CONFIDENTIAL COMPUTING

Some basic concepts and terminology

Prof. Dr. Christof Fetzer

Confidential Computing: Application Domains

CLOUD COMPUTING

- Protect cloud-native applications
- Protect AI applications
- Protect cloud services itself

EDGE CLOUD

- Protect applications and cloud
    - despite limited physical security

AIR-GAPPED SYSTEMS

- Protect applications despite having no connectivity

EMBEDDED SYSTEMS

- Protect devices with limited connectivity & limited physical security

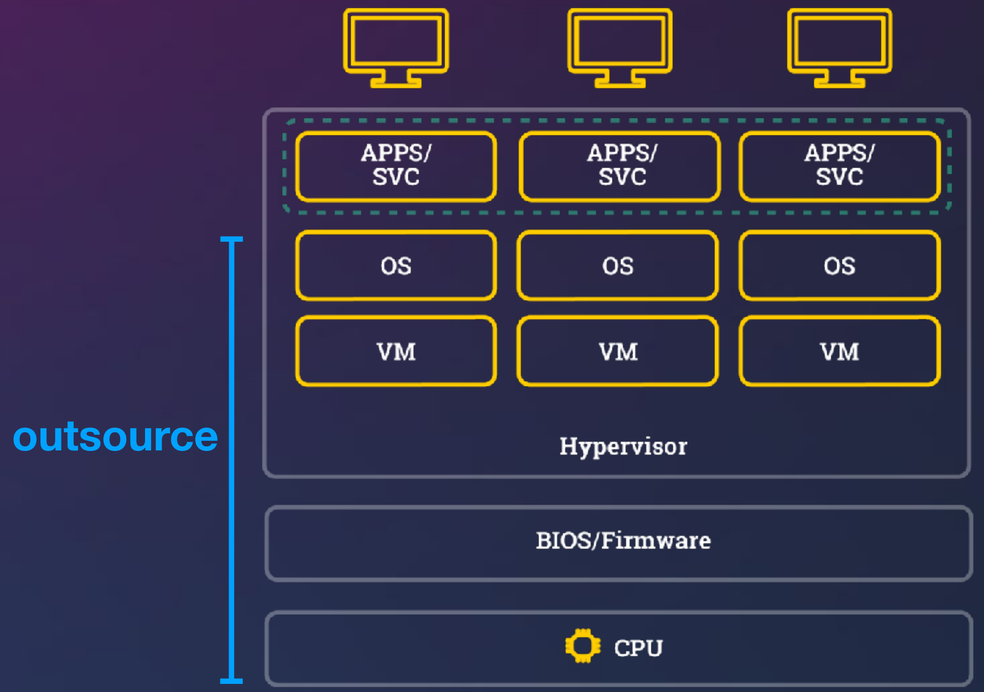
eHEALTH DOMAIN

- e.g., protect patient data

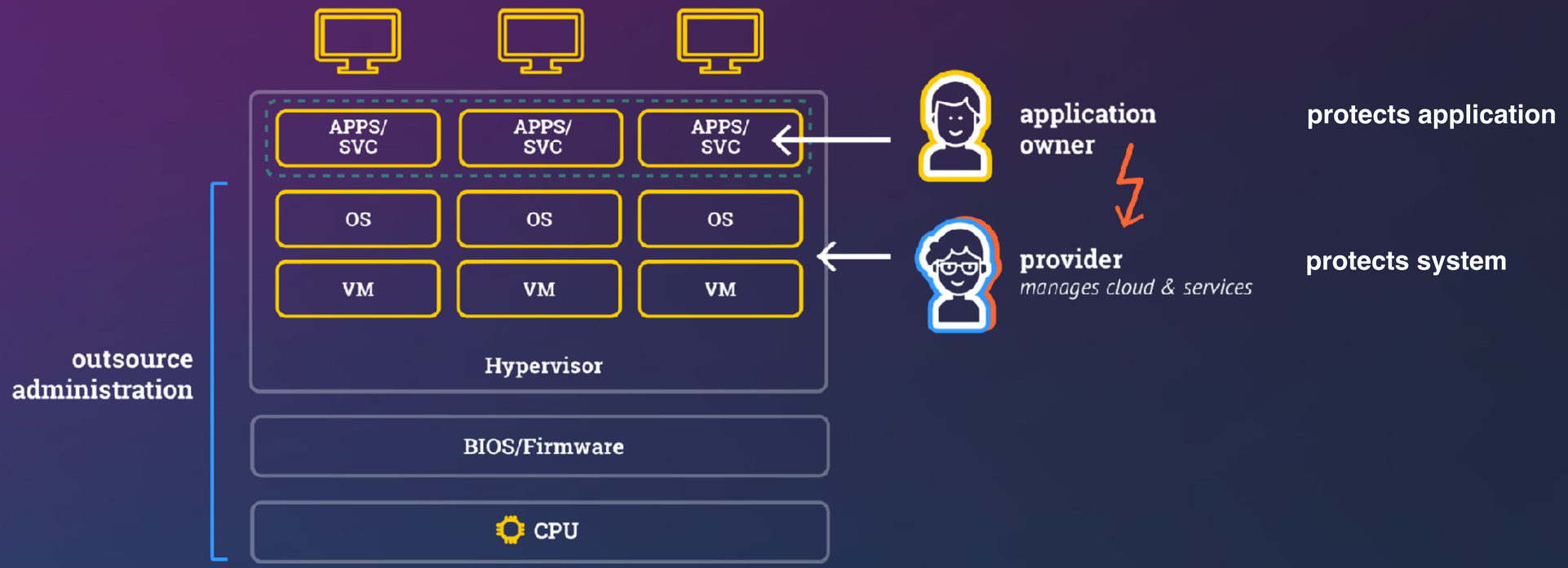
Cloud: Outsource Infrastructure

Operating one's own computing infrastructure is not easy:

