SCONE (Secure CONtainer Environment)

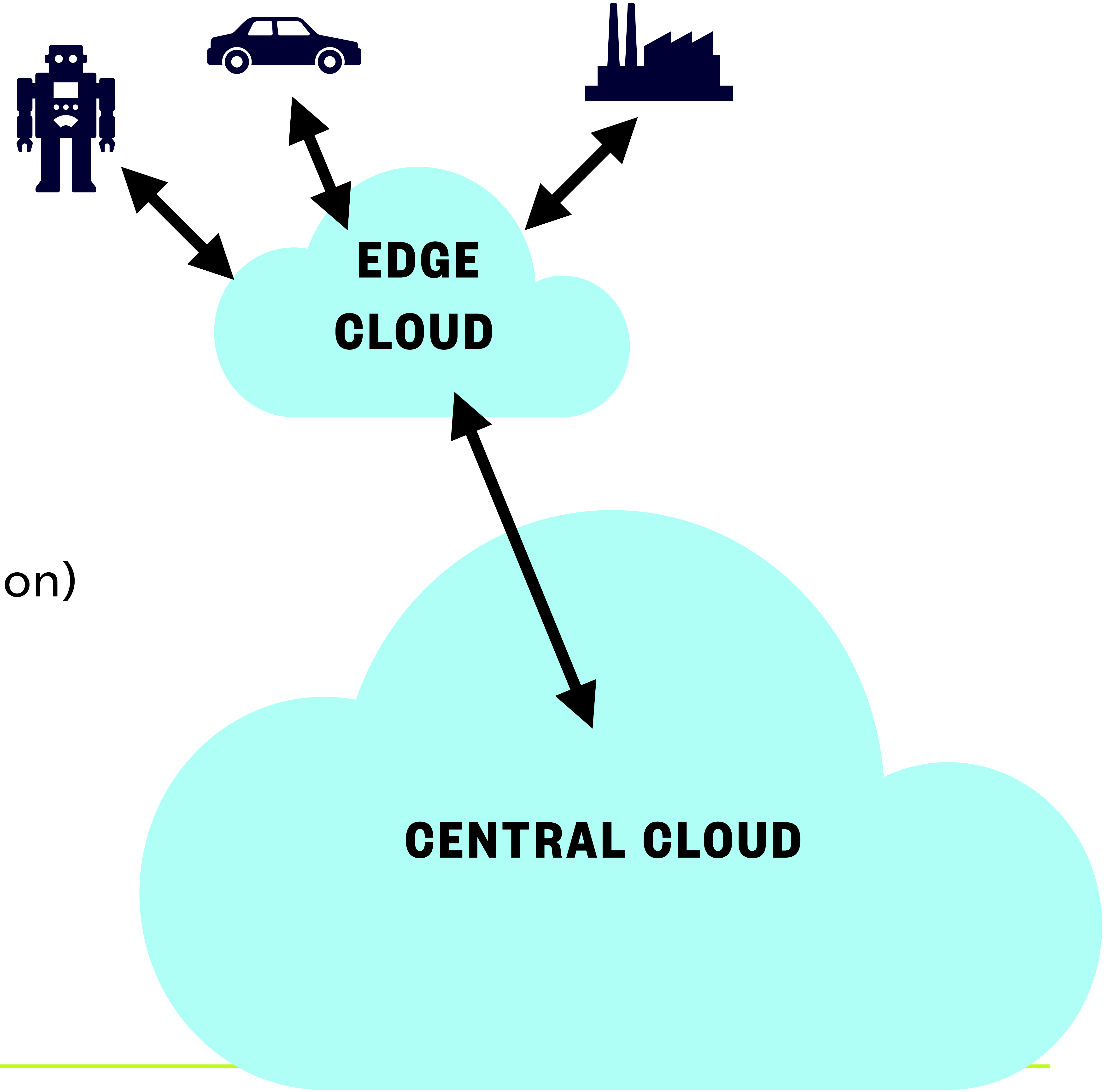
Trust Management in Edge Clouds with SCONE

PROF. CHRISTOF FETZER, PHD, TU DRESDEN, CHRISTOF.FETZER@TU-DRESDEN.DE

SCONEDOCS.GITHUB.IO

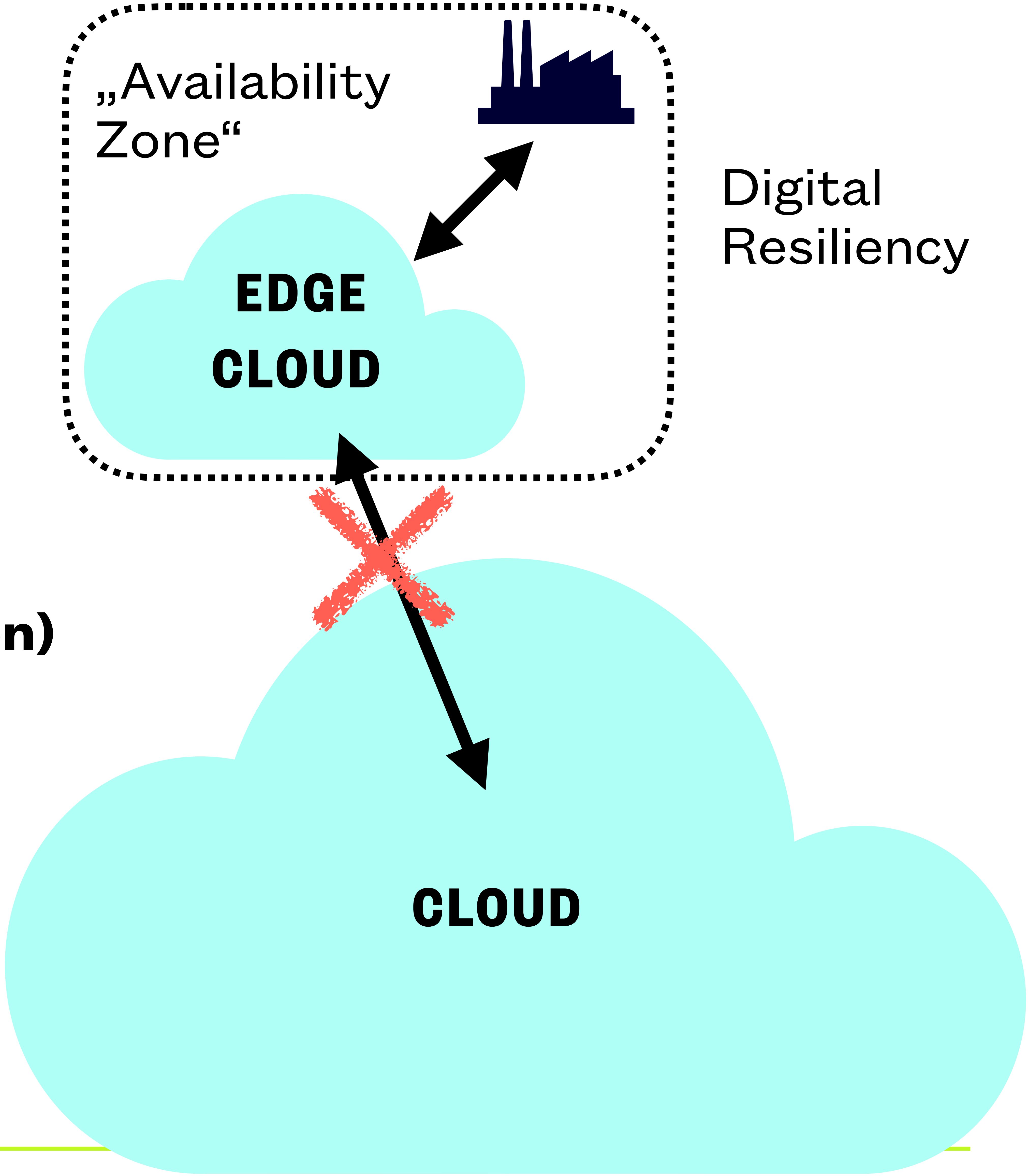
Motivation

- **Edge clouds provides clients with**
  - **Low latency**
  - **High Bandwidth**
  - Availability/Digital Resiliency (autonomous operation)



