ROS 2 on Autonomous Vehicles

Christopher Ho
Sumanth Nirmal
Juan Pablo Samper
Serge Nikulin
Anup Pemmaiah
Dejan Pangercic
Jan Becker

Apex.AI®

Shinpei Kato

Tier IV

Introduction

Autonomous vehicles will…

…give hours back to commuters,

…change the way the world is connected,

…disrupt industries, and

…generate lots of value.

Autonomous vehicles are big robots with…

…sensors,

…actuation, and

…lots of algorithms,

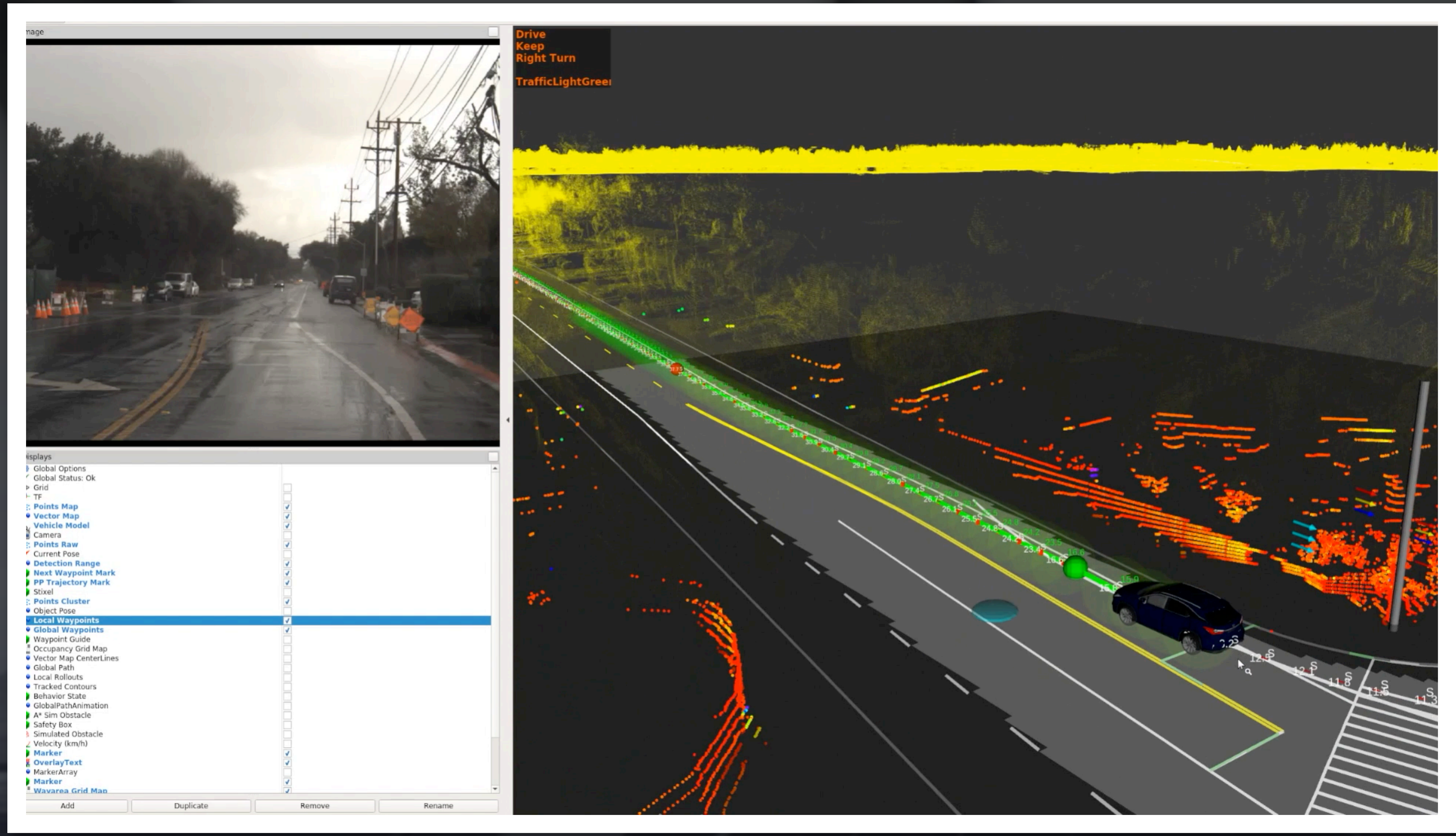
…and they can cause a lot of damage.

Can we make an autonomous vehicle[1] using ROS 2?

