Security Engineering

Modern Operating Systems 2: Cloud security, sandboxing, virtualization and containers.

Today

- Last time: Mostly about access control
- Today: Mostly about isolation

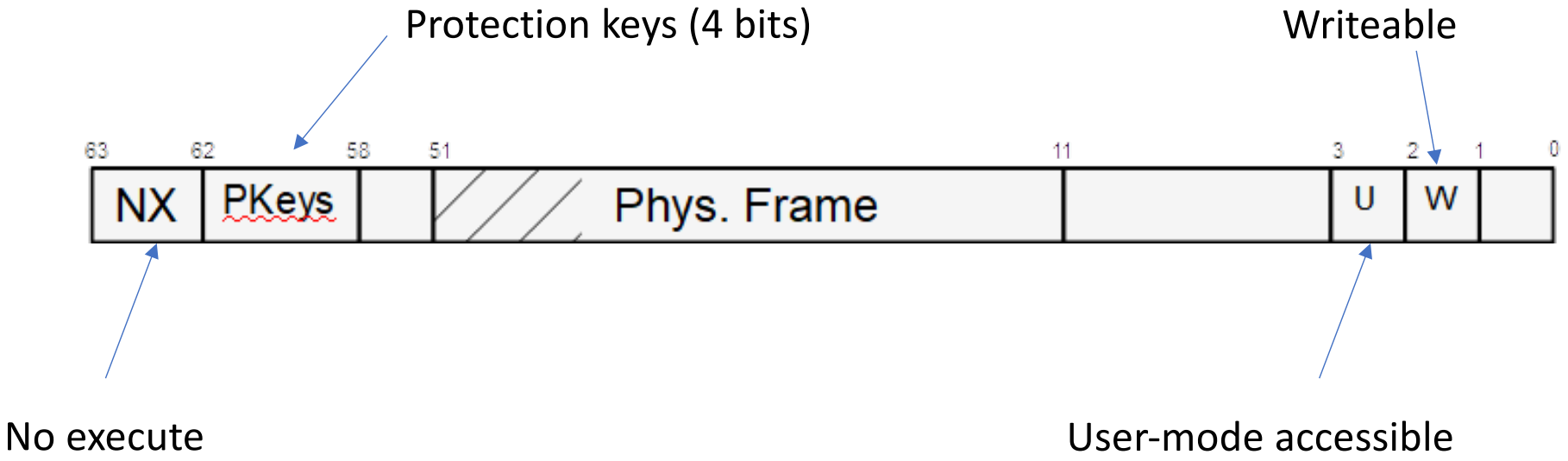
Challenges

- How do you stop others, using the same system(s), from being able to read your data / hack your software?
- I mean, really stop them (side channels, bugs in TCB)?
- How do you know a cloud provider is even running the software you've asked it to, without any tampering (remote attestation)?

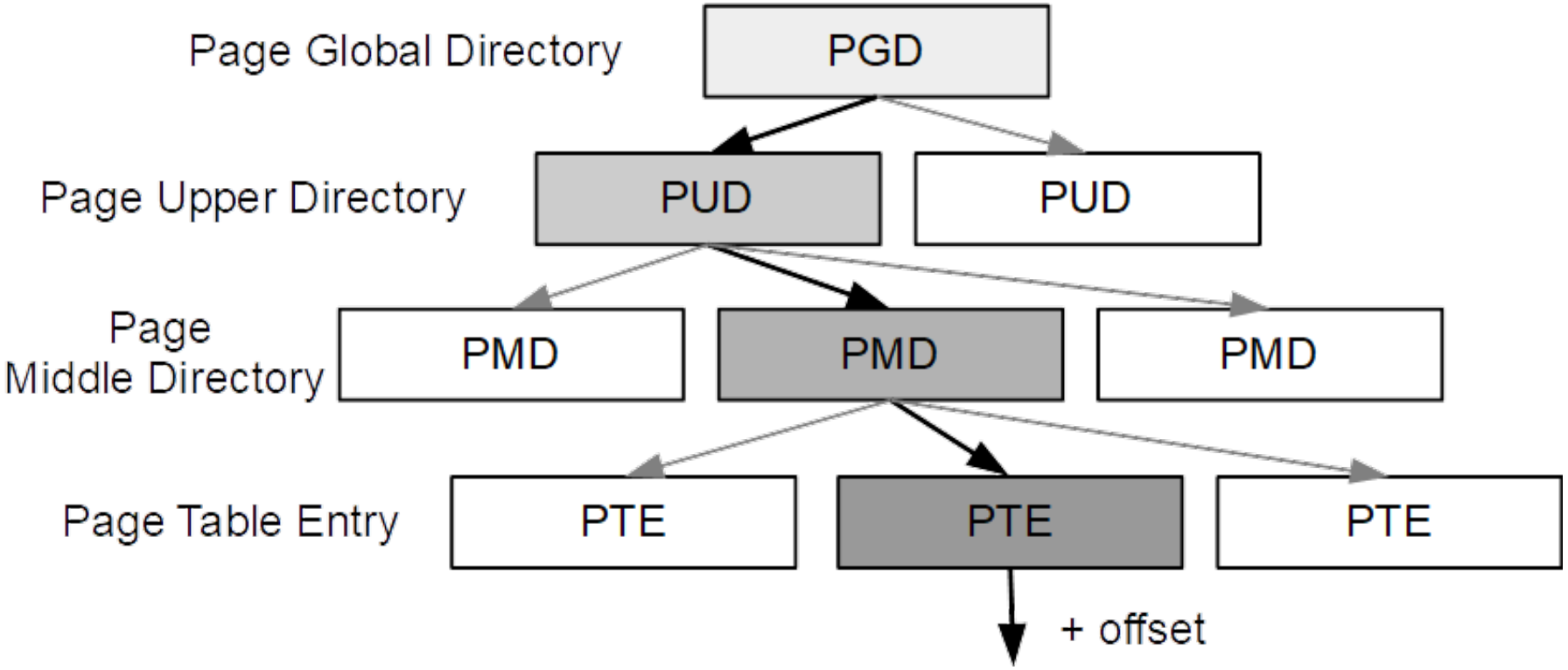
Cloud Security

- Access control often specific to provider e.g. AWS Identity and Access Management – lock in
- Google App Engine: Provide only indirect access to networking/file system, by "safe" versions that integrate MAC and call other cloud infrastructure.
- Defence in depth: Integrate sanitization, mitigation/hardening, functionality reduction, indirection (Python -> NaCL) and ptrace filtering and logging.
- Why would you pay Amazon to run your own local servers? https://aws.amazon.com/blogs/containers/introducing-amazon-ecs-anywhere/

Isolation: Processes and Memory Management



Page-Table Walks



Operating-System Page-Table Protection Features

- Data Execution Prevention (NX XOR W) – forces attackers to use Return-Oriented Programming.
- Address Space Layout Randomisation

Isolation: Sandboxing, Virtualization and Containers

- Containers e.g. Docker: between-process with filtering
- Virtualization e.g. VirtualBox: whole separate “guest” OS on top of a shared “host” OS
- Sandboxing e.g. Chrome, eBPF: often within-process

Isolation: Sandboxing

- Pioneered by Sun with the JVM
- Restrict the environment in various ways: temporary access to a single directory, communication via same-origin policy
- Prevent access of address space outside of a predefined region, e.g. the Javascript interpreter's memory in a browser (Spectre klaxon).

Sandboxing and Isolation in Chrome

- Cross-website theft increasingly important, so Site Isolation not just Renderer Isolation.
- Increasingly done by process-level isolation: also useful for Spectre.
- The “same-origin policy” gets complicated when you have untrusted ads in the same tabs…
- Vulnerabilities often exploited by Drive-by-Download attacks.

