Hand Me Your SECRET, MCU!

Microarchitectural Timing Attacks on Microcontrollers are Practical

Cristiano Rodrigues | Sandro Pinto, PhD

(Centro ALGORITMI / LASI, Universidade do Minho)

Hand Me Your SECRET, MCU!

Microarchitectural Timing Attacks on Microcontrollers are Practical

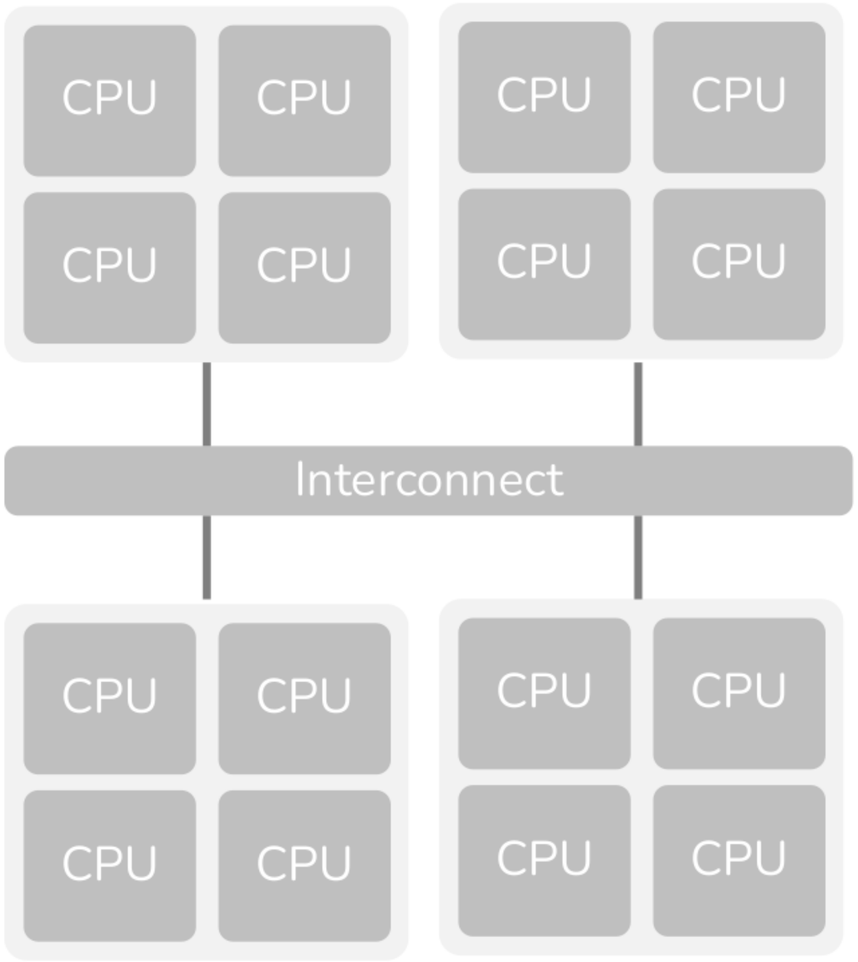
Cristiano Rodrigues | Sandro Pinto, PhD

(Centro ALGORITMI / LASI, Universidade do Minho)

CENTROALGORITMI

Microarchitectural

SIDE-CHANNEL Attacks

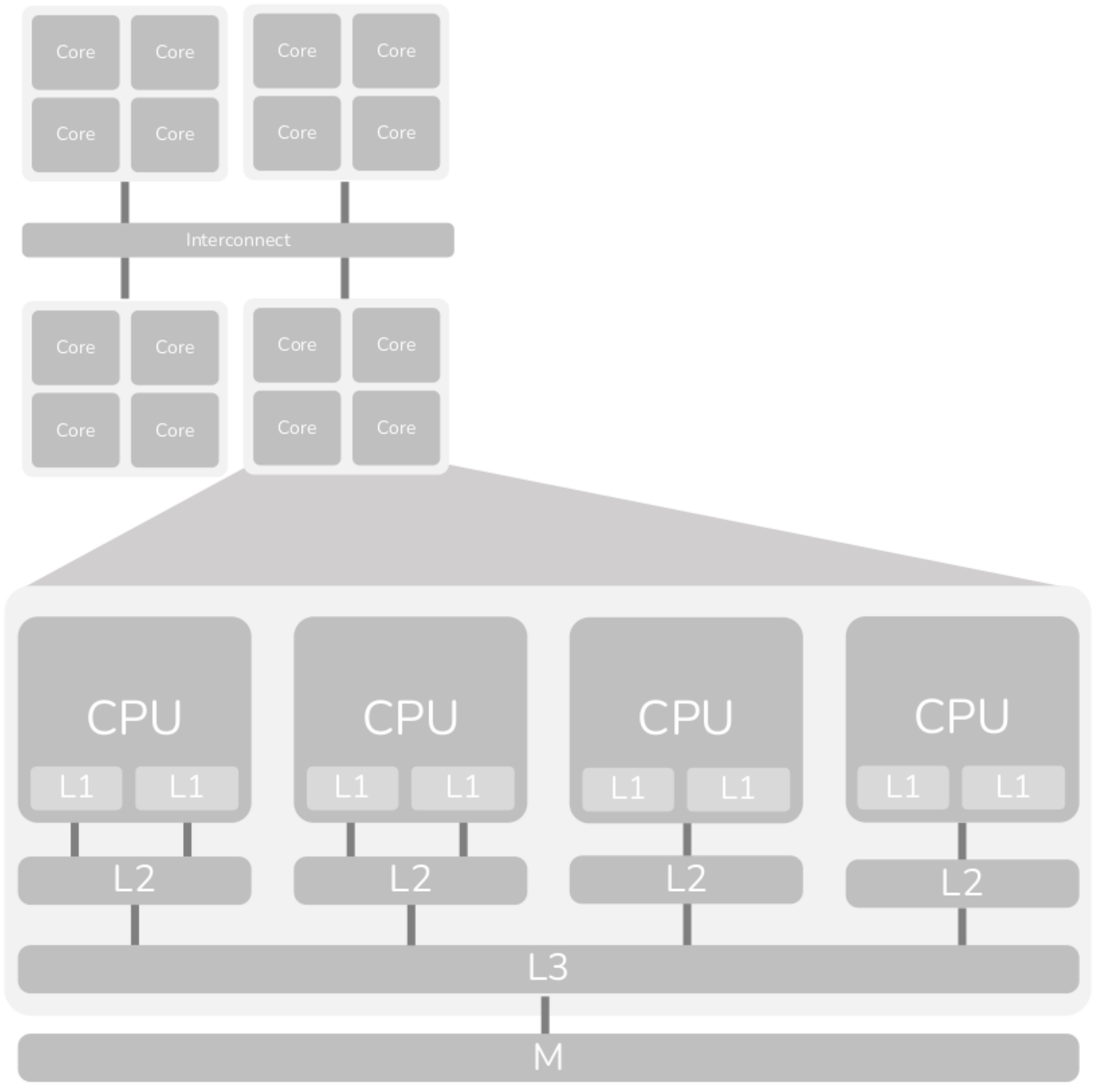


Microarchitectural

SIDE-CHANNELS

Servers, PCs, Mobile

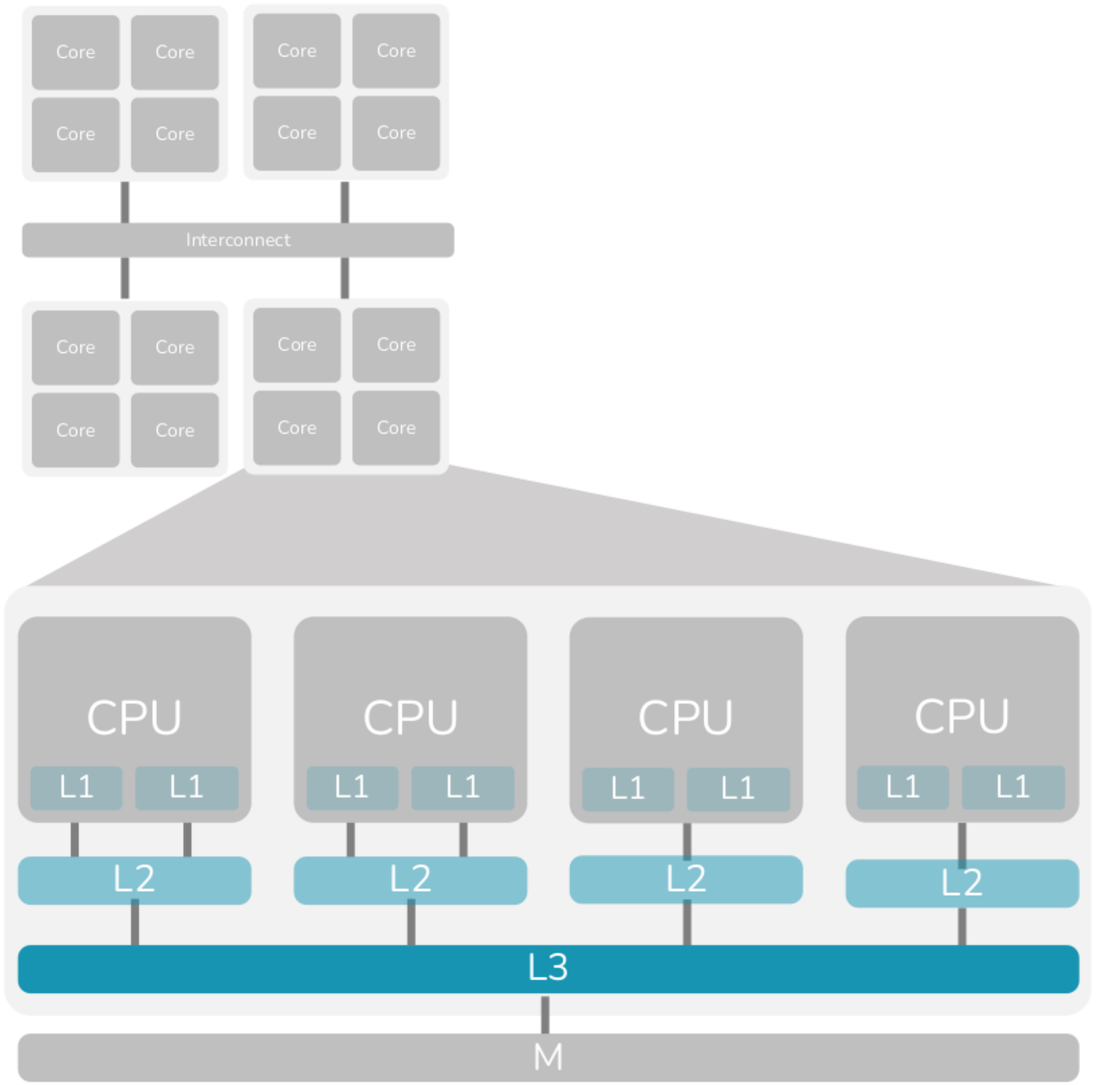
intel AMD arm



Microarchitectural SIDE-CHANNELS

Servers, PCs, Mobile

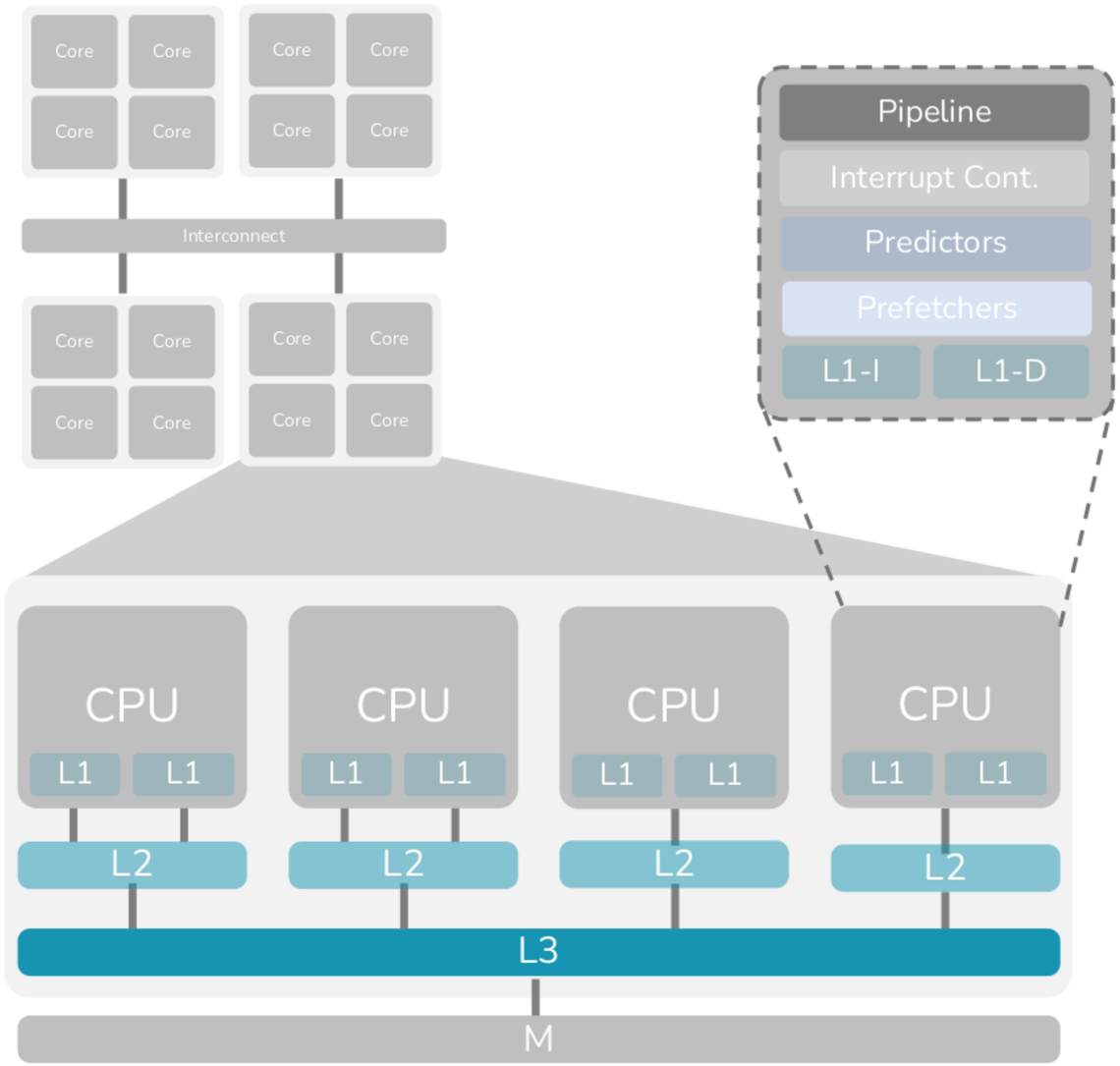
intel AMD arm



Microarchitectural SIDE-CHANNELS

Servers, PCs, Mobile

intel AMD arm

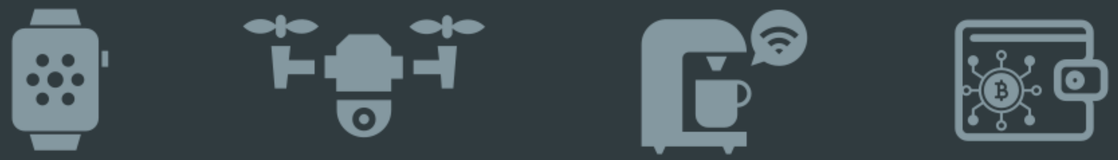


Microarchitectural SIDE-CHANNELS

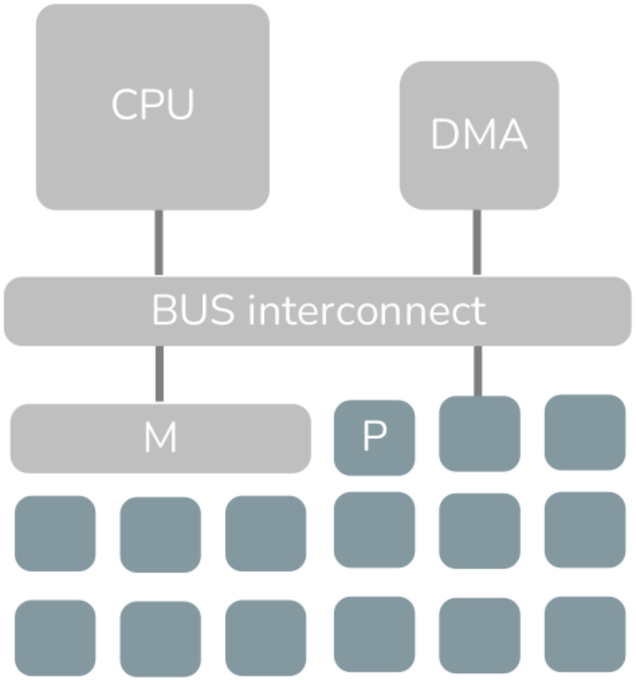
Servers, PCs, Mobile

intel AMD arm

Microcontrollers

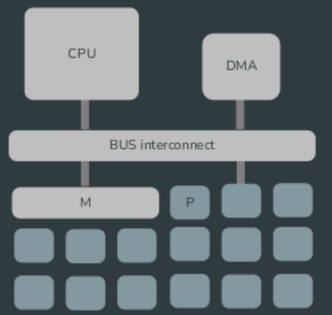


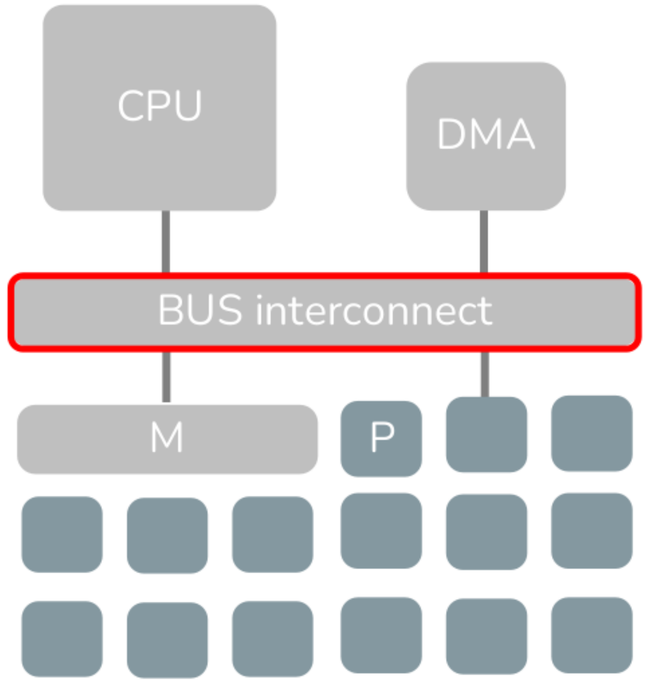
arm



Which unique microarchitectural elements on MCUs may create new channels, and how can they be used to mount effective attacks?

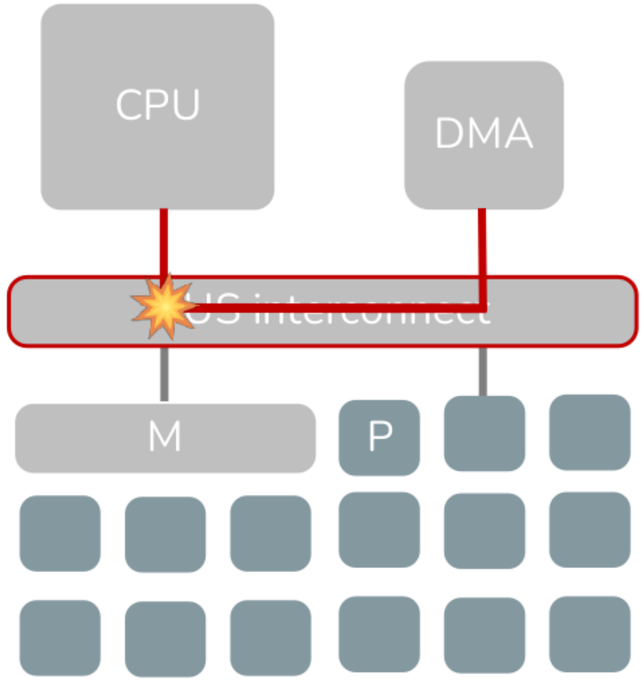
Research Question





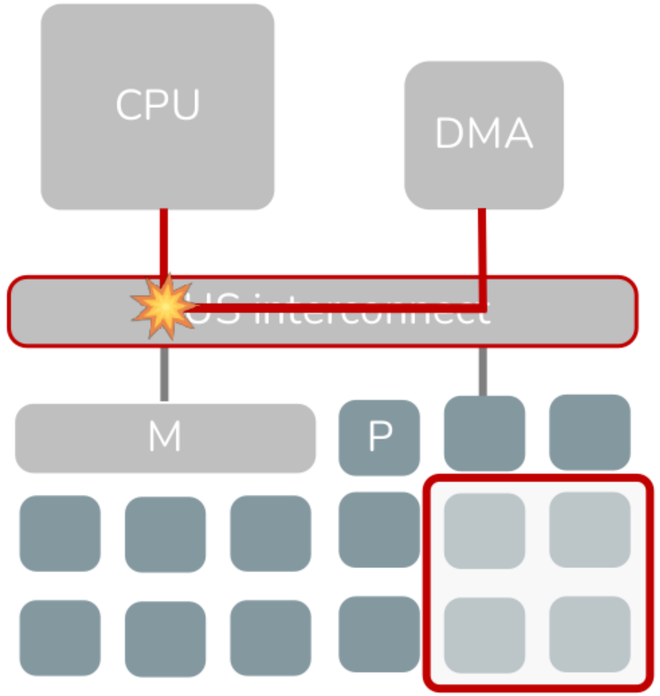
Novel Channel

BUS Interconnect



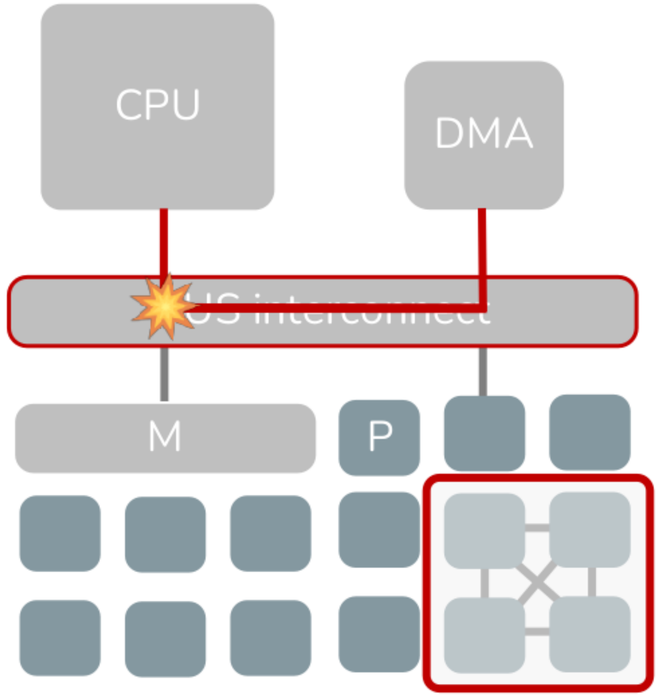
BUS Interconnect

Arbitration logic



Hardware Gadgets

Novel Concept

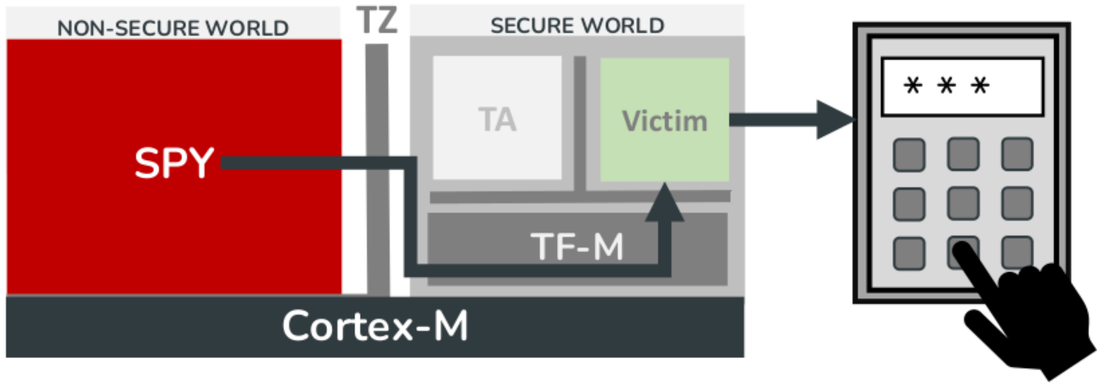


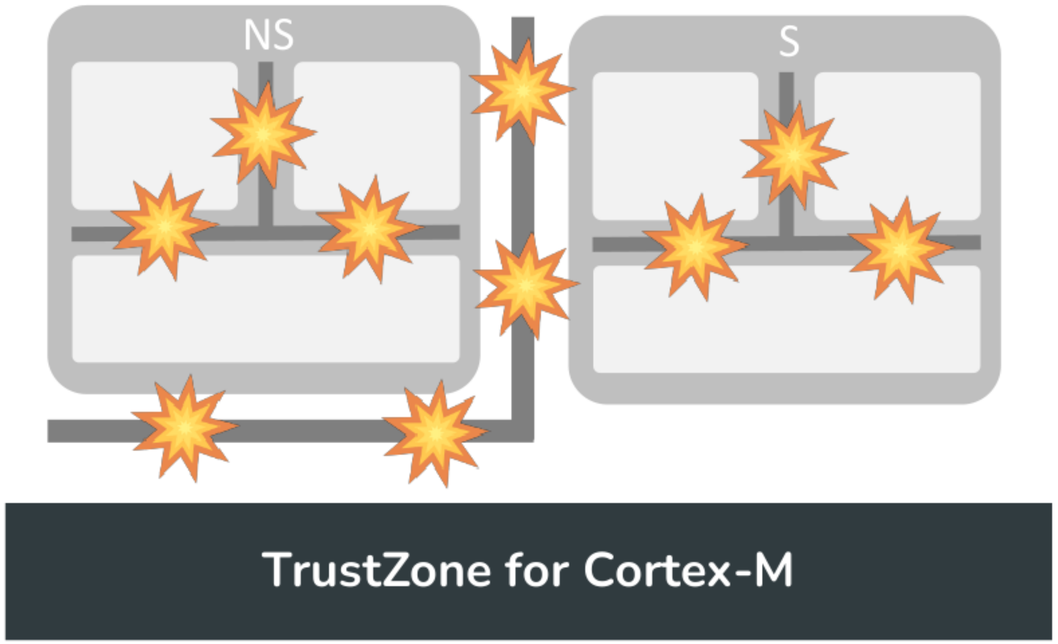
Hardware Gadgets

Smart Gadget Network

BUSted

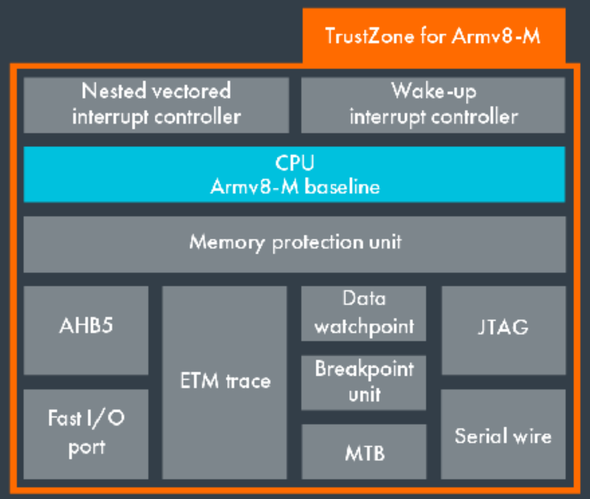
Attack





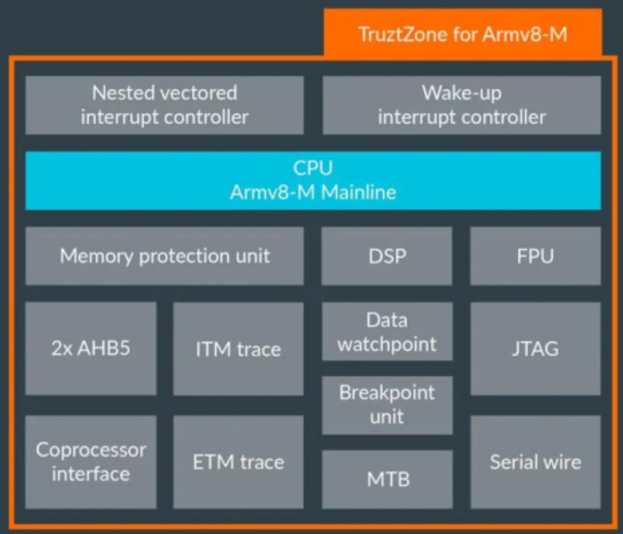
BUSted Attack

arm CORTEX®-M23



arm CORTEX®-M33

Common Criteria EAL6+ Certified



AGENDA

01 **Introduction**
Motivation and Side-Channels “101”

02 **Hidden Threat**
Novel Side-Channel Source: MCU Bus Interconnect

03 **“Toy” Example**
Basic Attack Example

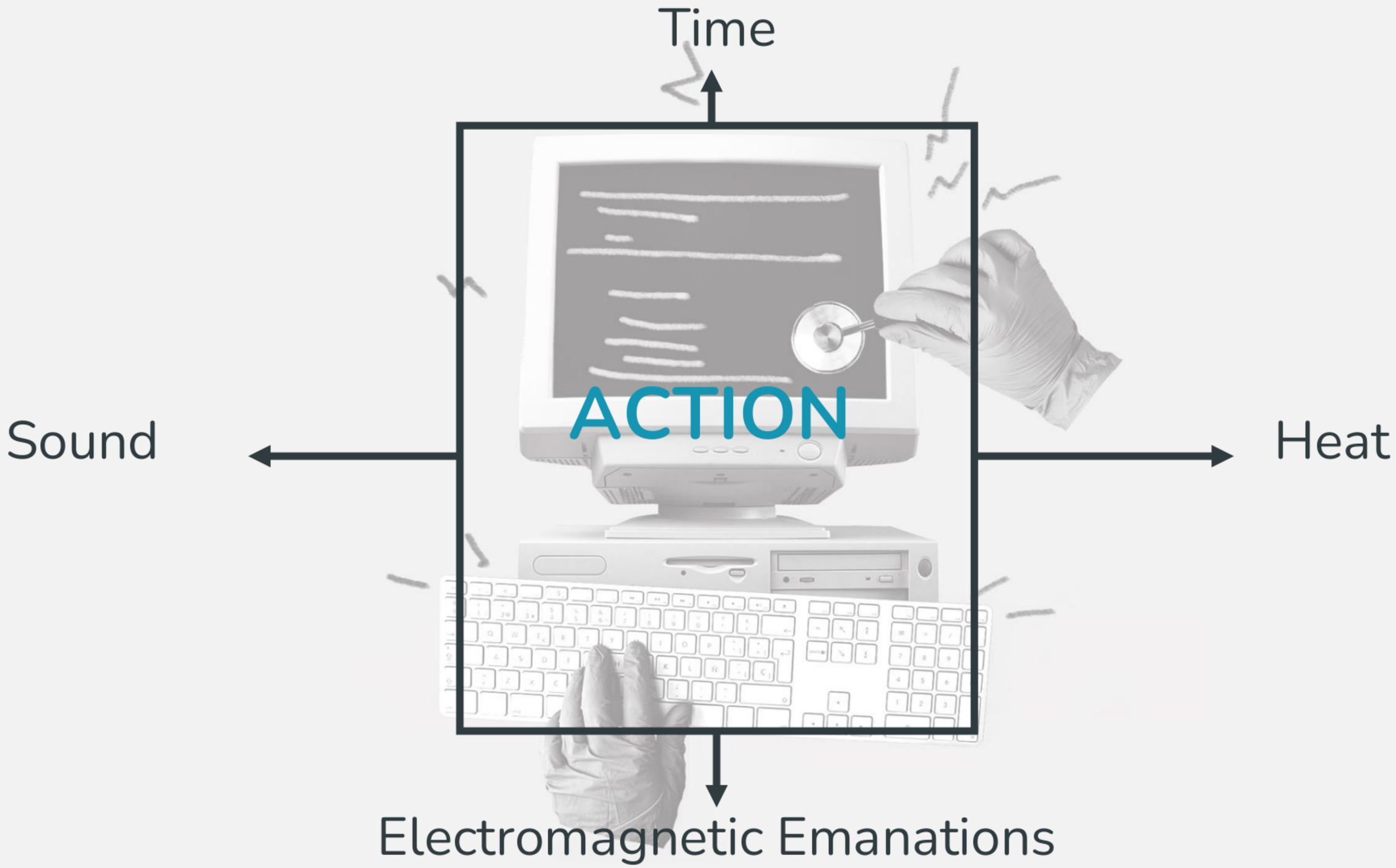
04 **BUSted Attack**
Microarchitectural Side-Channel Attacks on MCUs

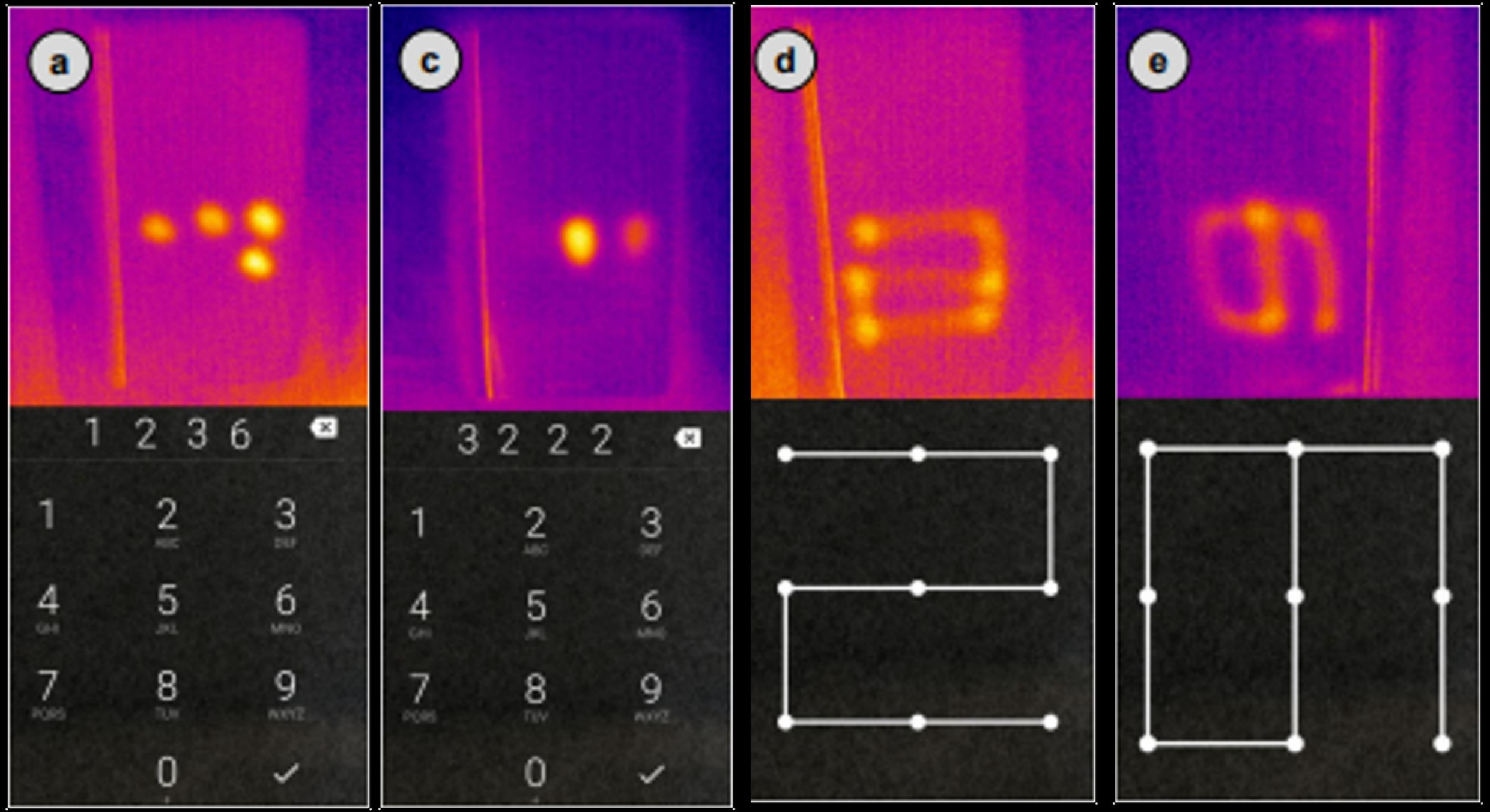
05 **“Live” Demo**
Demo of BUSted Attack

06 **Summary**
Responsible Disclosure and BH Sound Bytes

Introduction

Motivation and Side-Channels “101”


Time
Sound
ACTION
Heat
Electromagnetic Emanations



"Stay Cool! Understanding Thermal Attacks on Mobile-based User Authentication" by Abdelrahman and Khamis

“RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis” by Daniel Genkin

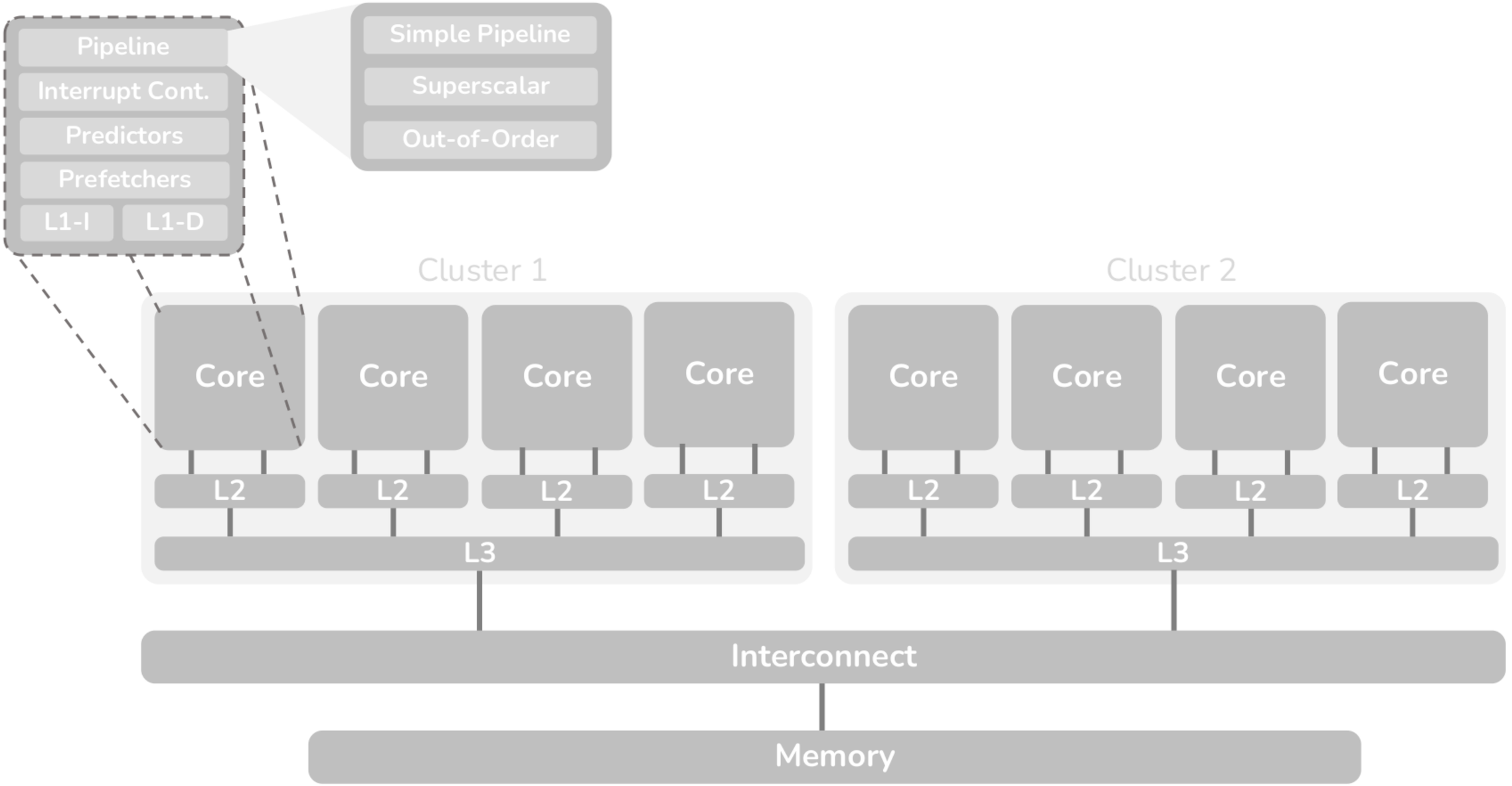
“Lamphone: Passive Sound Recovery from a Desk Lamp's Light Bulb Vibrations” by Ben Nassi

Lamphone Lightbulb Spies

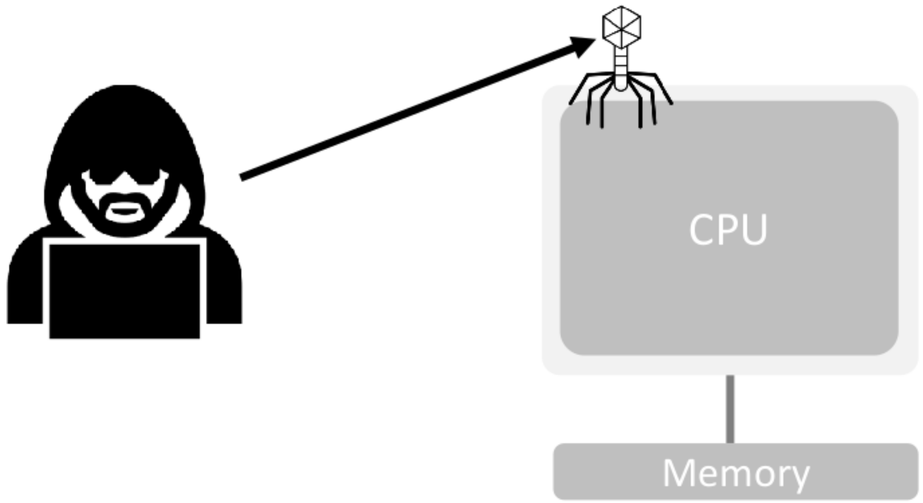
MELTDOWN

SPECTRE

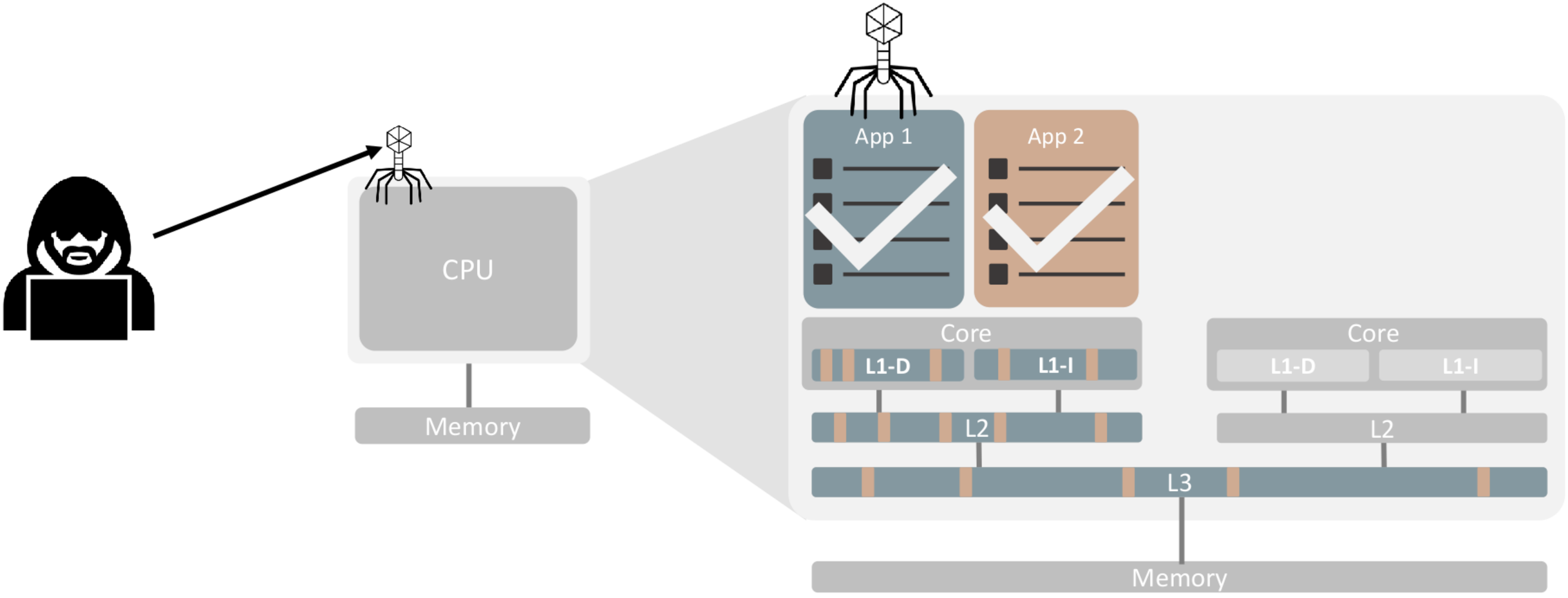
TIMING DIFFERENCES


Pipeline
Interrupt Cont.
Predictors
Prefetchers
L1-I
L1-D
Simple Pipeline
Superscalar
Out-of-Order
Cluster 1
Cluster 2
Core
Core
Core
Core
Core
Core
Core
Core
L2
L2
L2
L2
L2
L2
L2
L2
L3
L3
Interconnect
Memory

Microarchitectural Attacks



Microarchitectural Attacks




TECH / SECURITY
Protecting your
latest security
Spectre is goi
Dieter Bohn
Spectre attacks against websites still a serious threat, Google warns
Ben Dickson 17 March 2021 at 12:10 UTC
Updated: 28 June 2021 at 09:12 UTC
The New Y
SECURITY MAY 21, 2018 8:54 PM
ter Meltdown and Spectre, Another Scar
pook.js – New side-channel attack can bypass Google Chrome's protections against Spectre-style exploits
Jessica Haworth 10 September 2021 at 11:06 UTC
Updated: 10 September 2021 at 11:24 UTC
em
Kernel panic! Wh
bugs affecting nearly every
Devin Coldewey @techcrunch / 1:47 AM GMT • January 4, 2018
New speculative
from Intel chips' internal
Intel-specific vulnerability was found by researchers both insi
ZDNET tomorrow belongs to those who embrace it today
Home / Tech / Security
New CrossTalk attack impacts Intel's mobile, desktop, and server CPUs
Ron Mill
/ 3:31 PM GMT • January 26, 2018

APUs

MCUs

Servers

Mobile

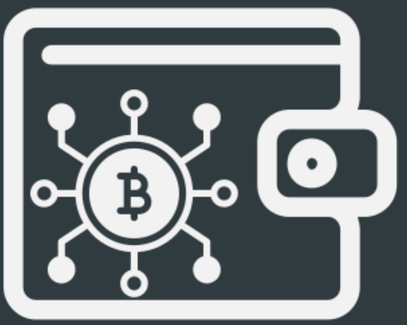
Drones

Wearables

PCs

Appliances

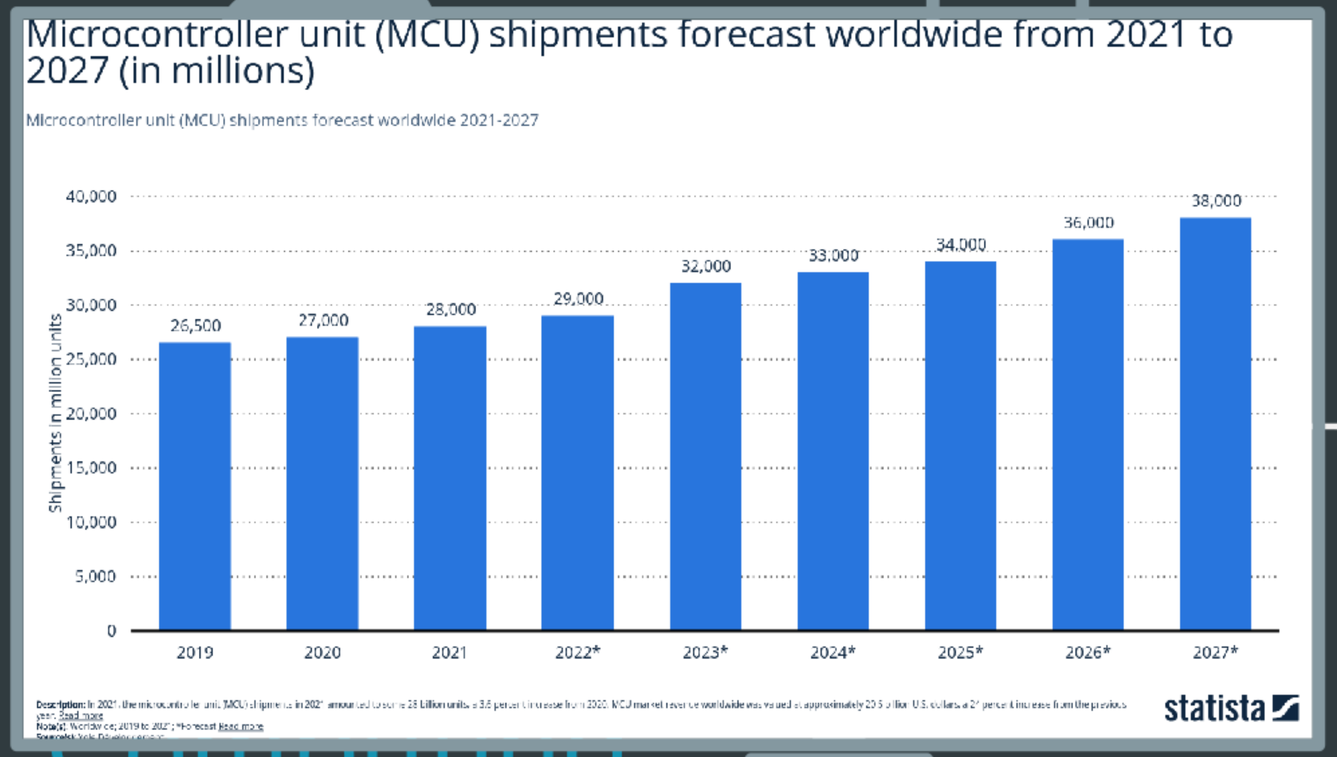
Hardware Wallets



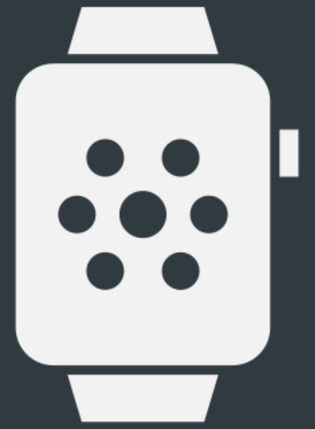
Computing spectrum

APUs

MCUs



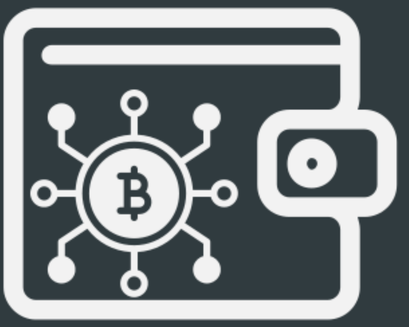
Drones



Wearables

Appliances

Hardware Wallets

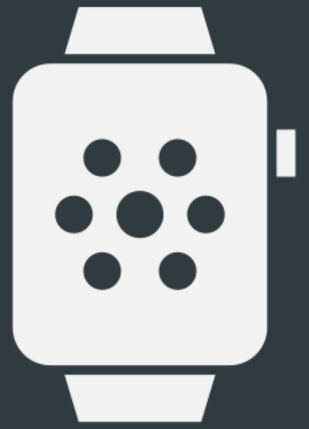


Computing spectrum

APUs

MCUs

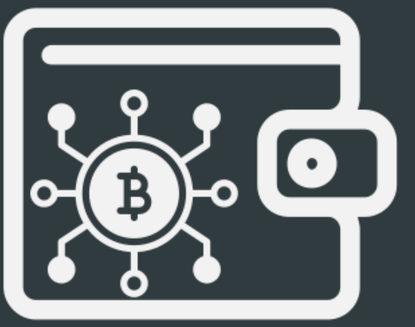
Drones



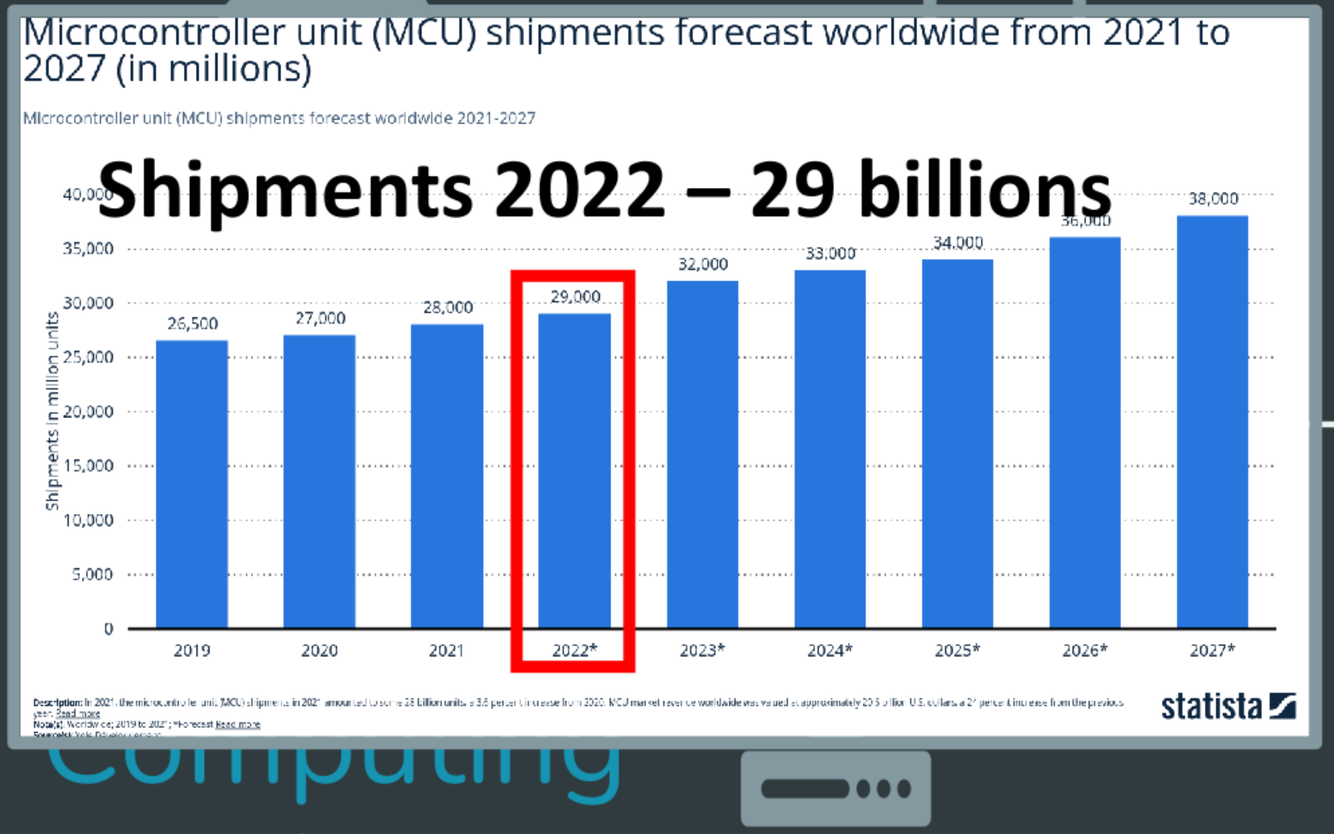
Wearables

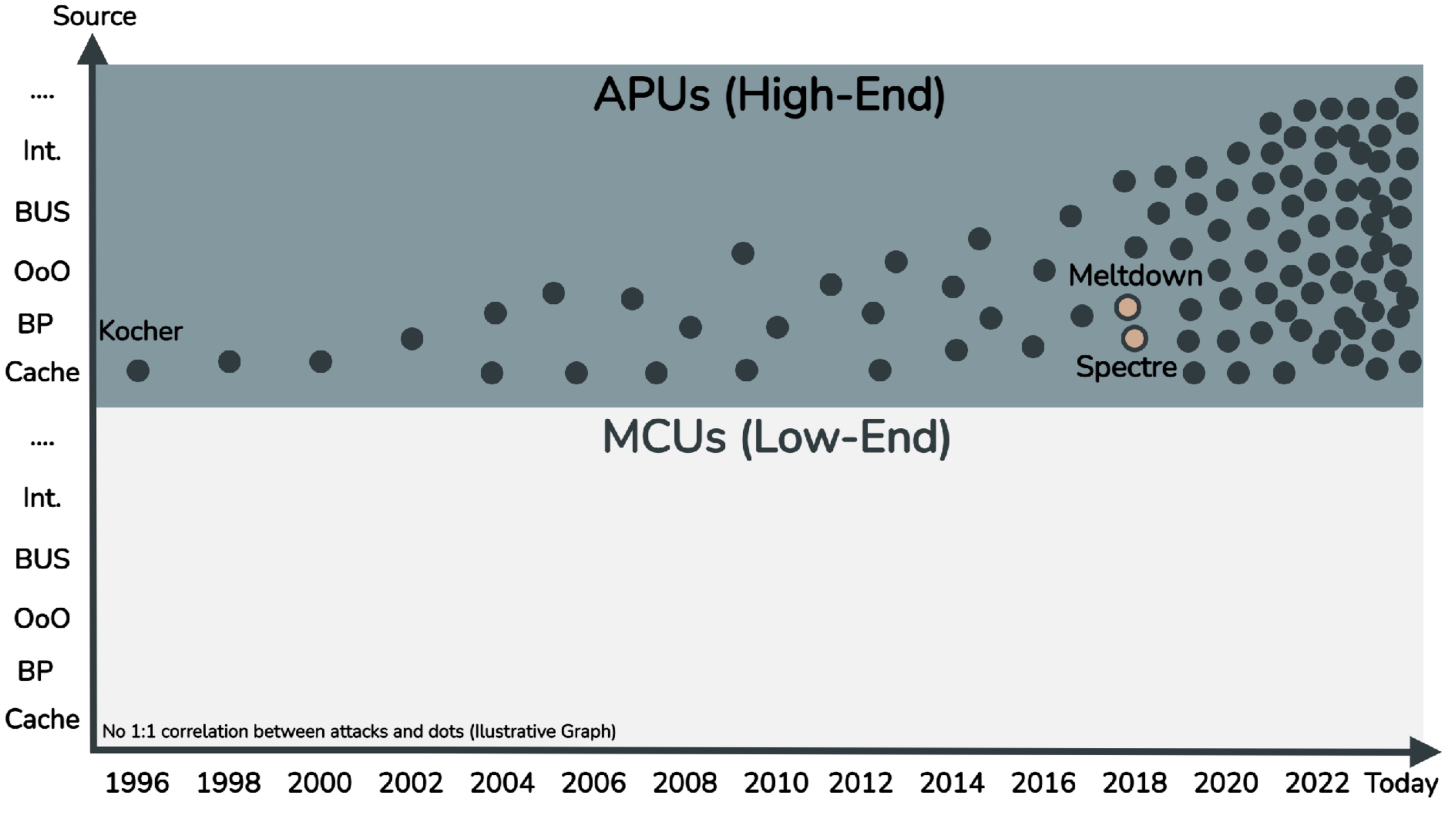
Appliances

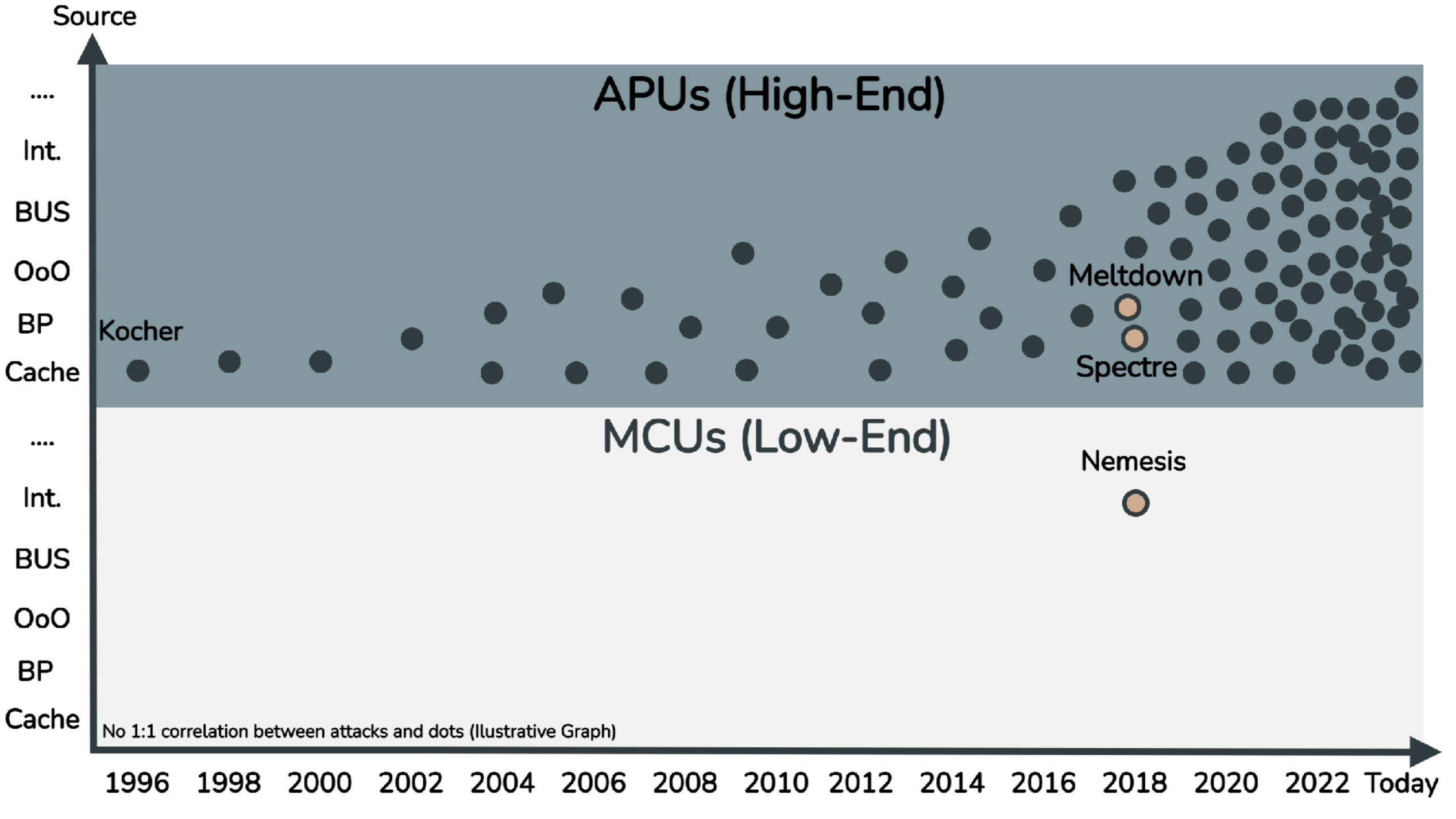
Hardware Wallets



Computing spectrum



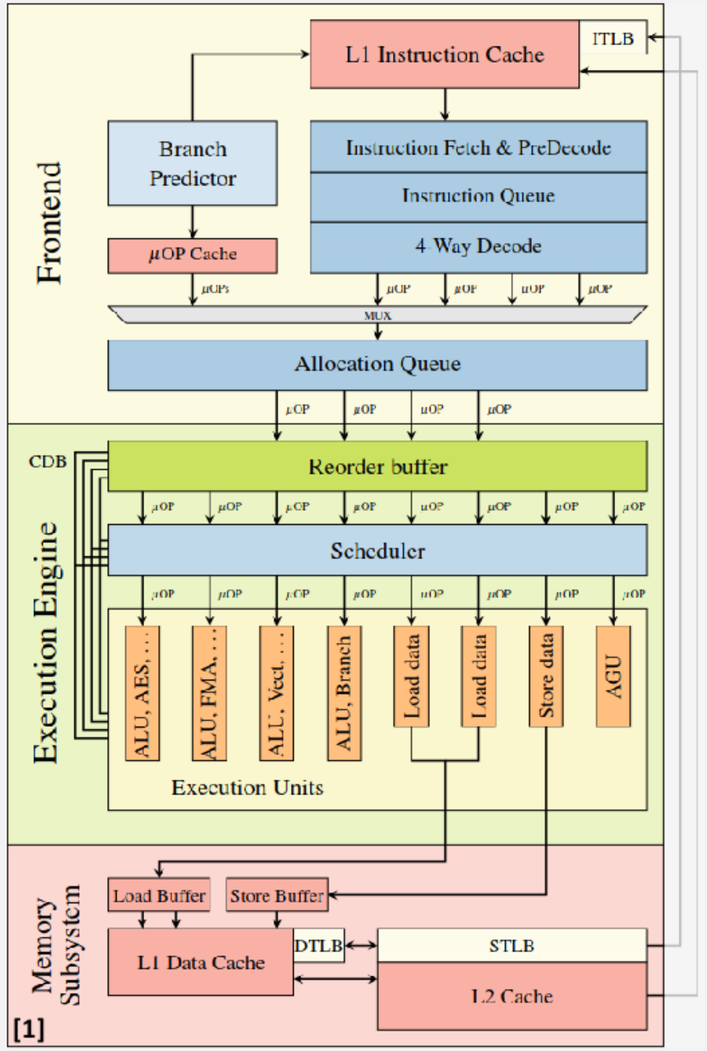

Source
....
Int.
BUS
OoO
BP
Cache
APUs (High-End)
Kocher
Meltdown
Spectre
MCUs (Low-End)
....
Int.
BUS
OoO
BP
Cache
No 1:1 correlation between attacks and dots (Ilustrative Graph)
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
Today


Source
....
Int.
BUS
OoO
BP
Cache
APUs (High-End)
Kocher
Meltdown
Spectre
MCUs (Low-End)
Nemesis
....
Int.
BUS
OoO
BP
Cache
No 1:1 correlation between attacks and dots (Ilustrative Graph)
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
Today

