Qiling Framework: Introduction

2021

twitter: @qiling_io https://qiling.io

KaiJern Lau, kj -at- qiling.io

About xwings

JD.COM

Beijing, Stays in the lab 24/7 by hoping making the world a better place

- IoT Research
- Blockchain Research
- Fun Security Research

Qiling Framework

Cross platform and multi architecture advanced binary emulation framework

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

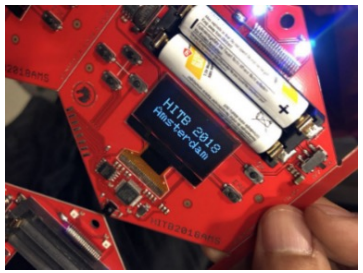
Badge Maker

HACKERSBADGE.COM

Electronic fan boy, making toys from hacker to hacker

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

Badge Designer for Hacking Conferences



Some Recent Talk (Partial)

- 2016, Qcon, Beijing, Speaker, nRF24L01 Hijacking
- 2016, Kcon, Beijing, Speaker, Capstone Unicorn Keystone
- 2017, Kcon, Beijing, IoT Hacking Trainer
- 2018, Kcon, Beijing, IoT Hacking Trainer
- 2018, Brucon, Brussel, Speaker, IoT Virtualization
- 2018, H2HC, San Paolo, Speaker, IoT Virtualization
- 2018, HITB, Beijing/Dubai, Speaker, IoT Virtualization
- 2018, beVX, Hong Kong, Speaker, HackCUBE - Hardware Hacking
- 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

Qiling Framework

- Cross platform and cross architecture binary instrumentation framework
- Emulate and instrument ARM, ARM64, MIPS, X86 and X8664
- Emulate and instrument Linux, MacOS, Windows and FreeBSD
- High-level Python API access to register, CPU and memory
- 2,500+ Github star, more than 13,000+ pypi download, 70+ contributors worldwide

About lazymio && kabeor && dataisland

~ $ whoami
Lazymio

~ $ file Lazymio
The sheperd lab, JD security, Security Engineer.
CTF player, member of Lancet.
GeekPwn 2019 Hall of Fame.

~ $ ls -l Lazymio
Reverse engineering.
Binary analysis.
Writing code for fun.

~ $ which Lazymio
Github: https://github.com/wtdcode
Blog: https://blog.lazym.io/
Twitter: https://twitter.com/pwnedmio

Name: kabeor

Security Engineer at The Shepherd Lab, JD Security.

Core developer of Qiling.

BlackHat Asia & Europe 2020 - Speaker

China kanxue SDC 2020 - Speaker

HITB Training 2020 - Speaker

Github: https://github.com/kabeor

Blog: https://kabeor.cn

Twitter: https://twitter.com/Angrz3_K

This is dataisland.👋

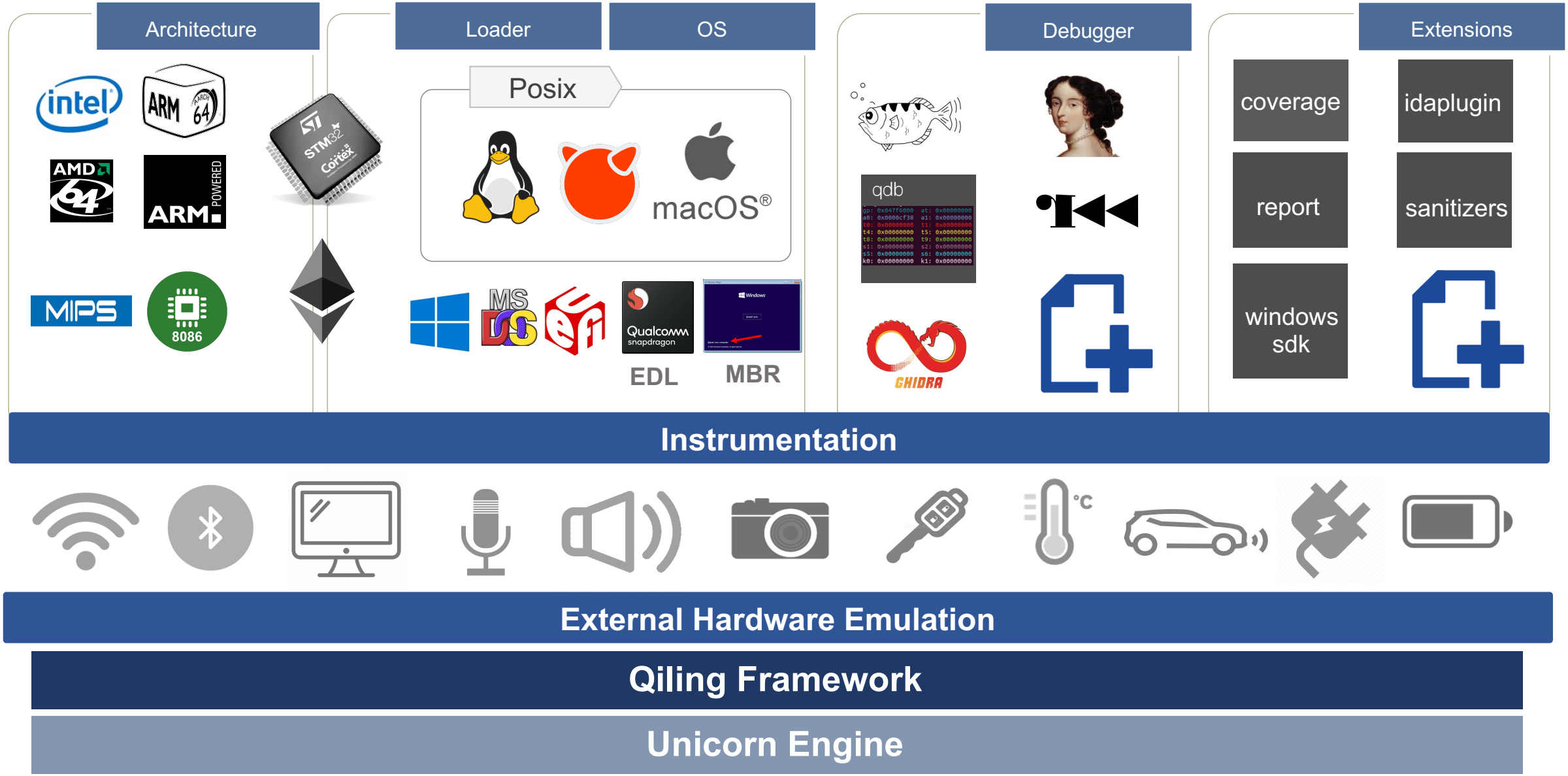
Security Engineer, Qiling Core Developer, CTF Player.

Research:
firmware, compiler
binary analysis
reverse engineering

https://github.com/cla7aye15I4nd

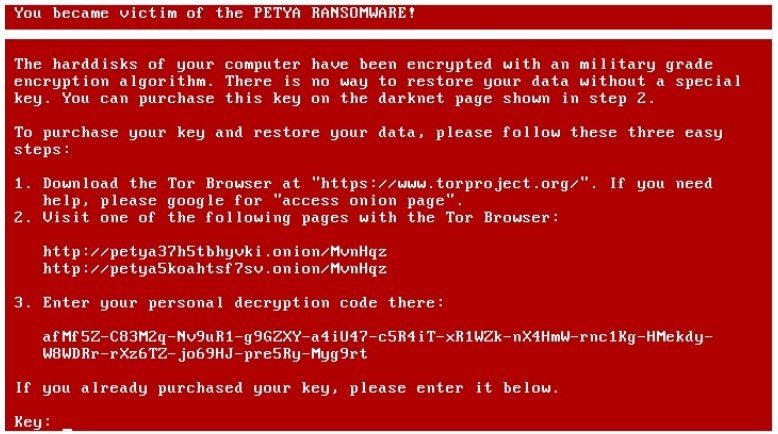
What Is Qiling Framework

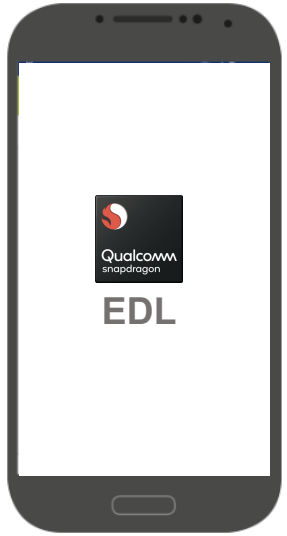
Overview

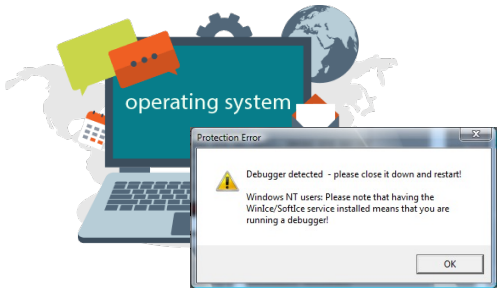

Architecture
Loader
OS
Debugger
Extensions
Posix
macOS®
qdb
EDL
MBR
coverage
idaplugin
report
sanitizers
windows sdk
Instrumentation
External Hardware Emulation
Qiling Framework
Unicorn Engine

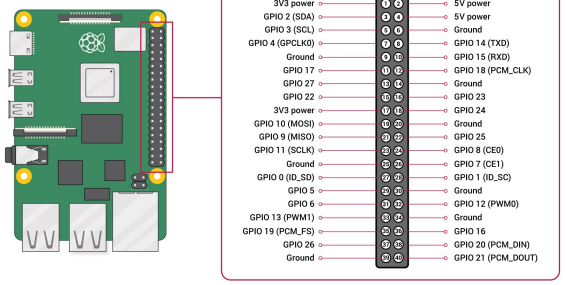
Why Do We Need Qiling Framework

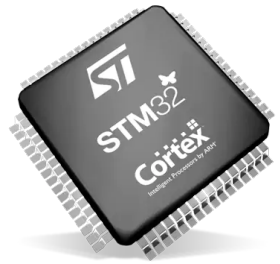
Current Virtual Machine Limitation











- MBR
- UEFI
- Smart Contract
- Micro Computing Unit
- Anti-Anti Debug
- Qualcomm EDL

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

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

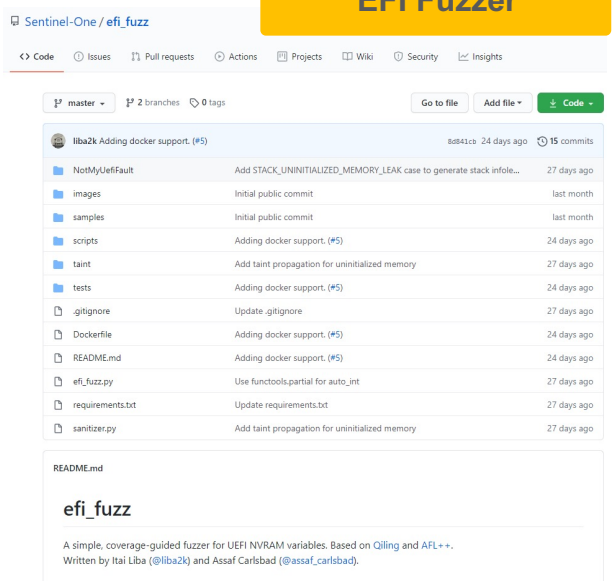
Zelos

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

The Framework

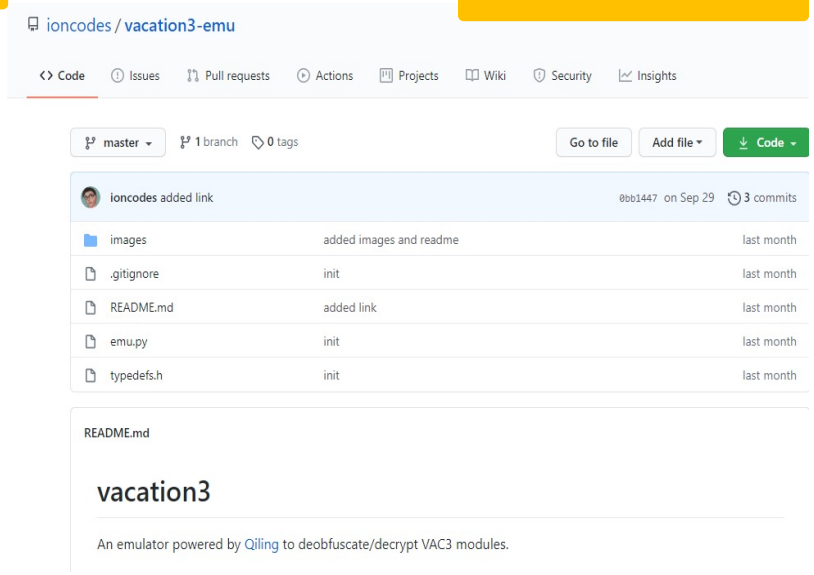
Framework, Not a Tool

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)

