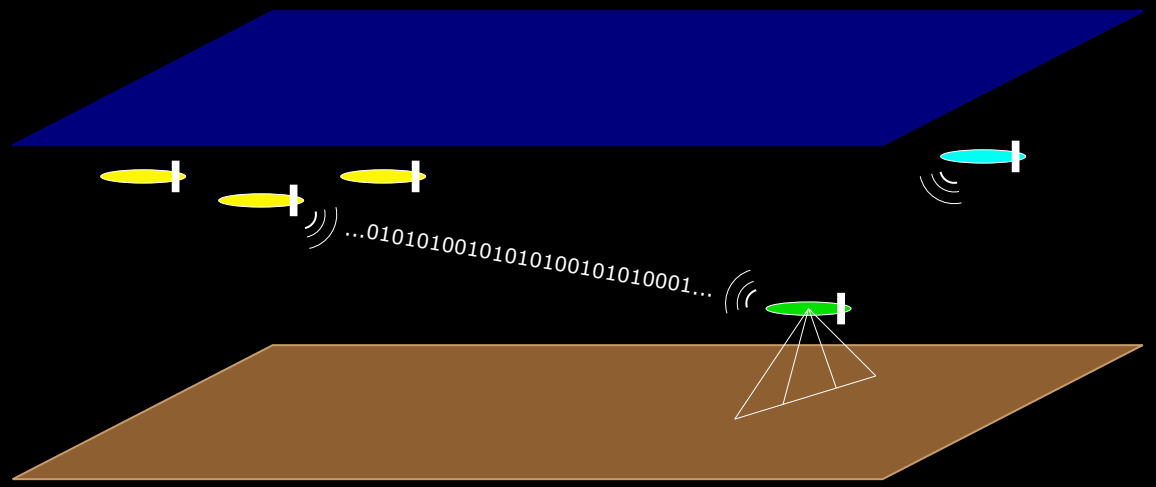
Goby3: A new open-source middleware for nested communication on autonomous marine vehicles



Toby Schneider

GobySoft, LLC

Woods Hole, MA, USA

gobysoft.org
aquatic software

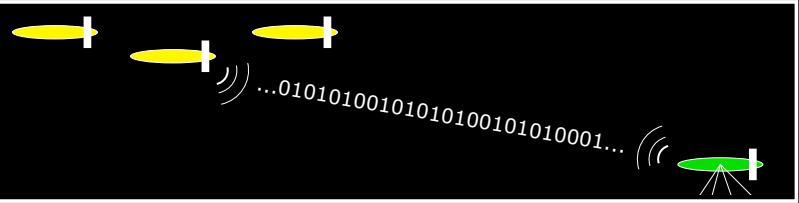
MOOS-DAWG 2017, Cambridge, MA

Evolution towards cluster autonomy

Multi-vehicle autonomy has many prerequisites ->
Success is based on the quality of the foundations!

