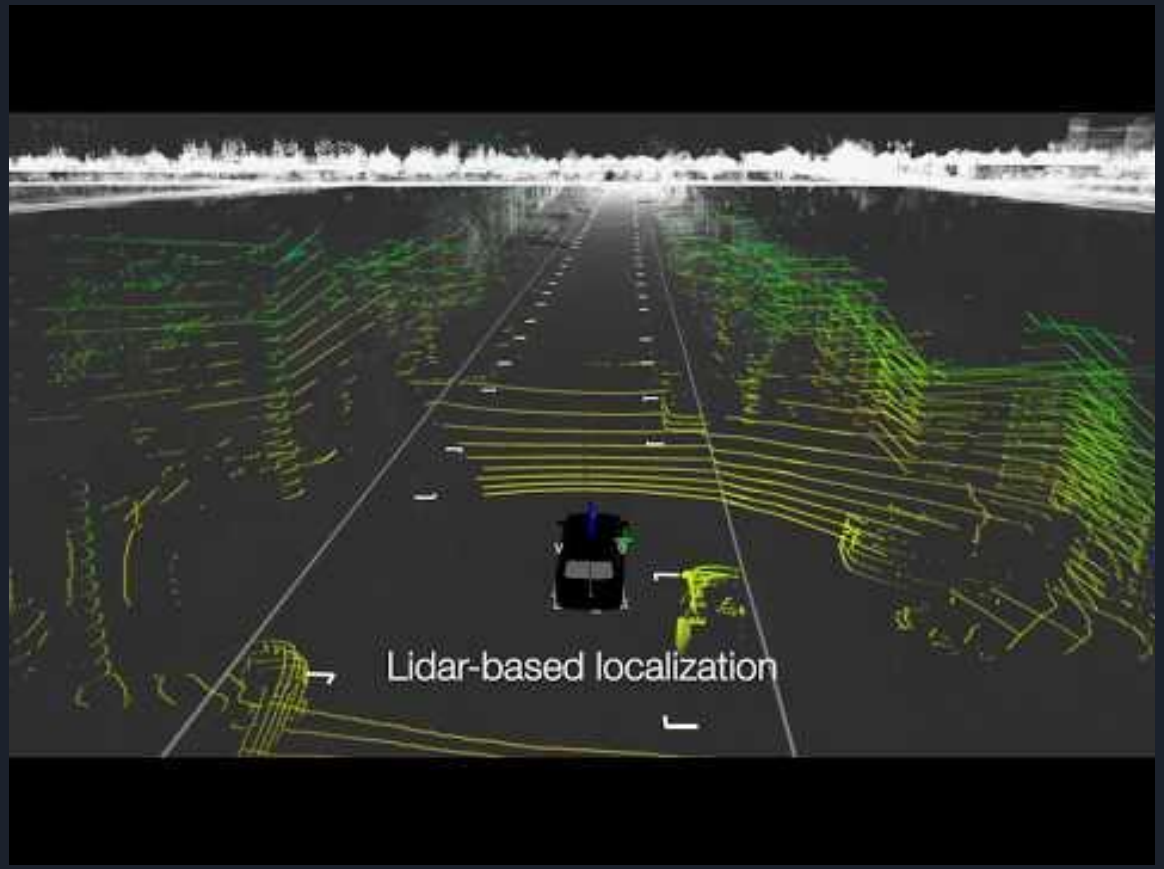
Migrating a large ROS 1 codebase to ROS 2

A community perspective

Geoffrey Biggs (Tier IV/The Autoware Foundation)
Esteve Fernandez (Apex.AI/The Autoware Foundation)

Autoware



Today’s Pop Quiz

How do you move a large existing community of users from ROS 1 to ROS 2, while rebuilding your software from scratch to take advantage of the new capabilities of ROS 2, without causing the community to collapse?

