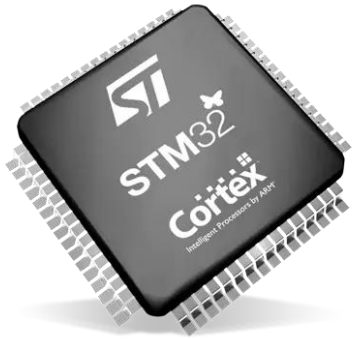
EVM Mode

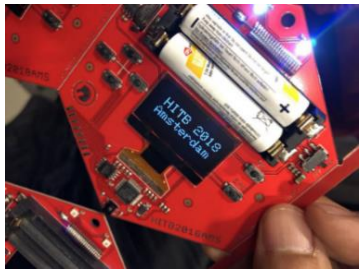
Software Mode

Hardware Mode

MCU vs SoC

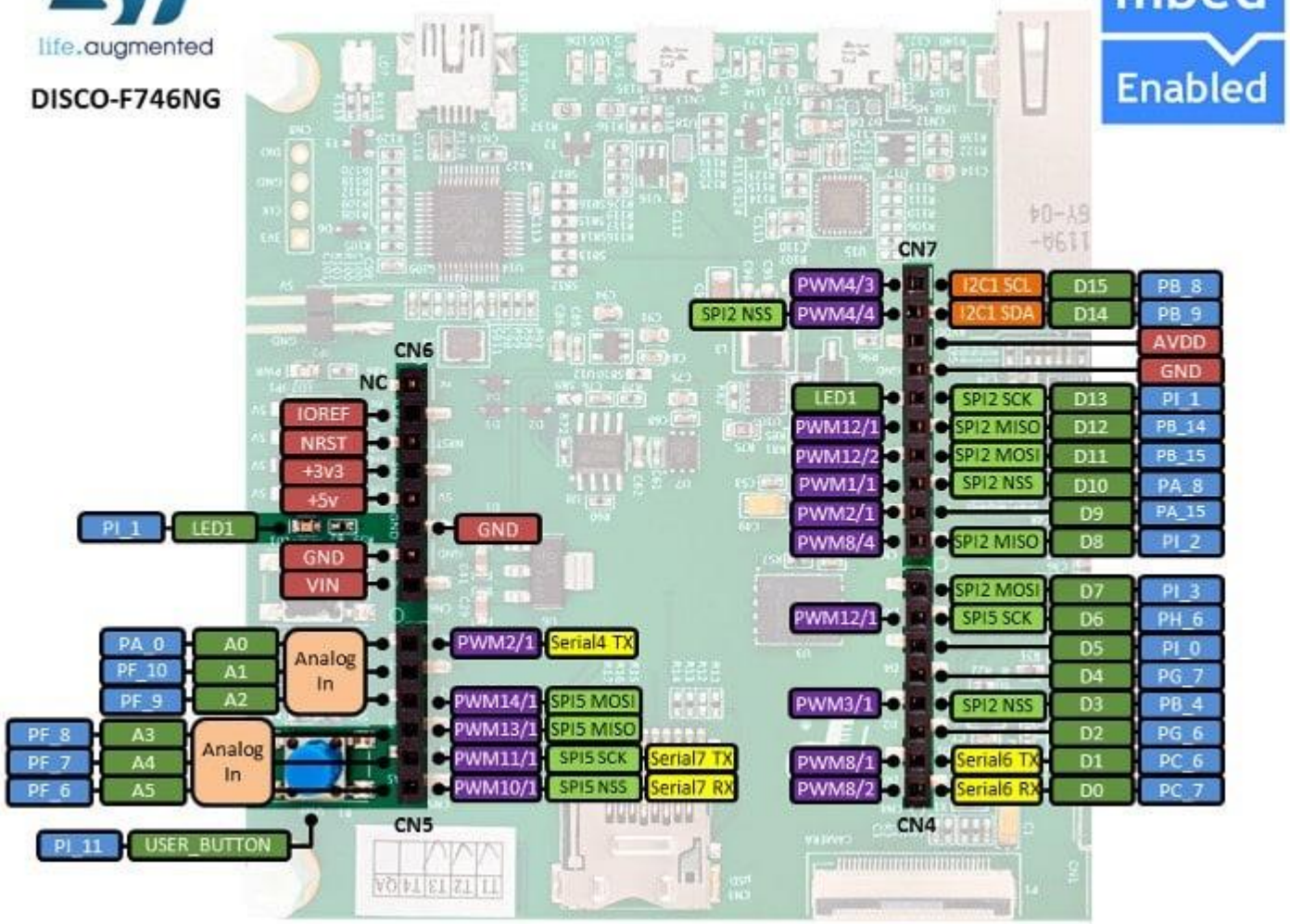

nvSRAM
DAC
HV FETs
Timer
LCD Drive
PWM
UART
ADC
MCU
Counter
Custom Logic
CapSense
PGA
Amplifier
Filter
M8
8051
ARM
Cortex M3
CAN
I²C
USB
Wireless
SPI
Made for iPod
PSoC


