µRAI: Securing Embedded Systems with Return Address Integrity

Naif Saleh Almakhdhub[1,4,5,6]

Abraham A. Clements[3,4,5]

Saurabh Bagchi[1,4]

Mathias Payer[2,5]

1

2

3

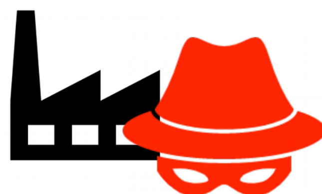
4

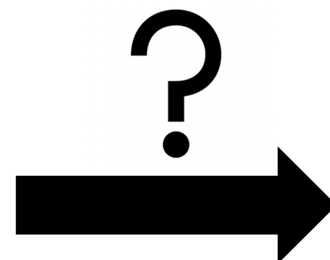
5

6

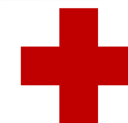
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND-XXXX

Current State of Security





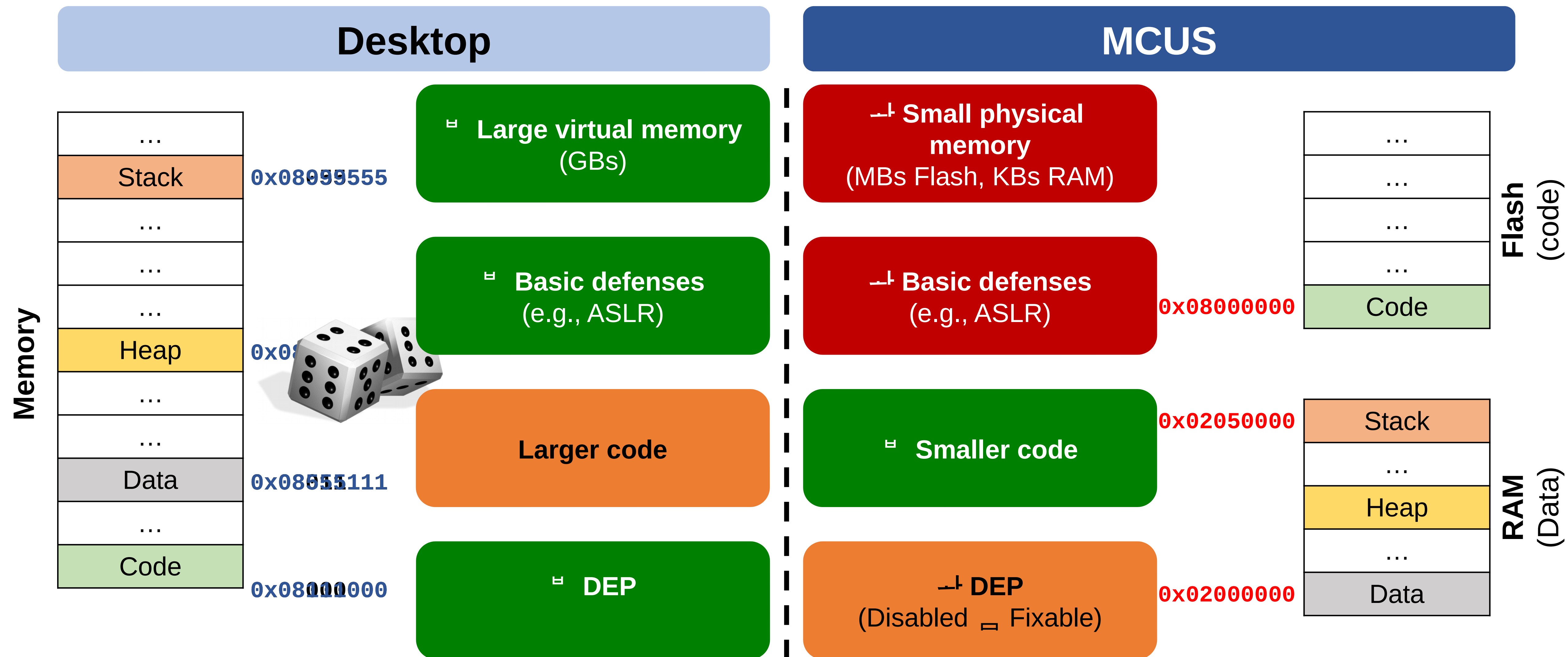
Target:
Embedded and IoT devices Running Microcontroller Systems (MCUS)



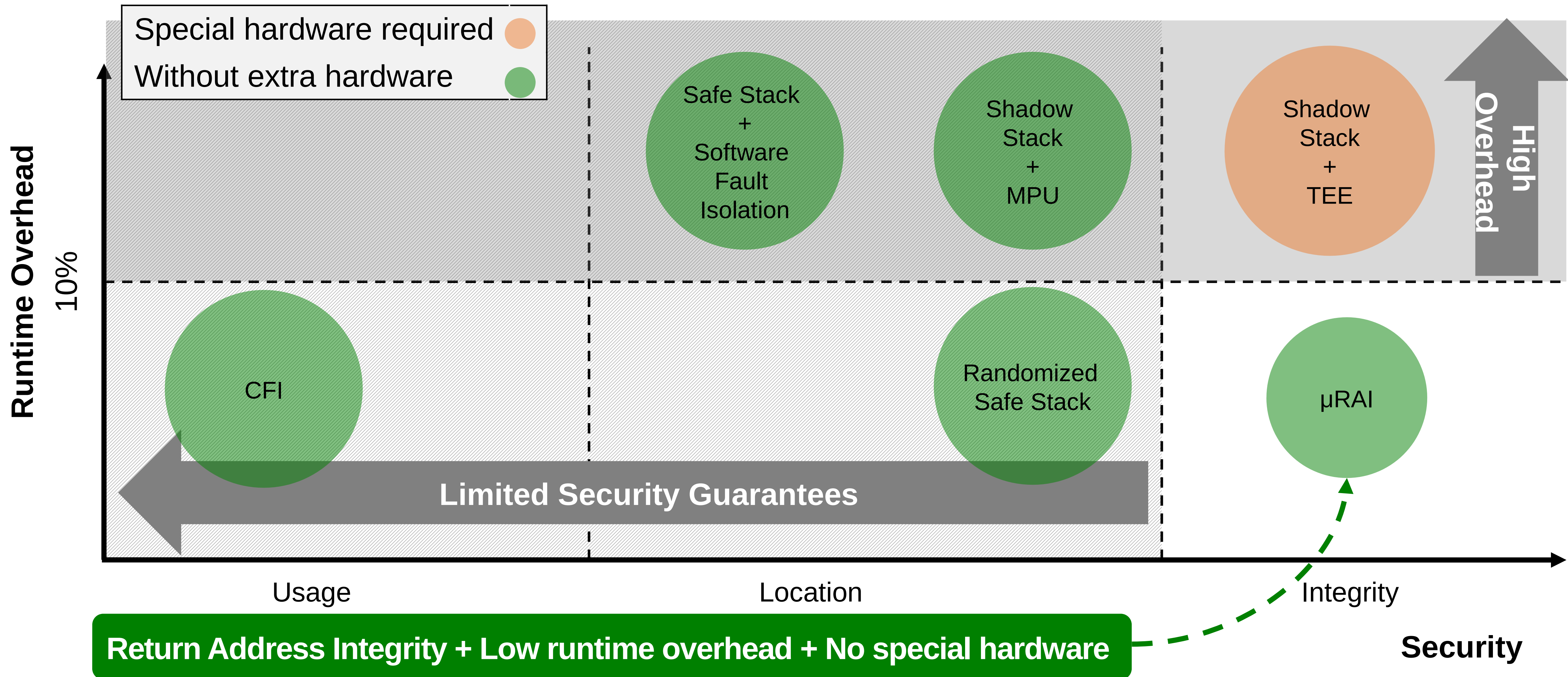
Attack:
Control-flow Hijacking

[1] https://www.wired.com/story/broadpwn-wi-fi-vulnerability-ios-android/
[2] https://keenlab.tencent.com/en/2020/01/02/exploiting-wifi-stack-on-tesla-model-s/
[3] https://www.securityweek.com/rise-ics-malware-how-industrial-security-threats-are-becoming-more-surgical

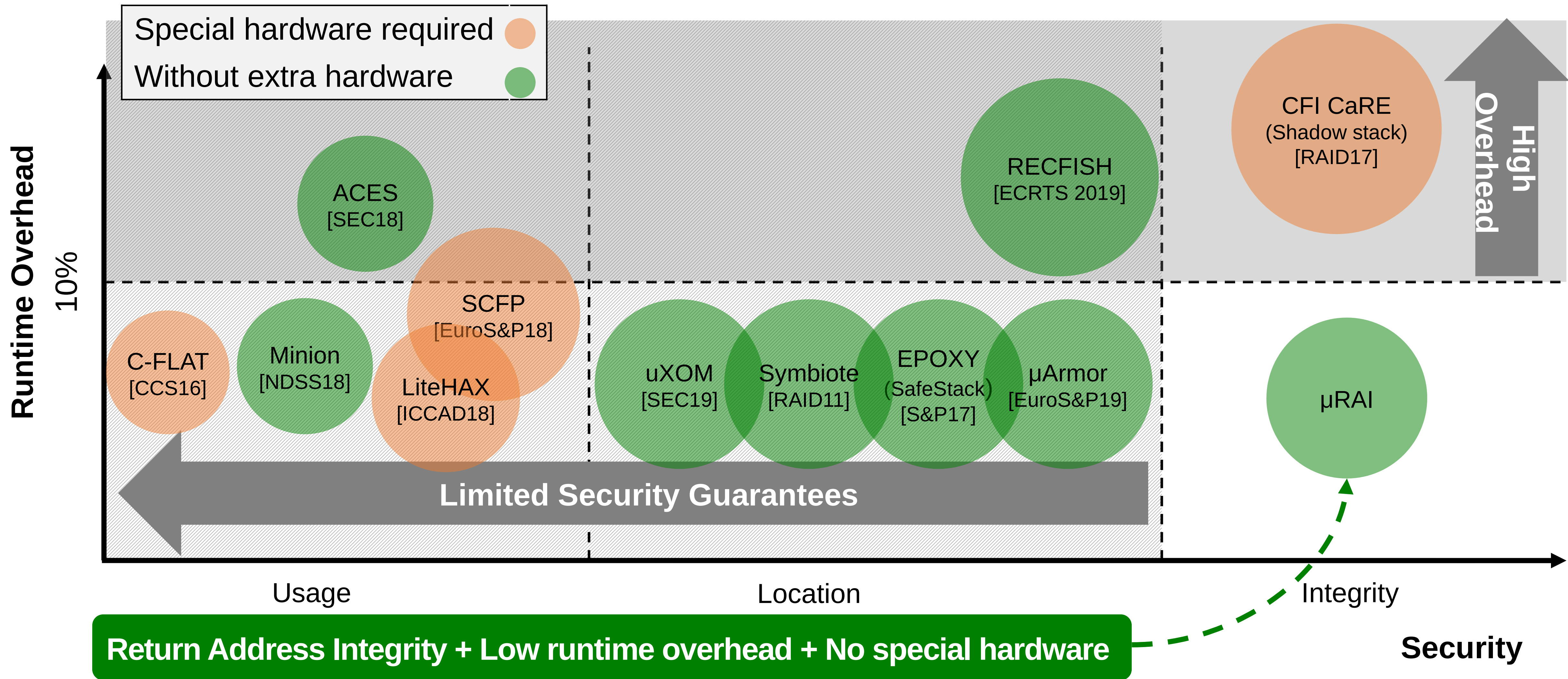
MCUS Challenges



MCUS Defenses for Return Addresses (Conceptual)



MCUS Defenses for Return Addresses (Related Work)



PURDUE UNIVERSITY EPFL Sandia National Laboratories hexhive King Saud University

Return Address Integrity (RAI)

- **Every attack requires corrupting a return addresses by *overwriting* it**

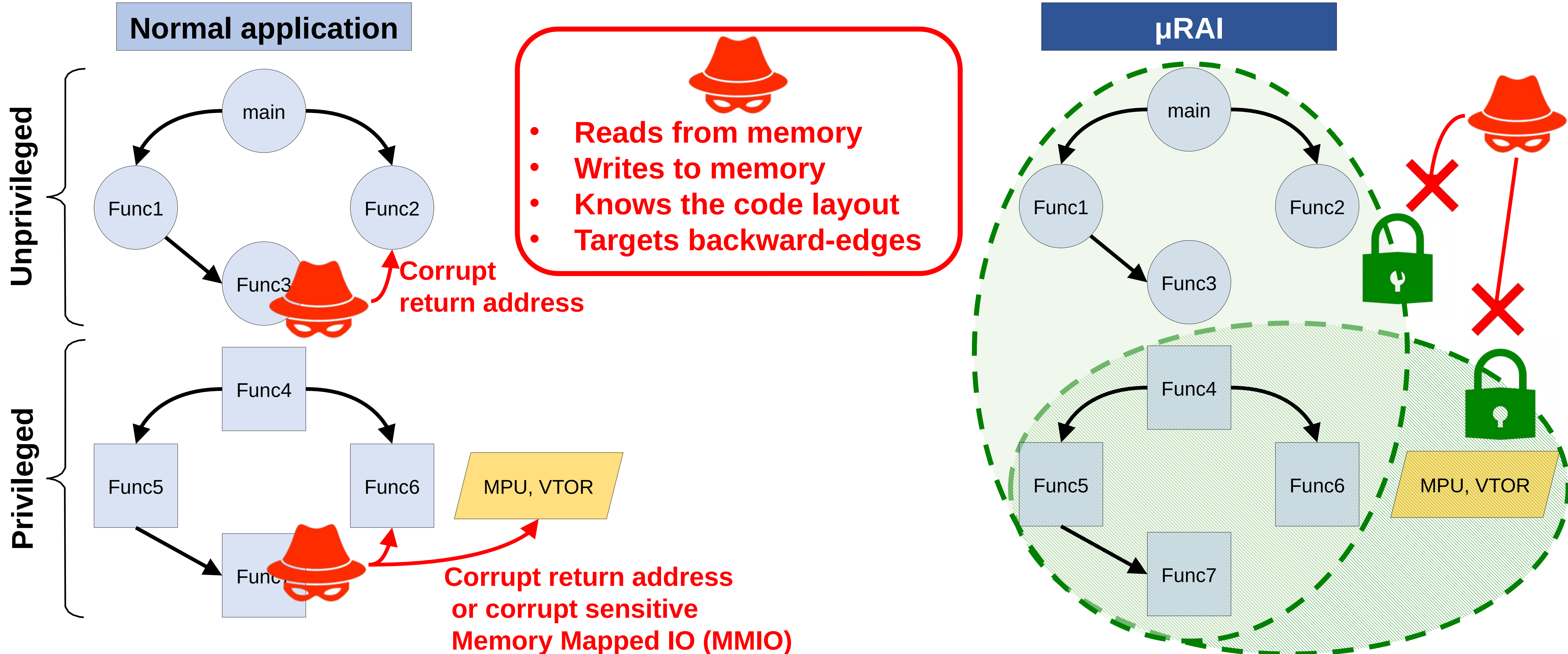
- **Main limitation of defenses ㅂ return addresses are in writable memory**
  - Example: Information hiding

- **Key solution is to *prevent an attacker from corrupting return addresses*.**

RAI Property:

- **Ensure the return address is never writable except by an authorized instruction.**
- **Return addresses are never pushed to the stack or any writable memory by an adversary.**

Threat Model & µRAI Protection

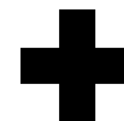


µRAI: Overview

1

Enforces the RAI property

State Register



Jump Table
Jump return_location1
Jump return_location2
…

Read + eXecute

2

Protects exception handlers and privileged execution

Exception handler software-fault isolation

3

Low runtime overhead

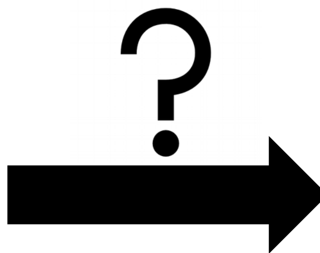
Relative jump target lookup routine

PURDUE UNIVERSITY EPFL Sandia National Laboratories hexhive جامعة الملك سعود King Saud University

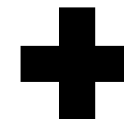
μRAI: Overview

1

Enforces the RAI property



State Register

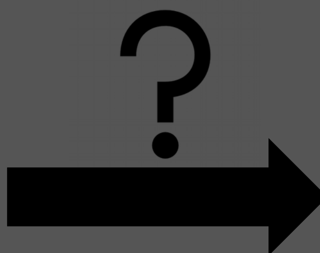


Jump Table
Jump return_location1
Jump return_location2
…

Read + eXecute

2

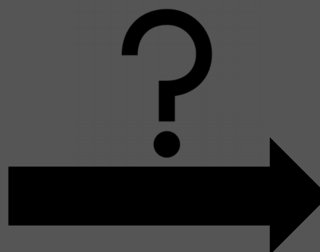
Protects exception handlers and privileged execution



