Privacy-Preserving Data Processing at Scale

How Much Can You Trust Your Cloud Provider?

CLOSER 2024

Prof. P. Felber
University of Neuchâtel
pascal.felber@unine.ch

Agenda

Privacy-preserving data processing at scale

- **Scalability** perspective
  - From HPC to the cloud (vertical vs. horizontal)
  - Outsourcing data processing (on-premises vs. off-premises)
- **Privacy** perspective
  - Threats and vulnerabilities
  - Protecting data and computations
  - Towards confidential computing
  - Practical security with TEEs

Data processing at scale: HPC

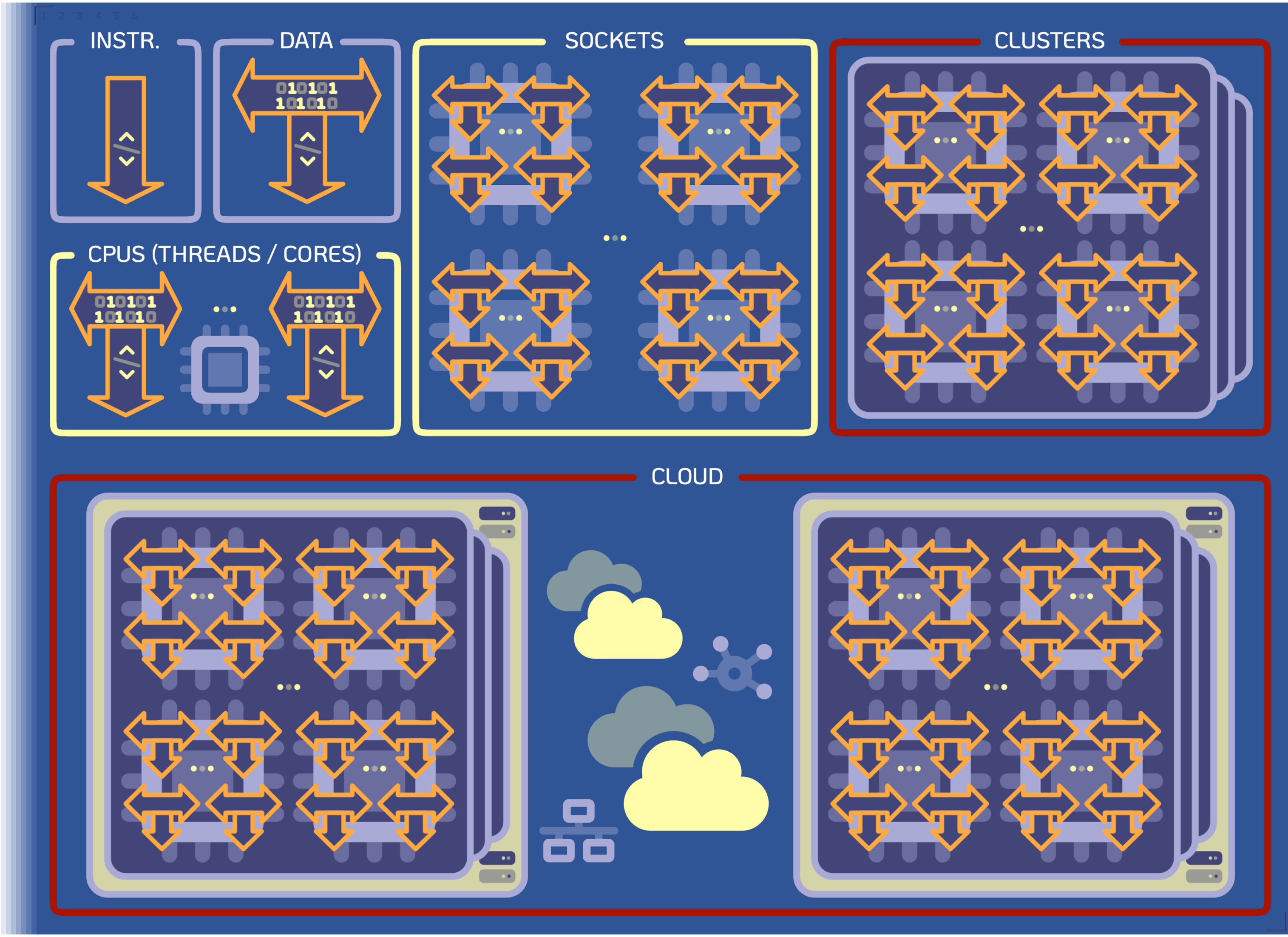
- HPC reaching beyond computational science
  - Digitalisation of society (producing data, using services)
  - Processing capabilities moving to the end users
- New **needs**: wealth of new problems and applications
  - End-user applications: games, multimedia (music, video), …
  - Big data: from information-generating technologies, e.g., mobile computing, sensor/social networks
  - Cryptocurrencies, machine learning, artificial intelligence!
- New **means**: multi-cores, GPUs, FPGAs/ASICs
  - Aggregation of computers (clusters) and data centres (cloud)

Scalability: a HW perspective

- Specialised ISA
  - SIMD (e.g., AVX)
- Parallelism
  - Threads, multi-cores
  - Multi-processors

  ⇒ **Vertical**
- Distribution
  - Clusters, data centres
  - Cloud infrastructures

  ⇒ **Horizontal**



From HPC to cloud computing

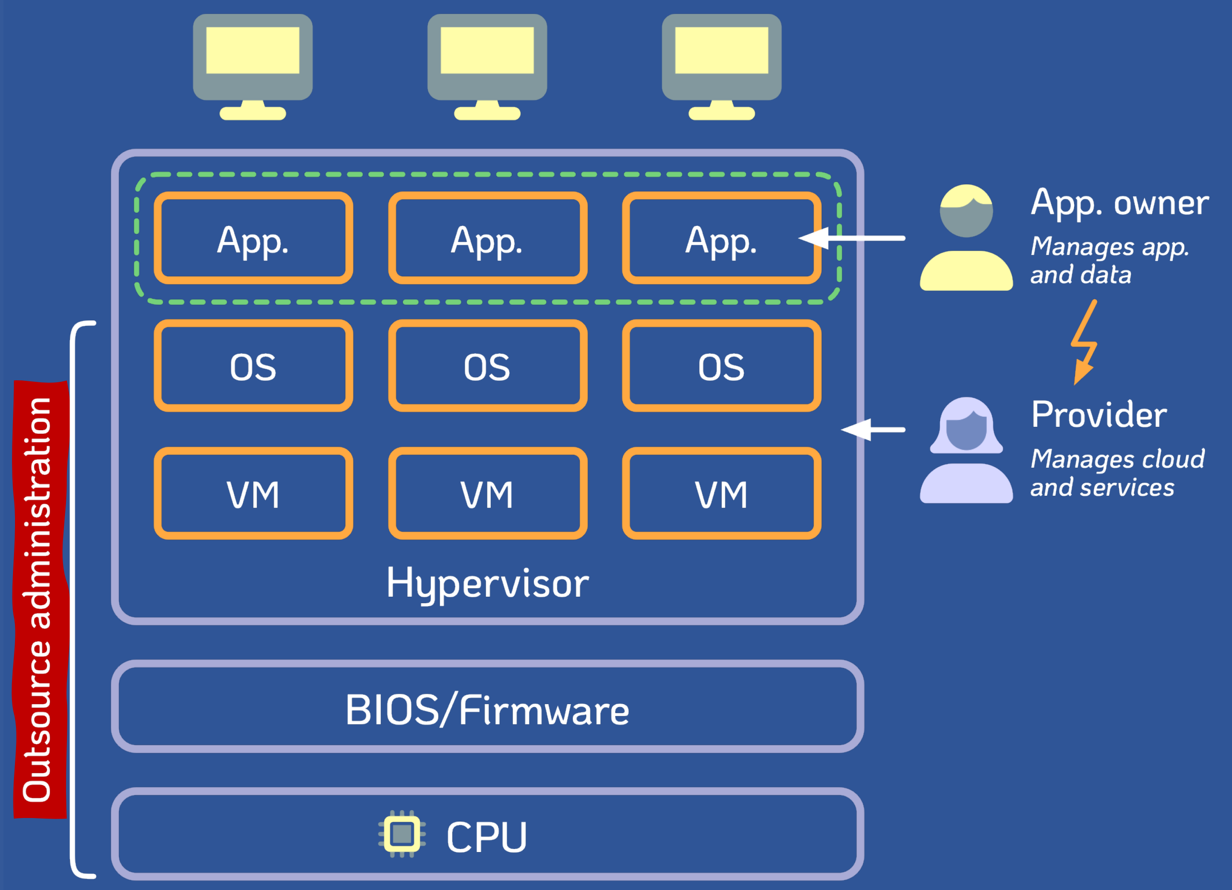
- Horizontal scalability is “unlimited”
  - Clusters and data centres provide massive computing power
  - Cloud computing federates data centres
- Cloud is an **appealing** paradigm
  - Cost savings due to sharing (economies of scale)
  - Affordable for SMEs
  - Widely applicable: IaaS, PaaS, SaaS, DaaS, FaaS, ?aaS
  - Easy/ubiquitous access to data

Cloud: Outsource infrastructure

- Operating own computing infrastructure is not easy
  - Data centre, hypervisors, operating systems, containers, services, *etc.*