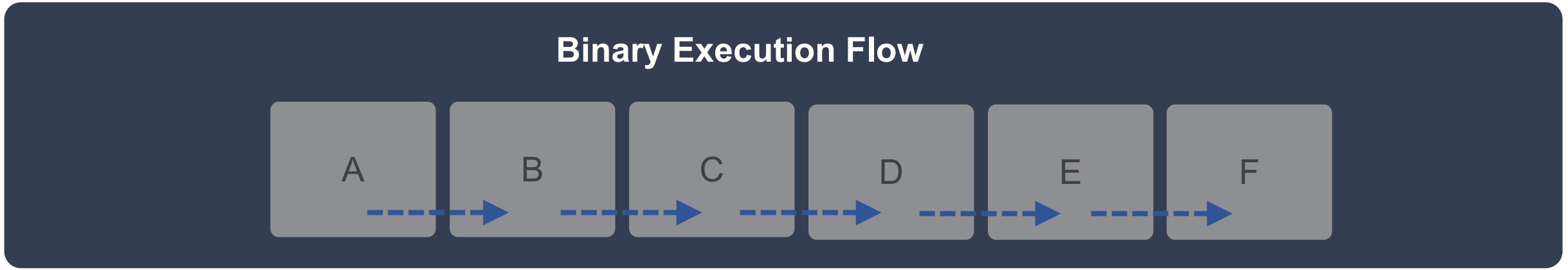
Qiling Framework

CPU Architecture | Loader | OS | Debugger | Extensions

Tools built on top of Qiling Framework

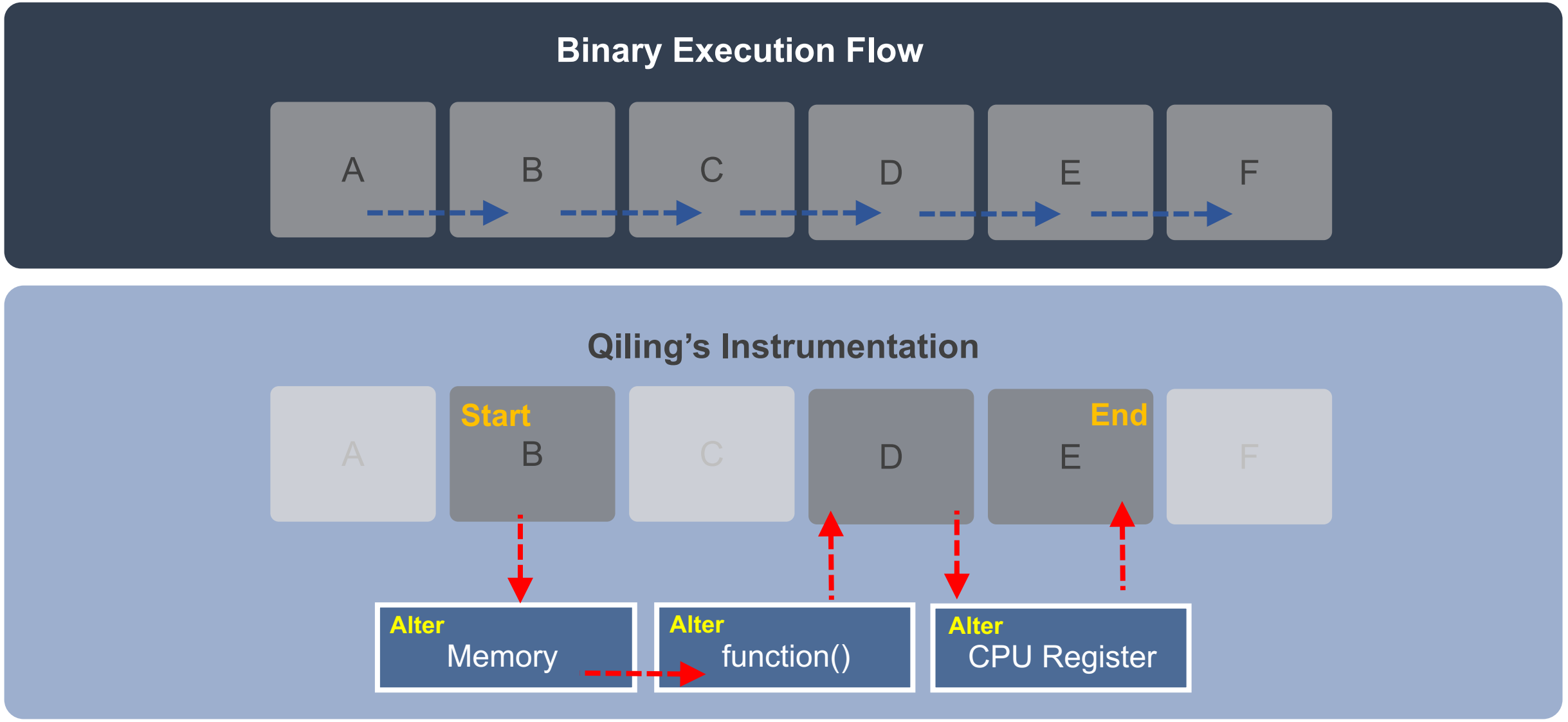
Instrumentation

What Is Instrumentation



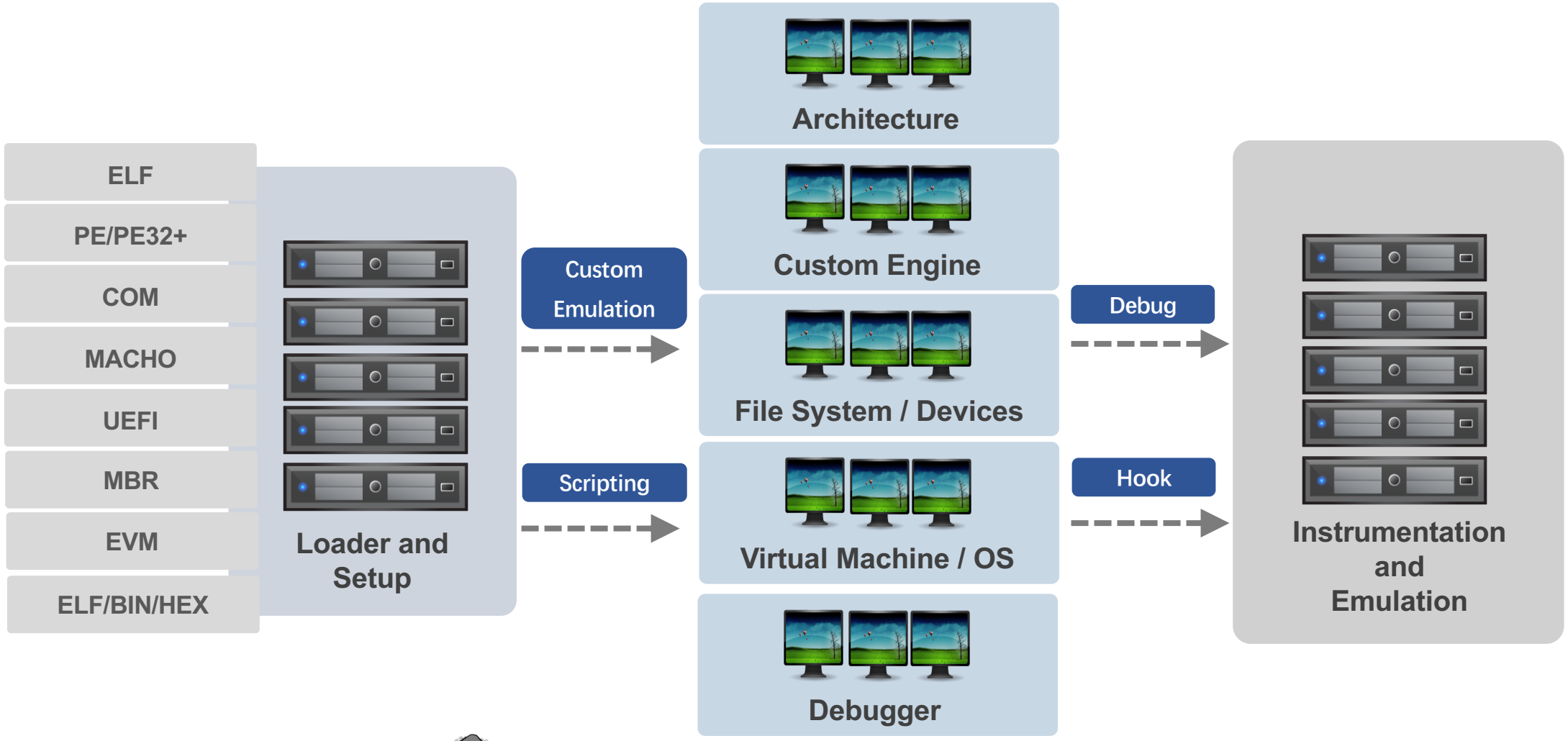
One Function After Another

What Is Instrumentation



Qiling Framework and its Mode

Qiling Framework: The 3 Mode



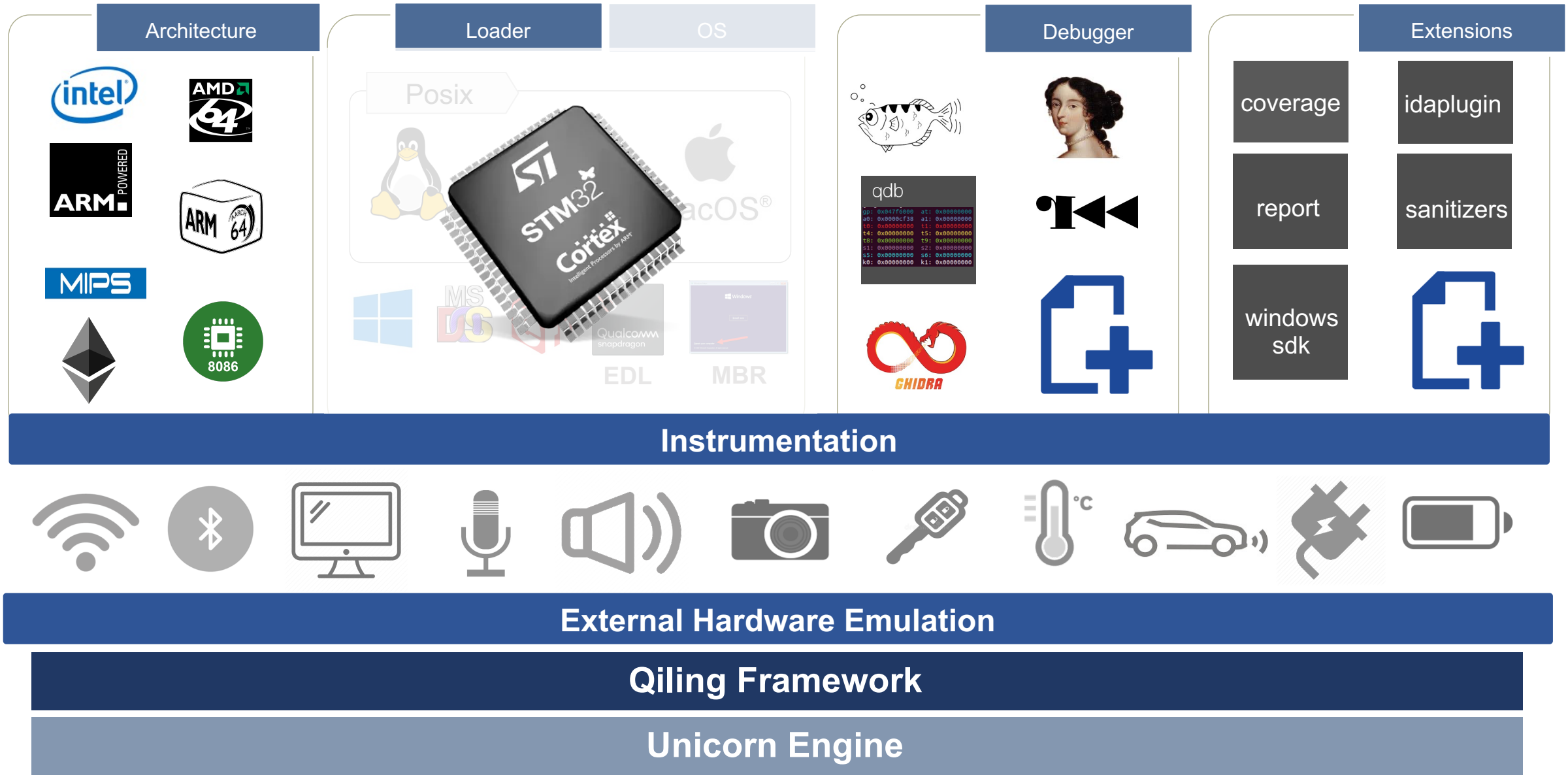
Hardware Mode

EVM Mode

Software Mode

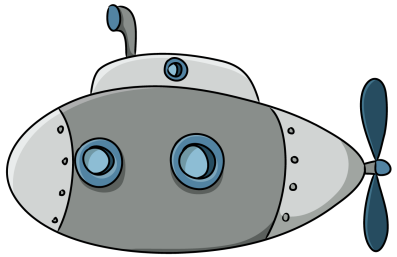
Qiling and Hardware

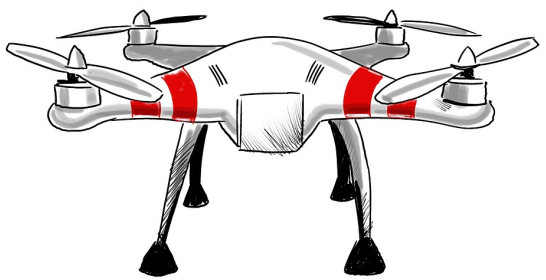
Overview

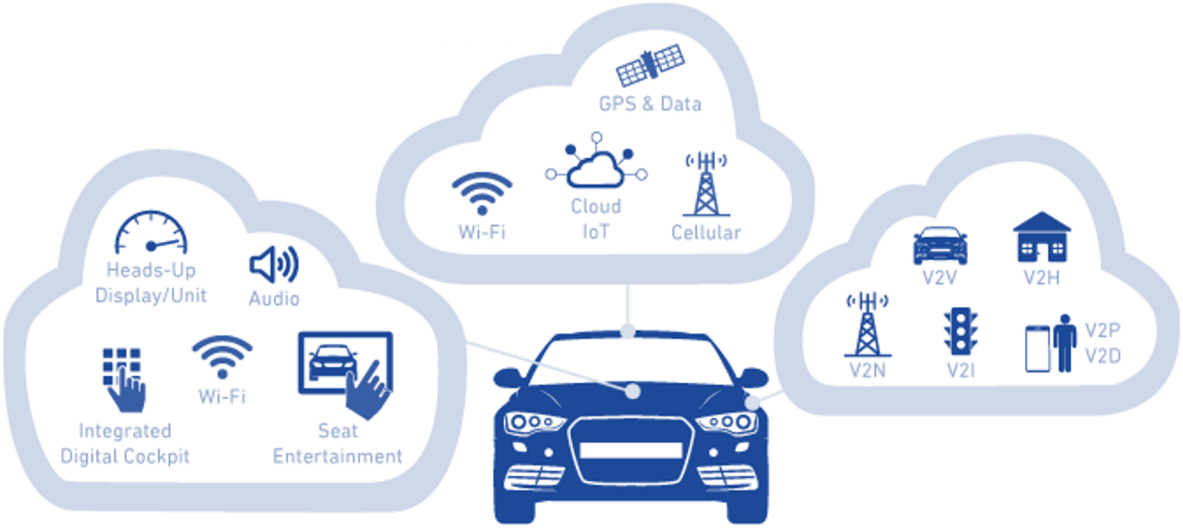

Architecture
Loader
OS
Debugger
Extensions
Posix
STM32
Cortex
EDL
MBR
qdb
GHIDRA
coverage
idaplugin
report
sanitizers
windows sdk
Instrumentation
External Hardware Emulation
Qiling Framework
Unicorn Engine

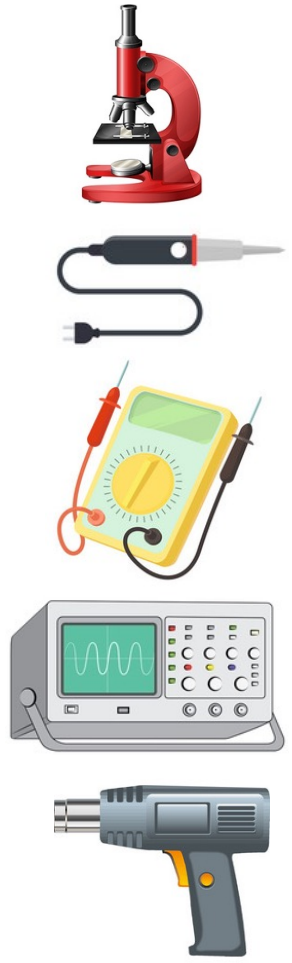
Make IoT Reverse Engineering Great

Today’s IoT Analysis









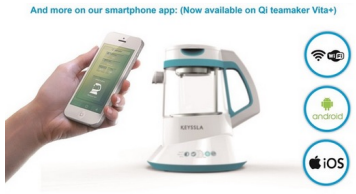
To reverse a firmware

First, you need a IoT, If it not too expensive to own one

Second, you need to purchase expensive equipment (oscilloscopes, probes, soldering iron and tec)

How We Fix It

- GPS
- Wi-Fi
- Bluetooth
- Audio
- Screen



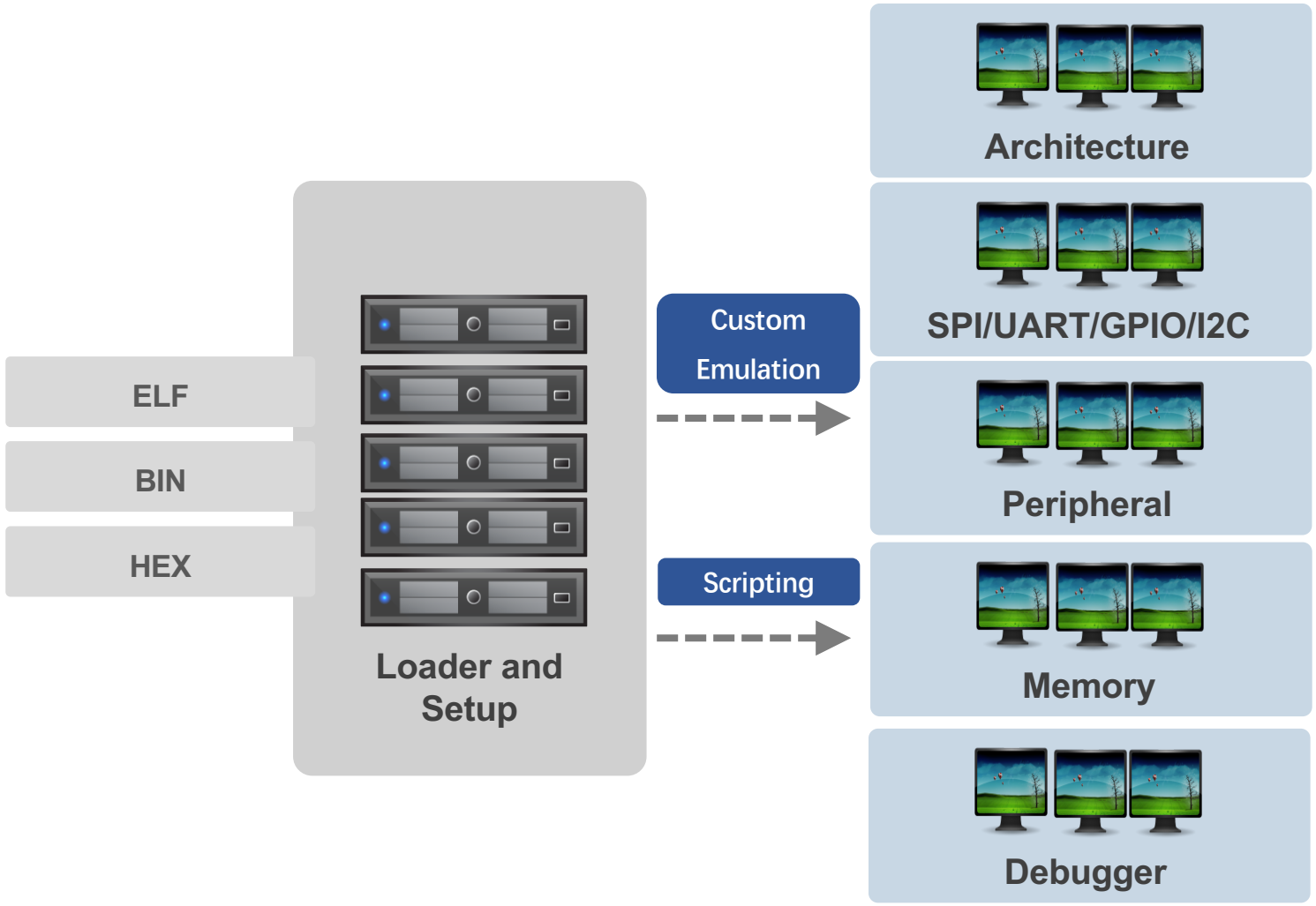
No Actual Hardware Needed

Qiling Framework

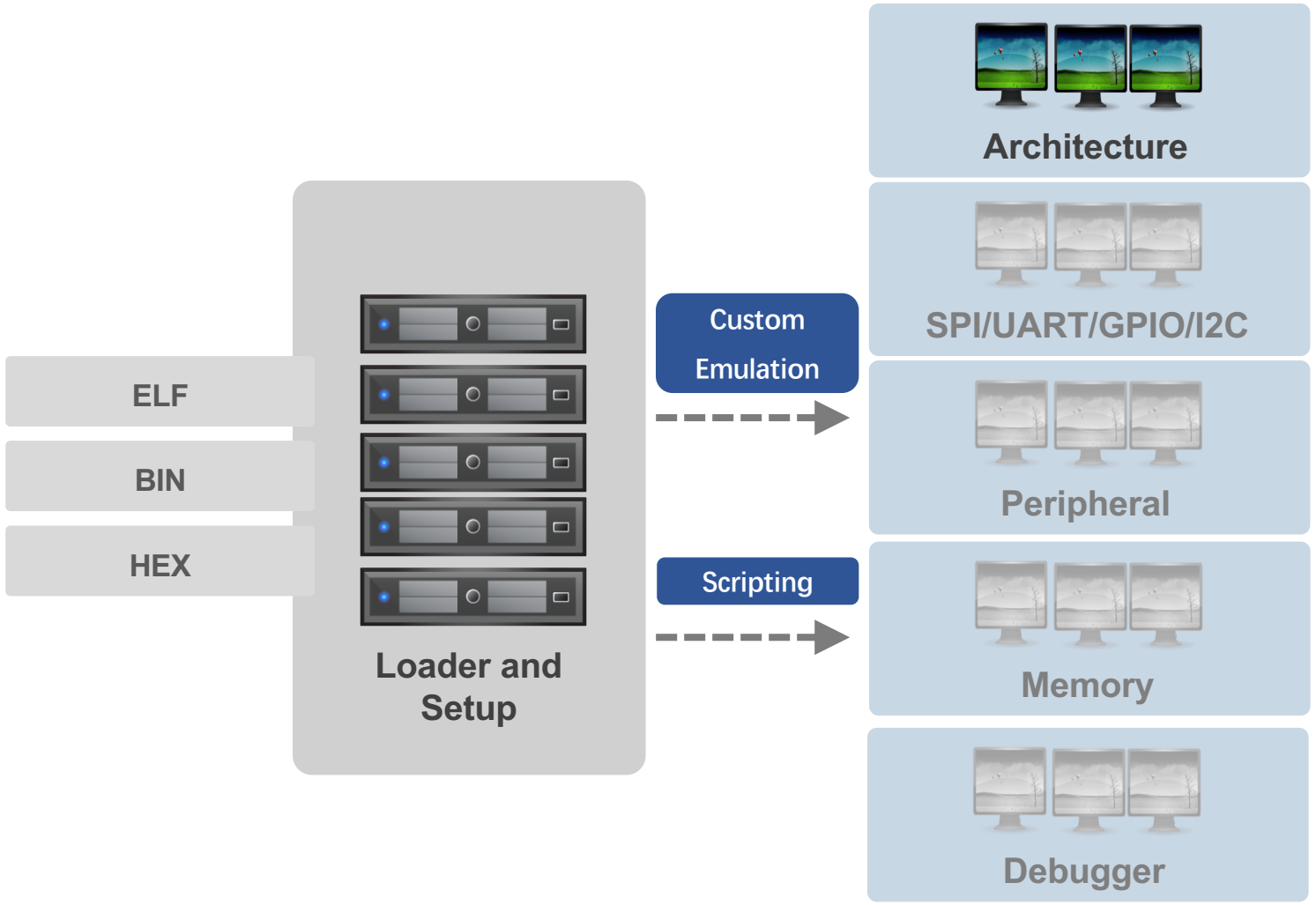
Bring the entire firmware into emulation
with virtual devices support

Hardware Mode and APIs

Qiling Framework Hardware Mode and Its Interface

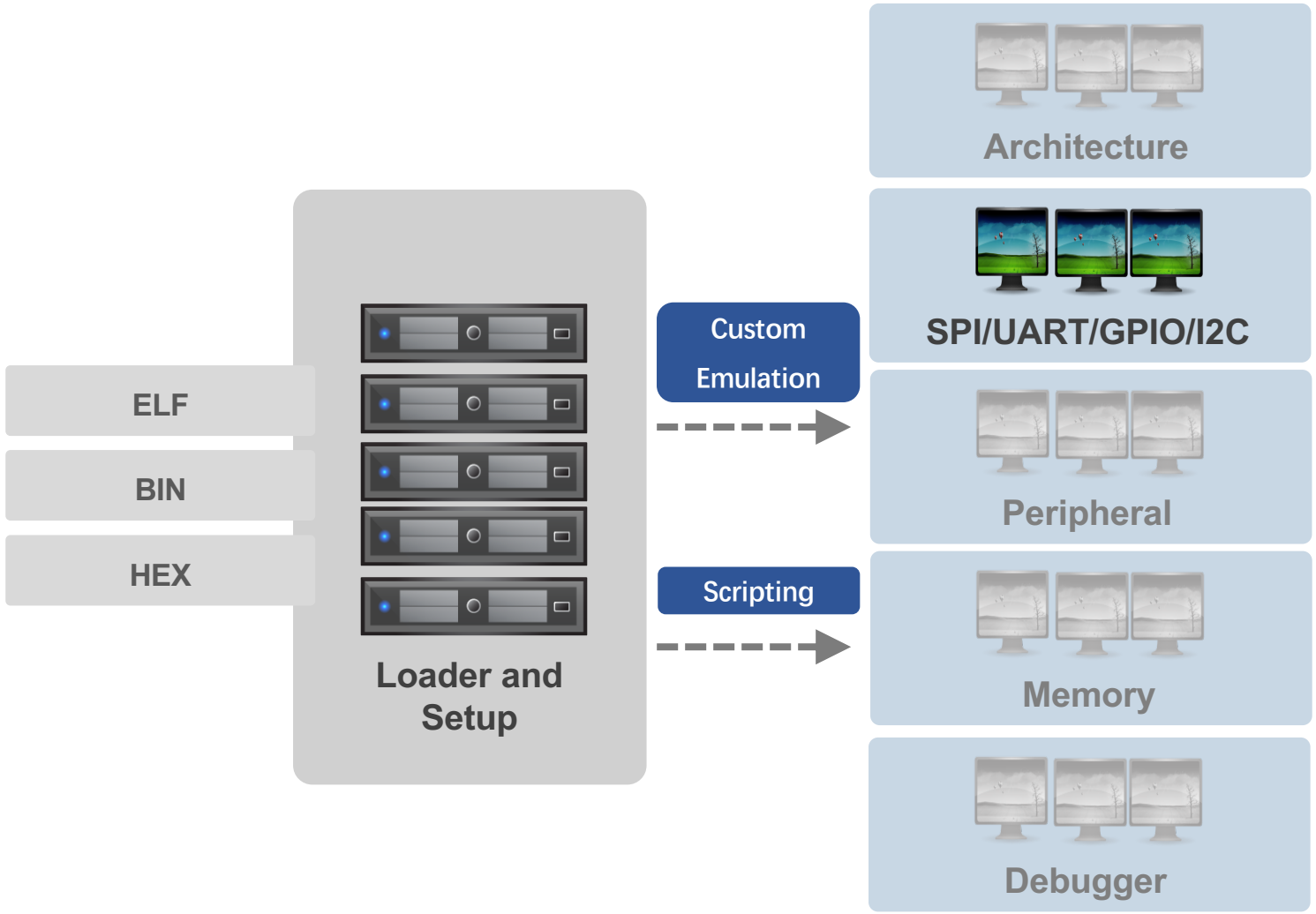


CPU APIs


ELF
BIN
HEX
Loader and Setup
Custom Emulation
Scripting
Architecture
SPI/UART/GPIO/I2C
Peripheral
Memory
Debugger

