ROS 2 as a Cybernetic Framework

Lyle Johnson
Sr. Field Application Engineer

Apex.AI®

Introduction

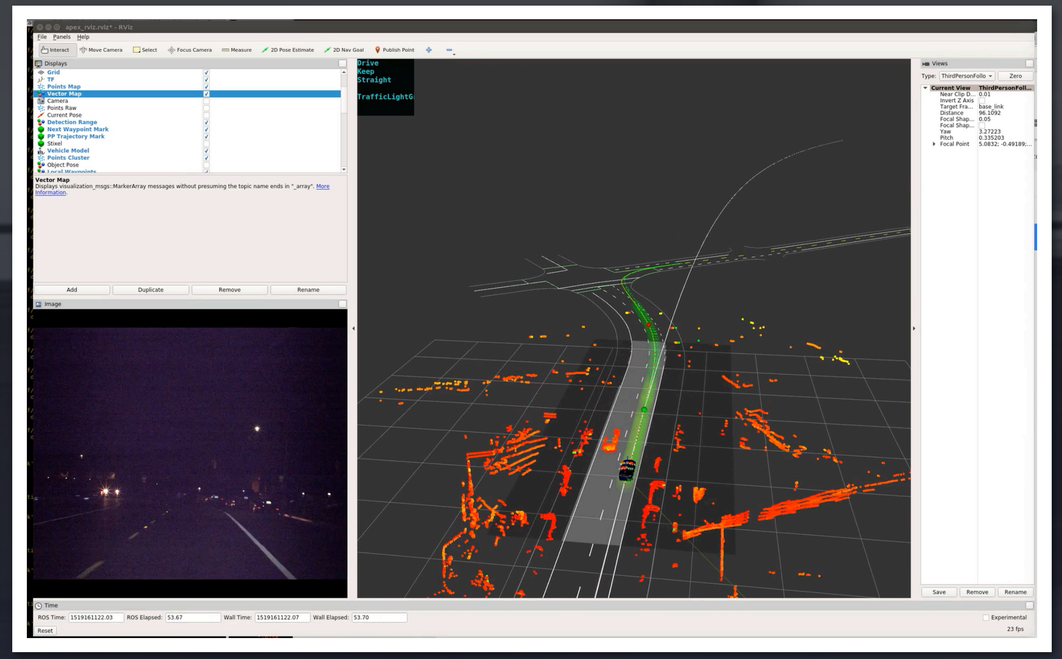
Autonomous vehicles will:

- Give hours back to commuters
- Change the way the world is connected
- Disrupt industries

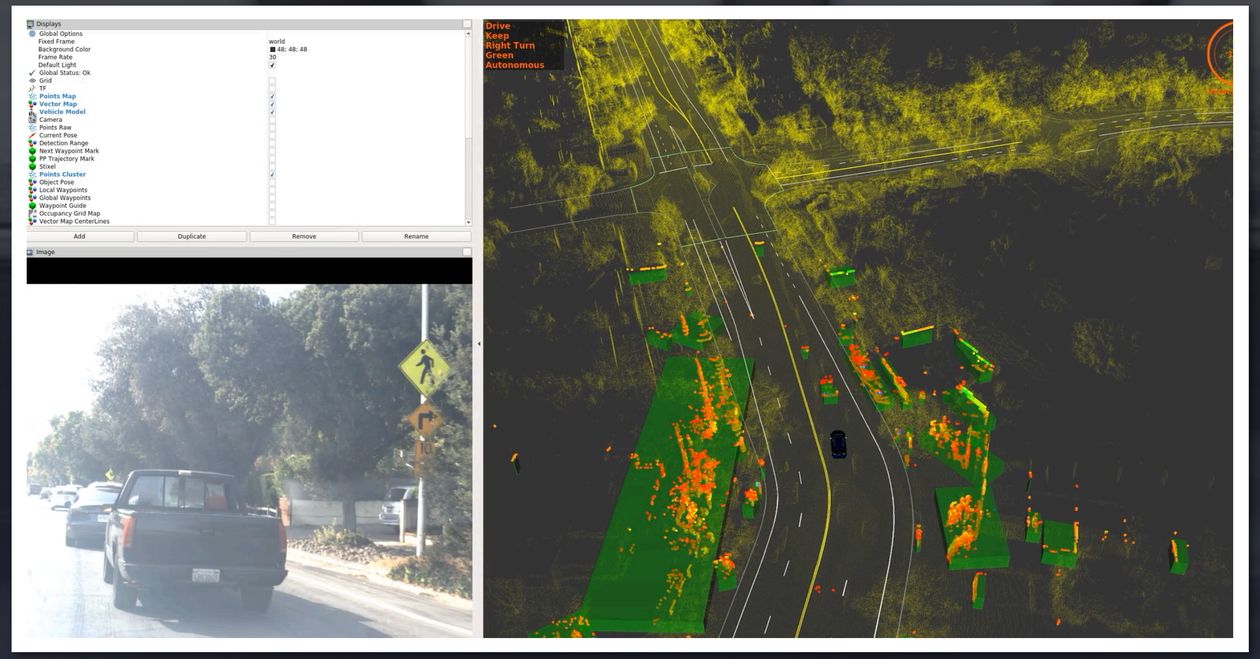
Autonomous vehicles are big robots (cybernetic system) with:

- Sensors
- Actuators
- Many algorithms
- Ability to cause a lot of damage

Driving is dangerous and stressful



Almost head-on crash



Overtaking double yellow line

Autonomous driving from different angles

1. High Level Questions

- Will it happen? When?
- How will it happen?
- Which application will be first?
- Where will it happen first?

