Enarx

Protection for data in use

Mike Bursell
Office of the CTO

Nathaniel McCallum
Sr. Principal Engineer

https://enarx.dev

Red Hat

Enarx overview

Red Hat

The Enarx 5-bullet overview

Red Hat

The Enarx 5-bullet overview

- Uses TEEs (SGX, SEV, TDX, etc.) for confidential workloads

The Enarx 5-bullet overview

- Uses TEEs (SGX, SEV, TDX, etc.) for confidential workloads
- Easy development and deployment

The Enarx 5-bullet overview

- Uses TEEs (SGX, SEV, TDX, etc.) for confidential workloads
- Easy development and deployment
- Strong security design principles

Red Hat

The Enarx 5-bullet overview

- Uses TEEs (SGX, SEV, TDX, etc.) for confidential workloads
- Easy development and deployment
- Strong security design principles
- Cloud-native → Openshift, kubernetes

Red Hat

The Enarx 5-bullet overview

- Uses TEEs (SGX, SEV, TDX, etc.) for confidential workloads
- Easy development and deployment
- Strong security design principles
- Cloud-native → Openshift, kubernetes
- Open source: *project*, not production-ready (yet)

CONFIDENTIAL COMPUTING CONSORTIUM

Red Hat

The Problem

Red Hat

The Need for Confidentiality and Integrity

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

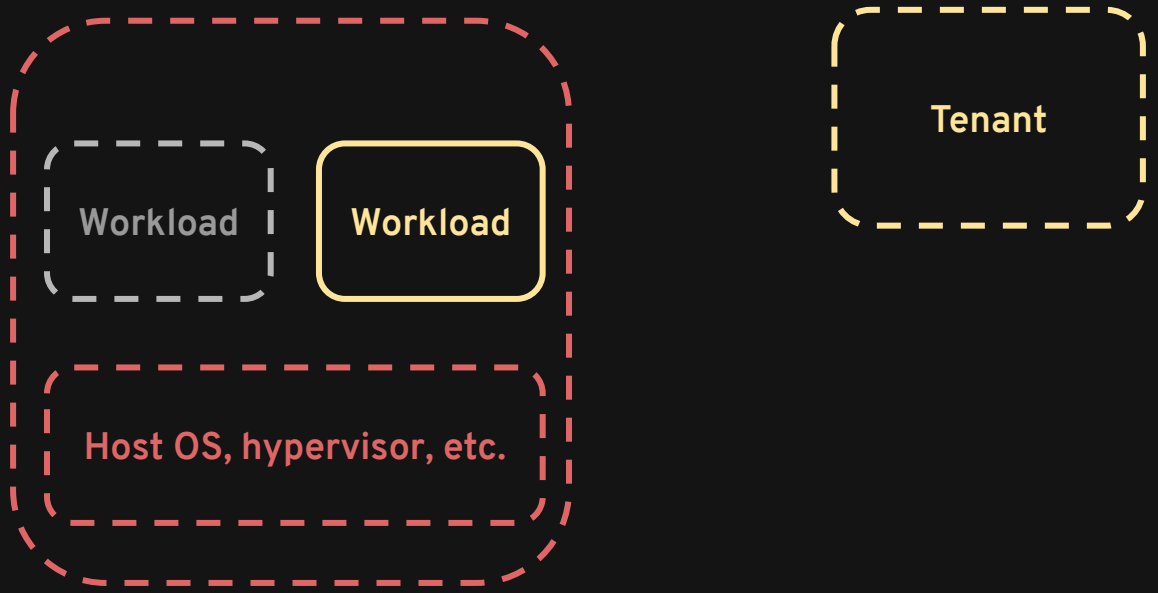
Red Hat

Isolation

Workloads and the host

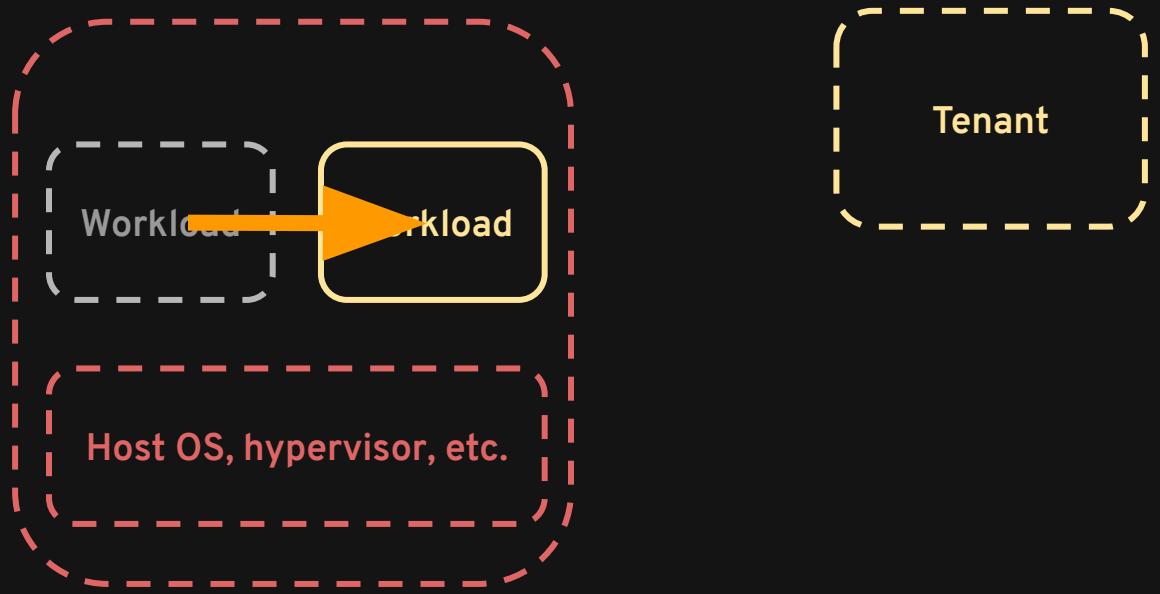
Red Hat

The 3 types of isolation

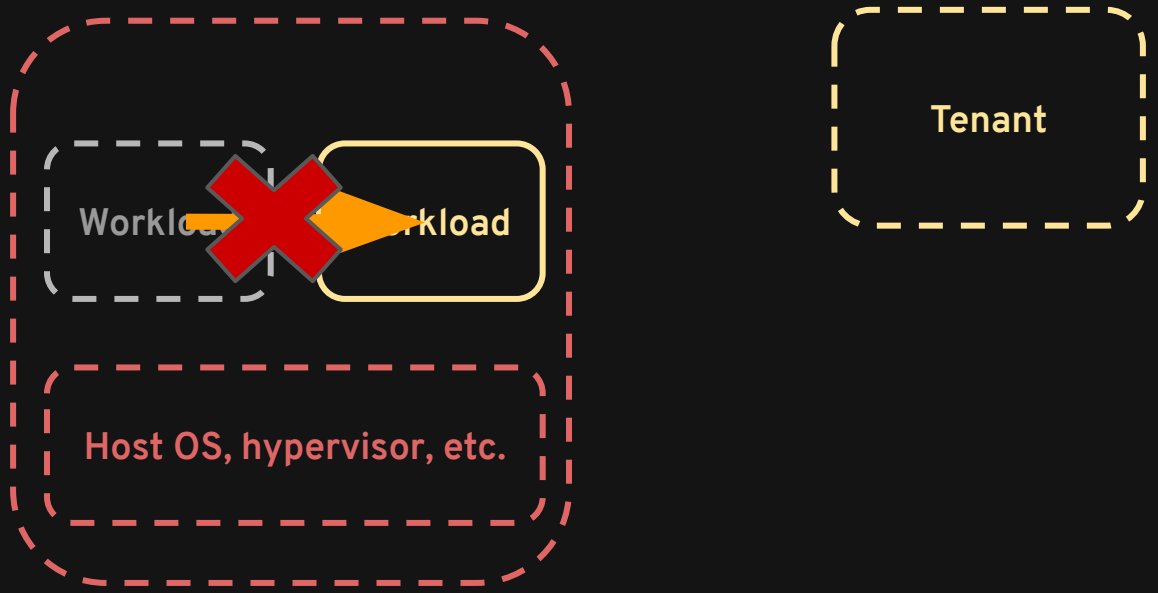


Red Hat

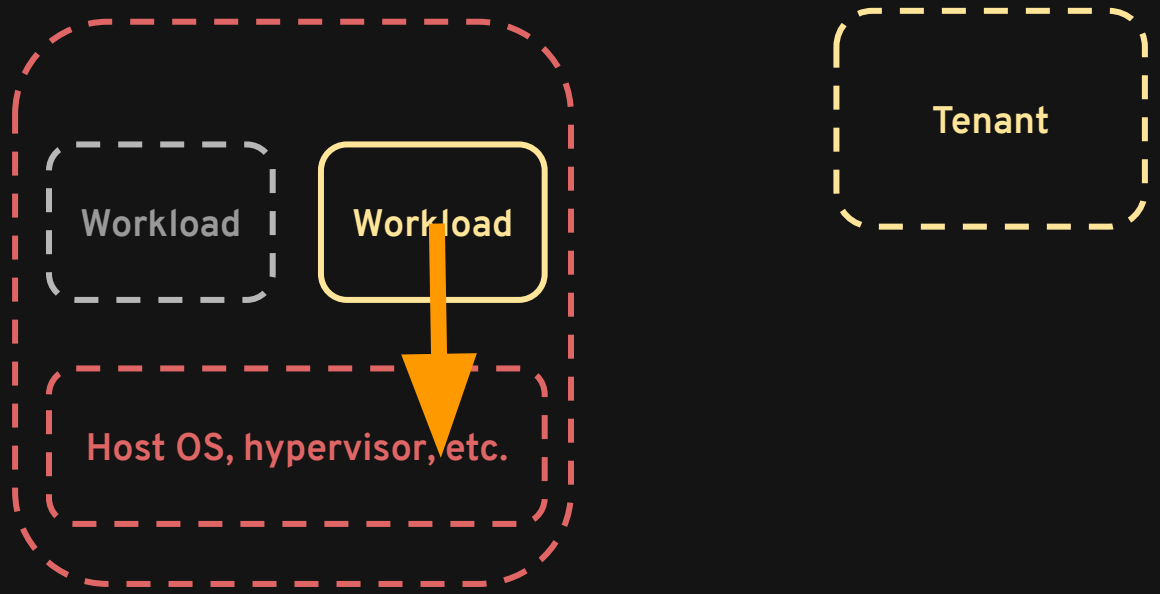
Workload from workload isolation



Workload from workload isolation

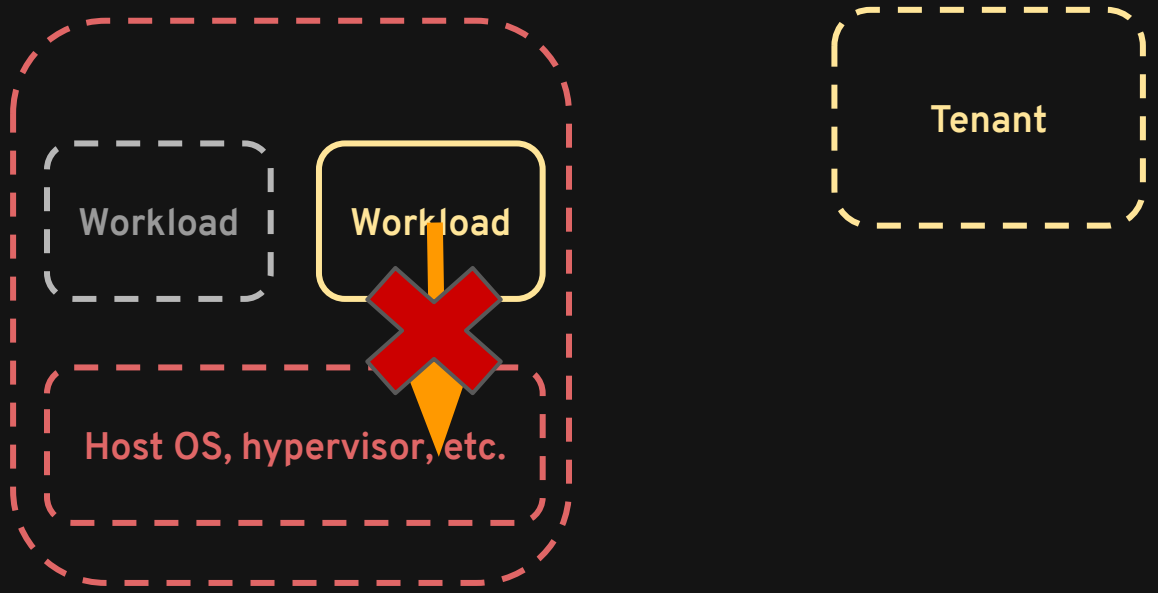


Host from workload isolation

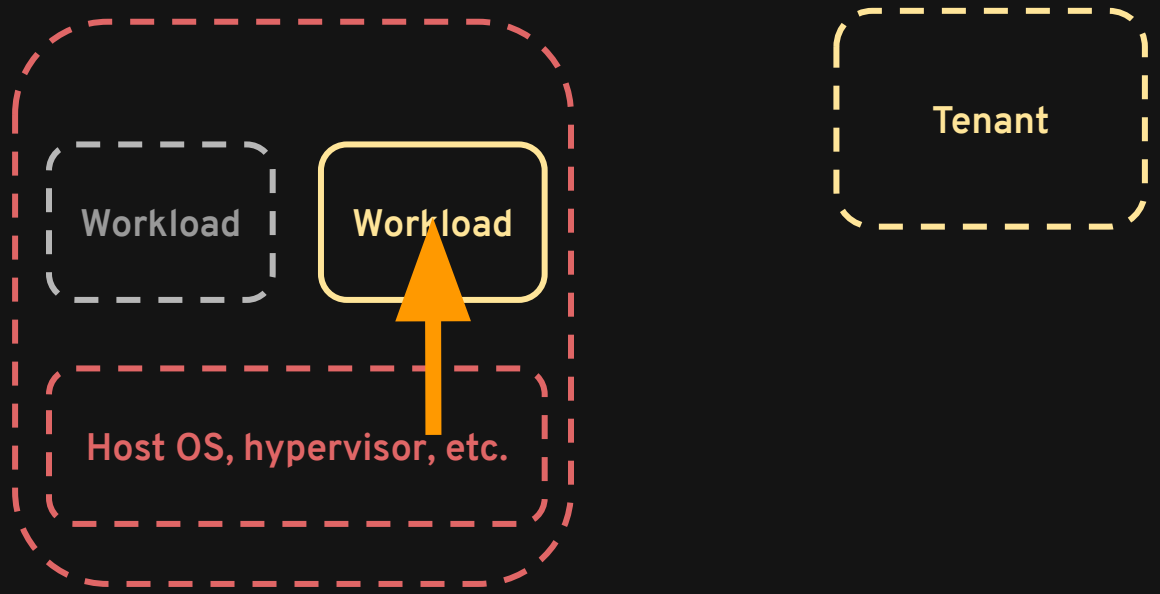


Red Hat

Host from workload isolation

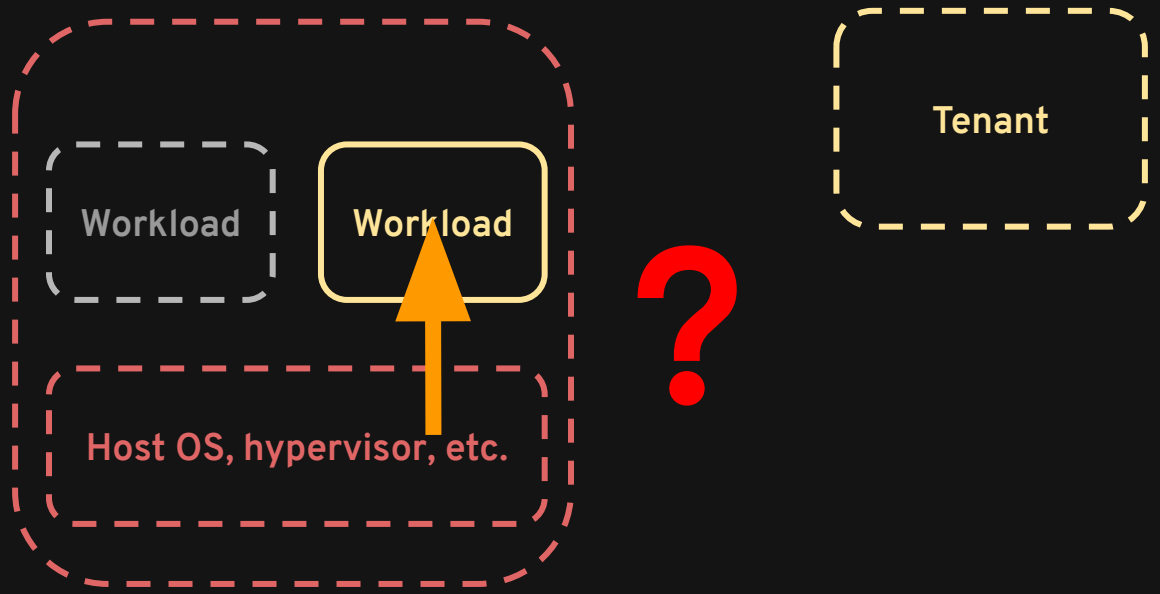


Workload from host isolation



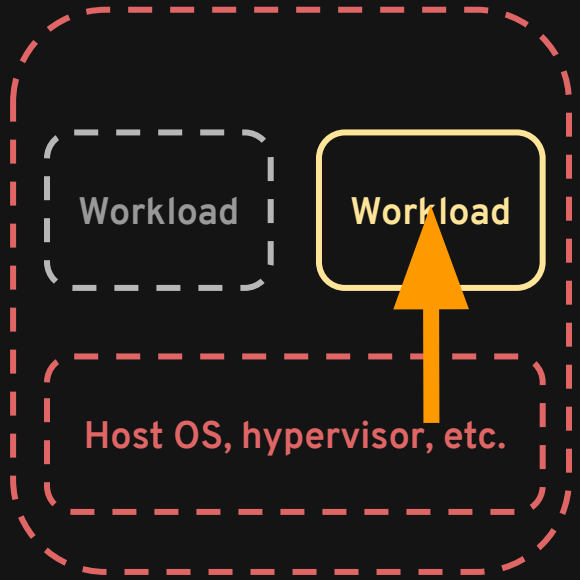
Red Hat

Workload from host isolation



Red Hat

Workload from host isolation



?

Sensitive workloads

Cloud for regulated sectors

- Healthcare, Finance, Government, Enterprise, ...

Vulnerable hosts

- Edge, IoT, ...

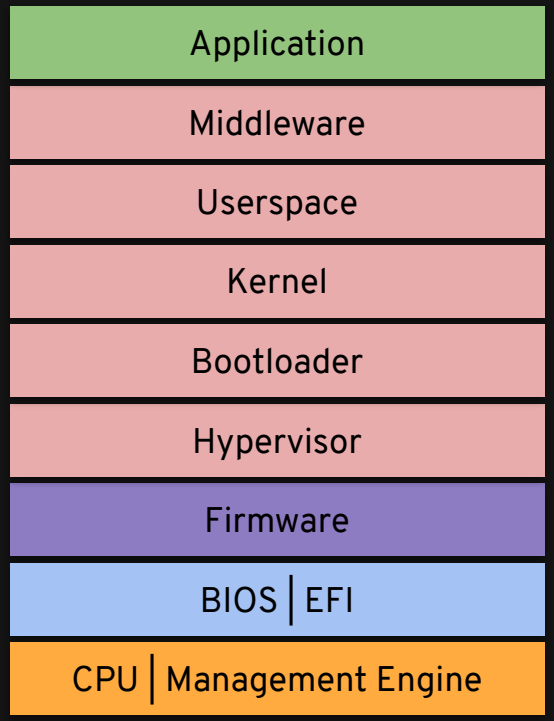
Red Hat

Isolation

The stack

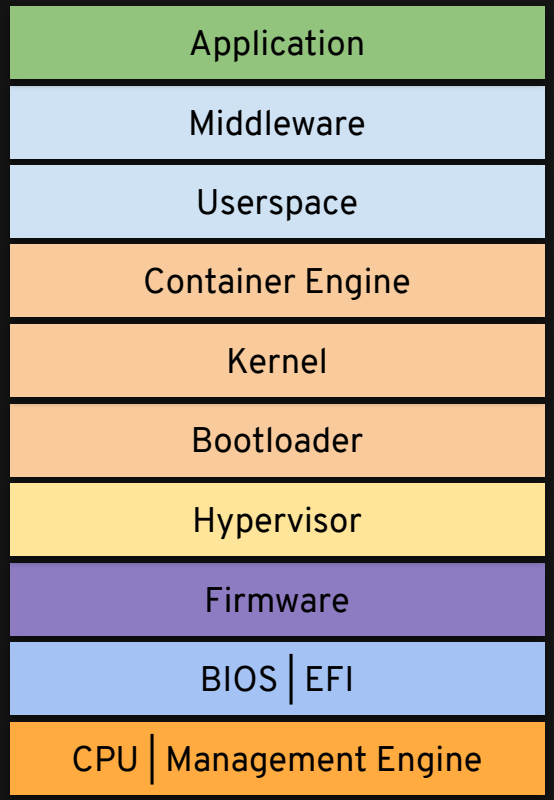
Red Hat

Virtualization Stack



Red Hat

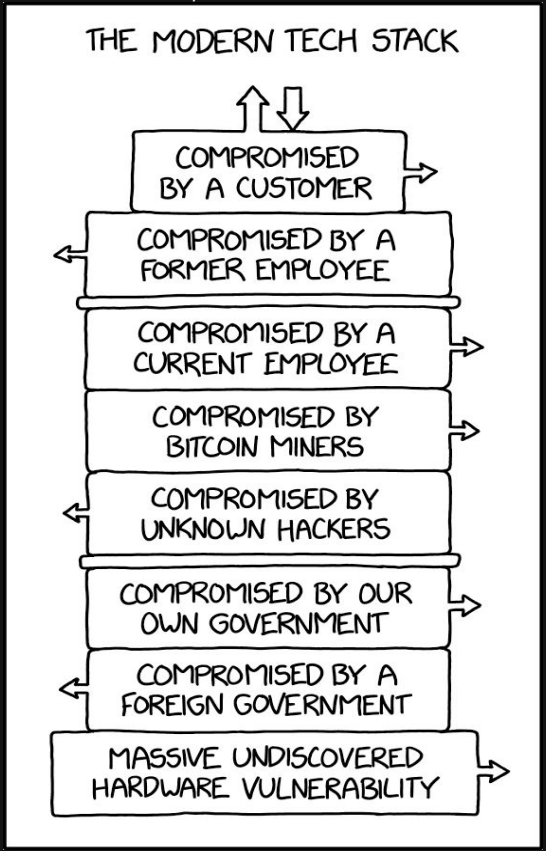
Container Stack



Red Hat

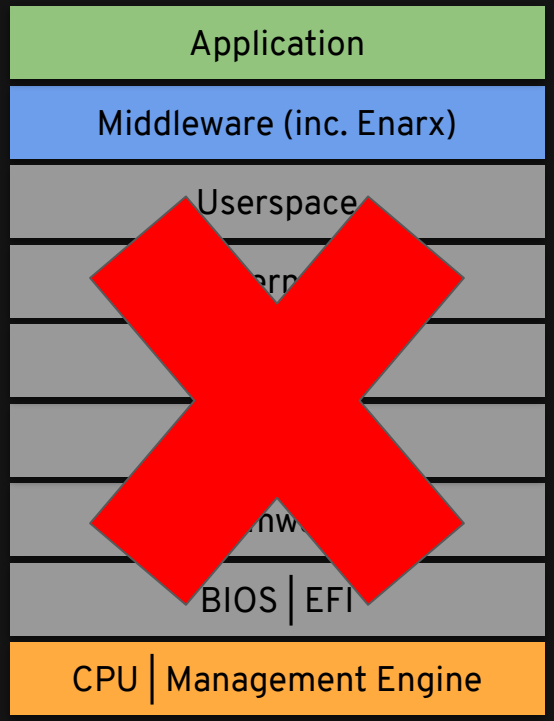
Virtualization Stack

as seen by xkcd (xkcd.com/2166)



