TRUSTONIC

Global Platform™

TEEs on automotive ECUs, mixed criticalities, spectrum: today & tomorrow

Richard Hayton

Chief Strategy and Innovation Office, Trustonic Ltd.

Chair Automotive Task Force, GlobalPlatform

Chair Trusted Environments and Services Committee, GlobalPlatform

The story so far

Hardware Centric Approach

- Device (ECU) per function
- Requirements specified in concrete hardware terms from a “real time” perspective
- Complex physical system. Expensive to build and dependant on many suppliers
- Lowest common denominator system security (e.g. CAN)
- Fixed function

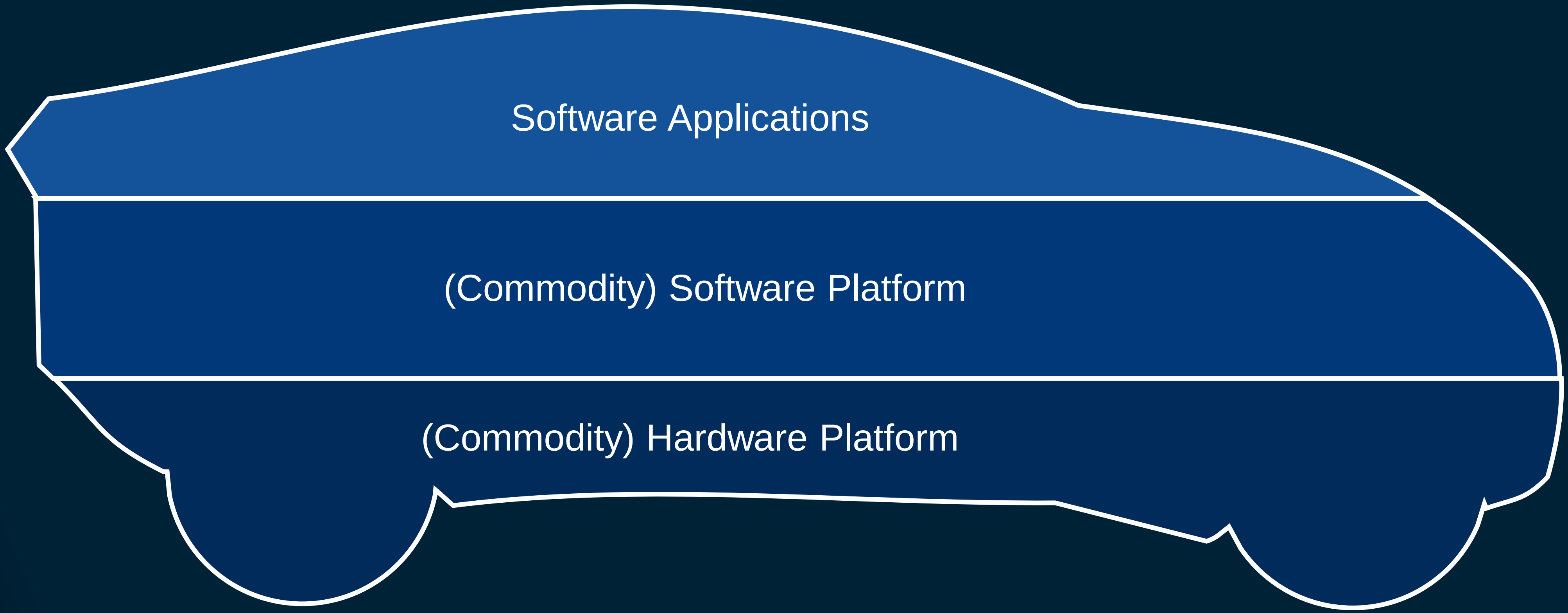
Drivers

- Is software a better way
- Perhaps requirements were too strong(?)
- Money to be saved?
- Regulators demand better security
- Customers expect app-like update frequency

