
# Joint presentation:
# Container Based Architecture for AGL

Automotive Grade Linux All Member Meeting
October 22-23, 2019

**Naoto Yamaguchi**
AISIN AW CO.,LTD.

**Tadao Tanikawa**
Panasonic Corporation

**Kazumasa Mitsunari**
WITZ CO.,LTD.

# Today presenters

Name :
Naoto Yamaguchi

Company :
AISIN AW CO.,LTD.

Career :
Automotive platform software engineer since 2007.

Name :
Tadao Tanikawa

Company :
Panasonic Corporation

Career :
GNU/Linux system since 1996, Embedded Linux for mobile since 2004, and Linux for Automotive since 2012.

Name :
Kazumasa Mitsunari

Company :
WITZ CO.,LTD.

Career :
Automotive software engineer since 2015

AUTOMOTIVE GRADE LINUX

# Outline

- Instrument Cluster EG
- Concept
- Container Based Architecture
  - Overview
  - Key technology : Graphics
  - Key technology : Sound
  - Key technology : CAN
- Conclusion
- Q&A

# Outline

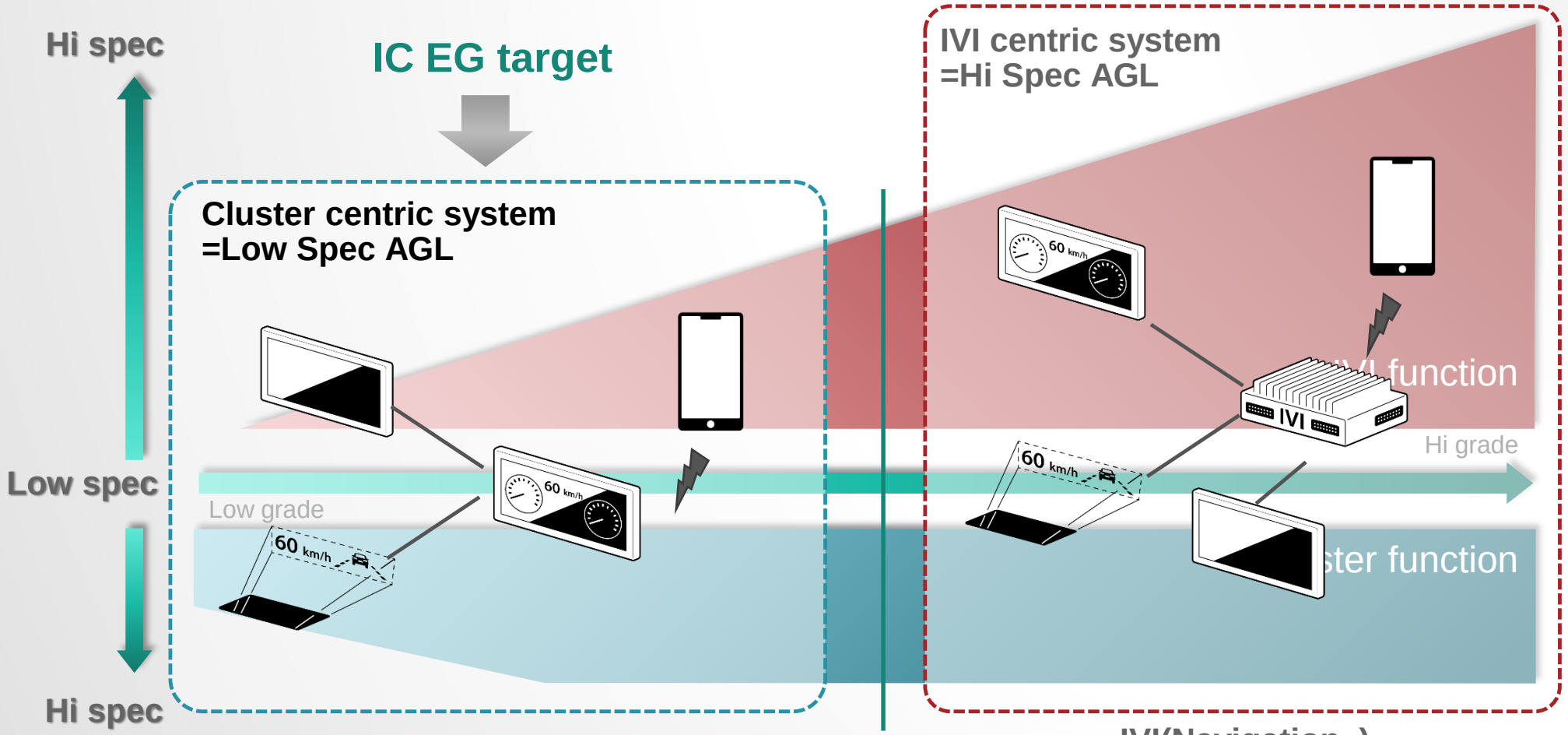
- **Instrument Cluster EG**
- Concept
- Container Based Architecture
  - Overview
  - Key technology : Graphics
  - Key technology : Sound
  - Key technology : CAN
- Conclusion
- Q&A

# Instrument Cluster EG

- Motivation
  - Create a base platform for instrument cluster by using Linux.
  - Solve some of the product development issues in AGL community.

- Members
  - Suzuki (Leader), Toyota, Honda, Mazda
  - Denso, Panasonic, Continental, Bosch, Nippon Seiki
  - Denso Ten, Aisin AW

# EG scope and system image?




Cluster(HUD)
+
Smart phone connection(SDL..)
for small IVI function

IVI(Navigation..)
+
Cluster(HUD)
+
Smart phone connection(SDL..)

AUTOMOTIVE GRADE LINUX

# Outline

- Instrument Cluster EG
- **Concept**
- Container Based Architecture
  - Overview
  - Key technology : Graphics
  - Key technology : Sound
  - Key technology : CAN
- Conclusion
- Q&A

# What are the product development issues?

1. **Quality and Robustness**
   - Functional safety is required.
   - Quality management is required.

2. **Lightweight**
   - Constraints on boot time are severe.
   - Current AGL stack is heavyweight.

# Functional safety

**Main function is the very function of our system**

- Requires advanced quality management.
- Requires open innovation.
- Requires cyber security.
- Requires fast boot.
- Requires various functions.
- …

Main target of IC-EG EG

**Safety function ensures vehicle safety**

- What function does it include?
- Which OS do you use?
- Which communication method do you use?

Collaborate ELISA to find a solution.

Main function

Safety function

Isolation method

Functional safety will be discussed in the ELISA Project.

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

**Main function and safety function are isolated by isolation method.**

- Hardware separation? Using hypervisor?

Collaborate ELISA to find a solution.

AUTOMOTIVE GRADE LINUX

# What are the product development issues?

## 1. Quality and Robustness

- Functional safety is required.
  - **Collaborate with ELISA Project**
- Quality management is required.

## 2. Lightweight

- Constraints on boot time are severe.
- Current AGL stack is heavyweight.

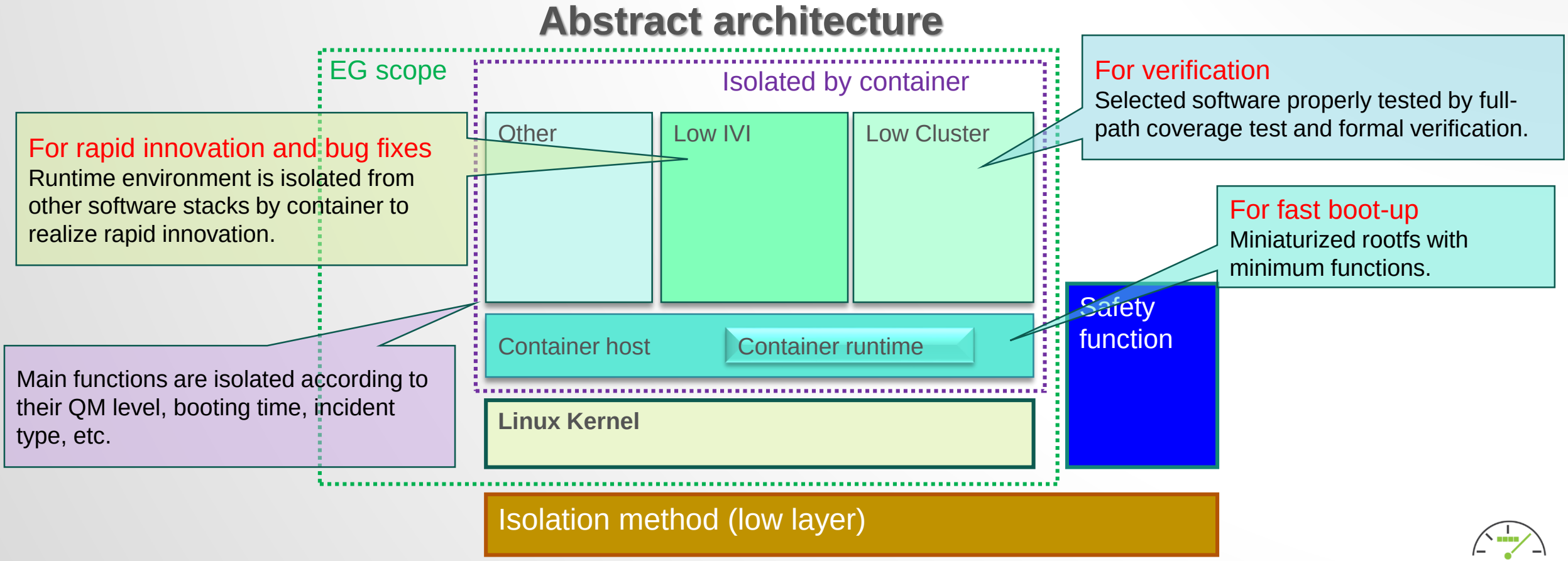
# Puzzles in automotive quality management

- There are many puzzles in the automotive system (main function).

| IVI | Instrument Cluster |
| --- | --- |
| • Rapid innovation<br>  • New features are added<br>  • Short-term development<br>  • Rapid bug fixes | Puzzle<br>• Advanced quality management<br>  • Full path coverage testing<br>  • Formal verification<br>  • Careful bug fixes |
| • Various functions<br>  • Many pre-installed applications<br>  • Applications installed from store | Puzzle<br>• Selected functions<br>  • Combinational verification<br>  • Fast boot-up |

# QM Isolation

- Our answer to the puzzle issues is “one more isolation method" which takes one-more layer to isolate the functions by using Linux container technology.



# What are the product development issues?

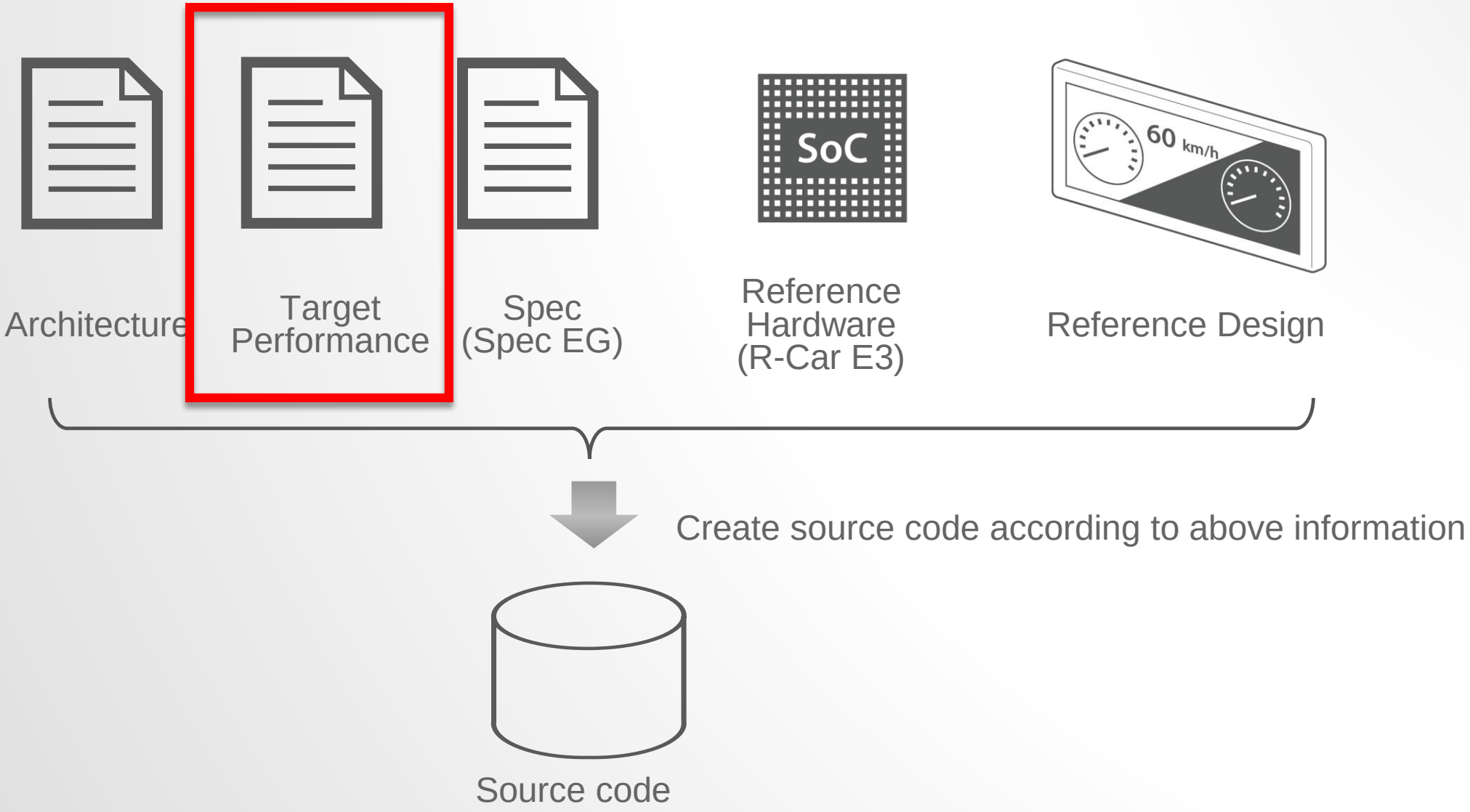
## 1. Quality and Robustness

- Functional safety is required.
  - **Collaborate with ELISA Project**
- Quality management is required.
  - **QM Isolation**

## 2. Lightweight

- Constraints on boot time are severe.
  - **QM Isolation**
- Current AGL stack is heavyweight.

