Different strategies for Mixed Criticality
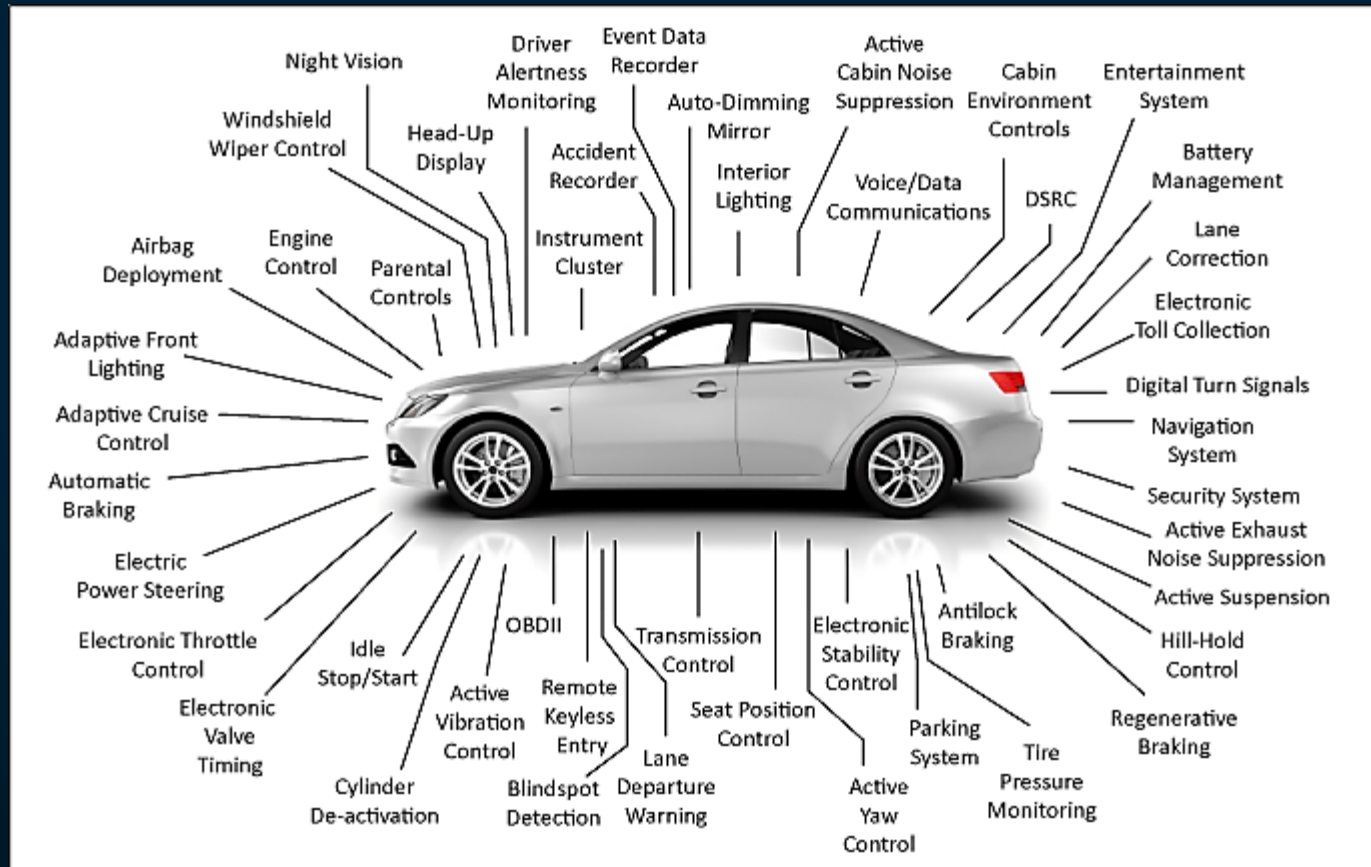
Dr Richard Hayton

Chief Strategy and Innovation Office, Trustonic Ltd.

Chair Automotive Task Force, GlobalPlatform

Chair Trusted Environments and Services Committee, GlobalPlatform

Isolation for Safety

- Safety critical services need to be robust

- The traditional approach is to build separate 'hardware' solutions for each safety problem

-

- The automotive industry is being challenged to adopt "software defined" approaches to reduce cost and increase flexibility

- How can we maintain robust systems in a software defined world

- And how does "security" change the picture?

