HARDENING SECURE BOOT ON EMBEDDED DEVICES FOR HOSTILE ENVIRONMENTS

riscure

Niek Timmers

niek@riscure.com

@tieknimmers

AVAILABLE

Albert Spruyt

albert.spruyt@gmail.com

PULSE

Cristofaro Mune

c.mune@pulse-sec.com

@pulsoid

WHY THIS TALK?

SOME HISTORY...

2003	Hacking the Xbox
2008	Hacking the iPhone @ 25c3
2010	Console Hacking 2010 @ 27c3
2011	Xbox 360 reset glitch
2013	20 ways past secure boot @ HITB KUL
2016	Hacking Nintendo 2016 @ 33c3
2016	Bypassing Secure Boot using Fault Injection @ Black Hat Europe
2017	Secure Initialization of TEEs; when secure boot falls short @ Euskalhack
2018	Nintendo Switch

SECURE BOOT IS STILL OFTEN VULNERABLE...

OUR GOAL

Create a *Secure Boot guidance* for *designers*, *implementers* and *integrators*.

WHITE PAPER

"Notes on Designing Secure Boot."

We are working on it!

THIS PRESENTATION

Offensive focus

Known and new attacks

New perspectives

AGENDA

Introduction

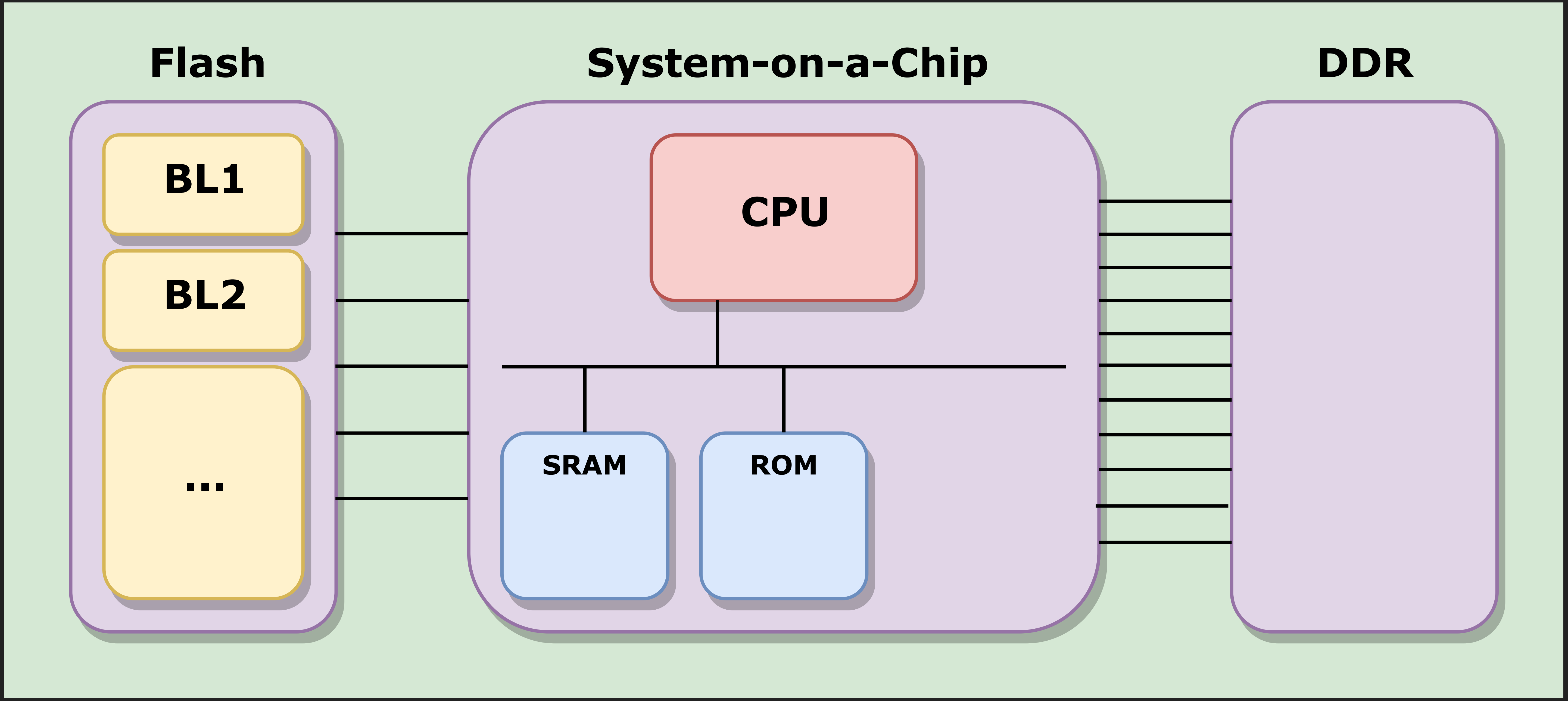
Secure Boot

Attacks and Mitigations

Demo

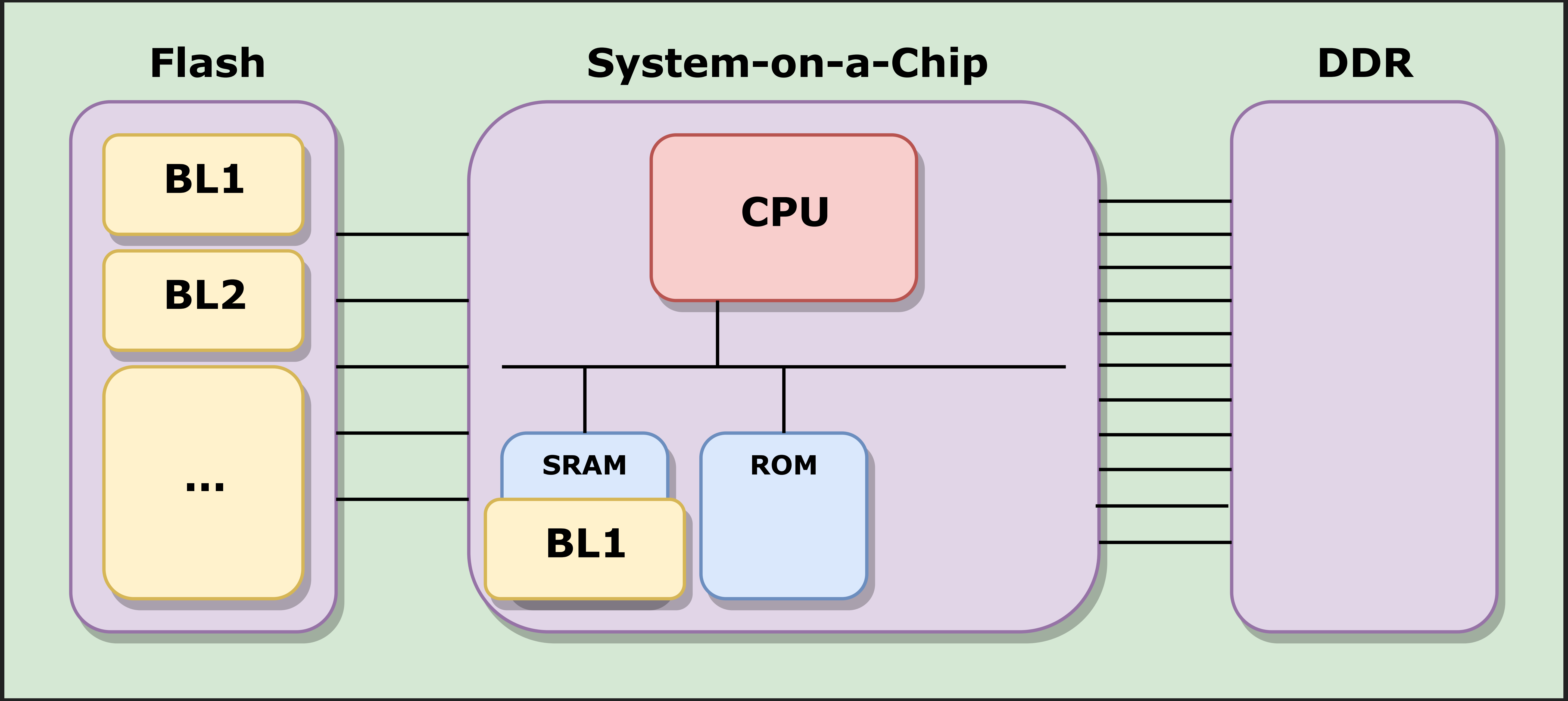
Takeaways

GENERIC DEVICE



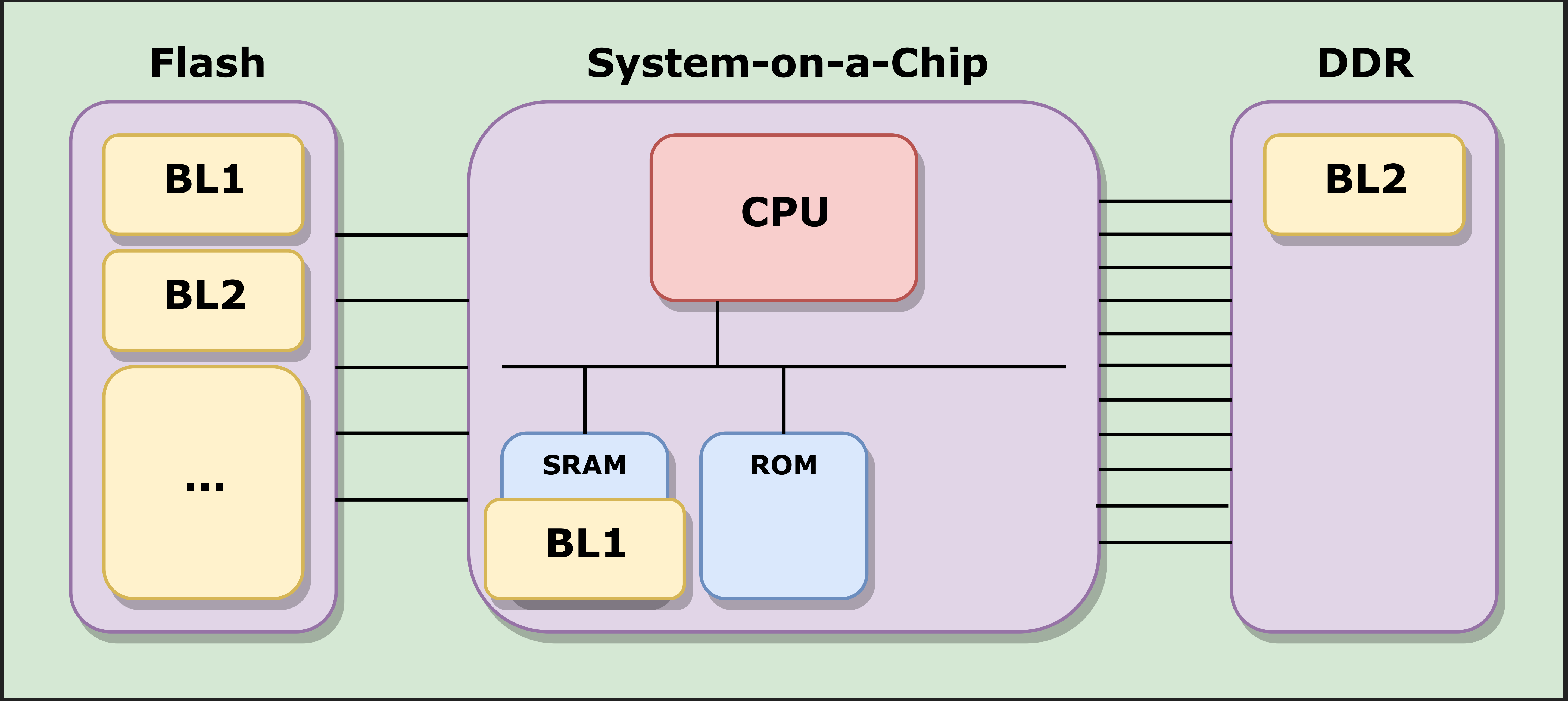
Device is turned off

GENERIC DEVICE



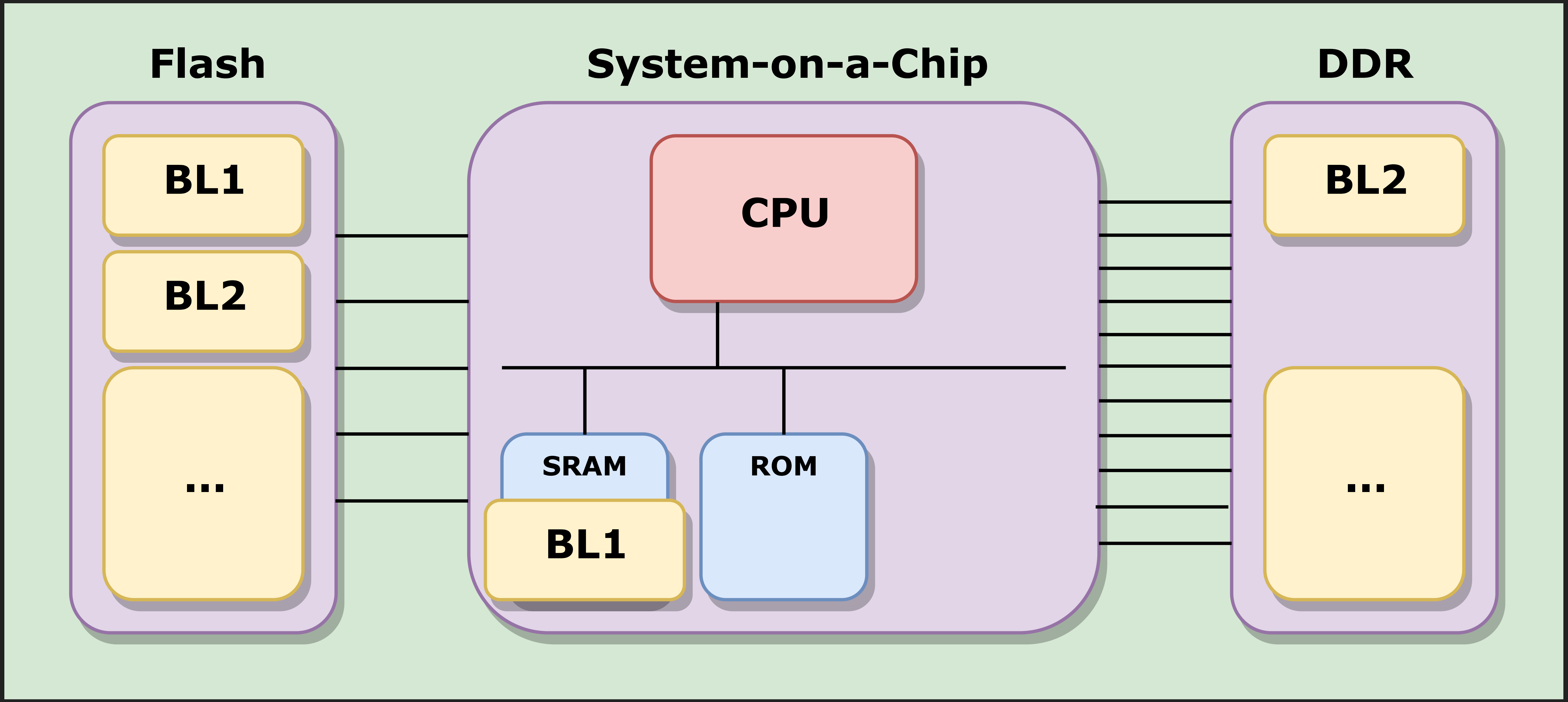
ROM code loads BL1 into internal SRAM

GENERIC DEVICE



BL1 initializes DDR and loads BL2 into DDR

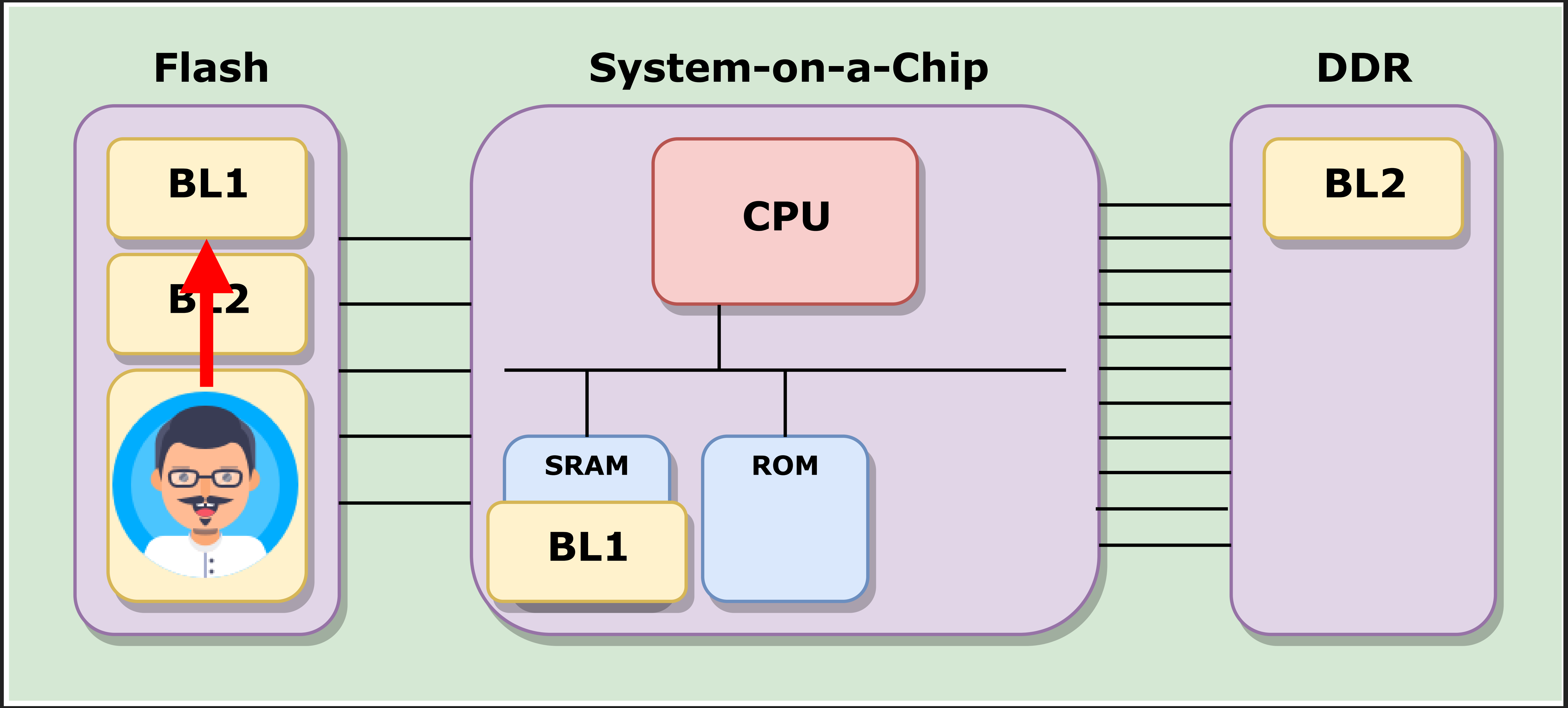
GENERIC DEVICE



And then, more is loaded and executed...

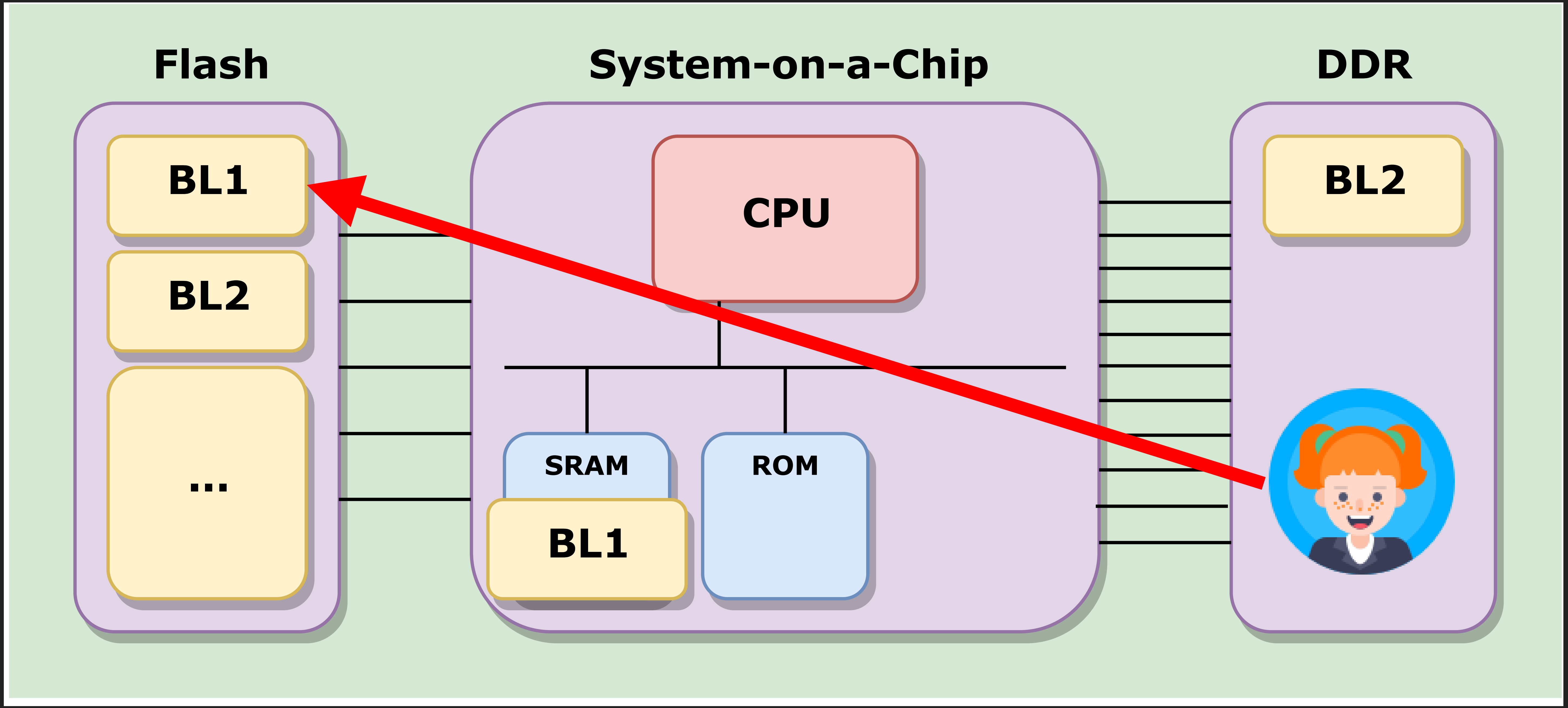
TWO MAJOR THREATS...