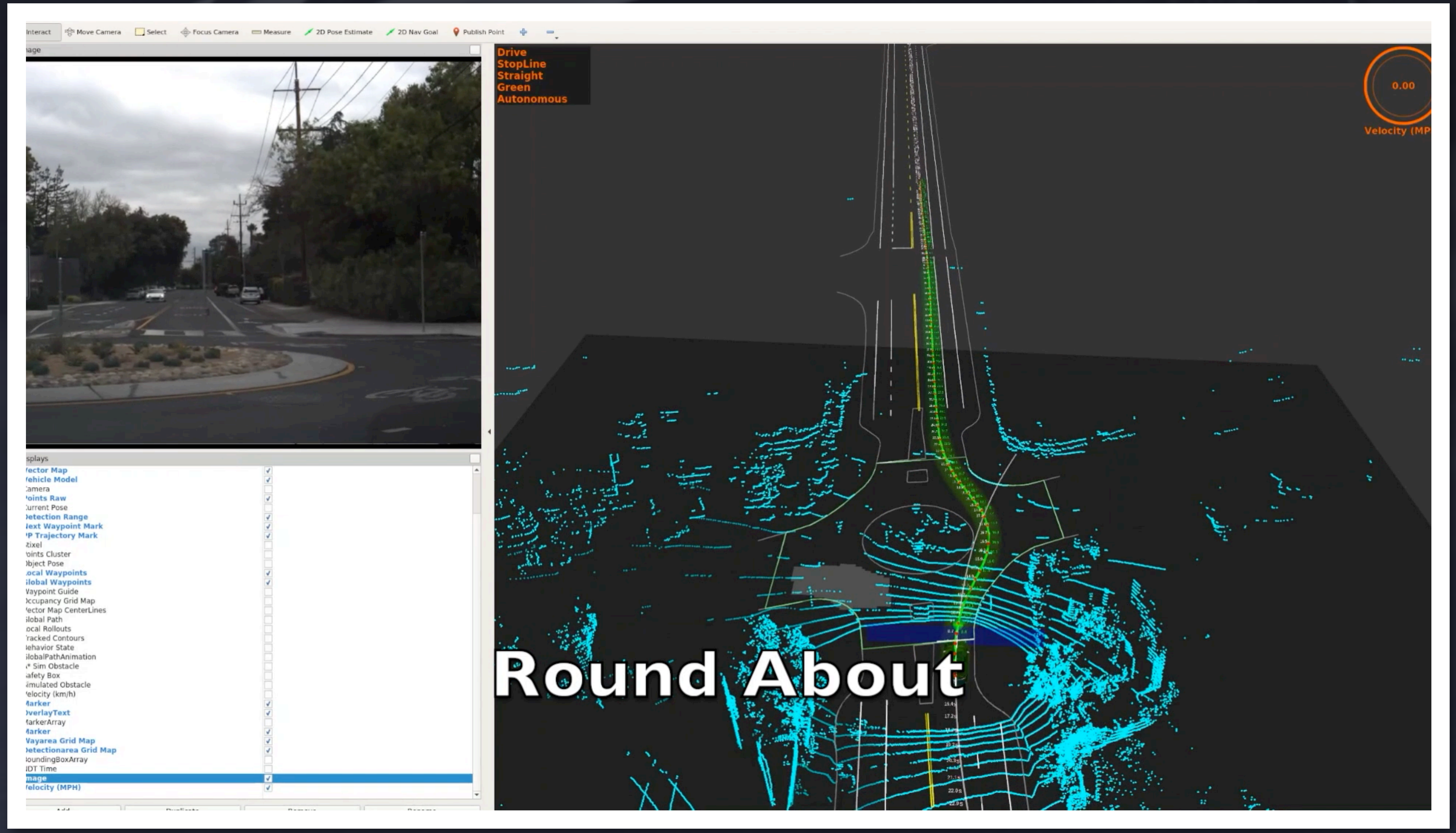
Yes[2]

1) a large robotic system in a safety critical application.