tomorrow

Multi-Vehicle Autonomy



today

Communications Hardware

Robust Multi-Vehicle Middleware

Inexpensive Platforms

Single-Vehicle Autonomy

yesterday

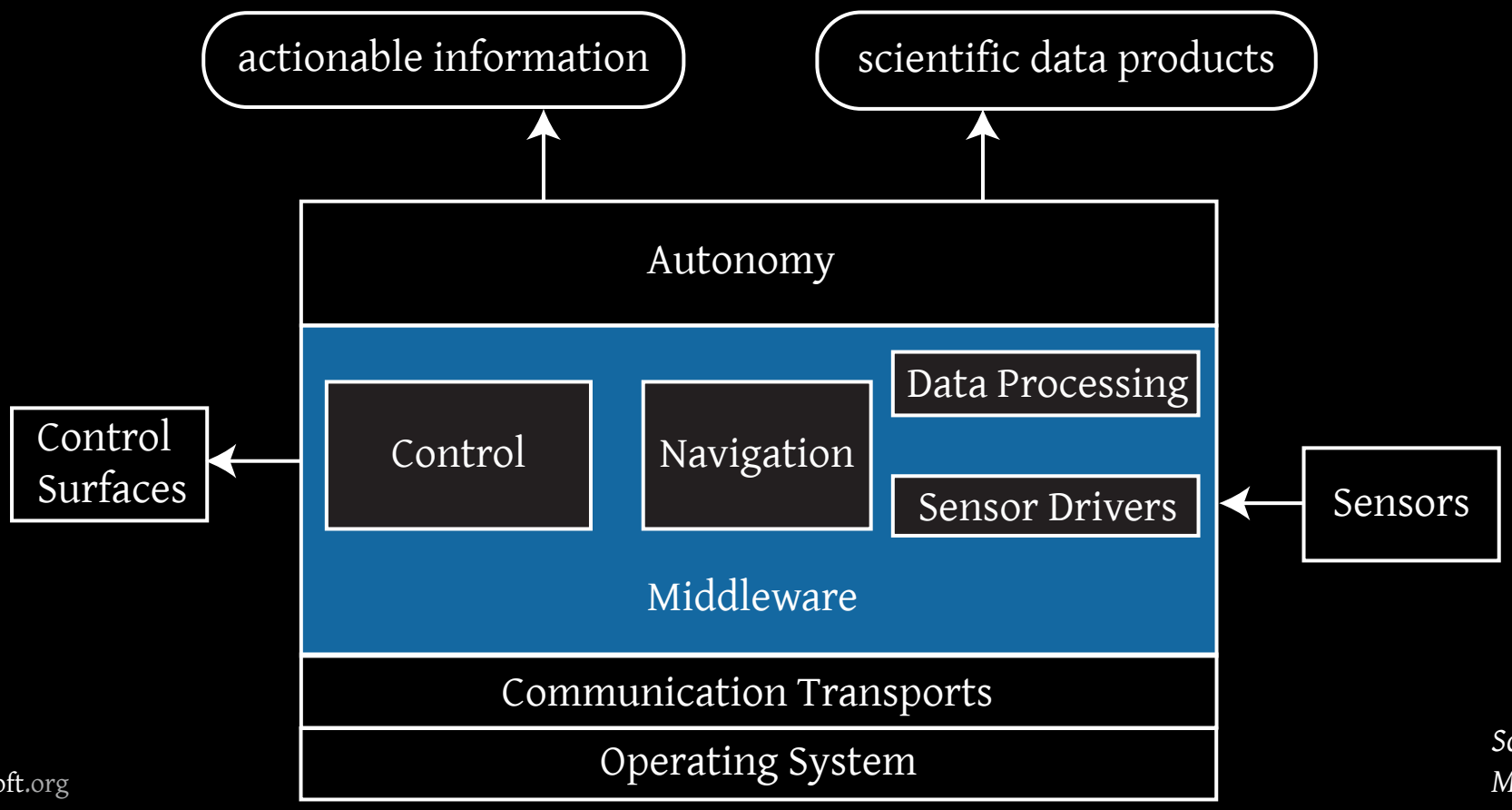
Reliable Platforms

Robust Single-Vehicle Middleware

What is middleware?

Primary role: unified asynchronous communication mechanism. Typically pub/sub or request/reply models.

Secondary roles: tools for process init, logging, management, and monitoring.



Existing middleware used in AUVs

MOOS - Mission Oriented Operating Suite

- *Transport:* TCP w/ central broker
- *Marshalling*: CMOOSMsg: string, double or binary

ROS - Robot Operating System

- *Transport:* TCP w/ central broker;
  shared pointer interthread comms
- *Marshalling:* ROSMsg: user-defined "structs"

LCM - Lightweight Communications and Marshalling

- *Transport:* UDP multicast interprocess comms,
  no central broker
- *Marshalling:* LCMType: conceptually similar to ROSMsg

None are specifically designed for marine vehicles.
All require specific transport/marshalling schemes.

Goby3 Innovations

Nested communications

- Communications approach varies based on throughput/ latency scale.

Neutral about choice of transport & marshalling scheme

- Allows easily inclusion of innovations on inter-vehicle communications
- New or project-specific marshalling languages can be used without middleware modification
- Interoperability with existing middleware

