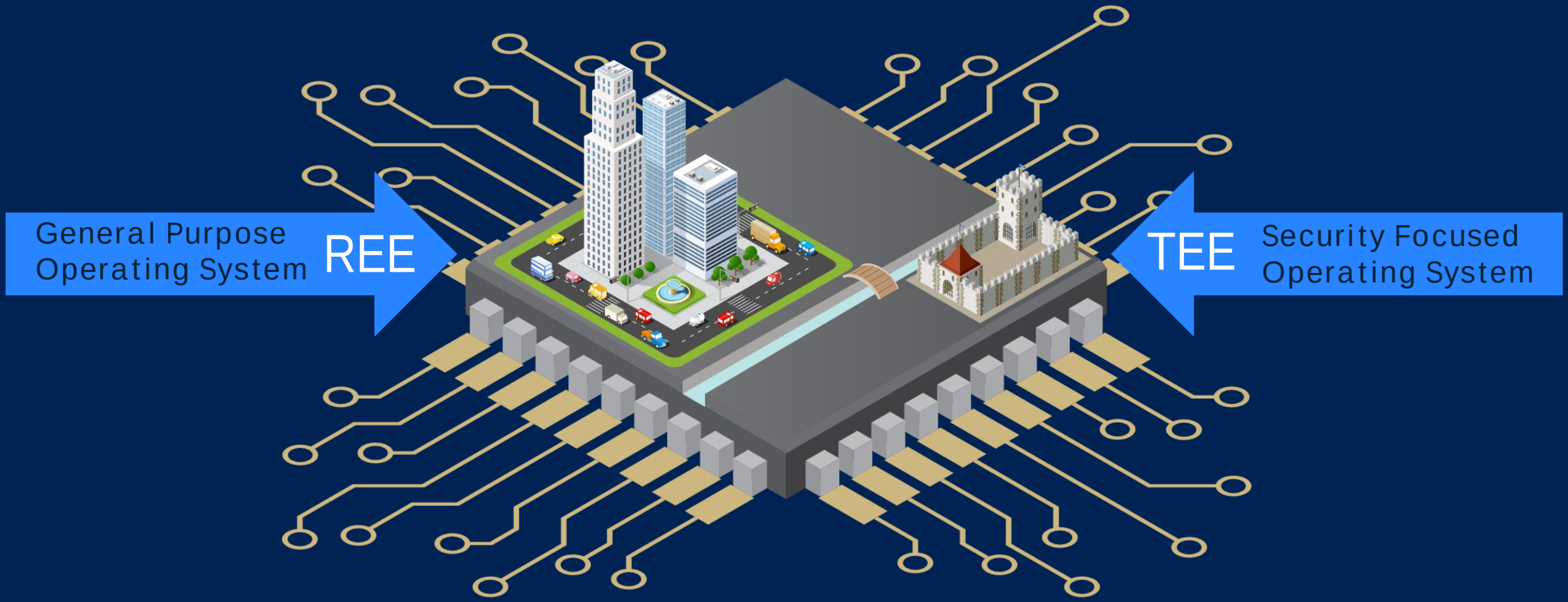


Global Platform®

TEE 2.0 Specification Development

Richard Hayton (rhayton@qti.qualcomm.com)
TES Committee Chair

What is a TEE?