2. Technology

- LiDAR [Yes / No]
- HD Maps [Yes / No]
- DL [Yes / No]
- Real world vs. virtual testing
- Frameworks: ROS 2, Adaptive AUTOSAR, etc.

3. Business

- Incumbents vs. challengers
- Will we continue to buy cars in the future?
- Insurance

4. Societal

- Accidence rates
- How will cities change?
- When will it be illegal to drive?

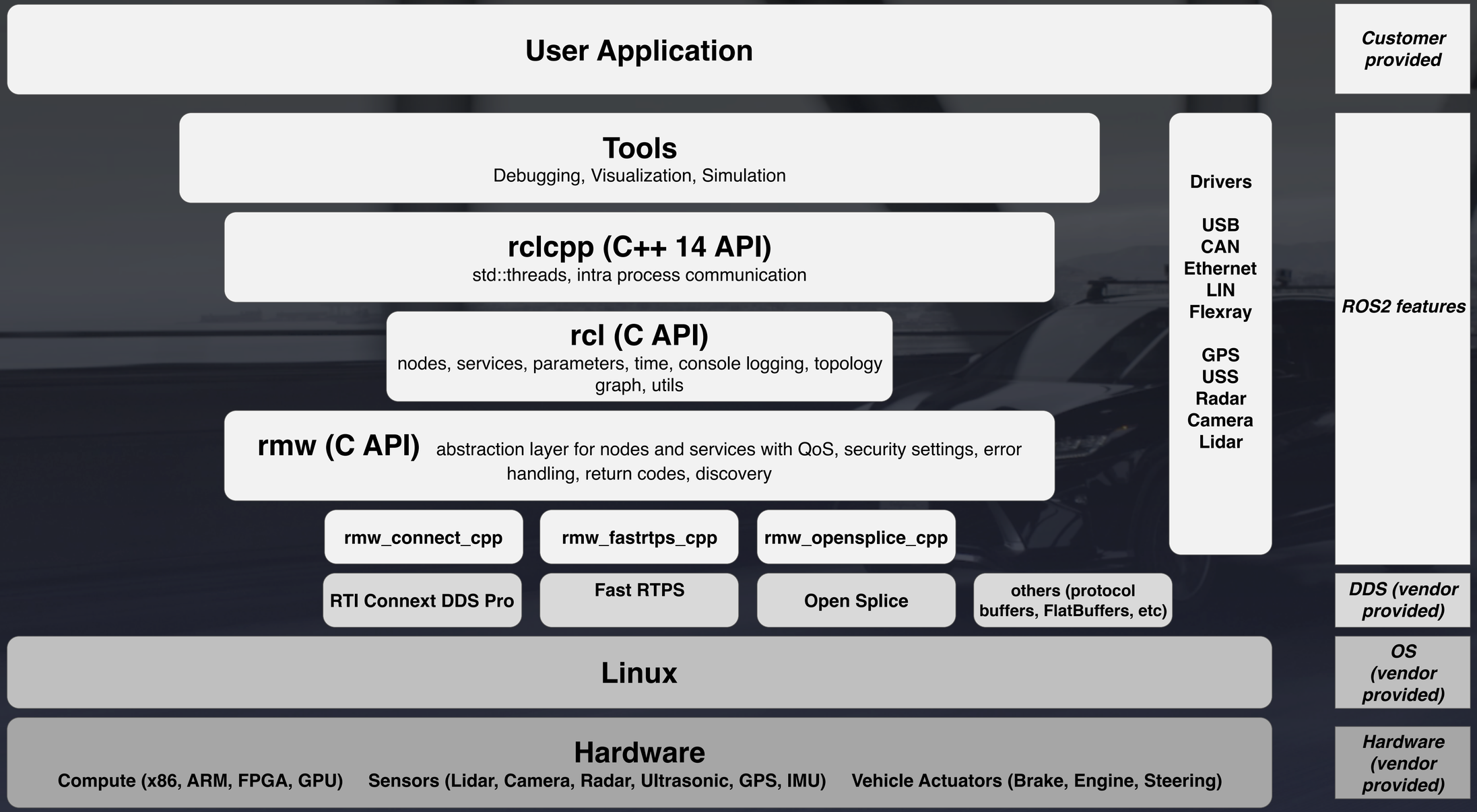
ROS 2 introduction

- **ROS 1** came out of Stanford University in 2007 as was developed for many years by Willow Garage
  - Apex.AI considers ROS 1 as the most widespread and feature rich robotics framework available
  - ROS 1 enables developers to write **hardware agnostic** applications and to **rapidly prototype and debug applications**
- Open Robotics began the **ROS 2** project in 2015 to address **shortcomings of ROS 1**, WRT production robotics environments
  - Antiquated APIs
  - Low code quality
  - Non-standard middleware
  - Lack of real-time capabilities
  - Nodes without managed lifecycles
  - Lack of security features
  - Lack of testing and documentation
  - Lack of automotive ECU support

ROS 2 introduction

- Open Robotics considered autonomous vehicles and the following use cases **before starting development for ROS 2**
    - Small embedded platforms
        - Smaller, optimized code base compared to ROS 1
        - Targets the **NVIDIA Drive PX 2** and **Renesas R-Car H3**
    - Real-time systems
        - **Memory management**, **communication mechanism**, and **threading model** allow for ROS 2 to be real-time (upon hardening)
    - Production environments
        - ROS 2 can run on an RTOS such as **QNX**
    - Prescribed patterns for building and structuring systems
        - **Lifecycle management** and **static configurations** for deployment
    - System architecture that supports **hardware abstraction** (next slide)

ROS 2 software architecture



ROS 2 system architecture

roslaunch on ECU 1

Production Process

Executor
Responsible for handling thread queues.

