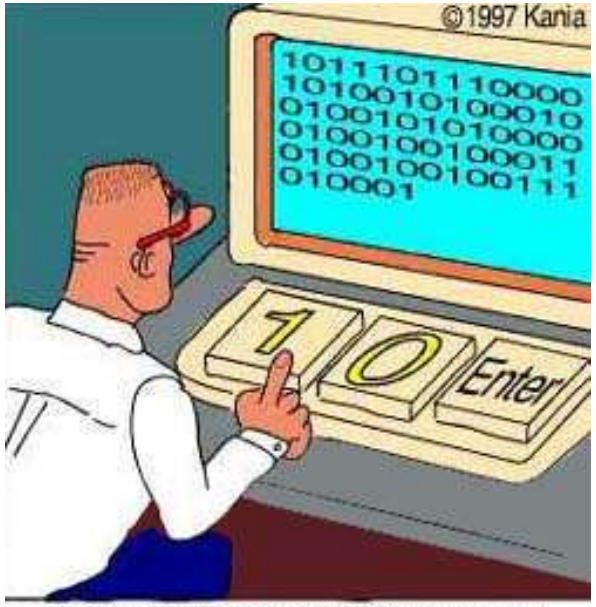
Reality bites

An odyssey into how things (may) work

Per Söderstam, Semcon

Who am I

- Per Söderstam
- MSc.ECE, Chalmers
- Ericsson, …, Halmstad University, …, Semcon
- Multi-processor high performance radar HW/SW to 8-bit uC remote control
- Embedded systems/software design



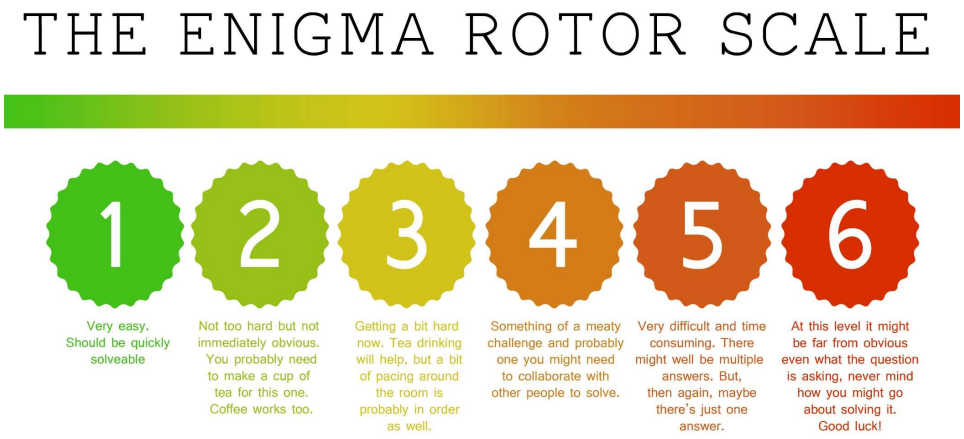
Real programmers code in binary.

Outline

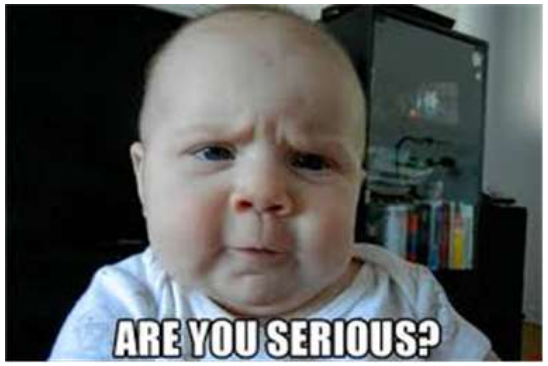
- Lecture
  - Working in/with complex systems
  - APIs and Porting
  - The Android CAR API
  - How does it work (Pay no attention to the code behind the curtain)
  - How to design APIs
- Why
  - In preparation for your task
  - Understanding of internal Android mechanisms
  - Understanding of API in general

