uVisor :

tiny
hypervisor/microkernel-like tiny security
kernel at the foundation of mbed OS

Jim Huang (黃敬群) <jserv>
張家榮 <JaredCJR>

開源操作系統技術年會 / Nov 28, 2015

成功大學 National Cheng Kung University

Abstract

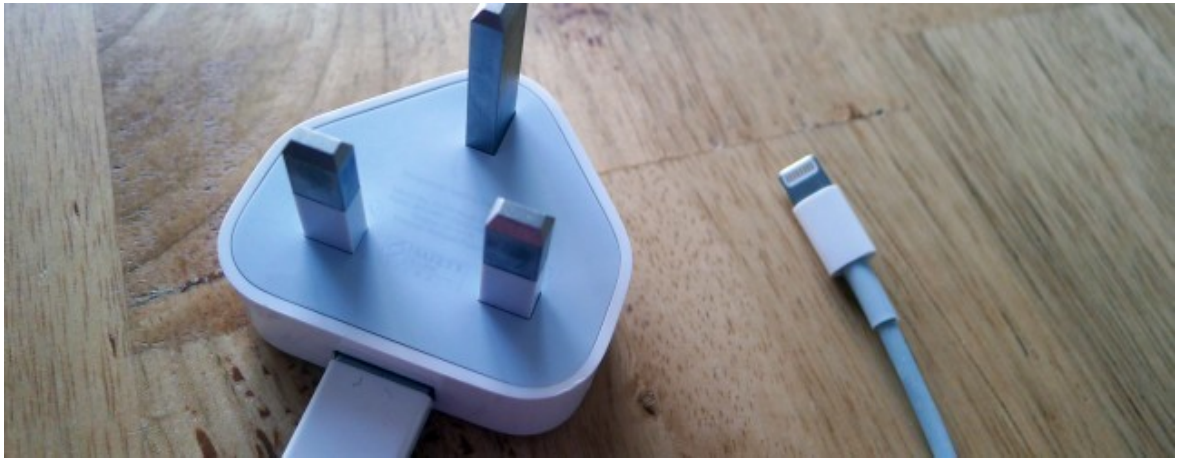
- IoT systems require an effective security framework where application code, protocol stacks, firmware distribution and installation, key provisioning, device management and diagnosis even under targeted attacks.
- This talk presents advanced security features in ARM mbed OS for ARM Cortex-M processor to secure firmware updates and the cloud communication.
  - how memory protection unit (MPU) is used in practice by developers on mbed OS to compartmentalize code and sensitive data while accelerating development.

Attack!

你被 USB 充電裝置出賣

Attack iOS through USB charger!

- BlackHat 2013
  - MACTANS: INJECTING MALWARE INTO IOS DEVICES VIA MALICIOUS CHARGERS
  - http://www.blackhat.com/us-13/briefings.html#Lau
- "we demonstrate how an iOS device can be compromised within one minute of being plugged into a malicious charger. We first examine Apple's existing security mechanisms to protect against arbitrary software installation, then describe how USB capabilities can be leveraged to bypass these defense mechanisms."



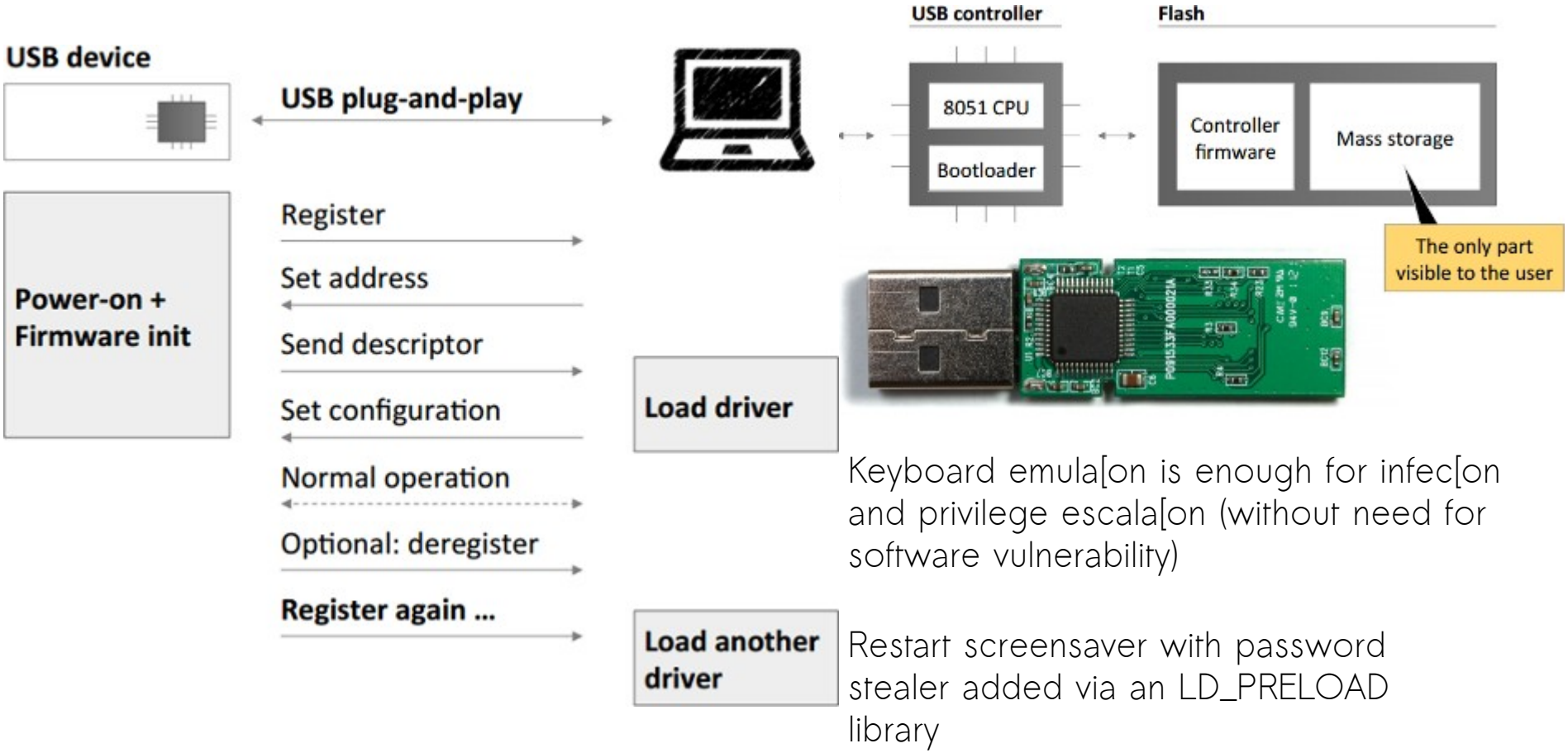
背叛你的 USB Mass Storage 裝置

Plug and "Pray"

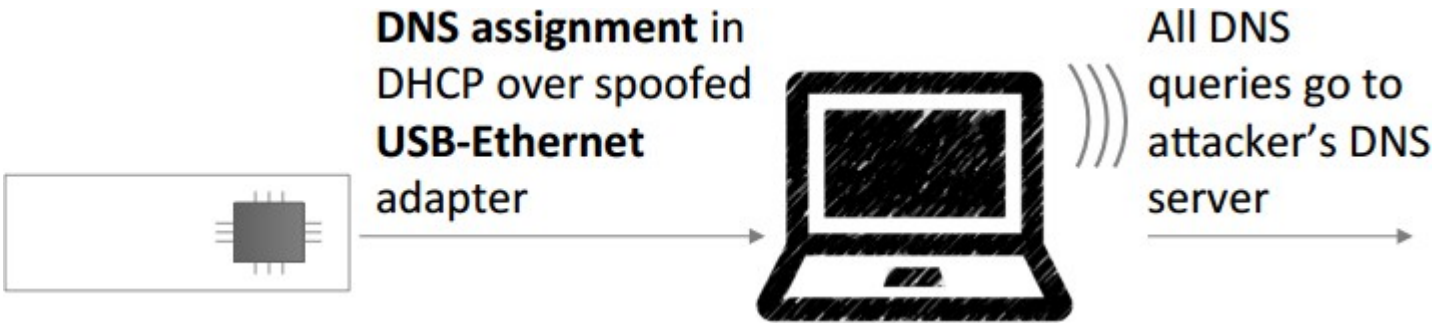
- BlackHat 2014

  BadUSB — On accessories that turn evil

  https://srlabs.de/badusb/



實體和網路協同攻擊



- USB 裝置偽裝為 Ethernet 裝置
- 在 DHCP 往返的過程中，給定惡意的 DNS 伺服器，但電腦端仍保持連線，沒察覺到 DNS 設定已變更
  - redirection attack

