Truly portable Vehicle Applications using Webassembly & WASI

Mathias Danzeisen, Research Engineer, ETAS GmbH

27 April 2023

COVESA
Accelerating the future of connected vehicles

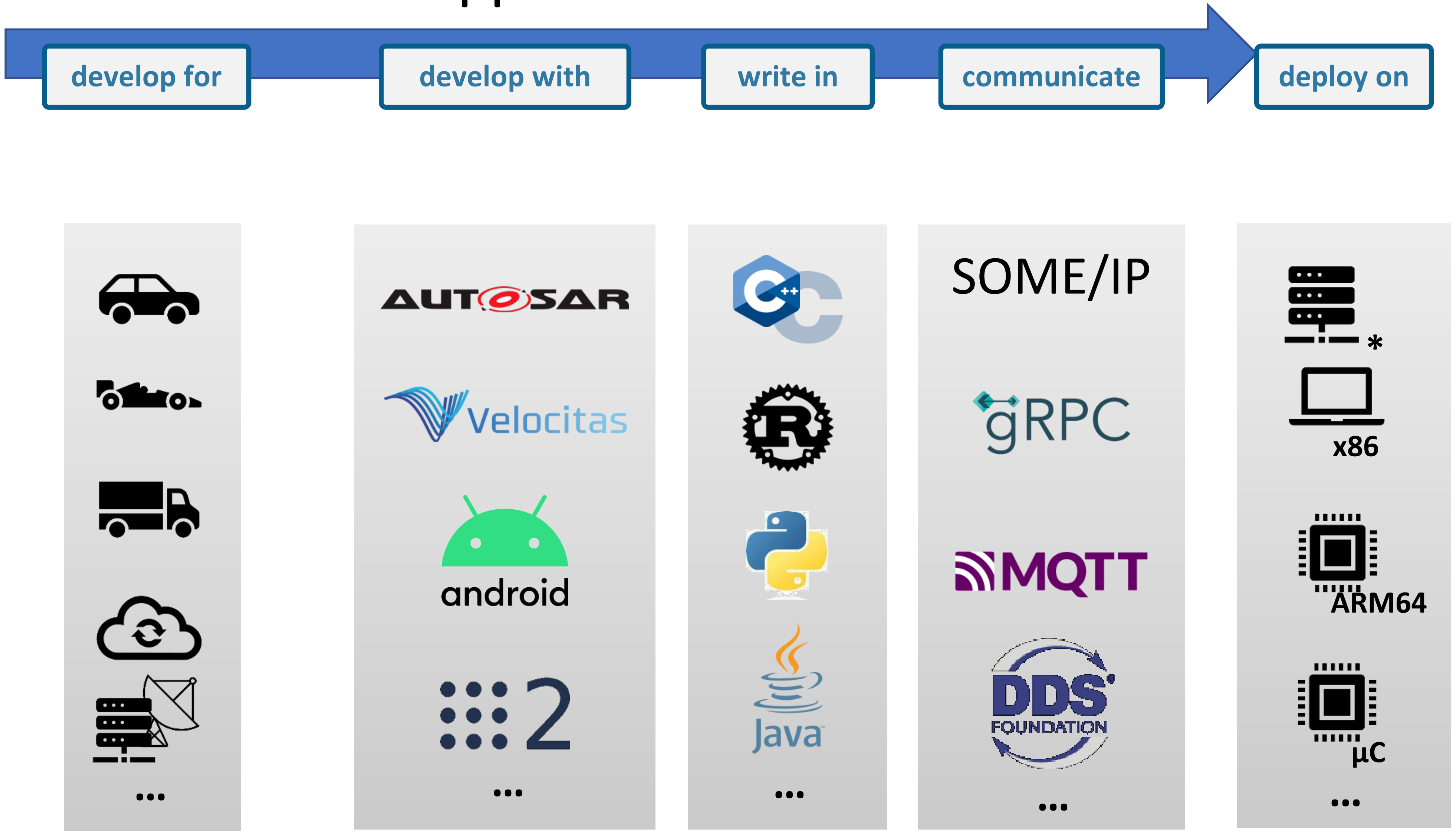
ETAS

Do we have Portable Vehicle Applications?

We want

Application Software

…



Do we have Portable Vehicle Applications?