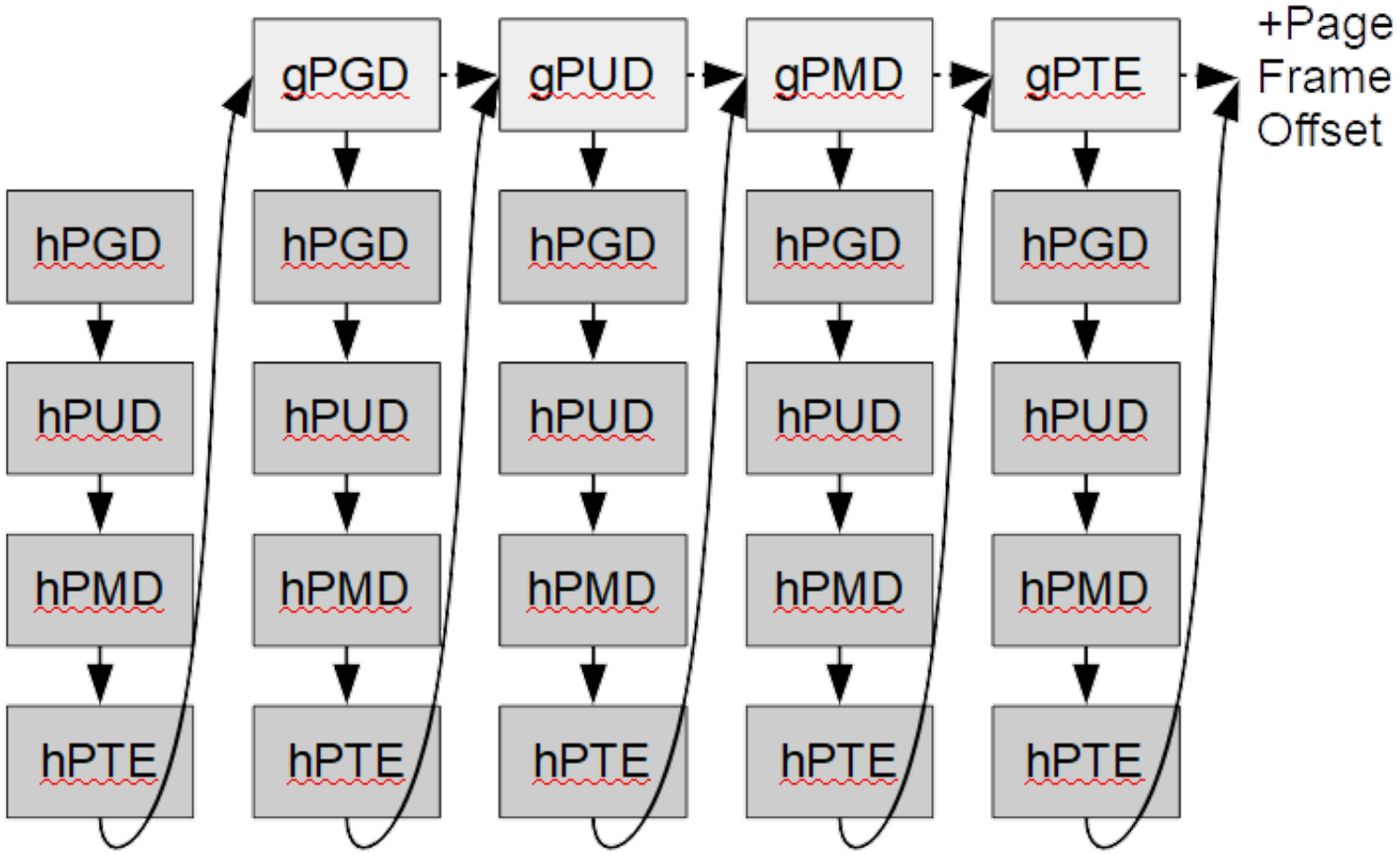
Sandboxing in eBPF

- Extended Berkeley Packet Filter: Run sandboxed user code inside the kernel!
- Typically used for network packet filtering and security analysis
- Turing incomplete, so guaranteed to terminate, and no unreachable code allowed.
- Optional pointer arithmetic prevention; guaranteed bounds checking (modulo Spectre).
- Kernel-API filtering based on type (e.g. network only)

Isolation: Virtualization

- Replaces the entire operating system, and runs a “guest” operating system on top of a “host” via hypervisor.
- Powers cloud computing.
- HW support such as Intel VT-x makes things cleaner and faster.
- Why more secure? The hypervisor can be much smaller than a full OS and so easier to code-review and secure, right???

Page-Table Walks in Virtualized Systems



Virtualization

- Can have different guests and hosts (e.g. Windows on OSX)
- Get flexibility and containment
- Samsung Knox: get a VM for an employer to lock down and manage remotely, next to normal Android on the same device.

Virtualization: Challenges

- Subtle issues around monitoring: If you're going to check all ACLs on your server, what about the containers or virtualized systems?
- Trouble at the interface: people still need to share data between VMs and ad-hoc mechanisms such as USB devices
- Bromium: VM per app, messy at the interface with untrusted files sent via host. Need specific exceptions and plugins, like Outlook being prevented from rendering files itself.

Isolation: Containers

- Cheaper than a VM, but less secure. E.g. Docker
- Shared kernel, but isolates application code and libraries from the rest of the system -- not shared anymore.
- NSjail: trap processes in a "jail": restrict syscalls by seccomp, change root to a local directory.
- Virtualise some parts but not others e.g. PIDs, IPC and namespaces.
- Syscall filtering too, and sandboxing.

Containers: Issues

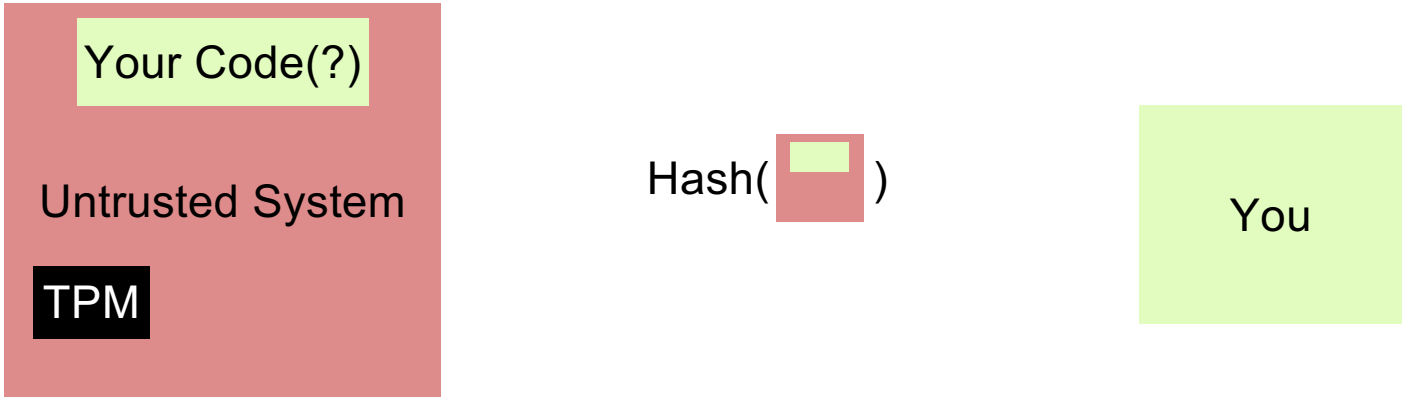
- Not the same as VM, and not really meant for data isolation -- don't expect the same isolation as with a VM -- the trusted code base is still massive.
- Really for deployability, not security. Many subtle bugs, such as blank root passwords as defaults!
- On the flipside, deployability might *be* a security feature – why?

Google

- SRS: "*Google compartmentalizes by **role, location, and time**. When an attacker tries to compromise a compartmentalized system, the potential scope of any single attack is greatly reduced. If the system is compromised, the incident management teams have options to disable only parts of it to purge the effects of the compromise while leaving other parts operational."*
- For microservices, this means run jobs that don't need access to the same things as **different accounts**, means **splitting across multiple data centres**, and **rotating keys over time** to limit scope and force attacker to maintain presence.