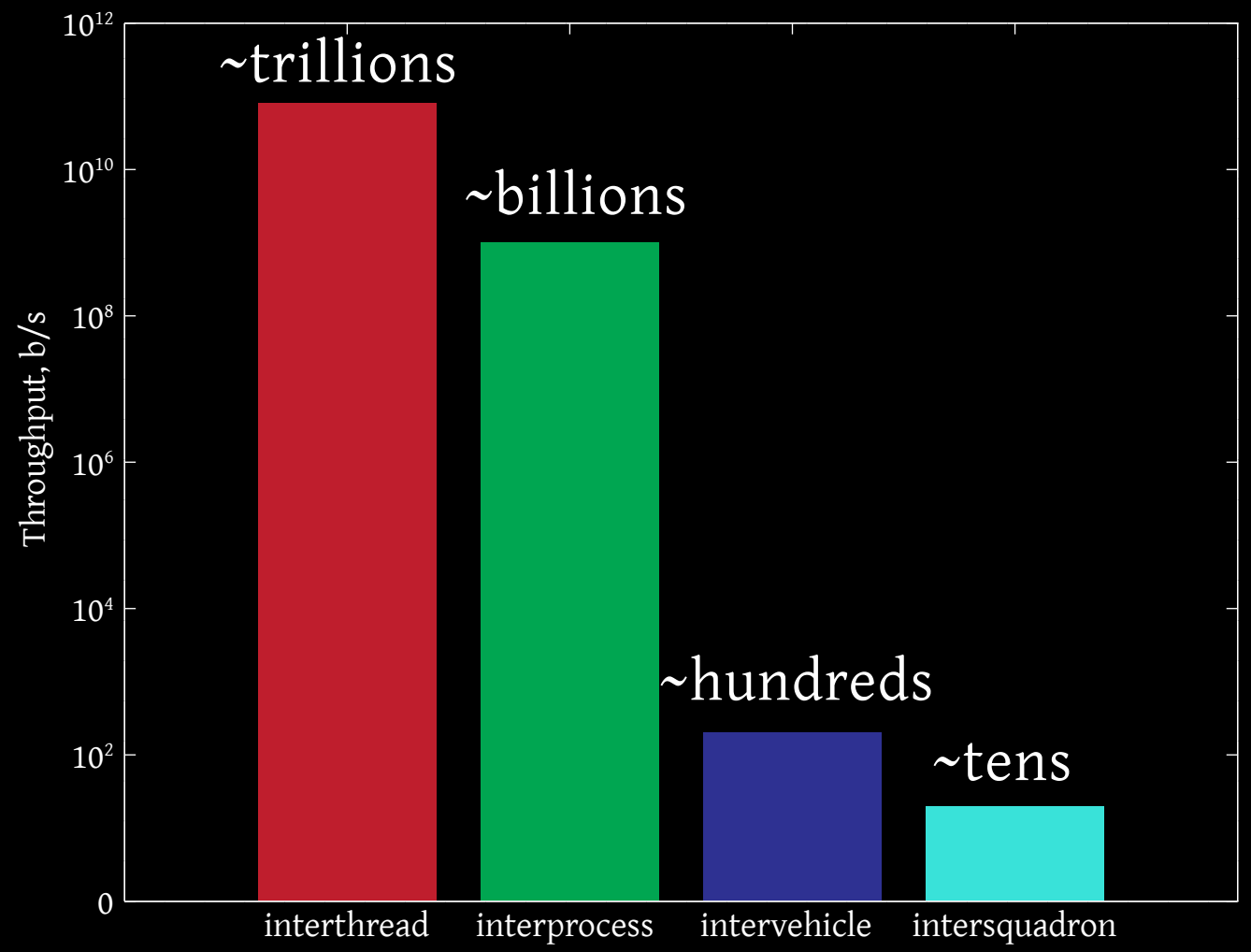
Nested communications

Core idea: Networking approaches are fundamentally different for different characteristic latency and throughput scales.

