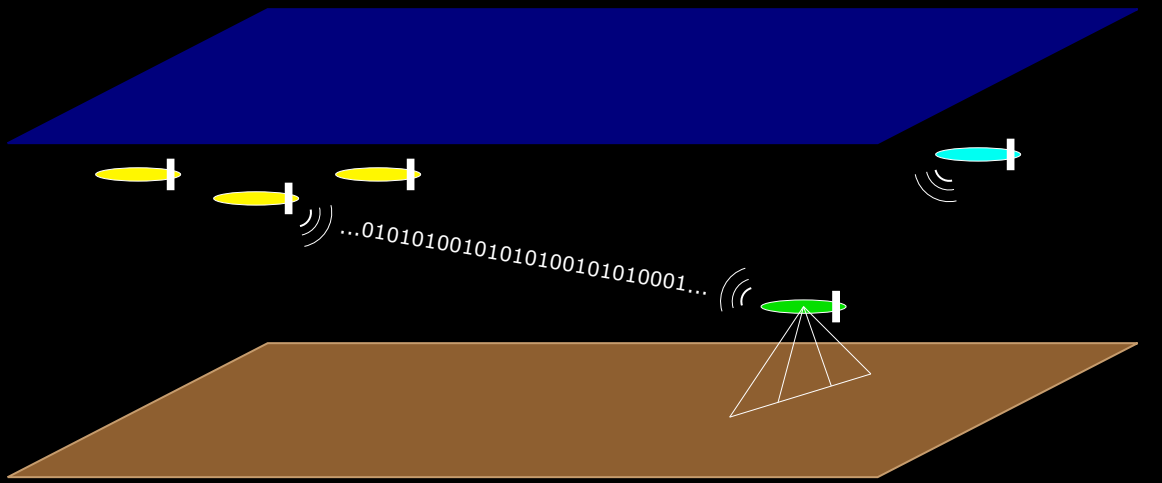
Goby3 in ROS 2

ROS Maritime Community Group Meeting, Oct 2024



Toby Schneider

GobySoft, LLC

https://github.com/GobySoft

gobysoft.org
aquatic software

What is Goby?

- A C++ library with modular components for:
  - acoustic communications (drivers, MAC, queuing)
  - utilities for marine software
  - time functions, supporting faster-than-realtime sims
- A "nested comms" publish/subscribe middleware:
  - a common transport interface for thread-to-thread, process-to-process, and vehicle-to-vehicle comms
  - application base classes for quickly writing processes
  - extensible marshalling schemes, supporting Protocol Buffers, MAVLink, DCCL, and easily many more.
  - a growing list of useful applications (GUIs, GPSD, logger, ...) and threads (I/O: serial, UDP, TCP, CAN, ...)

Why create Goby?

Marine robotics is small field of engineering, but with some relatively unique technical problems:

- communications (low throughput / high latency)
- harsh environments (expensive deployments)
- wide array of sensors, many specific to seawater

