Technische Universität Dresden

barkhausen institut

Trusted Computing

Advanced Operating Systems

Carsten Weinhold

Lecture Goals

Goal: Understand principles of:

- Authenticated booting, relation to (closed) secure booting
- Remote attestation
- Sealed memory
- Protection of applications from the OS
- Point to implementation variants (TPM, SGX, TrustZone)

Lecture Non-Goals

Non-Goal:

- Deep discussion of cryptography
- Lots of details on TPM, TCG, TrustZone, SGX, ...
  → Read the documents once needed

Key Goals of Trusted Computing

- Prevent certain software from running
- Know which computer system do I communicate with
- Know which stack of software is running …
    - … in front of me?
    - … on my server somewhere?
- Know the configuration of the software that is running
- Know that (how?) software and data are protected
- Restrict access to certain secrets to certain software
- Protect an application from the OS

Usage Examples (1)

Digital Rights Management (DRM):

- Vendor sells content
- Vendor creates key, encrypts content with it
- Client downloads encrypted content, stores it locally
- Vendor sends key, but only specific software shall be able to use it
- Has to work also when client is offline
- **Vendor does not trust the client**

Usage Examples (2)

Virtual machine by cloud provider:

- Client rents compute and storage (server / container / virtual machine)
- Client provides its own operating system (OS)
- Needs to ensure that provided OS is used
- Needs to ensure that provider cannot access data
- **Customer does not trust cloud provider**

Usage Examples (3)

Industrial Plant Control:

- Remote operator sends commands, receives sensor data
- Local technicians occasionally run maintenance / selftest software, install software updates, ...
- **Local technicians are not trusted**

Terminology

Trusted Computing …

Trusted Computing Base (TCB):

- Set of all components *(hardware, software, procedures)* that must be relied upon to enforce a security policy

Trusted Computing (Technology):

- Particular technology, often comprised of authenticated booting, remote attestation, and sealed memory

Trusted Computing Group (TCG):

- Consortium behind a specific trusted computing standard

Trusted Computing Terminology

Measuring:

- Process of obtaining metrics of platform characteristics
- Example: Hash code of (running) software

Attestation:

- Vouching for accuracy of (measured) information

Sealing:

- Binding information to a (measured) configuration

Notation: Hashes and Keys

Hash: H(M)

- Collision-resistant hash function H applied to content M

Asymmetric key pair: $\mathbf{E_{pair}}$ consisting of $\mathbf{E_{priv}}$ and $\mathbf{E_{pub}}$

- Asymmetric private/public key pair of entity **E**, used to either conceal (encrypt) or sign some content
- $\mathbf{E_{pub}}$ can be published, $\mathbf{E_{priv}}$ must be kept secret

Symmetric key: E

- Symmetric key of entity **E**, must be kept secret (“secret key”)

Notation: Result of Operations

Digital Signature: {M}E_{priv}

- **E_{priv}** signed message M
- **E_{pub}** can be used to verify that **E** has signed **M**
- **E_{pub}** is needed and sufficient to check signature

Concealed Message: {M}E_{pub}

- Message **M** concealed (encrypted) for **E**
- **E_{priv}** is needed to unconceal (decrypt) **M**

Identification of Software

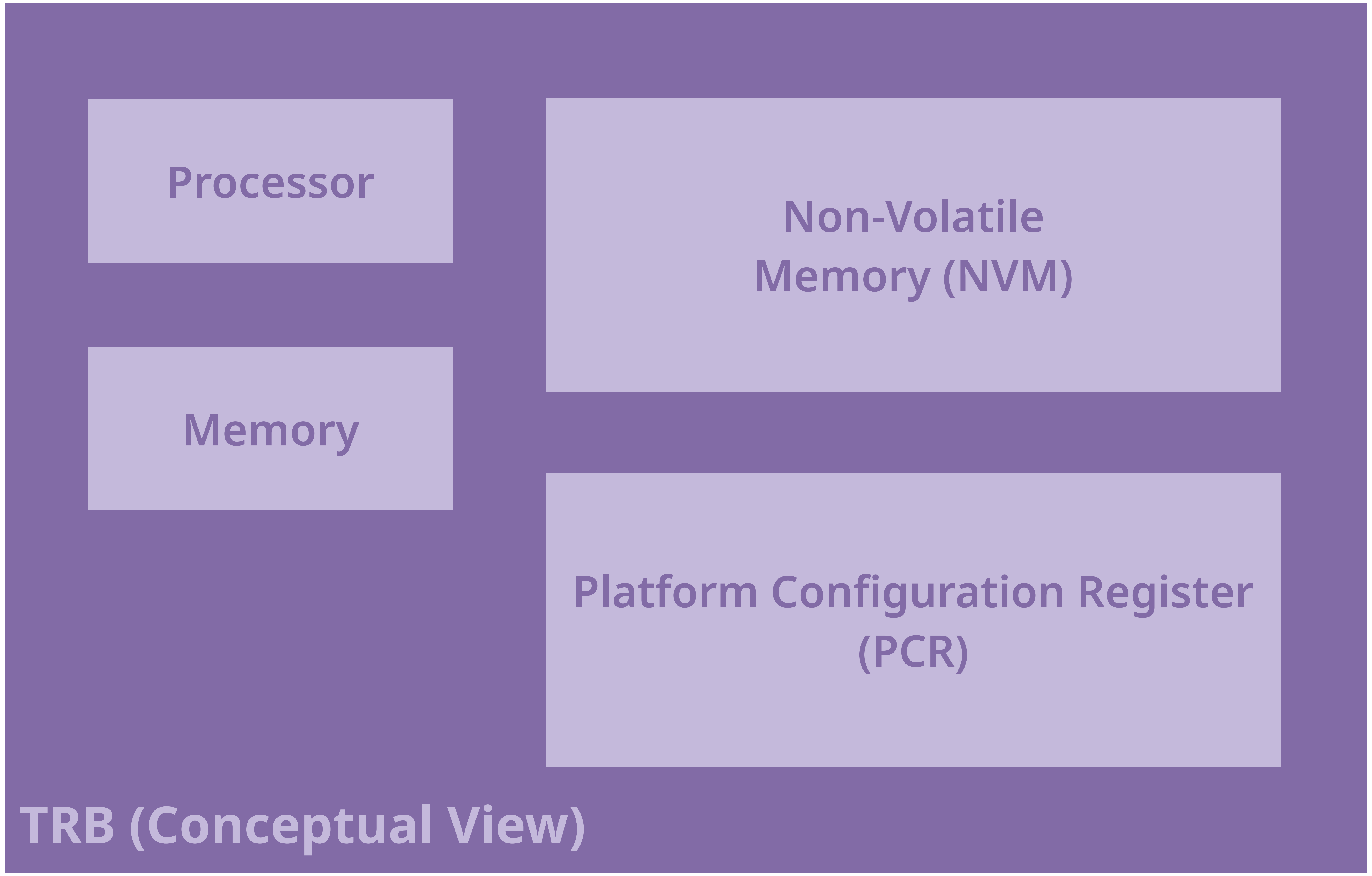
Example: program vendor FooSoft (FS)

Software identity **ID** must be known

Several ways to identify software:

- By hash: $\mathbf{ID_{Program} = H(Program)}$
- By signature: $\mathbf{\{Program, ID_{Program}\}FS_{priv}}$
  - Use $\mathbf{FS_{pub}}$ to check signature
  - $\mathbf{(H(Program), FS_{pub})}$ can serve as $\mathbf{ID_{Program}}$
- By certificate: signature includes metadata (e.g., name, vendor, version, …)

Tamper-Resistant Black Box (TRB)



Measuring + Attesting

Secure Booting ("There can be only one OS")

OS stored in read-only memory (flash)

Hash H(OS) in TRB NVM, preset by manufacturer:

1. Load OS code
2. Compare **H(loaded OS code)** to preset **H(OS)**
3. Abort if hash is different

Public key FS_{pub} in TRB NVM, preset by manufacturer:

1. Load OS code
2. Check signature of loaded OS code using $\mathbf{FS_{pub}}$
3. Abort if signature check fails

Authenticated Booting ("Choose your OS")

Steps:

1. Preparation by OS and TRB vendors
2. Booting & measuring
3. Remote attestation

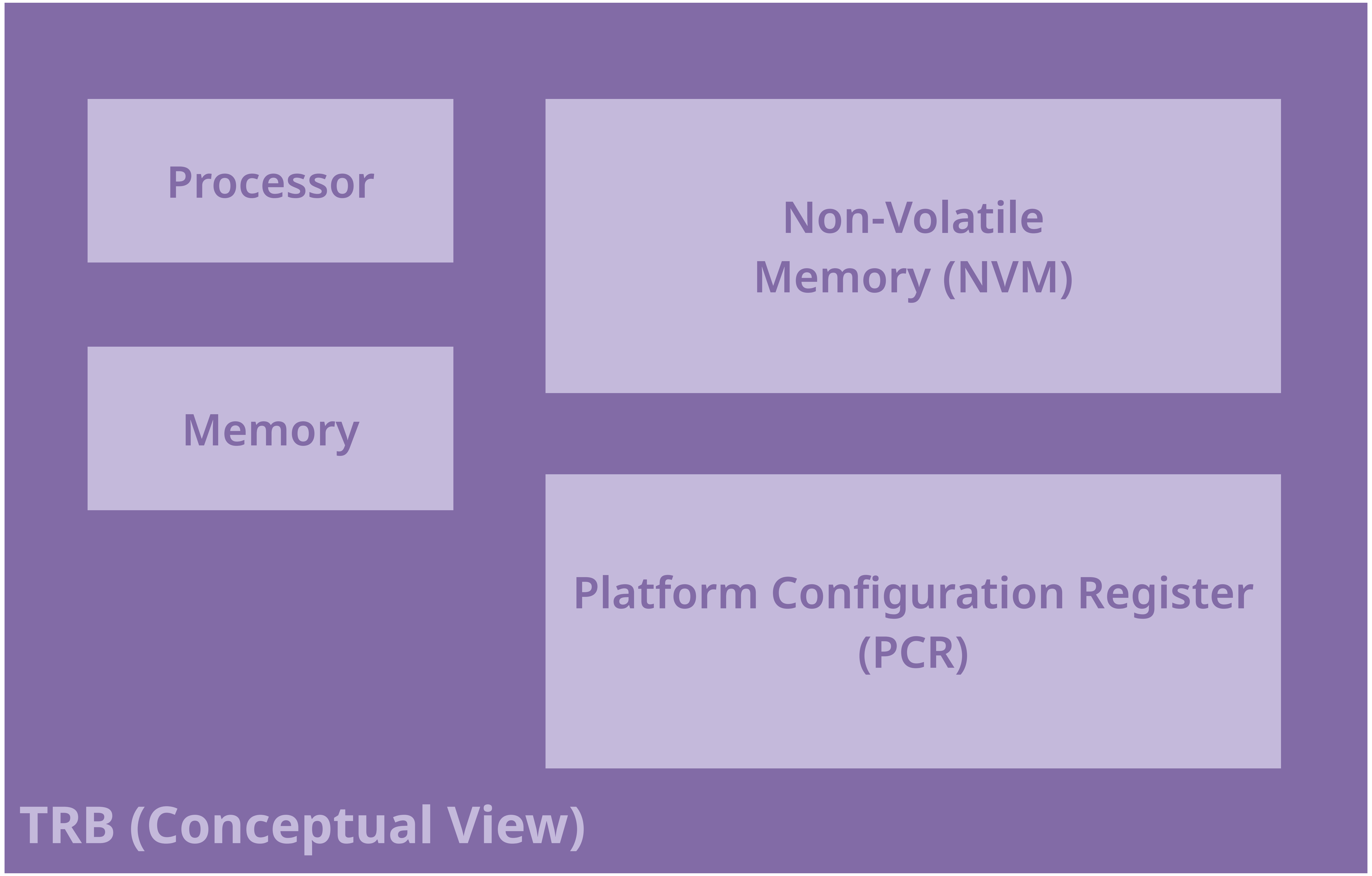
OS Identity and Certification

1a) Preparation by OS vendor:

- Certifies: **{**„a valid OS“**, H(OS)}OSVendor$_{\mathbf{priv}}$**
- Publishes identifiers: **OSVendor$_{\mathbf{pub}}$** and **H(OS)**

- Has to be done <u>once</u> for each OS release / patch

Tamper-Resistant Black Box (TRB)

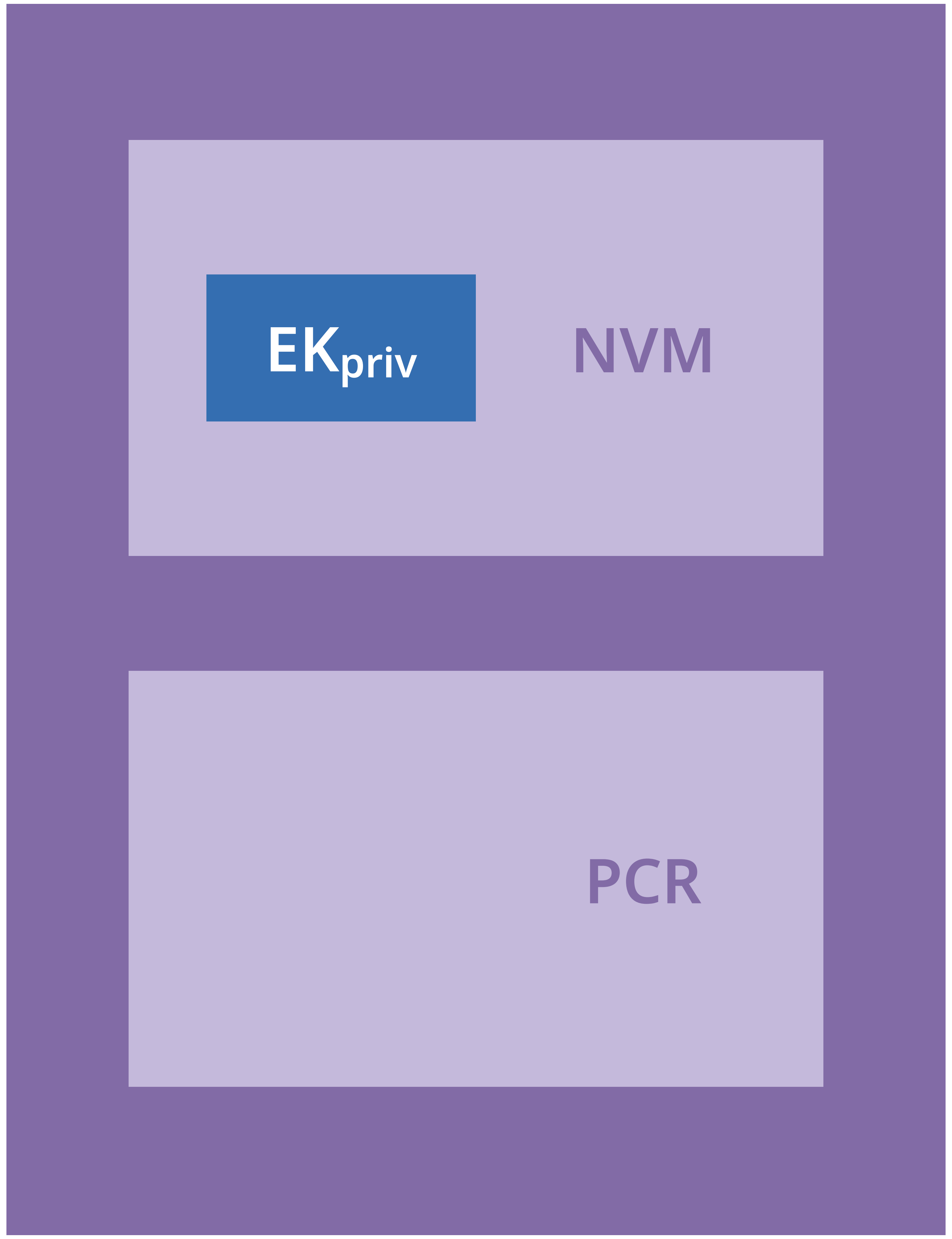


TRB Initialization

1b) Preparation by TRB vendor:

- TRB generates "Endorsement Key" pair: $\mathbf{EK_{pair}}$
- TRB Stores $\mathbf{EK_{priv}}$ in TRB NVM
- TRB publishes $\mathbf{EK_{pub}}$
- TRB vendor certifies:
  {"a valid EK", $\mathbf{EK_{pub}}$**}**$\mathbf{TRBVendor_{priv}}$

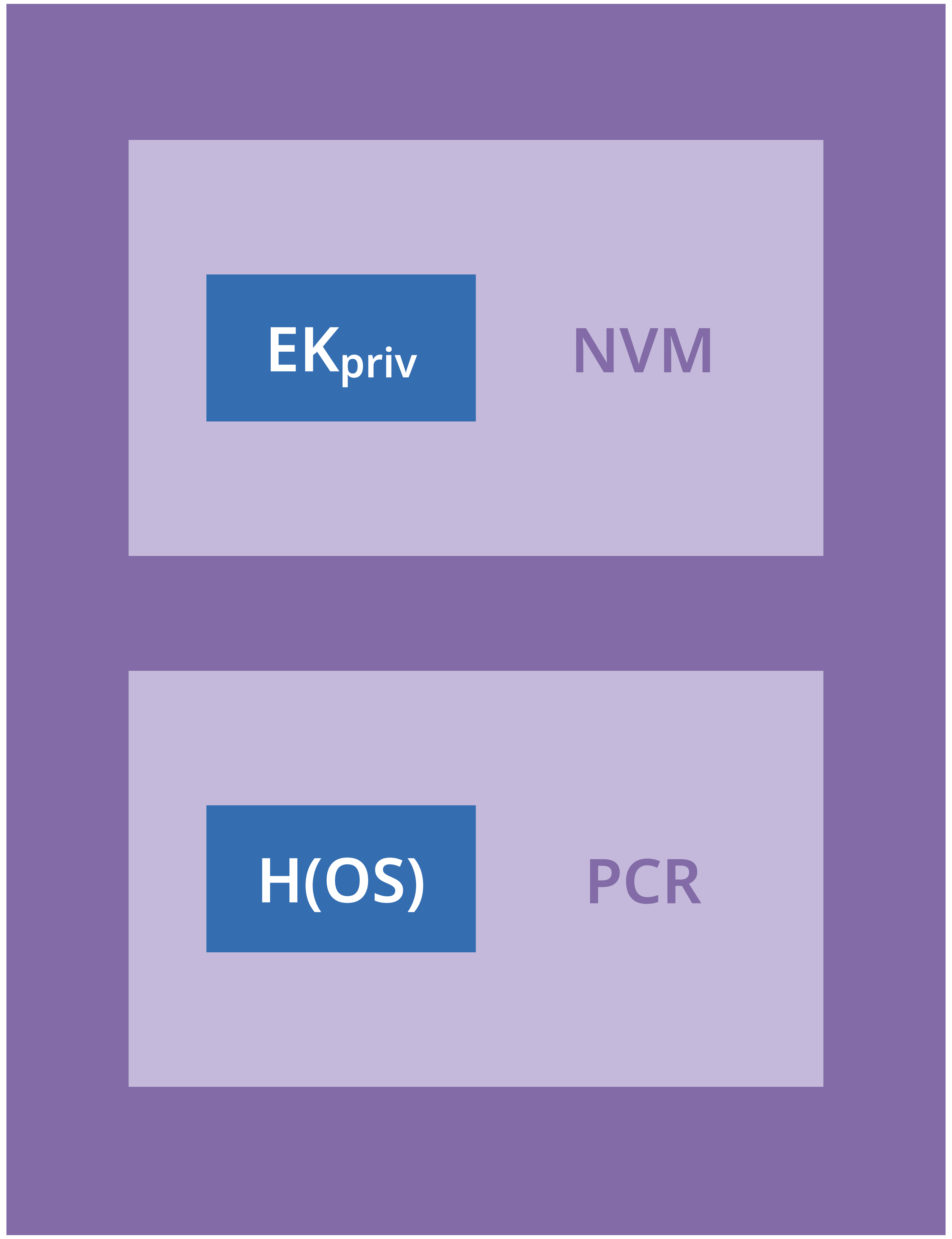
- Has to be done <u>once</u> at manufacturing time



Authenticated Booting

2) Booting & measuring:

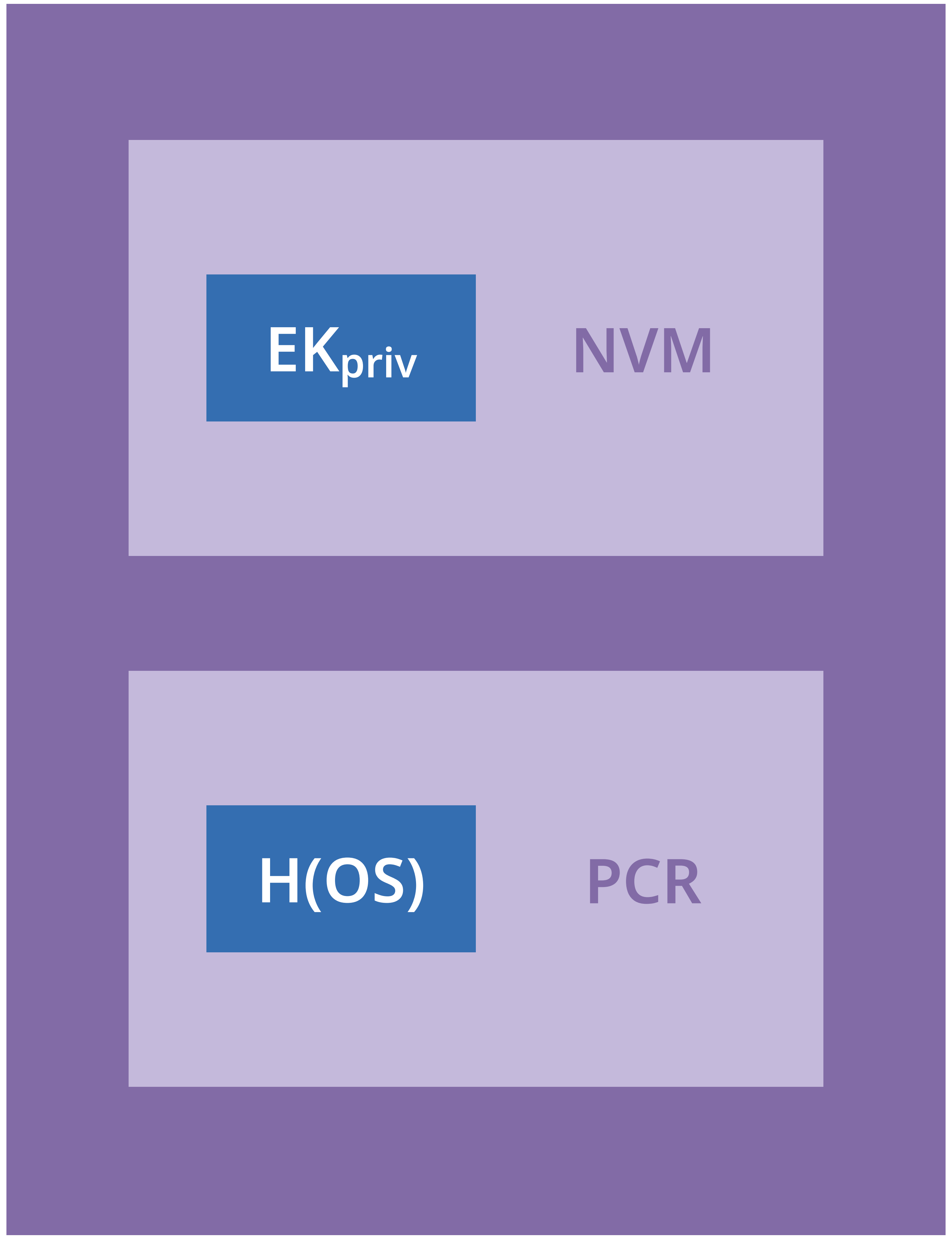
- TRB resets
- TRB computes ("measures") hash **H(OS)** of loaded OS
- Records **H(OS)** in platform configuration register **PCR**
- TRB starts OS
- **Note: PCR** not directly writable
- Has to be done at <u>every</u> boot