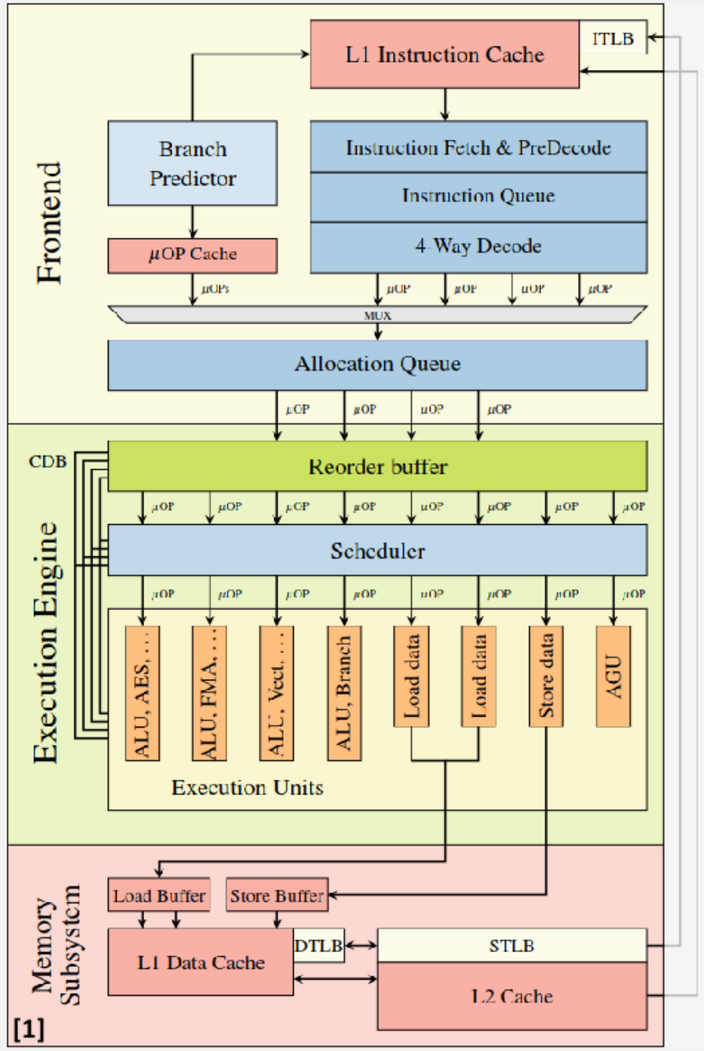
Intel Skylake Microarchitecture

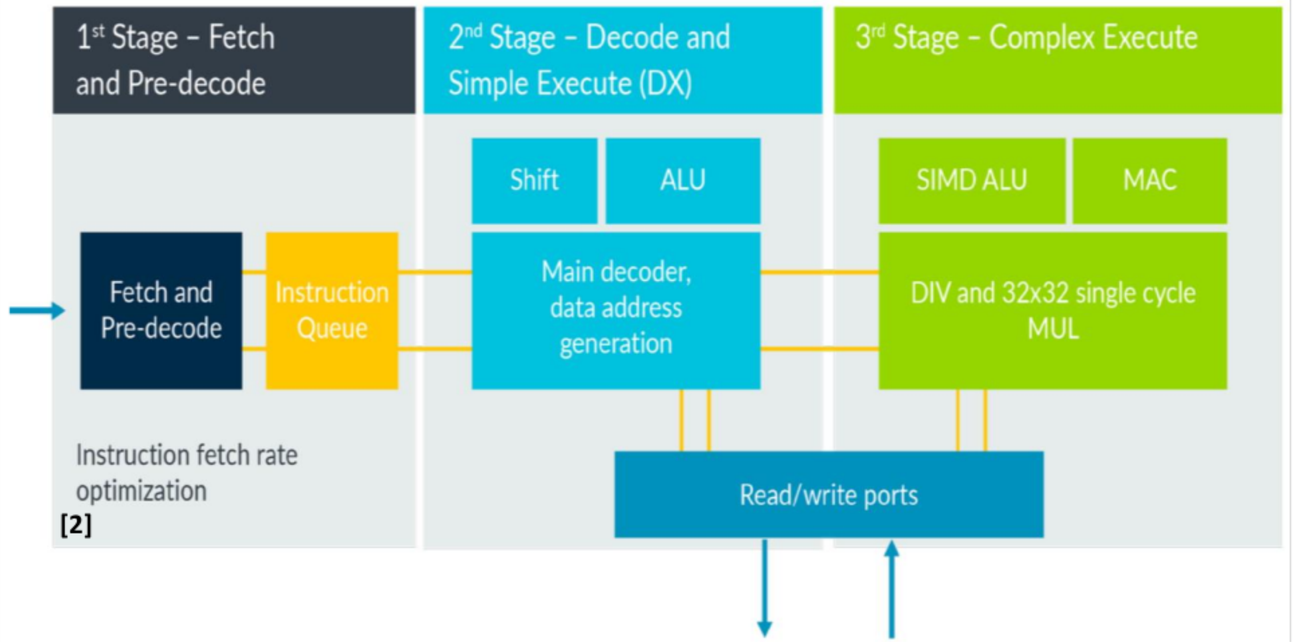
Arm Cortex-M33 Microarchitecture

Arm Cortex-M33 Microarchitecture



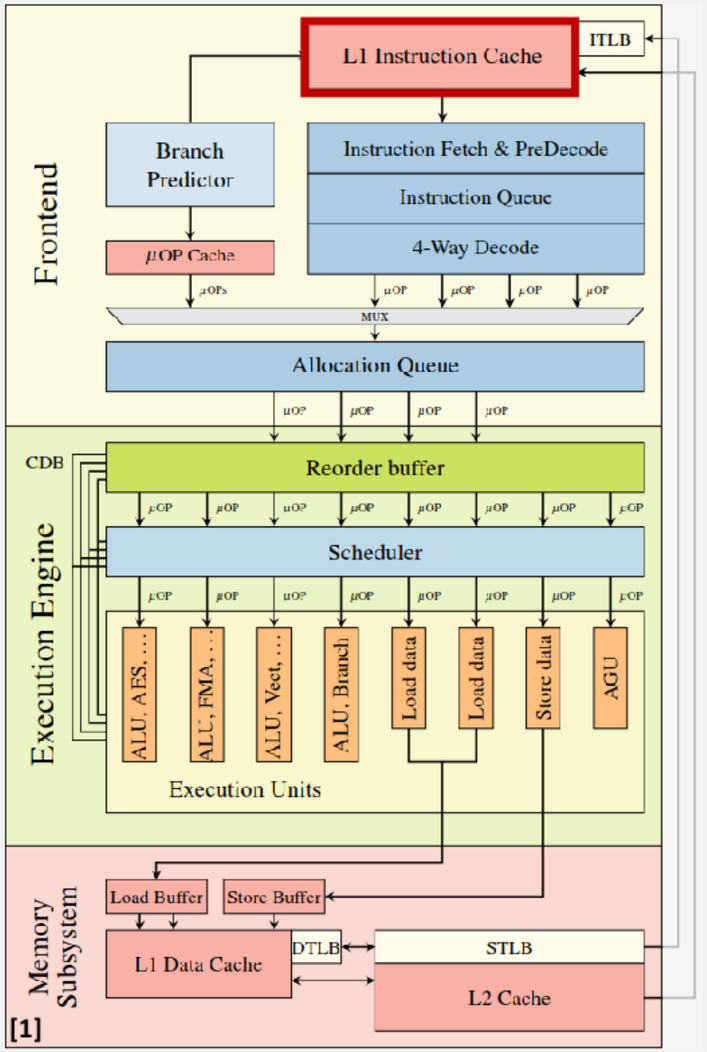
[1] - Meltdown: Reading Kernel Memory from User Space

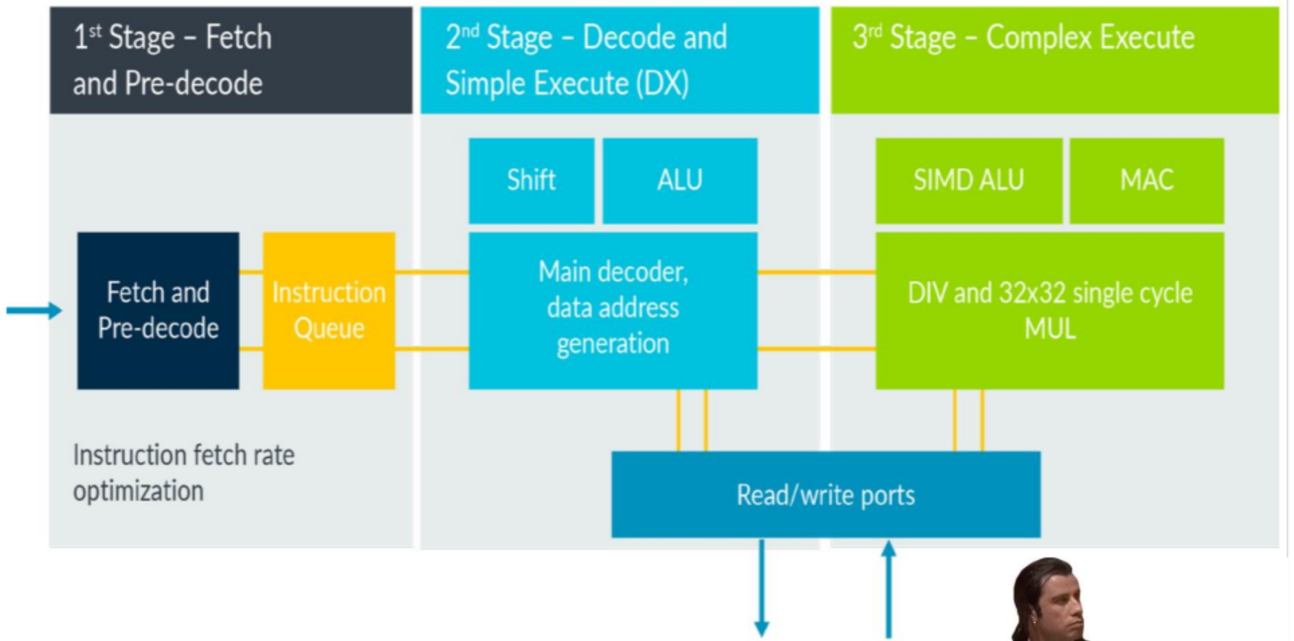




[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

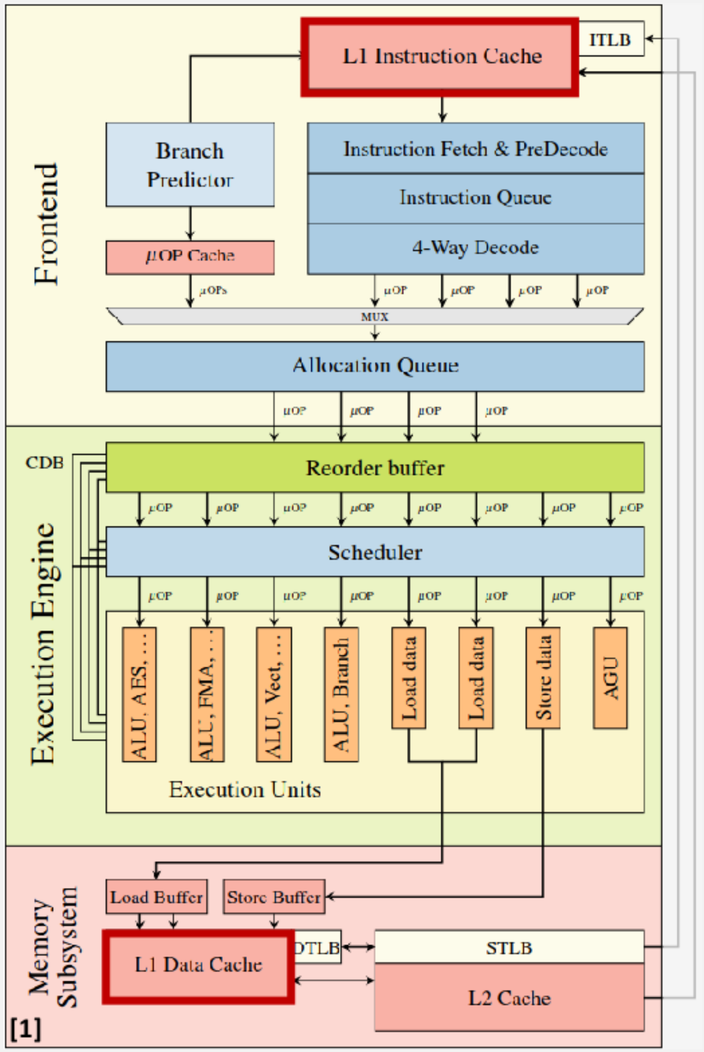
- L1-I

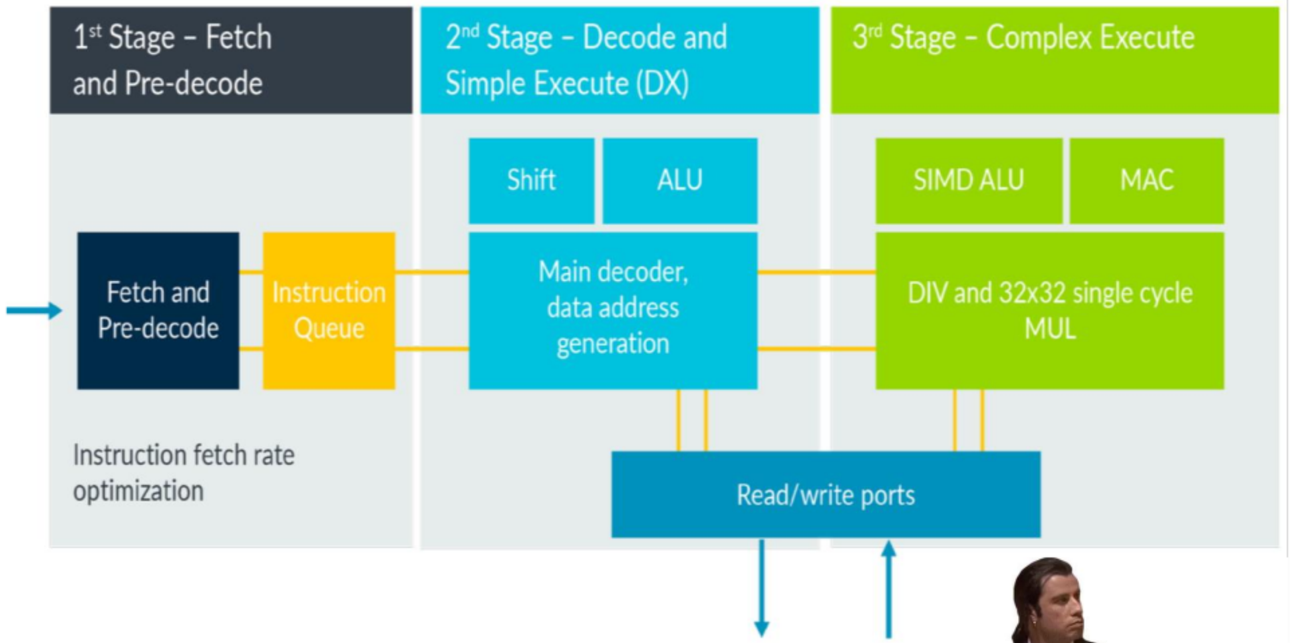




[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

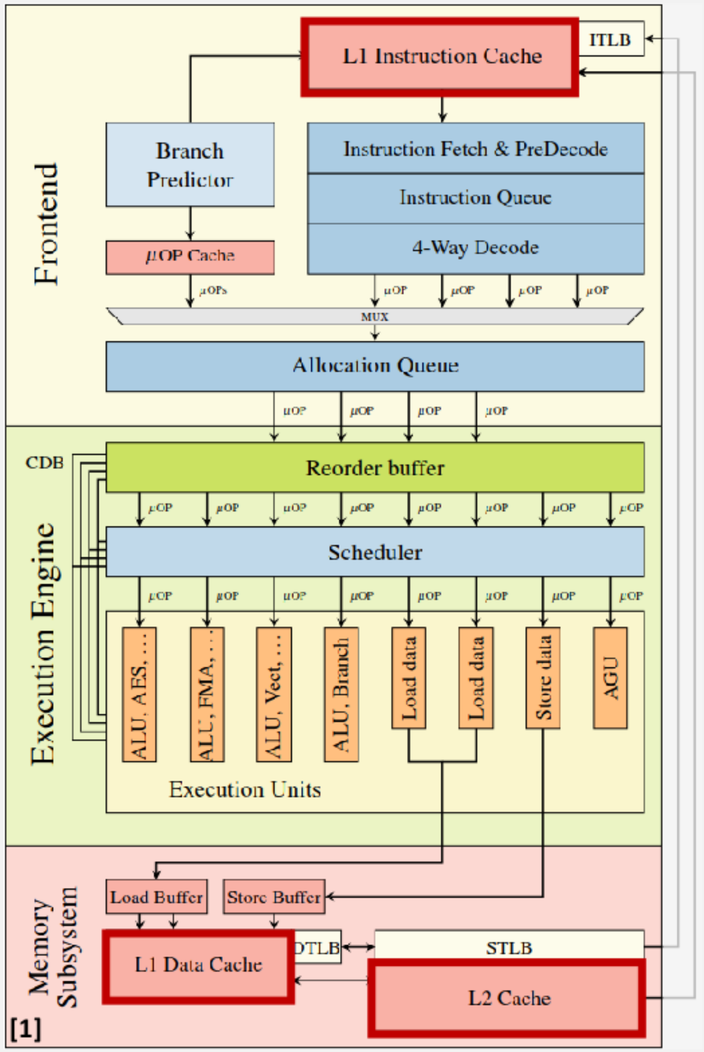
- L1-I
- L1-D

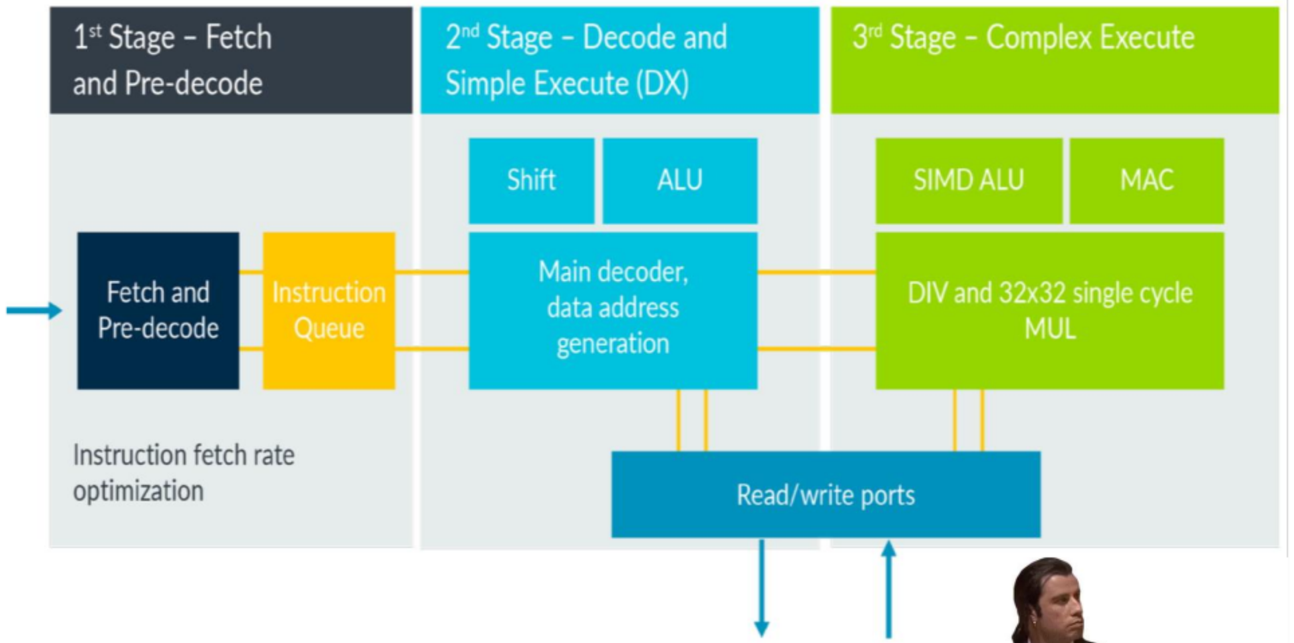




[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

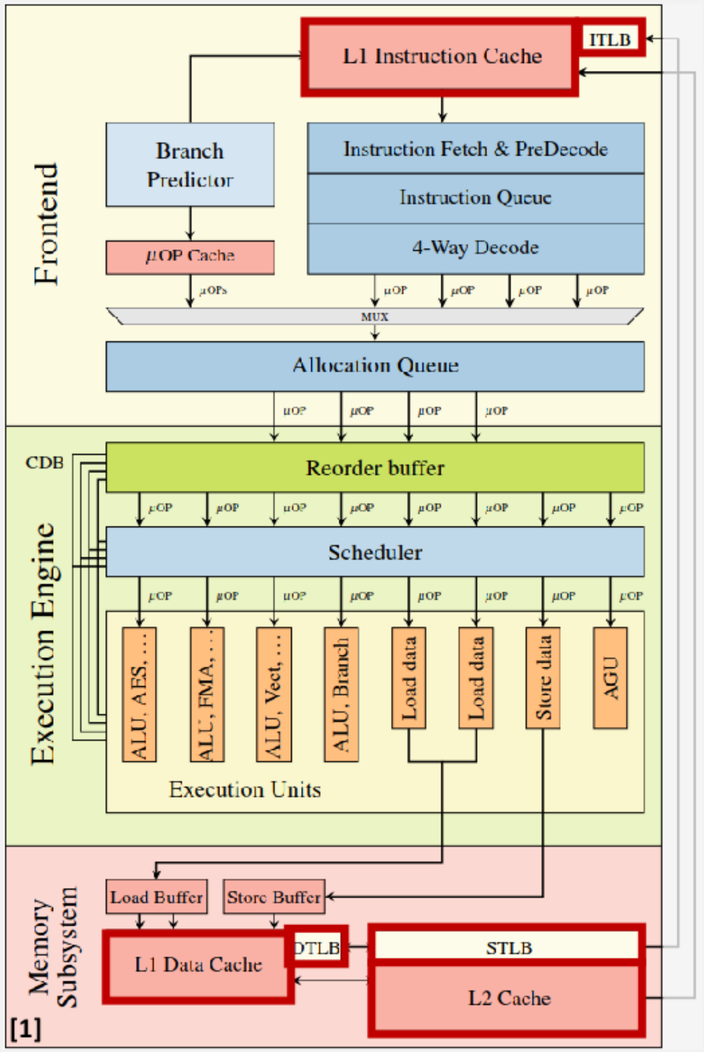
- L1-I
- L1-D
- L2

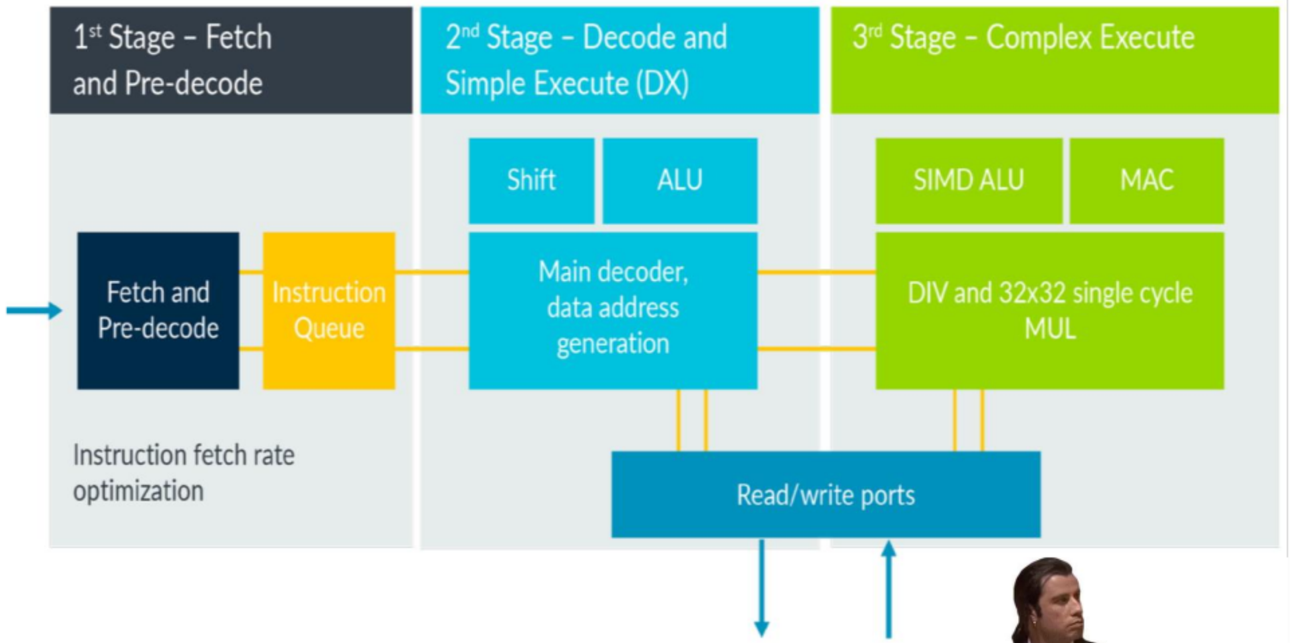




[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

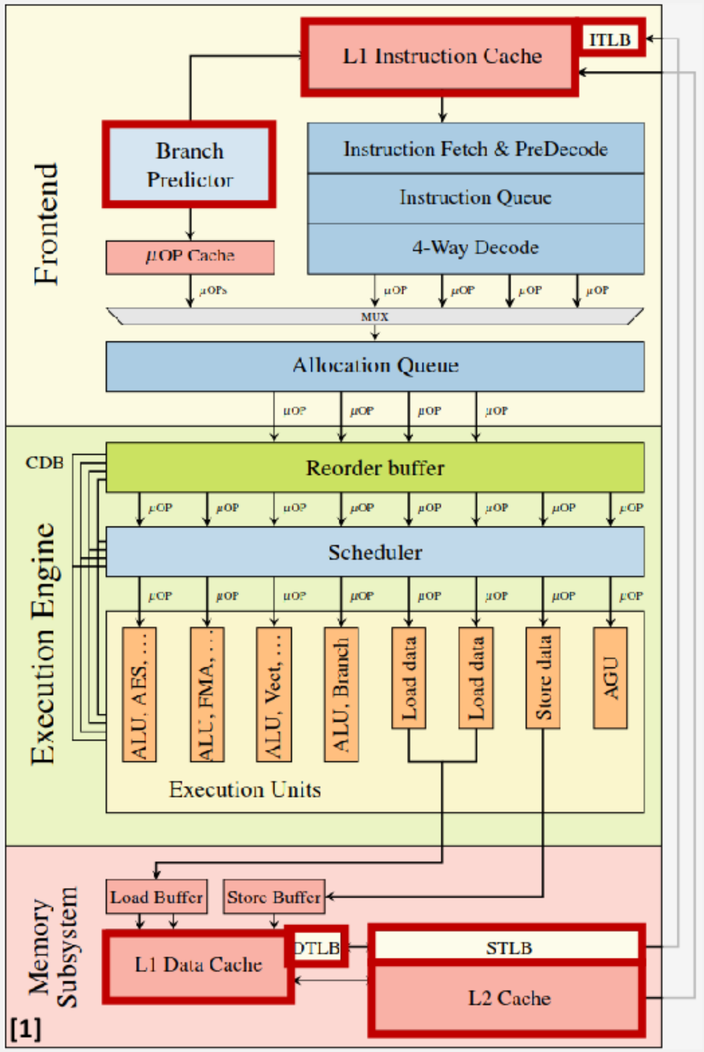
- L1-I
- L1-D
- L2
- TLBs

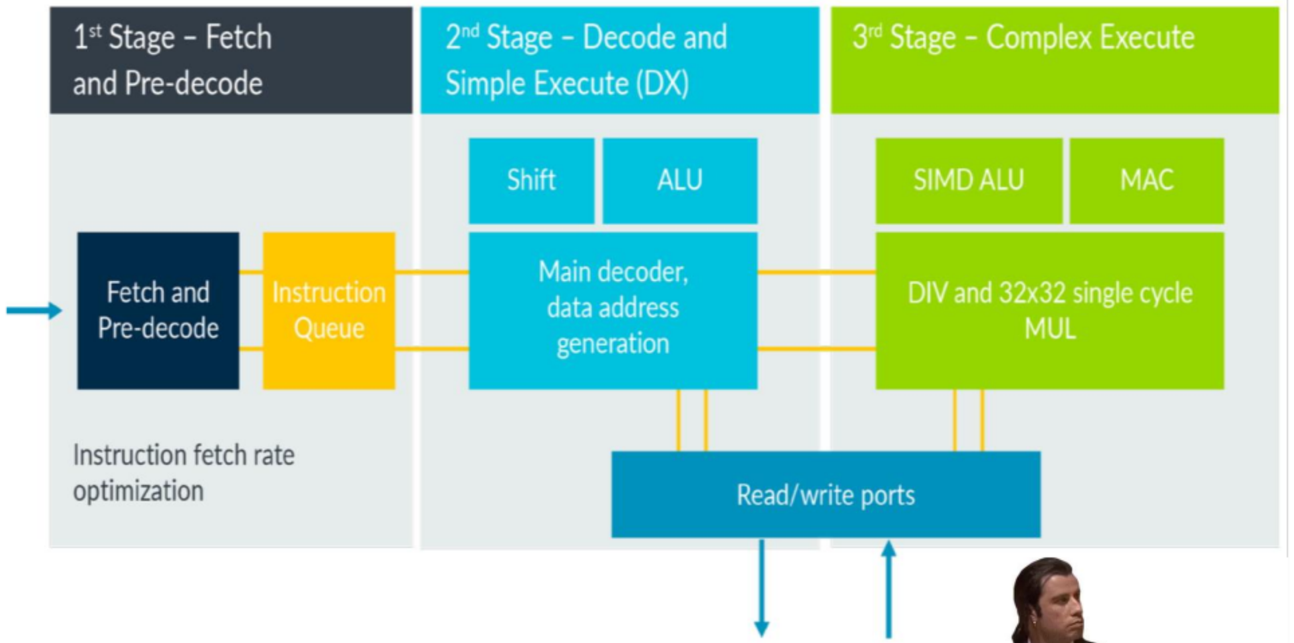




[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

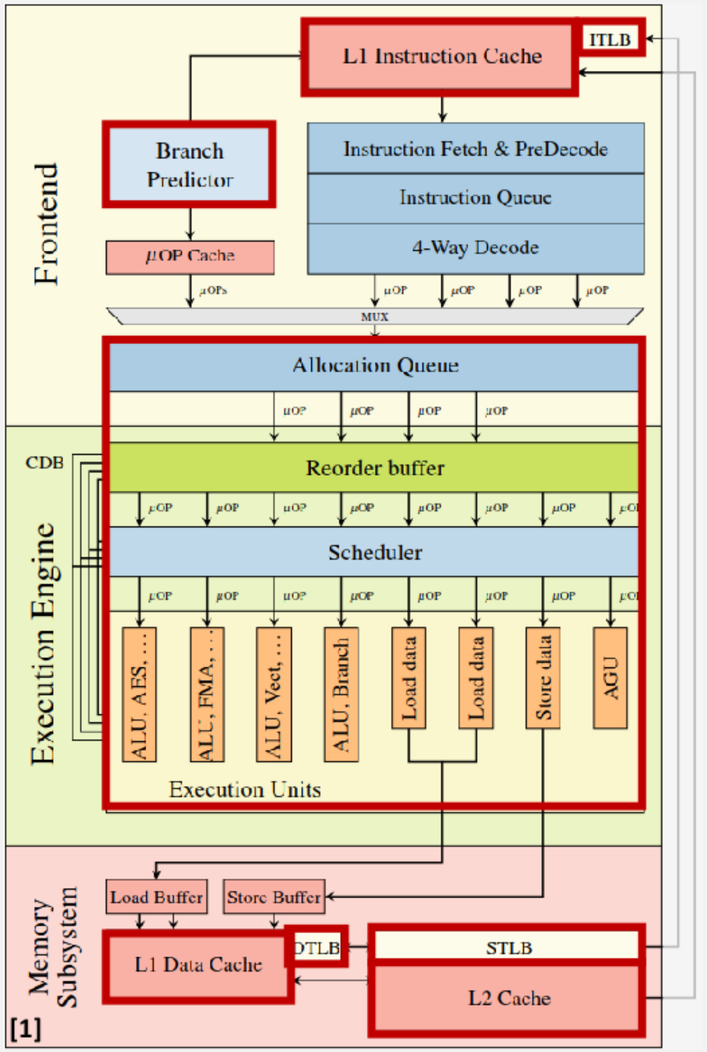
- L1-I
- L1-D
- L2
- TLBs
- BP

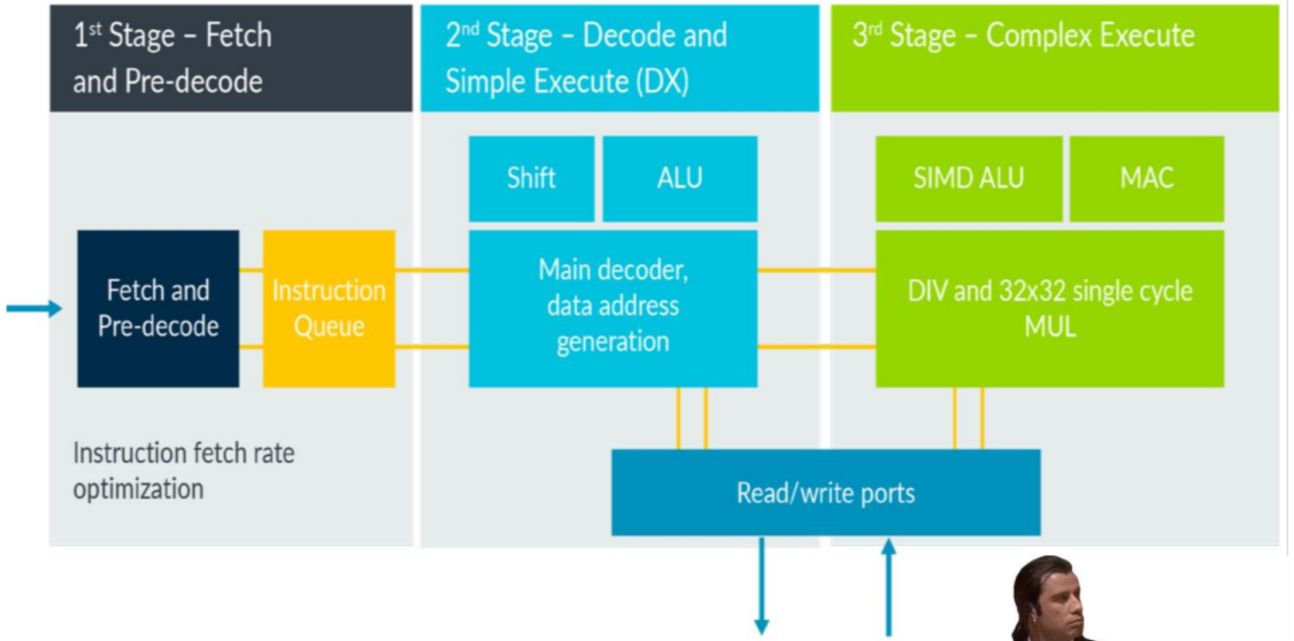




[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

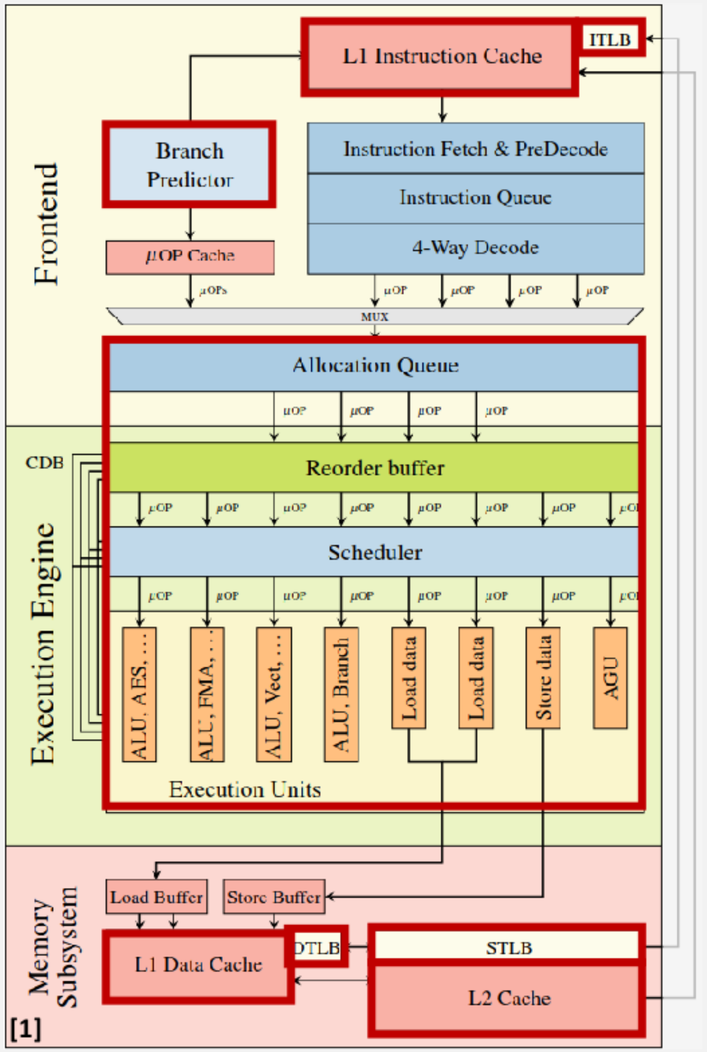
- L1-I
- L1-D
- L2
- TLBs
- BP
- OoO

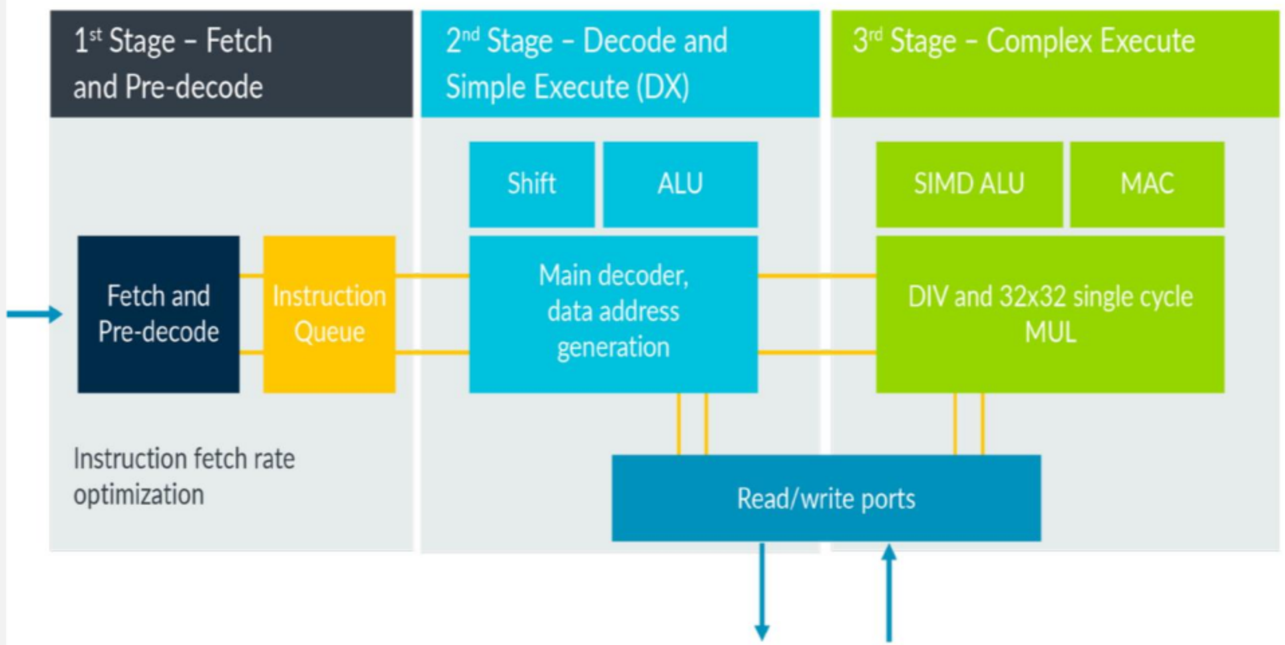




[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

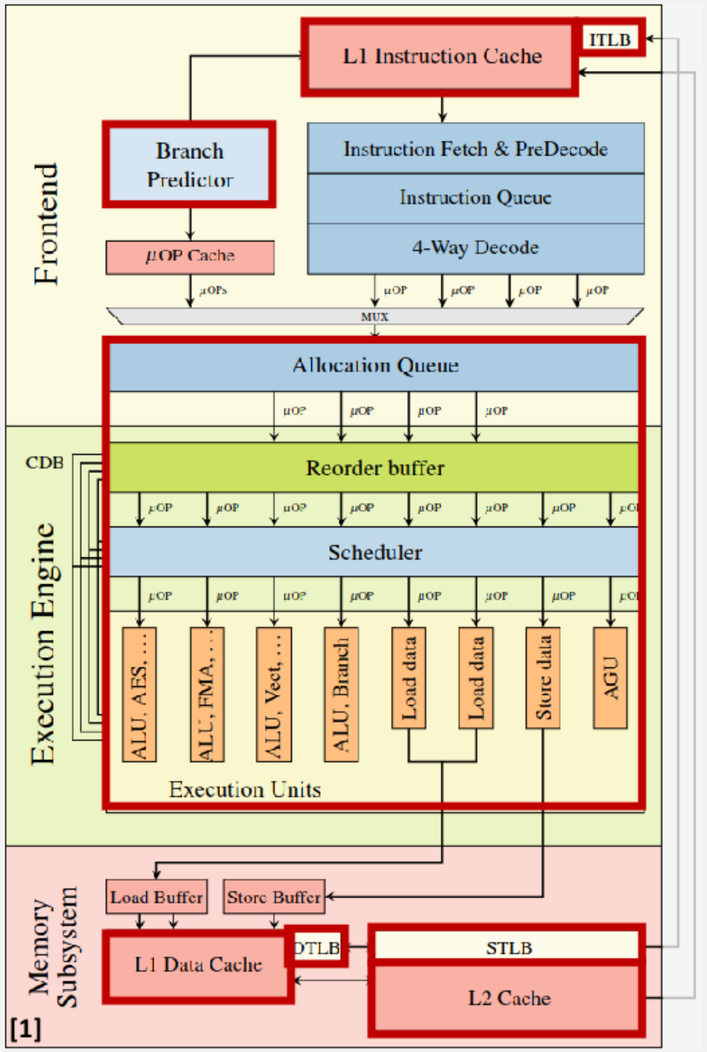
- L1-I
- L1-D
- L2
- TLBs
- BP
- OoO

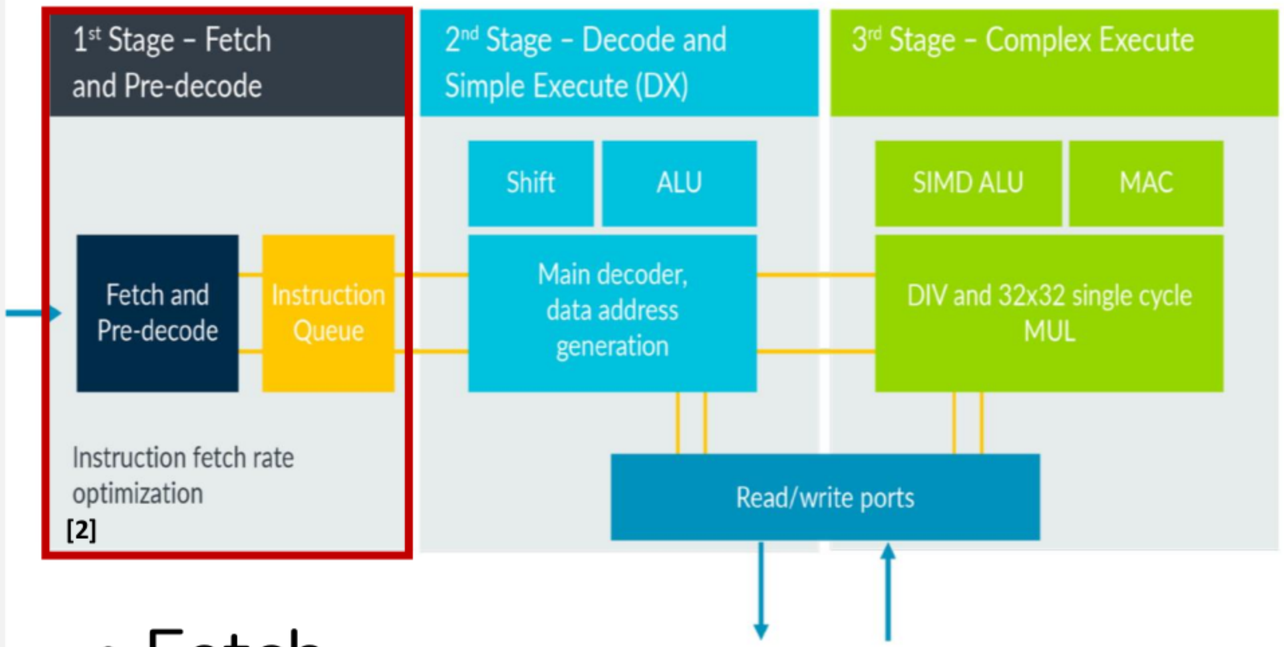




[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

- L1-I
- L1-D
- L2
- TLBs
- BP
- OoO

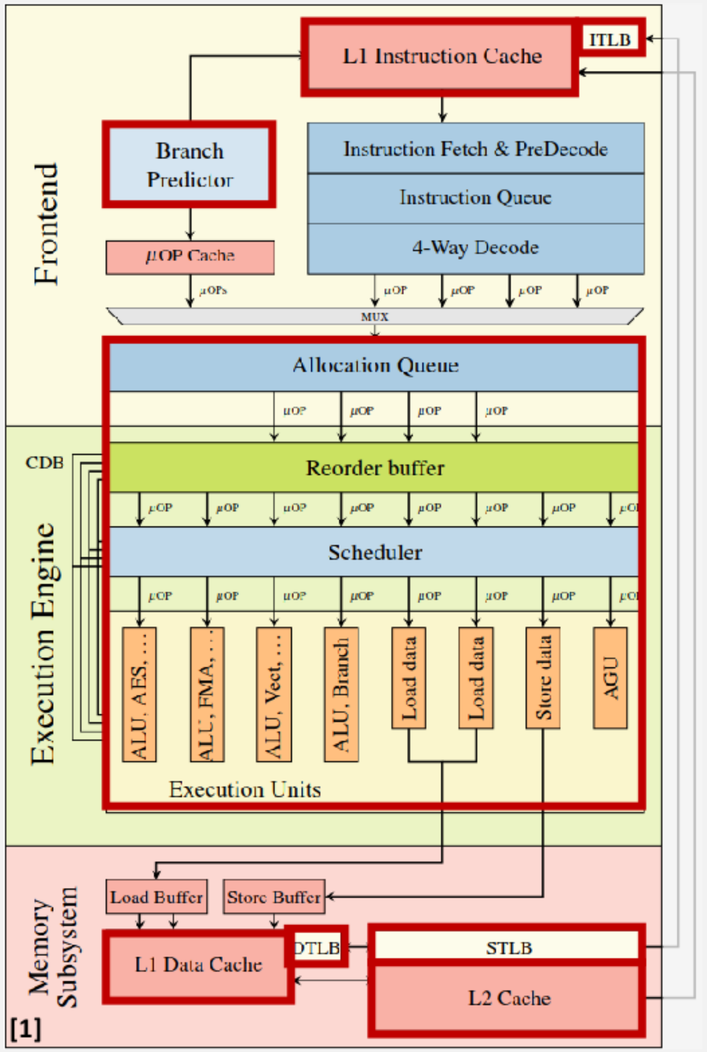


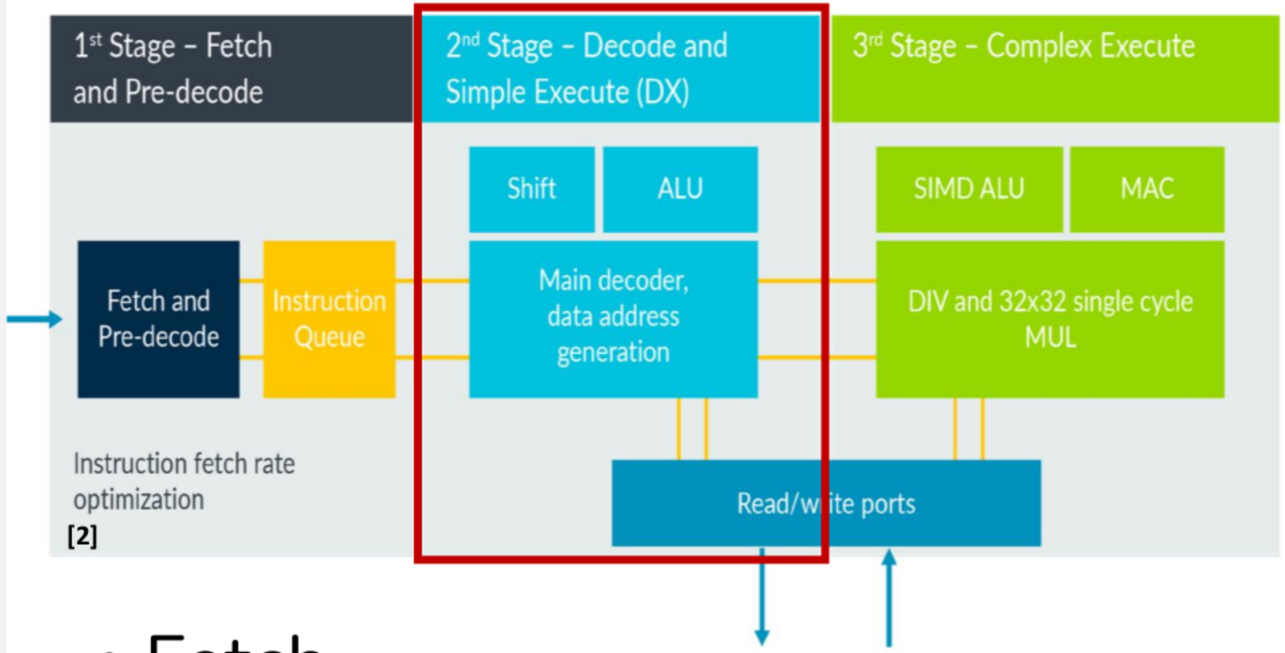


- Fetch

[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

- L1-I
- L1-D
- L2
- TLBs
- BP
- OoO

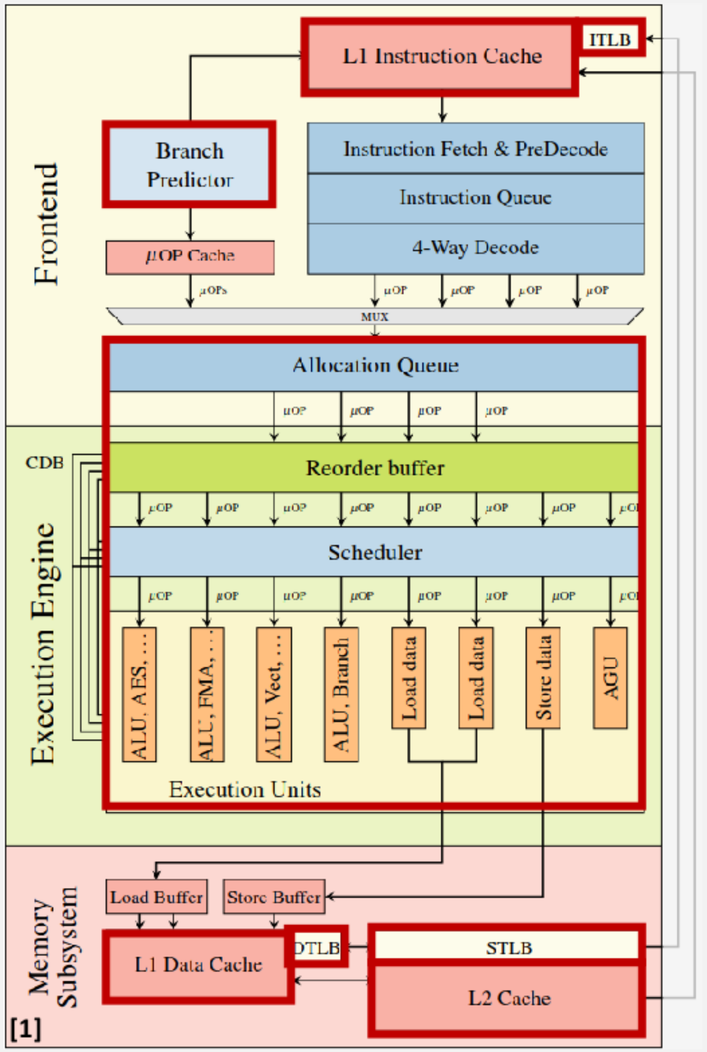


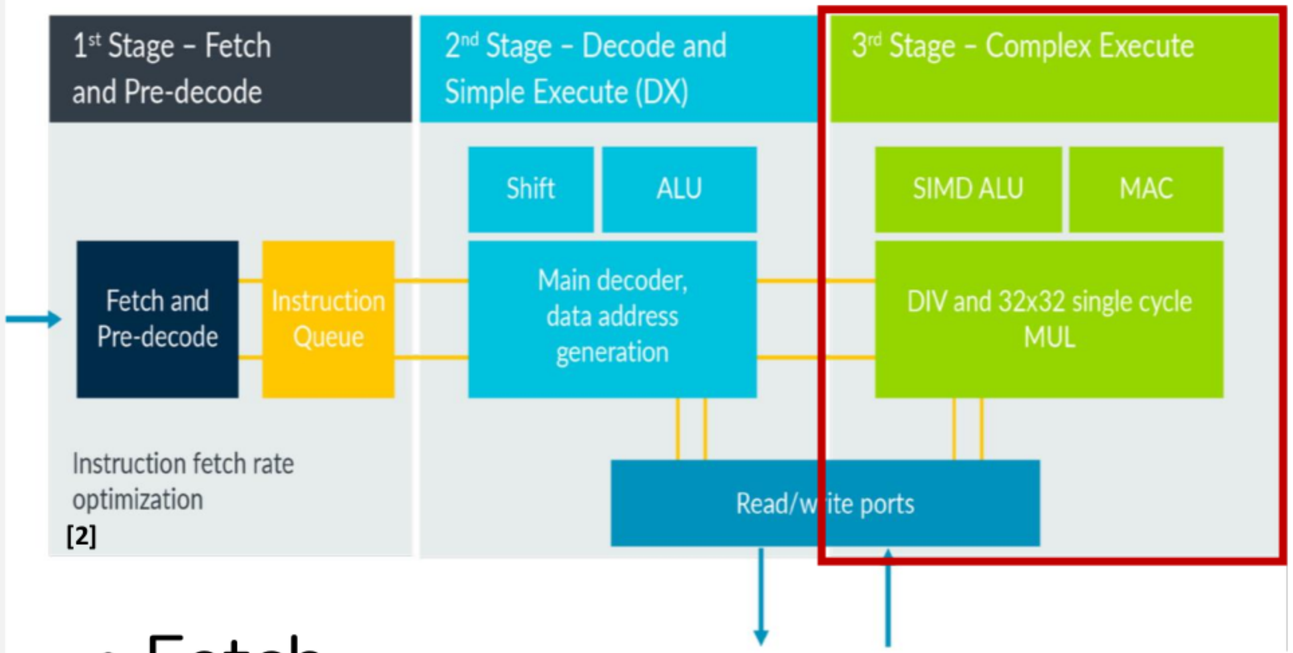


- Fetch
- Decode

[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

- L1-I
- L1-D
- L2
- TLBs
- BP
- OoO

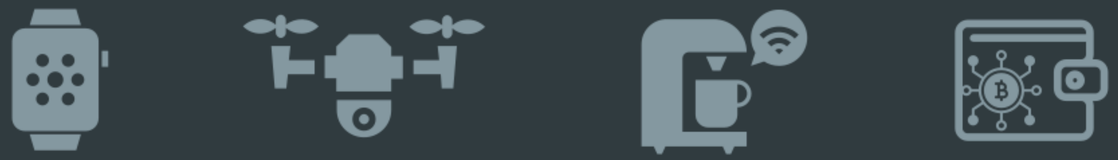




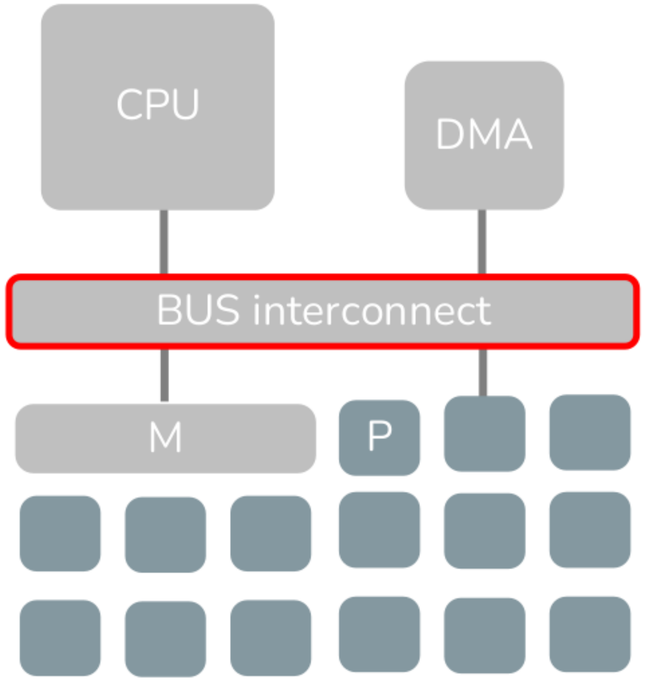
- Fetch
- Decode
- Execute

[1] - Meltdown: Reading Kernel Memory from User Space
[2] - Arm Cortex-M33 Processor Datasheet