- *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

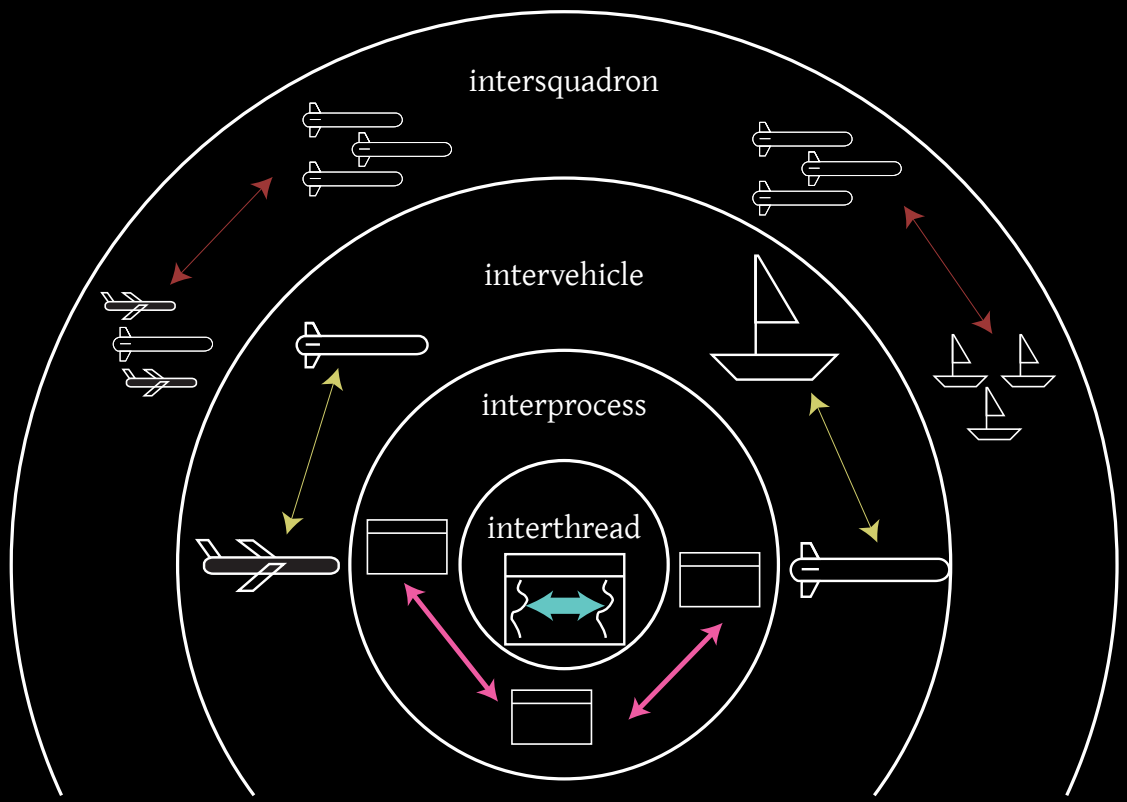
Characteristic throughput scales for AUVs:



Nested Communications

Each conceptual layer will required different transport and marshalling schemes.

Goby3 allows this and provides a common interface paradigm for pub/sub and cross layer forwarding.



Reference Implementation

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

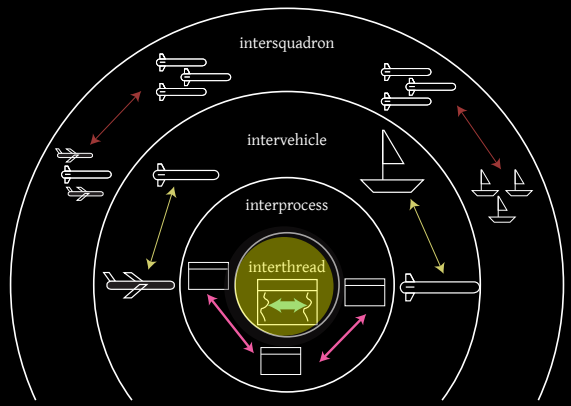
For immediate use, it also provides a three level open-source reference implementation

- Interthread: C++11 shared pointers: all types supported
- Interprocess: ZeroMQ (TCP/UNIX Sockets): Google Protobuf and DCCL types
- Intervehicle: Goby-Acomms slow link transports and DCCL types.

Interthread

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

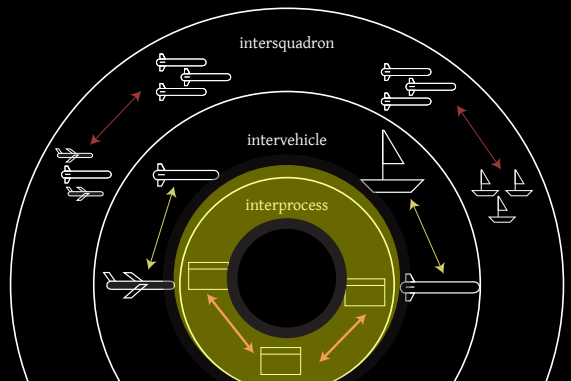
- Thread publisher ---shared_ptr--->Thread subscriber(s)
- Allows users unfamiliar with thread memory management to easily write multithreaded code.
- Allows multi-process code to be changed to multi-thread when needed to improve performance without changing message passing paradigm.



Interprocess

Reference implementation uses ZeroMQ (TCP or Unix Sockets). Types supported are DCCL and Google Protocol Buffers, but trivially extensible to any serializable type.

