



Apex.AI

Doing Real-time With ROS 2:
Capabilities And Challenges

Dejan Pangercic

Definition of Real-time

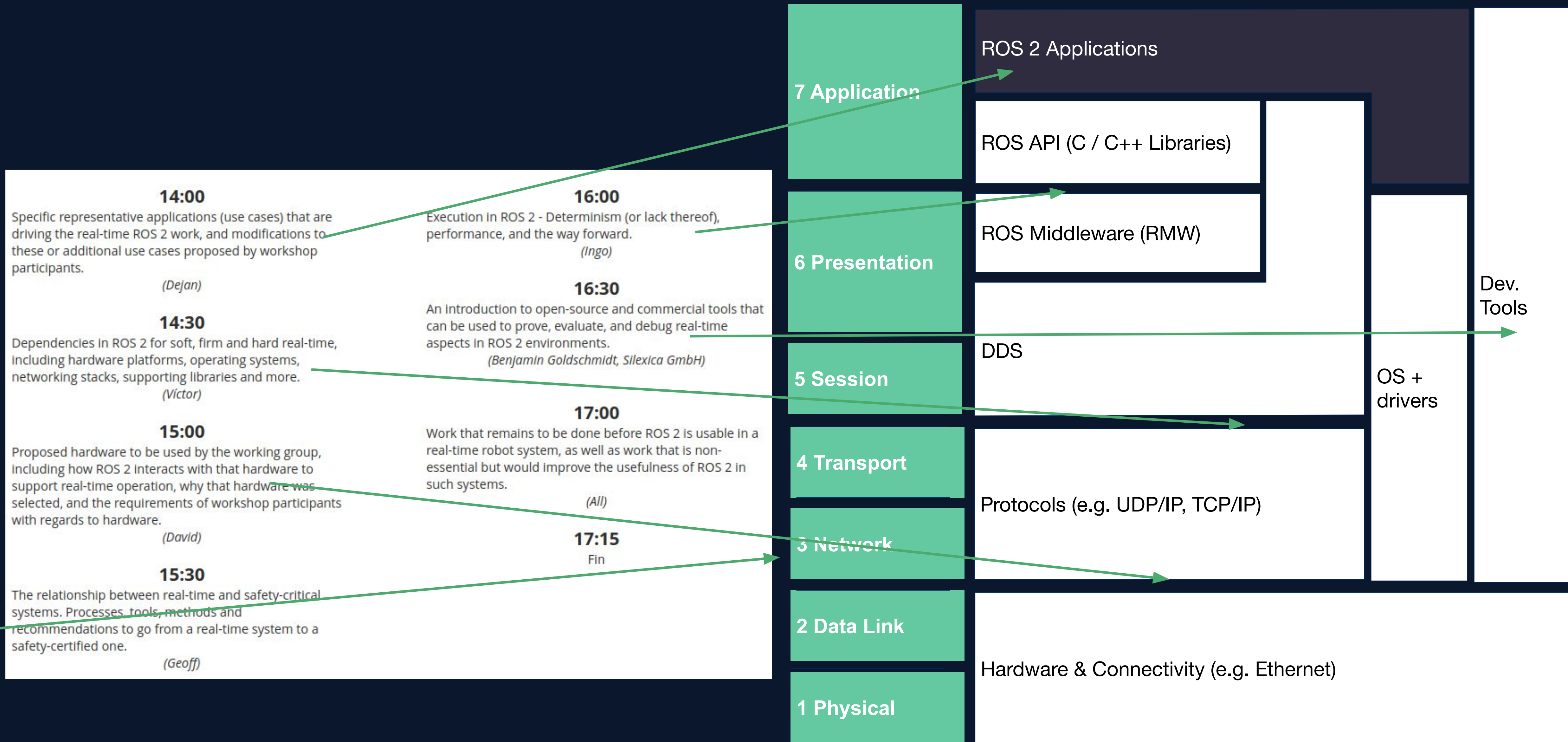
Real-time Definition (control):

Real-time (control) system means that the (control) system must provide the (control) responses or actions to the stimulus or requests within specific times, which therefore depend not just on what the system does but also on how fast it reacts.

Zhang, P. (2008). Industrial control technology: a handbook for engineers and researchers. William Andrew.

3 Different Angles

Angle 1: ROSCon 2019: Real-time Workshop



Angle 2: History - ROS and Real-time Publications

1. A motivating theoretical example
2. Introduction to real-time computing
 - Hard real-time: Missing a deadline is considered a system failure
3. Dependencies and best practices
 - RTOS: Linux RT PREEMPT, QNX
 - Middleware: DDS
 - Prioritize real-time threads
 - Avoid sources of non-determinism: memory allocations, blocking synchronization, printing and logging (to hard disk), non real-time transport protocols (TCP/IP), page faults
4. ROS 2 design
5. ROS 2 Node Lifecycle
6. ROS 2 communications: inter-process, intra-process, same-thread
7. Real-time [benchmark demo](#)
 - Maintained by Carlos and Lander now, can be seen at Alias Robotics or Apex.AI booths