Software Defined Vehicles

Software Applications

(Commodity) Software Platform

(Commodity) Hardware Platform
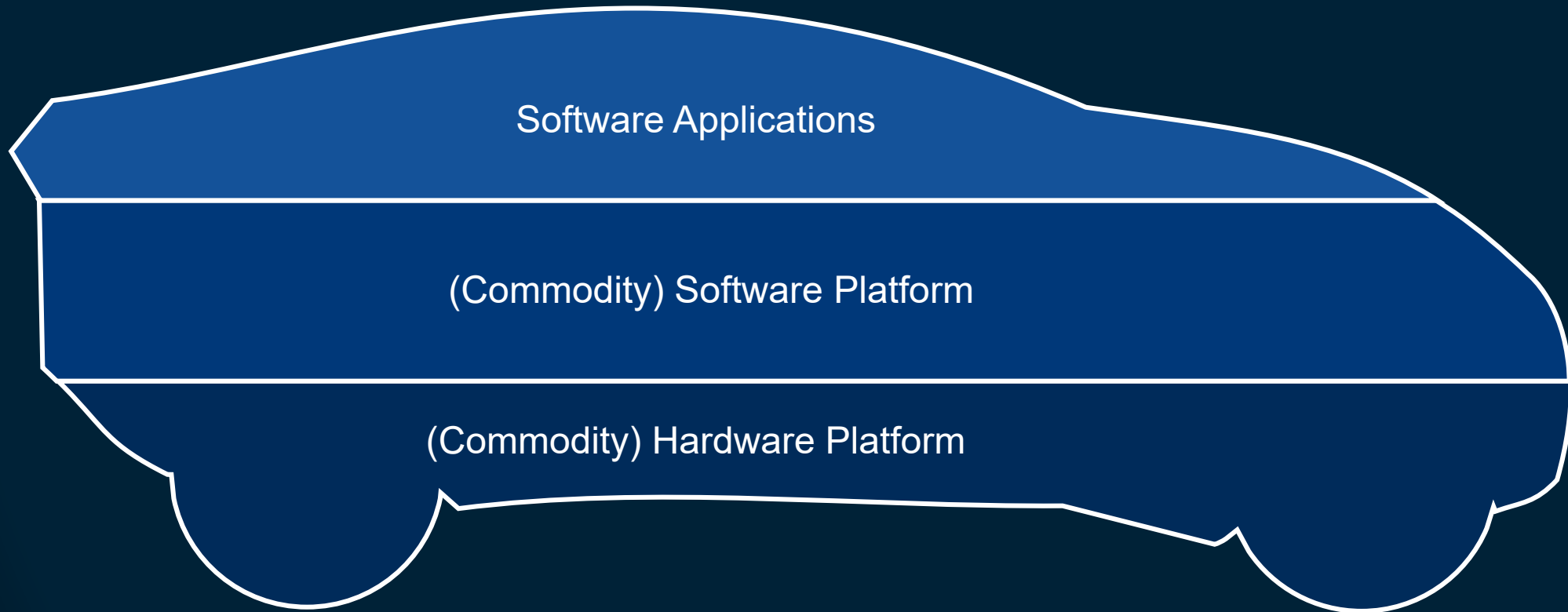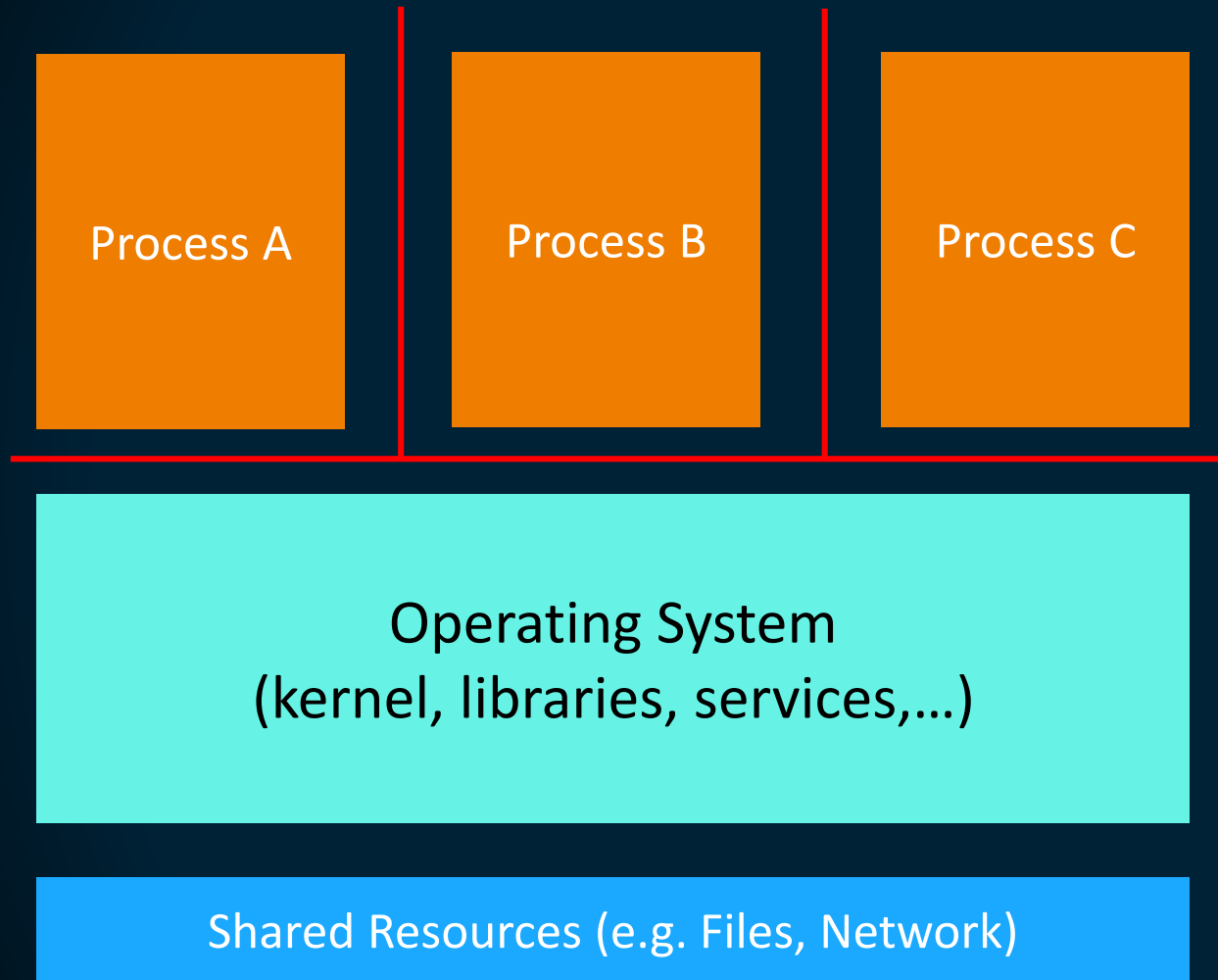
Robustness Needs for Mixed Criticality
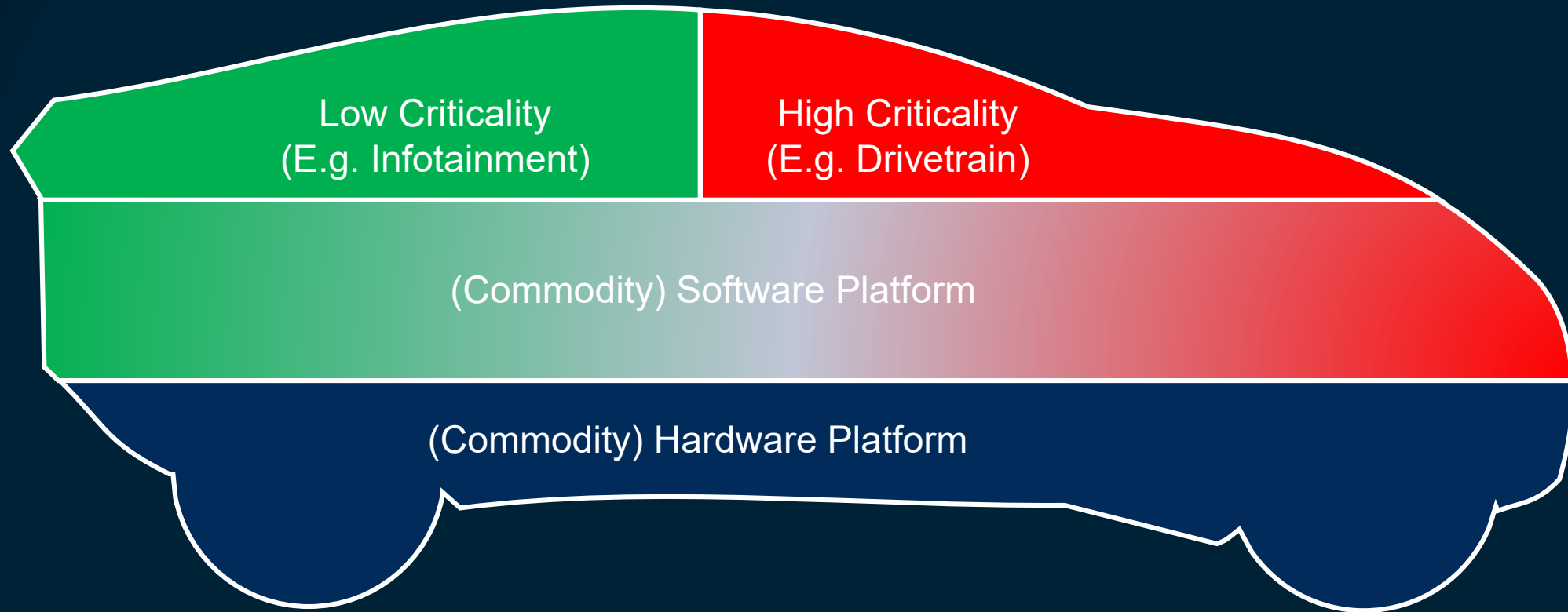
- - **Failure** (failure of low criticality does not impact high criticality)
- - **Performance** (degradation of low criticality does not impact high criticality)
- - **Update** Resilience (update to low criticality does not impact high criticality)
- - **Security** (attack on low criticality does not 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?