Exception handler software-fault isolation

3

Low runtime overhead



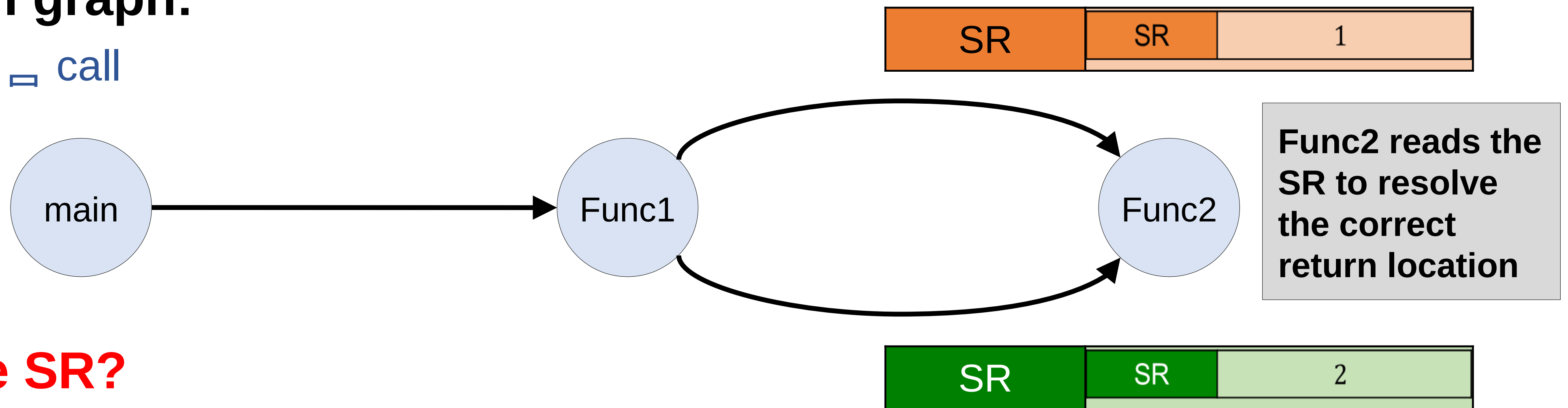
Relative jump target lookup routine

PURDUE UNIVERSITY EPFL Sandia National Laboratories hexhive King Saud University

µRAI and the State Register

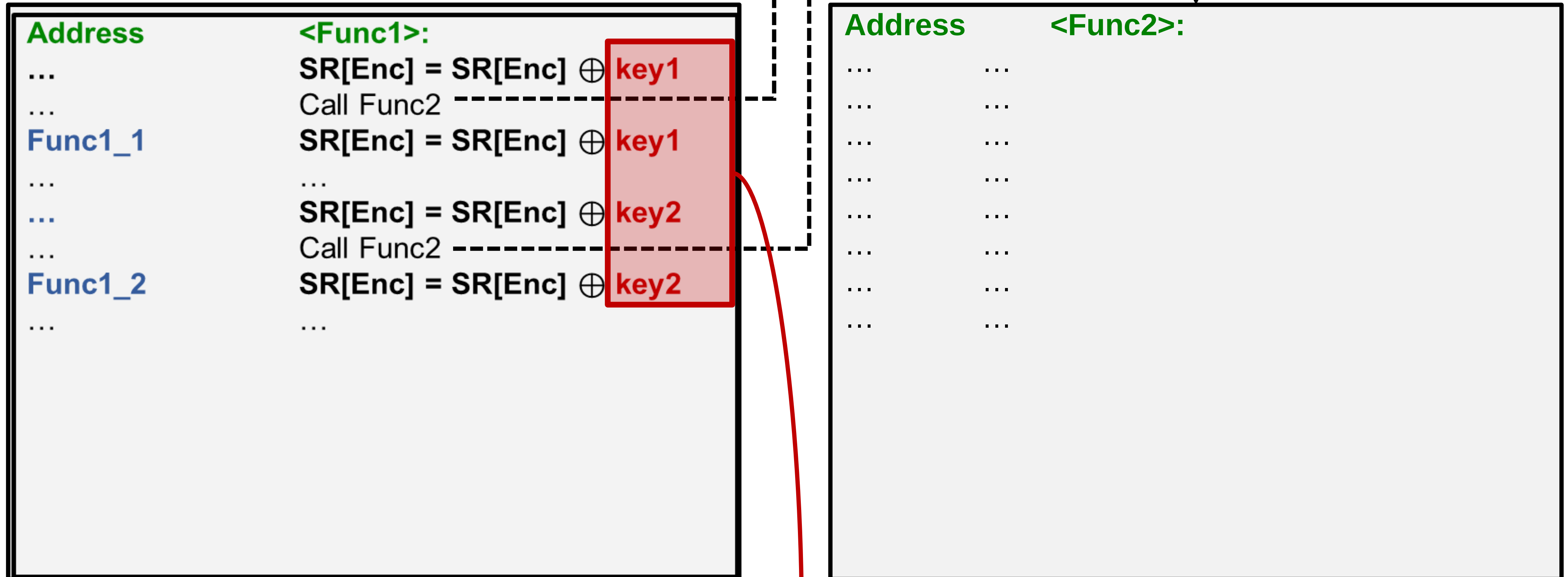
- **State Register (SR):**
  - Can be any general-purpose register → exclusively used by µRAI
  - Never spilled → cannot be overwritten through a memory corruption
  - Does not contain a return address → encoded values to resolve the return location

- **Example call graph:**
  - Each edge → call



- **How encode SR?**
- **An XOR chain**

µRAI: Terminology



- **Function Keys (FKs): Hard-coded keys used to encode the SR**

µRAI: Terminology

SR[Recursive]	SR[Encoded]
0	C

Address	<Func1>:
…	SR[Enc] = SR[Enc] ⊕ key1
…	Call Func2
Func1_1	SR[Enc] = SR[Enc] ⊕ key1
…	…
…	SR[Enc] = SR[Enc] ⊕ key2
…	Call Func2
Func1_2	SR[Enc] = SR[Enc] ⊕ key2
…	…

Function ID (FID)	Return Target
C	Jump return_location1
ELSE	Jump ERROR

Address	<Func2>:
…	…
…	…
…	…
…	…
…	…
…	…
…	…
…	…

Function ID (FID)	Return Target
C ⊕ key1	Jump Func1_1
C ⊕ key2	Jump Func1_2
ELSE	Jump ERROR

- **Function IDs (FIDs): Possible values of the SR for the function**

µRAI: Terminology

SR[Recursive]	SR[Encoded]
0	C

```
Address      <Func1>:
...          SR[Enc] = SR[Enc] ⊕ key1
...          Call Func2
Func1_1      SR[Enc] = SR[Enc] ⊕ key1
...          ...
...          SR[Enc] = SR[Enc] ⊕ key2
...          Call Func2
Func1_2      SR[Enc] = SR[Enc] ⊕ key2
...          ...
```

Function ID (FID)	Return Target
C	Jump return_location1
ELSE	Jump ERROR

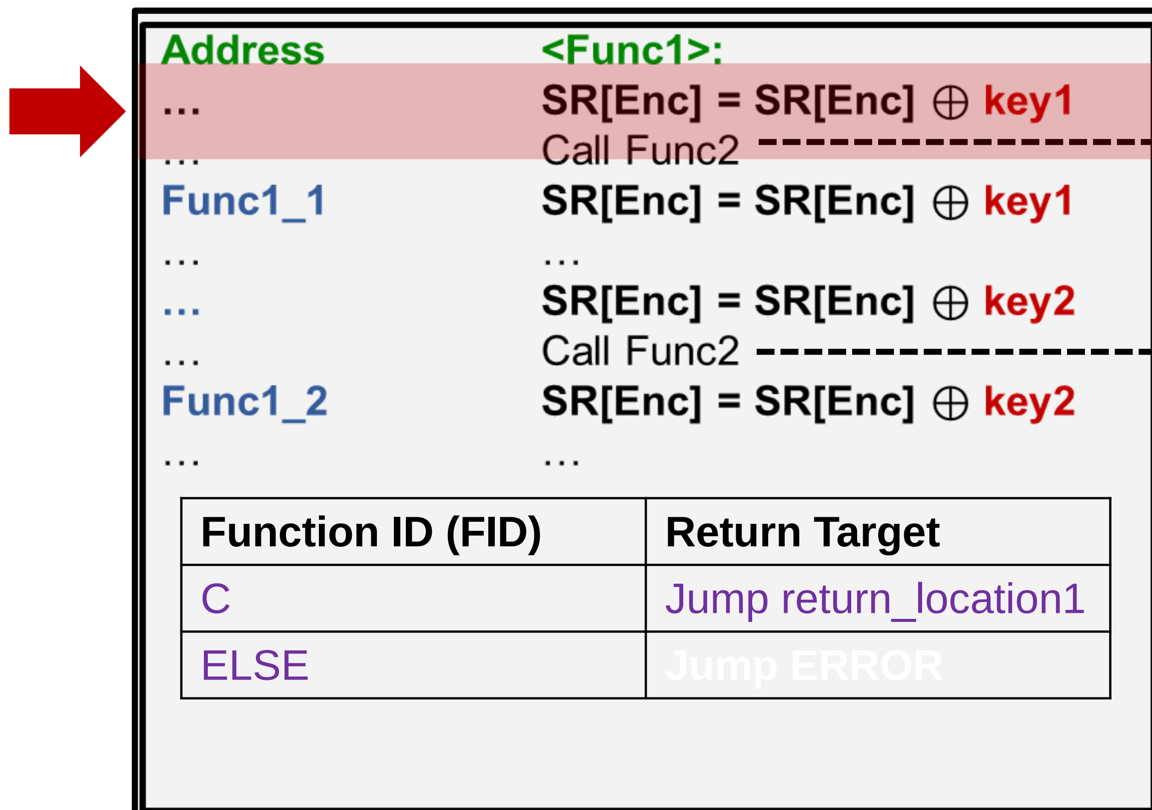
```
Address      <Func2>:
...          ...
...          ...
...          ...
...          ...
...          ...
...          ...
...          ...
...          ...
```

Function ID (FID)	Return Target
C ⊕ key1	Jump Func1_1
C ⊕ key2	Jump Func1_2
ELSE	Jump ERROR

- **Function Lookup Table (FLT): List of FIDs for the function.**

µRAI: Transformation

SR[Recursive]	SR[Encoded]
0	C



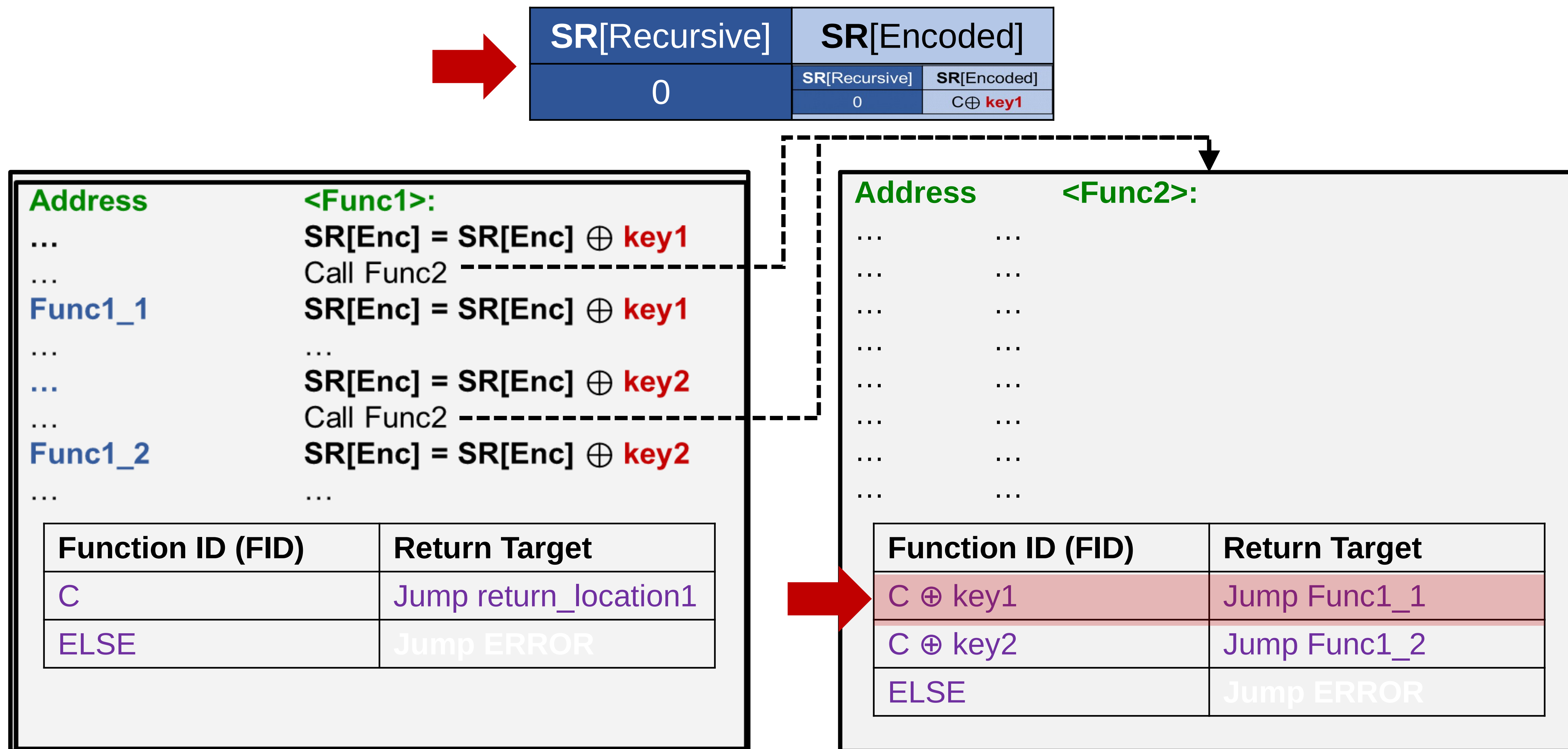
Address <Func2>:

…	…
…	…
…	…
…	…
…	…
…	…
…	…
…	…

Function ID (FID)	Return Target
C ⊕ key1	Jump Func1_1
C ⊕ key2	Jump Func1_2
ELSE	Jump ERROR

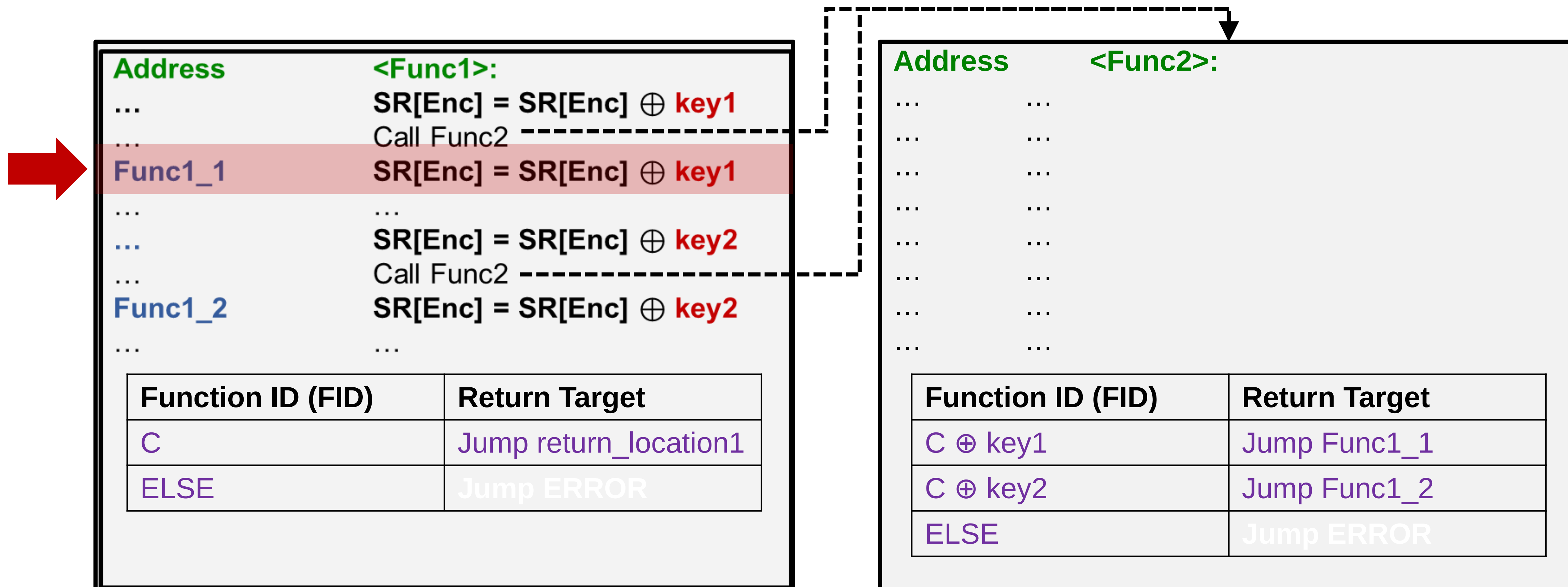
- **Encode the SR and call Func2**

µRAI: Transformation



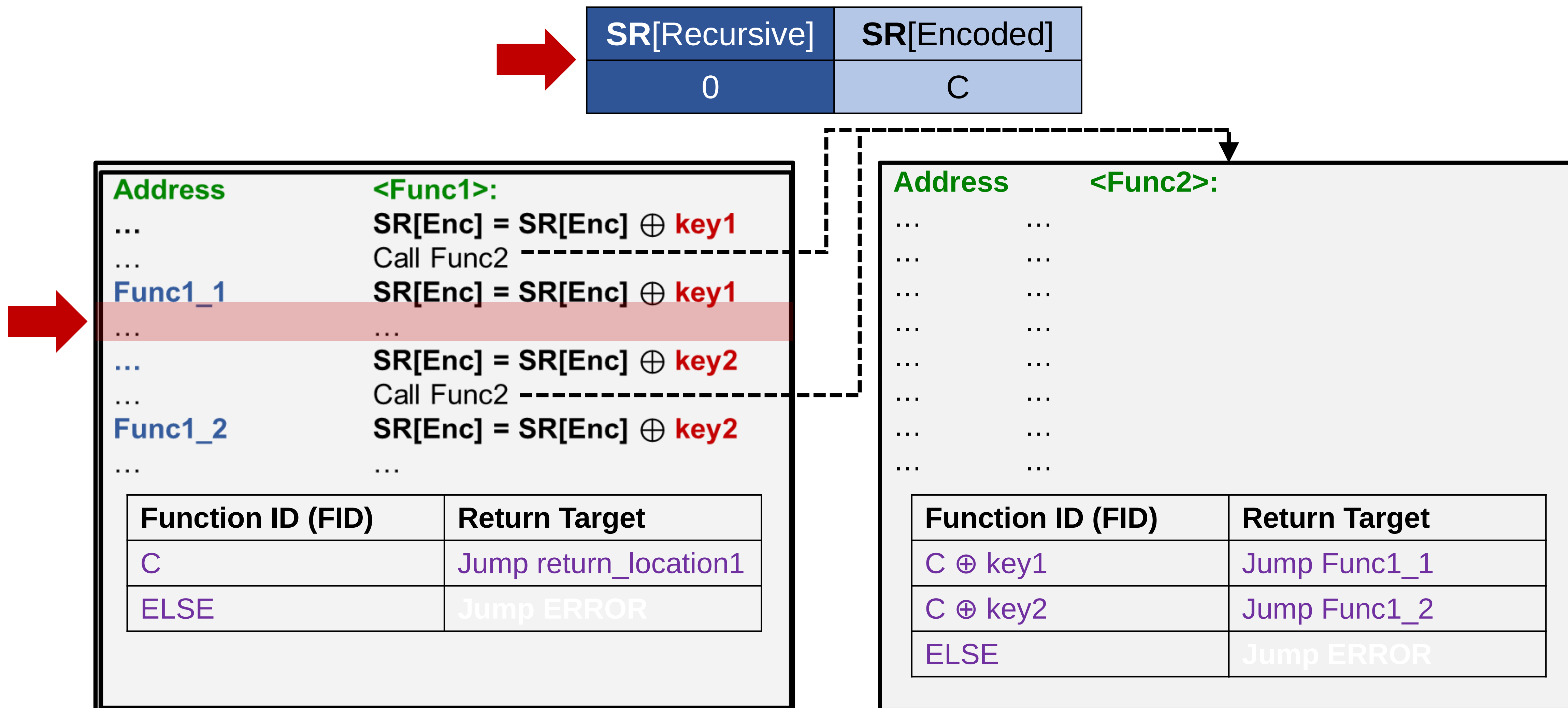
- **Func2 reads the SR and executes the corresponding direct jump**