# Current AGL stack is too heavy weight



# What are the product development issues?

## 1. Quality and Robustness

- Functional safety is required.
  - **Collaborate with ELISA Project**
- Quality management is required.
  - **QM Isolation**

## 2. Lightweight

- Constraints on boot time are severe.
  - **QM Isolation**
- Current AGL stack is heavyweight.
  - **Determine target performance**

# What are the product development issues?

1. **Quality and Robustness**
   - Functional safety is required.
     - **Collaborate with ELISA Project**
   - Quality management is required.
     - **QM Isolation**

**Most important keyword:**

**QM Isolation**

2. **Lightweight**
   - Constraints on boot time are severe.
     - **QM Isolation**
   - Current AGL stack is heavyweight.
     - **Determine target performance**

# Outline

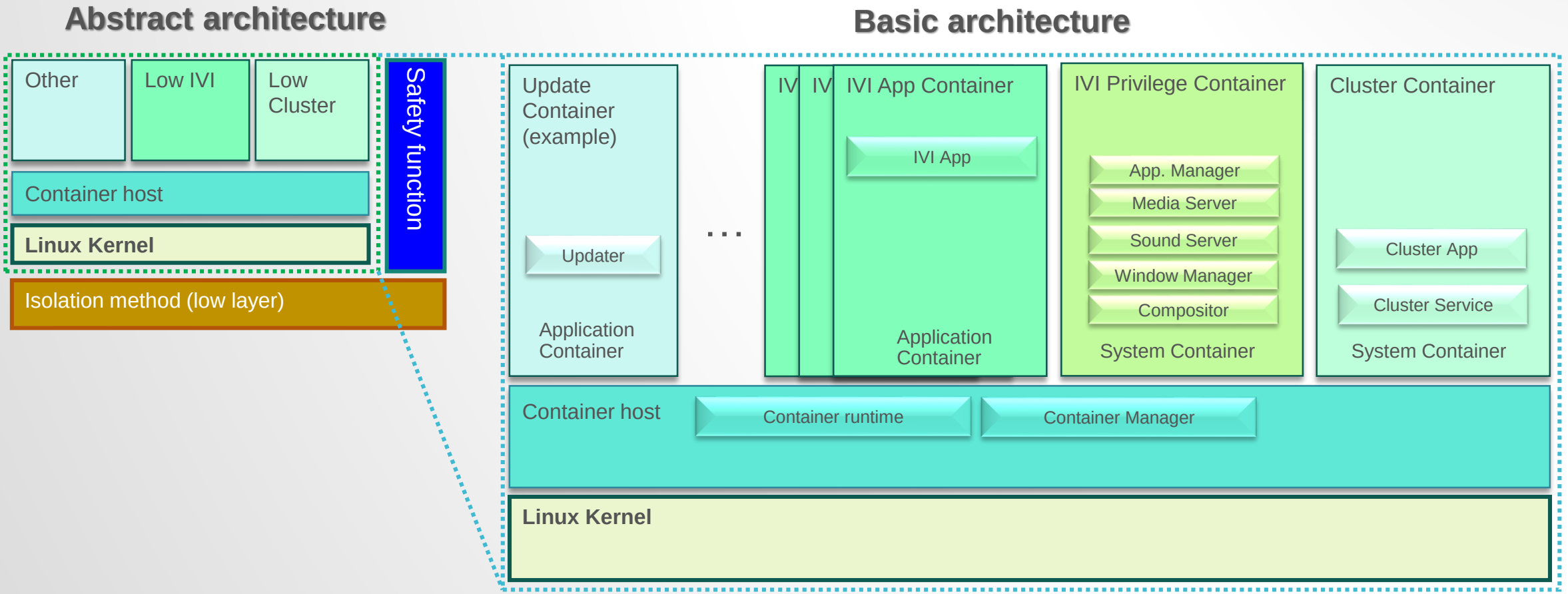
- Instrument Cluster EG
- Concept
- **Container Based Architecture**
  - **Overview**
  - Key technology : Graphics
  - Key technology : Sound
  - Key technology : CAN
- Conclusion
- Q&A

# What is QM isolation?

- "One more isolation" is a method to take one-more layer to isolate the functions with Linux container technology.
- Why container?
  - Linux container technology
    - Isolate root filesystems on Linux kernel by using **chroot.**
      - Isolates software stack in accordance with their QM level.
    - Control resource (such as cpu, memory) by using **cgroups.**
      - Guarantees the resources to instrument cluster.
    - Hide resources from other containers by using **namespace.**
      - Protects cluster resources from other functions.

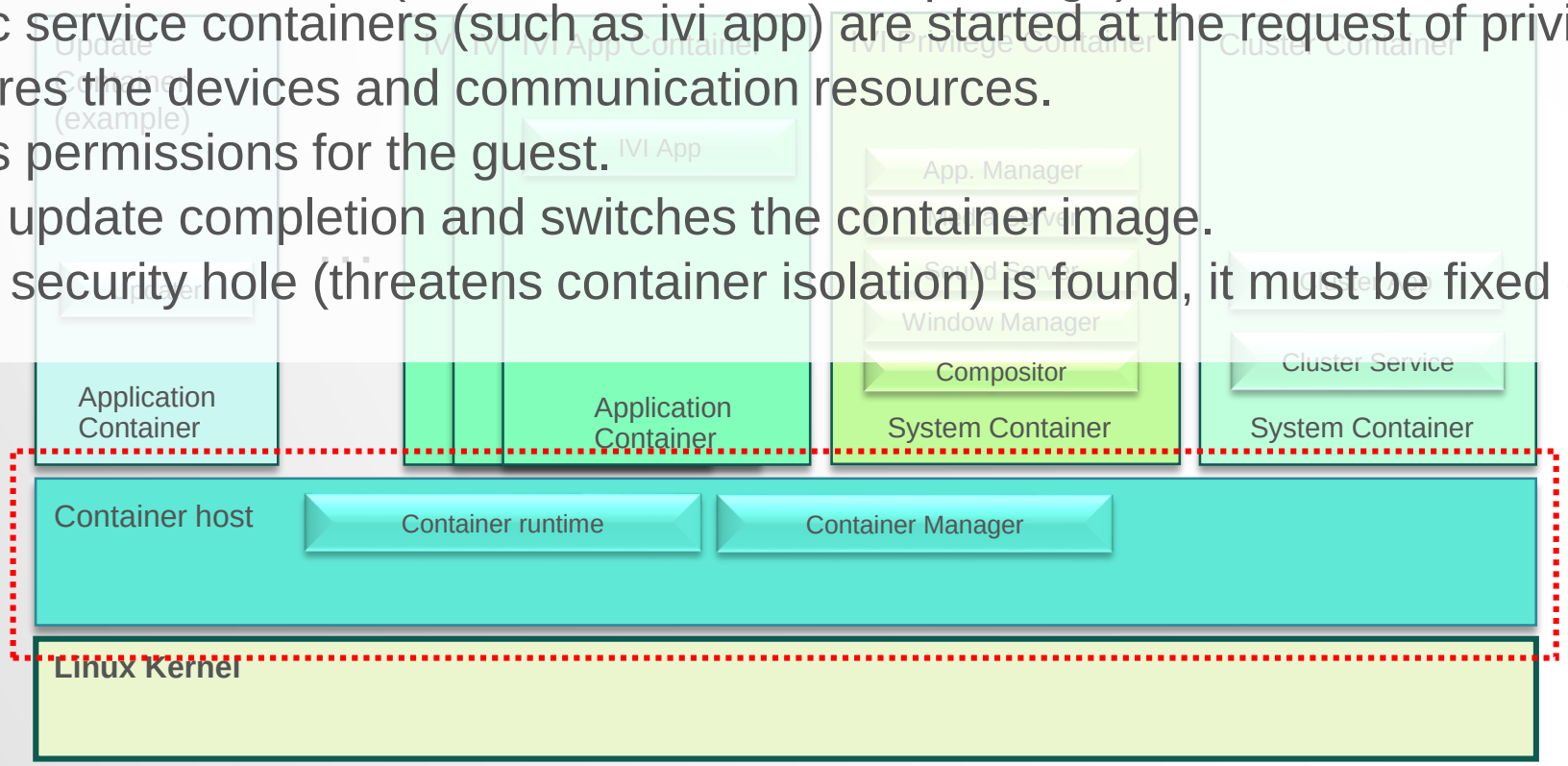
# Container-based architecture

- Basic architecture
  - Following is a breakdown of the abstract architecture:



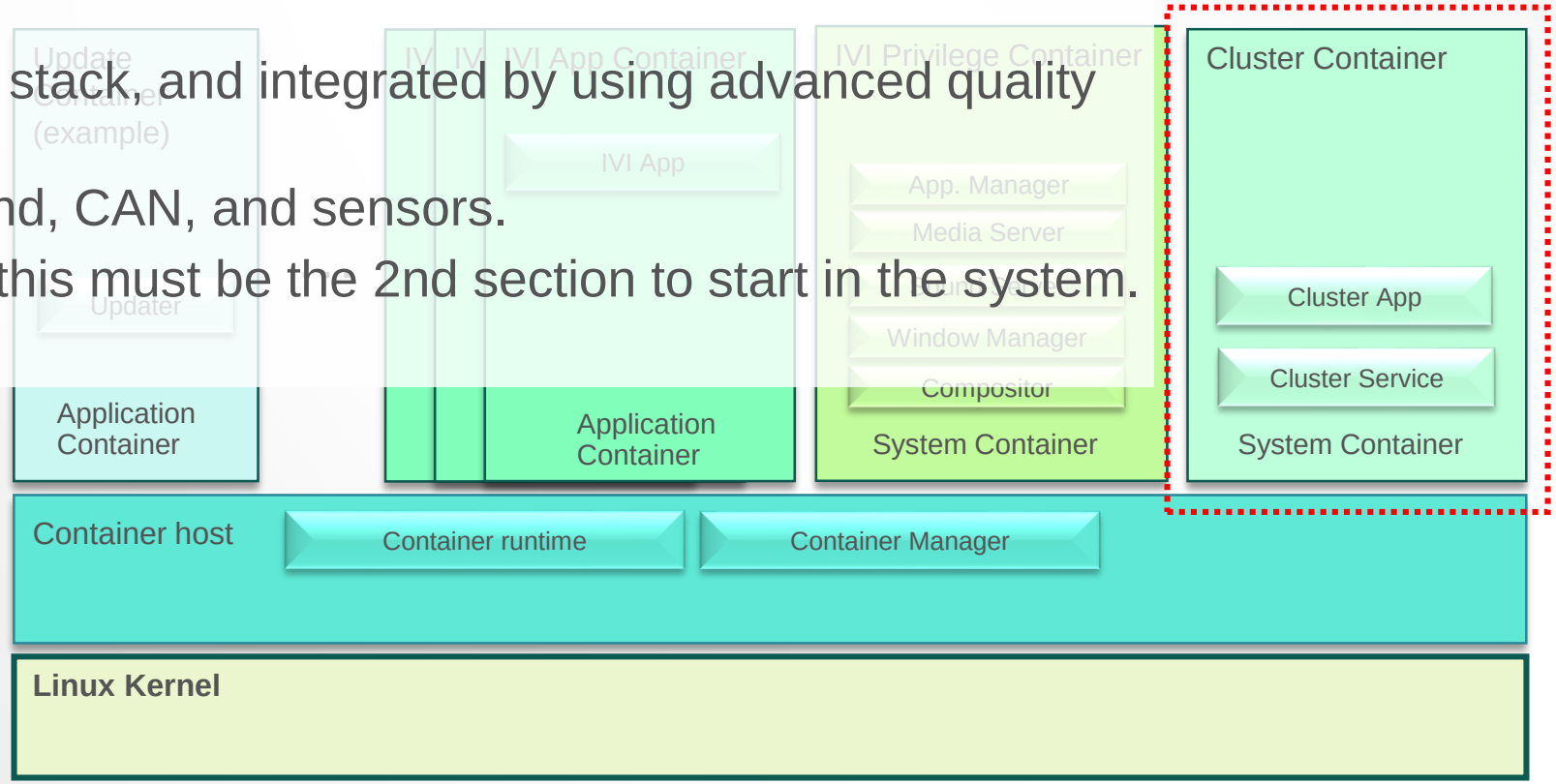
# Automotive use case – Role of each container

- Container host
  - Manages lifecycle of each container.
    - To achieve fast boot-up, it needs to be lightweight.
    - Static service containers (such as cluster and ivi privilege) are launched on boot-up, and dynamic service containers (such as ivi app) are started at the request of privileged container.
    - Configures the devices and communication resources.
    - Controls permissions for the guest.
    - Detects update completion and switches the container image.
    - When a security hole (threatens container isolation) is found, it must be fixed quickly.



