ROS-Industrial Basic Developer's Training Class

August 2017

Southwest Research Institute

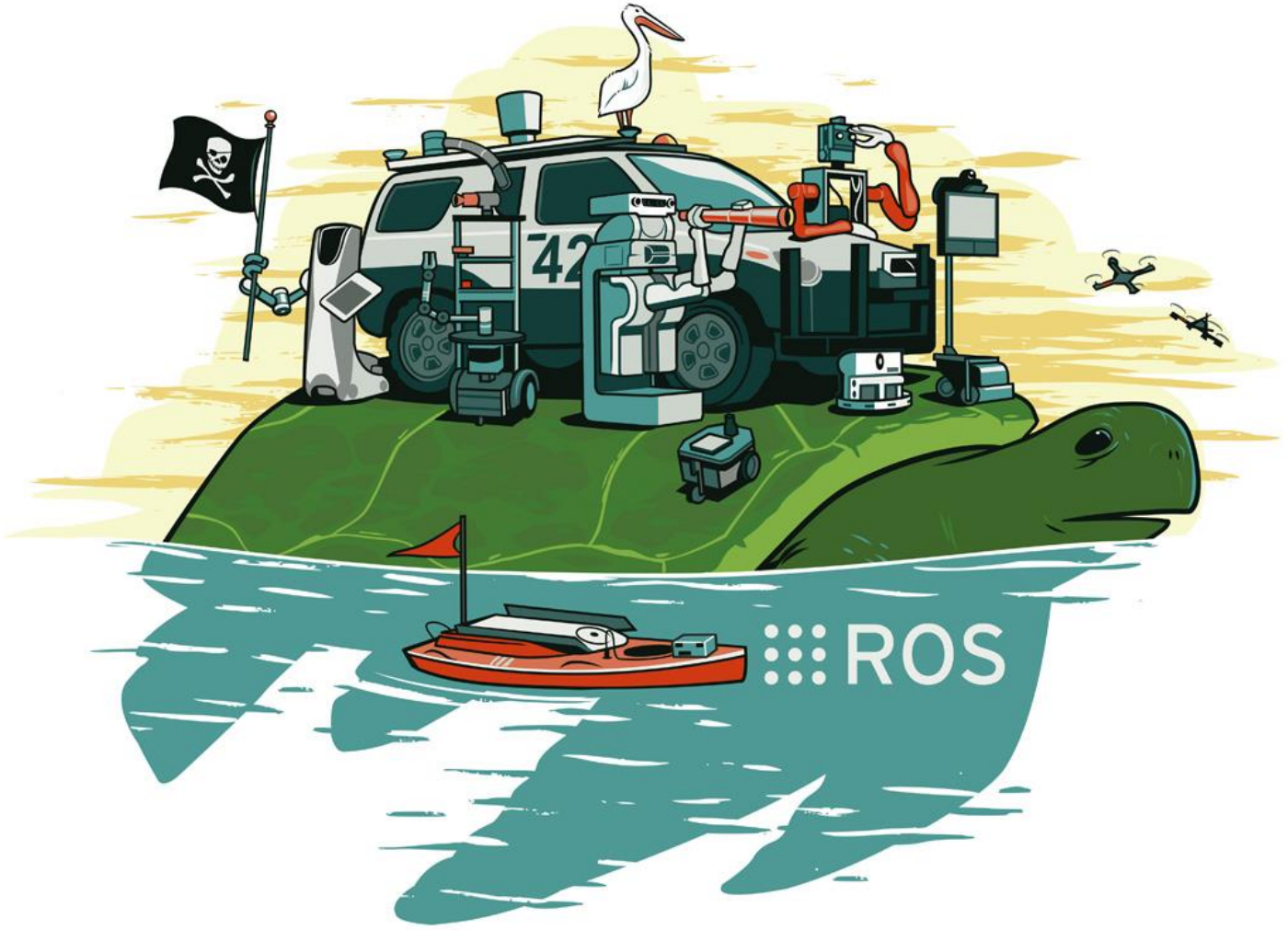
Session 1:
ROS Basics

Southwest Research Institute

Outline

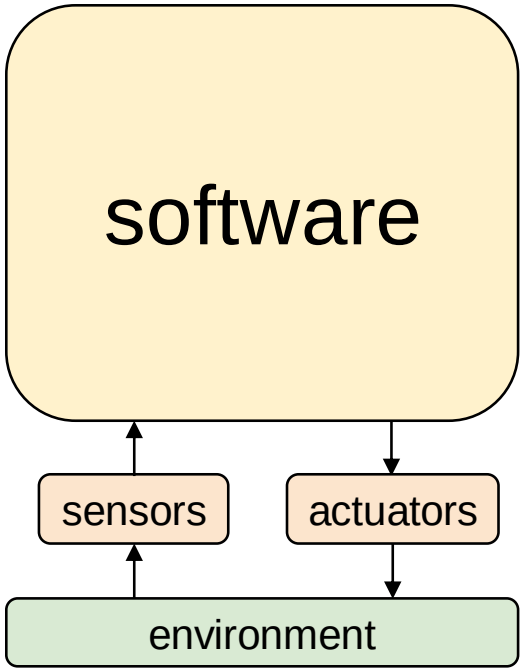
- Intro to ROS
- Catkin (Create workspace)
- Installing packages (existing)