µRAI: Transformation



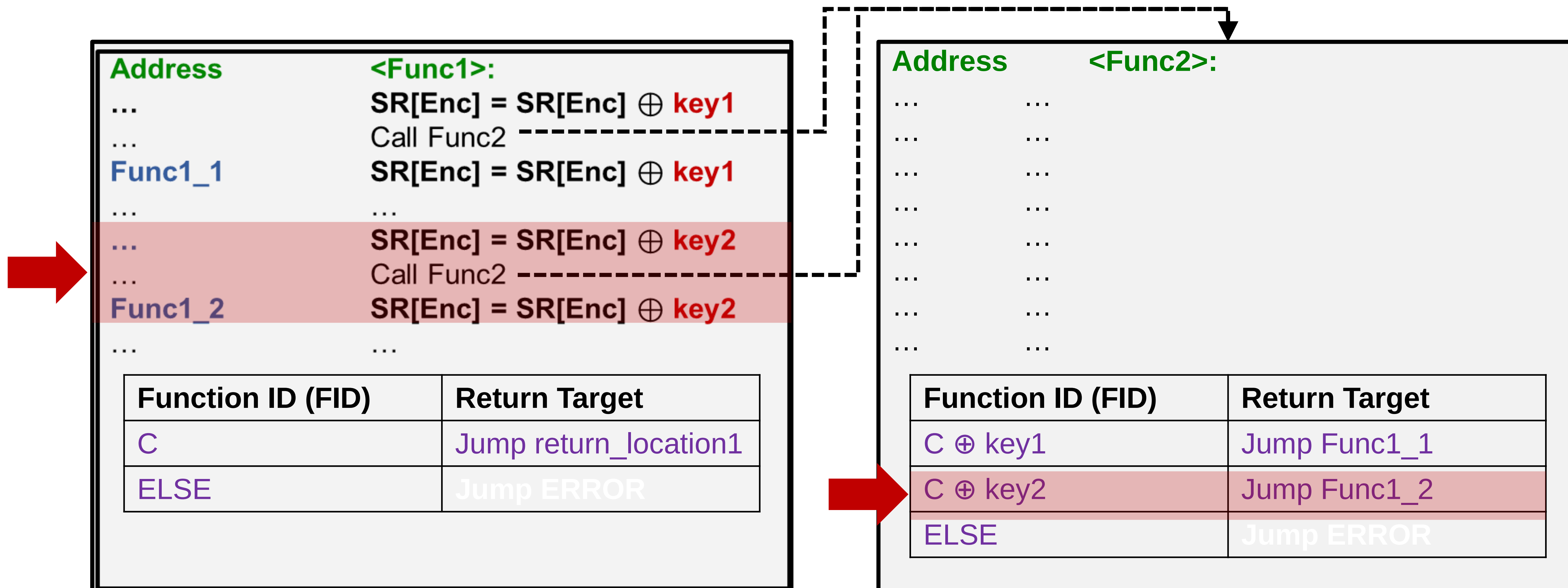
- **Func2 returns correctly and the SR is decoded**

µRAI: Transformation



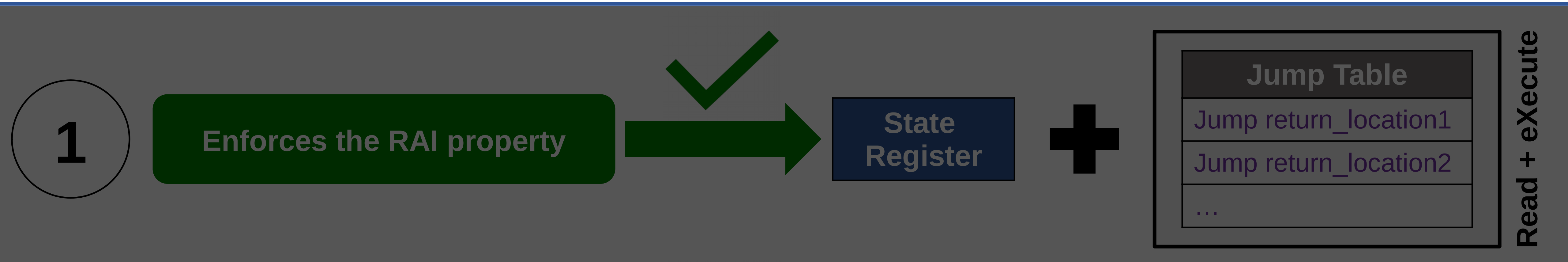
- **The previous SR value is restored**

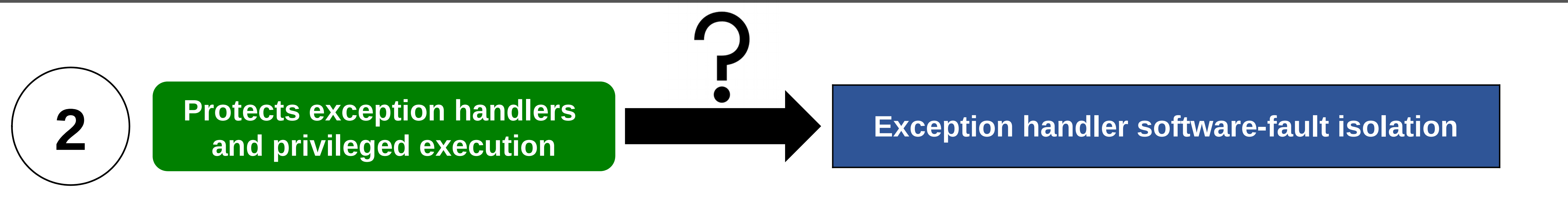
µRAI: Transformation

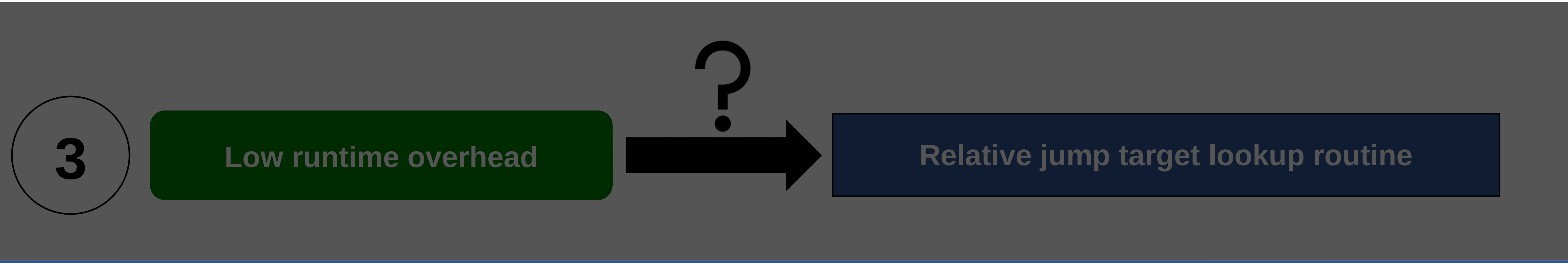


- The same happens for other calls. Func1 can then return correctly

μRAI: Overview

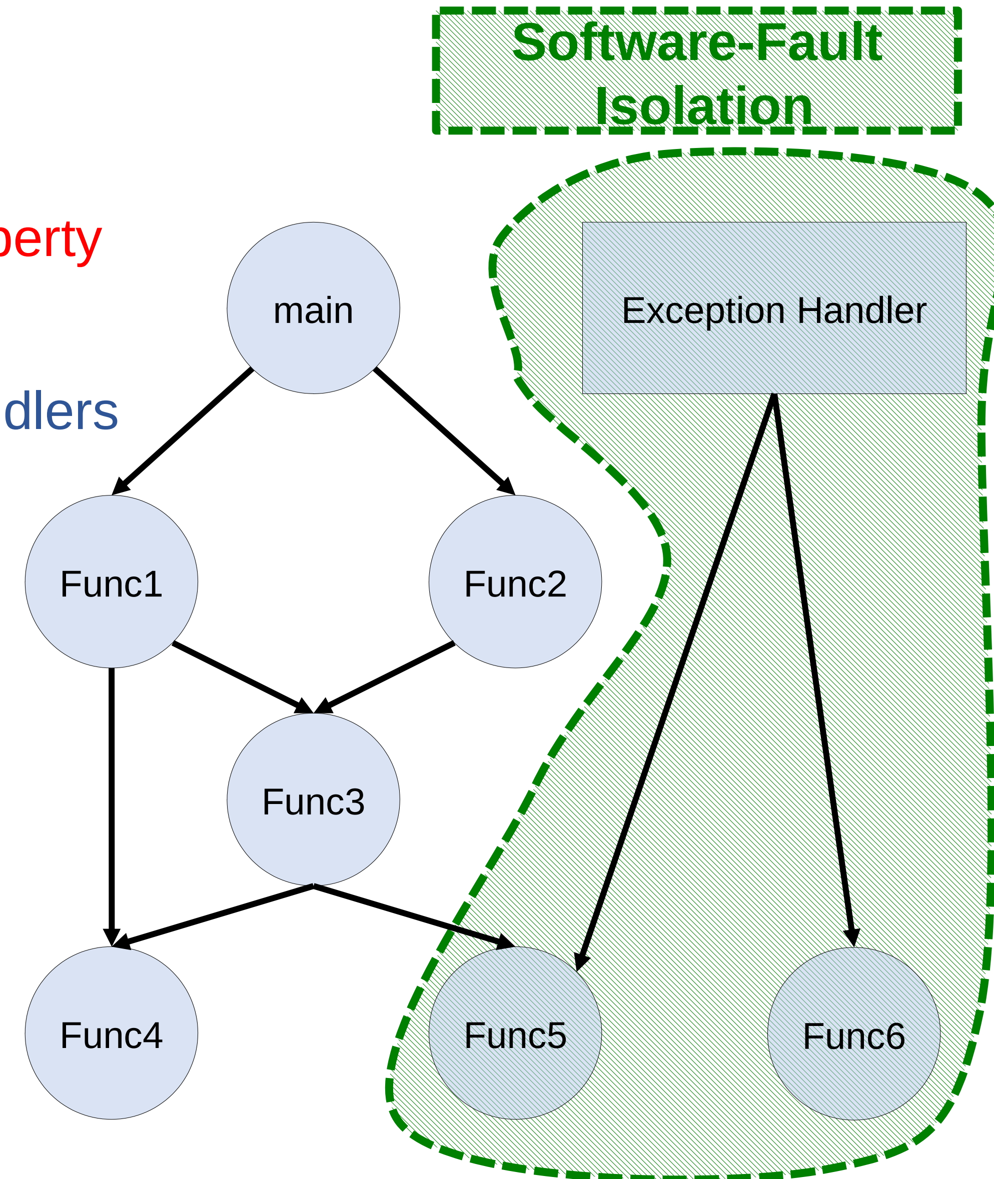




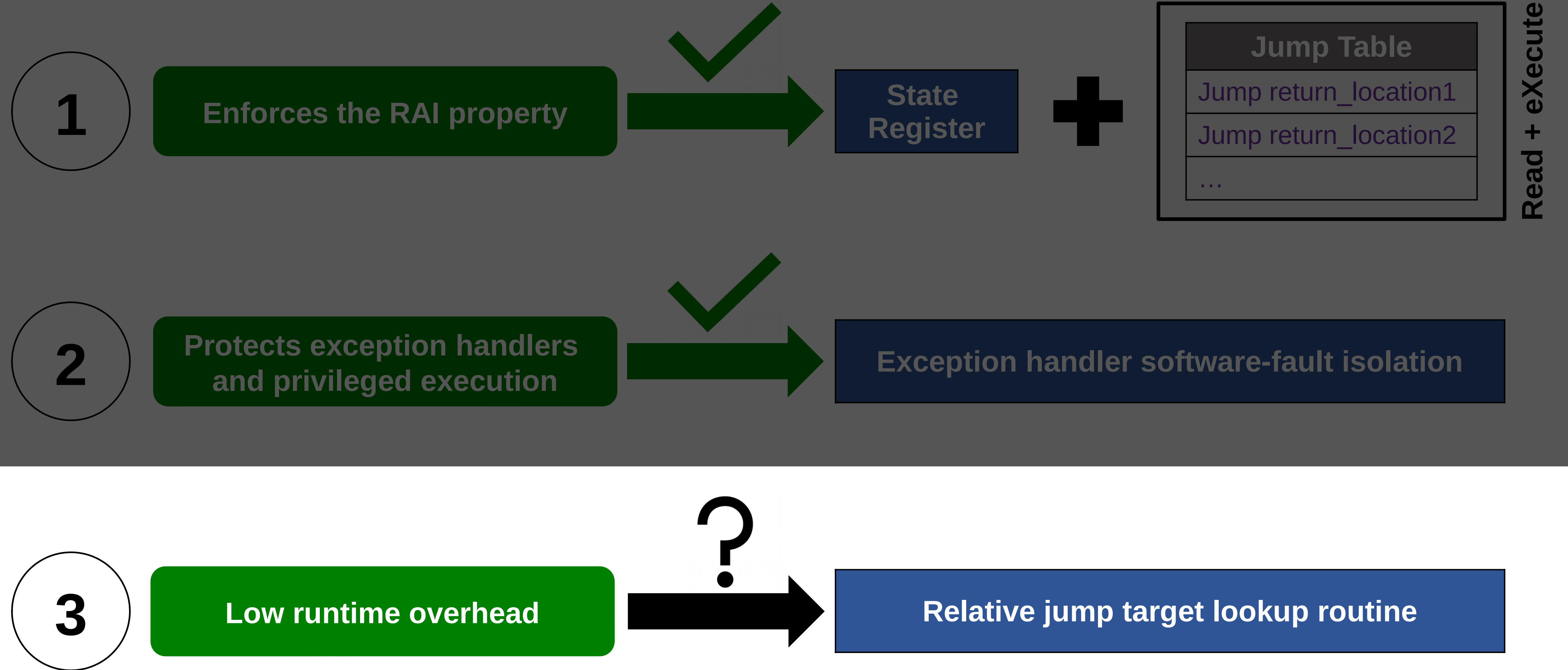


µRAI: Enforce RAI for exception handlers

- **Exception handlers execute with privileges**
  - Can disable the MPU → enable code injection
  - Can corrupt exception stack frame → break RAI property
- **Solution:**
  - Apply SFI only to functions callable by exception handlers
  - Limit SFI overhead compared to full-SFI

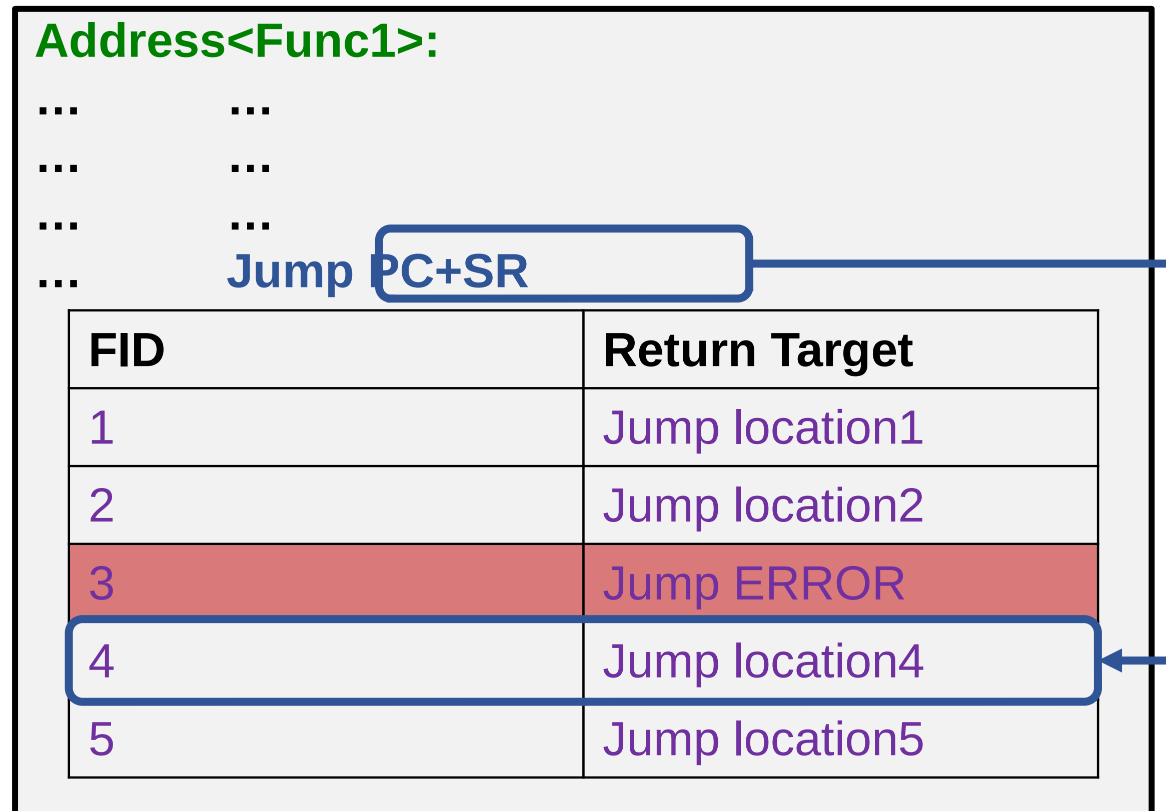


µRAI: Overview



Target Lookup Routine (TLR)

- **How can we find the correct direct jump in the FLT efficiently?**
  - Use a relative jump before the FLT
  - Resolve the correct return location efficiently regardless of FLT size



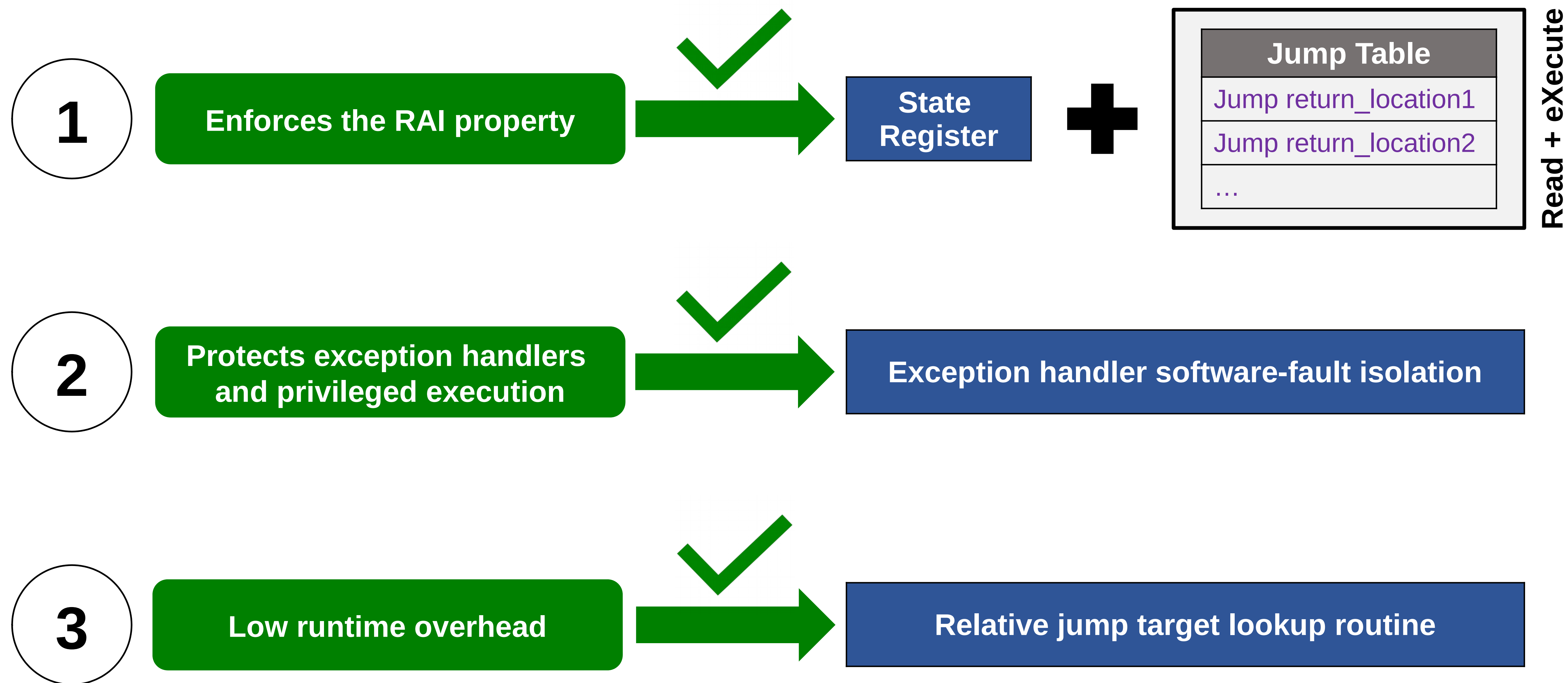
Assume the correct return location is **location4**

Comparing all FID can be slow!