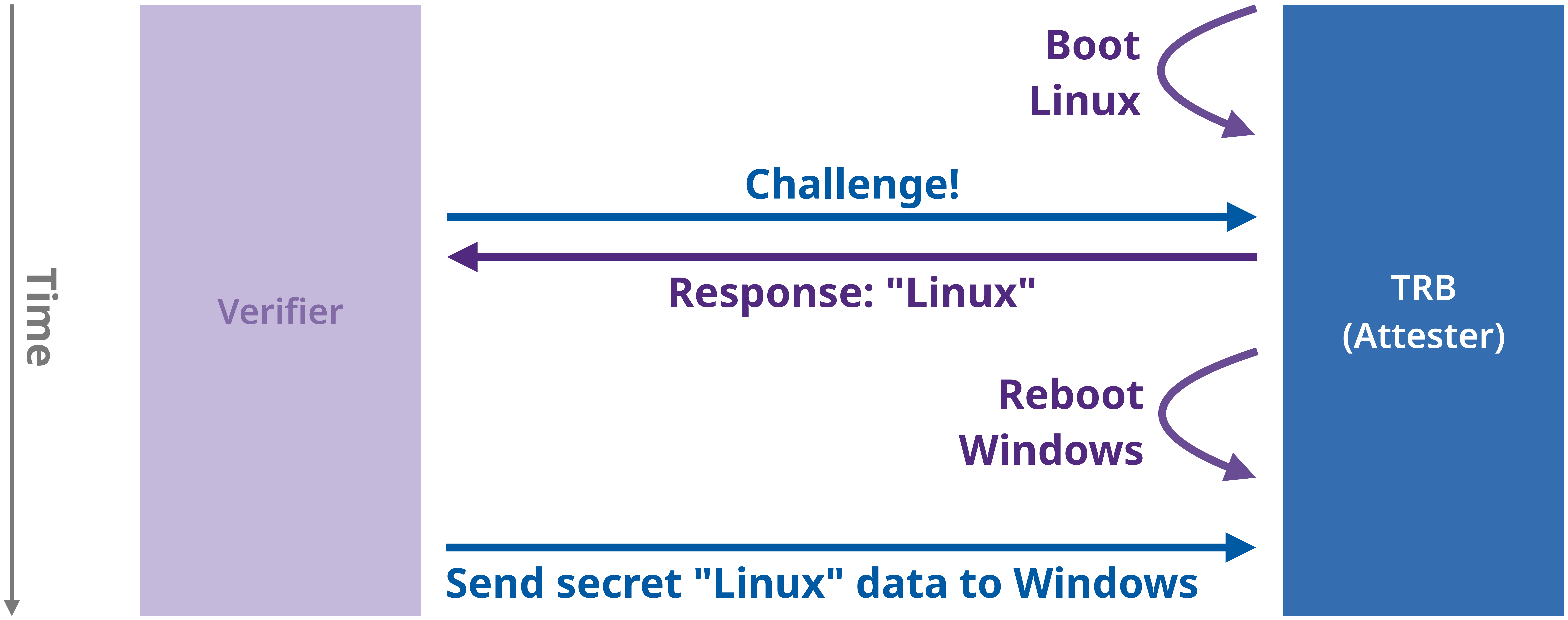
Remote Attestation (Simplified)

3) Remote Attestation:

- Remote computer ("verifier") sends "challenge": **NONCE**
- TRB signs **{NONCE, PCR}EK$_{priv}$** (**evidence** or **attestation report**) and sends it to verifier
- Verifier evaluates evidence: checks signature, decides if OS identified by **H(OS)** in **PCR** is OK
- Can be done many times while attested system is active



Problem



Problem: Time-of-check, time-of-use (TOCTOU) attack possible

Solution: Create new key pair for protecting data until next reboot

Booting (Considering Reboot)

At each boot, TRB does the following:

- Computes **H(OS)** and records it in **PCR**
- Creates two key pairs for the booted, currently active OS:
  - **$\text{ActiveOSAuthK}_{\text{pair}}$** /* for authentication (signing) */
  - **$\text{ActiveOSConK}_{\text{pair}}$** /* for concealing (encryption) */
- TRB certifies:
  $\{\text{ActiveOSAuthK}_{\text{pub}}, \text{ActiveOSConK}_{\text{pub}}, \text{H(OS)}\}\text{EK}_{\text{priv}}$
- Hands over to booted OS
- OS uses keys as "session keys”

Attestation (Considering Reboot)

Remote Attestation:

- Challenger sends: **NONCE**
- Currently booted, active OS generates response:
  {ActiveOSConK$_{pub}$, ActiveOSAuthK$_{pub}$, H(OS)}EK$_{priv}$
  {NONCE}ActiveOSAuthK$_{priv}$

Client sends data over secure channel:

- **{**data for active OS**}ActiveOSConK$_{pub}$**

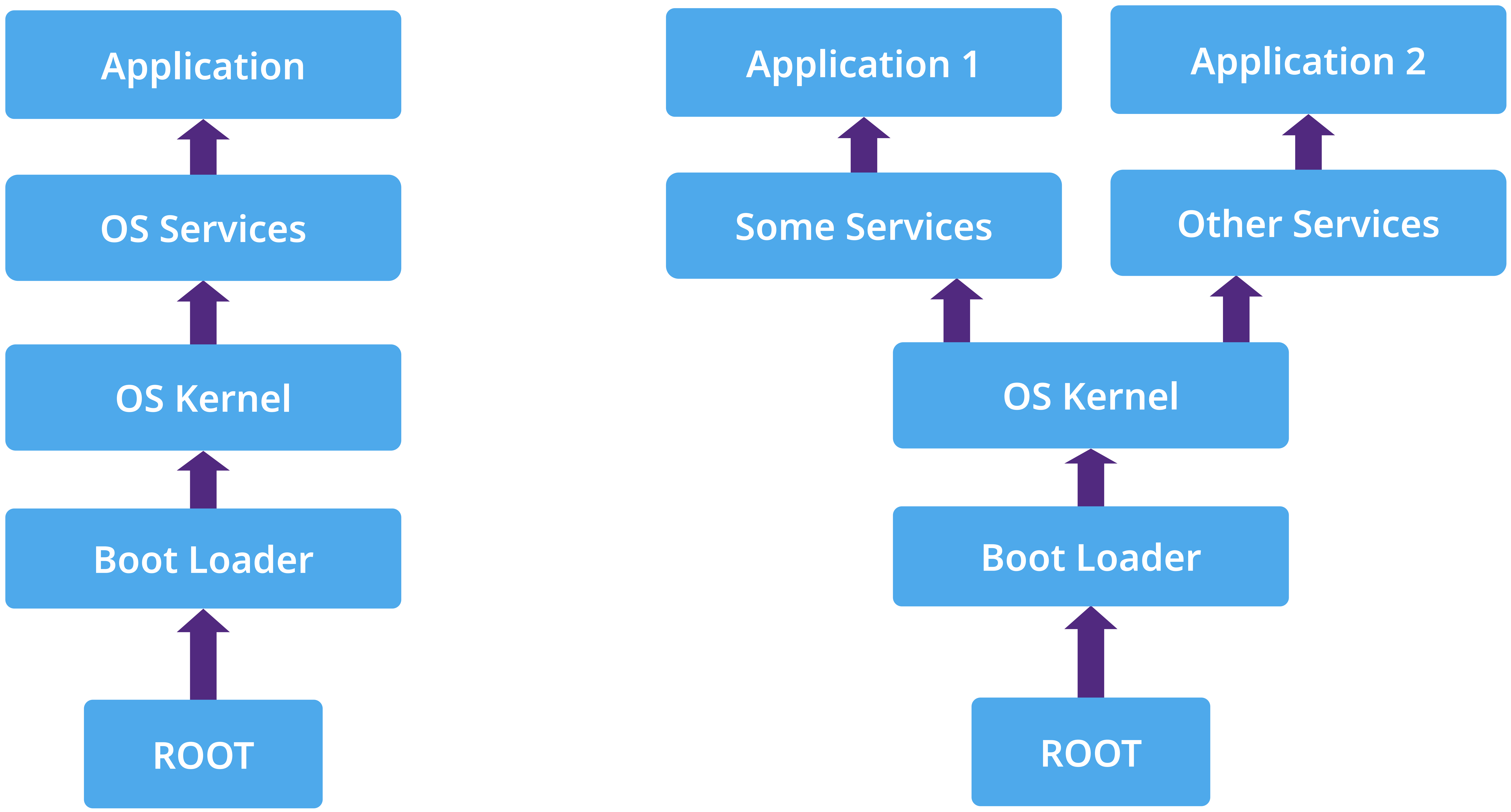
Assumptions

Authenticated booting and **remote attestation** as presented are secure, if:

1) TRB can protect **EK_{priv}, PCR,** running OS
2) OS can protect **"Active OS"** keys
3) Rebooting destroys content of:
    - **PCR**
    - **"Active OS"** keys in memory

Measurement Granularity

Software Stacks and Trees



Software Stacks and Trees

Two Concerns:

- Remote attestation of one process (leaf in tree)
- Very large Trusted Computing Base (TCB) for booting
  → Microkernel-based OS lecture

Software Stacks and Trees

Two Concerns:

- Remote attestation of one process (leaf in tree)
- Very large Trusted Computing Base (TCB) for booting
  → Microkernel-based OS lecture

Software Stacks and Trees

Key pairs per level of tree:

- OS controls applications → generate additional key pair per application
- OS certifies:
  - **{Application 1, App1K_{pub}}ActiveOSAuth$_{priv}$**
  - **{Application 2, App2K_{pub}}ActiveOSAuth$_{priv}$**

