# Virtualization:
## Hypervisors, Containers and Device Abstraction

Randy Martin
Senior Technical Product Manager

# The Evolution of VIRTIO and QNX

**VirtIO**

**2016**

QNX engineering start shared graphics initiatives with Android.
Jaguar XJ demo @ CES2017

**2019**

OEMs start standardizing on native but custom Android

**2020**

Beginning of Google & QNX VirtIO engineering collaboration

**2021**

Google VirtIO initial implementation Cuttlefish & Android 11 = slow

Beta of automotive version of VirtIO "Trout"

**2022**

Official release of automotive version of VirtIO "Trout"

VirtIO components appear in production vehicles (e.g. VirtIO Video & VirtIO Sound)

**2024**

Next generation cockpits adopt full VirtIO in support of cloud, multi-SoC and global platform strategies

OEMs asking for standard Android not SoC custom Android

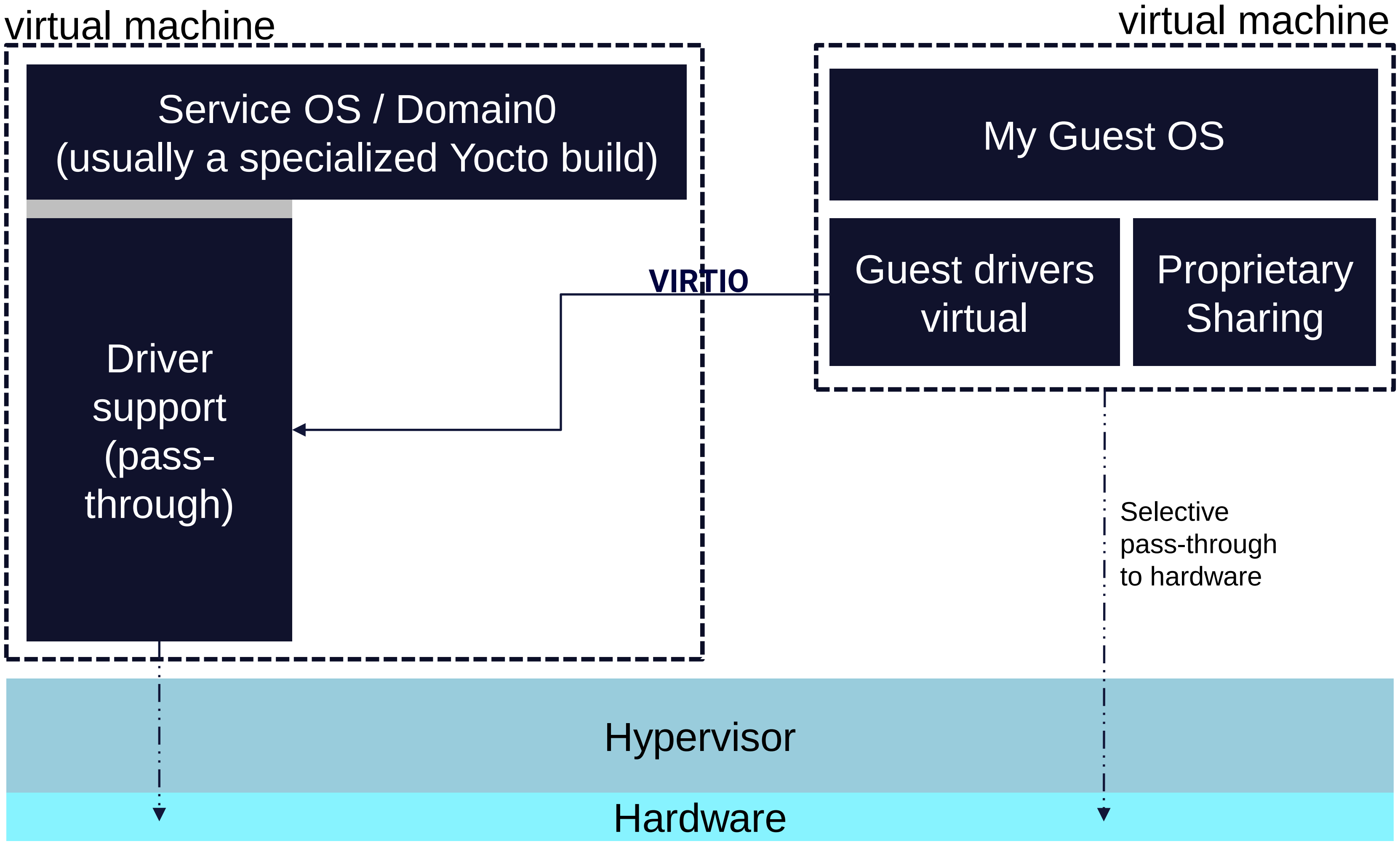
Google
STELLANTIS
VOLVO
QNX
... and more.

# Hypervisor Design is still the most important choice

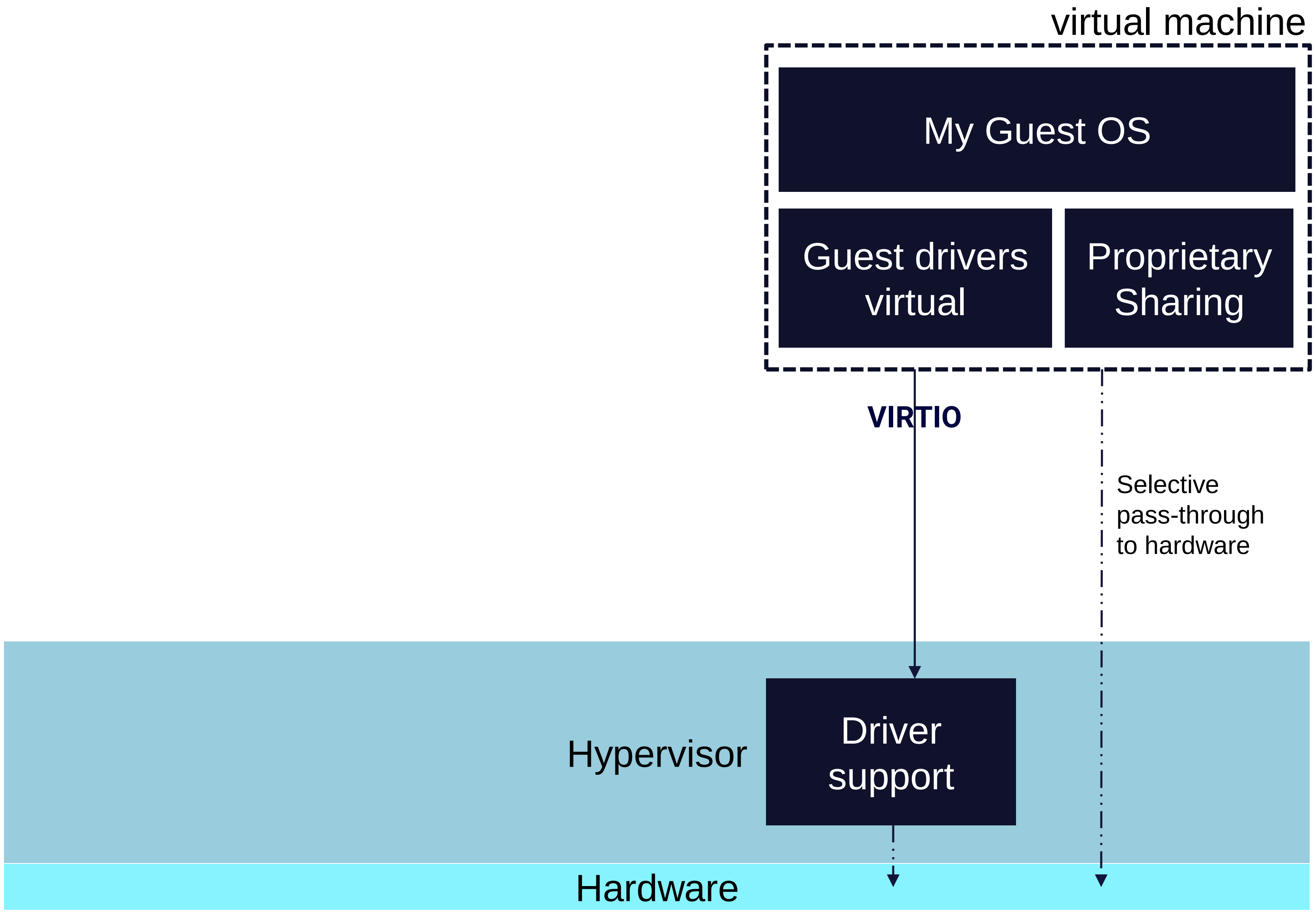
1. Service OS in its own Virtual Machine

2. Service OS in the Hypervisor Host

# Service OS exists in its own virtual machine



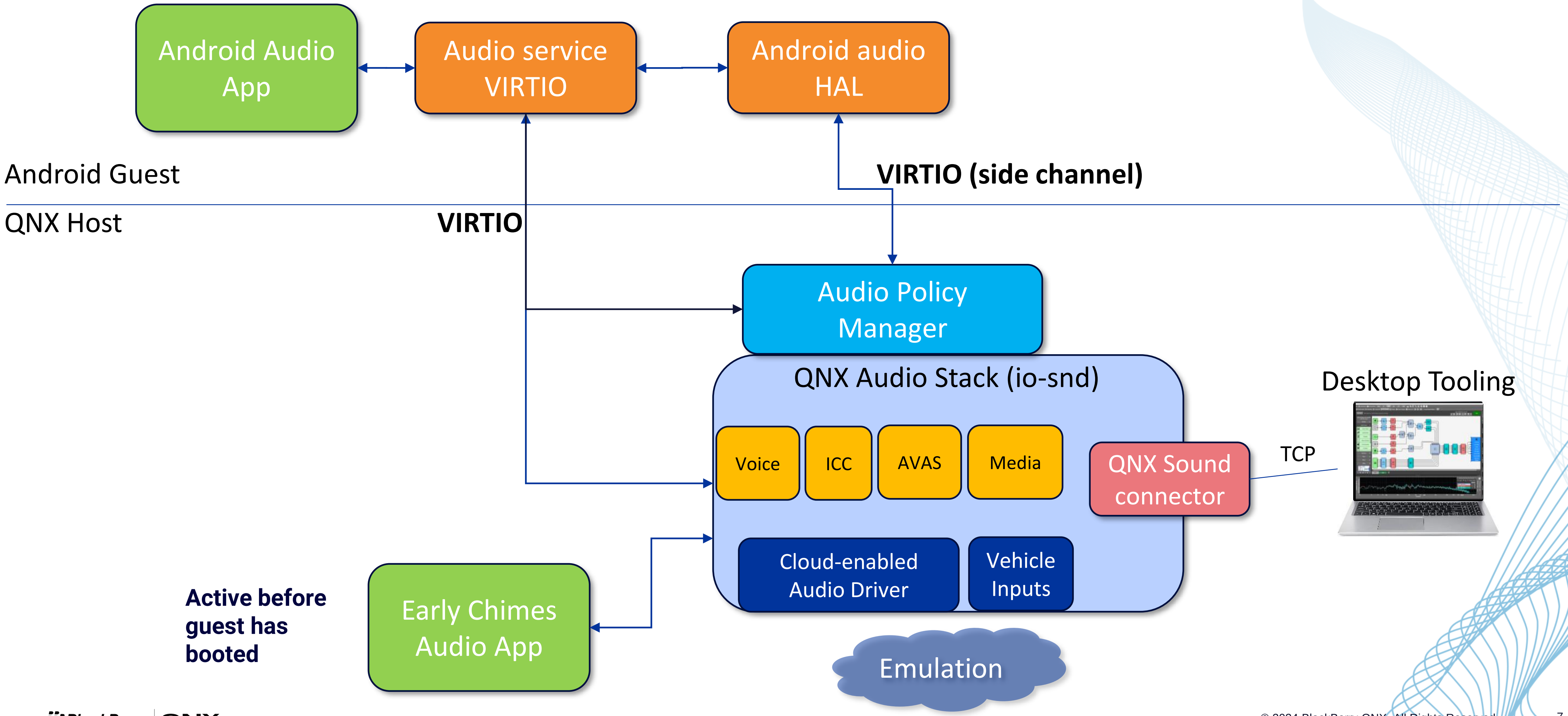
# Service OS exists in the Hypervisor Host