STM32
Cortex


HITB 2018
Amsterdam

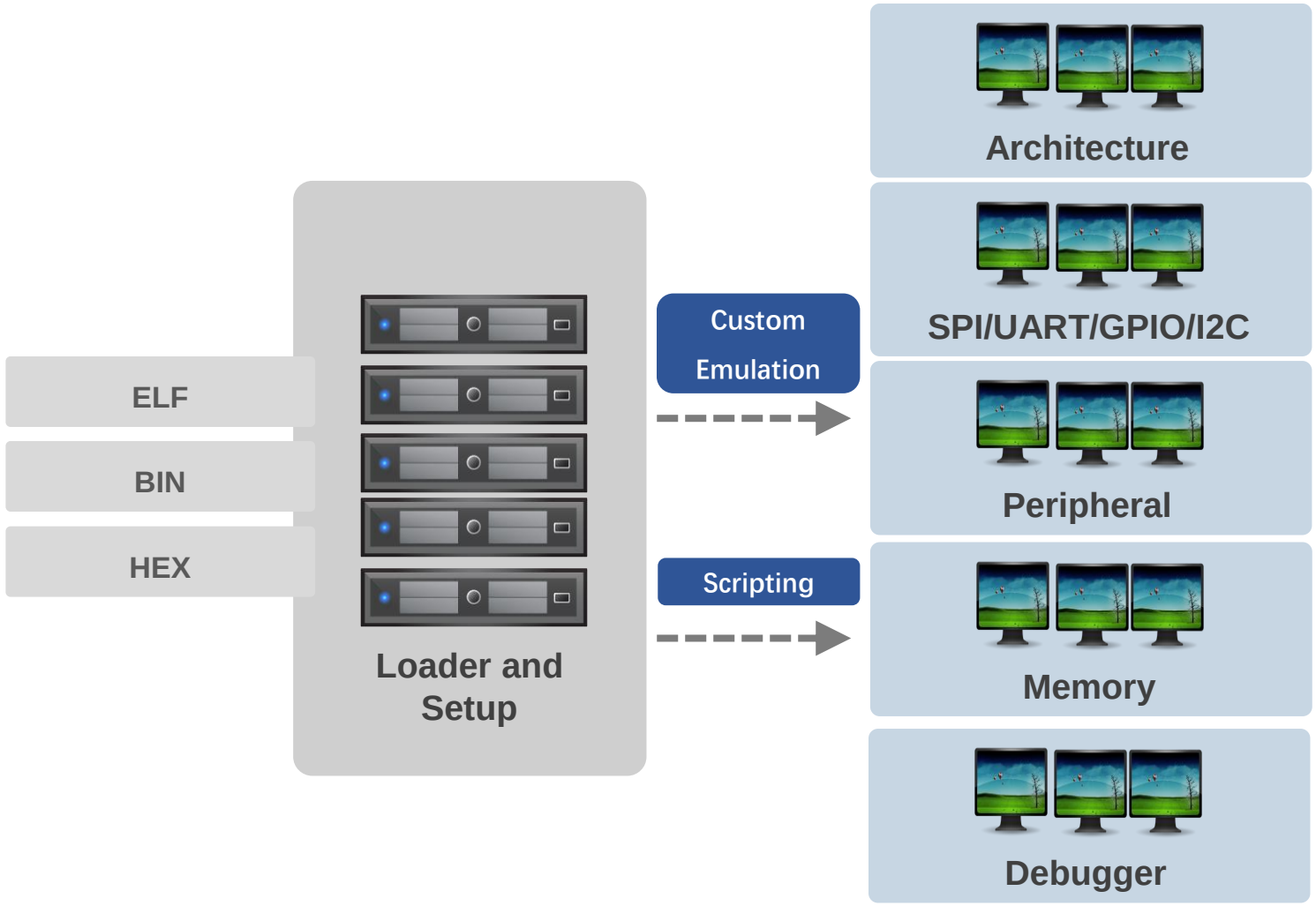

THE SHEPHERD LAB
The Shepherd Lab


HITB2018
HITB2018PEK

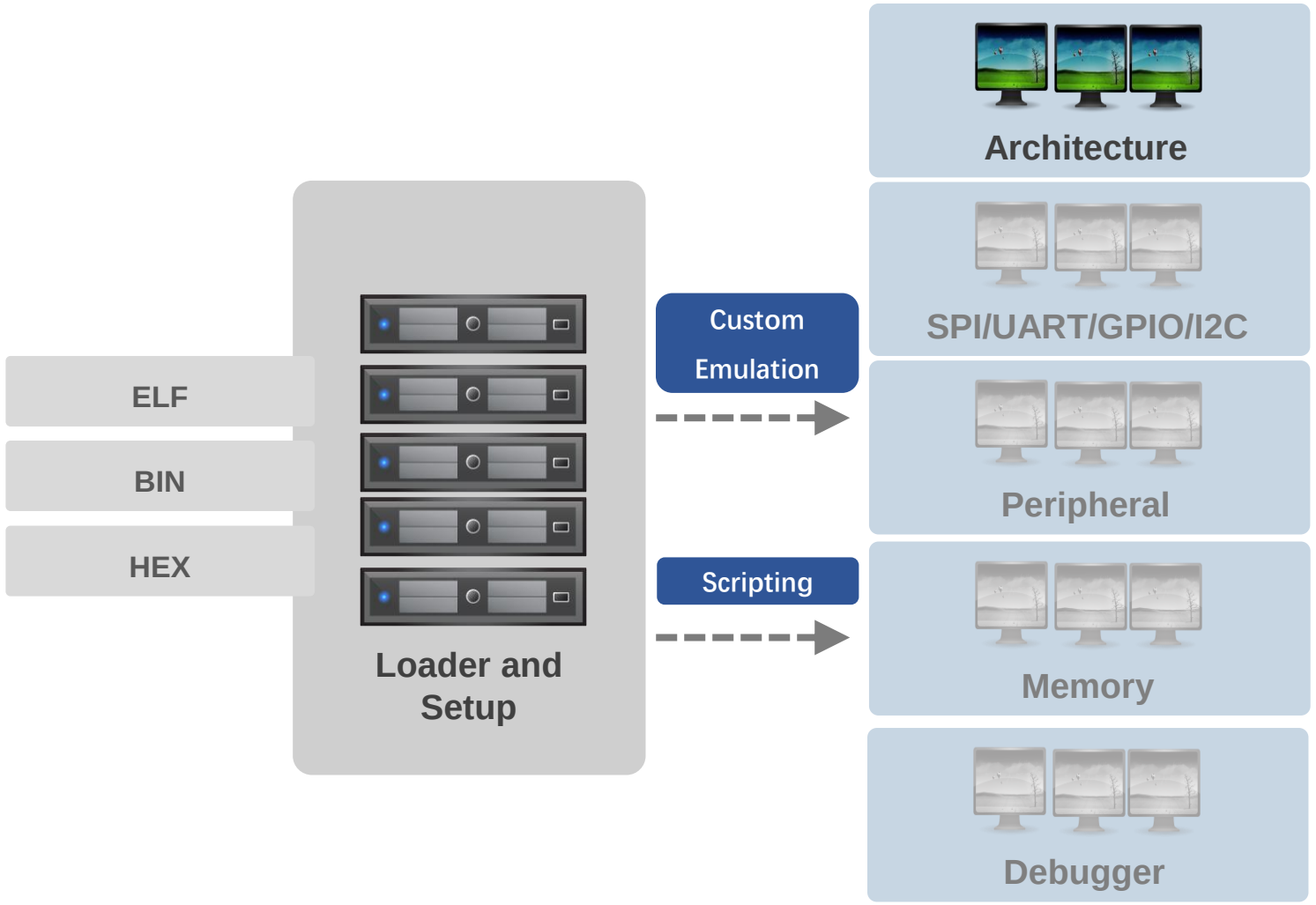

life.augmented
DISCO-F746NG
mbed Enabled
CN6
CN7
CN5
CN4
NC
IOREF
NRST
+3v3
+5v
GND
VIN
PI_1
LED1
PA_0
PF_10
PF_9
PF_8
PF_7
PF_6
PI_11
USER_BUTTON
A0
A1
A2
A3
A4
A5
Analog In
Analog In
PWM2/1
Serial4 TX
PWM14/1
PWM13/1
PWM11/1
PWM10/1
SPI5 MOSI
SPI5 MISO
SPI5 SCK
SPI5 NSS
Serial7 TX
Serial7 RX
PWM4/3
PWM4/4
SPI2 NSS
I2C1 SCL
I2C1 SDA
D15
D14
PB_8
PB_9
AVDD
GND
LED1
PWM12/1
PWM12/2
PWM1/1
PWM2/1
PWM8/4
SPI2 SCK
SPI2 MISO
SPI2 MOSI
SPI2 NSS
SPI2 MISO
D13
D12
D11
D10
D9
D8
PI_1
PB_14
PB_15
PA_8
PA_15
PI_2
PWM12/1
SPI2 MOSI
SPI5 SCK
D7
D6
D5
D4
D3
D2
D1
D0
PI_3
PH_6
PI_0
PG_7
PB_4
PG_6
PC_6
PC_7
PWM3/1
SPI2 NSS
PWM8/1
PWM8/2
Serial6 TX
Serial6 RX

