Enarx

Protection for data in use

Mike Bursell
Office of the CTO

https://enarx.io

Nathaniel McCallum
Sr. Principal Engineer

Red Hat

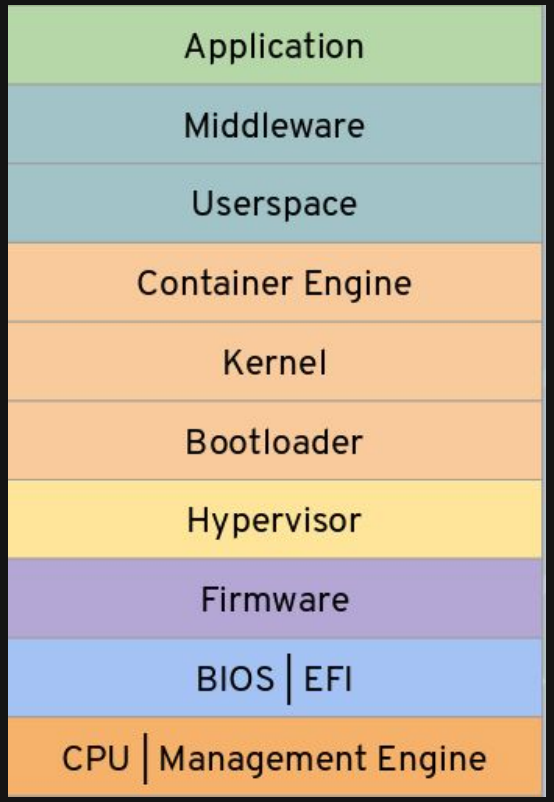
The Problem

The Need for Confidentiality and Integrity

- Banking & Finance
- Government & Public Sector
- Telco
- IoT
- HIPAA
- GDPR
- Sensitive enterprise functions
- Defense
- Human Rights NGOs
- ...

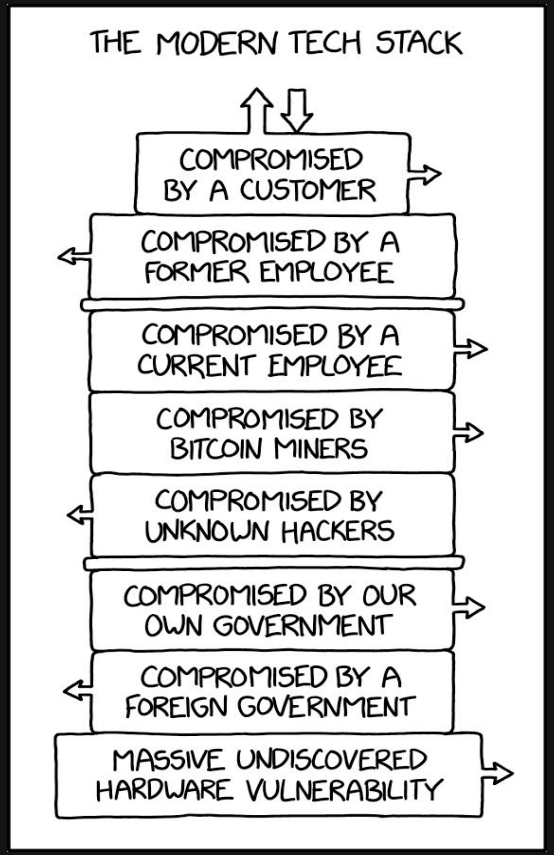
Virtualization Stack

Container Stack



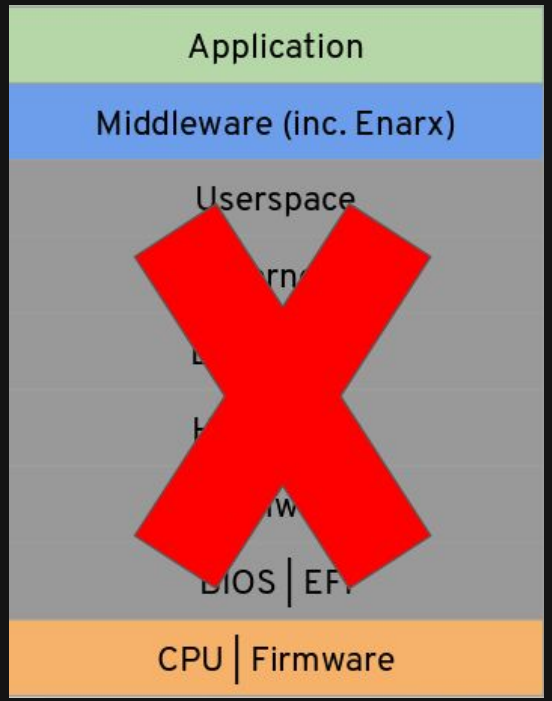
Red Hat

https://xkcd.com/2166/



Red Hat

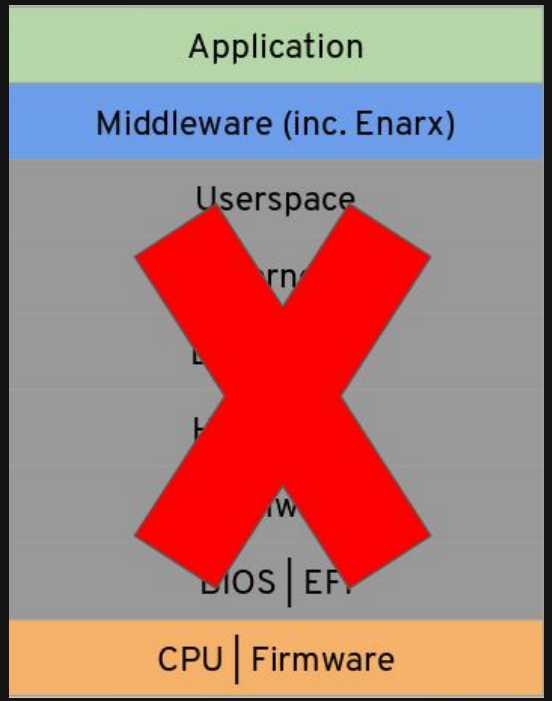
The Plan



Red Hat

The Principles

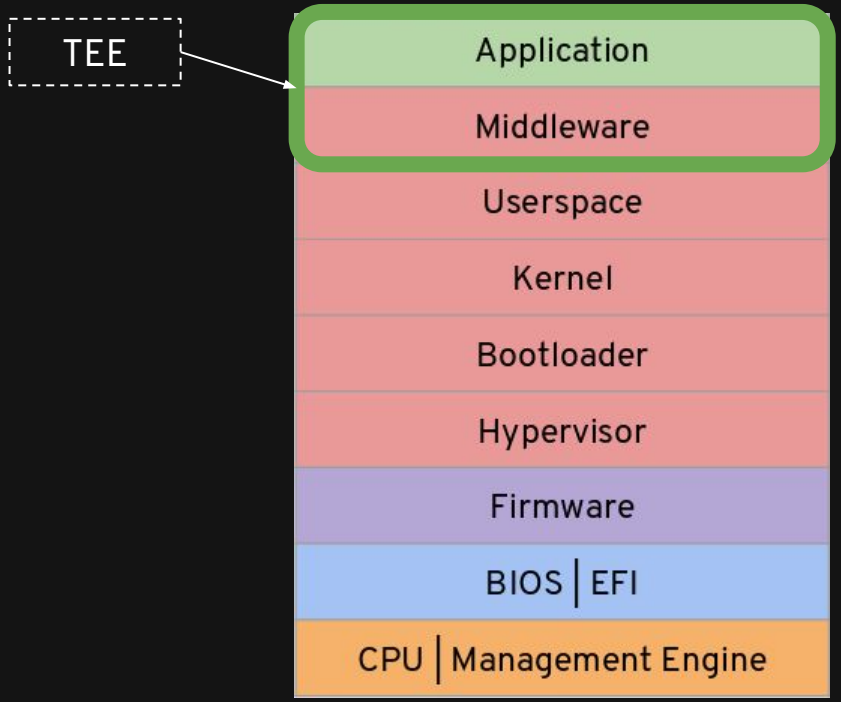
Don’t trust the host
Don’t trust the host owner
Don’t trust the host operator
All hardware cryptographically verified
All software audited and cryptographically verified



Trusted Execution Environments

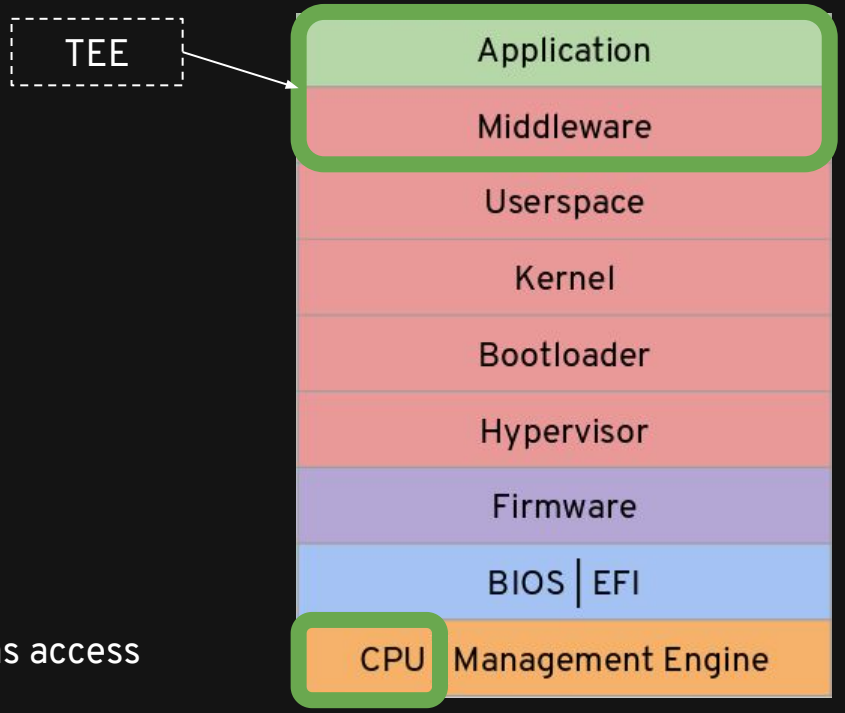
Red Hat

What's a TEE?



Red Hat

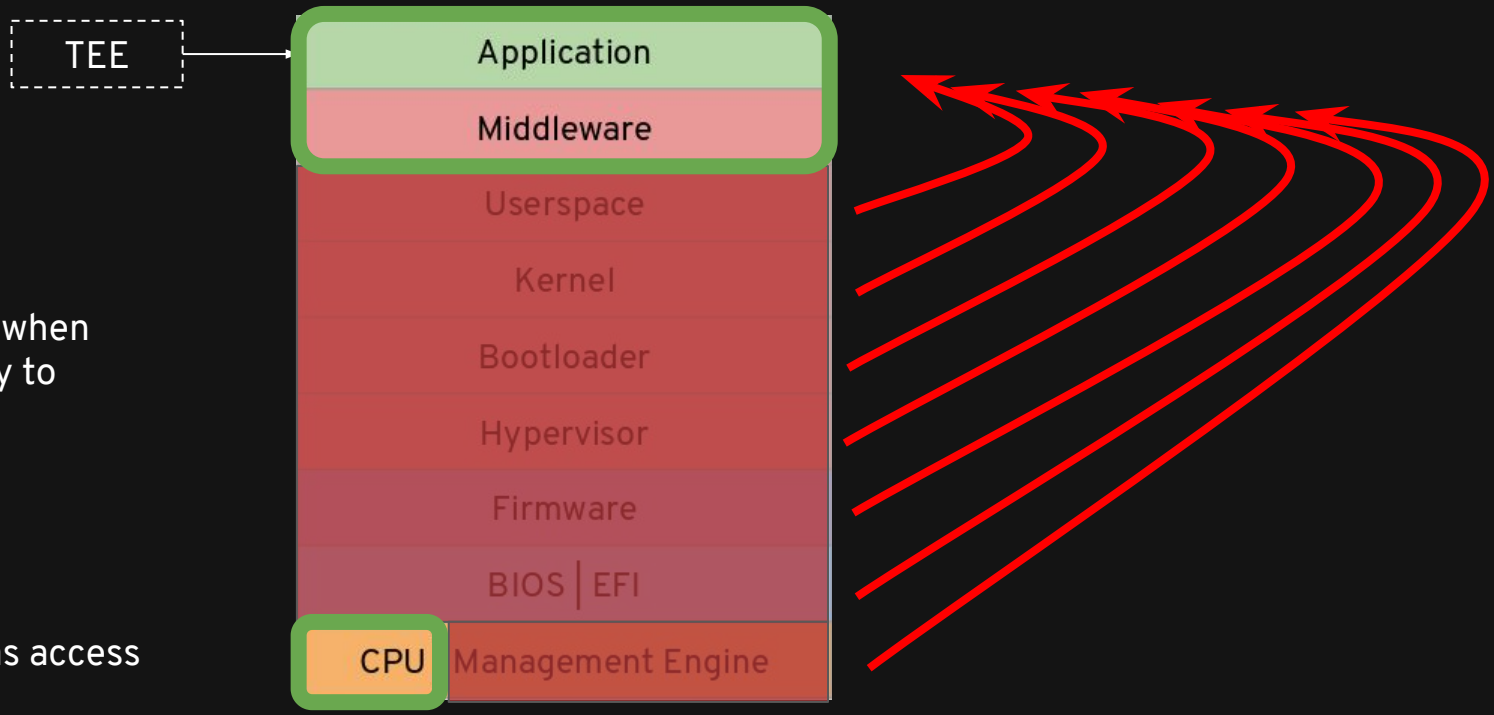
What's a TEE?



Only the CPU has access

Red Hat

What's a TEE?

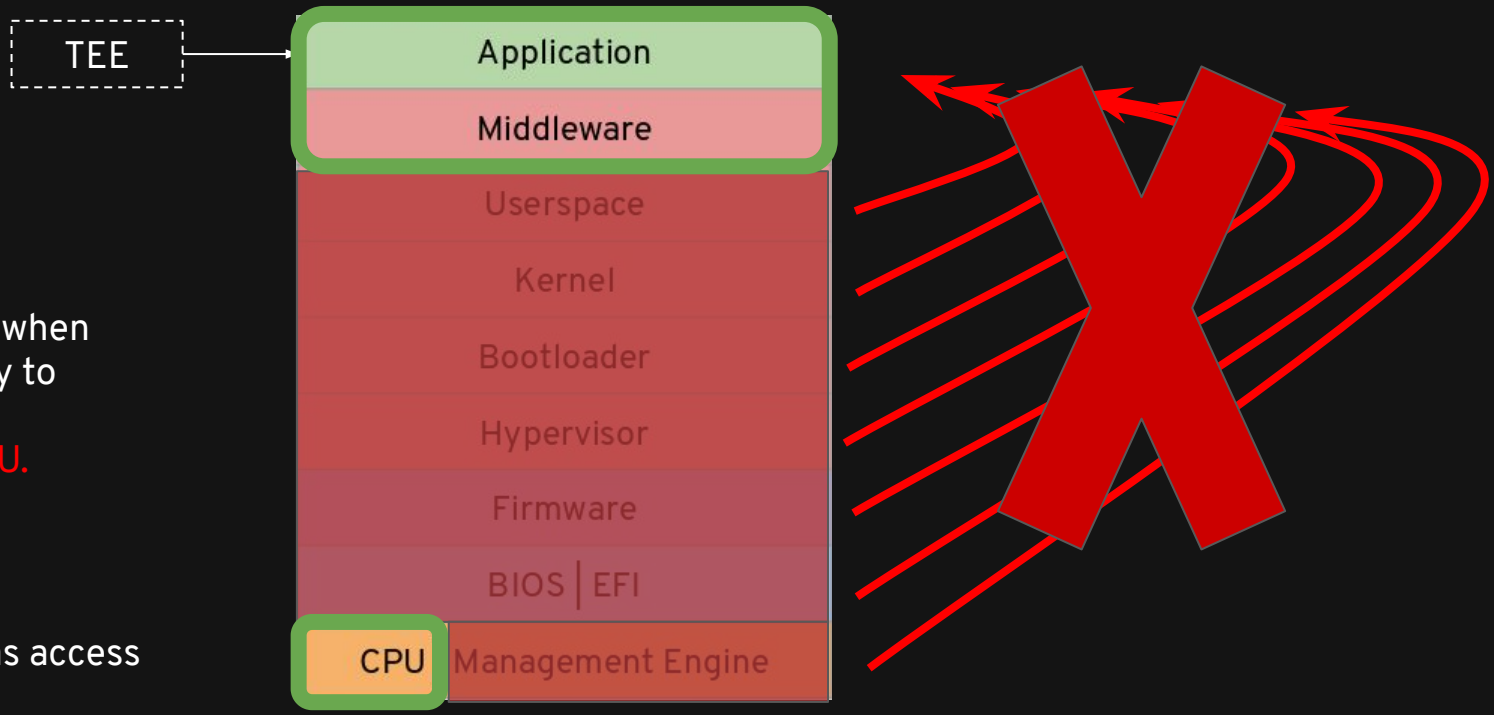


What happens when other layers try to access?

Only the CPU has access

Red Hat

What's a TEE?



What happens when other layers try to access?
Blocked by CPU.

Only the CPU has access

Red Hat

Trusted Execution Environments

TEE is a protected area within the host, for execution of sensitive workloads

Trusted Execution Environments

TEE is a protected area within the host, for execution of sensitive workloads

TEE provides:

- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

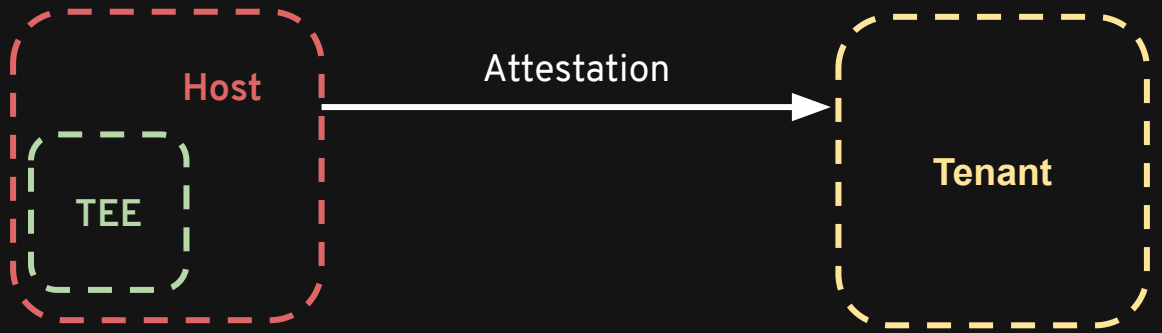
Trusted Execution Environments

Q. “But how do I know that it’s a valid TEE?”

TEE provides:

- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Trusted Execution Summary

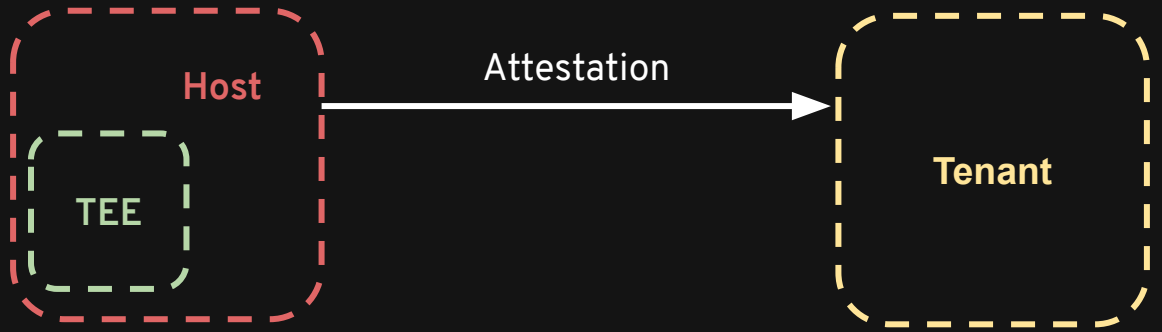


Q. “But how do I know that it’s a valid TEE?”
A. Attestation

TEE provides:

- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Trusted Execution Summary



Attestation includes:

- Diffie-Hellman Public Key
- Hardware Root of Trust
- TEE Measurement

TEE provides:

- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Trusted Execution Summary

Attestation includes:

- Diffie-Hellman Public Key
- Hardware Root of Trust
- TEE Measurement

TEE provides:

- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Trusted Execution Models

Process-Based

- Intel SGX (not upstream)
- RISC-V Sanctum (no hardware)

VM-Based

- AMD SEV
- IBM PEF (no hardware)
- Intel MKTME (no attestation[1])

