



ROS-Industrial Basic Developer's Training Class

July 2023



Southwest Research Institute





Session 1: ROS Basics



Southwest Research Institute





Outline



- Intro to ROS
- ROS Workspaces & Colcon
- Installing packages (existing)
- Packages (create)
- Nodes
- Messages / Topics





An Introduction to ROS



(Image taken from Willow Garage's "What is ROS?" presentation)





ROS1 and ROS2



- ROS1 has been around since 2008
 - Uses custom TCP/IP middleware
- ROS2 is a ground-up reimaging of ROS
 - Started in 2014
 - Built on DDS, middleware proven in industry
 - Now on 9th named release (Iron)



This class will focus on
ROS2 Humble





ROS1 and ROS2



- Community is currently in transition!
 - Final ROS1 release (Noetic) is out (EOL in 2025)
 - All critical features are now supported in ROS2
- ROS-Industrial will take time to transition
 - Many breaking changes / conceptual differences
 - Vision is industrial robots will become native ROS devices





ROS Versions



ROS 1

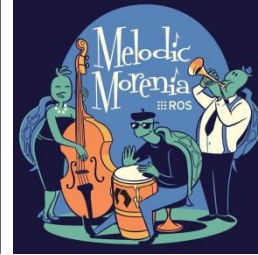


Box Turtle
Mar 2010

...
...



Lunar
2017 - 2019



Melodic
2018 - 2023



Noetic
2020 - 2025



EOL

ROS 2



Ardent
Dec 2018

...
...



Foxy (LTS)
2020 - 2023



Galactic
2021 - 2022

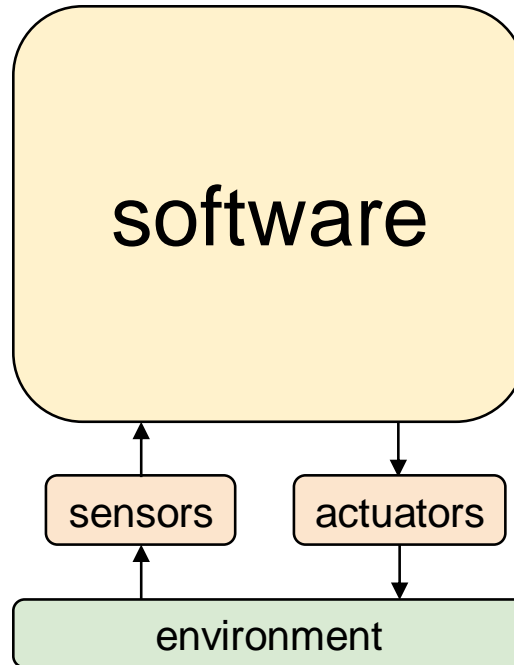


Humble
2022-2027





ROS : The Big Picture



All robots are:

Software connecting Sensors to Actuators
to interact with the Environment

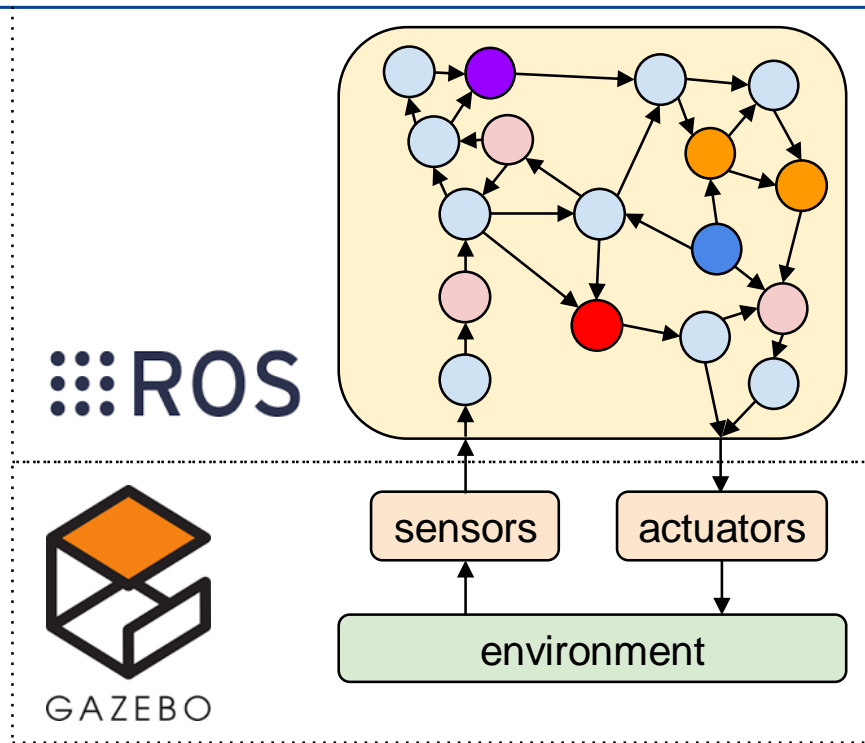


(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)





ROS : The Big Picture



- Break Complex Software into Smaller Pieces
- Provide a framework, tools, and interfaces for distributed development
- Encourage re-use of software pieces
- Easy transition between simulation and hardware

(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)





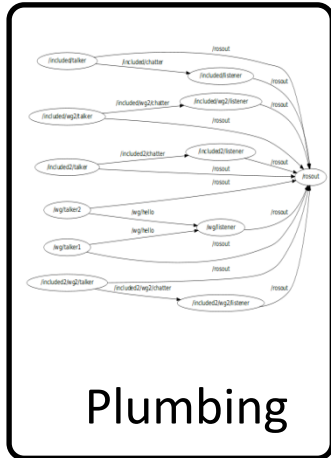
What is ROS?



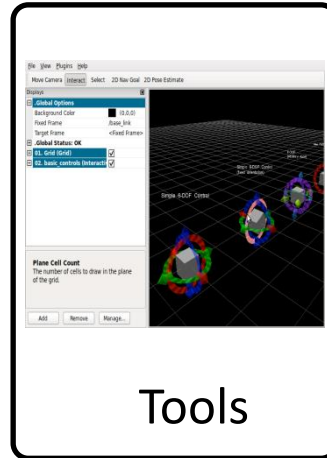
ROS is...



=



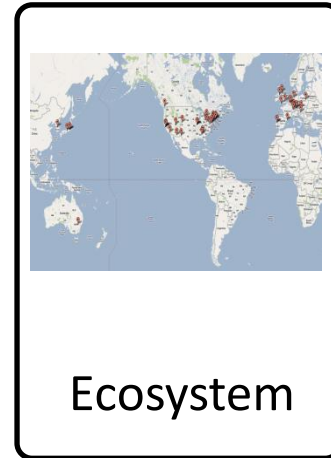
+



+



+

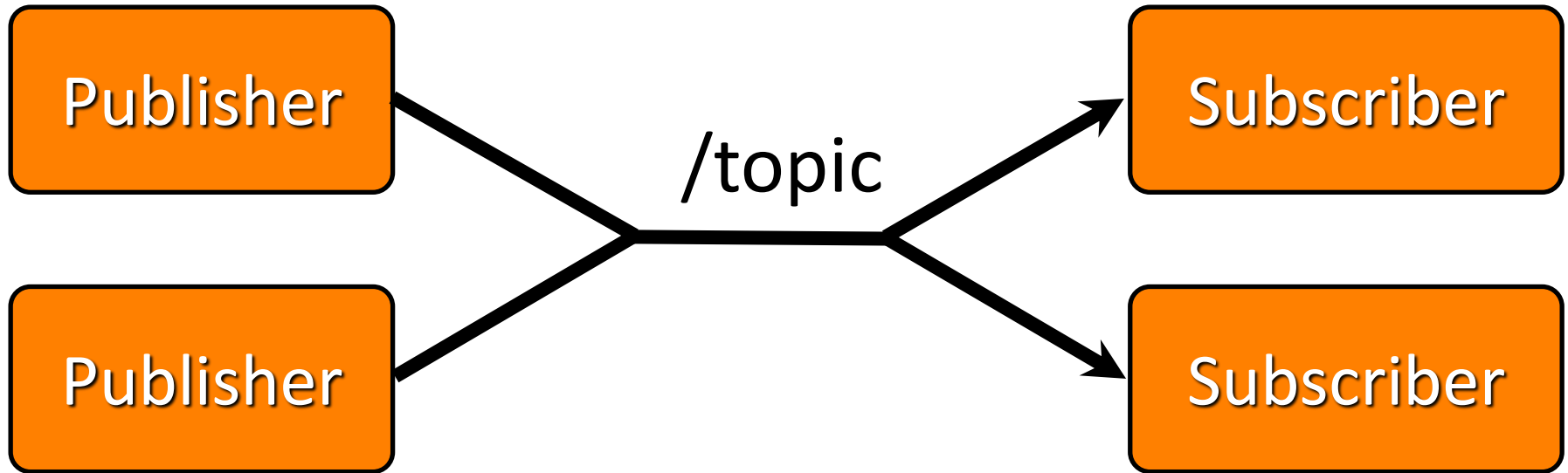


(Adapted from Willow Garage's "What is ROS?" Presentation)





ROS is... plumbing





ROS Plumbing : Drivers



- 2d/3d cameras
- laser scanners
- robot actuators
- inertial units
- audio
- GPS
- joysticks
- etc.

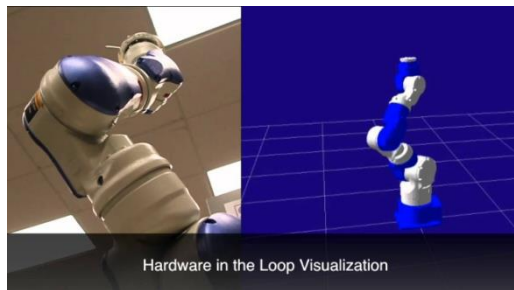
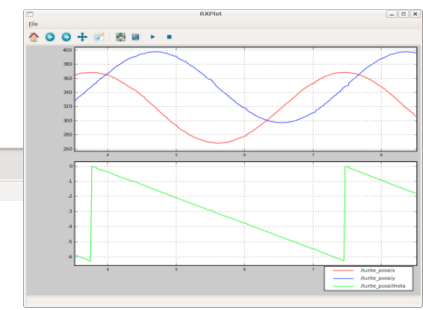
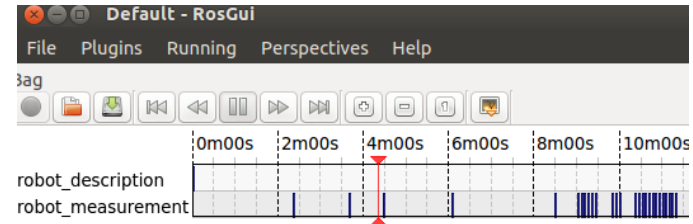
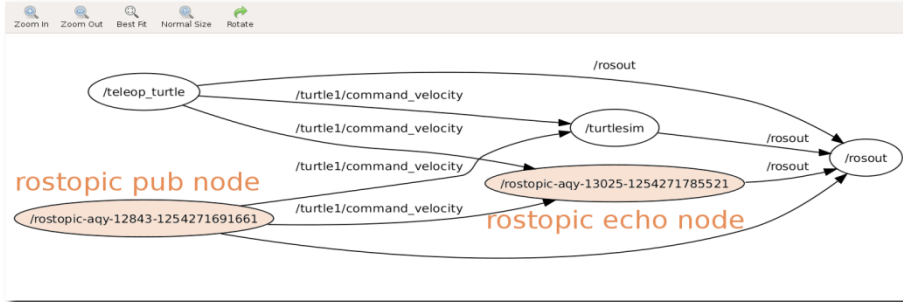


(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)





ROS is ...Tools



- logging/plotting
- graph visualization
- diagnostics
- visualization

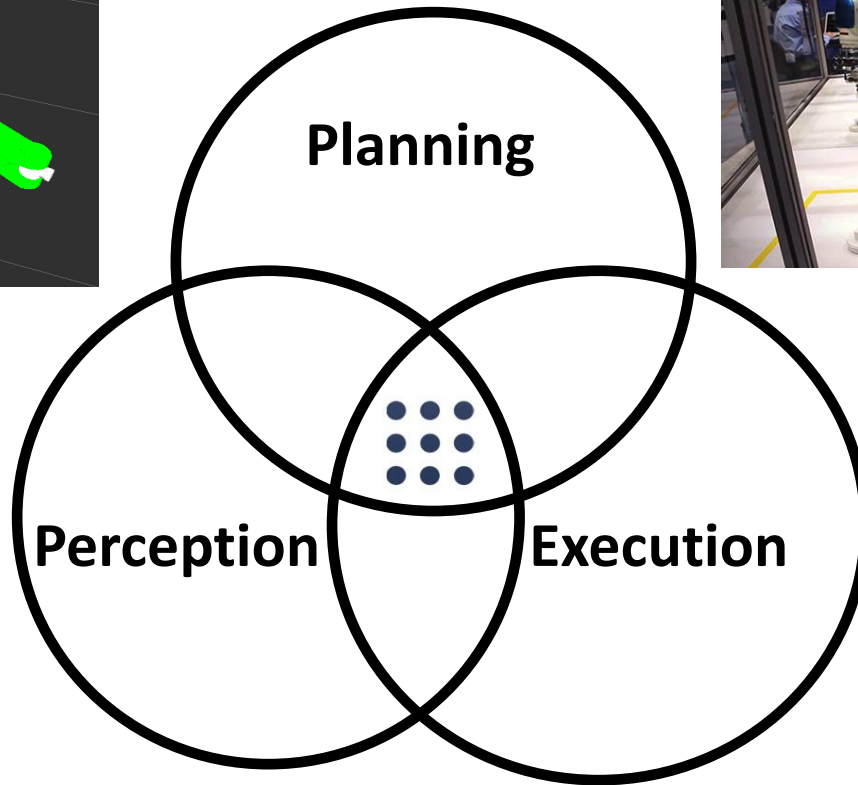
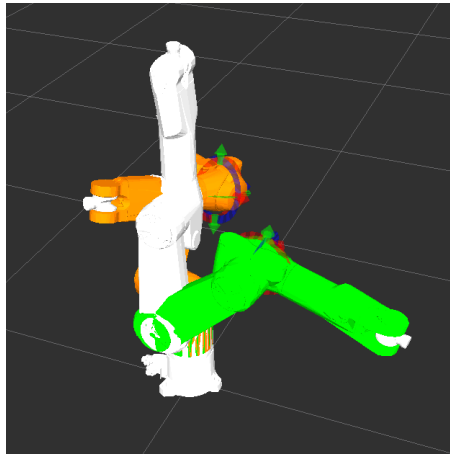
Message	Severity	Topic
#12 The input topic '/narrow_stereo/left/image_raw' is not yet advertised	Warn	/narrow_s
#10 The input topic '/narrow_stereo/right/image_raw' is not yet advertised	Warn	/narrow_s
#11 The input topic '/narrow_stereo/right/camera_info' is not yet advertised	Warn	/narrow_s
#8 The input topic '/narrow_stereo/left/image_raw' is not yet advertised	Warn	/narrow_s
#9 The input topic '/narrow_stereo/left/camera_info' is not yet advertised	Warn	/narrow_s
#7 Holding arms	Info	/arm_hol
#18 The input topic '/wide_stereo/right/camera_info' is not yet advertised	Warn	/wide_st
#16 The input topic '/wide_stereo/left/camera_info' is not yet advertised	Warn	/wide_st
#17 The input topic '/wide_stereo/right/image_raw' is not yet advertised	Warn	/wide_st
#6 The input topic '/wide_stereo/left/image_raw' is not yet advertised	Warn	/wide_st
#5 Moving torso up	Info	/arm_hol