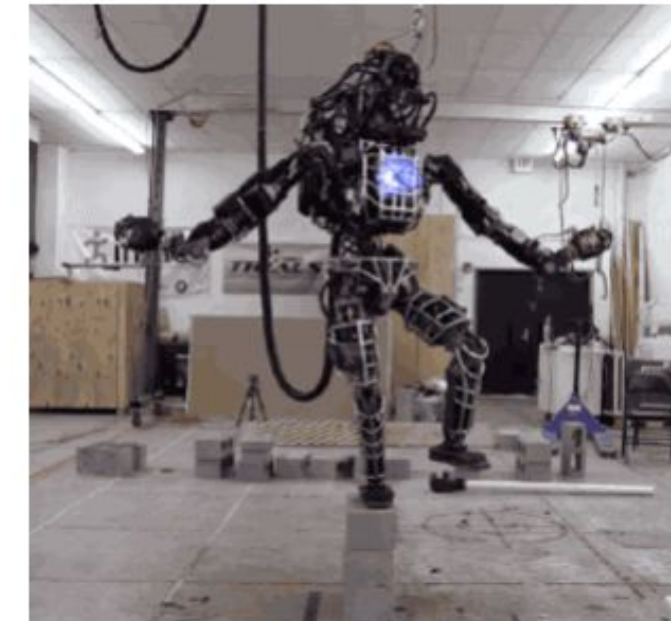
Real-time control in ROS and ROS 2.0

Jackie Kay

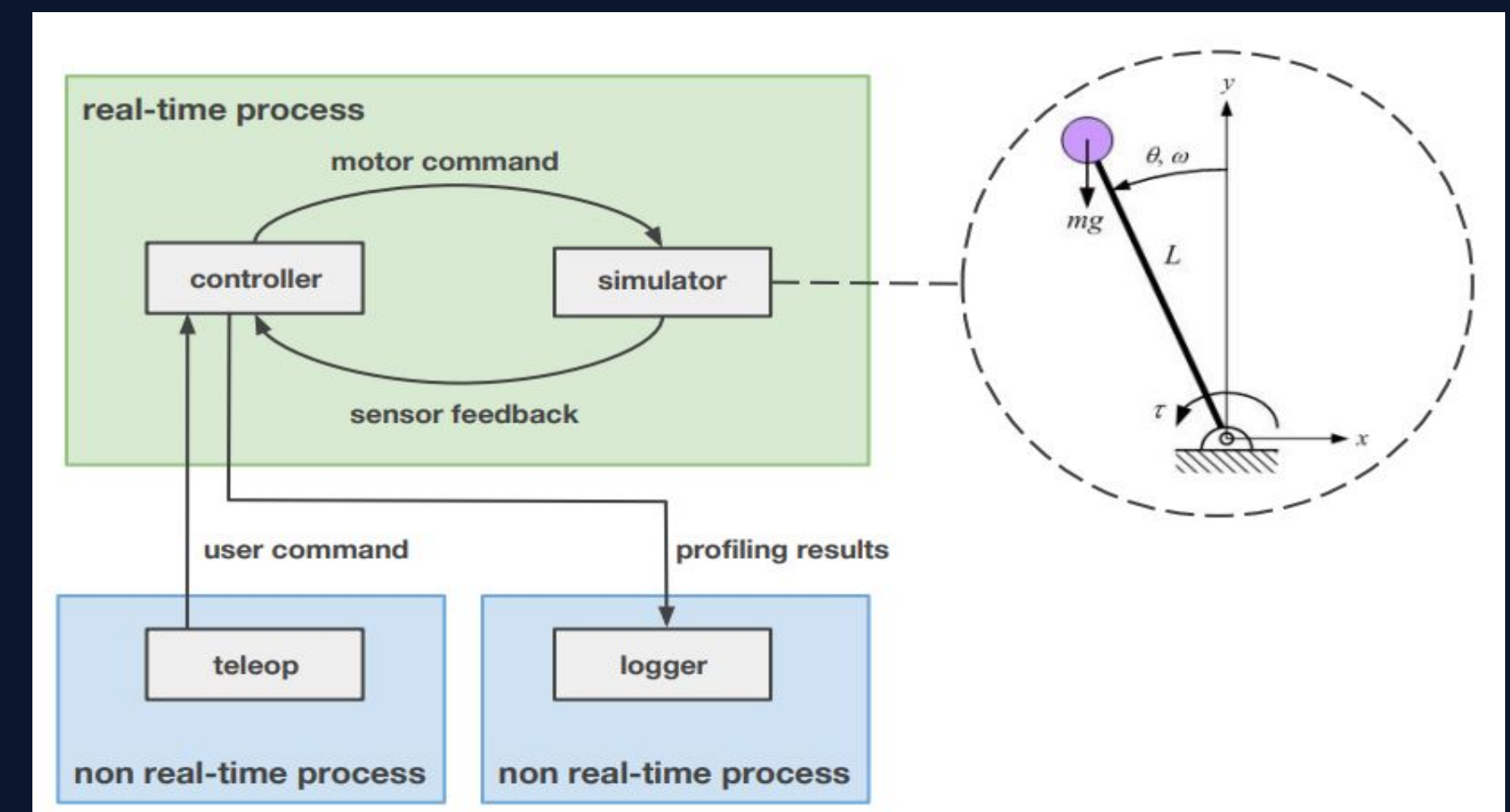
jackie@osrfoundation.org

Adolfo Rodriguez Tsouroukdissian

adolfo.rodriguez@pal-robotics.com

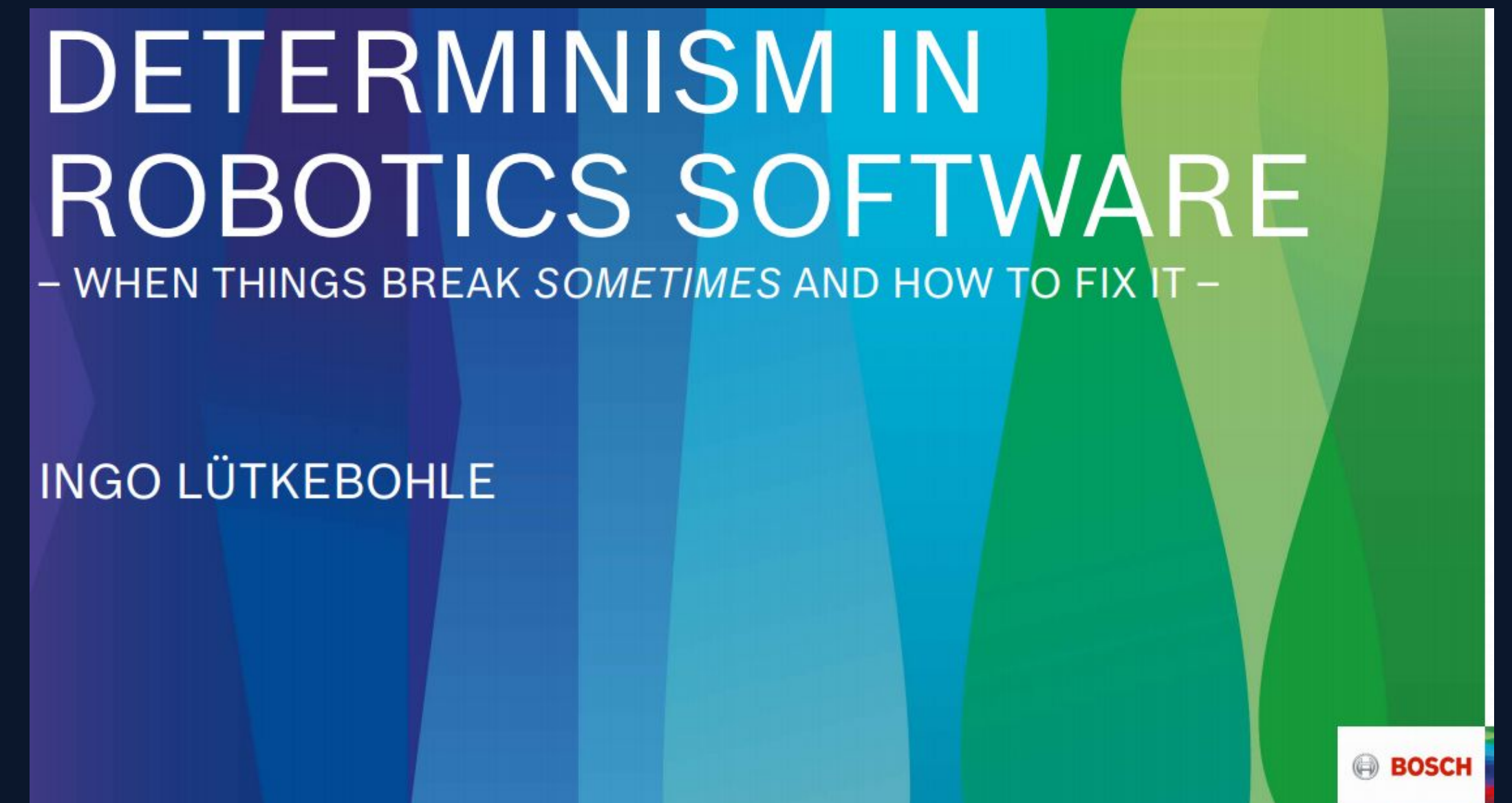


Open Source Robotics Foundation



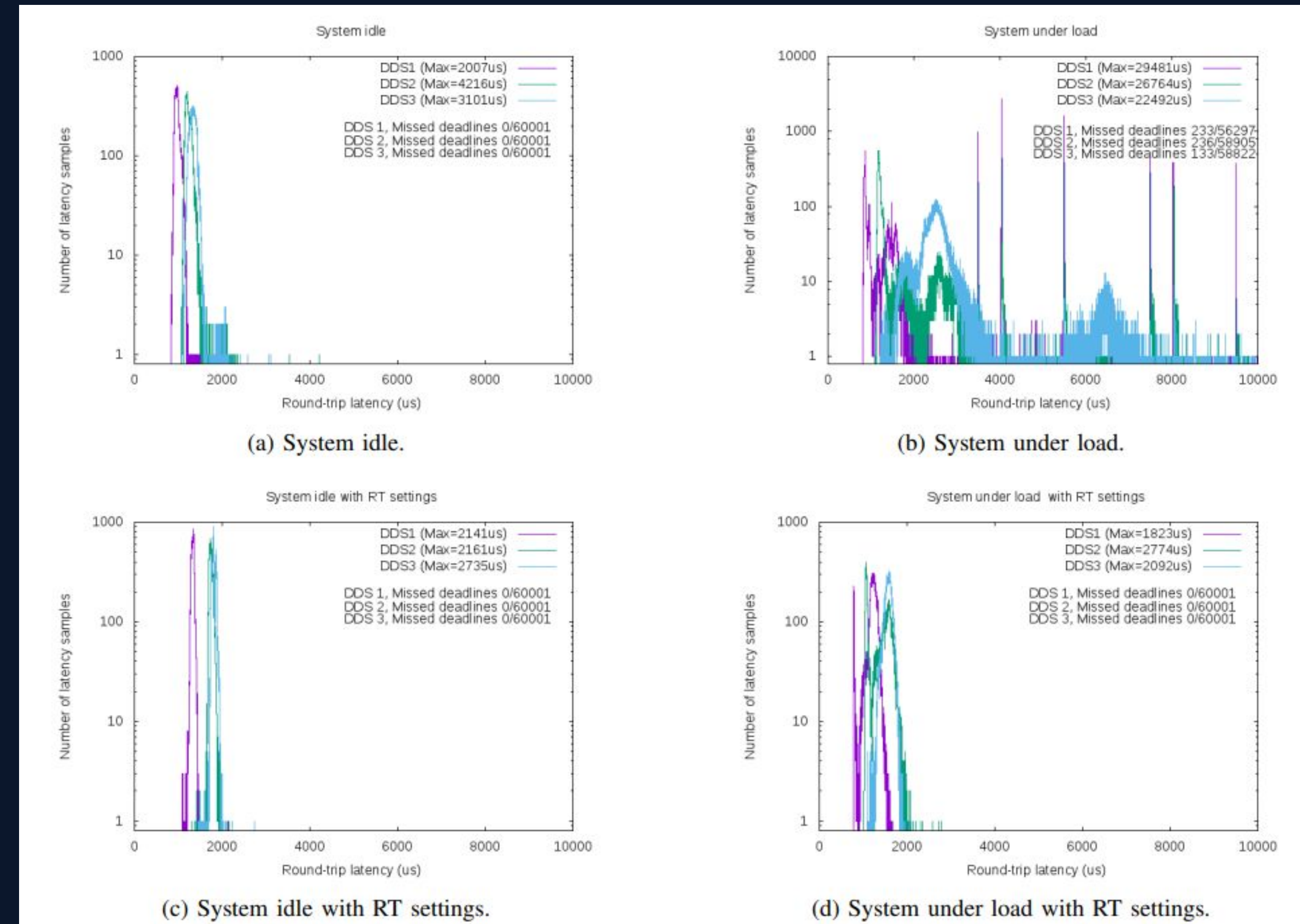
Recap: ROS and Real-time Publications

1. Definition of determinism:
 - A deterministic system will always produce the same output when starting conditions and inputs are the same
2. Dissected a use case for navigation in warehouses:
 - `move_base`
3. Dissected internals of roscpp pub/sub loop
4. Timestamping of sensor data and synchronization
5. Data aggregation policies
6. Sampling and sampling effects
7. Architectural choices when building a robotics system
 - data-triggered
 - time-triggered
 - queue sizes
8. Tracing:
 - LTTng



Recap: ROS and Real-time Publications

1. Time Synchronization in modular collaborative robots
2. Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications
3. Real-time Linux communications: an evaluation of the Linux communication stack for real-time robotic applications
4. Time-Sensitive Networking for robotics



