A technical introduction to Enarx

Rust + WebAssembly + TEEs = Confidential Computing

Mike Bursell
Office of the CTO

Nathaniel McCallum
Sr. Principal Engineer

https://enarx.dev

Red Hat

State of the project

(skip to the interesting part)

State of the project

- End-to-end demo available
  - SEV (AMD) - working!
  - SGX (Intel) - imminent!

State of the project

- End-to-end demo available
  - SEV (AMD) - working!
  - SGX (Intel) - imminent!
- Stand-alone PoC framework soon
  - Early February 2021
  - Ready for WebAssembly workloads
  - Restricted networking/storage

State of the project

- End-to-end demo available
  - SEV (AMD)
  - SGX (Intel) - imminent!
- Stand-alone PoC framework soon
  - Early February 2021
  - Ready for WebAssembly workloads
  - Restricted networking/storage
- Documentation next
  - Initial focus on HOWTOs and design documentation

Enarx overview

The Enarx 5-bullet overview

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

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

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)

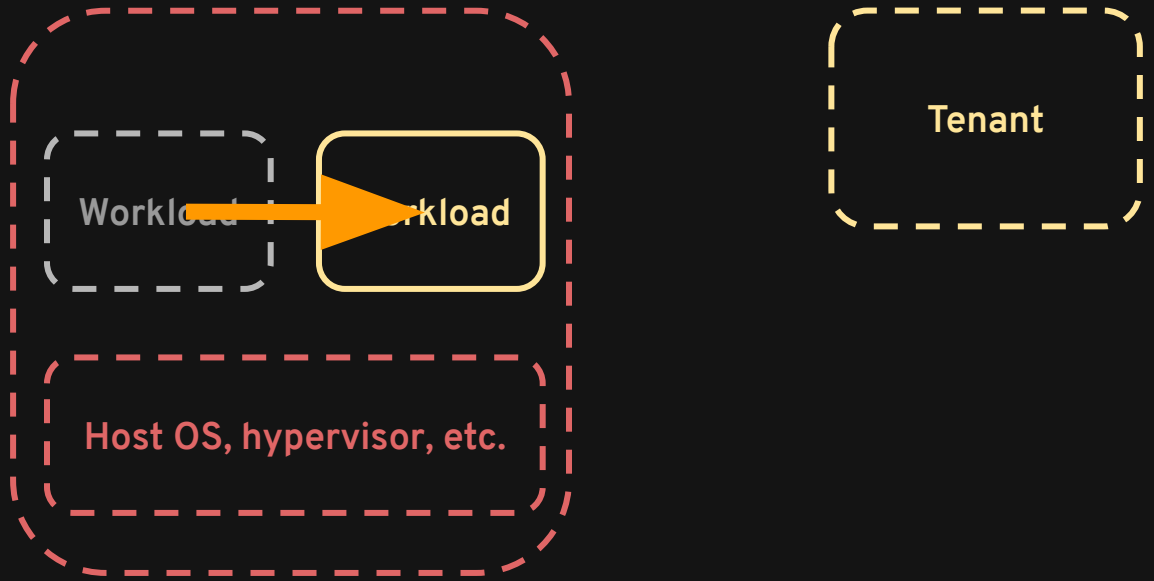
Isolation

Workloads and the host

Red Hat

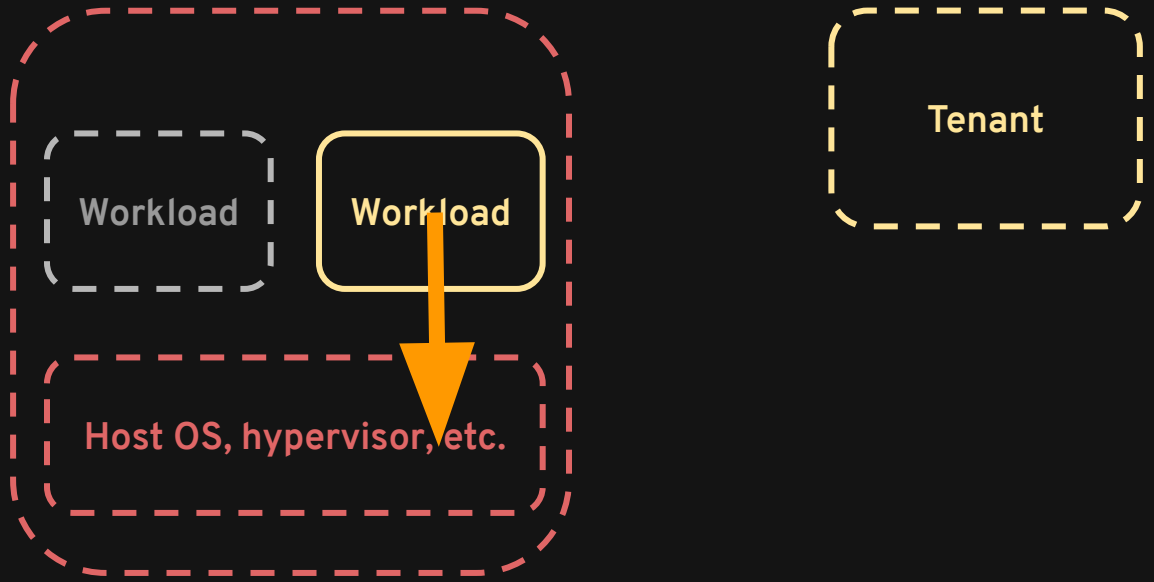
The 3 types of isolation

Workload from workload isolation

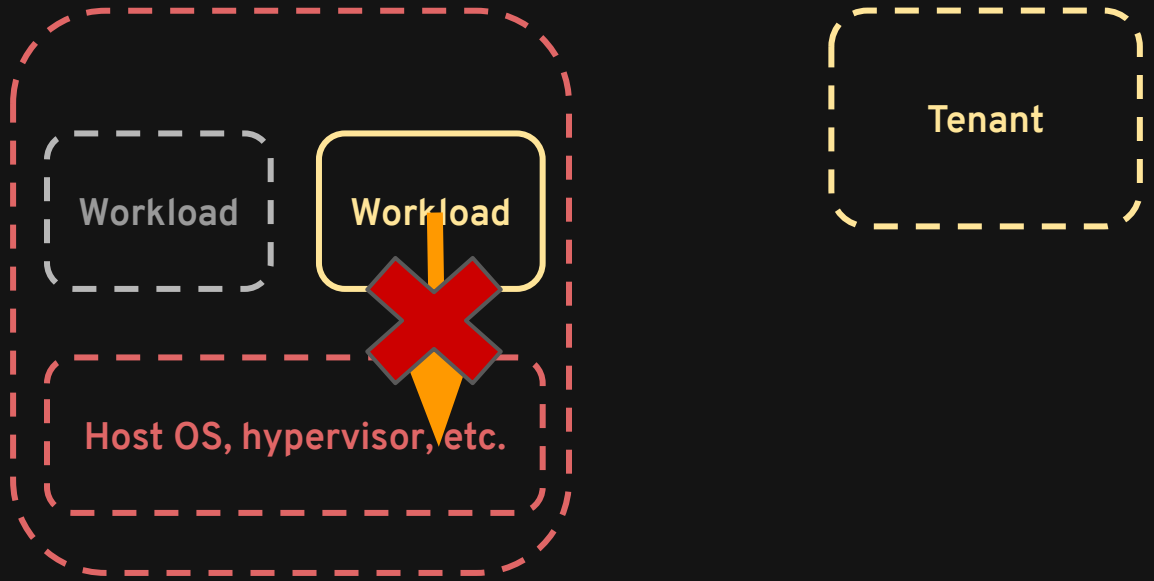


Workload from workload isolation

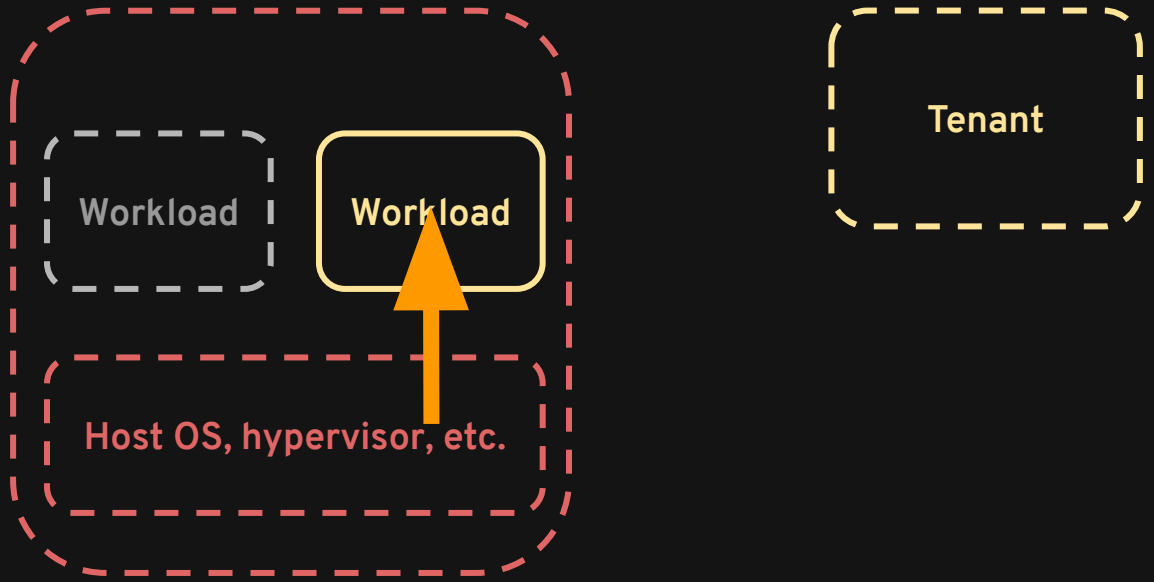
Host from workload isolation



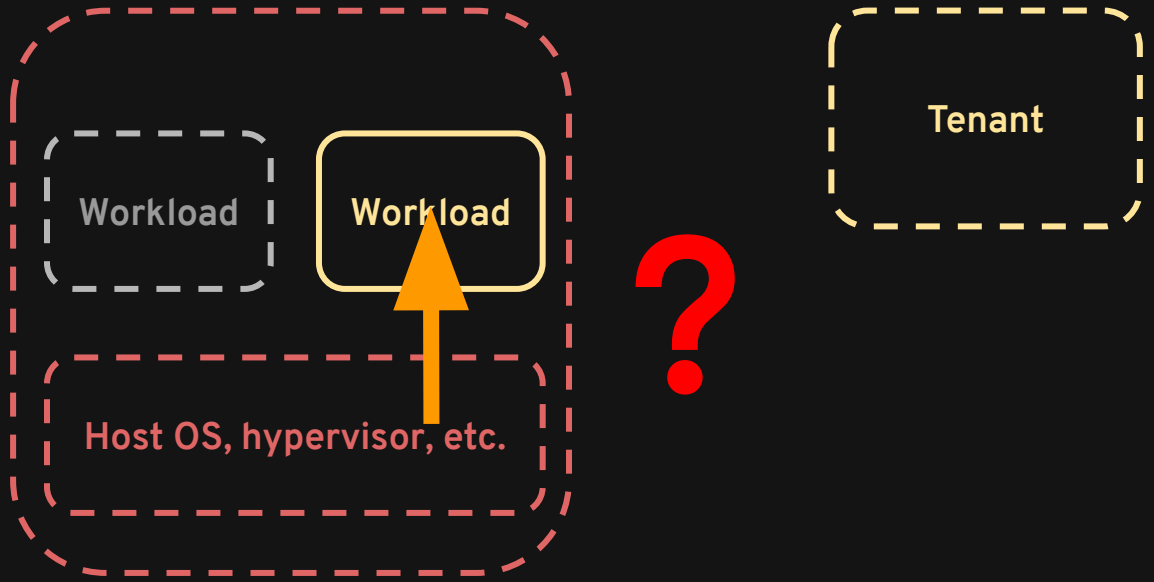
Host from workload isolation



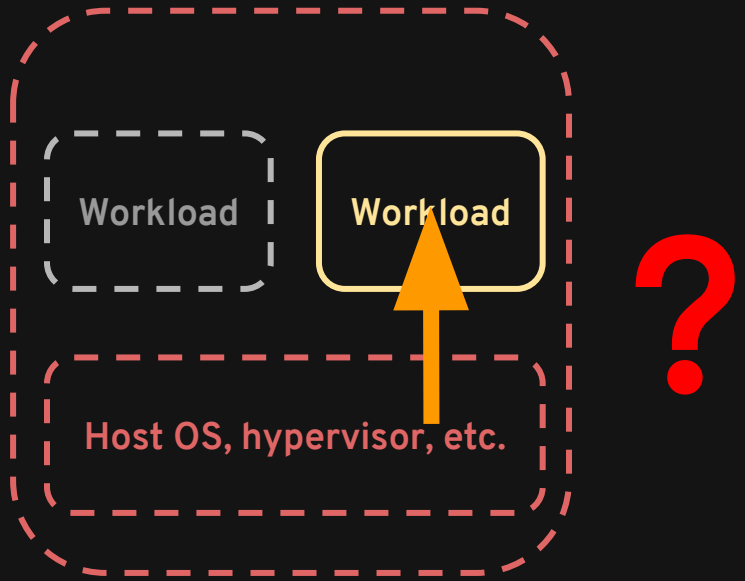
Workload from host isolation



Workload from host isolation



Workload from host isolation



Tenant

Sensitive workloads

Cloud for regulated sectors

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

Vulnerable hosts

- Edge, IoT, ...

TEEs sound great, but…

Red Hat

TEEs sound great, but…

TEEs sound great, but…

1. Different platforms → separate development

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

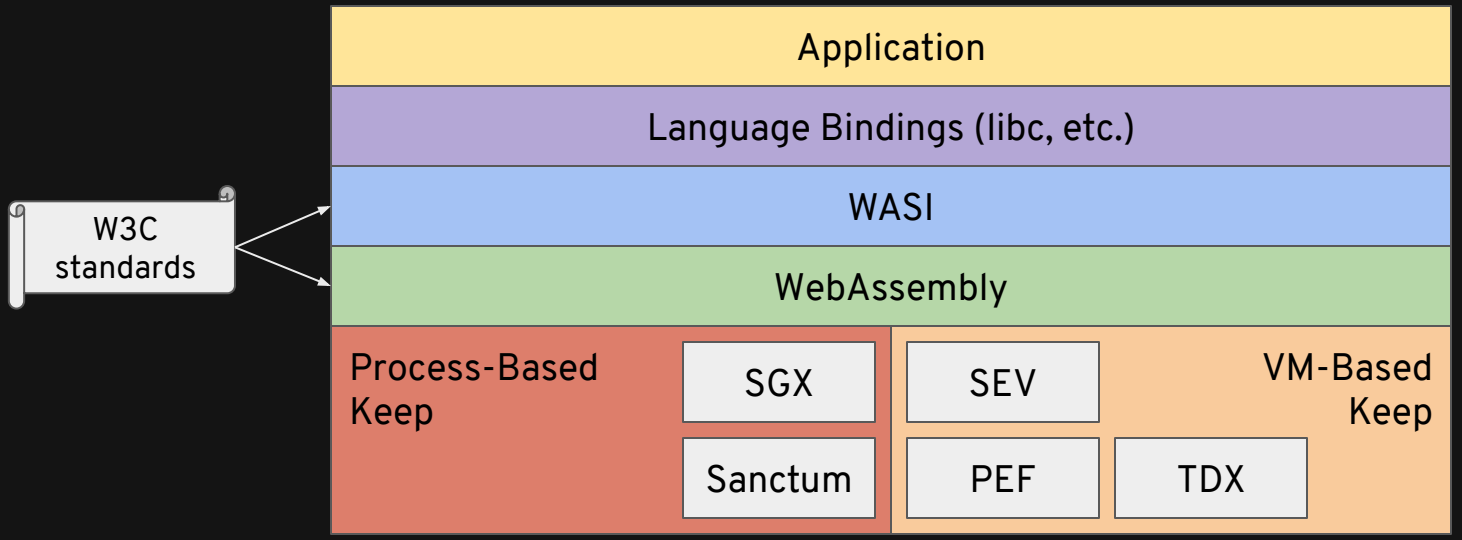
TEEs sound great, but…

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!

