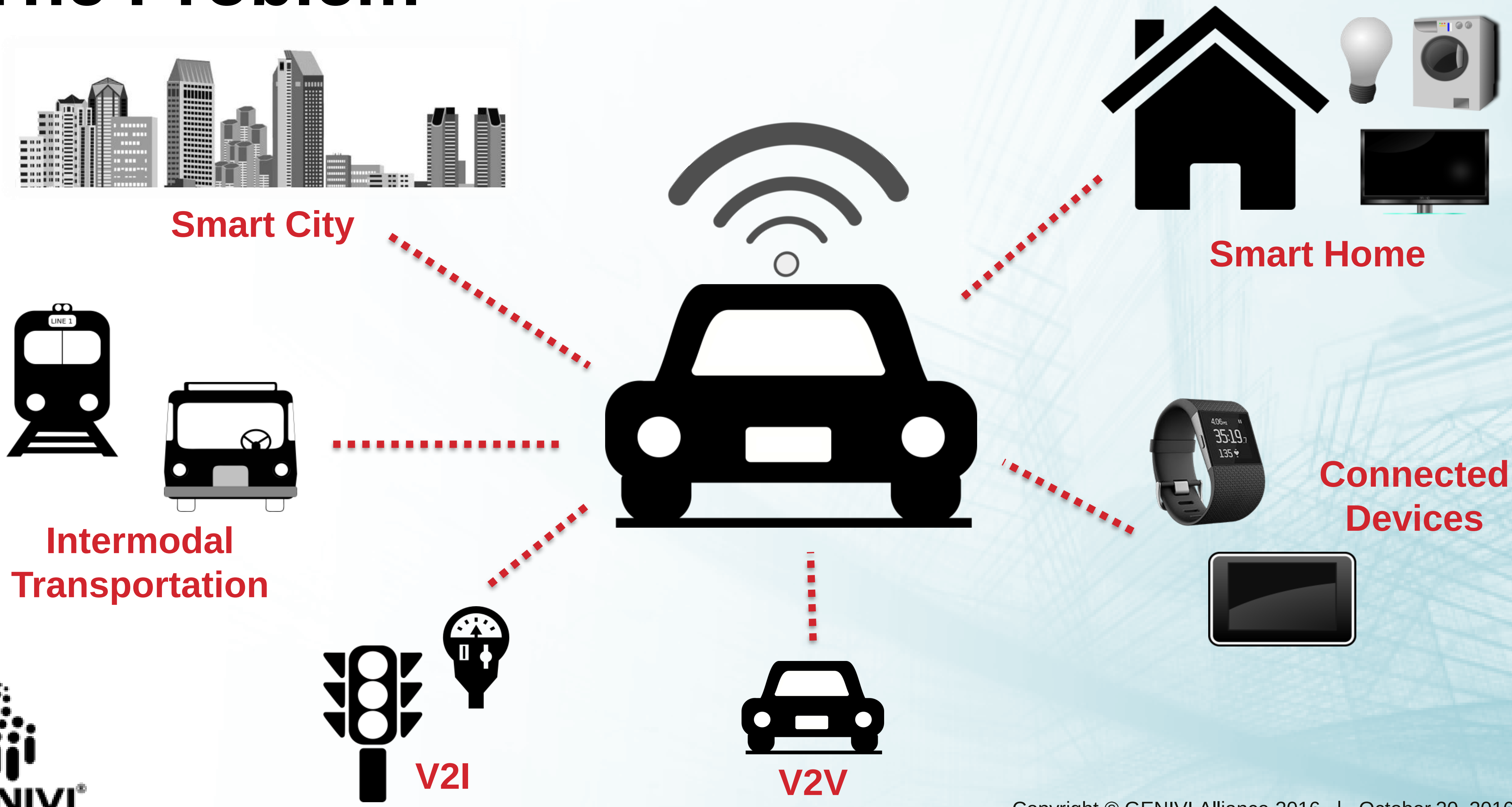
Vehicle Data Interfaces

May 11, 2017 | Enabling the Connected Car

Rudolf J Streif

Networking Expert Group Lead, GENIVI Alliance

The Problem



Smart City

Smart Home

Intermodal Transportation

Connected Devices

V2I

V2V

The Challenge

- Providing access to vehicle status information and data to cloud services, web applications, mobile devices and more.
- There is no standard convention for a vehicle data API.
- OEMs wish to be able to easily extend a standard API with signals and controls for their purposes.
- Security mechanisms are required that provide authentication and authorization to access vehicle signals and control.
- Design that decouples signal interface from the electrical architecture of the vehicle.

Conventional Approach – “Fat API”

- An API for every signal or control:

```
var vehicle = navigator.vehicle;
vehicle.vehicleSpeed.get().then(function (vehicleSpeed) {
    console.log("Vehicle speed: " + vehicleSpeed.speed);
}, function (error) {
    console.log("There was an error"); });
var vehicleSpeedSub = vehicle.vehicleSpeed.subscribe(function (vehicleSpeed) {
    console.log("Vehicle speed changed to: " + vehicleSpeed.speed);
    vehicle.vehicleSpeed.unsubscribe(vehicleSpeedSub);
});
```

- Issues with this approach:
  - Addition of new signals and controls requires change of the specification.
  - Challenges maintaining backwards compatibility.
  - Complexity in providing per-API authorization and access control.
  - Single end-point addressing.

