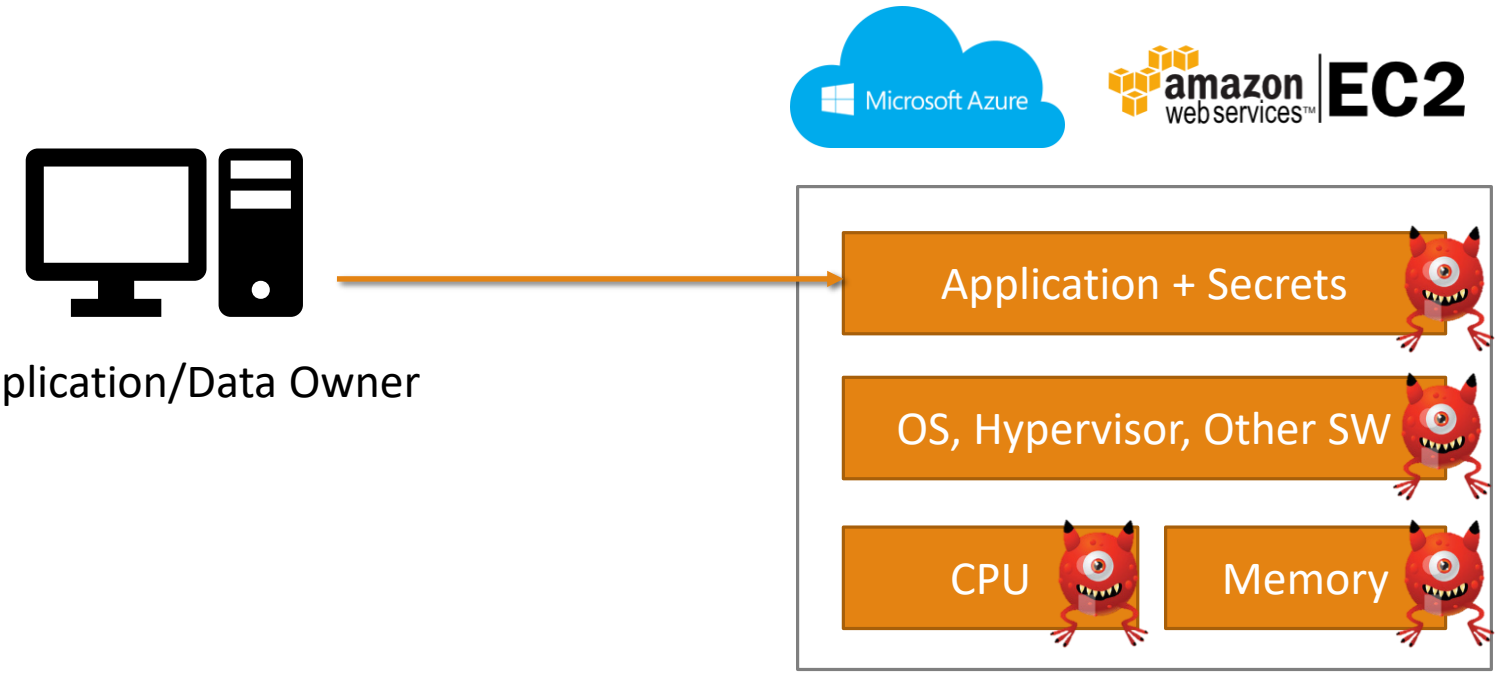
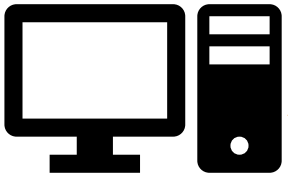
Komodo: Using Verification to Disentangle Secure-Enclave Hardware from Software

Andrew Ferraiuolo†, Andrew Baumann, Chris Hawblitzel, Bryan Parno*
Microsoft Research, Cornell University†, Carnegie Mellon University*

