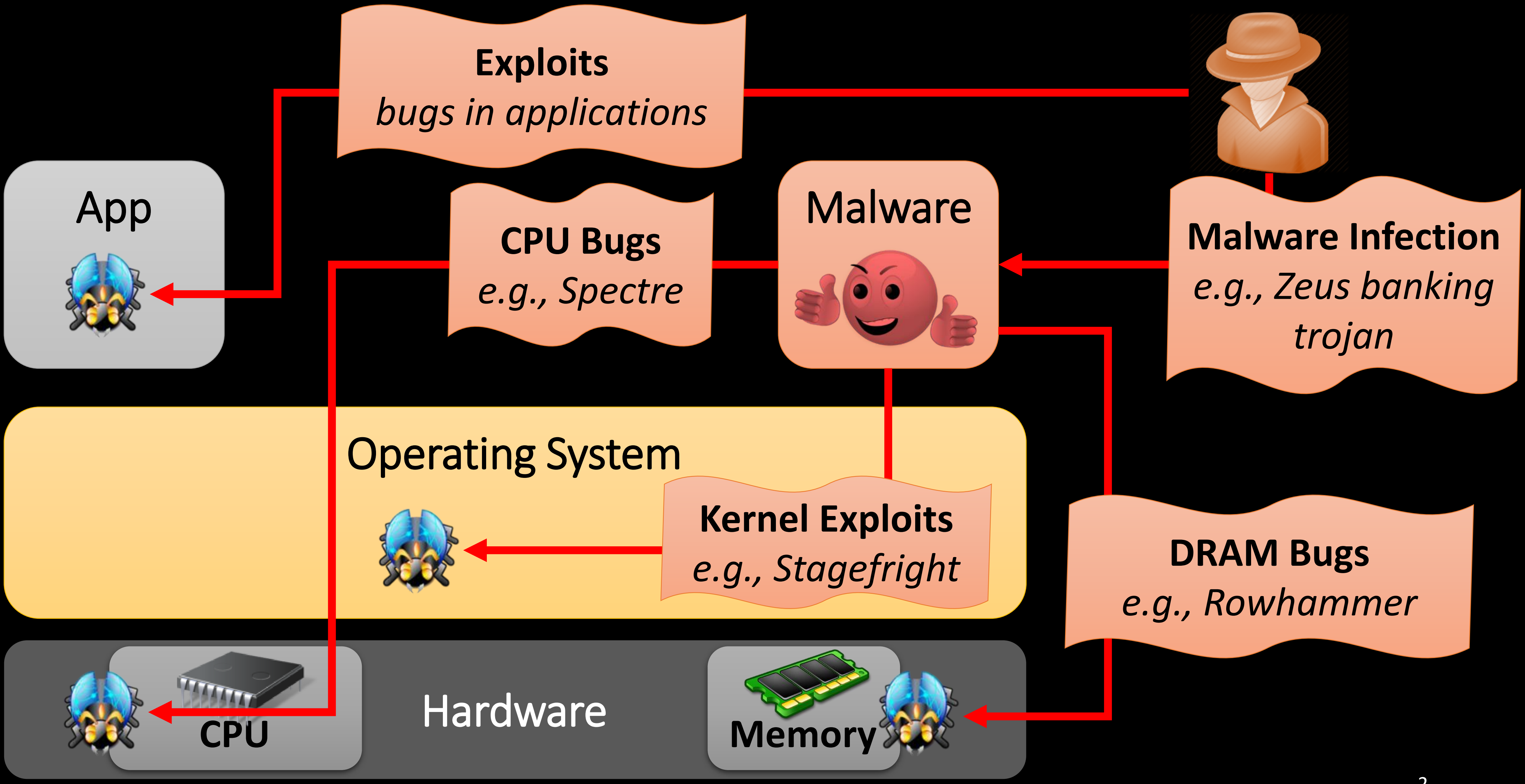
Seminar on the Security of Software and Hardware Interfaces, Rennes, INRIA, France
8 November, 2019

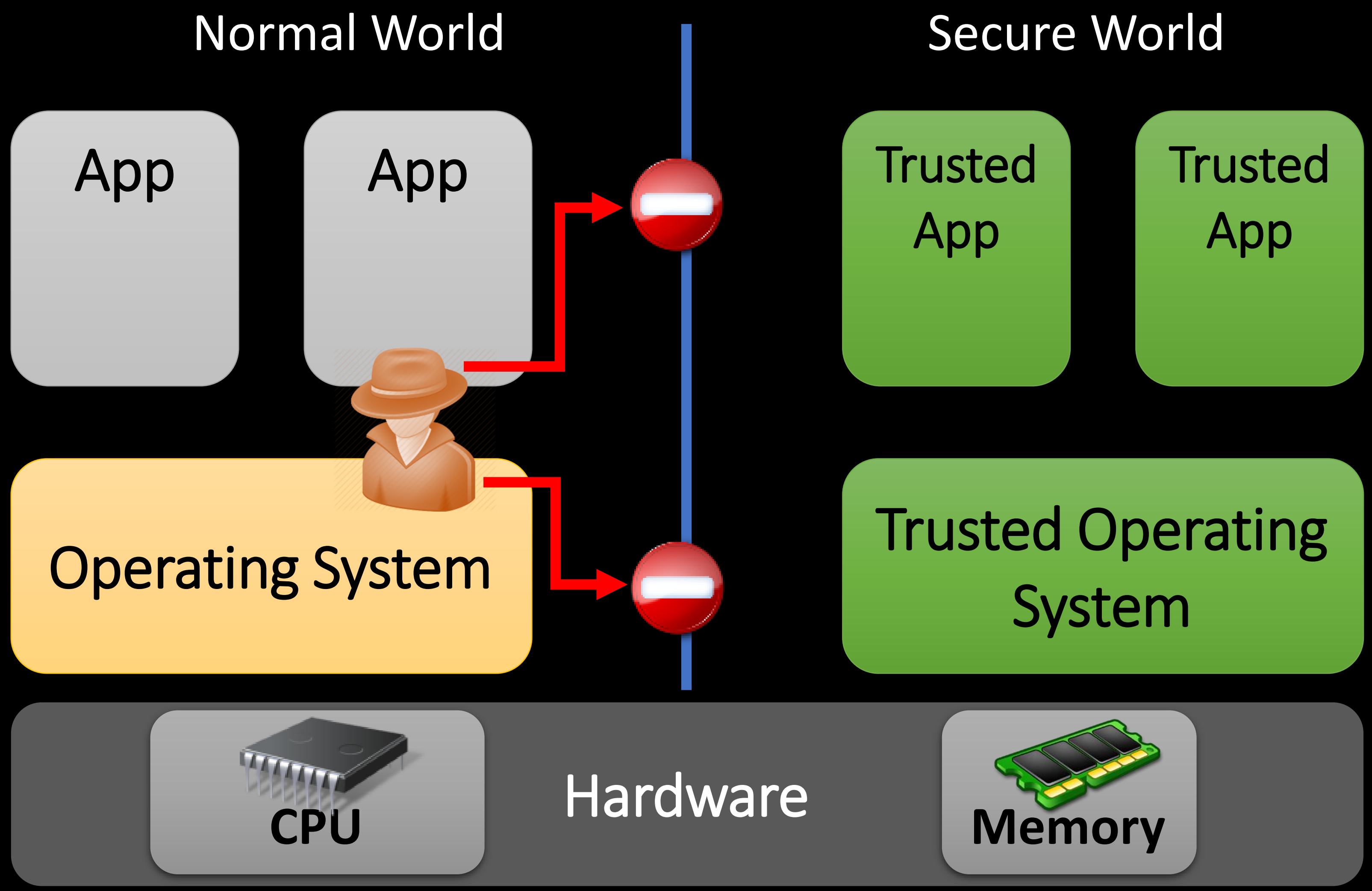
Memory Corruption Attacks in the Context of Trusted Execution Environments

Lucas Davi
Secure Software Systems
University of Duisburg-Essen, Germany

Why Hardware-Assisted Application Security?



Hardware-Assisted Security Enables Implementation of Trusted Execution Environments (TEEs)

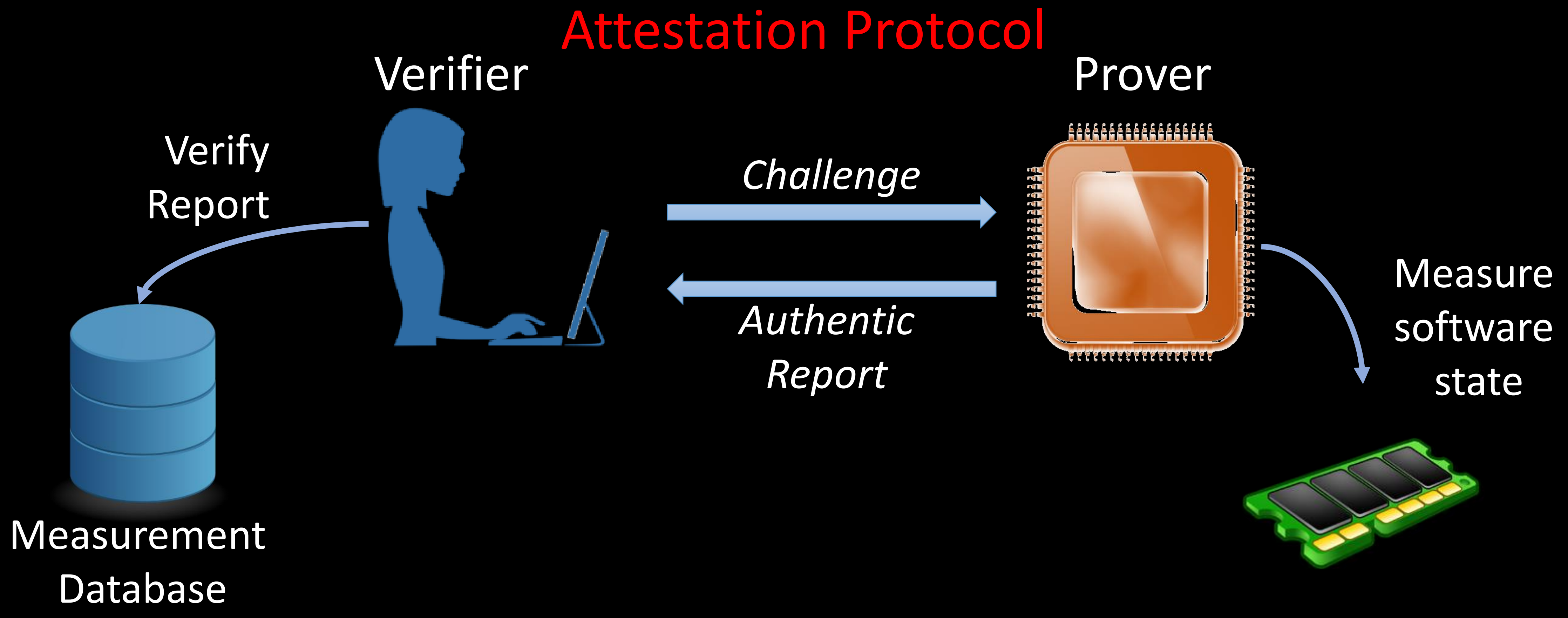


Popular TEE Implementations:

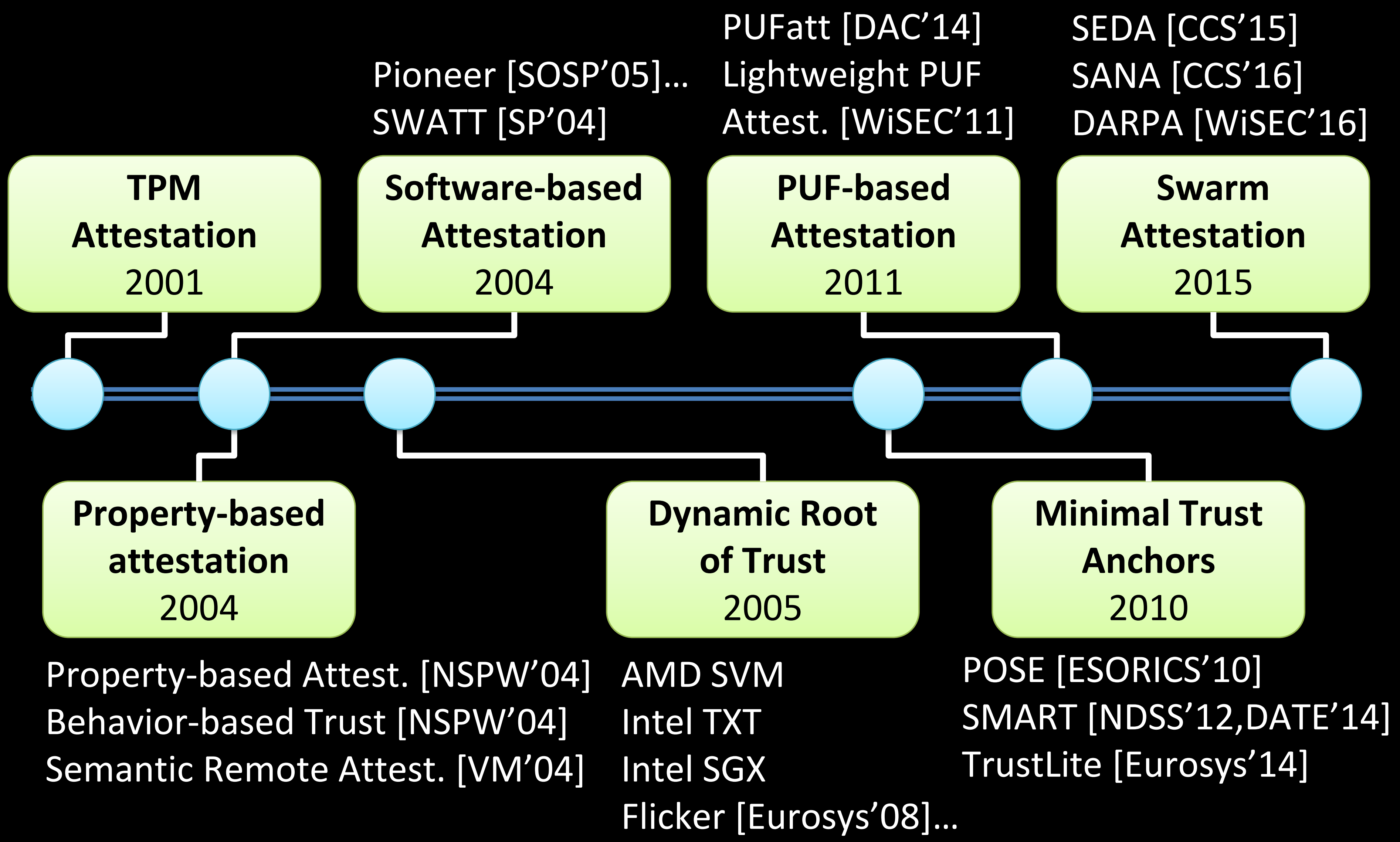
- ARM TrustZone
- Intel Software Guard Extensions (SGX)