Use SR as an index of a jump table

Align the FLT make SR = 4

µRAI: Overview



Evaluation

- **Five MCUS applications on Cortex-M4:**
  - PinLock
  - FatFs_uSD
  - FatFs_RAM
  - LCD_uSD
  - Animation

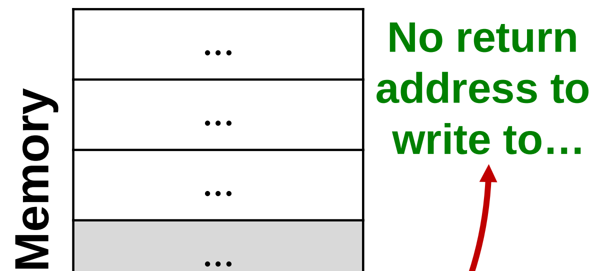
- **CoreMark benchmark[1]**
  - Standard MCUS performance benchmark

[1] EEMBC, "Coremark - industry-standard benchmarks for embedded systems," http://www.eembc.org/coremark

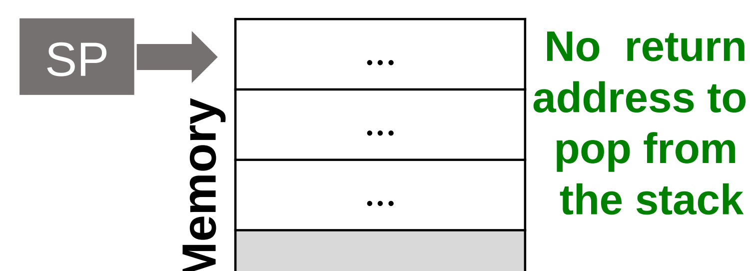
Security Evaluation Using PinLock: Unlock The Lock

Buffer overflow

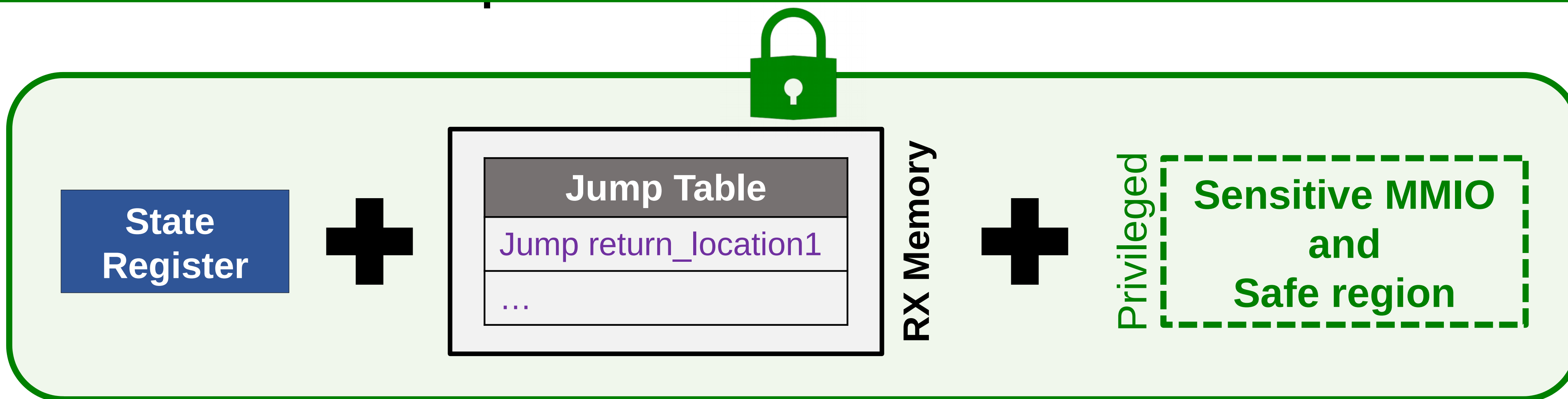
Arbitrary write



Stack pivot

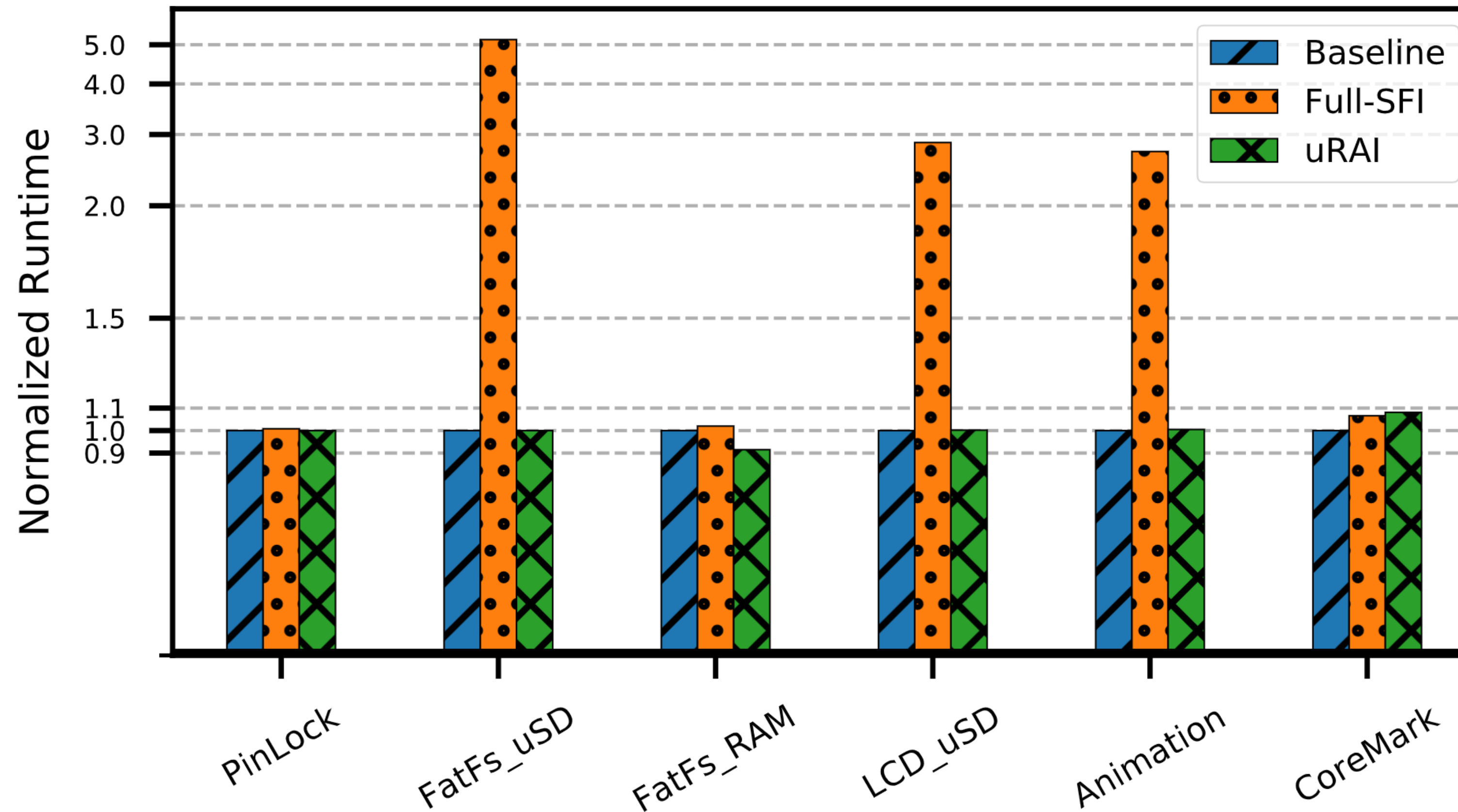


µRAI prevents all control-flow hijacking attack scenarios targeting return addresses



PURDUE UNIVERSITY
EPFL
Sandia National Laboratories
DCSL
hexhive
King Saud University

Performance results



- **Requiring full-SFI results in high overhead ⇒ average of 130.5%**
- **µRAI results in low overhead ⇒ average of 0.1%**

µRAI: Conclusion

- **Control-flow hijacking on MCUS is a threat**

- **µRAI secures MCUS against control-flow hijacking**
  - Enforces the RAI property for MCUS → protects backward edges
  - Complemented with type-based CFI → end-to-end code pointer protection

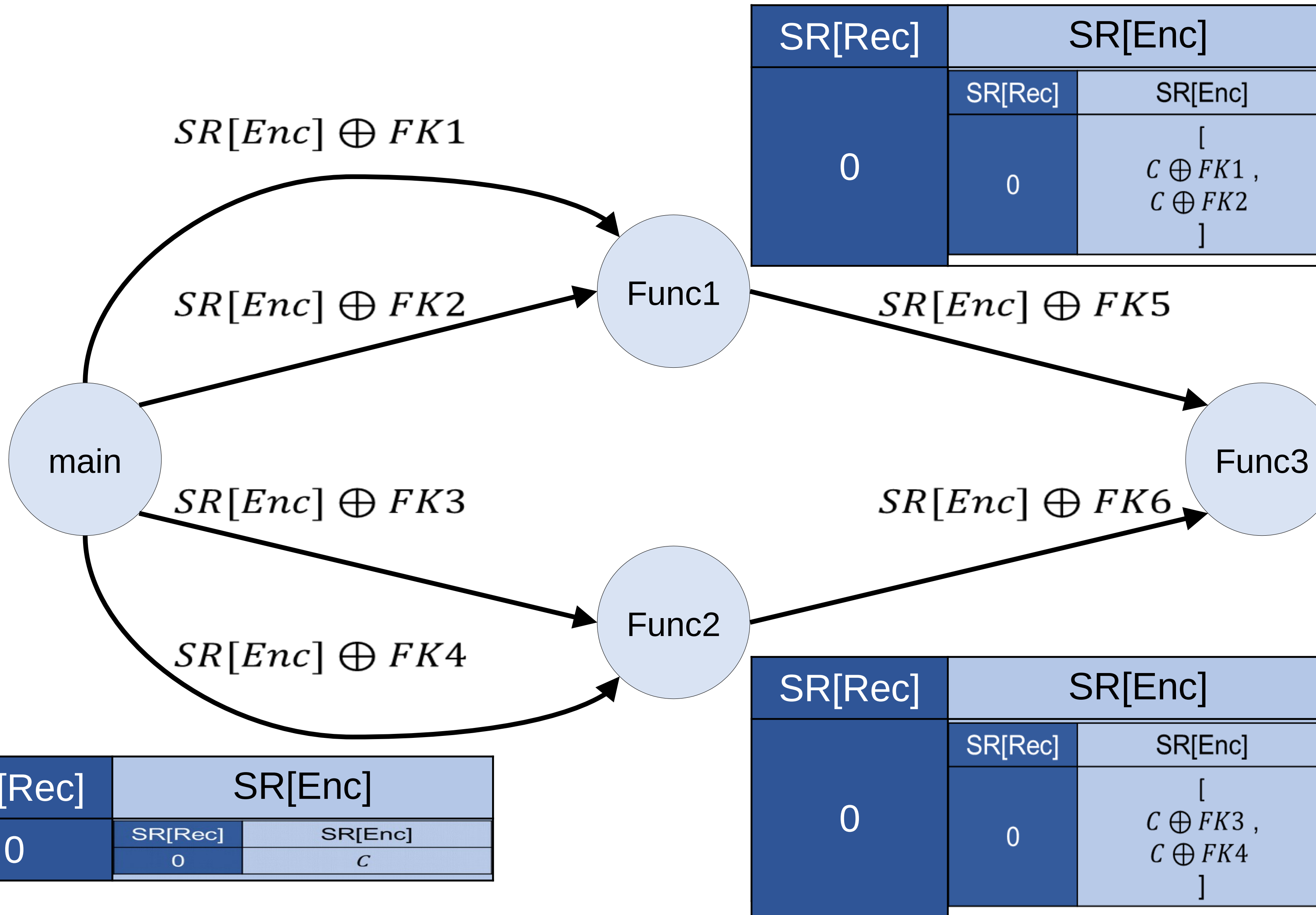
- **Presents a portable encoding scheme**
  - Does not require special hardware features (only a register and an MPU)
  - Applicable to other systems

- **Low runtime overhead**

https://github.com/embedded-sec/uRAI

Backup Slides

Challenge: Path Explosion



- **Func3 has 2 call sites**
- **FLT is size is 4!**

SR[Rec]	SR[Enc]	
	SR[Rec]	SR[Enc]
0	0	[$C \oplus FK1 \oplus FK5$, $C \oplus FK2 \oplus FK5$, $C \oplus FK3 \oplus FK6$, $C \oplus FK4 \oplus FK6$]

Path Explosion Solution: Segmentation

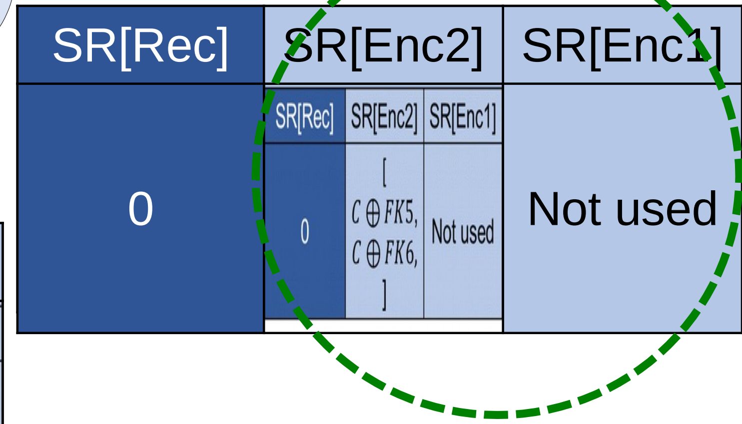
- **State register segmentation**

Recursion counter (Higher N bits)			Encoded value (Lower 32-N bits)			
Segment 1	...	Segment K	Segment M	…	Segment 2	Segment 1

- **Functions only use the bits in their assigned segment.**

Path Explosion Solution: Segmentation

- **Segmentation reduces FLT from 4 to 2**



SR[Rec]	SR[Enc2]			SR[Enc1]		
	SR[Rec]	SR[Enc2]	SR[Enc1]	SR[Rec]	SR[Enc2]	SR[Enc1]
0	0	C	C	0	C	C

$SR[Enc1] \oplus FK1$

$SR[Enc1] \oplus FK2$

$SR[Enc1] \oplus FK3$

$SR[Enc1] \oplus FK4$

$SR[Enc2] \oplus FK5$

$SR[Enc2] \oplus FK6$

main

Func1

Func2

Func3

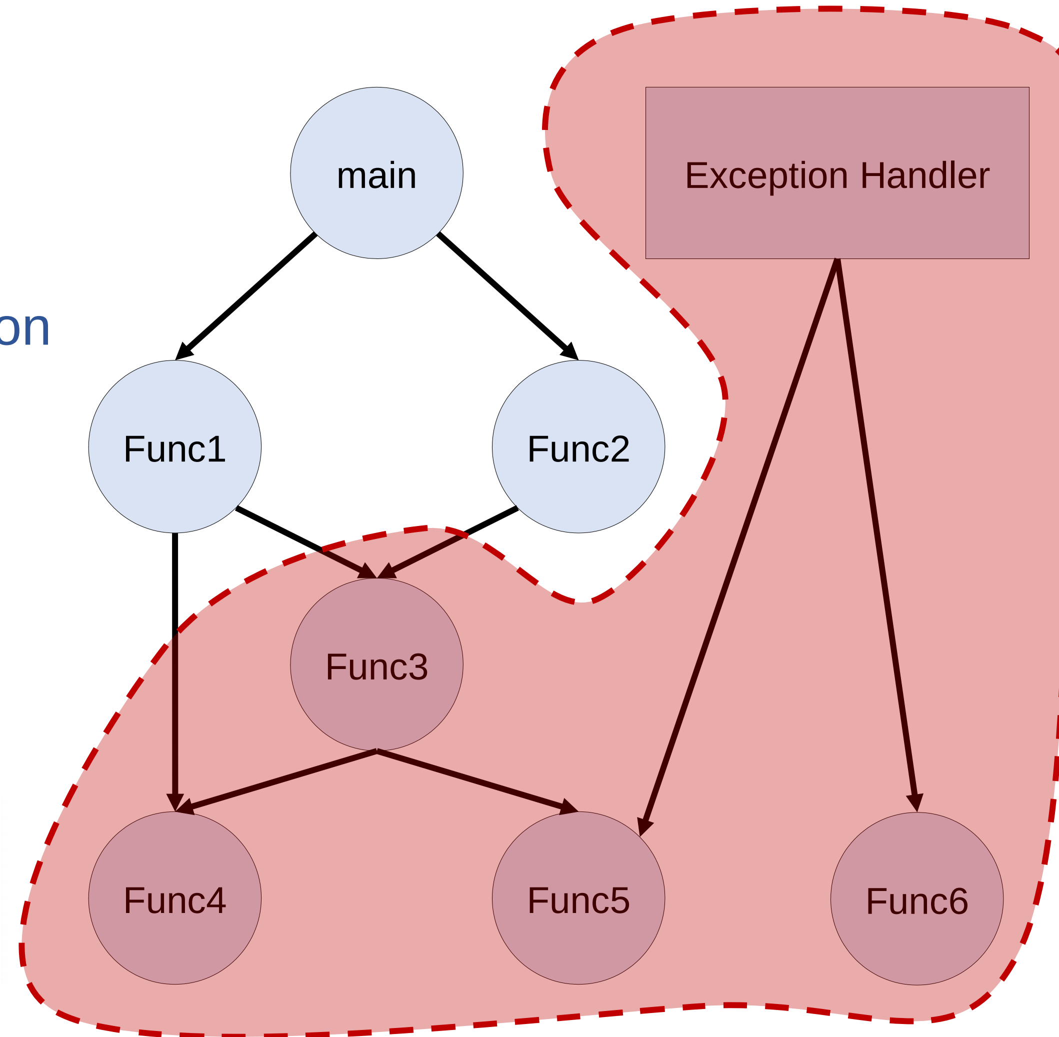
SR[Rec]	SR[Enc2]			SR[Enc1]		
	SR[Rec]	SR[Enc2]	SR[Enc1]	SR[Rec]	SR[Enc2]	SR[Enc1]
0	0	C	[$C \oplus FK1$, $C \oplus FK2$]	0	C	[$C \oplus FK1$, $C \oplus FK2$]

SR[Rec]	SR[Enc2]			SR[Enc1]		
	SR[Rec]	SR[Enc2]	SR[Enc1]	SR[Rec]	SR[Enc2]	SR[Enc1]
0	0	C	[$C \oplus FK3$, $C \oplus FK4$]	0	C	[$C \oplus FK3$, $C \oplus FK4$]

SR[Rec]	SR[Enc2]			SR[Enc1]
	SR[Rec]	SR[Enc2]	SR[Enc1]	
0	0	[$C \oplus FK5$, $C \oplus FK6$,]	Not used	Not used