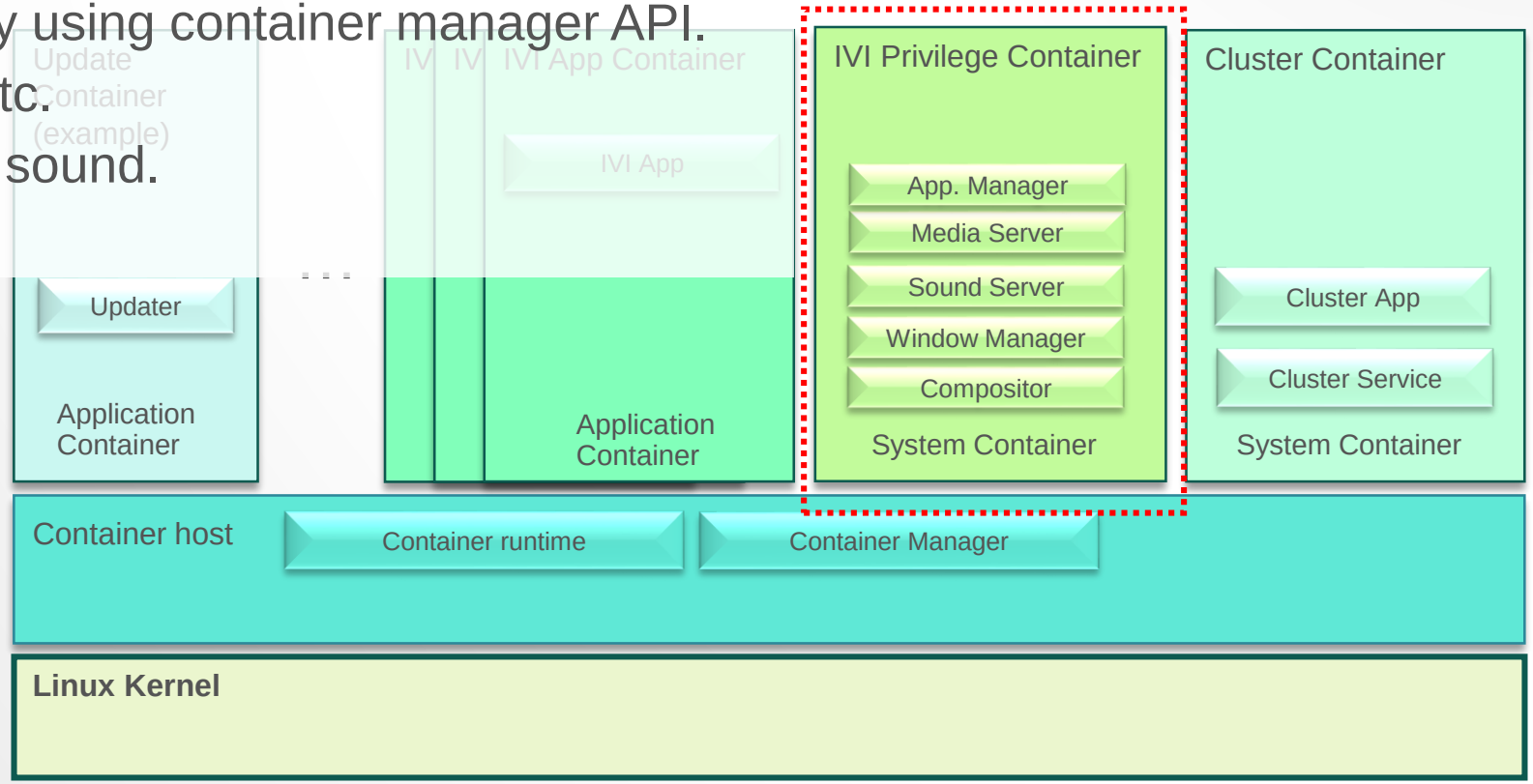
# Automotive use case – Role of each container

- Cluster
  - Provides cluster function
    - Contains the cluster software functions such as meter drawing and fuel calculation.
    - Built with a limited software stack, and integrated by using advanced quality management method.
    - Needs a display, GPU, sound, CAN, and sensors.
    - To realize the fast boot-up, this must be the 2nd section to start in the system.

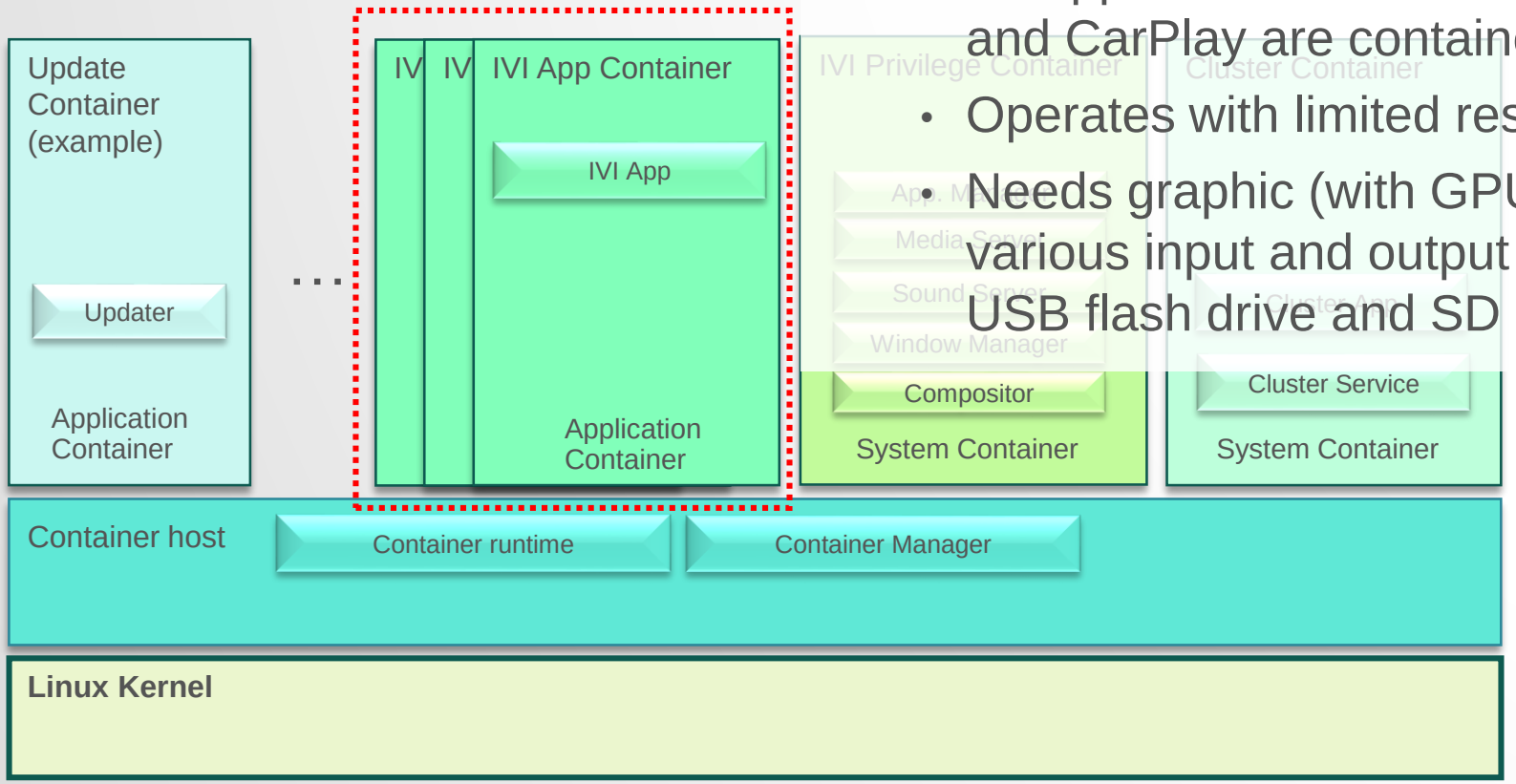


# Automotive use case – Role of each container

- IVI Privilege
  - Responsible for management.
    - Manages sound and graphics for guests except for cluster.
    - Manages IVI applications by using container manager API.
      - Capabilities, resources, etc.
    - Needs a display, GPU, and sound.



# Automotive use case – Role of each container




- IVI App
  - Provides IVI function
    - IVI applications such as silicon audio player, telephone, and CarPlay are contained separately in the unit.
    - Operates with limited resources and permissions.
    - Needs graphic (with GPU), sound, CAN, IP network, and various input and output of dynamic devices (such as USB flash drive and SD card).

# Automotive use case – Many issues

- Key Issue
  - Graphics management
    - How to isolate and share the graphics stacks.
  - Sound management
    - How to isolate and share the sound device.
  - CAN network management
    - How to deliver and hide CAN data.
- Other issues
  - Management of dynamic devices (USB, SD card, etc.)
  - IP network management
  - Container management and update

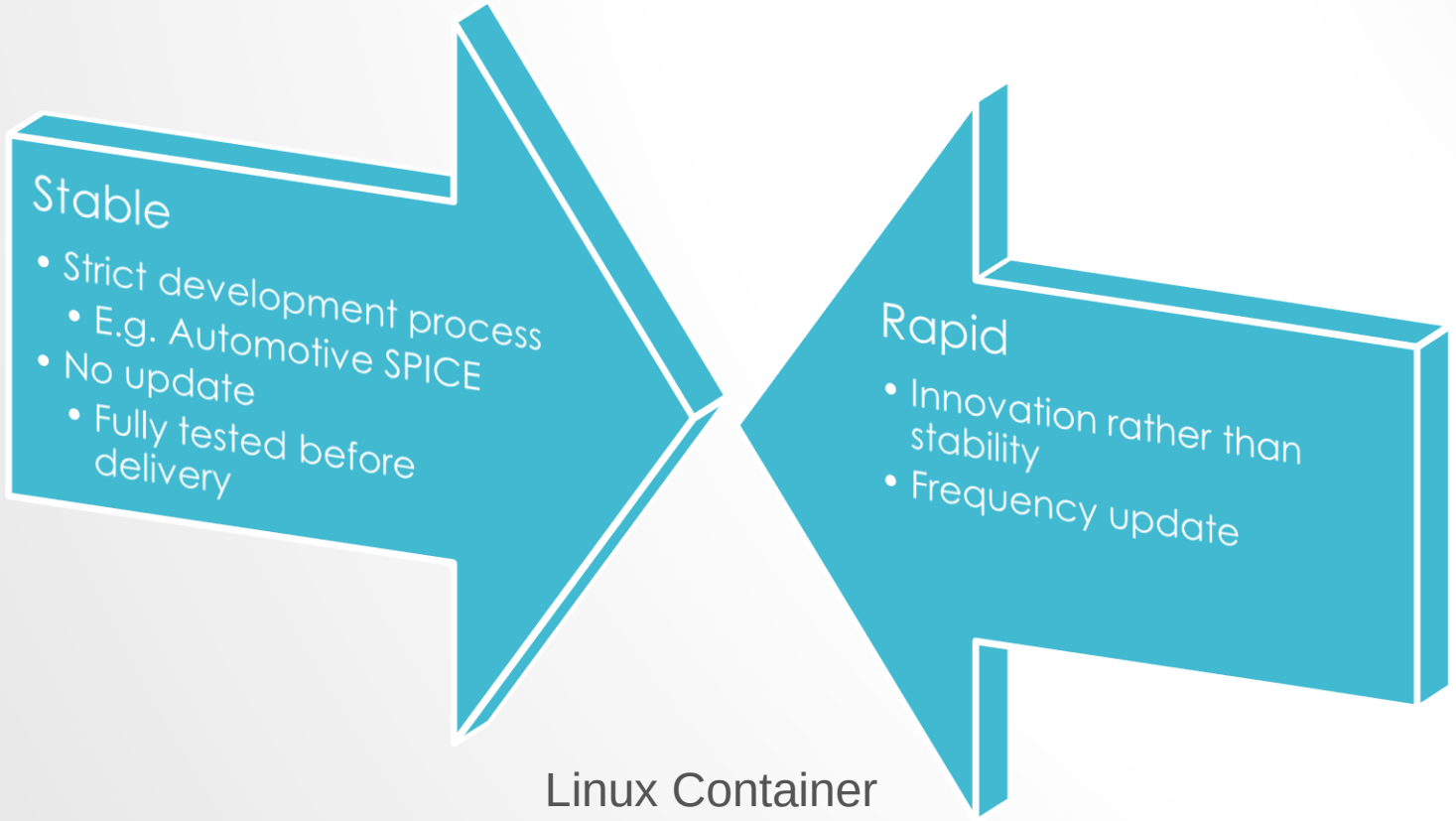
# Outline

- Instrument Cluster EG
- Concept
- **Container Based Architecture**
  - Overview
  - **Key technology : Graphics**
  - Key technology : Sound
  - Key technology : CAN
- Conclusion
- Q&A

