Integrity Verification of Software-defined Infrastructures

Prof. Antonio Lioy

< lioy@polito.it >

Politecnico di Torino

FOSAD (Bertinoro, IT)

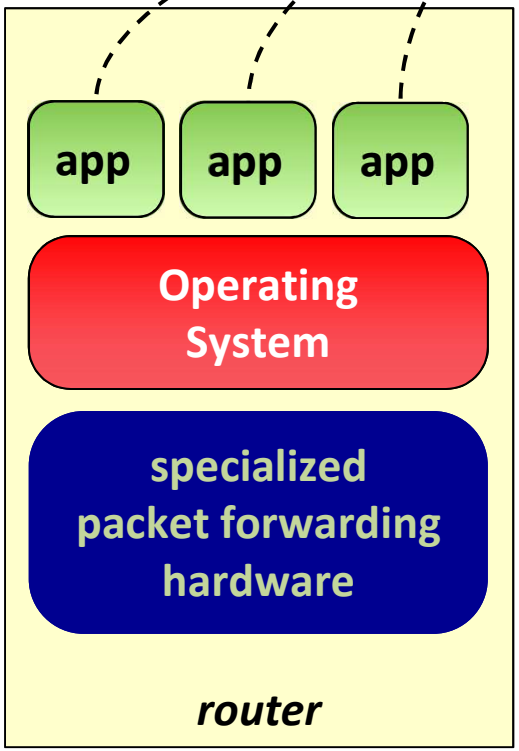
August 29-30, 2019

Acknowledgement

Part of this presentation is based on material prepared by Ludovic Jacquin of Hewlett Pack.ard Enterprise

The "classic" network approach

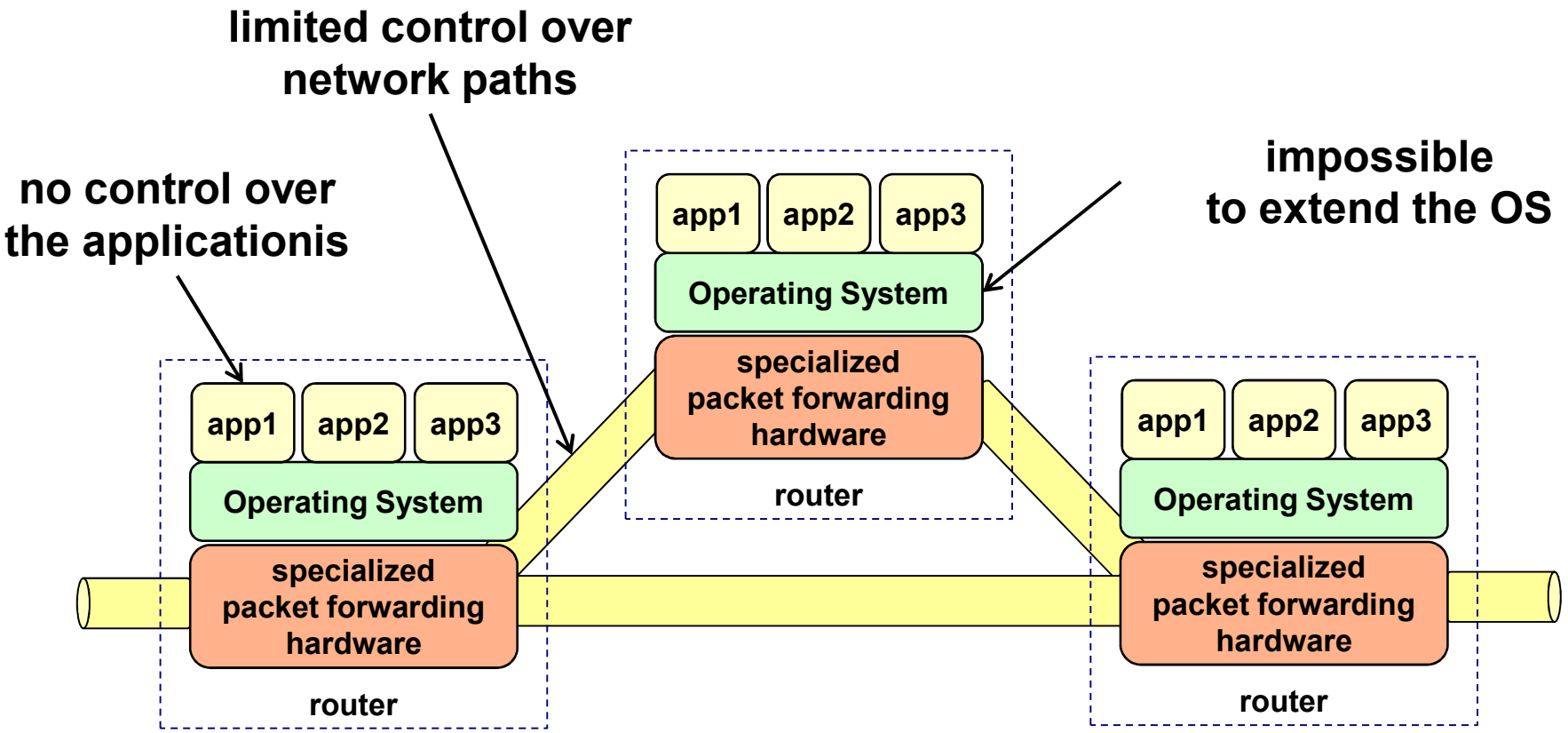
OSPF, BGP, multicast, differentiated services,
Traffic Engineering, NAT, firewall, MPLS, …



specified in more than 8000 RFCs
millions of LOC

500 M gate, 10 GB RAM
high energy consumption

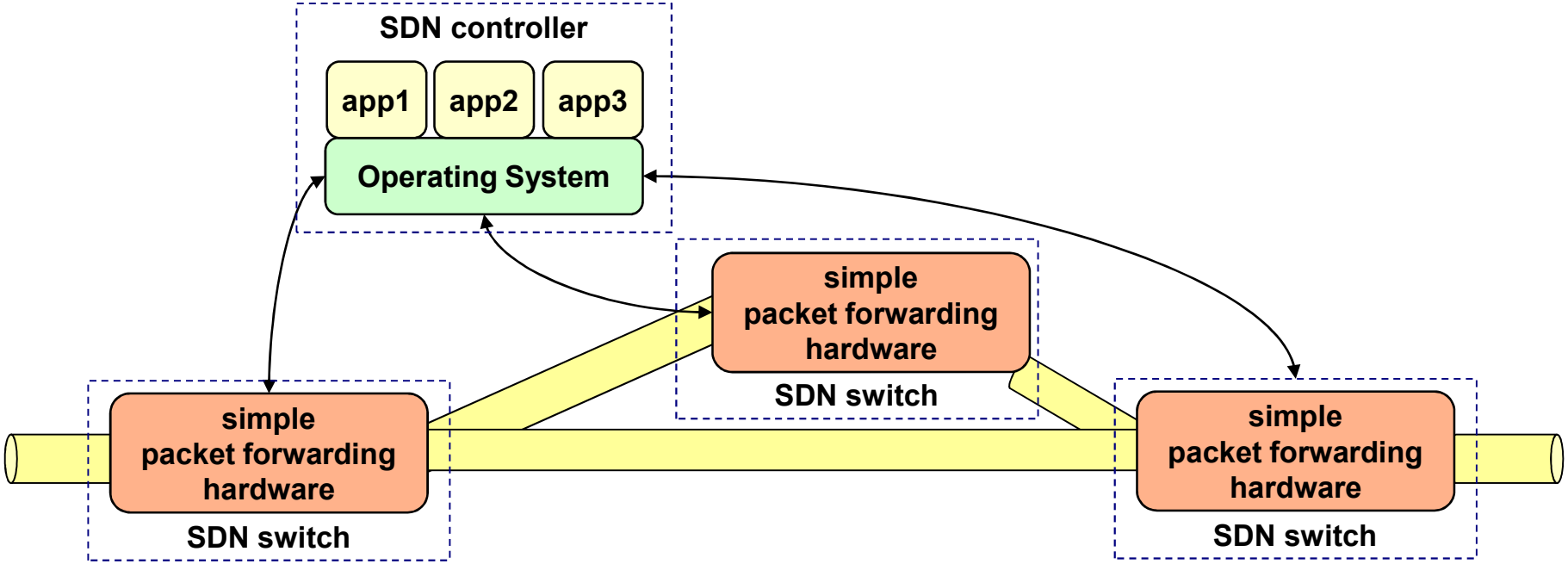
Classic Internet and the innovation problem



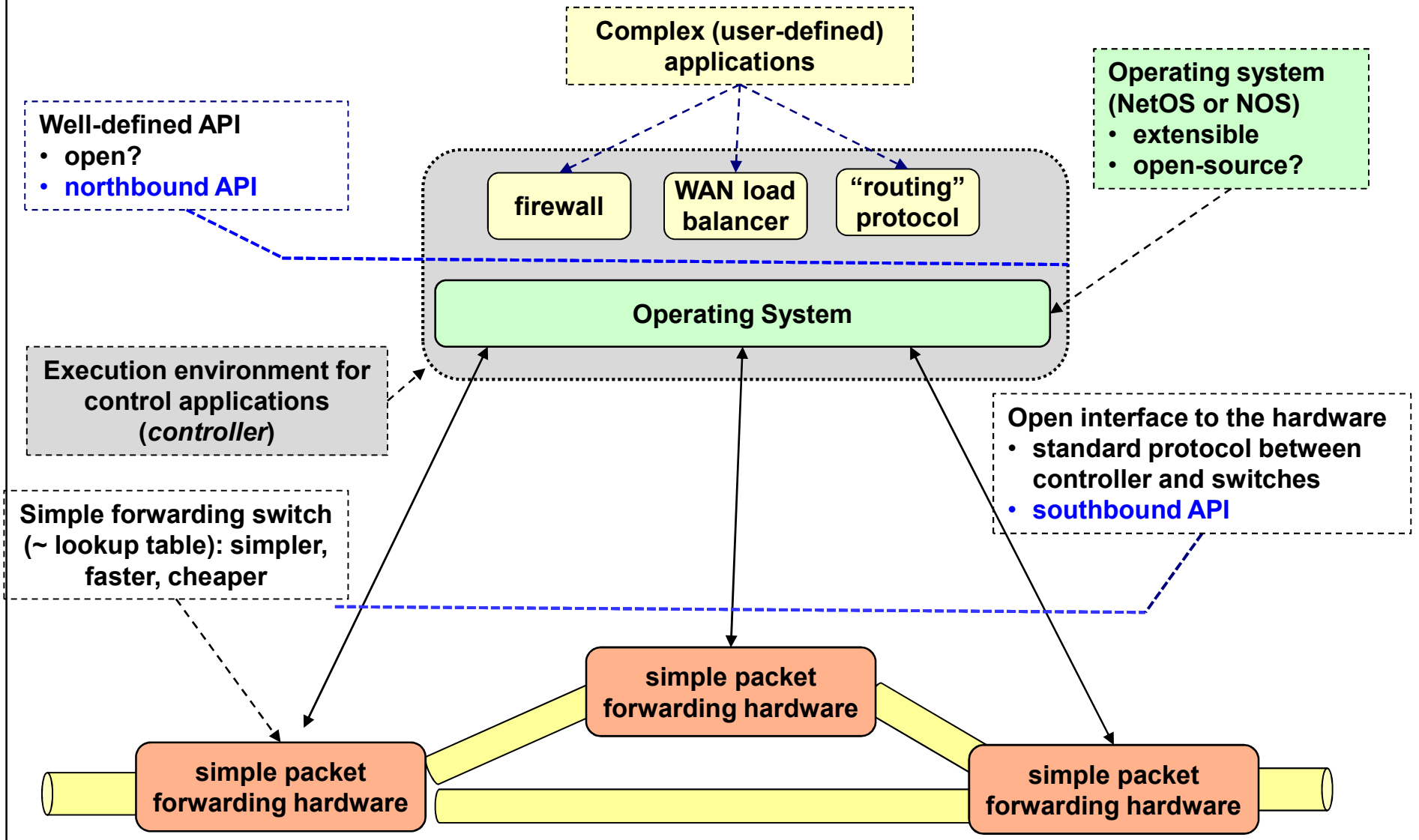
impossible (or very difficult)
to innovate in the infrastructure

Software Defined Network(ing)

