ETH zürich

Department of Computer Science,
Institute of Information Security

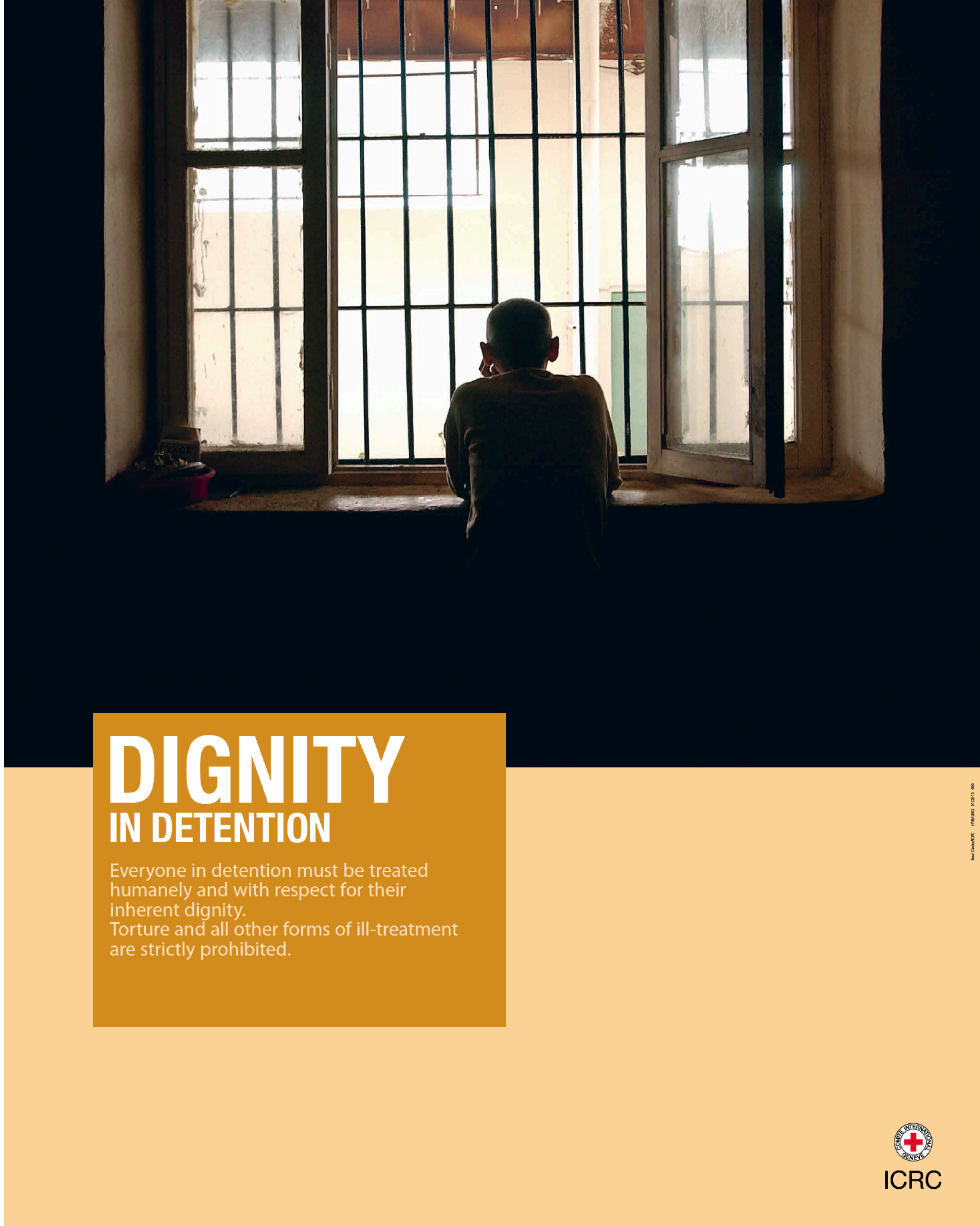
A Sovereign Cloud for the ICRC

Ivan Puddu

@AMLD EPFL 29/03/22

ICRC Mission

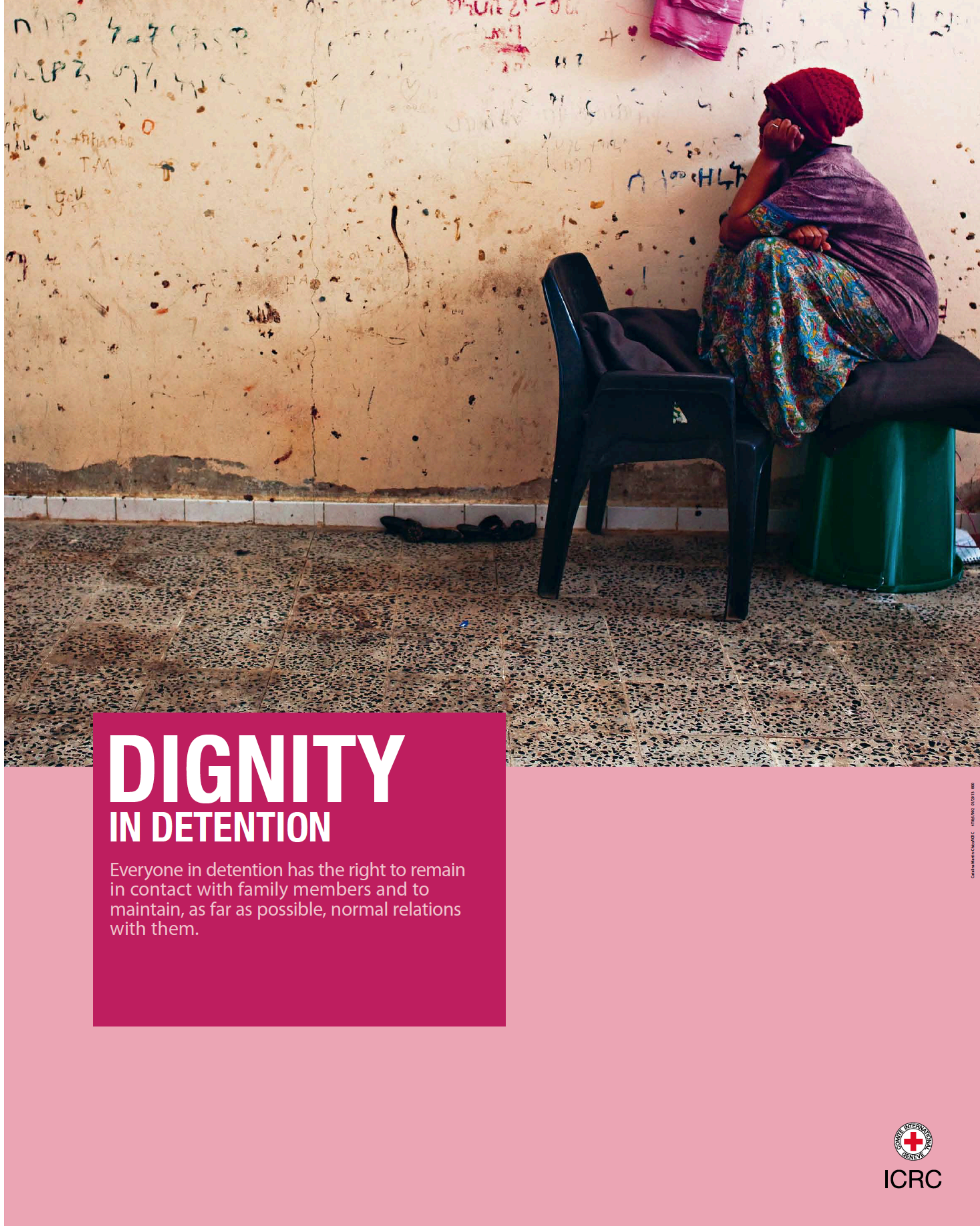
- Prevention and punishment of acts of torture and other form of ill-treatment under international humanitarian law (IHL) and international human rights law (IHRL)



Everyone in detention must be treated humanely and with respect for their inherent dignity.
Torture and all other forms of ill-treatment are strictly prohibited.

