ETH zürich

ROTE: Rollback Protection for Trusted Execution

Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, Srdjan Capkun

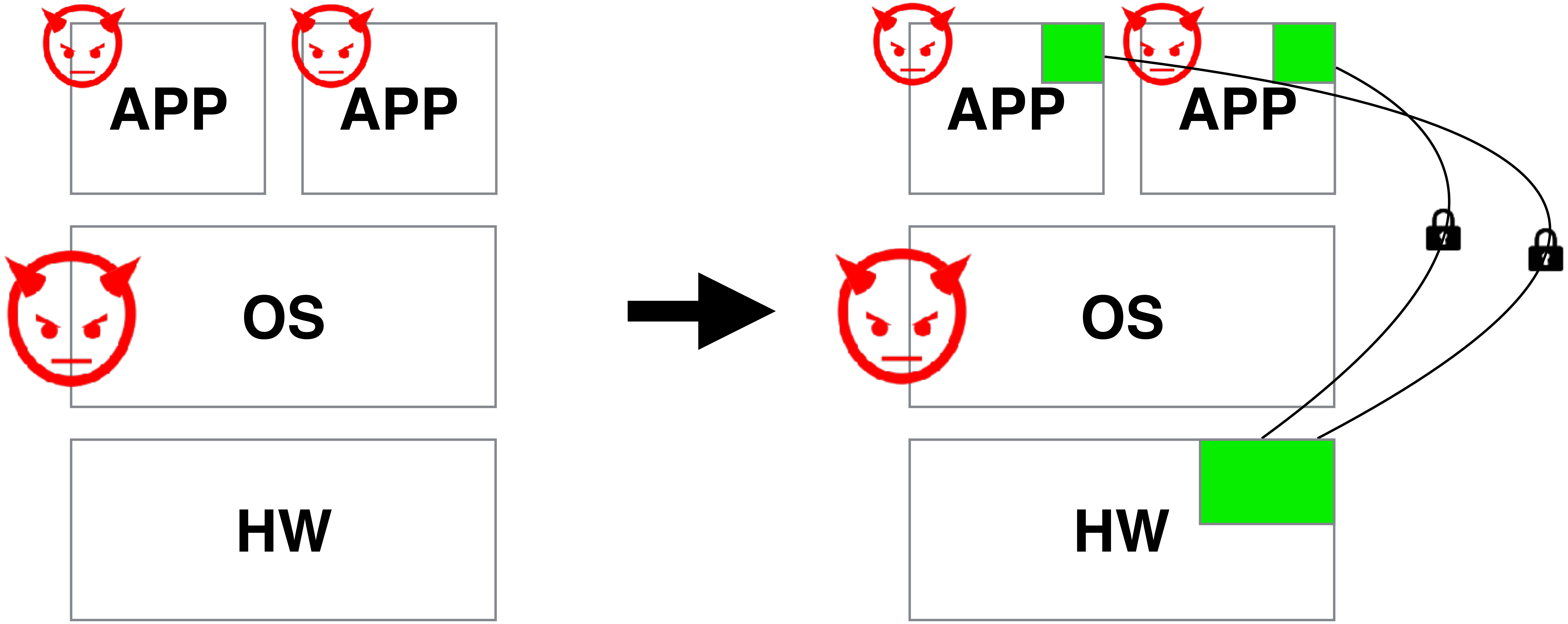
Siniša Matetić

ETH Zurich
Institute of Information Security

August 18th, 2017

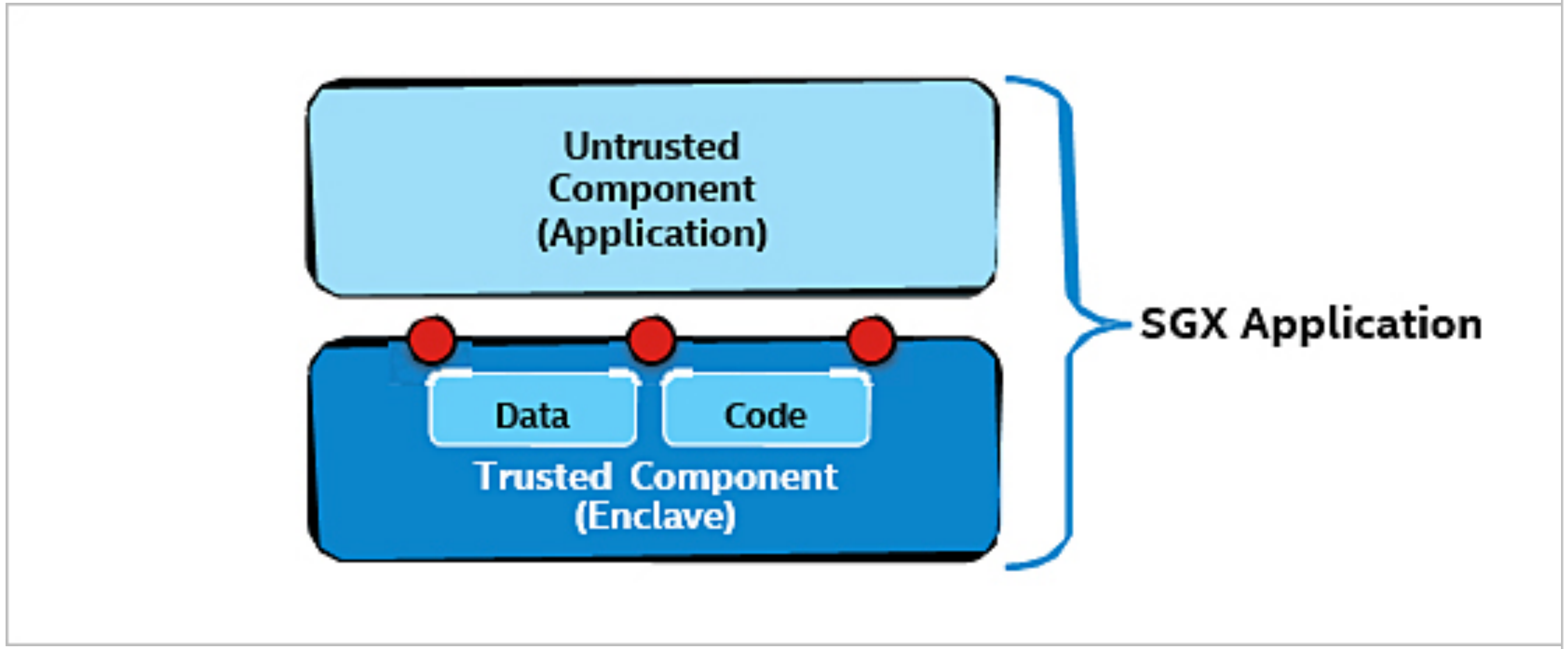
Introduction

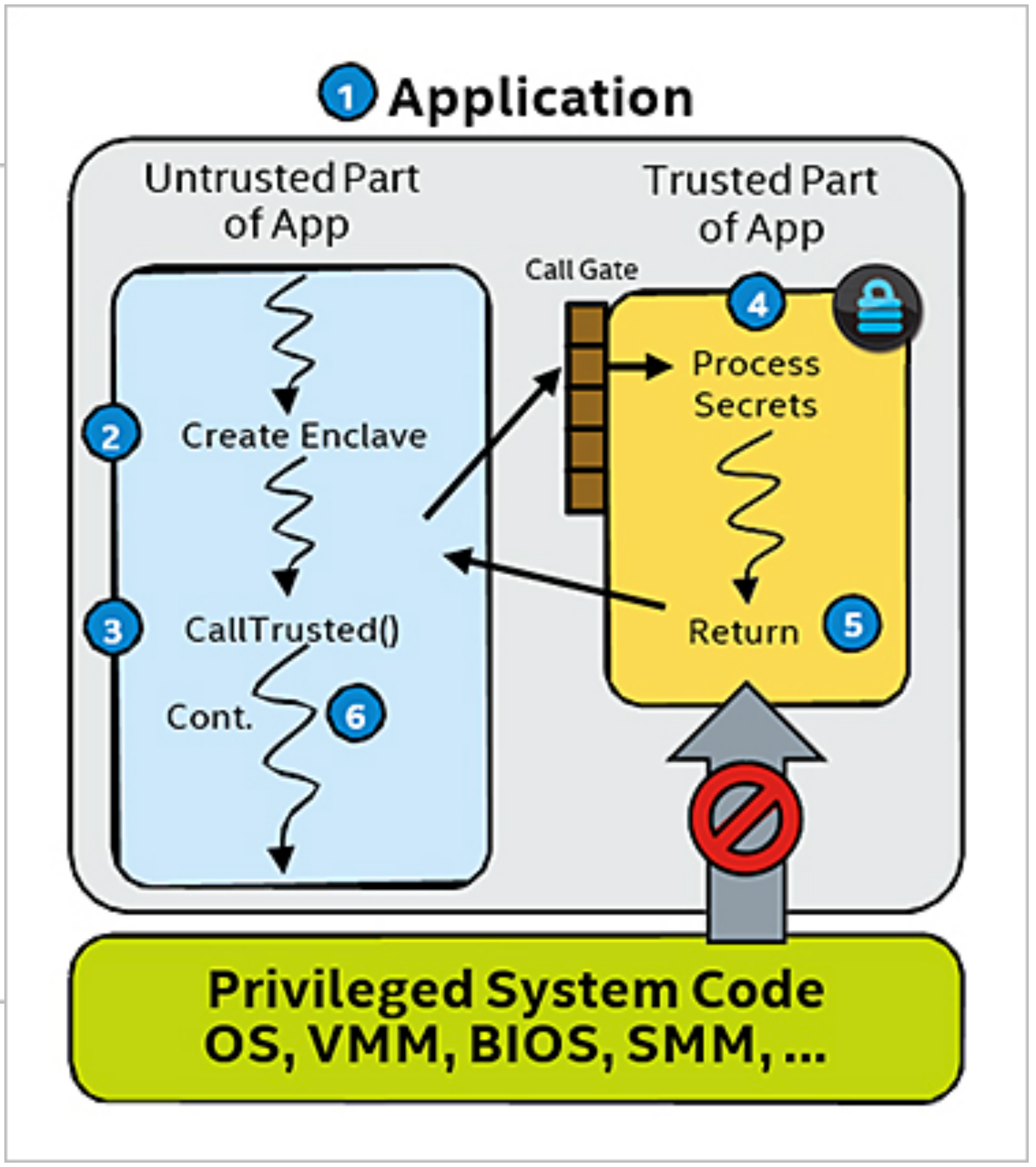
- Intel Software Guard Extensions (SGX)
  - Intel's new architecture containing new instructions and protective mechanism in the processor
- Regular systems are vulnerable to various attacks



Introduction

- Enables trusted execution of security-critical application code
  - SGX enclaves
- Isolation from the untrusted system software, other enclaves and peripherals
- Security perimeter is the processor itself

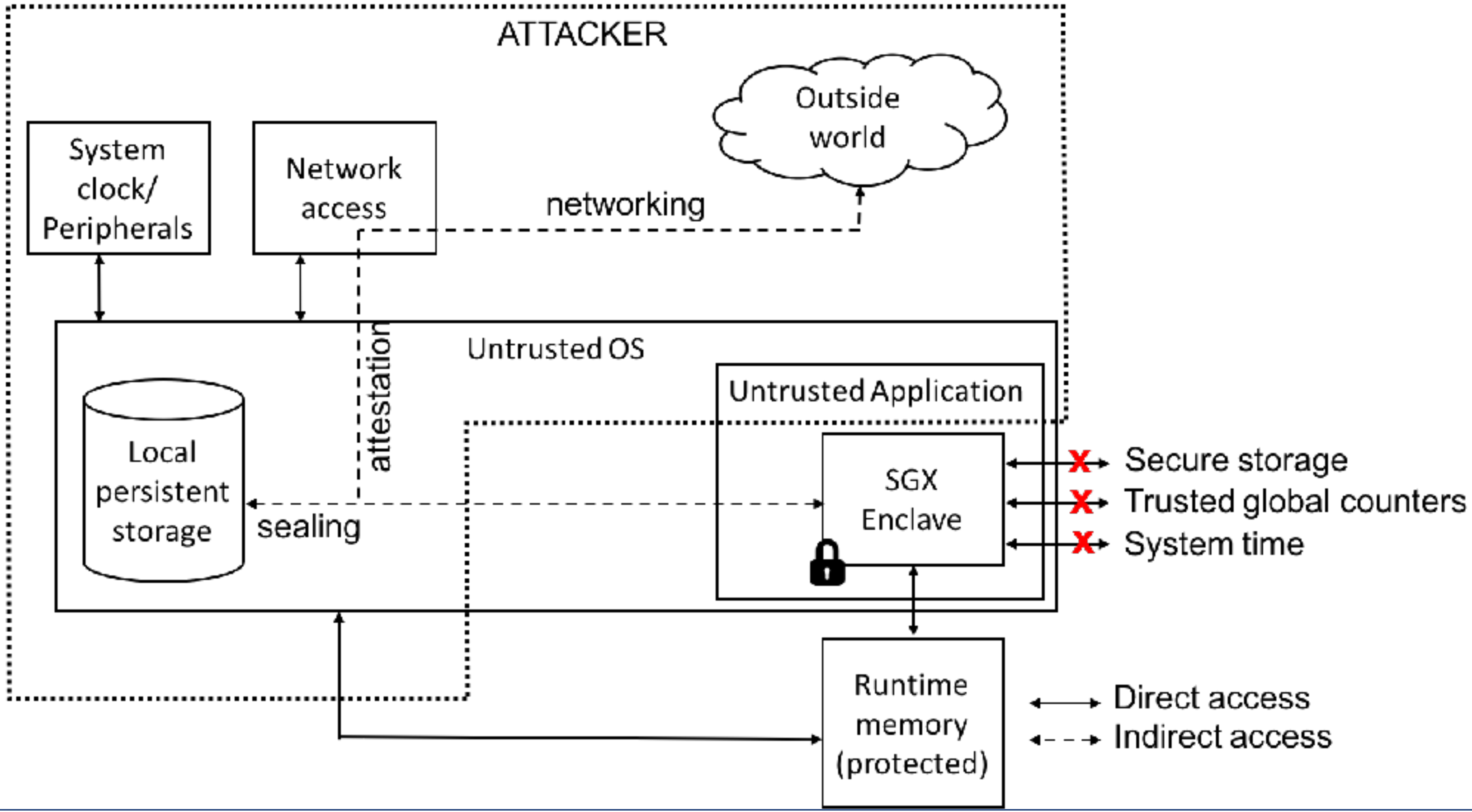




Images taken from software.intel.com

Introduction

- **Sealing** - storing data for persistent storage across executions which gives confidentiality and authentication
  - but what about **integrity?**
- Processors are equipped with certified cryptographic keys
  - enables remotely verifiable **attestation** statements