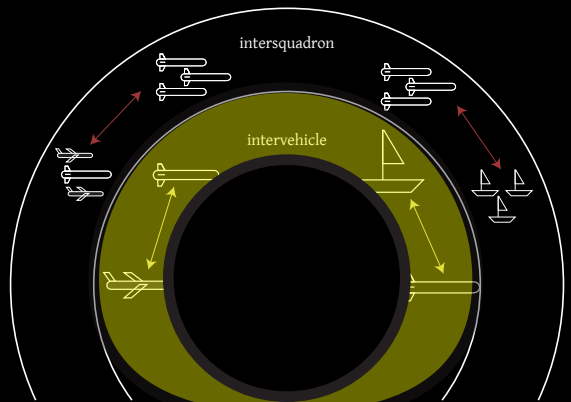
- Broker process handles multiple publishers and subscribers.
- Subscription forwarding means publishers that have no subscribers use no bandwidth
- Future releases can provide support for non-C++ languages since ZeroMQ has many bindings.
- The inner layer is typically InterThread.
- MOOS Comms is roughly analogous to this layer



Intervehicle

Uses existing (from Goby2) "slow link" communication library. DCCL (libdccl.org) used as marshalling language.

- ModemDriver layer: Common interface to "slow link" devices with implementations for various acoustic, satellite, etc. modems.
- Queue layer: Time-dynamic priority queuing
- DCCL: 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)
- (in progress) Subscription forwarding
- The inner layer is typically InterProcess.



