University of BRISTOL

# Hardware Security

Dr. Sana Belguith

University of BRISTOL

# Rootkit

bristol.ac.uk

# Purpose

- Give attacker a permanent root access to a system
- Hide its presence
  - Hide from filesystem
  - Hide its activity
  - etc…
- Steal information
- Allow remote code execution

# Typical attacker steps

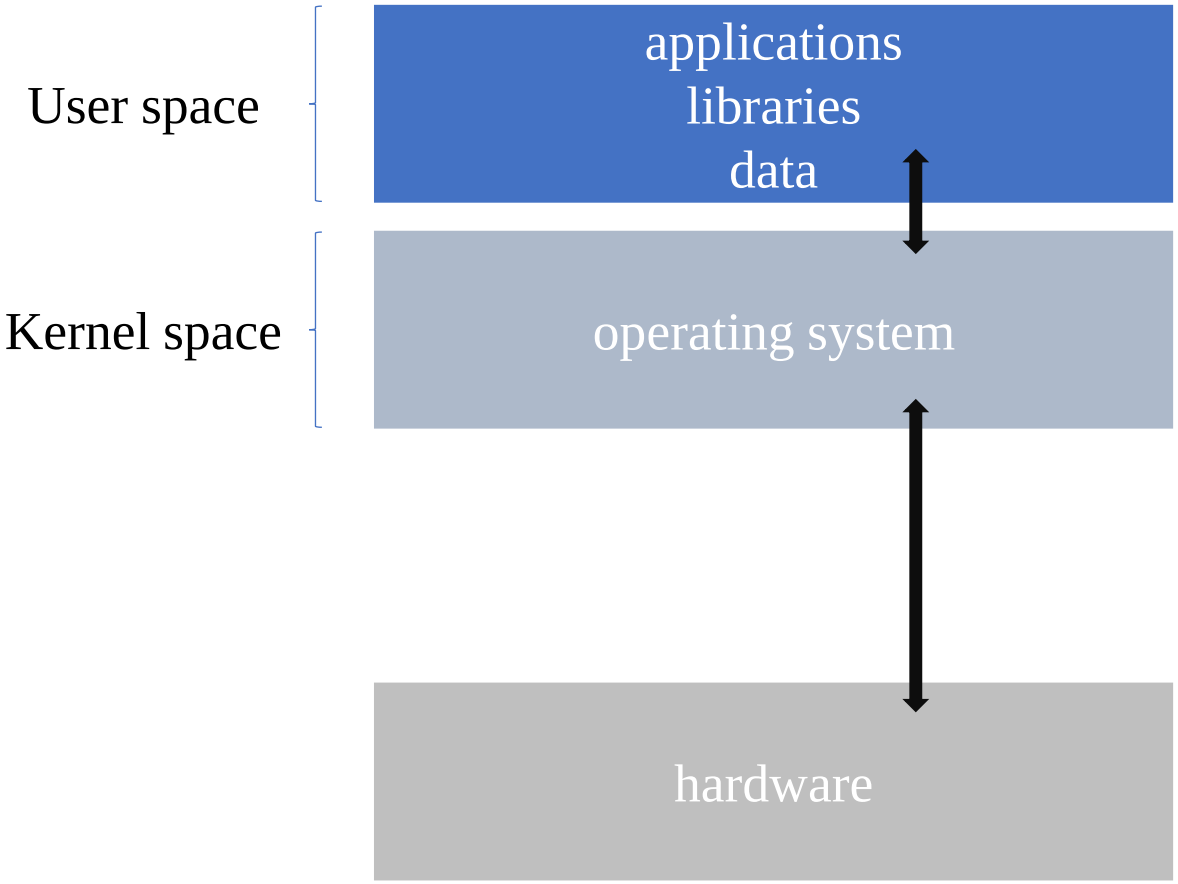
- Initial intrusion (e.g. exploit remote execution)
- Open remote access (e.g. reverse shell)
- Privilege escalation
- Download the malicious payload (our rootkit)
- Install rootkit
- Perform malicious action on command
  - DDOS
  - Steal data
  - etc…