- Packages (create)
- Nodes
- Messages / Topics

An Introduction to ROS



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

ROS : The Big Picture

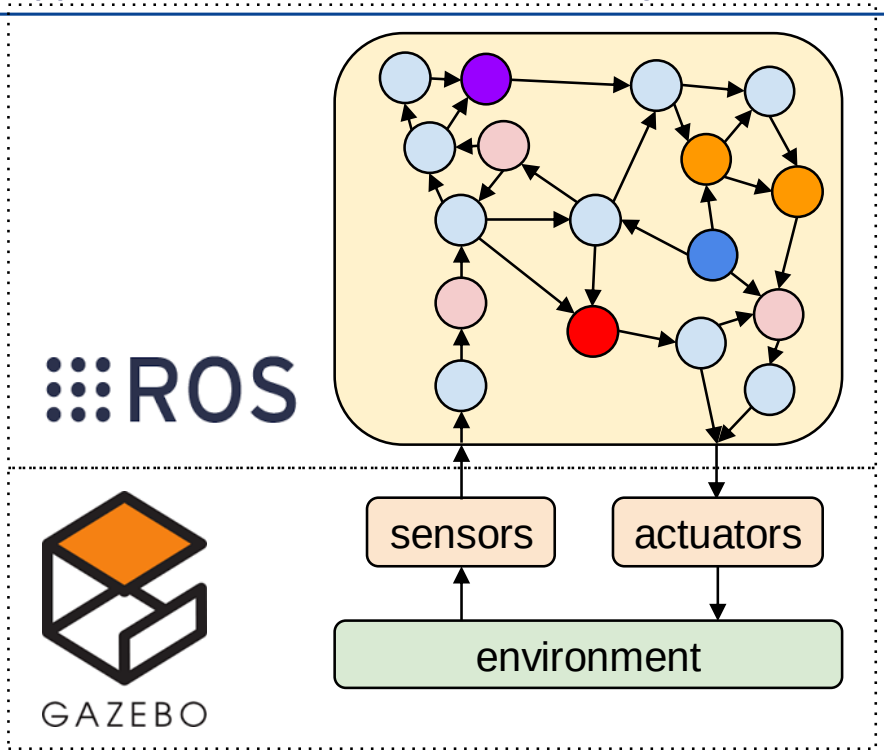


All robots are:

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)

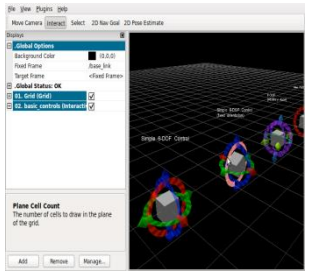
What is ROS?