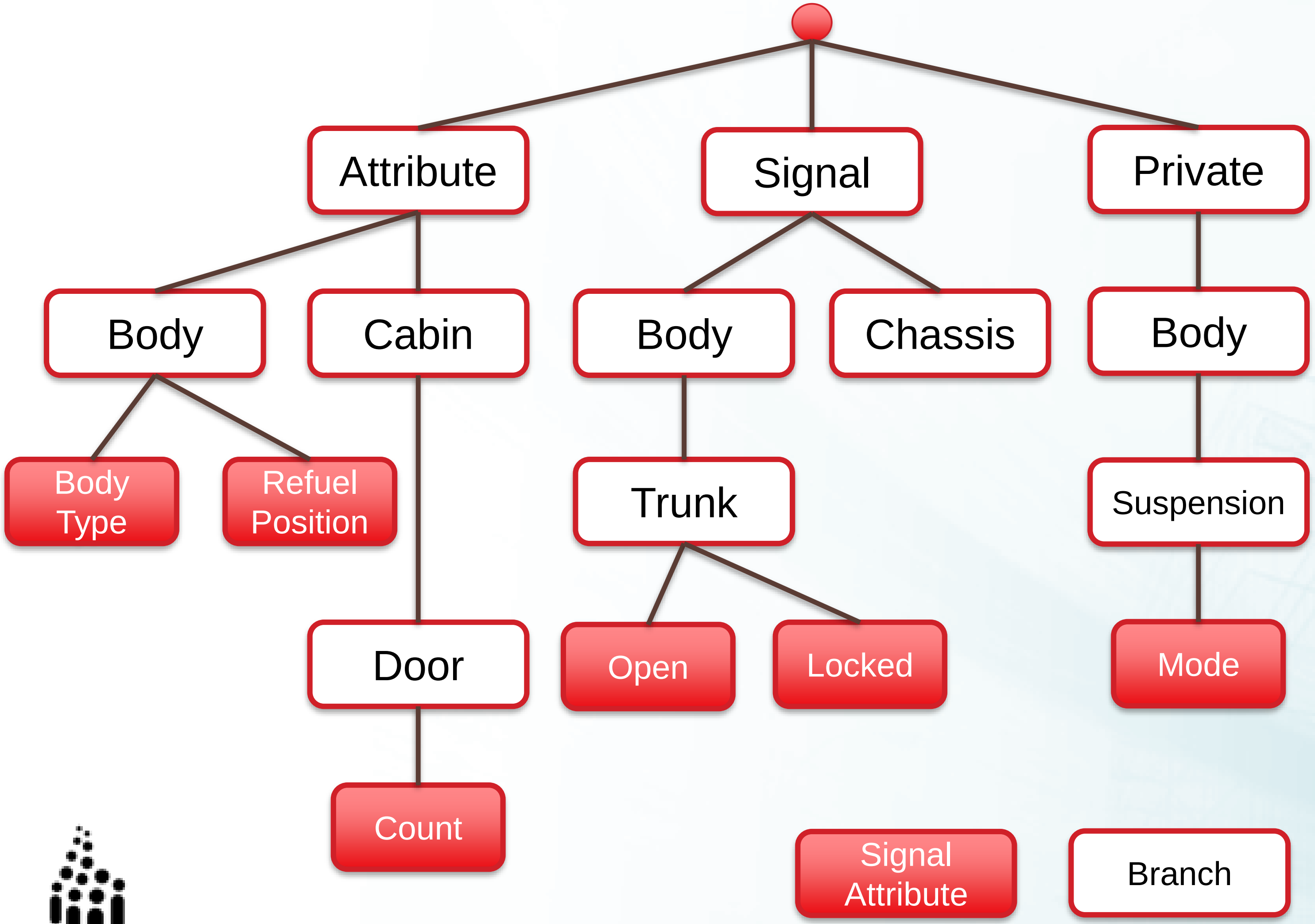
New Approach – Services with Signal Tree

- The core services *get*, *set*, *subscribe*, *unsubscribe*, *getVSS* and *authorize* are provided by a network server.
  - The services *get*, *set*, *subscribe* and *unsubscribe* provide access to vehicle signals and controls.
  - The service *getVSS* allows clients to query the server for available signals.
  - Using the *authorize* service, the client presents a security token to the server for authentication and authorization.
- Vehicle Signals and Controls are identified as nodes of a vehicle signal tree.
  - A fully qualified signal name addresses a single signal node.
  - Wildcards for branches and node names provide for addressing of signal groups.

Vehicle Signal Tree

Vehicle Signal Specification

Vehicle Signal Tree



- Tree structure provides for hierarchical access to signals and attributes.
- Branches group signals and attributes into entities that logically belong together.
- Wildcards allow access to entire sets of signals.

Addressing

Signal.Chassis.Brake.FluidLevel
Signal.Drivetrain.FuelSystem.Level
Attribute.Cabin.Door.Count
Attribute.Engine.Displacement

- Dot-notation for name path.
- Last path component, called node, represents the signal or attribute.
- Leading path components represent the branches.
- Wildcards can be used to address multiple signals and/or branches.

Specification Format

```
- Signal.Drivetrain.Transmission:
  type: branch
  description: Transmission-specific data
- Signal.Drivetrain.Transmission.Speed:
  type: Int32
  min: -250
  max: 250
  unit: m/s
  description: Current vehicle speed, sensed by gearbox
```

- Formatted as YAML lists
- Simple conversion into other formats such as JSON, France IDL, CSV, and more
- # denotes a comment or a directive
- Extensible – standard fields are defined, additional fields can be added as needed

Specification Format – Branch Description

```
- Signal.Drivetrain.Transmission:
  type: branch
  description: Transmission-specific data
```

- Fields
  - `type` – always set to branch for a branch
  - `description` – informative text describing the branch

Specification Format – Signal Description

```
- Signal.Drivetrain.Transmission.Speed:
  type: Int32
  min: -250
  max: 250
  unit: m/s
  description: Current vehicle speed, sensed by gearbox
```

- Fields
  - type – data type expressed as France IDL data type
  - unit – SI unit unless the type is Boolean
  - min, max – unless the type is Boolean or enumeration
  - enum – enumeration values for enumeration
  - description – informative text describing the signal