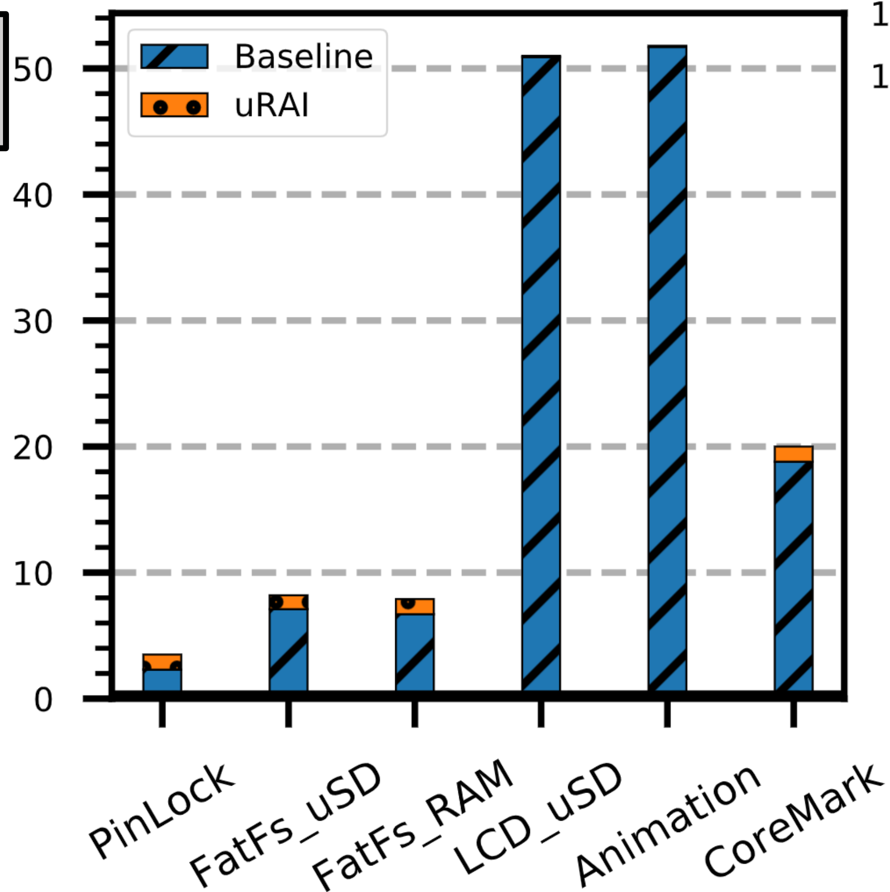
µRAI: Scalability

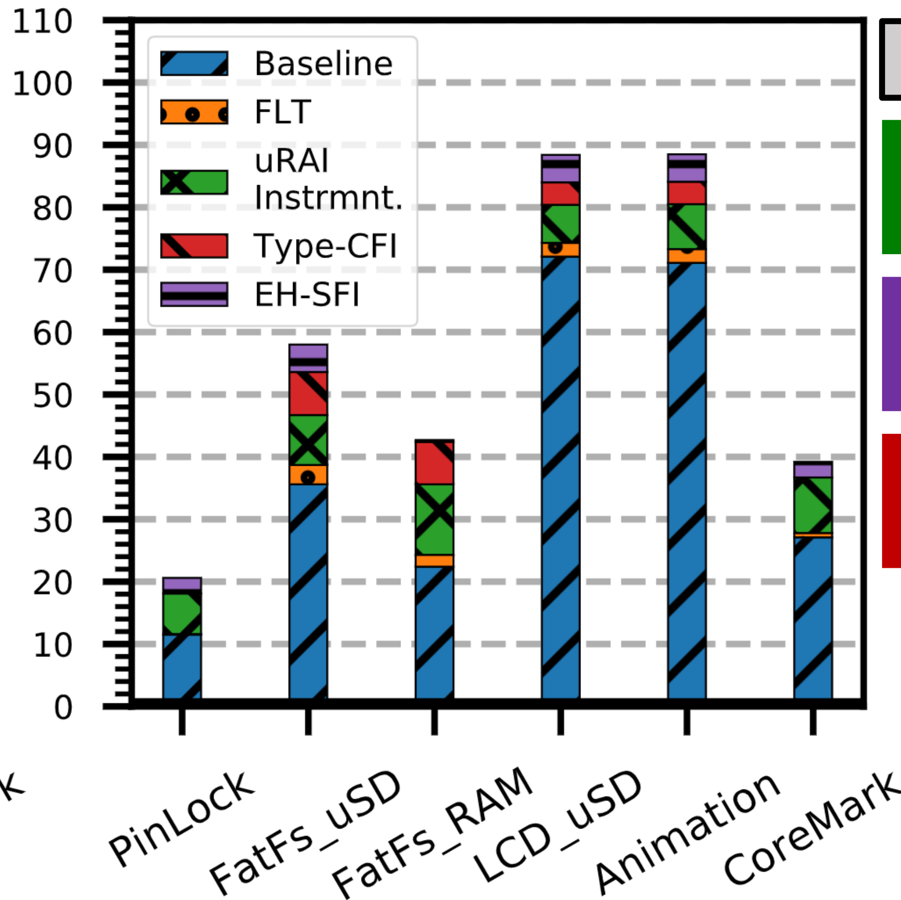
- **What if no more values can be found for the SR?**
- **Solution:**
  - Partition the call graph if no solution is found
  - Entering a new partition
    - Save and reset the SR to a privileged safe region
  - Returning to a previous partition
    - Restore the SR



Memory results

Average 15.2%





Average

FLT + Instrmnt. 34.6%

EH-SFI 9.5%

Type-CFI 10%

- **Moderate overhead for large applications**

µRAI vs. backward edge Type-based CFI

App	Type-based CFI Target Set	
	Max.	Ave.
PinLock	8	3
FatFs_uSD	94	21
FatFs_RAM	94	27
LCD_uSD	49	11
Animation	49	11
CoreMark	52	12
Overall Average	**58**	**14**

µRAI eliminates the remaining attack surface for control-flow bending attacks[1]

- **[1] N. Carlini, A. Barresi, M. Payer, D. Wagner, and T. R. Gross, "Controlflow bending: On the effectiveness of control-flow integrity," in USENIX SEC15**

Store Instructions Protected with EH-SFI

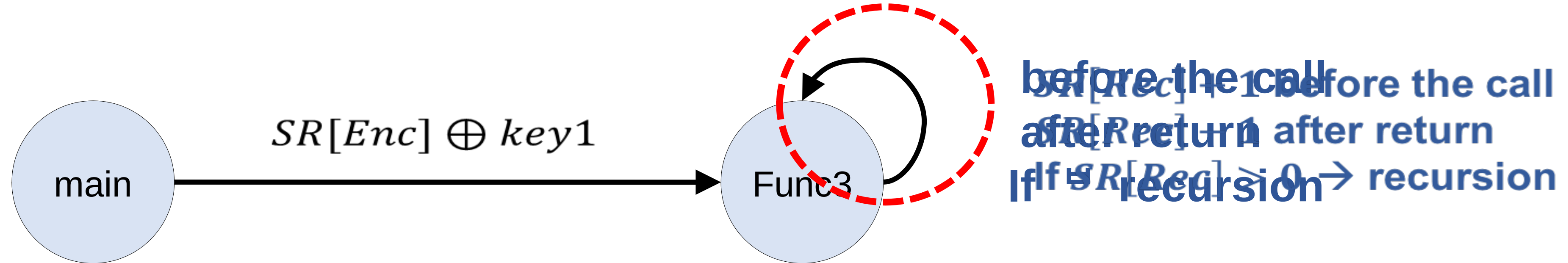
App	# of Store instruction			
	Static	Total	(Static/Total)%	Dynamic
PinLock	56	516	10.9	7
FatFs_uSD	99	1,802	5.5	906K
FatFs_RAM	7	1,116	0.6	7
LCD_uSD	99	2,814	3.5	48K
Animation	99	2,760	3.6	66K
CoreMark	56	1,024	5.5	7

SR Layout

- **The SR has two parts:**
  - ENC: Encoded value
  - REC: Recursion counter

- **Cannot use XOR with recursion**
- **Collision occurs with existing values Func1**

→ $\boldsymbol{SR \oplus ANY\ KEY \oplus ANY\ KEY = SR}$



SR[Rec]	SR[Enc]
0	SR[Rec]: 0, SR[Enc]: C

SR[Rec]	SR[Enc]
0	SR[Rec]: 0, SR[Enc]: $C \oplus key1$

µRAI: Transformation

SR[Rec]	SR[Enc]
0	C

Address	<Func1>:
…	SR[Enc] = SR[Enc] ⊕ key1
…	Call Func2
Func1_1	SR[Enc] = SR[Enc] ⊕ key1
…	…
…	SR[Enc] = SR[Enc] ⊕ key2
…	Call Func2
Func1_2	SR[Enc] = SR[Enc] ⊕ key2
…	…

Function ID (FID)	Return Target
C	Jump return_location1
ELSE	Jump ERROR

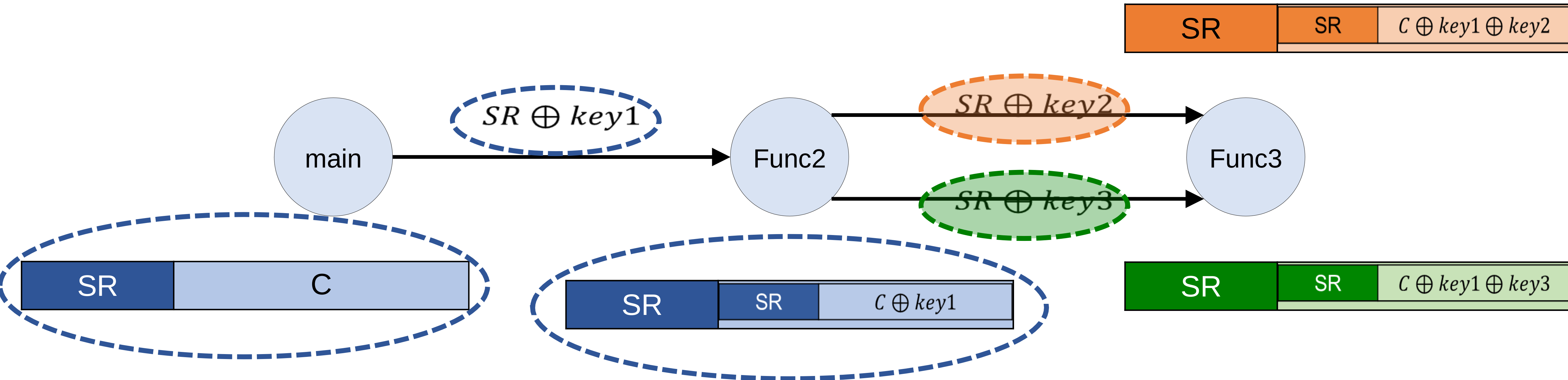
Address	<Func2>:
…	SR[Rec]++
…	Call Func2
Func2_1	SR[Rec]--
…	…
…	IF SR[Rec] > 0
…	Jump Func2_1
…	ELSE
…	…

Function ID (FID)	Return Target
C ⊕ key1	Jump Func1_1
C ⊕ key2	Jump Func1_2
ELSE	Jump ERROR

- **If recursive ⇒ use a counter (*recursion is discouraged in MCUS*)**

SR Encoding Illustration

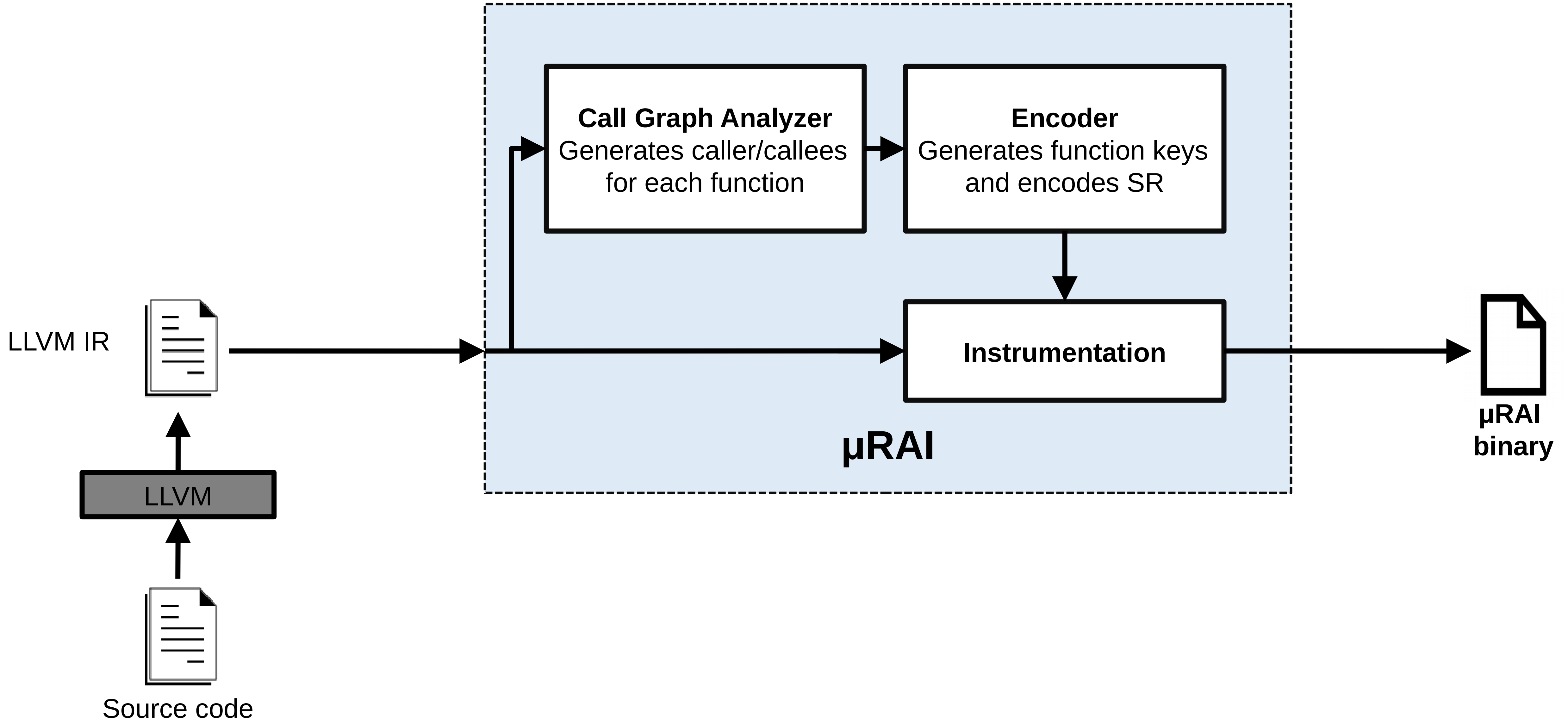
- **Consider the following call graph with the SR initialized to a value = C**
- **Each edge ⊟ call**
- **XOR before the call and after returning with *hardcoded keys***
  - An edge is walked ⊟ XOR the SR



XORing again after returning ⊟ restores the previous SR

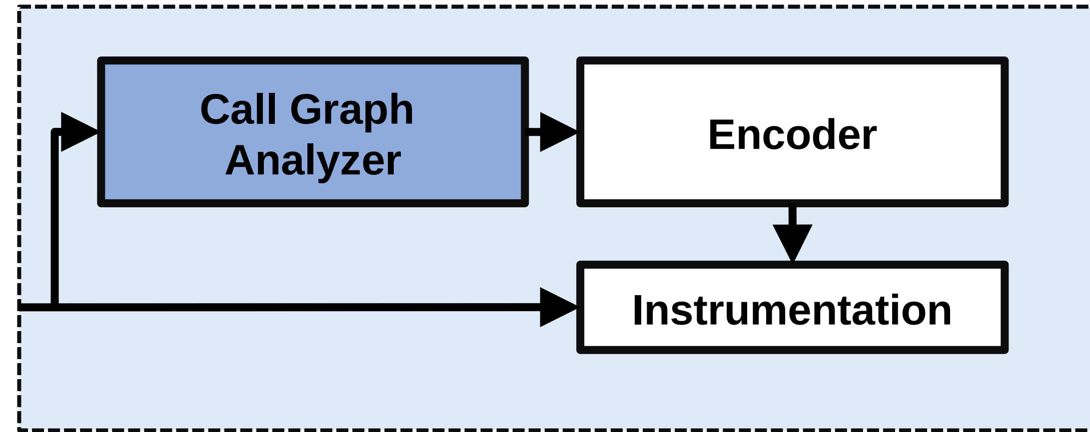
PURDUE UNIVERSITY EPFL Sandia National Laboratories hexhive King Saud University

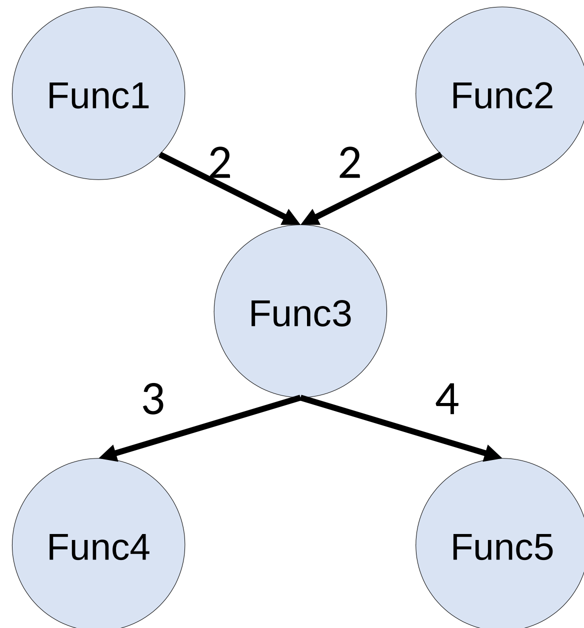
µRAI's Overview



µRAI: Workflow

- Generates caller/callee list.
- Sets the minimum possible FLT.
- Example: Func5 is called from 4 locations → Min. FLT >= 4

