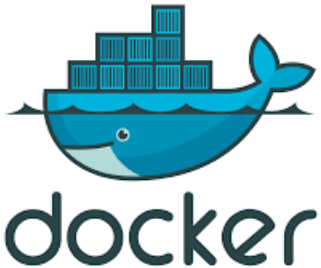
Migrating from ROS1 to ROS2 choosing the right bridge

Our System

- ROS1 + Bazel + Docker + Flow



FLOW

C++14, header-only library for multi-stream data synchronization.

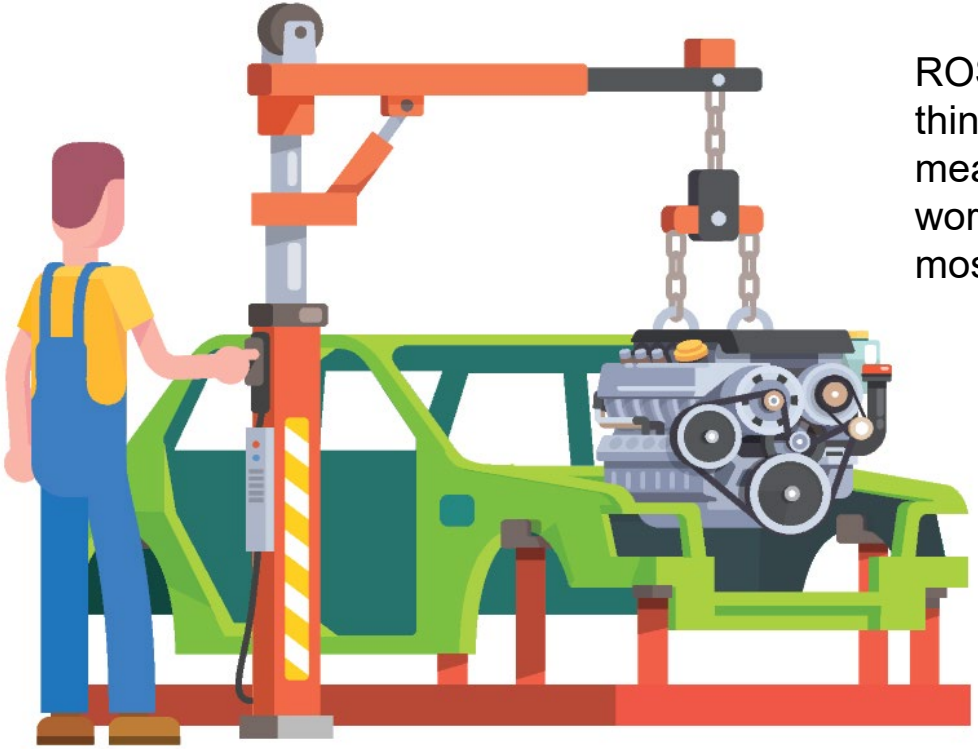
- 808 Topics
- 326 Custom ROS Messages
- >300k Lines of C++ code
- 74 Nodes
- Freight100 Computer Specs
  - Freight100 v1 : 4 Cores @ 3.00GHz + 16G RAM
  - Freight100 v2 : 8 Cores @ 2.60GHz + 32G RAM

ROS2 Conversion Strategies

1. Everything-at-once
2. Node-by-node using ros1_bridge
3. Topic-by-topic using combined ROS1/ROS2 Nodes

For Us : All-at-once Conversion

… is like replacing the engine in a moving car



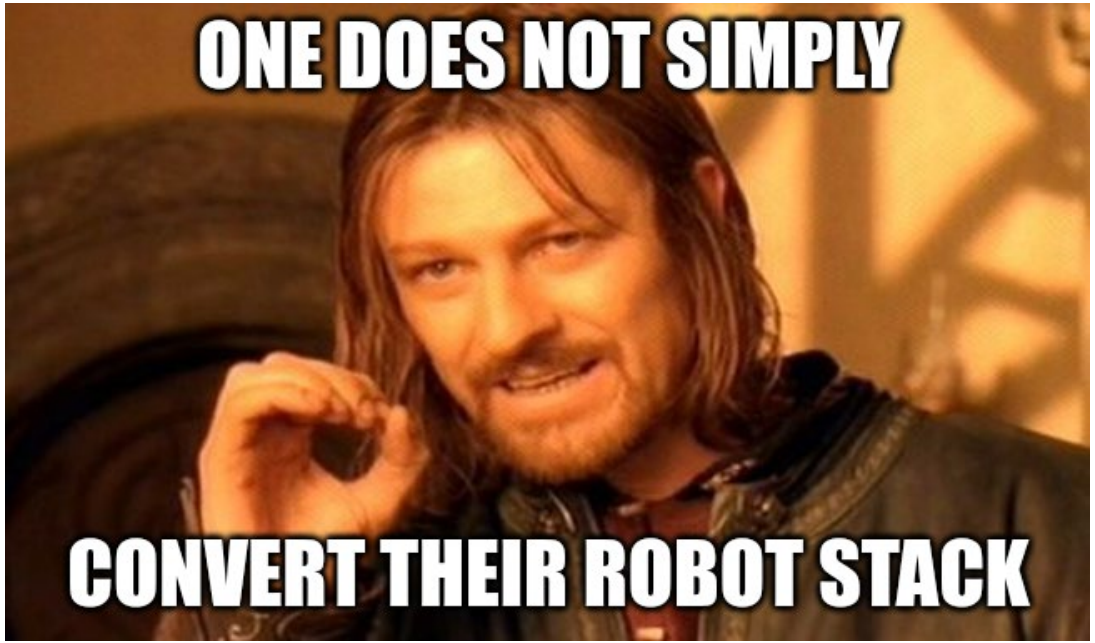
ROS is the engine that keeps things running. Switching it, means integration tests won't be working when they are needed most

Either stop making other changes and just focus on ROS2 … OR have merge conflicts with other new features

What about unanticipated issues that take a lot of extra time?