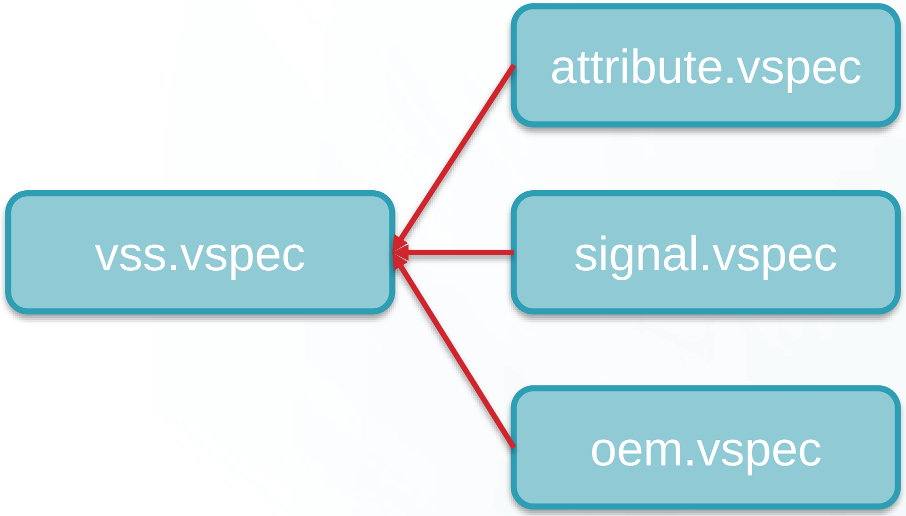
GENIVI®

Specification Format – Attribute Description

```
- Attribute.Cabin.Door.Count:
  type: Uint8
  value: 4
  description: Current vehicle speed, sensed by gearbox
```

- Fields
  - Same as signal
  - `value` – attribute setting
- Attributes are used to describe configuration data.

Aggregate File Inclusion



attribute.vspec

vss.vspec

signal.vspec

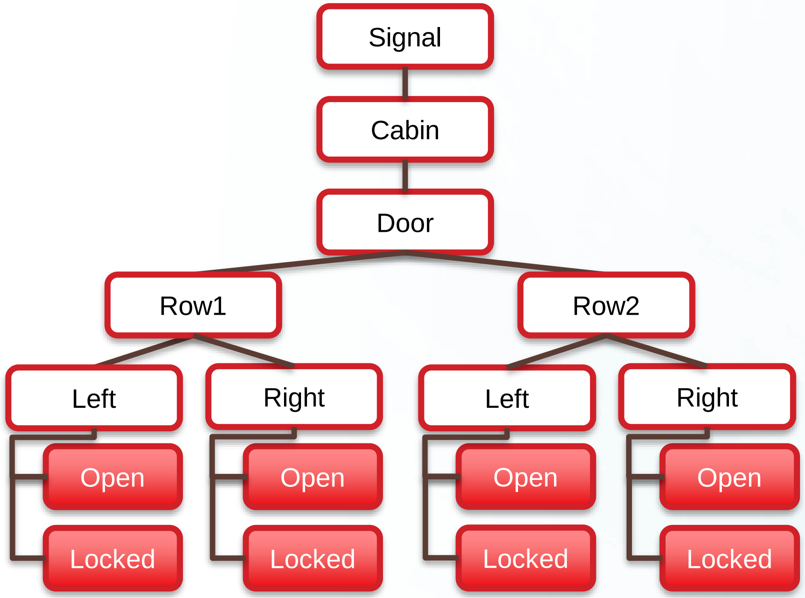
oem.vspec

vss.vspec

```
# top level vspec
#include attribute.vspec
#include signal.vspec
#include oem.vspec
```

- Vehicle signal specification files (vspec) can include other vspec file using the `#include` directive.
- Content of the included file is inserted into the including file at the position of the `#include` directive.
- Facilitates collaboration and minimizes editorial conflicts.

Reuse File Inclusion



door.vspec

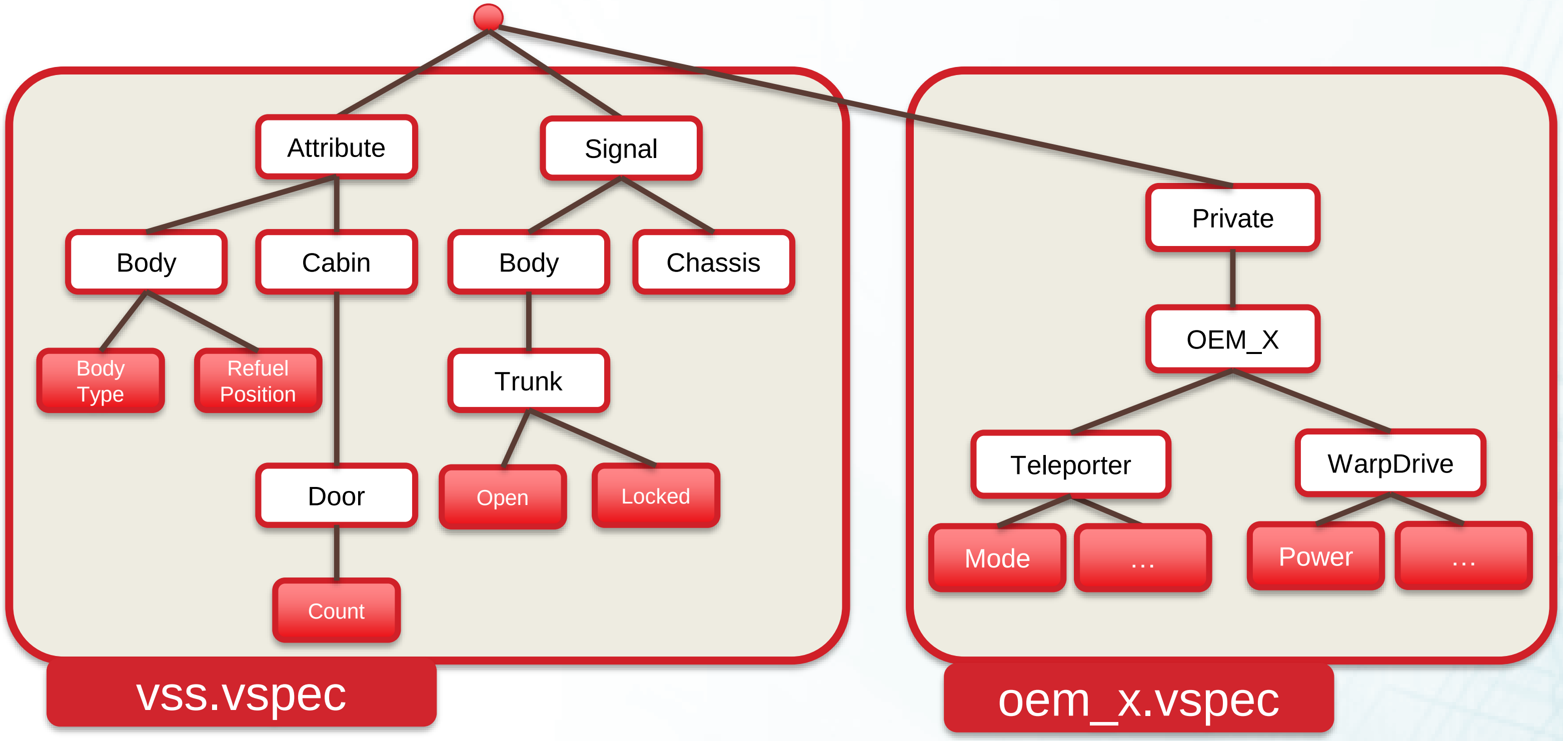
```
# door signals
-  Open:
   type: Boolean
   description: Door is open
-  Locked:
   type: Boolean
   description: Door is locked
```