ATTACKERS



Attacker 1: hardware hacker modifies flash

ATTACKERS



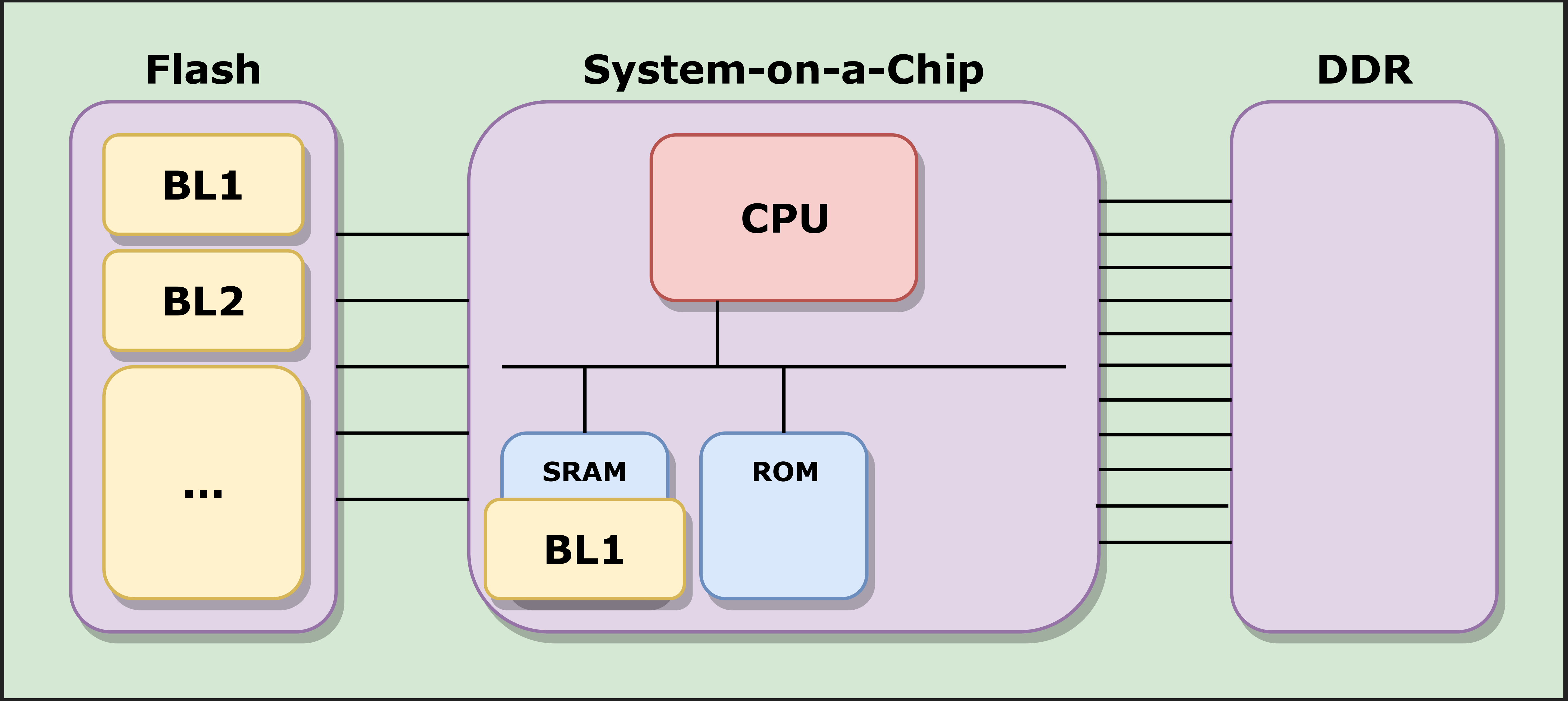
Attacker 2: (remote) software hacker modifies flash

THEREFORE WE NEED SECURE BOOT

SECURE BOOT

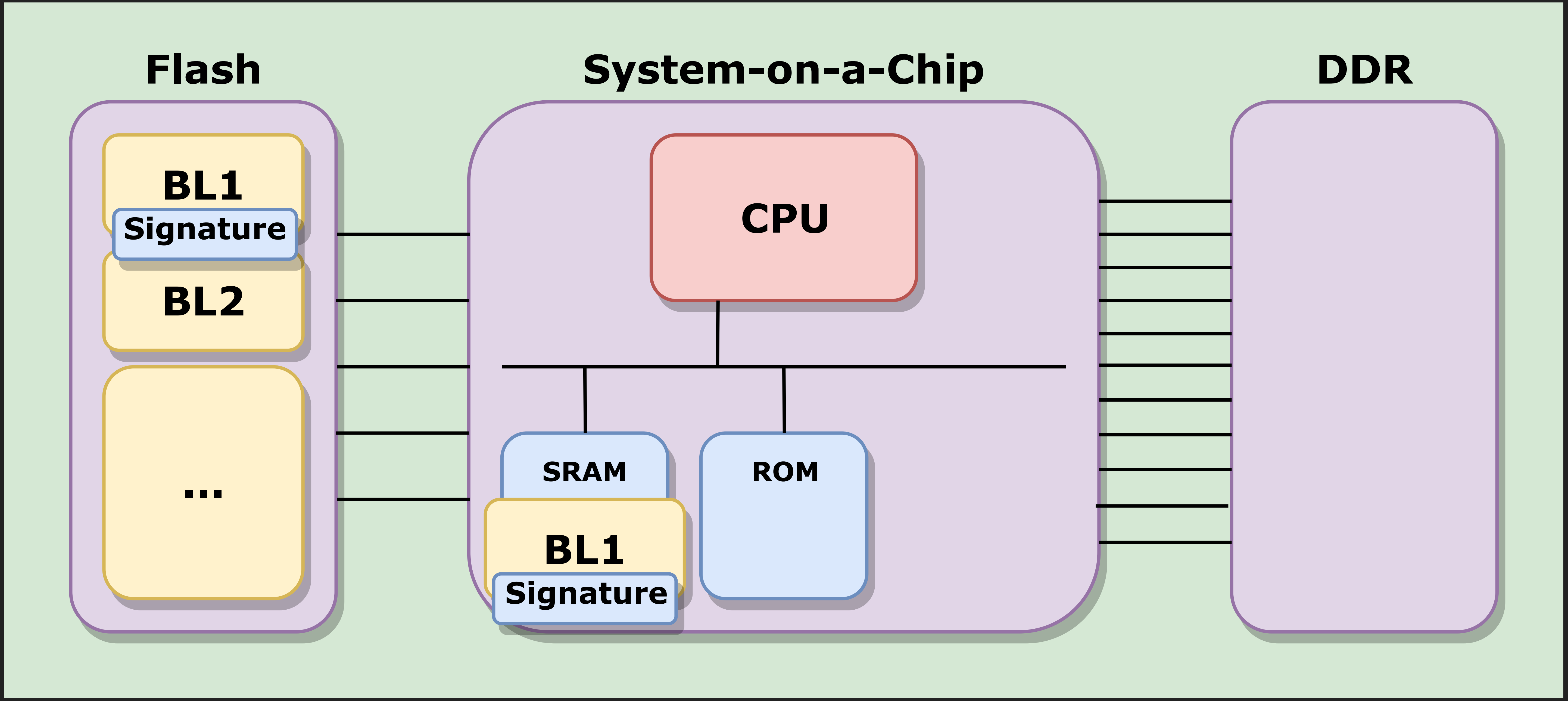
- Authentication of loaded images
- Root of trust embedded in hardware
  - i.e. immutable code and data (e.g. ROM, OTP)

SECURE BOOT



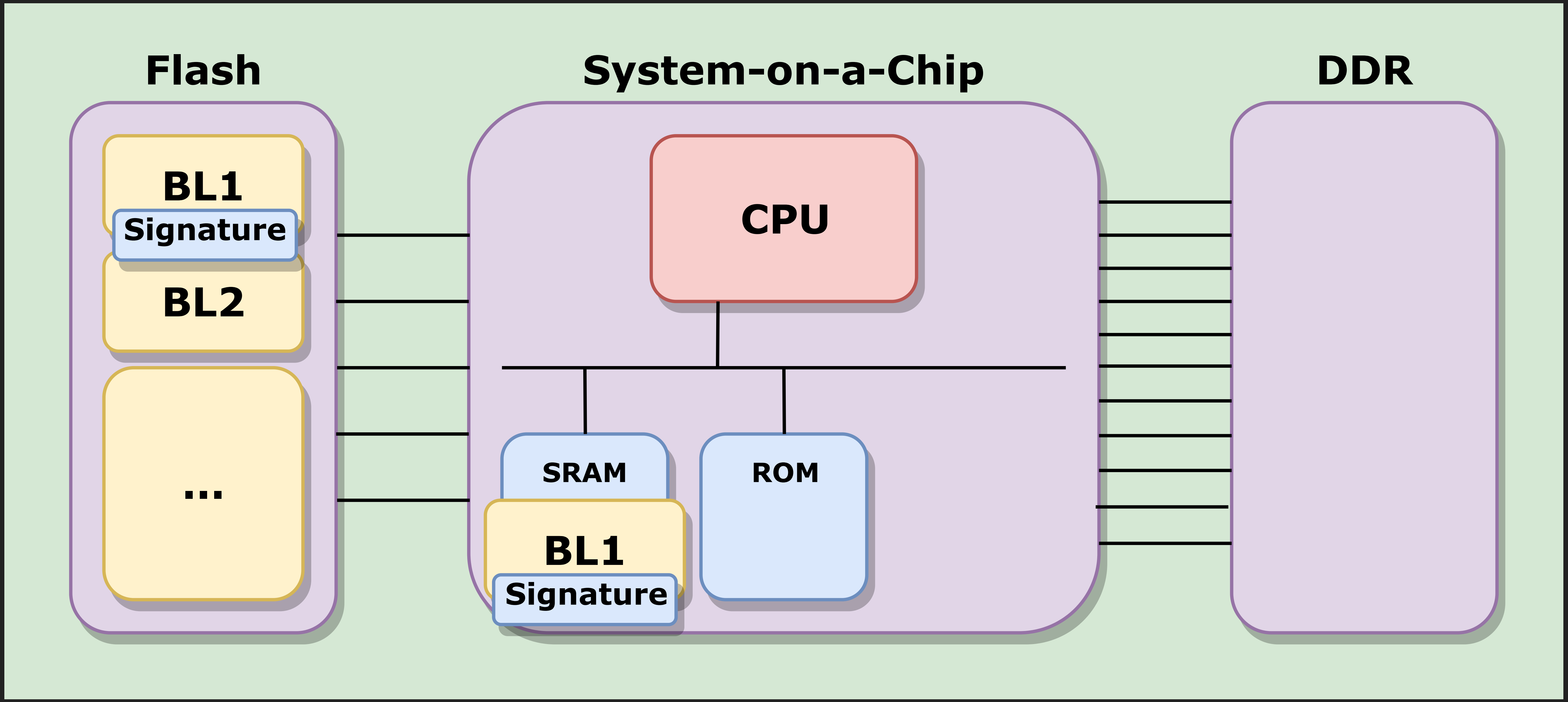
ROM has copied BL1 to SRAM

SECURE BOOT



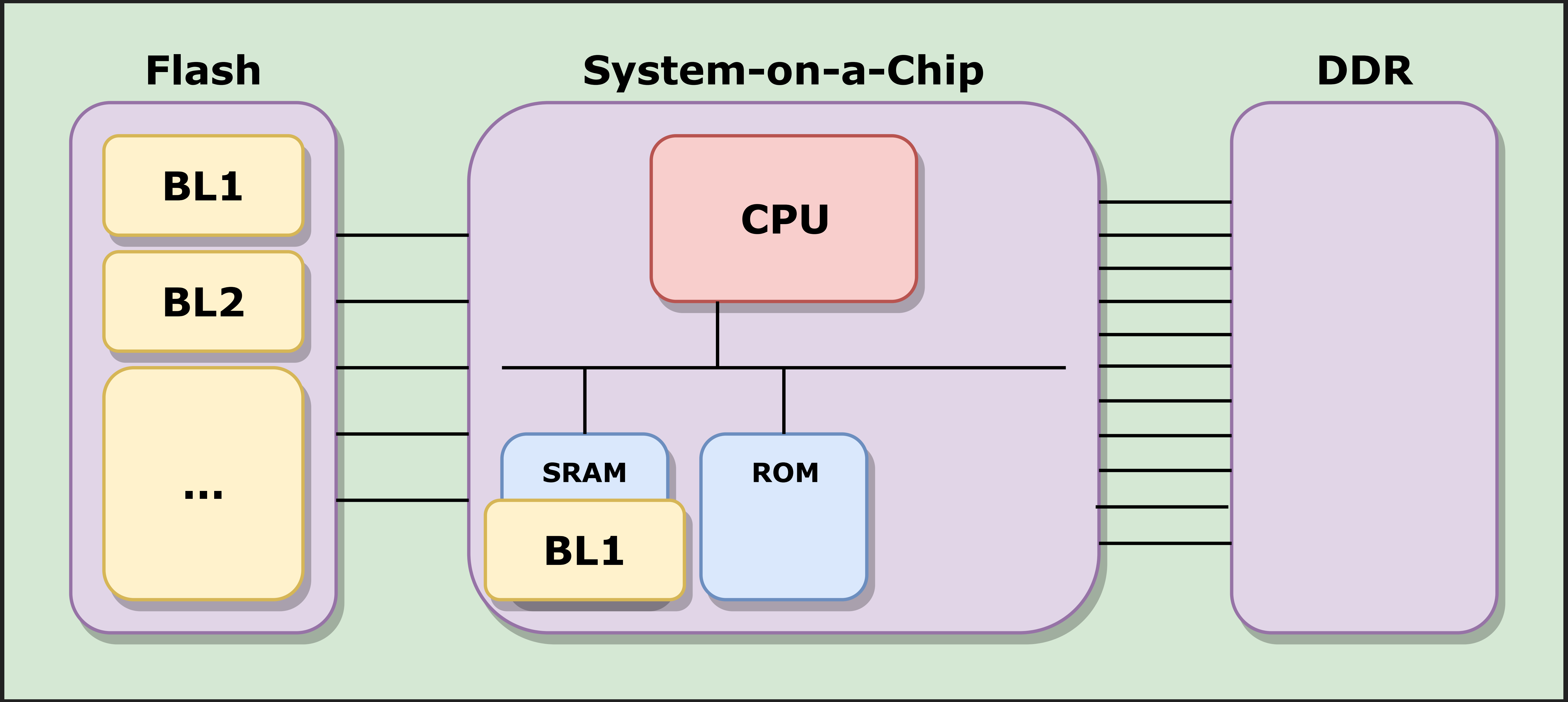
ROM calculates the BL1 hash

SECURE BOOT



ROM compares the hash against the reference from the signature

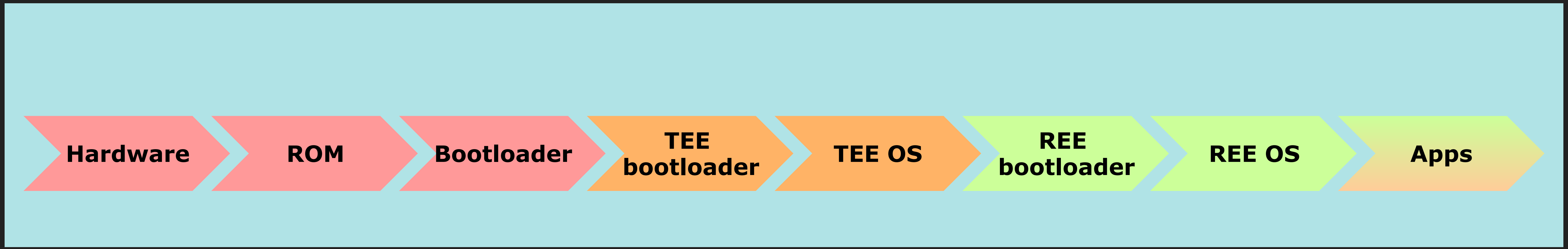
SECURE BOOT



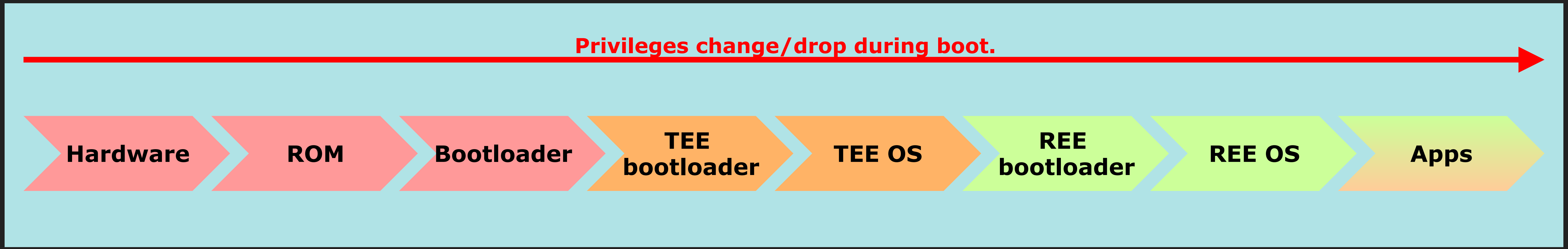
BL1 is executed

THE REAL WORLD IS A LITTLE MORE COMPLEX...

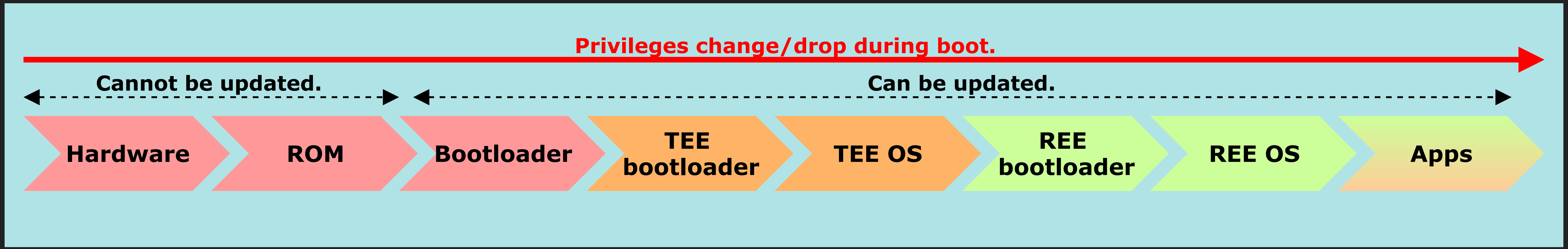
SECURE BOOT FLOW



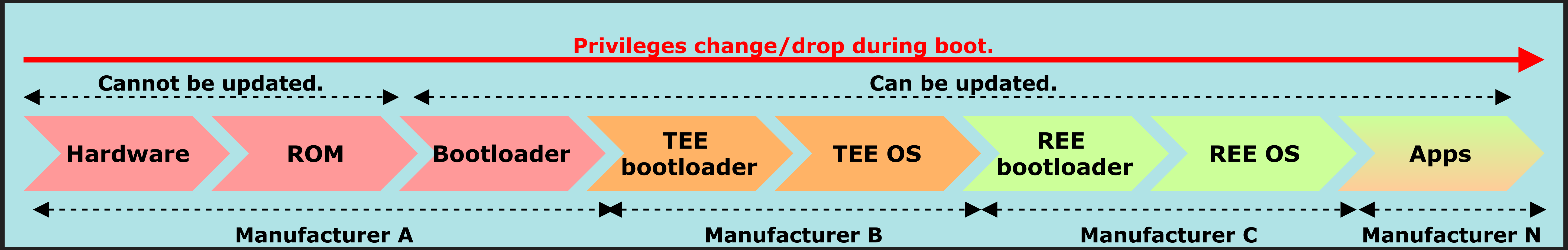
SECURE BOOT FLOW



SECURE BOOT FLOW



SECURE BOOT FLOW



Lots of different interests!

MITIGATING THREATS

- Modifying code/data in flash
- Insecure updates
- Creating a persistent foothold
- Access to keys, code and crypto engines
- Escalating privileges (e.g. REE to TEE)

ATTACK SURFACE

Broken design

OR

Broken implementation

ATTACK SURFACE

Broken design

OR

Broken implementation

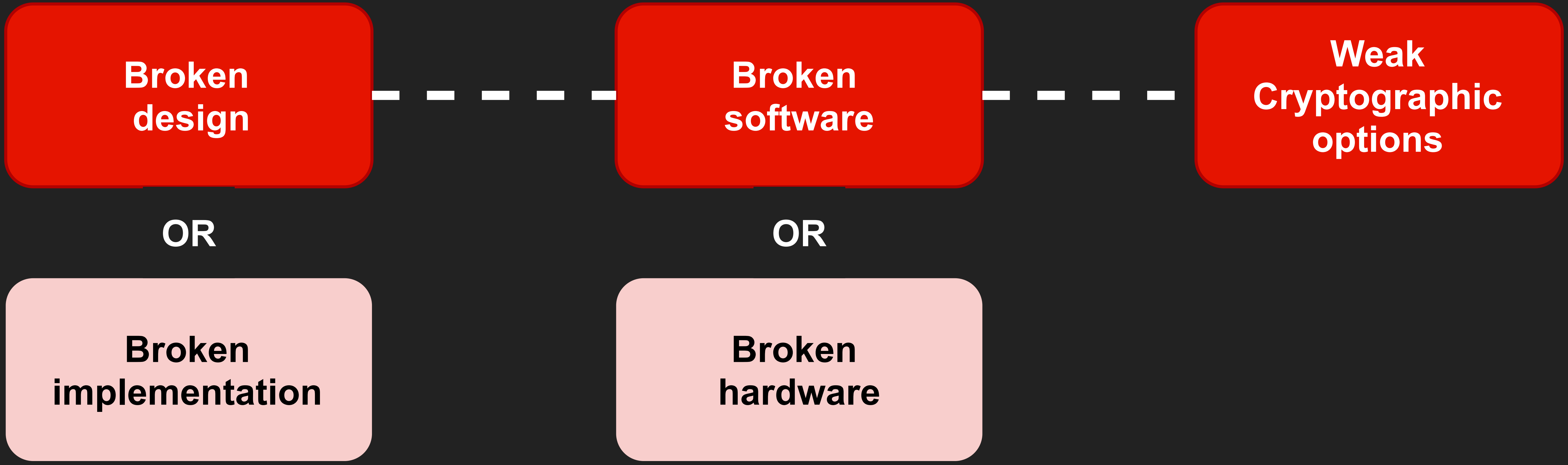
Broken software

OR

Broken hardware

WHAT GOES WRONG IN THE FIELD...