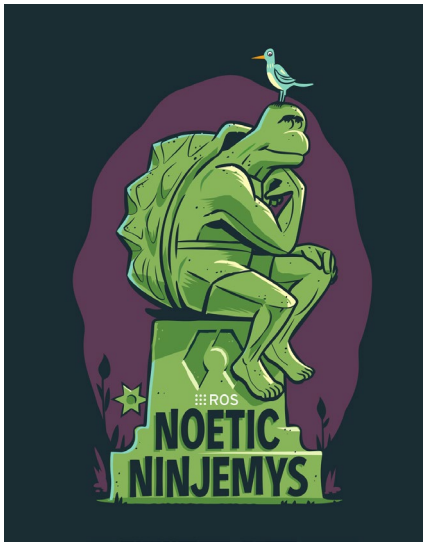
ROS2 Conversion Strategies

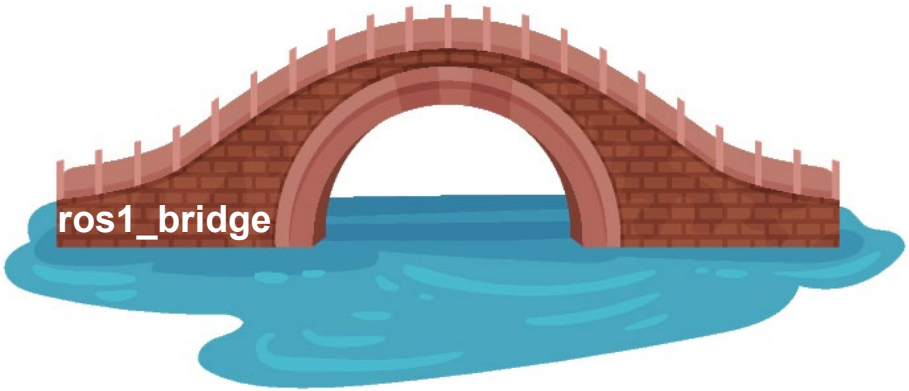
1. ~~Everything at once~~: Can’t break work into smaller pieces
2. Node-by-node
3. Topic-by-topic



ROS2 Conversion Strategies

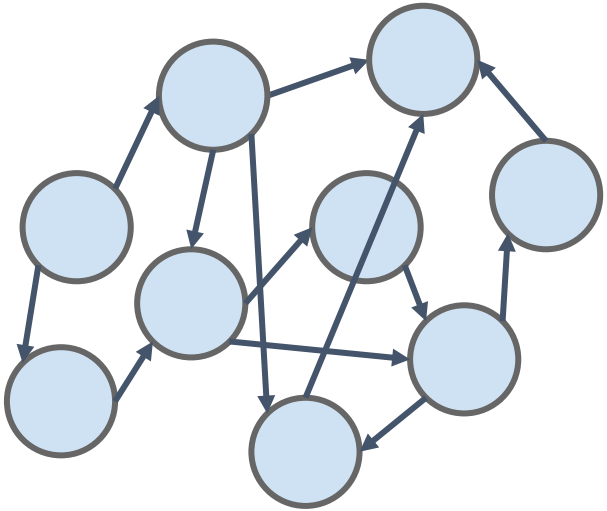
1. ~~Everything at once~~
2. Node-by-node : Using ros1_bridge
3. Topic-by-topic