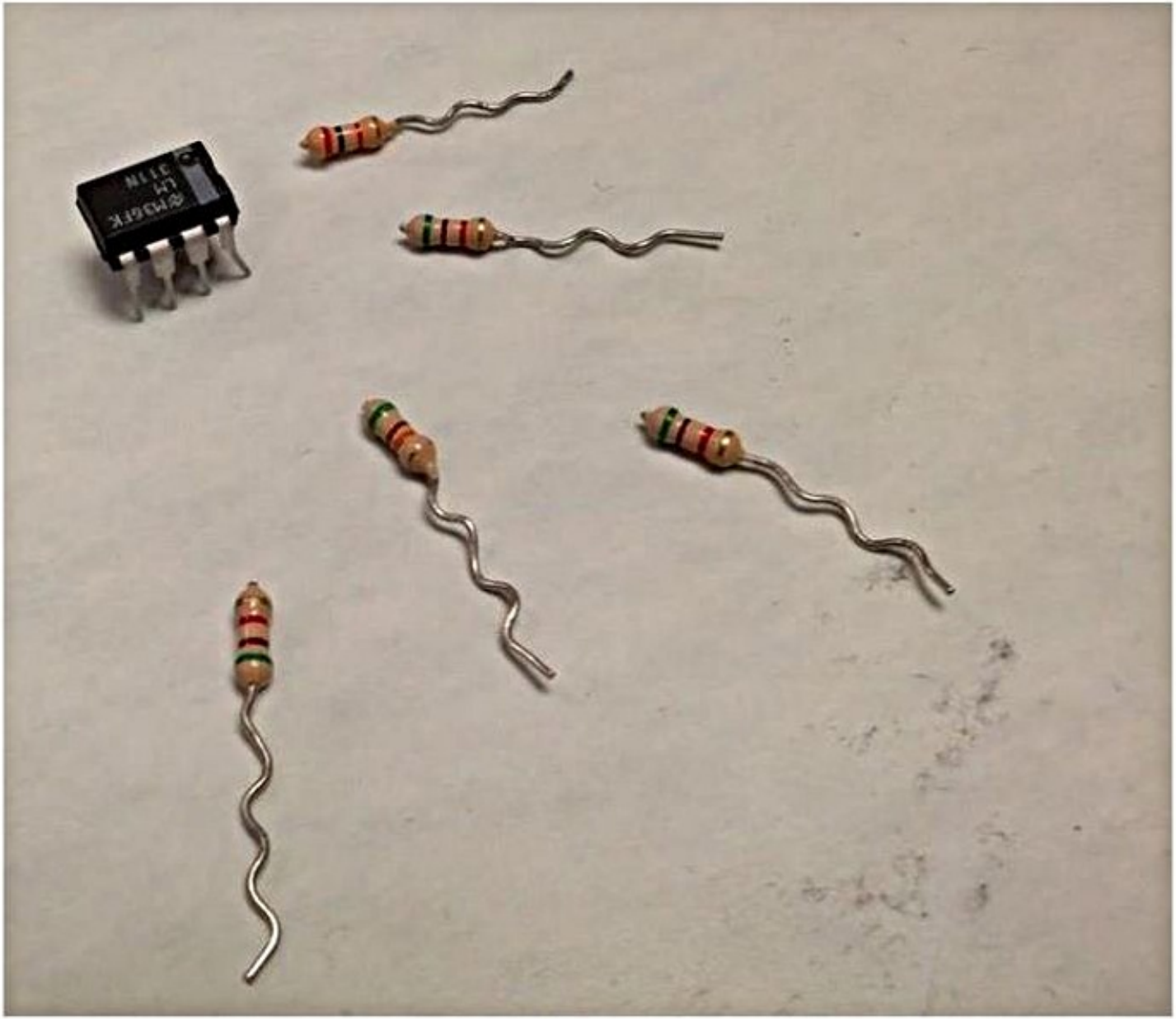
- USB Redirection via RDP

  Easy Print / Drive Redirection / Smart Card Redirection

  Plug-and-Play Device Redirection / Input Redirection / Audio Redirection / Port Redirection

Source:USB attacks need physical access right? Andy Davis

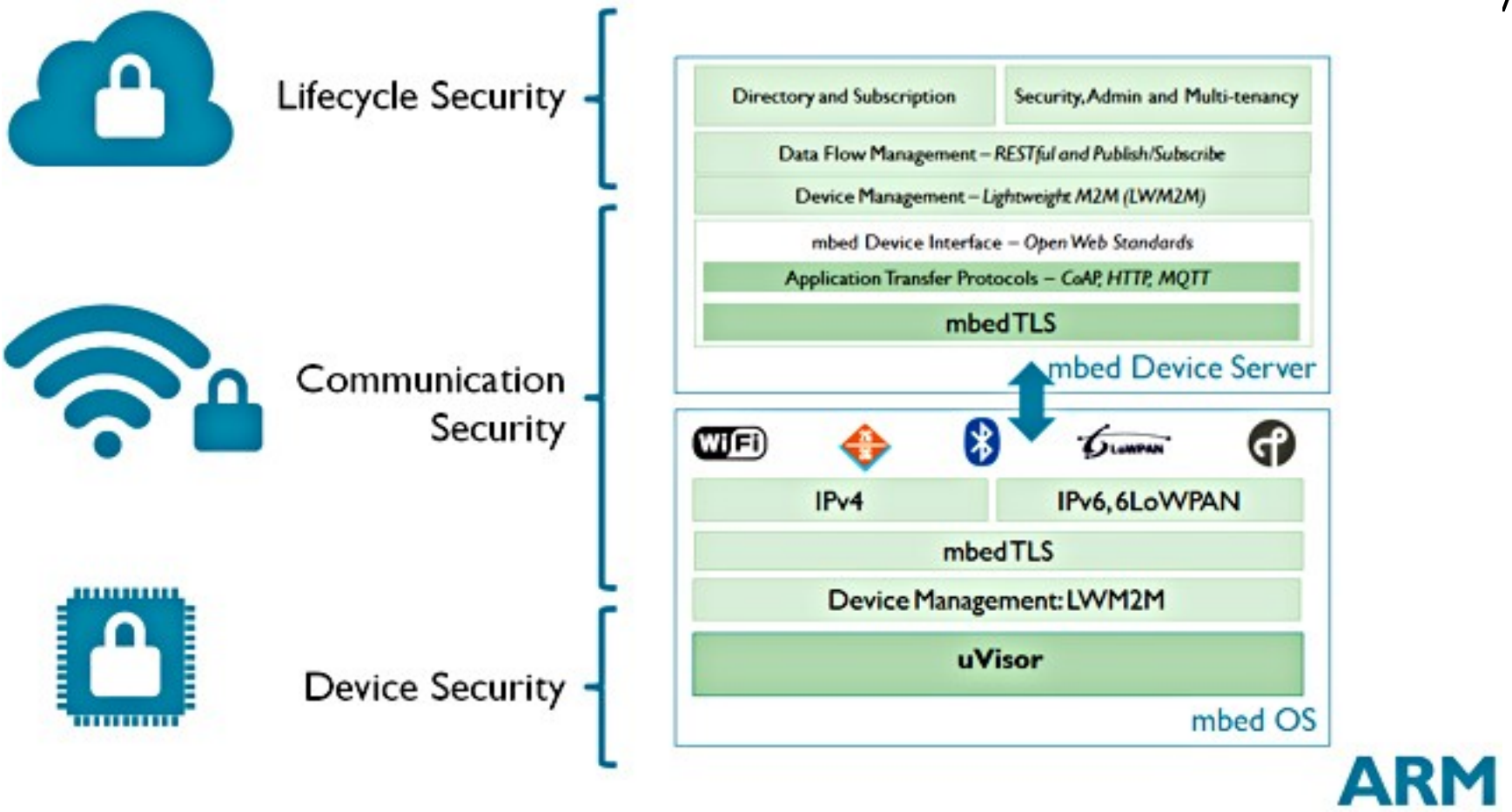
軟件
[固件]
硬件



ARM Cortex-M profile: Deeply Embedded Devices

- Power awareness; solid and limited applications
- Multi-tasking or cooperative scheduling is still required
- IoT (Internet of Things) is the specialized derivative with networking facility
- Communication capability is built-in for some products
- Example: AIRO wristband (health tracker)

IoT Security



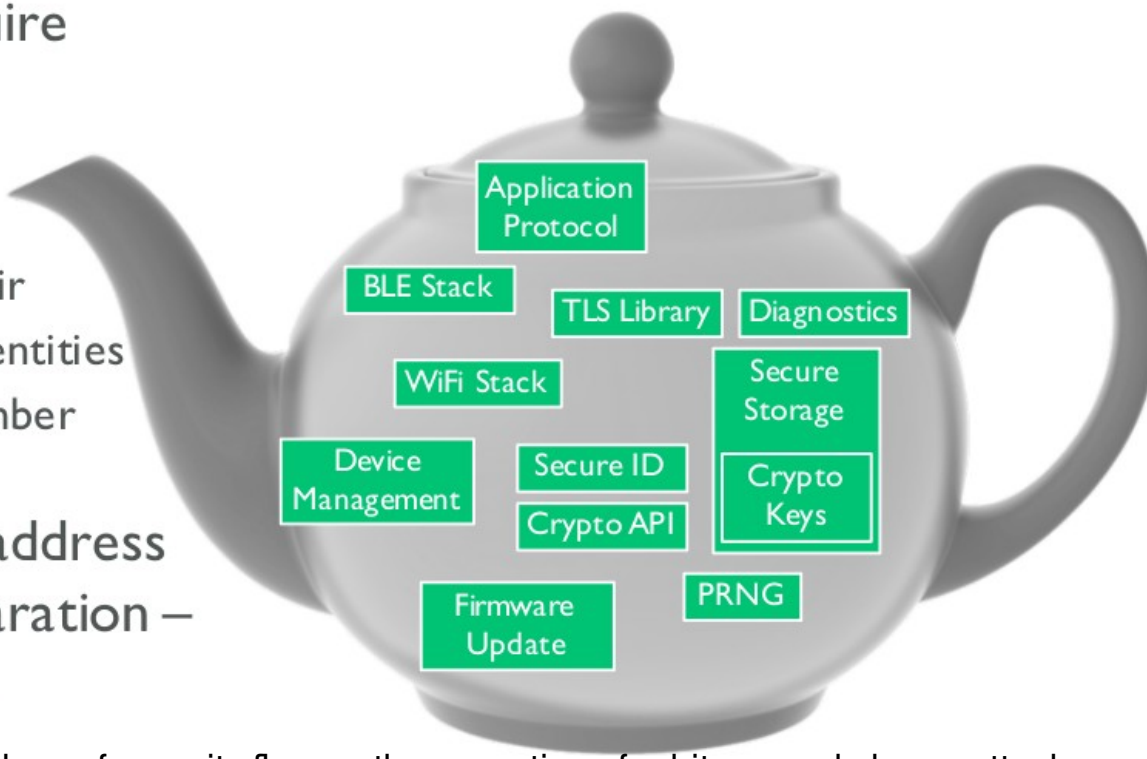
- ARM mbed OS
- ARM mbed uVisor
- ARM mbed TLS

- TrustZone in ARMv8-M
- security lifecycle management
- Apache License

- because of the huge amount of code involved in maintaining WiFi connections or enabling ZigBee or BLE communication, the resulting attack surface is almost impossible to verify and therefore compromises device security

IoTeapot "Hello World" Example – The Attacker View

- Even simple IoT products require complex components
  - Secure server communication over complex protocols
  - Secure firmware updates over the air
  - Unclonable cryptographic device identities
  - Cryptography APIs and random number generation
- Existing IoT solutions use flat address spaces with little privilege separation – especially on microcontrollers

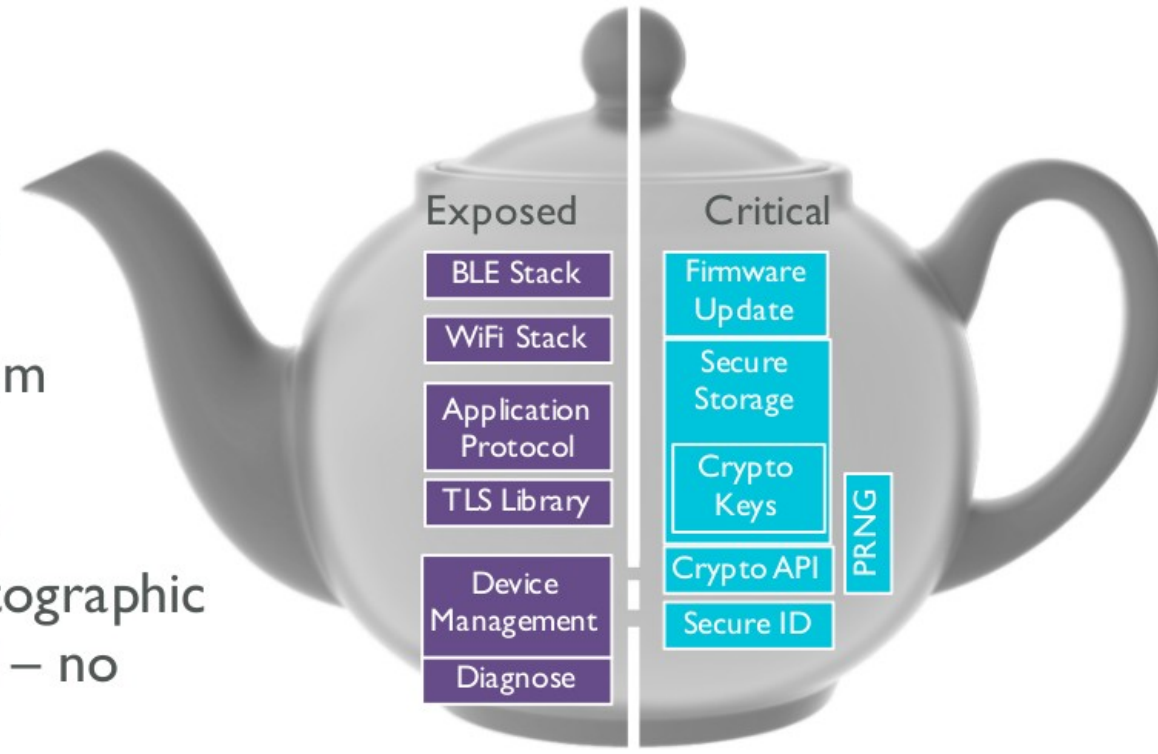


- the recovery from a common class of security flaws – the execution of arbitrary code by an attacker
  - Even a hardware-enforced root of trust and a secure boot loader will not fix that problem: the resident malware can run safely from RAM and block reset commands or flash erasing as part of a denial-of-service attack.