Introducing Enarx

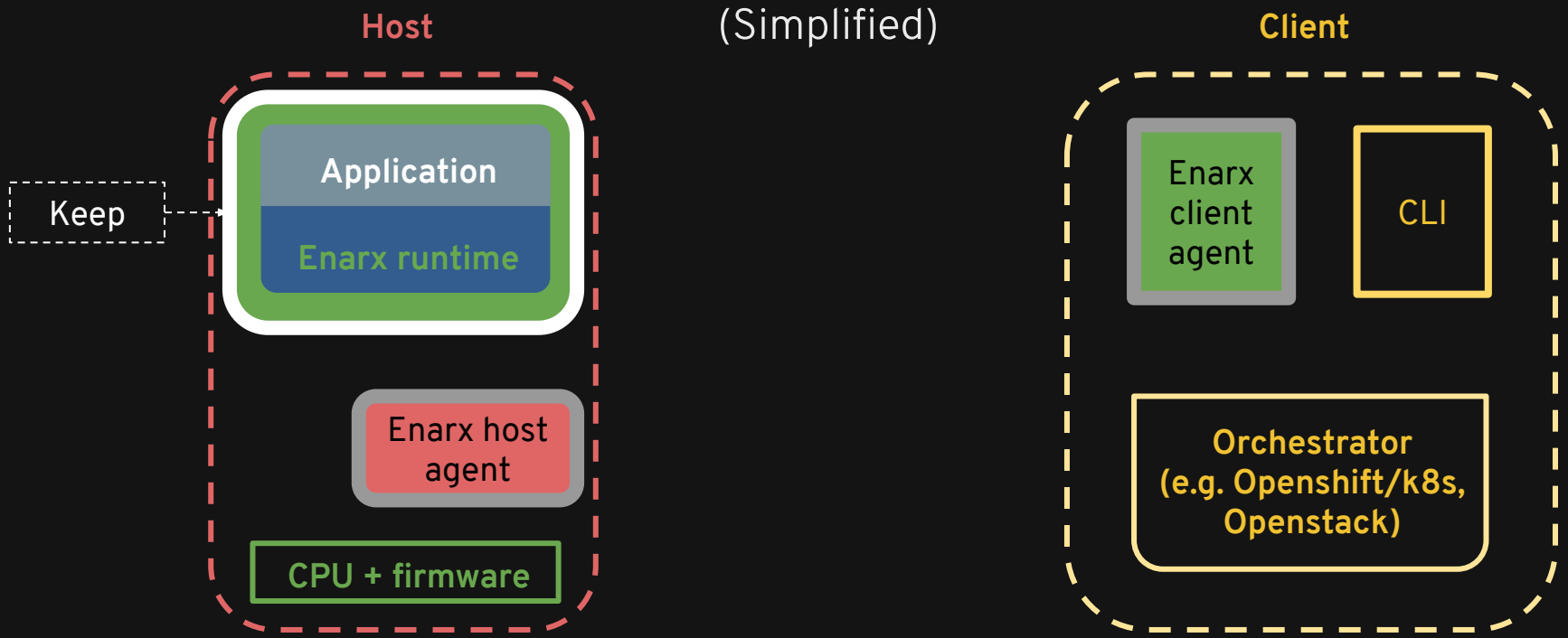
Enarx Runtime Architecture



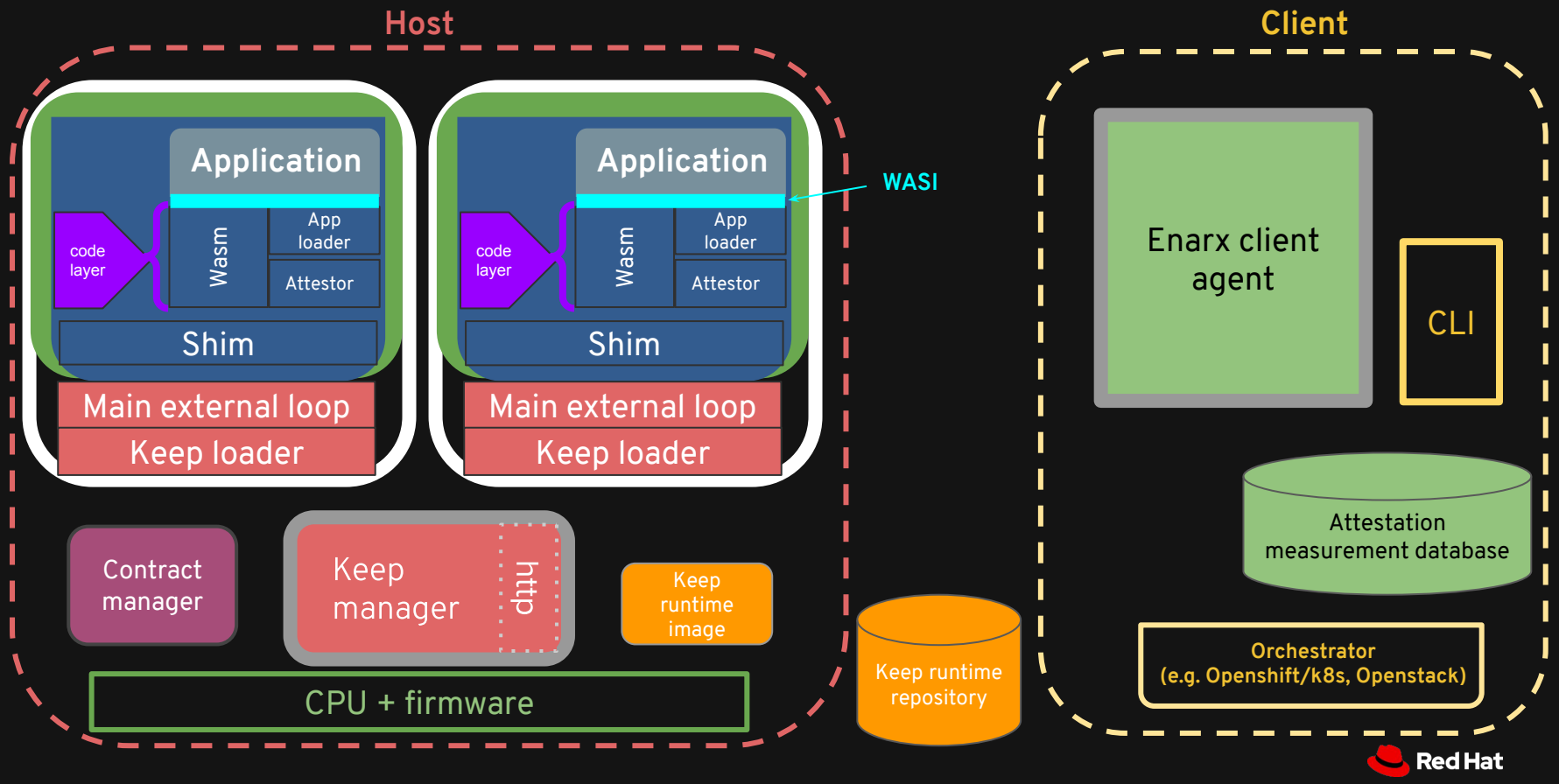
Architectural View

Enarx architectural components & integrations

(Simplified)