Microcontrollers



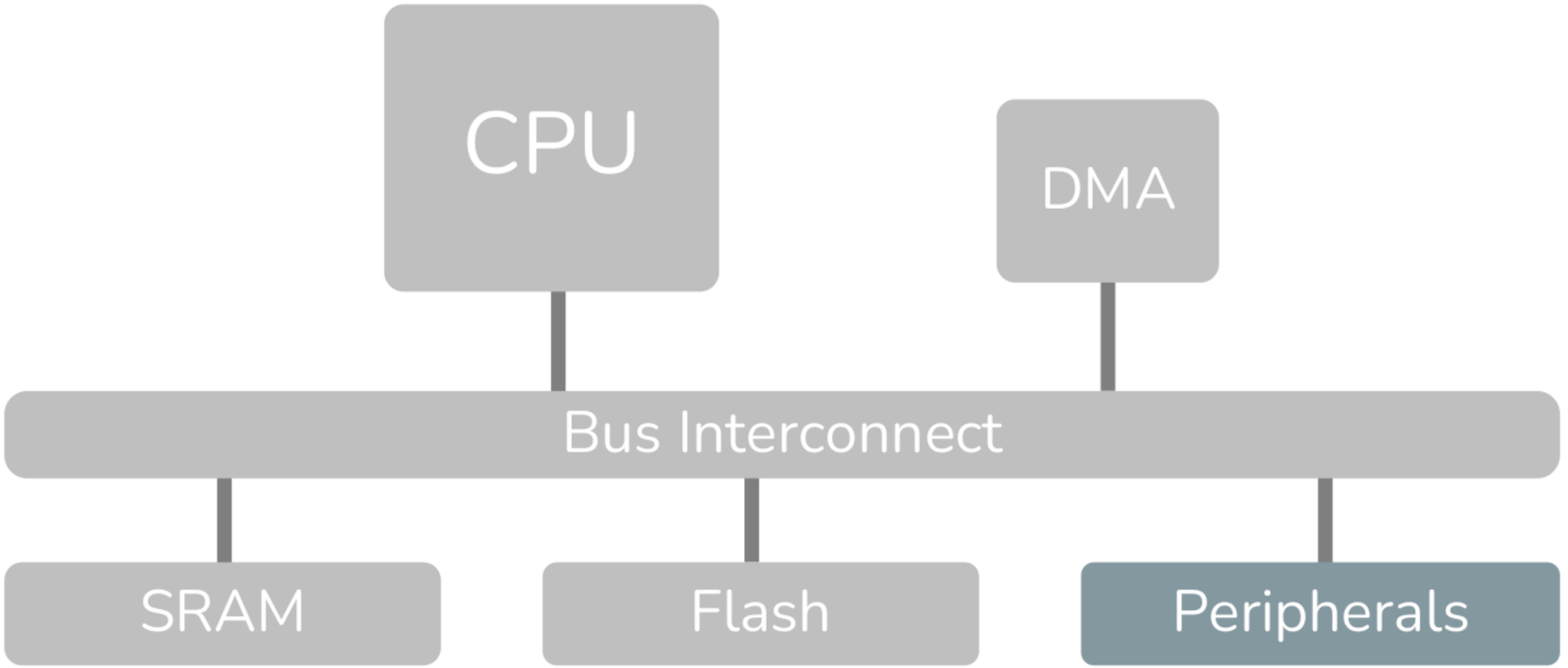
arm



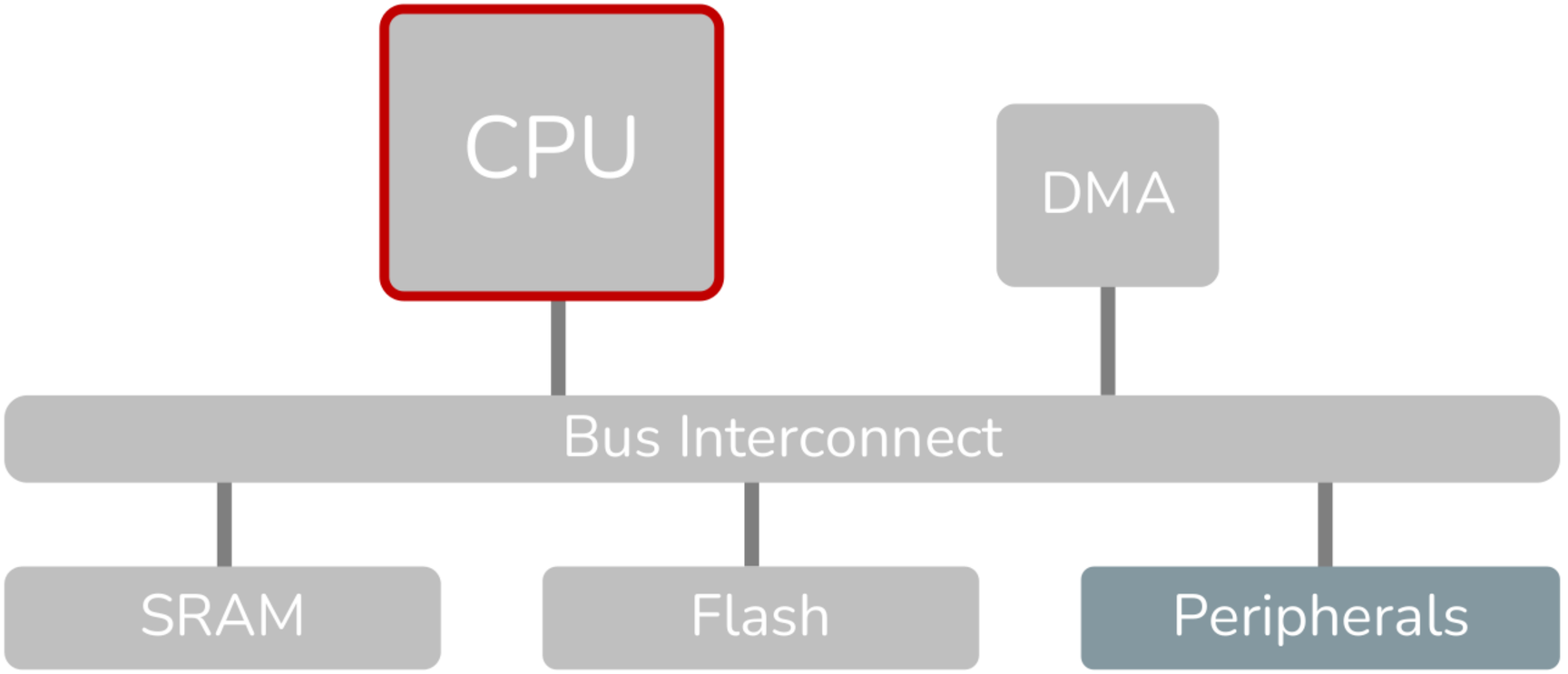
Hidden Threat

Novel Side-Channel Source: MCU Bus Interconnect

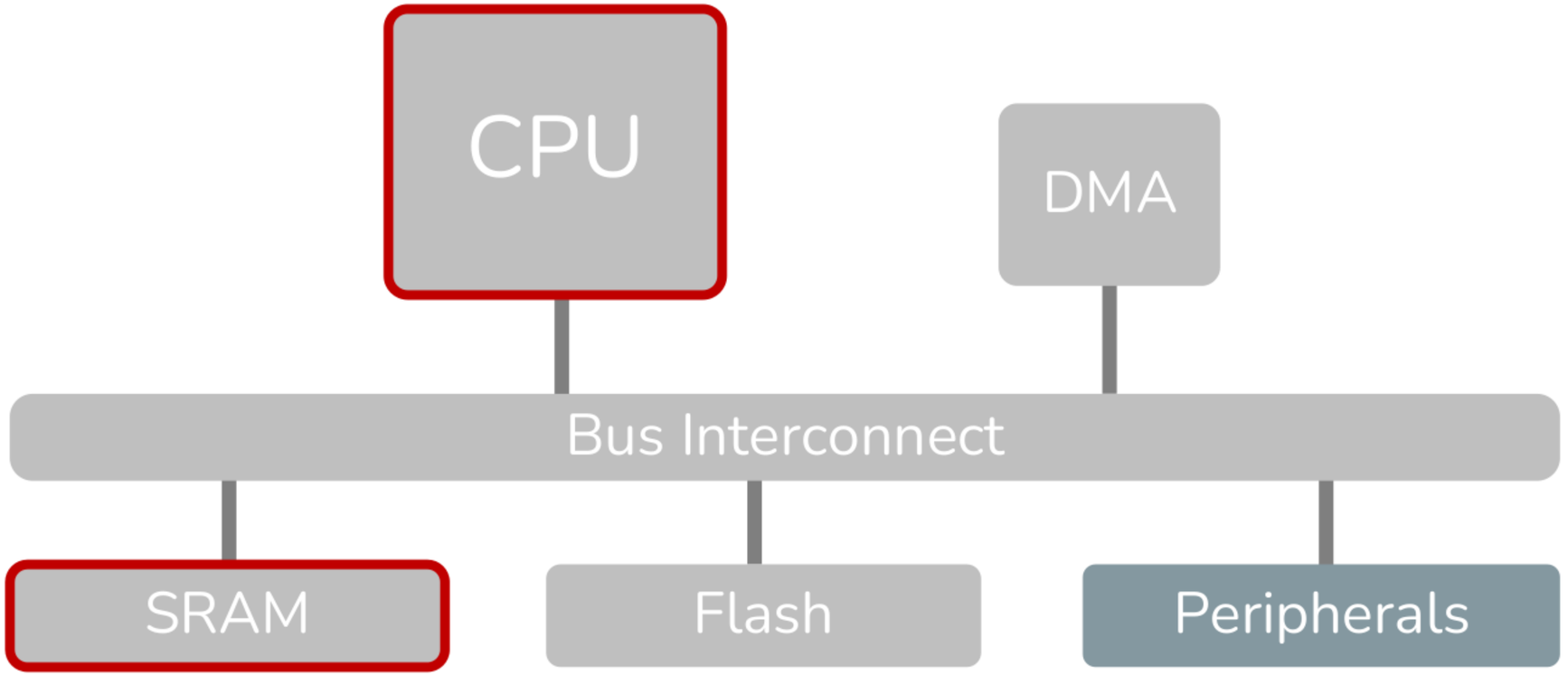
MCU Bus Interconnect as a Threat



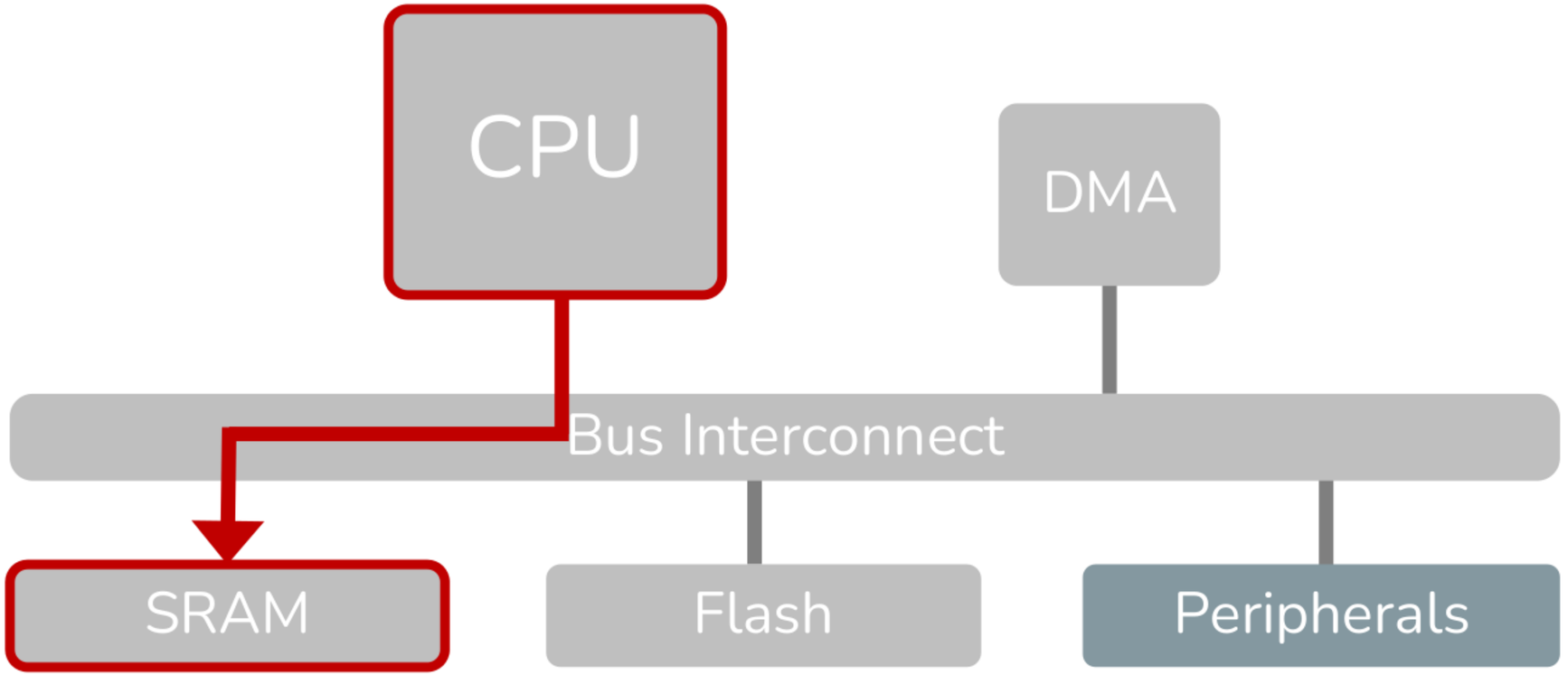
MCU Bus Interconnect as a Threat



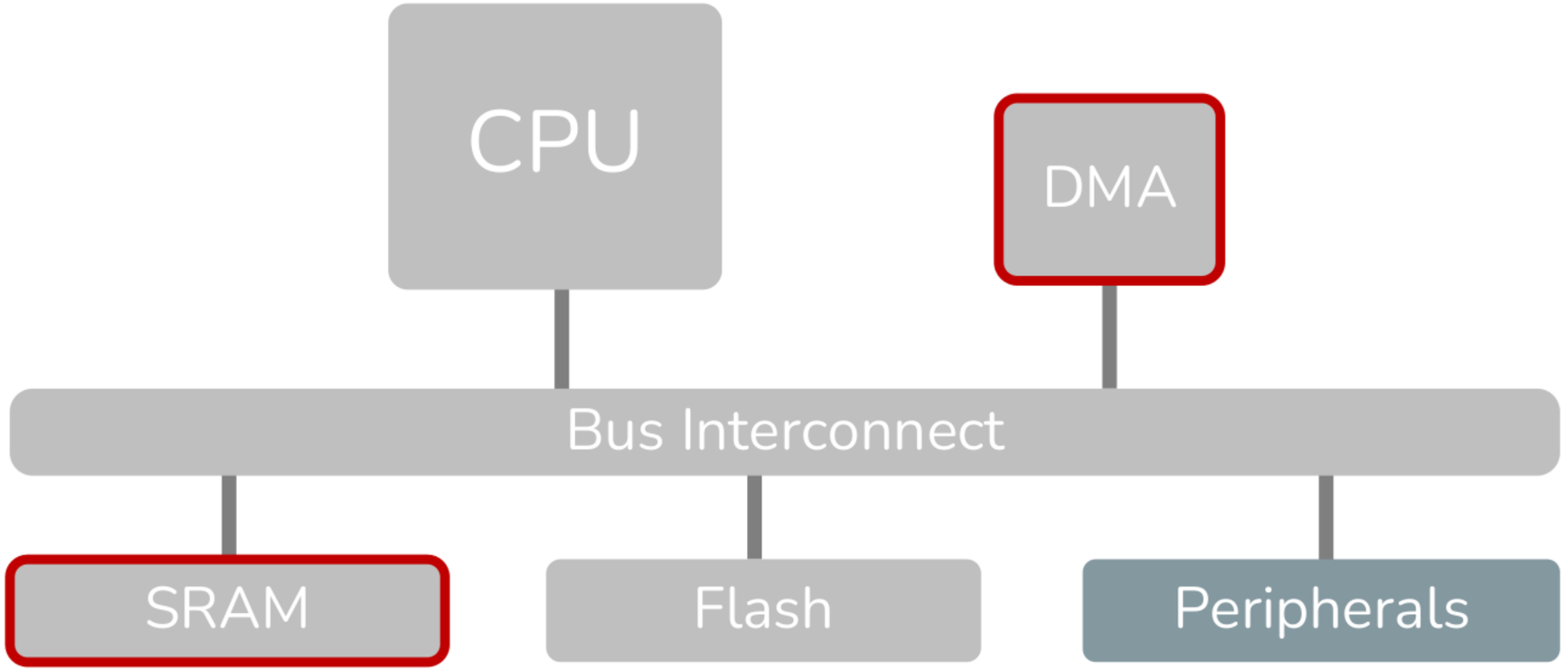
MCU Bus Interconnect as a Threat



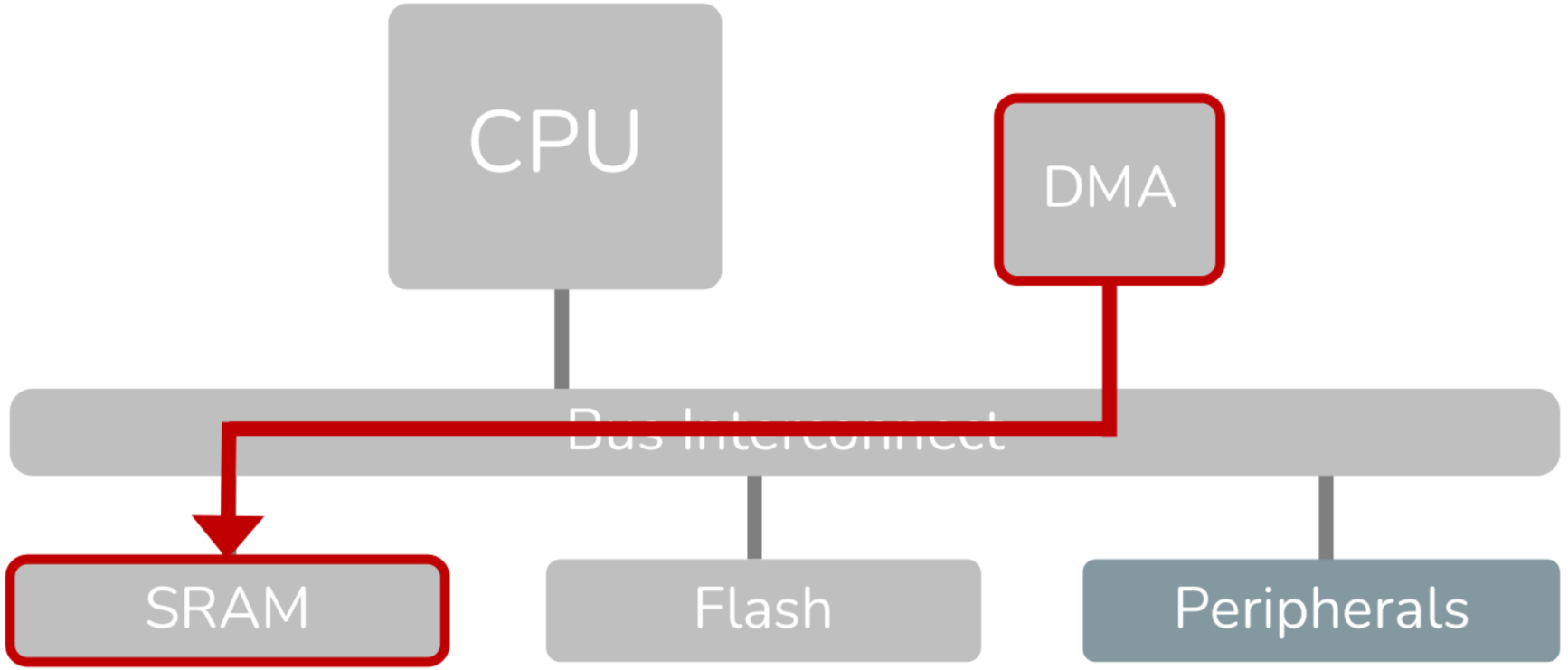
MCU Bus Interconnect as a Threat



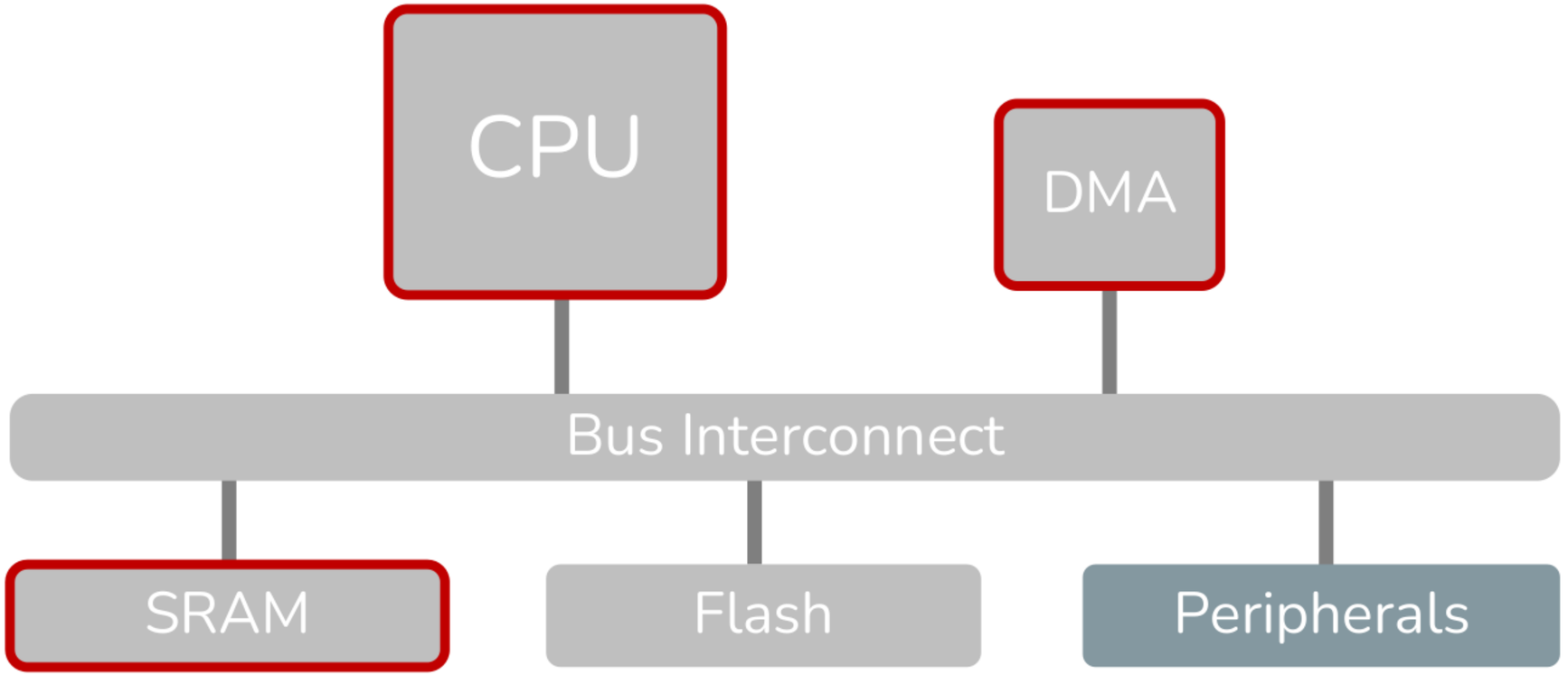
MCU Bus Interconnect as a Threat



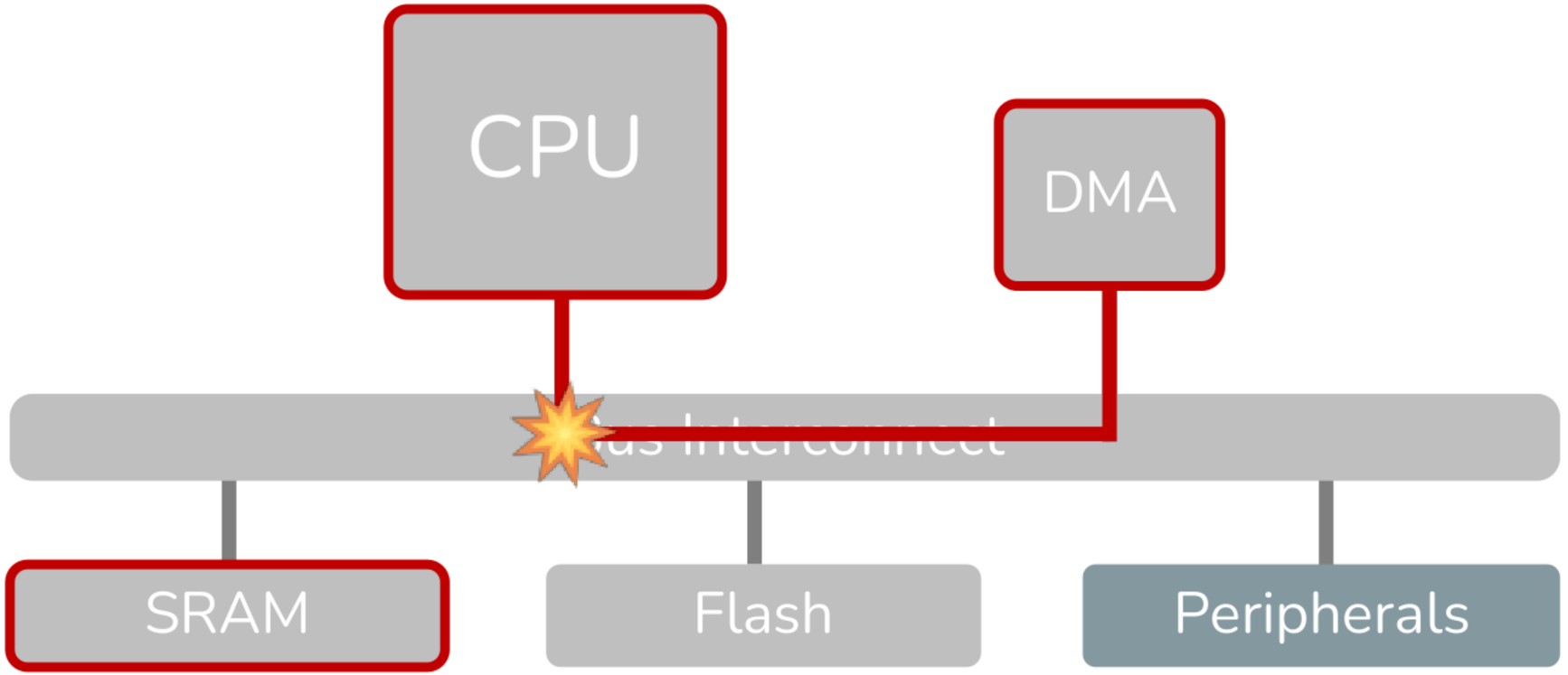
MCU Bus Interconnect as a Threat



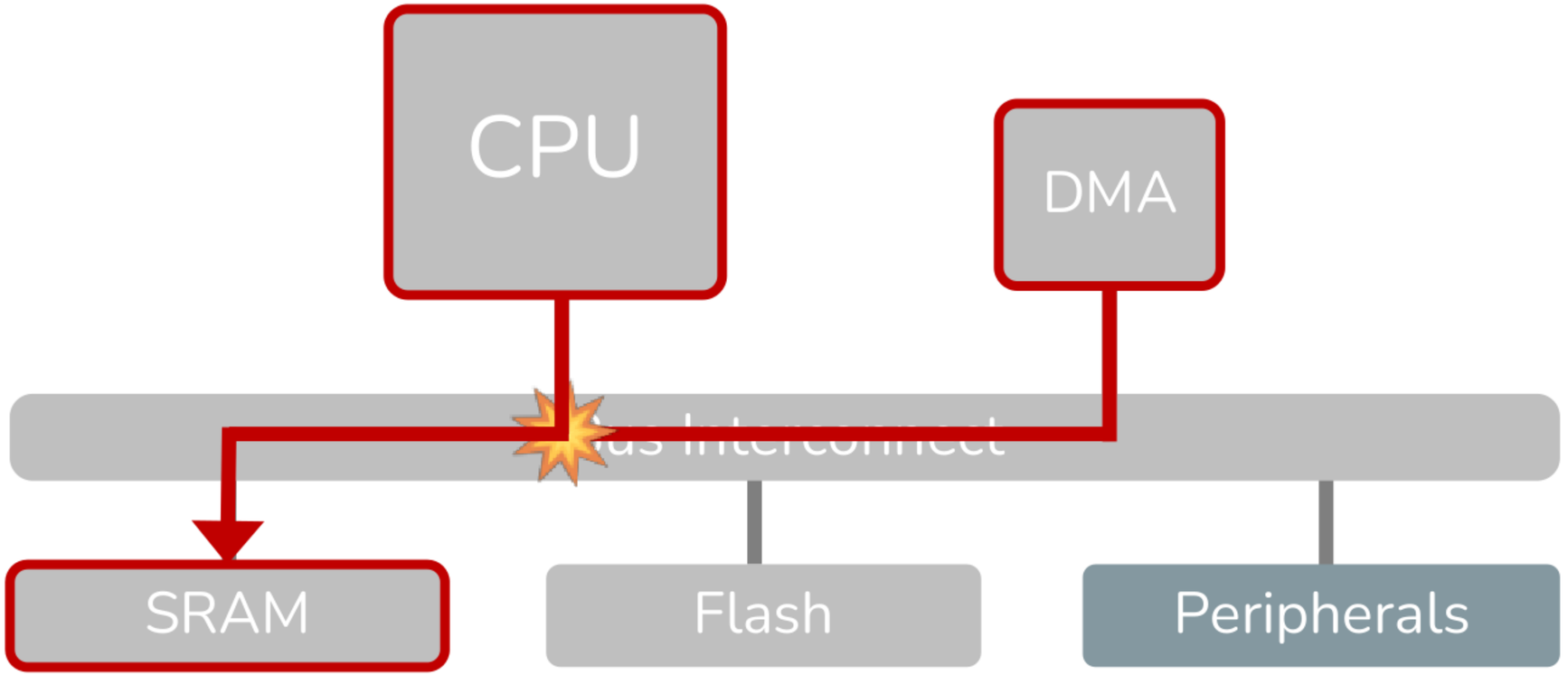
MCU Bus Interconnect as a Threat

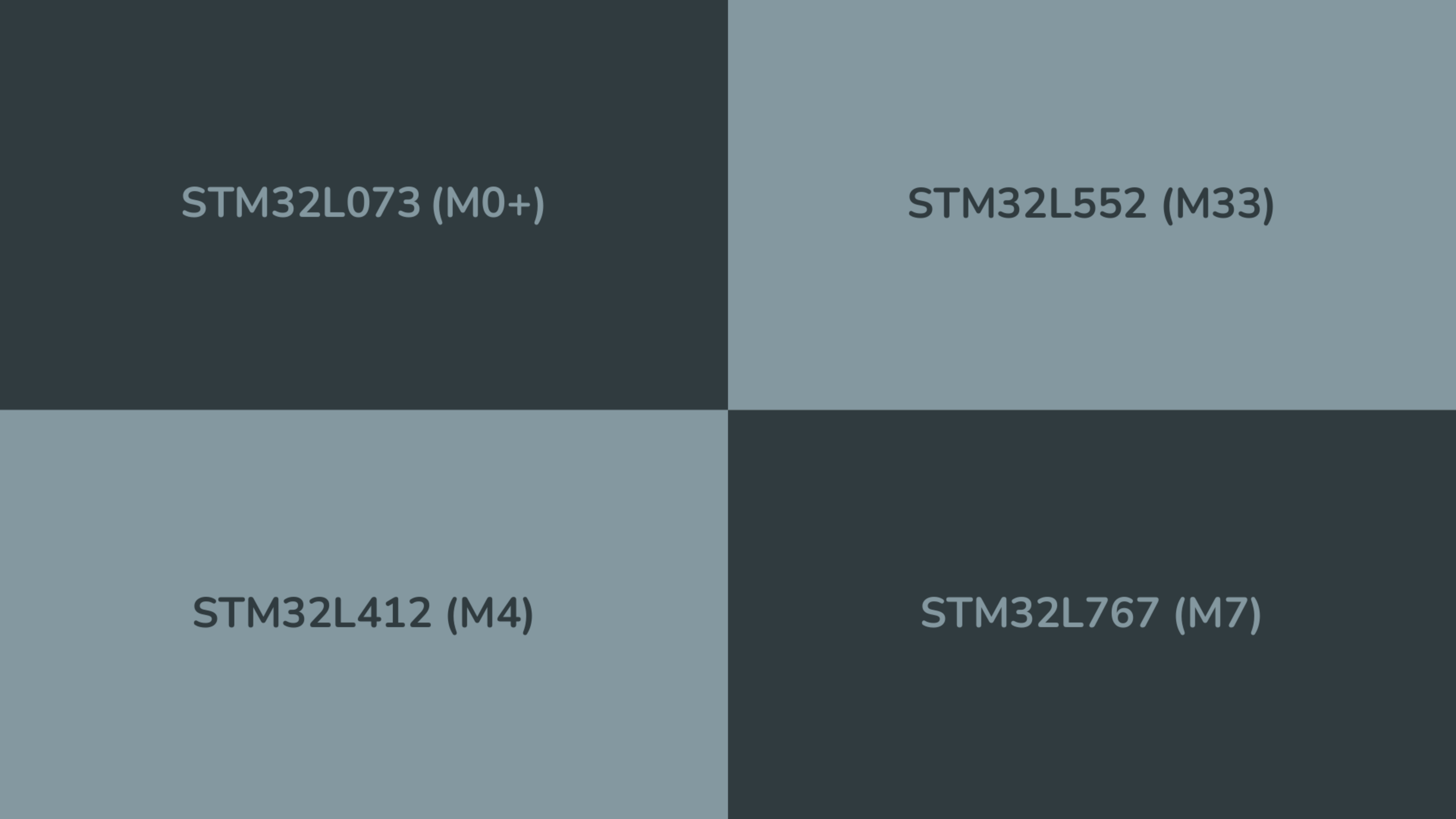


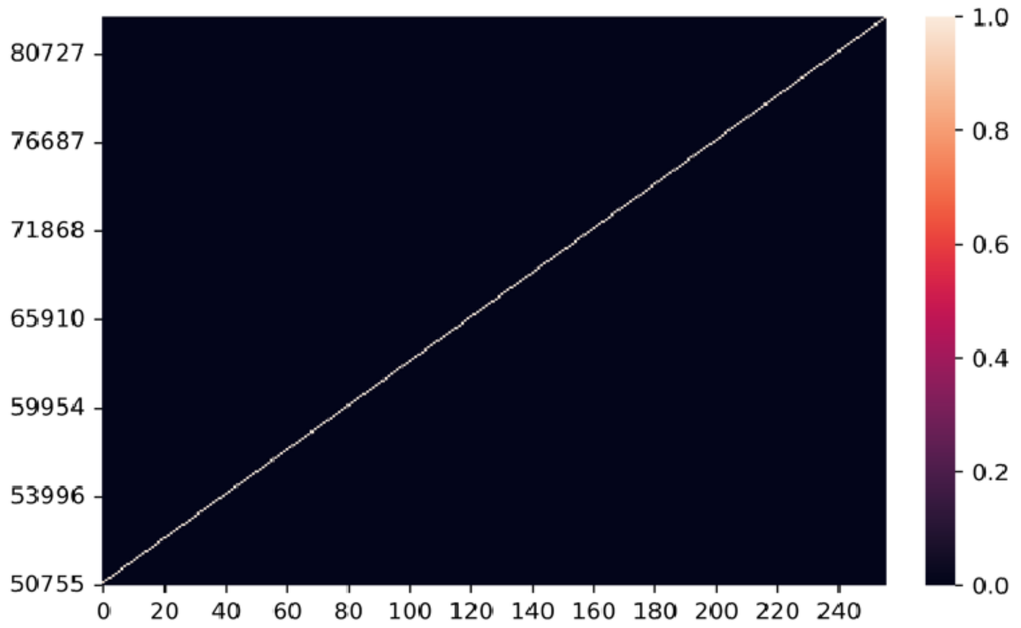
MCU Bus Interconnect as a Threat



MCU Bus Interconnect as a Threat



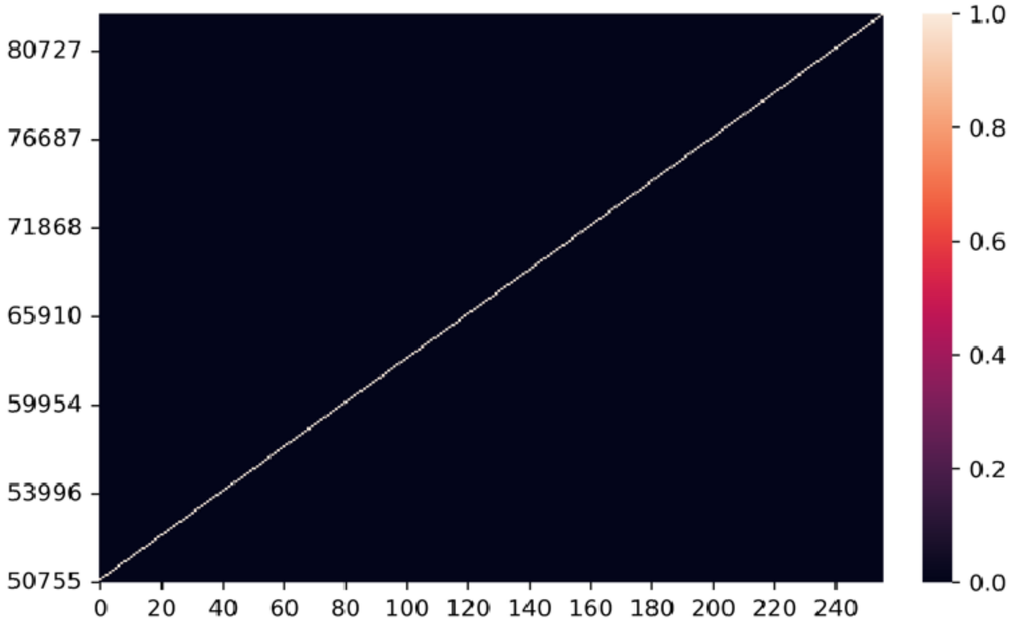

STM32L073 (M0+)
STM32L552 (M33)
STM32L412 (M4)
STM32L767 (M7)

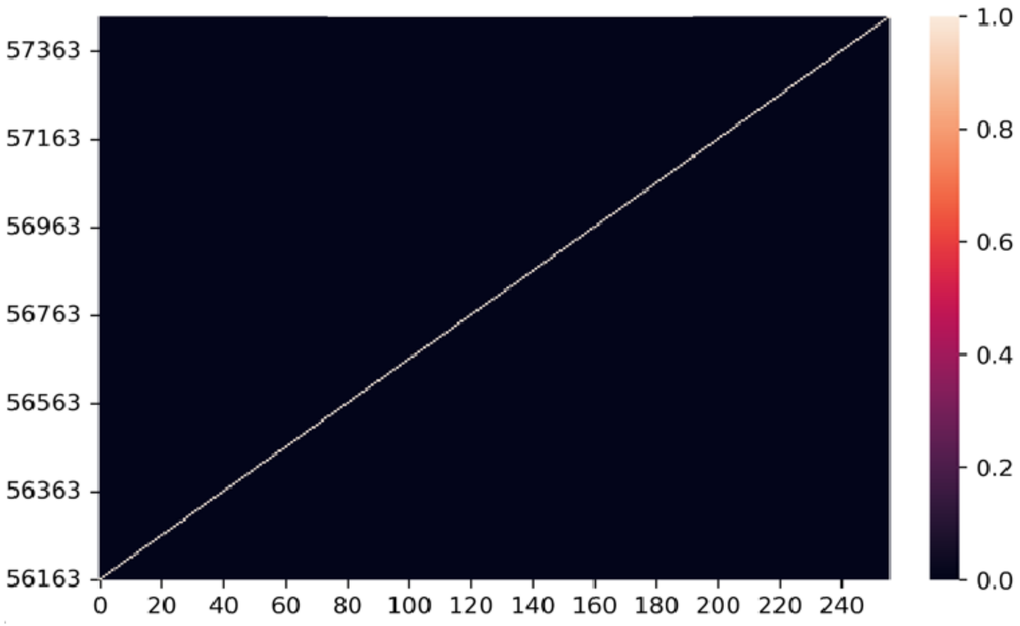


STM32L552 (M33)

STM32L412 (M4)

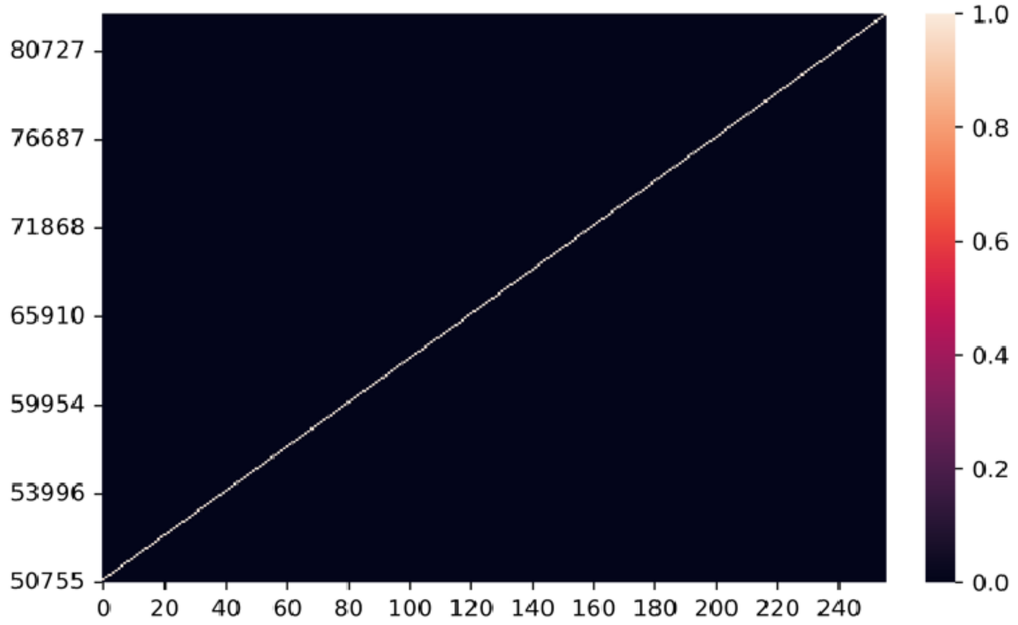
STM32L767 (M7)

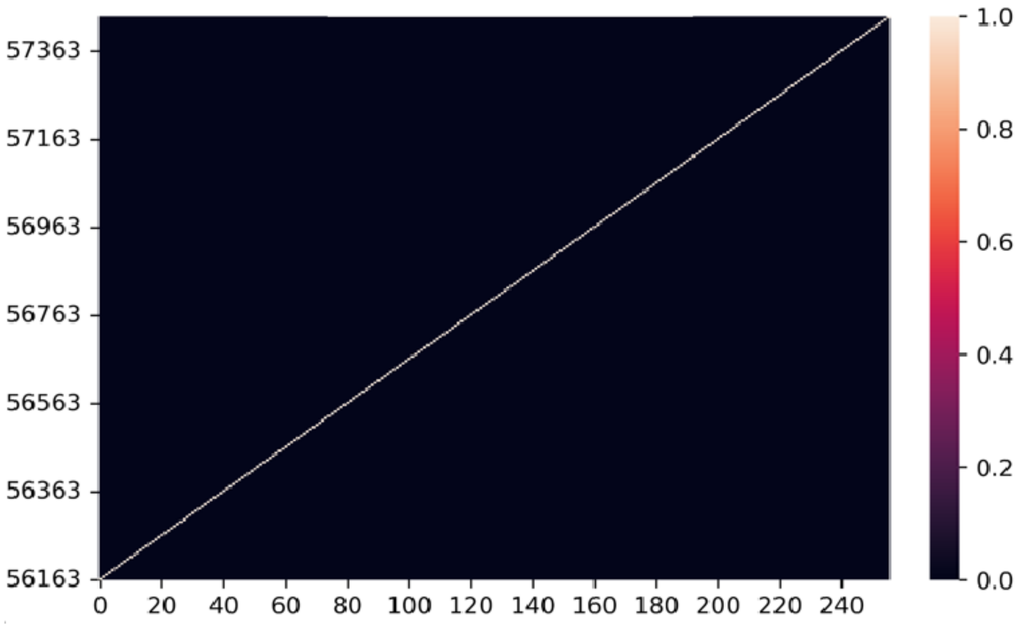


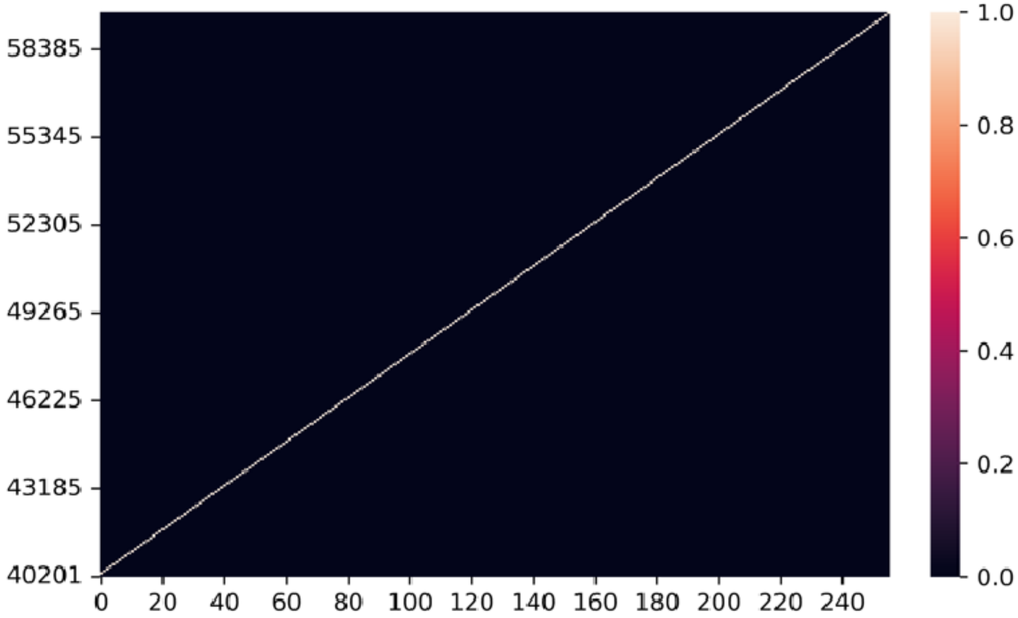


STM32L412 (M4)

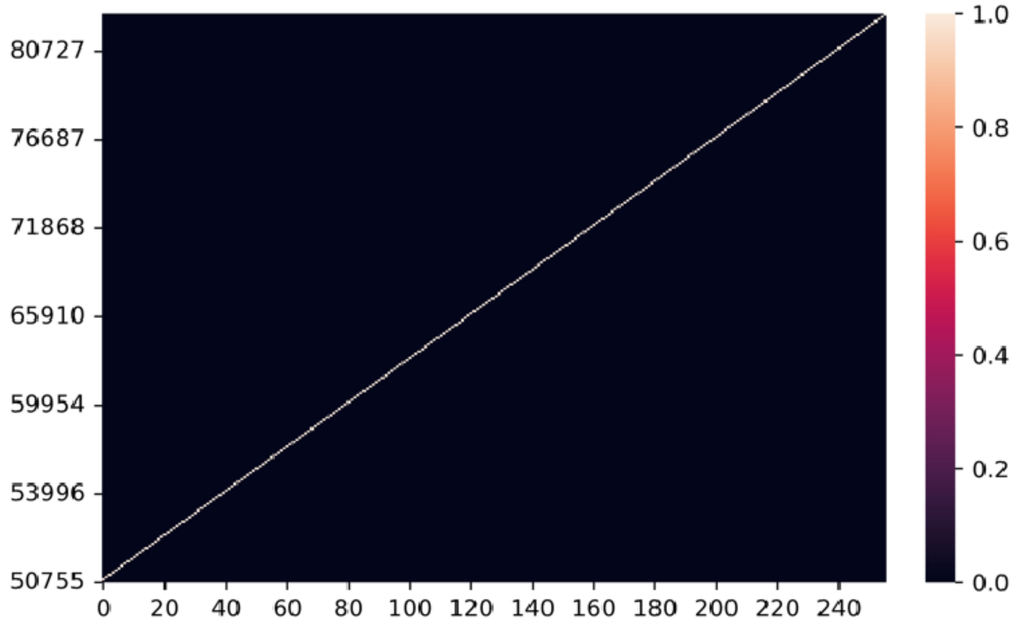
STM32L767 (M7)

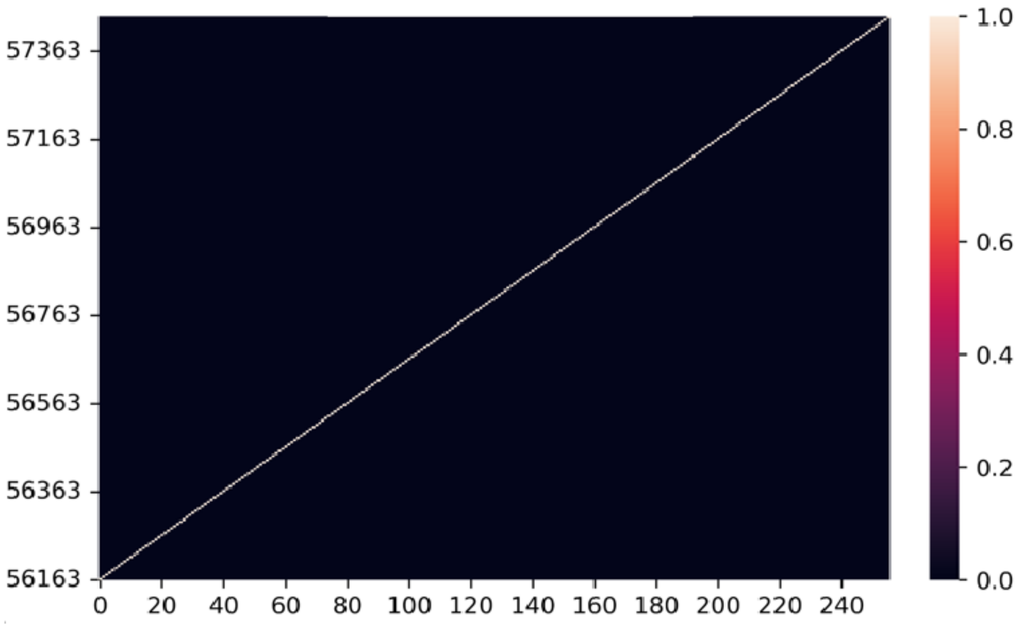


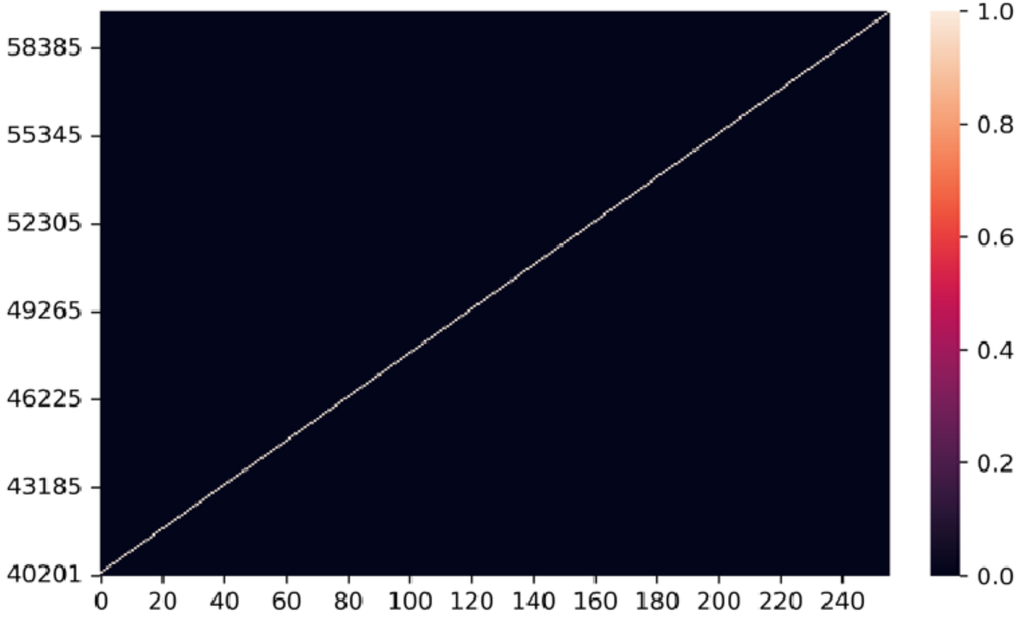


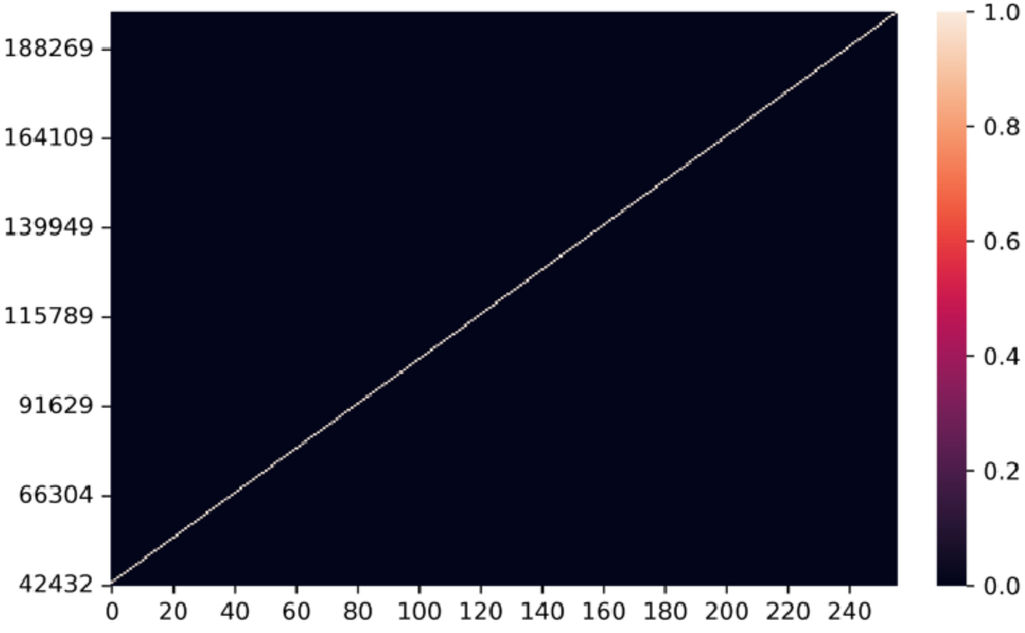


STM32L767 (M7)


80727
76687
71868
65910
59954
53996
50755
0 20 40 60 80 100 120 140 160 180 200 220 240
1.0
0.8
0.6
0.4
0.2
0.0

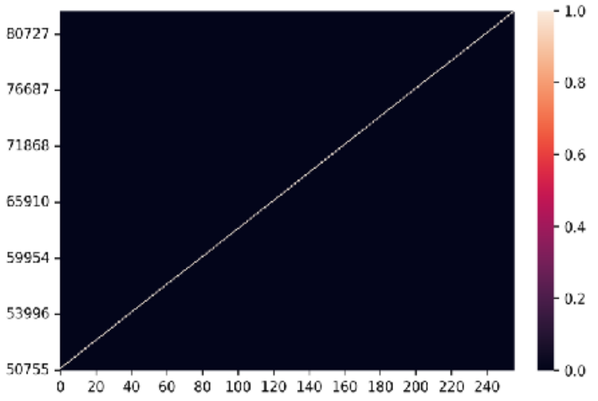

57363
57163
56963
56763
56563
56363
56163
0 20 40 60 80 100 120 140 160 180 200 220 240
1.0
0.8
0.6
0.4
0.2
0.0


58385
55345
52305
49265
46225
43185
40201
0 20 40 60 80 100 120 140 160 180 200 220 240
1.0
0.8
0.6
0.4
0.2
0.0

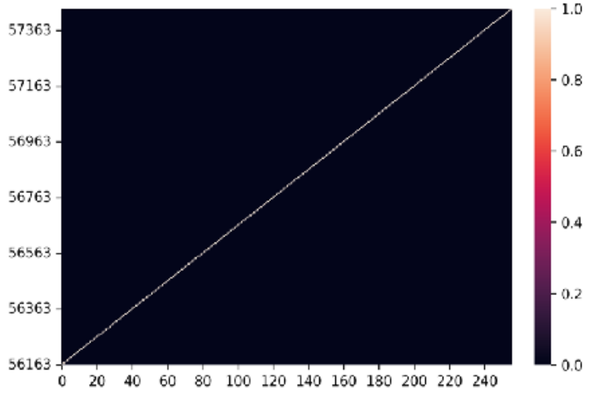

188269
164109
139949
115789
91629
66304
42432
0 20 40 60 80 100 120 140 160 180 200 220 240
1.0
0.8
0.6
0.4
0.2
0.0

MCU Channels

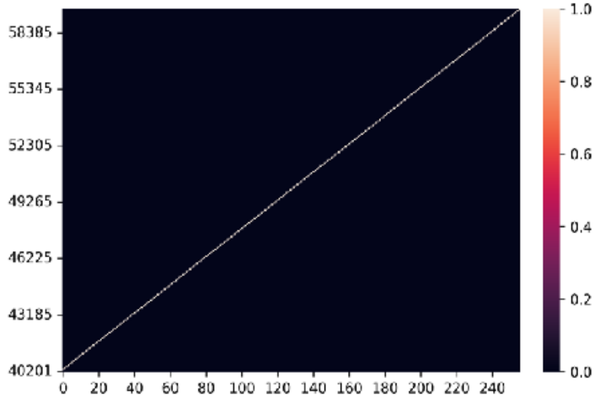
STM32L073 (M0+)



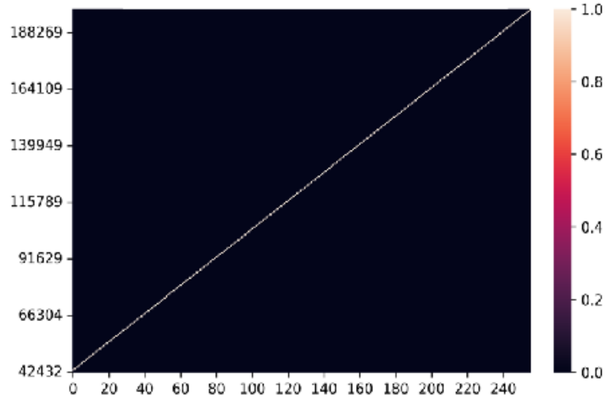
STM32L552 (M33)



STM32L412 (M4)

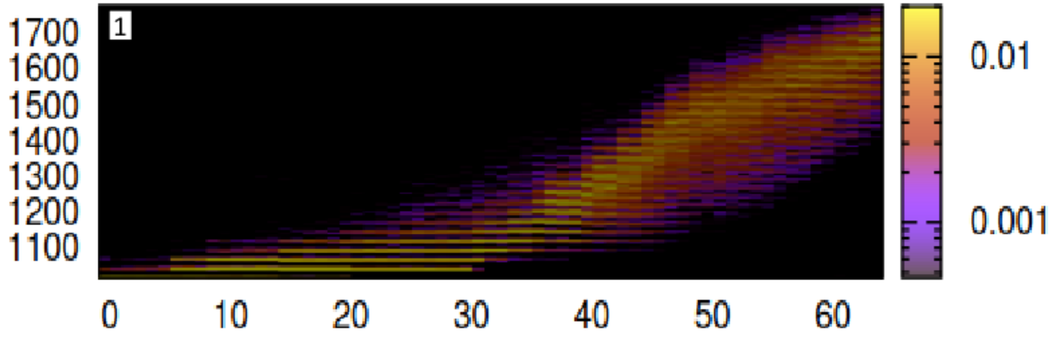


STM32L767 (M7)

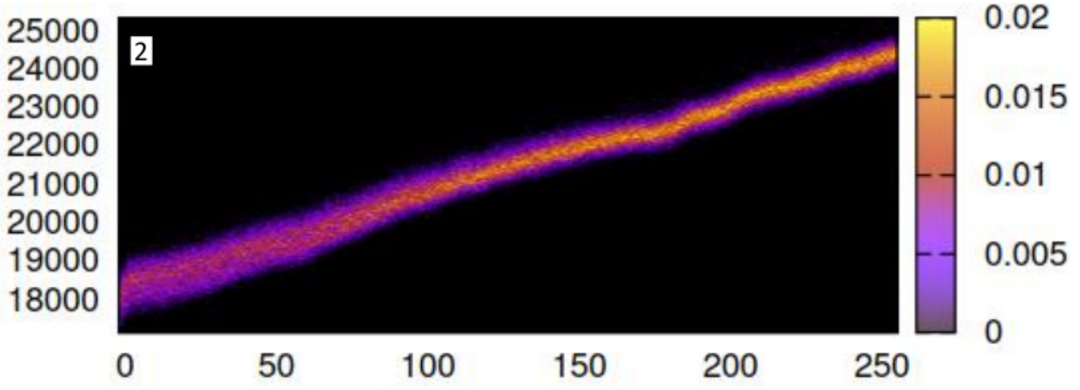


APU Channels

Skylake TLB Channel



Arm Cortex-A9 L1-I Channel

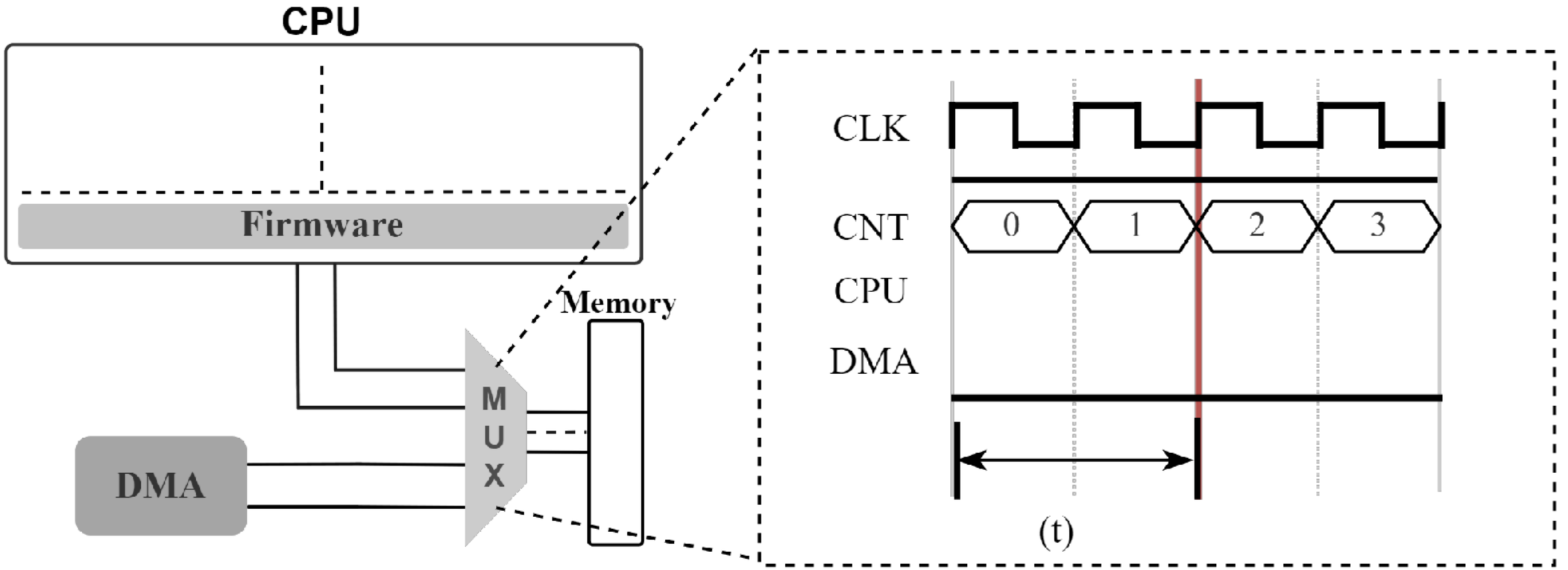


[1,2] – "Your Processor Leaks Information – and There's Nothing You Can Do About It", Ge et al.