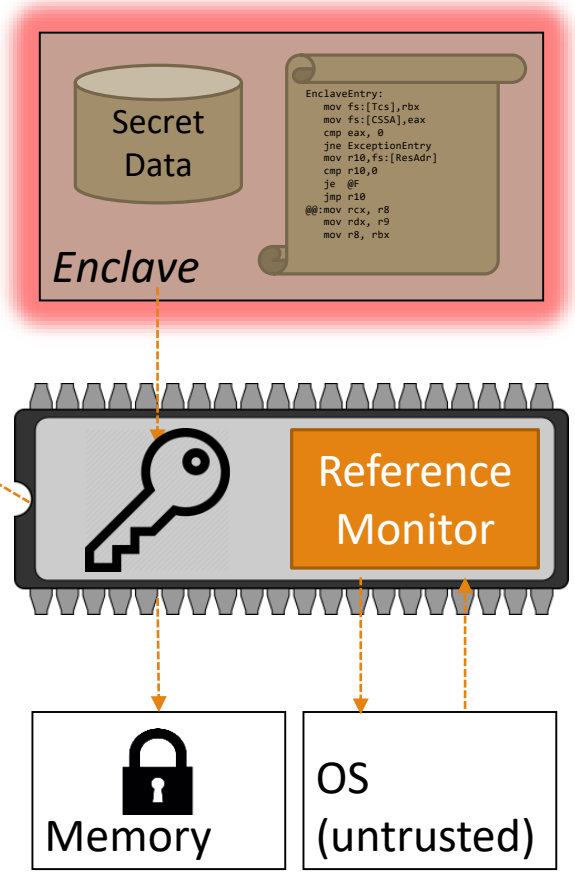
Secure Remote Computation



Intel SGX



Memory encryption
Remote attestation
SGX instructions
    Implement a *reference monitor*



SGX Limitations – Slow to Evolve

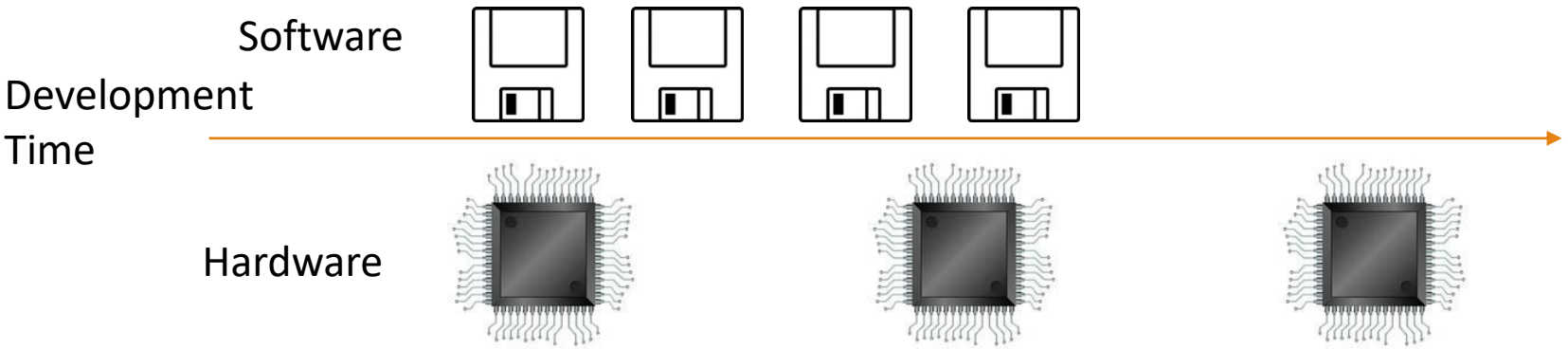
Software developers must wait for Intel to make changes

Change is necessary

- SGX1 had no support for dynamic memory management
- SGX2 was announced in 2014. Still no implementation!

SGX instructions are primarily microcode

- **Software at the slow pace of hardware!**



SGX Limitations – Root of Trust?

SGX is complex

- Approaching a microkernel in hardware

Hardware is no more trustworthy than software

- Hardware vulnerabilities: f00f, cache poisoning, VT-D vuln., others
- Purely axiomatic basis for trust

SGX vulnerabilities have already been found (CVE-2017-569)