cabin.vspec

```
# doors
#include door.vspec Signal.Cabin.Door.Row1.Left
#include door.vspec Signal.Cabin.Door.Row1.Right
#include door.vspec Signal.Cabin.Door.Row2.Left
#include door.vspec Signal.Cabin.Door.Row2.Right
```

- Specification fragments are included at a specific position of the signal tree.
- Specification fragments can be reused and an update is automatically reflected everywhere where the fragment is used.

Private OEM Extensions



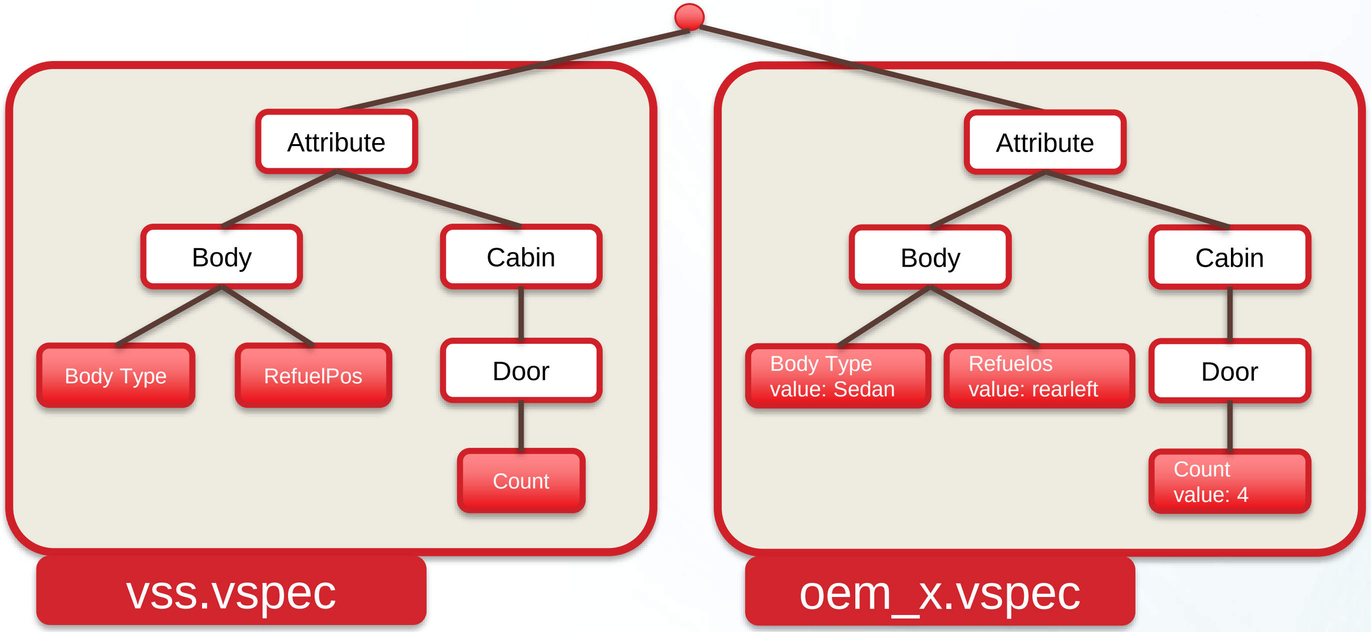
oem_x.vspec

```
# Include standard vspec
#include vss.spec

# Add proprietary signals
- Private.OEM_X.Teleporter.Mode:
  …
- Private.OEM_X.WarpDrive:
  …
```

- OEMs can use GENIVI vspec as a starting point and add proprietary signals.
- Use cases for
  - Reserved use by OEM and chosen vendors;
  - Public use by 3rd party application developers.
- Mature private extensions intended for public use can be submitted for VSS inclusion.