- **introduces the ability to program the network itself (!)**
- **based on three pillars:**
  - separation of the control and forwarding functions
  - centralization of the control part
  - clear and well-defined interfaces (northbound, southbound)

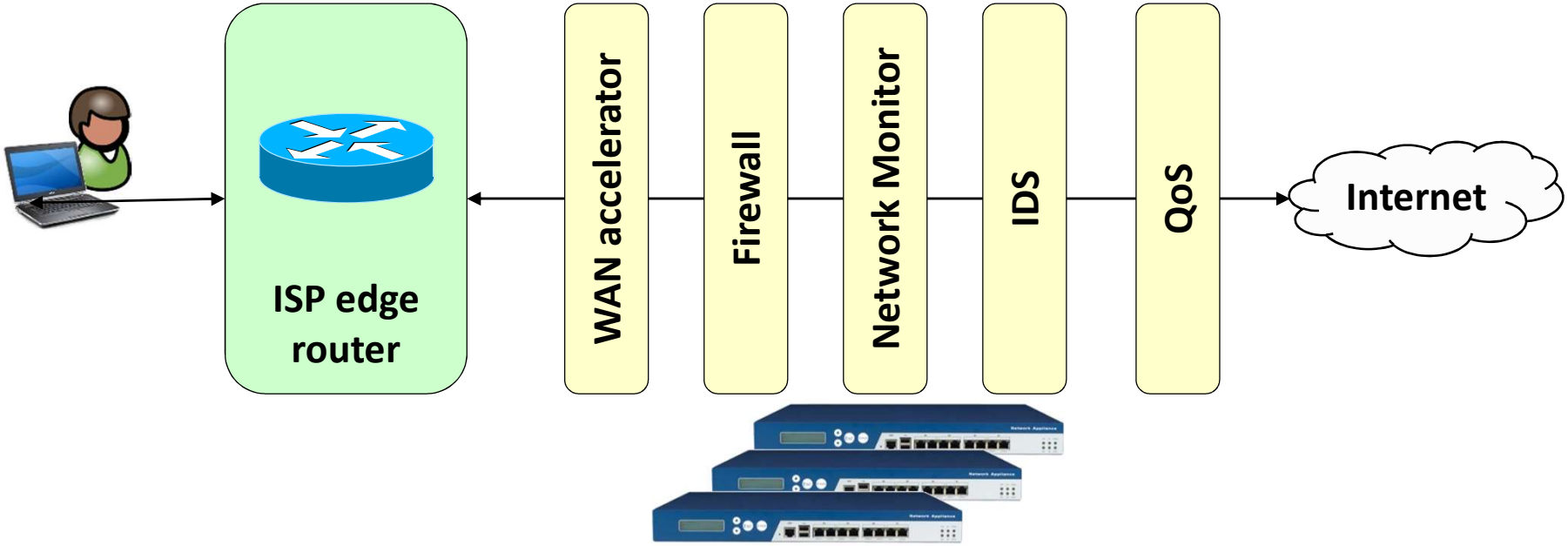


SDN components



Service Function Chaining

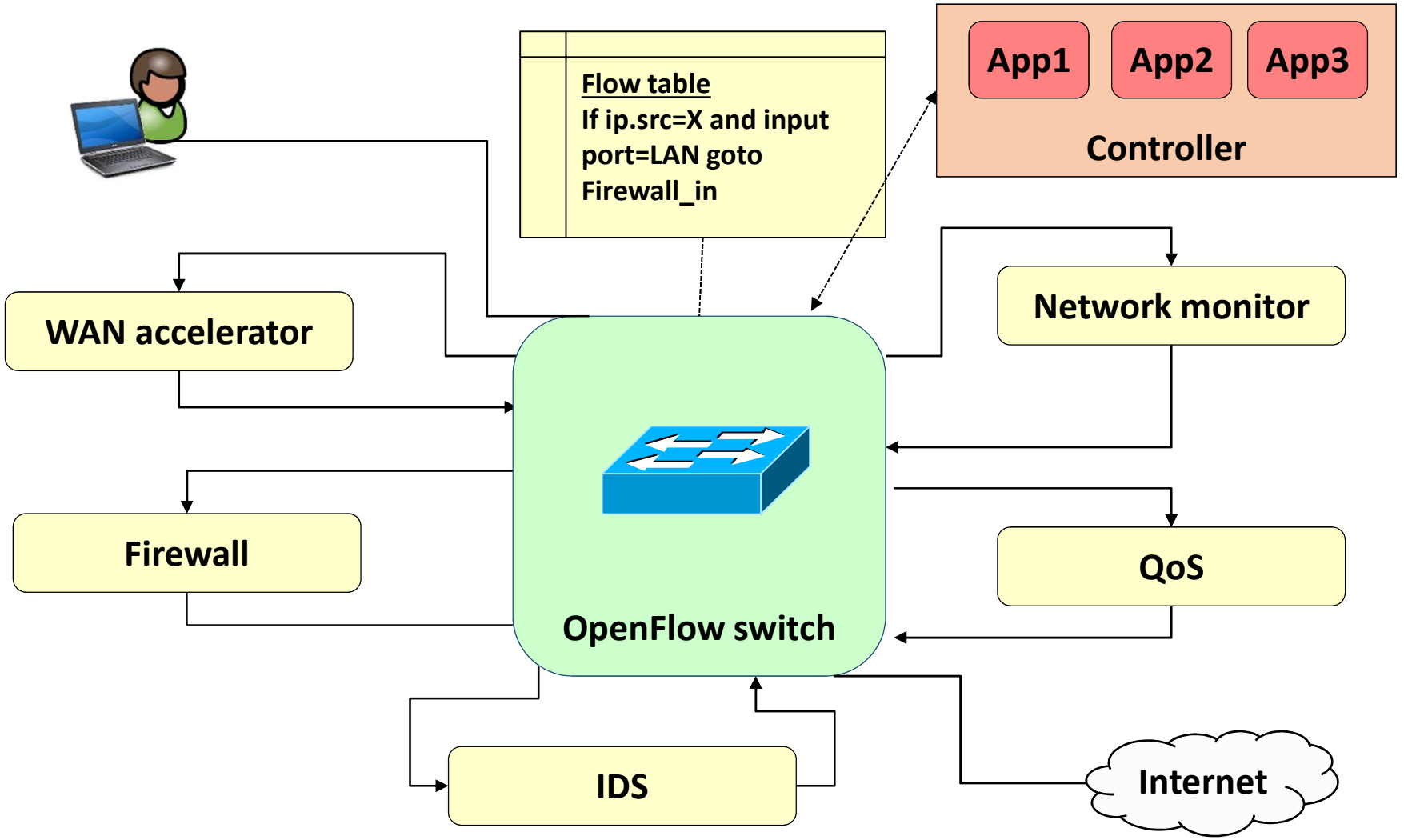
- often (particularly at the network edge) we need to chain different dedicated hardware appliances to provide added-value services
- this is what is called a chain of network functions



Several (practical) problems with SFC

- **hardware resources not used at best**
  - some appliances may sustain an heavy load, while other may be almost unloaded and we are not able to share the available hardware resources (e.g. CPU, memory) between different services
- **service disruption when modifying the service chain**
  - each time we add/remove a middlebox, we have to disrupt the service
- **not easy to differentiate services among tenants**
  - what if a tenant buys a “secure access to the Internet”, but others don’t? How can we avoid that the traffic of the second tenant goes through the FW as well?
    - the firewall must support explicit configuration of the user privileges (i.e. per-application configuration)

Service Function Chaining with SDN



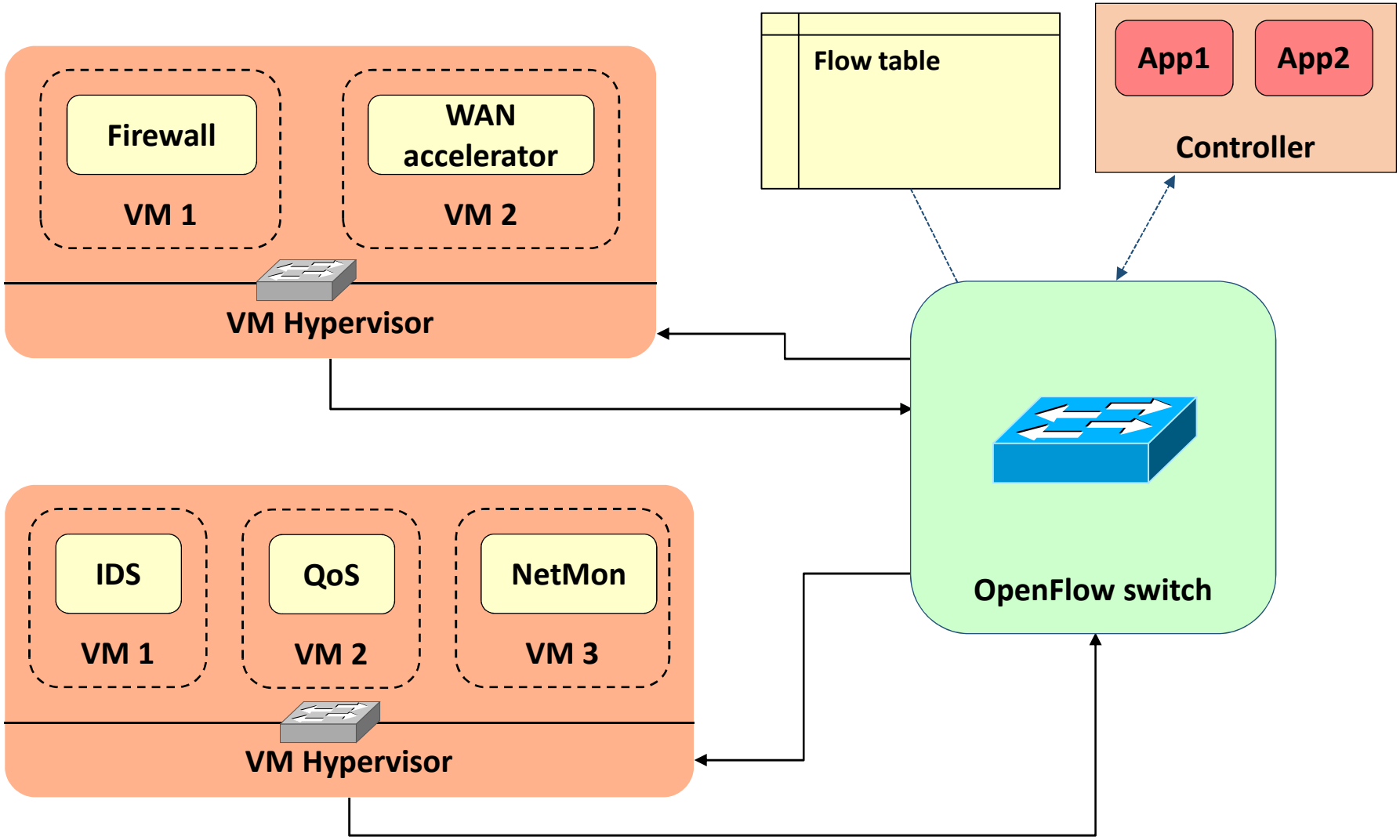
SFC with SDN: characteristics

- **agility in provisioning new services**
  - install the box, then “routing” is done via sw, instead of connecting the box to the others with physical wires
- **maintenance and reliability**
  - cabling is done once
- **different customers can have different service chains**
  - “routing” via software
  - possible to change routing decisions based on other parameters (e.g. application layer content)
- **still difficult to partition a physical appliance among different tenants**
  - many small business customers, each asking for a firewall service

Network Functions Virtualization

- **four main components:**
  - fast standard hardware (e.g. Intel servers)
    - Commercial-off-the-shelf (COTS) hardware
  - software-based network functions
    - network functions, previously running on a dedicated appliance, now become a software image, running on a standard server
  - computing virtualization (e.g. Linux KVM)
    - all virtualization pros (quick provisioning, scalability, mobility, reduced CapEx/OpEx, multitenancy, …)
  - standard API (i.e. ETSI framework)
- **NFV is the capability to run any network function on a standard hardware, possibly with computing virtualization to achieve an efficient use of resources**