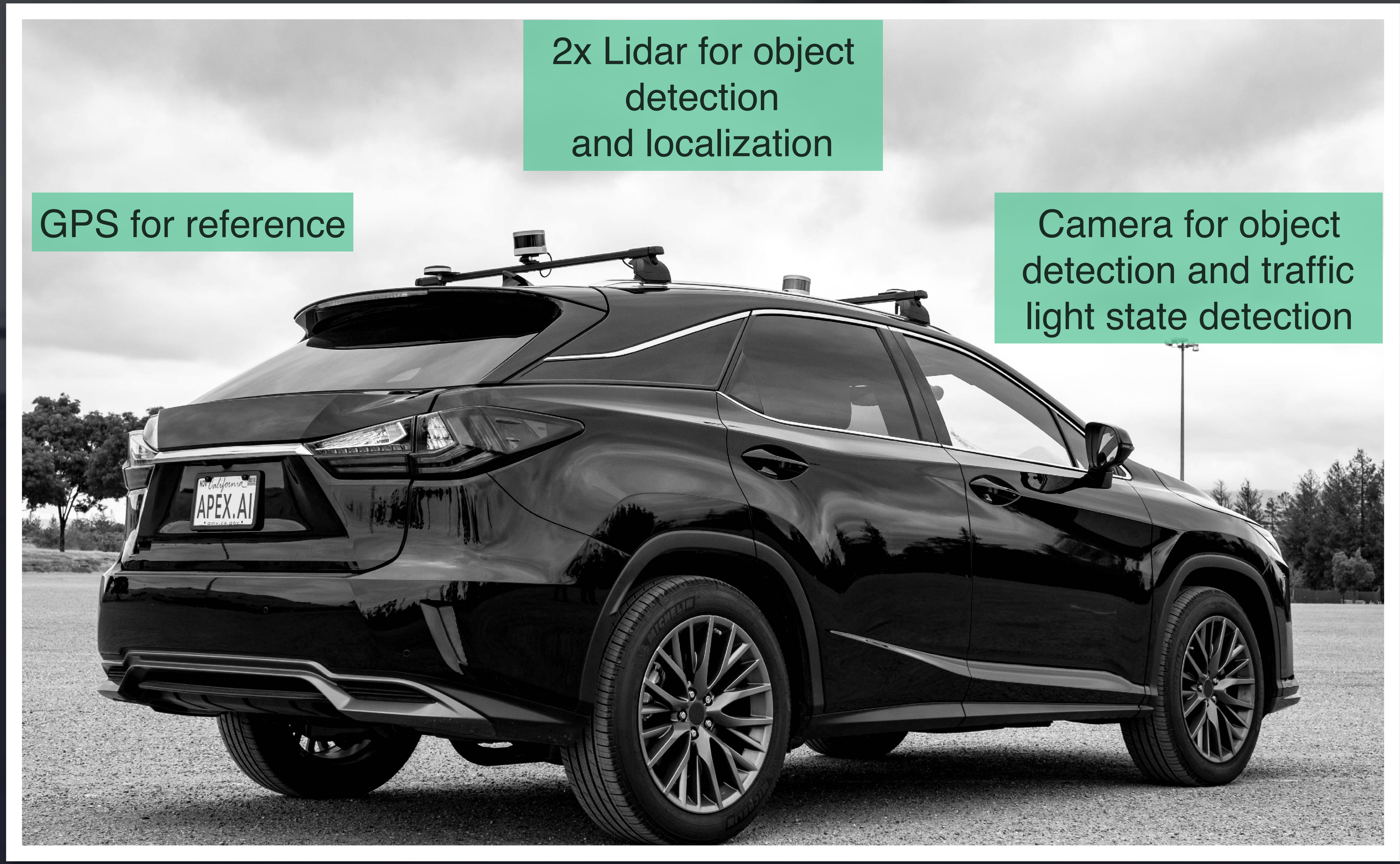
2) with caveats.