We want

Application Software

…

develop for	develop with	write in	communicate	deploy on
Vehicle Signal Specification ✓	**?**	**?**	**?**	**?**
	AUTOSAR	C++	SOME/IP	*
	Velocitas		gRPC	x86
	android		MQTT	ARM64
	2	Java	DDS FOUNDATION	µC
…	…	…	…	…

COVESA
Accelerating the future of connected vehicles

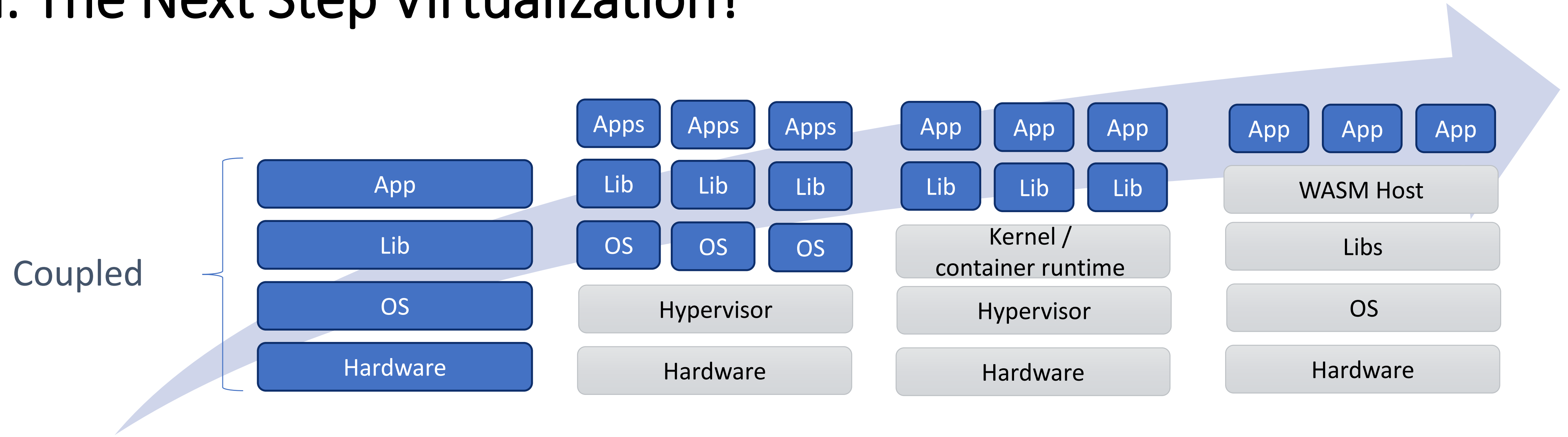
What is WebAssembly(Wasm)?

- Wasm defines an Instructions Set and an Execution Model
- Design Goals:
  - Portable code (Interpreted, Ahead-of-time compiled, JIT)
  - Performance near native-code performance
  - Safe and secure sandboxed
  - Streamable
  - Language-independent (C/C++, Rust, Java,…)
- Developed by W3C with support from major browsers, but with increasing support to run outside the browser
- Use cases span Cloud, Edge and embedded Devices (e.g. ARM M3/4)

Wasm designed to run code "fast, safe and efficient" written in any language on any platform*

*) https://webassembly.github.io/spec/core/intro/introduction.html#design-goals

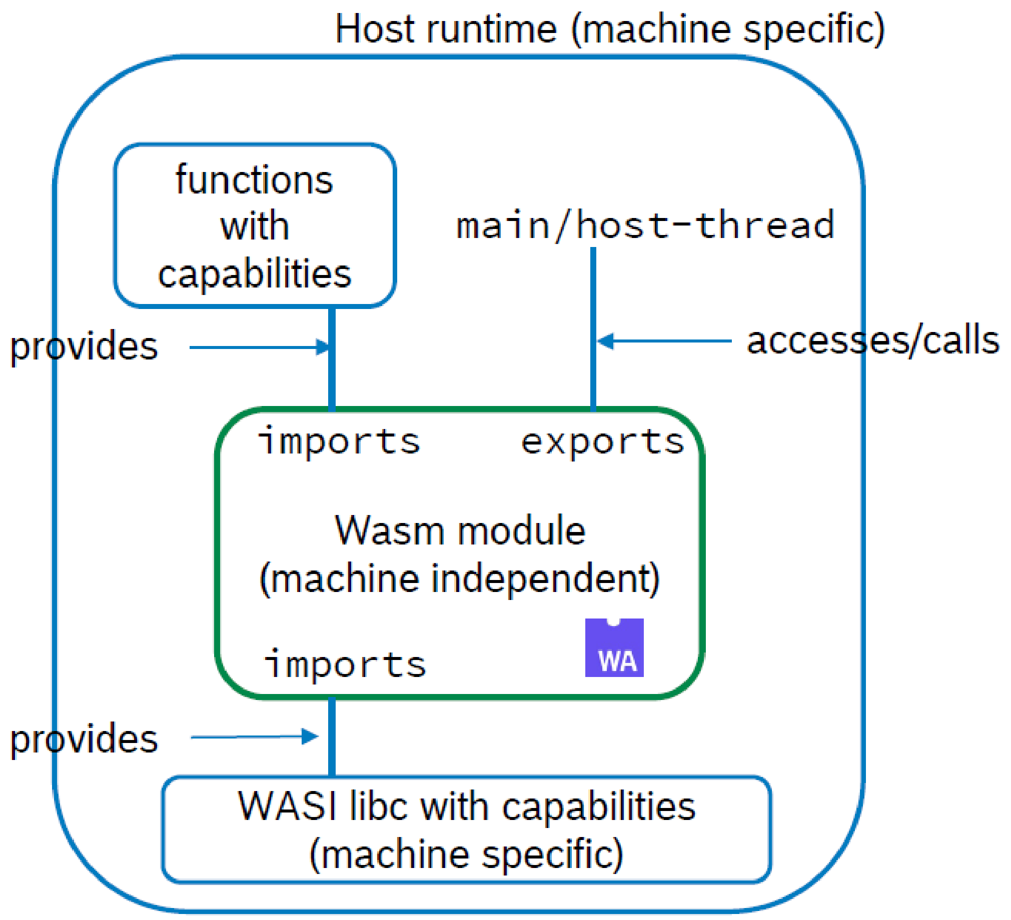
Wasm: The Next Step Virtualization?