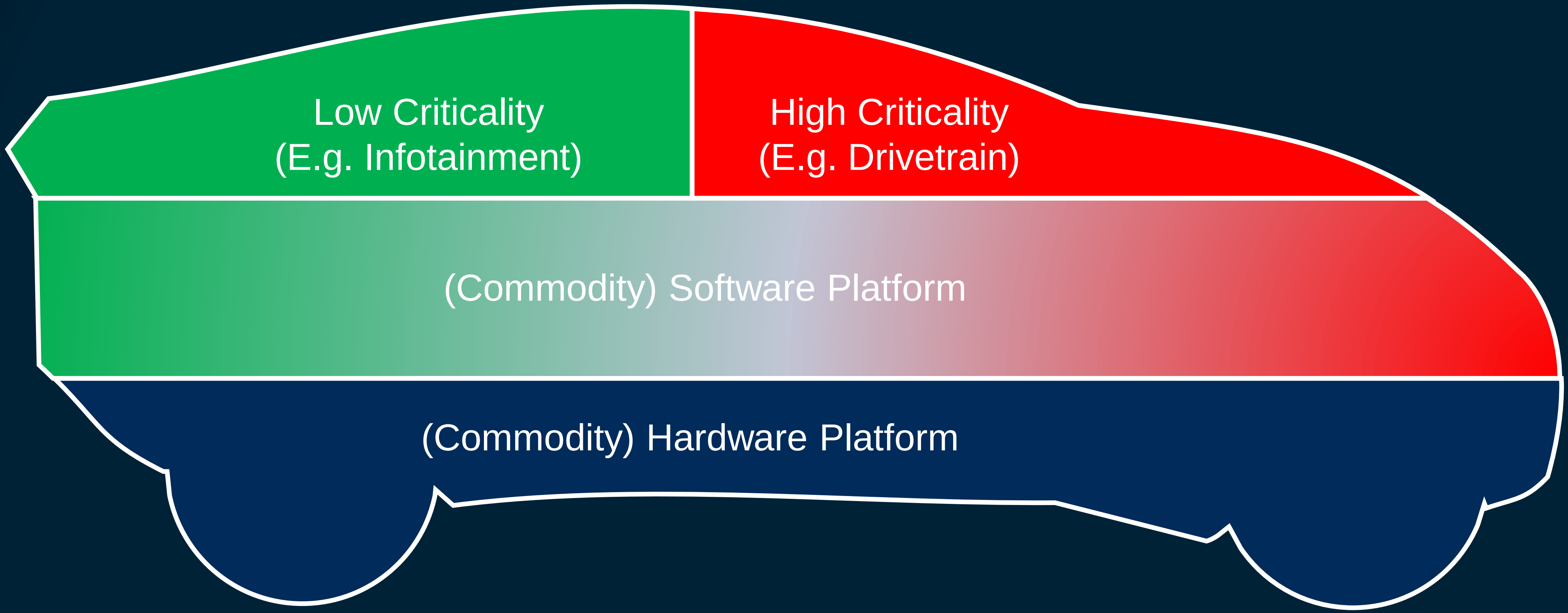
Software Centric Approach

- ‘App’ per function
- Functions specified in software, sharing common hardware / peripherals
- Commodity hardware
- Complex software system
- Up to the minute security
- (but needs constant update)
- Promise of feature updates.
- (But need to change business model?)

Software Defined Vehicles



Robustness Needs for Mixed Criticality

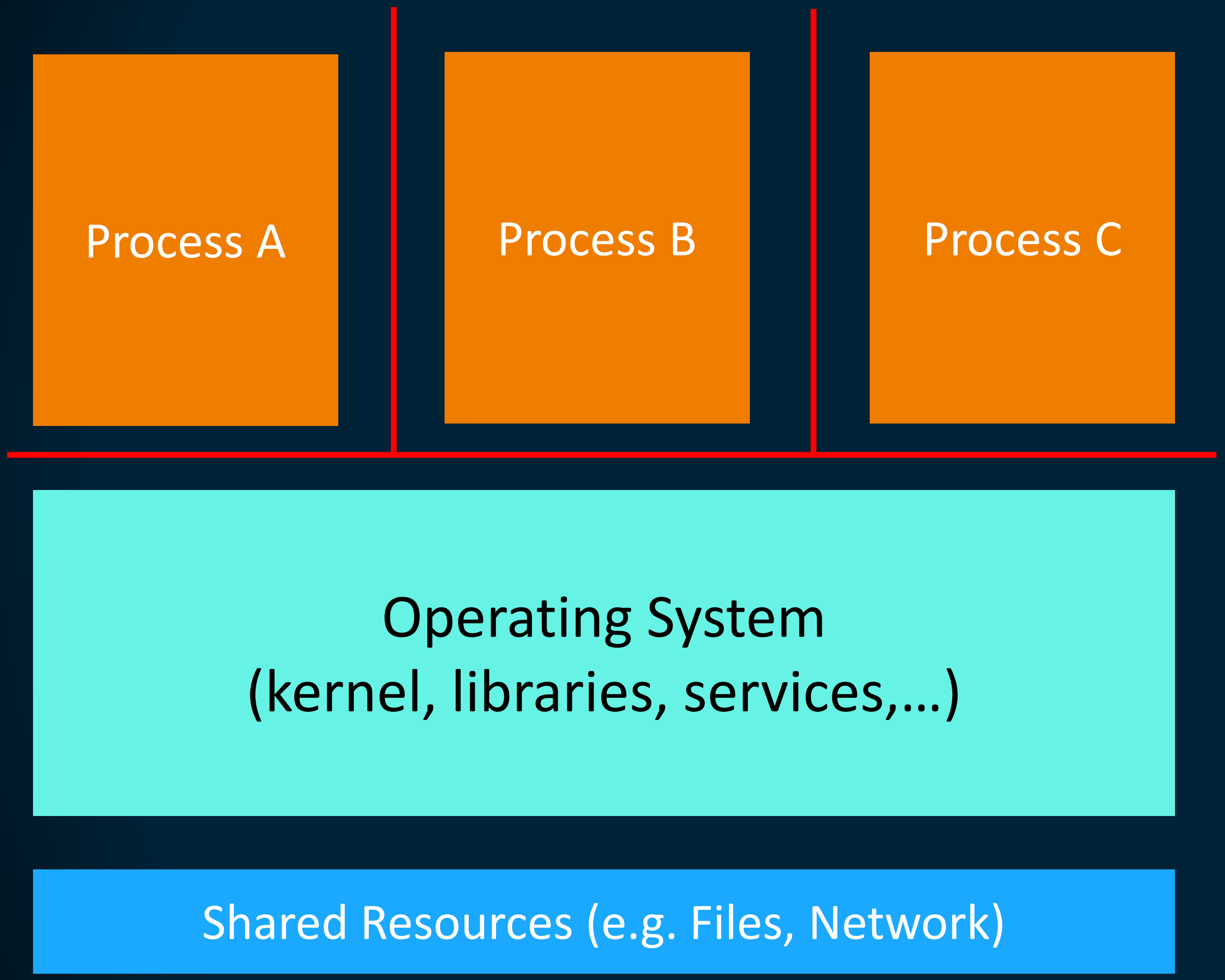


- - **Security** (attack on low criticality does not impact high criticality)
- - **Failure** (failure of low criticality does not impact high criticality)
- - **Performance** (degradation of low criticality does not impact high criticality)
- - **Update** Resilience (update to low criticality does not impact high criticality)

Sharing & Isolation Technologies

- Modern CPUs are incredibly powerful (but not cheap)
- Processors, Containers and Hypervisors allow compute resources to be shared whilst providing isolation
- This is great for flexibility
- How does it stack up for robustness?