Principle of Remote Attestation

- **Goal**: Check if prover is <u>now</u> in a trustworthy state



History of Remote Attestation



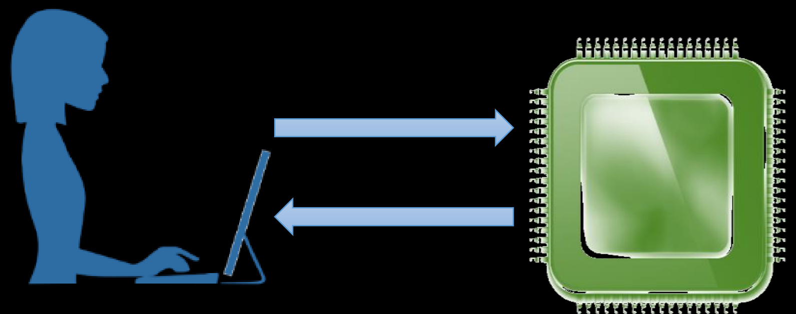
Key Limitation:

current binary attestation schemes do not address run-time (memory corruption) attacks

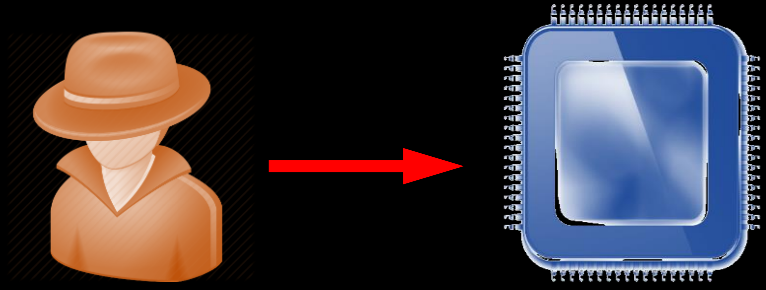
CONTROL-FLOW ATTESTATION

RUN-TIME ATTACKS AGAINST INTEL SGX

TEE BUG FINDING



Embedded System
with
ARM TrustZone



Intel SGX

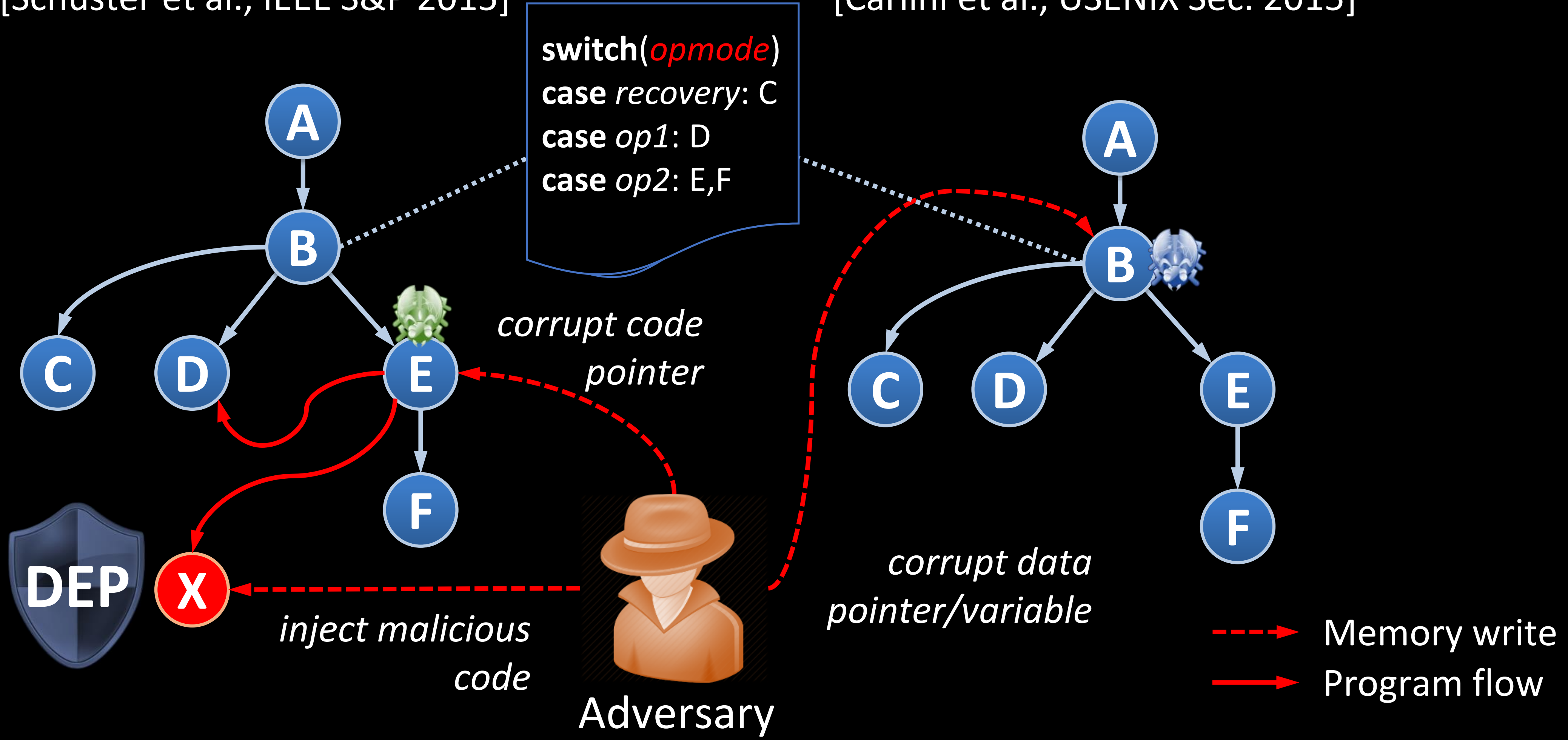
Problem Space of Run-time Attacks

Control-Flow Attack

[Shacham, ACM CCS 2007]
[Schuster et al., IEEE S&P 2015]

Non-Control-Data Attack

[Chen et al., USENIX Sec. 2005]
[Carlini et al., USENIX Sec. 2015]



Related Work

Control-flow integrity (CFI) [Abadi et al., CCS'05]

Data-flow integrity (DFI) [Castro et al., OSDI'06]

Code-pointer integrity (CPI) [Kuznetsov et al., OSDI'14]

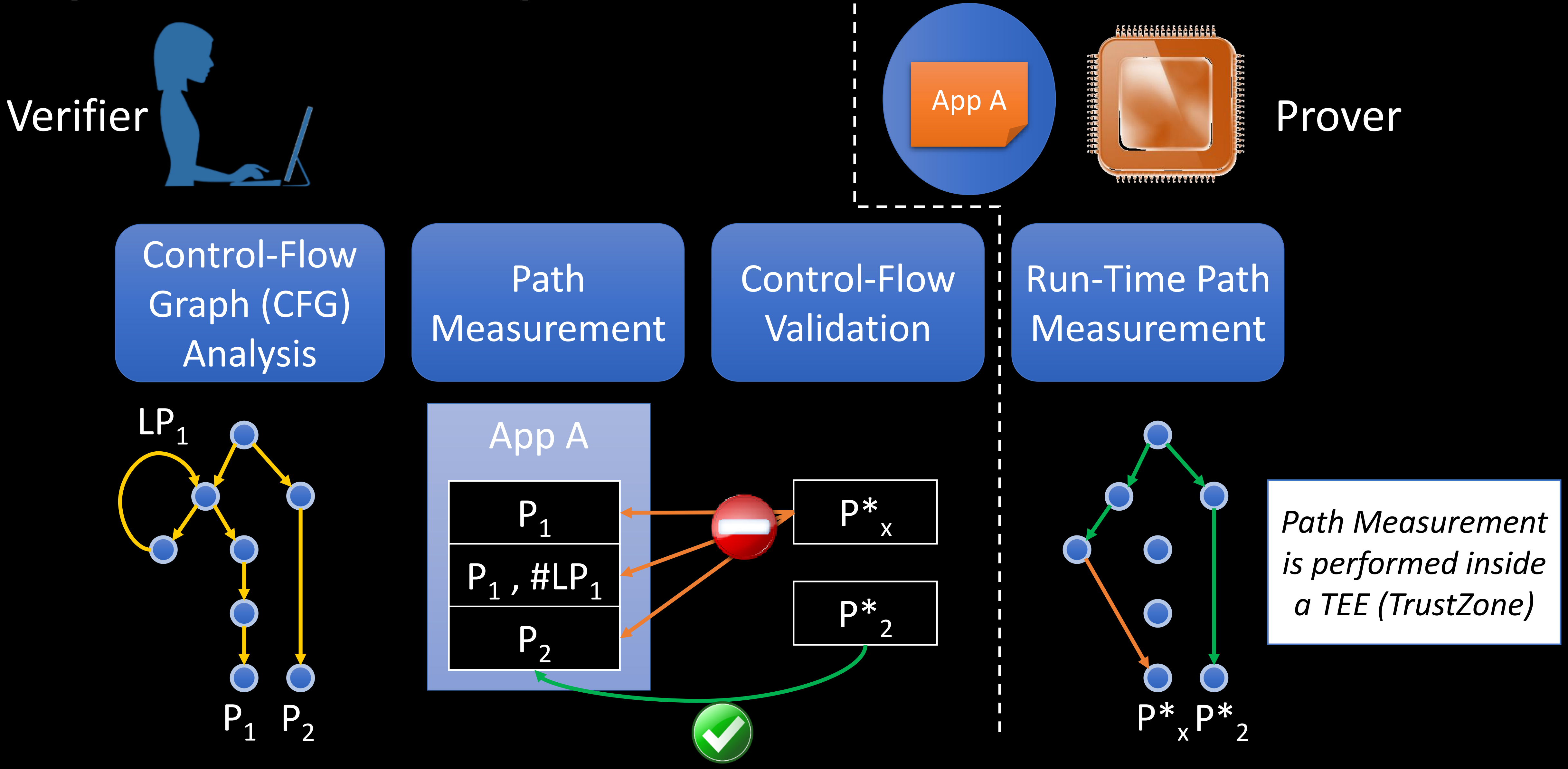
Remote Dynamic Attestation [Kil et al., DSN'09]

Not suitable for control-flow attestation

- Integrity-based schemes usually target a specific runtime attack class
- These schemes only output whether an attack occurred but don't attest the control-flow path

C-FLAT

[Abera et al., CCS 2016]

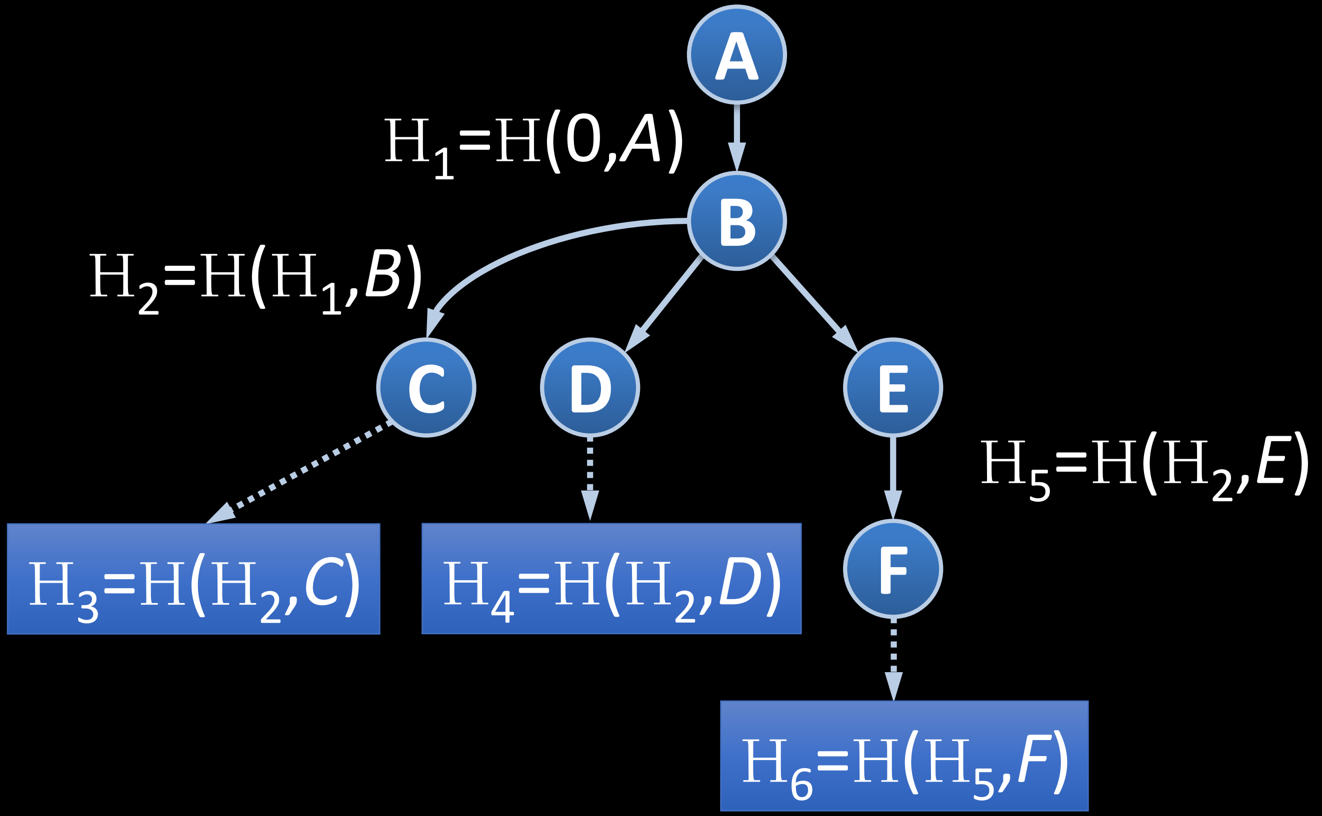


How to attest the executed control flows without transmitting all executed branches?