Red Hat

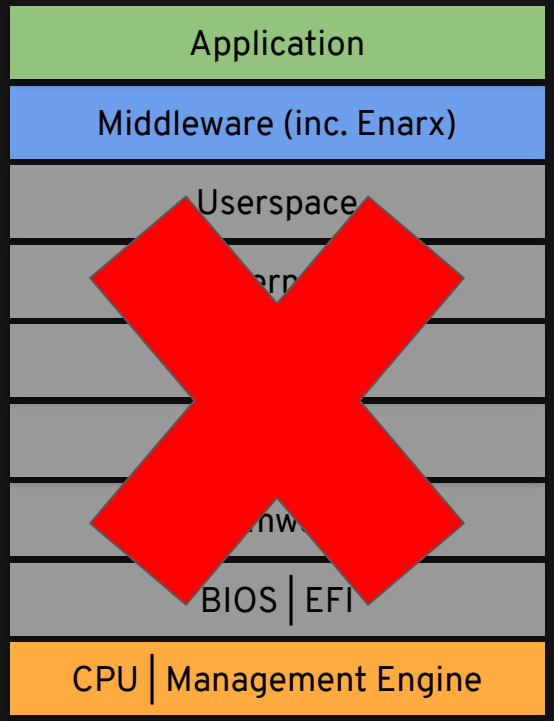
The Plan



Red Hat

The Principles

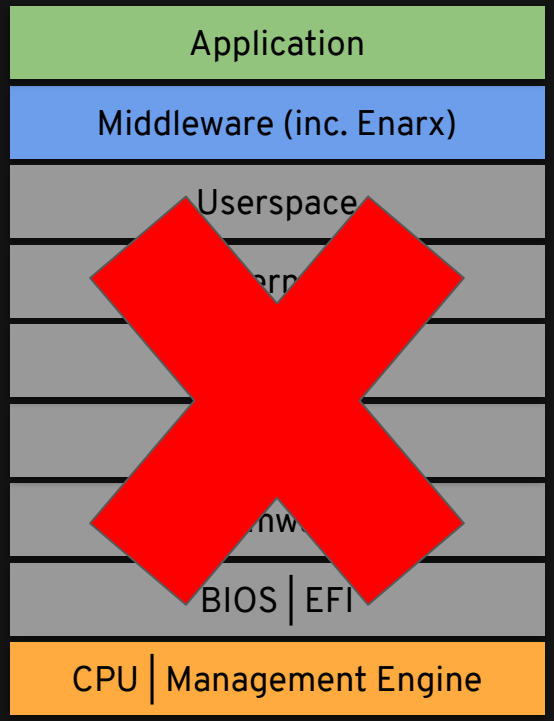
Don't trust the host
Don't trust the host owner
Don't trust the host operator
All hardware cryptographically verified
All software audited and cryptographically verified



Red Hat

The Fit

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

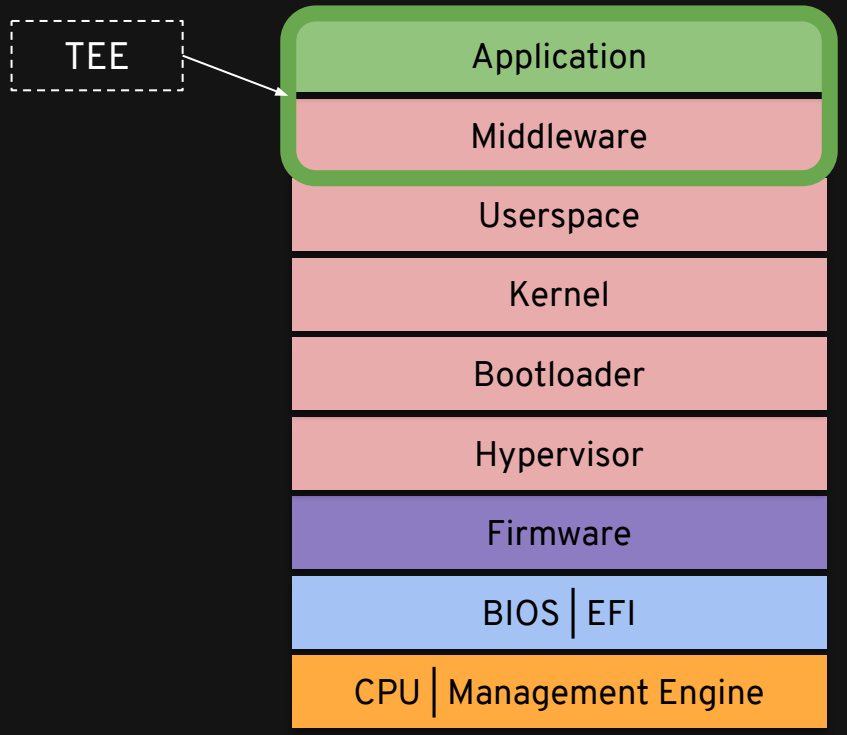


Well suited to microservices
Well suited to sensitive data or algorithms
Easy development integration
Simple deployment
Standards based: WebAssembly (WASM)

Trusted Execution Environments

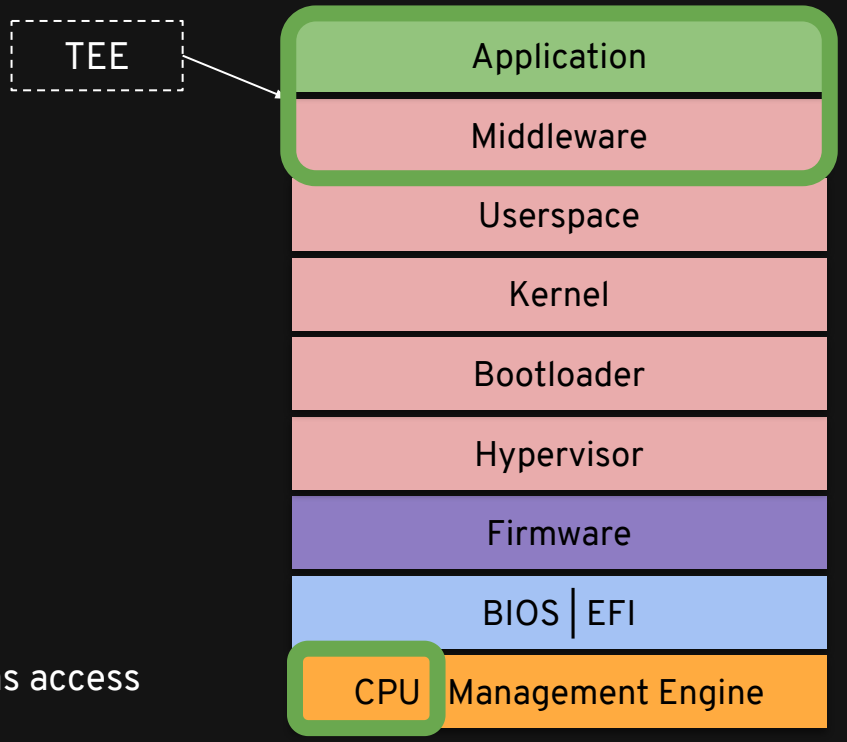
Red Hat

What's a TEE?



Red Hat

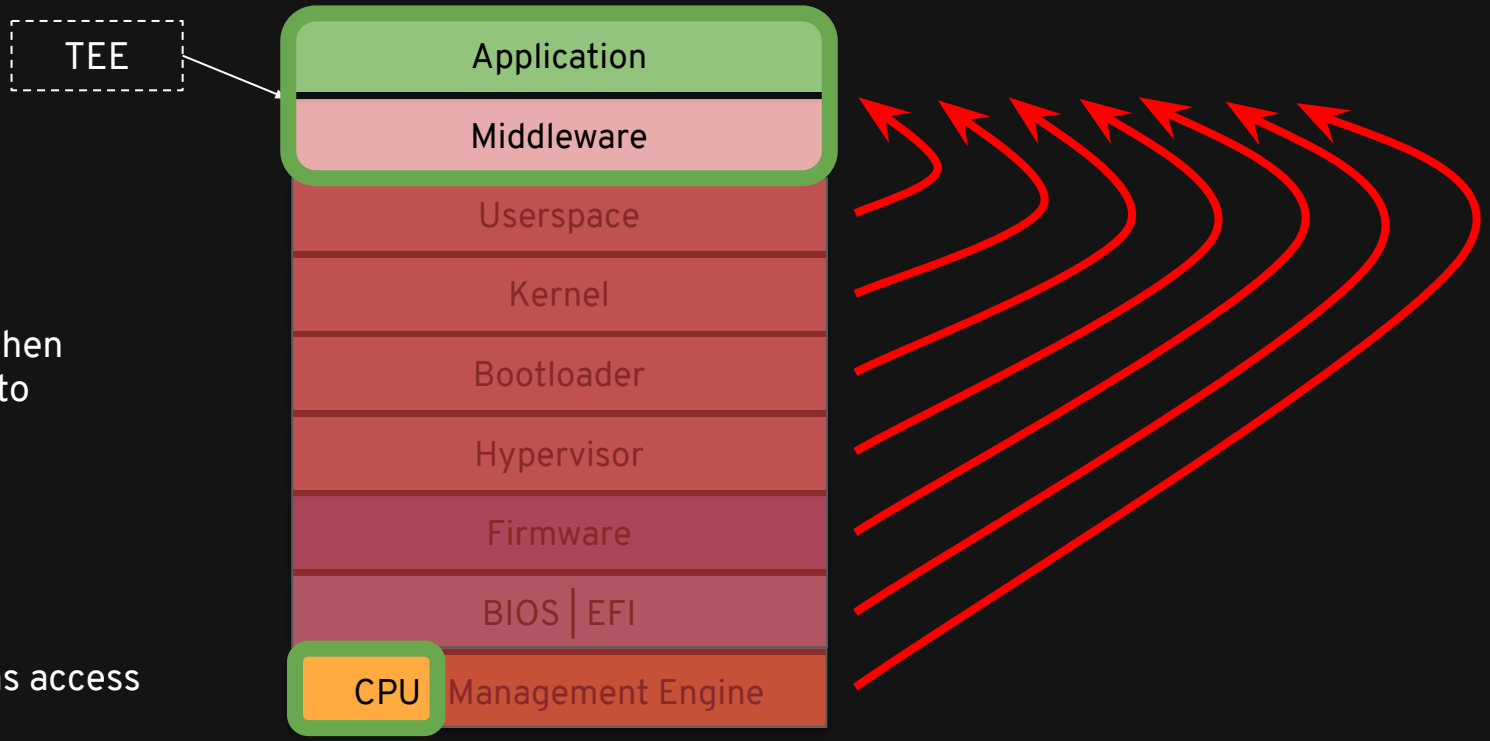
What's a TEE?



Only the CPU has access

Red Hat

What's a TEE?

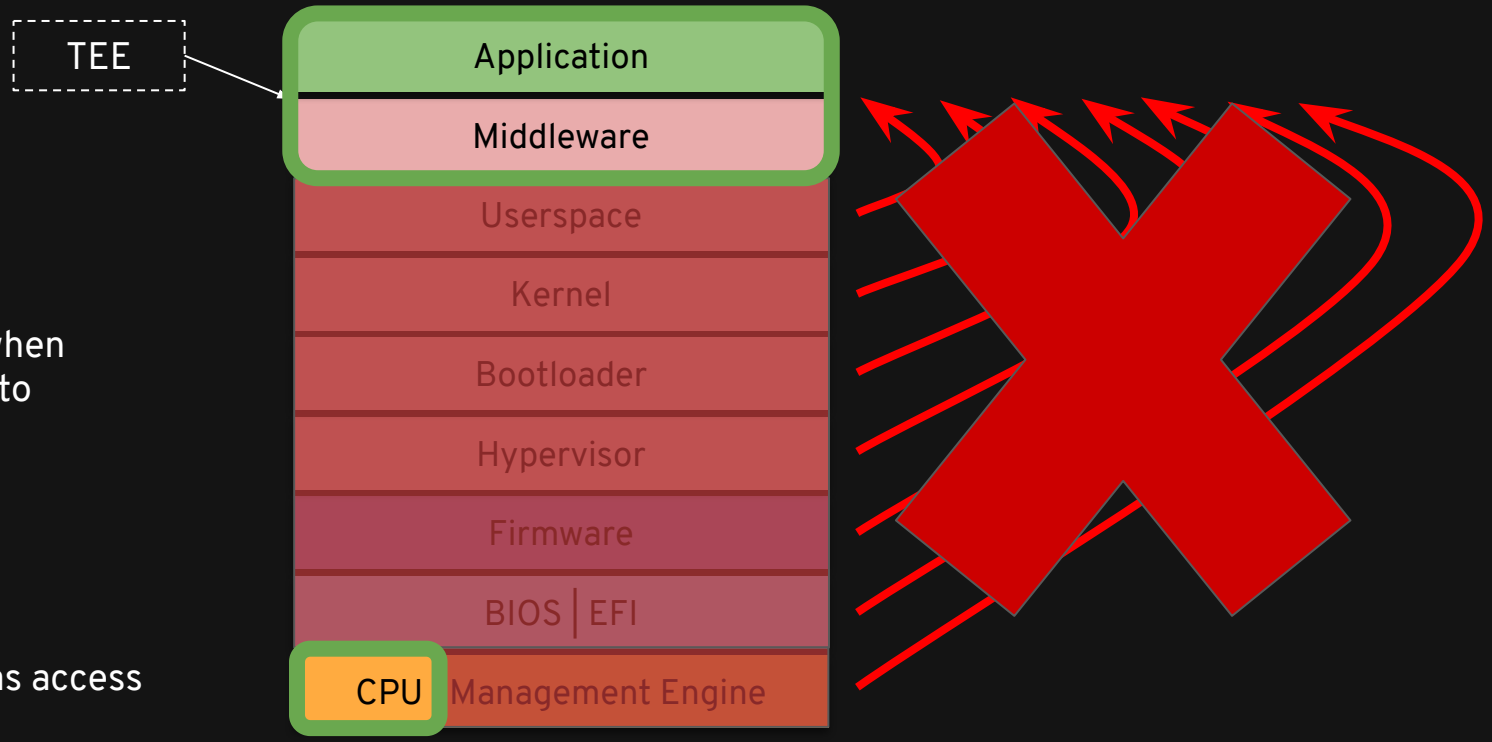


What happens when other layers try to access?

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

Red Hat

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

Red Hat

Trusted Execution Environments

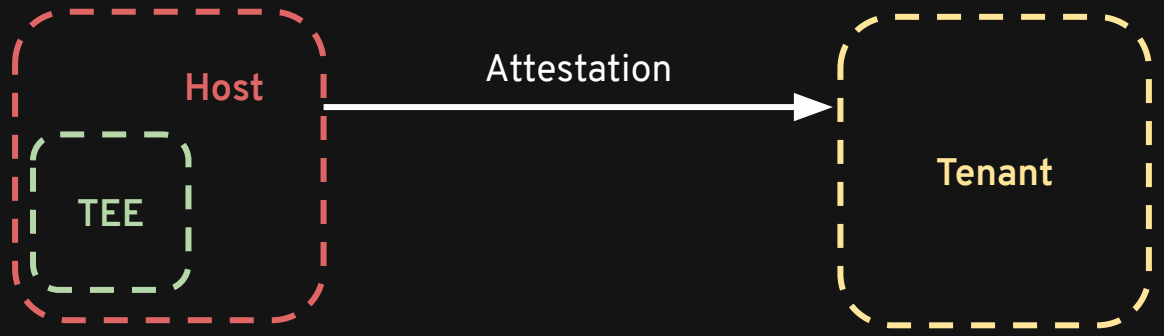
Q. “But how do I know that it’s a valid TEE?”

TEE provides:

- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Red Hat

Trusted Execution Summary



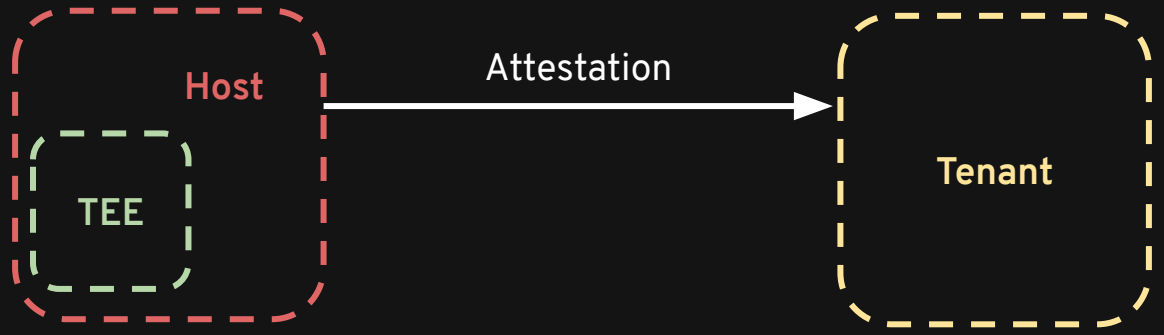
Q. “But how do I know that it’s a valid TEE?”
A. Attestation

TEE provides:

- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Red Hat

Trusted Execution Summary



Attestation includes:

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

TEE provides:

- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Red Hat

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 TDX

Not a TEE: TrustZone, TPM

Red Hat

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

Red Hat

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

TEEs sound great, but…

Red Hat

TEEs sound great, but…

Red Hat

TEEs sound great, but…

1. Different platforms → separate development

Red Hat

TEEs sound great, but…

1. Different platforms → separate development
2. Different SDKs → restricted language availability

TEEs sound great, but…

1. Different platforms → separate development
2. Different SDKs → restricted language availability
3. Different attestation models → complex, dynamic trust decisions

TEEs sound great, but…

1. Different platforms → separate development
2. Different SDKs → restricted language availability
3. Different attestation models → complex, dynamic trust decisions
4. Different vendors → vulnerability management woes

TEEs sound great, but…

1. Different platforms → separate development
2. Different SDKs → restricted language availability
3. Different attestation models → complex, dynamic trust decisions
4. Different vendors → vulnerability management woes

… I just want to deploy workloads!

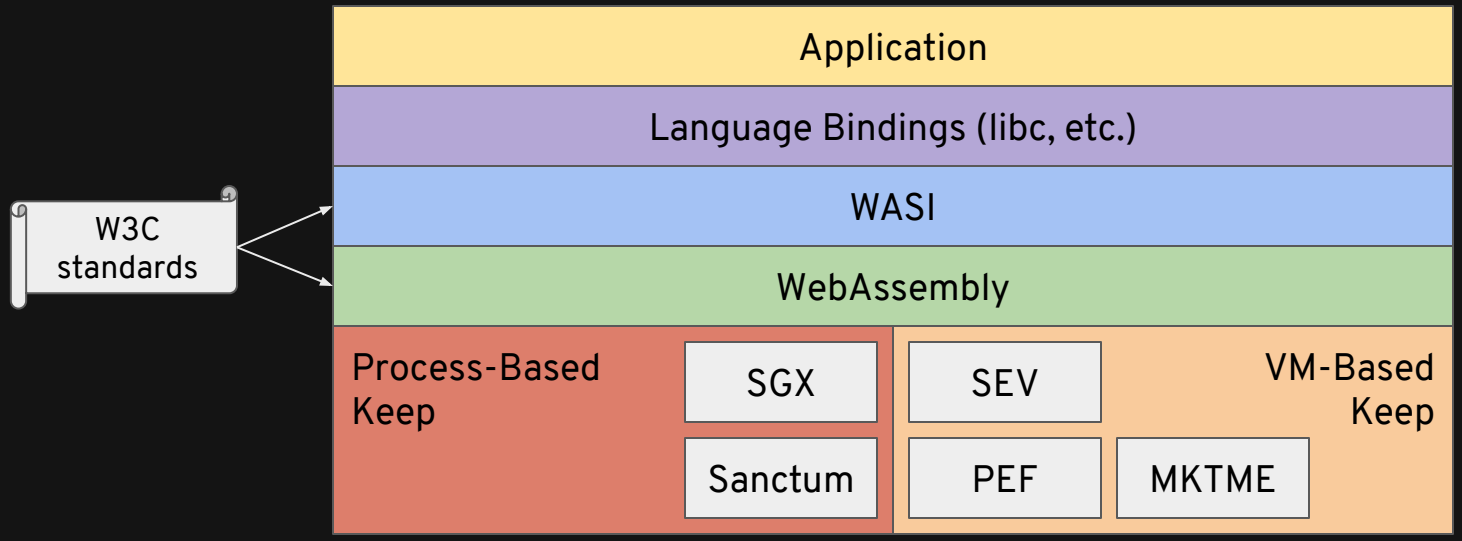
Red Hat

On which technology do I build my application?

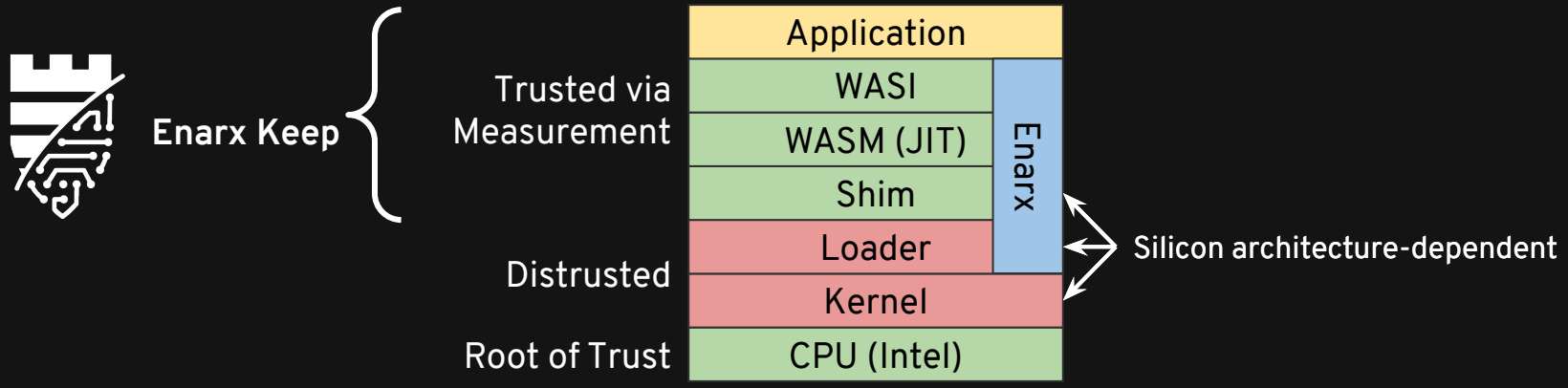
Red Hat

Introducing Enarx

Enarx Runtime Architecture

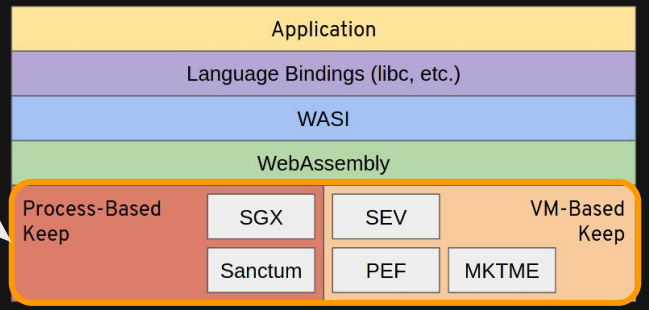


Layers - process-based Keep



Keep - process or VM-based

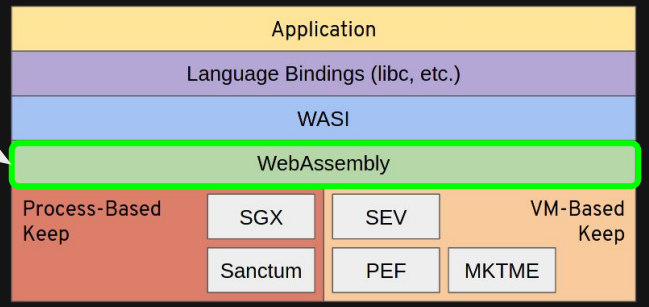
- Core Keep
- Platform-specific
  - Hardware (CPU): silicon vendor
  - Firmware: silicon vendor
  - Software: Enarx



Architecture varies between VM/Process-based platforms

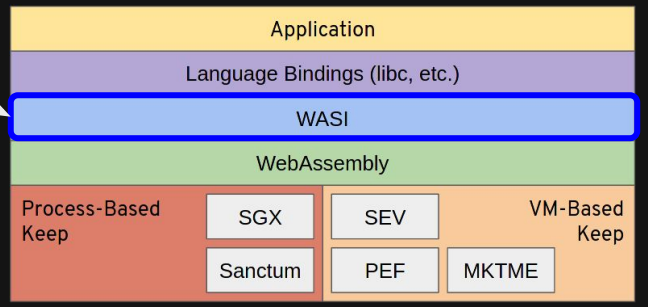
WebAssembly (WASM)

- 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)

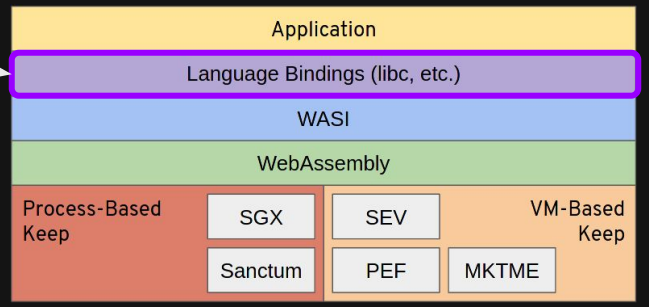
- W3C Standards Track
- Heavily inspired by a subset of POSIX
- Primary goals:
  - Portability
  - Security
- libc implementation on top
- Capability-based security:
  - No absolute resources
  - Think: openat() but not open()



Language Bindings (libc, etc.)

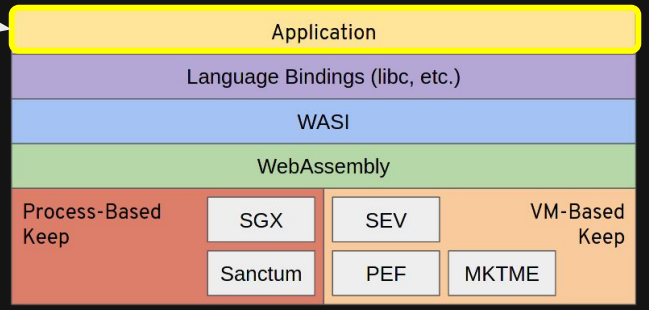
Compilation targets and includes, e.g.