Service functions chaining with NFV



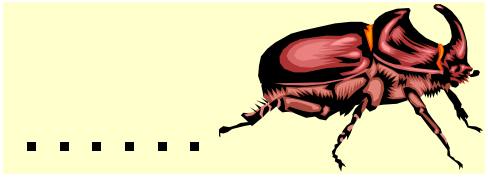
NFV and cloud

- **NFV can be seen as a way to bring network services in the world of cloud technologies**
  - cloud: hosts web servers, database servers, big data applications, etc.
  - NFV: adds also network services to that picture
- **although apparently NFV can be realized mostly with existing technologies, in practice:**
  - cloud frameworks may not support well traffic steering, although they support well traditional LAN services
  - network services are I/O intensive, while traditional cloud services are mostly CPU intensive
    - some technologies need to be tuned (and/or modified) to support the high amount of network traffic that is generated by network services

NFV and SDN

- **NFV is about computing, SDN is about network paths**
- **NFV requires SDN for flexible traffic steering**
  - although, a point-to-point Ethernet is often enough for most of the purposes
- **NFV and SDN are complementary**
  - one does not depend upon the other
  - you can do SDN only, NFV only, or SDN and NFV
- **a lot of discussions about SDN, not much debate about NFV**

Software bugs

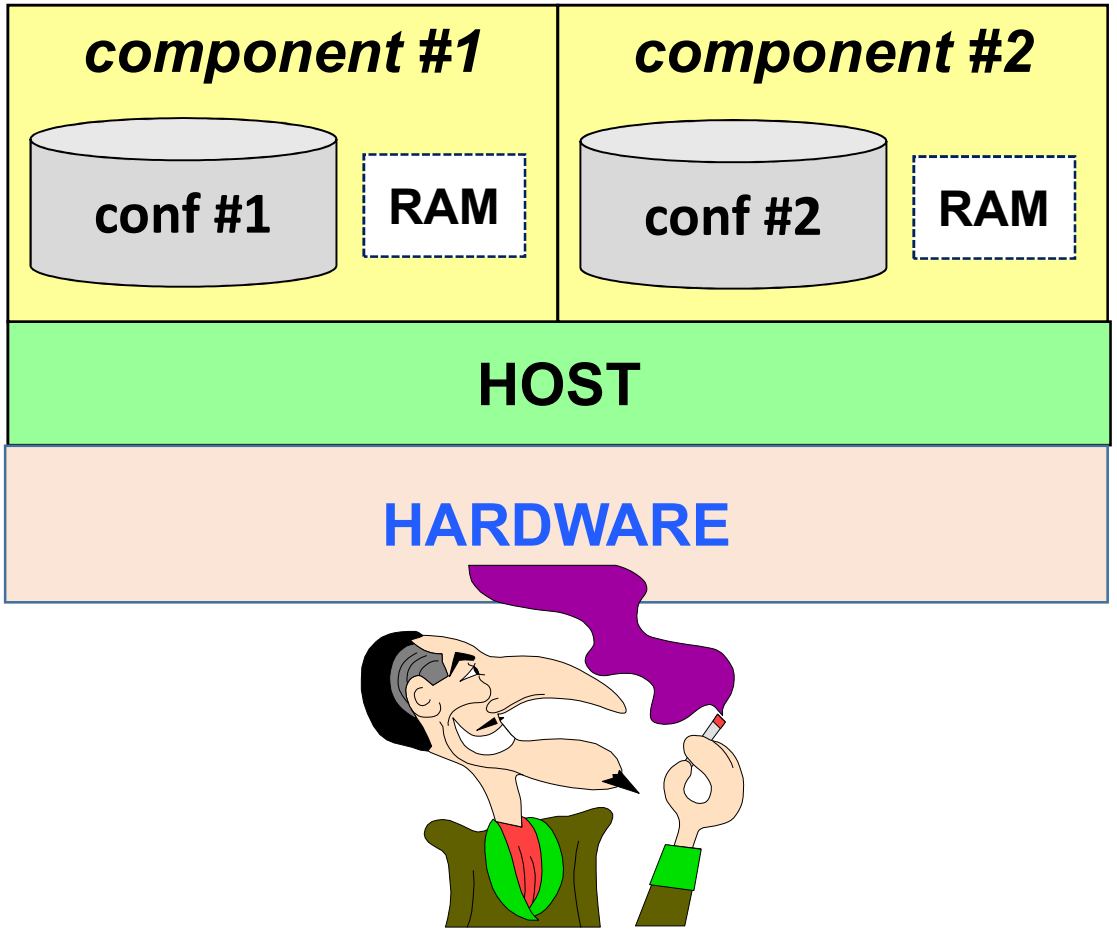


- **plenty of them ☺**
- **new bugs uncovered or introduced as a result of bug fixing ☺ ☺**
- **bad news:**
  - scarce attention to security in software development
  - complexity increases vulnerability
- **good news:**
  - situation is improving (but more awareness is needed)
  - (some) software modules are small enough that formal verification is possible
  - … but interactions and large components are still risky

SDN / NFV attack surface

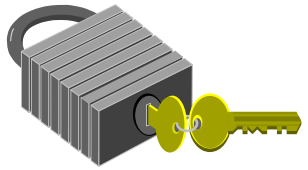
- management / control / data plane components
- software interfaces / APIs
- network channels
- host system, hypervisor, containers, …
- humans (network managers, hardware technicians, …)
- hardware bugs (!) and backdoors (!!)
- what / who can you trust?

Trust (and integrity)



Hardware root of trust

- **useful to have a stronger foundation (can still be attacked by physical access, unless made tamper-resistant/proof)**
- **platform-dependent or -independent**
  - TPM (Trusted Platform Module)
  - Intel TXT
  - AMD Secure Processor
  - ARM TrustZone
  - UEFI secure / trusted boot
- **important to create a TEE (Trusted Execution Environment)**
  - chain of trust (from firmware up to applications)



The security CIA triad

- **three main security properties:**
  - Confidentiality
  - Integrity
  - Availability
- **Trusted Computing (TC) focus on the first two, plus**
  - Privacy

Symmetric key cryptography

- **same key used by different parties**
  - encryption – decryption
- **usually fast operations**
- **challenge: key distribution**
- **main algorithms:**
  - AES, ChaCha20
  - HMAC

Asymmetric key cryptography

- **private/public key pair**
- **public key meant for large distribution**
  - encryption (for the receiver)
  - signature verification
- **private key must be kept secret**
  - decryption (by the receiver)
  - signature generation
- **usually slow**
- **main algorithms:**
  - RSA, ECC

Digests (hashes)

- **cryptographic one-way function**
  - arbitrary length input
  - fixed length output
  - easy to calculate
  - infeasible to reverse
  - resistant to collision (different data, same hash)
- **main algorithms:**
  - SHA-2, SHA-3
  - HMAC (for symmetric "signature" i.e. keyed-digest)

Authenticity is more than signature

- **we want to make sure information is fresh:**
  - timestamps
    - but they need to be secure
  - nonces
    - fresh random large number
    - must be unpredictable to the attacker

Introduction to Trusted Computing

What is Trusted Computing?

- **a trusted component/platform is a component/platform that behaves as expected**
- **trust is not the same as good/secure**
  - the behaviour needs to be verified against the expected behaviour
- **attestation**
  - presentation of verifiable evidence of the platform's state
- **Root of Trust**
  - inherently trusted component

What is Trusted Computing? (cont.)

- **Trusted Computing defines schemes for establishing trust in a platform that are based on identifying its hardware and software components**
- **The Trusted Platform Module (TPM) provides methods for collecting and reporting these identities**
  - TPM can be a physical chip, a portion of an IC, or a piece of software (typically firmware)
  - different implementations offer different levels of trust, based on attacker and trust models
- **a TPM used in a computer system reports on the hardware and software in a way that allows determination of expected behaviour and, from that expectation, establishment of trust**

Trusted Computing Base (TCB)

- **collection of system resources (hardware and software) that is responsible for maintaining the security policy of the system**
  - an important attribute of a TCB is that it be able to prevent itself from being compromised by any hardware or software that is not part of the TCB.
- **the TPM is not the trusted computing base of a system, rather, a TPM is a component that allows an independent entity to determine if the TCB has been compromised**
  - in some uses, the TPM can help prevent the system from starting if the TCB cannot be properly instantiated

Root of Trust

- **a component that must always behave in the expected manner because its misbehaviour cannot be detected**
  - building blocks for establishing trust in a platform
- **Root of Trust for Measurement (RTM)**
  - measure and send integrity measurement to RTS
  - usually the CPU executes the CRTM (Core Root of Trust for Measurement)
- **Root of Trust for Storage (RTS)**
  - shielded/secured storage
- **Root of Trust for Reporting (RTR)**
  - entity that securely reports the content of RTS
- **the TPM chip is both RTS and RTR, appropriate firmware is RTM**

Measurements

- **TPM used:**
  - to store keys (identity, encryption)
  - to store critical values
  - to report values in a trustworthy manner
    - signature over value+challenge
- **values stored in PCR (Platform Configuration Registers) that act as accumulators with the measure-and-extend operation**
  - M = hash(component)
  - PCR.new = hash(PCR.old || M)
  - exec(component)
- **if execution sequence is fixed (e.g. boot) the final value is predictable and can be easily checked, else we need also the list of executed components**

Chain of trust

- **component A measures component B**
  - stores the measurement in RTS
- **component B measures component C**
  - stores the measurement in RTS

- **using RTR, a verifier can securely retrieve B's and C's measurements from the RTS**
  - B and C can only be trusted if A is trustworthy

Primer on TPM and attestation

Trusted Platform Module (TPM) overview

- **inexpensive (< $1)**
  - available on most servers, laptop, PC
- **tamper-resistant (i.e. difficult to tamper)**
  - but not tamper-proof (i.e. impossible to tamper)
- **crypto-engine (slow) + protected storage (limited, but can be extended externally)**
  - + sealing (!) = keys can be used only when the system is in a certain "state"
- **certified Common Criteria EAL4+**
- **“passive component”**
  - actually it's an active one
  - … but needs to be driven by the CPU
  - cannot prevent boot, but can protect data
- **two main versions: 1.2 and 2.0**

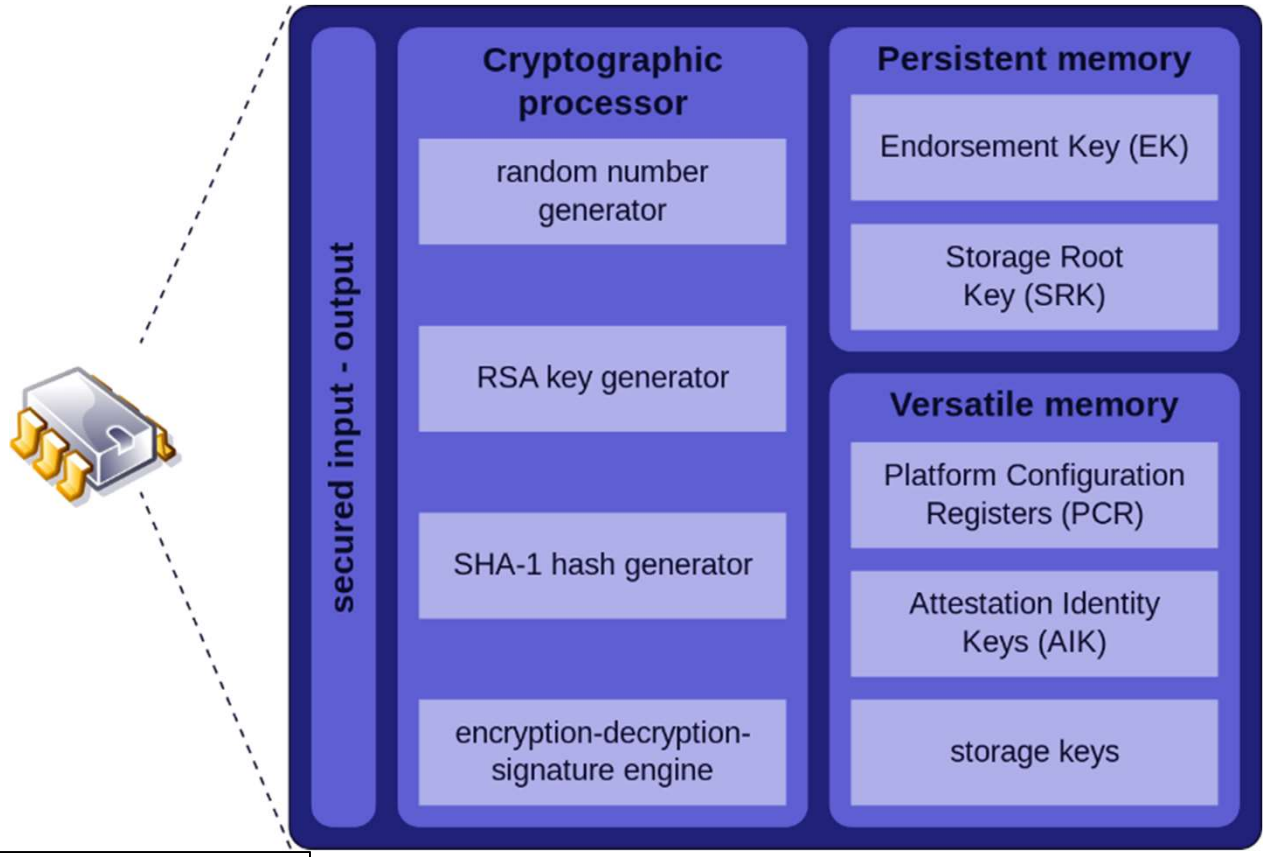
TPM alternative implementations

- **software (!) simulator**
  - in firmware (required before boot)
  - in the OS (only for the applications)
  - implemented in a co-processor (e.g. Intel SGX)
- **embedded in the CPU**