ICRC Mission

- Prevention and punishment of acts of torture and other form of ill-treatment under international humanitarian law (IHL) and international human rights law (IHRL)



Everyone in detention has the right to remain in contact with family members and to maintain, as far as possible, normal relations with them.

Catalina Martin-Chico/ICRC 4165/002 01/2013 800

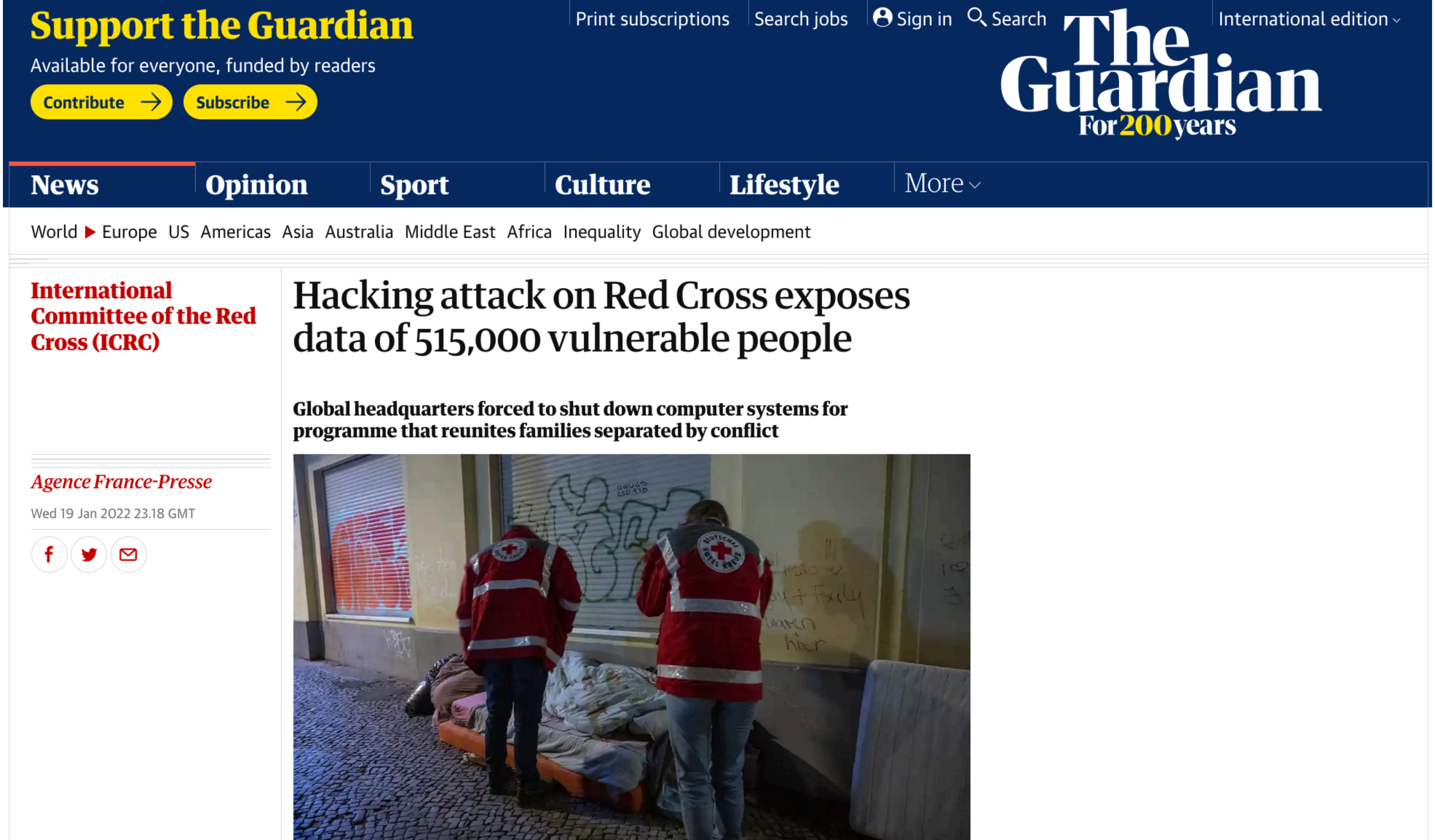
ICRC Mission

- Sensitive information is collected as part of the ICRC missions:
  - Which prisoner is kept in which prison
  - Who is crossing a border and when, which route are they taking
- The ICRC would like to be able to compute on this data
  - E.g., potentially use ML to reconnect lost relatives
  - They lack the infrastructure and expertise for this, thus offloading to the cloud is attractive for them

ICRC + Cloud

Drawbacks

- The information that the ICRC stores in the cloud might give a tactical advantage in an armed conflict
  - Data on the cloud might be subpoenaed by a judge
  - Can be a target of intelligence agencies
- *Lawful access* to ICRC cloud data can prevent them from fulfilling their mission
  - The ICRC can lose access to war prisons
  - Beneficiaries might not trust the ICRC with their data



International Committee of the Red Cross (ICRC)

Hacking attack on Red Cross exposes data of 515,000 vulnerable people

Global headquarters forced to shut down computer systems for programme that reunites families separated by conflict