General Purpose
Operating System
REE
TEE
Security Focused
Operating System

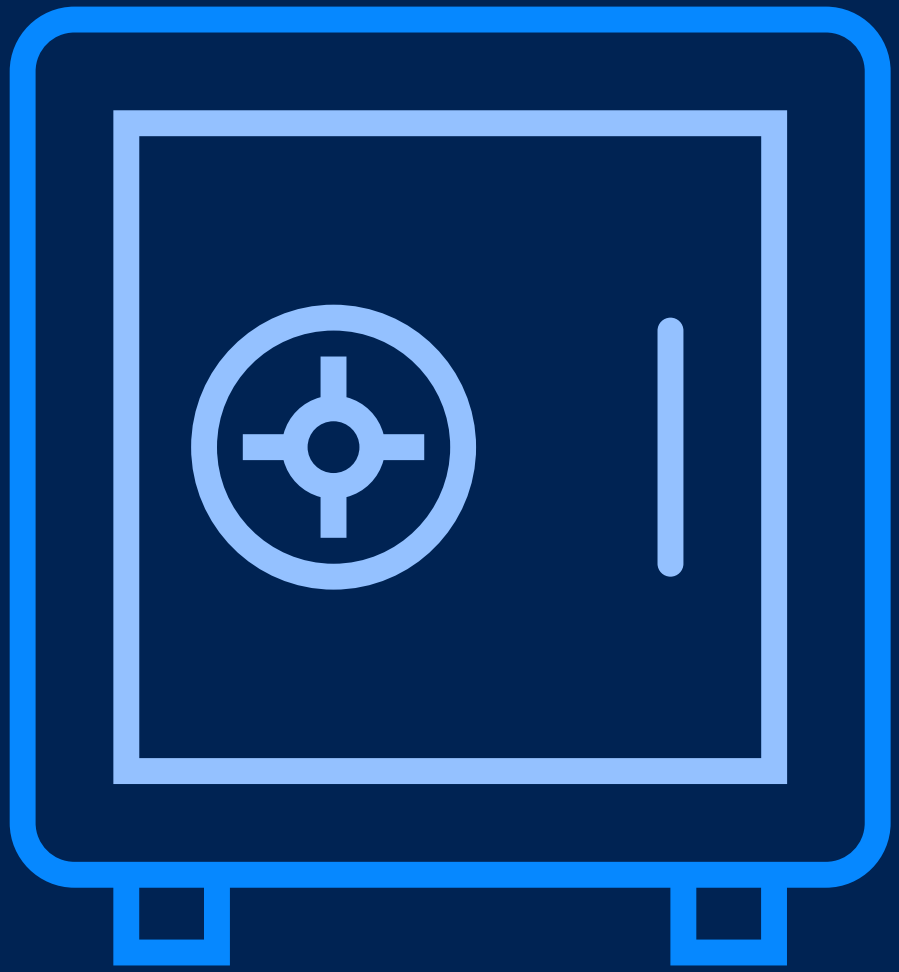
TEE Concept

Isolation is a fundamental security concept

Trusted Execution Environments isolate security critical **code** and **secrets**

GlobalPlatform TEE standardizes the whole environment

- What it needs to be secure
- Features such as crypto & storage
- How to write code – Programmer APIs.



TEE Concept

TEEs enable secure *CODE*

TEEs are operating systems running multiple security focused Trusted Applications.

Over time the range and complexity of Trusted Applications has increased significantly



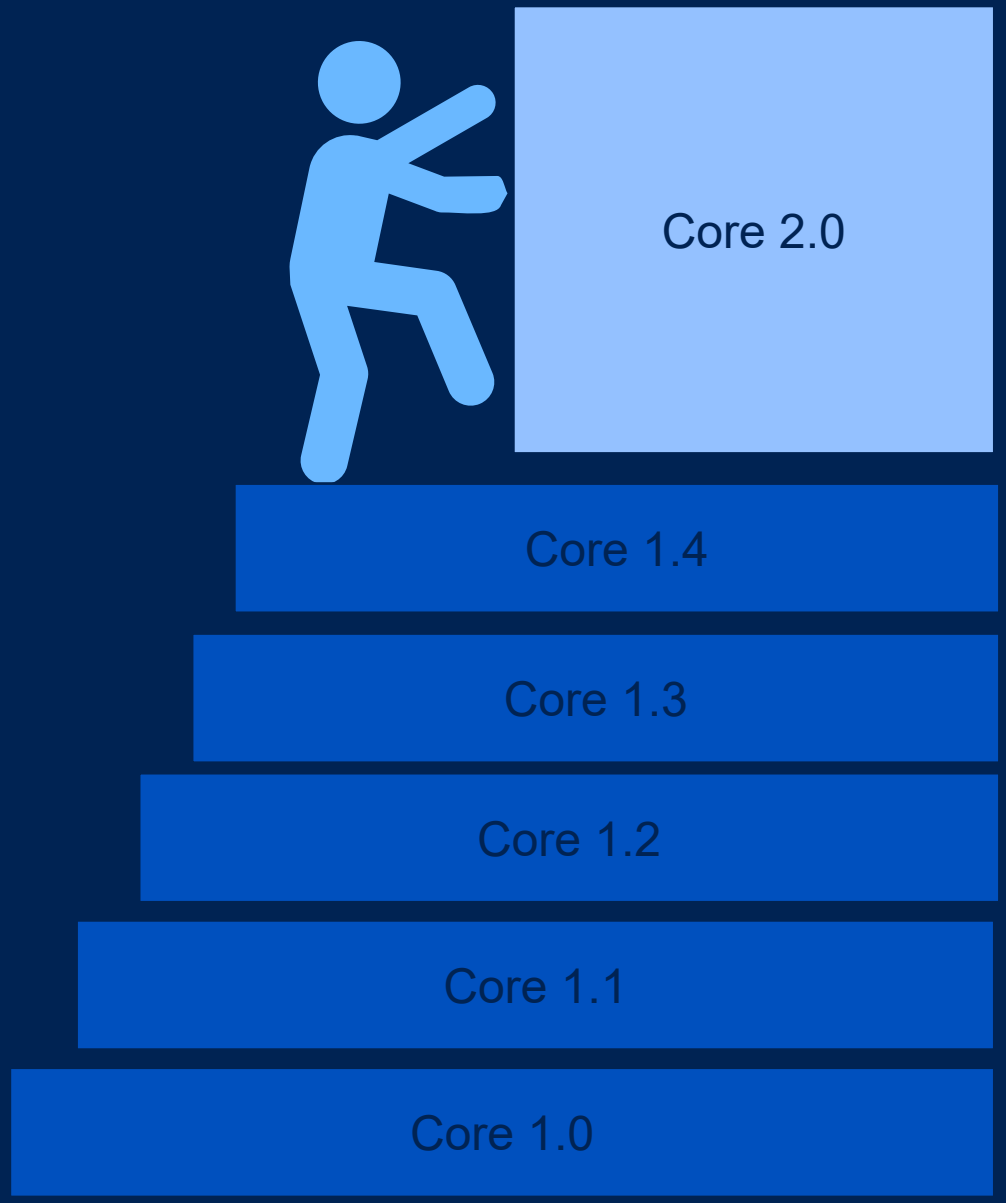
Core 2.0

Core 2.0 is an effort to prepare TEEs for the next wave of applications, and to better standardize aspects of TEE design that have been left to implementors.