Environment	**PC (Datacenter)**	**Cloud (public)**	**Cluster (e.g.K8S)**	**Wasm**
Format	Image	VM	Container	WASM Module
Developer Responsibility	App & Libs & OS & Hardware	App & Libs & OS	App & Libs	Business Logic
Abstraction	Hardware	CPU	OCI/Linux kernel	Sandbox, WASI
Security	System	OS	Process	Capability-based

Based on: https://cosmonic.com/blog/

WASI – The Standardized WebAssembly System Interface



Host runtime (machine specific)
functions with capabilities
main/host-thread
provides
accesses/calls
imports
exports
Wasm module (machine independent)
imports
WA
provides
WASI libc with capabilities (machine specific)

- a specification to run WebAssembly outside the web
- family of APIs for WebAssembly
- currently is a subset of POSIX APIs (insecure and thread-unsafe APIs are dropped)
- focus on system-oriented APIs (files, networking, messaging, machine-learning,…)
- WASI vision is “capability based” system interface
- Interface definition language (component proposal)

Wasm Interface Type (WIT) format

```
world my-world {
  import host: interface {
    use pkg.types.{errno}
    record ldata {
      ino: u64,
      size: u64,    // ...
    }
    log: func(param: ldata) -> result<errno>
  }
  export run: func()
}
```

world
- Top level defintion of wasm component

Import & export
- Gives interfaces direction

interface
- Collection of function and types

use
- references

record/data types
- Complex data types can defined e.g.: record, enum, flag, union, variant

func
- Named functions with parameters and return value

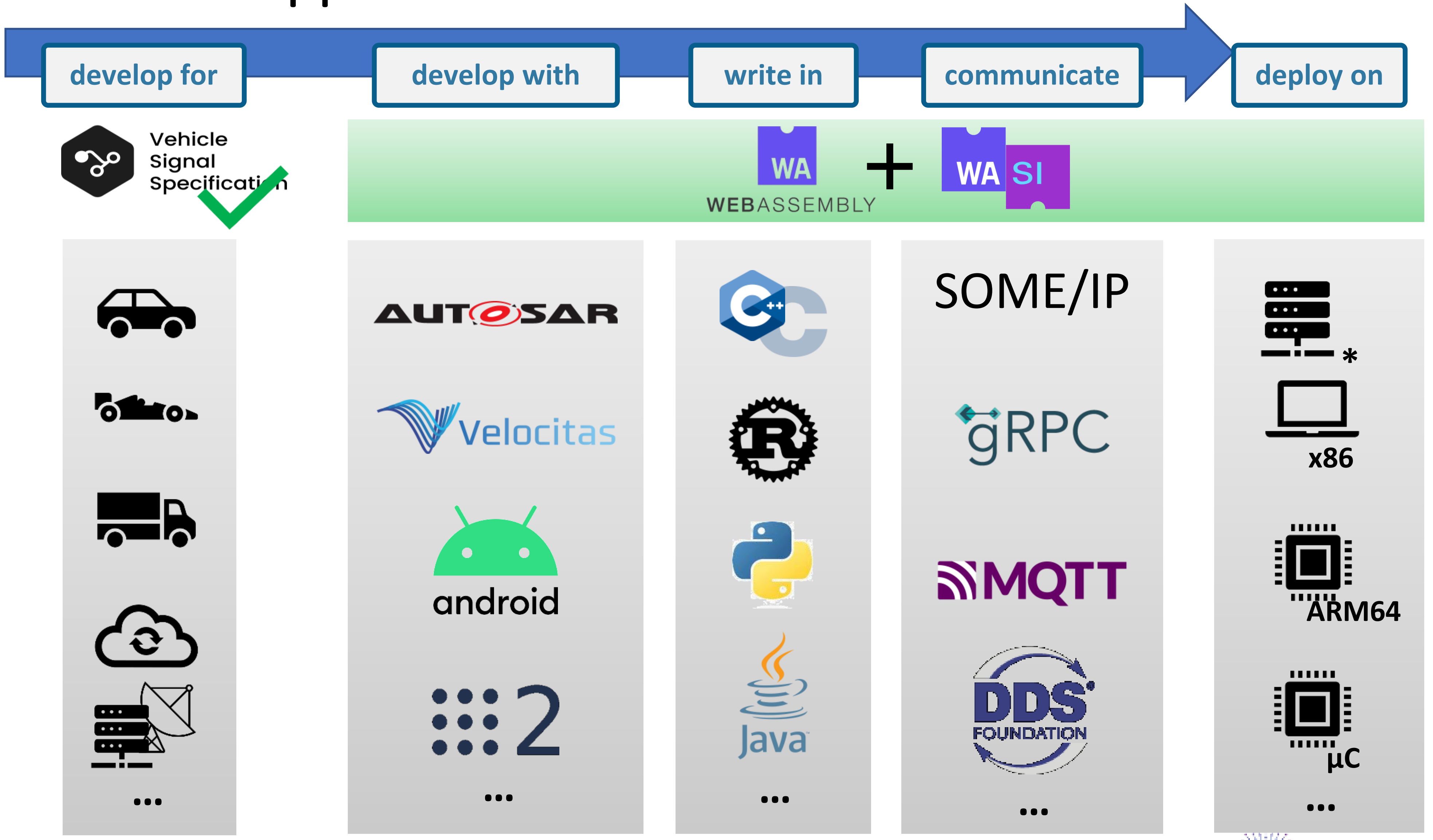
See full specification athttps://github.com/WebAssembly/component-model/blob/main/design/mvp/WIT.md