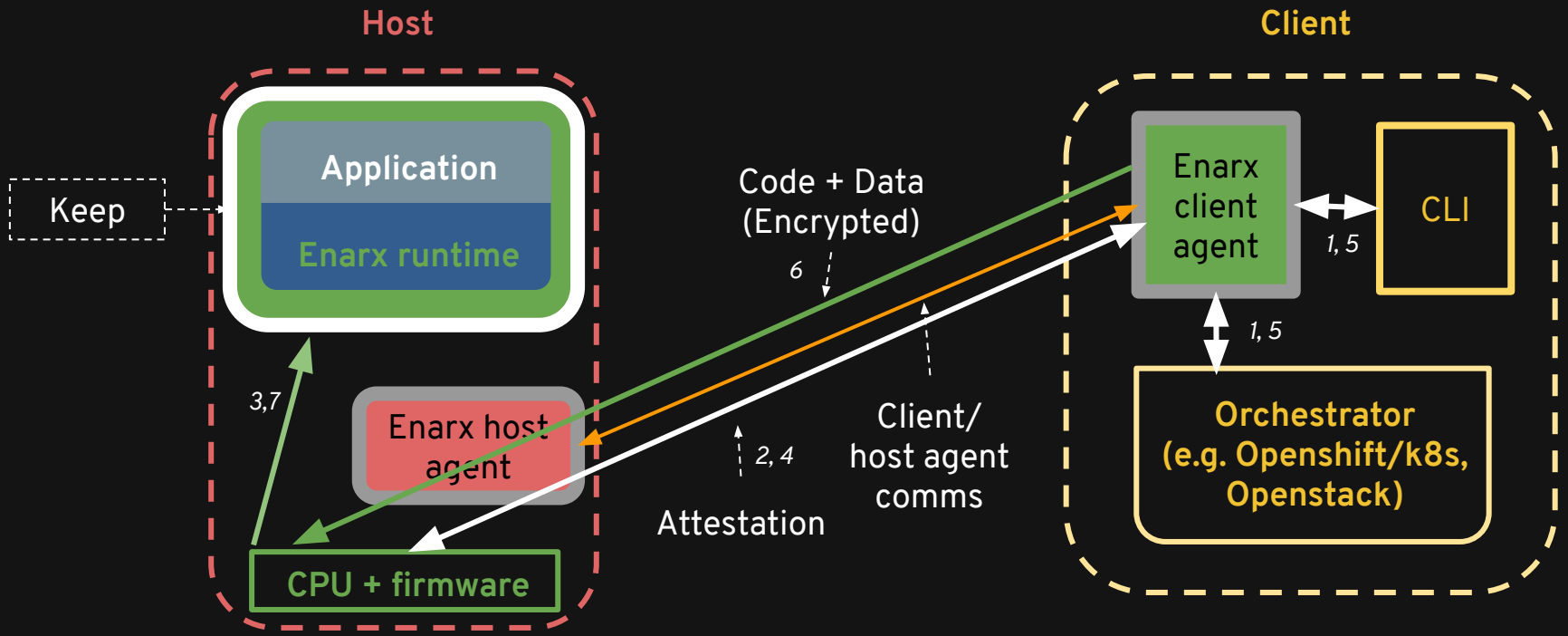
Enarx architectural components



Demo Time!

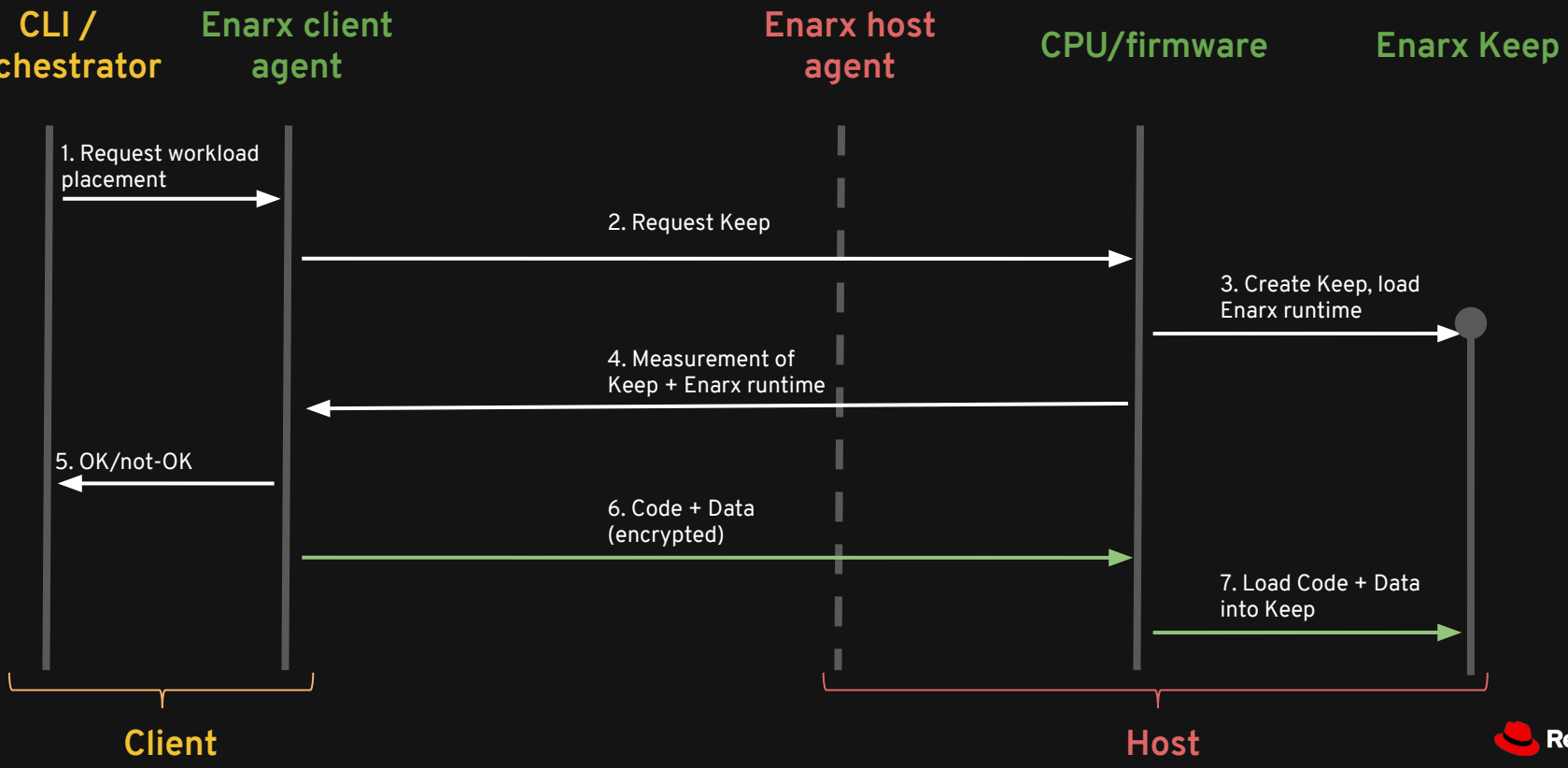
Process flow

Enarx architectural components



Red Hat

Enarx attestation process diagram



Red Hat

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

The importance of openness

Diversity of thought and experience

The importance of openness

Diversity of thought and experience

- Discover use cases
- Explore architecture
- Challenge designs
- Review implementation
- Increase test coverage
- Clarify documentation