Core 2.0 will essentially build on Core 1.x but is has grander ambitions than 1.x changes.

We are allowing some compatibility breaks, but expect 90% compatibility with core features, and the ability for TEEs to support Core 1.x and Core 2.x TAs simultaneously

Automotive is a key driver of new requirements



Timeline…



Requirements
Gathering thoughts and requirements

Focused on input from GP Working Groups, but open to all

Encourage members to join Committee

Committee work (Spec Working Group)
- to standardize new Core 2.0 APIs.
- Adjust scope for 2.0 .v. 2.x
- Potential impact on other TEE specs(?)

Pre-Publication
- Review rounds:
  - Committee
  - GP members
  - Public
- Publication

Core 2.0 Topics

Larger, more complex applications

- LibC / Porting Support
- Multithreading in Trusted Apps
- Concurrent Request Dispatch

Safety and Security

- RUST
- Safe C APIs
- Fault Resistance

Heterogeneous workloads

- Resource Isolation (FFI)
- Memory Sharing (DMABuffs)
- Multiple TEEs and Service Providers

An interoperable TEE ecosystem

- Alignment of TEE and Micro-TEE concepts
- Callbacks to NWDa and between TAs
- Standardizing TEE Drivers

Threads

Core 1.0s explicitly disallow many types of concurrency

However, there is a strong desire for multithreading:

- Within a TEE, to support multiple TAs executing at once (allowed in Core 1.0)
- Within a Trusted App to service many clients at once (disallowed in Core 1.0)
- Within a Trusted App to divide execute a larger task in parallel (possible by adding proprietary APIs)
- There is also a trend to isolate more functions into different TAs to enable greater isolation
  - This in turn increases the desire for more flexible threading / TA-TA calls.

The “most basic” proposal for Core 2.0 is to embrace PThreads as a programming API

However, there is a lot more subtlety involved

- Breaking the principle of single-call-at-a-time has impacts on other APIs
- APIs need to be thread safe (or not)
- New opportunities to handle old problems using threads (e.g. keep alive).
- But there is also a desire to limit impact on existing systems – especially NWD client stacks.
- *This is a deep technical topic, catch me over coffee!*

RUST

Core 1.x APIs are C-based

Several TEEs also support C++ or RUST in *non-standard, and sometimes ugly ways (for example requiring unsafe calls to C APIs).*

A major topic for Core 2.0 is to define RUST bindings in the GP Specification, so that each feature is given both C and RUST bindings.

There is also work underway to build a RUST to C mapping, with a strong probability this will be donated to GlobalPlatform under an open-source license.

Example:

```
fn encrypt_with_new_key(data:&[u8],iv:&[u8]) ->
 Result<(Vec<u8>,ObjectHandle)>
{
 let mut k  = ObjectHandle::new(
                ObjectType::Aes, 256)?;
 let mut op = OperationHandle::new(
                Algorithm::AesCbcNoPad,
                OperationMode::Encrypt, 256)?;
 let mut out: Vec<u8> = Vec::new();
 k.generate_key(256, &[])?;
 op.set_key(&k)?;
 op.cipher_init(&iv)?;
 op.cipher_do_final(&data, &mut out)?;
 Ok((out,k));
}
```

Ecosystem: Drivers

An earlier view of TEEs was that each SIP would standardize on one TEE, and that standardization should focus on the Trusted Application Writers.

However, today many SIPs have a need to support multiple TEEs, as different customers have different preferences/requirements.

This means there is value in standardizing the HAL between the TEE software and the CPU/Hardware it uses

This is a complex topic, but one Core 2.0 will consider

An example

A SIP may have a solution with a hardware crypto engine that is controlled via a mix of shared memory and bus interfaces.

Today a different TEE driver for this hardware would have to be written for every TEE that wished to use it, even on the same chip.

We will consider standardized TEE APIs forfeatures such as memory mapping and interrupts to allow the driver to run as a GP TA.

All still to be discussed…

Ecosystem: TEEs & Micro-TEEs

Arm has recently donated technology related to Micro TEEs to GlobalPlatform (see later talk)

An aim of Core 2.0 *and* Micro TEE activities is to align the two existing Ecosystems