Trust: TPMs and Enclaves

Remote Attestation



Remote Attestation

Remote Attestation

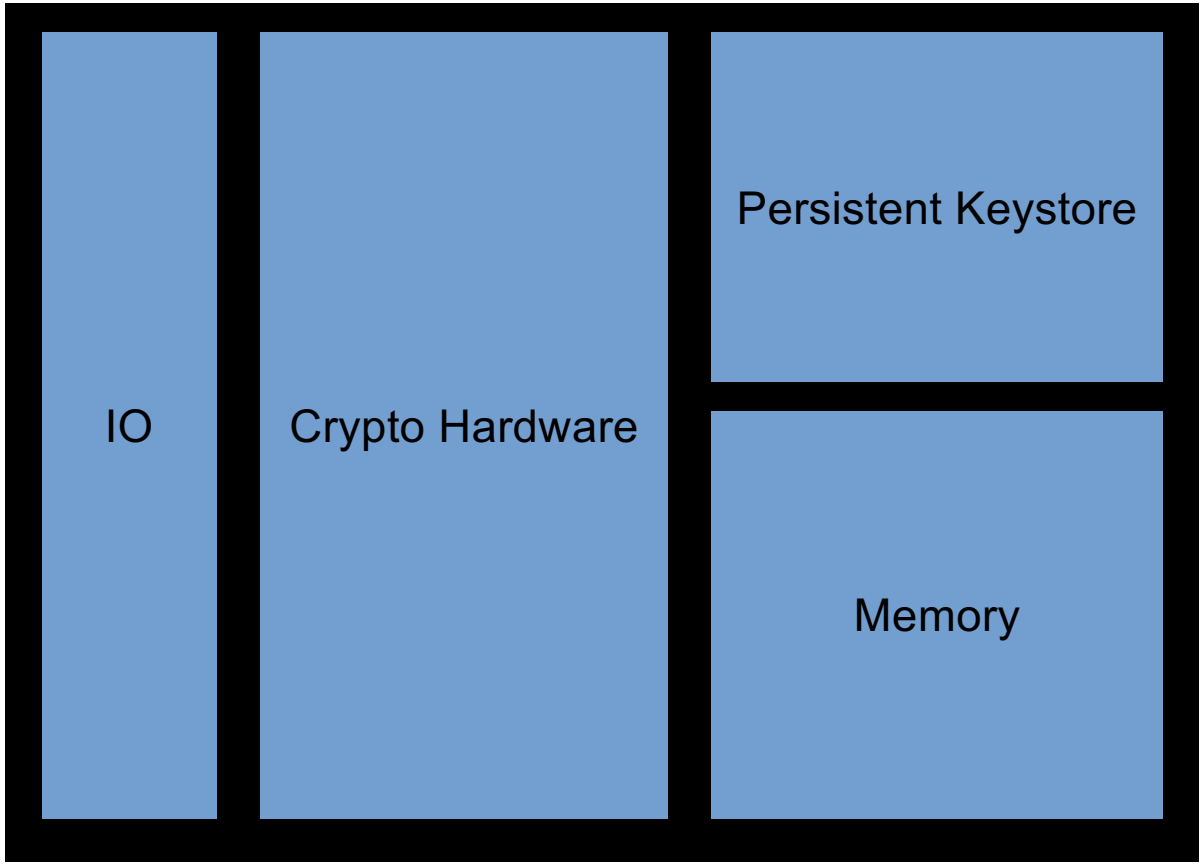
Your program

```
If(registered(video))
        Play(video);
```

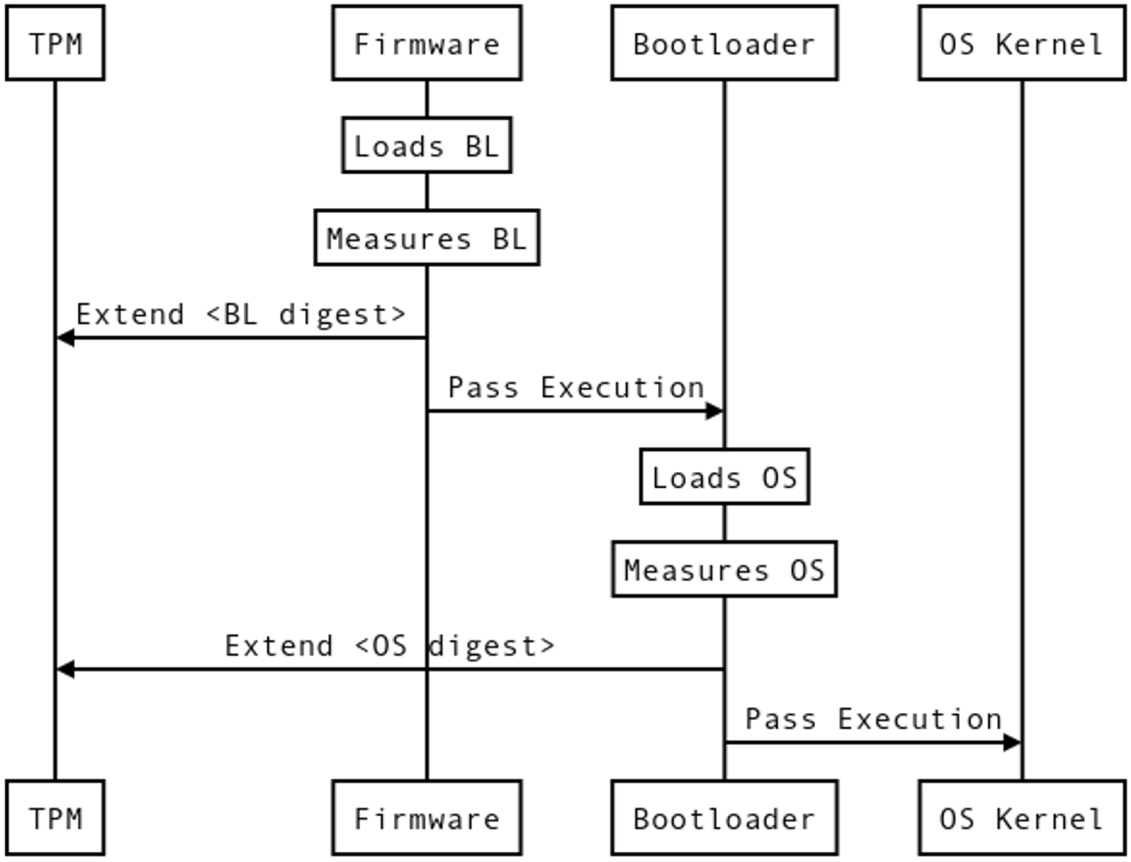
Absolutely definitely your program, honest.

```
~~If(registered(video))~~
        Play(video);
```