(Adapted from Willow Garage's "What is ROS?" Presentation)





ROS is...Capabilities

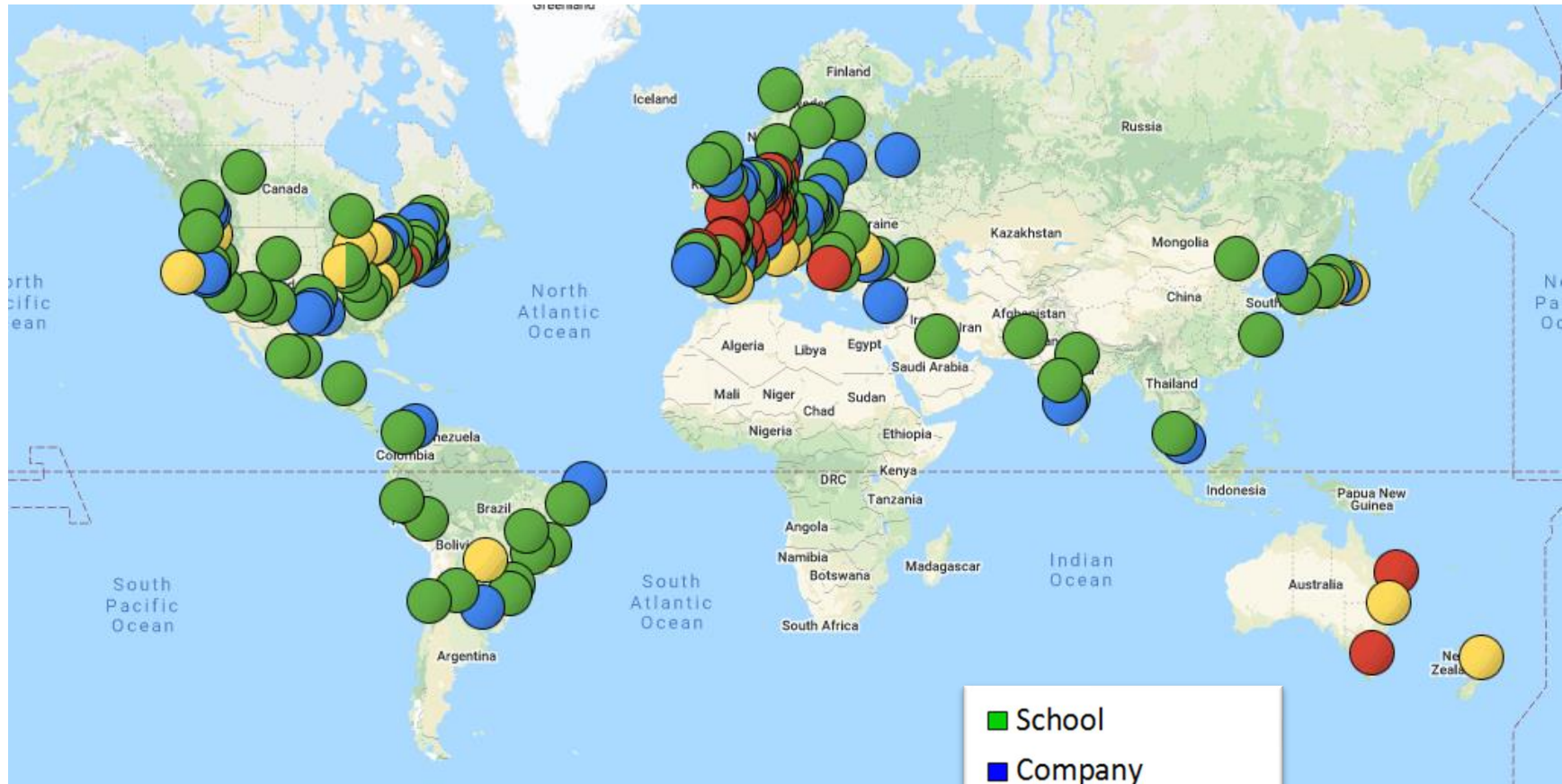


(Adapted from Willow Garage's "What is ROS?" Presentation)





ROS is... an Ecosystem



<http://metrorobots.com/rosmap.html>

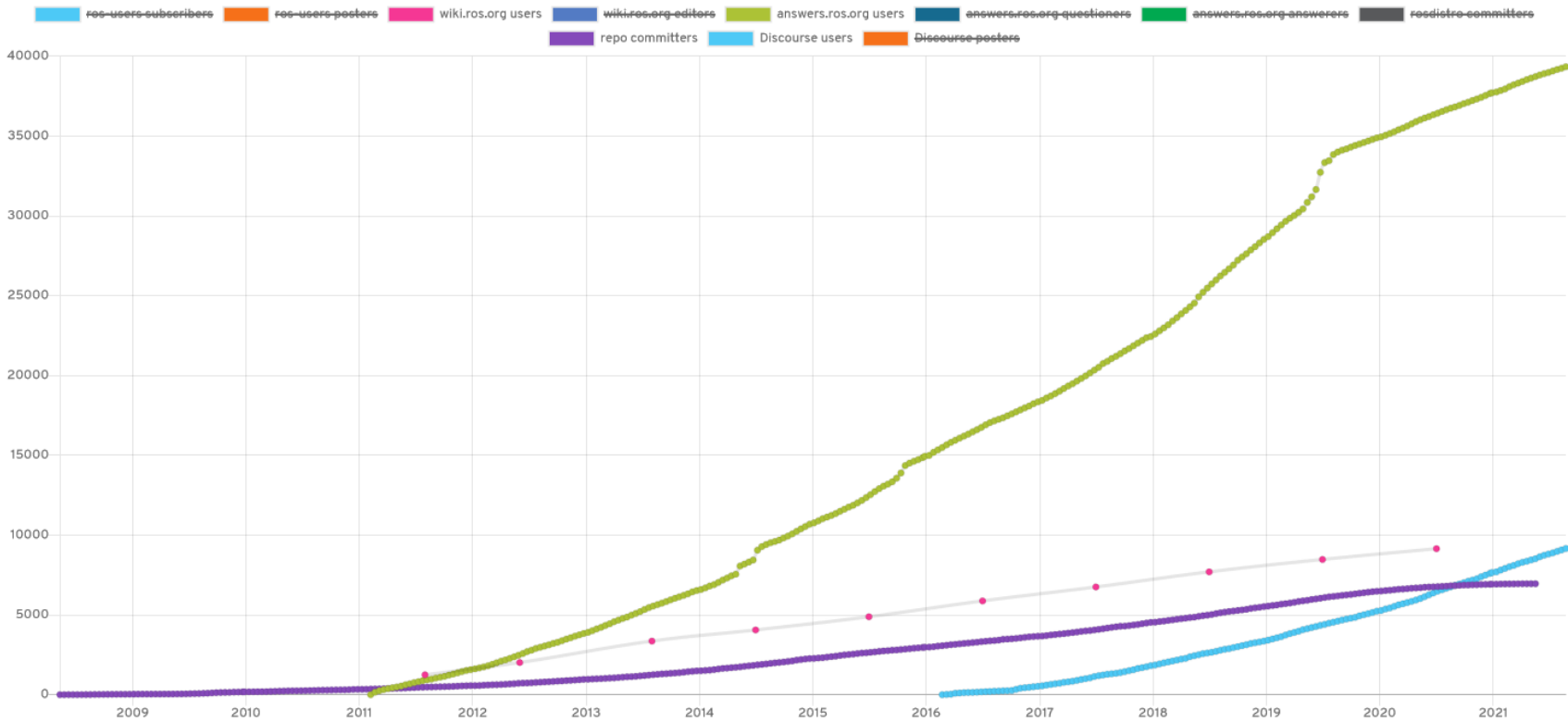




ROS is a growing Ecosystem



Number of ROS Users



A collection of different metrics for measuring the number of users in the ROS community.



<https://metrics.ros.org/>





ROS is International



unique wiki visitors Jul 2020

1.	China	39,080 (19.26%)
2.	United States	31,853 (15.70%)
3.	Japan	16,766 (8.26%)
4.	Germany	14,521 (7.16%)
5.	South Korea	12,583 (6.20%)
6.	India	10,700 (5.27%)
7.	Taiwan	5,904 (2.91%)
8.	United Kingdom	4,150 (2.05%)
9.	France	3,994 (1.97%)
10.	Singapore	3,881 (1.91%)
11.	Canada	3,748 (1.85%)
12.	Italy	3,590 (1.77%)
13.	Hong Kong	3,509 (1.73%)
14.	Spain	2,936 (1.45%)
15.	Russia	2,820 (1.39%)

visitors per million people

1. Singapore: 683
2. Hong Kong: 475
3. Taiwan: 252
4. South Korea: 244
5. Germany: 175
- ...
9. USA: 96

(<http://wiki.ros.org/Metrics> "Community Metrics Report" August 2020)

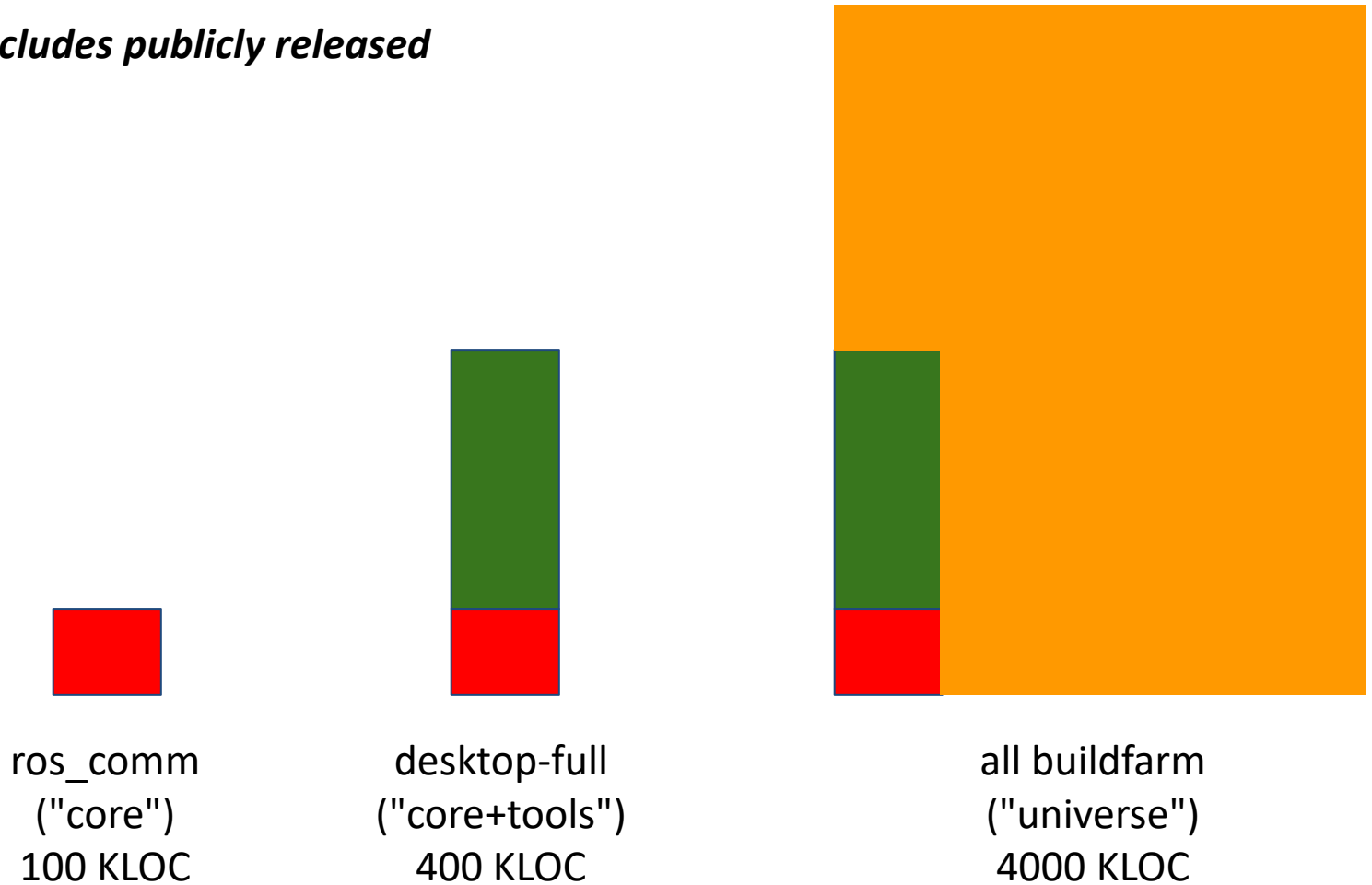




ROS is a Repository



only includes publicly released code!