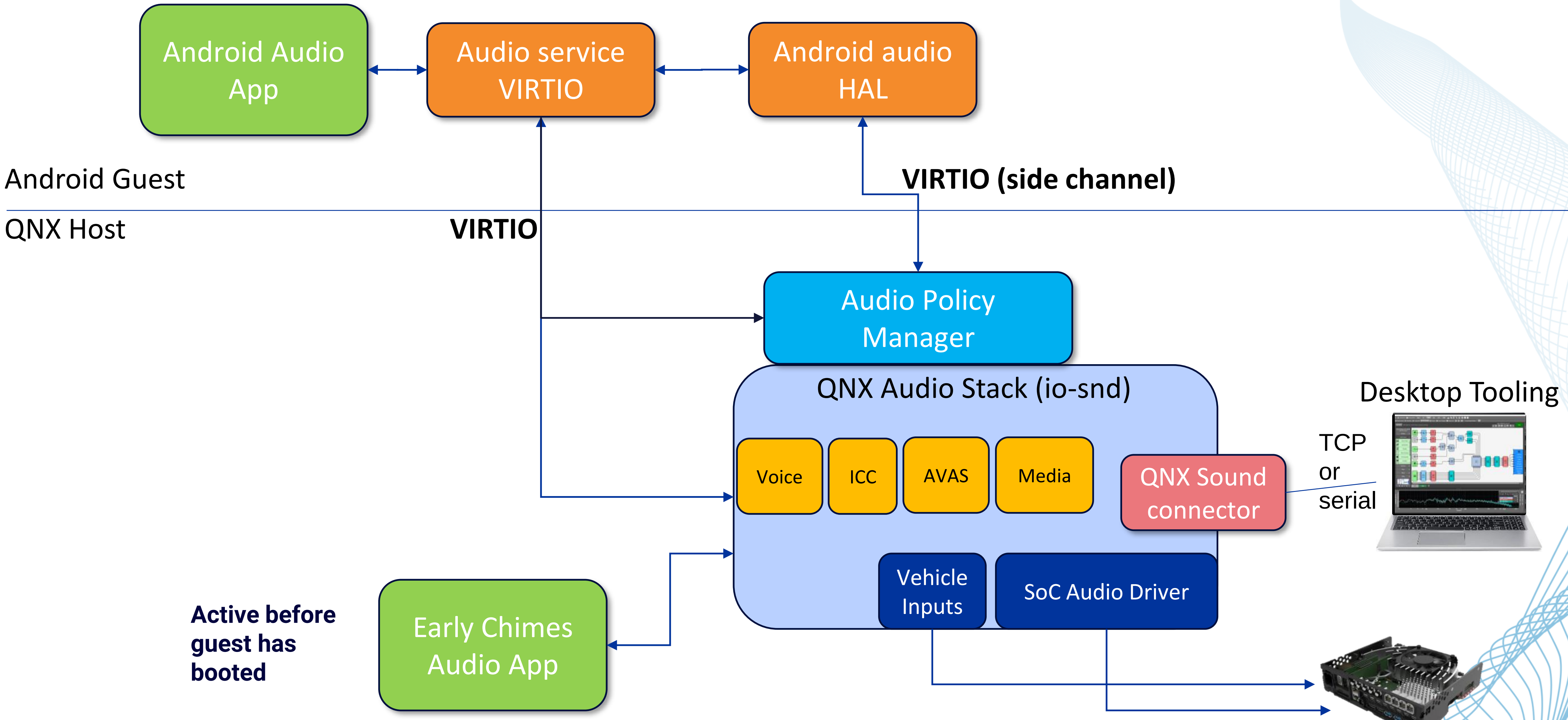
# Sharing Example: Audio Requirements (Acoustics)

1. Want audio available (chime) as soon as target boots ( < 1second)
2. Want Audio to be free of any noticeable affects when a barge-in or safety chime is needed *regardless* of what is currently playing
3. Ability to easily add in new features such as multi-zone voice capture, in-car communications, new media processing features such as Dolby Atmos etc.
4. Want to be able to move software between SoC vendors. Do not want to be tied to a specific DSP or hardware solution
5. Cloud-first design and test