Komodo

Enclave management in software

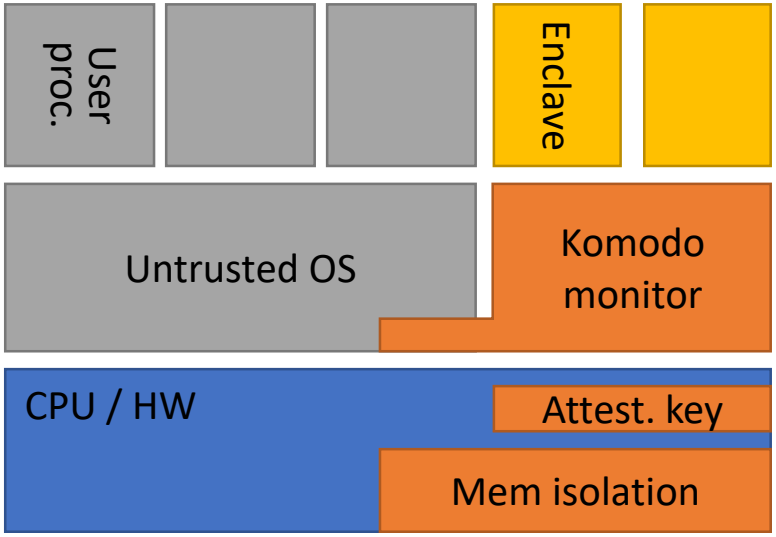
Evolve independently of hardware

Trust through formal verification

Komodo Architecture

Komodo monitor software:

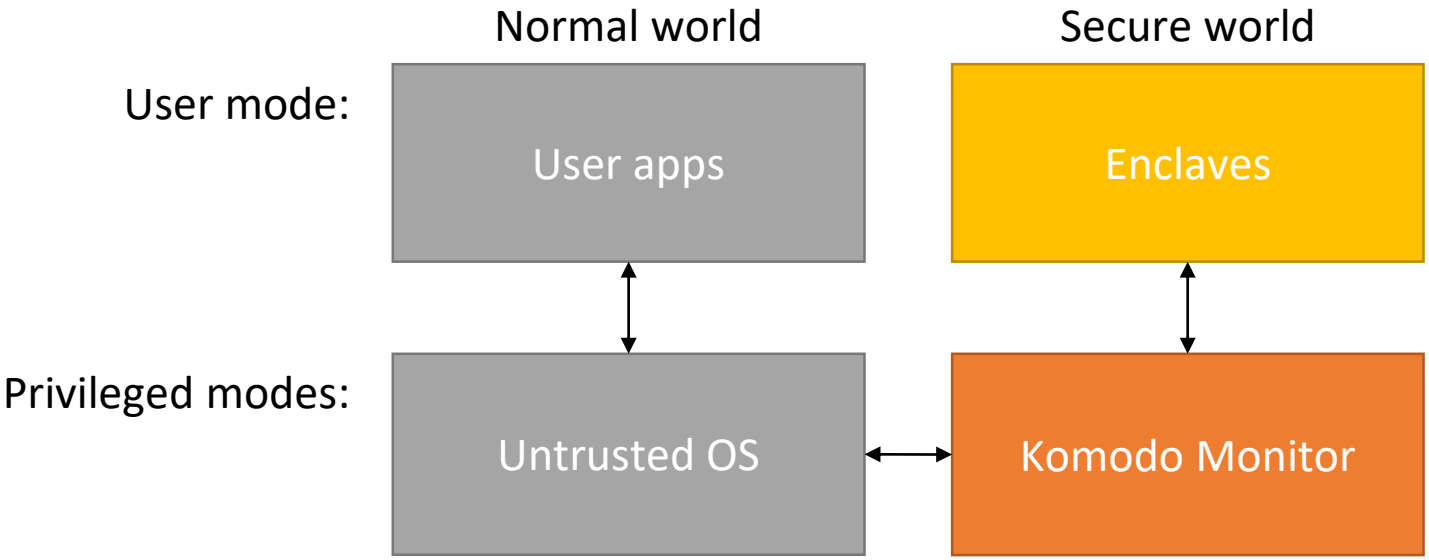
- Mimics SGX instructions
- Minimal hardware requirements
  - Supported by commercial processors



Hardware Requirements:

- Isolated memory
  - Encryption (Intel/AMD), partitioning (ARM)
- Key-generation for attestation
  - Trusted Platform Module (many processors)
- Protection modes for enclave, monitor
  - Machine mode (RISC-V), TrustZone (ARM)