Hardware Mode and APIs

Qiling Framework Hardware Mode and Its Interface



CPU APIs



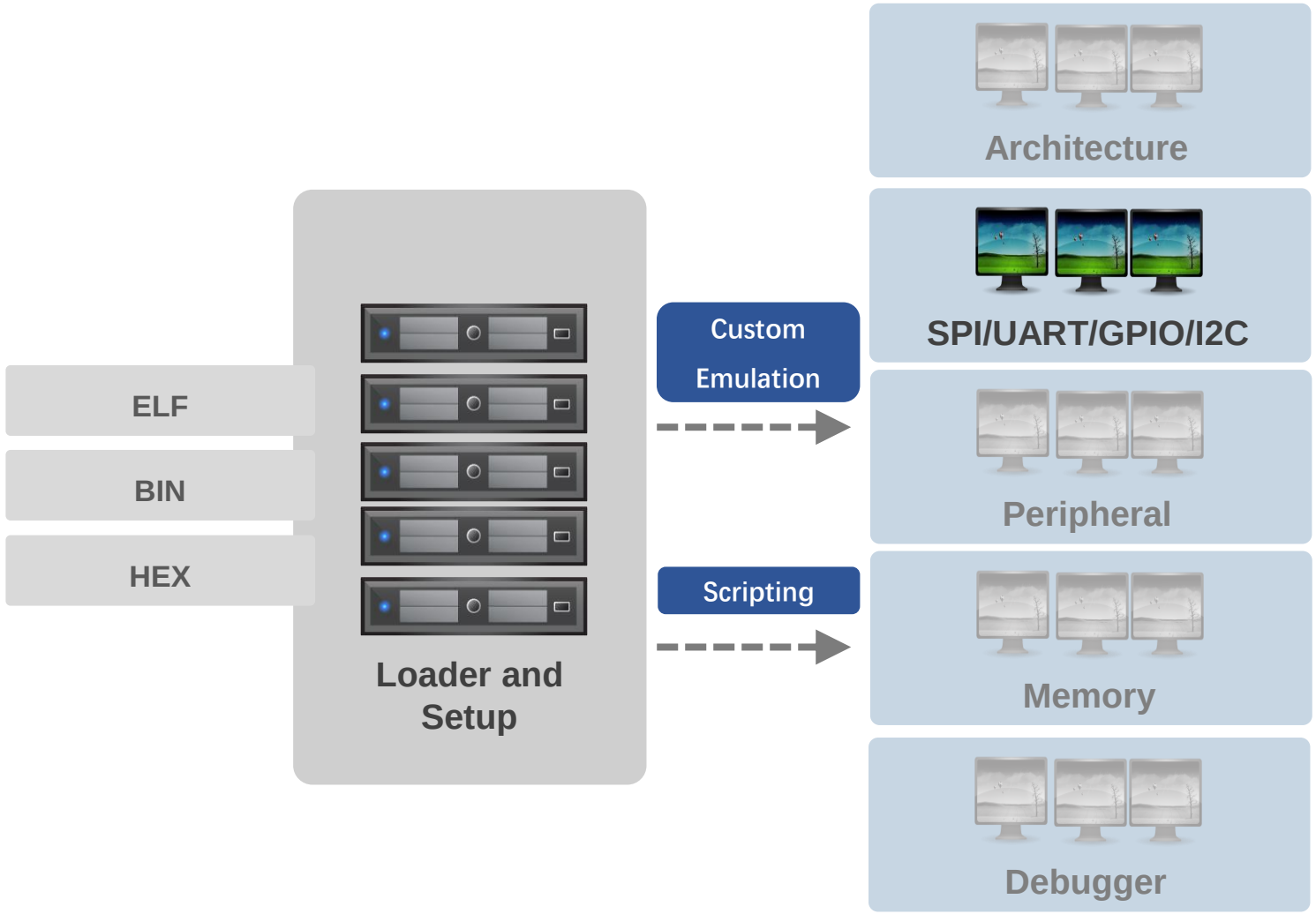
- Access to Register
- Reading register
  - ql.arch.regs.r0
- Writing to register
  - ql.arch.regs.r0 = 0x41
- Different Hooks
  - ql.hook_code()
  - ql.hook_address()
- Interrupt Handle
  - soft_interrupt_handler()
  - hard_interrupt_handler()

```
ql = Qiling(['../rootfs/mcu/gd32vf103/blink.hex'], archtype="riscv64",
                    env=gd32vf103, verbose=QL_VERBOSE.DEBUG)

ql.hw.create('rcu')
ql.hw.create('gpioa').watch()
ql.hw.create('gpioc').watch()

delay_cycles_begin = 0x800015c
delay_cycles_end = 0x800018c

def skip_delay(ql):
    ql.reg.pc = delay_cycles_end

ql.hook_address(skip_delay, delay_cycles_begin)
ql.hw.gpioc.hook_set(13, lambda : print('Set PC13'))

ql.run(count=20000)
```

```
[+]     [gpioa] Set PA1
[=]     [GPIOC] [0x8000250] [R] OCTL = 0x0
[=]     [GPIOC] [0x8000258] [W] OCTL = 0x2000
[+]     [gpioc] Set PC13
Set PC13
[=]     [GPIOA] [0x8000268] [R] OCTL = 0x2
[=]     [GPIOA] [0x8000270] [W] OCTL = 0x6
[+]     [gpioa] Set PA2
[=]     [GPIOA] [0x80001f0] [R] OCTL = 0x6
[=]     [GPIOA] [0x80001f6] [W] OCTL = 0x4
[+]     [gpioa] Reset PA1
[=]     [GPIOC] [0x8000208] [R] OCTL = 0x2000
[=]     [GPIOC] [0x8000212] [W] OCTL = 0x0
[+]     [gpioc] Reset PC13
[=]     [GPIOA] [0x8000222] [R] OCTL = 0x4
[=]     [GPIOA] [0x8000228] [W] OCTL = 0x0
[+]     [gpioa] Reset PA2
[=]     [GPIOA] [0x8000238] [R] OCTL = 0x0
[=]     [GPIOA] [0x8000240] [W] OCTL = 0x2
[+]     [gpioa] Set PA1
[=]     [GPIOC] [0x8000250] [R] OCTL = 0x0
[=]     [GPIOC] [0x8000258] [W] OCTL = 0x2000
[+]     [gpioc] Set PC13
Set PC13
```