Attribute Declaration and Definition



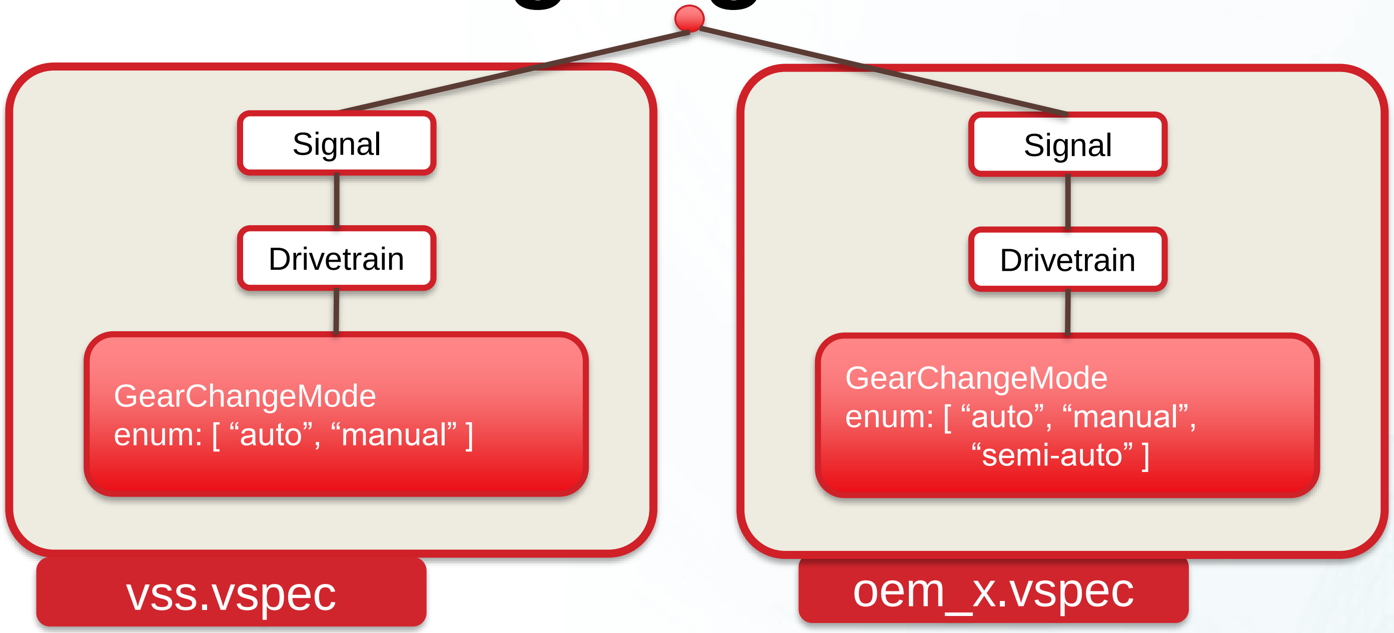
oem_x.vspec

```
# Include standard vspec
#include vss.spec

# Override/define attributes
-  Attribute.Body.BodyType:
   value: Sedan
-  Attribute.Cabin.Door.Count:
   value: 4
```

- Standard VSS either
  - Only declares an attribute or
  - Declares and attribute and assigns a default value.
- Declaration is overridden by definition in an OEM- or model-specific VSS file with the correct value.

Overriding Signal Definitions



vss.vspec

```
# Default signal definitions
-  Signal.Drivetrain.GearChangeMode:
   enum: [“auto”, “manual” ]
```

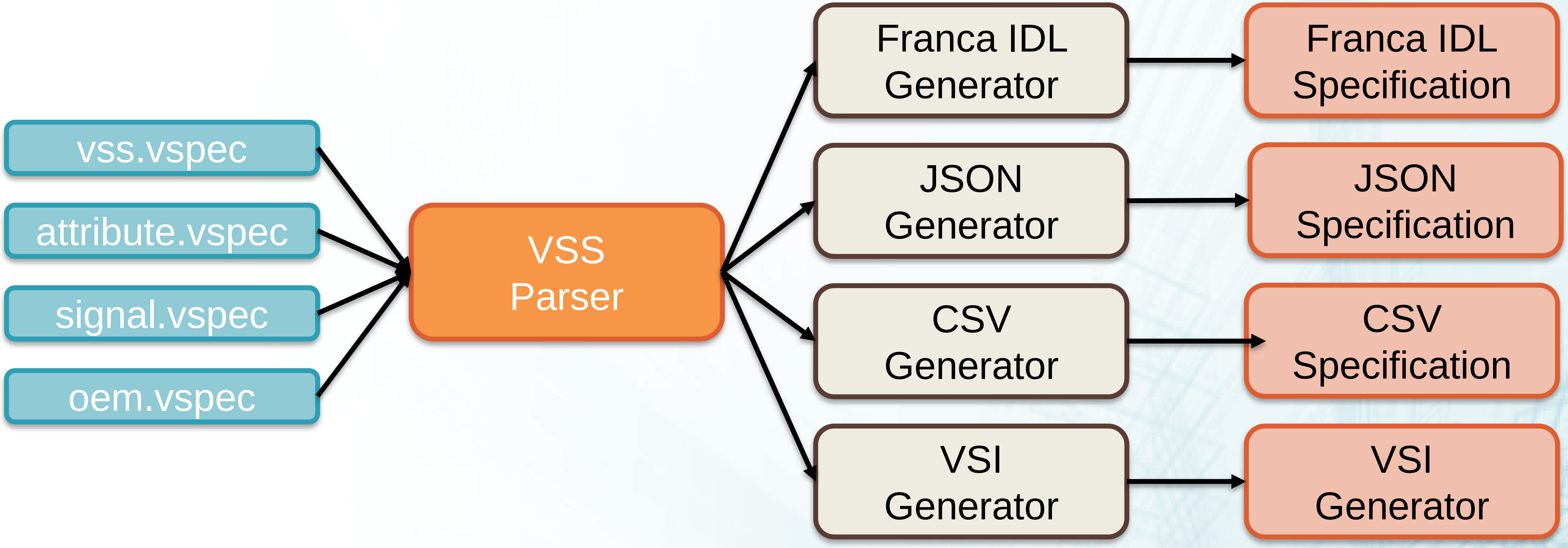
oem_x.vspec

```
# Include standard vspec
#include vss.spec

# Override/define signal definitions
-  Signal.Drivetrain.GearChangeMode:
   enum: [“auto”, “manual”,
            “semi-auto” ]
```

- Standard vspec lacks setting or has incorrect setting for a OEM/model etc.
- OEM/model-specific vspec can override the setting.