(From Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics")





ROS Programming

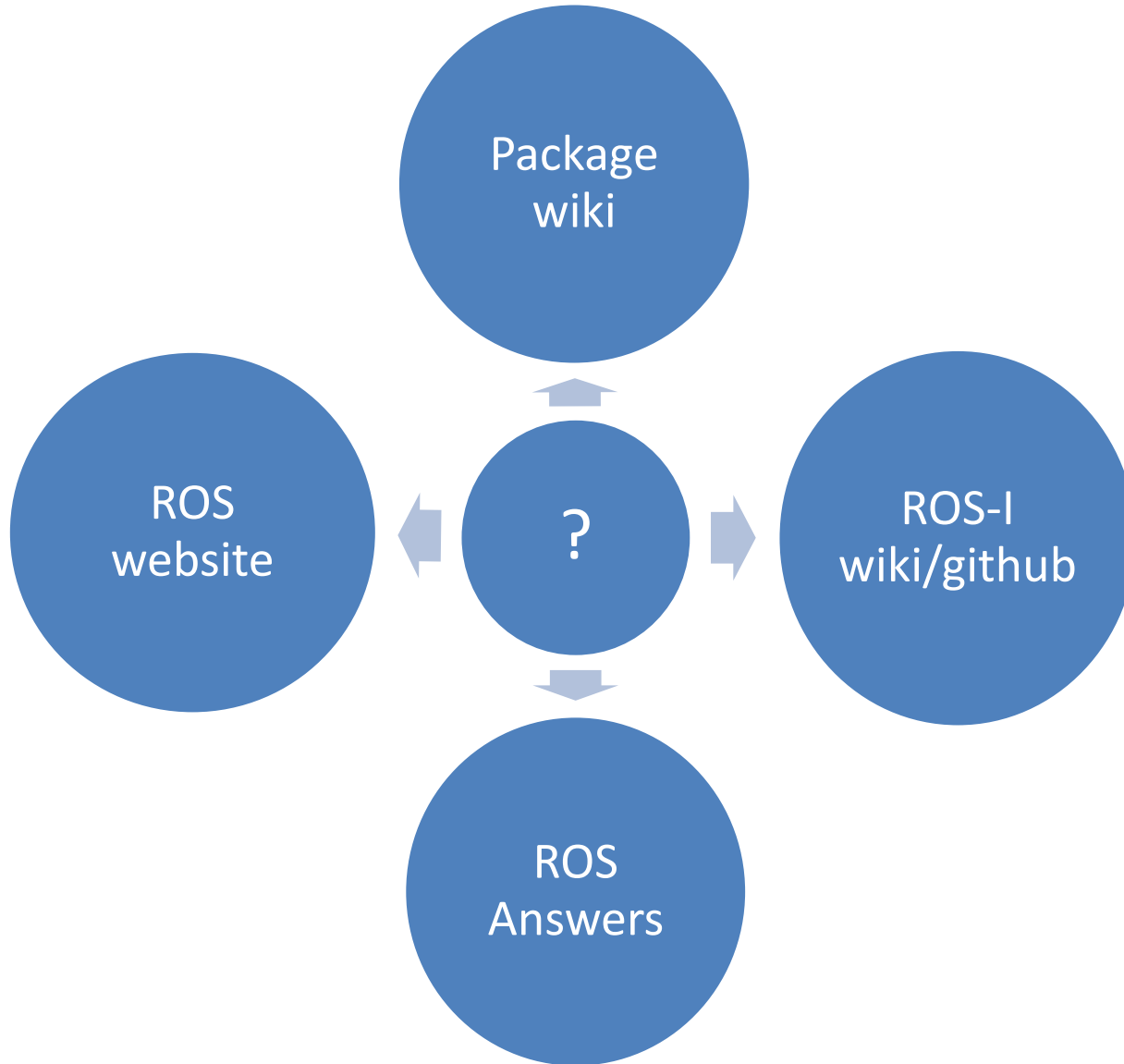


- ROS uses **platform-agnostic** methods for most communication
 - DDS, TCP/IP Sockets, XML, etc.
- Can intermix programming languages
 - Current 1st Tier support: C, C++, Python
 - We will be using C++ for our exercises





ROS Resources





ROS.org Website



http://ros.org

ROS

About Why ROS? Getting Started Get Involved Blog

What is ROS?
 The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

[Read More](#)

ROS Foxy Fitzroy
 Foxy Fitzroy is the latest ROS 2 LTS release. It's supported on Ubuntu Focal, macOS and Windows 10. Get Foxy Fitzroy now!

[Download](#)

ROS Noetic Ninjemys
 ROS Noetic Ninjemys is latest ROS 1 LTS Release targeted at the Ubuntu 20.04 (Focal) release, though other systems are supported to varying degrees.

[Download](#)

Wiki
Find tutorials and learn more

ROS Answers
Ask questions. Get answers

Forums
Hear the latest discussions

ROS1
but still relevant

- Install Instructions
- ROS Answers
- Forums (Discourse)





ROS2 Documentation



<http://docs.ros.org>

- Install
- Tutorials
- Concepts
- APIs

ROS 2 Documentation: Humble

HUMBLE HAWKSBILL

Search docs

- Installation
- Distributions
- Tutorials
- How-to Guides
- Concepts
- Contact
- The ROS 2 Project
- API Documentation
- Related Projects
- Glossary
- Citations

» ROS 2 Documentation

[Edit on GitHub](#)

You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at [Iron](#).

ROS 2 Documentation

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers and state-of-the-art algorithms to powerful developer tools, ROS has the open source tools you need for your next robotics project.

Since ROS was started in 2007, a lot has changed in the robotics and ROS community. The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn't.

This site contains the documentation for ROS 2. If you are looking for ROS 1 documentation, check out the [ROS wiki](#).

If you use ROS 2 in your work, please see [Citations](#) to cite ROS 2.

Getting started

- Installation
 - Instructions to set up ROS 2 for the first time
- Tutorials
 - The best place to start for new users!
 - Hands-on sample projects that help you build a progression of necessary skills
- How-to Guides
 - Quick answers to your "How do I...?" questions without working through the Tutorials
- Concepts
 - High-level explanations of core ROS 2 concepts covered in the Tutorials
- Contact
 - Answers to your questions or a forum to start a discussion





ROS Package Index



http://index.ros.org

ROS Index beta

ABOUT INDEX ← DOC ← CONTRIBUTE STATS Search ROS

Home > ROS 2 Overview

Index
ROS 2 Overview

ROS 2 Documentation

[EDIT ON GITHUB](#)

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

Since ROS was started in 2007, a lot has changed in the robotics and ROS community. The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn't.

Here you will find the official documentation on **ROS 2**, the newest version of ROS.

If you're looking for documentation on ROS 1 (i.e., ROS as it has existed for several years, and what you might be using right now), check the [ROS wiki](#).

Where to start

Newcomers and experienced ROS users should consult this overview of our user-centric content to find what they're looking for.

- Installation pages will help you setup ROS 2 for the first time. You can choose your platform as well as the installation type and distribution that suits your needs.
- Tutorials walk you through learning ROS 2, whether you're learning from scratch or looking for

FOXY ELOQUENT DASHING NOETIC MELODIC

rclcpp package from rclcpp repo

[rclcpp](#) [rclcpp_action](#) [rclcpp_components](#) [rclcpp_lifecycle](#)

GITHUB-ROS2-RCLCPP

Overview **0** Assets **20** Dependencies **0** Tutorials

Package Summary

Tags *No category tags.*

Version 0.8.4

License Apache License 2.0

Build type [AMENT_CMAKE](#)

Use [RECOMMENDED](#)

Repository Summary

Checkout URI <https://github.com/ros2/rclcpp.git>

VCS Type [git](#)

VCS Version [eloquent](#)

Last Updated [2020-06-26](#)

Dev Status [MAINTAINED](#)

CI status [No Continuous Integration](#)

Package Description

The ROS client library in C++.

Additional Links

No additional links.

Maintainers

Dirk Thomas

Authors

No additional authors.

- Install Instructions
- Tutorials
- Package Info
- Still NEW – see ROS1 Wiki





Package Wiki



<http://wiki.ros.org/<packageName>>

tf
electric fuerte groovy hydro **indigo** jade Documentation Status

[geometry](#): [angles](#) | [eigen_conversions](#) | [kdl_conversions](#) | [tf](#) | [tf_conversions](#)

Package Summary

✓ Released ✓ Continuous integration ✓ Documented

tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

- Maintainer status: maintained
- Maintainer: Tully Foote <tfoote AT osrfoundation DOT org>
- Author: Tully Foote, Eitan Marder-Eppstein, Wim Meeussen
- License: BSD
- Source: git <https://github.com/ros/geometry.git> (branch: indigo-devel)

Contents

1. What does tf do? Why should I use tf?
2. Paper
3. Tutorials
4. Code API Overview
5. Frequently asked questions
6. Command_line Tools

Package Links

- [Code API](#)
- [Msg/Srv API](#)
- [Tutorials](#)
- [Troubleshooting](#)
- [FAQ](#)
- [Changelog](#)
- [Change List](#)
- [Roadmap](#)
- [Reviews](#)

Dependencies (15)
Used by (275)
Jenkins jobs (7)

7.2 change_notifier

change_notifier listens to /tf and periodically republishes any transforms that have changed by a give /tf_changes topic.

7.2.1 Subscribed Topics

/tf (tf/tfMessage)
Transform tree.

7.2.2 Published Topics

/tf_changes (tf/tfMessage)
Reduced transform tree.

7.2.3 Parameters

~polling_frequency (float, default: 10.0)

Frequency (hz) at which to check for any changes to the transform tree.

~translational_update_distance (float, default: 0.1)

Minimum distance between the origin of two frames for the transform to be considered changed.

~angular_update_distance (float, default: 0.1)

Minimum angle between the rotation of two frames for the transform to be considered changed.

- Description / Usage
- Tutorials
- Code / Msg API
- Source-Code Link
- Bug Reporting

“ROS1 Only”
But still relevant for most packages





ROS Answers



http://answers.ros.org



https://robotics.stackexchange.com

ROS ANSWERS

Hi there! Please sign in help

ALL UNANSWERED search or ask your question ASK YOUR QUESTION

21,783 questions

Sort by » by date by activity by answers by votes RSS

Contributors

How to save static transforms in bag files? (1 answer, 12 views)

pcl 1.72 installation question (9 answers, 9 views)

freect_launch with Kinect (3 answers, 3 views)

Problem using serial write (-1 votes, 14 views)

schunk_svh_driver : Can't locate node svh_controller in package schunk_svh_driver (1 answer, 8 views)

Broken url in tutorial (9 answers, 9 views)

Tags: ROS x6, tf x6, indigo x4, launch x4, ros-industrial x3, turtbot x3, camera x2, images x2, image_transport x2, imageTransport x2

ROS ANSWERS

Hi there! Please sign in help

ALL UNANSWERED search or ask your question ASK YOUR QUESTION

How can I use Motoman stack with ROS Indigo?

motoman indigo ros-industrial

1 have Ubuntu 14.04

answered Aug 26 '14

updated Aug 26 '14

add a comment

1 answer

Sort by » oldest newest most voted

2 1

2015-04-22: updated as `industrial_core` has been released into indigo.

(I'm assuming no prior ROS experience in this answer... well, some experience)

While there hasn't been an official release of the `roscams` packages into indigo, I've had good results building them from source. As far as I know there are no incompatibilities, but you obviously do this at your own risk

You could do something like this (in your current catkin workspace):

```
cd /path/to/your/catkin_ws/
# checkout the desired version of the motoman repository.
# if you'd rather use the development versions, use "-b hydro-devel" or "-b indigo-devel".
git clone -b hydro https://github.com/ros-industrial/motoman.git
# we need to make sure you have all dependencies installed.
# this step should install "industrial_robot_client" for you
cd ..
roscpp install --from-paths src --ignore-src --rosdistro indigo
# now build
catkin_make
```

This should successfully build the Motoman ROS nodes. You'll still have to set up your controller, but those steps are identical to the Hydro version (see `motoman_driver/Tutorials` on the ROS wiki).

Question Tools: Follow, 1 follower, subscribe to see feed

Stats: Asked: Aug 25 '14, Seen: 81 times, Last updated: 12 hours ago

Related questions: Installing ROS industrial on INDIGO, Problem using ur5ur10 with moveit in indigo, Building indigo on 14.10, Universal robots calibration efforts, I can't install open ai on indigo, what Linux version should I install instead, Tabletop or object detector for indigo, urg nodes on ros indigo, Mavros topics not publishing?, ROS indigo install on ubuntu 14.04 failed, Code Blocks and catkin indigo?

- Quick responses to Good Questions
- Search by text or tag
- Don't re-invent the wheel!





ROS is a Community



- No Central “Authority” for Help/Support
 - Many users can provide better (?) support
 - ROS-I Consortium can help fill that need
- Most ROS-code is open-source
 - can be reviewed / improved by everyone
 - we count on **YOU** to help ROS grow!





What is ROS to you?



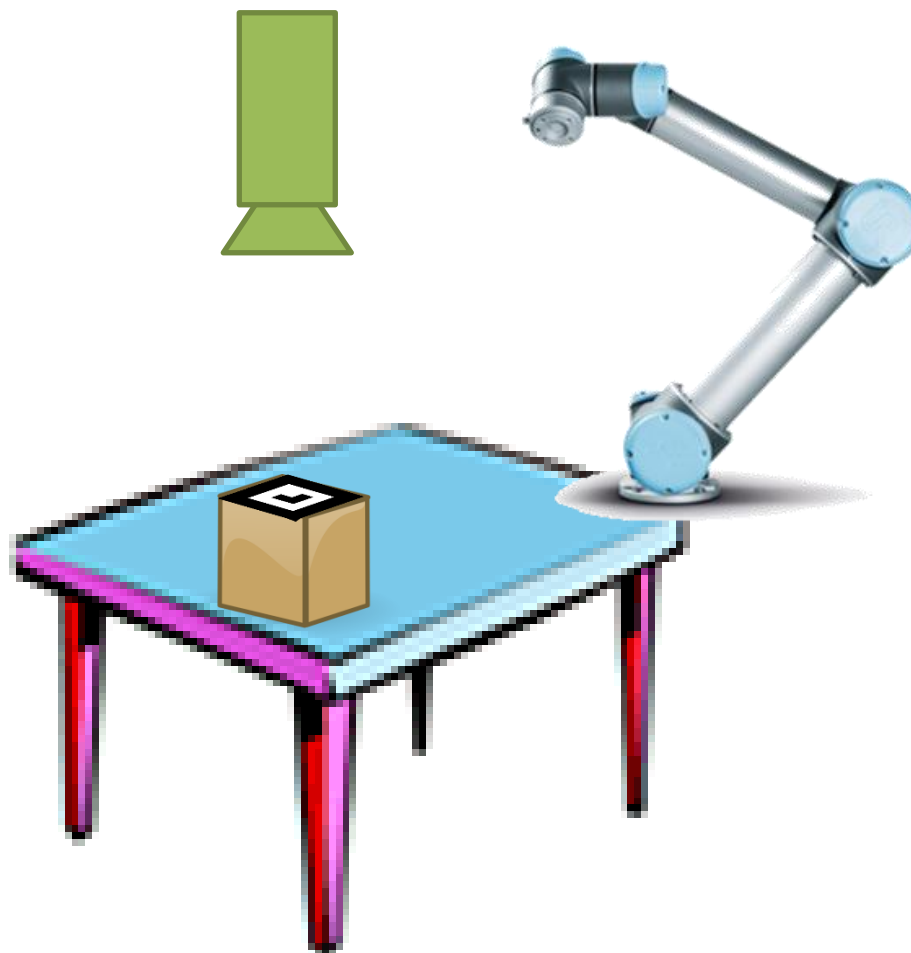
Training Goals:

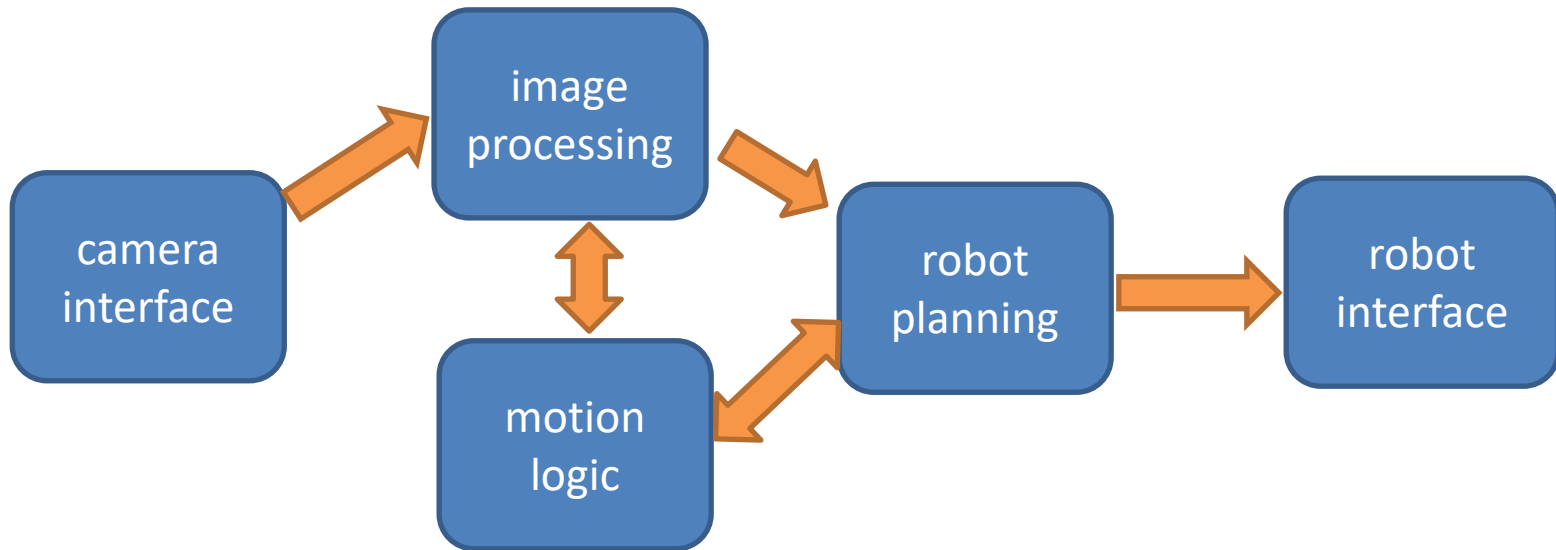
- Show you ROS as a software framework
- Show you ROS as a tool for problem solving
- Apply course concepts to a sample application
- Ask lots of questions and break things.