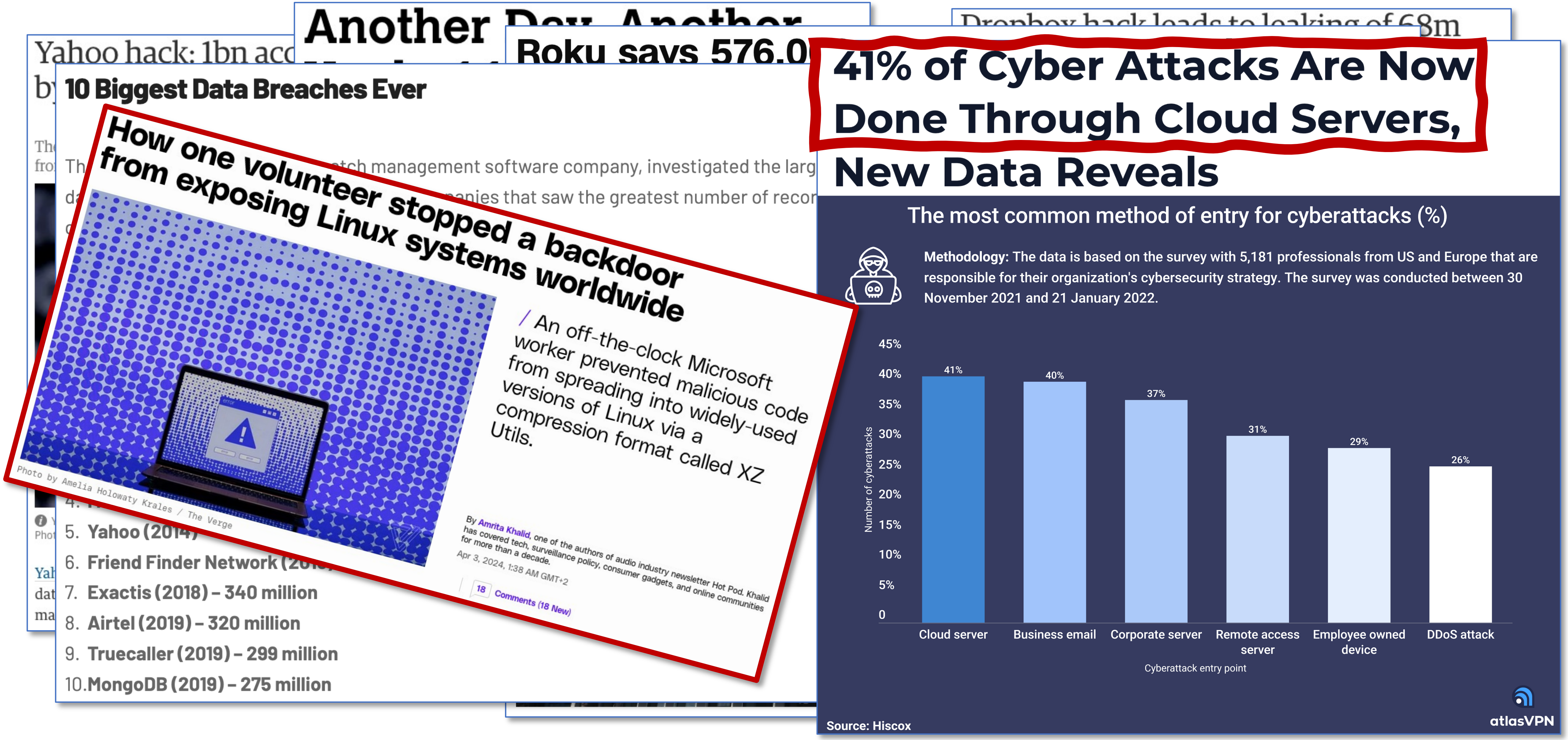
⇒ **Outsourcing**

  - Zero maintenance
- Yet, tempting to attack
  - Remotely accessible
  - Infrastructure, software, data must be secured!



Based on slides by C. Fetzer (TU Dresden)

Why is cloud security important?



Why is data so important?

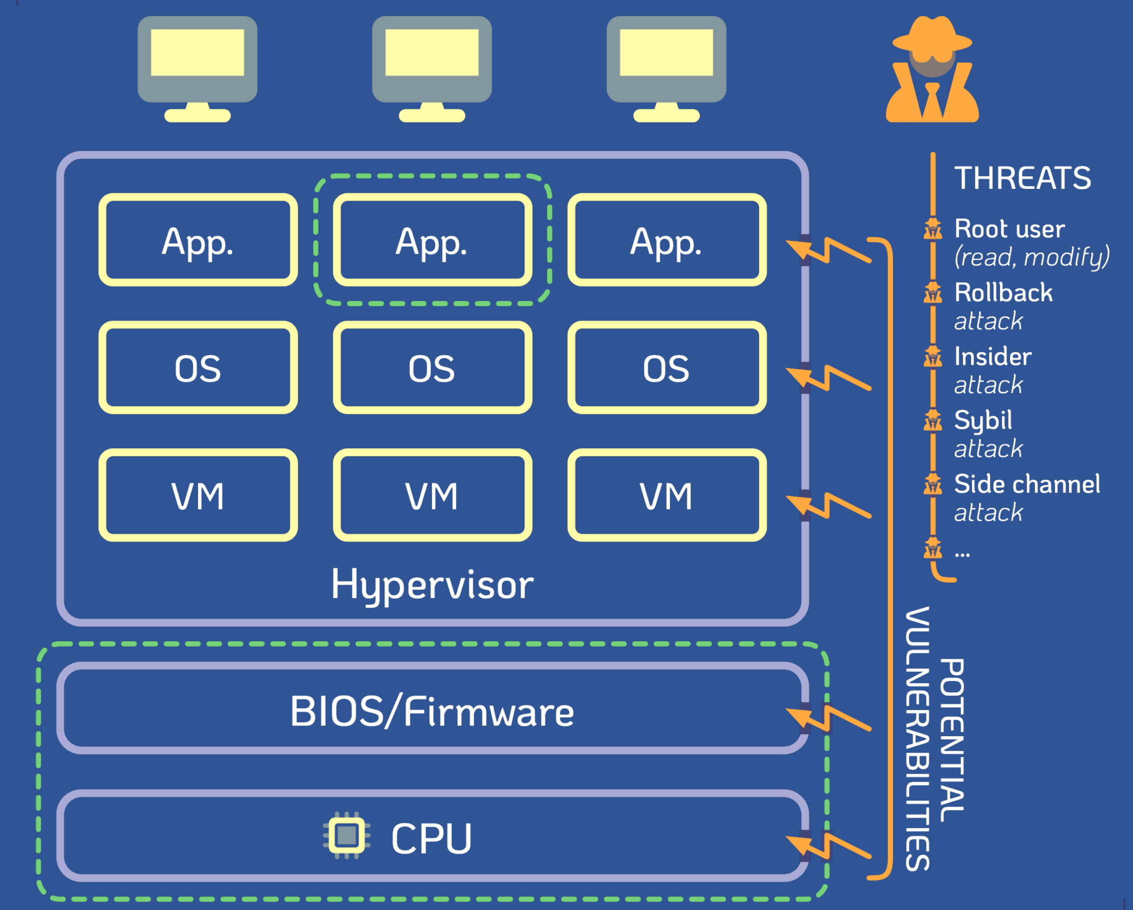
- Data is a **key asset** for businesses
  - Moving data offsite is an inherent security risk

⇒ **Data must be protected at all times**

- Data **at rest** (storage) or **in flight** (transmission)
  - Encryption helps
- Data **in use** (processing)
  - Secure processing of encrypted data is very hard
  - Cryptographic techniques are not practical (yet)

Clouds have a big "attack surface"

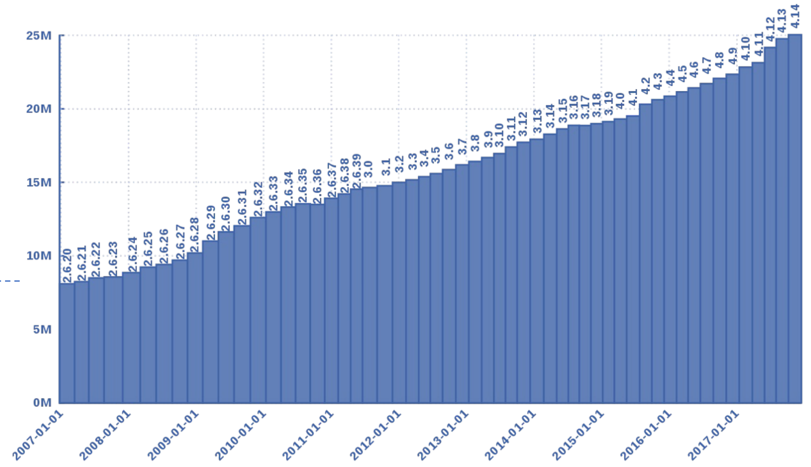
- Cloud infrastructures are inherently complex
  - Each layer has its own set of potential **vulnerabilities**
  - Multi-tenancy: applications must be isolated
- The *whole stack* must be protected from attacks
  - Wide range of **threats**: privileged access, insiders, attacks and exploits, …



Based on slides by C. Fetzer (TU Dresden)

The software stack is huge

- Cloud platforms contain enormous amounts of code that must be trusted
  - Linux kernel: 27+ MLOC
  - OpenStack: 20+ MLOC
  - KVM: 200+ kLOC



- Cloud platforms are effectively a trusted computing base (TCB): all components of the system are critical to security
  - Software, hardware

Bugs are a reality

- More code ⇒ more bugs
  - Vulnerabilities may lead to disclosure of confidential data
- Xen hypervisor: 450+ vulnerabilities (as of 2024)
  [https://www.cvedetails.com/product/23463/XEN-XEN.html?vendor_id=6276]
- Linux kernel: 4000+ vulnerabilities (as of 2024)
  [https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33]
- Especially bad in privileged software
  - May result in unrestricted access to the system
- Protected mode (rings) is not sufficient
  - Flaws and exploits can lead to privilege escalation
  - The attack surface is the whole software stack

Software attacks in the cloud

- Performed **remotely** (run software on victim's machine)
- Control-flow hijacking
  - Execute arbitrary code on the target machine by modifying the application's control flow
- Code injection attack
  - Overwrite the return address by writing beyond the allocated buffer on the stack (inject code) and jump to the injected code
- Return-oriented programming (ROP)
  - Hijack control flow by corrupting stack (no injection) and jump to sequences of instructions (gadgets) already present in memory (e.g., **`libc`**) ending with a return

Hardware attacks in the cloud

- Performed **locally** (physical access to victim's machine)
- Bus snooping
  - Dump CPU ⇔ memory communication
- Cold boot attacks
  - Power cycle the machine, boot to a lightweight OS, dump memory contents...

  ...or remove memory modules, plug into another machine, dump memory contents
  - DRAM retains its state for a short period of time

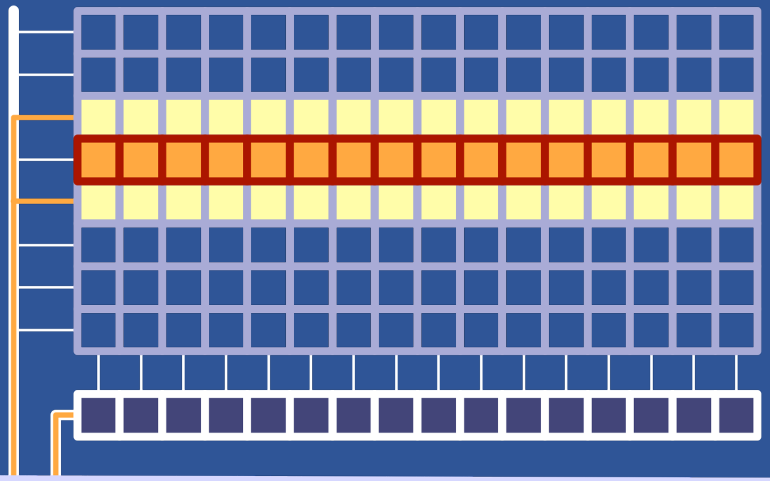
Some examples

- “Row hammer” attack
  - Attack the system by causing bit-flips in memory
  - Carefully chosen addresses can result in privilege escalation
- “Heartbleed” bug
  - Buffer overrun in OpenSSL cryptographic software library
  - The attacker can obtain sensitive data from server’s memory: passwords, private keys, …
- “Meltdown”, “spectre” and other side-channel attacks
  - Allow a program to access the memory and secrets of other programs and the operating system

⇒ Sound “theoretical” solutions fail in “real” systems!