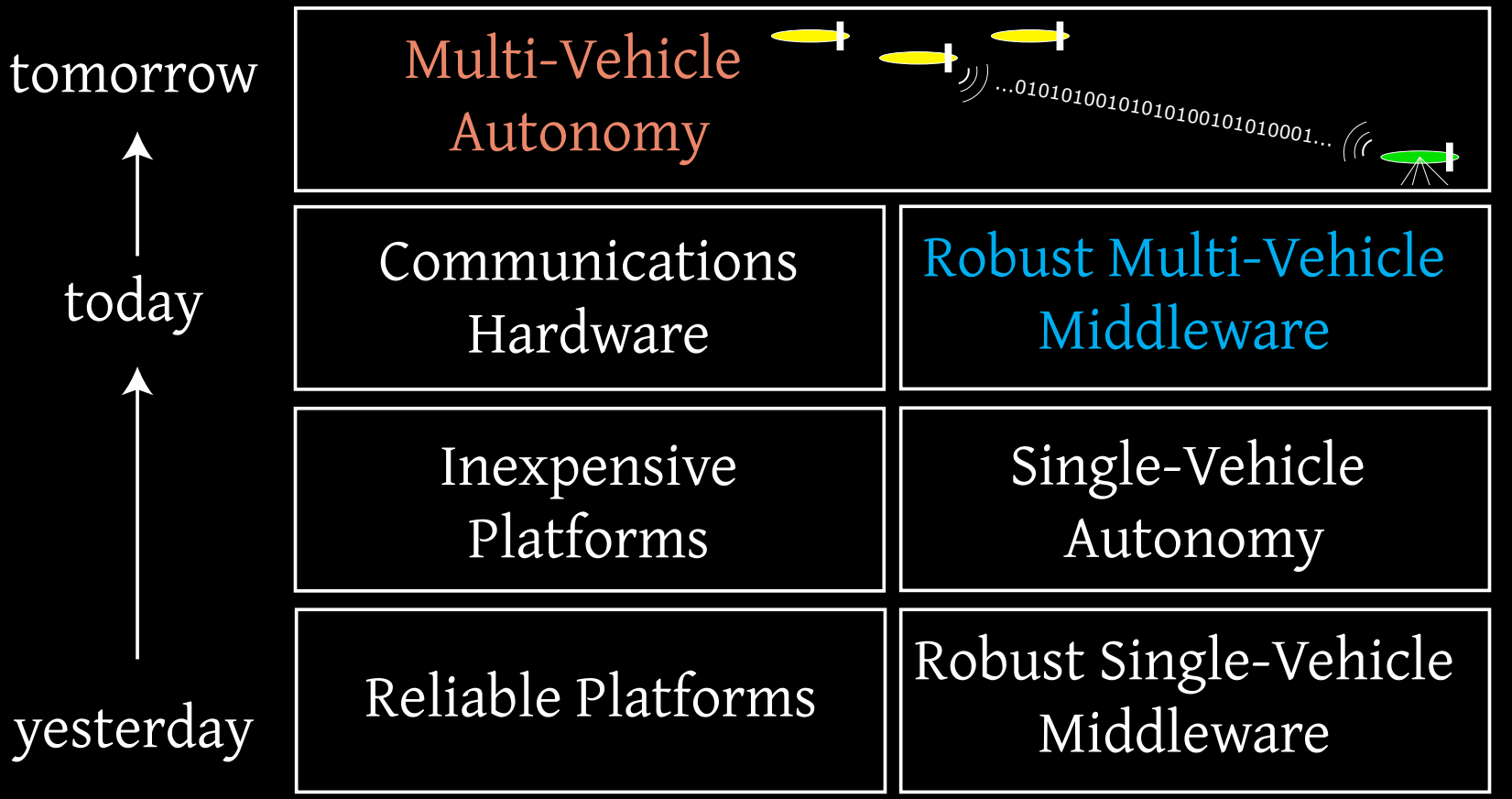
My hope:

Goby is a beginning to some solutions.

More importantly, it is open source so that we (as a community of oceanographic engineers) can build on it, and