C-FLAT Measurement Function

Cumulative Hash Value: $H_i = H\,(\,H_{i-1}, N\,)$

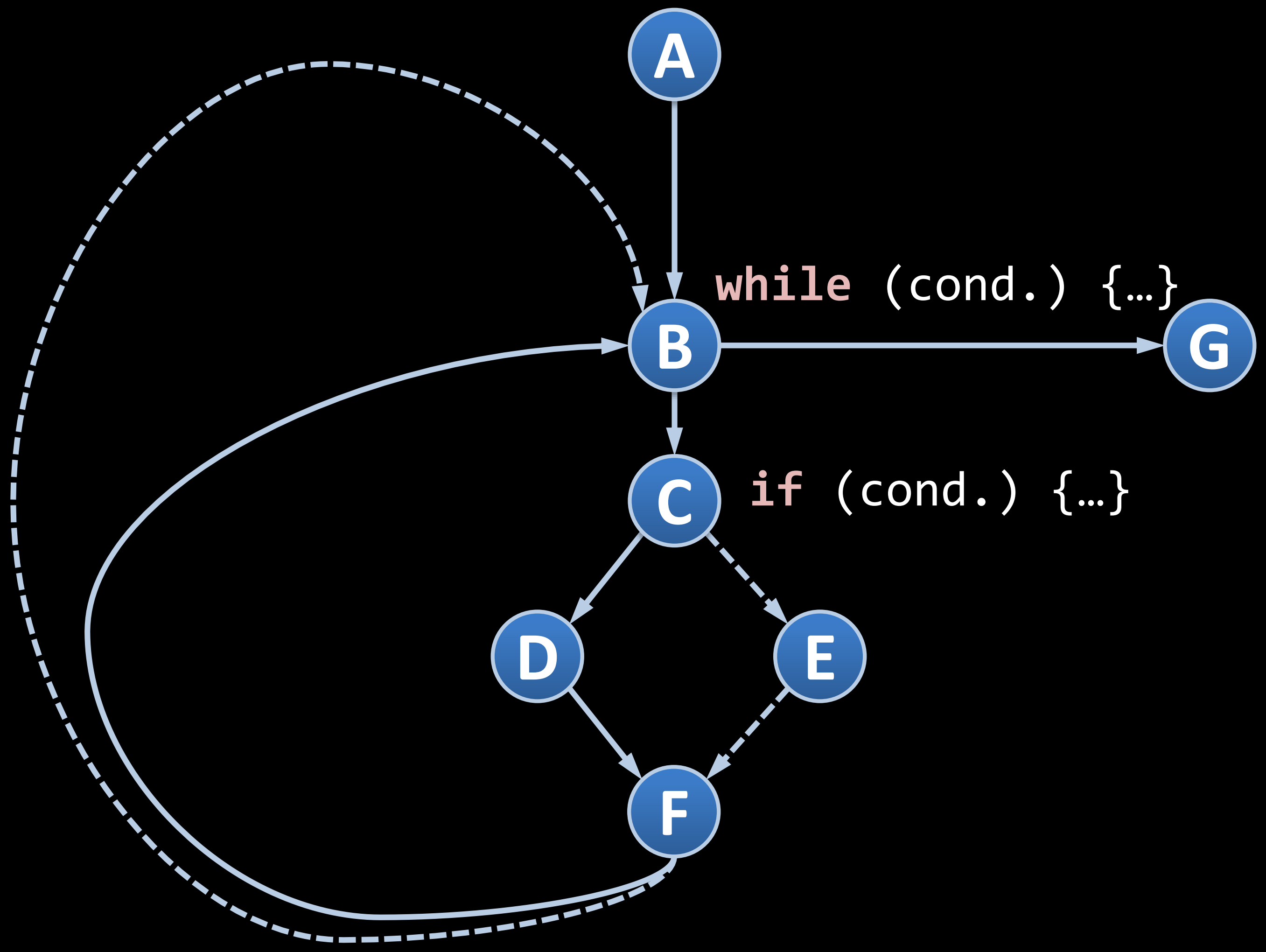
- H_{i-1} - previous hash result
- **N** - instruction block (node) just executed



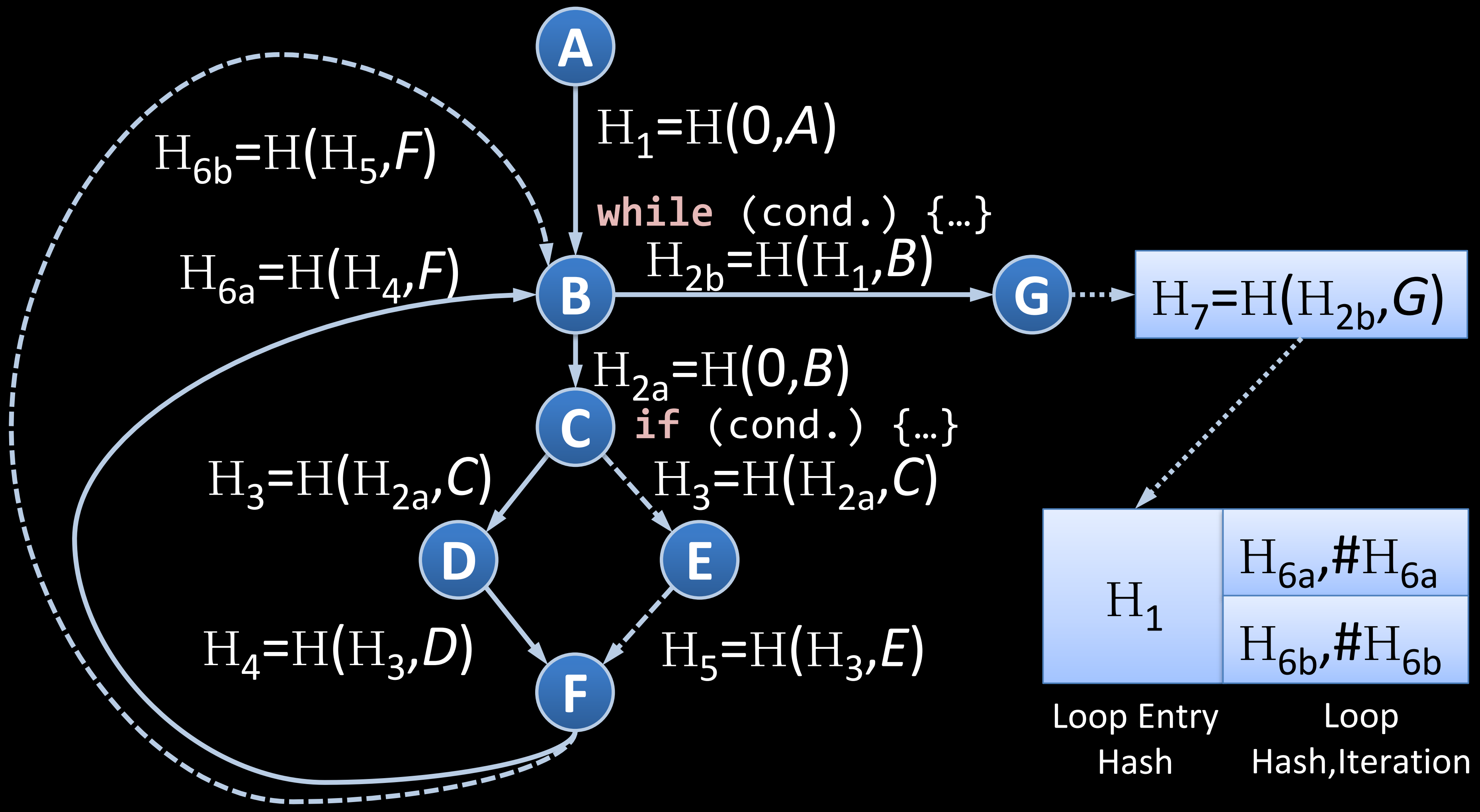
Loops are a challenge!

Different loop paths
and loop iterations lead to many valid
hash values

C-FLAT: Loop Handling

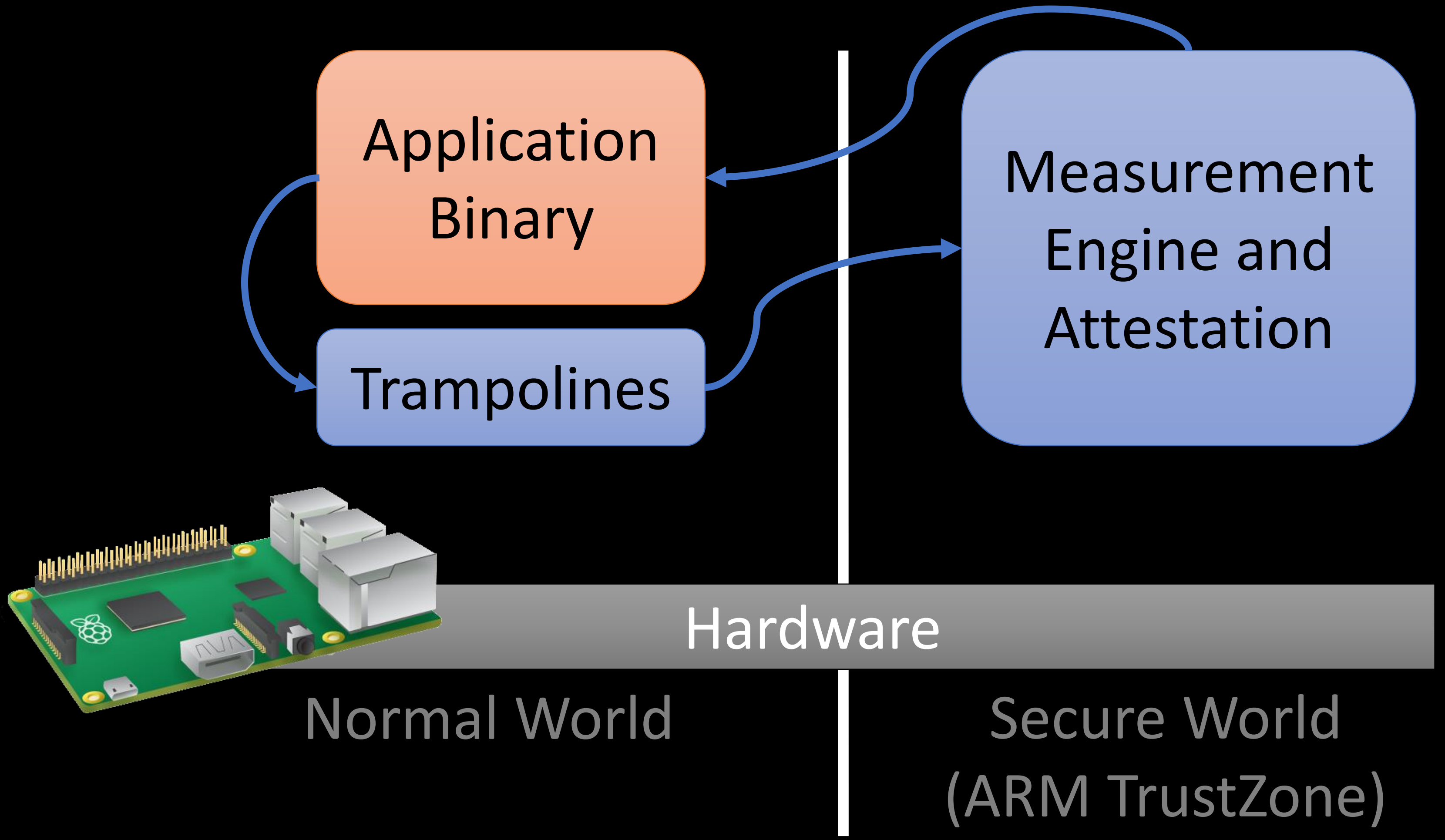


C-FLAT: Loop Handling



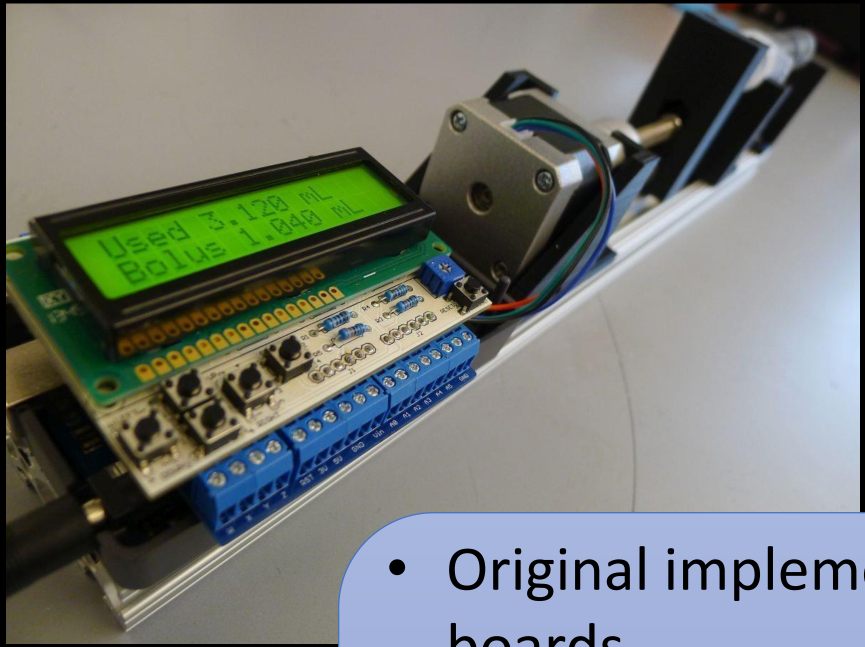
Prototype Architecture

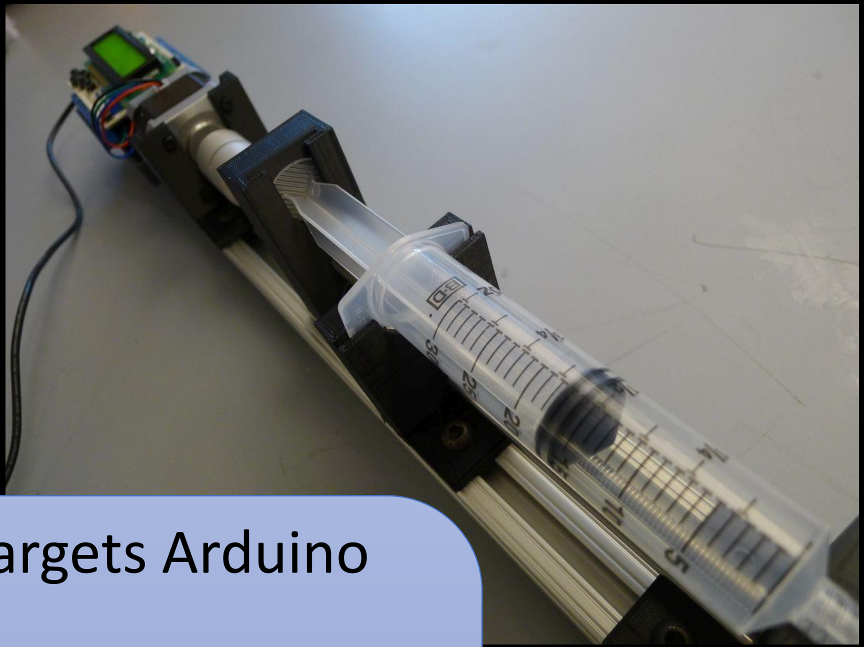
- Implementation on Raspberry Pi 2



Evaluation: Syringe Pump

Source: https://hackaday.io/project/1838-open-syringe-pump





- Original implementation targets Arduino boards
- We ported the code to Raspberry Pi
- 13,000 instructions with 332 CFG edges of which 20 are loops
- Main functions are **set-quantity** and **move-syringe**