- **baseline: who / what do YOU trust?**

TPM 1.2 functionalities overview

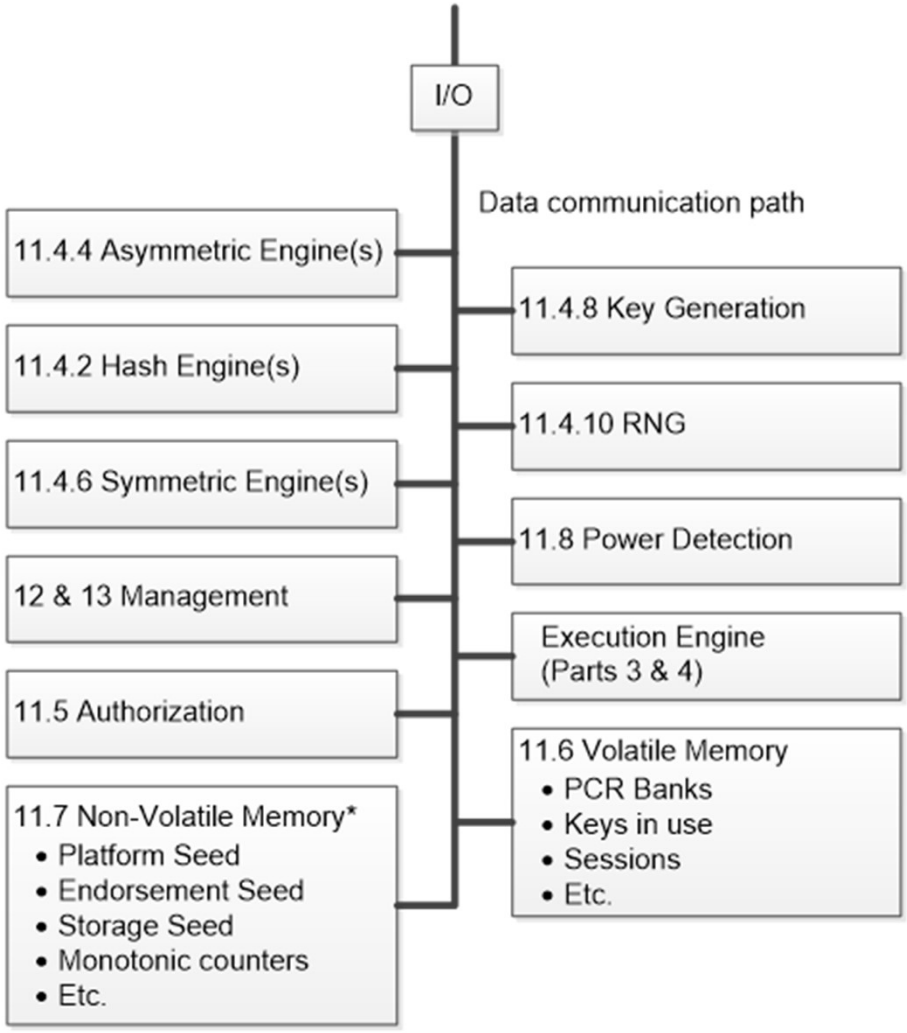
- fixed set of algorithms
- one storage hierarchy for platform user
- sealing to PCR value



Source Wikipedia
Author: Eusebius (Guillaume Piolle)

TPM 2.0 functionalities overview

- cryptographic agility
- three key hierarchies
- policy-based authZ (generalized sealing)



Source TCG specification

* NV memory may be provided by a system chip with the data going to/from NV in a protected form. What is kept in the "TPM" in that case is a cached copy of the NV contents.

Using a TPM for a platform's identity

- **cryptographically strong and hardware-based/protected identities**
- **the TPM is designed to create secure identities**
  - identity = private key of an asymmetric key pair
- **the TPM can hold the private key, making it tamper-resistant to host attacks (trojan, side-channel, etc.)**
  - it's important to restrict access/use of the private key
    - TPM's authorization mechanisms (password, policy)
- **the public key of an identity is usually certified, creating a chain of trust rooted in a trusted CA**
  - platform's identity + public key certificate = platform's credentials

TPM 2.0 three hierarchies

- **platform hierarchy**
  - for platform's firmware (for the manufacturer)
  - NV storage, keys, and data
- **endorsement hierarchy**
  - for the privacy administrator
  - keys and data
- **storage hierarchy**
  - for the platform's owner (usually also the privacy administrator)
  - NV storage, keys and data
- **each hierarchy has:**
  - dedicate Authorization (password) and Policy
  - specific seed for generating the primary keys

Ephemeral hierarchy

- **also named the "null" hierarchy**
- **useful when the TPM is used as crypto co-processor**
- **deleted every time the TPM goes through a power cycle**

Using a TPM for securely storing data

- **physical isolation**
  - storage in the TPM
    - Non-Volatile (NV) RAM
    - primary keys
    - permanent keys
  - very limited space
  - Mandatory Access Control

Using a TPM for securely storing data

- **cryptographic isolation**
  - storage outside of the TPM
    - platform HDD/SSD
    - keys or data
    - BEWARE! blob needs to be protected
  - encrypted by the TPM
  - Mandatory Access Control

TPM objects

- **primary keys:**
  - endorsement keys, storage keys
  - derived from one of the primary seeds
    - the TPM does not return the private value
  - can be re-created by using the same parameters
    - assuming the primary seed has not been changed
- **keys & Sealed Data Objects:**
  - protected by a Storage Parent key
    - storage parent key needed in the TPM to load/create a key/SDO
  - randomness comes from the TPM RNG
  - the TPM returns the private part – protected by the storage parent key
    - the private part needs to be stored somewhere

TPM object's area

- **public area**
  - use to uniquely identify an object
- **private area**
  - object's secrets
  - only exists in the TPM
- **sensitive area**
  - encrypted private area
  - use for storage outside of the TPM

TPM's object public area (and names)

- object's type
- hash algorithm used for the name
- object's attributes
- authorisation policy
- parameters (e.g. key size for RSA)
- unique (e.g. public key for a asymmetric key pair)

- Object's Name = nameHashAlg || Hash(public area)
- Qualified Name (QN) = Hash(QN_Parent || Object's Name)
- these names are important because they are used in the HMAC computation for authorization and this prevents substitution attacks (performed by a MITM at external operations)

TPM object's attributes

Usage (for a key K):

- **sign: K can be used for generating a signature**
- **decrypt: K can be used for decrypting ciphertext**
- **restricted: K can only operate on TPM internal data**

sign	*decrypt*	*restricted*	**Quick description**
0	0	0	A data blob. Can be accessed using TPM2_Unseal().
0	0	1	Not allowed. The TPM will not load or create an object with this setting.
0	1	0	A decryptinng key, but not a storage key.
0	1	1	A storage key.
1	0	0	A signing key.
1	0	1	A key for signing TPM generated data (e.g. PCR quote).
1	1	0	A general-purpose key, but not a Storage Key.
1	1	1	Not allowed. No useful purpose and incompatible with FIPS specifications

TPM object's attributes (continued)

- **Authorization:**
  - userWithAuth: User role can be done with the authValue of the object
  - adminWithPolicy: Admin role must fulfill authPolicy
  - noDA: dictionary attack protection disabled
- **Duplication:**
  - fixedParent: the object can not be duplicated under another storage parent
    - note: one of the parents could be duplicated though
  - fixedTPM: the object can not be used outside of this TPM that created the object
  - encryptedDuplication: controls if the private part can be extracted in plaintext

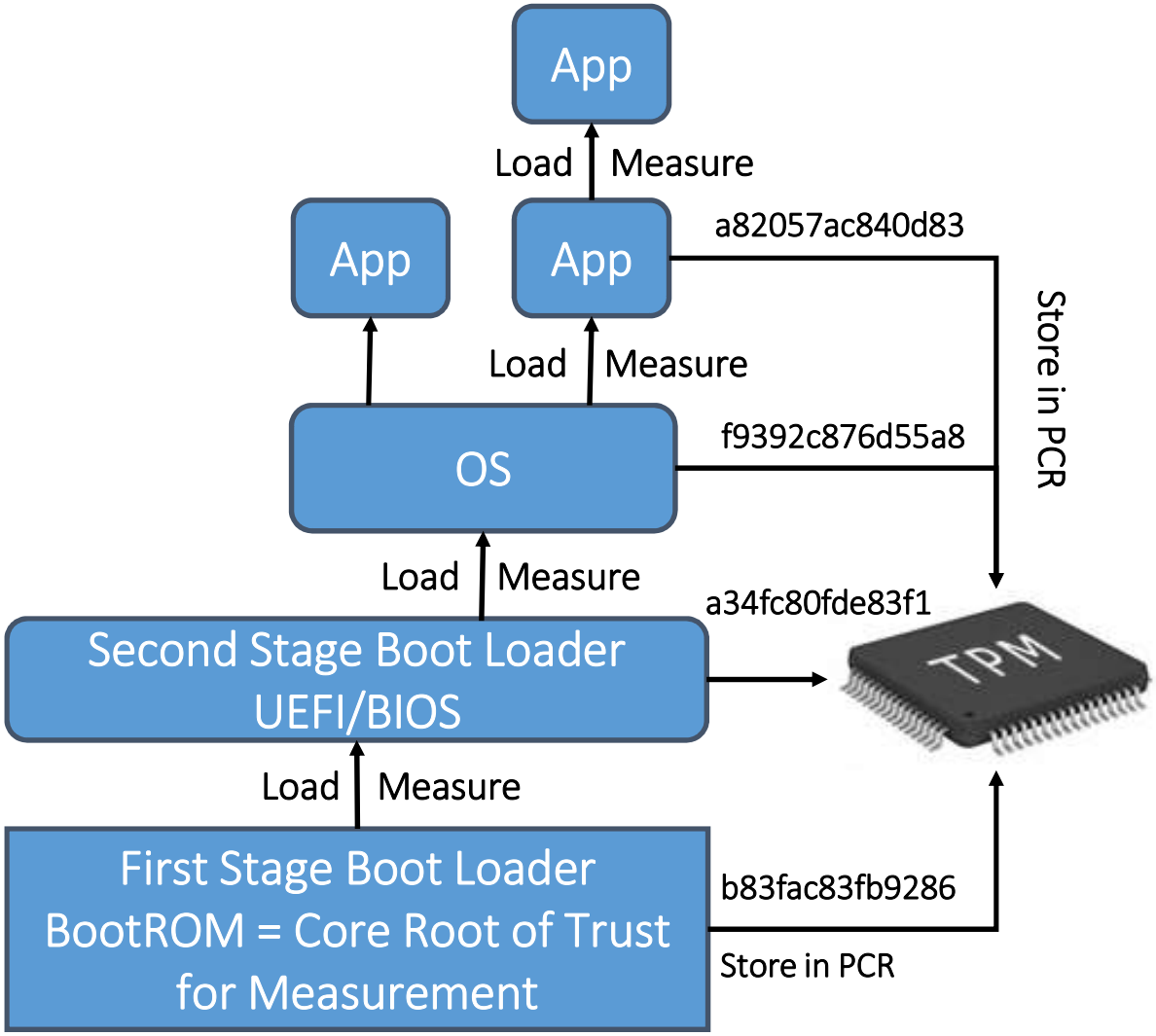
TPM object's attributes (continued)