Node **Node**

Node **Node**

Non-Production Process
(Testing, Visualization, etc.)

Node **Node**

Node **Node**
Publishers
Subscribers
Services
Parameters

Parameter Server

Pure DDS Process

Camera

roslaunch on ECU 2

Production Process

Executor
Responsible for handling thread queues.

Node **Node**

Node **Node**
Publishers
Subscribers
Services
Parameters

Pure DDS Process

Pure DDS Process

Connext DDS Micro, FastRTPS, etc.

Shared Memory / UDP | UDP | Shared Memory / UDP

```
…
auto message = std_msgs::msg::Image();
message.data = get_camera_image();
publisher_->publish(message);
…
```

```
…
void image_callback(const std_msgs::msg::Image::SharedPtr msg){
cv::Mat frame(msg->height, msg->width, encoding2mat_type(msg->encoding),
const_cast<unsigned char *>(msg->data.data()), msg->step);
viewImage(frame); }
…
```

ROS 2 summary

In summary, the following shortcomings from ROS 1 were addressed in ROS 2

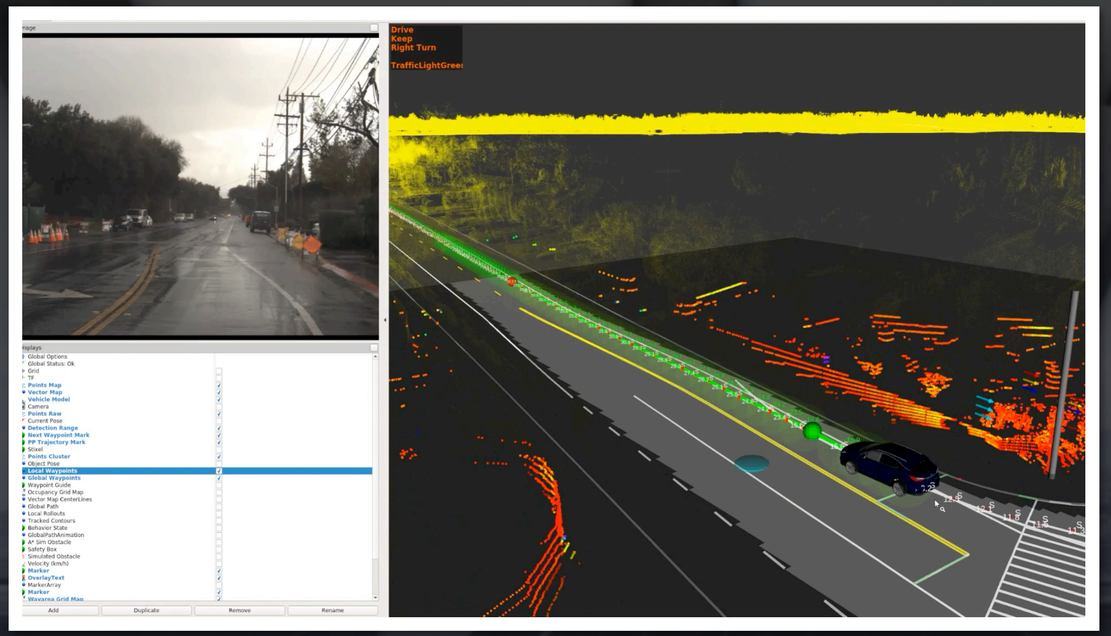
- **Improved APIs**
  - APIs are using C++ 14
- **Improved code quality**
  - Modern Continous Integration (CI) infrastructure
    - https://ci.ros2.org
- **Standardized middleware**
  - DDS: well defined data model and offers Quality of Service (QoS) configuration
- **Real-time capable**
  - Note that ROS 2 **is not** real-time out-of-the-box
- **Managed nodes with lifecycle**
  - Managed nodes and launch system
- **Native security mechanisms**
  - Authentication, access control list, and encryption
- **Testing and documentation**
  - Code linters, unit & integration tests, CI on x86_64 and aarch64
  - Design documentation is added to the existing tutorial and API documentation