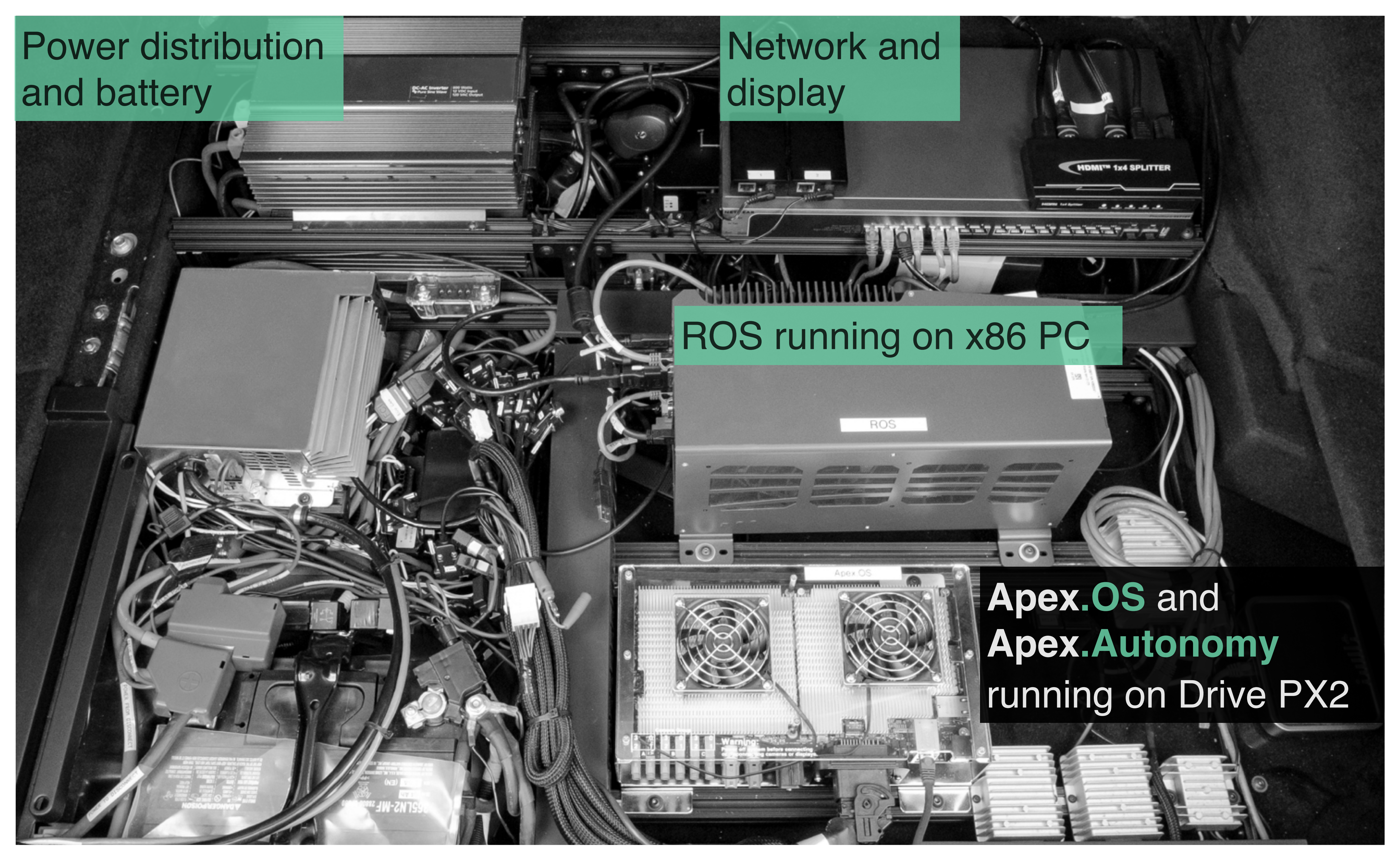

Drive
Keep
Right Turn
TrafficLightGree


4 Way Stop Intersection
Drive
Keep
Straight
Green
Autonomous

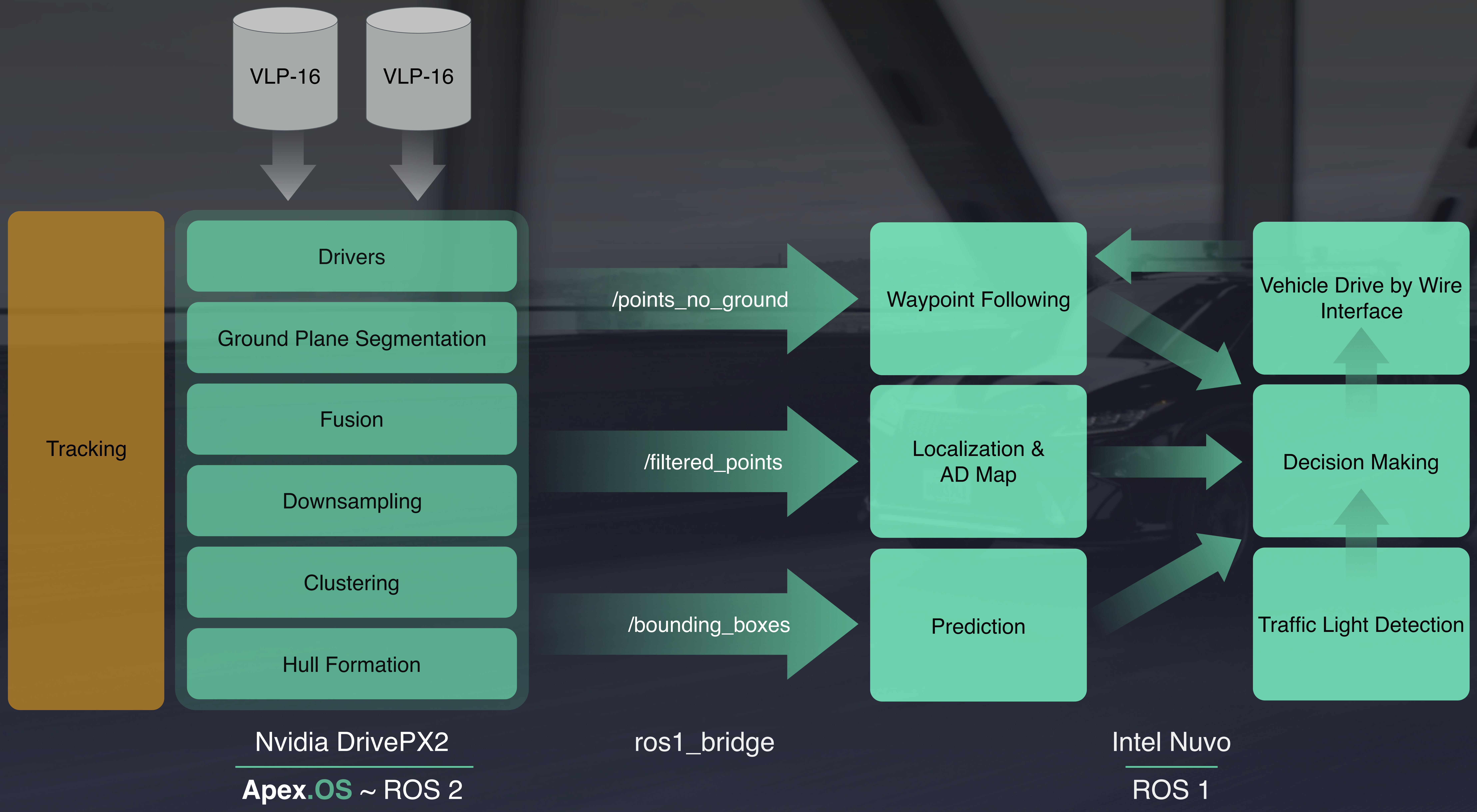

Drive
StopLine
Straight
Green
Autonomous
Round About

Our Autonomous Driving Setup





Our ROS 2 - ROS 1 Setup



Recap: ROS 2 vs. ROS 1

ROS 2 ⟷ ros1_bridge ⟷ ROS

ROS 2 has all the core features needed to build a large robot

- Node API, topics, services
- Parameter server
- Command line introspection
- Composition
- TF

In addition to extra features missing from ROS 1

- Deterministic roslaunch
- Rclcpp_lifecycle
- DDS (best effort and reliable QoS)
- Data Security
- Layered architecture

Why we still need ROS 1

- Legacy Algorithms
- Rviz
- Rosbag
- Rqt_graph
- Rqt_plot
- Gazebo
- Console Logging (to file, rosout topic)

Limitations of ROS 2 for Autonomous Driving

ROS 2 is missing features needed for safety-critical applications

1. Hard Real-Time
   - OS Primitives (memory, synchronization)
   - Real Time Logging
   - Waitsets
   - Large Memory Support
2. Robustness and Security
   - Managed System
   - OS Security
3. Testing and Certification

Apex.OS is an automotive ROS 2 for safety-critical applications

Automotive ROS 2: Hard Real Time

Deterministic resource usage and runtime is necessary for a safety critical system

- Memory
- Threads
- Blocking calls

ROS 2 is still too dynamic for hard real-time

1. Memory
   - Allocation on subscription
   - std::string
   - std::vector
   - std::exception
2. Blocking calls