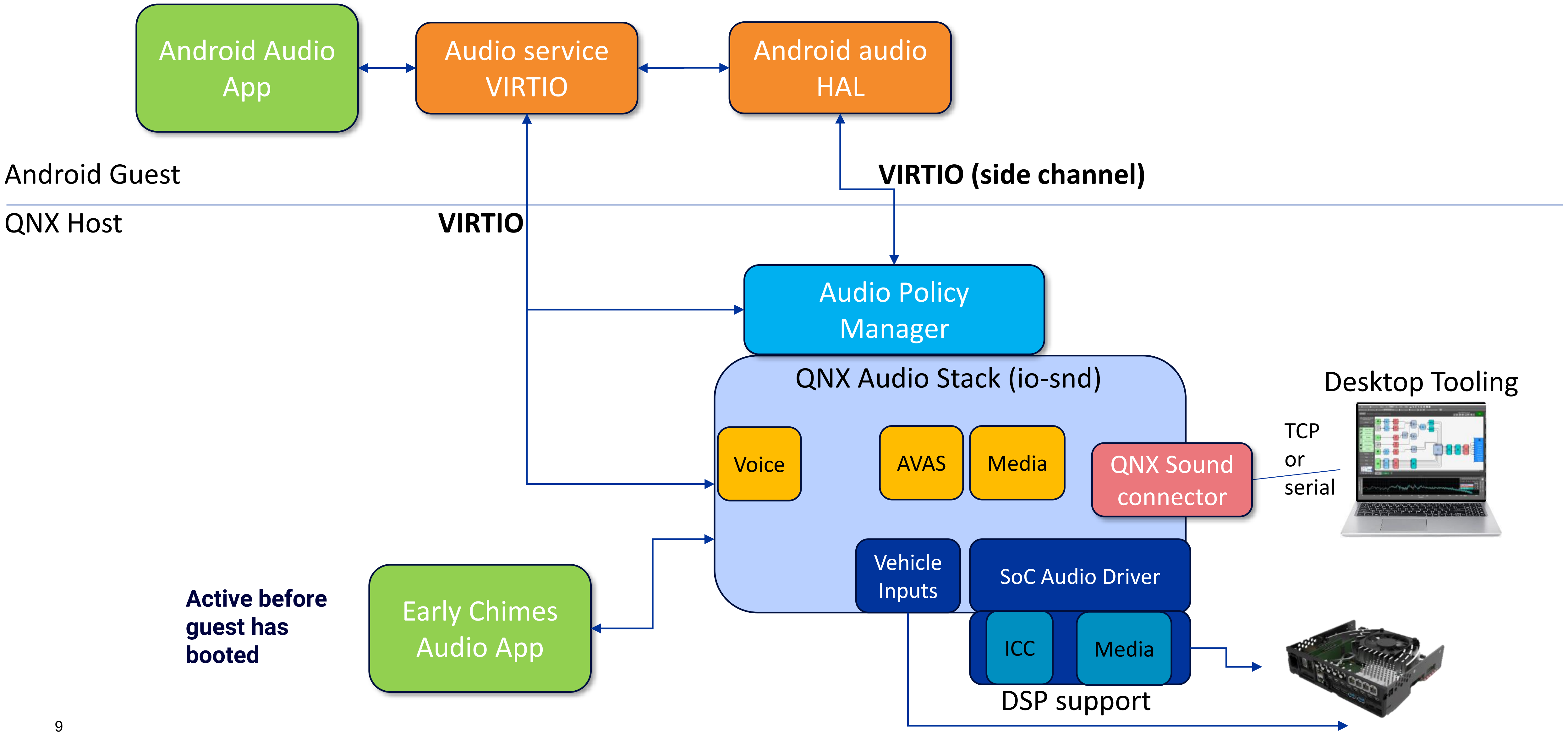
# Acoustics in the Cloud



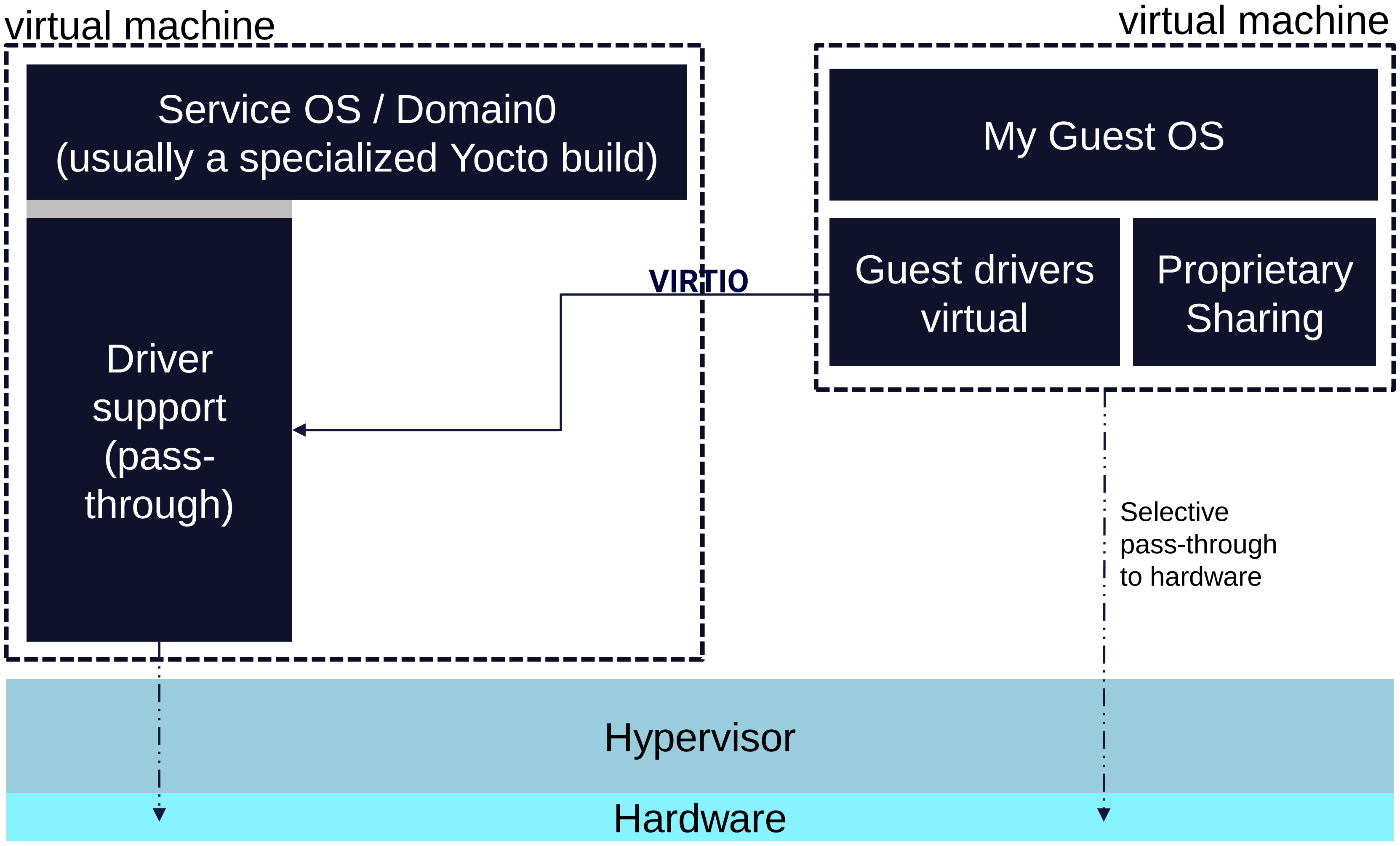
# Acoustics on the SoC



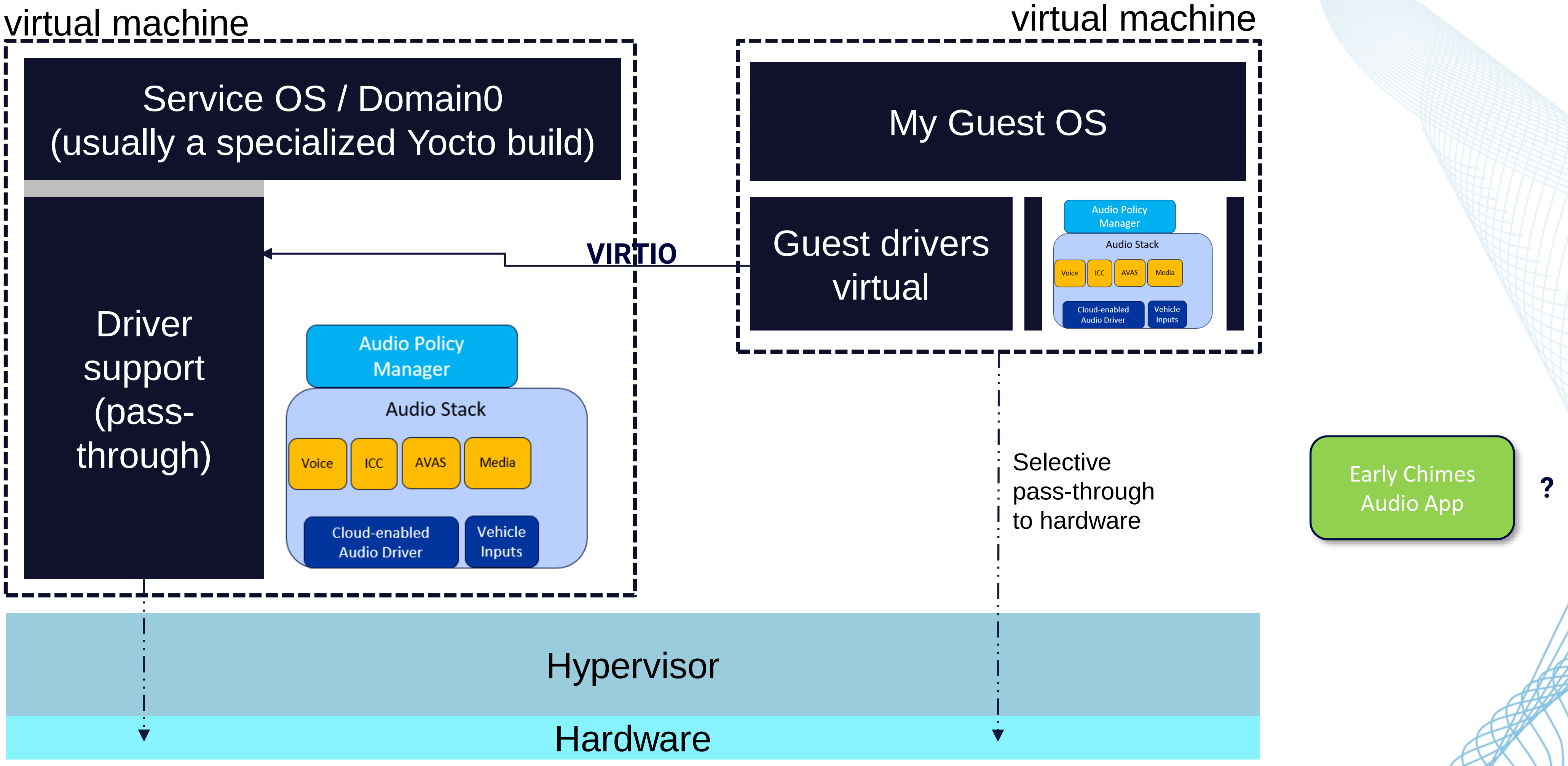
# Acoustics on the SoC with DSP Support



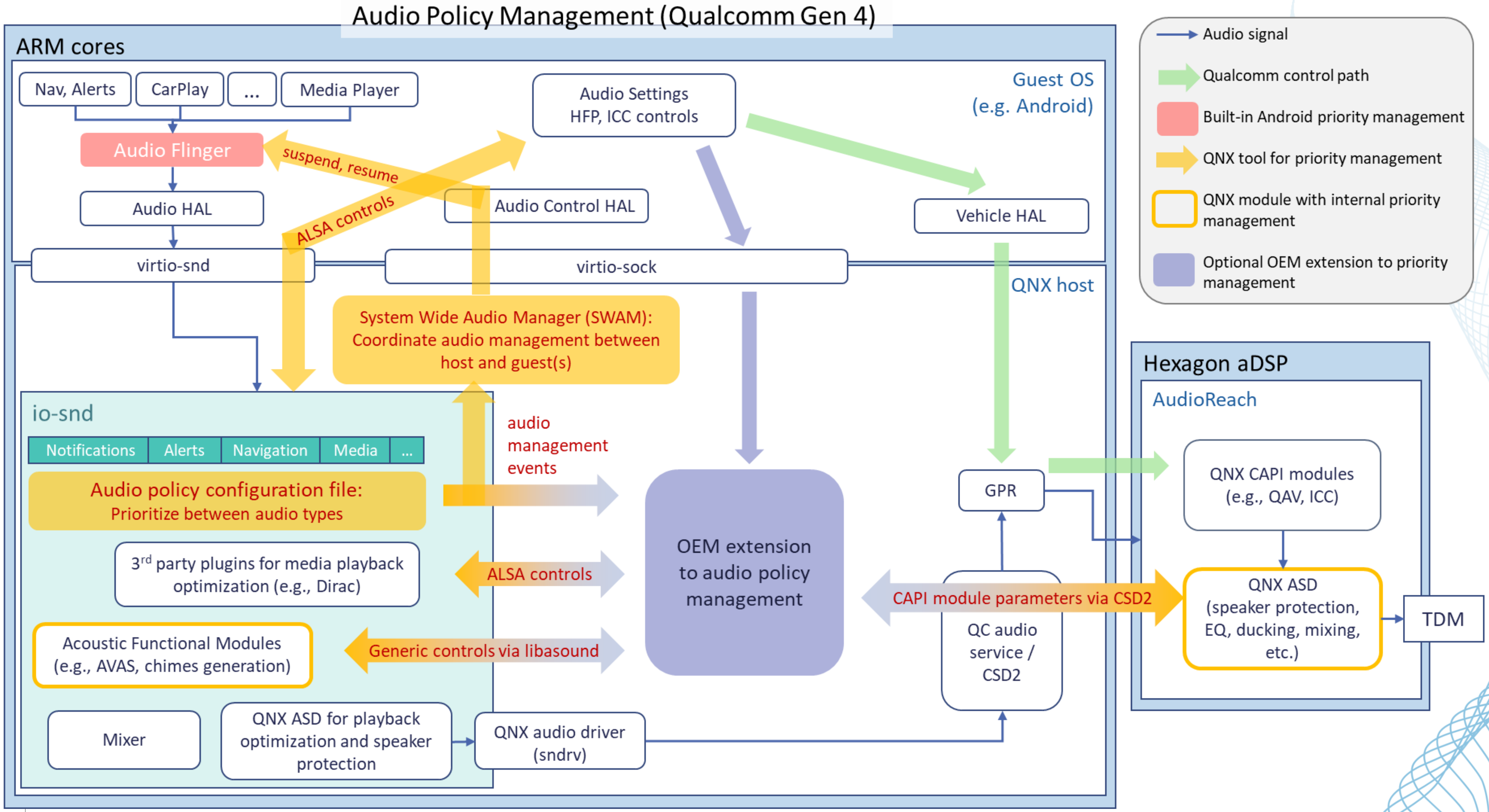
# How would I build audio sharing in a Service OS design?



# How would I build audio sharing in a Service OS design?



# QNX Example: Audio Sharing Detail View



# Hypervisor Futures

- More devices: VIRTIO and non-VIRTIO
  - Virtio-media, USB over IP (cloud), Cloud Bluetooth, DRM (Widevine L1), machine learning, Android SDV

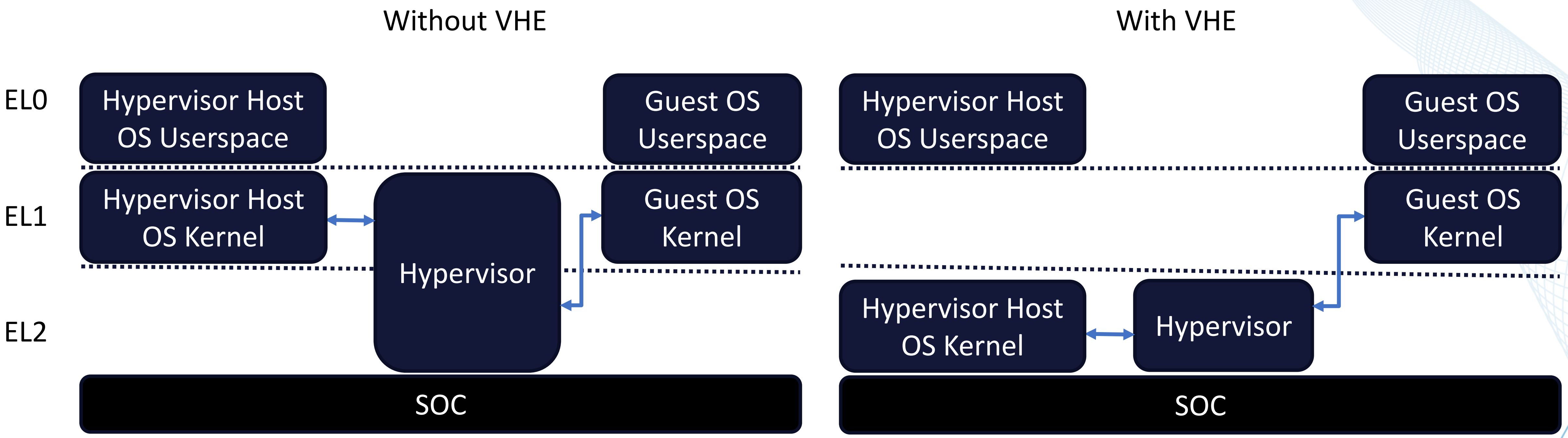
- More hardware
  - Next generation SoC support for both ARM and x86 partners