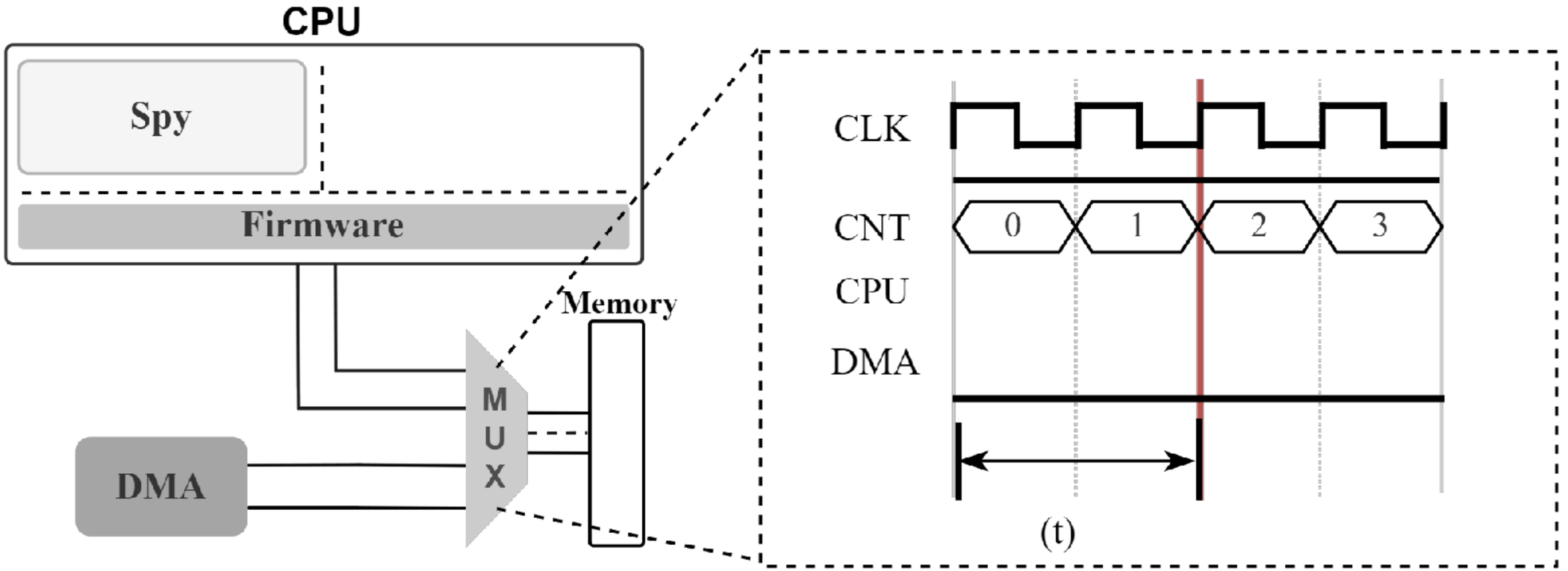
“Toy” Attack

Basic Attack Example

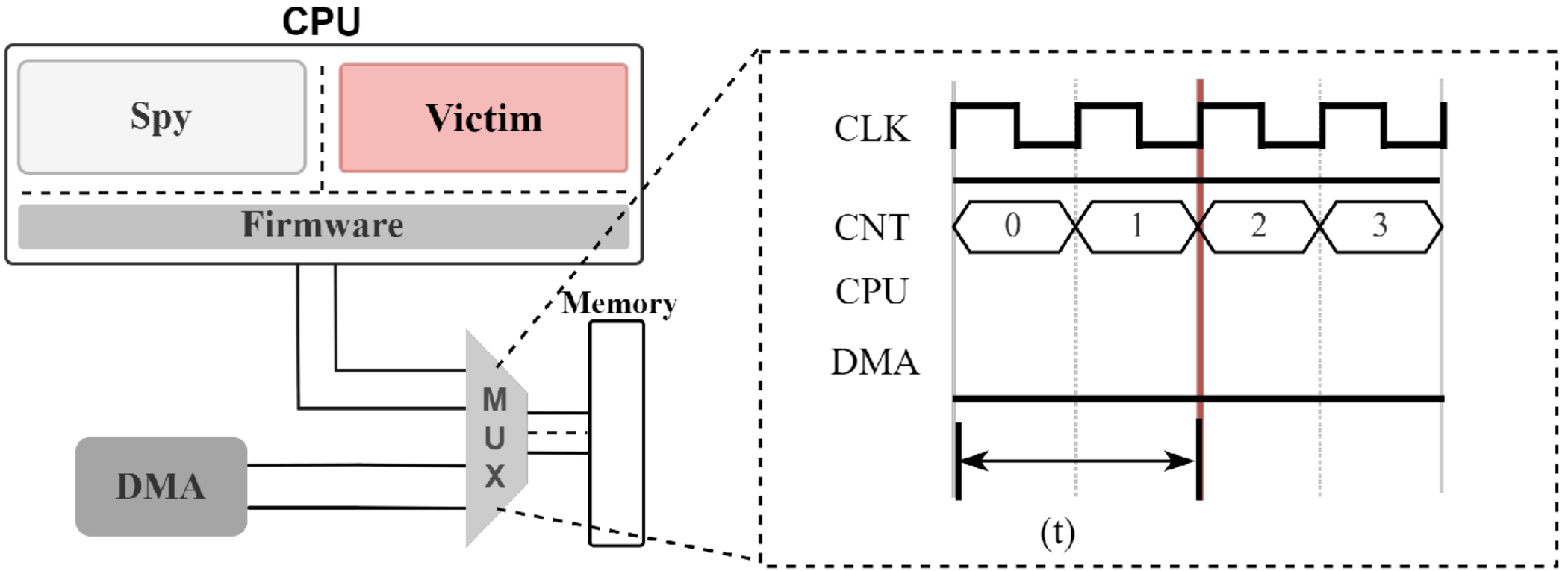
Attack Overview – The Basics



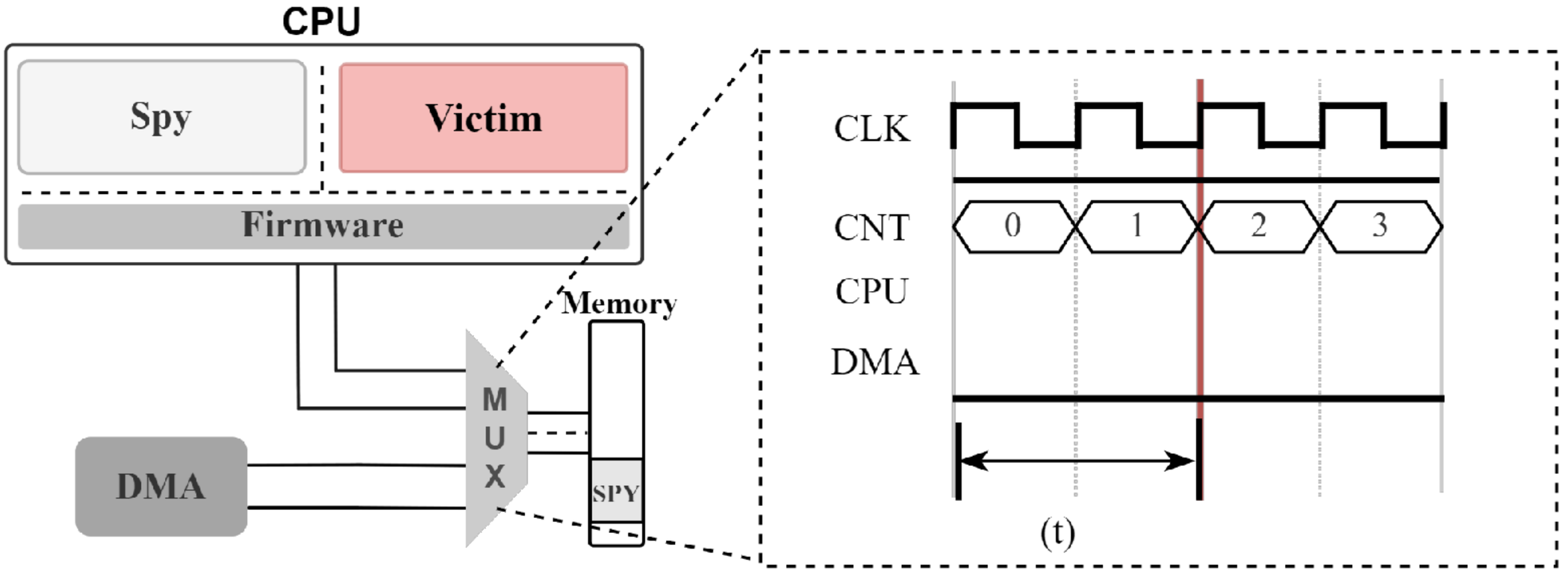
Attack Overview – The Basics



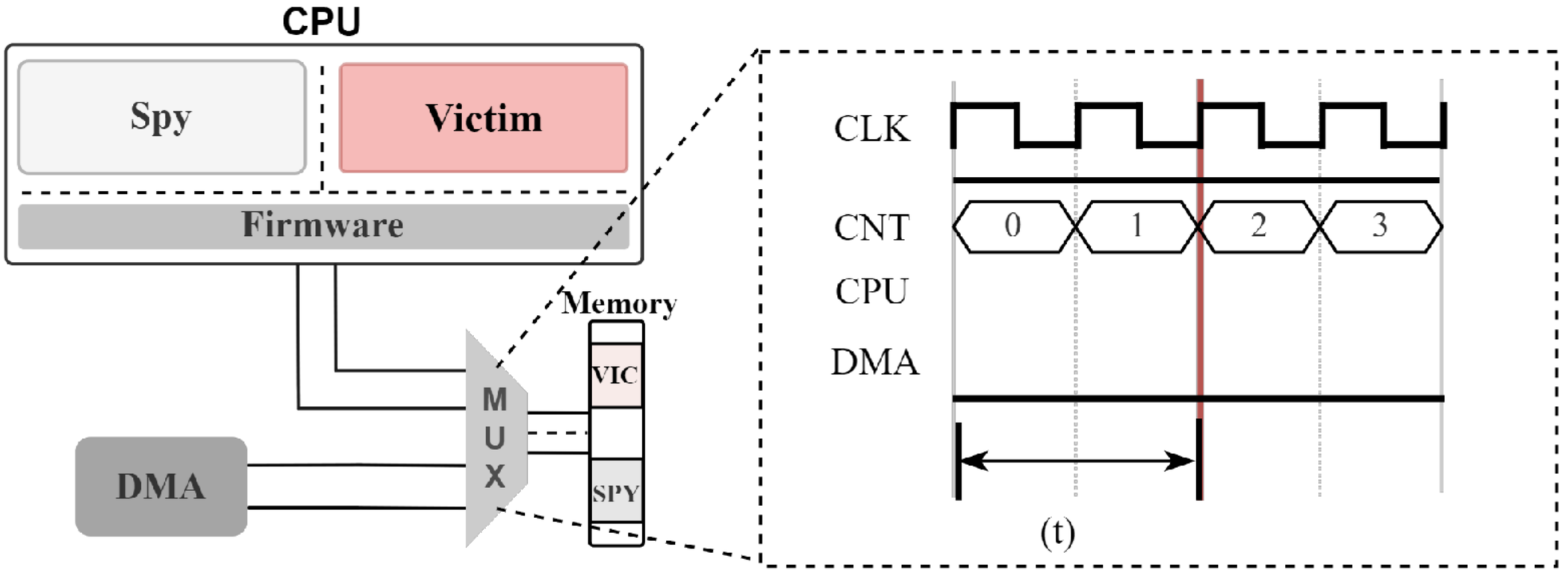
Attack Overview – The Basics



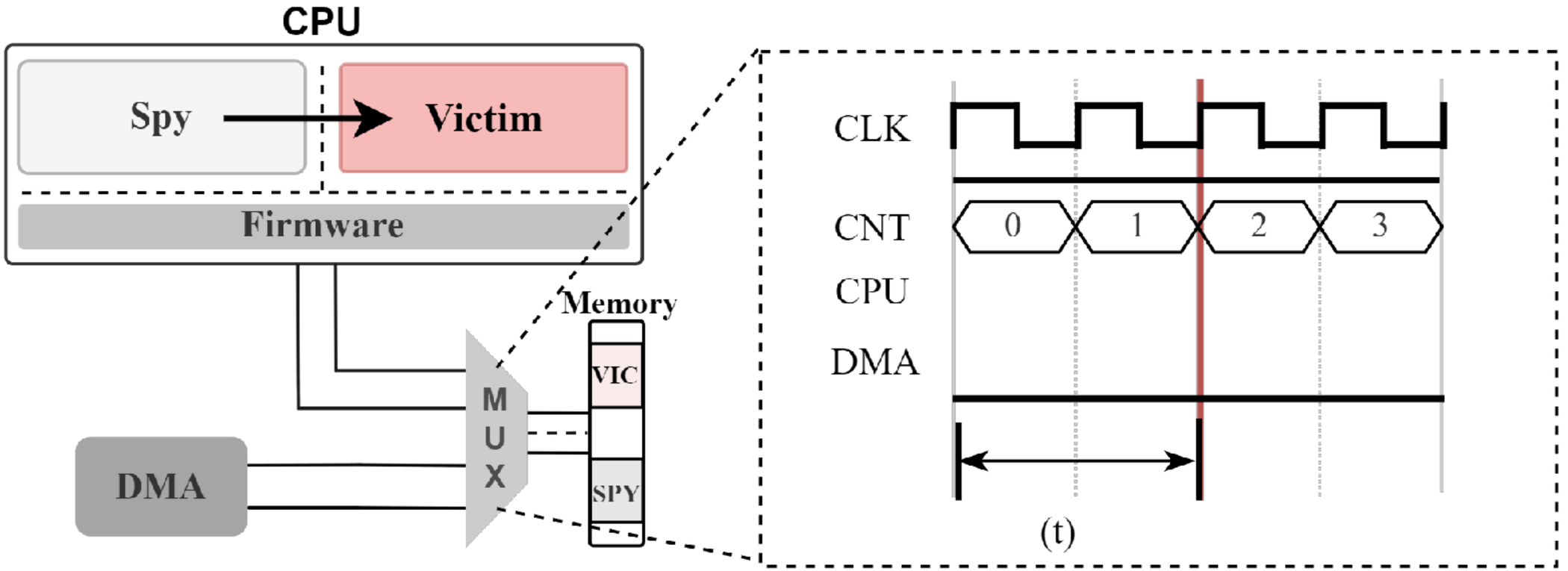
Attack Overview – The Basics



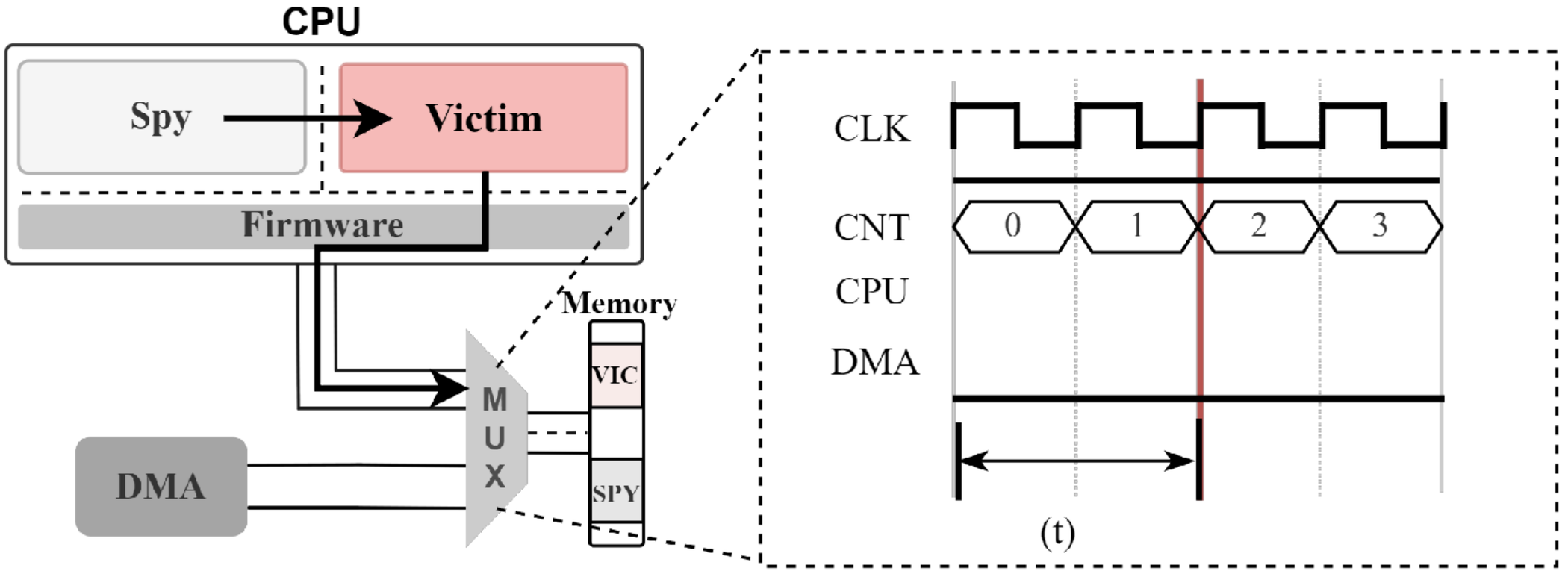
Attack Overview – The Basics



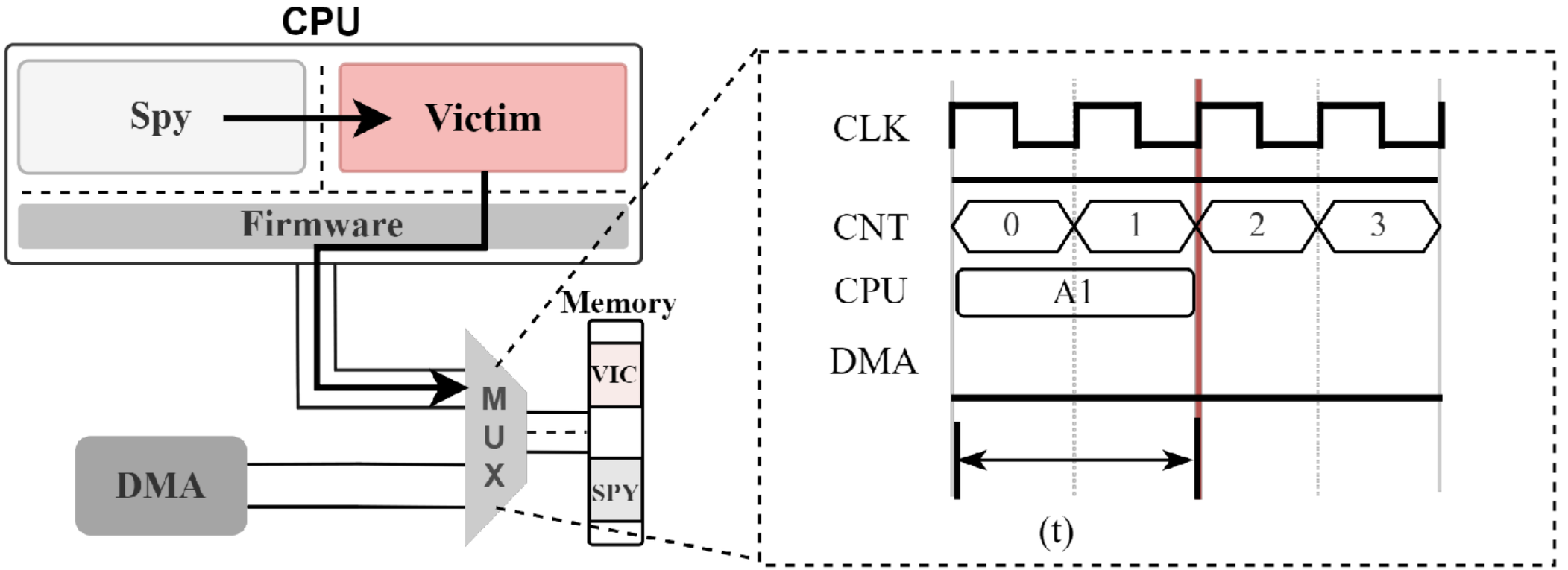
Attack Overview – The Basics



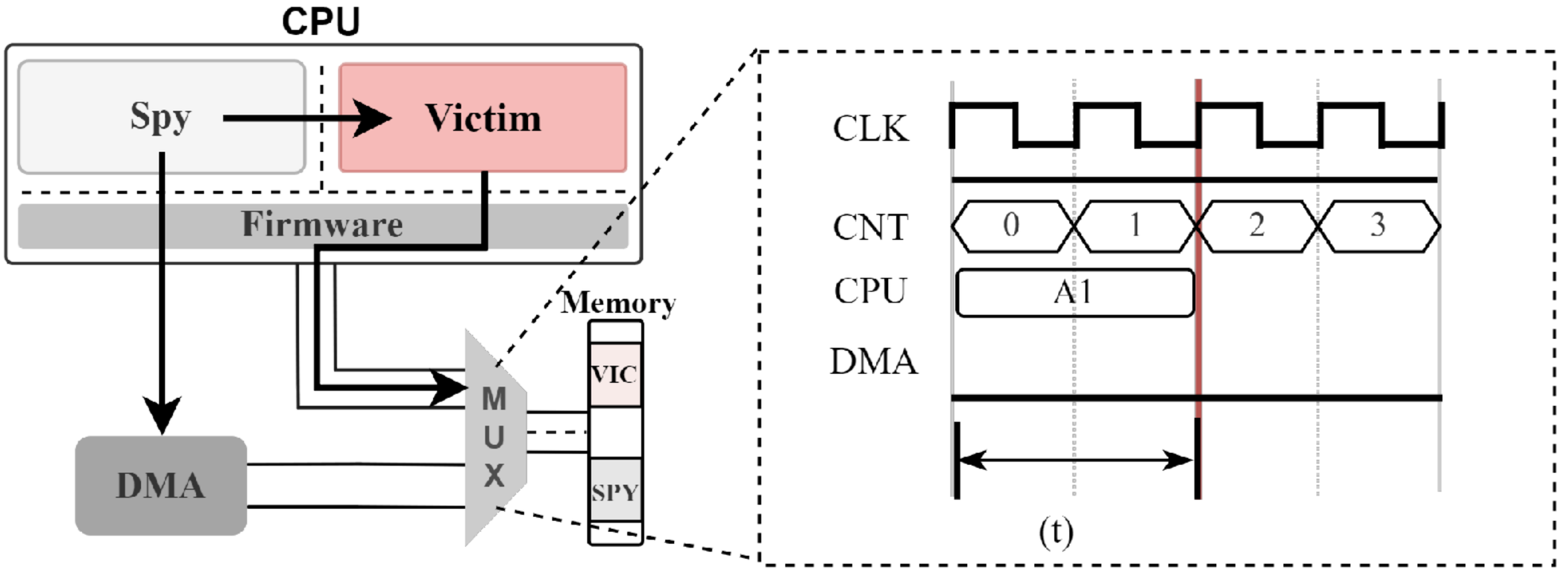
Attack Overview – The Basics



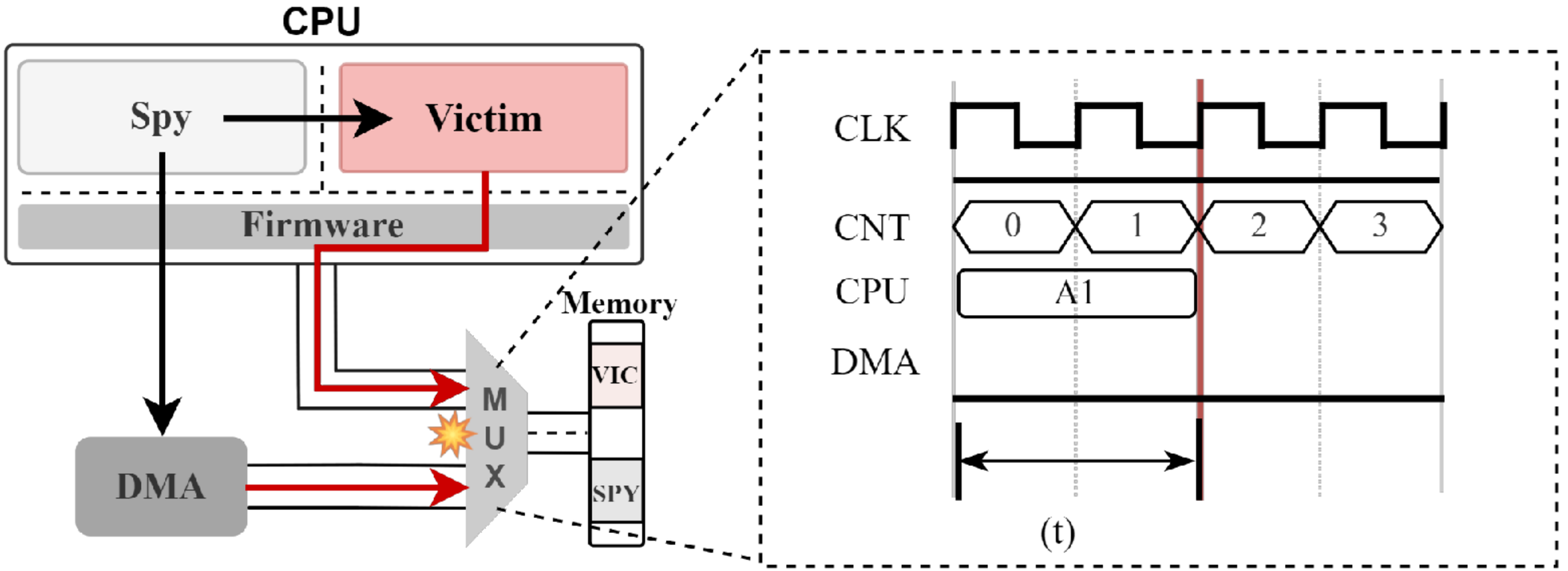
Attack Overview – The Basics



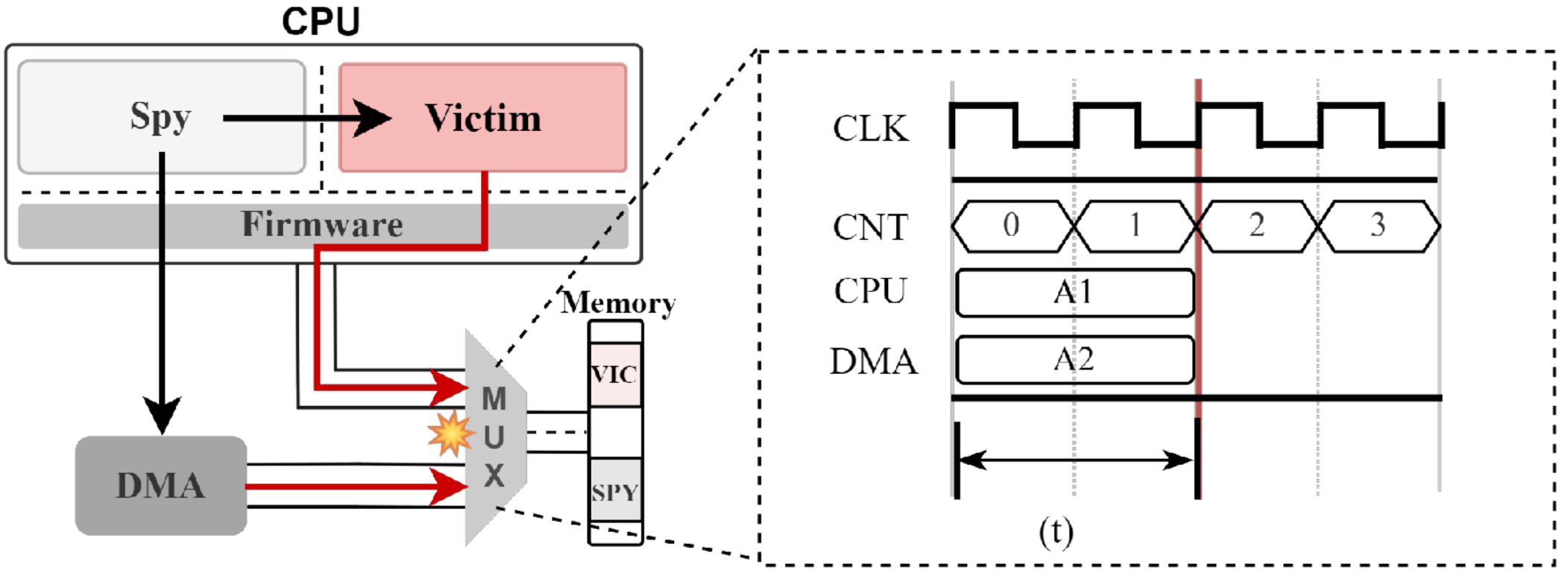
Attack Overview – The Basics



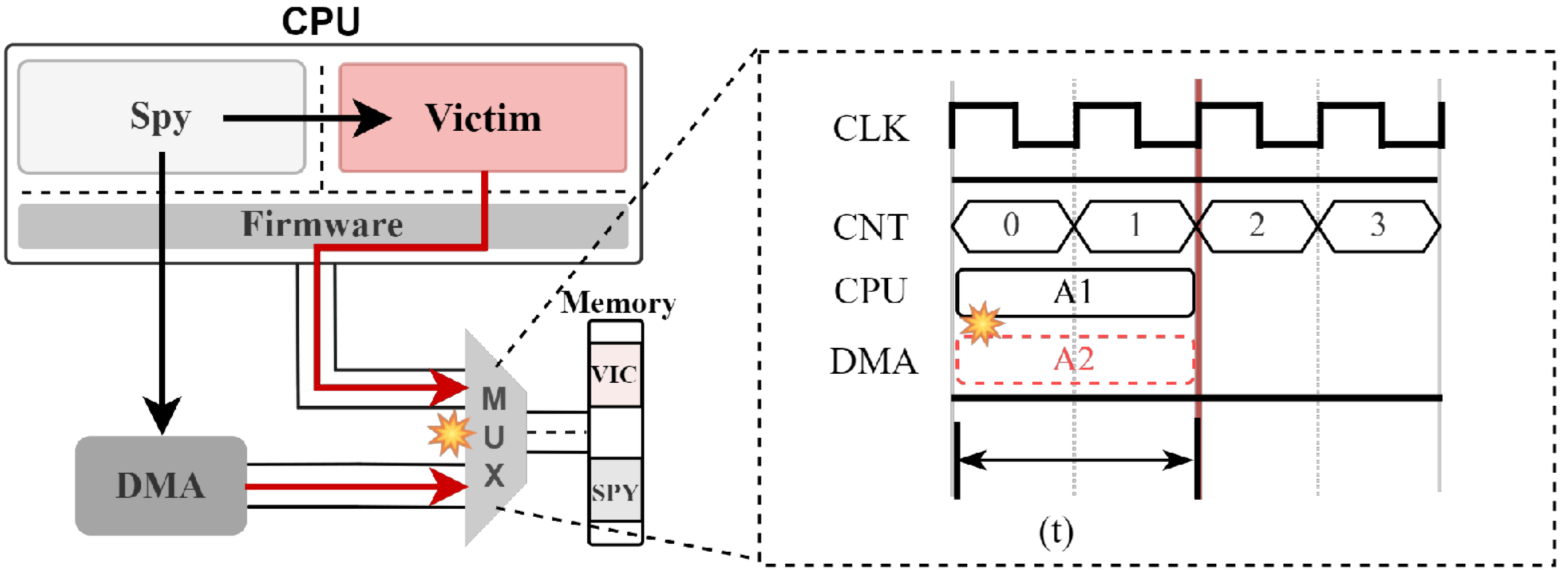
Attack Overview – The Basics



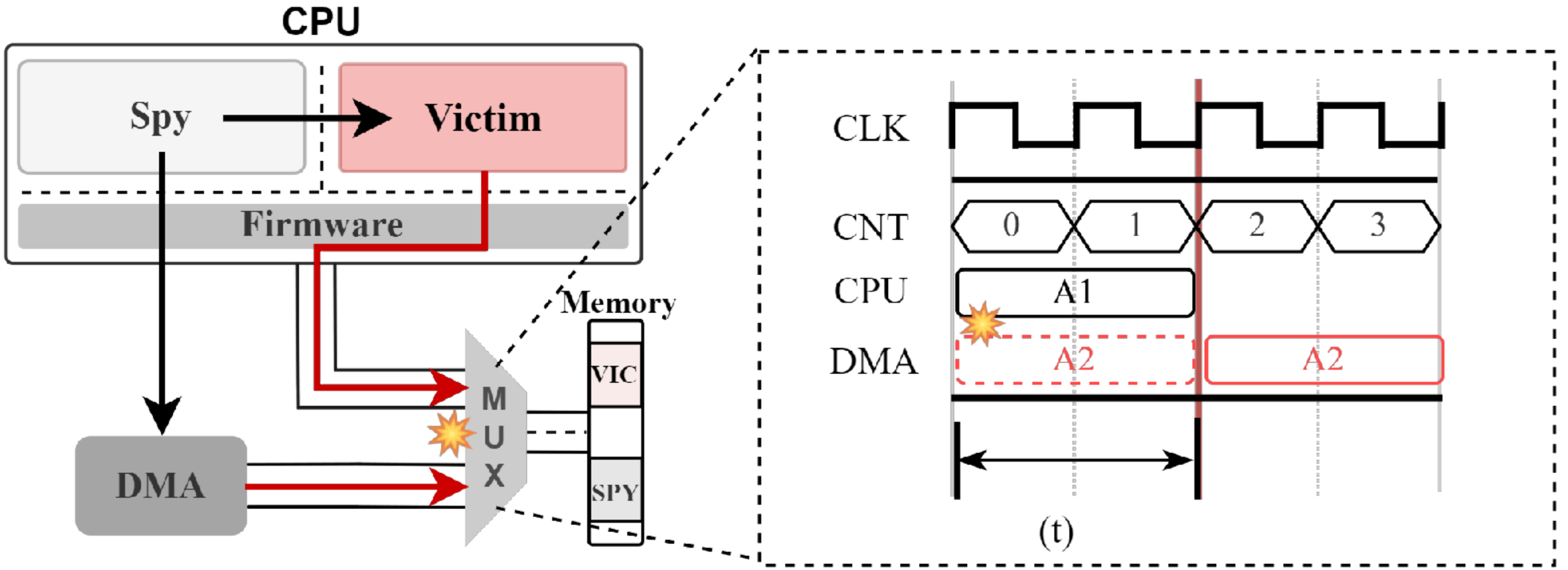
Attack Overview – The Basics



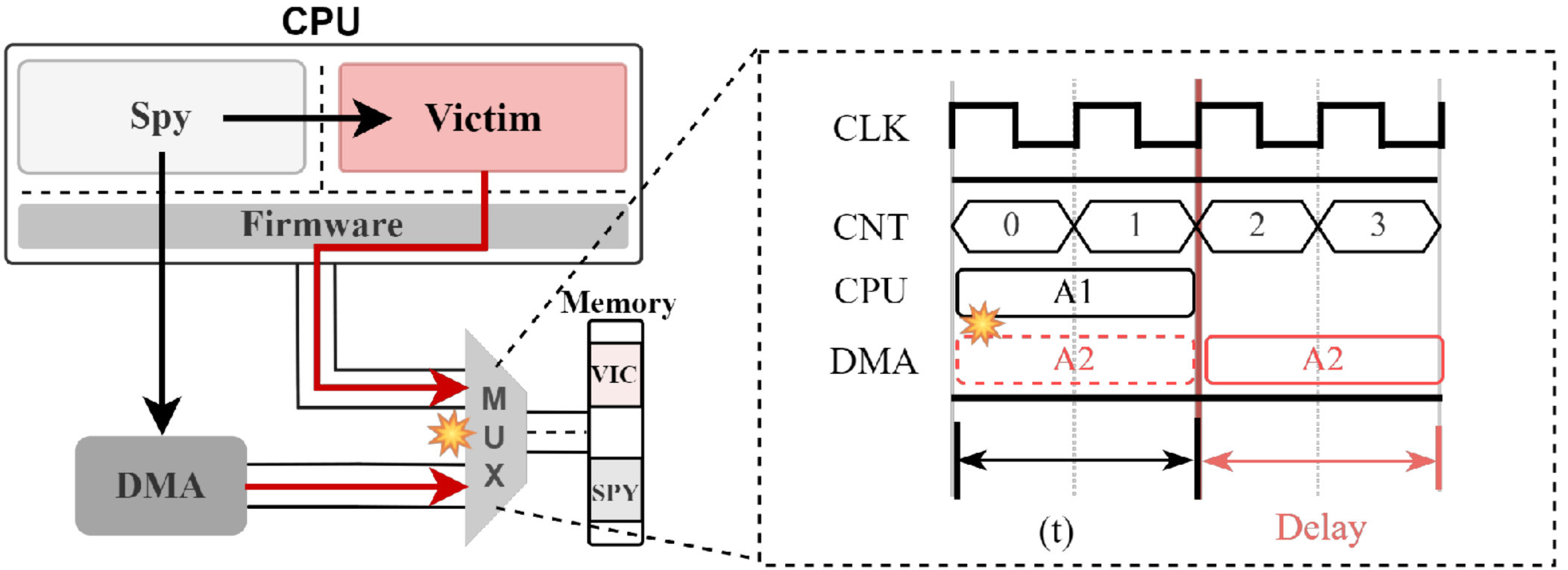
Attack Overview – The Basics



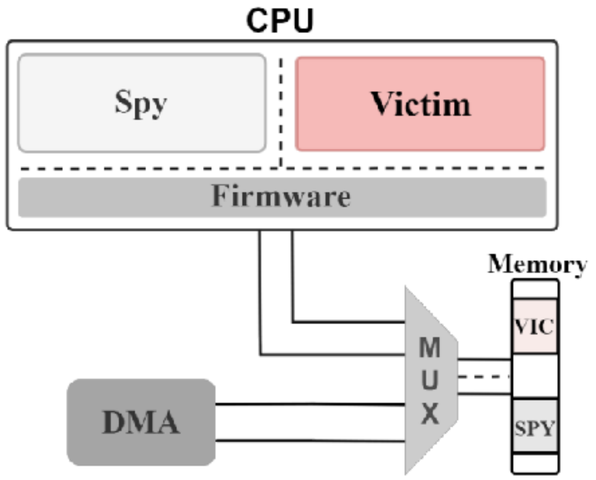
Attack Overview – The Basics



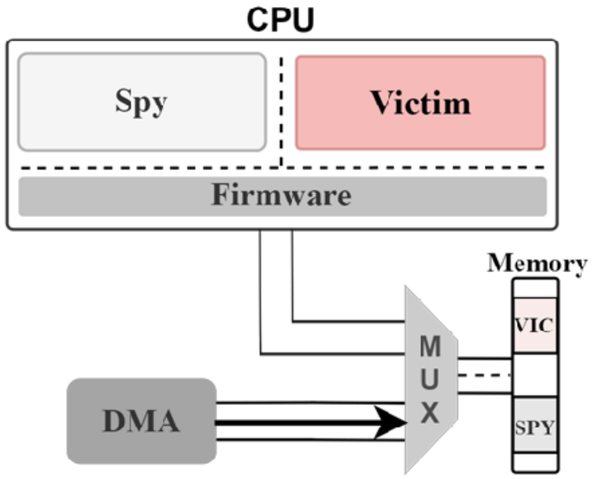
Attack Overview – The Basics



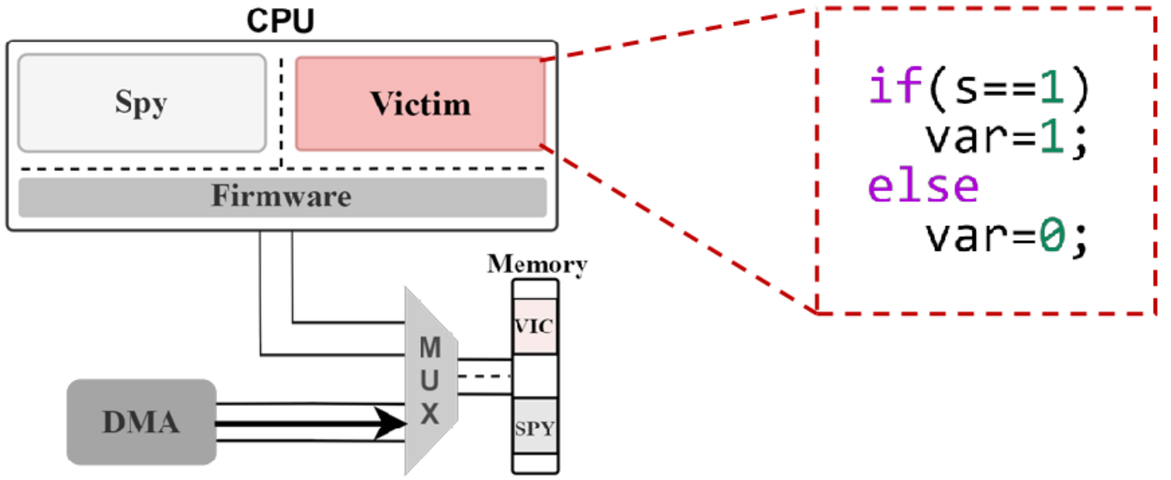
Attack Overview – Toy Example



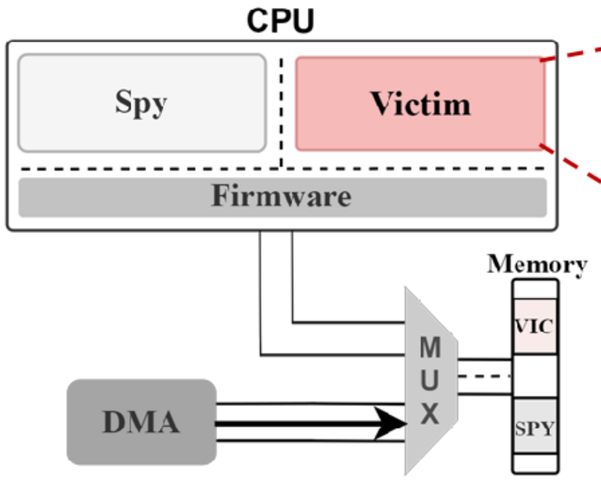
Attack Overview – Toy Example



Attack Overview – Toy Example



Attack Overview – Toy Example

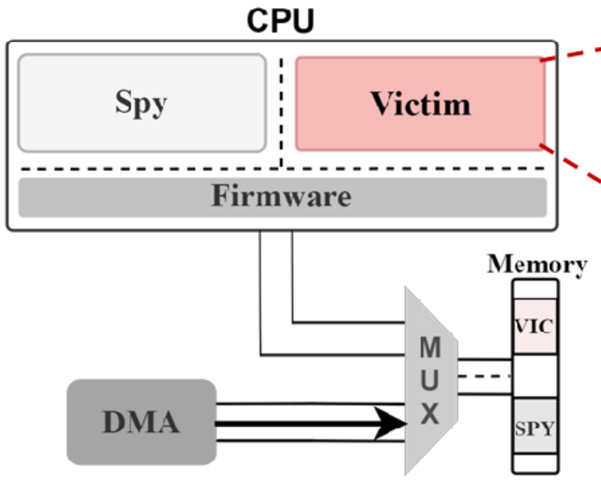



```
if(s==1)
  var=1;
else
  var=0;
```

```
      cmp   r3, #0
      beq.n ELSE
IF:   movs  r3, #1
      str   r3, [r7, #0]
      b.n   END
ELSE: movs  r3, #0
      str   r3, [r7, #0]
END:  nop
```

If-else statement compiled for Arm Cortex-M33 (-O0)

Attack Overview – Toy Example

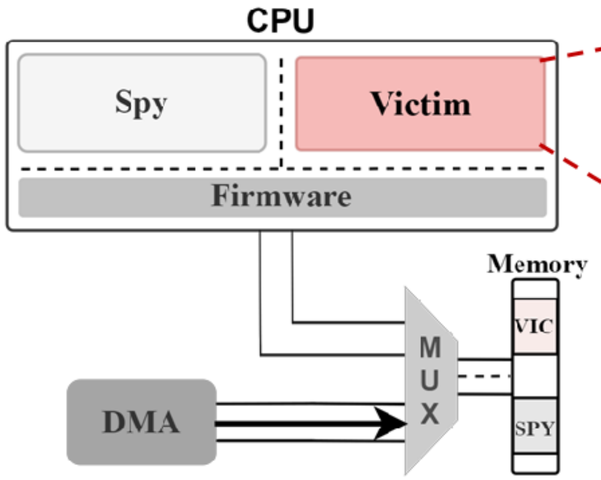



```
if(s==1)
  var=1;
else
  var=0;
```

```
        cmp   r3, #0
        beq.n ELSE
IF:     movs  r3, #1          1 clk
        str   r3, [r7, #0]    1 clk
        b.n   END             2 clk
ELSE:   movs  r3, #0
        str   r3, [r7, #0]
END:    nop                   1 clk
```

If-else statement compiled for Arm Cortex-M33 (-O0)

Attack Overview – Toy Example

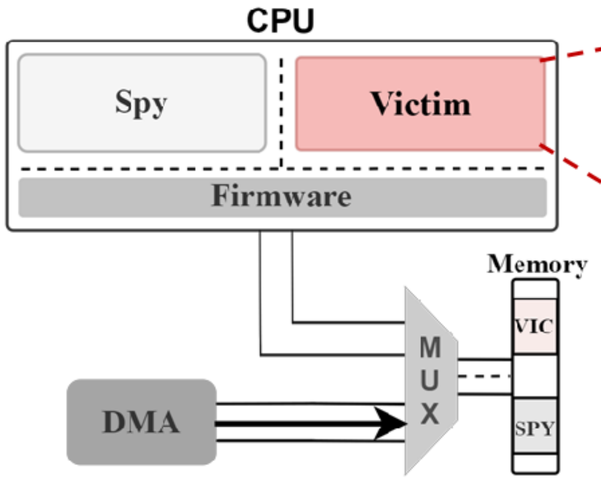



```
if(s==1)
  var=1;
else
  var=0;
```

```
        cmp   r3, #0
        beq.n ELSE                    3 clk (else), 1 clk (if)
IF:     movs  r3, #1
        str   r3, [r7, #0]
        b.n   END
ELSE:   movs  r3, #0                  1 clk
        str   r3, [r7, #0]            1 clk
END:    nop
```

If-else statement compiled for Arm Cortex-M33 (-O0)

Attack Overview – Toy Example

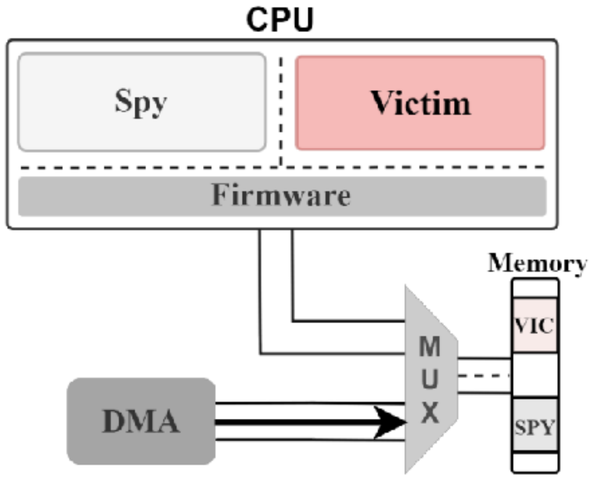



```
if(s==1)
  var=1;
else
  var=0;
```

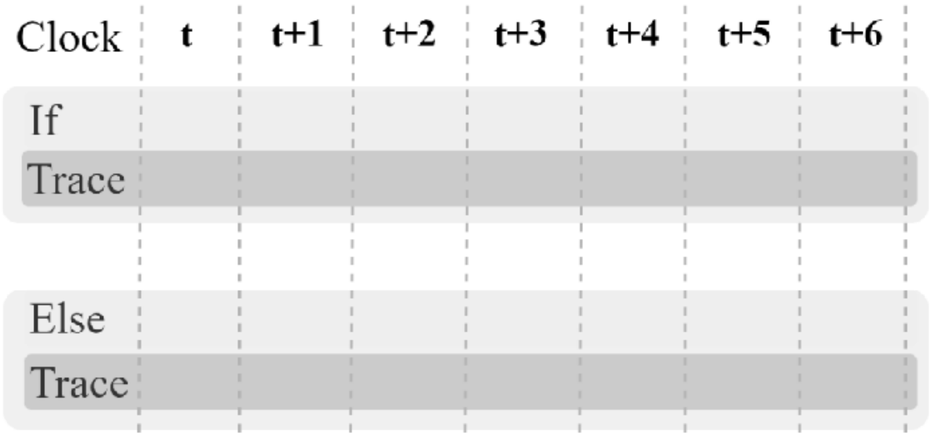
```
      cmp   r3, #0                          1 clk
      beq.n ELSE                  3 clk (else), 1 clk (if)
IF:   movs  r3, #1                          1 clk
      str   r3, [r7, #0]                    1 clk
      b.n   END                             2 clk
ELSE: movs  r3, #0                          1 clk
      str   r3, [r7, #0]                    1 clk
END:  nop                                   1 clk
```

If-else statement compiled for Arm Cortex-M33 (-O0)

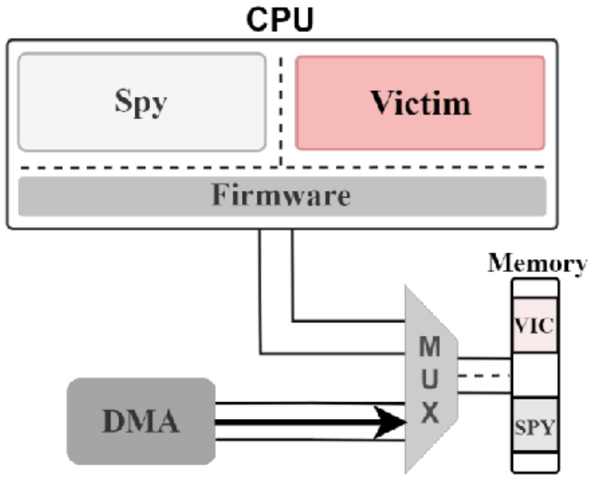
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```



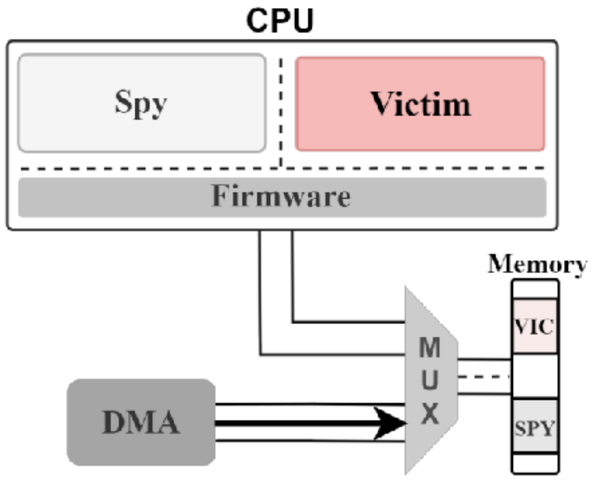
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp						
Trace	---						
Else							
Trace							

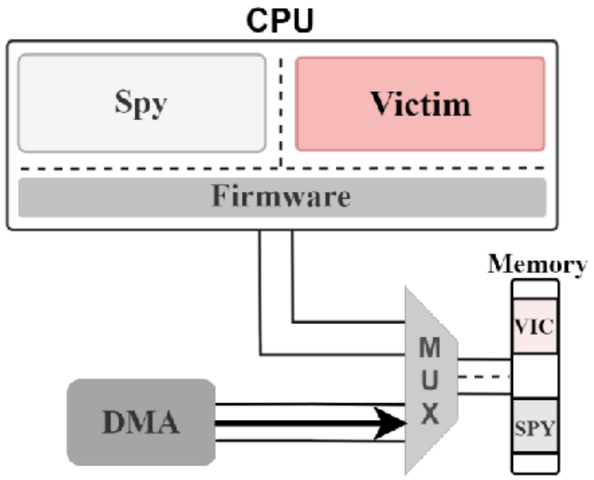
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq					
Trace	---	---					
Else							
Trace							

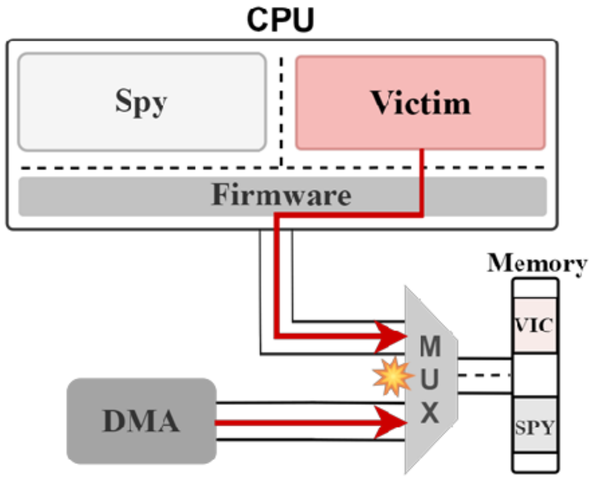
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs				
Trace	---	---	---				
Else							
Trace							

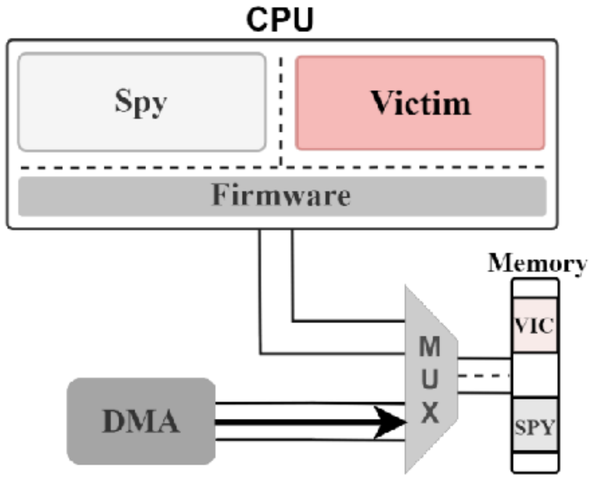
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str			
Trace	---	---	---	X			
Else							
Trace							

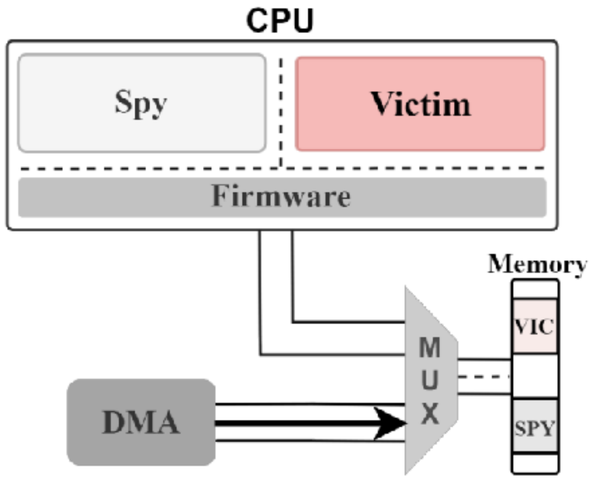
Attack Overview – Toy Example




```
        cmp   r3, #0                  1 clk
        beq.n ELSE                    3 clk (else), 1 clk (if)
IF:     movs  r3, #1                  1 clk
        str   r3, [r7, #0]            1 clk
        b.n   END                     2 clk
ELSE:   movs  r3, #0                  1 clk
        str   r3, [r7, #0]            1 clk
END:    nop                           1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b		
Trace	---	---	---	X	---		
Else							
Trace							

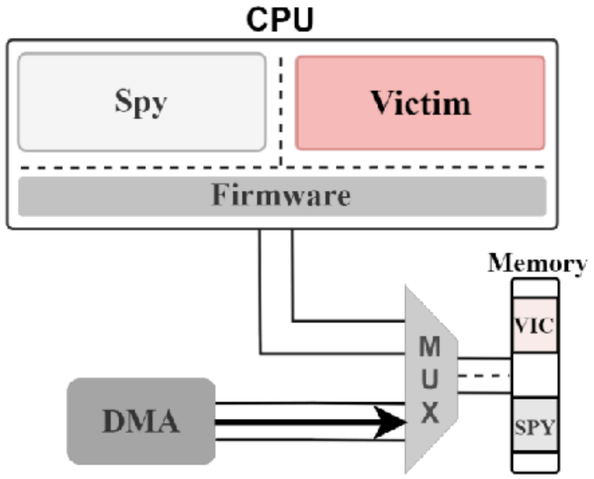
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	
Trace	---	---	---	X	---	---	
Else							
Trace							

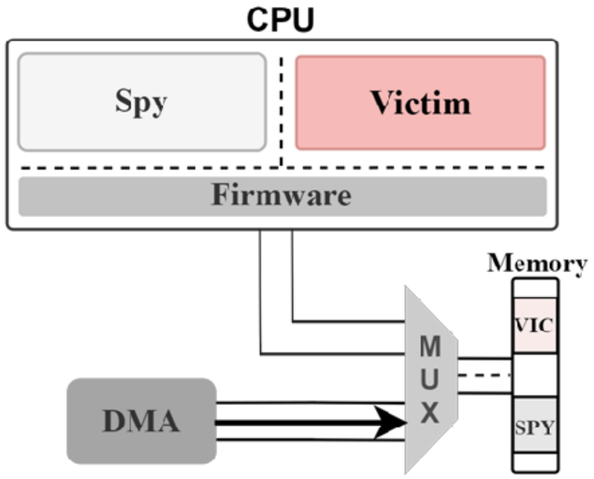
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else							
Trace							

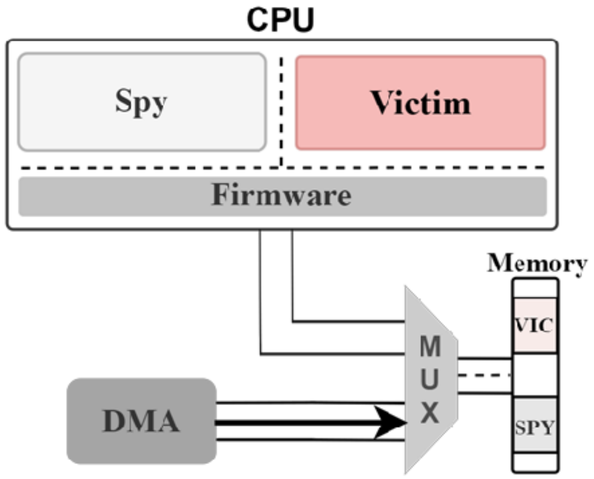
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else							
Trace							

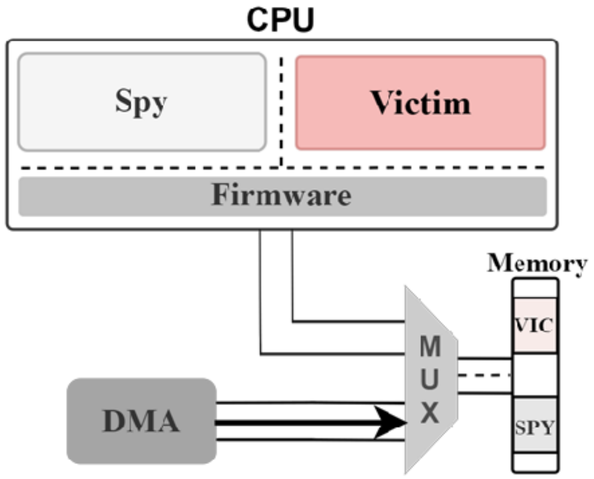
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else	cmp						
Trace	---						

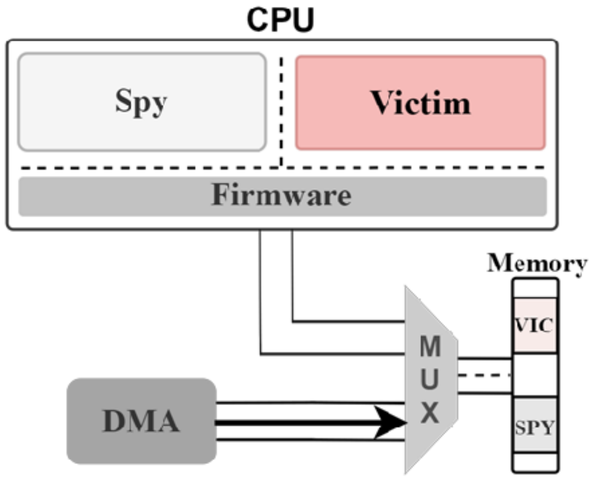
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else	cmp						
Trace	---						

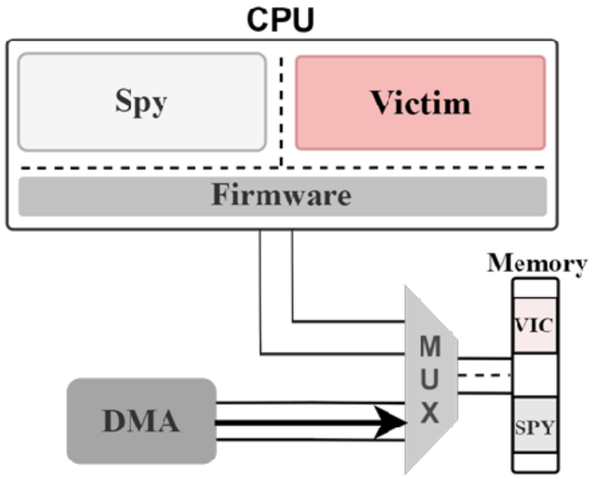
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else	cmp	beq					
Trace	---	---					

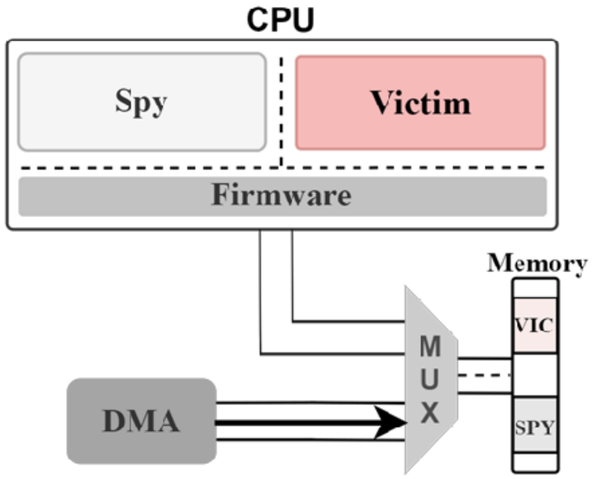
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else	cmp	beq	beq				
Trace	---	---	---				

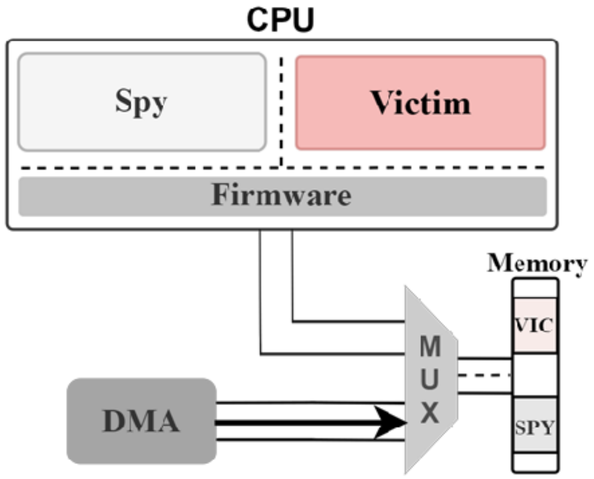
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else	cmp	beq	beq	beq			
Trace	---	---	---	---			

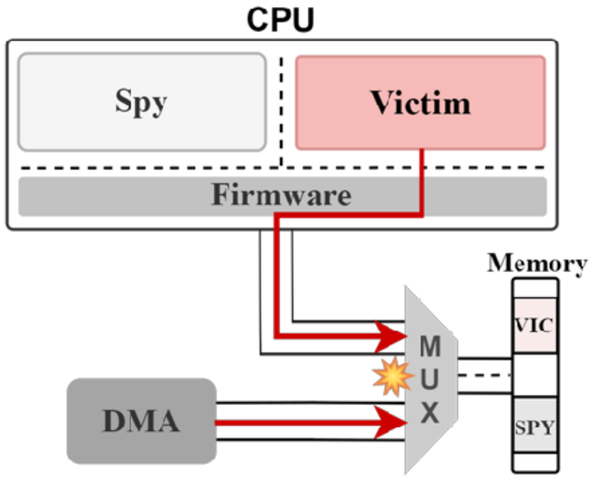
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else	cmp	beq	beq	beq	movs		
Trace	---	---	---	---	---		

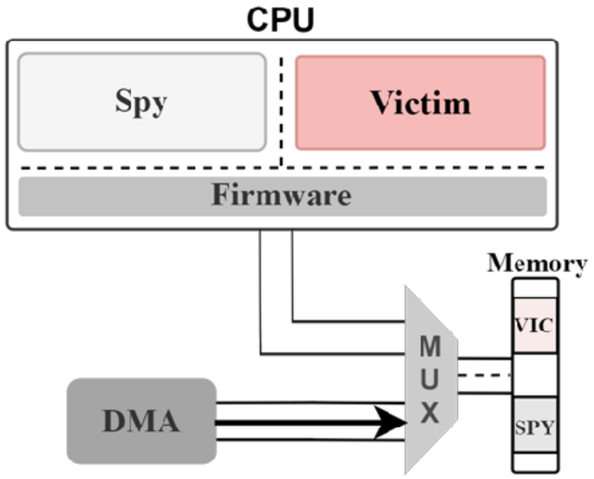
Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else	cmp	beq	beq	beq	movs	str	
Trace	---	---	---	---	---	X	

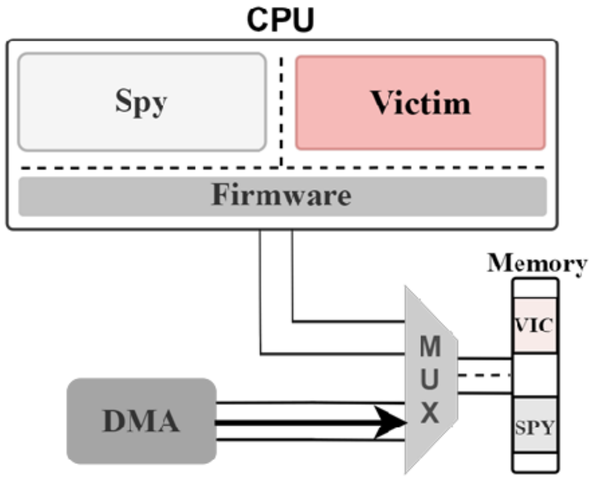
Attack Overview – Toy Example



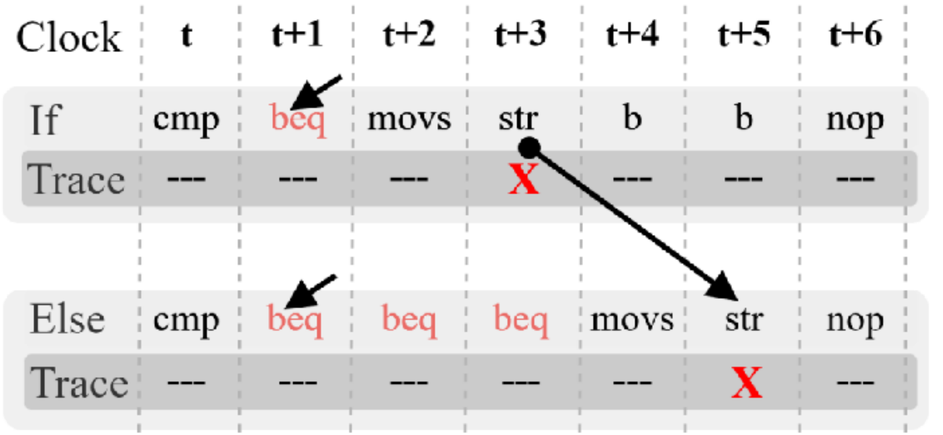

```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```

Clock	t	t+1	t+2	t+3	t+4	t+5	t+6
If	cmp	beq	movs	str	b	b	nop
Trace	---	---	---	X	---	---	---
Else	cmp	beq	beq	beq	movs	str	nop
Trace	---	---	---	---	---	X	---

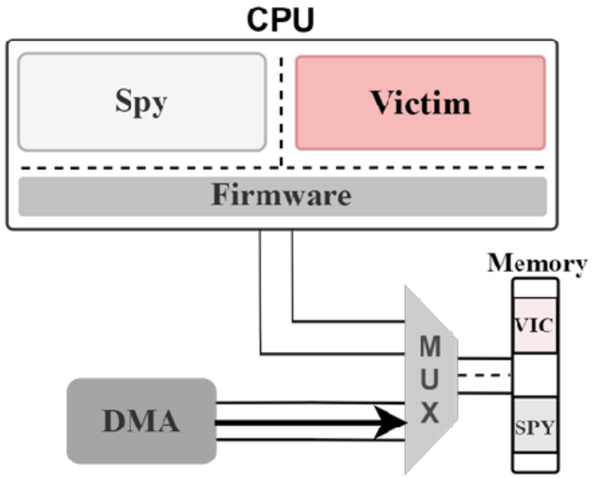
Attack Overview – Toy Example



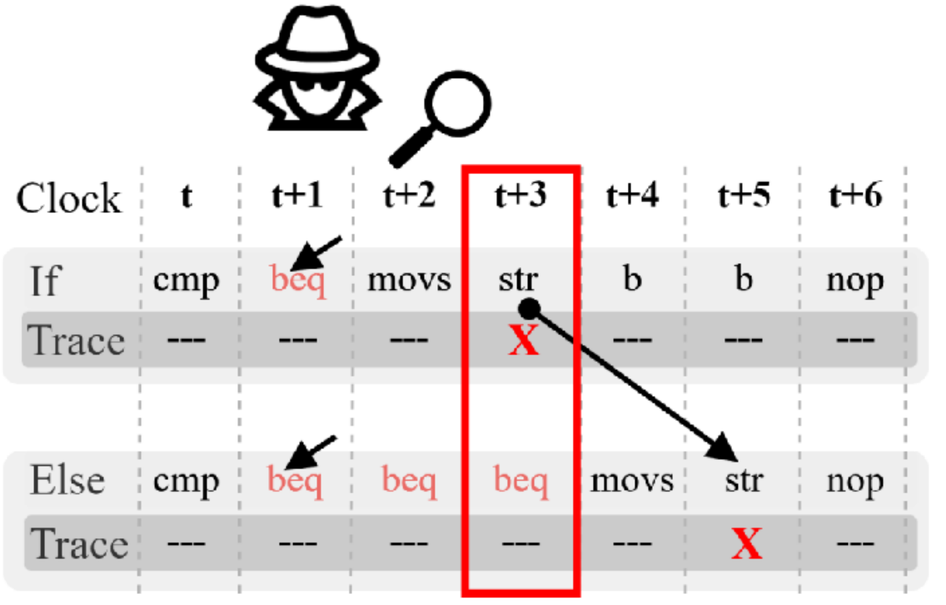

```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.n   END                       2 clk
ELSE:   movs  r3, #0                    1 clk
        str   r3, [r7, #0]              1 clk
END:    nop                             1 clk
```



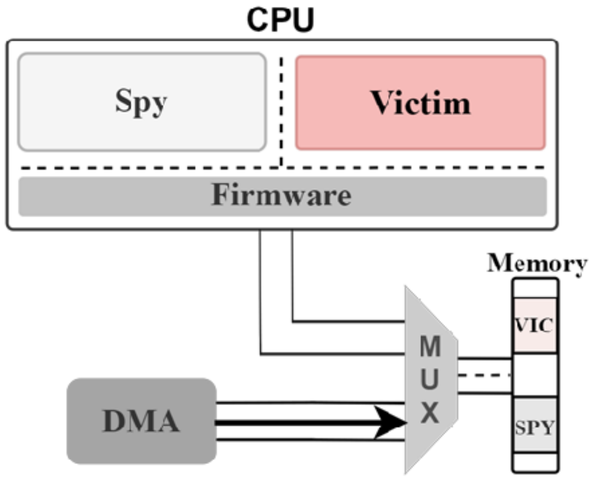
Attack Overview – Toy Example




```
        cmp   r3, #0              1 clk
        beq.n ELSE                3 clk (else), 1 clk (if)
IF:     movs  r3, #1              1 clk
        str   r3, [r7, #0]        1 clk
        b.n   END                 2 clk
ELSE:   movs  r3, #0              1 clk
        str   r3, [r7, #0]        1 clk
END:    nop                       1 clk
```

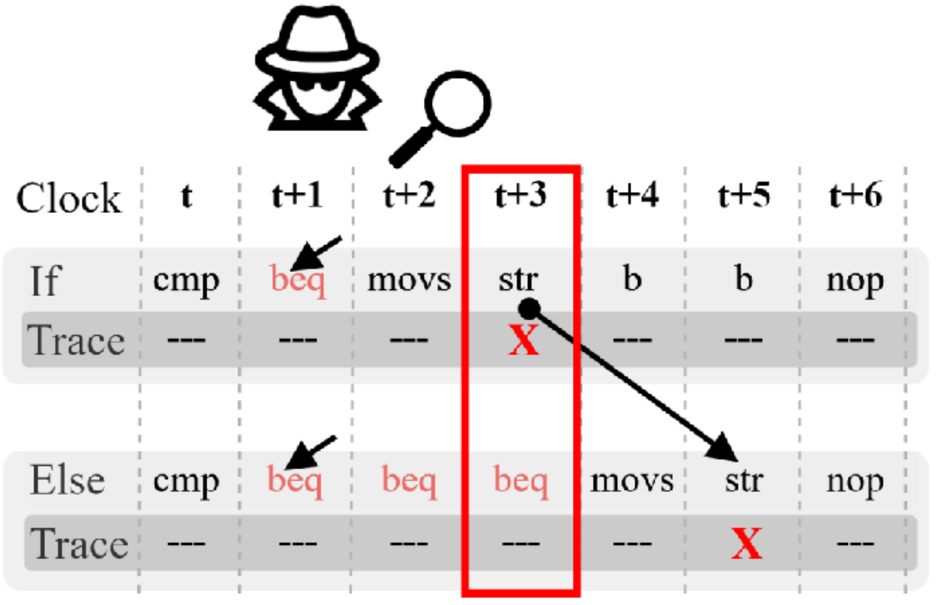


Attack Overview – Toy Example

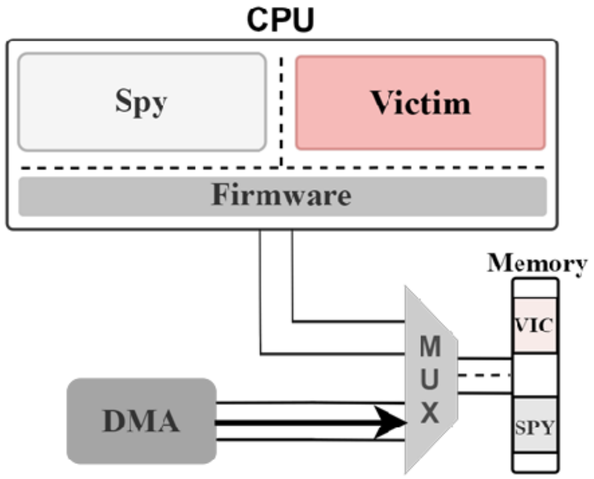



```
        cmp   r3, #0          1 clk
        beq.n ELSE            3 clk (else), 1 clk (if)
IF:     movs  r3, #1          1 clk
        str   r3, [r7, #0]    1 clk
        b.r                   2 clk
ELSE:   mov                   1 clk
        str                   1 clk
END:    nop                   1 clk
```

```
if(s==1)
  var=1;
else
  var=0;
```

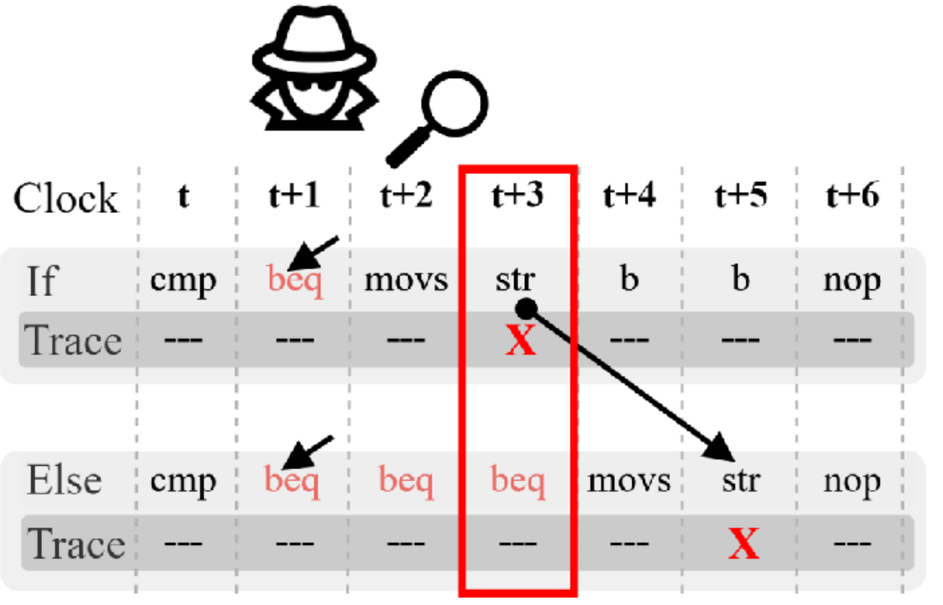


Attack Overview – Toy Example




```
        cmp   r3, #0                    1 clk
        beq.n ELSE                      3 clk (else), 1 clk (if)
IF:     movs  r3, #1                    1 clk
        str   r3, [r7, #0]              1 clk
        b.r                             2 clk
ELSE:   mov                             1 clk
        str                             1 clk
END:    nop                             1 clk
```

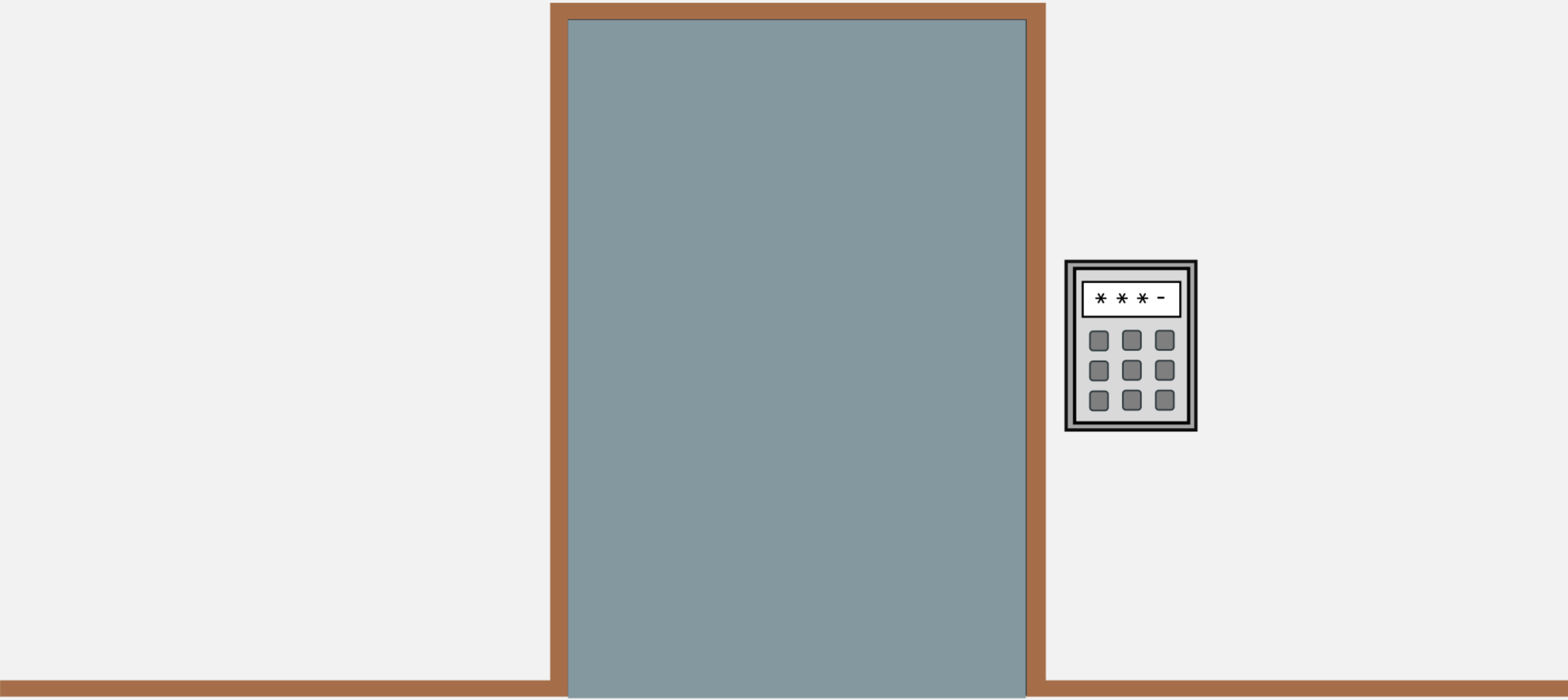
```
if(s==1)
   var=1;
else
   var=0;
```