ROS is…

=

Plumbing

+



Tools

+

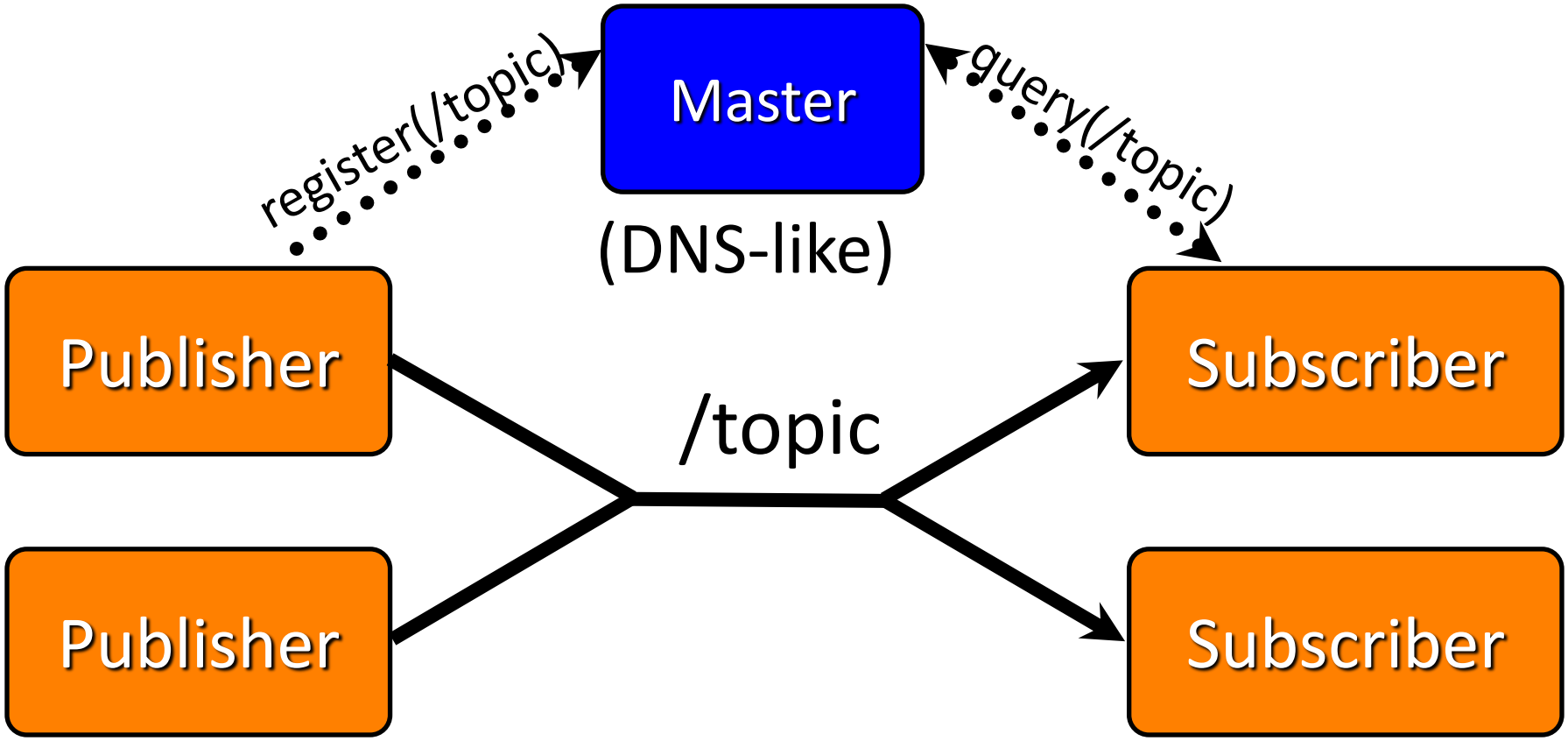
Capabilities

+

Ecosystem

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

ROS is… plumbing



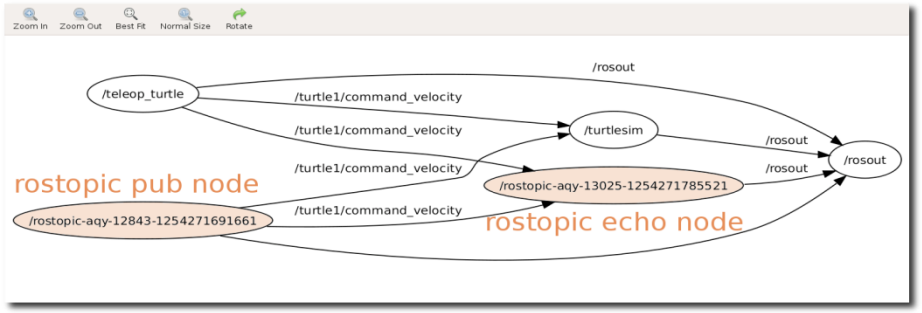
(Adapted from Willow Garage's "What is ROS?" Presentation)