- Virtualization Host Extensions

- Safety

- SDP8 features: core clusters, jitter-free scheduling, interrupt processing

# Virtualization Host Extensions (VHE)

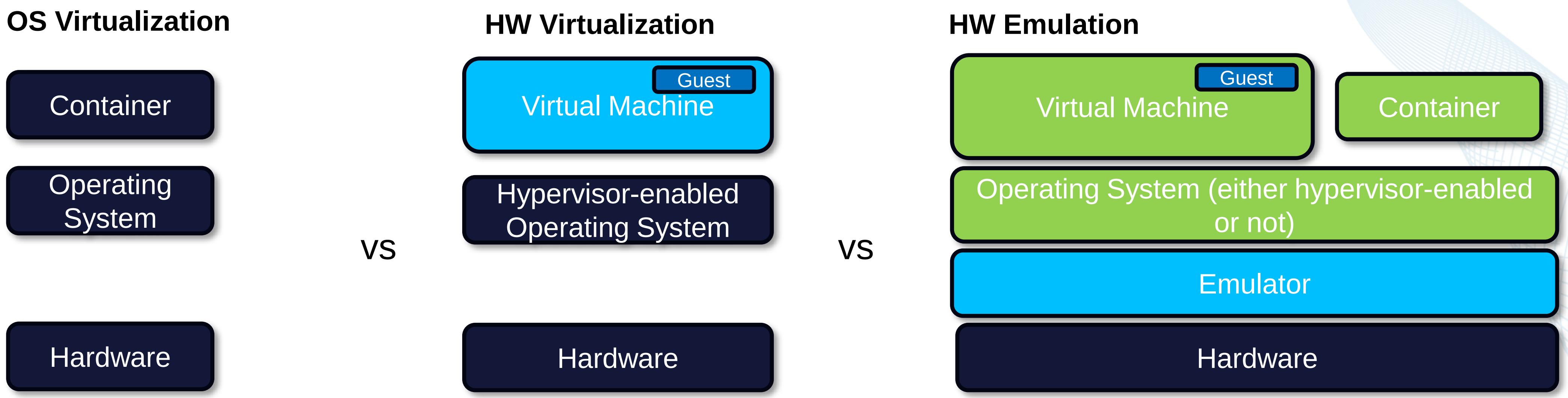



- Many context-switches between EL1 and EL2

- Host OS Kernel runs in EL2: less context switching
  - Estimated at 20% typical performance boost
- QNX Hypervisor benefits from turning on VHE
  - SDP8 version only

# Containers

# What is a Container?




Operating systems must match:
e.g. QNX Container on QNX OS,
Linux Container on Linux OS.

Container holds a set of programs that run in a restricted environment

## Container Model

Hardware must match between virtual machine and operating system:
But Guest OS running in virtual machine can be a different OS than underlying Operating System, but must be same hardware.

## Virtual Machine Model

Emulator can present different hardware to the operating system than real hardware.
A Virtual machine will use HW virtualization on top of emulator.
Container will use OS virtualization on top of emulator.

## Emulation Model

# Container Trends

As per VDC Research Report 2023:

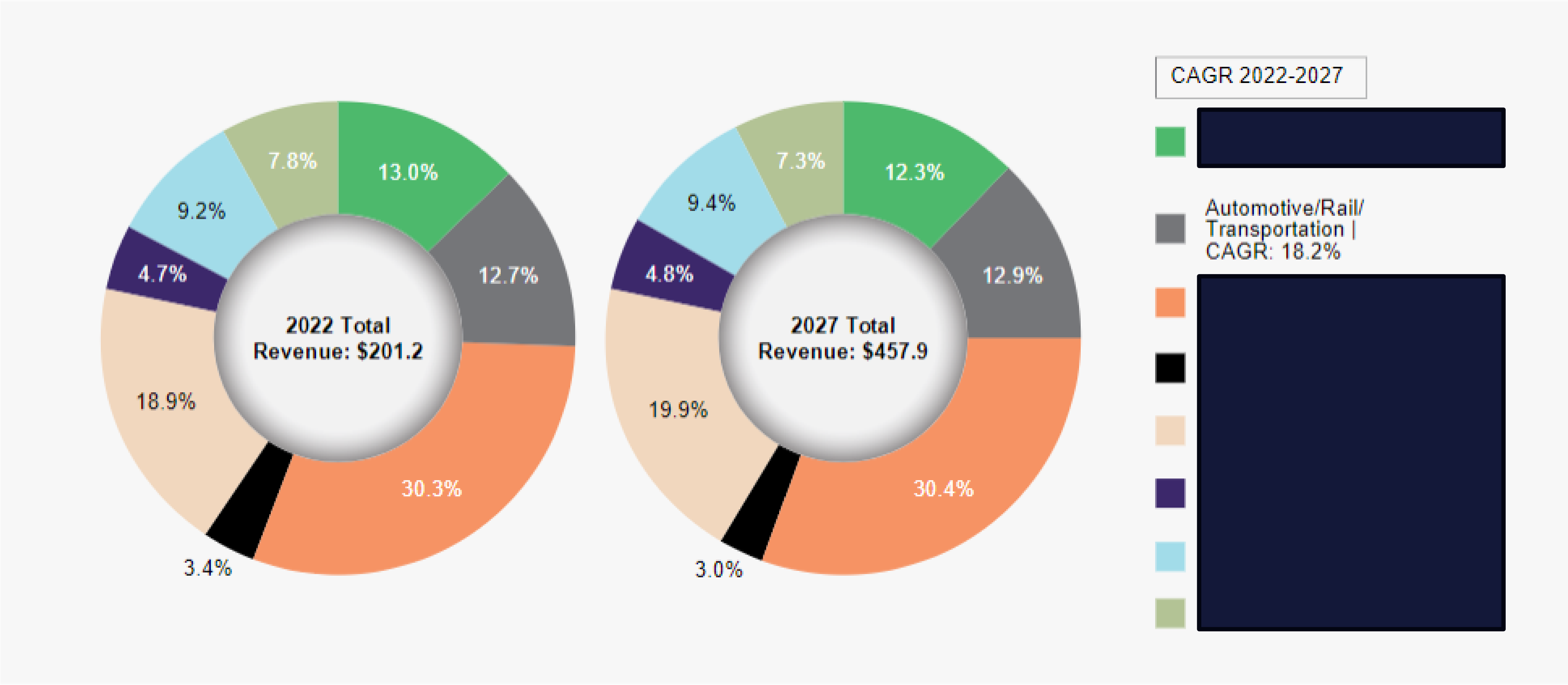
- Over 50% of automotive OEMs either already have or expect to have OTA (Over-The-Air) software support within next 3 years
  - ISO 21434 (Security update requirements) will be a force driving the adoption of containers used for OTA

Kubernetes is lead orchestration software (as the base connection agent for most tool solutions)

- Source
  - https://www.datatronic.hu/en/containerisation-in-automotive-industry/

**Exhibit 6: Global Revenue of Containers and Related Services by Vertical Market**

*(Percentage of Revenue)*




Revenue is in millions USD



Automotive 'Container market' expected to grow from $26M in 2022 to $59M in 2027.

Includes runtime environments and services

# Glossary

- **Container**
  - “Containers are executable units of software that package application code along with its libraries and dependencies. They allow code to run in any computing environment, whether it be desktop, traditional IT or cloud infrastructure.”

- **OCI Open Container Initiative** https://opencontainers.org/
  - an open governance structure for creating open industry standards around container formats and runtimes (specifications are free to all)

- **Docker** www.docker.com
  - an ‘open’ platform for developing, shipping, and running applications
  - Commercially licensed via subscription. Free options available for personal use only.

- **Kubernetes** https://Kubernetes.io
  - open-source system for automating deployment, scaling, and management of containerized applications (Apache v2, free to all)
  - Uses CRI Container Runtime Interface. Specific to Kubernetes. https://kubernetes.io/docs/concepts/architecture/cri/
    - Allows any vendor to support Kubernetes orchestration
  - CRI-O (Orchestrator) is the implementation

# Forces Driving Containers Adoption in Embedded Systems

## Containers

1.

Embedded software vendors are promoting containers:

Some vendors claim OCI-compliance

‘real-time containers’

Kubernetes control plane support

2.

Linux (all variants) have containers:

Many solutions are already used in development environments.
Wanting to leverage this knowledge in embedded edge device

3.

3rd Parties are already building container-centric models for the edge:

e.g. Mimik (edgeEngine)

Support OCI-compliant containers

4.

Engineering team awareness is changing toward use of containers:

- Why not just use a light-weight virtual machine? When and why would an embedded system need containers?
- The value of containers is increasing

5.

Usage cases are changing:

- Most involve the deployment mechanism (i.e. OTA updates)
- Edge-based orchestration
- Micro-services
- Runtime constrained environments

BlackBerry | QNX

# Example Use Cases for Containers in Embedded

**Packaging Solution:** Bundle the necessary software artifacts into a contained environment for testing and deployment.

**Over the Air Updates:** Utilize entire ecosystem of container orchestration tools like Kubernetes to manage deployment to Target.

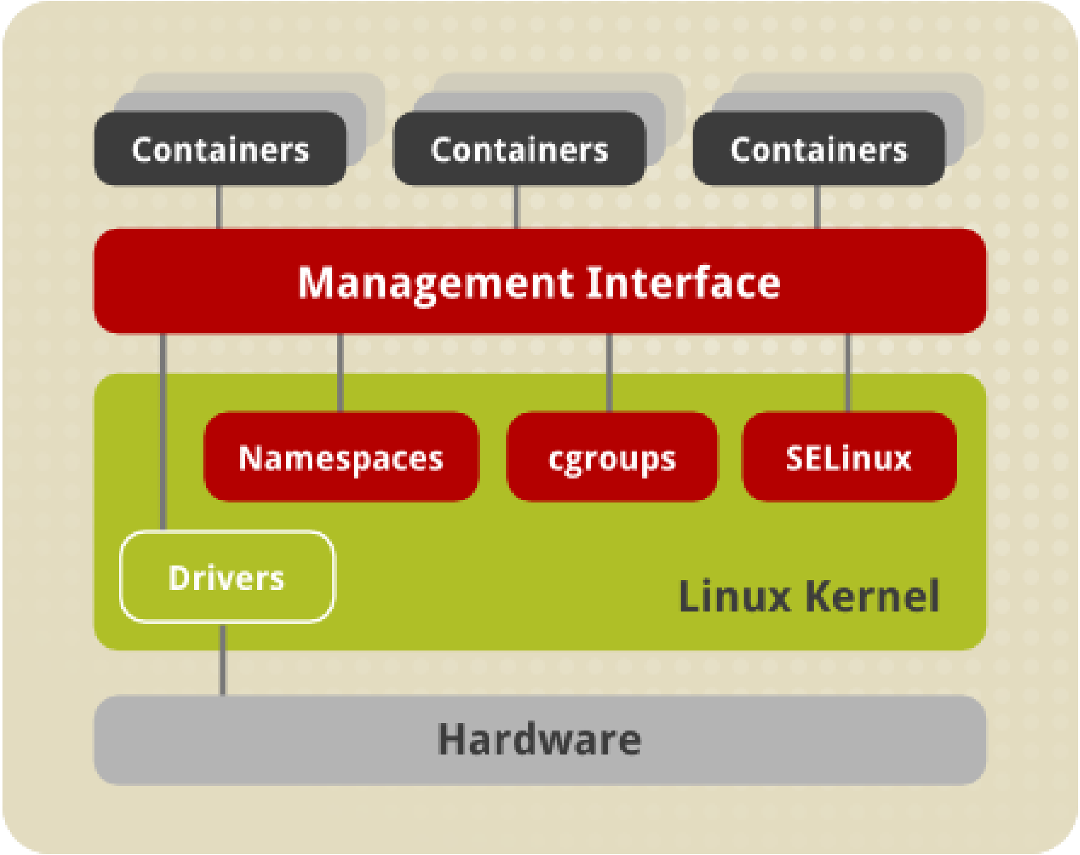
**Restricted Runtime:** Run a set of processes in a restricted environment.