Series of papers, Acutronic Robotics

Angle 3: Industrial Use Cases That Require Real-time Software



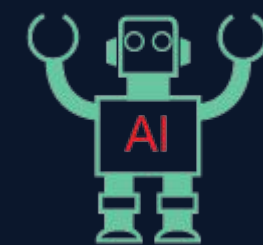
Autonomous cars



Autonomous Trucks



Autonomous agricultural robots



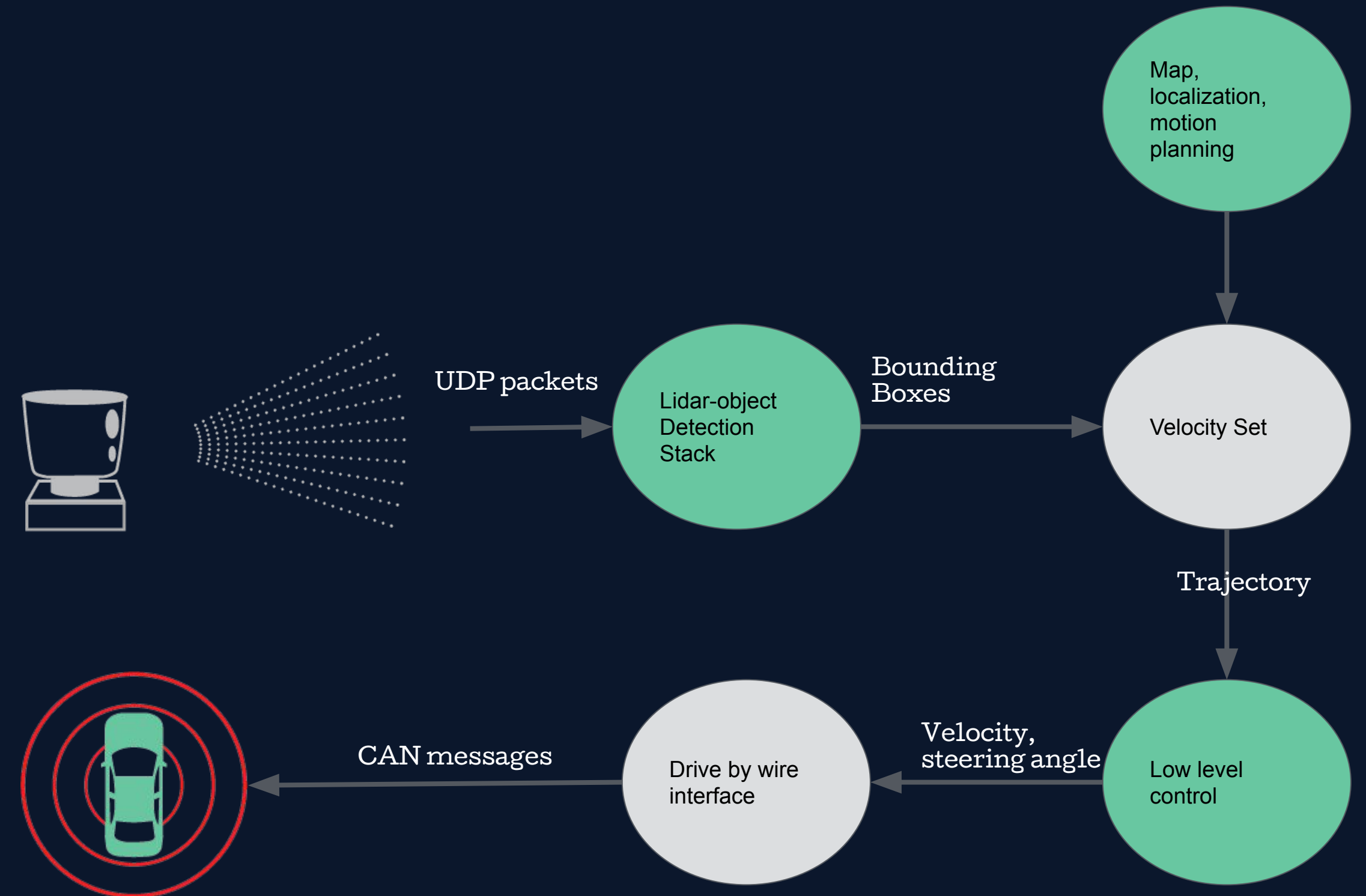
Indoor logistics platforms



Airborne drones

Use Case Autonomous Driving

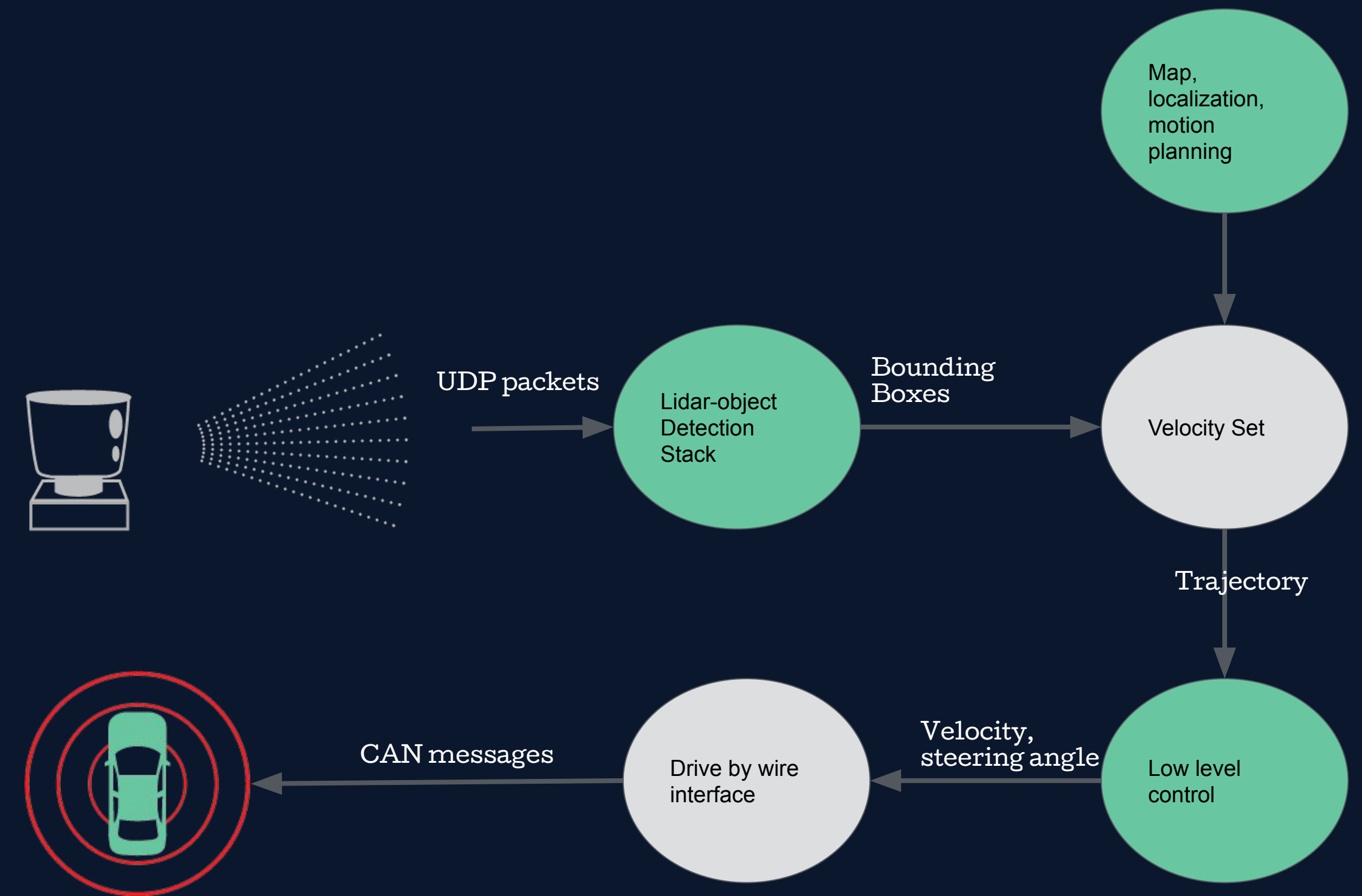
1. Autonomous driving car drives on the straight road and suddenly encounters an obstacle in the front
2. Simplified HW/SW architecture:



Use Case Autonomous Driving

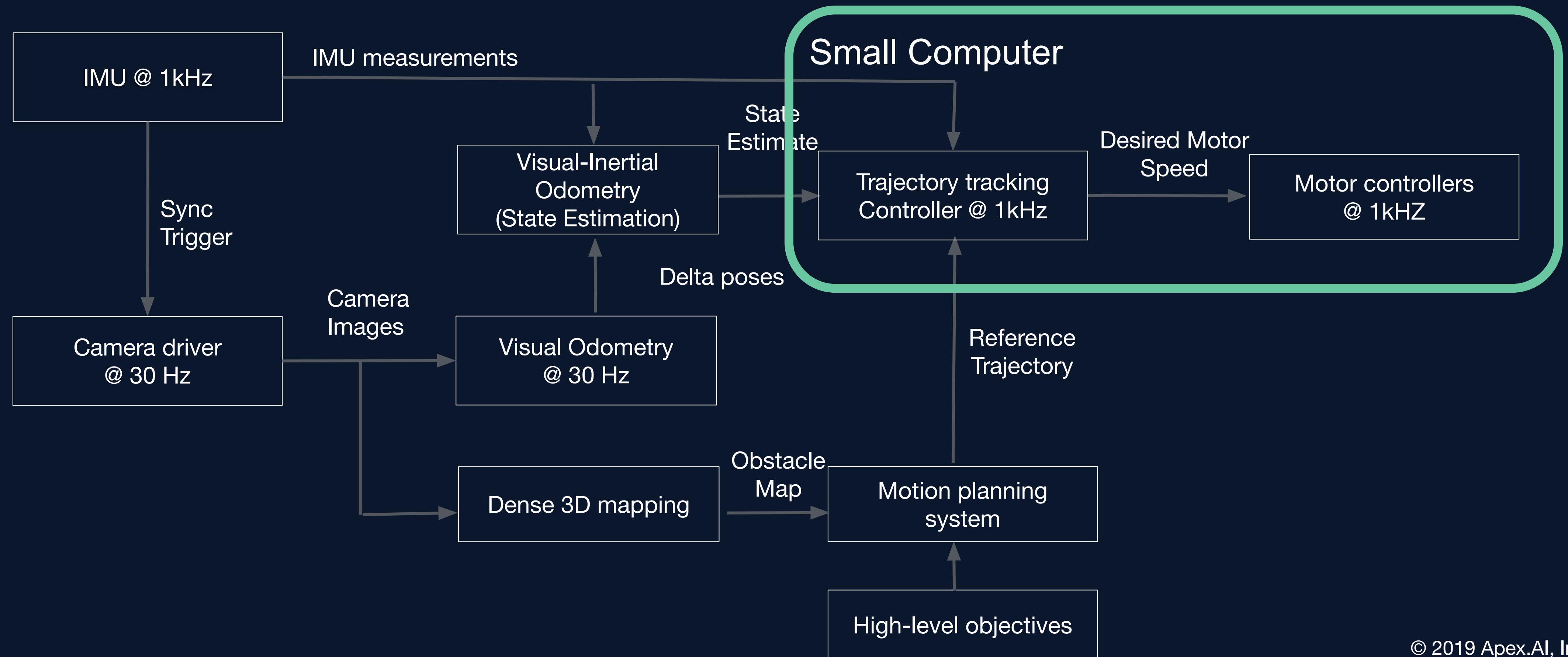
Real-time requirements

- An entire pipeline runs with 10Hz which is the output rate of a lidar sensor
- Car travels at 40mph (17.88m/s)
- Bounding boxes are coming in every 100ms
- Exactly at the point when we would start detecting a vehicle in front of us, we get a **dropped** or **mis-synchronized** frame
- The lidar sensor has a reliable detection range up to 40m and we can break with the maximum 6m/s² deceleration
- If we travel with 17.88m/s and break with 6m/s²: $t = (v_s - v_e) / a = 3s$ to stop
- In terms of distance this means $(d = 1/2 * a * t^2) = (d = 1/2 * 6m/s^2 * (3s)^2) = 27m$
- This adds 200ms to the stopping time $d = 1/2 * 6m/s^2 * (3s)^2 + 0.2s * 17.88m/s = 30.58m$
- Remember: we have **40m** to stop
- **7 dropped** frame means collision



Use Case Airborne Drones

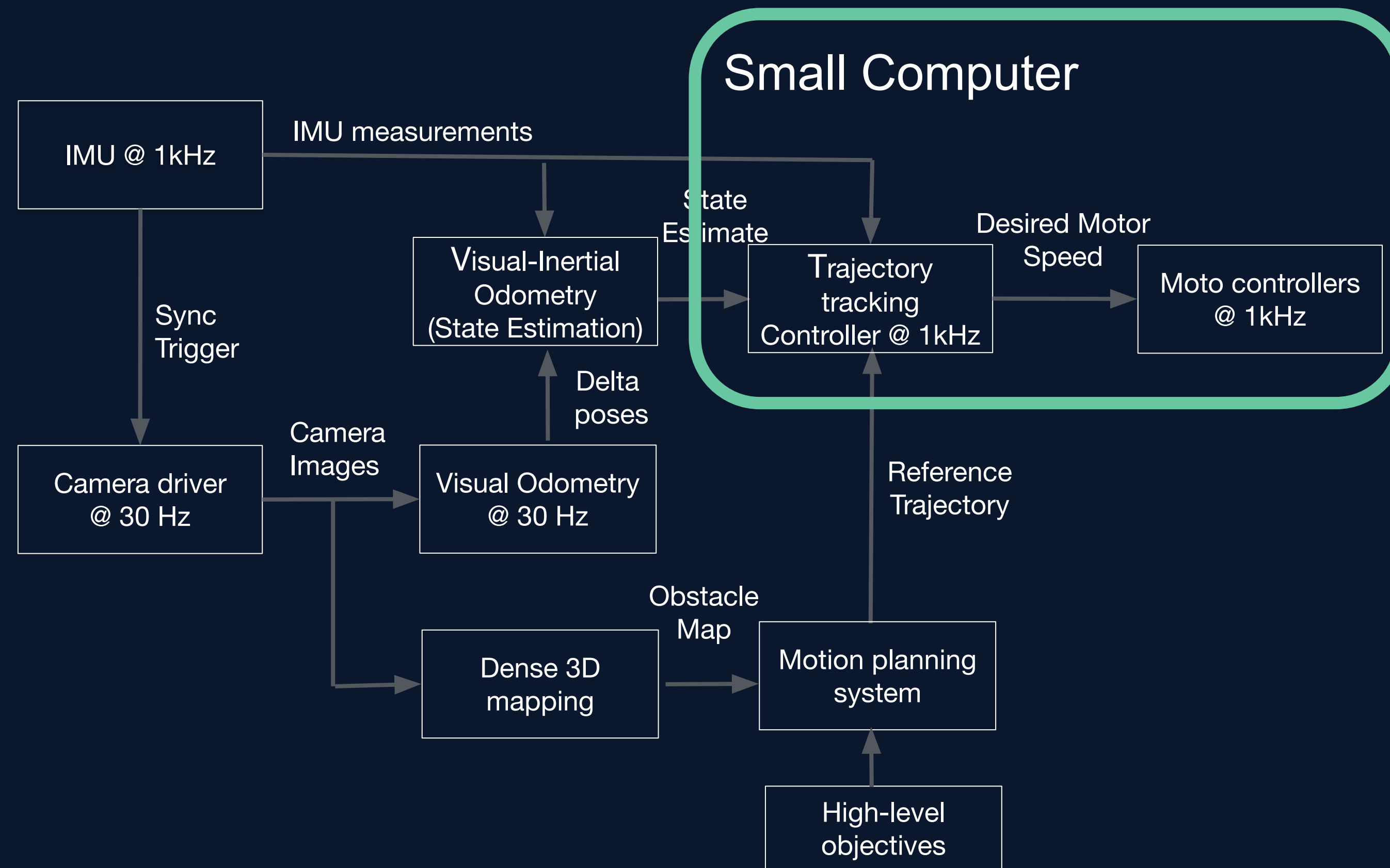
1. Drone with visual navigation and obstacle avoidance
2. Drone has to be capable to navigate and avoid obstacles in real time
3. Simplified HW/SW architecture:



Use Case Airborne Drones

Real-time requirements

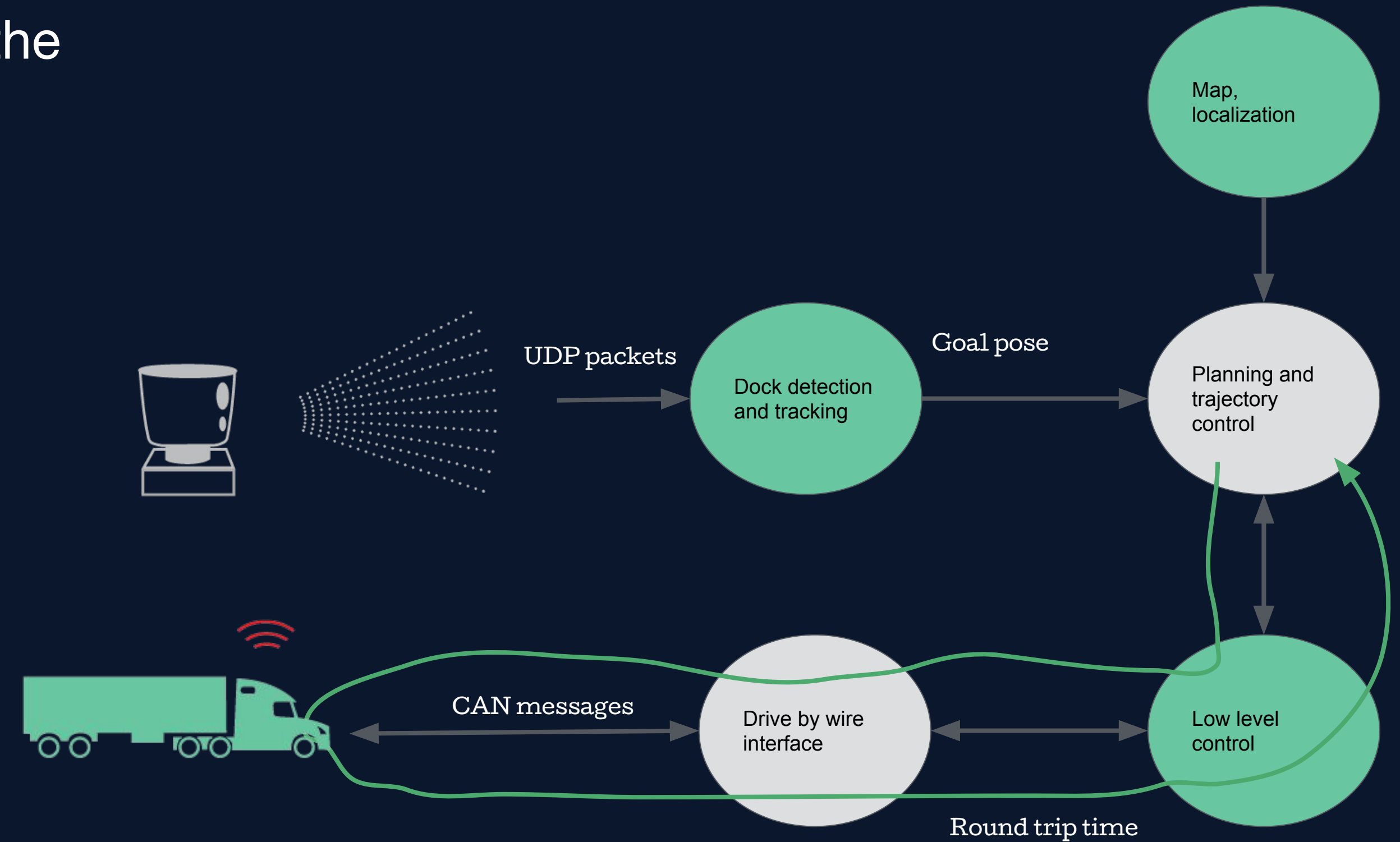
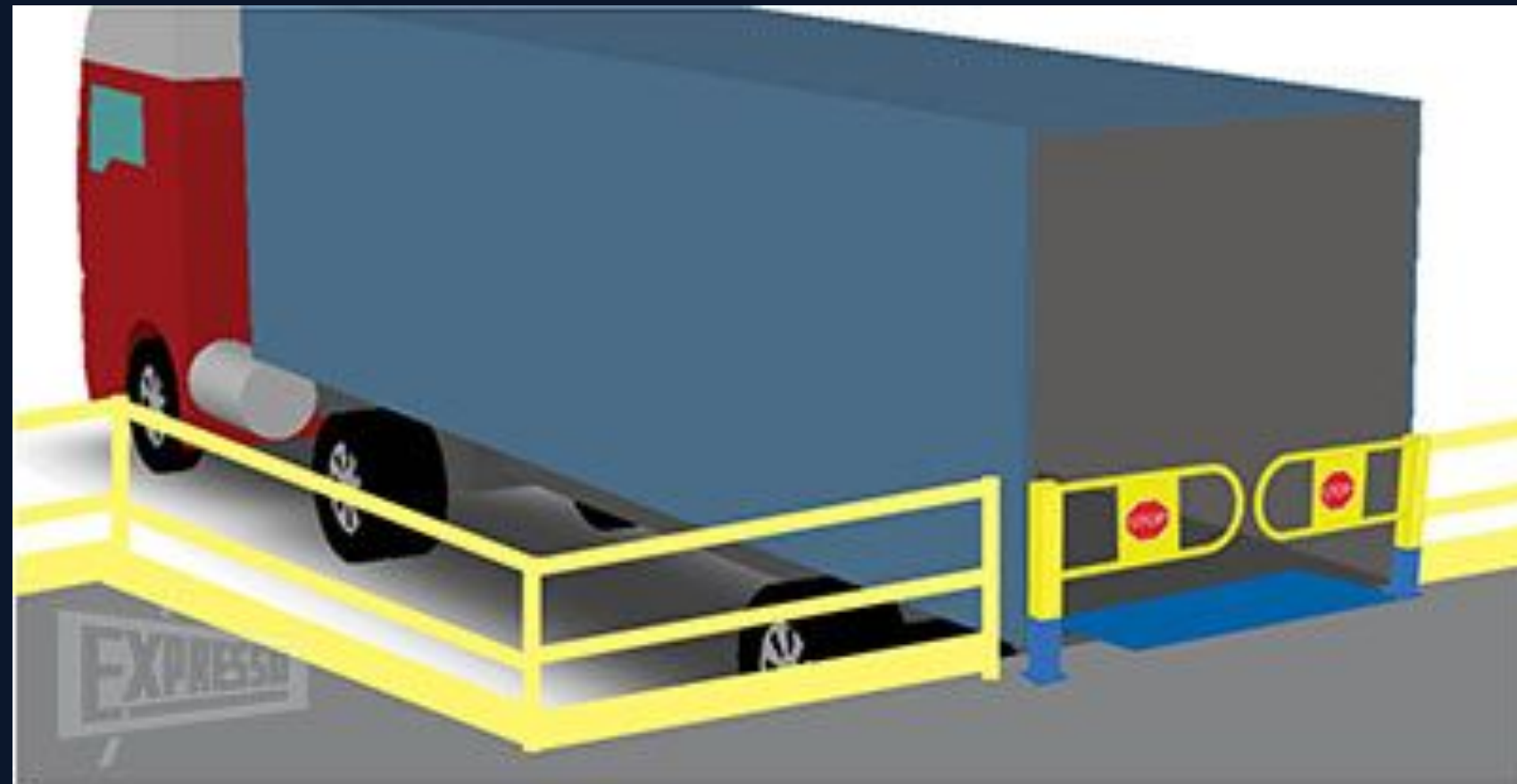
- IMU has to be real-time at **highest rate possible**
- **Time synchronized** camera images
- **Big-small** computer architecture
 - Hardware communication layer in between
- Running on the "big computer": camera driver, visual odometry, dense mapping, motion planning
- Running on uC: trajectory tracking, state estimation, motor control
- **Jitter** in the communication between big and small computer, jitter in process scheduling
- Camera images might come **too late** which can cause everything downstream in the pipeline to miss a deadline
- The motion planner could not recompute a new trajectory within a **deadline**