Examples: CPU

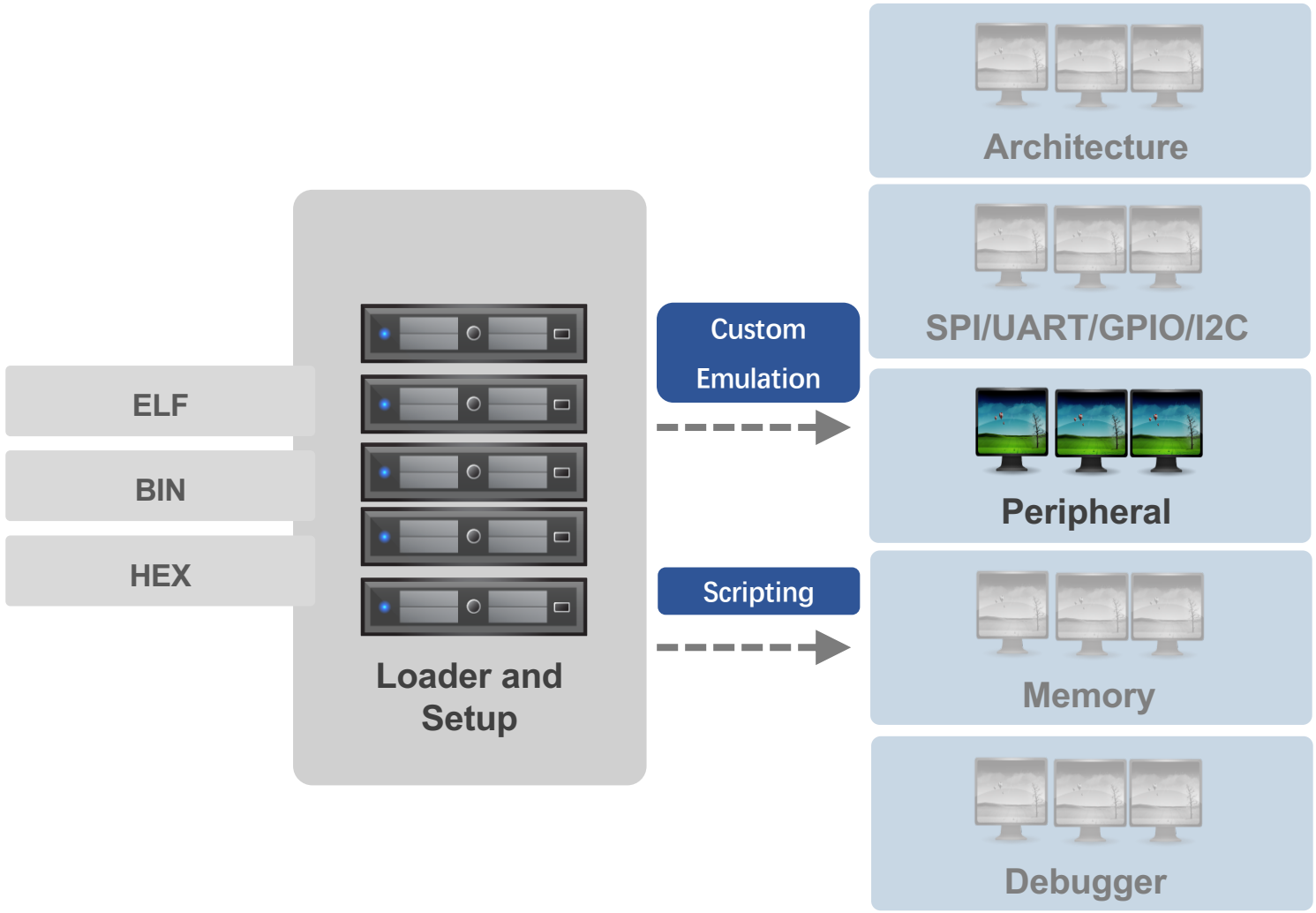
Interface APIs



Examples: SPI/UART/GPIO/I2C

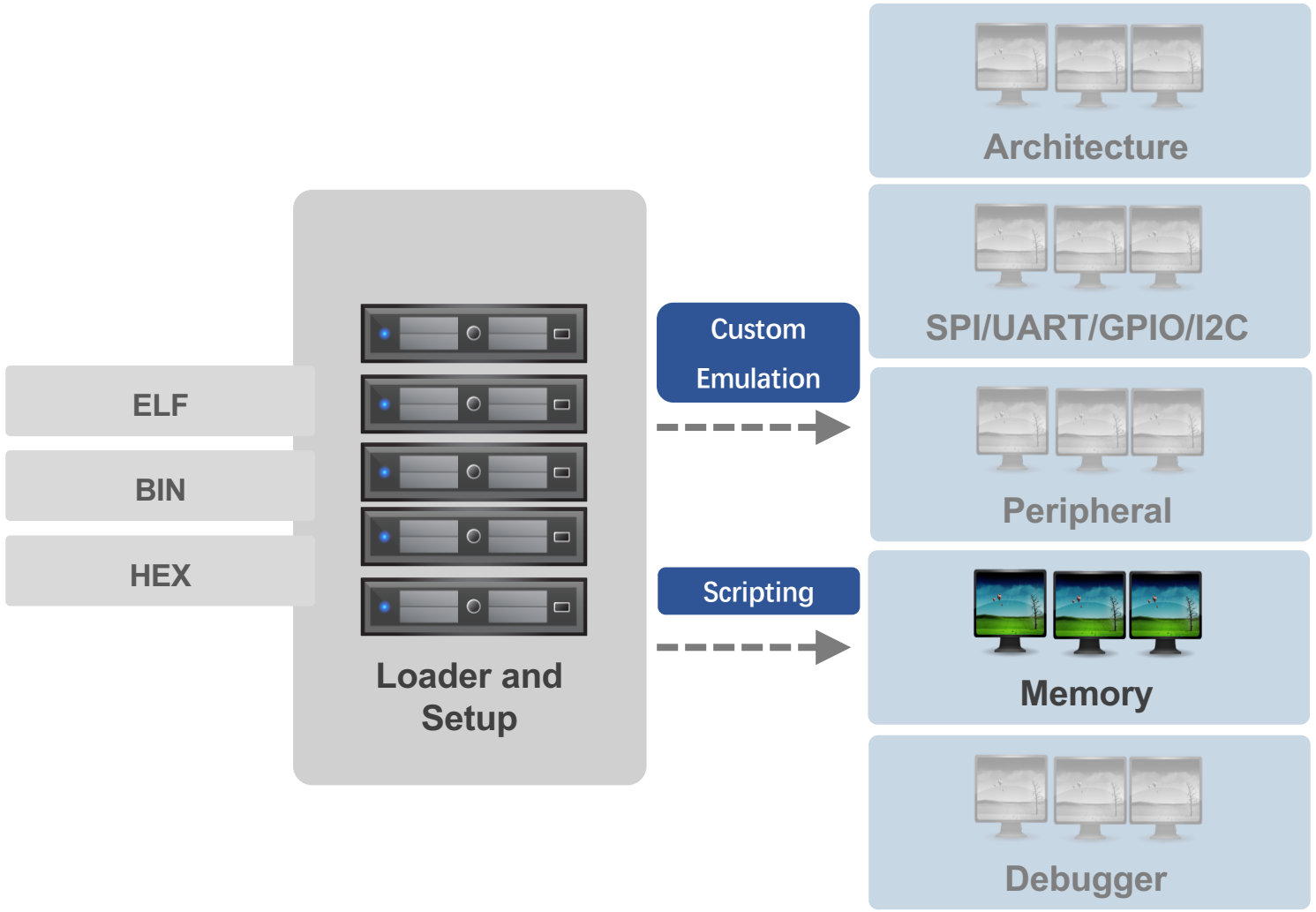

```
ql = Qiling(["../examples/rootfs/stm32f411/hex/hello_usart.hex"],
            archtype="cortex_m", profile="stm32f411", verbose=QL_VERBOSE.DEFAULT)

# create/remove
ql.hw.create('USART', 'usart2', (0x40004400, 0x40004800))
ql.hw.create('STM32F4RCC', 'rcc', (0x40023800, 0x40023C00))
ql.run(count=2000)
```

Peripheral APIs



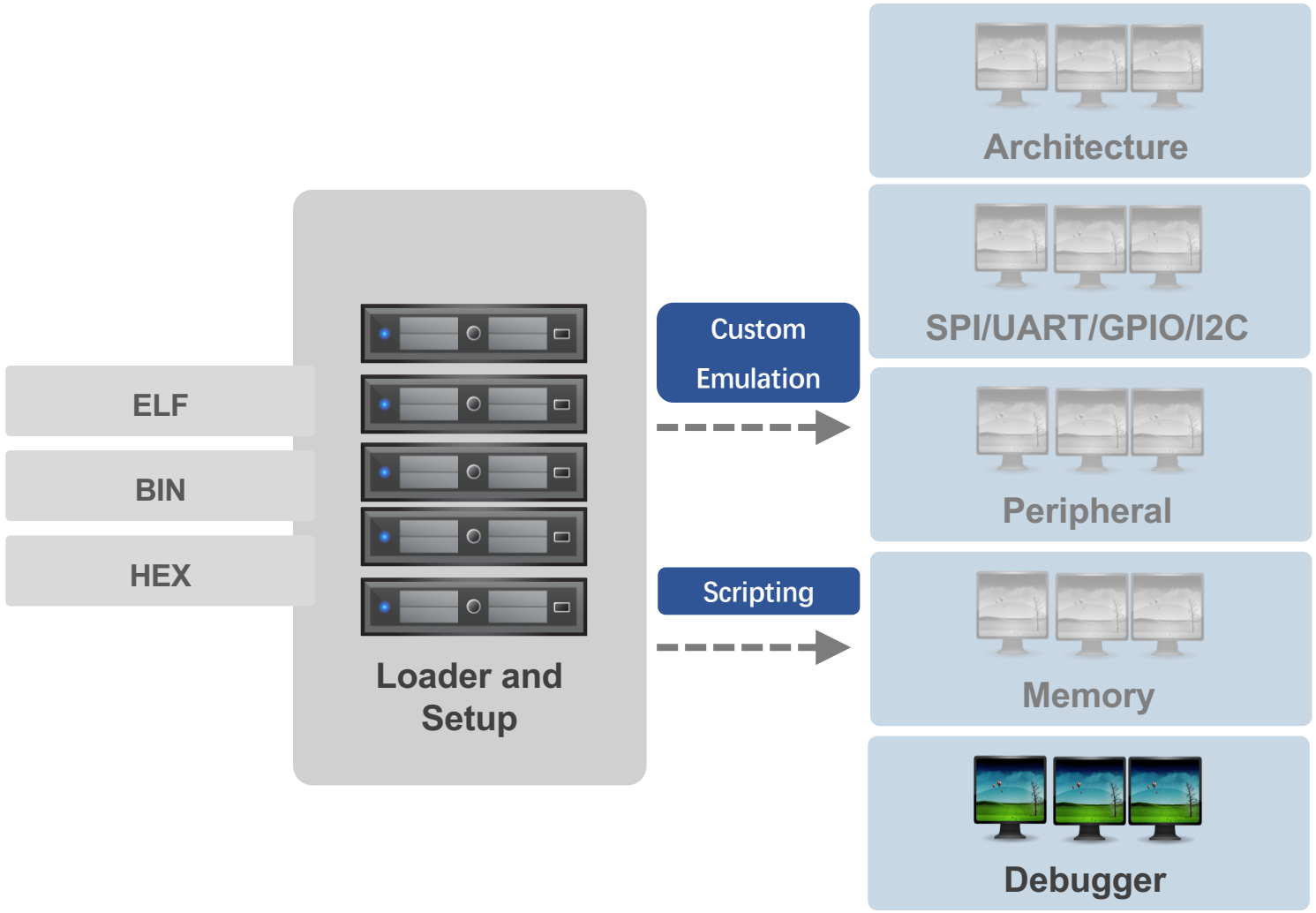
Examples: Peripheral

Memory APIs


ELF
BIN
HEX
Loader and Setup
Custom Emulation
Scripting
Architecture
SPI/UART/GPIO/I2C
Peripheral
Memory
Debugger