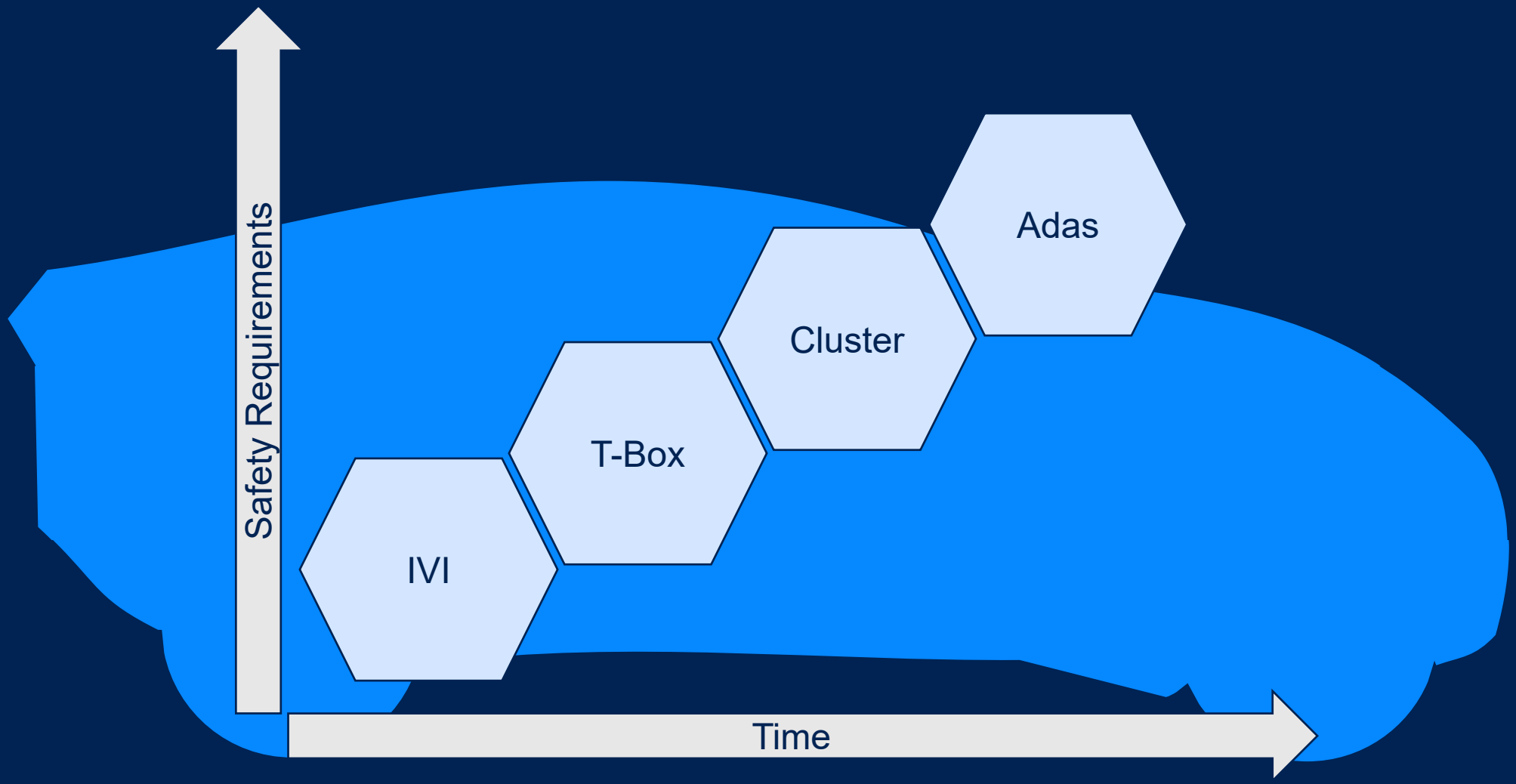
Some of this will come down to language / terminology used in specifications

However, there is also the opportunity to cross pollinate, and better align concepts.

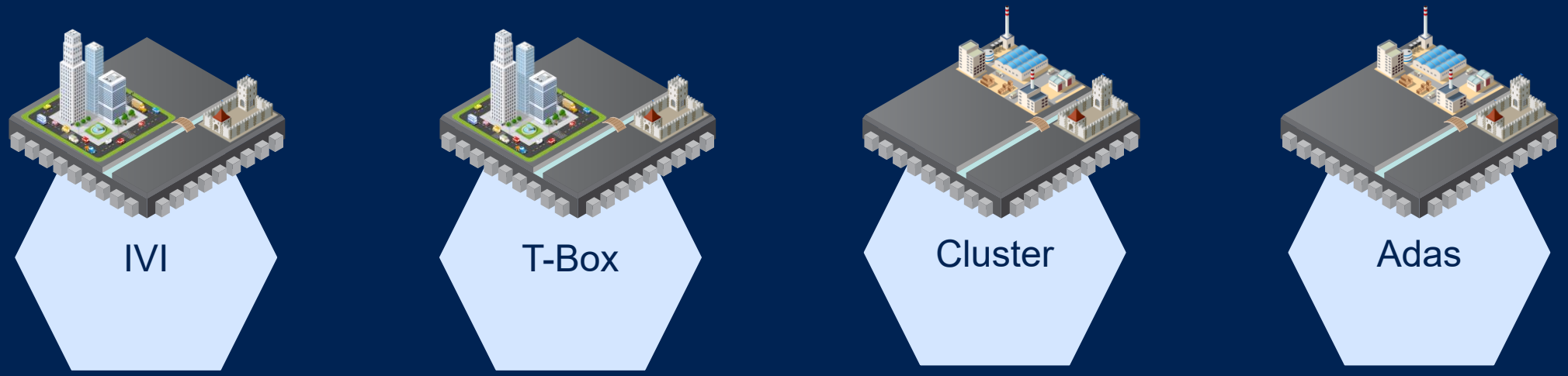
Starter for 10:

- Adopt Memory Handles as first class objects
  - These are core in FFA/FFM architectures, but also present elsewhere (e.g. DMABuffs in Unix)
  - This is a gap in GP Core 1.x APIs we can fill
- Adopt session-less APIs
  - Core 1.x supports only session-based comms
  - FF-M has session-based and session-less APIs
  - Session-less APIs could be a good addition for performance and simplicity.