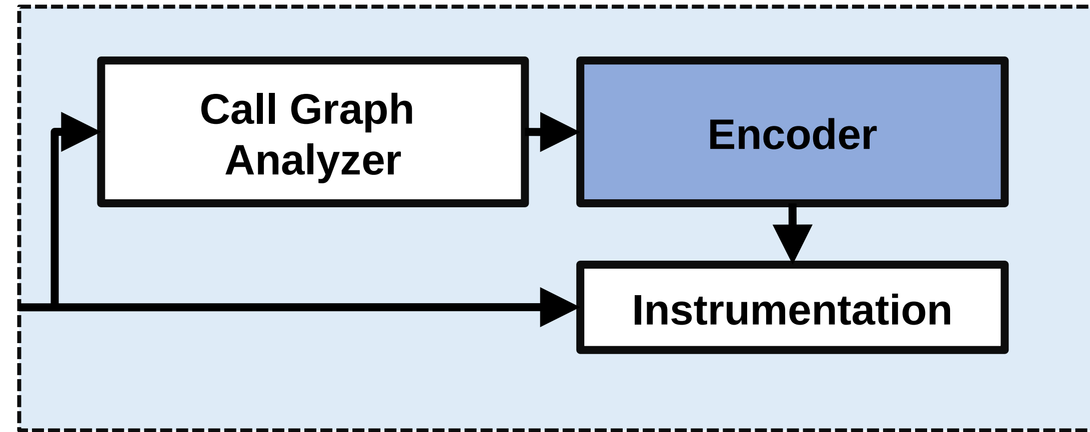


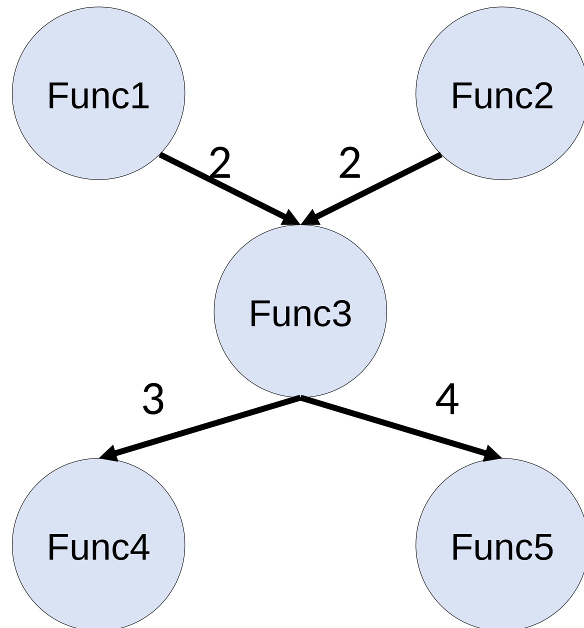


Function	Min. FLT	DFS FLT	Segmented FLT
Func1	-		
Func2	-		
Func3	4		
Func4	3		
Func5	4		

µRAI: Workflow

- Performs Depth First Search (DFS) on the call graph to generate initial FLT



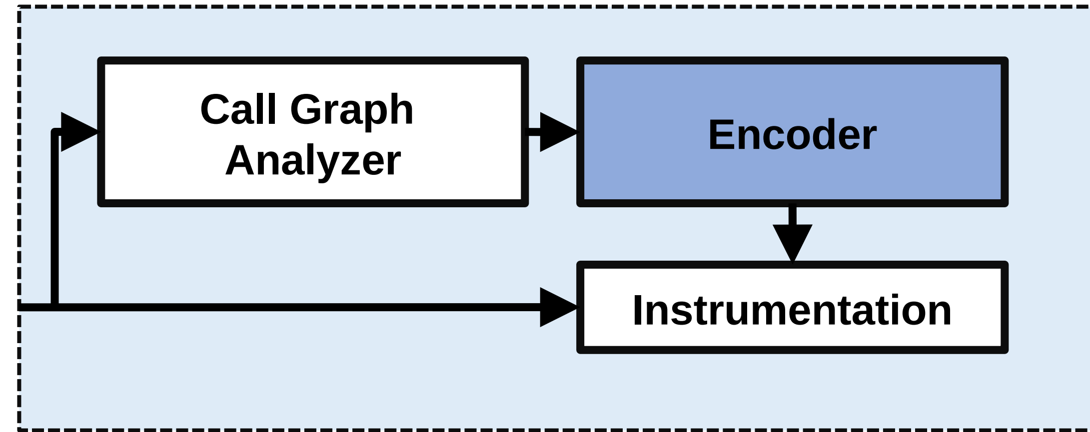


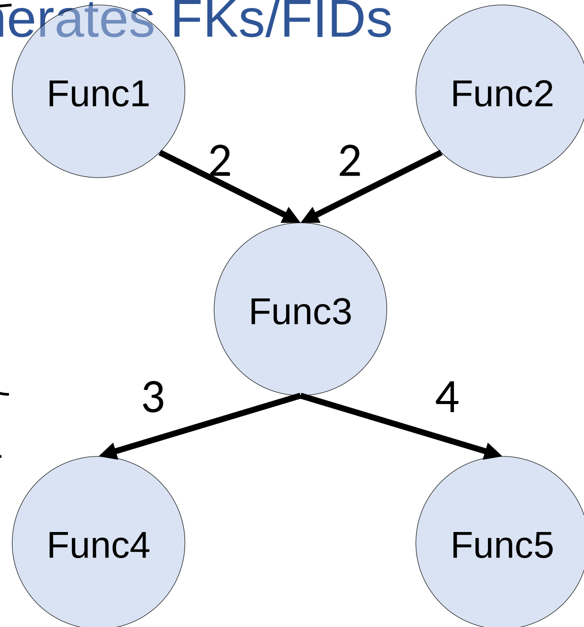
Function	Min. FLT	DFS FLT	Segmented FLT
Func1	-	**-**	-
Func2	-	**-**	-
Func3	4	**4**	
Func4	3	**12**	
Func5	4	**16**	

PURDUE UNIVERSITY EPFL Sandia National Laboratories hexhive جامعة الملك سعود King Saud University

µRAI: Workflow

- Performs Depth First Search (DFS) on the call graph to generate initial FLT
- Configures the SR segment size to reduce memory overhead
- Generates FKs/FIDs





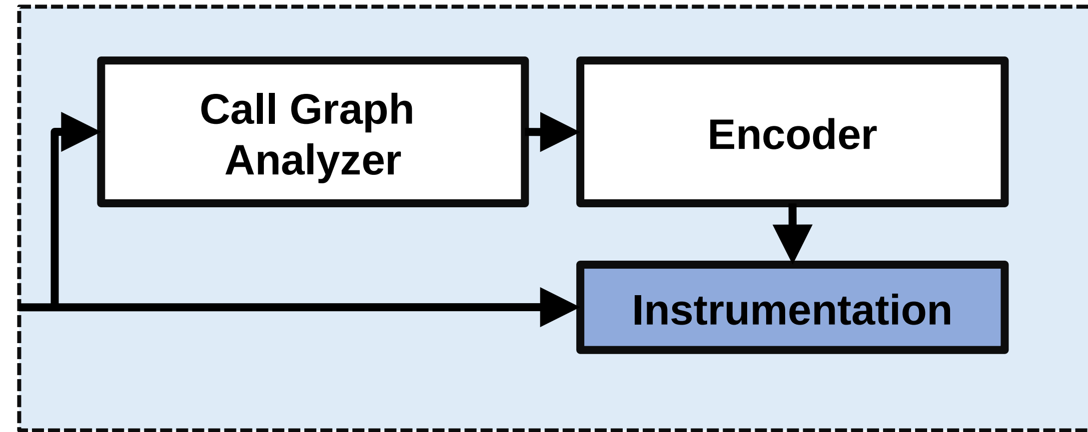
Use first SR segment

Use second SR segment

Function	Min. FLT	DFS FLT	Segmented FLT
Func1	-	-	**-**
Func2	-	-	**-**
Func3	4	4	**4**
Func4	3	12	**3**
Func5	4	16	**4**

µRAI: Workflow

- Instruments call sites with encoding/decoding instructions
- Remove any return instruction or uses of the SR
  - Example: `POP{PC}, PUSH{SR}`
- Instruments Target Lookup Routine (TLR) and FLT



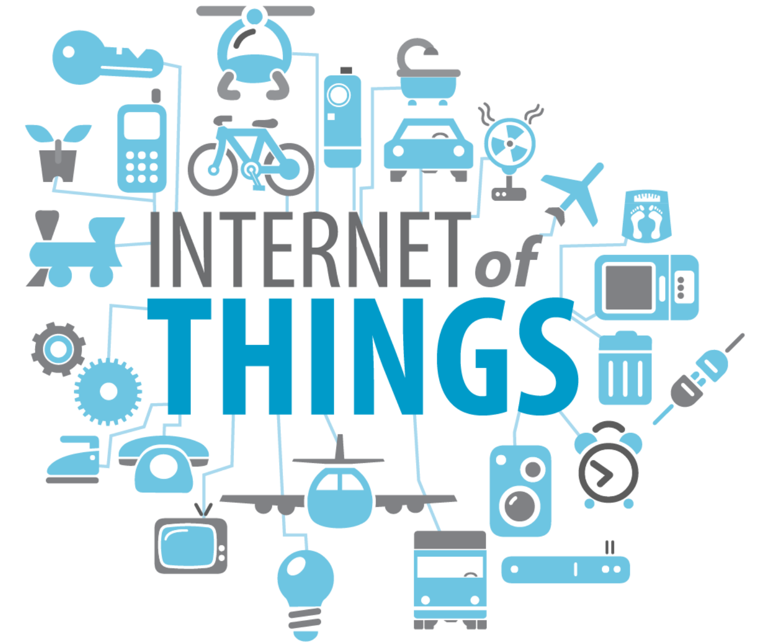

```
Address  <Func1>:
…        SR = SR ⊕ function_key
…        Call Func2
…        SR = SR ⊕ function_key
…        TLR: Execute the instruction where the FID = SR
```

FID	Return Target
1	Jump location1
2	Jump location2
…	…

Microcontroller Systems (MCUS)

- **Embedded systems and IoT run on Microcontroller systems (MCUS)**

- **MCUS:**
  - Run a single static binary application directly on hardware
  - Can be with/without an lightweight OS (bare-metal)
  - Direct access to peripherals and processor
  - Can be standalone device or part of larger system
  - Advanced hardware features are not commonly available
    - Example: *Trusted Execution Environment (TEE)*



- **Examples:**
  - WiFi System on Chip
  - Cyber-physical systems
  - UAVs

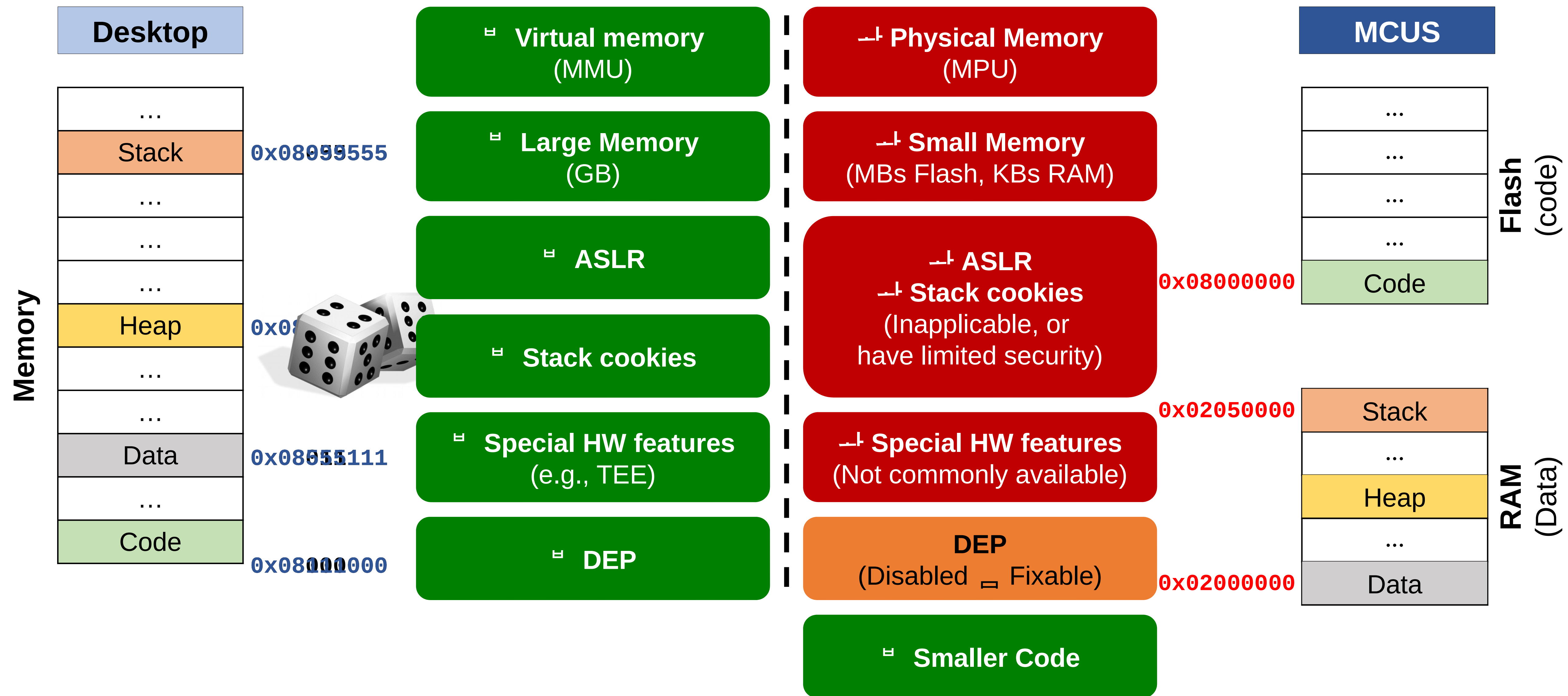
Security Evaluation Using PinLock

- **Attacker tries to unlock the lock using a vulnerability in `rx_from_uart`**
- **Attacker can read, write to anywhere in memory**
- **Attacker knows the entire code layout**
  - Even the current instance of the firmware

Attack	Prevented
Buffer overflow	ㅂ
Arbitrary write	ㅂ
Stack pivot	ㅂ

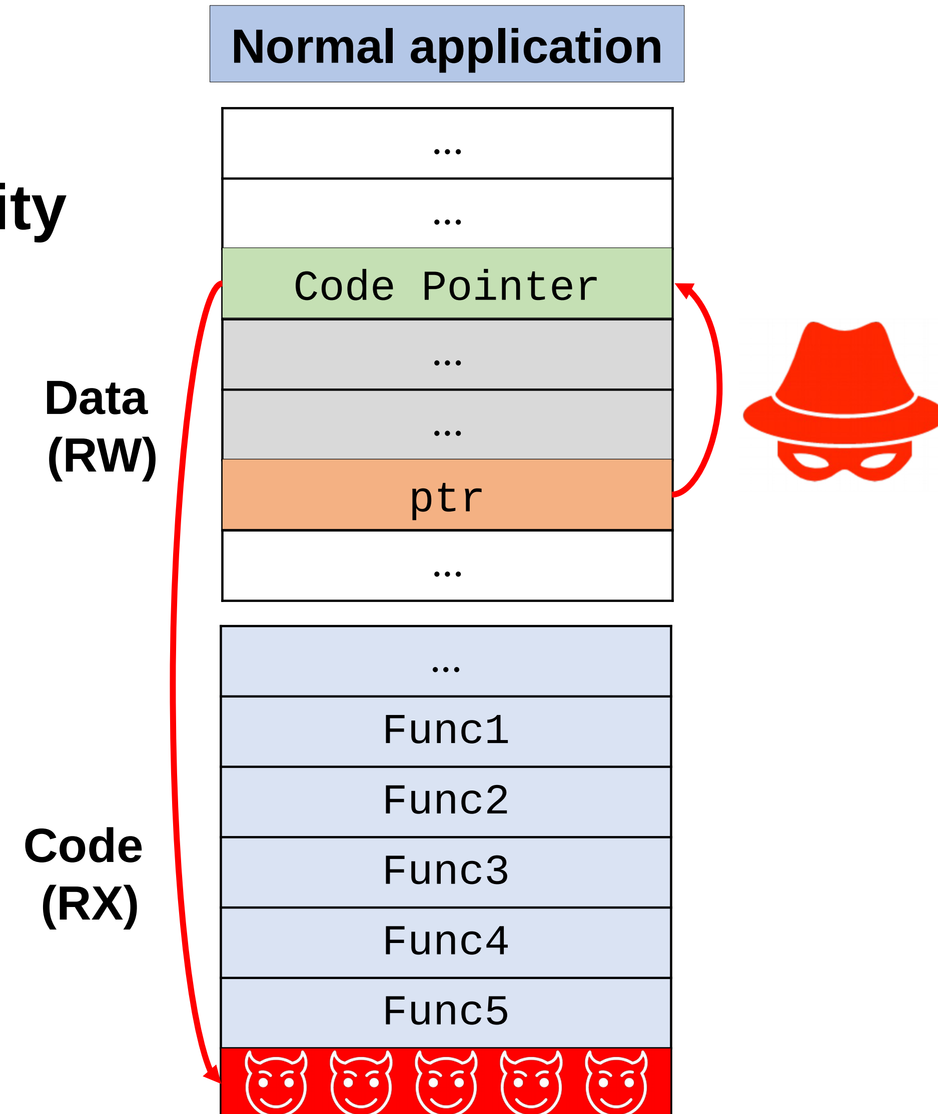
μRAI prevents all control-flow hijacking attack scenarios targeting return addresses

MCUS Challenges



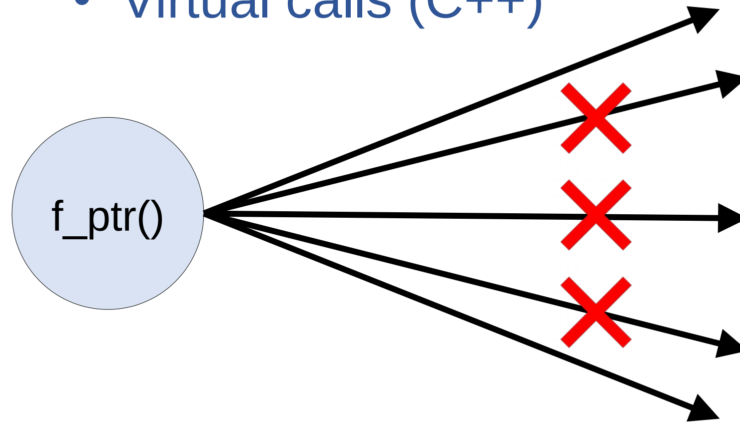
Control-Flow Hijacking

- **Attacker gains arbitrary execution**
- **Originates from memory corruption vulnerability**
- **Code pointers:**
  - Forward edges
  - Backward edges



Control-Flow Hijacking

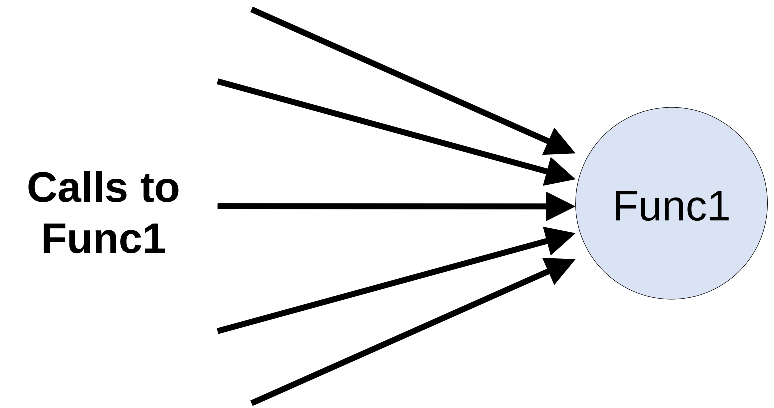
- **Forward edges:**
  - Function pointers
  - Virtual calls (C++)



- **Control-Flow Integrity (CFI):**
  - Calculates target set statically
  - Reduces target set significantly

