RAP-Track: Efficient Control Flow Attestation via Parallel Tracking in Commodity MCUs

Antonio Joia Neto, University of Zurich
Adam Caulfield, University of Waterloo
Ivan De Oliveira Nunes, University of Zurich

62 THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA IEEE Council on Electronic Design Automation SIGDA acm

Introduction

- Low-cost, energy efficient MCUs

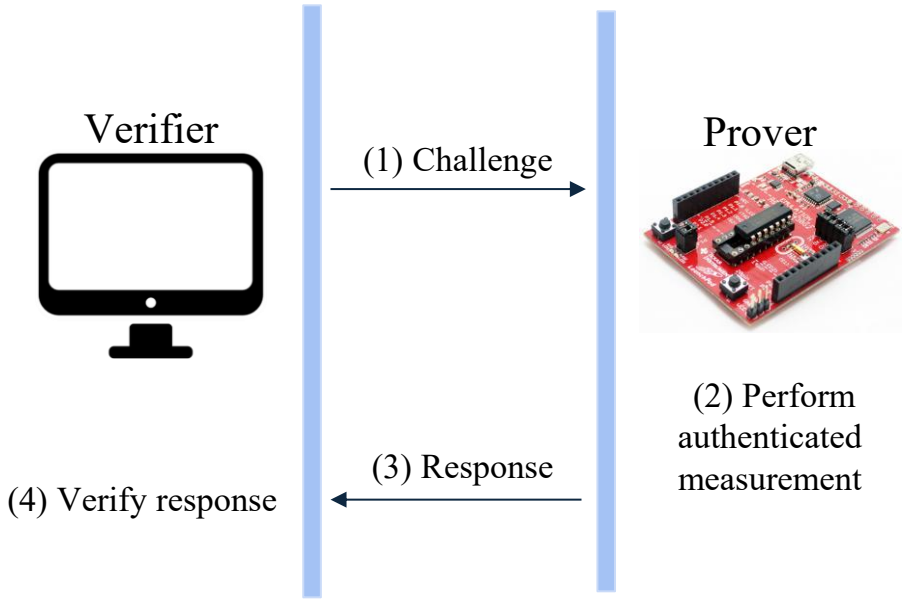
  - Limited processing power, memory size, memory protection, etc ..

- Resource constrained impact security features

- Execute safety-critical tasks in modern systems

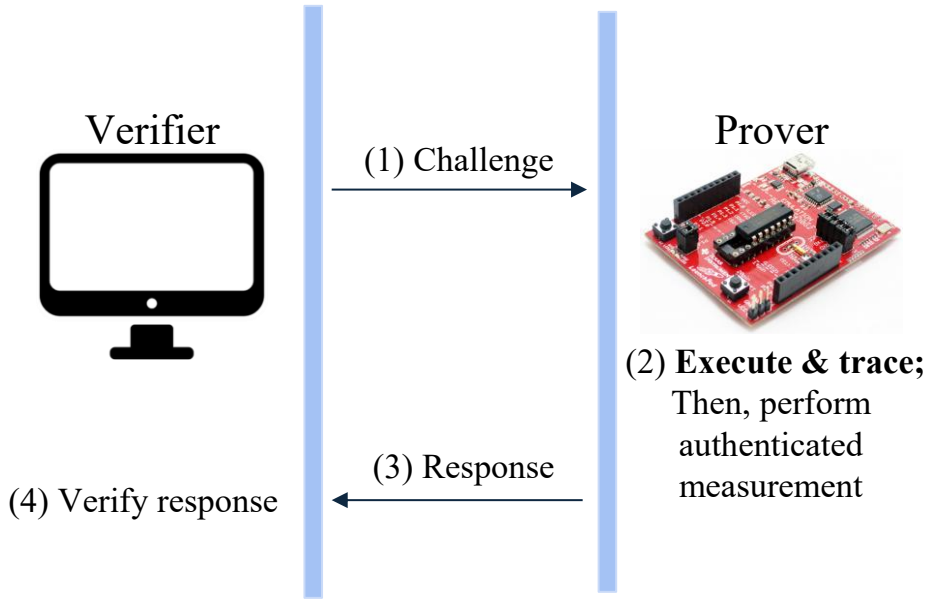
Remote Attestation

- A security mechanism designed to verify the integrity of remote device
- Ensures that the device's software state is untampered and operating as expected.
- Challenge and response protocol



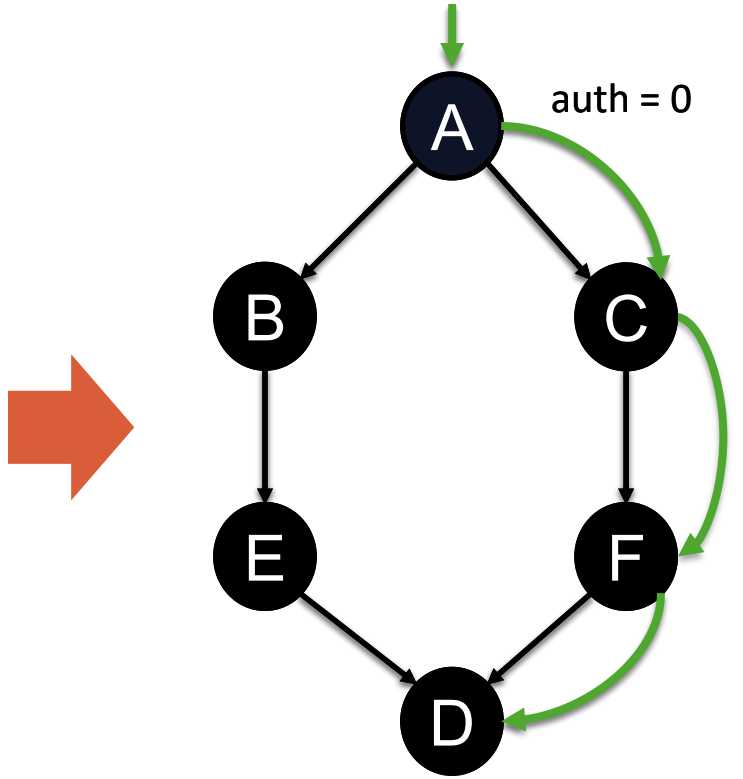
Control Flow Attestation (CFA)