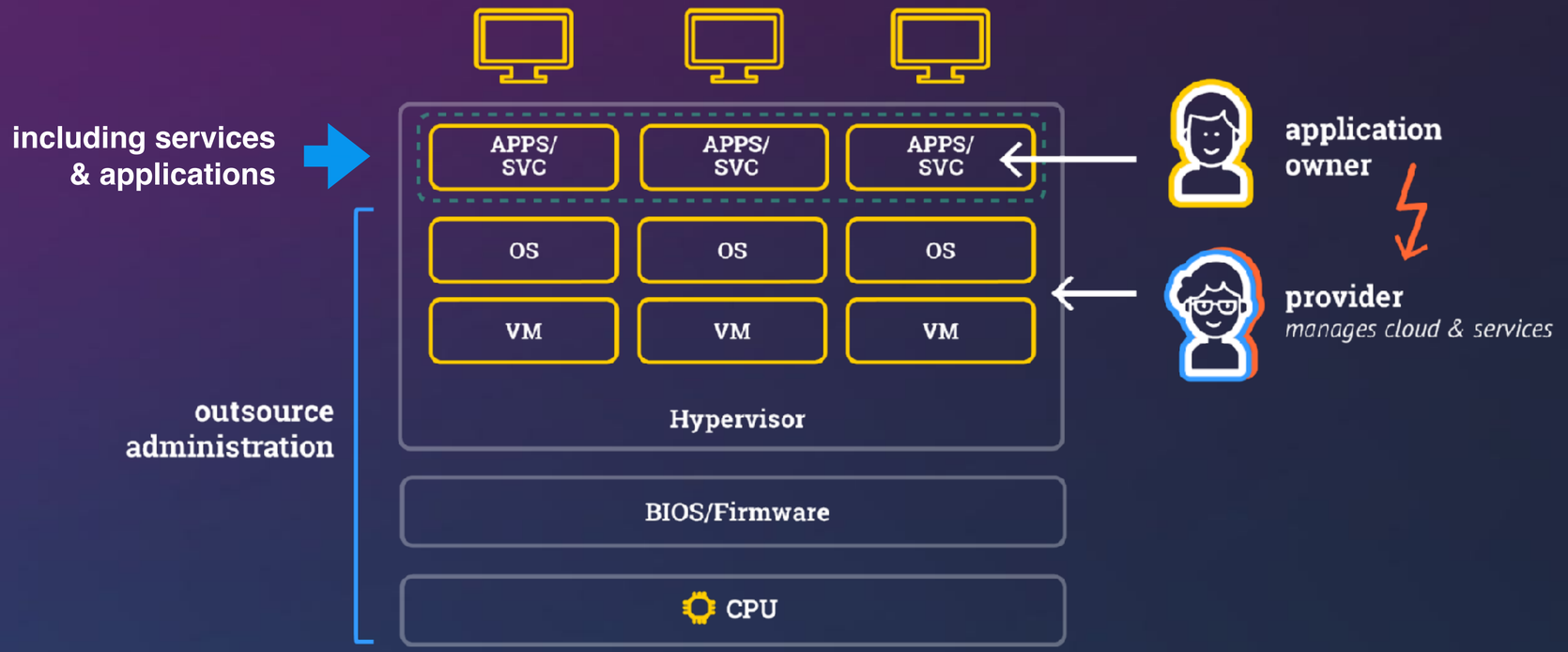
- data center
- hypervisors
- operating systems
- Kubernetes
- services
- ...



Different Stakeholders



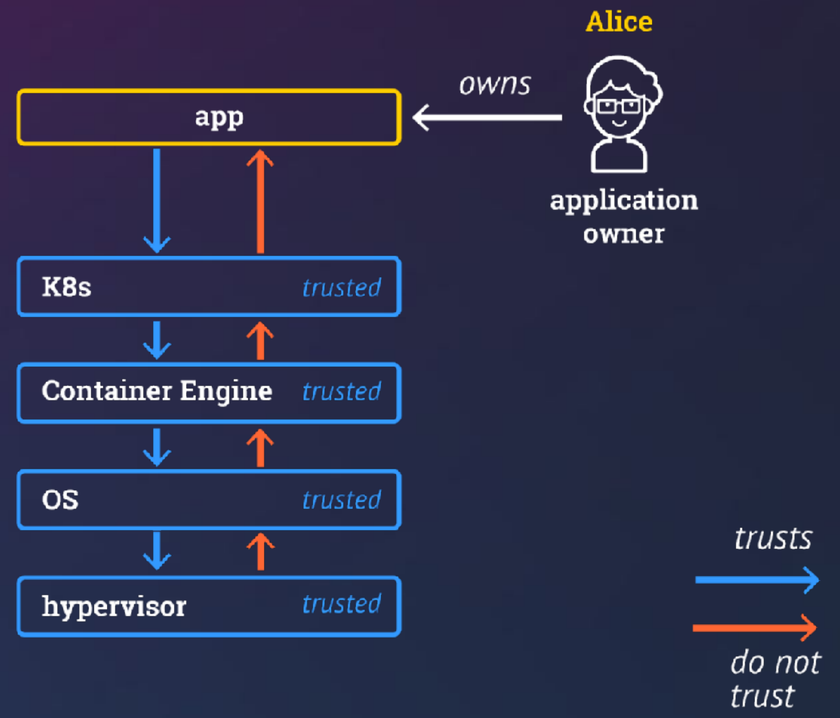
Confidential Managed Services



OBJECTIVE:
Outsource management of services and applications
provider must not have access to data / code / secrets.

Systems Security

- **Systems are structured in layers**
  - like operating system and hypervisor
- **Typically, systems security is bottom up**
  - layer i does not trust layer i+1
  - but layer i+1 trusts layer i
- **Examples:**
  - the operating system trusts the hypervisor
  - but the hypervisor does not trust the operating system