Sealing

The Need for Trusted Storage

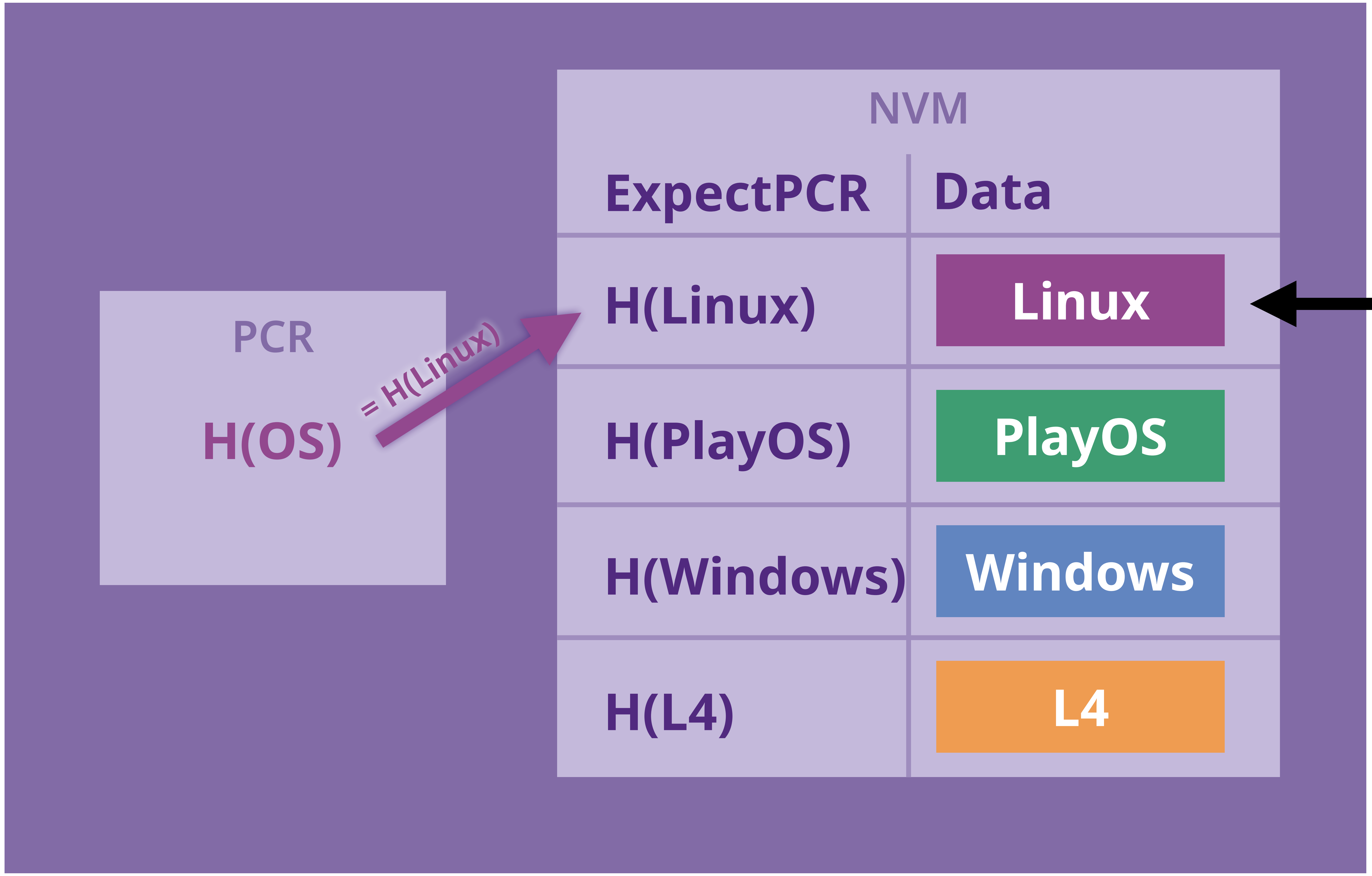
Use case from earlier example:

- Send data over secure channel after remote attestation
- Bind that data to software configuration via TRB

Problem: How to work with this data when offline?

- Must store data for time after reboot
- For example for DRM: bind decryption key for downloaded movie to specific machine with specific OS (and video player)