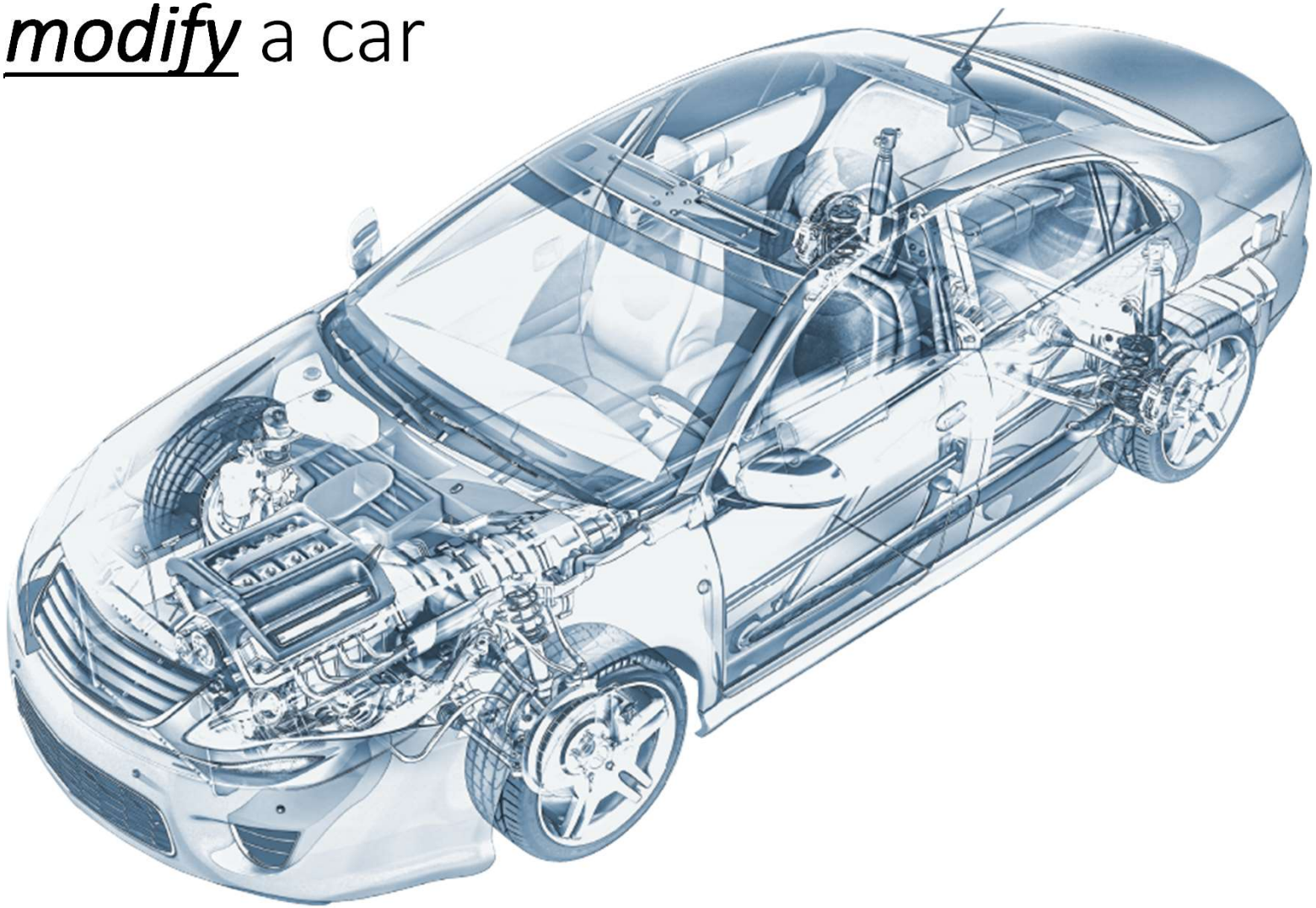
Development stages

- Engineering development is hard!
- Complexity does not scale linear
  - Think of a bridge over a stream, then think of a bridge over a river
  - One is a one man job without need for planning, the other is a huge project
- Techniques does not transfer
- The cost increase with complexity



Your task – ***modify*** a car





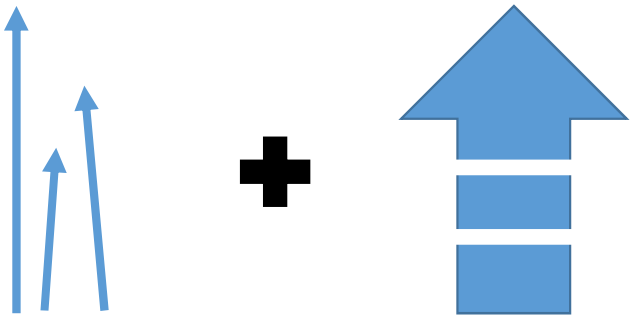
Handling development complexity

- Develop incrementally
  - What order? Have a plan…
    - We call those plans ***design***
  - Adding functionality peacemeal
  - Tracer bullet style
  - Layer style
  - Plan to throw one away
- Test at each stage

Techniques

Tracer round

- Develop one concept/function all the way
- The purpose is to try to hit as many system imposed snags as possible



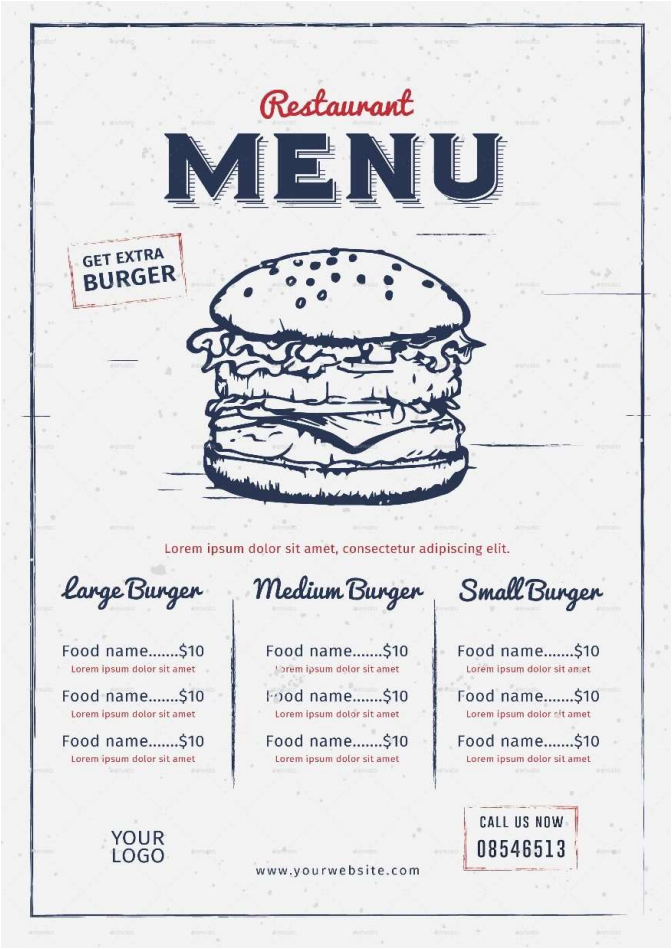
Layering

- Build bottom up
- Provide service interfaces for more (and more) complex functions
- The purpose is to hide lower level complexity through encapsulation
- Think DSL (Domain Specific Languages)
- **APIs**