- Rust: `--target wasm32-wasi`

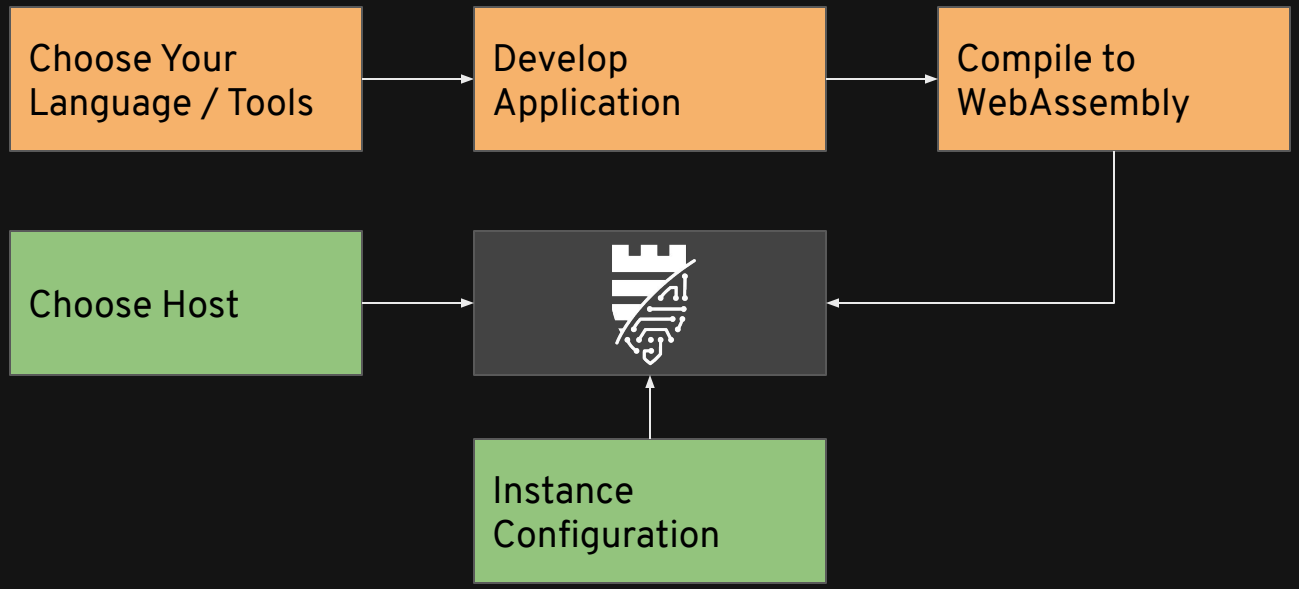


Application

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



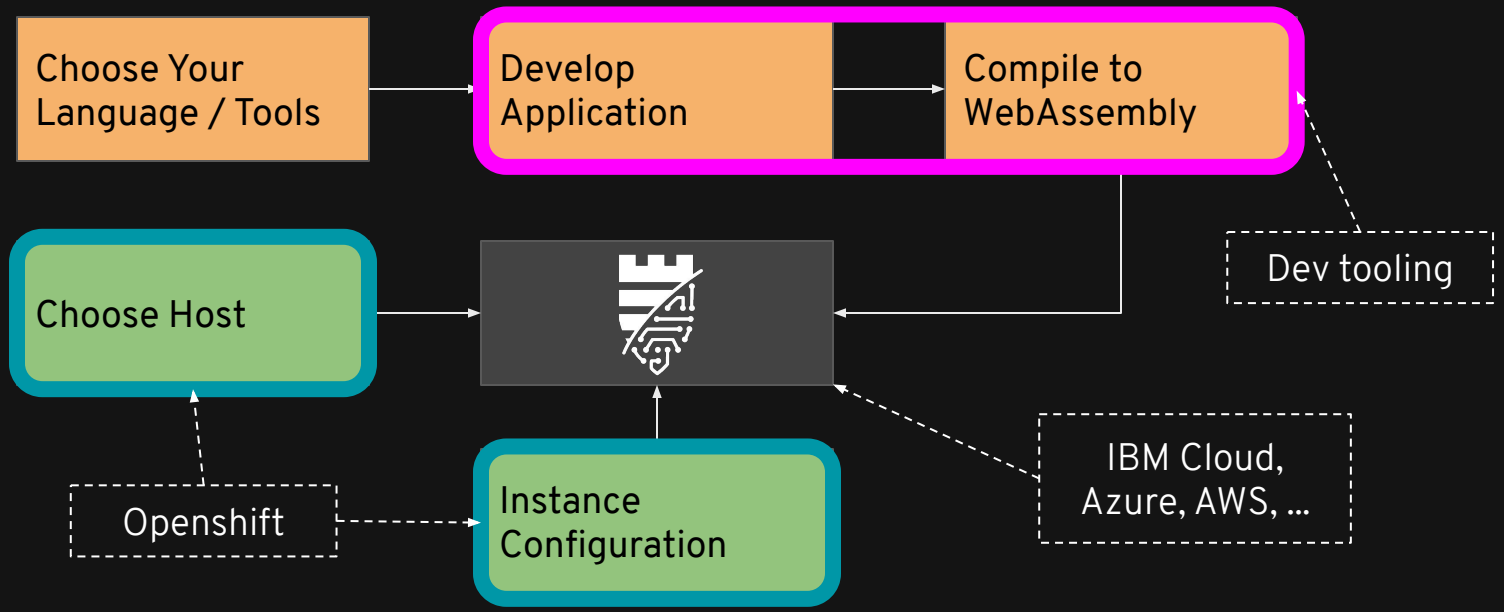
Enarx is a ~~Development~~ Deployment Framework



Red Hat

Enarx is a ~~Development~~ Deployment Framework

(Example components)

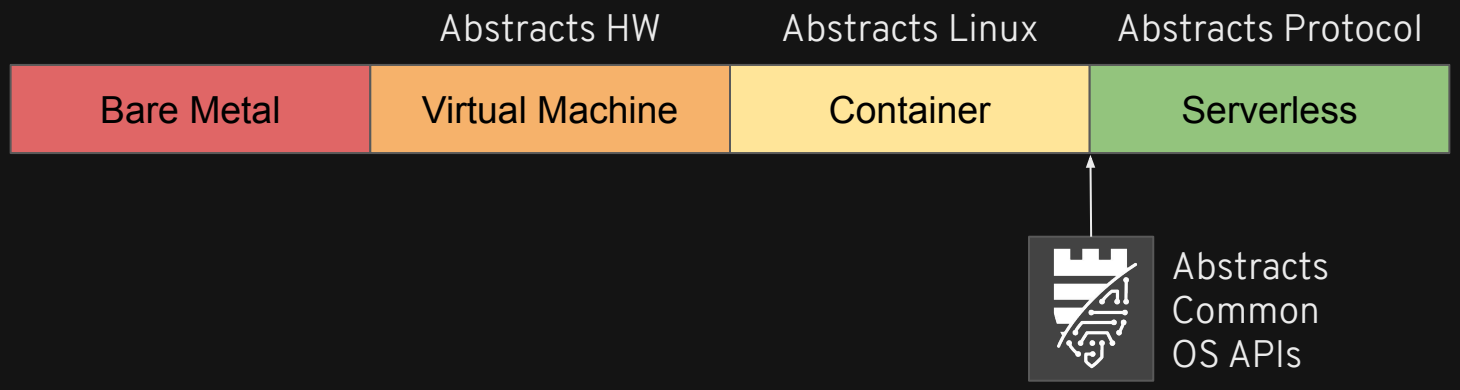


Bare Metal	Virtual Machine	Container	Serverless

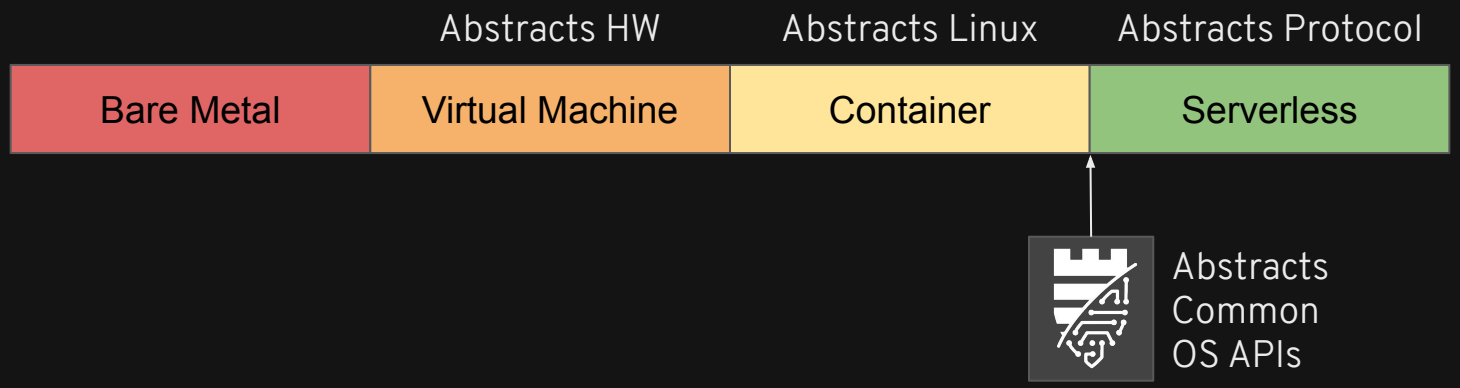
Red Hat

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

Red Hat


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

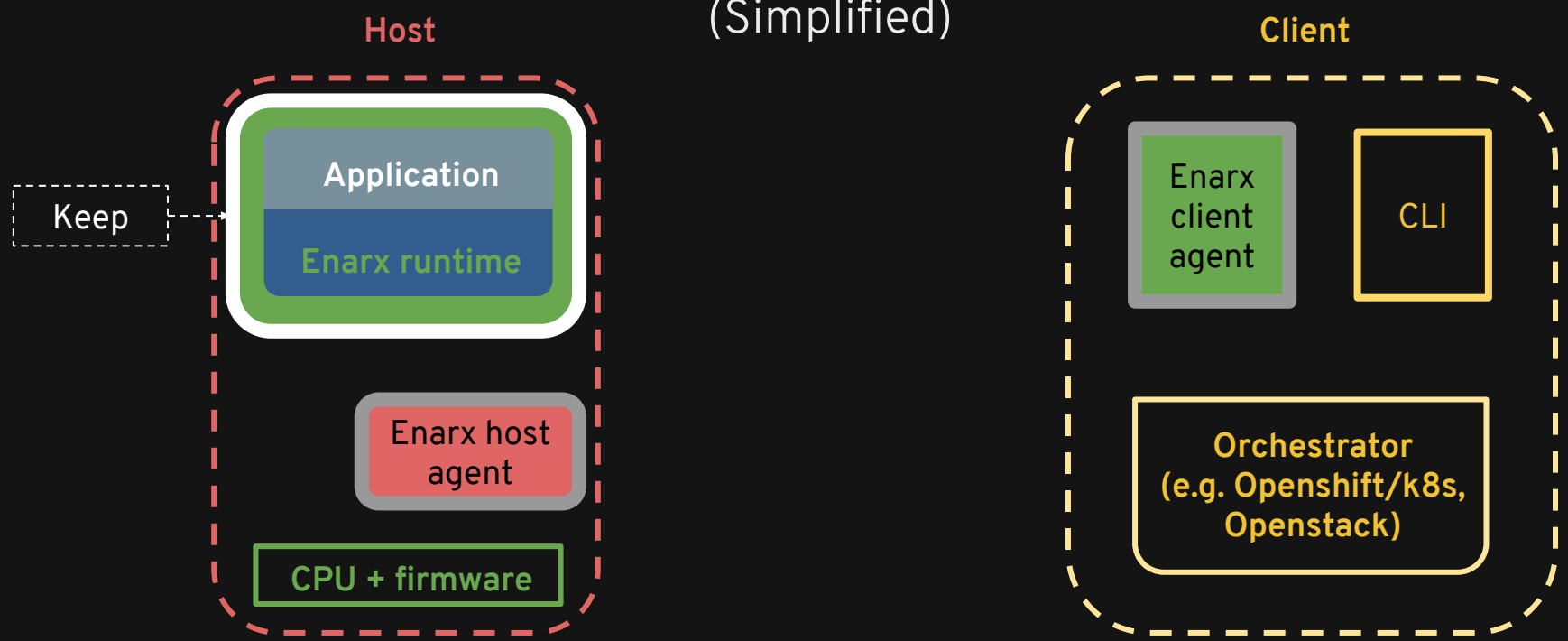
Red Hat



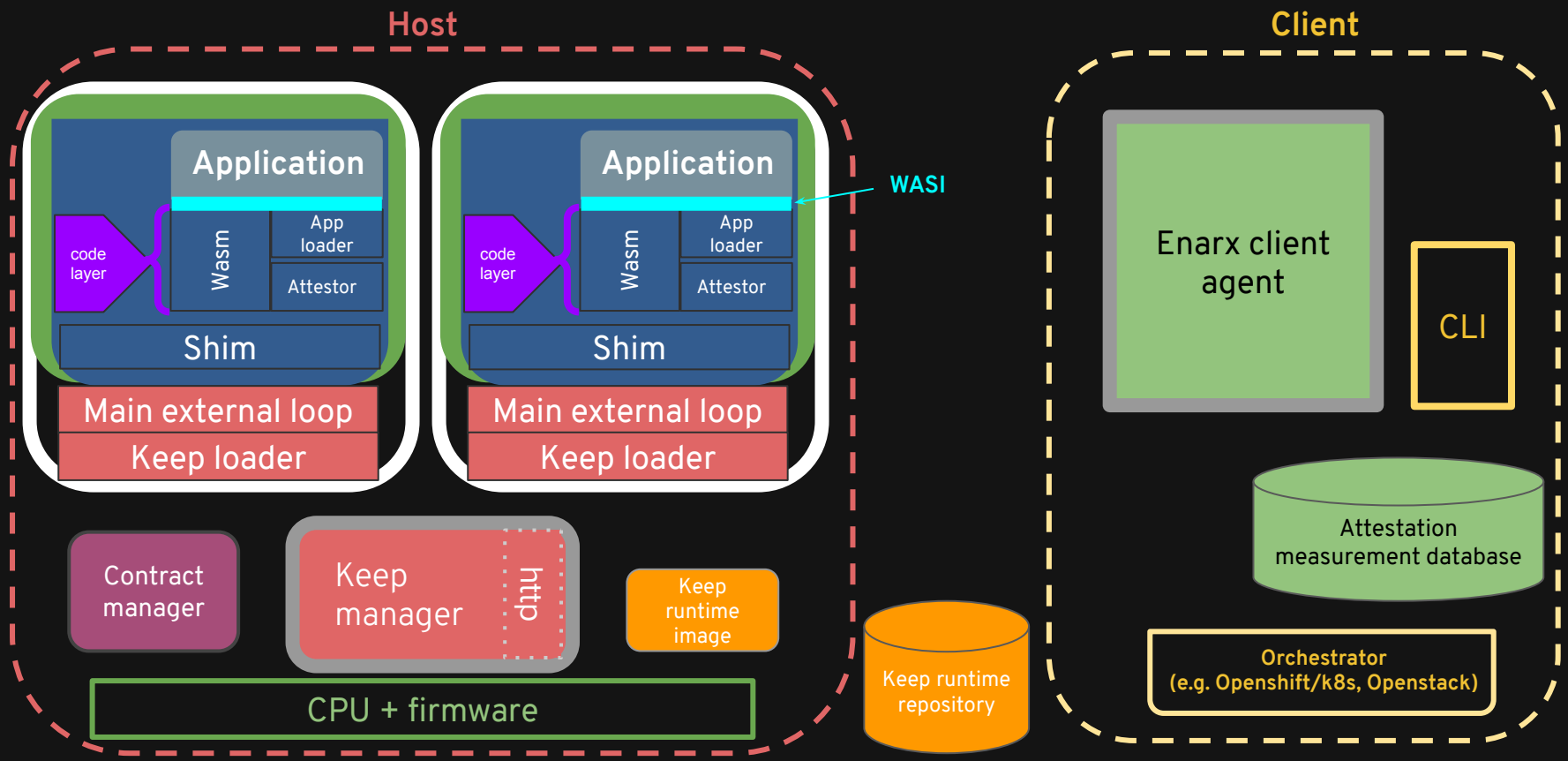
Just enough legacy support to enable trivial application portability.
Homogeneity to enable radical deployment-time portability.
No interfaces which accidentally leak data to the host.
Bridges process-based and VM-based TEE models.
No operating system to manage.

Architectural View

Enarx architectural components & integrations (Simplified)



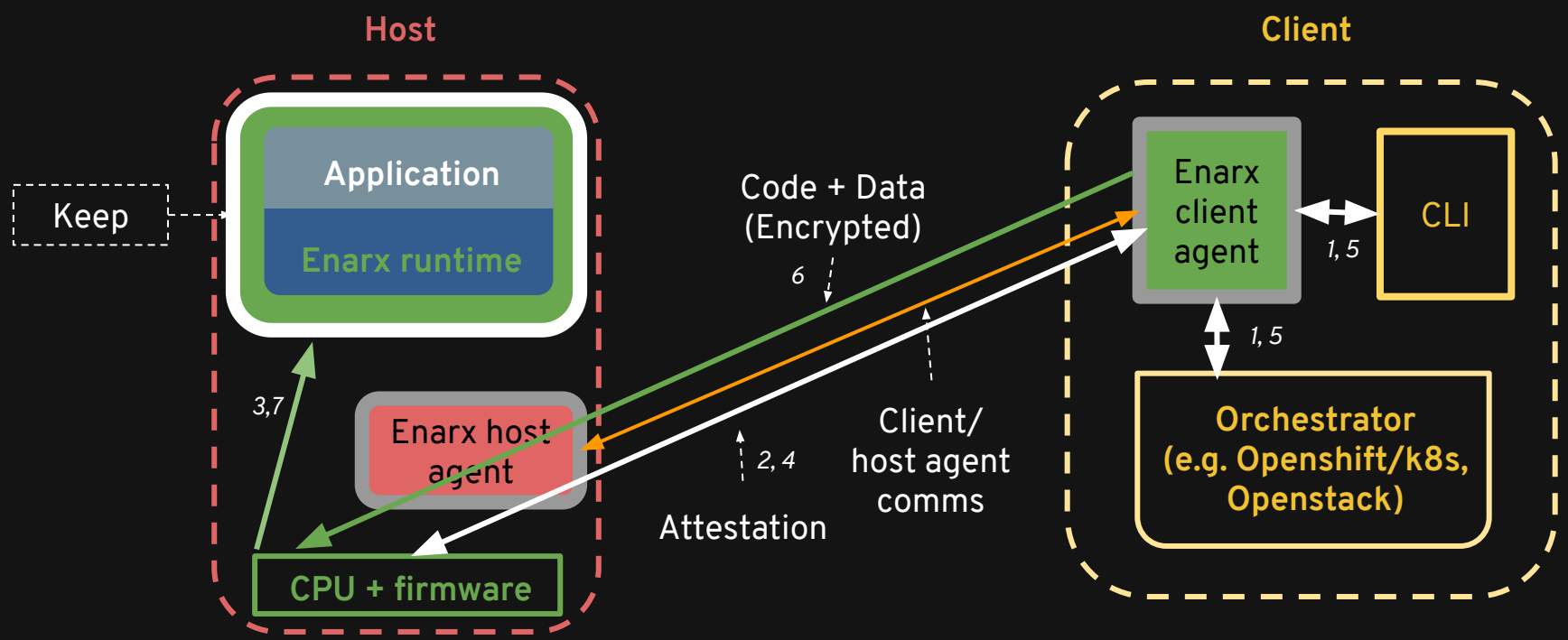
Enarx architectural components



Red Hat

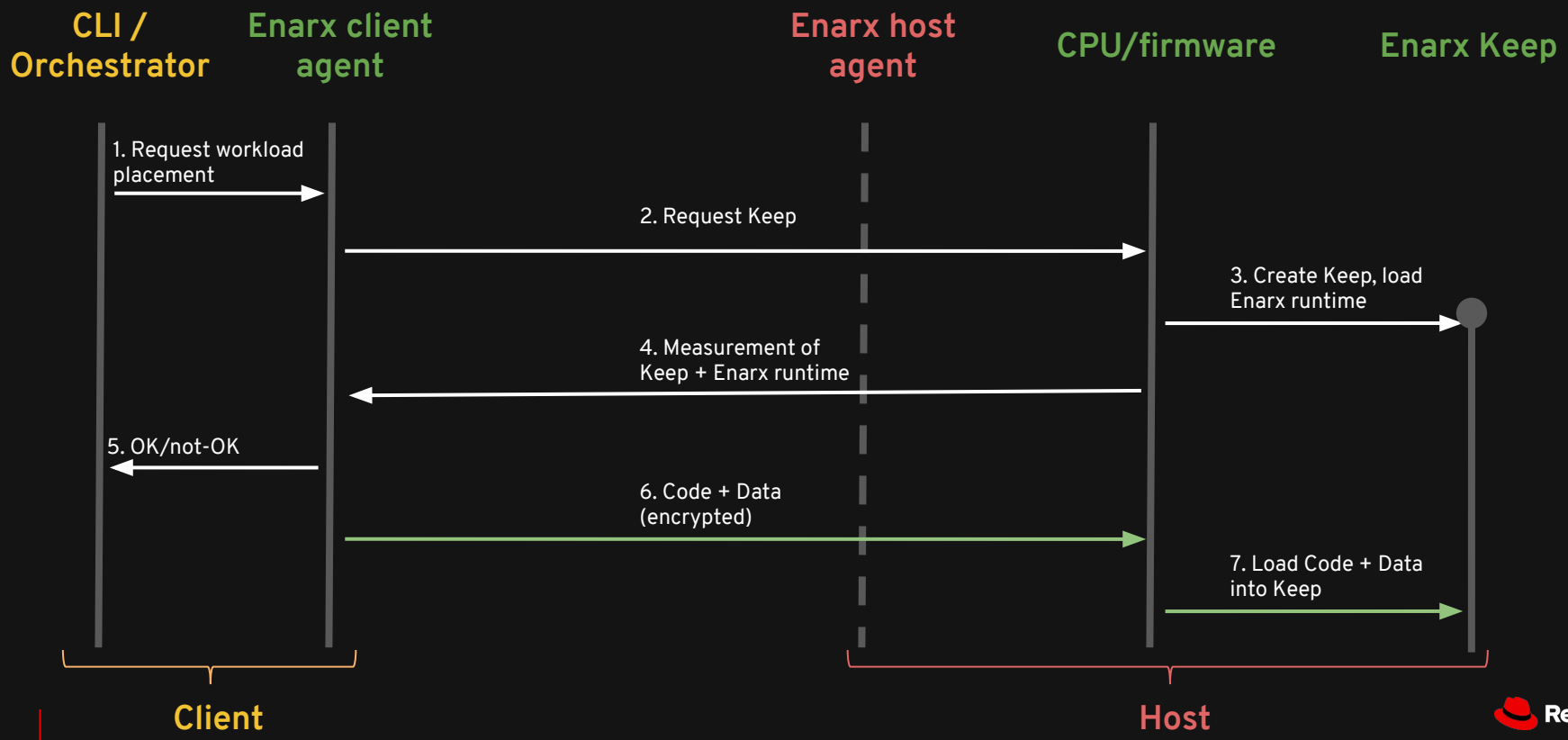
Process flow

Enarx architectural components



Red Hat

Enarx attestation process diagram



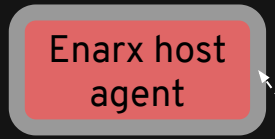
Red Hat

Component level trust

Enarx architectural components

Enarx Keep - trusted
Measured and attested
WebAssembly+WASI runtime
Inside a TEE instance

Enarx client agent - trusted
Works with orchestration/CLI
Manages attestation
Applies policy
Encrypts and transports workload

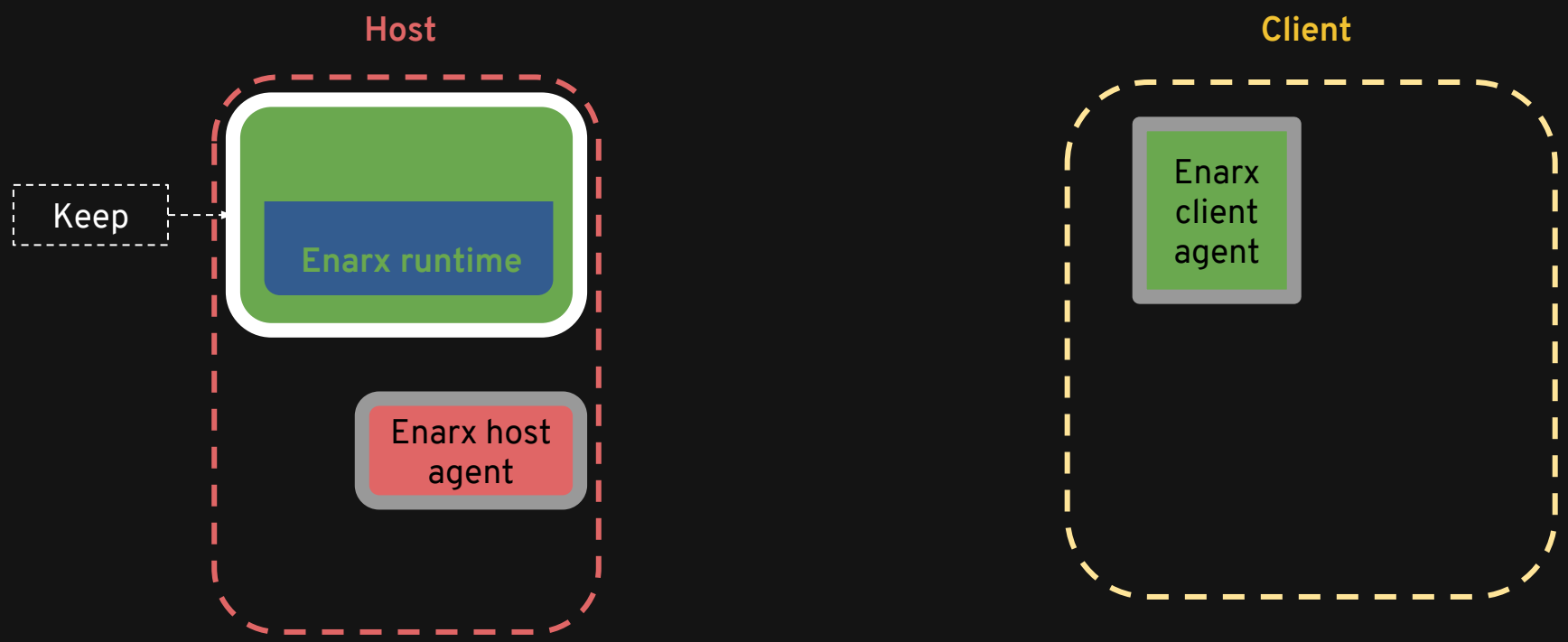


Enarx host agent - untrusted
Acts as proxy between Enarx client agent and:

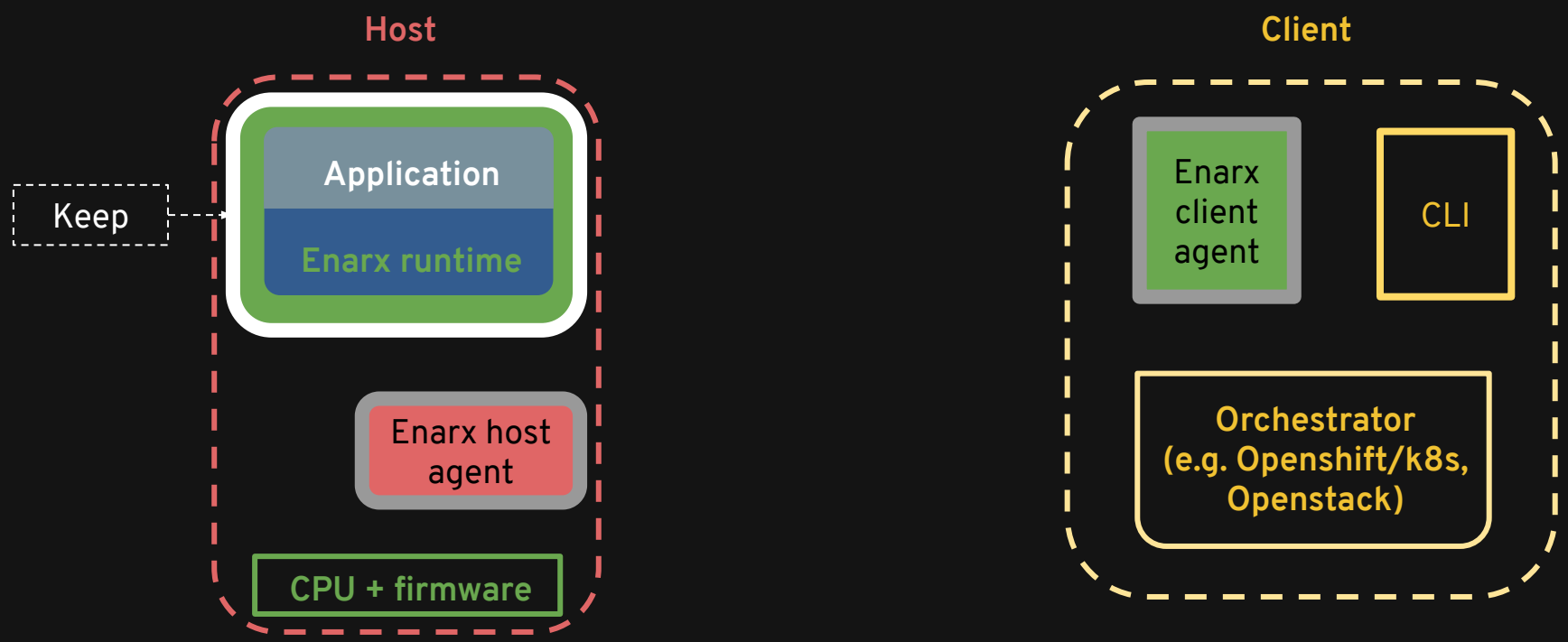
- CPU/firmware
- Enarx Keep

Red Hat

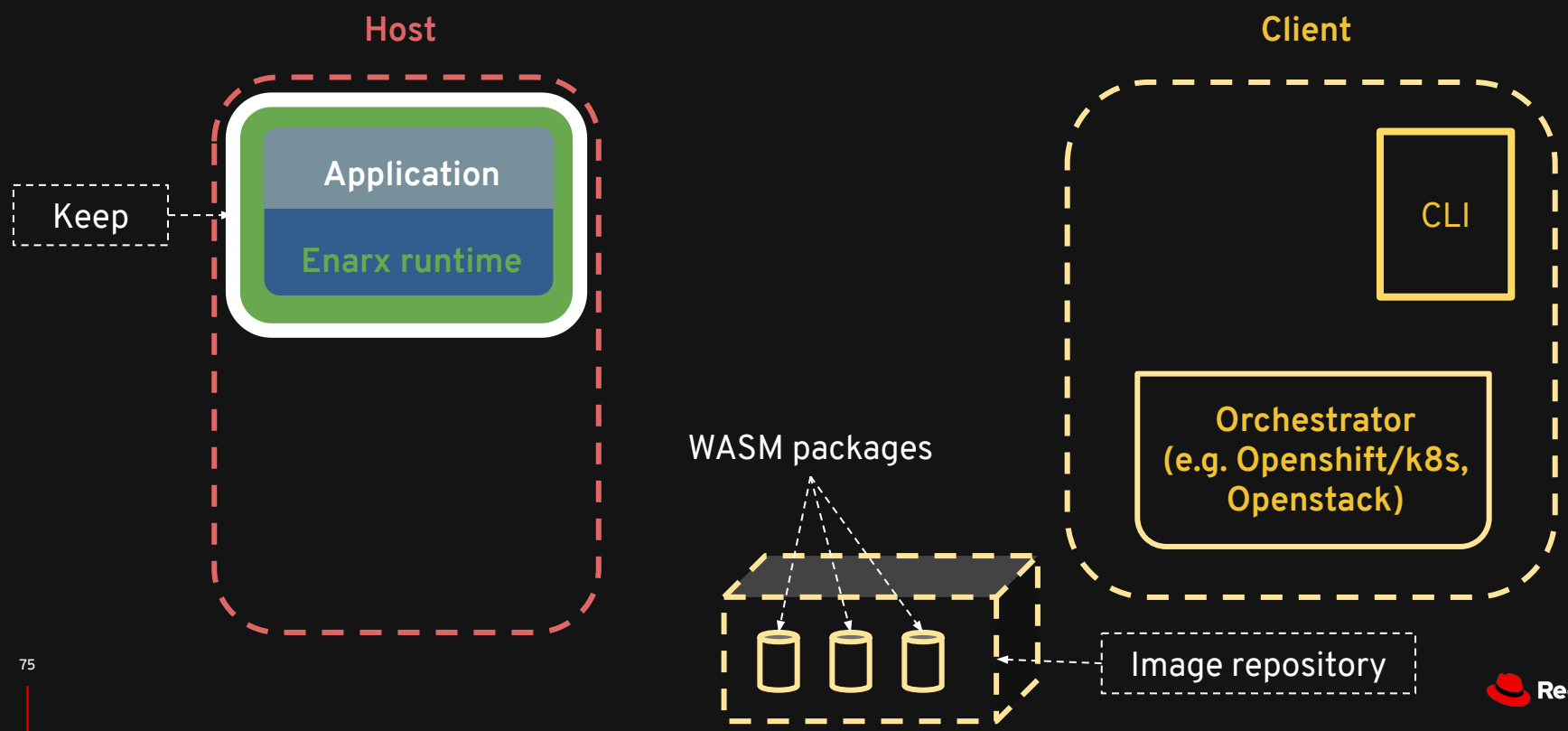
Enarx architectural components



Enarx architectural components

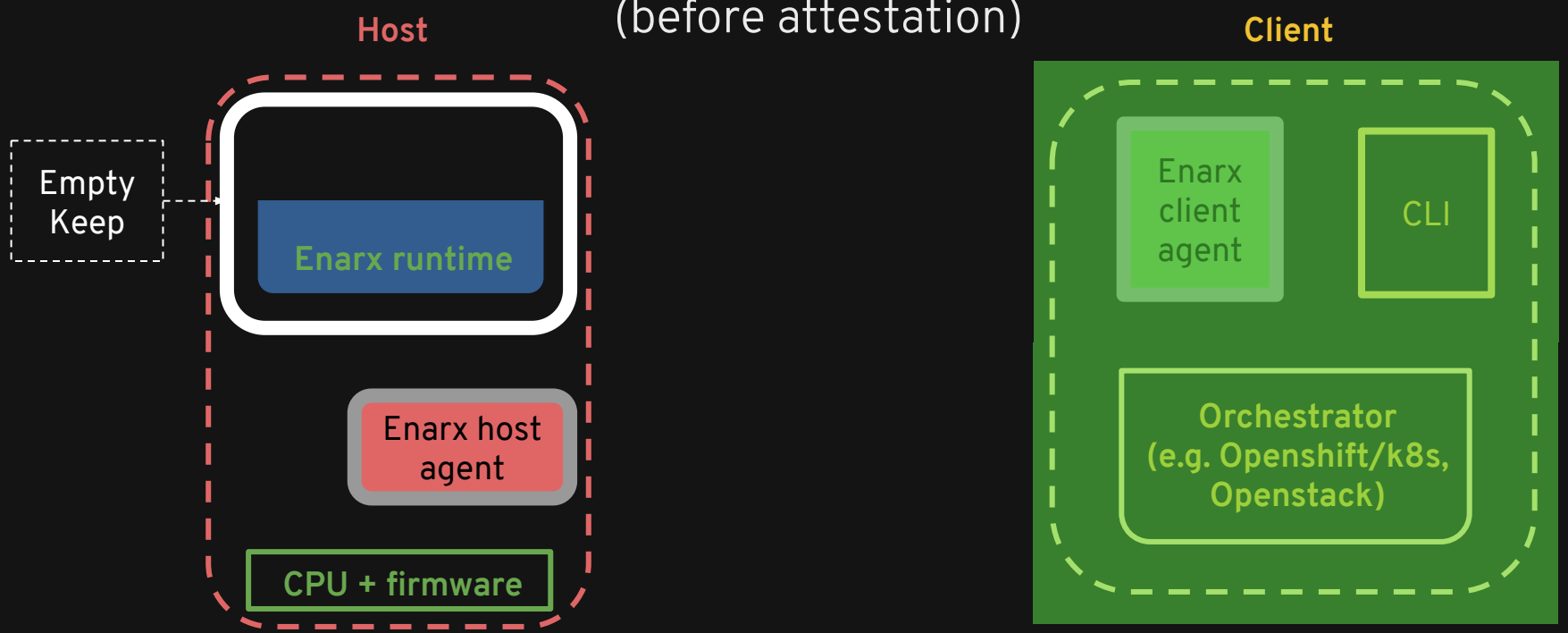


Basic deployment architecture

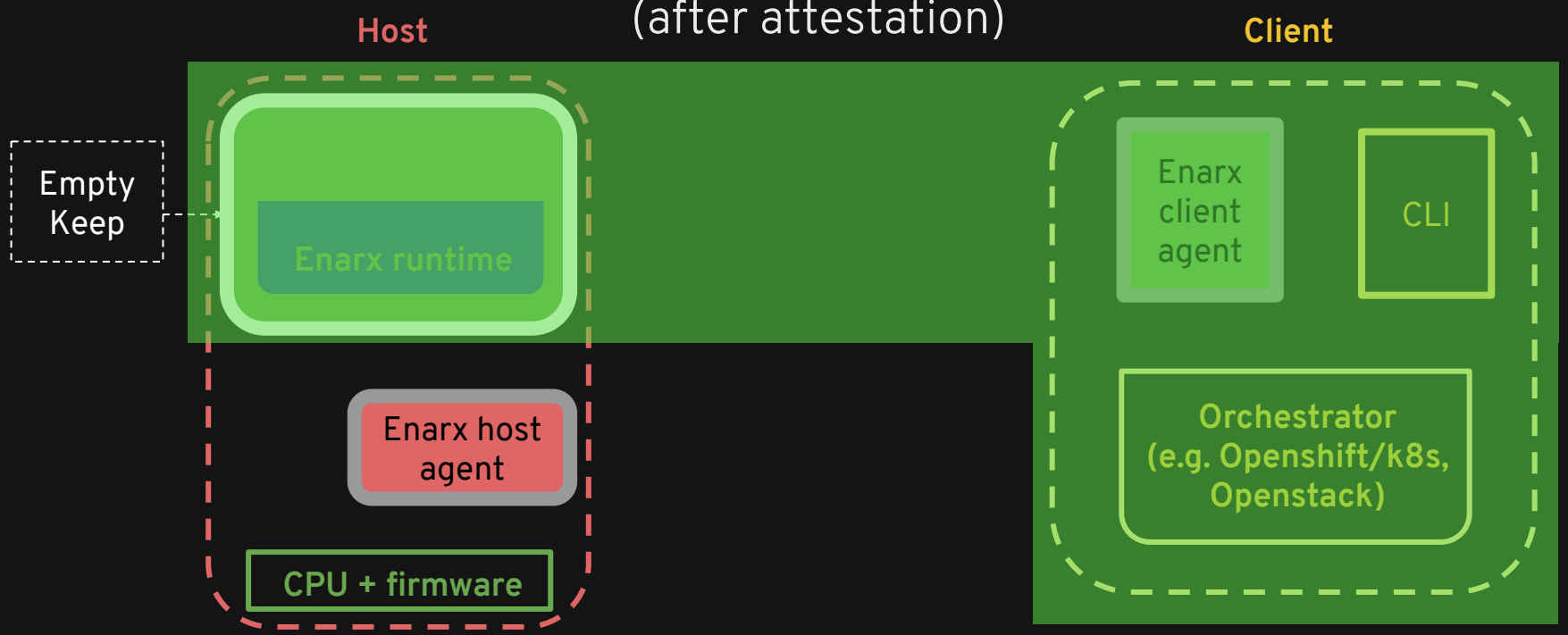


Red Hat

Standard Enarx trust domain (before attestation)

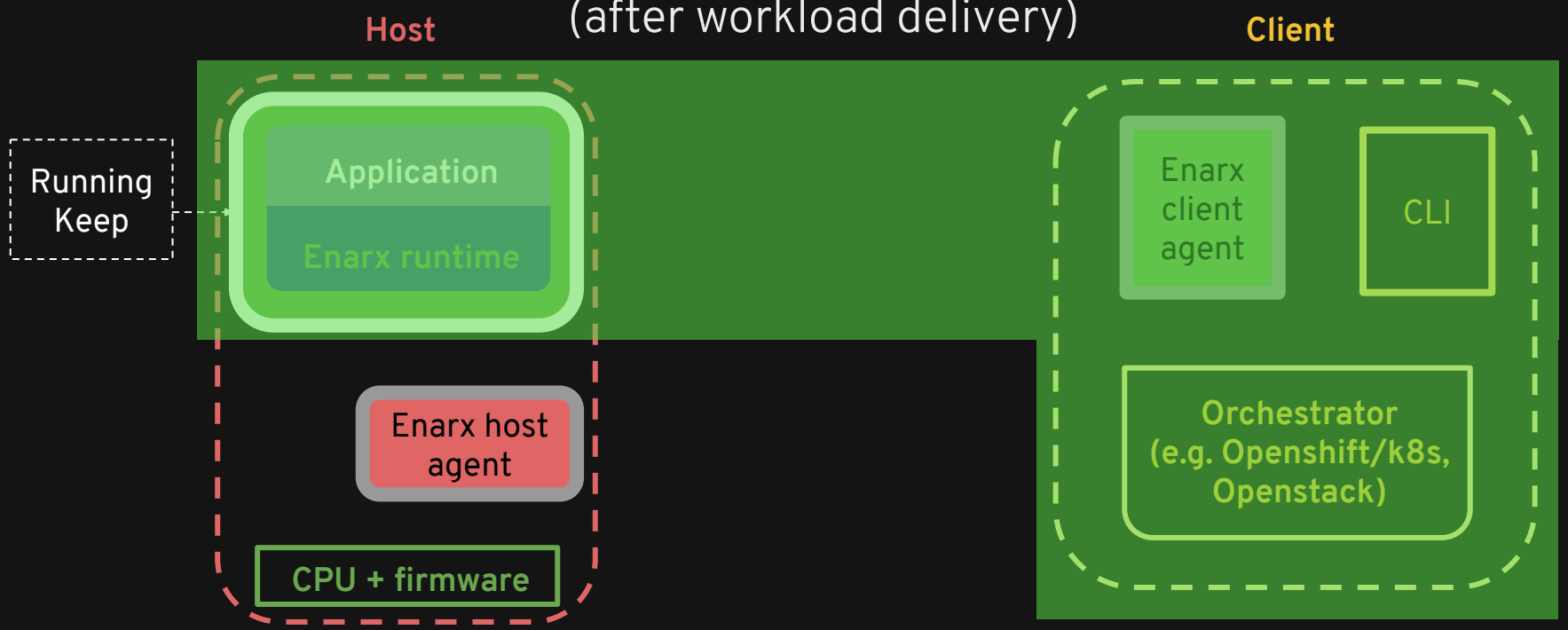


Standard Enarx trust domain (after attestation)



Red Hat

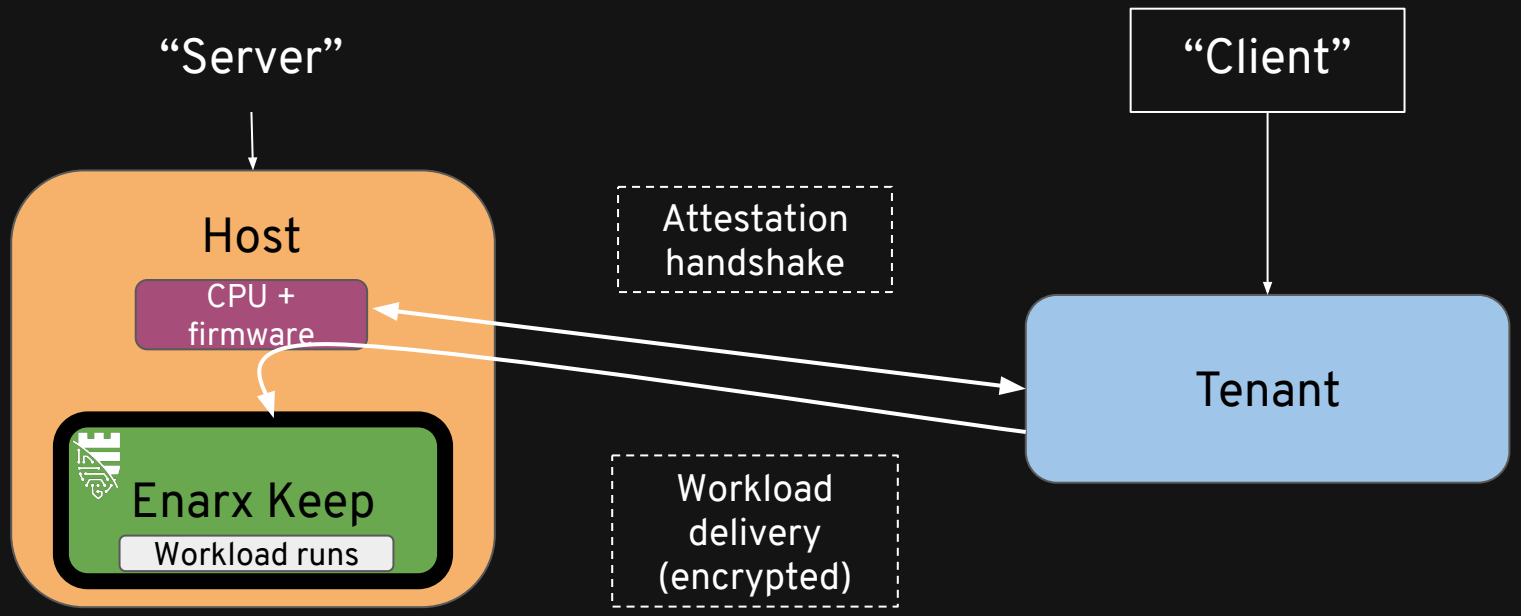
Standard Enarx trust domain (after workload delivery)



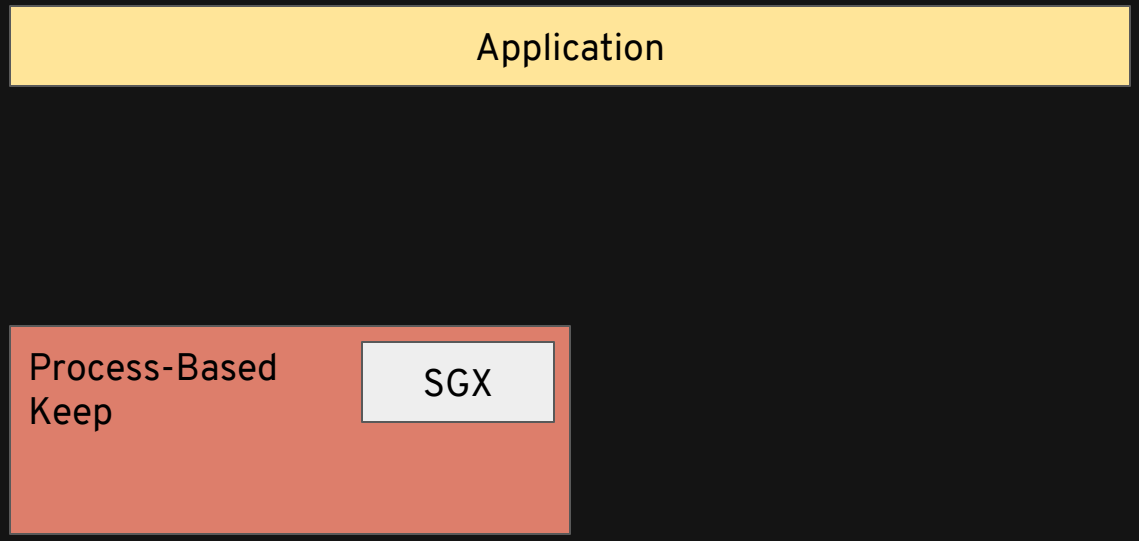
Demo Time!

Red Hat

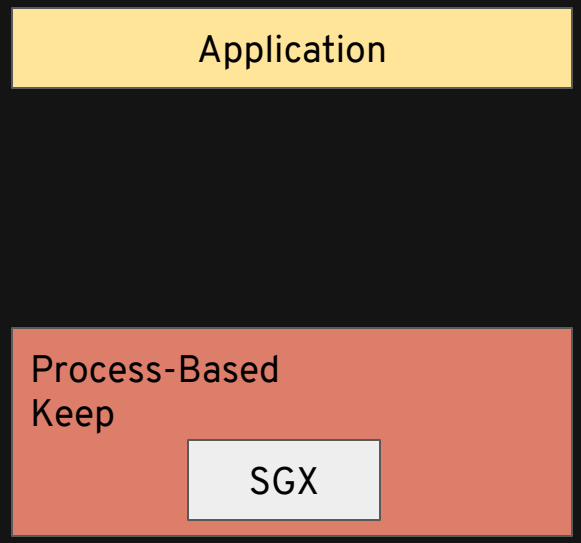
What’s the full picture?