The importance of openness

Diversity of thought and experience

Ease of involvement

The importance of openness

Diversity of thought and experience

Ease of involvement

- More voices
- More expertise
- More welcoming environment

The importance of openness

Diversity of thought and experience

Ease of involvement

Visibility of code and design

The importance of openness

Diversity of thought and experience

Ease of involvement

Visibility of code and design

- Auditability
- But only if combined with ***diversity*** and ***involvement***

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	

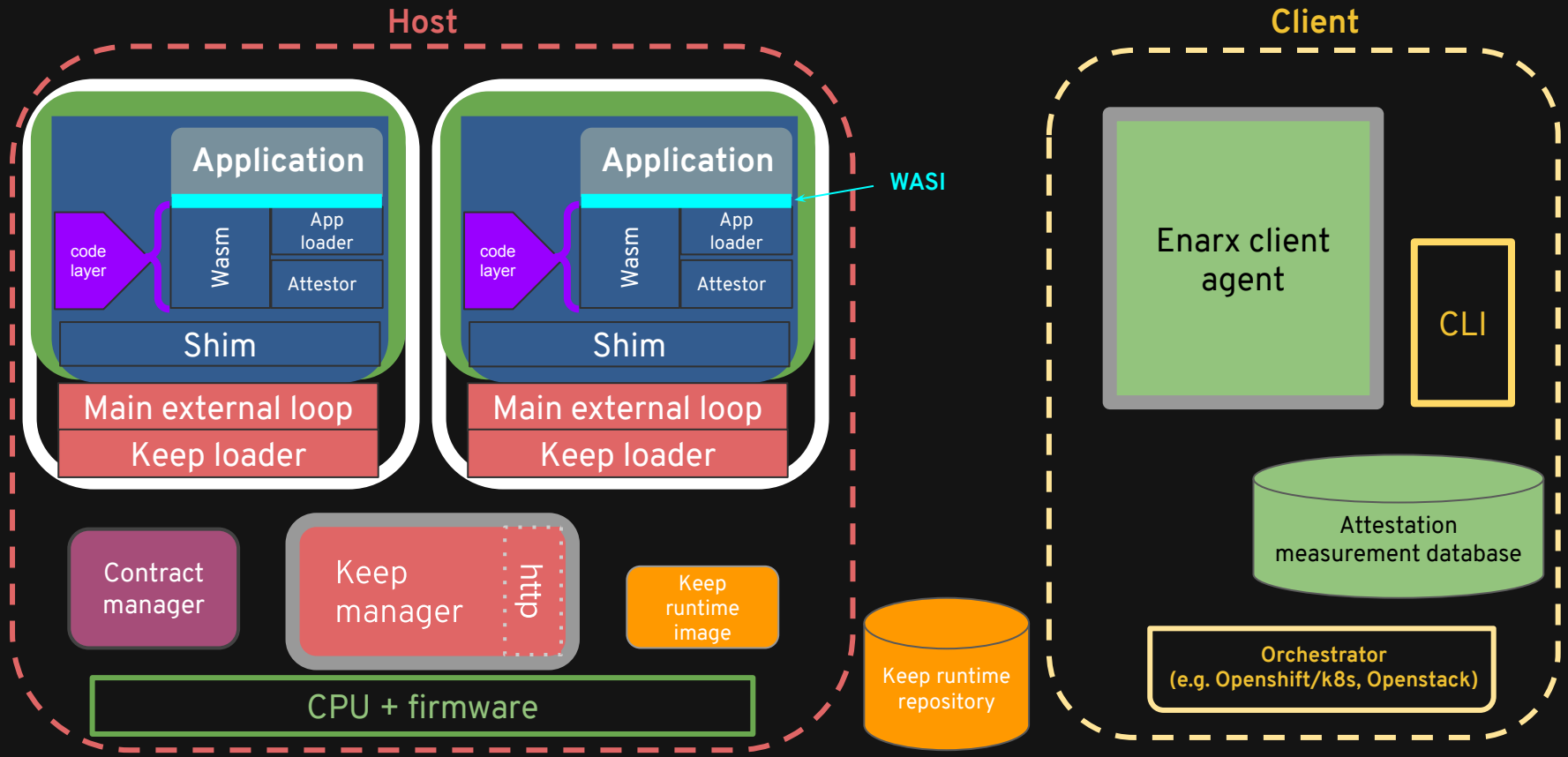
Red Hat

State of the project

State of the project

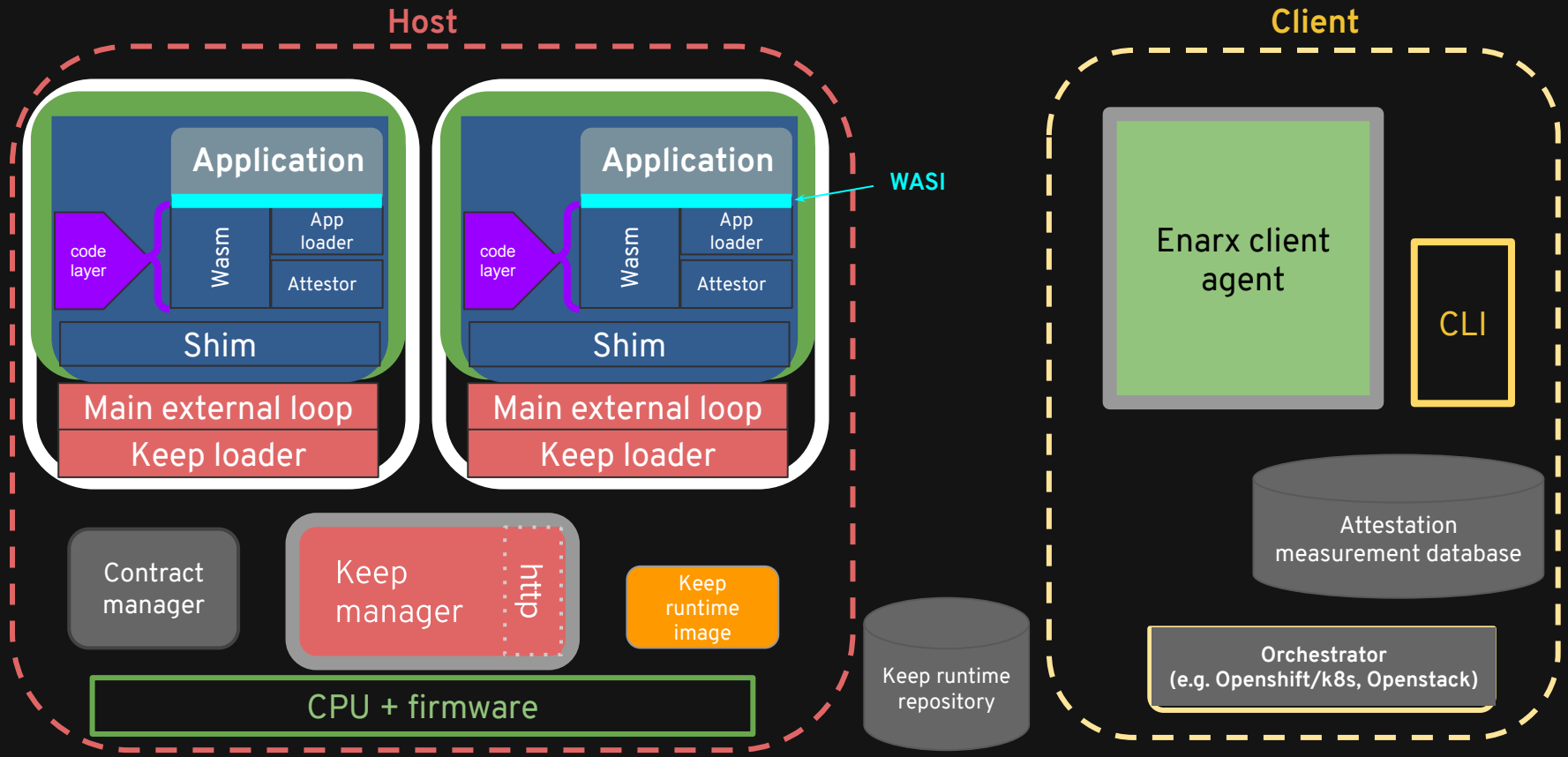
- End-to-end demo available
  - SEV (AMD)
  - SGX (Intel) - imminent!
- Stand-alone PoC framework soon
  - Early February 2021
  - Ready for WebAssembly workloads
  - Restricted networking/storage
- Documentation next
  - Initial focus on HOWTOs and design documentation

Enarx architectural components



Red Hat

Enarx architectural components



Red Hat

Key component repositories

- enarx-keepldr - Keep loading, execution and management
- enarx-wasmldr - loads and executes workloads
- enarx-keepmgr - manages multiple Keeps per host
- client - requests and attests Keeps, provisions workloads
- Infrastructure/glue
  - Platform-specifics (e.g. sev, sevctl, sgx)
  - koine - shared communications pieces
  - ciborium - CBOR-encoding/decoding crate

Note: some code awaiting merging as of 2021-01-26!

Getting involved

- Follow us on social media!
- Download, (compile,) run, test, report
  - Users / workload developers
  - Project developers
- Audit designs and implementations
- Document
- Community building and outreach

Useful technical skill-sets

Most important is willingness to learn

- SEV, SGX expertise
- WebAssembly
  - Compiler
  - WASI
- Micro-kernel/syscall
- Linux systems programming
  - Networking, storage
- Kubernetes/Openshift integration skills
- Security auditing/research
- Rust!

Where to find us

Website:	https://enarx.dev
Code:	https://github.com/enarx
Chat:	https://chat.enarx.dev
Twitter:	@enarxproject
LinkedIn:	https://www.linkedin.com/company/enarx
YouTube:	Search “Enarx”
License:	Apache 2.0
Language:	Rust (with a smattering of x86 Assembly)

Questions?