# Benefit of Linux container for GUI apps

Contradictions of products for automotive

Crush / Reset unacceptable
↓
Keep code simple & small




Rich functions / Network connection required
↓
Keep code fresh & healthy

Linux Container

Solution: Own runtime (chroot) for eash app
↓
~~Modifying code~~ How to integrate

AUTOMOTIVE GRADE LINUX

# GUI apps: How to do for Linux container ?

**Clients of window system**

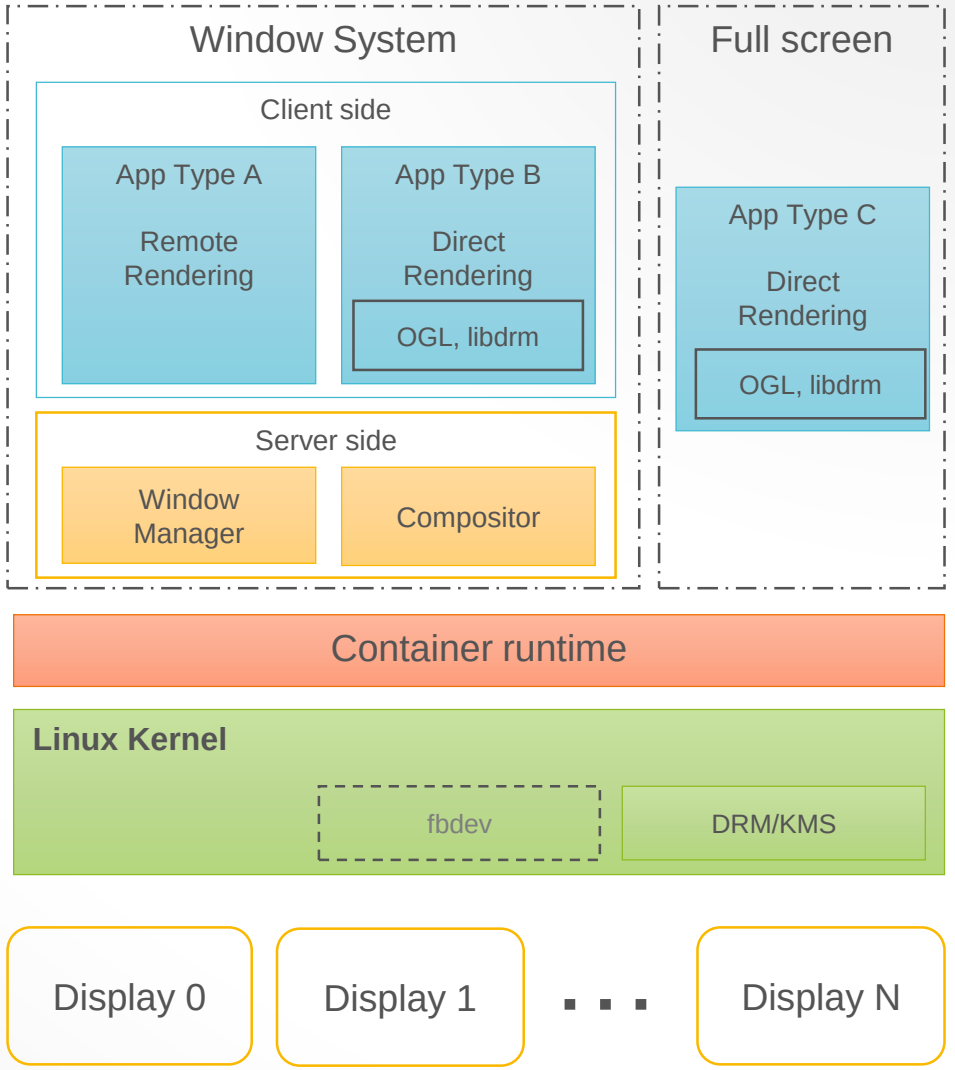
- Server – client model
- Nothing to be changed

**Apps of direct rendering**

- Open GLES abstracts HW/WS dependency
- Nothing to be changed

**Window system (compositor)**

- Depends on system configuration




AUTOMOTIVE GRADE LINUX

# Compositor: How to do?

**Clients of window system**

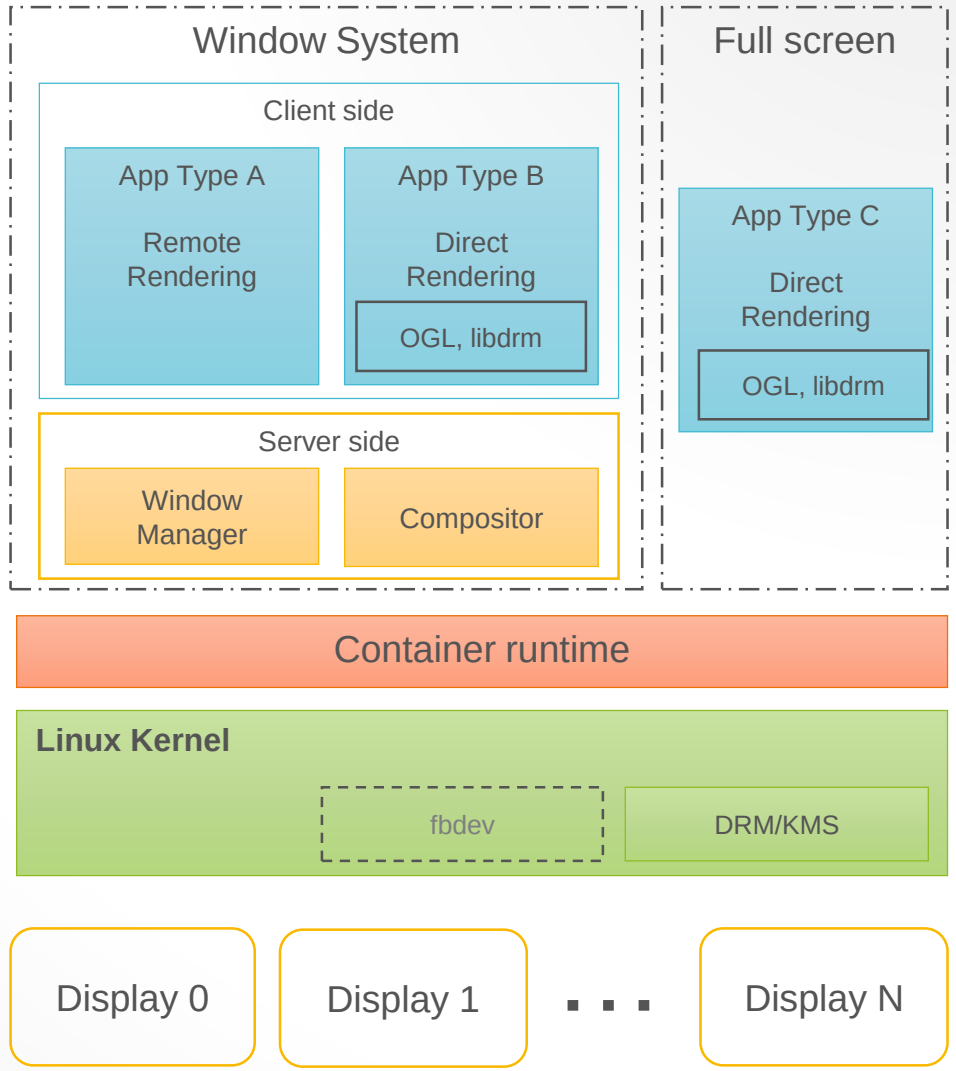
- Nothing to be changed

**Apps of direct rendering**

- Nothing to be changed

**Window system (compositor)**

- Control display, uses GPU directly via DRM/KMS
- Depends on system configuration




AUTOMOTIVE GRADE LINUX

# Output configurations: Separate and Unified

**Separate Display**

- IC uses display 0 only
- IVI uses display 1 only

Compositors separate by container

**Unified Display Simple**

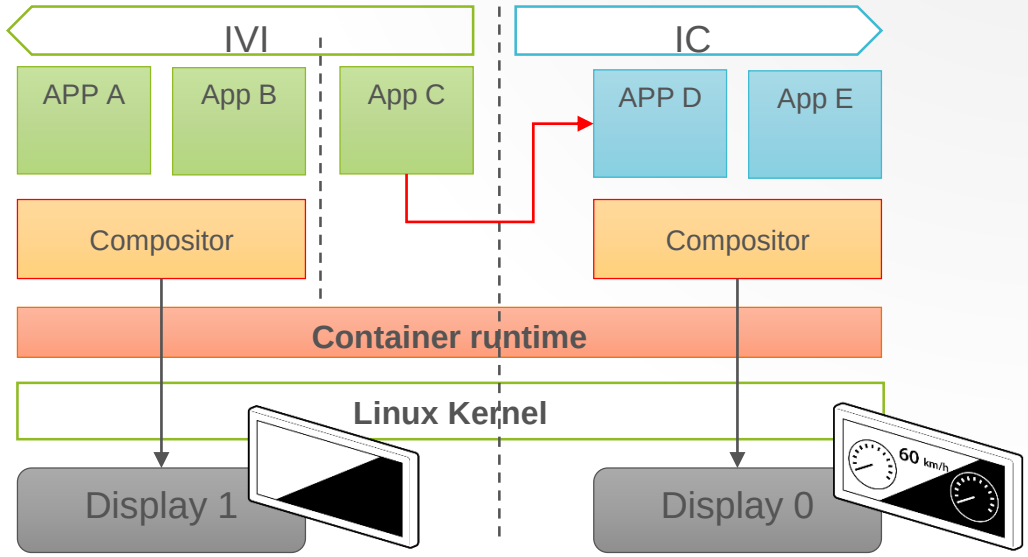
- IC shows additional information from IVI
- Fixed layout

Transfer views from IVI to IC

**Unified Display Advanced**

- IVI and IC overlap freely (flexible layout, seamless HMI)

Nested compositor inside and outside container





AUTOMOTIVE GRADE LINUX

# Output: Separate and Unified

**Separate Display**

- IC uses display 0 only
- IVI uses display 1 only

Compositors separate by container

**Unified Display Simple**

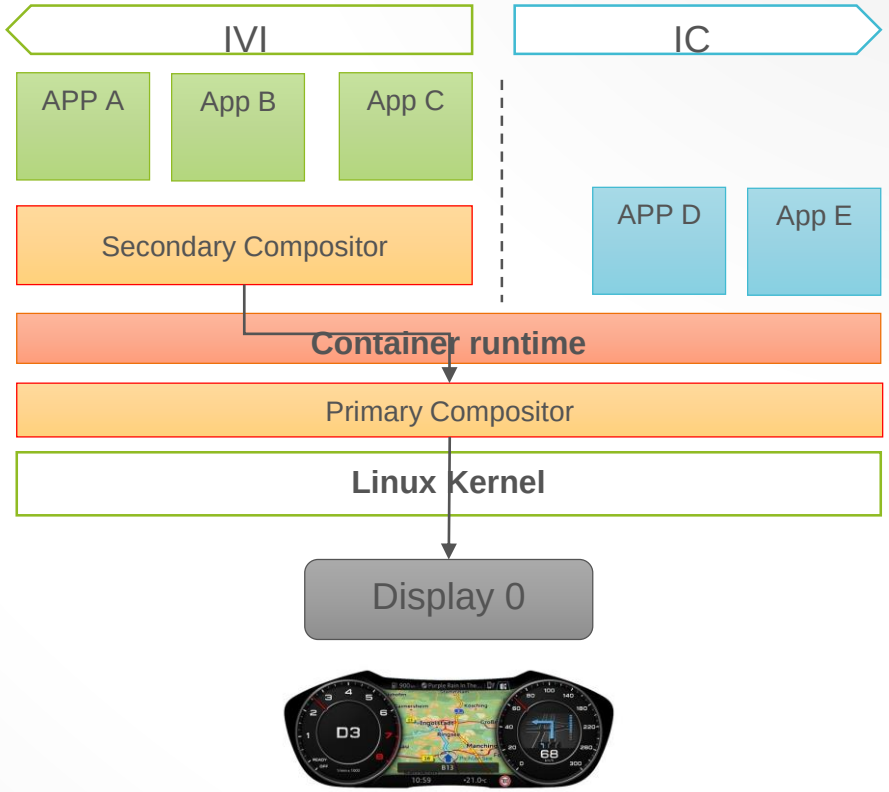
- IC shows additional information from IVI
- Fixed layout