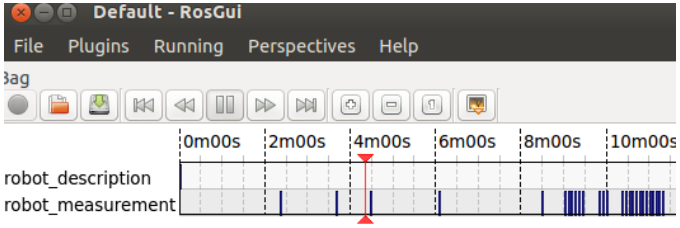
ROS Plumbing : Drivers

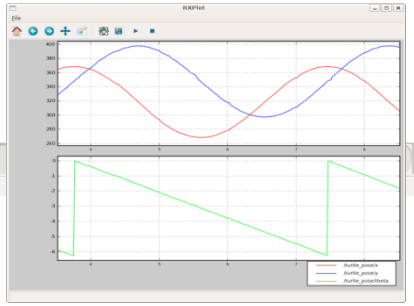
- 2d/3d cameras
- laser scanners
- robot actuators
- inertial units
- audio
- GPS
- joysticks
- etc.

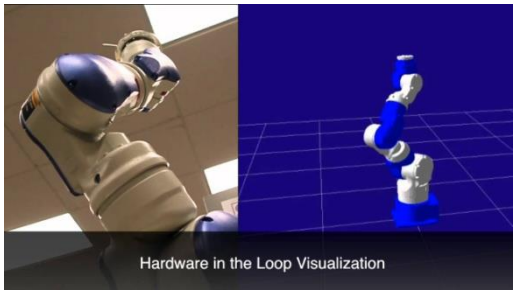
(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)