Agence France-Presse

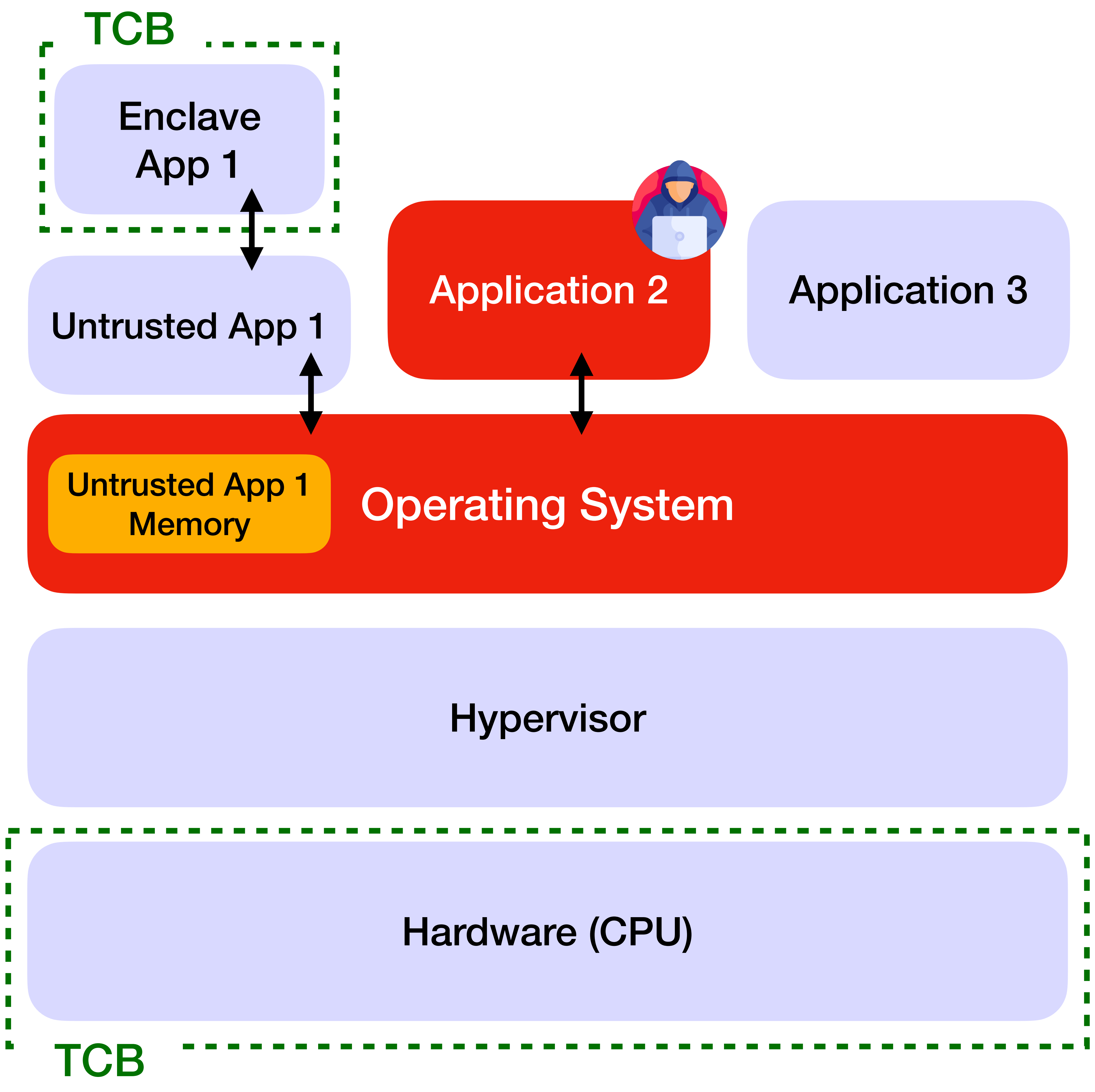
Wed 19 Jan 2022 23.18 GMT

ICRC Threat model

- Attacker with state level capabilities and lawful access to third party cloud infrastructure and the data stored in it
- ICRC facilities are physically protected and cannot be lawfully accessed
- ICRC agents cannot be coerced
- Manufacturer is trusted to produce CPUs/Hardware according to specification

TEEs - SGX

- CPU primitives isolate applications from a malicious OS/Hypervisor
- Drawbacks:
  - Side-channels
  - Need to trust a third party



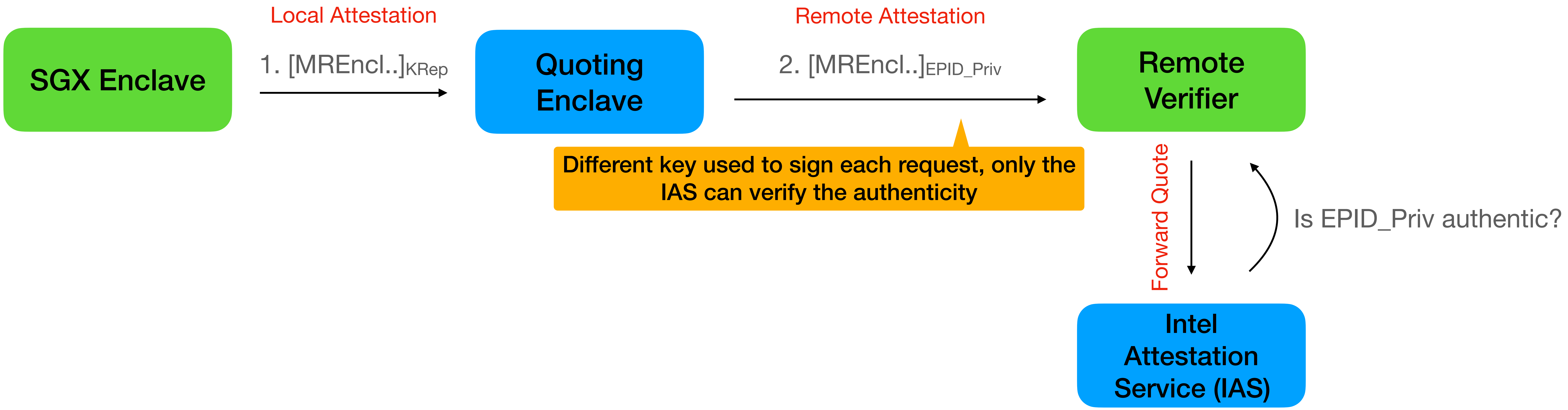
ICRC Threat model

Can we use a TEE?

- Concrete assumptions:
  - TEE manufacturer is trusted for manufacturing
  - TEE manufacturer is **not** trusted at runtime
  - The attacker compromised the OS (or is colluding with the CSP)
- Can we use SGX (+/- DCAP) or SEV (or a combination of them), under this threat model?

SGX Attestation

Non-Linkable Mode



SGX Enclave

Local Attestation

1. [MREncl..]$_{\text{KRep}}$

Quoting Enclave

Remote Attestation

2. [MREncl..]$_{\text{EPID\_Priv}}$

Remote Verifier

Different key used to sign each request, only the IAS can verify the authenticity