Source: ARM TechCon 2015

IoTeapot "Hello World" Example – Mitigation Strategies

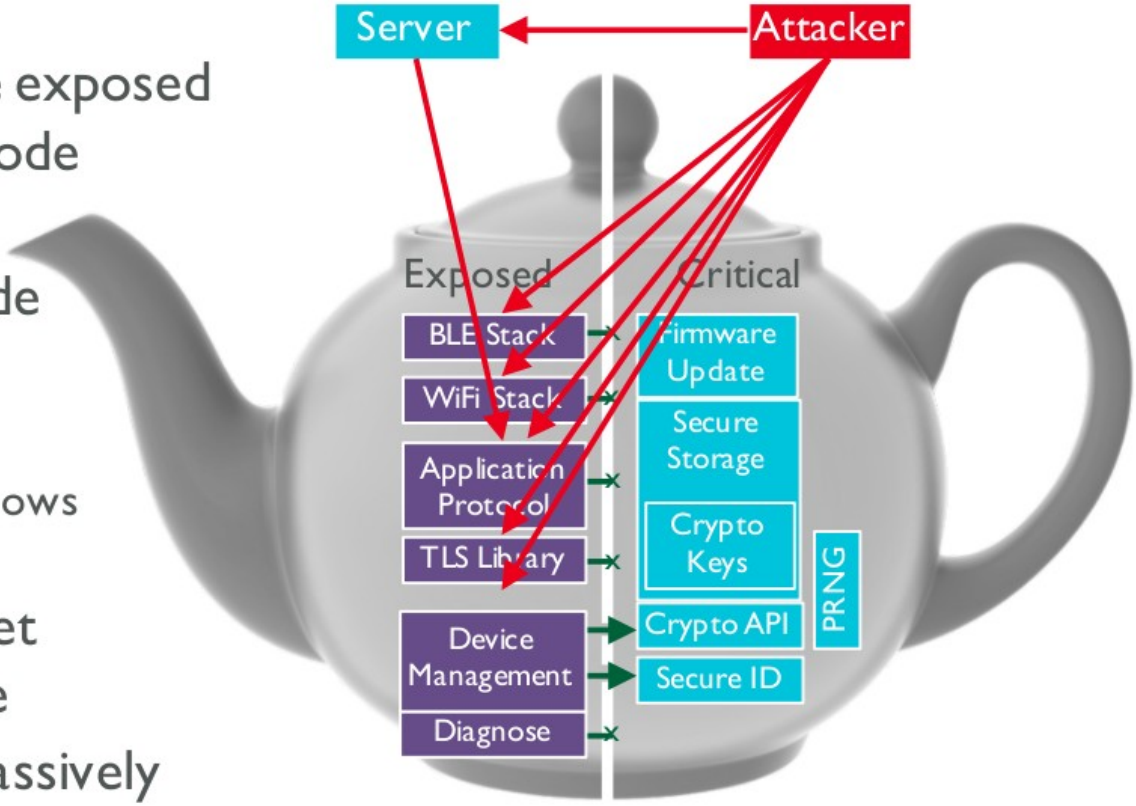
- Split security domains into
  - exposed uncritical code
  - protected critical code
- Keep footprint of critical code small to enable verification
- Protect key material and system integrity using the ARMv7-M hardware memory protection
- Public code operates on cryptographic secrets via defined private API – no access to raw keys



Source: ARM TechCon 2015

IoTeapot "Hello World" Example – Mitigation Strategies

- Attackers can compromise the exposed side without affecting critical code
- Using cryptographic hashes the integrity of the exposed side can be verified
  - Triggered on server request
  - Protected security watchdog box allows remote control
- Protected side can reliably reset exposed boxes to a clean state
- The device attack surface is massively reduced as a result



Source: ARM TechCon 2015

建構在 uVisor 之上的安全體系

Security Functionality:

- Cryptography
- Key Management
- Secure FW Upgrade
- Secure Identity
- Security Monitoring

Isolated

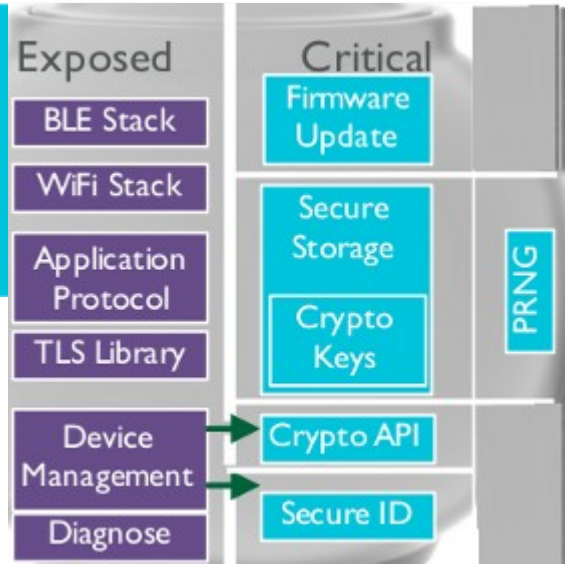
Strong Separation

Remainder of mbed OS:

- HAL + Drivers
- Scheduler
- Connectivity Stack(s)
- Device Management
- User Application Code and Libraries

Non-critical

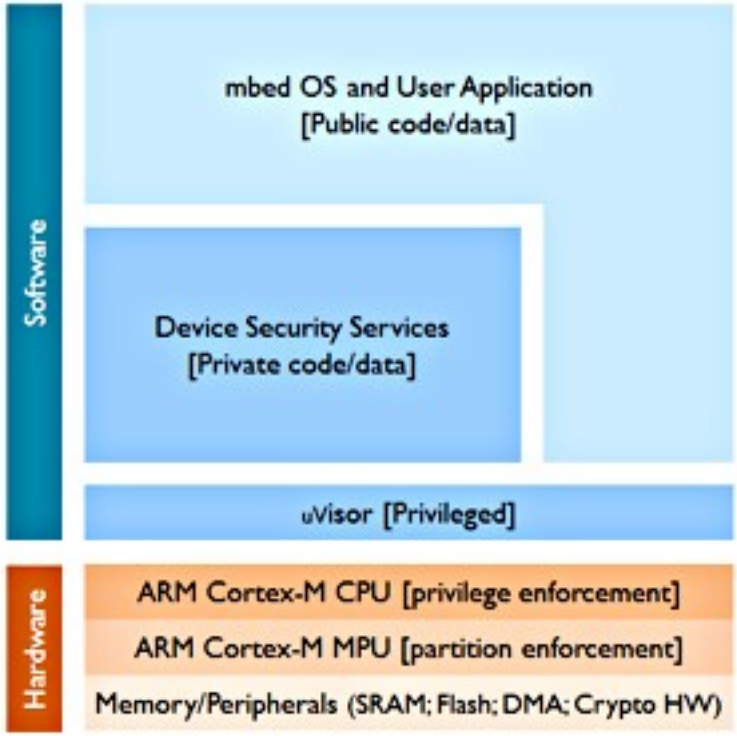
uVisor



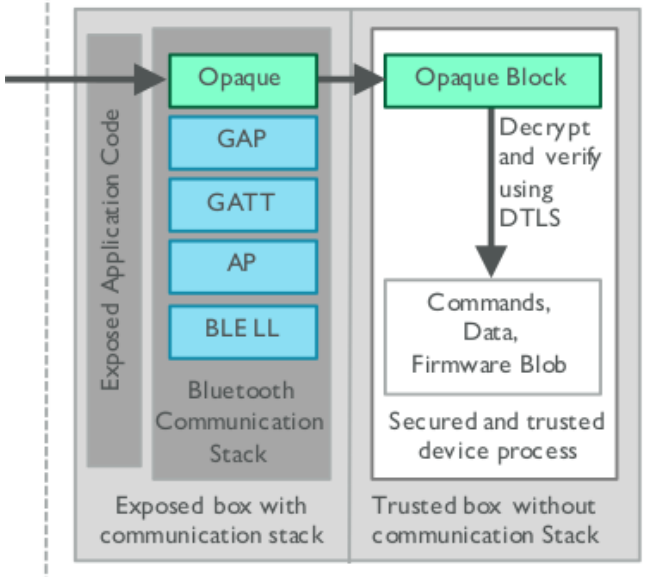
uVisor Design Principles

- Hardware-enforced security sandboxes

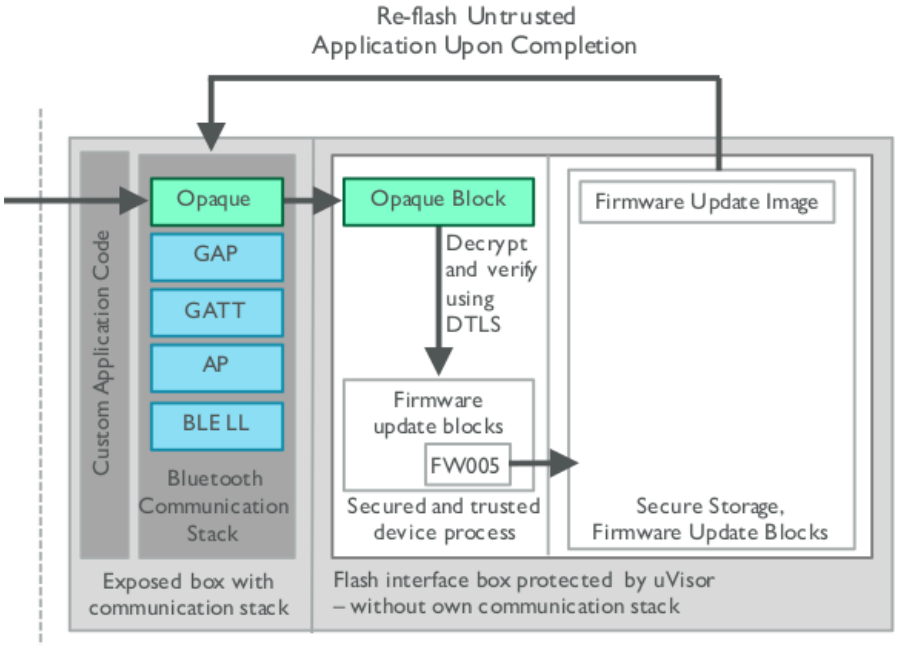
- Mutually distrustful security model
  - "Principle of Least Privilege"
  - Boxes are protected against each other
  - Boxes protected against malicious code from
  - broken system components, driver or other boxes
- Enforce API entry points across boxes
  - Box-APIs can be restricted to specific boxes
- Per-box access control lists (ACL)
  - Restrict access to selected peripherals
  - Shared memories for box-box communication