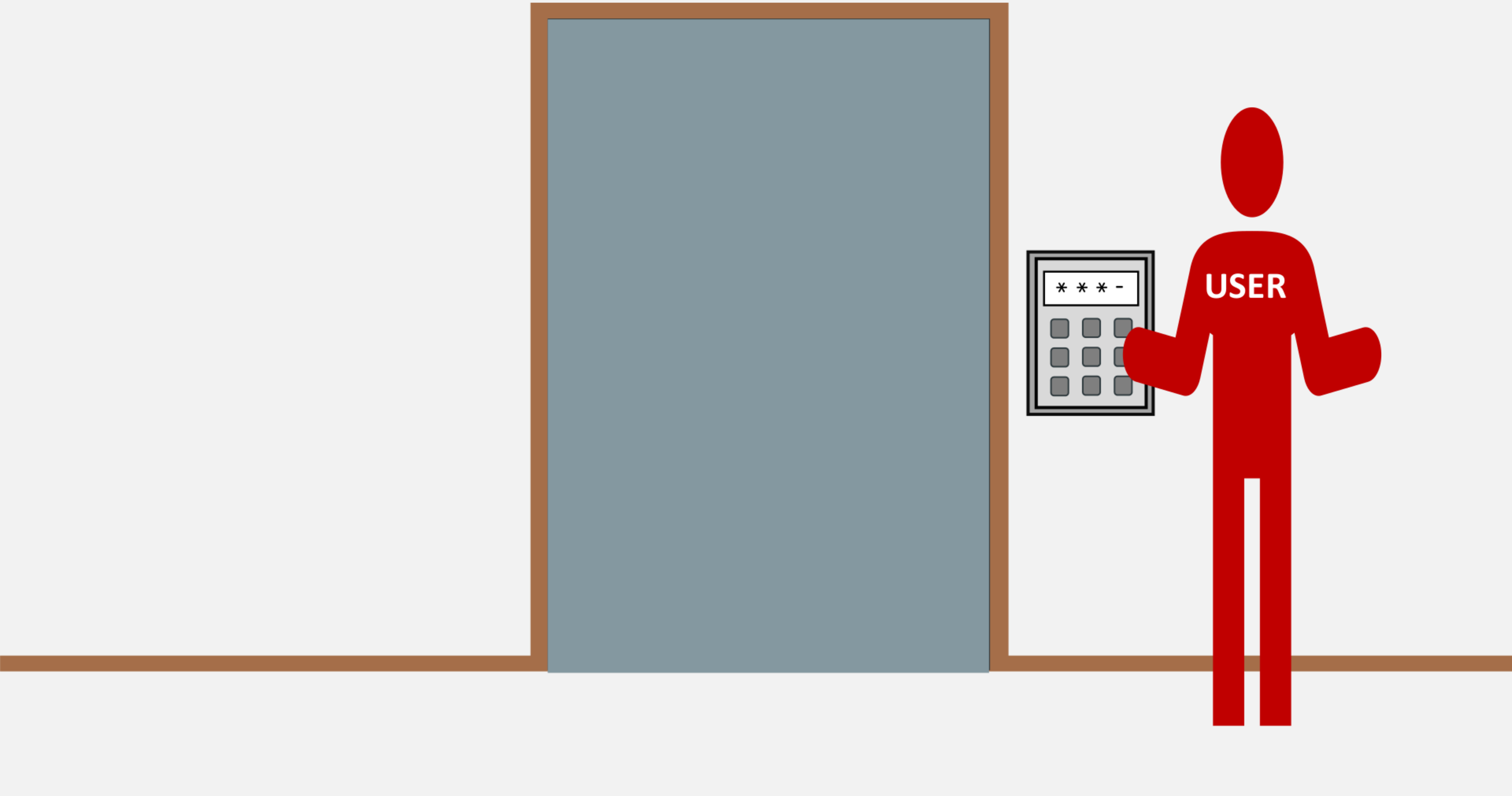


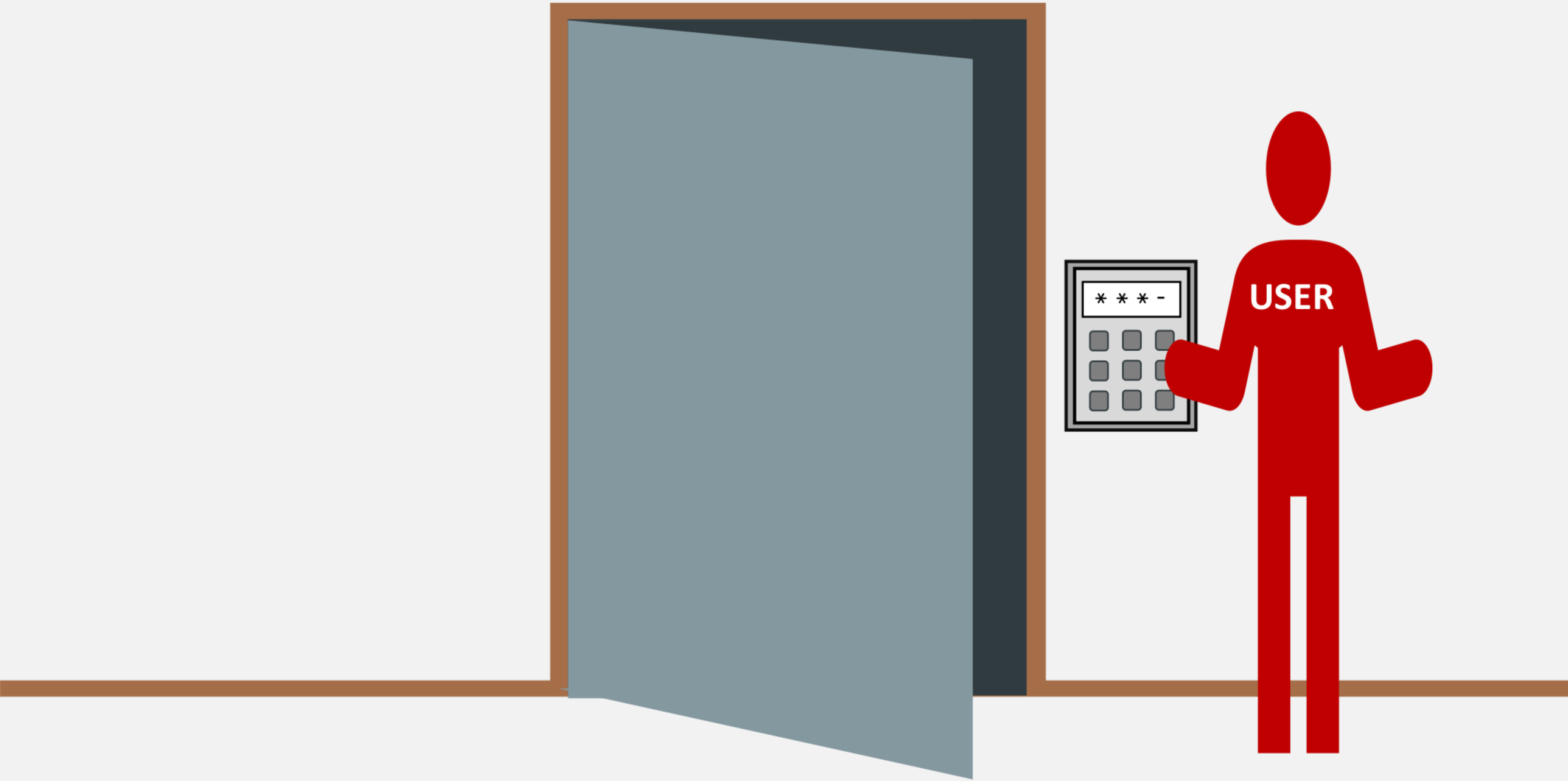
SECRET = 1

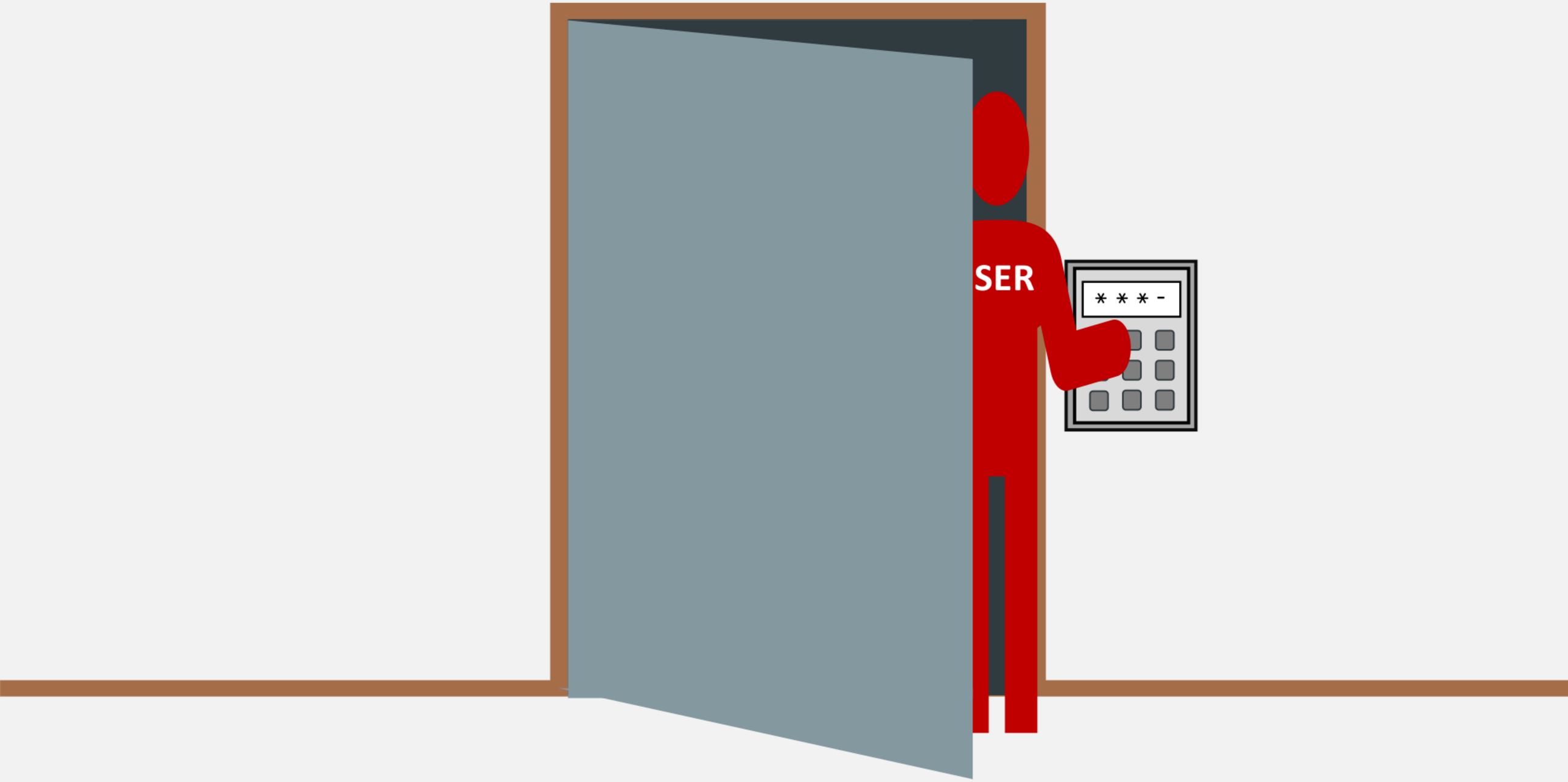
BUSted

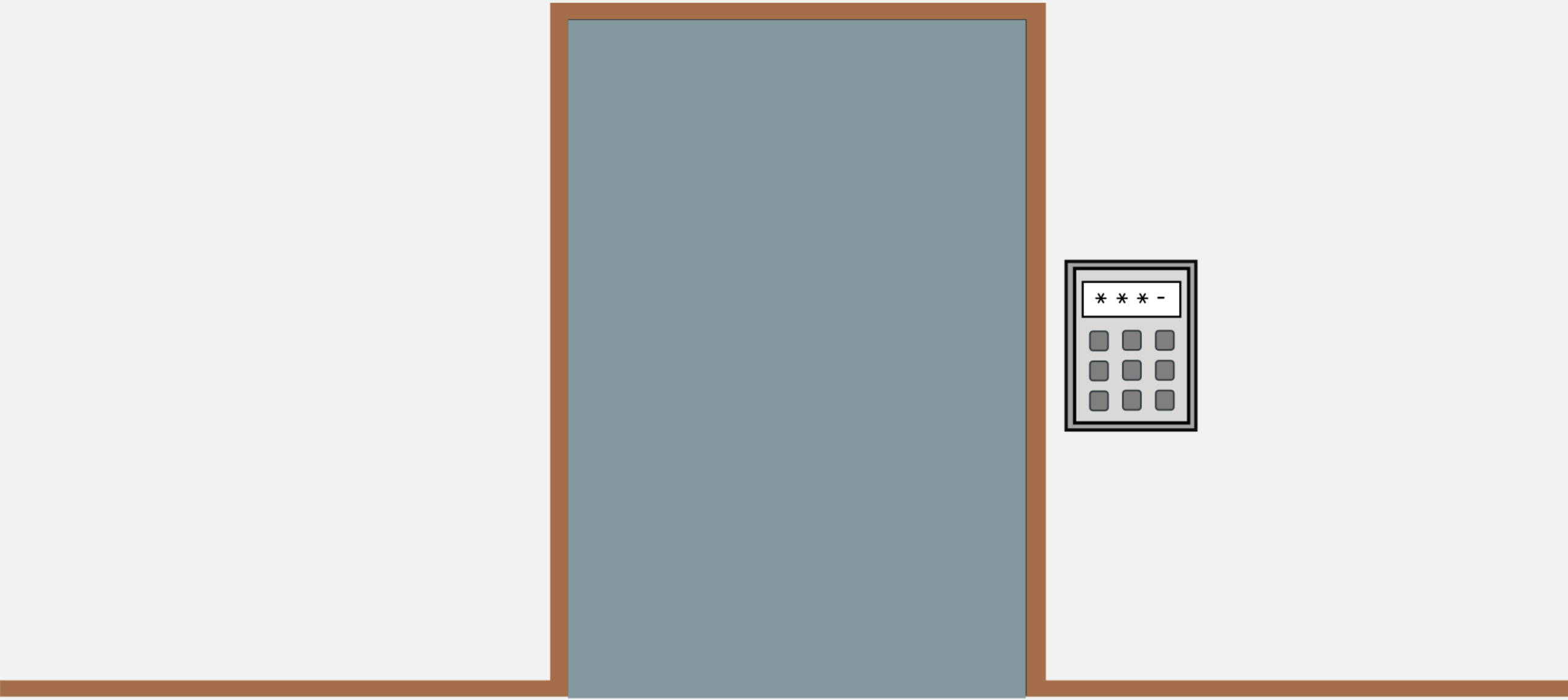
Microarchitectural Side-Channel Attacks on MCUs

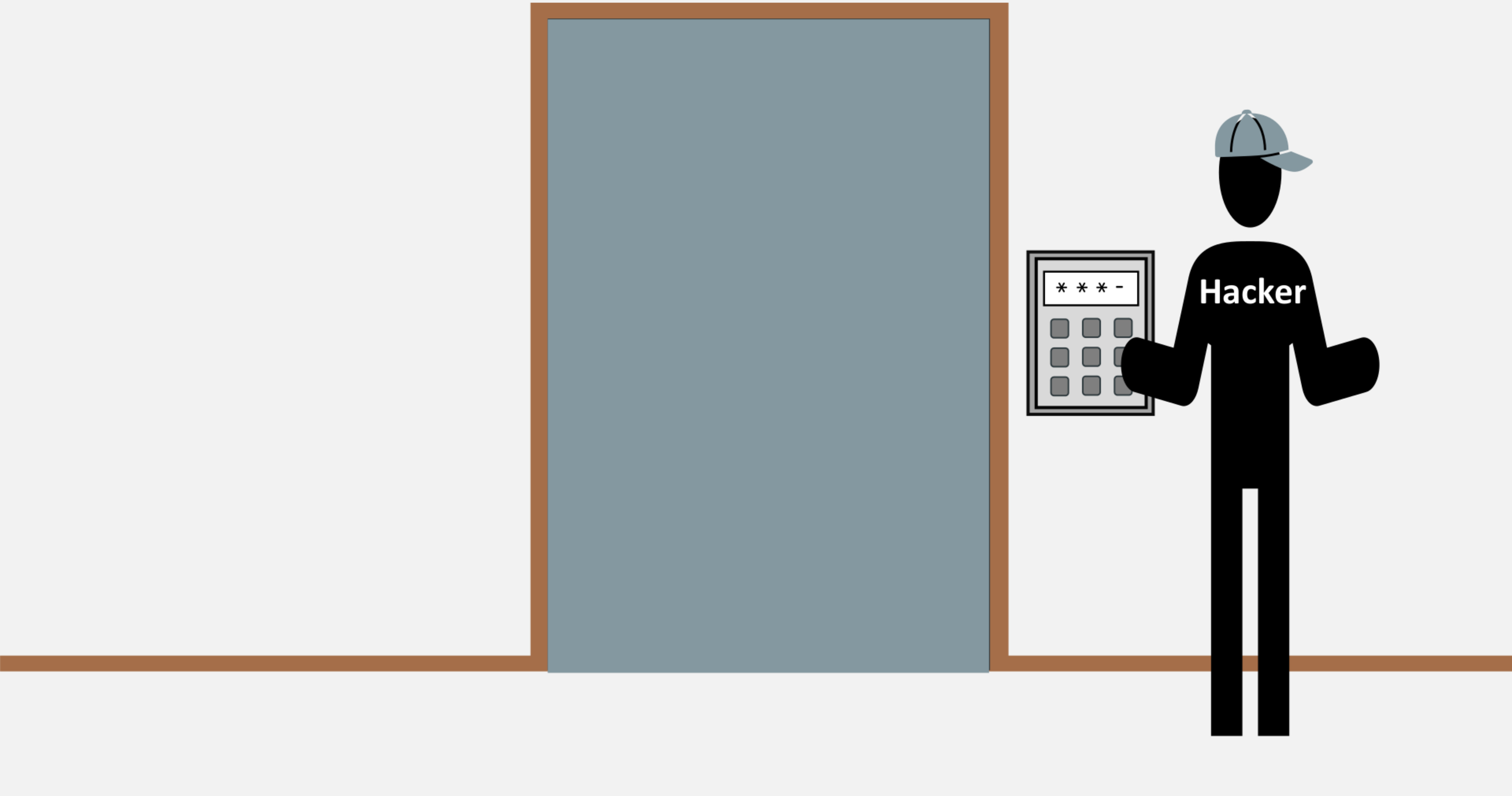


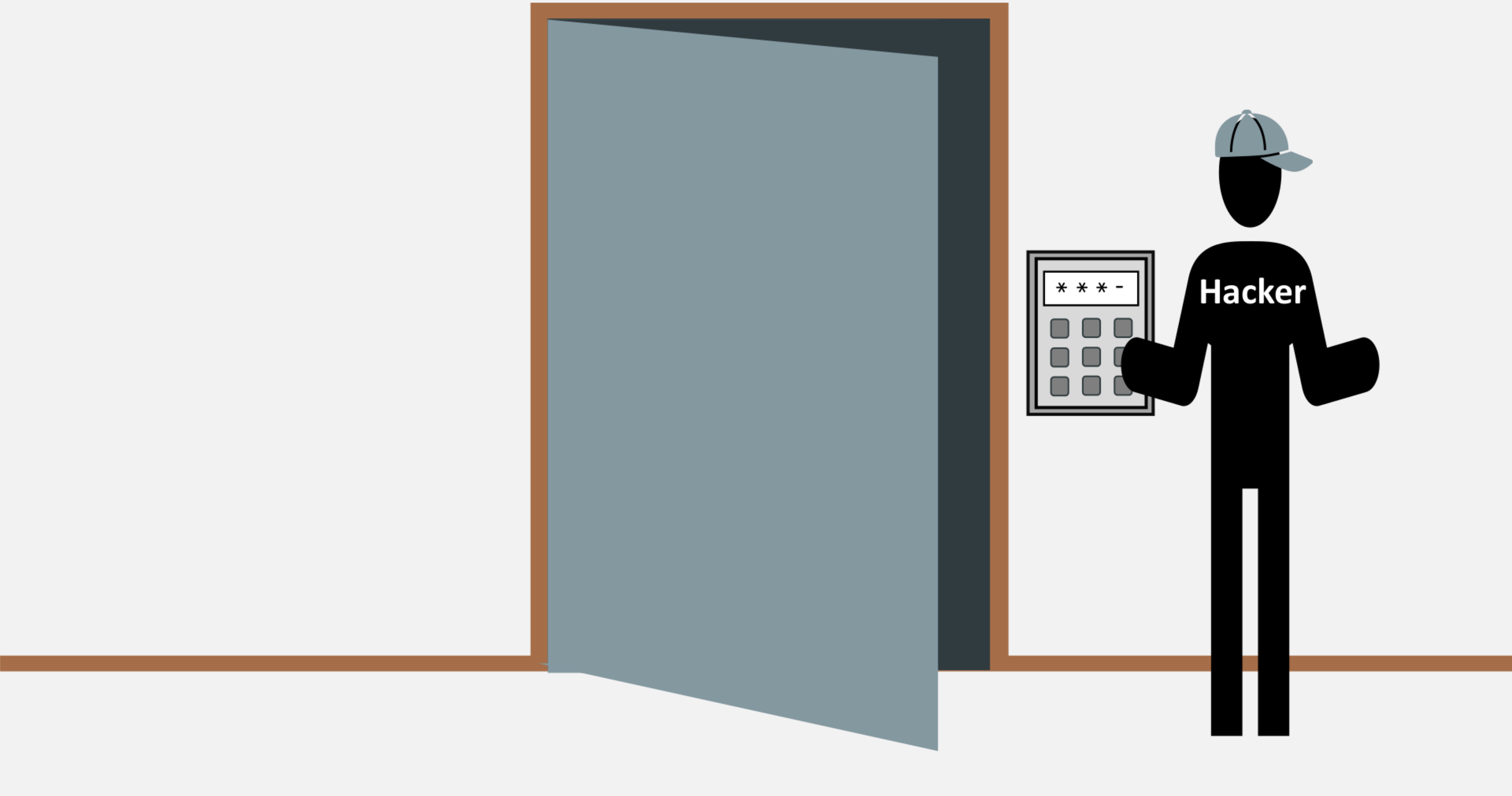

* * * -
USER

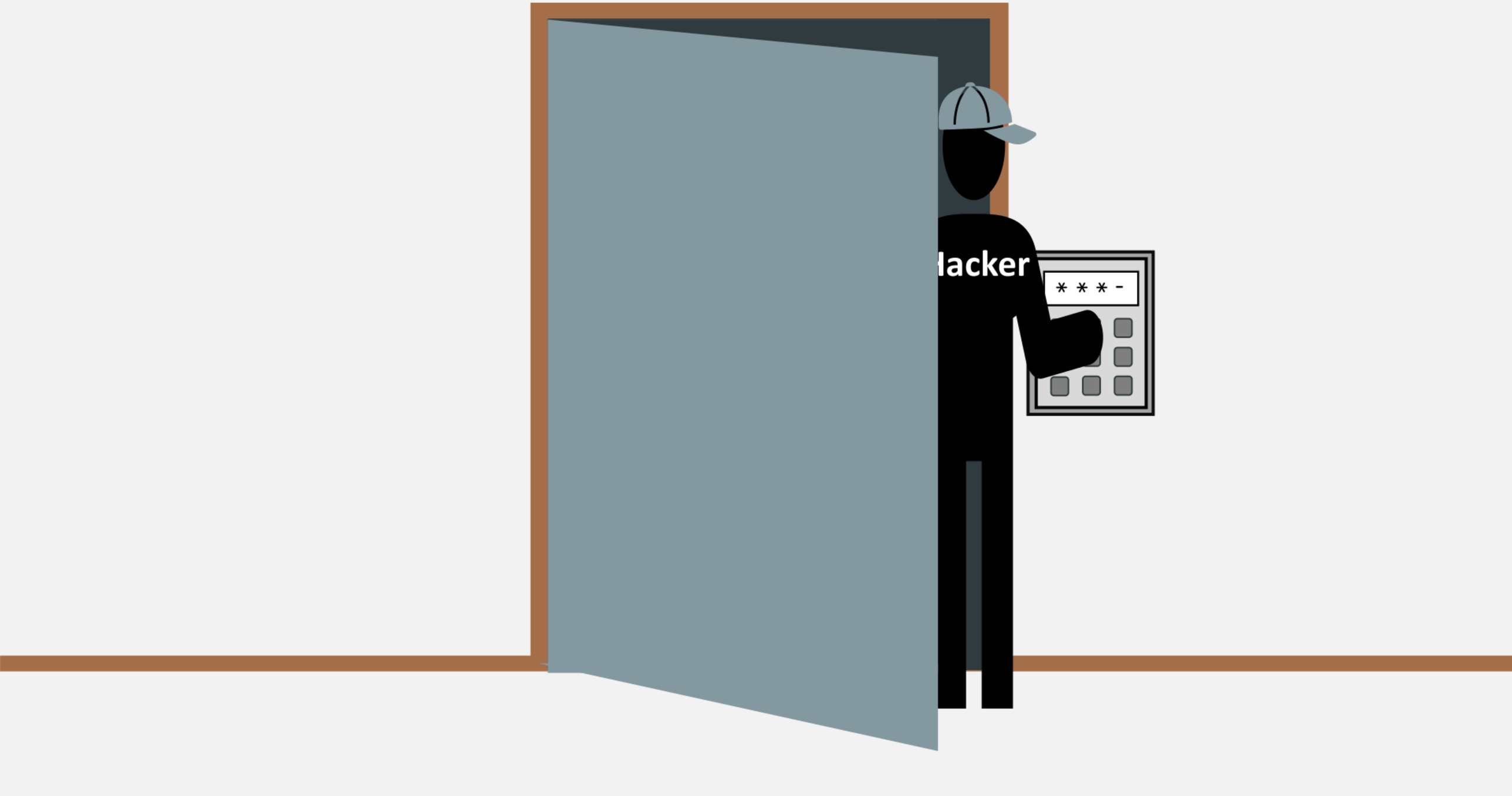

* * * -
USER

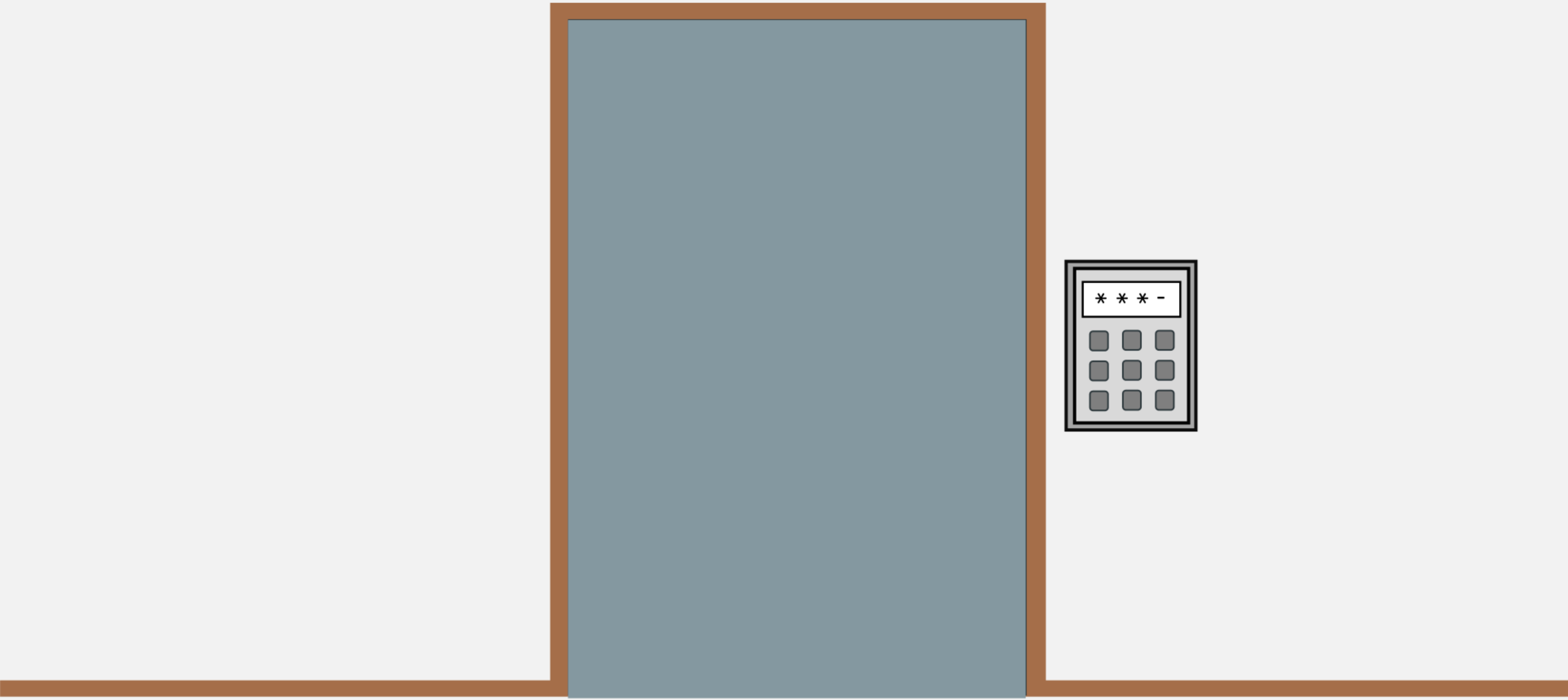

SER
* * * -

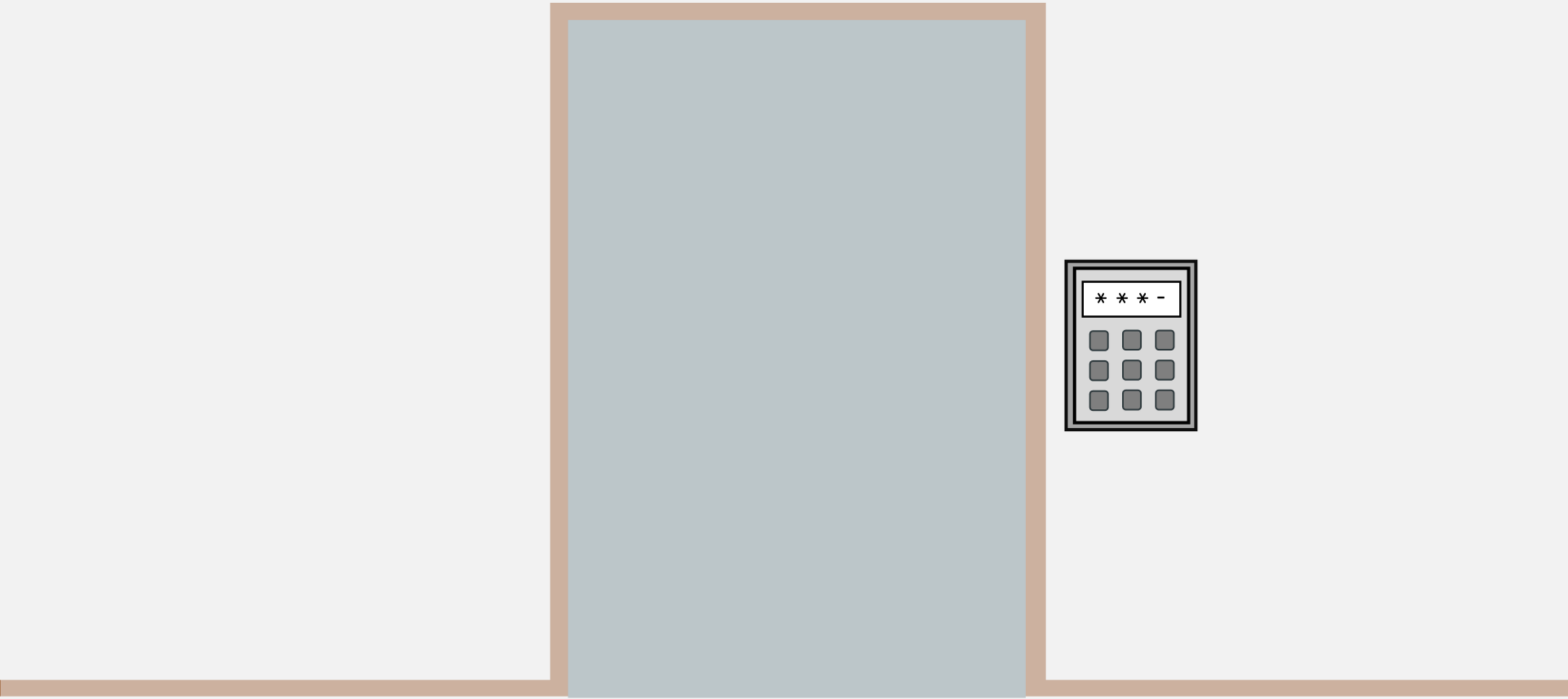



* * * -
Hacker

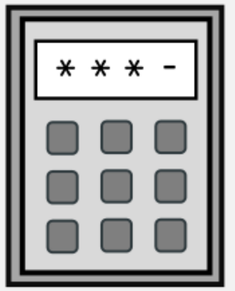

* * * -
Hacker

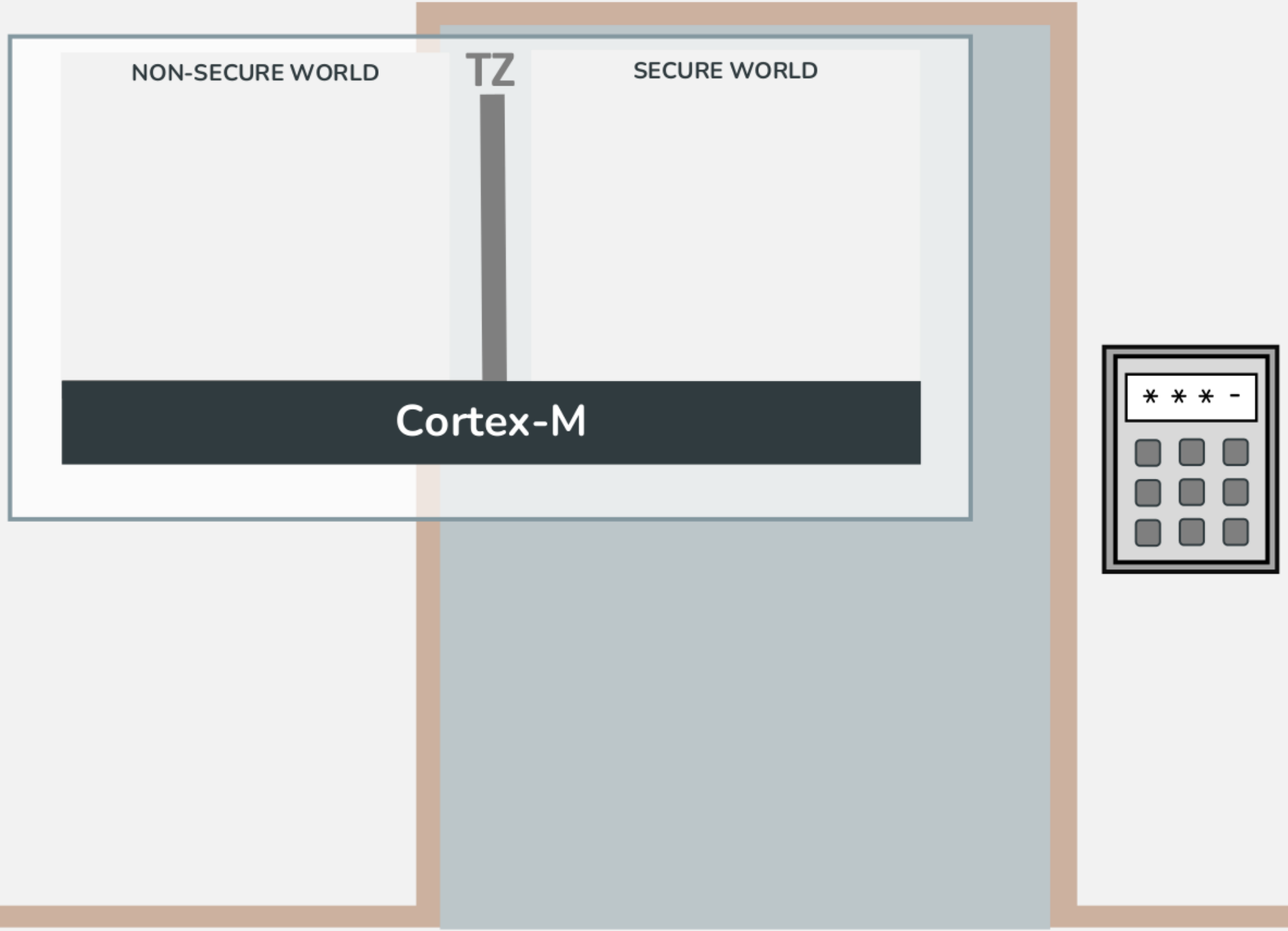

Hacker
* * * -

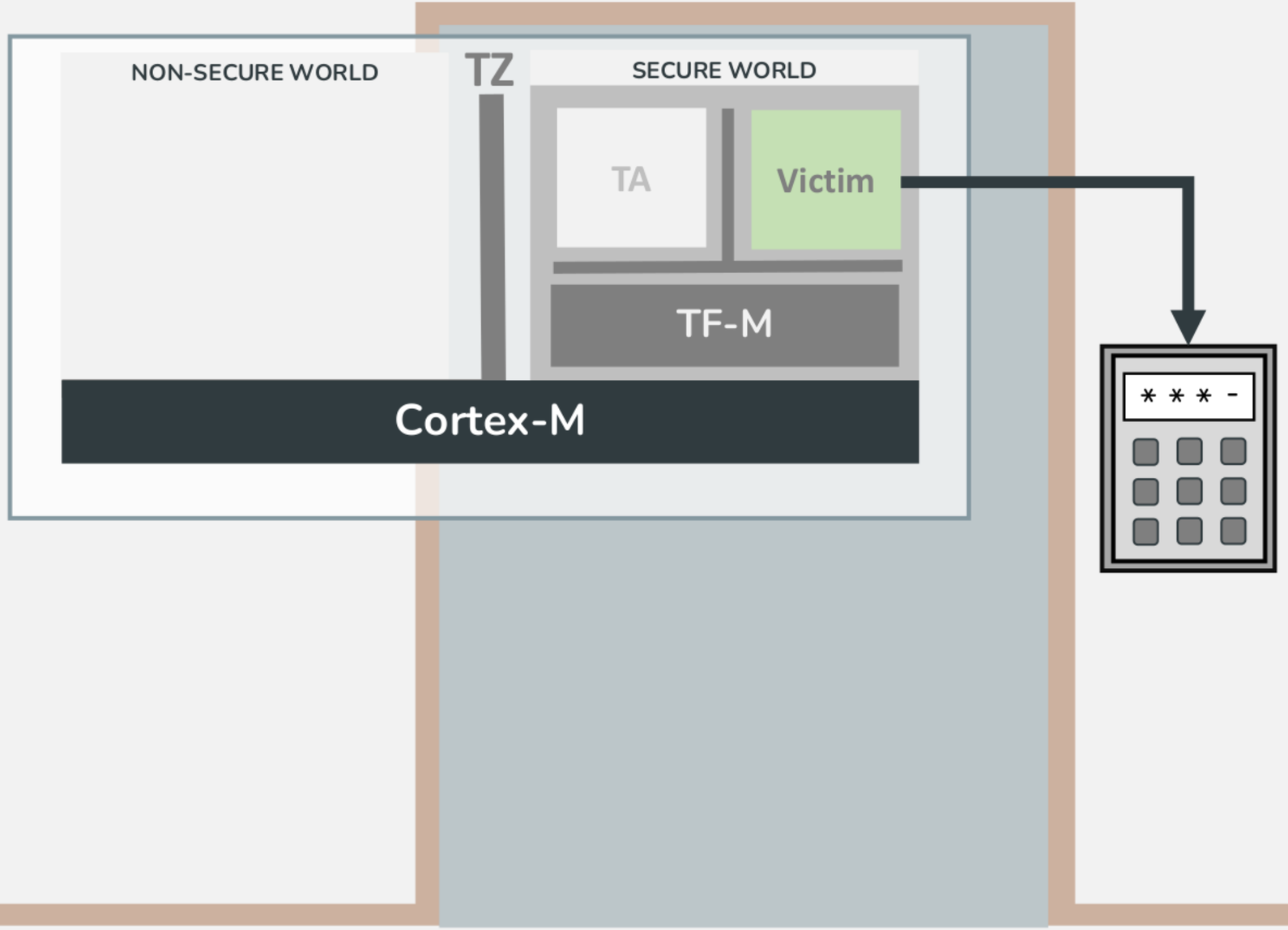


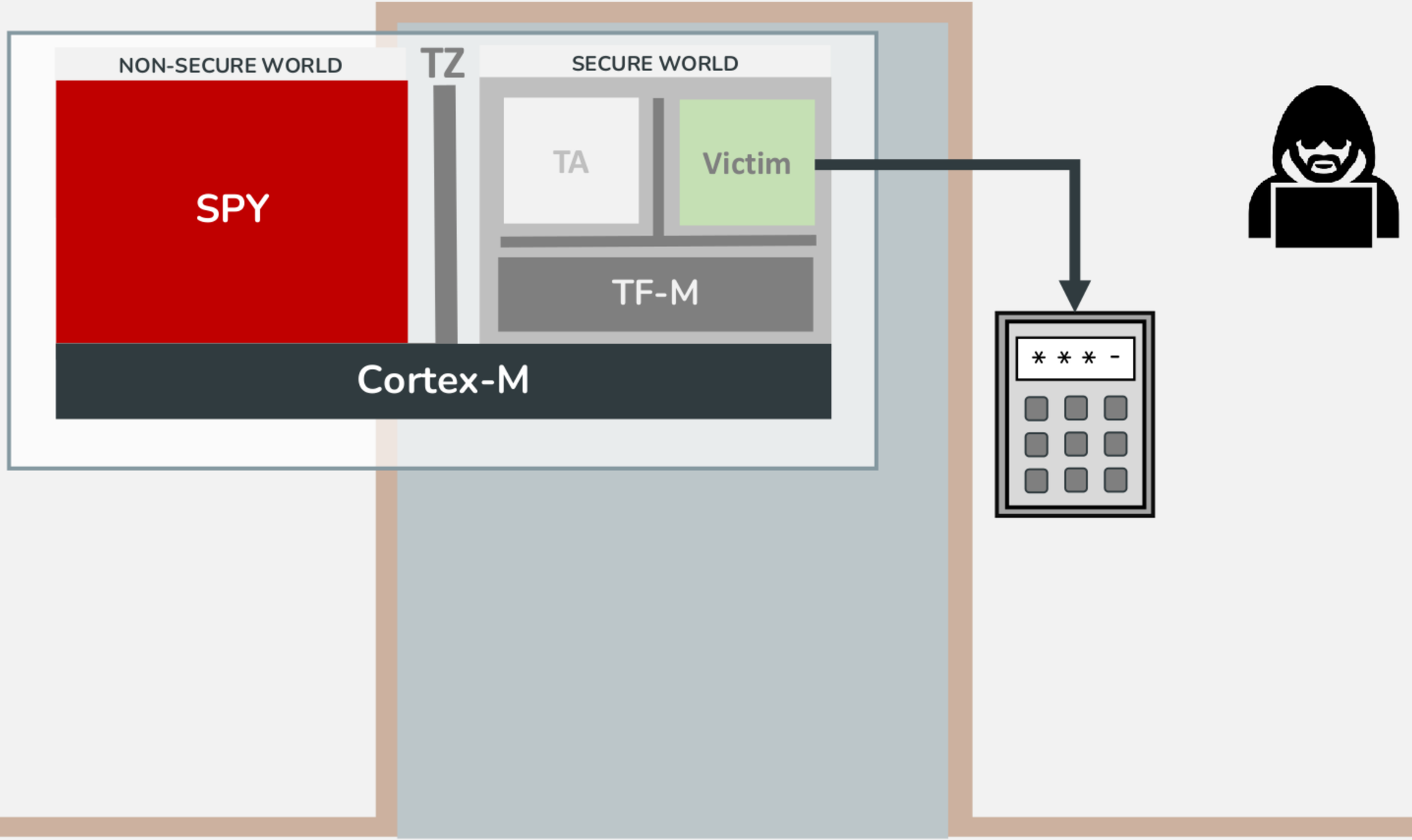


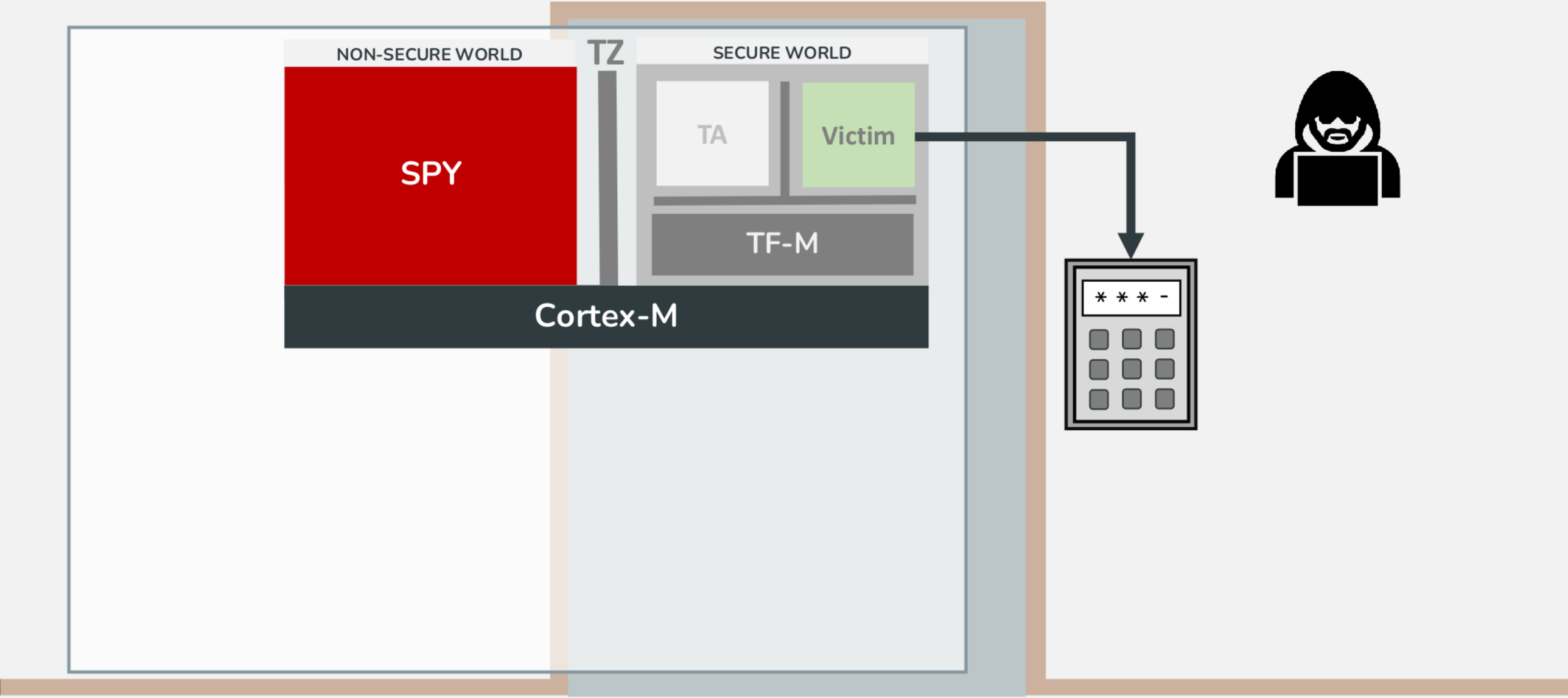
Cortex-M

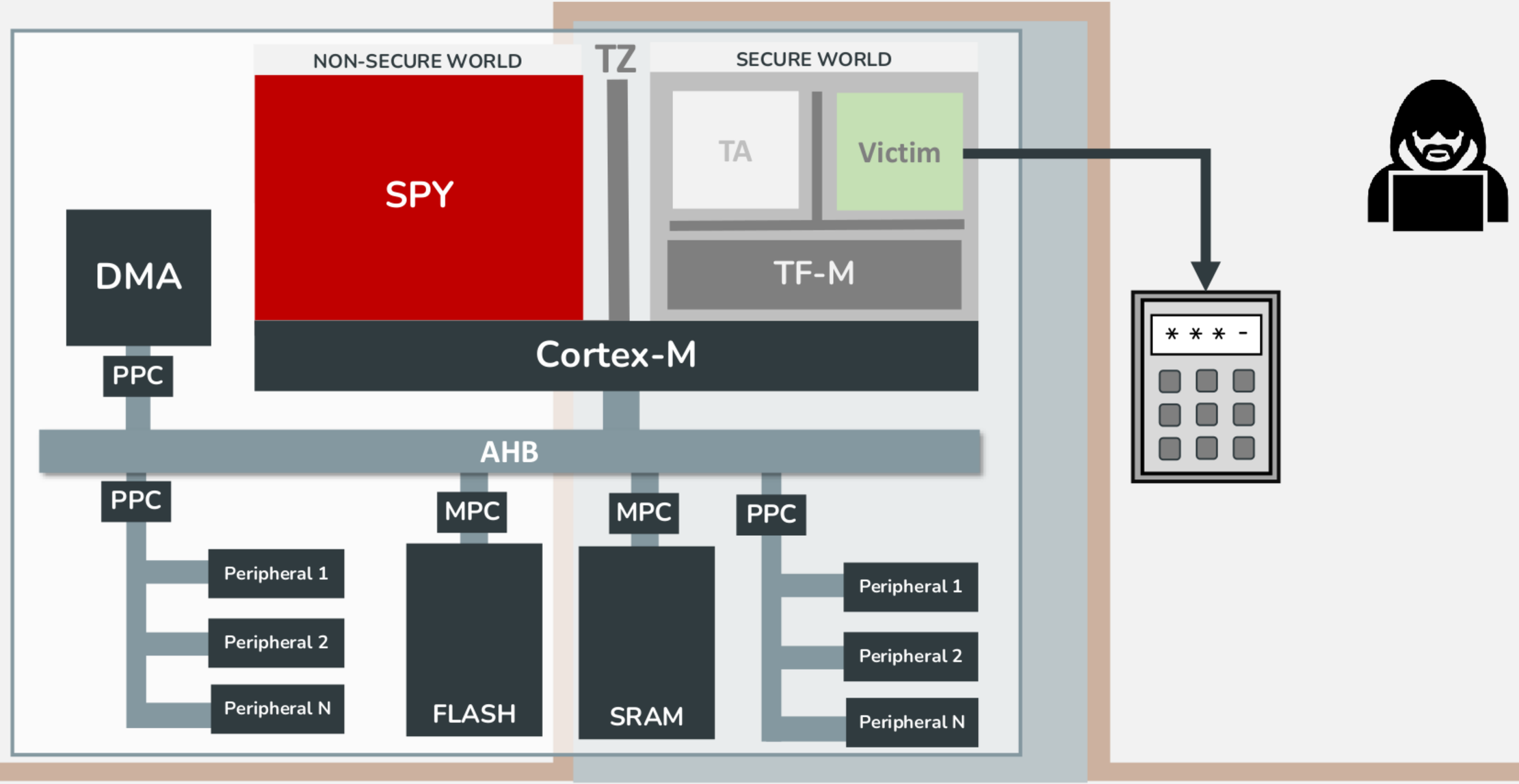


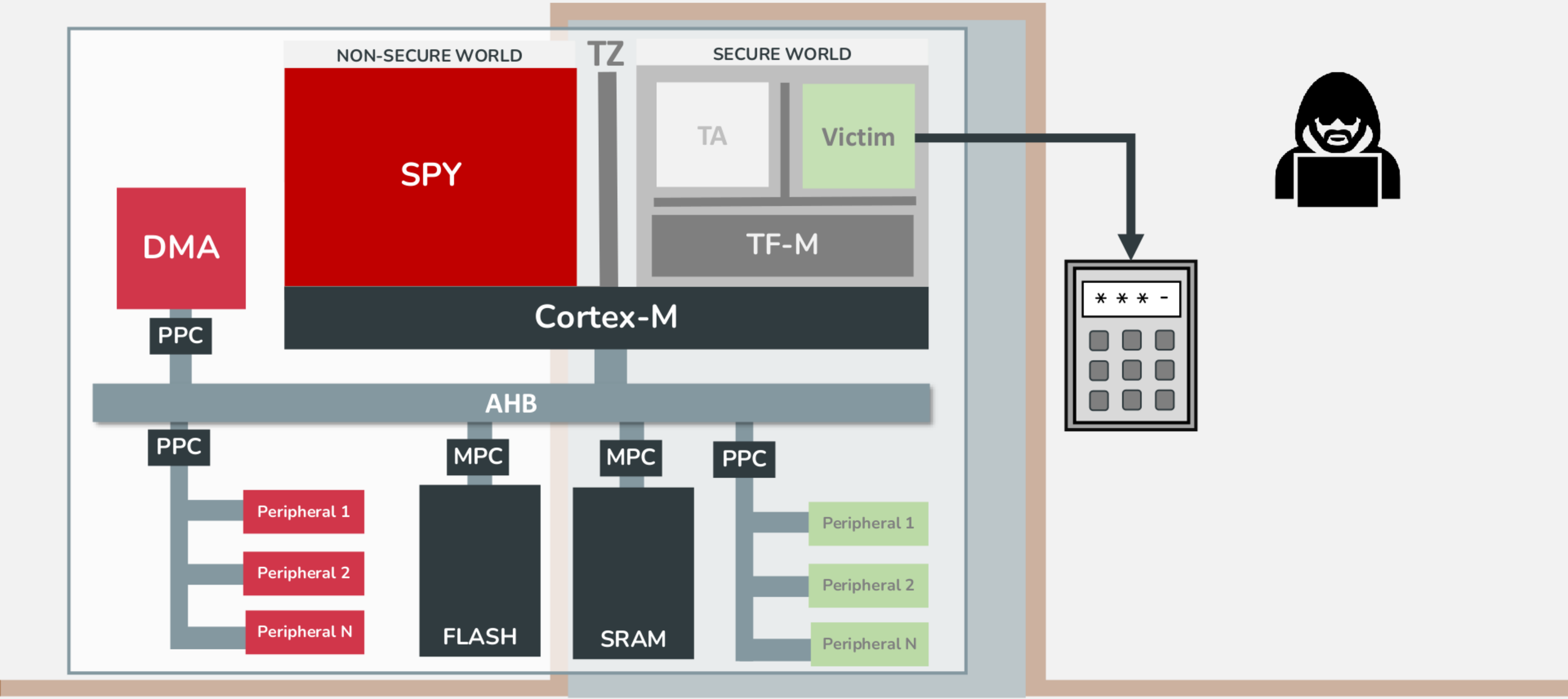

NON-SECURE WORLD
TZ
SECURE WORLD
Cortex-M
* * * -

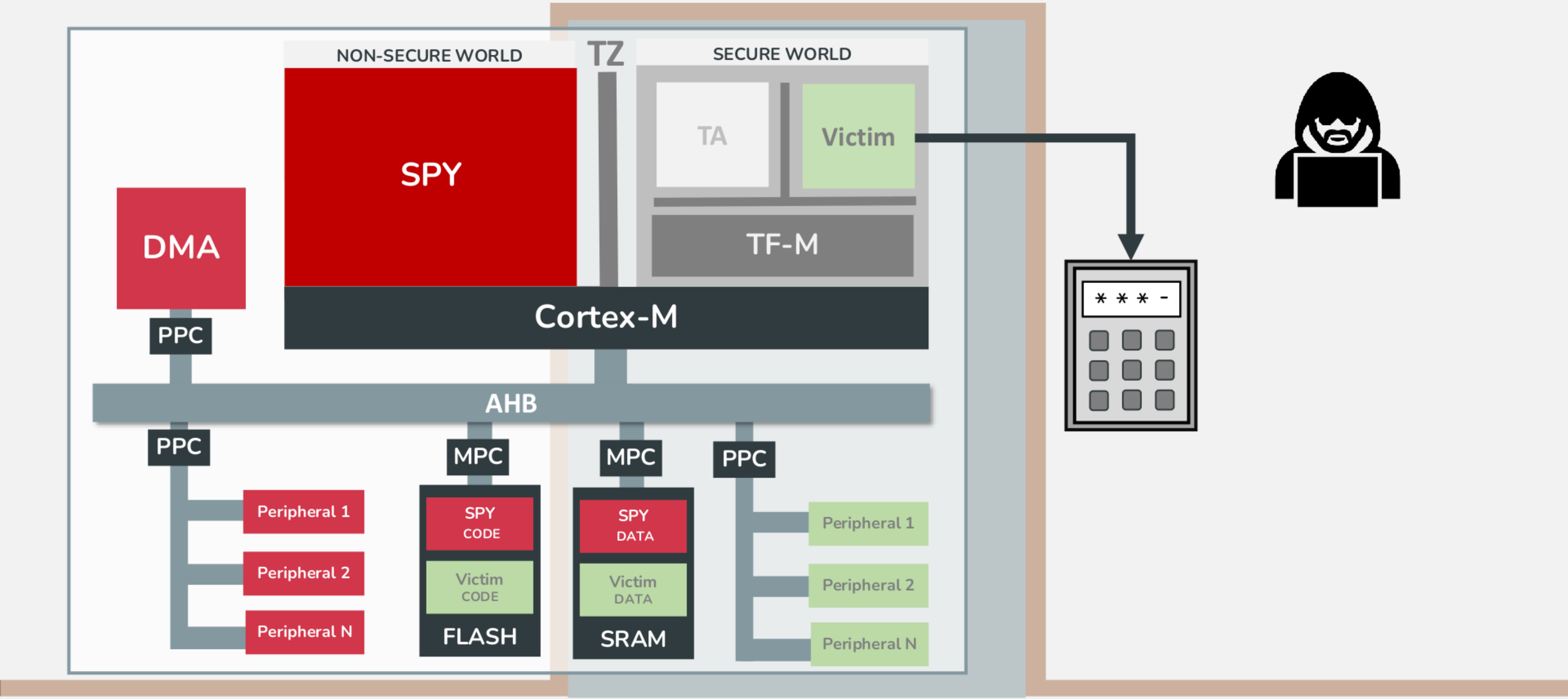

NON-SECURE WORLD
TZ
SECURE WORLD
TA
Victim
TF-M
Cortex-M
* * * -


NON-SECURE WORLD
TZ
SECURE WORLD
SPY
TA
Victim
TF-M
Cortex-M
* * * -

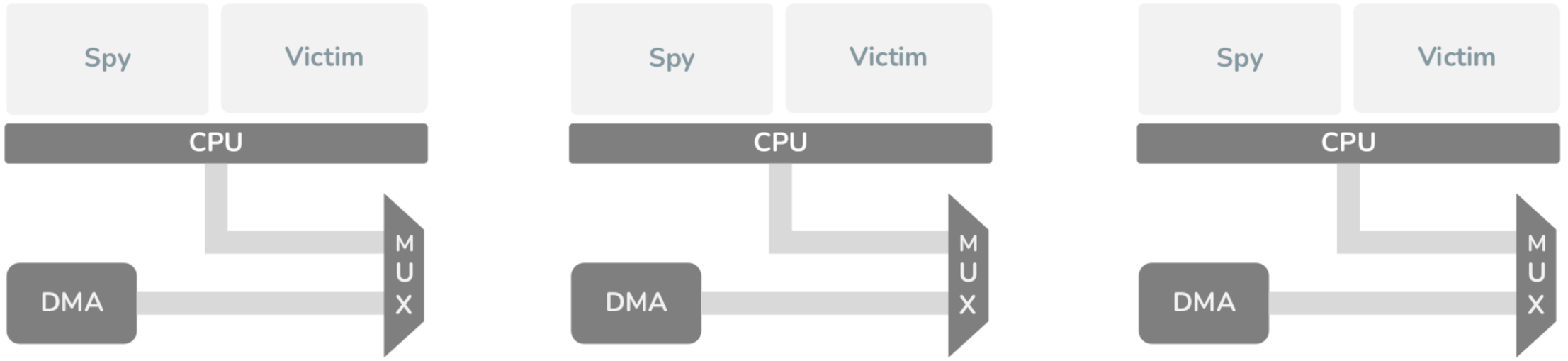

NON-SECURE WORLD
TZ
SECURE WORLD
SPY
TA
Victim
TF-M
Cortex-M
* * * -


NON-SECURE WORLD
TZ
SECURE WORLD
SPY
TA
Victim
TF-M
DMA
Cortex-M
PPC
AHB
PPC
MPC
MPC
PPC
Peripheral 1
Peripheral 2
Peripheral N
FLASH
SRAM
Peripheral 1
Peripheral 2
Peripheral N
* * * -


NON-SECURE WORLD
TZ
SECURE WORLD
SPY
TA
Victim
TF-M
DMA
Cortex-M
PPC
AHB
PPC
MPC
MPC
PPC
Peripheral 1
Peripheral 2
Peripheral N
FLASH
SRAM
Peripheral 1
Peripheral 2
Peripheral N
* * * -

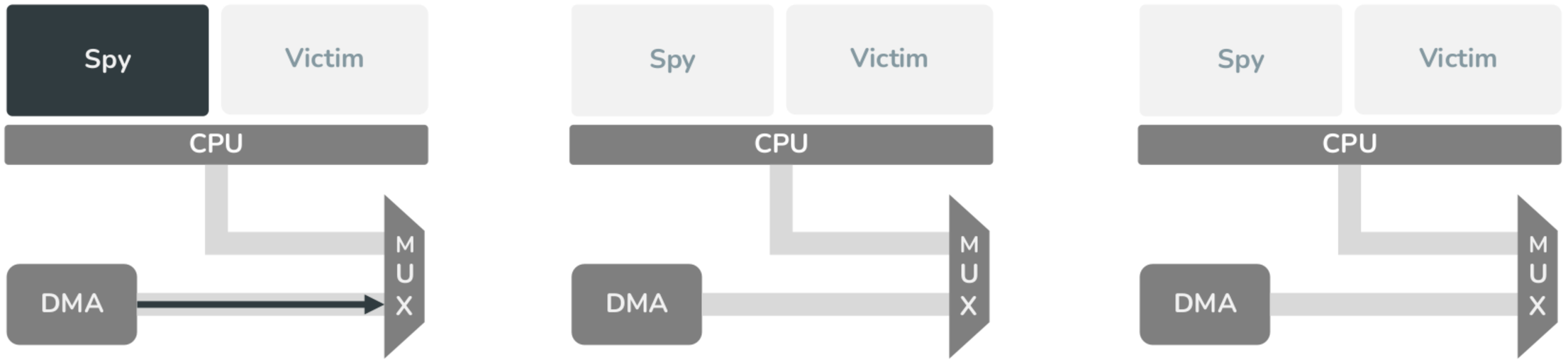

NON-SECURE WORLD
TZ
SECURE WORLD
SPY
TA
Victim
TF-M
DMA
Cortex-M
PPC
AHB
PPC
MPC
MPC
PPC
Peripheral 1
Peripheral 2
Peripheral N
SPY CODE
Victim CODE
FLASH
SPY DATA
Victim DATA
SRAM
Peripheral 1
Peripheral 2
Peripheral N
* * * -