Amlogic S905 SoC BootROM vulnerability



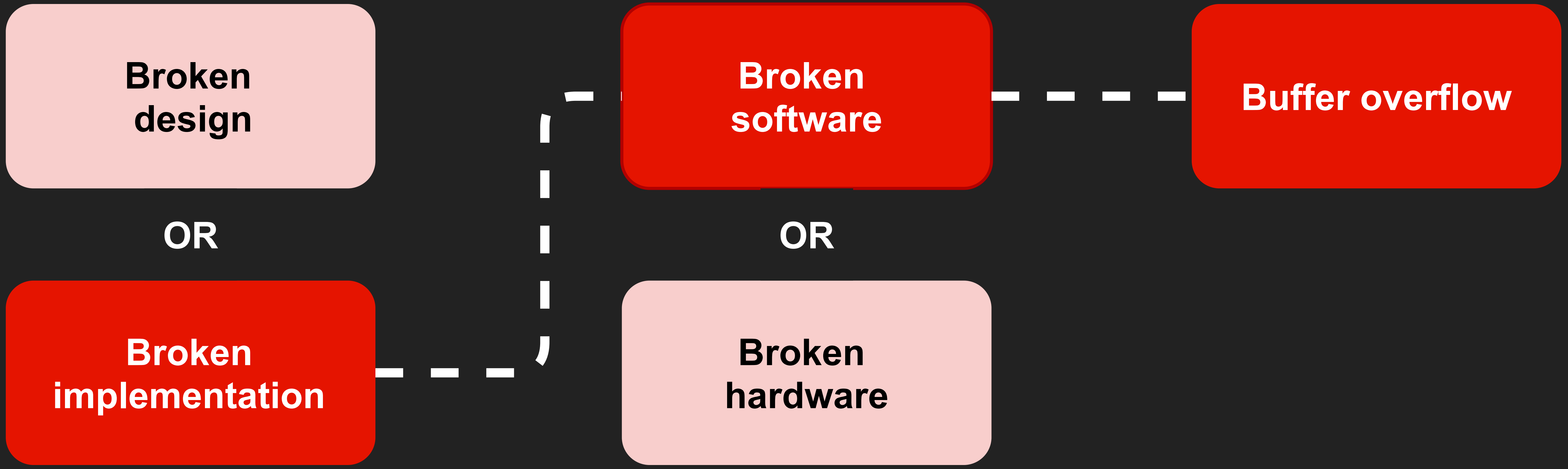
Secure Boot is bypassed, and BootROM is dumped, by downgrading from RSA to SHA

Credit: fredericb

MITIGATIONS:

- Do not support weak cryptographic options
- Limit the amount of options

Nintendo Switch BootROM vulnerability



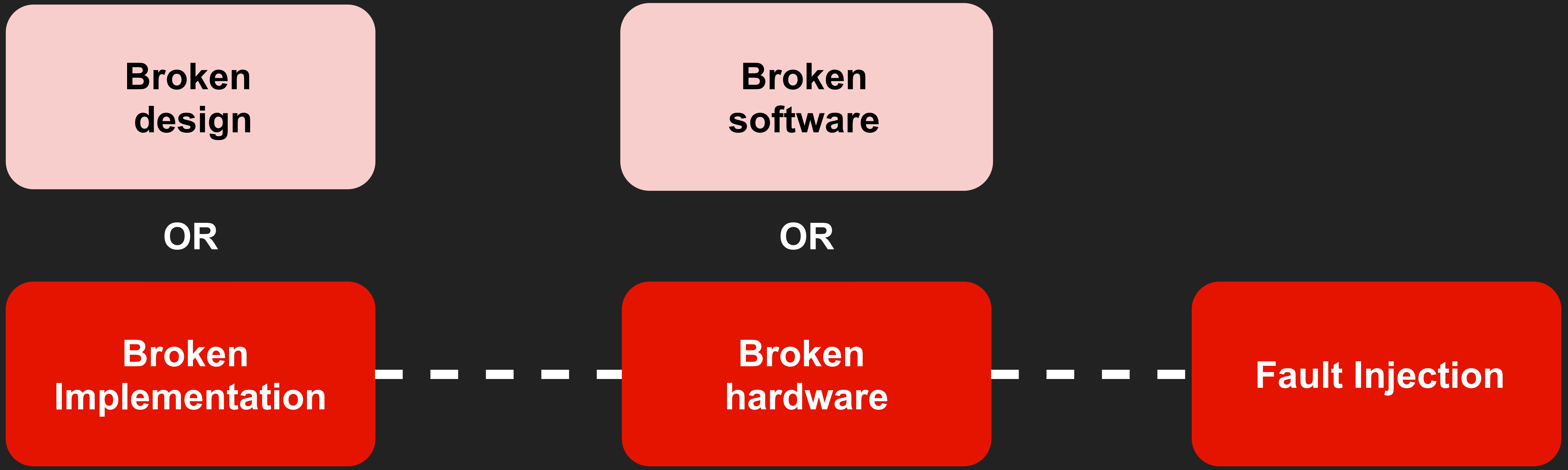
Buffer overflow in the USB recovery mode

Credit: fail0verflow and Cease & DeSwitch

MITIGATIONS:

- Write secure software ;)
- Make software exploitation hard
  - i.e. stack cookies, ASLR, CFI, etc.
  - Use memory protections to enforce W^X
    - e.g. MPU, MMU, IOMMU, etc.

SWITCH FAULT INJECTION



SKIP HASH VERIFICATION USING VOLTAGE FAULT INJECTION

FAULT INJECTION (FI)

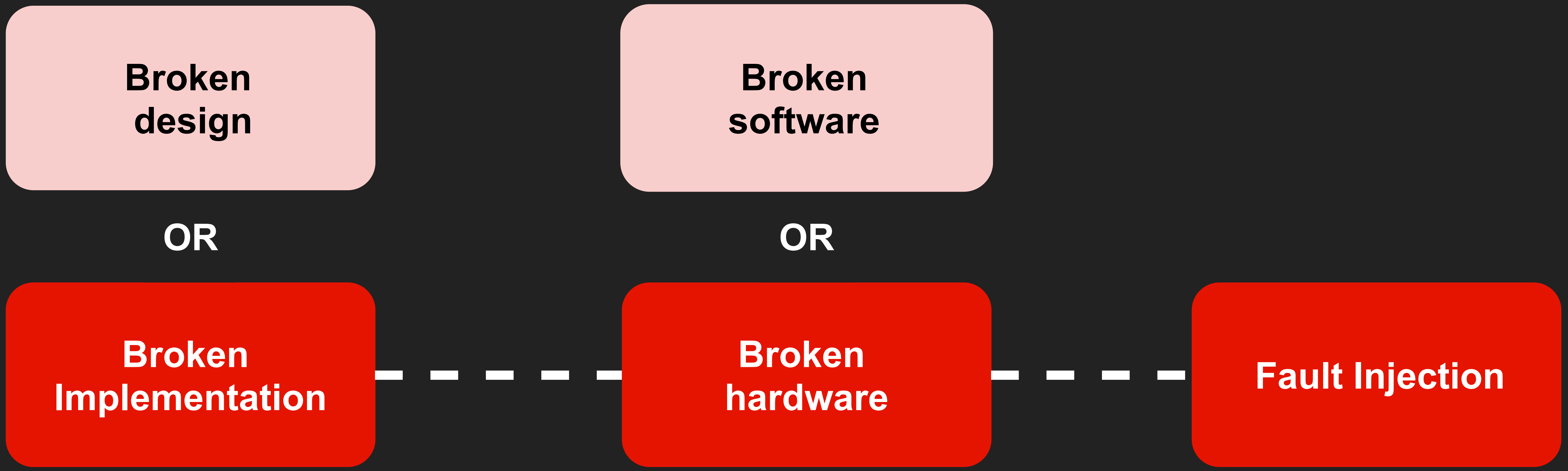
- Make glitches with e.g.: EM, light, clock, power, heat
- Use a glitch to introduce a fault in a device
- Model faults:
  - Instruction skipping
  - Instruction/data corruption

FI ALTERS THE INTENDED BEHAVIOR OF HW AND SW

FAULT INJECTION MITIGATIONS

- Software
  - Redundancy (e.g. double checks)
  - Random delays
- Hardware
  - Redundancy
  - Glitch detectors
  - Clock randomization

Viva La Vita Vida fault injection attack



Introducing a classic buffer overflow using Voltage Fault Injection

Credit: Yifan Lu and Davee @ 35c3

MITIGATIONS:

It's Fault injection so use FI mitigations

It's Software exploit so use exploit mitigations

DESIGNING SECURE BOOT AINT EASY!

ESPECIALLY CONSIDERING THE CONSTRAINTS...

Initializing hardware

Interfacing with peripherals

Performance

Code size

Keeping engineering cost low

Recoverability

Customer needs

IT'S IMPORTANT TO GET IT RIGHT

WRONG SECURITY IS EXPENSIVE

Tape out

Crisis management

PR damage

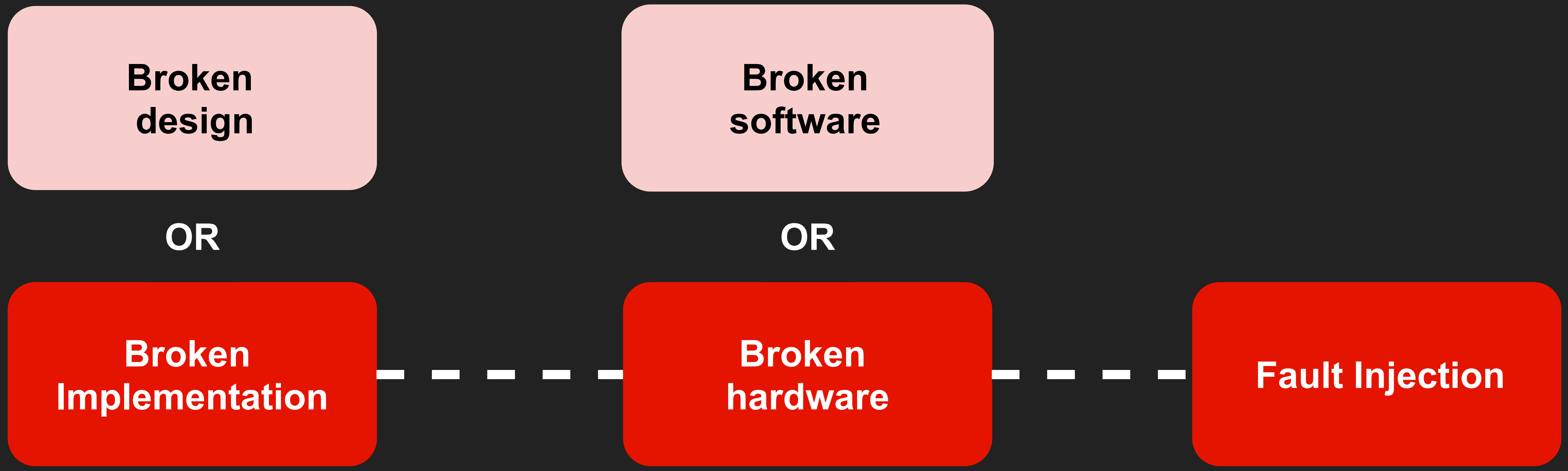
Time to market

Recall of devices/unsold inventory

Additional engineering time

HAS THE WORLD SEEN IT ALL?

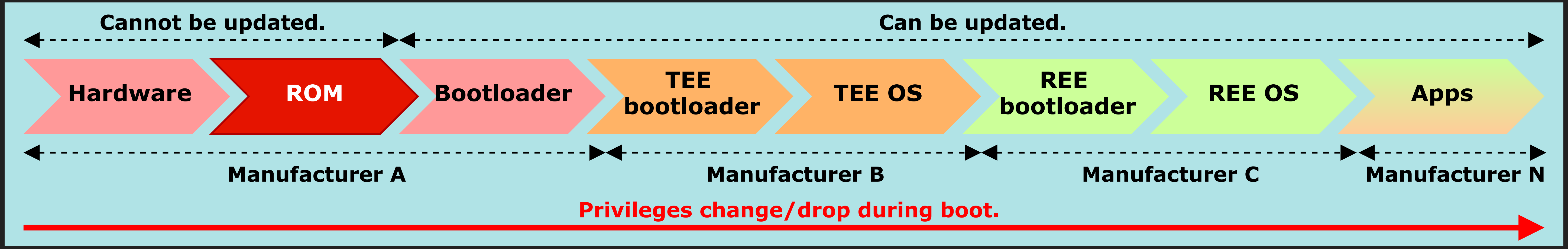
FAULT INJECTION ON OTP TRANSFER



Attacking Secure Boot before any code is executed!

LET'S LOOK AT THIS ONE IN DETAIL

OTP AND SECURE BOOT



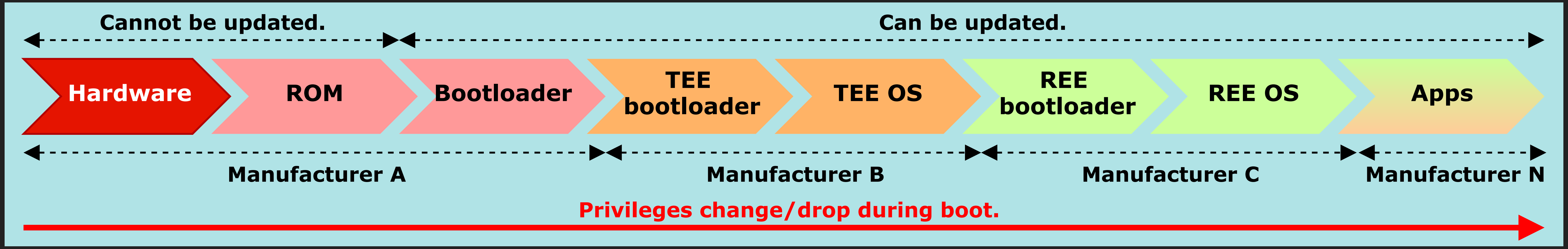
ROM code uses values from OTP for enabling/disabling security features.

EXAMPLE

```
memcpy(I_SRAM, I_FLASH, I_SIZE);                // 1. Copy image
memcpy(S_SRAM, S_FLASH, S_SIZE);                // 2. Copy signature

if (*(OTP_SHADOW) >> 17 & 0x1) {                // 3. Check if enabled
    if(SHA256(I_SRAM, I_SIZE, I_HASH)) {        // 4. Calculate hash
        while(1);
    }

    if(verify(PUBKEY, S_SRAM, I_HASH)) {        // 5. Verify image
        while(1);
    }
}

jump();                                         // 6. Jump to next image
```

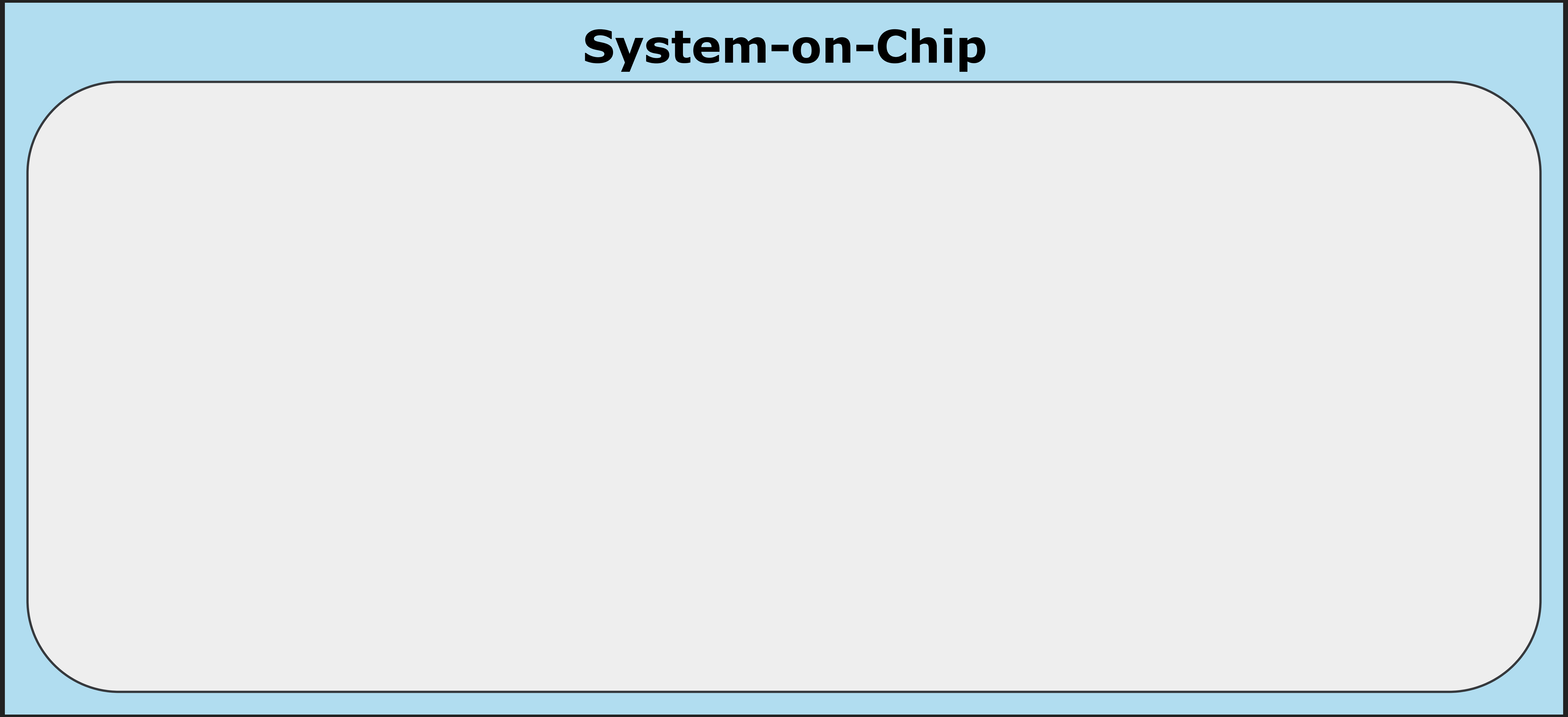
Value stored in shadow registers. Populated by OTP Transfer.

POPULATING SHADOW REGISTERS



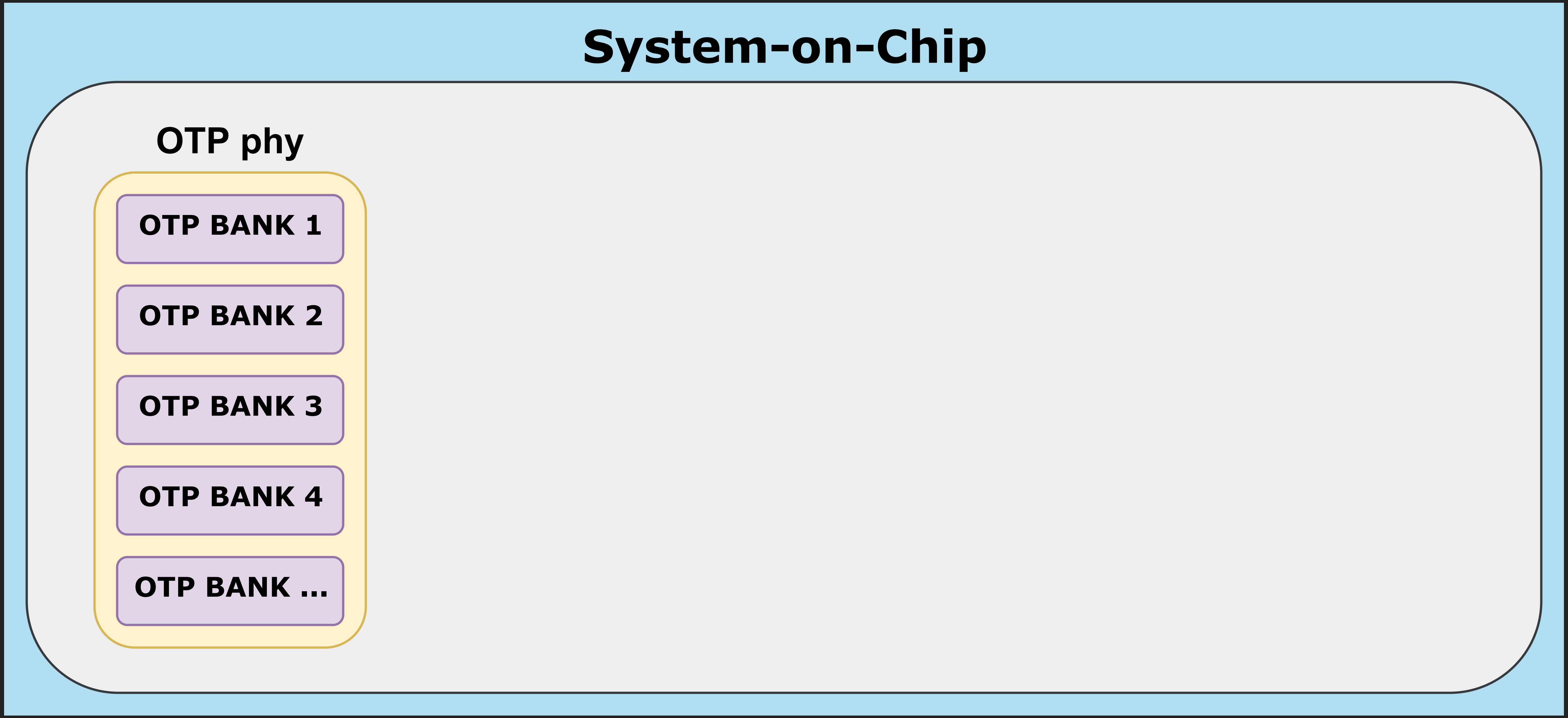
OTP Transfer performed in hardware. BEFORE any ROM code is executed.