Transfer views from IVI to IC

**Unified Display Advanced**

- IVI and IC overlap freely (flexible layout, seamless HMI)

Nested compositor inside and outside container



# Challenging: Display unified Safety and Non-safety

**Low level composition**

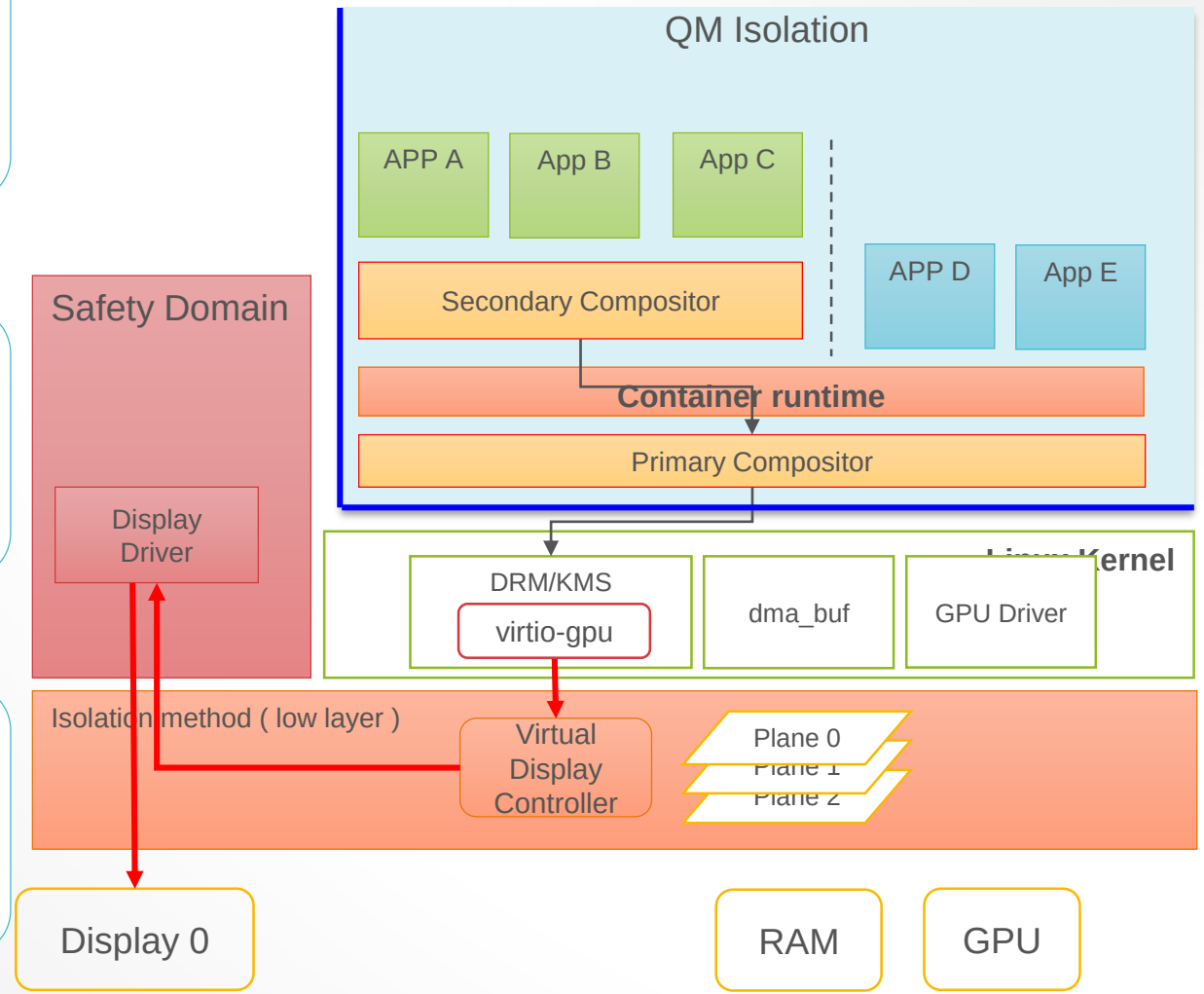
- Composition outside Linux for Safety
- Hardware overlay is preferable

**Virtualized output**

- Output device should be controlled outside Linux

**Collaboration with other projects**

- Virtualization technology (esp. I/O virtualization, e.g. virtio)
- Functional safety




AUTOMOTIVE GRADE LINUX

# Outline

- Instrument Cluster EG
- Concept
- **Container Based Architecture**
  - Overview
  - Key technology : Graphics
  - **Key technology : Sound**
  - Key technology : CAN
- Conclusion
- Q&A

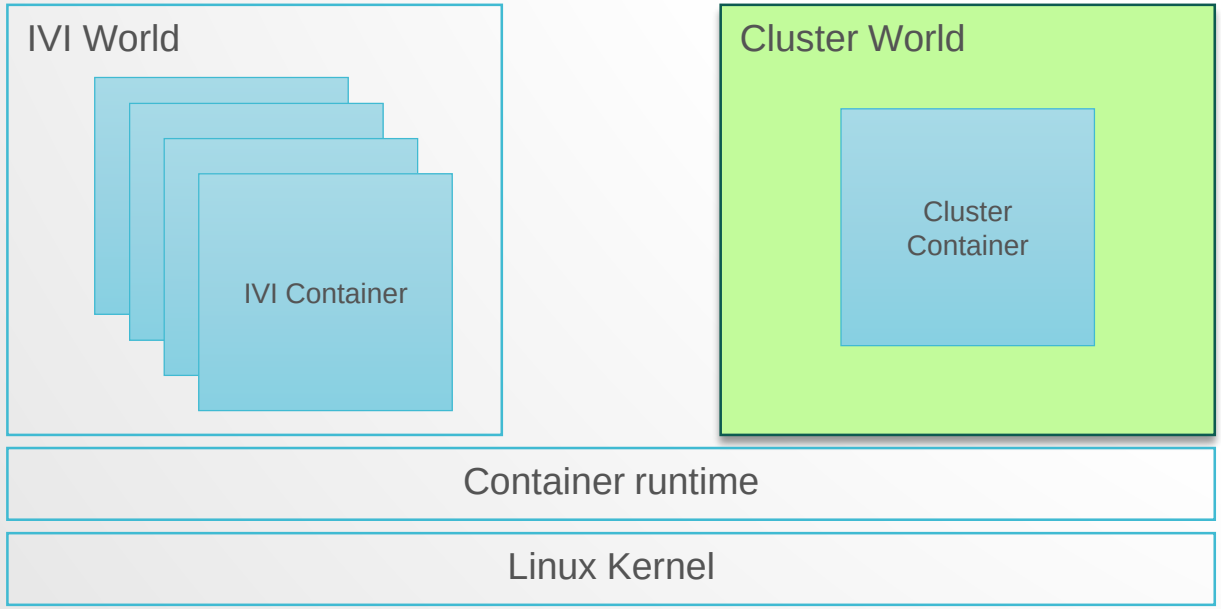
# Requirement – cluster sound -

## Functionality

- Simple sound. Beep, Alert, Winkers .. etc.
- Available to output mix sound

## Spec

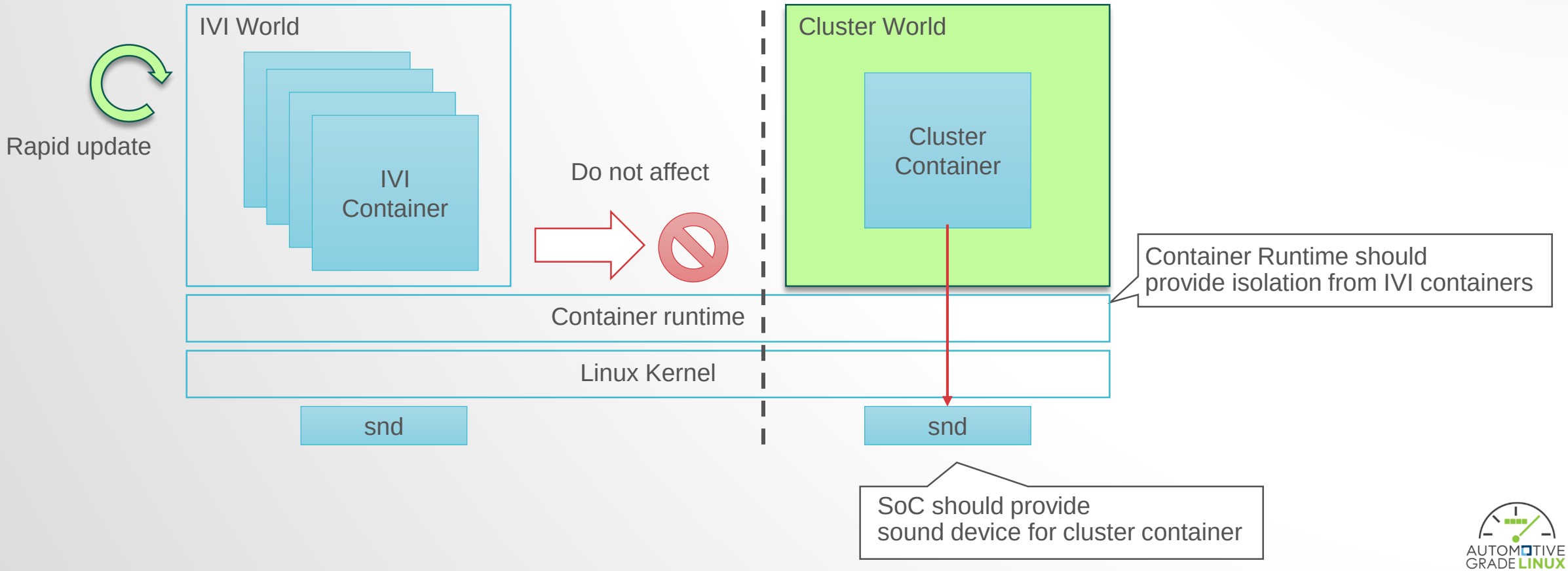
- Each source volumes are fixed. Must not be muted.
- Available within 200msec after startup
- Some source will be the target of ASIL-B according to system design or safety analysis



# Architecture – cluster sound -

Functionality is simple, but fast boot and stability are necessary

- Cluster doesn't need high functional sound server
- To improve the stability, device isolation by container(namespace) is good solution

