PULSE

System-level threats:

Dangerous assumptions in modern Product Security

Cristofaro Mune
(c.mune@pulse-sec.com)
@pulsoid

Me

- **Cristofaro Mune** *(@pulsoid)*
  - *Product Security Consultant/Researcher*
  - *Keywords:* TEE, IoT, Devices, HW, Fault Injection, Exploitation,...

- *(Public)* Research:
  - *[2009] - **“Hijacking Mobile Data connections”***
  - *[2010] - **AP Exploitation***
  - *[2015] - **Breaking WB cryptography***
  - *[2017] :*
    - ***TEEs secure initialization***
    - ***IoT exploitation***
    - ***Linux Privilege Escalation with Fault Injection***

Devices and markets

Networking

Physical Security

Mobile devices

Smart homes

Mobile Payments

Automotive

Who “owns” a device?

John McAfee 

@officialmcafee

Follow

Replying to @joshikml

Its my wallet, I am CEO of BitFi

2:15 PM - 20 Jul 2018

iPhone X components and suppliers (some)

Accelerometer

Bosch & Ivensense

Baseband Processor

Qualcomm

Batteries

Samsung & Shenzhen Desay Battery Technology

Chips

Cirus Logic, Samsung, TSMC, MicroSemi, Broadcom & NXP

DRAM

TSMC & SK Hynics

eCompass

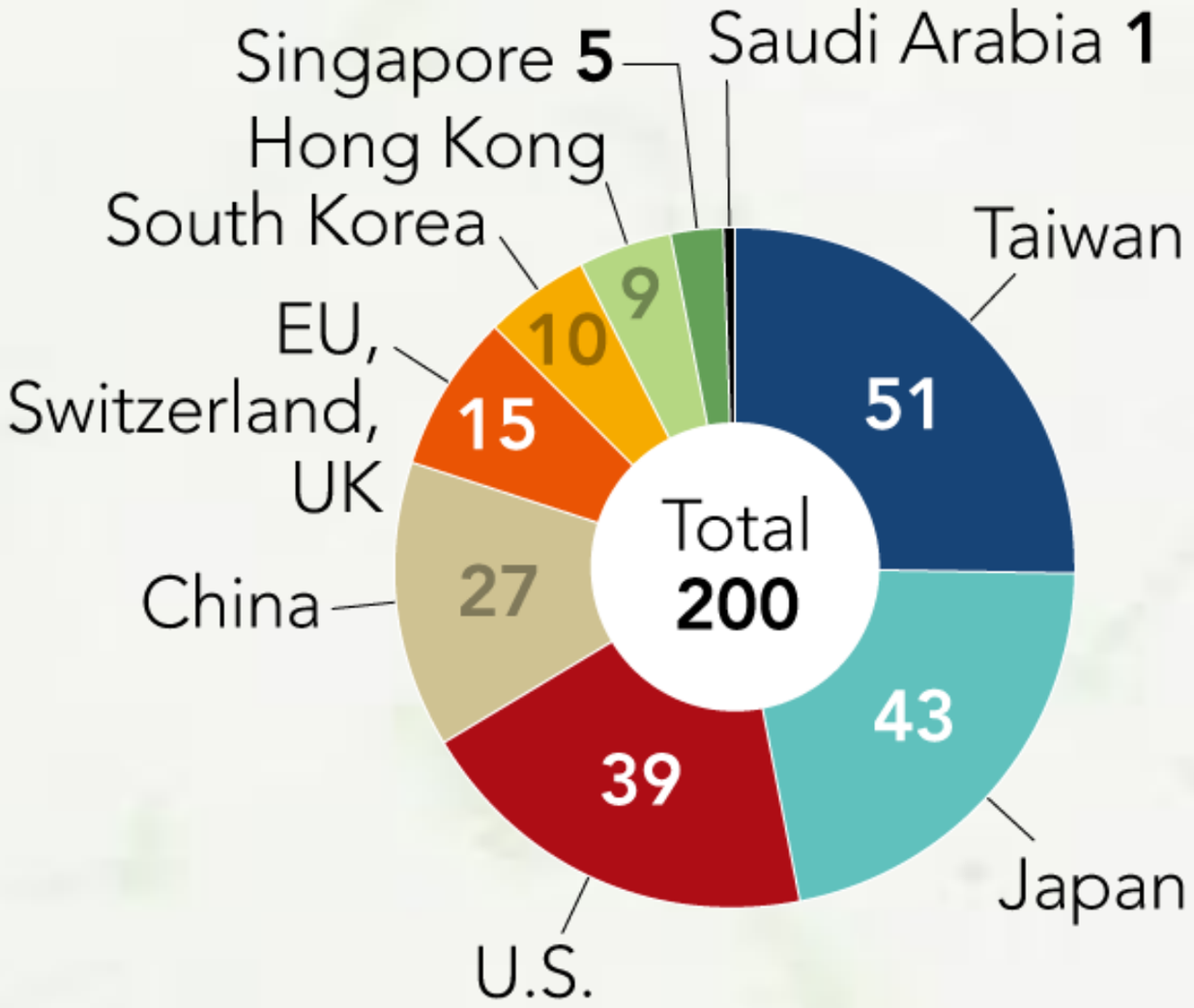
Alps Electric

*from capitalistlad.wordpress.org

Example: Apple suppliers 2018

Where Apple suppliers are headquartered

(number of companies)



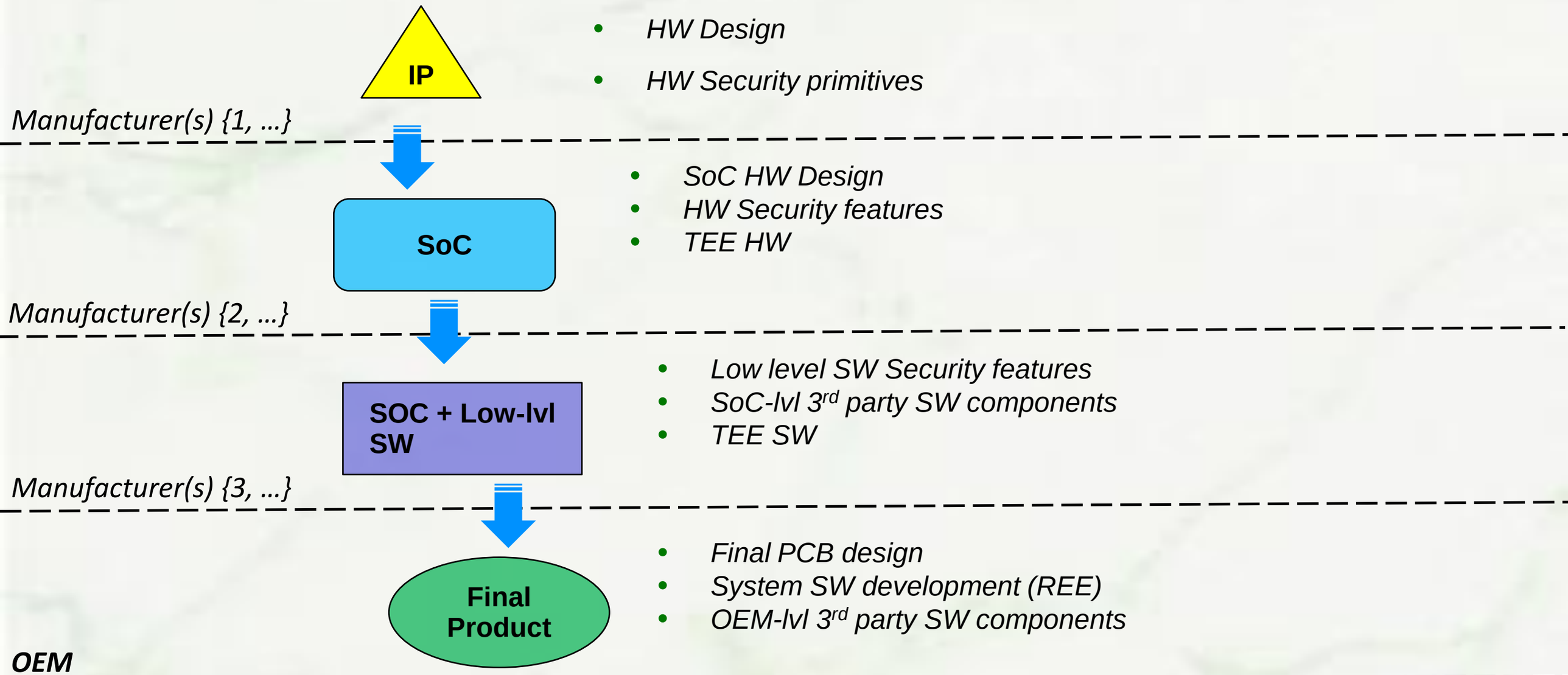
Source: Apple's list of suppliers for 2018

Who owns a device?

“Nobody.

Really, nobody FULLY owns a device.”

Device == ecosystem effort → PRODUCT



- *Each component in the chain has its own:*
  - Threat models, use cases and ***assumptions***

Assumptions?

*"Assumptions at boundaries may **cause system-level vulnerabilities**"*

Assumptions!