Idea: Portable Vehicle Applications with Wasm & WASI

We want

Application Software

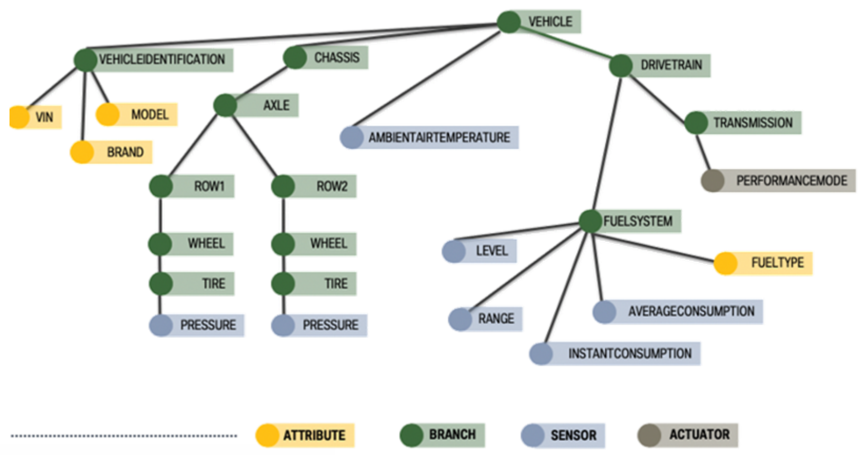
…



COVESA

Accelerating the future of connected vehicles

Mapping: vss -> wit



Constrains from wit format:

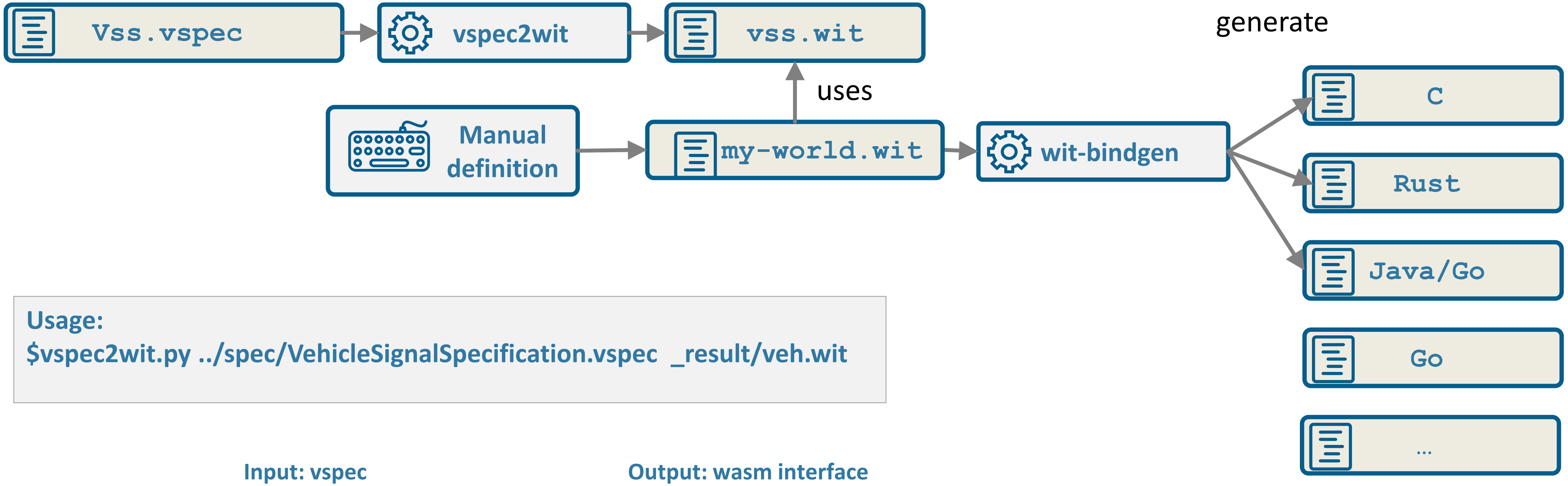
- nesting of interfaces not possible (As of now)
- Future & and stream types for interfaces not available (yet)
- Notation: Only small letters and no dots are allowed in the interface name