+ and -

+ Isolates execution, can handle untrusted OS
+ Can run many enclaves in parallel
+ Supports attestation
+ Supports sealing
+ Unlike with TPM, security boundary is the processor

- It is not system-wide (unlike TrustZone)

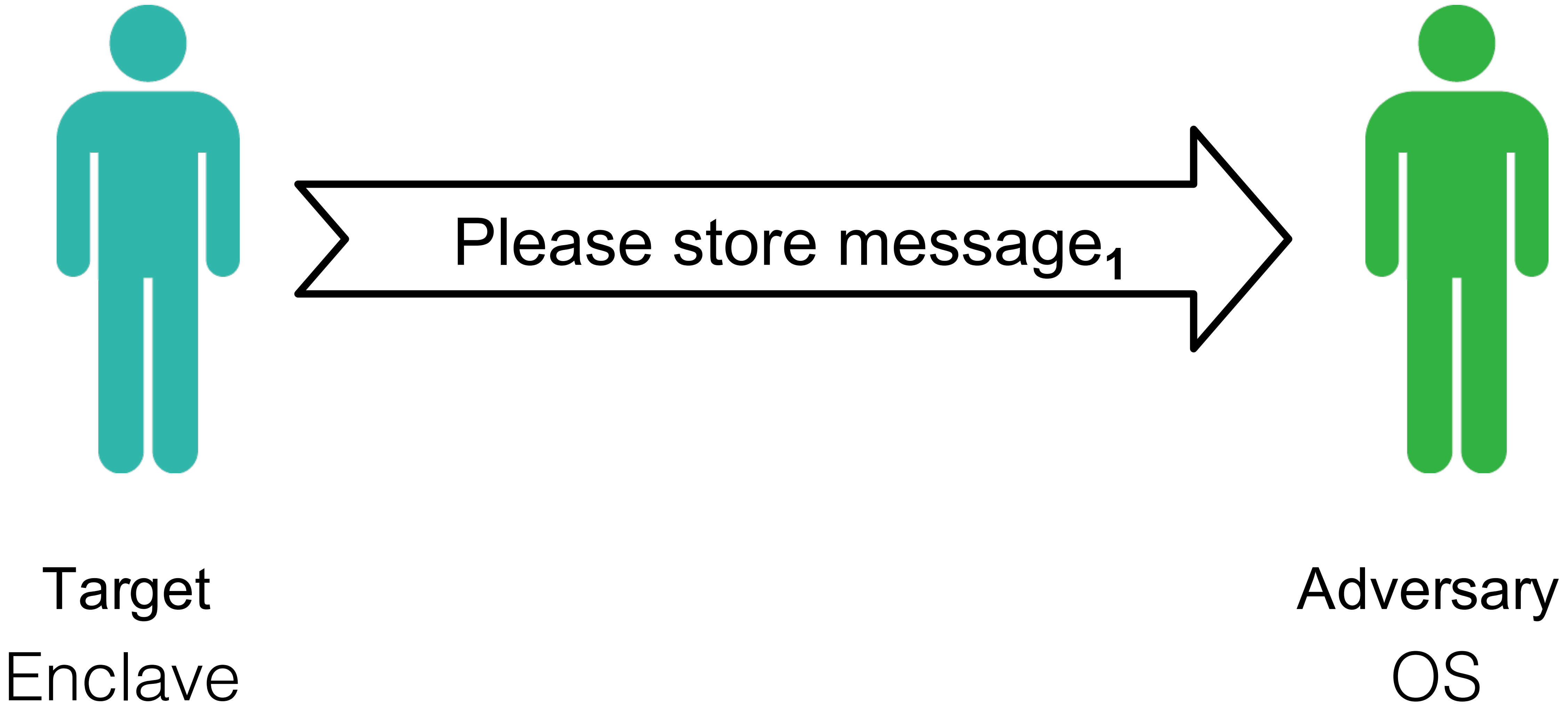
- No direct access to peripherals

- Side Channels [many recent works]