“A few recurring classes (out of my experience)…”

Completeness

Assumption of Completeness

*"Component-level information is **sufficient** for characterizing security impacts on the final system"**

**** Applies both to design and security assessments***

Example: SW security assessment

- A *strcmp()* classic implementation
- Taken from open-source project uClibc-ng
- Aimed at embedded devices
  - Latest available version: v1.0.31
- Full source code available

Any vulnerability?

```
31  int strcmp (const char *p1, const char *p2)
32  {
33    register const unsigned char *s1 = (const unsigned char *) p1;
34    register const unsigned char *s2 = (const unsigned char *) p2;
35    unsigned reg_char c1, c2;
36
37    do
38      {
39        c1 = (unsigned char) *s1++;
40        c2 = (unsigned char) *s2++;
41        if (c1 == '\0')
42          return c1 - c2;
43
44      }
45    while (c1 == c2);
46
47    return c1 - c2;
48  }
```

https://elixir.bootlin.com/uclibc-ng/v1.0.31/source/libc/string/generic/strcmp.c

Vulnerable?

Threat model (TM): ***only SW attacks***

Not vulnerable

But...

```
31  int strcmp (const char *p1, const char *p2)
32  {
33    register const unsigned char *s1 = (const unsigned char *) p1;
34    register const unsigned char *s2 = (const unsigned char *) p2;
35    unsigned reg_char c1, c2;
36
37    do
38      {
39        c1 = (unsigned char) *s1++;
40        c2 = (unsigned char) *s2++;
41        if (c1 == '\0')
42          return c1 - c2;
43
44      }
45    while (c1 == c2);
46
47    return c1 - c2;
48  }
```

Early termination

Timing attack

https://elixir.bootlin.com/uclibc-ng/v1.0.31/source/libc/string/generic/strcmp.c

What about now?

- If attacker is able:
  - to *access a measurable channel*
  - to *sample with sufficient precision*
- If compared quantity is a security asset:
  - e.g. A password, a MAC,...

Depends on the system.

Vulnerable

OK…which Threat Model applies?

Threat Models

SW Component

strcmp()

SW attacks only

Not Vulnerable

SW + Timing

Vulnerable

Secure component → Vulnerable system

TM(Component) = SW only

strcmp()

SECURE

*TM(System) = SW + **Timing attacks***

Final Product

SECURE

Vulnerable System

Remarks: Vulnerability

- Component level information insufficient
- Vulnerability cannot be assessed at component level
  - ***Depends on final system*** *Threat Model*
- Code review does not solve the problem
  - Unless a Threat Model is specified

- Only “next stage” integrators are able to assess vulnerability:
  - E.g: OEMs at the final product integration

Exploitability?

- String comparison:
  - *"**Impractical** in the vast majority of cases" 2015 – Morgan & Morgan*
  - Remote servers with *fast CPUs*

- But... ***IoT systems are much slower!***

that differ in the first character vs. strings that differ only at the 10th character. This indicates that timing attacks on regular string comparison have to be assumed feasible for any embedded system.*

** 2014 – Mayer, Sandin – "Time Trial"*

Demo

- Target: Arduino UNO
  - Clock speed: 16 Mhz
  - Media: Ethernet 100Mbit
- Numeric PIN: 8 digits

```
Candidate password: 31337890
Verifying: 31337890
Success!
Verification successful!
Password found!!!
[+] Attack Completed
Total requests: 68002
Bruteforce complexity: 100000000
Ratio: 0.07%
```

(Live demo: Come tomorrow at HITB Armory!)

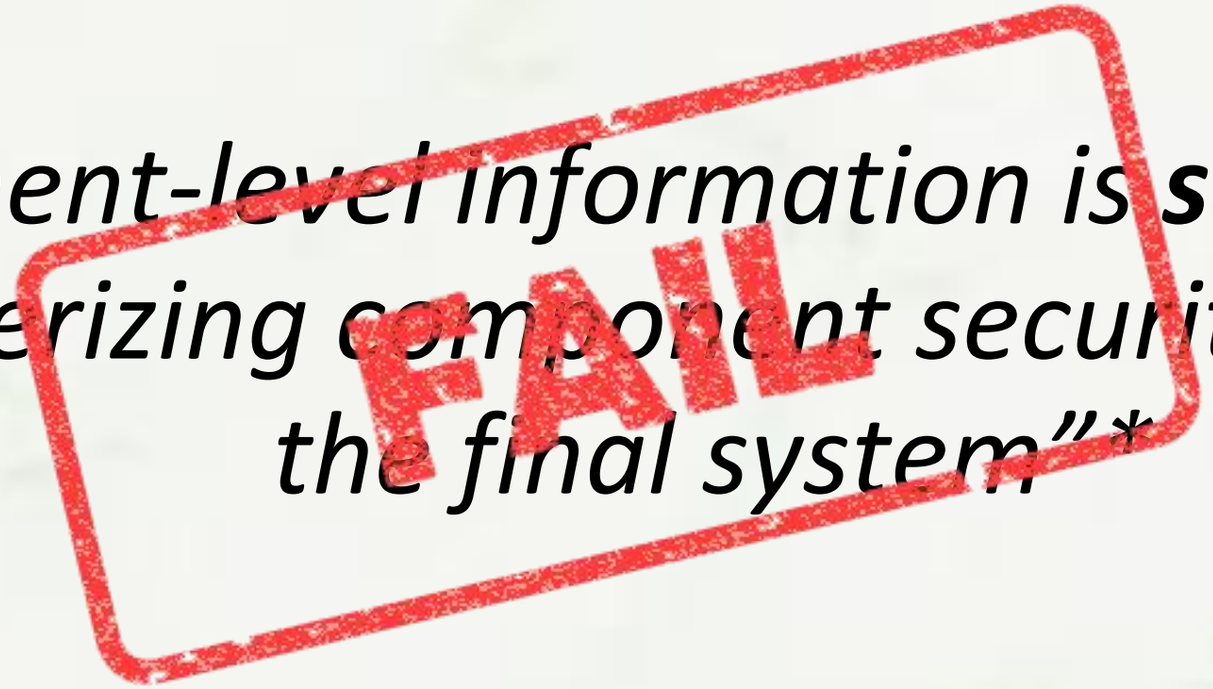
Remarks: Exploitability

- Exploitability cannot be assessed at component level
  - Even with full source code
- Impact ***depends on final system***:
  - E.g. Clock speed, measurable channels,...

Full impacts can be established only at final product integration

Assumption of Completeness

*"Component-level information is **sufficient** for characterizing component security impact on the final system"**



**** Applies both to design and security assessments***

Always ask for...

- *Threat Model*
- *Existing development security processes*:
  - SDLC, Design security reviews, Source code audits, Product Penetration Testing
- *Existing in-field security processes*:
  - Security fixes, Security maintenance (e.g. Firmware update for 5 yrs),...
- *Security evaluation reports*

“When buying components, you are also buying risks”

A good example: ARM TrustZone

Note

TrustZone technology is designed to provide a hardware-enforced logical separation between security components and the rest of the SoC infrastructure.

Lab attacks are outside of the scope of the protection provided TrustZone technology, although a SoC using TrustZone can be used in conjunction with an ARM SecurCore© smartcard if protection against physical attacks is needed for some assets.

Threat Model specified in documentation

Correctness

Assumption of Correctness

*"The system will always behave as intended. Correctly executing as specified."**

**** Applies both to design and security assessments***

Fault attacks

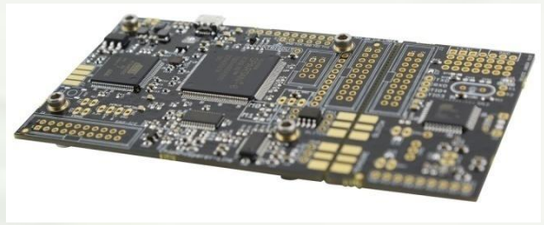
*“Introducing faults in a target **to alter its intended behavior**.”*

Clock *Voltage* *EM* *Laser*

- A controlled environmental change leads to altered behavior in a target
- Leverage a **vulnerability** in a ***hardware subsystem***

Assumption: Expensive

Chipwhisperer Lite

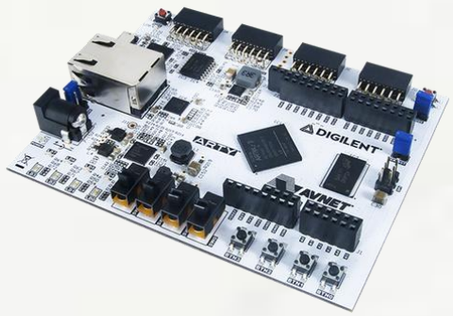


~$250

Microcontroller

< $30

FPGA



~$99

- Also *“Cheapscate: Attacking IoT with less than $60” - Raphael Boix Carpi*

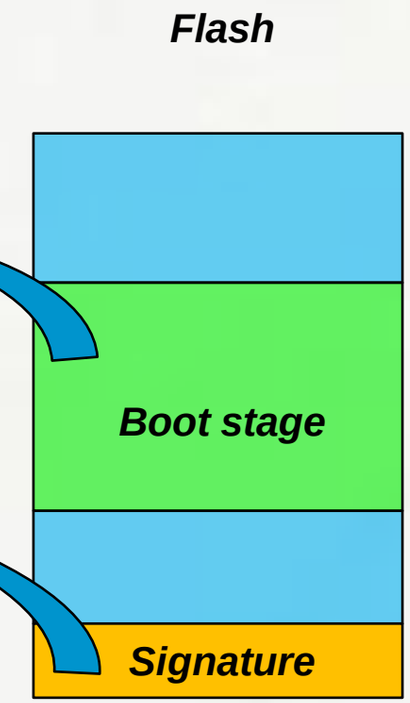
VCC glitching cost ($): < 300

Other assumptions

- **Physical access is required:**
  - *Wrong*. SW-initiated FI attacks can be performed remotely:
    - Rowhammer and CLKSCREW
- **No security decision point → Nothing to attack:**
  - *Wrong*. New fault models allow for direct code exection.
- **We have countermeasures in SW:**
  - *Wrong*. New fault models can completely bypass SW FI countermeasures
- **Our secure boot is encrypted. You need a key anyway:**
  - *Wrong*.
  - Wait a minute. ☺