Containers & Processes

Hypervisor

Regular Operating System Sharing (Processes)

| Process A | Process B | Process C |

Operating System
(kernel, libraries, services,…)
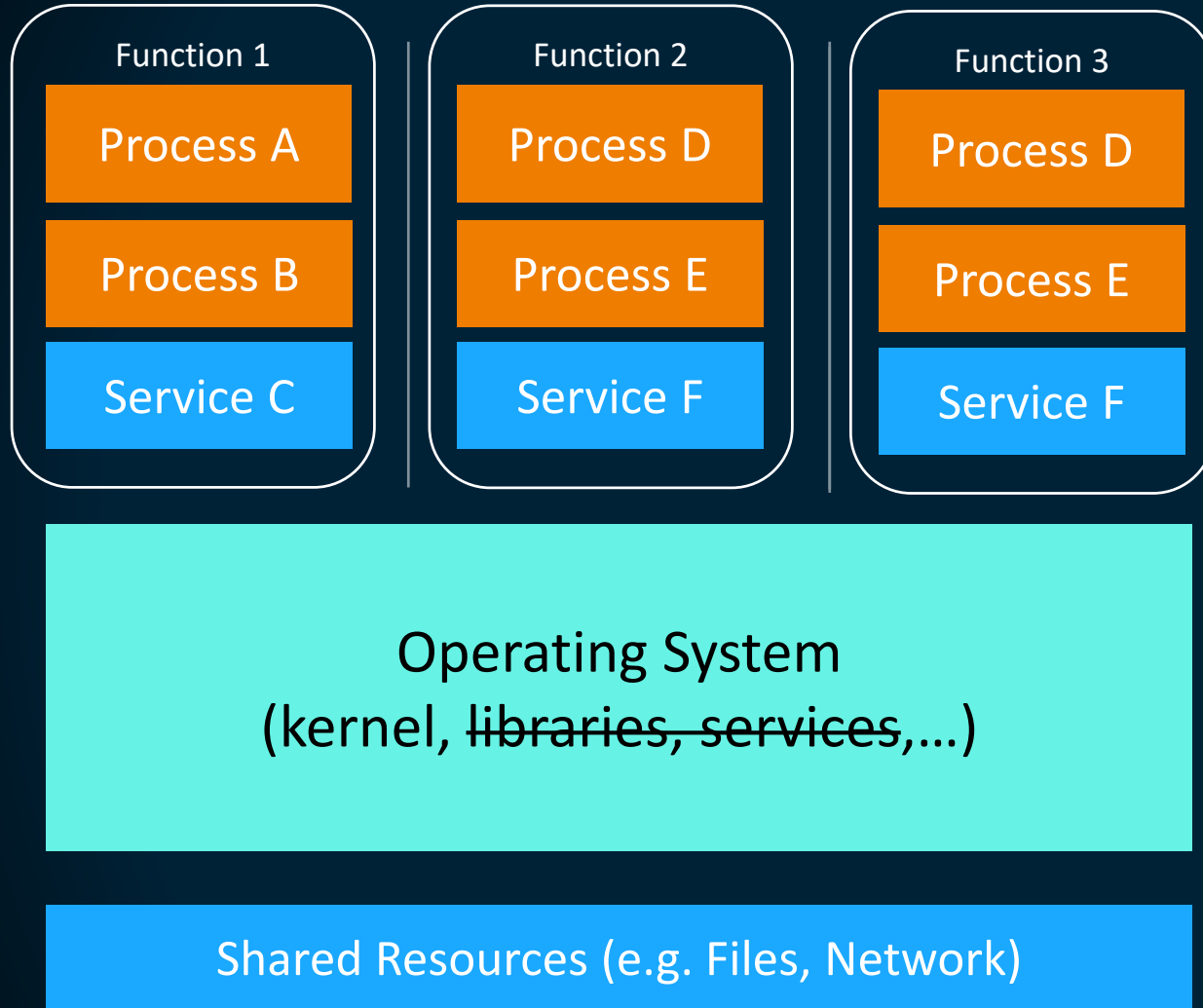
Shared Resources (e.g. Files, Network)