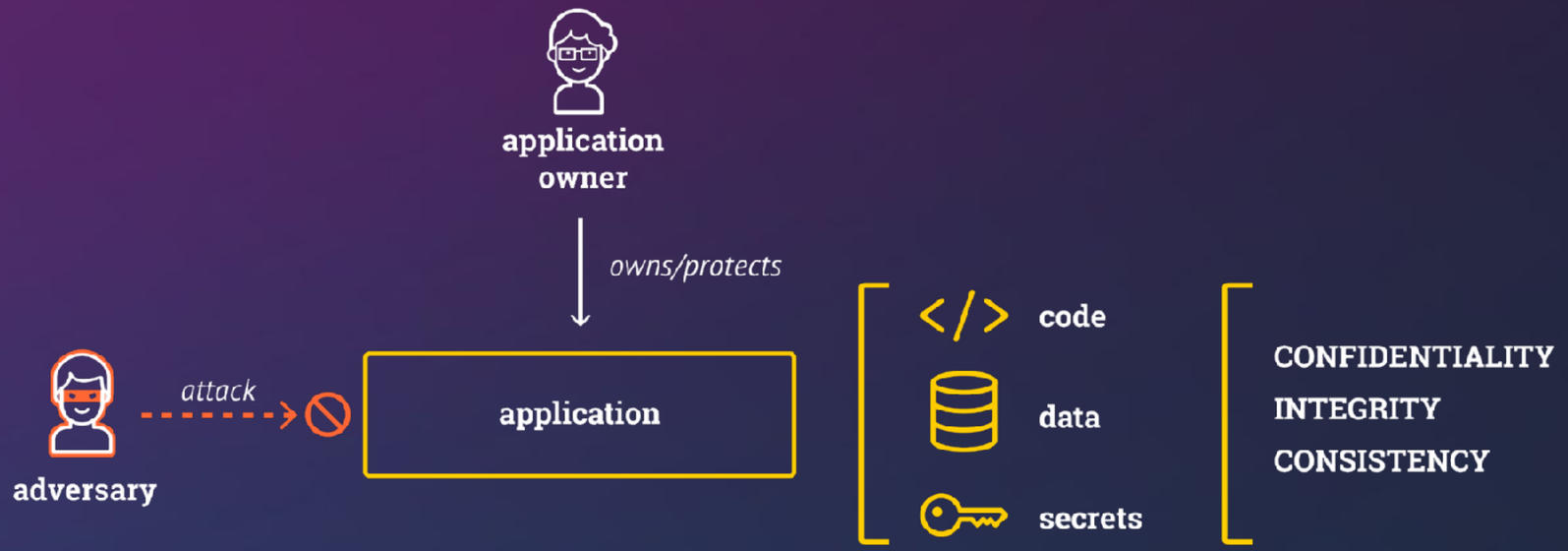


OBJECTIVE:
Protect system

Confidential Computing: Protect Applications

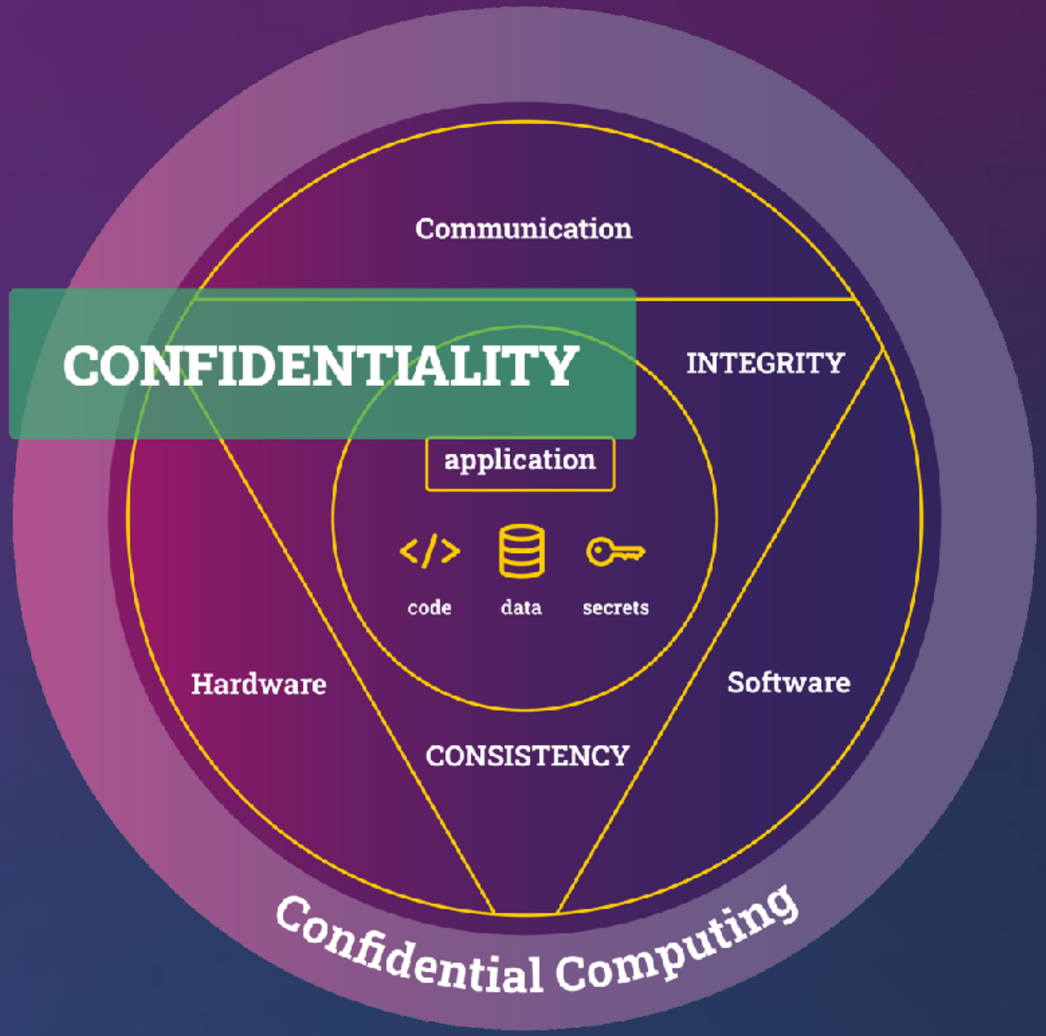
„application-oriented security“

Protect Code, Data, and Secrets



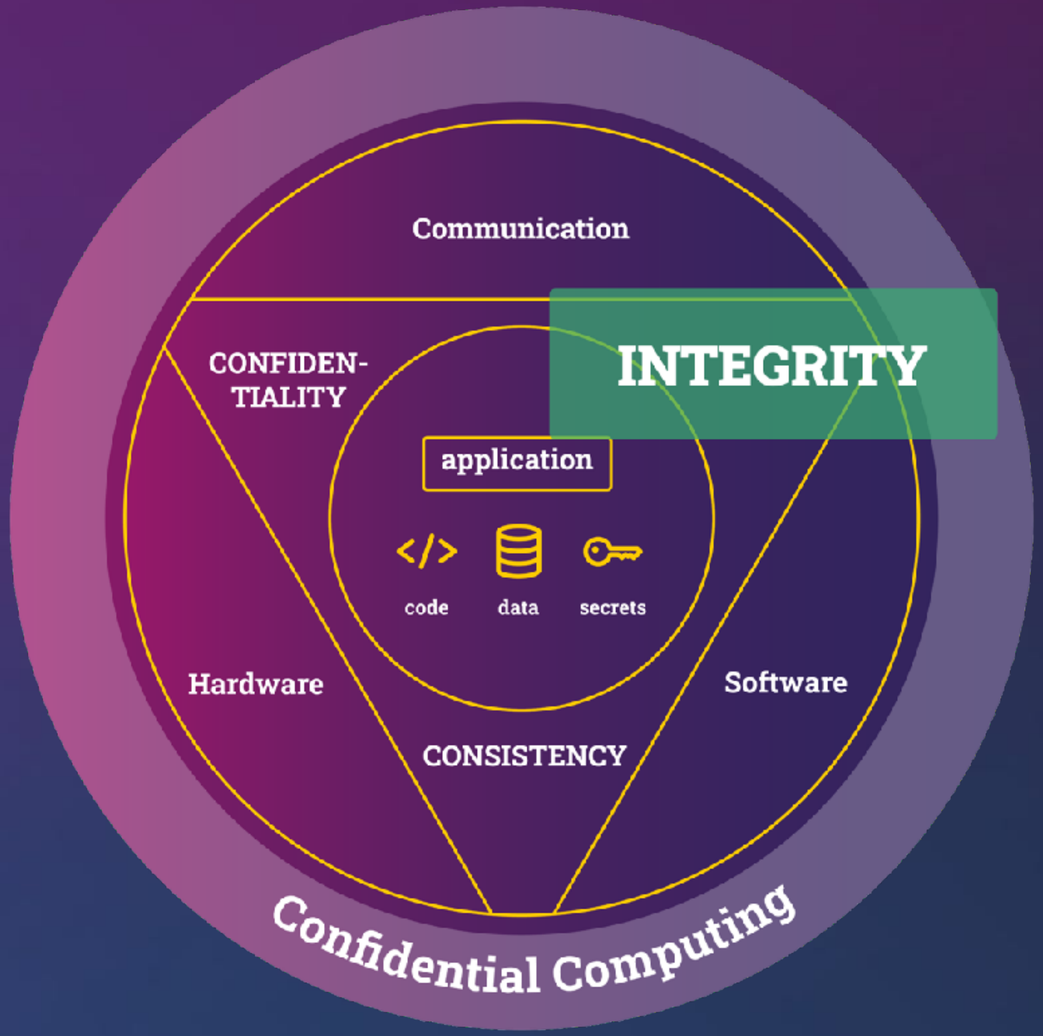
„application-oriented security“

Confidentiality



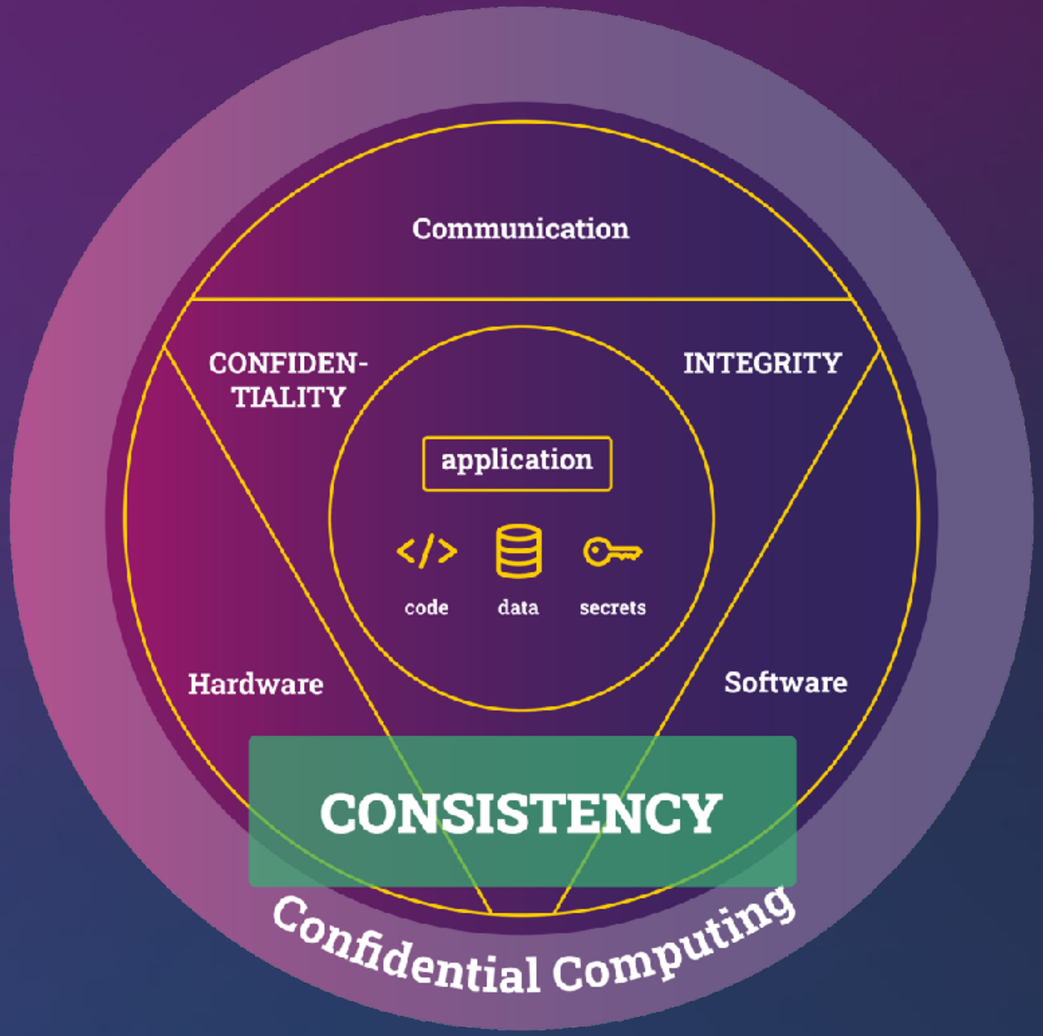
Confidentiality *is the property, that information (data, code, secrets) is not made available or disclosed to* ***unauthorized*** *individuals, entities, or processes.*

Integrity



Integrity *means that information (i.e., data, code, secrets) cannot be modified in an* ***unauthorized*** *or* ***undetected*** *manner.*

Consistency



Consistency *means that one always reads the latest information (i.e., data, code, secrets) written by an* ***authorized*** *entity.*

Detect if an adversary would provide an old copies (which are correctly encrypted but that have been updated).