- Box security not affected by communication stack exploits or infections outside of trusted box
- Resilient box communication over the available channels
  - Ethernet, CAN-Bus, USB, Serial
  - Bluetooth, Wi-Fi, ZigBee, 6LoWPAN



IoT Device owned by user.
Initial identity provisioned by System Integrator
Messages delivered agnostic of communication stack



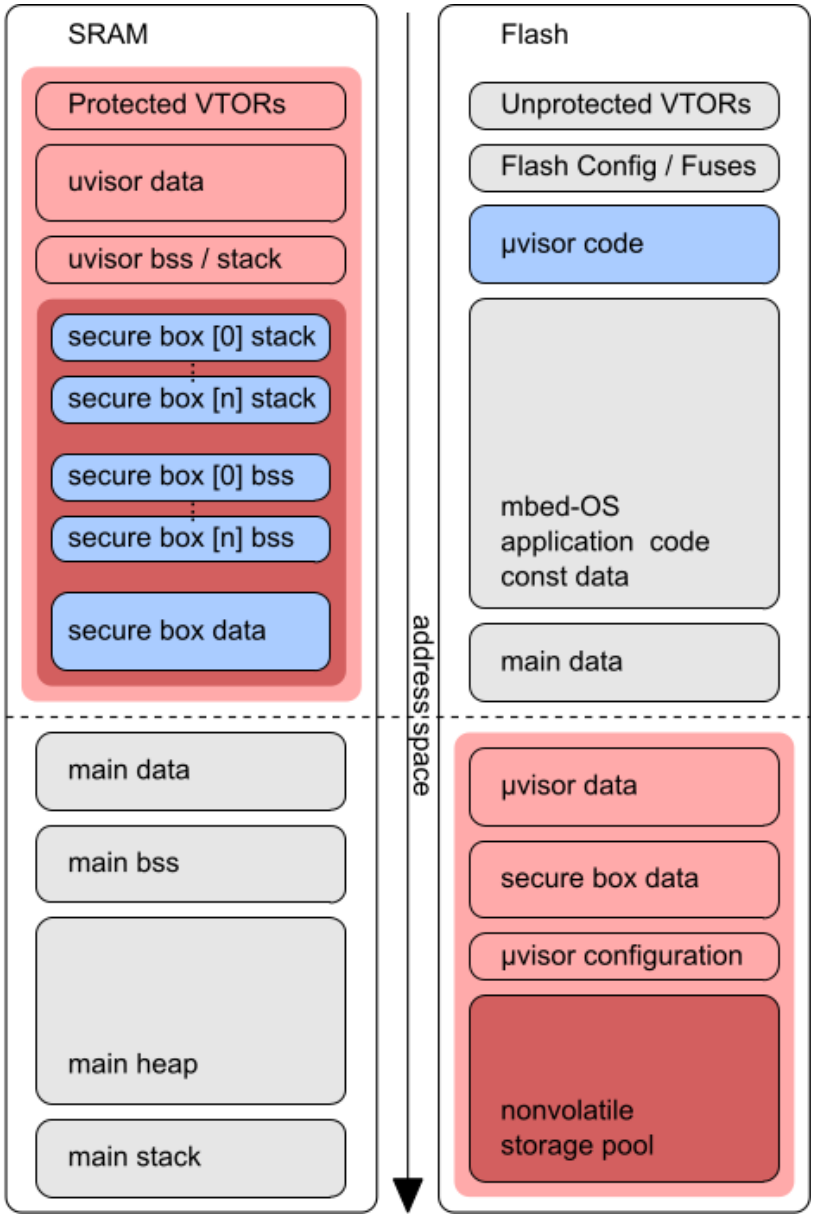
IoT device owned by user,
Initial identity provisioned by System Integrator,
Messages delivered independent of stacks

- Firmware manifest block augments existing firmware formats with safety and security features
- Crypto watchdog box enforces remote updates even for infected devices

Source: ARM TechCon 2015

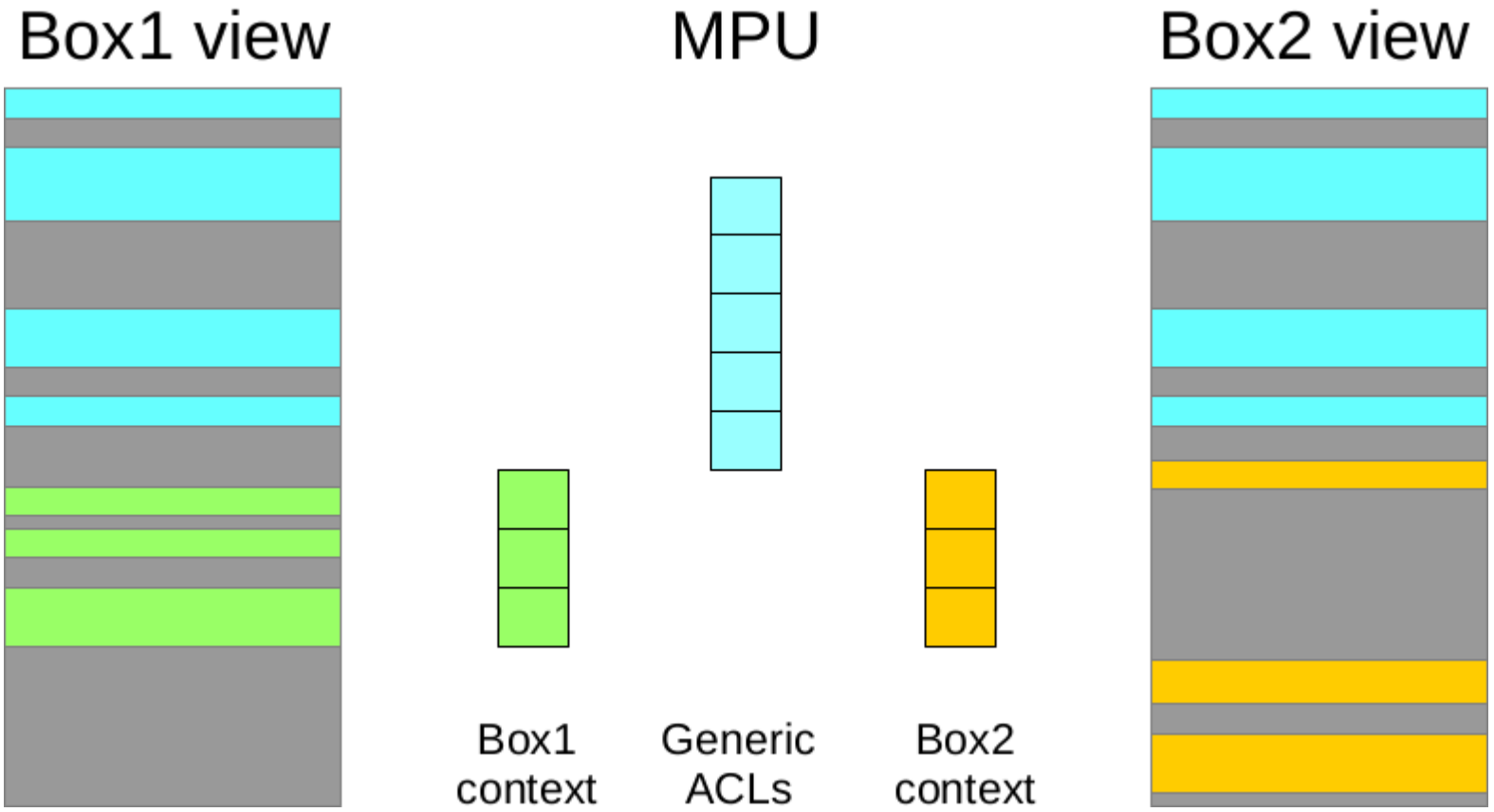
uVisor Memory Model

- uVisor allocates protected per-box stacks and detects under-/overflows during operation
- Main box memory accessible to all boxes
- All remaining per-Box data sections are protected by default:
  - Secure Per-Box Context Memory
  - Shared data/peripherals with other boxes on demand
  - uVisor resolves ACLs during boot and identifies ACL collisions
- uVisor code sections visible to everybody
- Empty flash memory is made available to the system as configuration storage – write access only through configuration API

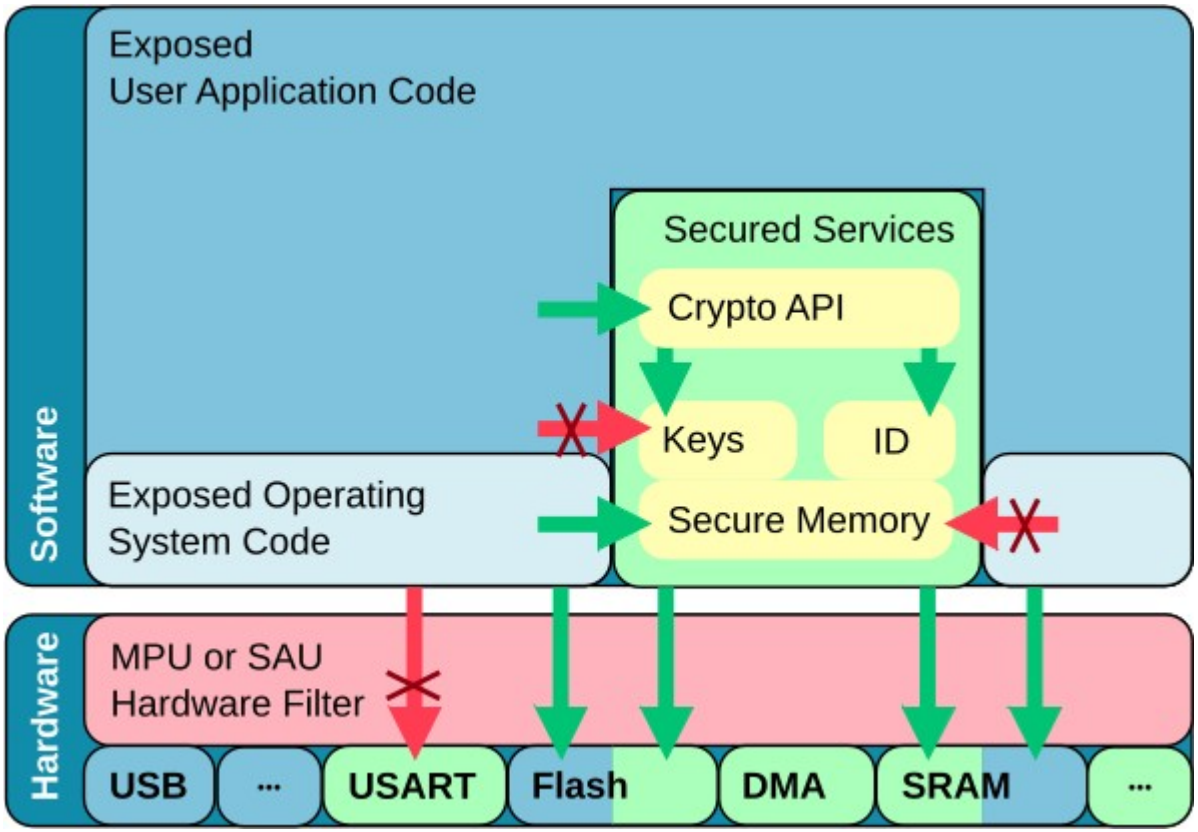


How ACLs are implemented

- ACLs and Box contexts isolation are implemented via MPU



uVisor Boot Sequence (ARMv7-M)