Regular Operating System Sharing (Processes)



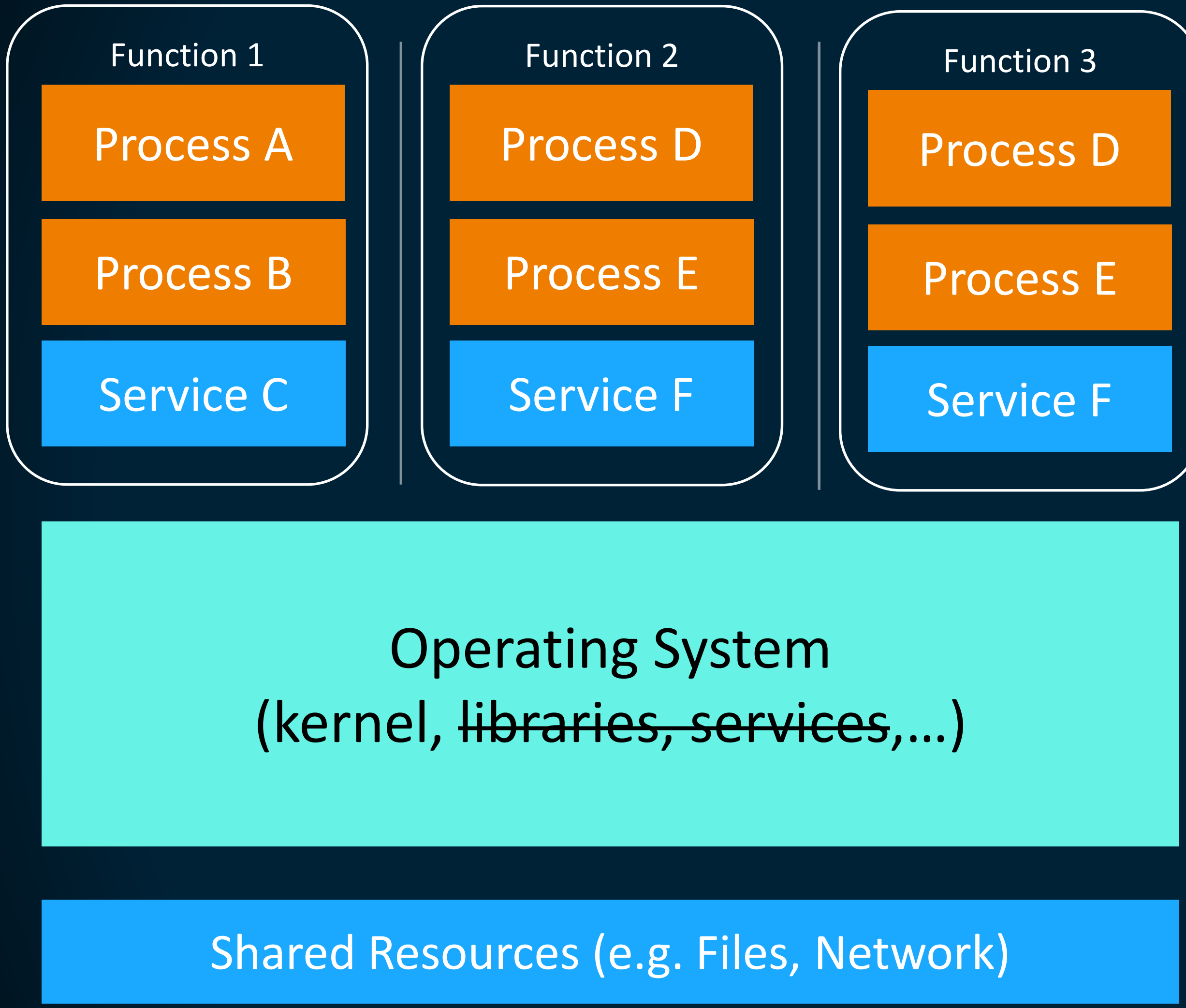
The operating system is shared

- It is responsible for isolating each process and for sharing of other resource
  - Processor (CPU) allocation
  - Physical memory allocation
  - File/Network/Peripheral access

Whilst the OS provides strong process isolation, it is far from perfect especially when shared services are considered

Most operating systems have limited isolation in terms of **Performance** and **Update.**

Containers



Containers are a brilliant solution to manage much of the software complexity in Linux

They allow a multi-process solution to be bundled and run against a known set of libraries