   - fprintf
   - fwrite
3. Non-RT DDS Implementation

To bridge the gap to hard real-time

- No resource allocation during runtime
- All operations are finite and bounded
- All potentially blocking calls have timeouts

Automotive ROS 2: Real Time Logging

Printing console is a nondeterministic blocking call

```
// Trace of RCLCPP_INFO call:
RCLCPP_INFO(logger, "foo");
RCUTILS_LOG_INFO_NAMED(logger.get_name(), "foo");
RCUTILS_LOG_COND_NAMED(...);
// ...
rcutils_log(...);
(*output_handler)
  (location, severity, name ? name : "", now, format, &args);
output_handler = g_rcutils_logging_output_handler;
g_rcutils_logging_output_handler =
  rcutils_logging_console_output_handler(...);
// Calls
fprintf(...);
```

A purpose-built real-time logger was built instead

- Logging call uses deterministic atomic operations
- Writes to a self-healing, fail-resistent ring-buffer in shared memory
- Buffer can be flushed with minimal overhead

```
#define APEX_PRINT(...) \
  apex::console::print( \
    static_cast<uint32_t>(__LINE__), \
    __FILE__, \
    __VA_ARGS__)
APEX_PRINT("Debug float value", 32.23F);
```

```
14941| apex_console_logging   | 2018-08-30 17:15:54.319515| Version: 0.0.0
14942| apex_console_logging   | 2018-08-30 17:15:54.319520| Formal build: No
14943| apex_console_logging   | 2018-08-30 17:15:54.319521| Debug float value: :+32.23
14944| apex_console_logging   | 2018-08-30 17:15:55.319628| Debug integer value: :-32
14945| apex_console_logging   | 2018-08-30 17:15:56.319741| This is a debug message
```