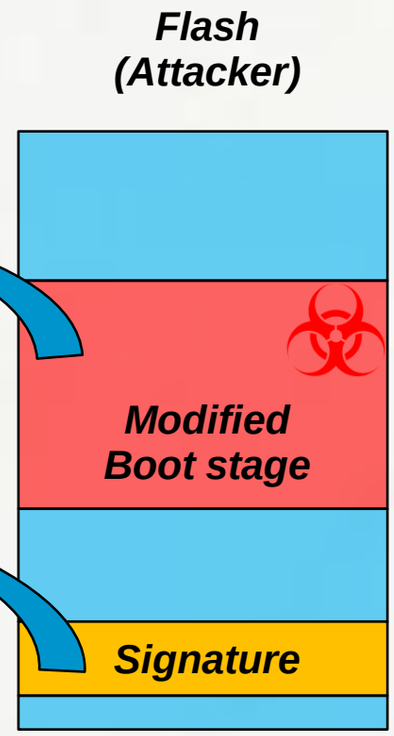
Secure Boot

```
 4  int load_exec_next_boot_stage(){
 5
 6      uint32_t stage_addr=0xd0000000;
 7      uint32_t sig_addr=0xc0000000;
 8
 9      // Copy stage from media to memory
10      load_next_stage(stage_addr);
11
12      // Copy signature to memory
13      load_signature(sig_addr);
14
15      // Verify signature
16      if(!verify_signature(stage_addr,sig_addr)) {
17
18          //Wrong signature. Hang.
19          while(1);
20      } else {
21
22          //Execute next stage
23          exec_stage(stage_addr);
24      }
```



Textbook attack

```
 4  int load_exec_next_boot_stage(){
 5
 6      uint32_t stage_addr=0xd0000000;
 7      uint32_t sig_addr=0xc0000000;
 8
 9      // Copy stage from media to memory
10      load_next_stage(stage_addr);
11
12      // Copy signature to memory
13      load_signature(sig_addr);
14
15      // Verify signature
16      if(!verify_signature(stage_addr,sig_addr)) {
17
18          //Wrong signature. Hang.
19          while(1);
20      } else {
21
22          //Execute next stage
23          exec_stage(stage_addr);
24      }
```



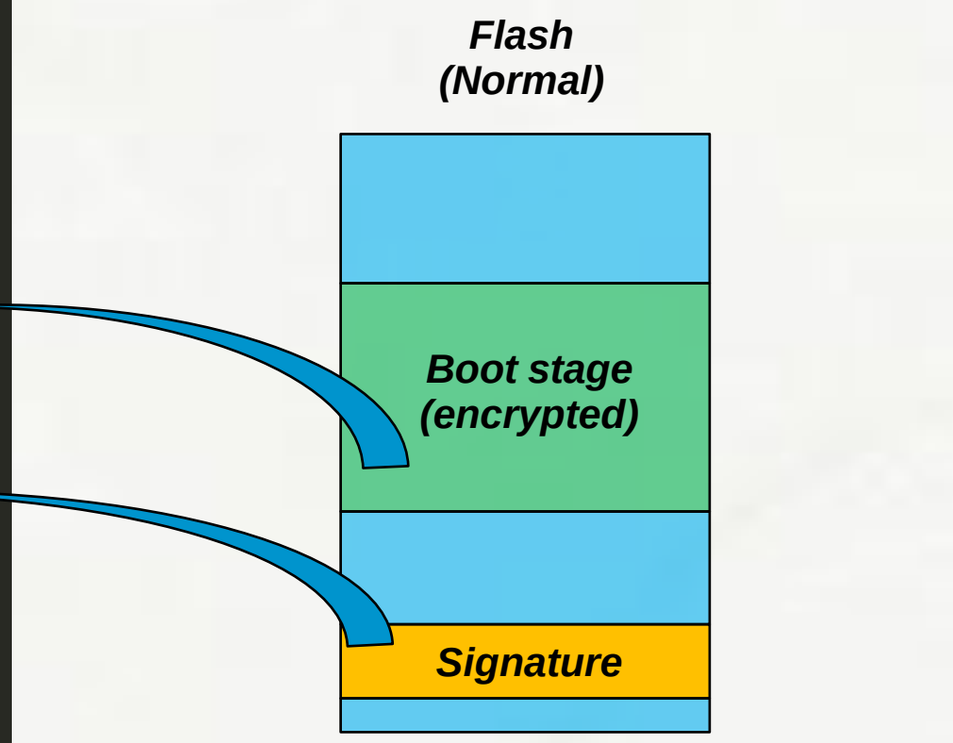
Glitch here → Signature bypass

Arbitrary code execution

“Instruction skipping”

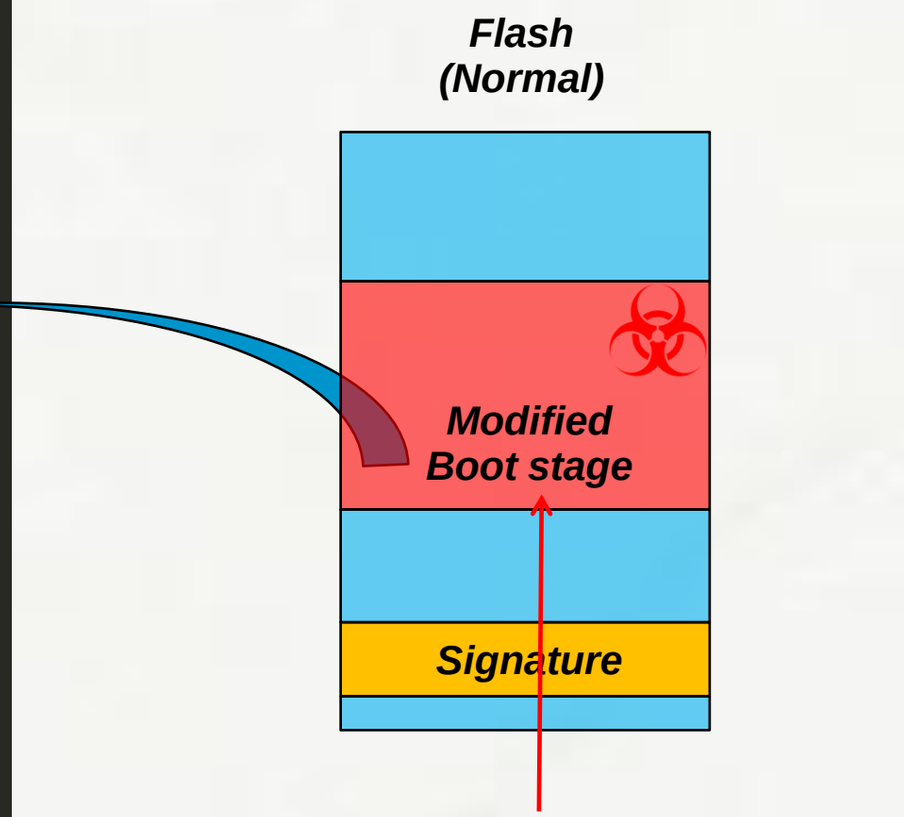
Encrypted Secure Boot

```
5  int load_exec_next_boot_stage(){
6
7      AES ctx;
8      uint32_t stage_addr=0xd0000000;
9      uint32_t sig_addr=0xc0000000;
10
11     init_AES_engine(&ctx, key_id);
12
13     // Copy stage from media to memory
14     load_encrypted_next_stage(stage_addr);
15
16     // Copy signature to memory
17     load_signature(sig_addr);
18
19     // Stage is encrypted. Decrypt first.
20     decrypt_stage(&ctx, stage_addr);
21
22     // Verify signature over stage plaintext
23     if(!verify_signature(stage_addr,sig_addr))  {
24
25         //Wrong signature. Hang.
26         while(1);
27     } else {
28
29         //Execute next stage
30         exec_stage(stage_addr);
31     }
```