University of BRISTOL

# Kernel rootkit

bristol.ac.uk

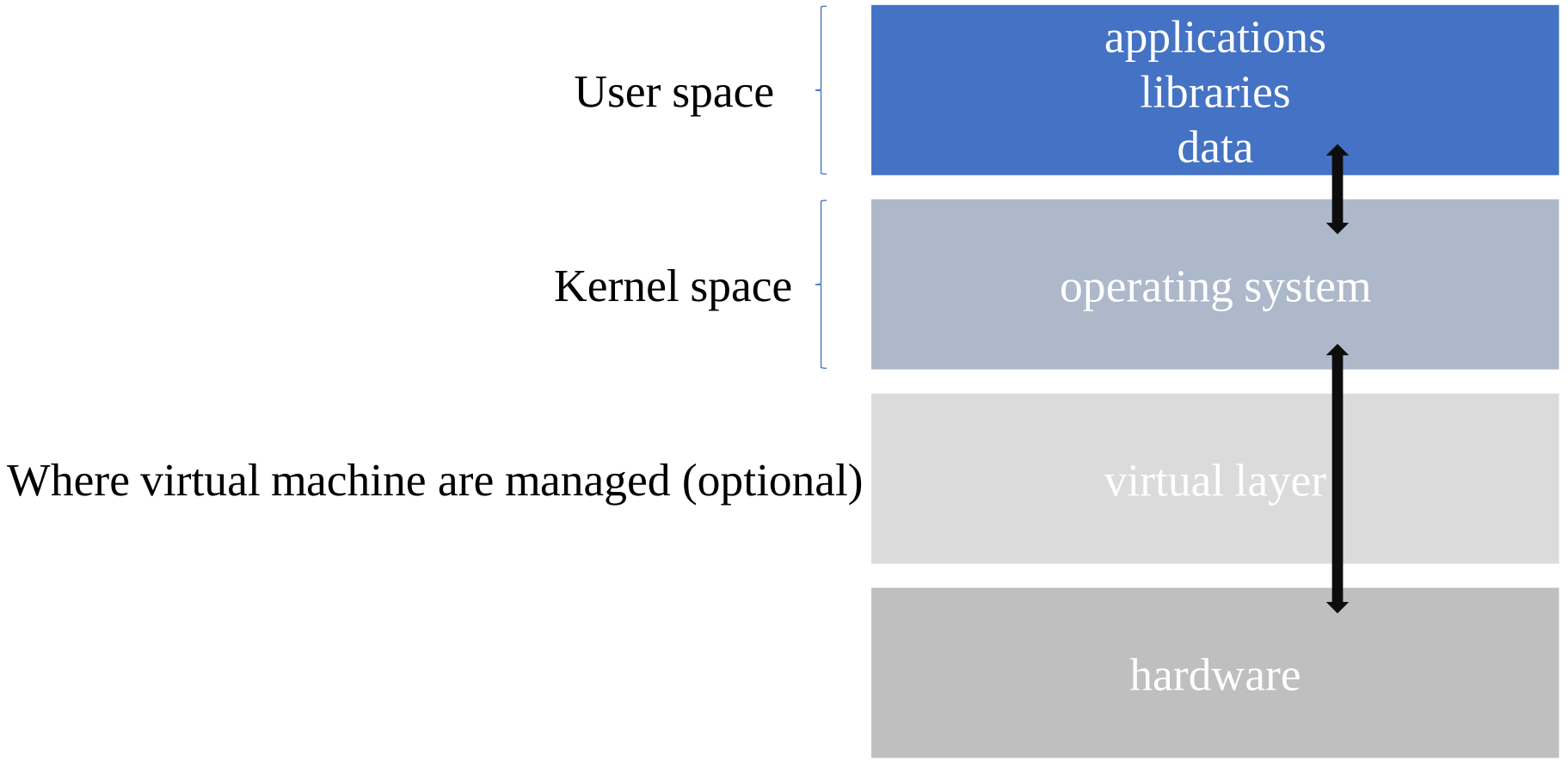
# Rootkit high-level understanding



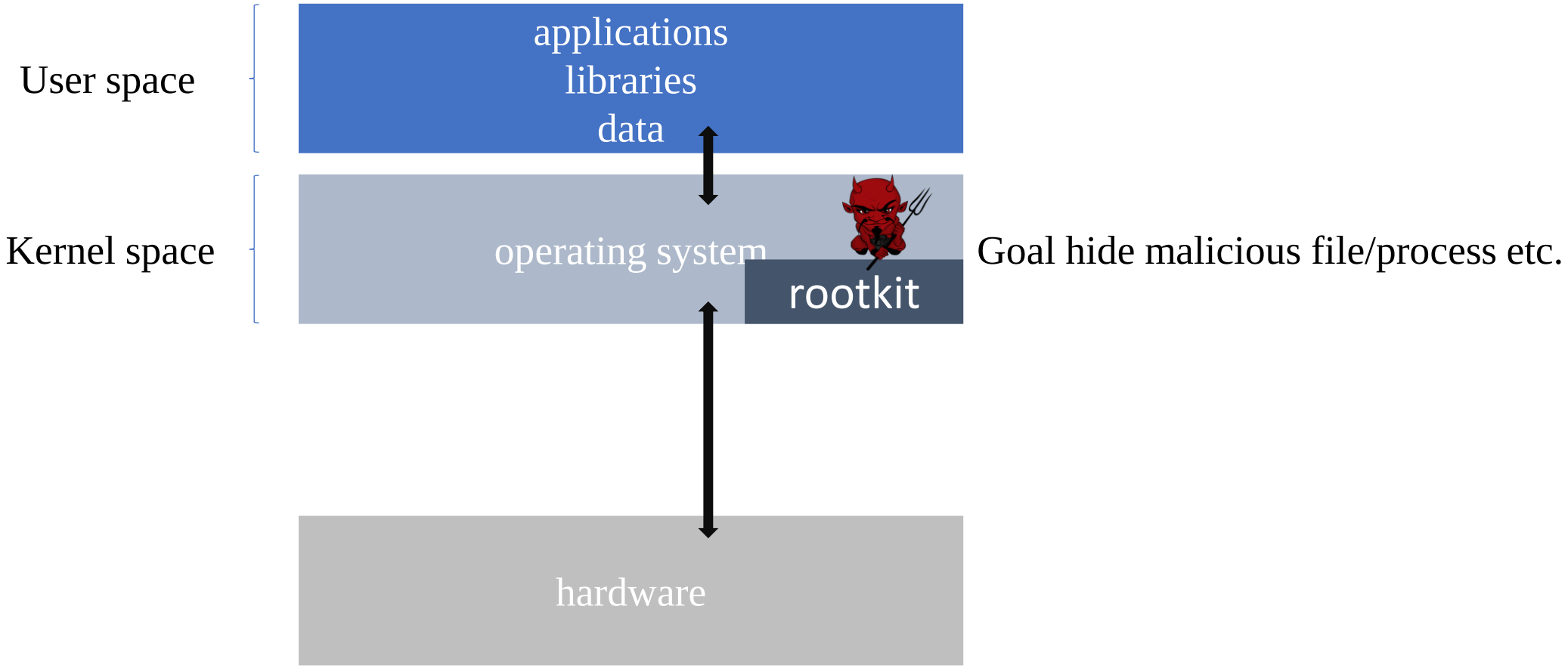

Where what user interact with exists

Where hardware can be controlled

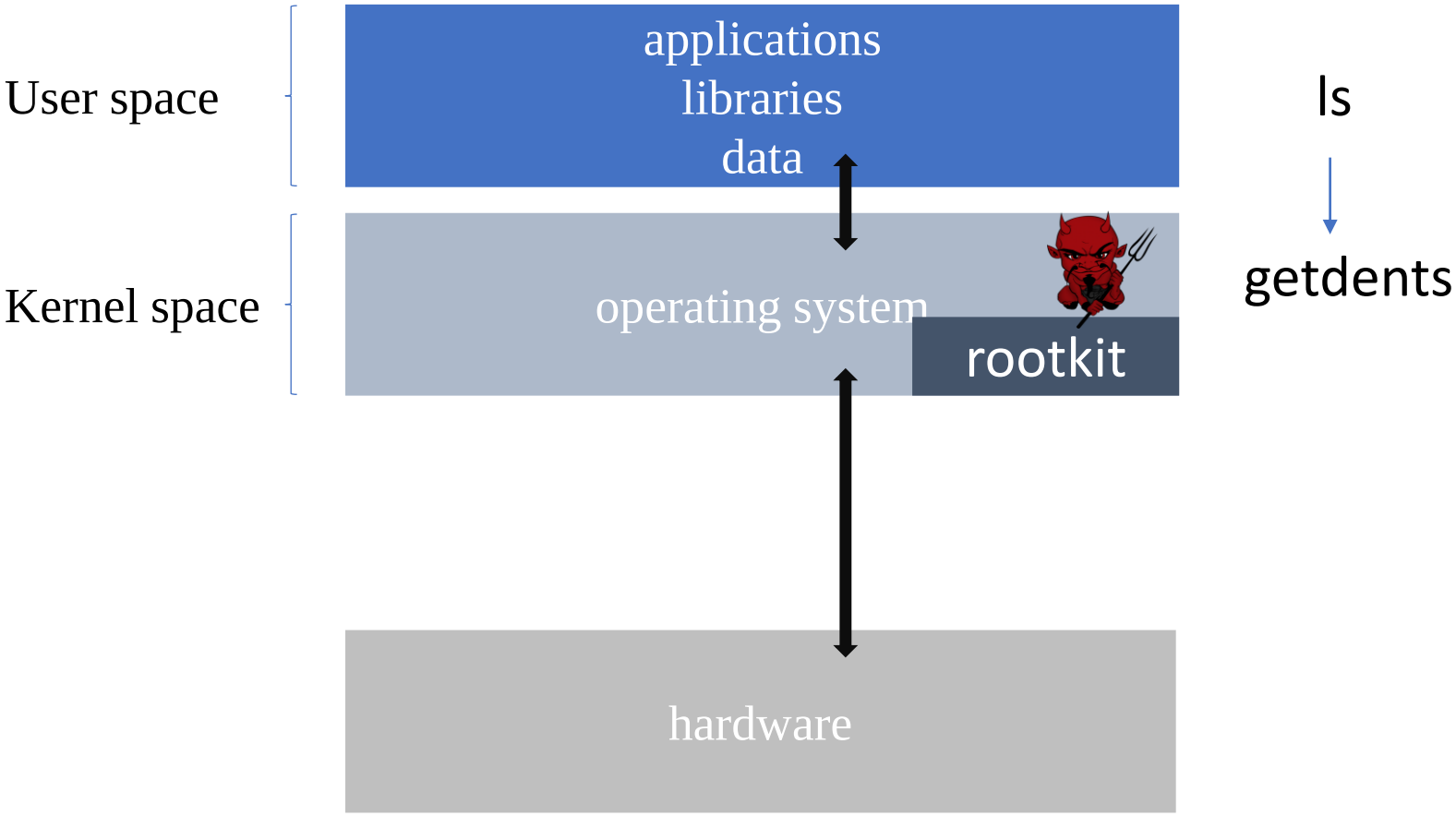
# Rootkit high-level understanding



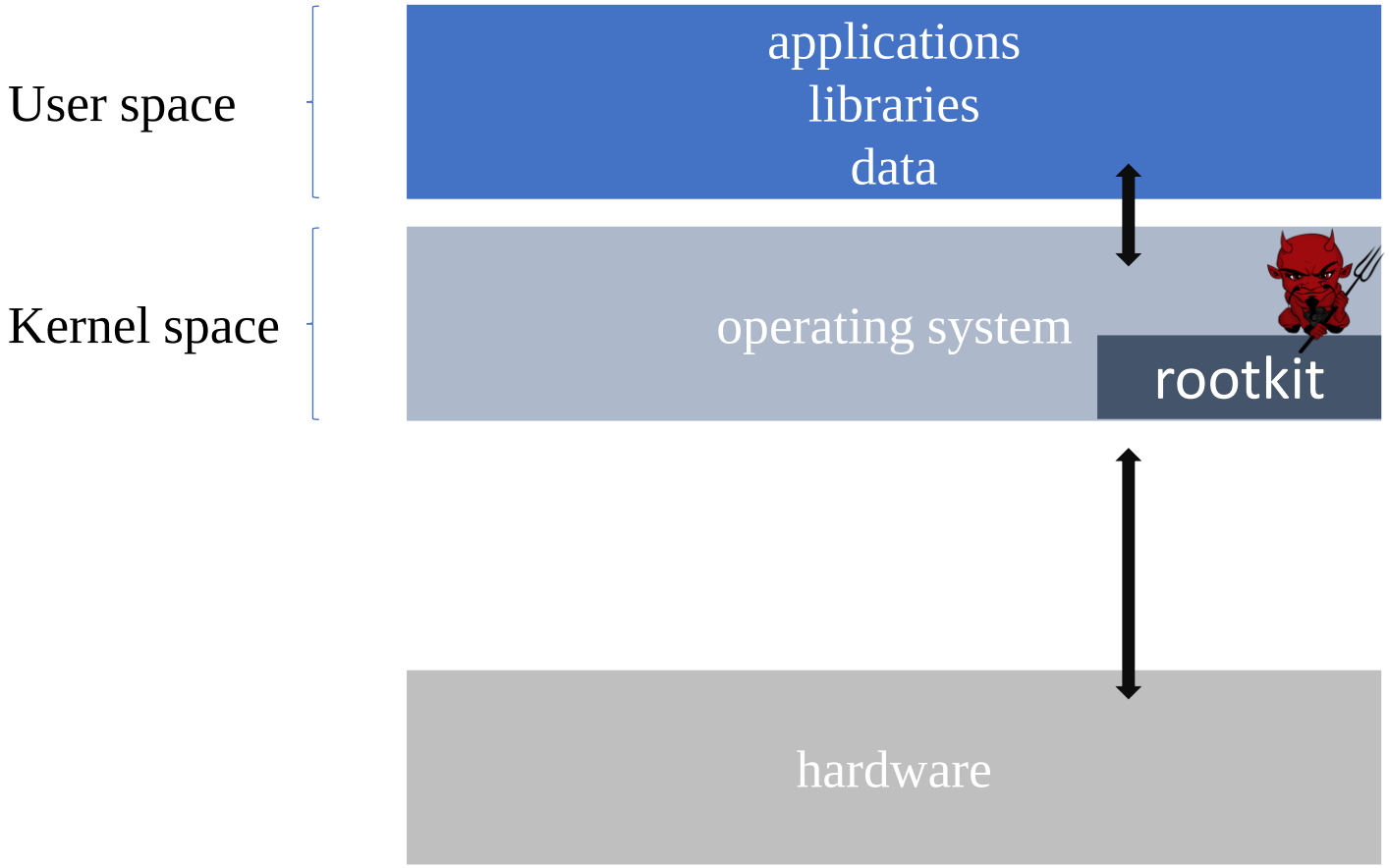
# Rootkit high-level understanding



# Rootkit high-level understanding



# Rootkit high-level understanding



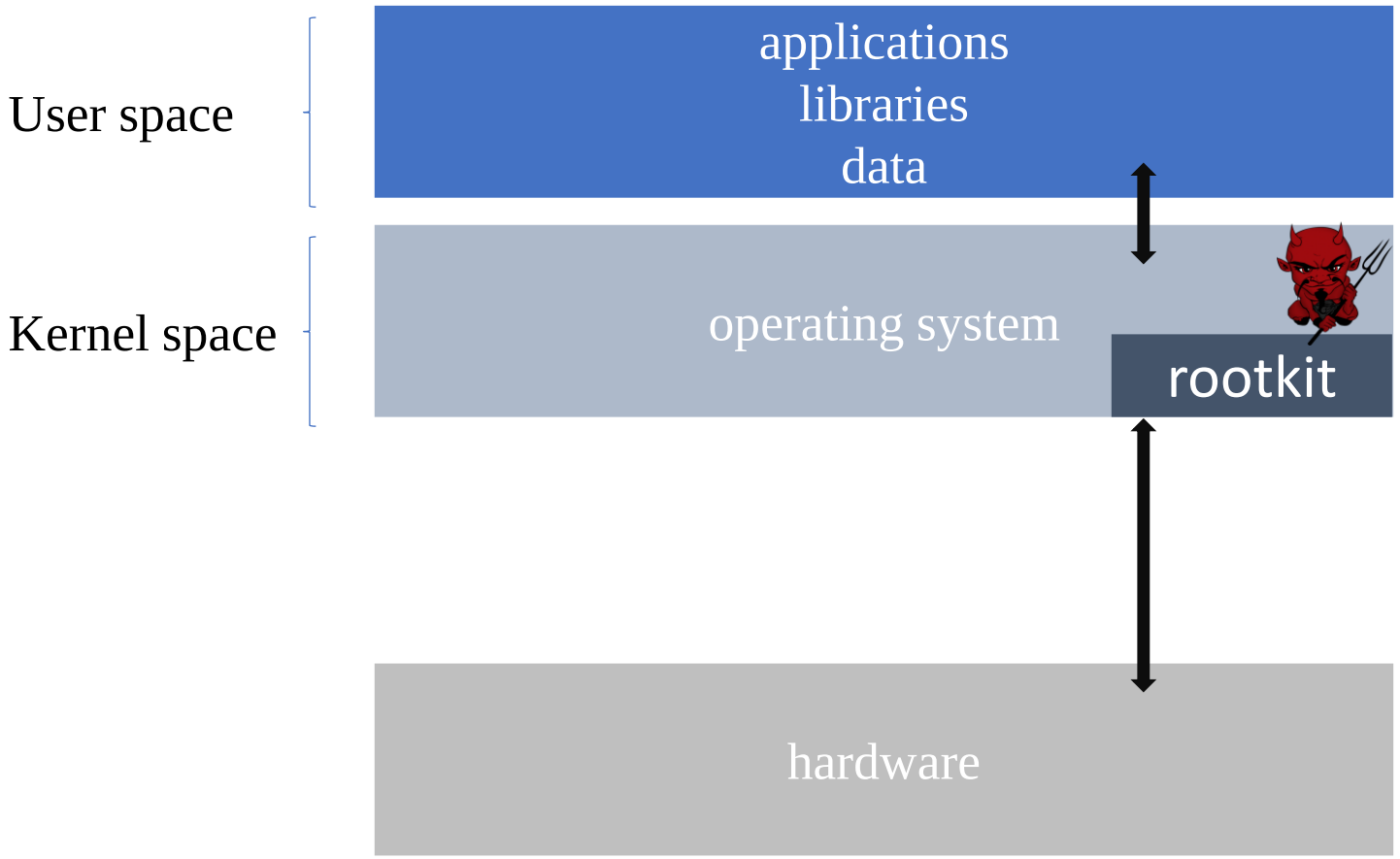

ls

Overrides getdents*

modify returned value to exclude ../malicious/

*get directory entries

# Rootkit high-level understanding

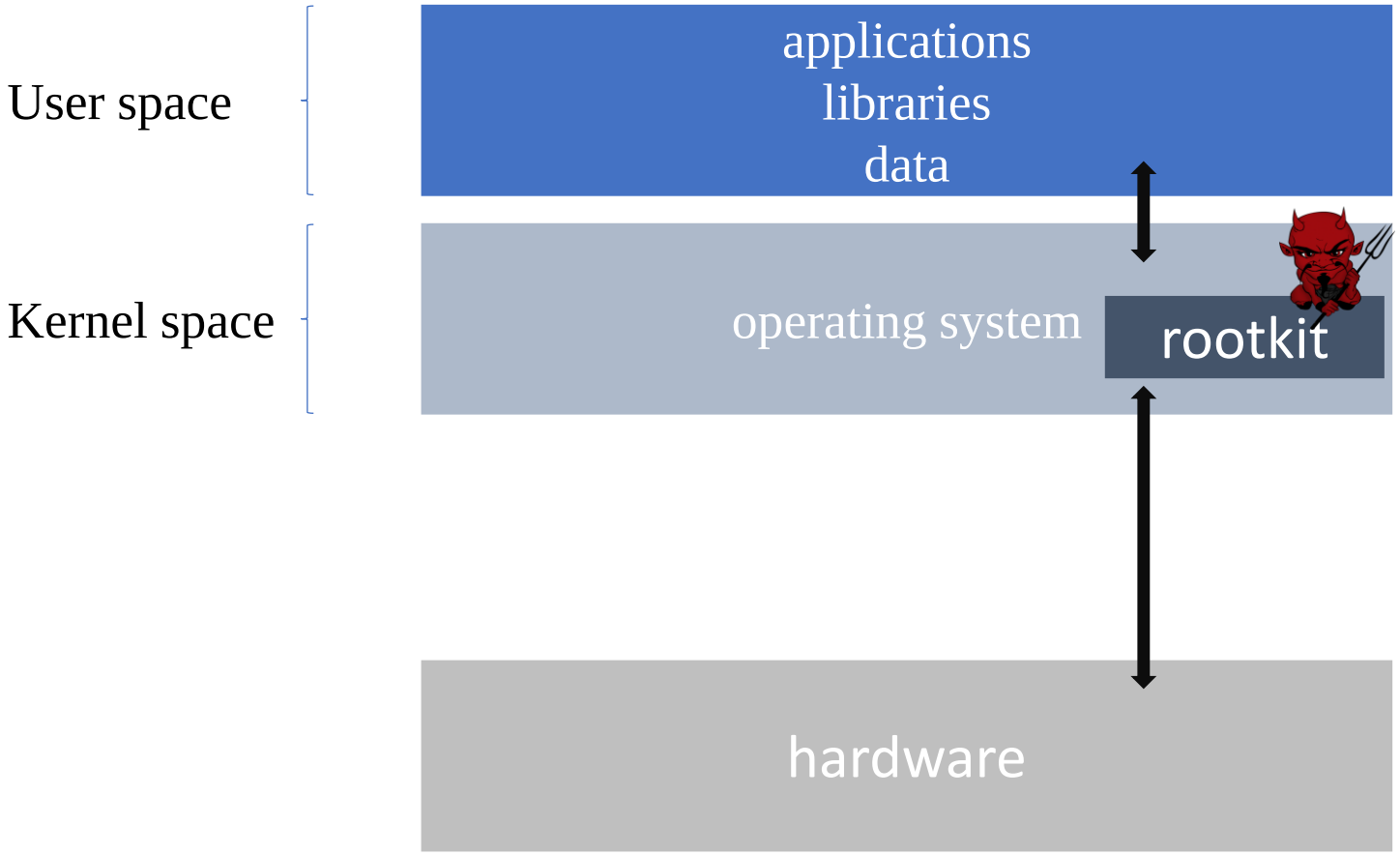



Modify the behaviour of anything that could reveal malware presence, e.g.:

- *ls*
- *ps*
- *lsmod*
- etc…

Give an easy mean to obtain root privileges

# Rootkit high-level understanding




Roughly three techniques

- Modify the kernel code;
- “Hooking” modify where certain functions point to;
- Modify data structure (e.g. active process list)

# Types of rootkit

- Application rootkit
- Kernel rootkit
- Virtualized rootkit
- Bootloader rootkit
- Hardware & firmware rootkit

# Types of rootkit

- Application rootkit
- Kernel rootkit
- Virtualized rootkit
- Bootloader rootkit
- Hardware & firmware rootkit

They can be prevented/detected by going (at least) one layer down

University of BRISTOL

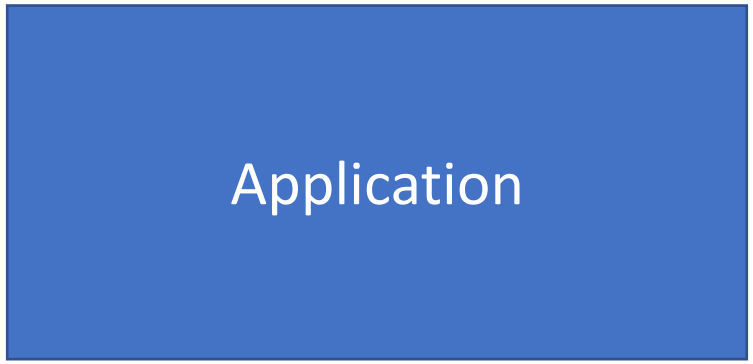
# Trusted Computing Base

bristol.ac.uk

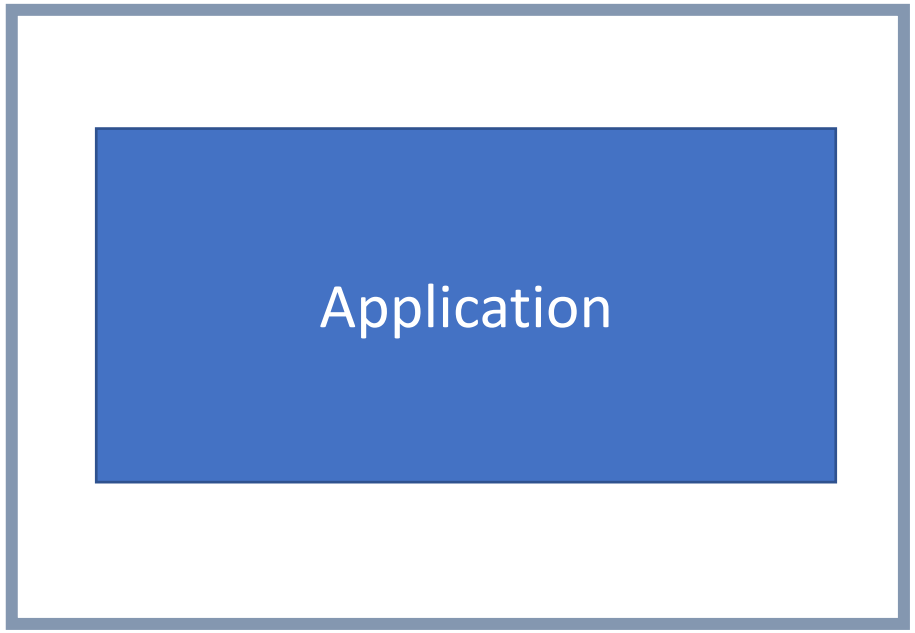
# Attack Surface

- The attack surface is all the possible ways for an attacker to compromise a “system”
  - Users;
  - Network;
  - Operating Systems;
  - Software;
  - Hardware;
  - etc.