Applying C-FLAT to Syringe Pump

main()

```
while (1) {
  if (serialReady()) {
    cfa_init;
    processSerial();
    cfa_quote;
  }
}
```

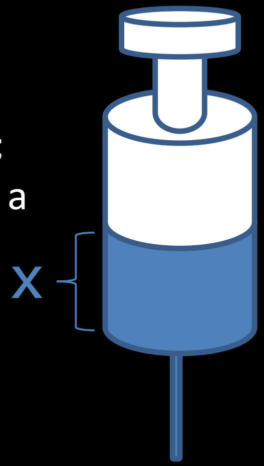
action(direction,bolus)

```
steps = bolus * steps_per_mL
if (direction == PUSH) {
  /* set stepper direction */
} else { /* PULL */
  /* set stepper direction */
}
for (steps) {
  /* move stepper */
}
```

processSerial()

```
if (input == '+') {
  action(PUSH,bolus);
  updateScreen();
}
else if (input == '-') {
  action(PULL,bolus);
  updateScreen();
}
```

bolus = dose of drug; volume of cylinder for a particular height



Please note that this slide shows a simplified view of the Syringe pump code and control-flow graph.

Final Hash Measurements

action(direction,bolus)

```
steps = bolus * steps_per_mL
if (direction = PUSH) {
  /* set stepper direction */
} else /* PULL */
  /* set stepper direction */
}
for (steps) {
  /* move stepper */
}
```

Final Measurements for PUSH, PULL operations:

b3 c5 ca c4 6f dc 6a d0
4a 80 10 09 af a3 59 70

e0 9a f6 48 11 65 17 94
a7 0b 06 f0 ba e4 75 75

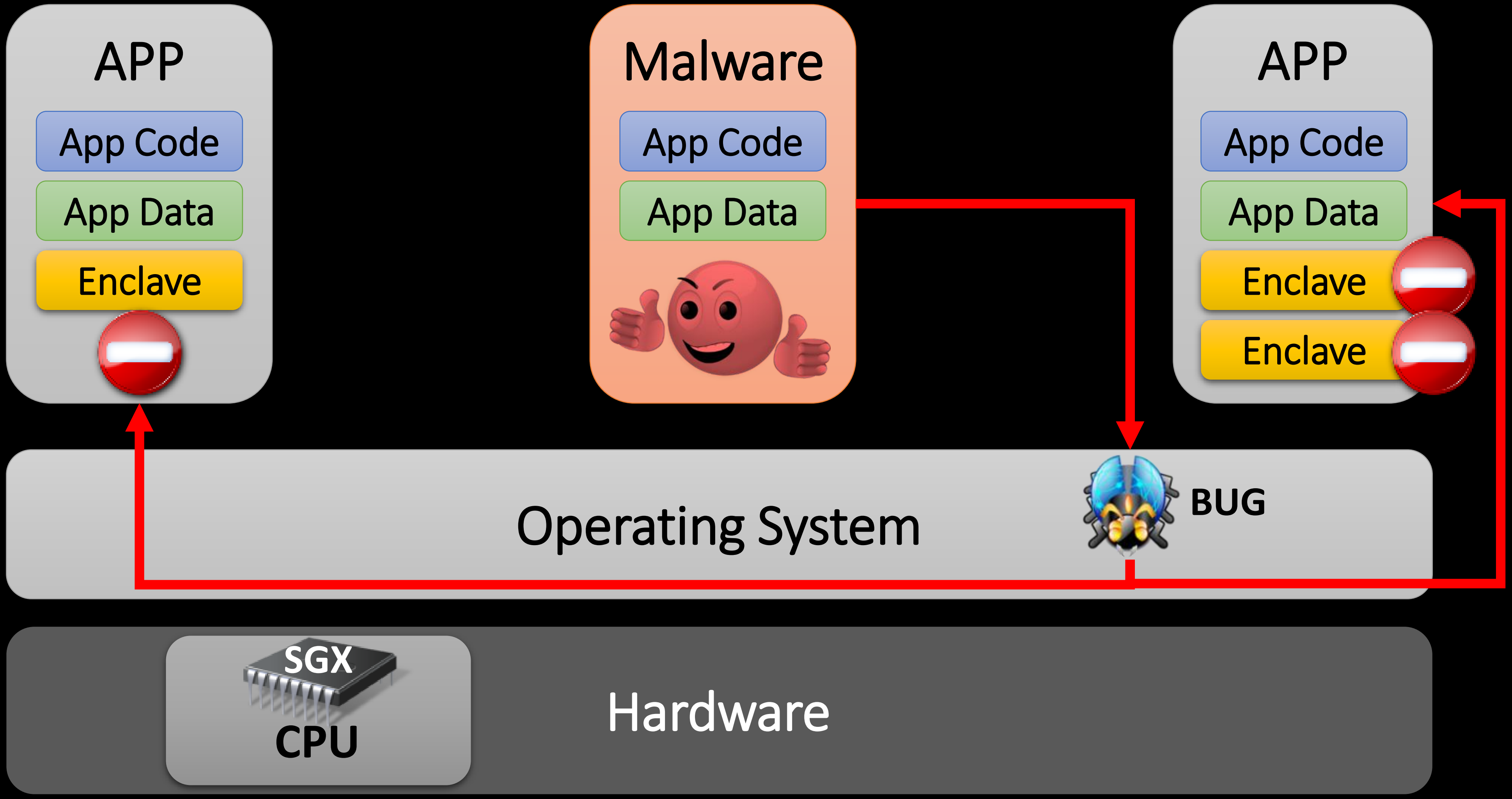
Loop Measurement:

97 78 fb fc 93 09 4e d7
ac 32 5d 65 eb 29 08 0c
(#iterations)

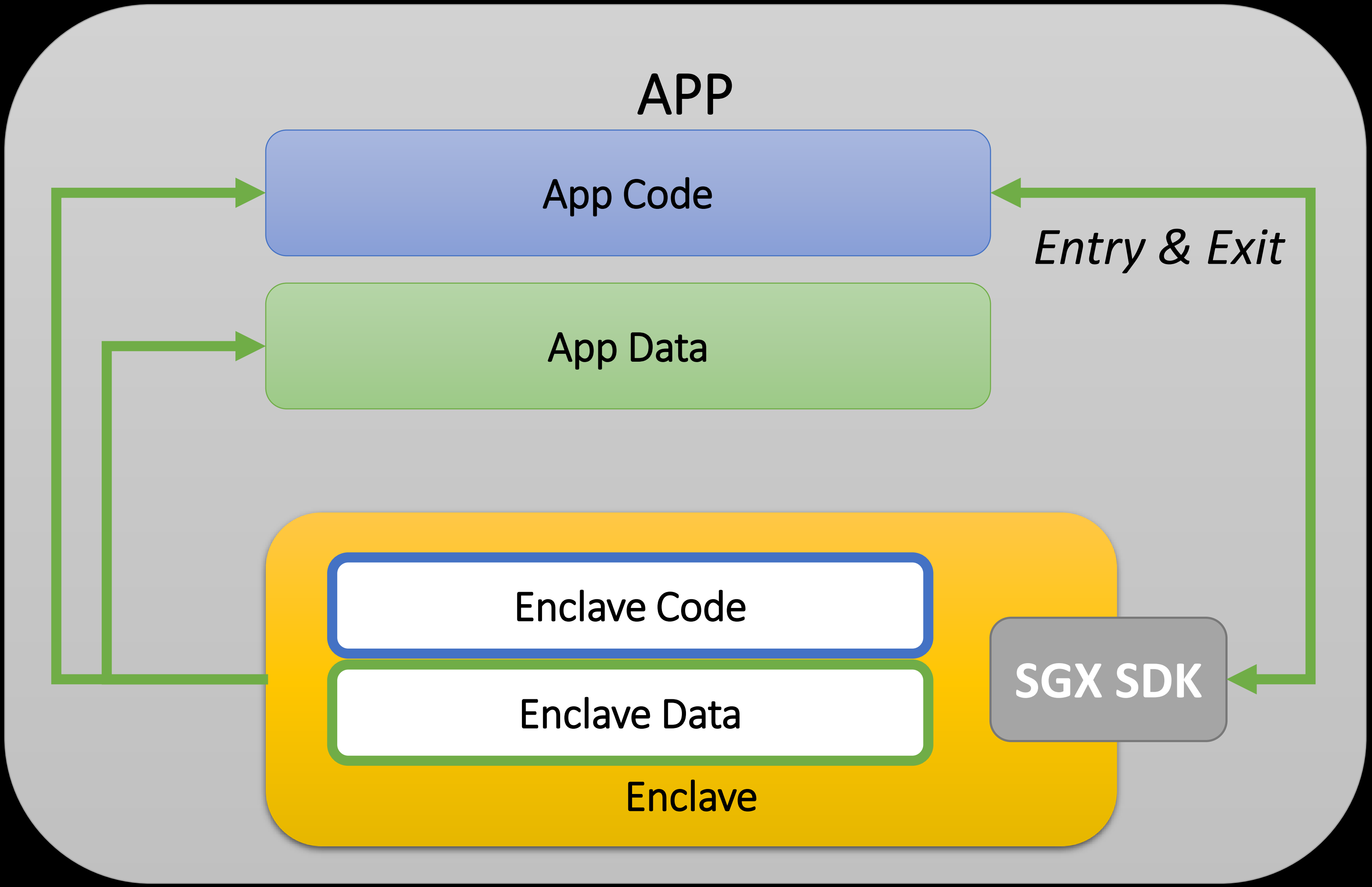
Open Questions

- How to address performance overhead?
  - ➢Tackled based on hardware assistance in a follow-up work, LO-FAT [DAC‘17]
- What can go wrong inside the TEE?
  - ➢Next part of this talk with focus on SGX

Overview on Intel SGX



App-Enclave Communication



Entry to Enclave code is only allowed at **pre-defined entry points**

Academic Research on Side-Channel Attacks Against SGX

Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems

Yuanzhong Xu, *The University of Texas at Austin*, *yxu@cs.utexas.edu*
Weidong Cui, *Microsoft Research*, *wdcui@microsoft.com*
Marcus Peinado, *Microsoft Research*, *marcuspe@microsoft.com*

Software Grand Exposure: SGX Cache Attacks Are Practical

Ferdinand Brasser[1], Urs Müller[2], Alexandra Dmitrienko[2], Kari Kostiainen[2], Srdjan Capkun[2], and Ahmad-Reza Sadeghi[1]

CacheZoom: How SGX Amplifies The Power of Cache Attacks

Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth

Boffins show Intel's SGX can leak crypto keys

Software Guard Extensions are supposed to hide data. But the 'Prime+Probe attack' fixes that

Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing

Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, and Hyesoon Kim, *Georgia Institute of Technology;* Marcus Peinado, *Microsoft Research*

Telling Your Secrets Without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution

Jo Van Bulck, *imec-DistriNet, KU Leuven;* Nico Weichbrodt and Rüdiger Kapitza, *IBR DS, TU Braunschweig;* Frank Piessens and Raoul Strackx, *imec-DistriNet, KU Leuven*

FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution

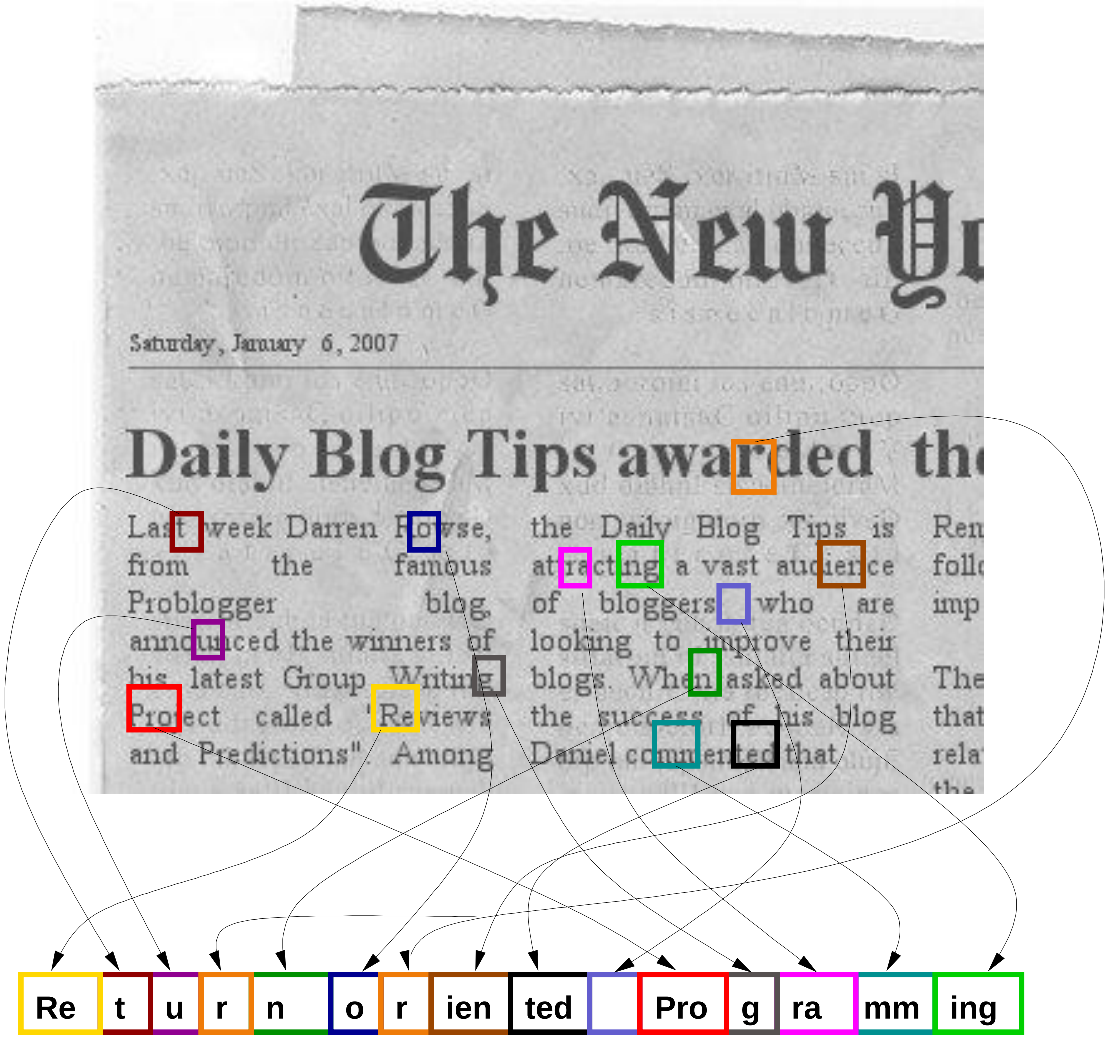
Jo Van Bulck, *imec-DistriNet, KU Leuven;* Marina Minkin, *Technion;* Ofir Weisse, Daniel Genkin, and Baris Kasikci, *University of Michigan;* Frank Piessens, *imec-DistriNet, KU Leuven;* Mark Silberstein, *Technion;* Thomas F. Wenisch, *University of Michigan;* Yuval Yarom, *University of Adelaide and Data61;* Raoul Strackx, *imec-DistriNet, KU Leuven*