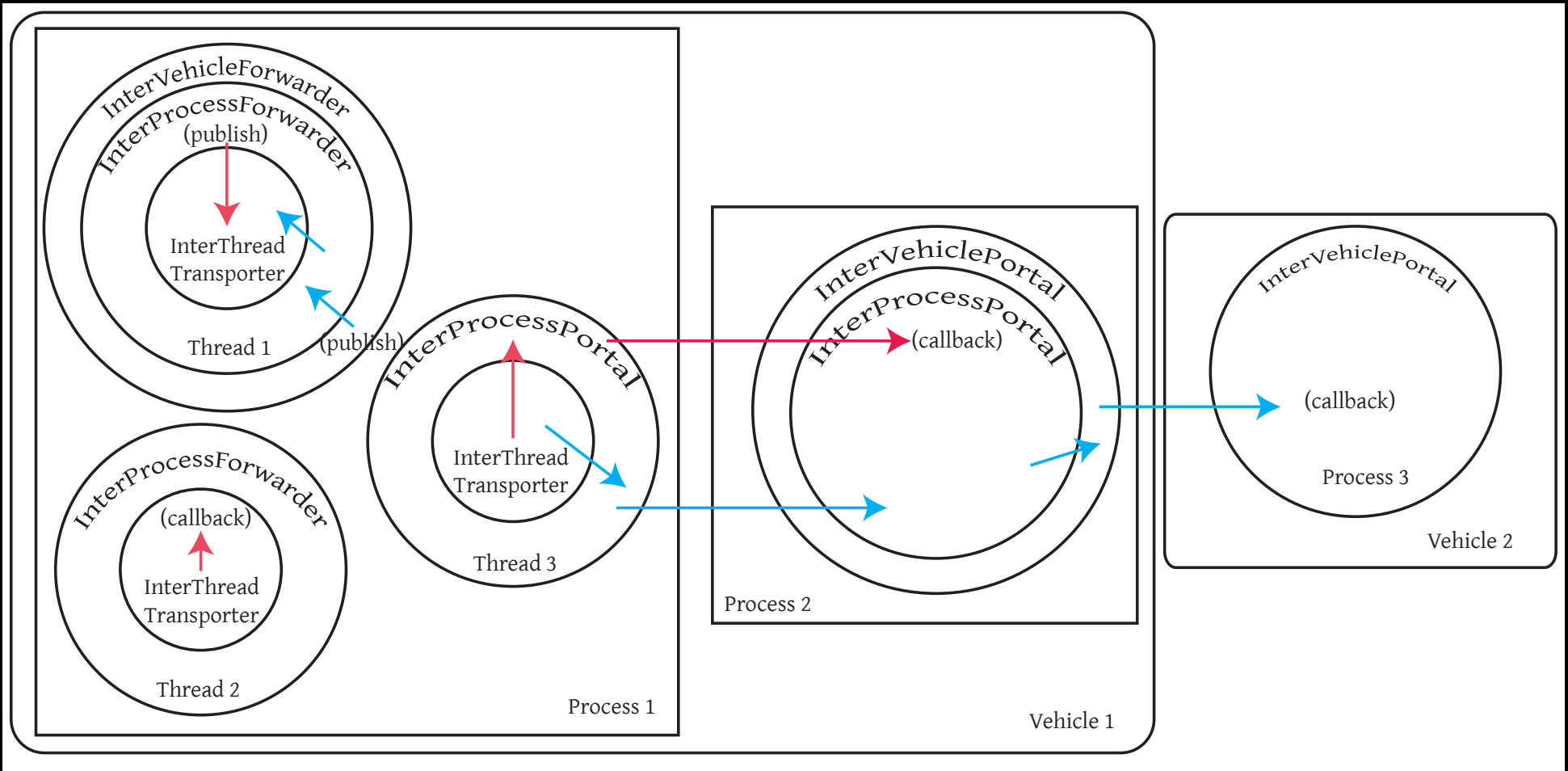
Communications Classes

Nested comms classes are in two categories (or concepts)

- Portal concept: connects one nested layer to the next. Exactly one portal per node (~= MOOS "community" at the interprocess level) per level.
- Forwarder concept: forwards data from inner layer nodes to the layer's portal via the inner layer.

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

Communications 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.

goby::SingleThreadApplication

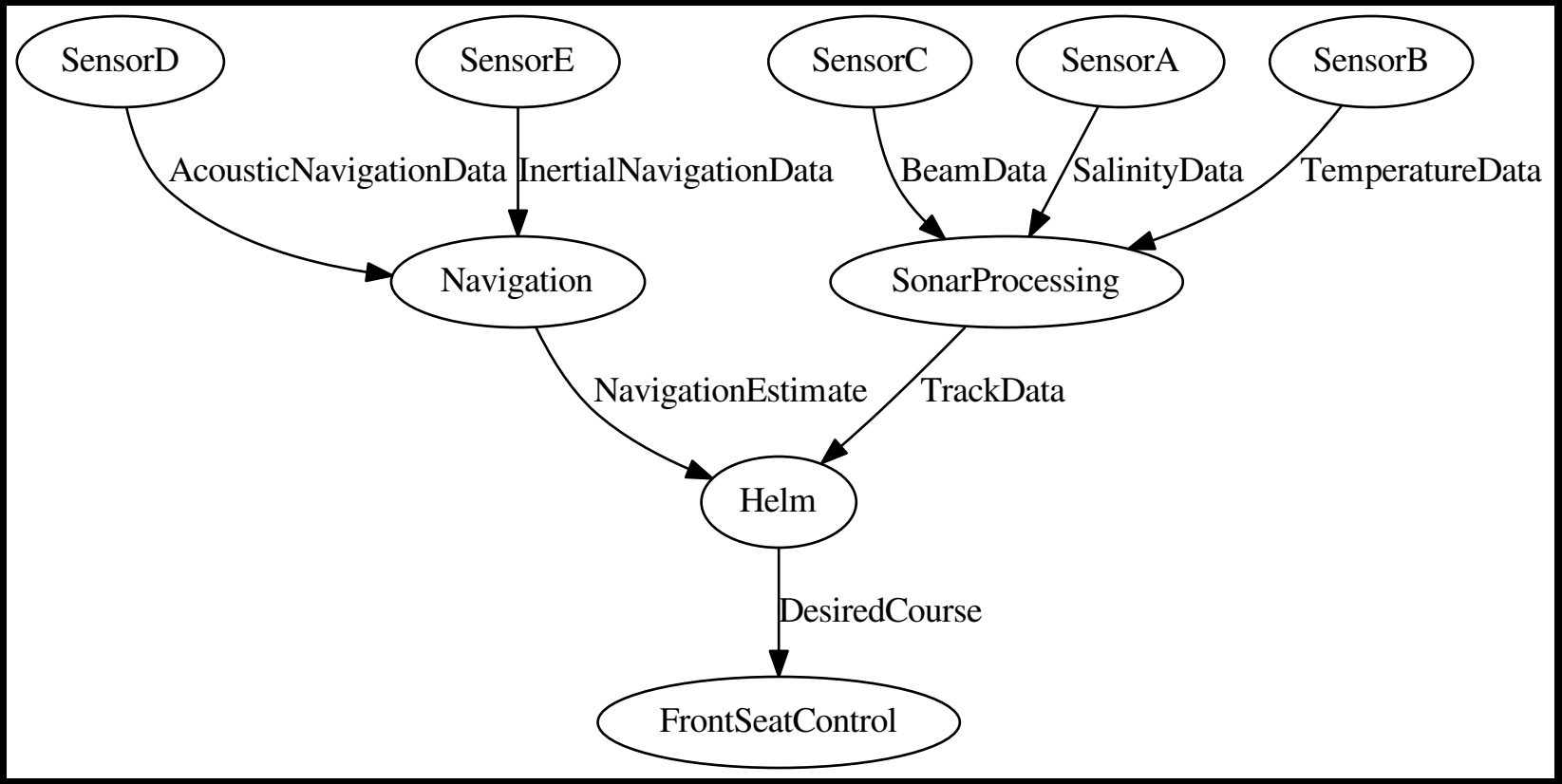
Rapid prototyping base class: Inspired by CMOOSApp, but with significant improvements.