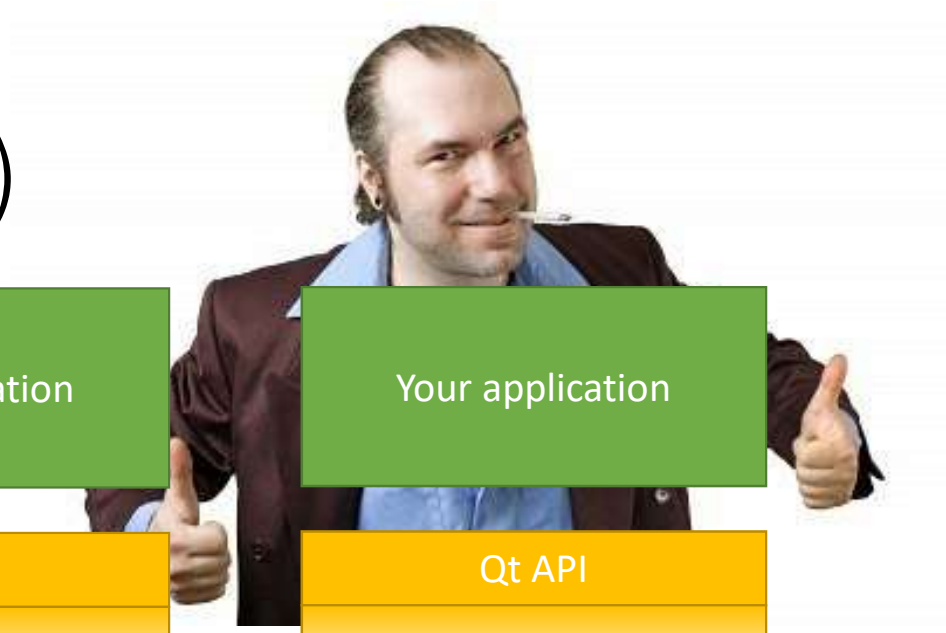
Application Programming Interface - API

- Interface to a module/component
- Hide internal workings
- Service menu/provider
- Rules definition, contract
- Methods, classes, protocols, conventions,…

- Facilitate **porting** software between platforms

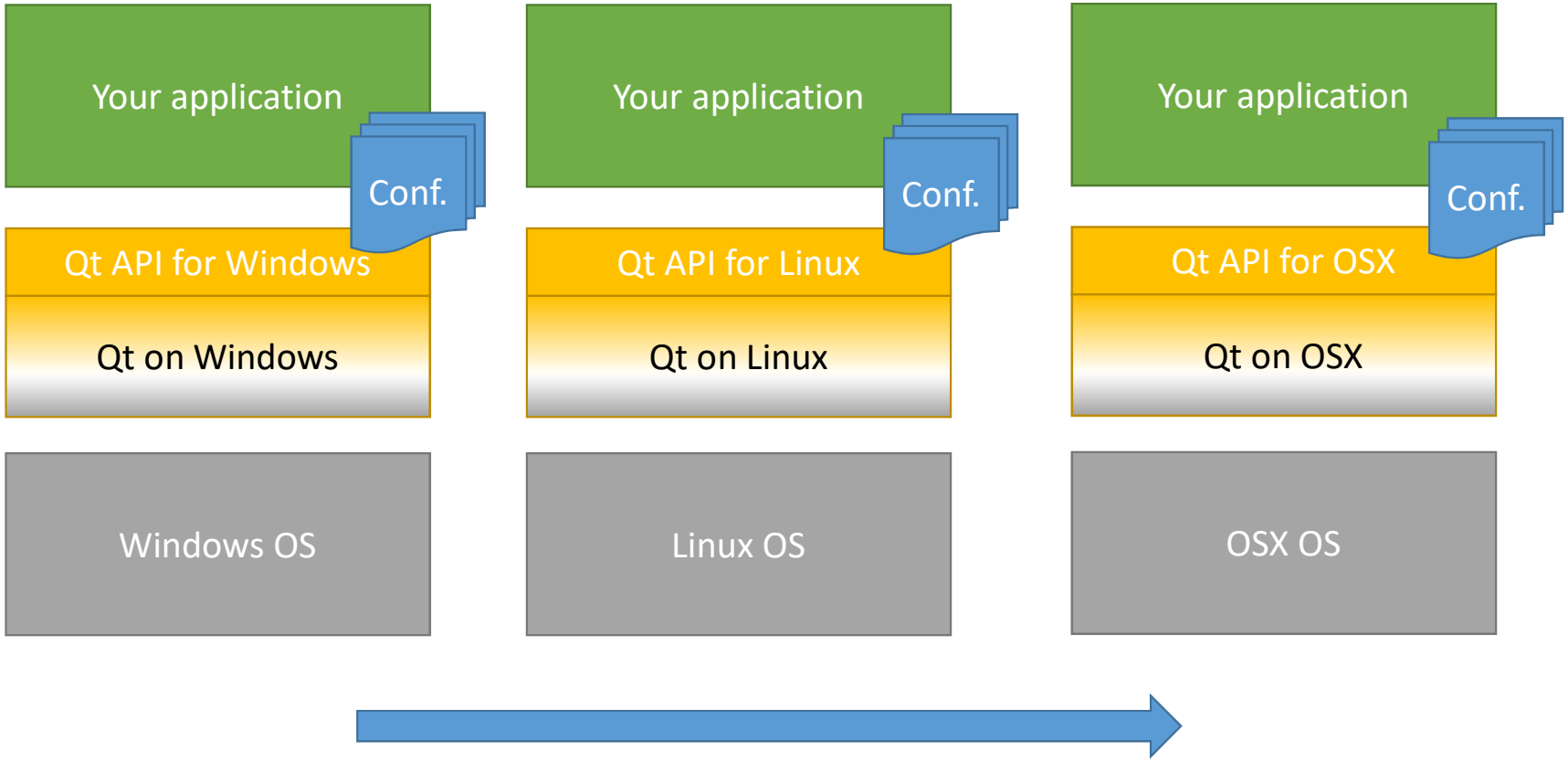


Porting (sales version)