Trusted Platform Module

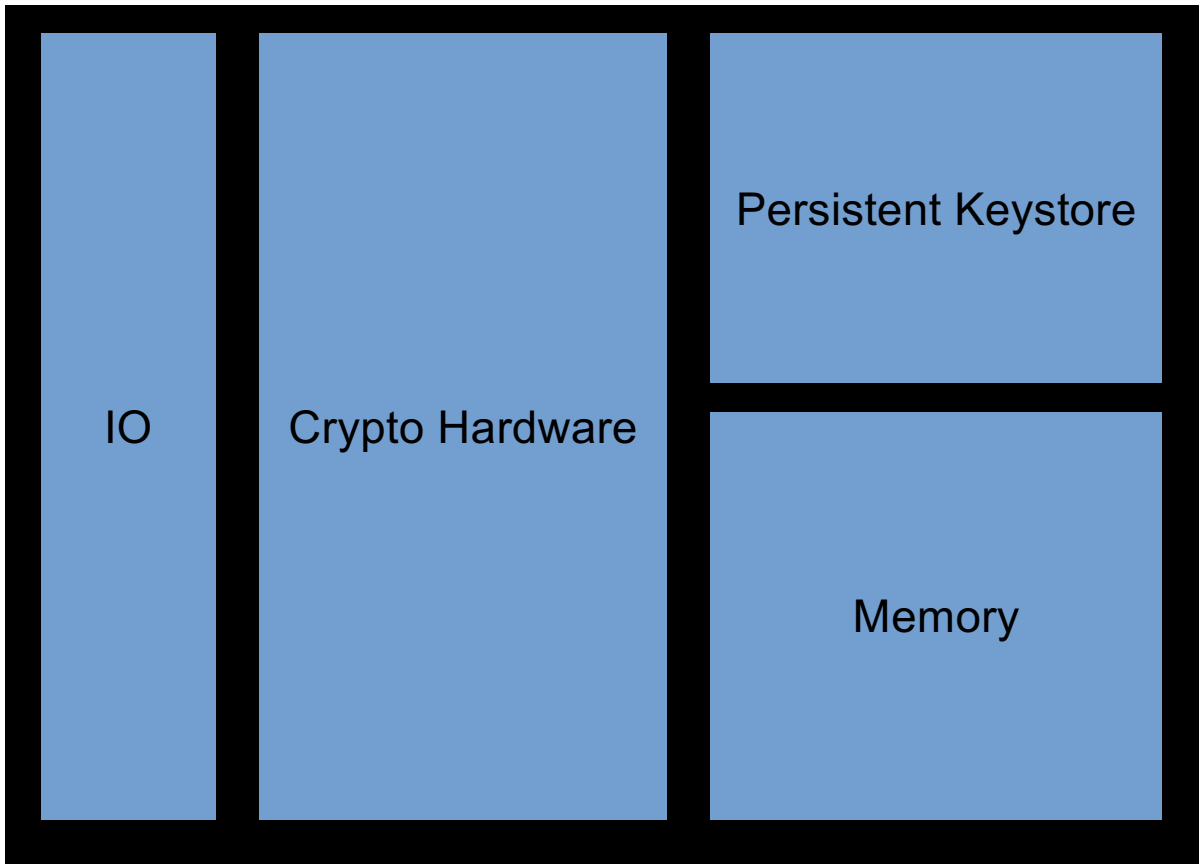


Chain of Verification



From google.github.io/tpm-js/

Secure Boot



Log

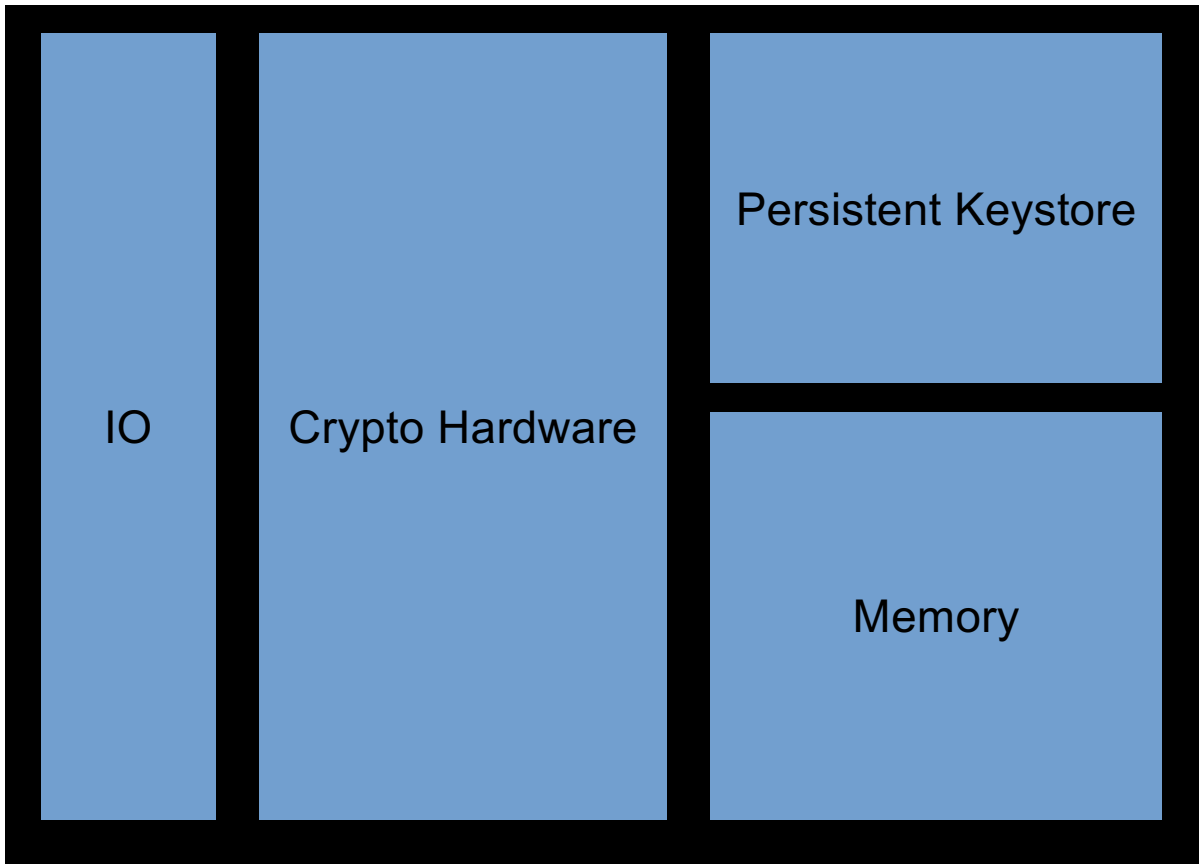
Software

Kernel

Bootloader

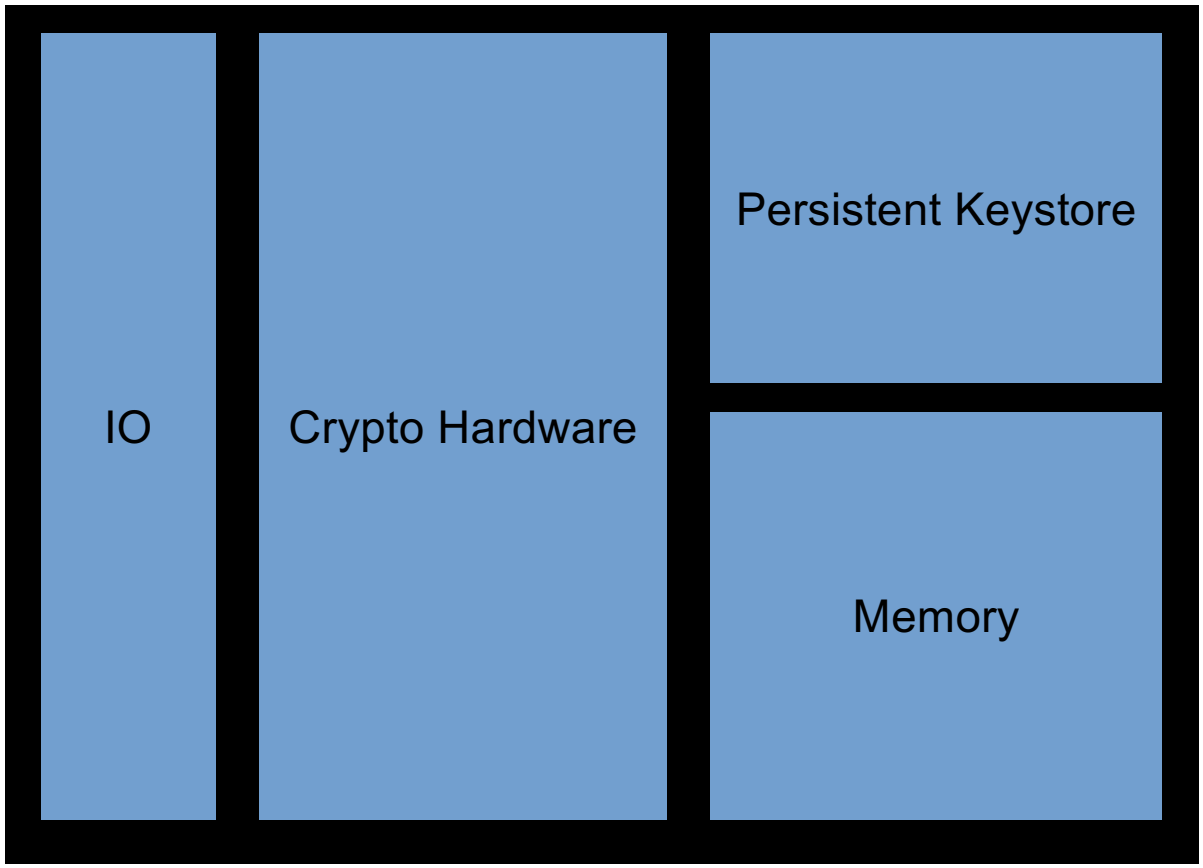
Firmware

Secure Boot


IO