Autoware.AI

- Open-source software for self-driving vehicles
- Based on ROS 1
- Great for prototyping:
  🙂 → 😀 → 😃 → 😄 →🥳
- Not so great for building products

Limits of ROS 1-Based Software

- Autoware.AI's design and implementation are constrained by ROS 1
- Extremely difficult to certify
  - ROS 1 is not certifiable without significant (many years and people) effort
  - (So is Autoware.AI, which makes it worse)
  - Determinism, memory safety, etc. are not possible
- Less than 6 years of life left
  - Last release in mid-2020
  - End-of-life in 2025
  - We don't want to maintain such a large open source software project ourselves - lose the benefits of the herd

Limits of ROS 1-Based Software

- Simple launch system
  - Hard to control startup order and timing of nodes
  - Node wait-for loops
- Custom protocol missing many features of modern communications middlewares
  - Security!
  - Real-time
  - Implementation suitable for embedded systems
  - Usability on lossy networks
- Nodes running out of lock-step
- Cannot choose when to compose nodes

Move to ROS 2 and get...

- Managed launching
- Node lifecycles
- DDS
  - QoS paradise
  - DDS-Security
- Composable nodes
- Consistent API
- Zero copy (in ROS 2 Eloquent)
- Many, many other features

Move to ROS 2 and get...

New ways to architect your system to achieve robustness, reliability, and safety, as well as efficiency

Porting to ROS 2: The Options

1. sed -i ‘s/ros/rclcpp/g’ *.h *.cpp
    - Gets you to ROS 2 quickly
    - Still requires some work meeting new APIs so isn’t a five-minute job
    - Behaviour is not guaranteed to be the same between ROS 1 and ROS 2
    - → Not safe without comprehensive tests
    - Miss most of the great new features of ROS 2

Porting to ROS 2: The Options (con't)

2. Start again and re-design
    - Re-think from the architecture up to take advantage of new capabilities of ROS 2
    - More work but better long-term results (so long as you finish it at some point
    - Can fix other deep problems with your code base at the same time
    - Your community will be … *unhappy* with you

Autoware: The Next Generation

- Autoware.Auto, the next generation of Autoware
- Aims to fix all the problems with Autoware.AI
  - High test coverage
  - Comprehensive and *readable* documentation
  - Modular code base to improve CI times, reusability and adaptability
  - Flexible and easy-to-extend architecture
  - Deterministic execution
- Provides a flexible framework for self-driving research and application development
- As close to production-ready as possible for an open-source project
- Better use of and contributions to upstream
- ROS 2-based

The plan with a capital P

- Throw out Autoware.AI (the ROS 1 version of Autoware)
- Design a new Autoware
  - ROS 2-based
  - Deterministic, real-time, memory safe, and all that other good stuff that we want for safety
  - Great new architecture that makes Autoware Even Better™
- Implement this new Autoware with software engineering best-practices
- The result: Everyone ~~loves~~ hates us for forcing them to move to a new, incomplete system!

Wait, hold on...

Porting Can Hurt A Community

- The project needs:
  - Re-design to take advantage of new capabilities of ROS 2
  - Probably a lot of cleaning up and breaking APIs
- But the community members need:
  - A gradual transition
  - A clear path to adoption of the ported software
  - Working software now

Porting Can Hurt A Community (con’t)

- Porting must be carefully managed if you want to bring the majority of your community with you

New plan!

- Start again, but don’t throw away the existing code base all at once
- Port the algorithms where appropriate, but
  - Redesign the architecture
  - Re-do the implementation to be memory-safe, deterministic, etc.
- Keep the community happy by building the Autoware of Theseus