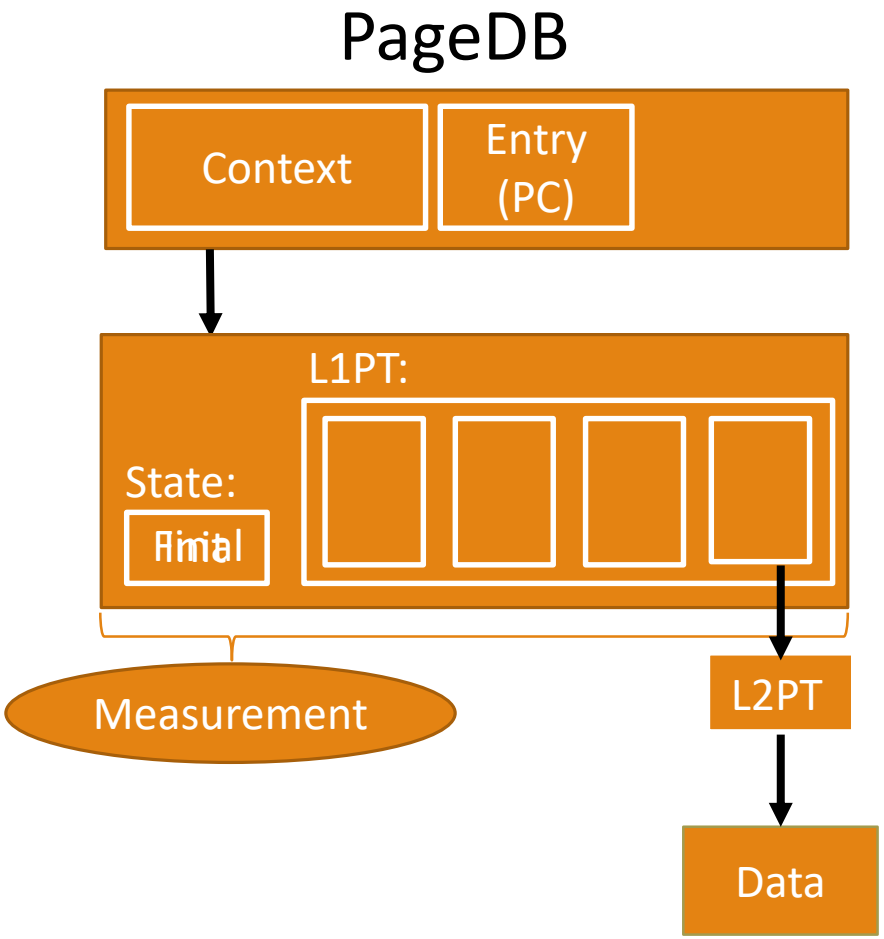
Prototype on ARM TrustZone



Secure-world memory is isolated from normal world.

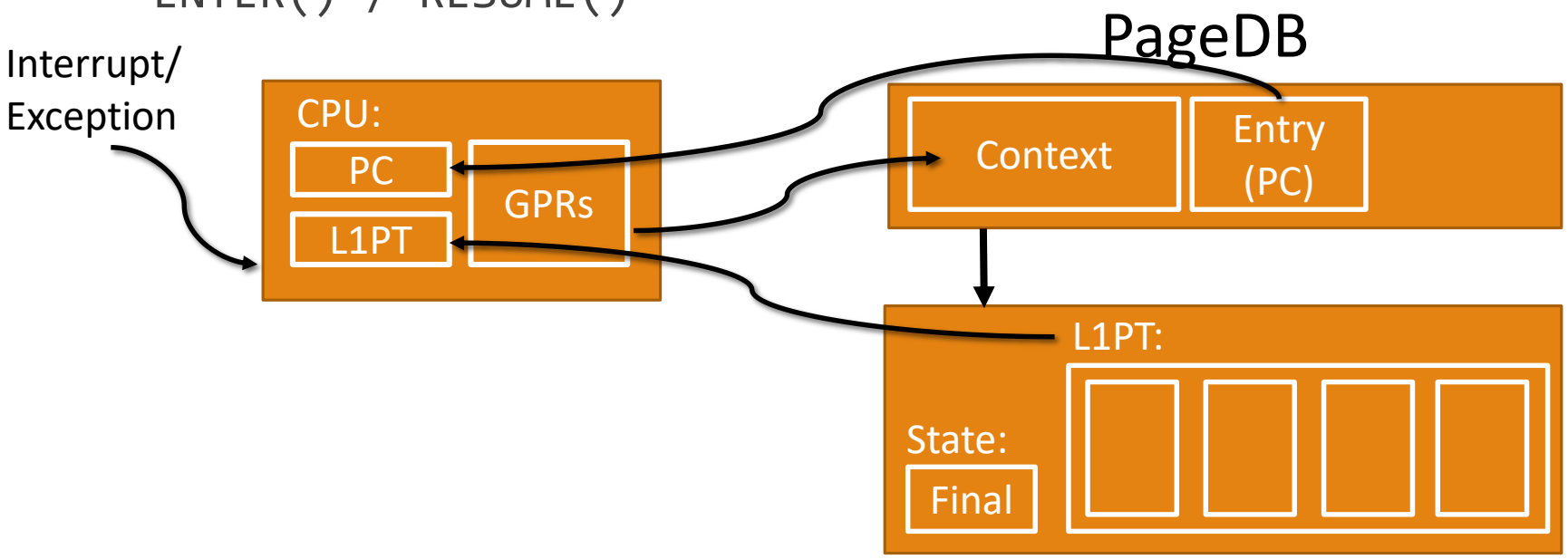
OS Monitor Calls: Creation

```
INIT_ADDRSPACE()

INIT_L2PT()

MAP_SECURE() /
     MAP_INSECURE()

INIT_THREAD()

FINALISE()
```