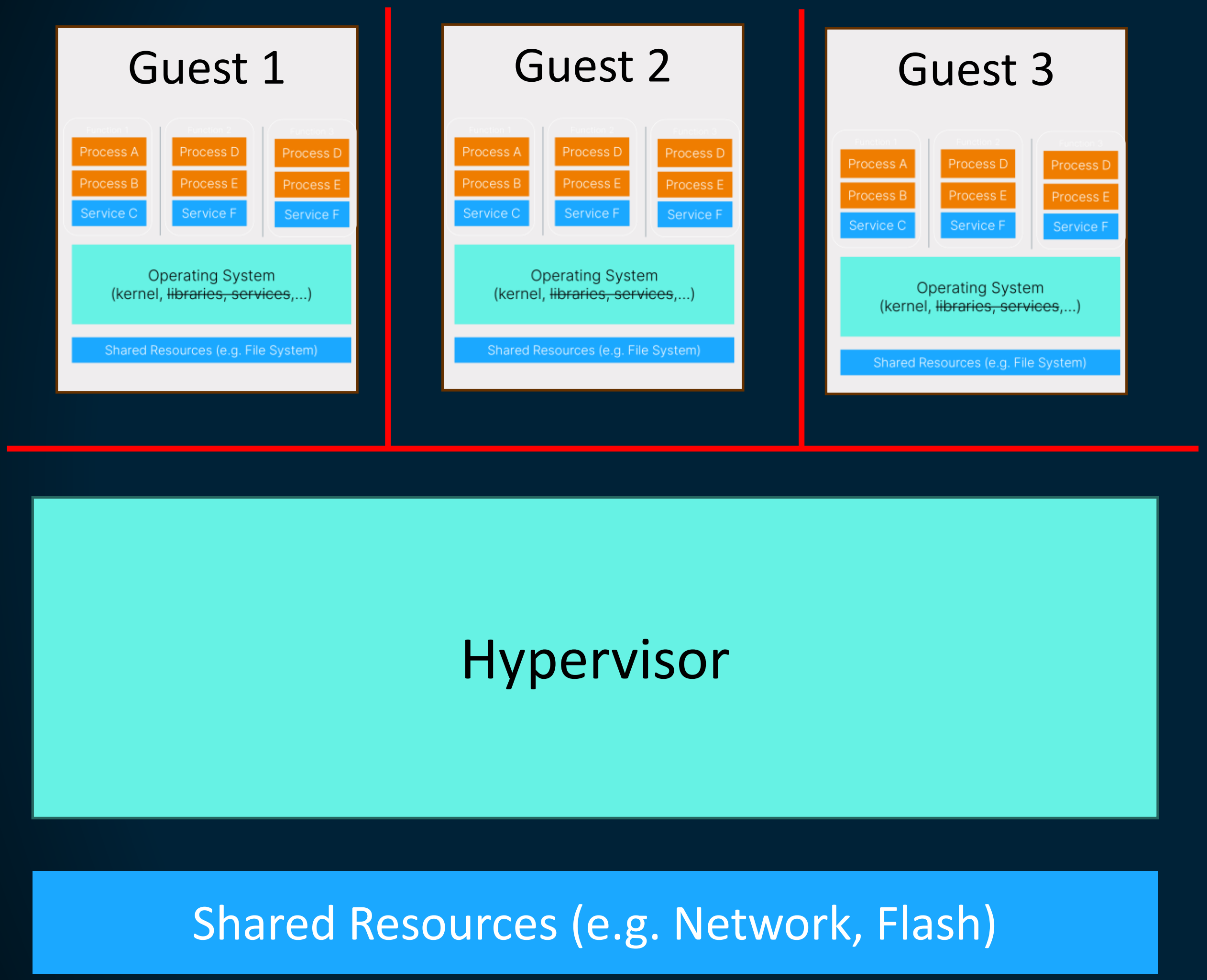
They also make it easier to update and manage software, improving isolation for **Update** and **Failure**

However, containers don't change the **security** or **performance** equations.

An attack on a process can still affect all other processes on the same host.

Containers are for management not security

Hypervisors



Hypervisors provide another layer of isolation and sharing

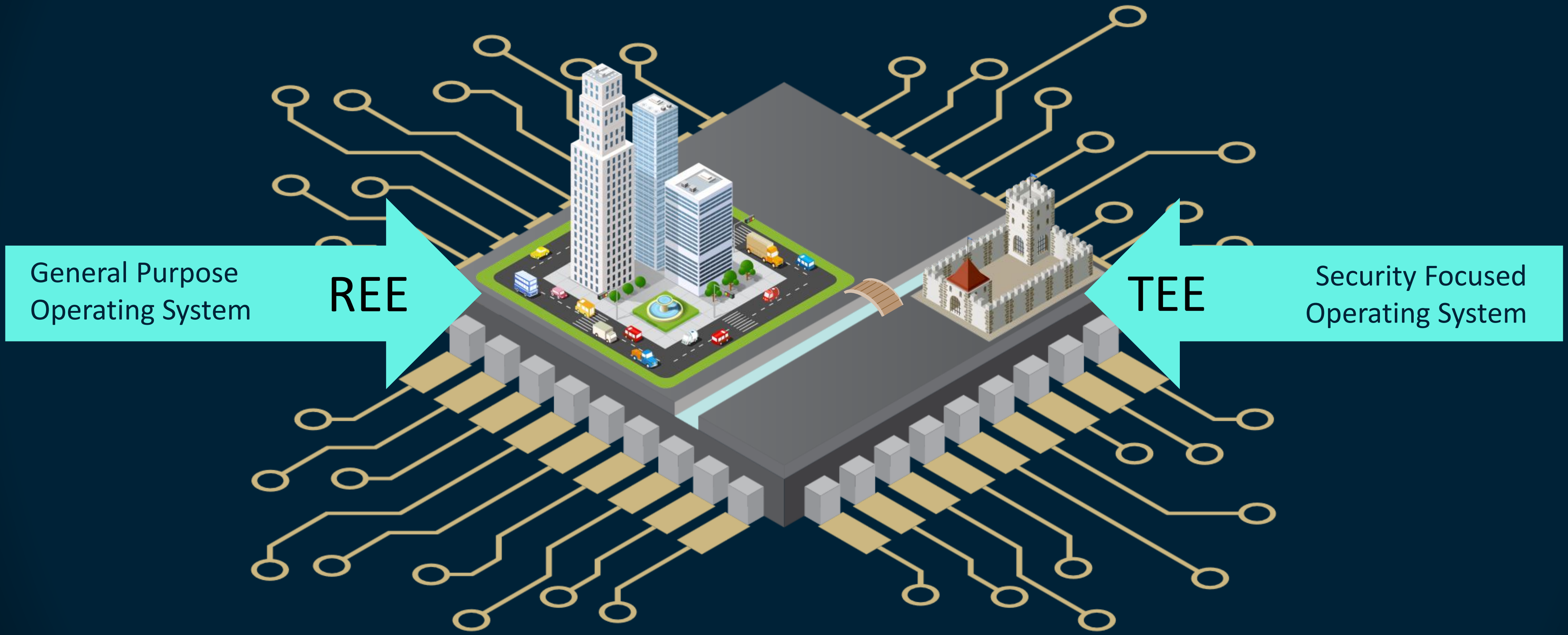
They isolate multiple operating systems (Guests) from each other, and allow each “virtualized” hardware, so that each acts as if it was on its own box.