Crypto Hardware
Persistent Keystore
Memory


Log
H(F)
Software
Kernel
Bootloader
Firmware

Secure Boot

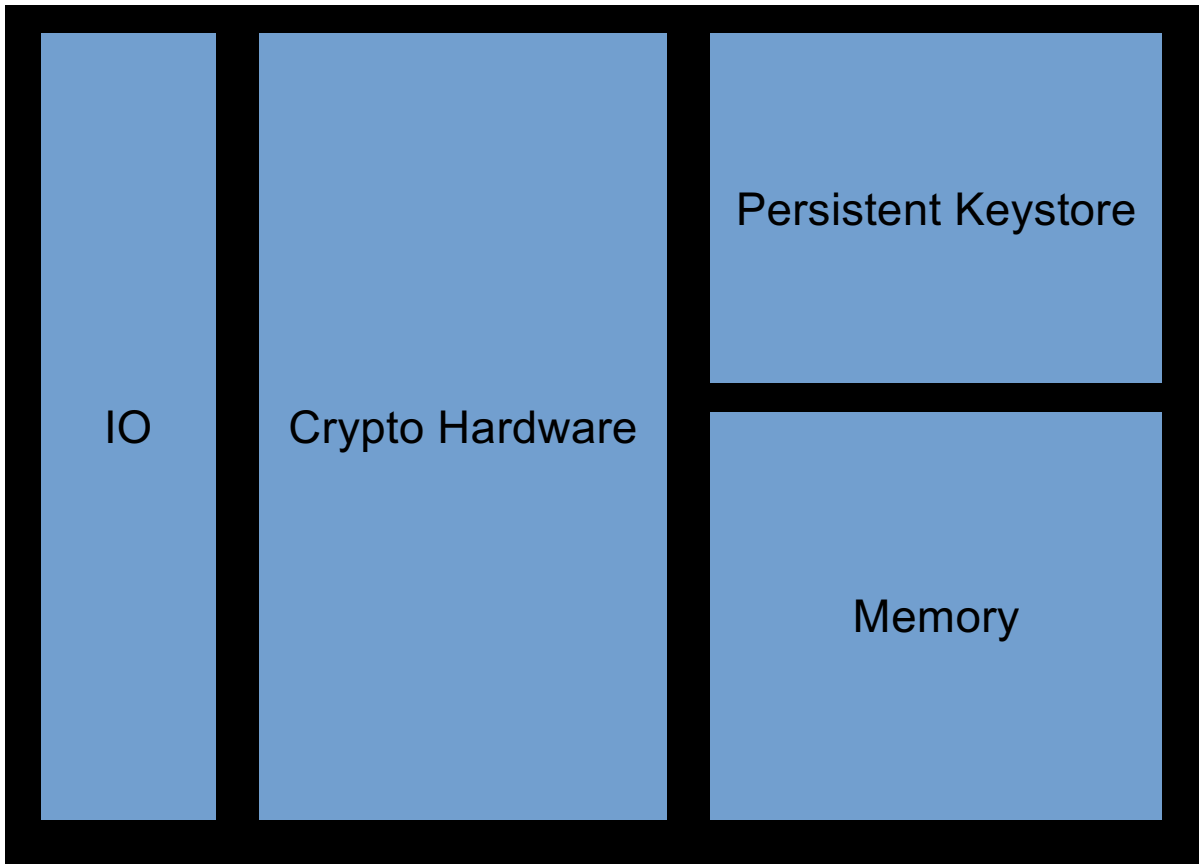


Log
H(F)
H(F,B)

⊕

Software
Kernel
Bootloader
Firmware

Secure Boot

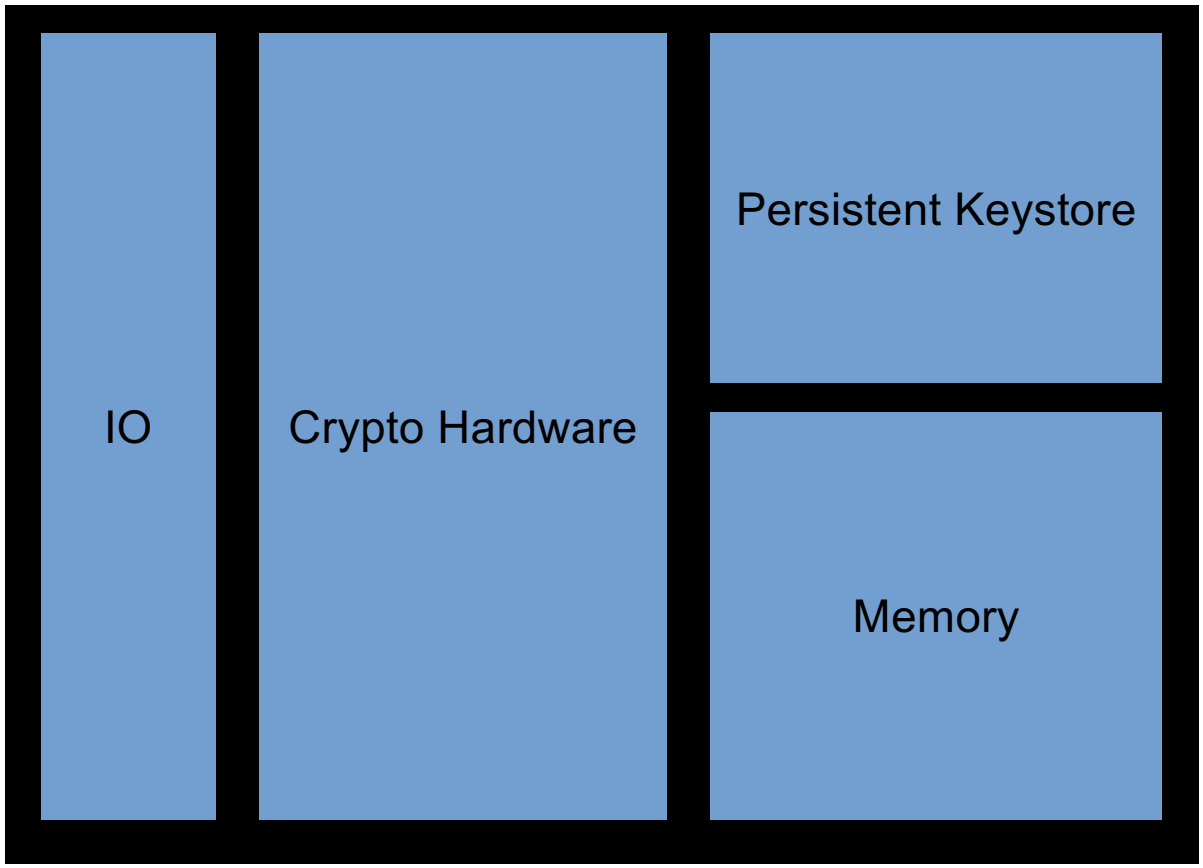


Log
H(F)
H(F,B)
H(F,B,K)