Example: “*Row hammer*” attack

- Attack the system by causing bit-flips in memory
  - Accessing physical bits causes neighboring bits to flip
  - Carefully chosen addresses can result in privilege escalation
- Effect
  - Sandbox escape
  - Corrupted page table



Rapid row activations (yellow) may change the values of bits stored in victim row (orange)

[wikipedia]

```
code1a:
  mov (X), %eax // read from address X
  mov (Y), %ebx // read from address Y
  clflush (X)   // flush cache for address X
  clflush (Y)   // flush cache for address Y
  jmp code1a
```

Example: “*Heartbleed*” bug

- Serious vulnerability in the popular OpenSSL cryptographic software library
  - Very widely used: **`apache`**/**`nginx`** (60+% of Web servers), email servers, chat servers, VPN, etc.
- Buffer overrun when replying to a heartbeat message
- Allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software
  - The attacker can obtain sensitive data from server’s memory: passwords, private keys, …

Cryptography for cloud security?

- Cryptography can help protect data in the cloud…
  - **Encryption** for confidentiality: information is not available or disclosed to unauthorised individuals, entities or processes
  - **Digital signature**, MACs, secure hashes, … for integrity: data cannot be modified in an undetected manner

…but how can we protect confidentiality and integrity in untrusted environments **while** enabling data processing?

- Data should be searchable (e.g., range queries) and updatable (e.g., aggregation), yet not leak information (e.g., statistical attacks)

Encrypted data processing

- Homomorphic encryption

  "a form of encryption which allows specific types of computations to be carried out on ciphertext and generate an encrypted result which, when decrypted, matches the result of operations performed on the plaintext"

  [wikipedia]

- Fully homomorphic encryption [Gentry 2010]
  - Supports **arbitrary functions** on **encrypted data**
  - Addition, multiplication, binary operations

Can homomorphic encryption be practical?

Suitable parameters are given in Table 1 as $t = 1024$, $D = 2$, and $n = 2048$ with the 58-bit prime $q = 144115188076060673$.

Homomorphic encryption

[https://eprint.iacr.org/2011/405.pdf]

t	D	n	$\lceil \lg(q) \rceil$	S_χ ms	SH.Keygen ms	SH.Enc ms	SH.Enc precomp. ms	SH.Dec deg 1 ms	SH.Dec deg 2 ms	SH.Add ms	SH.Mult ms	SH.Mult w/ deg red s
2	1	512	19	27								
	2	1024	38	55								
	10	16384	338	870								
	15	16384	513	864								
1024	1	1024	30	54	110	164	5	4	–	< 1	–	–
	2	2048	58	110	250	348	24	15	26	1	41	0.19
	3	2048	91	111	270	366	38	22	41	2	73	0.46
	3	4096	95	221	530	733	81	46	88	4	154	0.95
	4	4096	130	220	580	756	102	57	109	4	196	1.50
	5	4096	165	220	600	770	117	64	125	4	226	2.19
	5	8192	171	440	1250	1582	275	148	288	5	526	5.33
	10	8192	354	435	1720	1824	523	271	538	9	538	19.28
	10	16384	368	868	3690	3851	1260	664	1300	19	1593	48.23
	15	16384	558	863	5010	4805	2343	1136	2269	13	4411	126.25

Database of 1 million items
- Aggregation (1 addition per item): 15+ minutes
- Range query (1 multiplication per item): 10+ hours

Table 2: Timings for the somewhat homomorphic encryption scheme using the example parameters given in Table 1. The column labeled S_χ gives timing for sampling an element from the discrete Gaussian distribution χ. In the second column for SH.Enc, labeled prec., encryption is measured without sampling from χ, which is instead done as a precomputation. The two columns for SH.Dec correspond to decryption of a degree-1 and a degree-2 ciphertext, respectively. The last column gives the time taken for a ciphertext multiplication of two linear ciphertexts including the degree reduction resulting in a degree-1 ciphertext for the product. Measurements were done on a 2.1 GHz Intel Core 2 Duo using the computer algebra system Magma [BCP97].

Homomorphic encryption

- HELib: open-source homomorphic encryption library in C++ by IBM [https://github.com/homenc/HElib]
  - Many optimisations to make HE "*practical*", i.e., run faster
  - Low-level routines (set, add, multiply, shift, etc.)
- Still far from being practical
  - Orders of magnitude slower than operations on plaintext
  - Addition: ~1+ ms — Multiplication: ~10/100+ ms
  - HELib evaluated the AES-128 circuit in 36 hours in 2012 (vs. 2 ms in the clear) [https://mpclounge.files.wordpress.com/2013/04/hespeed.pdf]
- Several other libraries, e.g., Microsoft SEAL, OpenFHE [http://github.com/Microsoft/SEAL], [http://github.com/openfheorg]

Homomorphic encryption

[https://arxiv.org/pdf/2202.02960.pdf]

2.2B× slower in multiplication!

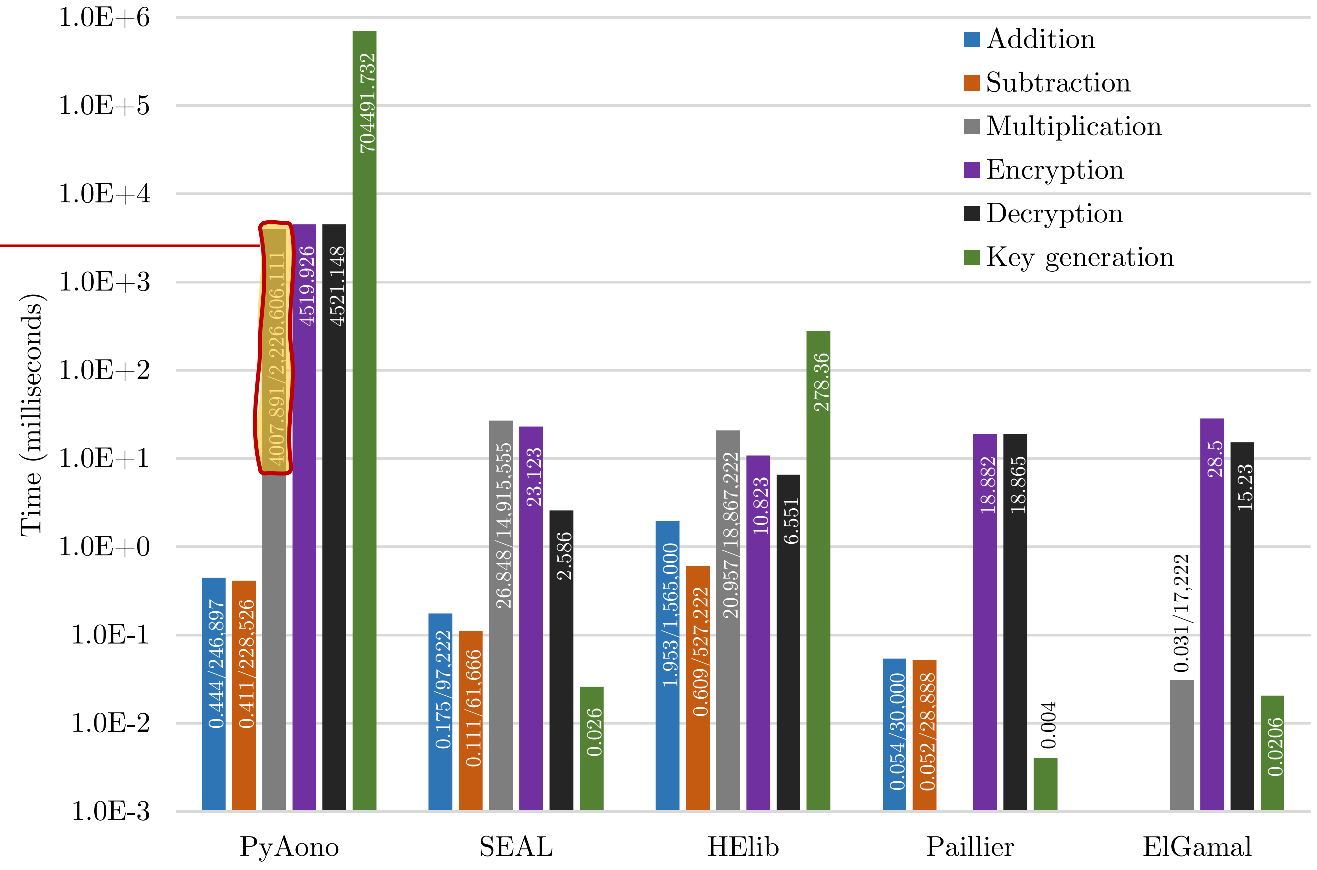
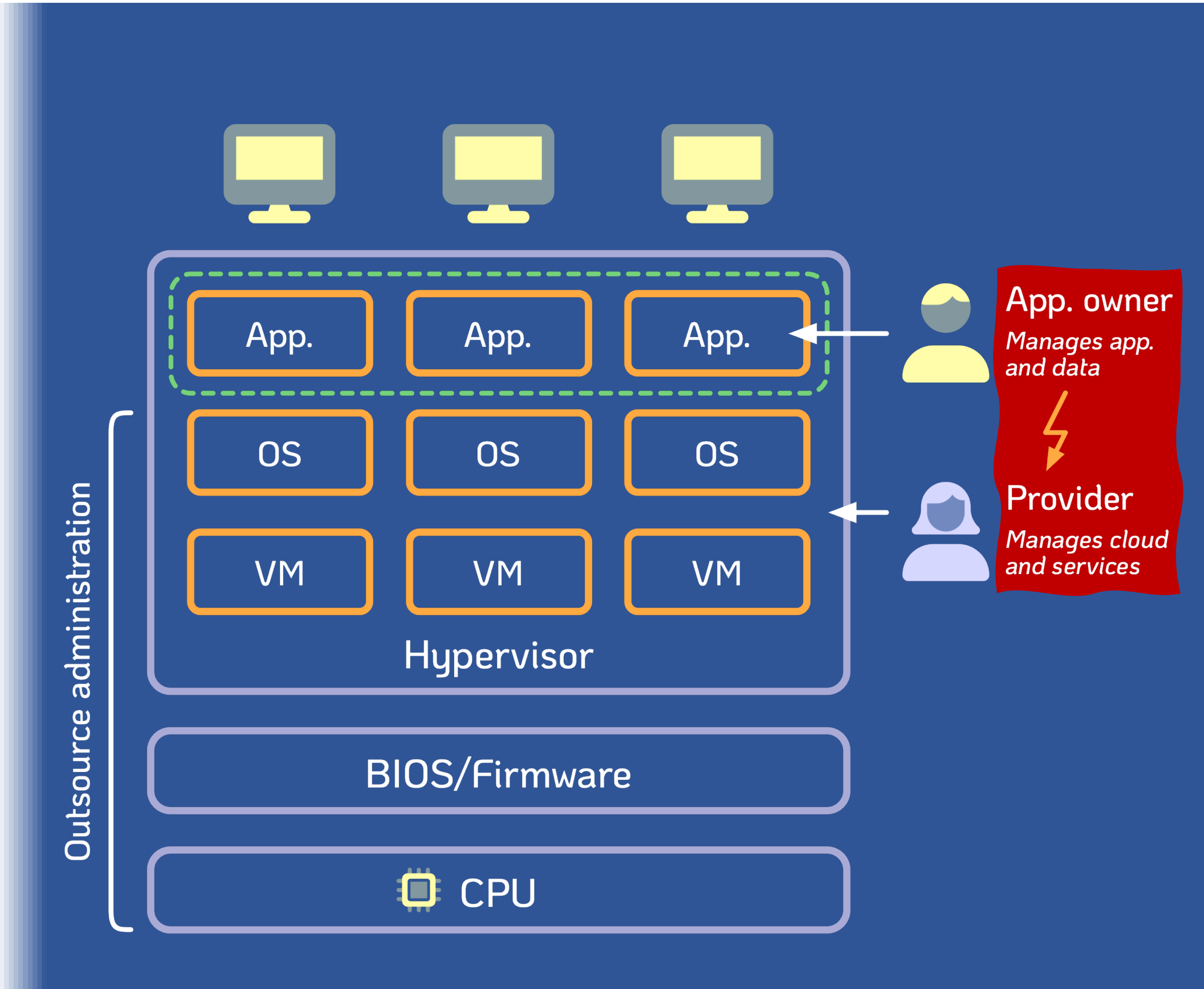


Fig. 1: Overall times for all operations compared between each other with default parameters (1000 iterations).
For operations of $+ - \times$, values are in form t/r, where t is time in ms, and r is the ratio of t and the time execution of the same operation took over plaintexts.
E.g., PyAono's addition is 246,897 times slower than plaintext addition.