Sealed Memory Principle

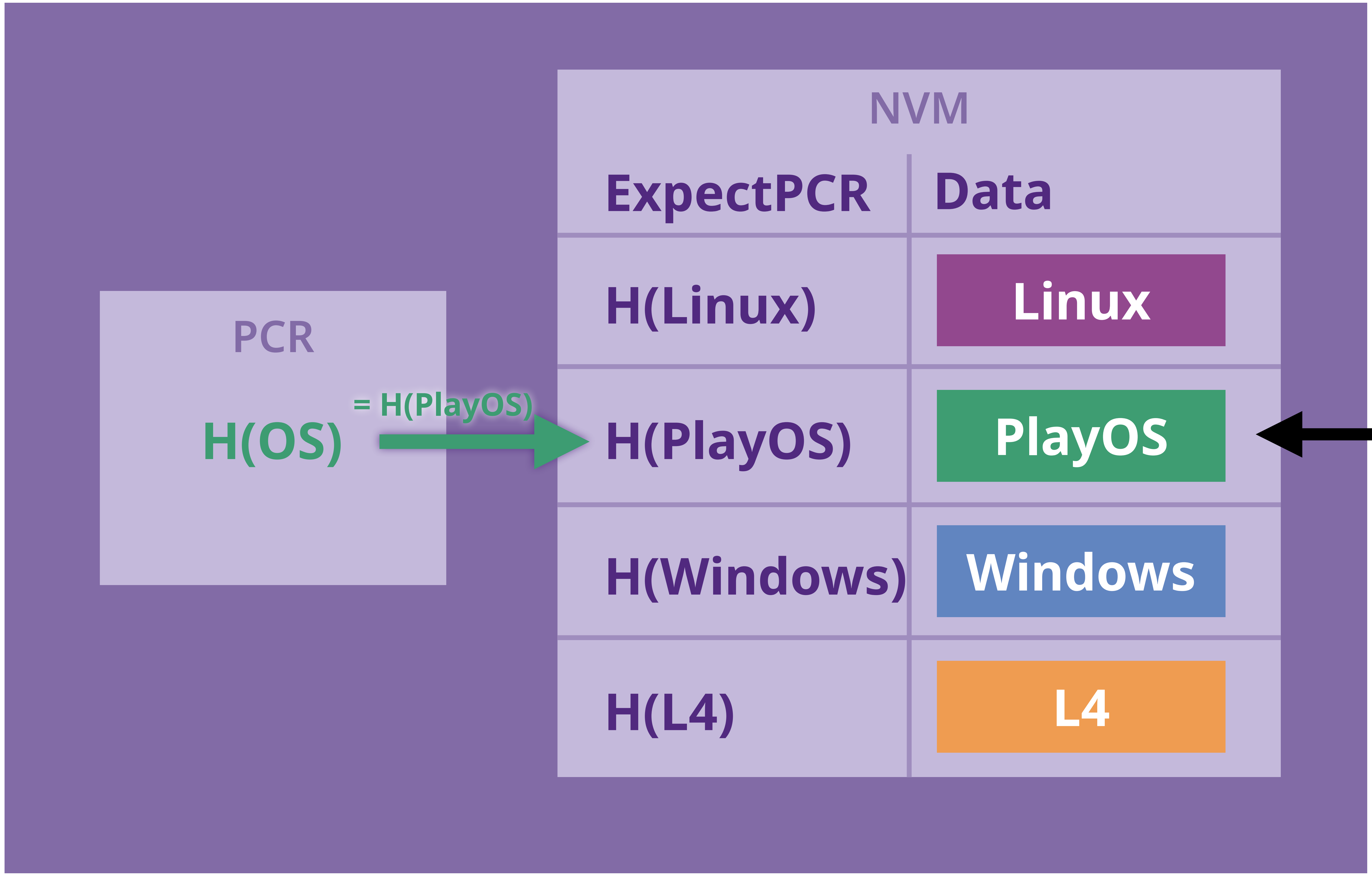


TRB (but not OS) can add / remove / read / write "Sealed Memory" slots

Can be accessed by currently active Linux OS

Other slots inaccessible due to PCR mismatch

Sealed Memory Principle

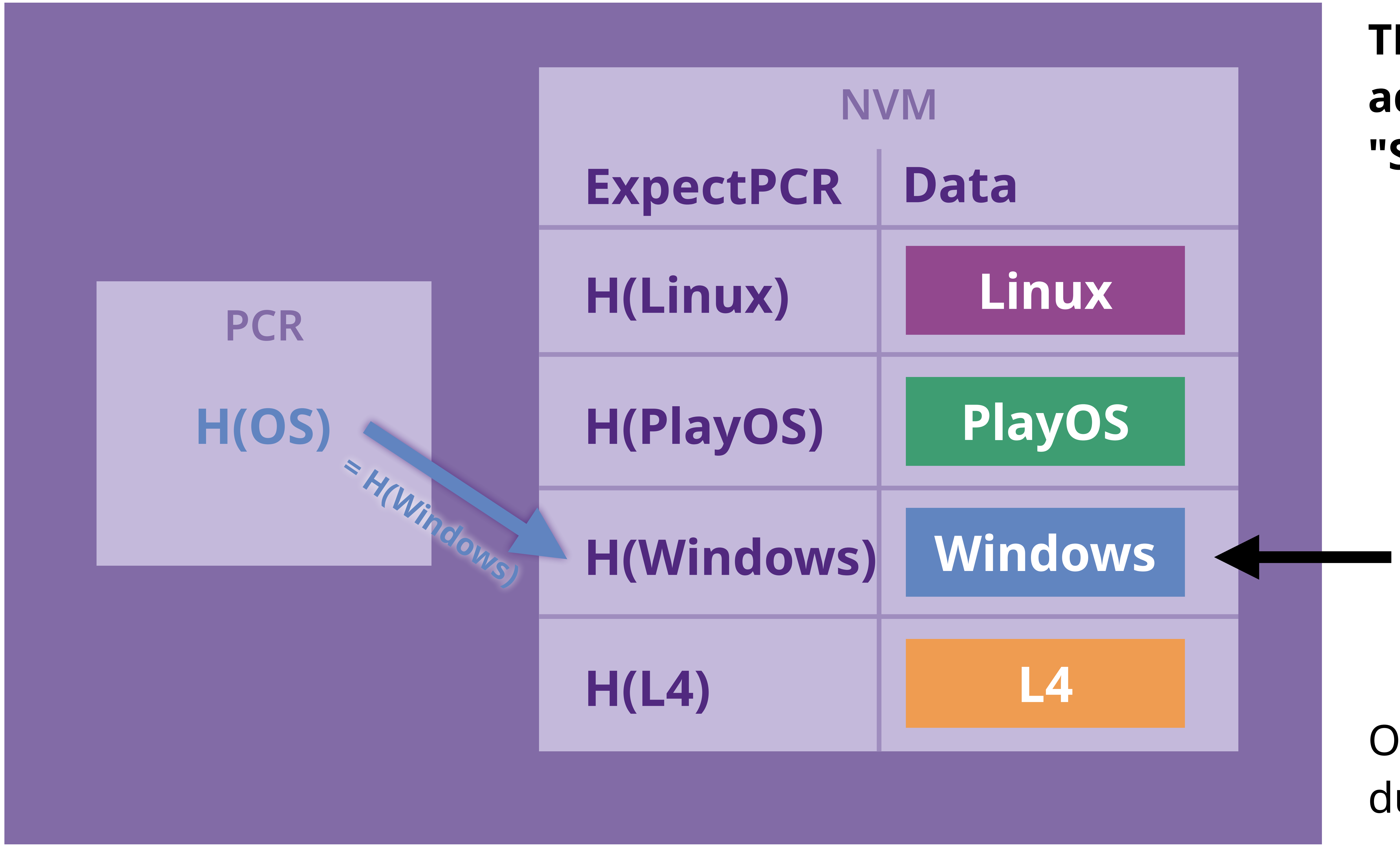


TRB (but not OS) can add / remove / read / write "Sealed Memory" slots

Can be accessed by currently active PlayOS

Other slots inaccessible due to PCR mismatch

Sealed Memory Principle

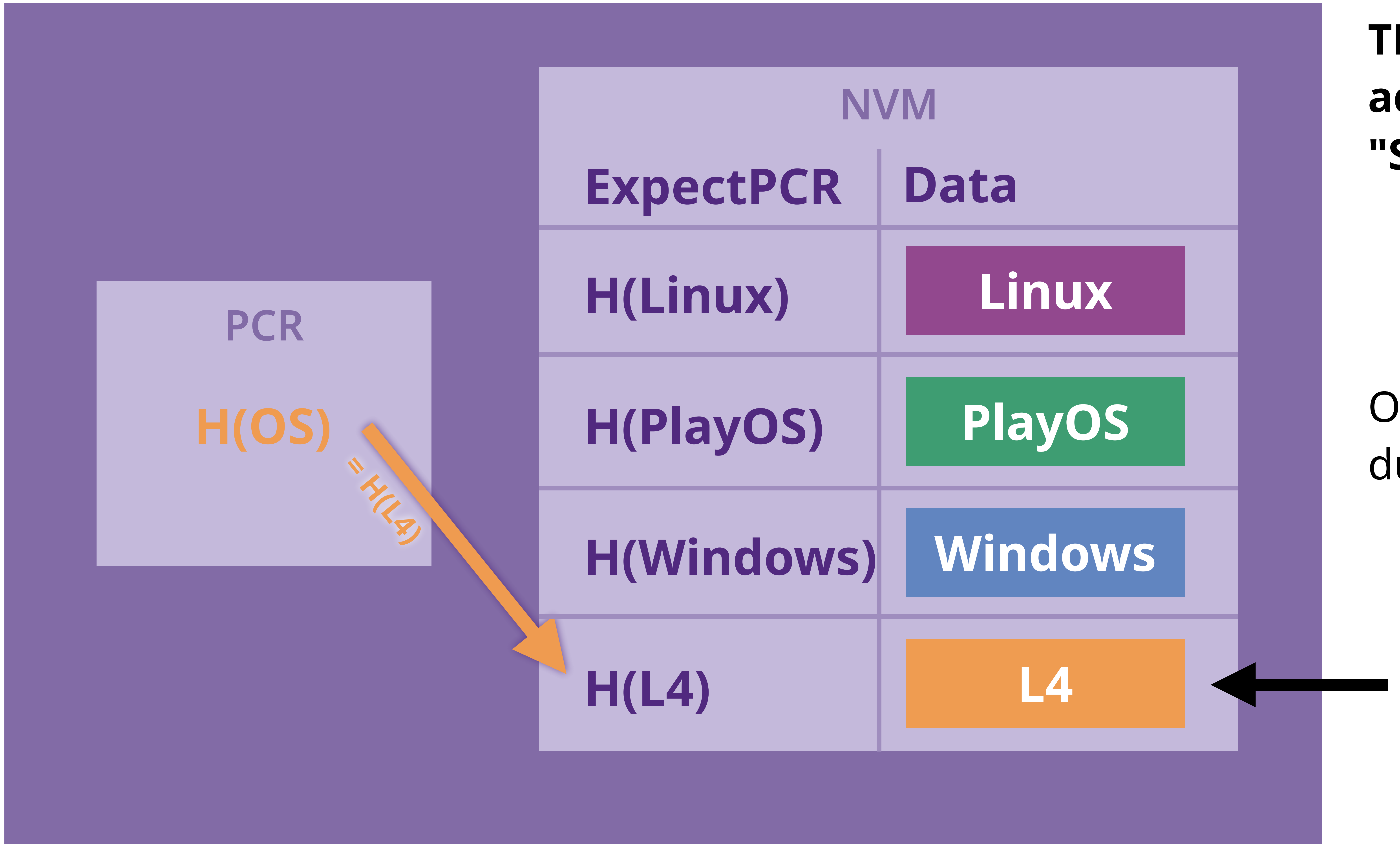


TRB (but not OS) can add / remove / read / write "Sealed Memory" slots

Can be accessed by currently active Windows OS

Other slots inaccessible due to PCR mismatch

Sealed Memory Principle



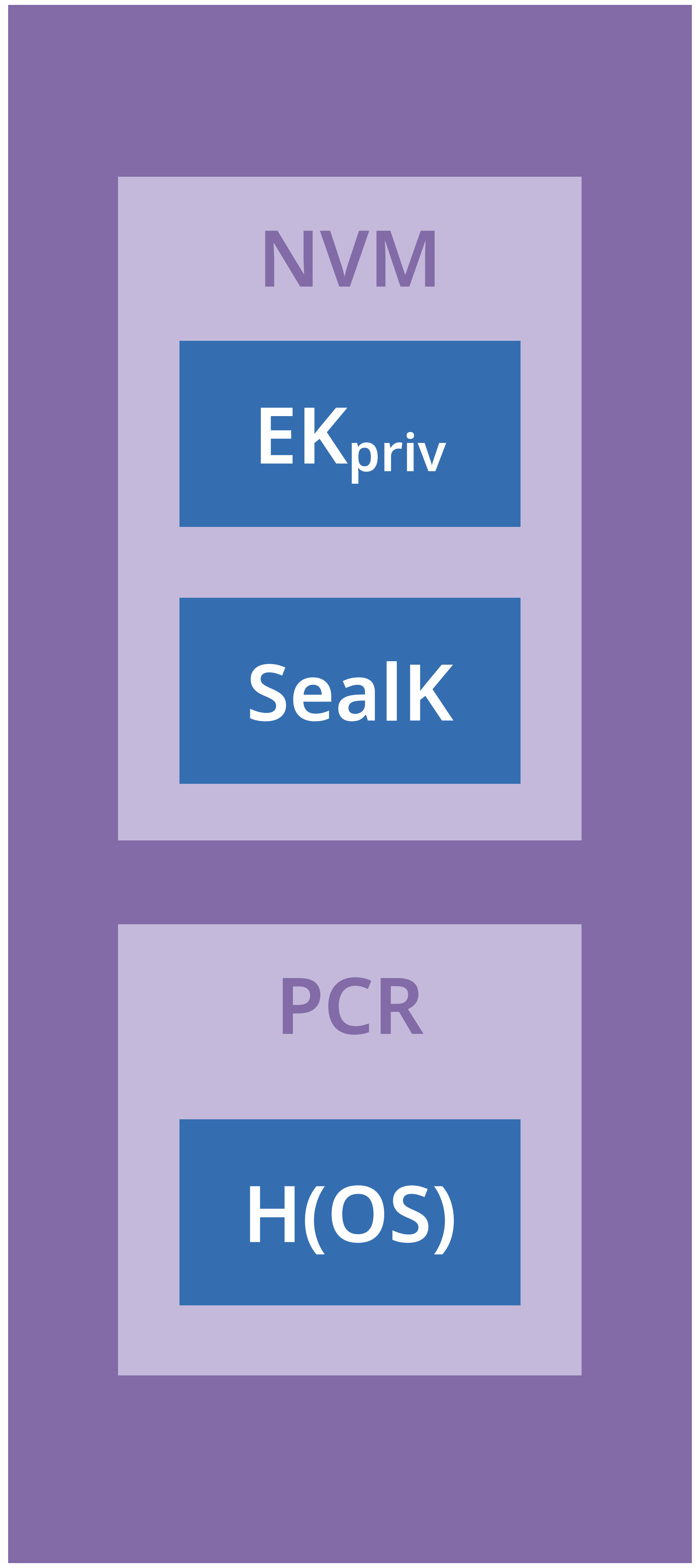
TRB (but not OS) can add / remove / read / write "Sealed Memory" slots

Other slots inaccessible due to PCR mismatch

Can be accessed by currently active L4 OS

Sealed Memory Implementation

- TRB creates secret symmetric key **SealK**
- TRB encrypts (**Seal**) and decrypts (**Unseal**) data using **SealK**
- **Seal(ExpectPCR, data)**
  → {ExpectPCR, data}SealK
- **Unseal({ExpectPCR, data}SealK) → data**
  iff current **PCR** == **ExpectPCR**
  else abort without releasing data



Sealed Memory Flexibility

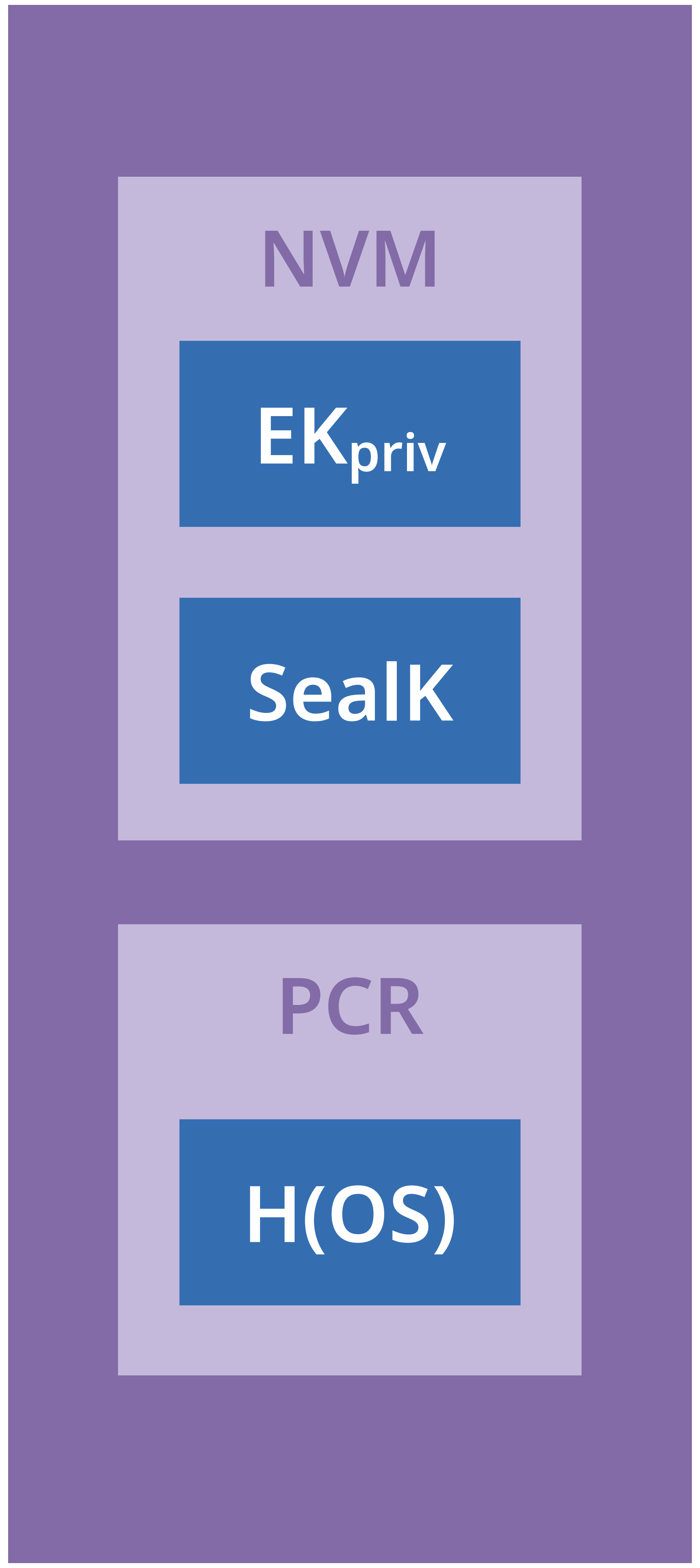
- Sealed (encrypted) data can be stored outside of TRB, so NVM capacity can be small
- When sealing, arbitrary "expected PCR" values can be specified (e.g., future version of OS, or entirely different OS)

{H(Linux), Linux }SealK

{H(Windows), Windows }SealK

{H(PlayOS), PlayOS }SealK

{H(L4), L4 }SealK



Examples

- Windows: **Seal (H(PlayOS), PlayOS_Secret)**
  → **sealed_message** (store it on disk)
- L4: **Unseal (sealed_message)**
  → PlayOS, PlayOS_Secret
  → ExpectPCR != PlayOS
  → **abort**
- PlayOS: **Unseal(sealed_message)**
  → PlayOS, PlayOS_Secret
  → ExpectPCR == PlayOS
  → **emit PlayOS_Secret**

Implementations

Tamper Resistant Black Box?