Challenges

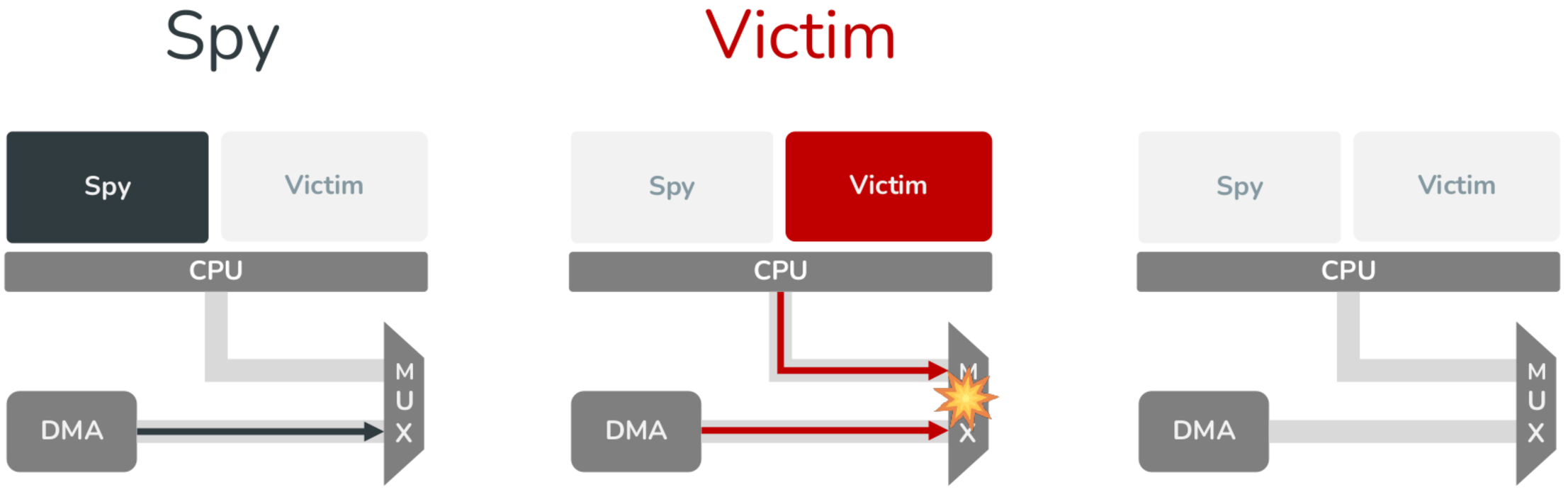

Spy
Victim
CPU
DMA
MUX
Spy
Victim
CPU
DMA
MUX
Spy
Victim
CPU
DMA
MUX

Challenges

Spy

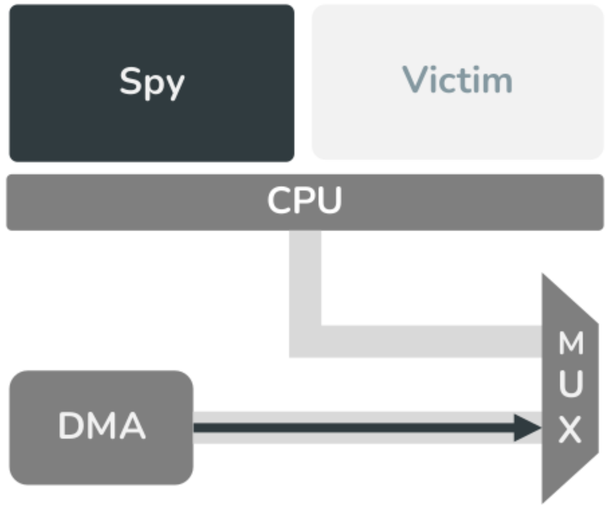


Challenges

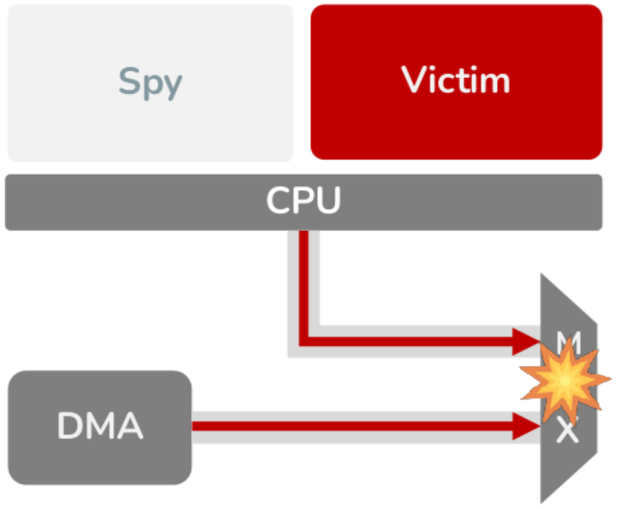

Spy
Victim
Spy
Victim
CPU
DMA
MUX
Spy
Victim
CPU
DMA
MUX
Spy
Victim
CPU
DMA
MUX

Challenges

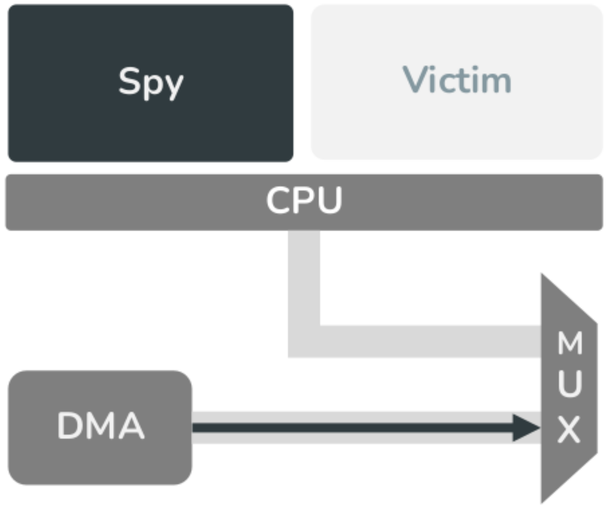
Spy



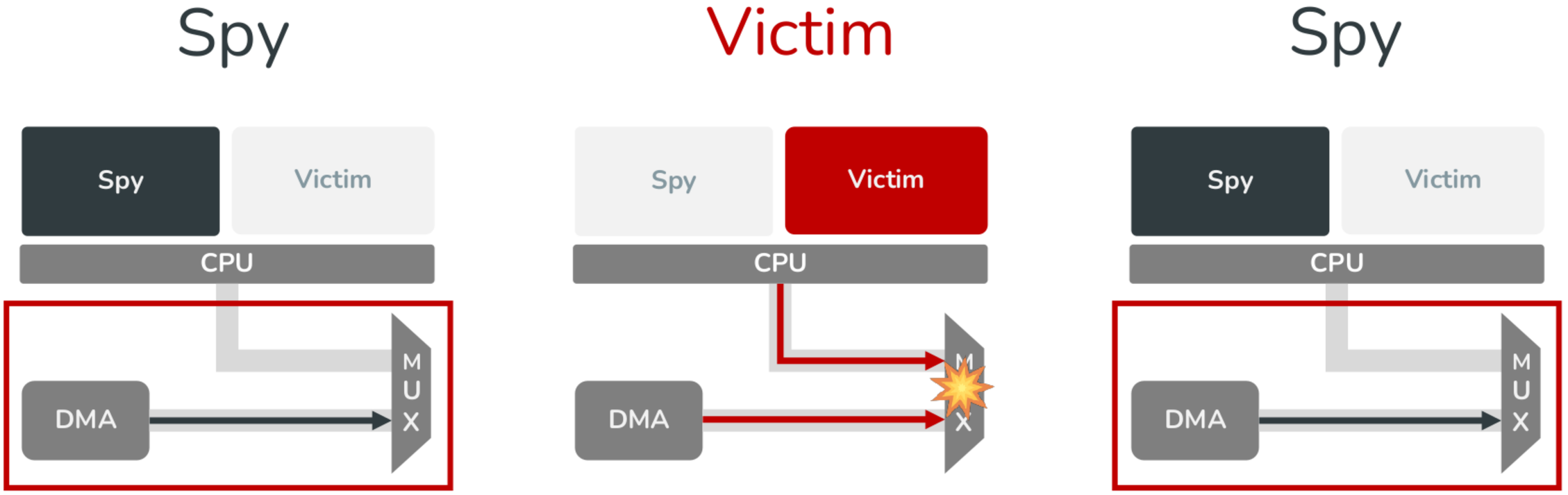
Victim



Spy



Challenges



No Difference

Challenges

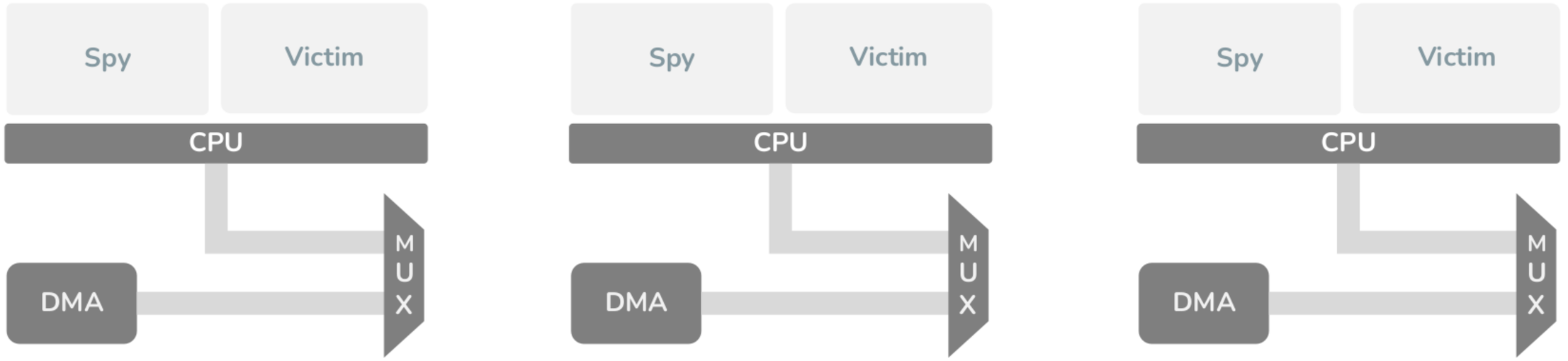
No Difference

Challenges

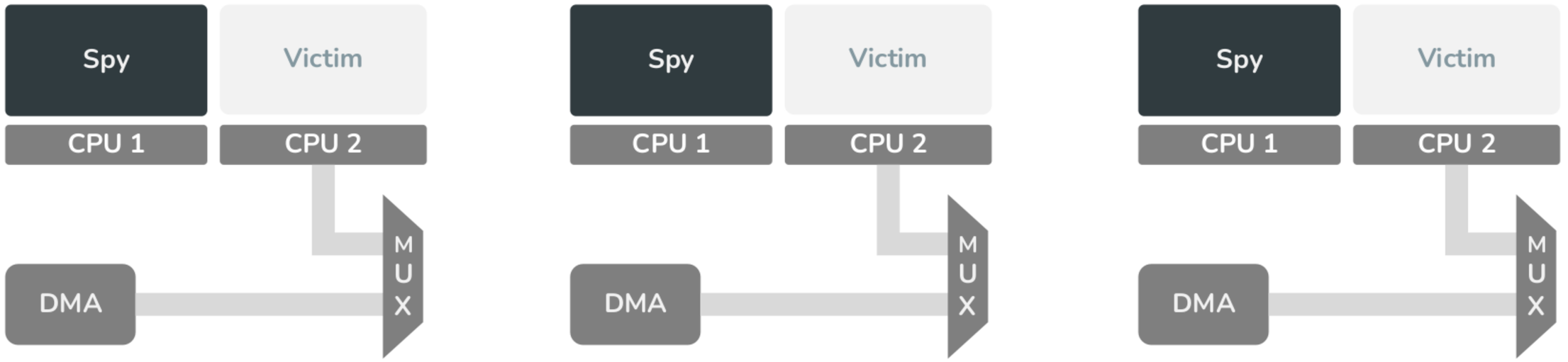
C1 - The bus is a stateless component;

No Difference

Challenges

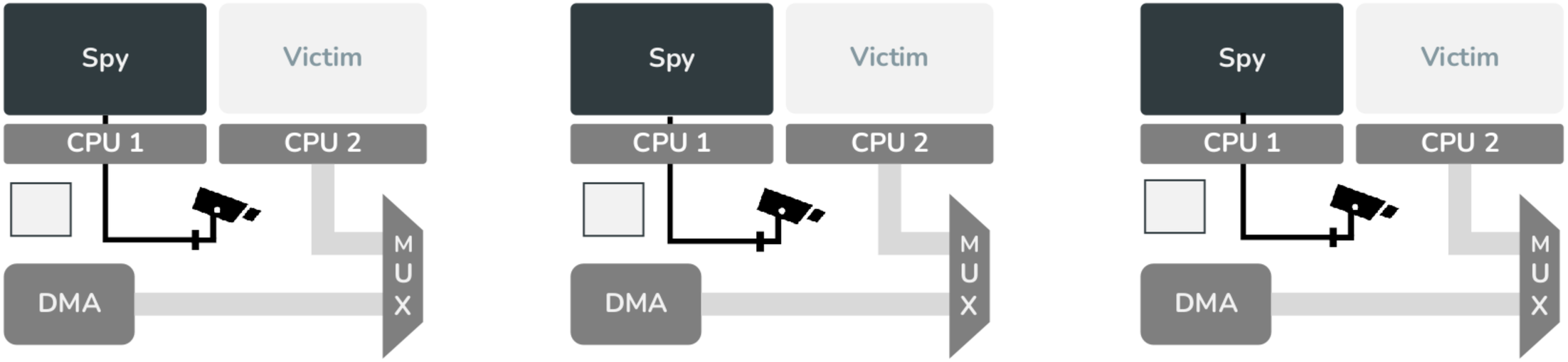

Spy
Victim
CPU
DMA
MUX
Spy
Victim
CPU
DMA
MUX
Spy
Victim
CPU
DMA
MUX

Challenges



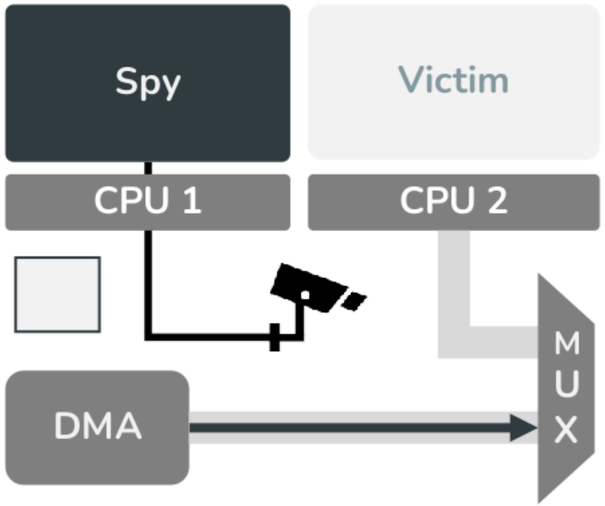
Dual-Core, Spy Always Executing

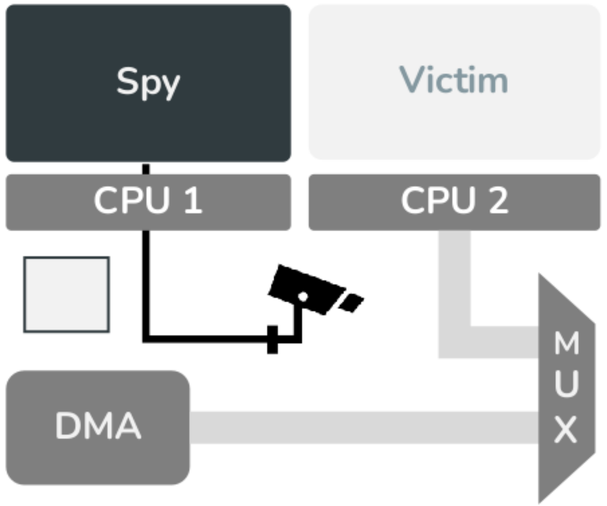
Challenges

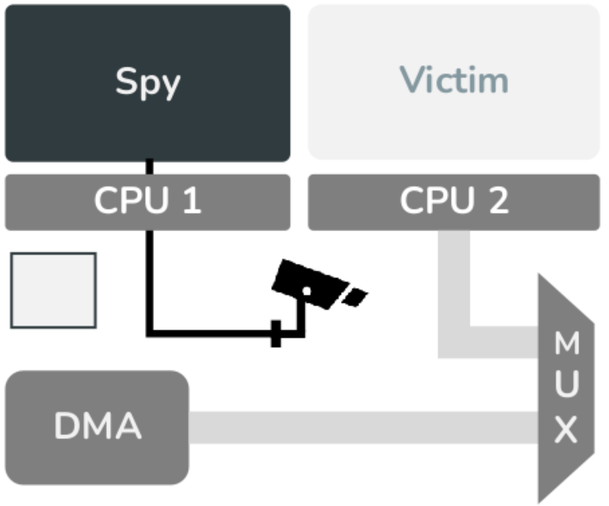


Always Spying

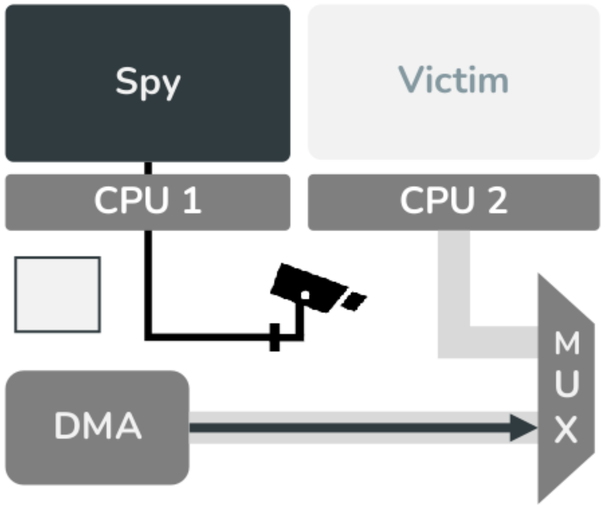
Challenges

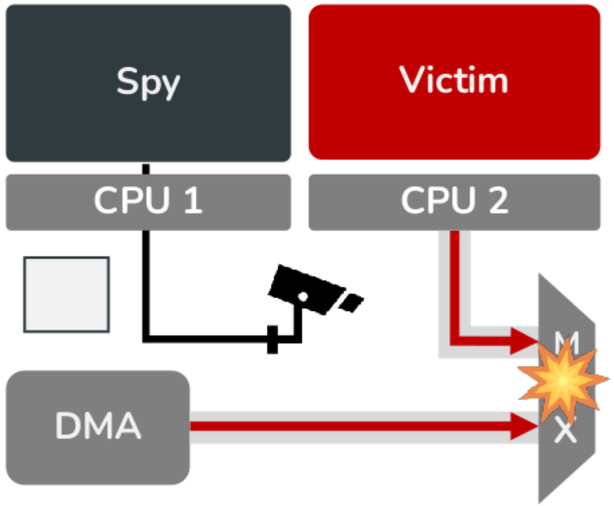

Spy
Victim
CPU 1
CPU 2
DMA
MUX

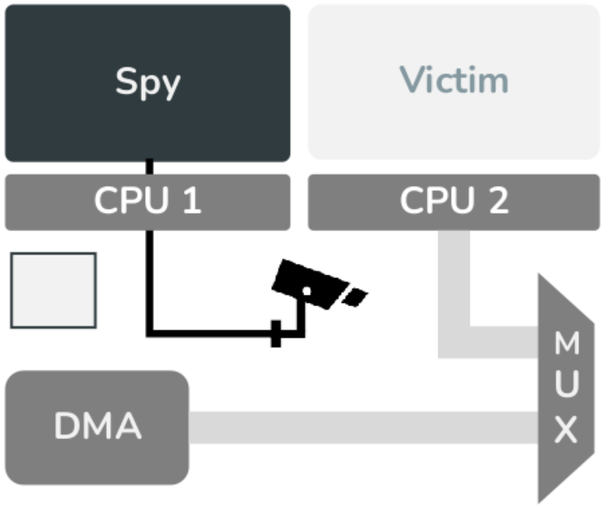

Spy
Victim
CPU 1
CPU 2
DMA
MUX


Spy
Victim
CPU 1
CPU 2
DMA
MUX

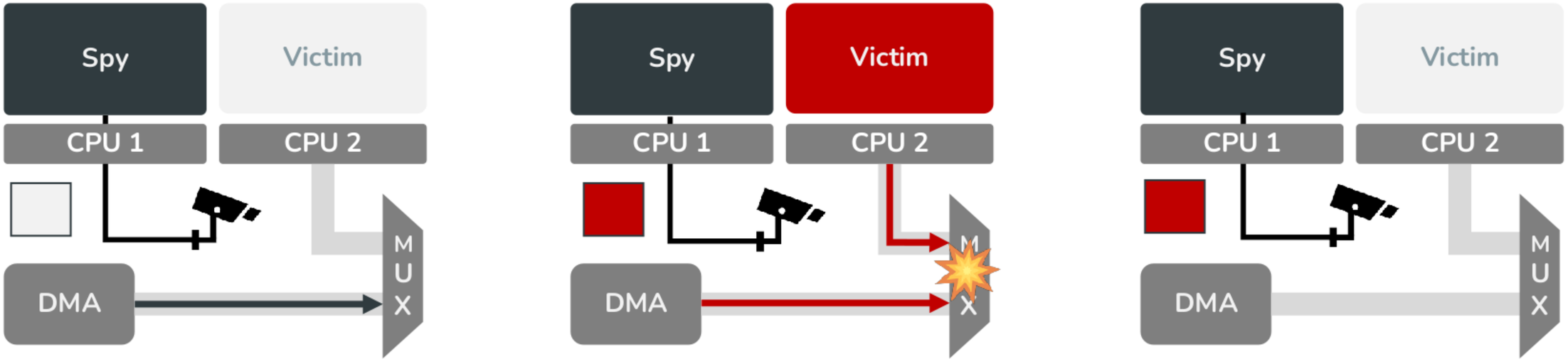
Challenges


Spy
Victim
CPU 1
CPU 2
DMA
MUX

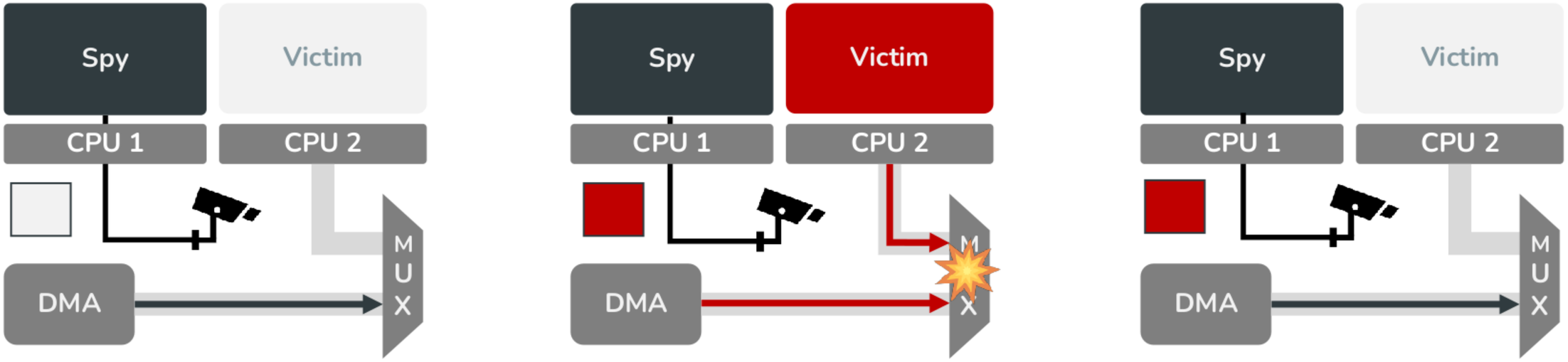

Spy
Victim
CPU 1
CPU 2
DMA
MUX


Spy
Victim
CPU 1
CPU 2
DMA
MUX

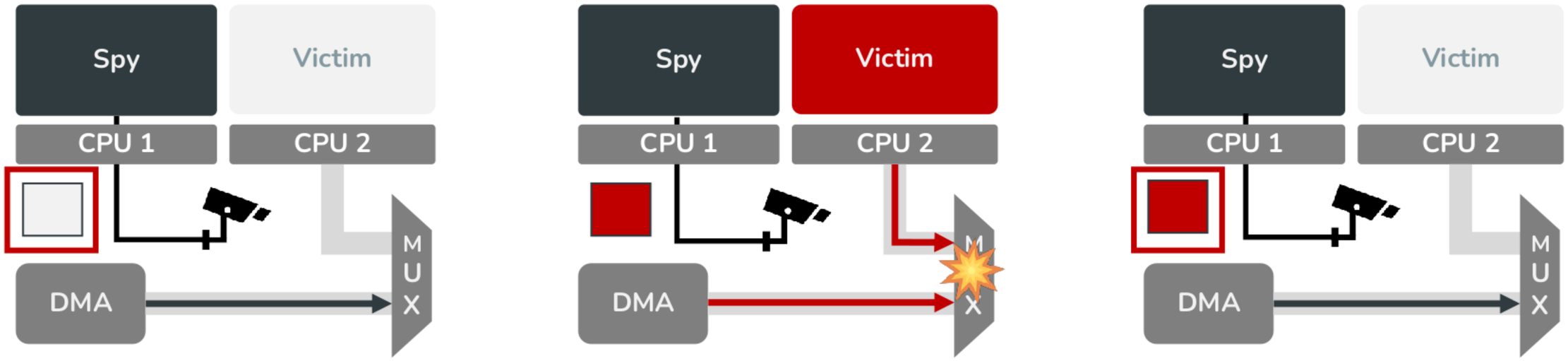
Challenges


Spy
Victim
CPU 1
CPU 2
DMA
MUX
Spy
Victim
CPU 1
CPU 2
DMA
MUX
Spy
Victim
CPU 1
CPU 2
DMA
MUX

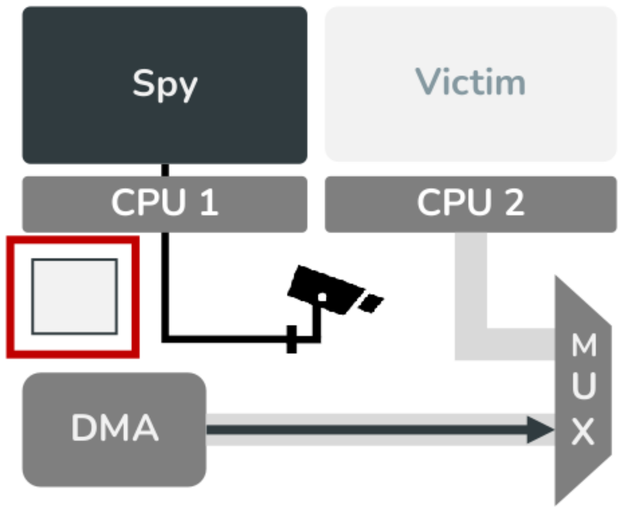
Challenges

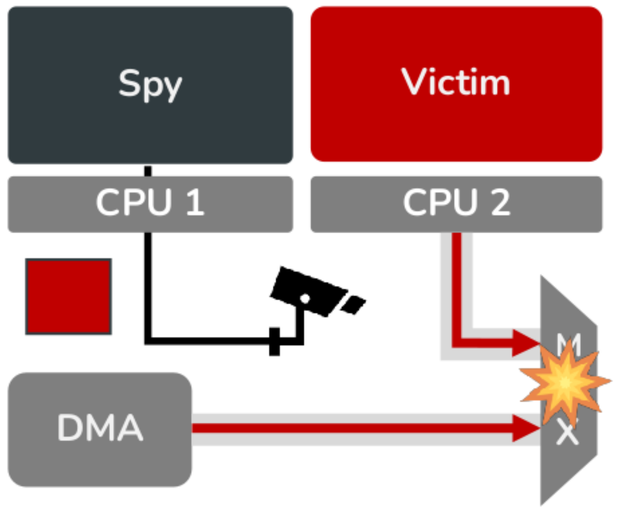

Spy
Victim
CPU 1
CPU 2
DMA
MUX
Spy
Victim
CPU 1
CPU 2
DMA
MUX
Spy
Victim
CPU 1
CPU 2
DMA
MUX

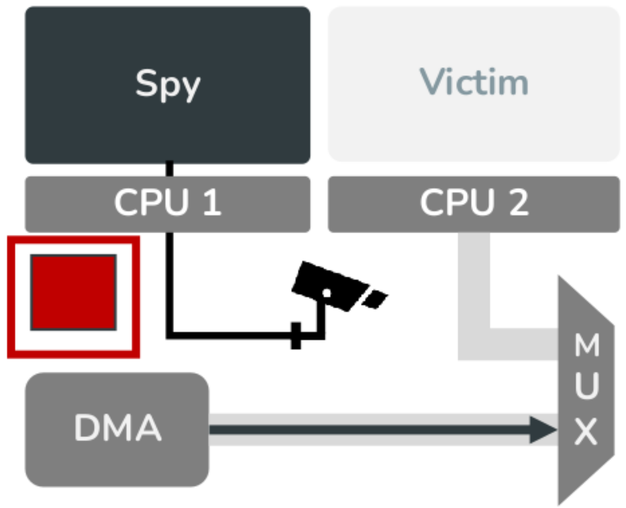
Challenges


Spy
Victim
CPU 1
CPU 2
DMA
MUX
Spy
Victim
CPU 1
CPU 2
DMA
MUX
Spy
Victim
CPU 1
CPU 2
DMA
MUX

Challenges

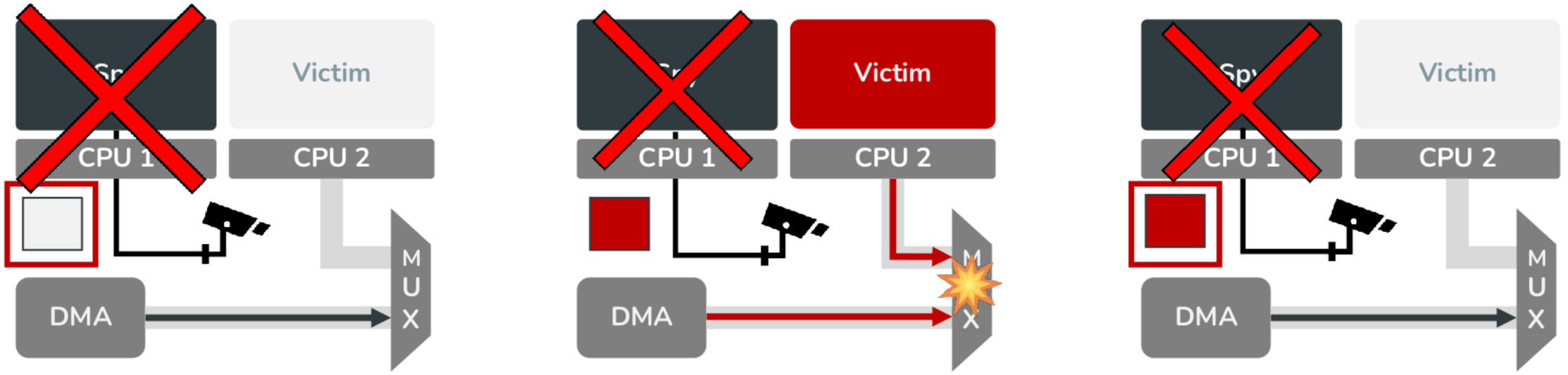






Different States

Challenges



Single-Core MCU!!!

Challenges

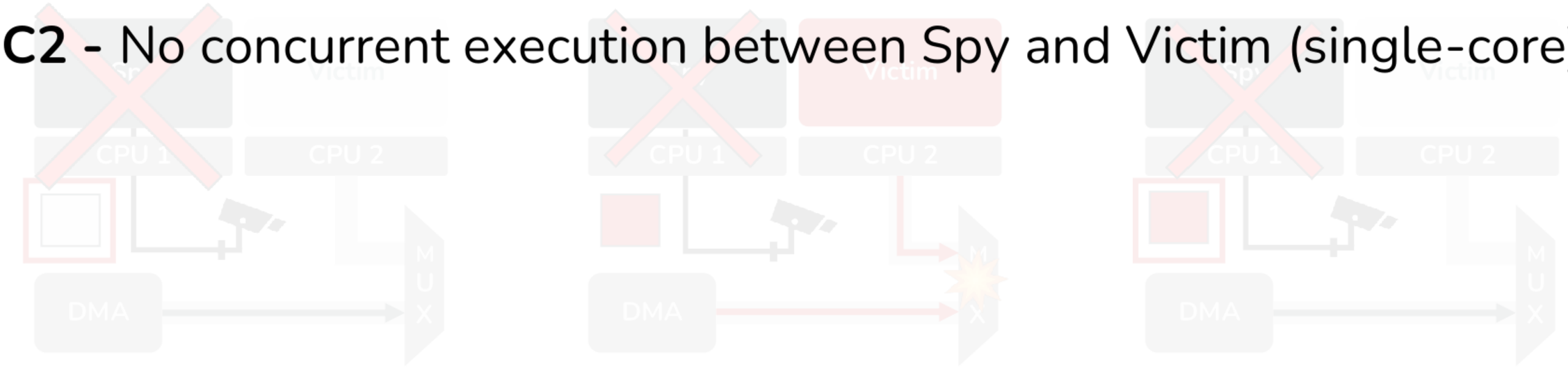
C1 - The bus is a stateless component;

Single-Core MCU!!!

Challenges

C1 - The bus is a stateless component;

C2 - No concurrent execution between Spy and Victim (single-core);



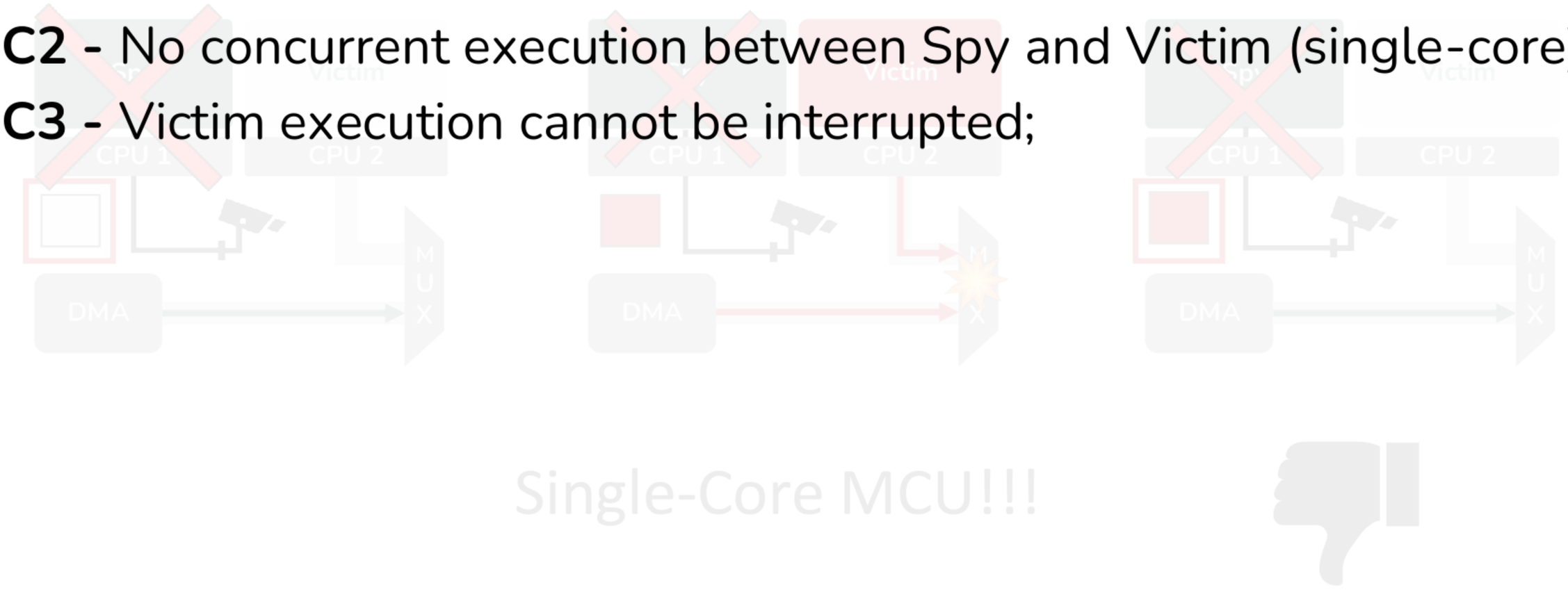
Single-Core MCU!!!

Challenges

C1 - The bus is a stateless component;

C2 - No concurrent execution between Spy and Victim (single-core);

C3 - Victim execution cannot be interrupted;



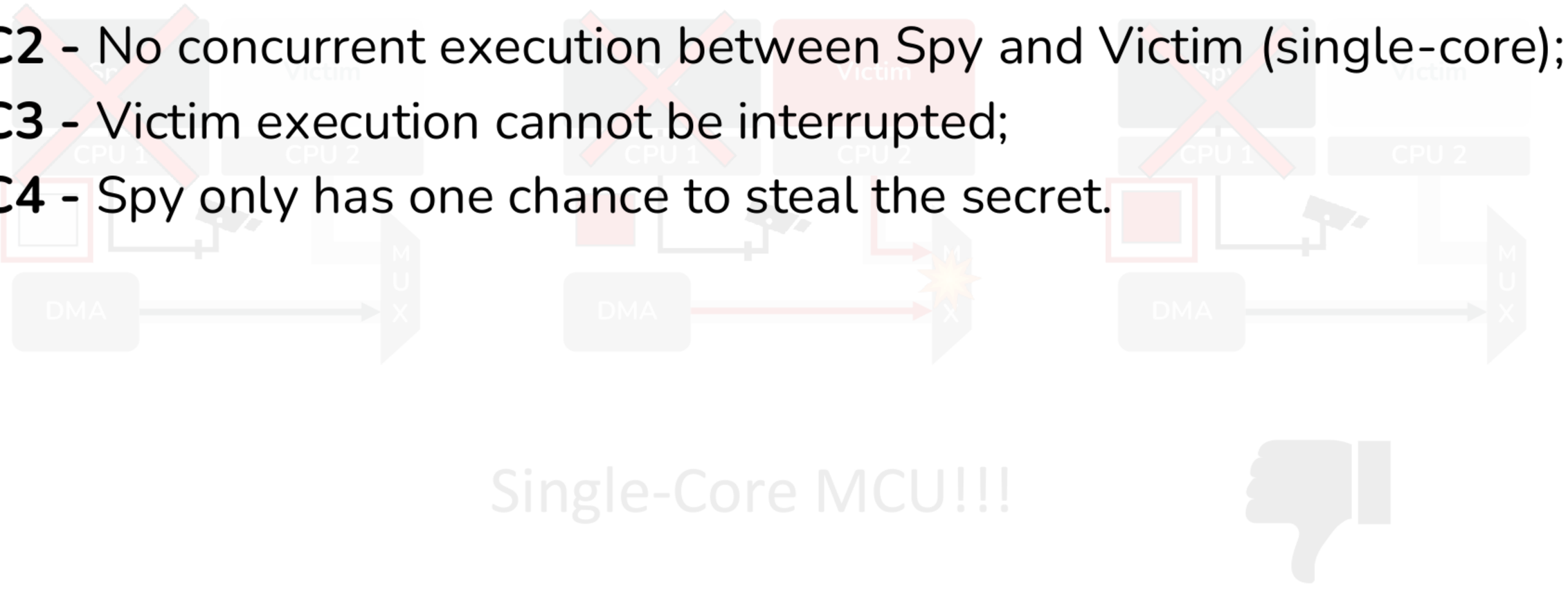
Challenges

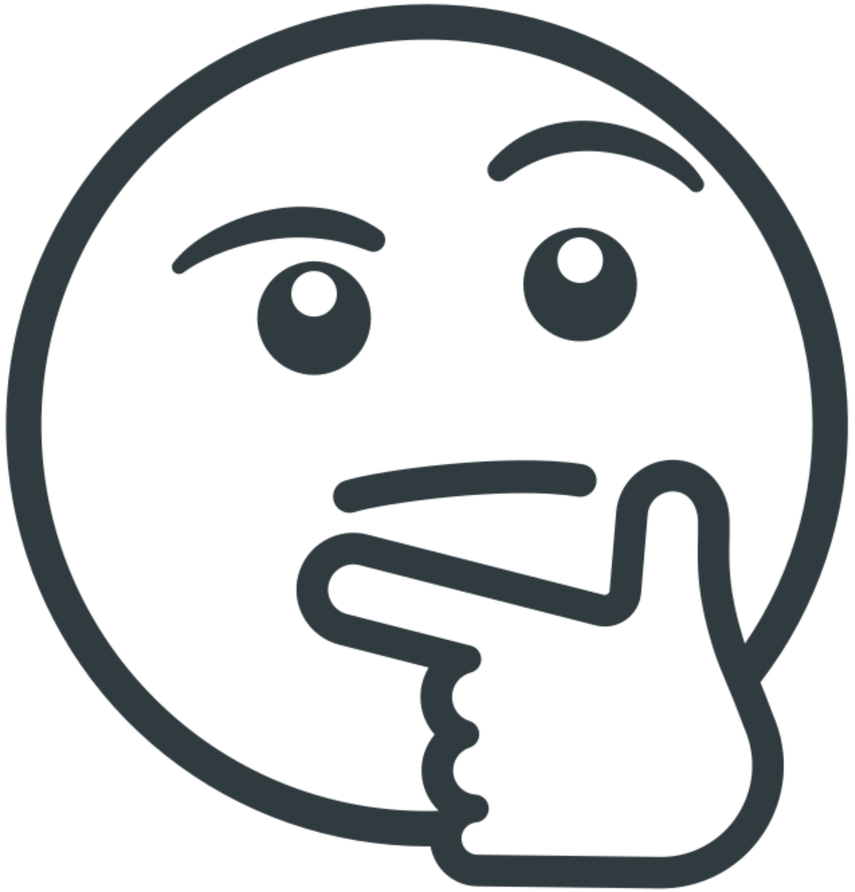
C1 - The bus is a stateless component;

C2 - No concurrent execution between Spy and Victim (single-core);

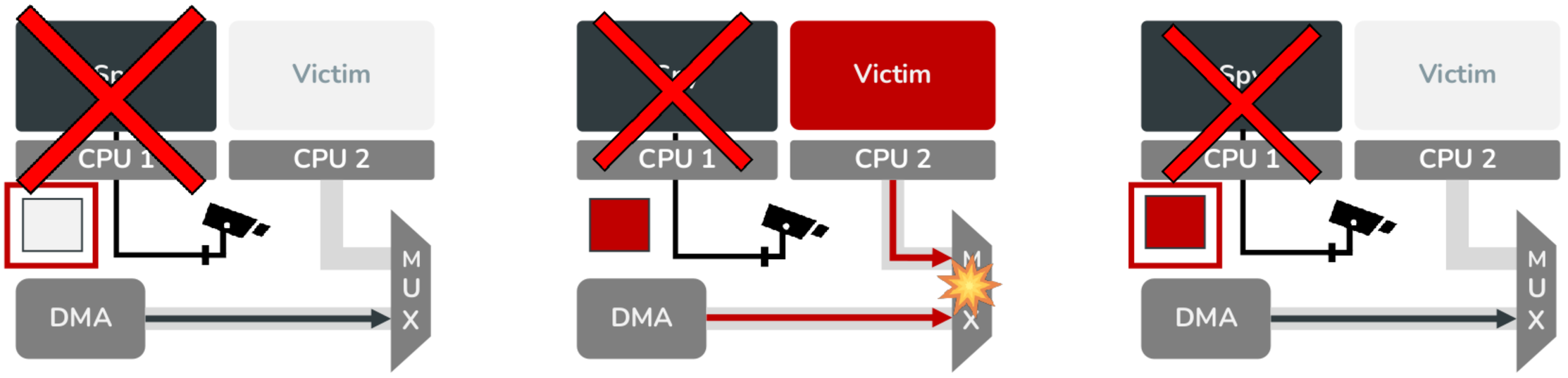
C3 - Victim execution cannot be interrupted;

C4 - Spy only has one chance to steal the secret.



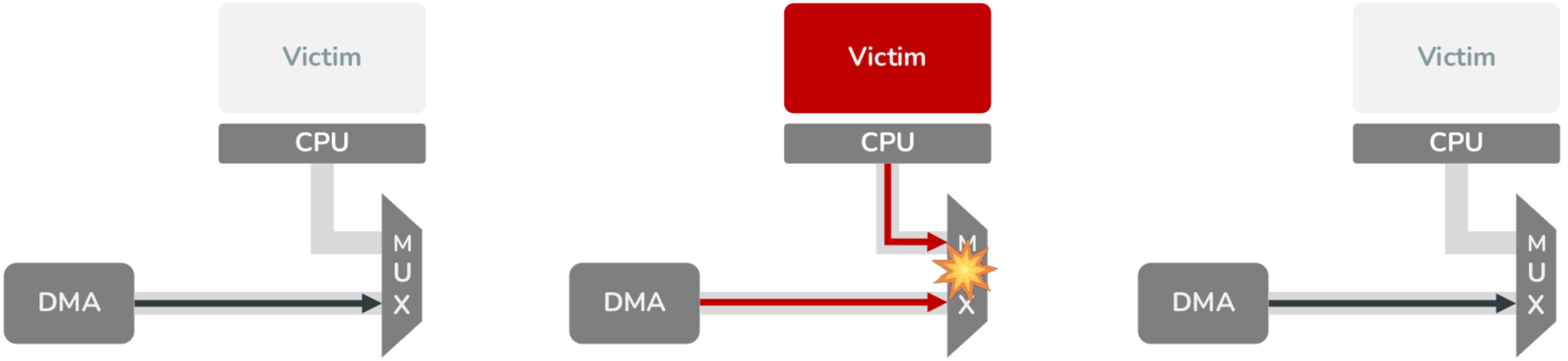


Solution



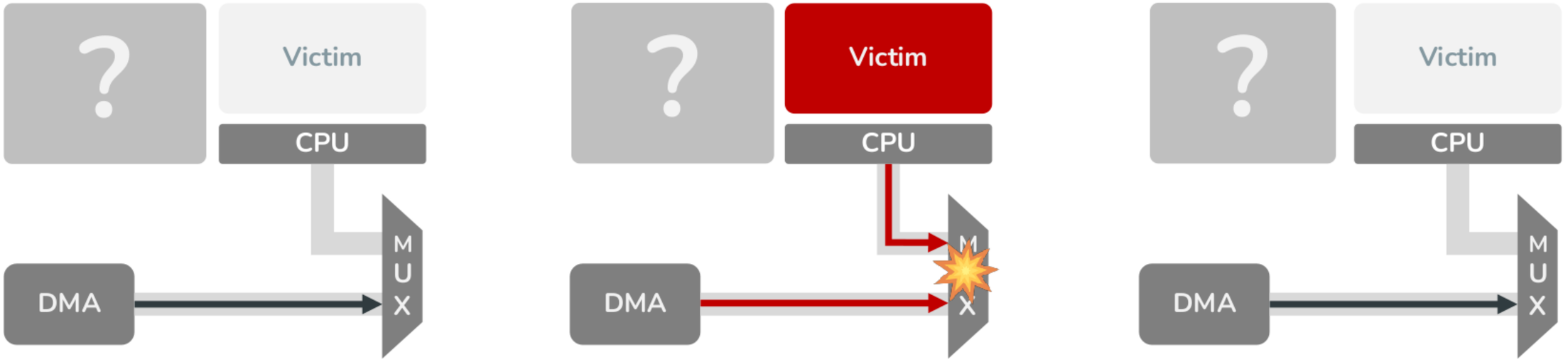
Single-Core MCU!!!

Solution



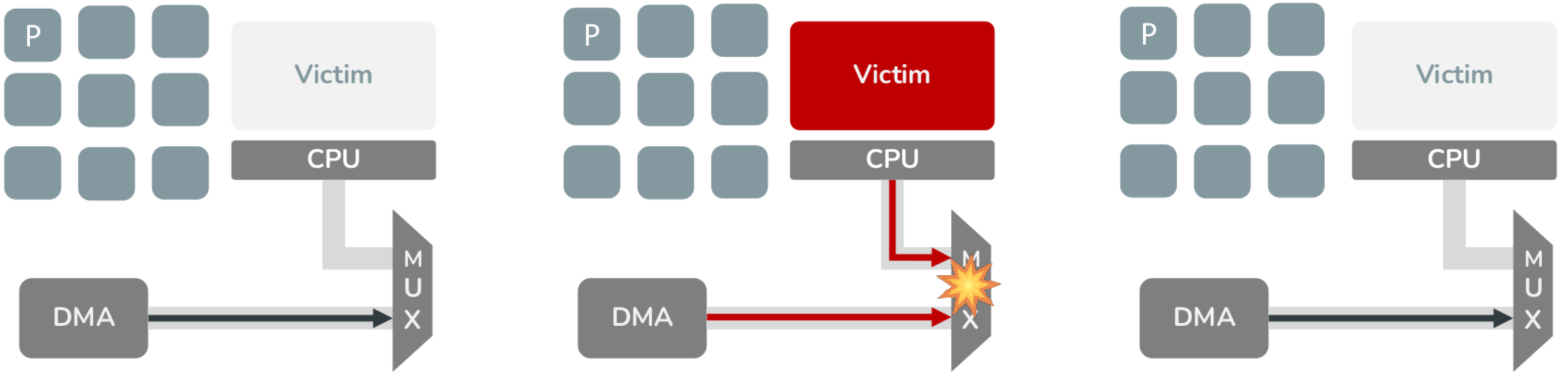
Single-Core MCU!!!

Solution



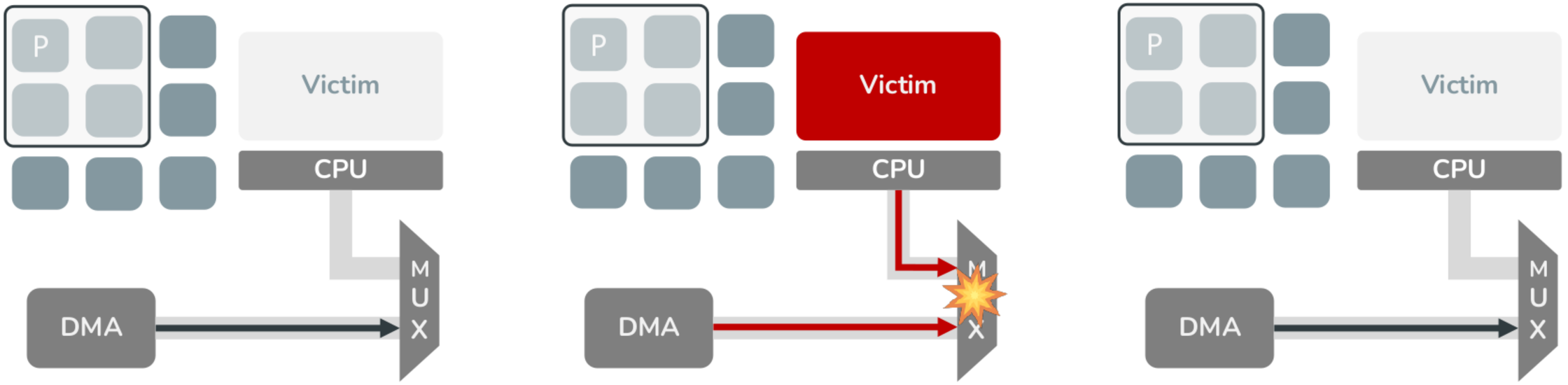
Single-Core MCU!!!

Solution



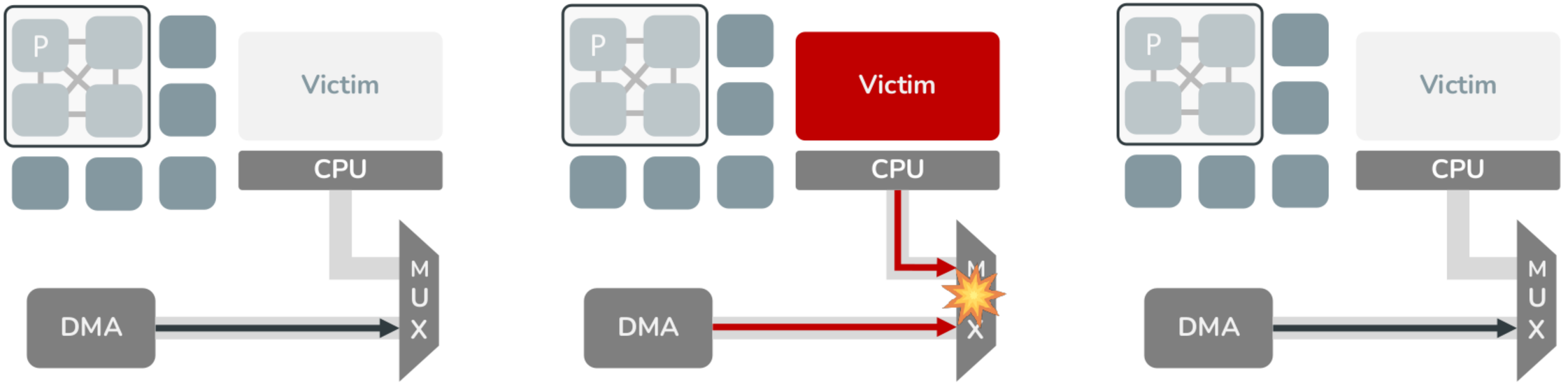
MCUs Have a lot of Peripherals!!

Solution



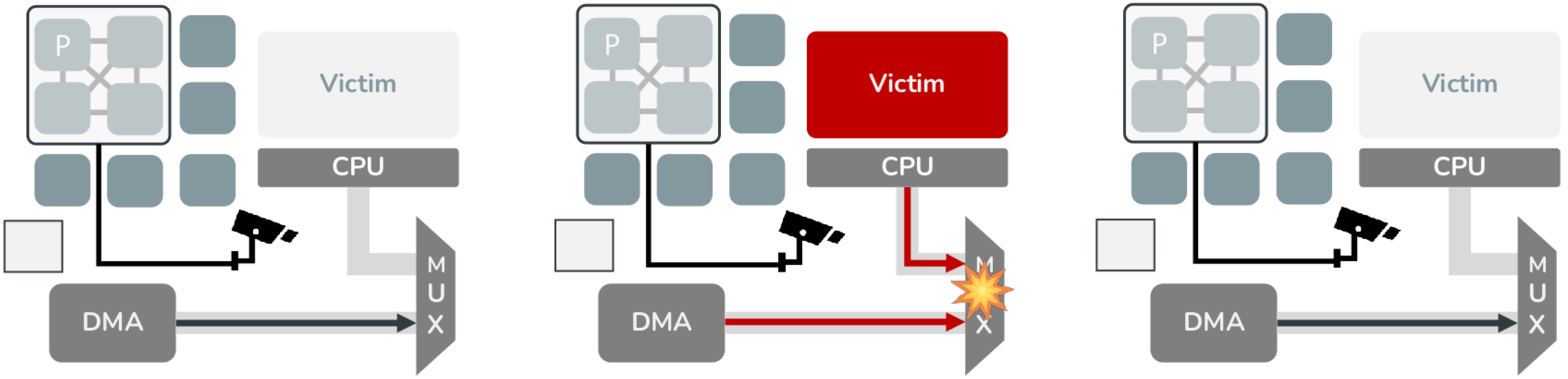
Group Some Peripherals!!

Solution



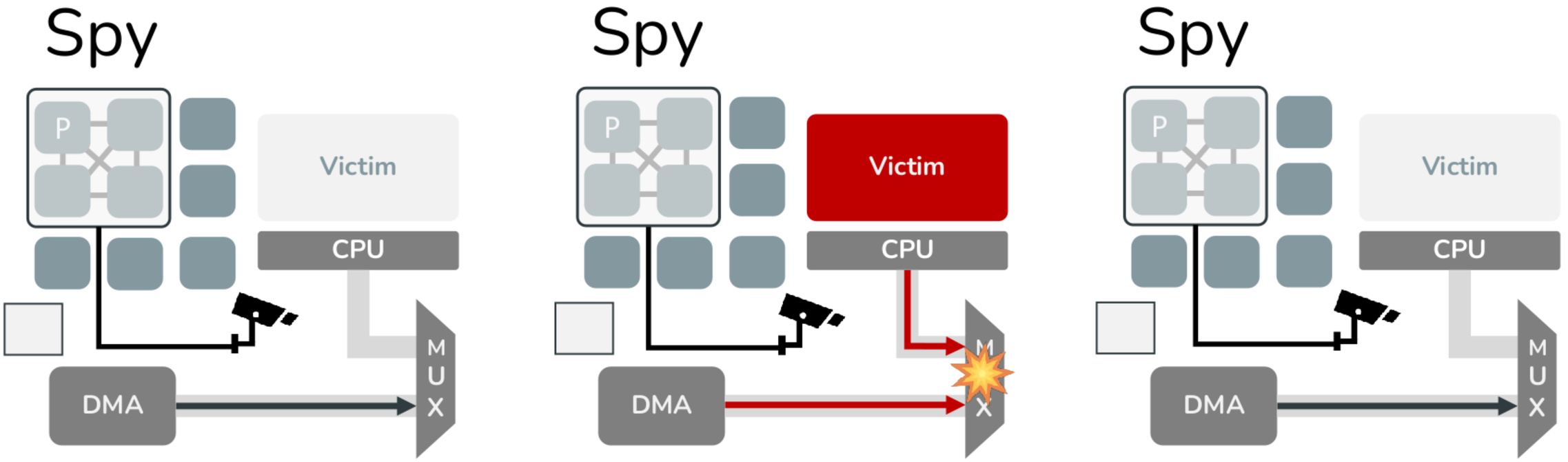
Interconnect the Peripherals!!

Solution



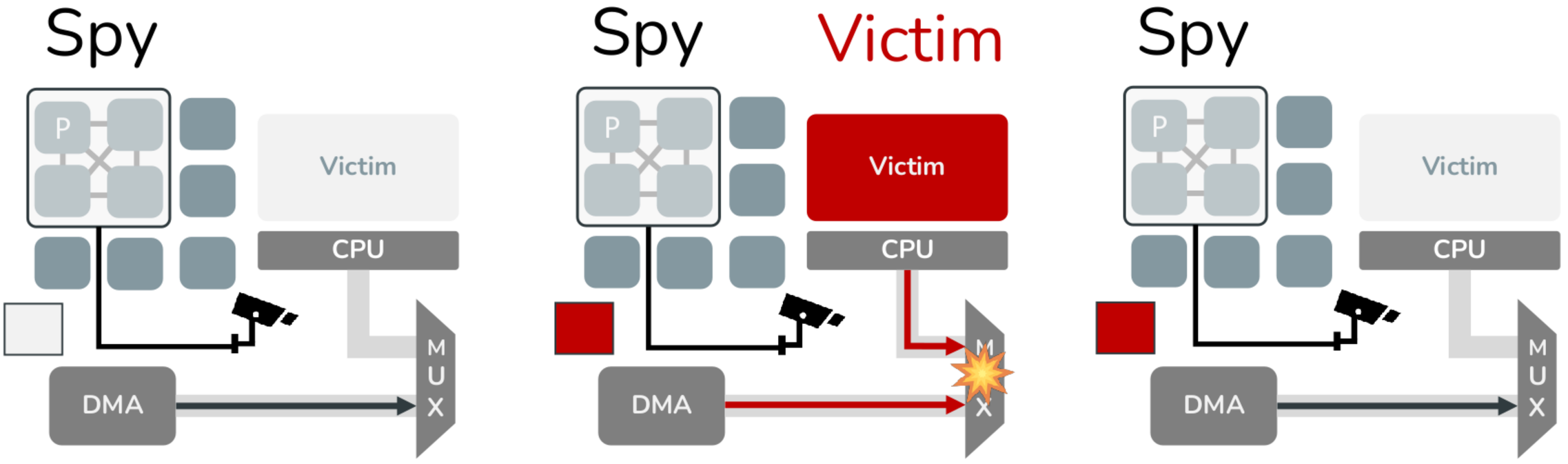
Voilà!!! Hardware Gadgets!

Solution



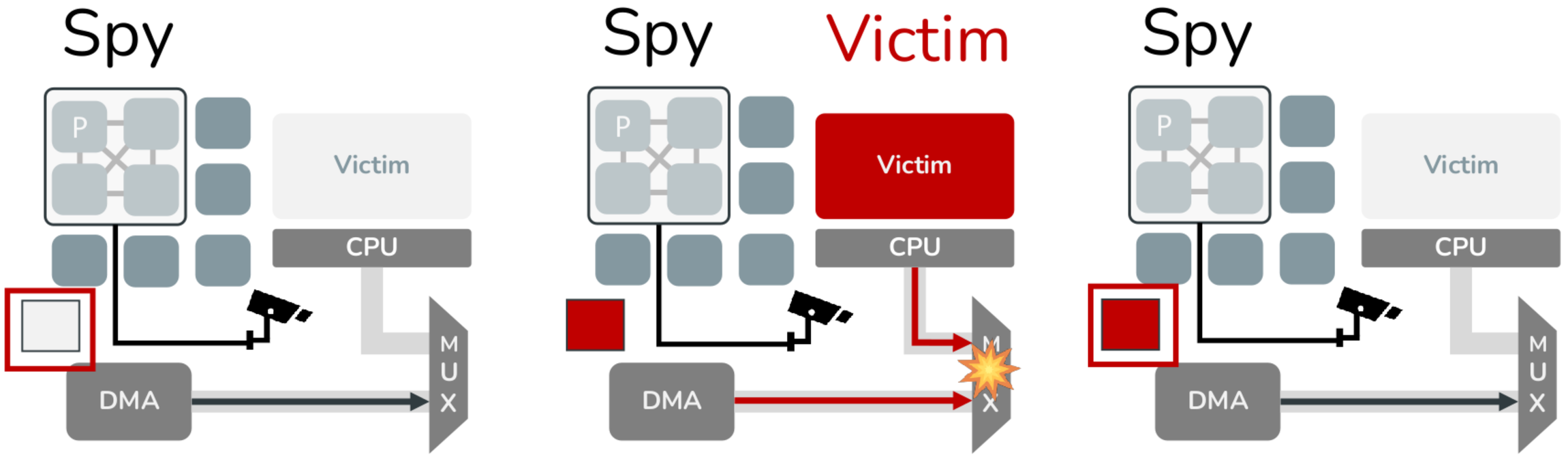
Always Spying

Solution



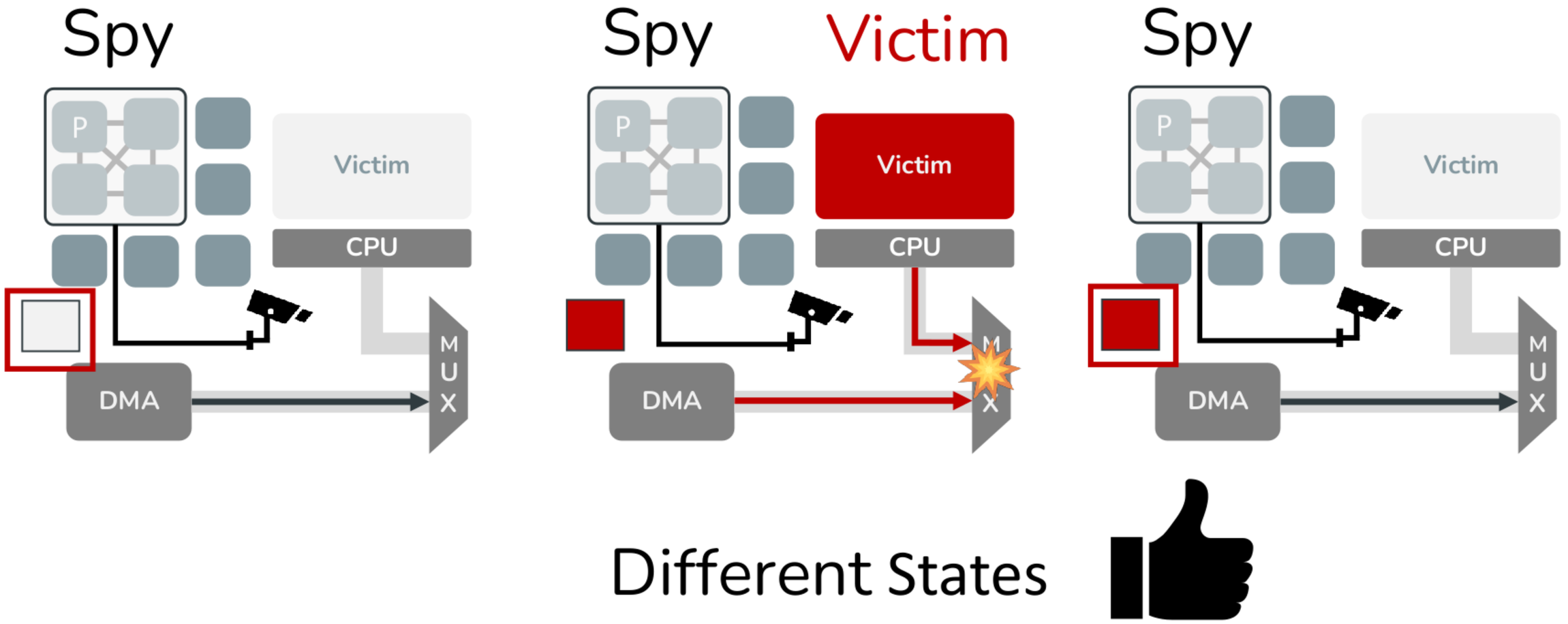
Always Spying

Solution



Always Spying

Solution



Different States

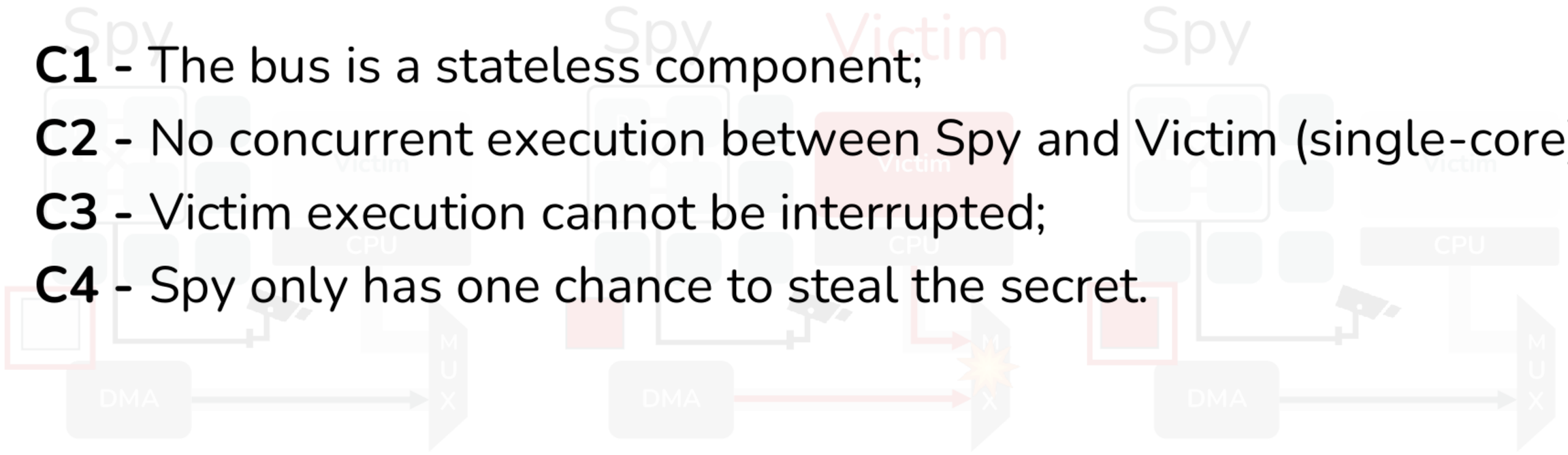
Challenges

C1 - The bus is a stateless component;

C2 - No concurrent execution between Spy and Victim (single-core);

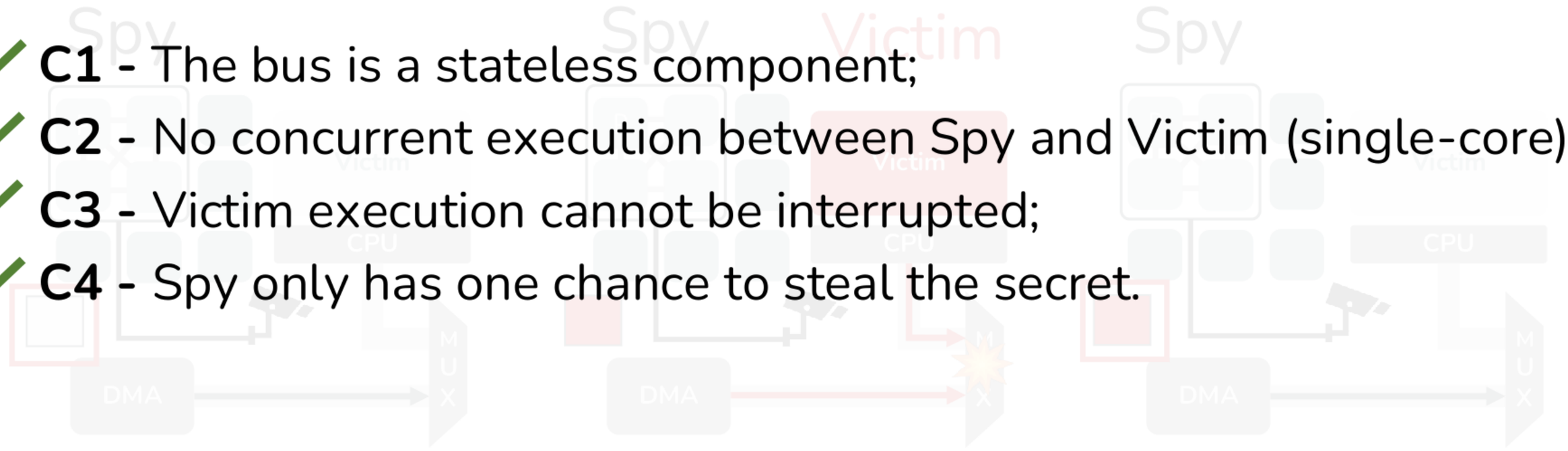
C3 - Victim execution cannot be interrupted;

C4 - Spy only has one chance to steal the secret.