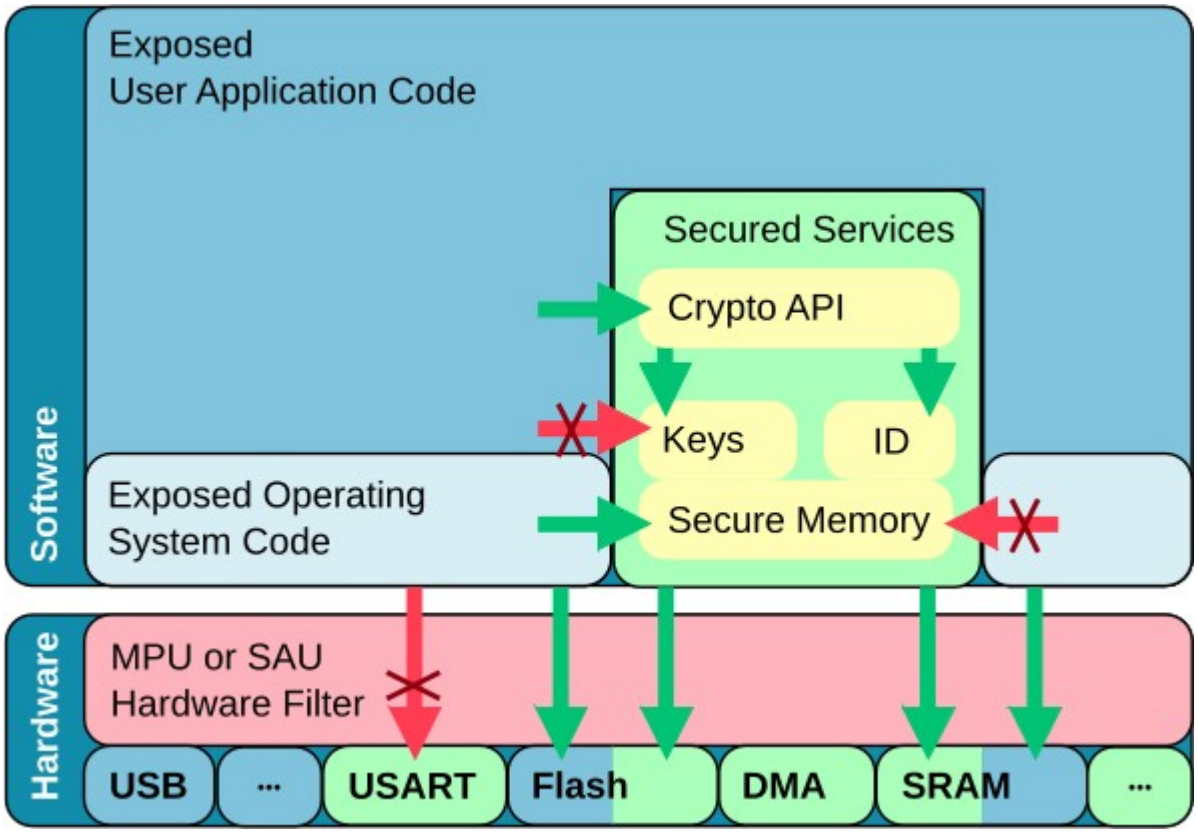
uVisor initialized first in boot process
→ Private stack and data sections
→ Private data sections in flash for storing secrets

Initialization of memory protection unit based on box ACL's
→ only necessary peripherals are accessible to box
→ Each box has private .bss data and stack sections

Relocation of interrupts vector table into secure memory

```
#include <uvisor-lib/uvisor-lib.h>
/* create background ACLs for the main box */
static const UvBoxAclItem g_background_acl[] = {
{UART0, sizeof(*UART0), UVISOR_TACL_PERIPHERAL},
{UART1, sizeof(*UART1), UVISOR_TACL_PERIPHERAL},
{PIT, sizeof(*PIT),UVISOR_TACL_PERIPHERAL}, };
UVISOR_SET_MODE_ACL(UVISOR_ENABLED, g_background_acl); /* set uvisor mode */
```

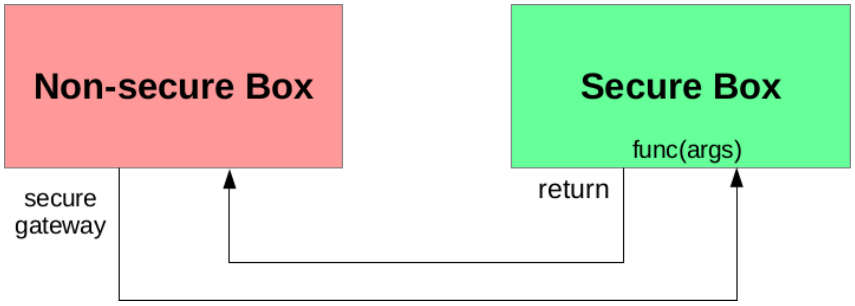
Protected Sandbox




```
/* private box context */
typedef struct {
uint8_t secret[SECRET_SIZE];
bool initialized;
} BoxContext;*/
```

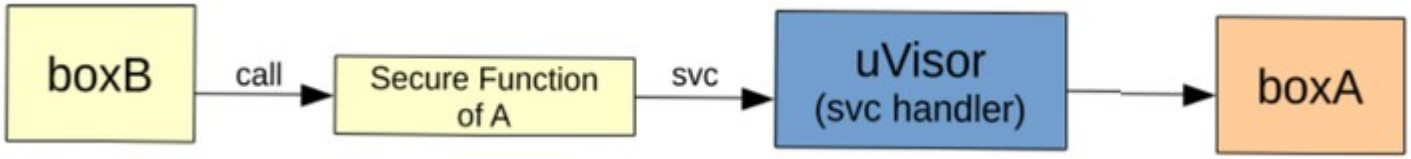
```
/* create ACLs for the module */
static const UvBoxAclItem g_box_acl[] = {
    {RNG, sizeof(*RNG), UVISOR_TACL_PERIPHERAL},
};

/* configure secure box compartment */
UVISOR_BOX_CONFIG(my_box_name, g_box_acl,
                  0x100 /* required stack size */, BoxContext);
```

Call Gateway

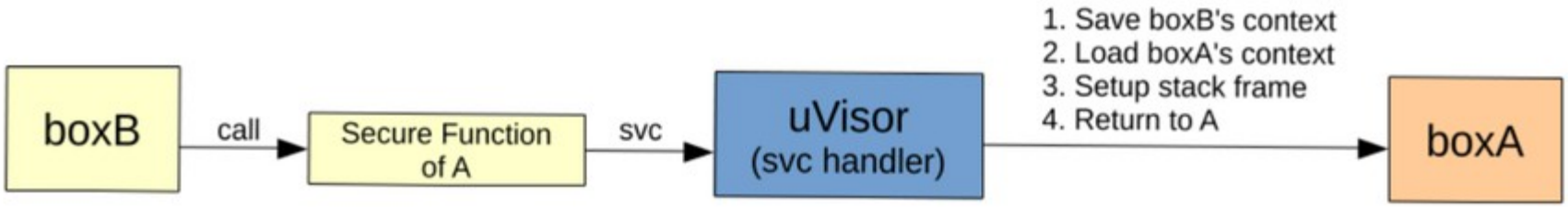


- Call gateways only accepted from flash memory
  - attacker has no write access to flash controller
- Metadata of call gateway at a fixed offset from uVisor gateway context switch – a supervisor call (SVC)
  - Contains pointer to target box configuration & target function
  - Guaranteed latency for cross-box calls
- Can limit access to specific caller boxes
- Security verified once during installation

```
/* the actual secure gateway */
#define secure_gateway(dst_box, dst_fn, ...)
({
    SELECT_ARGS(__VA_ARGS__)
    register uint32_t res asm("r0");
    asm volatile (
          "svc   UVISOR_API_SVC_CALL_ID\n"
          "b.n   skip_metadata%=\n"
          ".word UVISOR_SVC_GW_MAGIC\n"
          ".word dst_fn\n"
          ".word dst_box##_cfg_ptr\n"
          "skip_metadata%=:\n"
          : "=r" (res)
          : ASM_INLINE_ARGS(__VA_ARGS__)
    );
    res;
})
```



Function Call Gateway via SVC

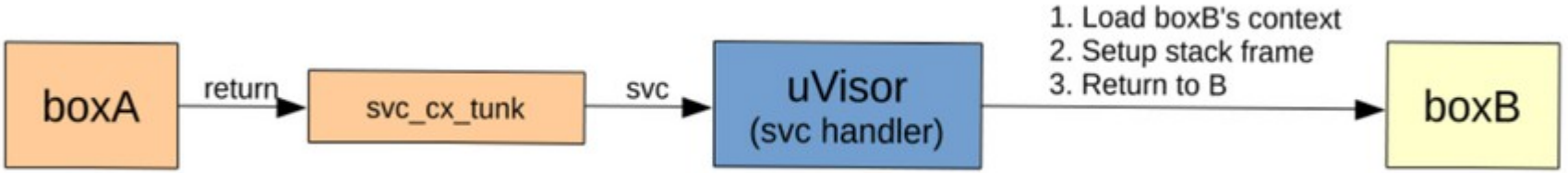


1.boxB 呼叫 boxA 所提供的 secure function，此時系統在 Unprivileged mode，正執行 boxB 的 context

2.secure function 裡會使用 svc 進入 uVisor 裡的 svc handler，此時系統在 Privileged mode

3.在 svc handler 裡做 context switch，從 boxB 切換至 boxA (`svc_cx_switch_in`)