Automotive ROS 2: Waitsets

Callbacks are the primary mechanism by which ROS handles the receipt of interprocess communication

```
const auto subscriber1_ptr =
  node_ptr->create_subscriber<std_msgs::msg::String>(
  "Topic1",
  bar);
const auto subscriber2_ptr =
  node_ptr-
>create_subscriber<geometry_msgs::msg::PointStamped>(
  "Topic2",
  foo);

// foo and bar get executed in an arbitrary order
rclcpp::spin(node_ptr);
```

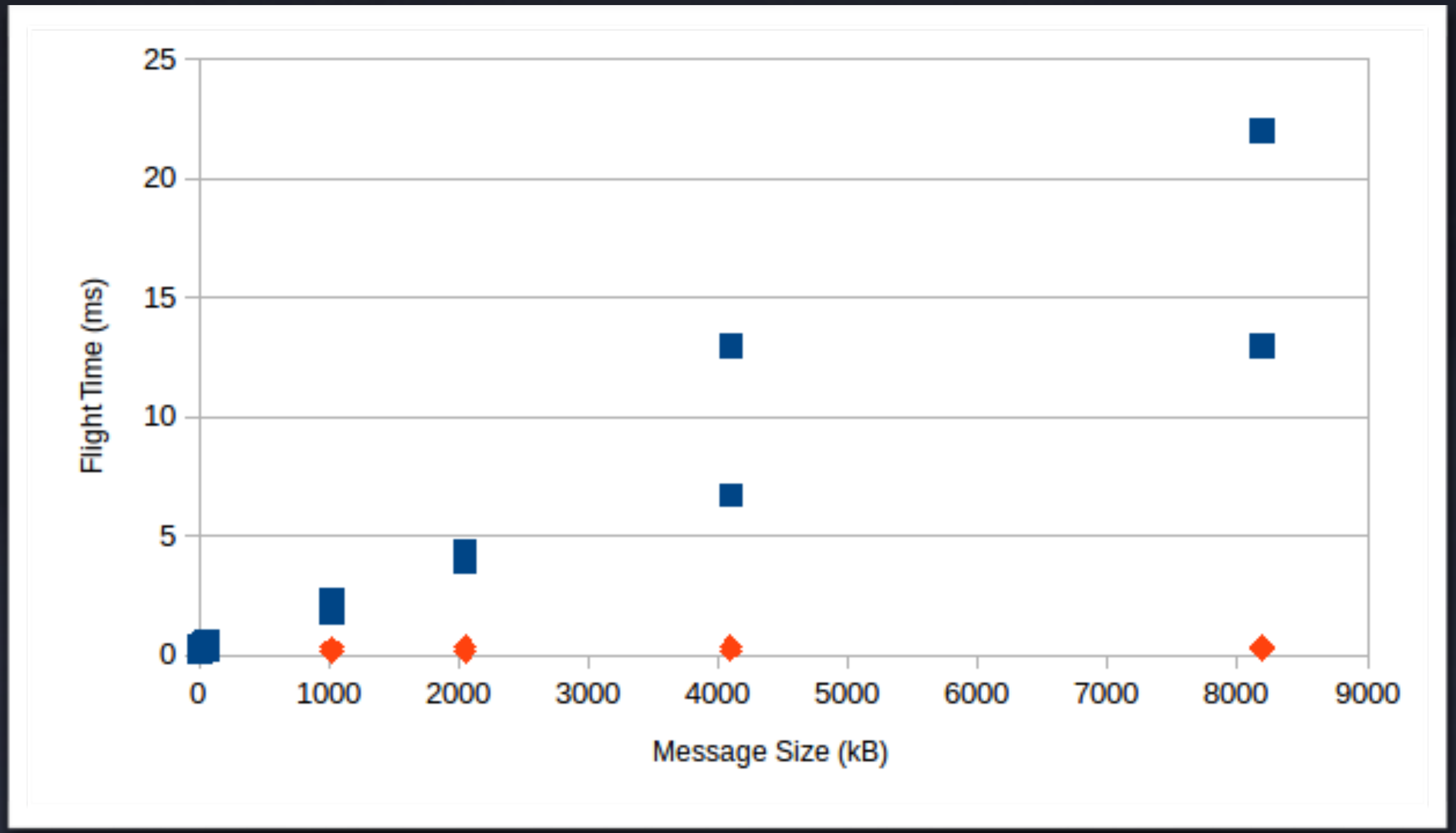
Waitsets better lend themselves to a deterministic execution order and error handling

```
rclcpp::Node node("Node");
auto sub1 =
  node.create_subscriber<std_msgs::msg::String>(
  "Topic1");
auto sub2 =
  node.create_subscriber<geometry_msgs::msg::PointStamped>(
  "aTopic2");
rclcpp::Waitset<2> ws({sub1, sub});

// Wait for 5 seconds.
ws.wait(5s);

auto msgs1 = sub1->take();
if(msgs1) {
  // always update to latest msgs2 if available
  // before acting on msg1
  auto msgs2 = sub2->take();
  if(msgs2) {
    handle_sample(msg2.data());
  }
  handle_sample(msg1.data());
} else {
  // react to not receiving msgs1 in time
}
```