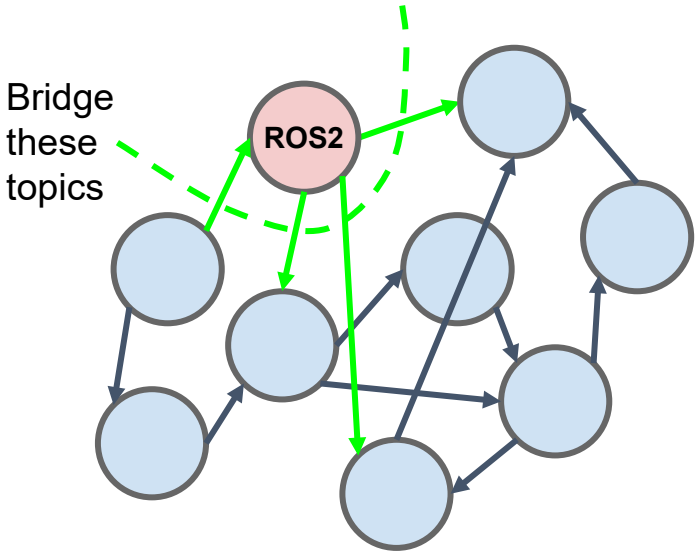




Node-by-node Conversion + ros1_bridge

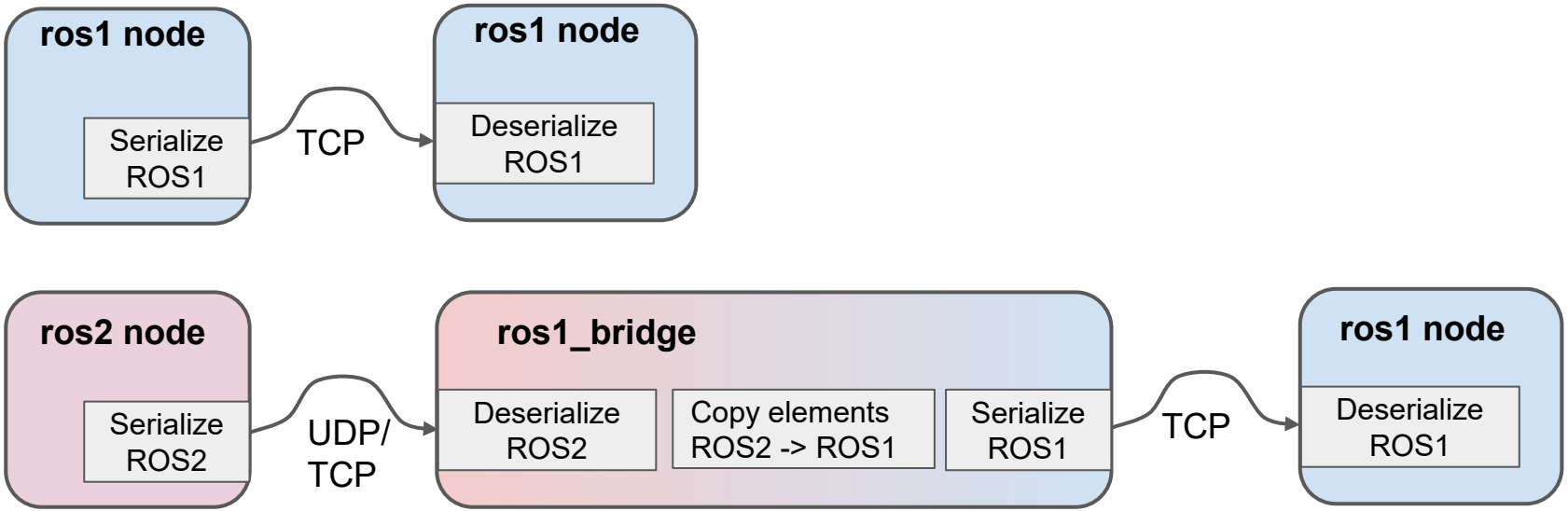
- Convert a single node to ROS2
- Bridge any topics that are connected to ROS1 nodes
- Incremental progress!





ros1_bridge : Overhead

- Extra (loop-back) network hop
- Extra deserialization
- Member-by-member copy ROS1 class -> ROS2 class
- Extra serialization



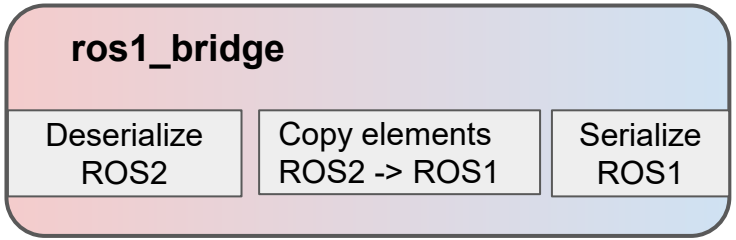
ros1_bridge : Latency and CPU Usage**

	Size (bytes)	BW (Mb/sec)	ROS1→ROS1 Latency (ms)	ROS2→ROS1 Latency (ms)	dynamic_bridge CPU %
sensor_msgs/Imu @ 100Hz	321	0.04	0.30	0.73	7.48
sensor_msgs/Image 640 **x** 360 **x** 3 @15Hz	691k	10.3	0.63	2.20	1.97 Msg Drops
sensor_msgs/Image ***reliable QOS**	691k	10.3	0.63	8.45	4.50

* ROS2 subscriber in dynamic_bridge defaults to “best effort” even if publisher is “reliable”

* Each process pinned to its own core with fixed frequency of 2.4Gz
All processes are running on same machine.

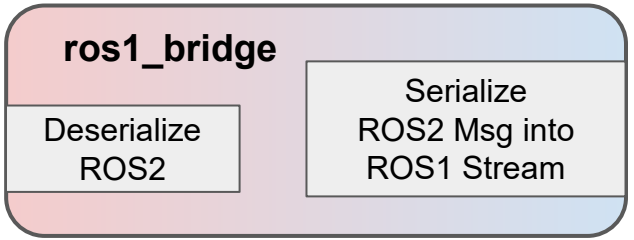
Optimization : Write ROS2 Msgs directly to ROS1 Stream




```
template<>
void Factory<...>::convert_2_to_1(
  const geometry_msgs::msg::Vector3 & ros2_msg,
  geometry_msgs::Vector3 & ros1_msg)
{
  ros1_msg.x = ros2_msg.x;
  ros1_msg.y = ros2_msg.y;
  ros1_msg.z = ros2_msg.z;
}
```

Each **`convert_2_to_1()`** call takes about 340µsec for a 640**x**360**x**3 sensor_msgs::Image