FI textbook attack insufficient

```
5  int load_exec_next_boot_stage(){
6
7      AES ctx;
8      uint32_t stage_addr=0xd0000000;
9      uint32_t sig_addr=0xc0000000;
10
11     init_AES_engine(&ctx, key_id);
12
13     // Copy stage from media to memory
14     load_encrypted_next_stage(stage_addr);
15
16     // Copy signature to memory
17     load_signature(sig_addr);
18
19     // Stage is encrypted. Decrypt first.
20     decrypt_stage(&ctx, stage_addr);
21
22     // Verify signature over stage plaintext
23     if(!verify_signature(stage_addr,sig_addr)) {
24
25         //Wrong signature. Hang.
26         while(1);
27     } else {
28
29         //Execute next stage
30         exec_stage(stage_addr);
31     }
```



Unknown encryption key

Signature bypass useless

Creating execution primitives…out of thin air

- ARM32 has an interesting ISA
- Program Counter (PC) is directly accessible

Valid ARM instructions

```
MOV r7,r1          00000001 01110000 10100000 11100001
EOR r0,r1          00000001 00000000 00100000 11100000
LDR r0,[r1]        00000000 00000000 10010001 11100101
LDMIA r0,{r1}      00000010 00000000 10010000 11101000
```

Corrupted ARM instructions may directly set PC

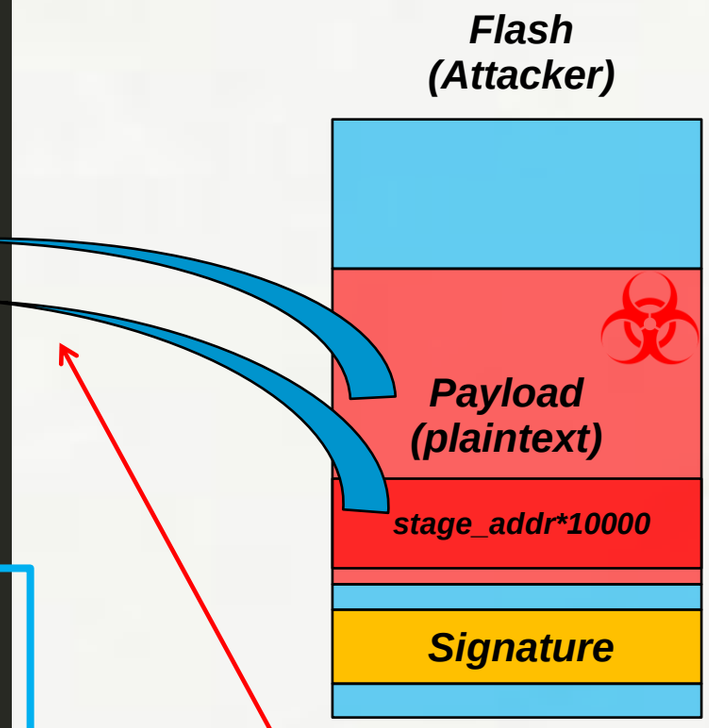
```
MOV pc,r1          00000001 11110000 10100000 11100001
EOR pc,r1          00000001 11110000 00101111 11100000
LDR pc,[r1]        00000000 11110000 10010001 11100101
LDMIA r0,{r1, pc}  00000010 10000000 10010000 11101000
```

Attack variations (SP-control) also affect other architectures

**also see [FDTC 2016]: Timmers, Spruyt, Witteman*

ROM code: secure boot + encryption

```
5  int load_exec_next_boot_stage(){
6
7      AES ctx;
8      uint32_t stage_addr=0xd0000000;
9      uint32_t sig_addr=0xc0000000;
10
11     init_AES_engine(&ctx, key_id);
12
13     // Copy stage from media to memory
14     load_encrypted_next_stage(stage_addr);
15
16     // Copy signature to memory
17     load_signature(sig_addr);
18
19     // Stage is encrypted. Decrypt first.
20     decrypt_stage(&ctx, stage_addr);
21
22     // Verify signature over stage plaintext
23     if(!verify_signature(stage_addr,sig_addr))  {
24
25         //Wrong signature. Hang.
26         while(1);
27     } else {
28
29         //Execute next stage
30         exec_stage(stage_addr);
31     }
```



Glitch while loading pointers → PC set to pointers → Code exec at stage_addr*

Never executed

**also see [FDTC 2016]: Timmers, Spruyt, Witteman*

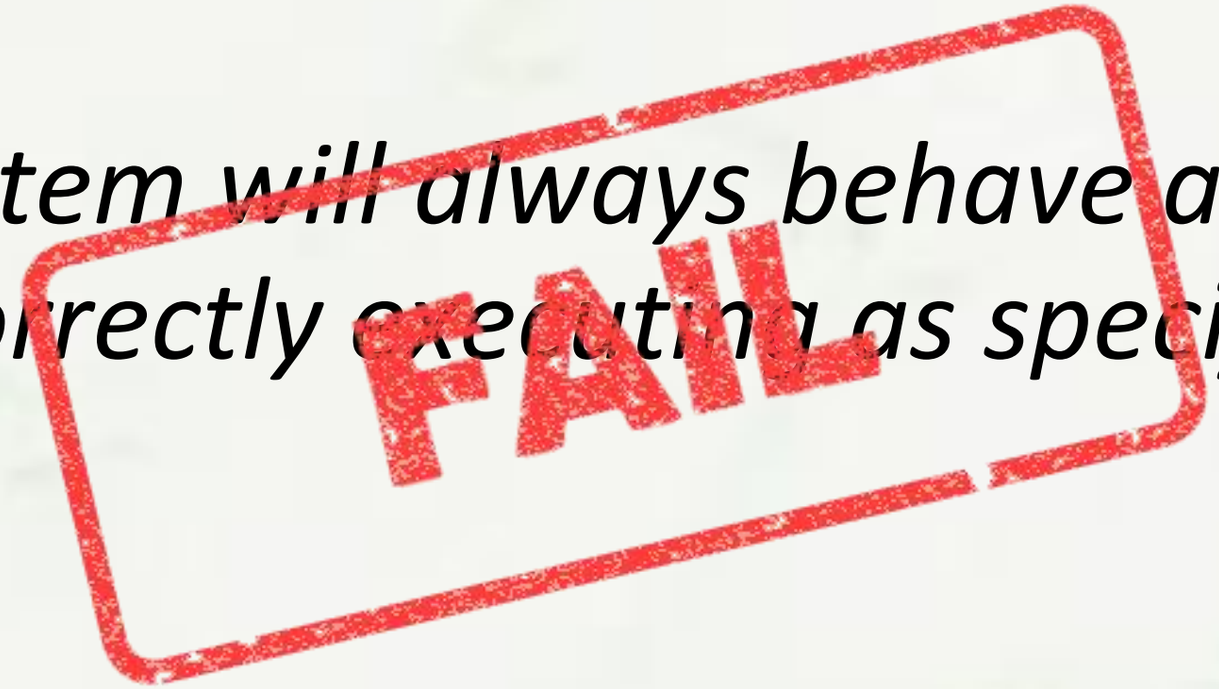
Remarks

- FI vulnerabilities located at physical level
- ***Cannot be identified via HW or SW code review***
- Only **testing the real implementation** can provide indications of vulnerability

- Testing better performed right after silicon integration:
  - e.g. SoC Manufacturer
- Vulnerable HW may affect entire classes of devices

Assumption of Correctness

*“The system will always behave as intended. Correctly executing as specified.”**



**** Applies both to design and security assessments***

FI in your threat model?

- Ask your SoC manufacturer for ***Security evaluation reports:***
- Do **NOT** rely only on HW/SW audits only
  - *Only testing can uncover FI vulnerabilities*