- Access to Register
- Reading register
  - ql.arch.regs.r0
- Writing to register
  - ql.arch.regs.r0 = 0x41
- Different Hooks
  - ql.hook_code()
  - ql.hook_address()
- Interrupt Handle
  - soft_interrupt_handler()
  - hard_interrupt_handler()

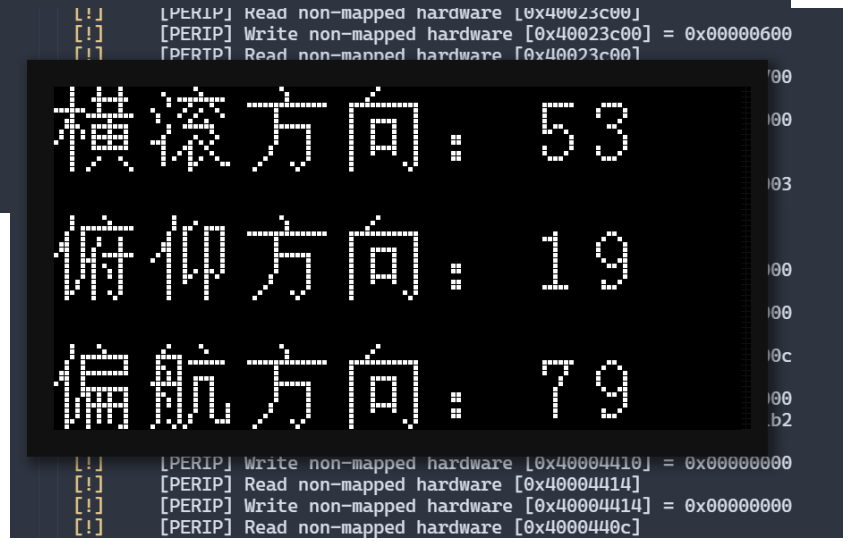
Interface APIs



- SPI
  - connect()
- UART
  - read()
  - write()
  - transfer()
  - step()
- GPIO
  - read()
  - write()
  - pin()
- I2C
  - read()
  - write()
  - send_address()
  - send_data()

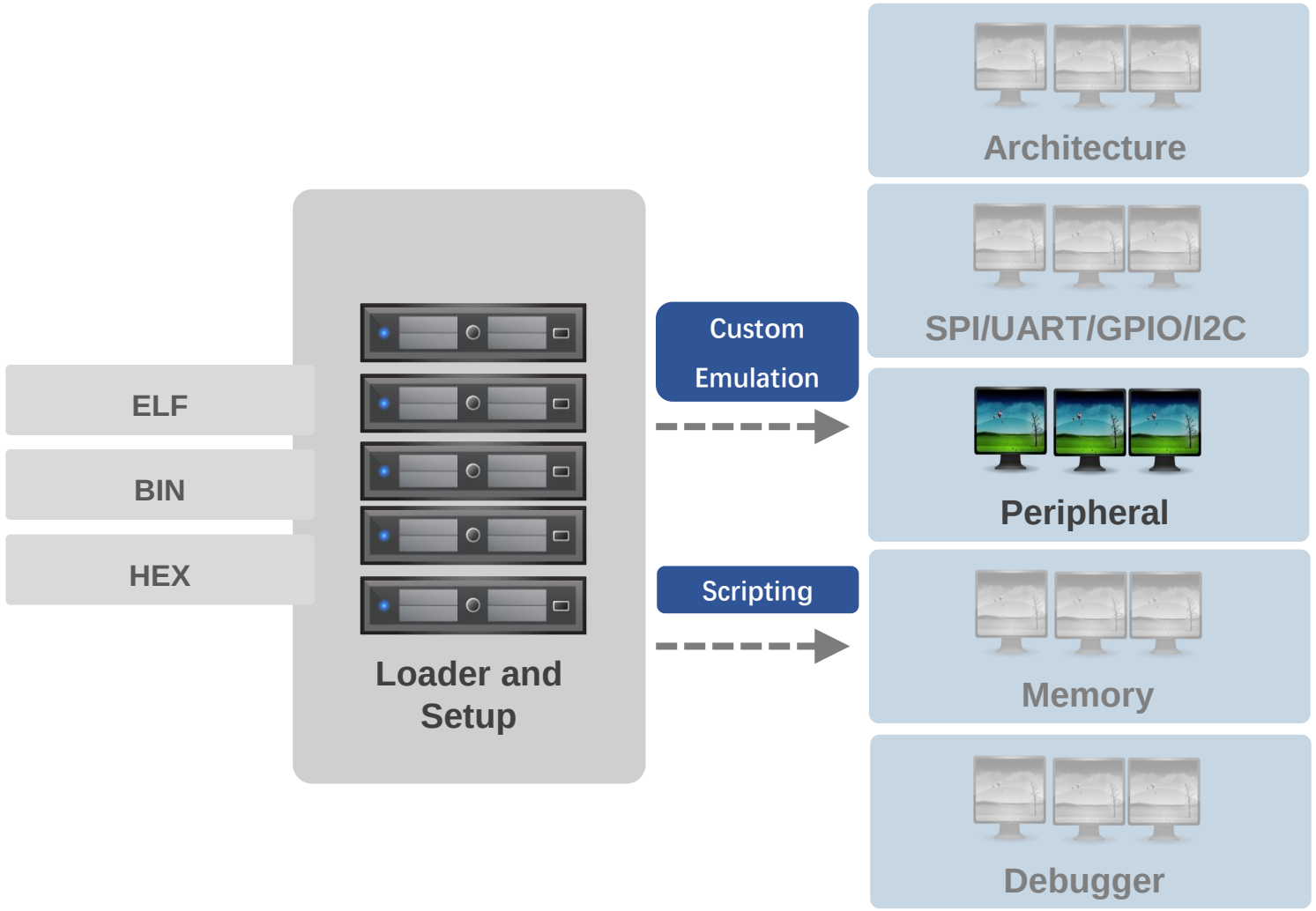
Examples: SPI/UART/GPIO/I2C

```
ql = Qiling(['../rootfs/mcu/stm32f411/oled12864.hex'],
            archtype="cortex_m", env=stm32f411, verbose=QL_VERBOSE.DEFAULT)

ql.hw.create('rcc')
ql.hw.create('gpioa')
ql.hw.create('gpiob')
ql.hw.create('gpioc')
ql.hw.create('spi1')

oled = PyGameSSD1306Spi(dc=(ql.hw.gpioc, 7))

ql.hw.systick.ratio = 2000

ql.hw.spi1.connect(oled)
ql.run(count=1000000)
```