Convert hierarchical structure of VSS

- Map tree to concatenated string
  - e.g.: vehicle-body-windshield-front-wiping-system-ispositionreached
- Map datatypes between VSS and wit
  - e.g.: boolean (Vss) -> bool (wit);
  - float (vss) -> float32 (wit)
- Keep naming wit naming conventions: VSS snake case name

```
// State of the supply voltage of the ECU
interface vehicle-lowvoltagesystemstate {
     enum vehicle-lowvoltagesystemstate-values {
          UNDEFINED,
          LOCK,
          …
          START,
     }
     subscribe: func() -> bool
     unsubscribe: func() -> bool
     get: func() -> vehicle-lowvoltagesystemstate-values
}
```

vsstooling enhancement: vspec2wit (draft)



Usage:
```
$vspec2wit.py ../spec/VehicleSignalSpecification.vspec  _result/veh.wit
```

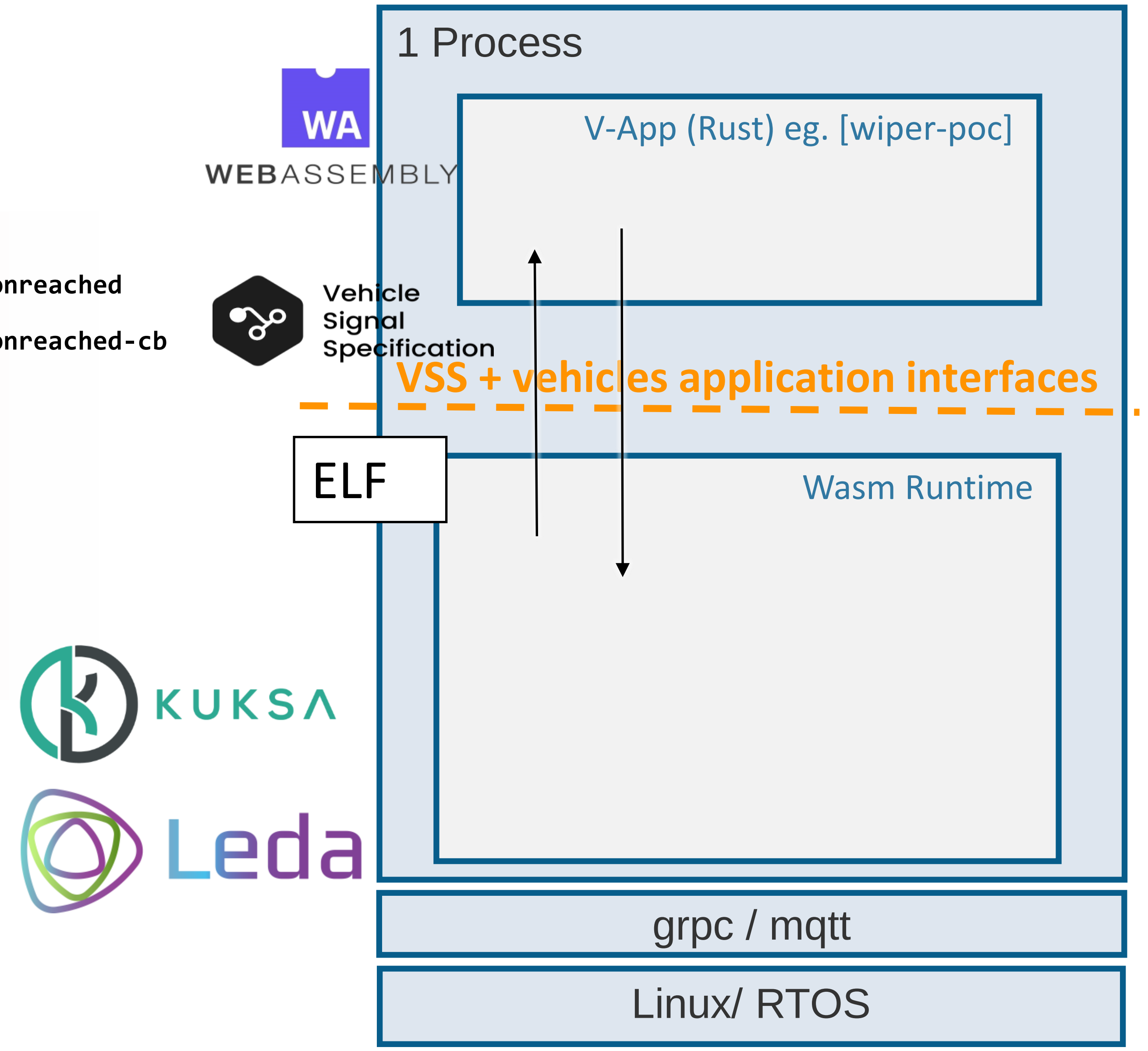
Input: vspec

Output: wasm interface

Proof of Concept (PoC)

Wasm component interfaces

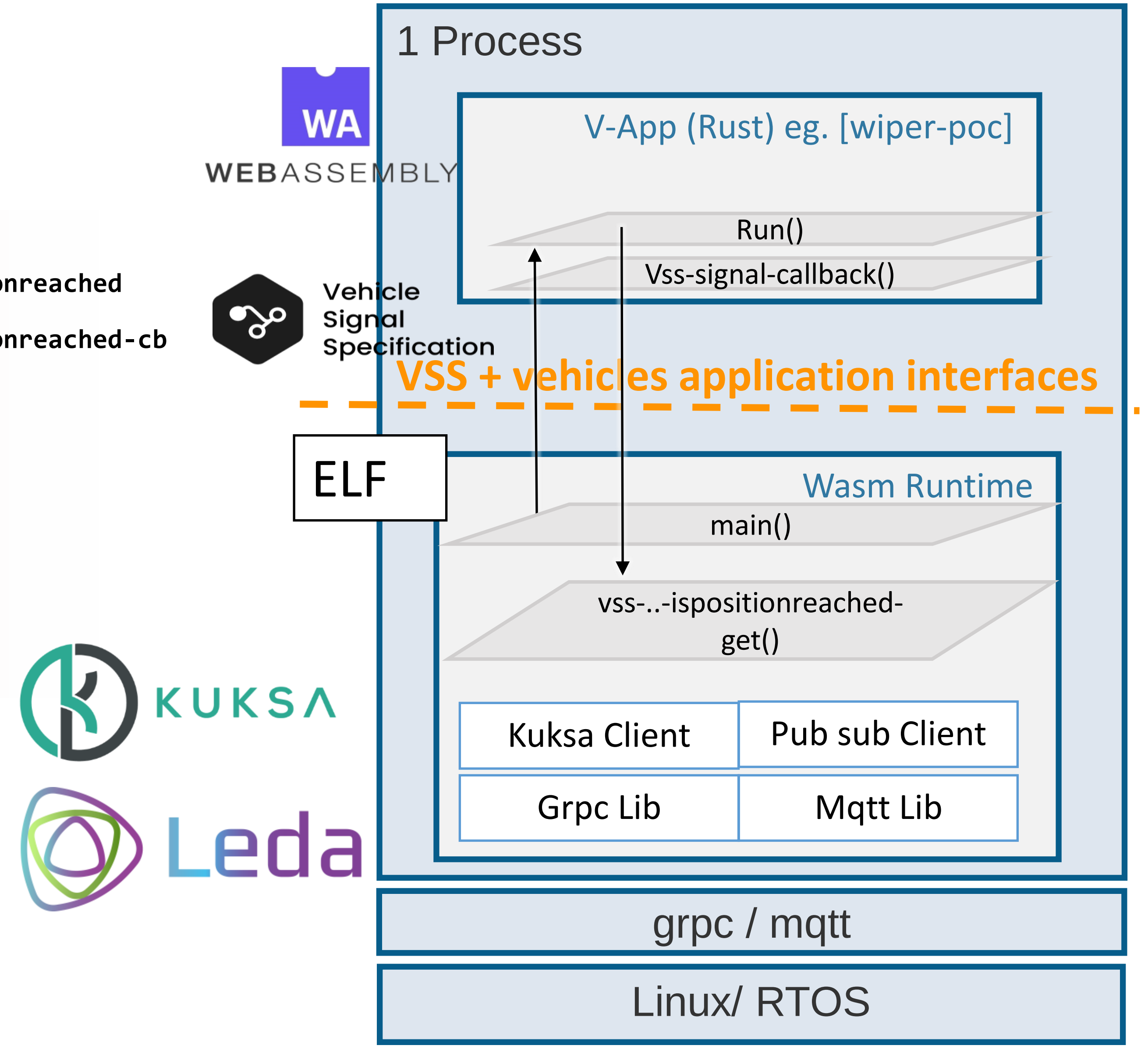
```
default world wiper-poc {
  import vehicle-body-<..>-ispositionreached: body.<..>.ispositionreached
  export vehicle-body-<..>-ispositionreached: body.<..>.ispositionreached-cb

export vehicle-app: interface{
     init: func()  -> bool
     run: func()  -> bool
  }
}
```


