



Toward a Service-Based Open Vehicle API for Software-Defined Vehicles

September 8, 2026

Hiroaki Takada

**Center for Embedded Computing Systems (NCES) /
Global Research Institute for Mobility in Society (GREMO)
Nagoya University**

Email: hiro@ertl.jp URL: <http://www.ertl.jp/hiro/>

Self-Introduction

Primary Positions

- ▶ Executive Director & Professor, Center for Embedded Computing Systems (NCES), Graduate School of Informatics, Nagoya Univ.
- ▶ Professor, Global Research Institute for Mobility in Society (GREMO), Institutes of Innovation for Future Society, Nagoya Univ.

Research Areas

- ▶ Real-Time Operating Systems, Software Platforms
- ▶ Real-Time Analysis and Scheduling Theory
- ▶ Functional Safety, Cybersecurity of Embedded Systems
- ▶ Automotive Embedded Systems, In-Vehicle Networks
- ▶ Dynamic Map, Cooperative Autonomous Driving, SDV

What is an SDV? – Our Working Definition

Software-defined Vehicle (broad sense)

- ▶ A vehicle whose behavior, functionality, and value are defined by software.

More practical definition for this talk

- ▶ A vehicle whose functions can be extended or modified after sale by adding or updating software via OTA – beyond IVI.

Why SDV matters: the DevOps analogy

- ▶ Software can be updated continuously.
- ▶ SDV applies DevOps to vehicles.
 - ▶ Continuous improvement of the product after delivery, based on real-world usage.

Evolution Steps of SDV and Open SDV

Step 0: Technical SDV

- ▶ Vehicle supports OTA updates for ECUs beyond IVI.

Step 1: Value Creation through SDV

- ▶ OEM continuously expands functionality and generates recurring revenue through software updates.

OEM-controlled SDV

Step 2: Open SDV (Our term)

- ▶ Allows third-party developers to install applications into the vehicle.

Ecosystem-enabled SDV (true openness)

Impact of Open SDV

Why Open SDV matters

- ▶ In closed SDV, new features require OEM involvement.
- ▶ In Open SDV, third parties can deploy applications without OEM intervention.

*Vehicles become platforms, not just products,
similar to smartphones.*

Value shift enabled by Open SDV

- ▶ Much innovation will come from the ecosystem, not only from OEMs.
- ▶ New recurring revenue opportunities for OEMs and developers.
- ▶ Vehicles gain functions far beyond what OEMs can supply alone.

Challenges Toward Realizing Open SDV

Handling Safety-Related Applications

- ▶ Should third parties be allowed to develop safety-related applications?
 - ▶ If NO → ecosystem shrinks.
 - ▶ If YES → responsibilities and regulations must be clearly defined.

Sustainable Business Model for Third-Party Developers

- ▶ There is a concern that SDV applications may not be profitable, because drivers spend far less daily time in vehicles than in smartphones.
 - ▶ Business models can rely on services, partnerships, or data – not necessarily on selling applications.
- ▶ Safety restrictions directly shape what kinds of business models are possible.

Vehicle API and Standardization

What is a Vehicle API?

- ▶ An interface through which applications access the vehicle

Why do Vehicle APIs need standardization?

- ▶ Third-party service providers need their applications to work across many vehicles.
 - ▶ This requires the same API to work across differences in vehicles and OEMs.
- ▶ Vehicle-independent application development also improves SDV software development efficiency.
- ▶ As OEMs collaborate with autonomous-driving system companies, standardizing the interface between the system and the vehicle makes it easier to:
 - ▶ Adopt different autonomous-driving systems
 - ▶ Avoid lock-in to a specific supplier and promote healthy competition

What If the Automotive Industry Does Not Standardize Vehicle APIs?

- ▶ Vehicle APIs defined by smartphone-platform companies (Google, Apple, Huawei) could become the de facto standard.
 - ▶ The vehicle could become a peripheral device for the smartphone.
 - ▶ Design freedom for hardware (vehicles and their electronic architectures) could be constrained.
- “Software Restricted Vehicle”*
 - ▶ SDVs could be absorbed into smartphone ecosystems.
- ▶ In fact, smartphone-platform companies are steadily moving in this direction.

Key Requirements for Vehicle APIs

Enable a viable third-party business

- ▶ Allow applications developed once to be deployed across many vehicles.
- ▶ Be easy to use for third-party application developers, who may not be automotive experts.