340µsec * 15Hz = 0.5% CPU Usage

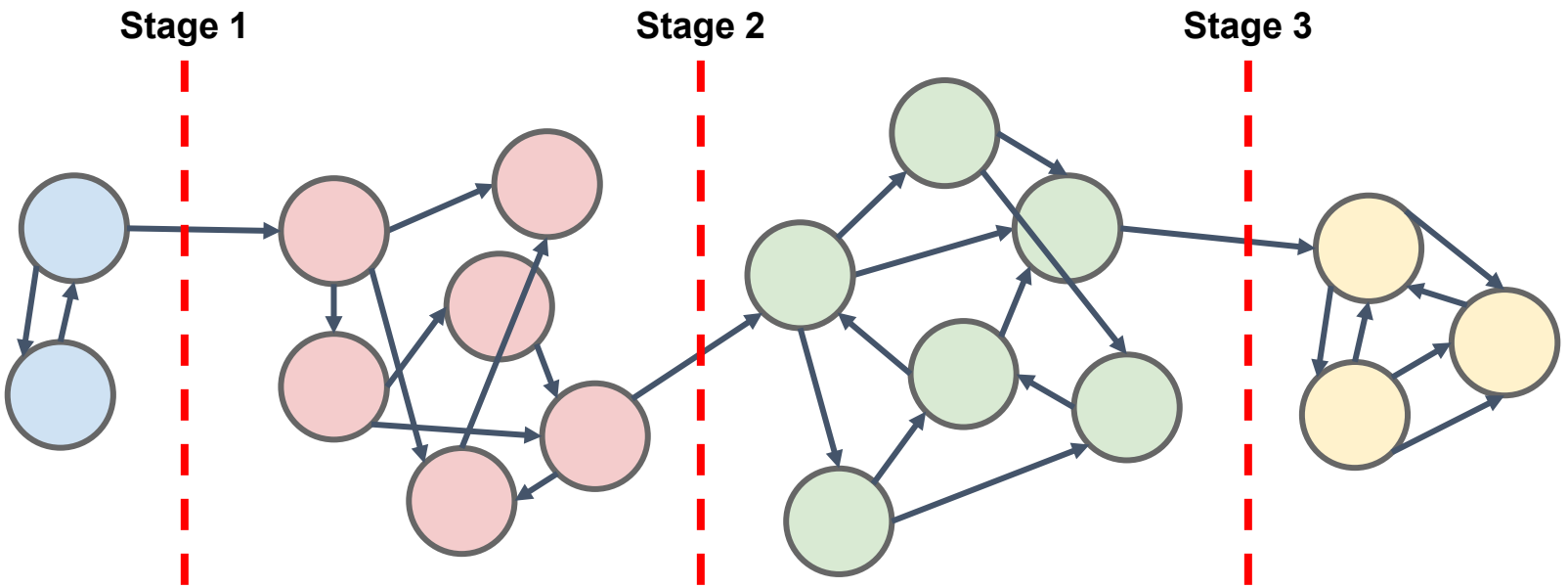



```
template<>
template<...>
Void
Factory<...>::msg_2_to_1_stream(
  STREAM_T & stream,
  ROS2_MSG_T & ros2_msg)
{
  stream.next(ros2_msg.x);
  stream.next(ros2_msg.y);
  stream.next(ros2_msg.z);
}
```

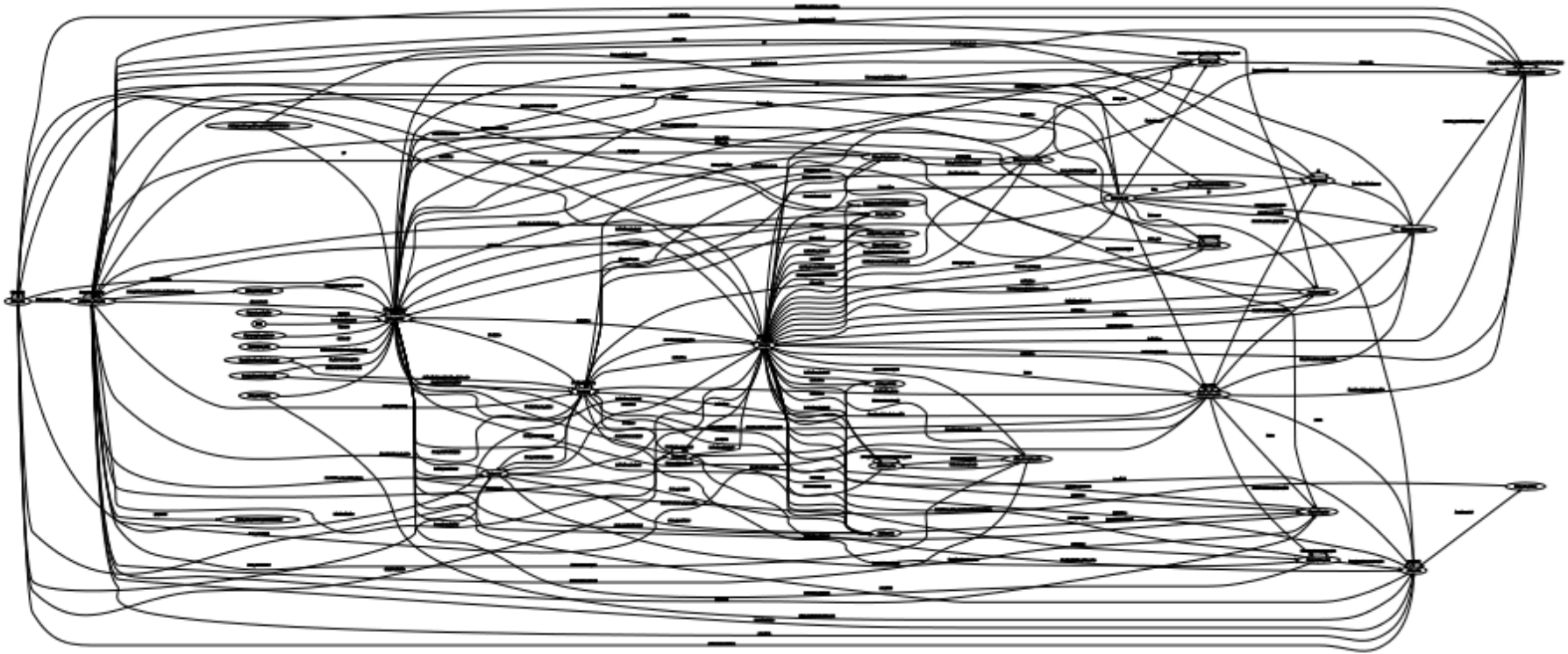
Each **`msg_2_to_1_stream()`** call also takes about 340µsec for same message

Ideal Node Graph (for bridged conversion to ROS2)

- Break up work by only converting a small cluster ROS1 to ROS2 at a time
- Ideally, only a small amount of connections to bridge between different clusters