- CFA extends RA to provide proof of runtime behavior of the application
- Prover records a trace of the control flow transfers (branches) during execution
- The trace and memory are measured to obtain proof about both static and runtime states



Control Flow Attestation (CFA)

```
A  if (auth)
B       do   function1();
C       else function2()
D  end
E  def function1(){…}
F  def function2(){…}
```



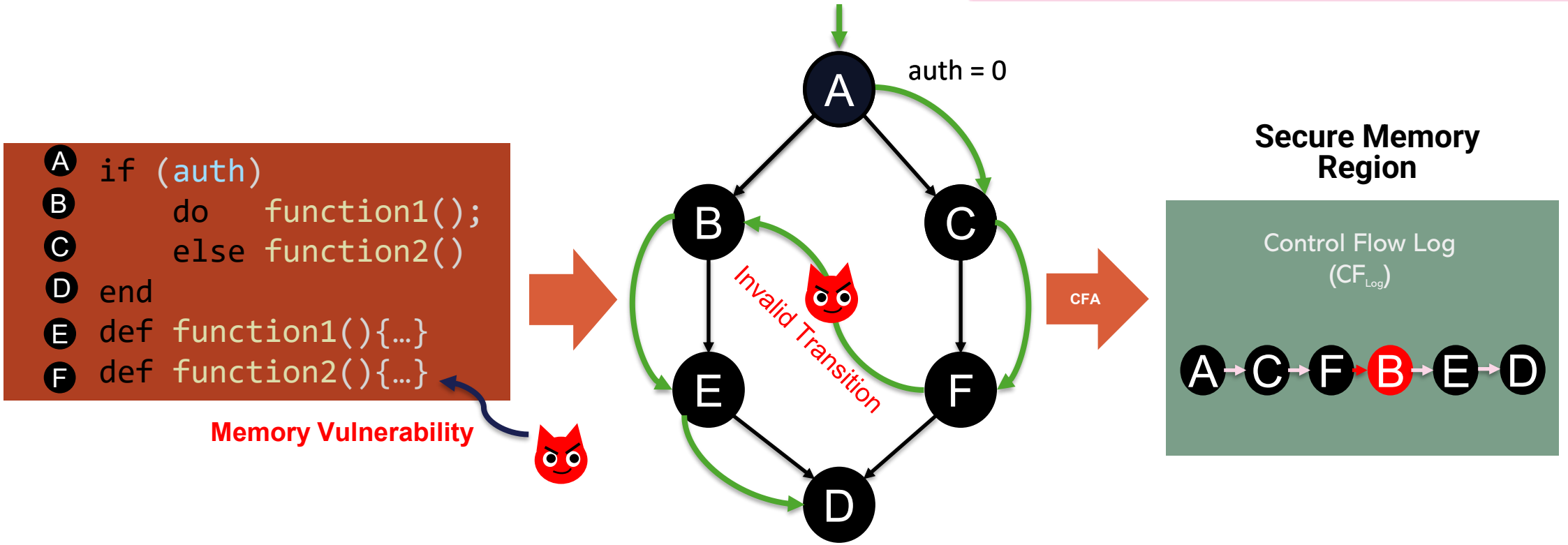
CFA

Secure Memory Region

Control Flow Log (CF_{Log})

A → C → F → D

Control Flow Attestation (CFA)



CFA Approaches

Instrumentation + Trusted Software

- Use binary instrumentation to prepend/replace branch instructions in the attested app with *trampoline* instructions
- These instructions *trampoline* into a trusted software region
- The trusted software logs the branch destination then returns
- **Problems:**
  - Significant overheads to application run-time
  - Significant increase in application binary size

CFA Approaches

Tracing Hardware (custom or commodity)

- Dedicated hardware module that detects and logs control flow transfers
- Can operate in parallel to the attested code
- No run-time overhead
- **Problems:**
  - Custom hardware: not immediately deployable
  - Commodity (relatively new!) :
    - General purpose
    - Records redundant data and overflows quickly

Micro Trace Buffer (MTB)

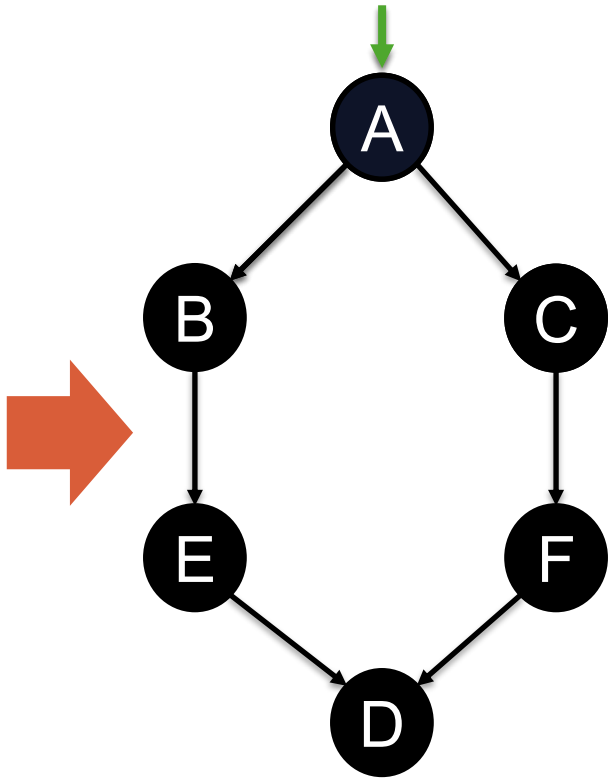
- A lightweight on-chip control flow tracing (available in ARM MCUs)

Key Features:

- Circular buffer in RAM for storing memory address values
- When activated**, records all changes in program flow** where the Program Counter does not increment sequentially.
  - **Function calls**, **branches**, and **interrupts**.
- Records the source and destination of instructions that break sequential execution

Micro Trace Buffer (MTB)

```
0x01  A  if (auth)
0x02  B      do function1();
0x03  C      else function2()
0x04  D  end
0x05  E  def function1(){…}
0x06  F  def function2(){…}
```