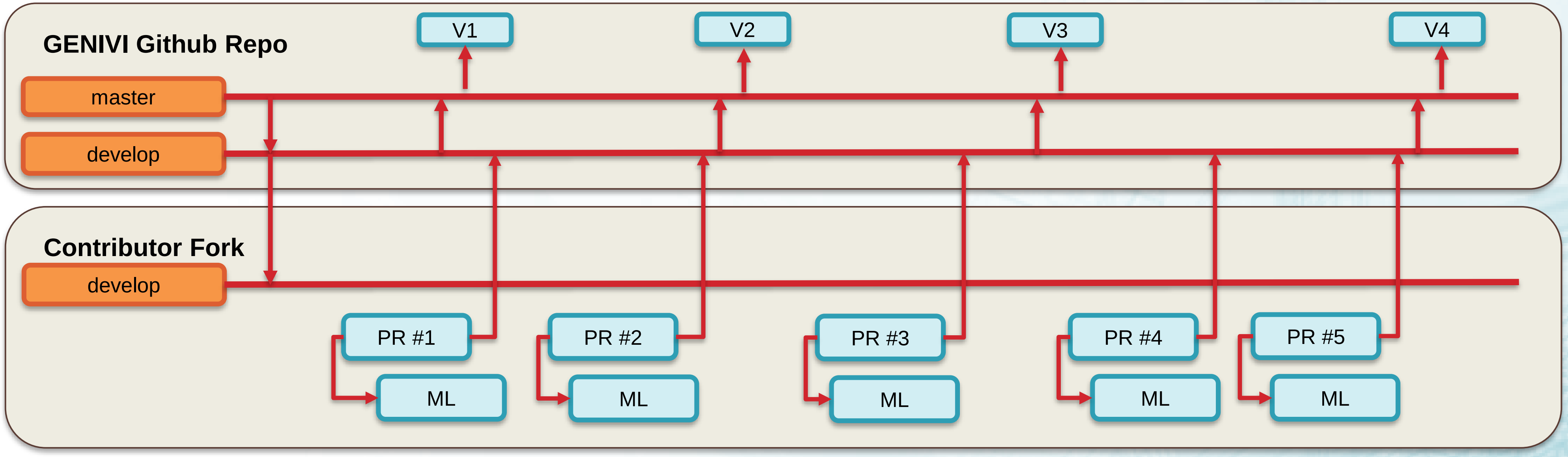
Format Transformation



- Tools written in Python transform VSS YAML (vspec) format into other formats.
- Standard Python library parses VSS YAML into a data structure.
- Output generators use the data structure to write their specific format.
- Output generators for Franca IDL, JSON, CSV and VSI are currently available. Other generators can easily be added.
- The VSI generator creates an alphabetically sorted list of the fully qualified signal and attribute names and assigns an index value to them.

Contribution and Releases

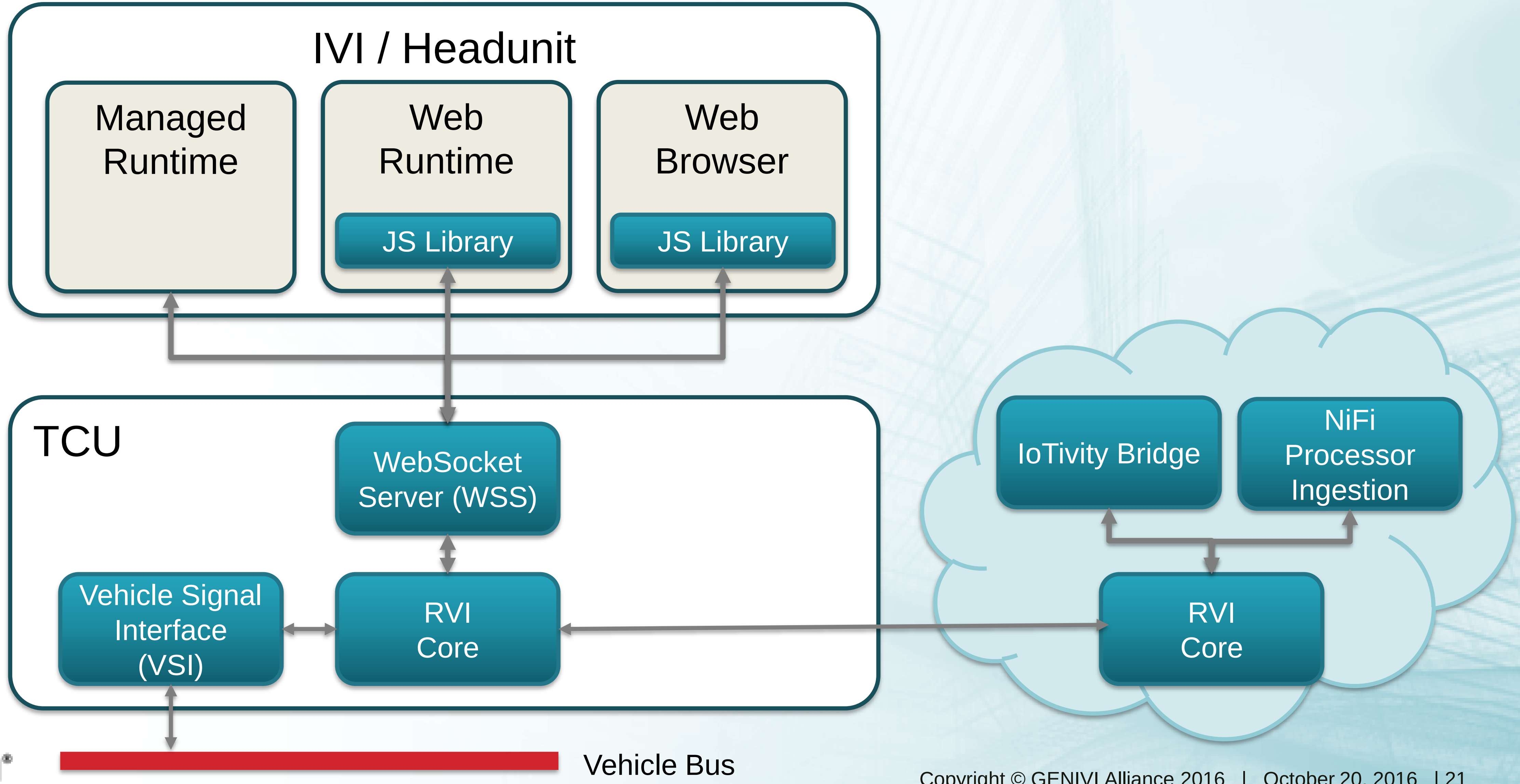
- Repository on Github under the GENIVI organization: https://github.com/GENIVI/vehicle_signal_specification