OTP TRANSFER 1/5



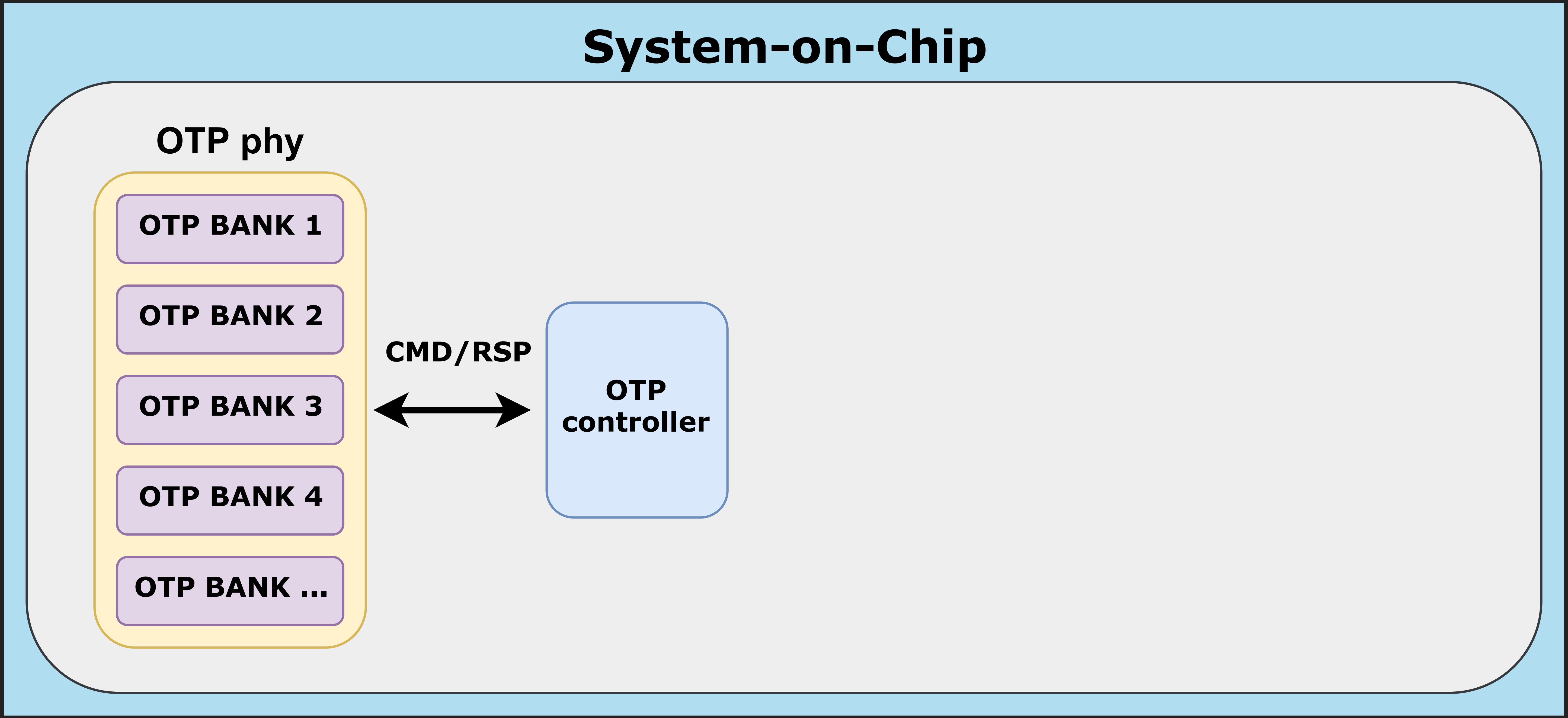
A typical System-on-Chip (SoC)

OTP TRANSFER 2/5



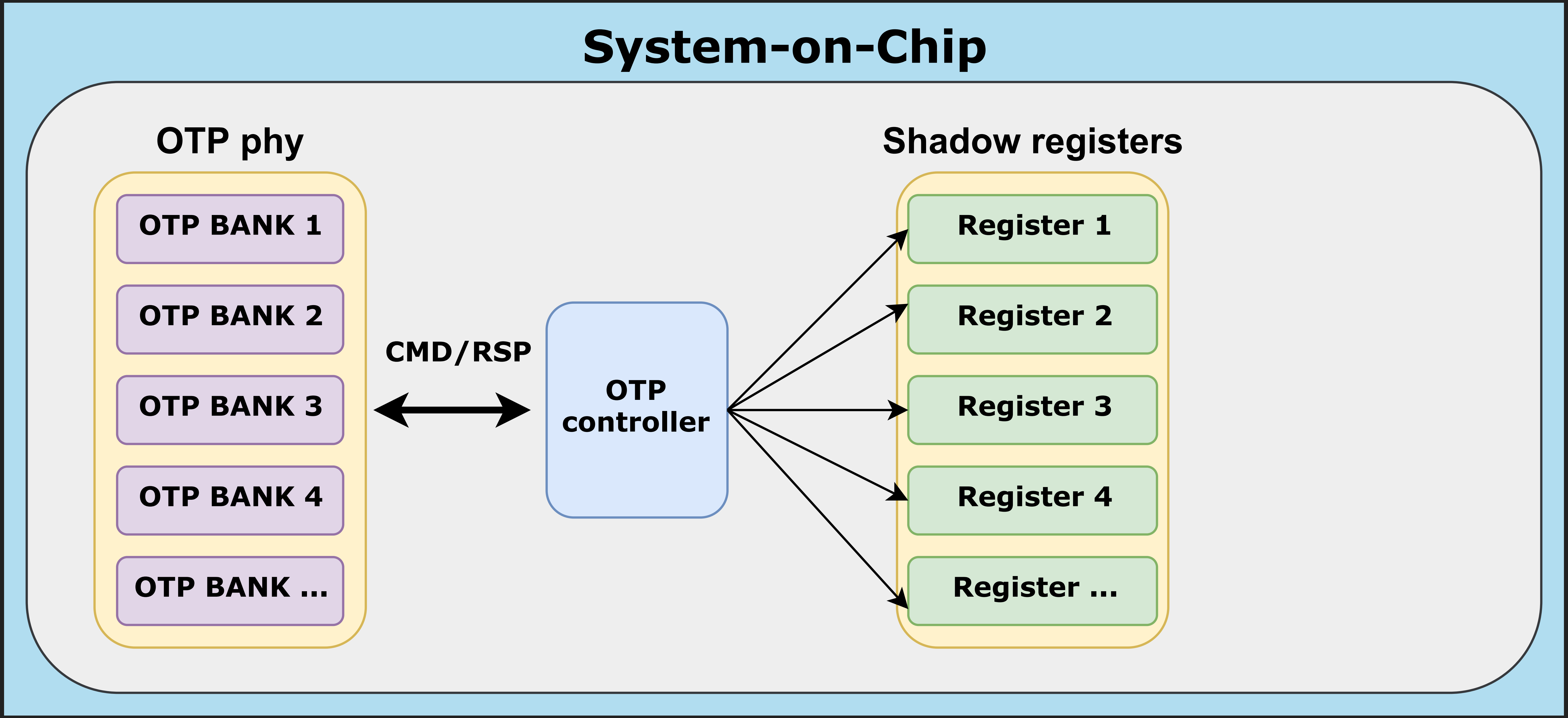
Contains a special OTP hardware block

OTP TRANSFER 3/5



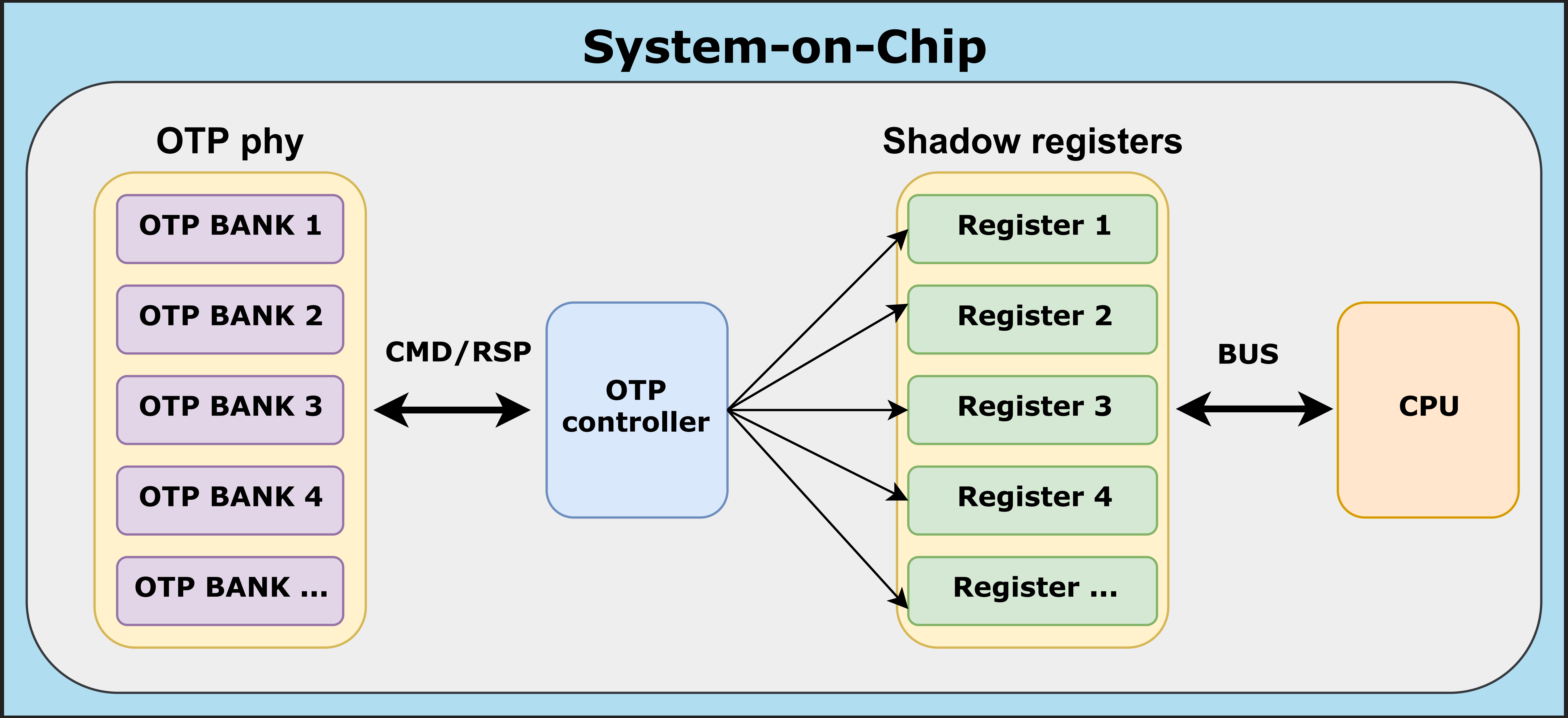
Which is wrapped by a hardware controller

OTP TRANSFER 4/5



This controller copies the OTP values to dedicated registers after SoC reset

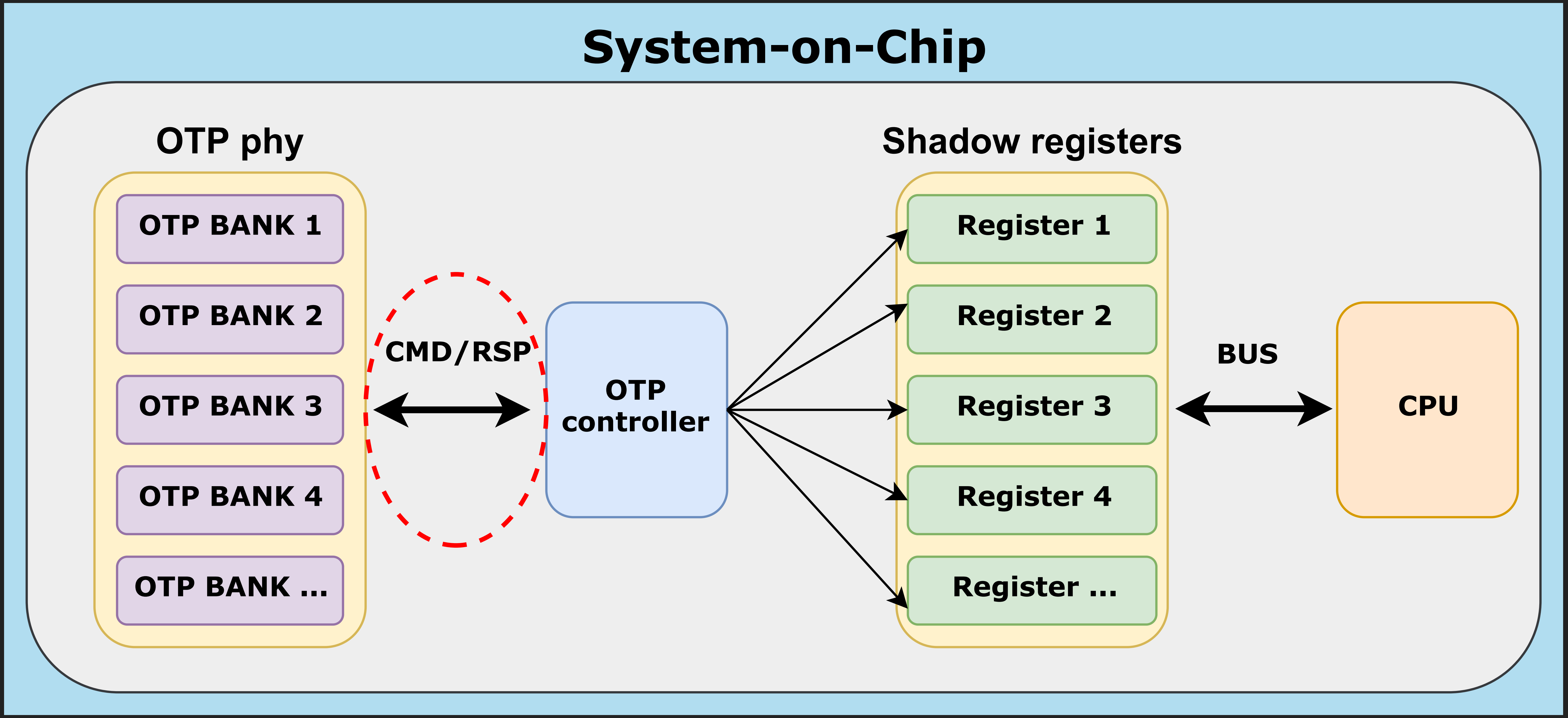
OTP TRANSFER 5/5



CPU is released from reset. Shadow registers can be read using system bus.

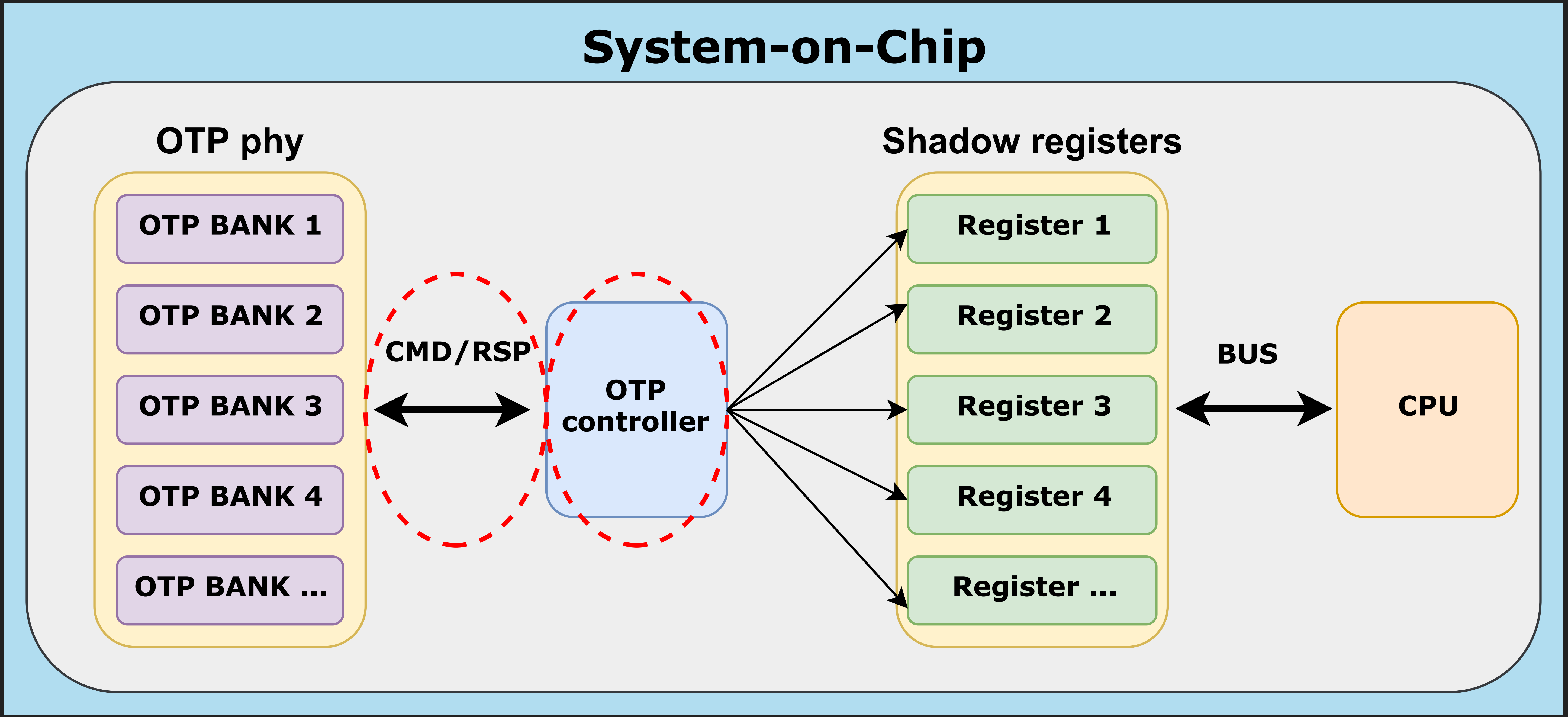
WHERE CAN WE ATTACK?

ANYWHERE!



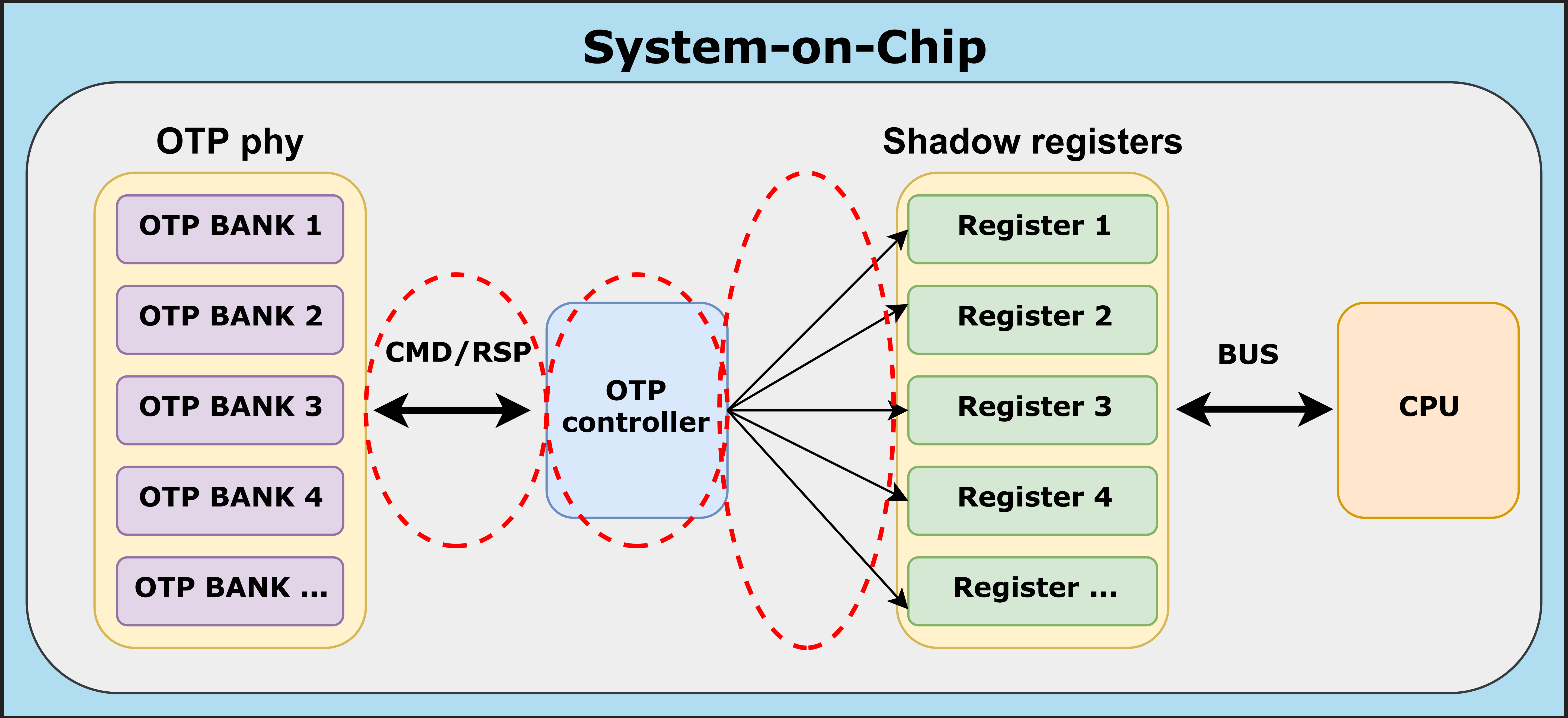
Attack the bus between the OTP PHY and the OTP controller.

ANYWHERE!



Attack the OTP controller directly.

ANYWHERE!



Attack the bus between the OTP controller and the shadow registers.

WE CAN AFFECT

SIGNATURE VERIFICATION

AND/OR

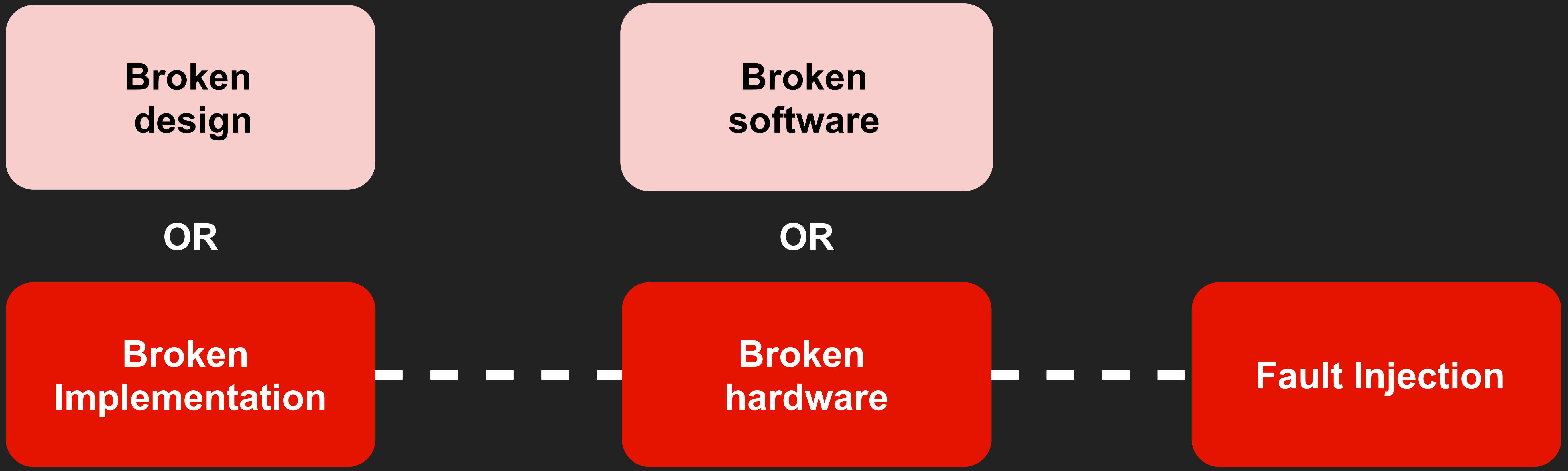
STAGE ENCRYPTION

BYPASSING

(ENCRYPTED) SECURE BOOT

THAT WAS FUN; LET'S DO ANOTHER ONE!

FAULT INJECTION ON ENCRYPTED SECURE BOOT



...WITHOUT AN ENCRYPTION KEY!