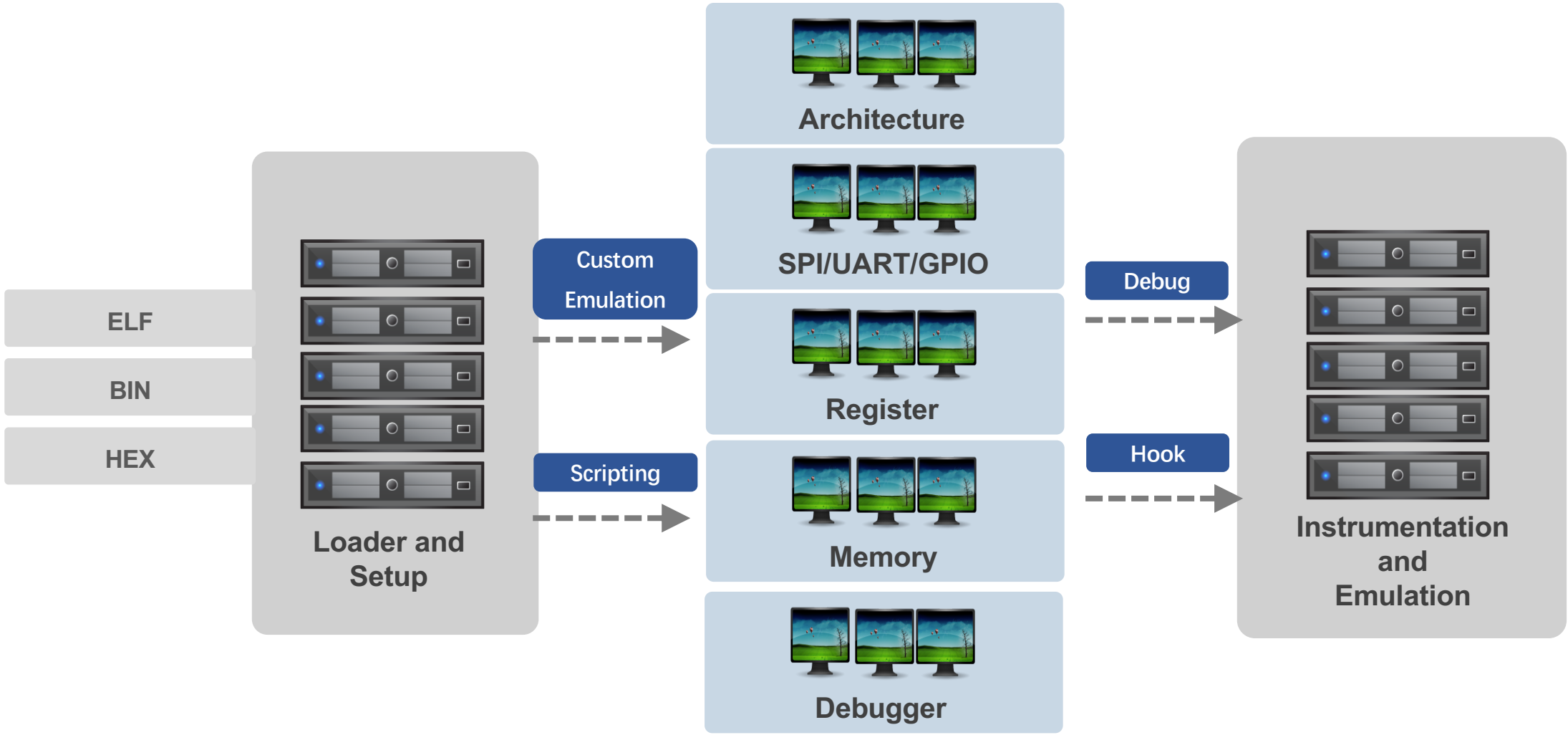
Examples: Memory

Debugger APIs



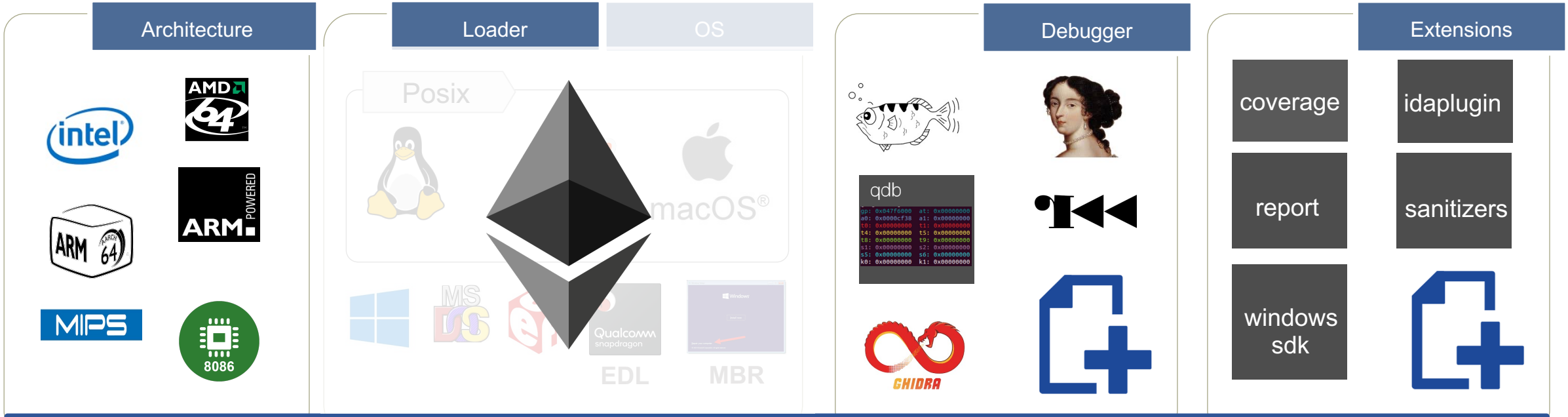
Examples: Debugger Level

Qiling Framework Hardware Mode



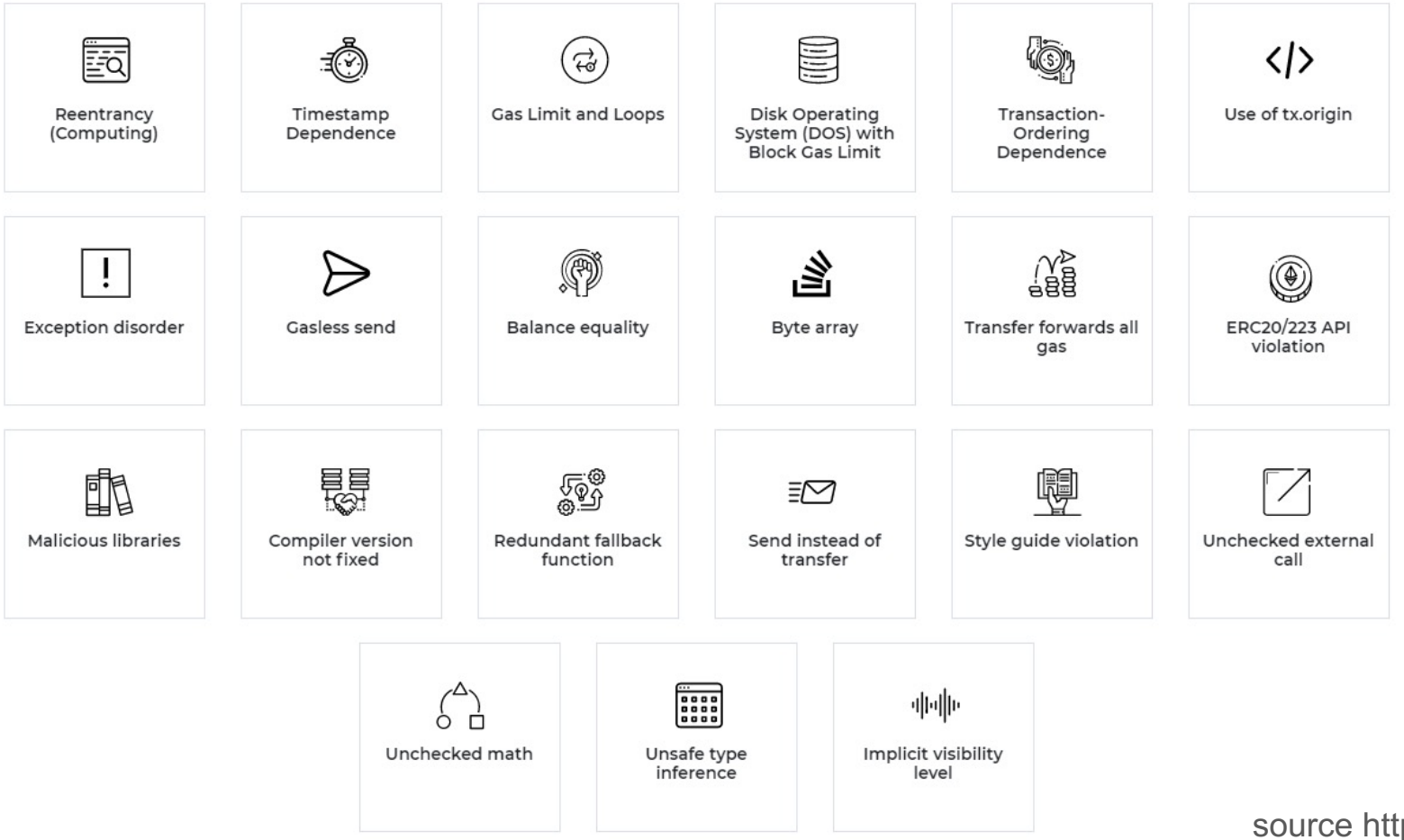
Qiling and EVM Engine

Overview


Architecture
Loader
OS
Debugger
Extensions
Posix
macOS®
EDL
MBR
qdb
GHIDRA
coverage
idaplugin
report
sanitizers
windows sdk
Instrumentation
External Hardware Emulation
Qiling Framework
Unicorn Engine

Make Smart Contract Analysis Smarter

Greater Functions Comes With Greater Bugs



source https://www.developcoins.com/

> Various types of vulnerabilities

> More complicated after DeFi

> 109B DeFi Market Cap, as of April 2021

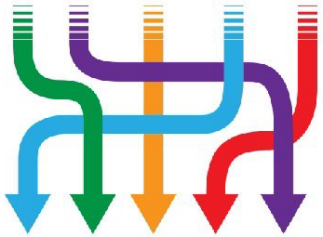
> 22B USD thief in 2019/2020

Today's Smart Contract Analysis Problems

- Binary Only Contracts
- Complex Symbolic Execution
- High False Positive
- Require Human Analysis

Dynamic Symbolic Execution

- Dynamic symbolic execution is a technique for *automatically exploring paths* through a program



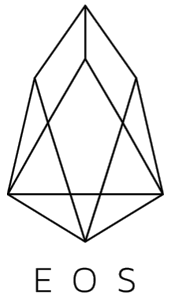
2

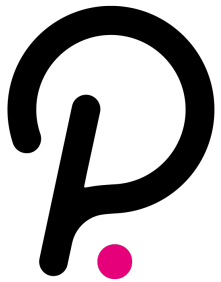
Dynamic cross contract emulation and debug is almost impossible

Not to mention close source smart contract

Wait, There are Official Emulator

Current Emulator, Symbolic Execution Limitation





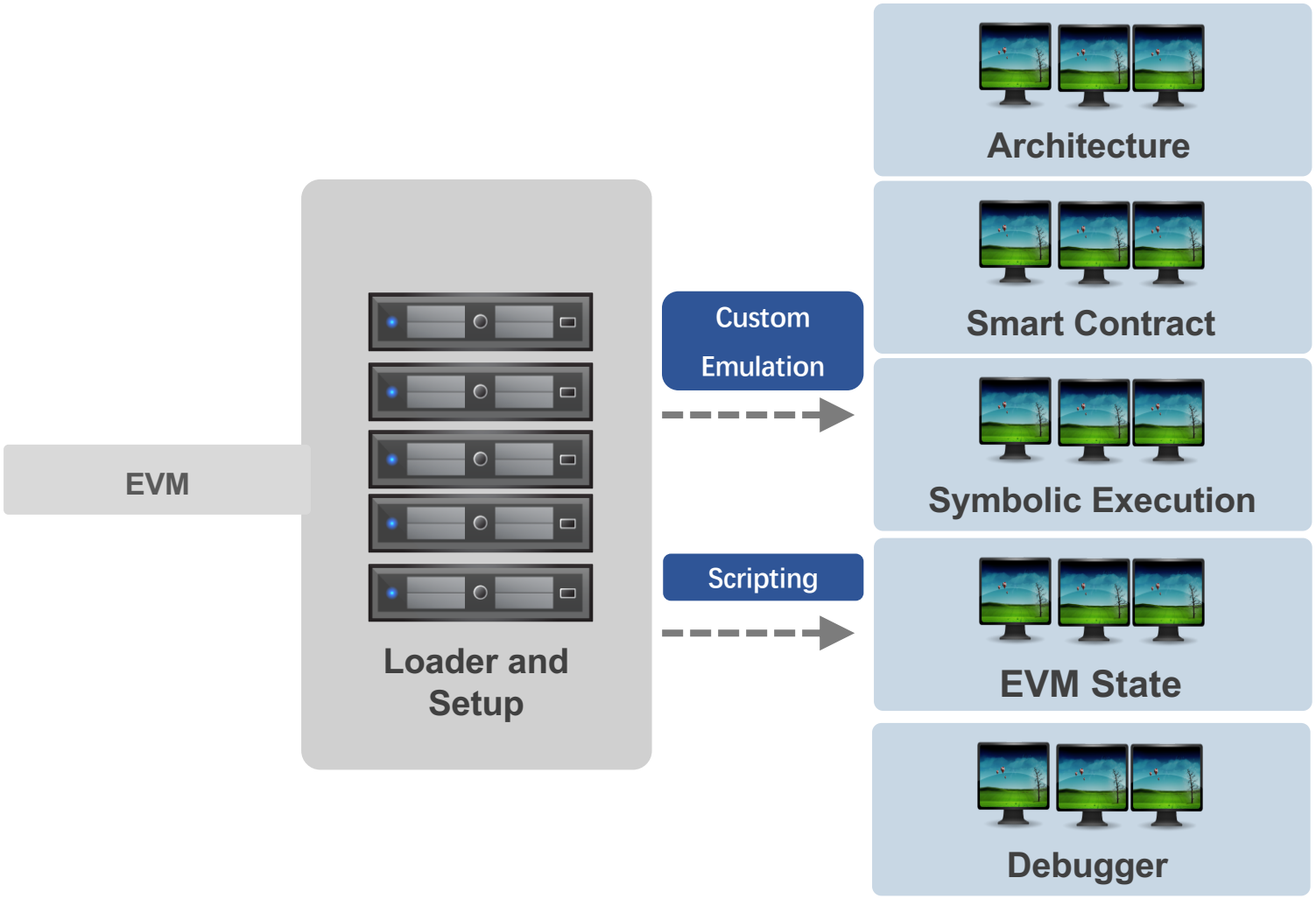
What Is Missing

- Dynamic Execution Hook
- Conditional Execution
- Contract Only Fuzzing
- Pattern Execution
- Live Debugging
- Real Instrumentation
- Not a Framework

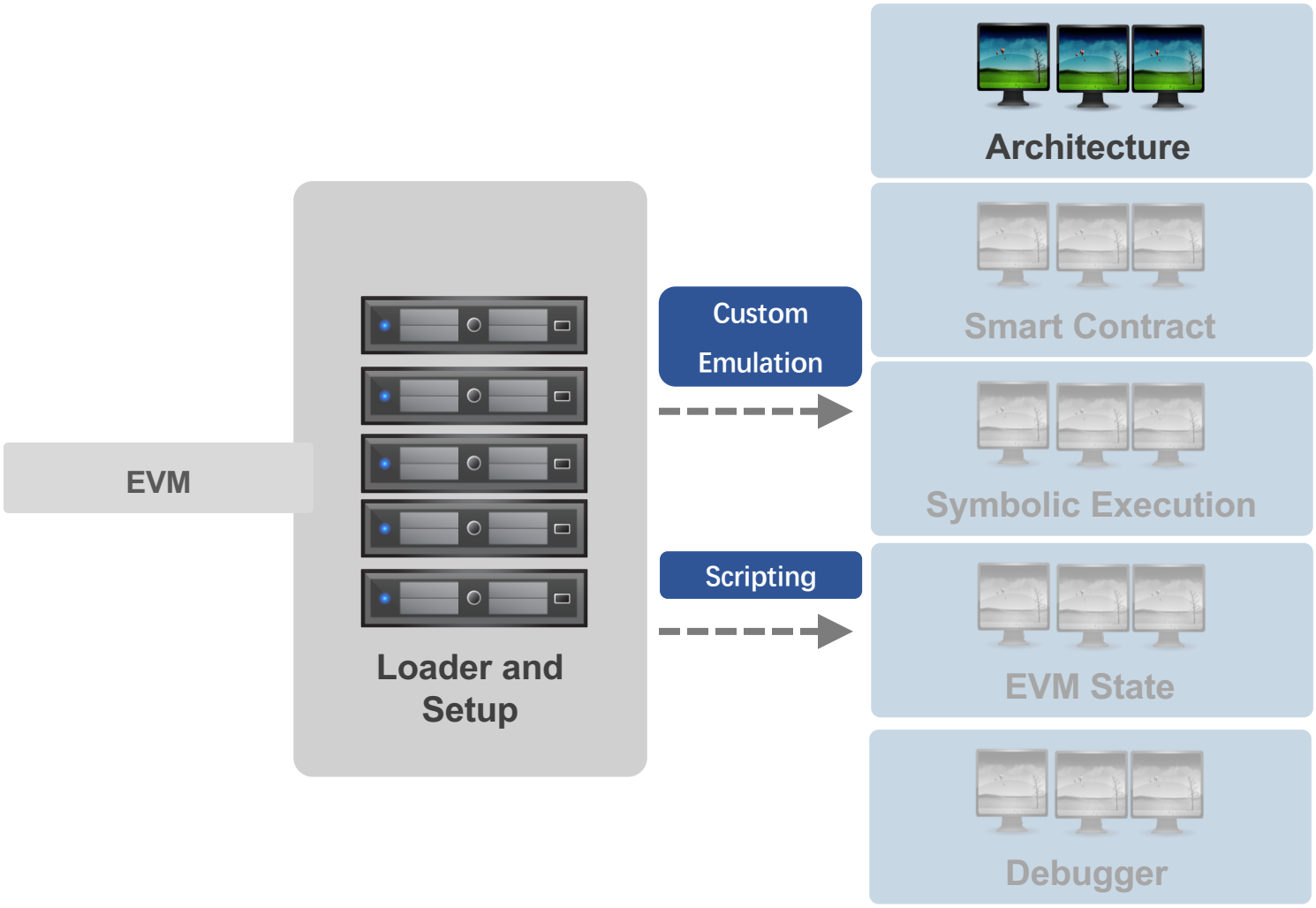
99% of the smart contract enabled block chain are EVM/WASM

EVM Mode and APIs

Qiling Framework EVM Mode and Its Interface



Architecture



- create_account()
- create_message()

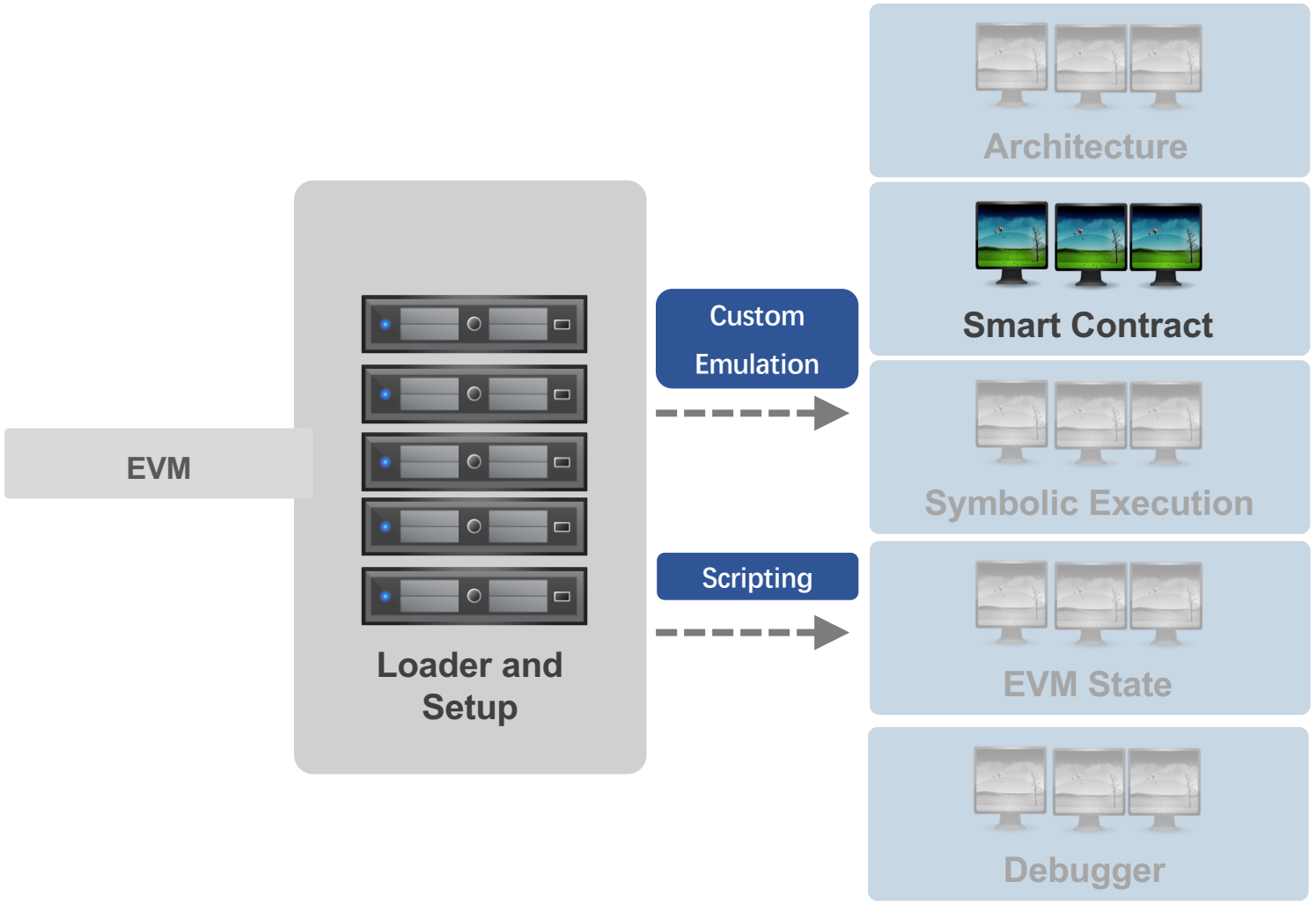
Examples: Architecture

- Initiate “Block Chain”
- Initiate “Smart Contract”

```
ql = Qiling(archtype="evm", verbose=4)
code = '0x6060604052341561000f57600080fd5b60405160208061031c8
argu = ql.arch.evm.abi.convert(['uint256'], [20])
code = code + argu

user1 = ql.arch.evm.create_account(balance=100*10**18)
user2 = ql.arch.evm.create_account(balance=100*10**18)
c1 = ql.arch.evm.create_account()
```

Initiate EVM State and Account DB

Operating System Instrumentation



- hook
  - hook_code()
  - hook_insn()
  - hook_address()
  - hook_del()
- analysis
  - analysis_func_sign()
  - disasm()

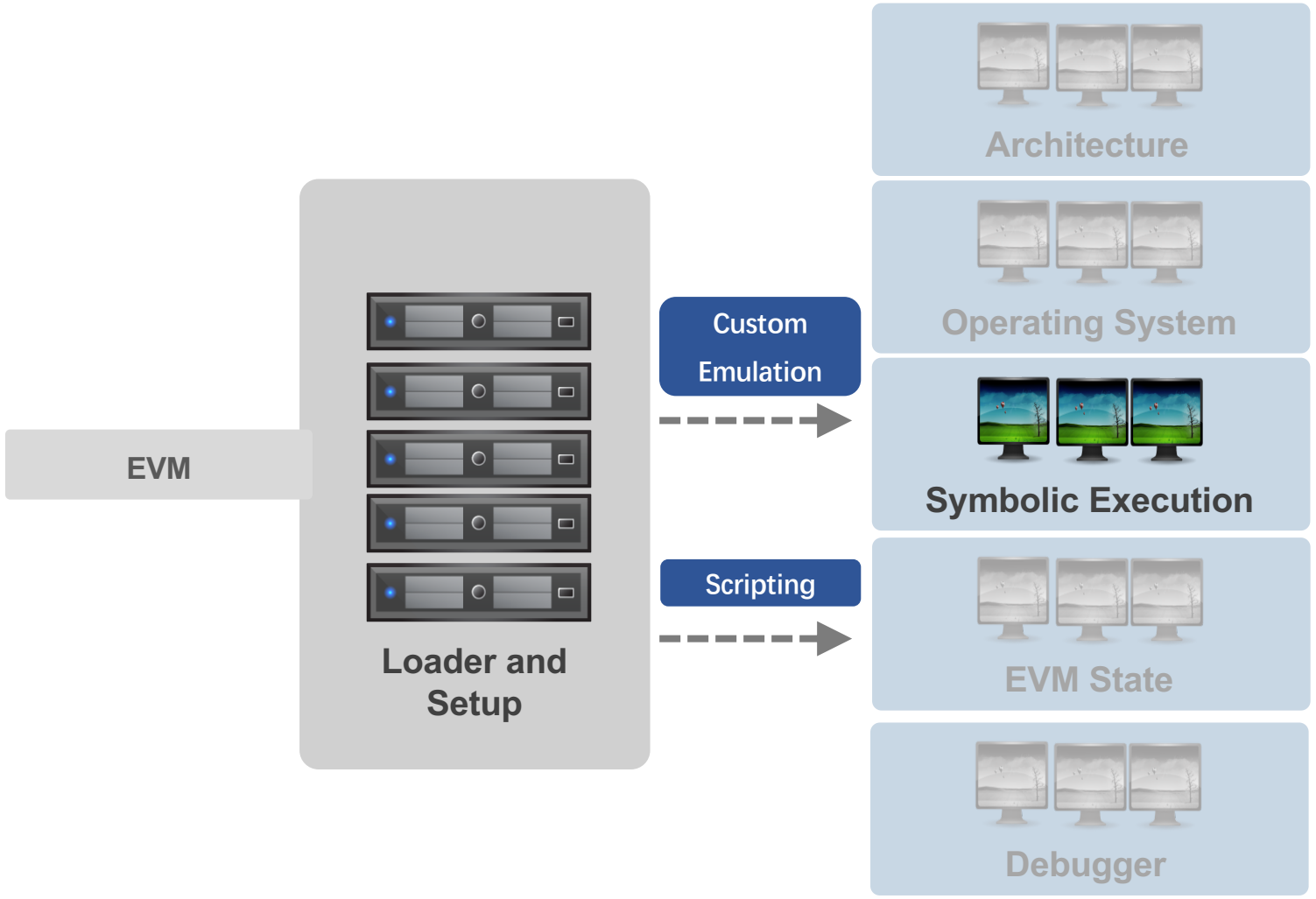
Examples: Smart Contract Hooking


```python
def hookcode_test(ql, *argv):
    print('\033[41;36m hook code success\033[0m ')

def hookinsn_test(ql, *argv):
    print('\033[41;34m hook insn success\033[0m ')

def h_addr(ql):
    print('success!')

h0 = ql.hook_code(hookcode_test)
h1 = ql.hook_address(h_addr, 9)

# message1: deploy runtime code
msg0 = ql.arch.evm.create_message(user1, b'', code=code, contract_address=c1)
ql.run(code=msg0)

ql.hook_del(h0)
ql.hook_del(h1)
h2 = ql.hook_insn(hookinsn_test, 'PUSH4')
```