OS Monitor Calls: Entry

`ENTER() / RESUME()`



Enclave Execution

Compute on data in its secure pages

Communicate with outside world

- Read/write insecure pages
- Register arguments/return values

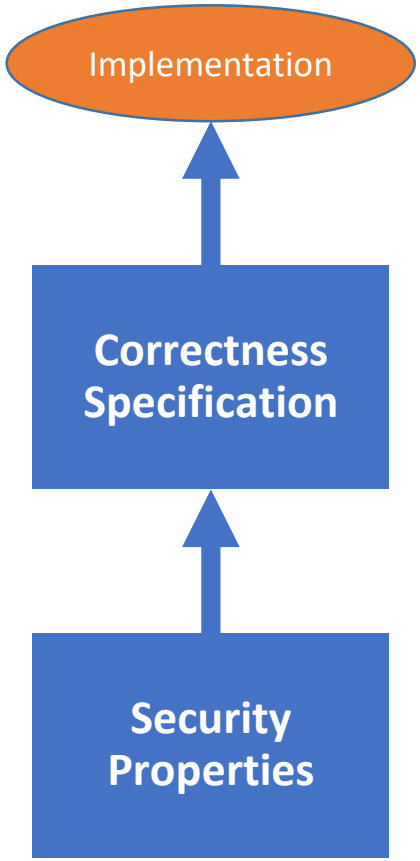
Komodo enclave API

- Create/verify attestations
- Secure source of randomness
- Map/unmap spare pages
- Exit thread

Verification

1) Prove Komodo conforms to specification of correct execution
   - Simpler, more abstract

2) Prove that correctness spec enforces security properties



Security Properties

Enclaves are protected from an OS + malicious enclave adversary:

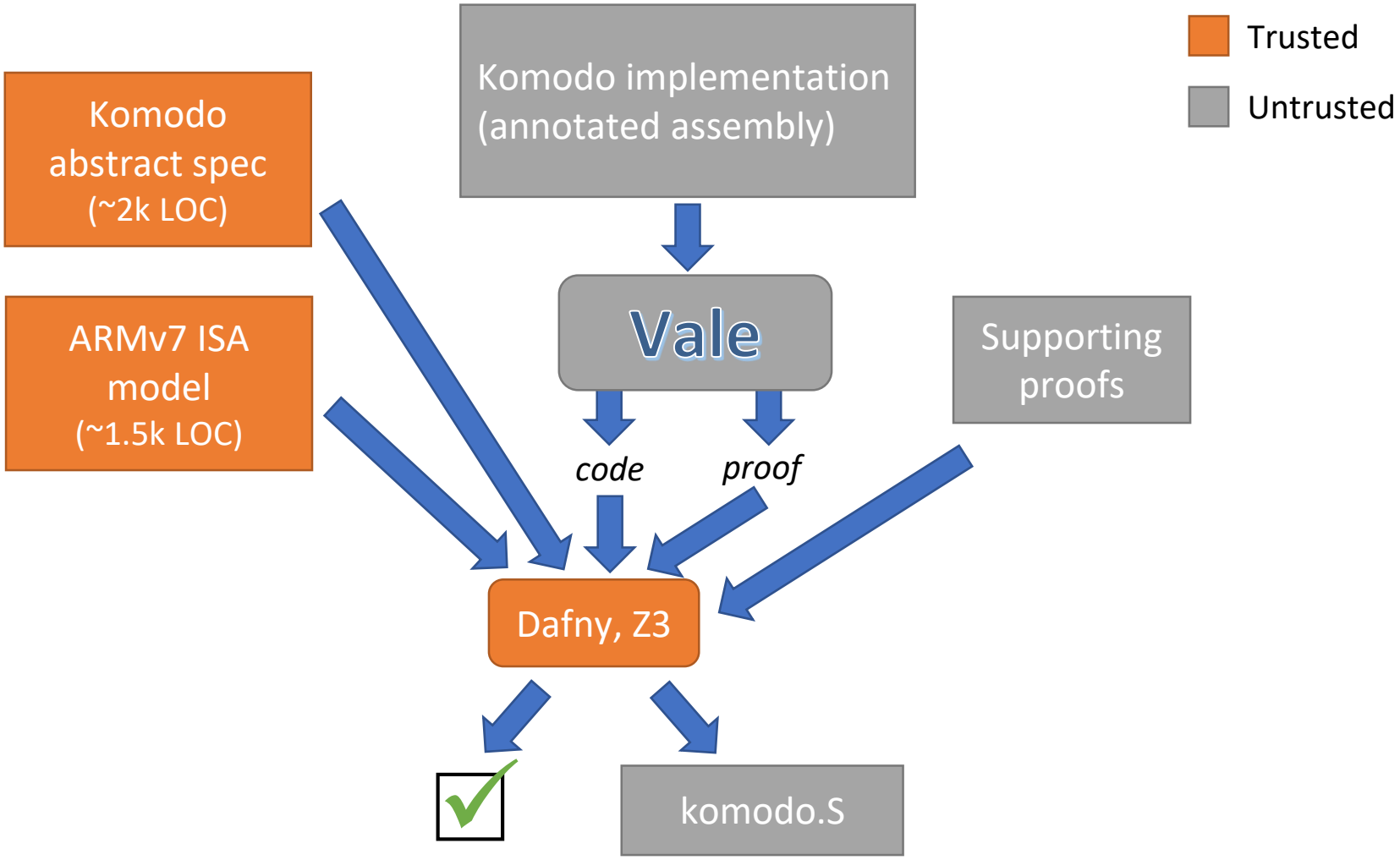
- Confidentiality – enclave secret state cannot leak to adversary
- Integrity – adversary cannot tamper with enclave trusted contents

Formalized as *noninterference* – adversarially-observable outputs are purely determined by adversarially-controlled inputs

Declassified to OS: exception type, dynamic allocation, return values, and insecure memory