PROBLEM:

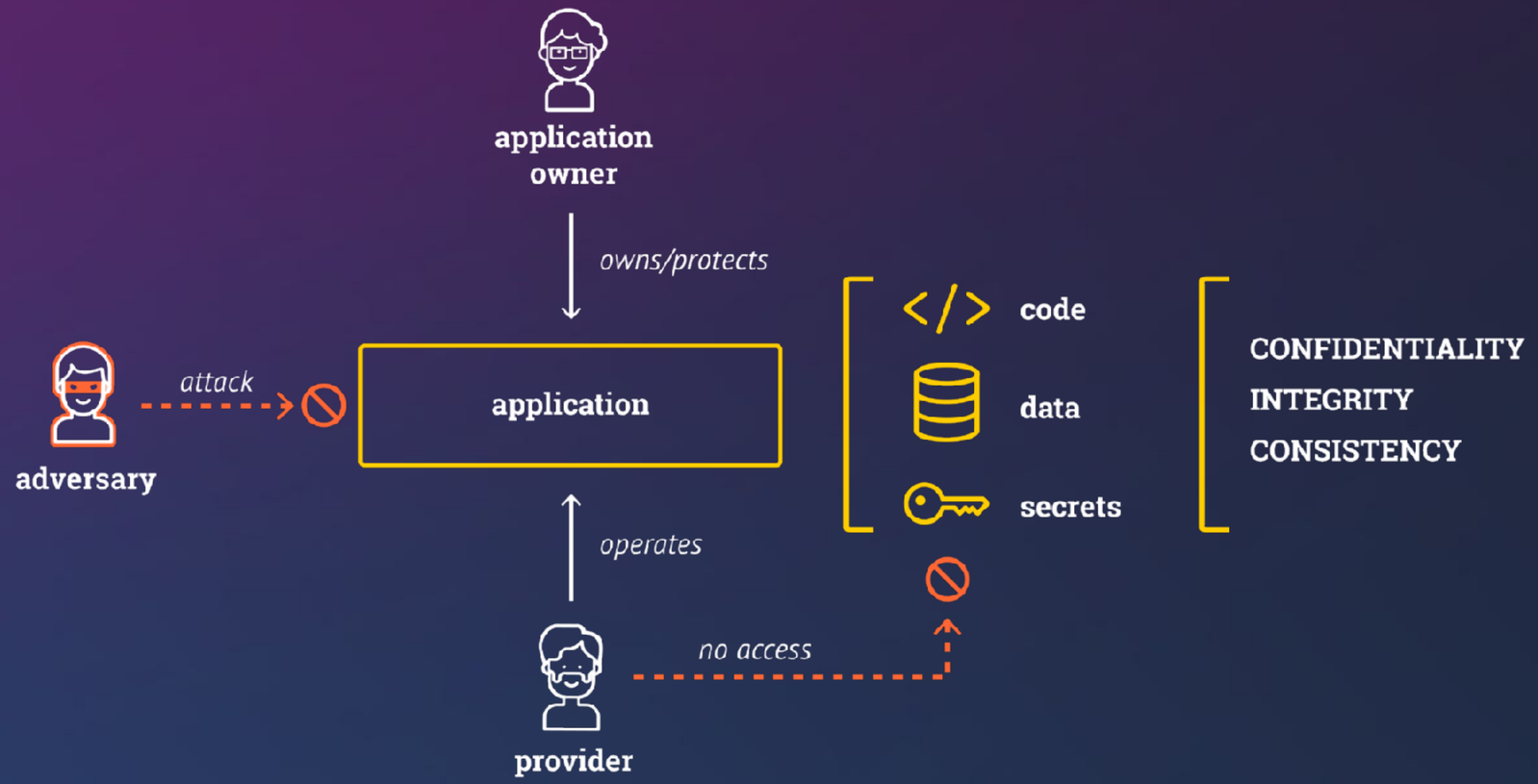
How to define who is authorized
if one cannot trust the operating / hypervisor / ... system?

Protection Goals of Confidential Computing

- **Confidentiality:**
  - only authorized users/programs can read
- **Integrity:**
  - only authorized users/programs can update
- **Consistency**
  - always accessing the last version

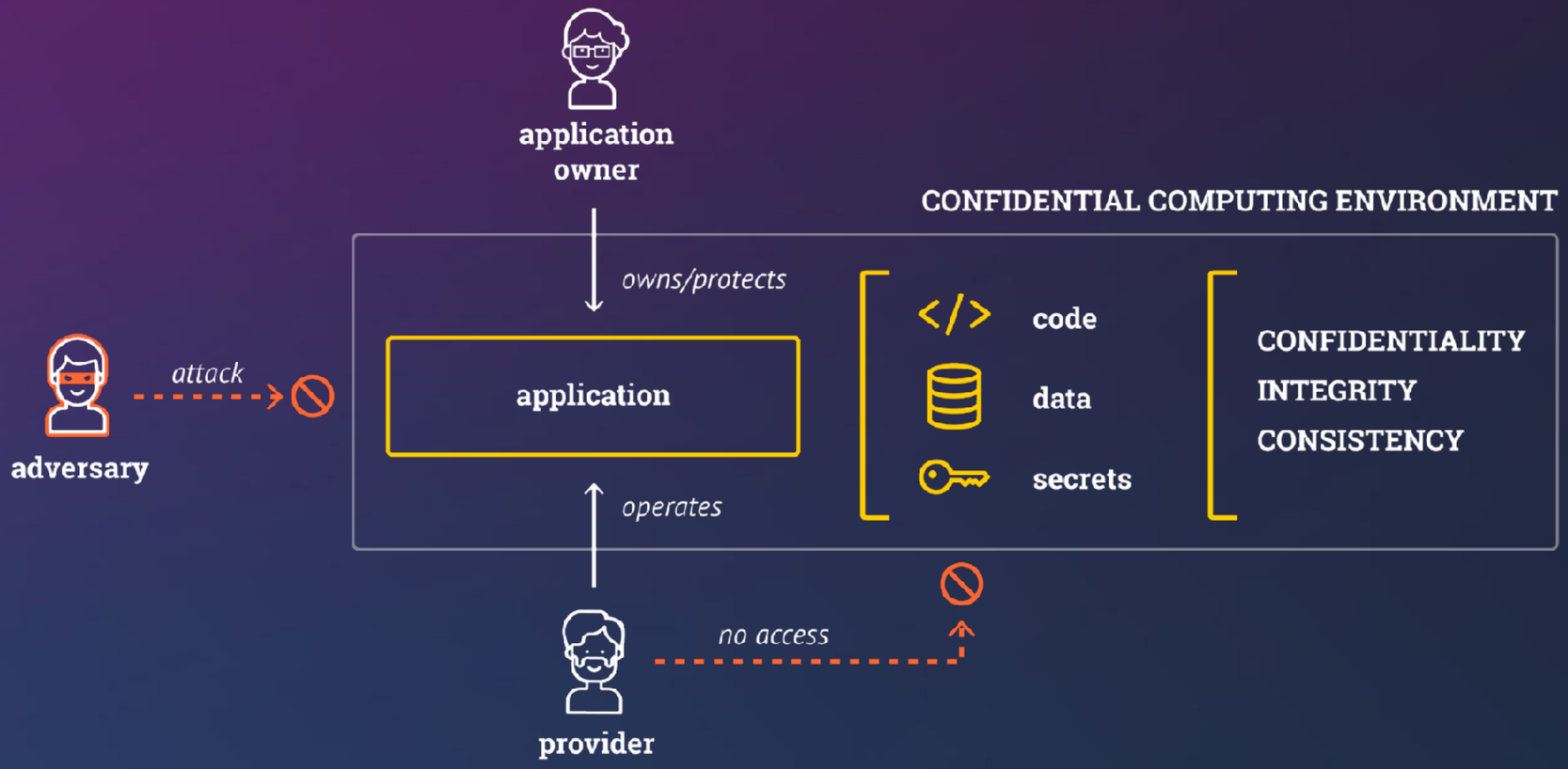
Who is authorized?