- **Creation:**
  - sensitiveDataOrigin: for symmetric keys, specifies if the TPM – or the caller – generates the key
- **Persistence:**
  - stClear: the object needs to be reloaded after a TPM reset or restart (the object can not be made persistent)

TPM Platform Configuration Register (PCR)

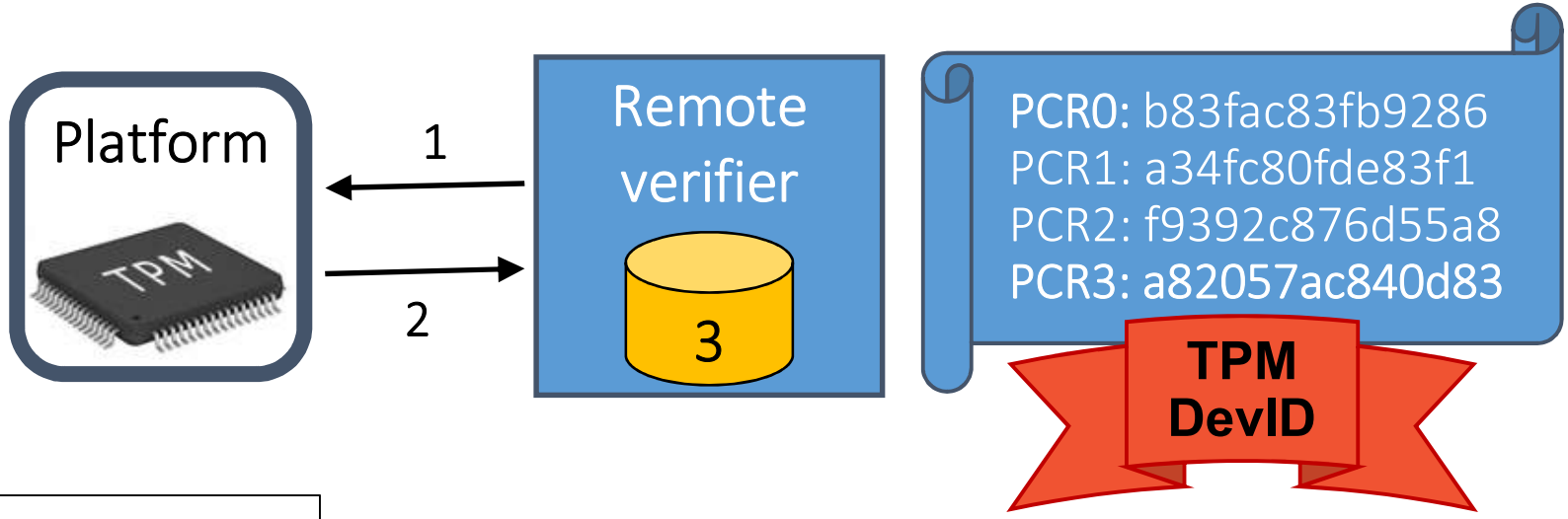
- **core mechanism for recording integrity of the platform**
  - only reset at platform reset (or with hardware signal)
  - malicious code cannot take its measurement back
- **PCRs are extended using a cumulative hash**
  - D = hash(something_to_be_protected)
  - PCR.new = hash(PCR.old || D)
- **can be used to gate access to other TPM objects**
  - e.g. BitLocker seals disk encryption keys to PCR values

Measured boot using TPM

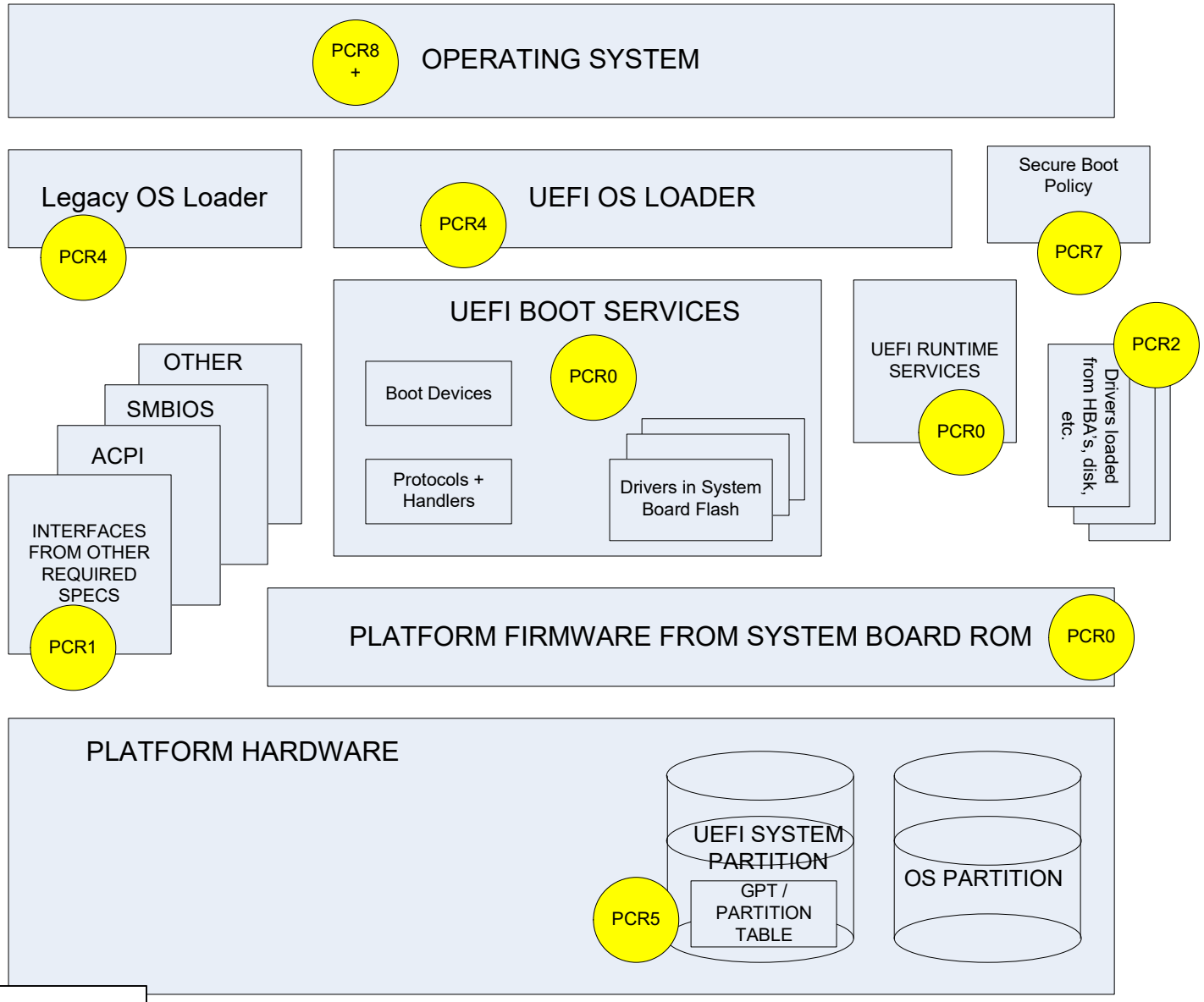


Remote attestation using TPM

1. **Challenge (nonce)**
2. **Signed measurements**
3. **Verification**
   - **validate signature**
   - **check measurements against Reference Measurements (golden values)**



TCG PC Client PCR utilization (architecture)



TCG PC Client PCR usage (detail allocation)

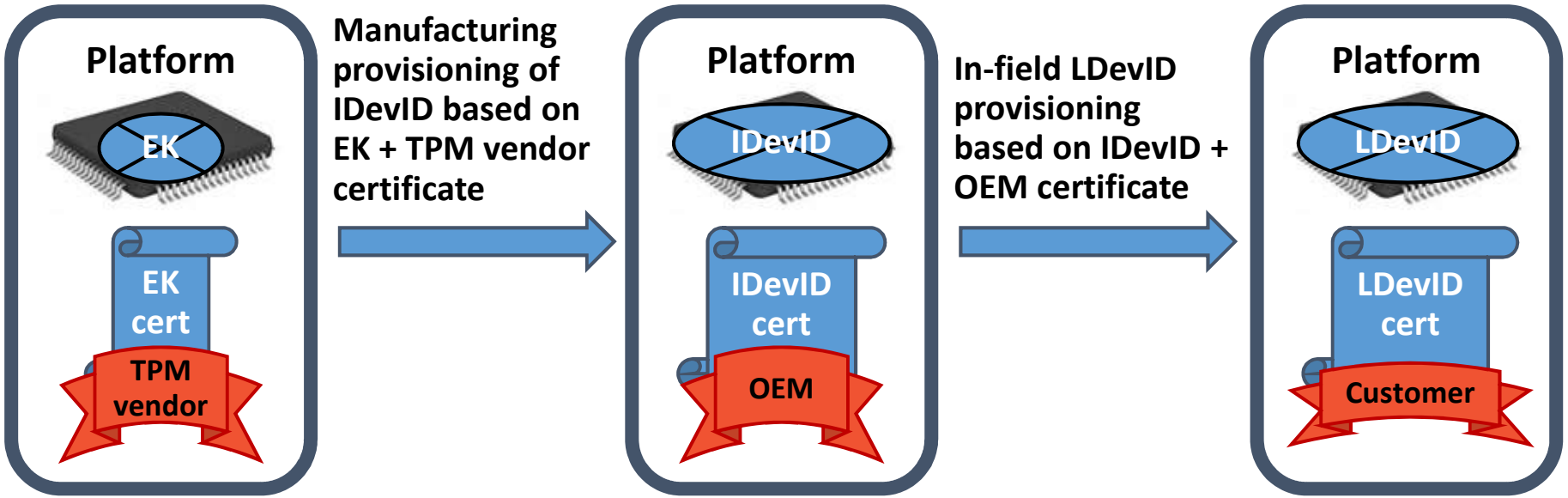
PCR Index	PCR Usage
0	SRTM, BIOS, Host Platform Extensions, Embedded Option ROMs and PI Drivers
1	Host Platform Configuration
2	UEFI driver and application Code
3	UEFI driver and application Configuration and Data
4	UEFI Boot Manager Code (usually the MBR) and Boot Attempts
5	Boot Manager Code Configuration and Data (for use by the Boot Manager Code) and GPT/Partition Table
6	Host Platform Manufacturer Specific
7	Secure Boot Policy
8-15	Defined for use by the Static OS
16	Debug
23	Application Support

Dynamic Root of Trust for Measurement (DRTM)