4. svc handler 結束後切換回 Unprivileged mode，並將 stack frame 裡的內容寫回 registers 裡，因此 PC 會指向 boxA 的實現函數



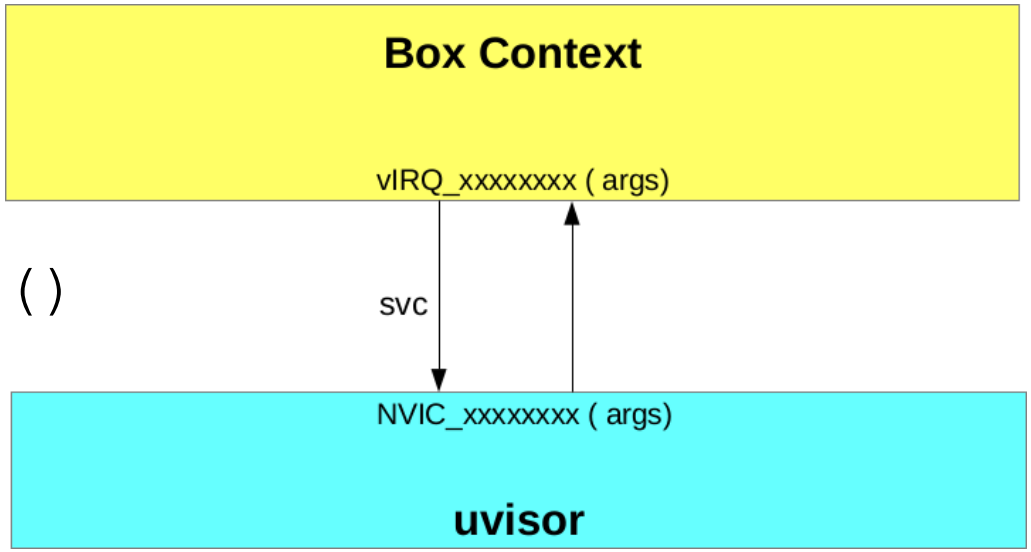
1.boxA 實現函數結束後，由於 LR 被設為 `svc_cx_tunk` 的位址，因此隨後會返回至 `svc_cx_tunk`。執行 `svc_cx_tunk` 之際再次呼叫 svc，回到 uVisor 裡的 svc_handler

2.在 svch_hadler 裡載入 boxB 的 context，從 boxA 的 context 切換回 boxB

3.svc handler 結束後返回 boxB

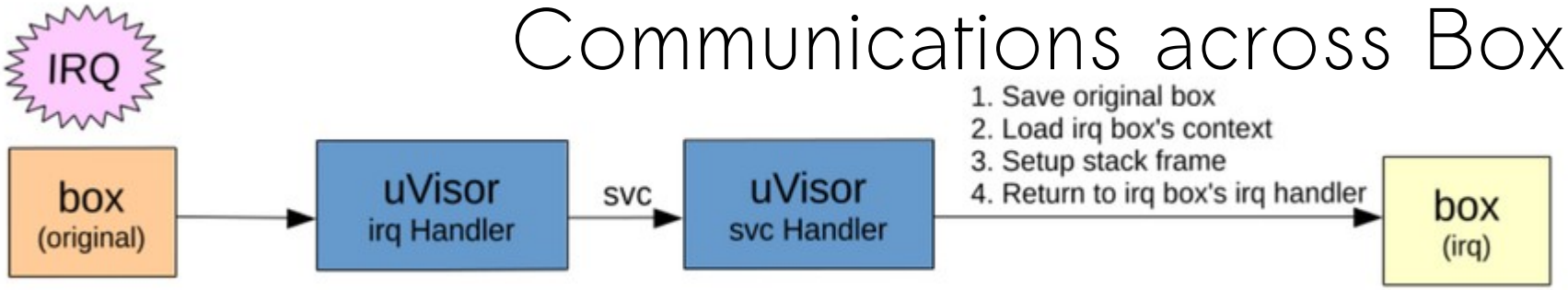
Interrupt Management APIs

- `vIRQ_SetVectorX()`
- `vIRQ_GetVector()`
- `vIRQ_EnableIRQ()`
- `vIRQ_DisableIRQ()`
- `vIRQ_ClearPendingIRQ()`
- `vIRQ_SetPendingIRQ()`
- `vIRQ_GetPendingIRQ()`
- `vIRQ_SetPriority()`
- `vIRQ_GetPriority()`
- `vIRQ_GetLevel()`



Interrupt Forwarding

Communications across Boxes



發生於 Interrupt Forwarding. uVisor 在其中最重要的任務即是將 irq 的處理轉發至相對應的 box. 流程如下 :

1.Interrupt 發生在執行某個 box 的時候

2.進入 uVisor 預設的 interrupt handler

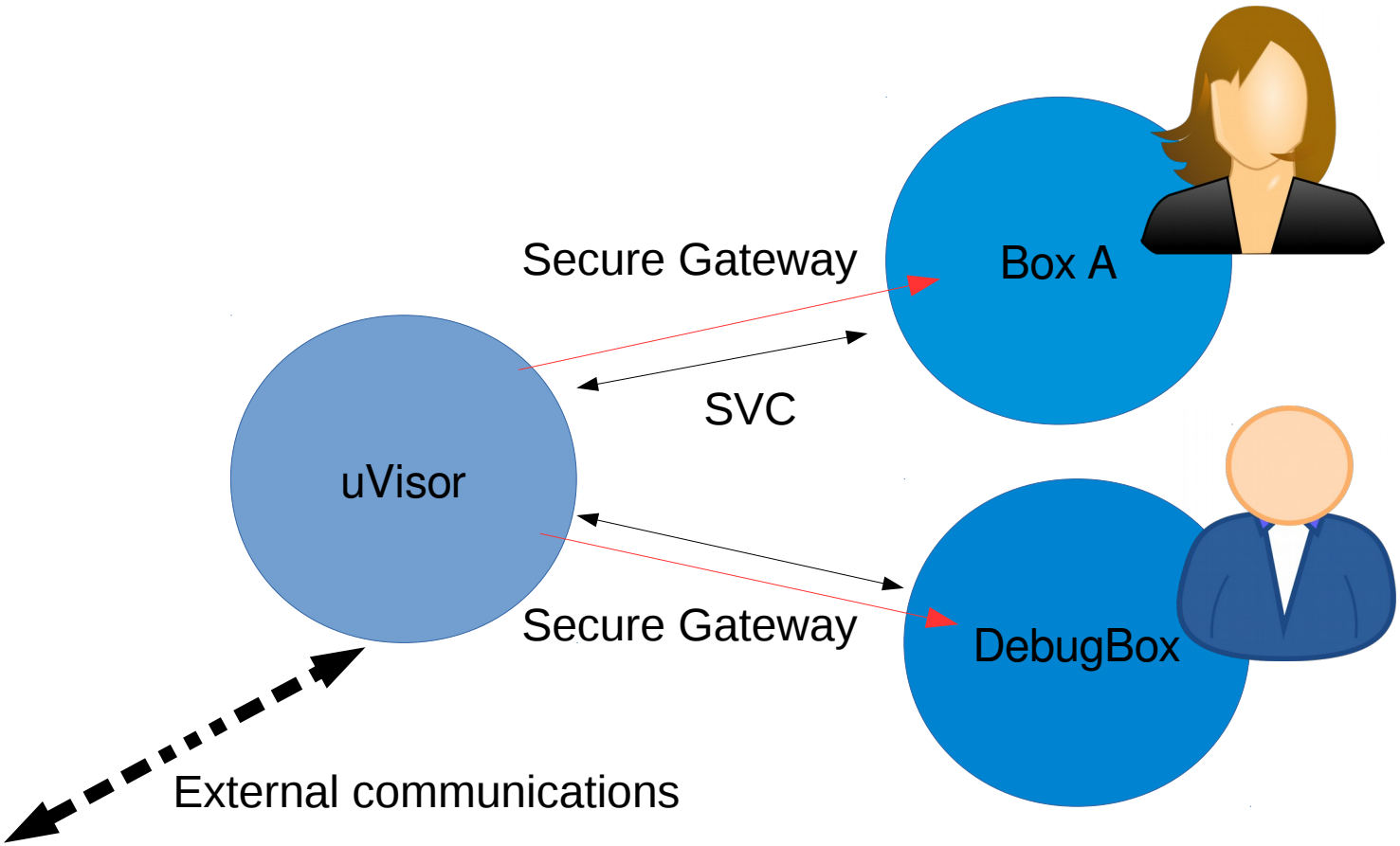
```
static inline __attribute__((always_inline))
              void unvic_isr_mux(void)
{
    asm volatile(
        "svc  %[unvic_in]\n"
        "svc  %[unvic_out]\n"
        "bx   lr\n"
        ::[unvic_in]  "i" (UVISOR_SVC_ID_UNVIC_IN),
          [unvic_out] "i" (UVISOR_SVC_ID_UNVIC_OUT)
    );
}
```

1.box irq handler 執行結束後 , 返回 lr 指的位址 , uVisor irq handler 裡的第二個 svc, 要注意的是 , 此時仍屬於 unprivileged mode.