- **Effective for MCUS**

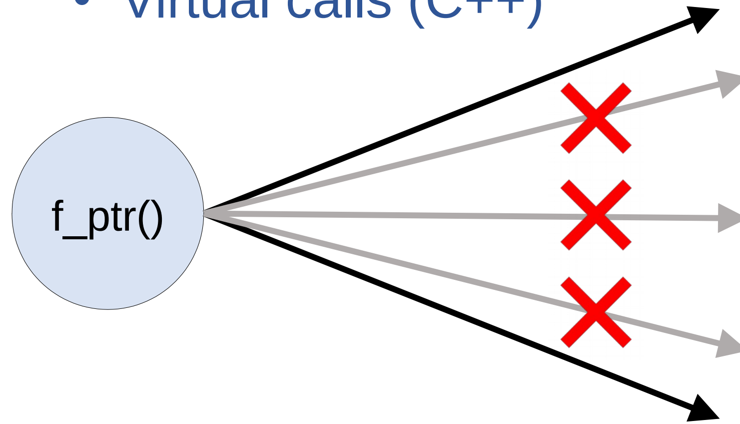
- **Backward edges:**
  - Return addresses



- **Current mechanisms:**
  - Limited security guarantees
    - Example: Large target set for CFI
  - High runtime overhead
  - Require special hardware

Control-Flow Hijacking

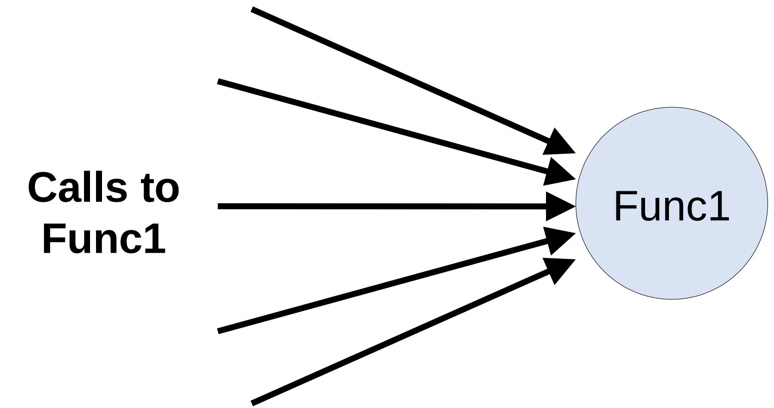
- **Forward edges:**
  - Function pointers
  - Virtual calls (C++)



- **Control-Flow Integrity (CFI):**
  - Calculates target set statically
  - Reduces target set significantly

- **Effective for MCUS** ᄇ

- **Backward edges:**
  - Return addresses



- **Control-flow hijacking attacks on backward-edges remain a threat.**
- **Example: Return Oriented Programming (ROP)**

Control-flow hijacking defenses

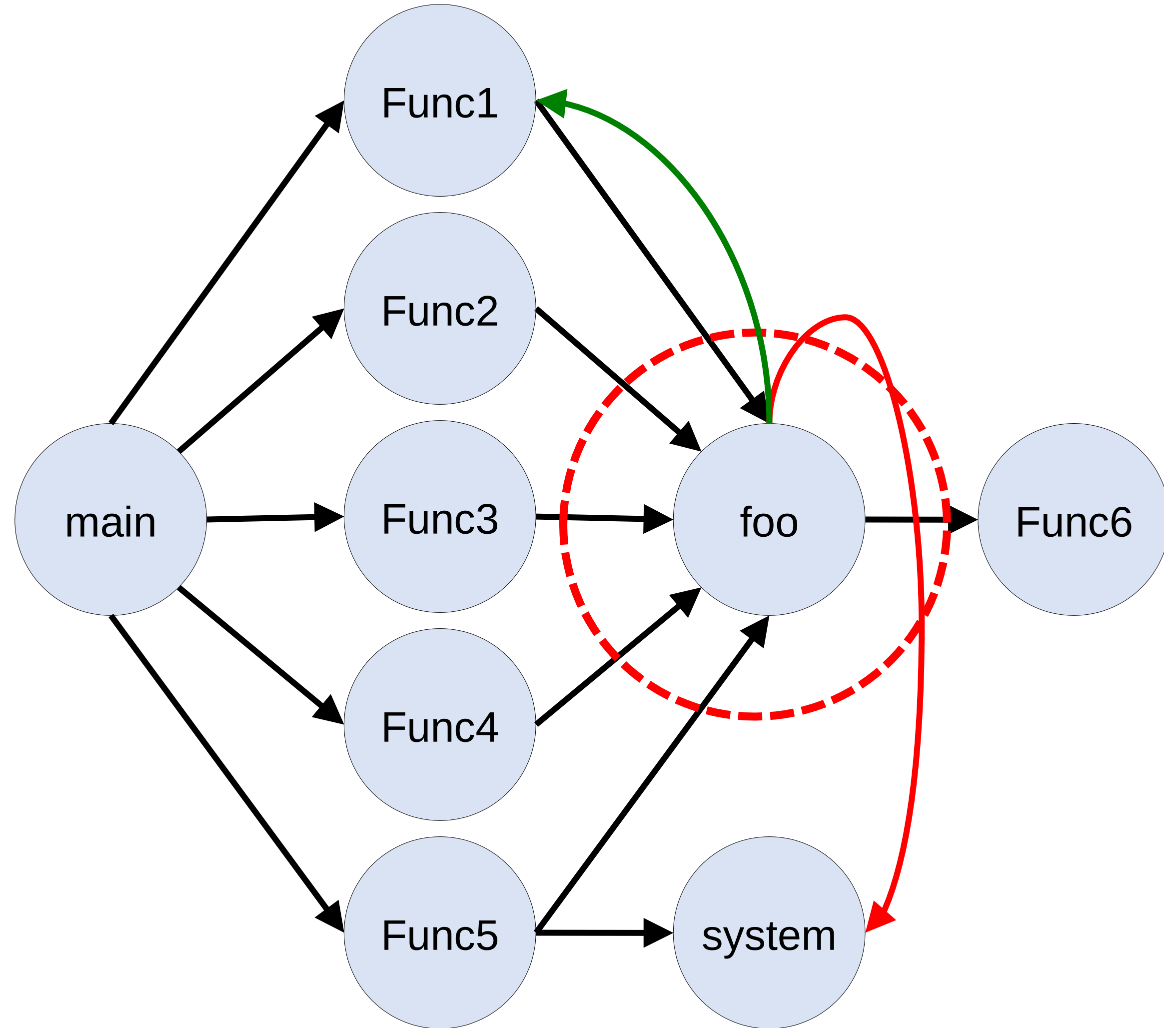
- **Great effort has been done by research community to protect the IoT.**
  - TyTan[DAC15], TrustLite[EurSys14], C-FLAT [CCS16], nesCheck[AsiaCCS17], SCFP[EuroS&P18], LiteHAX[ICCAD18], CFI CaRE [RAID17], ACES[SEC18], MINION [NDSS18], EPOXY [S&P17]

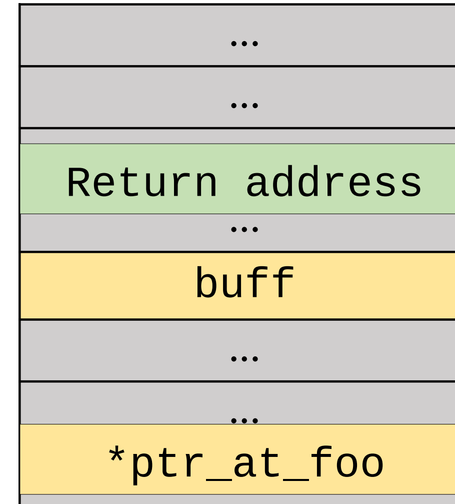
- **Unfortunately, current defenses suffer from one of the following:**

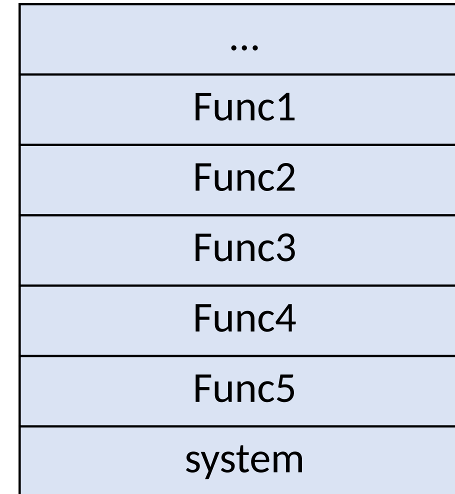
Limitation	Example of a Defense Mechanism
Information disclosure	Randomization
Only limit the attack surface	CFI (large target set)
Require extra hardware	Shadow stack
High overhead	Memory safety

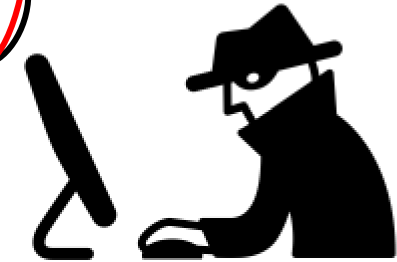
Normal Application

Sample call graph

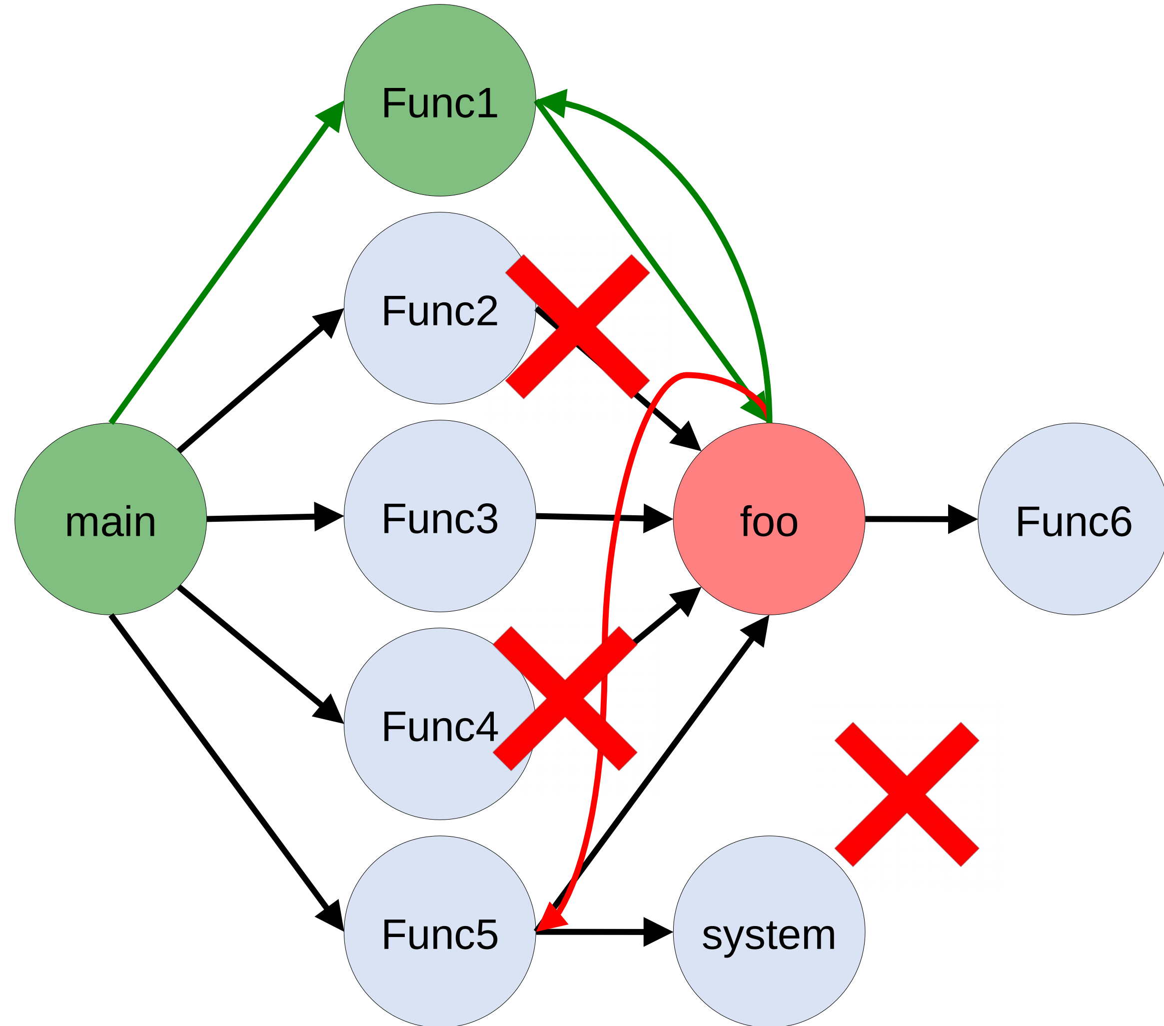


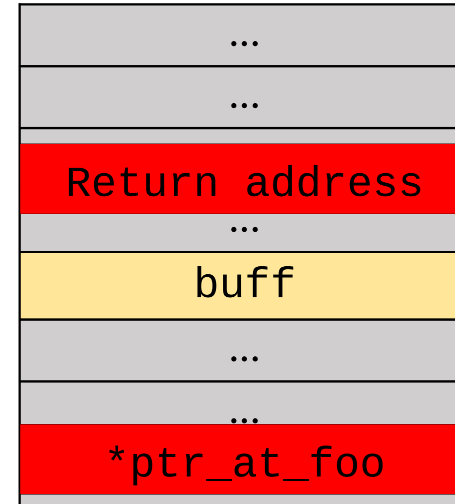


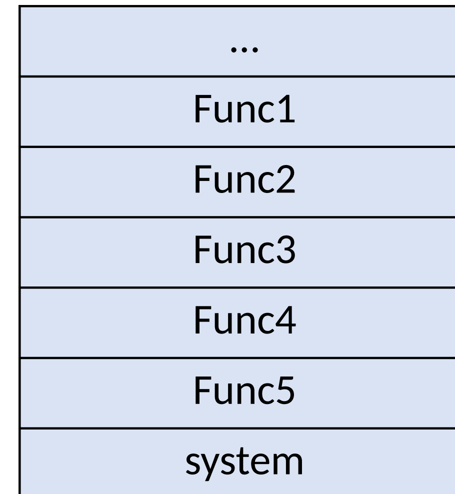




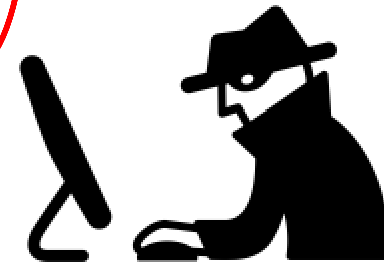
Usage Defenses: Control-Flow Integrity (CFI)





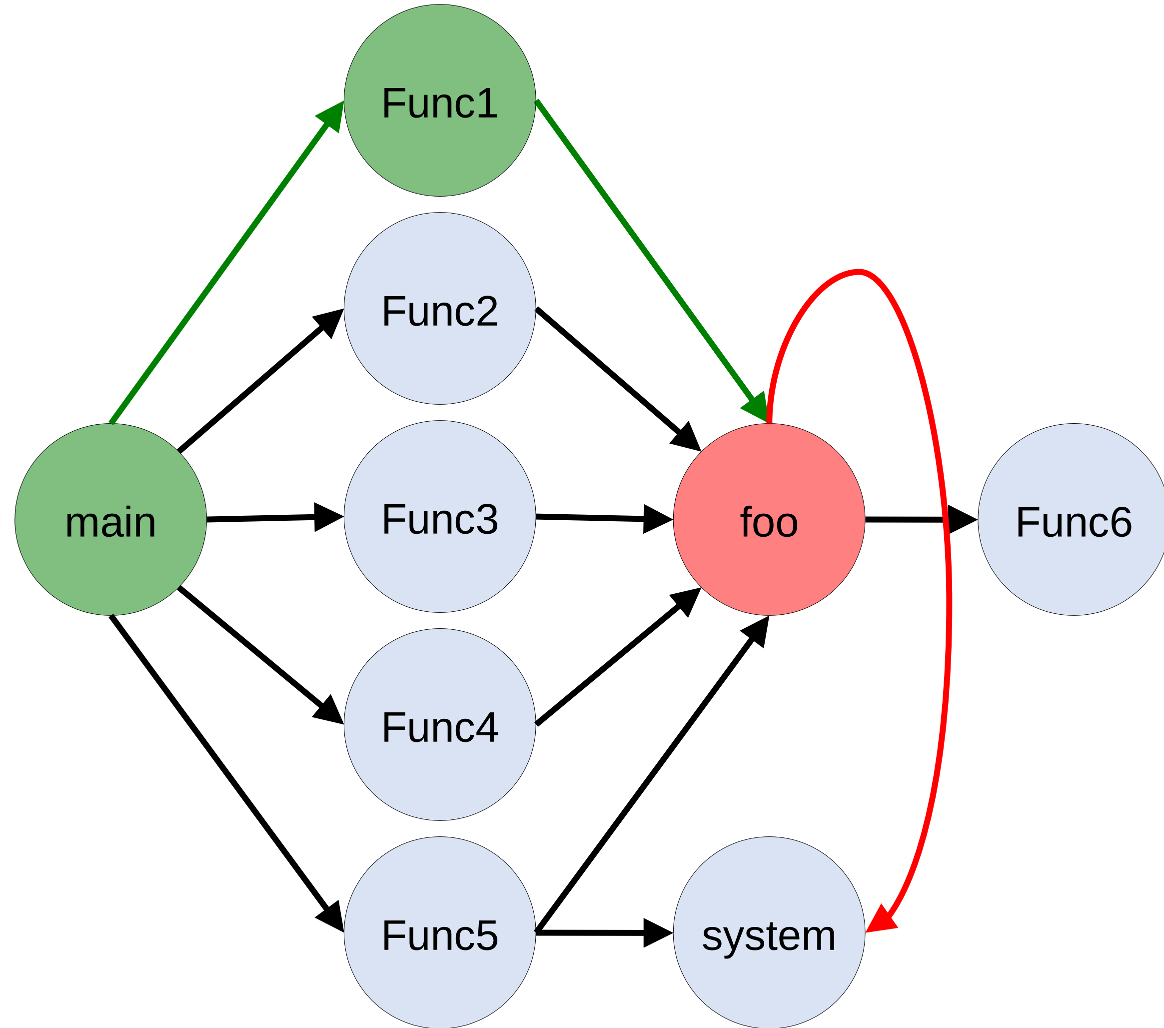


Defense:

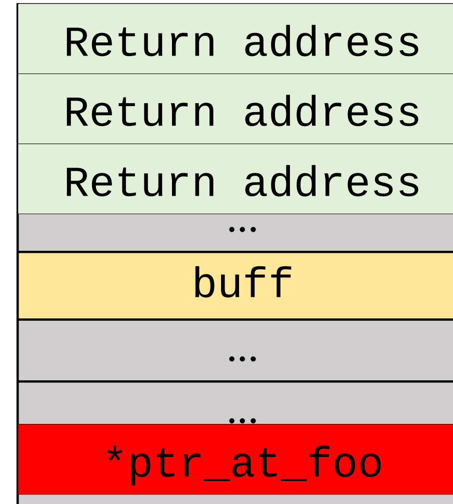


Location Defense: Randomization

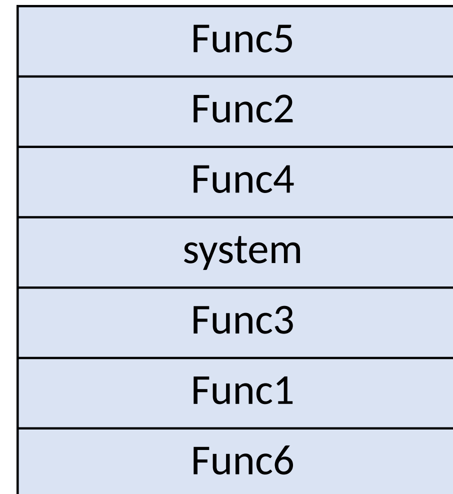
Sample call graph



Stack (RW Memory)