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 "general purpose" operating systems have limited isolation in terms of **Performance** and **Update.**

Containers

Function 1	Function 2	Function 3
Process A	Process D	Process D
Process B	Process E	Process E
Service C	Service F	Service F

Operating System
(kernel, ~~libraries, services~~,…)

Shared Resources (e.g. Files, Network)

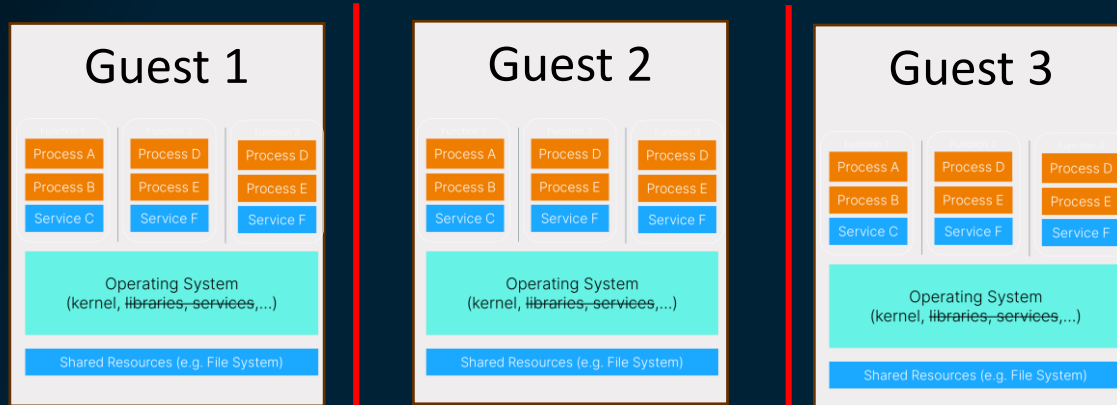Containers are a brilliant solution to manage much of the software complexity in Linux

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

Guest 1

Process A
Process B
Service C

Process D
Process E
Service F

Process D
Process E
Service F

Operating System
(kernel, ~~libraries, services,~~...)

Shared Resources (e.g. File System)

Guest 2

Process A
Process B
Service C

Process D
Process E
Service F

Process D
Process E
Service F

Operating System
(kernel, ~~libraries, services,~~...)

Shared Resources (e.g. File System)

Guest 3

Process A
Process B
Service C

Process D
Process E
Service F

Process D
Process E
Service F

Operating System
(kernel, ~~libraries, services,~~...)

Shared Resources (e.g. File System)

Hypervisor

Shared Resources (e.g. Network, Flash)

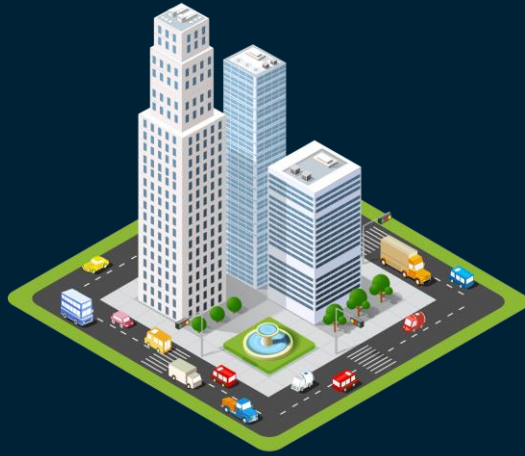Hypervisors provide another layer of isolation and sharing

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 s*eparation* 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
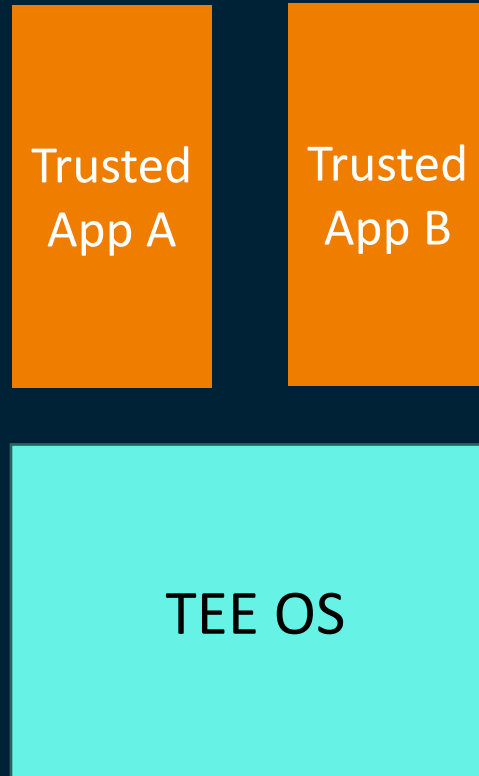
Isolation for Security

- Security sensitive services need to be isolated against both errors and attacks

- It is very hard to sufficiently isolate security systems using the techniques described so far

- Security services are therefore usually built "another way"
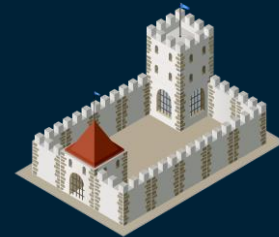
Trusted Execution Environments