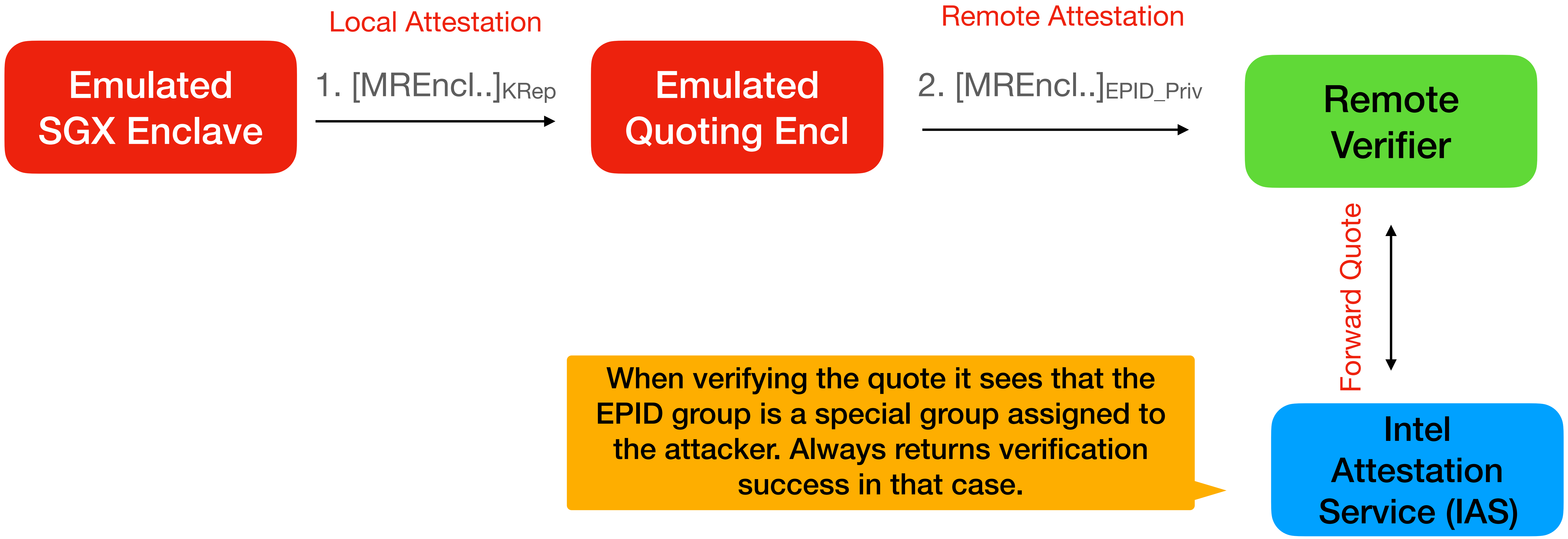
Forward Quote

Is EPID_Priv authentic?

Intel Attestation Service (IAS)

SGX Attestation attacks

Trivially broken case 1:

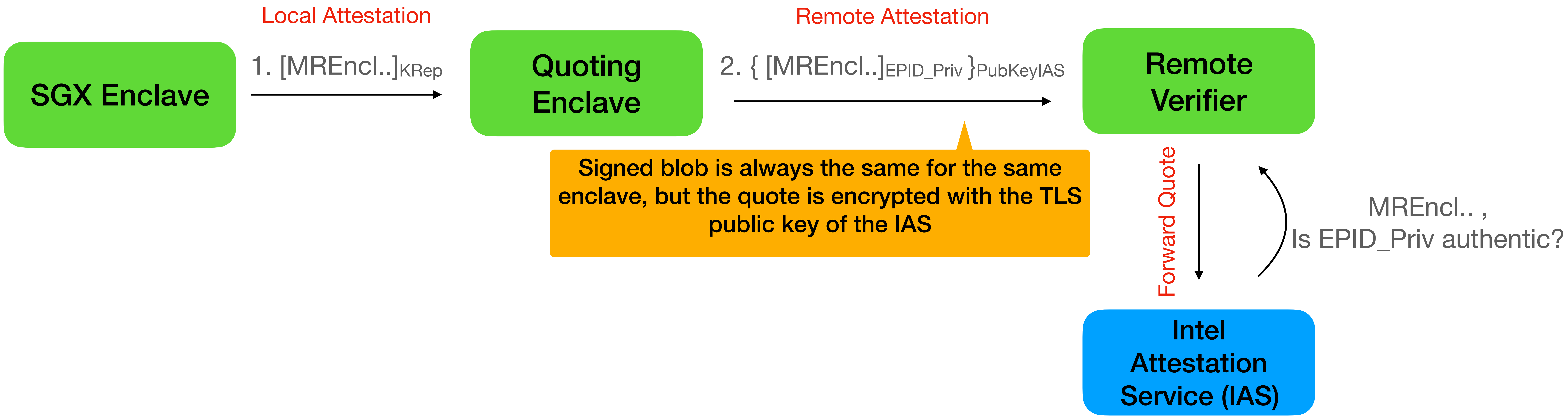


When verifying the quote it sees that the EPID group is a special group assigned to the attacker. Always returns verification success in that case.

- The remote verifier cannot distinguish attestation quotes, so does not realize that a new attacker controlled EPID_Priv key is being used for attestation

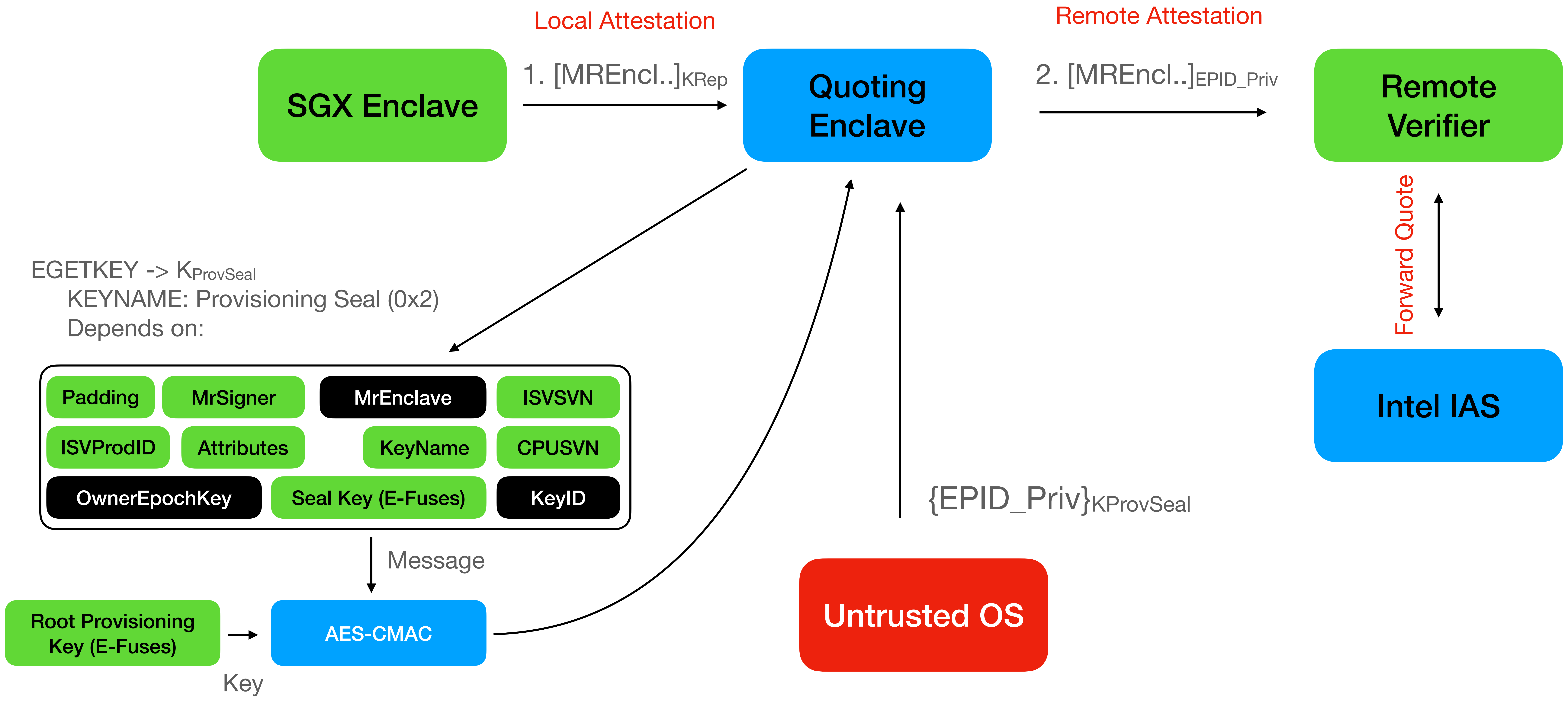
SGX Attestation

Linkable Mode



Signed blob is always the same for the same enclave, but the quote is encrypted with the TLS public key of the IAS