Do not prevent OEM differentiation

- ▶ Do not restrict design freedom for hardware.
 - ▶ In other words, be applicable to diverse vehicles and E/E architectures.

Provide a mechanism for handling safety-related applications

- ▶ Provide a mechanism that allows the OEM and the user to control which applications may access which APIs.
- ▶ Accommodate differences and changes in regulations and public acceptance.

Open SDV Initiative – Motivation and Activities

Why we launched Open SDV Initiative

- ▶ Momentum for defining a common service-based Vehicle API has *not yet emerged* among OEMs.
 - ▶ The slow progress of COVESA VSC reflects this situation.
- ▶ We decided to define it ahead of OEM-led standardization, launching the Open SDV Initiative in Oct. 2024.

Current activities

- ▶ Definition of the Service-Based Vehicle API (Logical Level), named “*Open SDV API*”
 - ▶ Early drafts released in Mar. 2025, Sep. 2025, and Mar. 2026 (under ongoing refinement).
- ▶ Prototype implementations
 - ▶ Simulation-based testing and initial in-vehicle trials.
 - ▶ Full-scale implementation needs more resources.

Open SDV API – Design Approaches

Vehicle Configuration Description

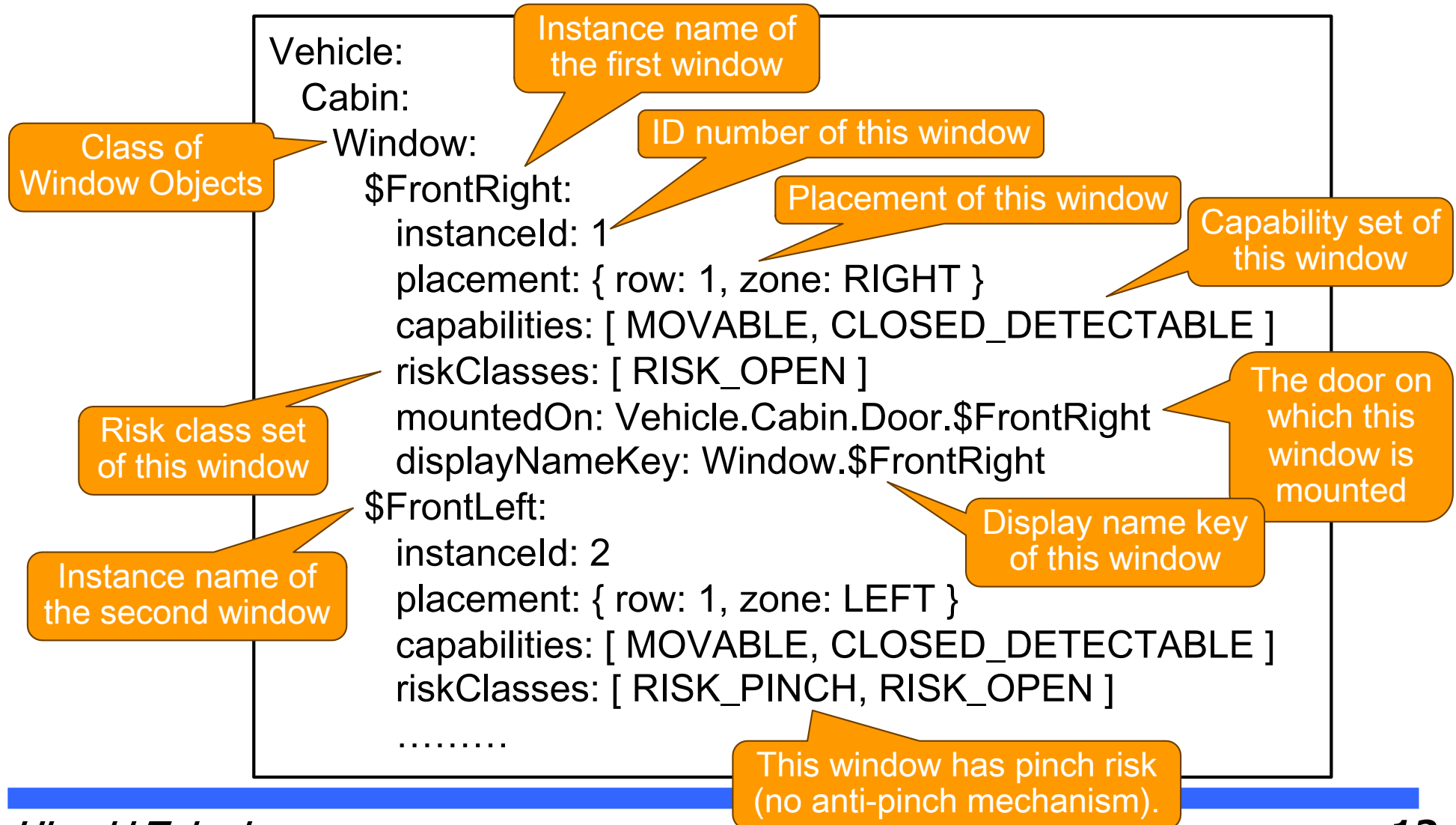
- ▶ Define a common way to describe differences among vehicles.
- ▶ The API does not need to completely hide all vehicle differences.
- ▶ Applications can adapt to vehicle-specific capabilities using the configuration description.