TEEs in automotive…



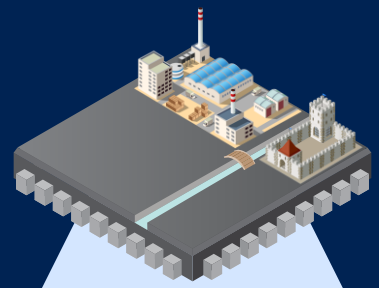
Multiple CPUs / Multiple TEEs (unrealistic picture)



Domain Consolidation



IVI



Adas

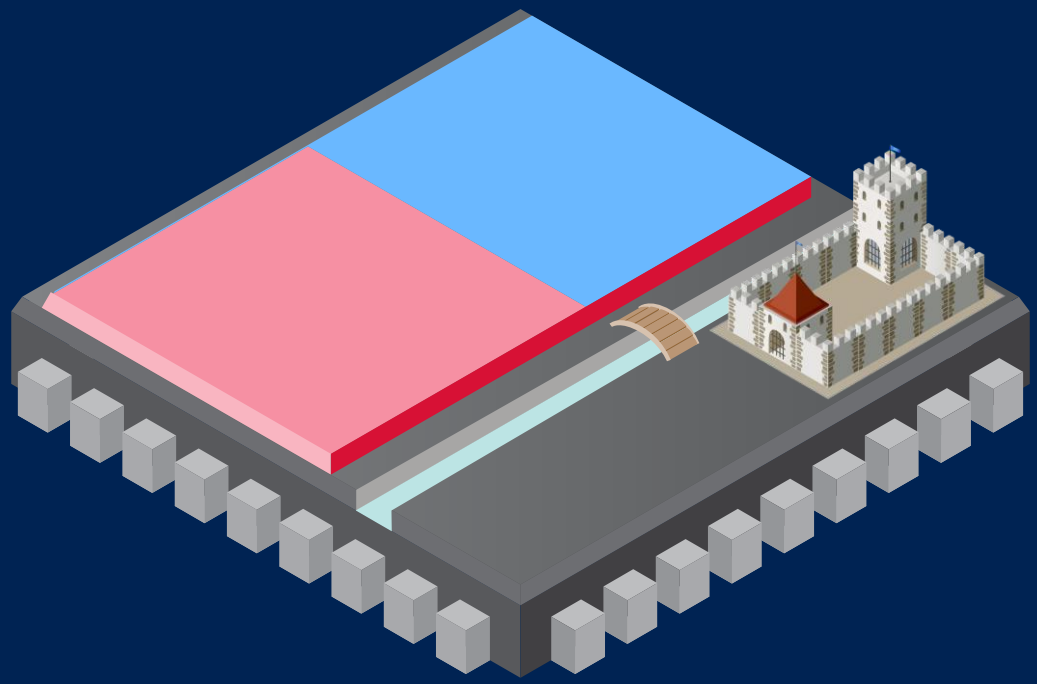