**CI/CD Pipelines:** Develop and test in the Cloud at scale; code deployed in containers can be easily integrated with customer's CI/CD pipeline designed using open-source tools and frameworks.

**Isolated In-Field Testing:** Deploy beta version of software stack to target without interfering in existing processes. Isolation across container instances ensure changes in one do not affect the others.

**SBOMs**: Software dependencies embedded within the container make generating an SBOM (Software Bill of Materials) more convenient.

# Linux Containers used as Foundation for Embedded Requirements




- **Namespaces**
  - Filesystem isolation (restricted view from within container)
  - Container has its own hostname (Nodename and domainname)
  - Isolation of shared memory objects and message queues per container
  - Unique process view of processes per container
  - Separate virtual network stack, firewall, routing
  - Root operation available within container but not root access to outside

- **Control Groups** (cgroups)
  - Allocation/restriction of
    - CPU time
    - Core allocation
    - Device allocation
    - System memory
    - Network bandwidth
    - Monitoring
    - Disk bandwidth (limits on input/output access to block devices)

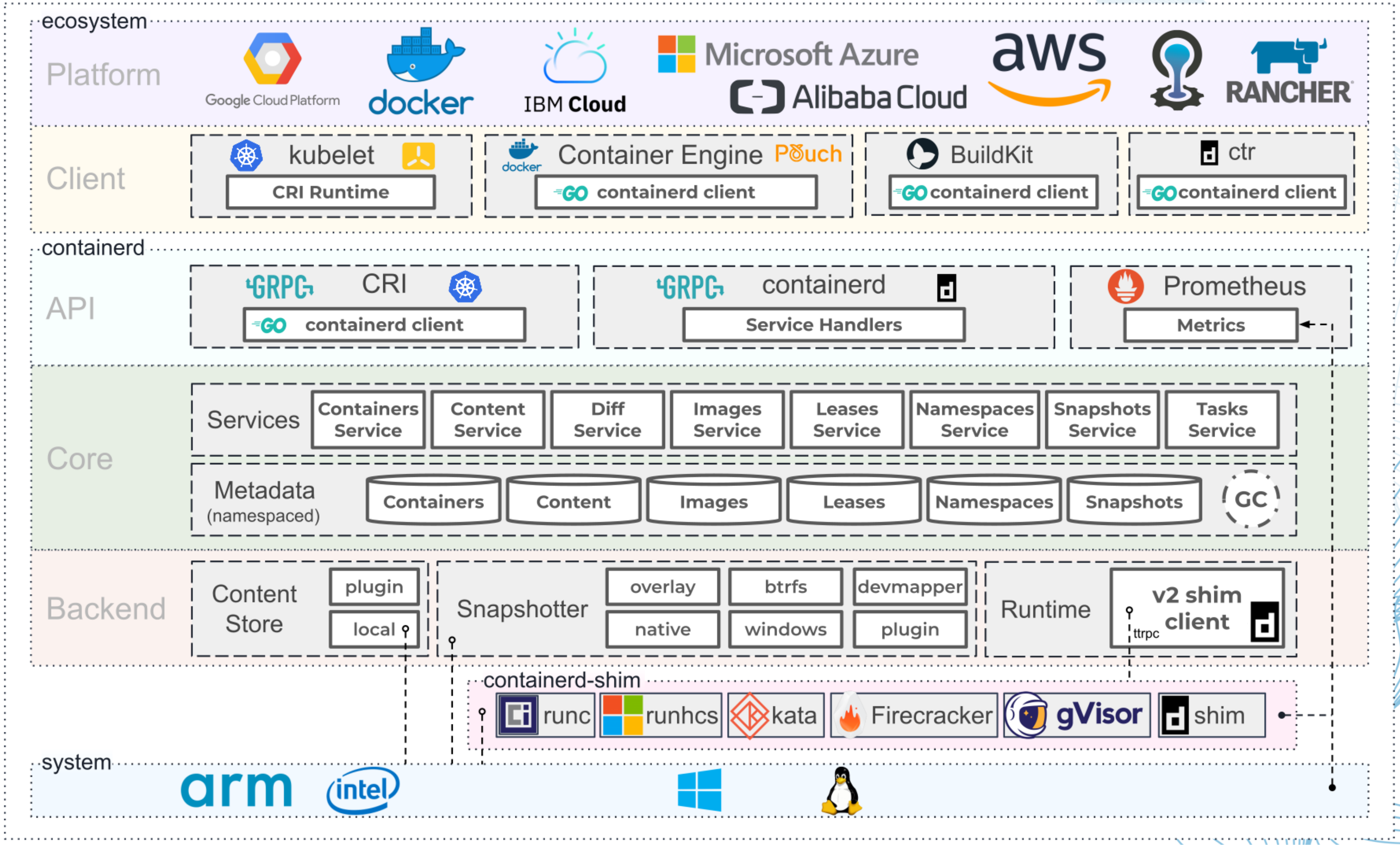
- **SELinux**
  - Security policy provides secure separation of containers
  - Including virtual devices (sVirt)

BlackBerry | QNX

# But how to bring this to Embedded?

## Linux/Windows ‘containered’ example: millions of lines of code

Manages the complete container lifecycle of its host system, from image transfer and storage to container execution and supervision to low-level storage to network attachments and beyond.




BlackBerry | QNX

Solution Overview

# QNX Containers

# QNX Containers: Summary

QNX Containers provide a QNX-based edge device with a standards-based environment for the running and management of container technology.

**Standards-based** solution:

- OCI compliant.
- Kubernetes-based toolchains for creation, deployment and management.
- Docker (industry standard) repositories are used for remote storage and retrieval. Local storage is also supported.

Follows the restrictions and security features available with QNX SDP8 including restrictions on networking, filesystems, devices, memory, communications, access control and CPU.

QNX Containers has its own TARA (Threat and Risk Analysis) as per our product release policy and ISO 21434 compliance. This restriction set provides **highly secure and isolated embedded containers** while still maintaining the high performance and hard realtime nature of the QNX SDP8 operating system.

QNX Containers co-exist with the QNX Hypervisor environment, allowing for **simultaneous use of both virtual machines and containers.**

QNX Containers are **extendable** without compromising existing design.

A **safety-certified** version is planned for future product.