Cloud and stakeholders: A matter of trust

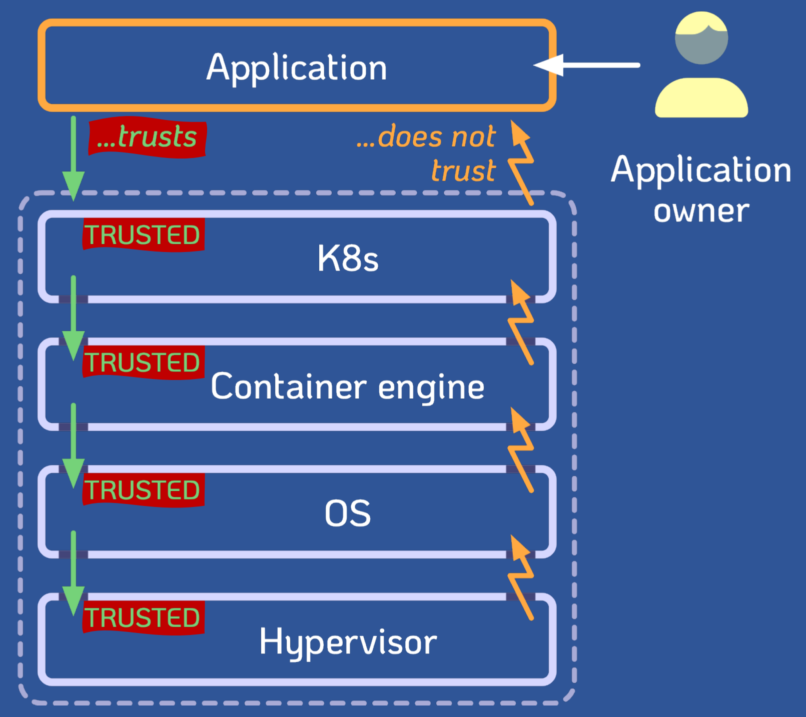
- Multiple parties share the same infrastructure
- Each stakeholder protects its own resources
  - The **application owner** protects the application
  - The **cloud provider** protects the system
- They do not necessarily **trust** each other



Based on slides by C. Fetzer (TU Dresden)

Systems security: Bottom-up

- Systems are structured in layers
  - E.g., OS and hypervisor
- Typically, systems security is bottom-up
  - Layer *i*+1 (↑) **trusts** layer *i* (↓)
    *…**but** layer i does **not** trust layer i+1*
  - E.g., the OS **trusts** the hypervisor
    *…**but** the hypervisor does **not** trust the OS*
    *…which does **not** trust the container engine, **nor** the layers above (K8s…)*



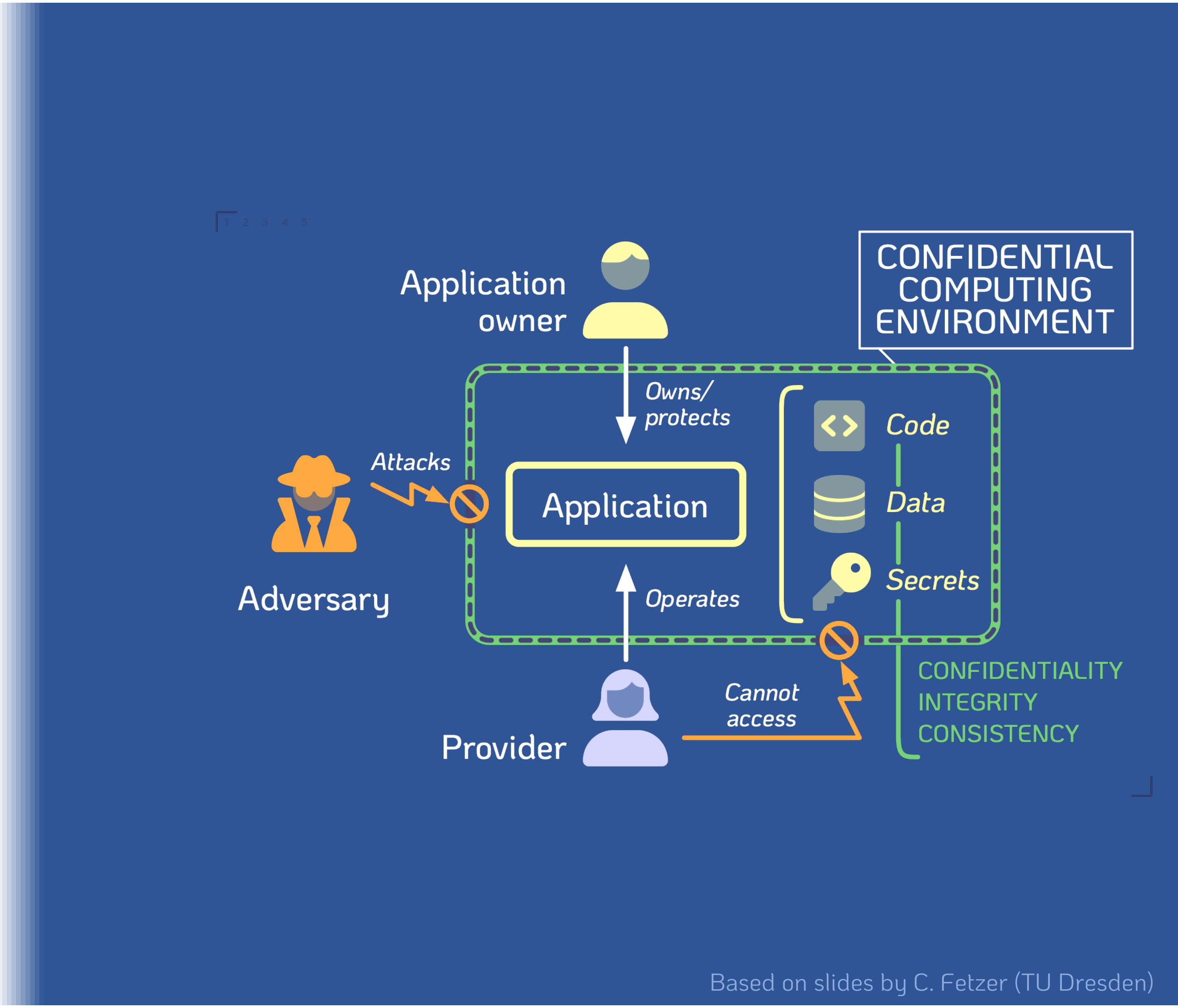
Based on slides by C. Fetzer (TU Dresden)

Confidential computing: Top-down

"Application-oriented security"

- The application owner protects his assets from adversaries
  - Code, data, secrets
- The cloud provider is not trusted

⇒ Confidential computing environment

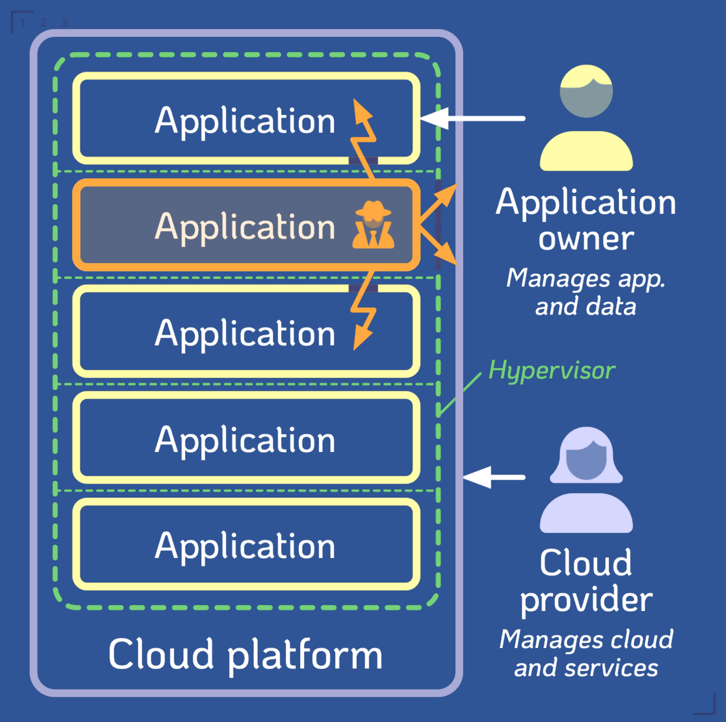


Trust is a two-sided problem

(1/2)

Provider's perspective

- Cloud provider needs to protect against malicious customers
  - Hypervisor-based isolation
  - Both security and performance
- One-way protection



Based on slides by C. Fetzer (TU Dresden)

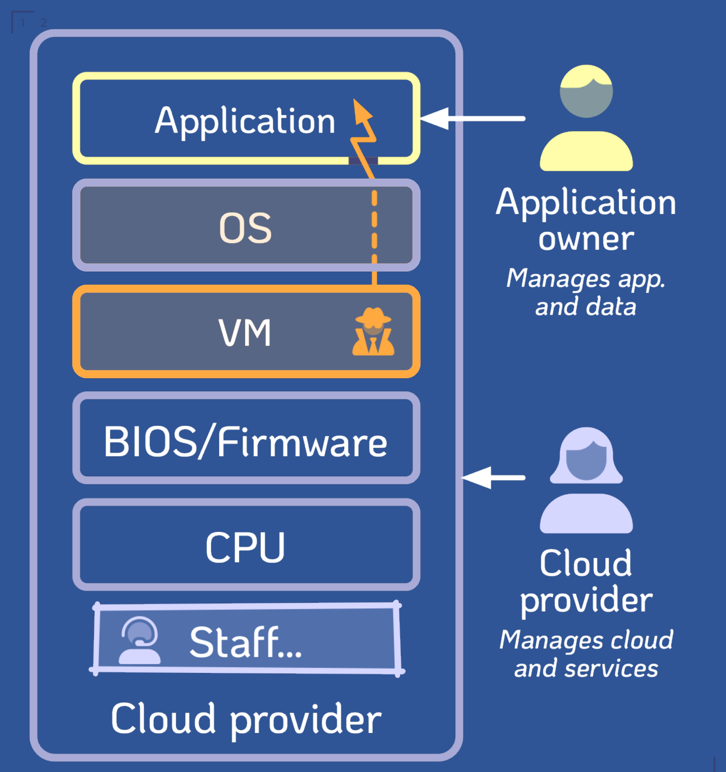
Trust is a two-sided problem (2/2)