Protect against Adversaries and Cloud Provider!

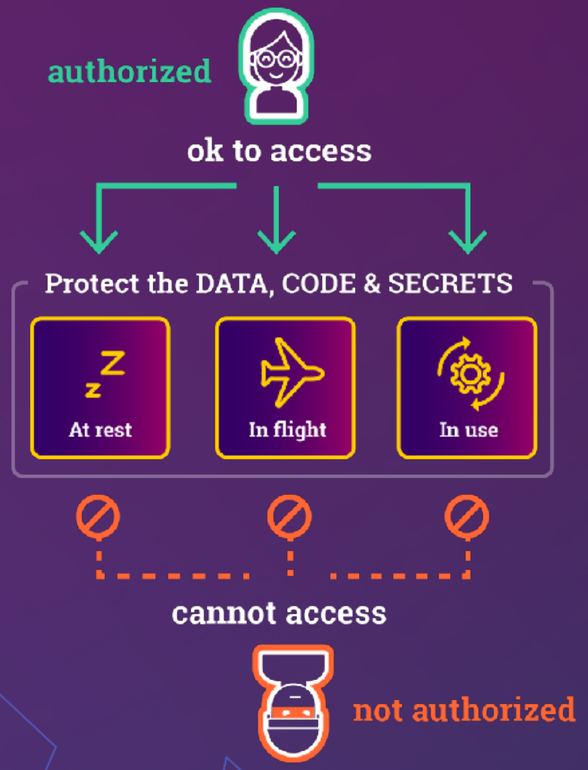


„application-oriented security“

Approach: Confidential Computing



„application-oriented security“



Protection in Use

- i.e., in main memory -

PROBLEM:

How to ensure confidentiality, integrity and consistency if adversary has root and hardware access?

PROBLEM:

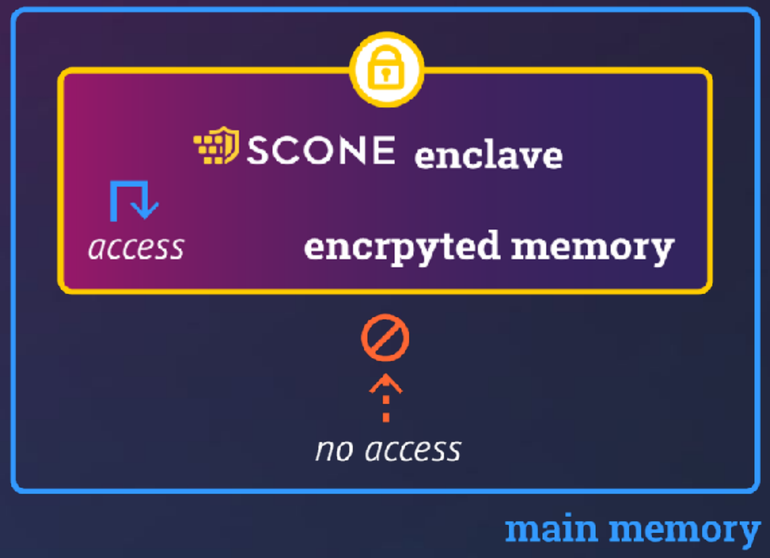
How to ensure confidentiality, integrity and consistency if adversary has root and hardware access?

APPROACH:

Run applications in Trusted Execution Environments (TEE) exclude access by root users!

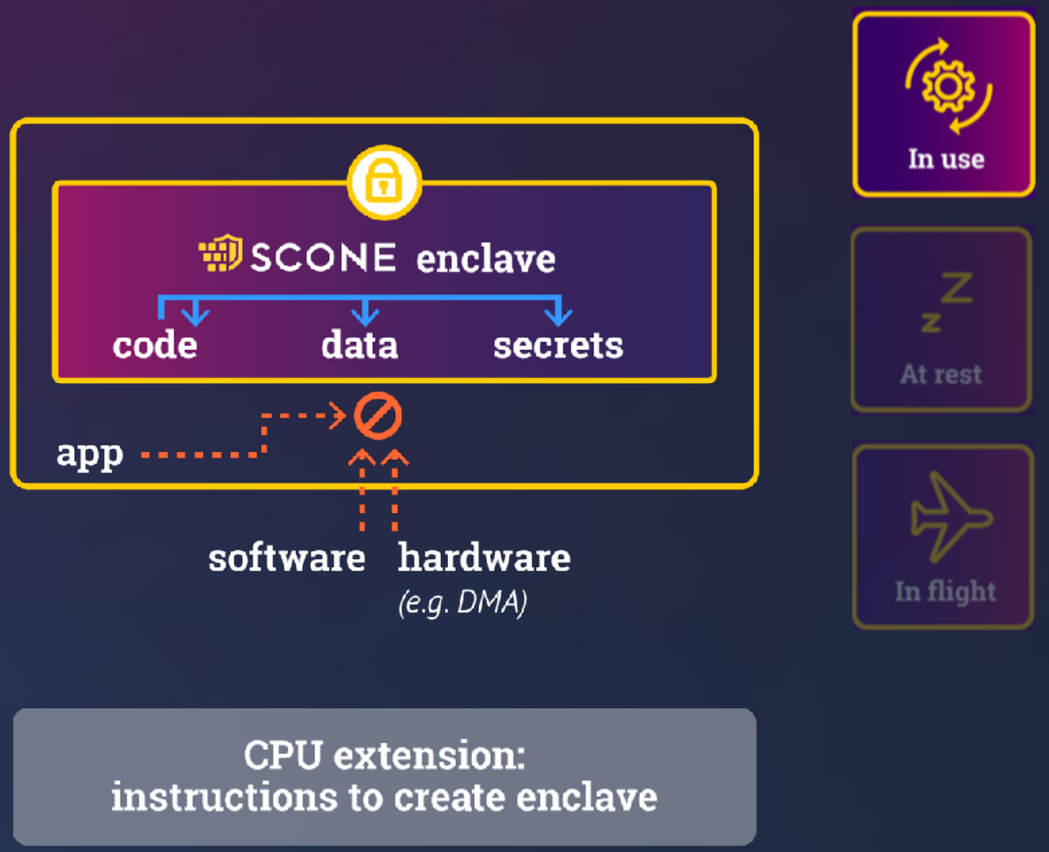
Trusted Execution Environment (TEE)