⊕

Software
Kernel
Bootloader
Firmware

Secure Boot

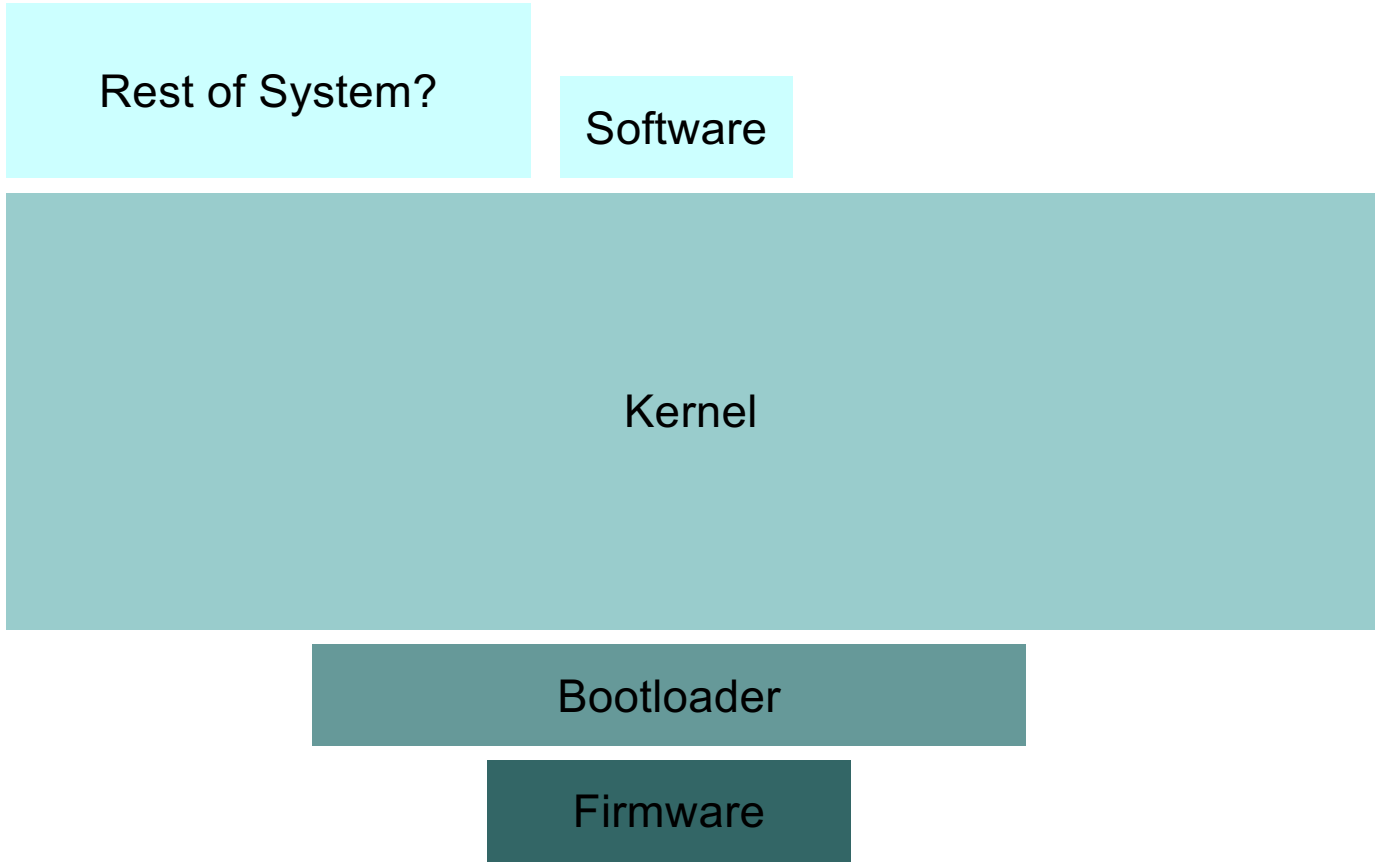


Log
H(F)
H(F,B)
H(F,B,K)
H(F,B,K,S)