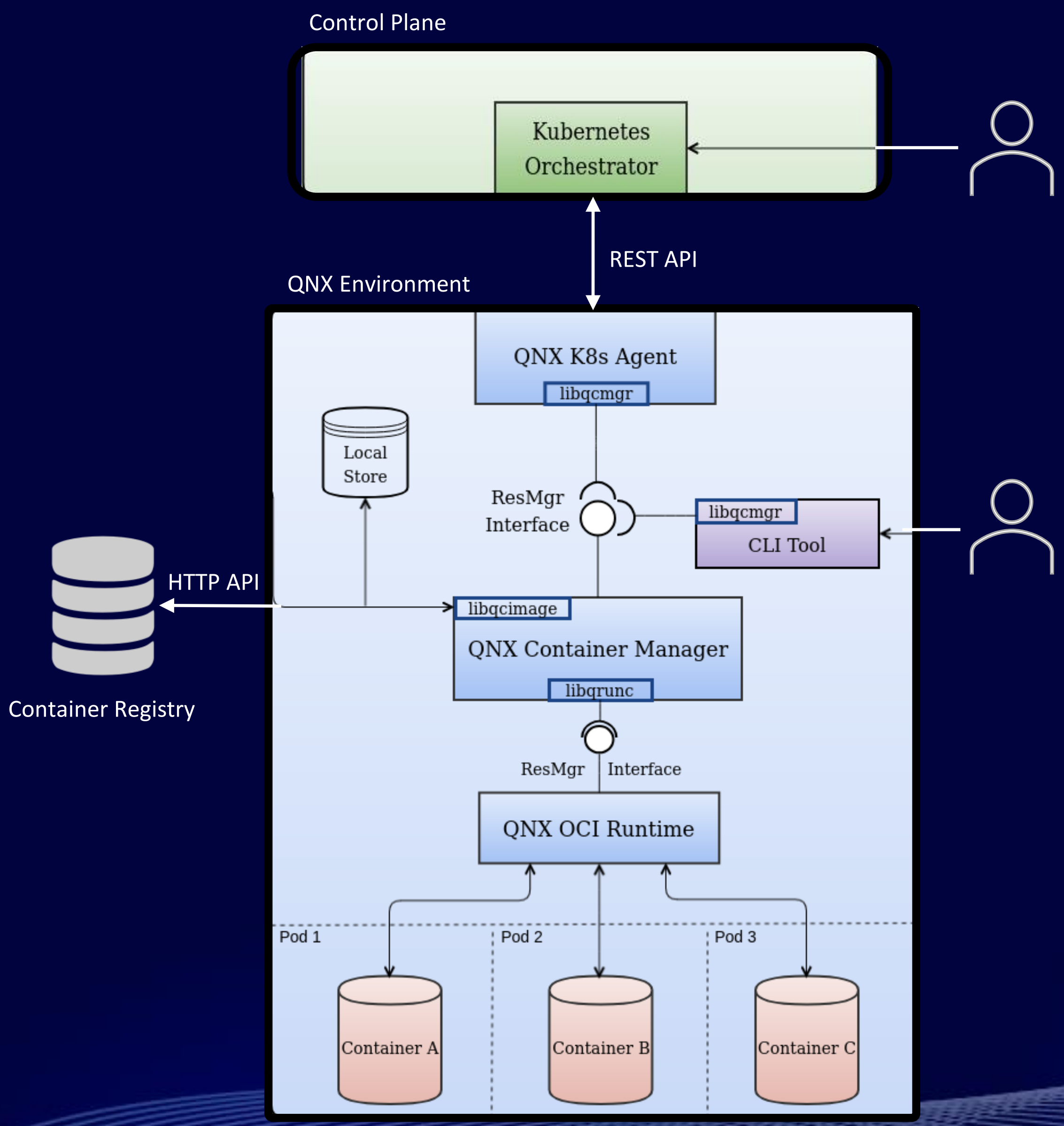


BlackBerry | QNX

# System Architecture

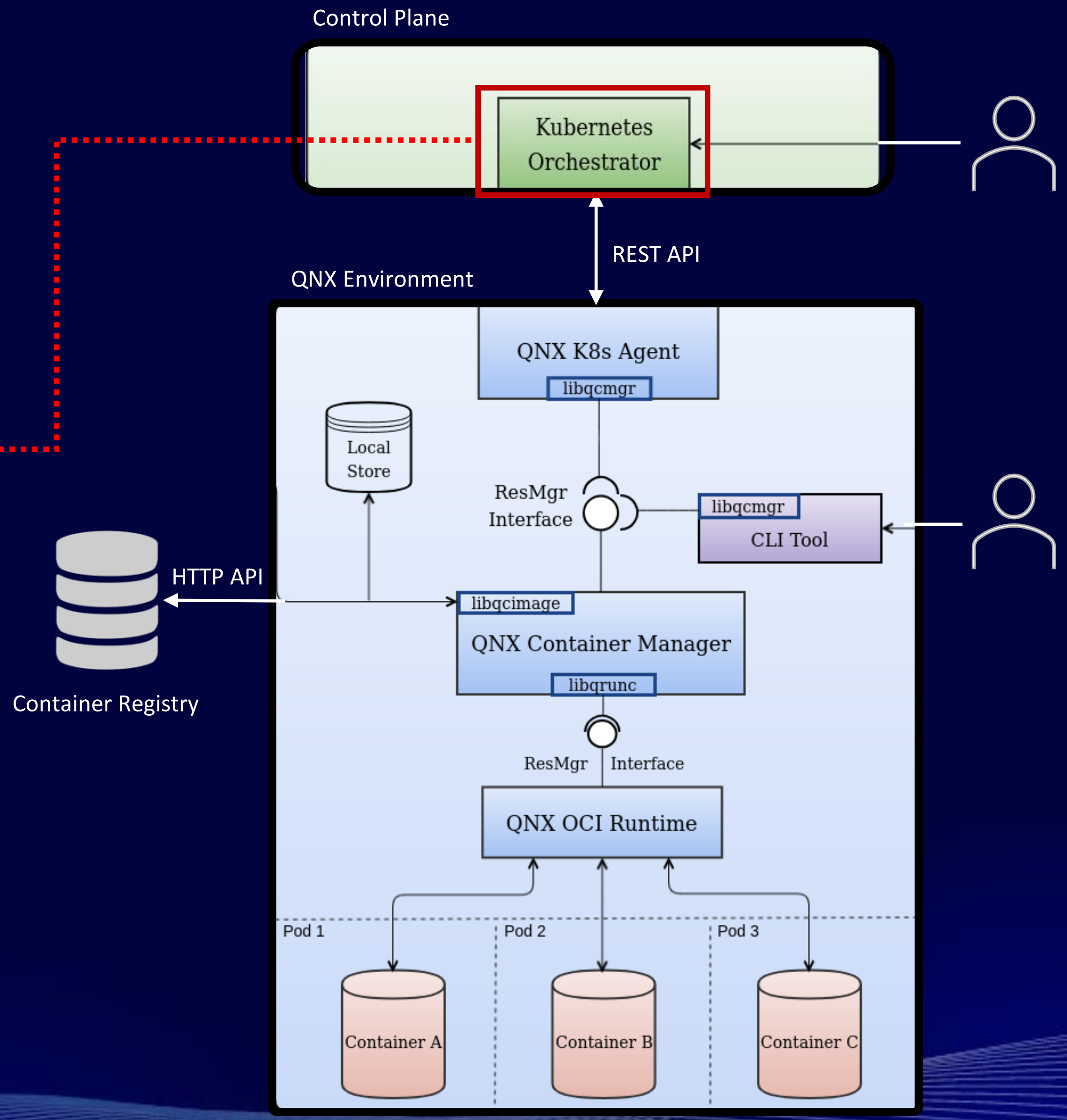
**Legend:**

- External Control Plane
- External Image Registry
- QNX Long-Running Process
- QNX Short-Running Process
- User Container Code
- User Input



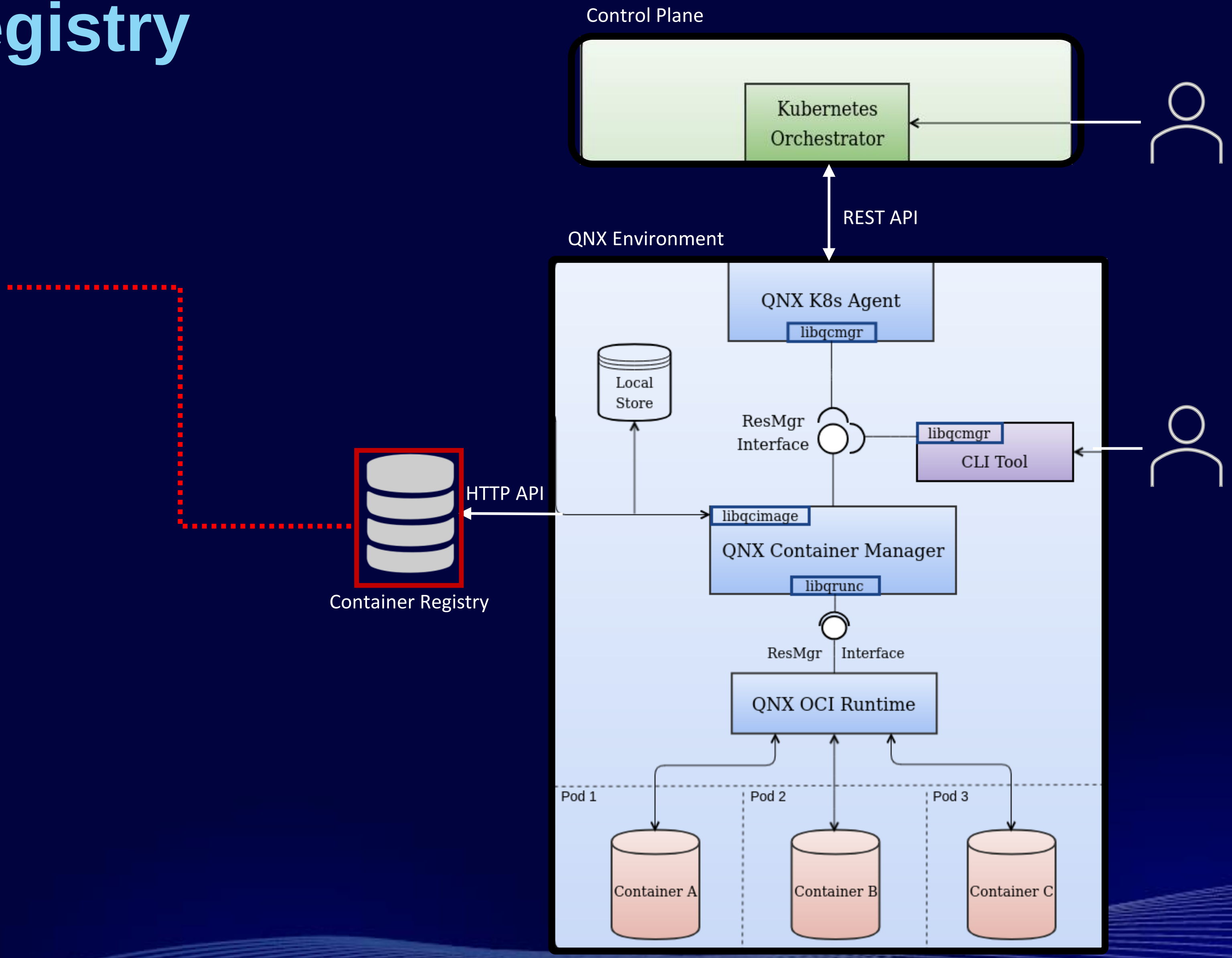
# Kubernetes Orchestrator

- Chose an orchestrator-driven solution
- Orchestration tools manage multiple nodes and automate provisioning, deployment, networking, scaling, availability and lifecycle management of containers across nodes
- Kubernetes is a widely adopted open-source container orchestration tool popular in cloud computing
- Typically hosted separate and designed to be used as a control plane of a multi-node container solution



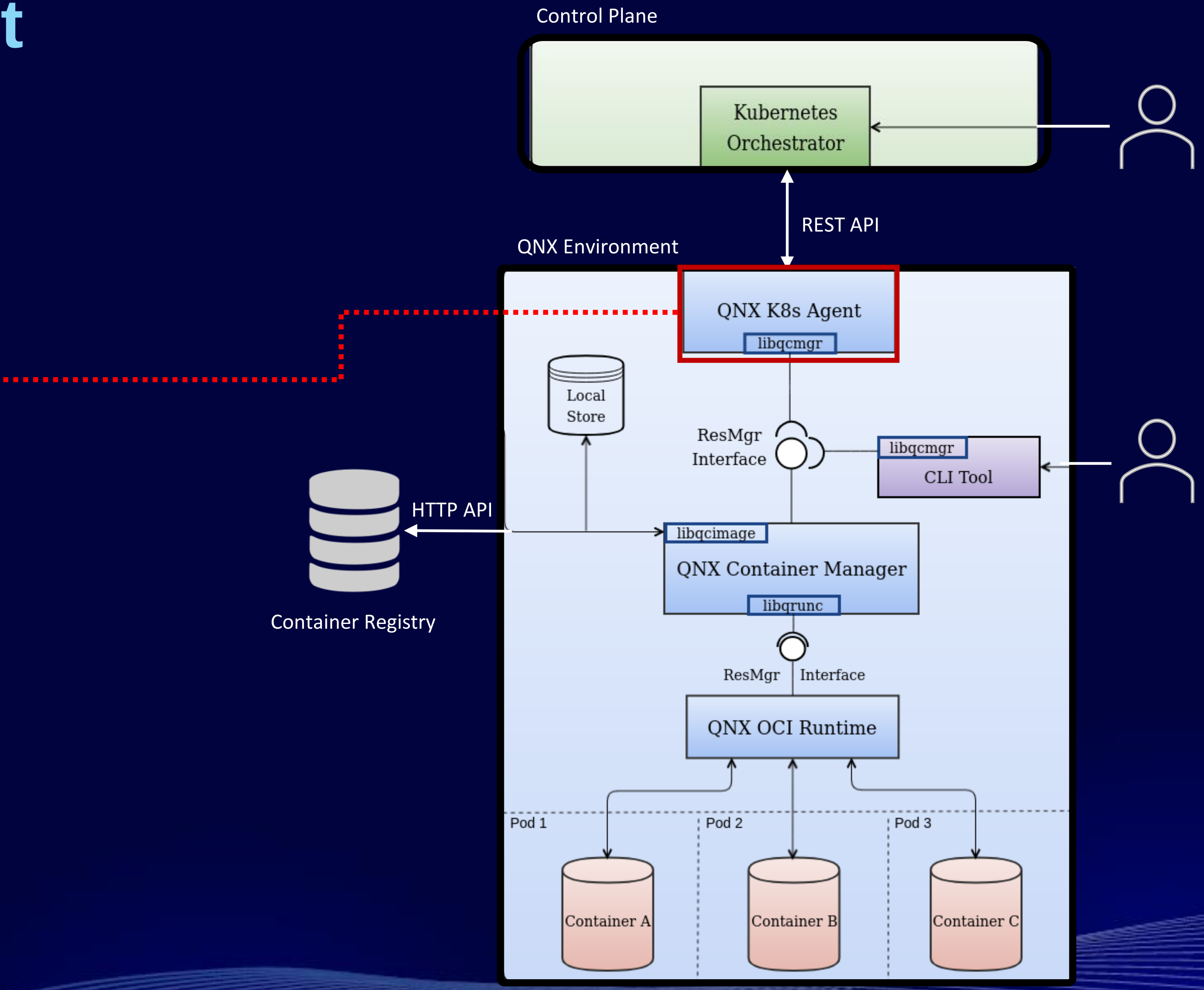
# Container Image Registry

- Third party server housing a collection of container image repositories
- Provides developers the means to easily store / share container images, while managing access control, permissions and authentication
- Docker Hub and AWS Elastic Container Registry (ECR) are popular examples