improve our use of software best practices along the way.

The Road to Swarms

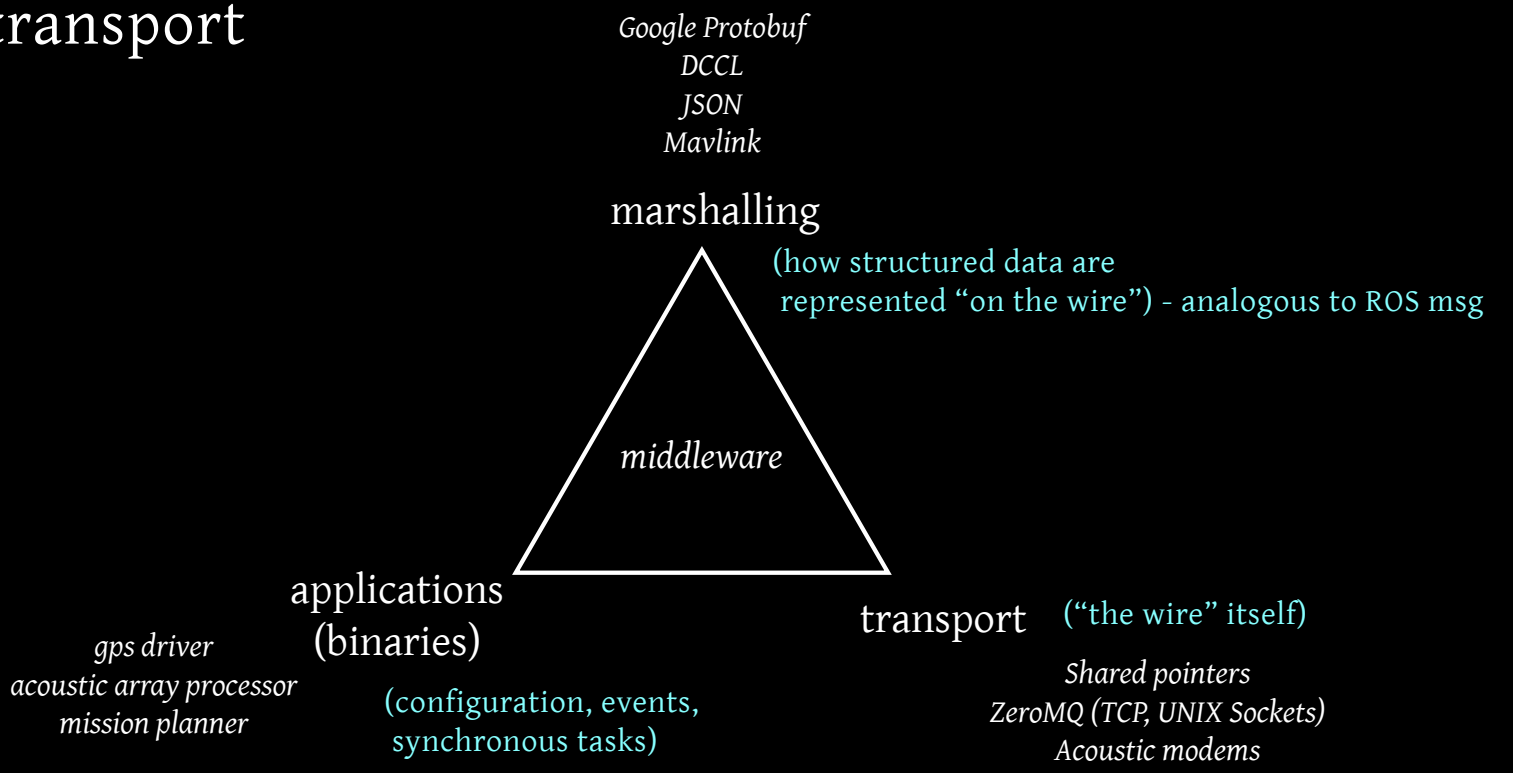
Multi-vehicle autonomy needs to be supported by multi-vehicle middleware.