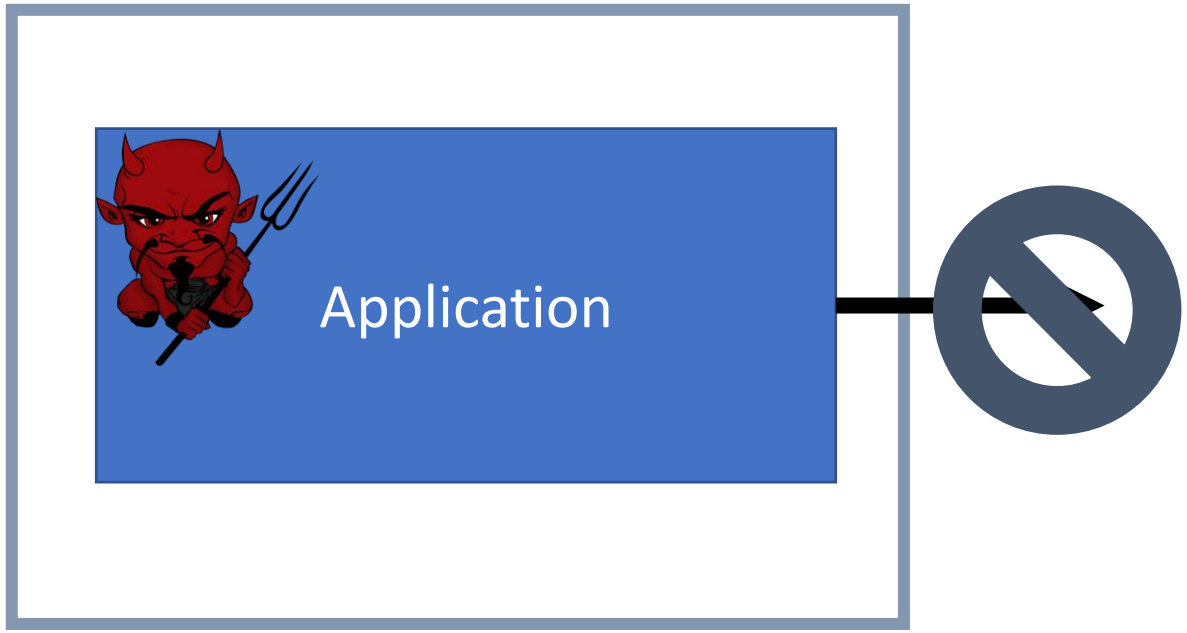
# Sandboxing



# Sandboxing

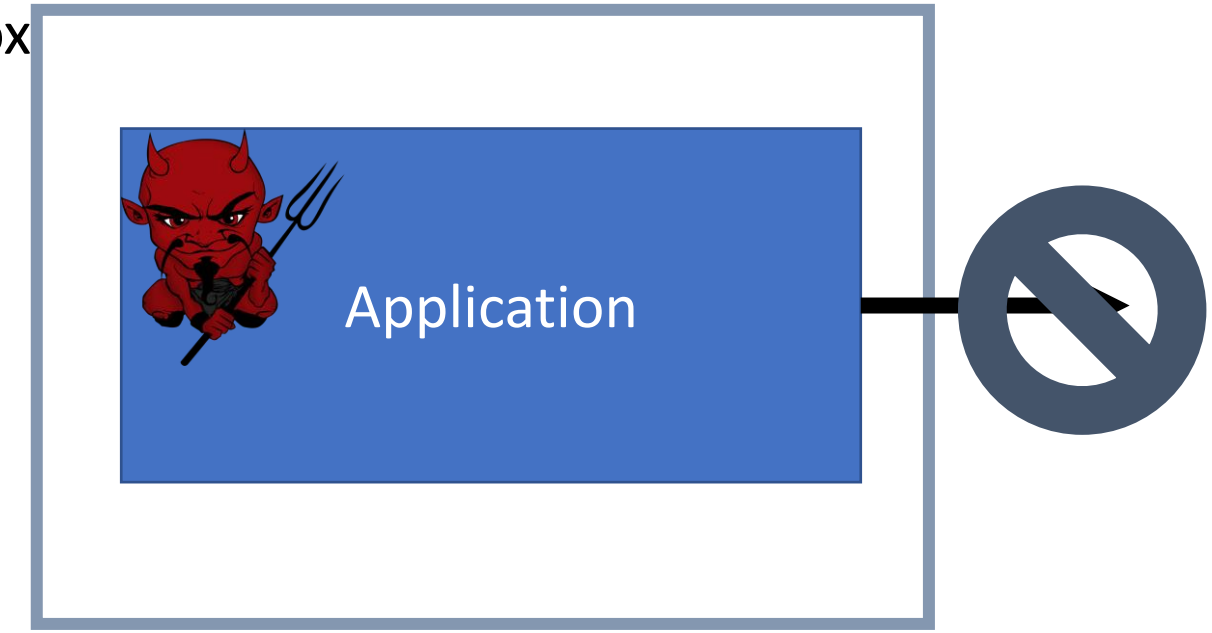


# Sandboxing

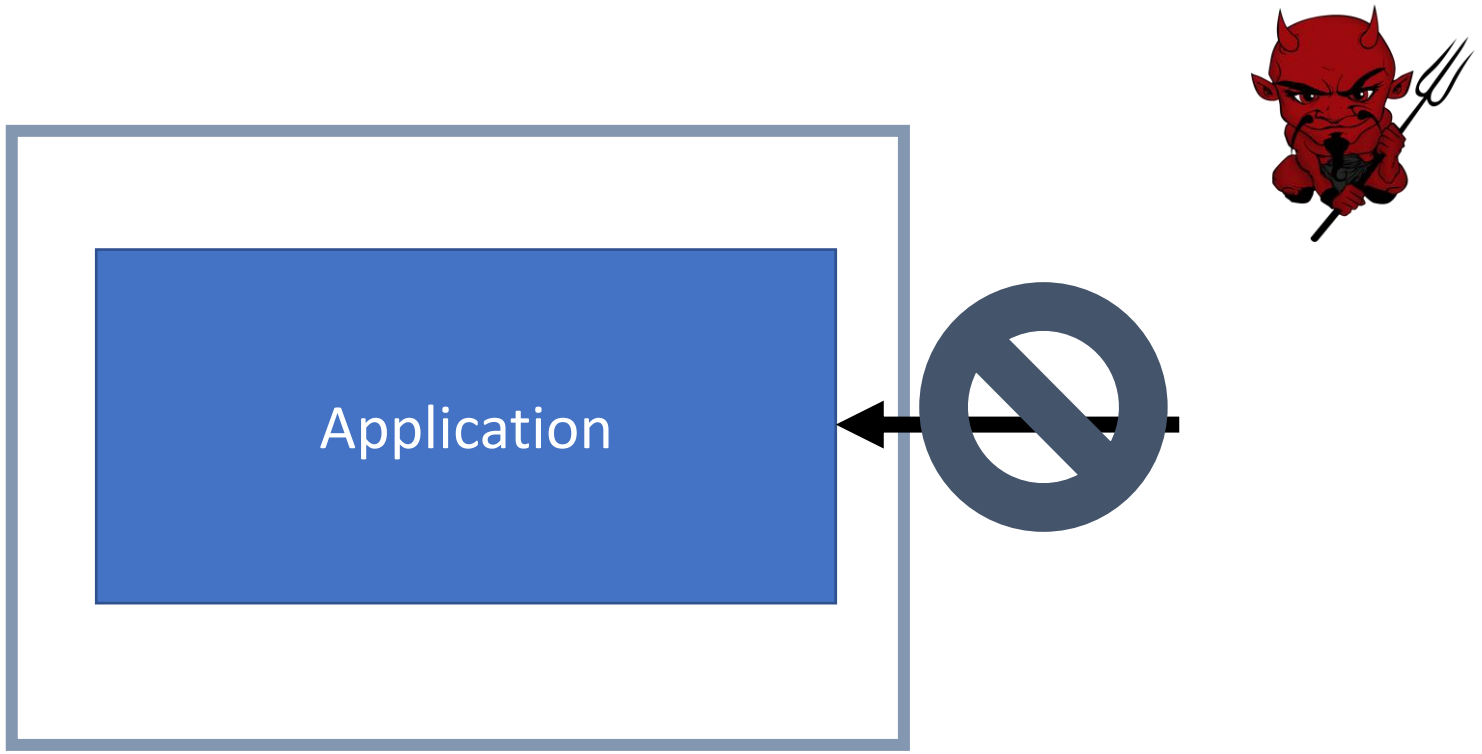


# Sandboxing

e.g. Browser Sandbox
chrome://sandbox

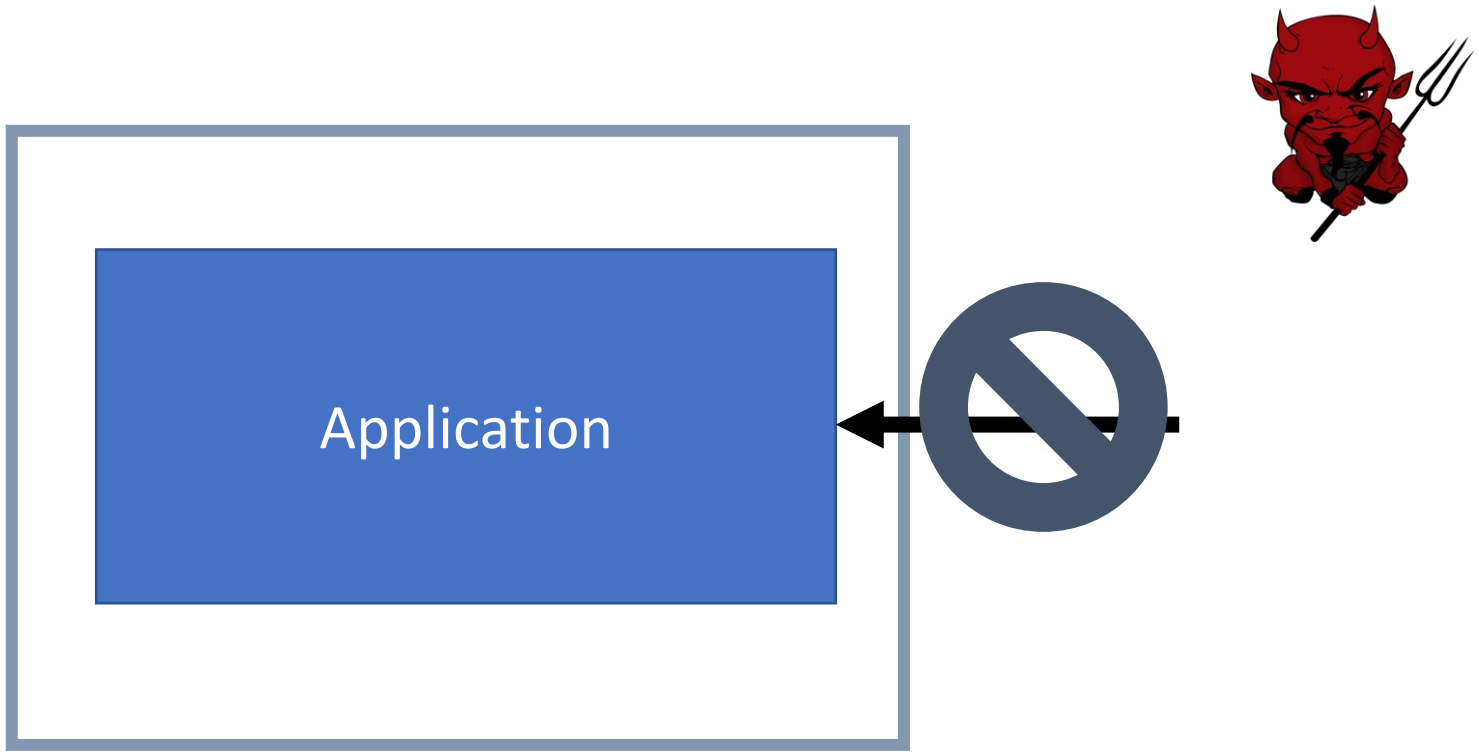


# Trusted Execution Environment



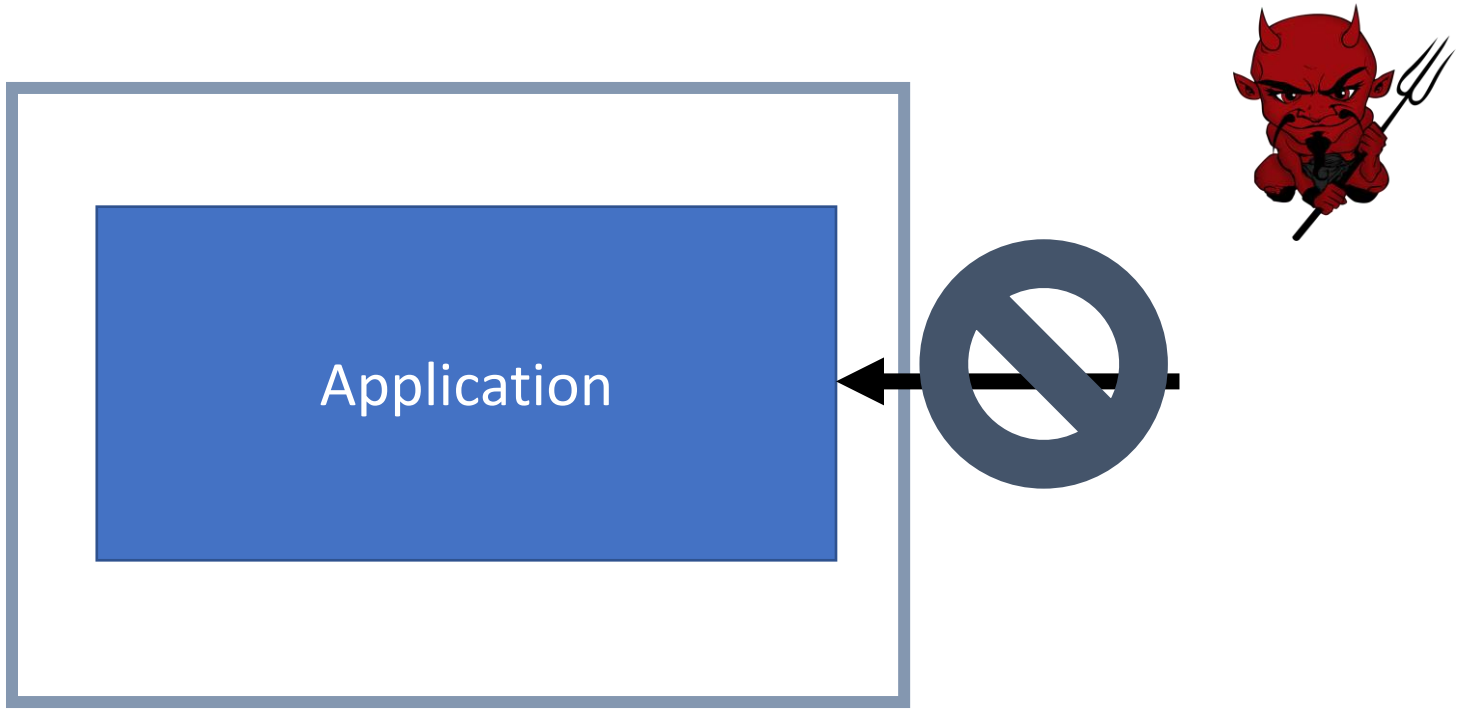
# Trusted Execution Environment

e.g. SGX enclave
see future video



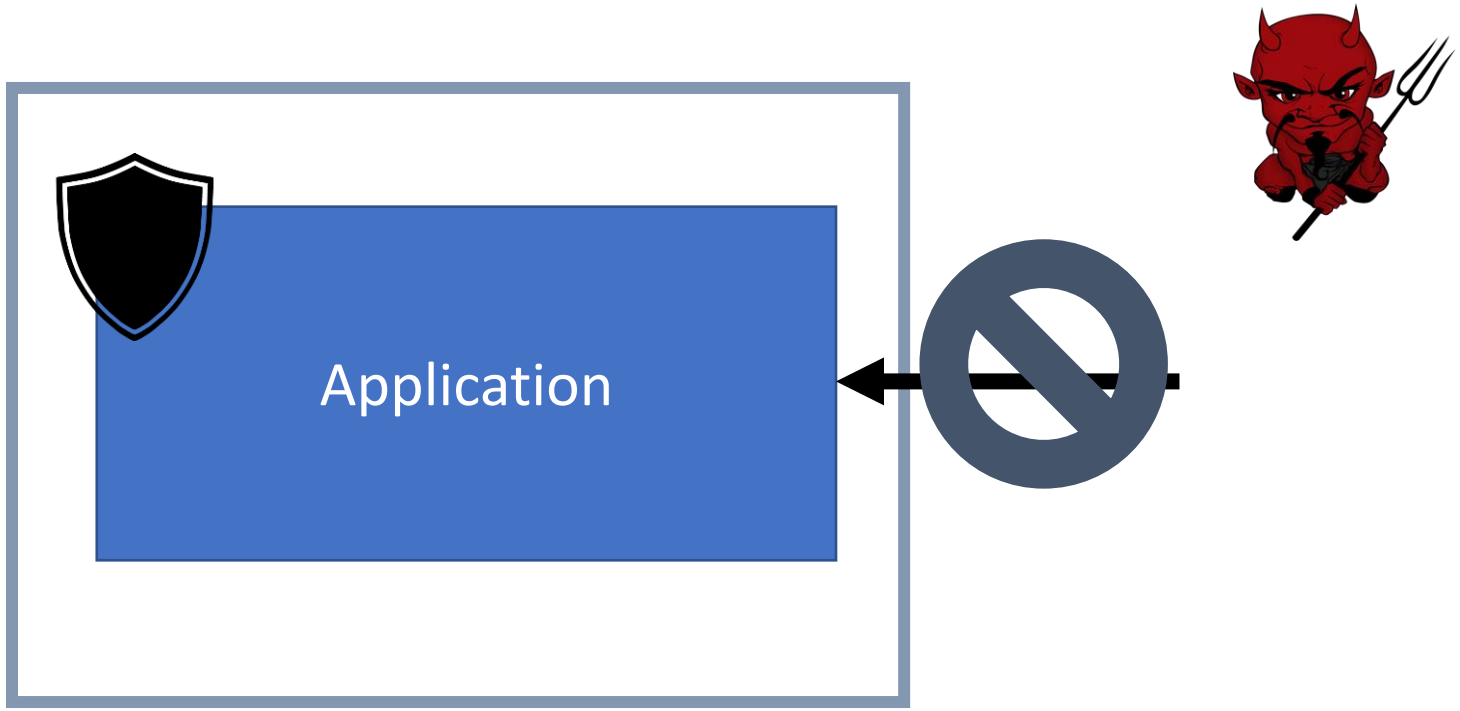
# Trusted Computing Base (TCB)

- What part of the system do I need to trust in order to achieve my objective



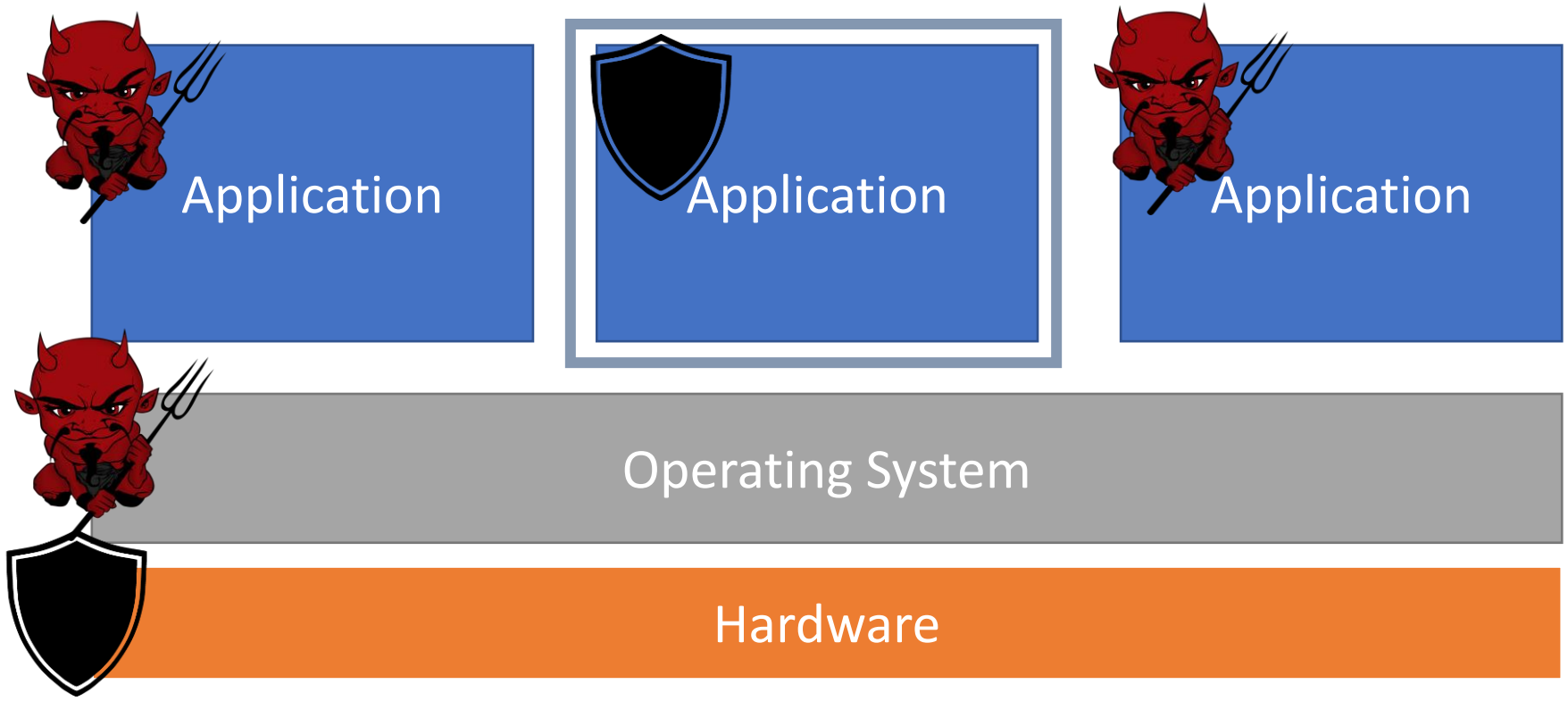
# Trusted Computing Base (TCB)