Ideally: includes CPU, Memory, …

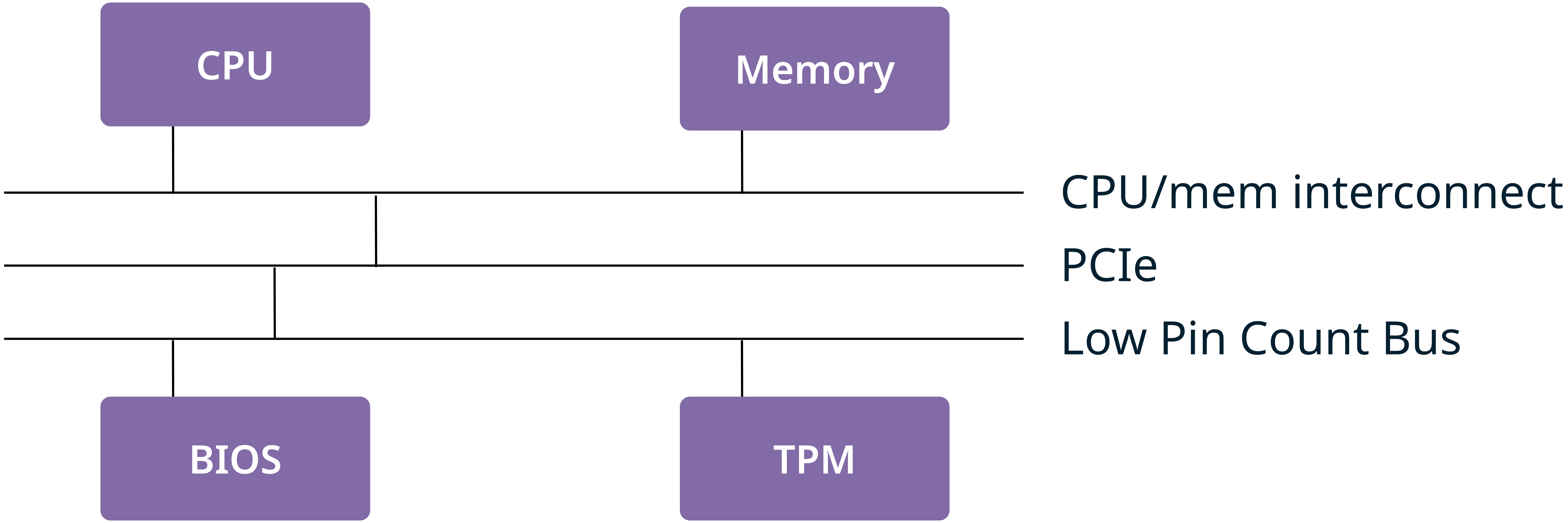
In practice:

- Additional physical protection (e.g., IBM 4758, → Wikipedia)
- Hardware extensions:
  - Trusted Platform Module (TPM):
    - No additional isolation
    - Requires careful design to allow firmware updates, etc.
  - Isolation via new privilege mode: Intel SGX, Arm TrustZone,…
  - Isolation via encrypted VMs: Intel TDX, AMD SEV, Arm CCA, …

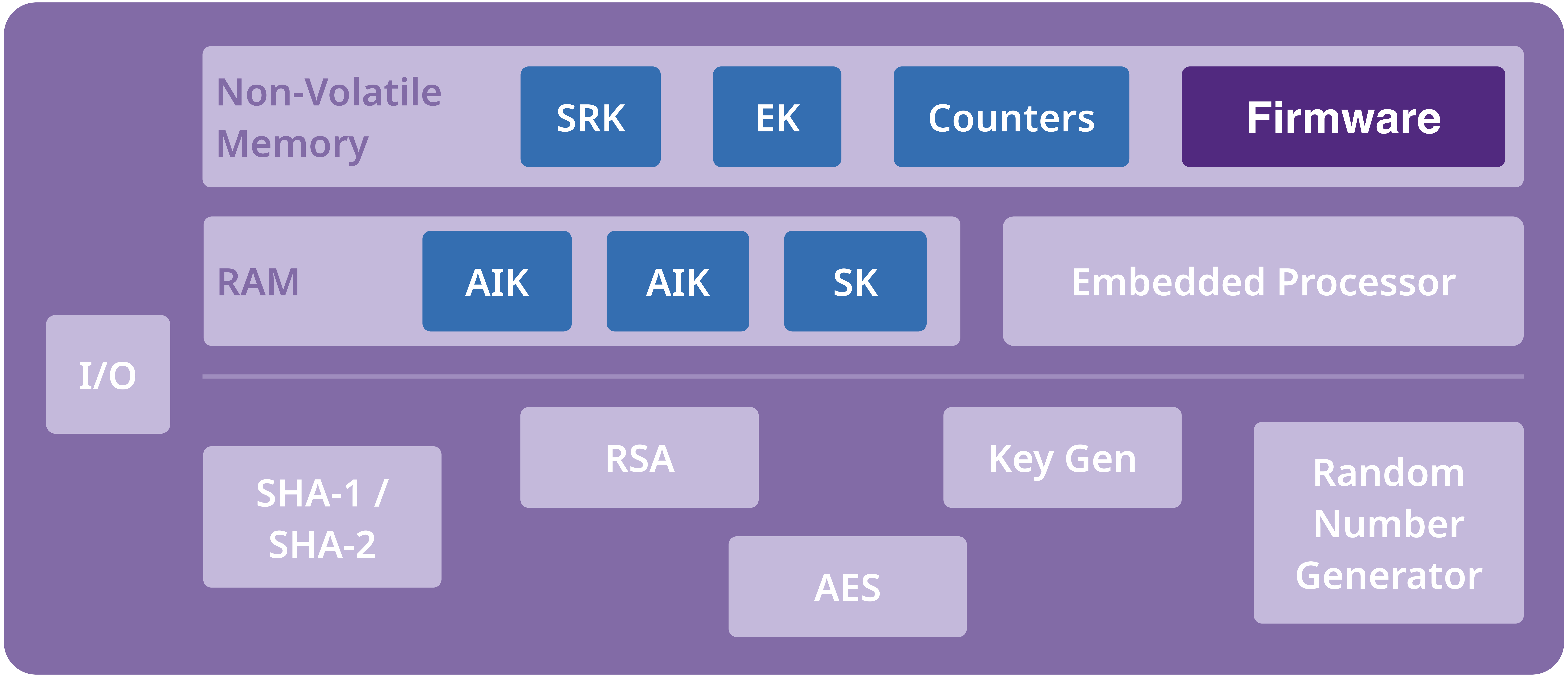
Trusted Execution Environment (TEE)

Confidential Computing

TCG PC Platform: Trusted Platform Module (TPM)



Trusted Platform Module



Protection of Application

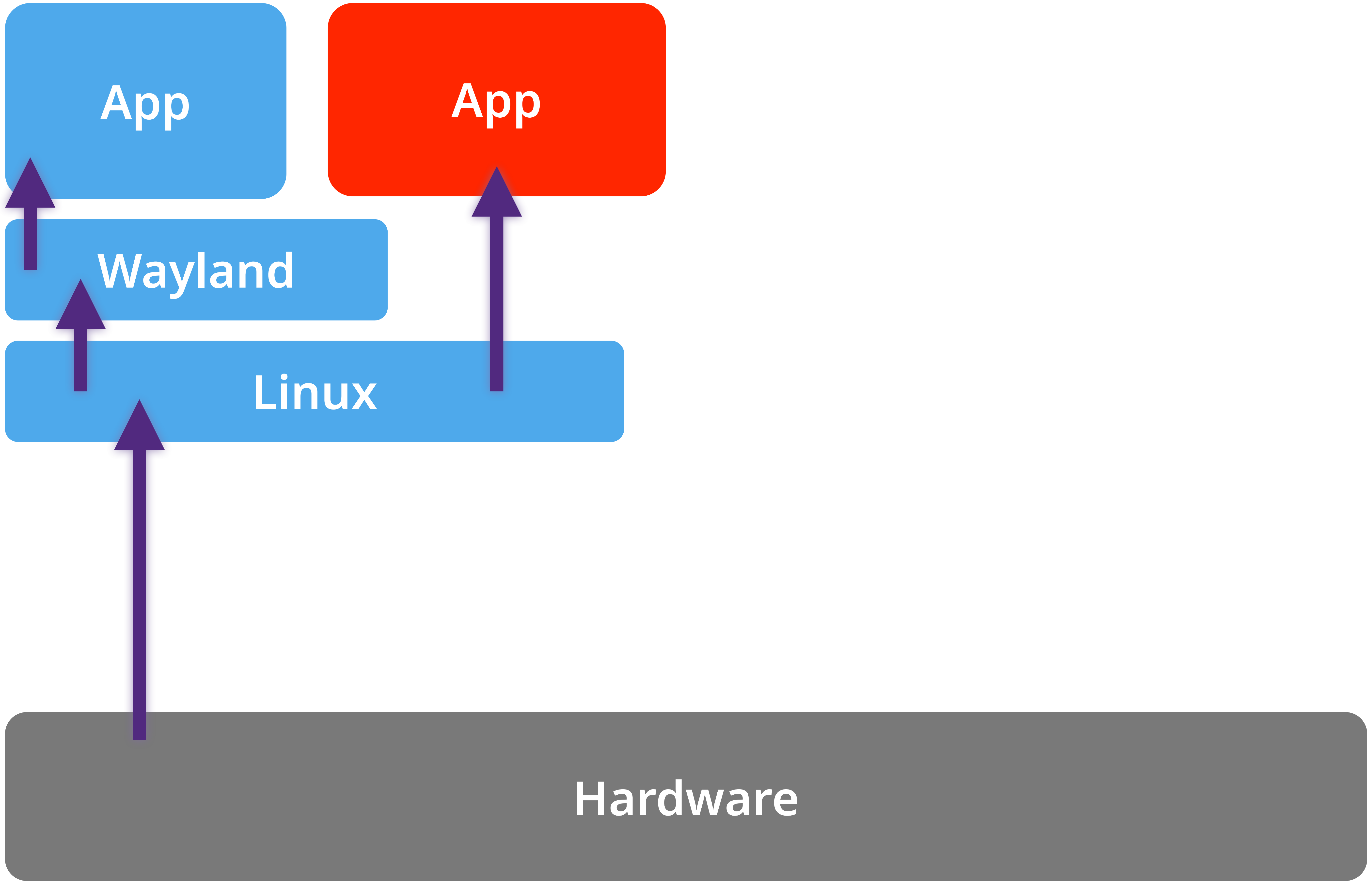
Principle Method:

- Isolate critical software
- Rely on small Trusted Computing Base (TCB)