Scan & Plan "Application"



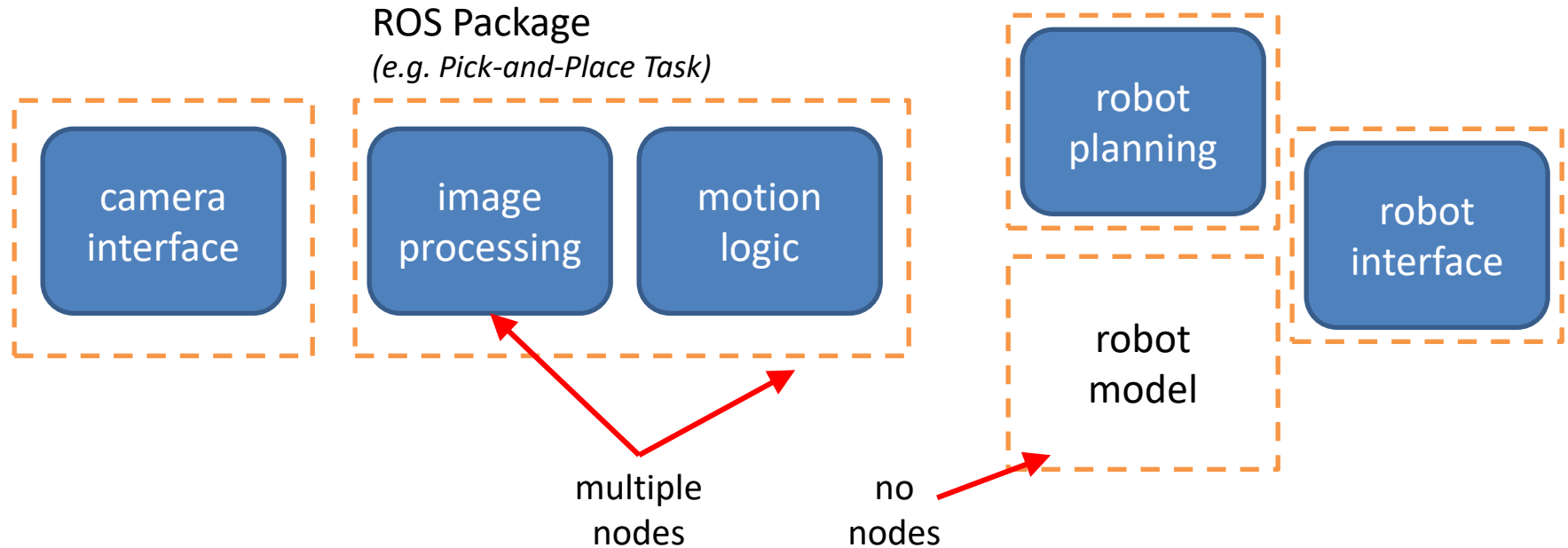


- A **Node** is a *standalone* piece of functionality
 - Most communication happens **between** nodes
 - Nodes can run on many different **devices**
 - Often one node per process, but not always





ROS Architecture: Packages

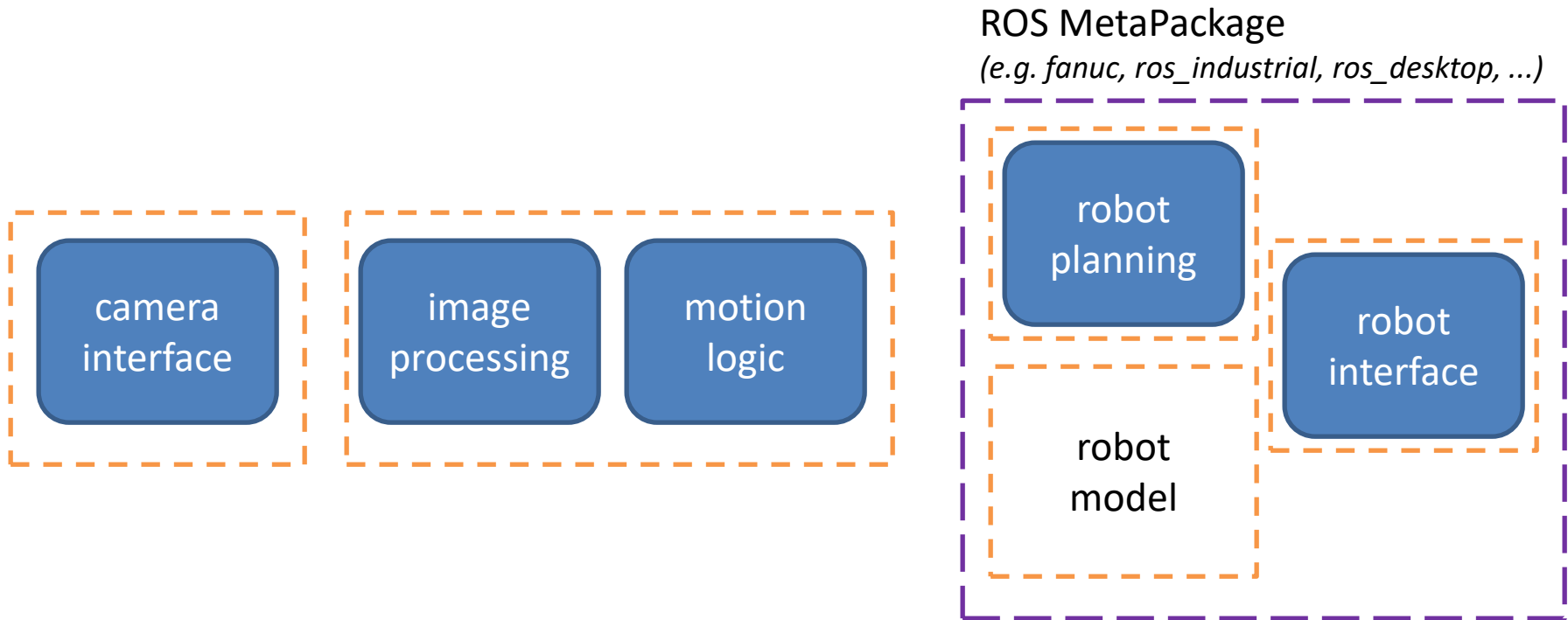


- **ROS Packages** are groups of related nodes/data
 - Files grouped in a single **directory**, with key **metafiles**
 - Many ROS commands are **package-oriented**





ROS Architecture: MetaPkg



- Some “**MetaPackages**” don’t have any content
 - Only dependency references to other packages
 - Mostly for convenient install/deployment

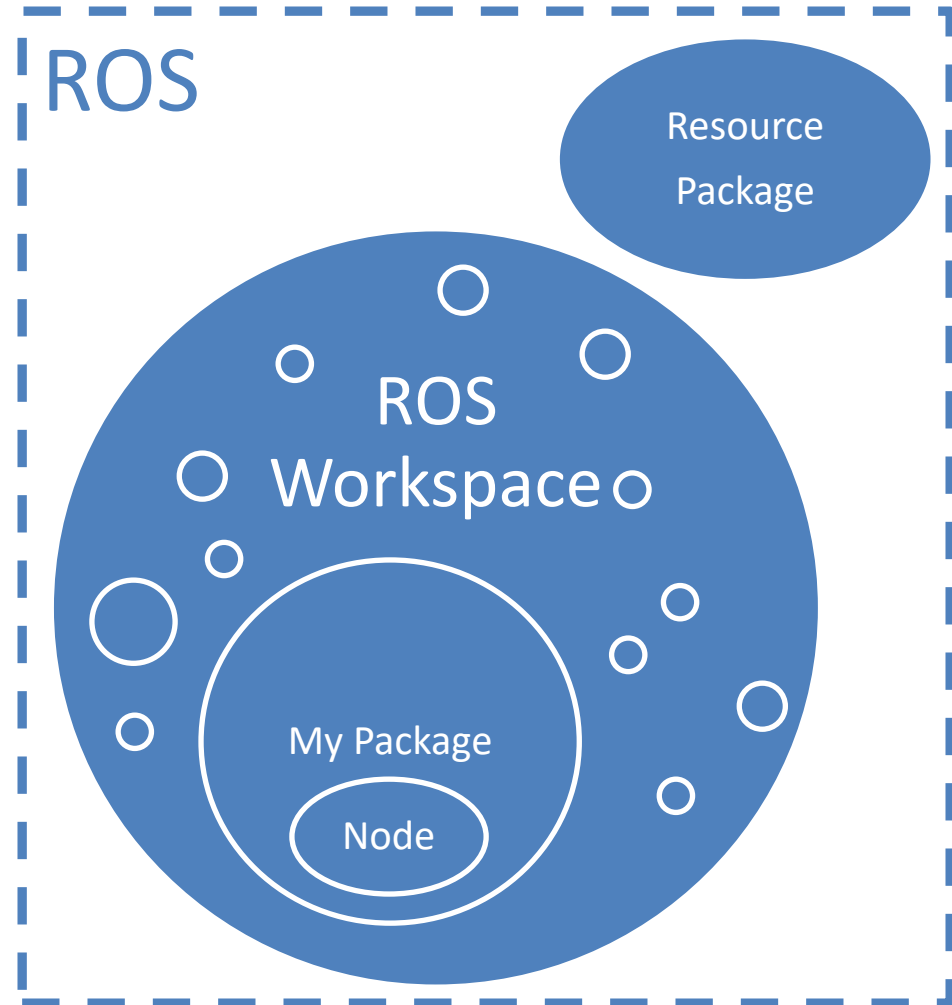




Day 1 Progression



- Install ROS
- Create Workspace
- Add “resources”
- Create Package
- Create Node
 - Basic ROS Node
 - Interact with other nodes
 - Messages
 - Services
- Run Node
 - `ros2 run`
 - `ros2 launch`





Installing ROS





Getting ROS2



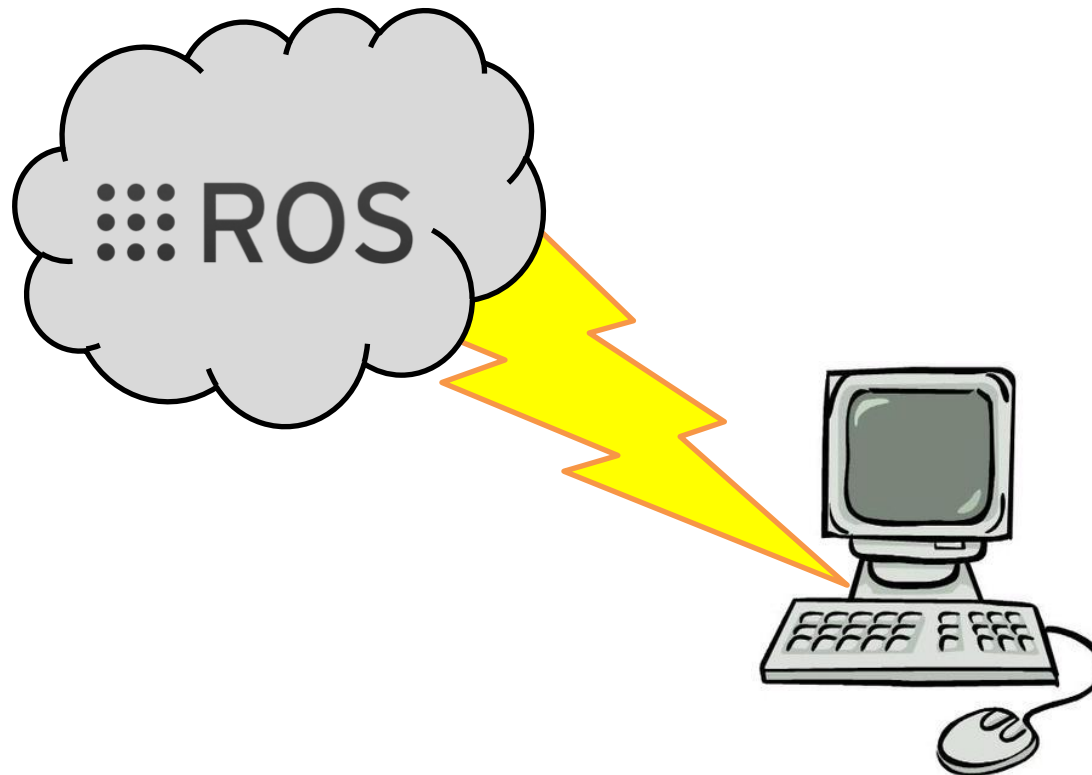
<https://index.ros.org/doc/ros2/Installation/humble/>