Goby3 (Meta?) Middleware

This triad represents composition:

- marshalling + transport + application support = Goby3 middleware
- "meta-middleware": bring your own marshalling + transport



Goby3 Innovations

Nested communications

- Communications approach varies based on characteristic throughput and latency scale.

Neutral about choice of transport & marshalling scheme

- Allows easier inclusion of external innovations on networking and data presentation.
- New or project-specific marshalling languages can be used without middleware modification
- Interoperability with existing middleware: MOOS and ROS 2

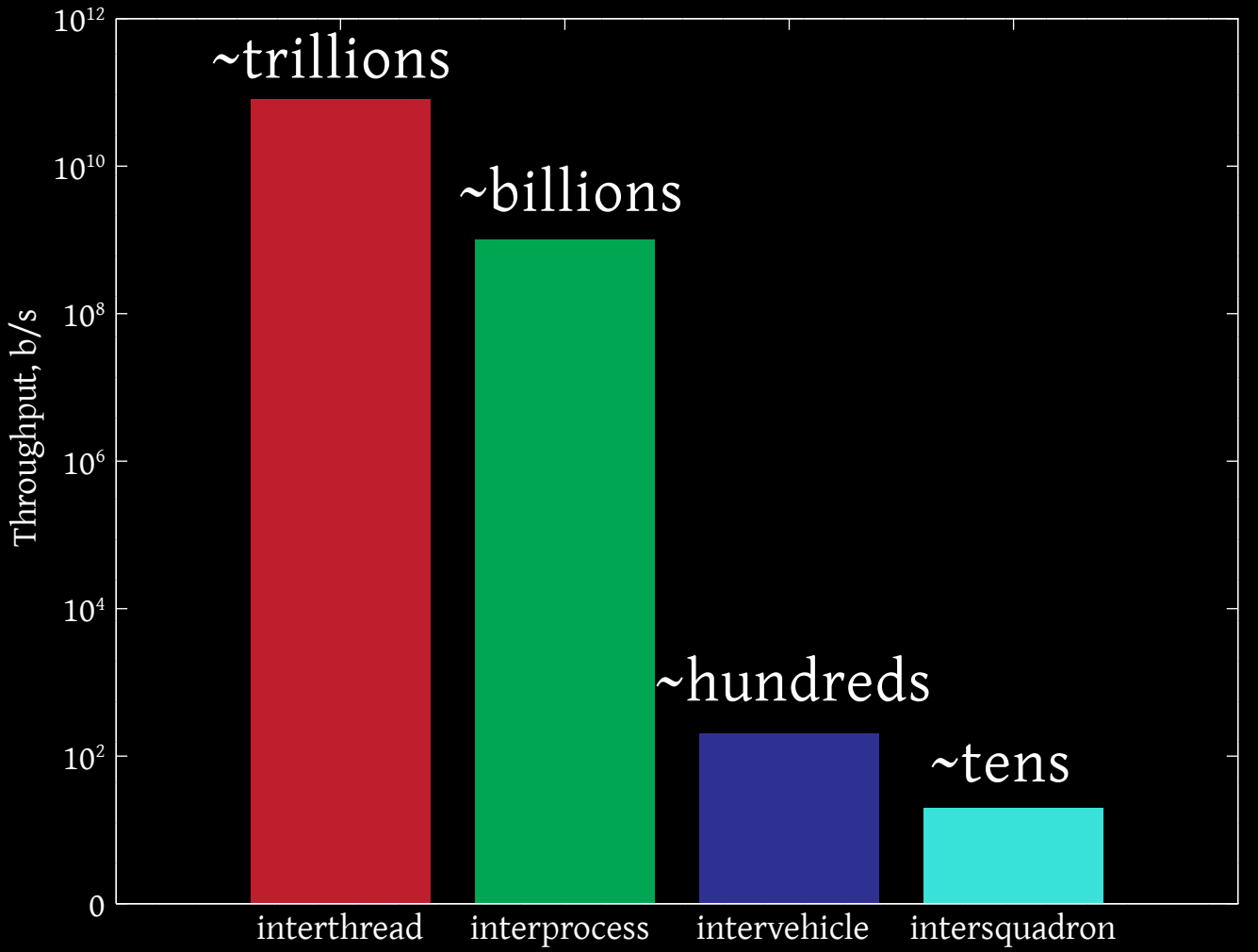
Nested Communications

Core idea: Networking approaches are fundamentally different for different characteristic latency and throughput scales (orders of magnitude).

- *Non-marine systems*: interprocess comms within and amongst vehicles are on the same scale.
- *Marine systems*: interprocess comms are an entirely different scale than intervehicle comms. Why?
  - Acoustic communications (fundamental physical limitations)
  - Sparse deployments (require satellite comms, long range radio).

Nested Communications

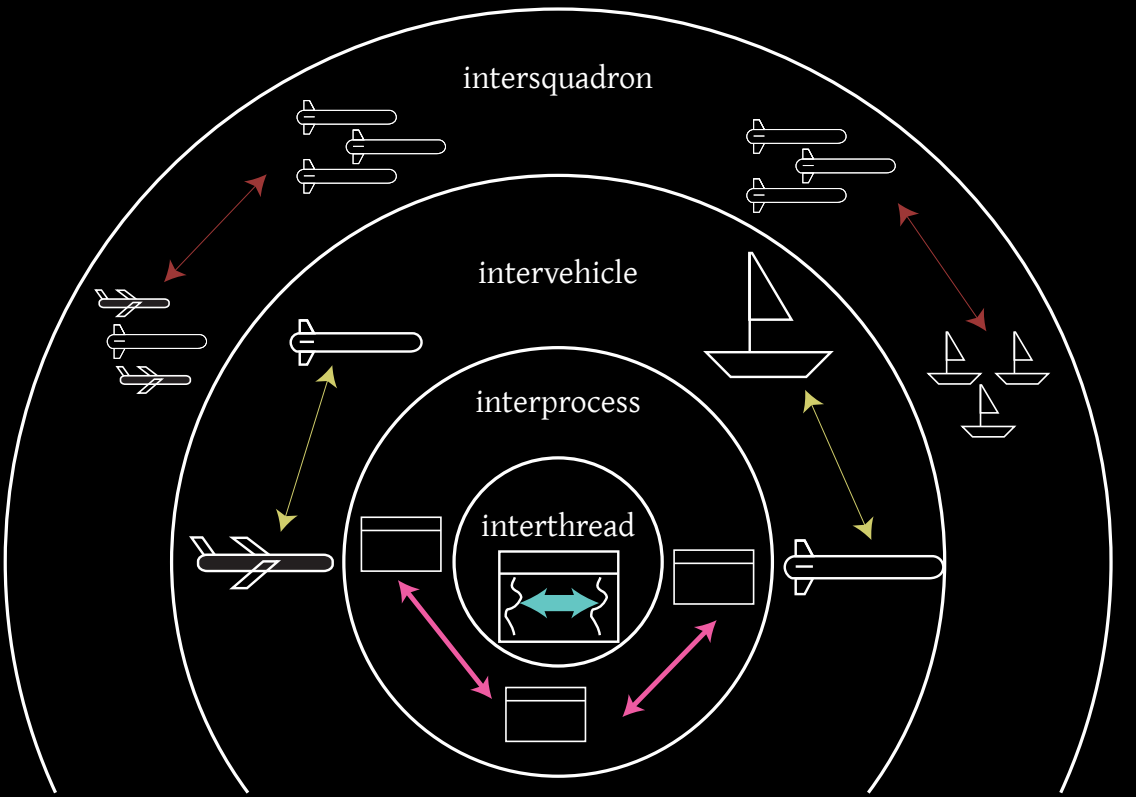
Characteristic throughput scales for AUVs:



Nested Communications

Each conceptual layer requires different transport and marshalling schemes.