General Purpose Operating System **REE** → ← **TEE** Security Focused Operating System
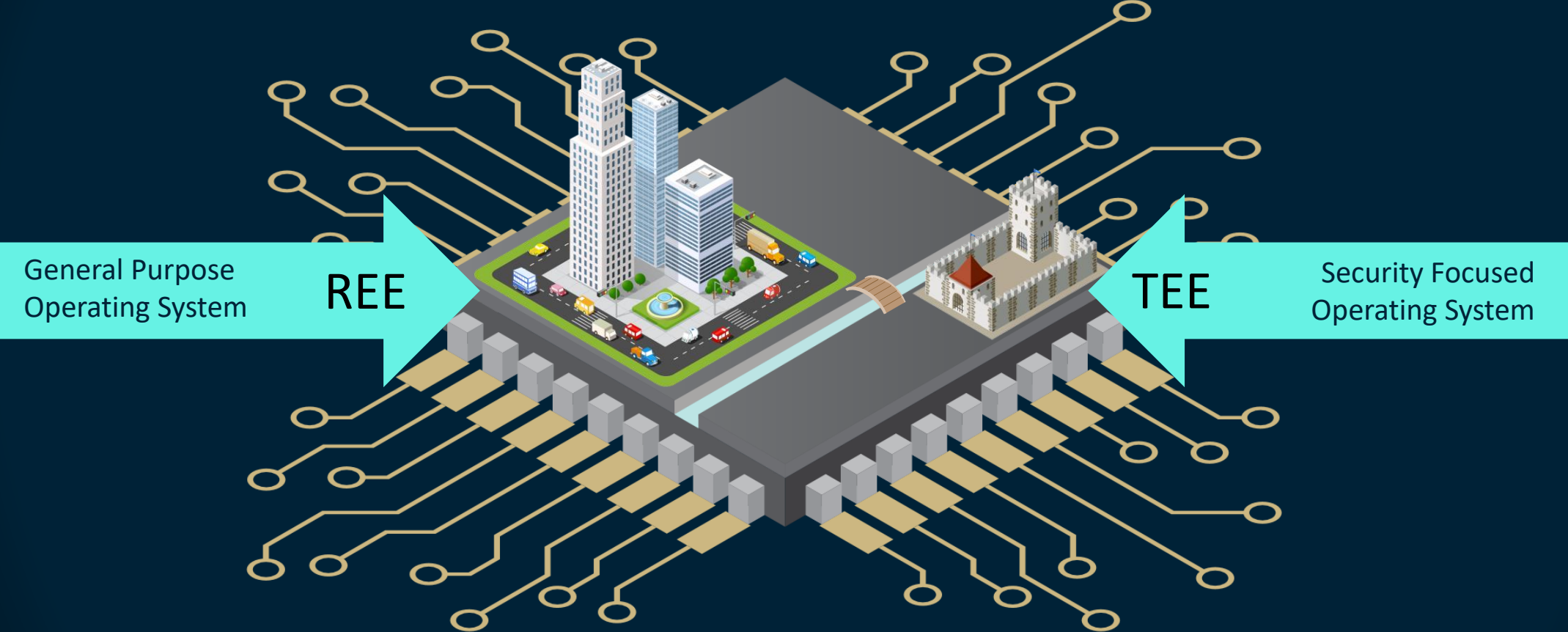
Comparing a TEE OS to a Regular OS

Trusted App A

Trusted App B

TEE OS

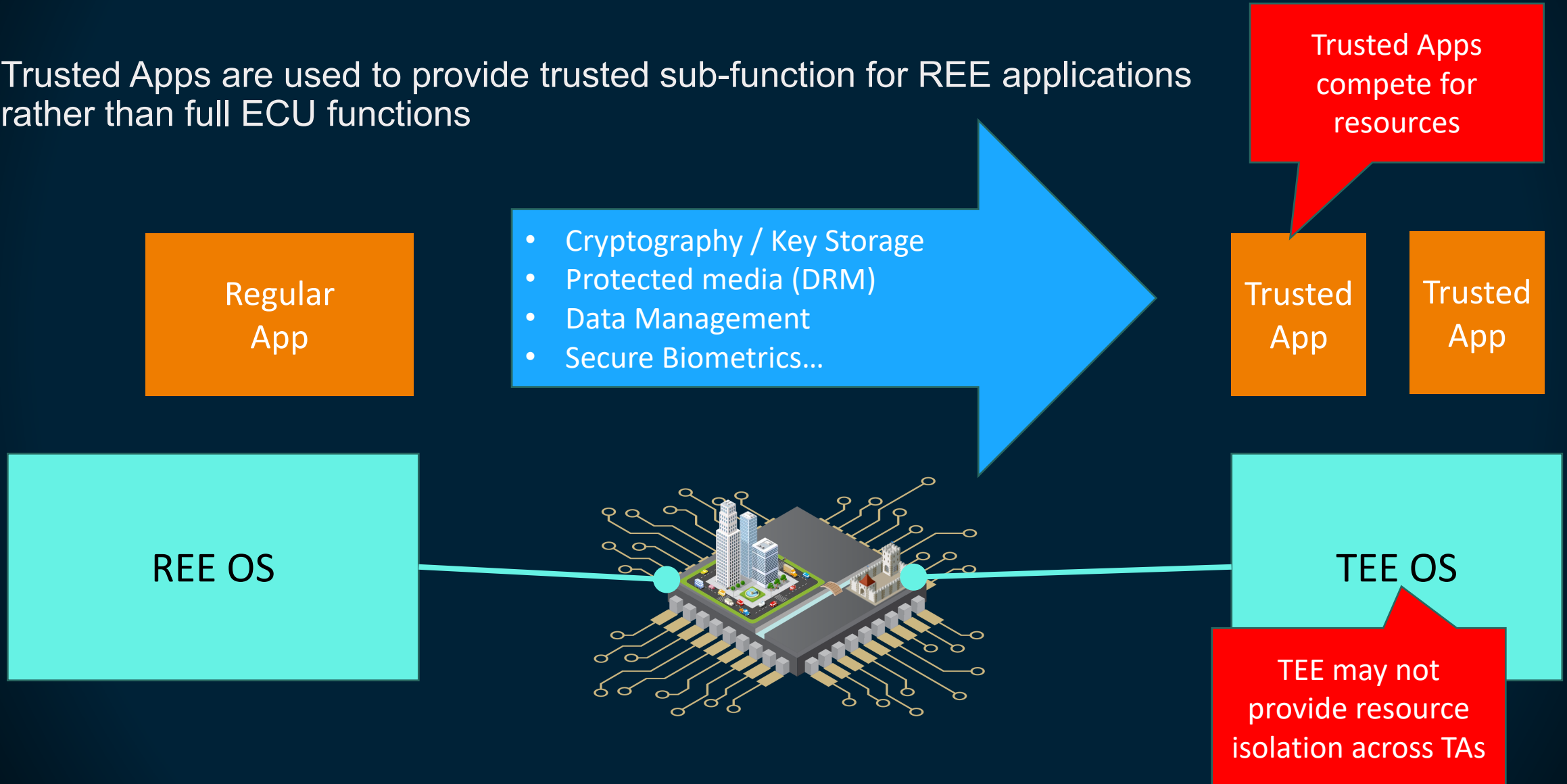A TEE OS is conceptually very similar to a regular OS in terms of isolation