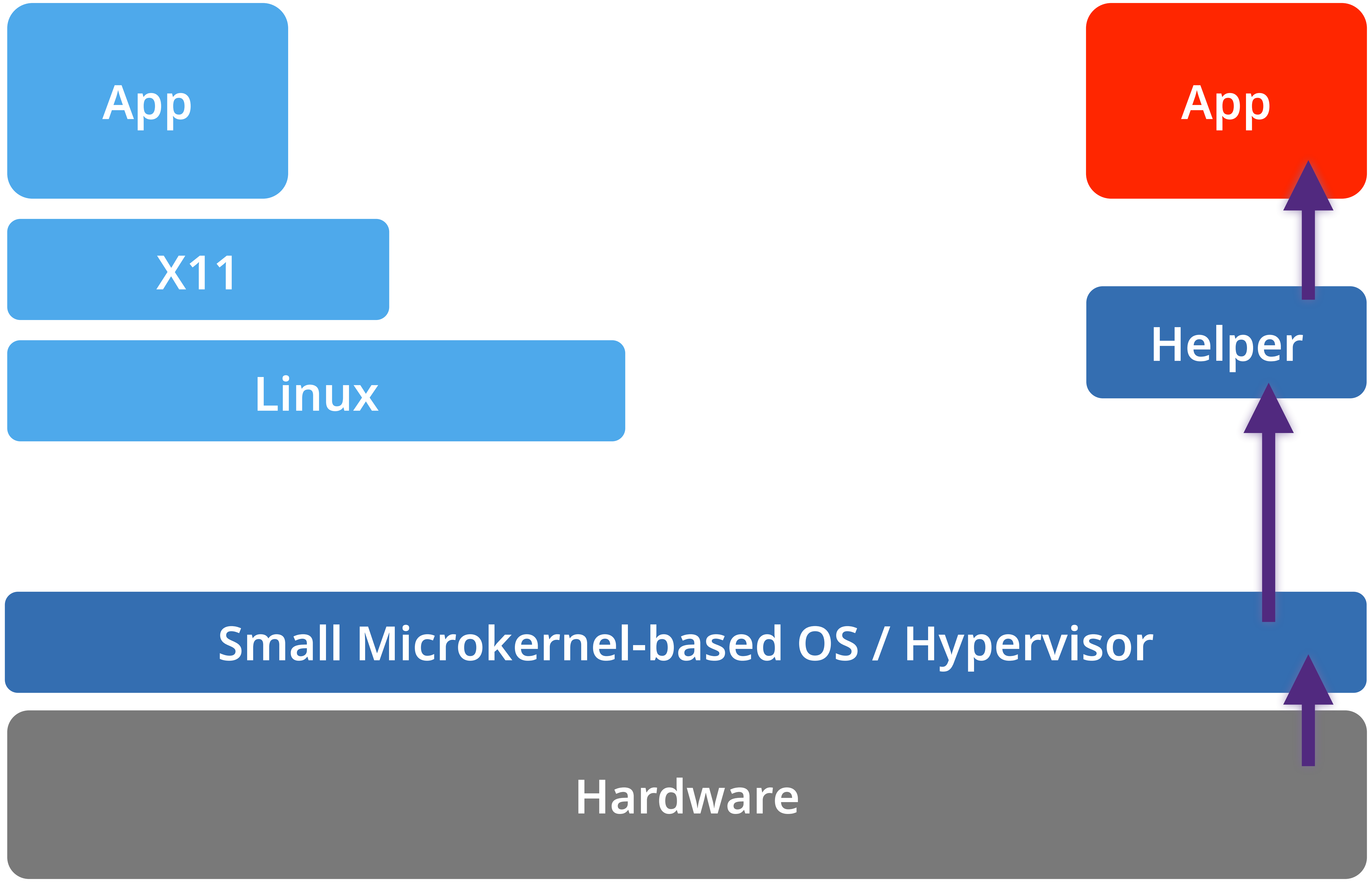
Ways to implement the method:

- Small OS kernels:
  microkernels, separation kernels, …
- Hardware / microcode support