SGX Attestation



Local Attestation

Remote Attestation

SGX Enclave

1. [MREncl..]$_{KRep}$

Quoting Enclave

2. [MREncl..]$_{EPID\_Priv}$

Remote Verifier

Forward Quote

Intel IAS

EGETKEY -> K$_{ProvSeal}$

KEYNAME: Provisioning Seal (0x2)

Depends on:

Padding | MrSigner | MrEnclave | ISVSVN

ISVProdID | Attributes | KeyName | CPUSVN

OwnerEpochKey | Seal Key (E-Fuses) | KeyID

Message

Root Provisioning Key (E-Fuses)

Key

AES-CMAC

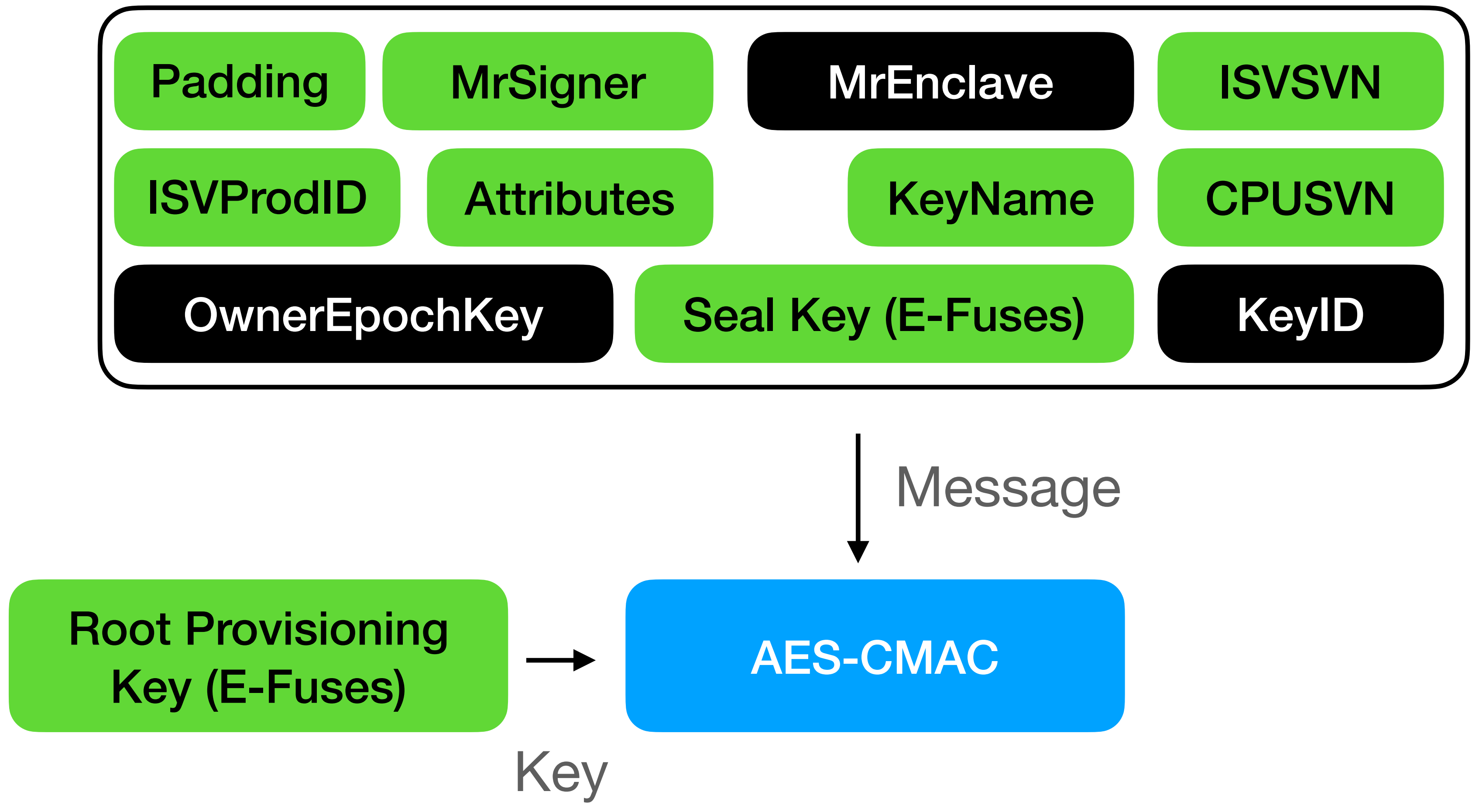
{EPID_Priv}$_{KProvSeal}$

Untrusted OS

SGX Attack 1

- Intel could make an enclave that spits out the current provisioning seal key or equivalently the private EPID key
- With that key the attacker can fake remote attestations
- The attack needs to be repeated every time there is a TCB update

EGETKEY -> $K_{ProvSeal}$
KEYNAME: Provisioning Seal (0x2)
Depends on:

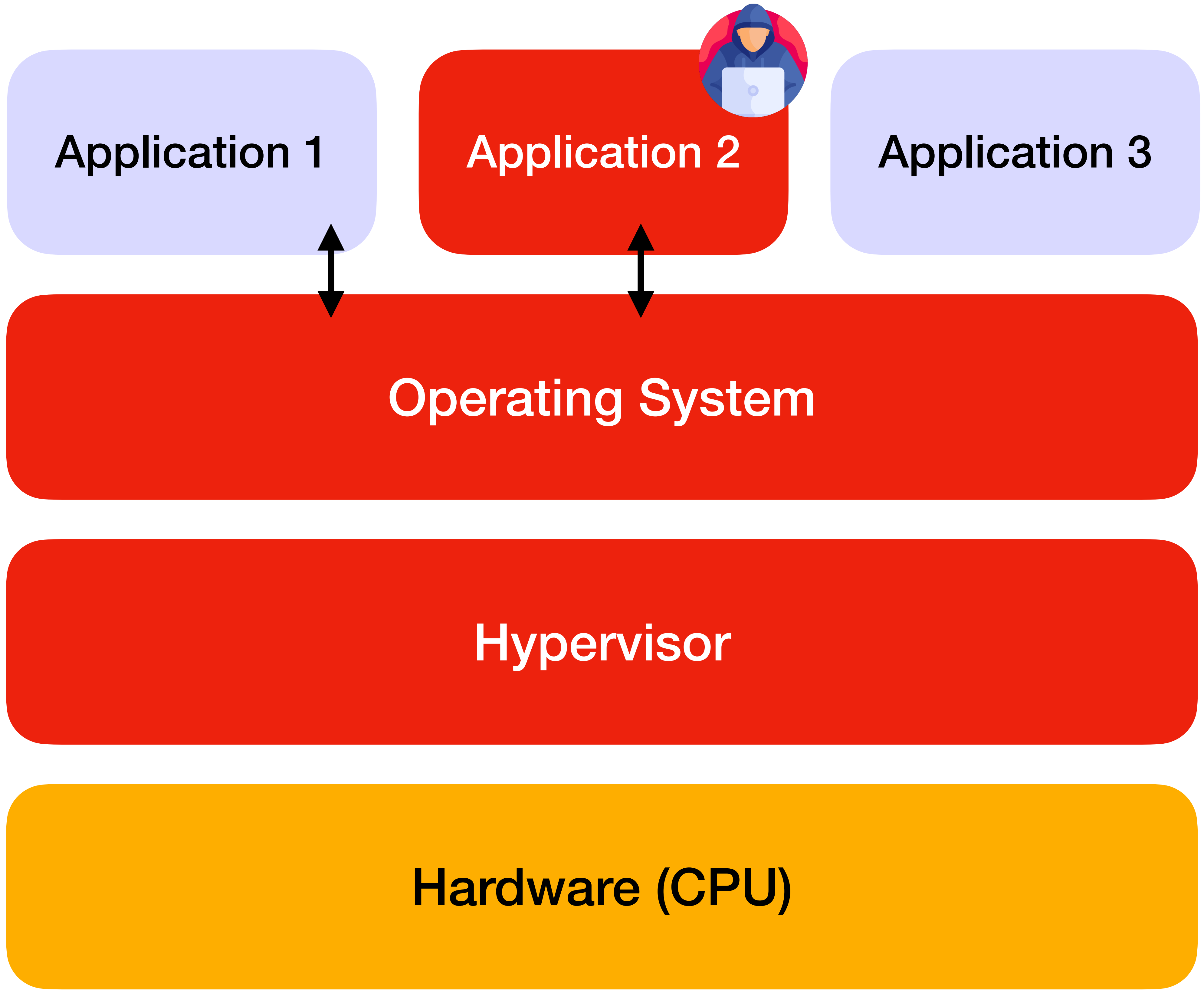


Recap on Trust Assumptions

- There is a difference between trusting a manufacturer for manufacturing and at runtime. The former need not imply the latter
- The root of trust of SGX/SEV is built on keys which are available at runtime to the CPU manufacturer
  - This does not fit in the ICRC attacker model, as the manufacturers can be compelled to act maliciously *at runtime*
- Can we provide TEEs guarantees without relying on a third party at runtime?

Computing Stack

- How can we reduce the TCB without (fully) trusting the CPU?
  - i.e. Without not trust the CPU manufacturer at runtime