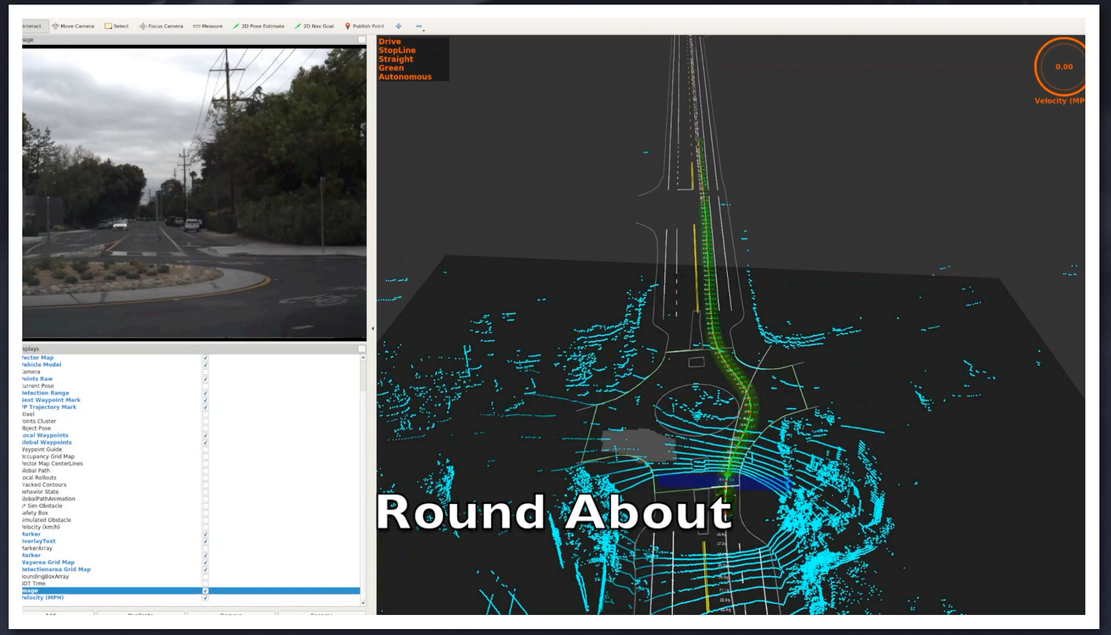
Can we make an autonomous vehicle[1] using ROS 2?

Yes[2]

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

[2)] with caveats.





Why do we need a framework like **Apex.OS**?

1. To abstract away sensor, actuation, and OS interfaces
2. Ease of use (topic names, discovery mode, message creation in DDS, remapping / name-spacing, etc.)
3. Separation of concerns (5Cs)

Apex.OS provides

- Nodes as Finite State Machines (important for fail safe / fail operational)
- System for deterministic node startup / stop
- Very well defined message interfaces (matured over the years, efficient in-memory representations)
- Services and Actions
- Tools (visualization, simulation, large data recording, etc.)
- Lots of packages for tasks common in robotics (cartographer, Autoware, g2o, etc.)
- Community

Why do we need a framework like **Apex.OS**?

roslaunch on ECU 1

Production Process

Executor
Responsible for handling thread queues.

Node **Node**

Node **Node**
Publishers
Subscribers
Services
Parameters

Non-Production Process
(Testing, Visualization, etc.)

Node **Node**

Parameter Server

Pure DDS Process

roslaunch on ECU 2

Production Process

Executor
Responsible for handling thread queues.

Node **Node**

Node **Node**
Publishers
Subscribers
Services
Parameters

Pure DDS Process

Pure DDS Process

Camera

Connext DDS Micro, FastRTPS, etc.

Shared Memory / UDP | UDP | Shared Memory / UDP

```
…
auto message = std_msgs::msg::Image();
message.data = get_camera_image();
publisher_->publish(message);
…
```

```
…
void image_callback(const std_msgs::msg::Image::SharedPtr msg){
cv::Mat frame(msg->height, msg->width, encoding2mat_type(msg->encoding),
const_cast<unsigned char *>(msg->data.data()), msg->step);
viewImage(frame); }
…
```

Apex.OS: abstracting ROS 2 system architecture into a few lines of code

```
…
auto message = std_msgs::msg::Image();
message.data = get_camera_image();
publisher_->publish(message);
…
```