Client's perspective

- Tenant is forced to trust the provider…

  …including personnel

  …including every software component
- Ideally, we want to trust only our service



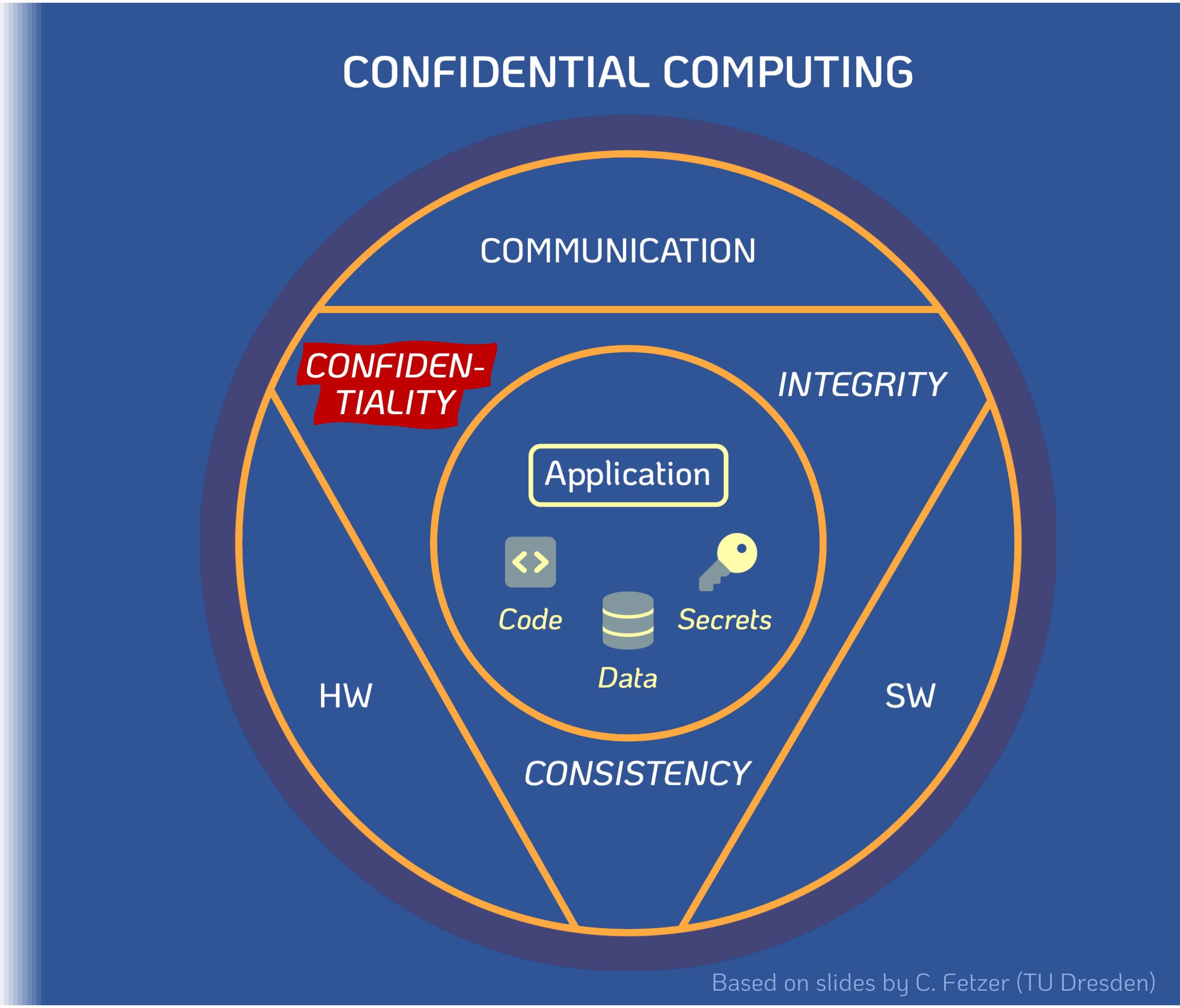
Based on slides by C. Fetzer (TU Dresden)

What is confidential computing?

Confidentiality

Guarantees that…
information (data, code, secrets) is not made available or disclosed to unauthorised individuals, entities, or processes

Only authorised users/programs can read

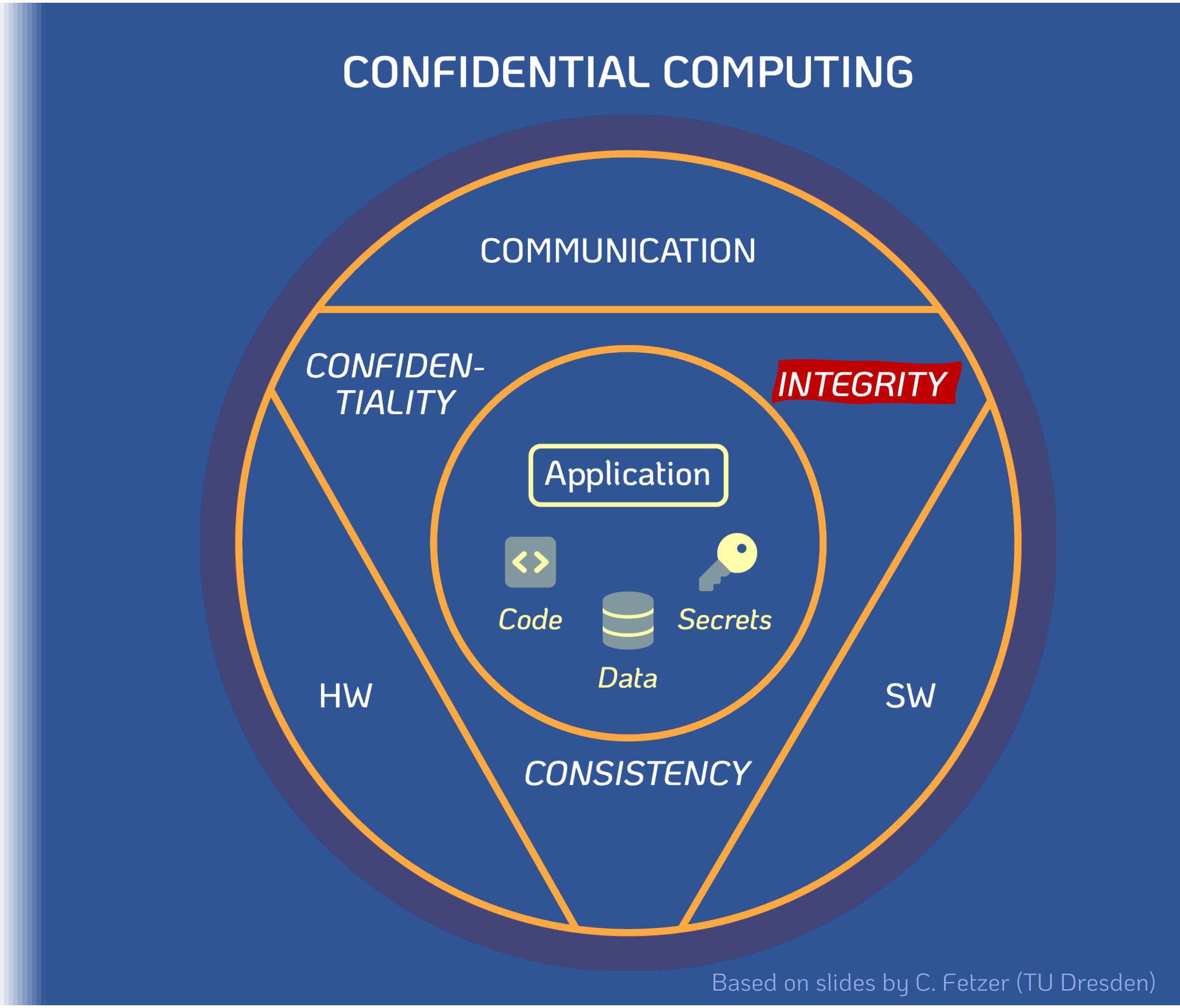


What is confidential computing? (2/3)

Integrity

Guarantees that… *information (data, code, secrets) cannot be modified in an unauthorised or undetected manner*

Only authorised users/programs can update



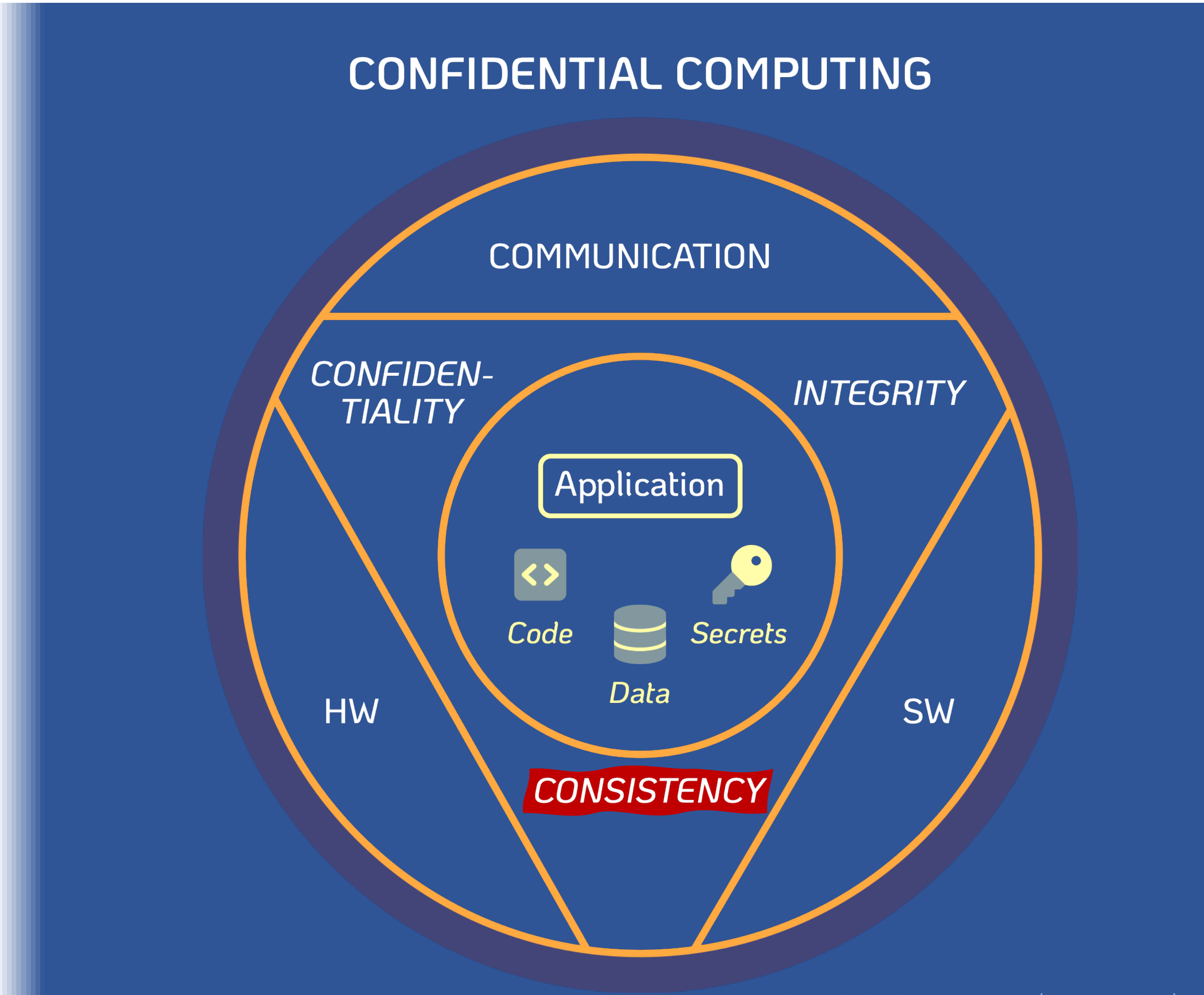
What is confidential computing? (3/3)