- Enclaves are a TEE:
  - CPU instruction set extension
  - service runs inside an encrypted memory region, aka, **enclave**
    - **random key** — generated by CPU, or
    - **external key** — given by external entity - **can we trust this key**?
  - only code running inside of enclave can see content
- **Hardware with different granularities:**
  - **enclave**: process runs inside encrypted memory (Intel SGX)
  - **encrypted VM**: a whole VM (AMD SEV, ARM Realms, Intel TDX)
    - can be used to implement enclaves



Enclaves

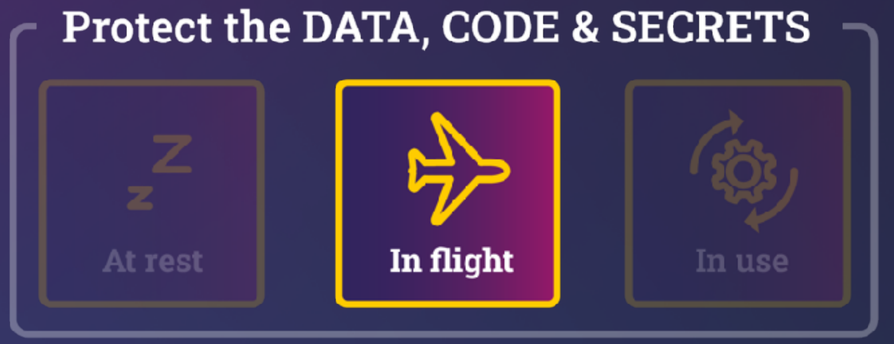
- **Protect data/code/secrets in use**
  (i.e, in main memory):
  - run application code in encrypted memory region (aka **enclave**)
  - only code in enclave can access region



SCONE: Secure CONtainer Environment
- platform for Confidential Computing

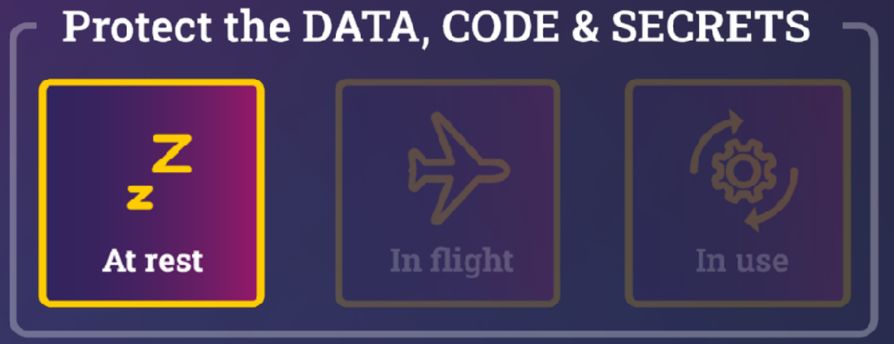
https://sconedocs.github.com

Protecting Data in Flight: SCONE Network Shield



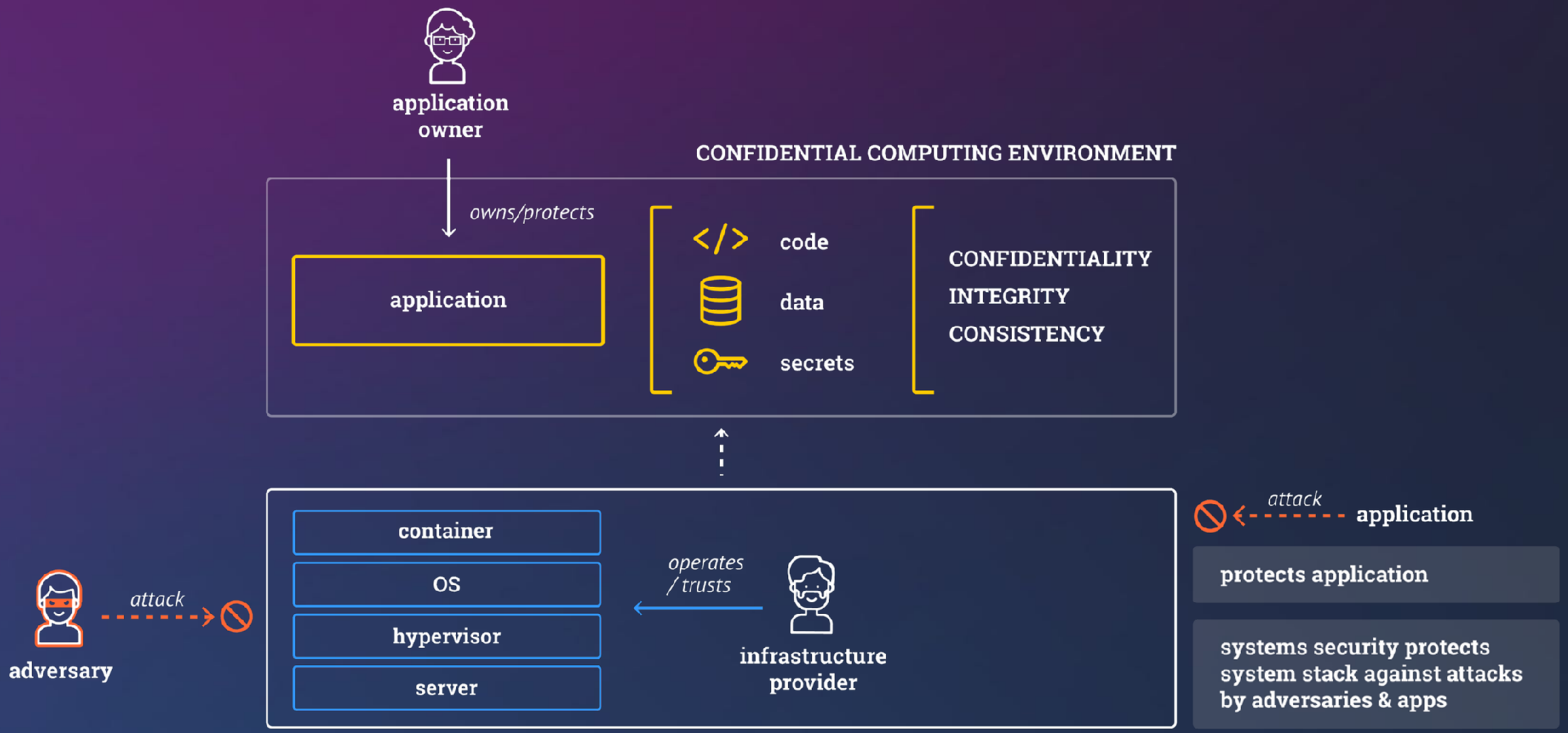
introduced in a later section

Protecting Data at REST: SCONE Fileshield



introduced in a later section

Complementary: Systems Security + Confidential Computing



Availability

- **DEFINITION:**
  - The probability that a system/ application / service will operate satisfactorily at a given point in time when
- **EXAMPLES:**
  - 99.99% (4 nines availability)
  - 99.999% (5 nines availability)
  - 99.9999% (6 nines availability)