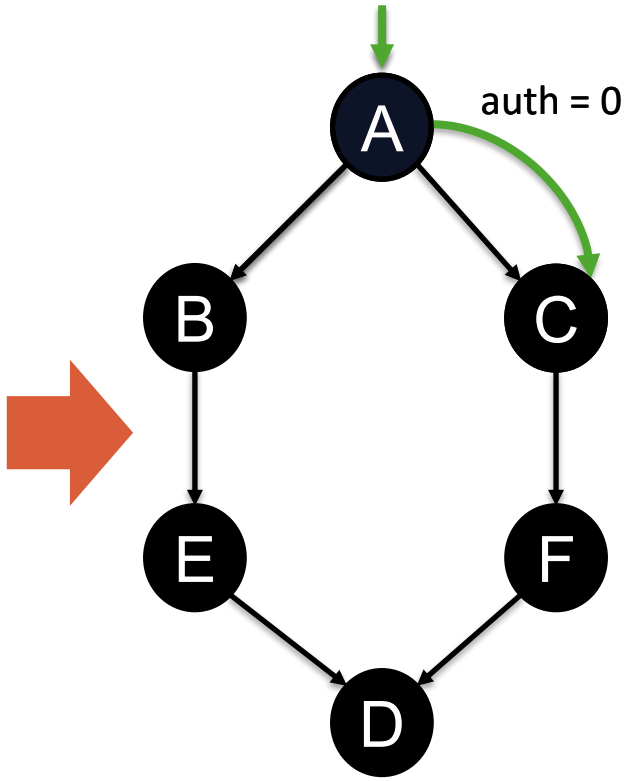


MTB

Source	Dest.

Micro Trace Buffer (MTB)

```
0x01  A  if (auth)
0x02  B      do function1();
0x03  C      else function2()
0x04  D  end
0x05  E  def function1(){…}
0x06  F  def function2(){…}
```

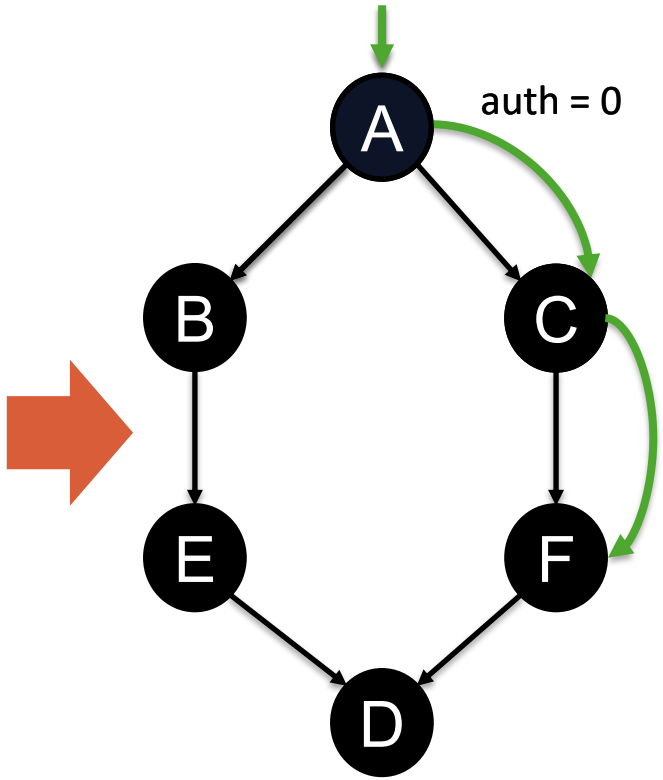


MTB

Source	Dest.
0x01	0x03

Micro Trace Buffer (MTB)

```
0x01  A  if (auth)
0x02  B      do function1();
0x03  C      else function2()
0x04  D  end
0x05  E  def function1(){…}
0x06  F  def function2(){…}
```

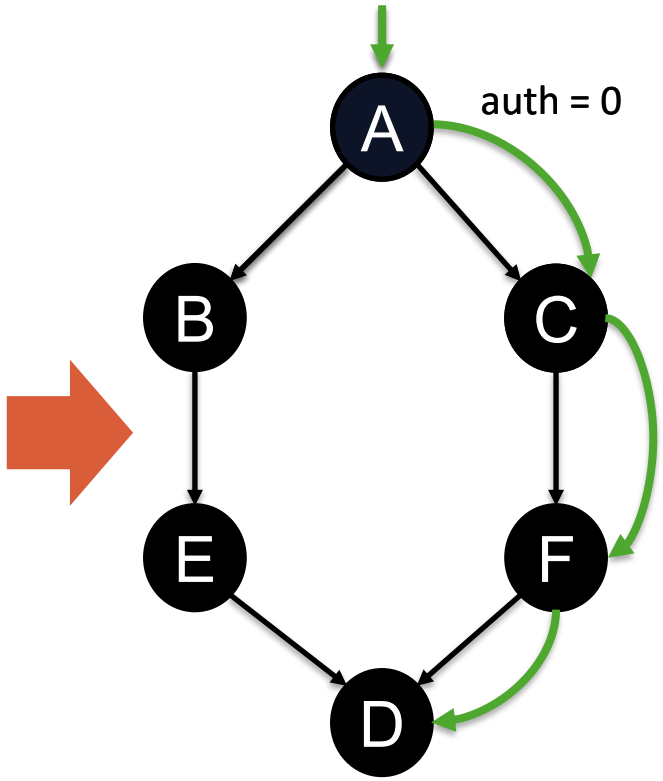


MTB

Source	Dest.
0x01	0x03
0x03	0x06

Micro Trace Buffer (MTB)

```
0x01  A  if (auth)
0x02  B      do function1();
0x03  C      else function2()
0x04  D  end
0x05  E  def function1(){…}
0x06  F  def function2(){…}
```