“Either you have countermeasures, or…

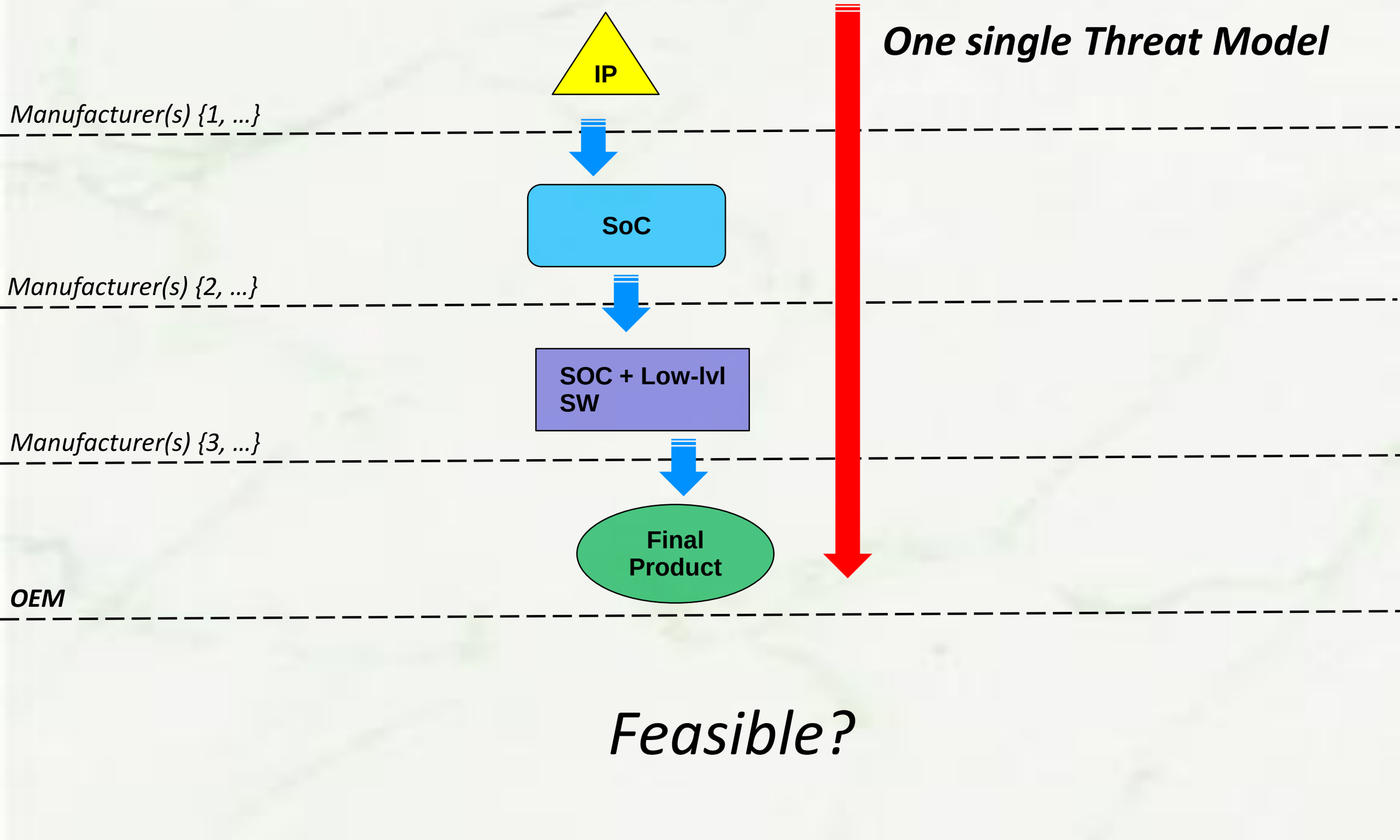
…you are painfully, ***desperately vulnerable”***

Consistency

Assumption of Consistency

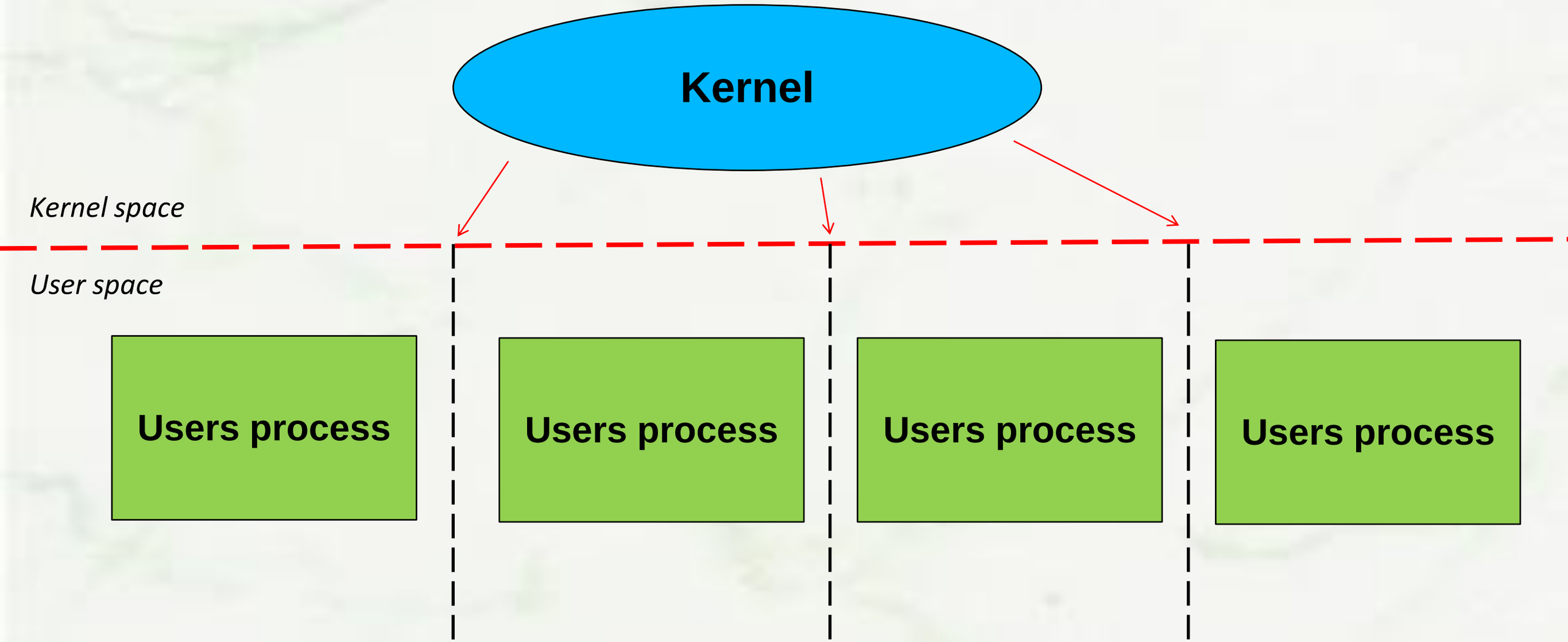
“The entire system has only one threat model and protects the same assets.”

Assumption

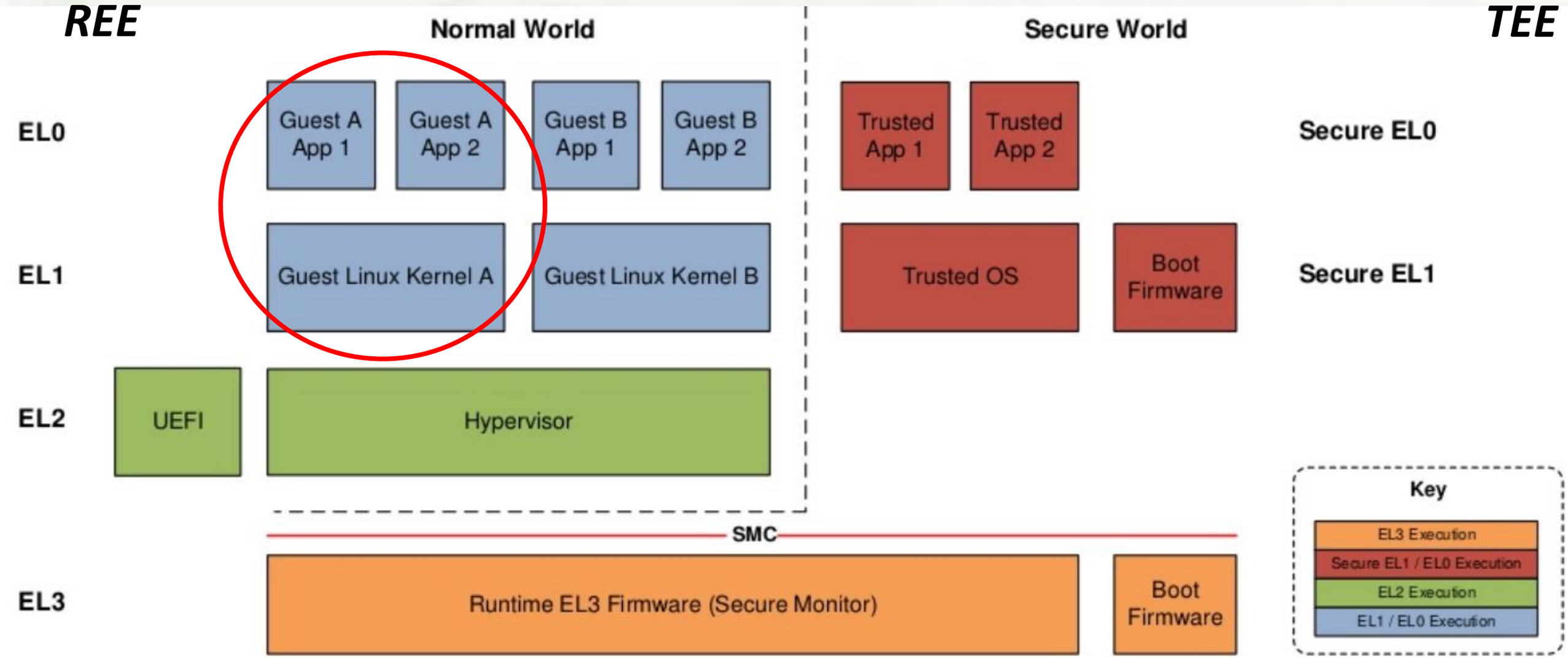


Feasible?

A typical Security Model...