WEBASSEMBLY
Vehicle Signal Specification
KUKSA
Leda
1 Process
V-App (Rust) eg. [wiper-poc]
VSS + vehicles application interfaces
ELF
Wasm Runtime
grpc / mqtt
Linux/ RTOS

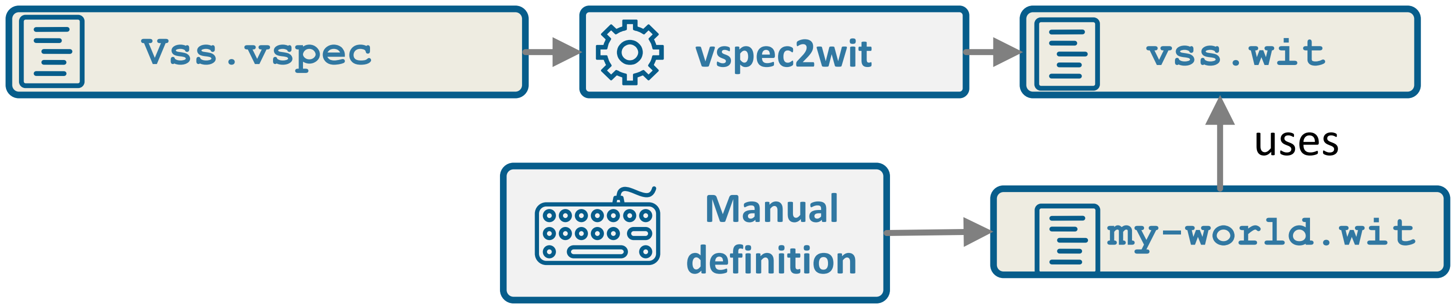
Proof of Concept (PoC)

Wasm component interfaces

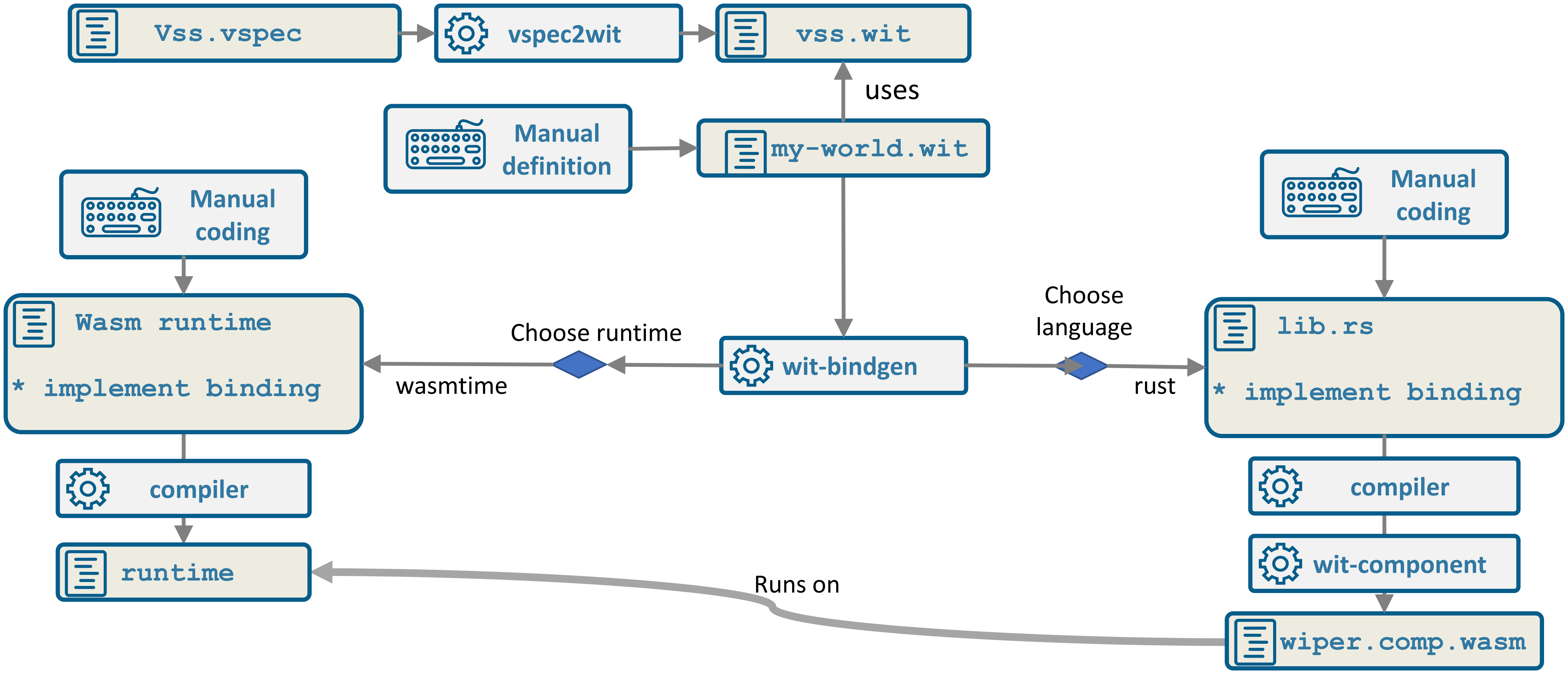
```
default world wiper-poc {
  import vehicle-body-<..>-ispositionreached: body.<..>.ispositionreached
  export vehicle-body-<..>-ispositionreached: body.<..>.ispositionreached-cb

export vehicle-app: interface{
     init: func()  -> bool
     run: func()  -> bool
  }
}
```