Exercise 1.0

Basic ROS Install/Setup

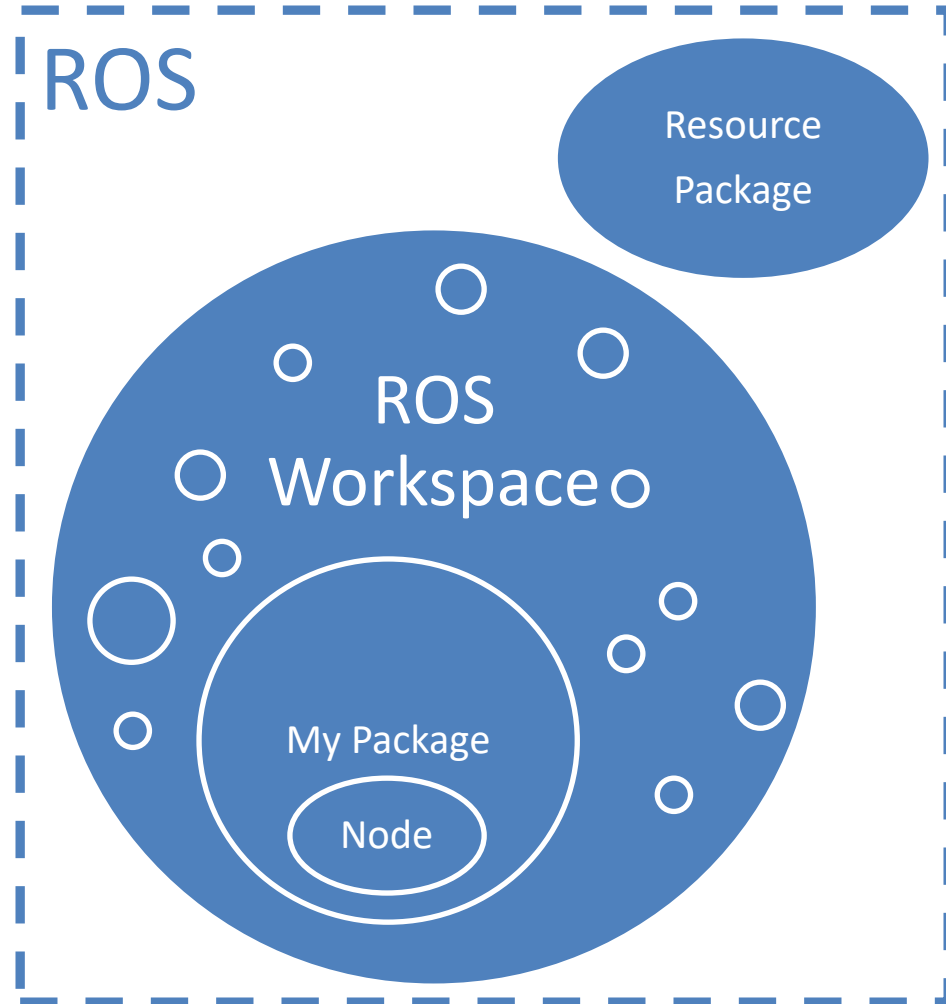




Day 1 Progression



- Install ROS (check install)
- Create Workspace
- Add “resources”
- Create Package
- Create Node
 - Basic ROS Node
 - Interact with other nodes
 - Messages
 - Services
- Run Node
 - ros2 run
 - ros2 launch





Creating a ROS Workspace

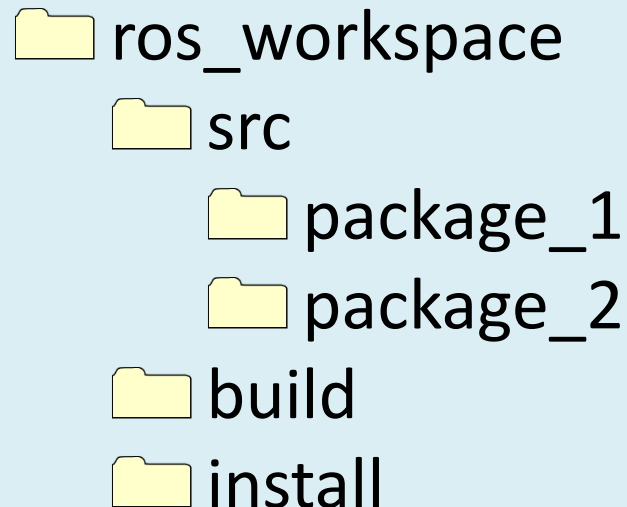




ROS Workspace



- ROS uses a specific directory structure:
 - each “project” typically gets its own **workspace**
 - all packages/source files go in the **src** directory
 - temporary build-files are created in **build**
 - results are placed in **install**





Build System



- ROS2 uses the **ament** build system
 - based on CMake
 - cross-platform (Ubuntu, Windows, embedded...)
 - simplifies depending on packages and exporting outputs to other packages





Build System



- ROS2 also uses the **colcon** build tool
 - Pure Python framework
 - Generates the workspace outputs:
 - Finds all packages in the src directory
 - Defines the build order based on dependencies
 - Invokes the build system for each package
 - CMake/Ament for C++ packages
 - Setuptools for pure Python packages
 - Can build ROS1 packages
 - but some packages may prefer to be built with the ROS1-legacy “catkin” build tools.





Colcon Build Process



Setup (one-time)

1. Create a workspace (arbitrary name and location)
 - `ros_ws`
 - `src` sub-directory must be created manually
 - `build`, `install` directories created automatically
2. Download/create **packages** in `src` subdir

Compile-Time

1. Run `colcon build` from the workspace root
2. Run `source install/setup.bash` to make this workspace visible to ROS





Colcon Build Notes



Colcon Build

- Always run from the workspace root
- Source workspaces of any dependencies before running build.
 - e.g. `source /opt/ros/humble/setup.bash`
- Can chain multiple workspaces together:
 - `base humble -> pcl_ws -> my_ws`
- Don't run from a terminal where you have "sourced" this workspace's setup file (can cause circular issues).
- Best Practice: Use a dedicated terminal window for building.
 - Don't do anything in that terminal window other than colcon build.

Source install/setup.bash

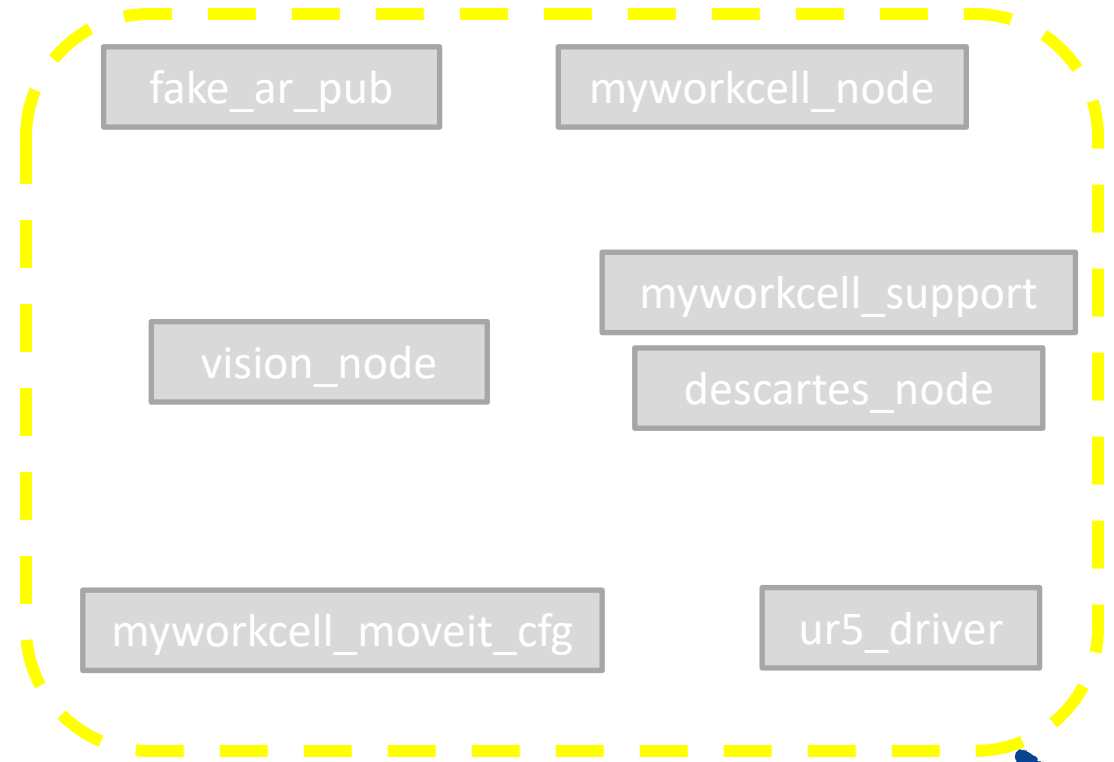
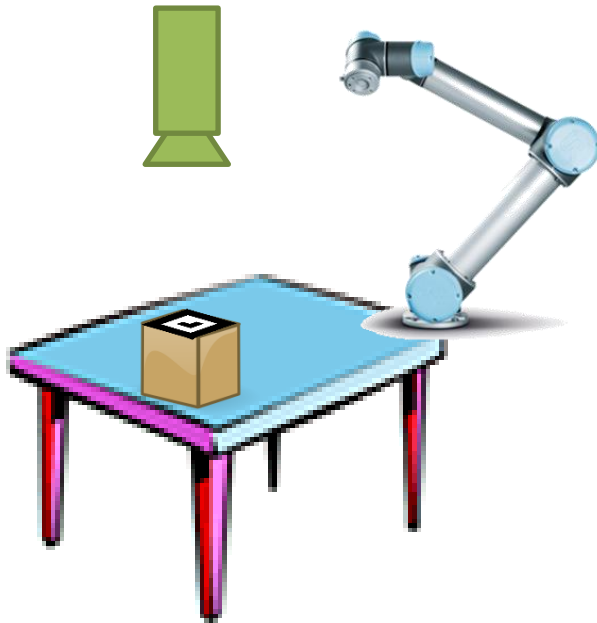
- Remember to source this setup file in EACH new terminal
- No need to also source the underlays' setup files
- May need to re-source after adding new packages
- Can add to `~/.bashrc` to automate this step
 - not recommended if using multiple ROS distros or working on multiple projects in parallel





Exercise 1.1

Create a ROS Workspace

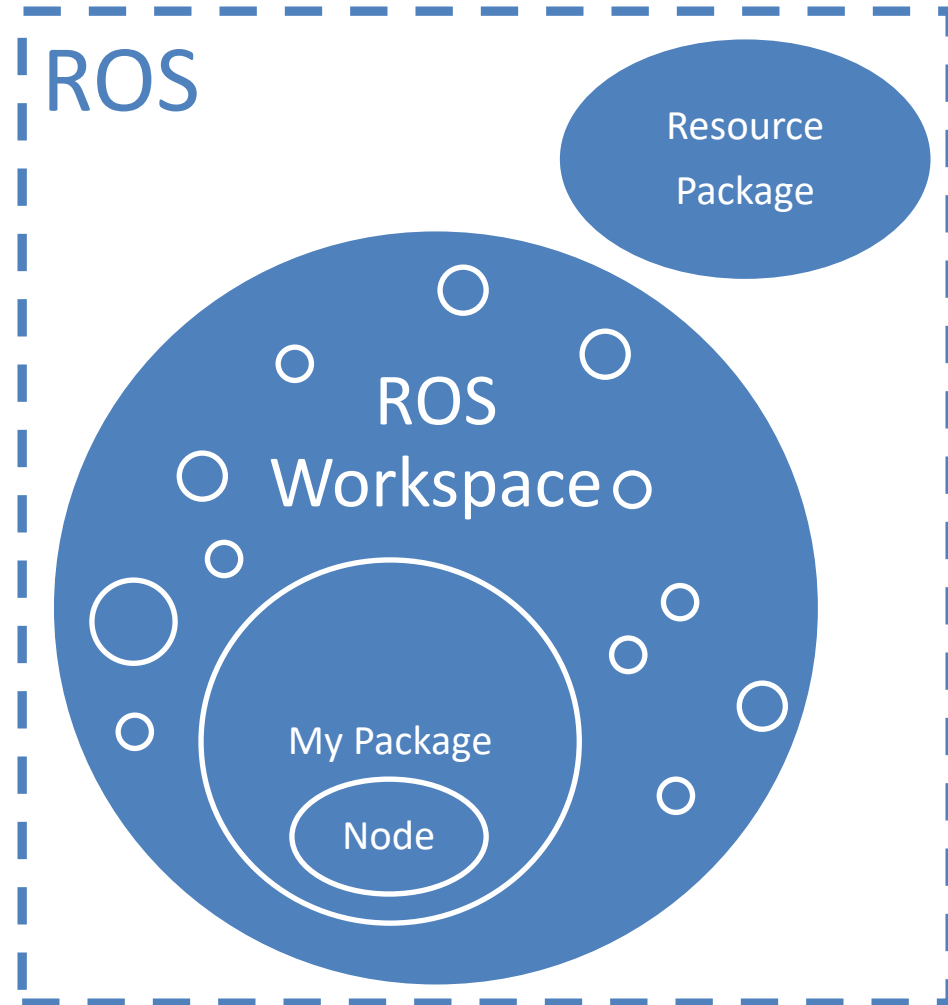




Day 1 Progression



- ✓ Install ROS
- ✓ Create Workspace
- Add “resources”
- Create Package
- Create Node
 - Basic ROS Node
 - Interact with other nodes
 - Messages
 - Services
- Run Node
 - ros2 run
 - ros2 launch





Add 3rd-Party Packages (a.k.a. “Resource” Packages)





Debian Packages

- Nearly “automatic”
- Recommended for end-users
- Stable
- Easy

Source Repositories

- Access “latest” code
- Most at Github.com
- More effort to setup
- Unstable*

Can mix both options, as needed





Finding the Right Package



- ROS Website (<http://index.ros.org>)
 - Search for known packages
- ROS Answers (<http://answers.ros.org>)
 - When in doubt... ask someone!
 - Migrating to <https://robotics.stackexchange.com>





Install using Debian Packages



```
sudo apt install ros-humble-package
```

↑ ↑ ↑ ↑ ↑ ↑

admin manage install all ROS pkgs ROS ROS package
permissions ".deb" new ".deb" start with `ros-` distribution name

Use "-" not "_"

- Fully automatic install:
 - Download .deb package from central ROS repository
 - Copies files to standard locations (`/opt/ros/humble/...`)
 - Also installs any other required dependencies
- `sudo apt-get remove ros-<distro>-<package>`
 - Removes software (but not dependencies!)