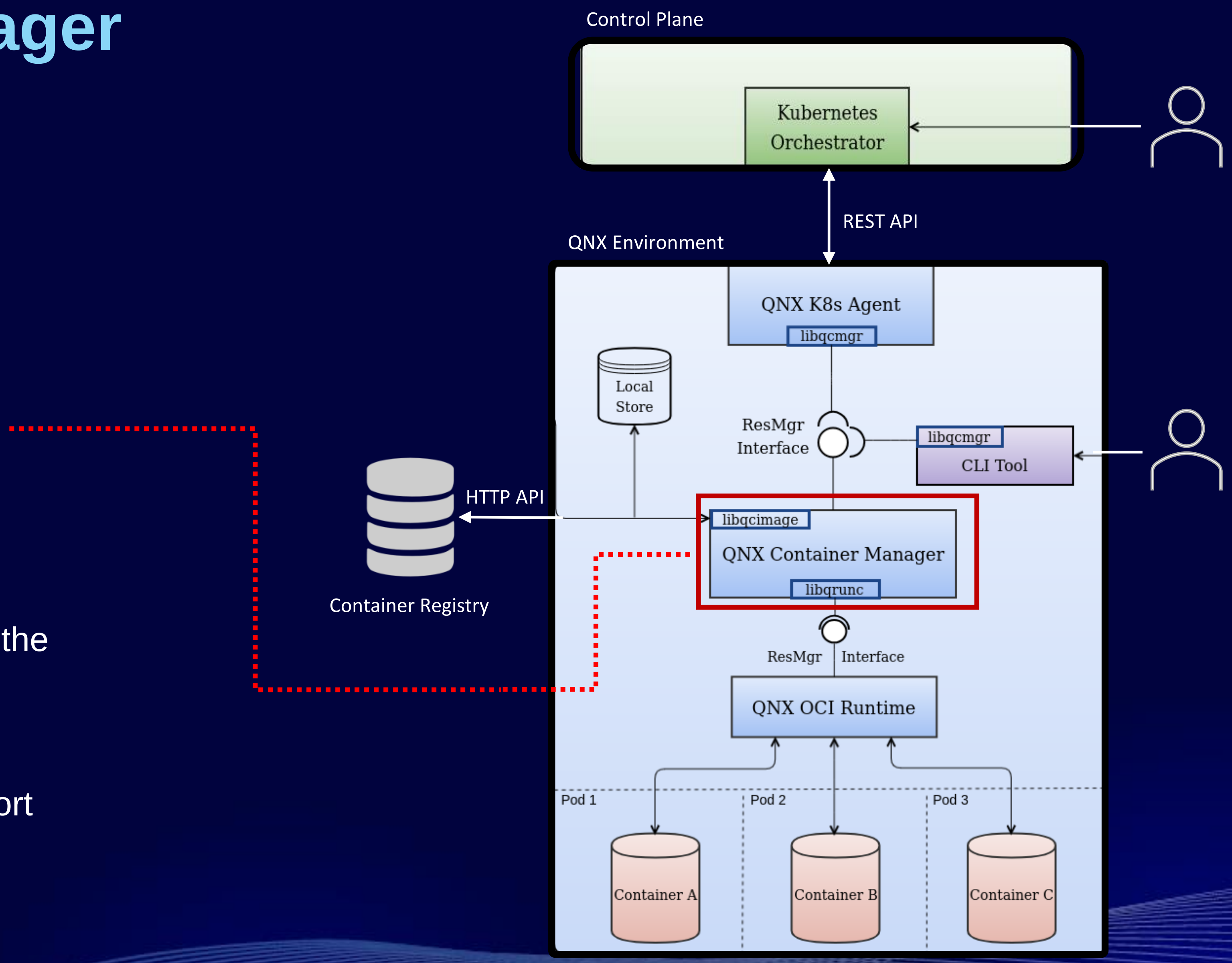
# QNX Kubernetes Agent

- Managed Kubernetes nodes require an agent, a Kubelet (Qubelet in QNX)
- Responsible for communication with Kubernetes controller for creating and running pods
- Pods are the smallest unit of configuration, representing a set of one or more containers and their configurations
- RESTful APIs are used for communication (representational state transfer)
- Listens for new pod configuration messages from the control plane and forwards requests through QNX resmgr framework to the QNX Container Manager



# QNX Container Manager

- Responsible for the management of containers on a single node (QNX Runtime Environment)
- Listens and acts on control messages from the orchestration control plane
- Interacts with the Container Image Library for image management
- Interacts with the Low-level Container Runtime for runtime management
- Monitors container states and notifies the control plane of any updates
- Qubelet separation from Qcmgr hides K8s details and allows for future support of other orchestration tools

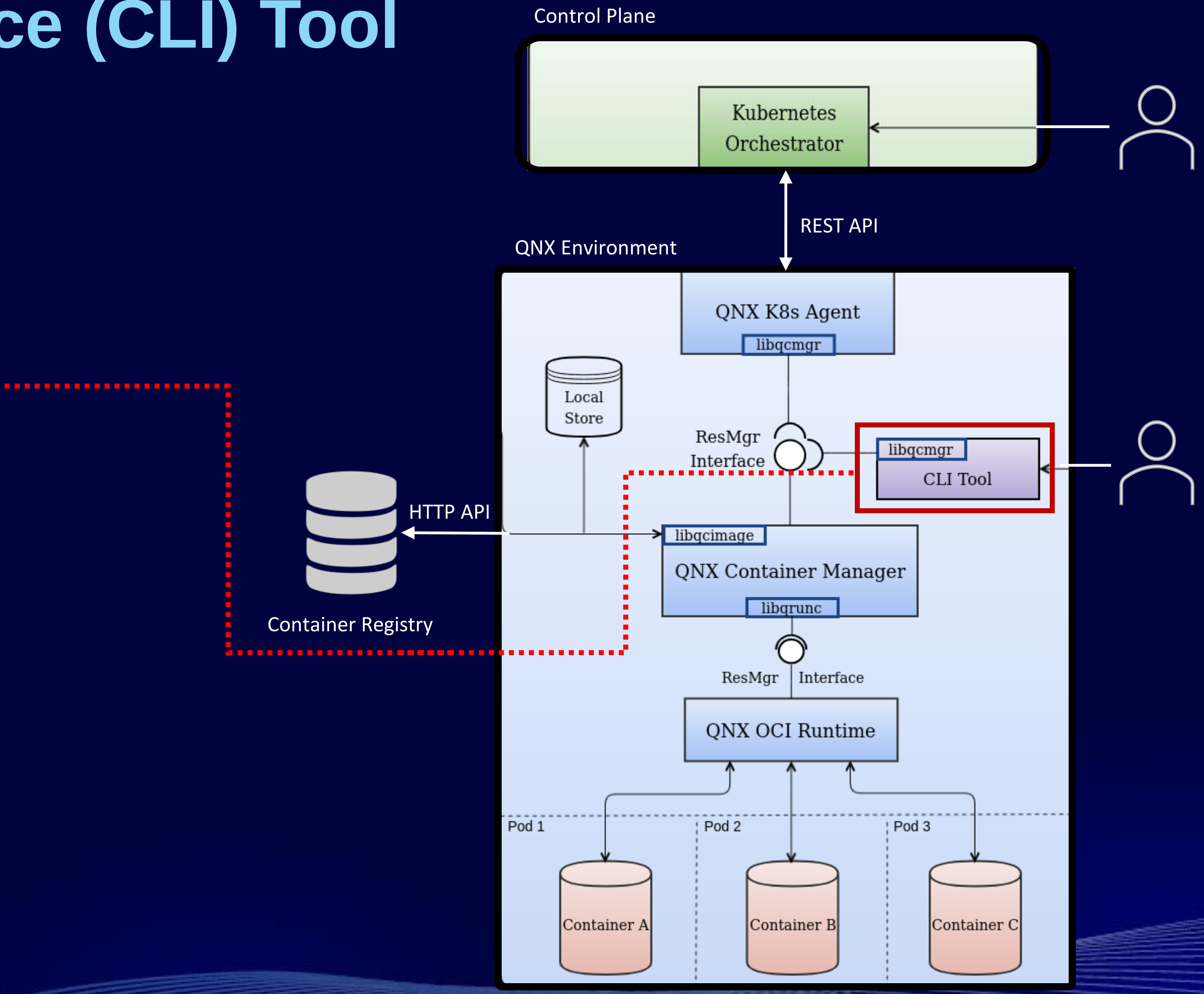


# Command Line Interface (CLI) Tool

- Command and Control CLI tool to manage container functionality

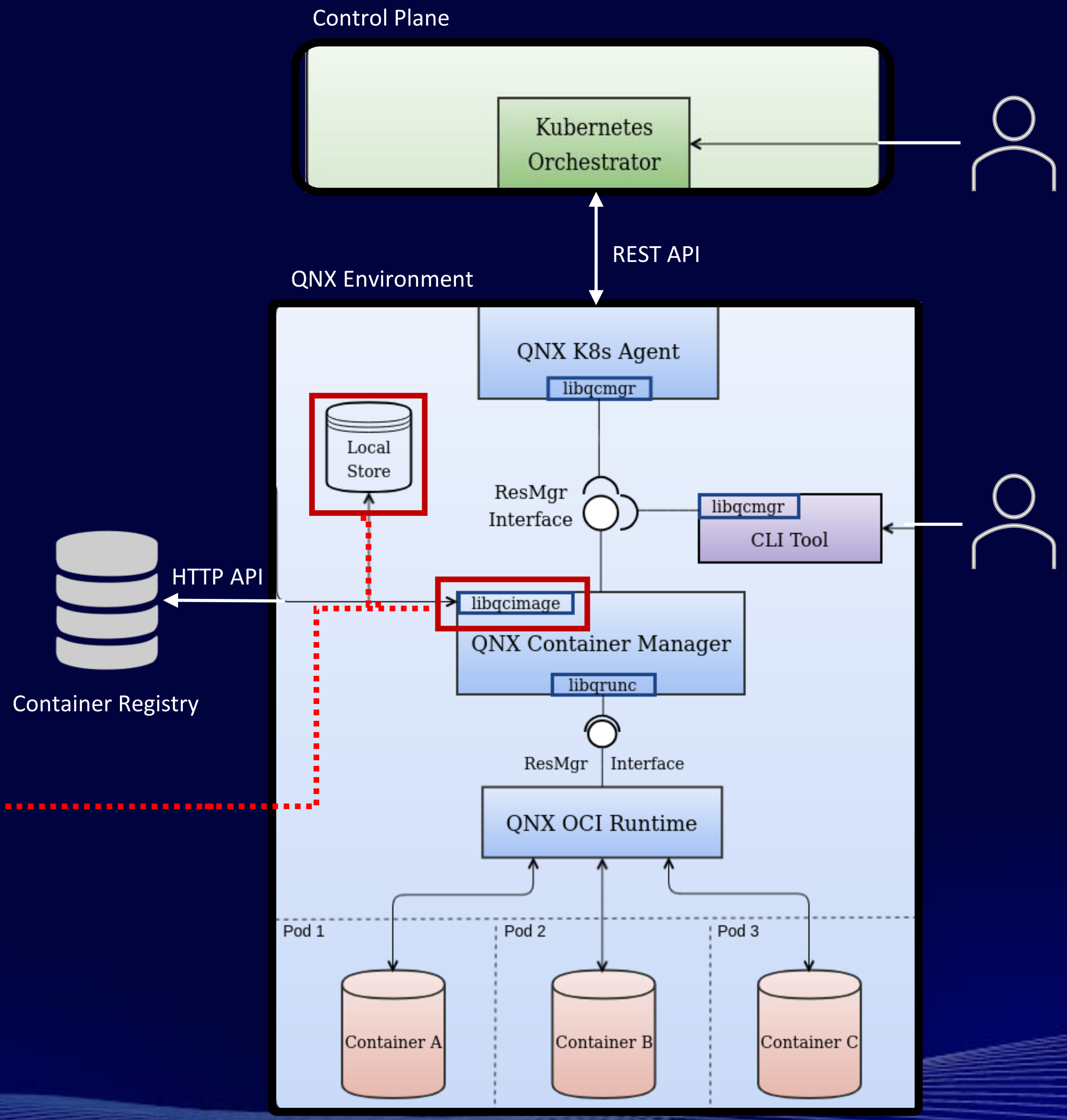
- Designed to utilize the same resource manager interface between Qubelet and the Container Manager