Durability

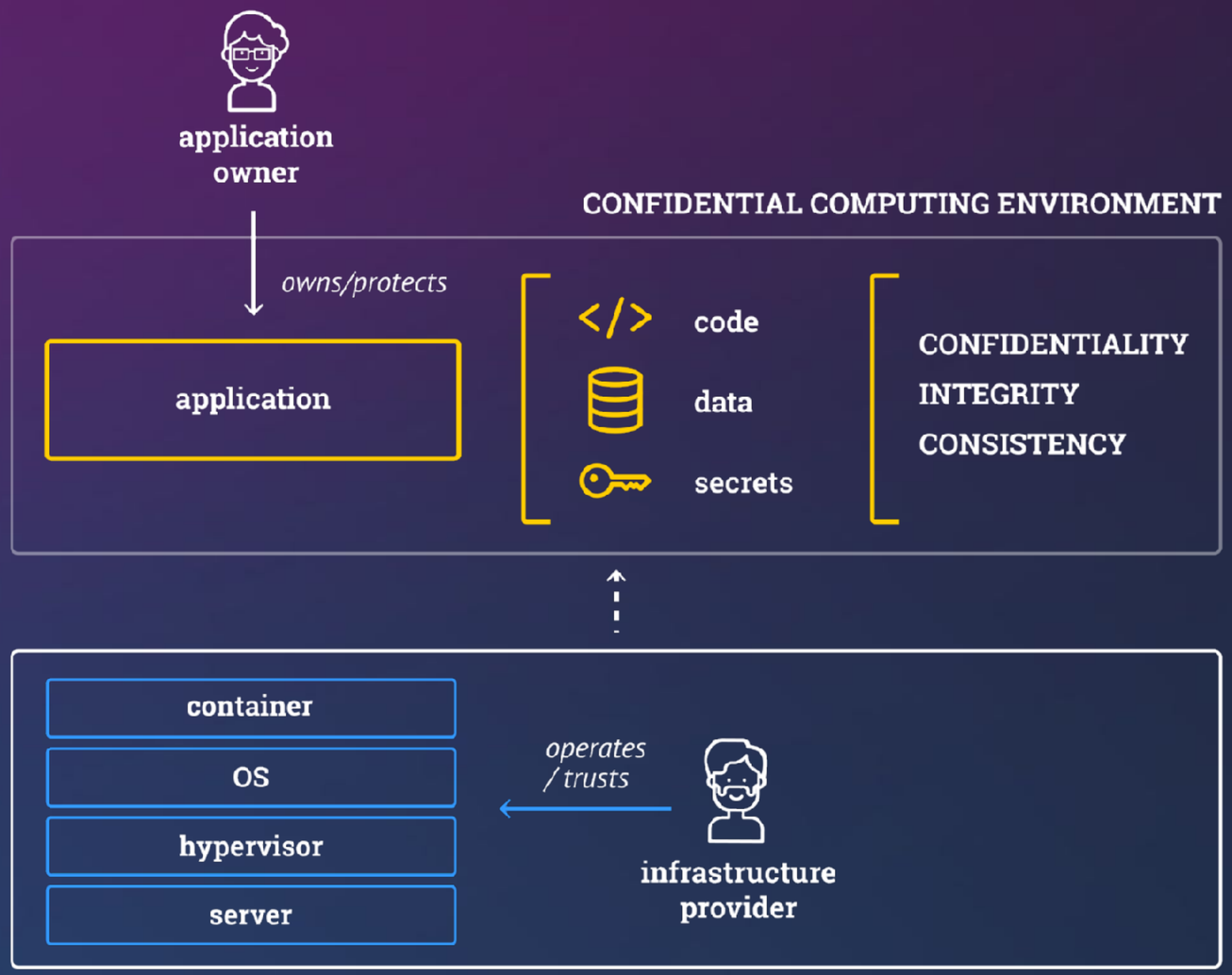
- **DEFINITION:**
  - The probability that a data item will be accessible after one year
- **EXAMPLES:**
  - 99.999999999% (11 nines durability)

Availability and Durability

- **Protection goals NOT addressed by confidential compute:**
  - **Availability**: the probability that information is available when it is needed
  - **Durability**: the probability that information will survive for one year
- **Approach**:
  - address this via SLA (Service Level Agreements) of provider
  - use **automatic reconciliation** of the services

introduced in a later section

Availability



General Approach:
we measure **availability** and **durability &** define SLA with provider

Implementation:
we use reconciliation to improve availability of services

Note:
Confidential Computing does not address availability nor durability

(SLA = Service Level Agreement)

Example

What information to protect?

- Protection of
  - **Code**, e.g., modern AI programs written in Python
  - **Data**, e.g., training data to create AI models
  - **Secrets**, e.g., key used to encrypt database

What information to protect?

- Protection of
  - **Code**, e.g., modern AI programs written in Python
  - **Data**, e.g., training data to create AI models
  - **Secrets**, e.g., key used to encrypt database
- **Example**:
  - *MariaDB* supports encryption of database
  - encryption key is stored in configuration file
  - configuration file protected via access control:
    - i.e., can be read and written by MariaDB (user) as well as any root (=privileged) user

Is Operating System access control sufficient?

- Sufficient if we define that
  - only **authorized user** can become root users

Is access control sufficient?

- Sufficient if we define that
  - only **authorized user** can become root users
- What about
  - **adversary** gaining root access (e.g., stealing credentials)?
  - **authorized** user laid off -> could become an **adversary**?

Is access control sufficient?

- Sufficient if we define that
  - only **authorized user** can become root users
- What about
  - **adversary** gaining root access (e.g., stealing credentials)?
  - **authorized** user laid off -> could become an **adversary**?
- **Approach**:
  - Define **threat model** that defines the power of the adversary