SIGNATURE VERIFICATION

```
memcpy(I_SRAM, I_FLASH, I_SIZE);                   // 1. Copy image
memcpy(S_SRAM, S_FLASH, S_SIZE);                   // 2. Copy signature

if (*(OTP_SHADOW) >> 17 & 0x1) {                   // 3. Check if enabled
    if(SHA256(I_SRAM, I_SIZE, I_HASH)) {           // 4. Calculate hash
        while(1);
    }

    if(verify(PUBKEY, S_SRAM, I_HASH)) {           // 5. Verify image
        while(1);
    }
}

jump();                                            // 6. Jump to next image
```

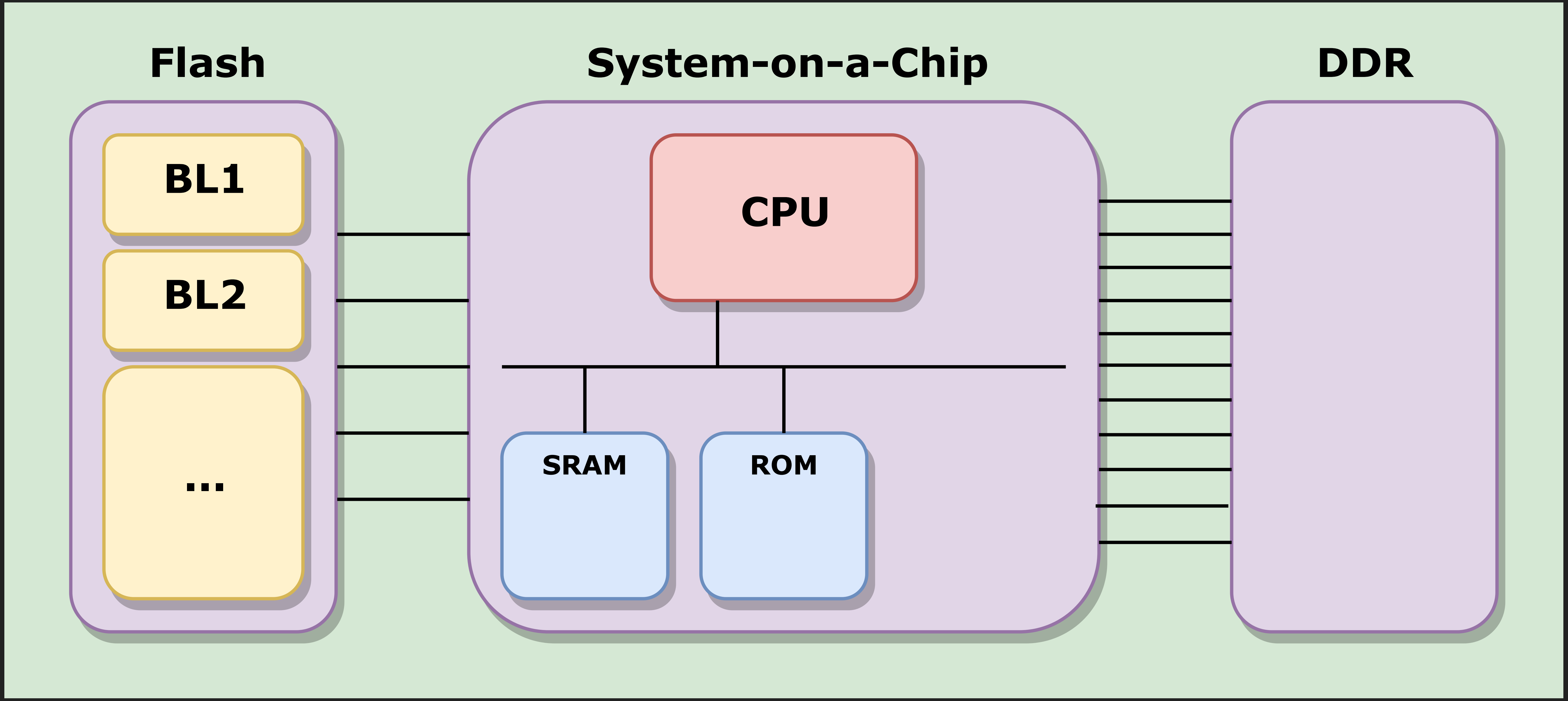
FAULT INJECTION FAULT MODEL

"Instruction skipping"

- Faults can cause "instruction not to be executed"
- Inaccurate but sufficient
- Widely adopted (by academia and industry)
- Useful for affecting the code flow

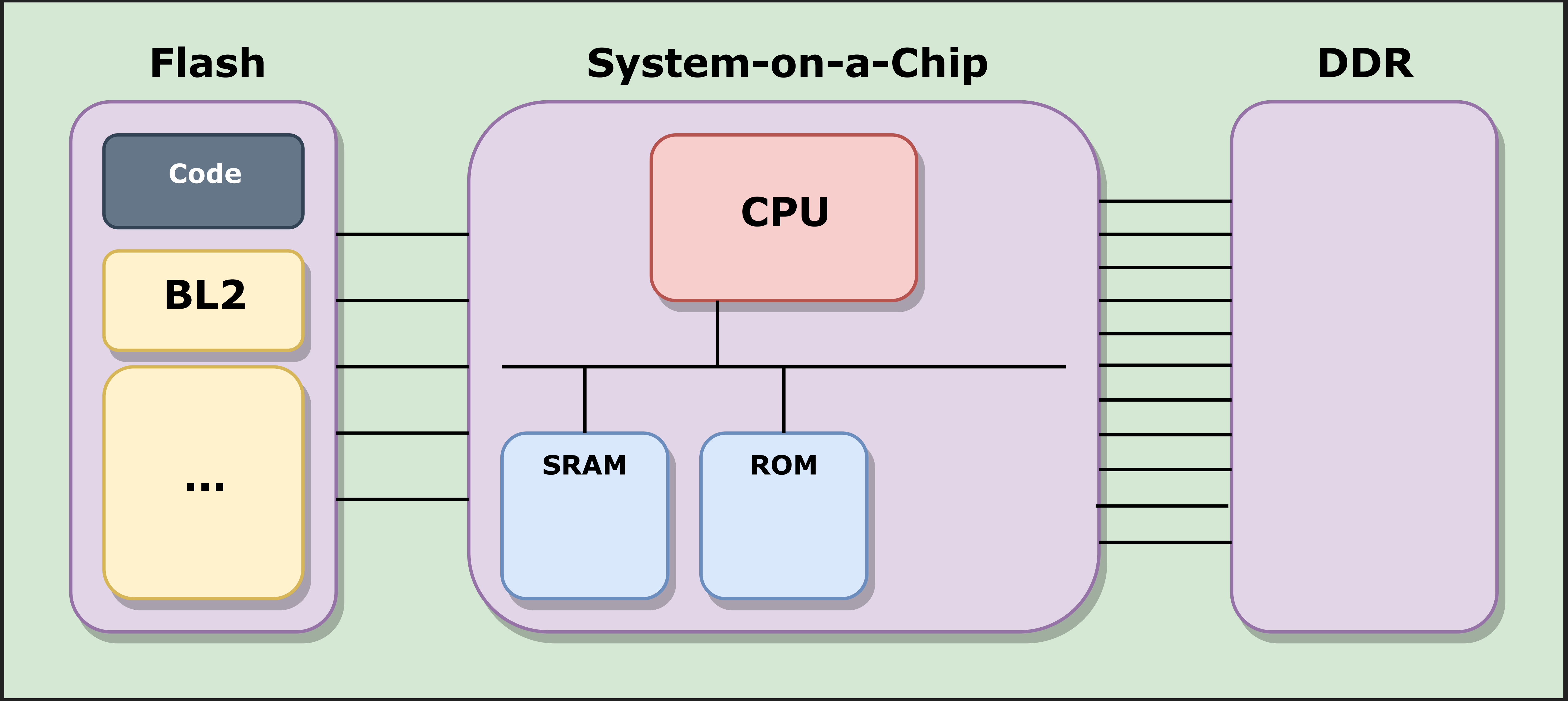
LET'S USE IT FOR BYPASSING SECURE BOOT!

A TEXTBOOK ATTACK 1/3



Device is turned off

A TEXTBOOK ATTACK 2/3



Replace BL1 with a malicious image

A TEXTBOOK ATTACK 3/3

```
memcpy(I_SRAM, I_FLASH, I_SIZE);                    // 1. Copy image
memcpy(S_SRAM, S_FLASH, S_SIZE);                    // 2. Copy signature

if (*(OTP_SHADOW) >> 17 & 0x1) {                    // 3. Check if enabled
    if(SHA256(I_SRAM, I_SIZE, I_HASH)) {            // 4. Calculate hash
        while(1);
    }

    if(verify(PUBKEY, S_SRAM, I_HASH)) {            // 5. Glitch here!
        while(1);
    }
}

jump();                                             // 6. Jump to next image
```

Skip verify function call and boot an malicious image

GLITCH AT THE RIGHT MOMENT AND PROFIT!

WHAT IF BL1 IS ENCRYPTED?

ENCRYPTED SECURE BOOT

```
memcpy(I_SRAM, I_FLASH, I_SIZE);                      // 1. Copy image
decrypt(SYM_KEY, I_SRAM, I_SIZE);                     // NEW: Decrypt image
memcpy(S_SRAM, S_FLASH, S_SIZE);                      // 2. Copy signature

if (*(OTP_SHADOW) >> 17 & 0x1) {                      // 3. Check if enabled
    if(SHA256(I_SRAM, I_SIZE, I_HASH)) {              // 4. Calculate hash
        while(1);
    }

    if(verify(PUBKEY, S_SRAM, I_HASH)) {              // 5. Glitch here!
        while(1);
    }
}

jump();                                               // 6. Jump to next image
```

The image is decrypted after it is copied and before it is verified!

THE MISSING KEY...

Encryption key needed for creating a malicious image

THAT'S WHY...

FI attacks are often considered infeasible when encrypted Secure Boot is used.

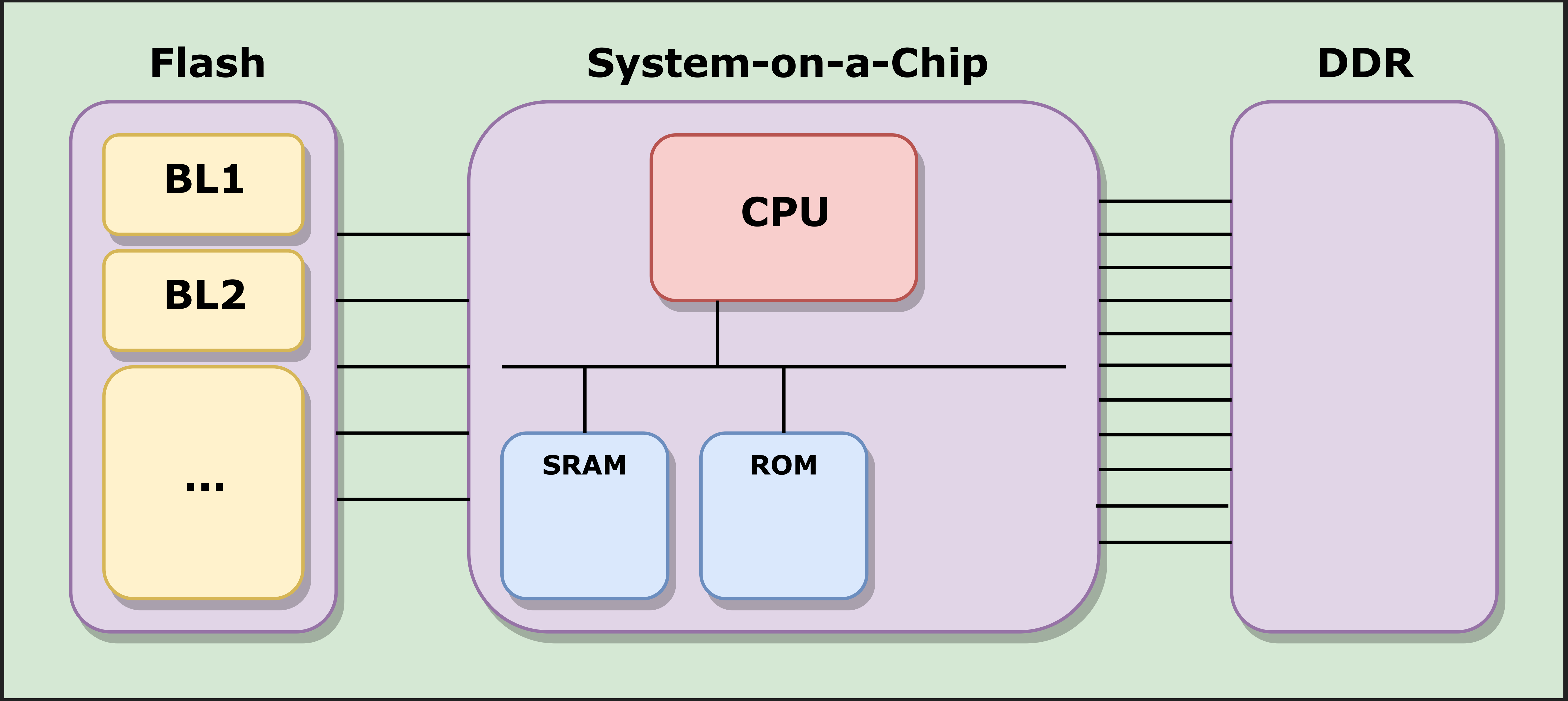
UNTIL NOW!

FAULT INJECTION FAULT MODEL

"Instruction corruption"

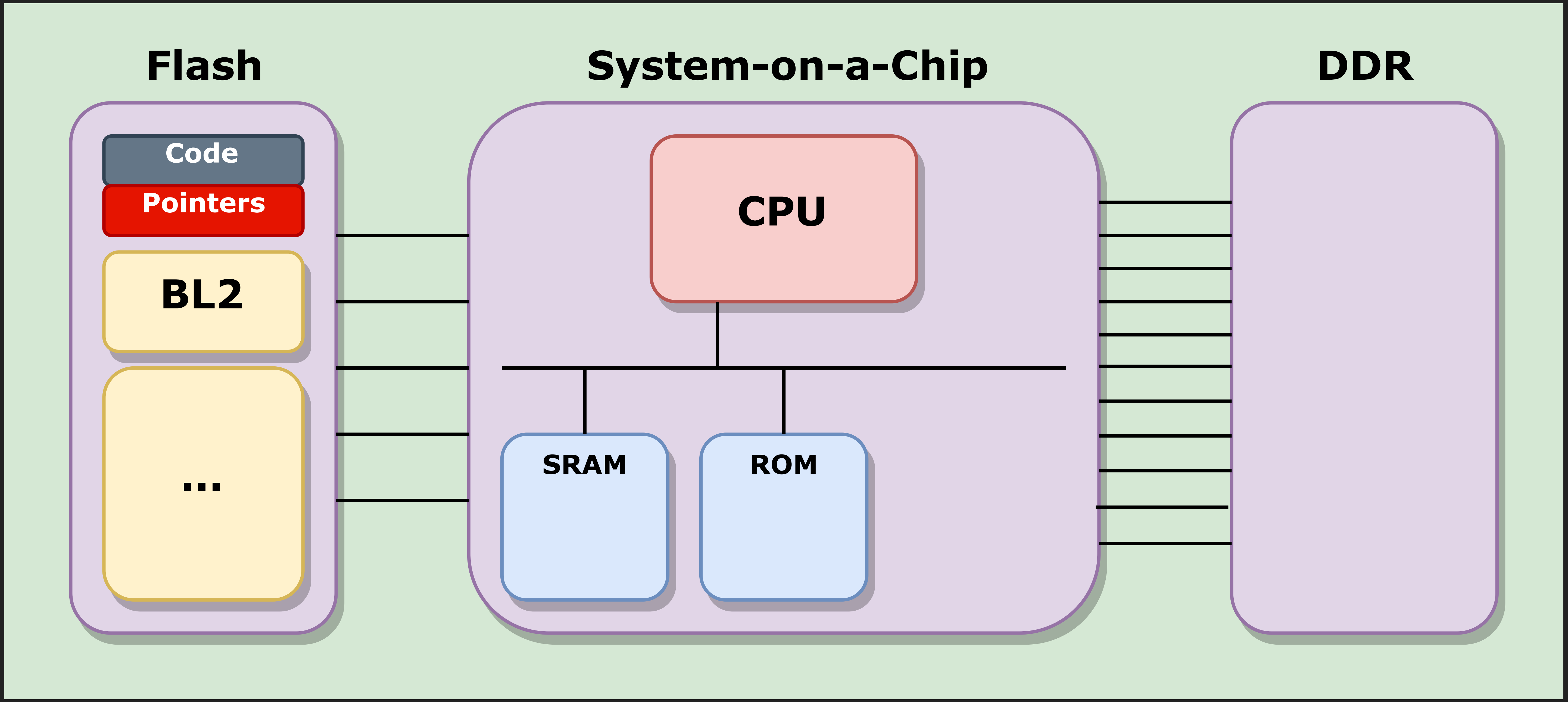
- Faults can modify instructions
- Destination register could be changed
- Fairly new application
- Great for modifying code and getting control

BYPASSING ENCRYPTED SECURE BOOT 1/4



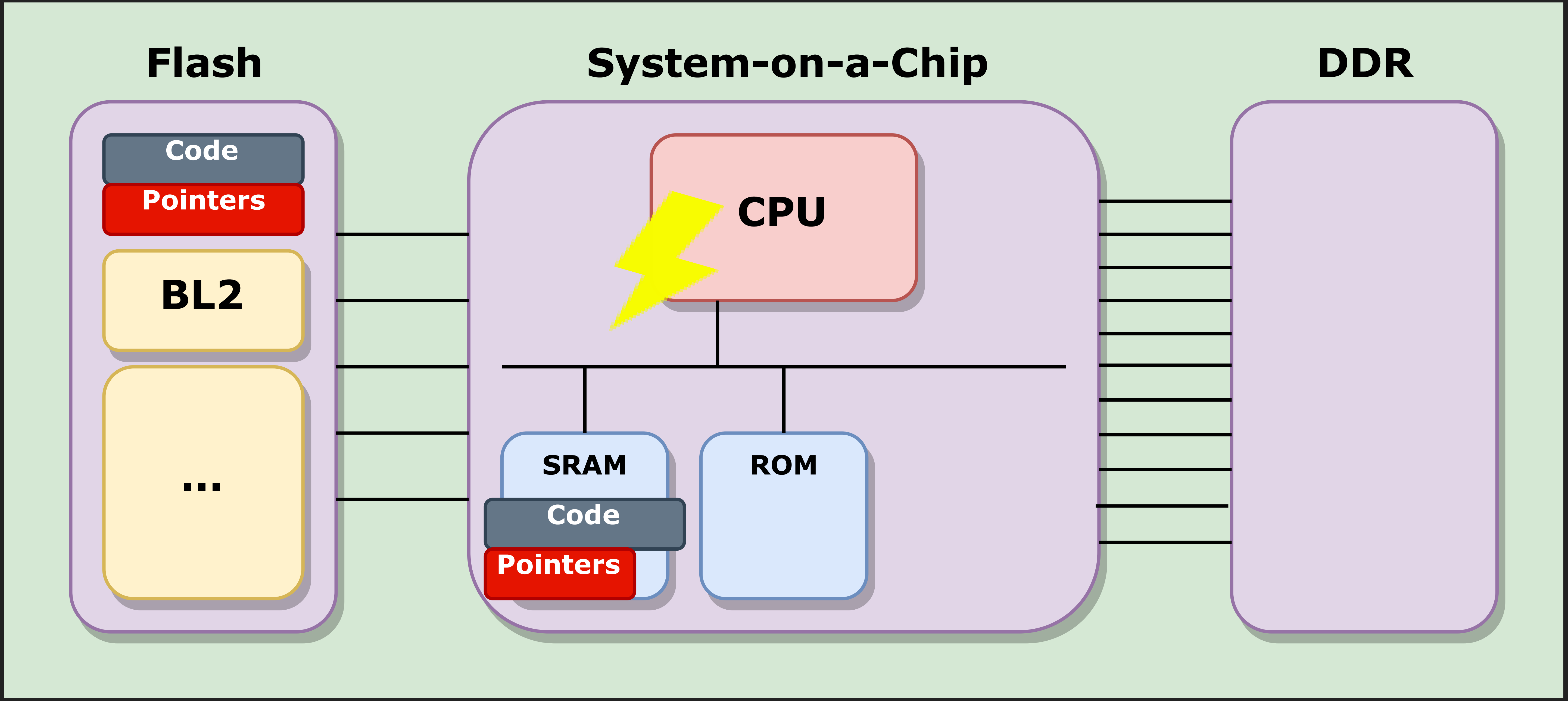
Device is turned off.

BYPASSING ENCRYPTED SECURE BOOT 2/4



Replace encrypted BL1 with plain text code and pointers to SRAM.

BYPASSING ENCRYPTED SECURE BOOT 3/4



Glitch is injected after code copy and while pointers are being copied.

BYPASSING ENCRYPTED SECURE BOOT 4/4

```
memcpy(I_SRAM, I_FLASH, I_SIZE);       // Glitch here!
decrypt(SYM_KEY, I_SRAM, I_SIZE);      // Before decryption
memcpy(S_SRAM, S_FLASH, S_SIZE);       // and

if(SHA256(I_SRAM, I_SIZE, I_HASH)) {   // before
    while(1);
}

if(verify(PUB_KEY, S_SRAM, I_HASH)) {  // verification!
    while(1);
}

jump();                                // CPU will never reach here
```

Glitch during pointers copy to assign a pointer to the program counter (PC).

RESULTING CODE EXECUTION

```
memcpy(I_SRAM, I_FLASH, I_SIZE);       // Glitch here!
.
.
.
.
.
.
.
.
.
.
.
((void *)())(pointer)();
```

Control flow is hijacked. The decryption and verification of the image is bypassed!

CONCRETELY SAID...

WE TURN

ENCRYPTED SECURE BOOT

INTO

PLAINTEXT UNPROTECTED BOOT

USING

A SINGLE GLITCH AND NO KEY!

PWN3D!

- Timing no so relevant
- Full PC control
- Bypass any SW FI countermeasure

FAULT INJECTION DEMO

ON ENCRYPTED SECURE BOOT!