-
- Thanks to Teo Tomic

Use Case Autonomous Trucks

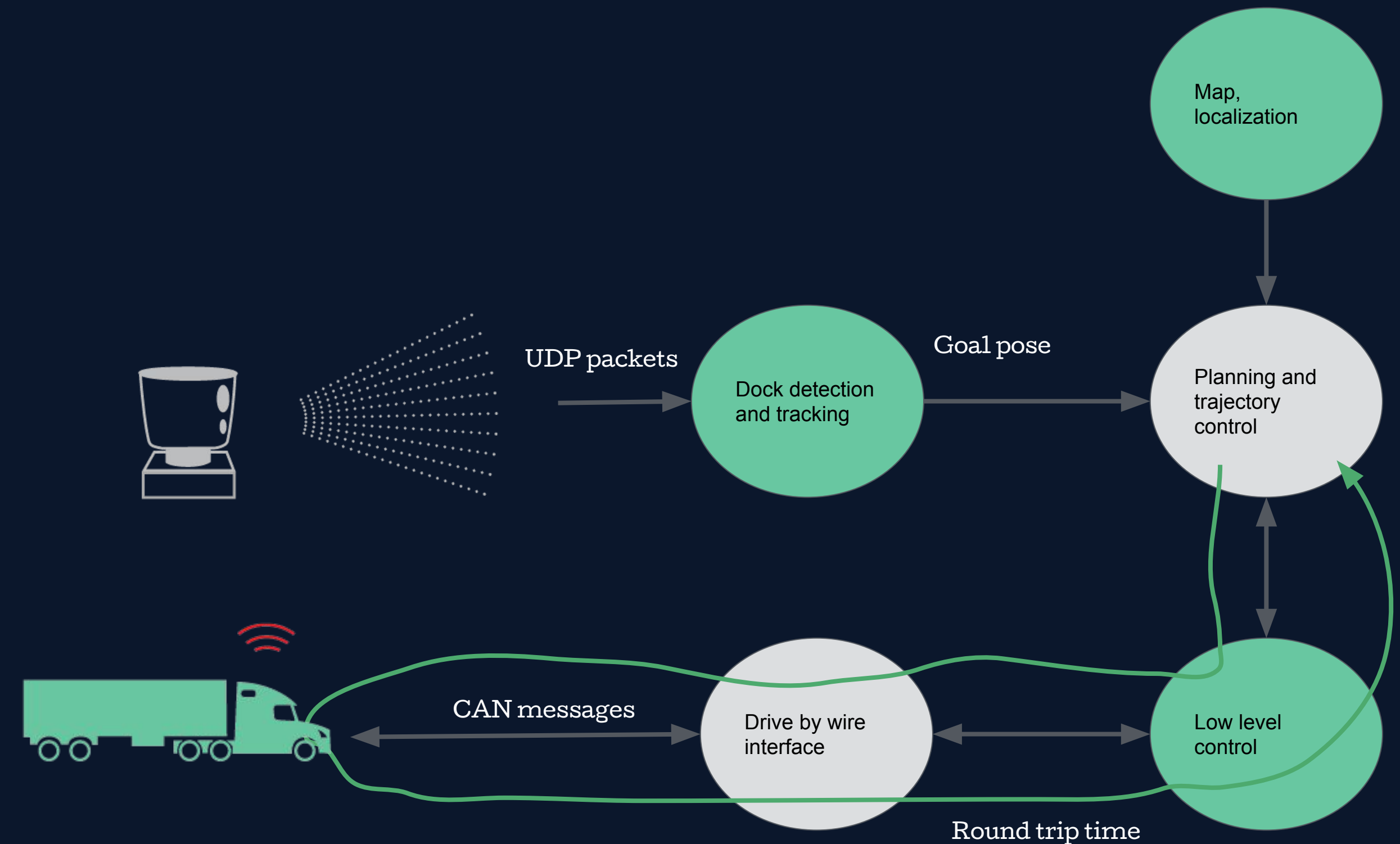
1. Autonomous truck > 25t has to reverse park into the loading dock with high accuracy < 9cm
2. Simplified HW/SW architecture:



Use Case Autonomous Trucks

Real-time requirements

- 100ms available for the maximum **round trip time**
- Round trip time to be **constant**
- High inertia of the truck and a high round trip time => predict the movement of the truck in the controller
- A low maximum round trip time and low **jitter** (variance) of this round trip is essential
- When too many (LiDAR) messages are **lost** in the high level to low level communication => emergency stop
- Problem: many control loops and **asynchronous data** communication



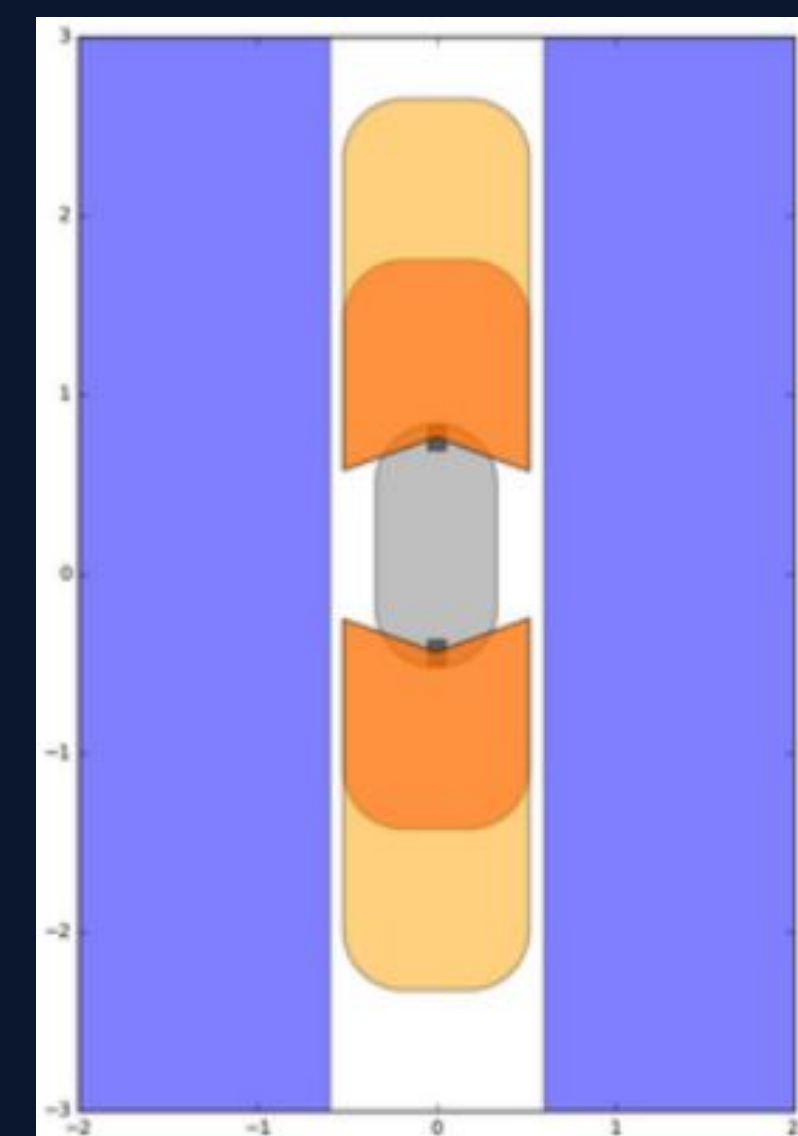
- Thanks to Tobias Augspurger

Use Case Logistics Robot

- Magazino's Toru robot has a sophisticated safe speed control when navigating between shelves in the warehouse
- LiDAR-based sensing
- When all fields are free => 1.5m/s
- Far field is blocked => 0.7m/s
- Near field is blocked => 0.3m/s