Domain Consolidation



IVI



Adas

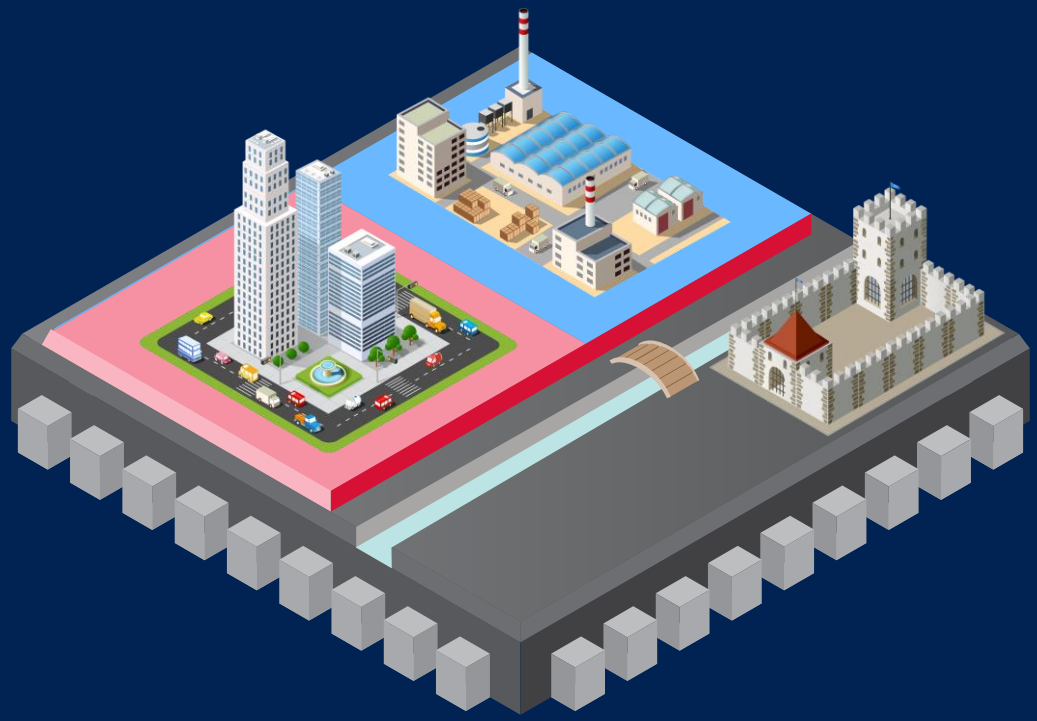


Domain Consolidation

IVI

Strong security isolation ✓

Strong safety isolation ✓

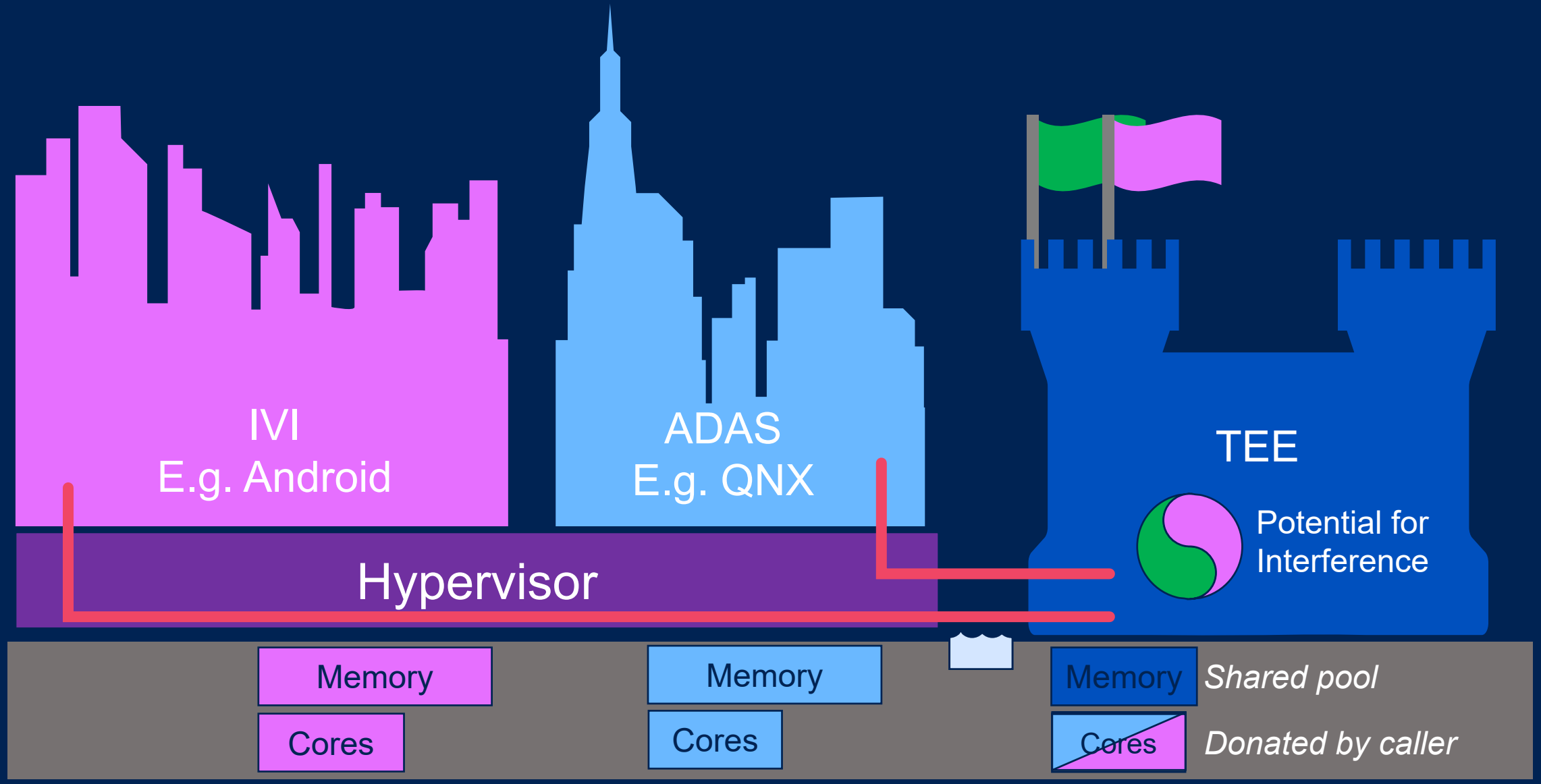