Important:

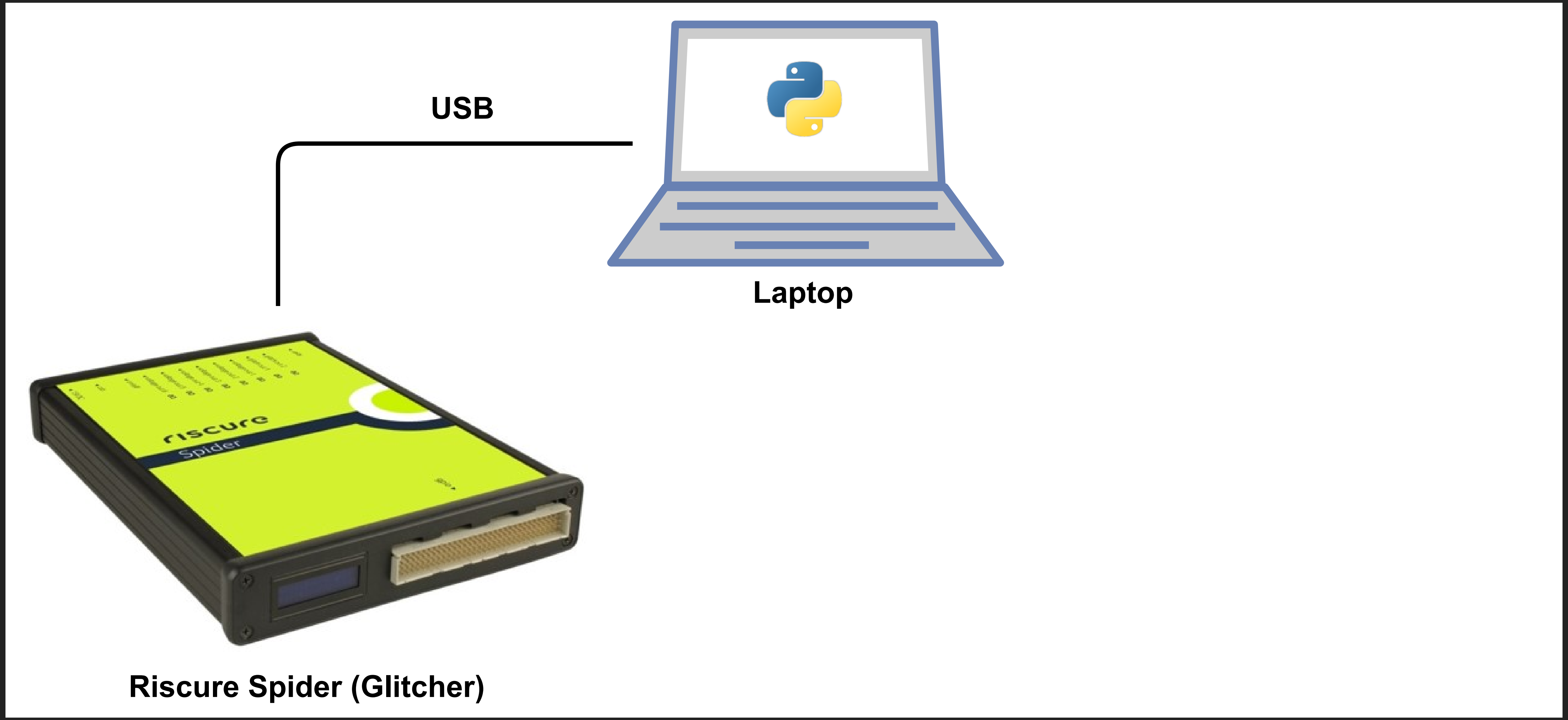
We are attacking a demo implementation!

FAULT INJECTION SETUP

Riscure Spider (Glitcher)

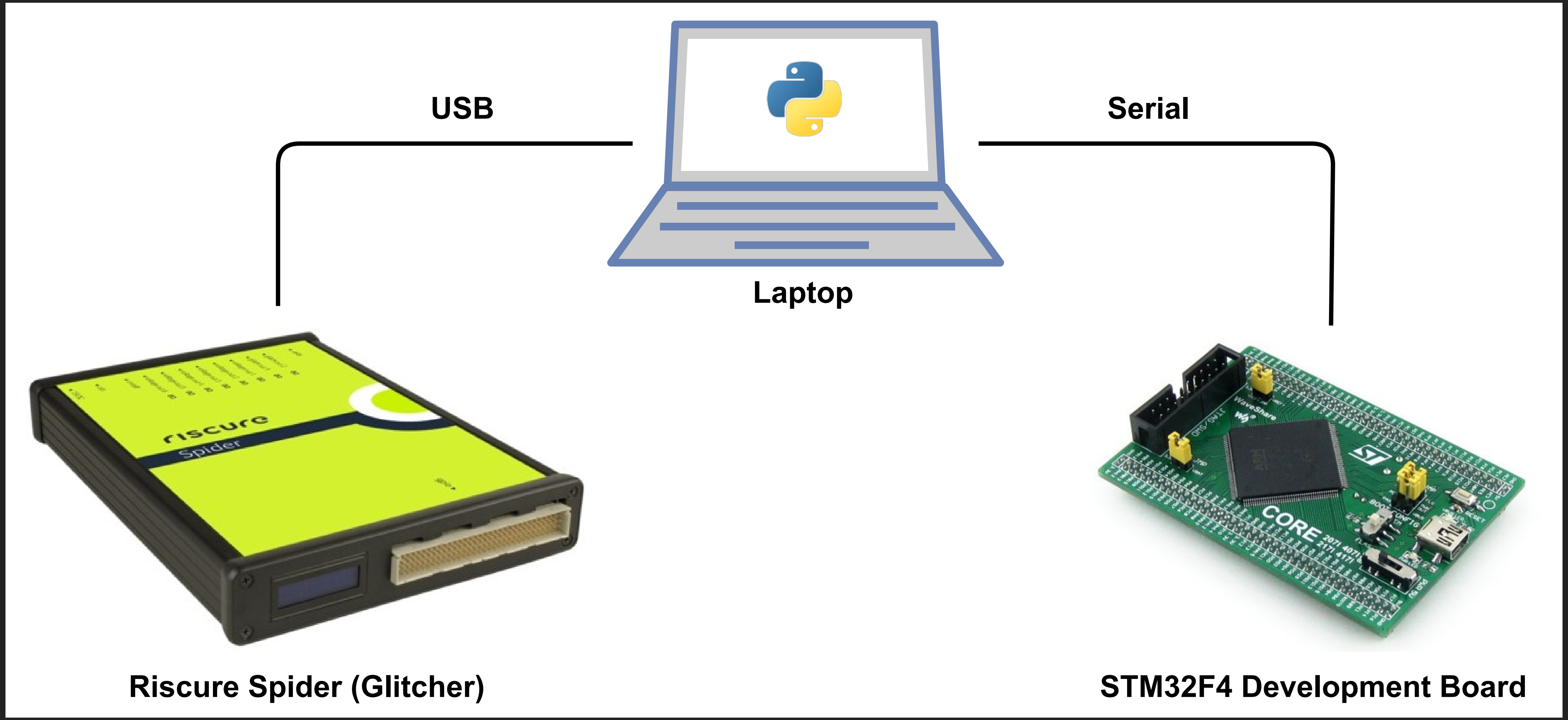
You can use NewAE's ChipWhisperer too!

FAULT INJECTION SETUP



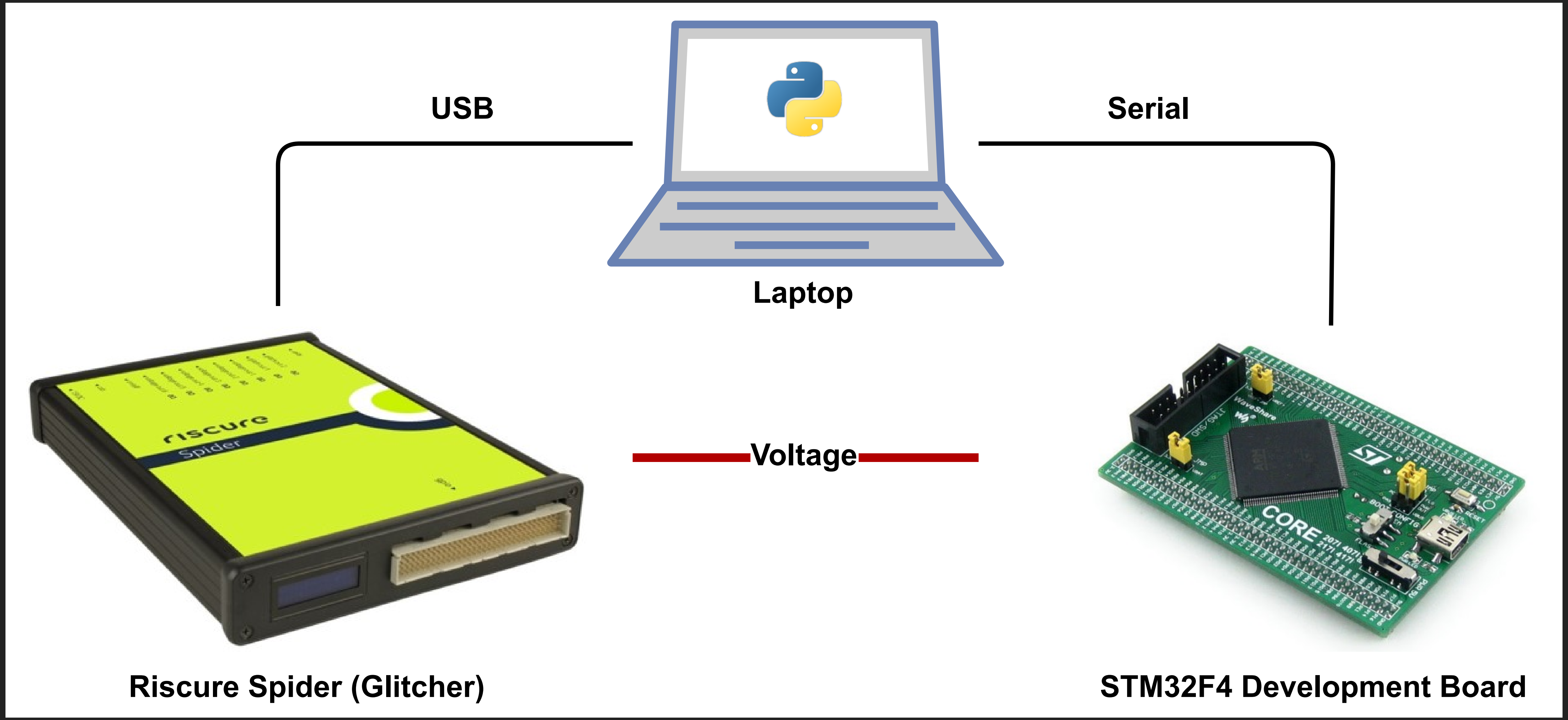
You can use NewAE's ChipWhisperer too!

FAULT INJECTION SETUP



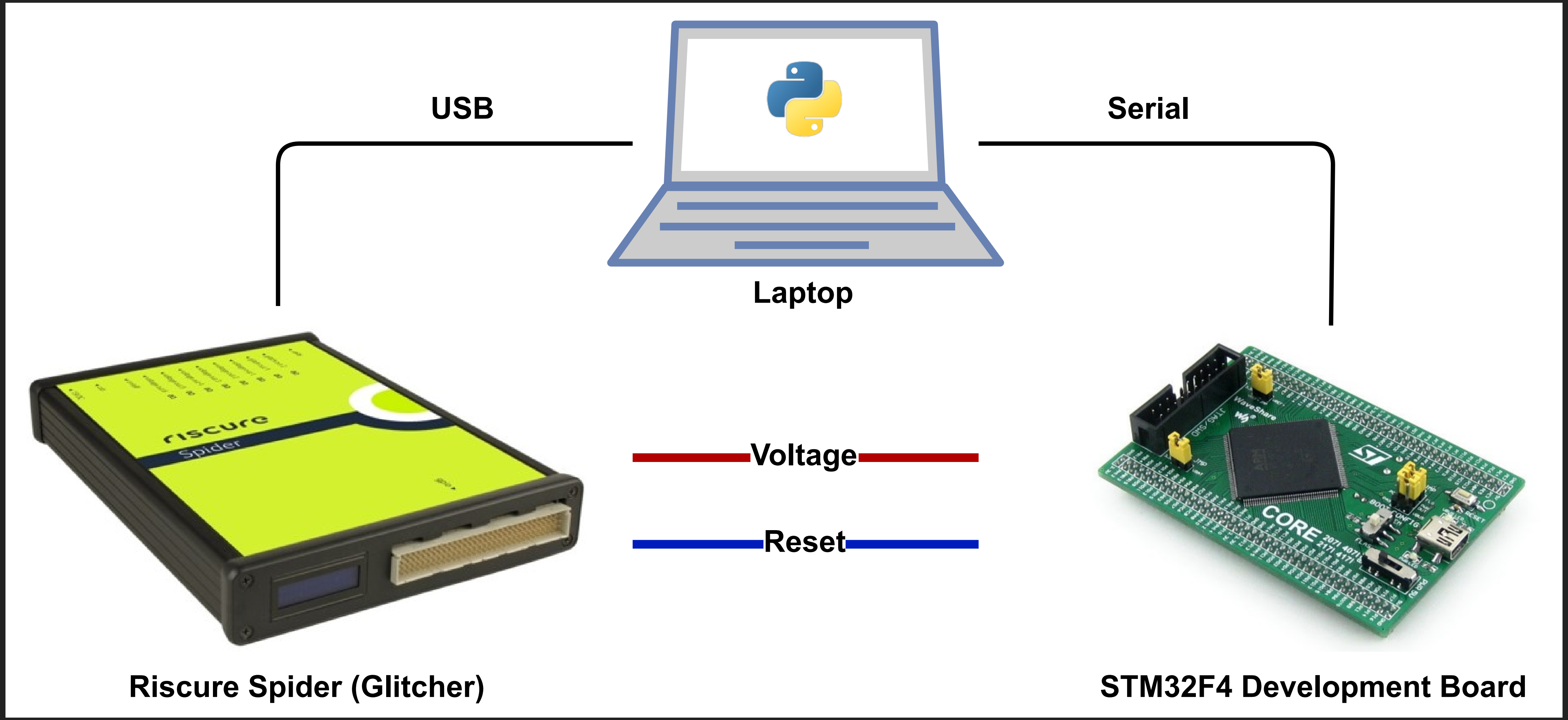
You can use NewAE's ChipWhisperer too!

FAULT INJECTION SETUP



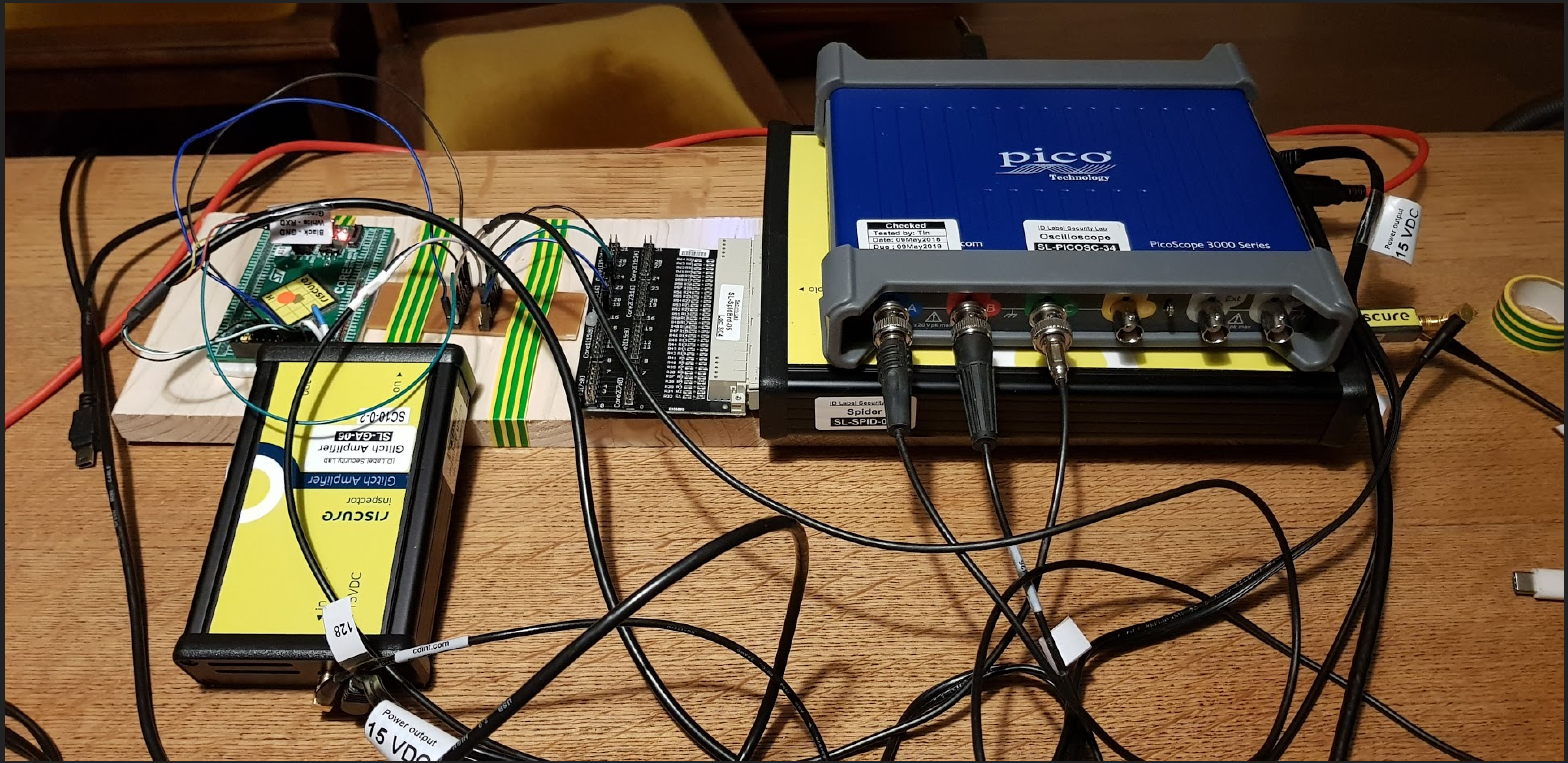
You can use NewAE's ChipWhisperer too!

FAULT INJECTION SETUP



You can use NewAE's ChipWhisperer too!

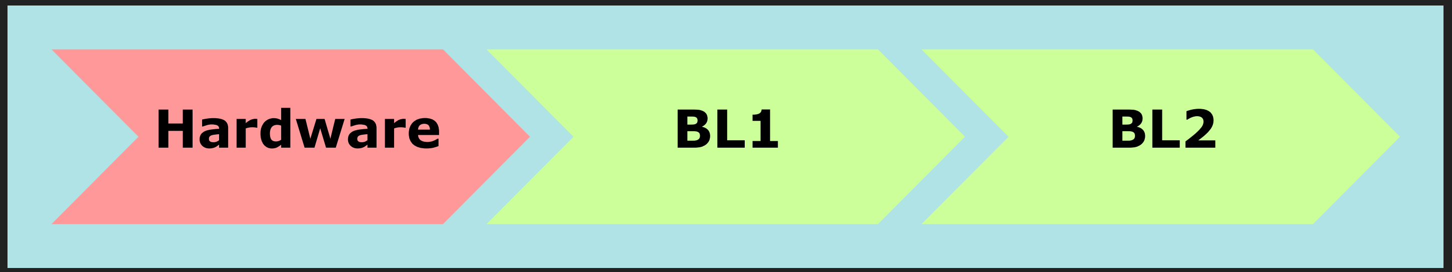
REAL WORLD FI SETUP

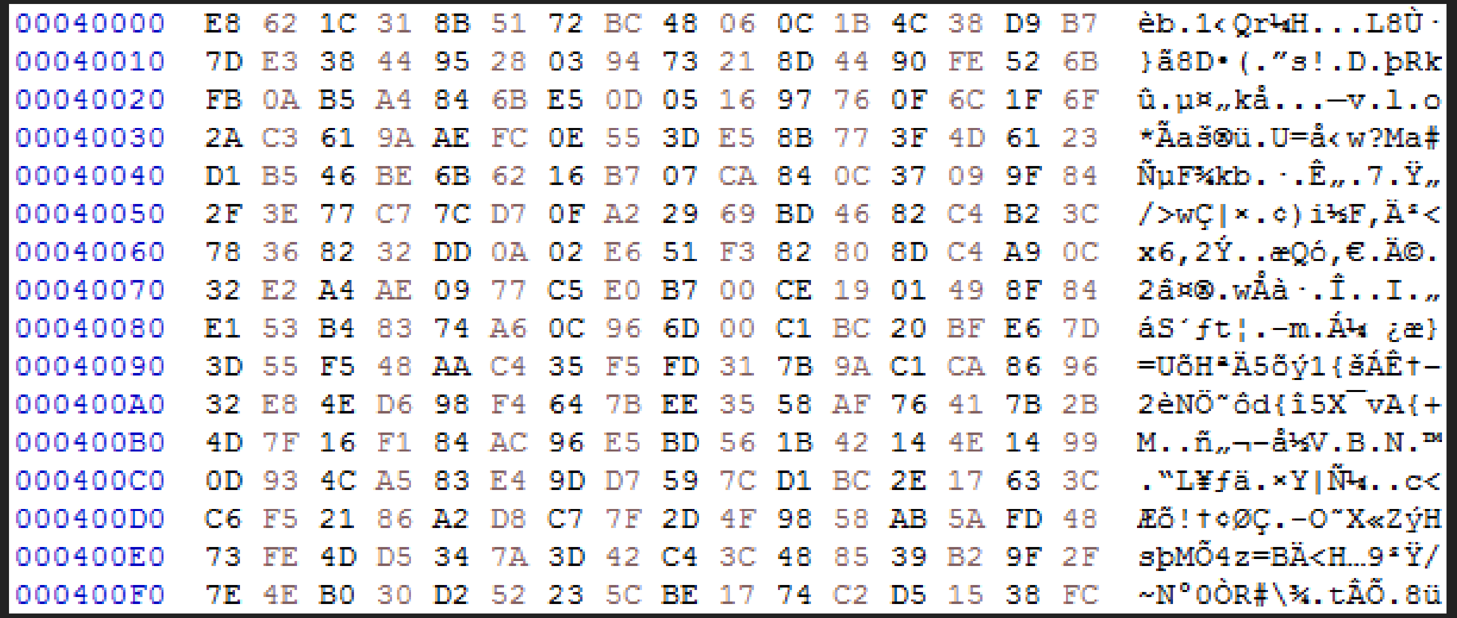


Even for simple setups there are cables everywhere...