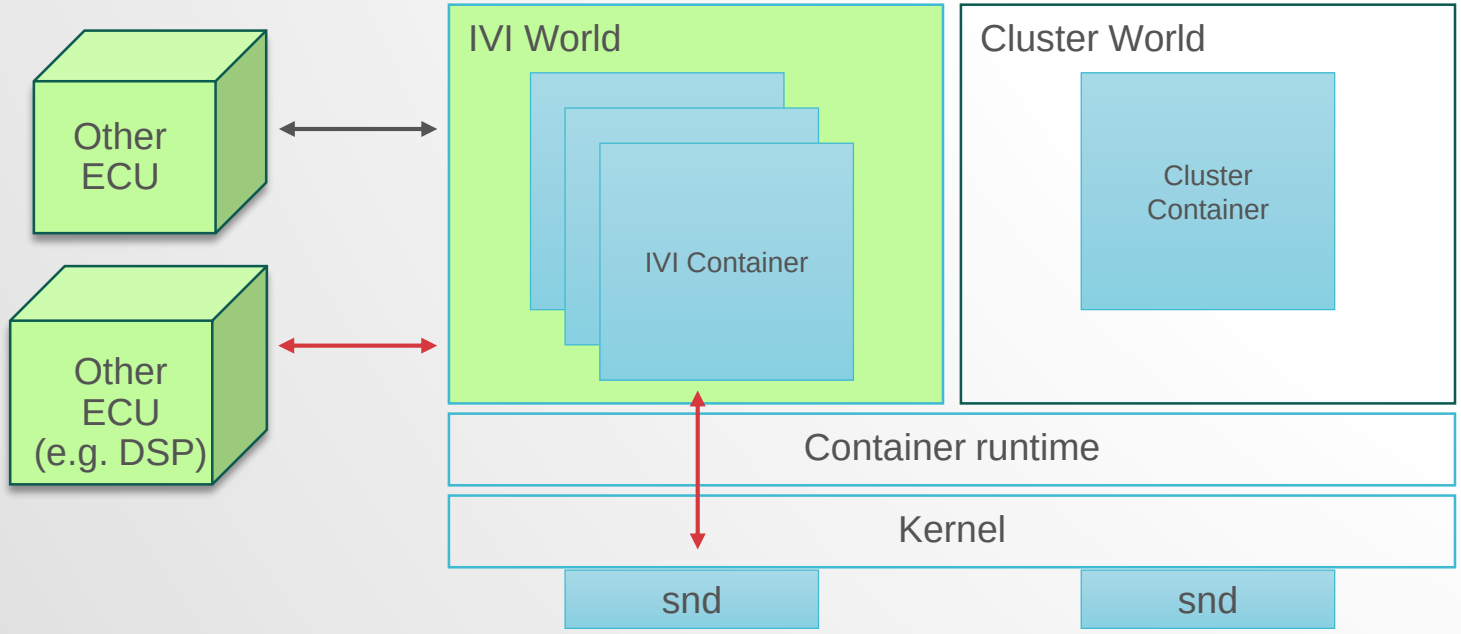


# Requirement – IVI sound 1/2 -

## Functionality

- **Functionality is same as current IVI system**
- Audio policy management is necessary. (※)
  - Active source change : automatically stop old source and play new source when user push buttons.
  - Interrupt source mixing : When car close to cross road IVI system reduce the volume of current source and mix with interrupt source e.g. Navigation Guidance.
- The device should be shared within IVI containers.(mixing/exclusive)
- Several ECUs becomes audio source/sink

- https://wiki.automotivelinux.org/eg-ui-graphics-req-audiorouting
- https://wiki.automotivelinux.org/eg-ui-graphics-req-multimedia
- https://wiki.automotivelinux.org/_media/agl_amm_2017_presentation_nishiguchi_a04.pdf



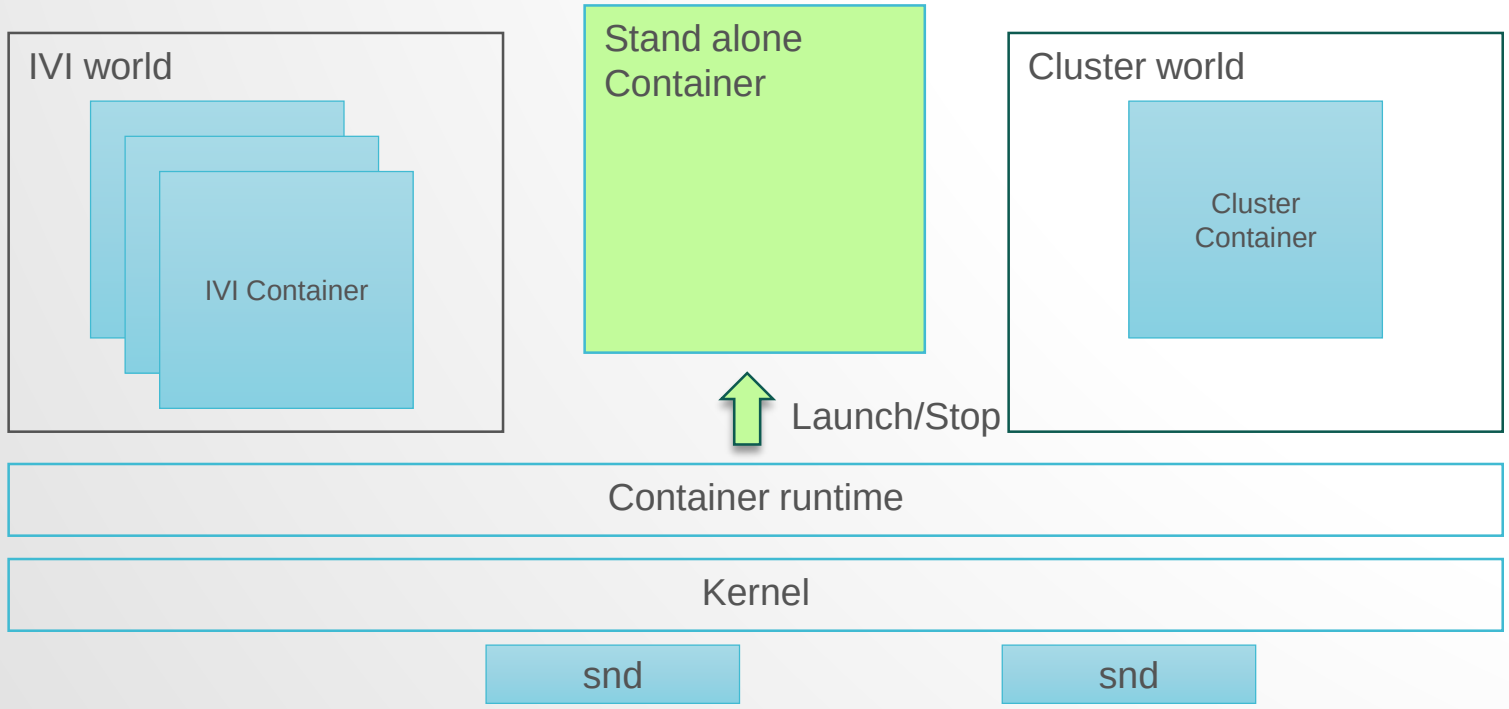
# Requirement – IVI sound 2/2 -

## Functionality

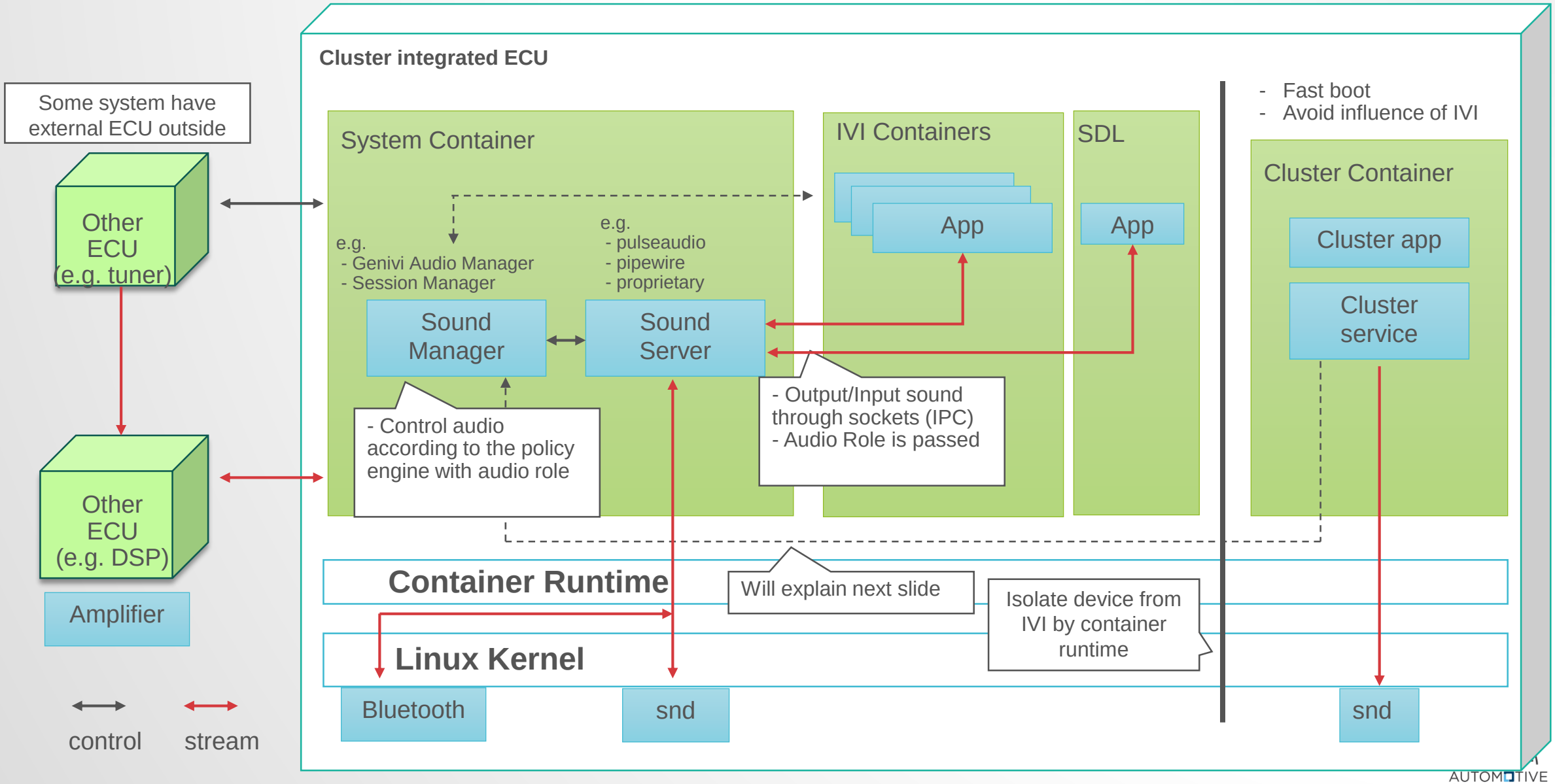
- Launch/Stop container dynamically

  Usecase example

  - User plug smartphone using USB cable
  - Start Smartphone link application container automatically