Your application	Your application	Your application
Qt API	Qt API	Qt API
Qt on Windows	Qt on Linux	Qt on OSX
Windows OS	Linux OS	OSX OS

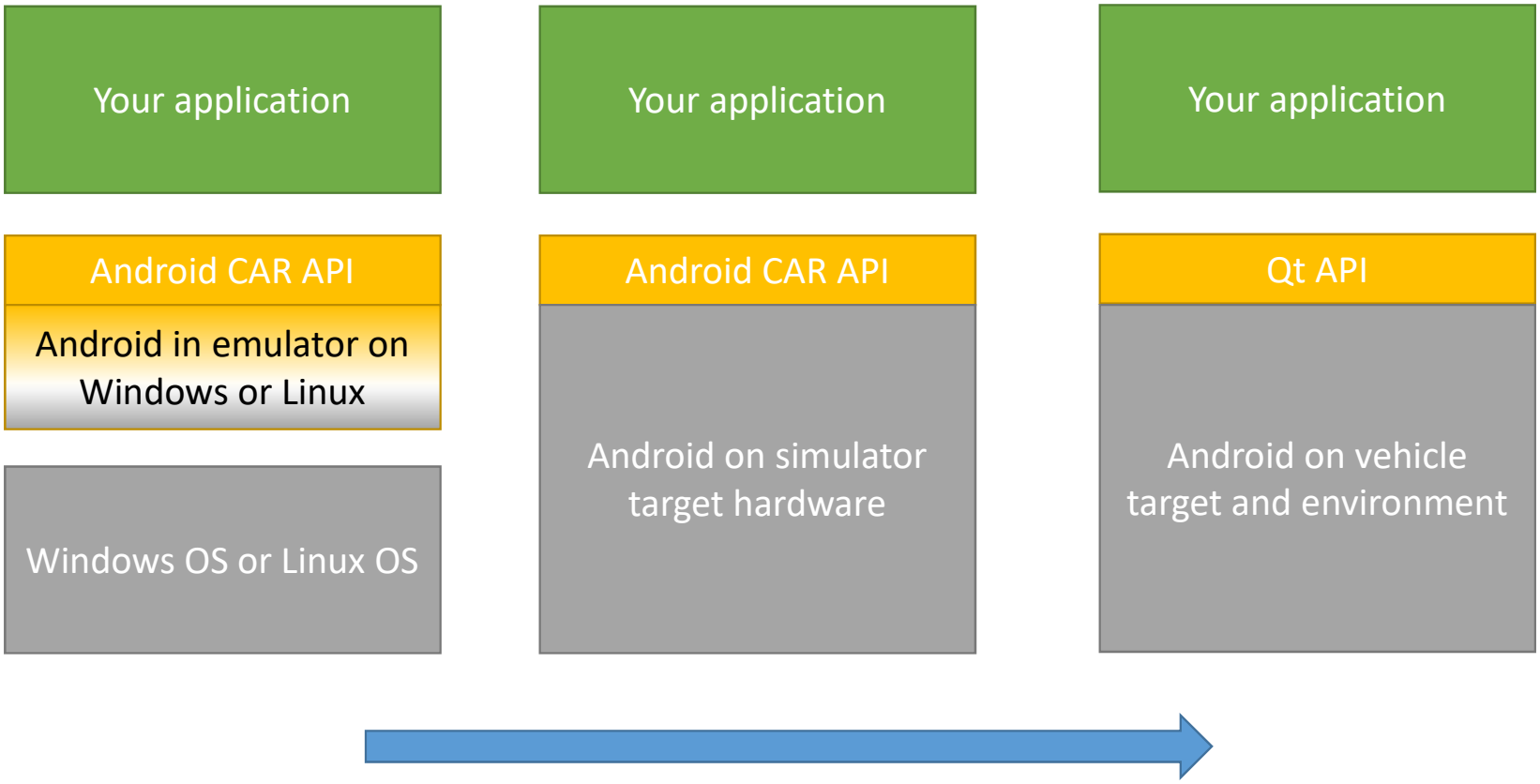
Porting (reality)