- SPI
  - connect()
- UART
  - read()
  - write()
  - transfer()
  - step()
- GPIO
  - read()
  - write()
  - pin()
- I2C
  - read()
  - write()
  - send_address()
  - send_data()

Peripheral APIs

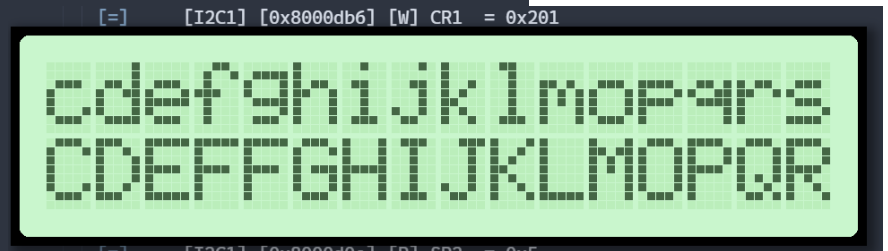


- Peripheral Management
  - step()
  - watch()
  - monitor()
  - recorder()
- Hardware Management
  - create()
  - delete()
  - load_env()
  - find()
- Supported Peripherals
  - UART
  - DMA
  - Power
  - Timer

Examples: Peripheral

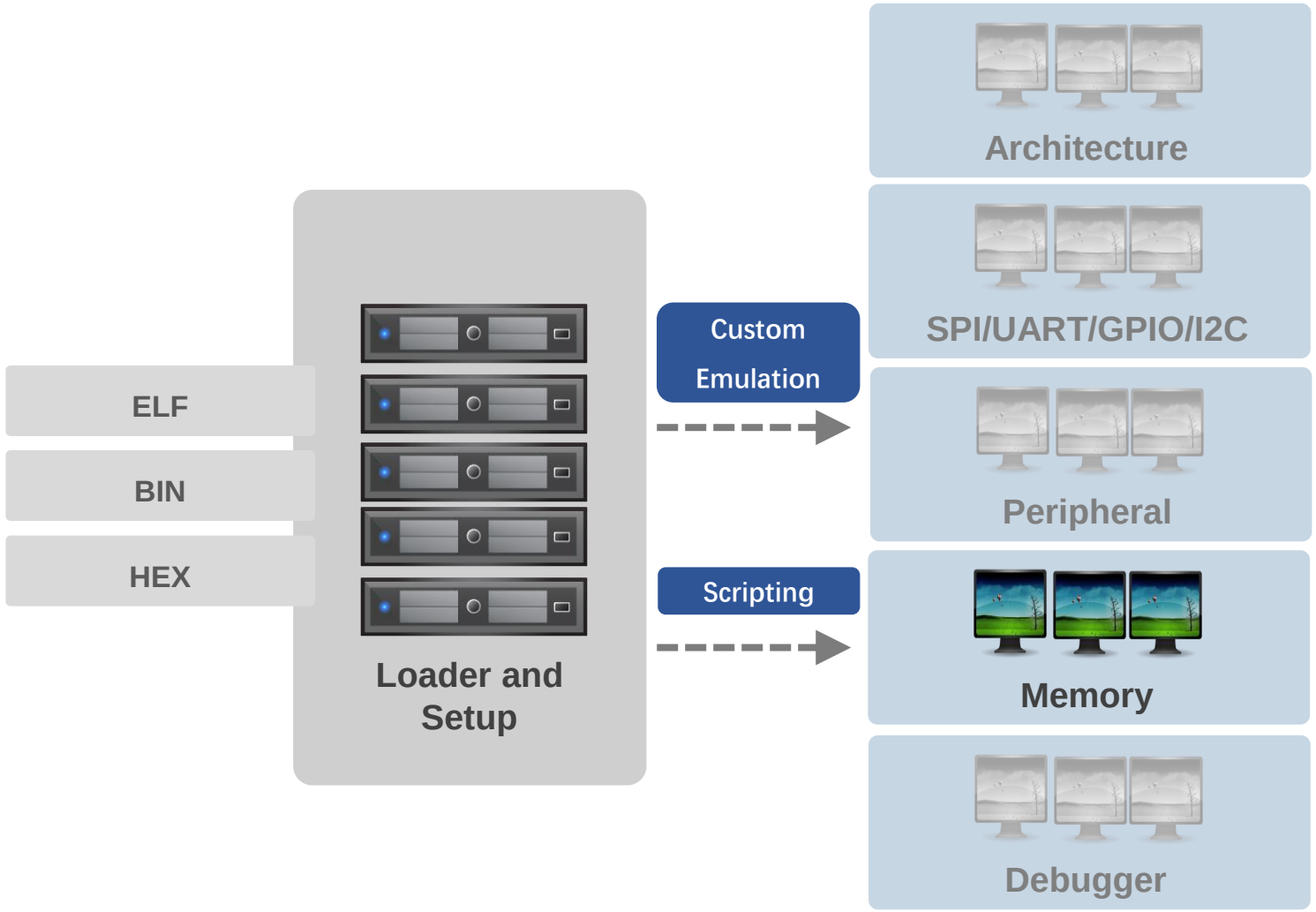
```python
def create(path, lcd):
    ql = Qiling([path], archtype="cortex_m", env=stm32f411, verbose=QL_VERBOSE.DEBUG)

    ql.hw.create('i2c1')
    ql.hw.create('rcc')
    ql.hw.create('gpioa')
    ql.hw.create('gpiob')

    ql.hw.i2c1.watch()
    ql.hw.i2c1.connect(lcd)

    ql.hw.systick.set_ratio(100)

    return ql

if __name__ == "__main__":
    lcd = PyGameLCD1602()

    ## Example 1
    create("../rootfs/mcu/stm32f411/i2c-lcd.hex", lcd).run(count=50000)

    ## Example 2
    create("../rootfs/mcu/stm32f411/lcd-plus.hex", lcd).run(count=100000)

    ## Example 3
    ql = create("../rootfs/mcu/stm32f411/i2cit-lcd.hex", lcd)

    delay_start = 0x8002936
    delay_end = 0x8002955
    def skip_delay(ql):
        ql.reg.pc = delay_end

    ql.hook_address(skip_delay, delay_start)
    ql.run(count=100000)

    lcd.quit()
```

```
[=]     [I2C1] [0x8000db6] [W] CR1  = 0x201
[=]     [I2C1] [0x8000d0e] [R] SR2  = 0x5
[=]     [I2C1] [0x8000a0e] [R] SR1  = 0x94
```