Not a TEE: TrustZone, TPM

1. Attestation is discussed here: https://patents.google.com/patent/US20190042463A1/en?oq=20190042463

Trusted Execution: Process-Based

PROS

- Access to system APIs from Keep

CONS

- Unfiltered system API calls from Keep
- Application redesign required
- Untested security boundary
- Fantastic for malware
- Lock-in

Trusted Execution: Virtual Machine-Based

PROS

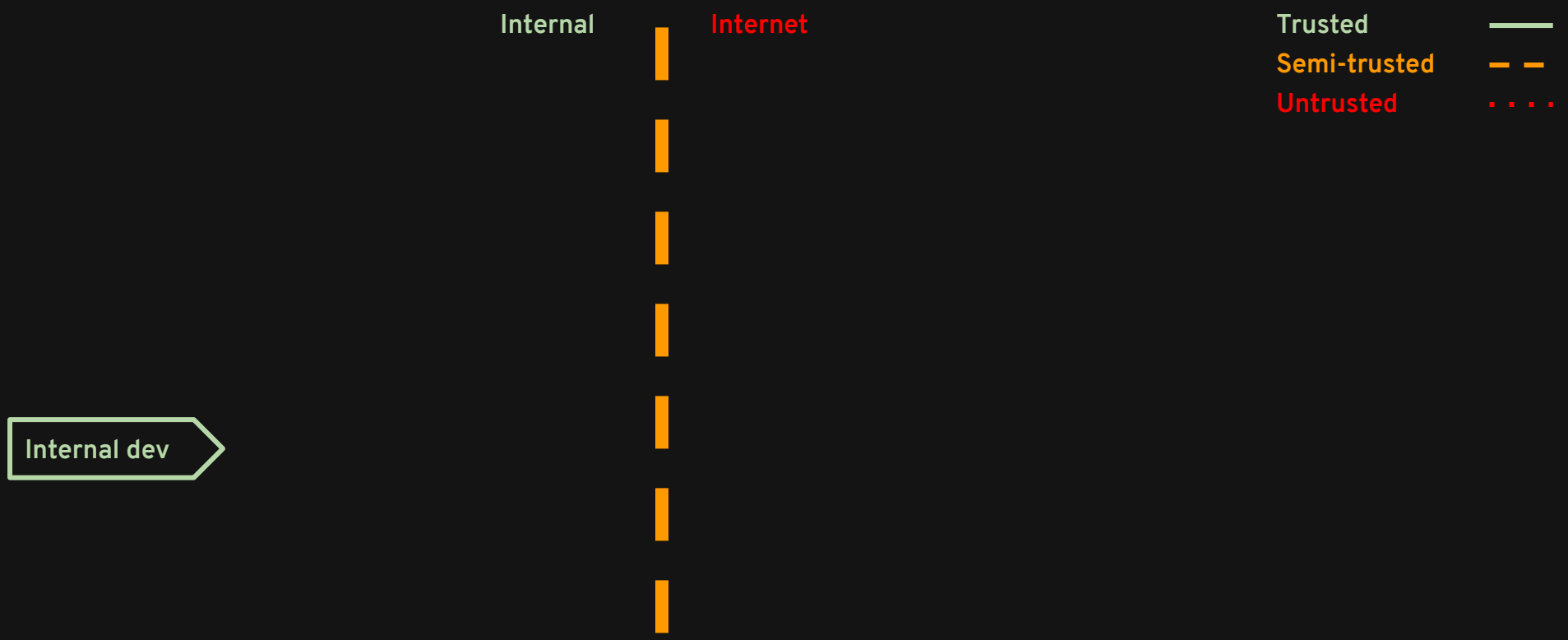
- Strengthening of existing boundary
- Run application on existing stacks
- Bidirectional isolation
- Limits malware

CONS

- Hardware emulation
- Heavy weight for microservices
- CPU architecture lock-in
- Duplicated kernel pages
- Host-provided BIOS