FLASH IMAGE MODIFICATION

VALID IMAGE

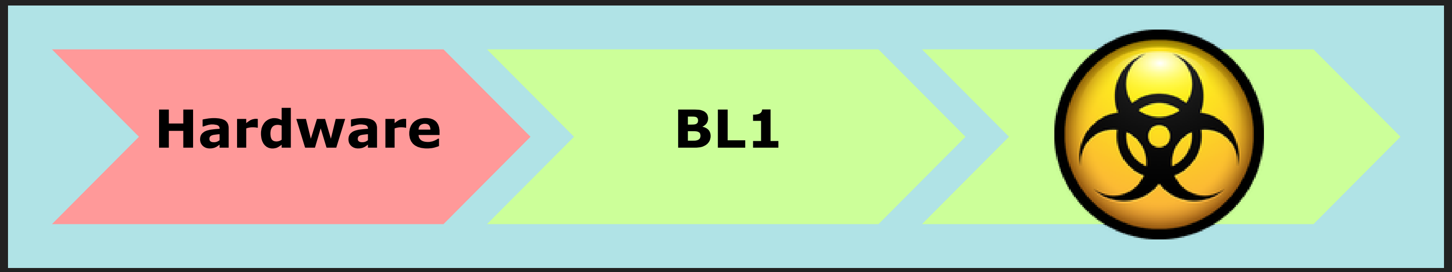


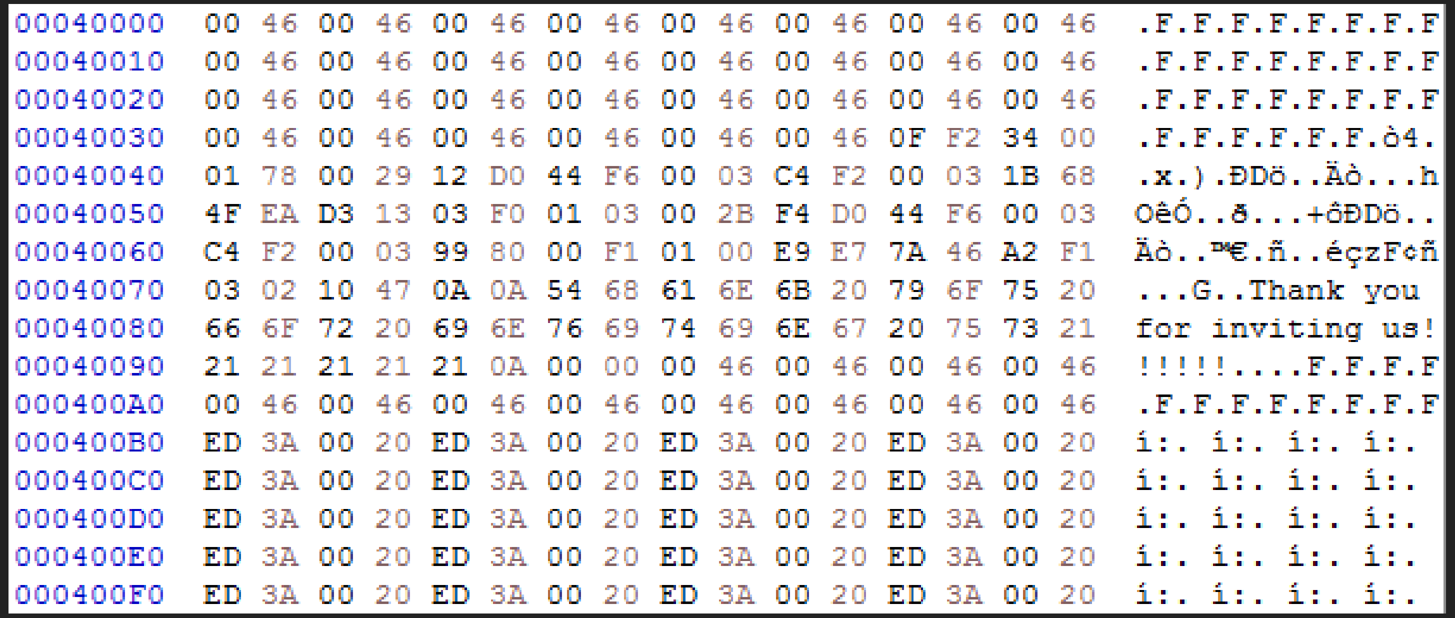


```
00040000  E8 62 1C 31 8B 51 72 BC 48 06 0C 1B 4C 38 D9 B7  èb.1‹Qr¼H...L8Ù·
00040010  7D E3 38 44 95 28 03 94 73 21 8D 44 90 FE 52 6B  }ã8D•(.”s!.D.þRk
00040020  FB 0A B5 A4 84 6B E5 0D 05 16 97 76 0F 6C 1F 6F  û.µ¤„kå...—v.l.o
00040030  2A C3 61 9A AE FC 0E 55 3D E5 8B 77 3F 4D 61 23  *Ãaš®ü.U=å‹w?Ma#
00040040  D1 B5 46 BE 6B 62 16 B7 07 CA 84 0C 37 09 9F 84  ÑµF¾kb.·.Ê„.7.Ÿ„
00040050  2F 3E 77 C7 7C D7 0F A2 29 69 BD 46 82 C4 B2 3C  />wÇ|×.¢)i½F‚Ä²<
00040060  78 36 82 32 DD 0A 02 E6 51 F3 82 80 8D C4 A9 0C  x6‚2Ý..æQó‚€.Ä©.
00040070  32 E2 A4 AE 09 77 C5 E0 B7 00 CE 19 01 49 8F 84  2â¤®.wÅà·.Î..I.„
00040080  E1 53 B4 83 74 A6 0C 96 6D 00 C1 BC 20 BF E6 7D  áS´ƒt¦.–m.Á¼ ¿æ}
00040090  3D 55 F5 48 AA C4 35 F5 FD 31 7B 9A C1 CA 86 96  =UõHªÄ5õý1{šÁÊ†–
000400A0  32 E8 4E D6 98 F4 64 7B EE 35 58 AF 76 41 7B 2B  2èNÖ˜ôd{î5X¯vA{+
000400B0  4D 7F 16 F1 84 AC 96 E5 BD 56 1B 42 14 4E 14 99  M..ñ„¬–å½V.B.N.™
000400C0  0D 93 4C A5 83 E4 9D D7 59 7C D1 BC 2E 17 63 3C  .“L¥ƒä.×Y|Ñ¼..c<
000400D0  C6 F5 21 86 A2 D8 C7 7F 2D 4F 98 58 AB 5A FD 48  Æõ!†¢ØÇ.-O˜X«ZýH
000400E0  73 FE 4D D5 34 7A 3D 42 C4 3C 48 85 39 B2 9F 2F  sþMÕ4z=BÄ<H…9²Ÿ/
000400F0  7E 4E B0 30 D2 52 23 5C BE 17 74 C2 D5 15 38 FC  ~N°0ÒR#\¾.tÂÕ.8ü
```

BL1 loads, decrypts and authenticates BL2 successfully

MALICIOUS IMAGE



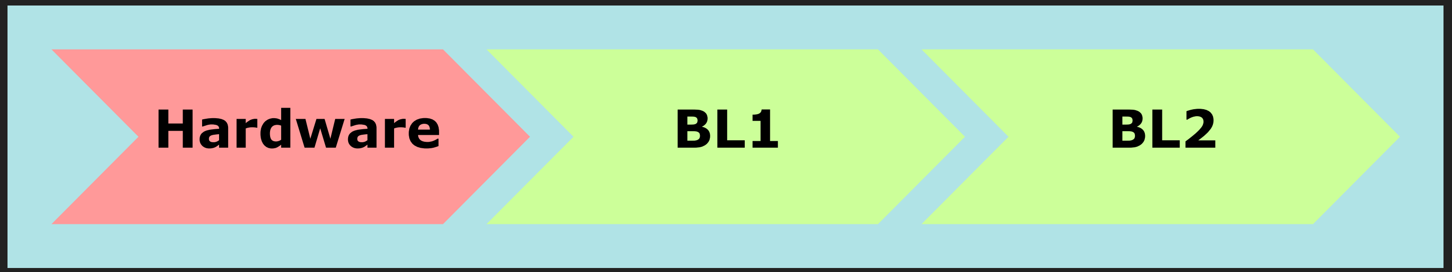


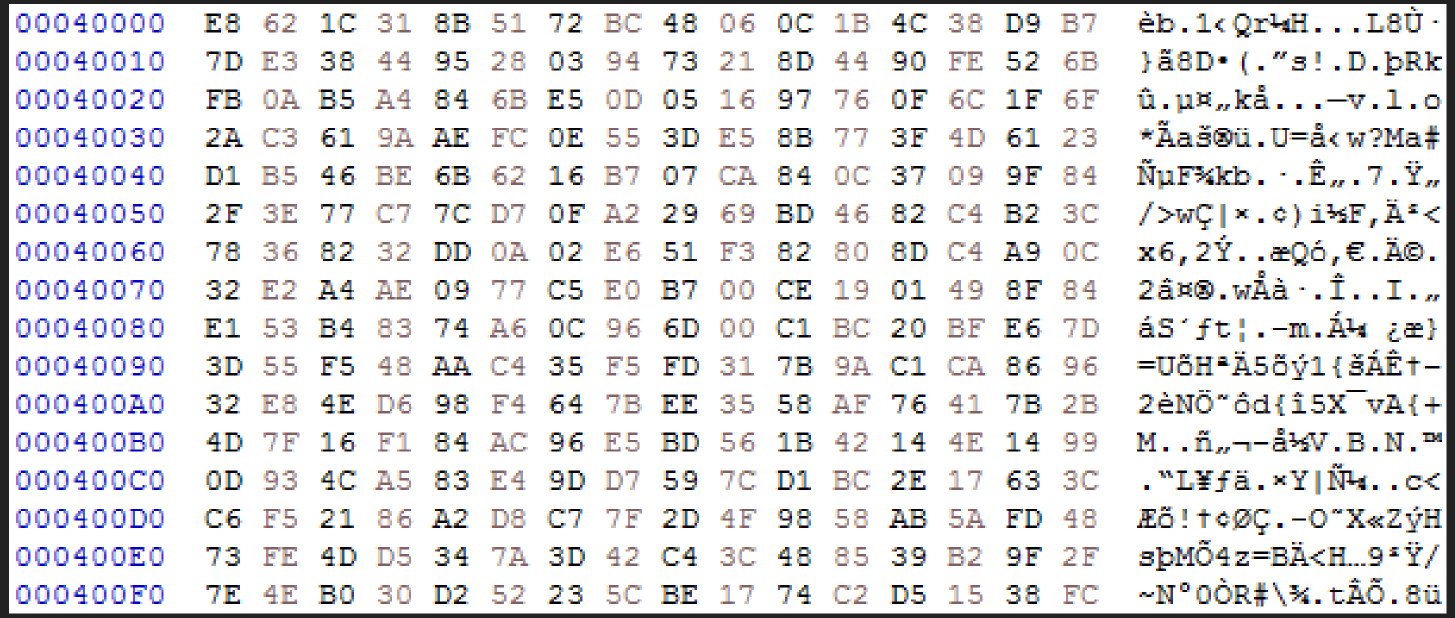
```
00040000  00 46 00 46 00 46 00 46 00 46 00 46 00 46 00 46  .F.F.F.F.F.F.F.F
00040010  00 46 00 46 00 46 00 46 00 46 00 46 00 46 00 46  .F.F.F.F.F.F.F.F
00040020  00 46 00 46 00 46 00 46 00 46 00 46 00 46 00 46  .F.F.F.F.F.F.F.F
00040030  00 46 00 46 00 46 00 46 00 46 00 46 0F F2 34 00  .F.F.F.F.F.F.ò4.
00040040  01 78 00 29 12 D0 44 F6 00 03 C4 F2 00 03 1B 68  .x.).ÐDö..Äò...h
00040050  4F EA D3 13 03 F0 01 03 00 2B F4 D0 44 F6 00 03  OêÓ..ð...+ôÐDö..
00040060  C4 F2 00 03 99 80 00 F1 01 00 E9 E7 7A 46 A2 F1  Äò..™€.ñ..éçzF¢ñ
00040070  03 02 10 47 0A 0A 54 68 61 6E 6B 20 79 6F 75 20  ...G..Thank you 
00040080  66 6F 72 20 69 6E 76 69 74 69 6E 67 20 75 73 21  for inviting us!
00040090  21 21 21 21 21 0A 00 00 00 46 00 46 00 46 00 46  !!!!!....F.F.F.F
000400A0  00 46 00 46 00 46 00 46 00 46 00 46 00 46 00 46  .F.F.F.F.F.F.F.F
000400B0  ED 3A 00 20 ED 3A 00 20 ED 3A 00 20 ED 3A 00 20  í:. í:. í:. í:. 
000400C0  ED 3A 00 20 ED 3A 00 20 ED 3A 00 20 ED 3A 00 20  í:. í:. í:. í:. 
000400D0  ED 3A 00 20 ED 3A 00 20 ED 3A 00 20 ED 3A 00 20  í:. í:. í:. í:. 
000400E0  ED 3A 00 20 ED 3A 00 20 ED 3A 00 20 ED 3A 00 20  í:. í:. í:. í:. 
000400F0  ED 3A 00 20 ED 3A 00 20 ED 3A 00 20 ED 3A 00 20  í:. í:. í:. í:. 
```

BL1 loads, decrypts but fails to authenticate BL2

FLASH IMAGE MODIFICATION

VALID IMAGE

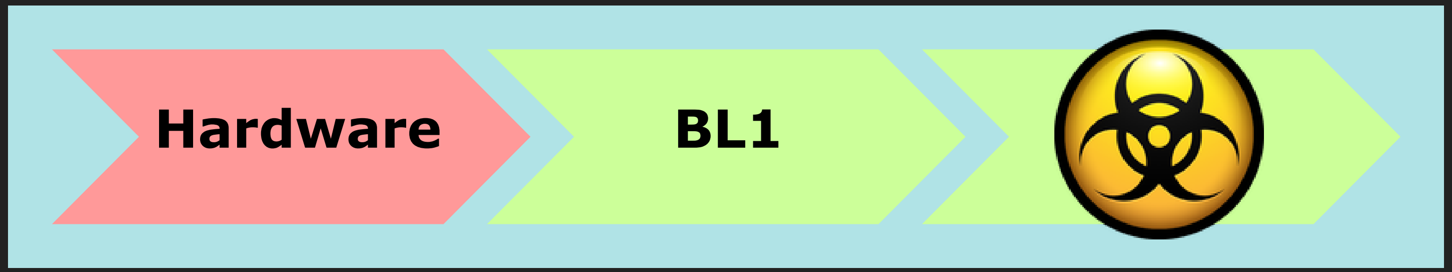


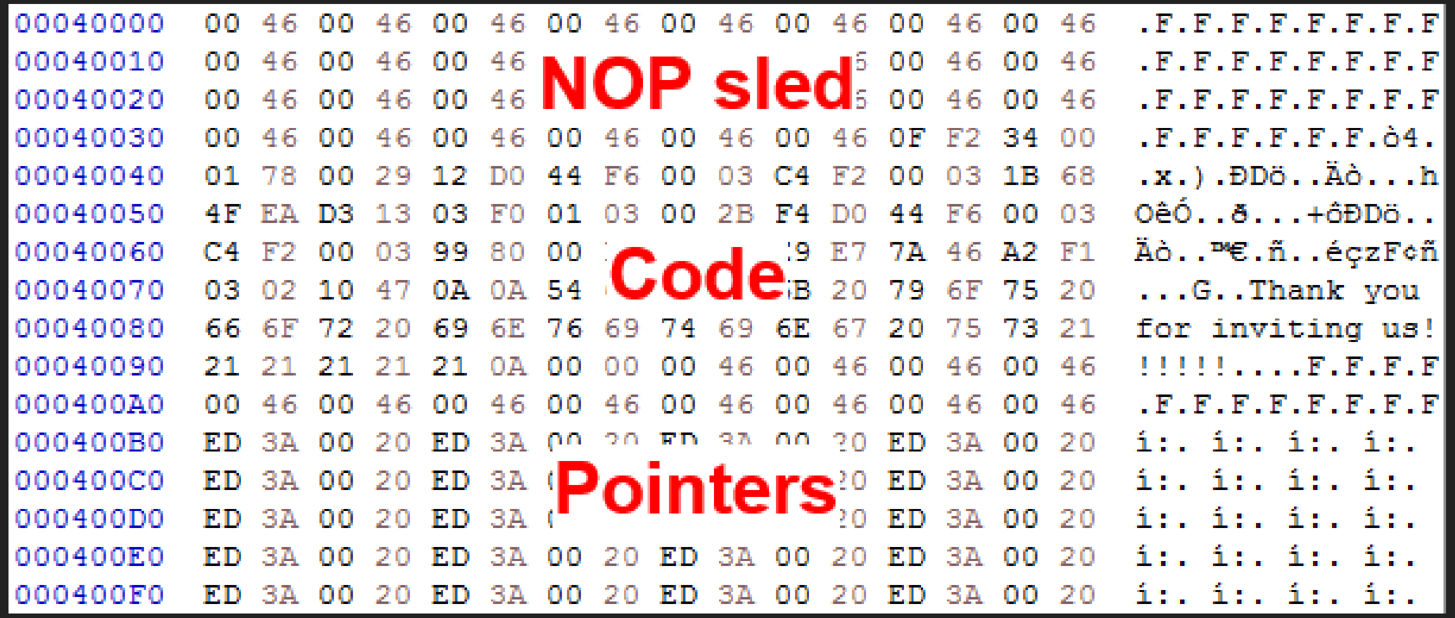


```
00040000  E8 62 1C 31 8B 51 72 BC 48 06 0C 1B 4C 38 D9 B7  èb.1‹Qr¼H...L8Ù·
00040010  7D E3 38 44 95 28 03 94 73 21 8D 44 90 FE 52 6B  }ã8D•(.”s!.D.þRk
00040020  FB 0A B5 A4 84 6B E5 0D 05 16 97 76 0F 6C 1F 6F  û.µ¤„kå...—v.l.o
00040030  2A C3 61 9A AE FC 0E 55 3D E5 8B 77 3F 4D 61 23  *Ãaš®ü.U=å‹w?Ma#
00040040  D1 B5 46 BE 6B 62 16 B7 07 CA 84 0C 37 09 9F 84  ÑµF¾kb.·.Ê„.7.Ÿ„
00040050  2F 3E 77 C7 7C D7 0F A2 29 69 BD 46 82 C4 B2 3C  />wÇ|×.¢)i½F‚Ä²<
00040060  78 36 82 32 DD 0A 02 E6 51 F3 82 80 8D C4 A9 0C  x6‚2Ý..æQó‚€.Ä©.
00040070  32 E2 A4 AE 09 77 C5 E0 B7 00 CE 19 01 49 8F 84  2â¤®.wÅà·.Î..I.„
00040080  E1 53 B4 83 74 A6 0C 96 6D 00 C1 BC 20 BF E6 7D  áS´ƒt¦.–m.Á¼ ¿æ}
00040090  3D 55 F5 48 AA C4 35 F5 FD 31 7B 9A C1 CA 86 96  =UõHªÄ5õý1{šÁÊ†–
000400A0  32 E8 4E D6 98 F4 64 7B EE 35 58 AF 76 41 7B 2B  2èNÖ˜ôd{î5X¯vA{+
000400B0  4D 7F 16 F1 84 AC 96 E5 BD 56 1B 42 14 4E 14 99  M..ñ„¬–å½V.B.N.™
000400C0  0D 93 4C A5 83 E4 9D D7 59 7C D1 BC 2E 17 63 3C  .“L¥ƒä.×Y|Ñ¼..c<
000400D0  C6 F5 21 86 A2 D8 C7 7F 2D 4F 98 58 AB 5A FD 48  Æõ!†¢ØÇ.-O˜X«ZýH
000400E0  73 FE 4D D5 34 7A 3D 42 C4 3C 48 85 39 B2 9F 2F  sþMÕ4z=BÄ<H…9²Ÿ/
000400F0  7E 4E B0 30 D2 52 23 5C BE 17 74 C2 D5 15 38 FC  ~N°0ÒR#\¾.tÂÕ.8ü
```

BL1 loads, decrypts and authenticates BL2 successfully

MALICIOUS IMAGE





```
00040000  00 46 00 46 00 46 00 46 00 46 00 46 00 46 00 46  .F.F.F.F.F.F.F.F
00040010  00 46 00 46 00 46                5 00 46 00 46  .F.F.F.F.F.F.F.F
00040020  00 46 00 46 00 46                5 00 46 00 46  .F.F.F.F.F.F.F.F
00040030  00 46 00 46 00 46 00 46 00 46 00 46 0F F2 34 00  .F.F.F.F.F.F.ò4.
00040040  01 78 00 29 12 D0 44 F6 00 03 C4 F2 00 03 1B 68  .x.).ÐDö..Äò...h
00040050  4F EA D3 13 03 F0 01 03 00 2B F4 D0 44 F6 00 03  OêÓ..ð...+ôÐDö..
00040060  C4 F2 00 03 99 80 00             9 E7 7A 46 A2 F1  Äò..™€.ñ..éçzF¢ñ
00040070  03 02 10 47 0A 0A 54             B 20 79 6F 75 20  ...G..Thank you 
00040080  66 6F 72 20 69 6E 76 69 74 69 6E 67 20 75 73 21  for inviting us!
00040090  21 21 21 21 21 0A 00 00 00 46 00 46 00 46 00 46  !!!!!....F.F.F.F
000400A0  00 46 00 46 00 46 00 46 00 46 00 46 00 46 00 46  .F.F.F.F.F.F.F.F
000400B0  ED 3A 00 20 ED 3A          20 ED 3A 00 20  í:. í:. í:. í:.
000400C0  ED 3A 00 20 ED 3A               20 ED 3A 00 20  í:. í:. í:. í:.
000400D0  ED 3A 00 20 ED 3A               20 ED 3A 00 20  í:. í:. í:. í:.
000400E0  ED 3A 00 20 ED 3A 00 20 ED 3A 00 20 ED 3A 00 20  í:. í:. í:. í:.
000400F0  ED 3A 00 20 ED 3A 00 20 ED 3A 00 20 ED 3A 00 20  í:. í:. í:. í:.
```

BL1 loads, decrypts but fails to authenticate BL2

TARGET BEHAVIOR

Valid image

```
[BL1]: Successfully started.
[BL1]: Loading BL2 successful.
[BL1]: Decrypting BL2 successful.
[BL1]: Authenticating BL2 successful.
[BL1]: Jumping to BL2...
[BL2]: Successfully started.
```

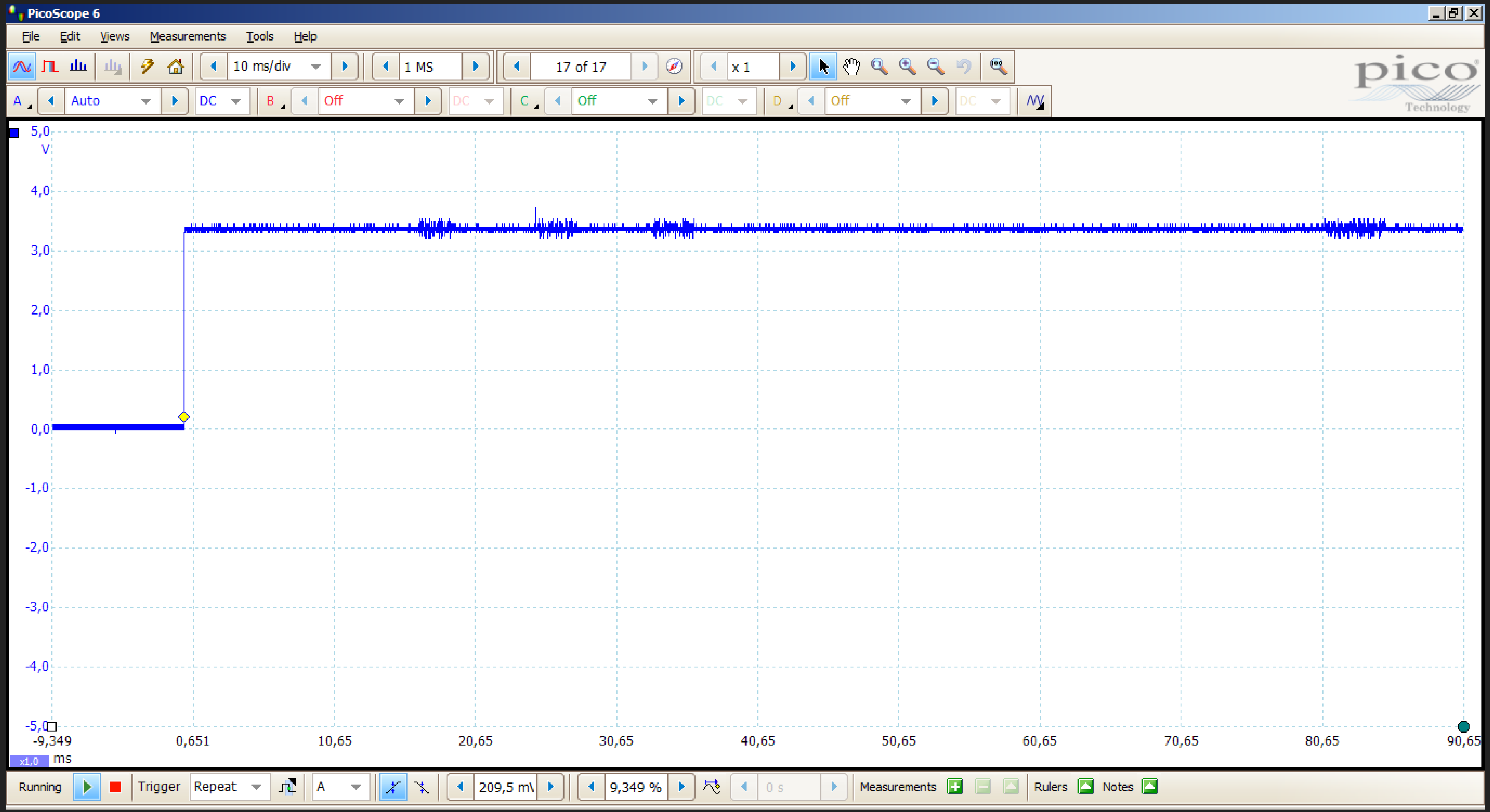
Malicious image

```
[BL1]: Successfully started.
[BL1]: Loading BL2 successful.
[BL1]: Decrypting BL2 successful.
[BL1]: Authenticating BL2 unsuccessful. Stopping!
```

Let's bypass it using fault injection!