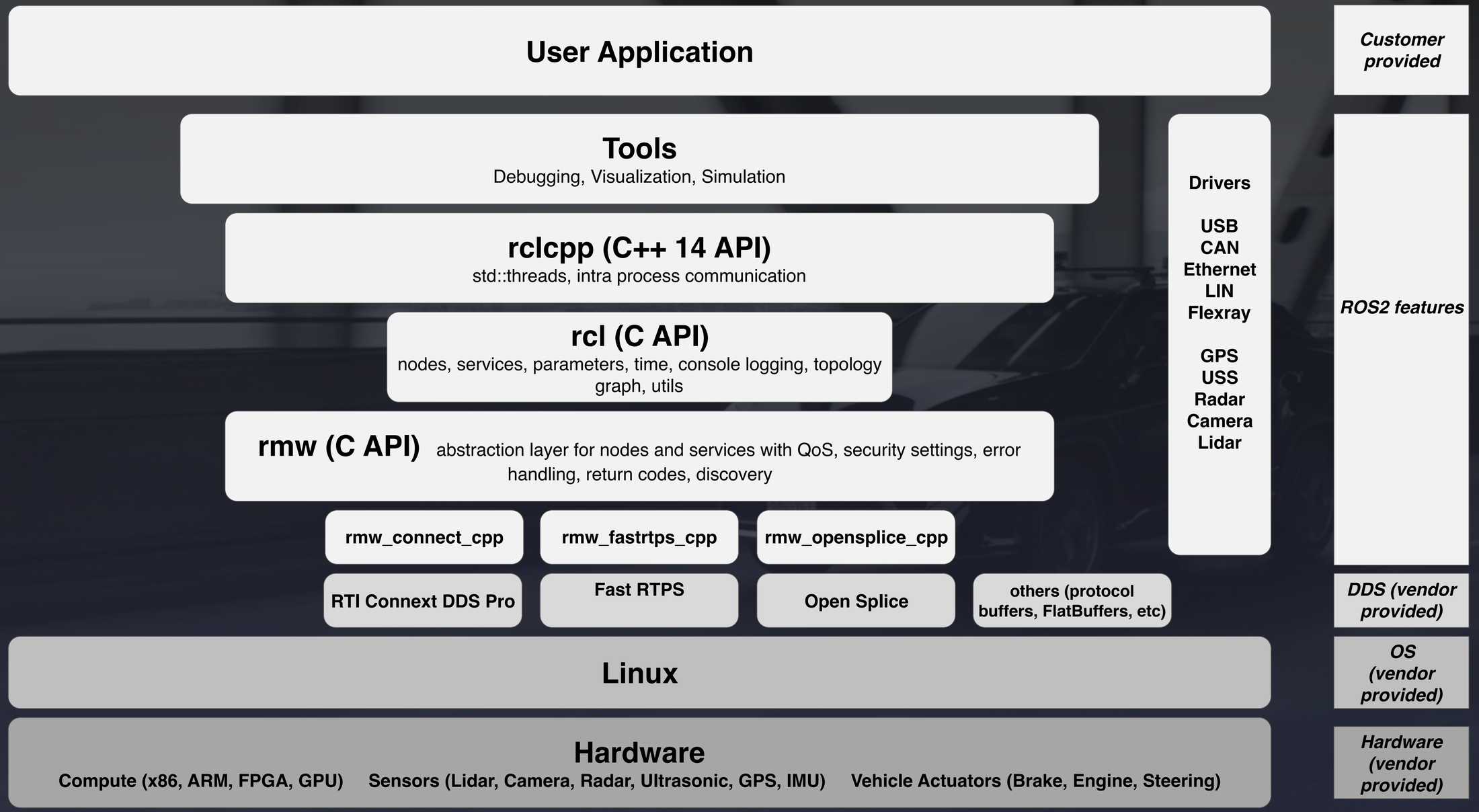
```
…
void image_callback(const std_msgs::msg::Image::SharedPtr msg){
cv::Mat frame(msg->height, msg->width, encoding2mat_type(msg->encoding),
const_cast<unsigned char *>(msg->data.data()), msg->step);
viewImage(frame); }
…
```

Apex.AI modifications to ROS 2

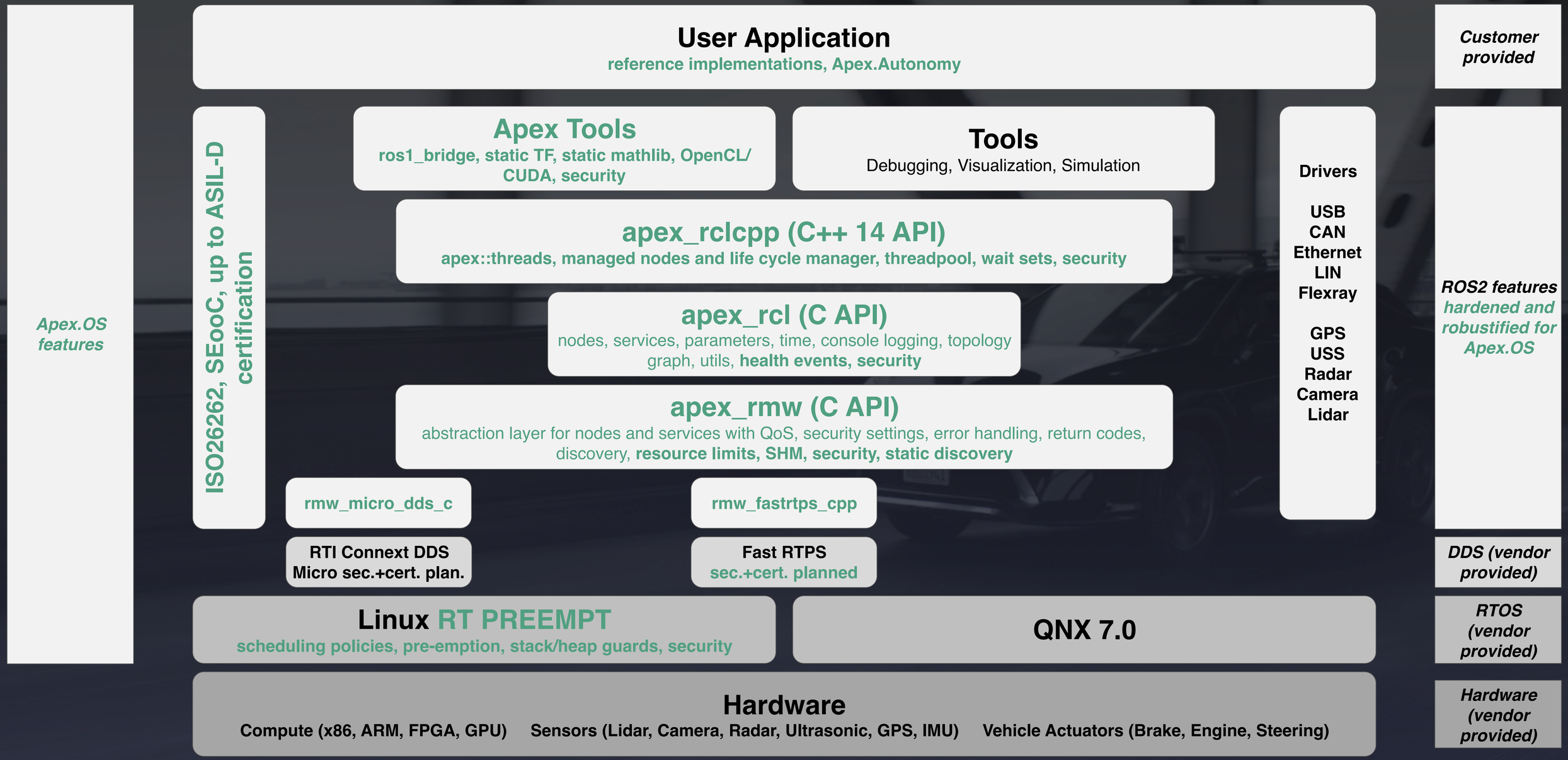
Apex.AI provides the following improvements and extensions built on top of ROS 2, which are collectively **released as Apex.OS**