Breaking things down with SGX

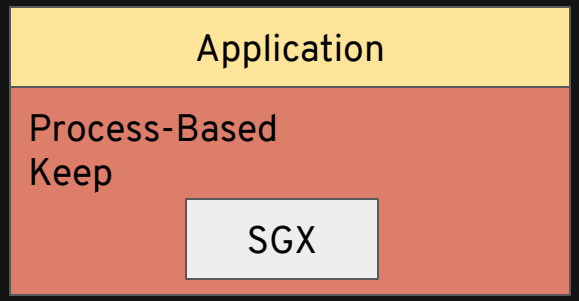


Breaking things down with SGX

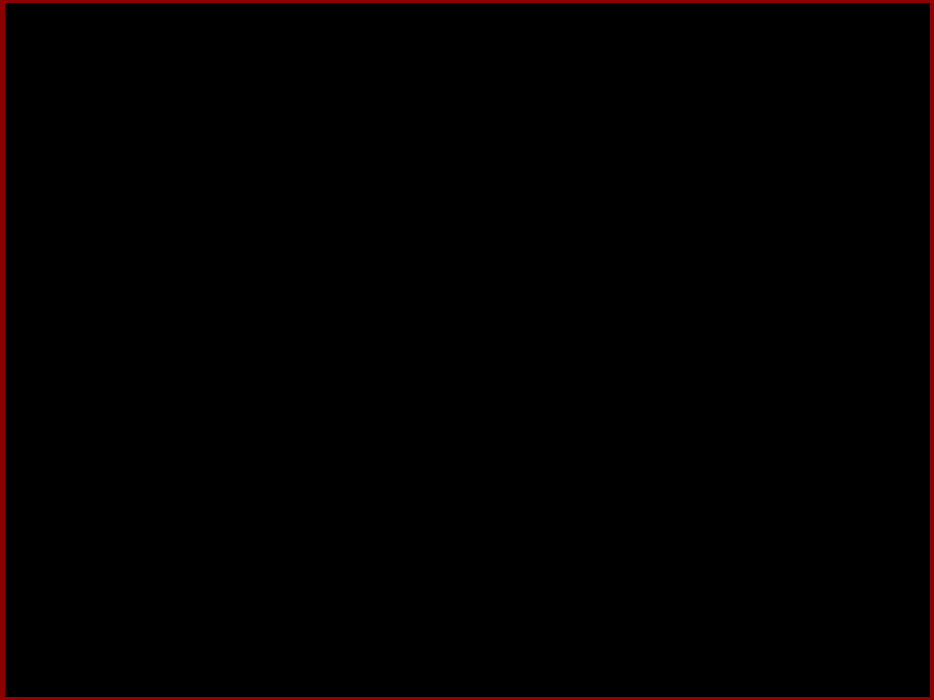


Red Hat

Breaking things down with SGX

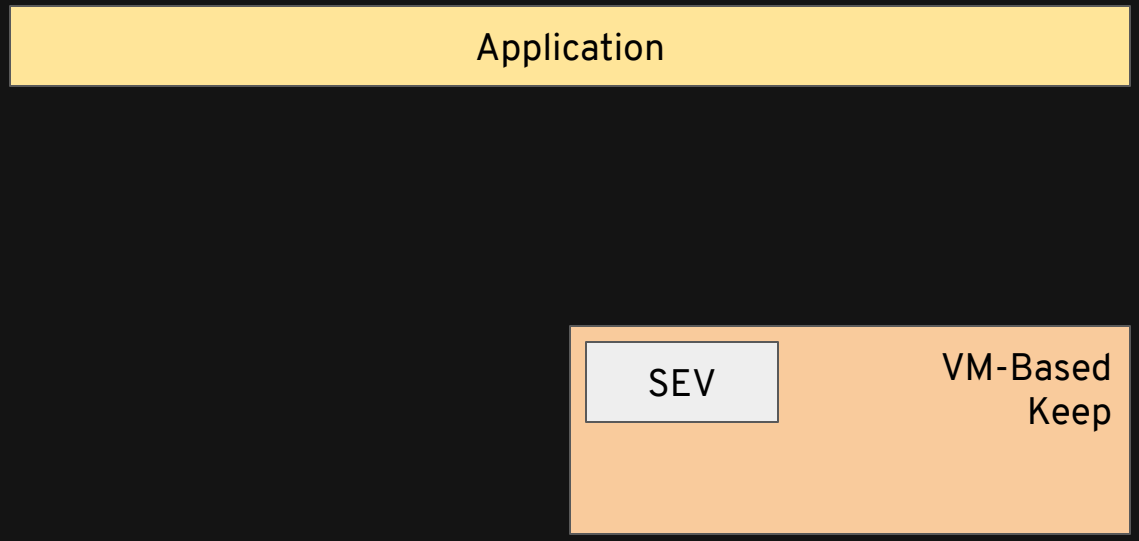


SGX demo

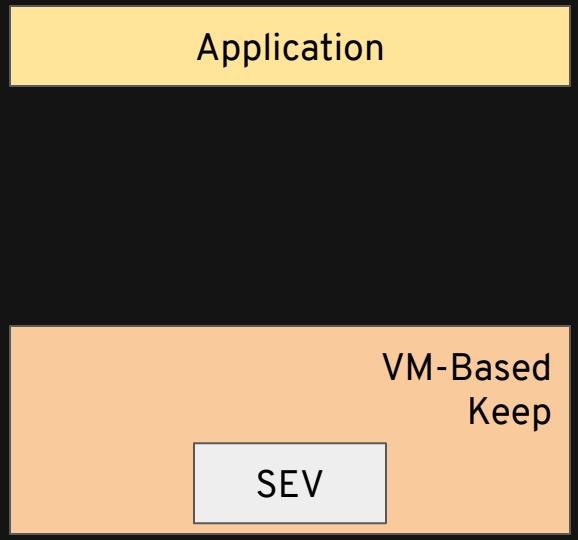


Red Hat

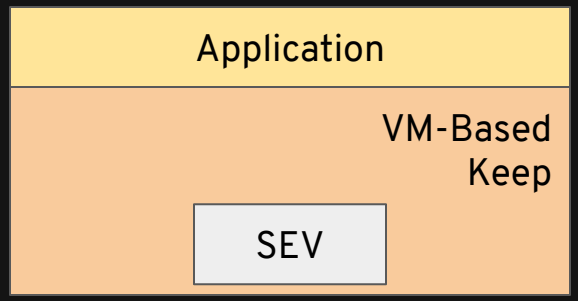
Breaking things down with SEV



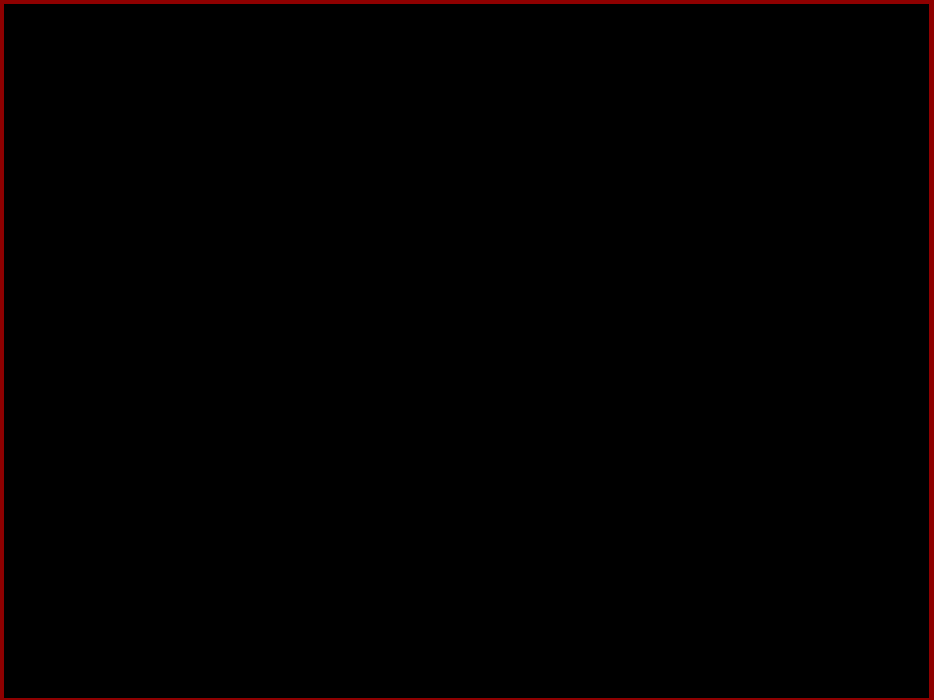
Breaking things down with SEV



Breaking things down with SEV

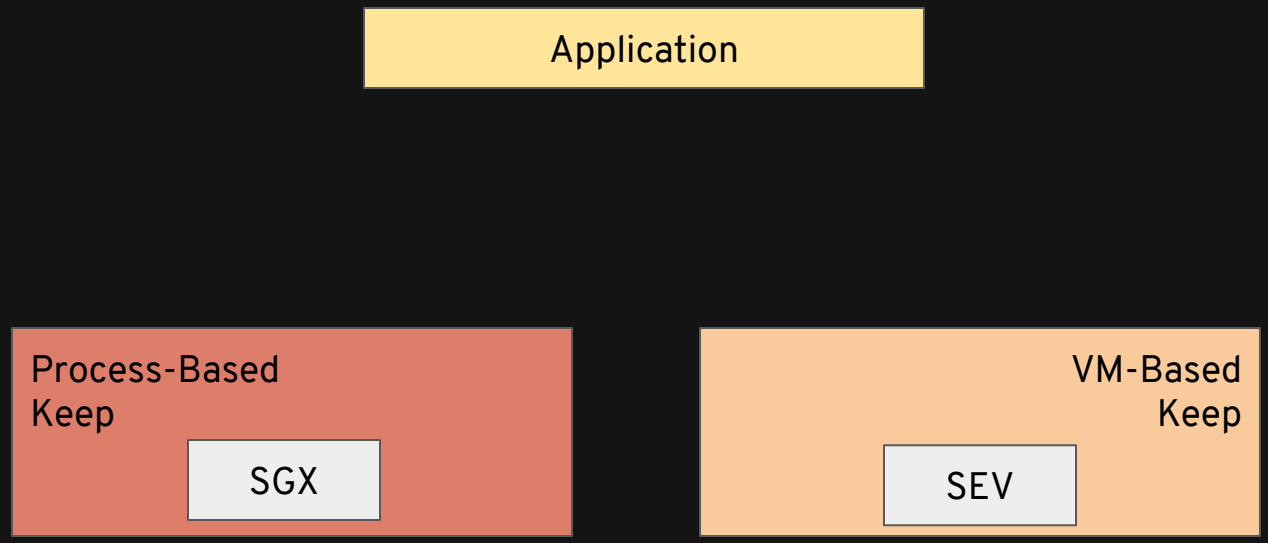


SEV demo

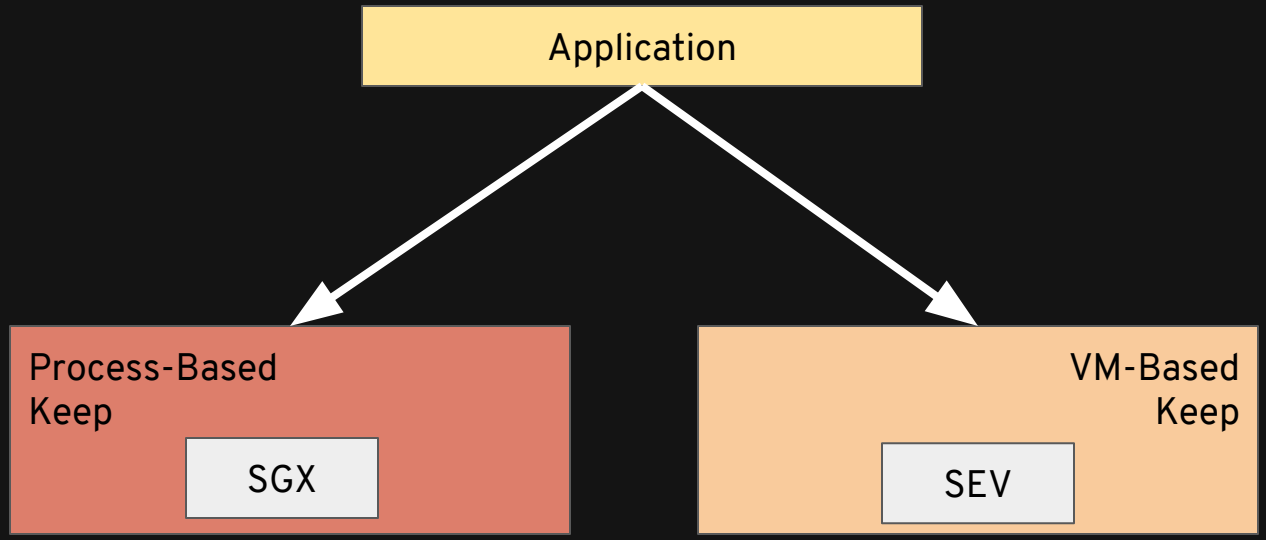


Red Hat

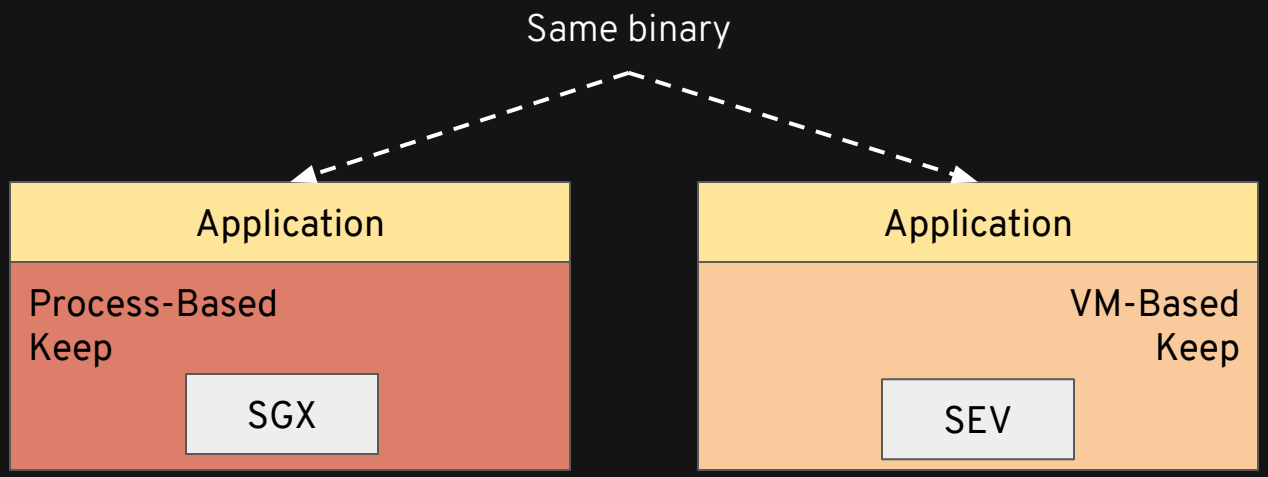
Where we'd like to be



Where we'd like to be

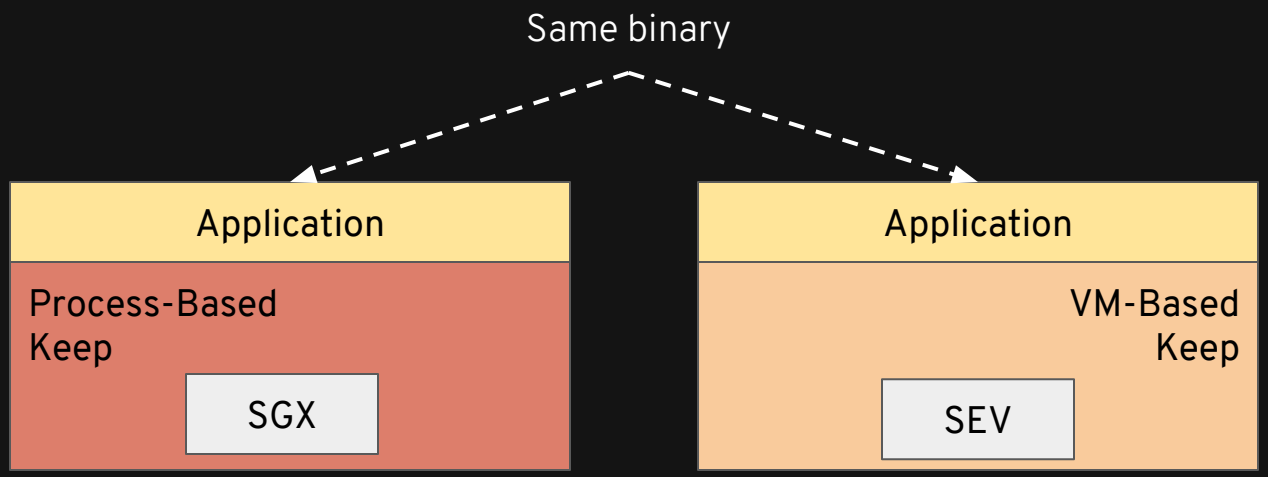


Where we'd like to be



Where ~~we would like to~~ be

Where we are



```
[nathaniel@localhost enarx]$ cat ~/test.c
#include <stdio.h>

int
main(int argc, char *argv[])
{
    printf("Good morning, that's a nice tnetennba!\n");
    return 0;
}
[nathaniel@localhost enarx]$ musl-gcc -fPIE -static-pie -o ~/test ~/test.c
[nathaniel@localhost enarx]$ file ~/test
/home/nathaniel/test: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), statically linked, BuildID[sha1]=f
ba649daba150448a9ea182aa87d16349f04e1de, with debug_info, not stripped
[nathaniel@localhost enarx]$ target/x86_64-unknown-linux-musl/debug/enarx-keep-sgx --shim target/x86_64-unknown-
linux-musl/debug/enarx-keep-sgx-shim --code ~/test
```

```
[nathaniel@localhost enarx]$ cat ~/test.c
#include <stdio.h>

int
main(int argc, char *argv[])
{
    printf("Good morning, that's a nice tnetennba!\n");
    return 0;
}
[nathaniel@localhost enarx]$ musl-gcc -fPIE -static-pie -o ~/test ~/test.c
[nathaniel@localhost enarx]$ file ~/test
/home/nathaniel/test: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), statically linked, BuildID[sha1]=f
ba649daba150448a9ea182aa87d16349f04e1de, with debug_info, not stripped
[nathaniel@localhost enarx]$ target/x86_64-unknown-linux-musl/debug/enarx-keep-sev --shim target/x86_64-unknown-
linux-musl/debug/enarx-keep-sev-shim --code ~/test
```

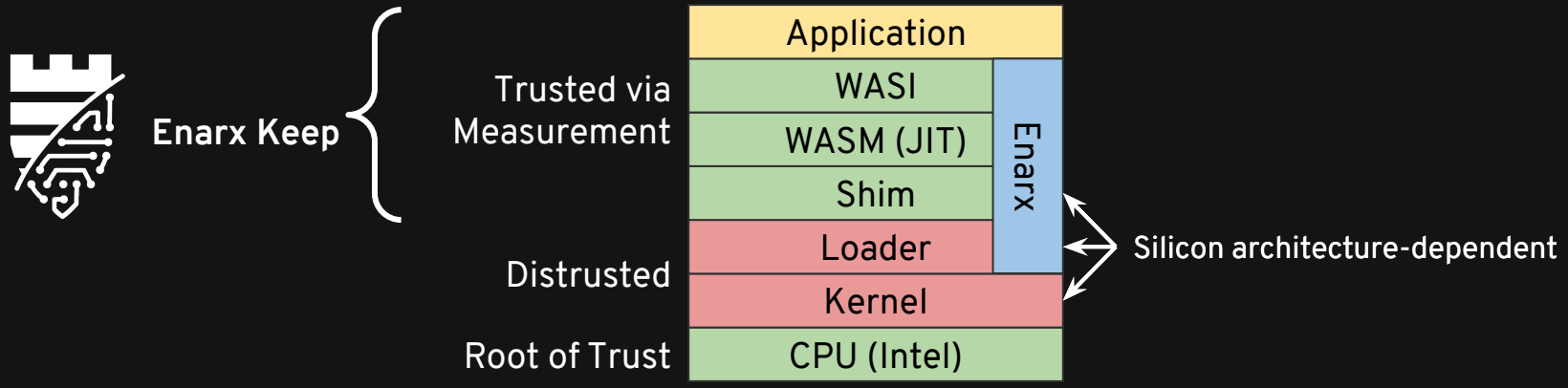
```
[nathaniel@localhost enarx]$ cat ~/test.c
#include <stdio.h>

int
main(int argc, char *argv[])
{
    printf("Good morning, that's a nice tnetennba!\n");
    return 0;
}
[nathaniel@localhost enarx]$ musl-gcc -fPIE -static-pie -o ~/test ~/test.c
[nathaniel@localhost enarx]$ file ~/test
/home/nathaniel/test: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), statically linked, BuildID[sha1]=f
ba649daba150448a9ea182aa87d16349f04e1de, with debug_info, not stripped
[nathaniel@localhost enarx]$ target/x86_64-unknown-linux-musl/debug/enarx-keep-sgx --shim target/x86_64-unknown-
linux-musl/debug/enarx-keep-sgx-shim --code ~/test
```

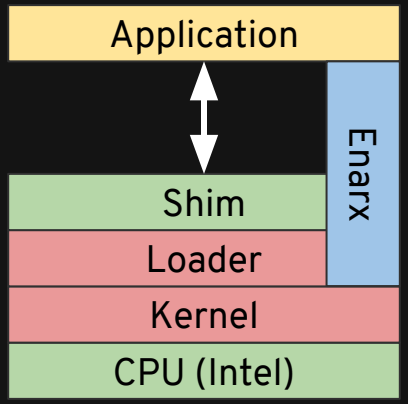
```
[nathaniel@localhost enarx]$ cat ~/test.c
#include <stdio.h>

int
main(int argc, char *argv[])
{
    printf("Good morning, that's a nice tnetennba!\n");
    return 0;
}
[nathaniel@localhost enarx]$ musl-gcc -fPIE -static-pie -o ~/test ~/test.c
[nathaniel@localhost enarx]$ file ~/test
/home/nathaniel/test: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), statically linked, BuildID[sha1]=f
ba649daba150448a9ea182aa87d16349f04e1de, with debug_info, not stripped
[nathaniel@localhost enarx]$ target/x86_64-unknown-linux-musl/debug/enarx-keep-sev --shim target/x86_64-unknown-
linux-musl/debug/enarx-keep-sev-shim --code ~/test
```