ICRC Sovereign Cloud

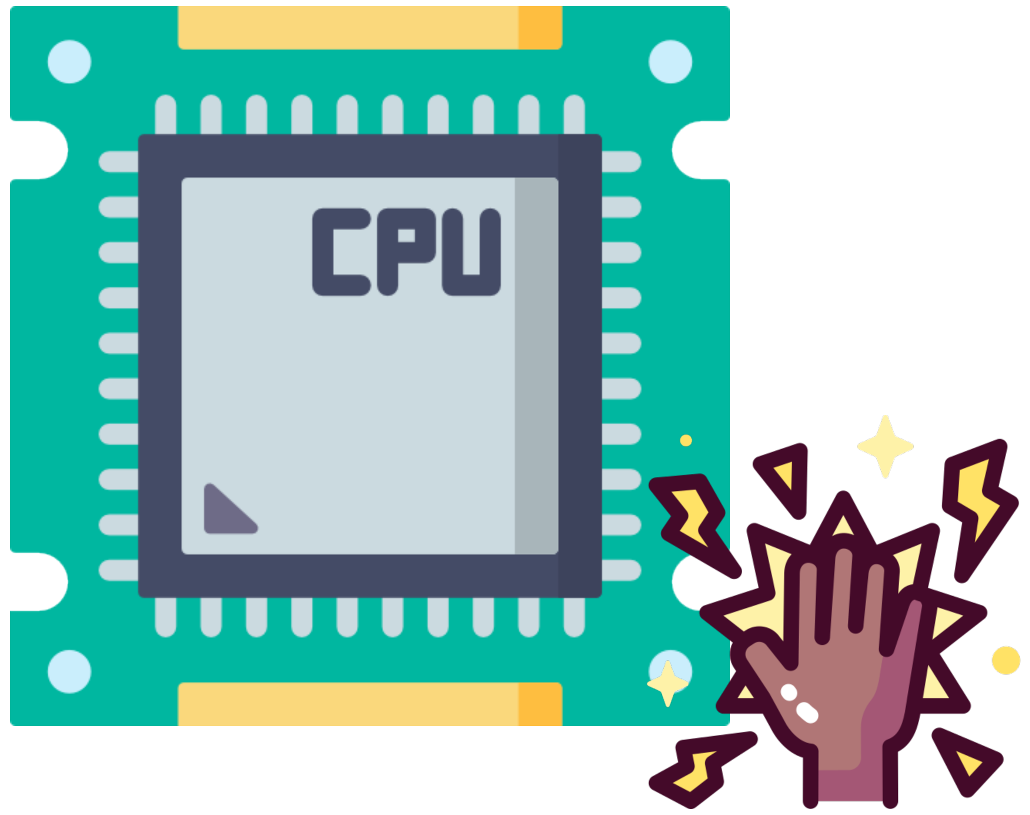
Problem analysis

IDT

Number: 1	Addr: 0xaa
Number: 2	Addr: 0xab
…	…

Application / VM

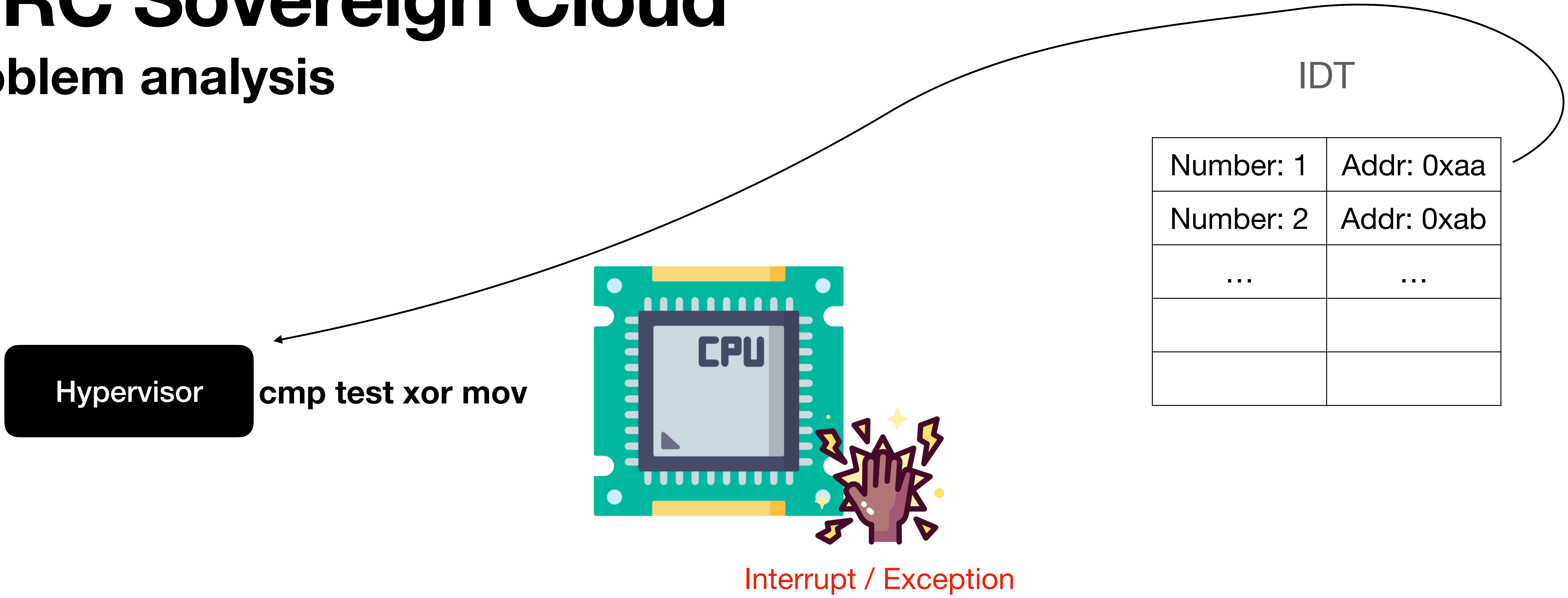
cmp test xor mov



Interrupt / Exception

ICRC Sovereign Cloud

Problem analysis



cmp test xor mov

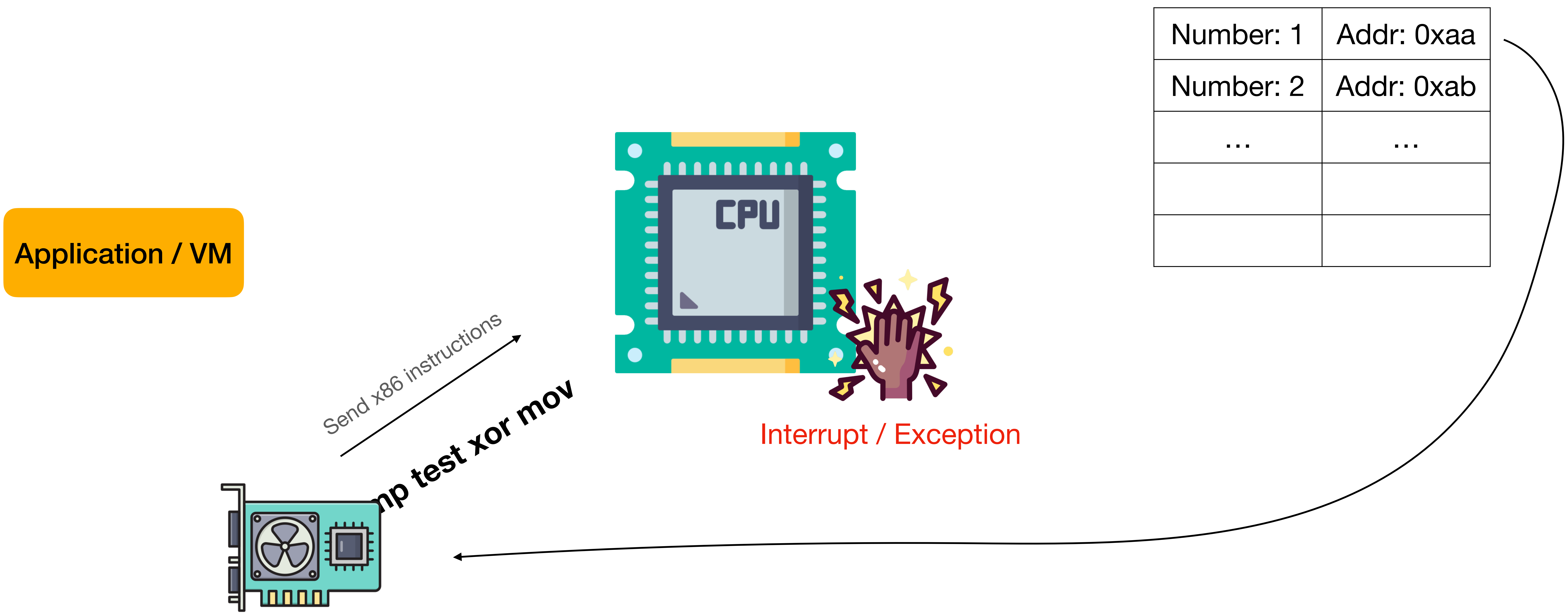
IDT

Number: 1	Addr: 0xaa
Number: 2	Addr: 0xab
…	…

ICRC Sovereign Cloud

- We can redirect all hypervisor entry points to the memory region of a device we control instead
- This would only be temporary, when execution is done, we can restore the previous execution environment
  - During the ICRC execution, VM migration and sharing a server with other customers will not be possible (although this might not be relevant on a separate cloud deployment)
- This solution is suited for a custom cloud deployment, i.e. in an ICRC facility but managed by a CSP