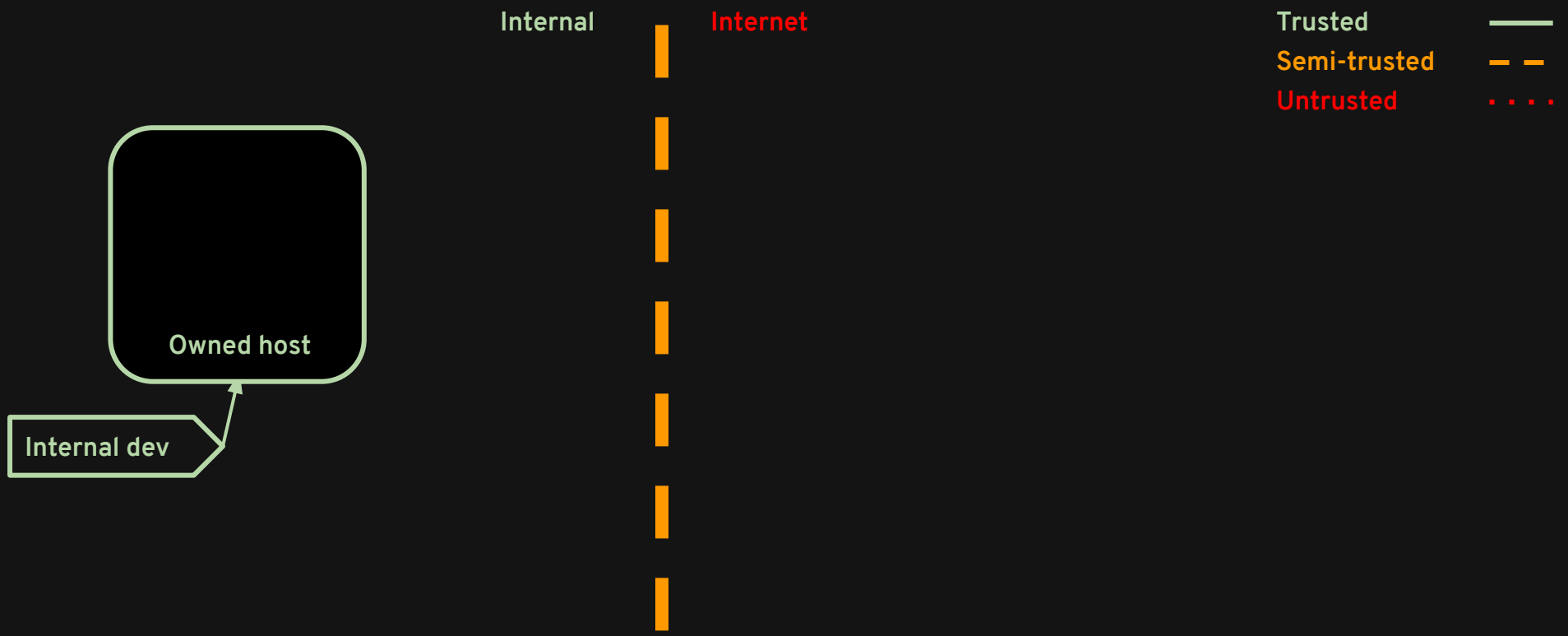
Open hybrid cloud and Enarx

Step 1: on premises



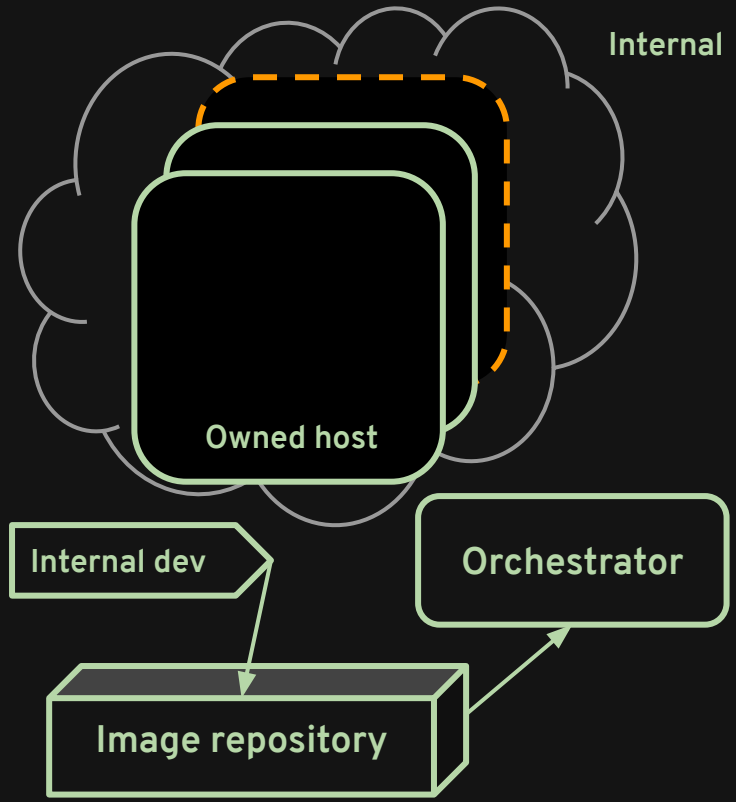
Red Hat

Step 1: on premises



Red Hat

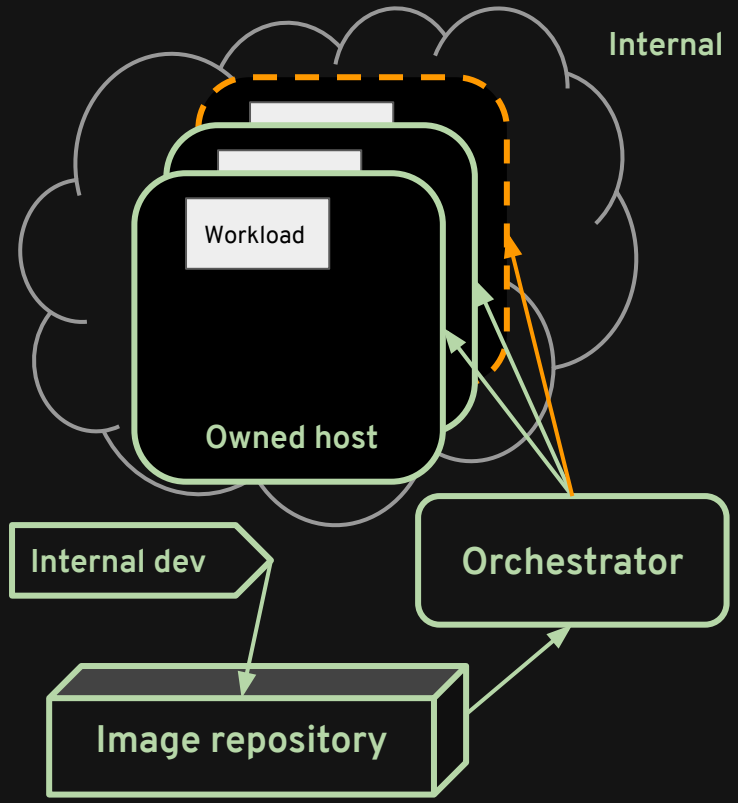
Step 2: private cloud



Trusted
Semi-trusted
Untrusted

Red Hat

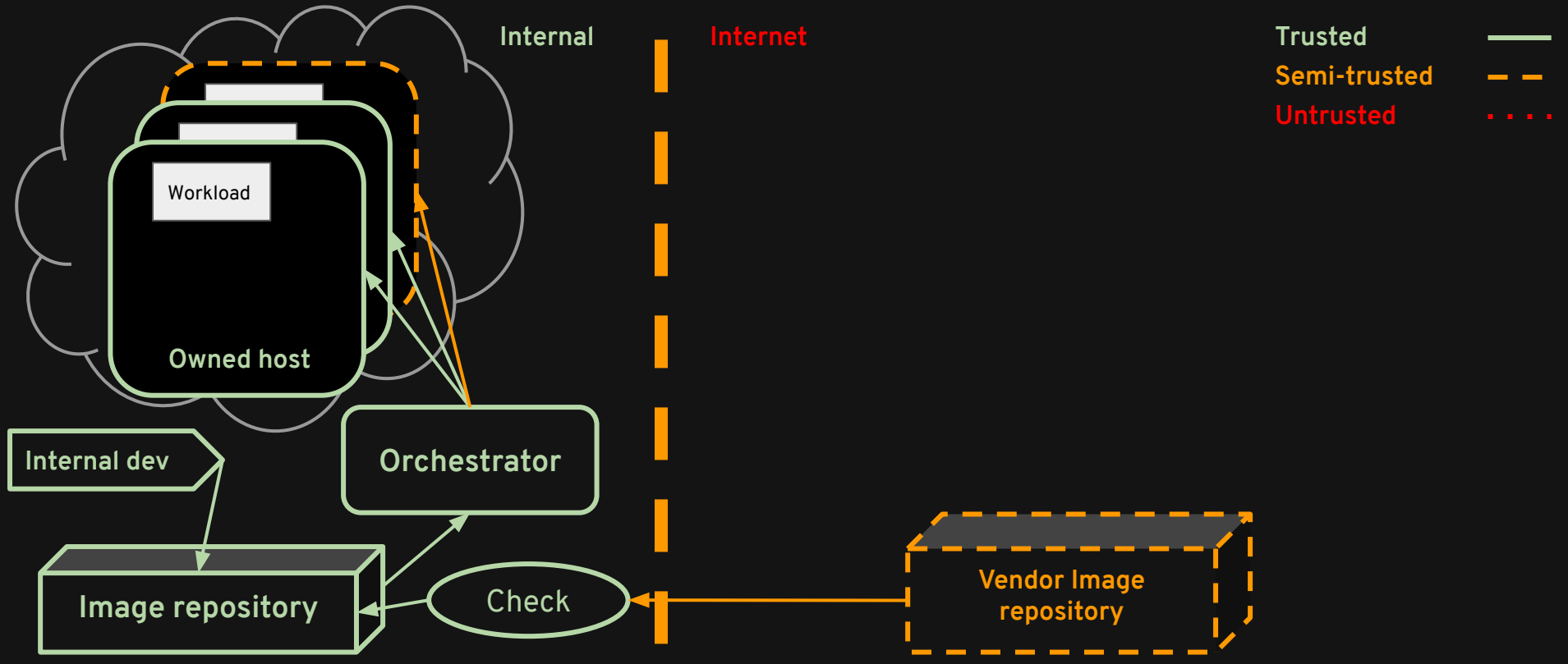
Step 2: private cloud



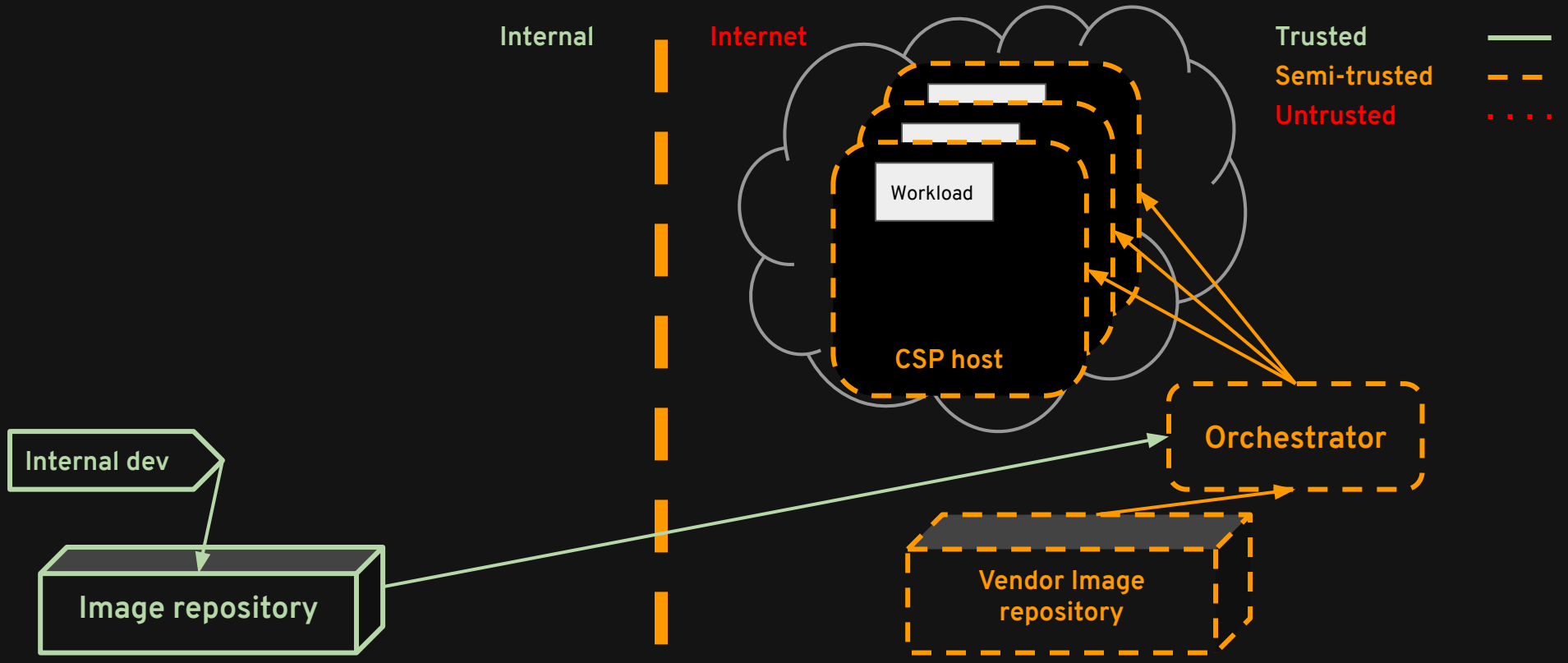
Trusted
Semi-trusted
Untrusted

Red Hat

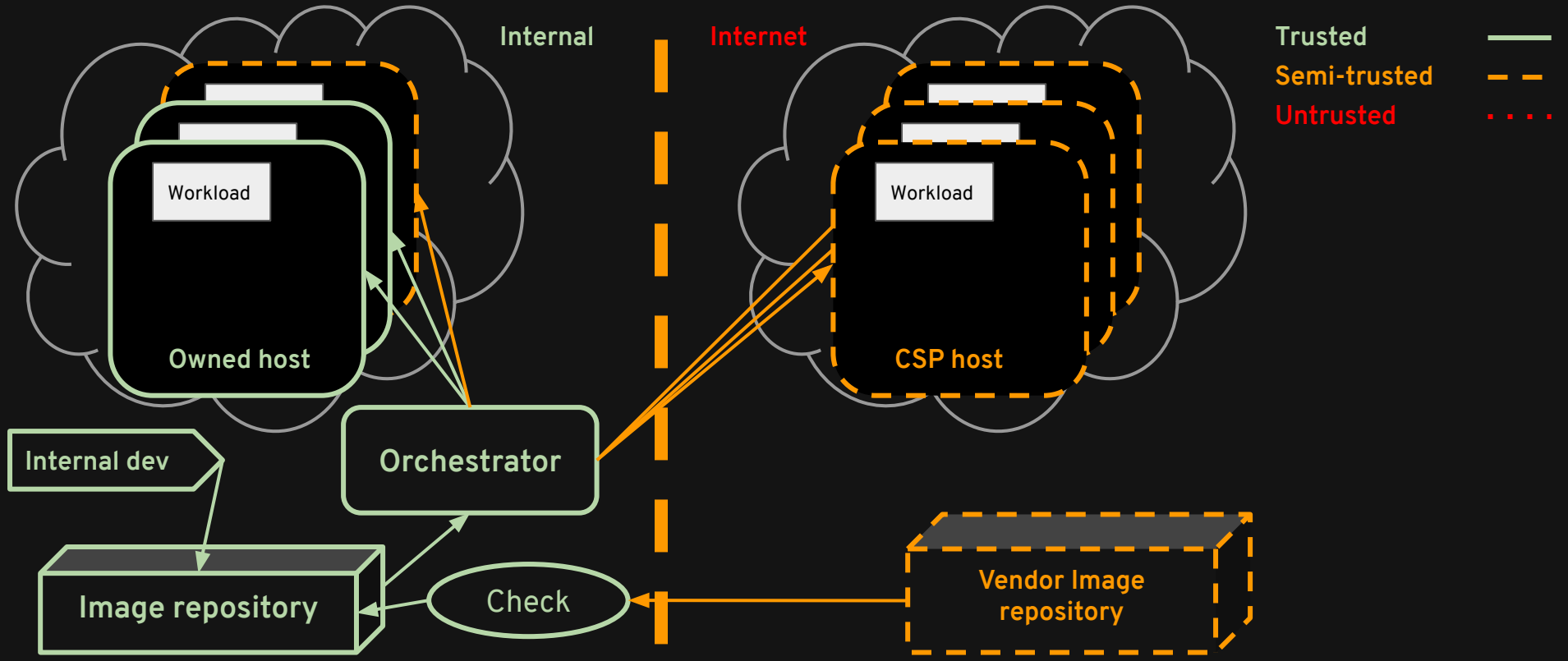
Step 2: private cloud



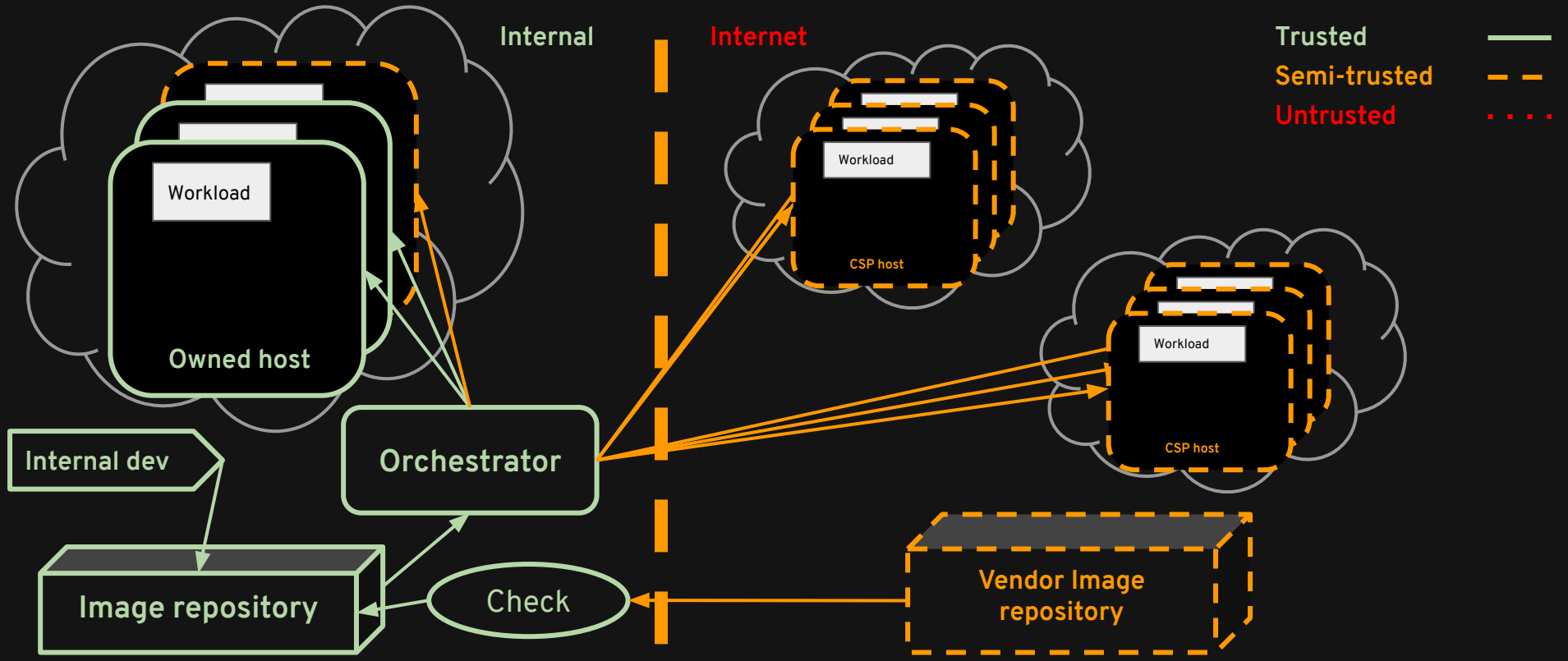
Step 3: public cloud



Step 4: hybrid cloud

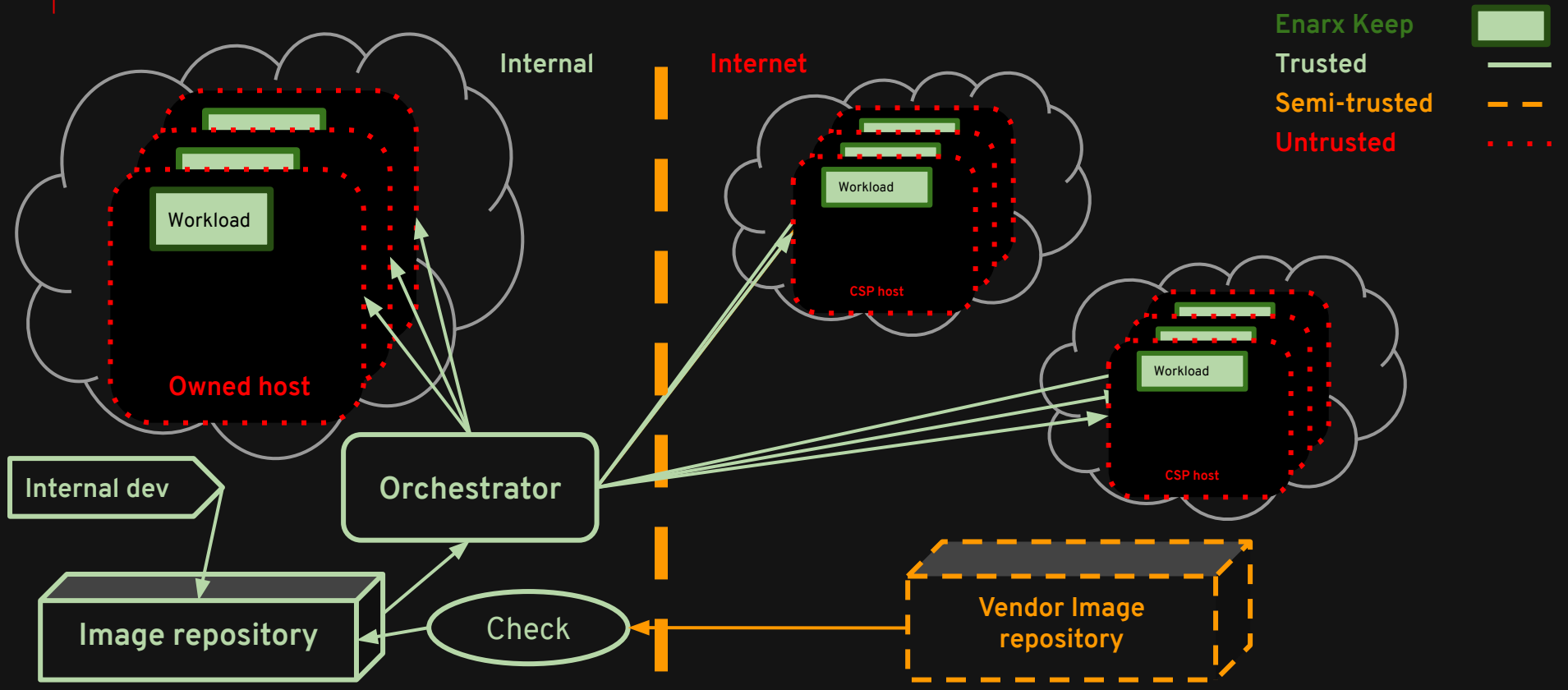


Step 5: hybrid multicloud



Red Hat

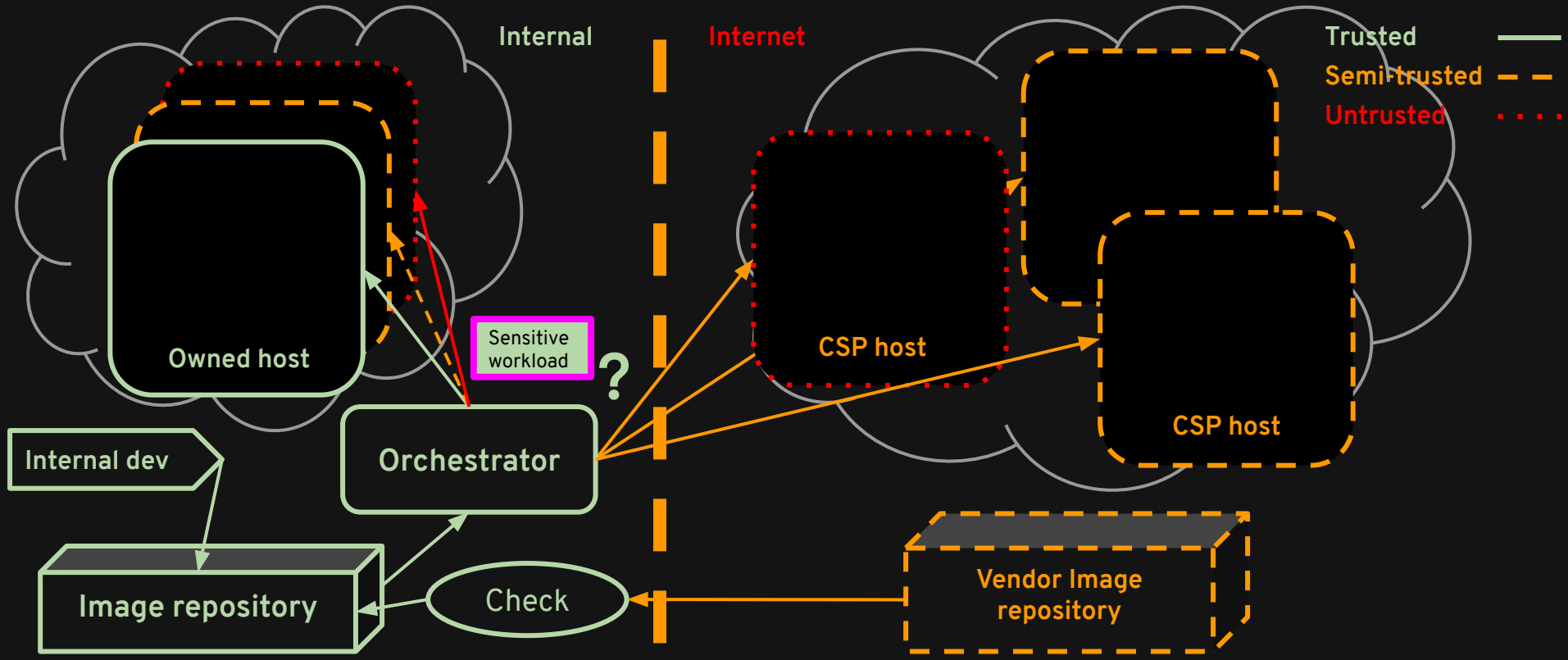
Step 6: Enarx hybrid multicloud



Red Hat

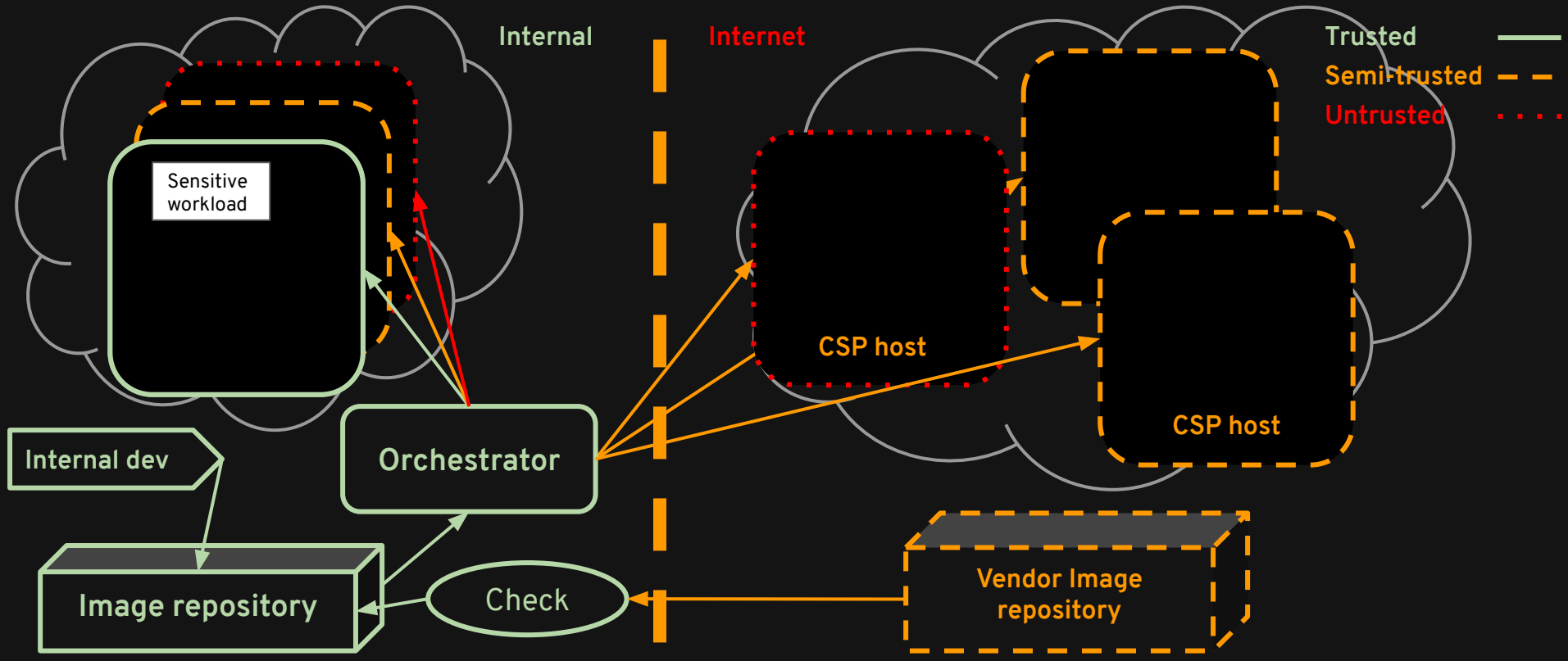
New options for workloads with Enarx

Mix and match for different workload types & Enarx



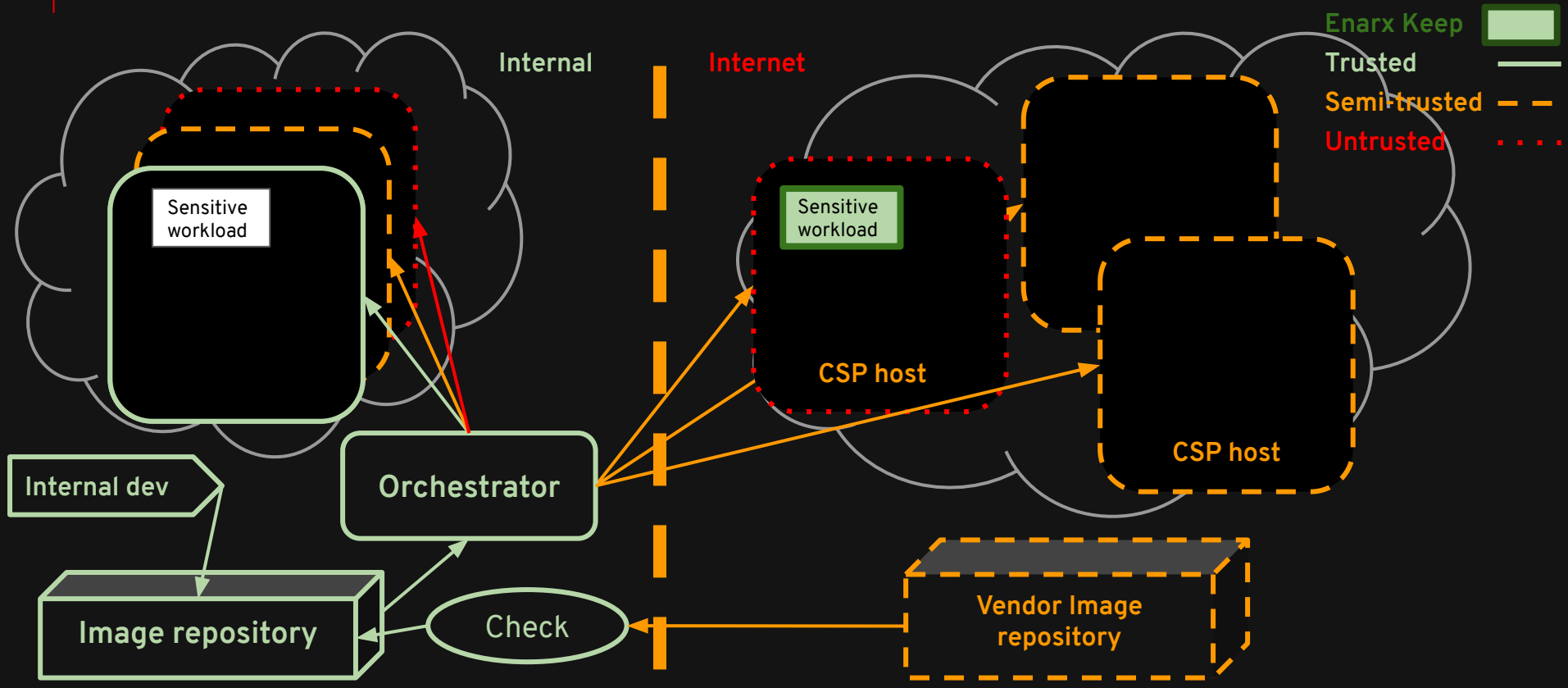
Red Hat

Mix and match for different workload types & Enarx