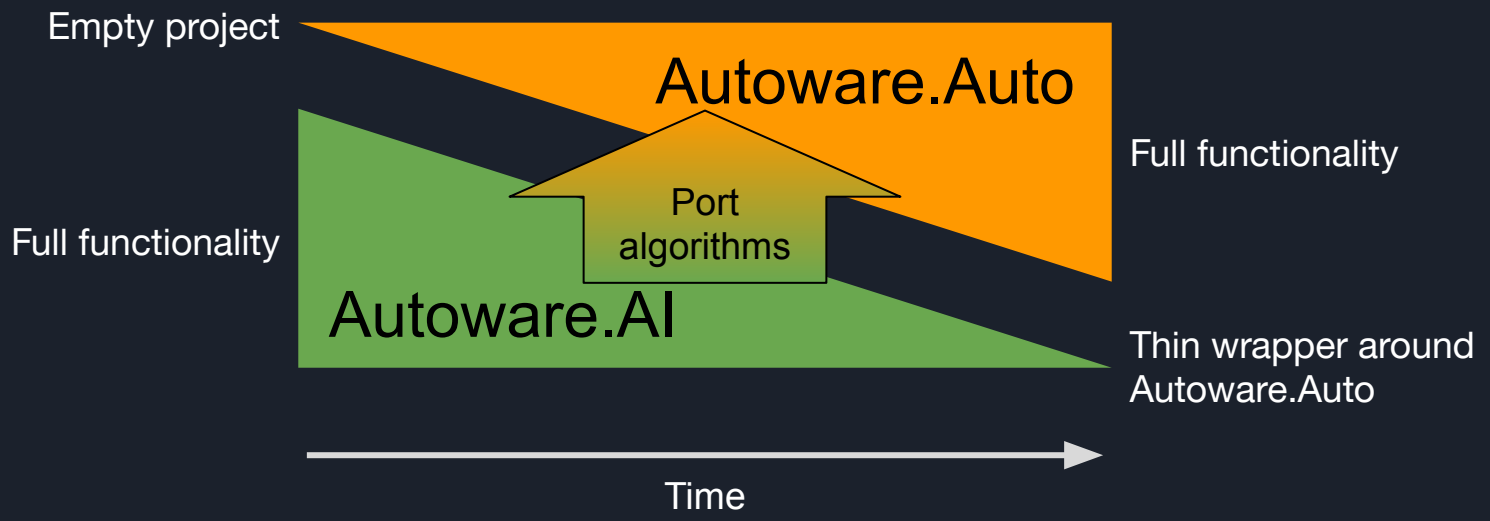
The Ship of Theseus

- Thought experiment that the ancient Greeks liked to play with
  - https://en.wikipedia.org/wiki/Ship_of_Theseus

1. Theseus was a hero
2. He had a ship, and did great things with it, so it was kept as a museum piece
3. Over time, bits rot and are replaced
4. When all the bits have been replaced, is it still the same ship that Theseus used?

- See also: Your grandfather's axe

Autoware.Auto: Theseus's Self-Driving Platform

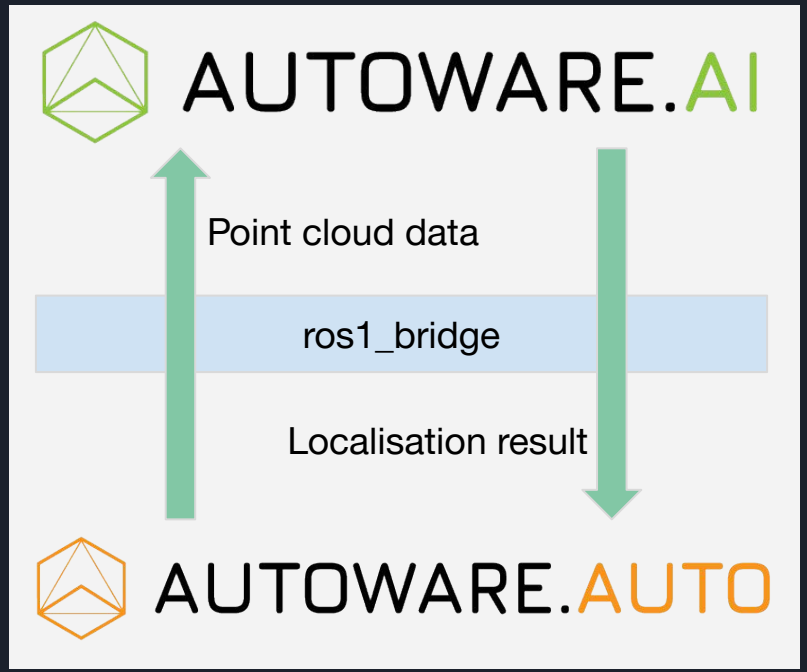


Autoware.Auto: Theseus' Self-Driving Platform

- Users of Autoware.AI can continue to use it
- Autoware.AI provides the missing pieces of Autoware.Auto
  - Via ROS 2 launch files and the ros1_bridge
  - Users of Autoware.Auto can do full self-driving
- Over time, parts of Autoware.AI are removed and that functionality used from Autoware.Auto
  - Via ROS 1 launch files and the ros1_bridge
- Replace Autoware.AI little by little to minimize disruption by users
- If things go perfectly to plan (), Autoware.AI users should not notice that they are actually using Autoware.Auto