https://enarx.dev

Red Hat

Back-up slides

Component level trust

Enarx architectural components

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

Enarx client agent

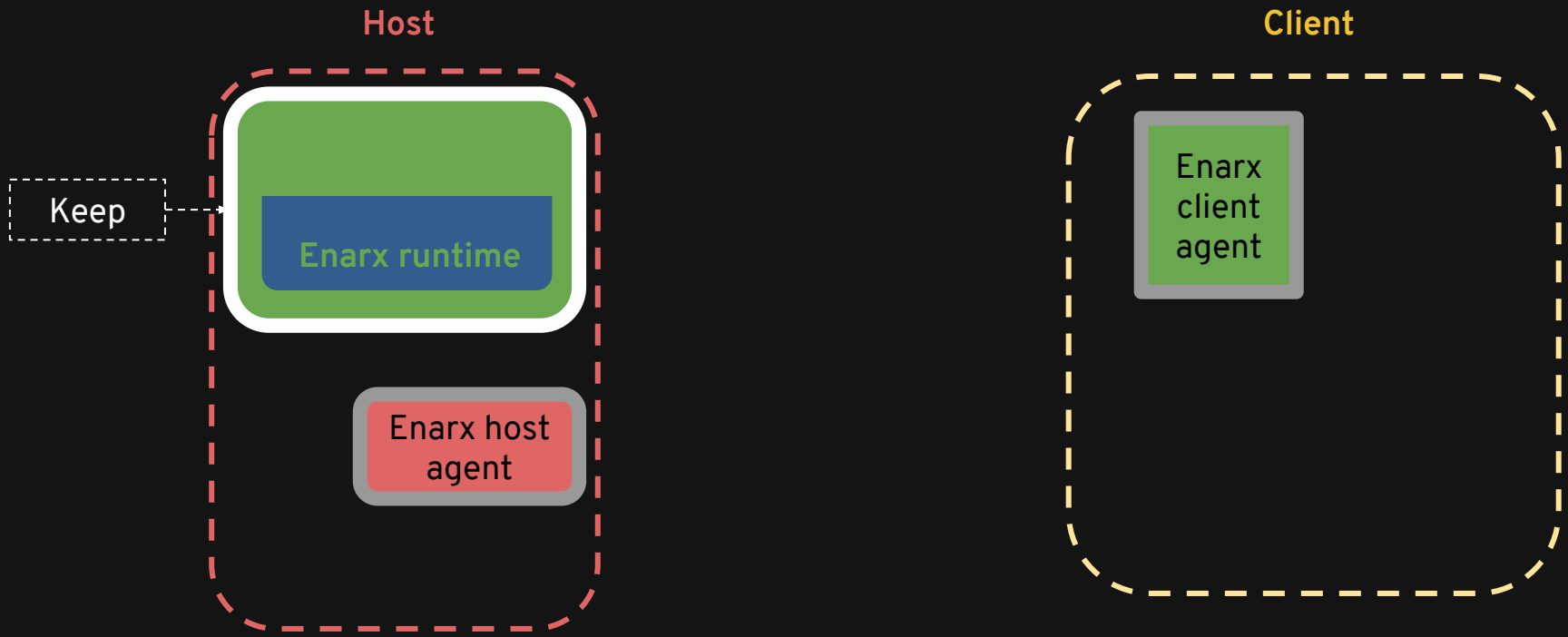
Enarx client agent - trusted
Works with orchestration/CLI
Manages attestation
Applies policy
Encrypts and transports workload

Enarx host agent

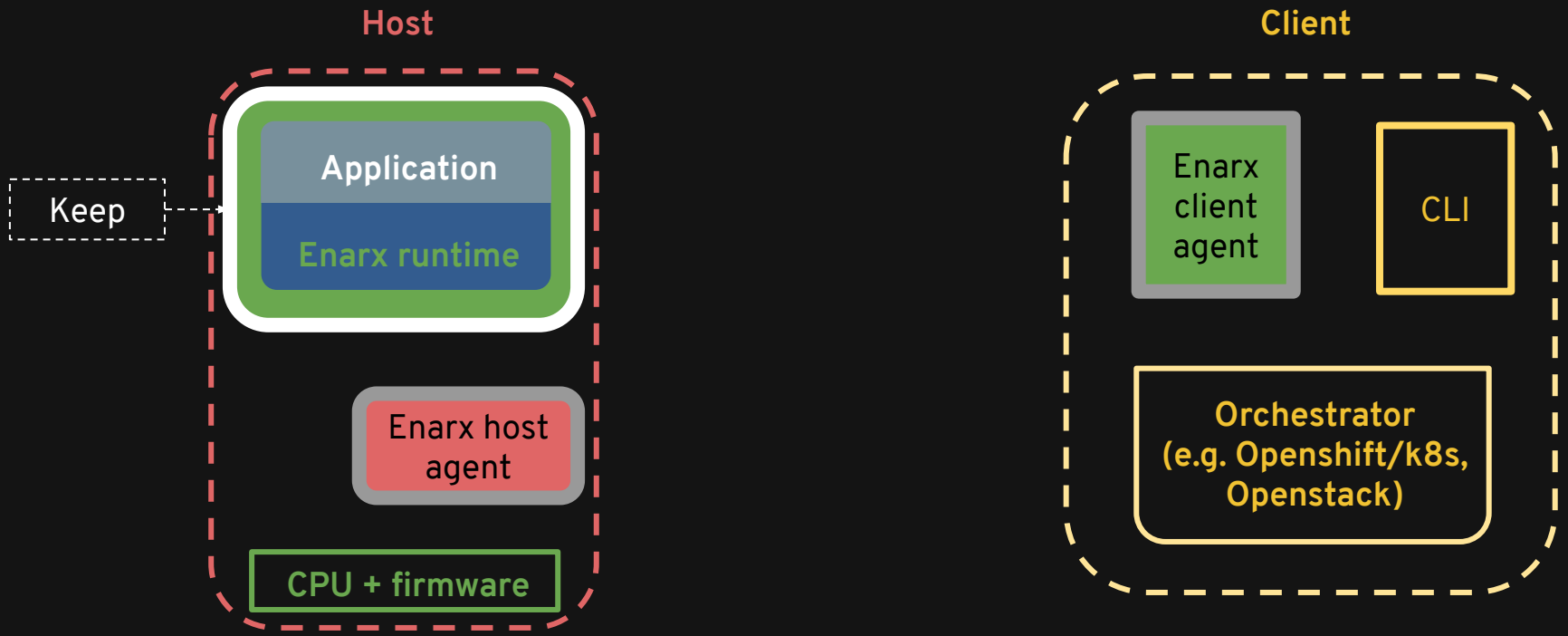
Enarx host agent - untrusted
Acts as proxy between Enarx client agent and:

- CPU/firmware
- Enarx Keep

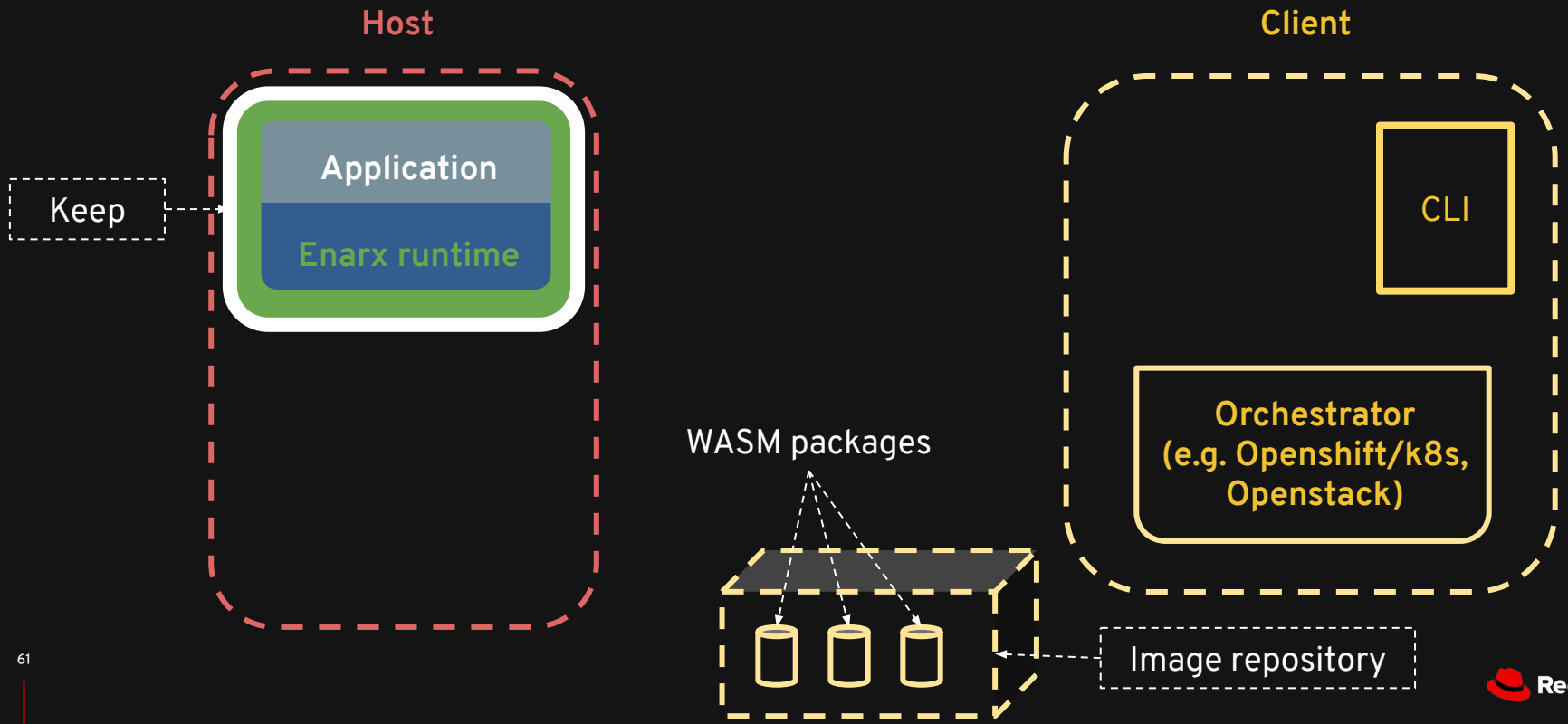
Enarx architectural components



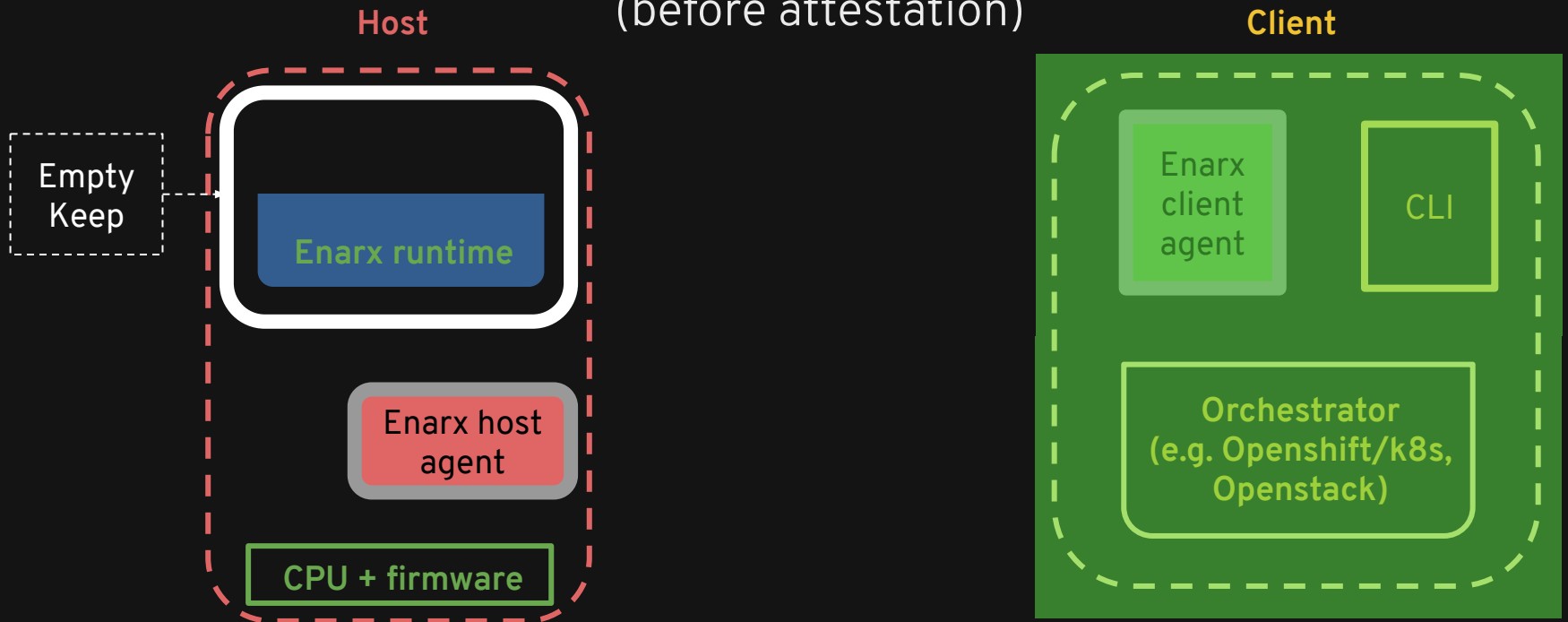
Enarx architectural components



Basic deployment architecture

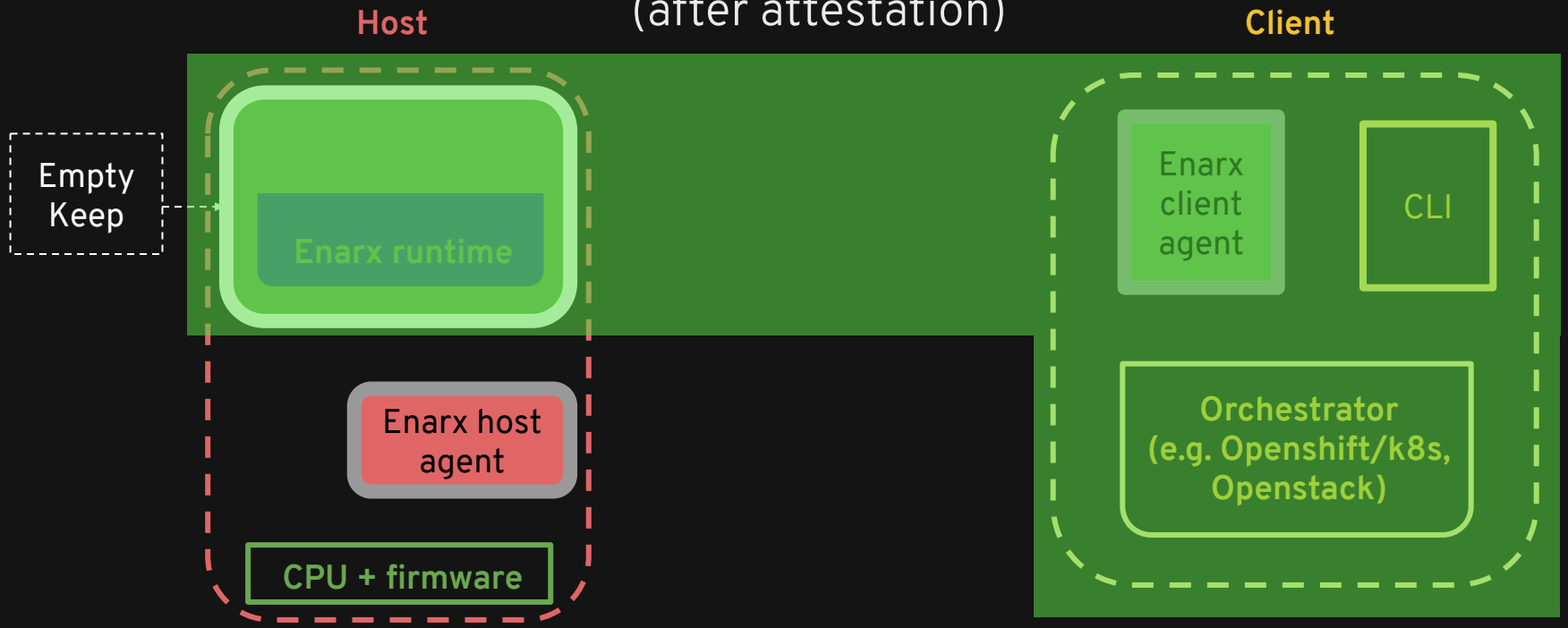


Standard Enarx trust domain (before attestation)