ROS is …Tools

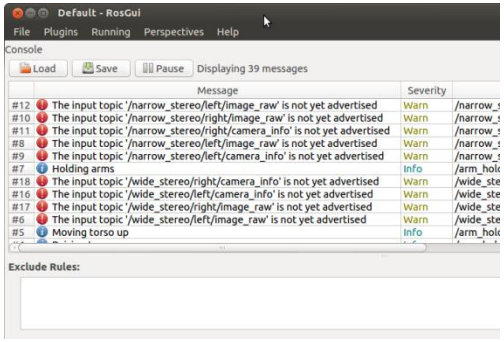






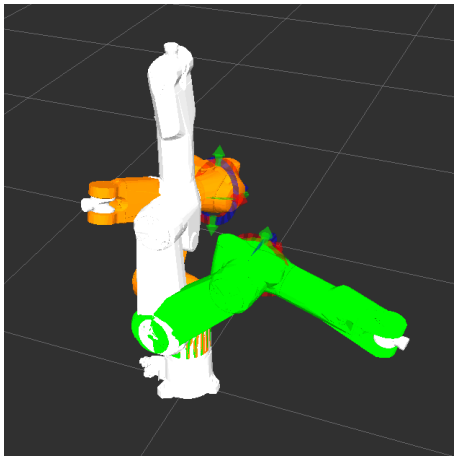


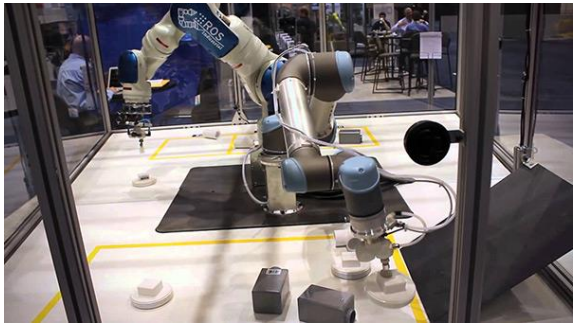
- logging/plotting
- graph visualization
- diagnostics
- visualization

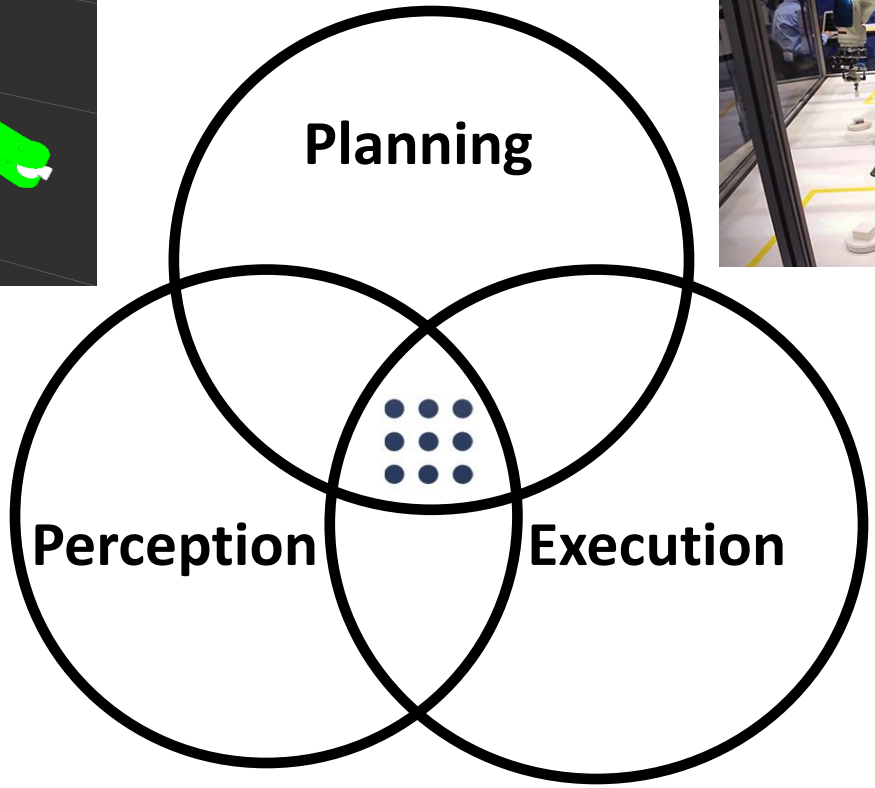


(Adapted from Willow Garage's "What is ROS?" Presentation)