Service-Based API for Third-Party Developers

- ▶ Provide an easy-to-use API for application developers.
- ▶ For vehicle control, service-based APIs are more suitable than signal-based APIs.
- ▶ Arbitration and locking are also easier to express at the service level.

Vehicle Configuration Description – Example

- ▶ An example of vehicle window configuration described in YAML



Open SDV API – Additional Design Policies

High-Level Abstraction Enabling OEM Differentiation

- ▶ Keep the API abstract enough not to constrain implementation methods.
- ▶ Leave room for OEM differentiation in vehicle and Vehicle OS implementations.

Independence from E/E Architecture

- ▶ Do not restrict the ECU on which applications run.
- ▶ Consider efficient implementation on current E/E architectures.

Define the Logical API First

- ▶ Define the API independently of programming languages, operating systems, and communication technologies.

Core and Extended Vehicle APIs

- ▶ Core Vehicle API: abstracts differences among vehicles.
- ▶ Extended Vehicle API: provides convenient functions for application developers.

Signal-Based API vs. Service-Based API

Signal-Based API

- ▶ Applications read and write signals representing vehicle states.
 - ▶ Examples: vehicle speed, door status, window position
- ▶ COVESA VSS/VISS takes this approach.

Service-Based API

- ▶ Applications request specific vehicle operations or functions.
 - ▶ Examples: open a window, lock a door, control vehicle motion
- ▶ COVESA VSC takes this approach.

Which abstraction is more appropriate for application developers?

Limitations of Signal-Based APIs for Vehicle Control

- ▶ Signals represent states, while applications often request operations. It can be difficult to determine whether an operation:
 - ▶ is still in progress,
 - ▶ has completed,
 - ▶ or has been overridden.
- ▶ It is difficult to express why an operation was overridden.
- ▶ Different ways of performing the same operation are difficult to represent.
- ▶ Adding signals for such semantics makes the API increasingly complex.

Arbitration and Access Control Are Essential

Arbitration

- ▶ Multiple applications may try to control the same vehicle resource.
- ▶ Higher-priority applications must be able to override lower-priority ones.
- ▶ Lower-priority applications must sometimes be prevented from interfering.

Access Control

- ▶ The platform must control which applications may access which resources.
- ▶ Safety-related operations may be permitted only for qualified applications.

These are part of the semantics of vehicle operations, not merely vehicle data.

Approaches to Introducing Arbitration

- ▶ How can arbitration be added to a signal-based API?