Automotive ROS 2: Large Memory Support

The maximum size of a UDP packet is 64kB

- Messages larger than 64kB require fragmentation
- Large messages are slower to transmit
- Exchanging pointers (8 B) to memory locations in shared memory is significantly faster for large data




```
/* SHM Publish */
// Initialize
apex::shared_memory::ShmArray<BigMsg> shm_pub(num_frames,
topic.c_str());
const auto pub_ptr = node_ptr-
>create_publisher<std_msgs::msg::Uint64>(topic);

// publish: write message to shared memory
BigMsg big_msg;
const uint64_t frame_num = 0U;
shm_pub[frame_num] = big_msg;
// publish: send frame number via DDS
std_msgs::msg::Uint64 msg;
msg.data = frame_num;
pub_ptr->publish(msg);

/* SHM Subscribe */
// Initialize
const apex::shared_memory::ShmArray<BigMsg> shm_sub(num_frames,
topic.c_str());

auto cb = [&](const std_msgs::msg::Uint64::SharedPtr msg) {
  // copy large message to local context
  // could also manipulate in shared memory for zero copy
  local_big_msg = shm_sub[msg->data];
};

const auto sub_ptr = node_ptr-
>create_publisher<std_msgs::msg::Uint64>(topic, cb);
```

Automotive ROS 2: Managed System

Deterministic startup order of nodes is important for large systems

- ROS 2 launch (Python) provides this capability
- ROS 2’s managed nodes allow individual nodes to react to failures

ROS 2 lacks mechanisms for the whole system to react to node failures

- Heartbeat (detect silent failures)
- Lifecycle Manager (coordinate system level responses)
- Shadow nodes (instant failure response for critical systems)
- Consensus

Automotive ROS 2: Security

ROS 2 exposes three kinds of security from DDS

- Message encryption
- Authentication
- Access Control
- Data Tagging
- Logging

This is insufficient to guard against corrupted or malicious binaries

- Memory hoggers
- CPU stressors
- Tailor-made DDS participants

ROS 2 also lacks key mechanisms such as

- Secure over-the-air (OTA) updates
- Secure key storage
- Integration with existing security infrastructure

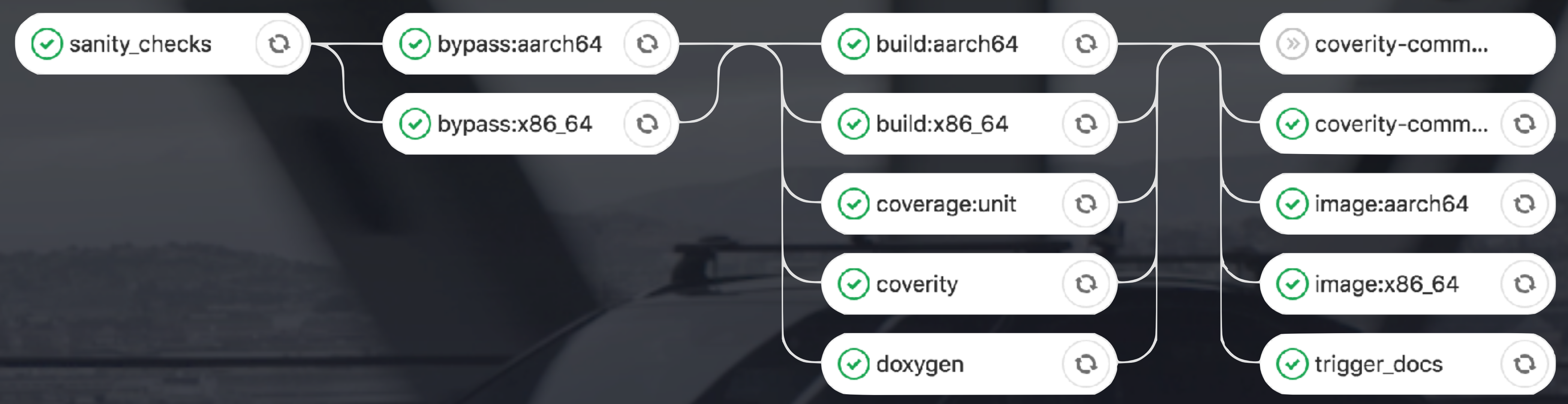
Automotive ROS 2: Testing and Certification