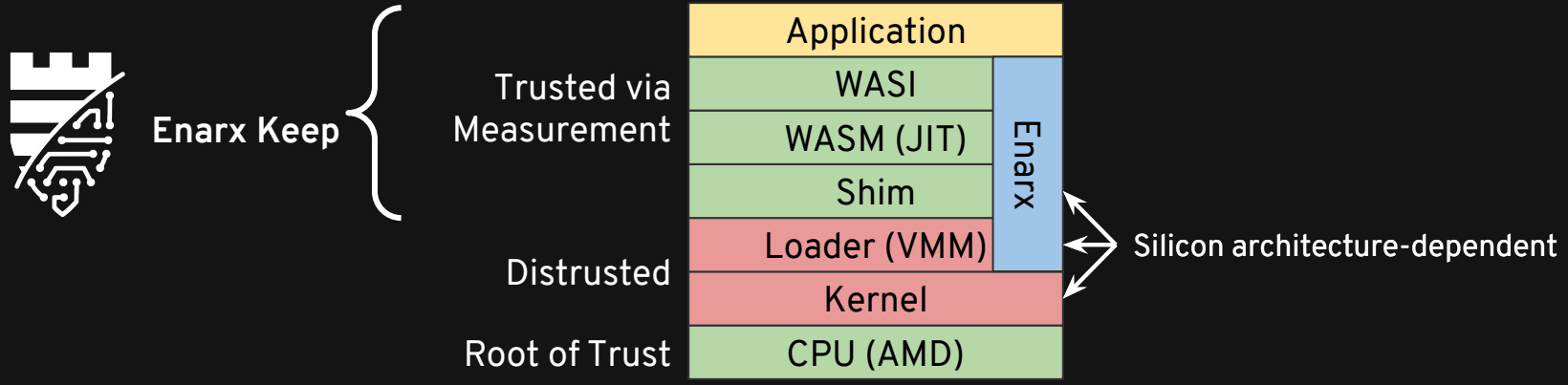
Layers - process-based Keep



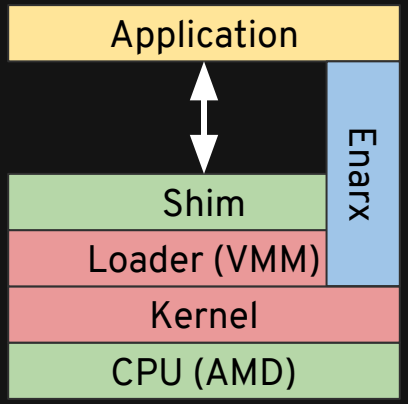
Layers (now) - process-based Keep



Layers - VM-based Keep

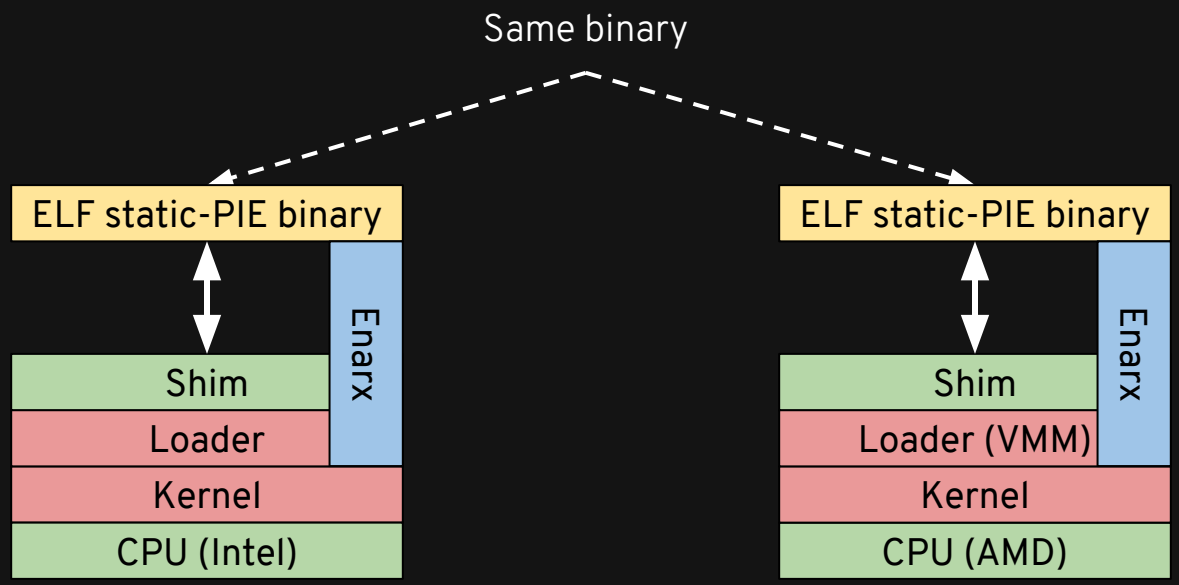


Layers (now) - process-based Keep

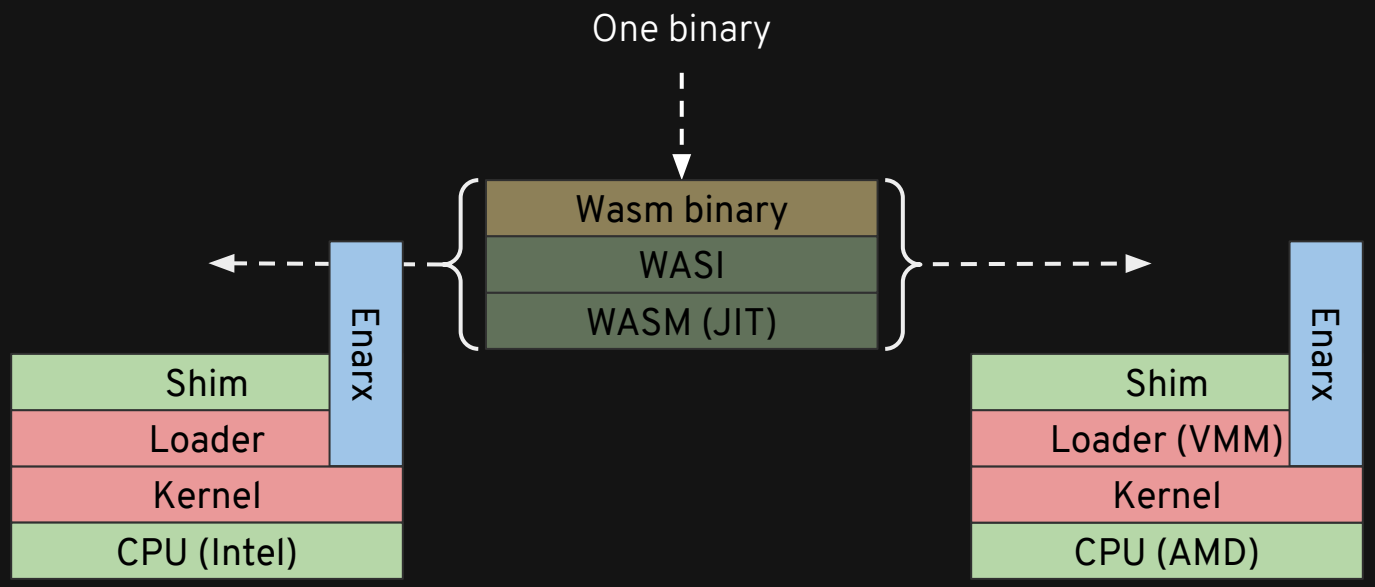


Where ~~we would like to~~ be

Where we are



Where we'd like to be *next*

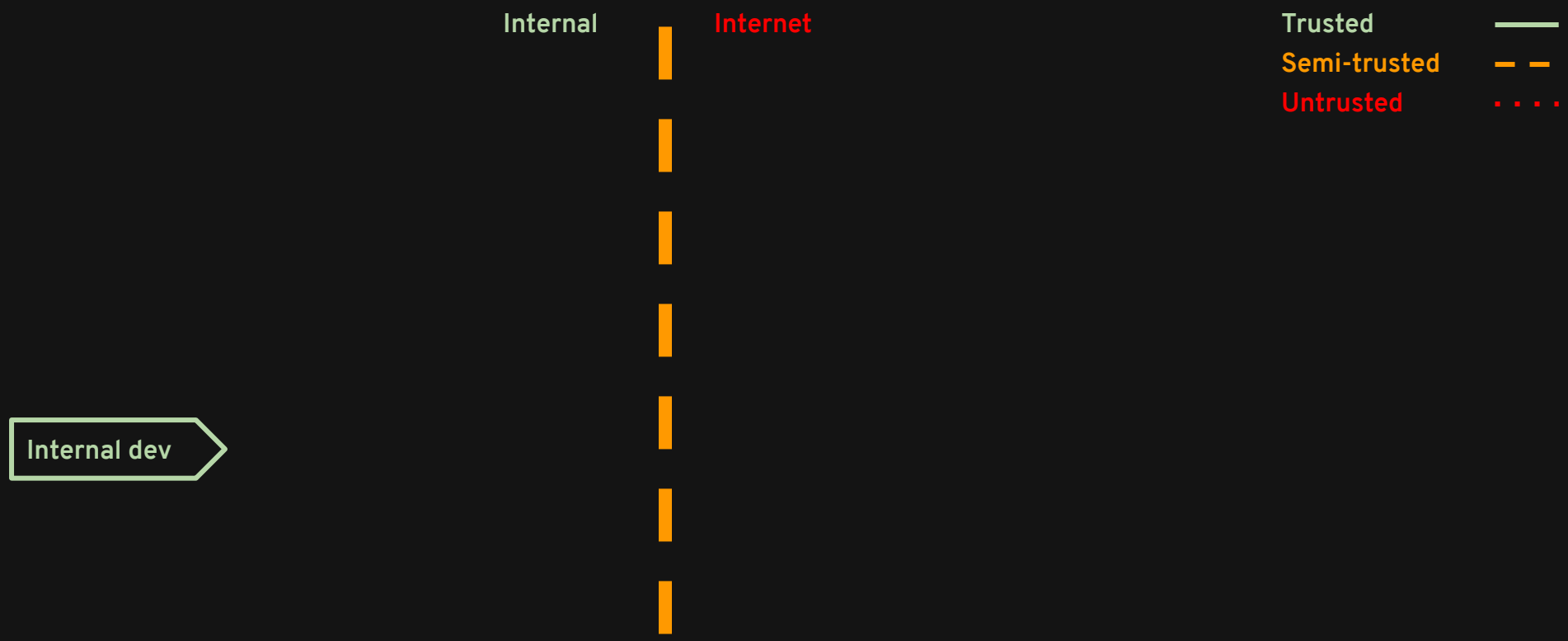


Thinking ahead

Open hybrid cloud and Enarx

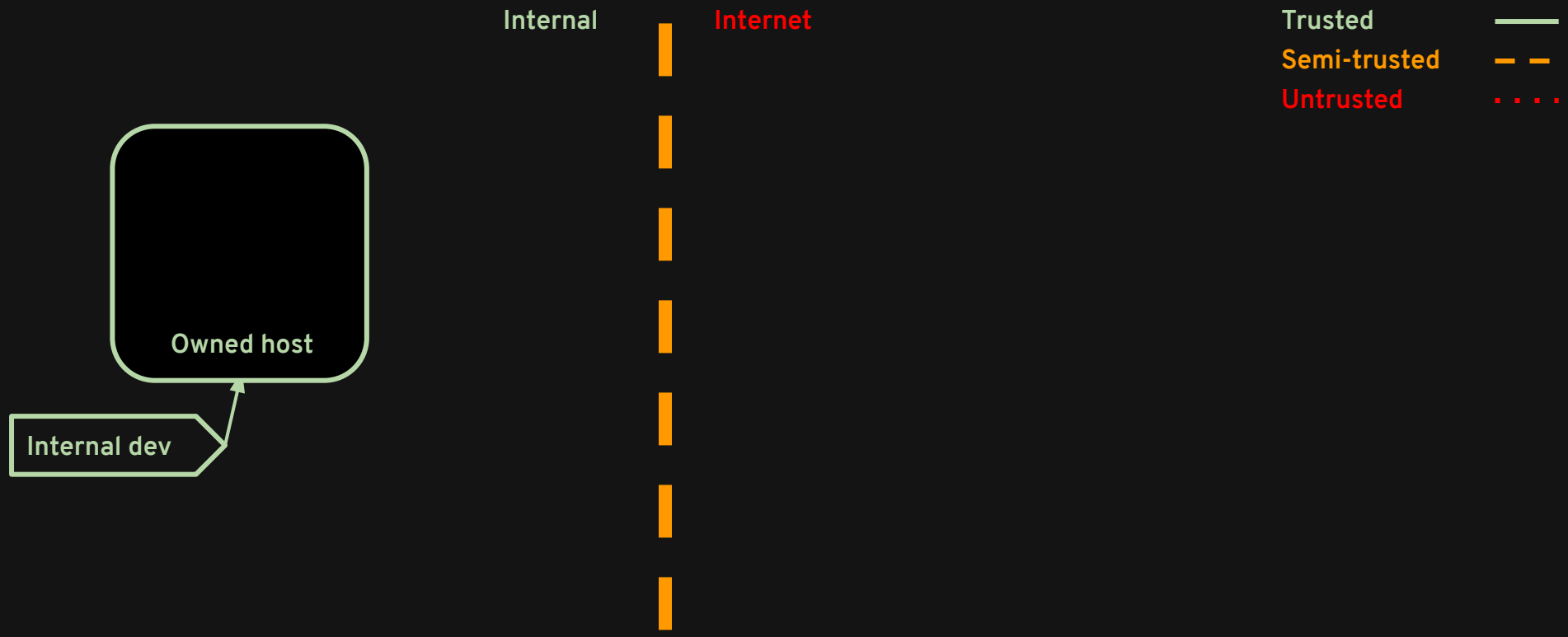
Red Hat

Step 1: on premises



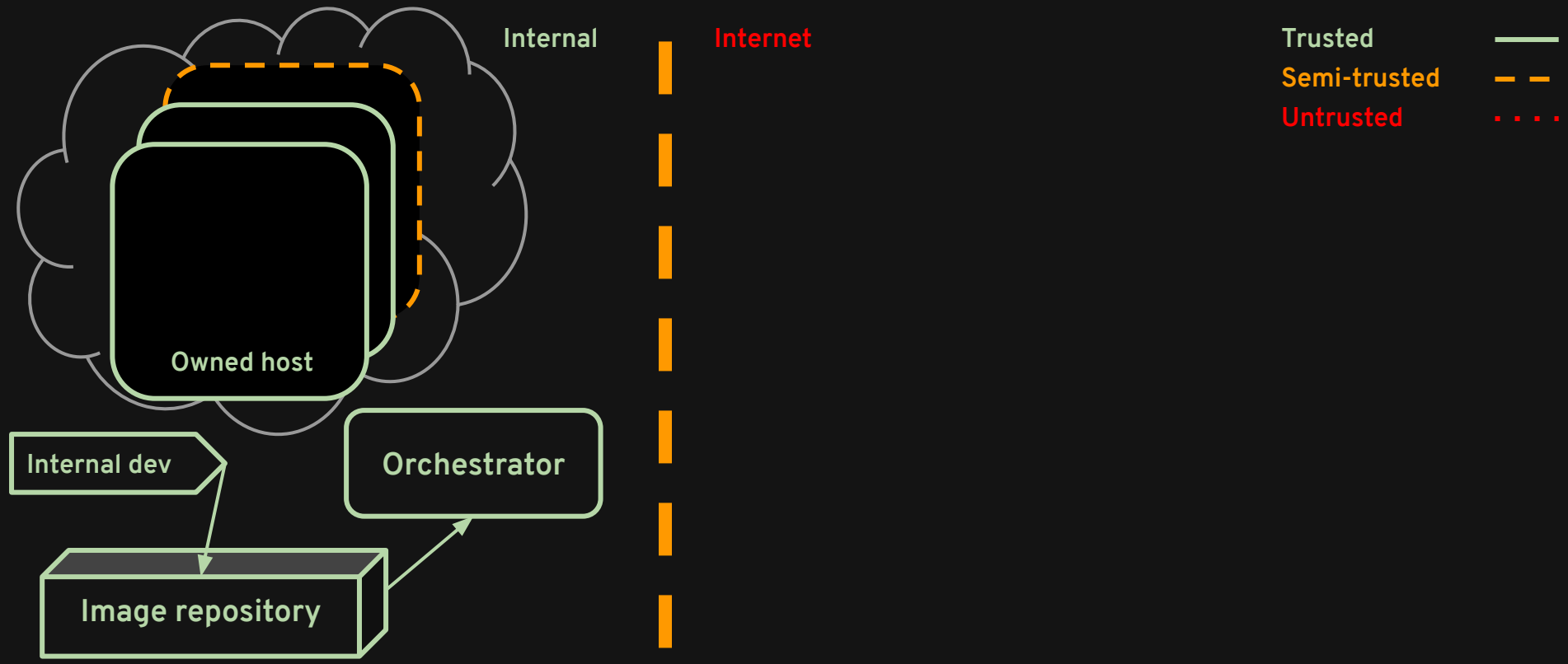
Red Hat

Step 1: on premises



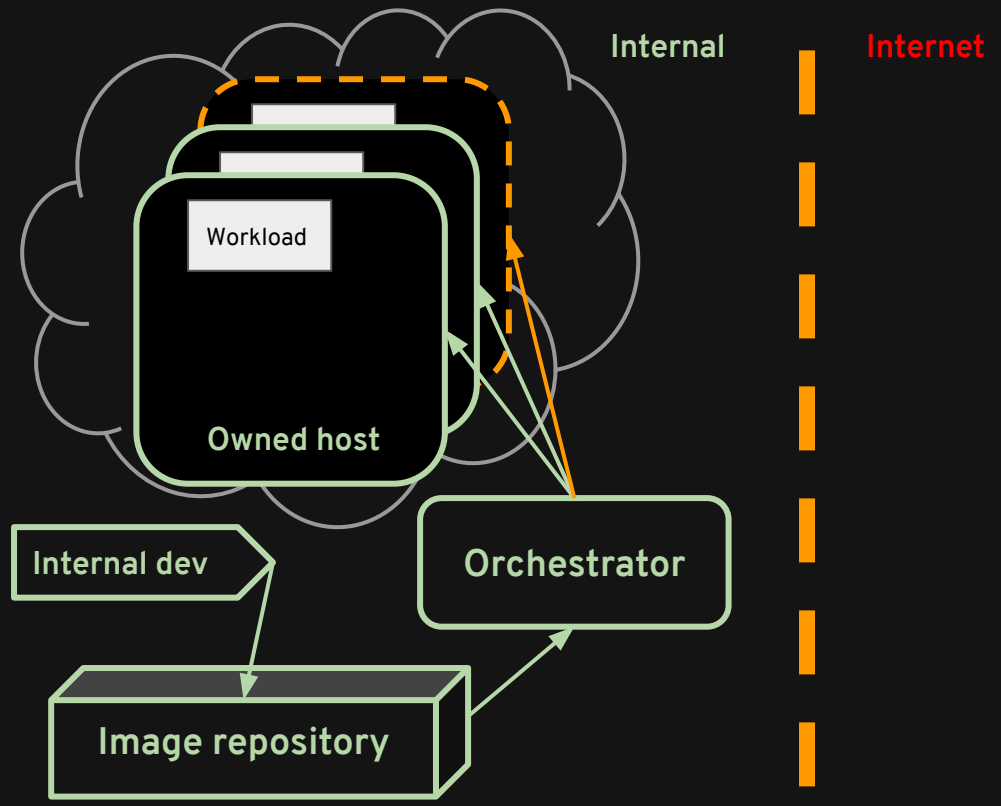
Red Hat

Step 2: private cloud



Red Hat

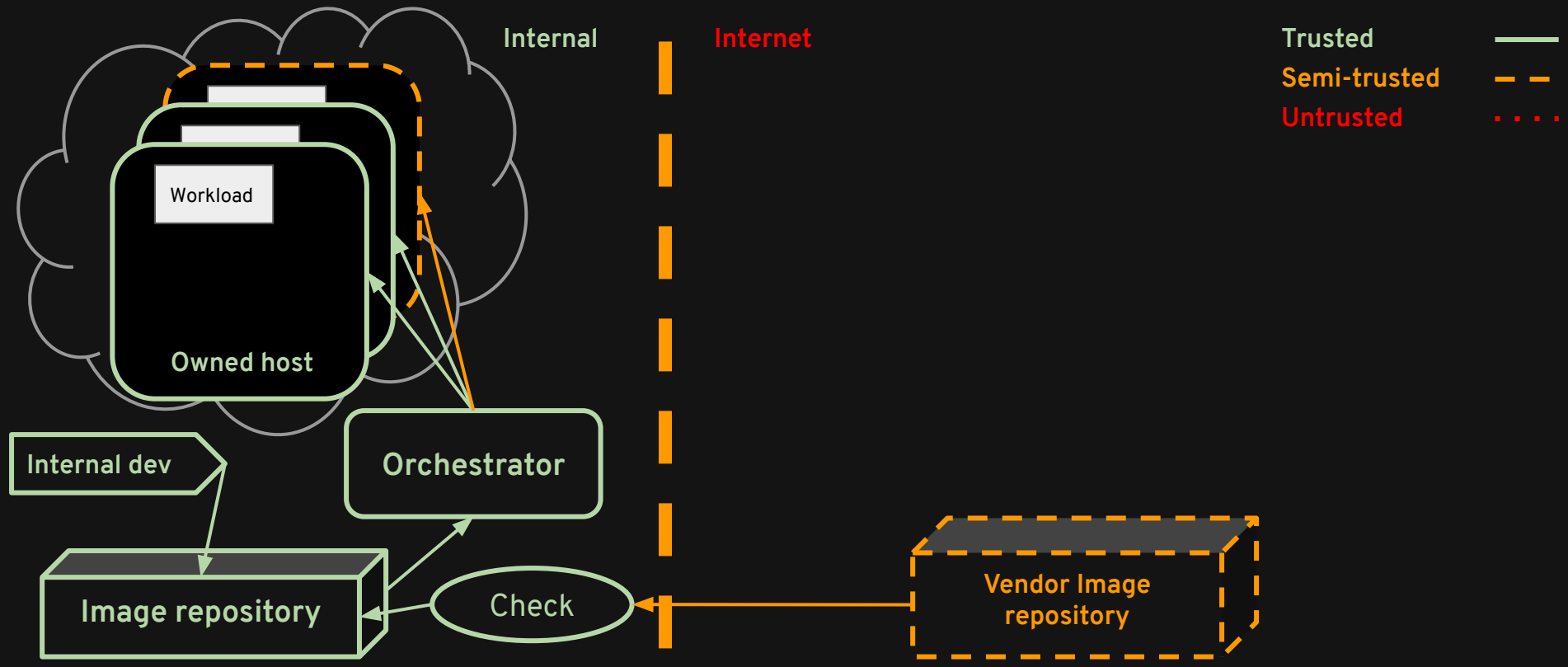
Step 2: private cloud



Trusted
Semi-trusted
Untrusted

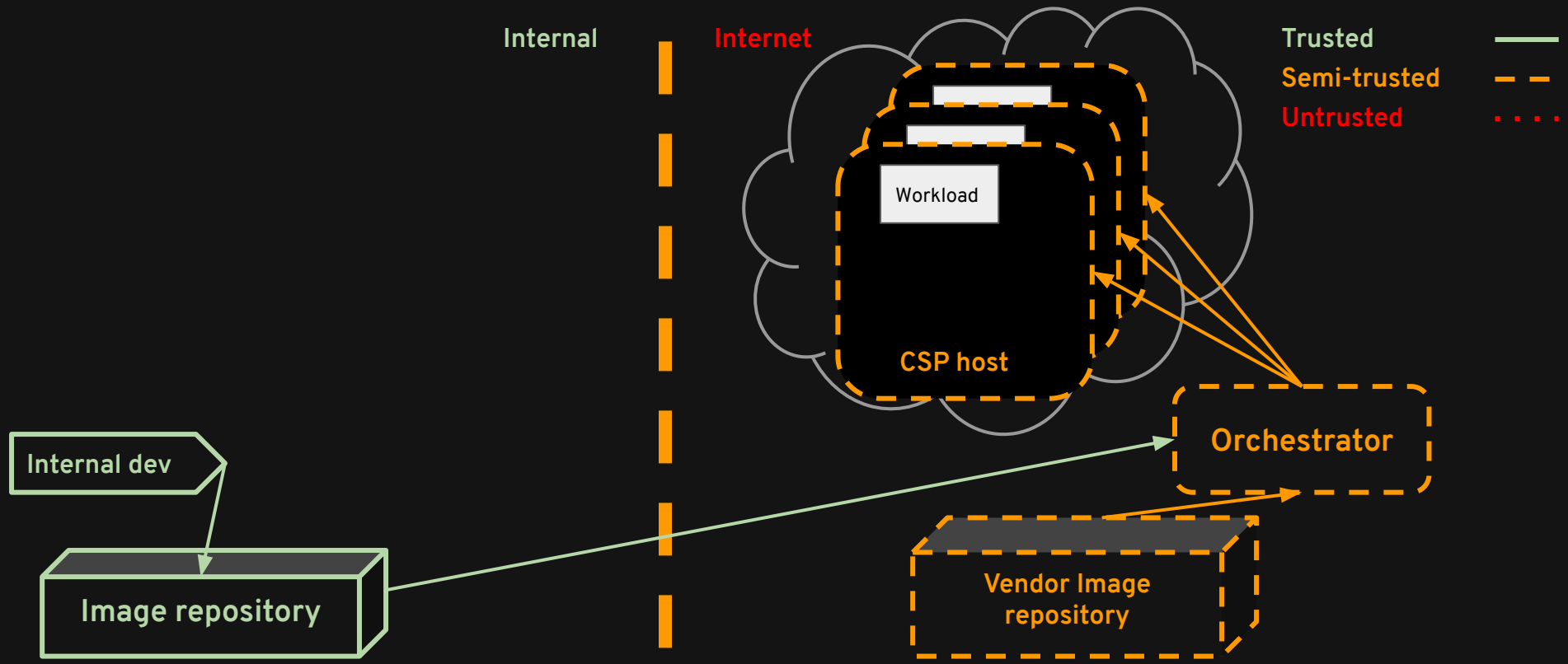
Red Hat

Step 2: private cloud



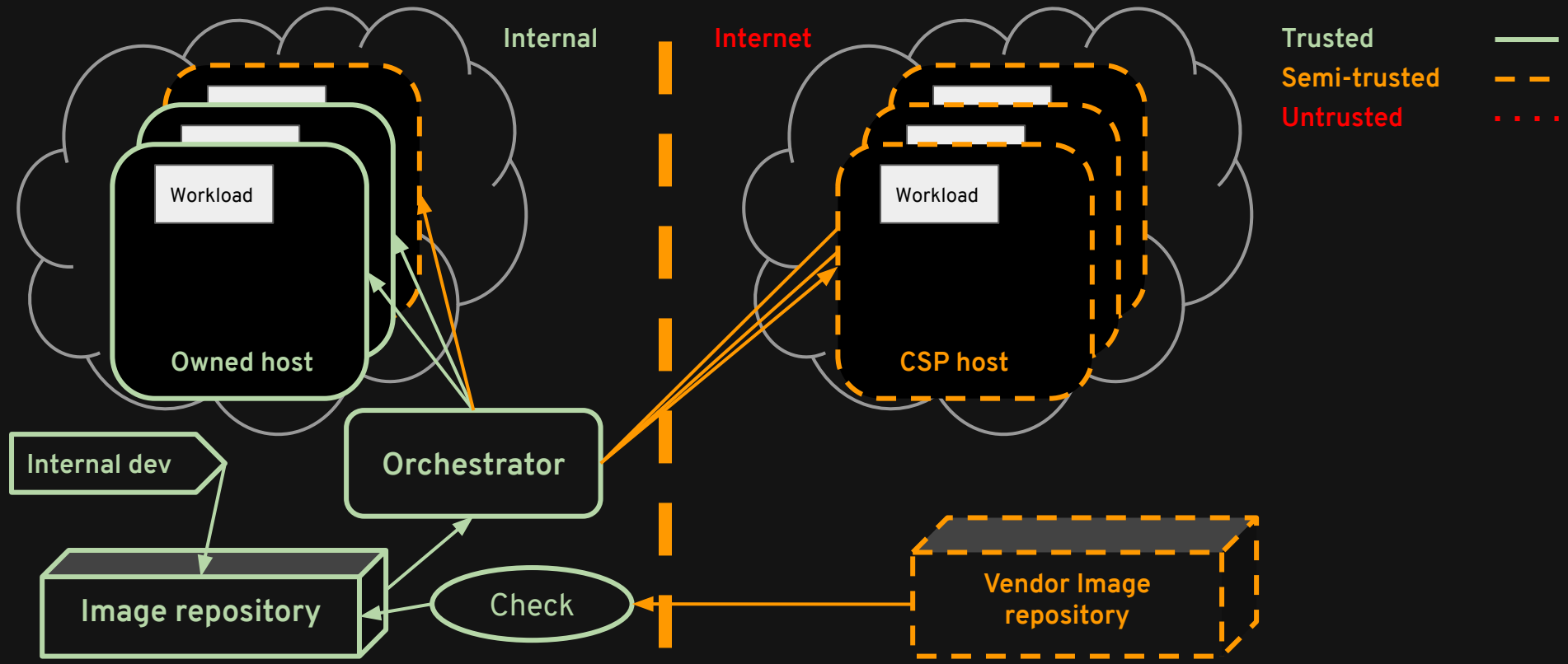
Red Hat

Step 3: public cloud



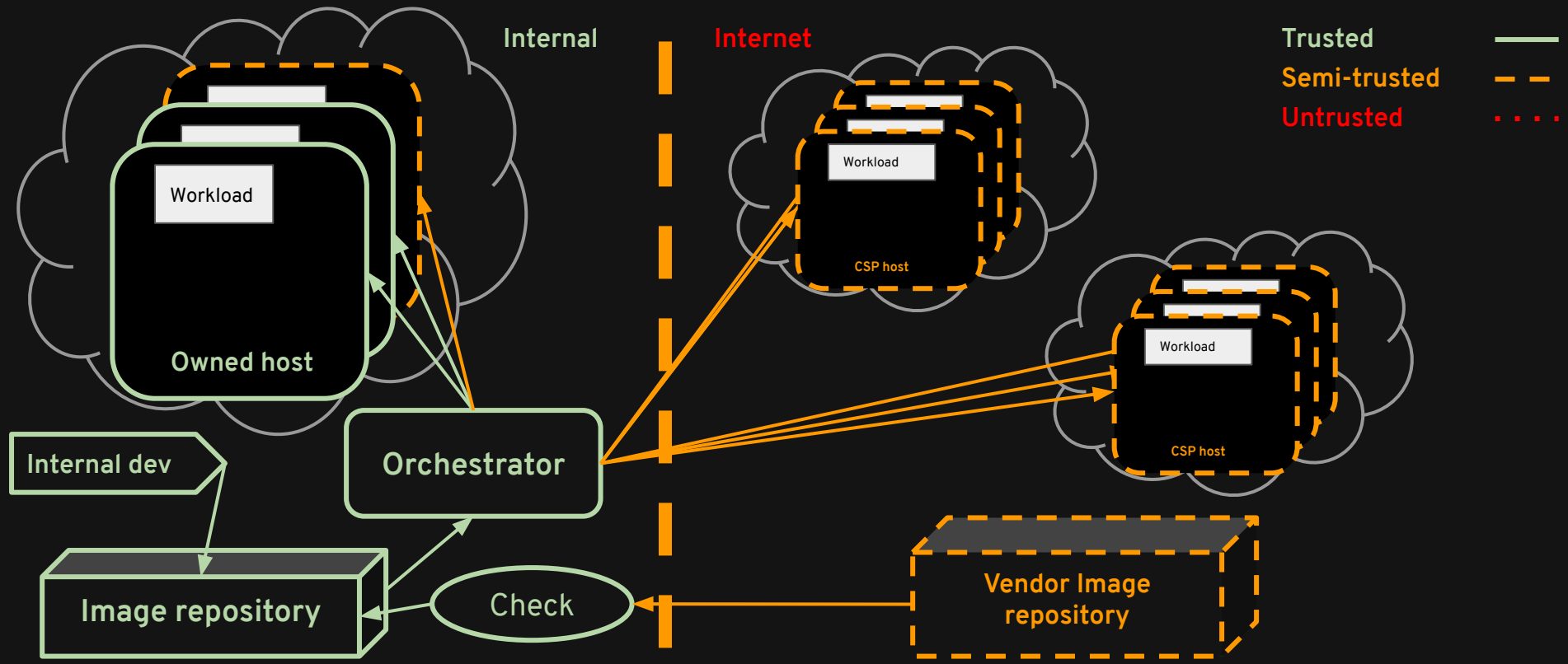
Red Hat

Step 4: hybrid cloud



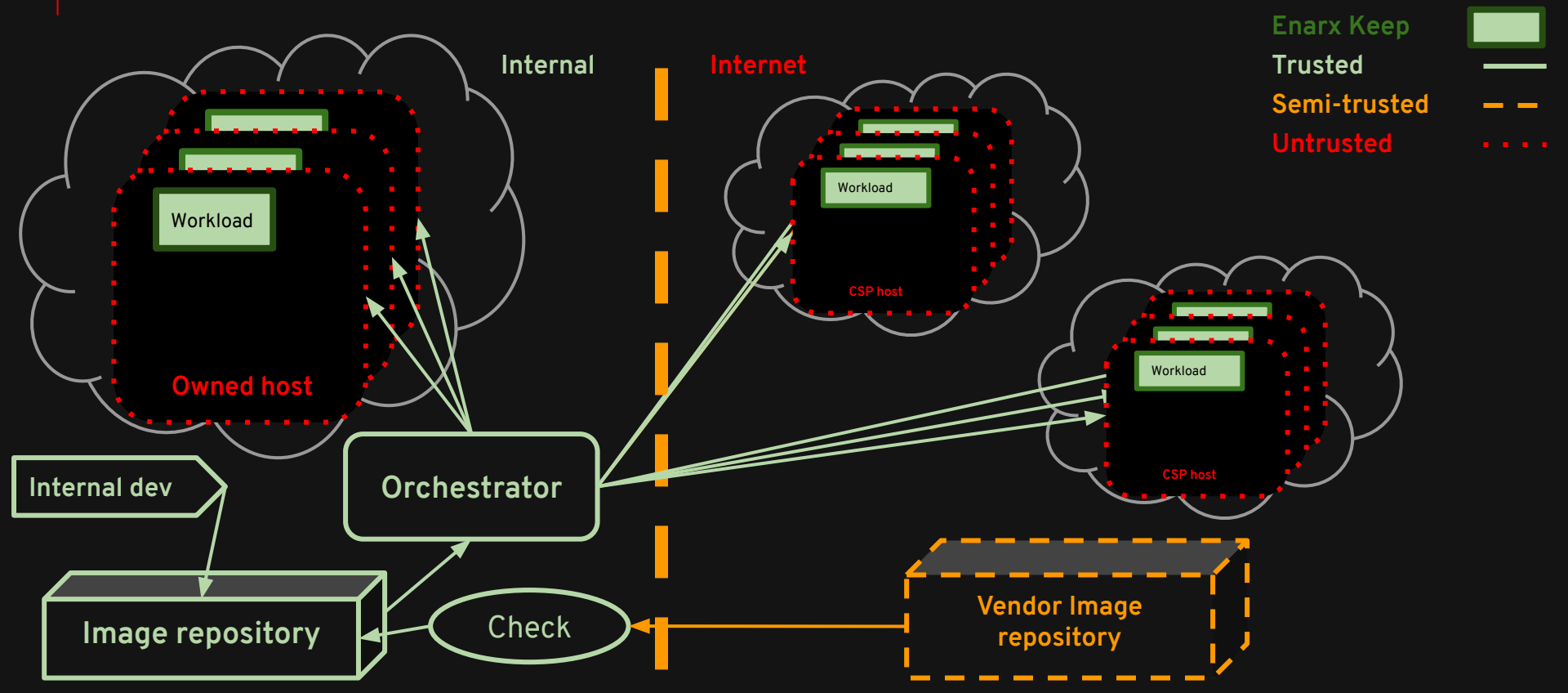
Red Hat

Step 5: hybrid multicloud



Red Hat

Step 6: Enarx hybrid multicloud

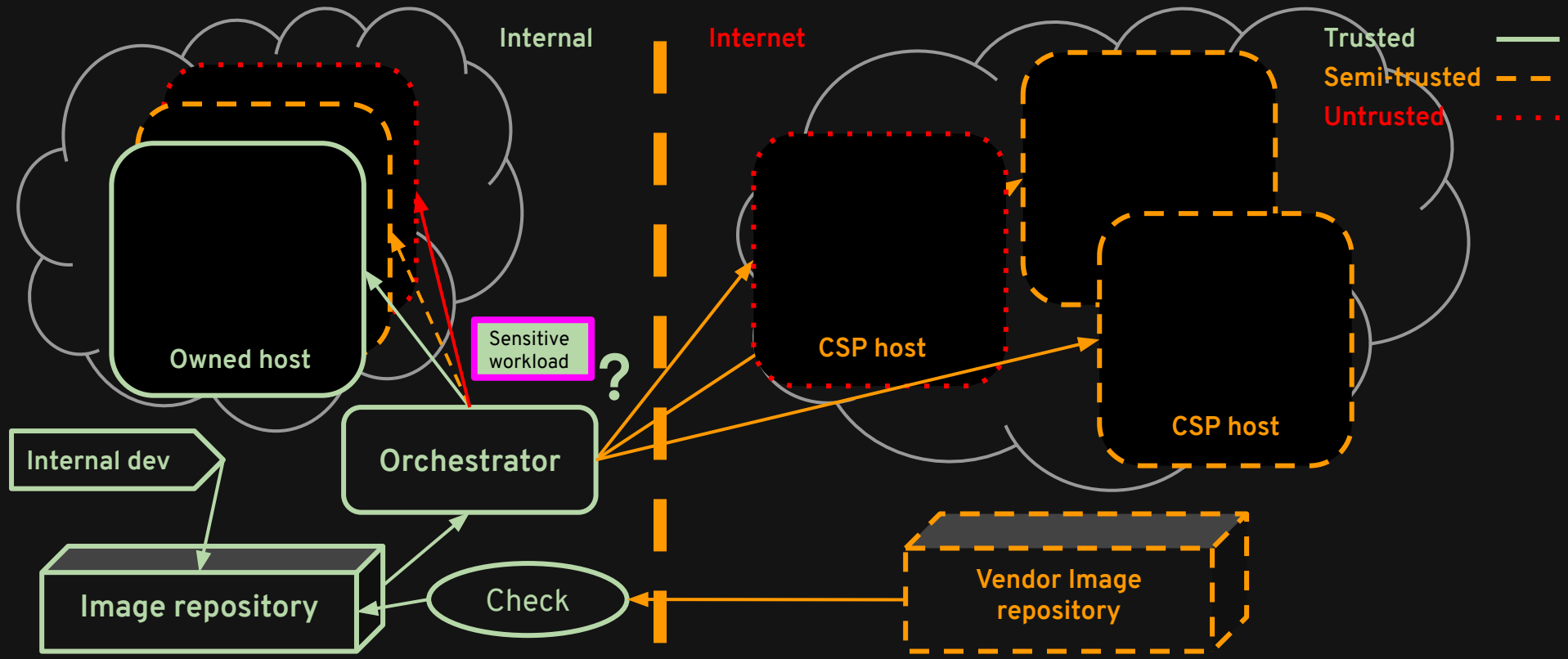


Red Hat

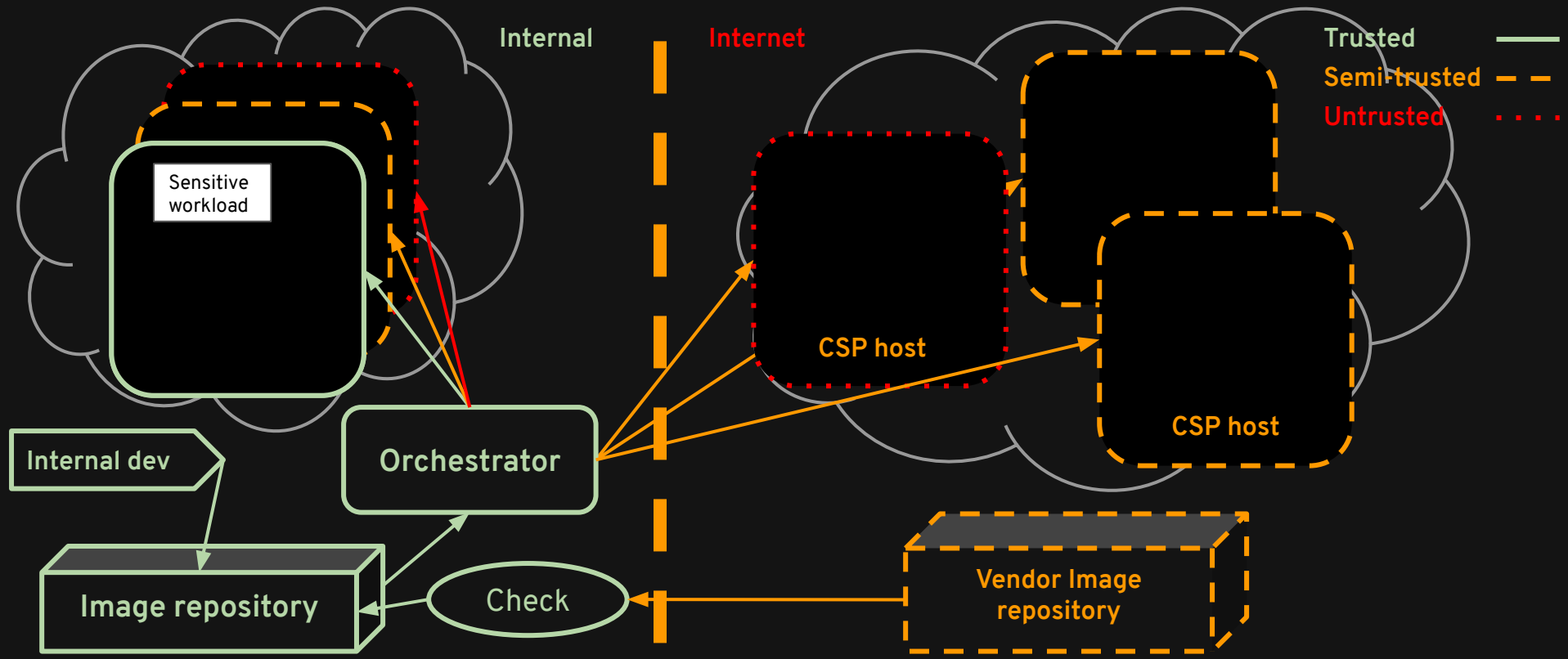
Thinking ahead

New options for workloads with Enarx

Mix and match for different workload types & Enarx

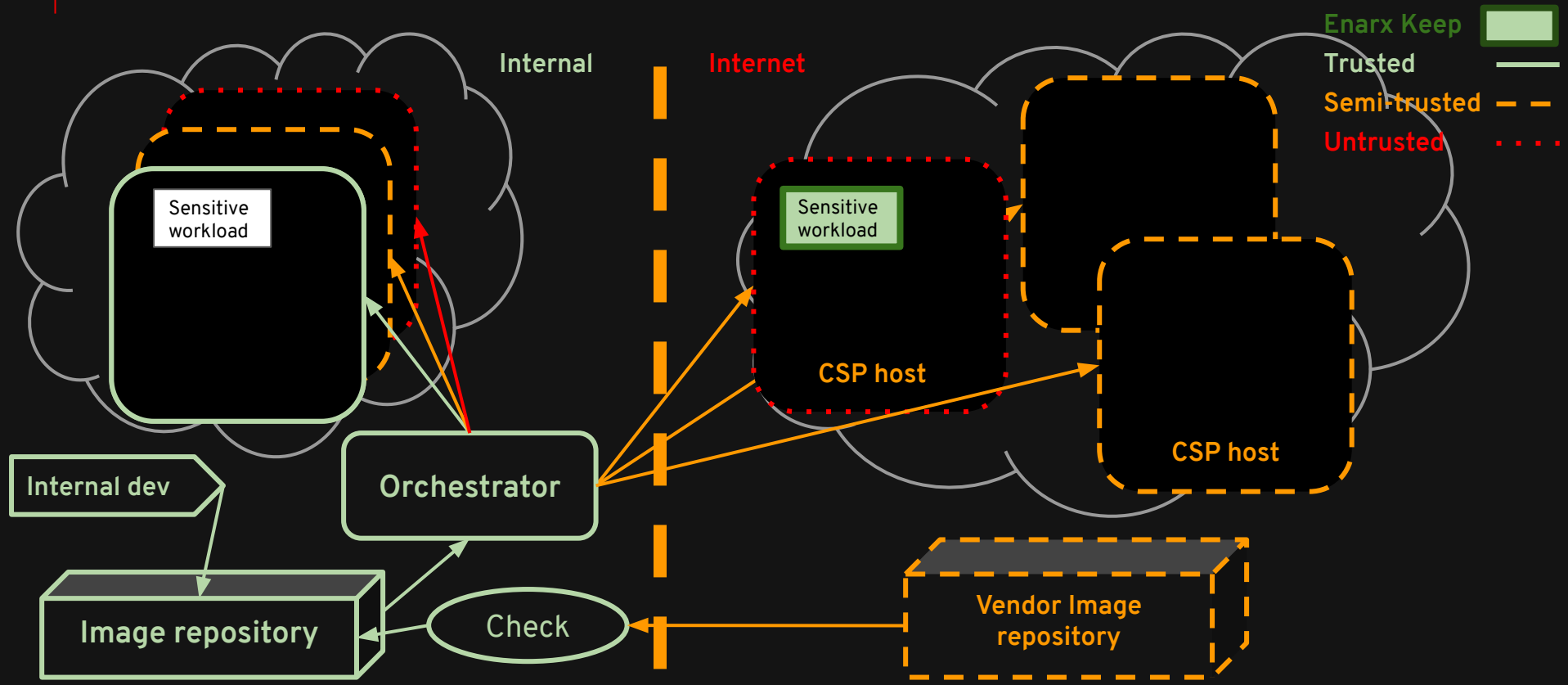


Mix and match for different workload types & Enarx



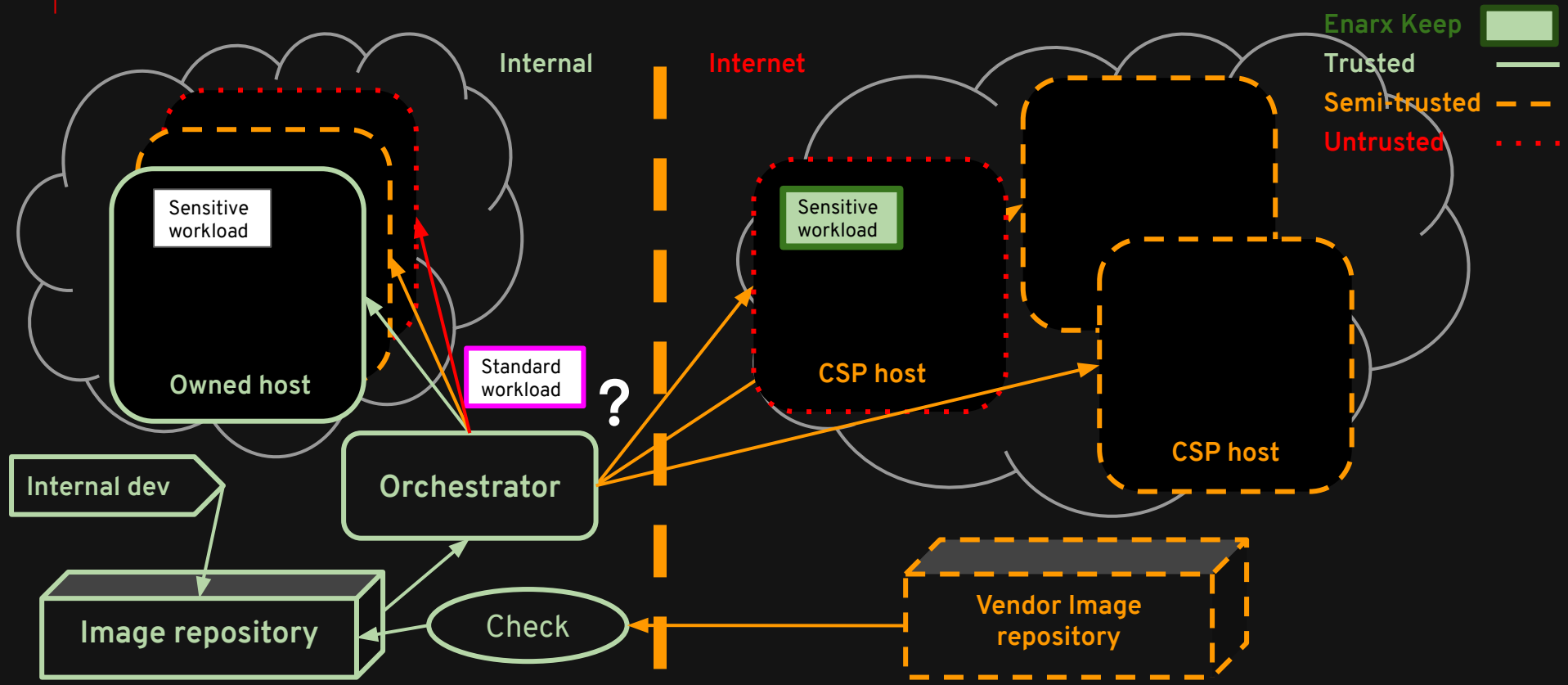
Red Hat

Mix and match for different workload types & Enarx