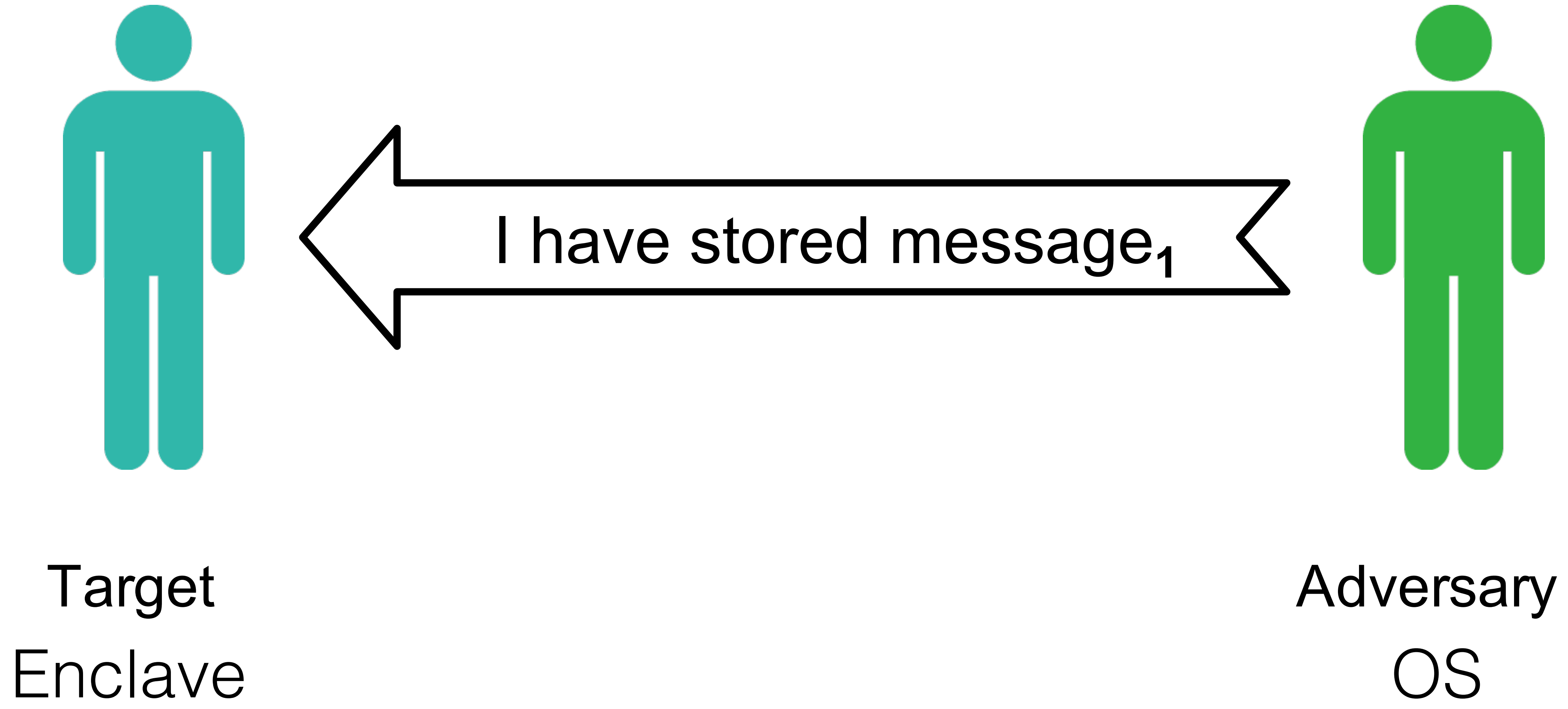
- No Rollback Protection

ETH zürich

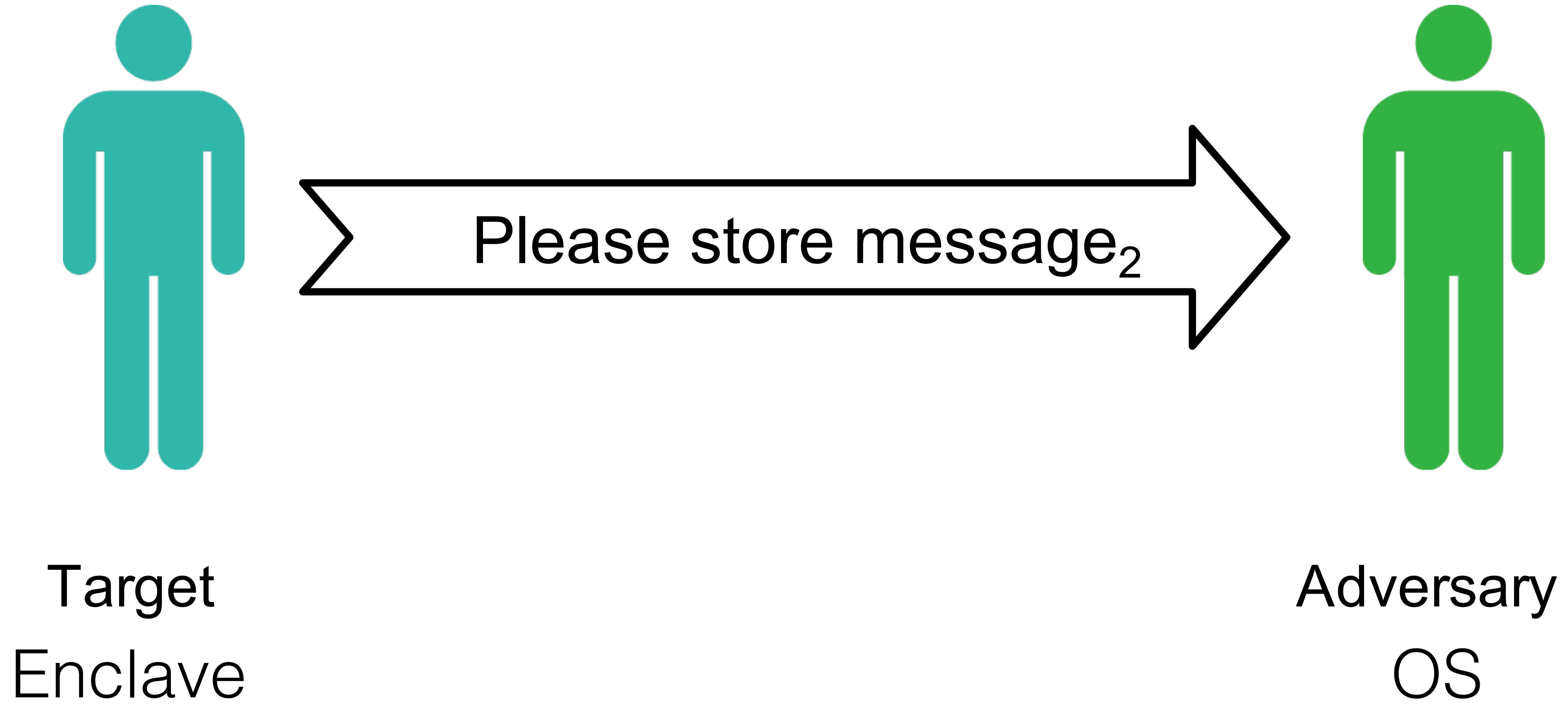
Example



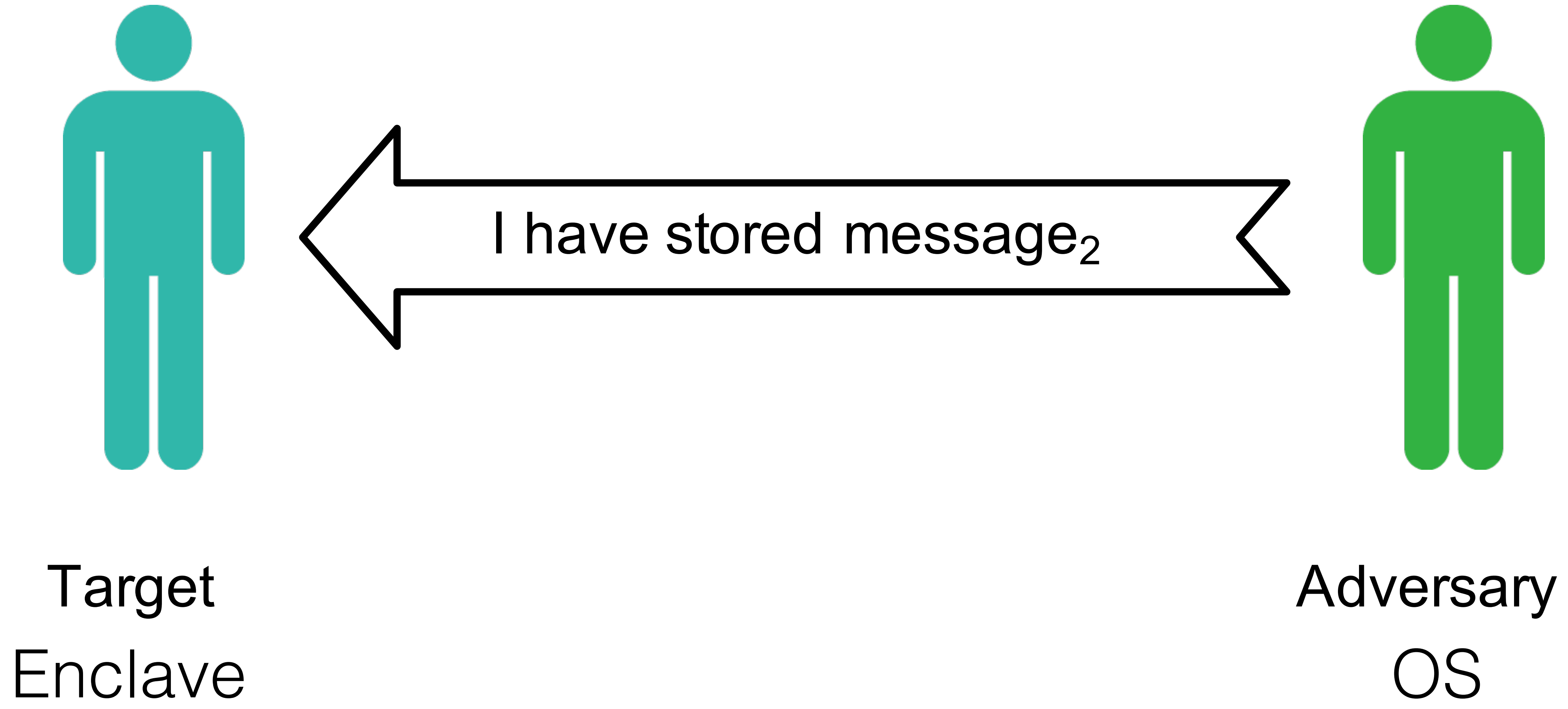
Example



Example



Example



Example

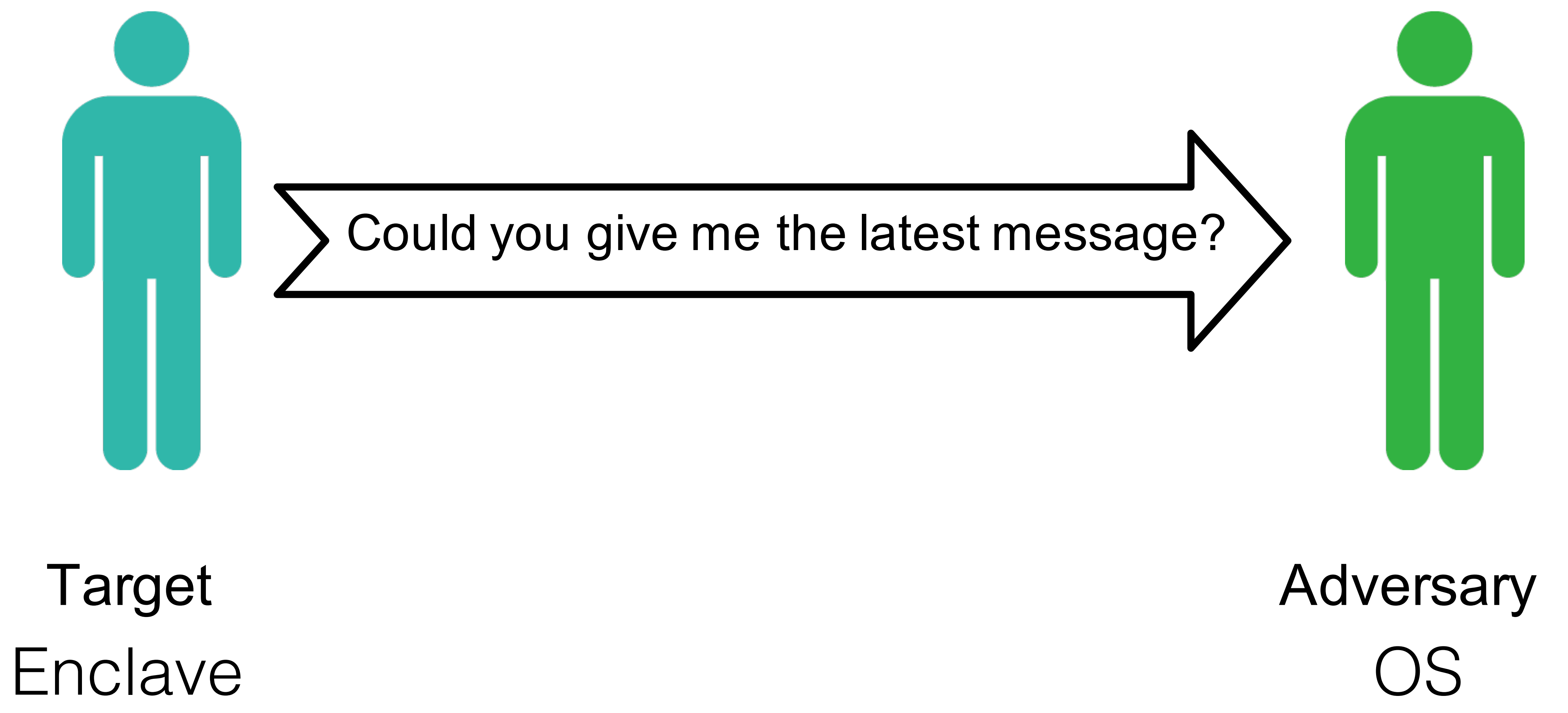
Target

Enclave

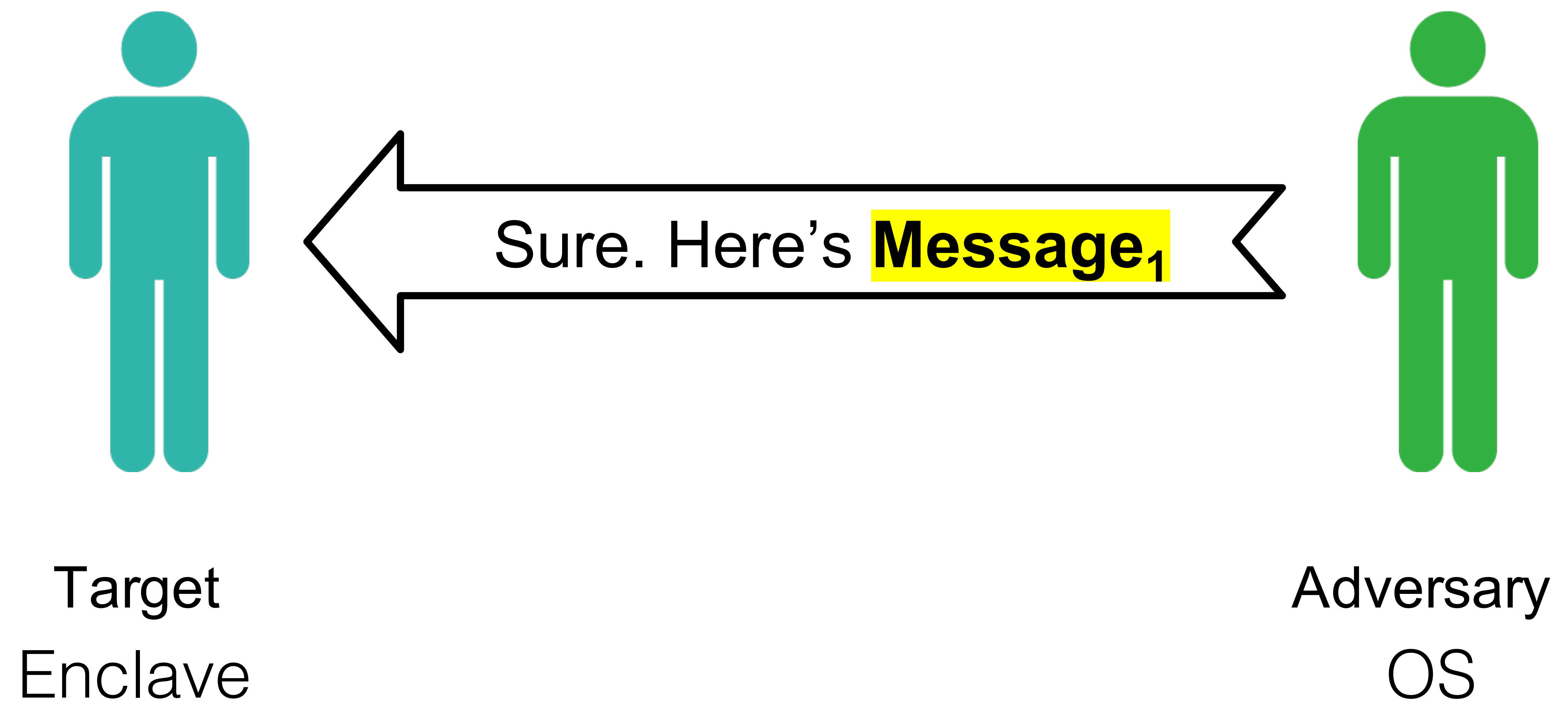
Adversary

OS

Example


Could you give me the latest message?
Target
Enclave
Adversary
OS

Example



Intel SGX - Protecting the Local State?

- **In a rollback attack a malicious OS replaces the latest sealed data with an older encrypted and authenticated version**
- Another way to violate state integrity is to create two instances of the same enclave and route update requests to one instance and read requests to the other **(restart, terminate, …)**.
- Enclaves cannot detect replay, because the processor **does not hold persistent state** across enclave executions (and platform reboots)

Example scenario

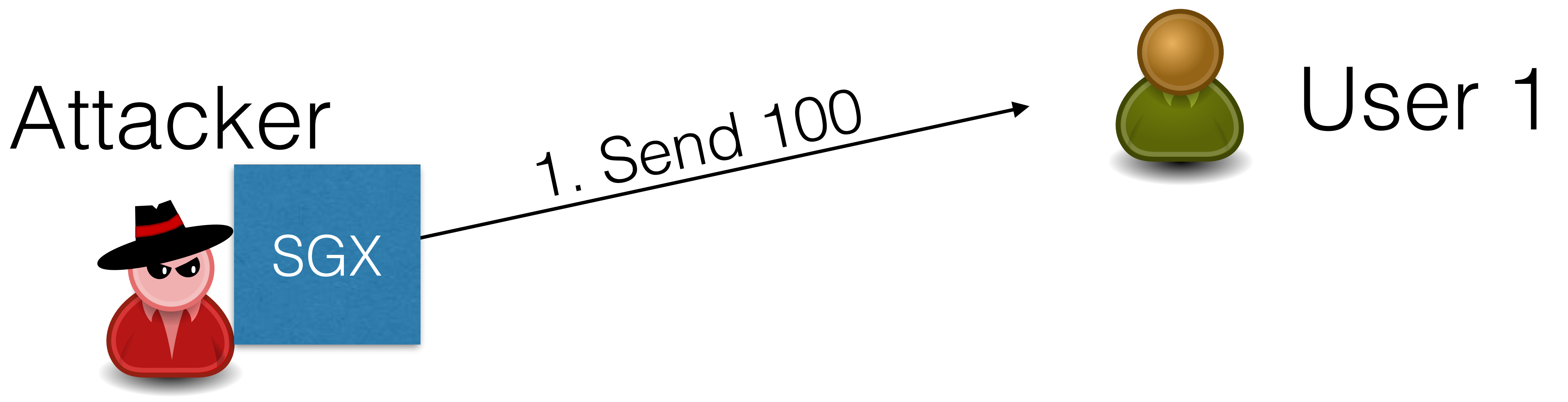
- Imagine a financial application where account balance is enforced by SGX

State 1: Initial bank account balance: 300

Example scenario

- Imagine a financial application where account balance is enforced by SGX

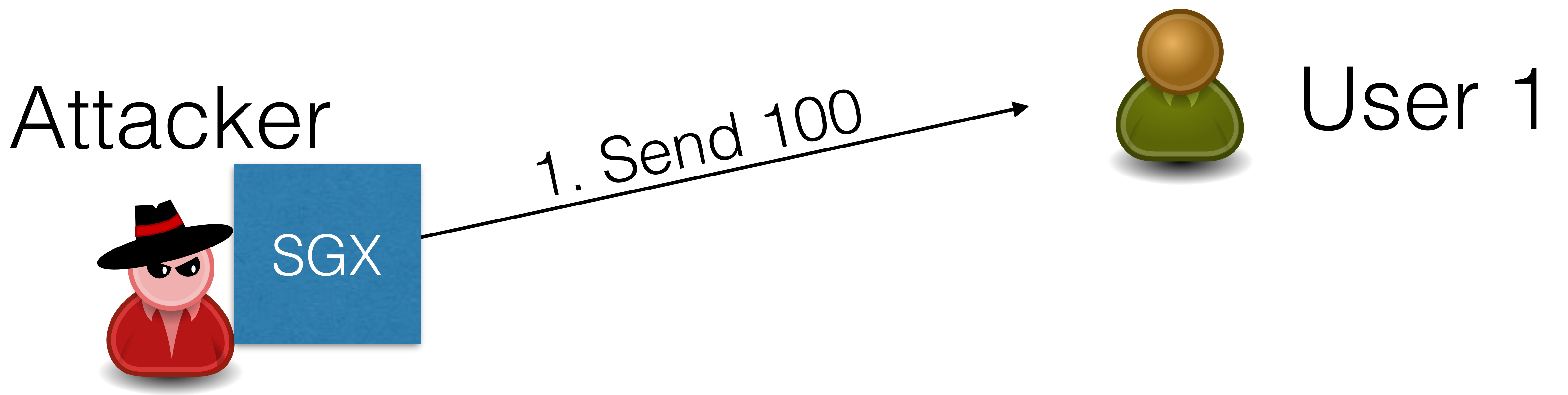
State 1: Initial bank account balance: 300