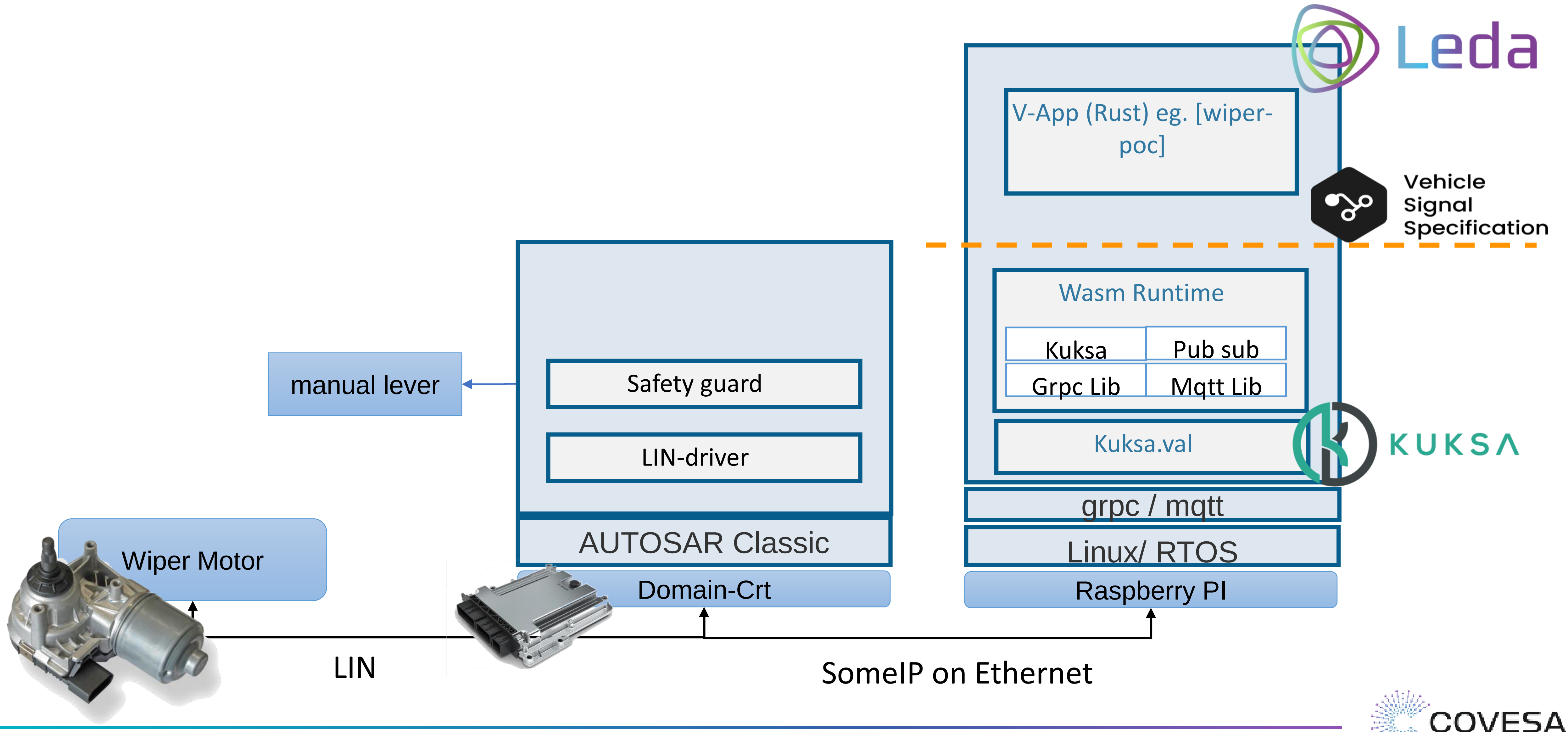
PoC Toolchain



PoC Toolchain



PoC Integration



COVESA

COVESA

Accelerating the future of connected vehicles

etas

Thank you!

Visit: etas.com

Mathias.Danzeisen@etas.com