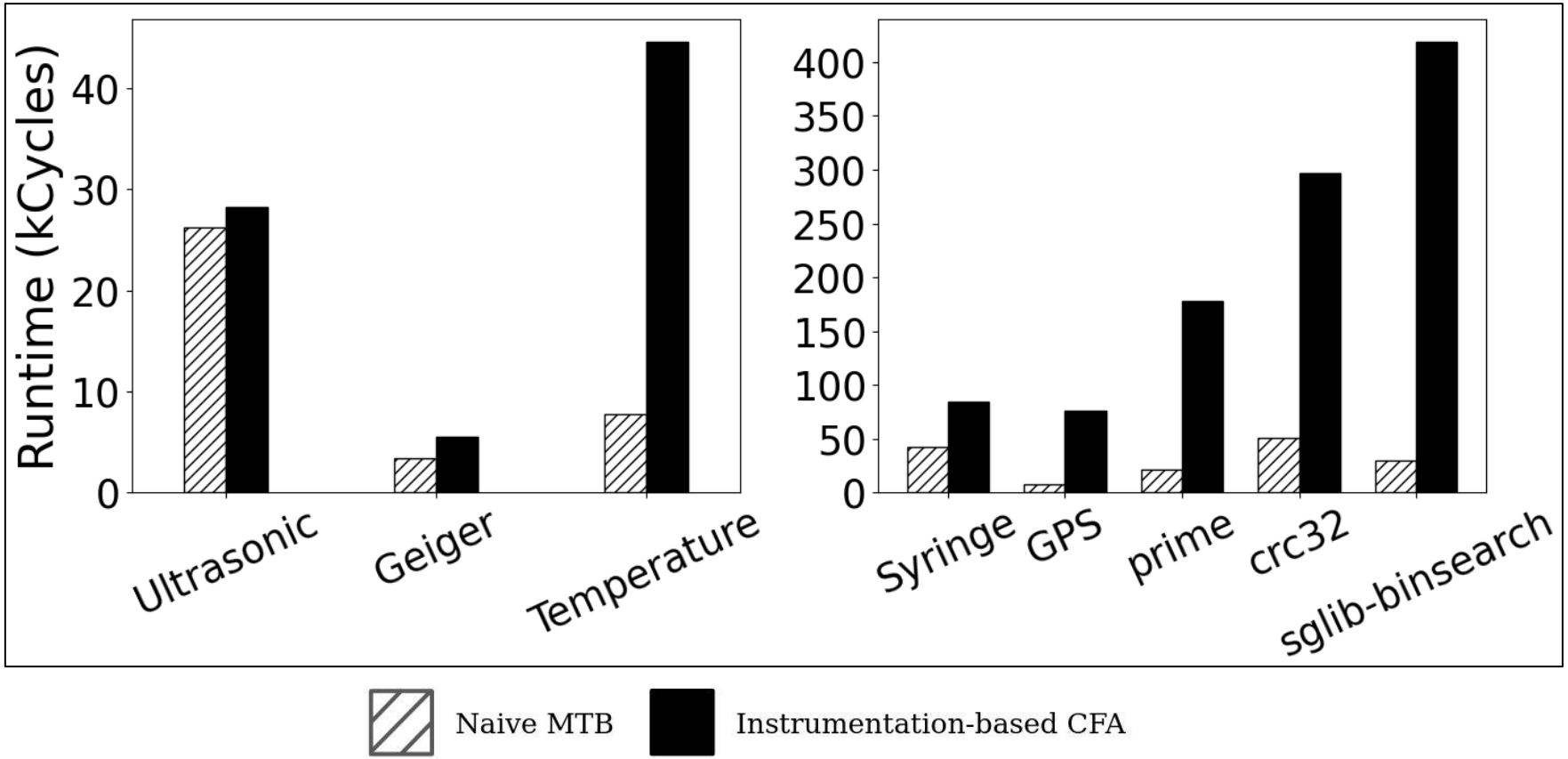


MTB

Source	Dest.
0x01	0x03
0x03	0x06
0x06	0x04

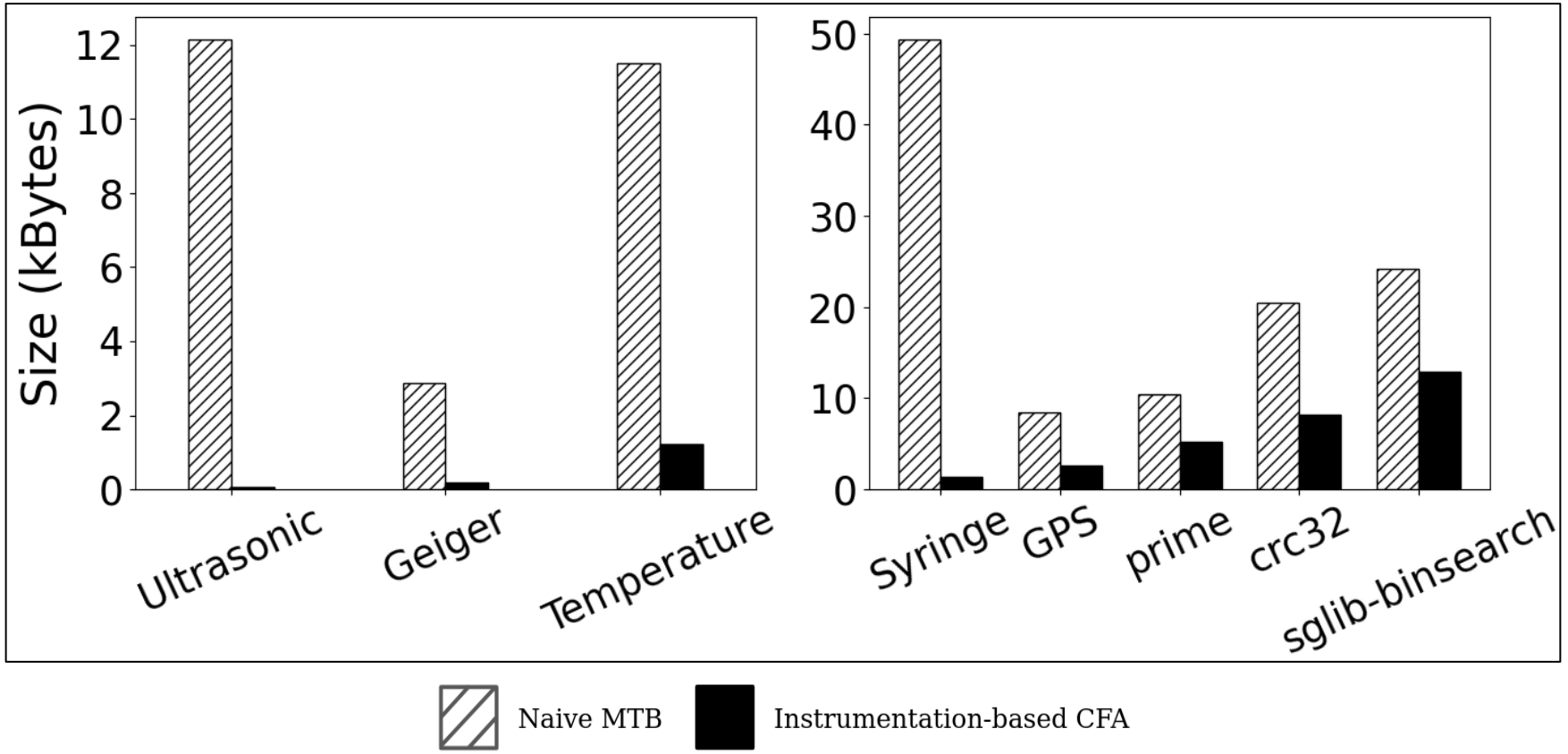
Problems of CFA Quantified

Instrumentation-based approaches incur significant run-time overheads (1.9-217x larger)



Problems of CFA Quantified

CF_{Log} cannot be optimized as easily in naïve MTB use, risking overflows or interrupts for transmission



RAP-Track

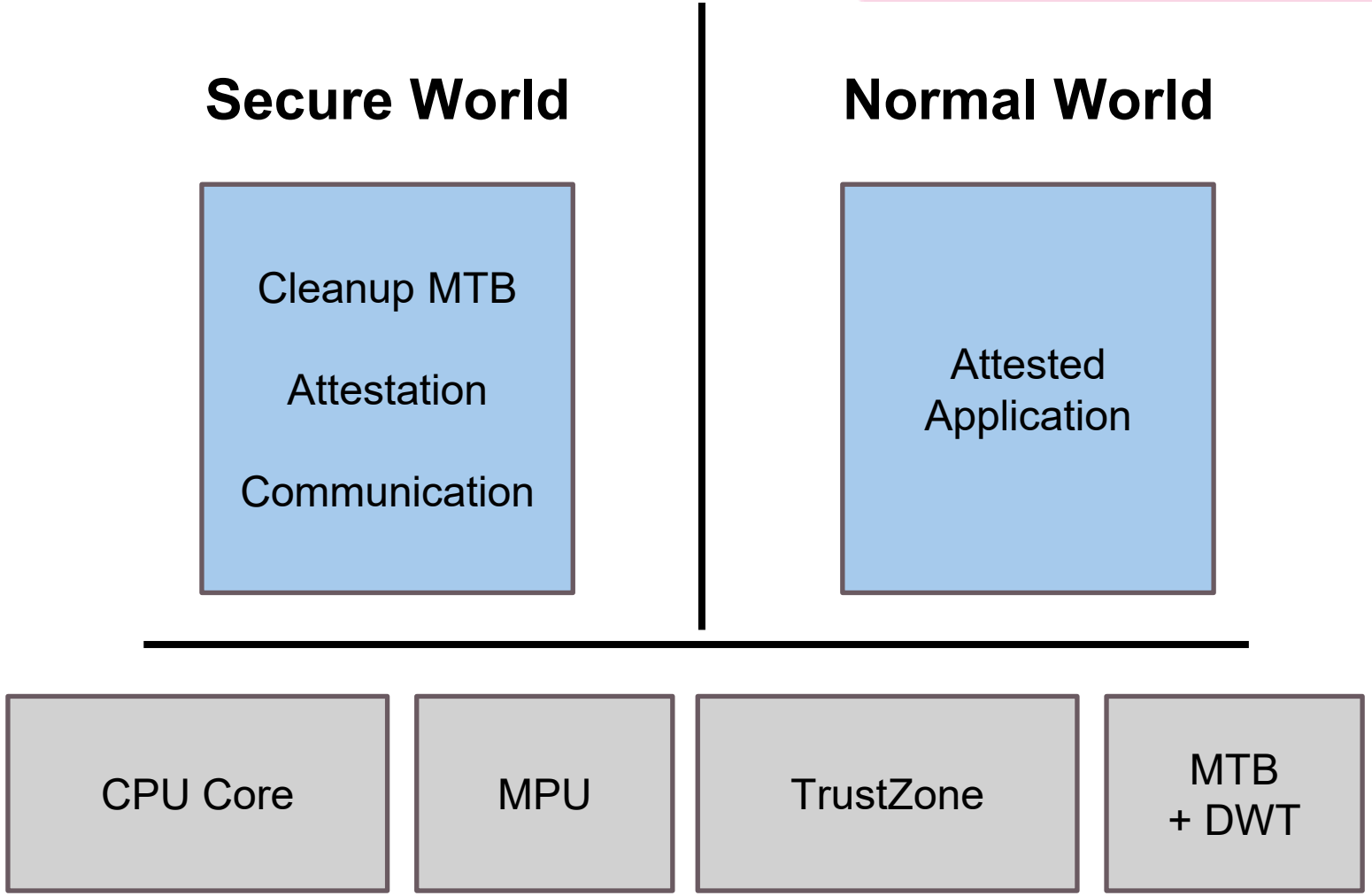
Key Idea:

- RAP-Track customizes the use of ARM's MTB and Data Watchpoint and Trace (DWT) for low-overhead control flow attestation (CFA).
- Strategic linking and static analysis to so MTB only records non-deterministic branches

Efficiency Gains:

- Fewer Secure World calls via buffered trace updates (MTB)
- Excludes deterministic/redundant branches from logs
- Reduces runtime and storage overhead