Adas

Strong security isolation ✓

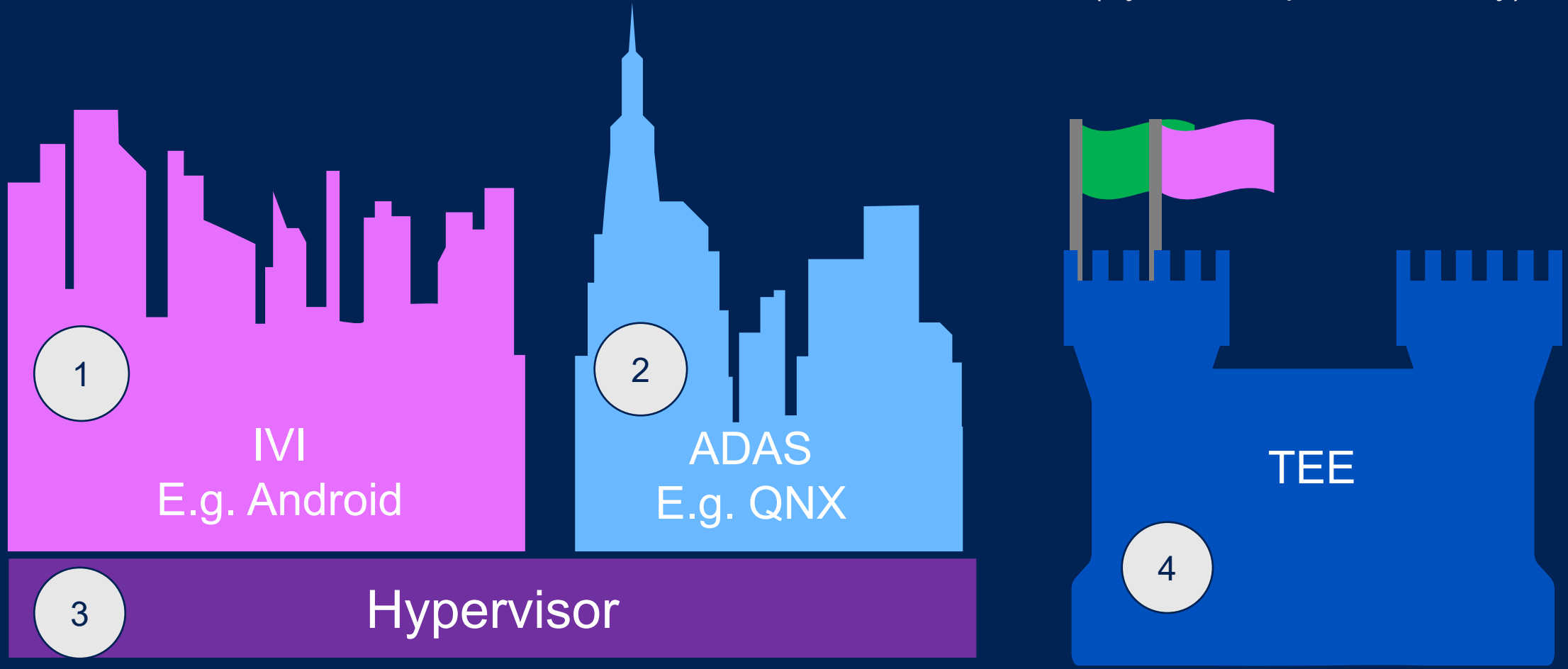
Strong safety isolation ?

Isolation in a TEE



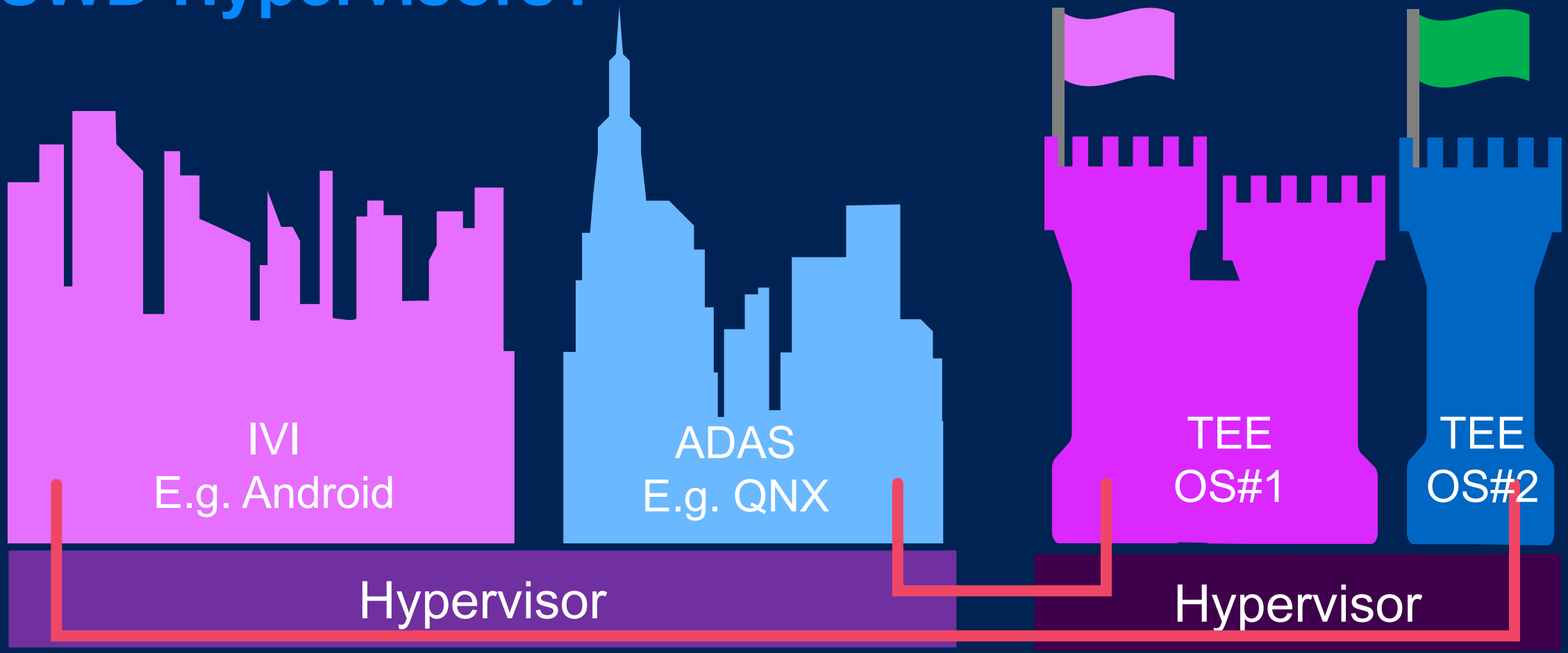
Where do device drivers live?