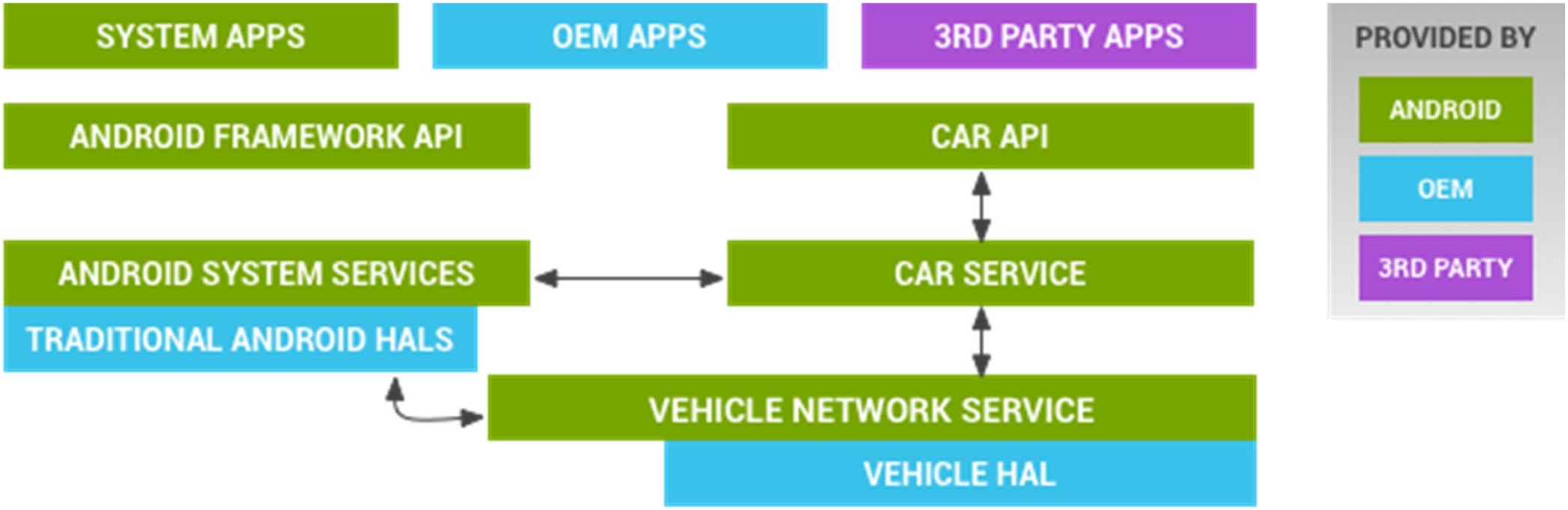
Your task – a plan to handle complexity

1. Commit to an existing interface/API
   - Android w/ CAR API
2. Build your application on a simple platform first
   - On host, fully controlled, high visibility, low cost (several iterations per hour)
3. Test your application in simulator
   - Controlled and bounded environment, closer to target, medium cost (one iteration per hour)
4. Integrate into vehicle
   - Actual target, low controllability, specialized tools for introspection, high cost (one iteration per day)