Depiction

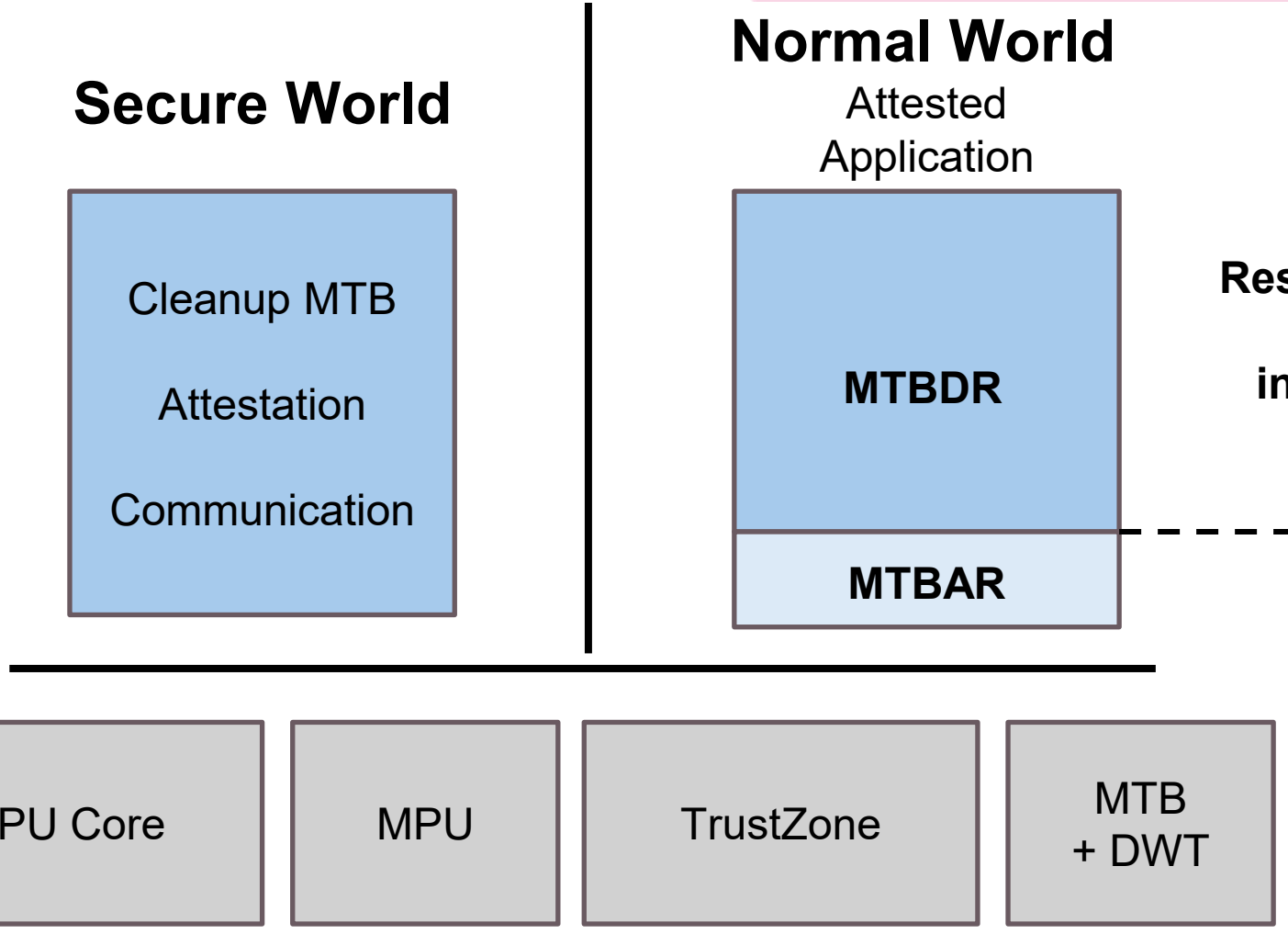

Secure World
Normal World
Cleanup MTB
Attestation
Communication
Attested Application
CPU Core
MPU
TrustZone
MTB + DWT

Depiction

Split application into two fixed regions:

MTB Activation Region (MTBAR)

MTB Deactivation Region (MTBDR)



Restructure binary so only relevant instructions are in MTBAR

MTBAR and MTBDR

MTBAR – *MTB Activation Region*

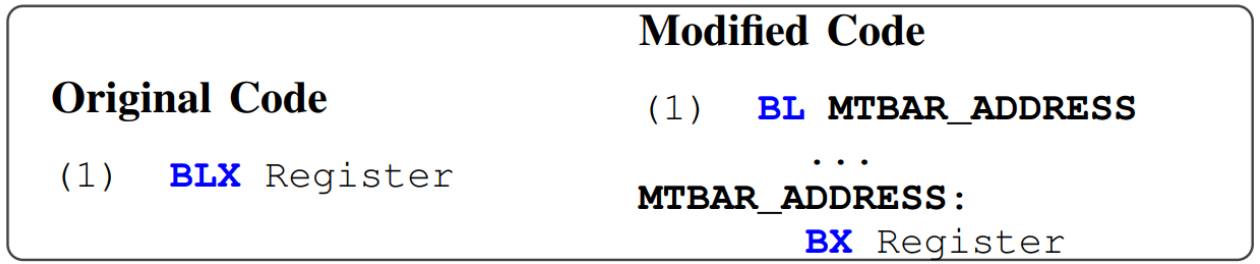
•Contains **non-deterministic branches**:
- Indirect calls
- Returns
- Conditional branches with variable outcomes

MTBDR – *MTB Deactivation Region*

•Contains **deterministic code paths**:
- Fixed branches
- Simple, statically analyzable loops

Trampoline Logic

- Rewrites indirect and conditional branches using trampolines
- Trampolines redirect to monitored regions (MTBAR)
- Ensures only essential control-flow transitions are logged



Example: Indirect call

Trampoline Logic

Original Code	Modified Code

`POP {PC}`	`B MTBAR_POP_ADDR`
	`...`
	`MTBAR_POP_ADDR:`
	`  POP {PC}`

Original Code	Modified Code

`LDR PC, [sp, #4]`	`B MTBAR_LDR_ADDR`
	`...`
	`MTBAR_LDR_ADDR:`
	`  LDR PC, [sp, #4]`

Returns and Indirect Jumps

Trampoline Logic

Original Code

```
     CMP R0,#0
(1)  BEQ taken_address
```

Modified Code

```
     CMP R0,#0
(1)  BEQ MTBAR_ADDRESS
       ...
MTBAR_ADDRESS:
     B taken_address
```

Non-loop conditional branch

Trampoline Logic

Original Code

```
loop_start:
      ...
      CMP R0,#0
(1)   BEQ loop_start
```