Example scenario

- Imagine a financial application where account balance is enforced by SGX

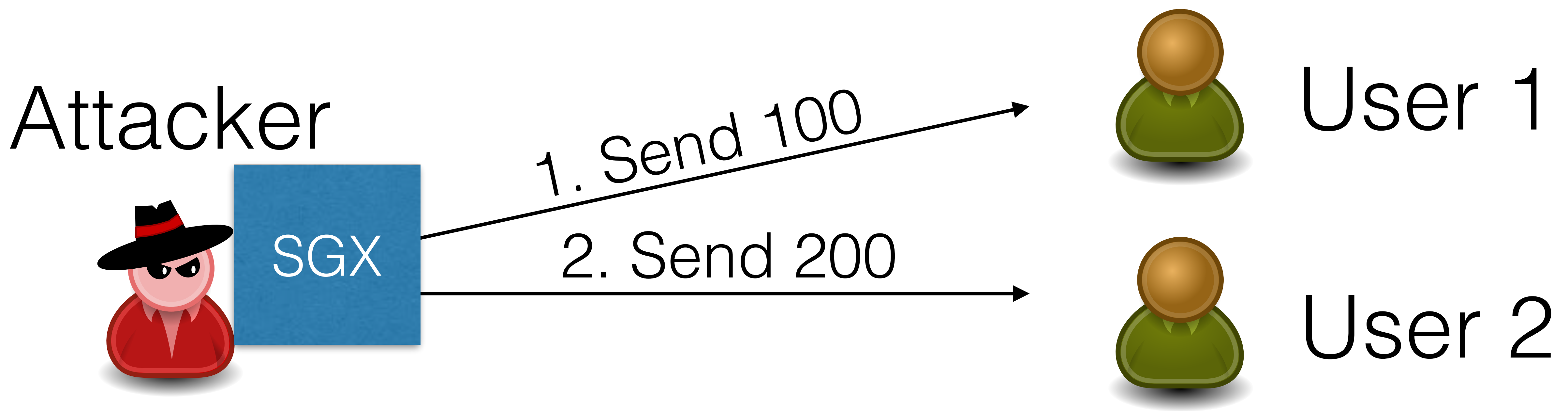
State 2: Initial bank account balance: 200



Example scenario

- Imagine a financial application where account balance is enforced by SGX

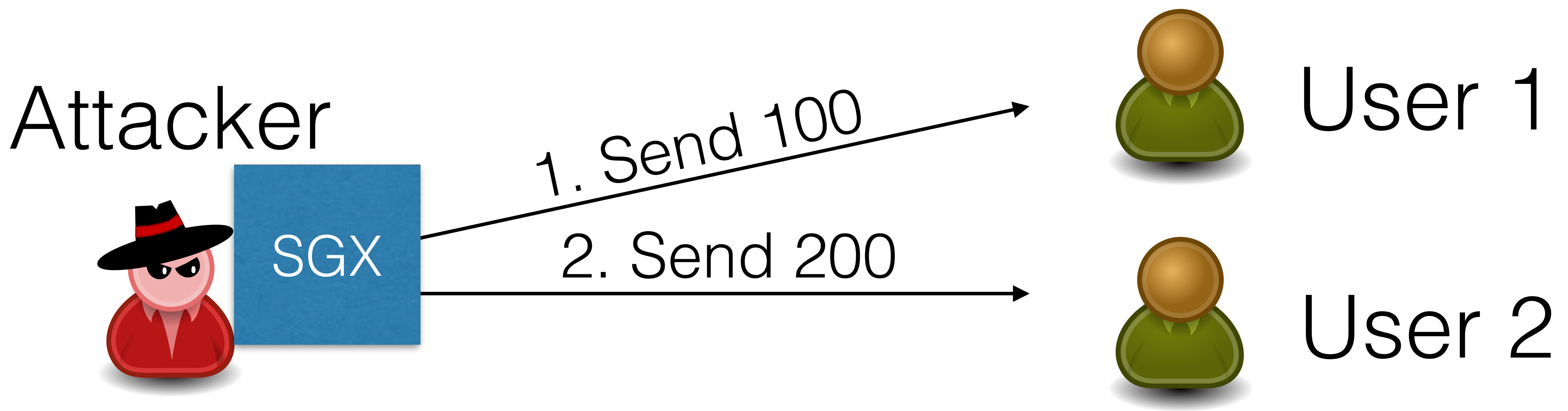
State 2: Initial bank account balance: 200



Example scenario

- Imagine a financial application where account balance is enforced by SGX

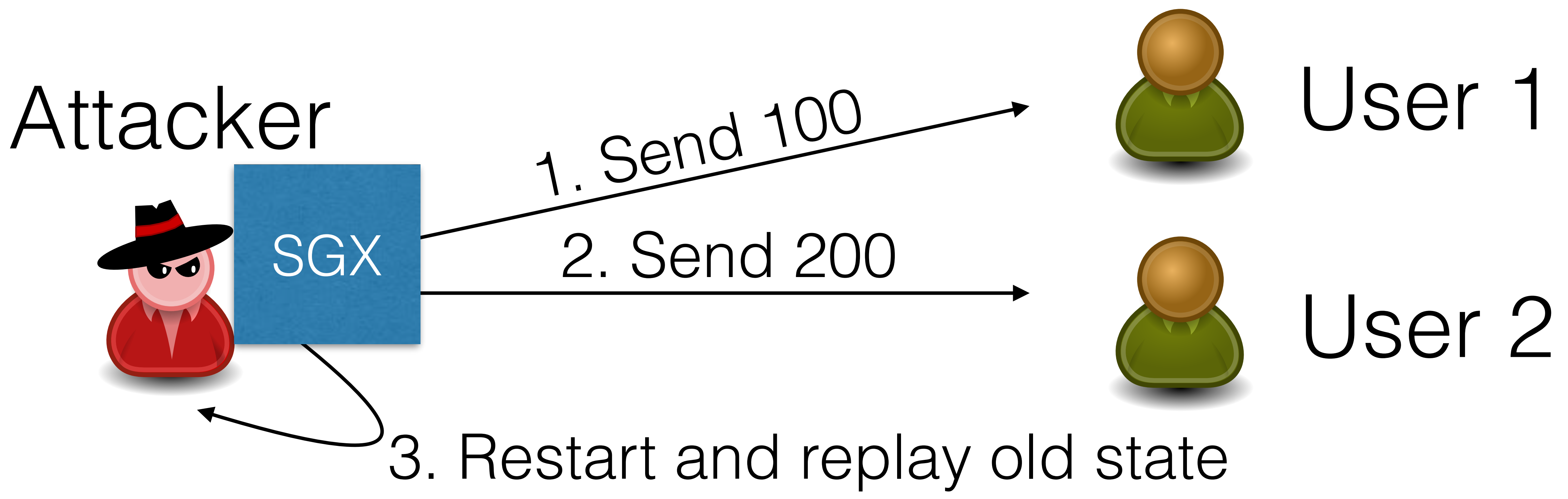
State 3: Initial bank account balance: 0



Example scenario

- Imagine a financial application where account balance is enforced by SGX

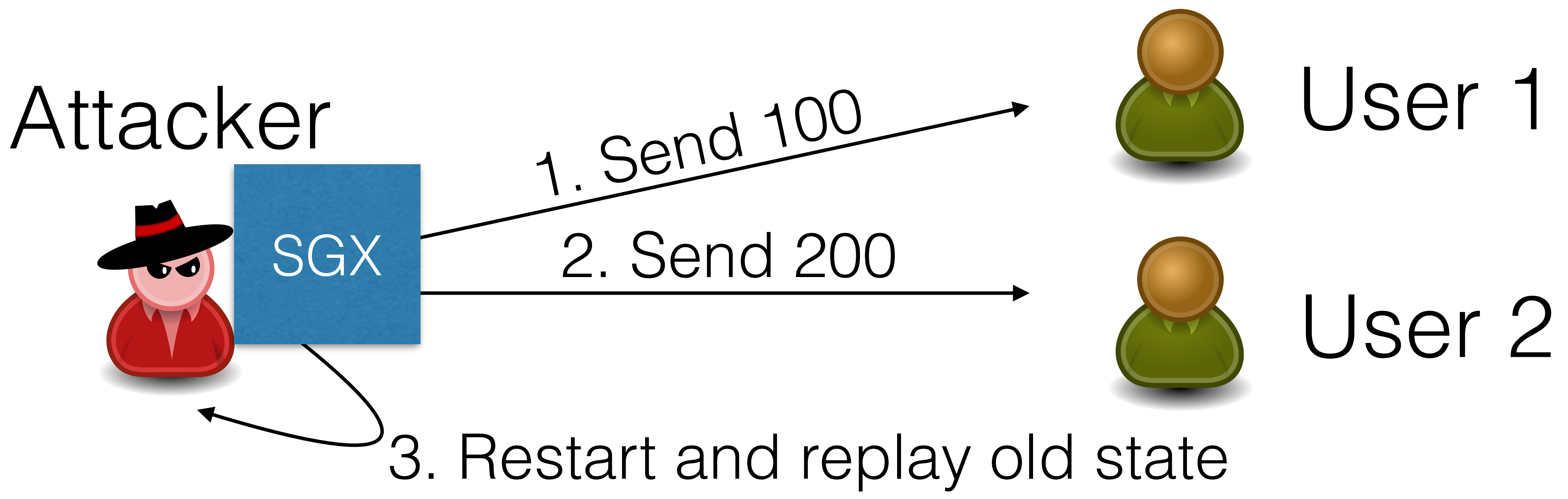
State 3: Initial bank account balance: 0