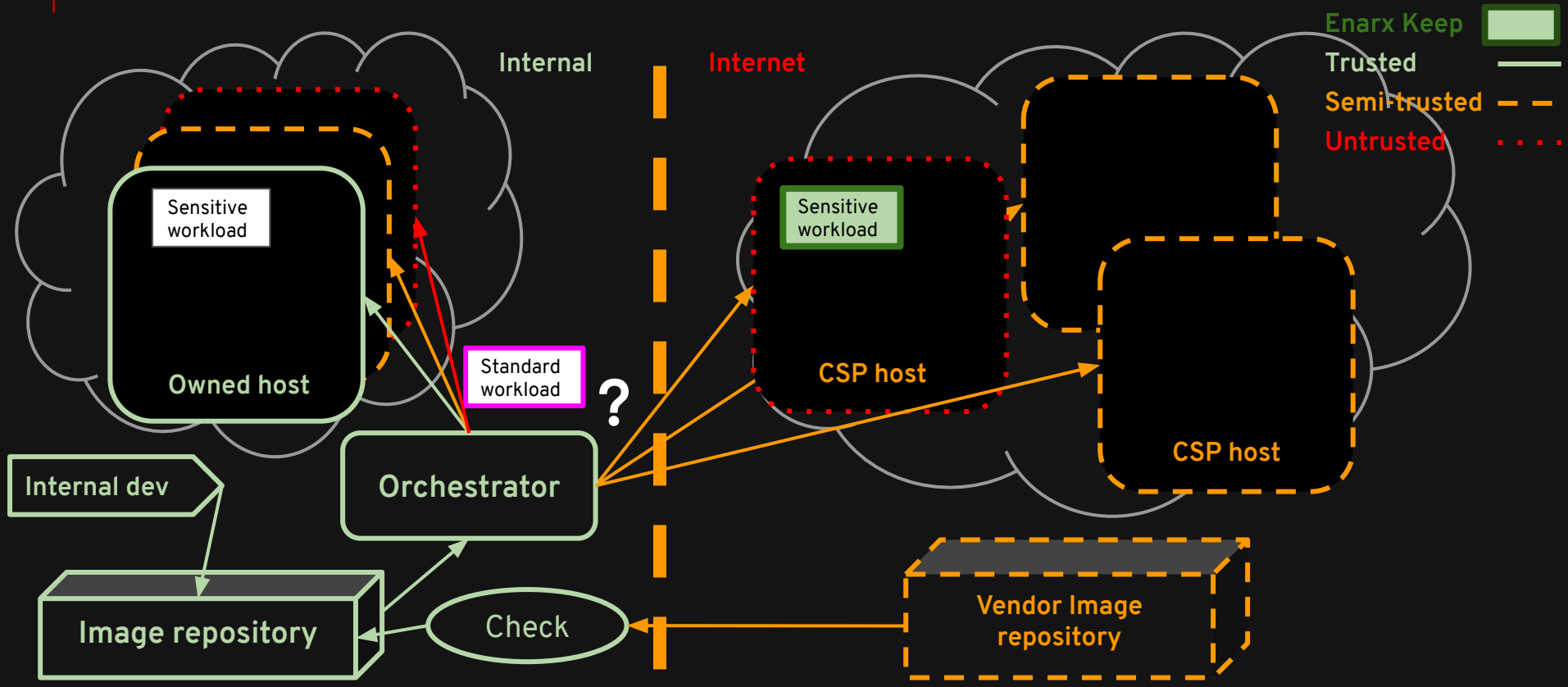
Red Hat

Mix and match for different workload types & Enarx



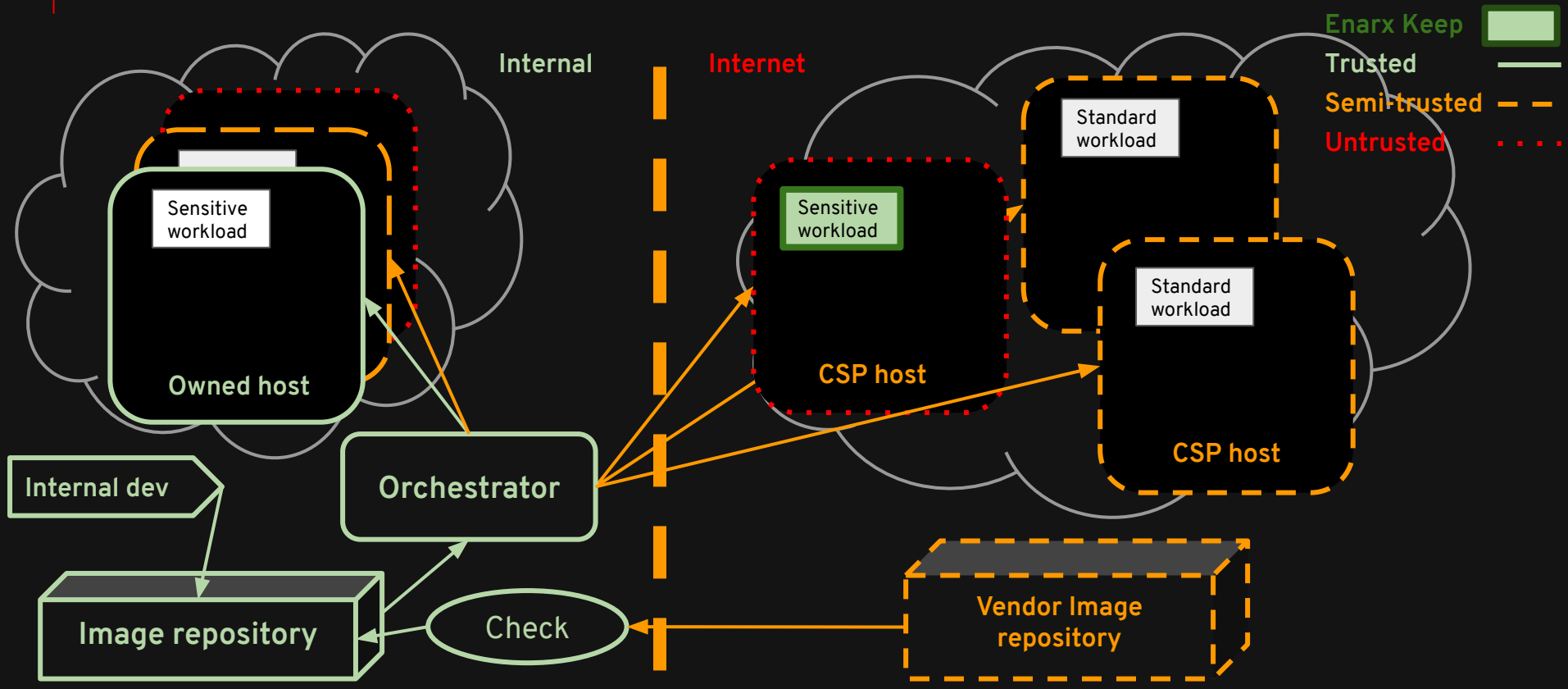
Red Hat

Mix and match for different workload types & Enarx



Red Hat

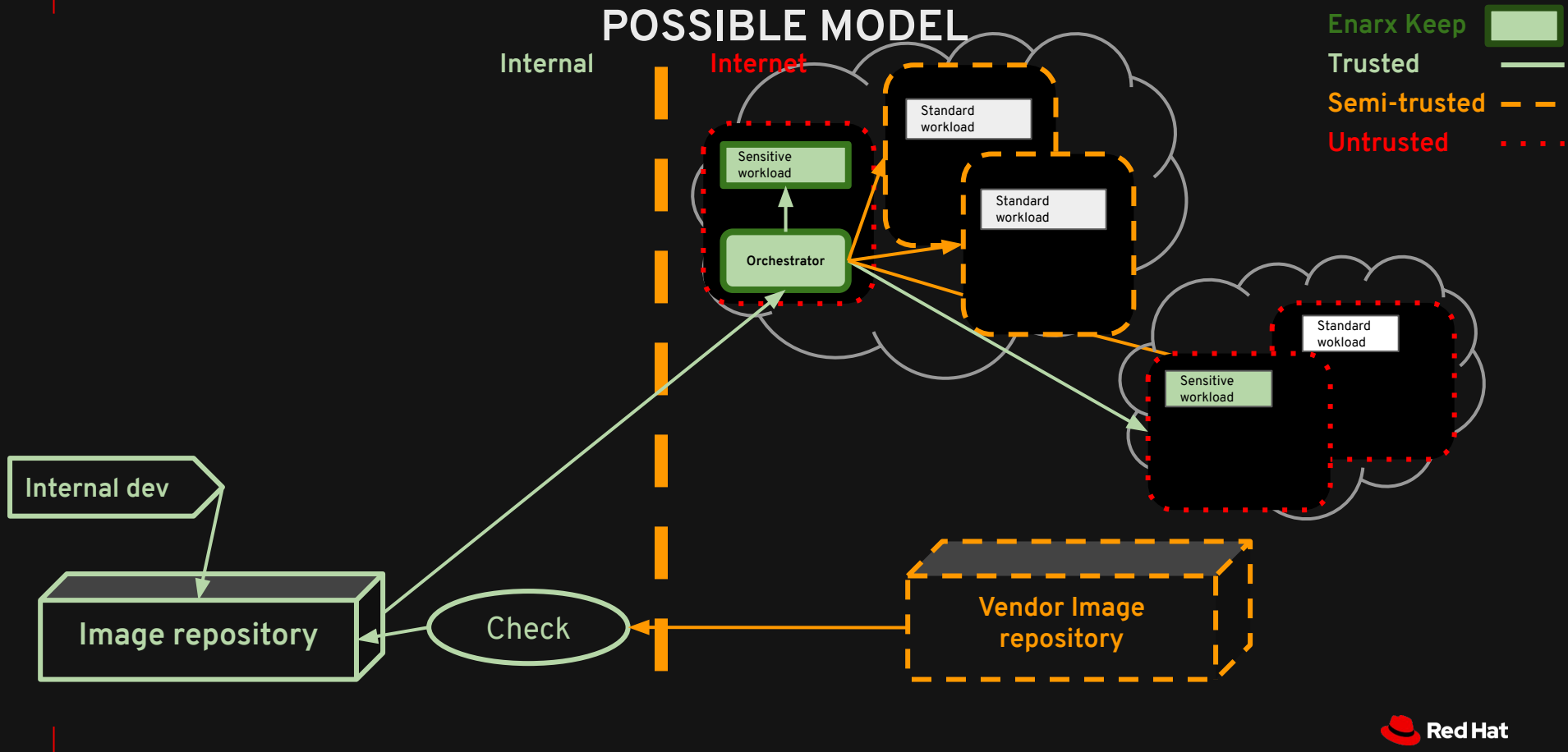
Mix and match for different workload types & Enarx



Red Hat

New options for orchestration with Enarx

Mix and match for different workload types & Enarx



Red Hat

Value of the trusted stack

How trusted stacks help

Hardware platform

Red Hat

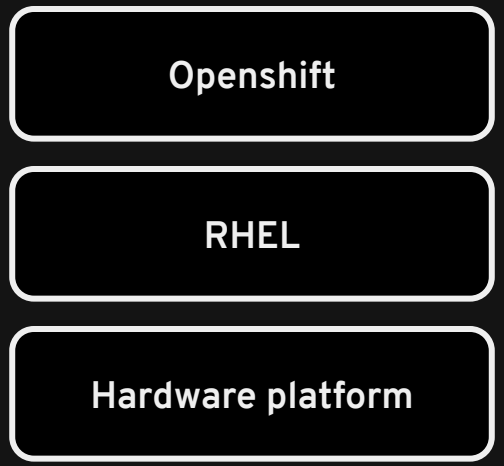
How trusted stacks help

RHEL

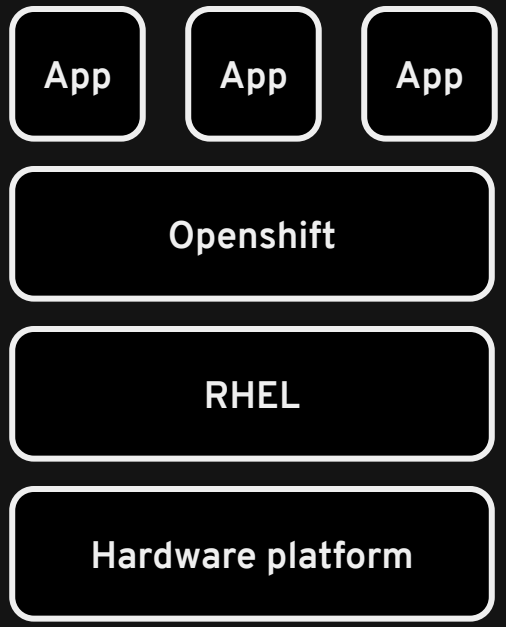
Hardware platform

Red Hat

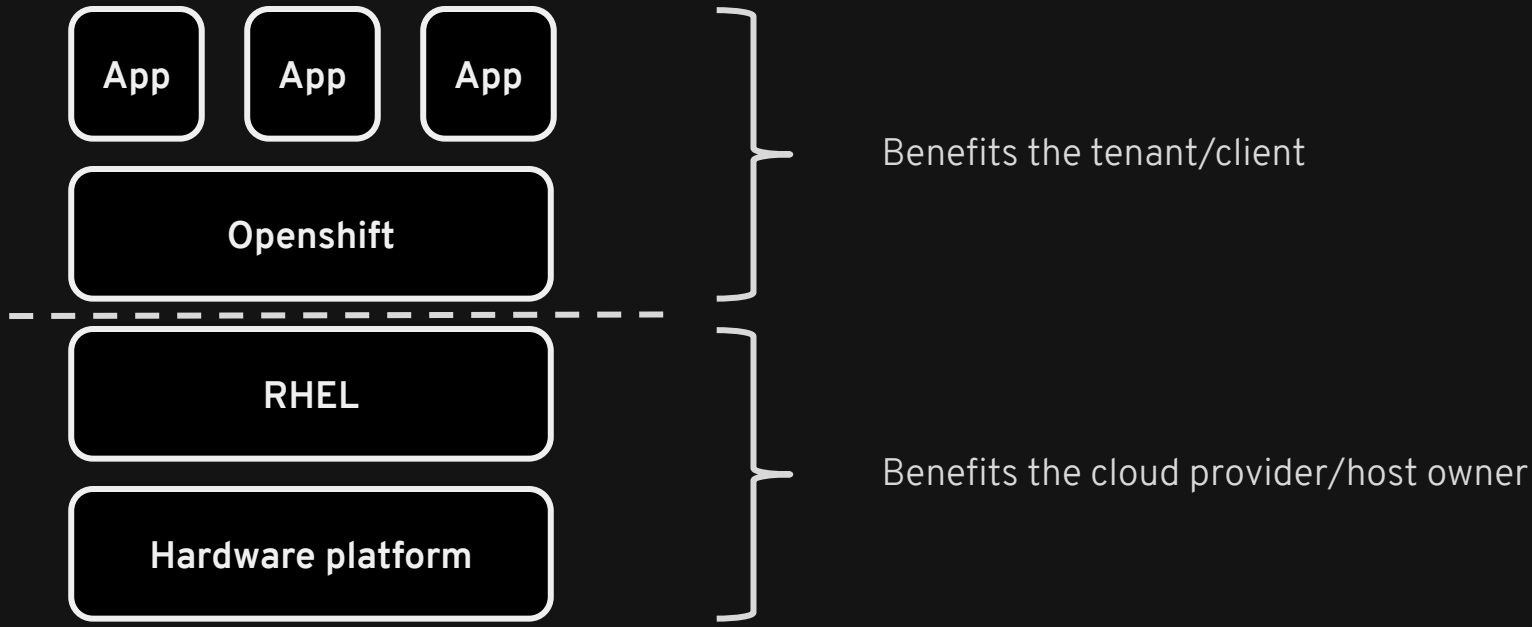
How trusted stacks help



How trusted stacks help

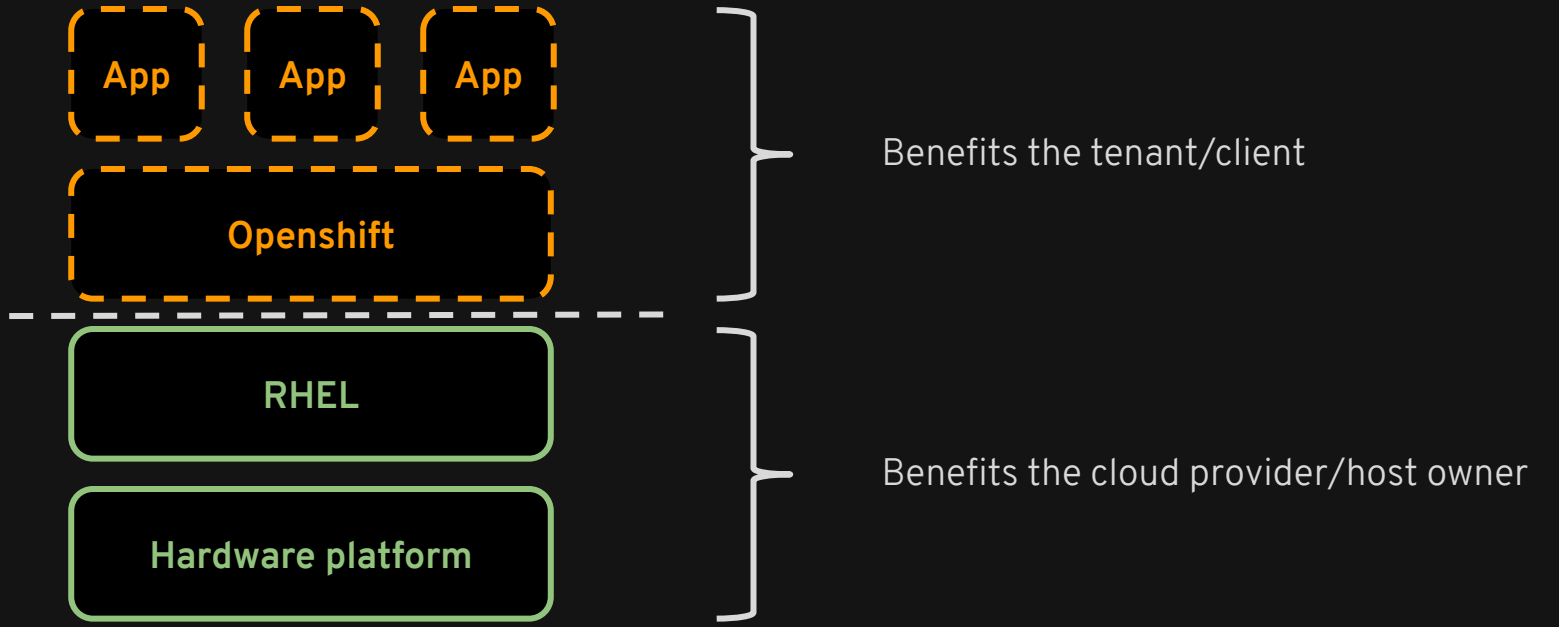


How trusted stacks help



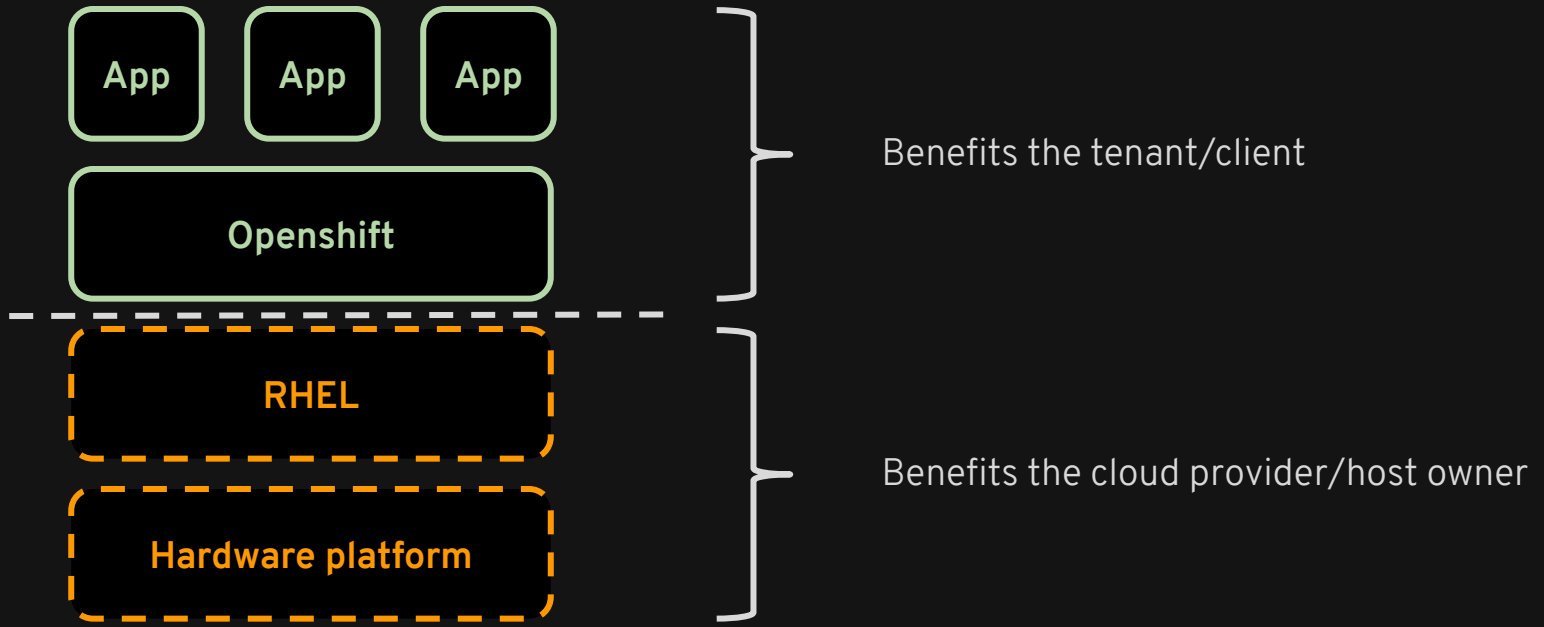
How trusted stacks help

Host view



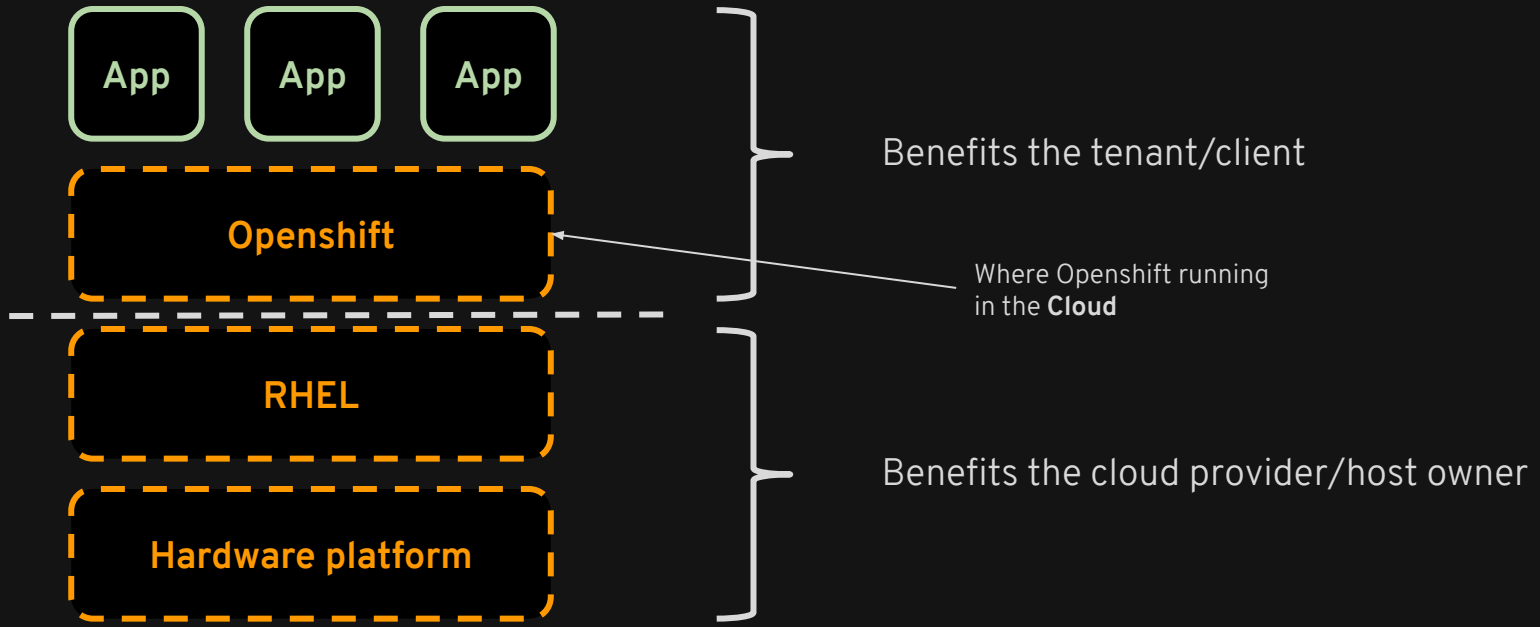
How trusted stacks help

Client view (hoped)



