TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

TRUSTED COMPUTING

CARSTEN WEINHOLD, HERMANN HÄRTIG

Lecture Goals

Goal: Understand principles of:

- Authenticated booting, relation to (closed) secure booting
- Remote attestation
- Sealed memory
- Dynamic root of trust, late launch
- 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

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

Key Goals of Trusted Computing

- Prevent certain software from running
- Which computer system do I communicate with?
- Which stack of software is running …
  - … in front of me?
  - … on my server somewhere?
- 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 wants to ensure that only specific software can 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 runs
- Needs to ensure that provider cannot access data
- **Customer does not trust cloud provider**

Usage Examples (3)

Industrial Plant Control:

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

Usage Examples (4)

Anonymity Service:

- Provides anonymous communication over internet (e.g., one node in mix cascade)
- User wants to be sure that the proxy actually anonymizes ingoing and outgoing connections
- **Anonymity-service provider not trusted**

Trusted Computing Terminology

Measuring:

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

Attestation:

- Vouching for accuracy of (measured) information

Sealed Memory:

- Binding information to a (software) 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_{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

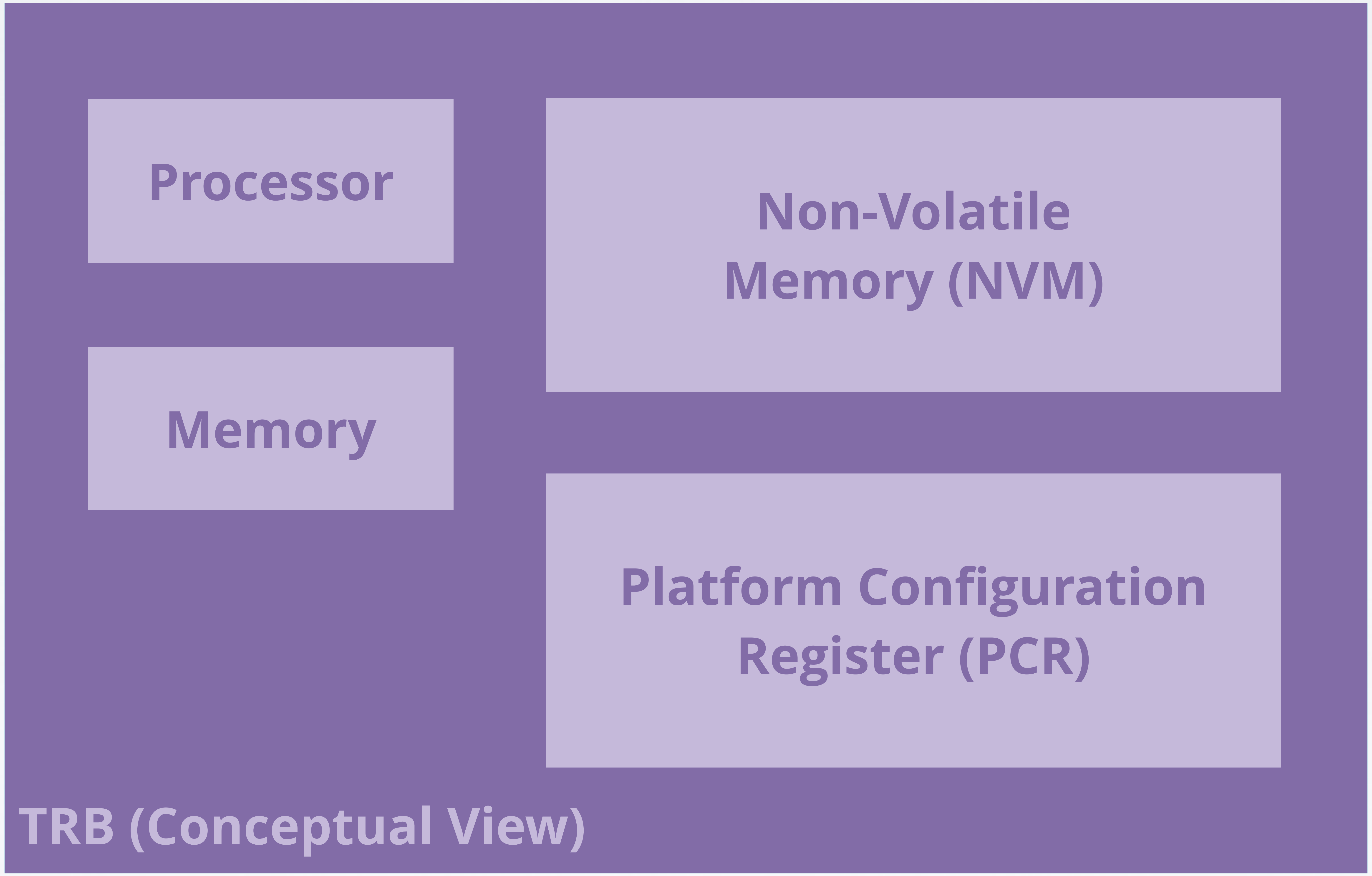
Example: program vendor FooSoft (FS)

Software identity **ID** must be known

Two ways to identify software:

- By hash: $\mathbf{ID_{Program} = H(Program)}$
- By signature: $\mathbf{\{Program, ID_{Program}\}FS_{priv}}$
  - Signature must be available (e.g., shipped with program)
  - Use $\mathbf{FS_{pub}}$ to check signature
  - $\mathbf{(H(Program), FS_{pub})}$ can serve as $\mathbf{ID_{Program}}$

Tamper-Resistant Black Box (TRB)



Secure Booting ("Burn in the OS")

OS stored in read-only memory (flash)

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

- Load OS code, compare **H(loaded OS code)** to preset **H(OS)**
- Abort if different

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

- Load OS code, check signature of loaded OS code using **FS_{pub}**
- Abort if check fails

TECHNISCHE UNIVERSITÄT DRESDEN

Authenticated Booting ("Choose your OS")

Steps:

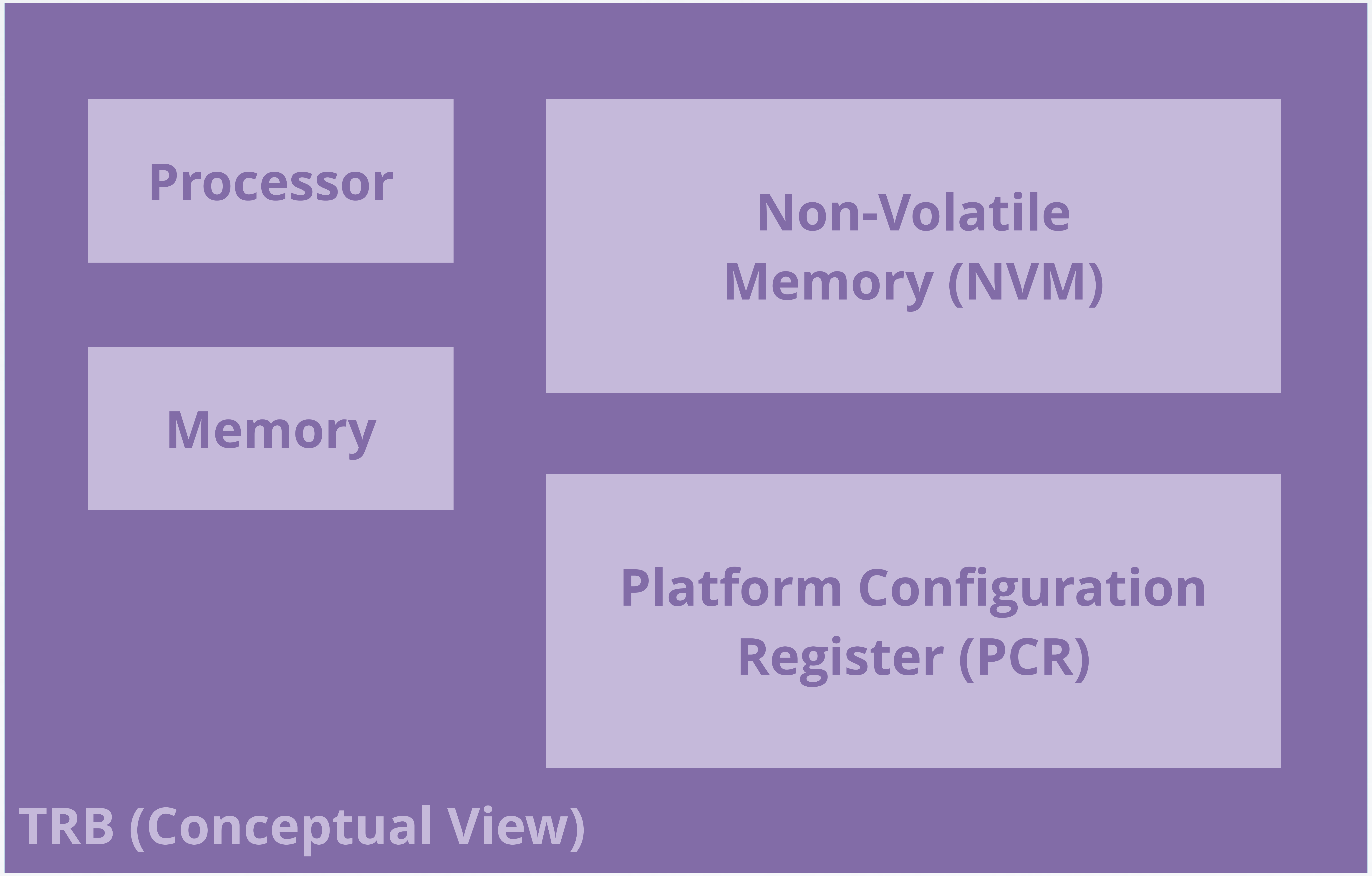
1) Preparation by OS and TRB vendors
2) Booting & measuring
3) Remote attestation