Goby3 allows this and provides a common interface paradigm for publish/subscribe and cross layer forwarding.



Transport(er) Classes

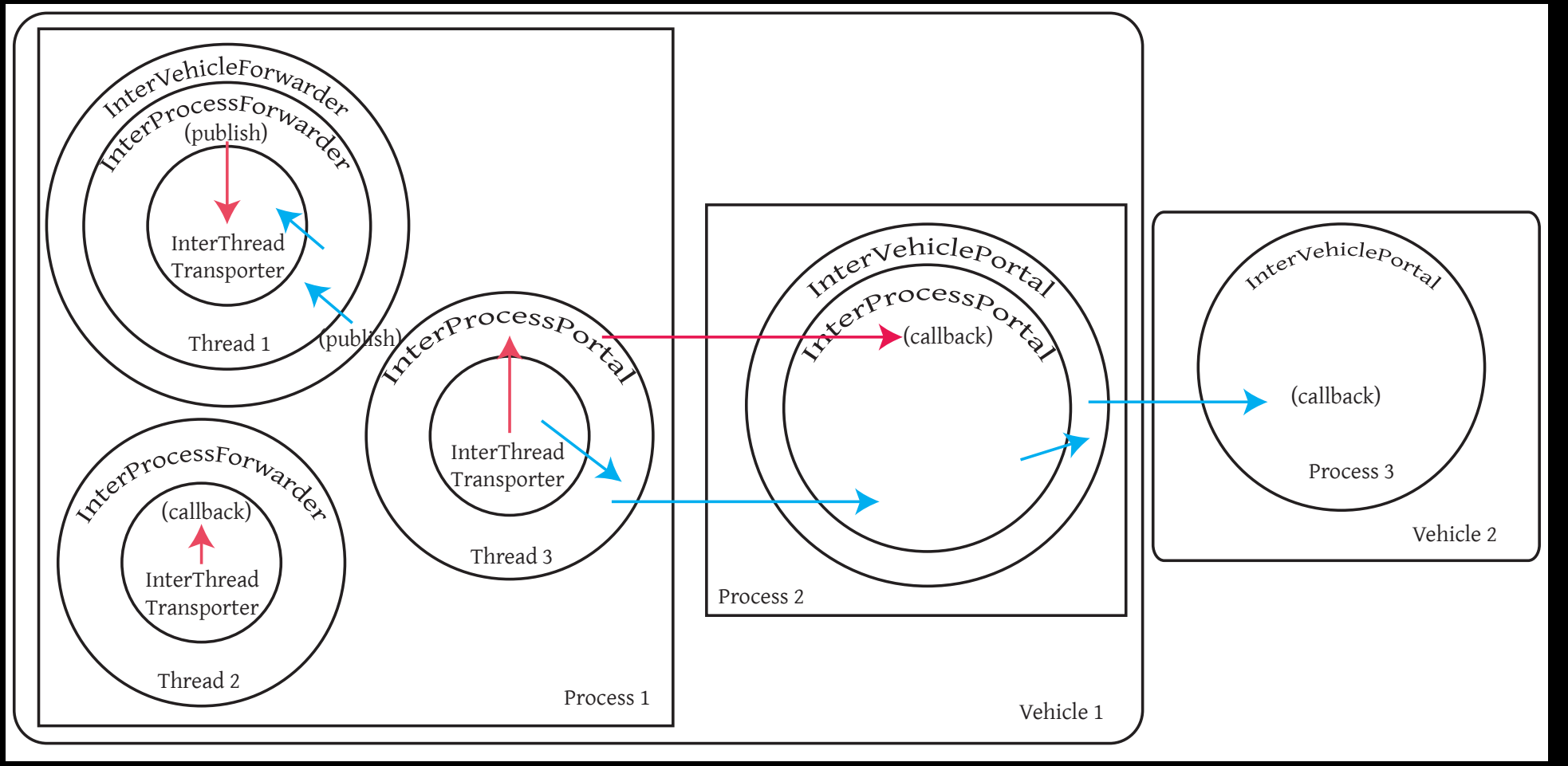
Nested comms transporter classes are in two categories (or concepts)

- Portal concept: connects one nested layer to the next. Exactly one portal per node per layer.
  - interthread: node == thread
  - interprocess: node == process
  - intervehicle: node == vehicle

  Forwarder concept: forwards data from inner layer nodes to the layer's portal. Any number per node.