The plan from a porting view

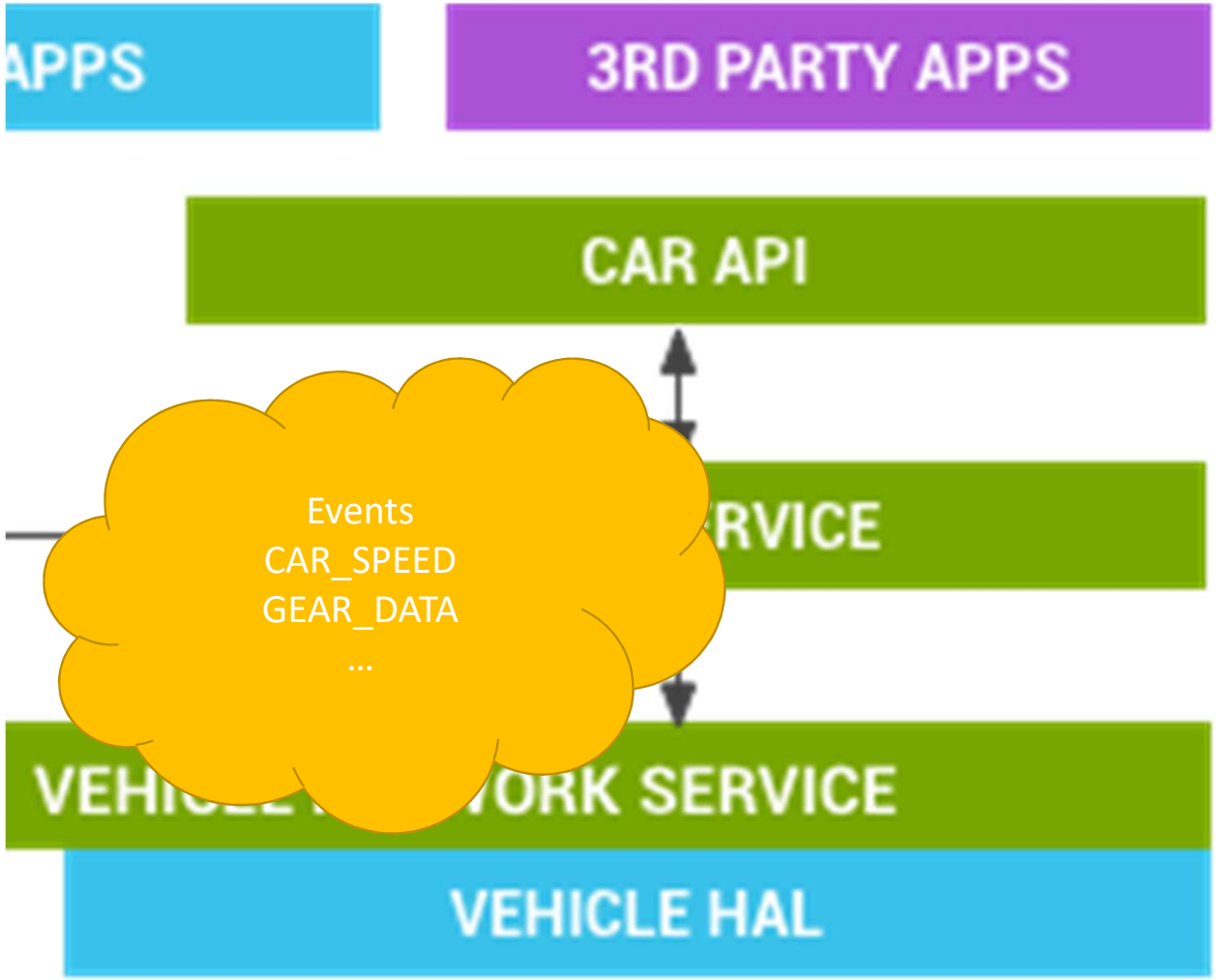


Android CAR API

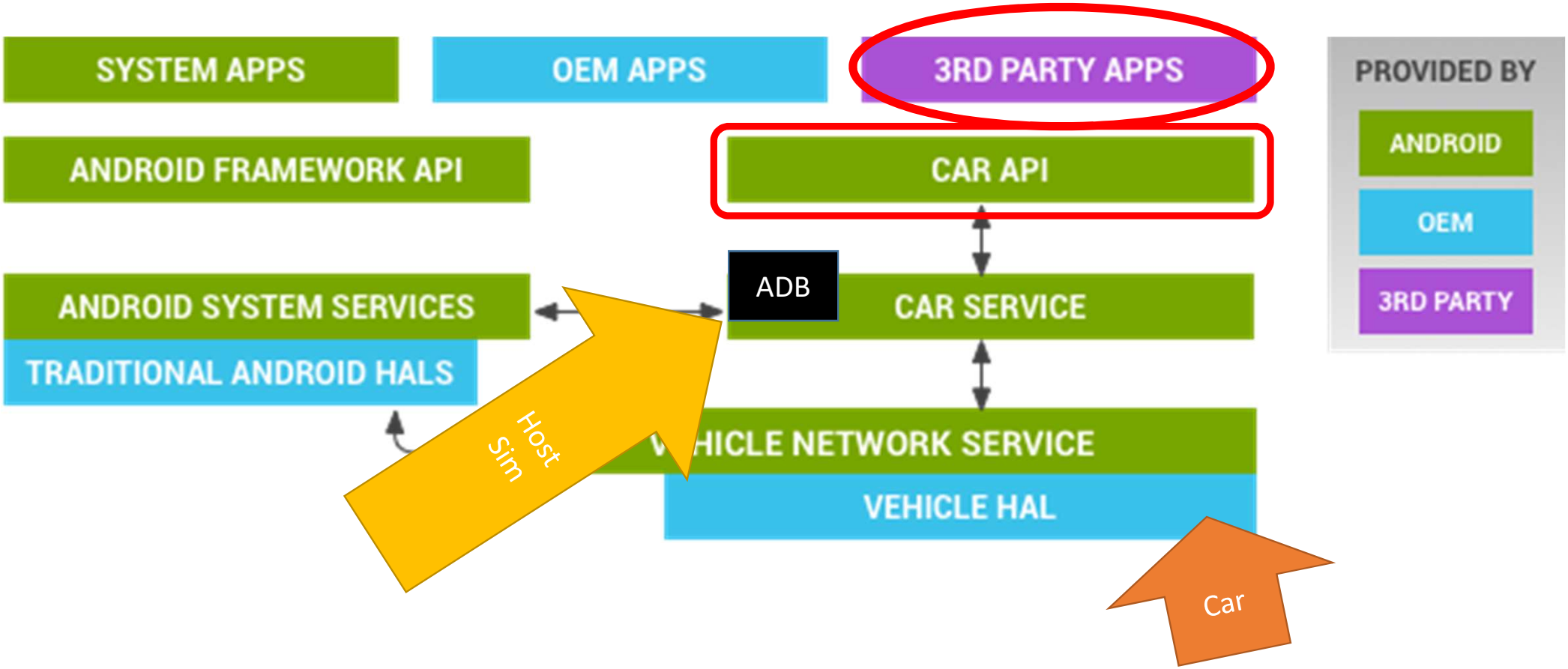


Android CAR API

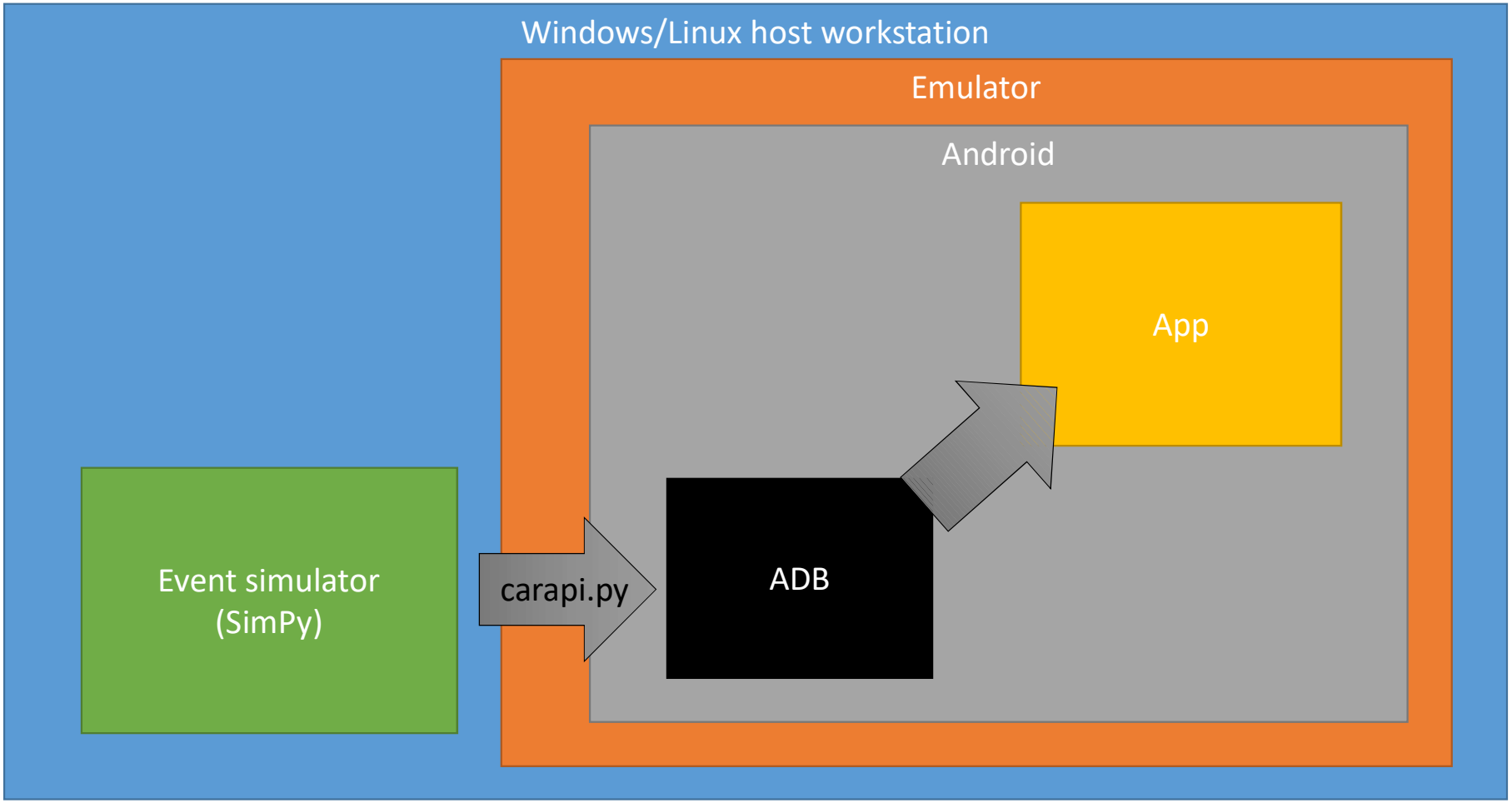
types.hal



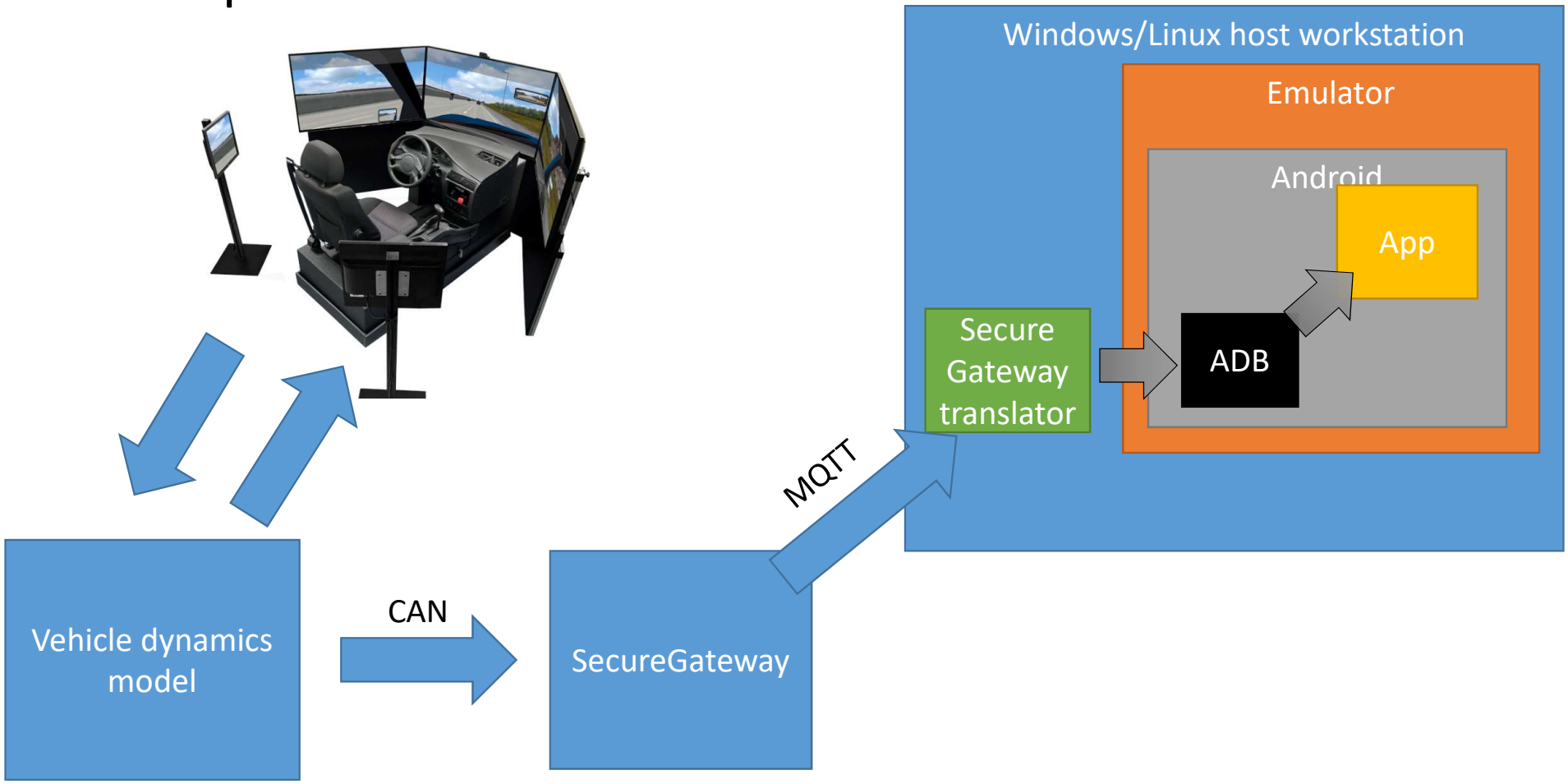
Android CAR API



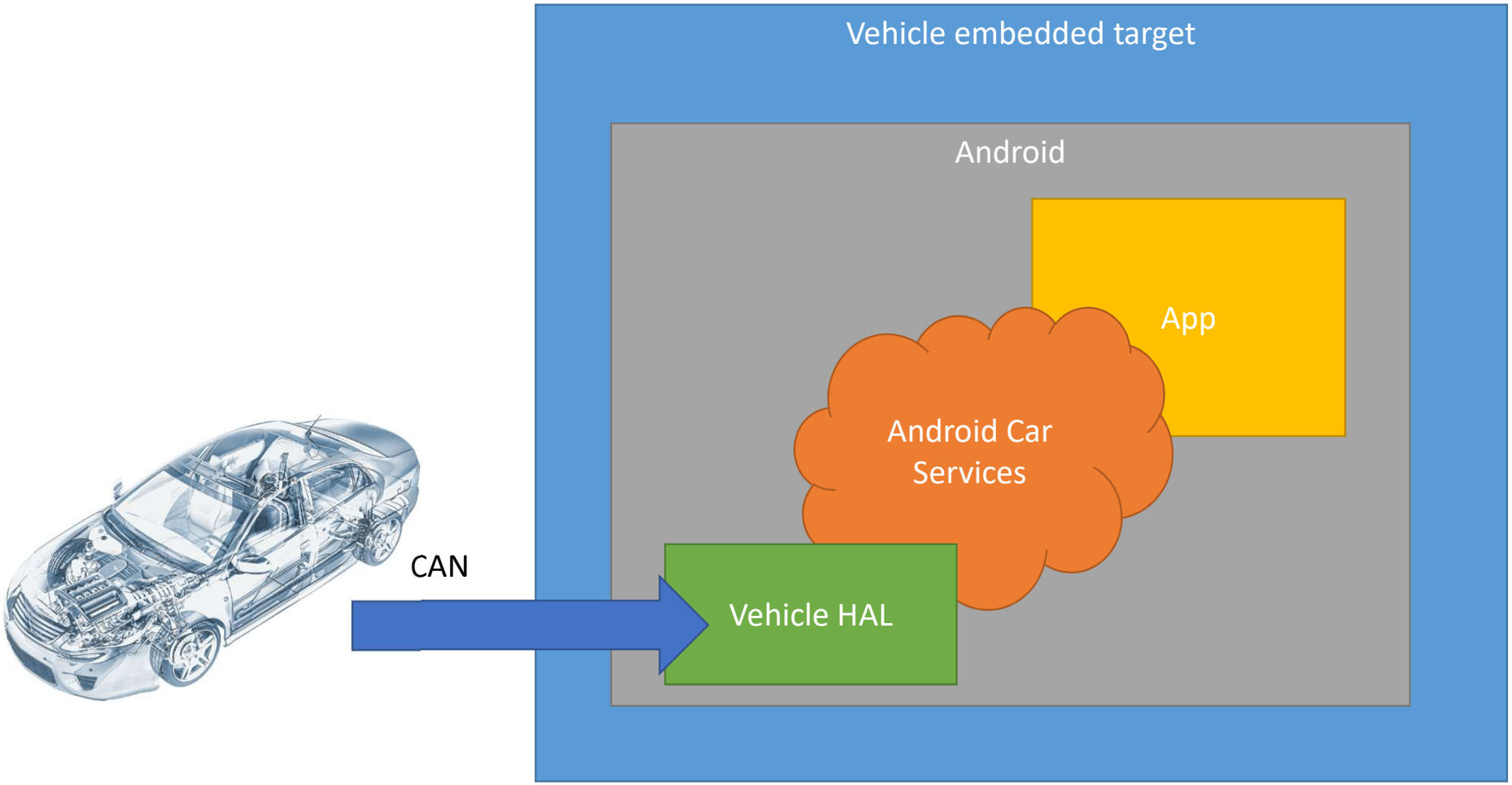
Development framework - host



Development framework - simulator



Finally… the car



Writing APIs

- All good, robust, usable code is based on modular design
- Well written component APIs tend to facilitate good, robust and usable applications
- Reuse

Thus…

- Allow API design to be an explicit, conscious part of your work
  - Take the time
  - Review
  - Rewrite, a published public API is forever (!)

Writing APIs

- Easy to learn and memorize
  - Consistent, “power-to-weight ratio”
- Leads to readable code
  - Naming conventions, prose
- Hard to misuse
  - Return codes and exceptions, parameters
- Easy to extend
  - Minimal, orthogonal
- Complete
  - Cover specifications

Document your API

- …for real: *document* your API! Classes, methods, constants, …
- …did you read the above line? Either way, read it again!

```
class Car {
…
/** Get current vehicle speed
  Current speed is defined as the mean over 100 ms.
 \return Vehicle speed in m/s.
*/
float speed();
…
};
```

Confused?

- Don’t worry, it won’t go away
- Learn to live with uncertainty and embrace change
- Be curious and tenatious, test, try, fail, discard, learn, kill your darlings
- …it will get easier, so that you may tackle harder tasks
- Never, never ever, be clever
- Never, never ever, cheat
- Thank you!