- Hook everything
  - ql.hook_code()
- Hook specific instruction
  - ql.hook_insn()
- Hook specific address
  - ql.hook_address()

Provide a variety of Instrumentation

Symbolic Execution



- create_analyzer()
- symbolic_cfg()

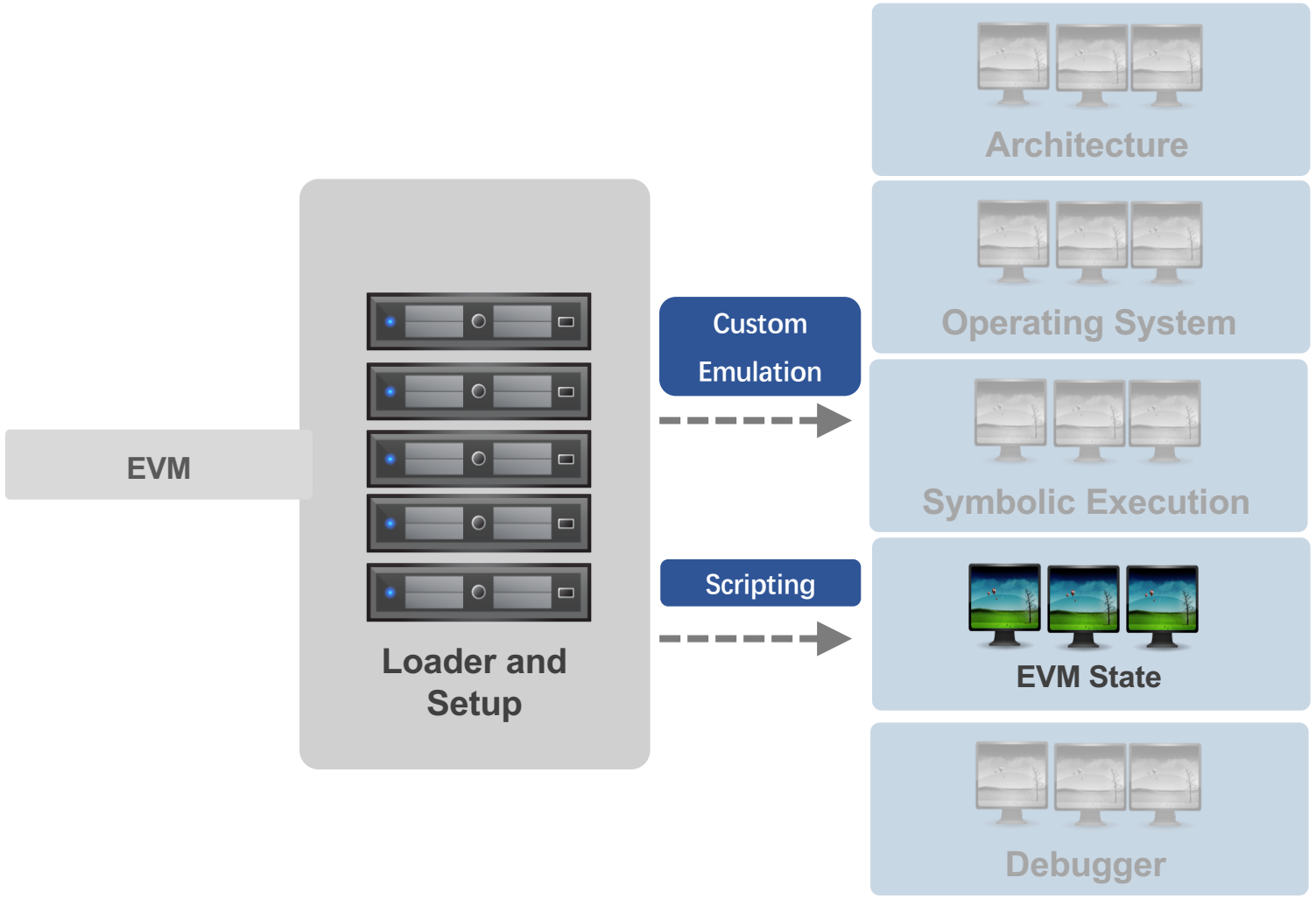
Examples: Symbolic Execution


```
contract = '6060604052341561000f57600080fd5b60405160208061031c83398
bal = ql.arch.evm.abi.convert(['uint256'], [20000000])
contract = contract + bal

svm = EthSVM()
svm.disassembler.load_from_bytecode(code=contract)
svm.create_analyzer(svm.disassembler)

sym_cfg = svm.symbolic_cfg()
```

- create_analyzer()
- symbolic_cfg()

Provides symbol execution analysis capabilities

EVM State



- Get
  - get_storage()
  - get_balance()
  - get_code()
  - …
- Set
  - set_storage()
  - set_balance()
  - set_code()
  - …
- Account
  - touch_account()
  - delete_account()

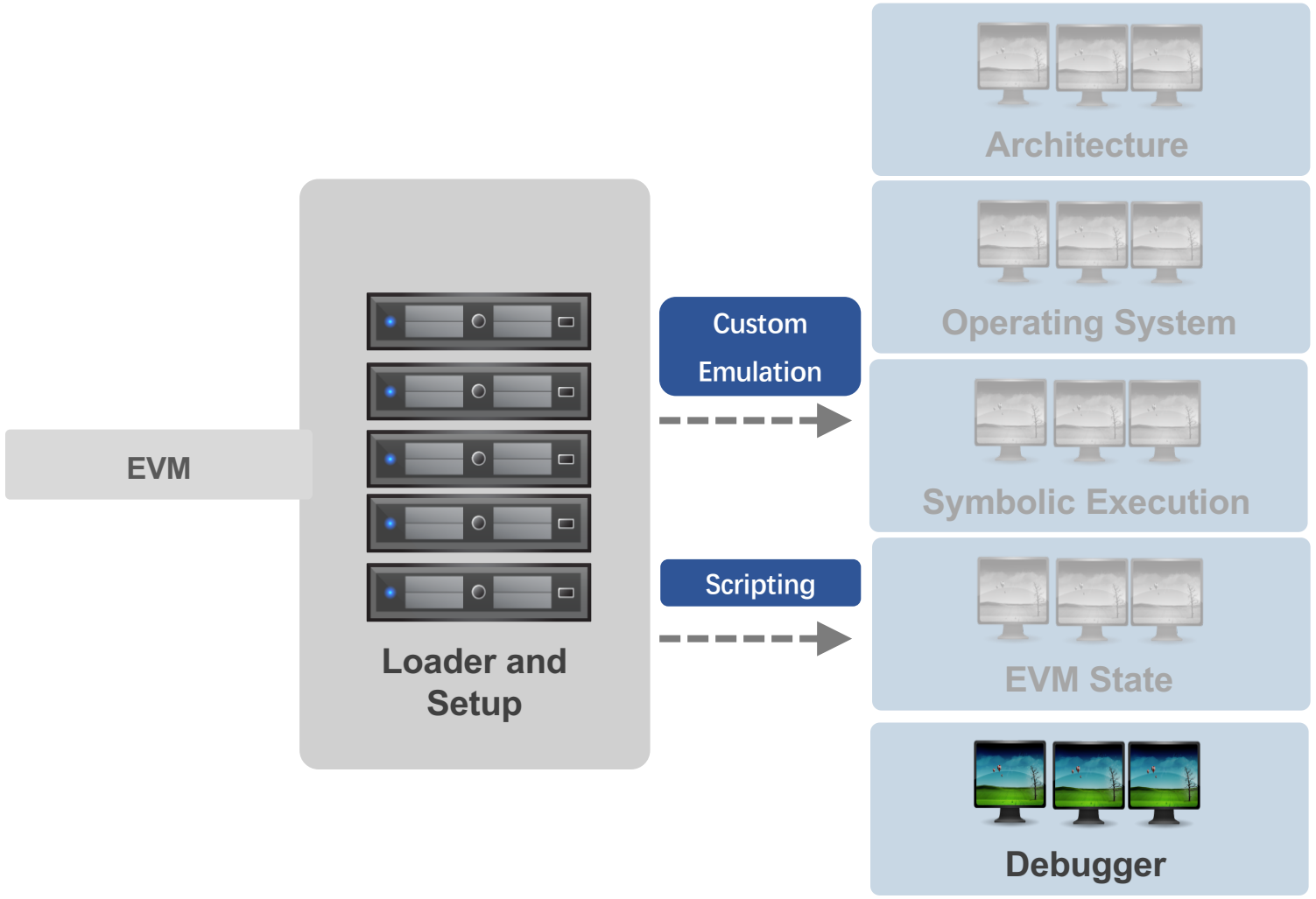
Examples: EVM State


```python
def get_storage(self, address: Address, slot: int, from_journal: bool
    return self._account_db.get_storage(address, slot, from_journal)

def set_storage(self, address: Address, slot: int, value: int) -> None
    return self._account_db.set_storage(address, slot, value)

def delete_storage(self, address: Address) -> None:
    self._account_db.delete_storage(address)

def delete_account(self, address: Address) -> None:
    self._account_db.delete_account(address)

def get_balance(self, address: Address) -> int:
    return self._account_db.get_balance(address)

def set_balance(self, address: Address, balance: int) -> None:
    self._account_db.set_balance(address, balance)

def delta_balance(self, address: Address, delta: int) -> None:
    self.set_balance(address, self.get_balance(address) + delta)
```

- Get
  - get_storage()
  - get_balance()
  - get_code()
- Set
  - set_storage()
  - set_balance()
  - set_code()
- Account
  - touch_account()
  - delete_account()

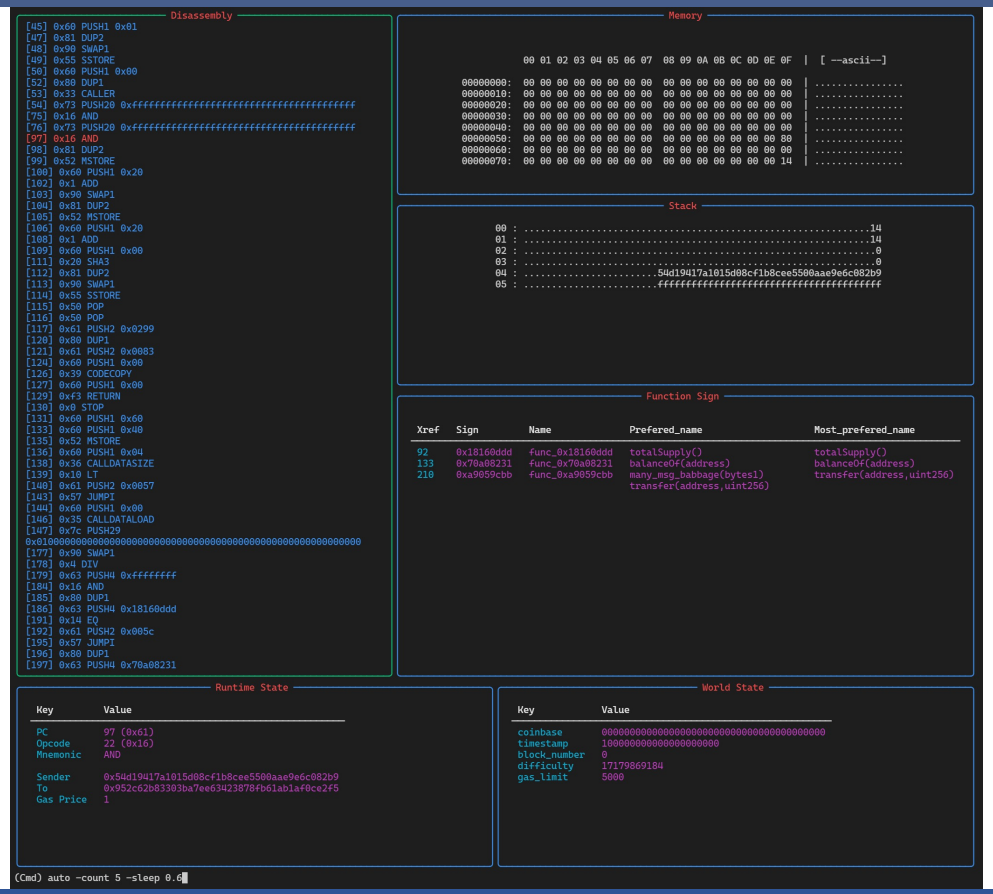
The ability to obtain EVM State in real time

Debug



- execute_debug()

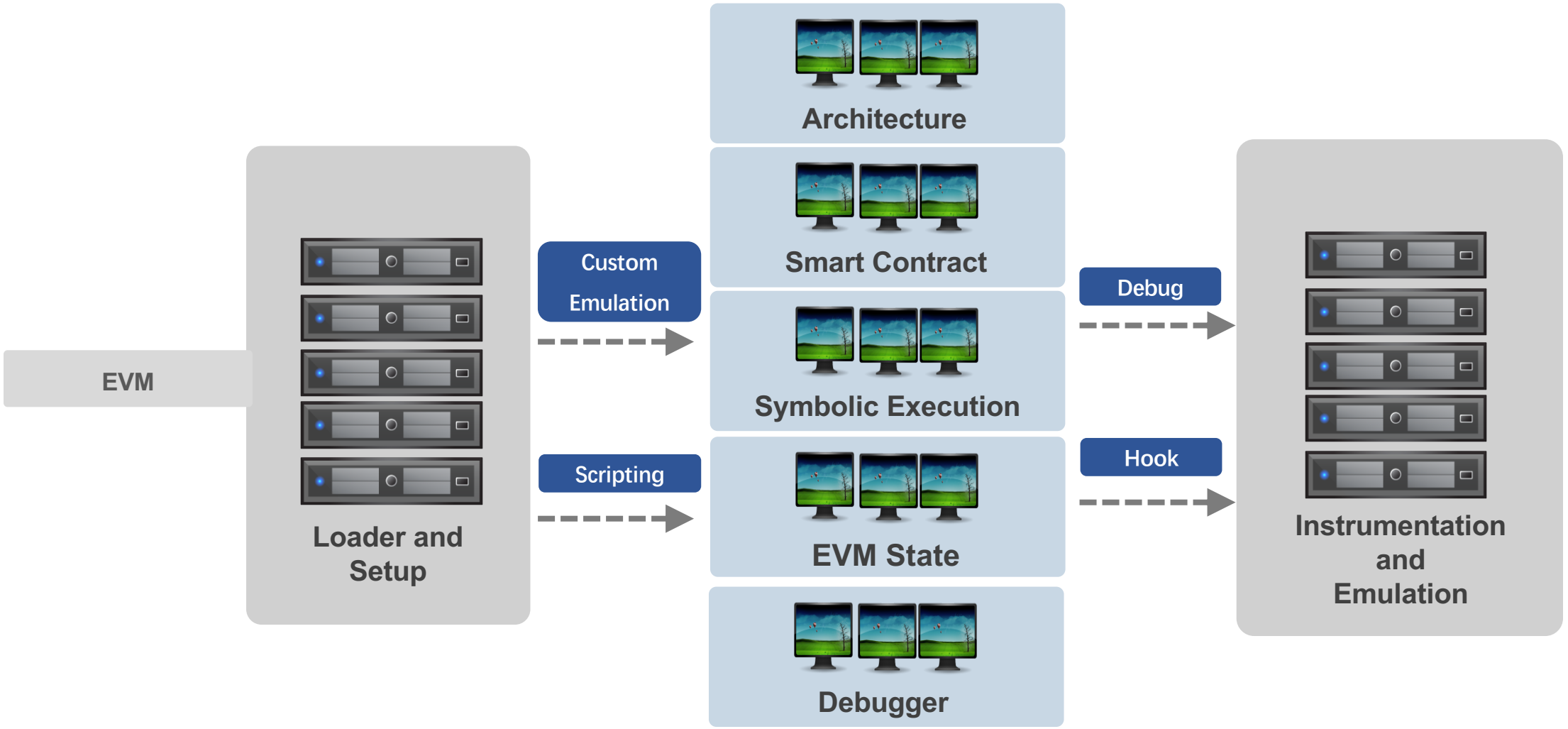
Examples: Debug



- execute_debug()
  - step in
  - continue
  - let breakpoint
  - list breakpoint