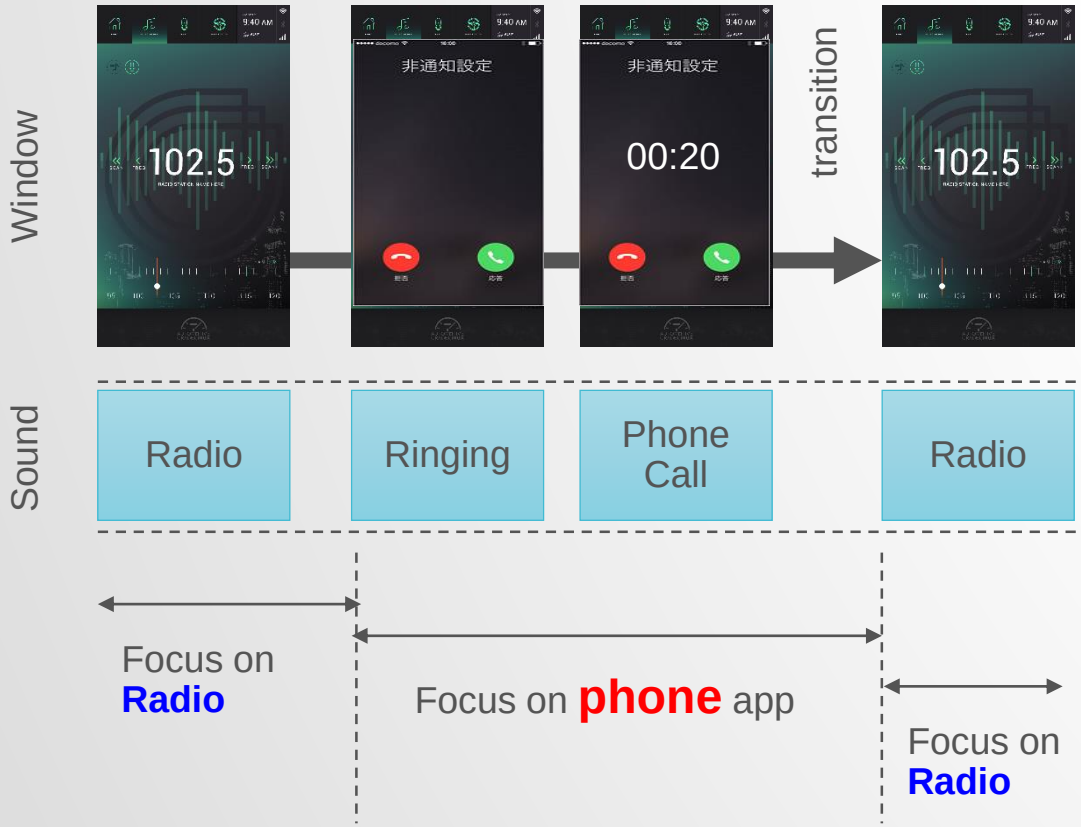


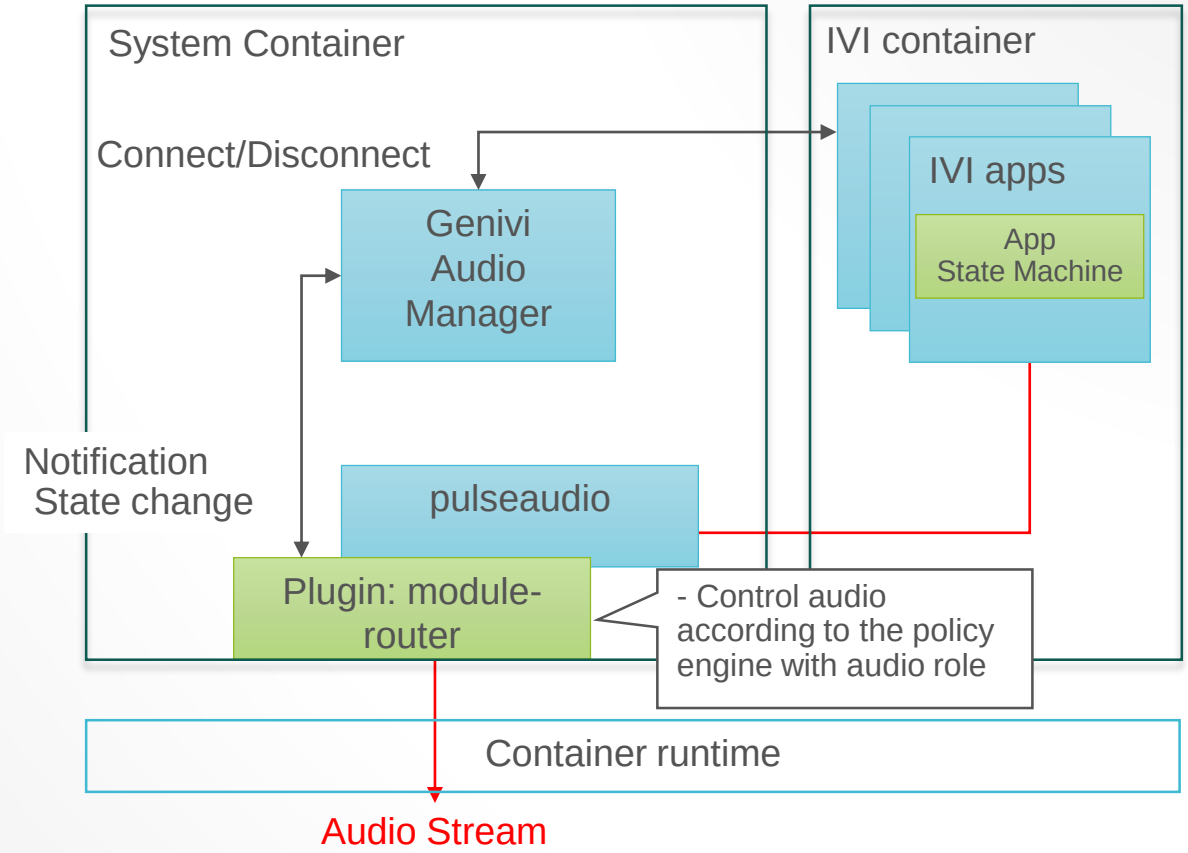
# Architecture – IVI sound -



# Architecture – IVI audio management -

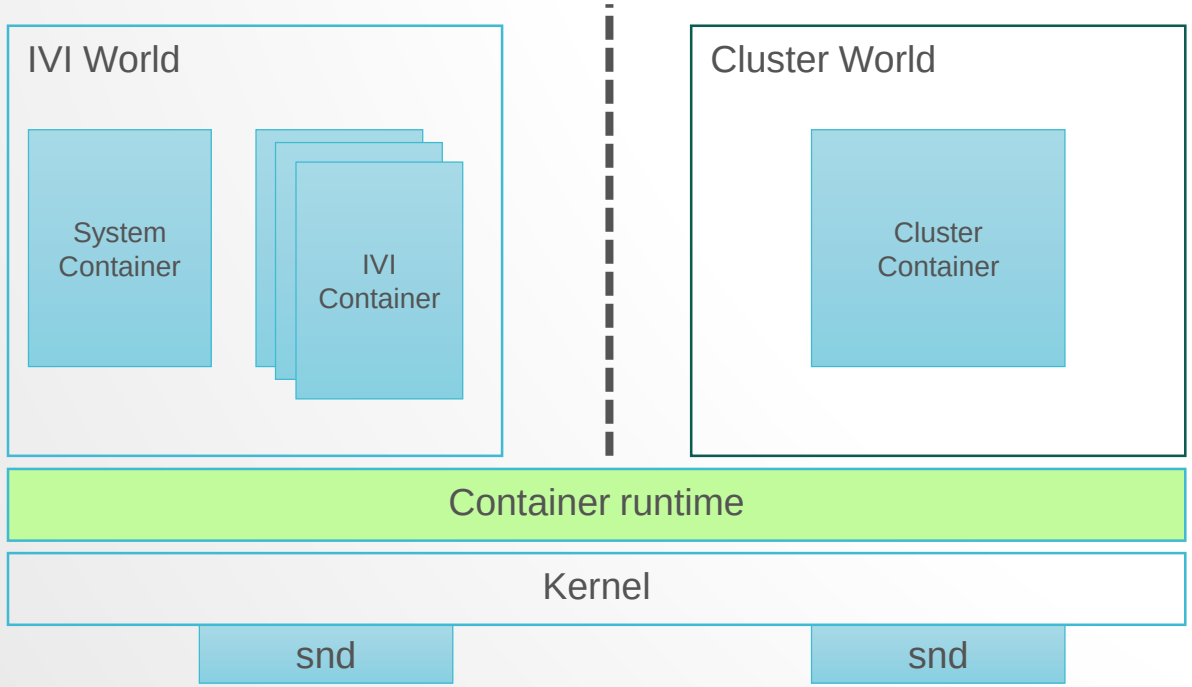
- Audio focus ( Sound Right )



# Conclusion of audio architecture

- Cluster Container will have isolated device by container runtime
- To share the sound device within IVI containers, use sound server in system container
- IVI container architecture is compatible with current IVI sound architecture.

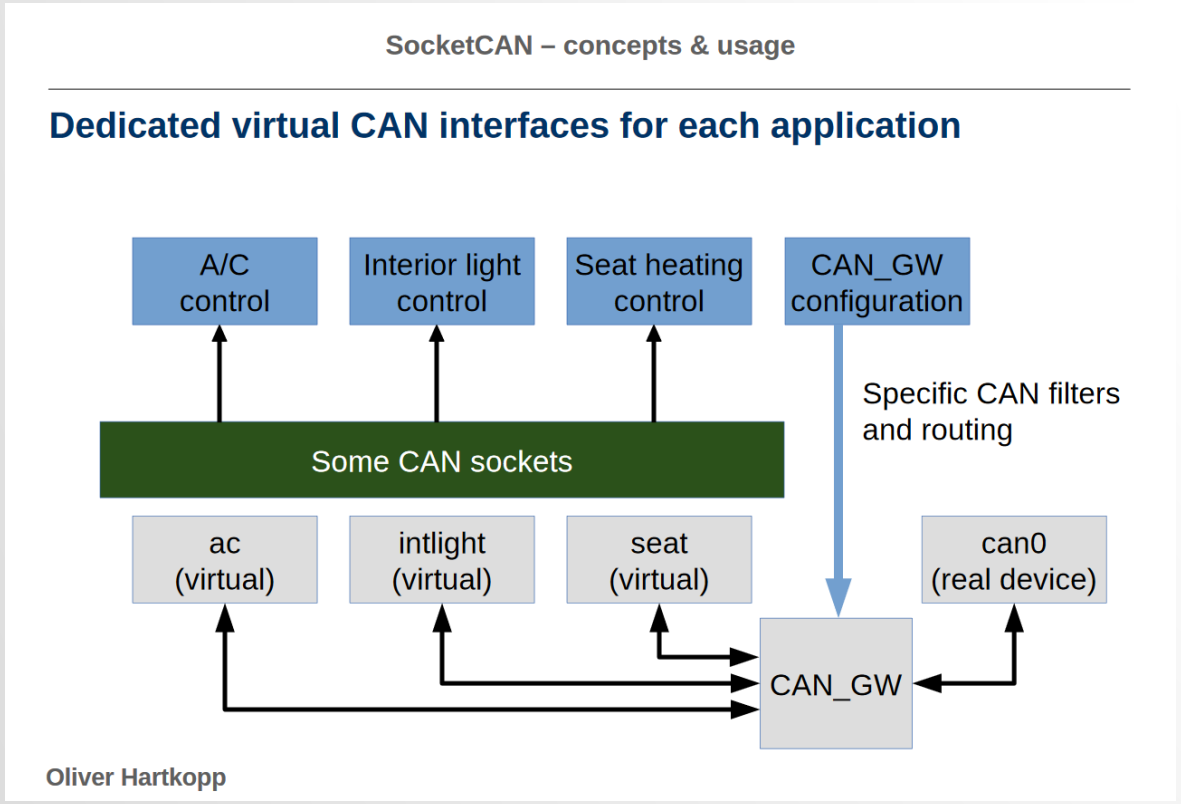


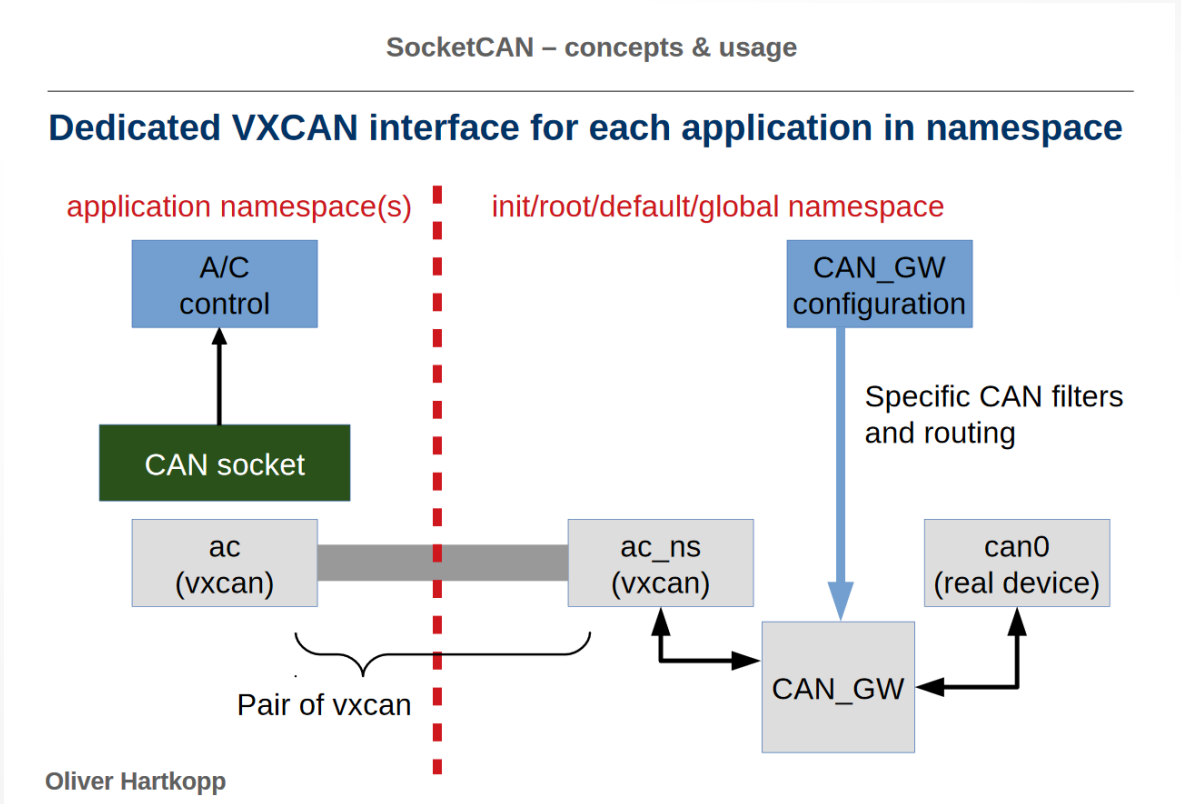
# Outline

- Instrument Cluster EG
- Concept
- **Container Based Architecture**
  - Overview
  - Key technology : Graphics
  - Key technology : Sound
  - **Key technology : CAN**
- Conclusion
- Q&A