How trusted stacks help

Client view (actual)

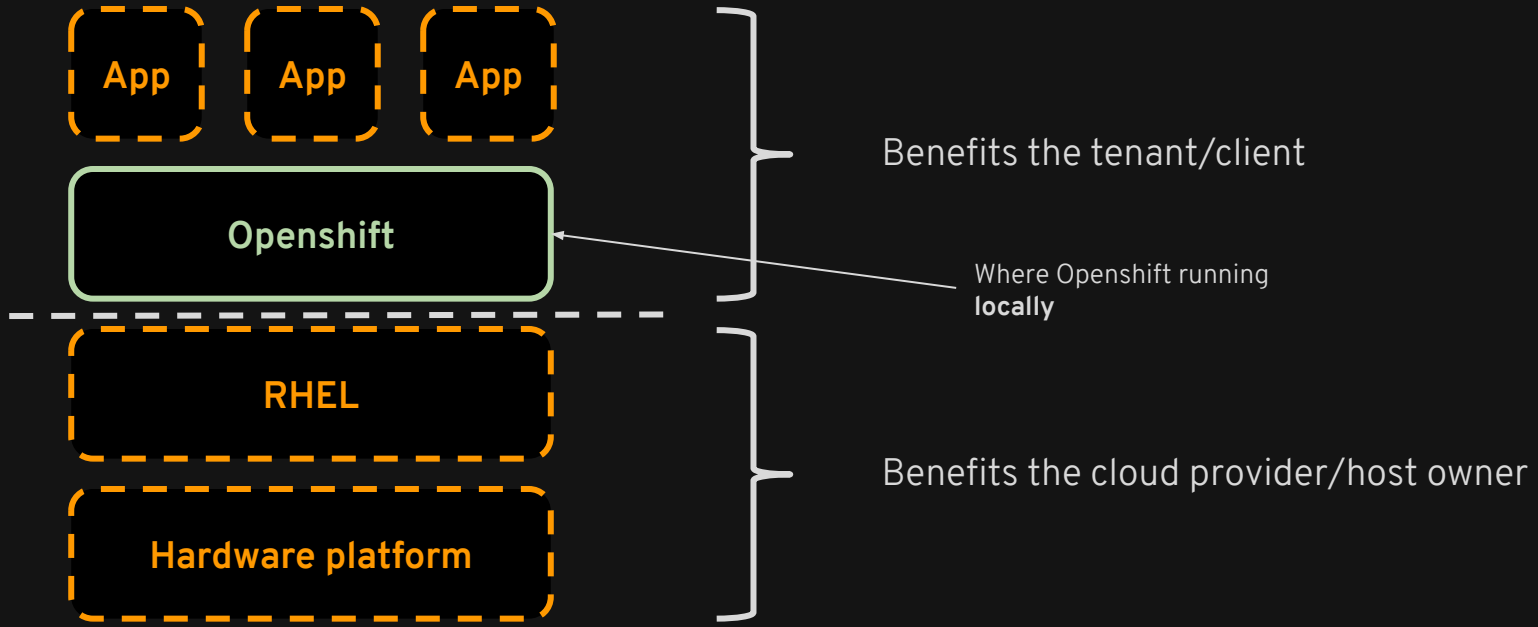


Enarx Keep
Trusted
Semi-trusted
Untrusted

Red Hat

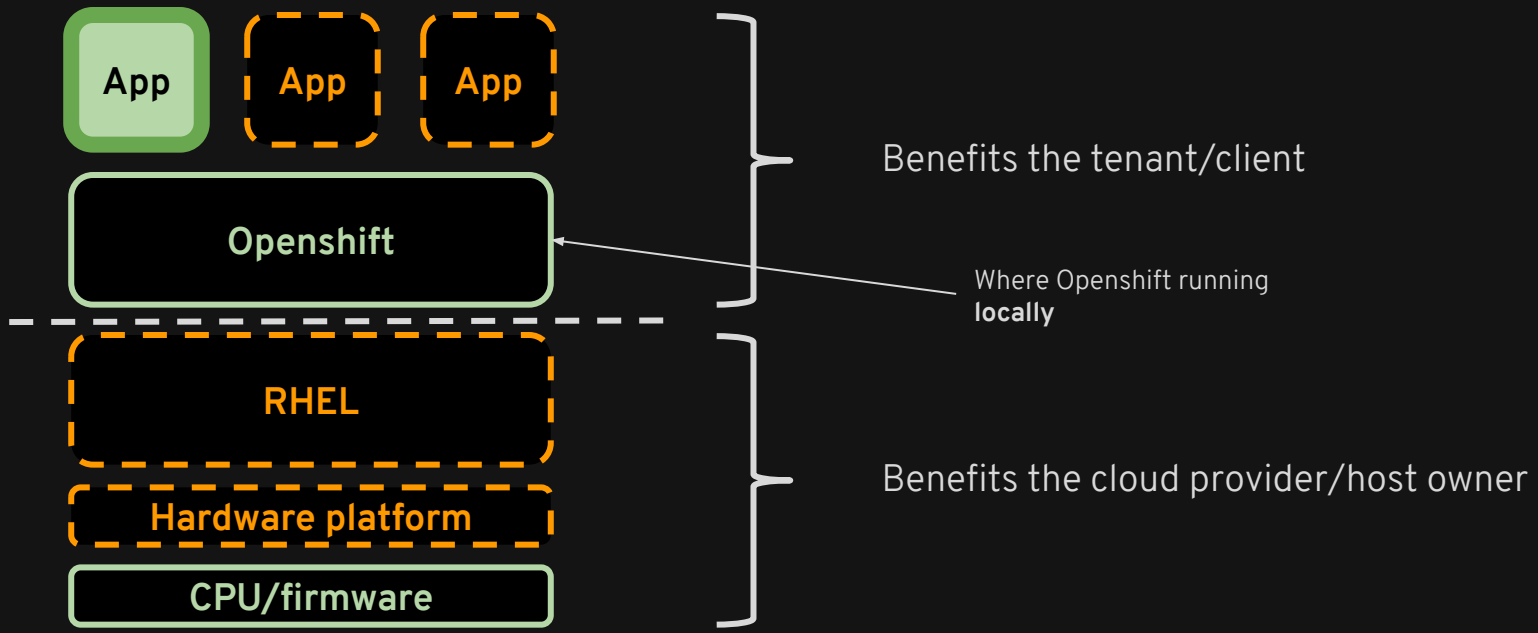
How trusted stacks help

Client view (actual)



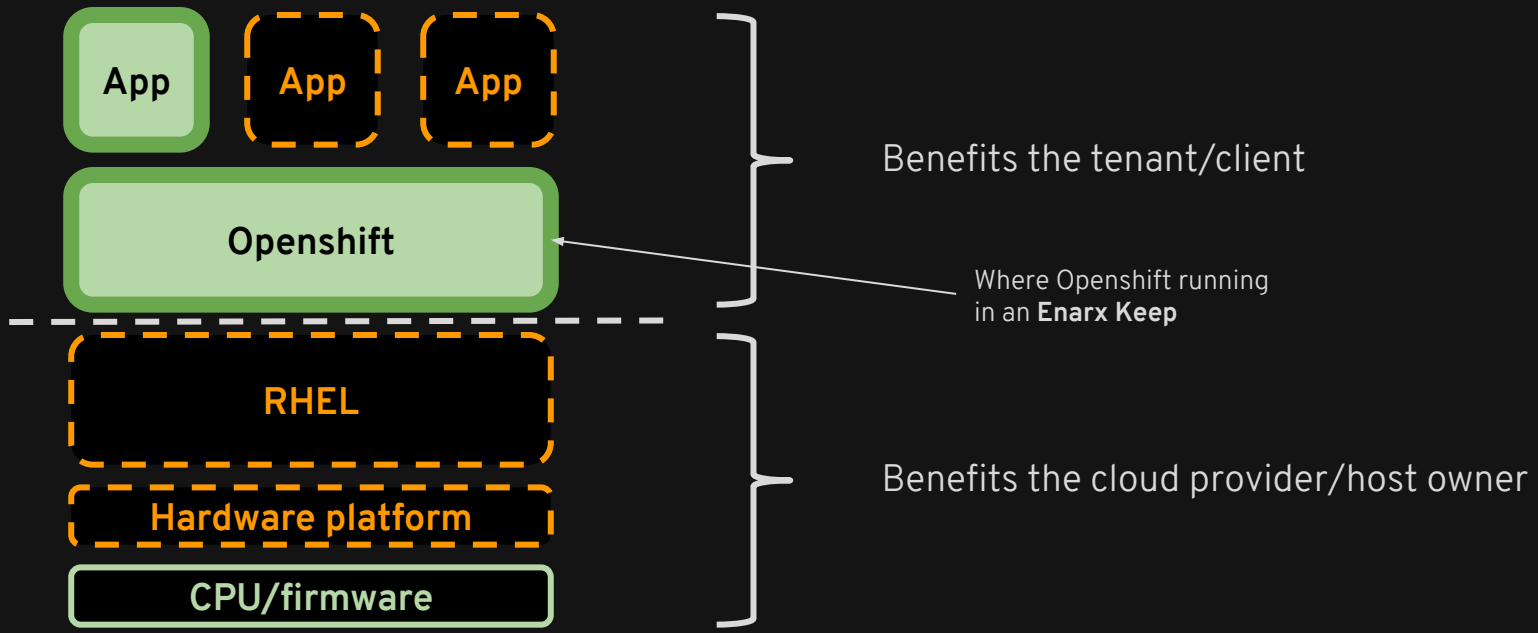
How trusted stacks help

Client view (with Enarx)



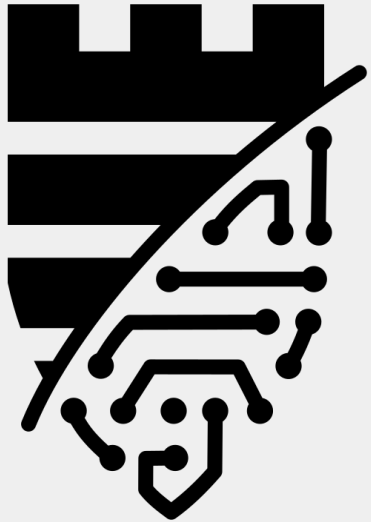
How trusted stacks help

Client view (with Enarx)



On which technology do I build my application?

Introducing Enarx



Red Hat

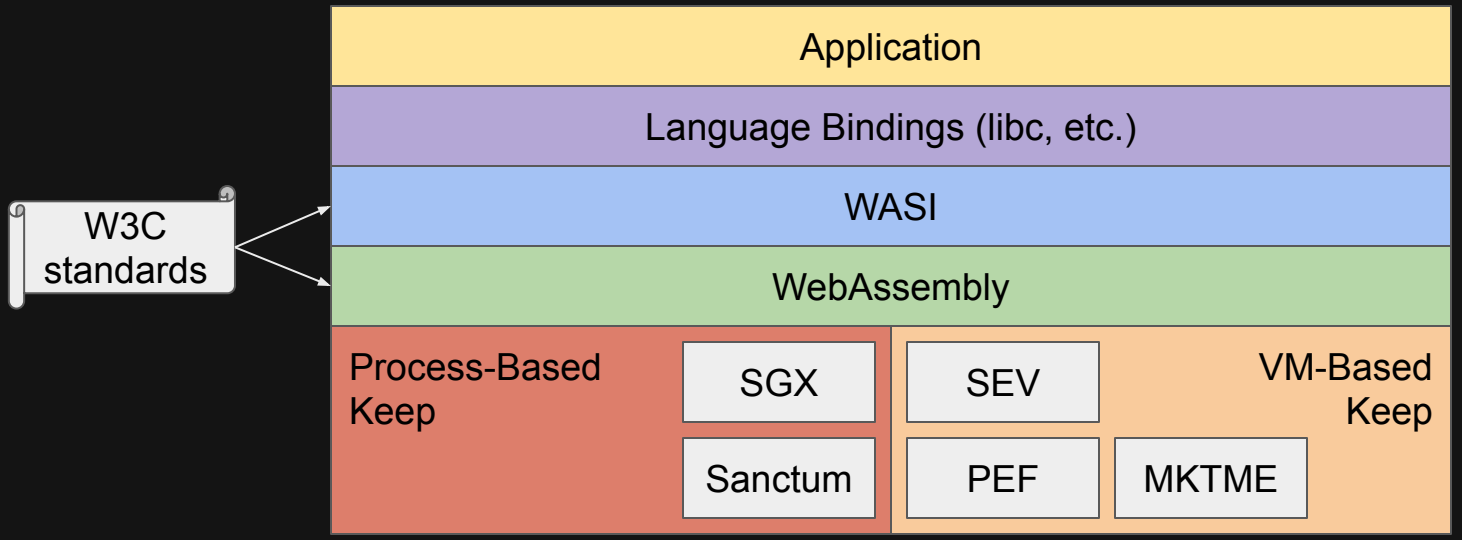
Enarx Principles

1. We don’t trust the host owner
2. We don’t trust the host software
3. We don’t trust the host users
4. We don’t trust the host hardware
    a. … but we’ll make an exception for CPU + firmware

Enarx Design Principles

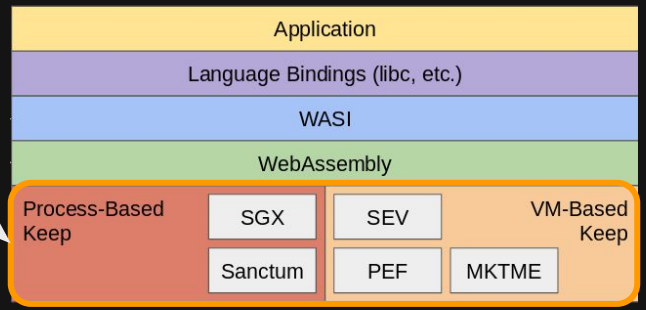
1. Minimal Trusted Computing Base
2. Minimum trust relationships
3. Deployment-time portability
4. Network stack outside TCB
5. Security at rest, in transit and in use
6. Auditability
7. Open source
8. Open standards
9. Memory safety
10. No backdoors

Enarx Architecture



Keep - process or VM-based

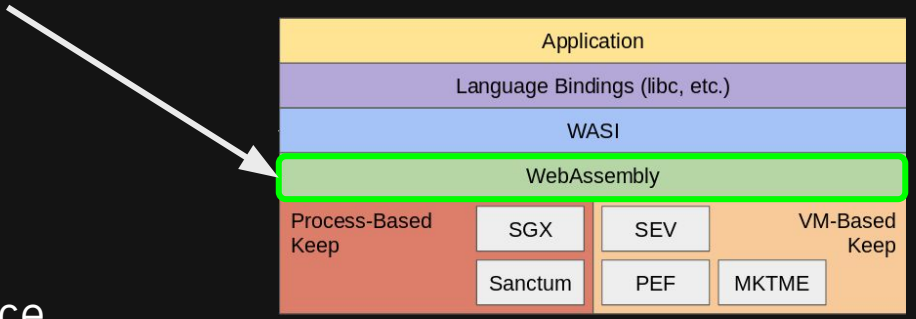
- Core Keep
- Platform-specific
  - Hardware (CPU): silicon vendor
  - Firmware: silicon vendor
  - Software: Enarx



Architecture varies between VM/Process-based platforms

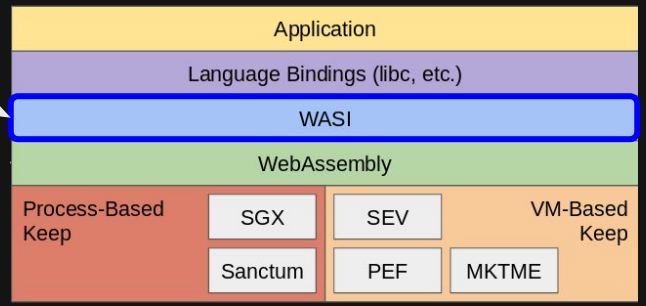
WebAssembly (WASM)

- W3C standard
- Stack Machine ISA
- Sandboxed
- Supported by all browsers
- Exploding in the “serverless” space
- Implementations improving rapidly
  - cranelift and wasmtime



WebAssembly System API (WASI)

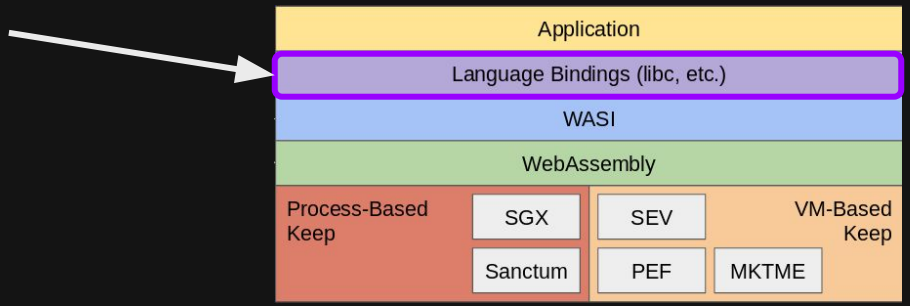
- W3C Standards Track
- Heavily inspired by a subset of POSIX
- Primary goals:
  - Portability
  - Security
- libc implementation on top
- Capability-based security:
  - No absolute resources
  - Think: openat() but not open()



Language Bindings (libc, etc.)

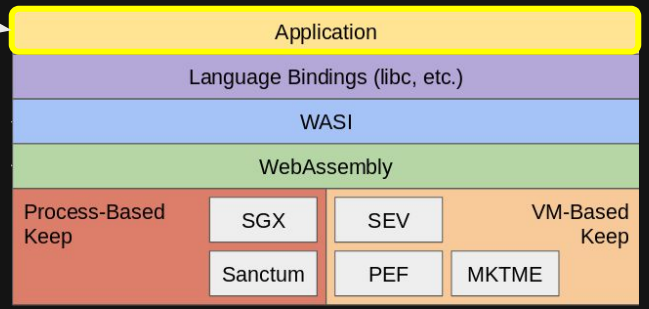
Compilation targets and includes, e.g.