Example scenario

- Imagine a financial application where account balance is enforced by SGX

State 1: Initial bank account balance: 300

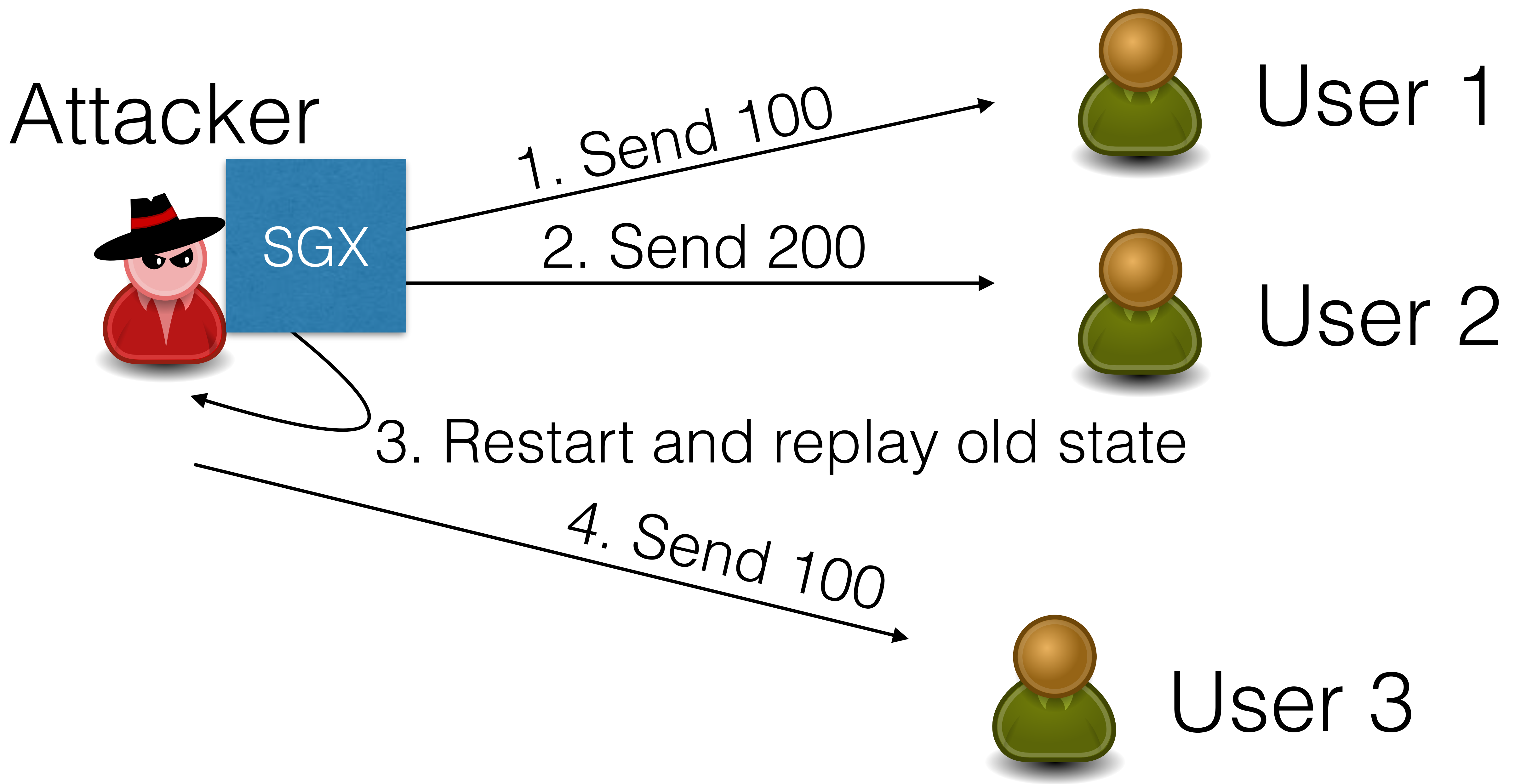


ETH zürich

Example scenario

- Imagine a financial application where account balance is enforced by SGX

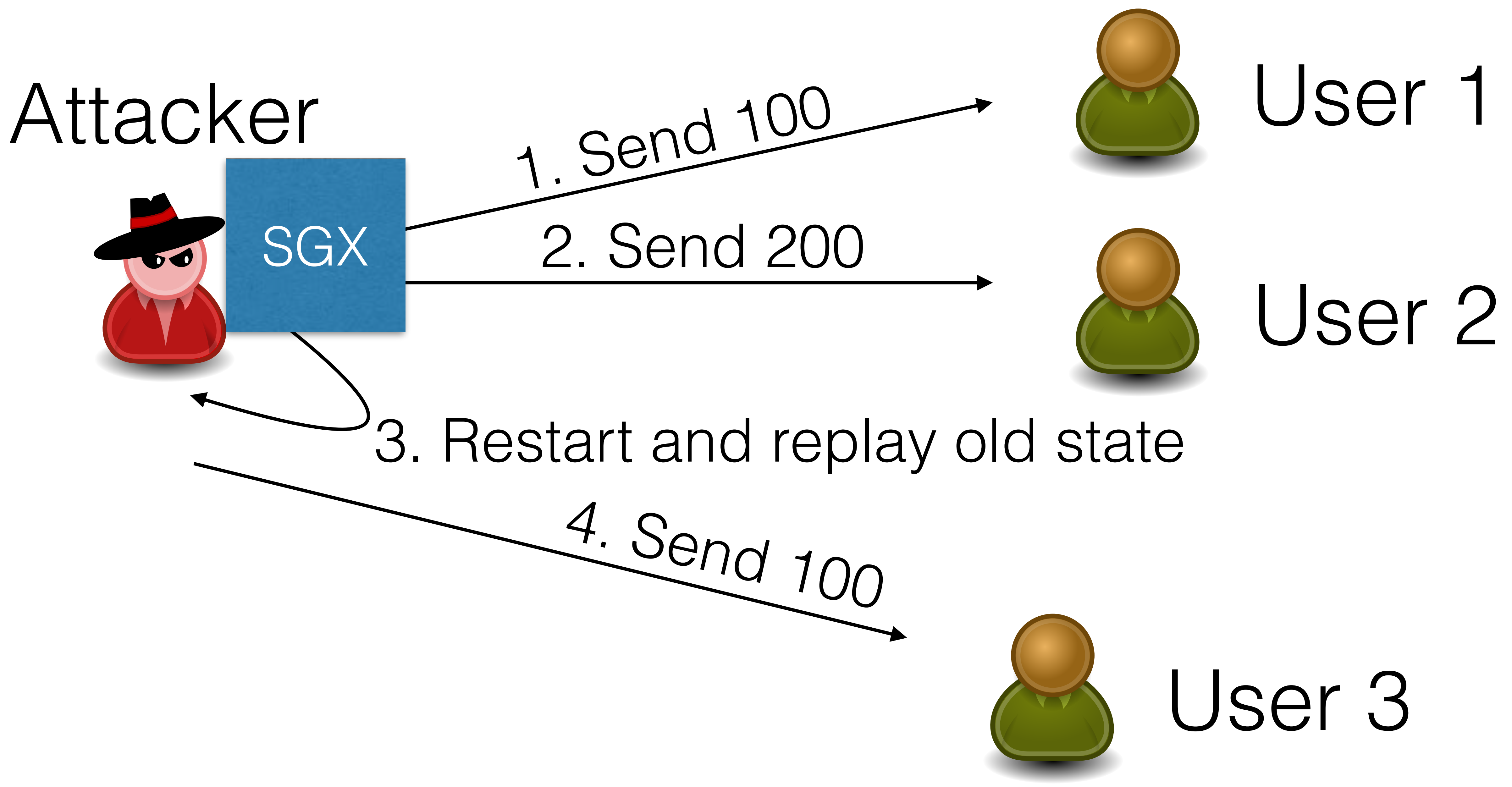
State 1: Initial bank account balance: 300



Example scenario

- Imagine a financial application where account balance is enforced by SGX

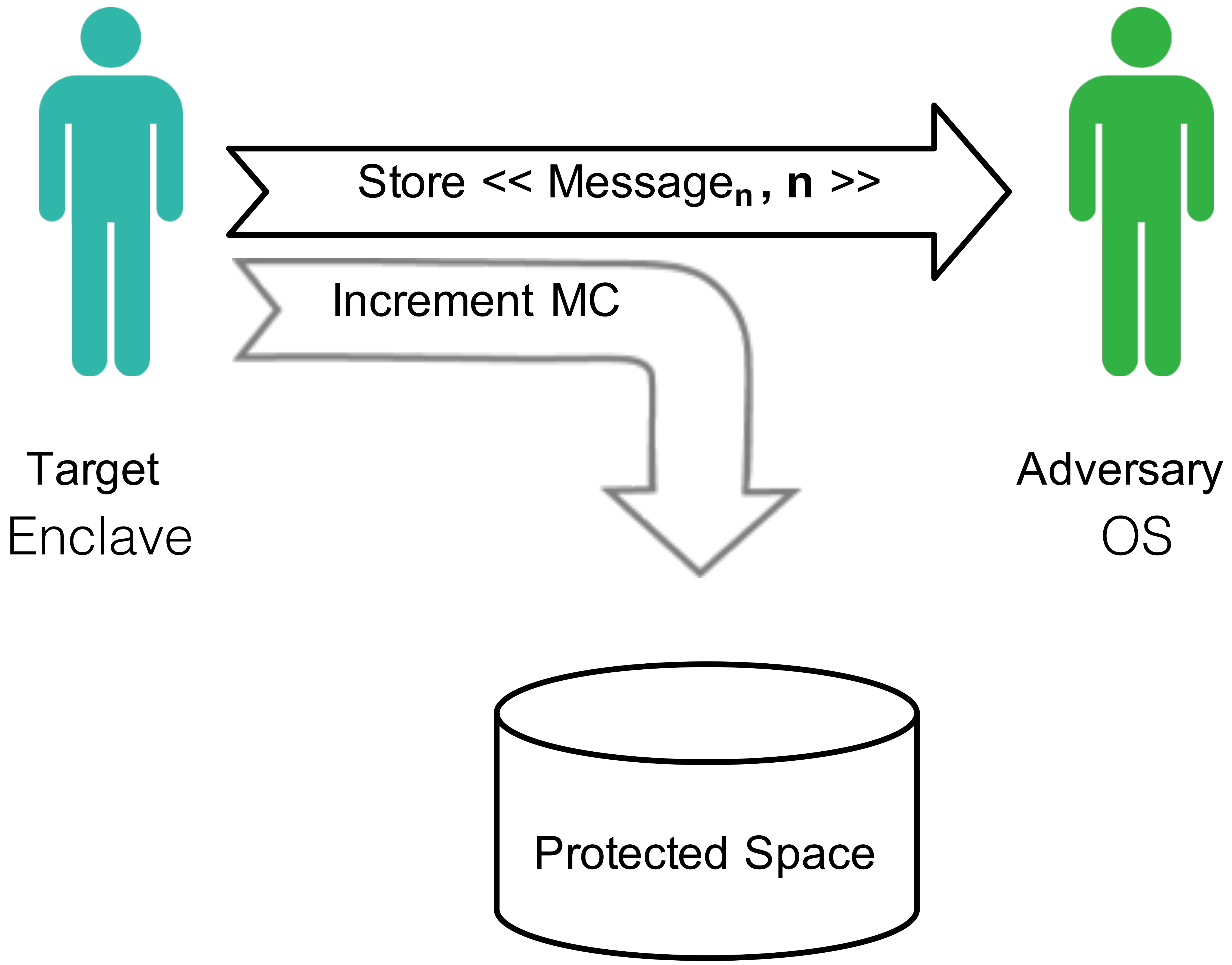
State 2: Initial bank account balance: 200



Main contributions of this work

- New security model for reasoning about the integrity and freshness of SGX applications
  - identified security weaknesses in existing SGX systems.
- SGX counter experiments showing limitation of the service
- Novel approach of realising rollback protection by storing enclave-specific counters in a distributed system
- Implemented ROTE system that ensures integrity and freshness of application data in a powerful adversarial model.
- Experimental evaluation showing only a small performance overhead for our system
  - in a low-latency network state update overhead is only 1-2 ms

Example solution

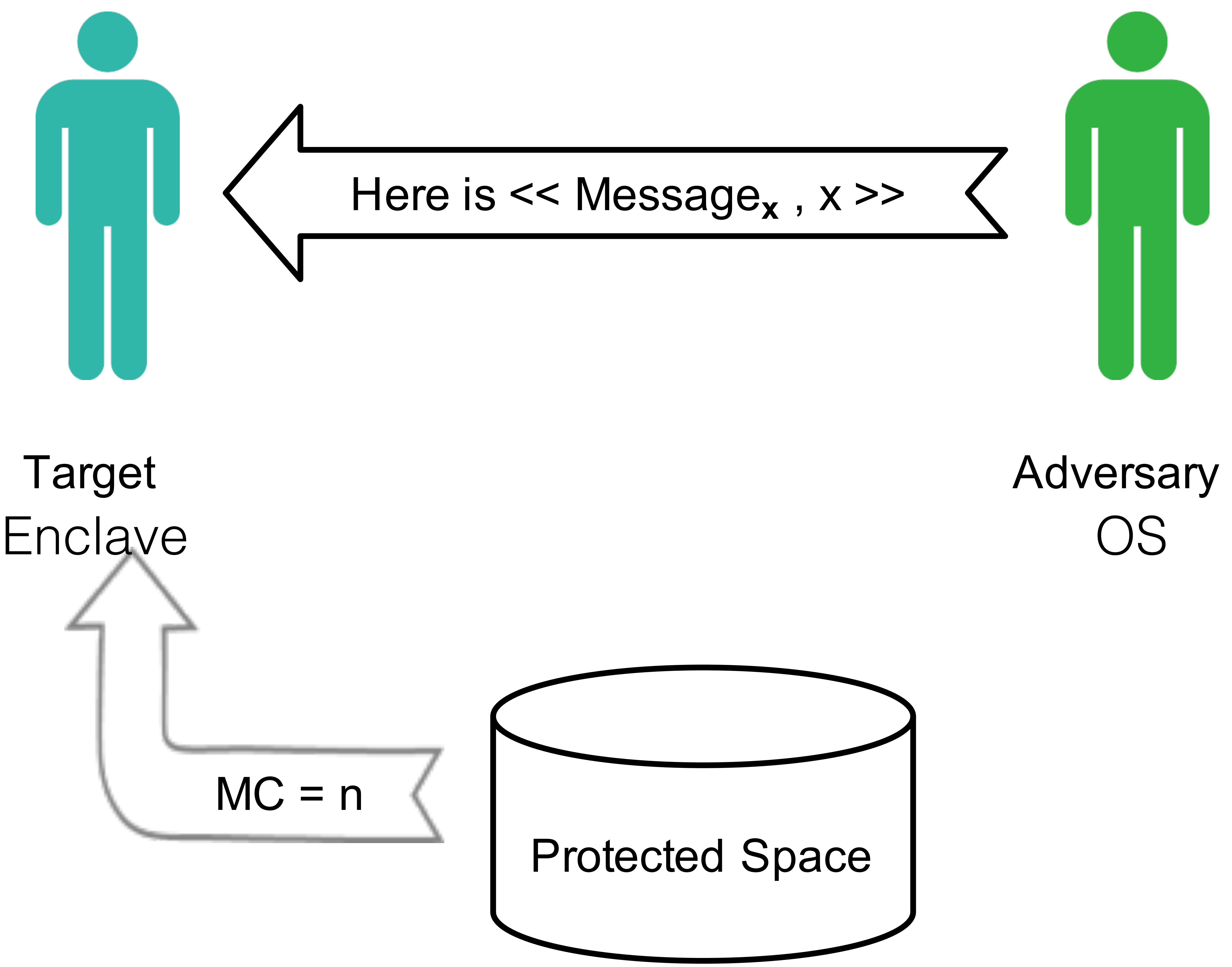


Example solution

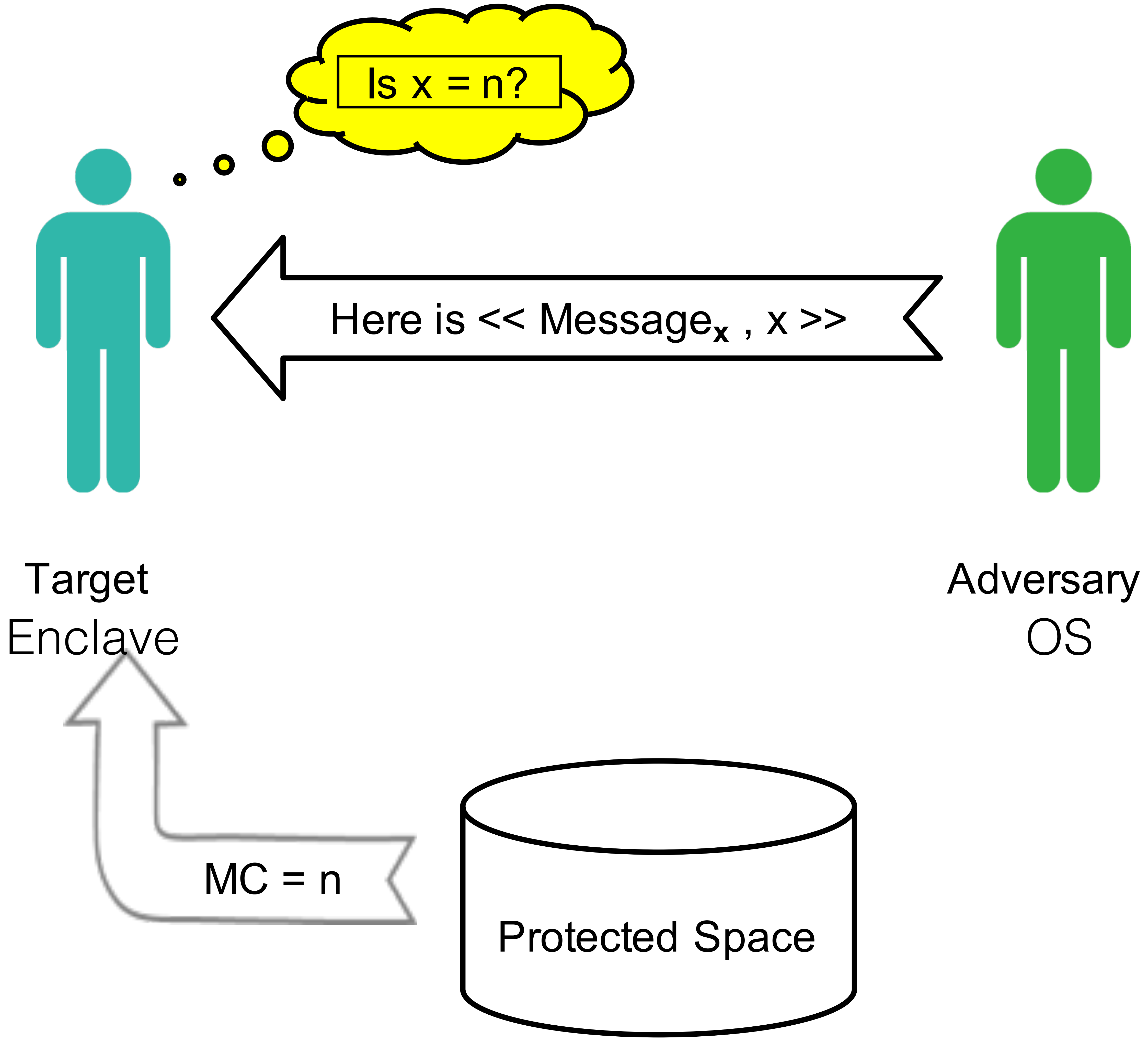
Target
Enclave

Adversary
OS

Example solution



Example solution



Existing solutions

- To address rollback attacks, two basic approaches are known:
  - use non-volatile memory element to store the state
  - maintain integrity information in a separate trusted server
- SGX supports Monotonic Counter service
  - limited security guarantees
  - poor performance (limits high-throughput transactions)
- Leveraging Trusted Platform Modules (TPMs)
  - similar limitations

Solution provided by Intel SGX

- **SGX supports Monotonic Counter service**
  - Stored in an off-CPU memory
  - Security concern: **counters stored in a flash memory that is also used by the BIOS, connected via an SPI bus.**
    This is a passive component.
  - Performance concerns: how practical is this?