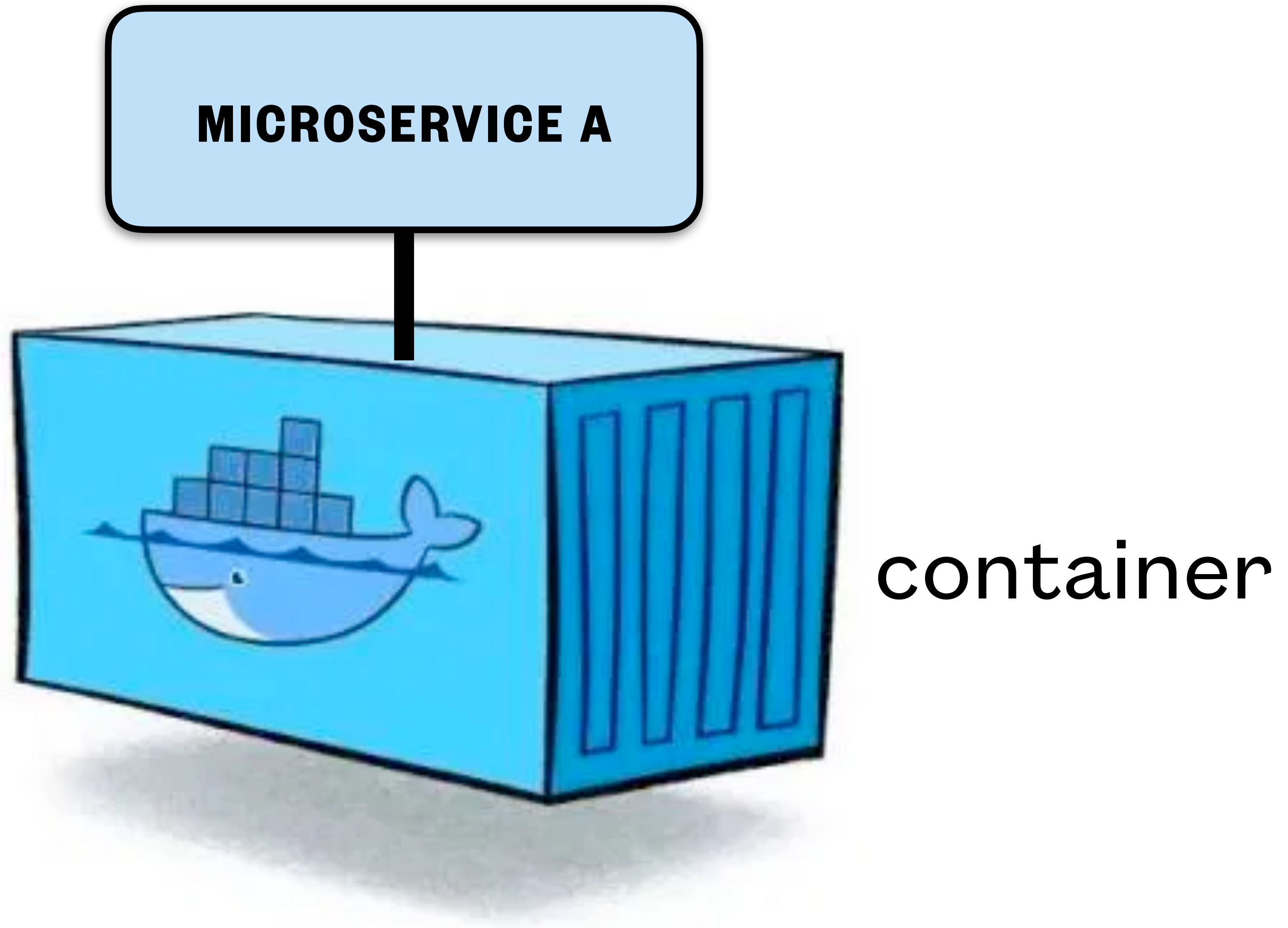
Motivation

- **Edge clouds provides clients with**
  - Low latency
  - High Bandwidth
  - **Availability/Digital Resiliency (autonomous operation)**



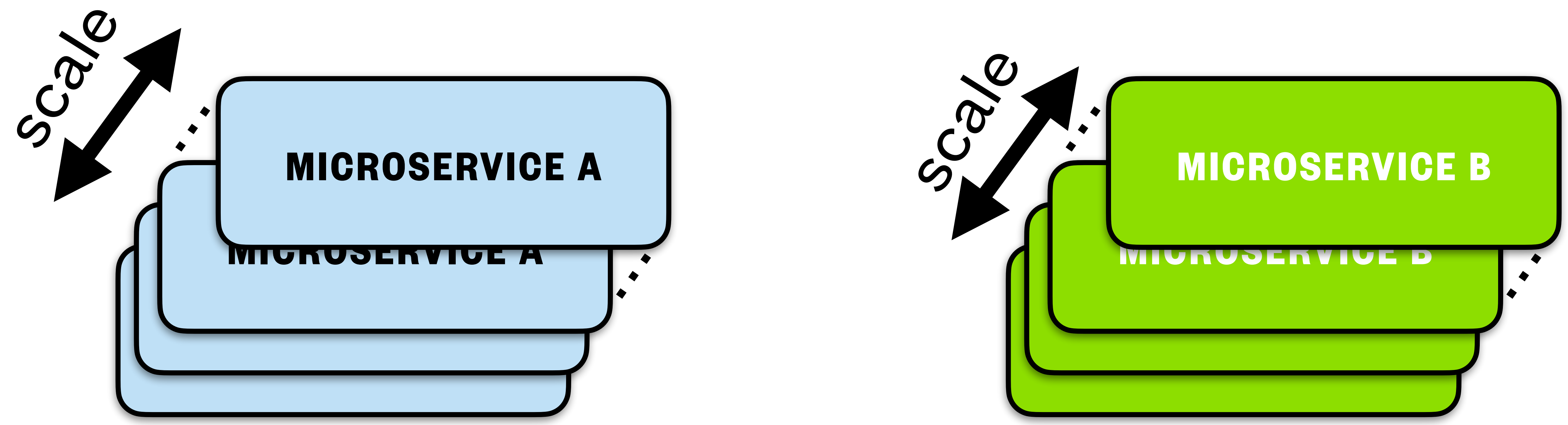
Expectations

- **Analysts expect**:
  - Large fraction of new cloud infrastructure investment will be spent on the **edge**
  - Almost all workloads will be deployed with **containers/ microservices**
  - Large fraction of edge cloud will be „pay per use“ („edge cloud **as a service**“)



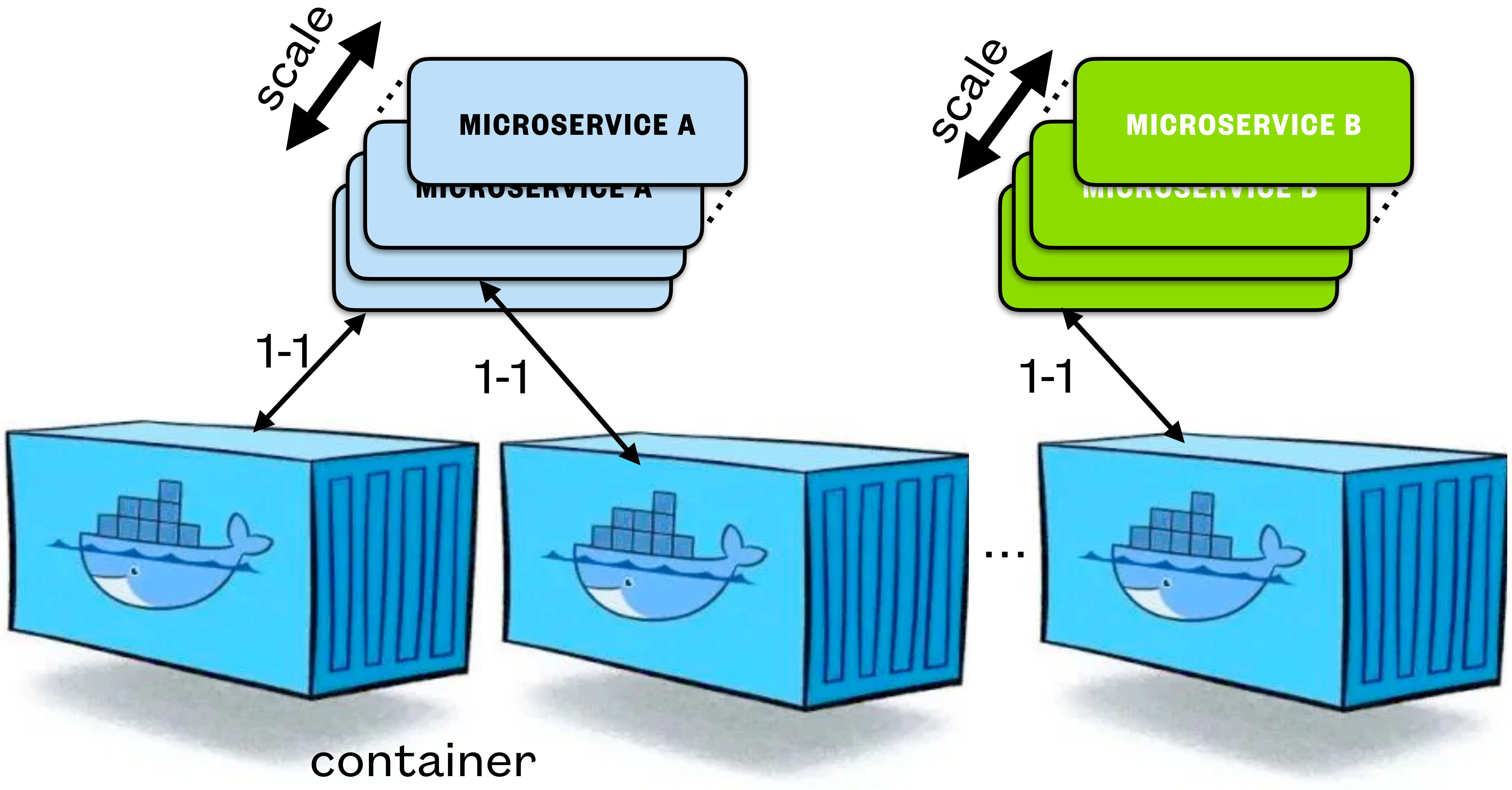
Application = Set of Microservices

- **Application**
  - Consists of **microservices**
  - Microservice provides single type of resources
  - **REST API** to access resources (**https API**)