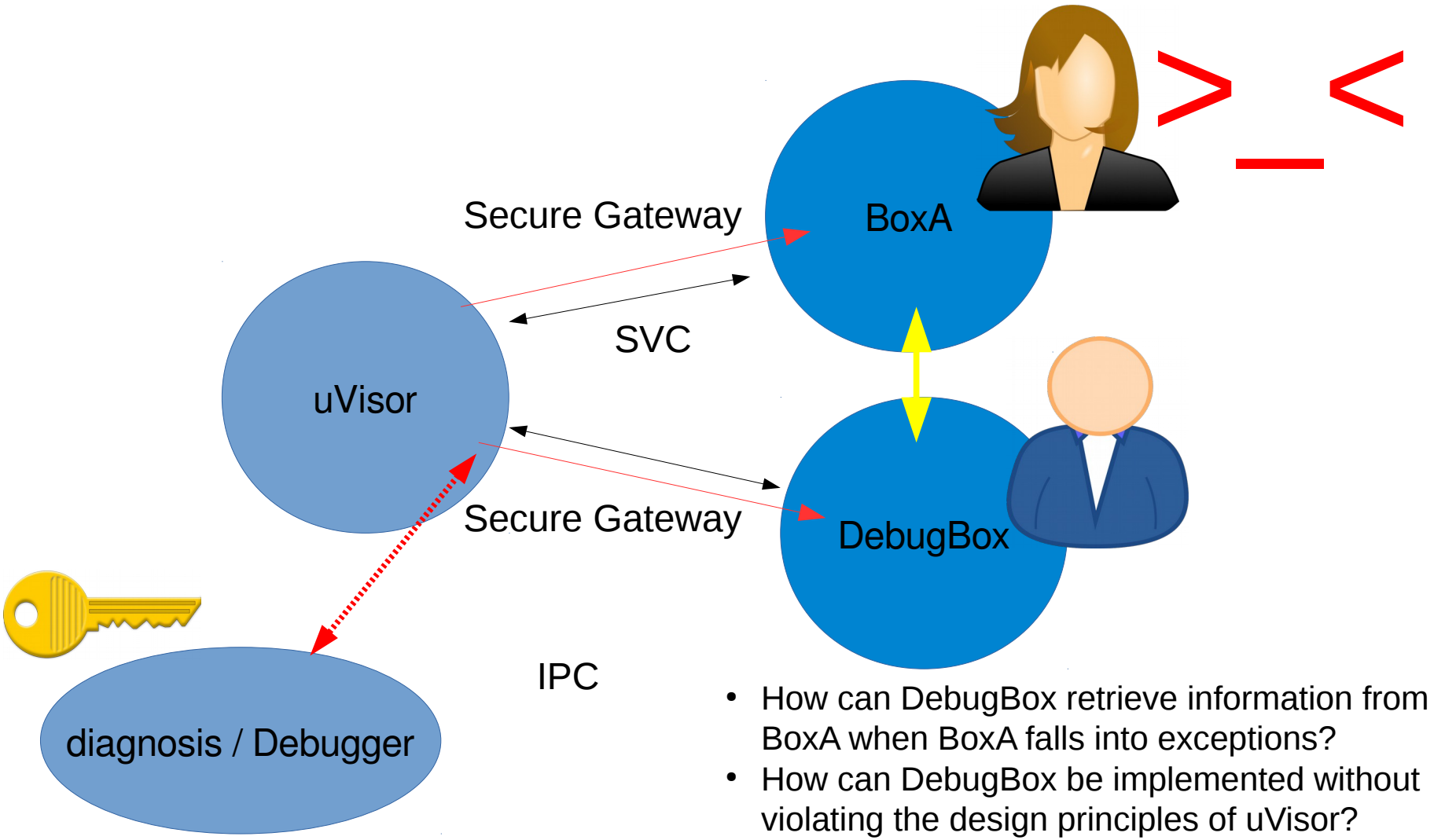
2.發出 svc 系統呼叫 , 再次進入 uVisor svc handler.

3.在 uVisor svc handler 裡重新載入原本被中斷的 box, 設定返回的 stack frame 使得在 svc handler 結束後能返回原本 box 的執行環境

Proposed DebugBox

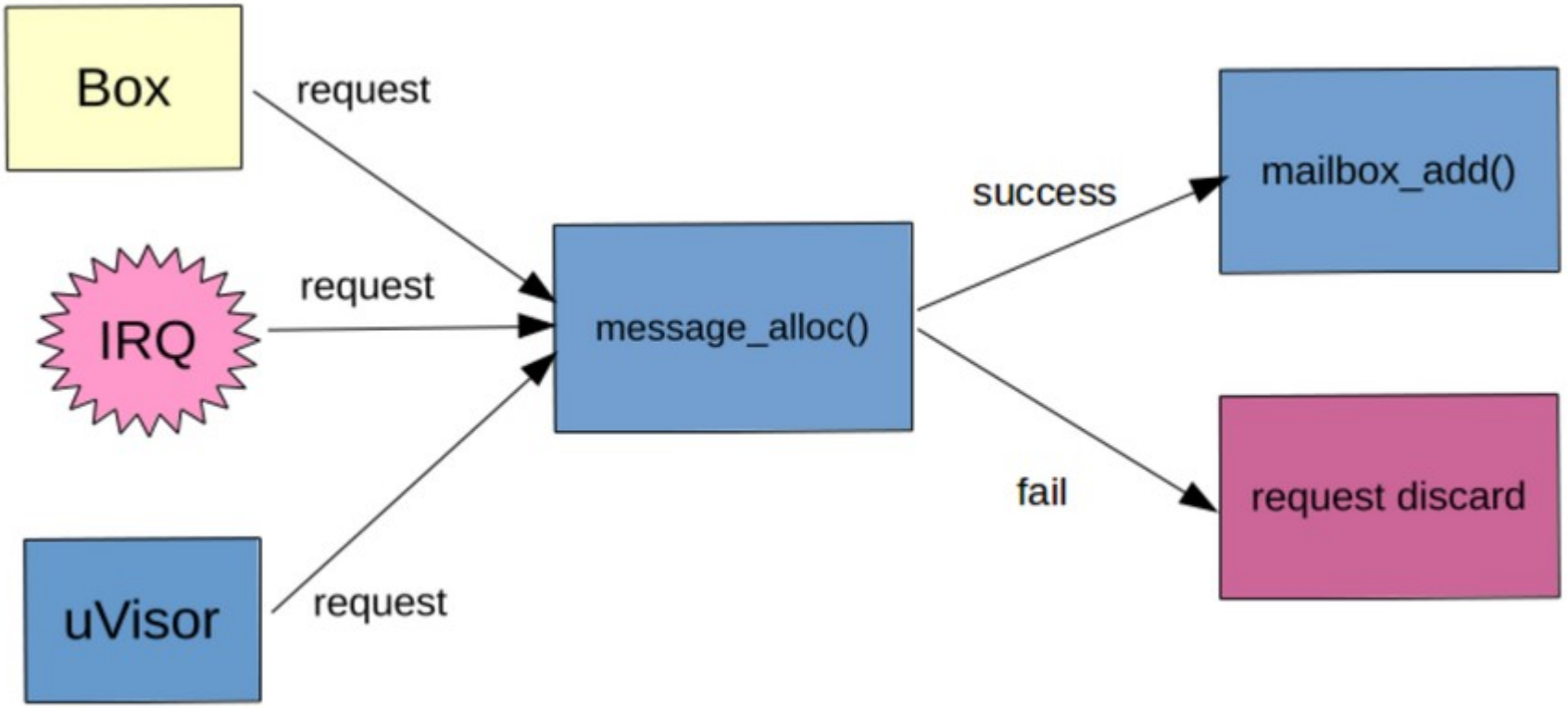


DebugBox



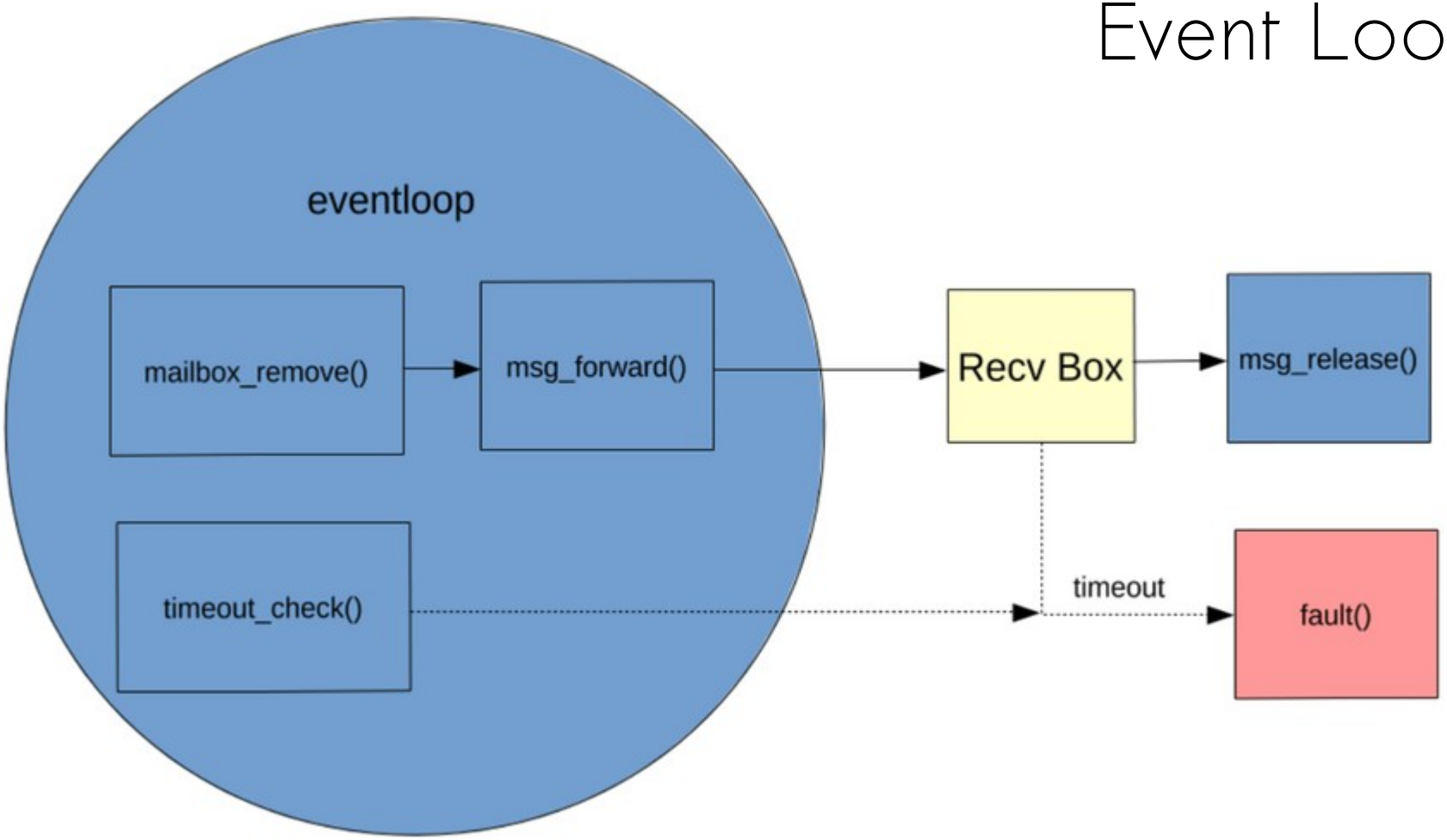
- How can DebugBox retrieve information from BoxA when BoxA falls into exceptions?
- How can DebugBox be implemented without violating the design principles of uVisor?