- Precisely captures what information is released

Verification Approach



Microbenchmark results

Operation	Cycles
Null SMC	123
Enter	496
Resume	625
Enter + Exit	**738**

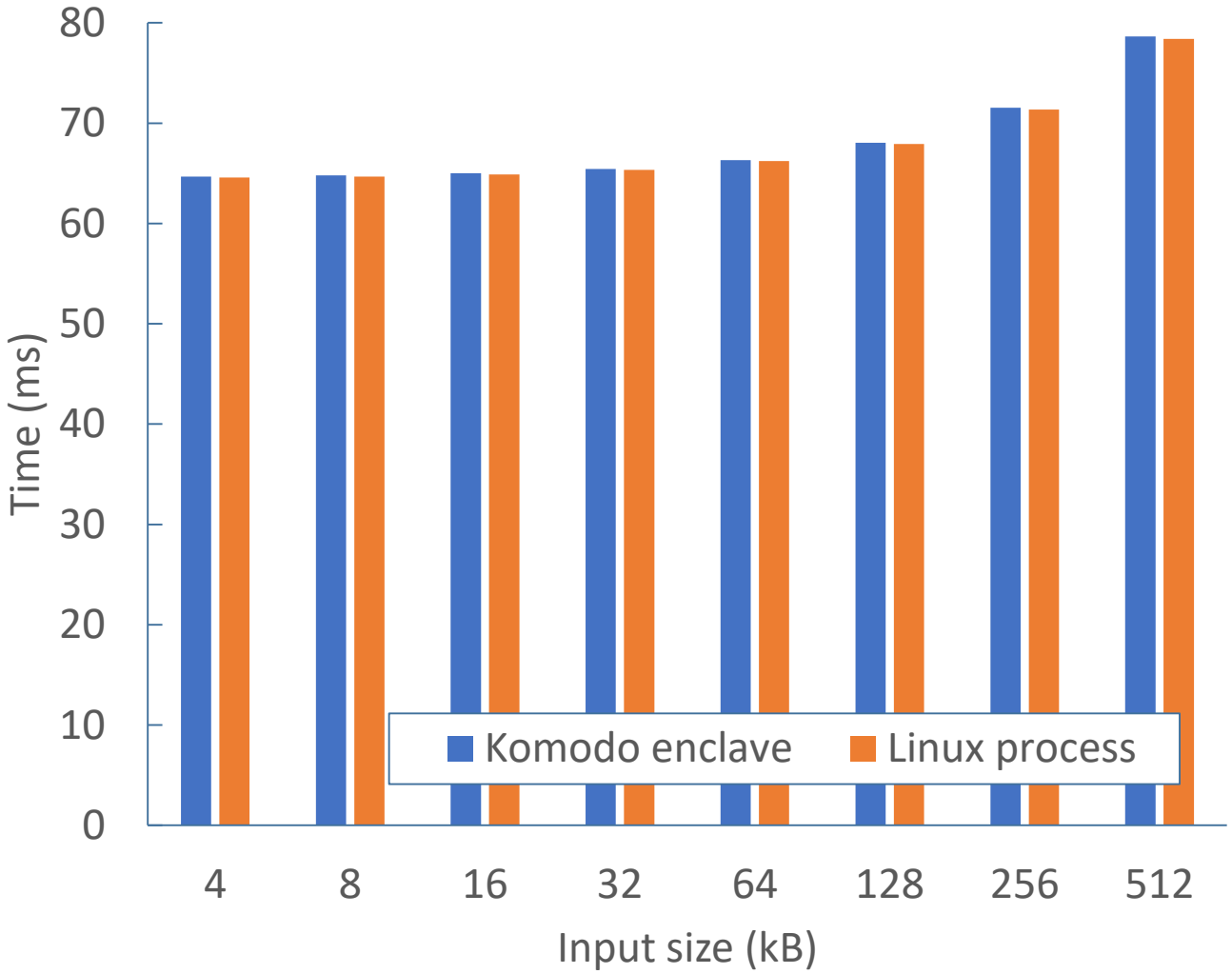
Prototype on Raspberry Pi 2.

- Bootloader: loads monitor into secure world memory + sets exception vectors

cf. SGX: ~7100 cycles for enter + exit [Eleos, Eurosys'17]

- In part because RasPi has a slower clock rate (900MHz vs 2GHz+)

Performance: Notary Application



Verification Effort

Total verification effort – 2 person-years

Source lines of code :

	Spec	Impl	Proof
Total	4,446	2,710	18,655

Security	**175**
Correctness	795
ARM	1,174
Other	2,302

Adaptability

Motivation: software can evolve more quickly than hardware

SGX2 extends SGX1 with dynamic memory management

- Specified **three years ago**. Still no implementation

We extended Komodo with dynamic memory in 6 person-months!

- Three weeks to re-establish security proofs

Related Work

CertiKOS / seL4

- Implement fully-featured microkernels
- Prove correctness, security properties
- Komodo is a simpler system, supports attestation

Sanctum

- Proposes RISC-V-based hardware that meets the needs of Komodo

Lessons Learned

A small code base is not a substitute for verification.

- Verification caught real bugs in our implementation

Trusted components require extra diligence

- We found bugs in trusted/unverified components

Verification tools can still improve

- Timeouts / proof instability

Conclusion

SGX defends against a powerful threat-model, but it has limitations:

- Slow to change
- Requires axiomatic trust

Komodo improves evolvability and security

- Implemented in software with minimal hardware requirements
- First formally-verified implementation of attested enclaves

Verification of software enclaves is tractable, permits evolution

- 2 person-years worth of total effort
- 6 person-months to add SGX2-like dynamic memory management

https://github.com/Microsoft/Komodo