Hypervisors must share (or allocate) cores, memory and peripherals to guests.

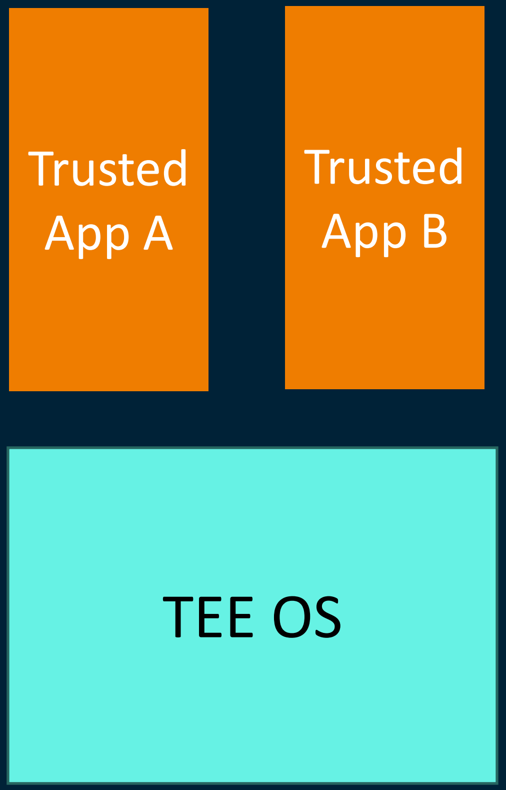
Memory is usually statically allocated, but *separation* Hypervisors also statically allocate cores. This means better isolation at the cost of overall performance.

Hypervisors are the accepted “best option” for providing strong isolation

Trusted Execution Environments



Comparing a TEE OS to a Regular OS



A TEE OS is conceptually very similar to a regular OS in terms of isolation

However, as TEEs are built for security the security isolation is **very good**

GlobalPlatform standardizes APIs and Security isolation – but says nothing about isolation related to **Performance, Failure or System Update.**

This is a new area of discussion within GlobalPlatform

A TEE OS is a service OS

Trusted Apps are used to provide trusted sub-function for REE applications rather than full ECU functions

Regular App

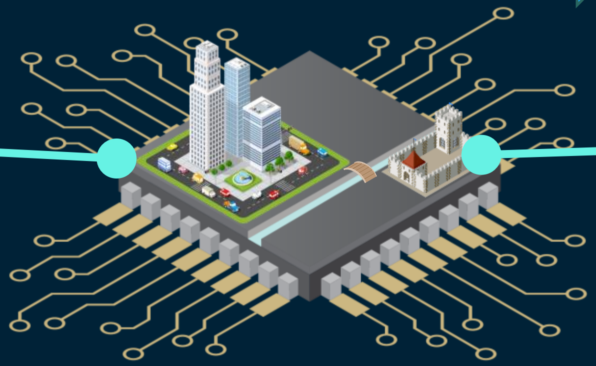
- Cryptography / Key Storage
- Protected media (DRM)
- Data Management
- Secure Biometrics…