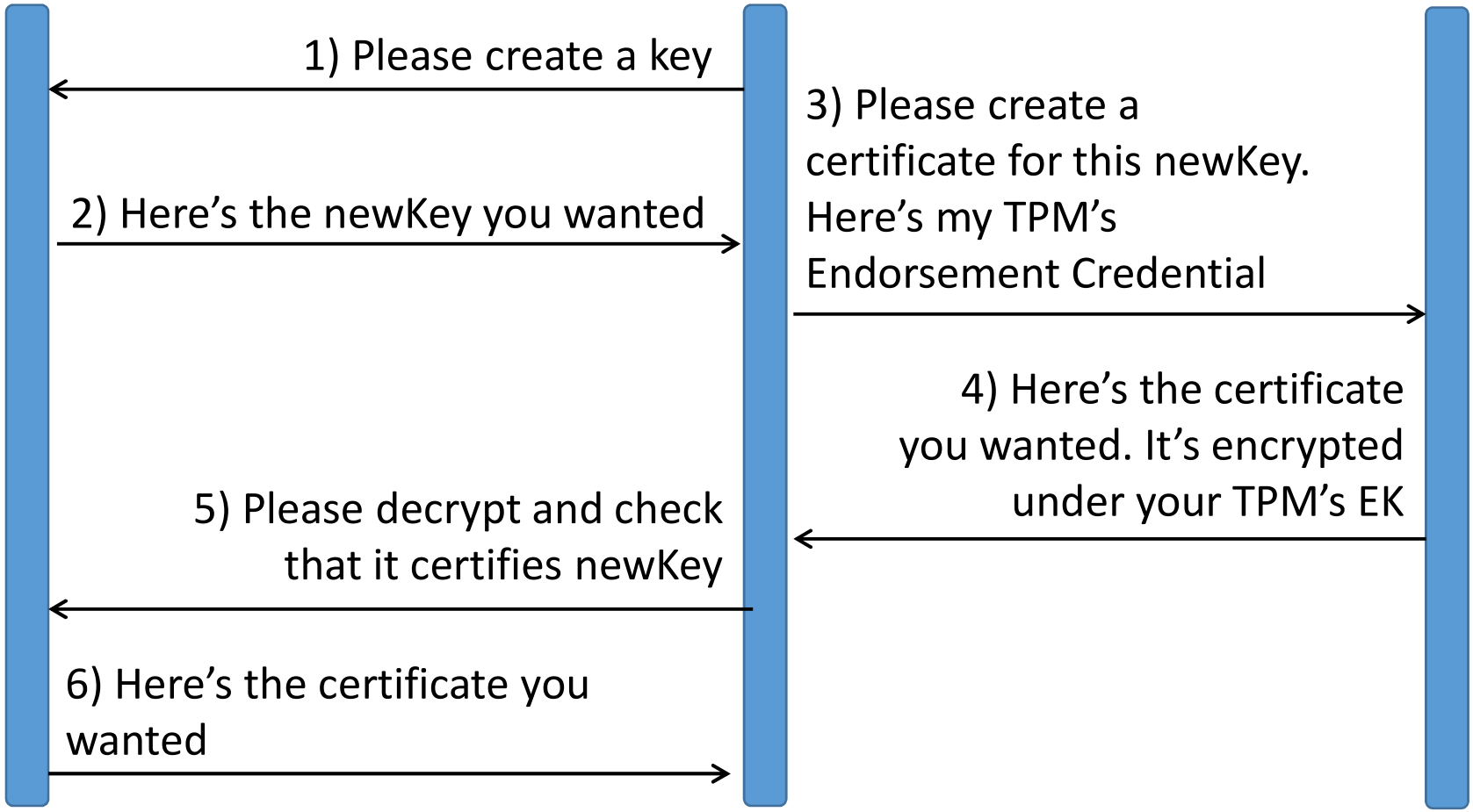
- **special processor command**
  - SINIT (Intel TXT) or SKINIT (AMD SVM)
- **stops all processing on the platform**
- **DRTM hashes contents of memory region**
  - stores measurement in dynamic PCR
- **transfer control to specified location in memory**
- **also called Late Launch**

Credentials chain of trust

- **from the TPM vendor to a customer-usable certificate**
- **IEEE 802.1AR – Secure Device Identity using TPM**
  - allows Zero-Touch management of a platform



TPM 2.0 Make/Activate Credentials



TPM basic authorization mechanism

- **password-like authorization**
- **the password can be protected using HMAC sessions**
  - coupled with asymmetric encryption of salt
- **but the TPM knows a lot about the platform states and can be configured:**
  - prevent object usage unless selected PCRs have specific values
  - prevent object usage after a specific time
  - prevent object usage unless authorized by multiple entities (i.e. key holders)

TPM 2.0 enhanced authorisation policies

- **a policy can be written as an equation using AND and OR, based on atomic policies**

  (A & B & C) | (D & E & F)

- **AND is based on cumulative hash (similar to PCR extend)**
  - Digest_left = H(H(H(0||A)||B)||C)
  - Digest_right = H(H(H(0||D)||E)||F)
- **BEWARE: the order of the atomic policies matters!**
  - (A & B & C) != (C & B & A)
- **OR is the hash all "input" policies**
  - only one of them needs to be matched
- **the final hash of the authorisation policy is in the public area**

TPM 2.0 atomic policy (examples)

- **TPM2_PolicyPCR: selected PCRs need to match a given value**
- **TPM2_PolicyTicket: check that a valid ticket is signed by a given entity (i.e. holder of a given private key)**
- **TPM2_PolicyNV: selected NV index should match a given value**
- **TPM2_PolicyCC: restrict a policy to a given TPM2 command**
- **TPM2_PolicyCpHash: restrict to given parameters**
- **But a object's policy is part of the public area!**
- **TPM2_PolicyAuthorized: allow an entity to sign a policy**
  - any policy signed by the entity – and being validated – is valid

Getting started with TPM 2.0

- **Microsoft TPM2.0 reference implementation**
  - Software simulator
  - https://github.com/Microsoft/ms-tpm-20-ref
- **Intel et al. Trusted Software Stack for TPM2.0**
  - https://github.com/tpm2-software/tpm2-tss
- **associated TPM2.0 TSS tools**
  - Script friendly tools
  - https://github.com/tpm2-software/tpm2-tools
- **IBM has open-source software**
  - software TPM, pseudo-TSS, etc.
- **free book by Springer**
  - http://dx.doi.org/10.1007/978-1-4302-6584-9

Linux's Integrity Measurement Architecture (IMA)

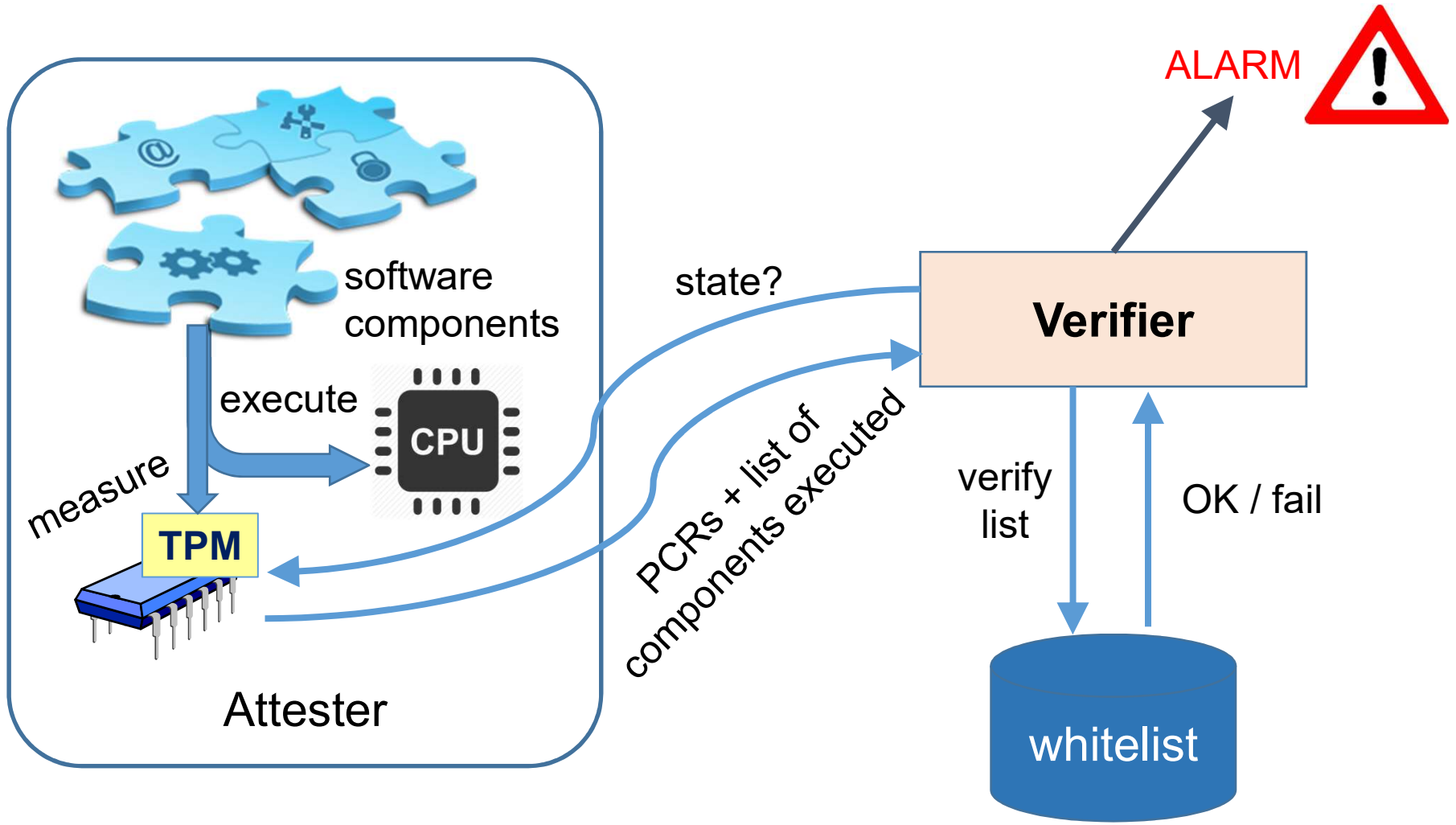
- **standard Linux module**
- **Collect**
  - measure a file before it is accessed (for read, execute, existence, …)
- **Store**
  - add the measurement to a kernel-resident list
  - extend the IMA PCR with this measurement
- **Appraise**
  - enforce local validation of a measurement against a "good" value stored in an extended attribute of the file
- **Protect**
  - protect a file's security extended attributes (including appraisal hash) against off-line attack

Linux IMA details

- **extension of UEFI measured boot to the OS and apps**
  - Linux IMA uses PCR10
  - 1st measurement: boot_aggregate
    - hash of TPM's PCR 0 to 7 (i.e. UEFI-related PCRs)
- **measurement configuration through IMA template**
  - mostly ima-ng, but can be customized
- **exposed in the kernel's securityfs**
  - /sys/kernel/security/ima/ascii_runtime_measurements

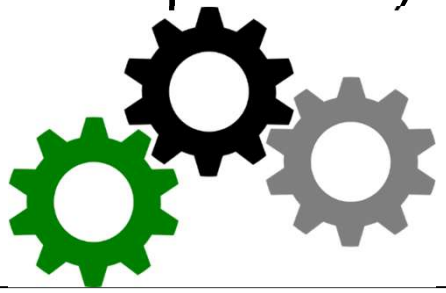
PCR	template-hash	template	filedata-hash	filename-hint
10	91f34b5[…]ab1e127	ima-ng	sha1:1801e1b[…]4eaf6b3	boot_aggregate
10	8b16832[…]e86486a	ima-ng	sha256:efdd249[…]b689954	/init
10	ed893b1[…]e71e4af	ima-ng	sha256:1fd312a[…]6a6a524	/usr/lib64/ld-2.16.so
10	9051e8e[…]4ca432b	ima-ng	sha256:3d35533[…]efd84b8	/etc/ld.so.cache

Remote attestation with TPM+IMA



Which is your trust perimeter?

- **download time**
  - check signature
- **load time**
  - measure components when loaded for execution
    - what is "executable"?
- **run time (components that change their behaviour while running)**
  - measure configuration files (when loaded or re-loaded)
    - beware of caching!
  - measure in-memory configuration (e.g. filtering or forwarding rules modified by CLI or network protocol)
    - needs appropriate firmware/host



What about virtualization?

- **VM vs. containers, OS-level vs. hw-level virtualization, virtualization inside virtualization, …**
- **problems for integrity monitoring of virtualized components:**
  - hardware root of trust
  - virtual boot
  - inspection of actions inside the component

Integrity monitoring in v-environments

- **containers better than VMM for monitoring**
  - same kernel as host
  - hence container operations can be monitored by the host
- **remote attestation for containers**
  - extended IMA (Integrity Measurement Architecture)
- **CoreOS trusted container**
  - store container state and configuration into TPM event log
  - does not cover modules executed inside container
- **Intel Clear containers**
  - uses Intel VT to run Container as VM w/ better performance
  - only load-time integrity of container image

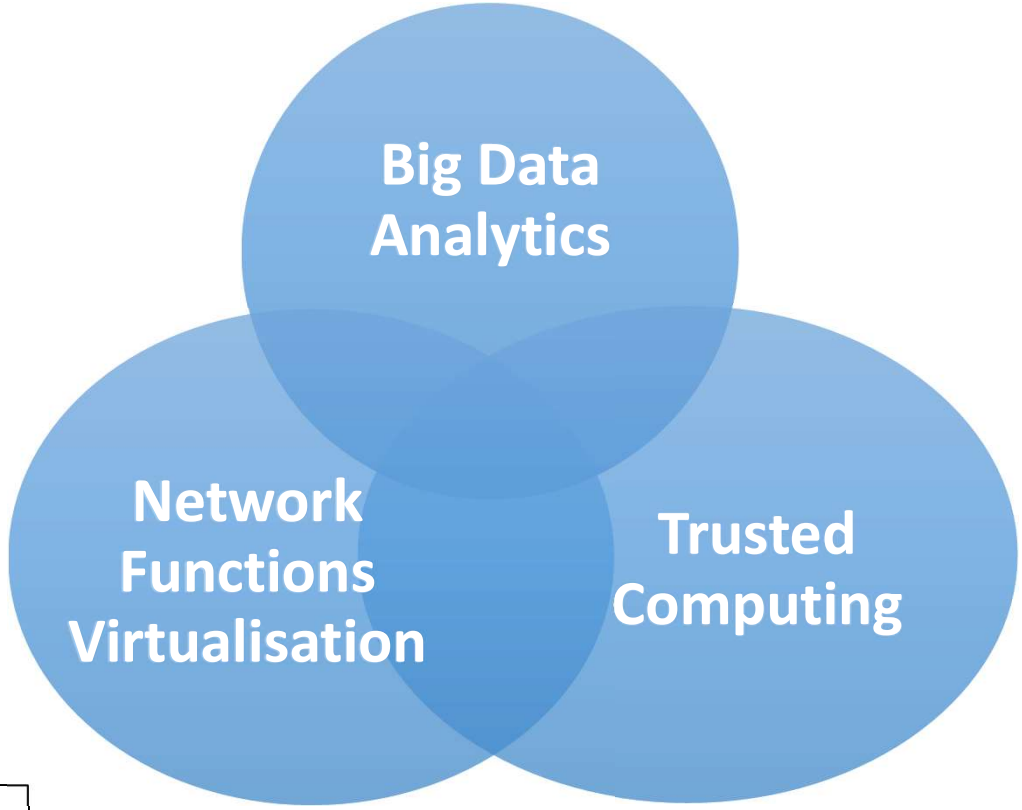
Audit and forensic analysis

- **node (e.g. ECU) / network behaviour cannot be given for granted any more**
- **increasingly important as more intelligence / computation is moved into the vehicle / network**
- **open questions:**
  - state at time T?
  - network path+processing for user U at time T?