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)**

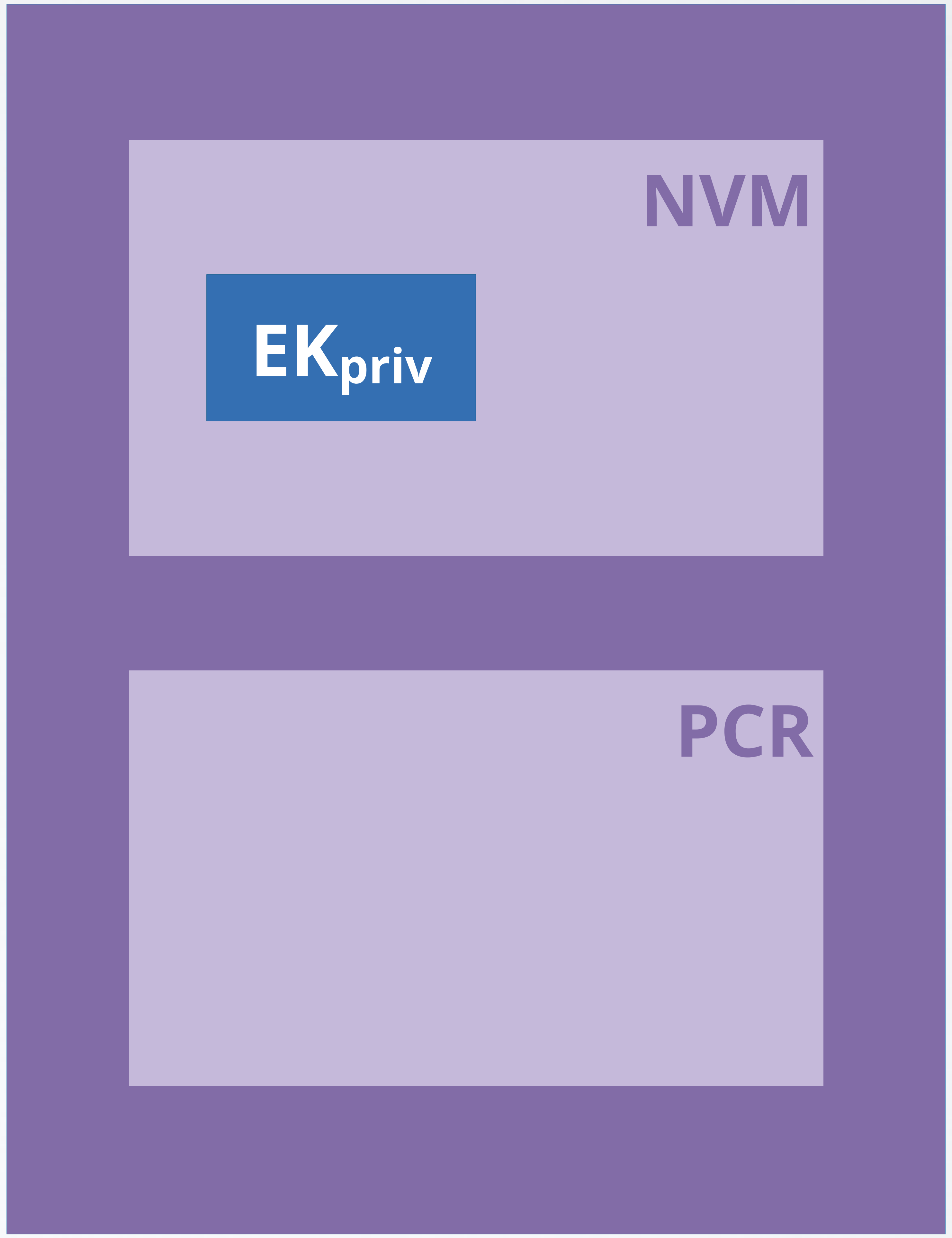
Tamper-Resistant Black Box (TRB)



TRB Initialization

1b) Preparation by TRB vendor:

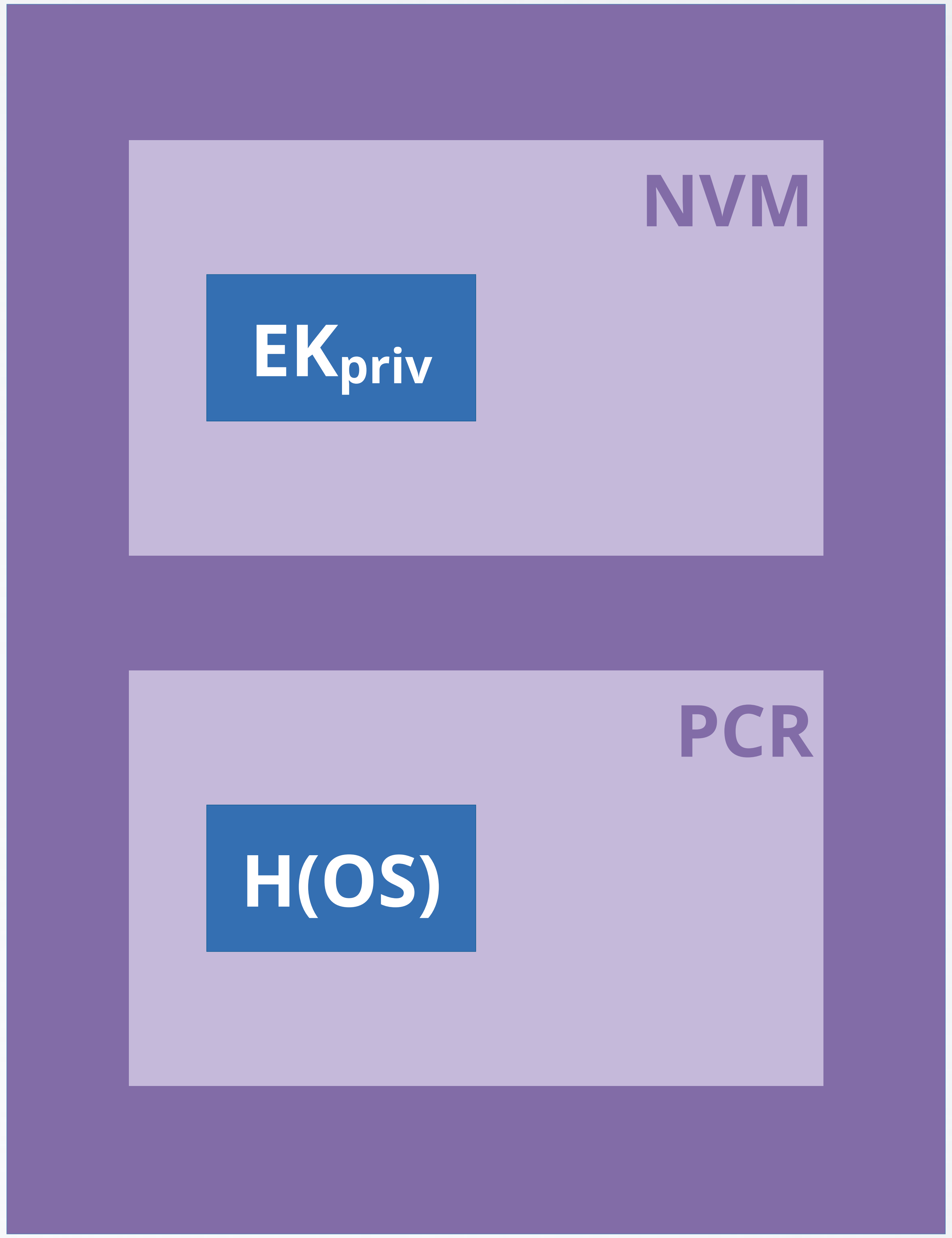
- TRB generates "Endorsement Key" pair: $\mathbf{EK_{pair}}$
- TRB Stores $\mathbf{EK_{priv}}$ in TRB NVM
- TRB publishes $\mathbf{EK_{pub}}$
- TRB vendor certifies: {"a valid EK", $\mathbf{EK_{pub}}$}$\mathbf{TRBVendor_{priv}}$



Authenticated Booting

2) Booting & measuring:

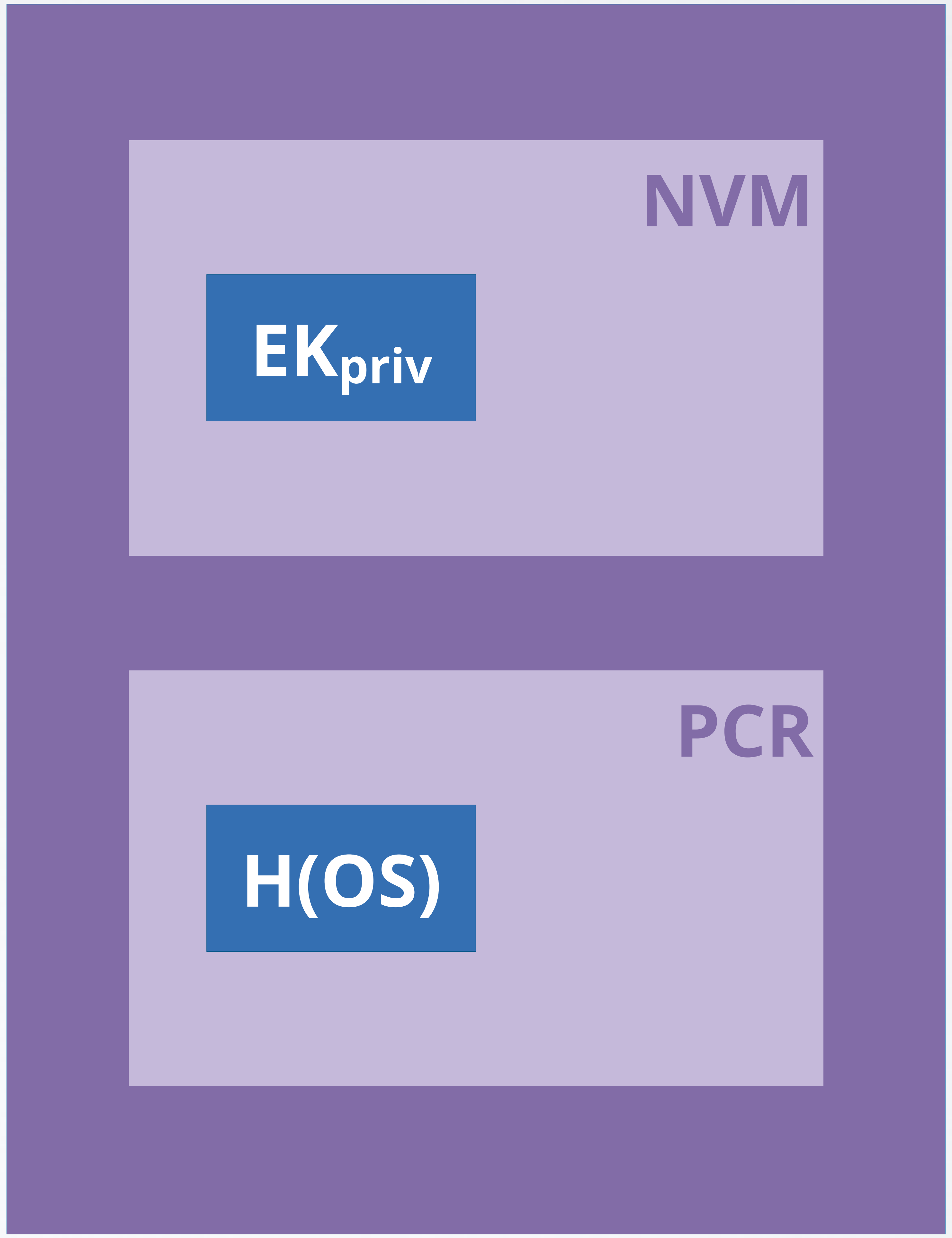
- TRB resets
- TRB computes ("measures") hash **H(OS)** of loaded OS
- Records **H(OS)** in platform configuration register **PCR**
- TRB starts OS
- **Note: PCR** not directly writable