- What part of the system do I need to trust in order to achieve my objective



# Trusted Computing Base (TCB)

- What part of the system do I need to trust in order to achieve my objective



# Trusted Computing Base (TCB)

- What part of the system do I need to trust in order to achieve my objective


University of BRISTOL

# Trusted Platform Module

bristol.ac.uk

# TPM (Trusted Platform Module)

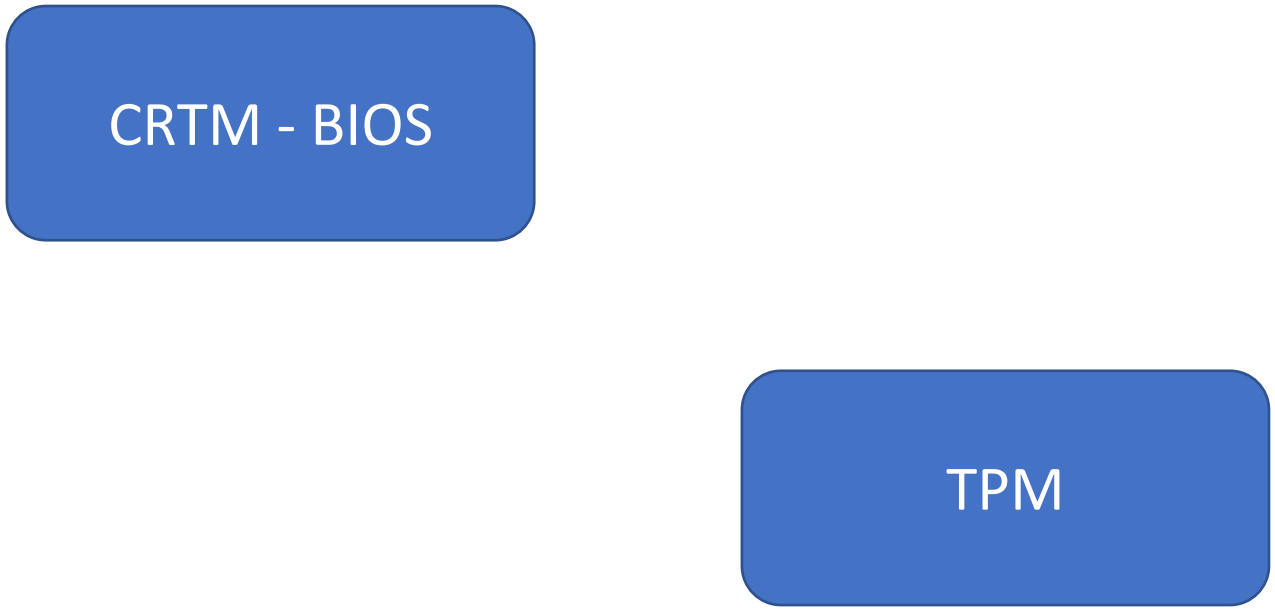
- Trusted Computing Group
  - Microsoft, Intel, IBM etc…
- Promoting standard for more trusted computing
  - Additional chip on the motherboard
  - … called TPM
- Used for
  - Disk encryption
  - **System Integrity**
  - Password protection
  - … and more

# Requirements

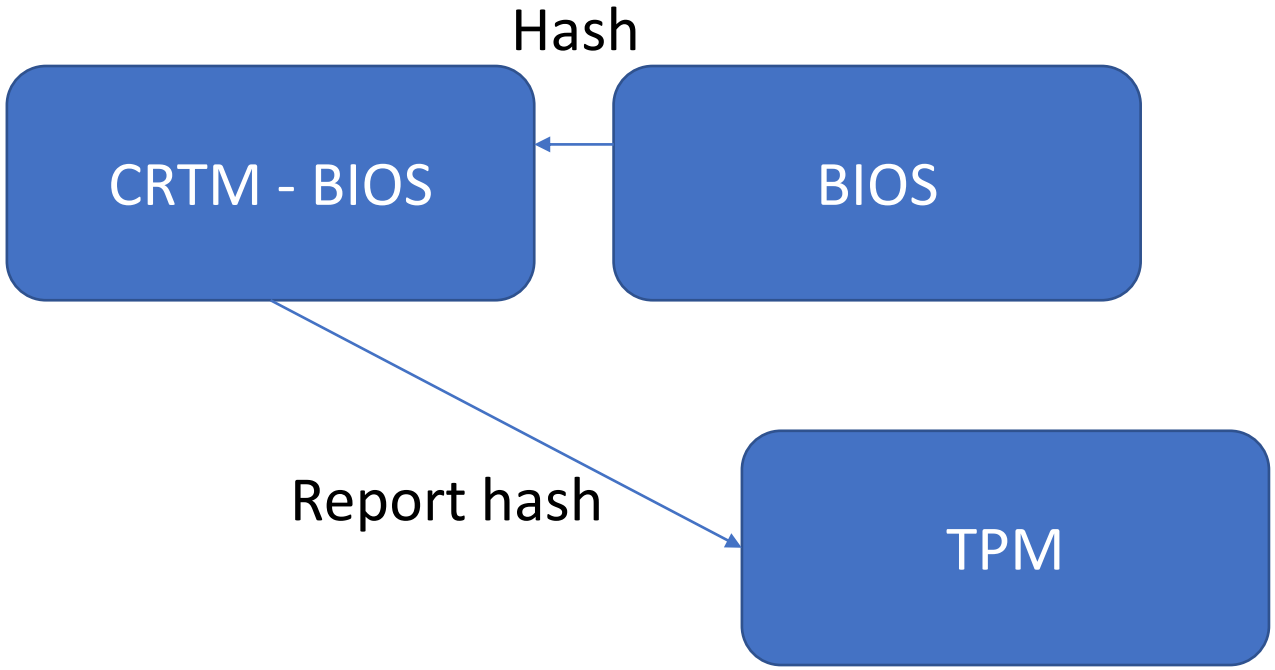
- We can achieve trust if we can verify the system has booted correctly
- We assume the PC hardware has not been modified
  - Key function is in the hardware TPM
- We need to monitor the boot process
  - Initial boot measure by the “Core Root of Trust” (CRTM)
  - Hash the BIOS, store results in TPM, start the BIOS
  - BIOS do its job, load the next stage, hash it store in TPM etc…

**Core Root of Trust :**The first piece of BIOS code that executes on the main processor during the boot process. On a system with a Trusted Platform Module the CRTM is implicitly trusted to bootstrap the process of building a measurement chain for subsequent attestation of other firmware and software that is executed on the computer system.