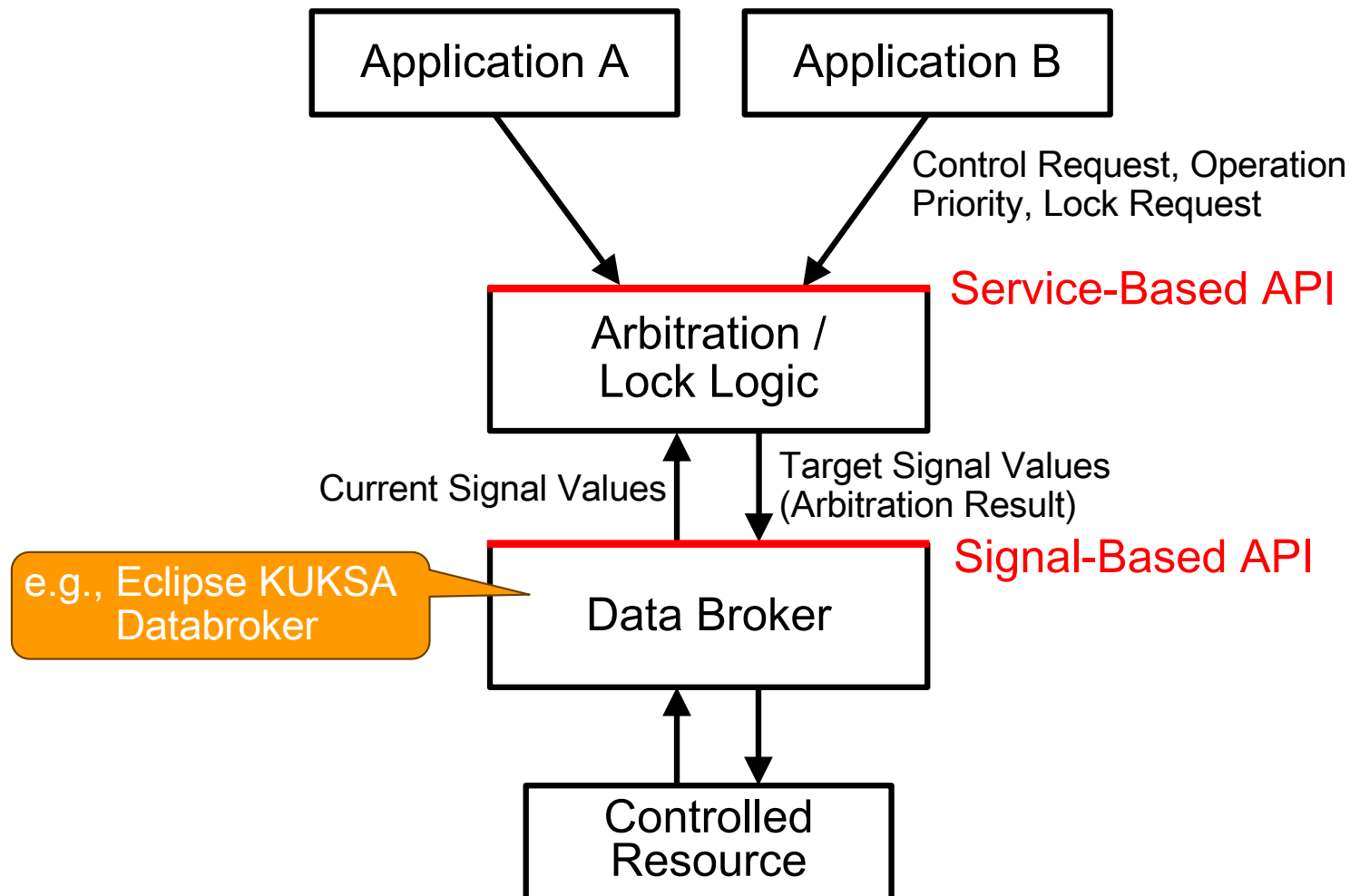
Approach 1: Add arbitration mechanisms to a signal-based API

- ▶ Add signals, or metadata attached to signals, representing operation priority, lock status, etc.
- ▶ This makes the signal model increasingly complex.
 - ▶ The simplicity of the signal-based approach is gradually lost.