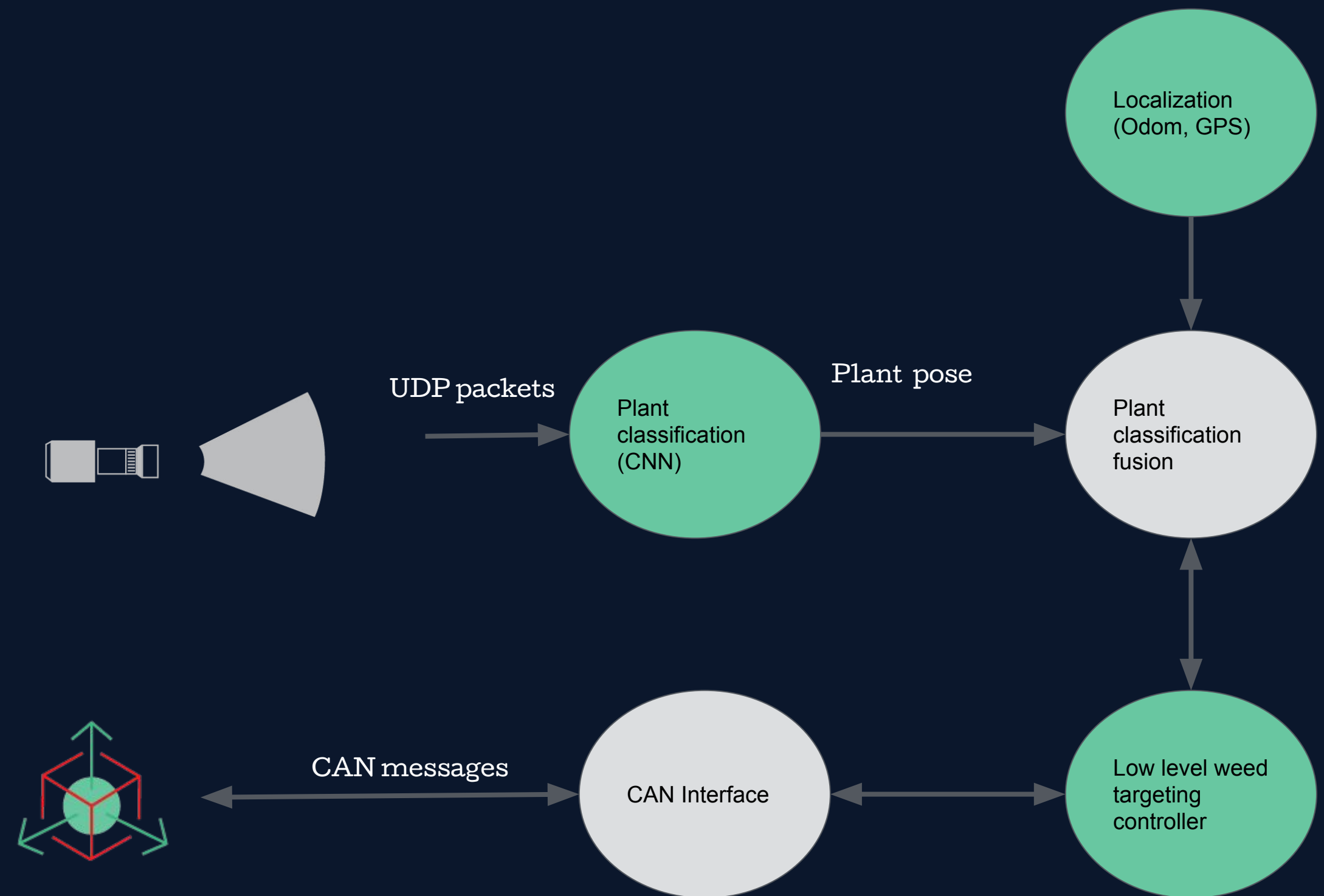


- Thanks to Kai Franke



Use Case Autonomous Weeding Robot

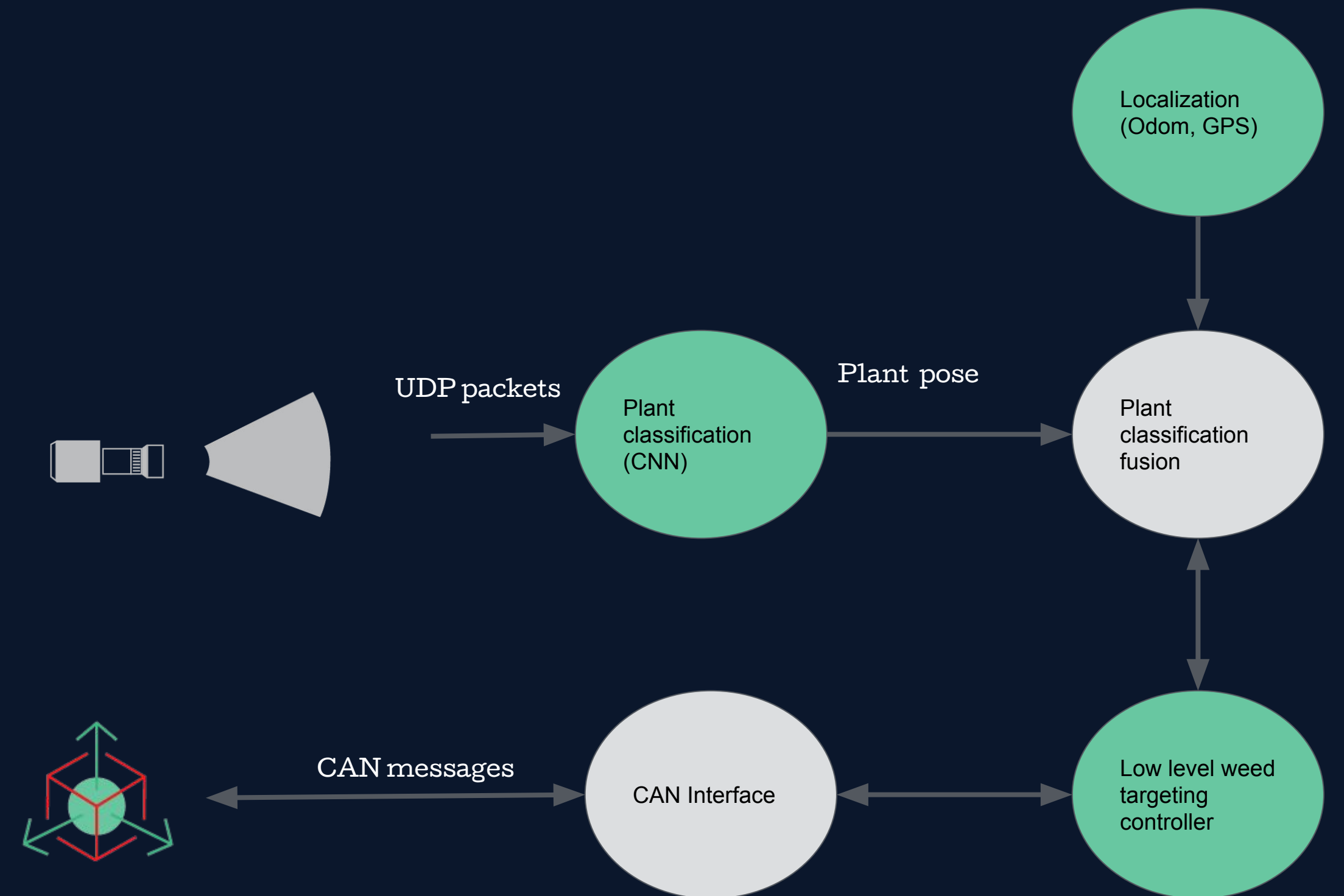
1. Autonomous weeding robot distinguishes plants with deep learning and mechanically removes weeds
2. Simplified HW/SW architecture:



Use Cases Autonomous Weeding Robot

Real-time requirements

- Entire pipeline runs at 10Hz which is the output rate of the camera
- Robot travels at 2,2mph (1m/s)
- Plant classification takes 100ms
- The weeding tool is 20cm from the camera.
- One frame **drop** is a 100ms delay
- In 200ms, the robot has travelled 20cm
- 20cm means that the robot can barely remove the weed
- Remember: the tool is located 20cm from the camera



ROS 2 Features for Real-time

Summary of pain points from above

- Lost or delayed messages
- Performance
- Scheduling
 - Jitter
- Inter-computer communication
- Patterns for event-based communication ([proactor vs reactor](#))
- Safety mechanisms realized with HW
- ROS 2 on HW for accelerate compute (GPU, FPGA)
- Timestamping of data
- Safety and certification
- Porting from ROS 1 => ROS 2