Installing from Source



- Find GitHub repo
- Clone repo into your workspace src directory

```
cd ros_ws/src  
git clone http://github.com/user/repo.git
```

- Build your colcon workspace

```
cd ros_ws  
colcon build
```

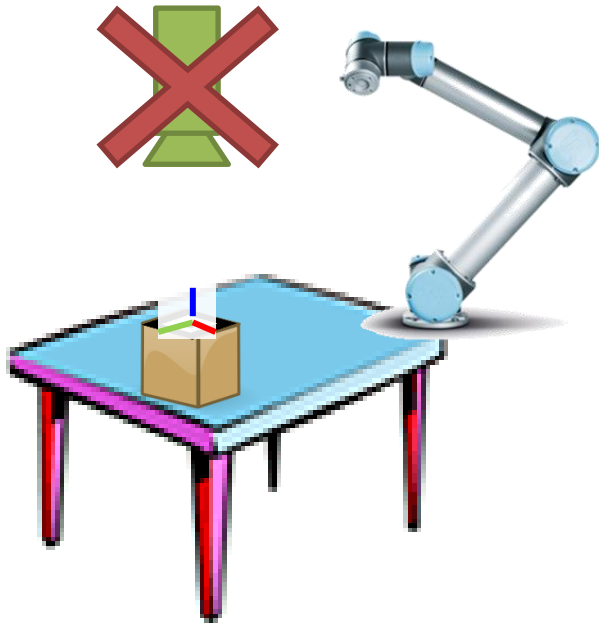
- Now the package and its resources are available to you





Exercise 1.2

Install “resource” packages



fake_ar_pub

myworkcell_node

vision_node

myworkcell_support

descartes_node

myworkcell_moveit_cfg

ur5_driver

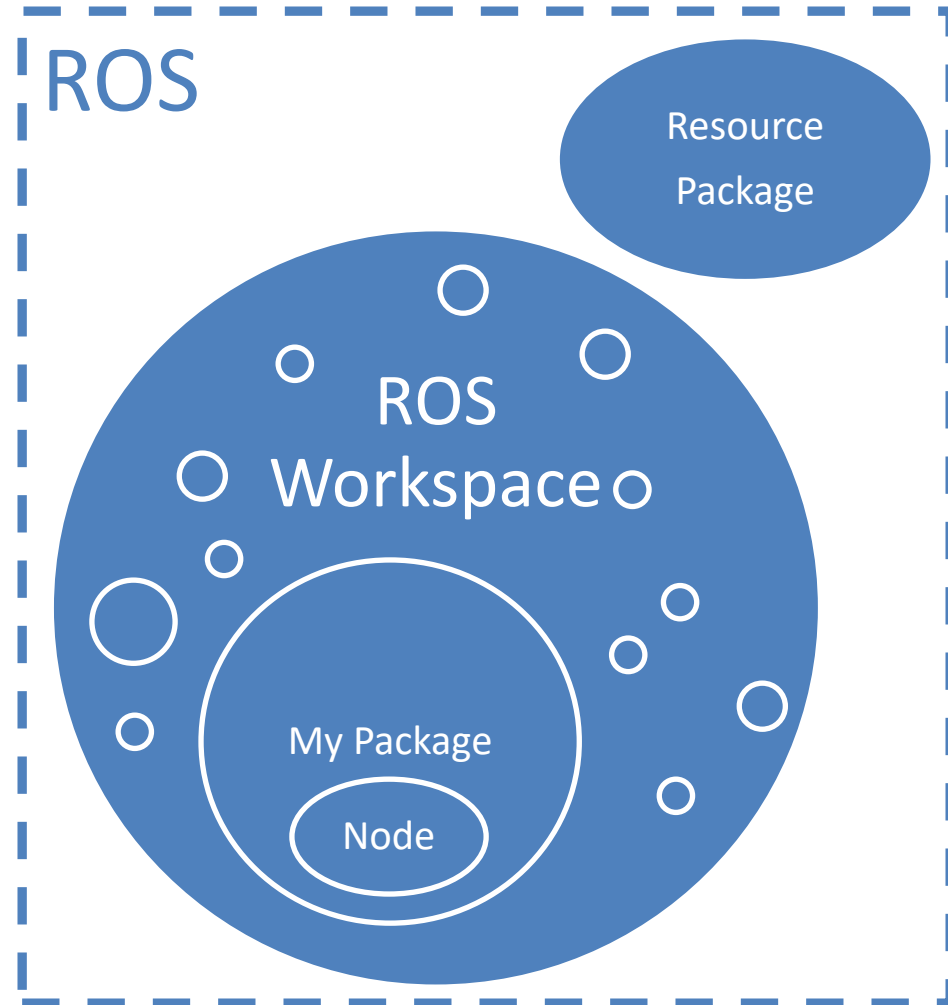




Day 1 Progression



- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- Create Package
- Create Node
 - Basic ROS Node
 - Interact with other nodes
 - Messages
 - Services
- Run Node
 - ros2 run
 - ros2 launch





ROS Packages

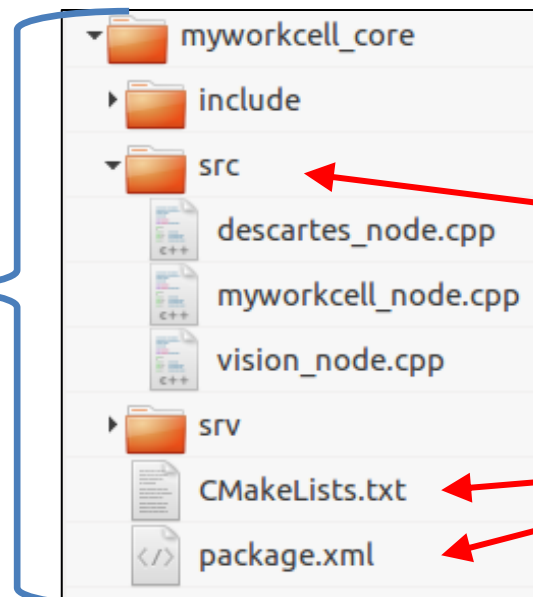
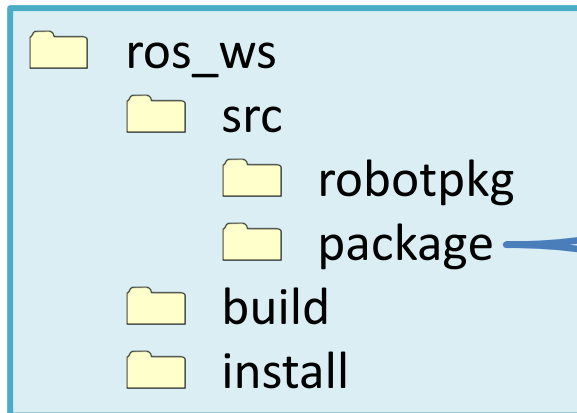




ROS Package Contents



- ROS components are organized into **packages**
- Packages contain several **required files**:
 - `package.xml`
 - **metadata** for ROS: package name, description, dependencies, ...
 - `CMakeLists.txt`
 - **build rules** for ament



package directory

package source-files
(vs. workspace src dir)

required files





package.xml



- Metadata: name, description, author, license ...

```
<?xml version="1.0"?>
<package format="2">
  <name>myworkcell_core</name>
  <version>0.0.0</version>
  <description>The myworkcell_core package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example: -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="ros-industrial@todo.todo">ros-industrial</maintainer>

  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!-- BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>

  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/myworkcell_core</url> -->

  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>message_generation</build_depend>
  <exec_depend>message_runtime</exec_depend>
  <depend>roscpp</depend>
  <depend>geometry_msgs</depend>
</package>
```





- Metadata: name, description, author, license ...
- Dependencies:
 - Common
 - `<buildtool_depend>`: Needed to **build** itself. (Typically *ament_cmake*)
 - `<build_depend>`: Needed to **build** this package.
 - `<exec_depend>`: Needed to **run** code in this package.
 - `<depend>`: Needed to **build, export, and execution** dependency.
 - Uncommon
 - `<build_export_depend>`: Needed to **build against** this package.
 - `<test_depend>`: Only *additional* dependencies for unit tests.
 - `<doc_depend>`: Needed to generate documentation.





- Provides **rules** for **building software**
 - template file contains many examples

```
add_executable(myNode src/myNode.cpp src/widget.cpp)
```

Builds program `myNode`, from `myNode.cpp` and `widget.cpp`

```
ament_target_dependencies(myNode rclcpp std_msgs)
```

Links node `myNode` to dependency headers and libraries

```
install(TARGETS myNode DESTINATION lib/${PROJECT_NAME})
```

Copies nodes/libraries to workspace's "install" directory





ROS Package Commands



- **ros2 pkg**

- `ros2 pkg create package_name`

- Create a new package, including template files*

- Common options (not required, but will help pre-fill templates):*

- `--build-type ament_cmake`

- `--node-name my_node`

- `--dependencies dep_pkg_1 dep_pkg_2`

- `ros2 pkg prefix package_name`

- Show directory where package_name is installed*

- `ros2 pkg list`

- List all ros packages installed (this is a BIG LIST!)*

- `ros2 pkg xml package_name`

- Show the package.xml file of package_name*





Create New Package



```
ros2 pkg create mypkg --node-name mynode  
--dependencies dep1 dep2
```

Easiest way to start a new package

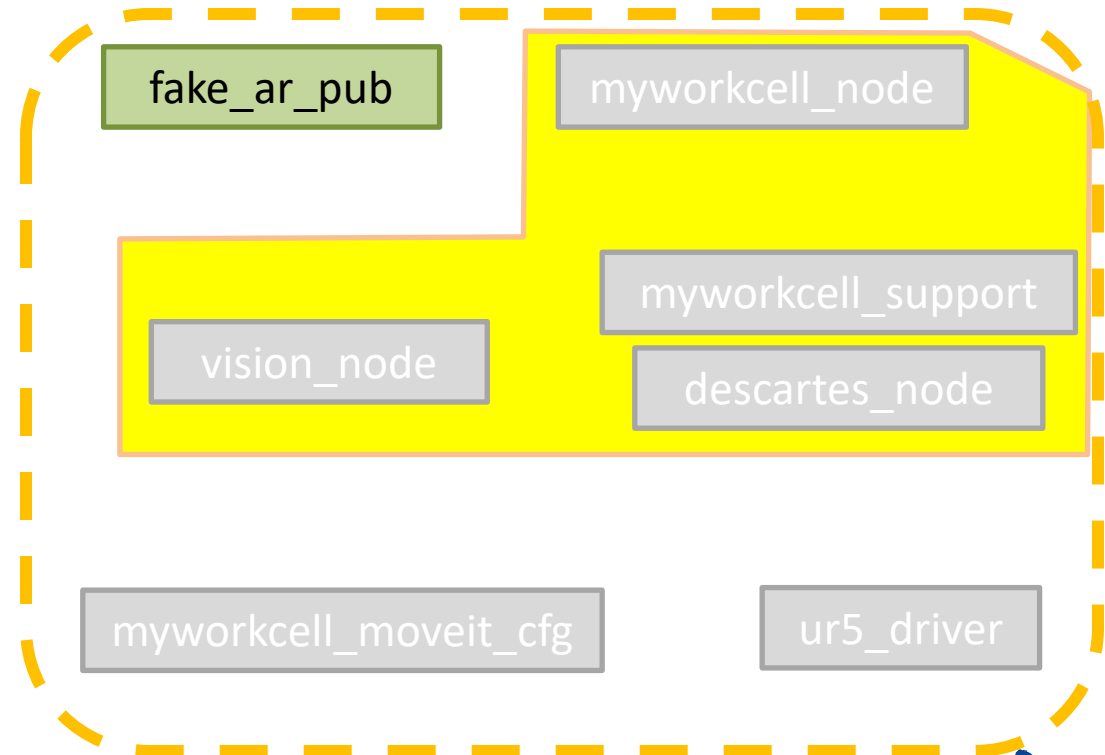
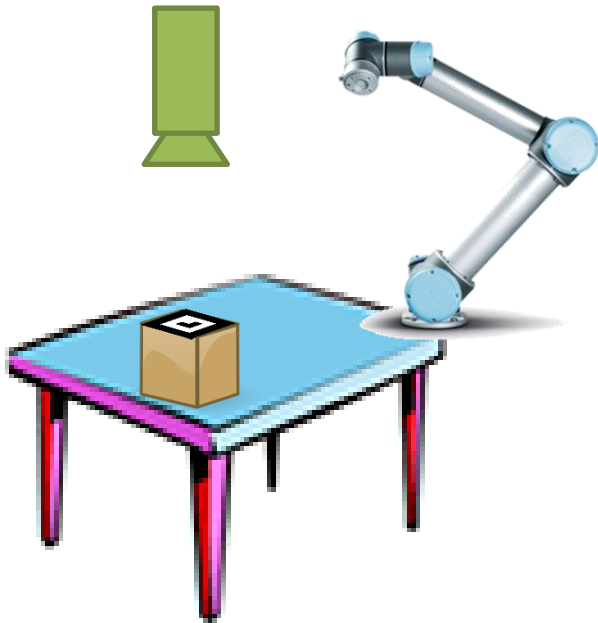
- create directory, required template files
- `mypkg` : name of package to be created
- `mynode` : name of node (main executable)
- `dep1/2` : dependency package names
 - automatically added to `CMakeLists` and `package.xml`
 - can manually add additional dependencies later





Exercise 1.3.1

Create Package

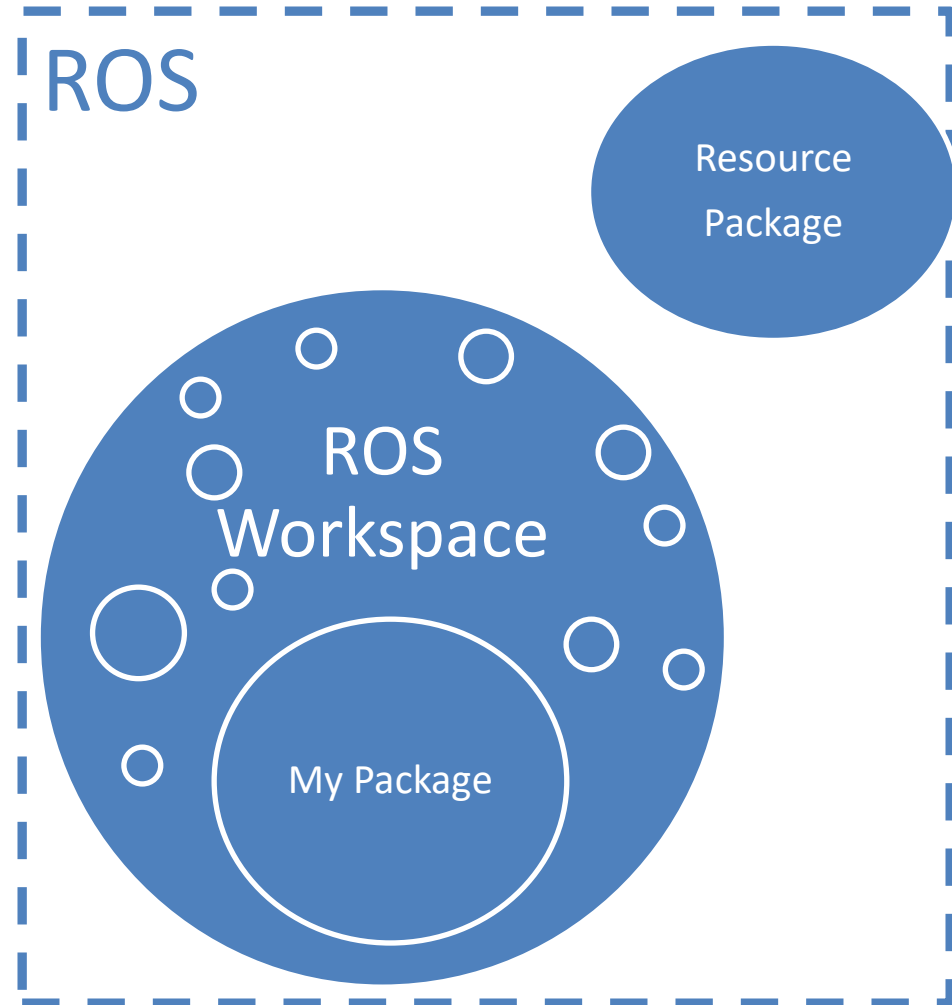




Day 1 Progression



- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ✓ Create Package
- Create Node
 - Basic ROS Node
 - Interact with other nodes
 - Messages
 - Services
- Run Node
 - ros2 run
 - ros2 launch





ROS Nodes





A Simple C++ ROS Node



Simple C++ Program

```
#include <iostream>

int main(int argc, char* argv[])
{

    std::cout << "Hello World!";

    return 0;
}
```

Simple C++ ROS2 Node

```
#include <rclcpp/rclcpp.h>

int main(int argc, char* argv[])
{
    rclcpp::init(argc, argv);
    auto node = make_shared<rclcpp::Node>("hello");

    RCLCPP_INFO(node->get_logger(), "Hello World!");

    return 0;
}
```





ROS2 Node Commands



- `ros2 run package_name node_name`
execute ROS node

- **ros2 node**
 - `ros2 node list`
View running nodes
 - `ros2 node info node_name`
View node details (publishers, subscribers, services, etc.)