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 effect on external systems (all critical for safety).**

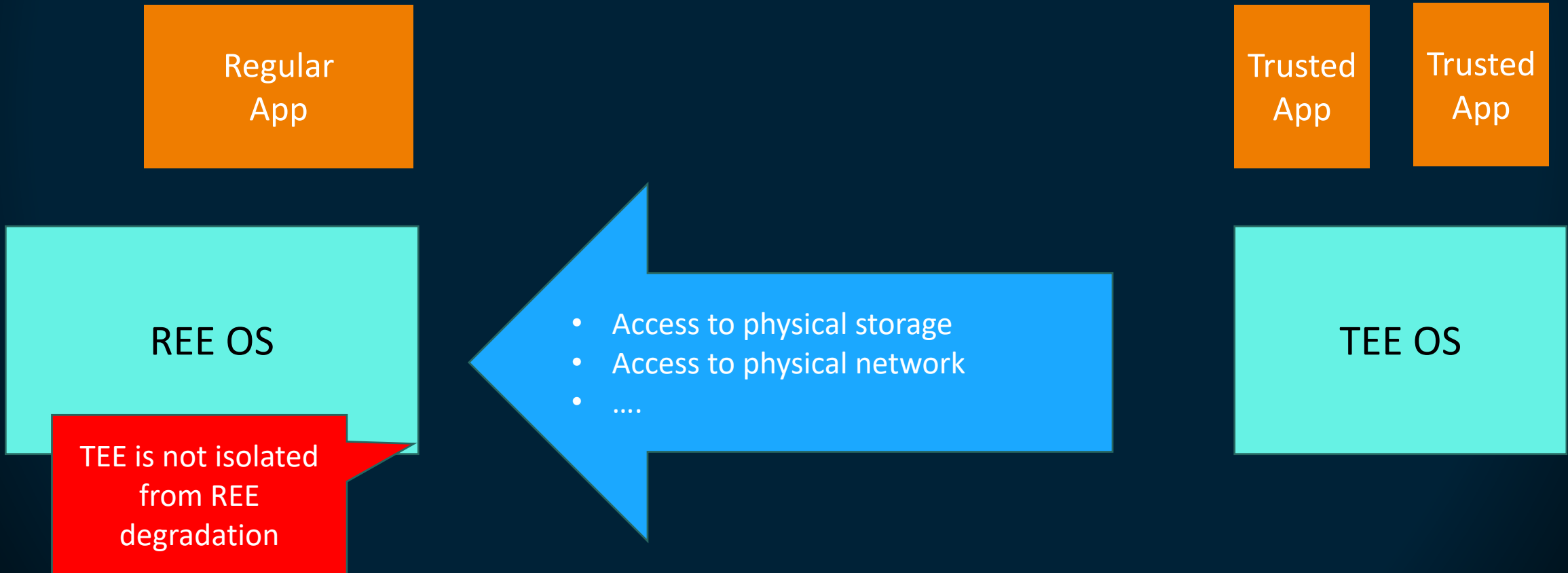This is a new area of discussion within GlobalPlatform

A TEE OS is a service OS

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

Trusted Apps compete for resources

Regular App

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

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

Regular App

Trusted App

Trusted App

REE OS

TEE is not isolated from REE degradation

- Access to physical storage
- Access to physical network
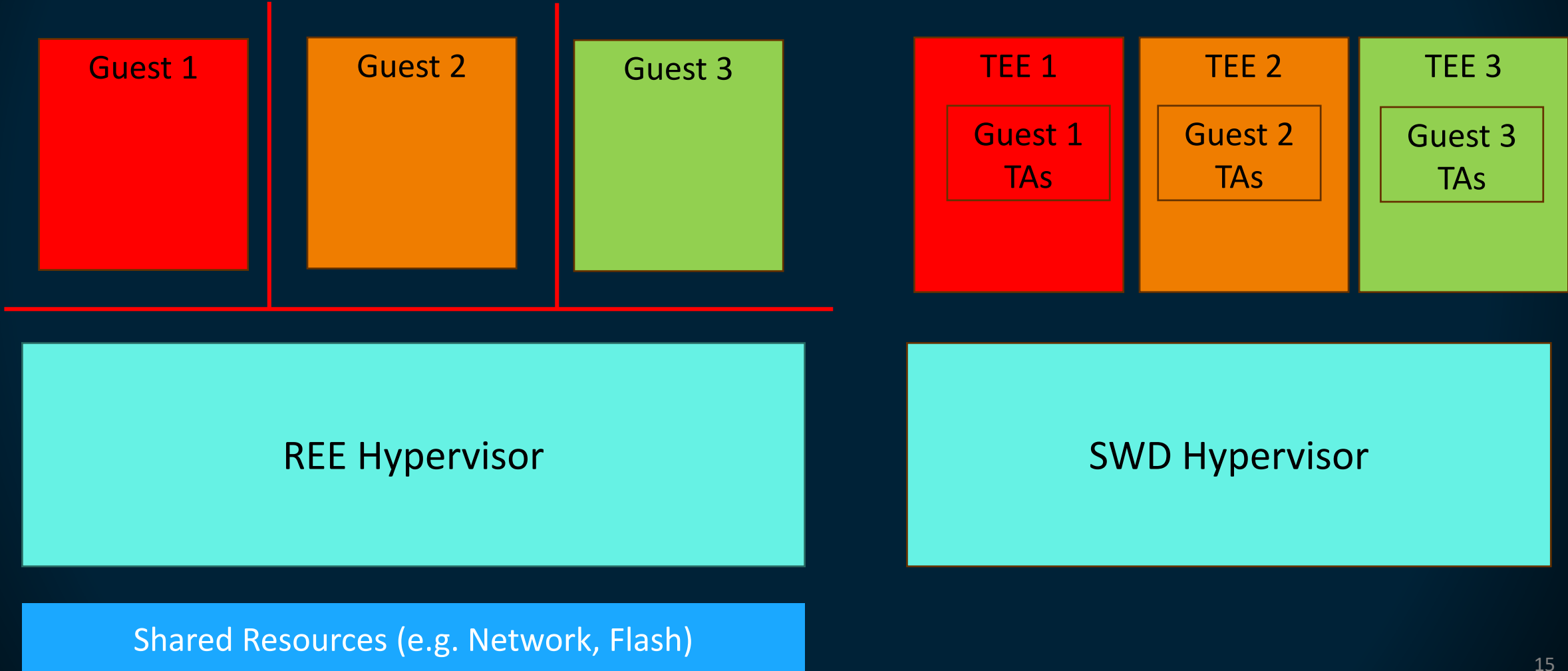- ….