- Static memory
- Hard real-time
- Callbacks vs Waitset
- Security
- Testing
- Certification
- Real-time I/O logging
- Production and research environments side-by-side
- Apex.OS extensions
- Support

ROS 2 software architecture



Apex.OS software architecture



Apex.OS features

Real-time	BSP support	Tools for development	Math library
RTOS support (QNX, Linux RT)	Documentation and 24/7 support	Managed nodes	Deterministic launch
Security	Certification	Time syncronization	Full testing coverage
Platform health and diagnostics	Device Drivers	aarch64 and x86_64	UDP and SHM
CAN, Ethernet, *Lin, FlexRay*	Console and data logging	Configuration	Reference implementations

Highlighted features: hard real-time

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

- Memory
- Threads
- Blocking calls

To bridge the gap to hard real-time

- No resource allocation during runtime
- Use of static constructs
  - apex::string
  - apex::vector
  - apex::exception with memory pool
- All operations are finite and bounded
- All potentially blocking calls have timeouts
- Use RT DDS Implementation (e.g. Connext DDS Micro)

Highlighted features: real time logging

A purpose-built real-time logger was built

- 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
```

Highlighted features: 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);
```

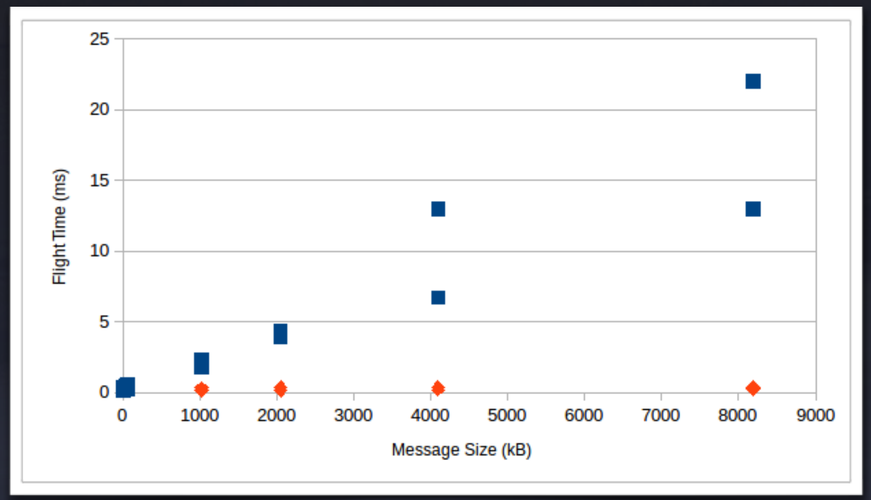
Waitsets better lend themselves to a deterministic execution order and error handling

```
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
}
```