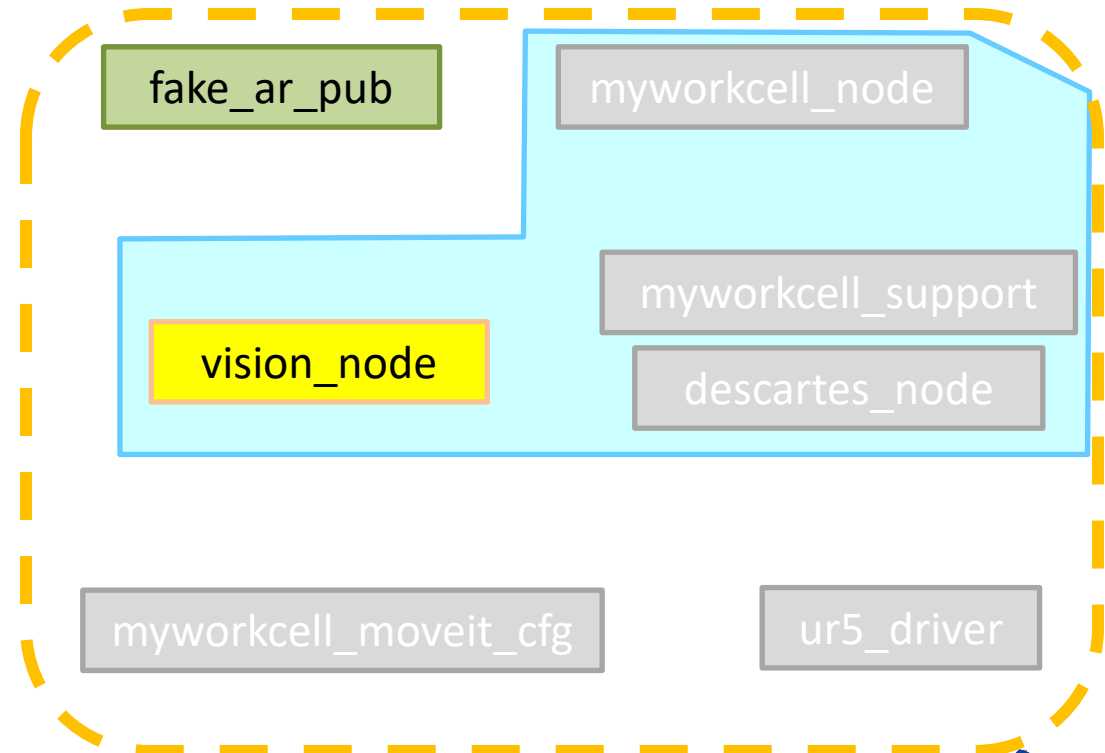
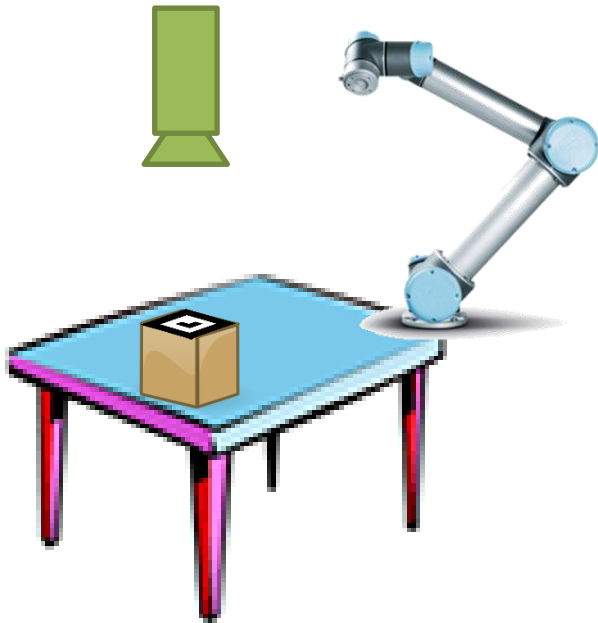


Exercise 1.3.2

Exercise 1.3.2

Create a Node:

*In myworkcell_core package
called vision_node*

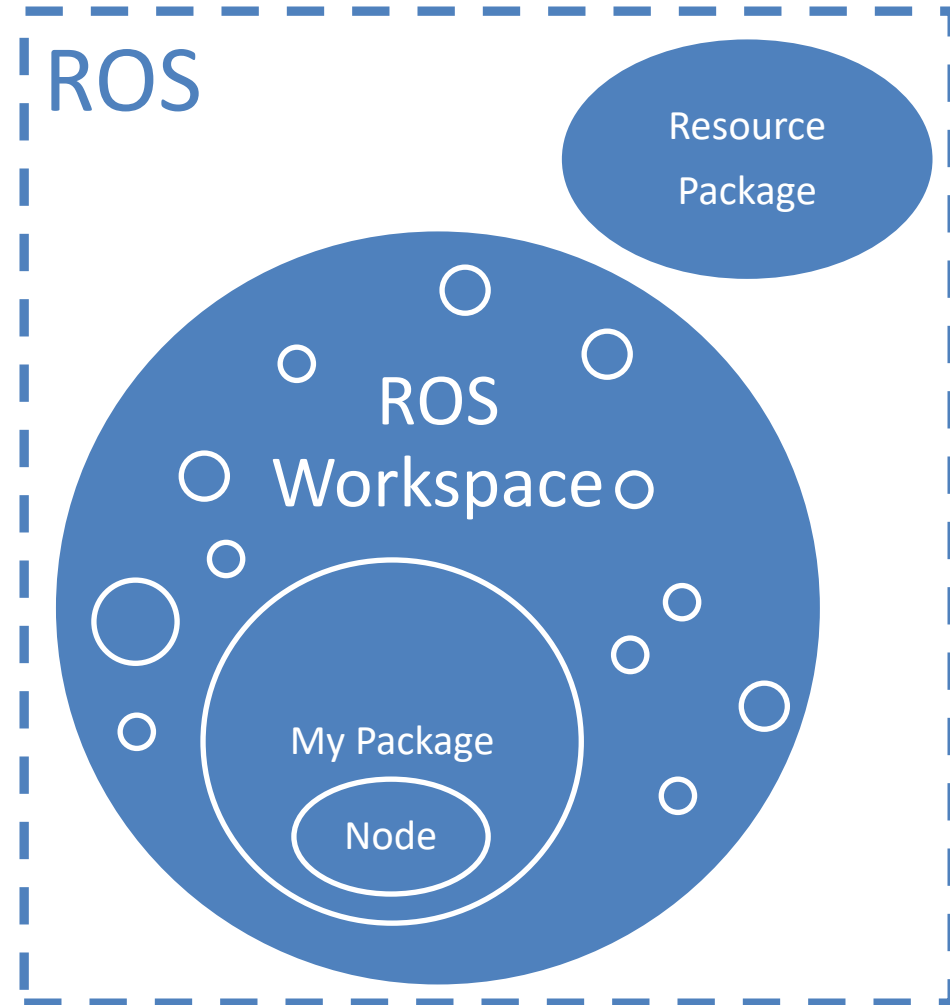




Day 1 Progression



- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ✓ Create Package
- ✓ Create Node
 - ✓ Basic ROS Node
 - Interact with other nodes
 - Messages
 - Services
- ✓ Run Node
 - ✓ ros2 run
 - ros2 launch



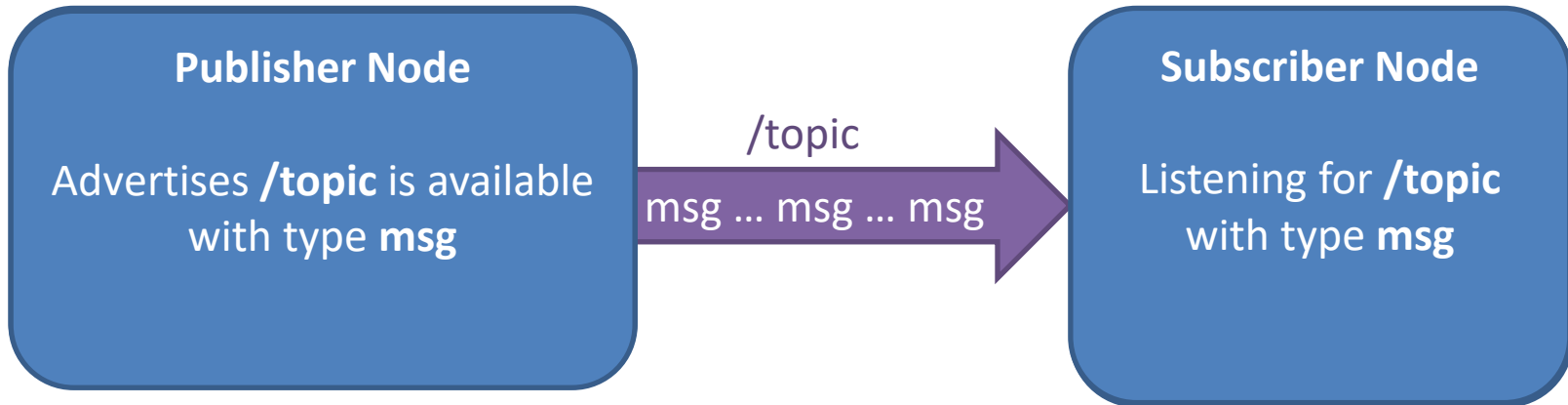


Topics and Messages





Topics are for **Streaming Data**

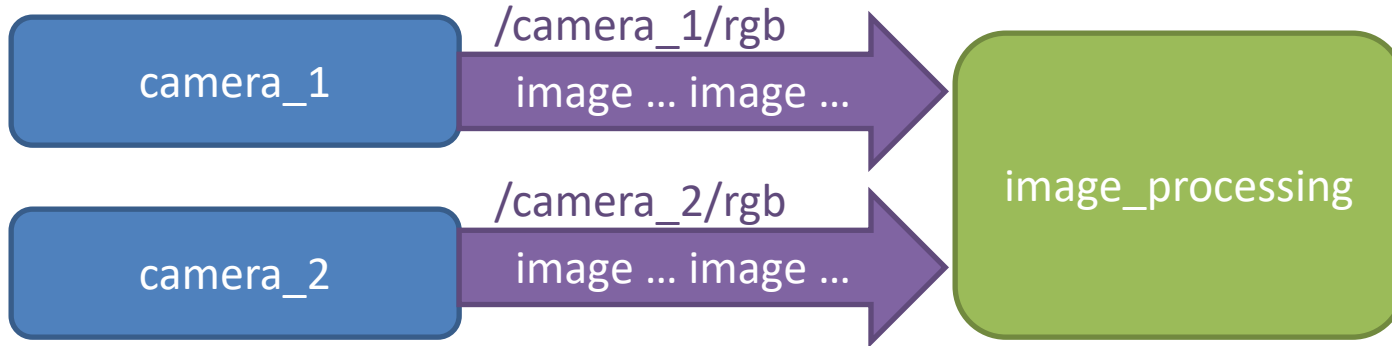




Topics vs. Messages

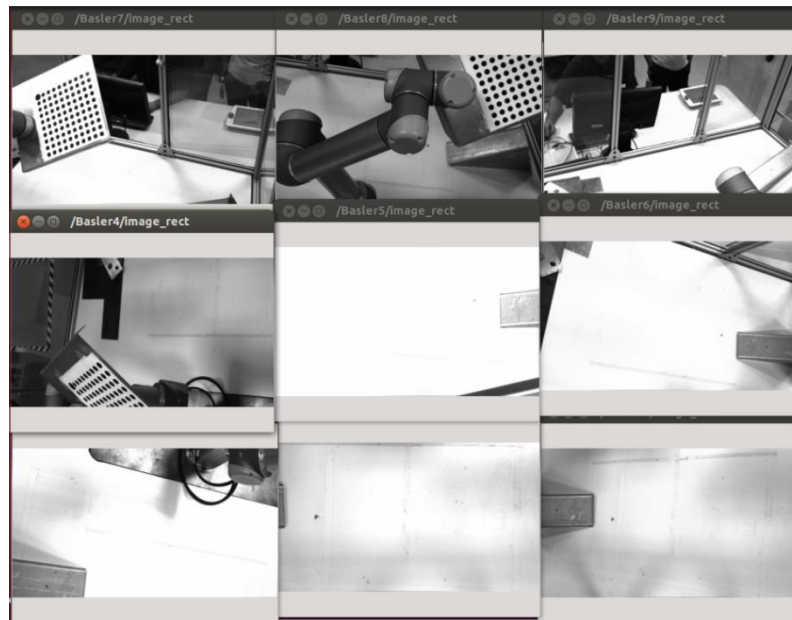
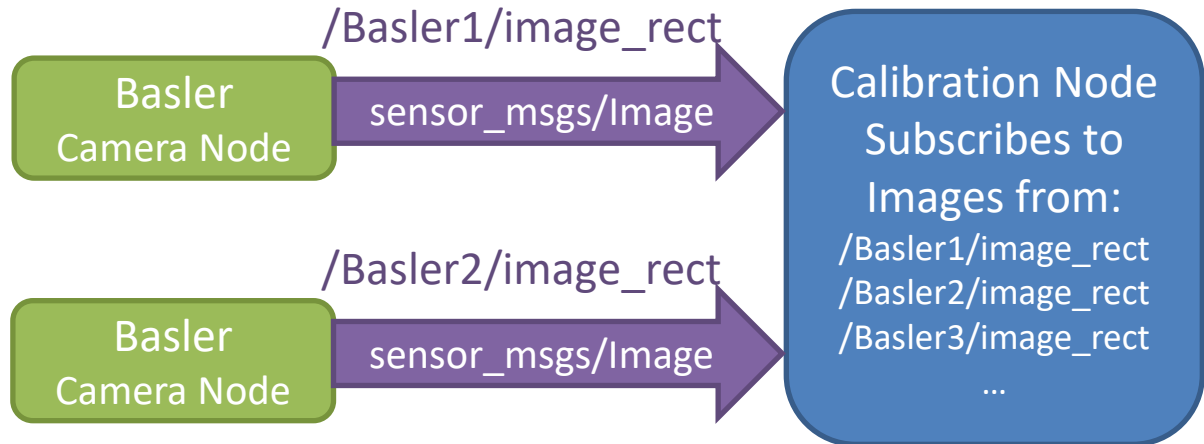