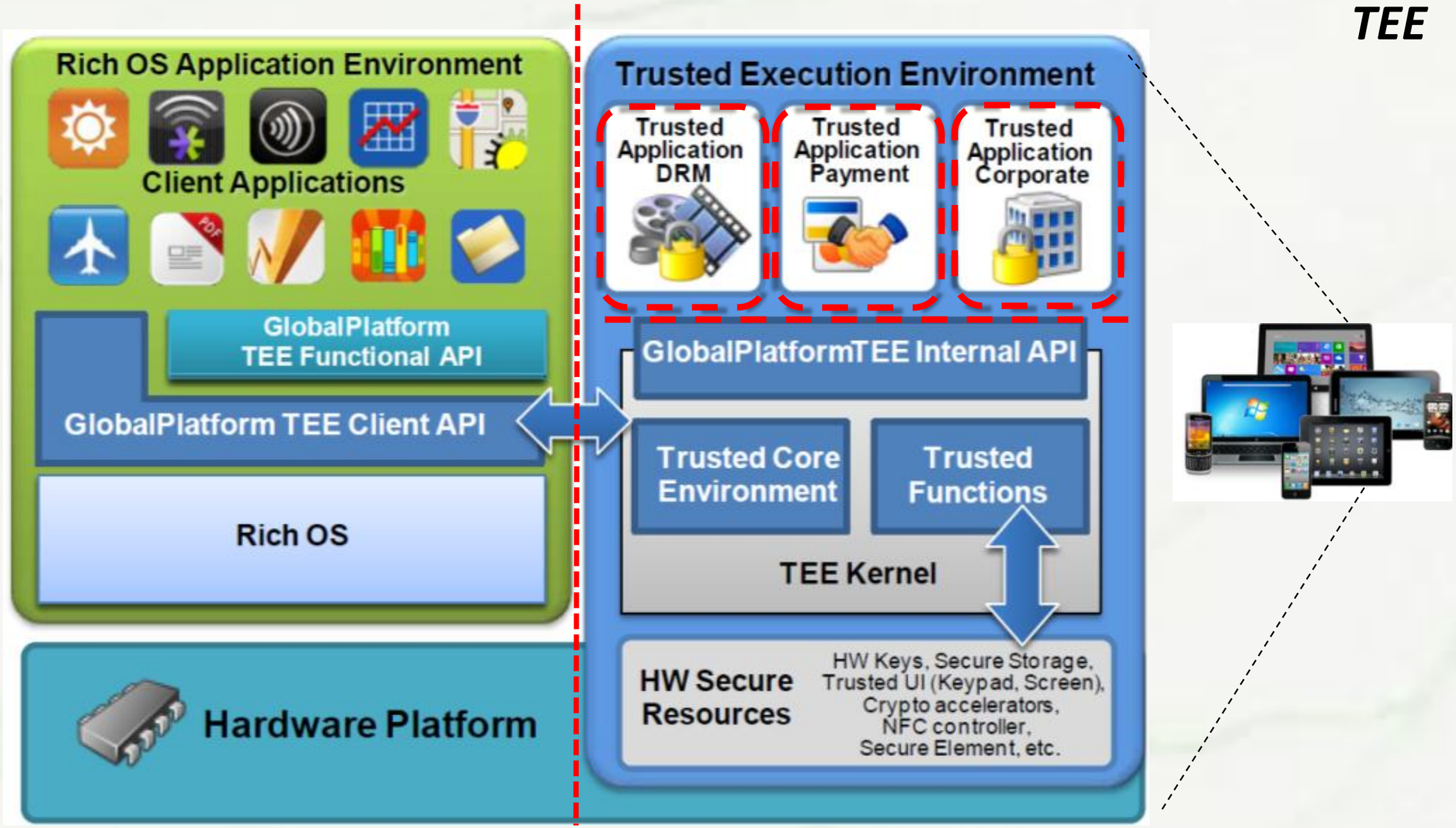


...with a TEE



ARM TrustZone SoC (TEE HW)

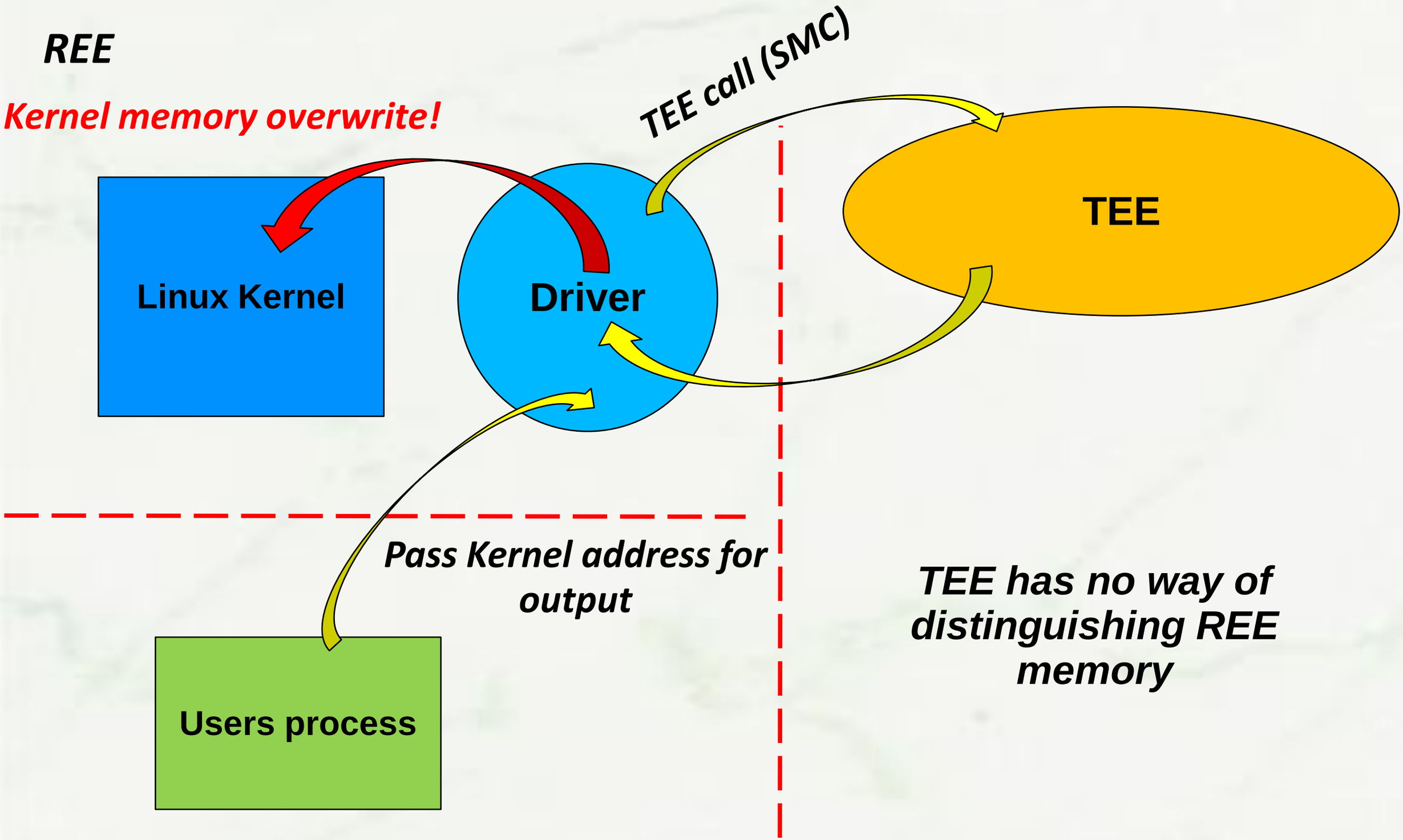
TEE security model



No intention to protect REE…

REE

Kernel memory overwrite!

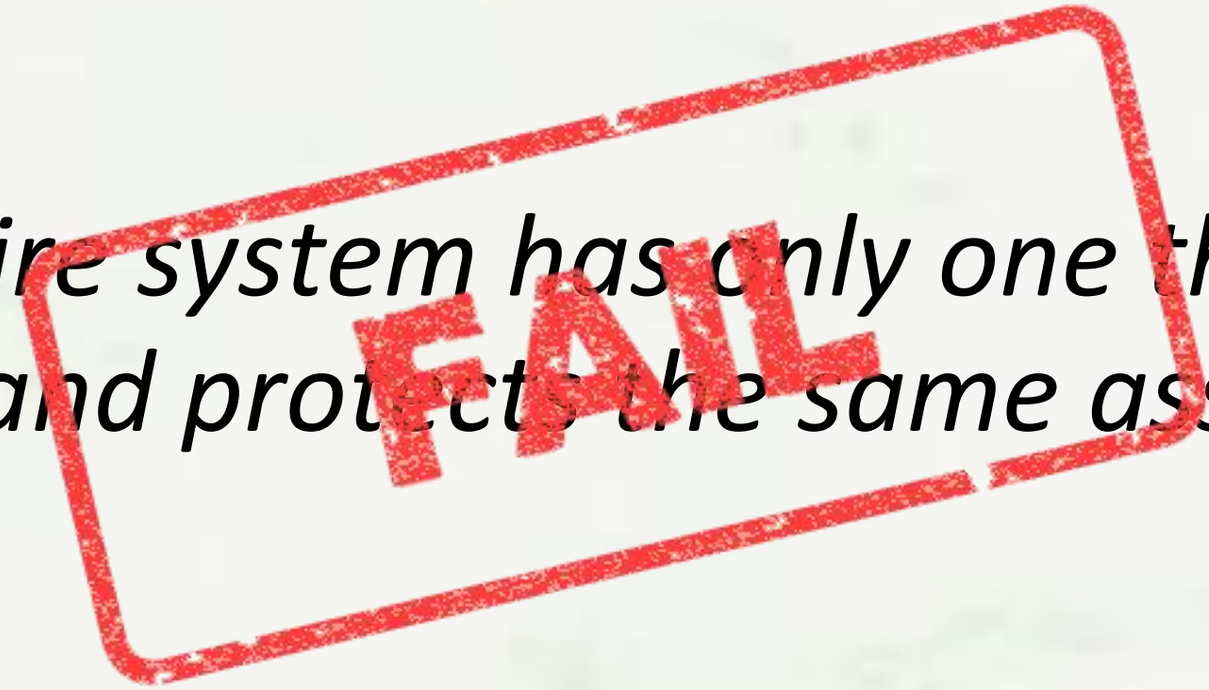


TEE has no way of distinguishing REE memory

Remarks

- Different security models may be present at the same time
- May not be aware of each others
- May be leveraged AGAINST each others

Assumption of Consistency



“The entire system has only one threat model and protects the same assets.”