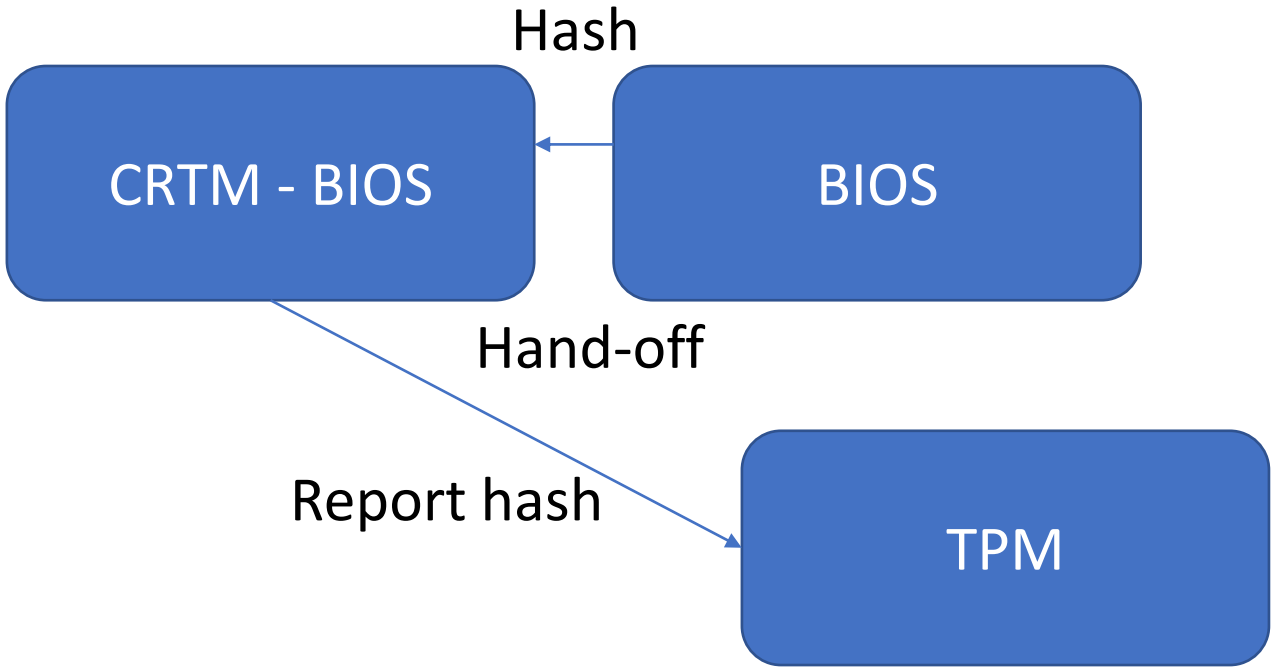
# Authenticated Boot



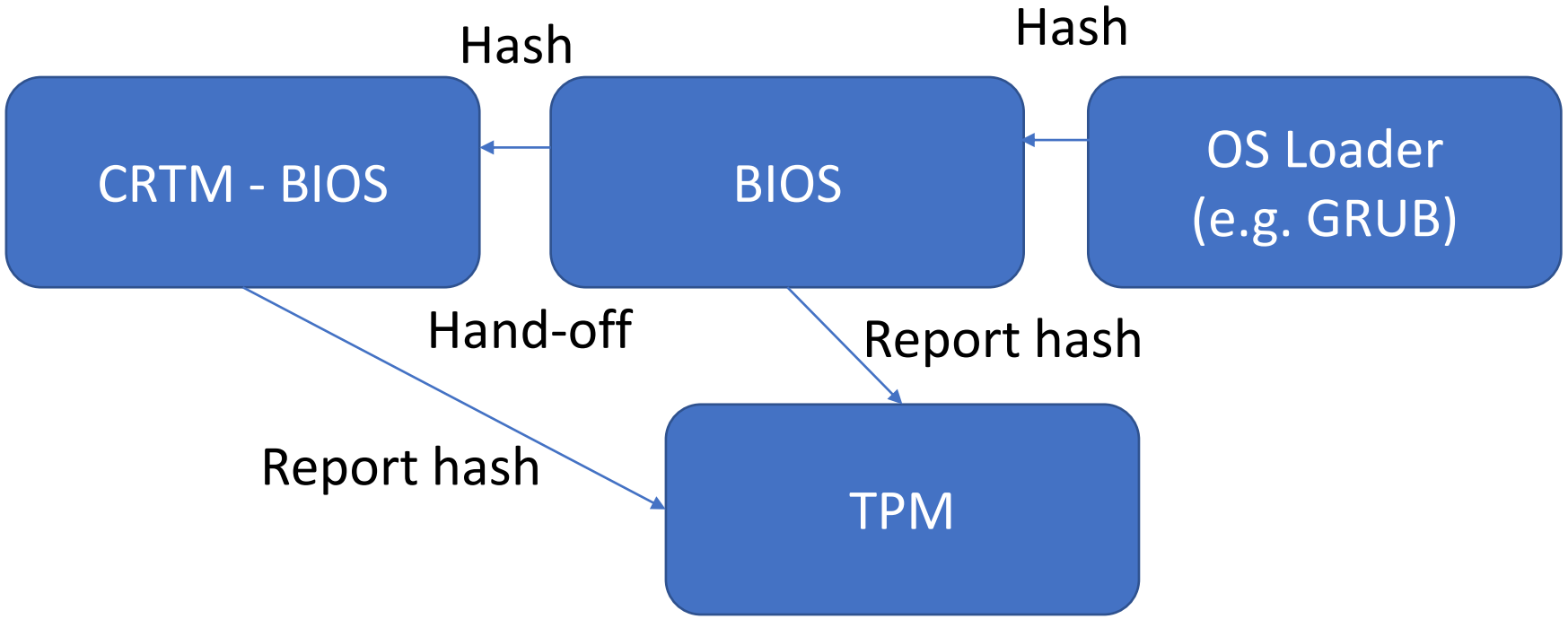
# Authenticated Boot



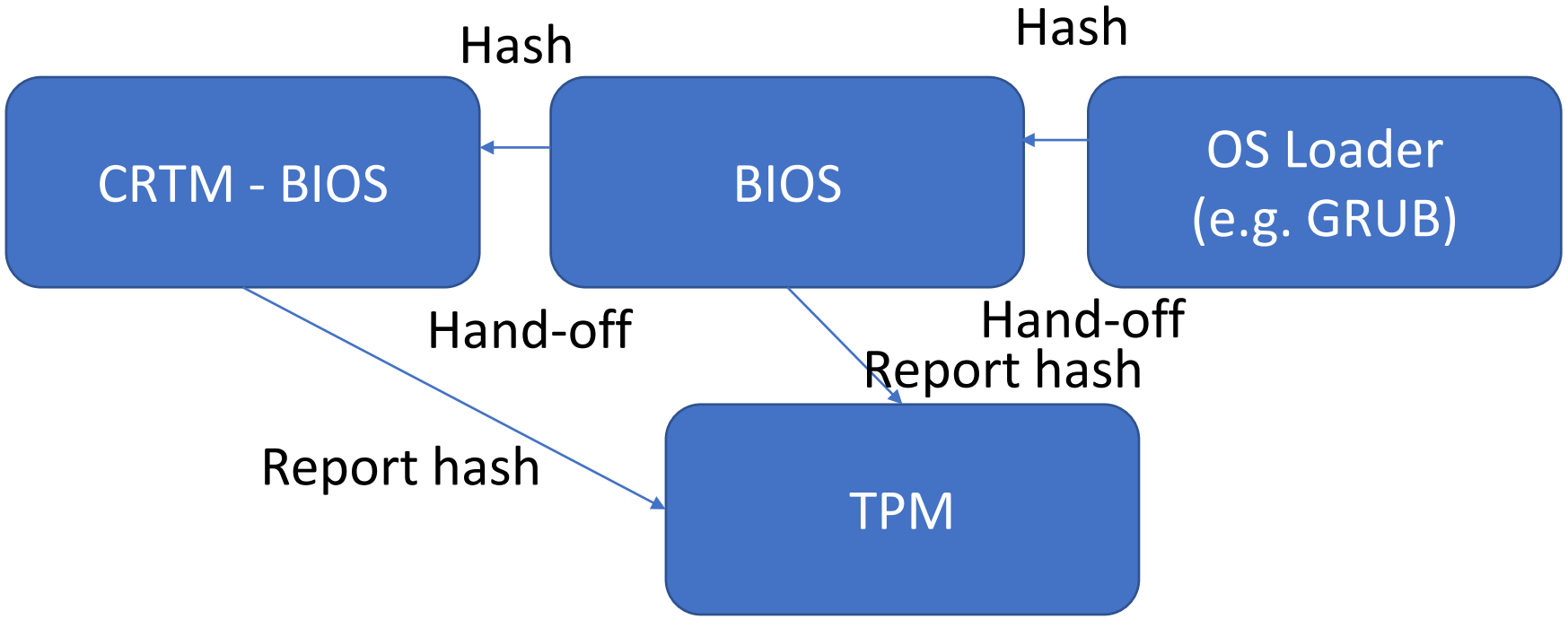
# Authenticated Boot



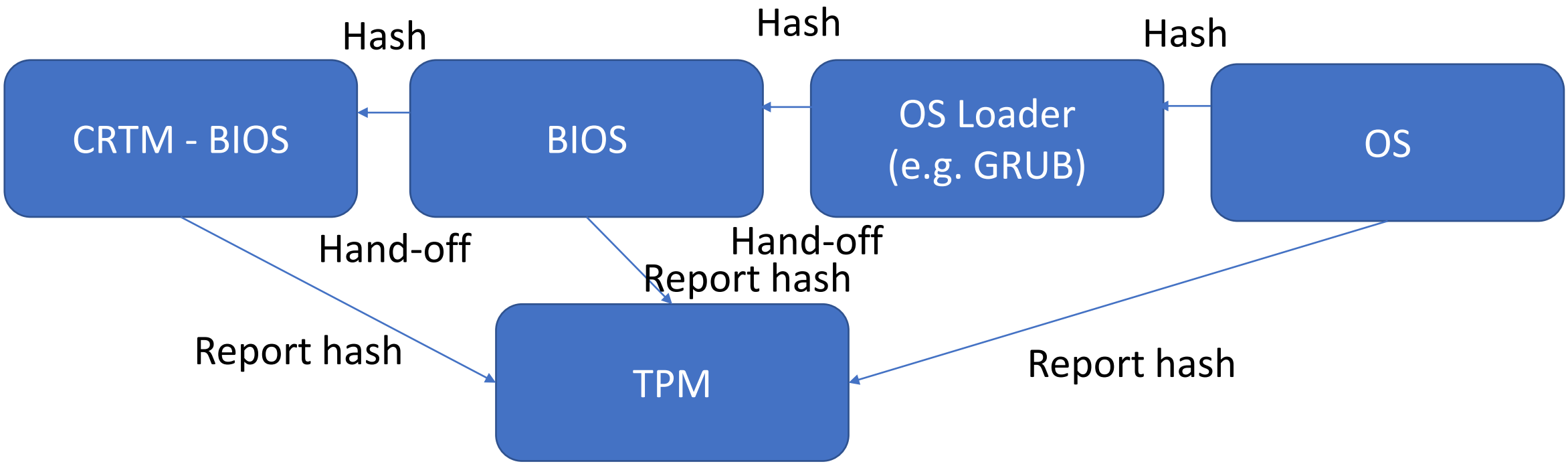
# Authenticated Boot



# Authenticated Boot



# Authenticated Boot



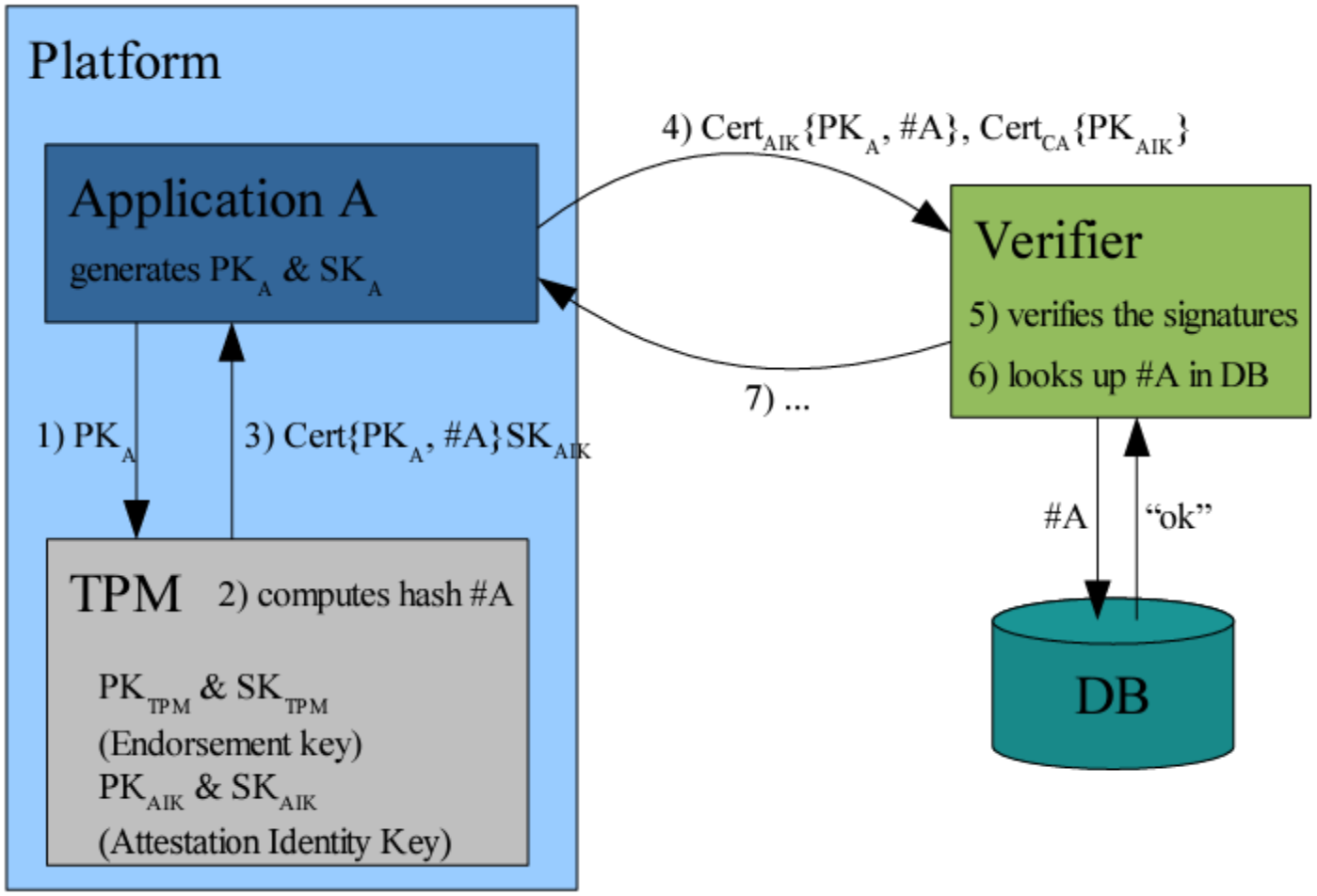
# TPM registers

- Platform configuration registers (PCRs)
  - Used to store platform integrity metrics
- A PCR hold a summary of a series of value
  - Not the entire chain of hash
  - The chain can be infinite
- A PCR register is extended
  - PCR = HASH(PCR | new measurement)
  - Shielded TPM location (i.e. cannot be modified from outside)
  - Measurement are provided by software

# Remote attestation

- Attestation is a mechanism for software to prove its identity.
- The goal of attestation is to prove to a remote party that your operating system and application software are intact and trustworthy.
- The verifier trusts that attestation data is accurate because it is signed by a TPM whose key is certified by the CA.

# Basic Remote attestation




1. The application "A" generates a public/private key pair PKA & SKA and asks the TPM to certify it.

2. The TPM computes a hash value #A of the executable code of program "A".

3. The TPM creates a certification including PKA and #A and signs it with the attestation identity key SKAIK.

4.When application "A" wishes to authenticate itself to a remote party, it sends the cert. of its public key and hash value #A along with a cert. issued to the TPM by a trusted certification authority (CA).

5. The remote party to verifies the cert. chain.

6. The remote party looks #A up in a database which maps hash values to trust levels.

7. If application "A" is deemed trustworthy, we continue the communication, probably by using PKA to establish a session key.

# What remote attestation tells you

Positive result

- Correct memory content
- Good device

Negative result

- Malfunctioning device
- Malicious device

No response

- Malfunctioning device
- Malicious device

# TPM and Remote Attestation

- PCR cannot be modified
  - Only reset at reboot
- TPM contains a key used to sign the attestation
- Verifier
  - Verify the TPM certificate/key
  - Verify the PCRs
- Attestation
  - PCRs value
  - sign(PCRs, challenge[nonce])

# TPM and Remote Attestation

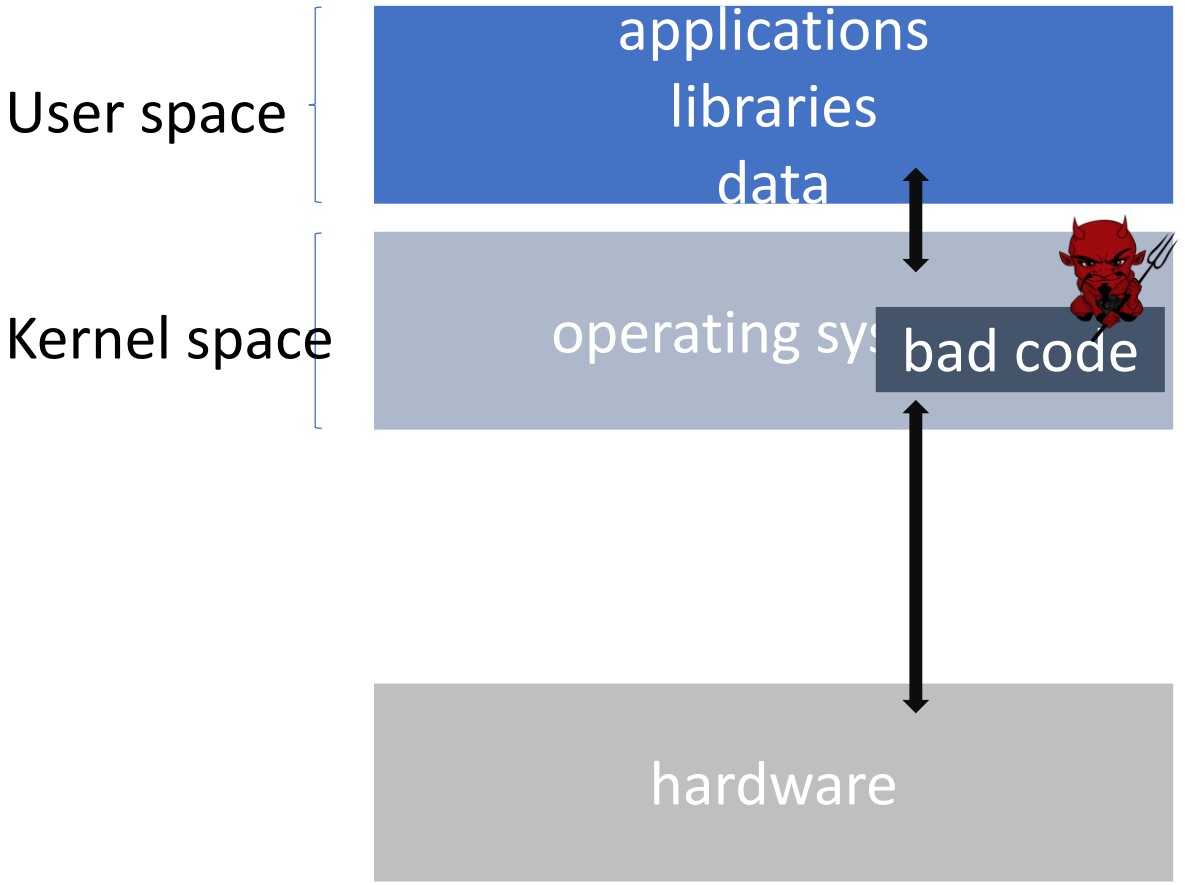
- You need not to stop at the OS
  - Can attest kernel modules (e.g. drivers)
  - Applications?
  - Configurations?
  - Scripts?