- Peripheral Management
  - step()
  - watch()
  - monitor()
  - recorder()
- Hardware Management
  - create()
  - delete()
  - load_env()
  - find()
- Supported Peripherals
  - UART
  - DMA
  - Power
  - Timer

Memory APIs

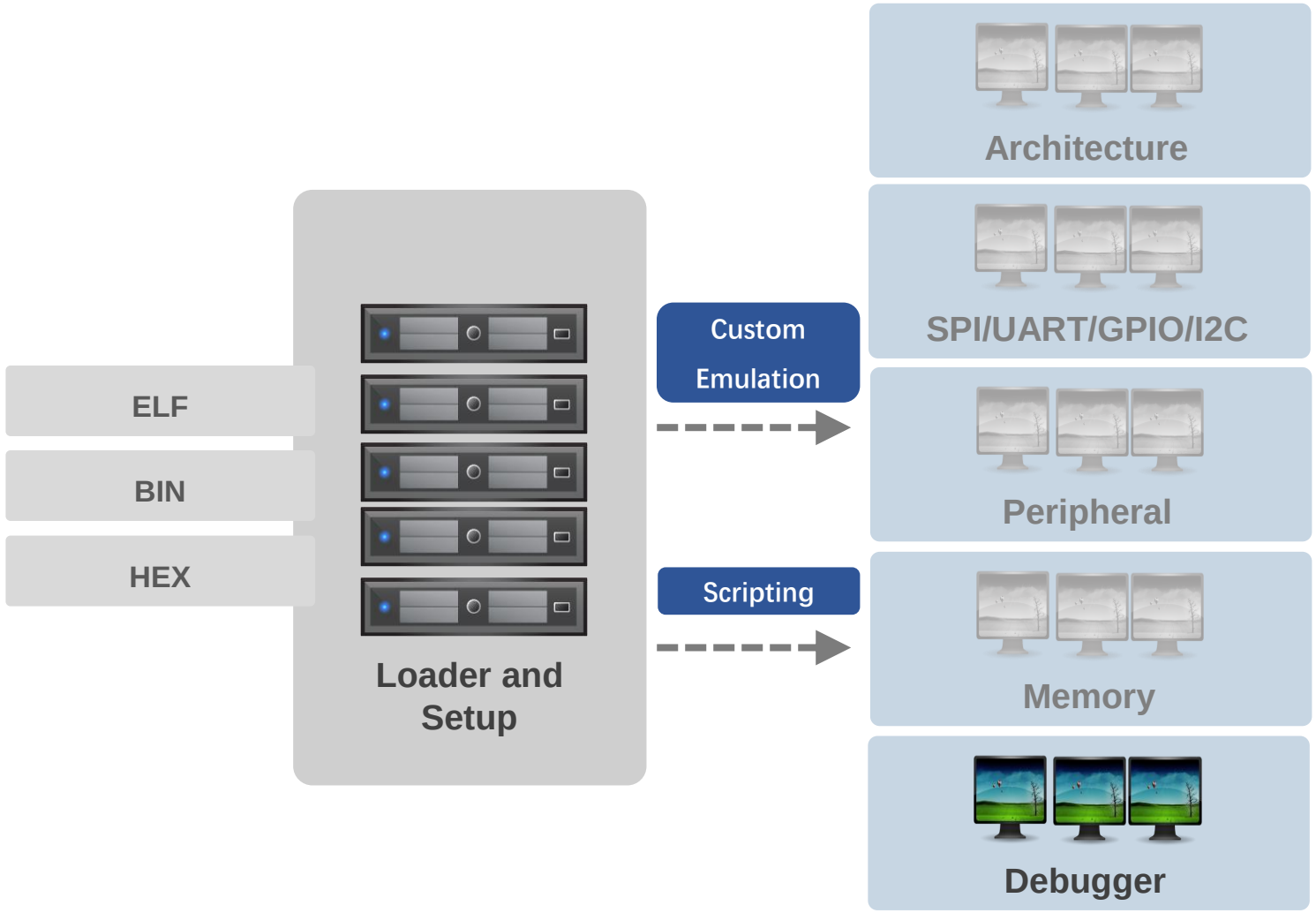


- Memory Management
  - read()
  - write()
  - align()
  - save()
  - restore()
  - search()
  - map()
  - unmap()
  - protect()
- MMIO
  - map_mmio()
- Hijack
  - patch()

Examples: Memory

```
ql = Qiling(["../../examples/rootfs/mcu/stm32f407/backdoorlock.hex"],
                    archtype="cortex_m", env=stm32f407, verbose=QL_VERBOSE.OFF)

ql.hw.create('spi2')
ql.hw.create('gpioe')
ql.hw.create('gpiof')
ql.hw.create('usart1')
ql.hw.create('rcc')

ql.hw.show_info()

print('Testing passwd', passwd)

ql.patch(0x8000238, b'\x00\xBF' * 4)
ql.patch(0x80031e4, b'\x00\xBF' * 11)
ql.patch(0x80032f8, b'\x00\xBF' * 13)
ql.patch(0x80013b8, b'\x00\xBF' * 10)

ql.hw.usart1.send(passwd.encode() + b'\r')

ql.hw.systick.set_ratio(100)
ql.run(count=1000000, end=0x8003225)
if ql.arch.get_pc() == 0x8003225:
    print('Success, the passwd is', passwd)
else:
    print('Fail, the passwd is not', passwd)
```

- Memory Management
  - read()
  - write()
  - align()
  - save()
  - restore()
  - search()
  - map()
  - unmap()
  - protect()
- MMIO
  - map_mmio()
- Hijack
  - patch()