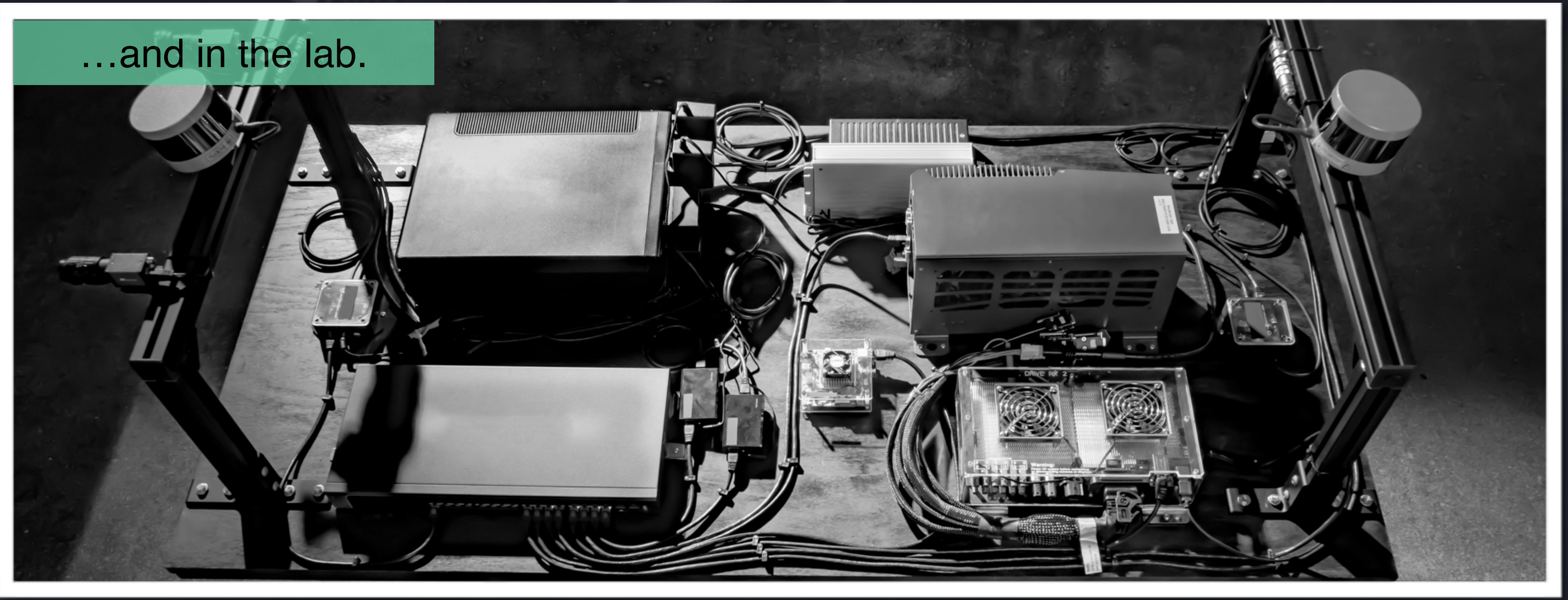
How do you prove code is safe?

Follow a functional safety standard (ISO 26262):

1. Analyze your use case
   - Write requirements
2. Follow a process
   - Document everything
   - Follow a coding standard
   - Analyze your code
   - Do code reviews
3. Write tests
   - Unit, integration, full stack, stress, fault, injection, requirements
   - SIL, HIL (every supported ECU, sensor)
   - Line, branch, MC/DC coverage

Testing in the cloud…




…and in the lab.

Giving Back

Tools	Static Exceptions
	Agile Development Environment
	(Inter-Process Communication) Performance Test
	ament_pclint
	AutowareAuto
ROS 2 Features	YAML Parameter Parser
Bugs	osrf_testing_tools_cpp
	orocos_kinematics_dynamics
	rcutils

Building Algorithms for Safety Critical Applications

Case Study: ROS 1 Velodyne Driver

As is:	Changes:
Memory allocation per packet	All memory allocation on startup
Nodelets as a proxy for threading	Threading with controllable stack size, priority, etc.
OS-specific system calls	OS-agnostic system calls
No failure handling	Notifies user on UDP timeout
Coarse point cloud discretization	Firing-level point cloud discretization
Non-RT logging	RT logging
Integration tests	Unit, integration, stress tests

Conclusion

ROS 1 is great

- Lots of tools and algorithms
- Community
- But can never be automotive grade

ROS 2 is even greater

- Enough features for serious development
- API is stable enough
- Can work with ROS 1
- Can be automotive grade

Apex.OS is automotive grade ROS 2

- Real-time
- Secure
- ISO 26262 certified

Apex.AI® is hiring!

We are actively recruiting developers and engineers:

- Framework
- Embedded
- Security
- Certification
- Algorithms

If you are interested to learn more, talk to us at ROSCon or apply at www.apex.ai

ROS 1 PubSub vs. Apex.OS PubSub