University of BRISTOL

# Intel SGX

bristol.ac.uk

# Rootkit high-level understanding



# Motivation

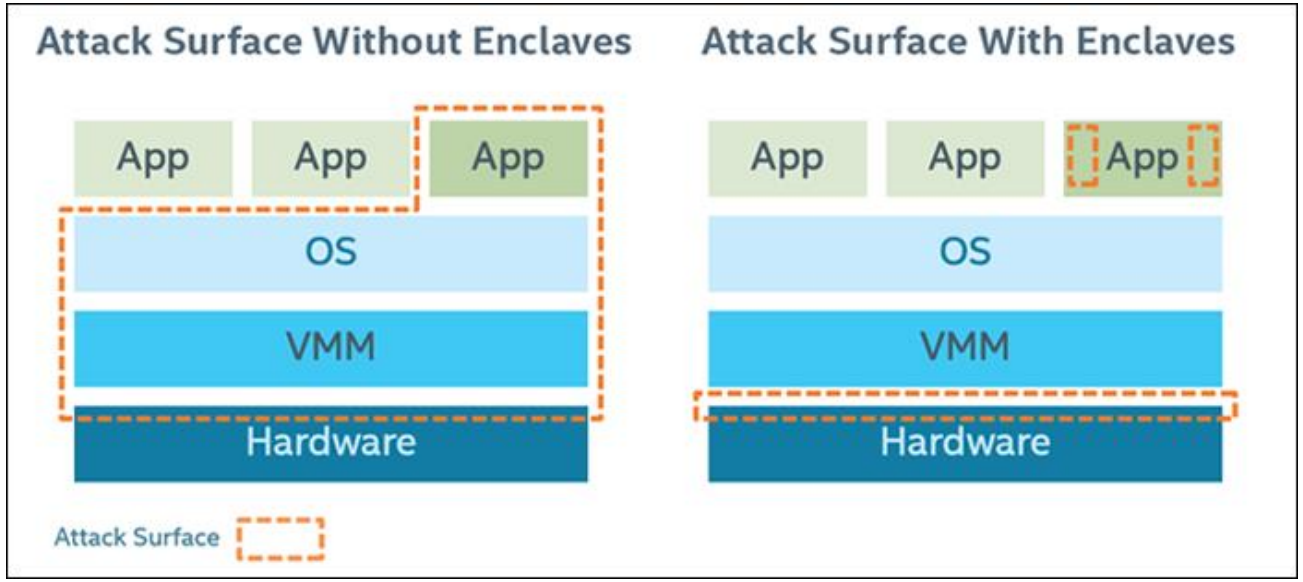
- An attacker can compromise
  - User space
  - Operating Systems
  - Even the hardware!
- What can we do?

Execute code in its own secure enclave!

# SGX Hardware supported enclave

**Idea:** run an application within some isolation unit so it cannot be affected by the OS

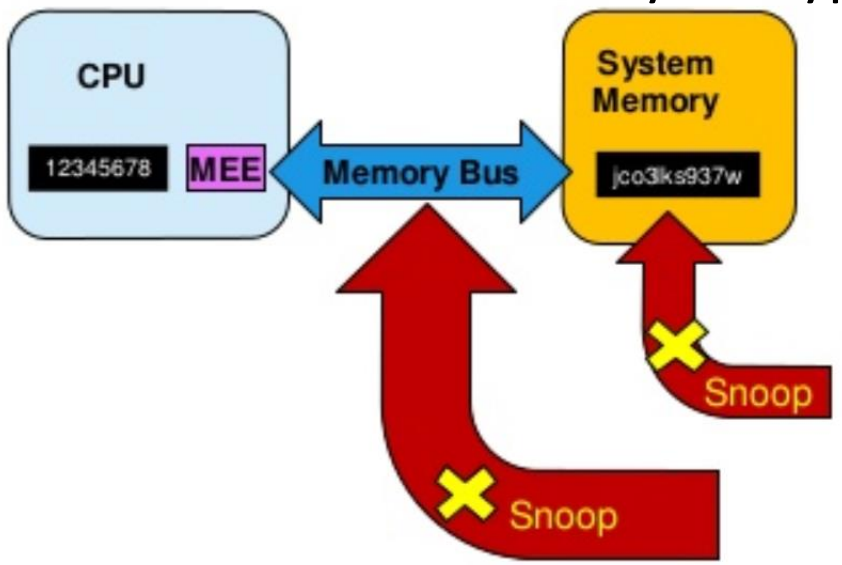
- Do not trust the OS or the VMM/hypervisor
- only need to trust the hardware
- reduce attack surface



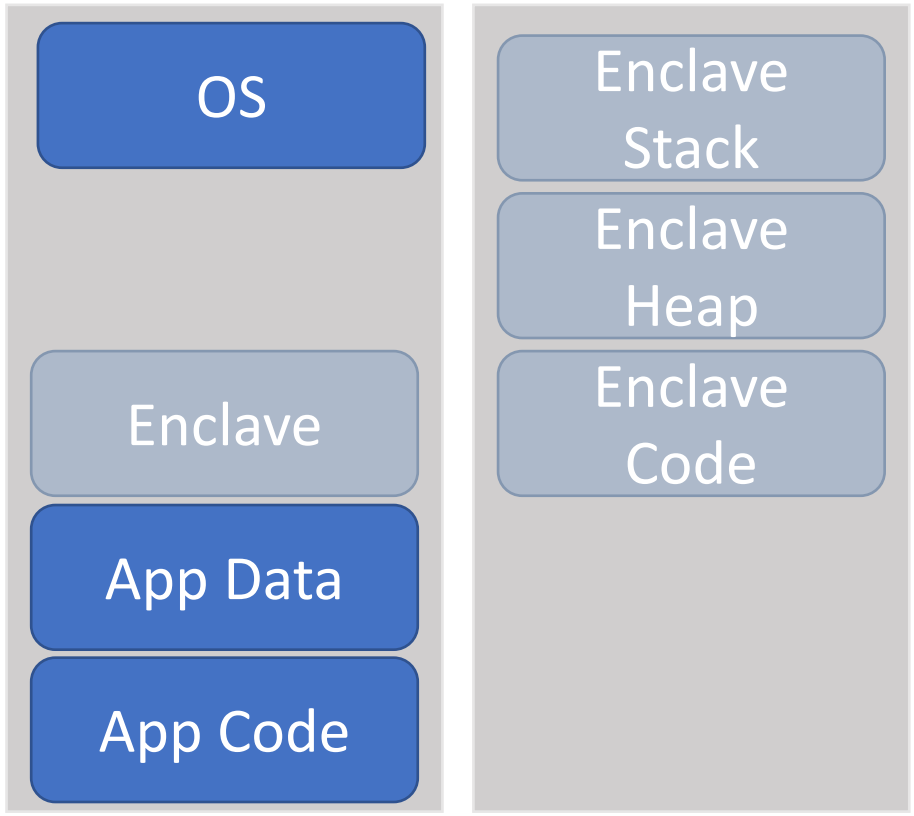
# SGX preventing memory snooping attack

- Security boundary is CPU package
- Data unencrypted inside the CPU
- Data outside the CPU is encrypted
- External memory reads and bus snooping only see encrypted data

* MEE: SGX Memory Encryption Engine



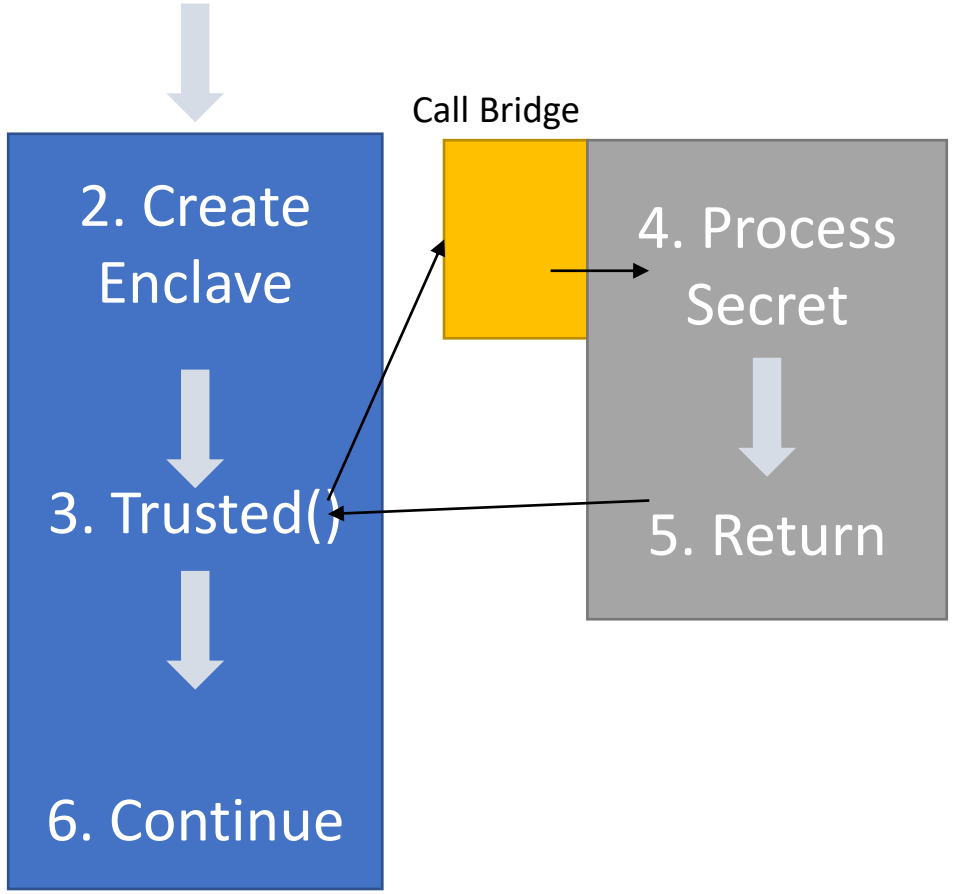
# SGX Programming environment




- Enclave has its own code and data
  - Provide confidentiality
  - Provide integrity
- Controlled entry point
  - Can enter enclave code only at specific point
  - Enclave execution takes over

# SGX Application Flow

1. Define and partition application into trusted and untrusted part
2. App create enclave
3. Trusted function is called
4. Code in enclave process some secret
5. Trusted function returns
6. App continue as normal