Recommendations

- Understand components' Security Model
  - Does it fit with your Security Model?
- Evaluate within system Threat Model

"There may be no consistency, across components and subsystems."

…expect none…

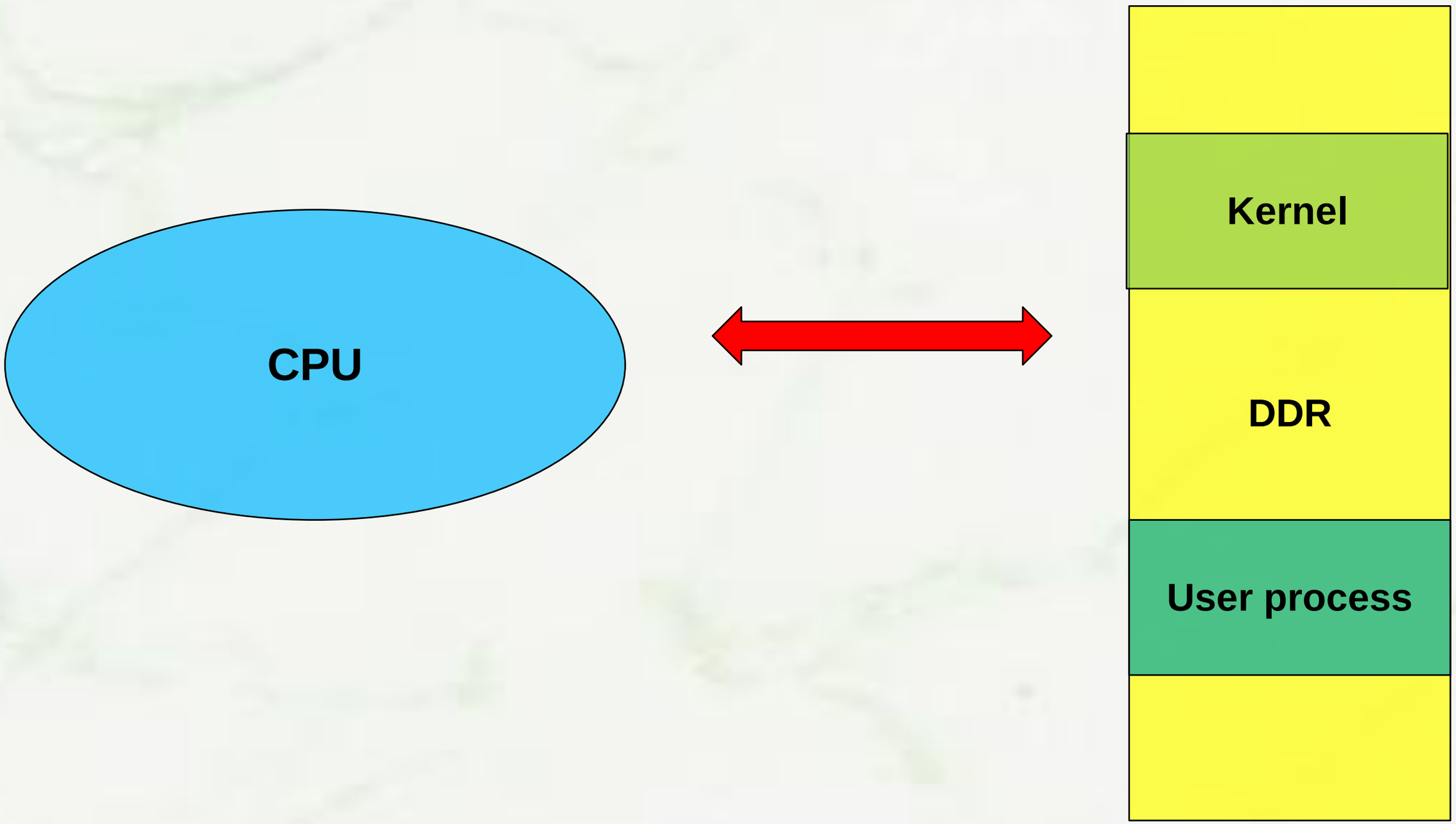
Isolation

Assumption of Isolation

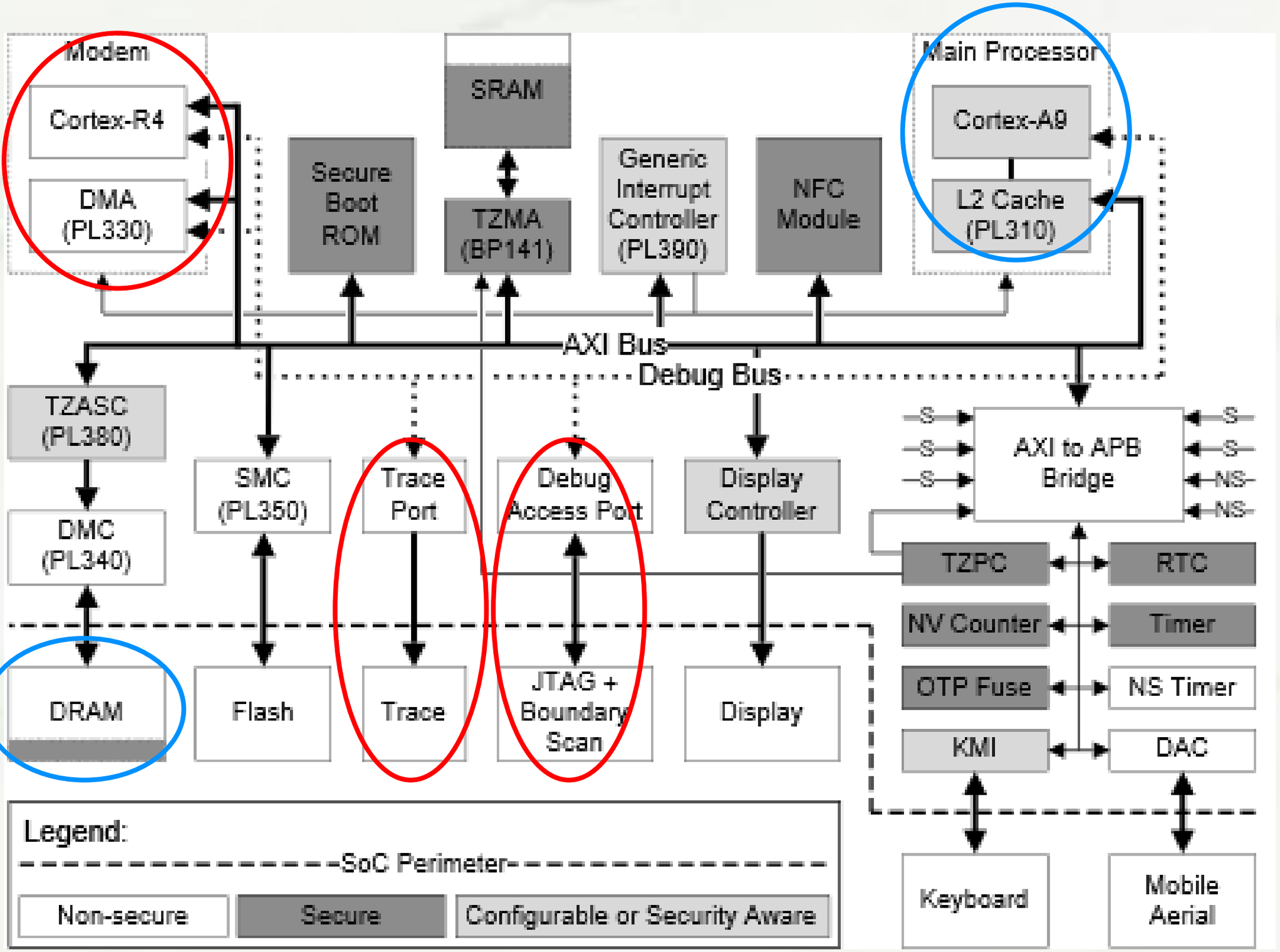
“There is only sub-system.