Experiments - SGX counter service

- Counter increment operation took **80 - 250 ms** (model dependent). Counter read operations took **60-140 ms**
- **1.05 M** writes render the NV counter (memory) unusable (wear)
- Reinstalling the SGX Platform Software (PSW) or removing the BIOS battery deletes all counters
- After reinstalling the PSW the platform software connects to Intel server. If connection not available, the counter service is unavailable
- Updates of an enclave every 250 ms => counters become unusable in few days.
  - with one increment per minute, the counters are exhausted in two years

SGX: System / Attacker Model

- Attacker:
  - enclave scheduling,
  - platform reboots,
  - control of the full software stack,
  - control over the complete communication channel, and
  - compromising the SGX hardware

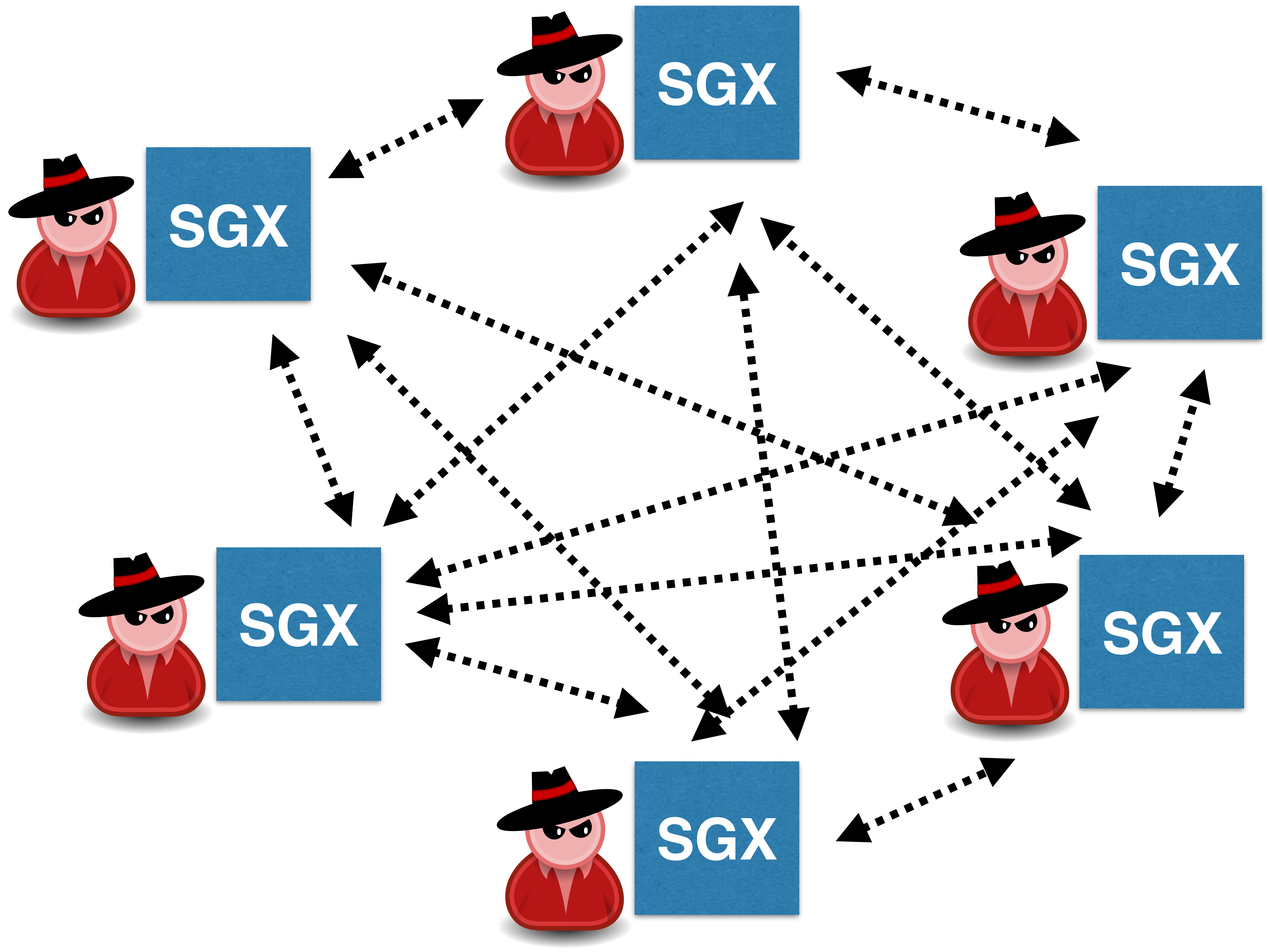
- One can achieve **all-or-nothing** rollback - the only way to violate data integrity is to reset the entire group to its initial state

Our Approach

- Intuition: A single platform cannot efficiently prevent rollback, but in many practical scenarios, multiple processors can be enrolled to assist each other

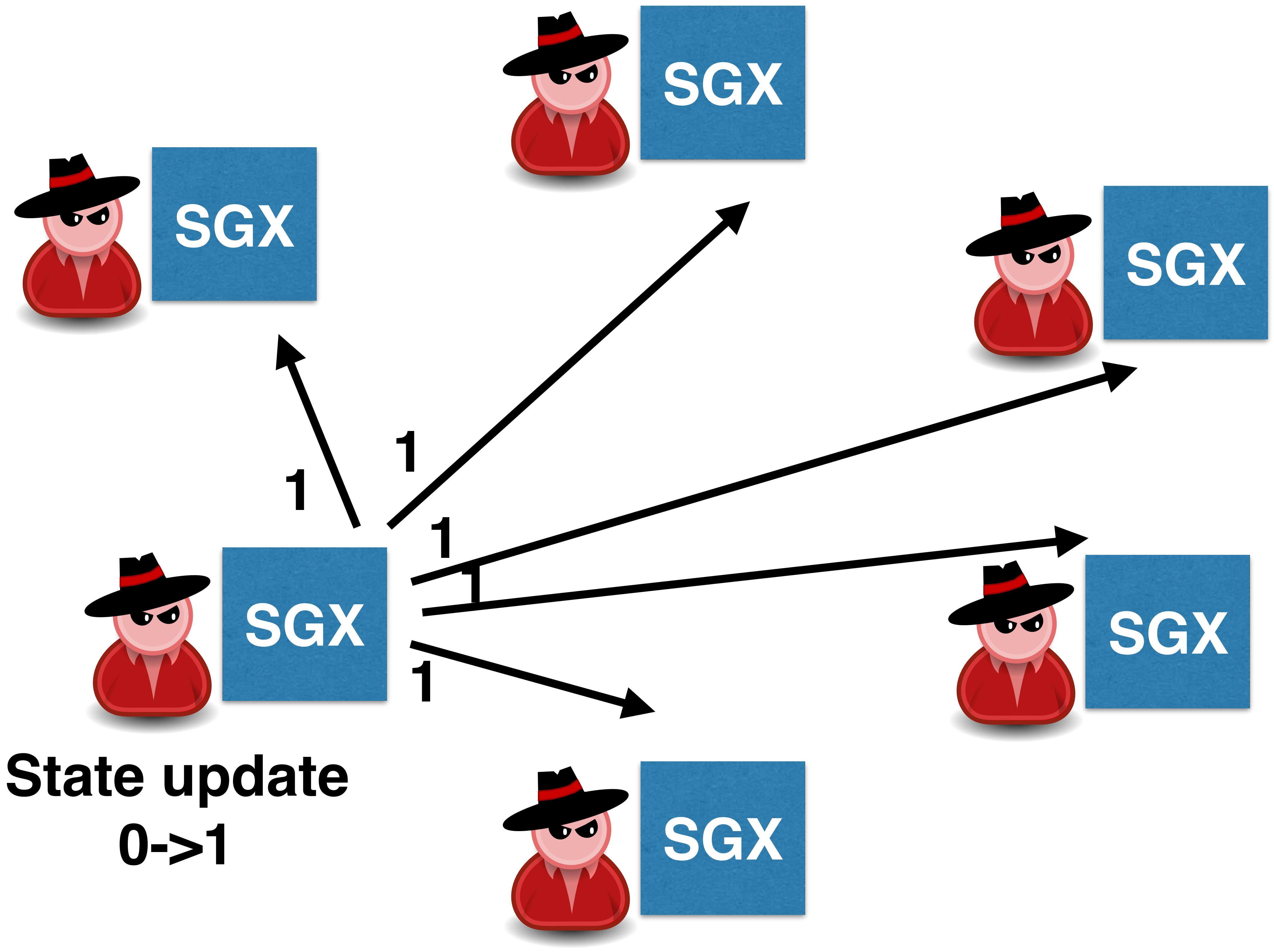
Our Approach

- We try to build a distributed system where each participating nodes provides state protection for all other nodes



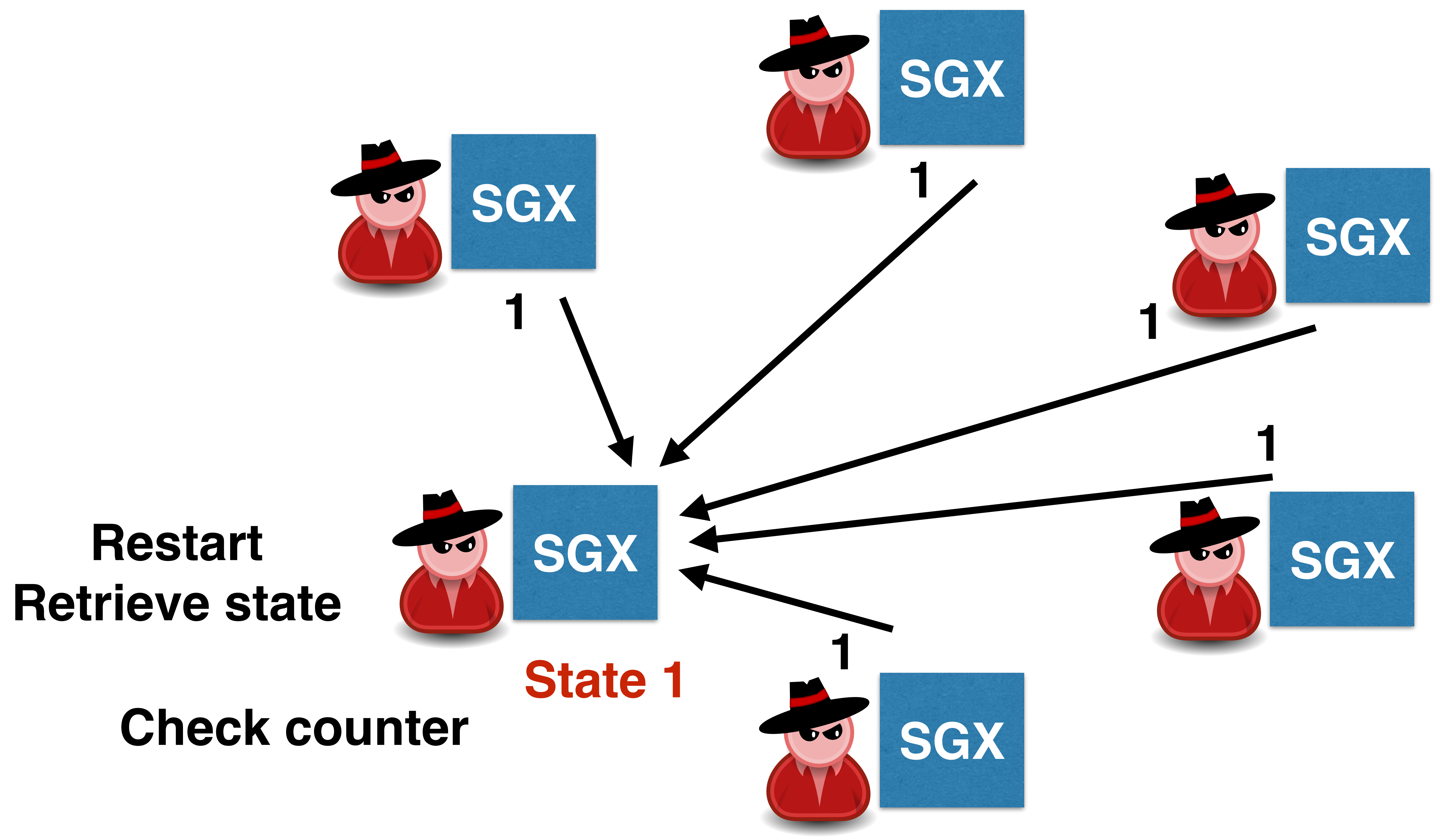
Our Approach

- When an enclave updates its state, it stores a counter to a set of enclaves running on assisting processors



Our Approach

- When the enclave needs **to recover** its state, it obtains counter values from assisting enclaves to verify that the recovered state data is of the latest version