SGXPECTRE Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution

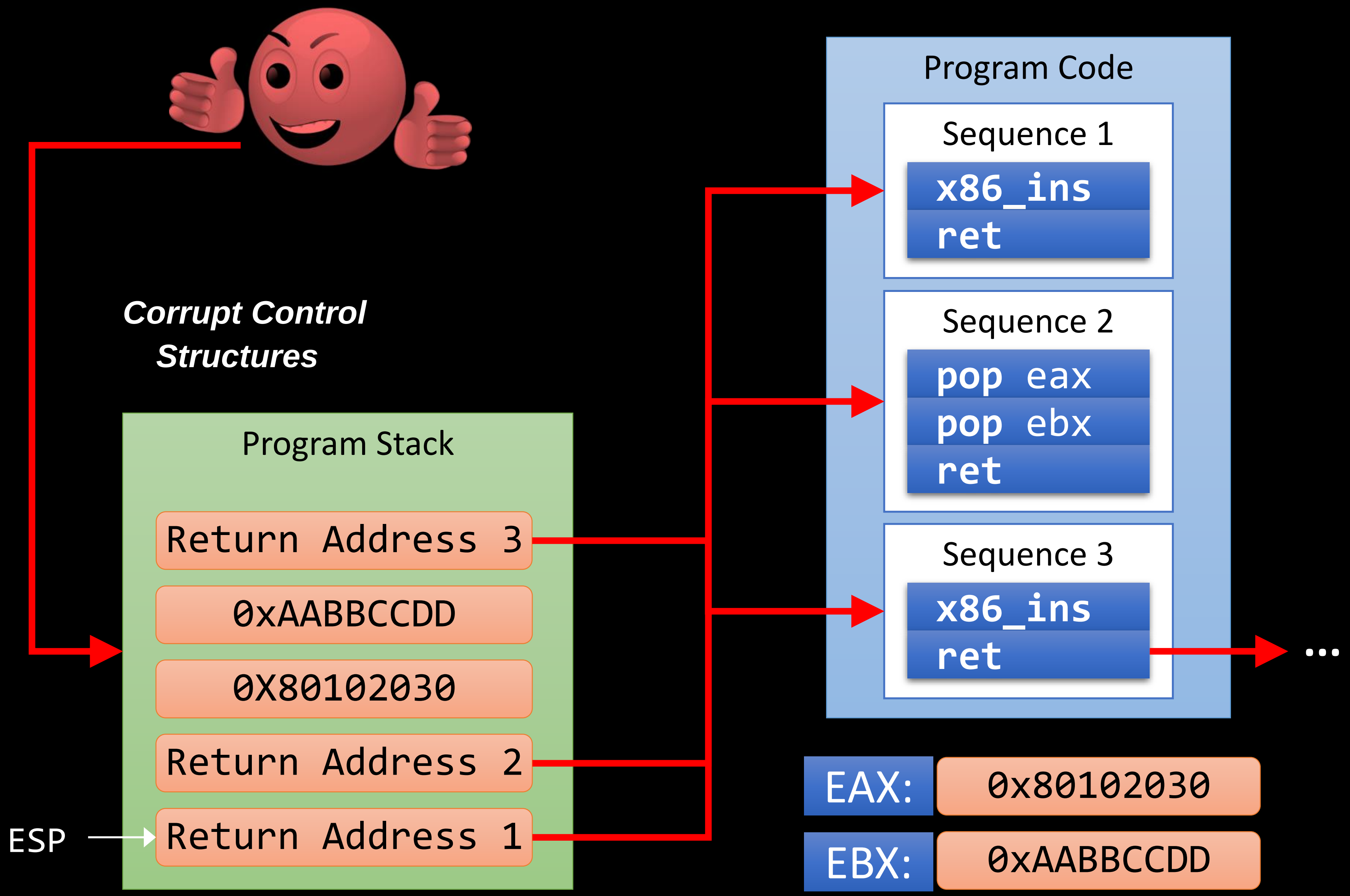
Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, Ten H. Lai
Department of Computer Science and Engineering

What about Return-Oriented Programming Attacks?

Return-Oriented Programming

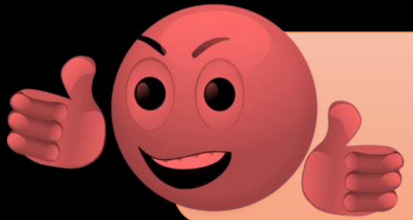


Return-Oriented Programming Attack



First Run-Time Attacks and Defenses Targeting Intel SGX

Related Work



Dark ROP
[USENIX Sec. 2017]

- Analyzes the threat of memory corruption vulnerabilities in the context of SGX
- Presents ROP attack against (unknown) encrypted enclave binaries
- Based on probing attacks
- Requires kernel privileges and ability to repeatedly crash the enclave

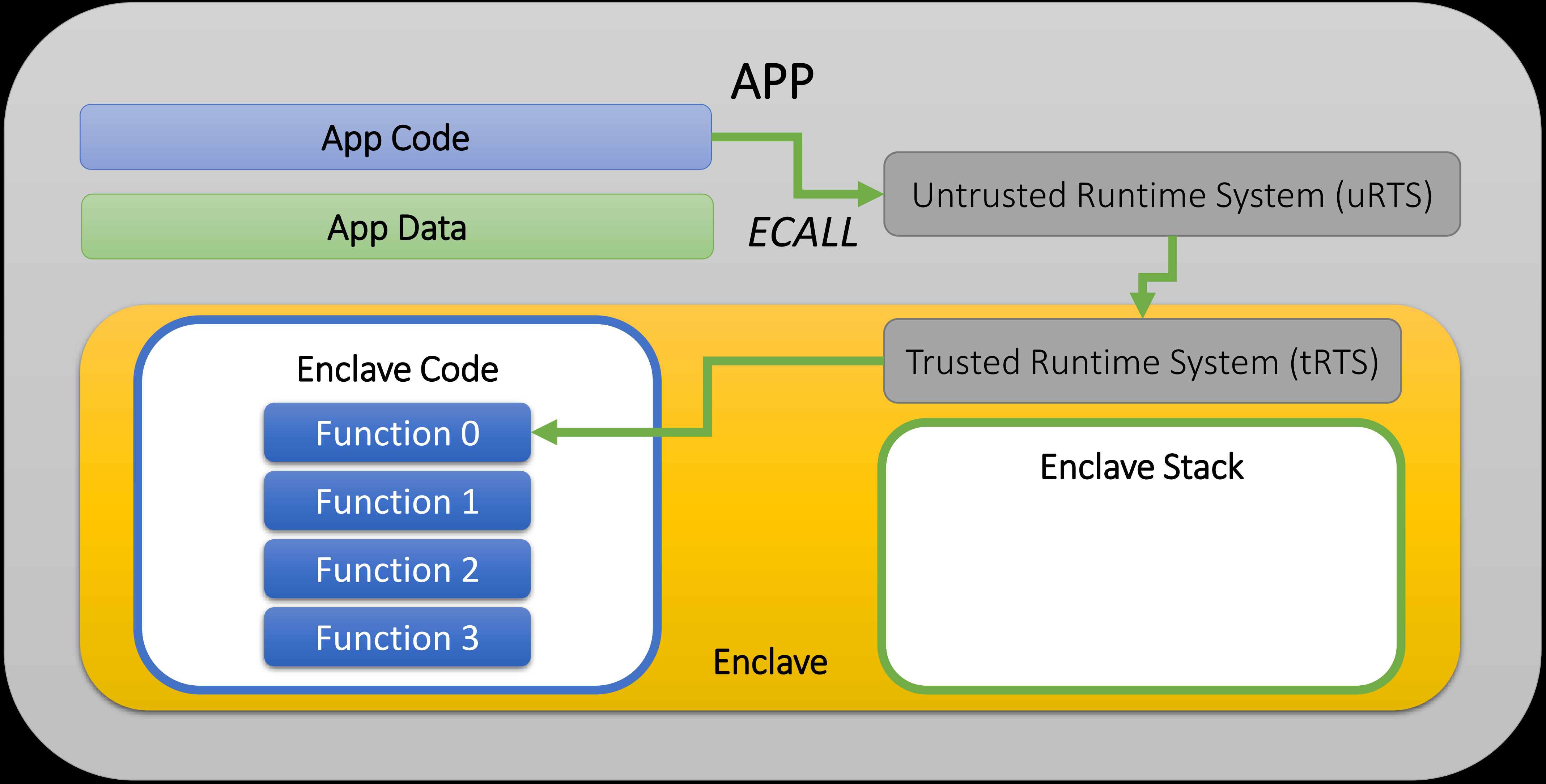
SGX-Shield
[NDSS 2017]

- Enforces fine-grained memory randomization of SGX enclave
- Software-based data execution prevention (DEP)
- Proposes control-flow integrity for return instructions

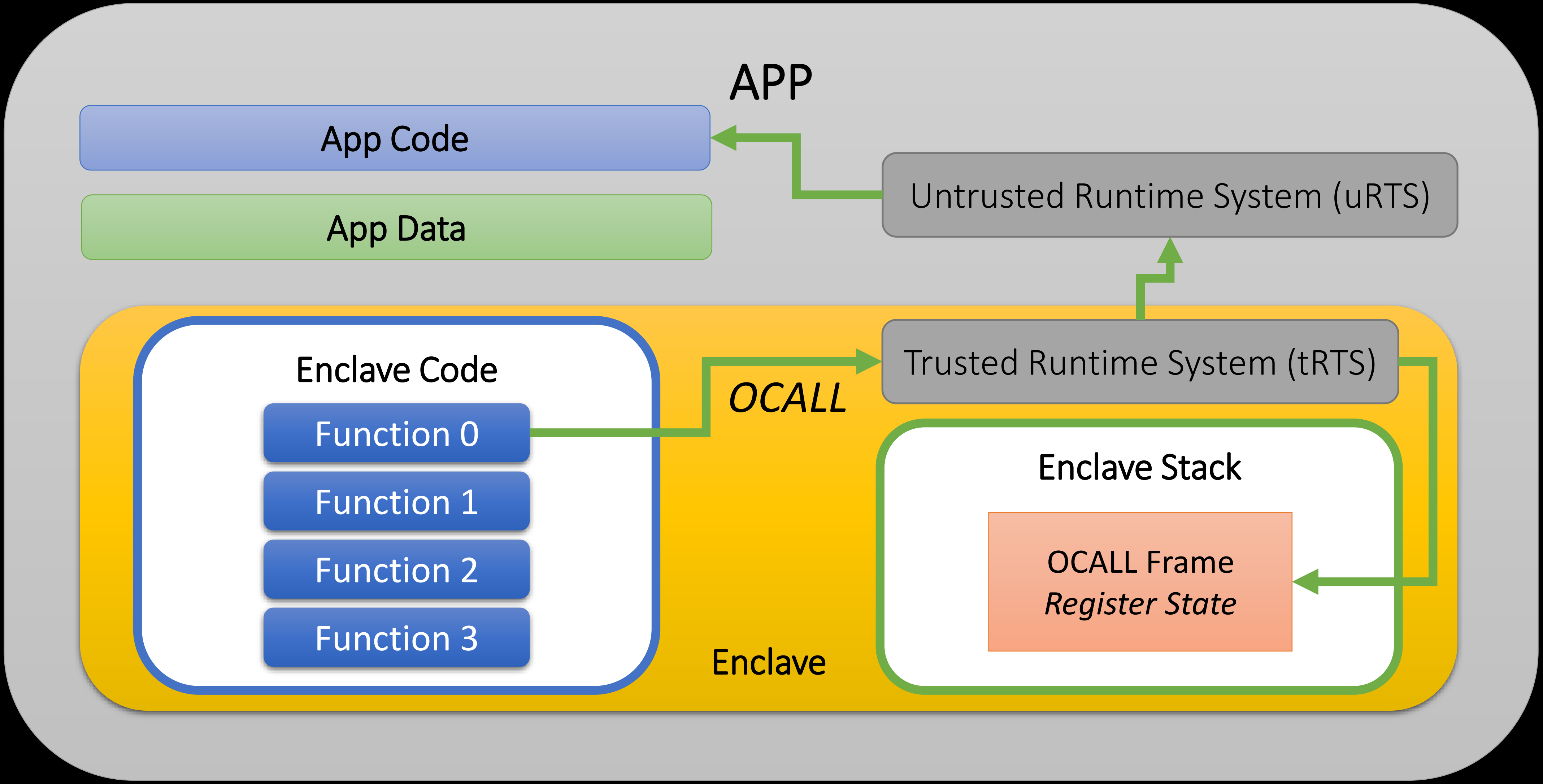
Can we bypass memory randomization in SGX?