- Rust: `--target wasm32-wasi`

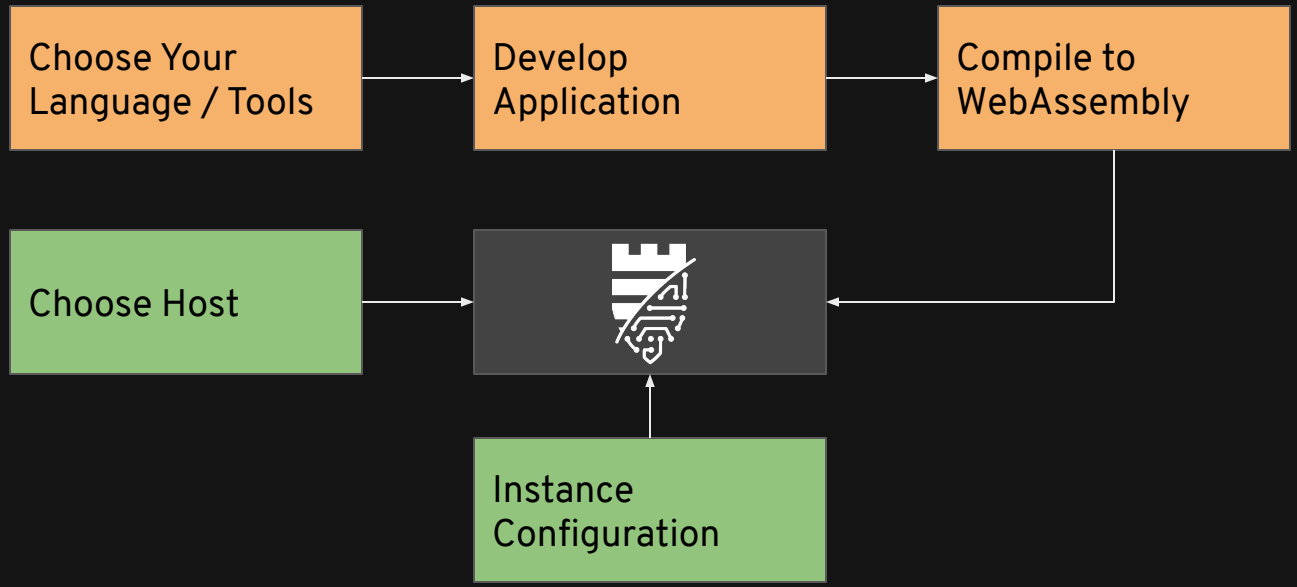


Application

- Written by
  - Tenant (own development)
    OR
  - 3rd party vendor
- Standard development tools
- Compiled to WebAssembly
- Using WASI interface

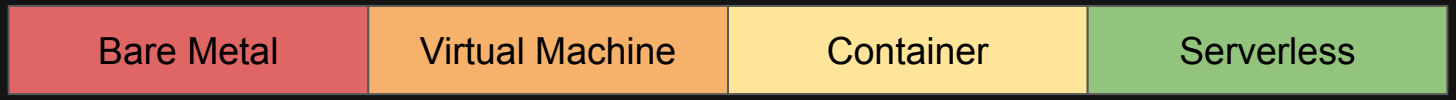


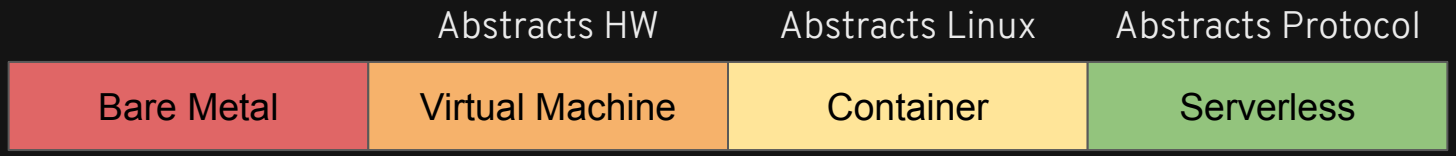
Enarx is a ~~Development~~ Deployment Framework

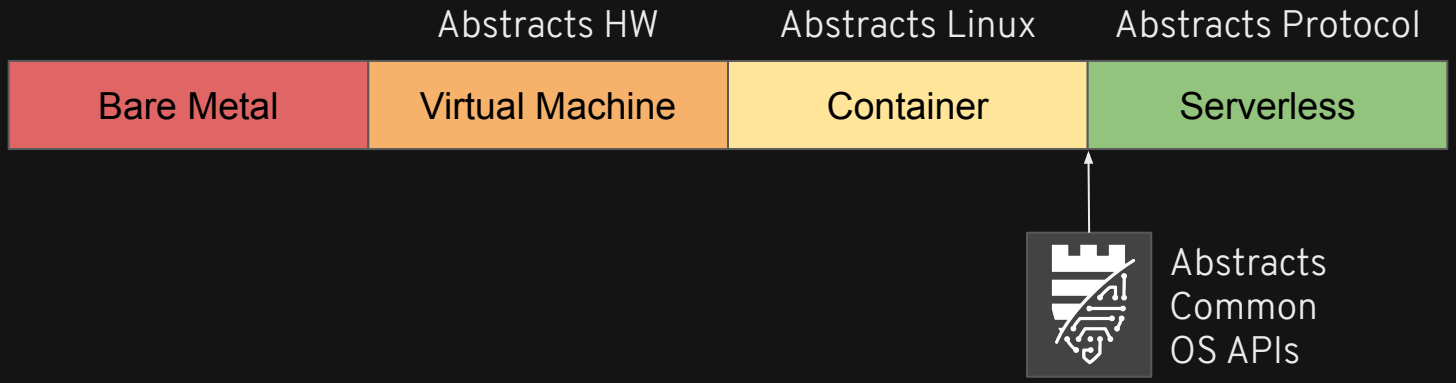


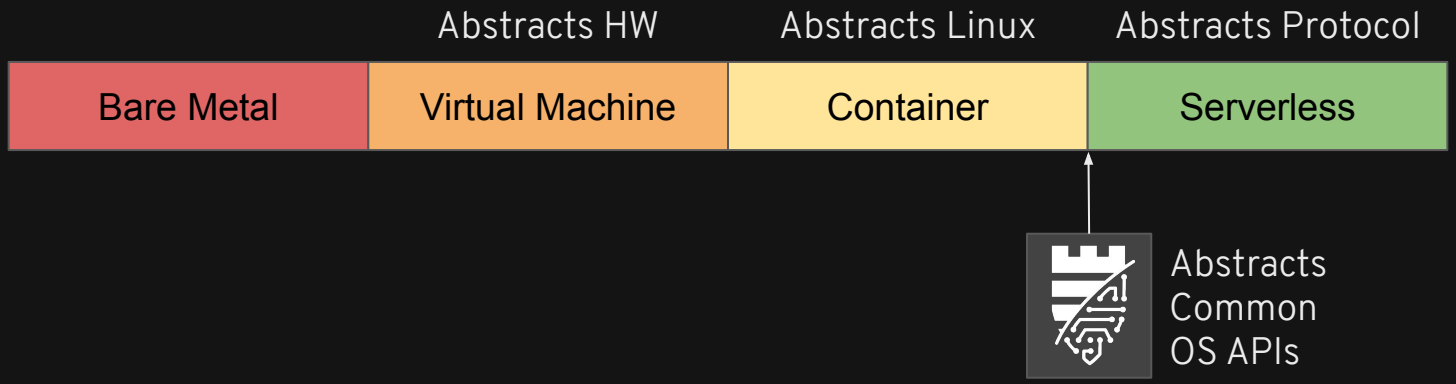
Best Practices? On By Default.

1. No Plaintext Networking (see **`cipherpipe`**)
2. No Plaintext Persisted Data
3. Independent Keep RNG
4. All Host APIs reviewed for data leakage


Bare Metal
Virtual Machine
Container
Serverless


Abstracts HW
Abstracts Linux
Abstracts Protocol
Bare Metal
Virtual Machine
Container
Serverless

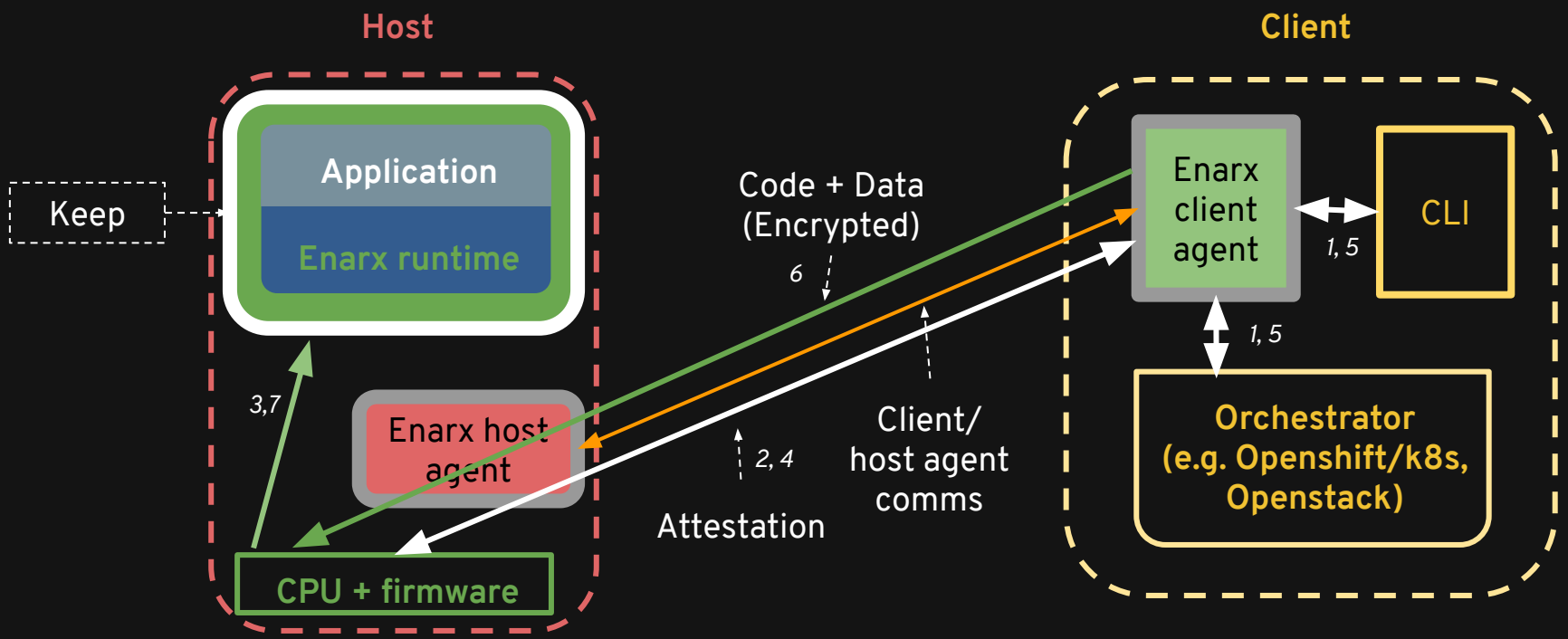

Abstracts HW
Abstracts Linux
Abstracts Protocol
Bare Metal
Virtual Machine
Container
Serverless
Abstracts
Common
OS APIs



Just enough legacy support to enable trivial application portability.
Homogeneity to enable radical deployment-time portability.
No interfaces which accidentally leak data to the host.
Bridges process-based and VM-based TEE models.
No operating system to manage.

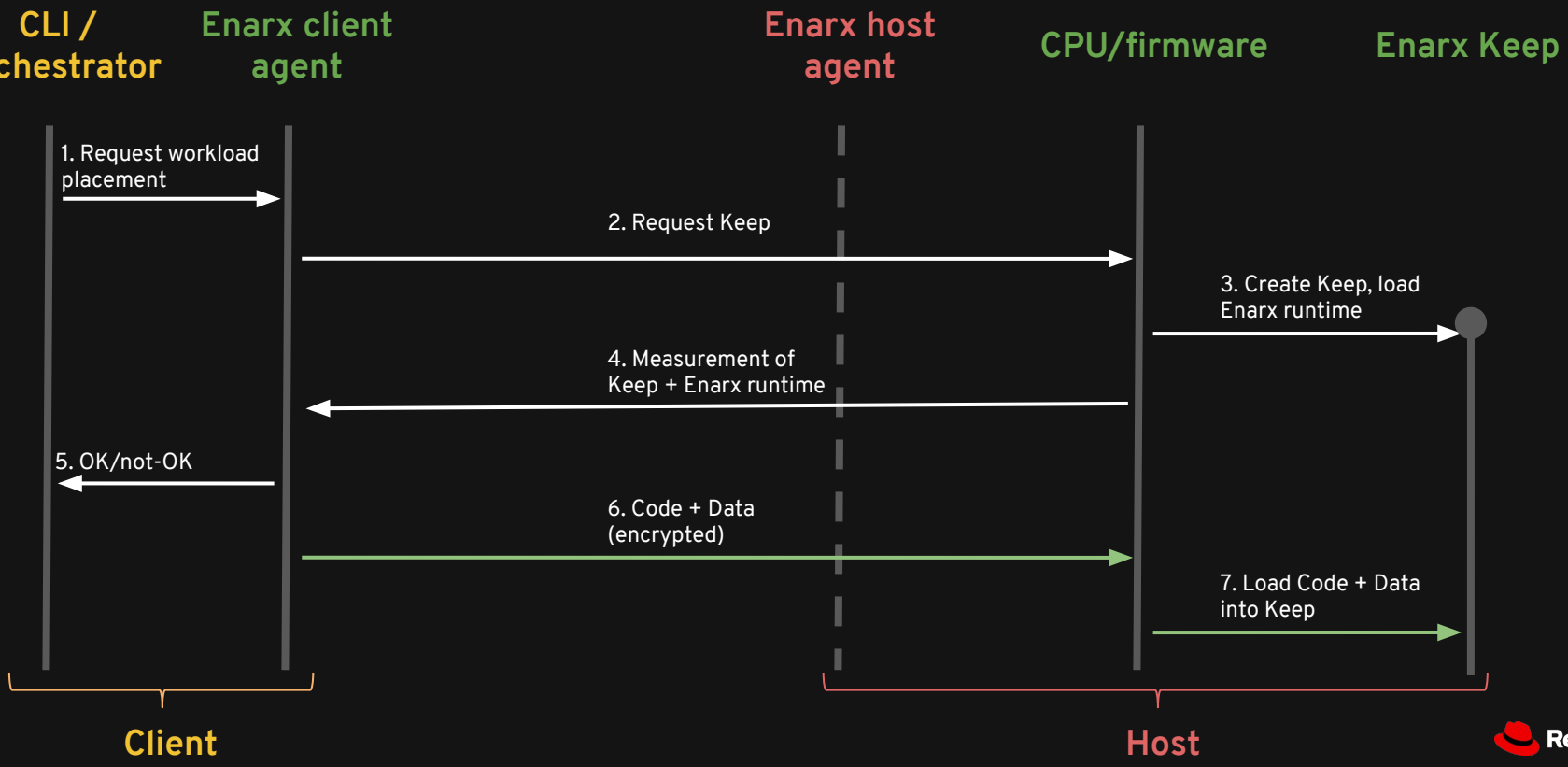
Process flow

Enarx architectural components



Red Hat

Enarx attestation process diagram

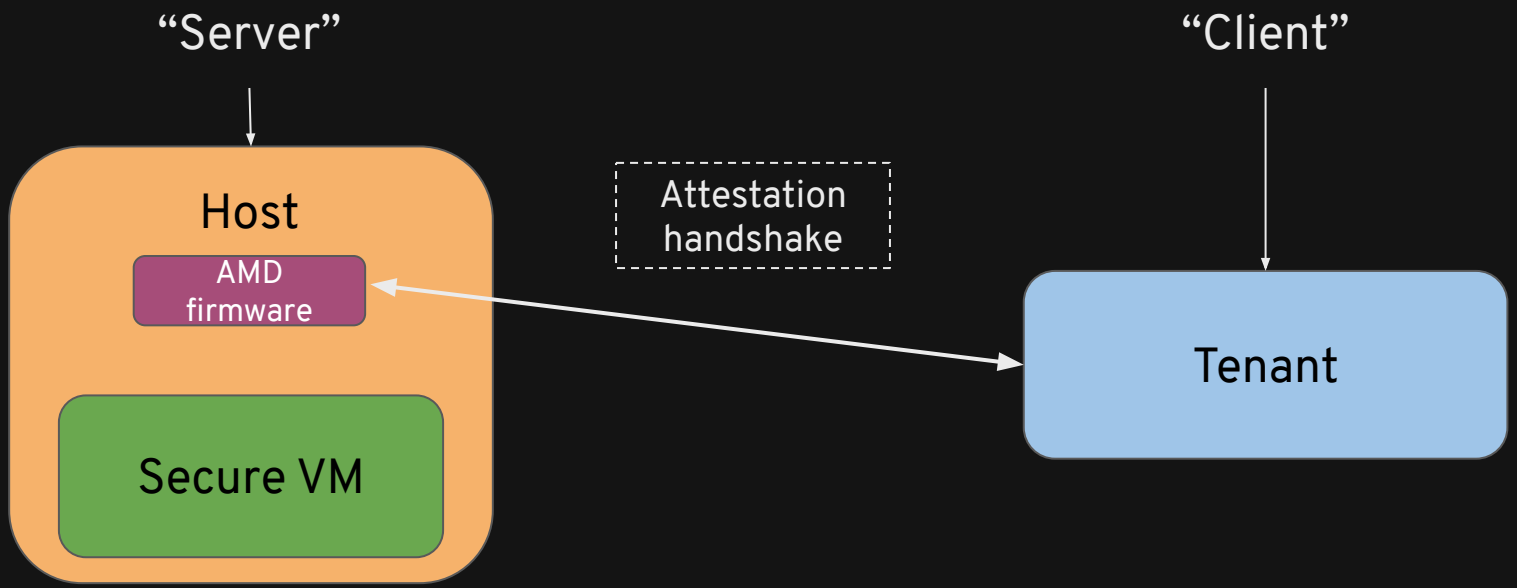


Red Hat

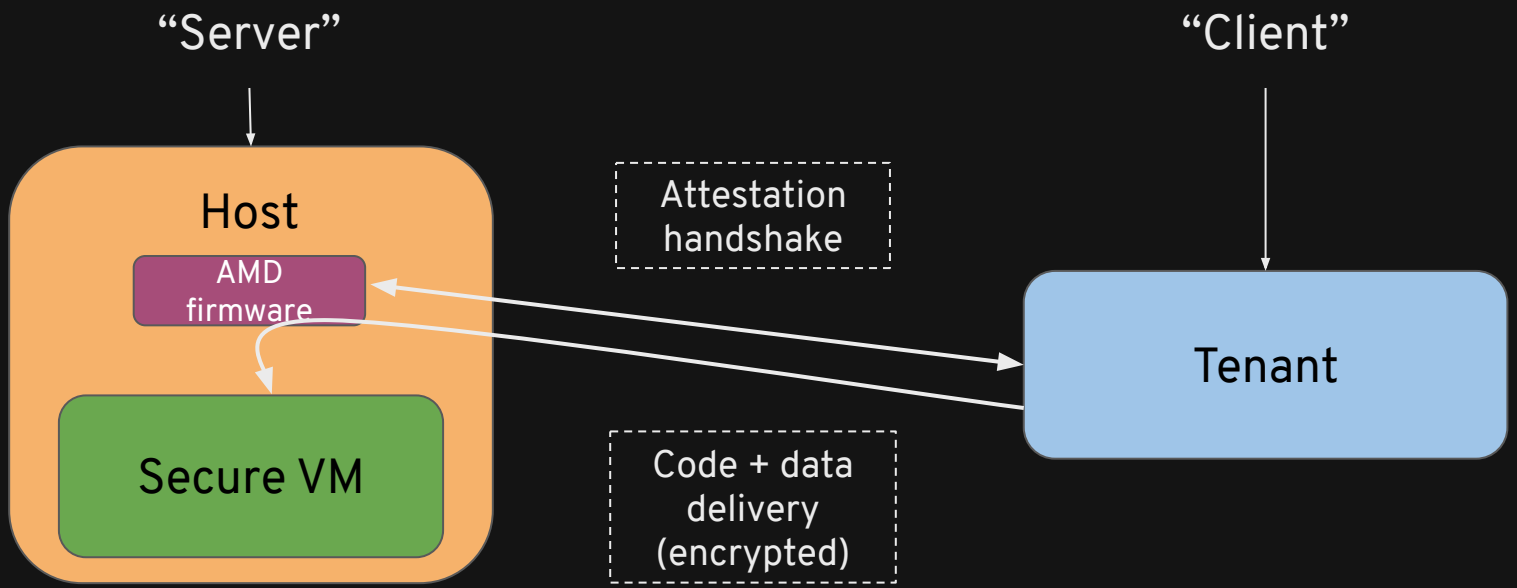
Demo Time!

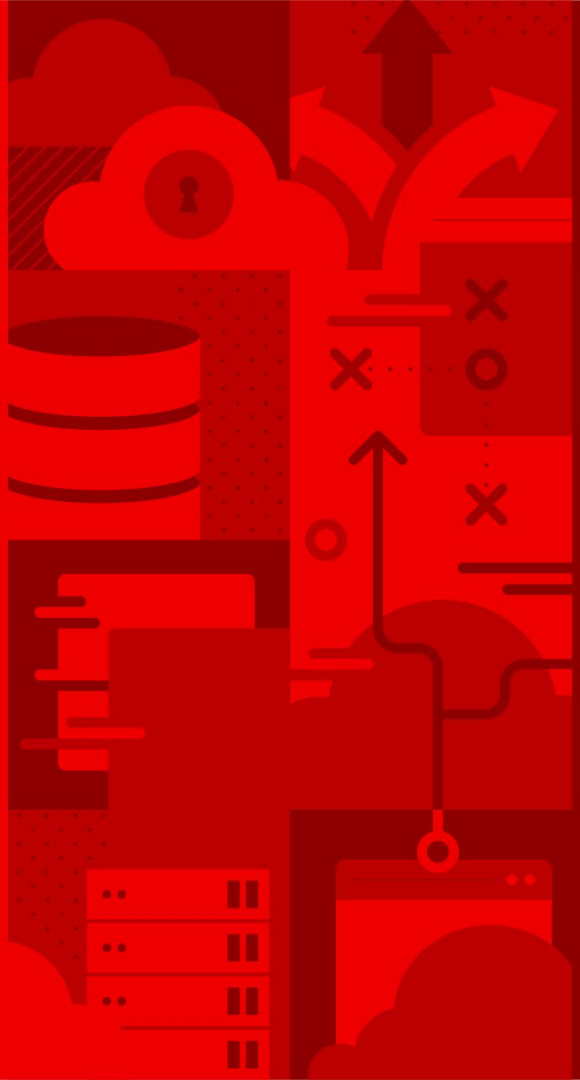
What will I see?