Modified Code

```
loop_start:
      ...
      CMP R0,#0
(1)   BEQ MTBAR_ADDRESS
      ...
MTBAR_ADDRESS:
      B loop_start
```

“Backward” loop conditional branch

Original Code

```
      CMP R0,#0
      BEQ loop_end
      ADD R1, R2, R3
      ...
loop_end:
      ...
```

Modified Code

```
      CMP R0,#0
      BEQ loop_end
(1)-> B MTBAR_ADDRESS
br_not_taken_addr:
      ADD R1, R2, R3
      ...
loop_end:
      ...
MTBAR_ADDRESS:
      B br_not_taken_addr
```

“Forward” loop conditional branch

+Plus, additional instrumentation for simple-loop optimizations

Evaluation

Setup:

- Implemented on V2M-MPS2-0318C FPGA board
- Based on ARM Cortex-M33 (v8) with TrustZone, MTB, and DWT

Code Footprint:

- Secure World size: 11 KB total
  - CFA Engine: 2.8 KB

Static analysis:

- Combination of python and bash scripts
- Operates directly on executables
- Uses ELFtools, Keystone

Evaluation

Compare RAP-Track to two CFA implementations:

Naïve MTB – using the MTB without any consideration for redundancy

TRACES – instrumentation-based CFA in ARM Cortex-M MCUs

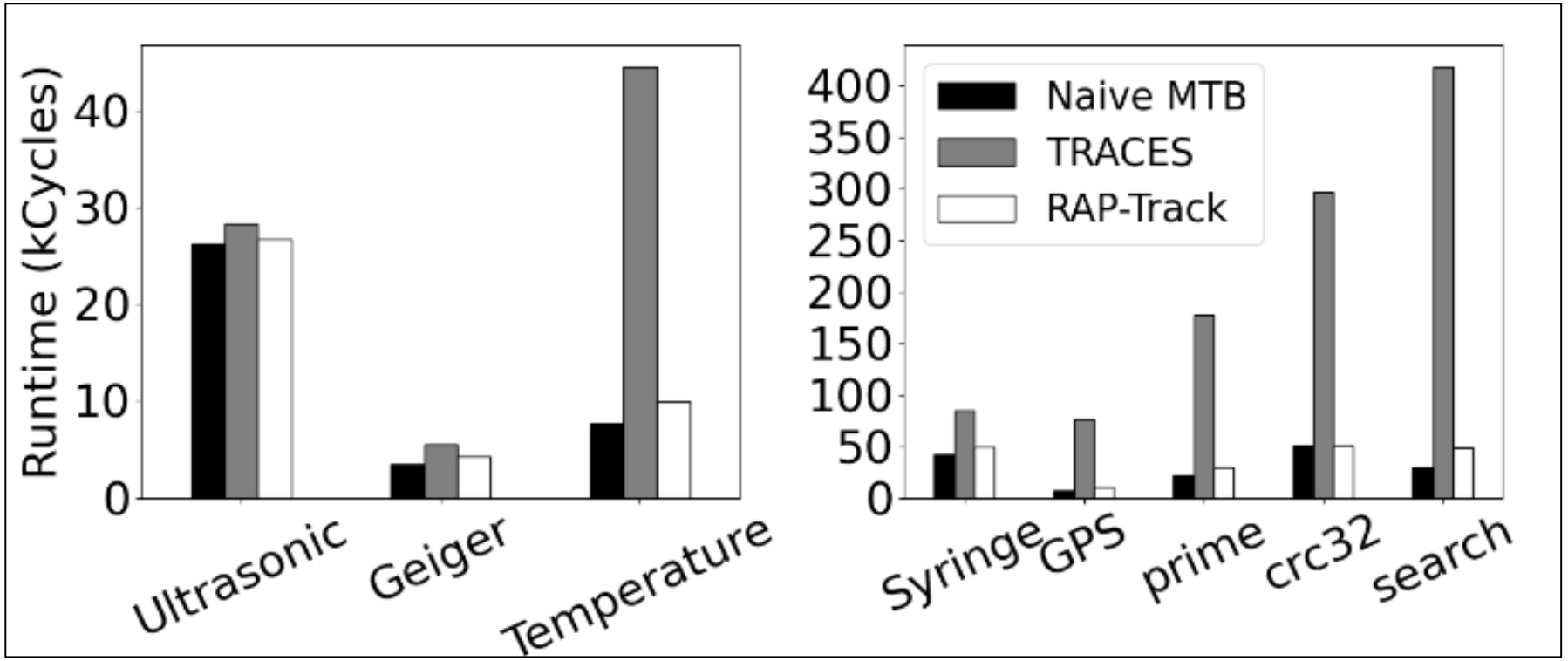
Evaluate by measuring:

- Application runtime
- CFLog size
- Code size

Evaluation

Runtime Analysis:

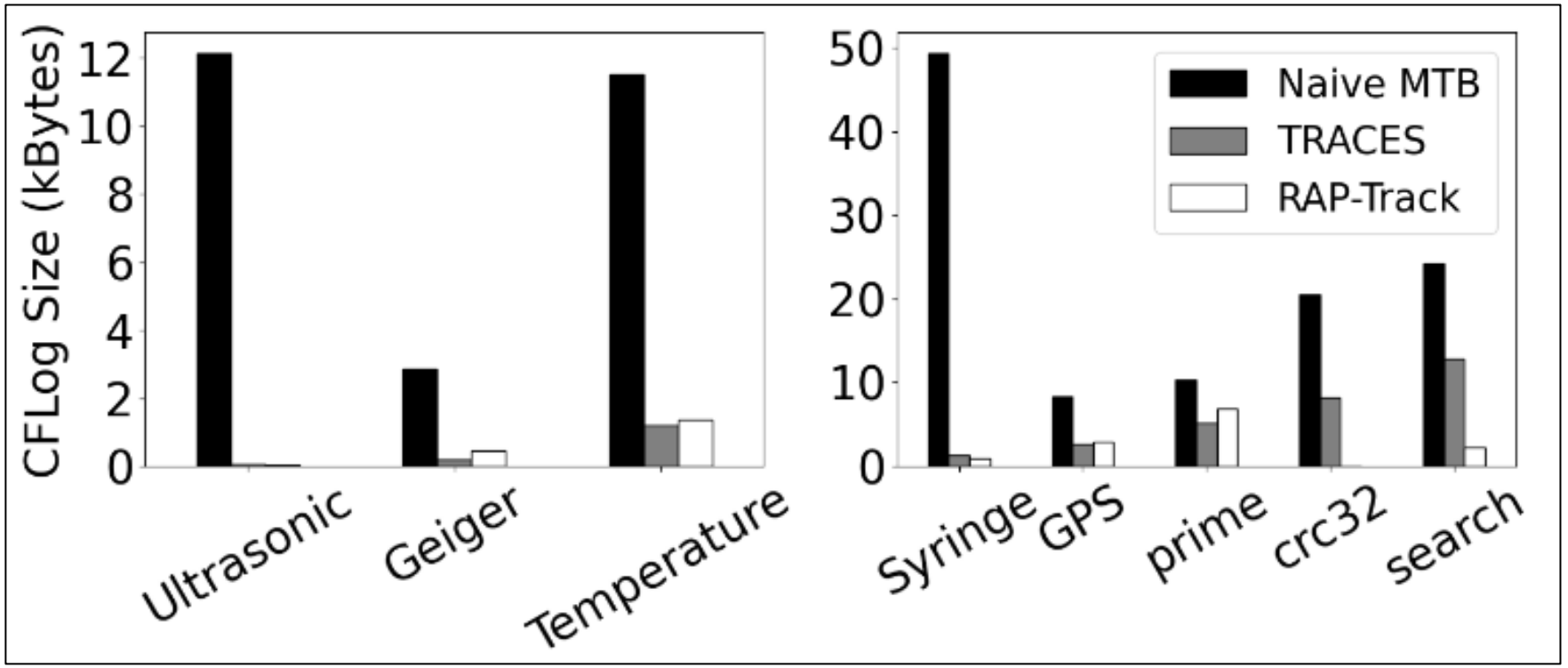
- RAP-Track has similar overhead as naïve MTB
- Compared to TRACES it reduces runtime overhead by **5.26%-88.4%**



Evaluation

CF_{Log} size:

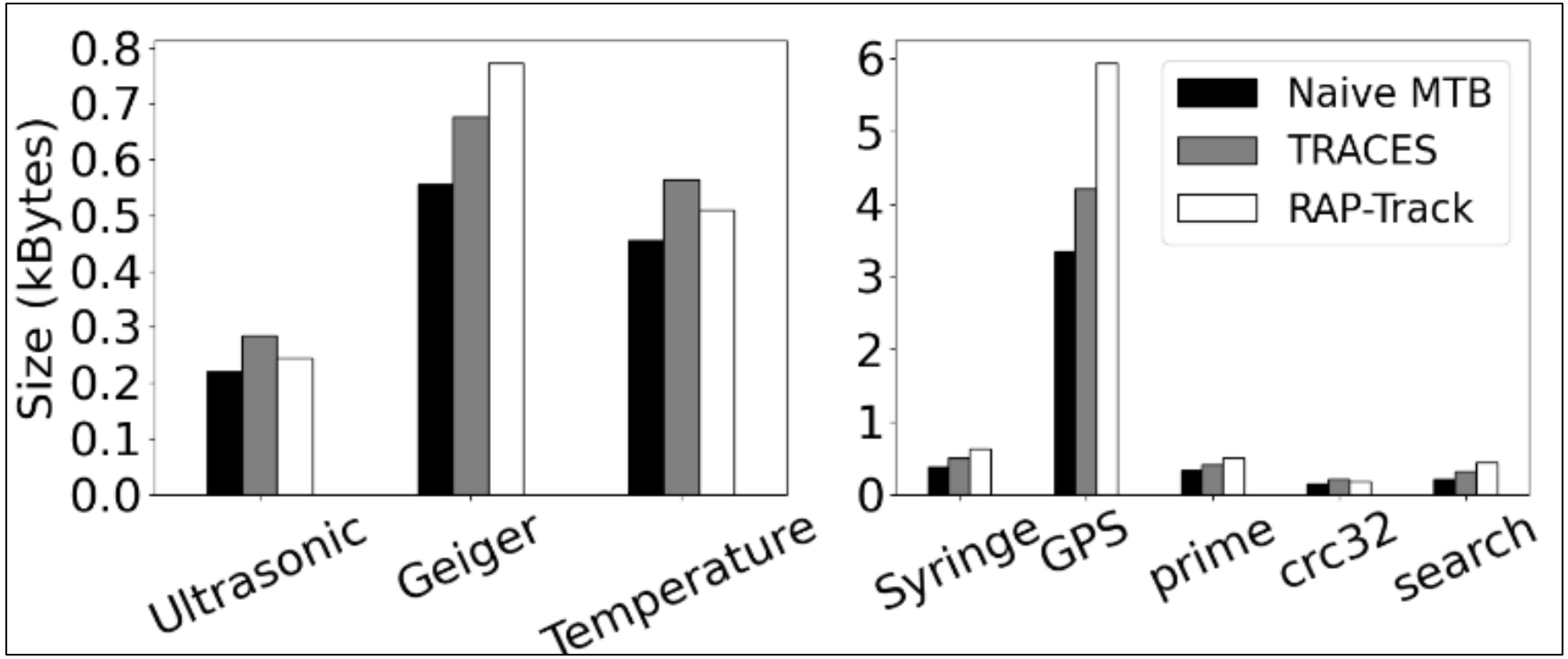
- RAP-Track reduces total recorded data from naïve MTB by **33.3-99.9%**
- Comparable results to TRACES (with software optimizations)



Evaluation

Code Size:

- RAP-Track always increased code size compared naïve MTB (**10.9-108%**)
- Code size compared to TRACES depends on the application
  - Could increase (**up to 41.5%**) or decrease (**up to 14.1%**)
- The savings for other factors (**up to 99.9% for CF_{Log}**, **up to 88.4% for runtime**) are more impactful



Conclusion

- RAP-Track enables efficient, lossless CFA for embedded systems
- Leverages existing hardware – no need for custom chips
- Provides reasonable tradeoff between performance and memory usage