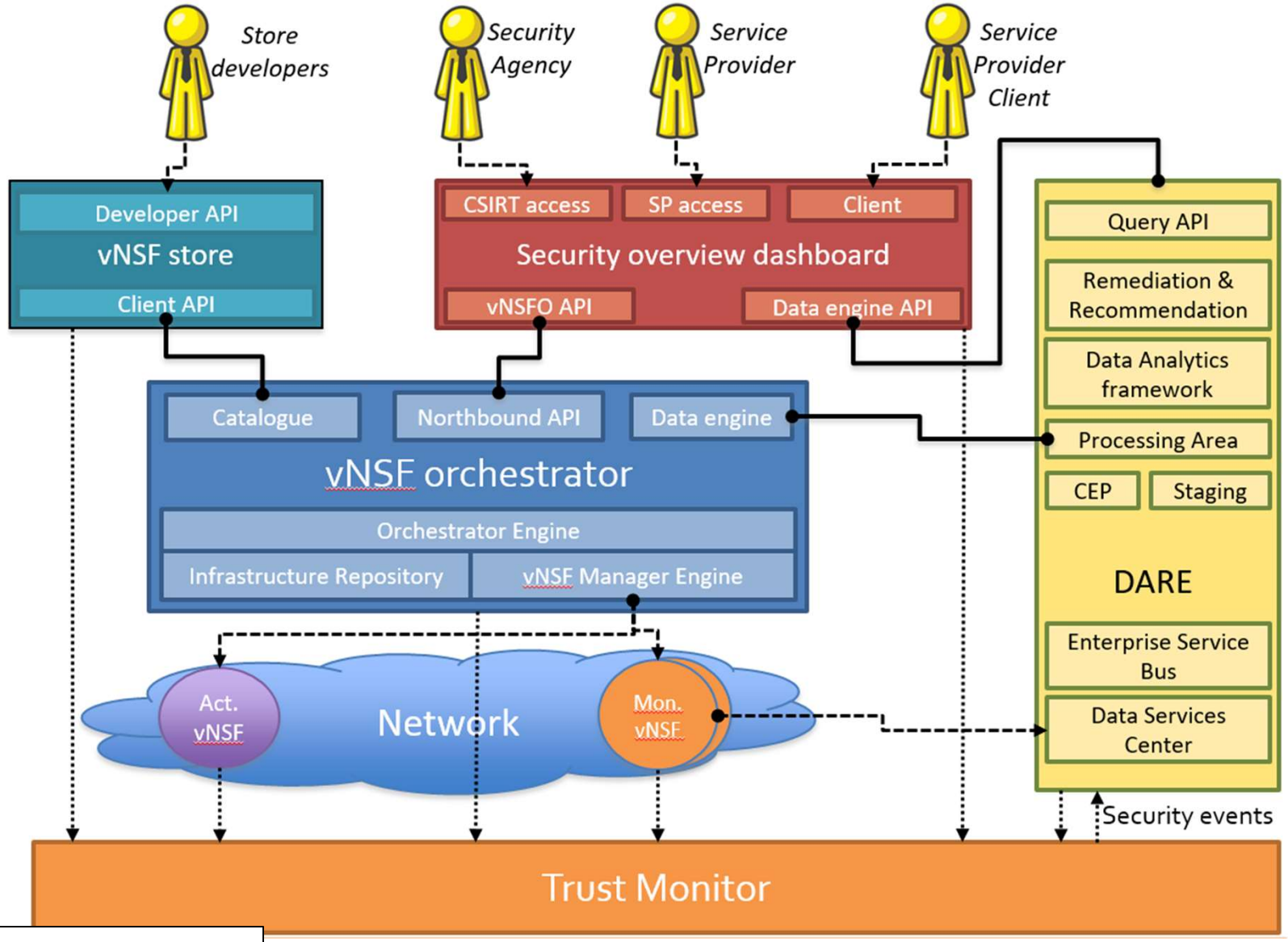
SHIELD design and architecture

SHIELD project mission

SHIELD aims to deliver an open solution for dynamically establishing and deploying virtual security infrastructures in ISP and corporate networks.



SHIELD architecture

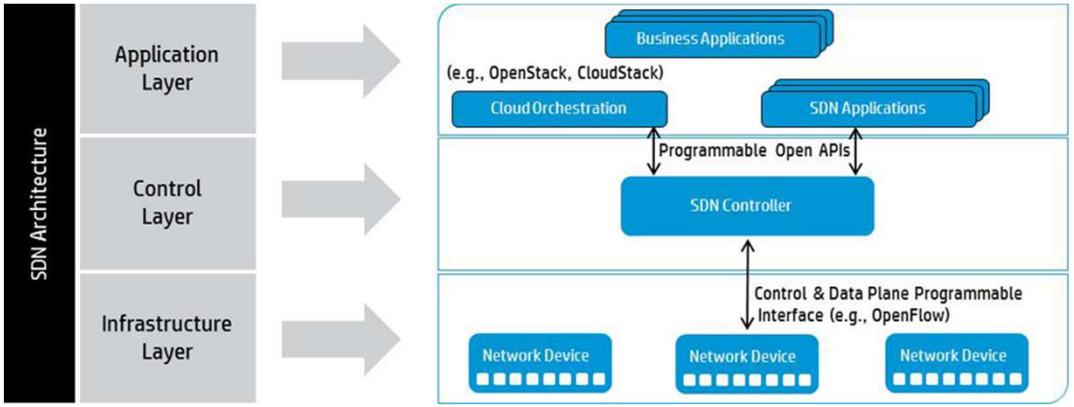


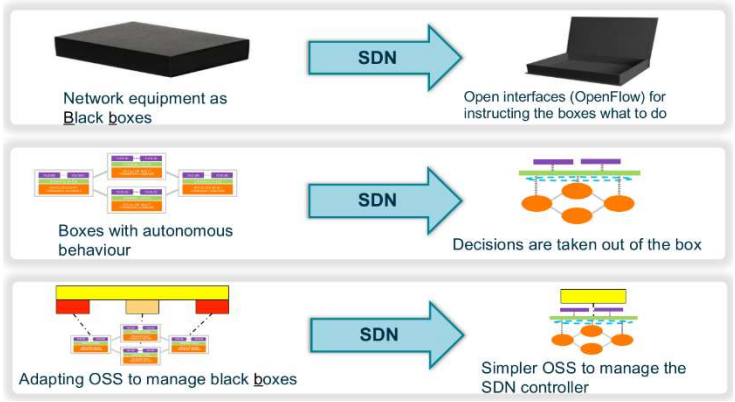
The virtualization gap

- **virtual security functions alone cannot be verified, evaluated.**
  - the state is remotely controlled by the orchestration
- **how to verify if a virtual function is working as intended?**
  - securely
  - automatically

The network data/control separation gap

- **the network is also dynamic**
  - use of the Software-Defined Network paradigm.
- **impossible to locally assess the SDN configuration.**
- **but the controller has full visibility of the network topology!**





SDN threat assessment

T1: Modification of control plane packs (from NE to the controller, i.e. table miss).

T2: Modification of control plane packs (from the controller to NE, i.e. rule update).

T3: Passive eavesdropping of the control plane.

- **TLS tunnel between SDN control and NEs**

T4: Malicious or faulty administration of network elements.

T5: Zero-day exploitation of network element firmware vulnerabilities

- **code analysis**

SDN threat assessment (cont.)

T6: Rule override by an SDN application

- **hardened SDN controller (policy on new rule creation)**

T7: Flashing of NE firmware (malicious software, persistent bootkits)

T8: Physical attacks (chip replacement, bus probing)

- **out of scope**

T9: Rogue SDN controller that alters configurations of network elements

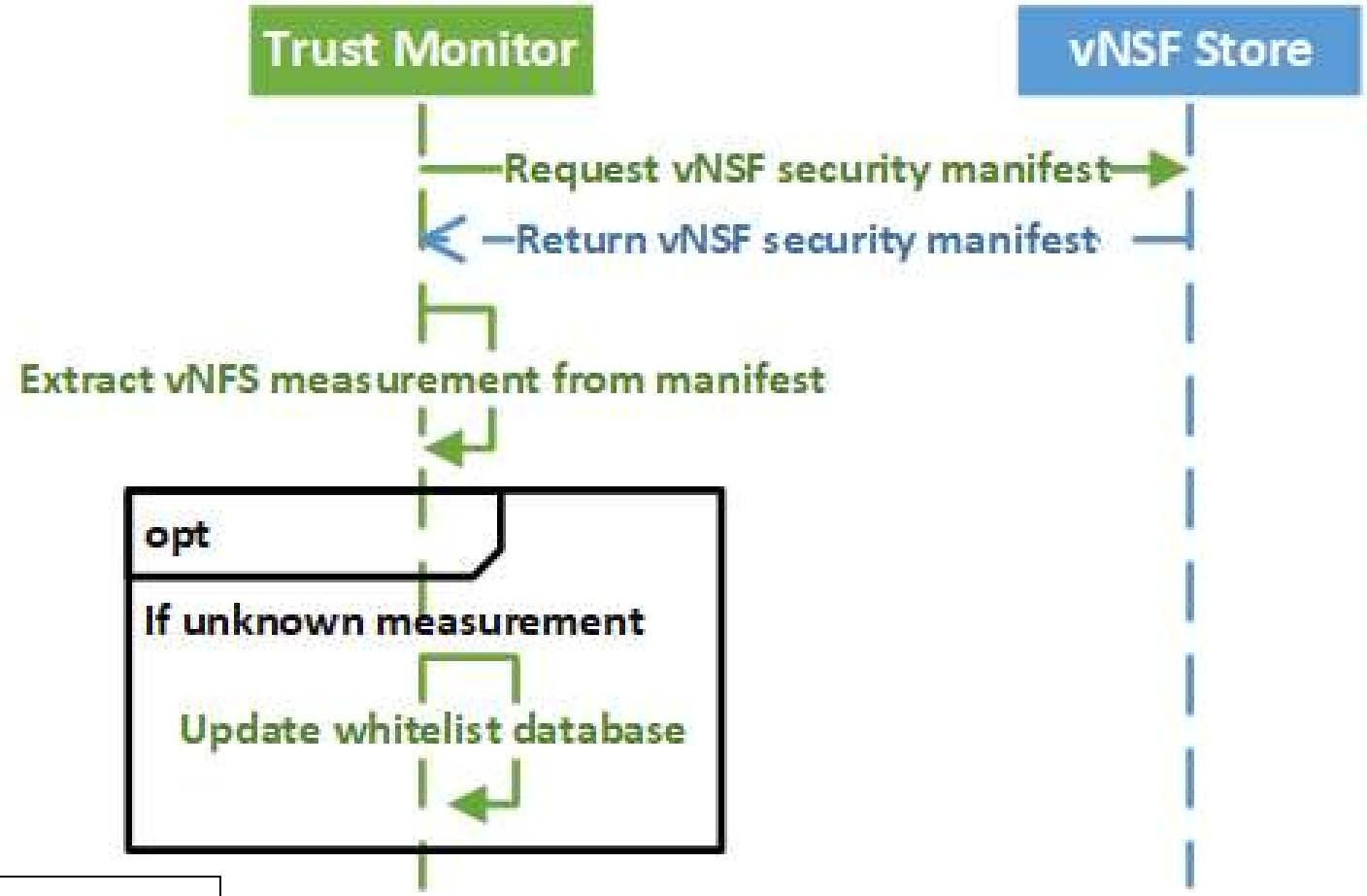
T10: Downgrade of NE's firmware (or simply out-of-date version)