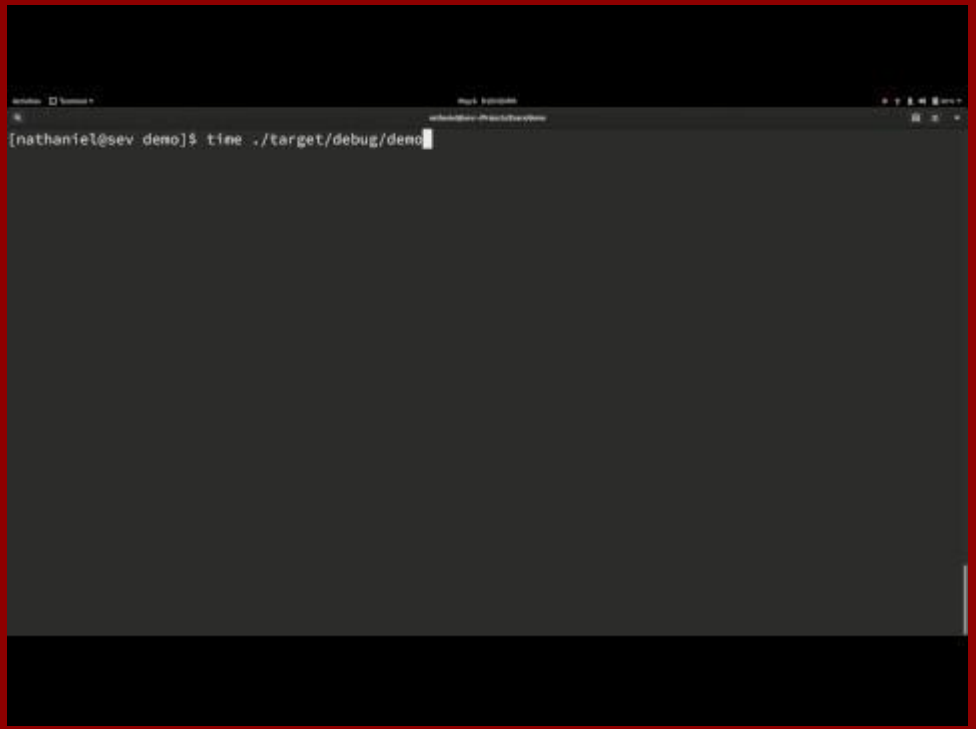
What will I see?



What will I see?

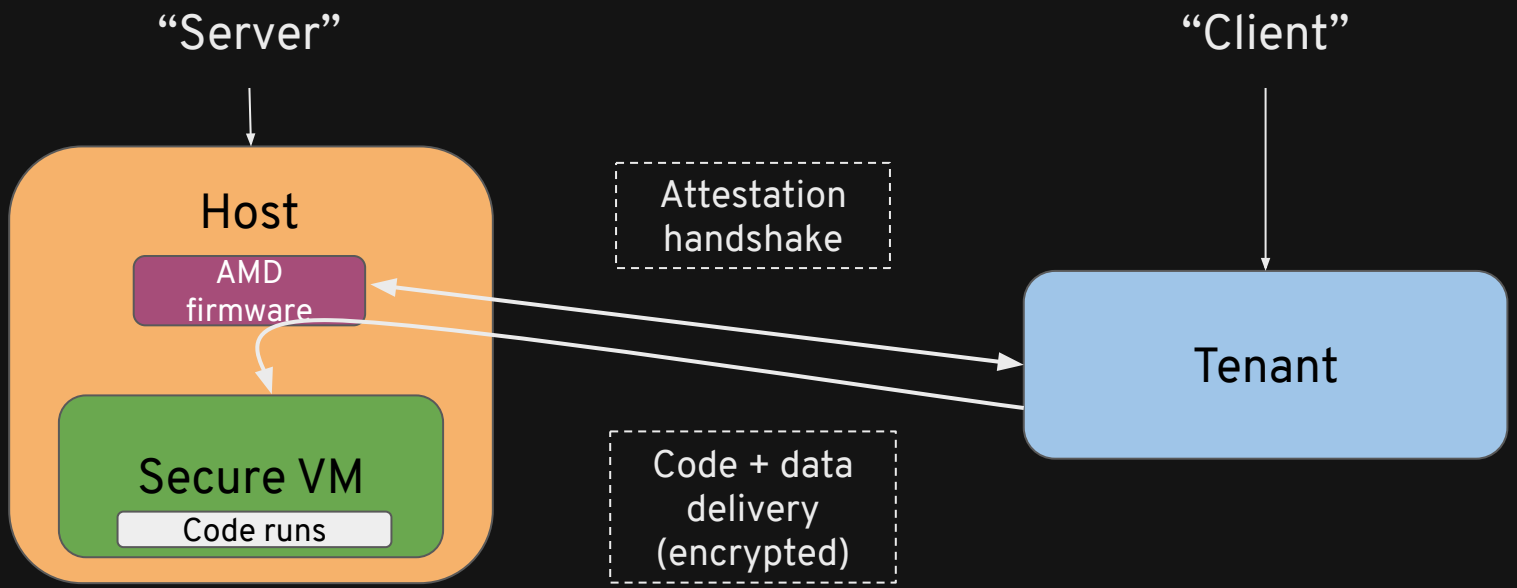




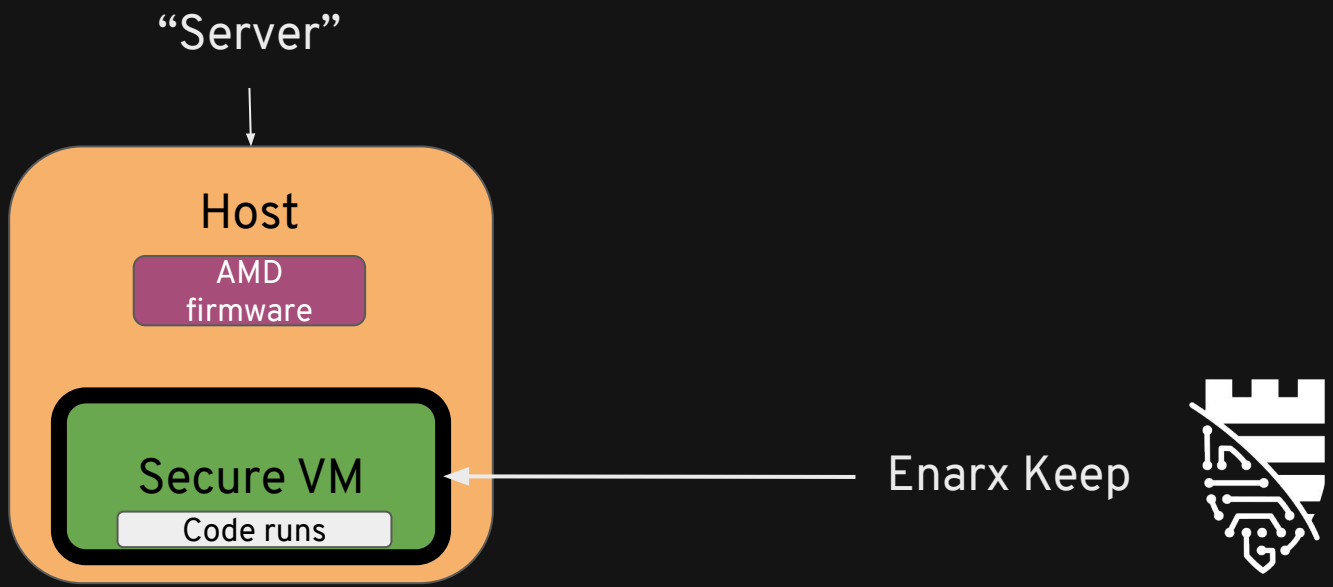


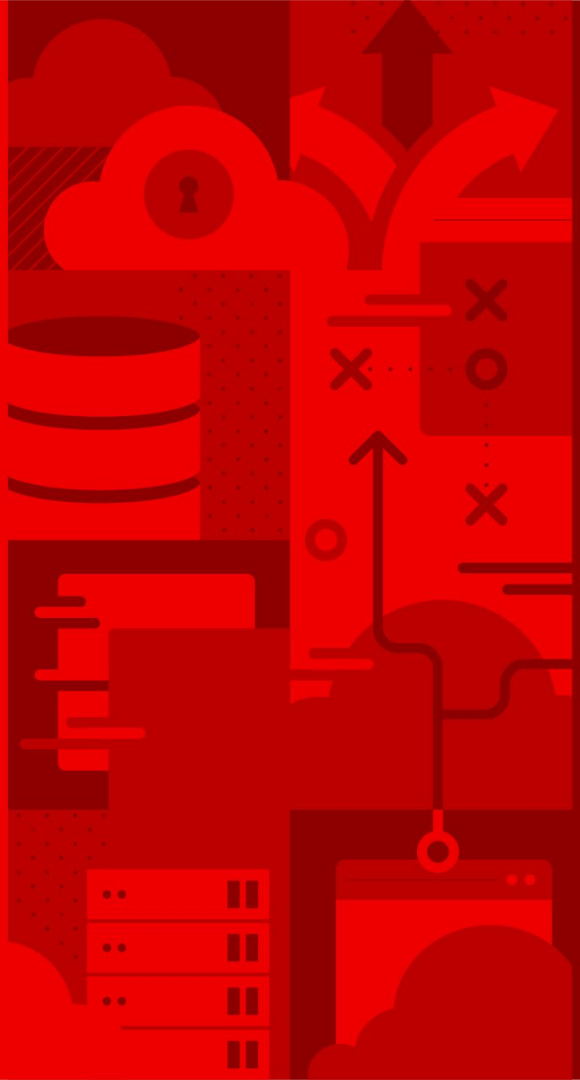
Red Hat

What did I see?



What did I see? (SEV)

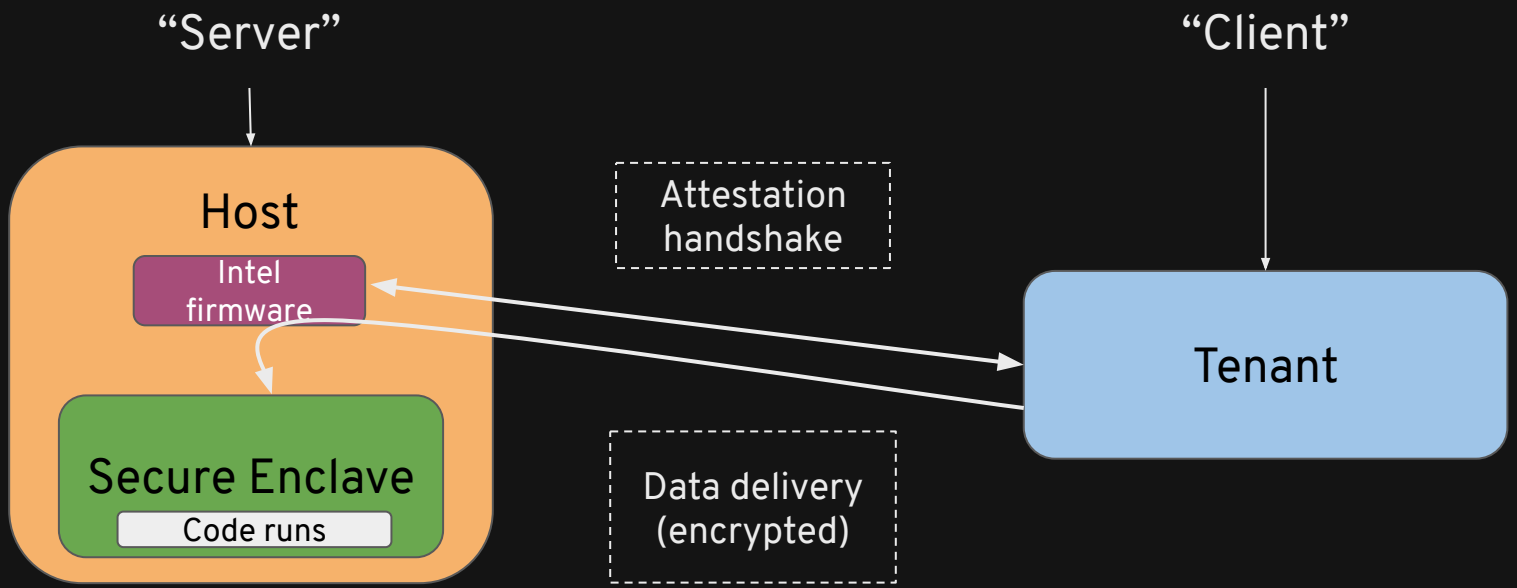




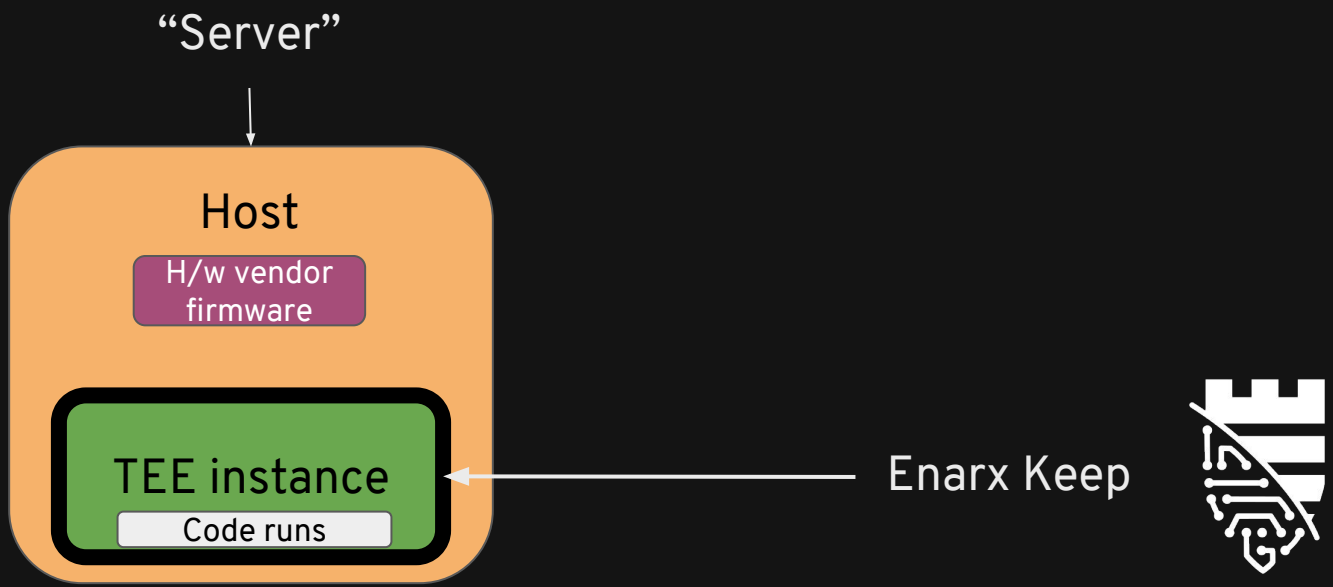
```
enarxnuc@enarxnuc:~$ cd Repos/enarx/demo/intel-sgx/
enarxnuc@enarxnuc:~/Repos/enarx/demo/intel-sgx$ time ./target/debug/attestation-tenant pck_cert_chain.pem 3 4
CLIENT < SERVER: Quote (Attestation)
CLIENT:          PCK cert chain OK
CLIENT:          Quote signature OK
CLIENT:          Attestation Key signature OK
CLIENT:          Enclave report hash OK

CLIENT:          Attestation Complete
CLIENT > SERVER: Tenant PubKey and Encrypted Data

7

real    0m0.647s
user    0m0.008s
sys     0m0.004s
enarxnuc@enarxnuc:~/Repos/enarx/demo/intel-sgx$
```


Red Hat

What did I see (SGX)?

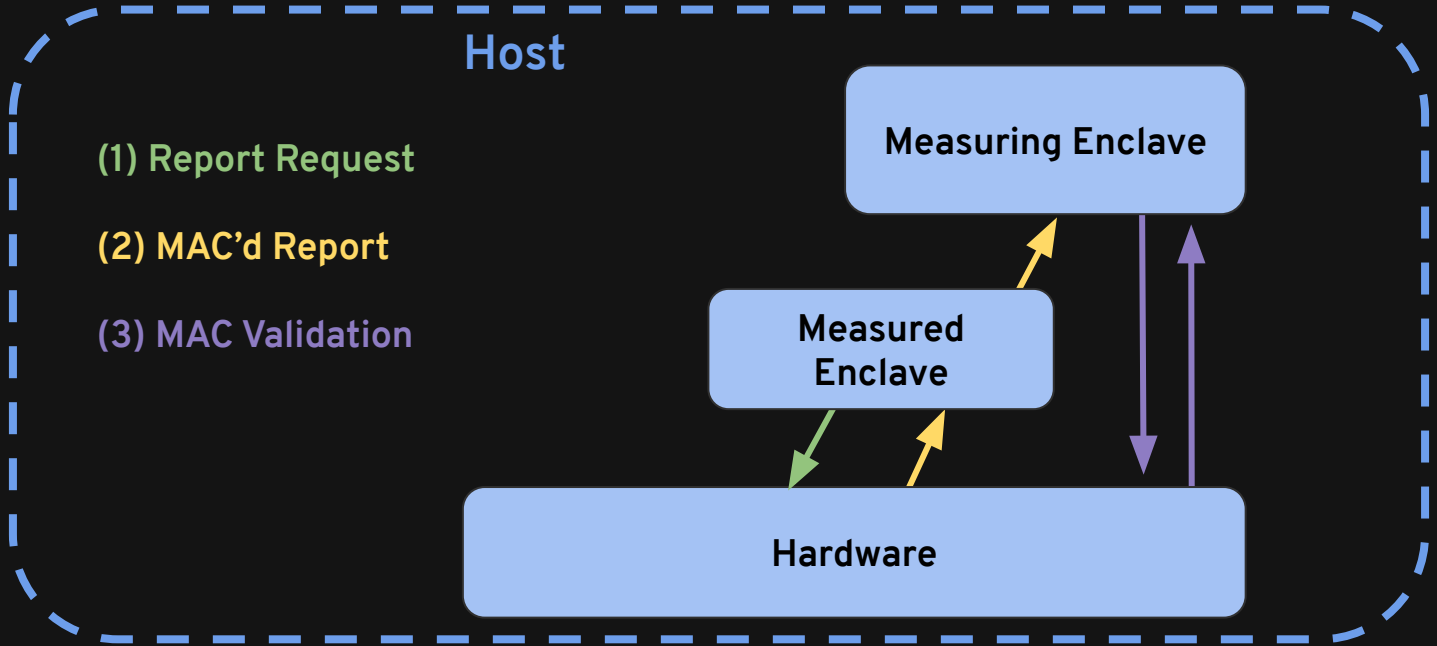


What did I see?