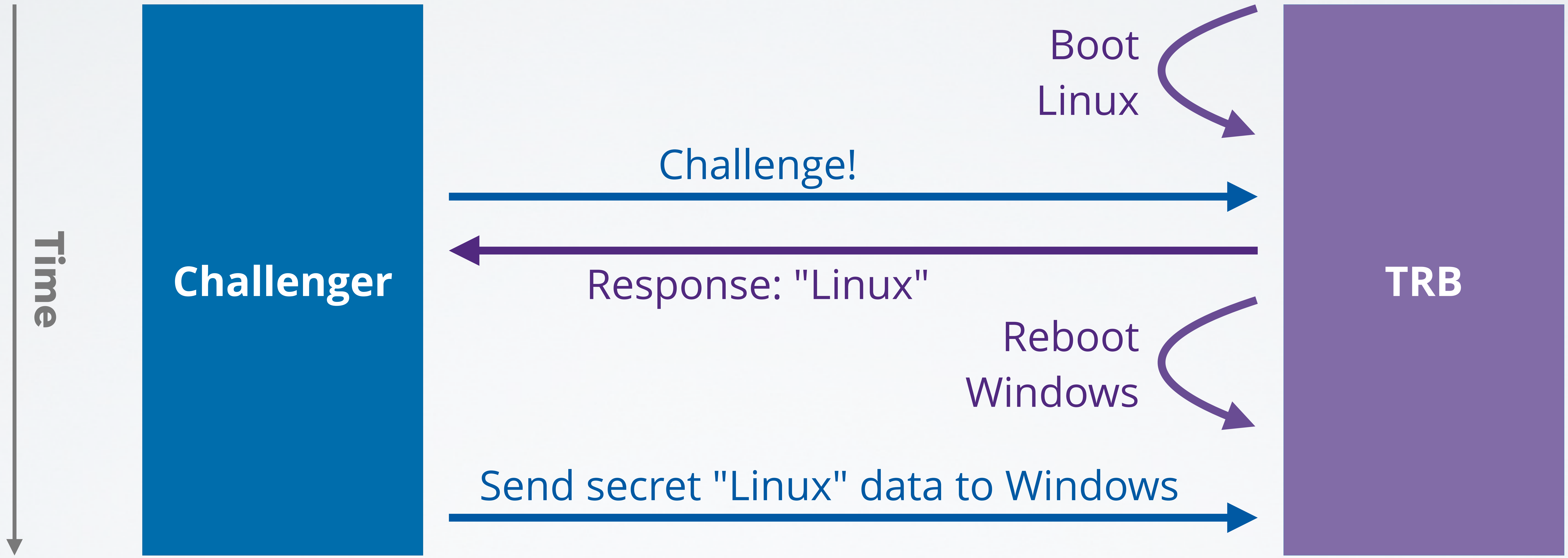
Remote Attestation (Simplified)

3) Remote Attestation:

- Remote computer sends "challenge": **NONCE**
- TRB signs **{NONCE, PCR}EK$_{\mathbf{priv}}$** (**evidence** or **attestation report**) and sends it to "challenger"
- Challenger evaluates evidence: checks signature, decides if OS identified by **H(OS)** in **PCR** is OK



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:
  - **$\textbf{ActiveOSAuthK}_{\textbf{pair}}$** /* for authentication (signing) */
  - **$\textbf{ActiveOSConK}_{\textbf{pair}}$** /* for concealing (encryption) */
- TRB certifies:
  $\{\textbf{ActiveOSAuthK}_{\textbf{pub}}, \textbf{ActiveOSConK}_{\textbf{pub}}, \textbf{H(OS)}\}\textbf{EK}_{\textbf{priv}}$
- Hands over to booted OS, to be used like "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:

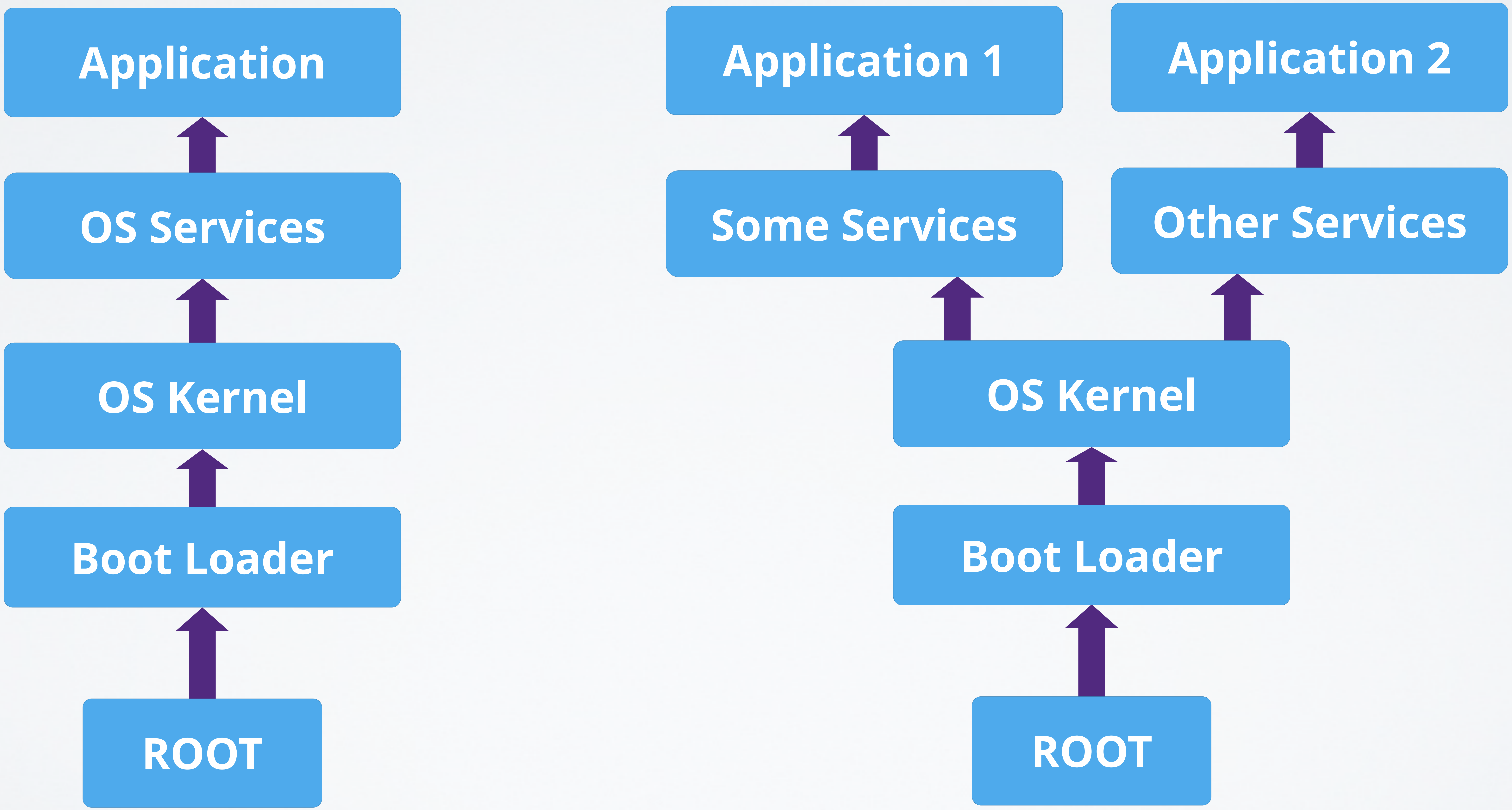
- **{**data for active OS**}ActiveOSConK$_{pub}$**

Assumptions

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

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

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 (including device drivers, etc.)

Software Stacks and Trees

Extend operation:

$$\mathbf{PCR_n = H(PCR_{n-1} \parallel new\ component)} \qquad [PCR_0=0]$$

Software Stack:

- 1 PCR value $\mathbf{PCR_n}$ after **n** components have been measured

Software "Tree":

- 1 PCR value $\mathbf{PCR_n}$ for each leaf at end of a branch of length **n**
- Needs multiple PCRs (1 per branch) that share state from **Root** to $\mathbf{PCR_{OS}}$, then diverge to leafs at $\mathbf{PCR_{App1}}$, $\mathbf{PCR_{App}}$, **...**

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$_{priv}$**

Late Launch/Dynamic Root of Trust

Problem: huge software to boot system

Solution: late launch

- Use arbitrary software to start system and load all software
- Provide specific instruction to enter “secure mode”
  - Put hardware in secure state (stop all processors, I/O, ...)
  - Measure software and record into PCR
- **AMD (skinit):** hashes arbitrary "secure loader" and start it
- **Intel (senter):** starts boot code (must be signed by Intel)

The Need for Trusted Storage

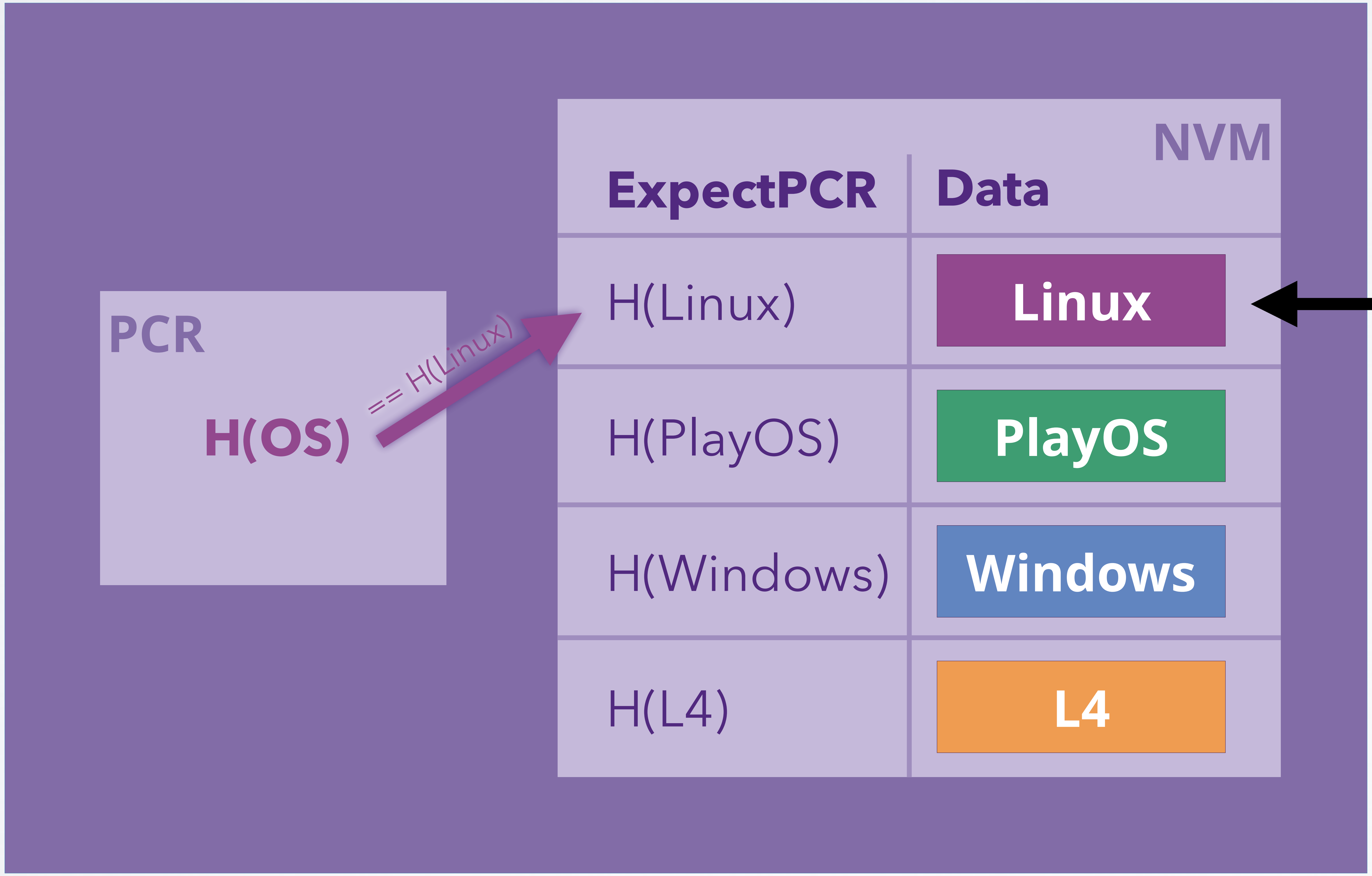
Use case from earlier example:

- 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

Sealed Memory Principle

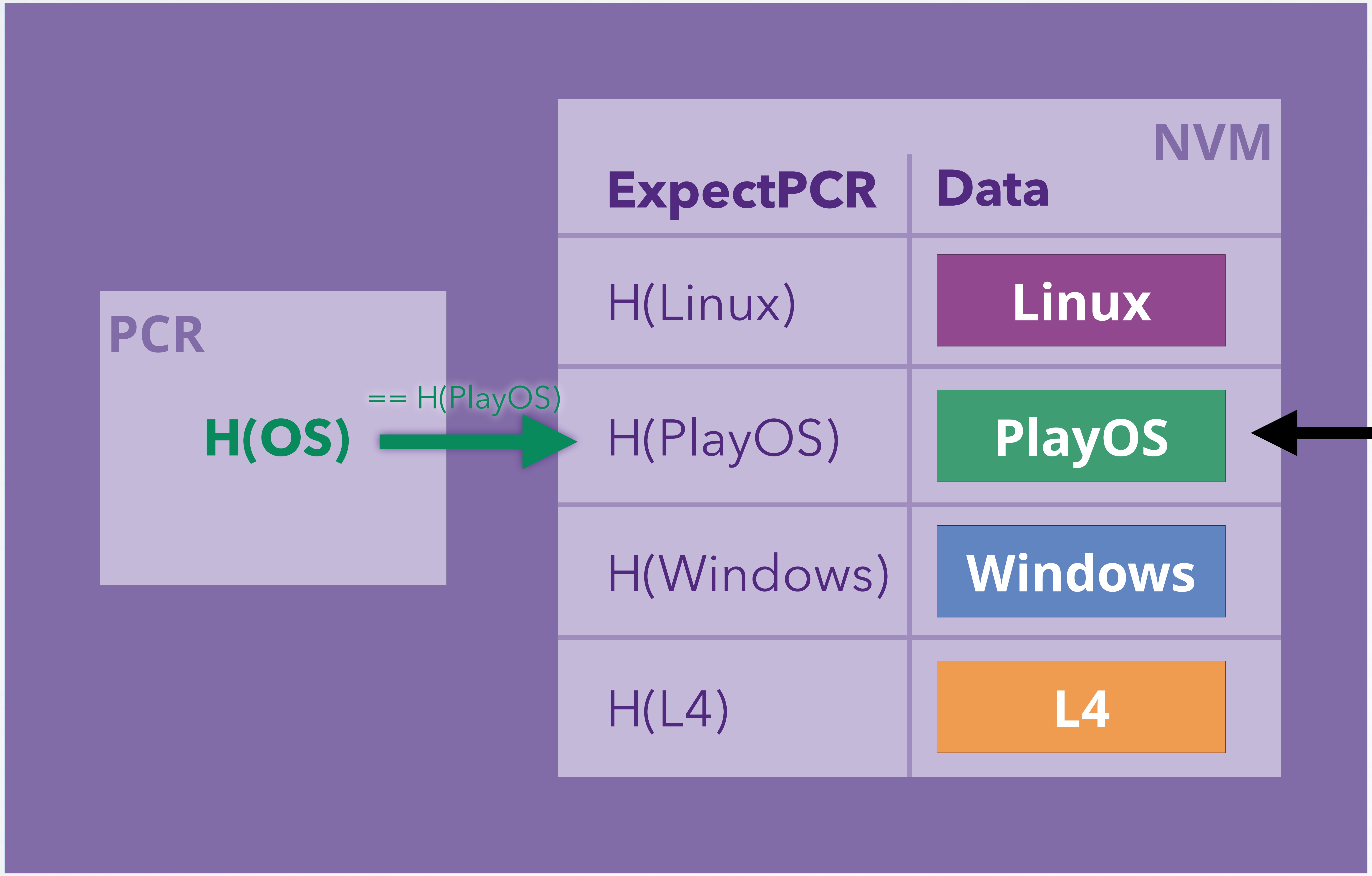


Add/remove/read/write "Sealed Memory" slots

Can be accessed by currently active OS

Other slots inaccessible due to PCR mismatch

Sealed Memory Principle

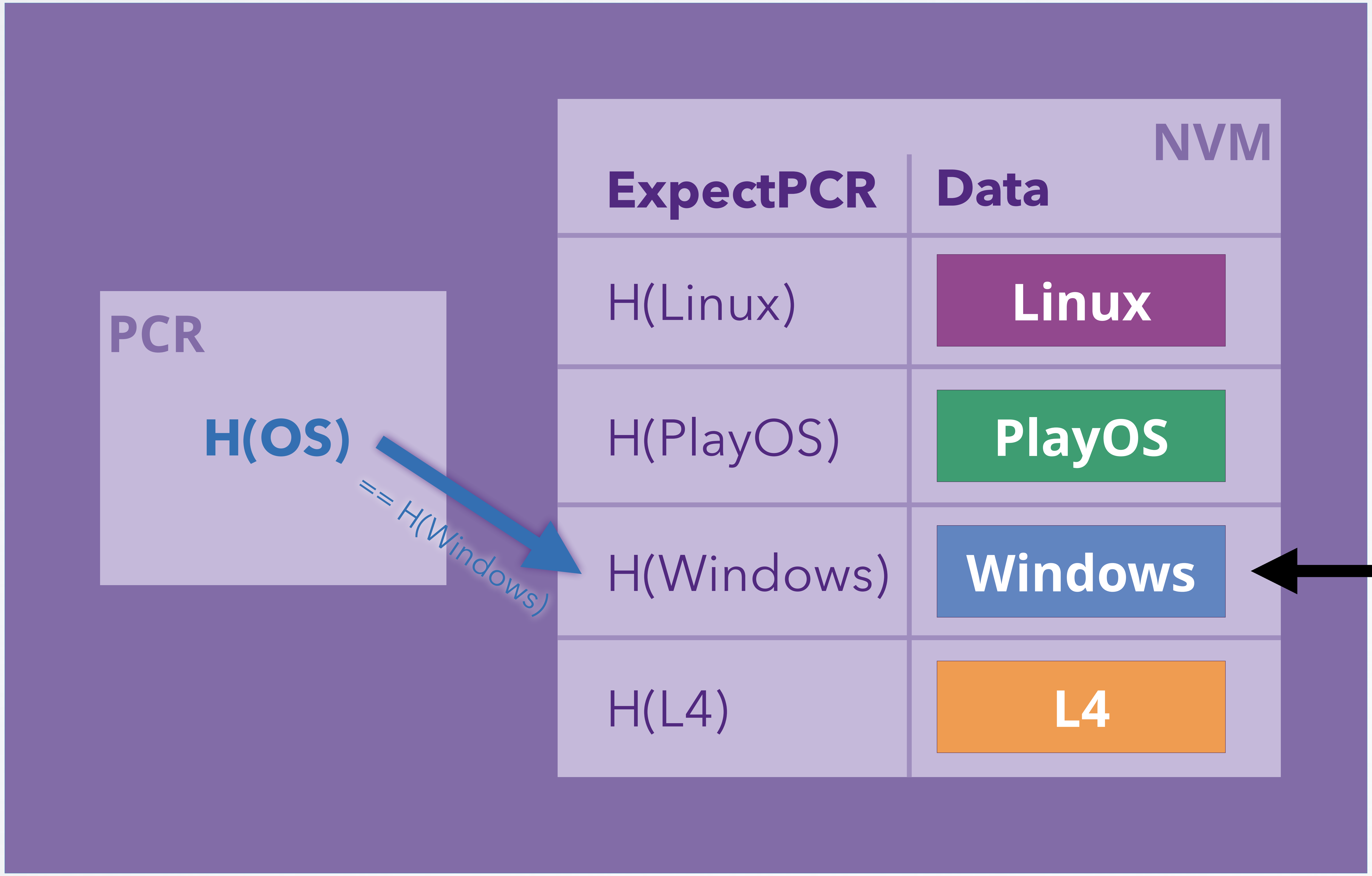


Add/remove/read/write "Sealed Memory" slots

Can be accessed by currently active OS

Other slots inaccessible due to PCR mismatch

Sealed Memory Principle

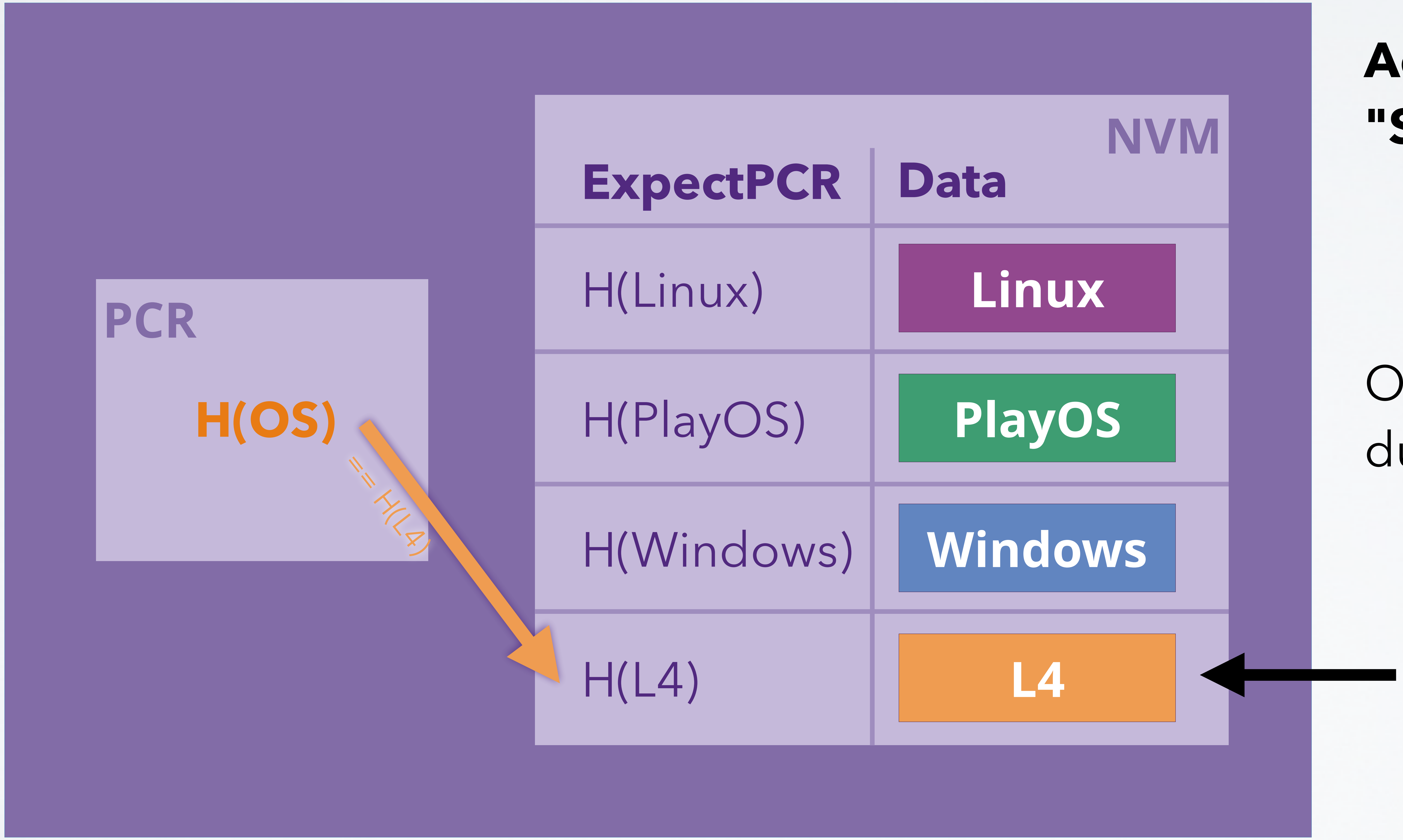


Add/remove/read/write "Sealed Memory" slots

Can be accessed by currently active OS

Other slots inaccessible due to PCR mismatch

Sealed Memory Principle



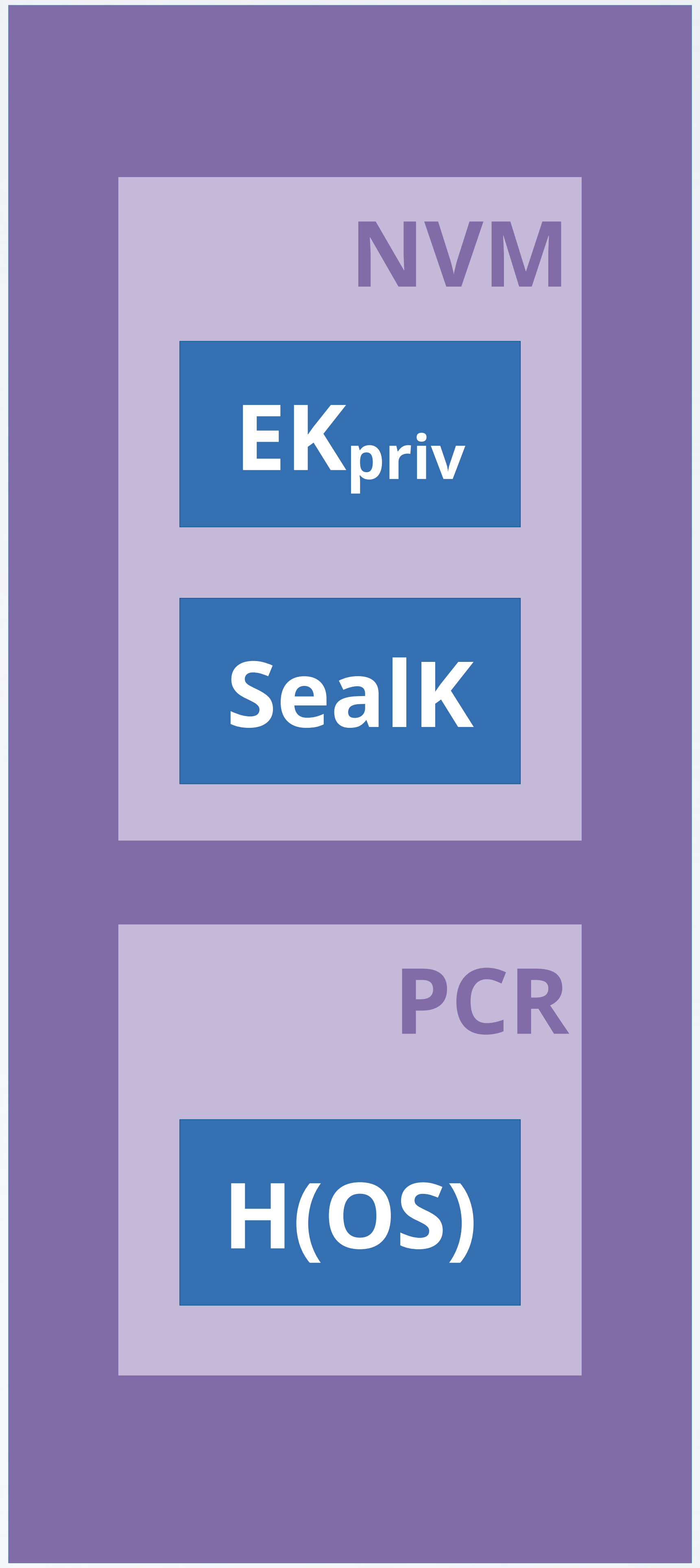
Add/remove/read/write "Sealed Memory" slots

Other slots inaccessible due to PCR mismatch

Can be accessed by currently active 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

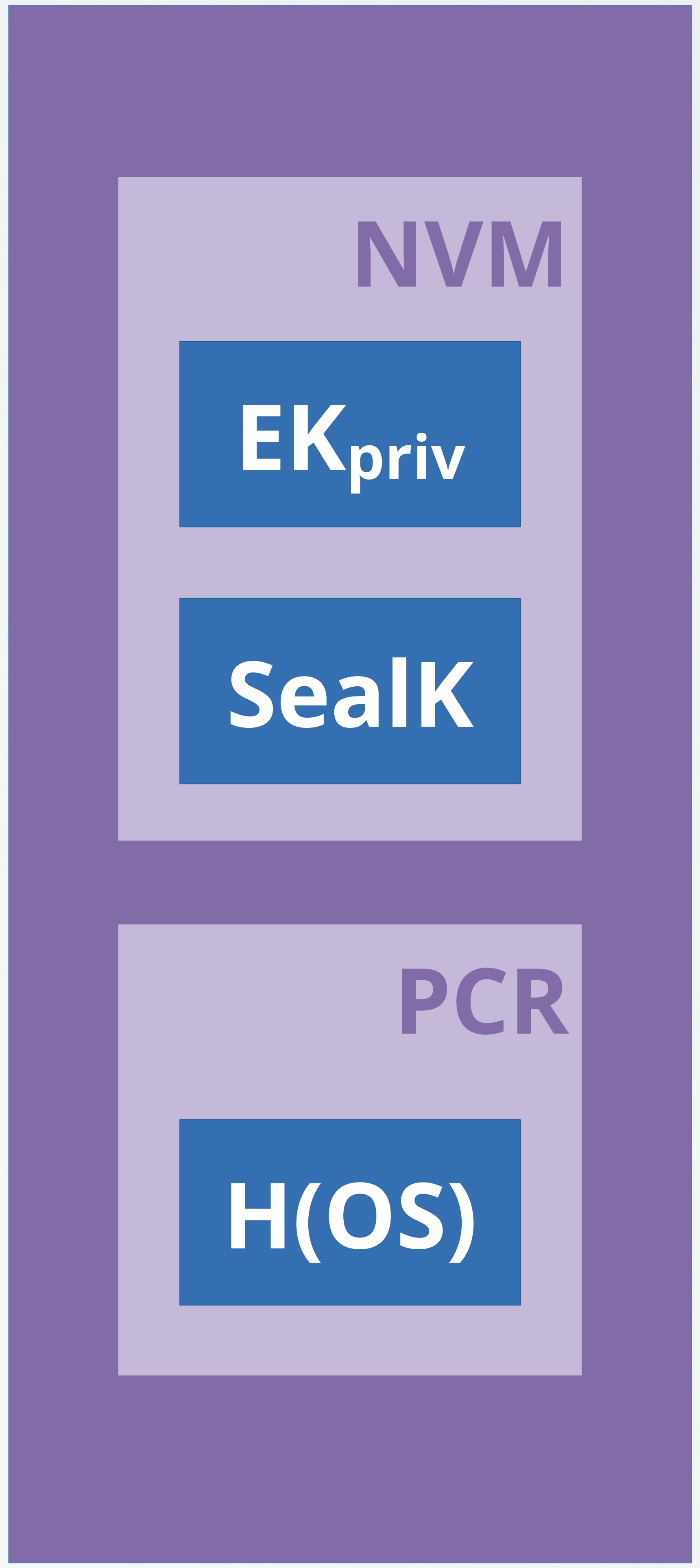
- Sealed (encrypted) data can be stored outside of TRB, allows to keep NVM small
- When sealing, arbitrary "expected PCR" values can be specified (e.g., future version of OS, or entirely different OS)