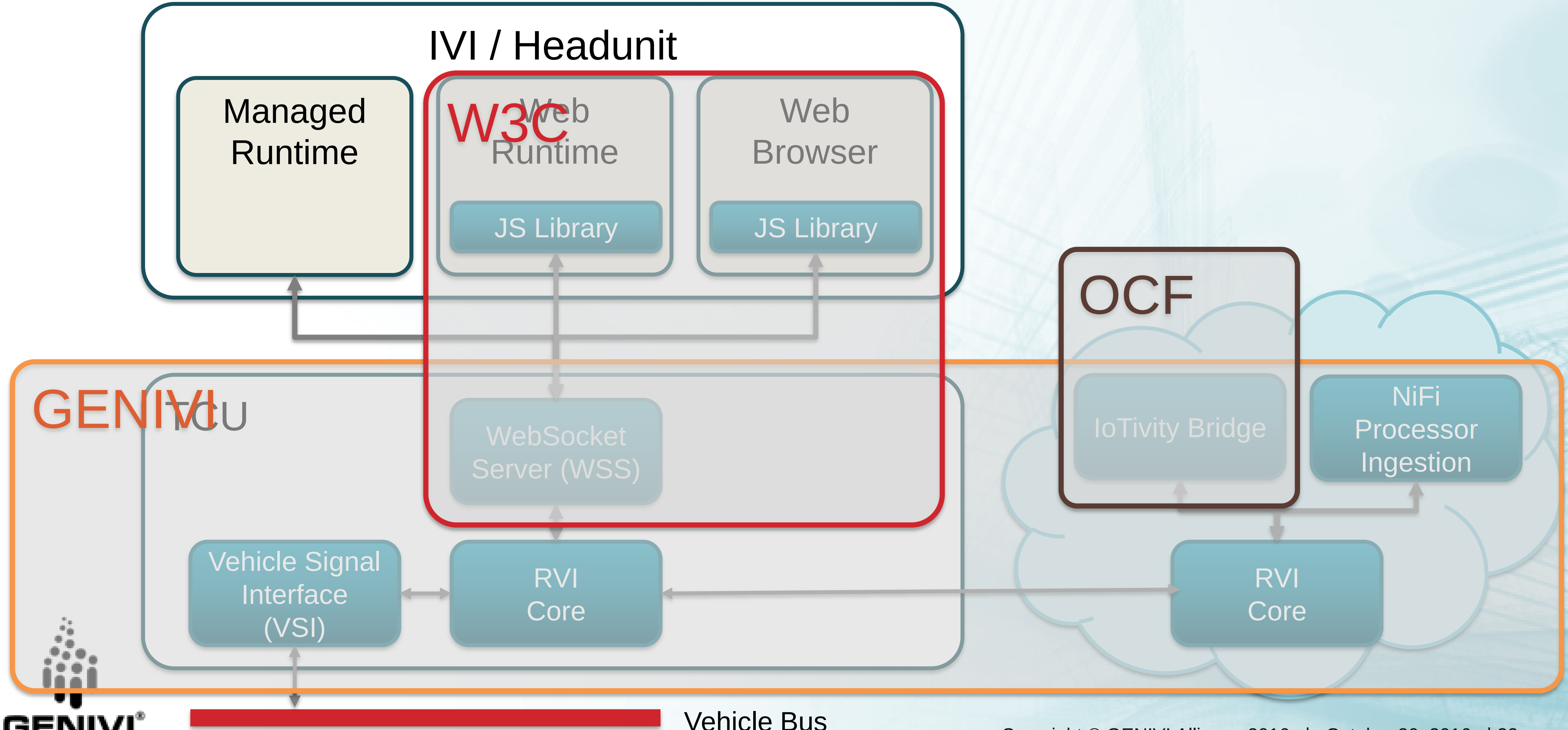
- Contributor forks GENIVI VSS repo.
- Contributor makes changes and submits pull-request against develop branch.
- Contributor e-mail genivi-projects mailing list pull-request info (hypertext link).
- Maintainer and contributors discuss and approve. Maintainer merges pull request.
- Releases are created by merging the develop branch into the master branch and tagging the master branch.