Consistency

Guarantees that…
one always reads the latest information (data, code, secrets) written by an authorised entity

⇒ *Detect if an adversary provides old copies (correctly encrypted but since updated)*

Always accessing the last version



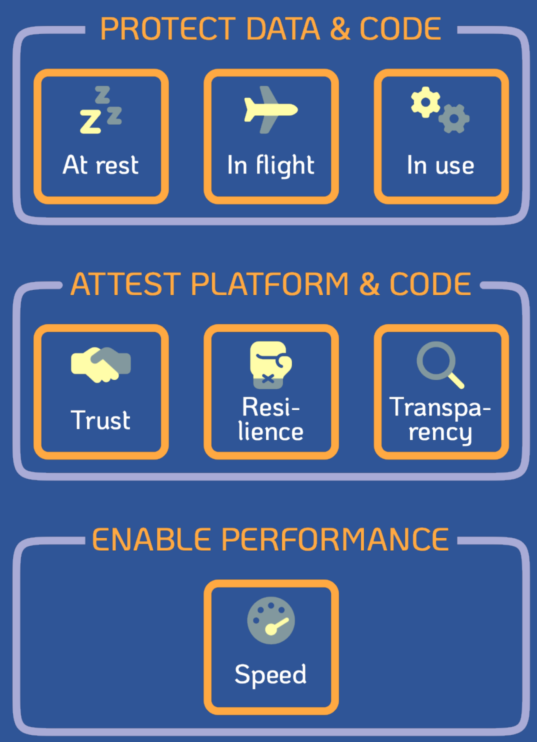
Based on slides by C. Fetzer (TU Dresden)

Confidential computing: Goals

1. Protect the data and the code from *unauthorised users*
   - At rest, in flight, in use
2. Attest the platform and the code
   - Only run unmodified applications on verified platform (attestation service)
3. Does not hamper performance

Who is authorised?
Who are the adversaries?

"Know thy enemy and know yourself…"



Based on slides by C. Fetzer (TU Dresden)

Who is authorised?

- Cloud infrastructures deal with many stakeholders (roles) and support multi-tenancy
  - **Infrastructure providers** operate computers and manage resources
  - **Service providers** operate the services
  - **Application providers** prepare "containerised" applications
  - **Data owners** provide and monetise the data
  - **Data scientists** use the applications and services
  - **Auditors check** the source code for vulnerabilities

    …
- Requires role-based access management (policies)

Who are adversaries?

[https://www.intel.com/.../threat-modeling.html]

Adversary	Description
Unprivileged Software Adv.	Typically known as a "user-space" adversary; capabilities are limited by the instruction set architecture (ISA) or hardware platform or x86/x64 (or IA-32/Intel 64) to the capabilities granted by the system software.
System Software Adv.	Full control over the operating system, or virtual machine monitor. This adversary can manipulate x86/x64 in any manner allowed by the instruction set architecture specification.
Startup code and SMM Adv.	All capabilities of the System Software Adversary, as well as control over initial boot code and system management mode. This adversary can manipulate x86/x64 in any manner allowed by the instruction set architecture specification. This adversary also has the ability to compromise system and platform firmware.
Network Adv.	Access to and may have control over various network fabrics that are used to connect the platform to other platforms, intranet, or extranet resources. This adversary can also interact with remote systems through predefined APIs.
Software Side Channel Adv.	Able to gather statistics from the CPU regarding execution and may be able to use them to extract secrets from software being executed. This adversary can also observe hardware resource usage to infer information and secrets from software being executed. This adversary can often directly influence resource usage (e.g., by causing contention) or by modulating an input to a victim program.
Simple Hardware Adv.	Physical access to the system and typically doesn't require expensive equipment or extraordinary training/specialty.
Skilled Hardware Adv.	Physical access to the system and additional equipment and/or training that isn't accessible to the average individual consumer.
HW Reverse Engineer Adv.	Physical access to the system, specialized tooling (which can be rented), and highly specialized expertise.
Authorized Adv.	Intel or partner-granted authority that has capabilities not available to unauthorized entities. This may include access to manufacturing facilities and systems, access design facilities and design systems or with access to devices that haven't completed all manufacturing steps.

The quest for (practical) security

- Production systems must be protected
  - Mission-critical, vulnerable to hackers
  - Manage sensitive data
- Distributed systems are exposed
  - Remote data and code and data must be protected
- Execution environments must be secure (both ways!)
  - Protect the environment from the application
  - Protect the application from the environment
- Performance must be preserved

⇒ **Leveraging trusted execution environments (TEEs)**

Trusted execution environments (TEEs)

- **TEEs** isolate applications from the rest of the system
  - Segregated area of memory and CPU protected by HW against powerful attacks
  - Its content is shielded from other applications, compromised OS and system libraries, attackers with physical access to the machine, …
- Uses "attestation" to verify SW and HW before execution
- Guarantees **data confidentiality** and **code integrity**
  - Prevents unauthorised parties outside TEE from reading data
  - Prevents unauthorised parties from replacing or modifying code in TEE

Data confidentiality and code integrity

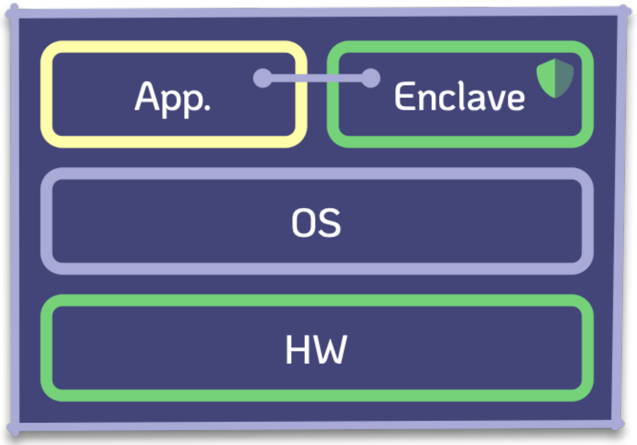
- **Data** in TEE never leaves the CPU package unencrypted
  - Outside the CPU, data is encrypted
  - In the TEE, data can be processed in plaintext
- **Code** is verified before execution by the CPU
  - Validates integrity of cache lines and virtual-to-physical addresses (e.g., by maintaining the root of a Merkle tree)
- Cryptographic operations performed by a dedicated memory encryption engine (MEE)
  - Transparently encrypts and decrypts memory (cache lines)
  - Provides support for efficient paging

Trusted execution environments (TEEs)

- Various TEE architectures exist and depend on the CPU
- They differ by their threat model and capabilities
  - **Intel SGX:** enclaves
  - **Arm TrustZone:** separate systems (two "worlds")
  - **AMD SEV:** virtualised systems (VMs)
  - **Intel TDX:** trusted domains (VMs)
  - **Arm CCA:** realms (system-wide hardware isolation)
  - **RISC-V:** several proposed extensions

  …

Intel SGX



Software guard extensions

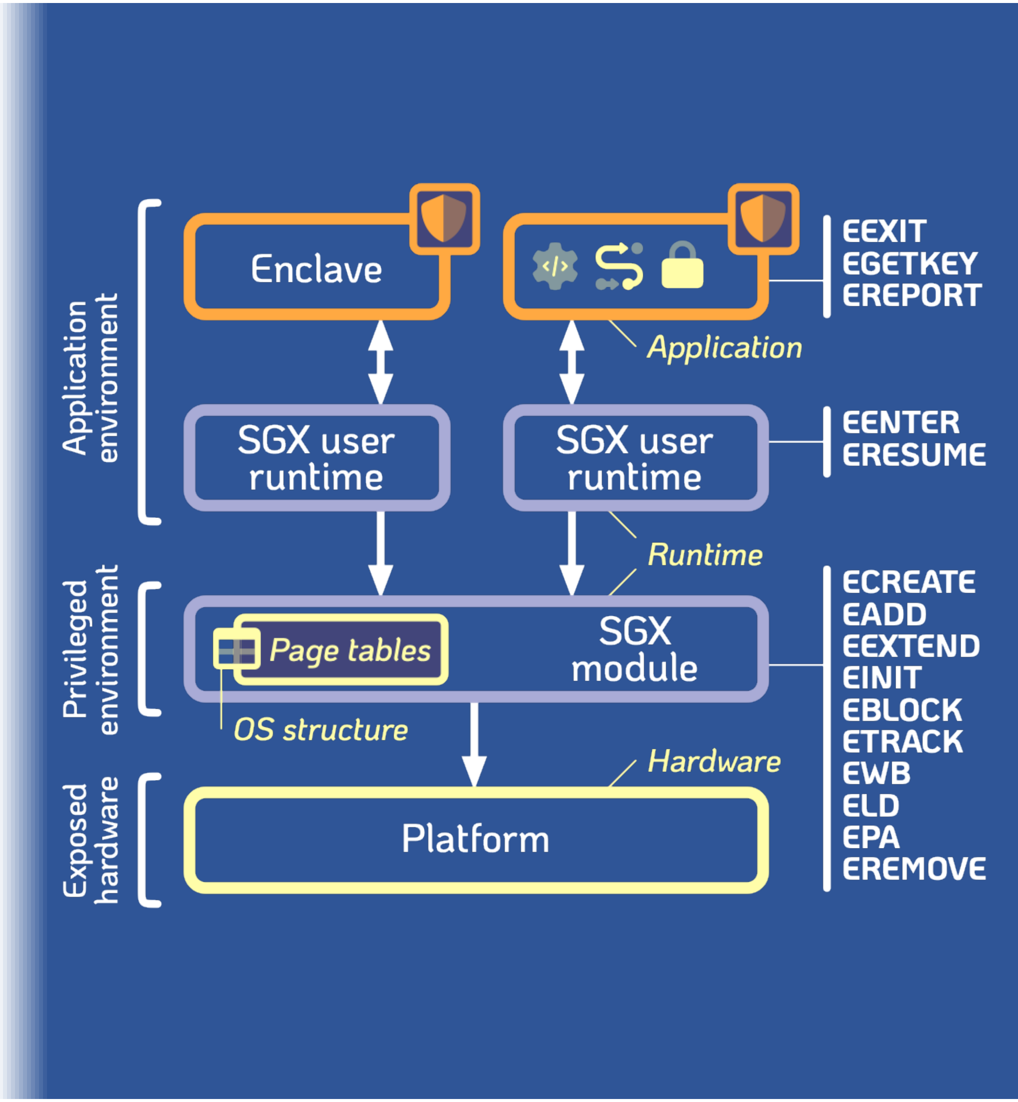
- Hardware extension in recent Intel CPUs since Skylake (2015)
- Protects confidentiality and integrity of code and data in untrusted environments
  - The platform is considered malicious by default
  - Only the CPU chip and the isolated region are trusted
  - Code is attested (via Intel attestation service)
- Code runs in an “enclave”: a piece of trusted software