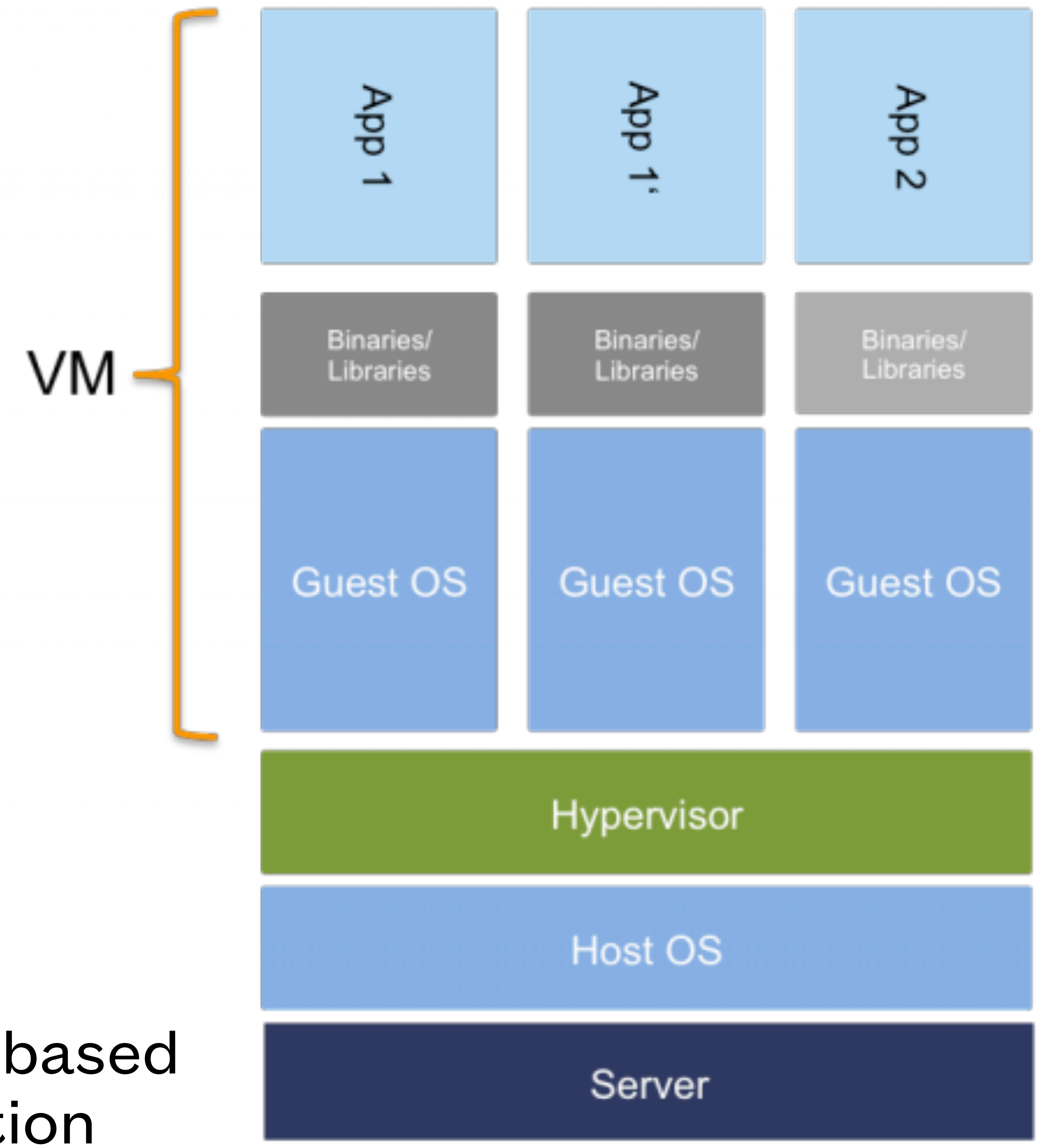
Application = Set of Microservices

- **Application**
  - Consists of **microservices**
  - Microservice provides single type of resources
  - **REST API** to access resources (**https API**)
- **Deployment:**
  - each microservice runs in a separate container

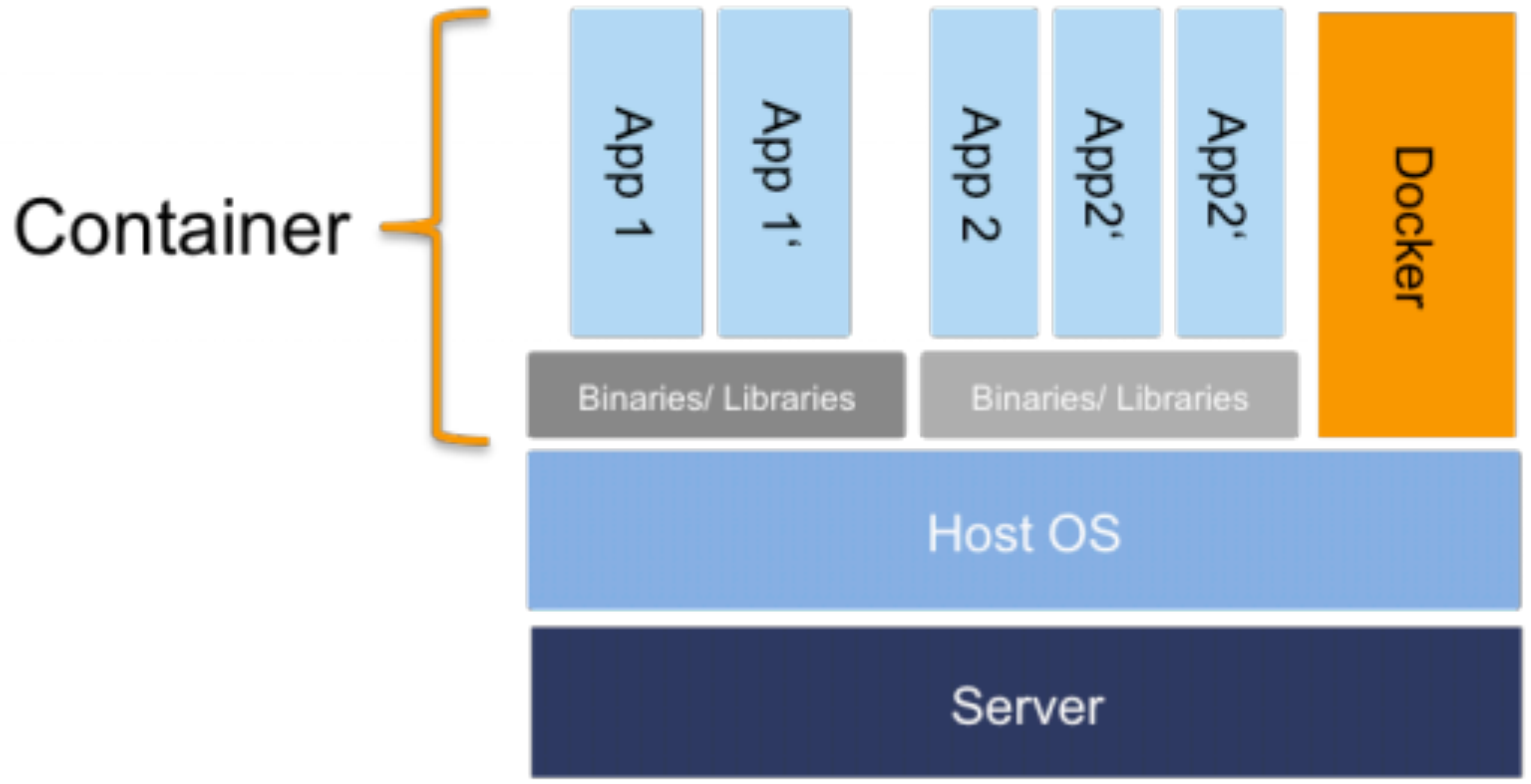


Container
Vs Virtual Machines (VM)

- **Containers**: OS-based virtualization
  - Less overhead ; very fast startup
  - Convenient interface (via Docker)



CPU-based isolation



OS-based Isolation

Quelle: Docker, Crisp Research, 2014

Digital Resiliency Challenges

- (Docker) **Containers:** are considered to be less secure than VMs
- **Edge cloud**: has **weaker physical security** than central cloud (e.g., shipping containers)
- **Managed Service**: Customer has to trust cloud provider to keep **data** and **code confidential**!

edge cloud as shipping container

(C) Cloud&Heat, 2020

General Approach

- **Customer**:
  - Delegates management of applications and services to (edge) cloud provider
- **Customer** and **Edge cloud provider** can use **monitoring** to show
  - availability,
  - durability,
  - latency, and
  - throughput.