Our Graph



Our Graph : Harder to untangle than a bowl of spaghetti

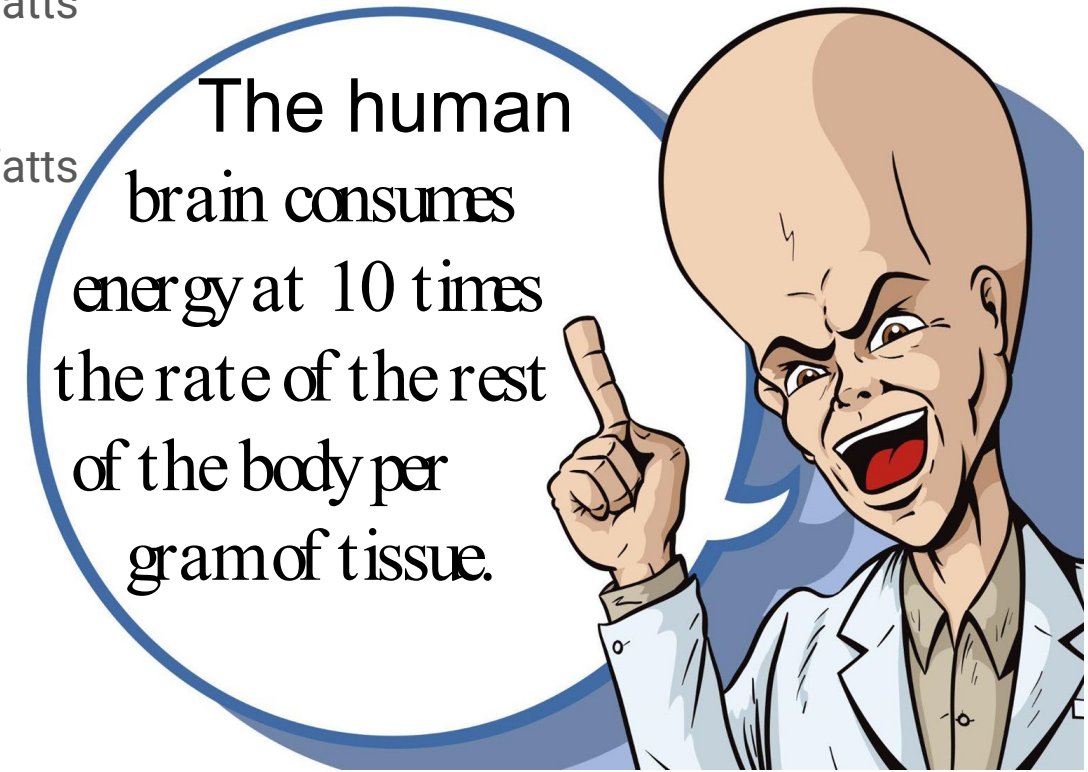
We Link Big Nodes & We Cannot Lie

- “Nodelet” message passing via shared pointers
- navigation_core_node :
  - 92 subscribed topics
  - 248 published topics
  - 358 connections :
    - 262 TCPROS, 96 INTRAPROCESS
- action_monitor :
  - 555 subscribed topics
  - 5 pub topics
  - 657 connections :
    - 651 TCPROS, 6 INTRAPROCESS
- fmcl_node :
  - 29 subscribed topics
  - 45 published topics
  - 106 connections :
    - 82 TCPROS, 24 INTRAPROCESS

CPU Usage Matters

Freight100 Power Usage Breakdown

- Stationary :
  - Computer & Sensors : 40 Watts
  - Drive Motors : 5 Watts
- Moving :
  - Computer & Sensors : 55 Watts
  - Drive Motors : 32 Watts

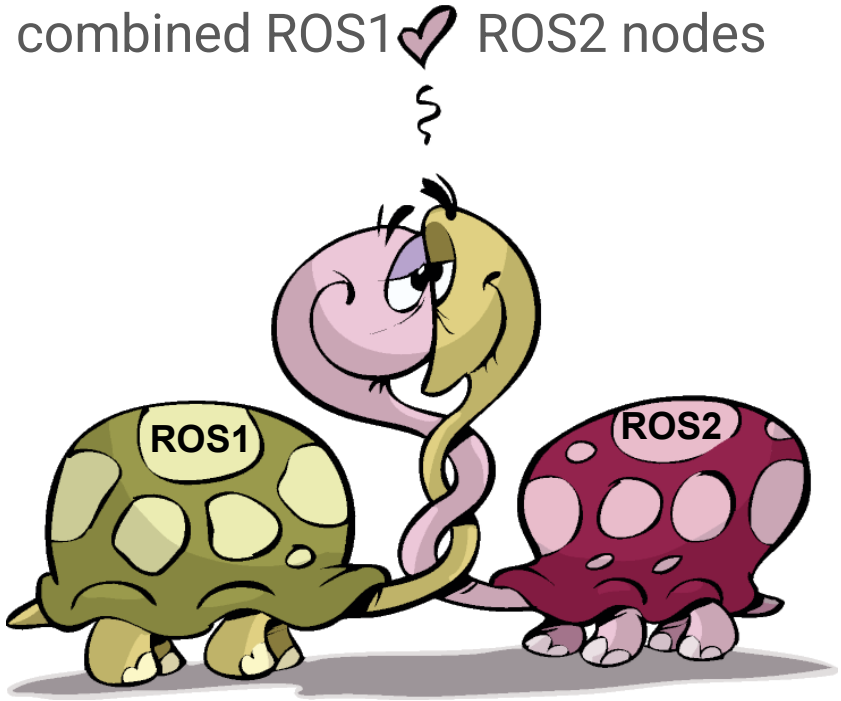


ROS2 Conversion Strategies

1. ~~Everything at once~~
2. ~~Node by node~~ : Cannot break graph in order to bridge fewer topics
3. Topic-by-topic