{H(Linux), Linux }SealK

{H(Windows), Windows }SealK

{H(PlayOS), PlayOS }SealK

{H(L4), L4 }SealK



Example

- **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**

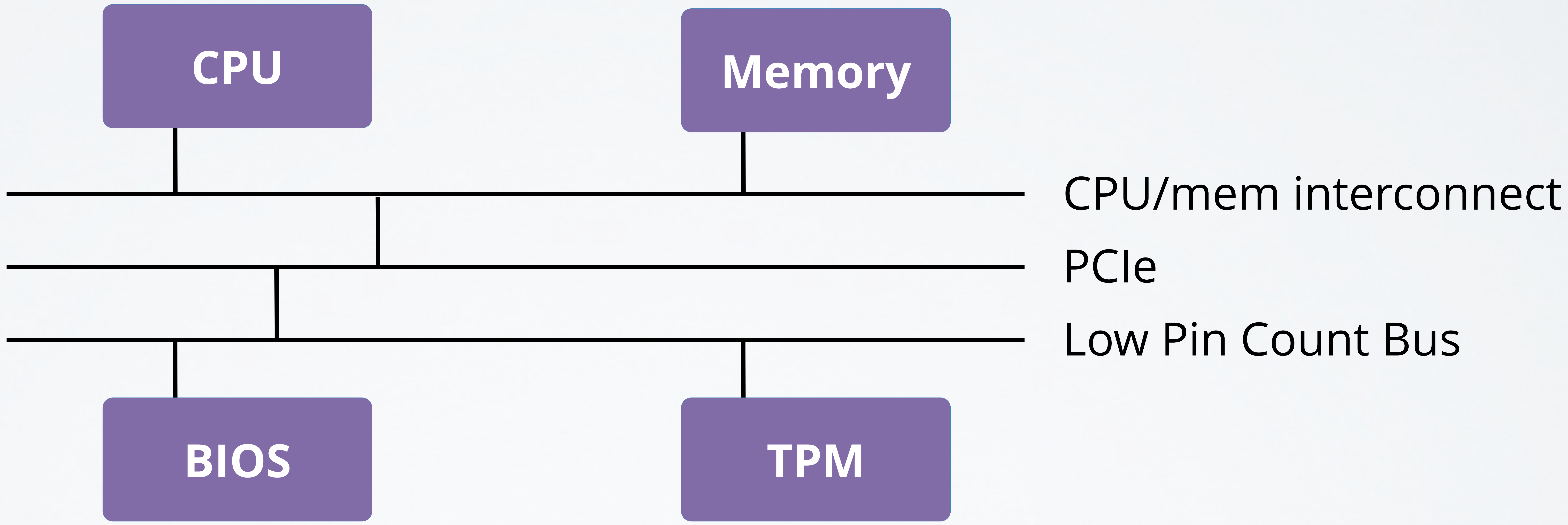
Tamper Resistant Black Box?

Ideally: includes CPU, Memory, ...

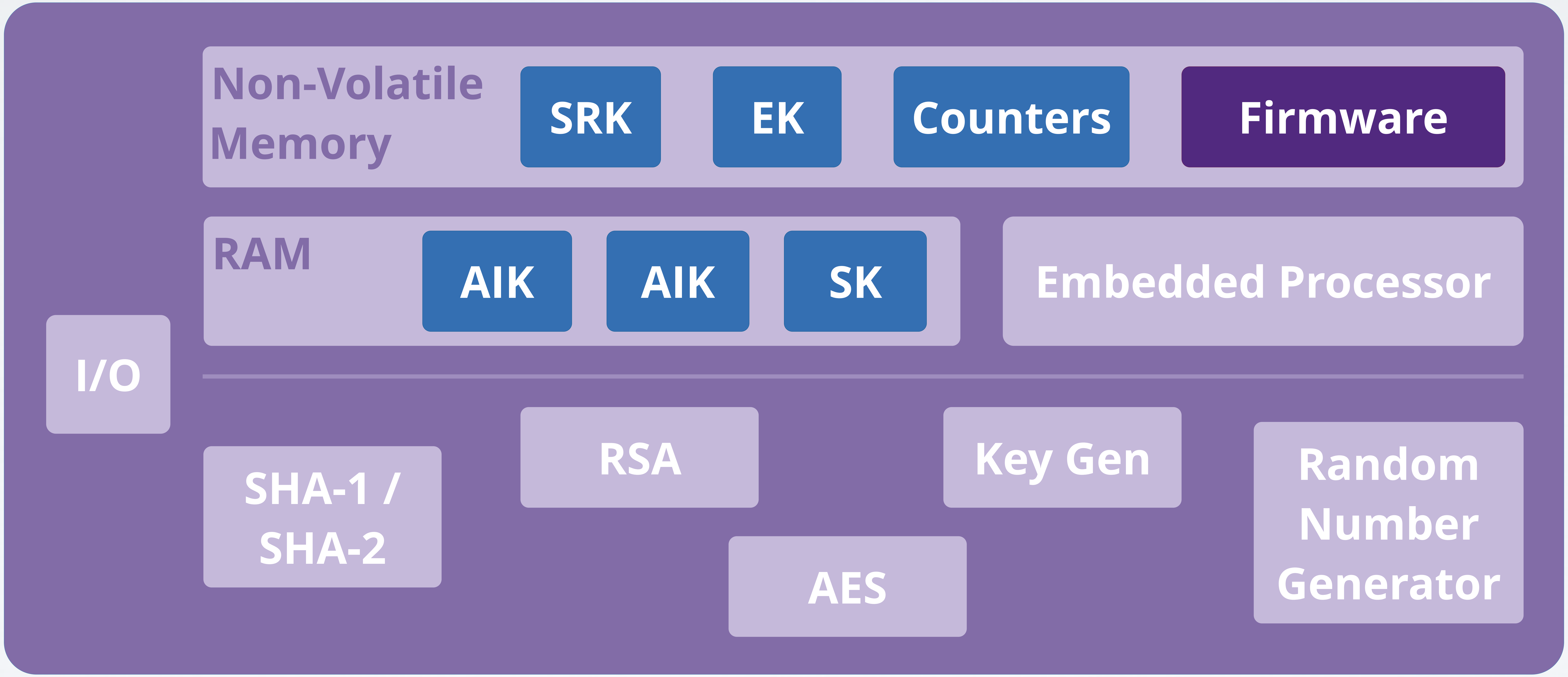
In practice:

- Additional physical protection (e.g., IBM 4758, → Wikipedia)
- Hardware support:
  - Trusted Platform Module (TPM): requires careful design to allow firmware updates, etc.
  - Add a new privilege mode: Intel SGX, Arm TrustZone,...
  - Add encrypted VMs: Intel TDX, AMD SEV, Arm CCA, ...

TCG PC Platform: Trusted Platform Module (TPM)



Trusted Platform Module



Protection of Application

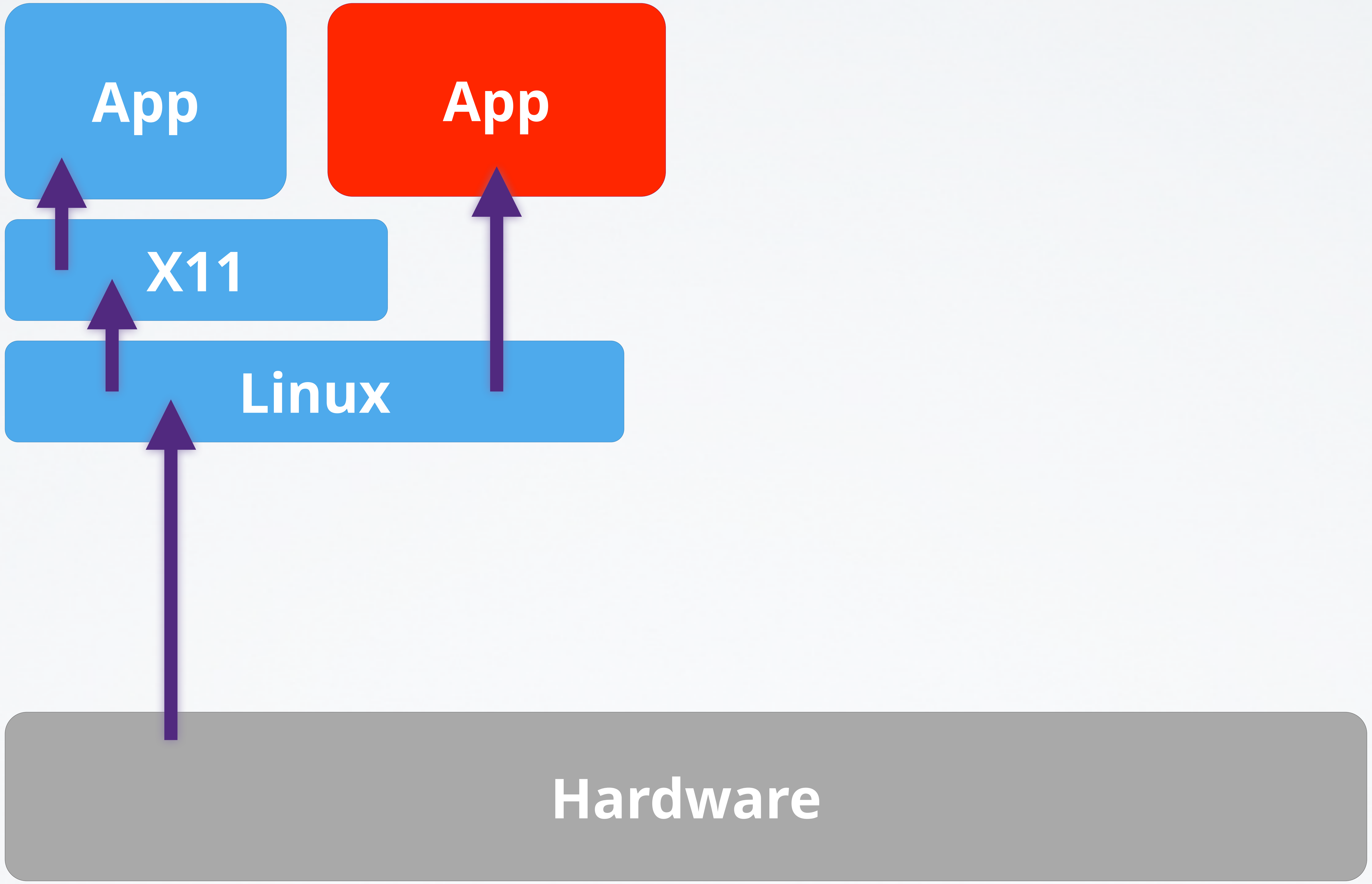
Principle Method:

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

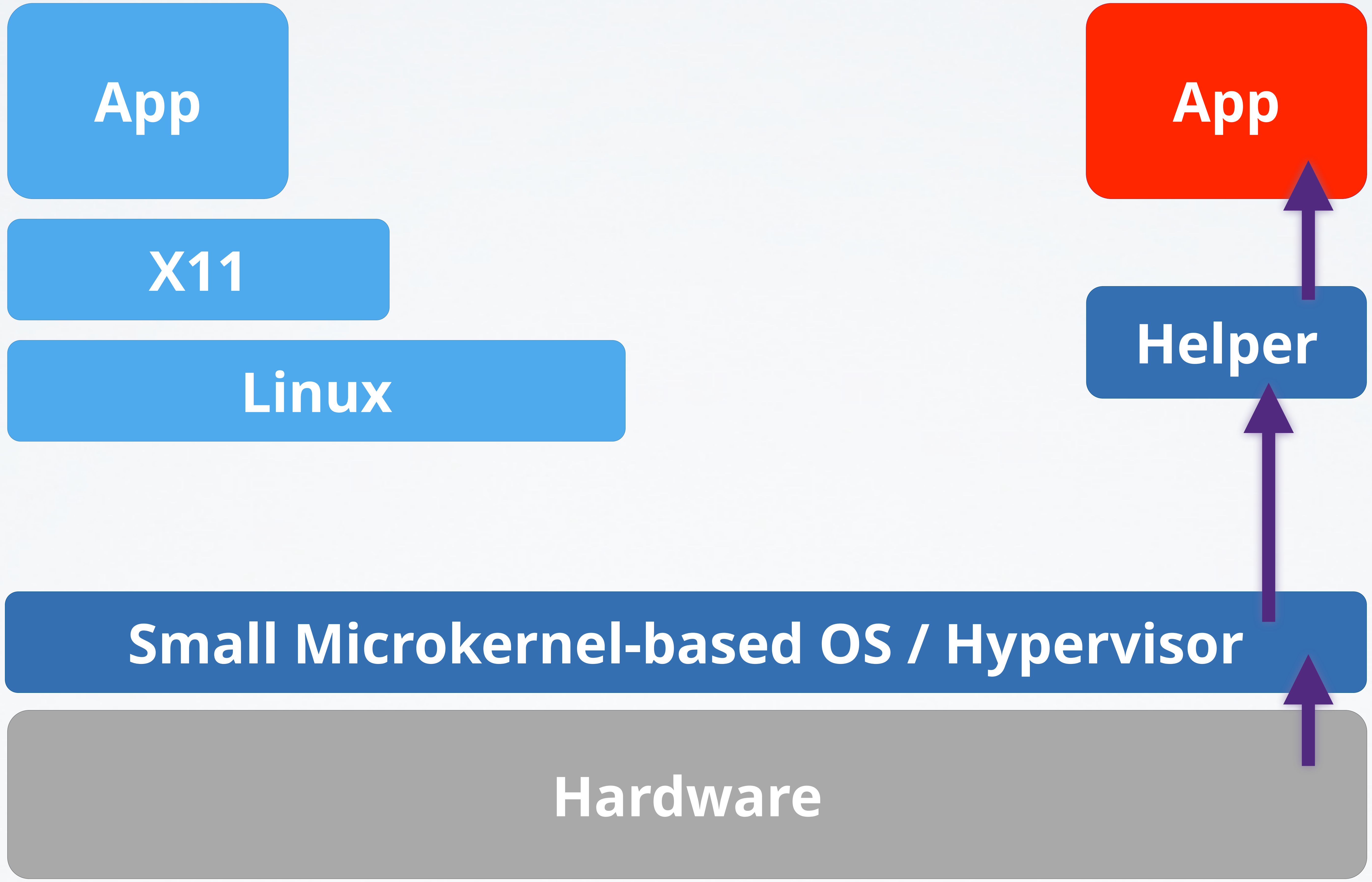
Ways to implement the method:

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

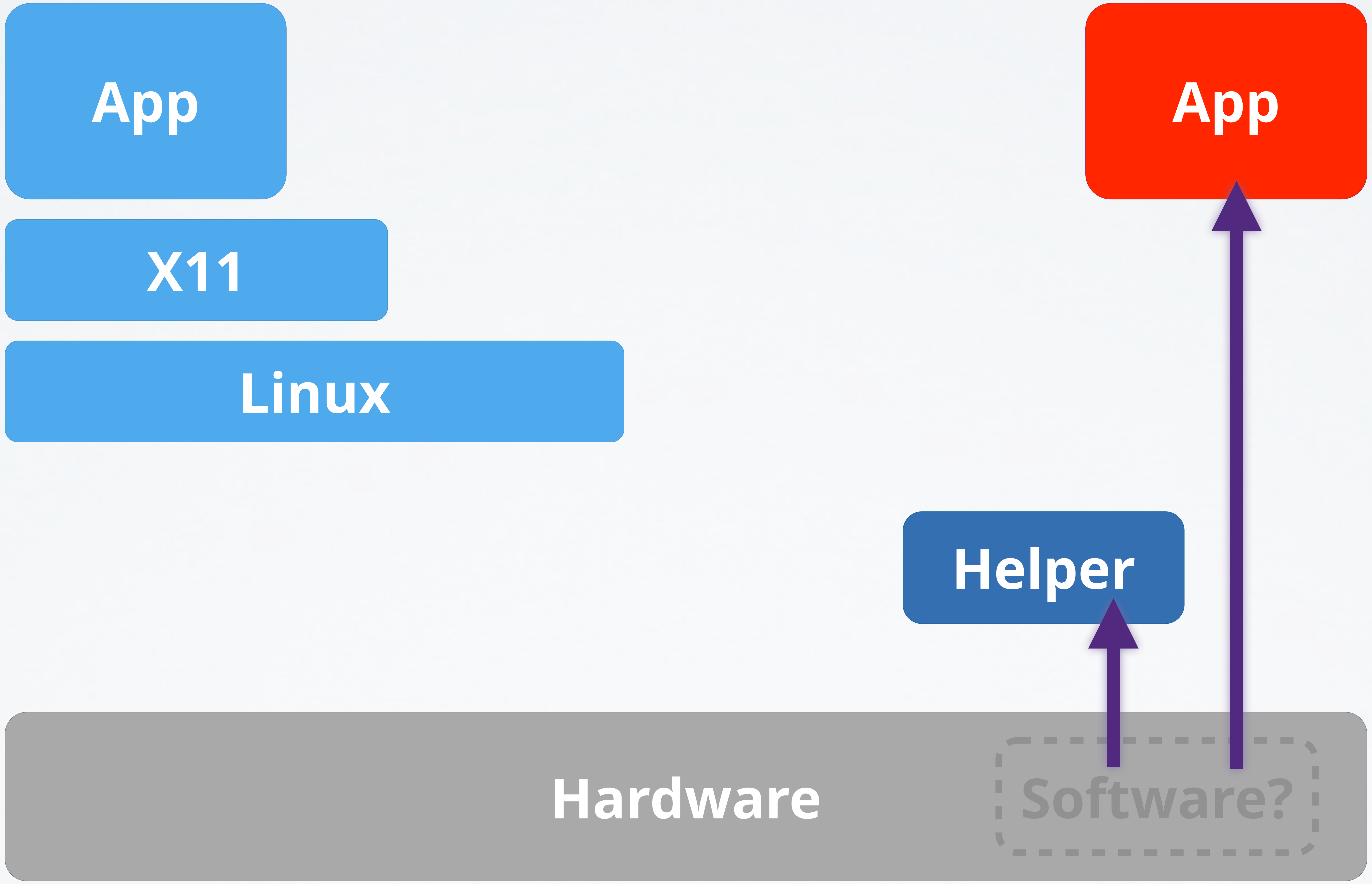
Trusted Computing Base: Big OS



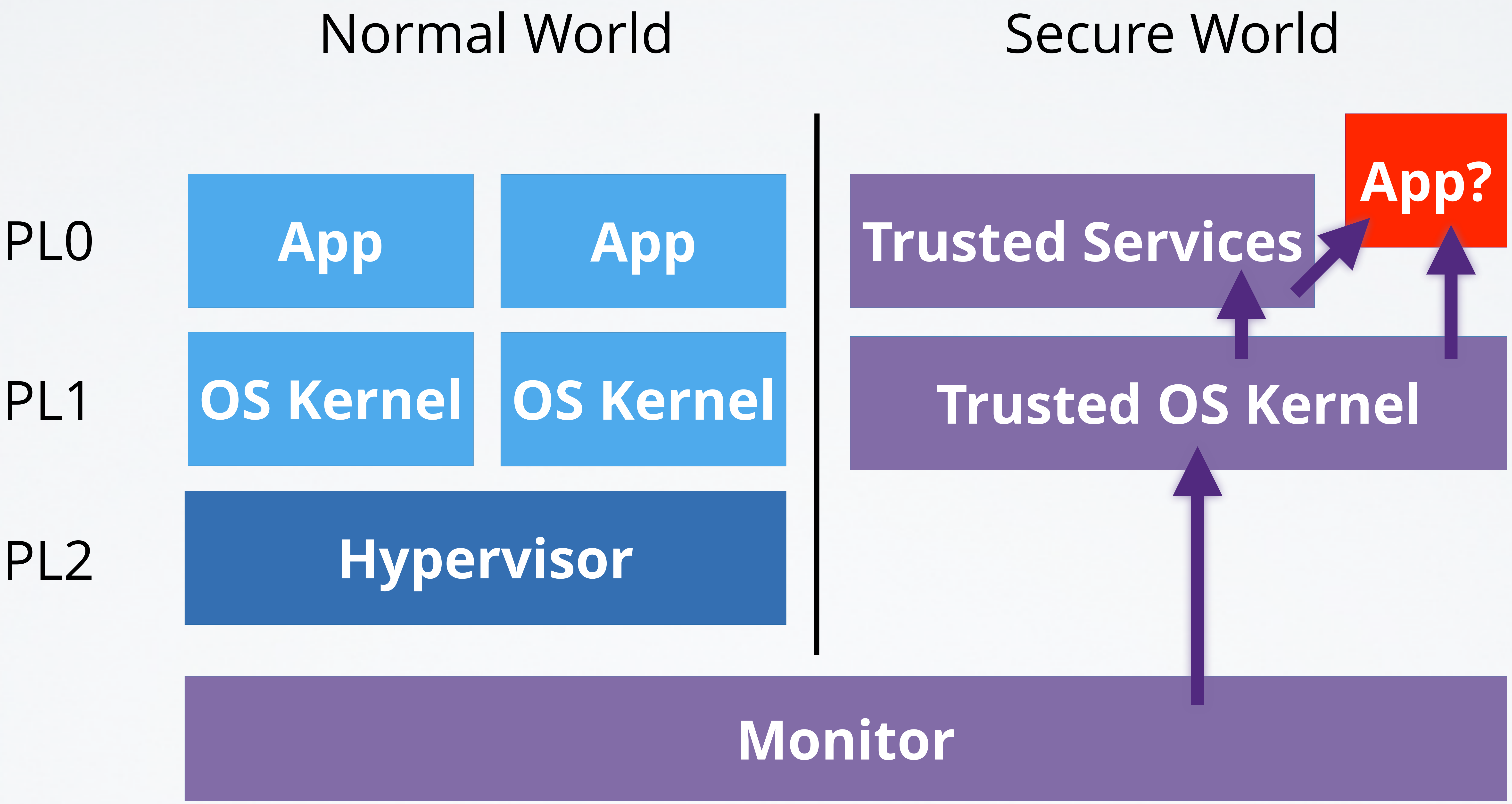
Trusted Computing Base: Small OS



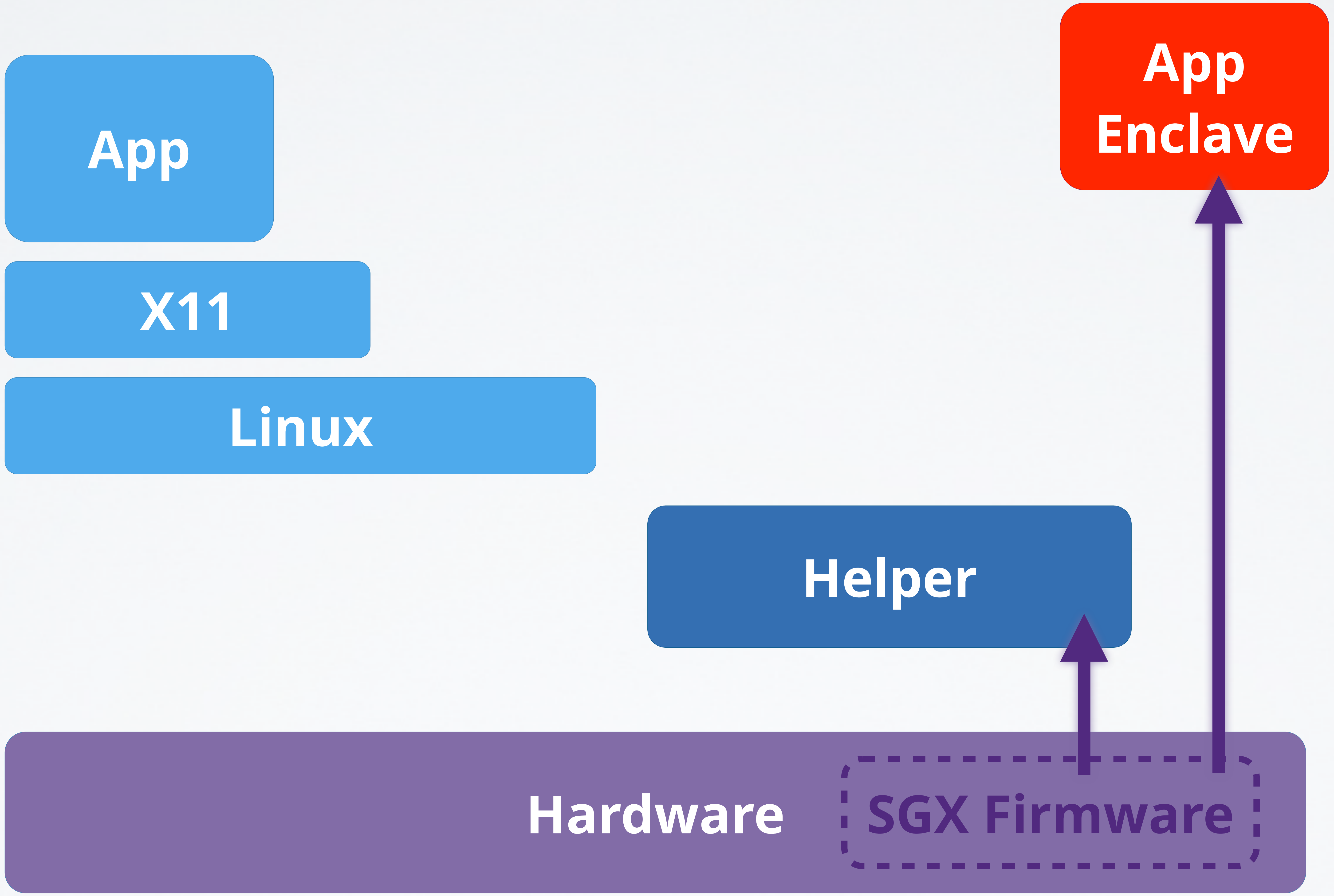
Trusted Computing Base: Only Hardware?



ARM TrustZone



Intel SGX

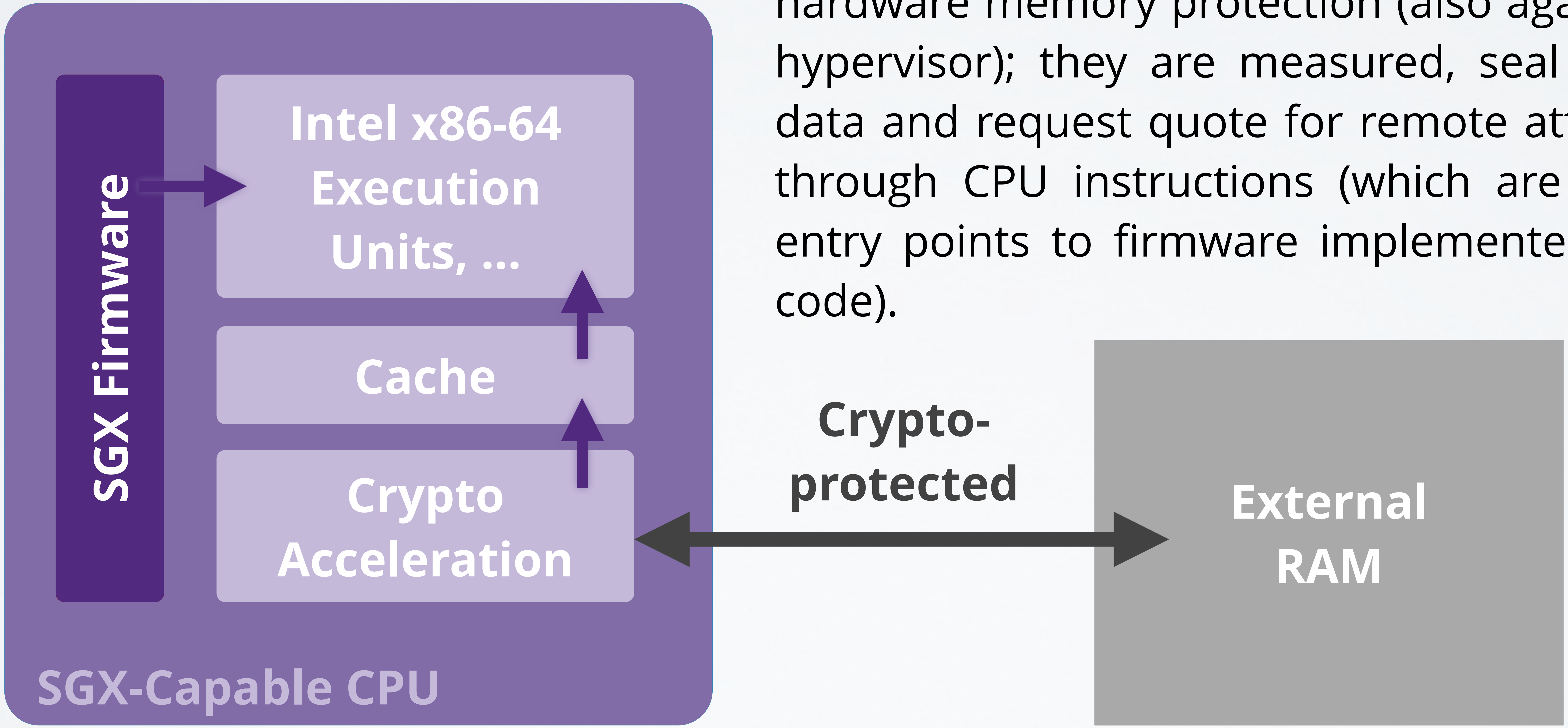


“Enclaves” for applications:

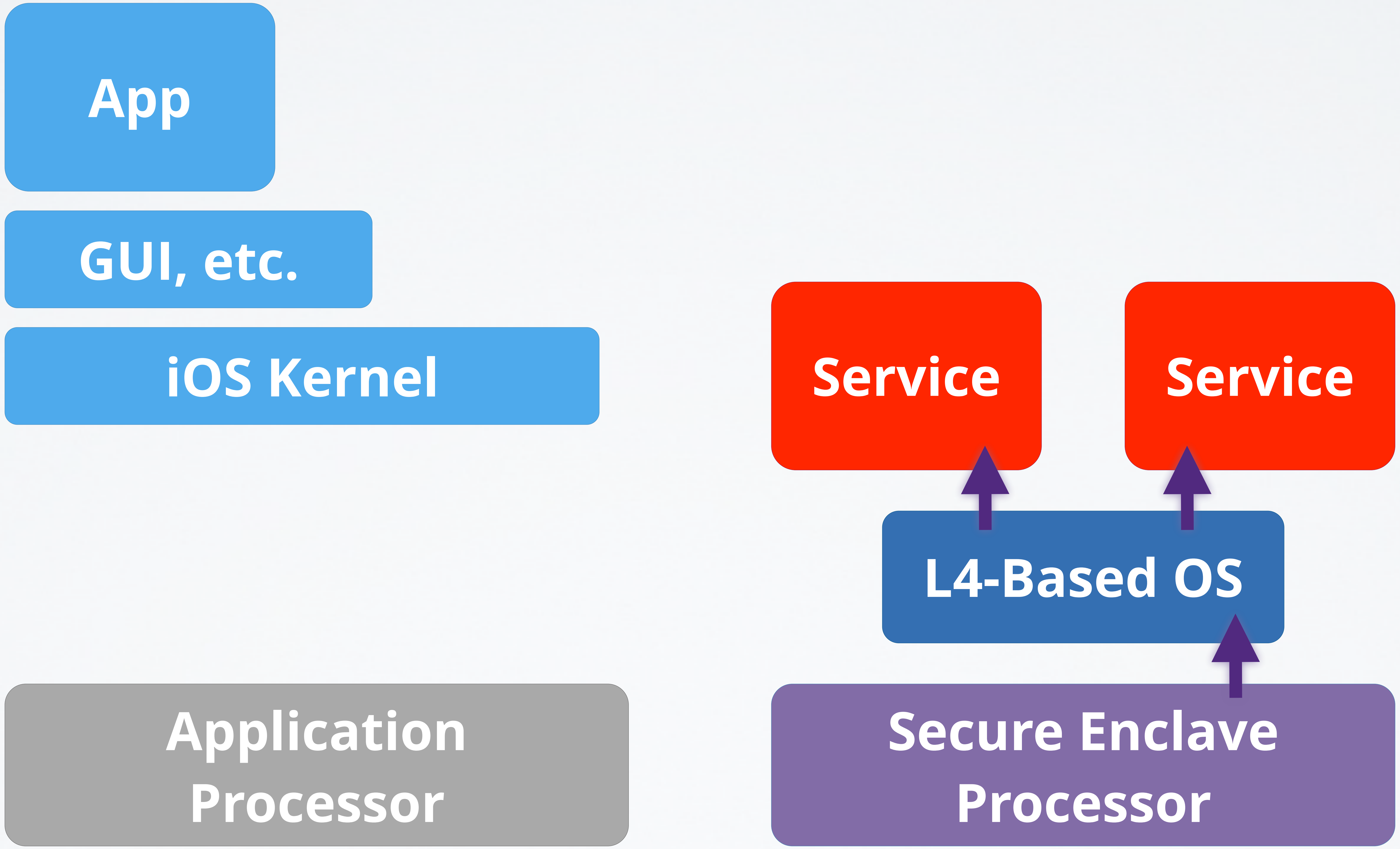
- Established per special SGX instructions
- Measured by CPU
- Provides controlled entry points
- Resource management via untrusted OS

Intel SGX

Applications executing in enclaves benefit from hardware memory protection (also against OS and hypervisor); they are measured, 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

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 vendor documentation