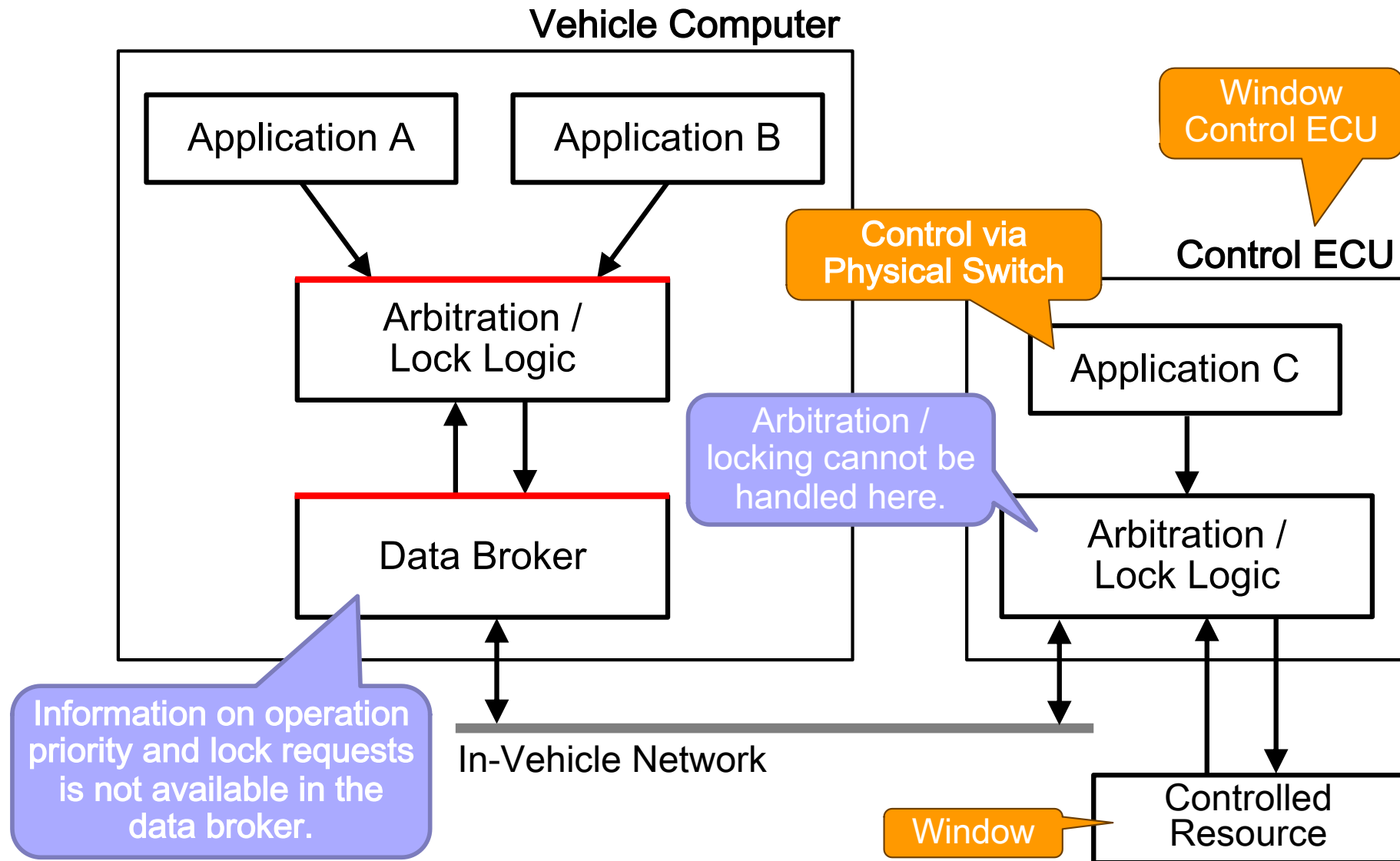
Approach 2: Add a service-based API layer above the signal-based API

- ▶ Introduce a service-based API above the signal-based API.
 - ▶ Implement arbitration and locking at the service level.
- ▶ The lower signal-based API can continue to be used for communication with vehicle resources.

Approach 2: Add a service-based API layer above the signal-based API



Challenge of Approach 2: Distributed Arbitration



Risk Control – Handling Safety-Related Functions

Risk-involving APIs are still defined

- ▶ Safety-related functions remain part of the API.

Vehicle describes its residual risks

- ▶ Residual risks (risks the vehicle cannot fully mitigate) differ from vehicle to vehicle.

App describes which risks it mitigates

- ▶ Each app declares the risks it can handle. The OEM assesses whether the app's risk mitigation is sufficient.

API call is permitted only if the app covers the vehicle's residual risks and user consent is given

This risk-control mechanism is defined in the API spec

- ▶ The spec defines how risks are described and how the platform verifies them.

Modularizing Safety Responsibility with Risk Classes

- ▶ Defining a Vehicle API also means defining the responsibility boundary between the application and the vehicle platform.
- ▶ A single fixed responsibility partition would limit the vehicles to which the API can be applied.
- ▶ Therefore, the Open SDV API allows multiple responsibility partitions.
- ▶ Risk classes represent which safety risks remain the responsibility of the application.

Risk classes modularize safety responsibility between the vehicle and the application.

Concluding Remarks

A standardized service-based Vehicle API is essential for the automotive industry

- ▶ It must work across diverse vehicles while still enabling OEM-specific differentiation.
- ▶ It must provide application arbitration and clear responsibility partitioning for safety.

We are defining such an API ahead of OEM-led standardization

- ▶ Through the Open SDV Initiative, we are prototyping and refining this approach.

We seek active collaboration with the Eclipse SDV community

- ▶ We would be happy to work together – either by joining our initiative or by collaborating within Eclipse SDV activities.