Challenge

- How can we ensure the **confidentiality** and **integrity** of
  - **Data,**
  - **Code** (e.g., Python code) and
  - **Secrets**
- Of **managed services without needing to trust the (edge) cloud provider?**

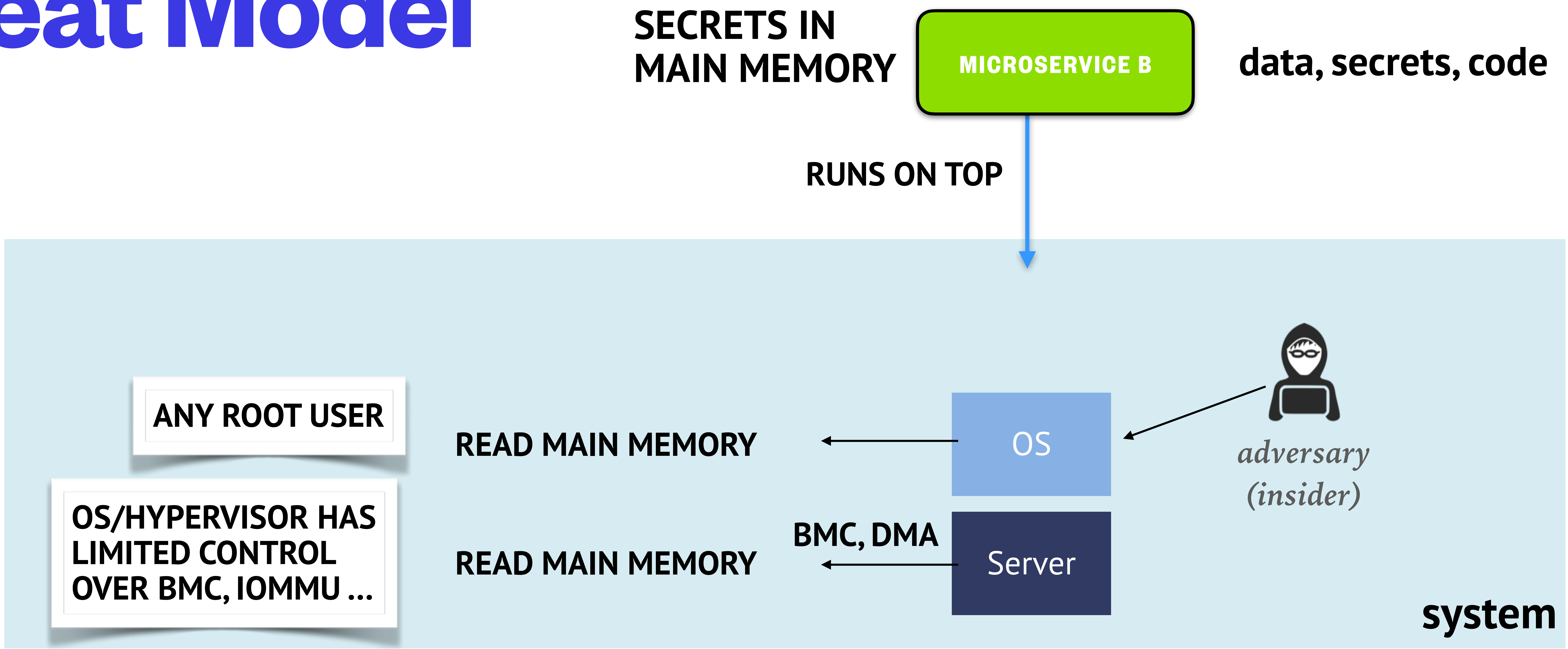
SCONE

THREAT MODEL

ADVERSARY HAS ROOT & HW ACCESS!

ADVERSARY MANAGES SERVICES!