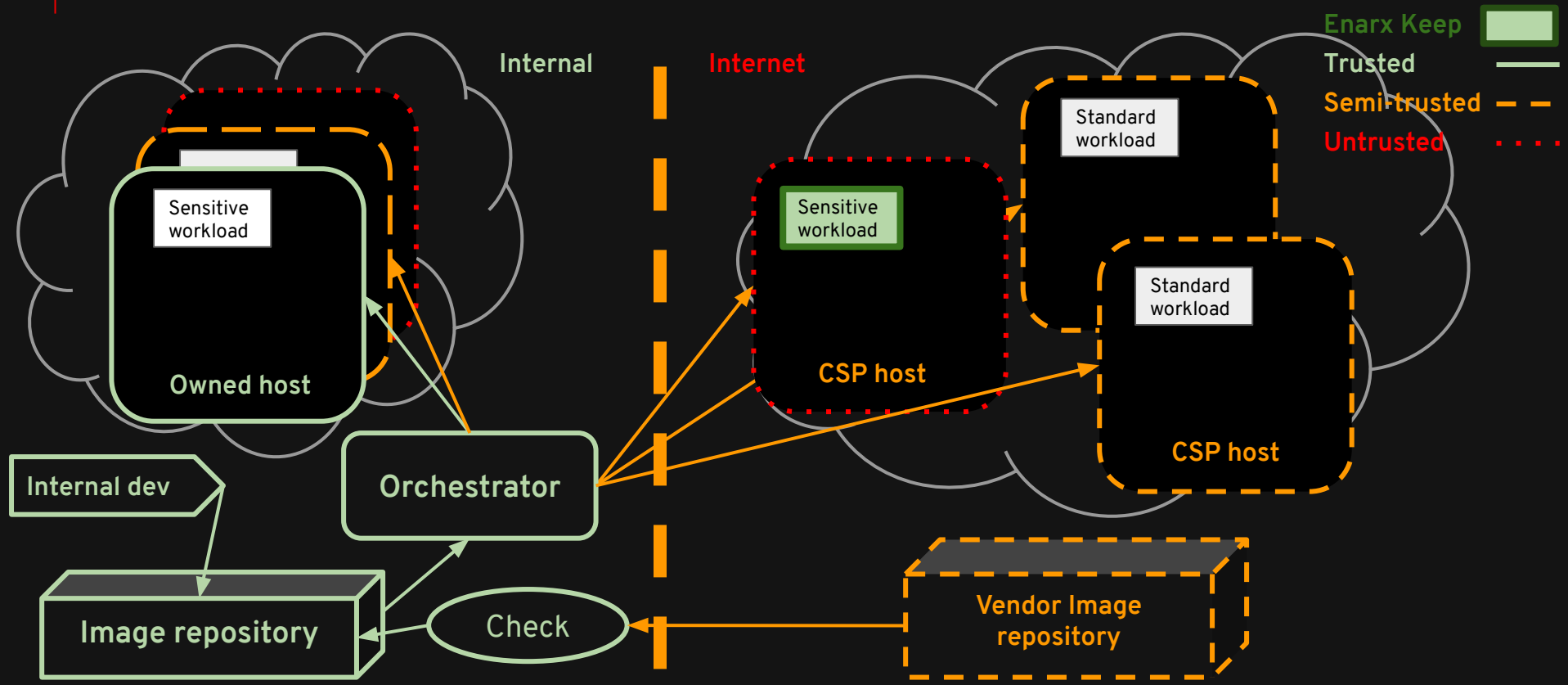
Red Hat

Mix and match for different workload types & Enarx



Red Hat

Mix and match for different workload types & Enarx



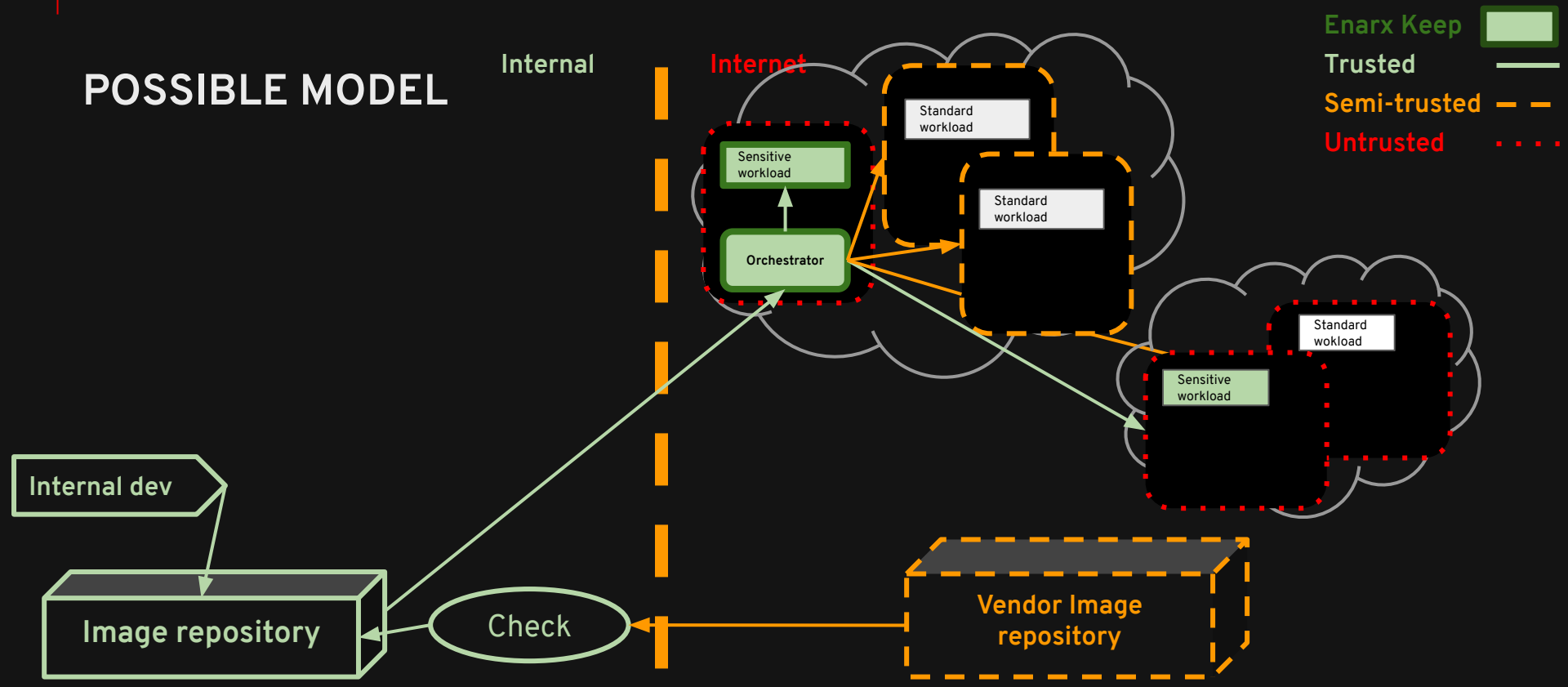
Red Hat

Thinking ahead

New options for orchestration with Enarx

Mix and match for different workload types & Enarx

POSSIBLE MODEL

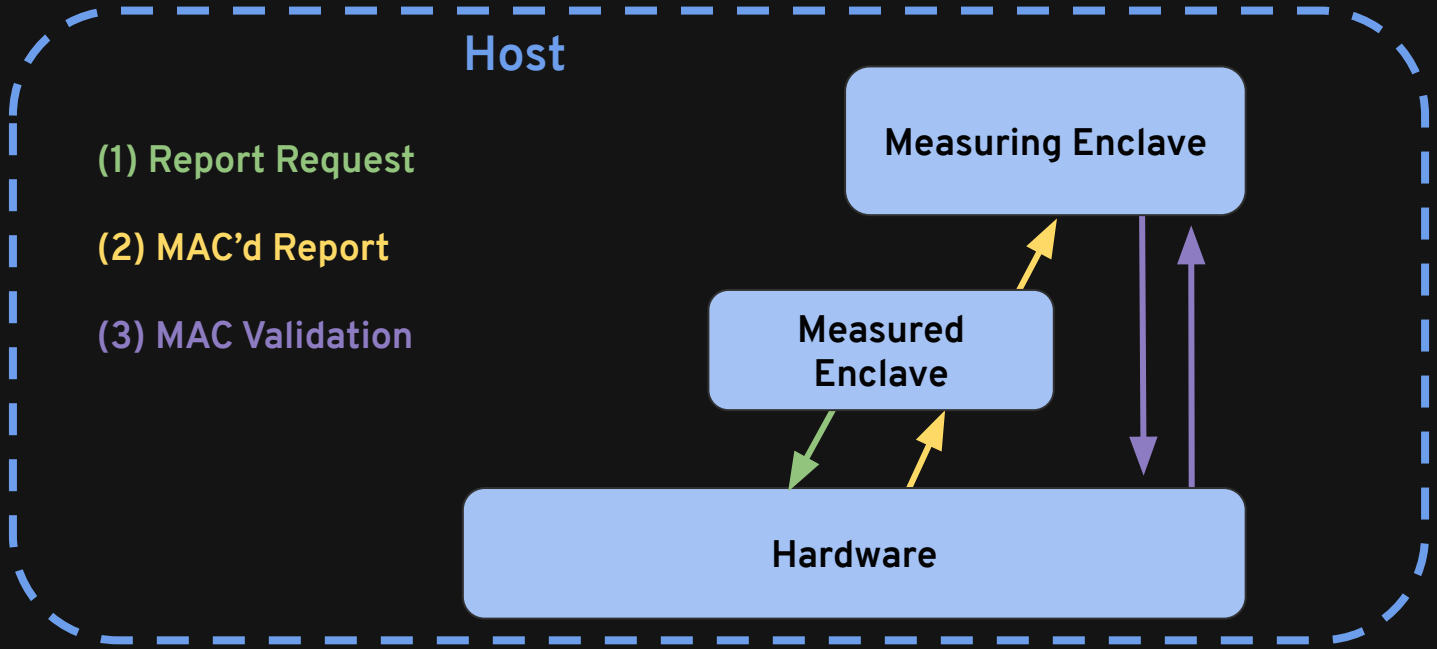


Red Hat

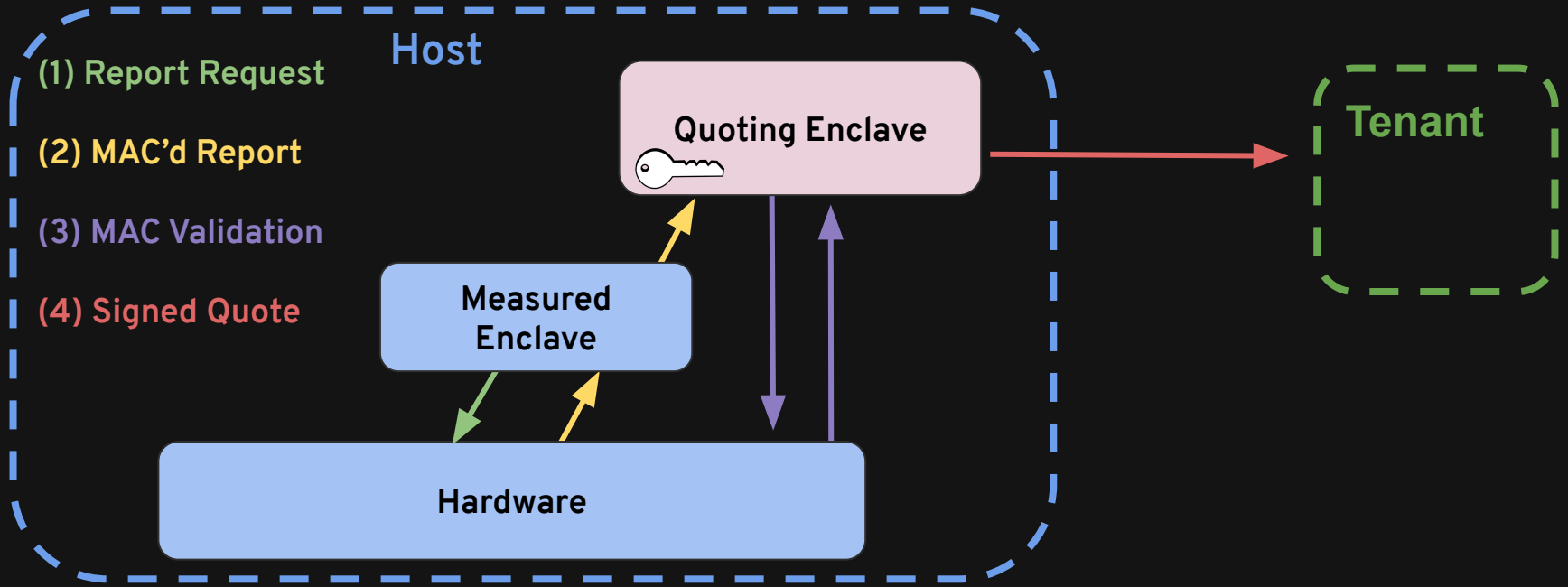
Technical details

Intel SGX

SGX Local Attestation

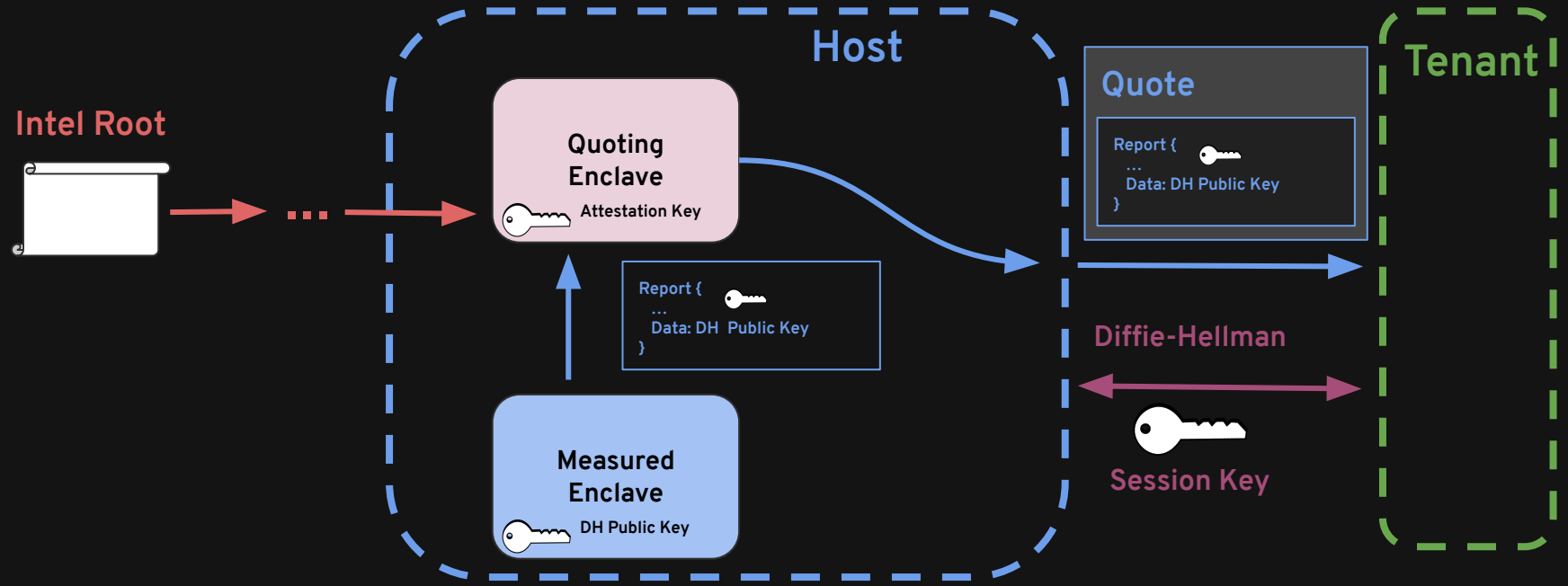


SGX Remote Attestation



Red Hat

SGX Secure Session

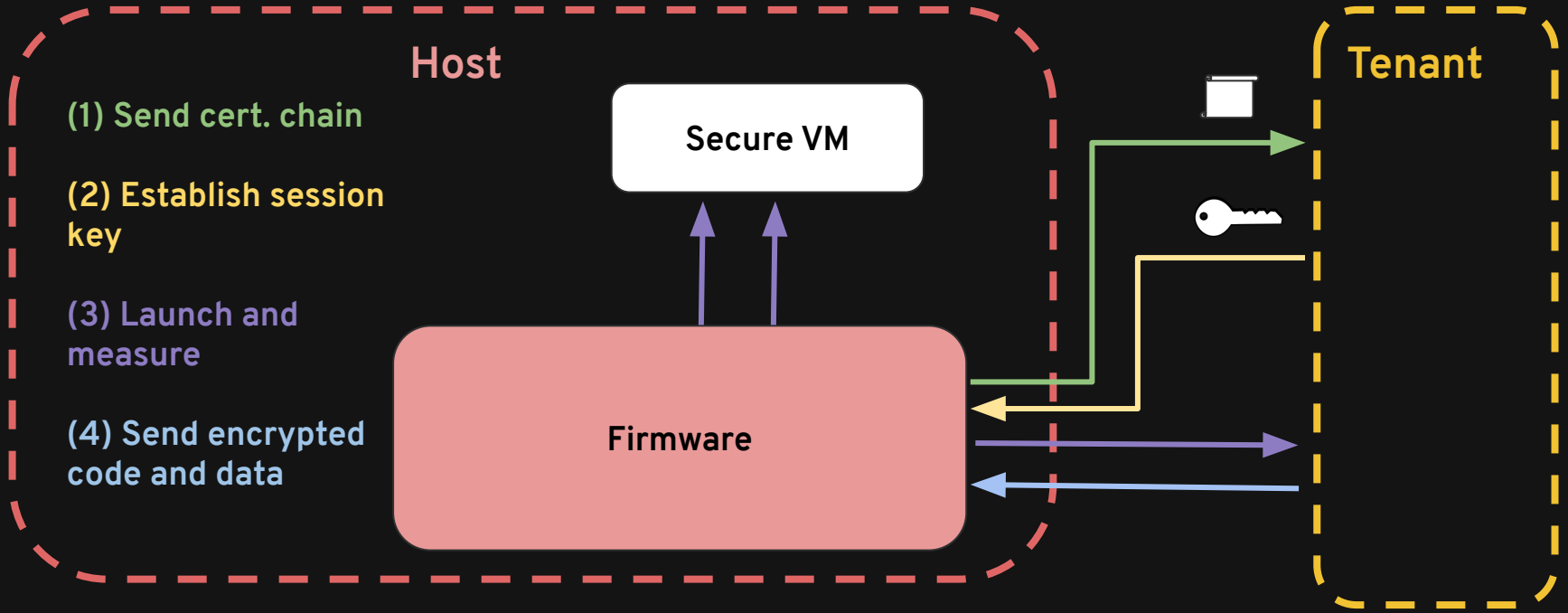


Red Hat

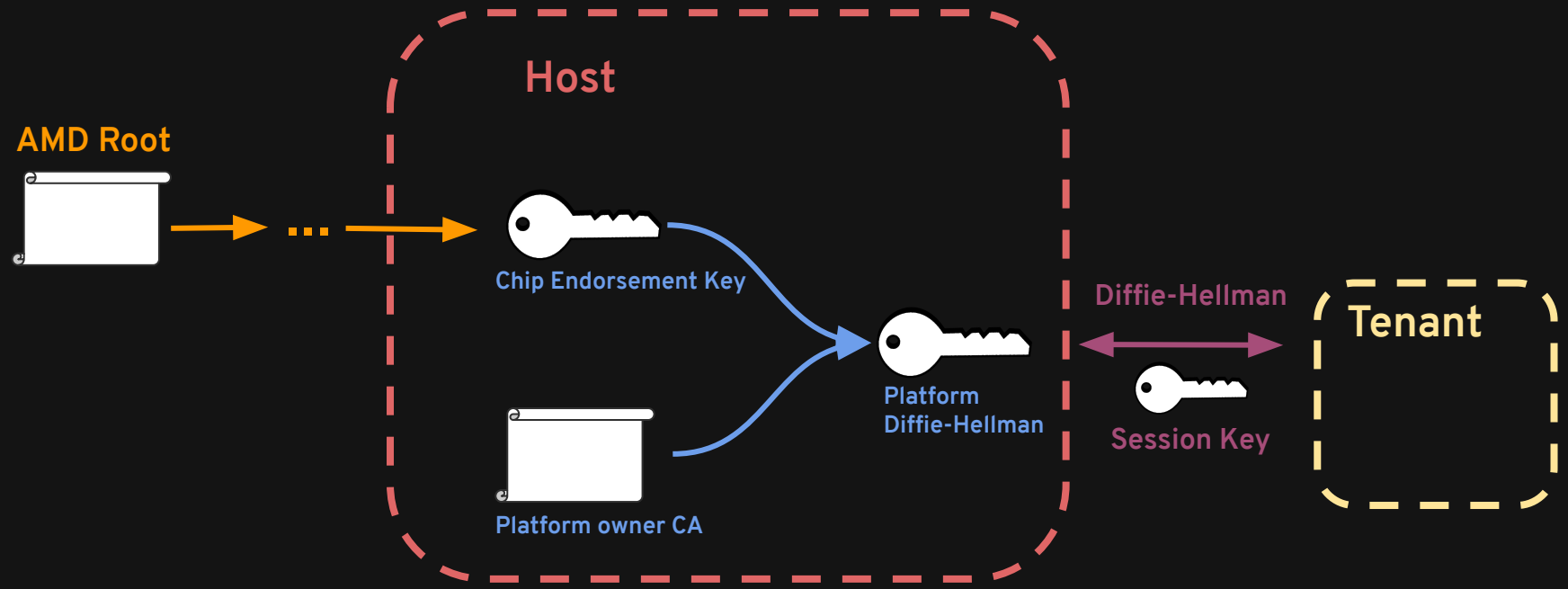
Technical details

AMD SEV

SEV Attestation



SEV Secure Session



Technical details

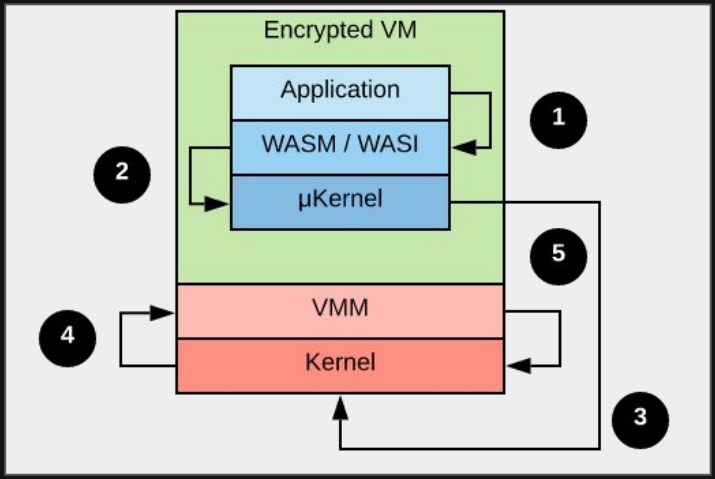
Enarx Virtualization Architecture

VM-based Keep

Red Hat

Syscall Propagation - VM-based

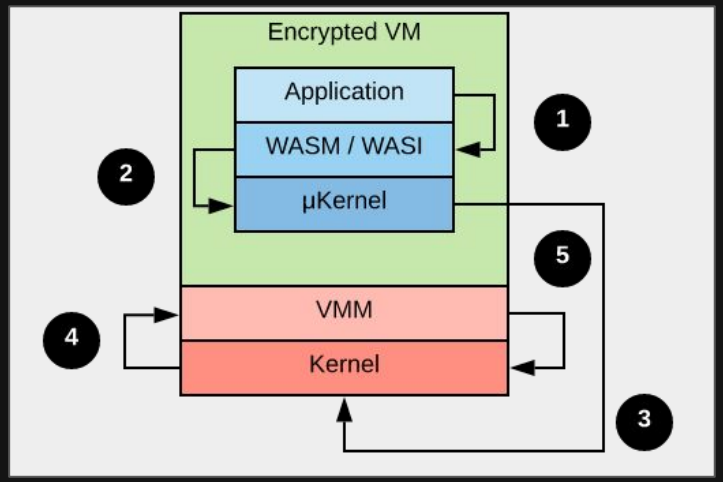
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.



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

Syscall Propagation - VM-based

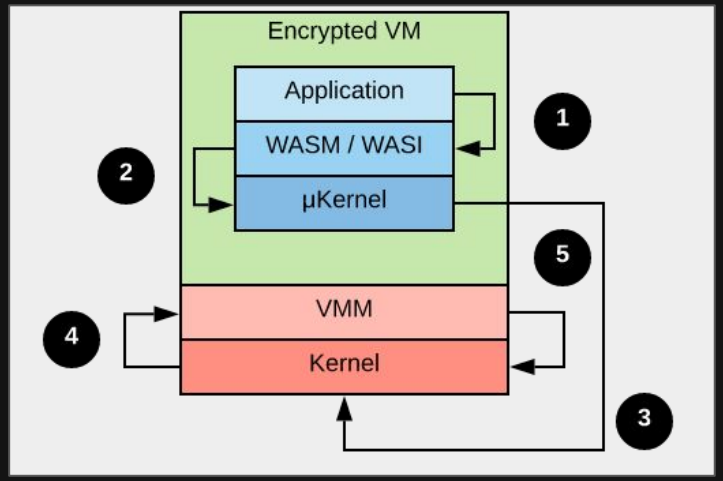
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.
2. The hand-crafted Rust code translates the WASI call into a Linux read() syscall, leaving Ring 3 to jump into the µKernel, which handles some syscalls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



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

Syscall Propagation - VM-based

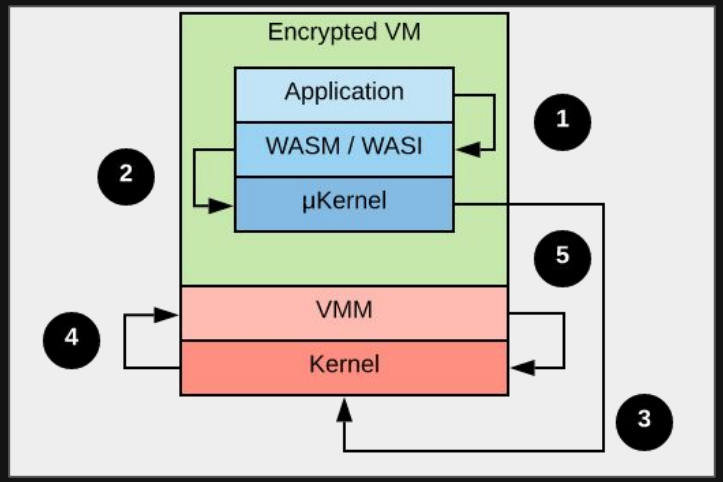