- Provided as an option for users who want to either:
  - Use the QNX Containers solution without Kubernetes orchestration and/or
  - Have an on-device method to manage containers without the need for external access



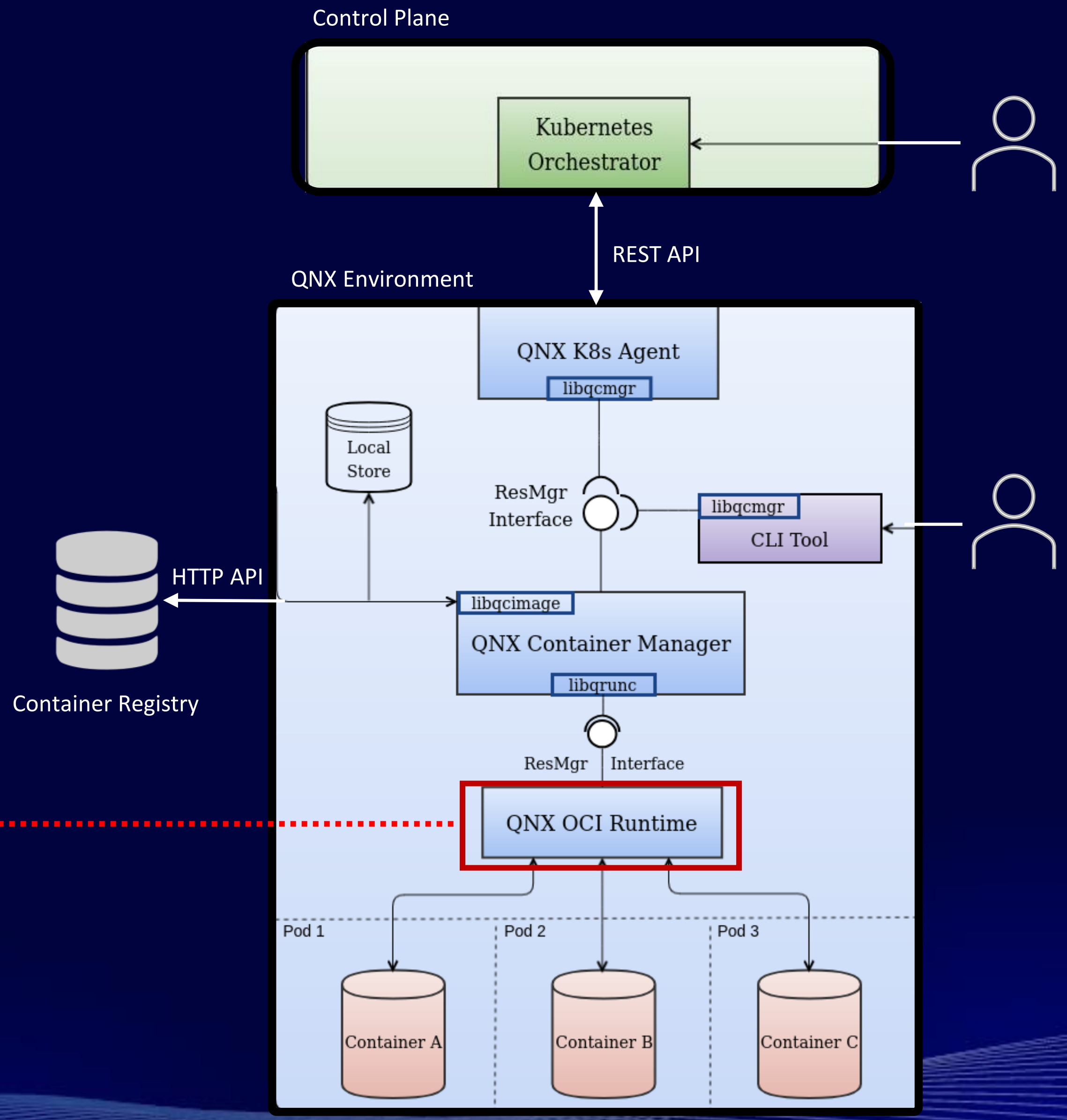
# Container Image Library

- Library responsible for downloading and managing images from a supported container registry
- Supports Docker Registry HTTP API V2
  - Any Private/Public AWS ECR
  - Docker Hub public registry
- Supports standard image formats
  - Docker Image V2, Schema 2
  - OCI Image Specification
- Uses local store to cache downloaded images and provide on-target access
- Once downloaded, image is unpacked into an Open Container Initiative (OCI) runtime bundle for the Low-level Container Runtime to consume



# Low-level Container Runtime

- Low level OCI-compliant component for running containers
- Responsible for creation and deletion of the container environment within the QNX system
- This environment consists of:
  - File system isolation
  - CPU and Memory Isolation
  - Launching and monitoring of container process
  - Handling container process logging
  - Stopping container process and cleaning up resources



# QNX Containers Roadmap

- QNX is building first product view this year
- Non-safety version for first release
- SDP 8 only
- Safety version following behind non-safety version (after release of QNX OS for Safety 8.0)
- Early Access program to be announced
  - If interested, please contact your QNX sales team

# Device Abstraction

# QNX Signal Service

## COMMON SIGNAL ACCESS

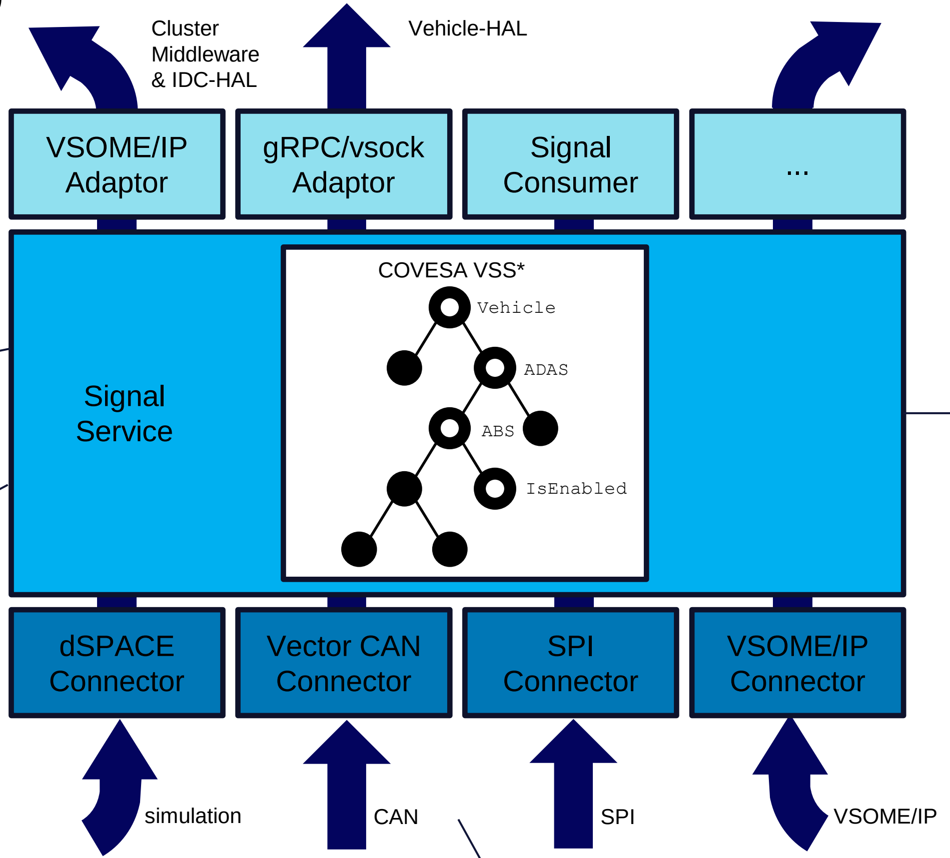
Integrates vehicle data into distinct systems (e.g., cluster, IVI, …)

Support adaptation to different deployments and architectures (e.g., Vera OS, AAOS, Android SDV, …)

Uses vehicle data directly

Exposes vehicle data in common form (COVESA VSS)

Supports reading from sensors and writing to actuators




Caches vehicle data and enables both synchronous "get" and asynchronous "subscribe"

Manages fine grained permission and access control to vehicle data

Enables advanced filtering on subscriptions (e.g., filter by rate, filter by value)

Support the development of Data Adaptors and Connectors with intuitive APIs and tooling

Connect vehicle data sources and interfaces (hardware, services, simulation sources, …)

*Connected Vehicle Systems Alliance Vehicle Signal Specification

# QNX Signal Service

## POSIX INTERFACE

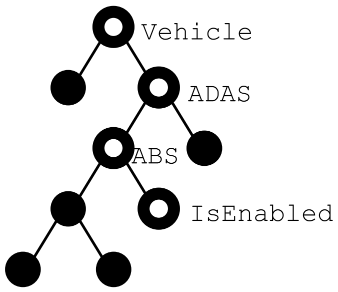
### COVESA VSS

- Open Source, standards-based tooling for defining and customizing a signal catalog
- Signals defined as tree structure with human readable path names (
  - e.g., Vehicle.ADAS.ABS.IsEnabled)
- Default definitions for ~2000 signals and growing

```
"Vehicle": {
  "children": {
    ”ADAS": {
      “children”: {
        ”ABS": {
          "children": {
            ”IsEnabled": {
              "datatype": "Boolean”,
              "description": "Indicates if ABS is
               enabled. True = Enabled. False = Disabled.",
              "type": "actuator",
              "x-euuid": "8589e0a5c2773ba0",
              "x-read-permission": "Vehicle.ADAS.READ",
              "x-write-permission": "Vehicle.ADAS.WRITE"
…
```

### SIGNAL SERVICE

- Loads the COVESA signal catalog
- Manages subscriptions//actuation
- Manages signal access/permissions
- Data Connectors and publishing




### POSIX INTERFACE

- Exposes signals via standard POSIX interfaces that reflect the VSS tree as directory structure
- Access using open/select/read/write/close
- Simple intuitive interface for creating Data Consumers, Adaptors and Connectors
- High performance IPC

```
/dev/signal/
   Vehicle/
      ADAS/
         ABS/
            IsEnabled
            IsEngaged
            IsError
         CruiseControl/
            IsActive
            IsEnabled
            IsError
            SpeedSet
```

**Example**

Signal Access

```
int fd = open(“/dev/signal/Vehicle/Cabin/HVAC/AmbientAirTemperature”,
O_RDONLY, O_NONBLOCK);
…
select(nfds, &readfds, null, null, &tv);
…
int num_bytes = read(fd, buffer, len);
```

Signal Publishing

```
int fd = open(“/dev/signal/Vehicle/Cabin/HVAC/AmbientAirTemperature”,
O_WRONLY);
…
int num_bytes = write(fd, value, len);
…
```

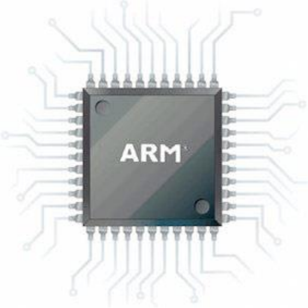
# Summary: Where Does Virtualization Go from Here?

## Software will be the key; hardware independent design

**Desktop PC (in vehicle trunk)**

**1990s**

**Windows/Linux Test and Research**




**A-Core, M-Core, R-Core, DSP**

**2010s to today**

**Mix of software components with safety and non-safety**

**SaaS (Chiplet)**

**Future**

**Safety-certified hypervisor-enabled RTOS becomes the most important component. Narrow to general AI on general-purpose SoC with general-purpose instruction set**

# Thank you