SGX architecture and API

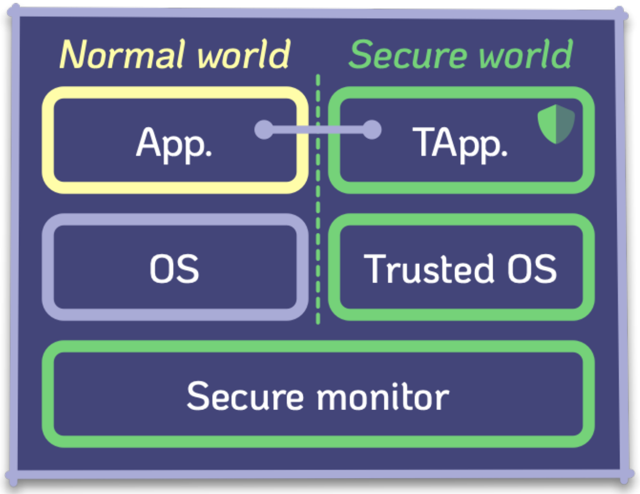
- Secure code runs "native speed"…

  …but API is quite complex
  - Need to heavily modify legacy code

  …small enclave page cache (EPC)
  - SGXv1: 128 MB (~96 MB w/out paging)
  - SGXv2: up to 1 TB
- Performance of memory accesses
  - Native speed in L1/L2/L3 cache
  - Reasonable within the EPC
  - Huge when paging to main memory



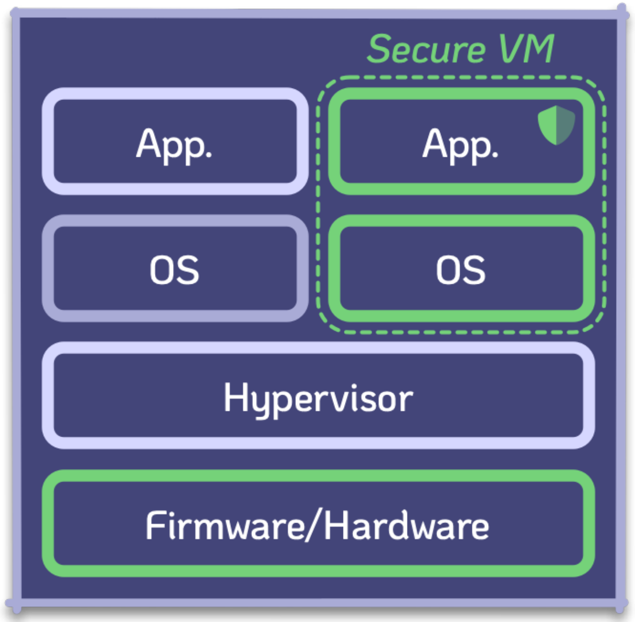
Arm TrustZone (TZ)



- TZ is widely spread on small and IoT devices with a Cortex-A/M processor
- Separates devices in two worlds
  - The **normal** world
  - The **secure** world
- One trusted application (TA) at a time
- Provides memory confidentiality but not integrity
- No built-in attestation service
- Limited memory per TA (~4–32 MB in practice)

AMD SEV

- SEV-SNP is supported on computers and servers with EPYC 7003+ series processors
- Each trusted environment is a secure virtual machine
- SEV-SNP provides both memory confidentiality and integrity
- Support for remote attestation
- Unlimited amount of addressable memory



Comparison of TEEs

[https://arxiv.org/pdf/2206.03780]

Features	SGX		TrustZone		SEV			RISC-V			
	Client SGX	Scalable SGX	TrustZone-A	TrustZone-M	Vanilla	SEV-ES	SEV-SNP	Keystone	Sanctum	TIMBER-V	LIRA-V
Integrity	●	◐	○	○	○	○	◐	●	○	○	○
Freshness	●	◐	○	○	○	○	◐	●	○	○	○
Encryption	●	●	○	○	●	●	●	●	○	○	○
Unlimited domains	●	●	○	●	◐	●	●	●	●	●	○
Open source	◐	◐	◐	◐	○	○	○	●	●	●	○
Local attestation	●	●	○	○	○	○	○	○	●	●	○
Remote attestation	●	●	◐	◐	●	●	●	●	●	●	●
API for attestation	●	●	◐	◐	○	○	●	●	●	●	●
Mutual attestation	○	○	◐	○	○	○	○	○	○	○	●
User-mode support	●	●	●	●	●	●	●	●	●	●	○
Industrial TEE	●	●	●	●	●	●	●	○	○	○	○
Isolation and attestation granularity	Intra-address space		Secure world		VM			Secure world	Intra-address space		
System support for isolation	μcode + XuCode		SMC	MPU	Firmware			SMC + PMP		Tag + MPU	PMP

Table 1: Comparison of the state-of-the-art TEEs.

Feature	Description
Integrity	An active mechanism preventing DRAM of TEE instances from being tampered with. Partial fulfilment means no protection against physical attacks.
Freshness	Protecting DRAM of TEE instances against replay and rollback attacks. Partial fulfilment means no protection against physical attacks.
Encryption	DRAM of TEE instances is encrypted to assure that no unauthorised access or memory snooping of the enclave occurs.
Unlimited domains	Many TEE instances can run concurrently, while the TEE boundaries (e.g., isolation, integrity) between these instances are guaranteed by hardware. Partial fulfilment means that the number of domains is capped.
Open source	Indicate whether the solution is either partially or fully publicly available.
Local attestation	A TEE instance attests running on the same system to another instance.
Remote attestation	A TEE instance attests genuineness to remote parties. Partial fulfilment means no built-in support but is extended by the literature.
API for attestation	An API is available by the trusted applications to interact with the process of remote attestation. Partial fulfilment means no built-in support but is extended by the literature.
Mutual attestation	The identity of the attester and the verifier are authenticated upon remote attestations. Partial fulfilment means no built-in support but is extended by the literature.
User mode support	State whether the trusted applications are hosted in user mode, according to the processor architecture.
Industrial TEE	Contrast the TEEs used in production and made by the industry from the research prototypes designed by the academia.
Isolation and attestation granularity	The level of granularity where the TEE operates for providing isolation and attestation of the trusted software.
System support for isolation	The hardware mechanisms used to isolate trusted applications.

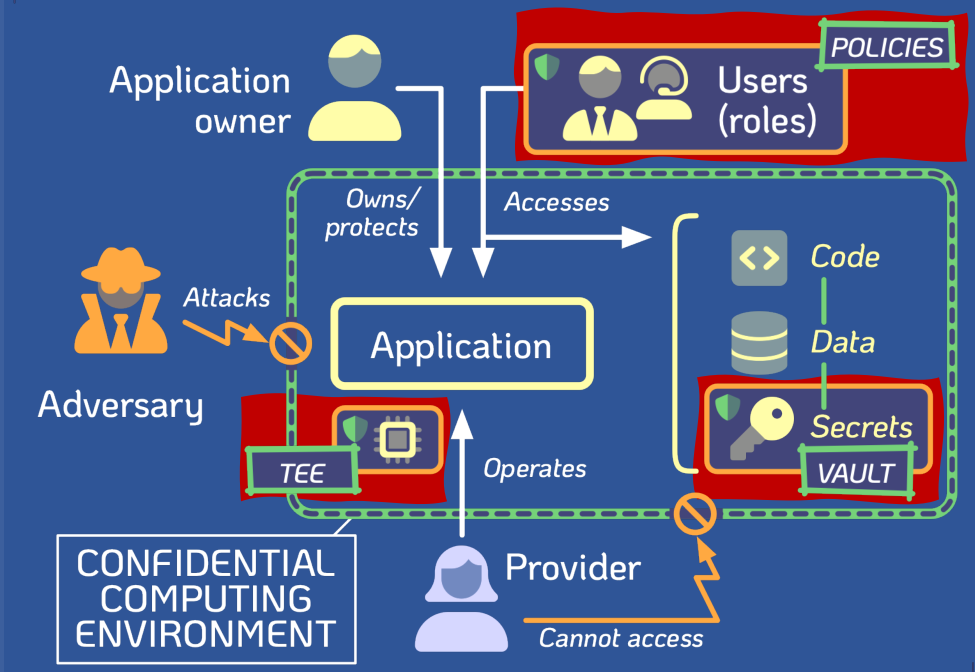
Table 2: Features of the state-of-the-art TEEs.

A complete CC architecture

- Many (distrustful) stakeholders require proper governance
  - Access control via policy engine
- Secrets (DB password, encryption key, etc.) must be protected
  - Confidential managed vault
- Performance should be preserved
  - All secure operations within TEEs

Complete confidential computing architectures are available

E.g., SCONE [https://sconedocs.github.io]



Based on slides by C. Fetzer (TU Dresden)