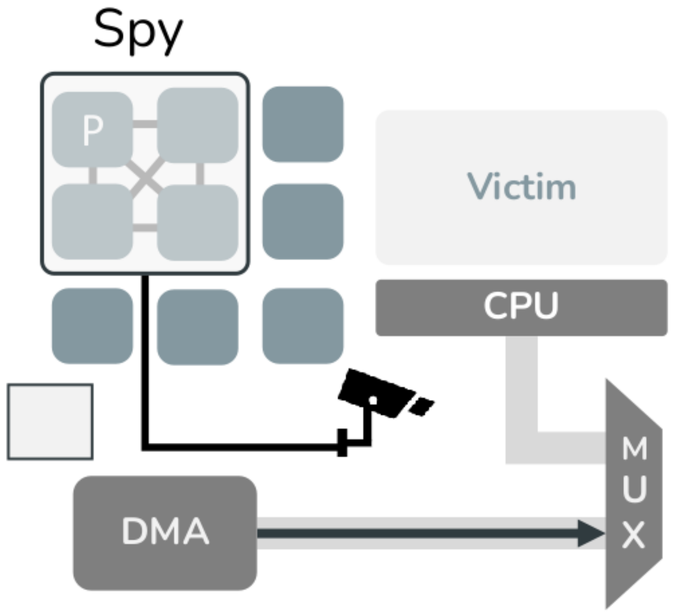


Challenges

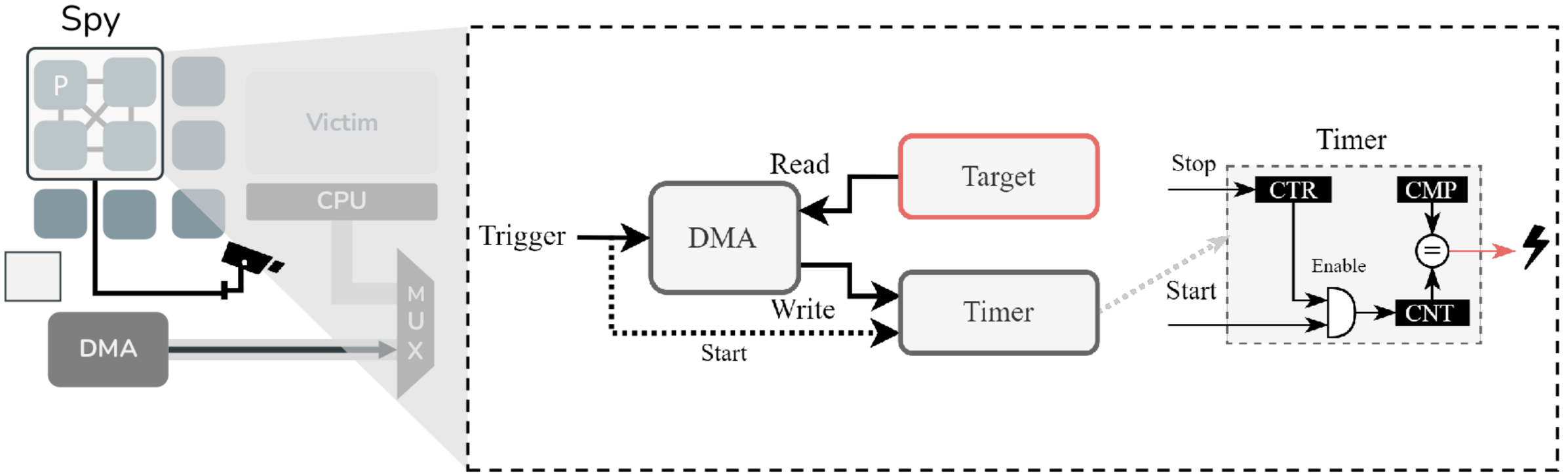
- ✓ **C1 -** The bus is a stateless component;
- ✓ **C2 -** No concurrent execution between Spy and Victim (single-core);
- ✓ **C3 -** Victim execution cannot be interrupted;
- ✓ **C4 -** Spy only has one chance to steal the secret.



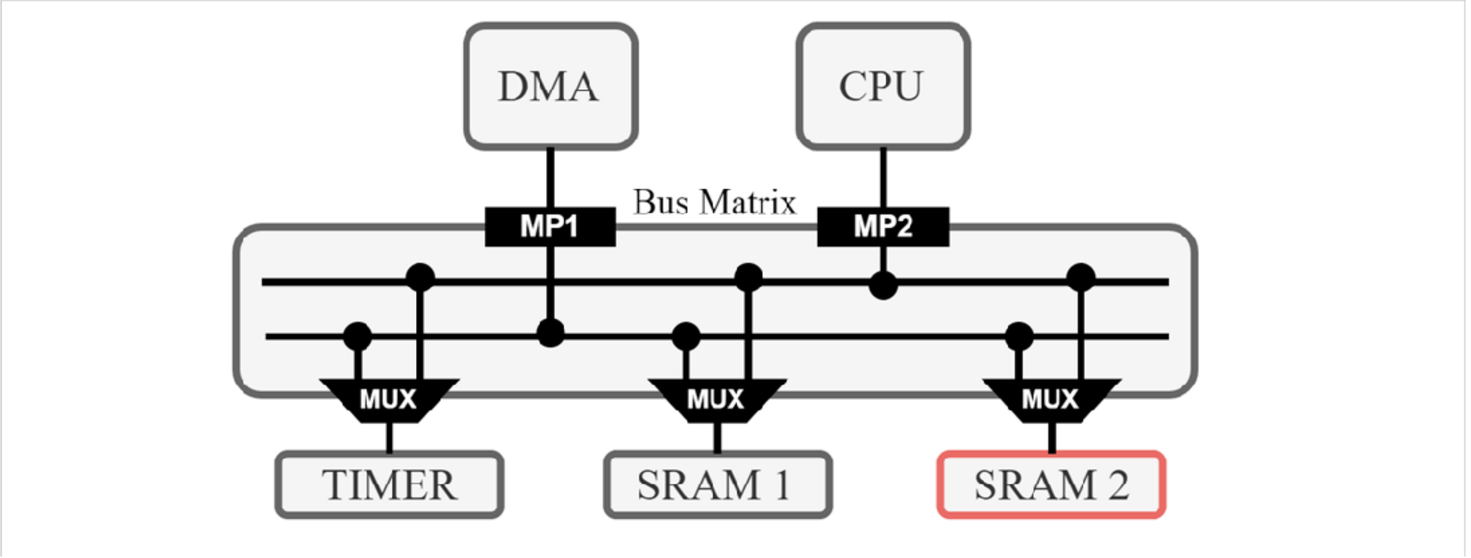
Hardware Gadgets



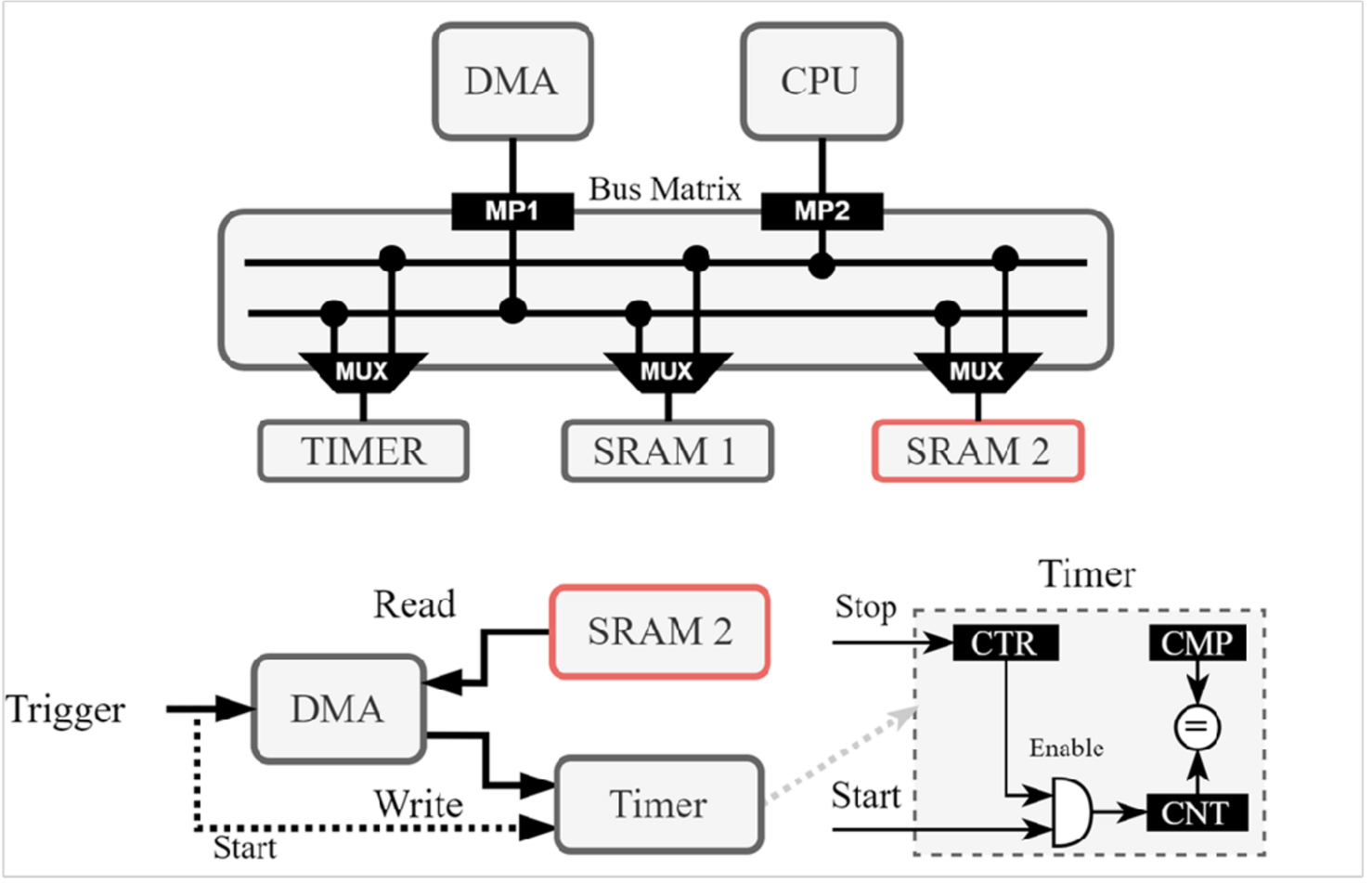
Hardware Gadgets



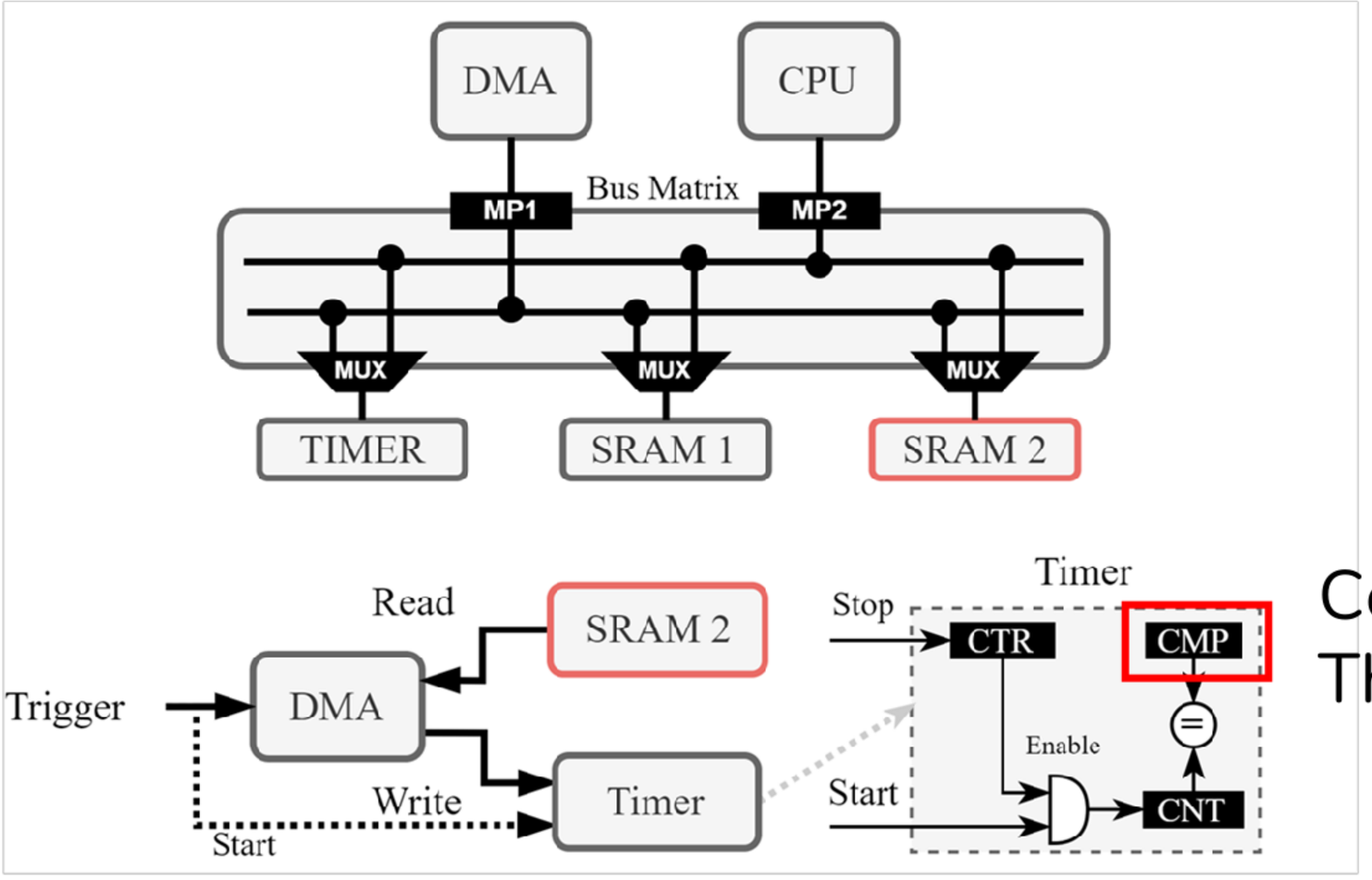
Hardware Gadgets



Hardware Gadgets

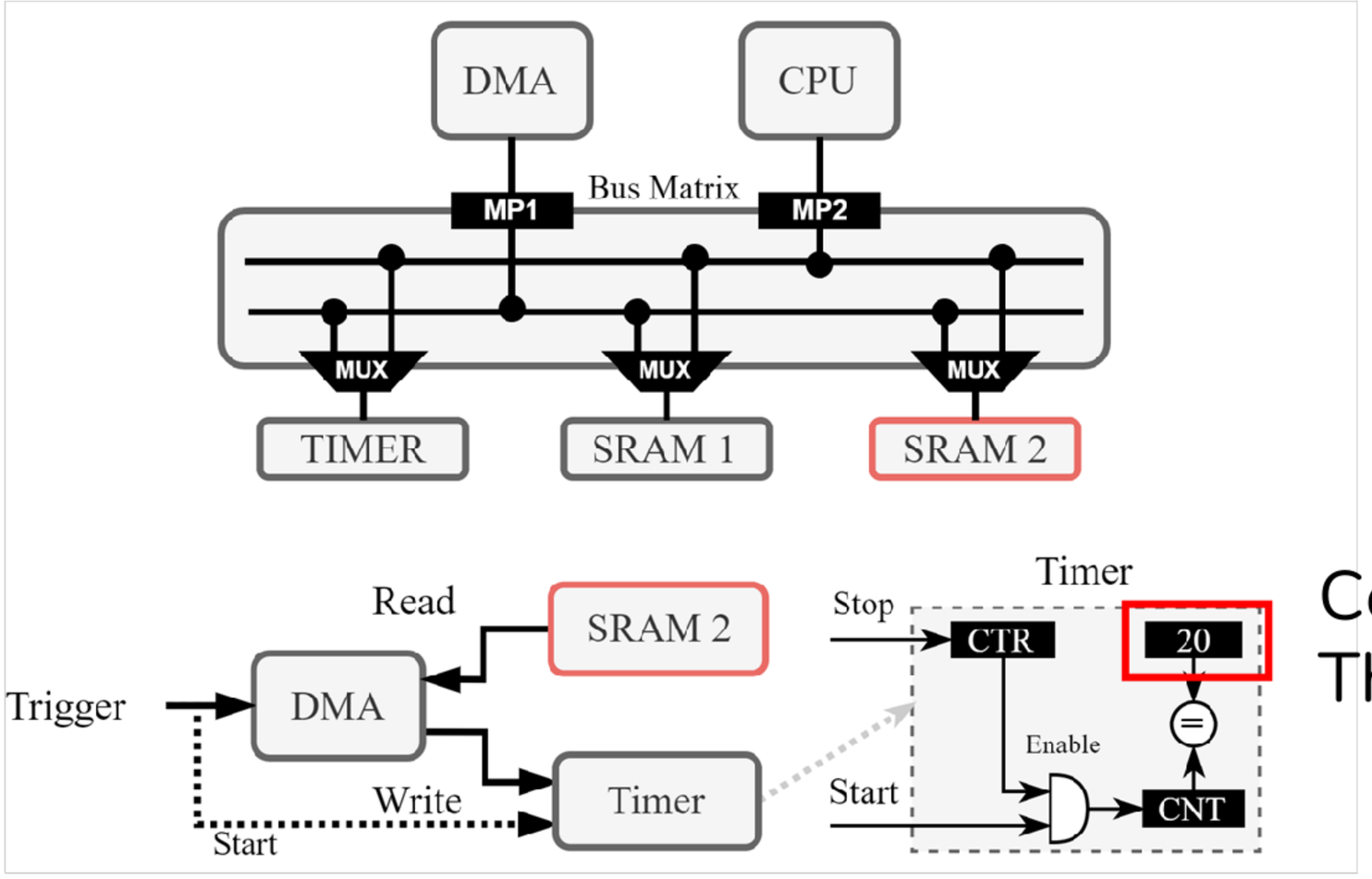


Hardware Gadgets



Contention Threshold

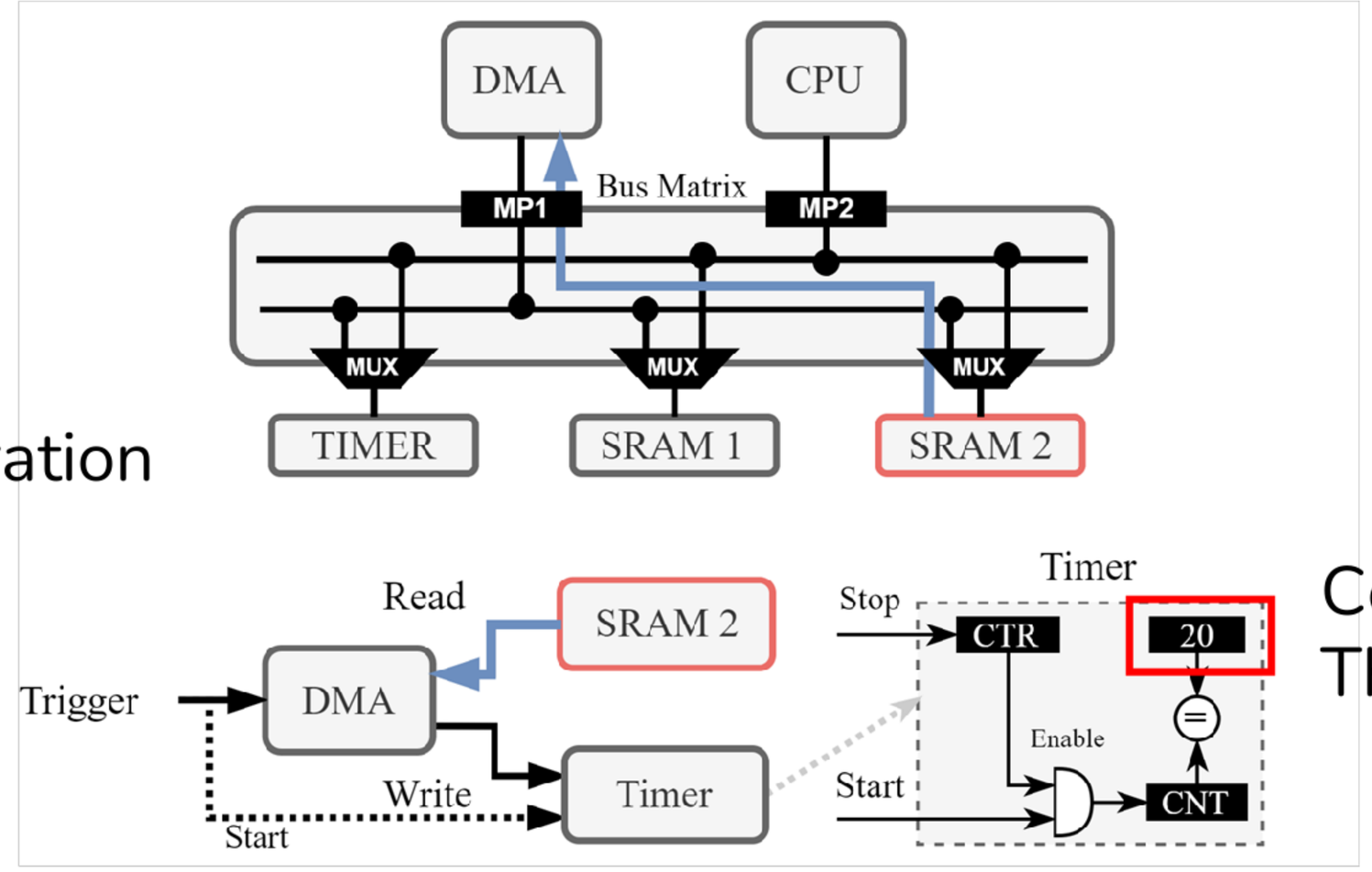
Hardware Gadgets



Contention Threshold

Hardware Gadgets

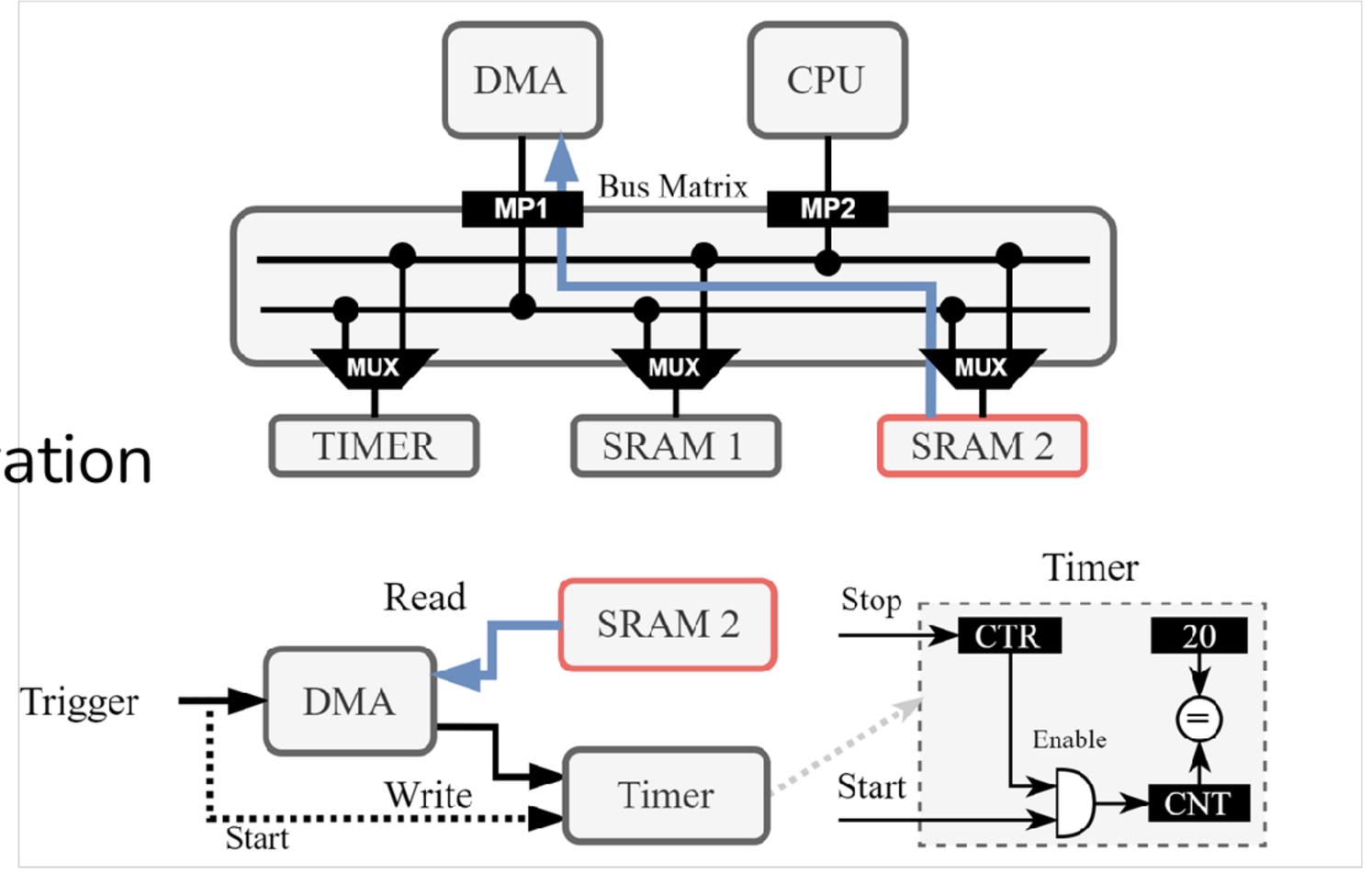
Normal Operation



Contention Threshold

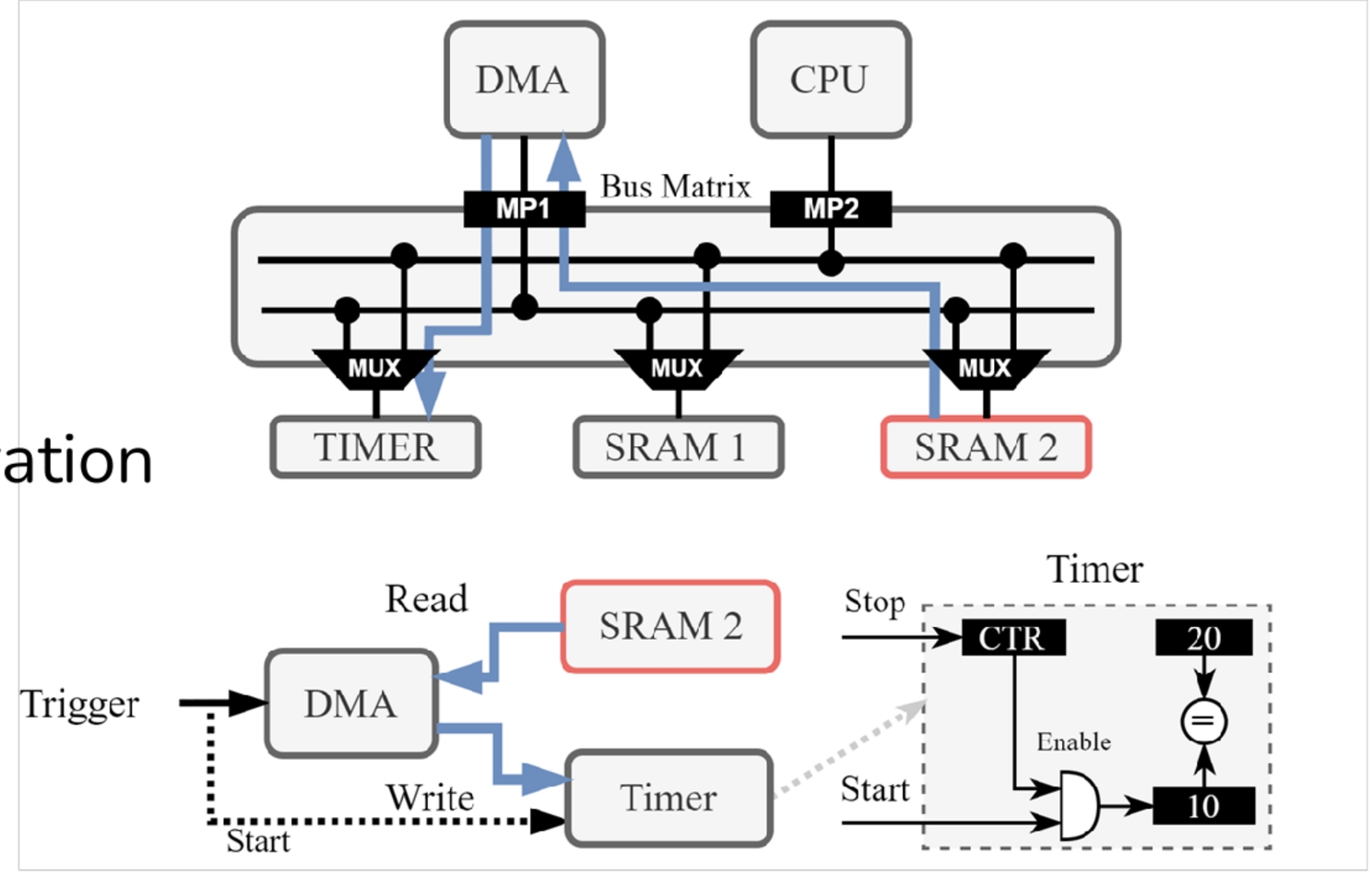
Hardware Gadgets

Normal Operation



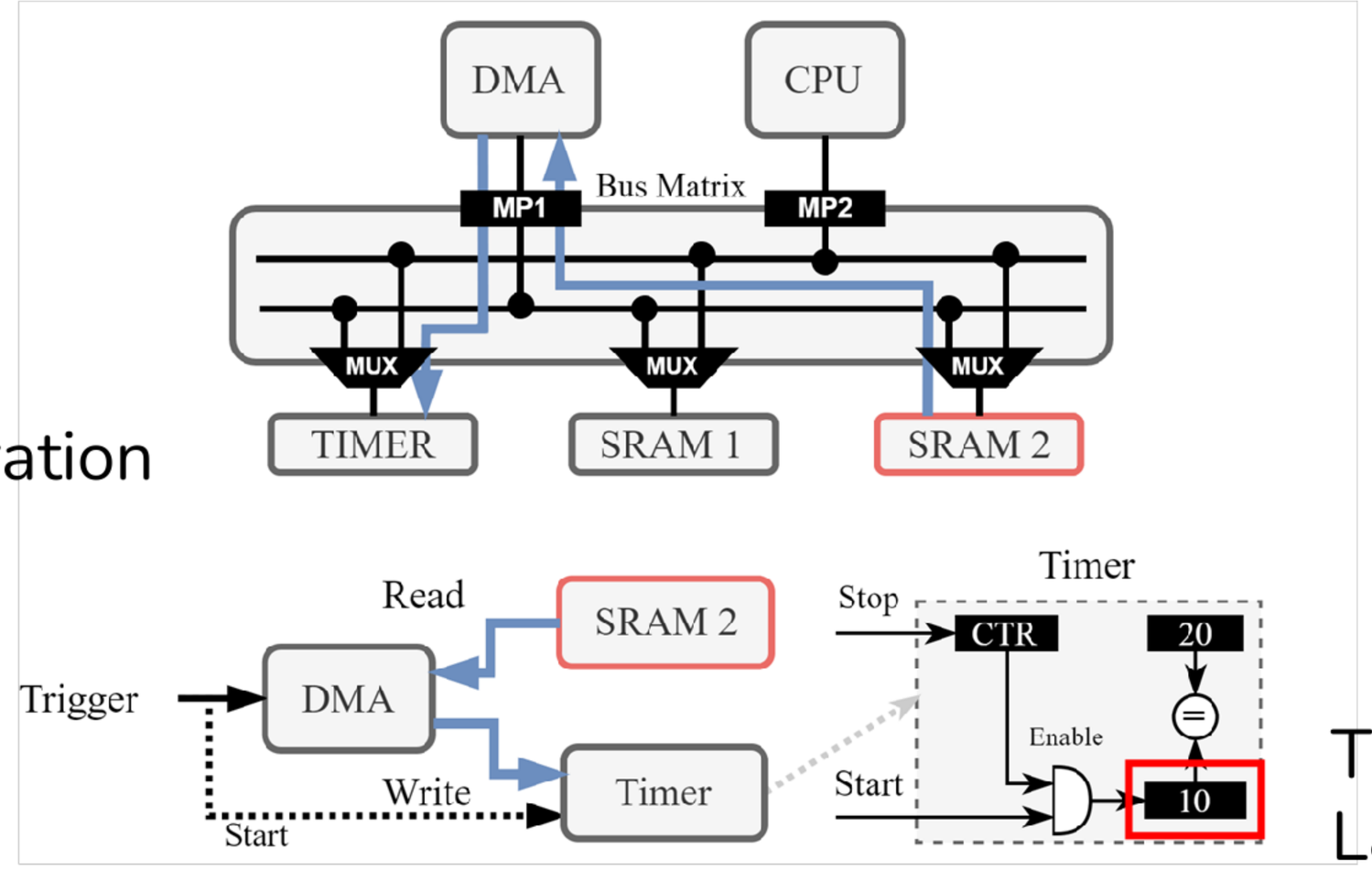
Hardware Gadgets

Normal Operation



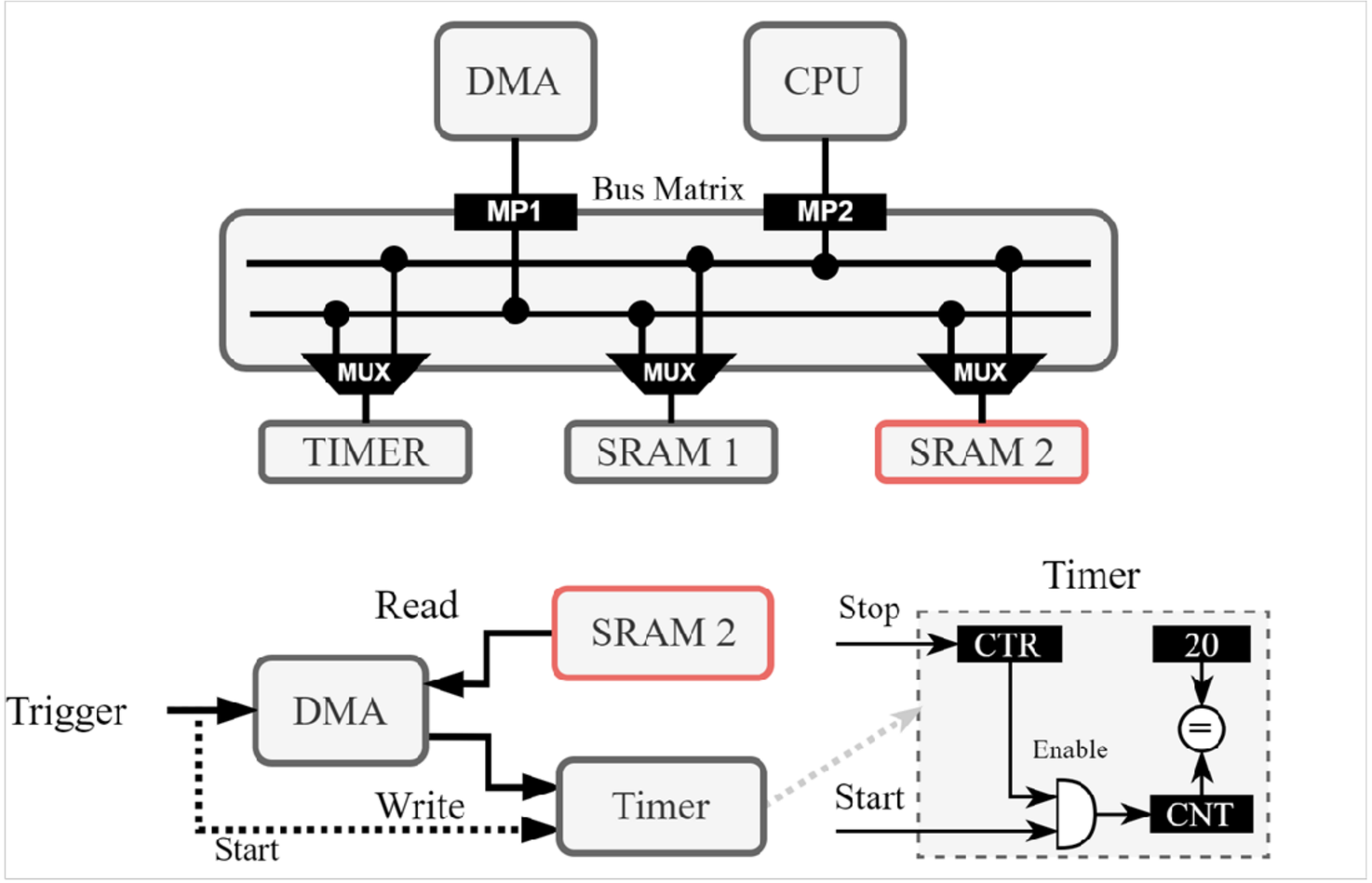
Hardware Gadgets

Normal Operation

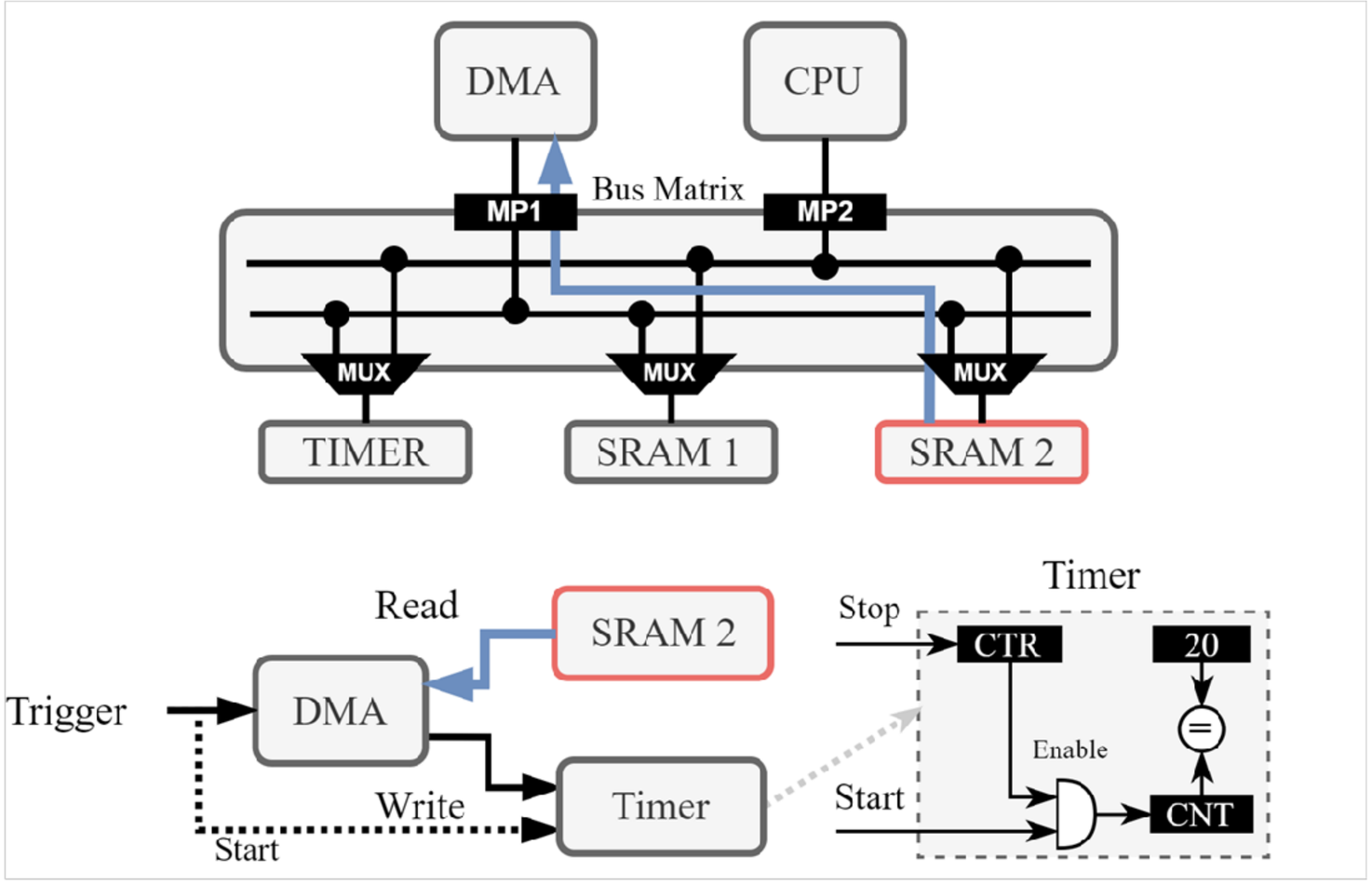


Transfer Latency

Hardware Gadgets

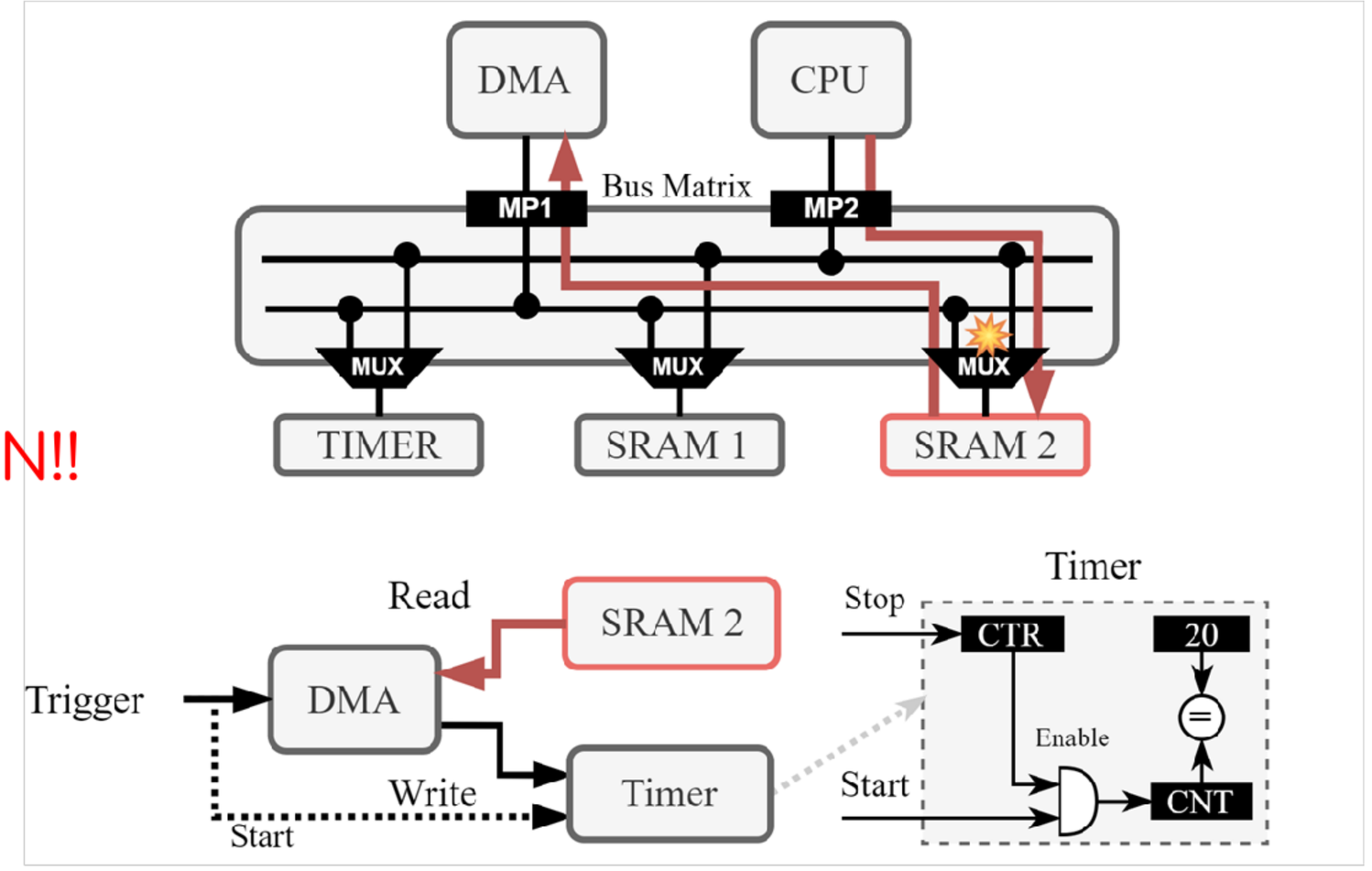


Hardware Gadgets



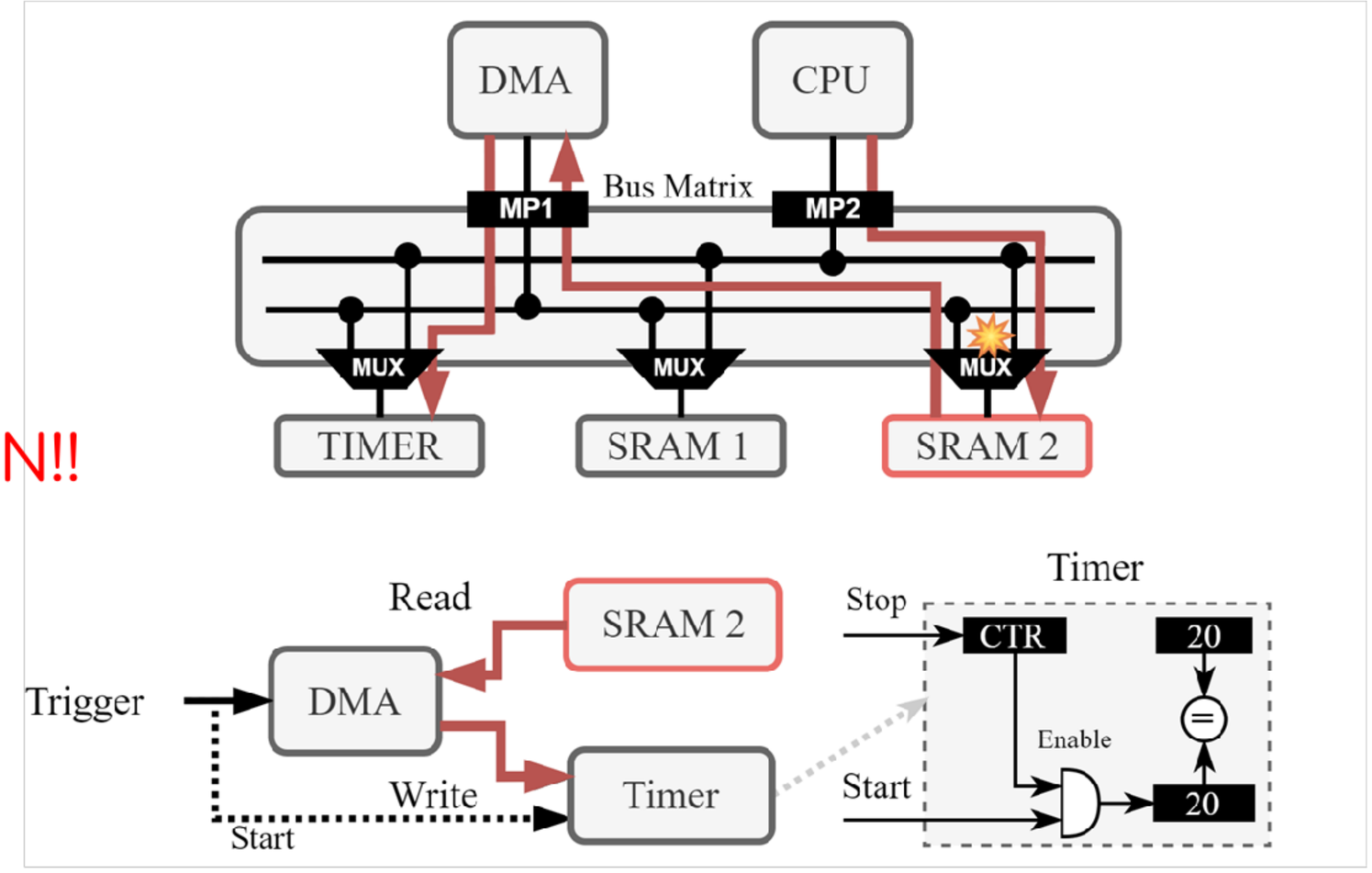
Hardware Gadgets

CONTENTION!!



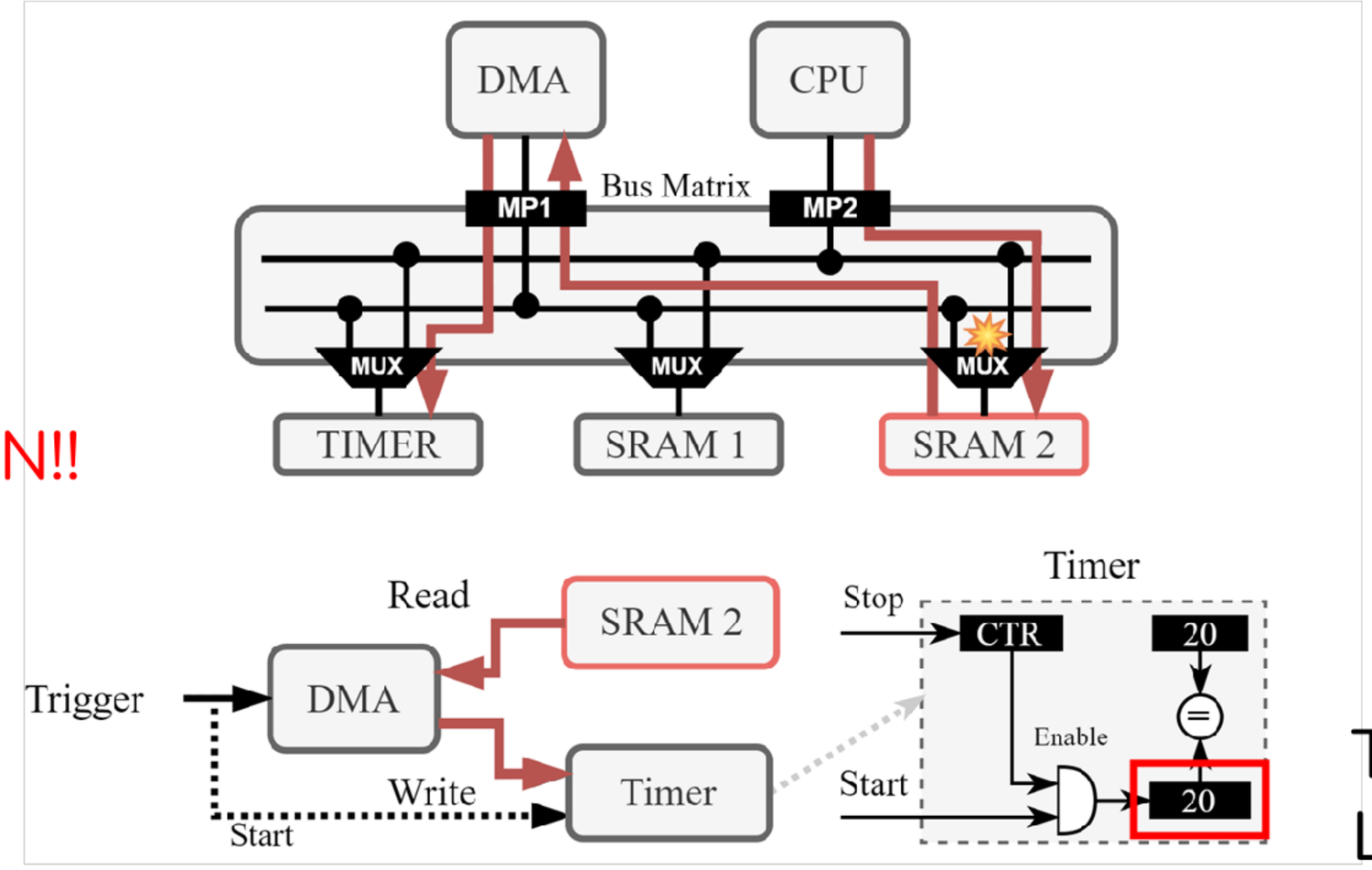
Hardware Gadgets

CONTENTION!!



Hardware Gadgets

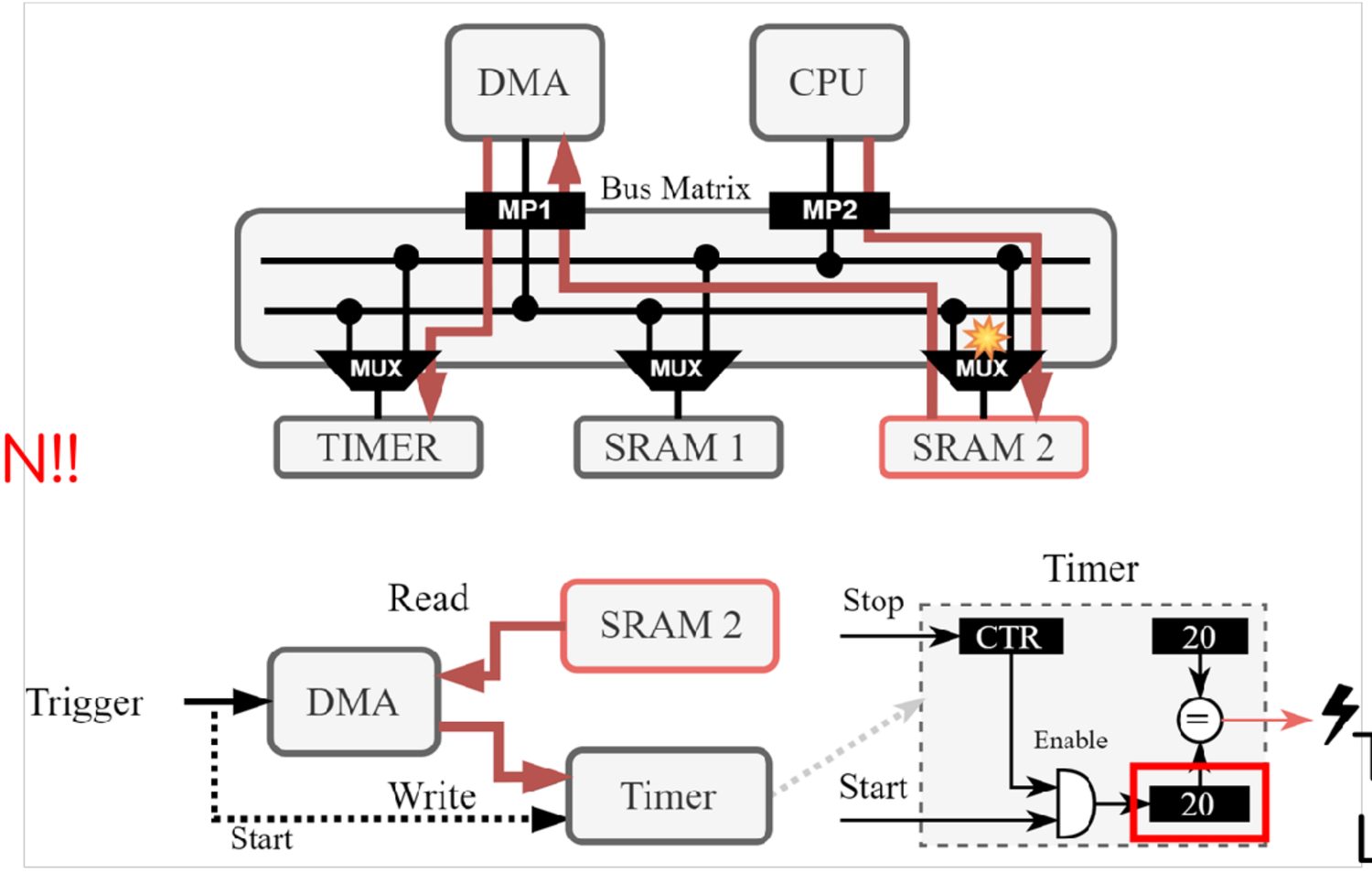
CONTENTION!!



Transfer Latency

Hardware Gadgets

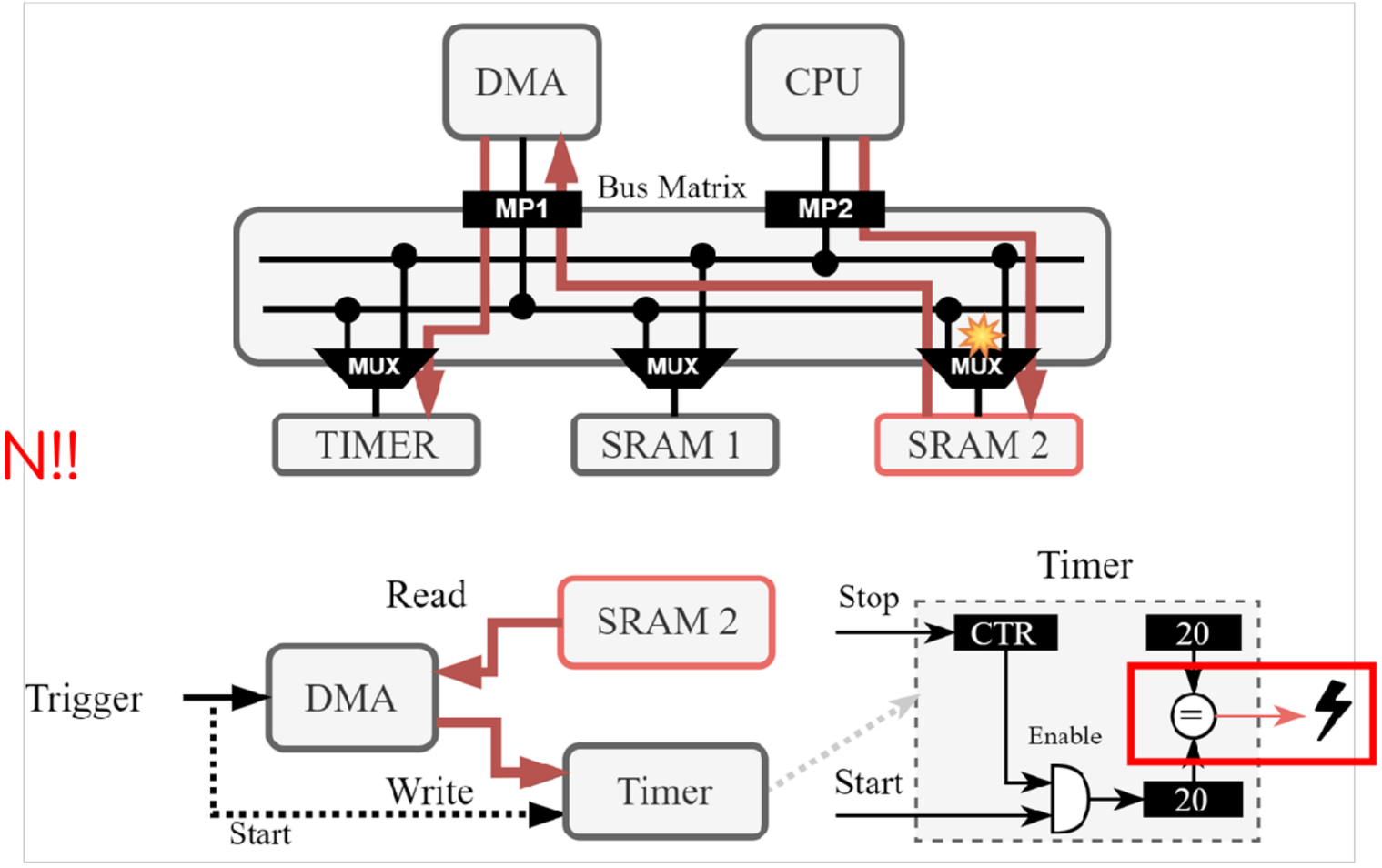
CONTENTION!!



Transfer Latency

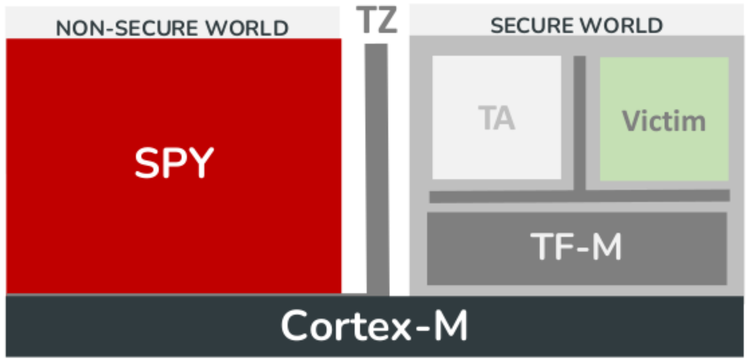
Hardware Gadgets

CONTENTION!!