[Biondo et al., USENIX Security 2018]
Our main observation is that the Intel SGX SDK includes dangerous return-oriented programming gadgets which are essential for app-enclave communication

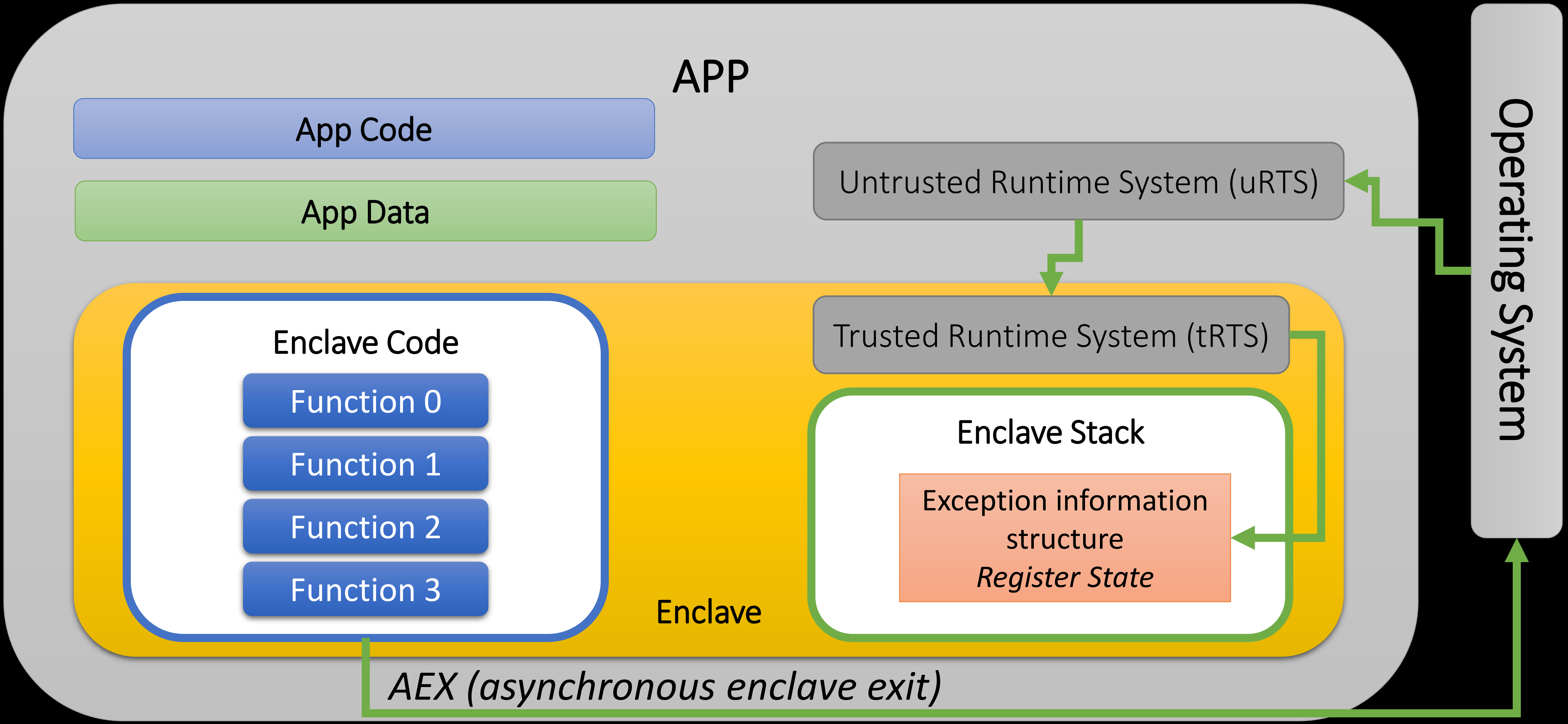
ECALL: Call into an enclave



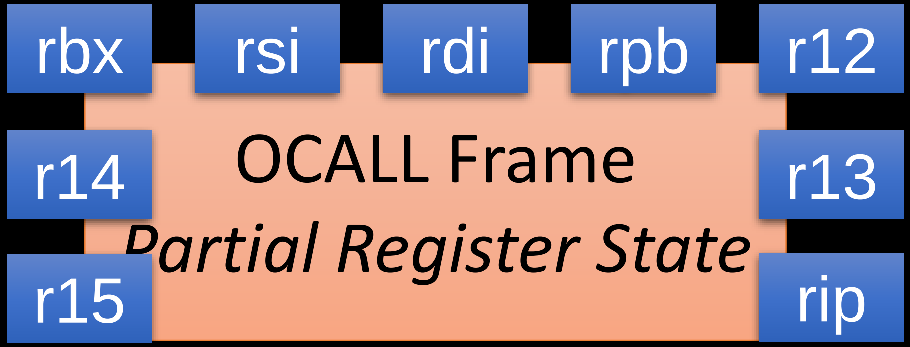
OCALL: Enclave Call to the Host Application



AEX: Asynchronous Enclave Exit (Exception)



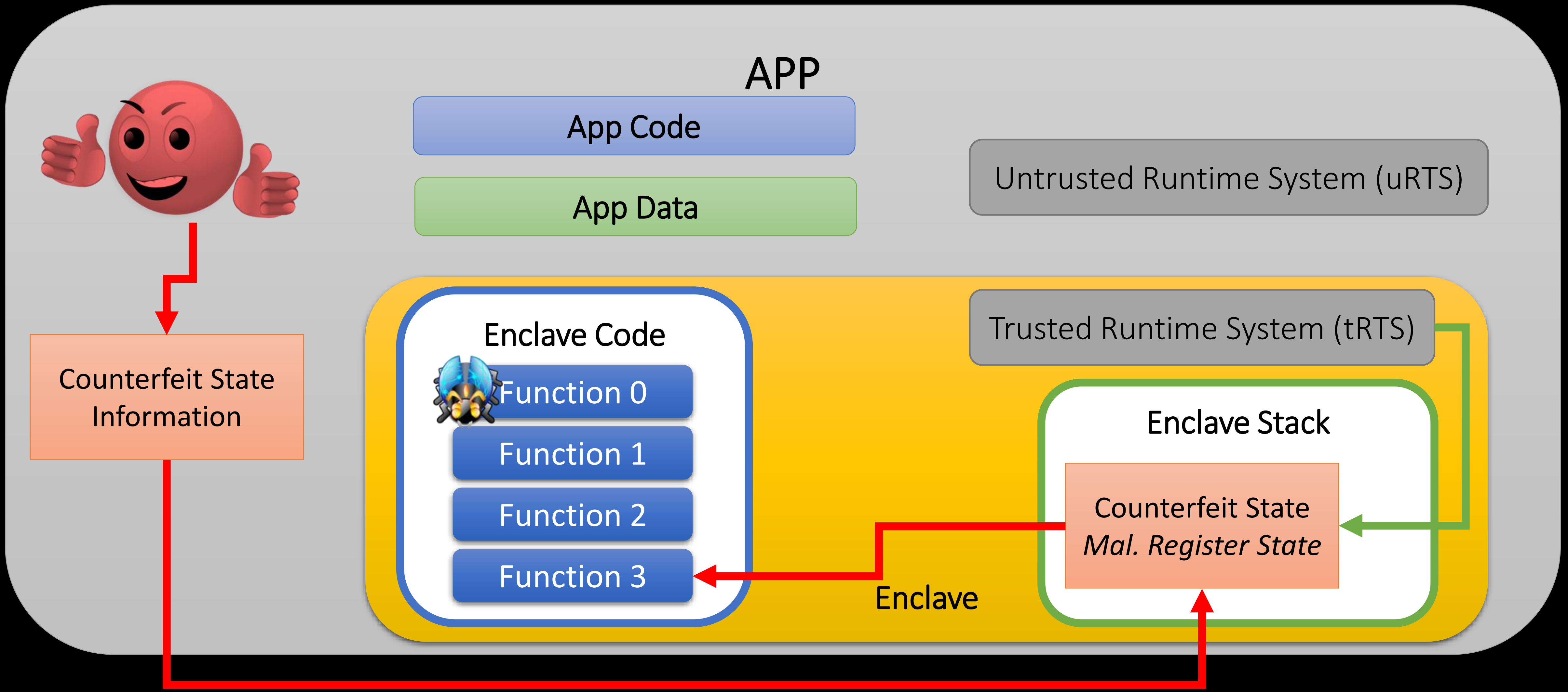
Restoring State is Critical



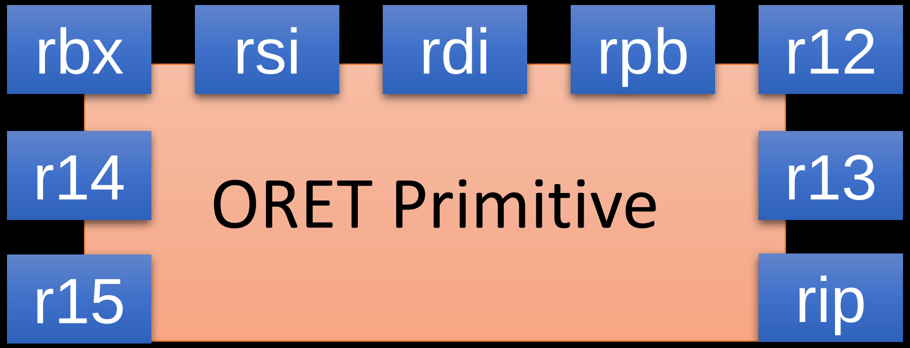
- When OCALL returns, the register state is restored by the tRTS function *asm_oret()*
- If an attacker manages to inject a frake ocall frame, he controls the subsequent state

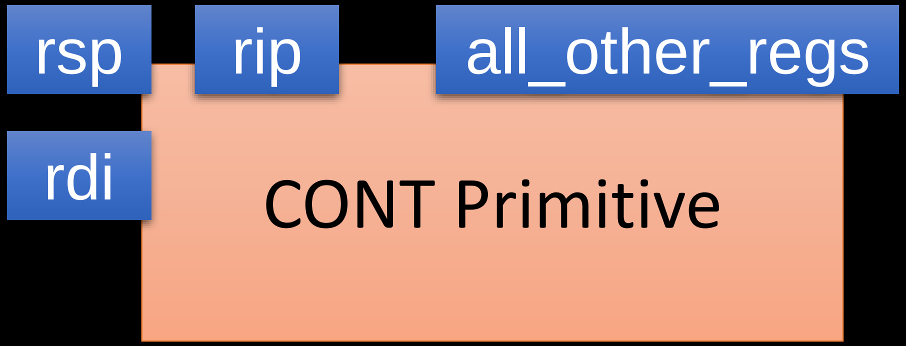
- After handling the exception, the register state is restored by the tRTS function *continue_execution()*
- If an attacker manages to inject a fake exception structure, he controls the subsequent state