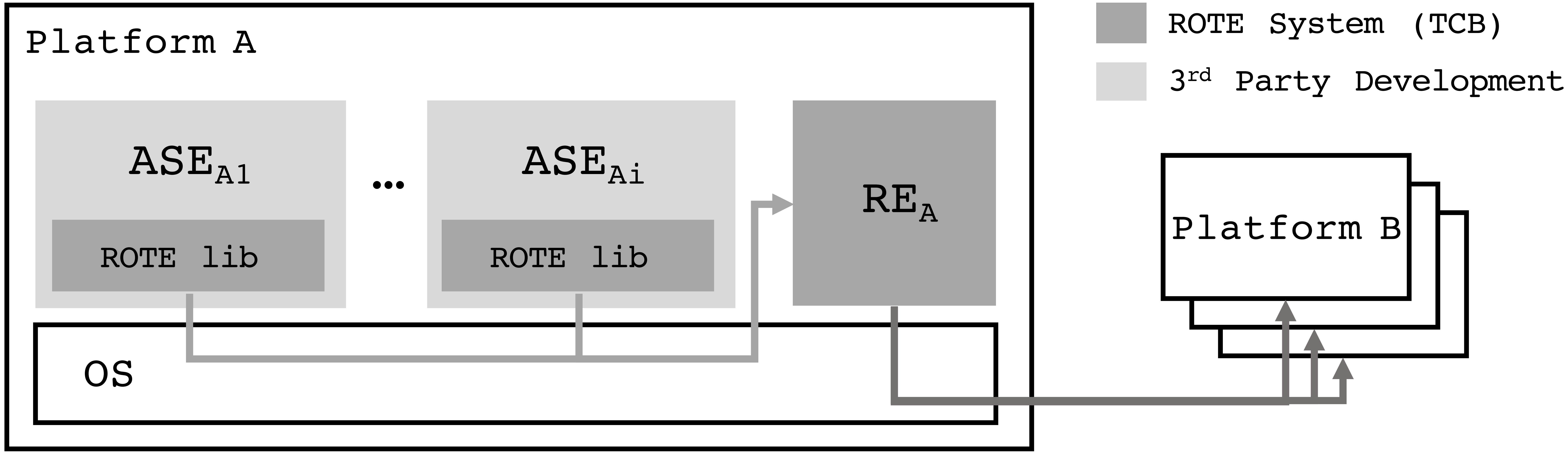


Challenges

1. Protection against the strong adversary model
2. Network partitioning
3. Coordinated enclave restarts
4. Multiple enclave instances
5. Group establishment

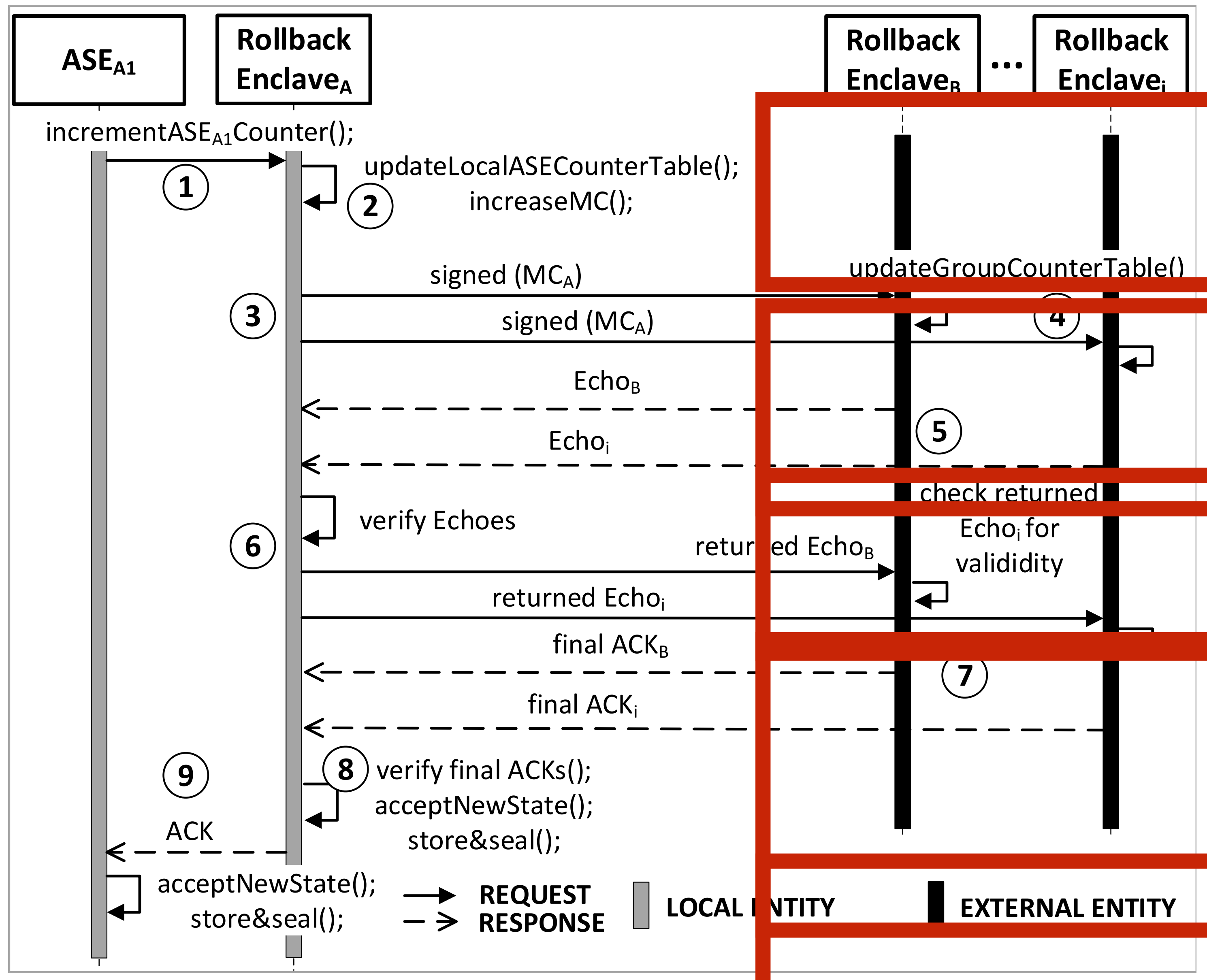
System architecture

- Multiple user applications with matching Application-Specific Enclave (ASE)
- System service, Rollback Enclave (RE), implements ROTE library that ASEs use
- The design choice of introducing a dedicated system service (RE) hides the distributed counter maintenance from the applications