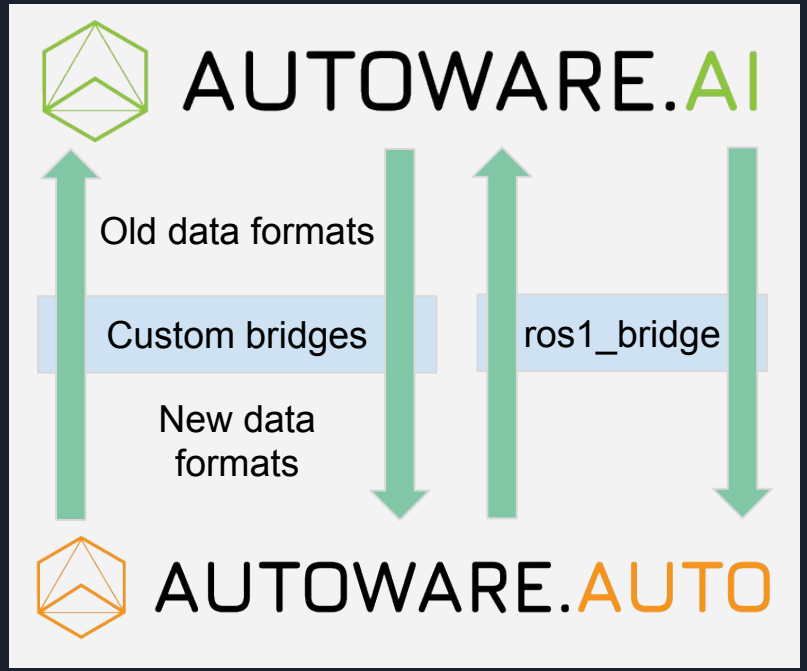
Bridging Two Autowares

- Using the ros1_bridge to join Autoware.AI and Autoware.Auto
- Bridge translates any topics and services where the data types have not been changed
- ros1_bridge does not handle actions, but Autoware.AI does not use actions



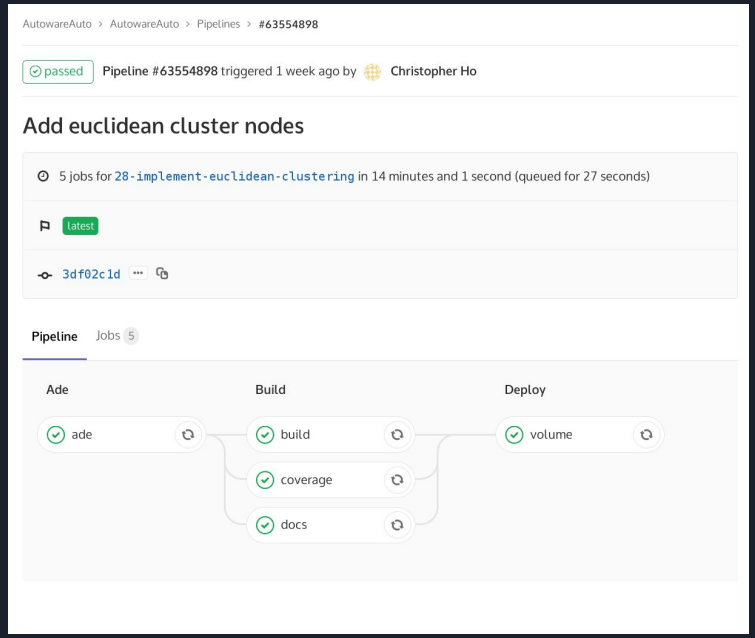
Bridging Two Autowares

- Where the data types have changed, a customised bridge is sometimes necessary
- Prefer to create custom bridges rather than restrict the new architecture