Basic Attack Idea



Two Attack Primitives

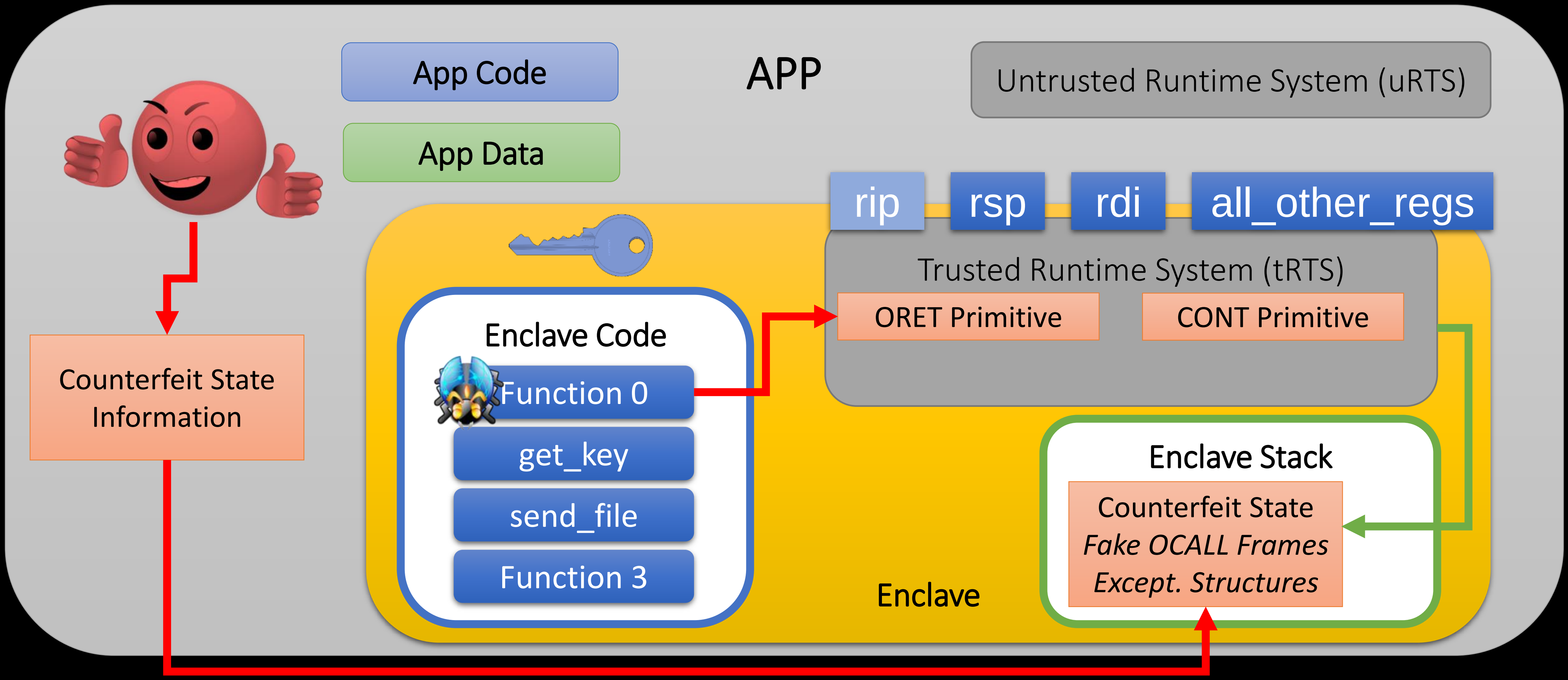




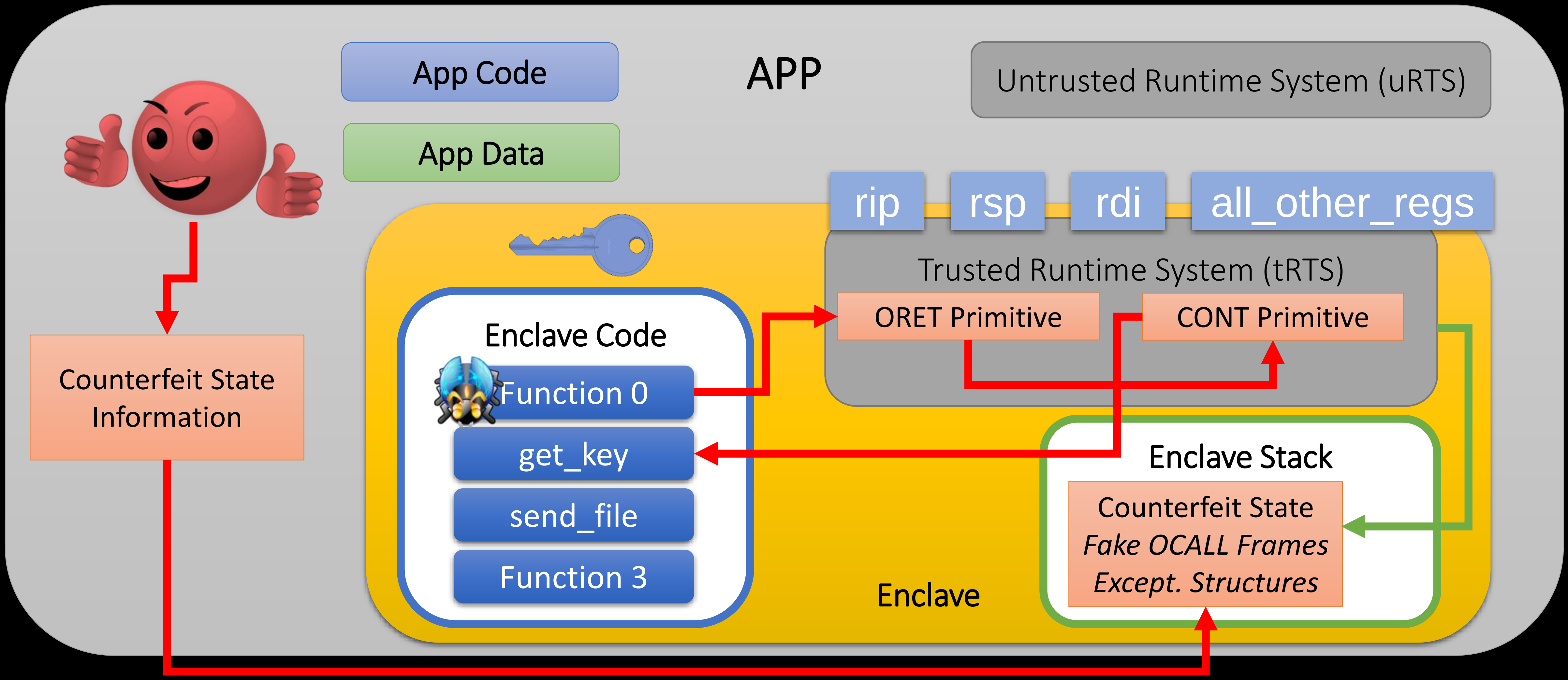
- Primitive to exploit OCALL mechanism
- It is based on injecting fake OCALL frames
- Prerequisites: stack control

- Primitive to exploit asynchronous exception handling in SGX
- Based on injecting fake exception structures
- Prerequisites: function pointer overwrite and control of rdi register

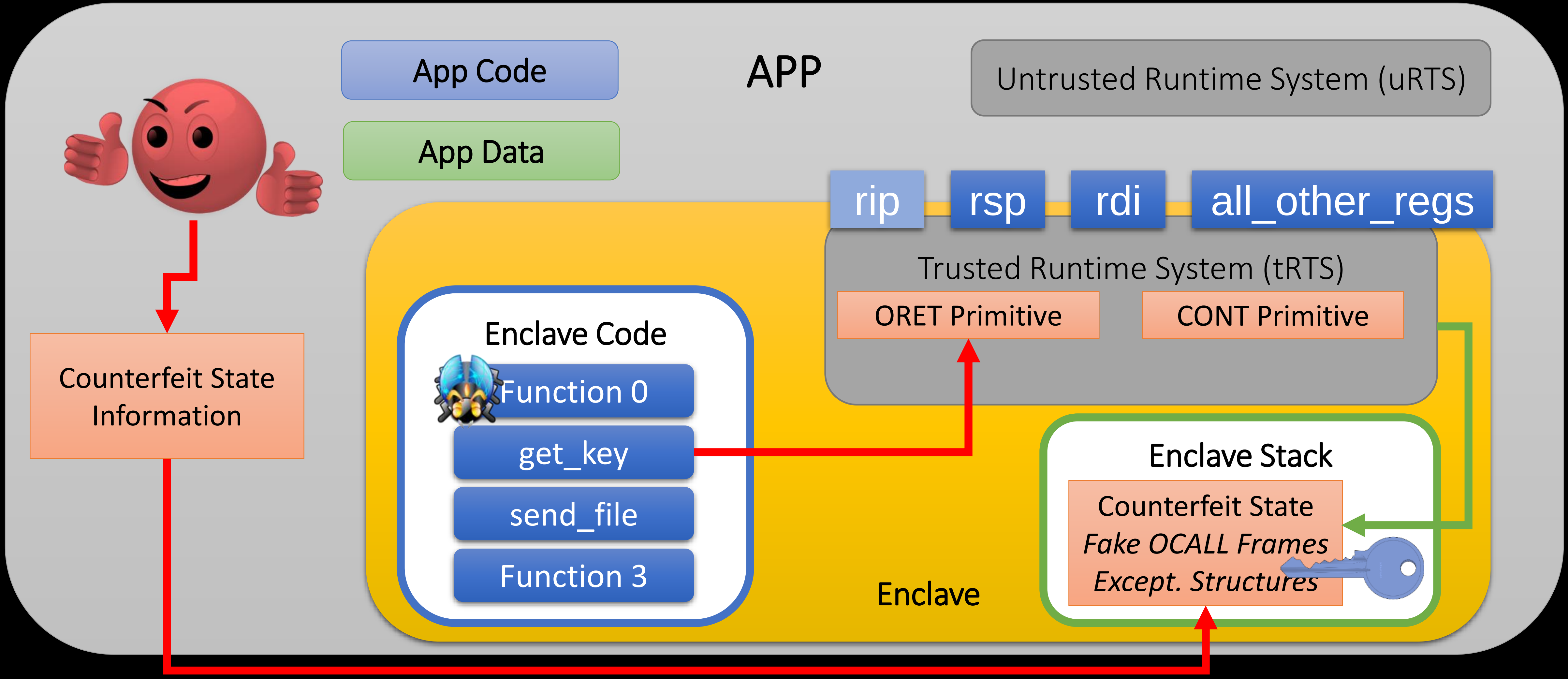
Attack Workflow for Stealing SGX-Protected Keys



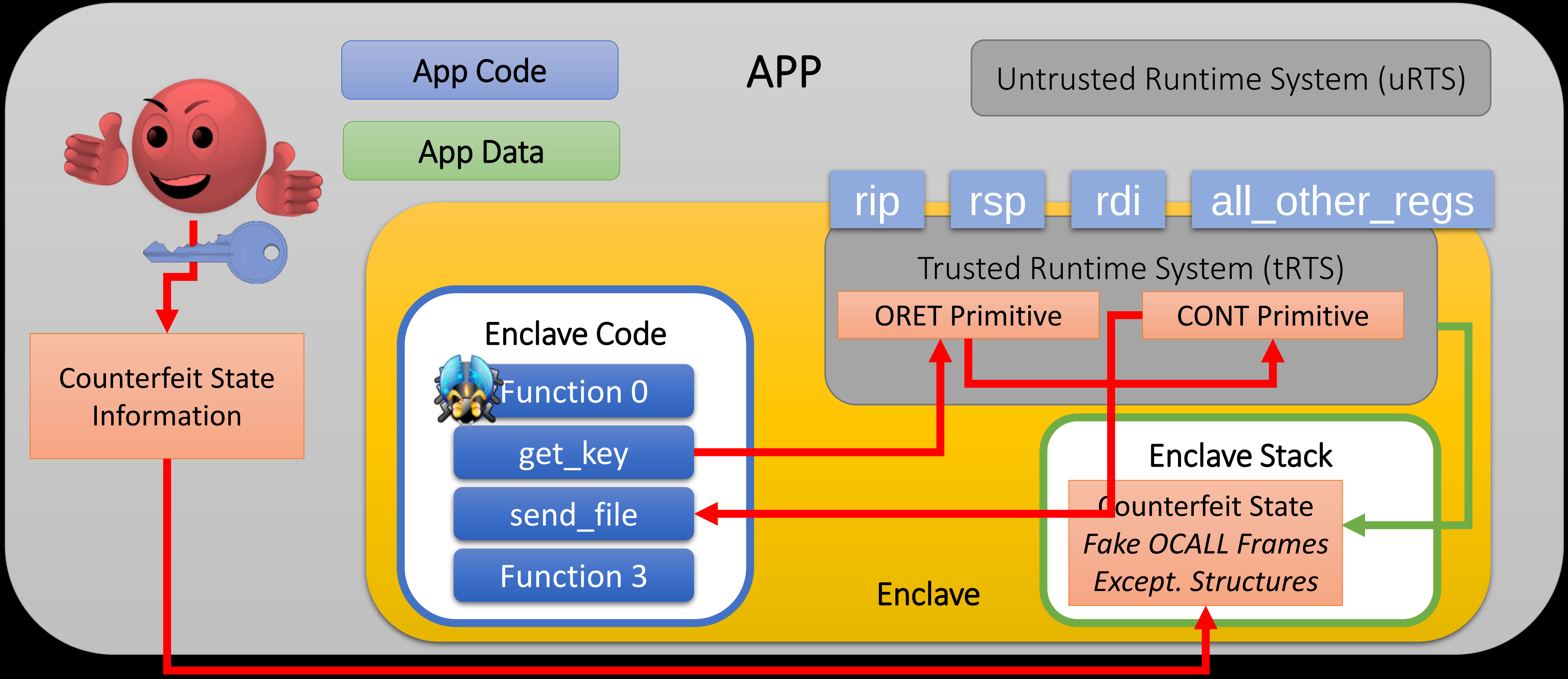
Attack Workflow for Stealing SGX-Protected Keys