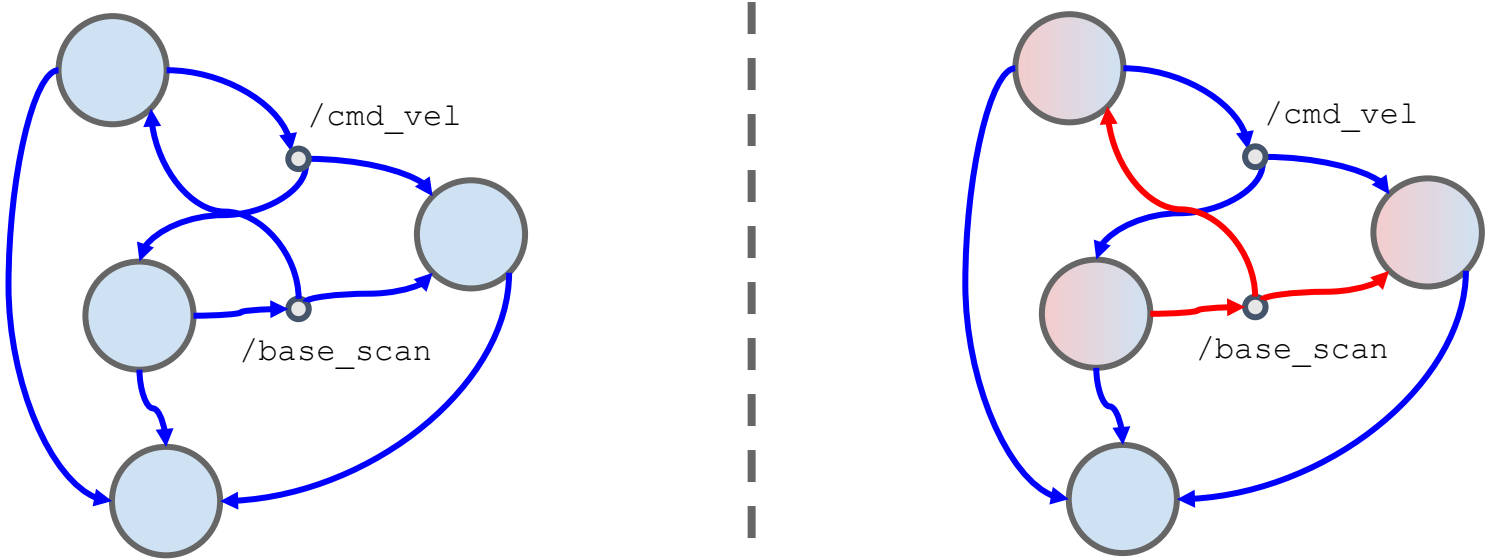
ROS2 Conversion Strategies

1. ~~Everything at once~~
2. ~~Node by node~~
3. Topic-by-topic : Using combined ROS1 ROS2 nodes



Ideal Mixed Node ROS1 -> ROS2 conversion

- Pick a ROS1 topic
- Convert all nodes publishing / subscribing to that topic to use ROS2 instead
- No extra overhead!!
- Easy incremental progress



Running both ROS1 and ROS2 in the same process

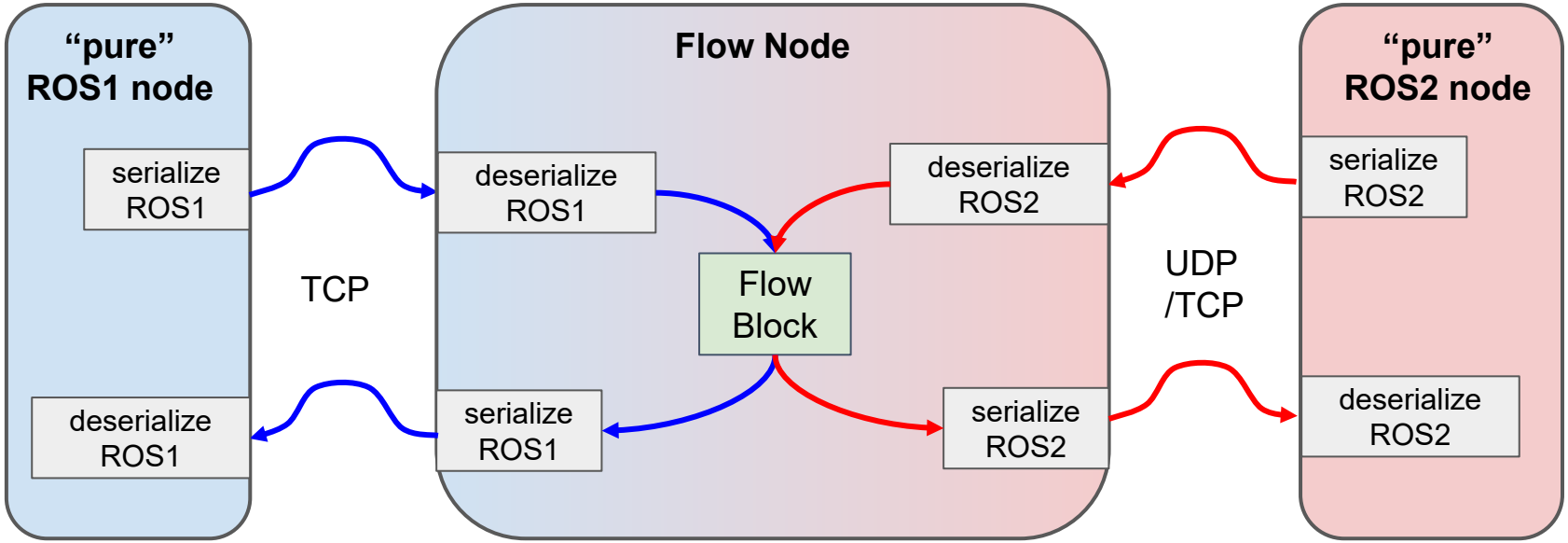
ros1_bridge already does this!!

```
// ROS 1 asynchronous spinner
ros::AsyncSpinner async_spinner(1);
async_spinner.start();

// ROS 2 spinning loop
rclcpp::executors::SingleThreadedExecutor executor;
while (ros1_node.ok() && rclcpp::ok()) {
  executor.spin_node_once(ros2_node);
}
```