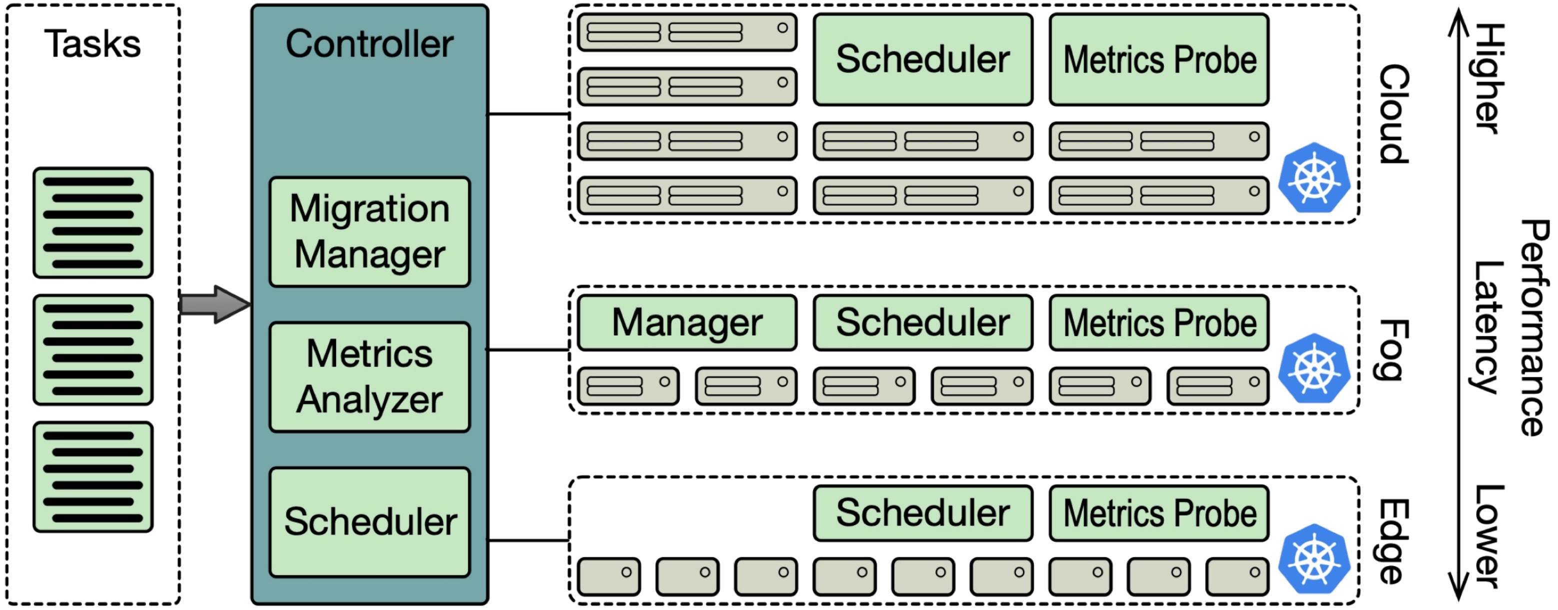
TEEs are no silver bullet

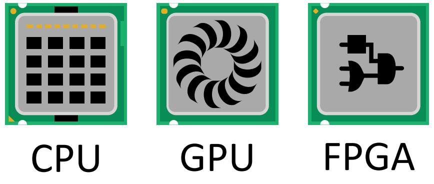
- Require some craft from programmers
  - SDK is only available for limited programming languages
  - Constrained development environments
- Might lack fundamental properties
  - E.g., attestation or integrity are not always supported
- Performance can be poor (e.g., memory limitations)
- Requires good knowledge of system issues
  - No POSIX API (hard to write or migrate existing applications)
- Continuous stream of (side-channel) attacks

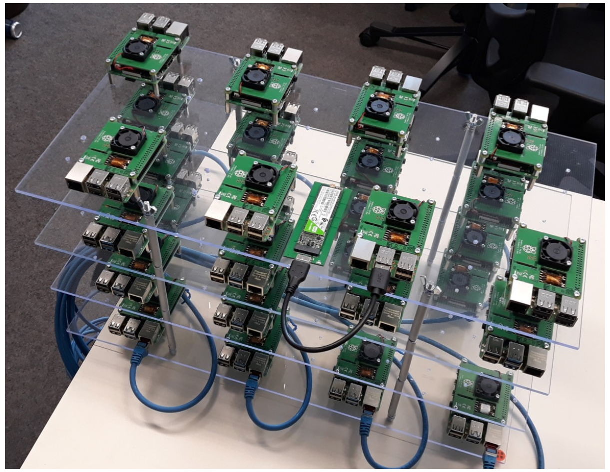
Security for energy-efficient HPC

[LEGaTO]

- Low-energy toolset for heterogeneous computing
  - Task scheduling across CPUs, GPUs, FPGAs, ASICs, Pi…
  - Objectives: scalability, energy-efficiency, dependability, security (with SGX)…



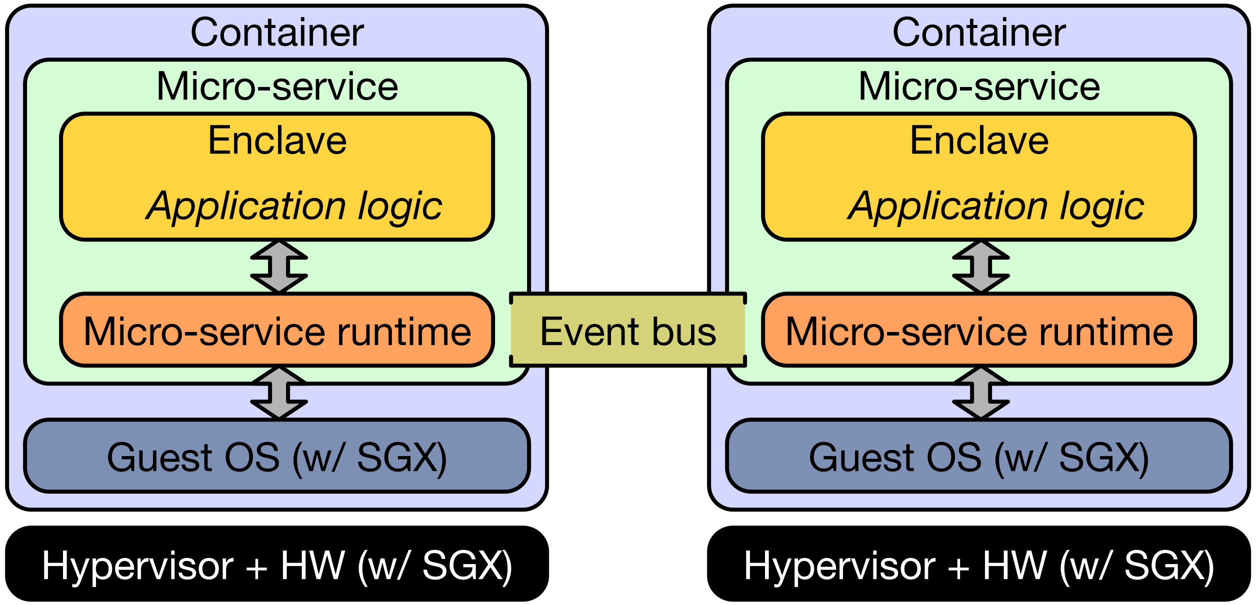




Security for public clouds

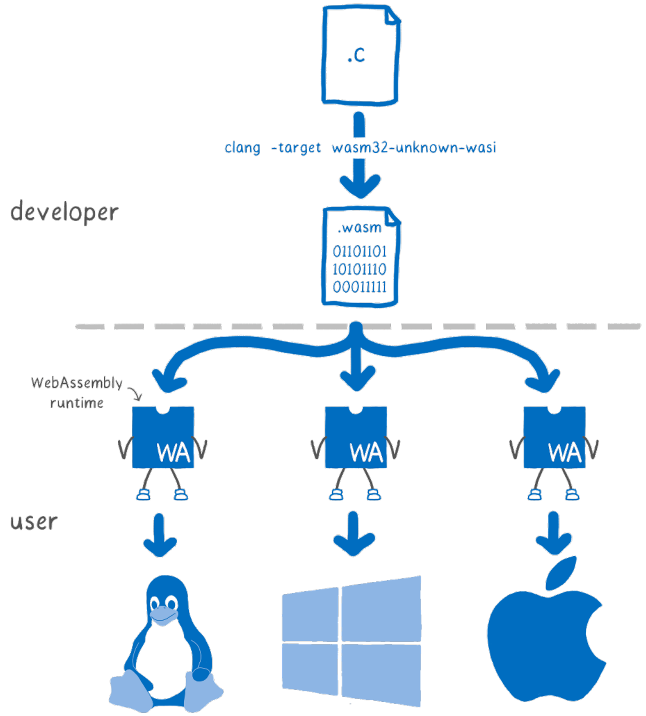
[SecureCloud], [SafeCloud]

- µ-services within containers in TEEs in (public) clouds
  - Full stack, multiple languages (C/C++, Go, Rust, Java, Python, Lua…), secure channels, SGX-aware scheduling, monitoring, core µ-services (communication, storage, map-reduce)…



Security for cloud-edge continuum [VEDLIoT]

- Wasm: standard for a bytecode format
  - Compilation target for mainstream programming languages
  - Universal runtime (not only for the web)
  - WebAssembly system interface (WASI) for system interactions
- Pros
  - **Lightweight** bytecode and specifications
  - Code execution is **sandboxed** (also protects the host)
  - Near-native **speed** with AOT and JIT compilation
  - **Same code** on cloud, edge, IoT devices: *cloud-edge continuum*



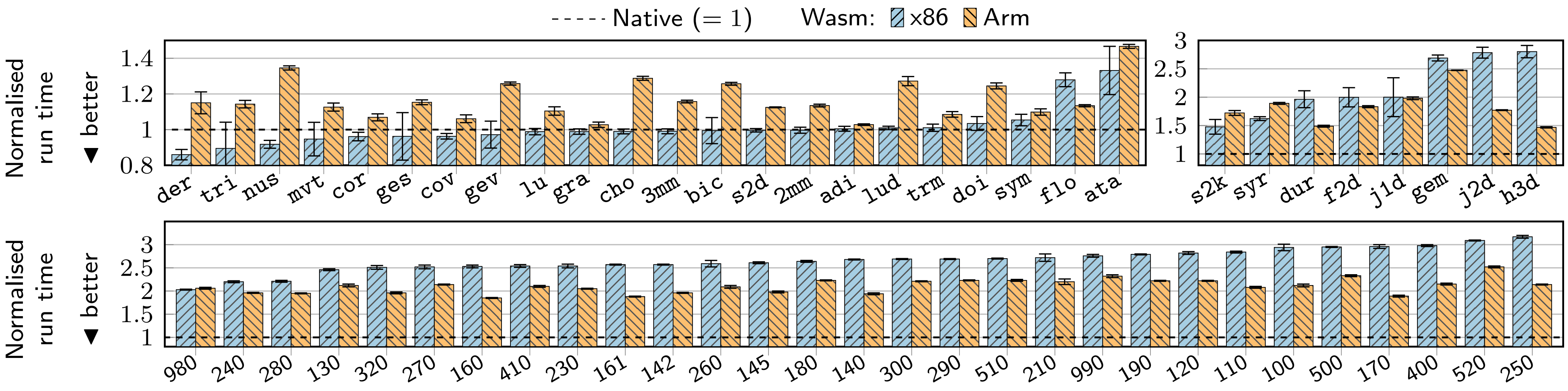
WebAssembly + TEEs

[VEDLIoT]

Twine for Intel SGX [ICDE'21] + WaTZ for Arm TrustZone [ICDCS'22]

- Execute Wasm code securely within TEE
- Leverage WASI to replace POSIX and deliver TEE features
- Benchmarks (Polybench/C and SQLite) show <3 slowdown

⇒ Confidential computing for the could-edge continuum



Wrapping up

- Scalability and security are often conflicting goals
  - Scalability can best be achieved by outsourcing
  - Security by keeping data and computations on-premises
- Recent advances in HW security extensions pave the way to privacy-preserving data processing in the cloud
  - Enabled by **confidential computing environments**
- Threats should not be underestimated
  - Multi-tenancy exposes data and computations to exploits
  - Vendors protect from different threat models
  - HW security is no silver bullet: need multiple protection layers