Mine”

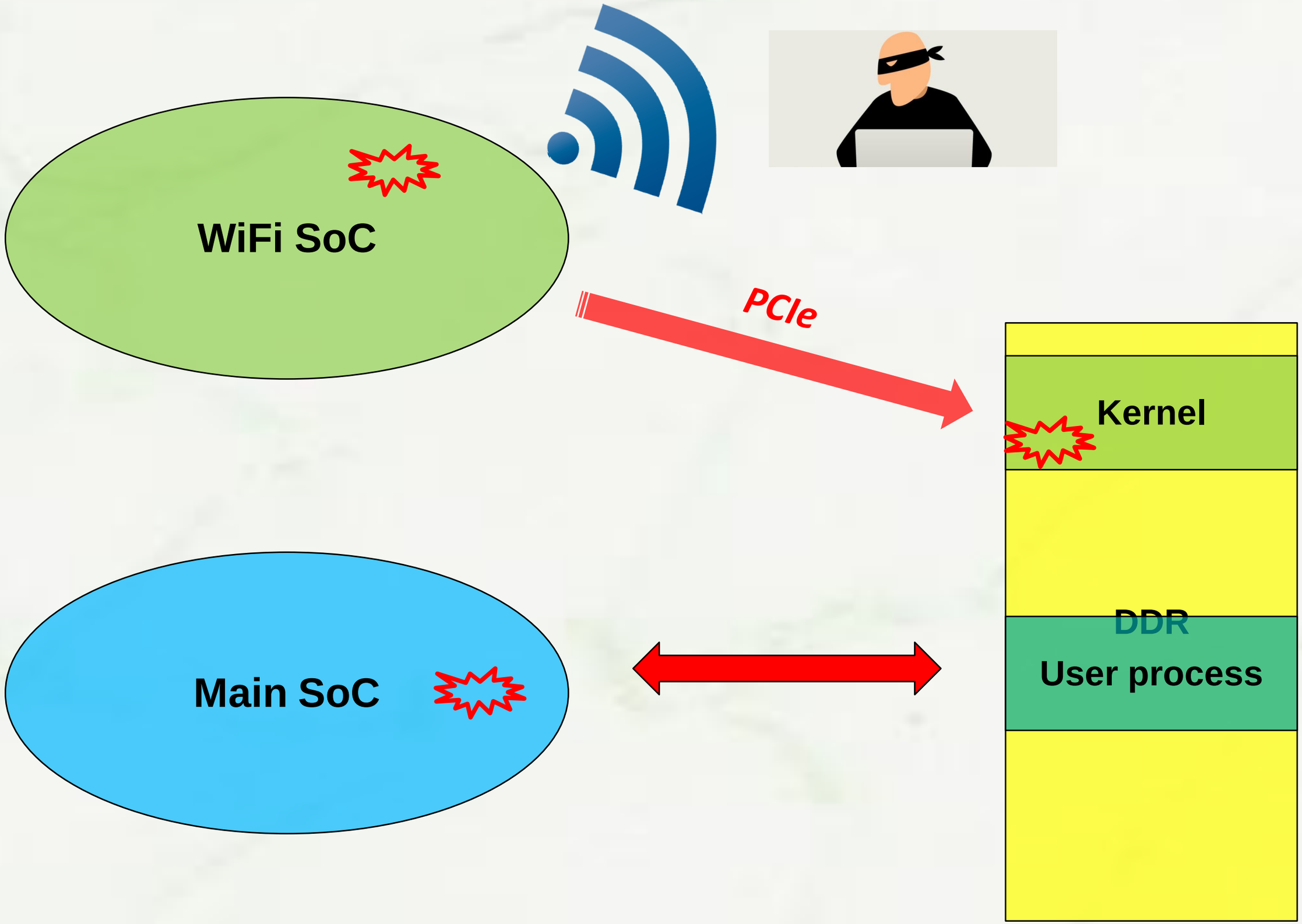
A simplistic model



Reality: other IPs can access DDR



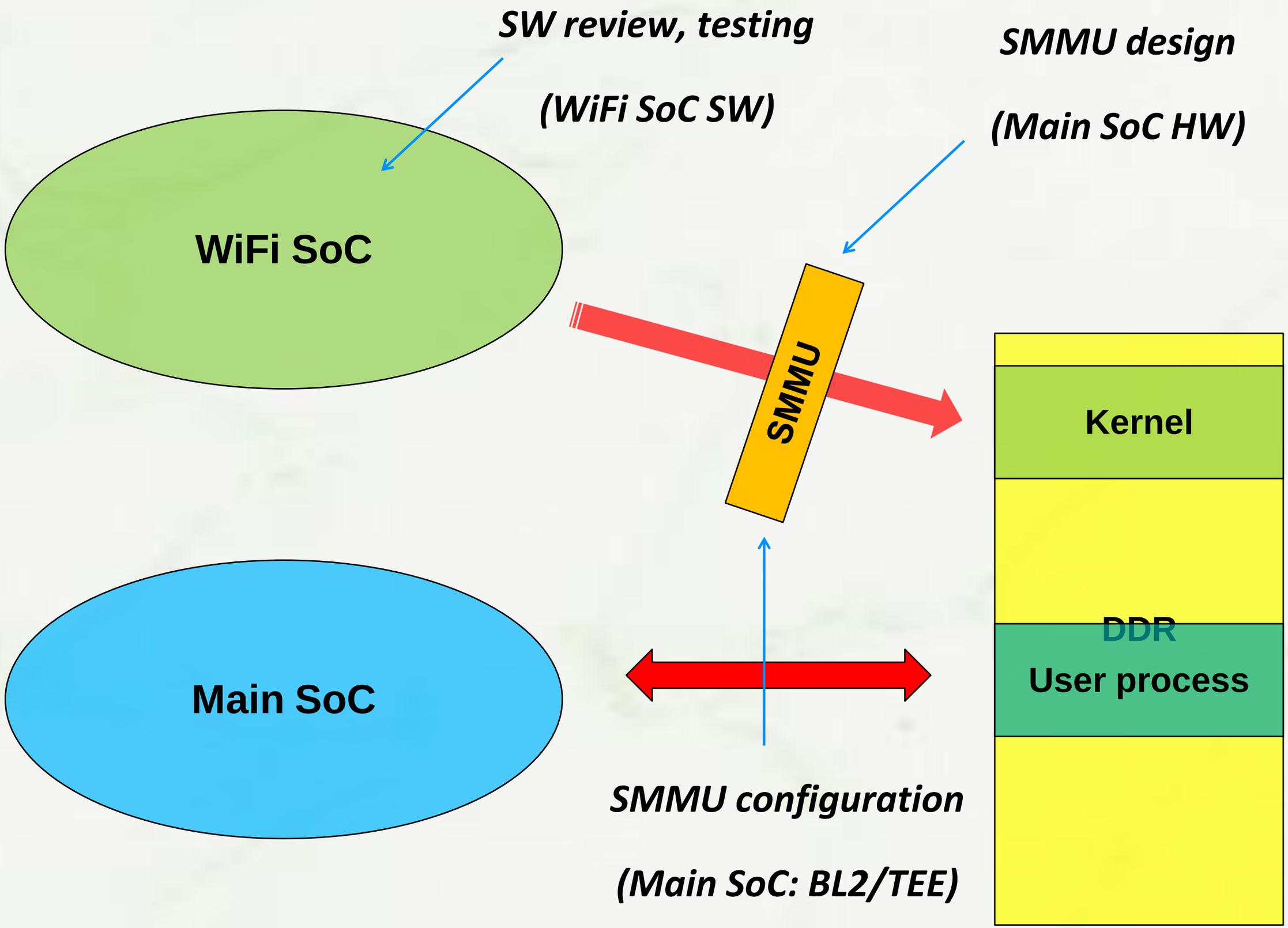
Example: Broadpwn



Remarks

- SoCs → “execution units”
- Other SoCs may have access to Main SoC DDR
- May not be aware of each others’ Security Models:
  - Kernel vs userspace in SoC1 → Plain addressable memory for SoC2

What could have been done?



And we are missing…

- Other execution units:
  - Audio/Video Processors
  - GPUs
  - Power Modules
  - …
- DMA-capable IPs:
  - USB
  - Firewire
  - PCMCIA
  - PCIe
  - …
- Other Bus masters IPs

There can be hundreds…

Assumption of Isolation

“There is only sub-system. Mine”

Reality

Bus masters

Conclusions

Reflections

- *System Security:*
  - *Threat Models may differ between Components*
  - *Security(System)* $\neq$ $\sum_i$ *Security(Component$_i$)*
- *Security Evaluation:*
  - *Context and **system-level information required** for assessment*
  - *Code reviews cannot identify all vulnerabilities (e.g. FI)*
- *Design:*
  - *HW and SW must cooperate. **Across the whole system**.*
    - *Regardless of Manufacturer, Department, Teams boundaries*
  - *Protect **FROM** other sub-systems*

“Unchecked assumptions at boundaries can be fatal”

Recommendations

- ***Estabish a system-level threat model:***
  - Apply and verify consistency everywhere
- For every HW/SW component:
  - *Gain understanding of use case and threat models*
  - *Test and review thoroughly*
- Security assessment and testing MUST consider*:*
  - *System Threat Model*
- For every 3rd party component ask:
  1. *Threat Model*
  2. *Security practices and processes*
  3. *A security evaluation report*
  - ***You are already paying for it.***

Contacts

PULSE

Cristofaro Mune

Product Security Consultant

c.mune@pulse-sec.com