Highlighted features: large data 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_subscription<std_msgs::msg::Uint64>(topic, cb);
```

Available in Connext DDS Micro EAR from January 2019

Highlighted features: managed system

Deterministic startup order of nodes is important for large systems

- roslaunch2 provides this capability
- Managed nodes allow individual nodes to react to failures

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

Highlighted features: security

Apex.OS exposes three kinds of security from DDS

- Message encryption
- Authentication
- Access Control

Adding guards against corrupted or malicious binaries

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

Key mechanisms for security integration:

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

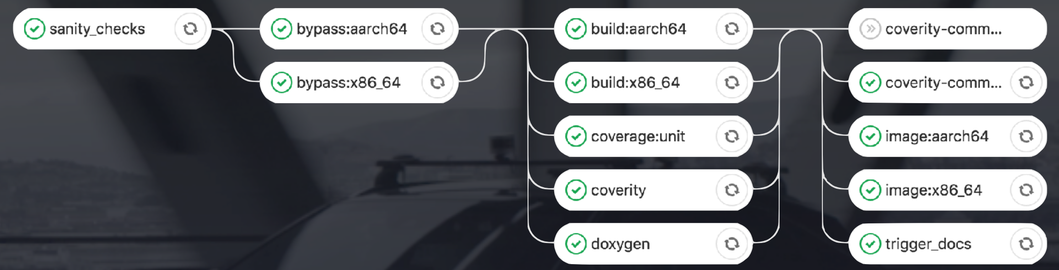
Highlighted features: testing and certification

How do you prove code is safe?

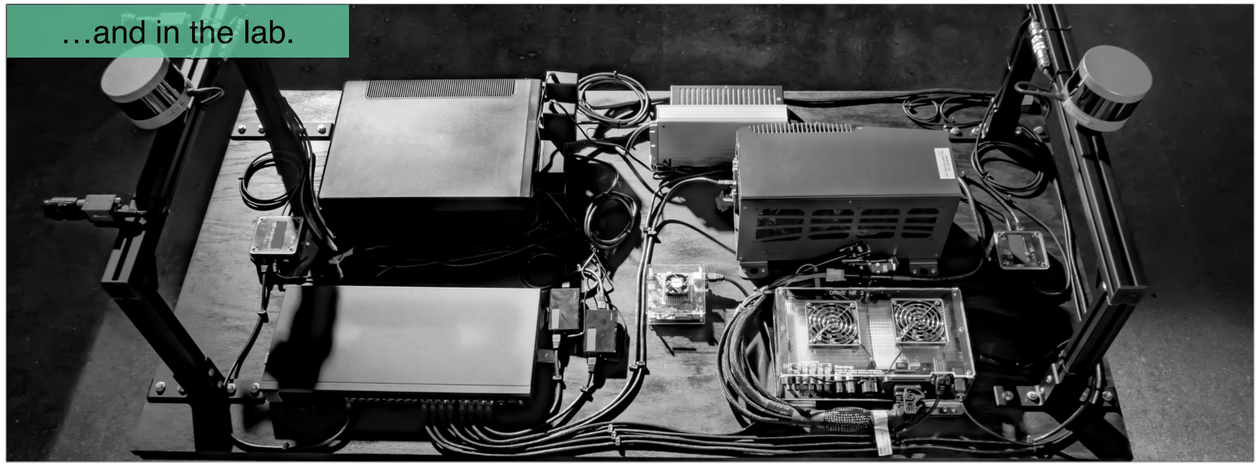
Follow a functional safety standard (ISO 26262)

- Analyze your use case
  - Write requirements
- Follow a process
  - Document everything
  - Follow a coding standard
  - Analyze your code
  - Host regular code reviews
- Write tests
  - Unit, integration, full stack, stress, fault injection, fuzzy, mutation, anomaly
  - Requirements
  - SIL, HIL (on every supported ECU & sensor)
  - Line, branch, function, and MC/DC coverage

Testing in the cloud…



…and in the lab.



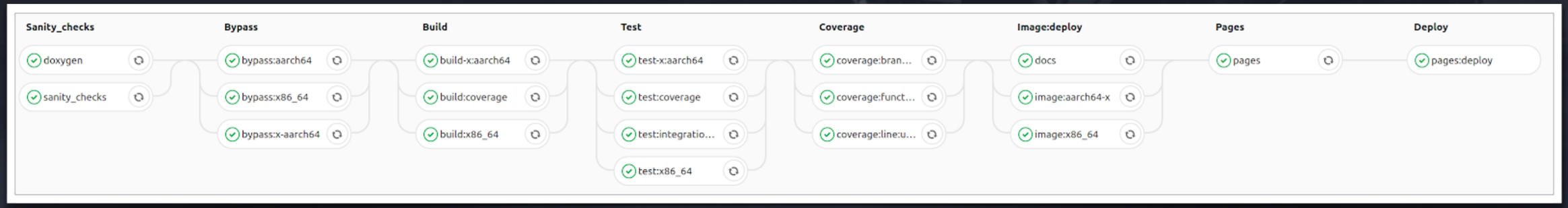
Highlighted features: testing and certification

Code Coverage

Current view: top level - src/apex_core/apexutils/src
Test: lcov.total.filtered
Date: 2018-10-20 05:57:02
Legend: Rating: low: < 75 % | medium: >= 75 % | high: >= 90 %

	Hit	Total	Coverage
Lines:	1686	1710	98.6 %
Functions:	230	230	100.0 %
Branches:	997	1018	97.9 %

Filename	Line Coverage		Functions		Branches	
apex_console.c	100.0 %	151 / 151	100.0 %	18 / 18	100.0 %	70 / 70
apex_dir.c	100.0 %	40 / 40	100.0 %	6 / 6	100.0 %	28 / 28
apex_event.c	100.0 %	98 / 98	100.0 %	9 / 9	100.0 %	78 / 78
apex_health.c	100.0 %	67 / 67	100.0 %	9 / 9	100.0 %	12 / 12
apex_memory.c	100.0 %	35 / 35	100.0 %	8 / 8	100.0 %	14 / 14