- Several companies are running ROS 2 on real-time HW (e.g Renesas R-Car H3)
- ROS 2 support for [QNX](#) and [VxWorks](#)
- Memory allocations and management
 - OSRF found and fixed memory allocations in [rmw, rcl, rclcpp](#)
 - “ROS2 Real-Time Behavior: Static Memory Allocation” => talk on Friday
- (Soft) real-time DDSes ([CoreDX](#), [Cyclone](#), FastRTPS 1.9.x)
- loaned messages (first iteration)
 - To be able to loan and give back memory to the middleware
- [New intraprocess mechanism](#)
- rclcpp::Lifecycle (ROS 2 node as an FSM)
- [Zero-copy transport](#)
 - See “A true zero copy RMW implementation for ROS2” talk on Friday
- Real-time [support package](#) (as a demo)

ROS 2 Features for Real-time

Summary of pain points from above

- Lost or delayed messages
- Performance
- Scheduling
 - Jitter
- Inter-computer communication
- Patterns for event-based communication ([proactor vs reactor](#))
- Safety mechanisms realized with HW
- ROS 2 on HW for accelerate compute (GPU, FPGA)
- Timestamping of data
- Safety and certification
- Porting from ROS 1 => ROS 2

- Porting articles:
 - a. [Chris Ho](#) blog post porting velodyne driver from ROS => ROS 2
 - b. “Migrating to ROS 2: Advice from Rover Robotics”, talk on Thursday
- Communication
 - a. “Quality of Service Policies for ROS2 Communications”, talk on Friday

ROS 2 Real-time WG

Members

- Bosch
- Windriver
- Intel
- AdLink
- Carlos
- Nobleo
- OSRF
- Amazon
- Ubiquity Robotics
- eSol
- TierIV
- Sony
- Victor

Join us:

- <https://index.ros.org/doc/ros2/Governance/> (See Real-time)
- Past meetings:
<https://discourse.ros.org/search?expanded=true&q=ROS2%20Real-time%20Working%20Group%20%23ng-ros%20in%3Atitle>
- Meetings every 2 weeks

Topics

- Real-time and static executor
- Single process, real-time rmw
- Static memory audit and improvements
- Tools for static and dynamic code analysis and tracing
- Performance testing platform

- **Planned for later:**
 - Fault isolation
 - Safety and Certification
 - Wait-sets
 - Real-time C++
 - Reference architectures for selected use cases


Modern C++ for Safety Critical Products

Apex.AI[®] Home Products Careers News ROSCon'19

Andreas Pasternak · Sep 28 · 1 min read

Safe Software for Autonomous Mobility with Modern C++, Presented at CppCon 2019

Updated: Sep 30



You are a C++ software engineer or a domain expert in robotics, and your boss asks you to make the software system you are developing "safe". How will you go about doing this? How can you build a functionally safe, reliable, and certifiable software system using modern C++?

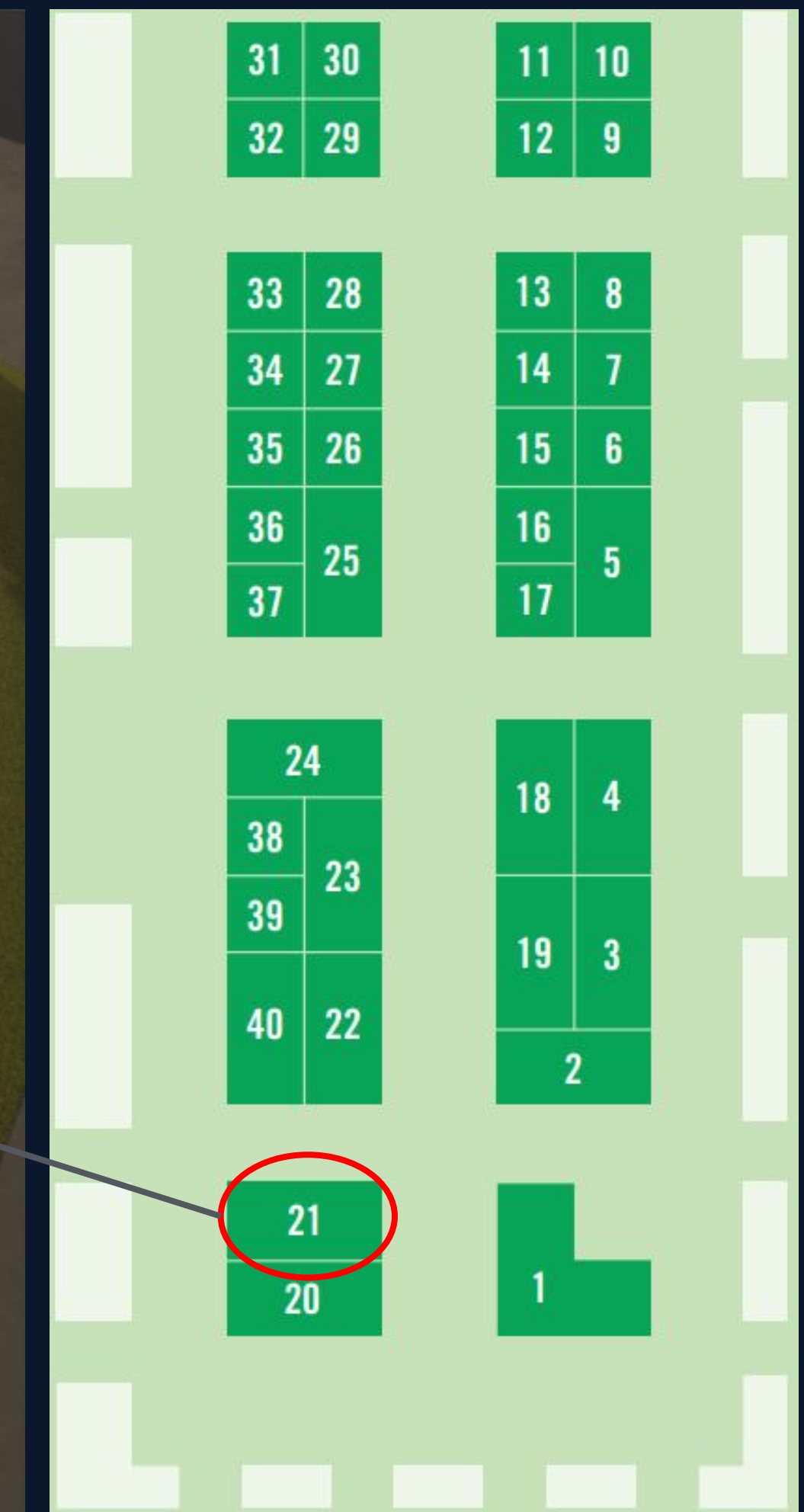
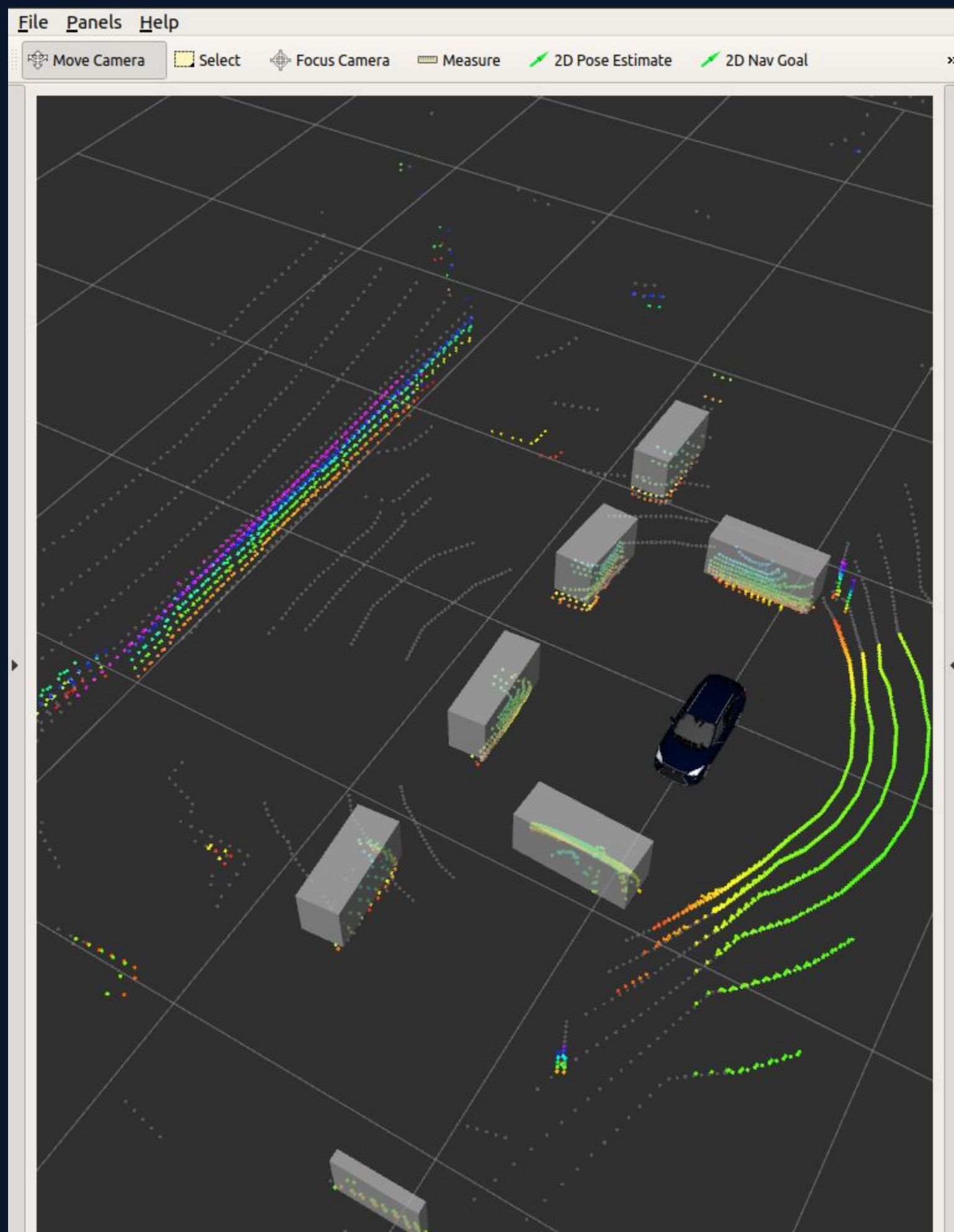
I just returned from [CppCon 2019](#) in Aurora, Colorado where I talked about using modern C++ to build reliable and real-time software for use in safety-critical autonomous mobility applications. I illustrated the challenges and solutions using examples such as avoiding memory allocations during exception handling, how to use memory pools for standard containers in a way which prevents memory fragmentation, additions [Apex.AI](#) made to the standard thread library to make it suitable for real-time applications and how to do failure injection to achieve the coverage required for the highest safety levels.