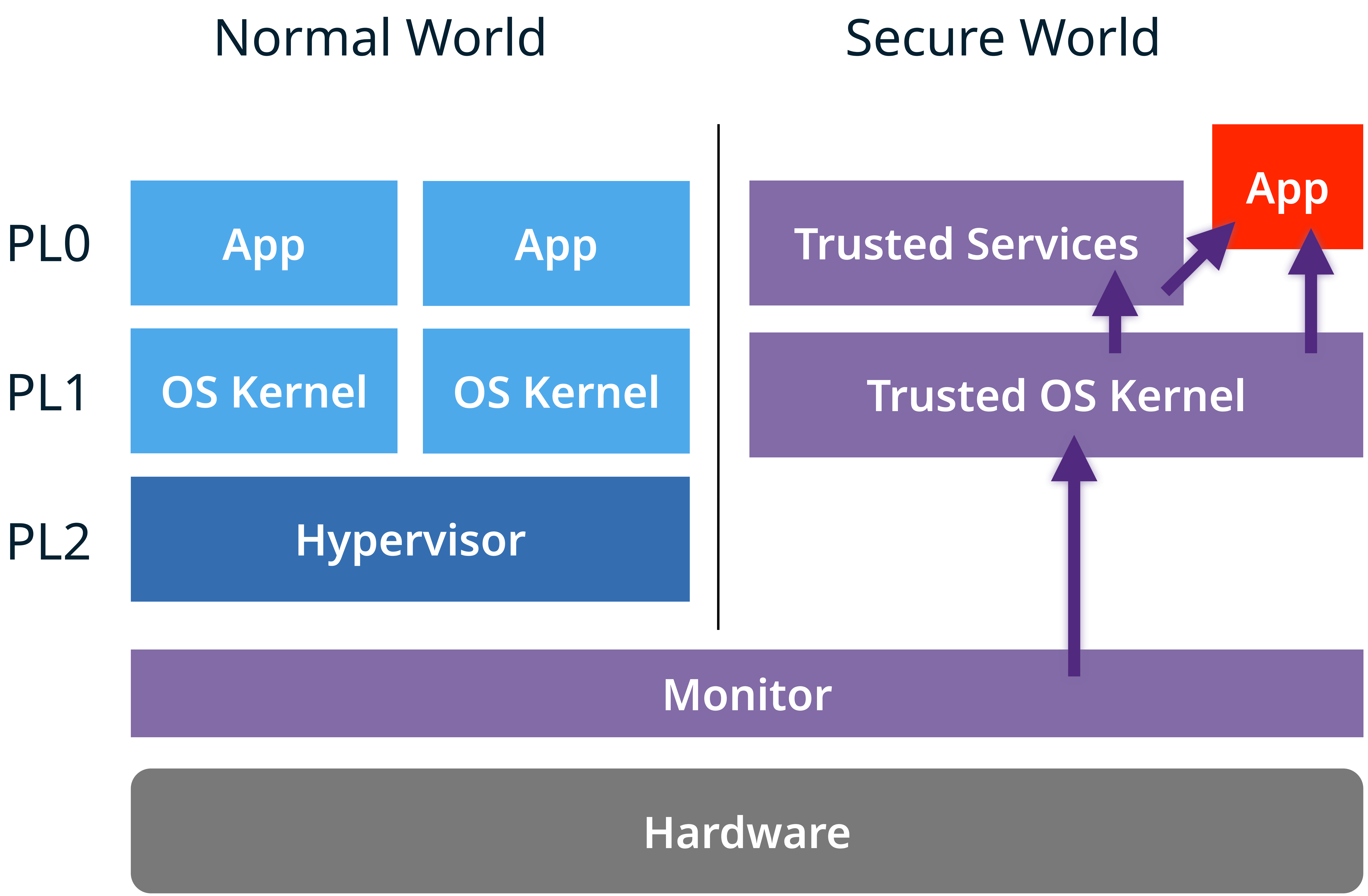
Trusted Computing Base: Big OS



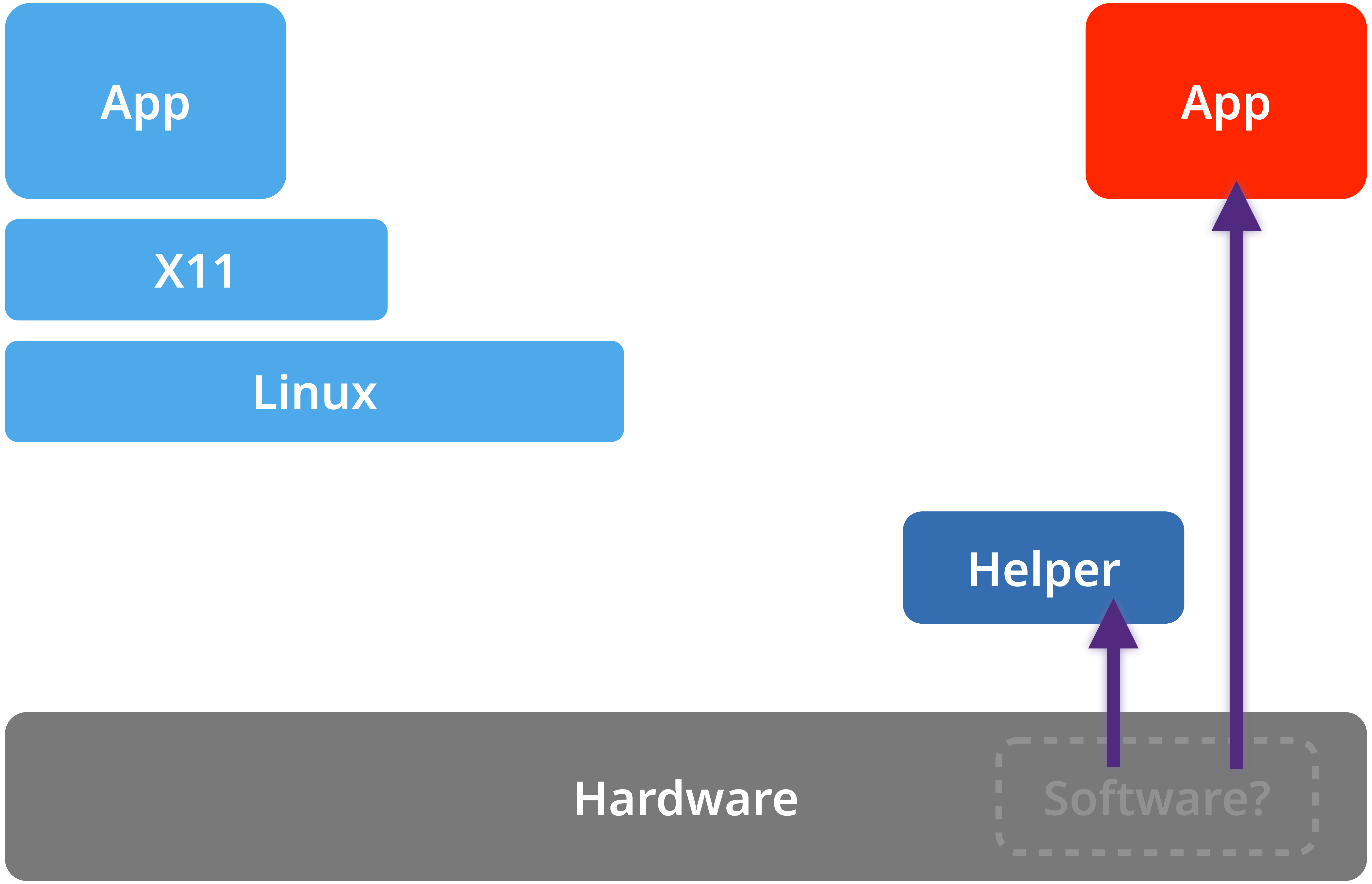
Trusted Computing Base: Small OS



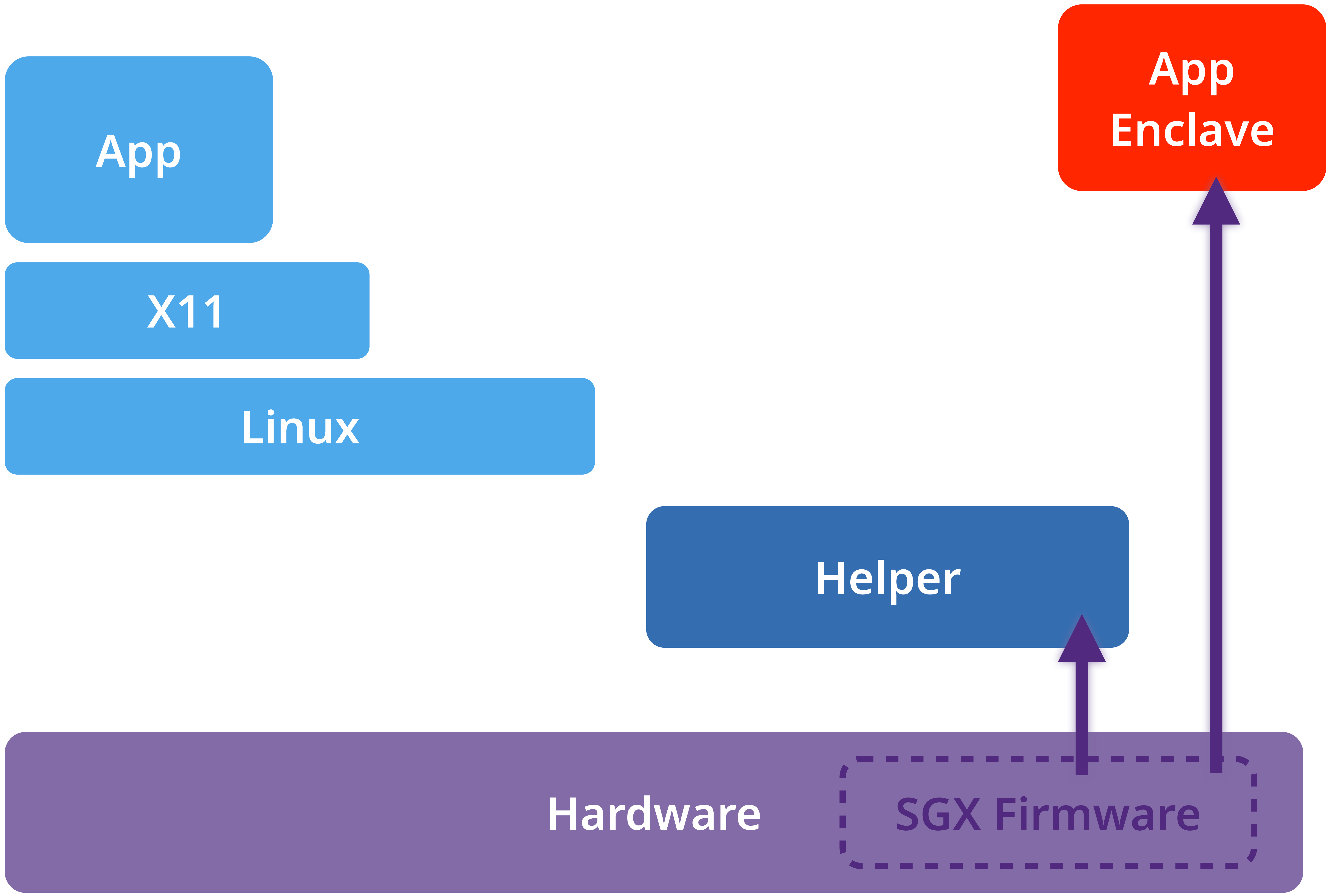
ARM TrustZone



Trusted Computing Base: Only Hardware?



Intel SGX

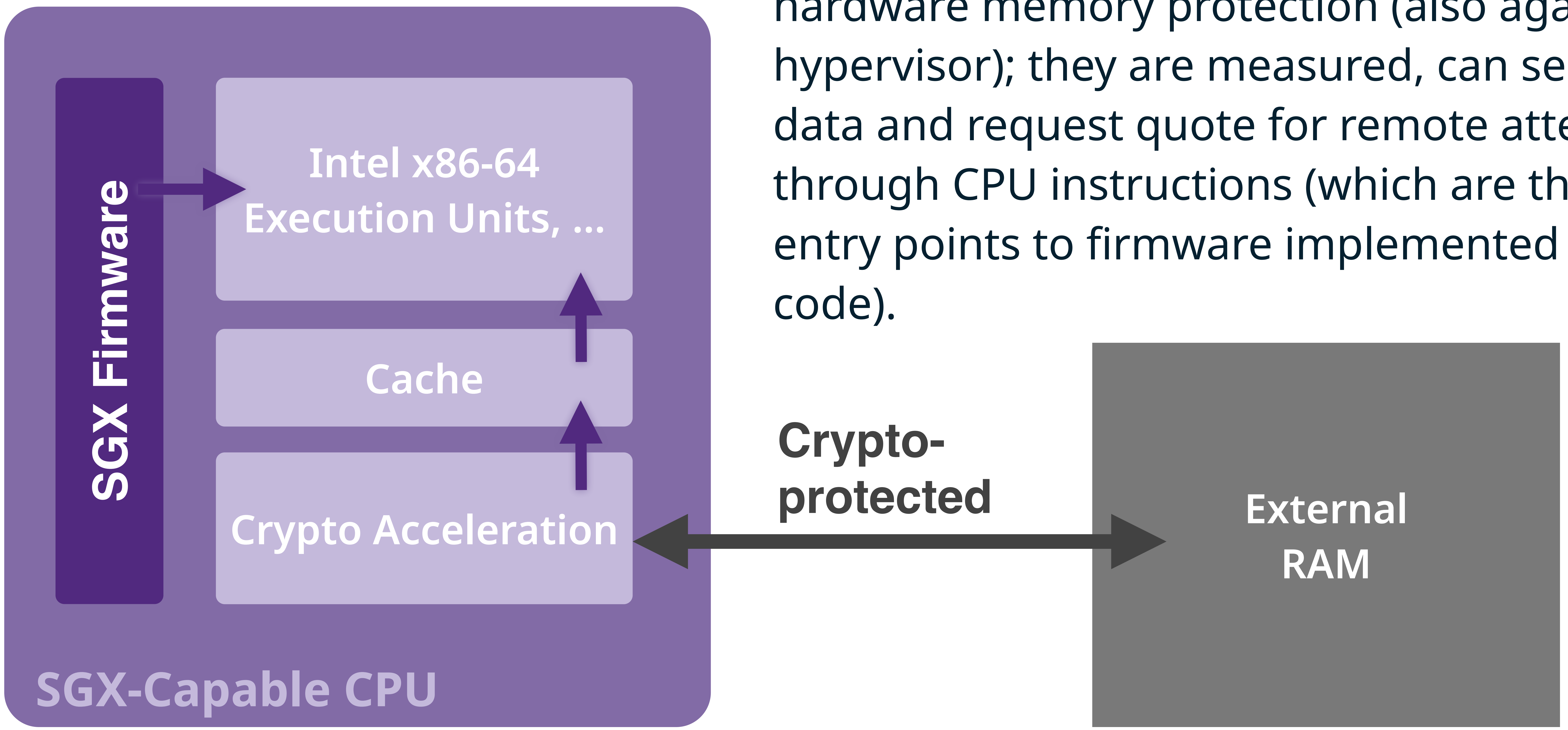


Intel SGX

“Enclaves” for applications:

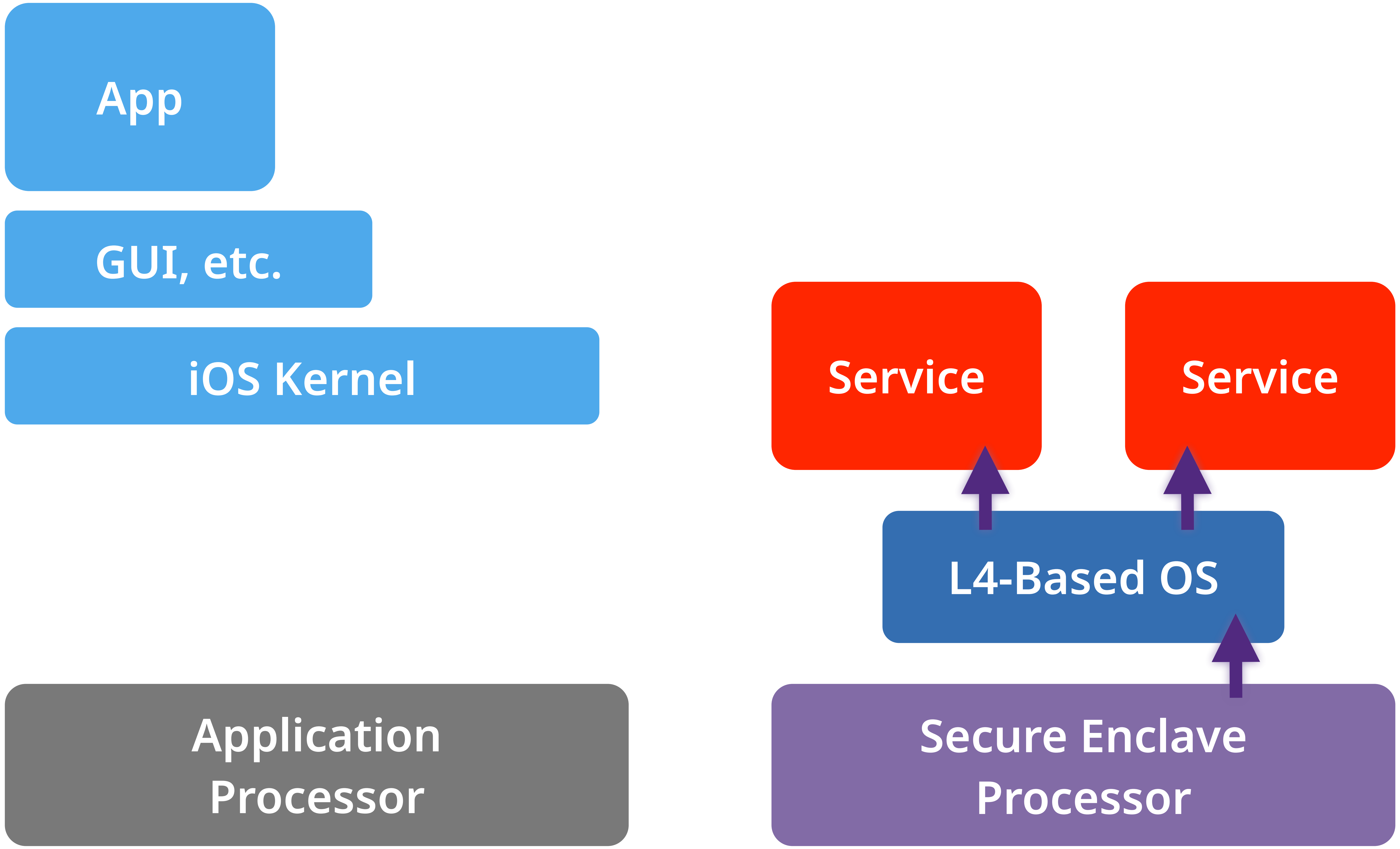
- Established per special SGX instructions
- Measured by CPU
- Provides controlled entry points
- Resource management via untrusted OS

Intel SGX



Applications executing in enclaves benefit from hardware memory protection (also against OS and hypervisor); they are measured, can seal and unseal data and request quote for remote attestation, all through CPU instructions (which are themselves entry points to firmware implemented as x86-64 code).

Apple Secure Enclave Processor



References

References

Important Foundational Paper:

"Authentication in Distributed Systems: Theory and Practice", Butler Lampson, Martin Abadi, Michael Burrows, Edward Wobber, ACM Transactions on Computer Systems (TOCS)

Technical documentation:

- Trusted Computing Group's specifications https://www.trustedcomputinggroup.org
- ARM Trustzone, Intel SGX, TDX, AMD SEV, Arm CCA, and other vendor documentation