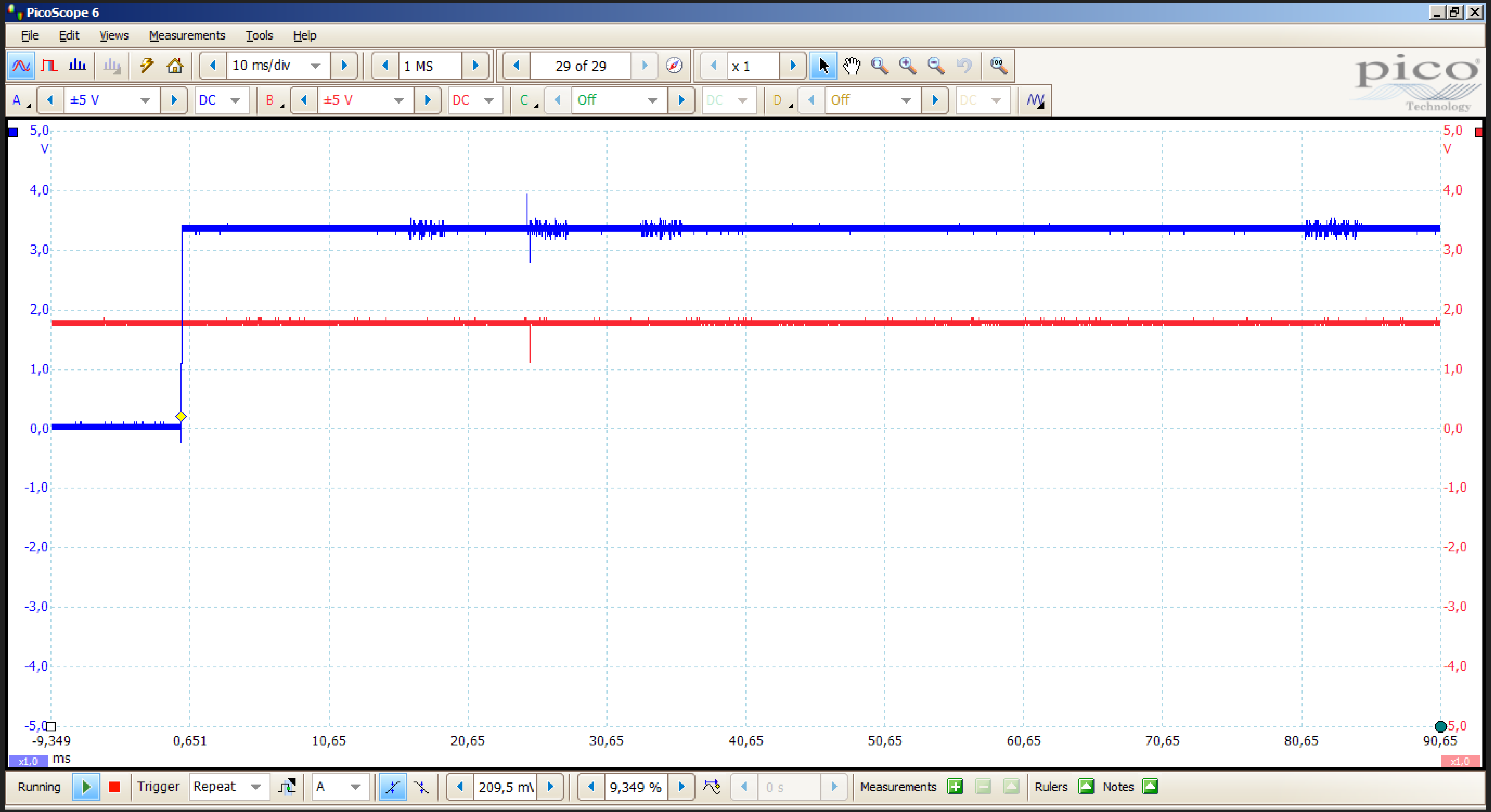
LET'S SWITCH TO THE OTHER LAPTOP

OSCILLOSCOPE 1/2



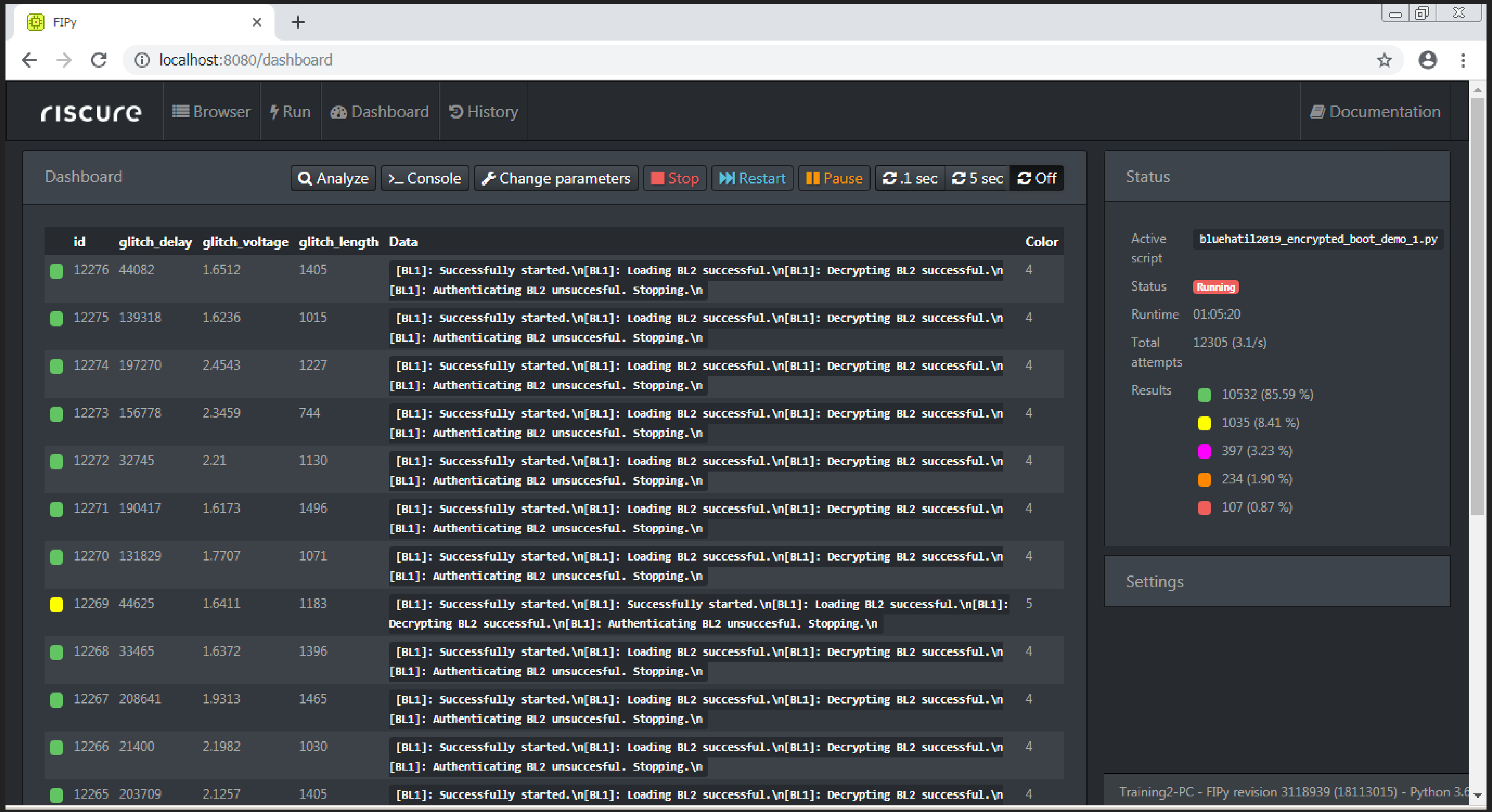
We reset the chip for each experiment.

OSCILLOSCOPE 2/2



We inject the glitch during the copy of BL2 by BL1.

FIPY 1/3



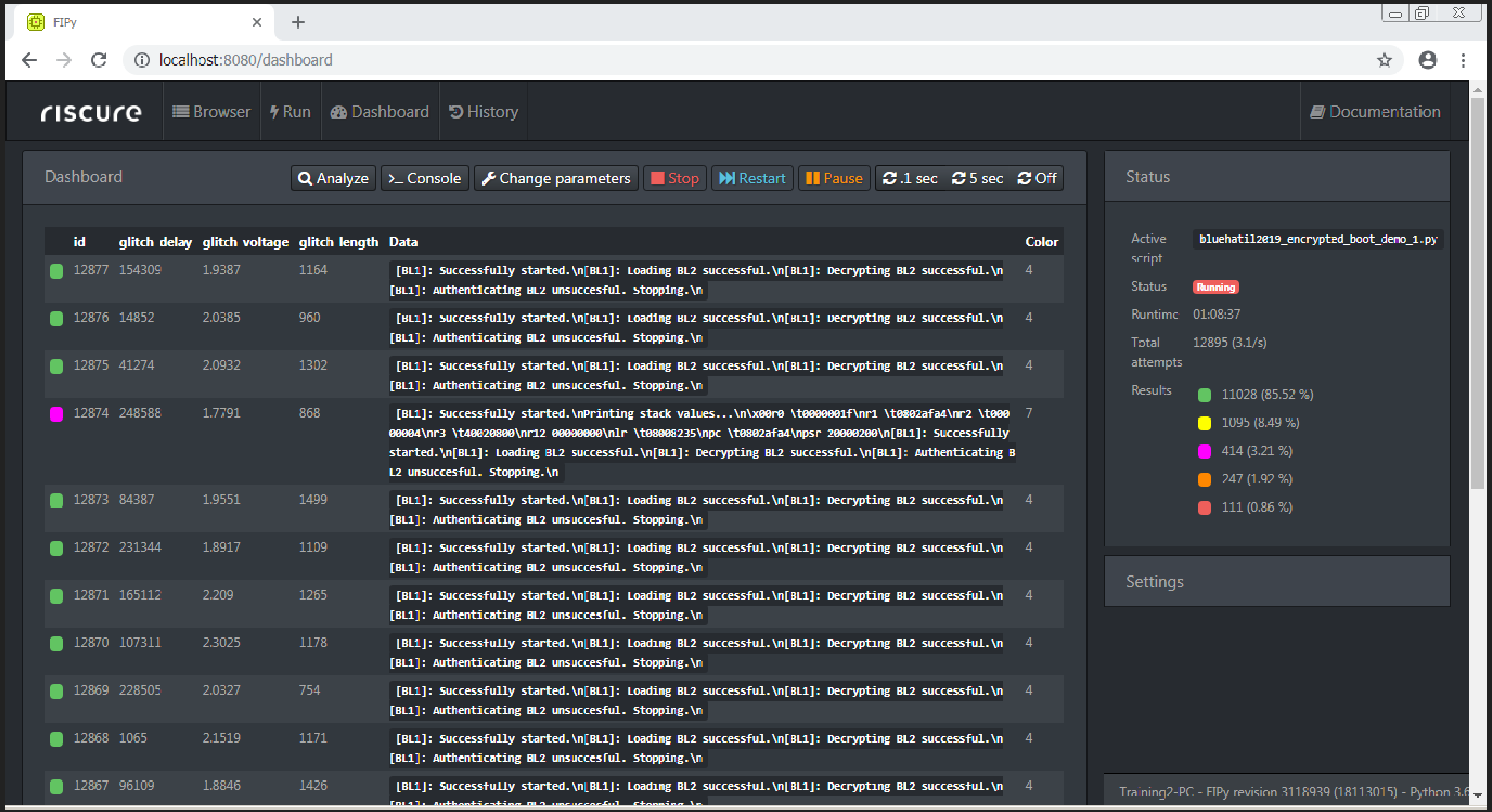
	id	glitch_delay	glitch_voltage	glitch_length	Data	Color
🟩	12276	44082	1.6512	1405	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟩	12275	139318	1.6236	1015	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟩	12274	197270	2.4543	1227	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟩	12273	156778	2.3459	744	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟩	12272	32745	2.21	1130	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟩	12271	190417	1.6173	1496	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟩	12270	131829	1.7707	1071	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟨	12269	44625	1.6411	1183	[BL1]: Successfully started.\n[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n	5
🟩	12268	33465	1.6372	1396	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟩	12267	208641	1.9313	1465	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟩	12266	21400	2.1982	1030	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n	4
🟩	12265	203709	2.1257	1405	[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n	4

Active script	bluehatil2019_encrypted_boot_demo_1.py
Status	Running
Runtime	01:05:20
Total attempts	12305 (3.1/s)
Results	🟩 10532 (85.59 %)
	🟨 1035 (8.41 %)
	🟪 397 (3.23 %)
	🟧 234 (1.90 %)
	🟥 107 (0.87 %)

Training2-PC - FIPy revision 3118939 (18113015) - Python 3.6

Experiments that had no affect on the target are colored green.

FIPY 2/3



	id	glitch_delay	glitch_voltage	glitch_length	Data	Color
🟩	12877	154309	1.9387	1164	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12876	14852	2.0385	960	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12875	41274	2.0932	1302	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟪	12874	248588	1.7791	868	`[BL1]: Successfully started.\nPrinting stack values...\n\x00r0 \t0000001f\nr1 \t0802afa4\nr2 \t00000004\nr3 \t40020800\nr12 00000000\nlr \t08008235\npc \t0802afa4\npsr 20000200\n[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	7
🟩	12873	84387	1.9551	1499	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12872	231344	1.8917	1109	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12871	165112	2.209	1265	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12870	107311	2.3025	1178	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12869	228505	2.0327	754	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12868	1065	2.1519	1171	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n[BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12867	96109	1.8846	1426	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n`	4

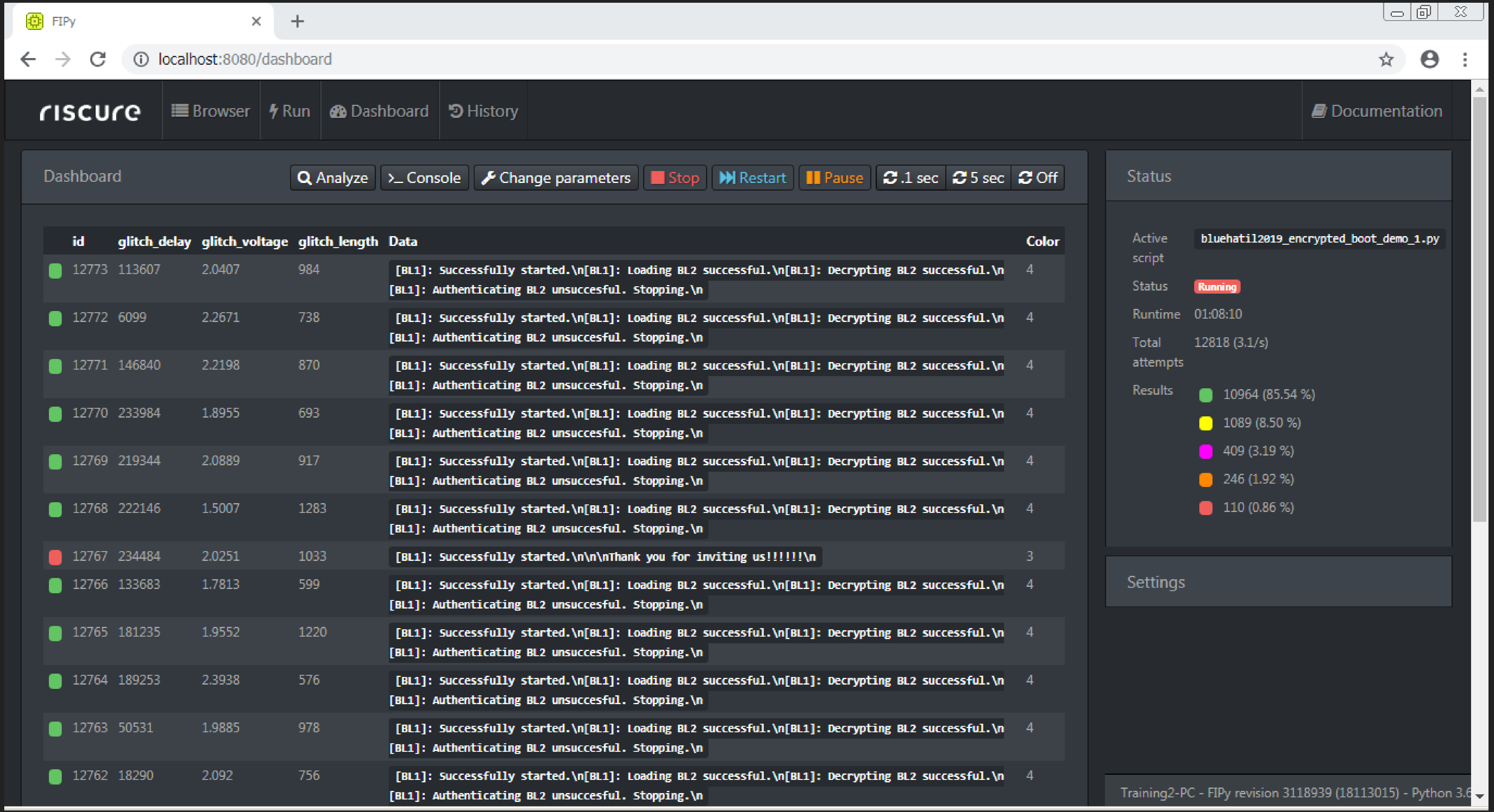
Status

Active script	`bluehatil2019_encrypted_boot_demo_1.py`
Status	Running
Runtime	01:08:37
Total attempts	12895 (3.1/s)
Results	🟩 11028 (85.52 %)
	🟨 1095 (8.49 %)
	🟪 414 (3.21 %)
	🟧 247 (1.92 %)
	🟥 111 (0.86 %)

Training2-PC - FIPy revision 3118939 (18113015) - Python 3.6

Experiments that resulted in a CPU expection are colored magenta.

FIPY 3/3



	id	glitch_delay	glitch_voltage	glitch_length	Data	Color
🟩	12773	113607	2.0407	984	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12772	6099	2.2671	738	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12771	146840	2.2198	870	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12770	233984	1.8955	693	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12769	219344	2.0889	917	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12768	222146	1.5007	1283	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟥	12767	234484	2.0251	1033	`[BL1]: Successfully started.\n\n\nThank you for inviting us!!!!!!\n`	3
🟩	12766	133683	1.7813	599	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12765	181235	1.9552	1220	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12764	189253	2.3938	576	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12763	50531	1.9885	978	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4
🟩	12762	18290	2.092	756	`[BL1]: Successfully started.\n[BL1]: Loading BL2 successful.\n[BL1]: Decrypting BL2 successful.\n [BL1]: Authenticating BL2 unsuccesful. Stopping.\n`	4

Status

Active script	`bluehatil2019_encrypted_boot_demo_1.py`
Status	Running
Runtime	01:08:10
Total attempts	12818 (3.1/s)
Results	🟩 10964 (85.54 %)
	🟨 1089 (8.50 %)
	🟪 409 (3.19 %)
	🟧 246 (1.92 %)
	🟥 110 (0.86 %)

Training2-PC - FIPy revision 3118939 (18113015) - Python 3.

Experiments that resulted in a successful bypass of secure boot are colored red.

WHAT NOW?

WHITE PAPER

"Notes on designing secure boot."

Coming soon!

HARDENING SECURE BOOT

Keep it simple

Minimize attacker choices

Authenticate everything

No weak crypto

Make software exploitation hard

Drop privileges

Make fault injection hard

Support anti-rollback

WHAT ELSE

SECURE SYSTEM/SW DEVELOPMENT LIFE CYCLE (SECURE SDLC)

- Continuous software review & testing
- Hardware security review & testing

KEY TAKEAWAYS

1. Secure boot is often not optimally hardened
2. Attack surface of secure boot is larger than expected
3. New perspectives on attacking secure boot

THANK YOU. QUESTIONS?

riscure

Niek Timmers

niek@riscure.com

@tieknimmers

AVAILABLE

Albert Spruyt

albert.spruyt@gmail.com

PULSE

Cristofaro Mune

c.mune@pulse-sec.com

@pulsoid