SHIELD's Trust Monitor

- **Remote Attestation verifier**
  - using TPM-based measured boot
- **Trust Monitor attests**
  - physical server & vNSFs (container)
  - SDN-enabled physical switches
  - firmware, software & configuration verification

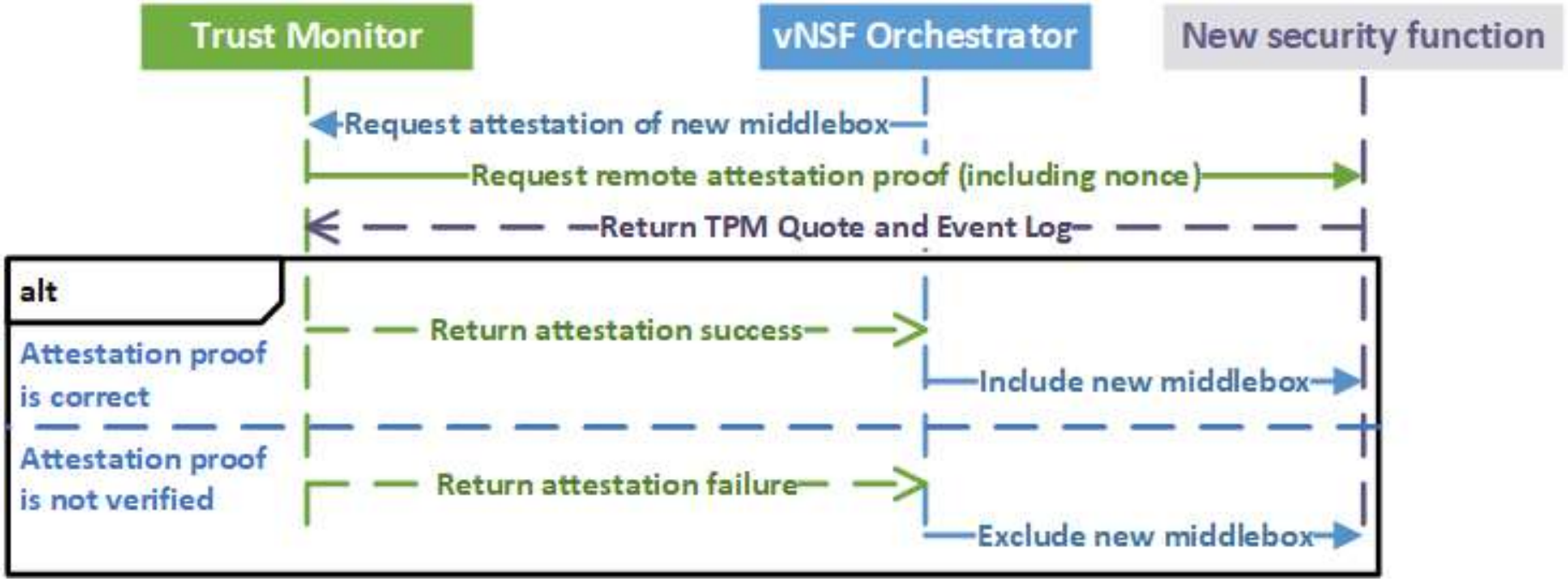
Golden value for virtual security functions

- **golden value created by the developers**
  - part of the security manifest



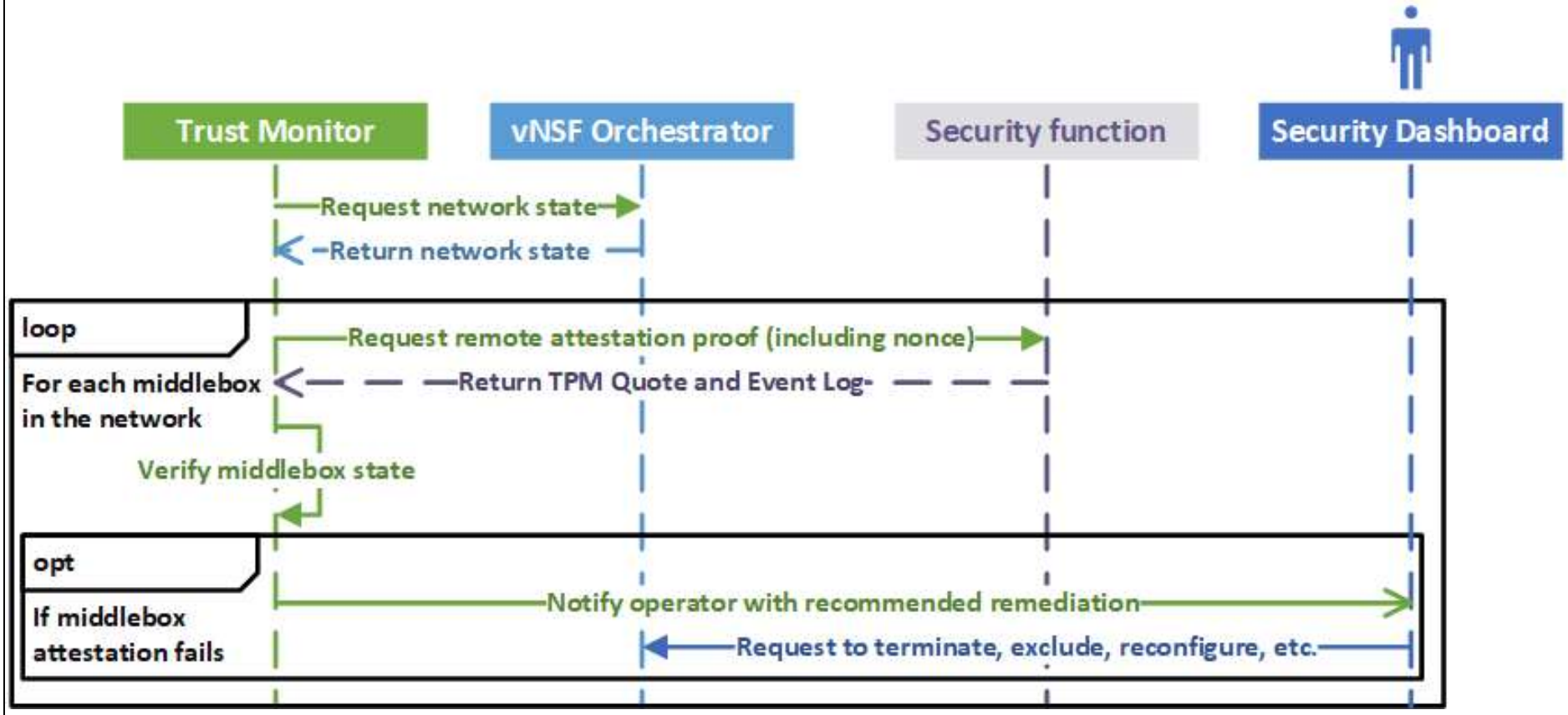
Initial deployment of a security function

- **security functions (and physical nodes) are verified before use**

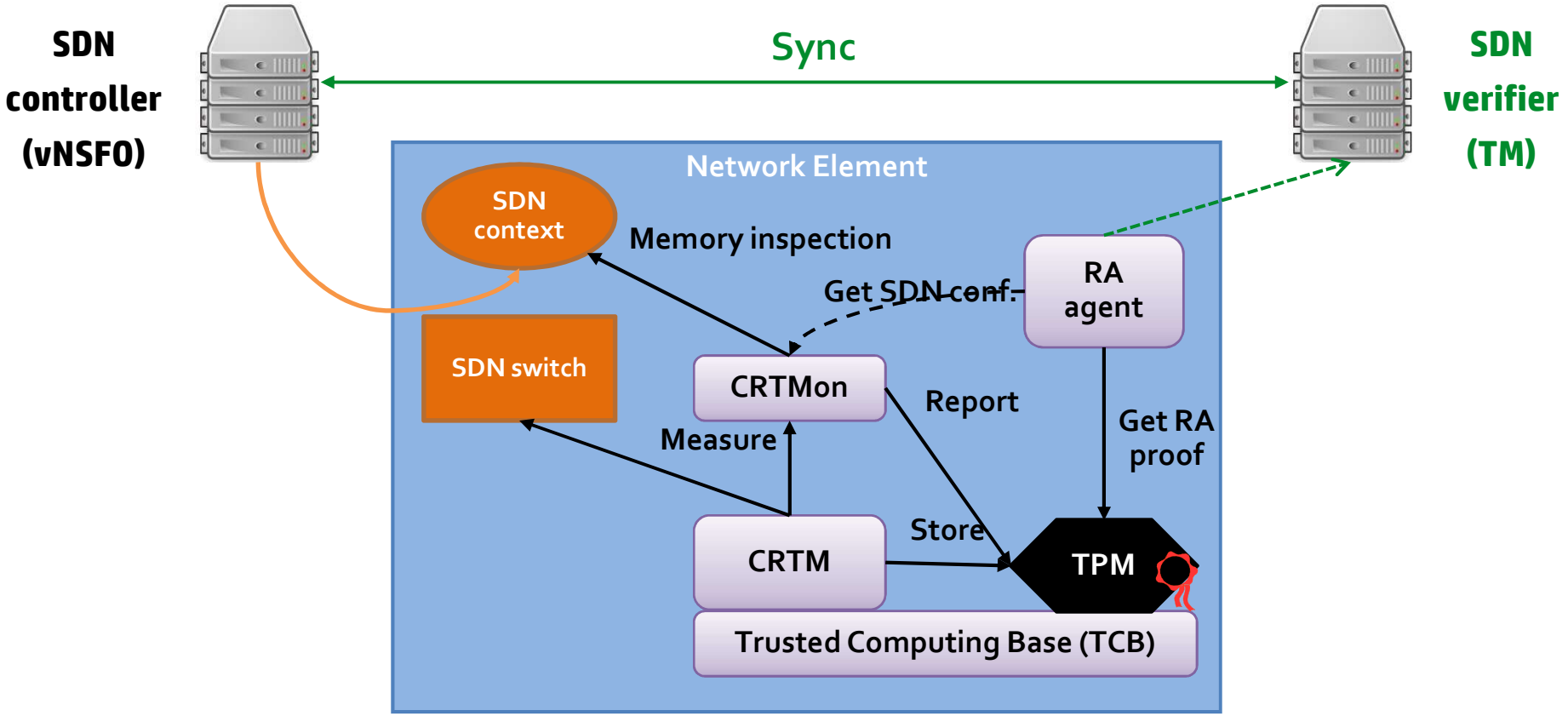


Periodic attestation of security functions

- continual attestation through the security functions lifecycle



Remote Attestation of SDN rules



SDN switches attestation

- **detection of unauthorised firmware, software and configuration**
- **detection of rogue SDN controller, incorrect SDN rules**
- **can handle 10k OpenFlow rules**
  - bottleneck between SDN controller & Trust Monitor
- **gap: no common (SDN controller-switch) identifier for rules**
- **around 2 to 3 seconds RTT**
  - 600ms for the TPM signature itself

SDN threats addressed

T1: Modification of control plane packs (from NE to the controller, i.e. table miss)

T2: Modification of control plane packs (from the controller to NE, i.e. rule update)

T4: Malicious or faulty administration of network elements

T7: Flashing of NE firmware (malicious software, persistent bootkits)

T9: Rogue SDN controller that alters configurations of network elements

T10: Downgrade of NE's firmware (or simply out-of-date version)

SHIELD attestation and remediation demo

https://www.youtube.com/watch?v=yKK9dYEyL-o

THANKS FOR YOUR ATTENTION !

QUESTIONS ?