Code (RX Memory)



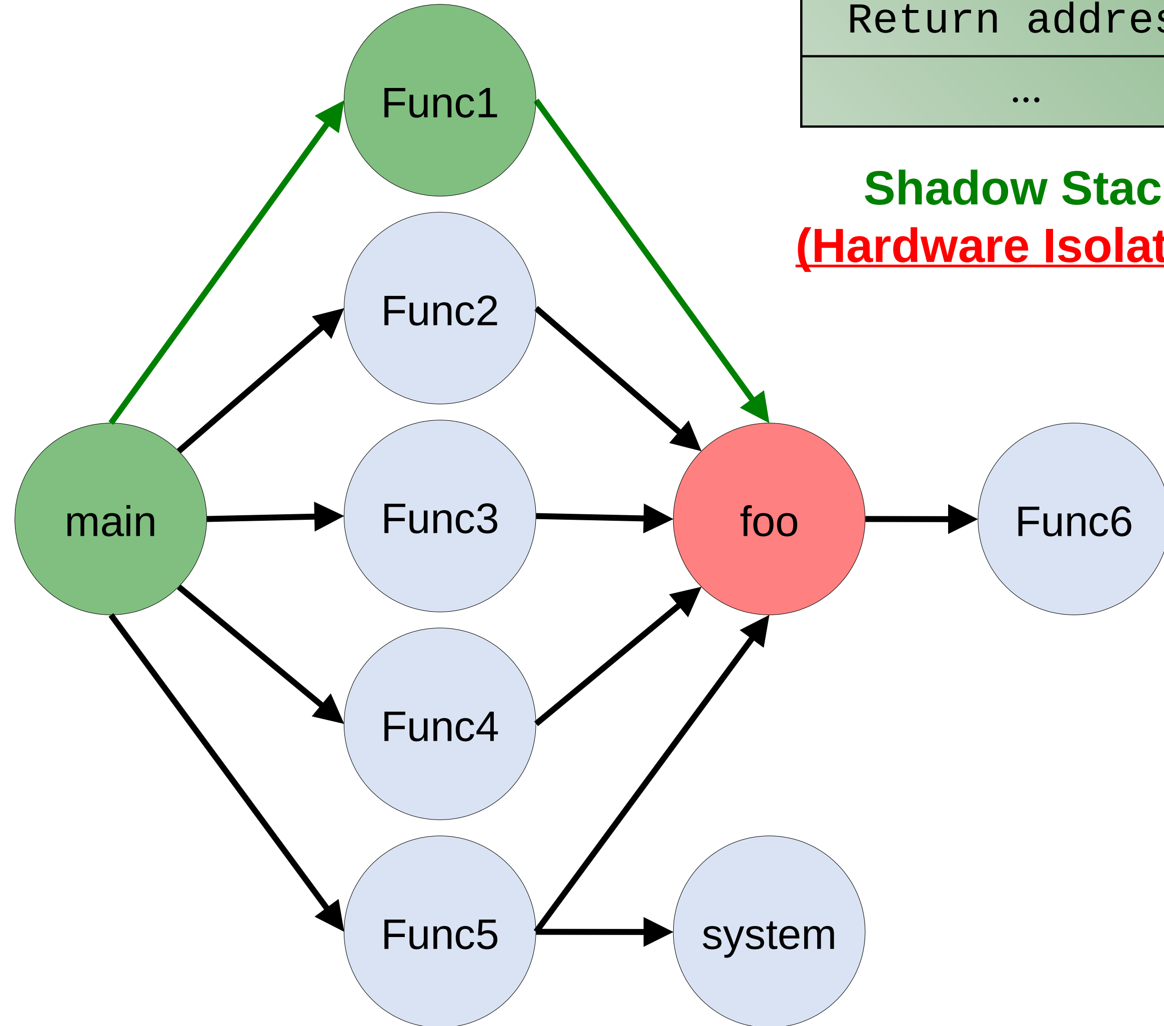
Defense:

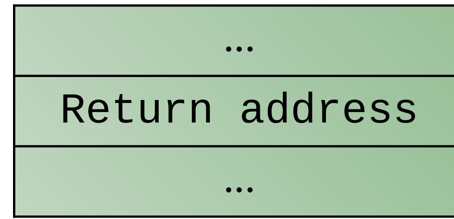
~~CFI~~

Disclose the layout

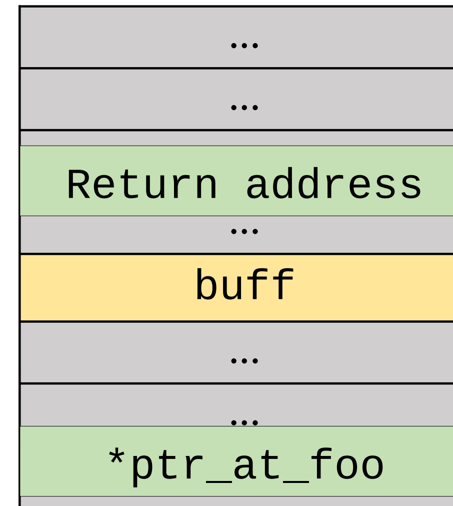
Integrity Defenses: Shadow Stack

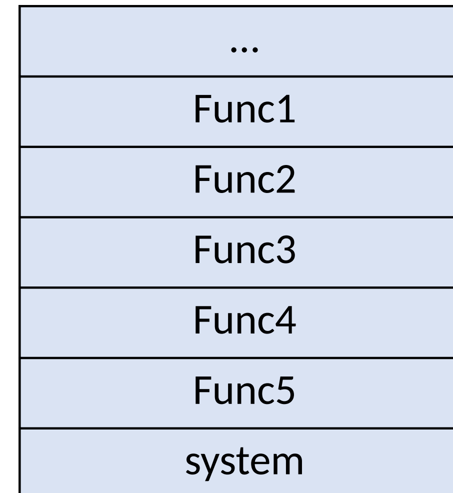
Sample call graph



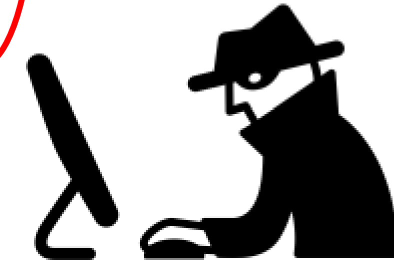


Shadow Stack
(Hardware Isolation)



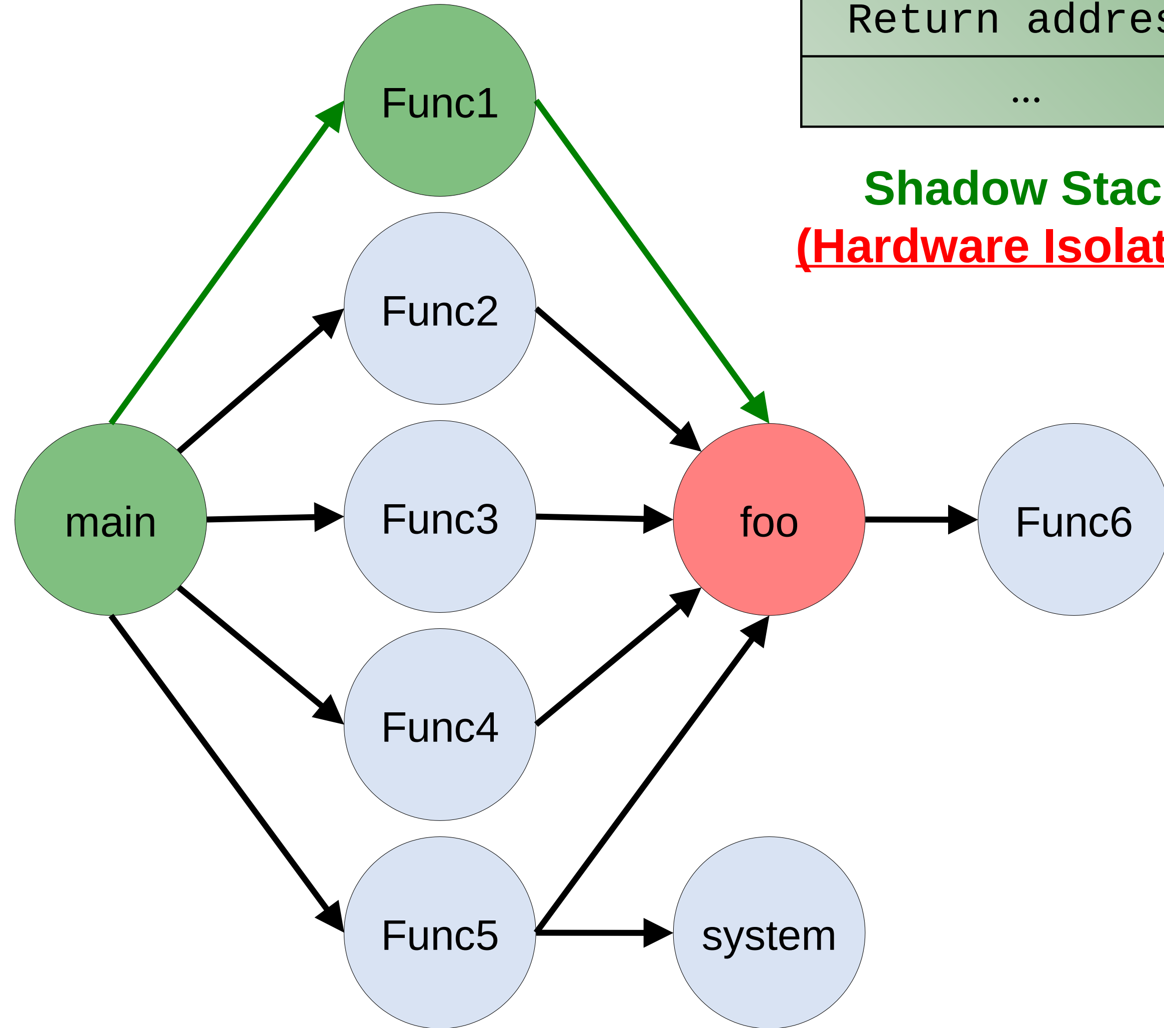


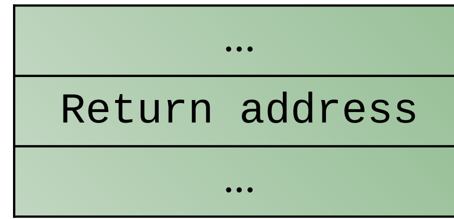
Defense:
~~CFI~~
~~Randomization~~



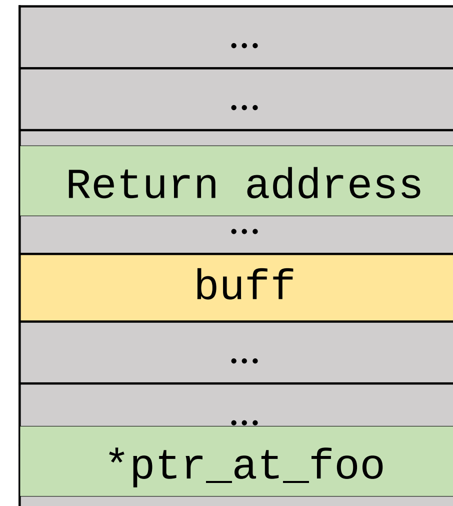
- System keeps 2 copies of return address
- Attacker cannot corrupt shadow stack
- Different return addresses
  - Attack detected

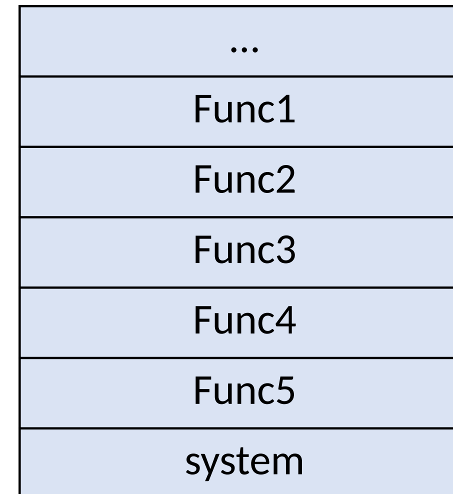
Integrity Defenses: Shadow Stack



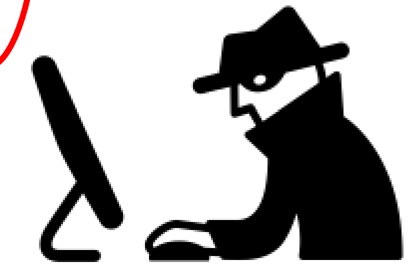


Shadow Stack
(Hardware Isolation)





Defense:
~~CFI~~
~~Randomization~~
~~Shadow stack~~



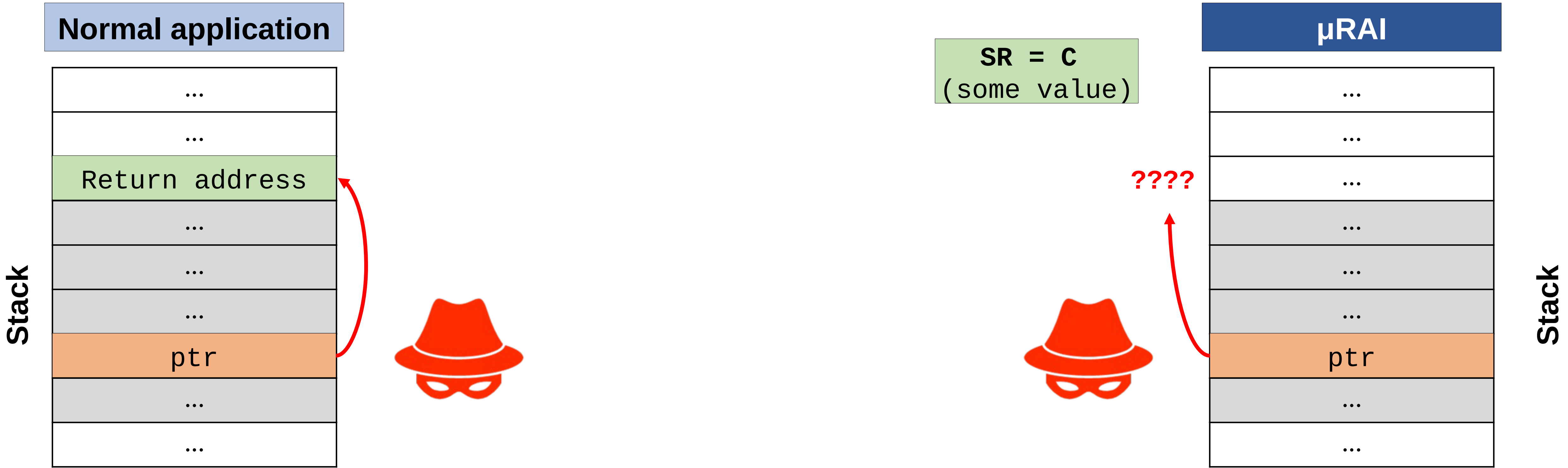
- TEE not available
- High overhead

State Register Layout

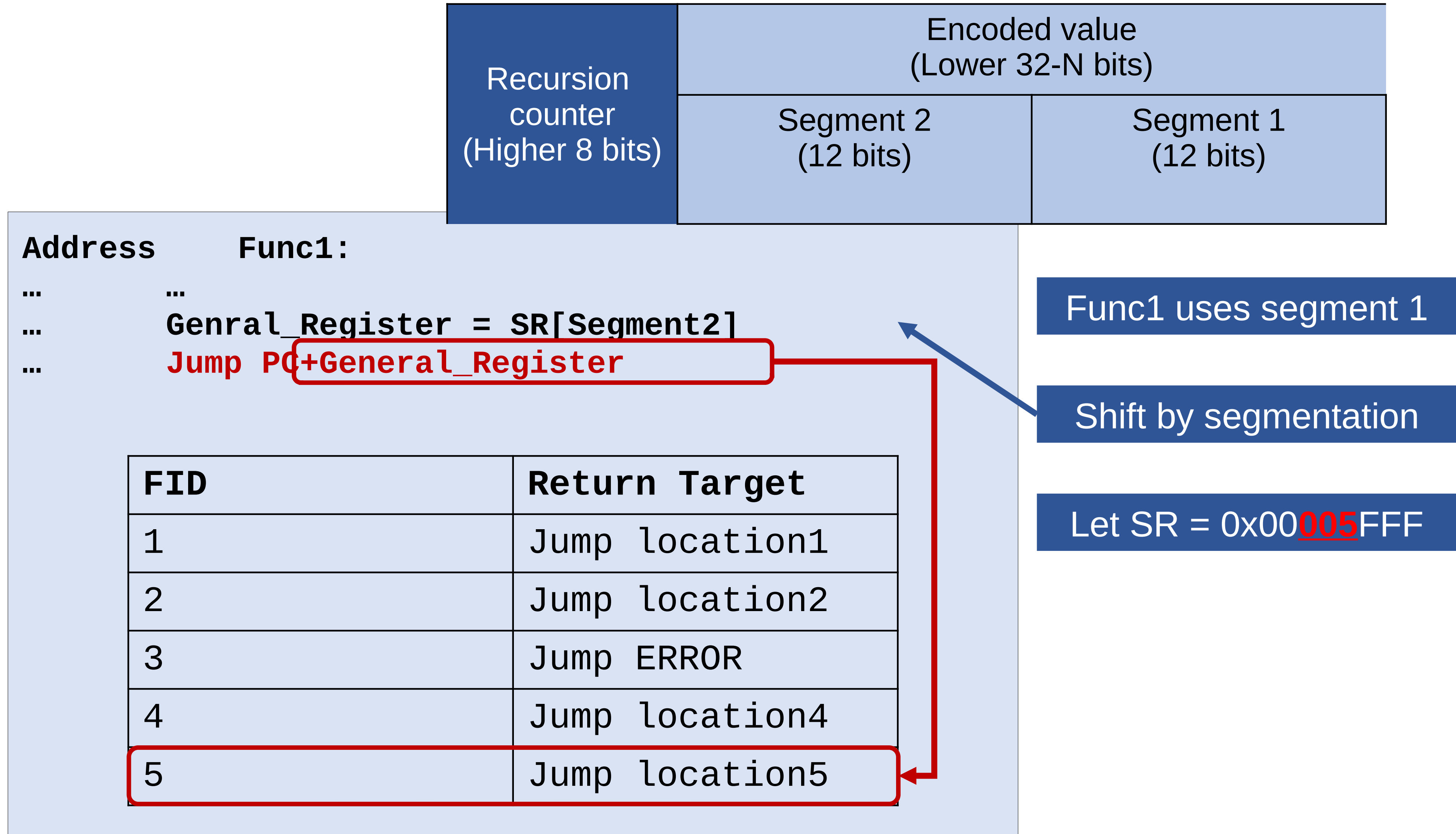
- **SR layout:**

µRAI Protection

- **Attacker:**
  - Has arbitrary write and read vulnerability
  - Knows the code layout, even the current instance of the firmware
  - Targets backward edges.
- **µRAI is complemented with DEP and type-based CFI for forward edge.**



TLR with Segmentation



PURDUE UNIVERSITY EPFL Sandia National Laboratories hexhive جامعة الملك سعود King Saud University

SR Encoding