apex_mutex.c	100.0 %	67 / 67	100.0 %	6 / 6	100.0 %	52 / 52
apex_queue.c	100.0 %	142 / 142	100.0 %	12 / 12	100.0 %	100 / 100
apex_rt.c	40.0 %	16 / 40	100.0 %	2 / 2	12.5 %	3 / 24
apex_semaphore.c	100.0 %	52 / 52	100.0 %	6 / 6	100.0 %	38 / 38
apex_shm.c	100.0 %	93 / 93	100.0 %	11 / 11	100.0 %	68 / 68
apex_strings.c	100.0 %	112 / 112	100.0 %	22 / 22	100.0 %	34 / 34
apex_task.c	100.0 %	229 / 229	100.0 %	19 / 19	100.0 %	176 / 176
apex_time.c	100.0 %	280 / 280	100.0 %	55 / 55	100.0 %	110 / 110
apex_udp.c	100.0 %	157 / 157	100.0 %	14 / 14	100.0 %	122 / 122
apex_wraparound.c	100.0 %	45 / 45	100.0 %	12 / 12	100.0 %	44 / 44
apexutils.c	100.0 %	102 / 102	100.0 %	21 / 21	100.0 %	48 / 48



Giving back

Tools	Static exceptions
	Awesome Development Environment
	(Inter-Process Communication) Performance test
	ament_pclint
	Autoware.Auto
ROS 2 Features	YAML parameter parser
Bugs	osrf_testing_tools_cpp
	orocos_kinematics_dynamics
	rcutils

Summary

ROS 1 is great

- Many tools and algorithms
- Fostered community
- *But can never be automotive grade*

ROS 2 is even better than ROS 1

- Enough features to enable serious development
- API is mostly stable
- Can work alongside ROS 1
- Can be automotive grade

Apex.OS is an automotive grade version of ROS 2

- Real-time
- Secure
- ISO 26262 certified
- Available in **January 2019** (EAR)

Apex.Autonomy

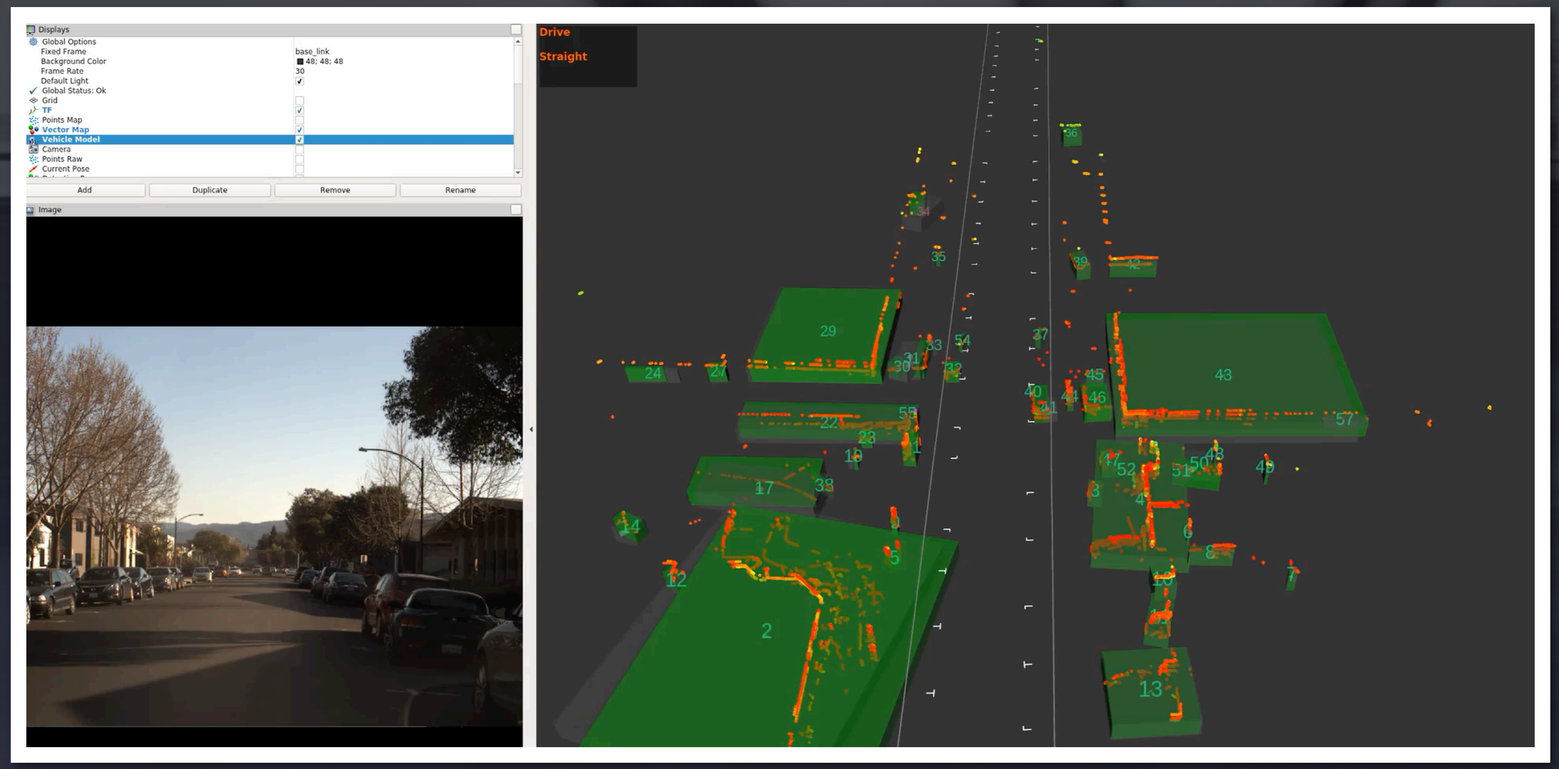
- Real-time
- Reliable
- High performance
- ISO 26262 certified

Contact: lyle.johnson@apex.ai

Apex.Autonomy - automotive grade LiDAR processing lib.

Tracking (EKF, Motion Models, Hungarian Assigner)

- Drivers
- Ground Plane Segmentation
- Fusion
- Downsampling
- Clustering
- Hull Formation



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

Apex.AI® is hiring!

We are actively recruiting the following positions:

- C++ software architect
- Robotics software
- Security
- Certification
- Algorithms
- Quality assurance
- Unix / Real-time
- Business development
- Executive assistant
- Engineering manager for open source software
- Functional safety and certification
- Middleware

If you are interested to learn more, apply at www.apex.ai