# Architecture overview of CAN

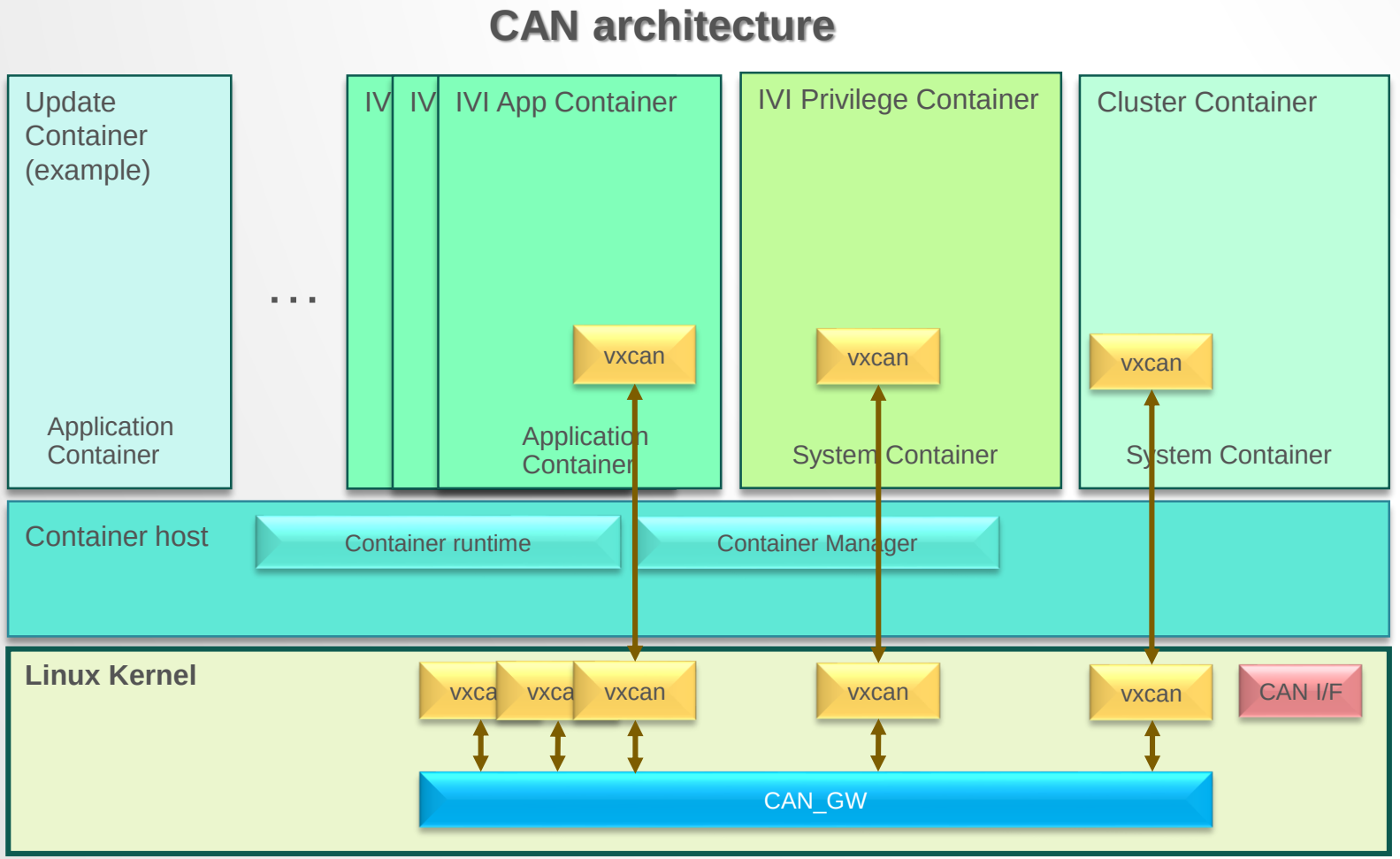
- Currently, Linux can support various CAN network-related functions.
  - https://wiki.automotivelinux.org/_media/agl-distro/agl2018-socketcan.pdf
  - Very good solutions!



# Architecture overview CAN

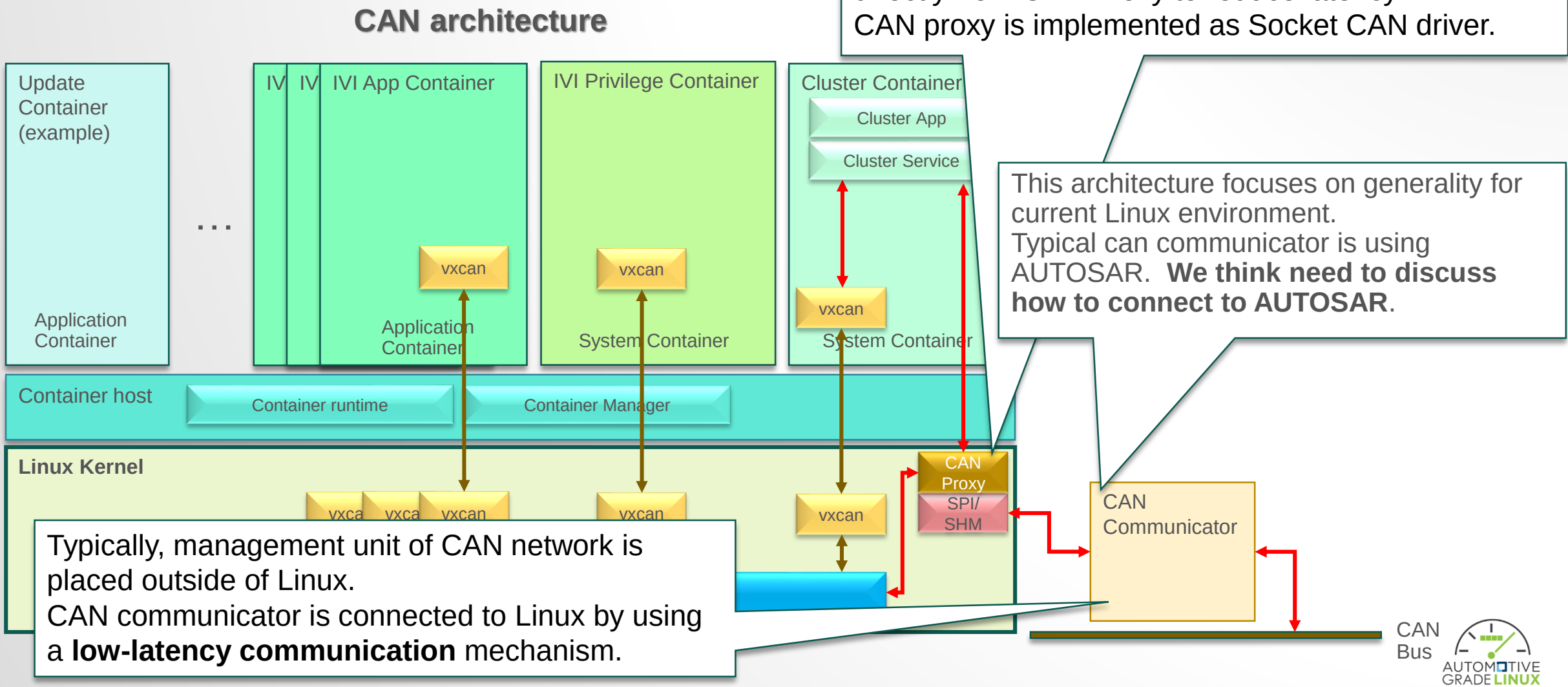
- CAN_GW and vxcan are used for routing CAN data between containers.



# How to connect CAN bus

- How CAN bus is connected?

CAN Proxy transfers data acquired from CAN communicator to a virtual bus in Linux.
Cluster service and application should receive data directly from CAN Proxy to reduce latency.
CAN proxy is implemented as Socket CAN driver.




This architecture focuses on generality for current Linux environment.
Typical can communicator is using AUTOSAR. **We think need to discuss how to connect to AUTOSAR**.

Typically, management unit of CAN network is placed outside of Linux.
CAN communicator is connected to Linux by using a **low-latency communication** mechanism.

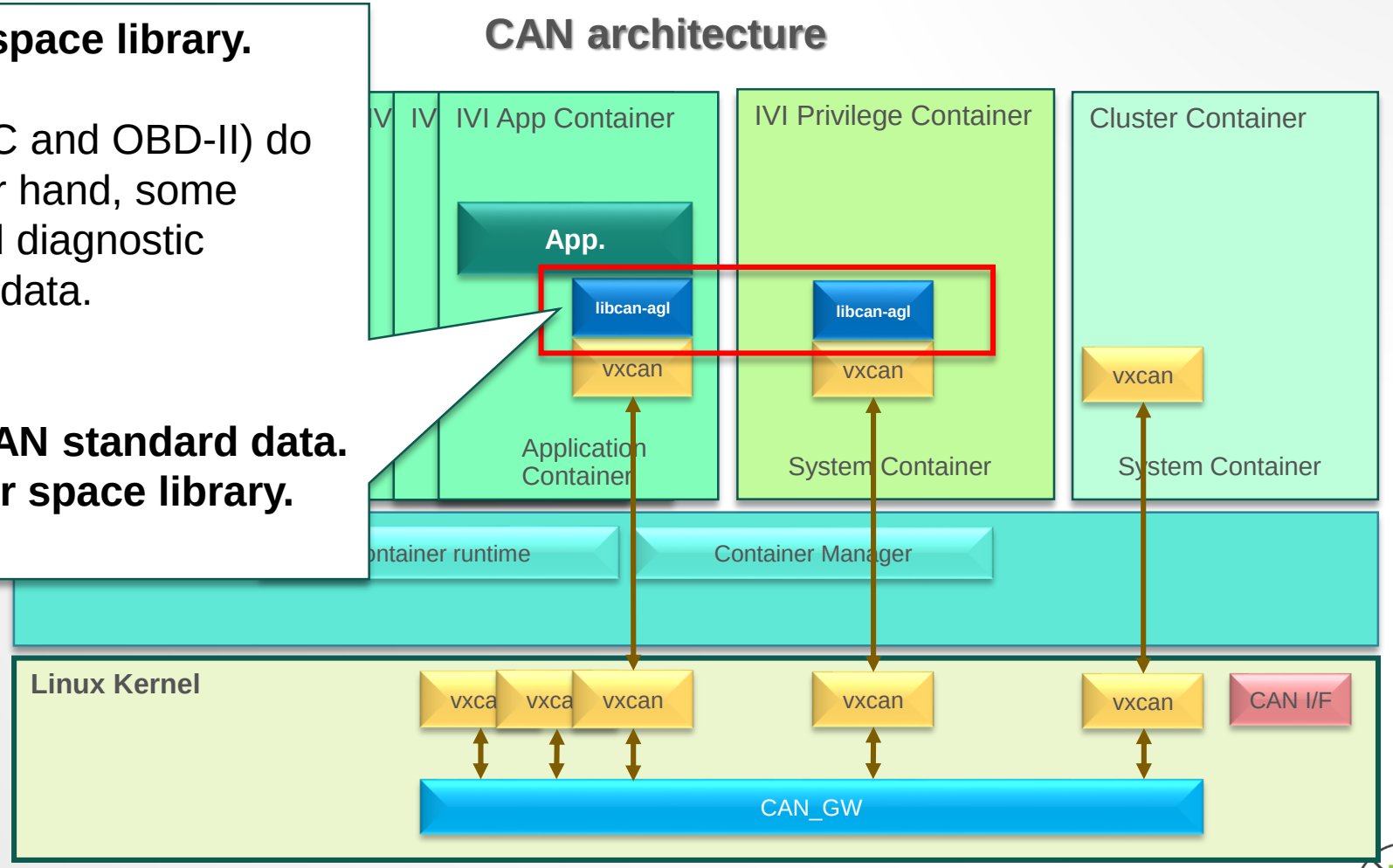
# How to abstract CAN data

- How to abstract CAN data.

**CAN data is abstracted by a user space library.**

CAN standard data (such as OpenXC and OBD-II) do not cover all CAN data.  On the other hand, some functions such as cluster service and diagnostic functions require non-standard CAN data.

**Virtual CAN bus should not use CAN standard data.**
**Conversion is implemented in user space library.**



# Conclusion of CAN architecture

- Summary of CAN
  - CAN architecture is based on Linux socket CAN.
  - CAN_GW and vxcan are used for routing CAN data between containers.
  - We think management unit of CAN network is placed outside of Linux.
  - Conversion of CAN standard data is implemented in user space library.
- Future agenda
  - Typical can communicator is using AUTOSAR.  We think need to discuss how to connect to AUTOSAR.

# Outline

- Instrument Cluster EG
- Concept
- **Container Based Architecture**
  - Overview
  - Key technology : Graphics
  - Key technology : Sound
  - Key technology : CAN
- **Conclusion**
- Q&A

# Conclusion

- Summary of our presentation
  - In this presentation, we described the concept and software architecture of AGL Instrument Cluster EG.
  - Mr. Tanikawa showed the graphics architecture. It's based on nested compositor concept.
  - Mr. Mitunari explained about the sound architecture, which is highly-compatible with current AGL sound architecture.
  - Lastly, we set forth the issues of CAN architecture and other areas.
- Future agenda
  - We will present other issues such as dynamic device, IP network, and container manager in next AMM .
  - For the current status, please visit the following link:
    - https://confluence.automotivelinux.org/display/IC/Instrument+Cluster+Home