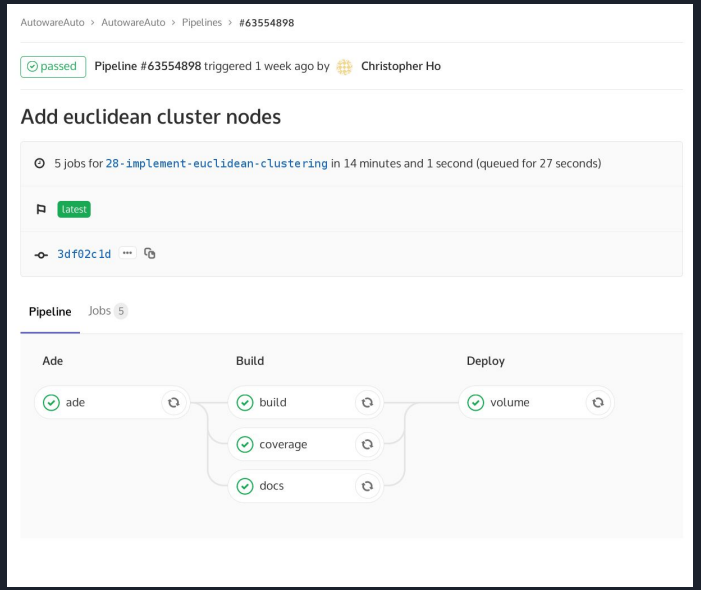
Improving Code Quality of Autoware

- Applying good software engineering practices
  - Rationale for every change recorded
  - High test coverage from the start
  - Use of CI not just for tests but for various design and code quality analyses



Improving Code Quality of Autoware (con't)

- Requirements on PRs are more strict
  - Must pass CI
  - Must have sufficient test coverage
  - Must have gone through a design review
  - Must meet coding style standards
  - Must not have unacceptable static analyser violations
  - Must add or update documentation (documentation reviews included)
- Comprehensive integration tests using `launch_testing`



Helping The Community Step Up

- Even if higher quality is a goal, the community may not be ready
  - “That contribution guide is really long…”
- Help your community learn how to meet your new, higher standards
- Mentor contributors!
  - You cannot throw out detailed contribution requirements without giving guidance
  - Encourage new contributors, don’t throw them to the wolves
- Contributors should know what they can expect of the process as well as what the process expects of them

Helping The Community Step Up (con't)

- Create detailed contribution guides
  - Describe the code review process in detail
  - Provide a detailed PR review guide for reviewers
- Provide tutorials on the software engineering practices you want used, e.g. how to
  - Design and implement for testability
  - Write effective tests
  - Check test coverage
  - Check for memory leaks
  - Write for and test deterministic execution

Helping The Community Step Up (con’t)

- Use automated tools to assist contributors
  - Make CI available to everyone so anyone can see their PR get checked
  - Provide automated code linters, static analysers, etc. so complying with rules is as simple as possible

Helping The Community Step Up (con’t)

Mentor Contributors!

New contributors especially will be discouraged by strict requirements. Walk them through the process and provide frequent encouragement!

End Goal

Autoware.Auto

- Full-functionality self-driving stack in ROS 2
- Near-production quality
- Strict quality control policies
  - Design reviews
  - Code quality maintenance
  - Safety considerations
- Well-documented and mentored contribution process

Autoware.AI

- Thin wrapper around Autoware.Auto using the bridge
  - Mostly launch files
- For those who won't or cannot move to ROS 2
- Limited functionality
- Maintained by its users

Autoware.Sandbox

- Box to hold proposed extensions and modifications to Autoware.Auto
  - For researchers and academics
- Less-strict quality policies
- No need to worry about safety
- Graduation process for algorithms to get them into Autoware.Auto

End Goal

Autoware.Auto

- Full-functionality self-driving stack in ROS 2
- Near-production quality
- Strict quality control policies
  - Design reviews
  - Code quality maintenance
  - Safety considerations
- Well-documented and mentored contributions

Autoware.AI

- The original Autoware
- Autoware.Auto as the bridge
- For those who won't or can't migrate to ROS 2
- Limited functionality
- Maintained by its users

Autoware.Sandbox

- Box to hold proposed extensions and modifications to Autoware.Auto
  - For researchers and academics
- Less-strict quality policies
- No need to worry about safety
- Graduation process for algorithms to get them into Autoware.Auto

A happy community!

Thanks!

Questions?

Links

- https://www.autoware.org/
- https://gitlab.com/autowarefoundation/autoware.auto/AutowareAuto
- https://gitlab.com/autowarefoundation/autoware.ai
- https://www.apex.ai/post/porting-algorithms-from-ros-1-to-ros-2