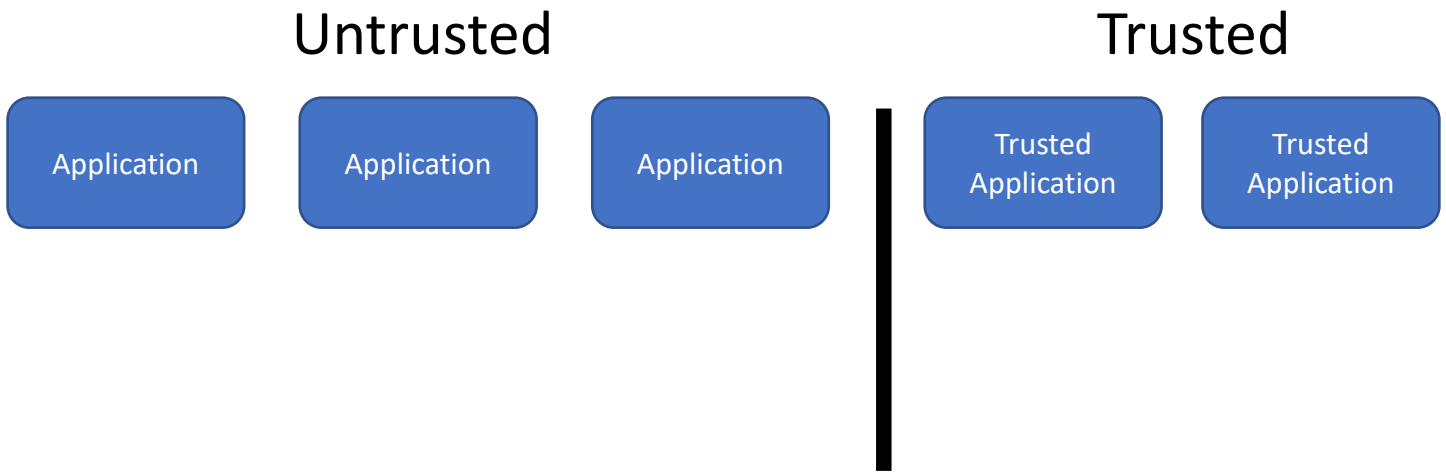


University of BRISTOL

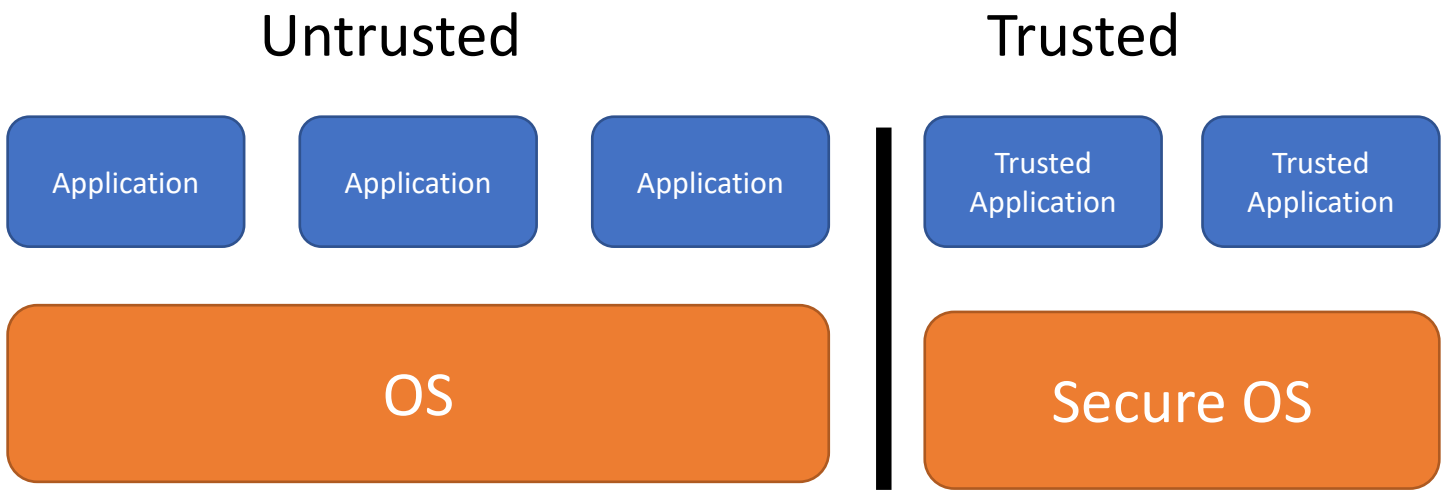
# ARM Trustzone

bristol.ac.uk

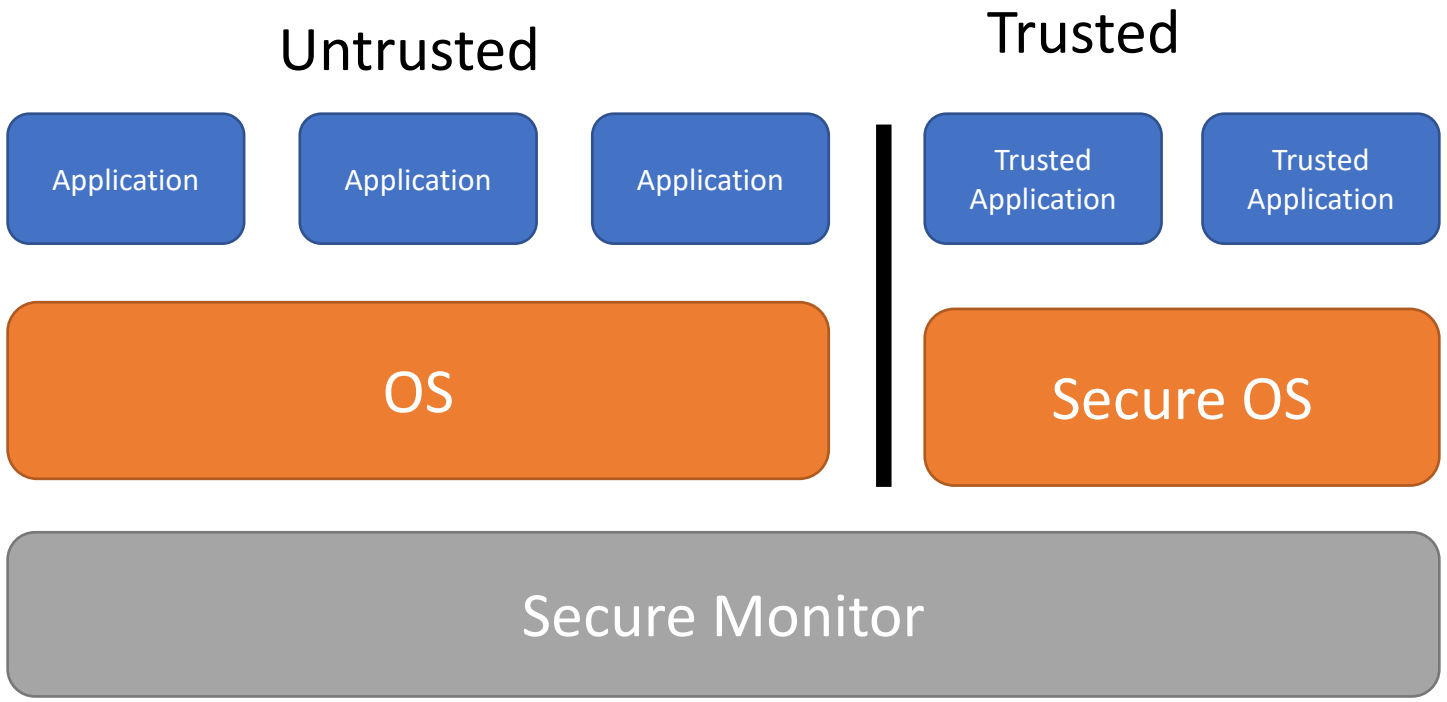
# ARM Trustzone



# ARM Trustzone

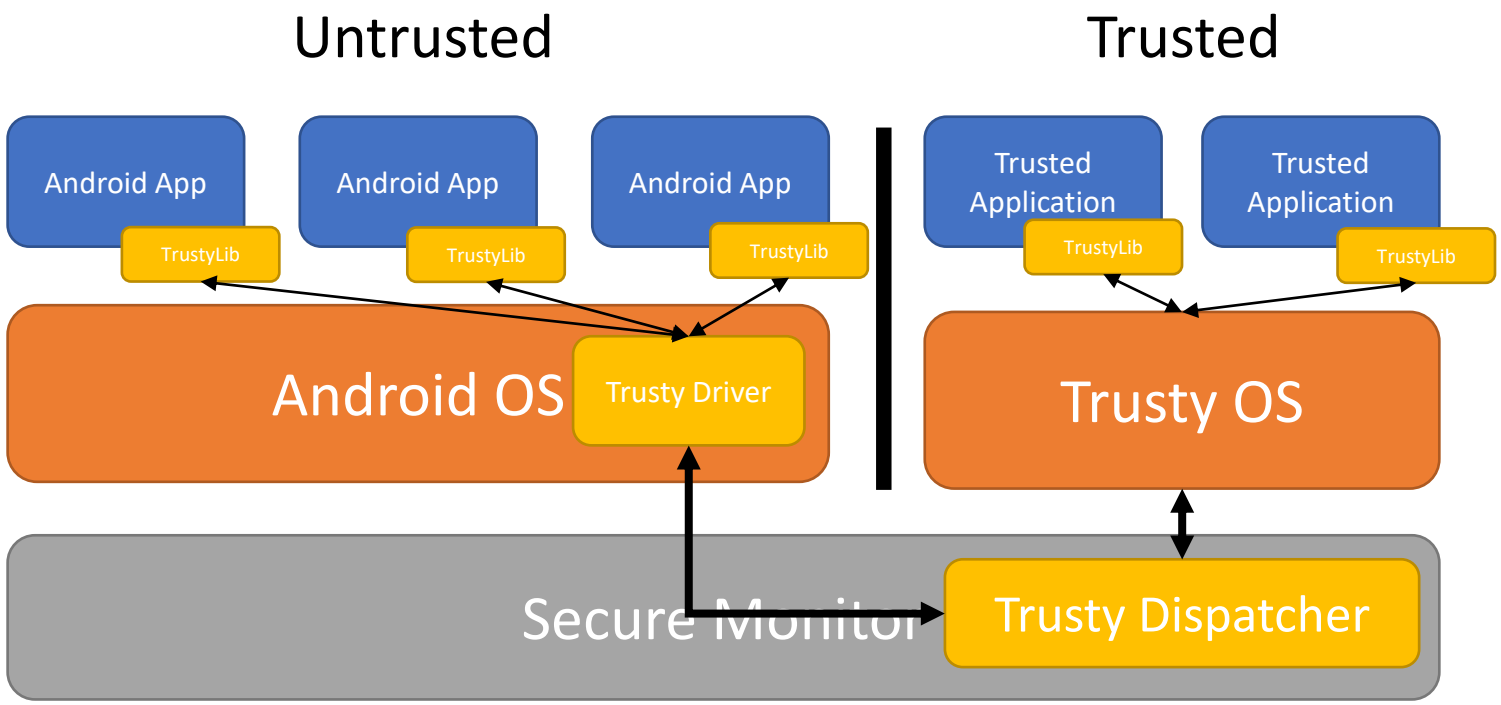


# ARM Trustzone

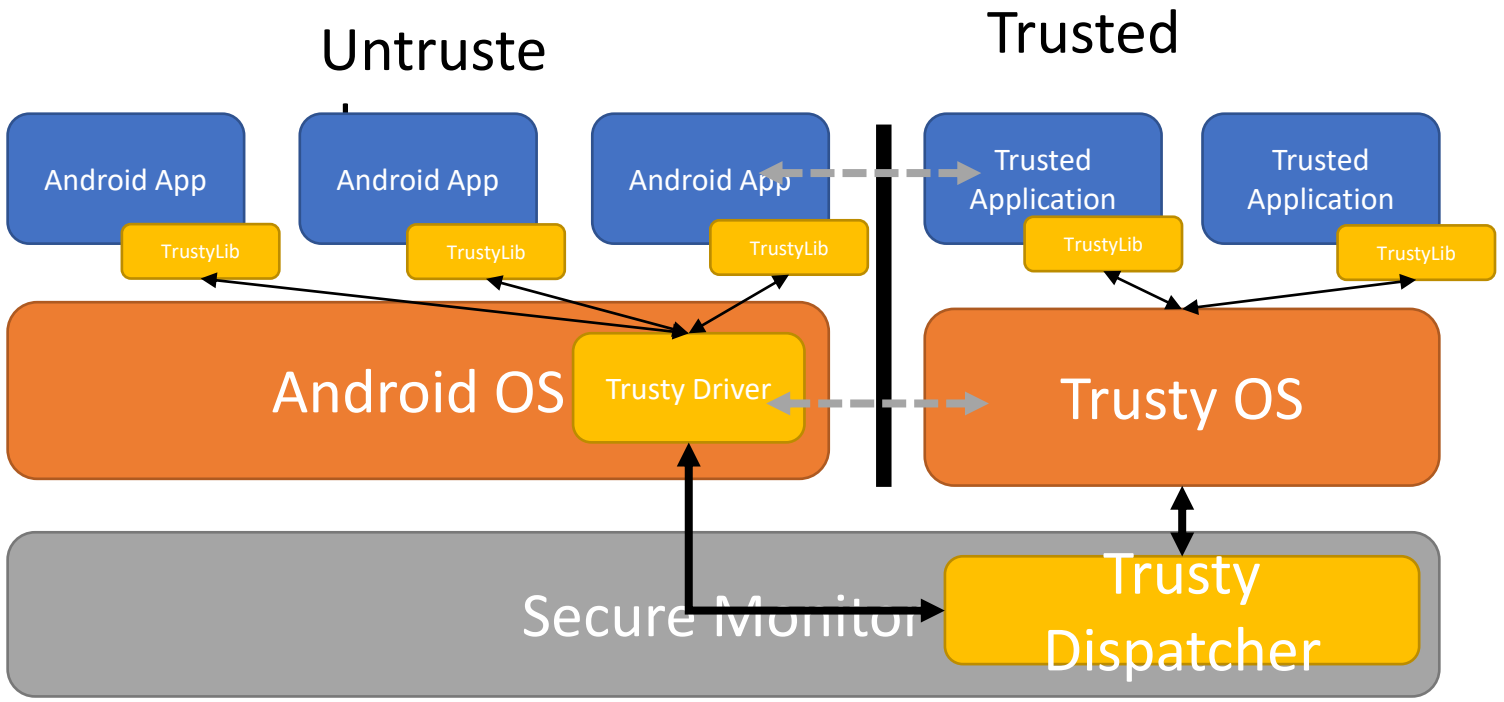



Own processor
Own RAM address
Own devices
etc…

# ARM Trustzone



# ARM Trustzone




Examples:

- Digital Right management
- Secure banking
- Multi-factor authentication
- etc…