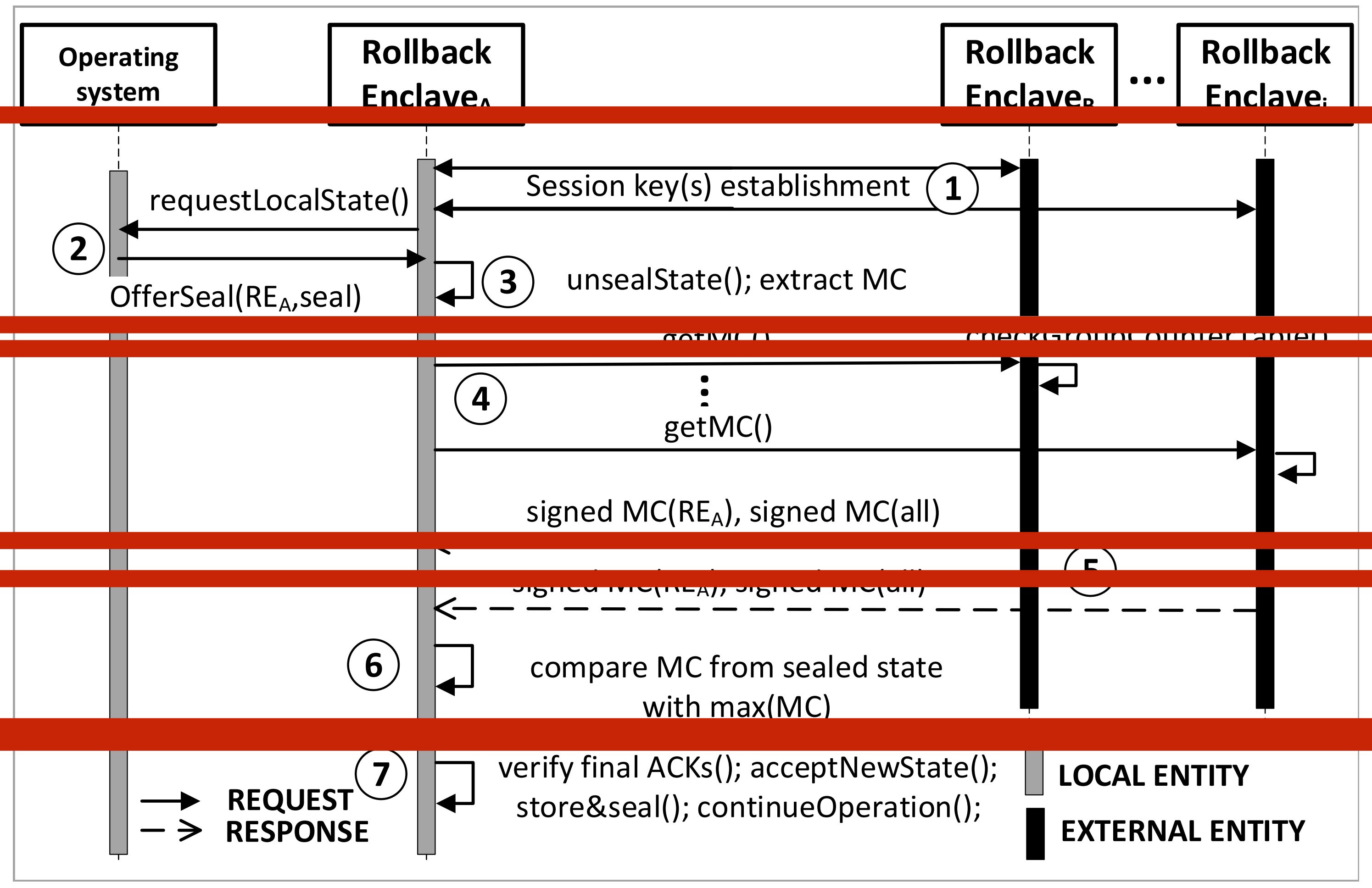


System protocols

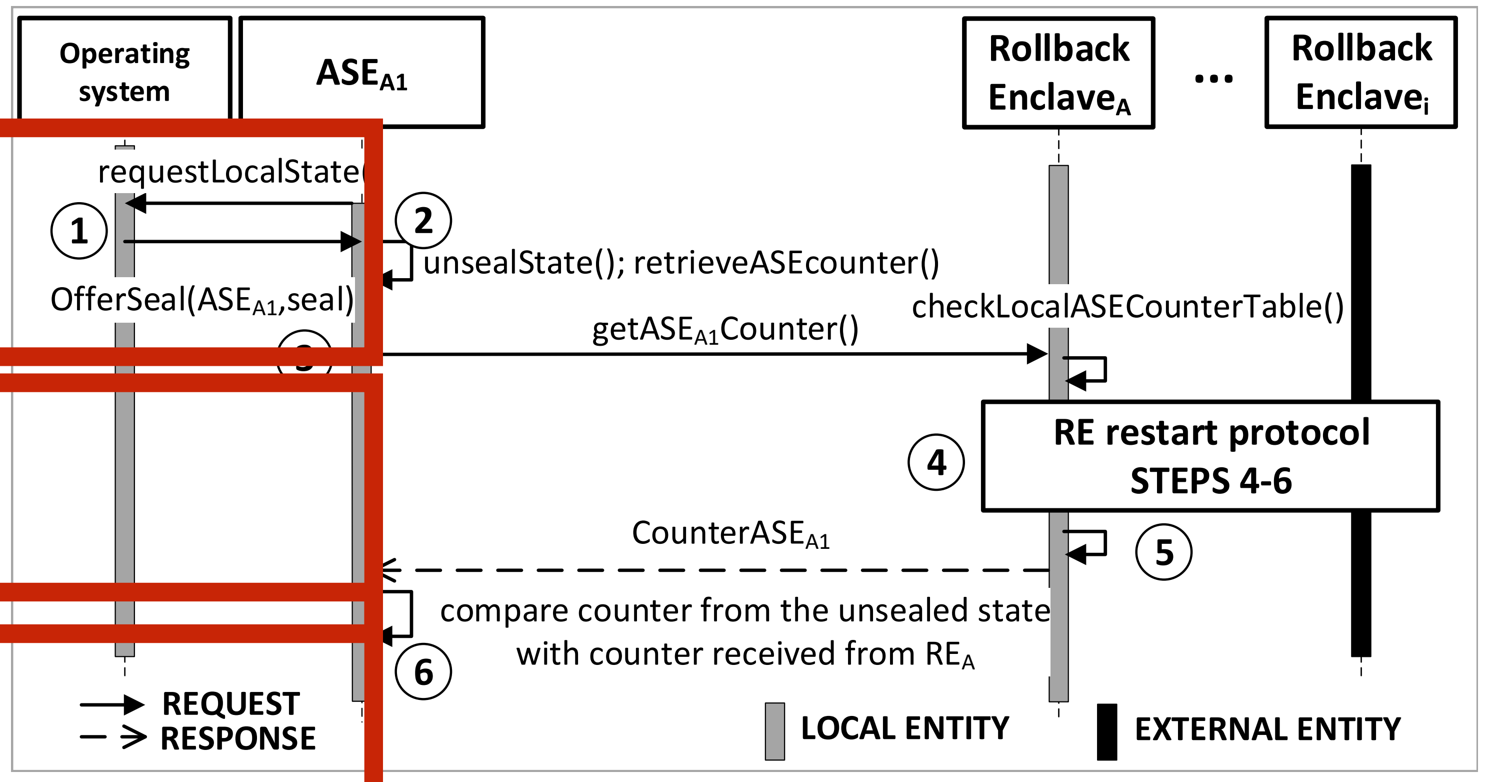
ASE State update protocol



RE restart protocol

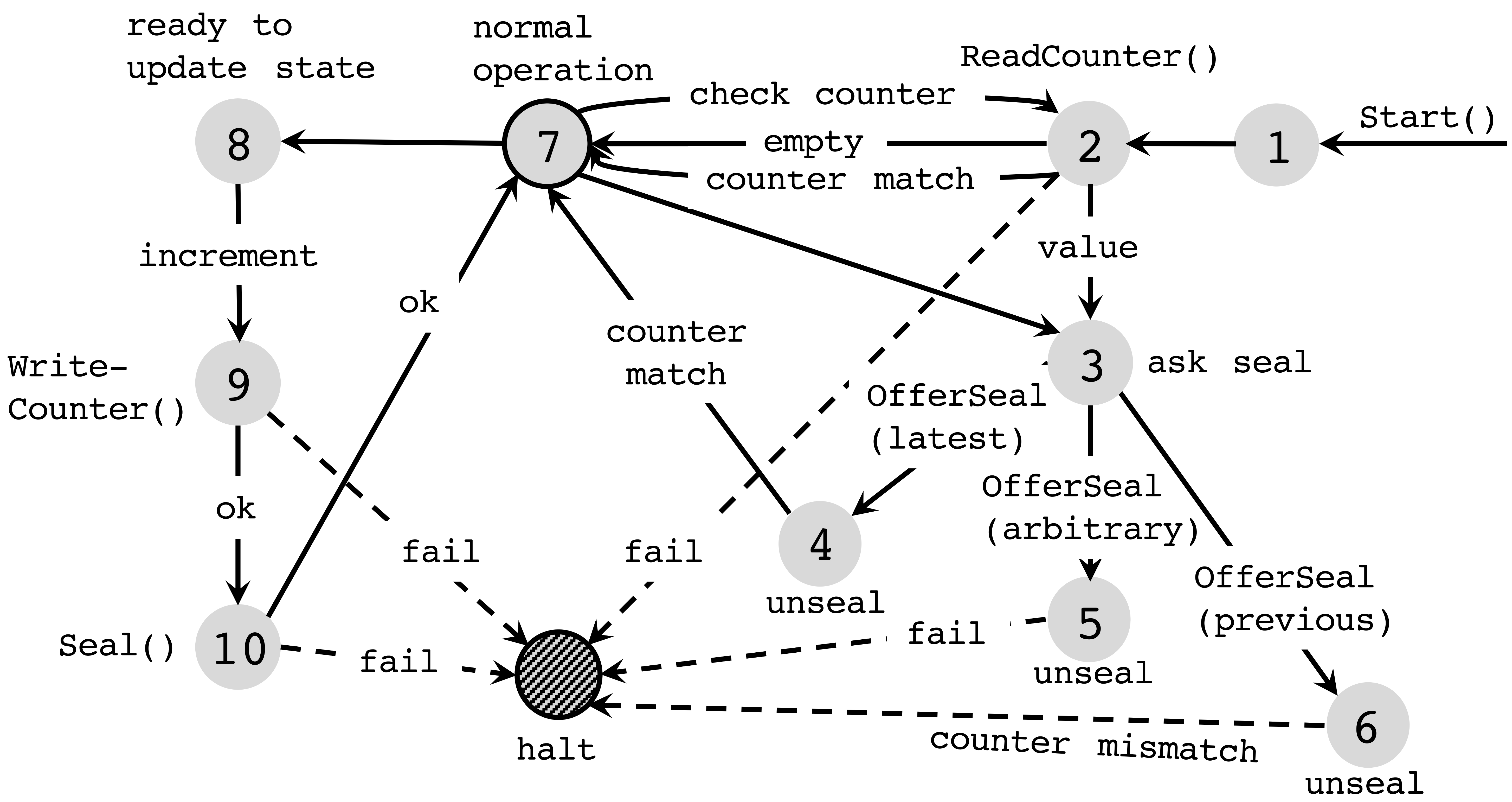


ASE start/read protocol



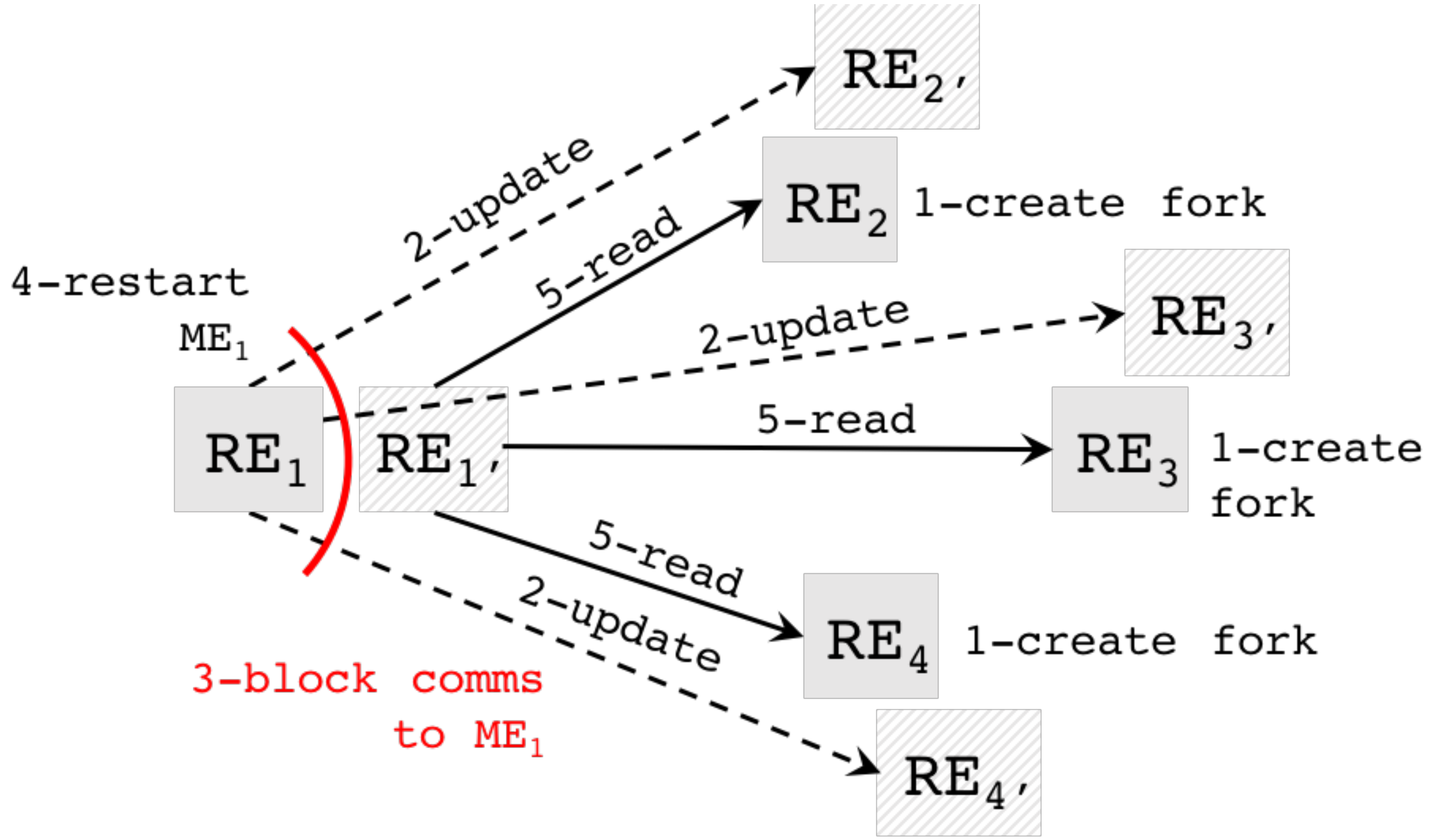
Security Analysis

- Basic intuition: Given a secure storage functionality (abstraction), the RE can verify that its state its the latest and rollback is prevented
  - First start
  - Sealing & Unsealing
  - Forking
  - Restart

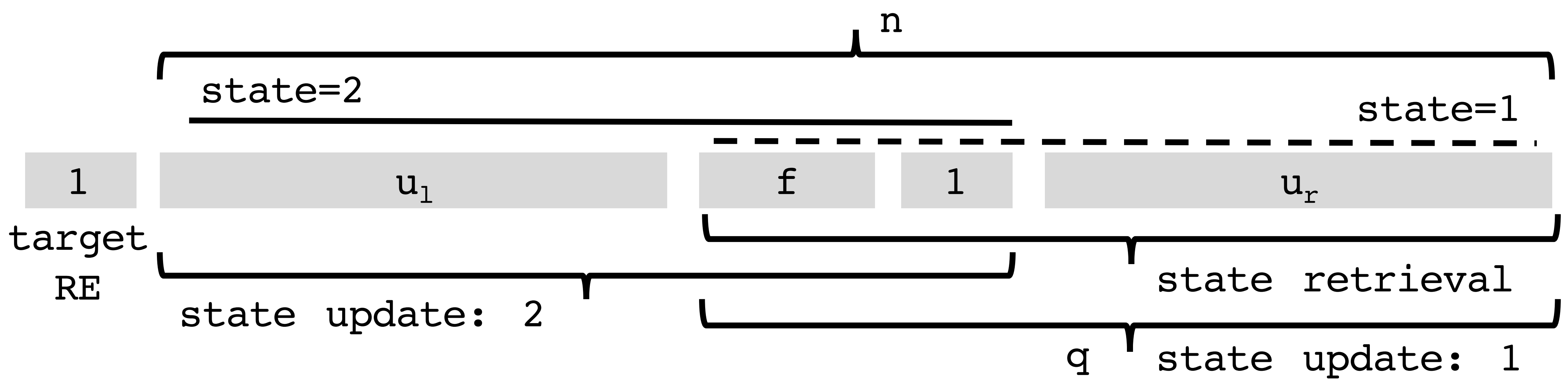


Security Analysis

- Realization of the secure storage functionality as a distributed system
  - Quorum size
  - Platform restarts
  - Forking attacks



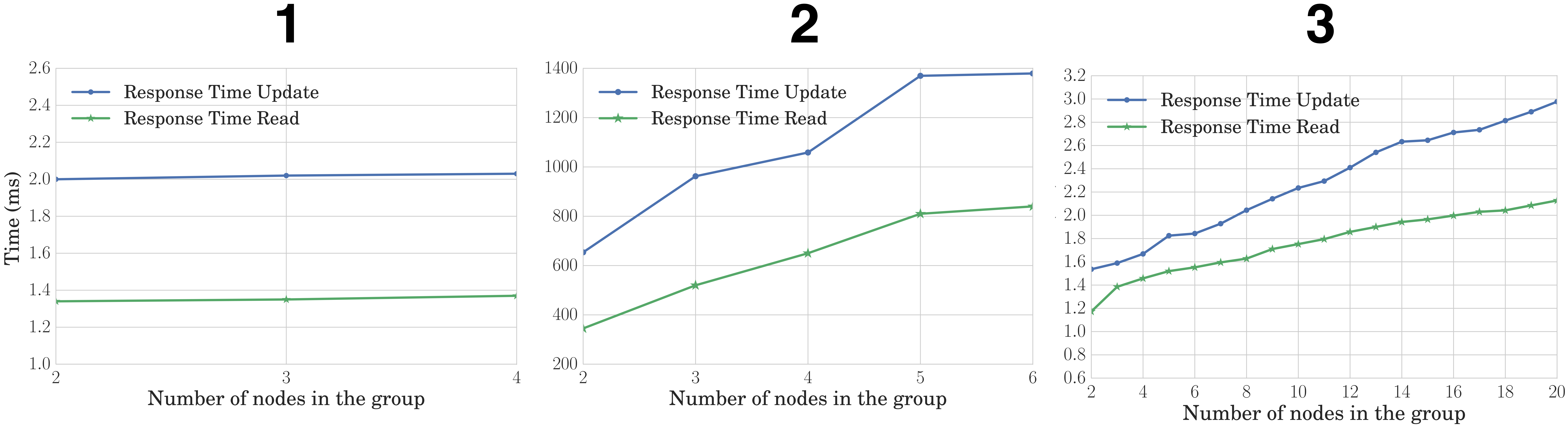
- **Dependency between the parameters is *n=f+2u+1***



Performance evaluation

- Experimental results - state update/read delay.

1. ROTE performance for a group within a local network,
2. Geographically distributed protection groups,
3. Simulated performance for a larger group within a local network.



Performance evaluation

Request type	State size (KB)	no rollback protection (ms)	ROTE system (ms)	SGX counter protection (ms)
Write state	1	3.85 (± 0.06)	5.17 (± 0.03)	160.7 (± 0.7)
	10	4.65 (± 0.05)	6.03 (± 0.03)	162.7 (± 1.6)
	100	6.49 (± 0.04)	7.83 (± 0.05)	169.1 (± 2.1)
Read state	1	0.06 (± 0.00)	1.41 (± 0.02)	61.04 (± 3.1)
	10	0.19 (± 0.00)	1.53 (± 0.01)	61.17 (± 3.1)
	100	1.76 (± 0.05)	3.1 (± 0.02)	62.74 (± 3.2)

Lessons learnt…

- The current SGX design and architecture clearly have some shortcoming that could be addressed in the future to strengthen its position
- Designing a distributed system that is both efficient and satisfies the required security properties proved to be quite a challenge
  - **http://scholar.harvard.edu/files/mickens/files/thesaddestmoment.pdf**
- During the whole project we stumbled upon numerous new attack vectors and thus had to change the core work to adapt
- Developing enclaves for Intel SGX is still buggy and cumbersome

Conclusion

- New security model for reasoning about the integrity and freshness of SGX applications
  - identified security weaknesses in existing SGX systems.
- SGX counter experiments showing limitation of the service
- Novel approach of realising rollback protection by storing enclave-specific counters in a distributed system
- Implemented ROTE system that ensures integrity and freshness of application data in a powerful adversarial model.
- Experimental evaluation showing only a small performance overhead for our system
  - in a low-latency network state update overhead is only 1-2 ms

ETH zürich

Thank you for your attention!
Any Questions?

sinisa.matetic@inf.ethz.ch