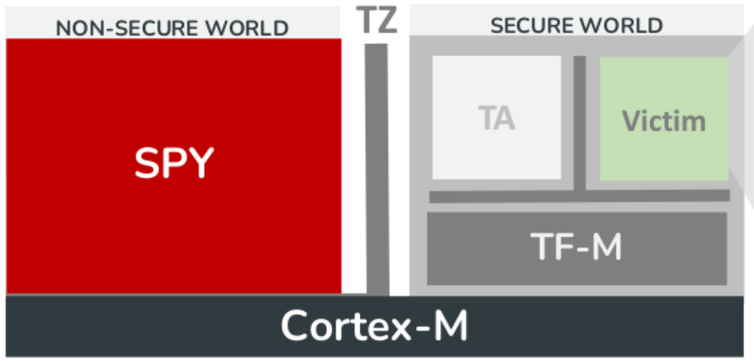


Interrupt (Detection)

BUSted Attack



BUSted Attack

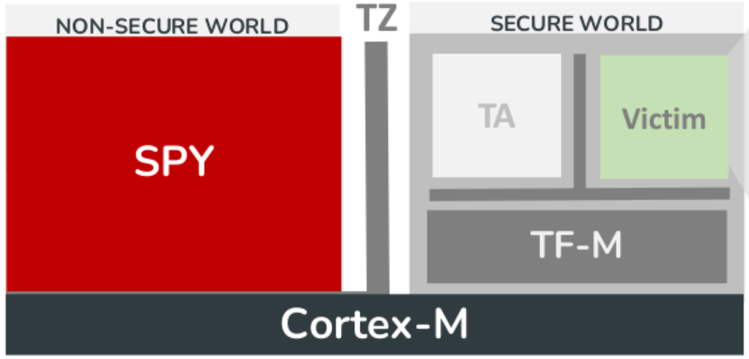



```
signed int read_keypad(void){
    int is_pressed, mask = 0x1;
    int new_key_state = get_keypad_state();
    for (int key = 0; key < KYPD_NB_KEYS; key++){
        is_pressed = (new_key_state & mask) & ~(key_state & mask);
        if (is_pressed)
          pin[pin_idx++] = key;
        else
          dummy_pin[dummy_pin_idx++] = key;
        dummy_pin_idx = 0;
        mask <<= 1;
    }
    key_state = new_key_state;
    return (4 - pin_idx);
}
void read_pin(){
  signed int pin_len = PIN_LEN;
  while(pin_len>0)
    pin_len = read_keypad();
}
```

Code based in Sancus and Texas Reference Implementation of a Keypad [1,2]

[1] https://github.com/sancus-tee/vulcan/blob/master/demo/ecu-tcs/sm_tcs_kypd.c
[2] Implementing An Ultra-Low-Power Keypad Interface With MSP430™ MCUs

BUSted Attack

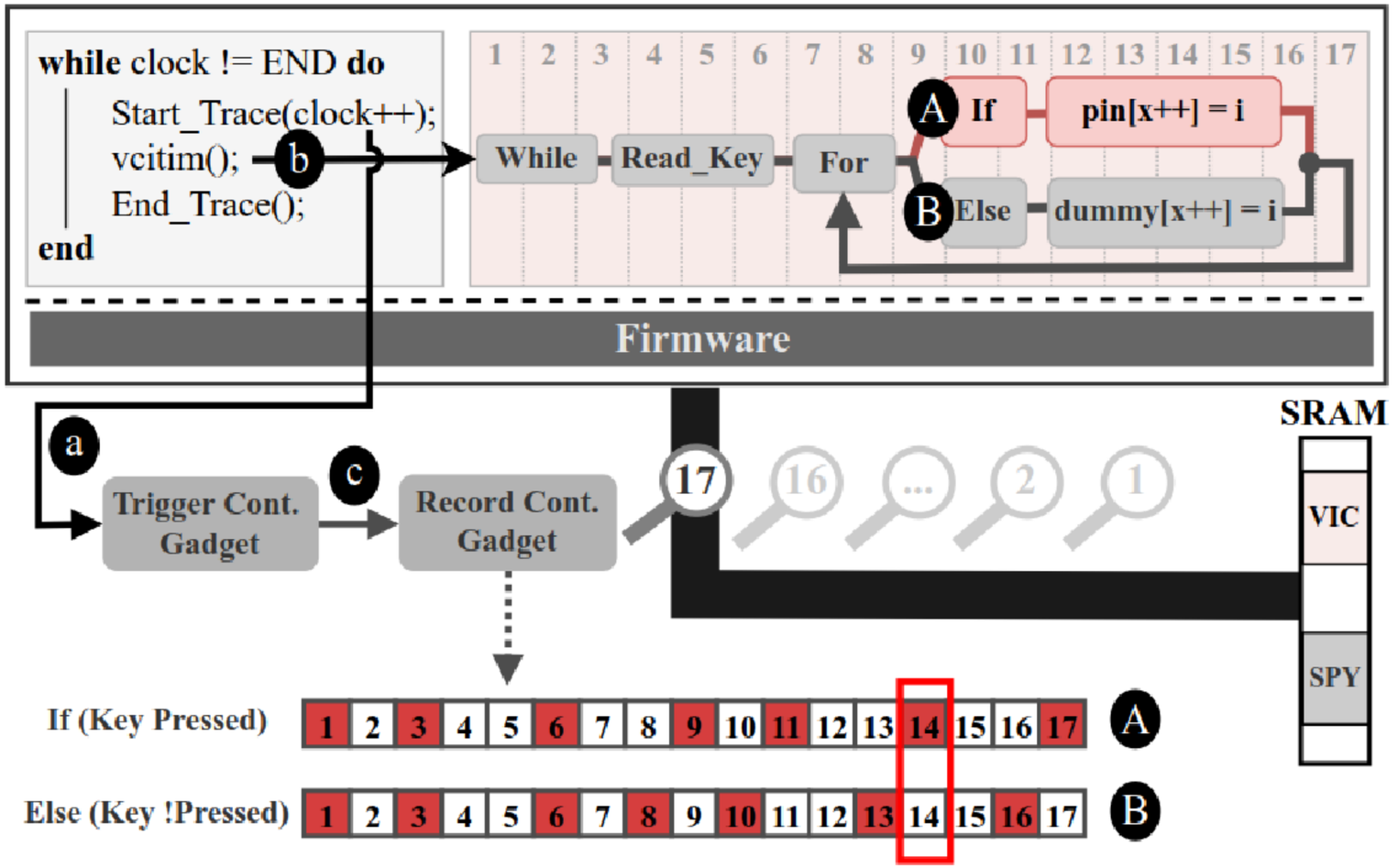



```
signed int read_keypad(void){
    int is_pressed, mask = 0x1;
    int new_key_state = get_keypad_state();                    read key
    for (int key = 0; key < KYPD_NB_KEYS; key++){
        is_pressed = (new_key_state & mask) & ~(key_state & mask);
        if (is_pressed)
          pin[pin_idx++] = key;                                if branch (leak)
        else
          dummy_pin[dummy_pin_idx++] = key;                    else branch
        dummy_pin_idx = 0;
        mask <<= 1;
    }                                                          for loop
    key_state = new_key_state;
    return (4 - pin_idx);
}
void read_pin(){
  signed int pin_len = PIN_LEN;
  while(pin_len>0)
    pin_len = read_keypad();                                   while loop
}
```

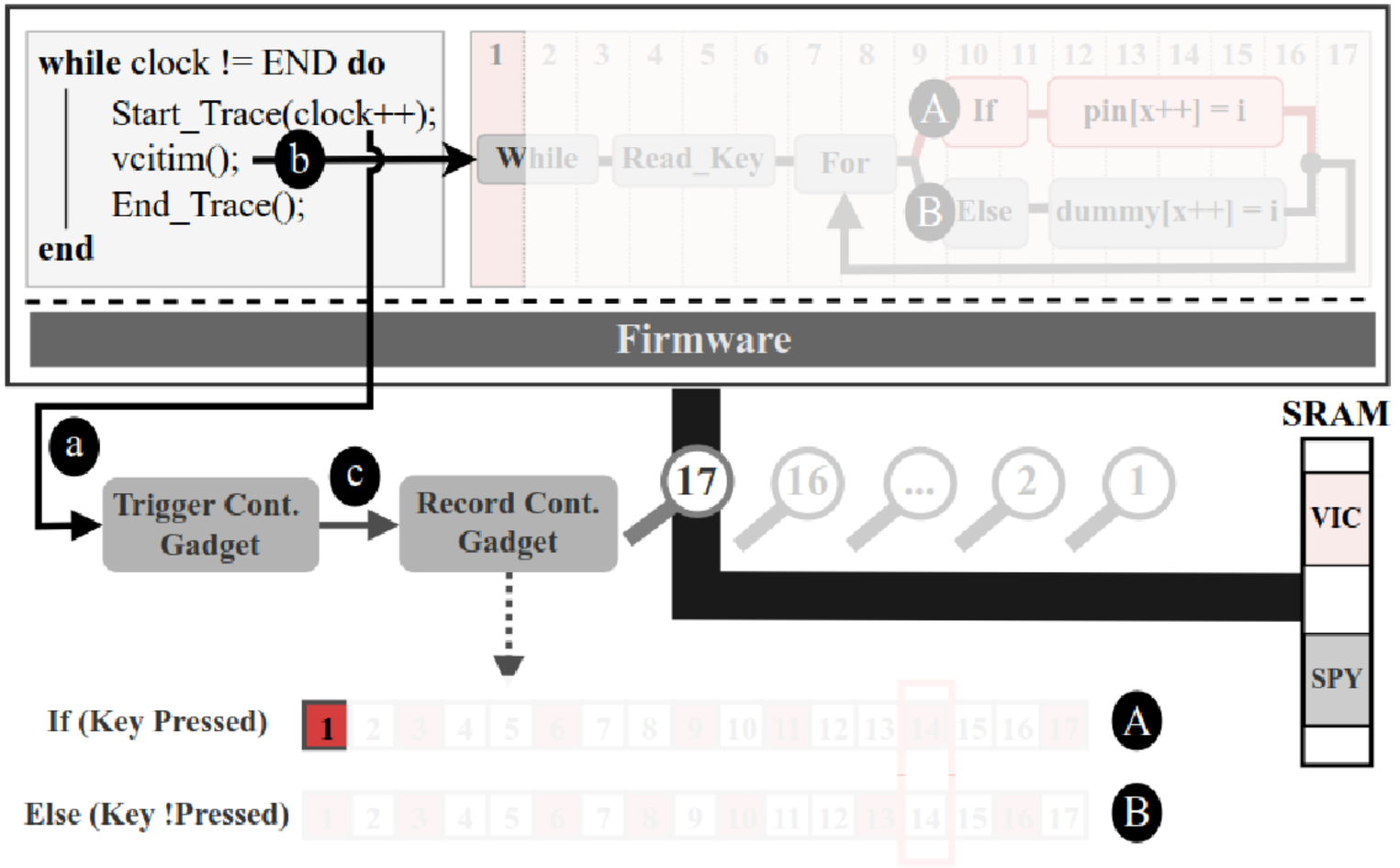
Code based in Sancus and Texas Reference Implementation of a Keypad [1,2]

[1] https://github.com/sancus-tee/vulcan/blob/master/demo/ecu-tcs/sm_tcs_kypd.c
[2] Implementing An Ultra-Low-Power Keypad Interface With MSP430™ MCUs

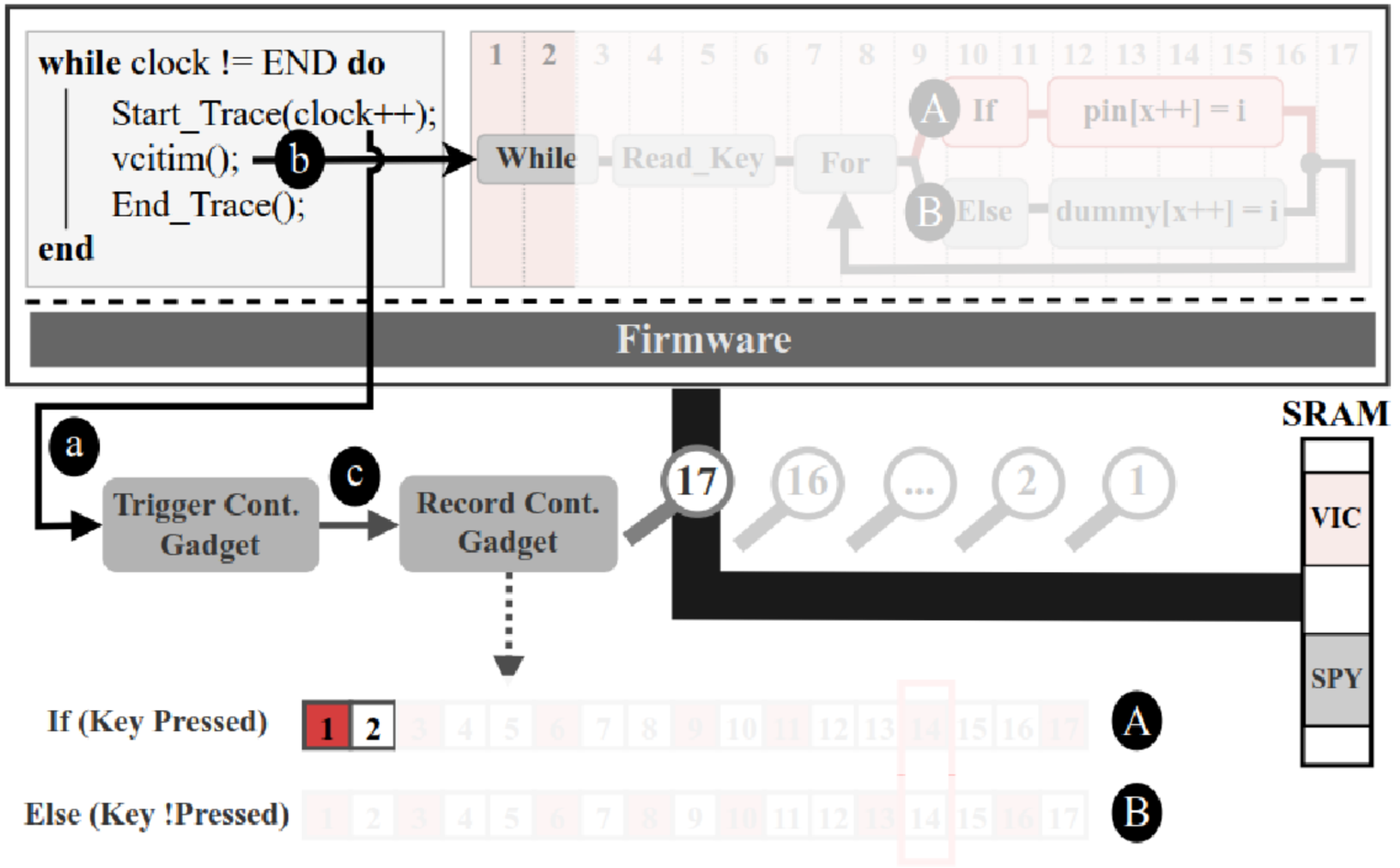
BUSted Profiling



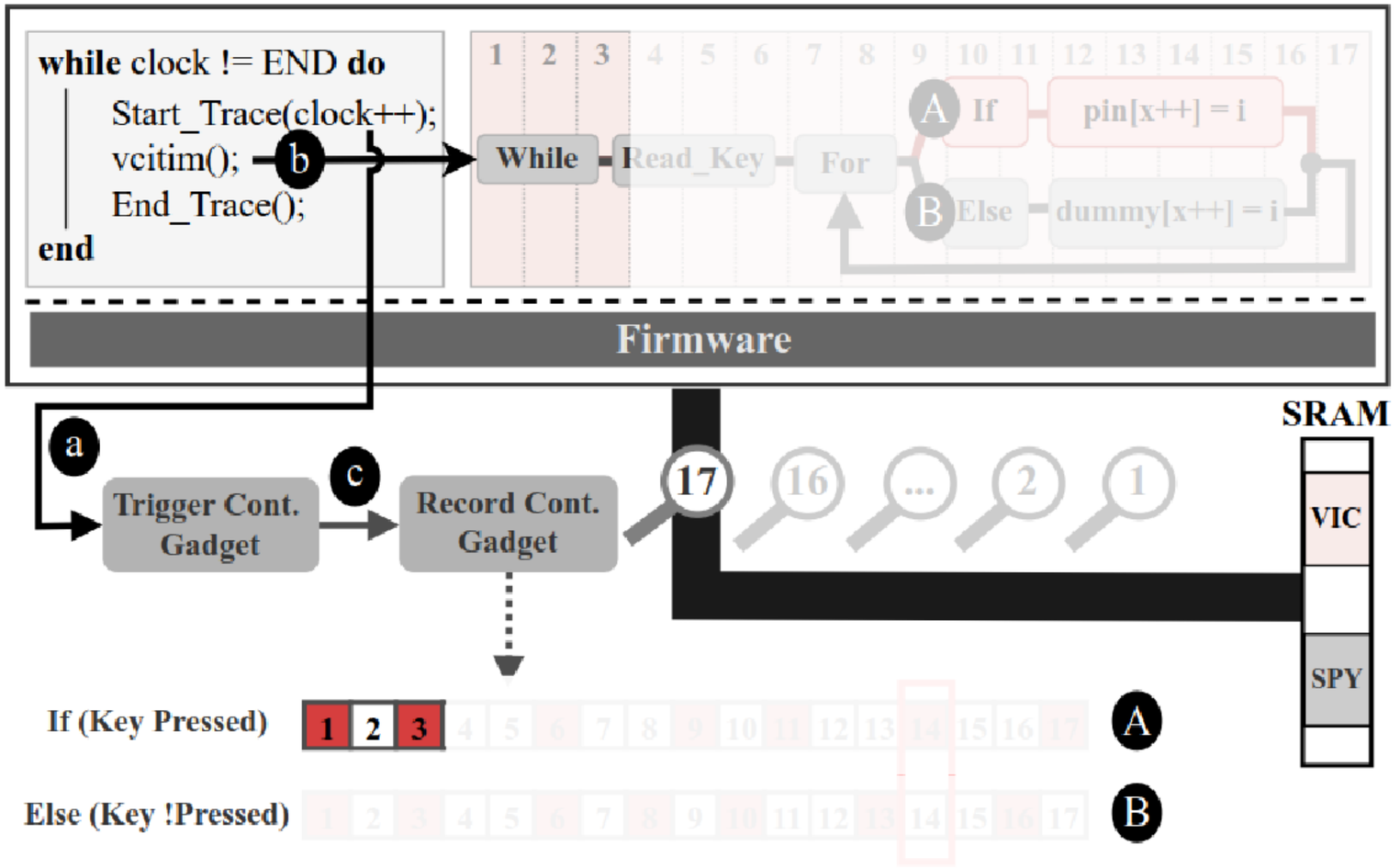
BUSted Profiling



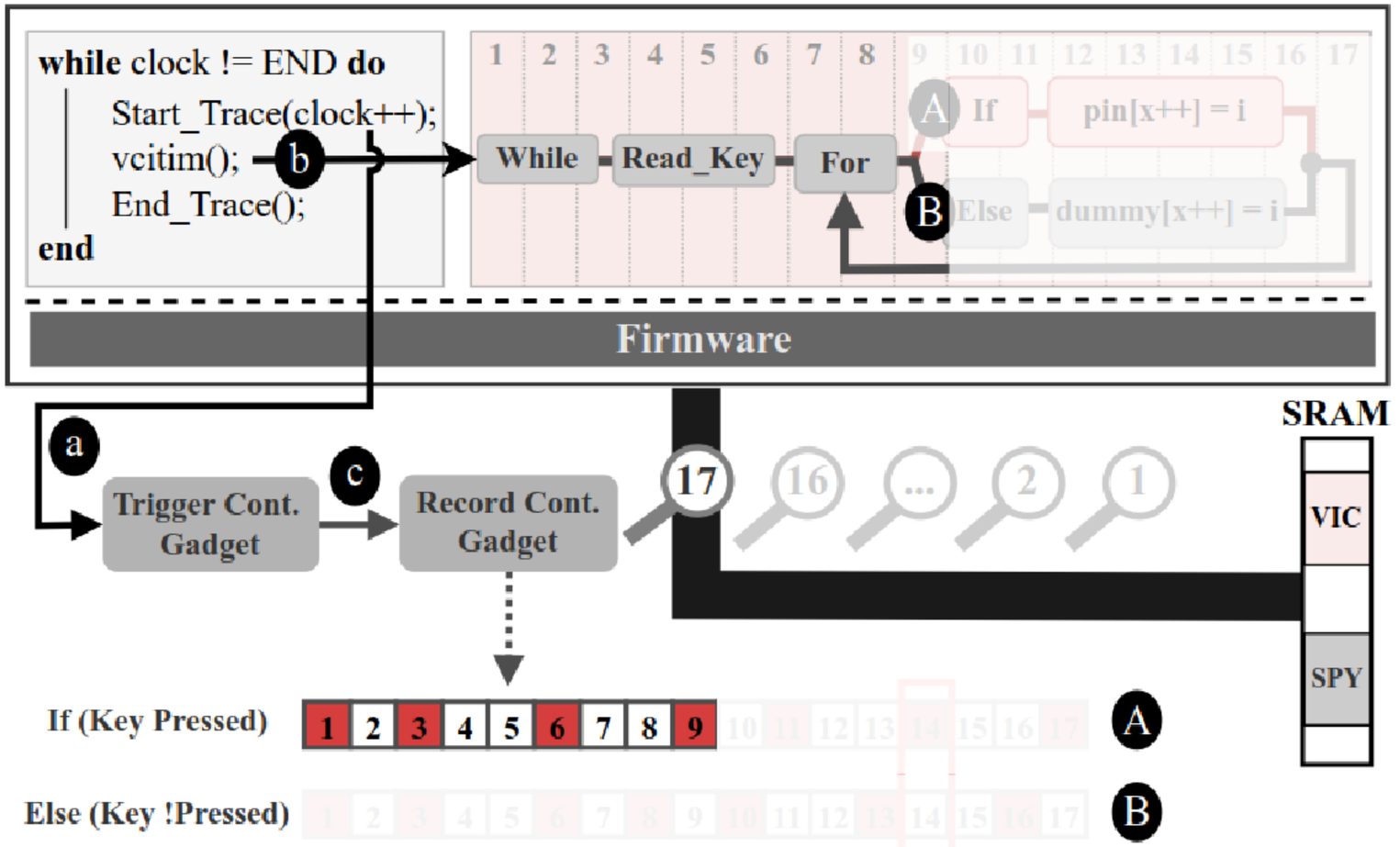
BUSted Profiling



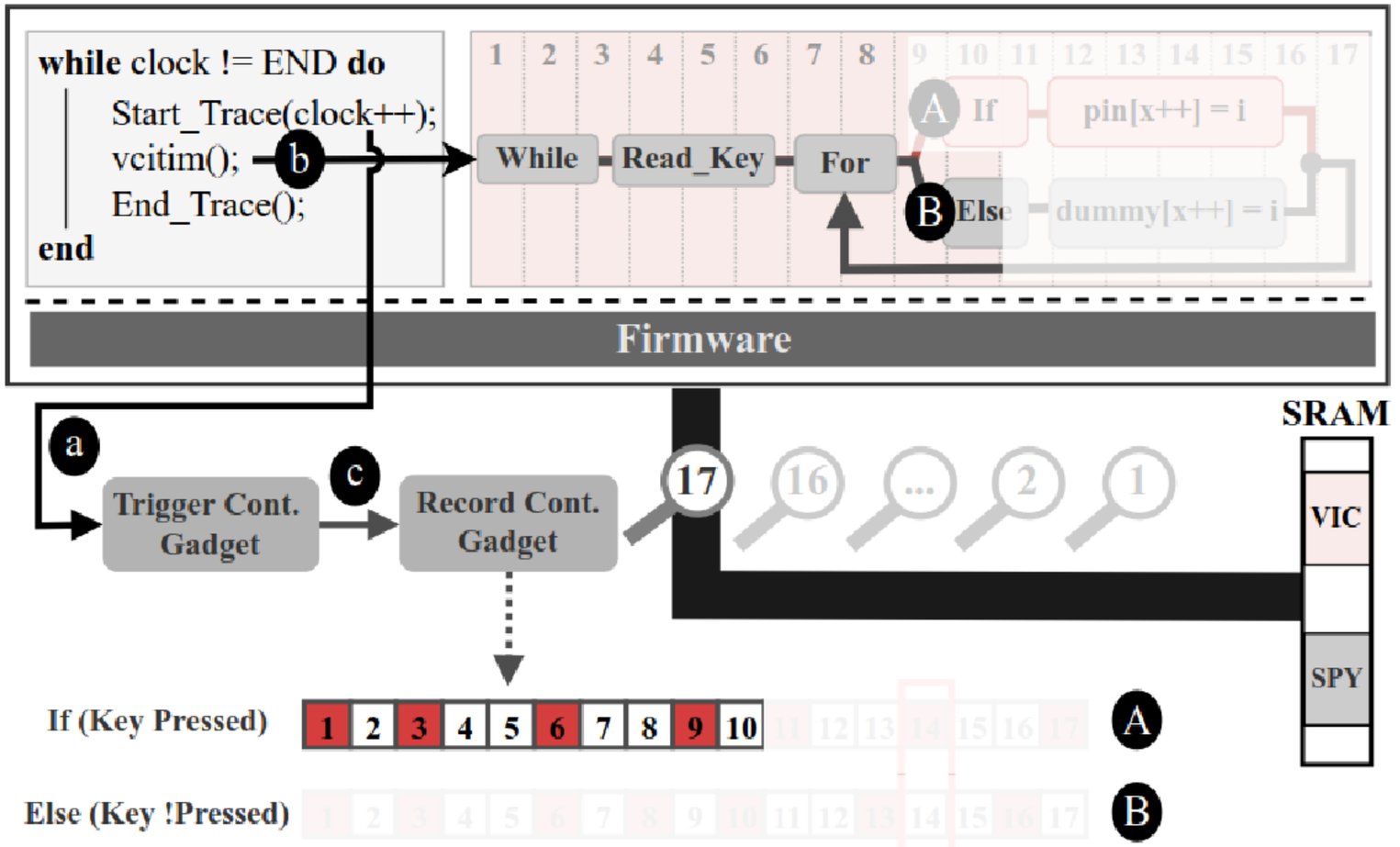
BUSted Profiling



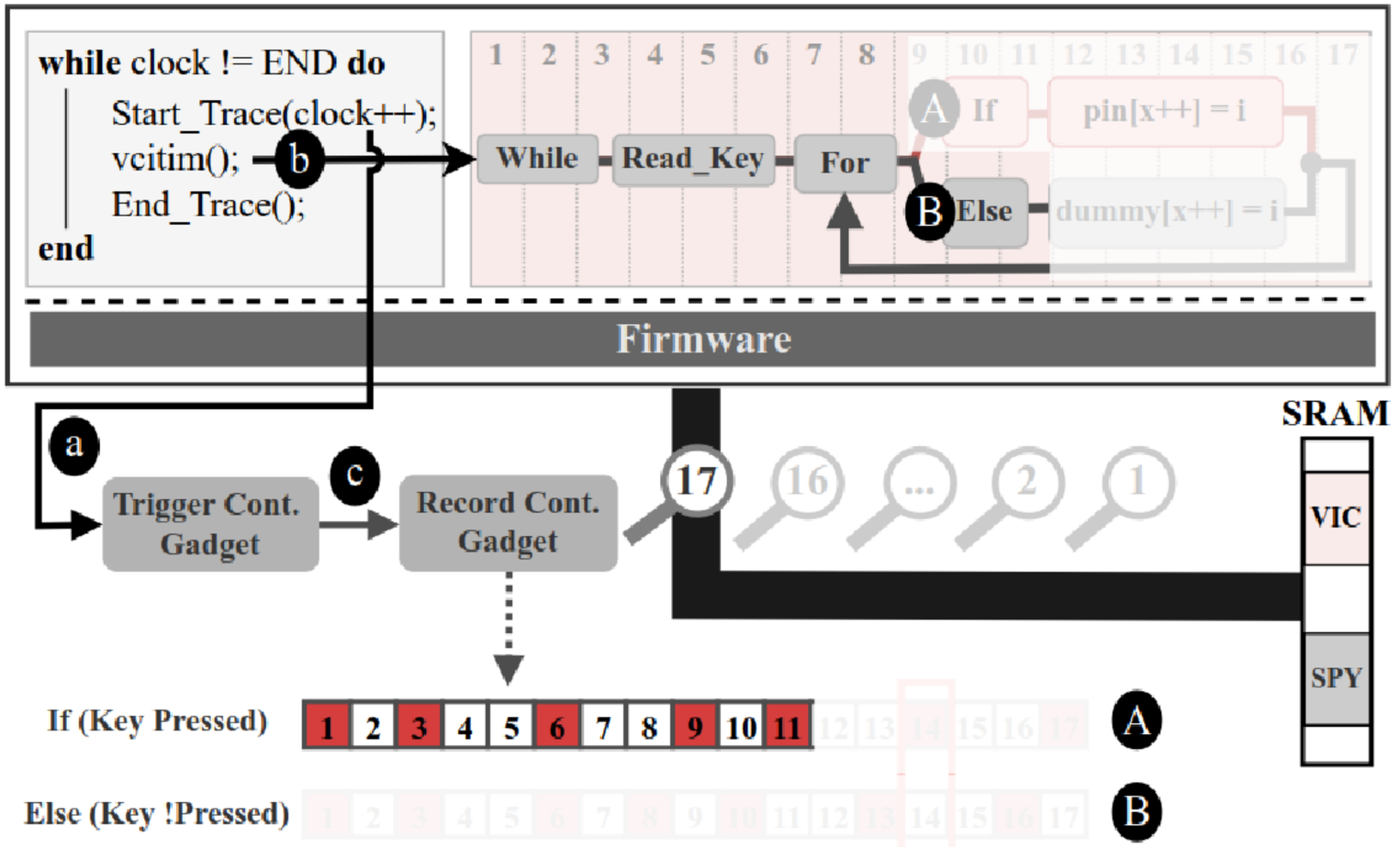
BUSted Profiling



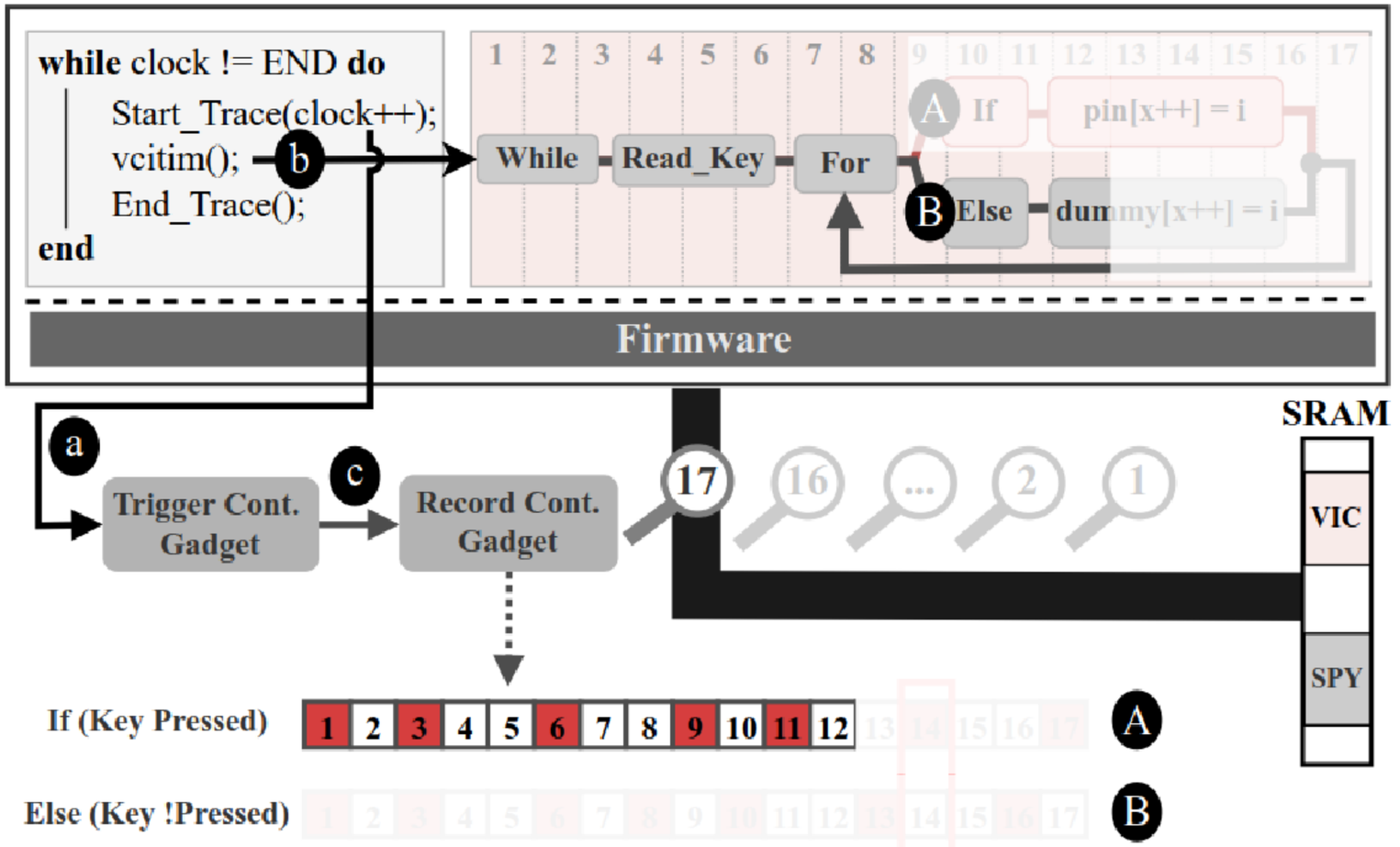
BUSted Profiling



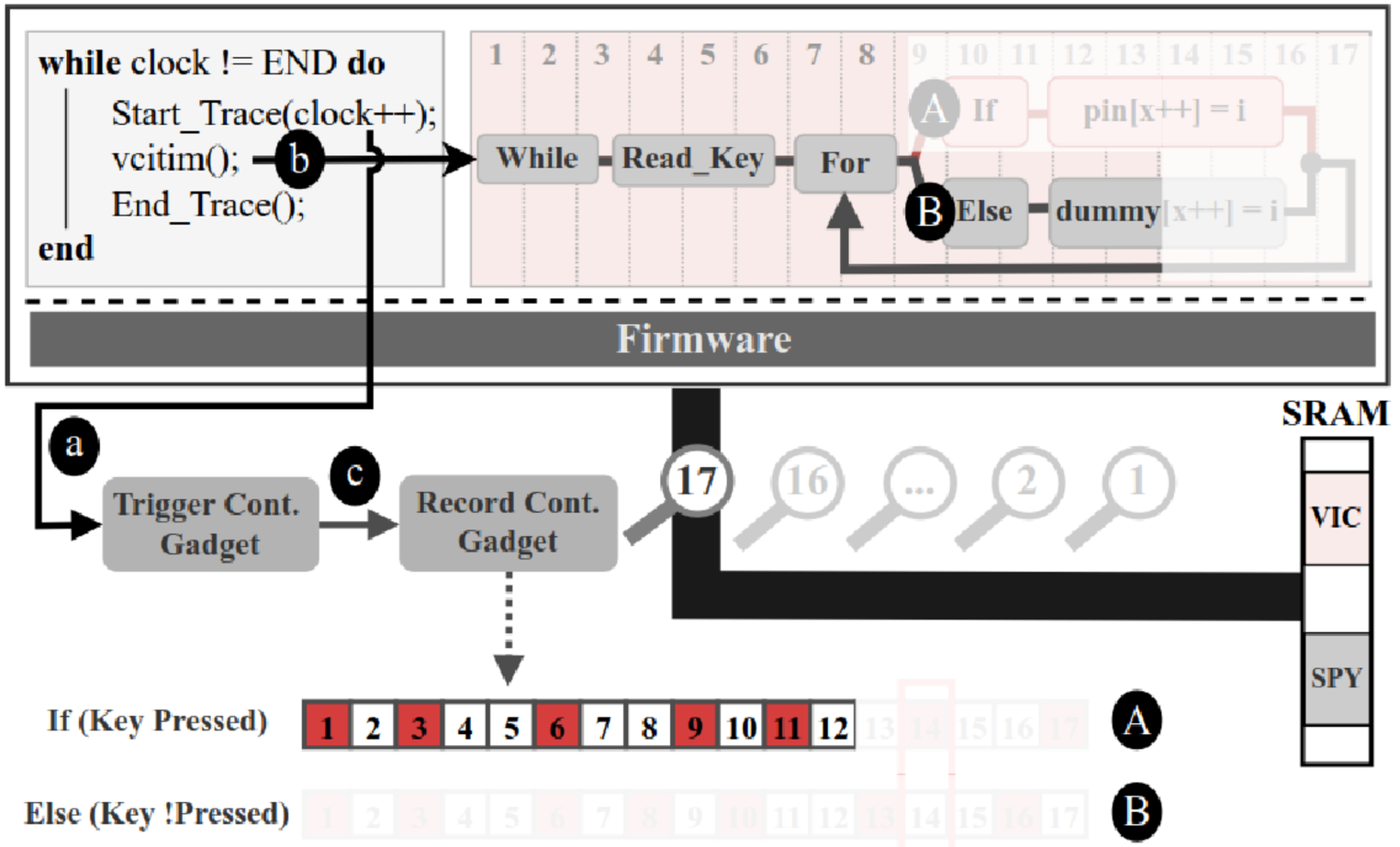
BUSted Profiling



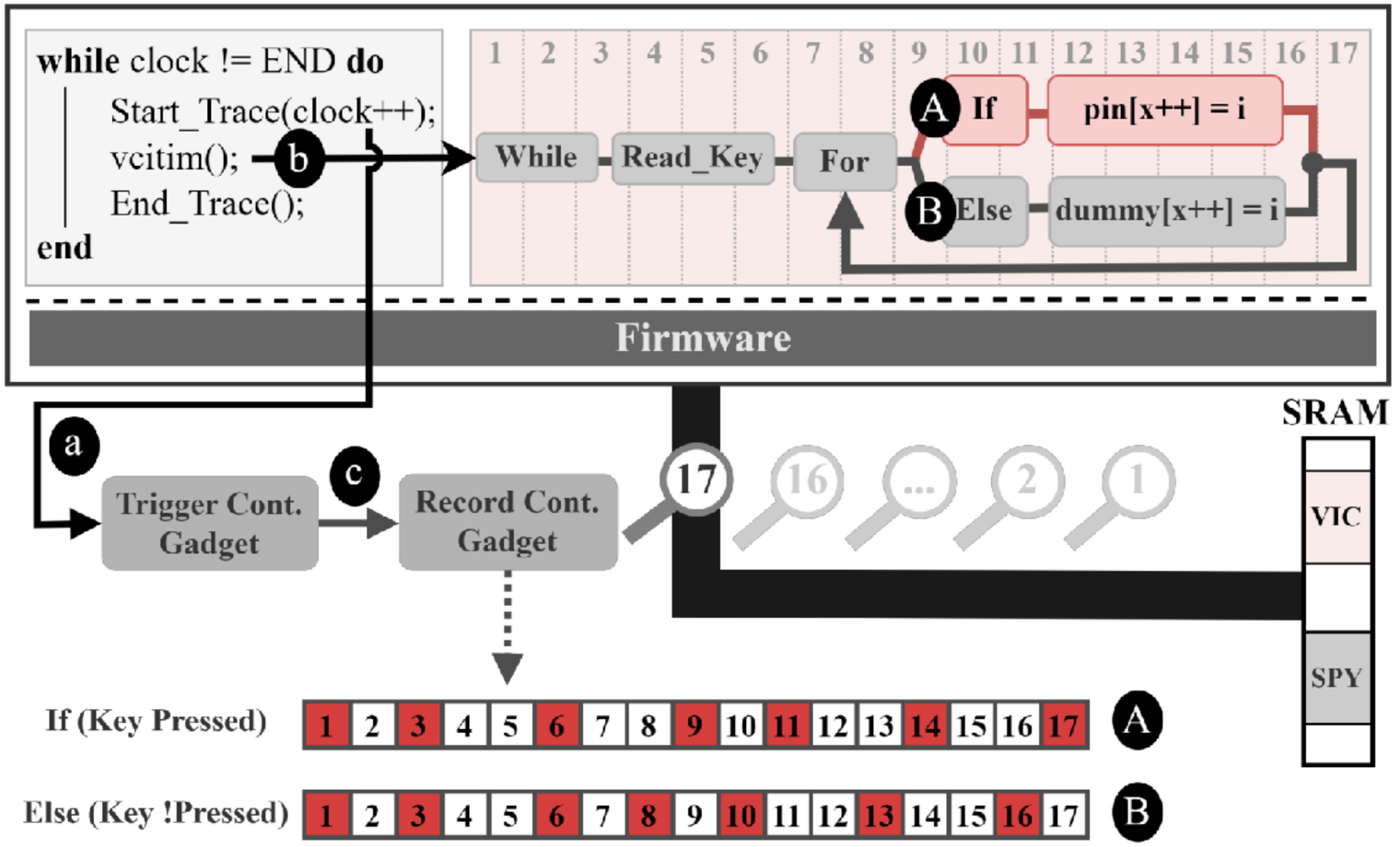
BUSted Profiling



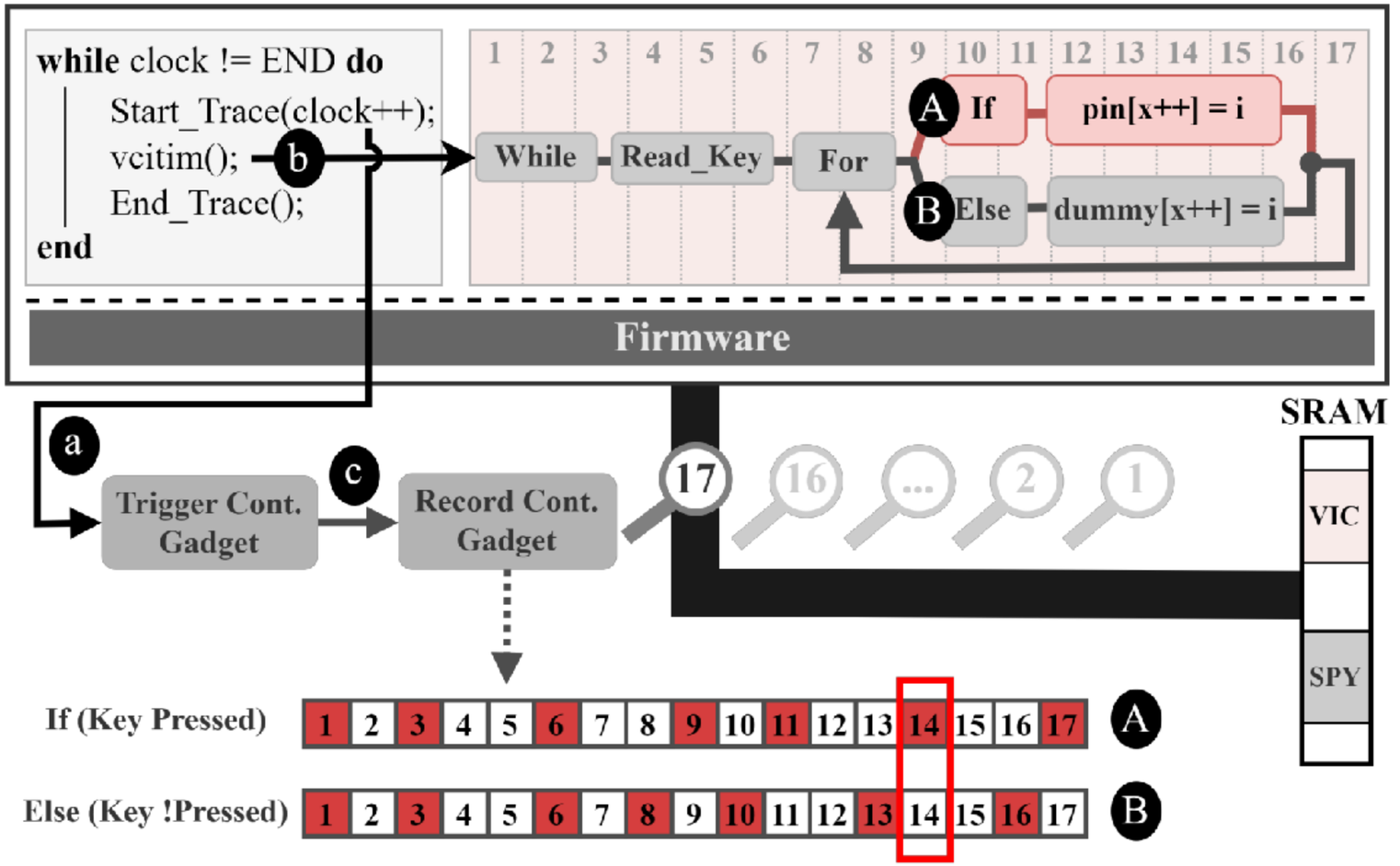
BUSted Profiling



BUSted Profiling

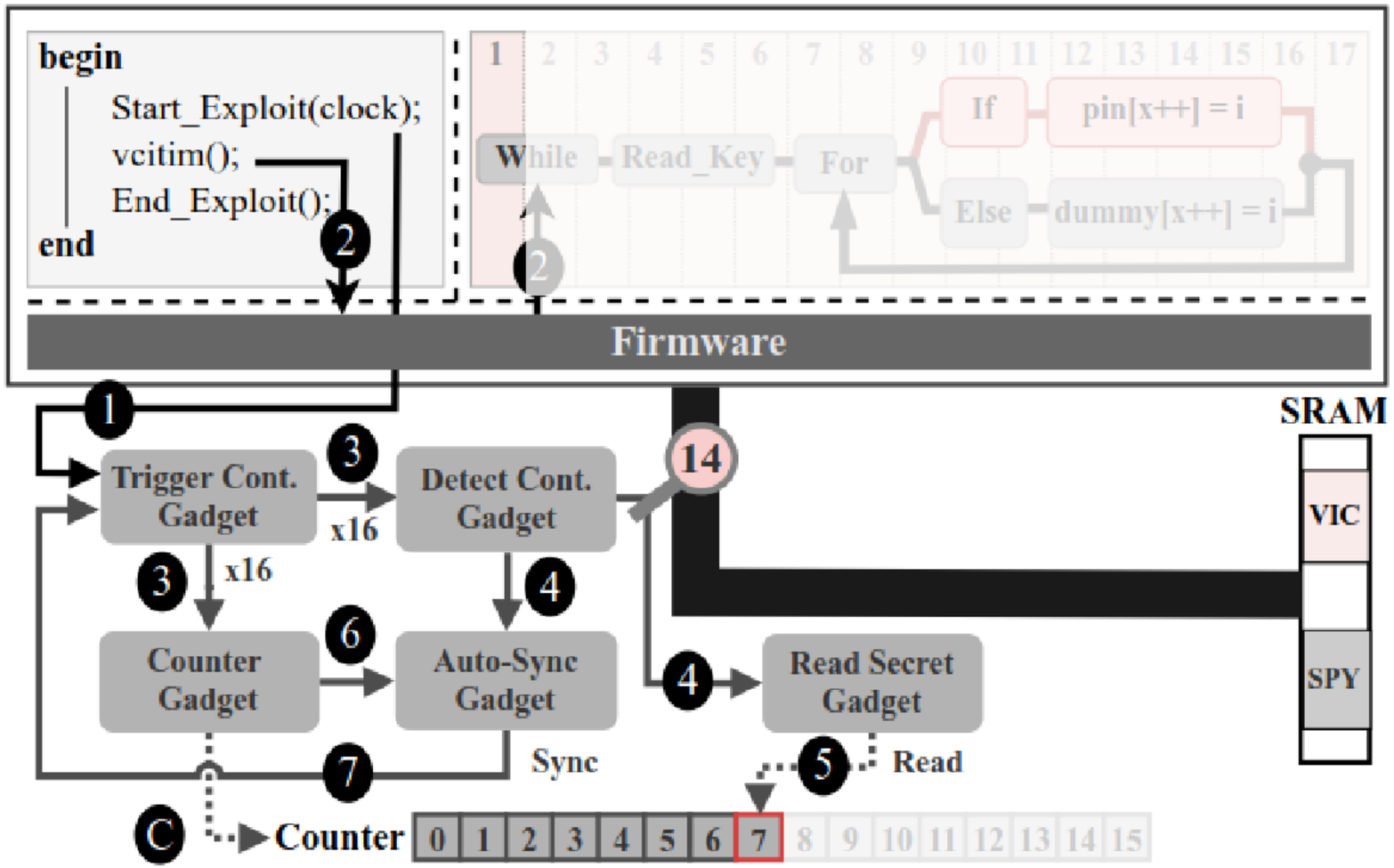


BUSted Profiling

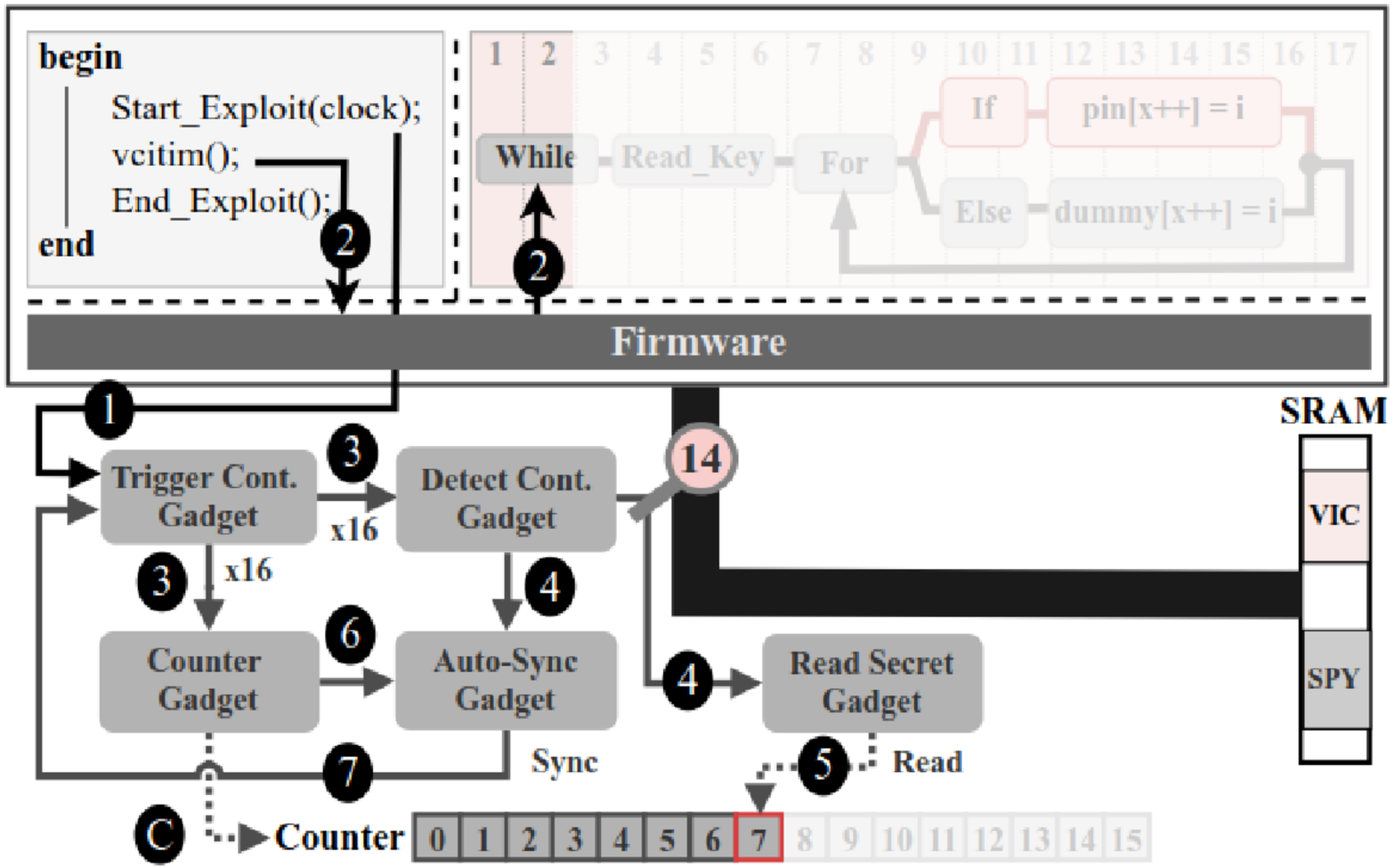


Monitor Clock Cycle 14

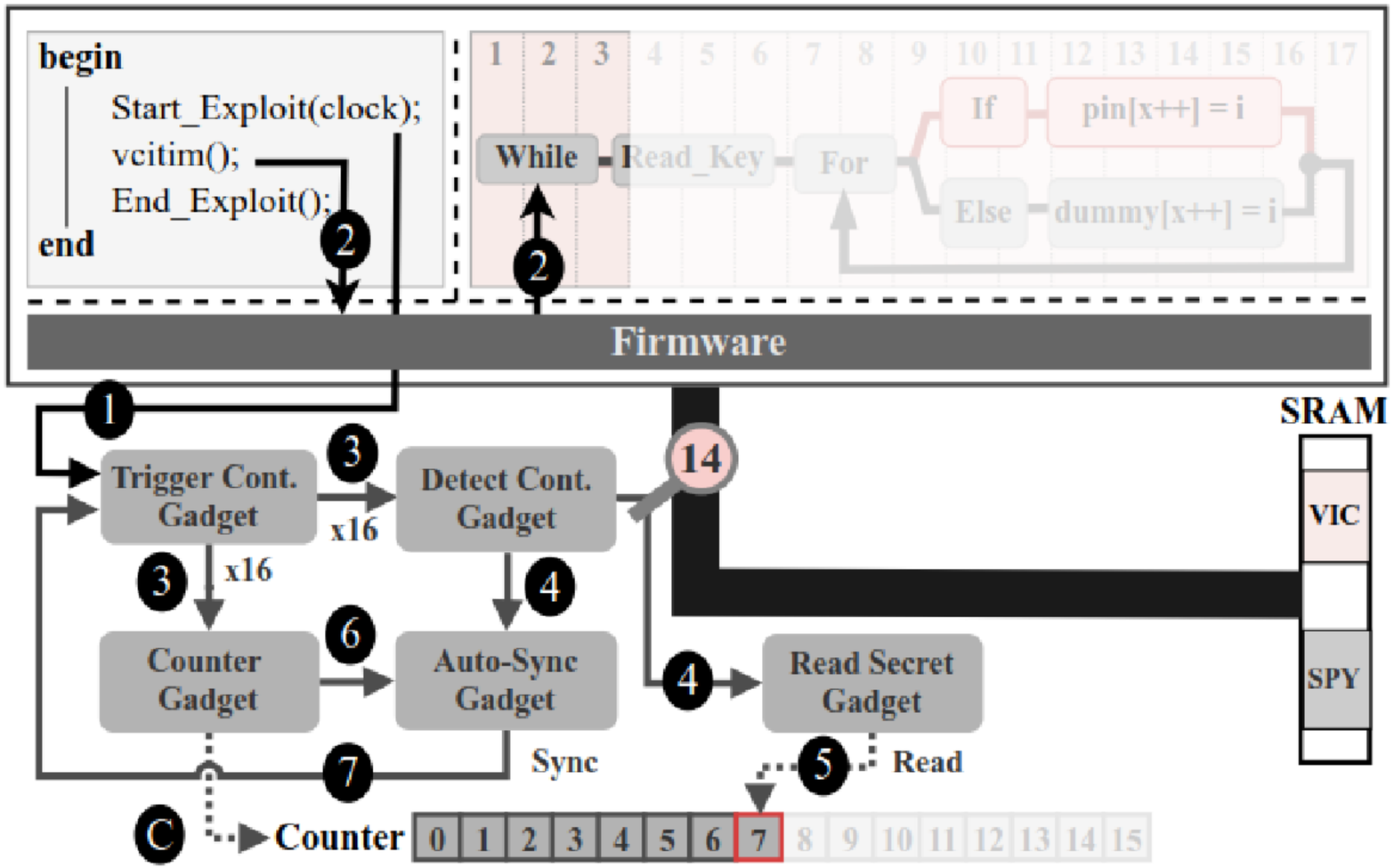
BUSted Exploitation



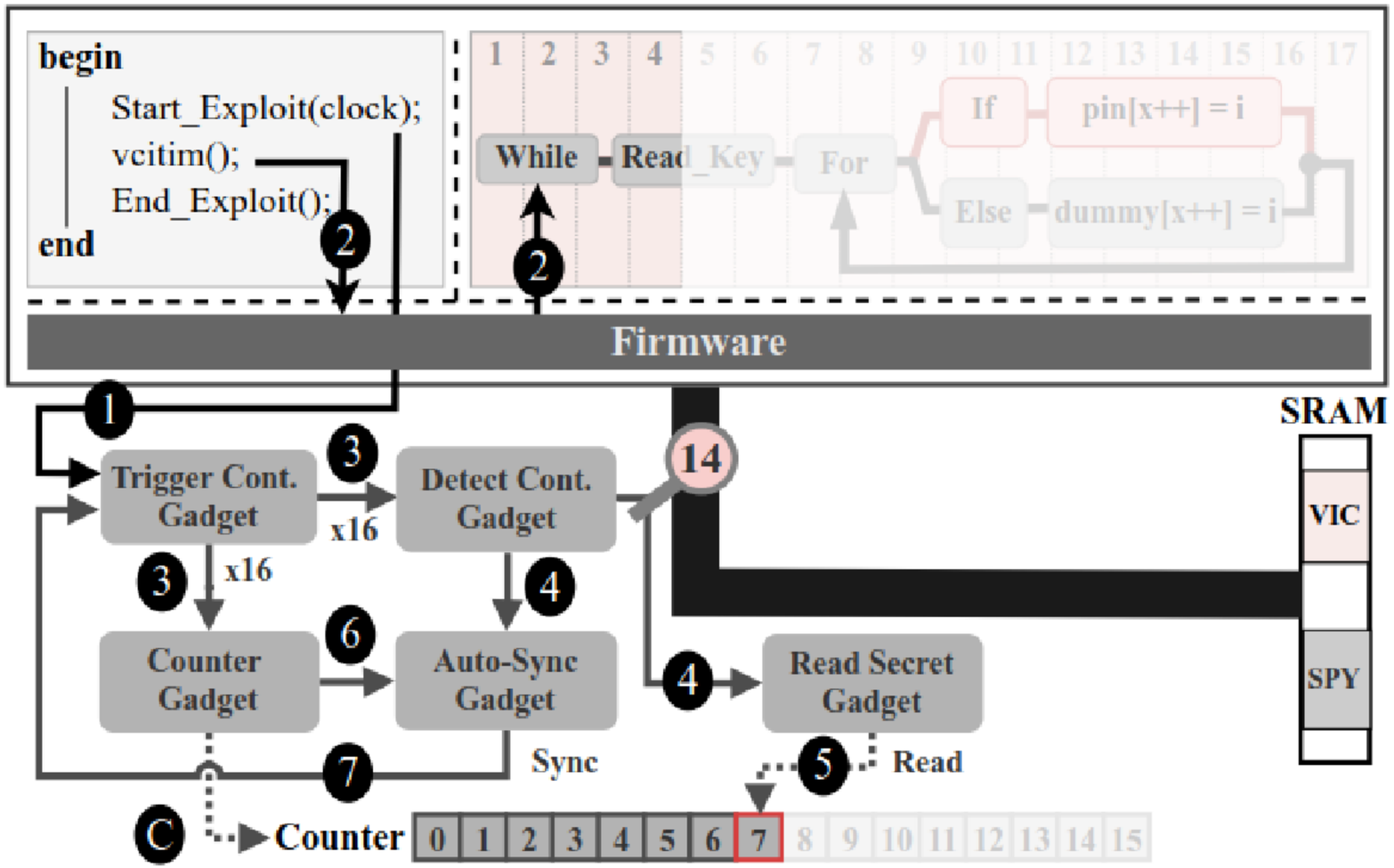
BUSted Exploitation



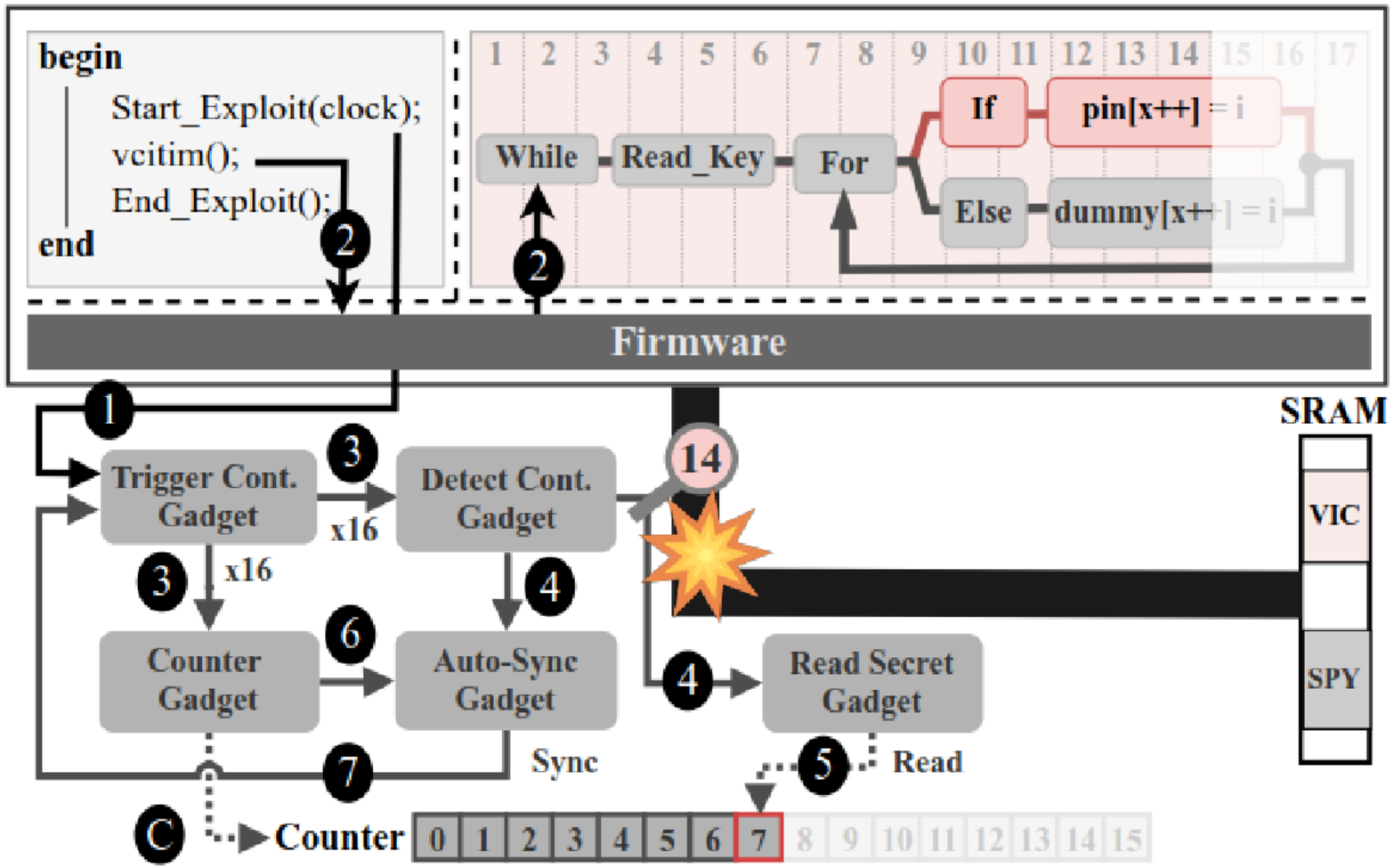
BUSted Exploitation



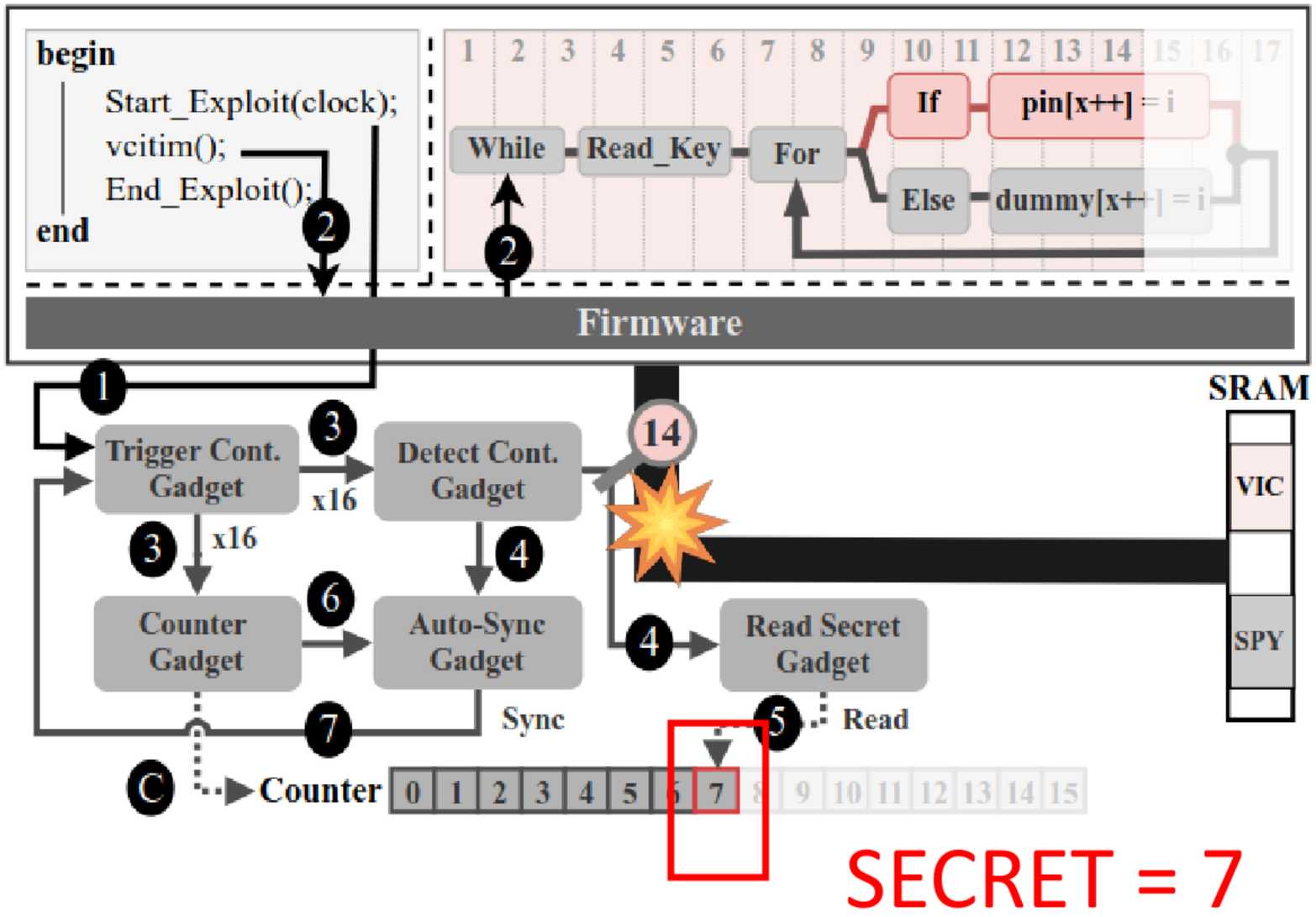
BUSted Exploitation



BUSted Exploitation



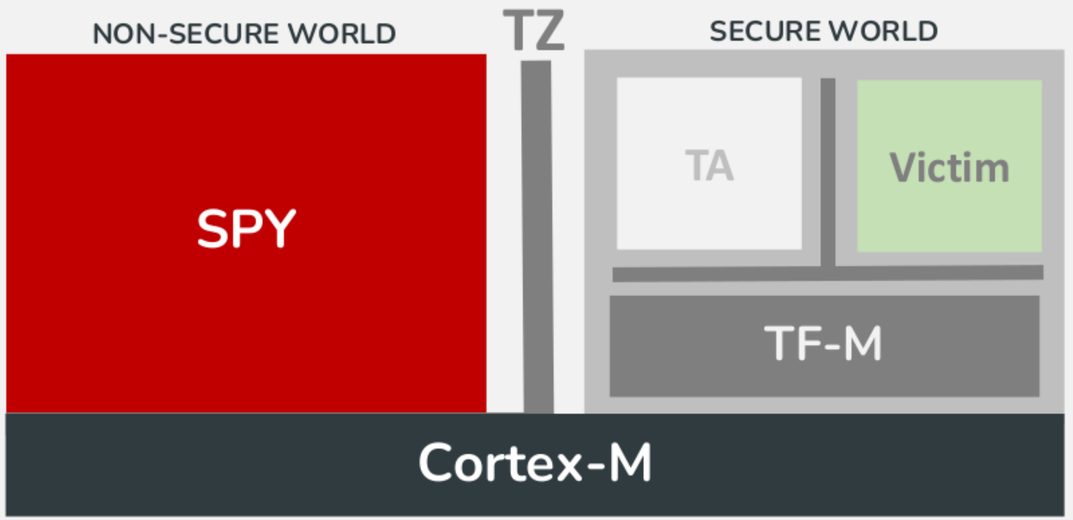
BUSted Exploitation



“Live” Demo

Demo of BUSted Attack

Under the hood



“Live” Demo

```
[INF] Starting bootloader
[INF] Swap type: none
[INF] Swap type: none
[INF] Bootloader chainload address offset: 0x13000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!
TF-M isolation level is: 0x00000002
Booting TFM v1.4.0
Non-Secure system starting

Jumping to S World
***********************************
Entering Read Keypad Secure Service
4-digit Secure PIN = 5 8 6 3
Leaving Read Keypad Secure Service
***********************************
Stolen PIN  = 5 8 6 3
Non-Secure end opertions
```

Summary

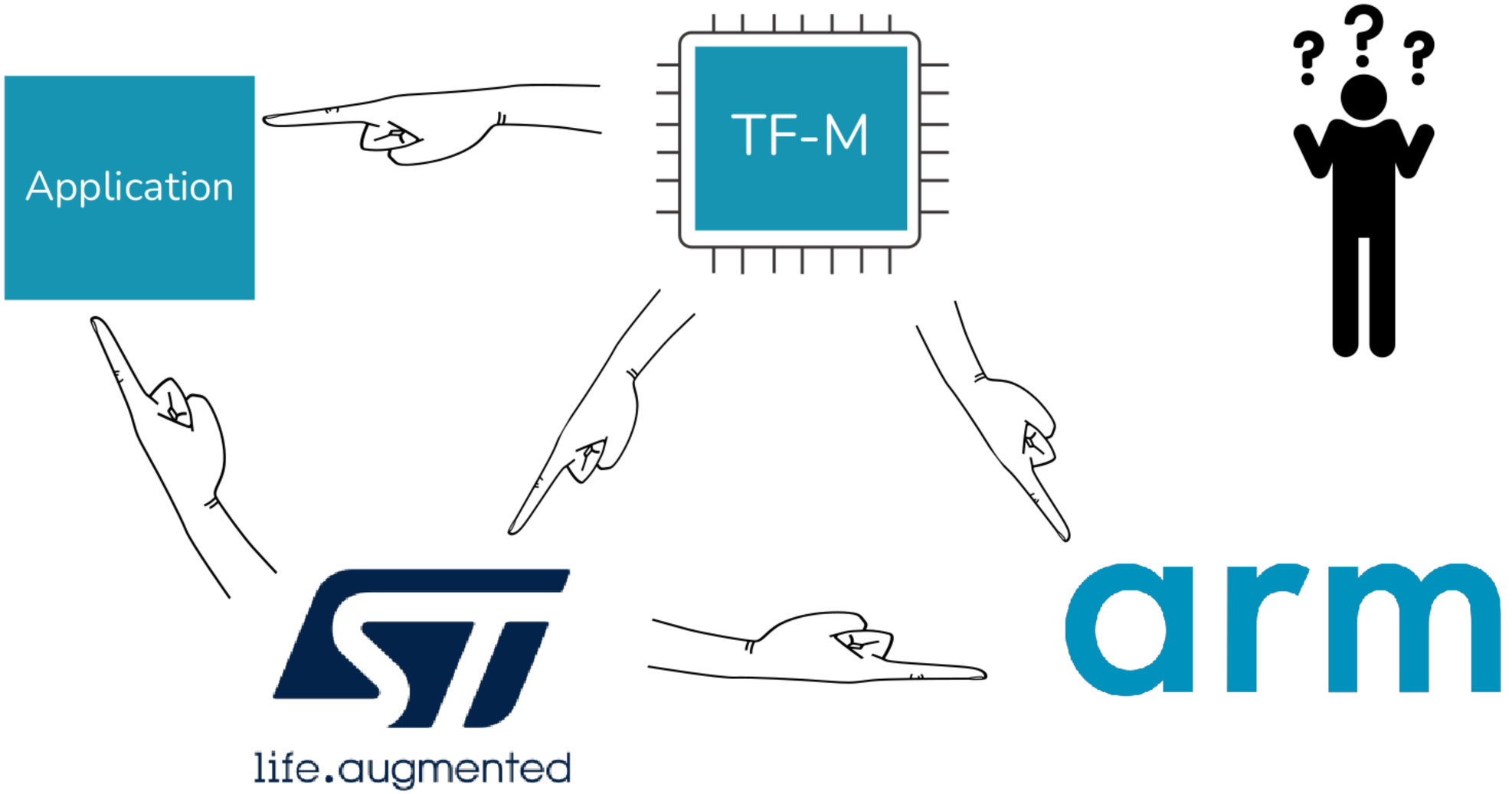
Responsible Disclosure, and Black Hat Sound Bytes

We have introduced the concept of **hardware gadgets** and presented **BUSted**, the **1st microarchitectural side-channel attack** exploiting **TrustZone-M** devices.

Responsible Disclosure

Responsible Disclosure

Responsible Disclosure



BH Sound Bytes

1. We **debunked** the common belief **that MCUs are not vulnerable** to **microarchitectural side-channel attacks**.
2. We presented **a new class** of software-based **microarchitectural side-channel attacks** affecting **MCUs**
3. We provided a **reference attack** that **bypasses** the TEE (**TrustZone-M**) of modern Armv8-M MCUs (Cortex-M33), **breaking all memory isolations!!**

THANK YOU!

Cristiano Rodrigues | Sandro Pinto, PhD
(Centro ALGORITMI / LASI, Universidade do Minho)

id9492@alunos.uminho.pt

LinkedIn - https://www.linkedin.com/in/cristiano-rodrigues-msc-engineer/
ResearchGate - https://www.researchgate.net/profile/Cristiano-Rodrigues-10
Github - https://github.com/ESCristiano

sandro.pinto@dei.uminho.pt

LinkedIn - https://www.linkedin.com/in/sandro-pinto-phd-40535455/
ResearchGate - https://www.researchgate.net/profile/Sandro_Pinto2
Github - https://github.com/sandro2pinto/

Q&A