Attack Workflow for Stealing SGX-Protected Keys

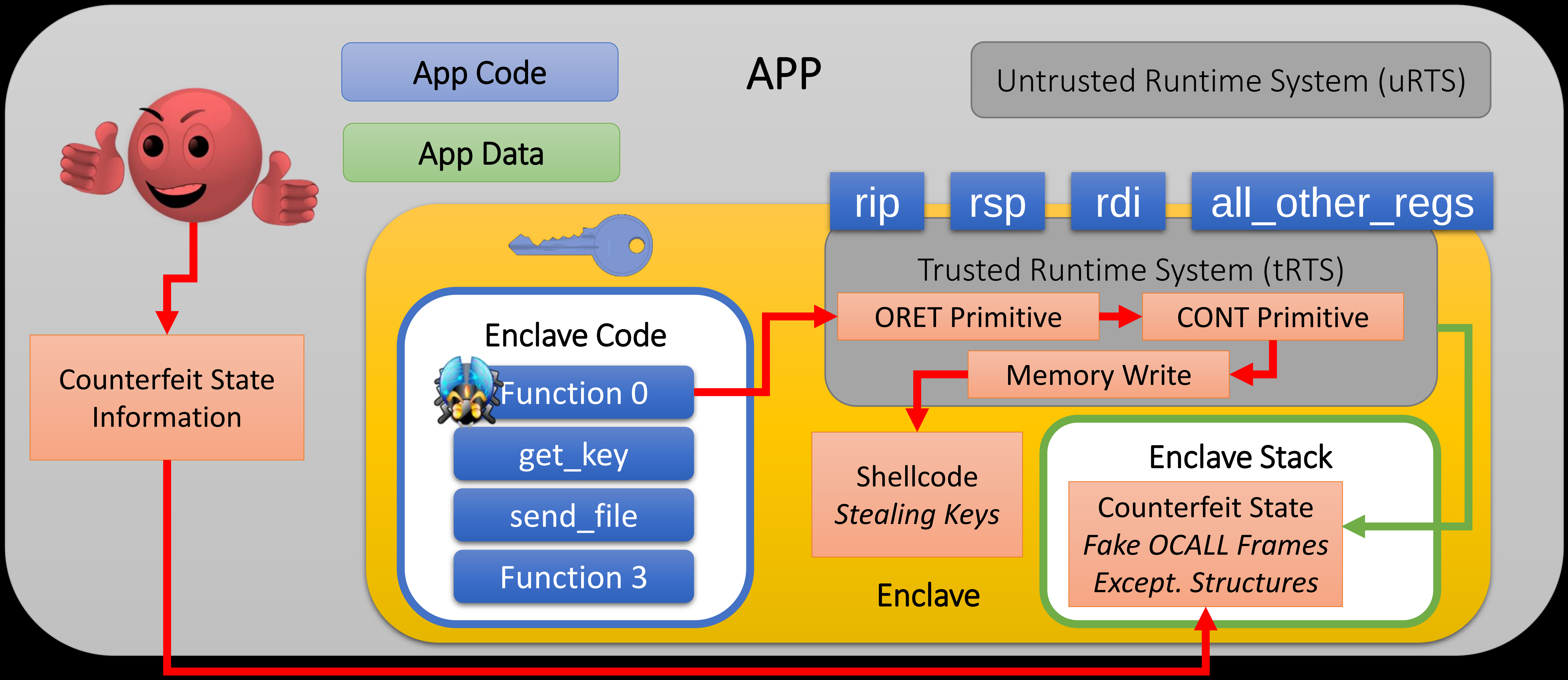


Attack Workflow for Stealing SGX-Protected Keys



However, this attack doesn‘t work if SGX-Shield randomizes the SGX address space

Revisited Attack to Bypass SGX-Shield

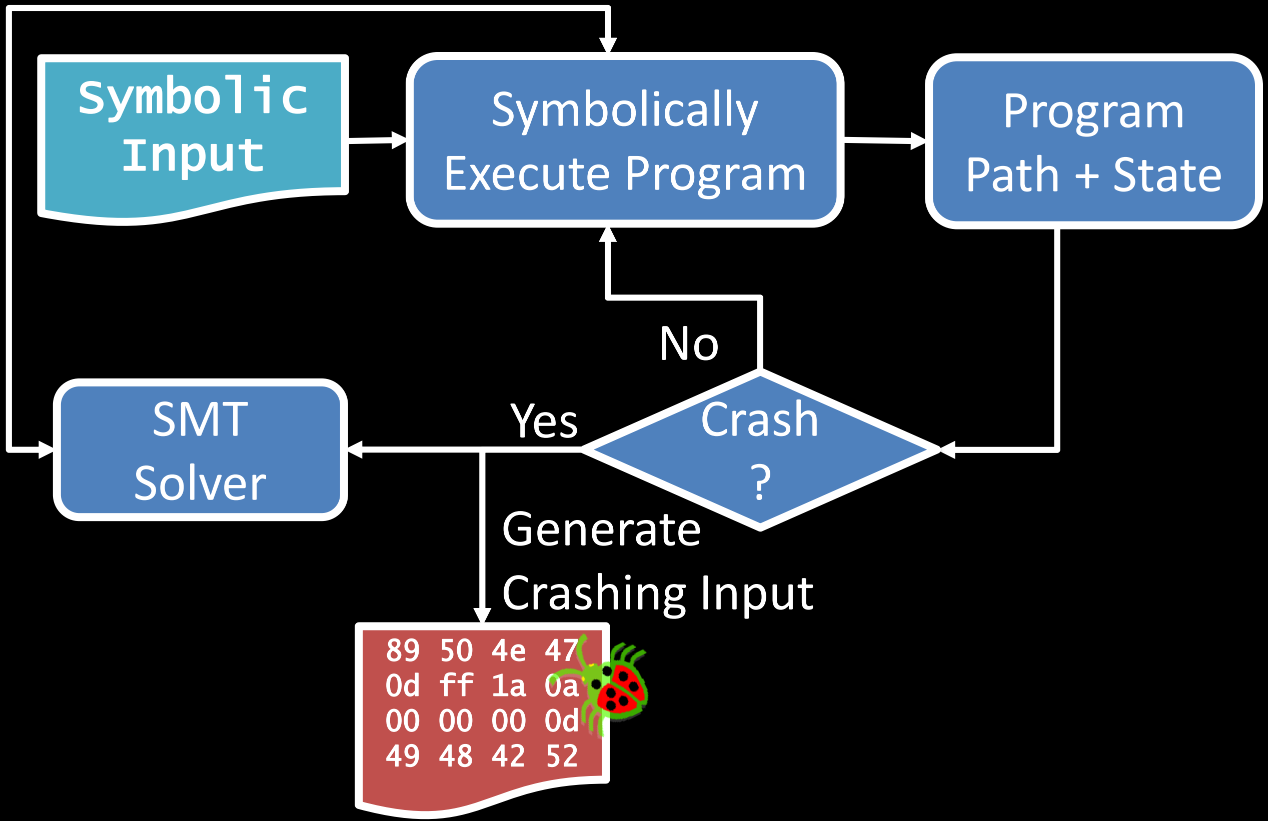


Possible Defenses

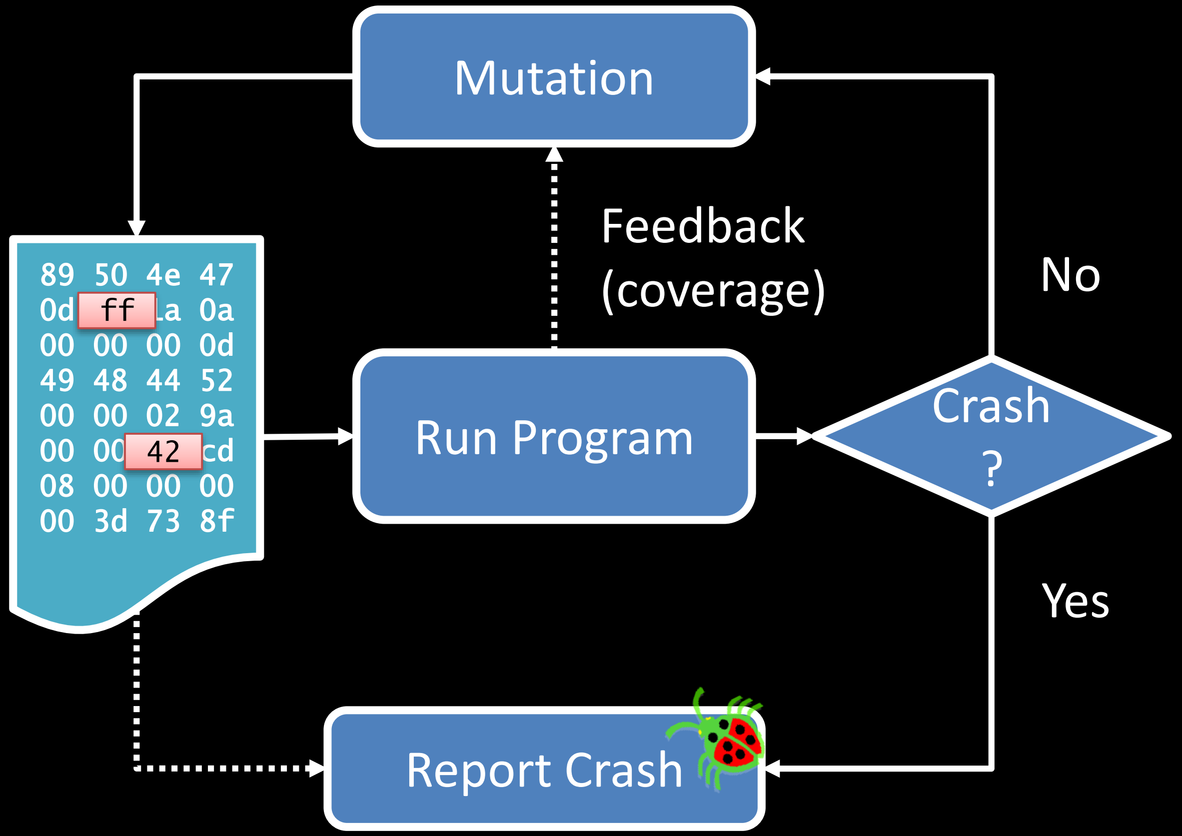
- Removing SDK from enclave memory?
  - Not feasible as OCALL, ECALL, AEX require the tRTS
- Randomizing SDK code?
  - Challenging, the tRTS is accessed through fixed entry points
- Discovering vulnerabilities beforehand?
  - Last part of this talk: research on fuzzing and symbolic execution

Background: Bug Discovery Techniques

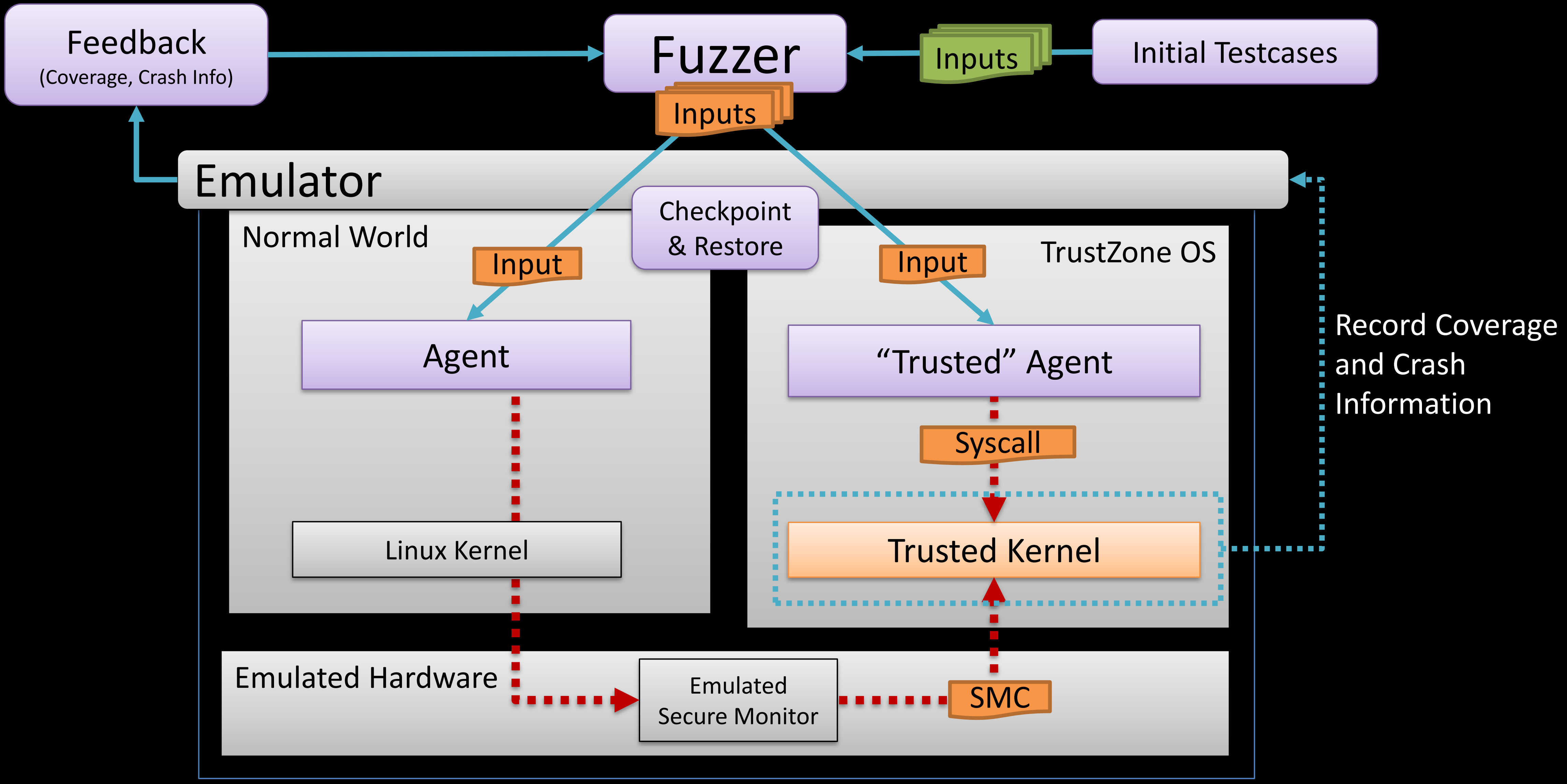
- **Symbolic Execution**
  - Emulate the program based on encoding the program state as symbolic variables
  - Utilize solver to find feasible crashing paths



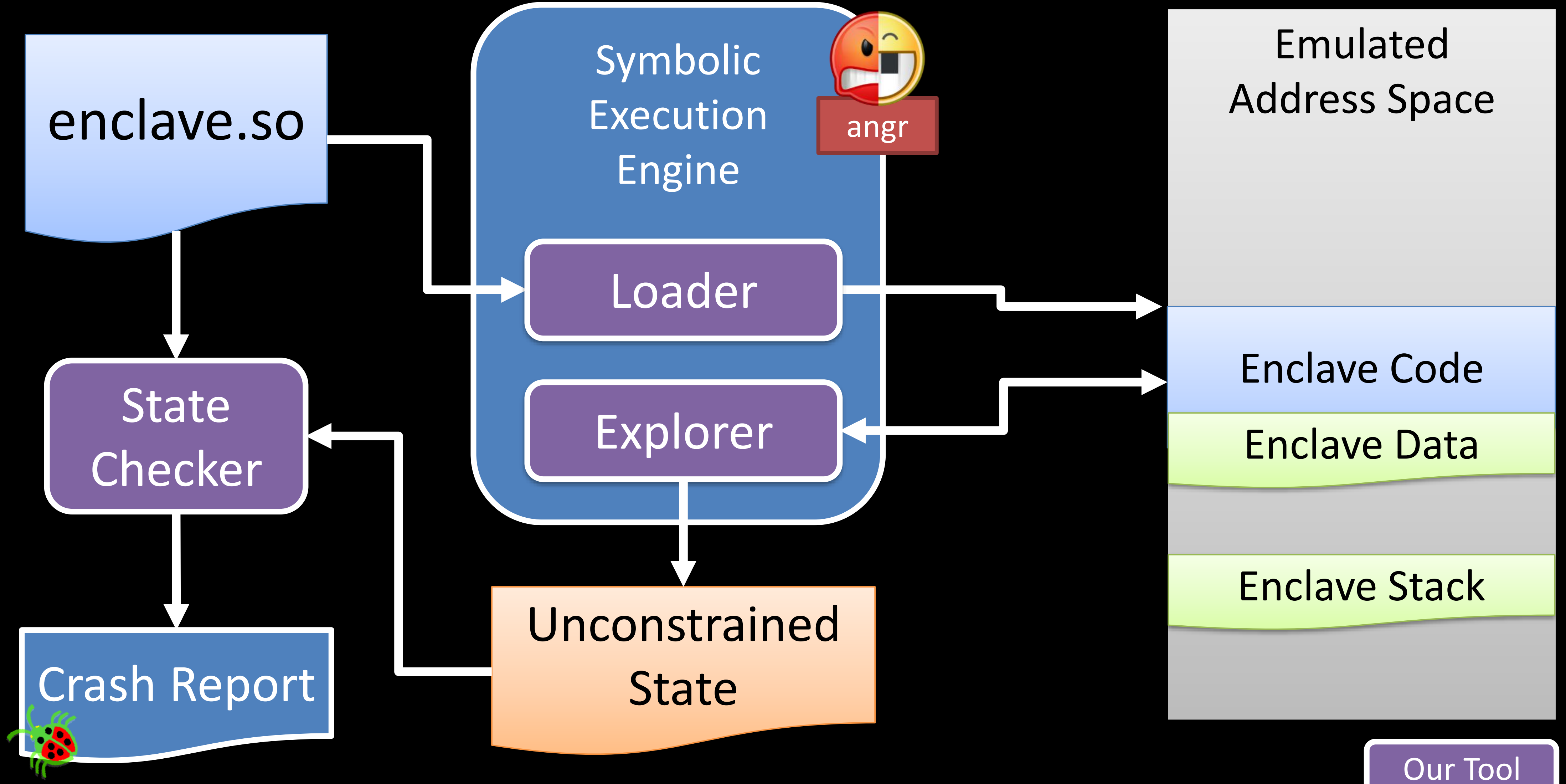
- **Fuzzing**
  - Probabilistically explore program paths
  - Find new inputs with random mutation



TrustZone OS Fuzzing



Symbolic Execution of SGX Enclaves



Harware-assisted application security is vital to implement trustworthy systems and enhanced security services → control-flow attestation

However, we need to make sure that an attacker cannot exploit bugs inside the TEE → return-oriented programming

Hence, research on bug finding in TEE code is crucial → fuzzing, symbolic execution