  For example: InterProcessPortal - connects the threads within a process to the rest of the processes. InterProcessForwarder - forwards data from other threads to the InterProcessPortal (and thus to other processes).

Portal/Forwarder Example



- blue is published by the InterVehicleForwarder on Vehicle 1 / Process 1 / Thread 1, and subscribed by the InterVehiclePortal on Vehicle 2 / Process 3
- red is published by the InterProcessForwarder on Vehicle 1 / Process 1 / Thread 1, and is subscribed by Vehicle 1 / Process 1 / Thread 2 & Vehicle 1 / Process 2.

Reference Implementation

Goby3 provides a framework for defining nested layers, and attaching transport layers and marshalling schemes.

For immediate use, it also provides a three-level open source reference implementation in C++14.

- Interthread: C++ shared pointers: all types supported
- Interprocess: ZeroMQ (TCP/UNIX Sockets): Google Protocol Buffers, DCCL, and JSON types, user extensible to any serializable type
- Intervehicle: Goby-Acomms slow link transports and DCCL types

Fully event driven; based on condition variable signaling.

Marshalling Schemes

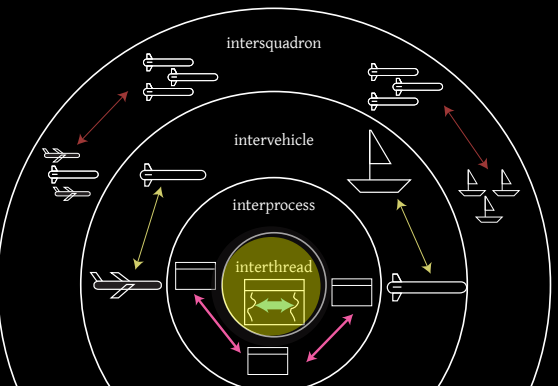
Marshalling is “the process of transforming the memory representation of an object to a data format suitable for storage or transmission” (Wikipedia)

- More concretely, in C++,
  struct/class -> array<byte> -> struct/class
- Many support multiple programming languages
- Example approaches include Google Protocol Buffers, MAVLINK, DCCL, JSON, MessagePack, ASN.1:
  - these are “marshalling schemes” in Goby3 (or just “schemes” for short)
- ROS msg is a marshalling scheme.

Transport: Interthread

Zero-copy publish/subscribe implementation using std::shared_ptr. No constraints on allowable message types representable in C++.

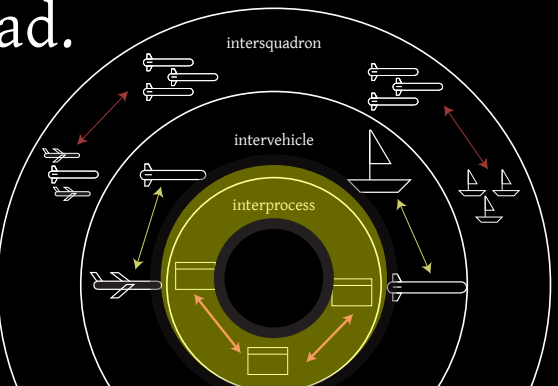
- publisher thread ---shared_ptr---> subscriber thread(s)
- Allows users unfamiliar with thread memory management to easily write multithreaded code.
- Allows multi-process code to be changed to multi-thread when needed to improve performance without changing message passing paradigm.



Transport: Interprocess

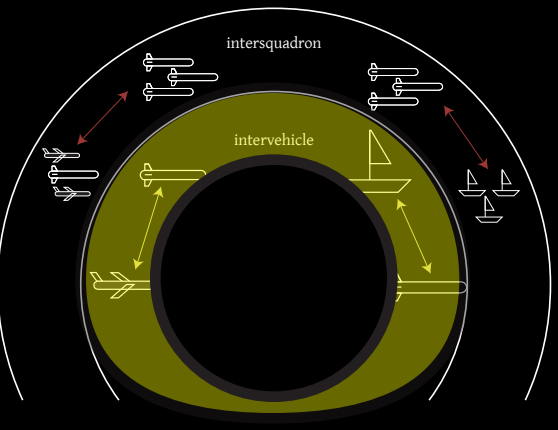
Reference implementation uses ZeroMQ. Types supported are DCCL and Google Protocol Buffers, but trivially extensible to any serializable type (by implementing functions for that marshalling scheme).