- Topics are **channels**, Messages are **data types**
 - Different topics can use the same Message type





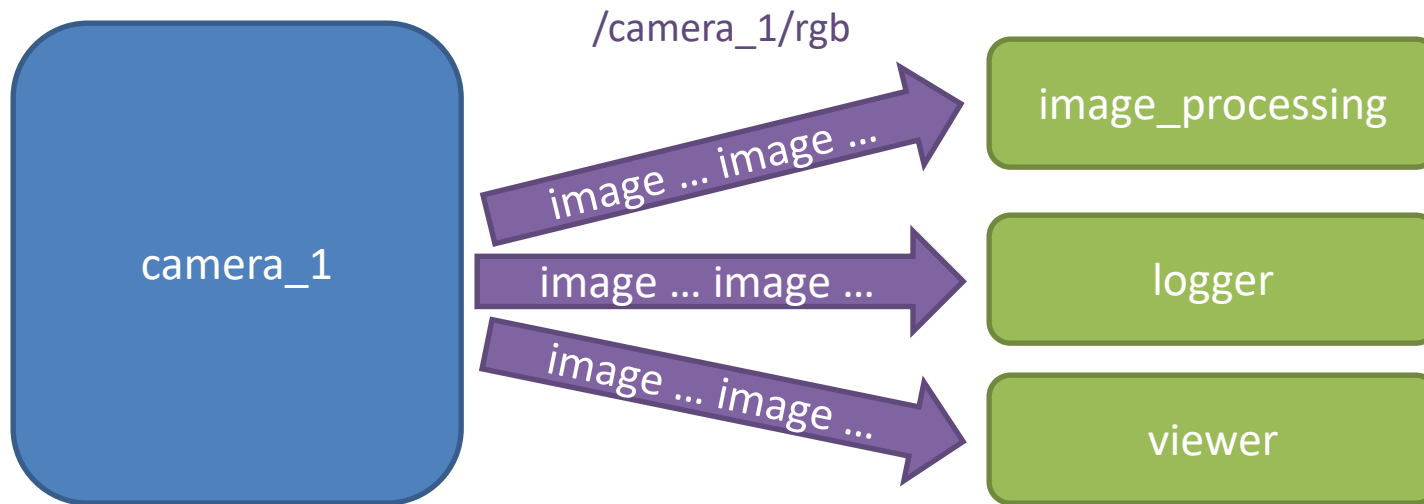
Practical Example





Multiple Pub/Sub

- Many nodes can pub/sub to same topic
 - comms are direct node-to-node





Topics : Details



- Each **Topic** is a stream of **Messages**:
 - sent by **publisher(s)**, received by **subscriber(s)**
- Messages are **asynchronous**
 - publishers don't know if anyone's listening
 - messages may be dropped
 - subscribers are event-triggered (by incoming messages)
- Typical Uses:
 - Sensor Readings: camera images, distance, I/O
 - Feedback: robot status/position
 - Open-Loop Commands: desired position





Quality of Service



- All ROS2 comms define a “Quality of Service” (QoS)
 - History/Depth - buffer N prior messages
 - Reliability - retry or discard dropped messages?
 - Durability - cache messages for late-joining subscribers?
 - Deadline - expected interval between messages
 - etc.
- All participants in a topic must have compatible QoS
 - Publishers - maximum QoS they can provide
 - Subscribers - minimum QoS they require
 - e.g. “reliable” subscriber won’t connect to “best-effort” publisher





QoS Profiles



- ROS provides default QoS profiles for different comms types.
 - Use these defaults, tweak them, or define your own application-specific QoS.
 - Default Profile (messages) queue=10, reliable, volatile
 - Services Profile queue=10, reliable, volatile
 - Sensor Profile queue=5, best-effort, volatile
 - Parameters Profile queue=1000, reliable, volatile





ROS Messages Types



- Similar to C structures
- Standard data primitives
 - Boolean: `bool`
 - Integer: `int8`, `int16`, `int32`, `int64`
 - Unsigned Integer: `uint8`, `uint16`, `uint32`, `uint64`
 - Floating Point: `float32`, `float64`
 - String: `string`
- Fixed length arrays: `bool[16]`
- Variable length arrays: `int32[]`
- Other: Nest message types for more complex data structure





Message Description File



- All Messages are defined by a `.msg` file

PathPosition.msg

comment



```
# A 2D position and orientation
```

other Msg type



```
std_msgs/Header header
```

```
float64 x # X coordinate
```

```
float64 y # Y coordinate
```

```
float64 angle # Orientation
```

data
type



field
name

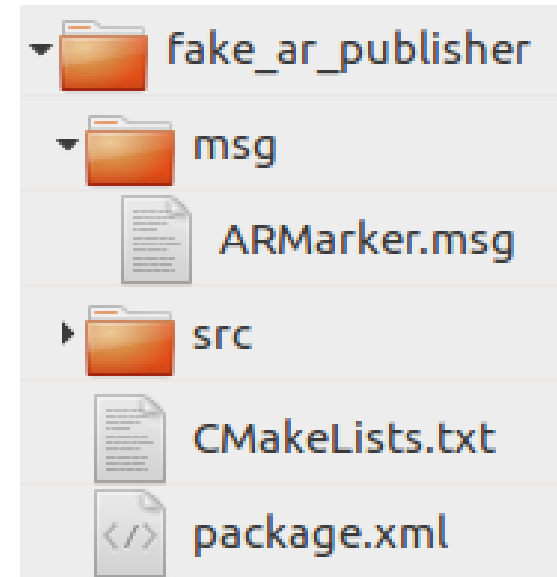




Custom ROS Messages



- Custom message types are defined in `msg` subfolder of packages
- **Modify** `CMakeLists.txt` to enable message generation.





- Lines needed to generate custom msg types

```
find_package(rosidl_default_generators  
REQUIRED)
```

```
rosidl_generate_interfaces(  
  msg/CustomMsg.msg  
  DEPENDENCIES ...)
```





package.xml



```
<build_depend> rosidl_default_generators </build_depend>
```

```
<exec_depend>rosidl_default_runtime</exec_depend>
```

```
<member_of_group>rosidl_interface_packages</member_of_group>
```





ROS Interface Commands



These commands show info about known ROS message types (+ services/actions, discussed later)

- `ros2 interface list`
 - Show all ROS message types currently available
- `ros2 interface package <package>`
 - Show all ROS message types in package <package>
- `ros2 interface show <package>/<message_type>`
 - Show the structure of the given message type





ROS Topic Commands



- `ros2 topic list`
 - List all topics currently subscribed to and/or publishing
- `ros2 topic type <topic>`
 - Show the message type of the topic
- `ros2 topic info <topic>`
 - Show topic message type, subscribers, publishers, etc.
- `ros2 topic echo <topic>`
 - Echo messages published to the topic to the terminal
- `ros2 topic find <message_type>`
 - Find topics of the given message type





“Real World” – Messages



- Use *rqt_msg* to view:
 - sensor_msgs/JointState
 - trajectory_msgs/JointTrajectory
 - sensor_msgs/Image
 - rcl_interfaces/Log



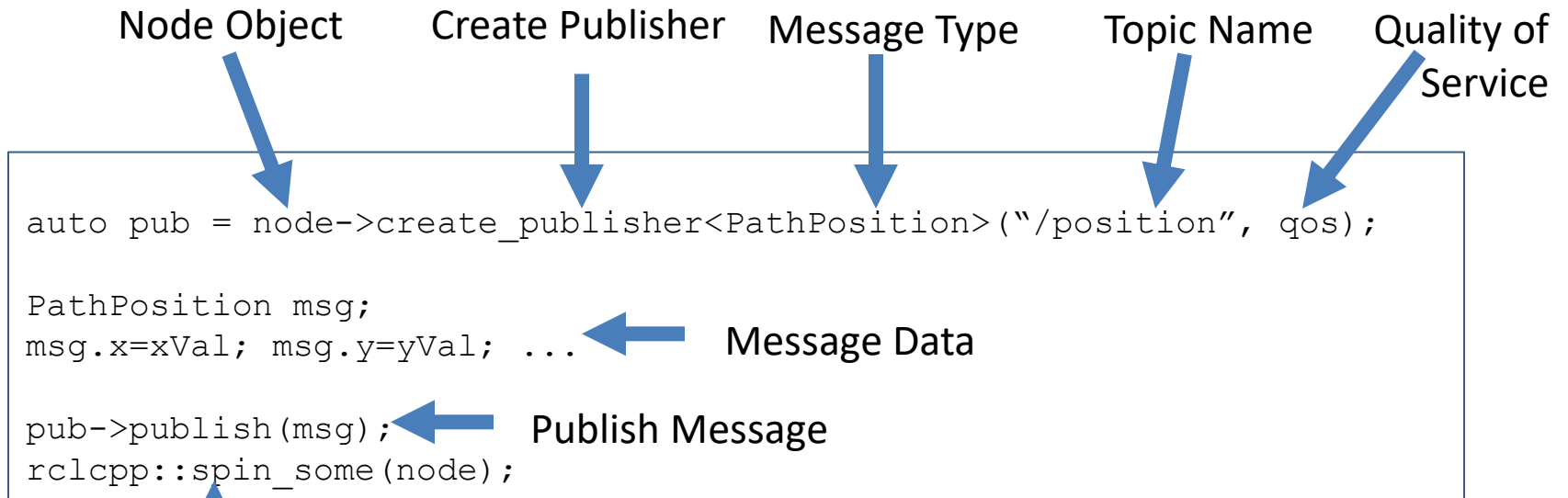


Topics: Syntax



- **Topic Publisher**

- Advertises available topic (*Name, Data Type, QoS*)
- Populates message data
- Periodically publishes new data



Background
Process





Topics: Syntax



- **Topic Subscriber**
 - Defines callback function
 - Listens for available topic (*Name, Data Type, QoS*)

Callback Function



Message Type



Message Data (IN)



```
void msg_callback(const PathPosition& msg) {  
    RCLCPP_INFO_STREAM(node->get_logger(), "Received msg: " << msg);  
}  
  
auto sub = node->create_subscription("/topic", qos, msg_callback);
```



Server Object



Service Name



Callback Ref

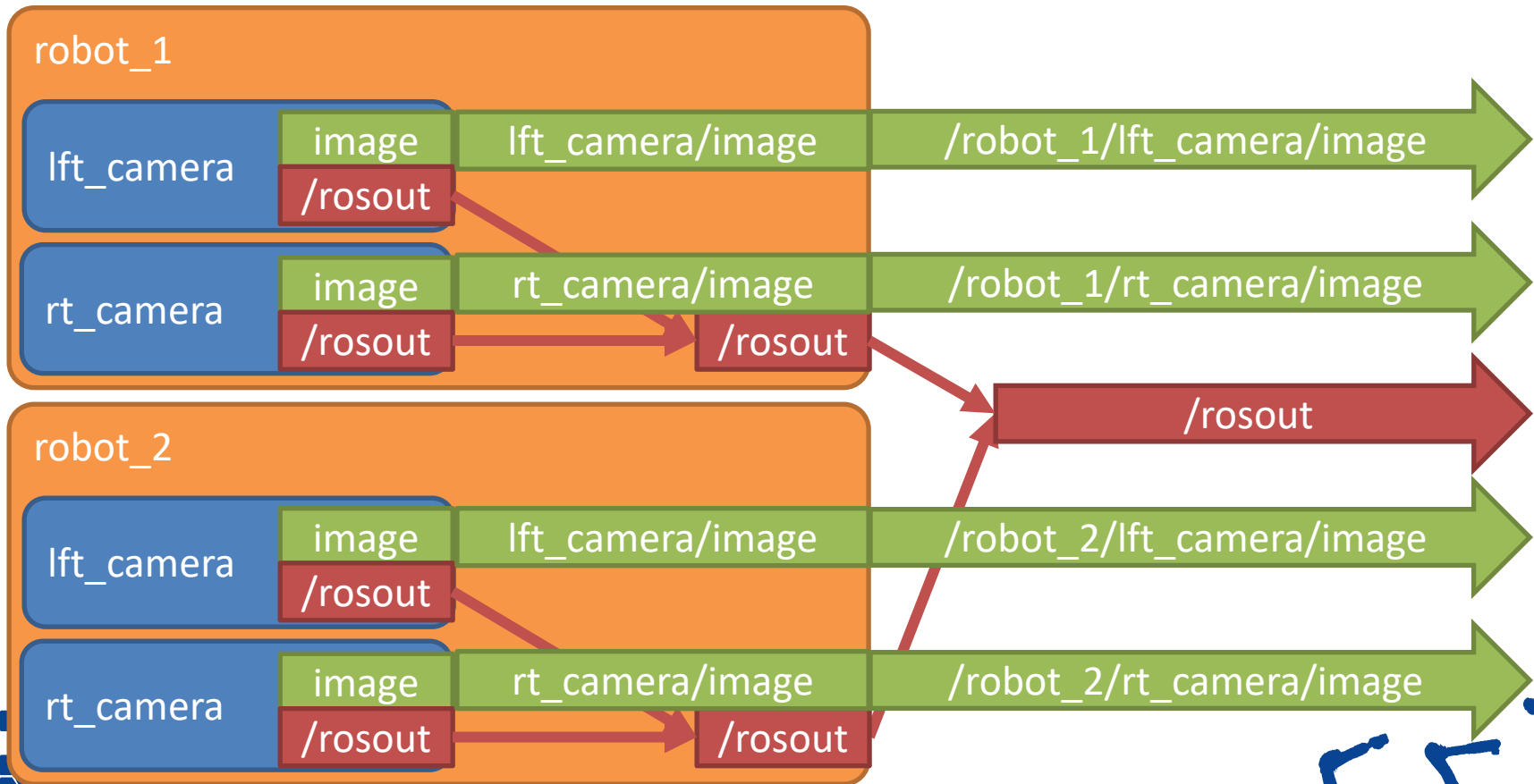




Namespaces



- ROS requires unique names for nodes/topics/etc.
- Namespaces allow separation:
 - Similar nodes can co-exist, in different “namespaces”
 - relative vs. absolute name references





Instead of text editor and building from terminal...

Use an IDE! ([detailed instructions here](#))



1. Launch QtCreator IDE from desktop shortcut
2. File -> New Project
3. Other Project -> ROS Workspace
4. Enter Project Properties:
 1. Name = "ROS2_Training" (or whatever)
 2. Distribution (should be auto-detected)
 3. Build System = Colcon
 4. Path = ~/ros2_ws
5. Build -> Build All
 1. you should see success in the "Compile" tab





Exercise 1.4

Exercise 1.4

Subscribe to fake_ar_publisher