If you are interested, you can find the [presentation here](#). Feel free to reach out to us via info@apex.ai with questions about this presentation or to learn more about how we use modern C++ to build [Apex.OS](#) and autonomous driving application software.

Topics

- C++ compiler and standard library
- Memory static standard containers and exceptions
- Threading and related architecture
- Failure injection

Real-time LiDAR Object Detection Application at Apex.AI Booth



Key features:

1. Apex.AI real-time ROS 2
2. Renesas R-Car H3 ECU
3. QNX, Linux with RT PREEMPT

Conclusions

1. Real-time ROS 2 depends on layers below (HW, RTOS, DDS) and layers above (Application). Use tools to help you with validation. Real-time affects safety requirements (and other way around).
2. Come join us in the ROS 2 Real-time Working Group to work on this challenge together:
<https://index.ros.org/doc/ros2/Governance/> (See Real-time)
3. Contact: dejan@apex.ai

Backup

1. Backup

What is needed for safe modern C++ based product

High level requirements

1. Qualified Toolchain
- 2. Memory static operation**
3. Real Time
4. Testability

This talk will exercise the above requirements on

1. C++ compiler and standard library
- 2. Standard containers** and exceptions
3. Threading and related architecture
4. Failure injection

Requirements and Challenges

Requirements

- Application must be memory static at runtime
- No memory fragmentation should occur
- One should be able to reason about resource usage and resource limits of containers to ensure that those are sufficient during the whole runtime of an application

Challenges

- Standard C++ containers allocate memory and allocate this memory using different strategies
- Stack memory is very limited, running out of stack is hard to recover from
- Memory pools can fragment, depending on how they are used
- Traceability of resource usage must be given to be able to prove a system does not run out of resources which would introduce a safety hazard

Memory pools and allocators are only one piece of the solution.

Solution

Strings (`std::string`)

Compile time fixed size string with stack storage in two flavors, one which silently truncates and one which throws on overflow

- No memory allocation
- Small size makes it well suitable to store on the stack
- Simple to use

Vectors (`std::vector`)

Runtime fixed size vector which allocates from heap on construction

- Memory for each vector is allocated before runtime
- Satisfies continuous memory guarantees
- Harder to use as not constructable or copyable during runtime

Node based containers (`std::map`, `std::set` etc.)

Memory-pool / allocator framework (for example <https://github.com/foonathan/memory>)

- Pools are allocated before runtime
- One pool per type prevents memory fragmentation
- Pools can be arbitrarily granular ensuring resource isolation and makes proving that the application does not run out of resources easier
- Unordered containers do not work with fixed size pools

<https://bduvenhage.me/performance/2019/04/22/size-of-hash-table.html>

Different container types require different solutions

What is needed for safe modern C++ based product

High level requirements

1. Qualified Toolchain
- 2. Memory static operation**
3. Real Time
4. Testability

This talk will exercise the above requirements on

1. C++ compiler and standard library
2. Standard containers and **exceptions**
3. Threading and related architecture
4. Failure injection

Requirements and Challenges

Requirements

- Application must be memory static at runtime, therefore exceptions need to be memory static
- One should be able to reason about resource usage and resource limits of exception management to ensure that those are sufficient during the whole runtime of an application
- Runtime of routines need to be bounded and well understood

Challenges (might not be the same for all compilers)

- Throwing an exception allocates memory on the compiler level
- Standard exceptions allocate memory to store the error strings
- Exception handling adds hard to track and hard to analyze execution branches to an application. This complicates runtime and test coverage analysis.

Exceptions allocate memory!

Making exceptions memory static

Modify how compiler allocates memory for exception

Example on how to do this for GCC: [ApexAI Static Exception](#)

```
extern "C" void * __cxa_allocate_exception(size_t
thrown_size);
extern "C" void __cxa_free_exception(void *thrown_object);
extern "C" __cxxabiv1::__cxa_dependent_exception *
__cxa_allocate_dependent_exception();
extern "C" void __cxa_free_dependent_exception
(__cxxabiv1::__cxa_dependent_exception *
dependent_exception);
```

Modify the standard C++ library to use a string storage which does not allocate

We're still working on this

Different compilers
required different
solutions

Open Challenges

4.27 Fault Handling

AV Rule 208

C++ exceptions **shall not** be used (i.e. *throw*, *catch* and *try* shall not be used.)

Rationale: Tool support is not adequate at this time.

JSF Coding Standard (2005)

- Exception handling creates additional branches which are hard to cover with tests and 100% branch coverage is mandatory for the highest safety levels

	Hit	Total	Coverage
Lines:	29	29	100.0 %
Functions:	9	9	100.0 %
Branches:	17	26	65.4 %

```
    :  
    :  
    :    /// Returns the timestamp to consider for the type.  
5091 :    static std::chrono::nanoseconds timestamp(const loaned_message_t & msg)  
    :    {  
[ + - ]:    10182 :    return std::chrono::seconds(msg.data().header.stamp.sec) +  
[ + - ]:    15273 :    std::chrono::nanoseconds(msg.data().header.stamp.nanosec);  
    :  
    :    }
```

Tool support for exceptions is still a challenge

What is needed for safe modern C++ based product

High level requirements

1. Qualified Toolchain
2. Memory static operation
- 3. Real Time**
4. Testability

This talk will exercise the above requirements on

1. C++ compiler and standard library
2. Standard containers and exceptions
- 3. Threading and related architecture**
4. Failure injection

Requirements and Challenges

Requirements

- Tasks need to have a deterministic runtime
- Execution of a task needs to be interruptible by a higher priority task
- Lower priority task must not block higher priority ones

Challenges

- Standard C++ threading library only provides very basic control over thread priorities, priority inheritance, CPU pinning etc.
- Execution in C++ is not preemptible (yet), making it hard to write any kind of real-time capable executor
- It is very difficult to ensure multithreaded code is correct because code execution is interleaved in a non-deterministic way
- If a data race occurs, the behavior of the program is undefined

Solutions

Custom standard threading library

A custom modern C++ threading library abstracts away the POSIX interface, which makes creating multithreaded application easier and hides implementation details

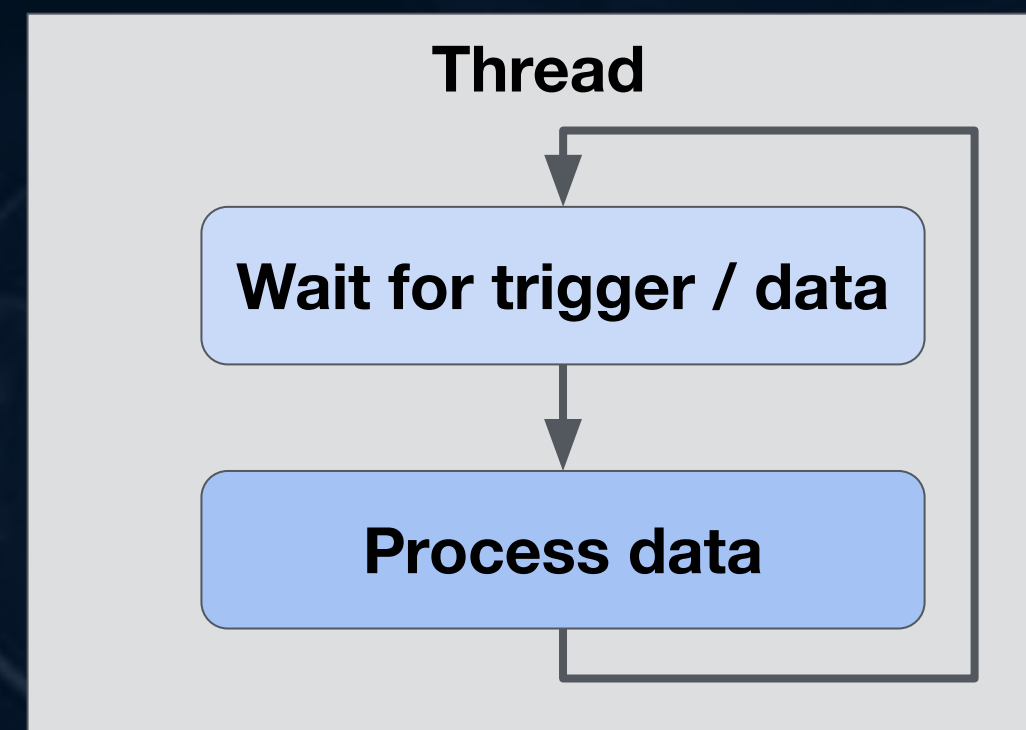
- Threads are created during initialization, but put on hold until started for runtime
- Thread priority, CPU pinning, etc. are configurable
- Mutexes support priority inheritance

Rely on the OS Scheduler

- Well understood
- Good tool support
- Disadvantage: More context switches than with an executor based architecture

Use advanced tools to ensure correctness

- Thread Sanitizer
- Clang Thread Safety Analysis
- Hellgrind



Library support,
Software architecture
and tools need to be
in place