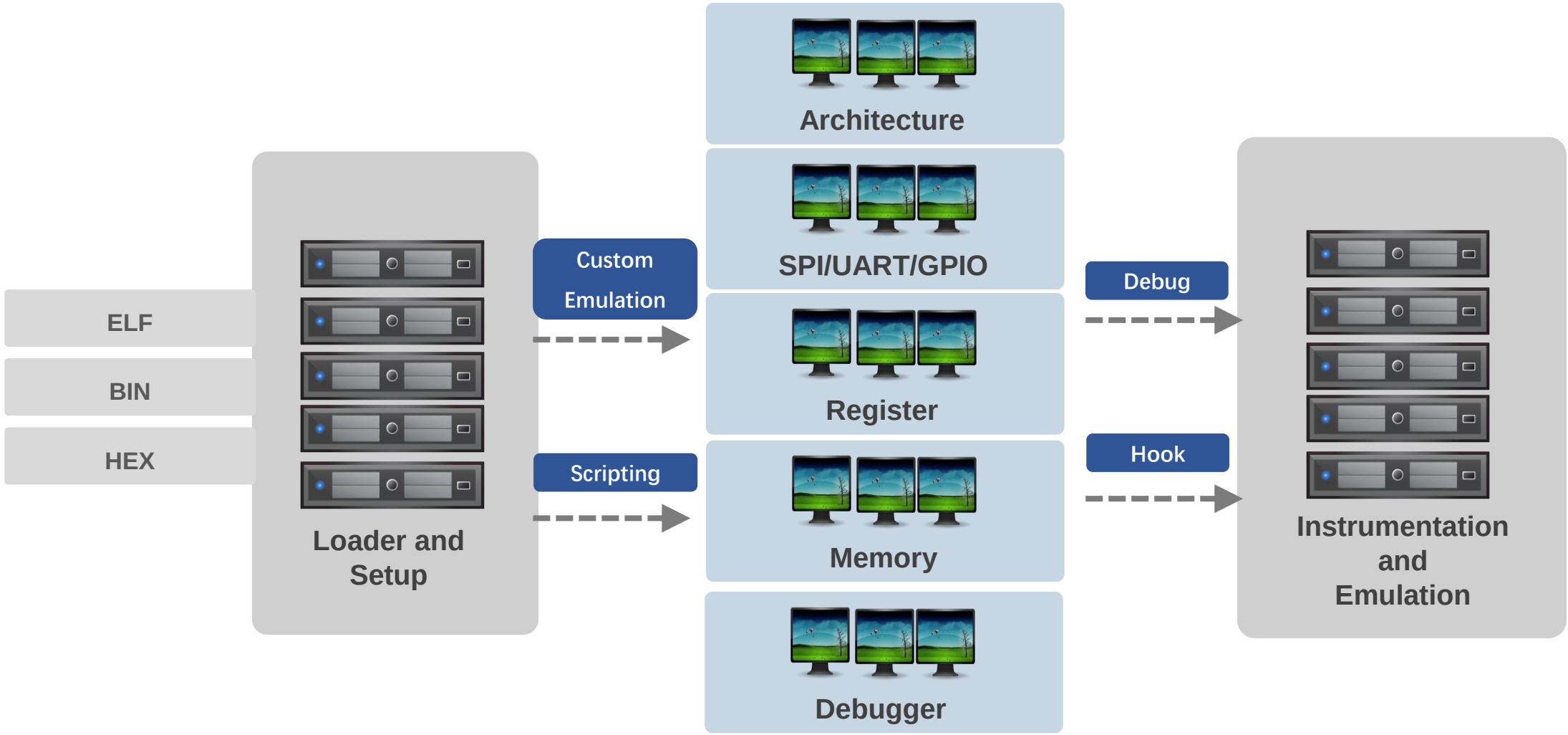
- Disassembly
  - create_disassembler()
- Debug
  - step()
  - stop()
  - run()

Examples: Debugger Level

```
[=]     0800193e [[FLASH]          + 0x00193e]  03 f8 01 1b          strb        r1, [r3], #1
[=]     08001942 [[FLASH]          + 0x001942]  f9 e7                b           #0x8001938
[=]     08001938 [[FLASH]          + 0x001938]  93 42                cmp         r3, r2
[=]     0800193a [[FLASH]          + 0x00193a]  00 d1                bne         #0x800193e
[=]     0800193e [[FLASH]          + 0x00193e]  03 f8 01 1b          strb        r1, [r3], #1
[=]     08001942 [[FLASH]          + 0x001942]  f9 e7                b           #0x8001938
[=]     08001938 [[FLASH]          + 0x001938]  93 42                cmp         r3, r2
[=]     0800193a [[FLASH]          + 0x00193a]  00 d1                bne         #0x800193e
[=]     0800193e [[FLASH]          + 0x00193e]  03 f8 01 1b          strb        r1, [r3], #1
[=]     08001942 [[FLASH]          + 0x001942]  f9 e7                b           #0x8001938
[=]     08001938 [[FLASH]          + 0x001938]  93 42                cmp         r3, r2
[=]     0800193a [[FLASH]          + 0x00193a]  00 d1                bne         #0x800193e
[=]     0800193e [[FLASH]          + 0x00193e]  03 f8 01 1b          strb        r1, [r3], #1
[=]     08001942 [[FLASH]          + 0x001942]  f9 e7                b           #0x8001938
[=]     08001938 [[FLASH]          + 0x001938]  93 42                cmp         r3, r2
[=]     0800193a [[FLASH]          + 0x00193a]  00 d1                bne         #0x800193e
[=]     0800193e [[FLASH]          + 0x00193e]  03 f8 01 1b          strb        r1, [r3], #1
[=]     08001942 [[FLASH]          + 0x001942]  f9 e7                b           #0x8001938
[=]     08001938 [[FLASH]          + 0x001938]  93 42                cmp         r3, r2
[=]     0800193a [[FLASH]          + 0x00193a]  00 d1                bne         #0x800193e
[=]     0800193e [[FLASH]          + 0x00193e]  03 f8 01 1b          strb        r1, [r3], #1
[=]     08001942 [[FLASH]          + 0x001942]  f9 e7                b           #0x8001938
[=]     08001938 [[FLASH]          + 0x001938]  93 42                cmp         r3, r2
[=]     0800193a [[FLASH]          + 0x00193a]  00 d1                bne         #0x800193e
[=]     0800193e [[FLASH]          + 0x00193e]  03 f8 01 1b          strb        r1, [r3], #1
[=]     08001942 [[FLASH]          + 0x001942]  f9 e7                b           #0x8001938
[=]     08001938 [[FLASH]          + 0x001938]  93 42                cmp         r3, r2
[=]     0800193a [[FLASH]          + 0x00193a]  00 d1                bne         #0x800193e
```

- Disassembly
  - create_disassembler()
- Debug
  - step()
  - stop()
  - run()