TEE OS

13

Hidden isolation challenges

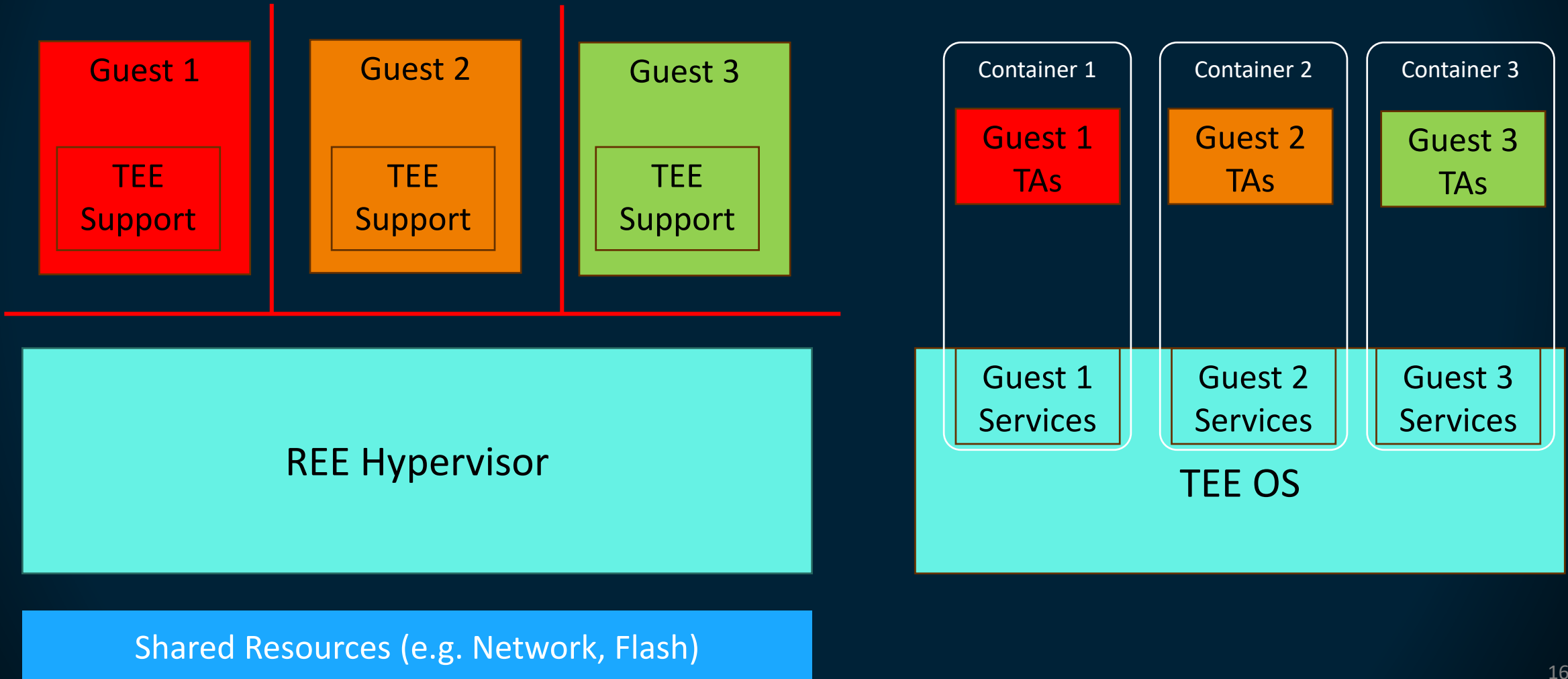- Priority Inversion; shared services; unexpected reliance on low criticality systems
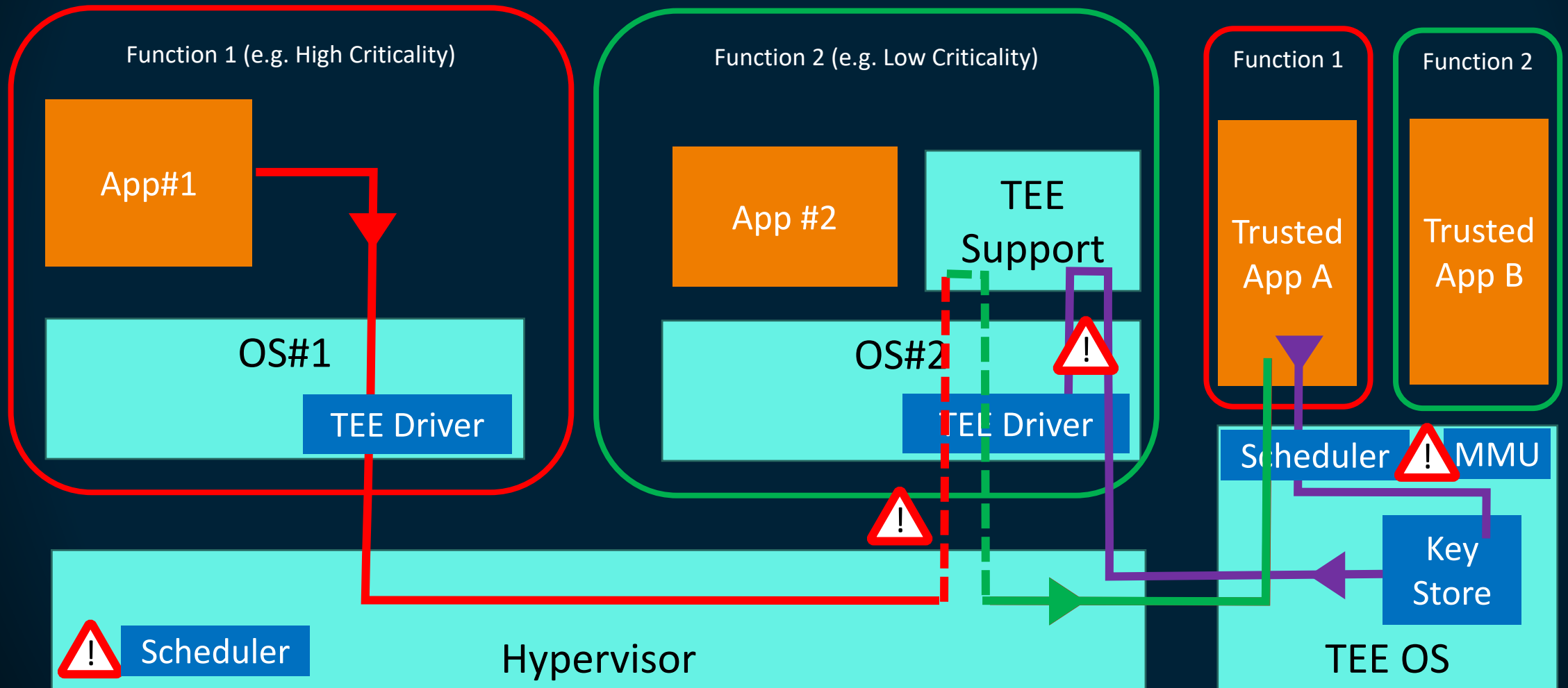
Meeting TEE Challenges (1)