ICRC Sovereign Cloud

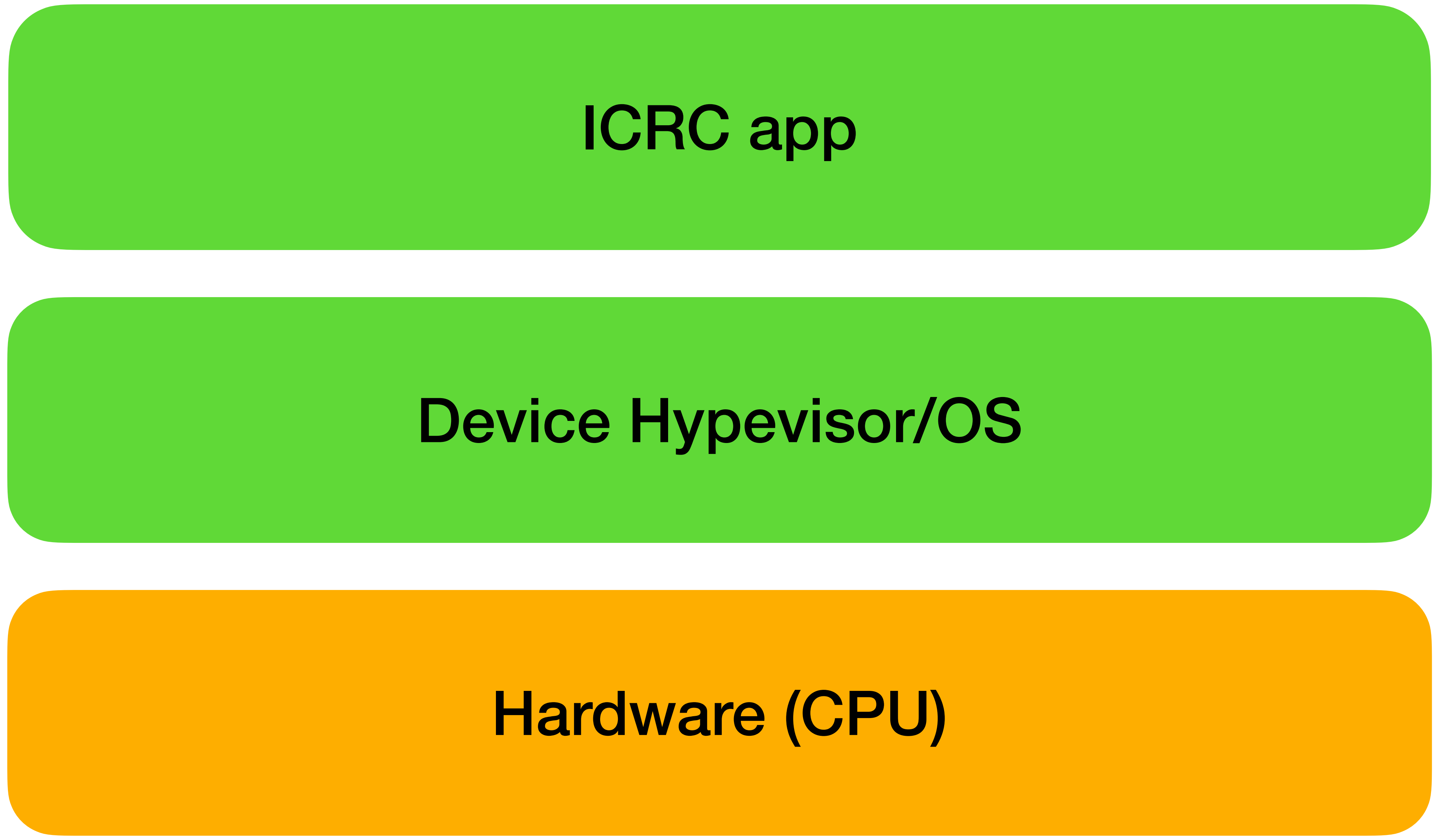


Number: 1	Addr: 0xaa
Number: 2	Addr: 0xab
…	…

ETH zürich

Computing Stack with our device

- After the device gains control over the system it takes over the whole computation stack

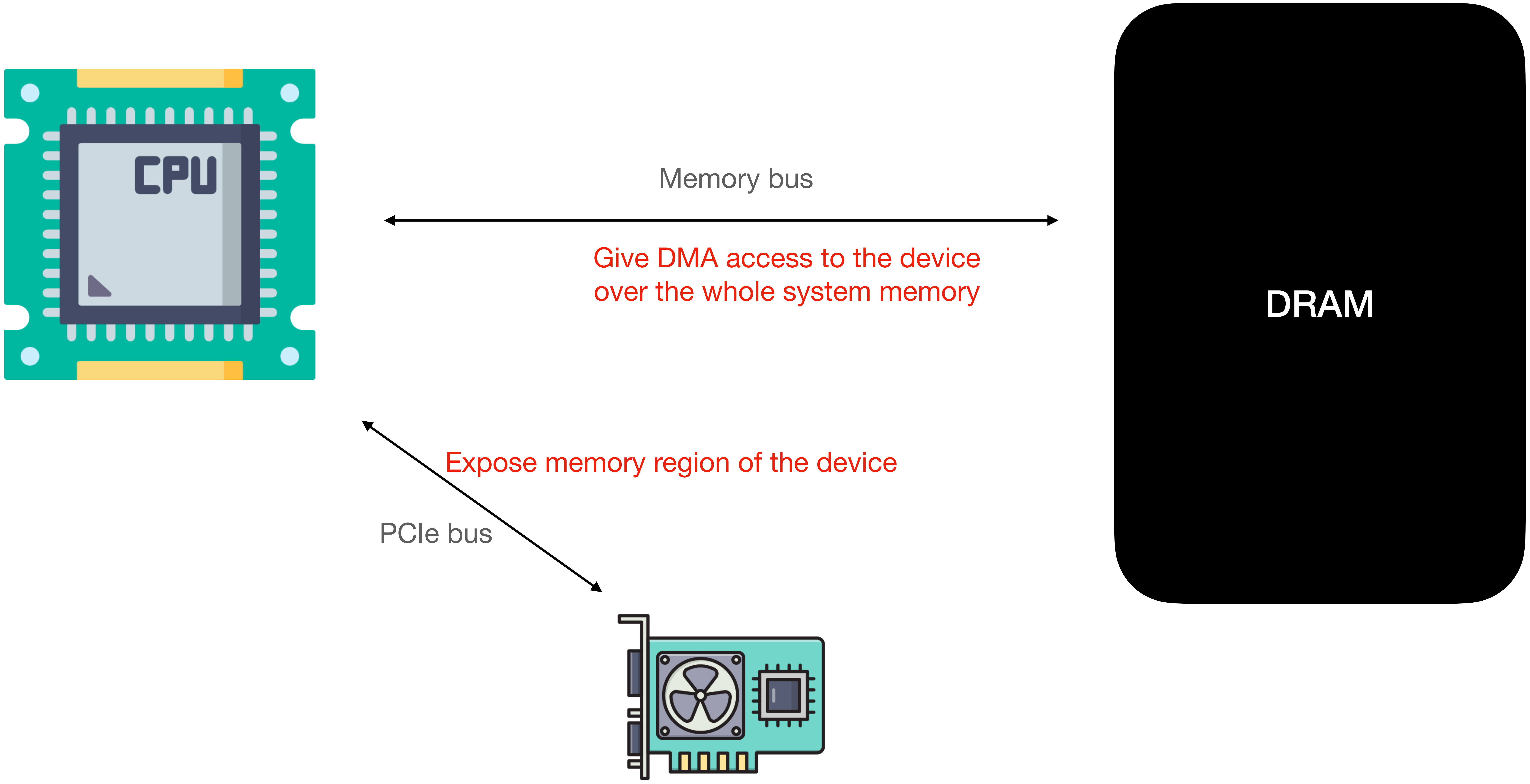


ICRC Sovereign Cloud

Device requirements

- Functionality-wise the device needs to:
    - Expose a readable memory region to the CPU (to serve instructions)
    - Be able to read the memory of the server in which it is installed
    - This is needed to handle page-fault exceptions or other interrupts
- PCIe + DMA gives us these primitives

An “embassy” in the Cloud



Memory bus

Give DMA access to the device over the whole system memory

DRAM

Expose memory region of the device

PCIe bus

When an interrupt/exception occurs the CPU will fetch instructions directly from the device over the PCIe bus

Sovereign Cloud

- Why do we need a separate device for this?
    - If the check was done by the CPU itself there would be no way of verifying that the IDT has been replaced (emulation vs real)
    - Can do key management / attestation without relying on a third party
- Device is simple
    - Data owner can own/manufacture the device

ICRC Sovereign Cloud

Notes on bus encryption/authentication

- Data between CPU and the device needs to be at least authenticated
  - A confidential channel can be built on top of this
- In PCIe v5 and v6, the CPU root complex (PCIe controller) has a key that can be configured and allows to secure the bus
- What about other devices in the PCIe bus?
  - The device can be the central point of communication, any bus transaction not initiated by the device should then indicate that something shady is going on

Summary

- Current TEEs manufacturers cannot be trusted at runtime
- Our device is able to build a TCB when needed to compute on sensitive data
- When not needed we can use the CSP to manage the ICRC cloud