Architecture

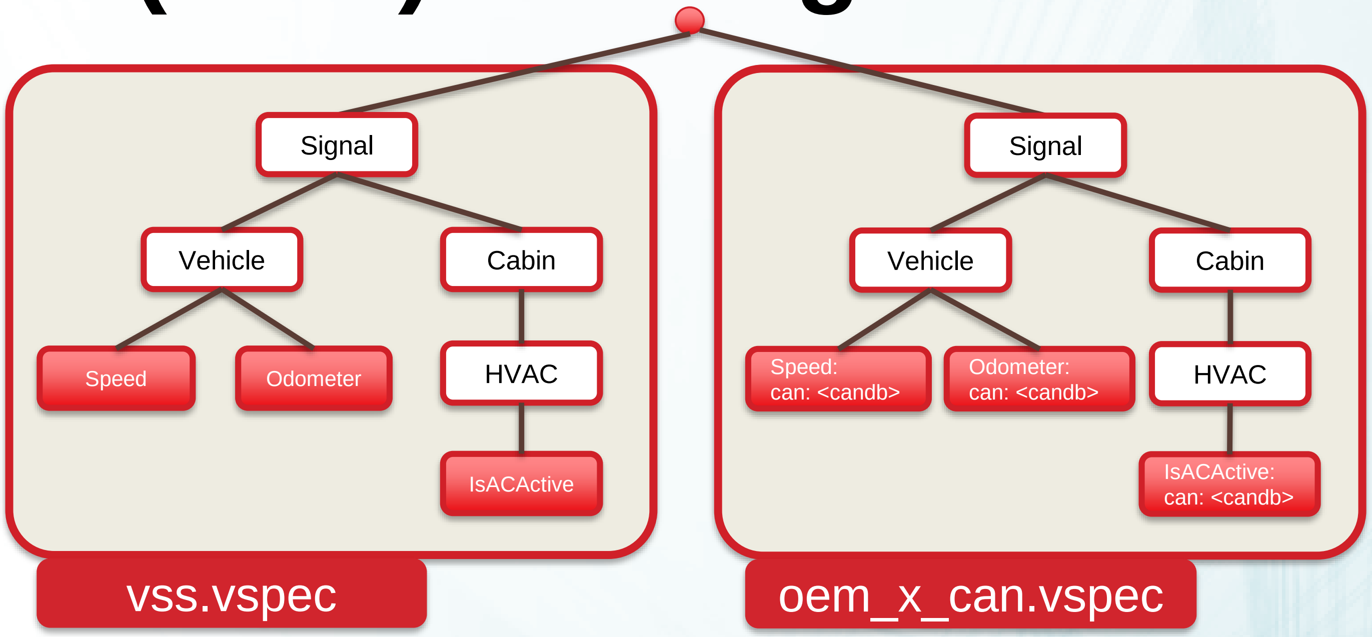
Vehicle Data Interfaces Architecture



Vehicle Data Interfaces Architecture



Vehicle Bus (CAN) Binding



oem_x_can.vspec

```
# Include standard vspec
#include vss.spec

# Add CAN DB field
- Vehicle.Speed:
  can: 47|16@0+ (0.01,0) [0|300] "0..300 kph, E = N * 0.01 + 0"
```

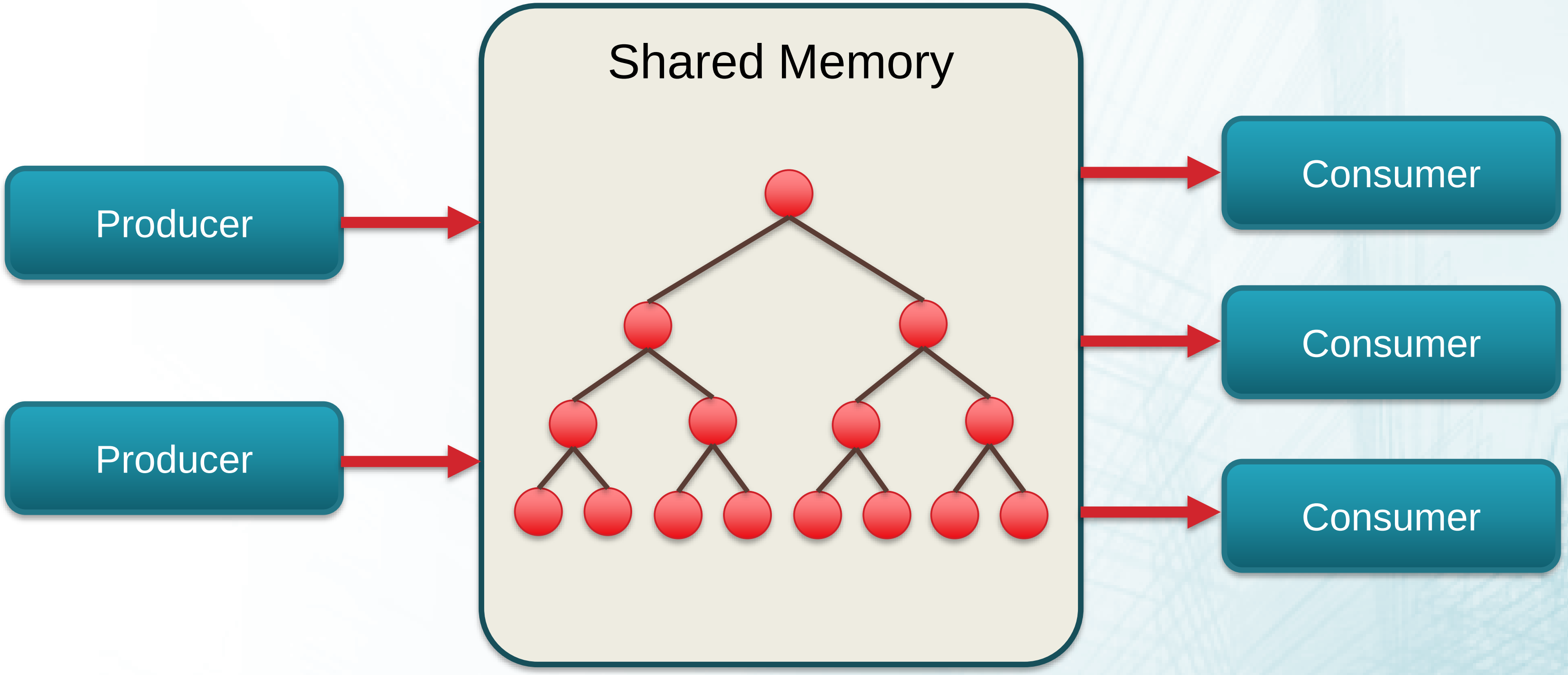
GDP VSS CAN Demo

Vehicle Signal Interface (VSI)

Vehicle Signal Interface (VSI) - Overview

- High-speed switchboard:
  - Up to 10 million transactions per second
  - Implemented in C
- Core library with API to implement VSI sources and sinks:
  - Interfaces to vehicle buses such as CAN.
  - Interfaces to RVI and/or other applications.
- Signals are identified by either name or ID. Two sets of APIs e.g.:
  - `int vsi_set_signal ( vsi_result* result );`
  - `int vsi_set_signal_by_name ( vsi_result* result );`
- Lookup functions to convert signal names to ID and vice versa:
  - Signal map can be imported from VSI file created by the `vss2vsi` transformation tool.
- Signals can be grouped and an application can listen to individual signals in the group or all signals.
- Signal switchboard is implemented as B-tree database in shared memory.

Vehicle Signal Interface (VSI) - Design



- Producers post signals into VSI shared memory where they are stored in a b-tree ordered by signal ID.
- Consumers read individual signals or signal groups from shared memory. Read functions return immediately if a signal has been posted or block until a signal arrives.
- Callback functions are not supported.

Questions?

Thank you!

Weekly Networking Expert Group Call

Mondays 0815 PT / 1715 CET / 1615 UTC

https://genivi.webex.com/genivi/j.php?MTID=mdb9482b92015e5cb7386c1a65e32a887

Meeting number: 579 975 193

Mailing List

https://mail.genivi.org/sympa/info/eg-nw

GENIVI®