- Automatic configuration and command-line parsing, and syntax checking into Google Protobuf message passed as a template parameter.
- Single virtual method: loop(), analog of CMOOSApp::Iterate()
- No other virtuals (Constructor handles the equivalent of OnStartup / OnConnectToServer).
- Contains an InterProcessPortal (no inner class, since this is a single-threaded application) which is accessed by transporter(), e.g. transporter().publish<...>(...) and transporter().subscribe<...>(...)

Goby3: Static Analysis

Emphasis on static type safety via template "groups" (think MOOS variable names, but defined at compile time via C++11 constexpr).

Also allows static generation of pub/sub graph:



Example (SingleThreadApplication)

publisher.cpp

```
#include "goby/middleware/single-thread-application.h"
#include "messages/nav.pb.h"
#include "messages/groups.h"
#include "config.pb.h"

using Base = goby::SingleThreadApplication<BasicPublisherConfig>;
using protobuf::NavigationReport;

class BasicPublisher : public Base
{
public:
   BasicPublisher() : Base(10 /*hertz*/)
      {
         // all configuration defined in BasicPublisherConfig is in cfg();
         std::cout << "My configuration int: " << cfg().my_value() << std::endl;
      }

   void loop() override
      {
         NavigationReport nav;
         nav.set_x(95 + std::rand() % 20);
         nav.set_y(195 + std::rand() % 20);
         nav.set_z(-305 + std::rand() % 10);

         std::cout << "Tx: " << nav.DebugString() << std::flush;
         transporter().publish<groups::nav>(nav);
      }
};

int main(int argc, char* argv[])
{ return goby::run<BasicPublisher>(argc, argv); }
```

groups.h

```
#include "goby/middleware/serialize_parse.h"
namespace groups
{
   // extern is currently required by g++ but not clang++
   extern constexpr goby::Group nav{"navigation"};
   extern constexpr goby::Group gps_data{"gps_data"};
   extern constexpr goby::Group gps_control{"gps_control"};
}
```