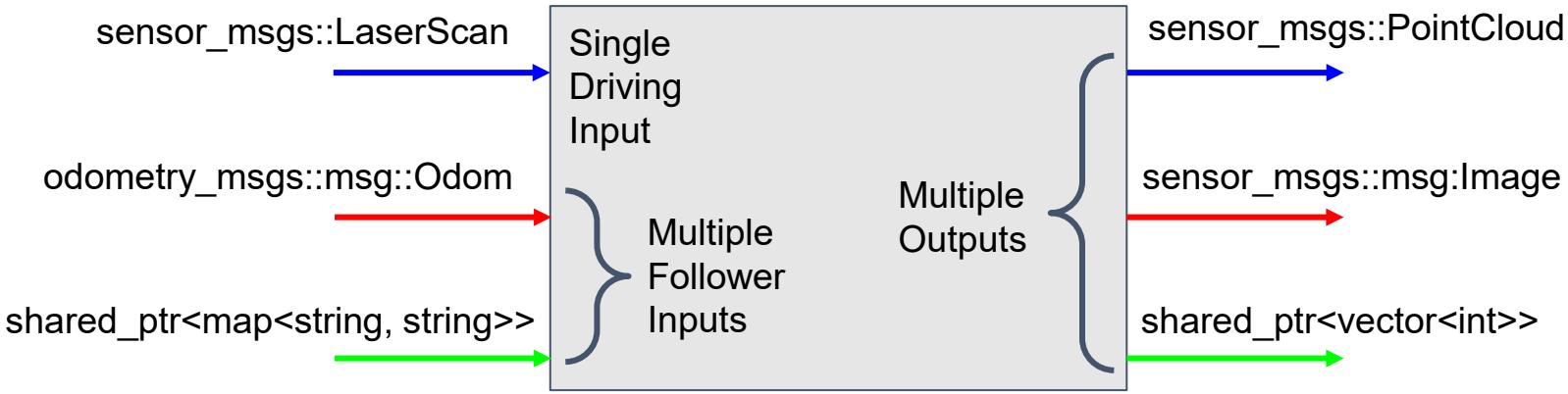
We are doing this in a non-standard way

- Using Bazel for a build system
- Using Flow library instead of ROS pub/sub directly



What is Flow?

- Similar to message_filters
  - deterministic synchronization using message timestamp
- Supports multiple transports
  - ROS1
  - ROS2
  - Local (any C++ objects that wrapped in std::shared_ptr)



What a ROS1 -> ROS2 Topic Change looks like in Flow

1. Find-Replace message type for topic that is being converted
   a. `#include <sensor_msgs/LaserScan.h>` → `#include <sensor_msgs/msg/LaserScan.hpp>`
   b. `sensor_msgs::LaserScan` → `sensor_msgs::msg::LaserScan`
2. Rebuild

**Flow was designed to eventually enable ROS1->ROS2 conversion

ROS2 Conversion Strategies

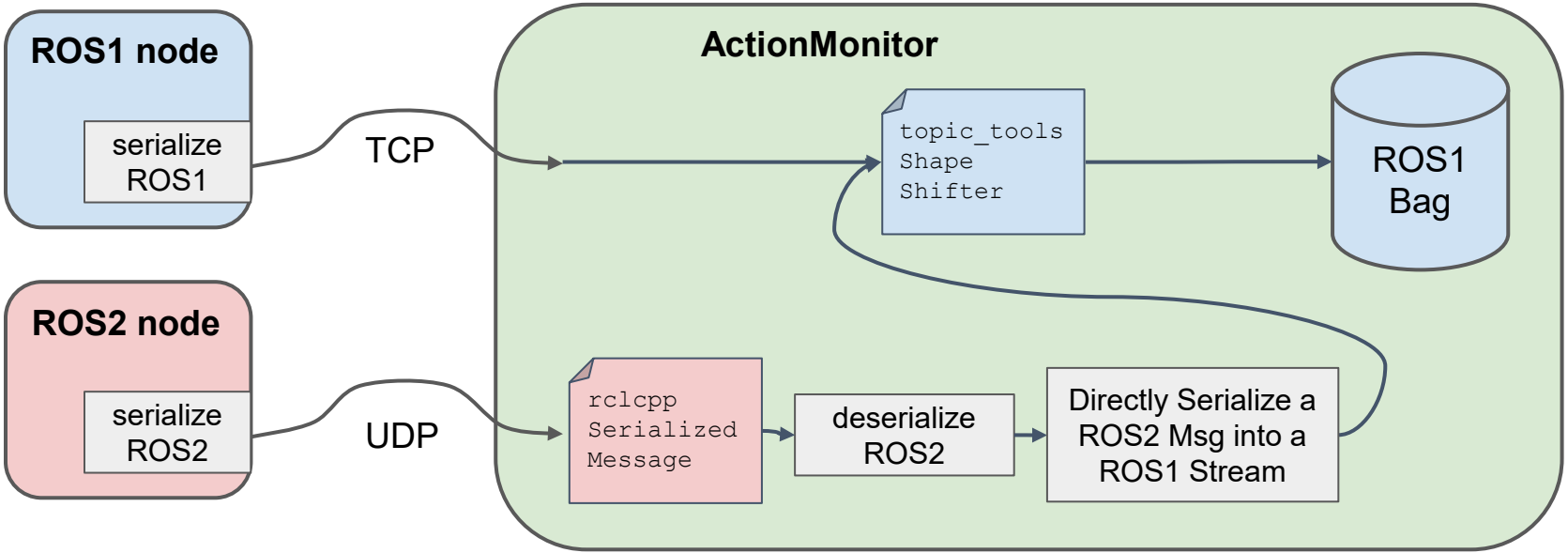
1. ~~Everything at once~~
2. ~~Node by node~~
3. Topic-by-topic
    - Very incremental
    - No extra overhead
    - Very easy with Flow

It Can’t be THAT easy?