Mailbox communications



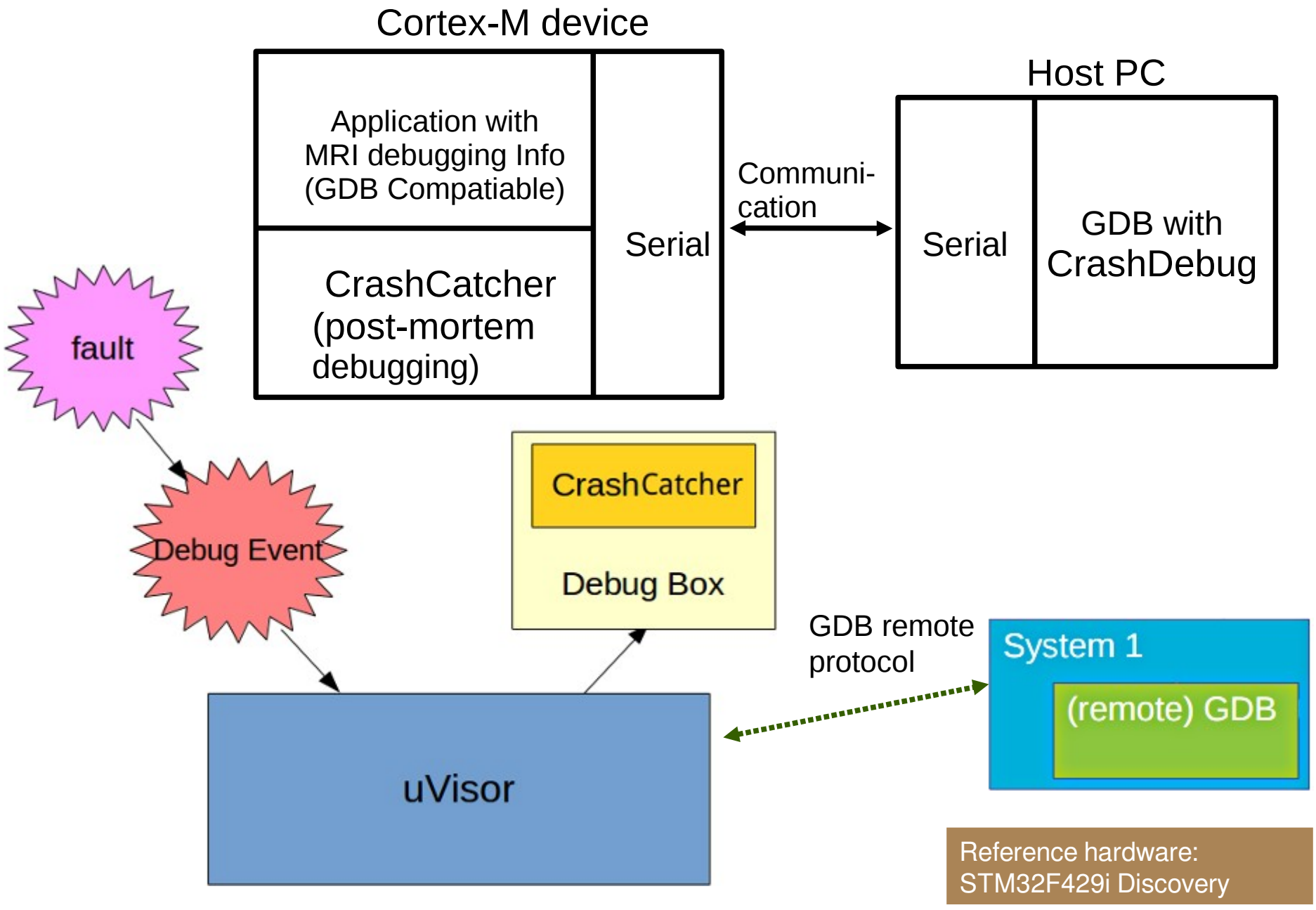
- The communication request instead of being transmitted immediately and delivered at an appointed time/at a desirable time, it is buffered in mailbox. The mailbox can be divided into two components: message queue and eventloop.

Event Loop

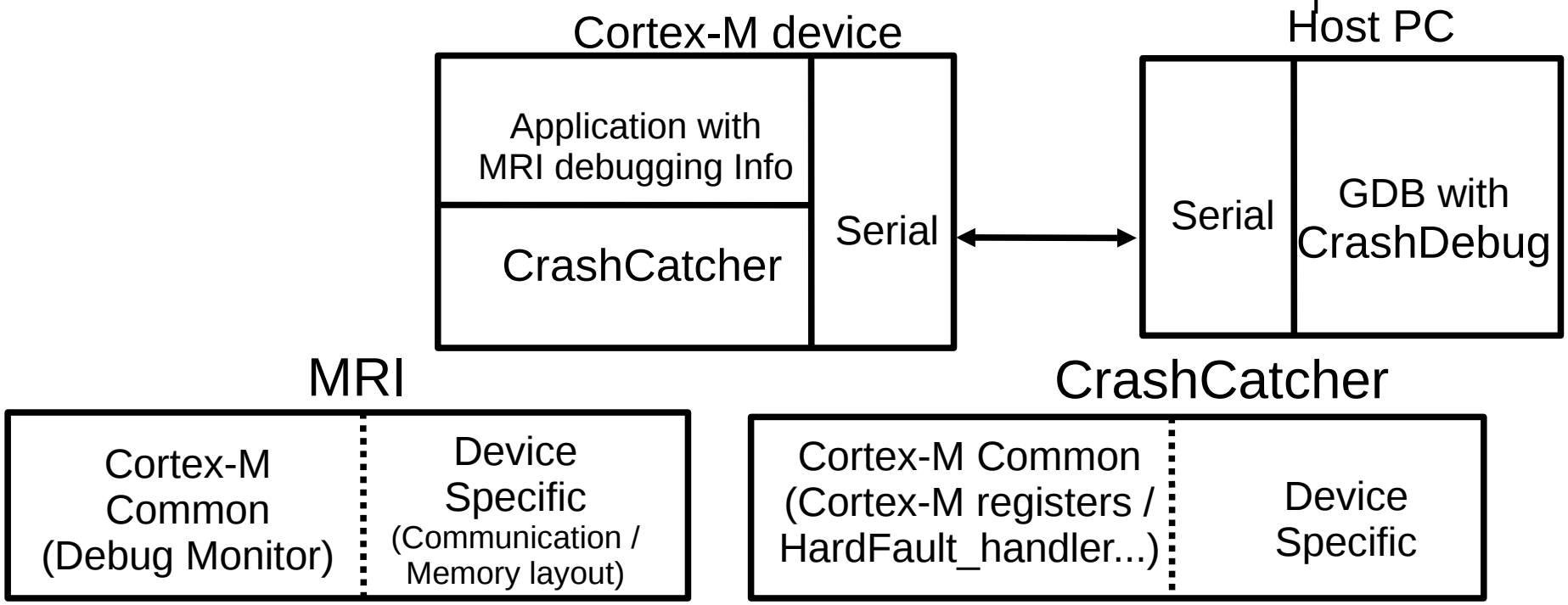


- Once Recv box completes its handling, the resource must return immediately and the initial box needs restoration. Furthermore, to avoid resource holding permanently, there is also a mechanism to recycle the resource automatically.

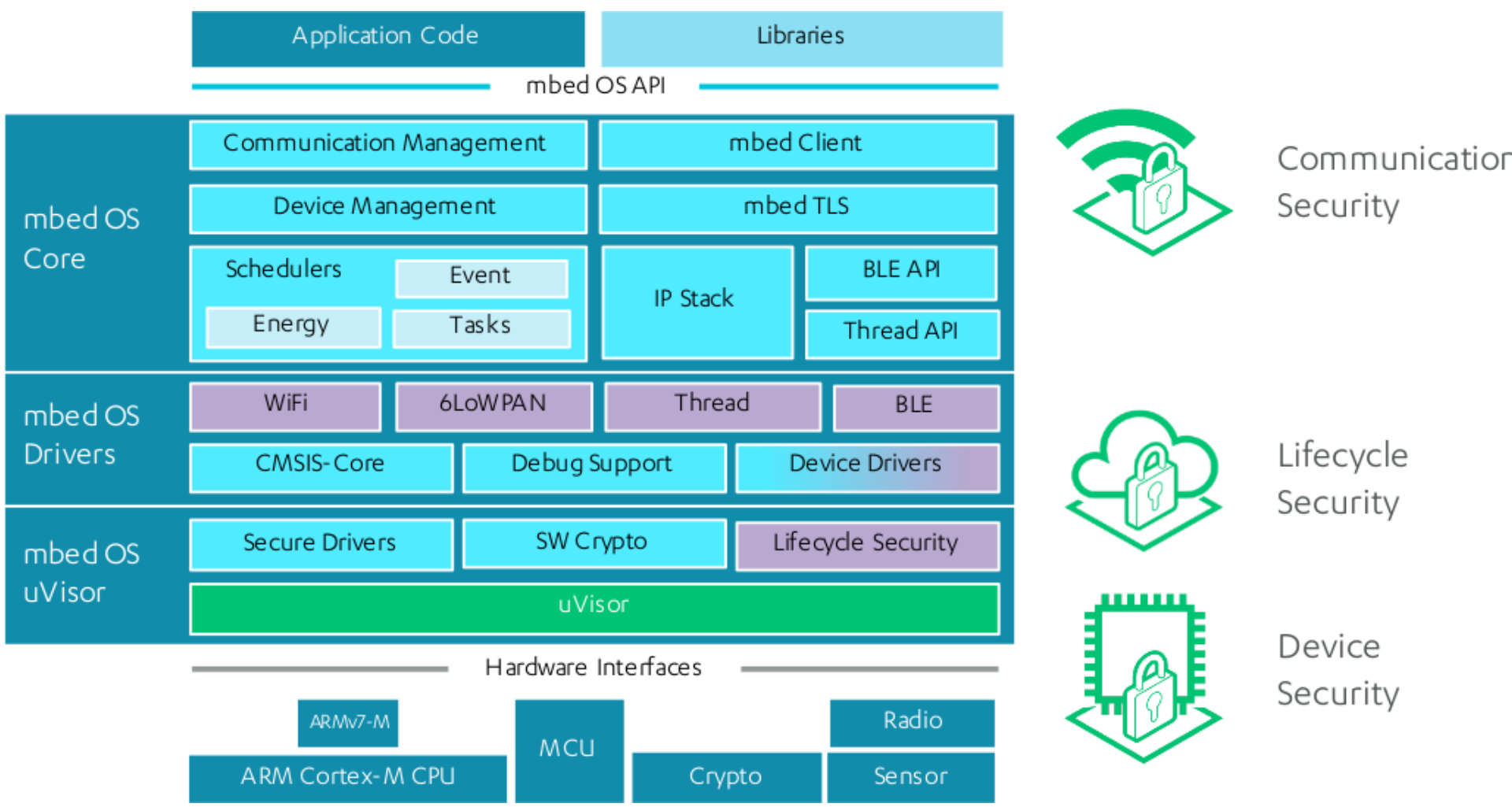
Ad-Hoc Debugging



MRI: Monitor for Remote Inspection



- MRI is a debug monitor which allows the GDB to debug applications running on Cortex-M devices using a full featured source level debugger with no extra hardware other than a serial connection.
- uVisor integration reduces cycles for exposed boxes
  - Still a secure product!
- Simple recovery from programming bugs in exposed code using secure boxes



- ARM Cortex-M processor enables highly deterministic real-time applications to develop high-performance low-cost platforms, and uvisor utilizes Cortex-M advantages to build the efficient and secure trusted computing base (TCB)

Reference

- Resilient IoT Security: The end of flat security models, Milosch Meriac, ARM TechCon 2015
- smart solutions for the internet of things, Genesi USA, Inc.
- Introduction to mbed-OS uvisor, Viller Hsiao
- ARMlock: Hardware-based Fault Isolation for ARM, North Carolina State University / Xi'an Jiaotong University / Florida State University
- Mactans: Injecting Malware into iOS Devices via Malicious Chargers