Trusted Apps compete for resources

Trusted App

Trusted App

REE OS

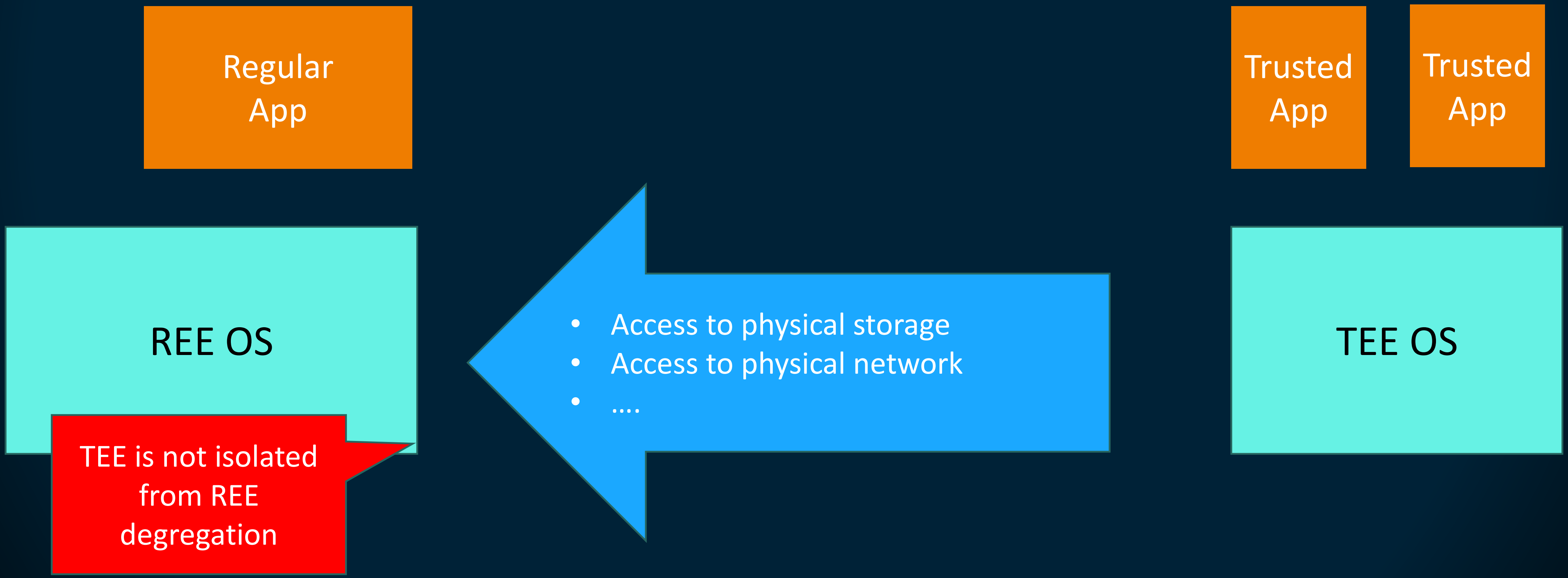


TEE OS

TEE may not provide resource isolation across TAs

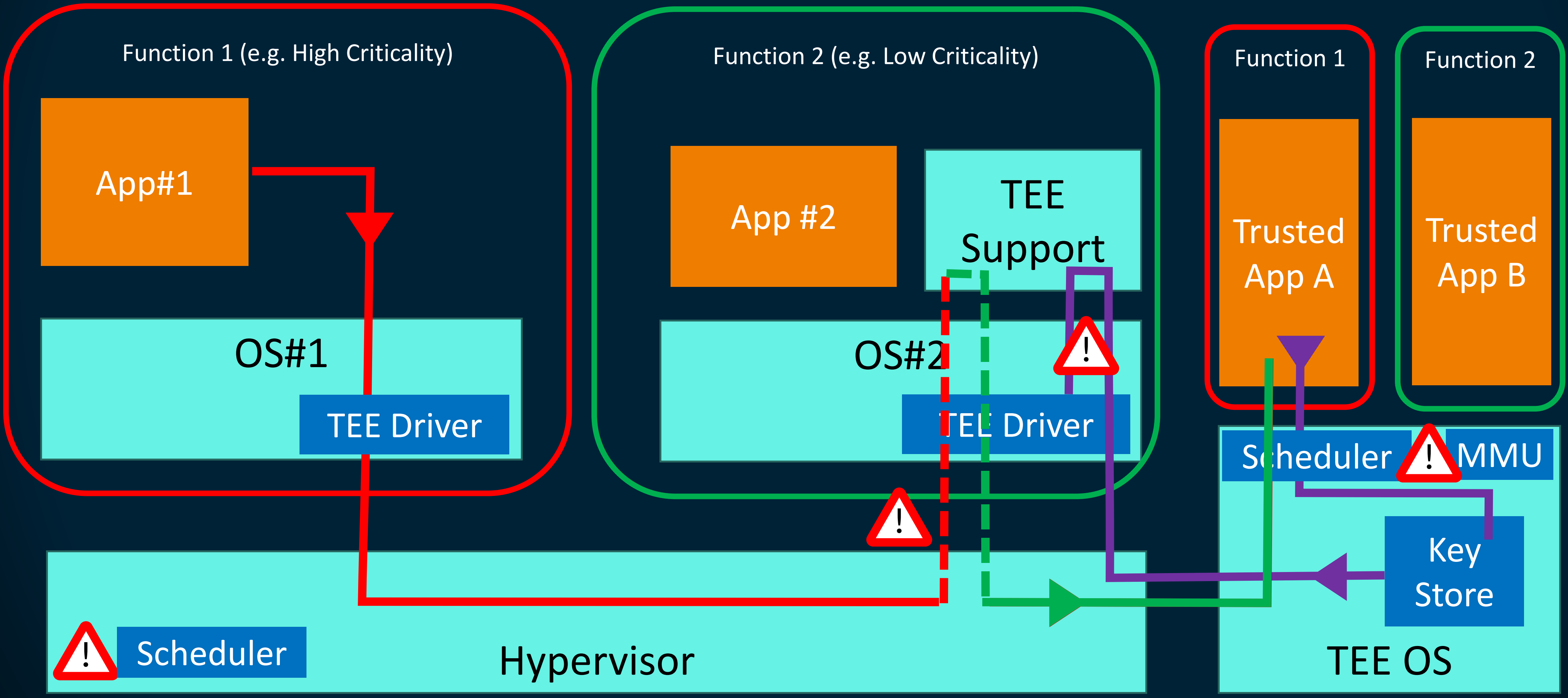
TEE OS usually relies on [a] REE OS