- Broker (gobyd) handles multiple publishers and subscribers over TCP or Unix Sockets.
- Subscription forwarding: publishers that have no subscribers use no bandwidth
- The inner layer (if any) is typically InterThread.
- MOOS, ROS comms is roughly analogous to this layer.



Transport: Intervehicle

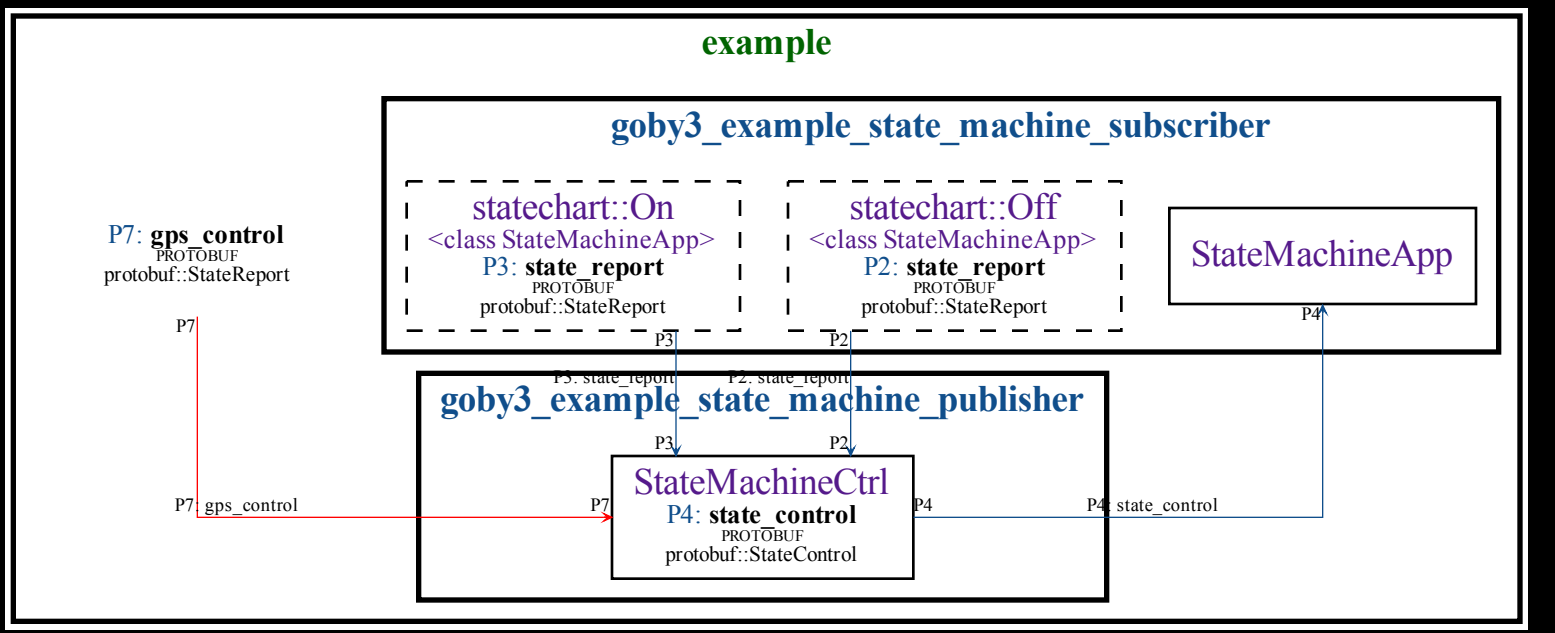
- ModemDriver: Common interface to “slow link” devices with implementations for various acoustic, satellite, etc. modems (MicroModem, Benthos, Iridium, ...).
- DCCL (libdccl.org): static unit-safe lossless marshalling that allows for physics-based bounds to create arbitrary sized fields (e.g. 6-bit int for temperature field for 0° to 40° C)
- Dynamic Buffer: deadline sensitive priority queuing
- Subscription forwarding: published data does not get sent if no subscriber exists.
- The inner layer is typically InterProcess.



Static Analysis

Emphasis on static type safety via template "groups" (goby::middleware::Group). (Somewhat analogous to ROS topics).