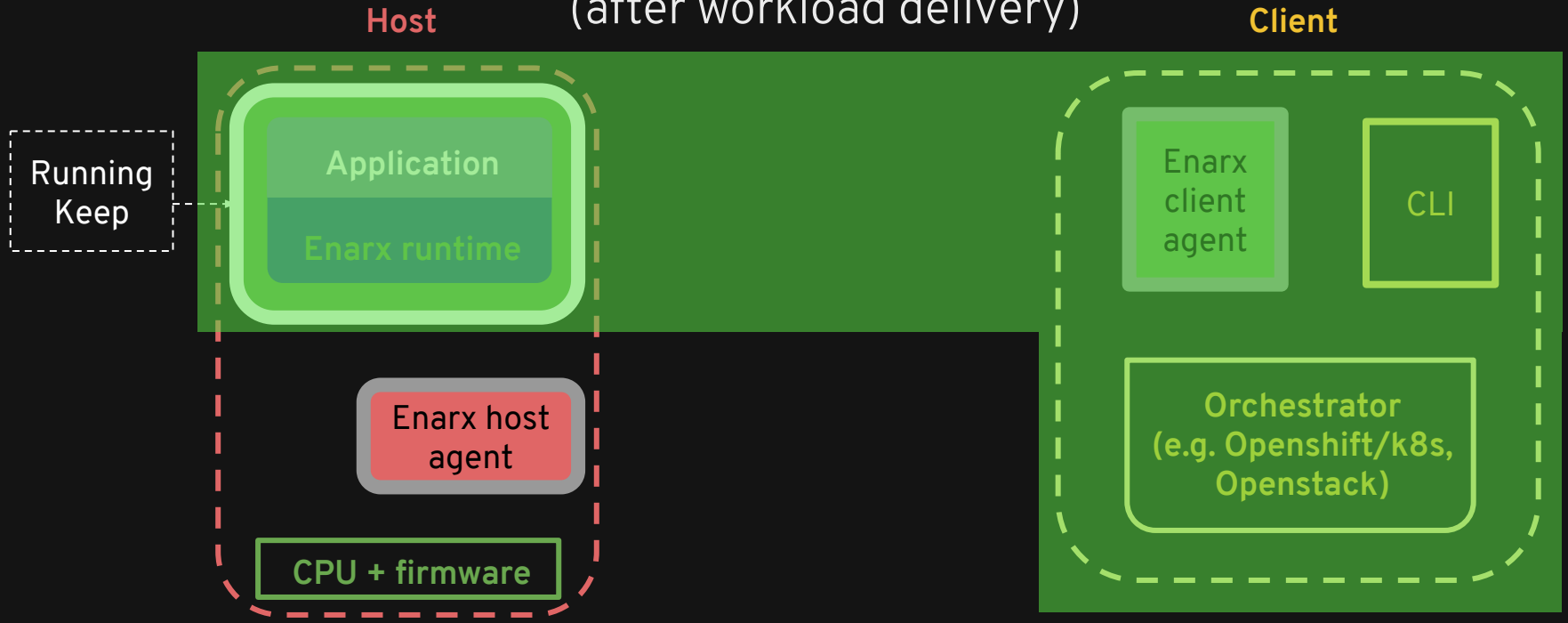


Red Hat

Standard Enarx trust domain (after attestation)



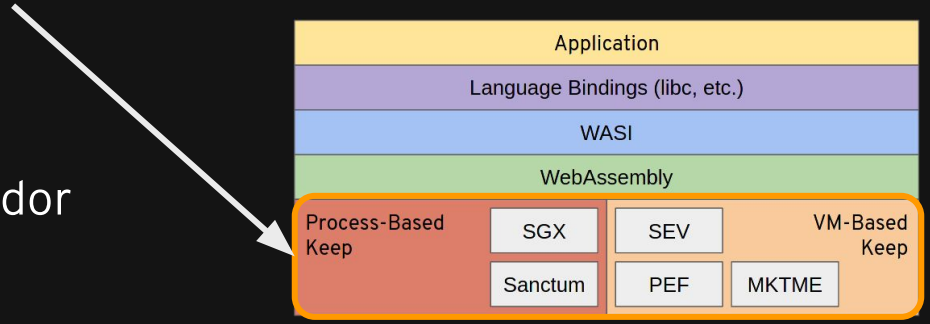
Standard Enarx trust domain (after workload delivery)



Keep layers

Keep - process or VM-based

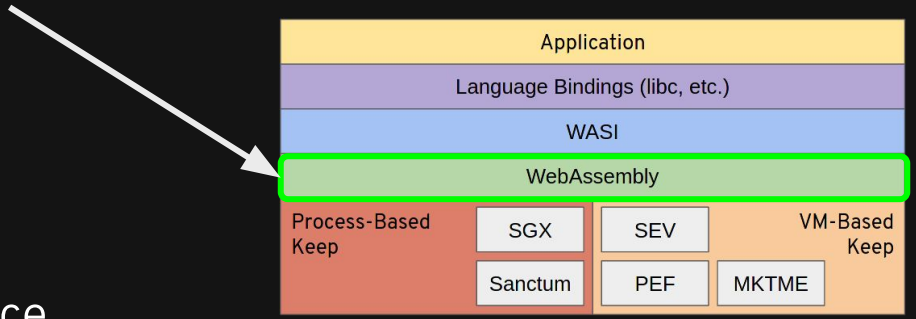
- Core Keep
- Platform-specific
  - Hardware (CPU): silicon vendor
  - Firmware: silicon vendor
  - Software: Enarx



Architecture varies between VM/Process-based platforms

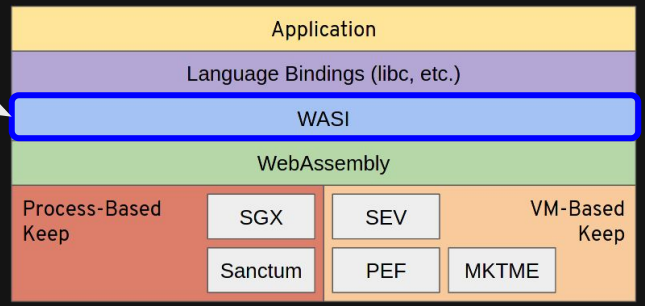
WebAssembly (WASM)

- 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)

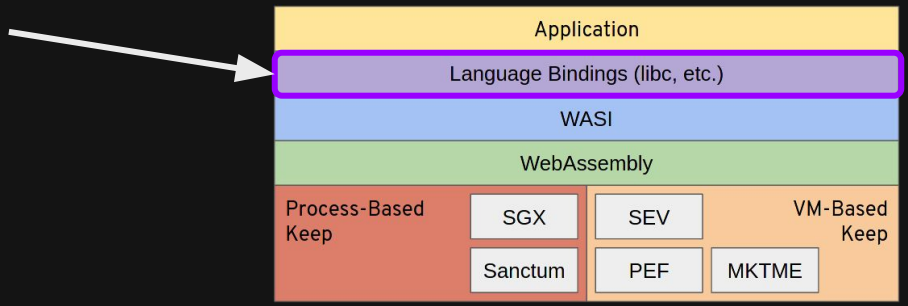
- W3C Standards Track
- Heavily inspired by a subset of POSIX
- Primary goals:
  - Portability
  - Security
- libc implementation on top
- Capability-based security:
  - No absolute resources
  - Think: openat() but not open()



Language Bindings (libc, etc.)

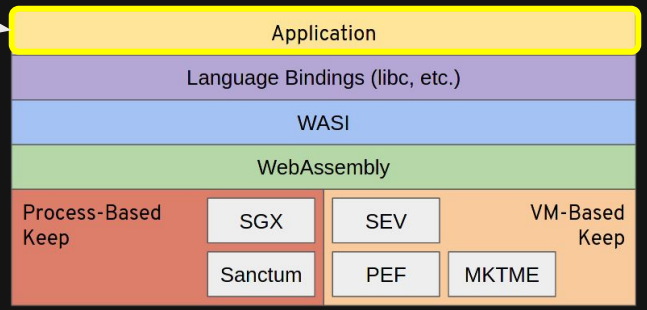
Compilation targets and includes, e.g.

- Rust: `--target wasm32-wasi`

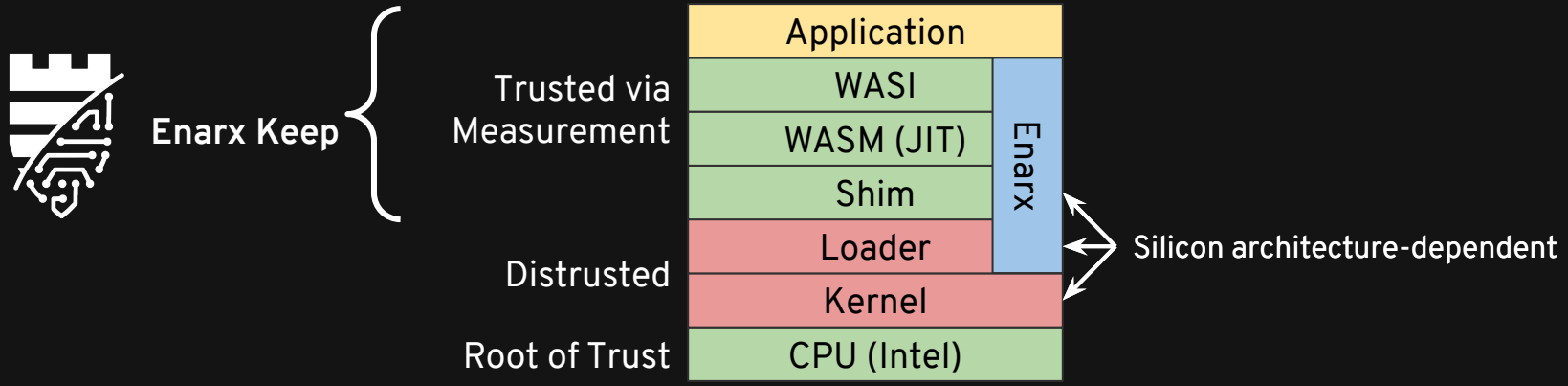


Application

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



Layers - process-based Keep



Layers - VM-based Keep