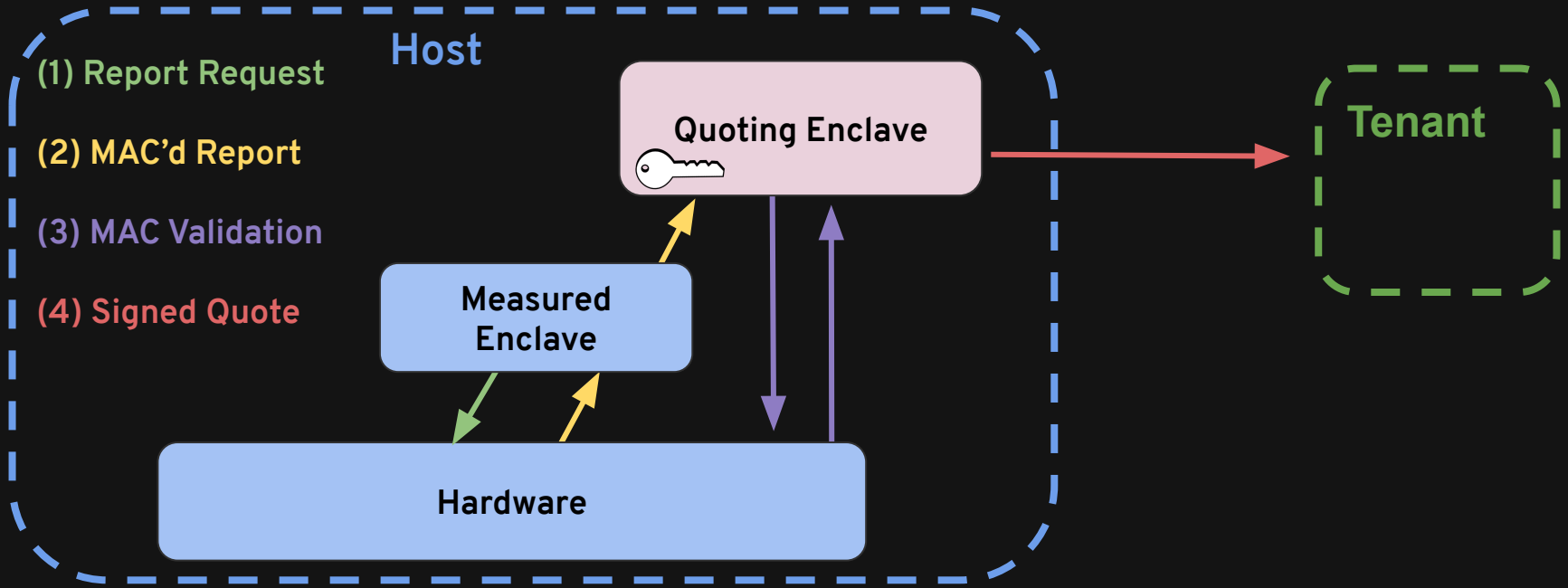


Intel SGX

SGX Local Attestation

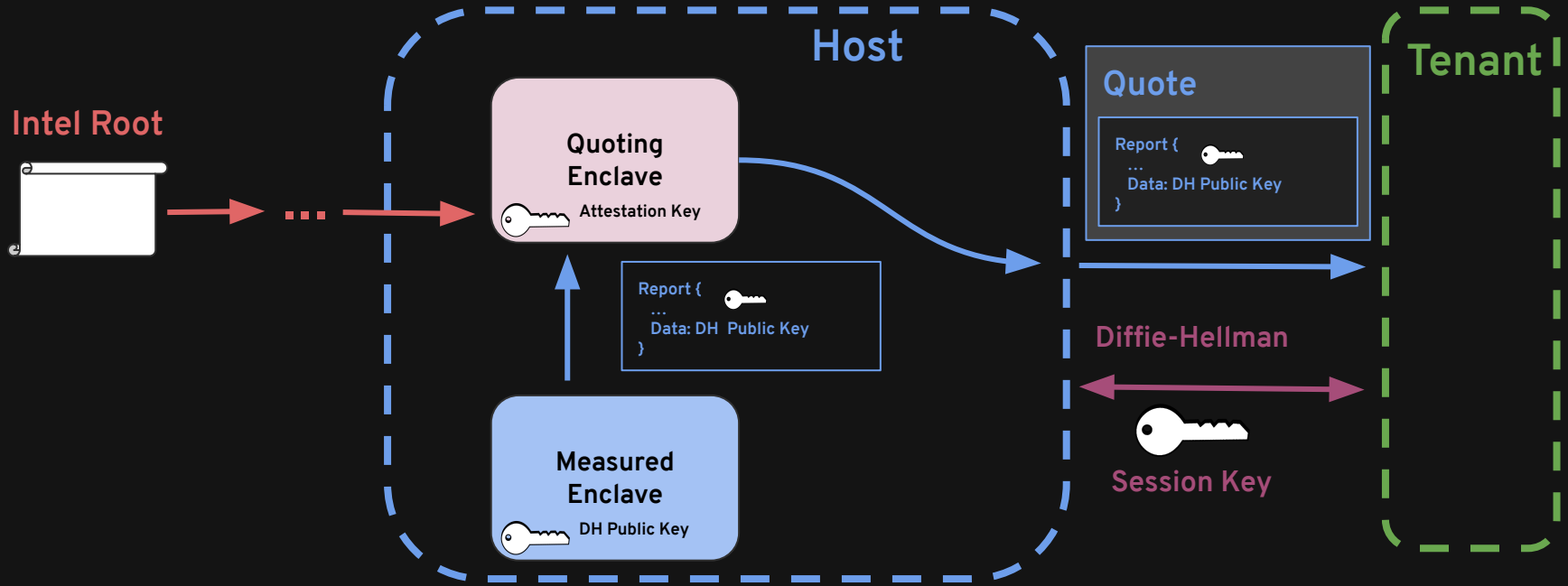


SGX Remote Attestation



Red Hat

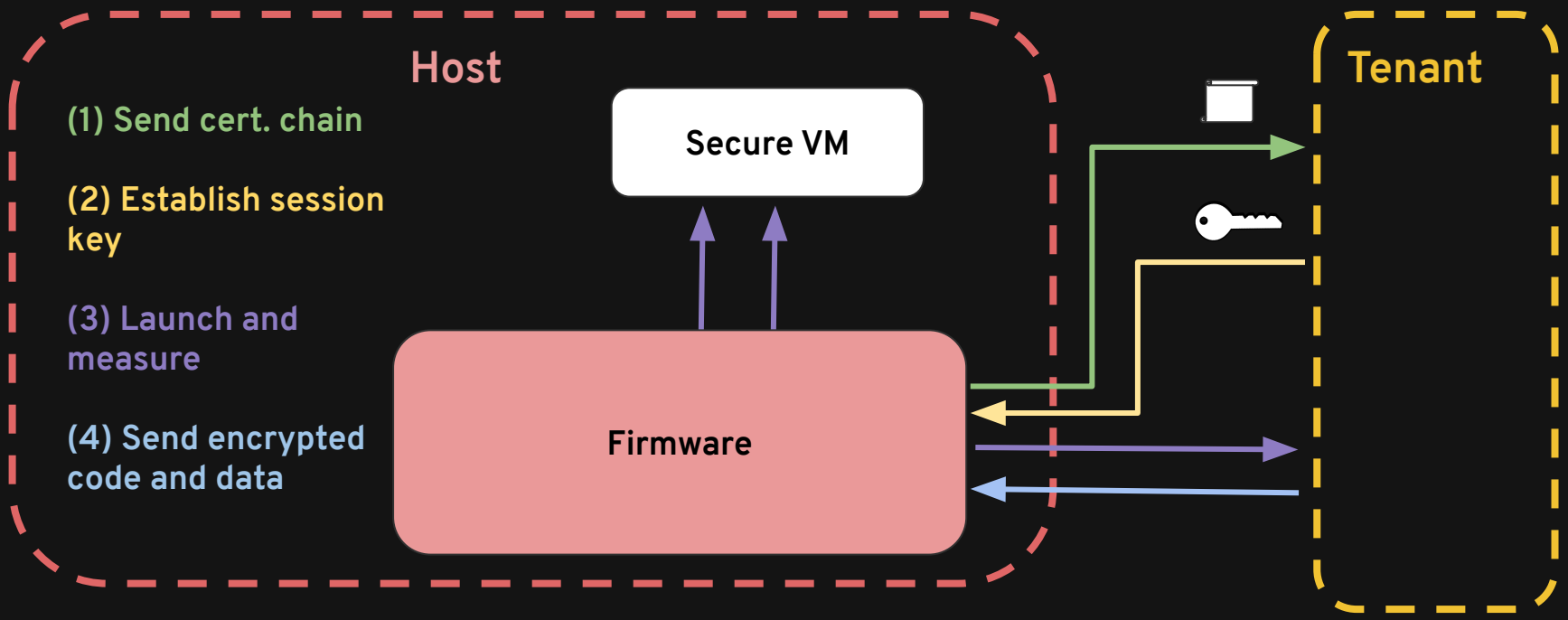
SGX Secure Session



Red Hat

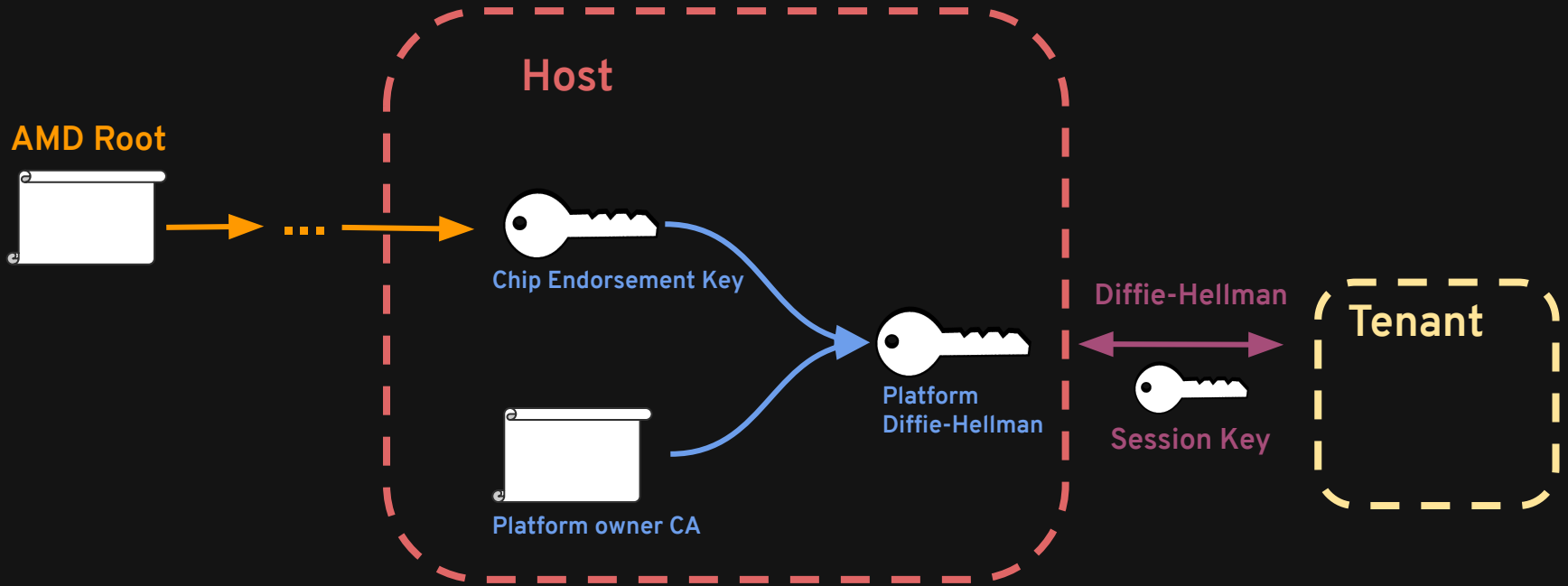
AMD SEV

SEV Attestation



Red Hat

SEV Secure Session



Red Hat

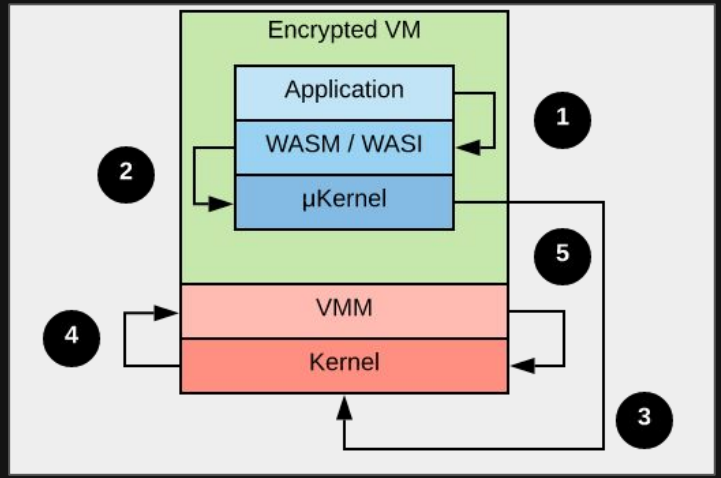
Enarx Virtualization Architecture

Red Hat

VM-based Keep

Syscall Propagation - VM-based

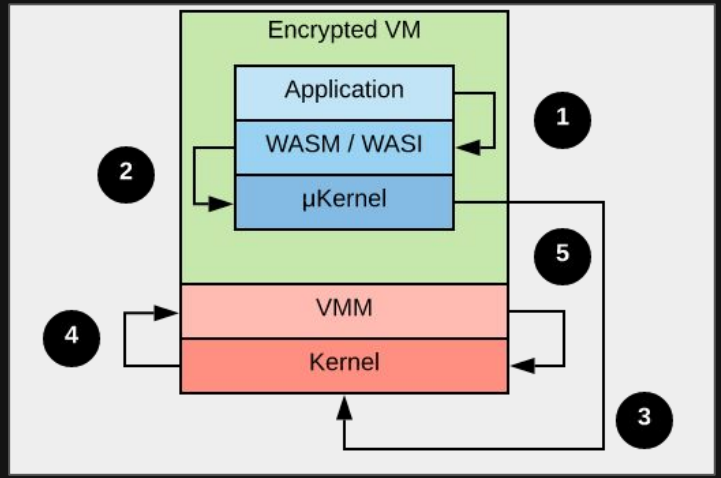
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.



https://github.com/enarx/enarx/wiki/Enarx-Syscall-Proxy

Syscall Propagation - VM-based

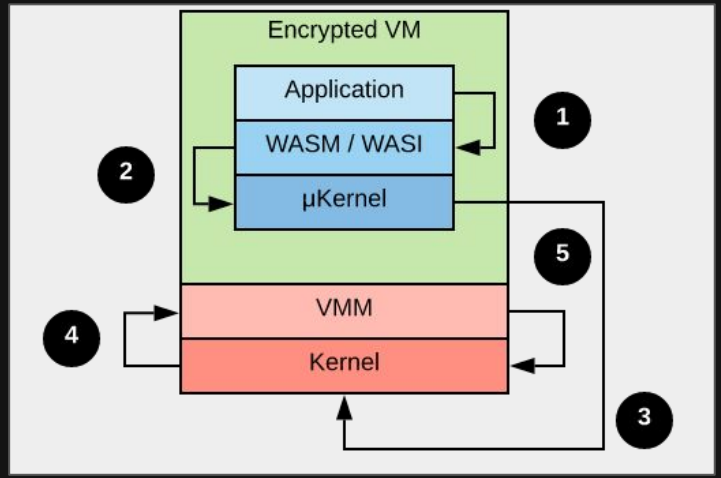
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.
2. The hand-crafted Rust code translates the WASI call into a Linux read() syscall, leaving Ring 3 to jump into the µKernel, which handles some syscalls internally.



https://github.com/enarx/enarx/wiki/Enarx-Syscall-Proxy

Syscall Propagation - VM-based

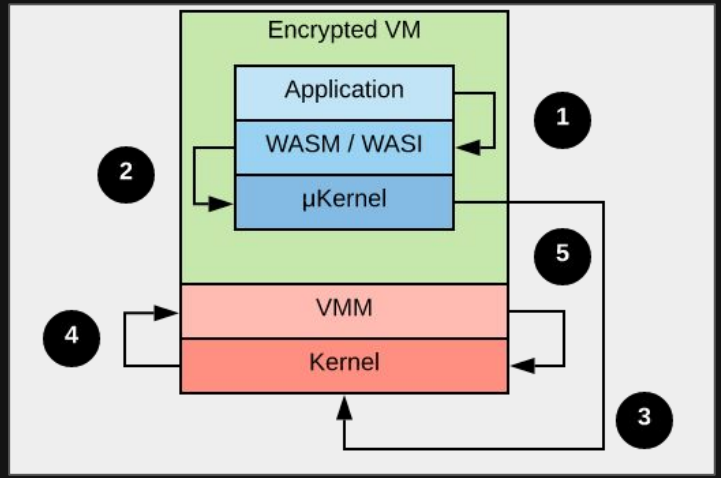
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.
2. The hand-crafted Rust code translates the WASI call into a Linux read() syscall, leaving Ring 3 to jump into the µKernel, which handles some syscalls internally.
3. (Future work) Guest µKernel passes the syscall request to the host (Linux) kernel. As an optimization, some syscalls may be handled by the host (Linux) kernel directly.



https://github.com/enarx/enarx/wiki/Enarx-Syscall-Proxy

Syscall Propagation - VM-based

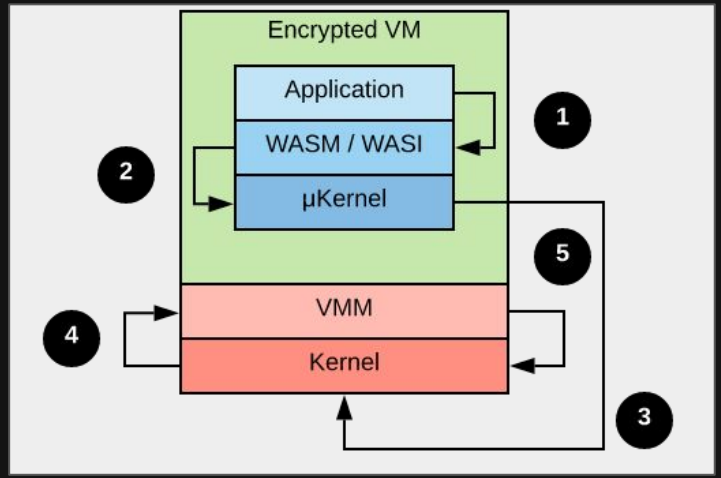
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.
2. The hand-crafted Rust code translates the WASI call into a Linux read() syscall, leaving Ring 3 to jump into the µKernel, which handles some syscalls internally.
3. (Future work) Guest µKernel passes the syscall request to the host (Linux) kernel. As an optimization, some syscalls may be handled by the host (Linux) kernel directly.
4. All syscalls which cannot be handled internally by the host kernel must cause a vmexit in the host VMM. Any syscalls which can be handled directly in the VMM are be handled immediately to avoid future context switches.



https://github.com/enarx/enarx/wiki/Enarx-Syscall-Proxy

Syscall Propagation - VM-based

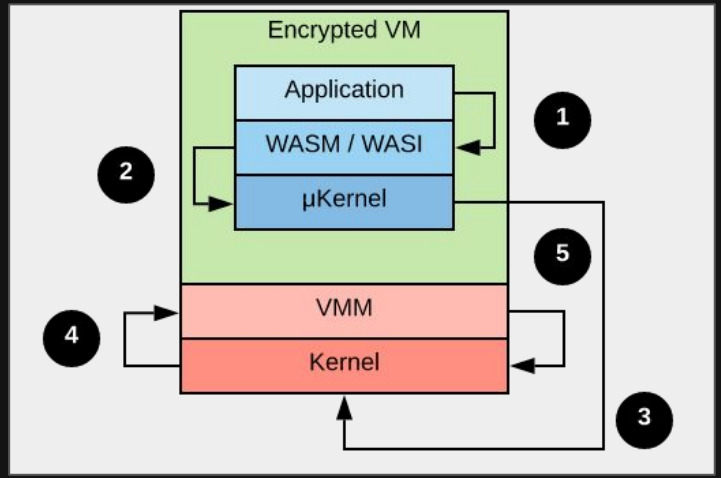
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.
2. The hand-crafted Rust code translates the WASI call into a Linux read() syscall, leaving Ring 3 to jump into the µKernel, which handles some syscalls internally.
3. (Future work) Guest µKernel passes the syscall request to the host (Linux) kernel. As an optimization, some syscalls may be handled by the host (Linux) kernel directly.
4. All syscalls which cannot be handled internally by the host kernel must cause a vmexit in the host VMM. Any syscalls which can be handled directly in the VMM are be handled immediately to avoid future context switches.
5. In some cases, the VMM will have to re-enter the host kernel in order to fulfil the request. This is the slowest performance path and should be avoided wherever possible.



https://github.com/enarx/enarx/wiki/Enarx-Syscall-Proxy

Syscall Propagation - VM-based

1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.
2. The hand-crafted Rust code translates the WASI call into a Linux read() syscall, leaving Ring 3 to jump into the µKernel, which handles some syscalls internally.
3. (Future work) Guest µKernel passes the syscall request to the host (Linux) kernel. As an optimization, some syscalls may be handled by the host (Linux) kernel directly.
4. All syscalls which cannot be handled internally by the host kernel must cause a vmexit in the host VMM. Any syscalls which can be handled directly in the VMM are be handled immediately to avoid future context switches.
5. In some cases, the VMM will have to re-enter the host kernel in order to fulfil the request. This is the slowest performance path and should be avoided wherever possible.



https://github.com/enarx/enarx/wiki/Enarx-Syscall-Proxy

Enarx Status

Red Hat

Current Status

1. SEV: Fully attested demo w/ custom assembly.
    a. Ketuvim: KVM library with SEV support
2. SGX: Fully attested demo w/ data delivery.
3. PEF: Ongoing discussions with POWER team.
4. WASM/WASI: Demo with some basic WASI functions.

Still To Do

https://github.com/enarx/enarx/issues/1

- Merge Ketuvim with rust-vmm
- Build:
  - Hypervisor
  - µKernel
  - WASI syscall propagation?
- Complete WASM JIT
- TLS networking stack
- Secure Clock?
- Research new platforms
- Openshift integration
- Much more...

Enarx Design Principles

1. Minimal Trusted Computing Base
2. Minimum trust relationships
3. Deployment-time portability
4. Network stack outside TCB
5. Security at rest, in transit and in use
6. Auditability
7. Open source
8. Open standards
9. Memory safety
10. No backdoors

We Need Your Help!

Website: https://enarx.io

Code: https://github.com/enarx

Gitter: https://gitter.im/enarx/

Master plan: https://github.com/enarx/enarx/issues/1

License: Apache 2.0

Language: Rust

Questions?

https://enarx.io

Red Hat