subscriber.cpp

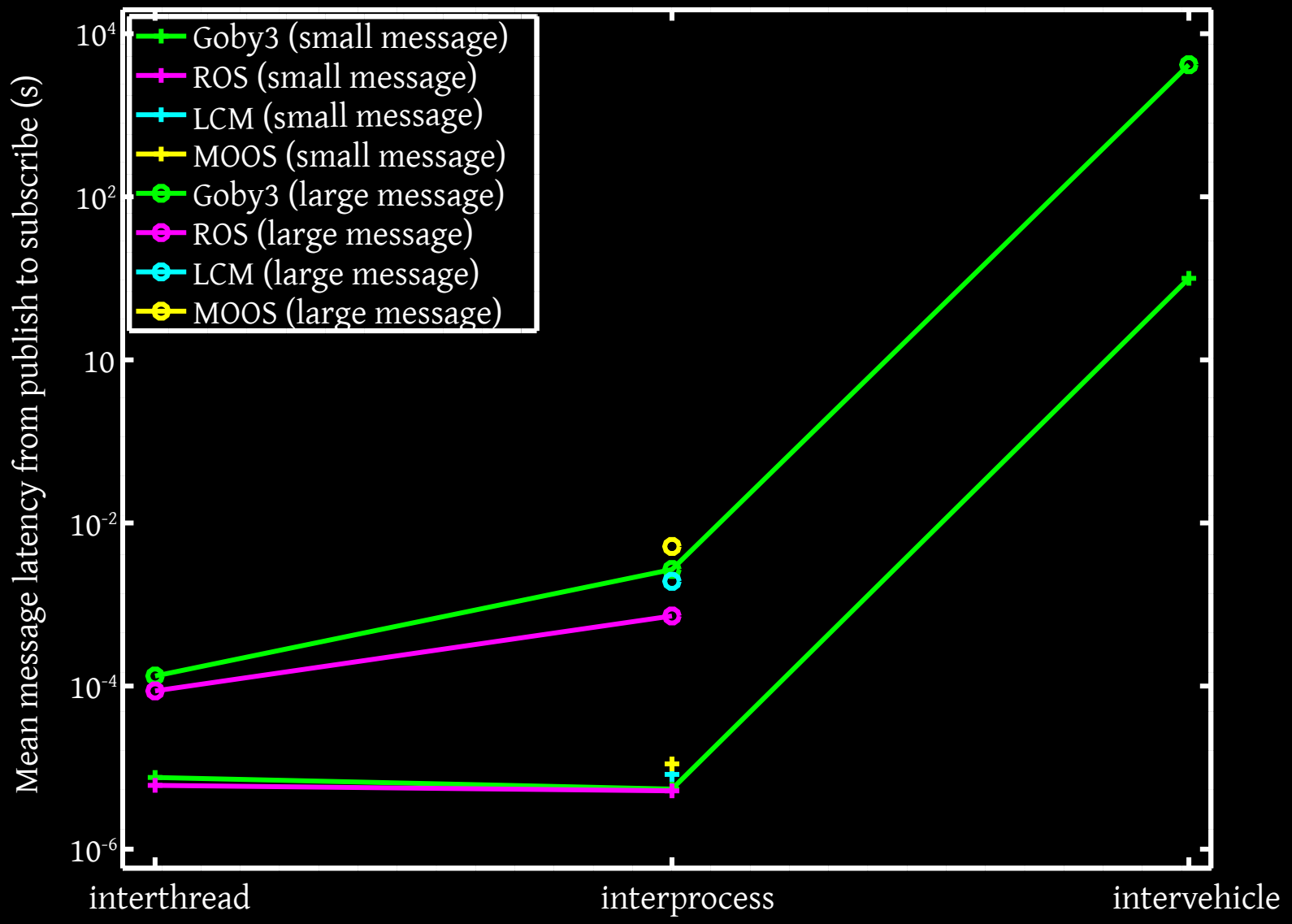
```
#include "goby/middleware/single-thread-application.h"
#include "messages/nav.pb.h"
#include "messages/groups.h"
#include "config.pb.h"

using Base = goby::SingleThreadApplication<BasicSubscriberConfig>;
using protobuf::NavigationReport;

class BasicSubscriber : public Base
{
public:
   BasicSubscriber()
      {
         auto nav_callback = [this] (const NavigationReport& nav)
            { this->incoming_nav(nav); };
         transporter().subscribe<groups::nav, NavigationReport>(nav_callback);
      }

   void incoming_nav(const NavigationReport& nav)
      {
         std::cout << "Rx: " << nav.DebugString() << std::flush;
      }
};

int main(int argc, char* argv[])
{ return goby::run<BasicSubscriber>(argc, argv); }
```

goby::MultiThreadApplication

Extends the SingleThreadApplication to multiple threads.

- Each thread is a subclass of an identical interface to the single-thread application (with a loop() virtual method, and a transporter() method to access communications.
- This approach avoids accidental thread memory contention if all data are stored at the class level (no globals).
- Any data that are shared through threads are published / subscribed via the InterThreadTransporter (think of this as a combined Portal/Forwarder, as InterThread never has an inner level).
- Threads can be spawned and joined as necessary. Joining happens as soon as loop() returns.

Performance Results



Goby3 gives comparable performance to existing middlewares

What can I do with this?

Goby3 is designed for:

- Rapid development of multiple vehicles' software
- Improved interoperability between research teams
- Ease to introduction to a project for new programmers without directly affecting existing code.
- Flexibility for future innovations in the community for new transport and marshalling languages.
- Specifically addressing the marine community (new ideas, bug fixes, pull requests, user feedback).

Currently "alpha" software:

http://github.com/GobySoft/goby3
http://github.com/GobySoft/goby3-examples