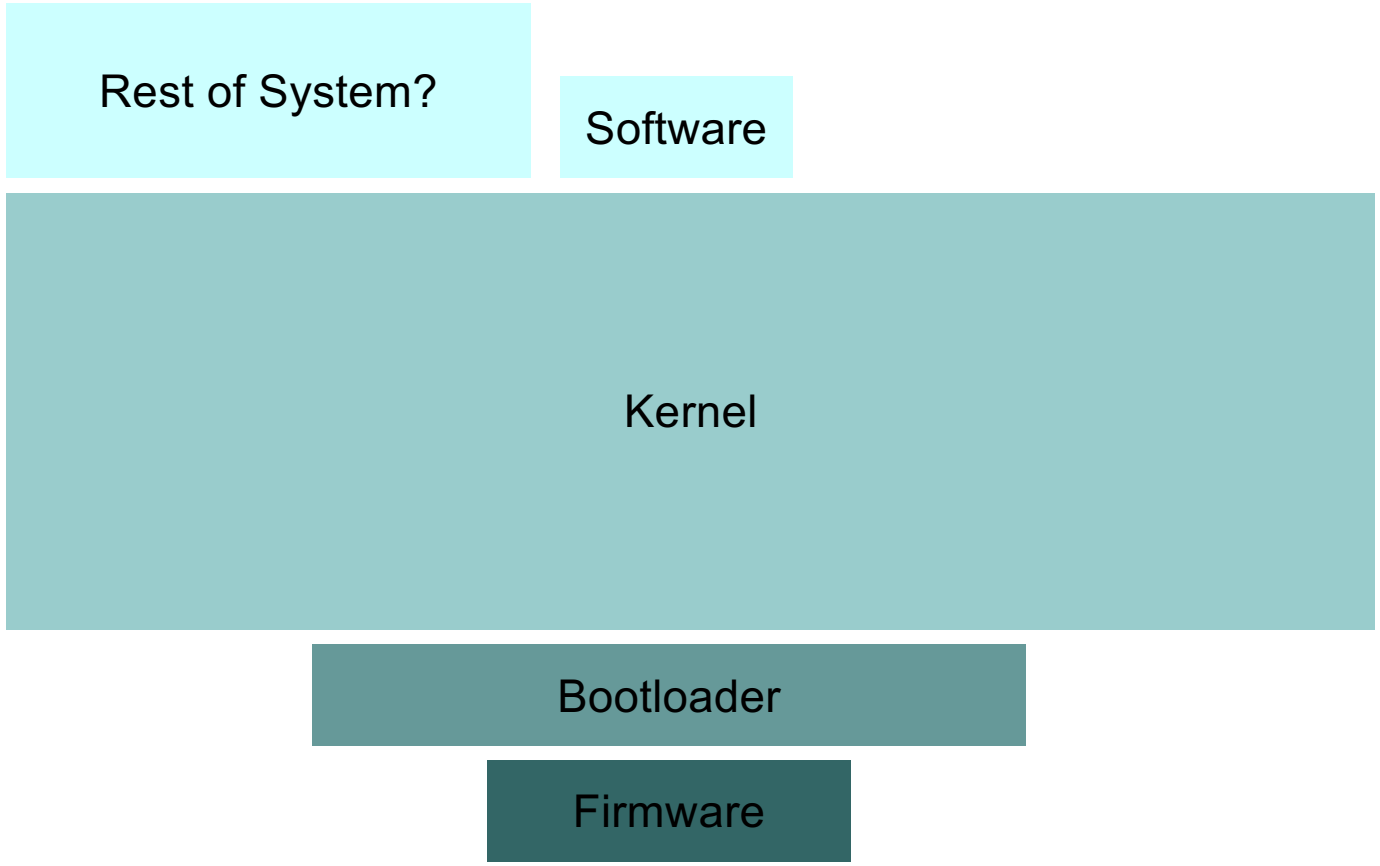
⊕

Software
Kernel
Bootloader
Firmware

Trusted Code Base

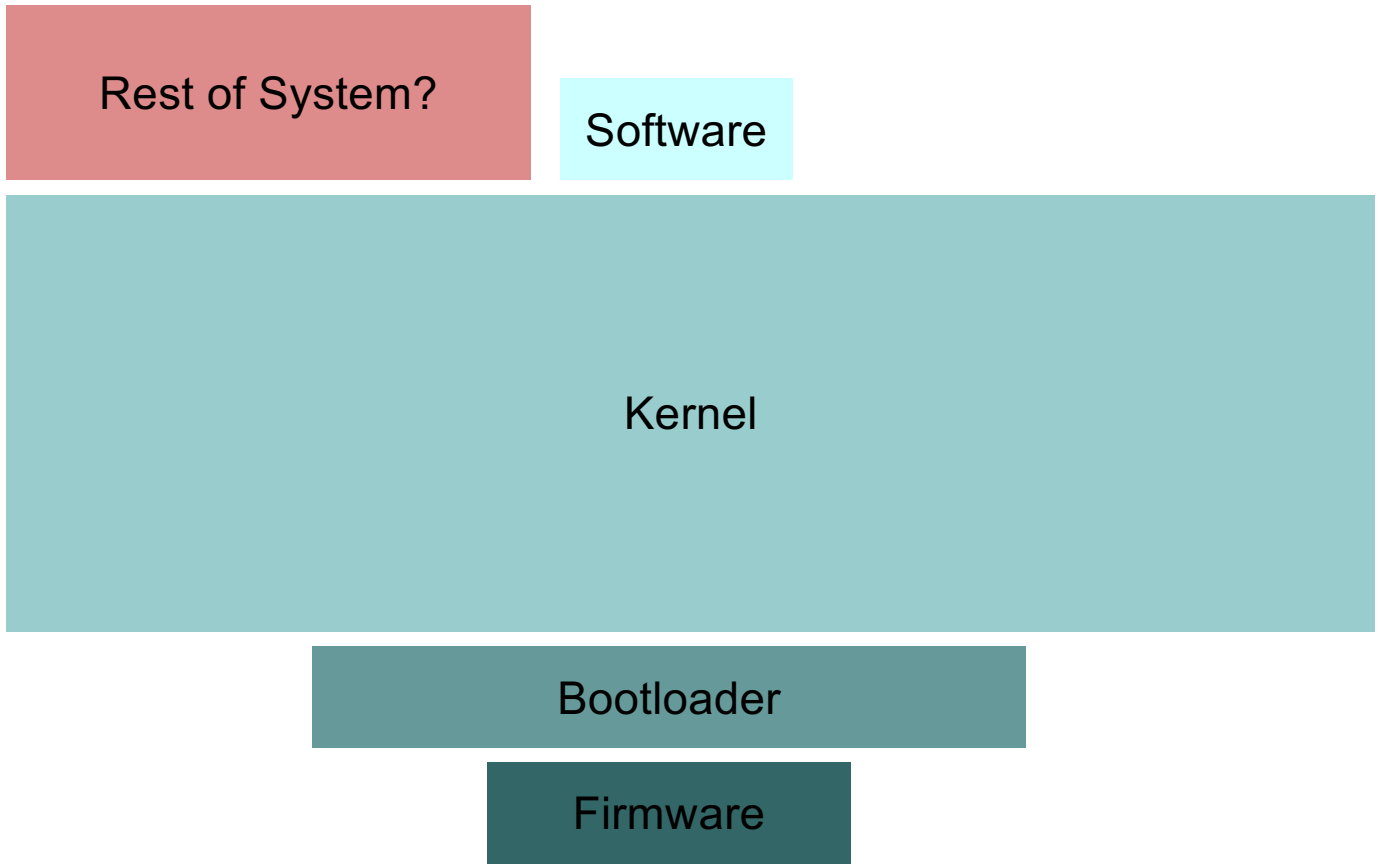


Trusted Code Base

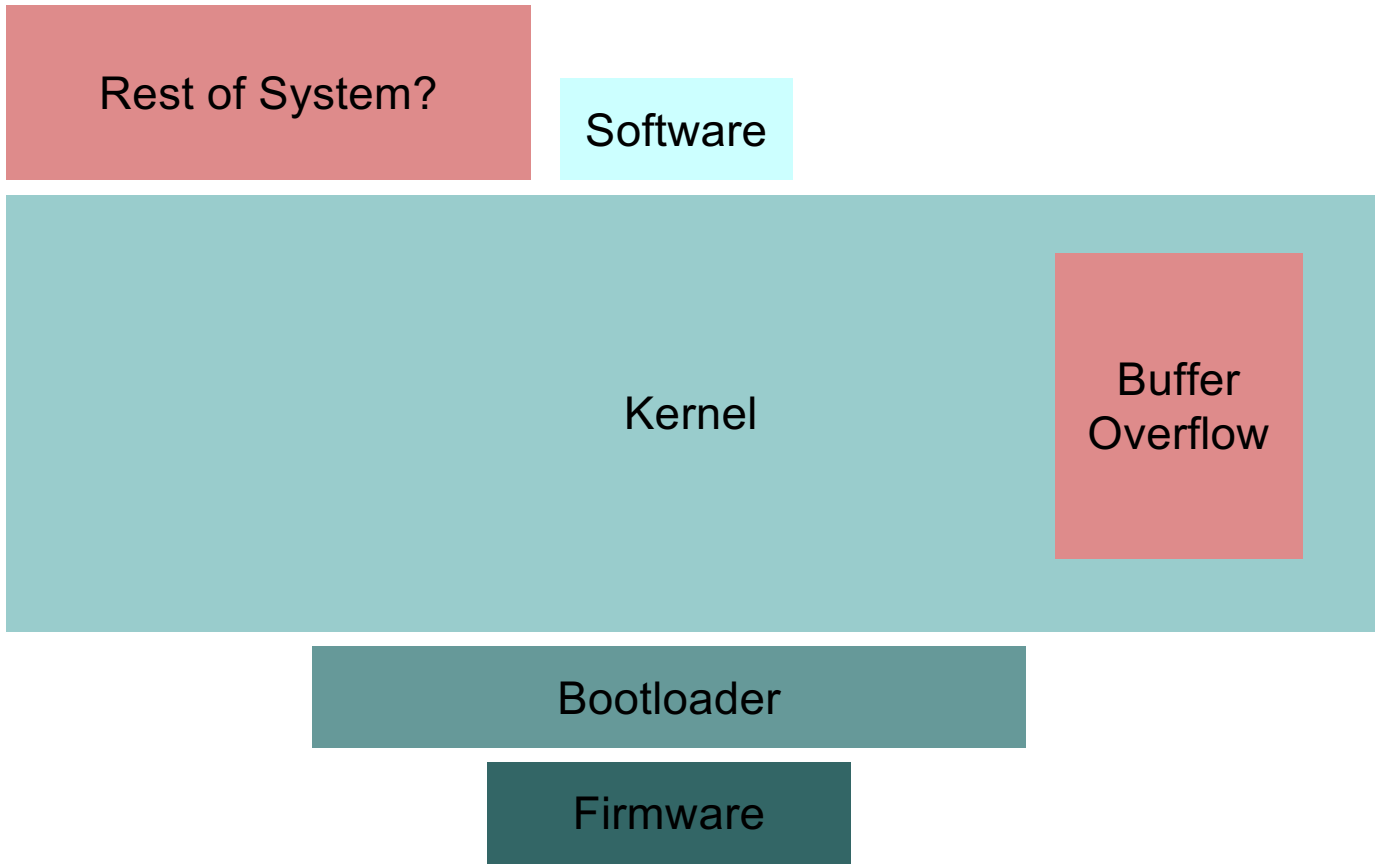


Secure Boot vs Measured Boot

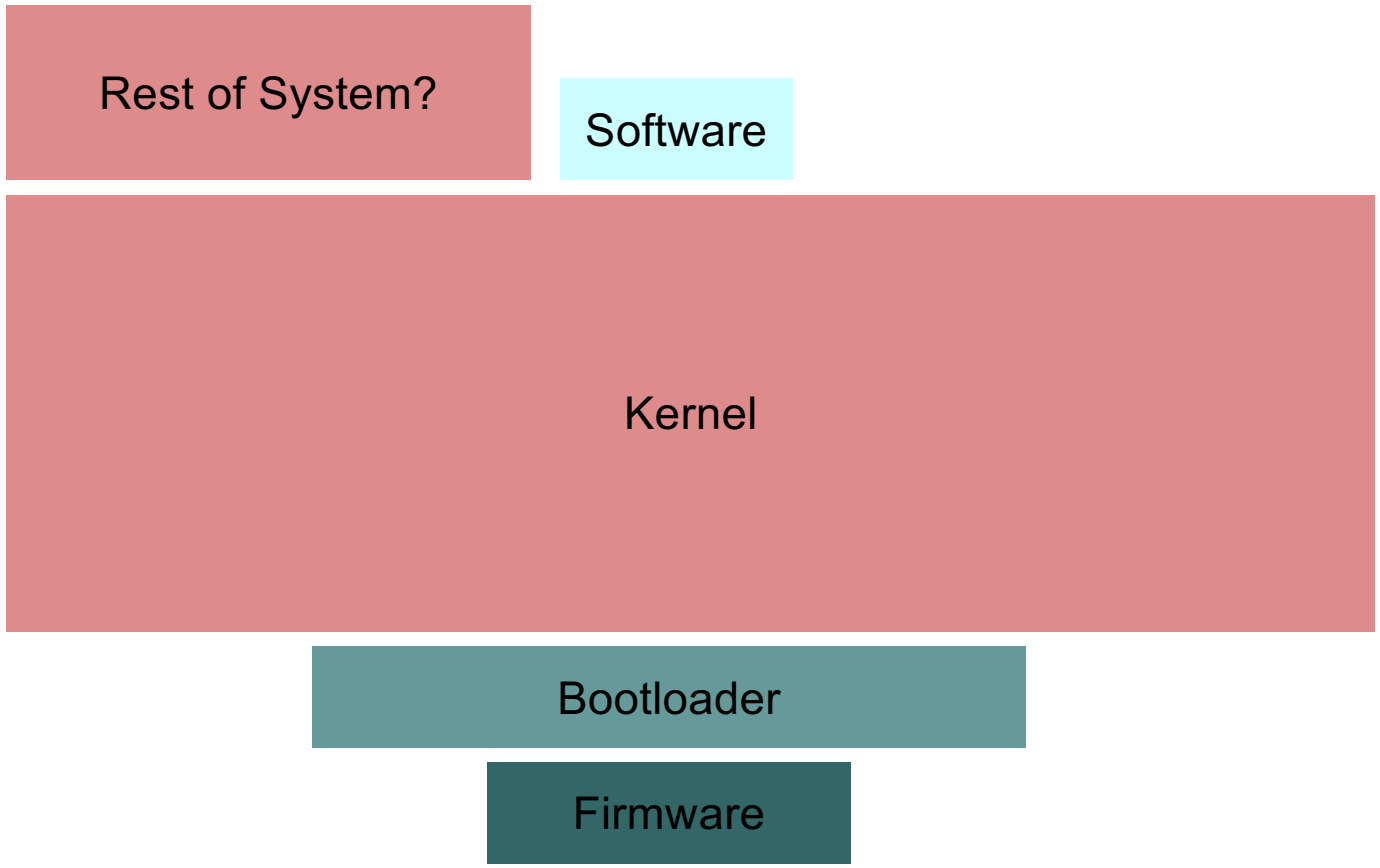
Trusted Code Base



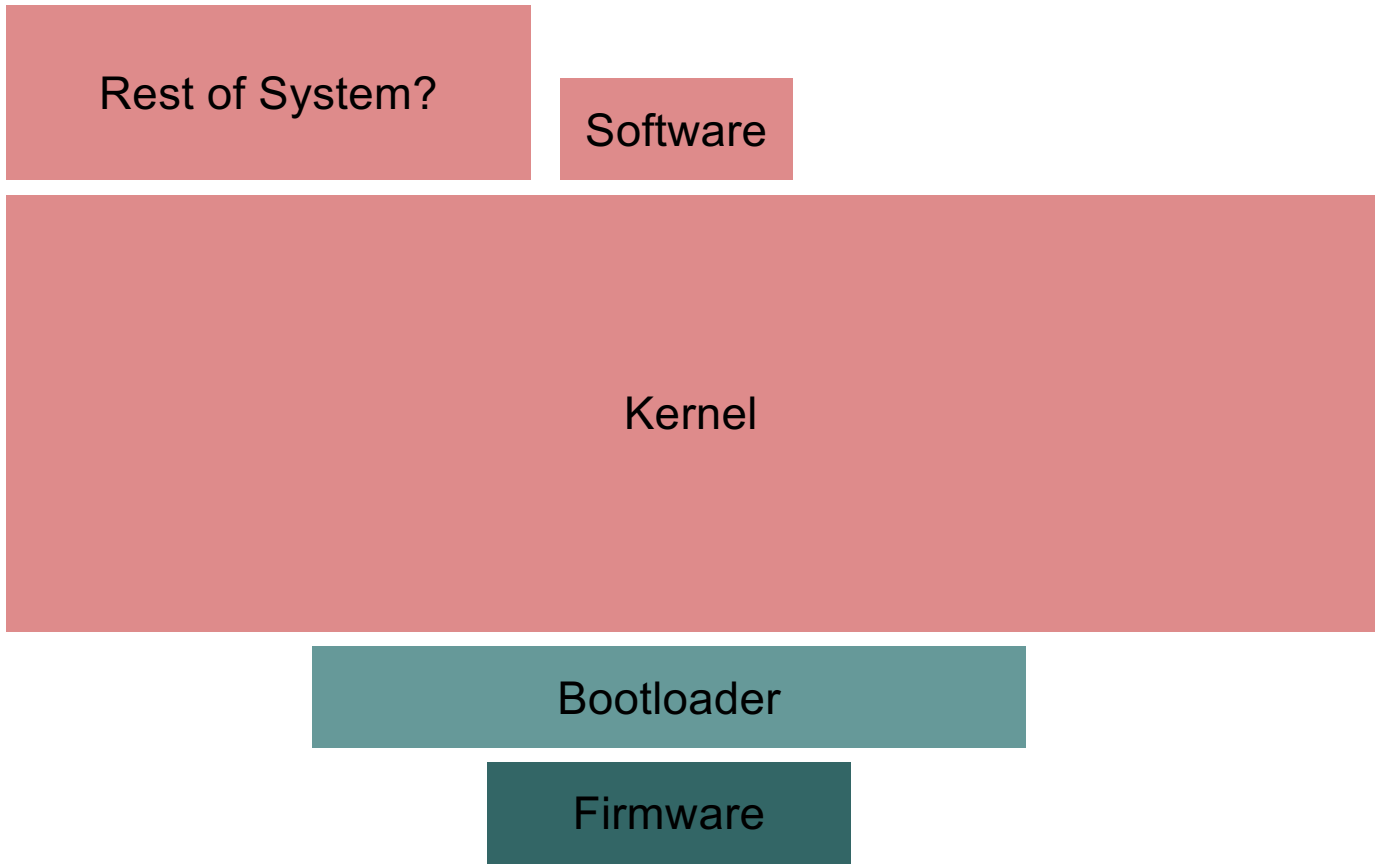
Trusted Code Base



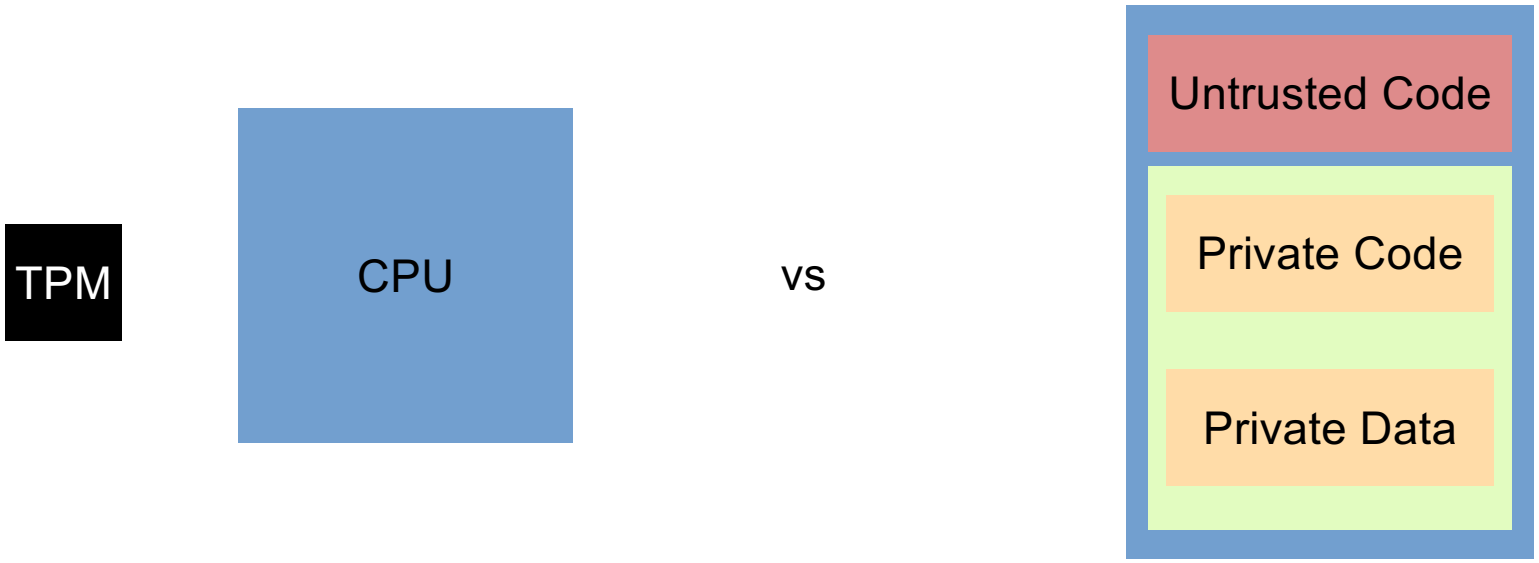
Trusted Code Base



Trusted Code Base



Enclaves (Trustzone, SGX, SEV)



Enclaves

- Isolate you from the OS, and give precise attestation of just your software, but…
- Tension around access versus isolation, e.g. Qualcomm's devices let a Trusted App map in regions of the host OS.
- Typically cause other security features to falter: often weak ASLR, large TCBs, information leakage especially over debug, no DEP…
- Side channels out-of-scope, and even Attestation can be broken by attacks on integrity (last lecture)

Further Reading

- Security Engineering Chapters 7, 6, 9, 27, 24
- J Gamblin, “Nearly 20% of the 1000 Most Popular Docker Containers Have No Root Password”, *Kenna Security Blog* May 20 2019
- Google SRS, Chapter 8: Design for Resilience, Chapter 7: Design for a Changing Landscape
- A Minimalist Approach to Remote Attestation, Francillon et al., DATE 2014
- Principles of Remote Attestation, Coker et al., International Journal of Information Security, 2011