Qiling Framework Hardware Mode



DEMO

Examples: Emulate RTOS

```python
import sys
sys.path.append("../..")

from qiling.core import Qiling
from qiling.const import QL_VERBOSE
from qiling.extensions.mcu.stm32f4 import stm32f411


def stm32f411_freertos():
    ql = Qiling(["../rootfs/mcu/stm32f411/os-demo.hex"],
        archtype="cortex_m", env=stm32f411, verbose=QL_VERBOSE.DEBUG)

    ql.hw.create('usart2').watch()
    ql.hw.create('gpioa').watch()
    ql.hw.create('rcc')

    ql.hw.systick.set_ratio(100)
    ql.run(count=200000)

if __name__ == "__main__":
    stm32f411_freertos()
```

```
[!]     [PPB] Write non-mapped hardware [0xe000ef34] = 0xc0000000
[+]     Received interrupt: 0x2
[+]     Received interrupt: 0x8
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[=]     [USART2] [0x8001372] [W] DR   = 0x46 ('F')
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[=]     [USART2] [0x8001372] [W] DR   = 0x72 ('r')
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[+]     Received interrupt: 0x8
[+]     Received interrupt: 0x8
[=]     [GPIOA] [0x8000f5e] [W] BSRR = 0x20
[+]     [gpioa] Set PA5
[+]     Received interrupt: 0x8
[+]     Received interrupt: 0x8
[+]     Received interrupt: 0x8
[=]     [GPIOA] [0x8000f64] [W] BSRR = 0x200000
[+]     [gpioa] Reset PA5
[+]     Received interrupt: 0x8
[+]     Received interrupt: 0x8
[=]     [USART2] [0x8001372] [W] DR   = 0x65 ('e')
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[=]     [USART2] [0x8001372] [W] DR   = 0x65 ('e')
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[=]     [USART2] [0x8001372] [W] DR   = 0x20 (' ')
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[=]     [USART2] [0x8001372] [W] DR   = 0x52 ('R')
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[=]     [USART2] [0x8001372] [W] DR   = 0x54 ('T')
[+]     Received interrupt: 0x8
[+]     Received interrupt: 0x8
[+]     Received interrupt: 0x8
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[=]     [USART2] [0x8001372] [W] DR   = 0x4f ('O')
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[=]     [USART2] [0x8001372] [W] DR   = 0x53 ('S')
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[=]     [USART2] [0x8001372] [W] DR   = 0xa ('\n')
[=]     [USART2] [0x800121e] [R] SR   = 0xc0
[+]     Received interrupt: 0x8
[+]     Received interrupt: 0x8
[=]     [GPIOA] [0x8000f5e] [W] BSRR = 0x20
```

Next

Roadmap 2022-2023

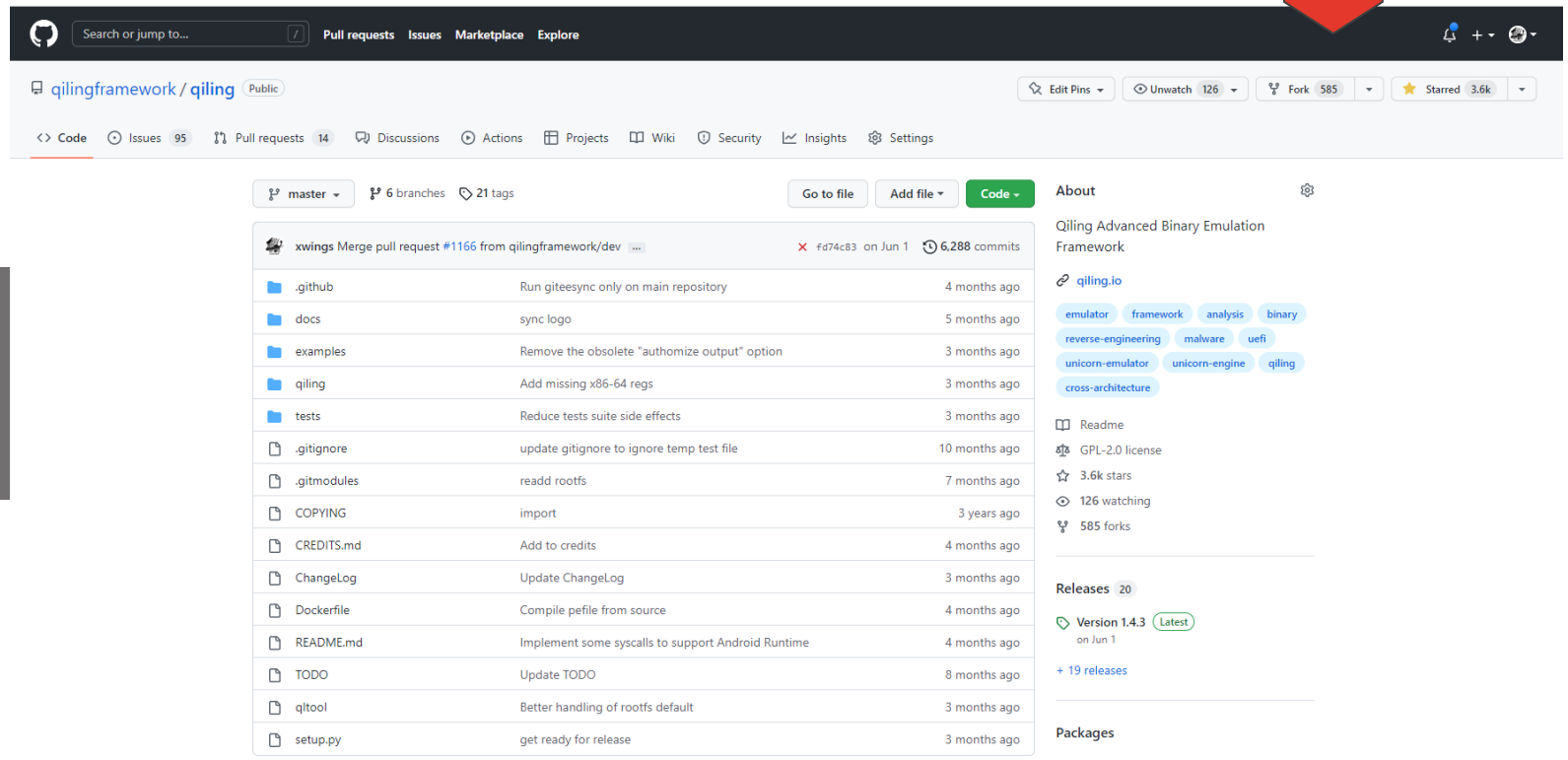
- Android Java bytecode layer instrumentation
- Forward to host implementation
- iPhoneOS/MacOS/M1 emulation support
- More robust Windows emulation
  - Introduce wine && Cygwin or something
- ~~Smart Contract emulation (EVM)~~
- ~~MCU emulation~~

- **To move on further, We are looking for VC / Investors**

Opensource Security Framework Since 2013

- About Qiling Framework
  - https://qiling.io
  - https://github.com/qilingframework/qiling
  - https://docs.qiling.io
  - http://t.me/qilingframework
  - @qiling_io

Star us



Questions