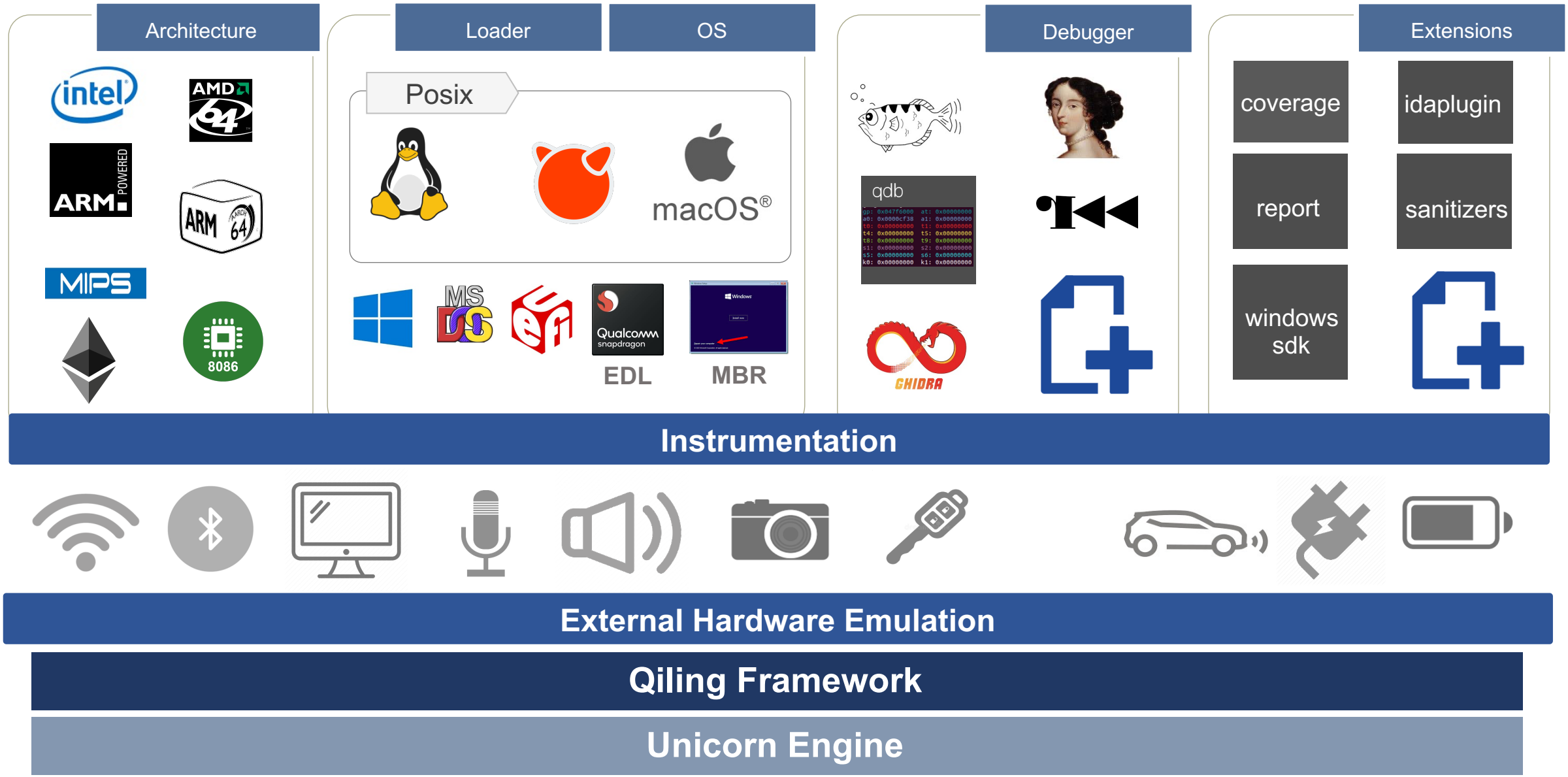
Realtime smart contract debugger

Qiling Framework EVM Mode



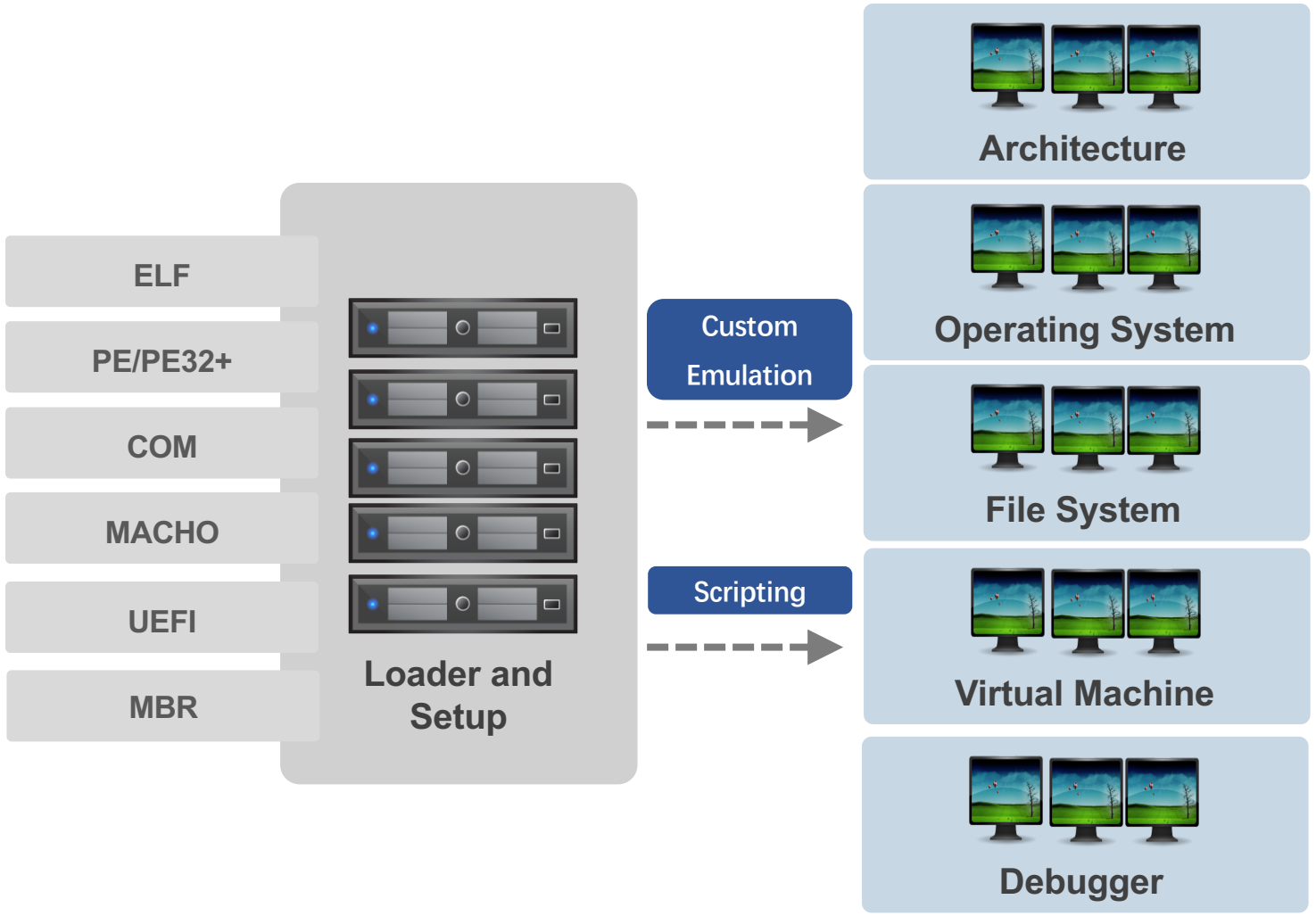
Qiling and Software

Overview

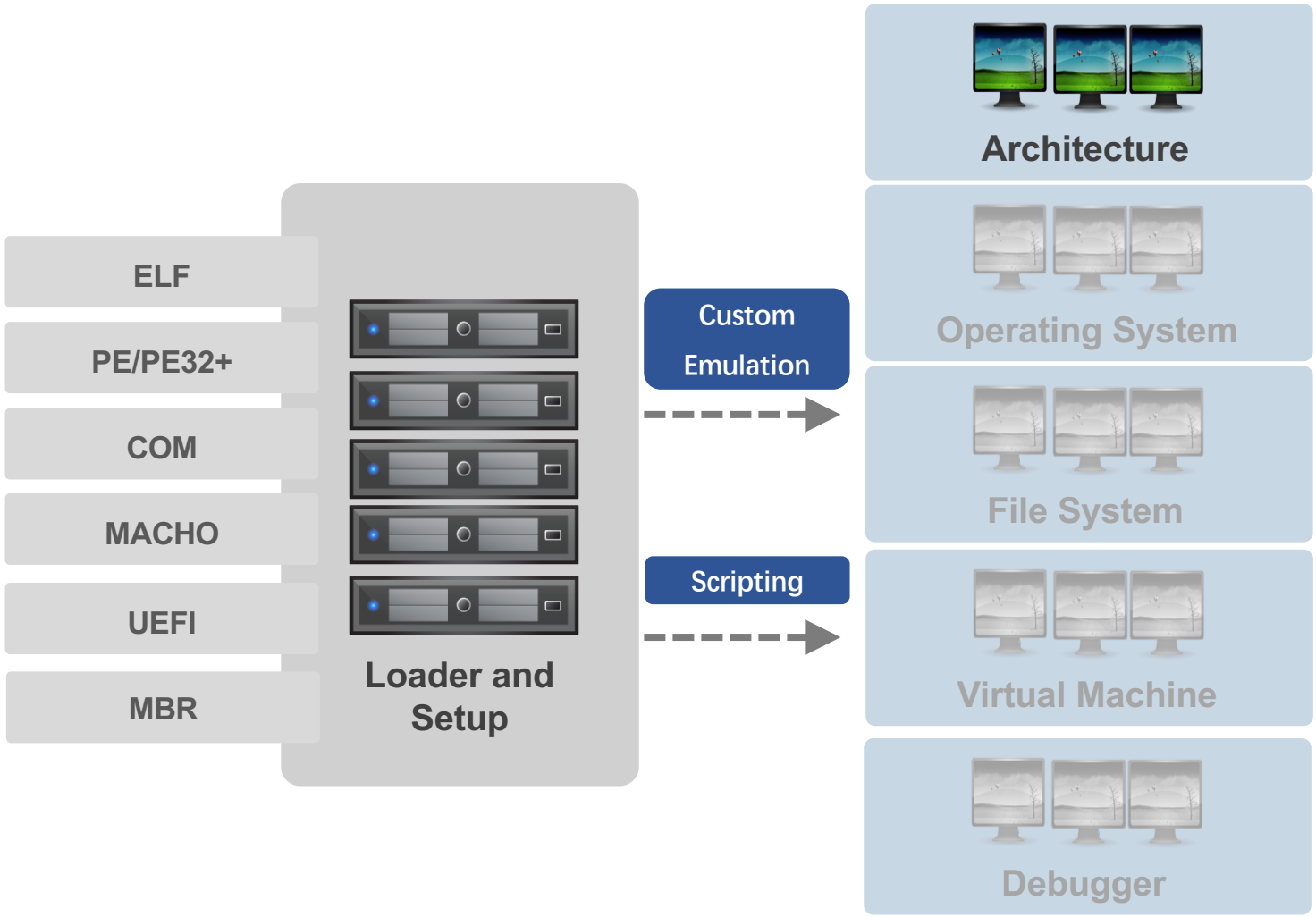

Architecture
Loader
OS
Debugger
Extensions
Posix
macOS®
qdb
GHIDRA
coverage
idaplugin
report
sanitizers
windows sdk
EDL
MBR
Instrumentation
External Hardware Emulation
Qiling Framework
Unicorn Engine

Software Mode and APIs

Qiling Framework and Its Interface



CPU Instrumentation



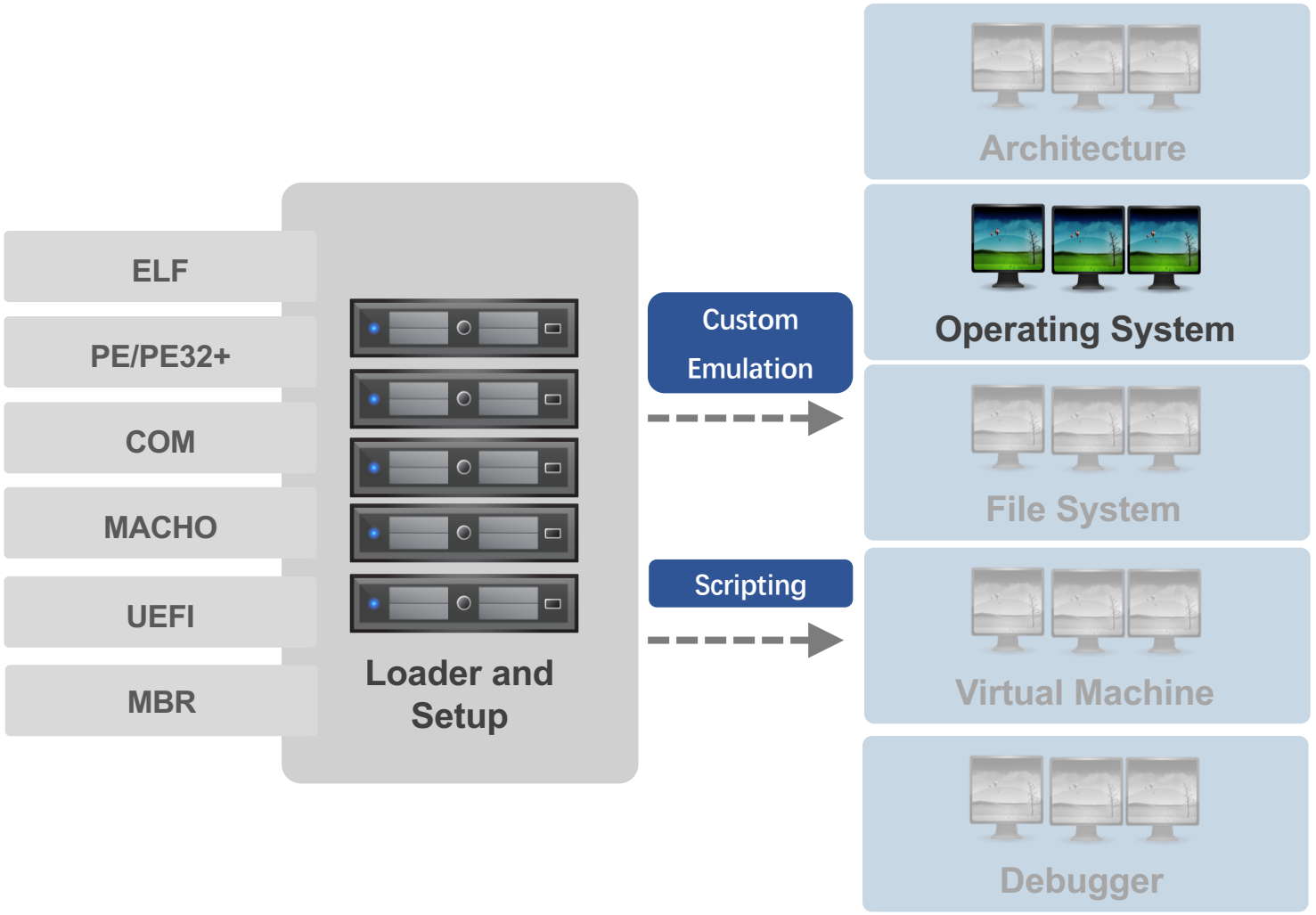
- Access to Register
- Reading register
  - ql.reg.eax
- Writing to register
  - ql.reg.eax = 0x41
- Differet Hooks
  - ql.hook_code()
  - ql.hook_address()
  - and more

CPU Instrumentation: Examples


```
def my_puts(ql):
    addr = ql.os.function_arg[0]
    print("puts(%s)" % ql.mem.string(addr))
    reg = ql.reg.read("rax")
    print("reg : 0x%x" % reg)
    ql.reg.rax = reg
    self.set_api = reg

def write_onEnter(ql, arg1, arg2, arg3, *args):
    print("enter write syscall!")
    ql.reg.rsi = arg2 + 1
    ql.reg.rdx = arg3 - 1
    self.set_api_onenter = True

def write_onexit(ql, arg1, arg2, arg3, *args):
    print("exit write syscall!")
    ql.reg.rax = arg3 + 1
    self.set_api_onexit = True

ql = Qiling(["../examples/rootfs/x8664_linux/bin/x8664_args","1234test", "12345678", "bin/x8664_hello"],  "../e
ql.set_syscall(1, write_onEnter, QL_INTERCEPT.ENTER)
ql.set_api('puts', my_puts)
ql.set_syscall(1, write_onexit, QL_INTERCEPT.EXIT)
ql.mem.map(0x1000, 0x1000)
ql.mem.write(0x1000, b"\xFF\xFE\xFD\xFC\xFB\xFA\xFB\xFC\xFC\xFE\xFD")
ql.mem.map(0x2000, 0x1000)
ql.mem.write(0x2000, b"\xFF\xFE\xFD\xFC\xFB\xFA\xFB\xFC\xFC\xFE\xFD")
ql.run()
```

- Access to Register
- Reading register
  - eax = ql.reg.eax
- Writing to register
  - ql.reg.eax = 0x41
- Differet Hooks
  - ql.hook_code()
  - ql.hook_address()
  - and more

Operating System Instrumentation



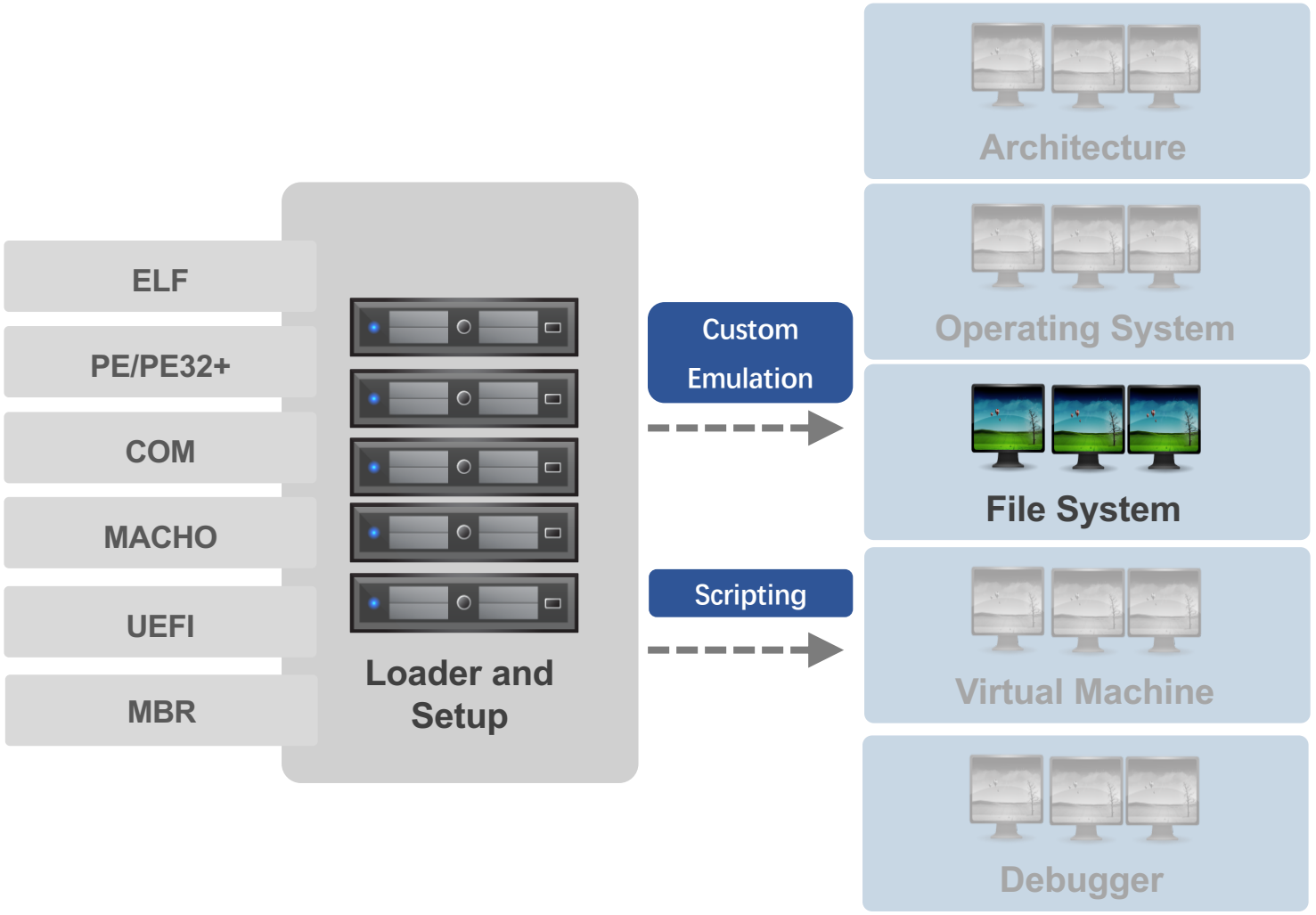
- Access to memory
  - ql.mem.read()
  - ql.mem.write()
- Search pattern from memory
  - ql.mem.search()
- Stack related operation
  - ql.stack_pop
  - ql.stack_push
- Syscall replacement
  - ql.set_syscall()
  - ql.set_api()
- Replace library call with
  - ql.set_api()