Problem	Solution(s)
Legacy nodes that don’t use Flow. →	• Convert to ROS2 using Flow • Can avoid bridge if most other Nodes use Flow
Binary incompatible libraries (ie class_loader) →	• Don’t use ROS2 libraries that use class_loader • Recompile ROS1 with again new version of class_loader?
ROS2 bag format changed →	• See next slide…

ROS2 Messages in ROS1 Bags

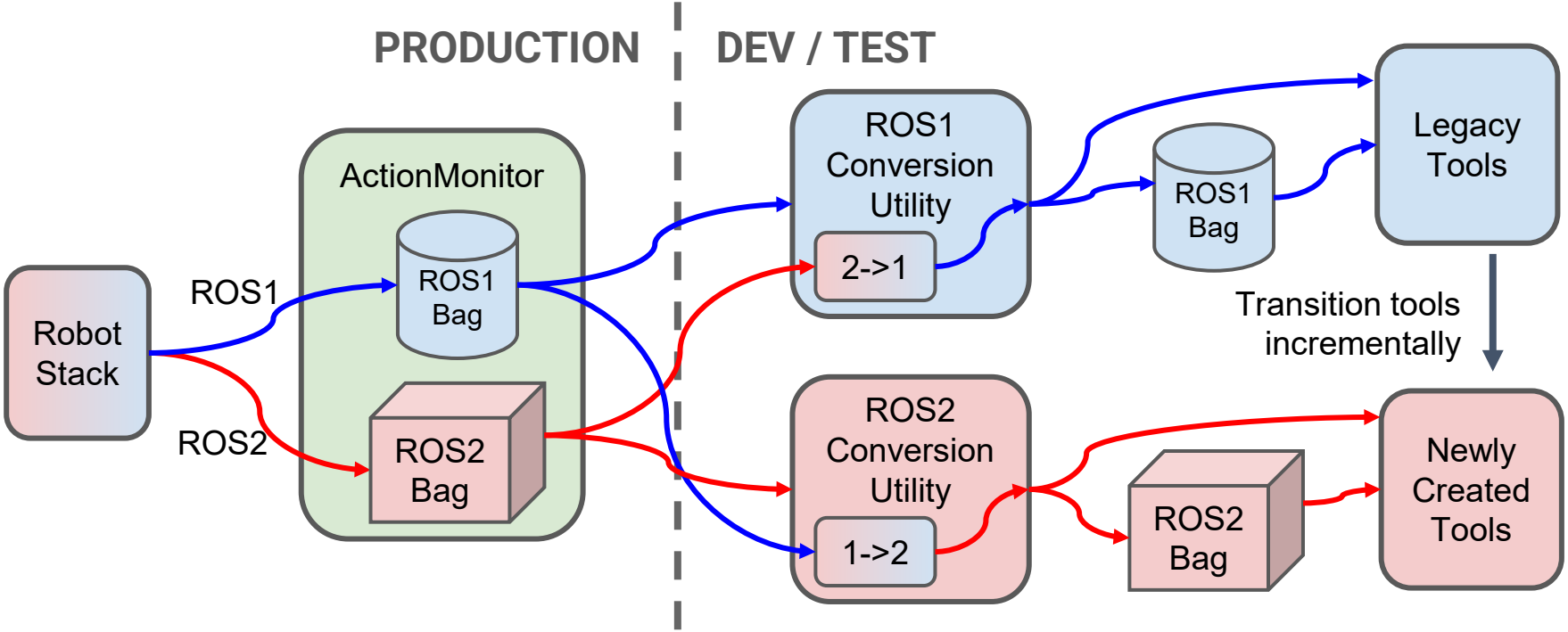
- For now, continue using ROS1 bag format
- Requires a ShapeShifter message with serialized ROS1 data
- Subscribing to a “generic” ROS2 topic –> type is not known at compile time
- PR for ros1_bridge to provide runtime conversion for generic types
- Some of the overhead of using ros1_bridge



Bagging : Longer Term

Record dual bags at once
- ROS1 topics -> ROS1 bag
- ROS2 topics -> ROS2 bag

Create utility to use dual bags
- Convert to ROS1 for legacy tools
- Convert to ROS2 for new tools



Summary

- Using combined ROS1/ROS2 nodes
  - Incremental conversion with no overhead
  - Use Flow and Bazel to create these combined nodes
- Initially continue using ROS1 bag format
  - Later use dual bags to provide a transition path for dev tooling
- Improvements to ros1_bridge
  - Direct serialization of ROS2 messages to ROS1 streams
  - Conversion of generic ROS2 SerializedMessage to ROS1 ShapeShifter

One Last Thing …

Fetch / Zebra is Hiring!

Visit our booth for more details.

fetch robotics
Now Part of Zebra Technologies

THANK YOU

Links

- ros1_bridge PR to support generic message conversion
- ros1_bridge PR to serialize/deserialize ROS2 messages into ROS1 streams
- ros_drake : ROS2 + Bazel
- Flow : C++14, Header-only library for multi-stream data synchronization