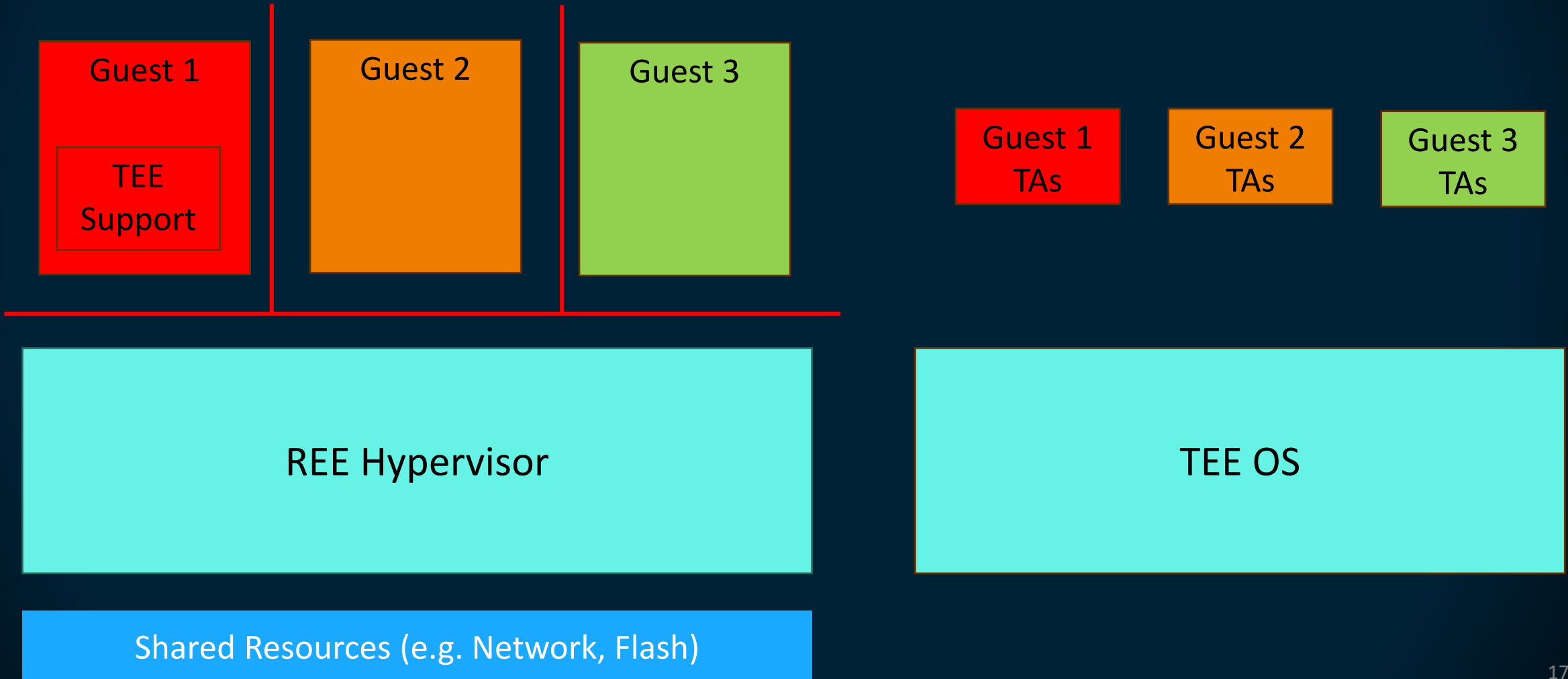- We can [in theory] introduce a hypervisor to secure world – but this is very heavyweight!

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