Example: Operating System


```python
from qiling import *

def my_syscall_write(ql, write_fd, write_buf, write_count, *args, **kw):
    regreturn = 0

    try:
        buf = ql.mem.read(write_buf, write_count)
        ql.nprint("\n+++++++++\nmy write(%d,%x,%i) = %d\n+++++++++" % (write_fd, write_buf, write_count, regreturn))
        ql.os.fd[write_fd].write(buf)
        regreturn = write_count
    except:
        regreturn = -1
        ql.nprint("\n+++++++++\nmy write(%d,%x,%i) = %d\n+++++++++" % (write_fd, write_buf, write_count, regreturn))
        if ql.output in (QL_OUTPUT.DEBUG, QL_OUTPUT.DUMP):
            raise

    ql.os.definesyscall_return(regreturn)


if __name__ == "__main__":
    ql = Qiling(["rootfs/arm_linux/bin/arm_hello"], "rootfs/arm_linux", output = "debug")
    # Custom syscall handler by syscall name or syscall number.
    # Known issue: If the syscall func is not be implemented in qiling, qiling does
    # not know which func should be replaced.
    # In that case, you must specify syscall by its number.
    ql.set_syscall(0x04, my_syscall_write)

    # set syscall by syscall name
    #ql.set_syscall("write", my_syscall_write)

    ql.run()
```

- Access to memory
  - ql.mem.read()
  - ql.mem.write()
- Search pattern from memory
  - ql.mem.search()
- Stack related operation
  - ql.stack_pop
  - ql.stack_push
- Syscall replacement
  - ql.set_syscall()
  - ql.set_api()
- Replace library call with
  - ql.set_api()

File System Instrumentation



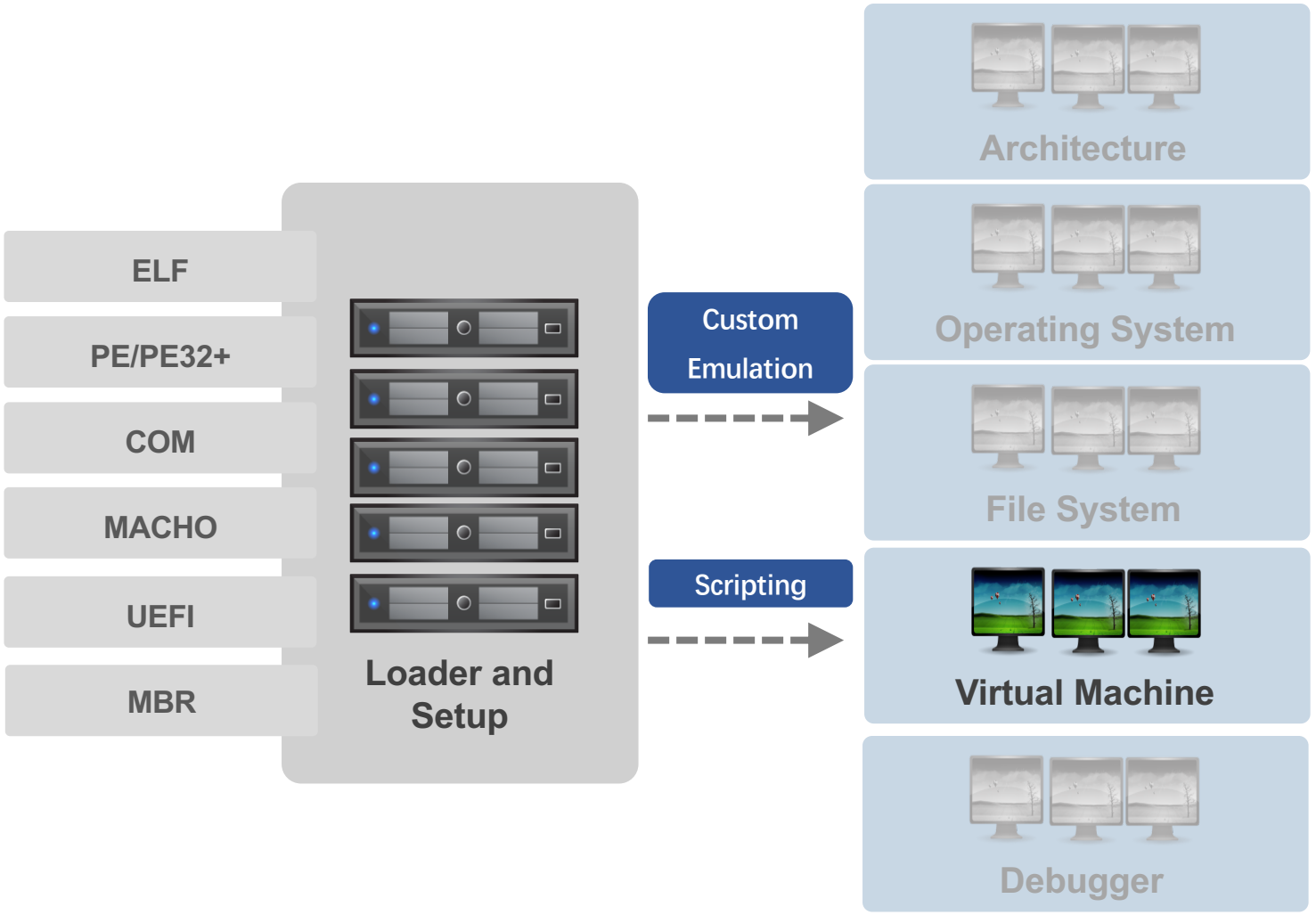
- Map host file
  - ql.fs_mapper()
- Hijack accessed file
  - ql.fs_mapper(hijack_func)
- Stdio replacement
  - stdin
  - stdout
  - stderr
- Patch file's memory before execution
  - ql.patch

Example: File System Instrumentation


```python
from qiling import *
from qiling.os.mapper import QlFsMappedObject

class Fake_urandom(QlFsMappedObject):

    def read(self, size):
        return b"\x01" # fixed value for reading /dev/urandom

    def fstat(self): # syscall fstat will ignore it if return -1
        return -1

    def close(self):
        return 0

if __name__ == "__main__":
    ql = Qiling(["rootfs/x86_linux/bin/x86_fetch_urandom"], "rootfs/x86_linux")
    ql.add_fs_mapper("/dev/urandom", Fake_urandom())
    ql.run()
```

- Map host file
  - ql.fs_mapper()
- Hijack accessed file
  - ql.fs_mapper(hijack_func)
- Stdio replacement
  - stdin
  - stdout
  - stderr
- Patch file's memory before execution
  - ql.patch

Virtual Machine Instrumentation



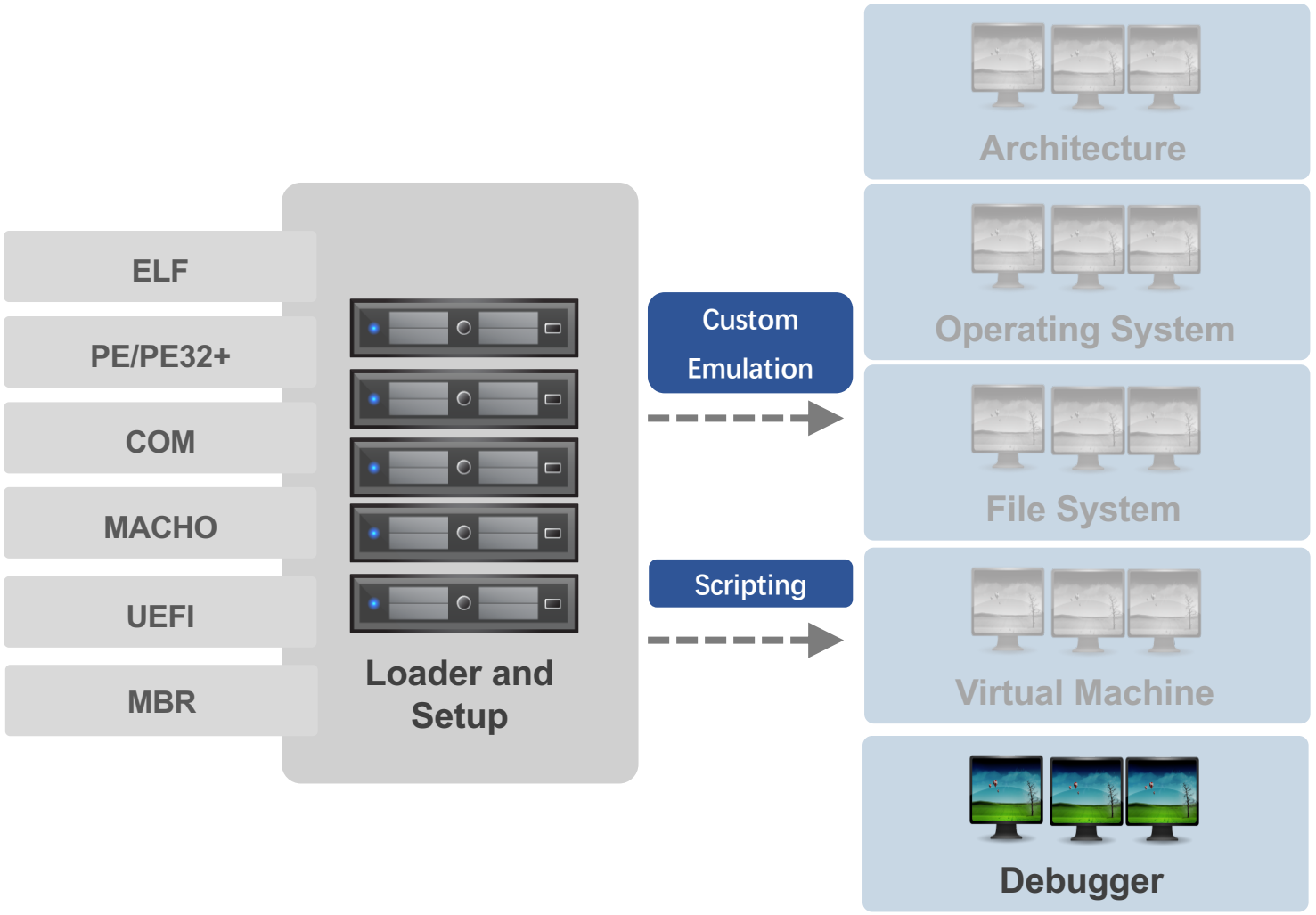
- Save current state
  - ql.save()
- Restore current state
  - ql.restore()
- Save/restore memory only
  - ql.mem.save()
- Save/restore register only
  - ql.reg.save()

Example: Virtual Machine Instrumentation


```python
def save_context(ql, *args, **kw):
    ql.save(cpu_context=False, snapshot="snapshot.bin")

def patcher(ql):
    br0_addr = ql.mem.search("br0".encode() + b'\x00')
    for addr in br0_addr:
        ql.mem.write(addr, b'lo\x00')


def check_pc(ql):
    print("=" * 50)
    print("[!] Hit fuzz point, stop at PC = 0x%x" % ql.reg.arch_pc)
    print("=" * 50)
    ql.emu_stop()

def my_sandbox(path, rootfs):
    ql = Qiling(path, rootfs, output="debug", verbose=5)
    ql.add_fs_mapper("/dev/urandom","/dev/urandom")
    ql.hook_address(save_context ,0x10930)
    ql.hook_address(patcher, ql.loader.elf_entry)
    ql.hook_address(check_pc,0x7a0cc)
    ql.run()

if __name__ == "__main__":
    nvram_listener_therad =  threading.Thread(target=nvram_listener, daemon=True)
    nvram_listener_therad.start()
    my_sandbox(["rootfs/bin/httpd"], "rootfs")
```

- Save current state
  - ql.save()
- Restore current state
  - ql.restore()
- Save/restore memory only
  - ql.mem.save()
- Save/restore register only
  - ql.reg.save()

Debugger



- Open API for RSP compatible Debugger
- Build In debugger – Qdbg
  - Able to reverse debug

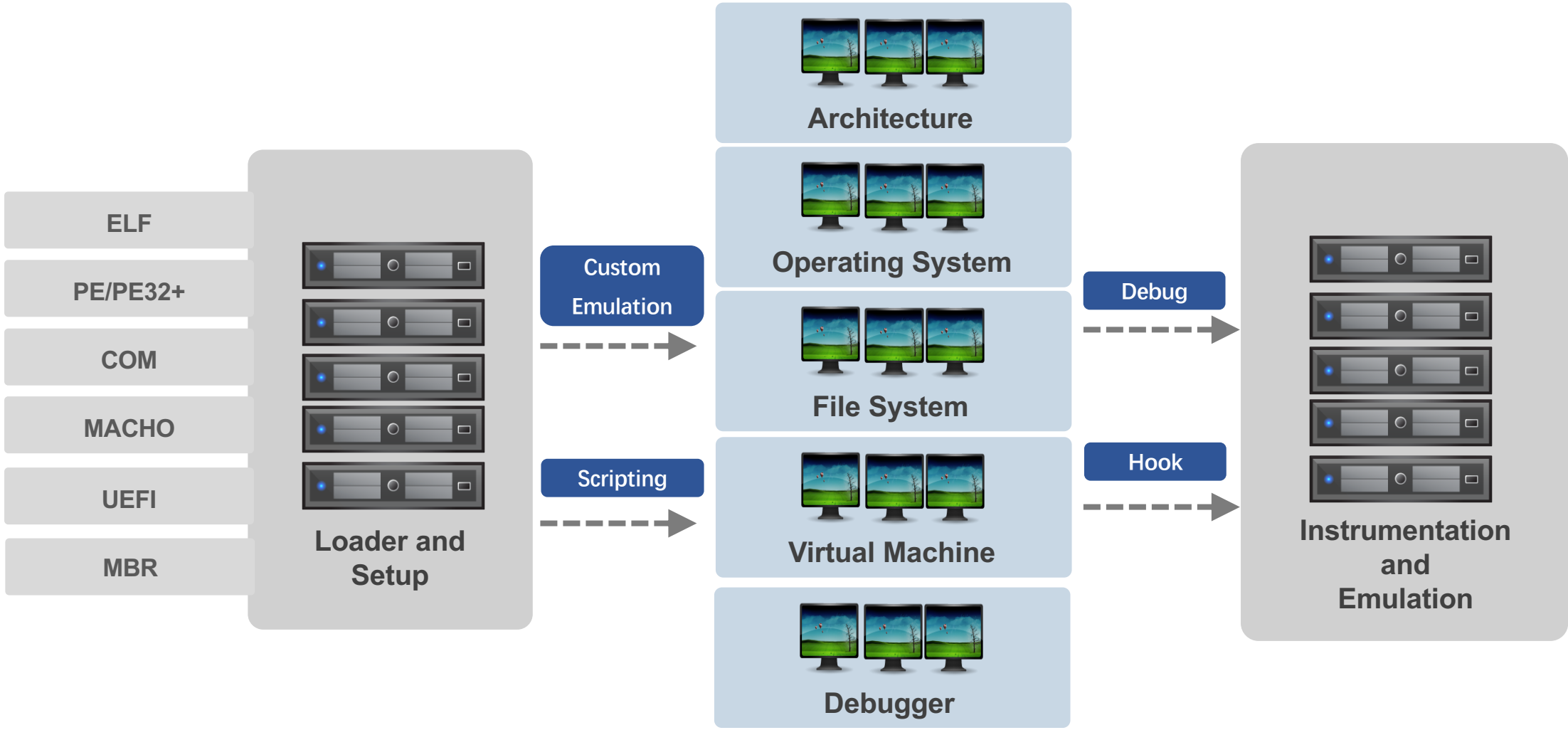
Example: Debugger


```python
def run_sandbox(path, rootfs, output):
    ql = Qiling(path, rootfs, output = output)
    ql.multithread = False
    ql.debugger = "qdb:rr" # switch on record and replay with rr
    # ql.debugger = "qdb:" # enable qdb without options
    ql.run()


if __name__ == "__main__":
    run_sandbox(["rootfs/arm_linux/bin/arm_hello"], "rootfs/arm_linux", "debug")
```

```python
from qiling import *

if __name__ == "__main__":
    ql = Qiling(["rootfs/x8664_linux/bin/x8664_hello"], "rootfs/x8664_linux", output = "debug")
    ql.debugger = "gdb:0.0.0.0:9999"
    ql.run()
```

- Open API for RSP compatible Debugger
- Build In debugger – Qdbg
  - Able to reverse debug

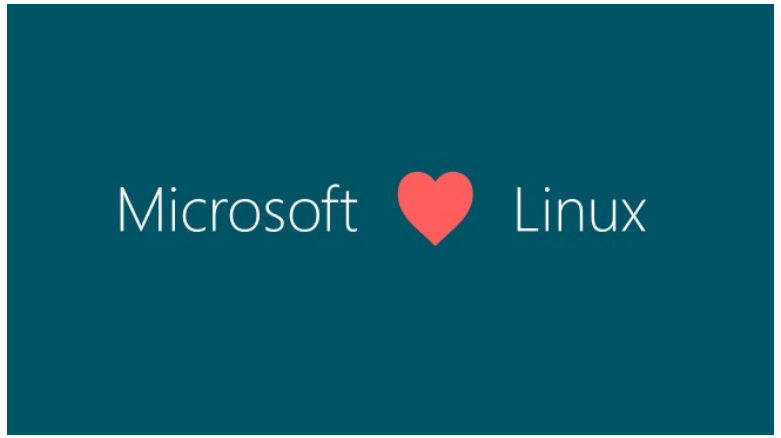
Qiling Framework Software Mode



Demo

Demo Setup

- **ONLY If you wish to try yourself**



- Required OS
  - Ubuntu 18.04 / 20.04
  - WSL2

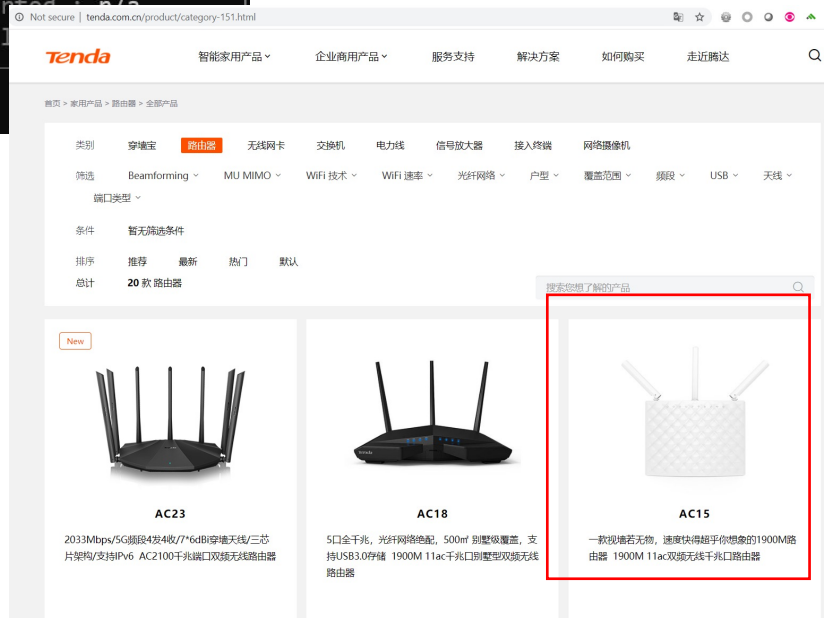
- Install Qiling Framework
  - sudo apt-get update
  - sudo apt-get upgrade
  - sudo apt install python3-pip git cmake build-essential libtool-bin python3-dev automake flex bison libglib2.0-dev libpixman-1-dev clang python3-setuptools llvm
  - git clone https://github.com/qilingframework/qiling.git
  - pip3 install --user https://github.com/qilingframework/qiling/archive/dev.zip

- Install AFL++
  - git clone https://github.com/AFLplusplus/AFLplusplus.git
  - cd AFLplusplus
  - make
  - cd unicorn_mode
  - ./build_unicorn_support.sh

Fuzzer

- Required Firmware
  - AC15
- Run Tenda AC15
  - start_tendaac15_httpd.py
  - Test with browser
- Check crash point
  - addressNat_overflow.sh
- How to find and save snapshot
  - saver_tendaac15_httpd.py
- How to build and run fuzzer
  - fuzz_tendaac15_httpd.py