Flash (storage)
SMMU (System/Peripheral Security)



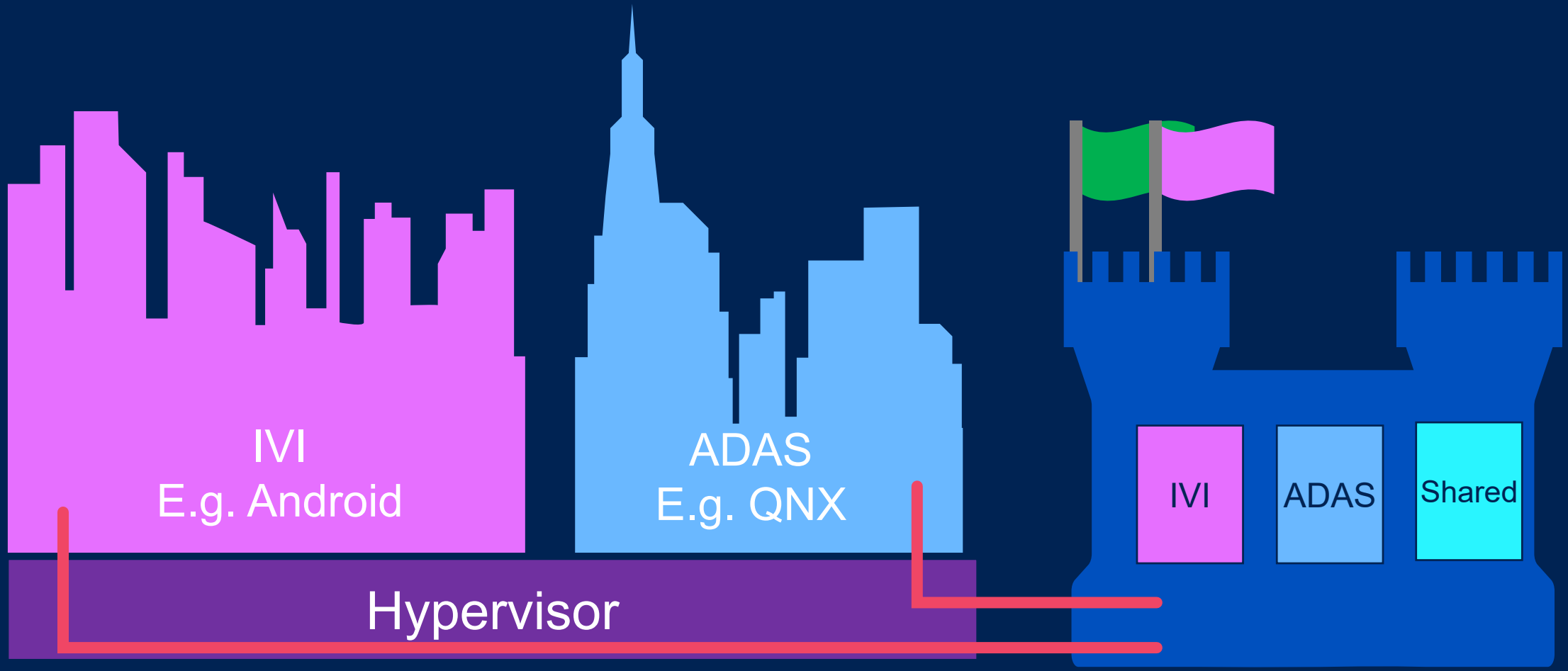
1. Bad. ADAS would rely on IVI
2. Possible – but ADAS is also now a device service
3. Push problem to hypervisor
4. Great for security. How do we isolate for safety…?

SWD Hypervisors?



- Supported in newer Arm cores – but not widely adopted yet
- Would allow each VM to be paired with a TEE OS
- Simplifies thinking about isolation
- But risks making sharing of device drivers even harder…

TEE Containers ?



Core 2.0 is explicitly looking at safety workloads

It is too early to draw any conclusions yet, but …

It is clear that in some circumstances, SWD hypervisors and multiple TEEs will be used

- For safety isolation
- To allow legacy / new sub-systems to co-exist
- For a variety of other reasons

There is also a demand for some form of containerization

- Desire to isolate VM services for security reasons (e.g. separate key stores)
- Tension between isolation and cost
- Possibly for safety isolation?

Summary

GlobalPlatform TEE's have been widely successful

They are found in virtually every mobile phone and set top box, plus every(?) IVI unit and millions of other domain controllers.

Core 2.0 is about preparing for the next wave of applications.

There are lots of other API tweaks / new API features that have been suggested by TEE users

We welcome any additional input – and especially from those able to join the Committee and work on the new specifications.



Global Platform®

Securing the digital future

→globalplatform.org