A hypervisor makes it possible to build *mix-criticality systems* and can integrate safely and securely:

- Software that is developed according to different quality standards:
  - Related to safety (different ASIL levels)
  - Related to security (different levels of trustworthiness, isolating a minimal trusted computing base)
  - Related to reliability

- Software that has different real-time (e.g. RTOS and generic OS) and boot-time requirements

In addition, a hypervisor:

- Enables the use of optimally suited operating systems and frameworks (Linux, Android, AUTOSAR, RTOS)
- Isolates faults (safety, reliability) and attacks (security) and supports ASIL decomposition
- Enables modular development and modular software updates
- Supports software reuse across platforms and integration of legacy software
Design Patterns for Safety

- Design the system to reduce the amount of software that must be developed according to ASIL processes and maximize as much software as is safe to not require development according to ISO26262:
  - use a hierarchical watchdog approach
  - use checking/monitoring strategies wherever possible
  - use E2E protection in the communication wherever possible
- The Hypervisor must provide freedom from interference to separate the VMs developed to different safety standards up to the requirements of ISO 26262 ASIL-B
Design Patterns for Security

- Enforce the MILS (Multiple Independent Levels of Security) architecture
- Most systems will want to use a layering of security technologies, whereby the upper domain:
  - provides better security assurance than the lower domain
  - manages the communication with the previous domain
- Hypervisor ensures strong separation between general purpose VMs (with external interfaces) and hardened VMs and provide secured access to trusted OS
- Hypervisor supports the SoC-specific hardware security features (such as Secure Boot, TrustZone,...)
- Hypervisor supports secure and modular software update to patch vulnerabilities in the field
Separation

• On most SoC’s several devices have DMA capabilities.
• To use such devices
  o safely (to ensure freedom from interference between the VMs)
  o securely (so that an attacker cannot abuse a DMA-capable device to break separation)
• the SoC needs the right hardware mechanisms (such as the presence of an SMMU or IO-MMU).
• In COQOS, the SMMU Controller is located in a VM (and can be integrated with the System Supervisor)
• **The availability of an SMMU Controller is SoC-specific**
Integrate Instrument Cluster and In-vehicle Infotainment functionality on a single device to save cost and provide an integrated driver experience.

Hypervisor separates functions with different requirements on real-time behavior and functional safety.

The displays must share information from different functions:
- infotainment
- real-time driver information
- safety-critical information

The system must interact with an AUTOSAR-based vehicle network.
an IC Client component is rendering *all* graphical elements – including safety-critical graphical elements.

an **independent** IC Guard component

- verifies the rendering of the safety critical graphical elements in time and in contents and
- hosts all safety relevant functions

an independent System Supervisor component supervises the computational fitness of the IC (hierarchical watchdog).
Mechanisms for device sharing

**in Hypervisor**

- Only used for UART (optionally)
- not recommended for other devices as the Hypervisor is minimalistic.

Example: UART

**device with virtualization support**

- COQOS supports this when the SoC hardware supports virtualized devices
- Recommended wherever the hardware supports it, as it tends to give the best performance and separation

Example: GPU on RCAR-H3

**low-level client-server**

- Single driver in VM that acts as "server"
- Driver-specific sharing logic is needed.
- Other VMs use "virtual driver"
- Compromise between performance and flexibility

Example: shared block device

**distributed frameworks over VNET**

- Allows reuse of existing frameworks for distributed applications in a virtualized environment over VNET.
- Supports complex sharing semantics at the cost of more overhead

Example: NFS, PULSE AUDIO
Virtualized IO devices are available for desktop and cloud applications because everyone uses standardized interfaces (virtio, xen, vmware)

- Disk
- Network

Embedded devices lack the ecosystem that cloud providers build upon

Challenges for virtualized IO devices in automotive

- High effort of SoC specific device virtualization
- Multimedia device virtualization
- Low amount of reusable virtual devices
### Device Virtualization Technologies

<table>
<thead>
<tr>
<th>Technology</th>
<th>Description</th>
<th>Reusability</th>
<th>Platform independence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard library (or layer) virtualization (OpenGL, DRM, Android HAL …)</td>
<td>Implement hypervisor specific standard libraries</td>
<td>As long as the same hypervisor is used</td>
<td>As good as vendor interface</td>
</tr>
<tr>
<td>Virtio</td>
<td>Implement virtio based devices that follow either existing standards or specify new ones</td>
<td>Virtio support is available in Linux, Android and many other operating systems</td>
<td>Builds upon the kernel-userspace interface of Linux and allows large flexibility because the devices themselves make no assumption about the hardware</td>
</tr>
<tr>
<td>HV vendor custom</td>
<td>Develop virtual devices optimized to be used with a particular hypervisor</td>
<td>As long as the same hypervisor is used</td>
<td>Implementation specific</td>
</tr>
</tbody>
</table>

Trade-off between development effort, reusability, platform independence, availability and maturity
Virtio

- Virtio “De-Facto Standard For Virtual I/O Devices” (Russel 2008)
- Standardized since March 2016 (OASIS VIRTIO-v1.0)
- Virtio provides interfaces for many devices
  - Block Storage
  - Network
  - Console
  - GPU
  - Input (hid)
  - Crypto device
  - vSock
  - File Server (9pfs)
  - Many more in development (vIOMMU, etc.)
- Still missing pieces
  - Audio
  - Sensors
  - Media Acceleration Offload (VPU)