By using constexpr, we can generate static pub/sub graphs (analogous to rqt Node Graph but at *compile time*):



Interoperability: ROS 2

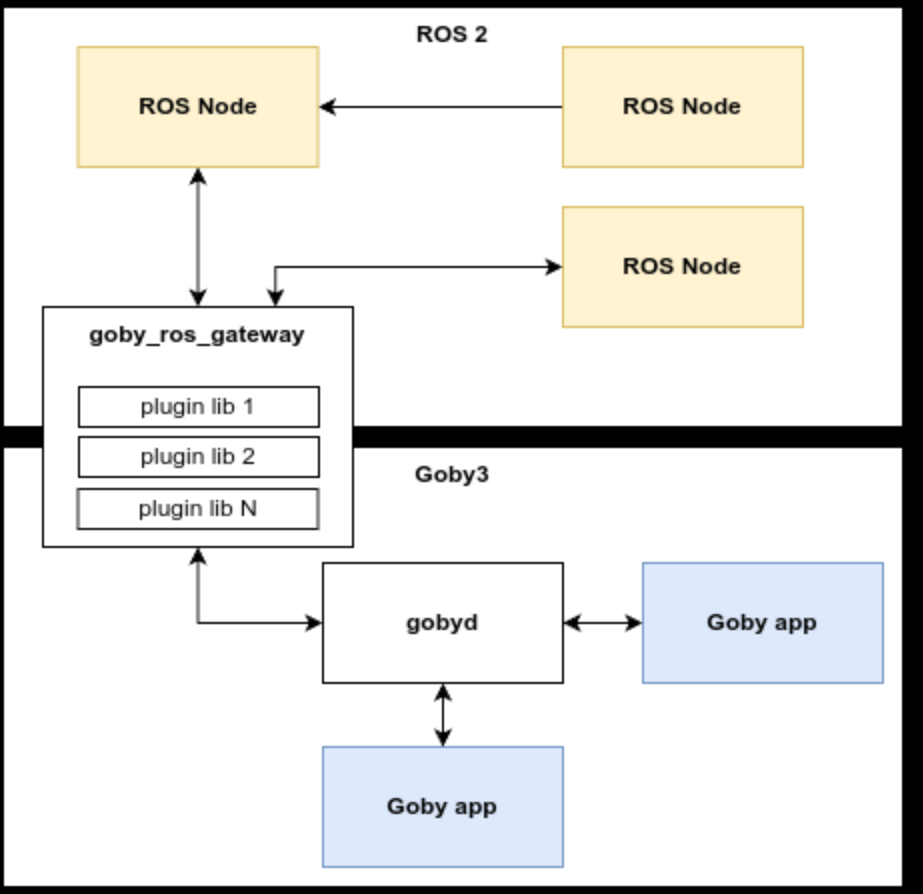
Why?

- ROS has substantial collection of useful robotics software packages.
- Goby has code (especially acoustic comms) that is useful to the marine community.
- Collaboration between entities with ROS codebase and Goby codebase (and/or MOOS codebase via Goby)

Interoperability: ROS 2

How?

- goby_ros_gateway: hybrid ROS 2 node / Goby3 app
- user plugins: define topics/groups and scheme translations



Interoperability: ROS 2

Main challenge is incompatible messaging:

- Goby3 supports any serializable marshalling scheme.
- ROS 2 has ROS msg. Using ROS msg without ROS is not straightforward (unlike Protobuf, JSON, etc.).

Three main use cases:

- Using existing ROS package in Goby3.
- Using existing Goby3 apps in ROS.
- Developing new Goby3 and ROS code in parallel.

Many different potential solutions to this problem!

Interoperability: ROS 2

Some possible solutions:

- Case 1: Using existing ROS package in Goby3.
  - [ROS]: Publish ROS msg as JSON (using rosidl_runtime_py.convert)
  - [goby_ros_gateway]: Pass through JSON msg
  - [new Goby app]: Subscribe to JSON version of ROS msg in Goby 3 app.
- Case 2: Using existing Goby3 apps (e.g. using Protobuf/DCCL) in ROS (*This is the scenario I will demo*).
  - [Goby app]: Publish Protobuf message (or DCCL)
  - [goby_ros_gateway]: Publish as ProtobufEncoded ROS msg (string pb_type and byte[] payload) (see goby_ros_examples::msg::ProtobufEncoded.msg)
  - [new ROS node]: Subscribe to ProtobufEncoded ROS msg and decode payload using Protobuf message.
- Case 3: Perhaps a combination of 1 & 2 solution.

Interoperability: ROS 2

Where?

- goby_ros_gateway:
  https://github.com/GobySoft/goby_ros_gateway

- user plugins examples:
  https://github.com/GobySoft/goby_ros_examples

Interoperability: ROS 2

When?

- Right now! (This code was finished last week, so I'd be happy to have feedback)
- (DEMO - Protobuf encoded scenario)
  - goby_ros_examples/include/goby_ros_examples/nav.proto
  - ros2_ws/src/goby_ros_examples/msg/ProtobufEncoded.msg