Features like storage or networking are usually delegate back to the REE



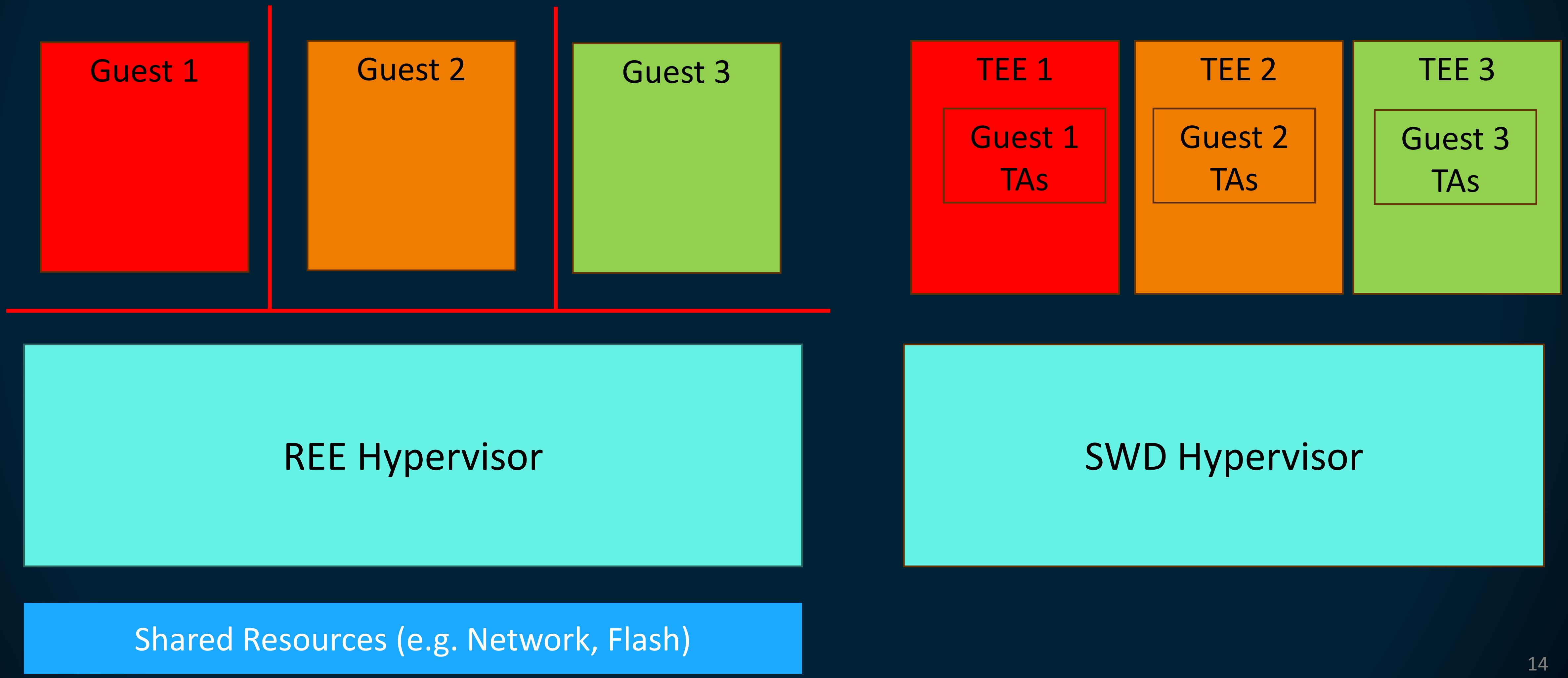
Hidden isolation challenges

- Priority Inversion; shared services; unexpected reliance on low criticality systems



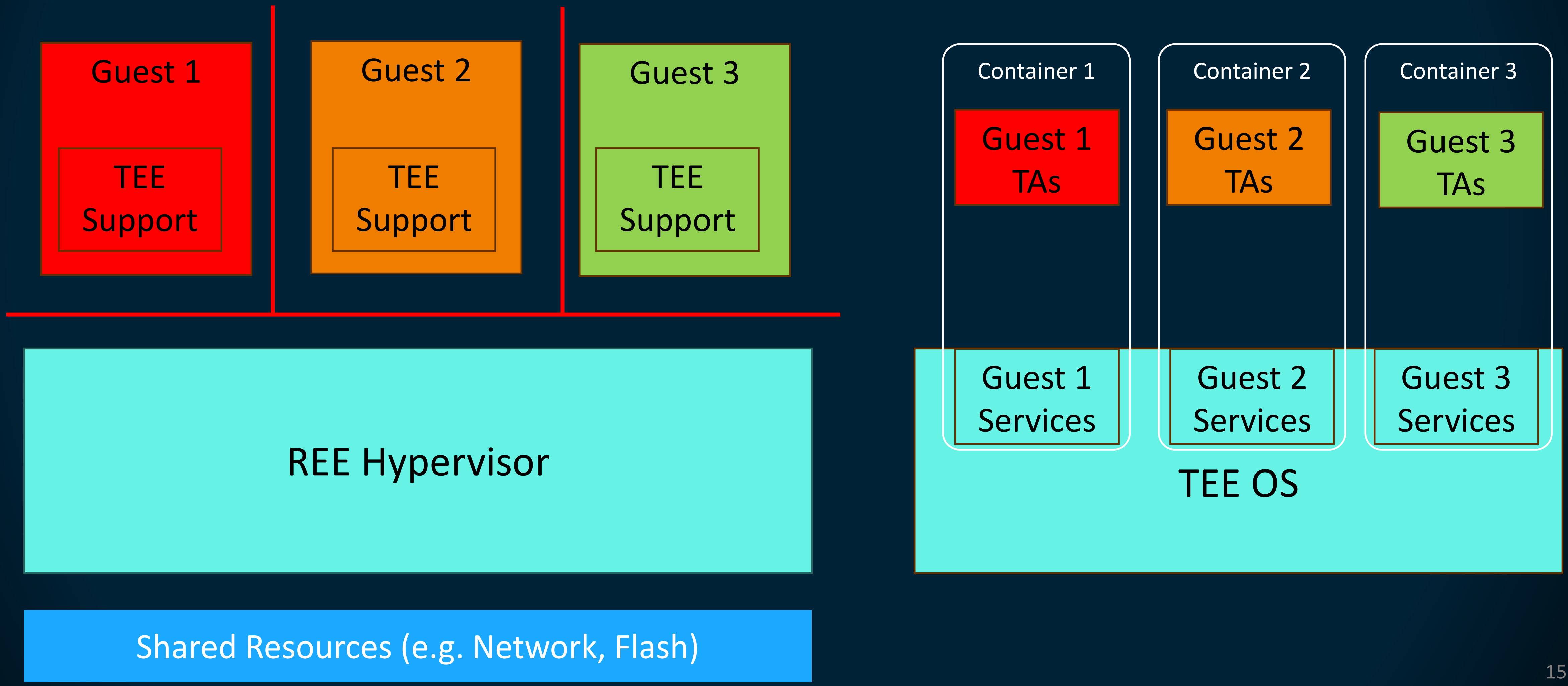
Meeting TEE Challenges (1)