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.
2. The hand-crafted Rust code translates the WASI call into a Linux read() syscall, leaving Ring 3 to jump into the µKernel, which handles some syscalls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

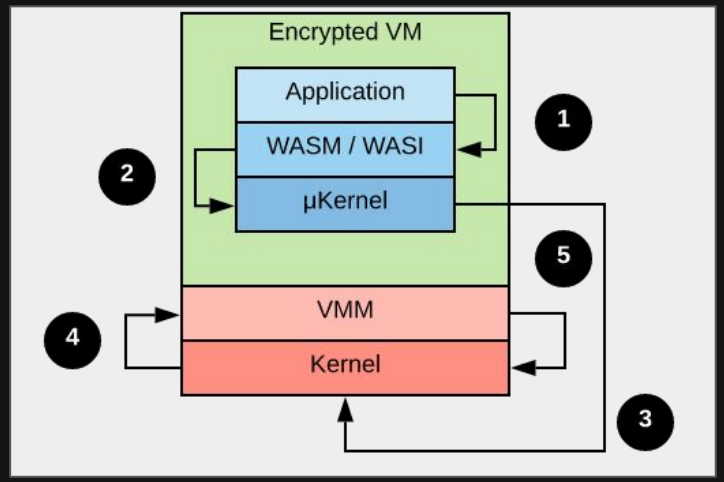
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.
2. The hand-crafted Rust code translates the WASI call into a Linux read() syscall, leaving Ring 3 to jump into the µKernel, which handles some syscalls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

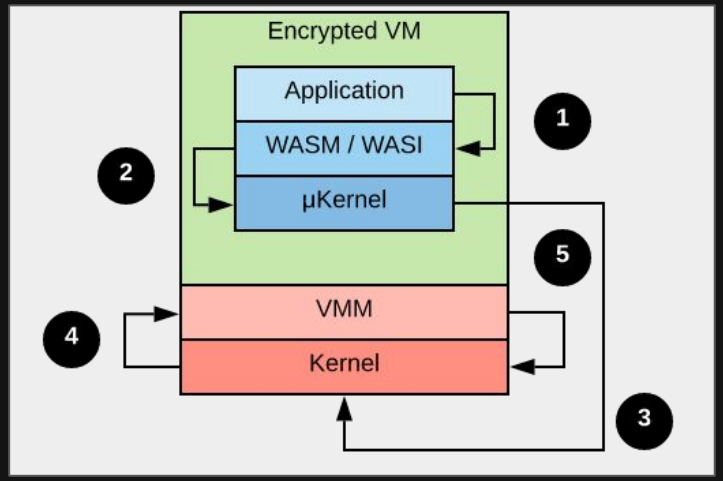
1. An Enarx application, compiled to WebAssembly, makes a WASI call, causing a transition from the JIT-compiled code into our guest userspace Rust code.
2. The hand-crafted Rust code translates the WASI call into a Linux read() syscall, leaving Ring 3 to jump into the µKernel, which handles some syscalls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

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 are an open project

Code	✓ GitHub	
Wiki	✓ GitHub	
Design	✓ GitHub	
Issues & PRs	✓ GitHub	
Chat	✓ Rocket.Chat	(Thank you!)
CI/CD resources	✓ Metal.equinix.com	(Thank you!)
Stand-ups	✓ Open to all	
Diversity	✓ Contributor Covenant CofC	

CONFIDENTIAL COMPUTING CONSORTIUM

Red Hat

We Need Your Help!

Website:	https://enarx.dev
Code:	https://github.com/enarx
Chat:	https://chat.enarx.dev
License:	Apache 2.0
Language:	Rust

Daily stand-ups open to all! Check the website wiki for details.

Questions?

https://enarx.dev

Red Hat