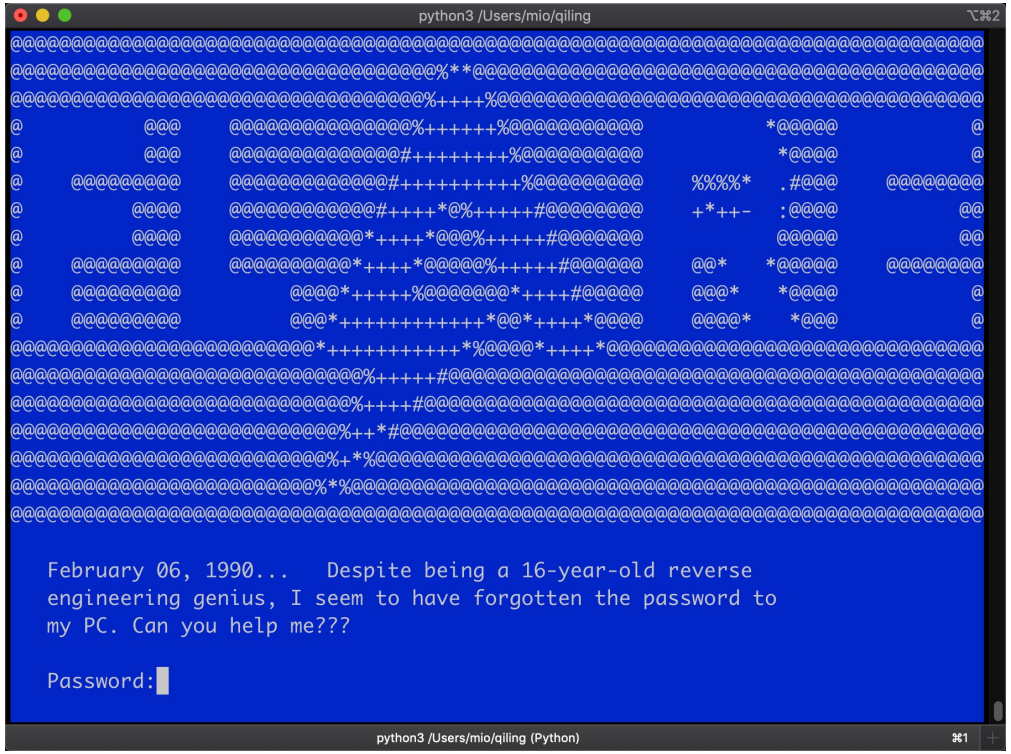
```
        american fuzzy lop ++2.65d (python3) [explore] {0}
┌─ process timing ─────────────────────┬─ overall results ─┐
│       run time : 0 days, 0 hrs, 12 min, 52 sec │  cycles done : 2  │
│  last new path : 0 days, 0 hrs, 0 min, 7 sec   │  total paths : 36 │
│ last uniq crash : 0 days, 0 hrs, 0 min, 15 sec │ uniq crashes : 1  │
│  last uniq hang : none seen yet                │   uniq hangs : 0  │
├─ cycle progress ─────┬─ map coverage ──────────┴───────────────────┤
│  now processing : 21*0 (58.3%)  │    map density : 1.55% / 1.60%   │
│ paths timed out : 0 (0.00%)     │ count coverage : 1.36 bits/tuple │
├─ stage progress ─────┼─ findings in depth ─────────────────────────┤
│  now trying : havoc             │ favored paths : 4 (11.11%)       │
│ stage execs : 2742/32.8k (8.37%)│  new edges on : 8 (22.22%)       │
│ total execs : 122k              │ total crashes : 9 (1 unique)     │
│  exec speed : 161.7/sec         │  total tmouts : 0 (0 unique)     │
├─ fuzzing strategy yields ──────┴─ path geometry ──────────────────┤
│   bit flips : 0/3480, 0/3468, 0/3444 │    levels : 4              │
│  byte flips : 0/435, 0/401, 0/385    │   pending : 25             │
│ arithmetics : 2/23.0k, 0/4022, 0/1454│  pend fav : 0              │
│  known ints : 1/2313, 0/10.5k, 0/16.6k│ own finds : 35            │
│  dictionary : 0/0, 0/0, 0/0          │  imported : n/a            │
│   havoc/rad : 17/48.6k, 1/1312, 0/0  │ stabil                     │
│   py/custom : 0/0, 0/0               │                            │
│        trim : 0.00%/154, 65.57%      │                            │
└──────────────────────────────────────┴────────────────────────────┘
[!] UC
```

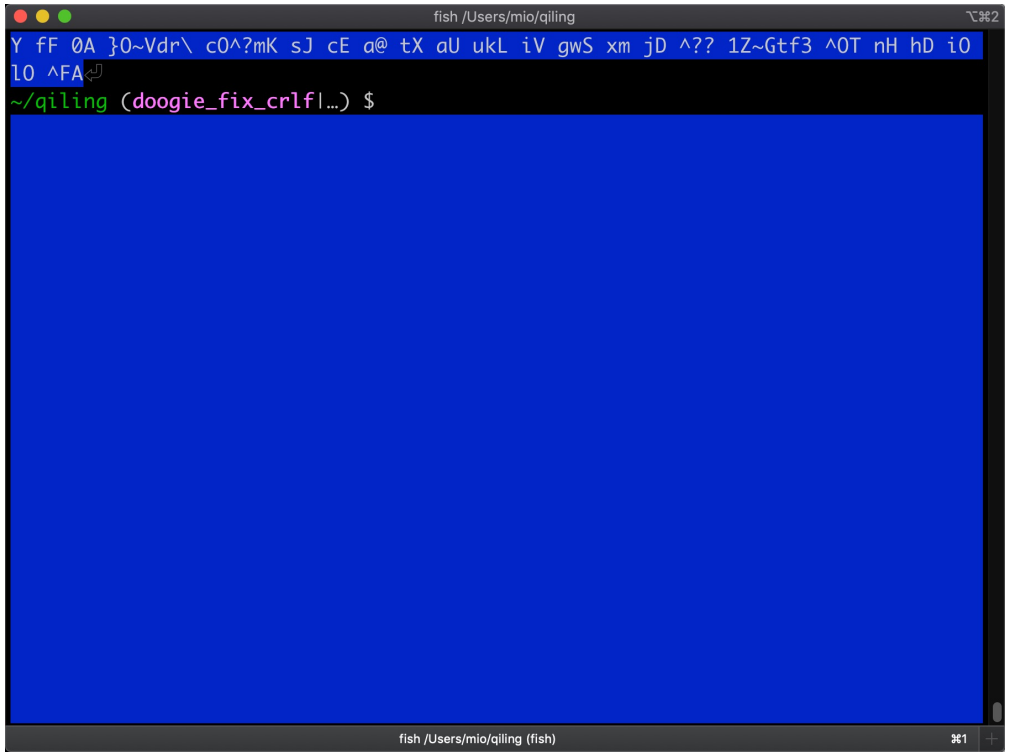


MBR Analysis

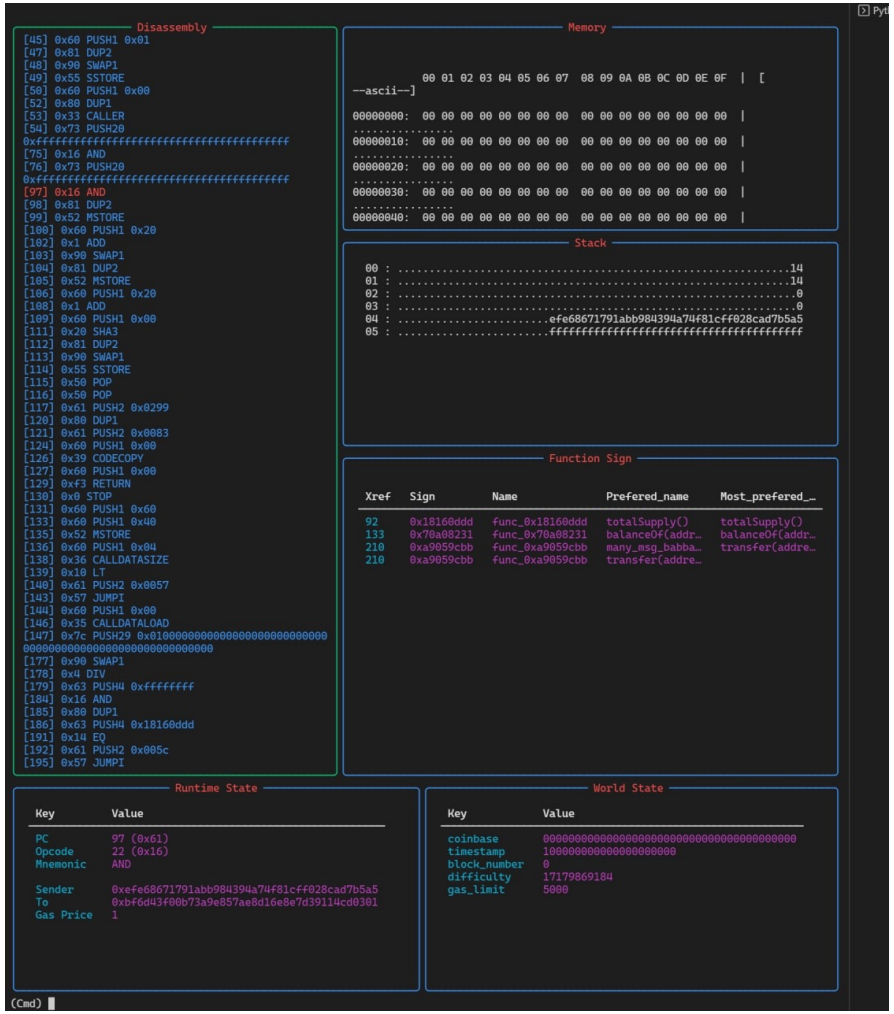
- Sample:
  - Flare-On 5 (2018) Challenge 8 - doogie
- MBR file
- Quick look by qltool.
  - python3 qltool run -f examples/rootfs/8086/doogie/doogie.bin --rootfs examples/rootfs/8086/ --console False
- Try some inputs, but only get gibberish.
- Tips: Feburary 06, 1990.

```
~/q/e/r/8/doogie (doogie|…) $ file doogie.bin
doogie.bin: DOS/MBR boot sector; partition 1 : ID=0x7, activ
e, start-CHS (0x0,32,33), end-CHS (0x3ff,254,63), startsecto
r 2048, 41938944 sectors
~/q/e/r/8/doogie (doogie|…) $
```

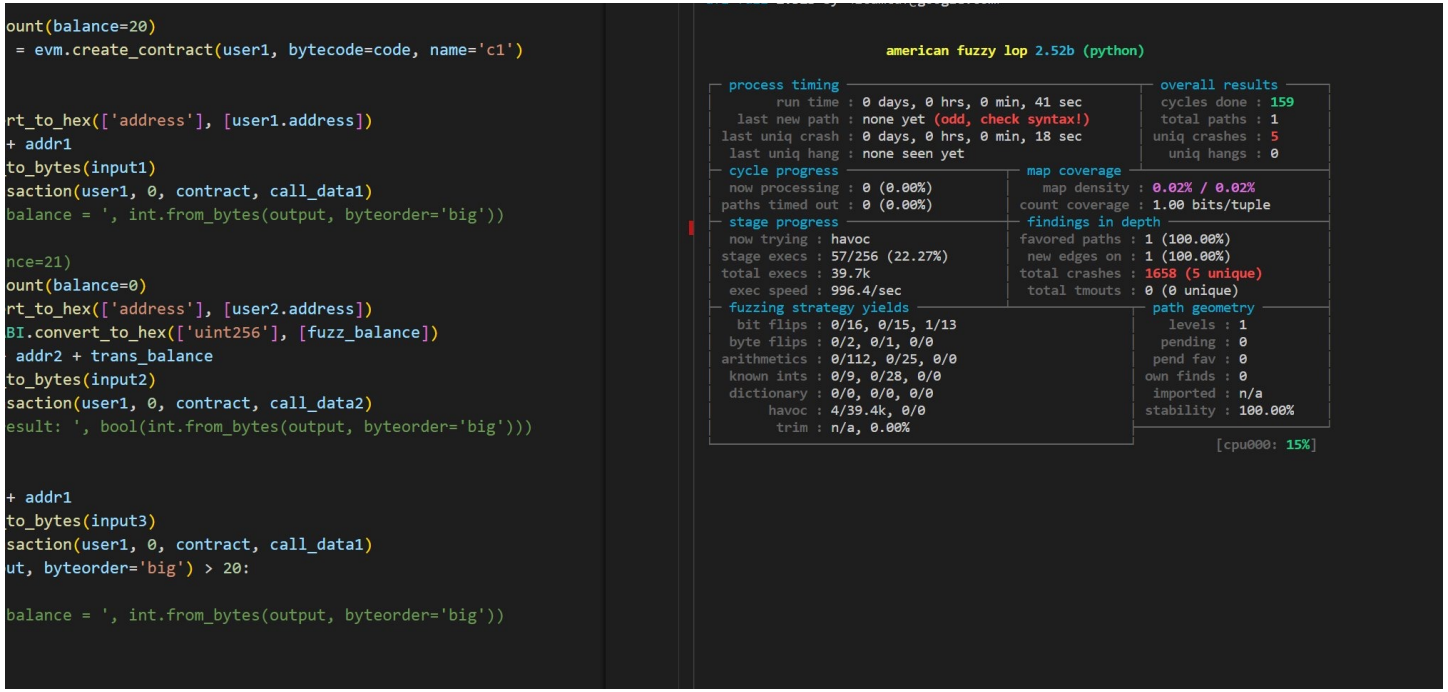




EVM Fuzzer and Debugger



- Real time execution debugger

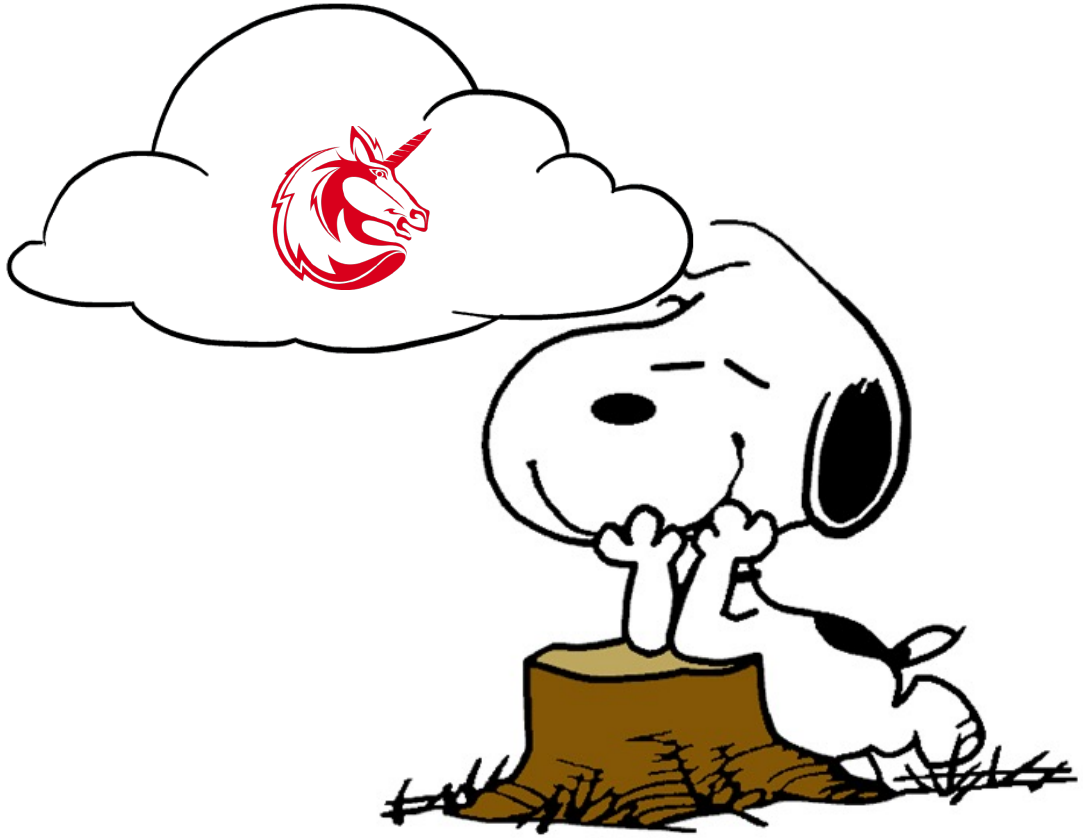


- AFL++
- With Qiling EVM engine

Next Step

Roadmap

- Force Unicorn Engine sync with QEMU 5, Code name **Unicorn 2**
  - More architectures, more CPU instructions set
  - Almost Done
  - **Looking for Release *Sponsor***
- Android Java bytecode layer instrumentation
- **Forward to host implementation**
- iPhoneOS/MacOS/M1 emulation support
- More robust Windows emulation
  - Introduce wine && Cygwin or something
- **ETA: Smart Contract emulation (EVM, soon WASM)**
- MCU emulation

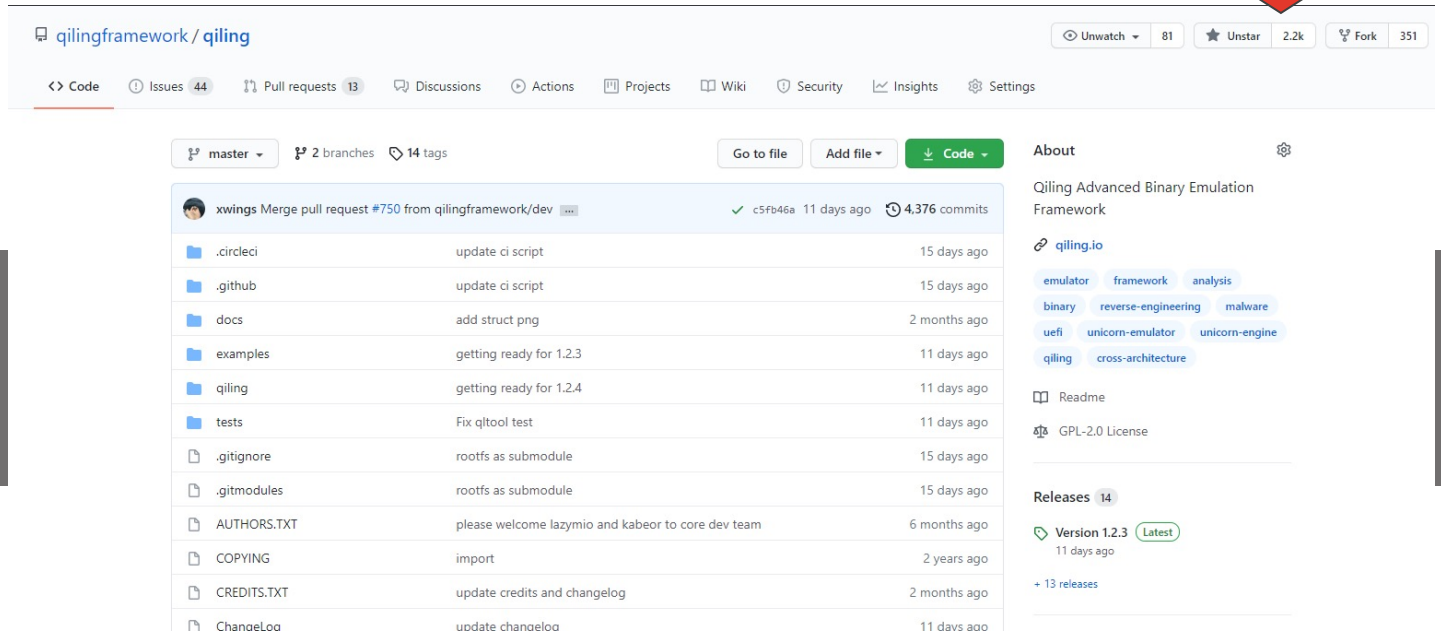


Join Us and Make Pull Request !!!

Everything Else

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

Star us



Questions