- We can [in theory] introduce a hypervisor to secure world – but this is very heavyweight!



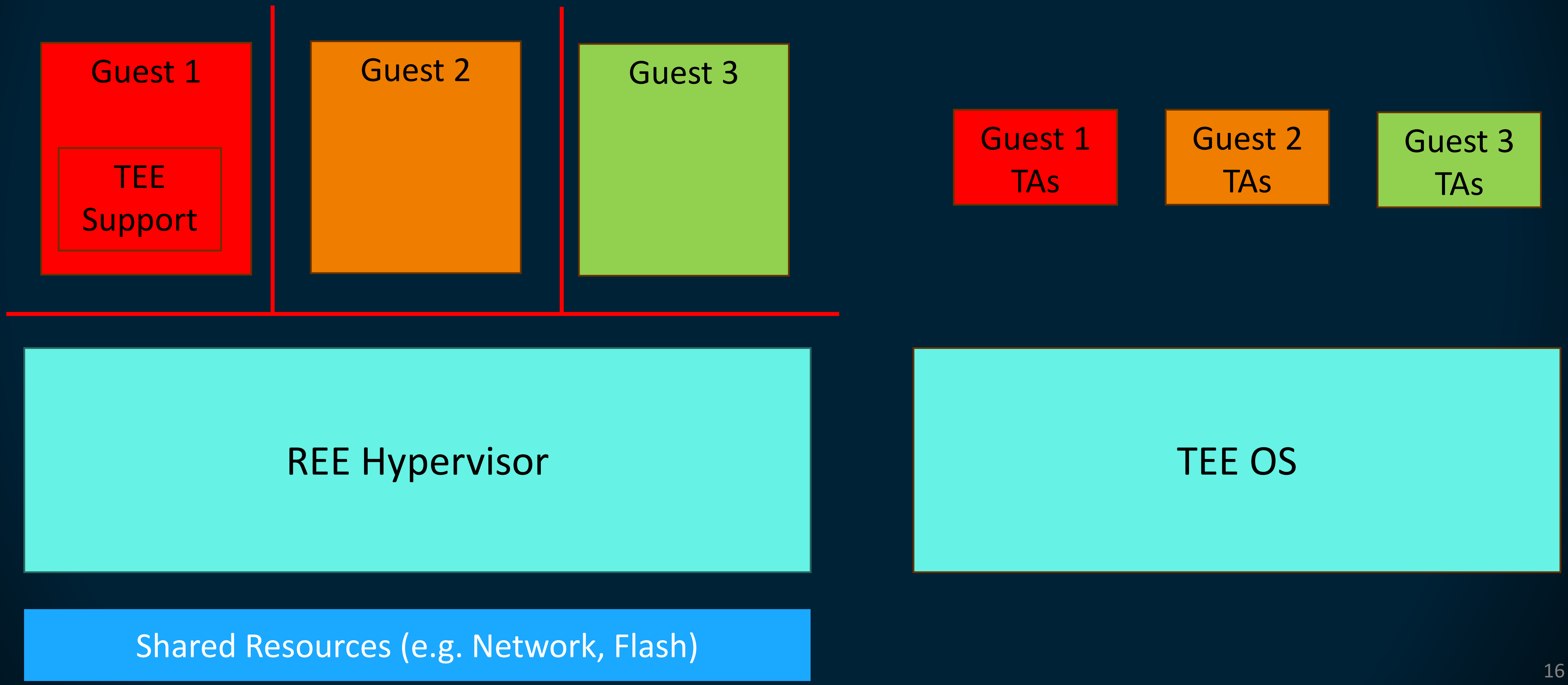
Meeting TEE Challenges (2)

- Could ‘containerizing’ the TEE and spreading support across guests solve isolation problems?



Meeting TEE Challenges (3)

- A common pragmatic option is to ensure the TEE support services are in a High Criticality guest



Summary

- Software Defined Vehicles need a combination of technologies
  - Containers
  - Hypervisors
  - TEEs

- The first-generation solutions statically allocated resources for different criticalities
  - Cores/Memory (Separation Hypervisors)
  - TEEs/Security Processors (Allocated to a single guest)

- There is a desire for more sharing to reduces costs / improve efficiency

- Different commercial solutions “may exist”
  - Not currently covered by standards
  - But GlobalPlatform is starting discussions