Threat Model

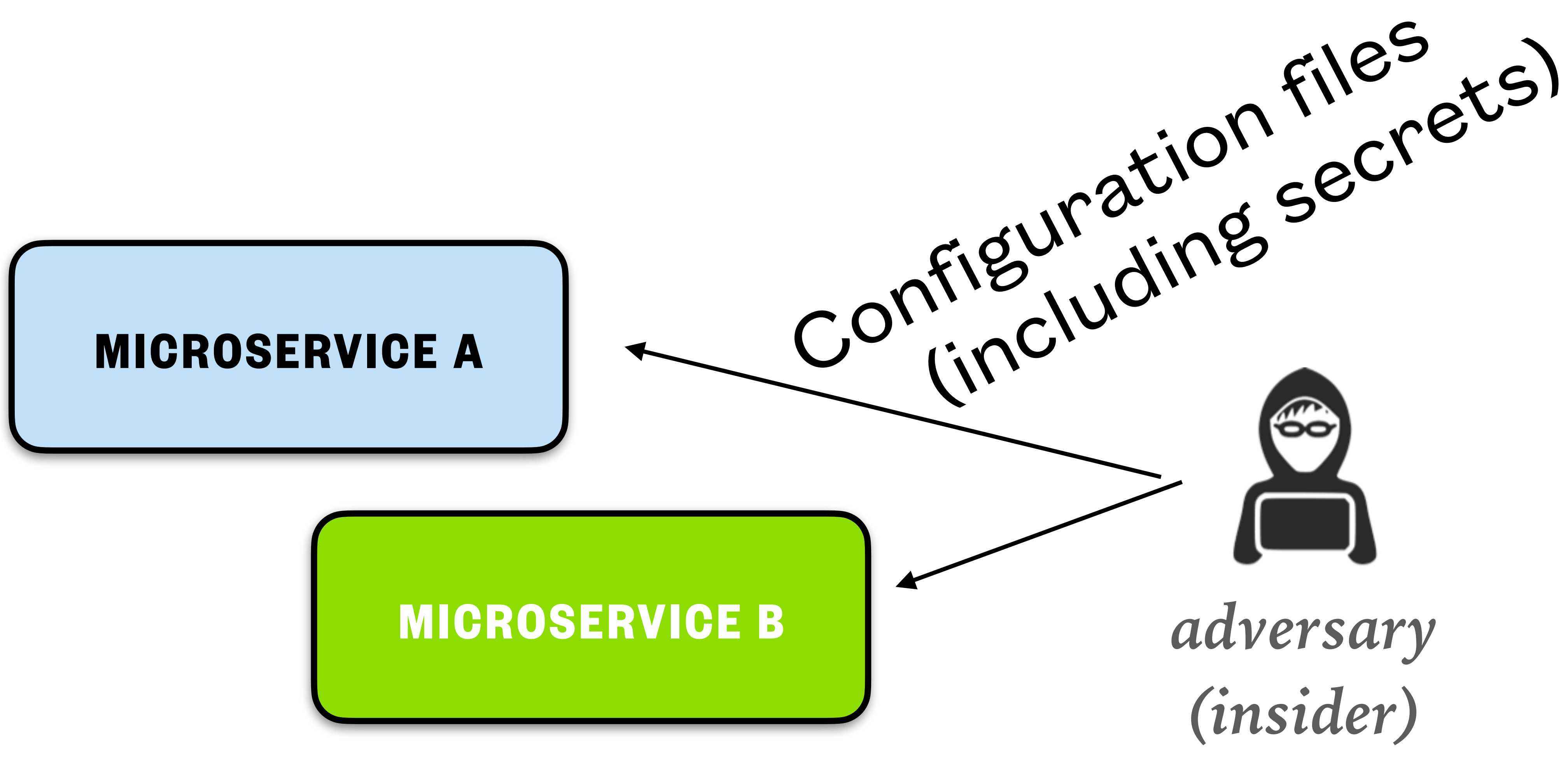


Adversary has hardware and root access

READ MAIN MEMORY: VIOLATES CONFIDENTIALITY

WRITING MAIN MEMORY: VIOLATES INTEGRITY

Threat Model



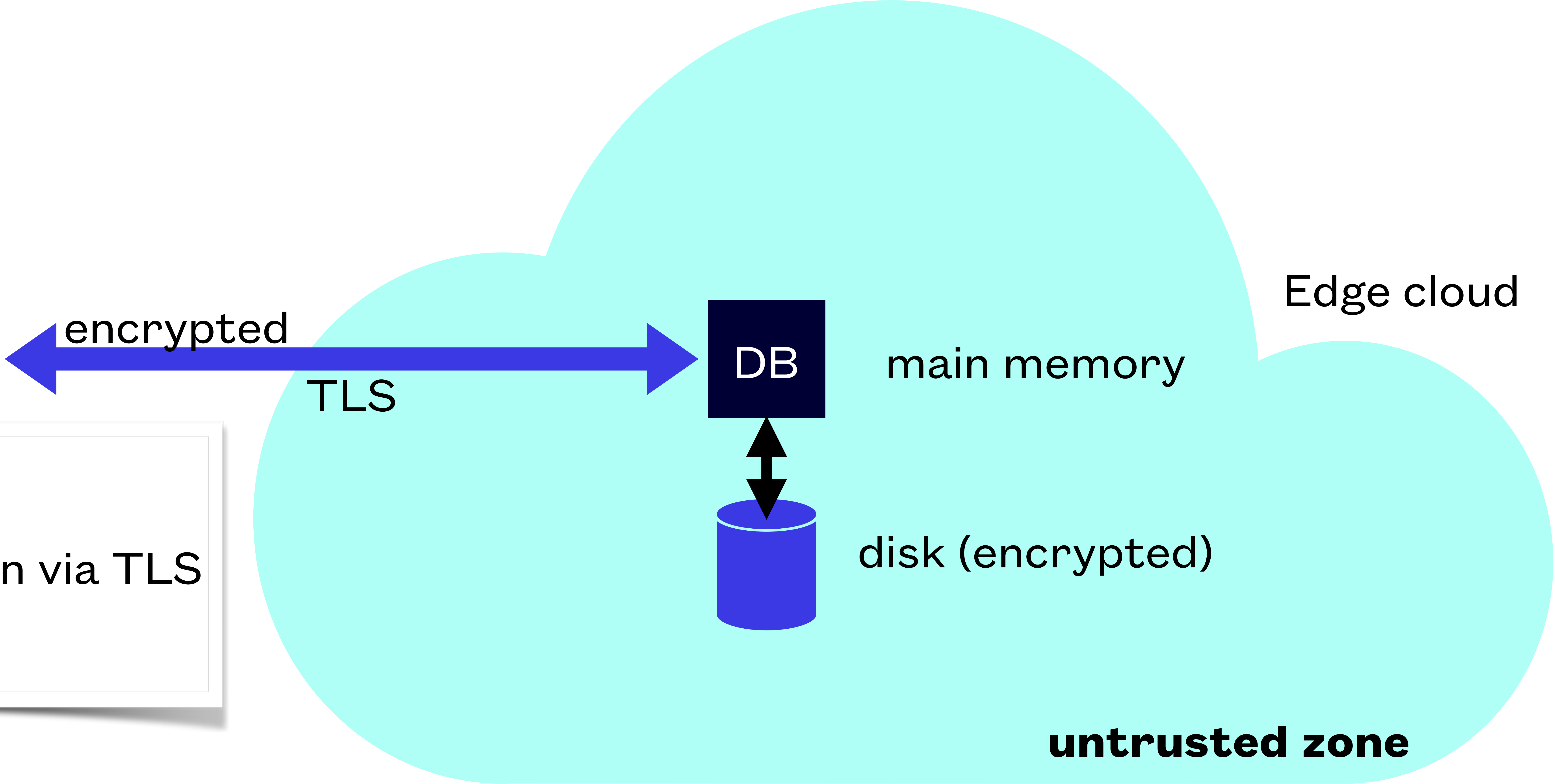
Adversary manages/operates services

SCONE

Example: Confidential, Managed DB Service

CHRISTOF FETZER, TU DRESDEN, CHRISTOF.FETZER@TU-DRESDEN.DE

Problem: How to protect database?



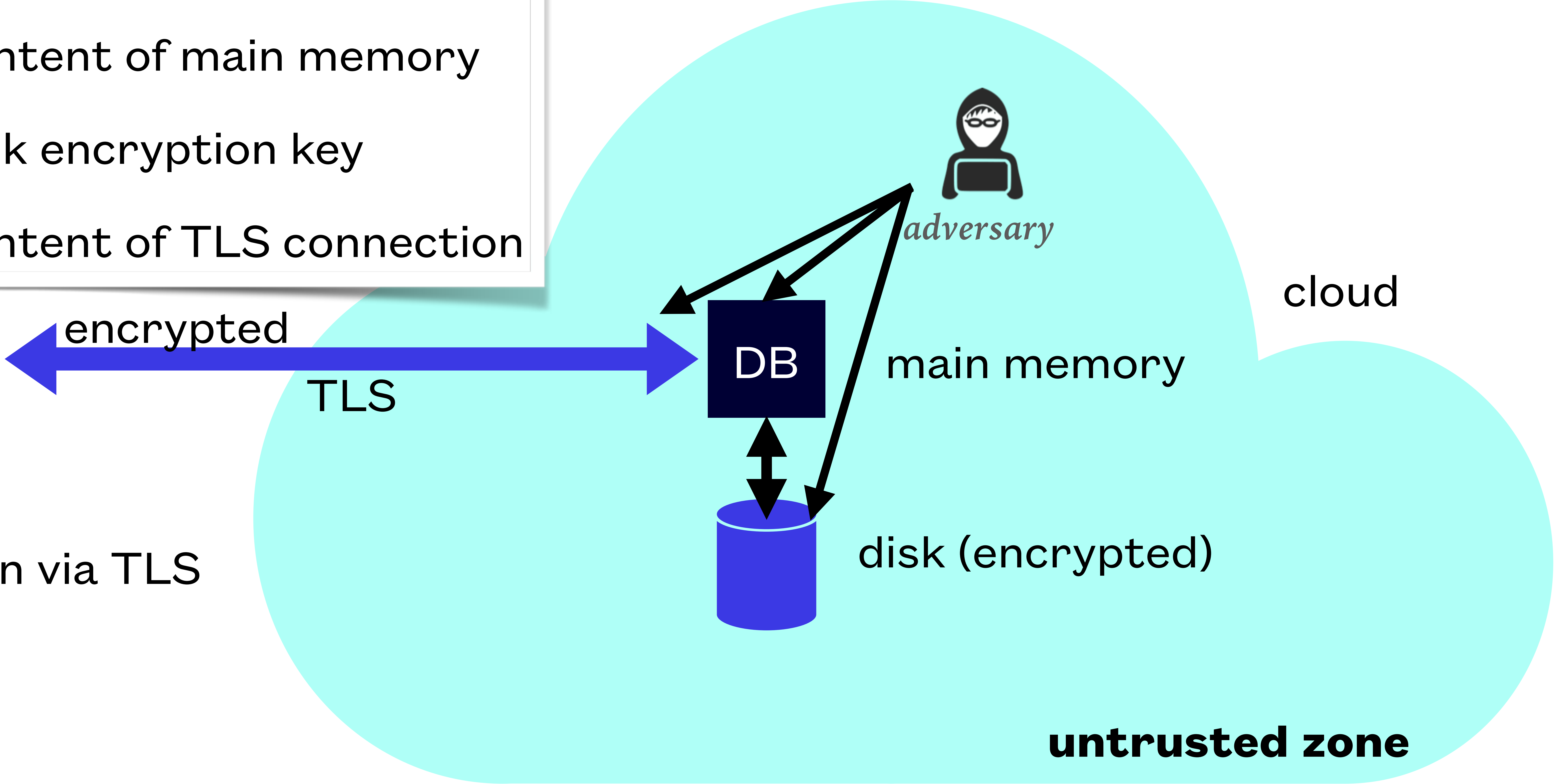
Standard:

- secure communication via TLS
- on disk encryption

Problem: How to protect DB in a cloud?

Problem: untrusted zone

- adversary can read content of main memory
- adversary can read disk encryption key
- adversary can read content of TLS connection



Standard:

- secure communication via TLS
- on disk encryption

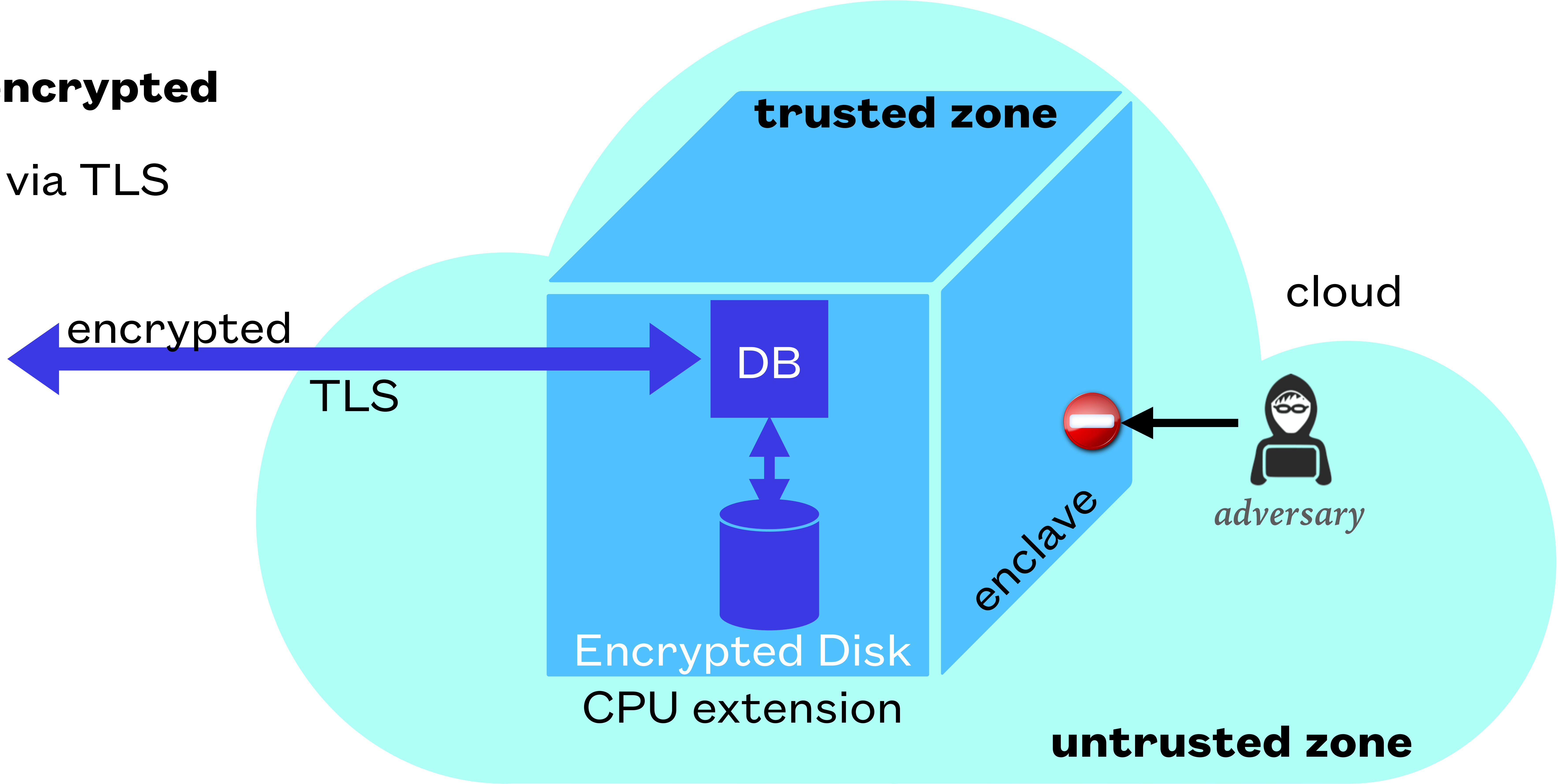
Approach: Execute in „Enclave“

Confidential Compute:

- main memory of DB encrypted

- secure communication via TLS

- on disk encryption

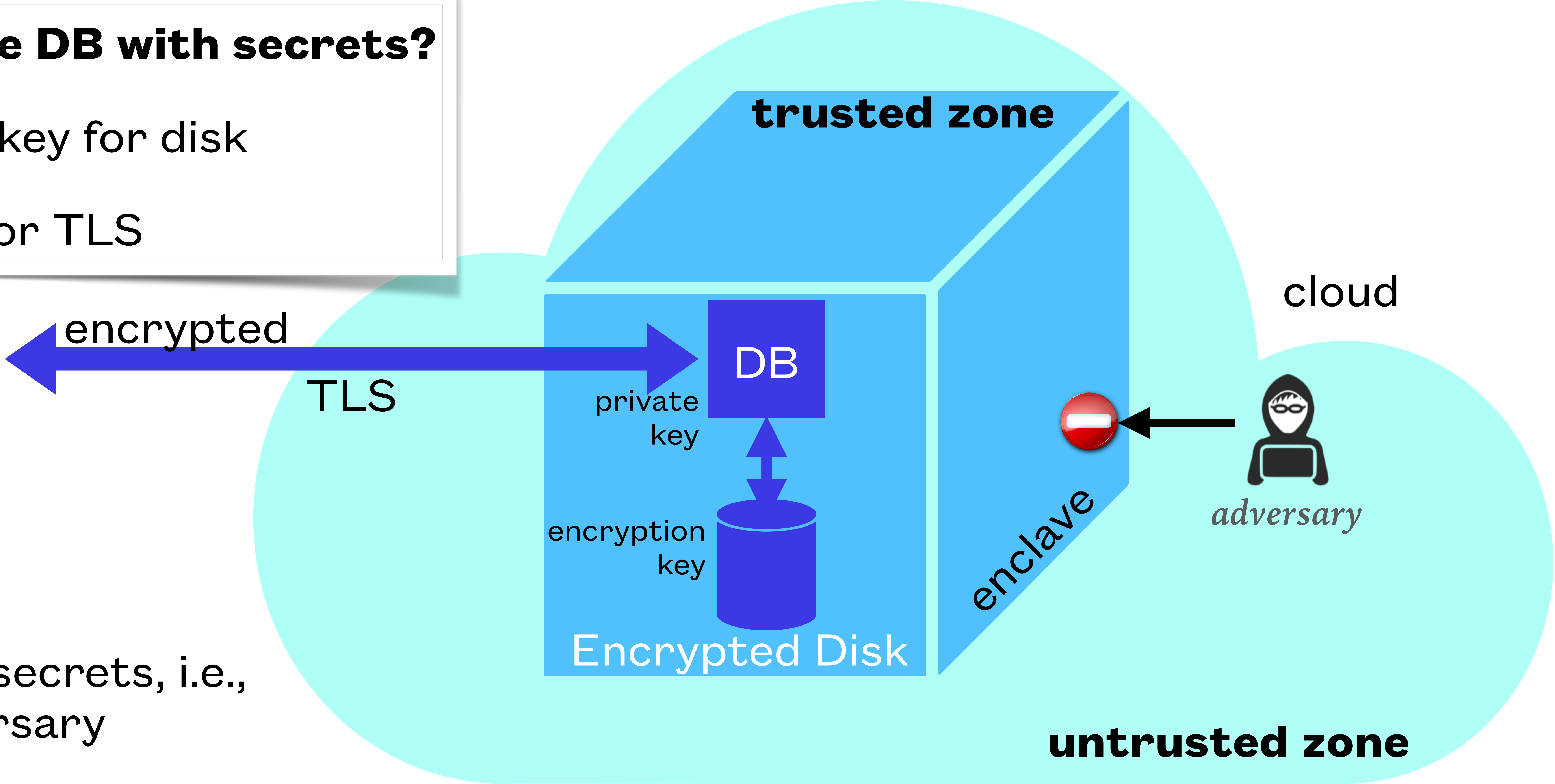


Enclave up to 64GB of main memory

Problem: Key Management

Problem: How to provide DB with secrets?

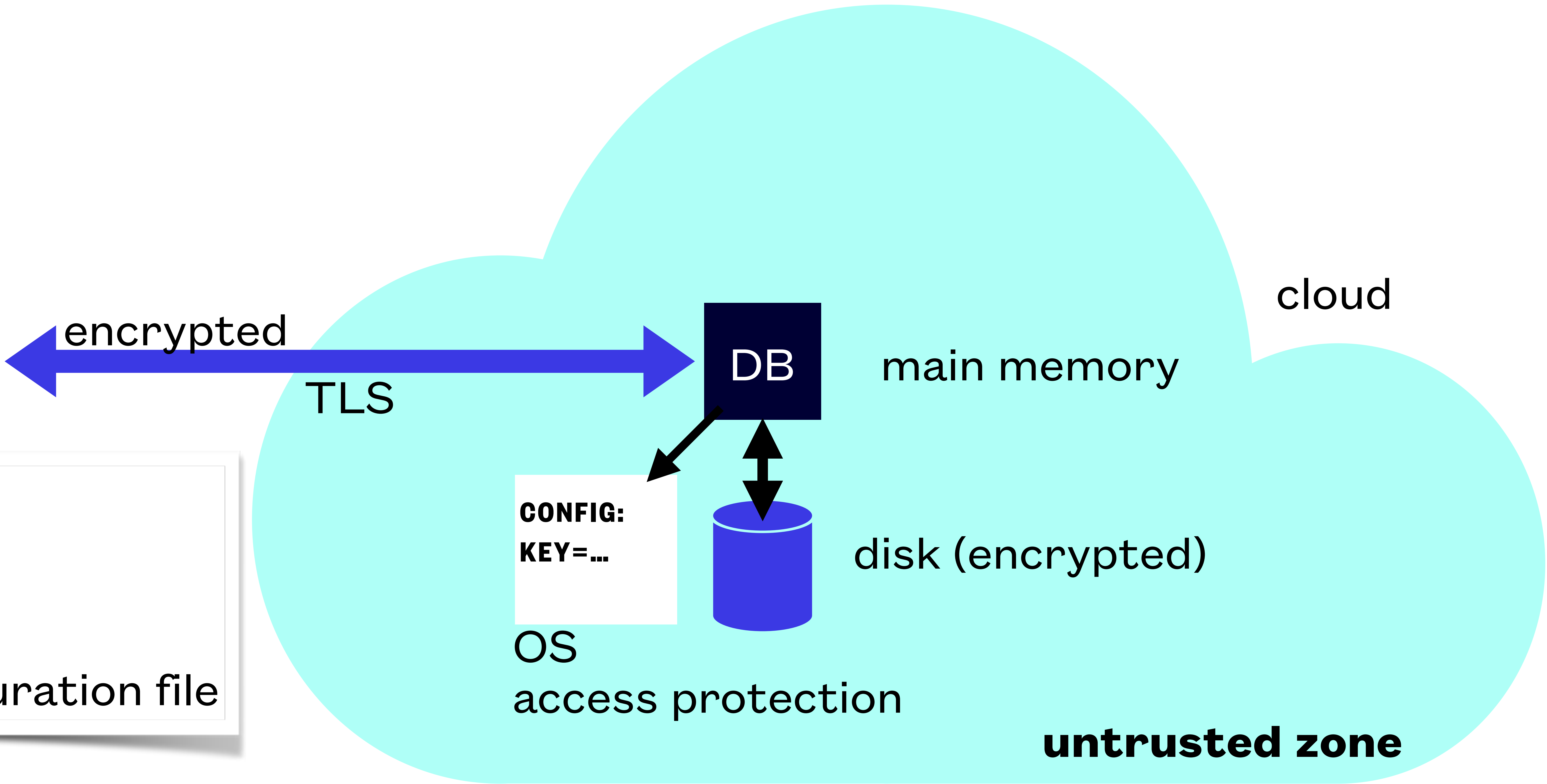
- We need an encryption key for disk
- We need a private key for TLS



Background:

- enclave starts without secrets, i.e., content known by adversary

Background



Standard:

- private key in file .key

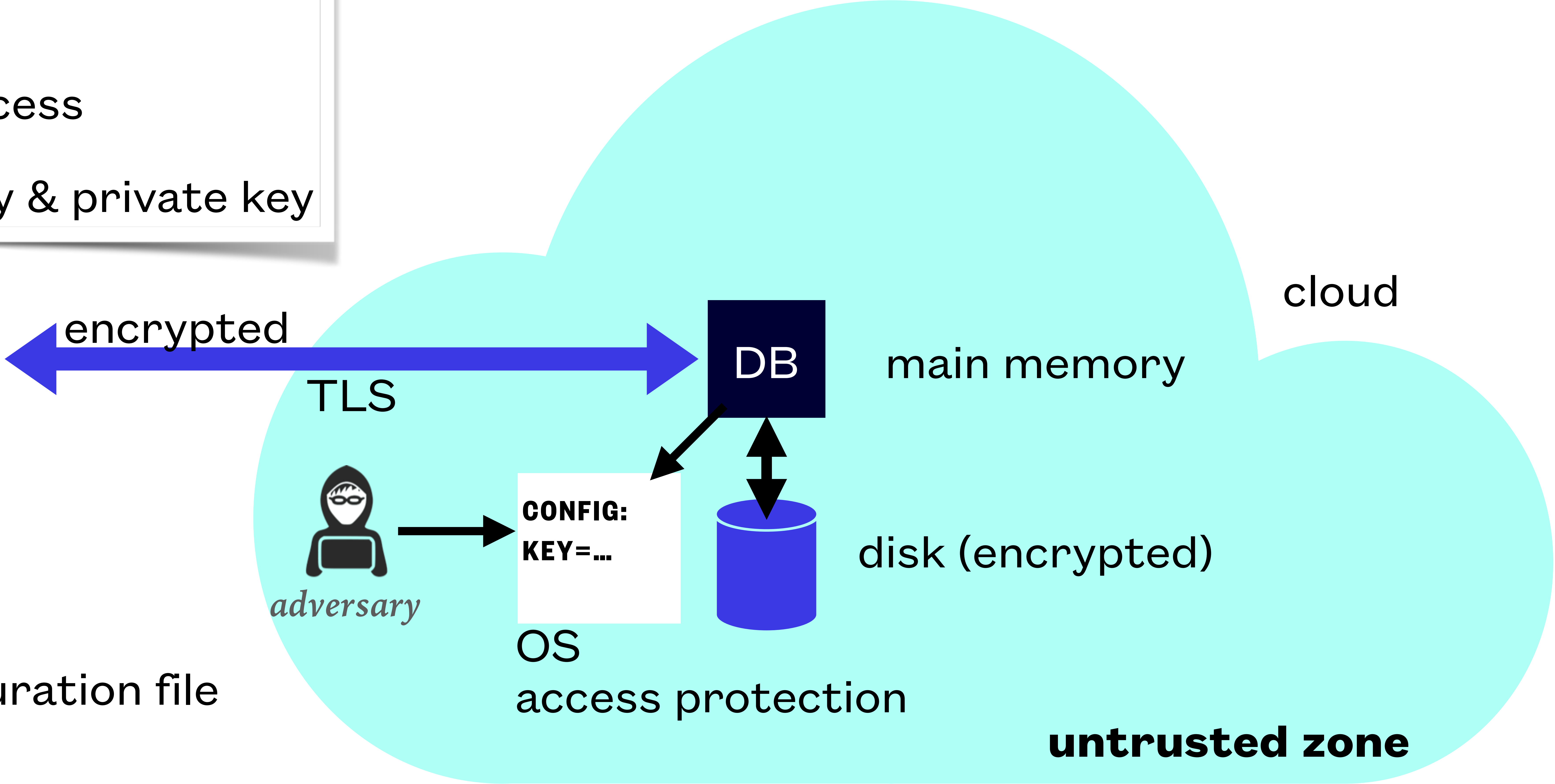
- encryption key in configuration file

Background

Problem:

- adversary with root access

- can read encryption key & private key



Standard:

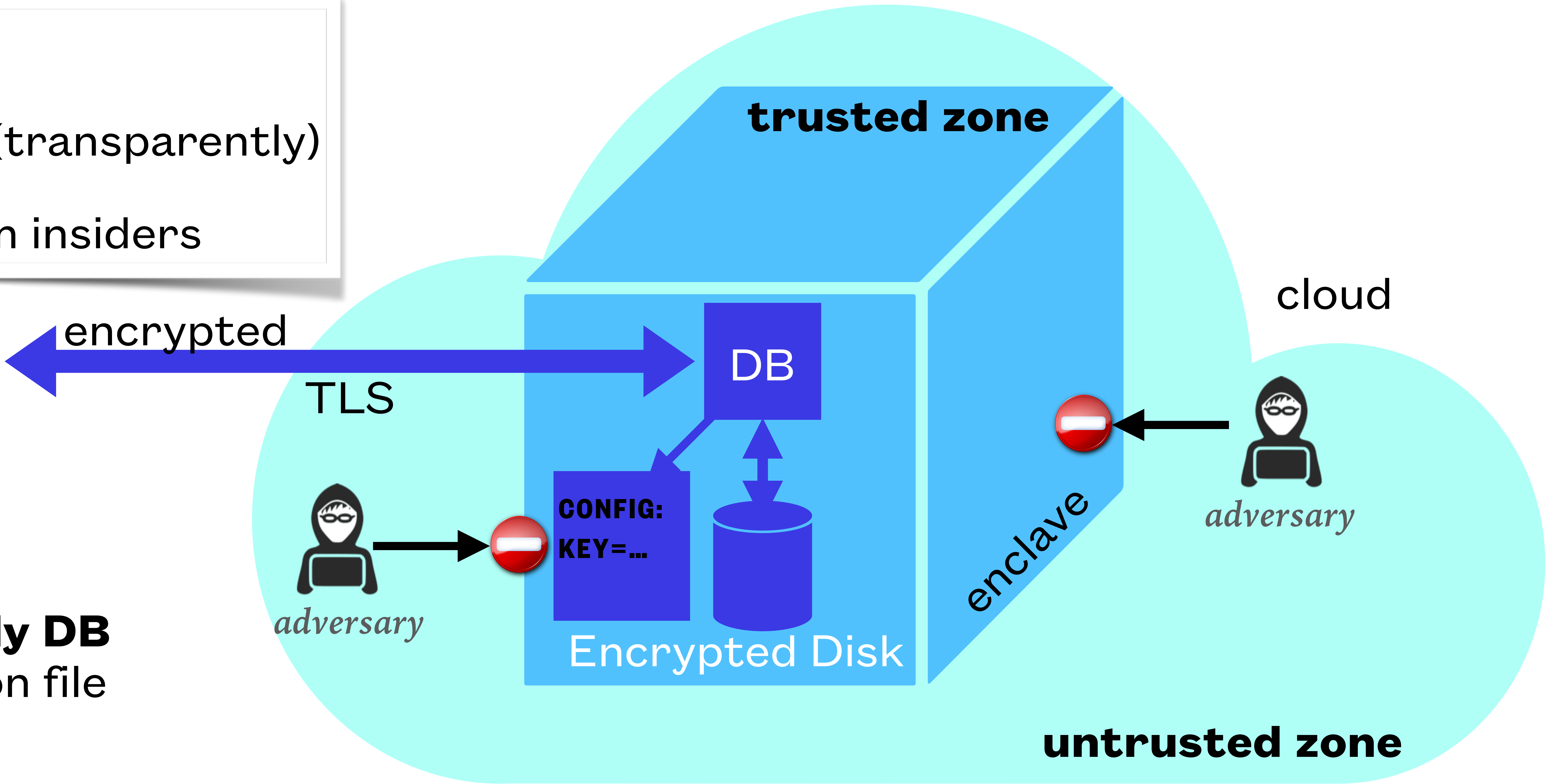
- private key in file .key

- encryption key in configuration file

SCONE Key Management

Approach:

- Files can be encrypted (transparently)
- Keys can be hidden from insiders



Background:

- SCONE ensures that **only DB** can decrypt configuration file

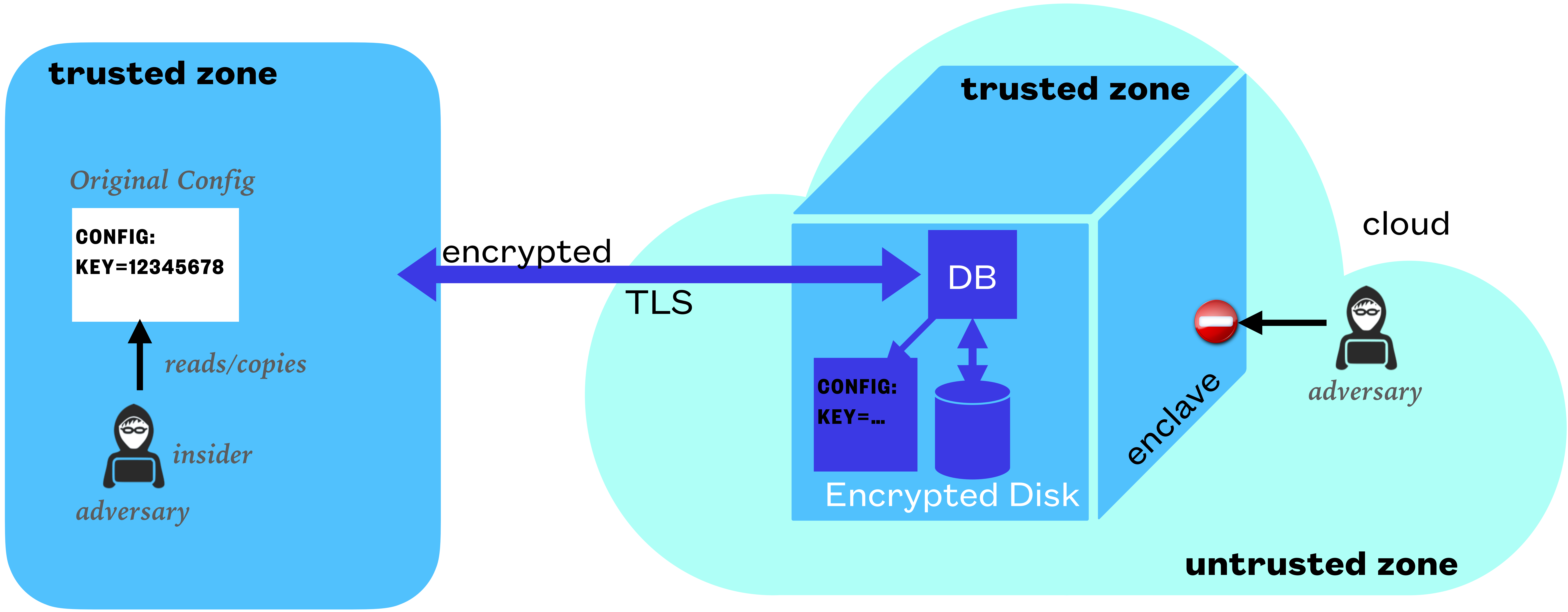
CHRISTOF FETZER, TU DRESDEN, CHRISTOF.FETZER@TU-DRESDEN.DE

SCONE

THREAT MODEL

INSIDER ATTACKS

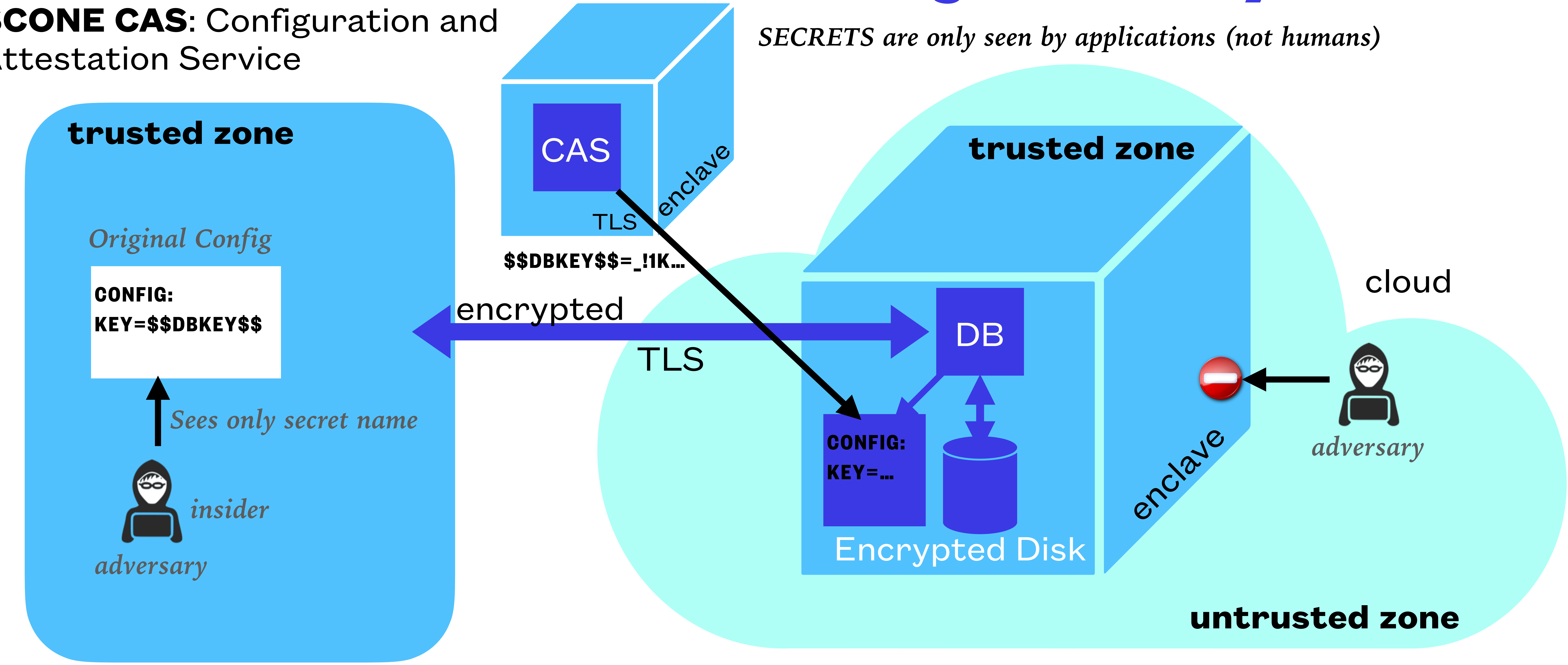
Threat Model: Insider Attacks



SCONE: Generate & Inject Keys

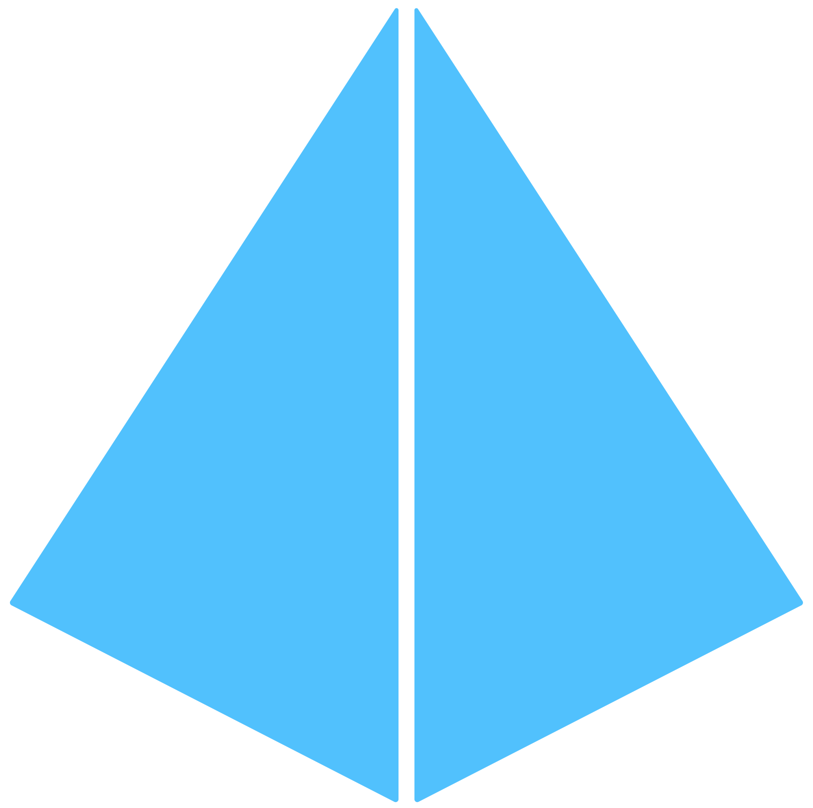
SCONE CAS: Configuration and Attestation Service

SECRETS are only seen by applications (not humans)



Better Trade-Offs Possible

Security



Outsourcing/ Sharing

Efficiency / Latency

Query Expressiveness

(equiality, conjunction, comparison, subset, range, wildcard)

- **Security**
  - end-to-end encryption & cryptographic access control
  - fine-grain access control against insider attacks
- **Outsourcing**:
  - provider cannot look into data
  - managed: provider can administrate service without seeing dat
- **Efficiency / low latency**: New SGC CPUs provides good performance
- **Query Expressiveness**:
  - Full query support

SCONE

Safety and Security

CHRISTOF FETZER, TU DRESDEN, CHRISTOF.FETZER@TU-DRESDEN.DE

Safety and Security

- **Problem: Critical Applications**
  - Ensure **confidentiality** of input, output and all data during processing
  - Ensure **integrity** despite **execution errors** (e.g., bit flips)
- **Approach**:
  - **ILR** (instruction-level redundancy): all instructions are executed twice
    - Similar to lock-step execution but in software on a single core
    - Applicable to modern CPUs (non-deterministic, no lock-step support)

SCONE Approach

- **Components:**
  - **Compiler-based approach**: application recompiled for ILR and to run inside of enclaves
  - **Attestation**: remote clients can ensure that they communicate with correct application

- **Automotive applications**:
  - Could offload even some critical (fail-stop) service in an edge cloud

Evaluation

- **Fault Injection:**
  - **Undervolting**: results in crash or wrong results (requires old firmware)
  - **ILR**: detects wrong executions and exits program

- **Overhead**:
    - Much lower than replicated execution on two CPUs or two hosts

SCONE

Confidential Cloud-Native Apps

CHRISTOF FETZER, TU DRESDEN, CHRISTOF.FETZER@TU-DRESDEN.DE

Cloud-Native Applications

- **Confidential Cloud-Native Applications**:
  - application consisting of multiple microservices
  - implemented in multiple programming languages and
  - SCONE supports the programming languages (compiled, JIT and interpreted)
  - One can run standard Kubernetes-based application - inside of SGX enclaves
  - One can achieve good performance and very good security
- **State of the art development support**
  - Integration with common CI/CDs and container-based deployment

SCONE

Summary

CHRISTOF FETZER, TU DRESDEN, CHRISTOF.FETZER@TU-DRESDEN.DE

Summary

- **Customer** can protect
  - **confidentiality** and
  - **integrity**
- of managed applications using SCONE-based **confidential compute**
- **Edge cloud provider** manages services and ensures
  - **Availability**
  - **High bandwidth**
  - **Low latency**

sconedocs.github.io

Questions?