ROS is…Capabilities

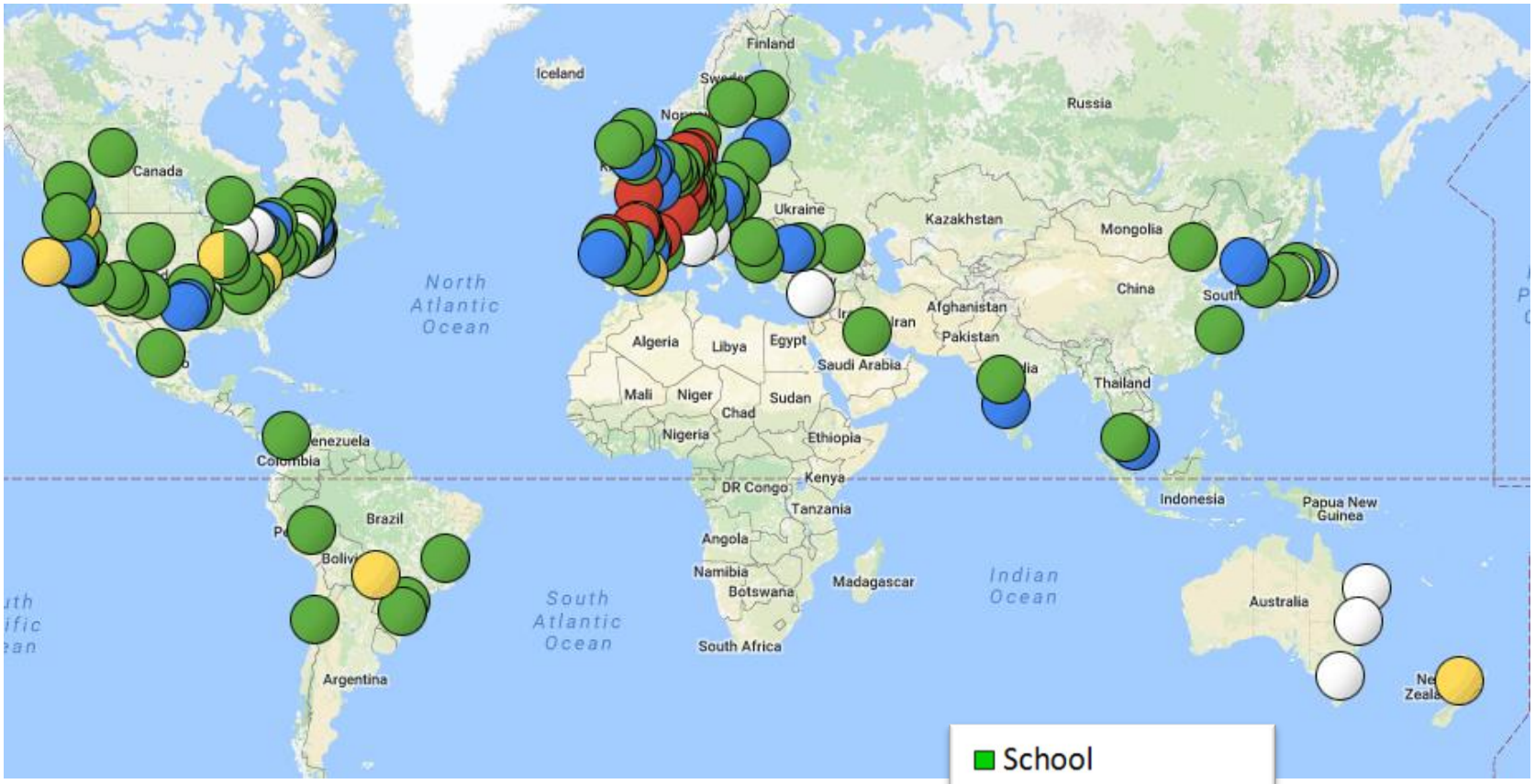






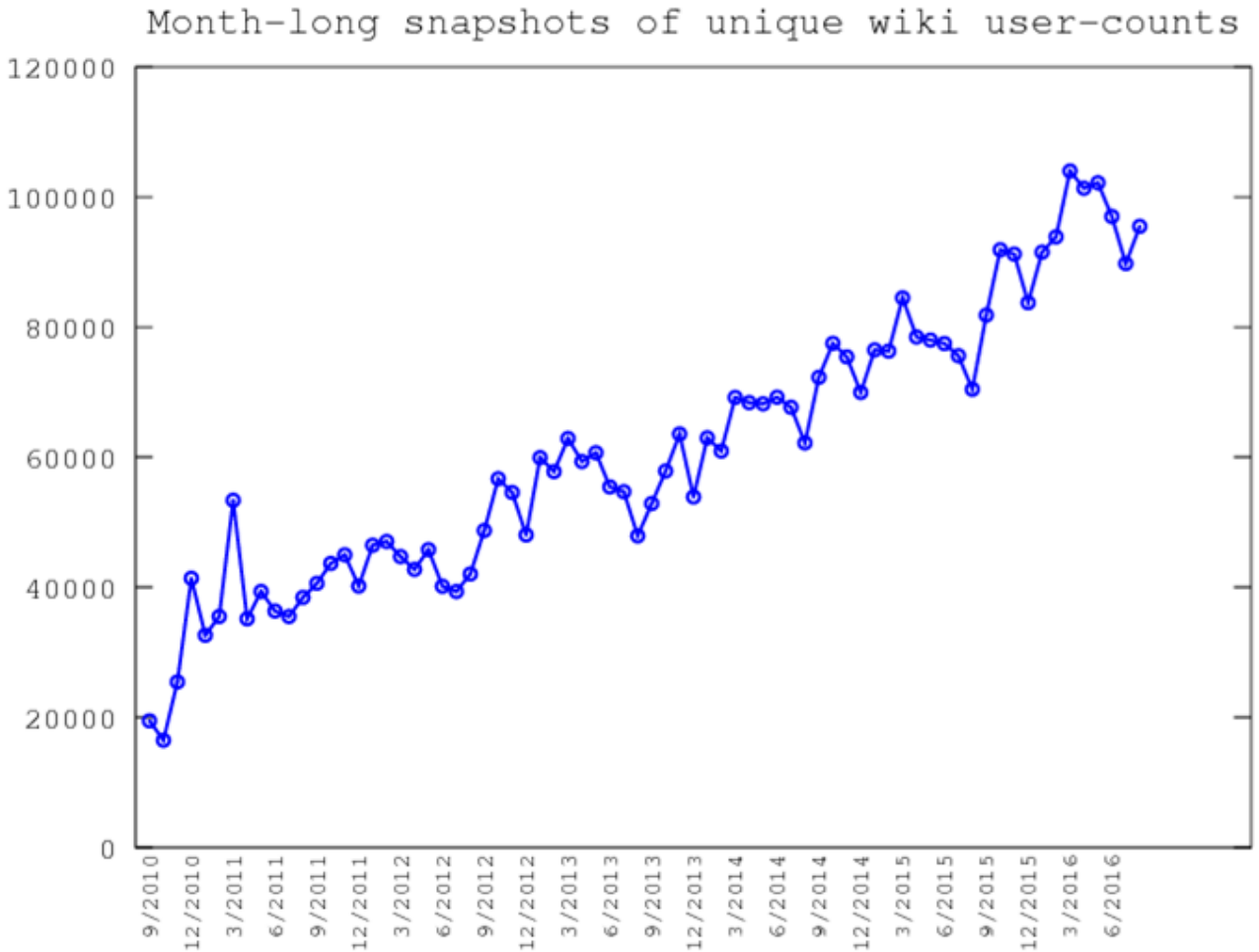
(Adapted from Willow Garage's "What is ROS?" Presentation)

ROS is… an Ecosystem



http://metrorobots.com/rosmap.html

ROS is a growing Ecosystem



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

ROS is International

unique wiki visitors Jan 2016 – July 2016

1.	United States	96,176 (22.76%)
2.	China	41,902 (9.92%)
3.	Germany	33,072 (7.83%)
4.	Japan	31,028 (7.34%)
5.	India	21,326 (5.05%)
6.	France	14,432 (3.42%)
7.	United Kingdom	13,225 (3.13%)
8.	Canada	11,278 (2.67%)
9.	South Korea	10,788 (2.55%)
10.	Spain	10,576 (2.50%)
11.	Italy	9,166 (2.17%)
12.	Russia	7,980 (1.89%)
13.	Taiwan	7,458 (1.76%)
14.	Brazil	6,683 (1.58%)
15.	Singapore	6,462 (1.53%)

visitors per million people

1. Singapore: 1167
2. Switzerland: 505
3. Germany: 404
4. Sweden: 394
5. Hong Kong: 370

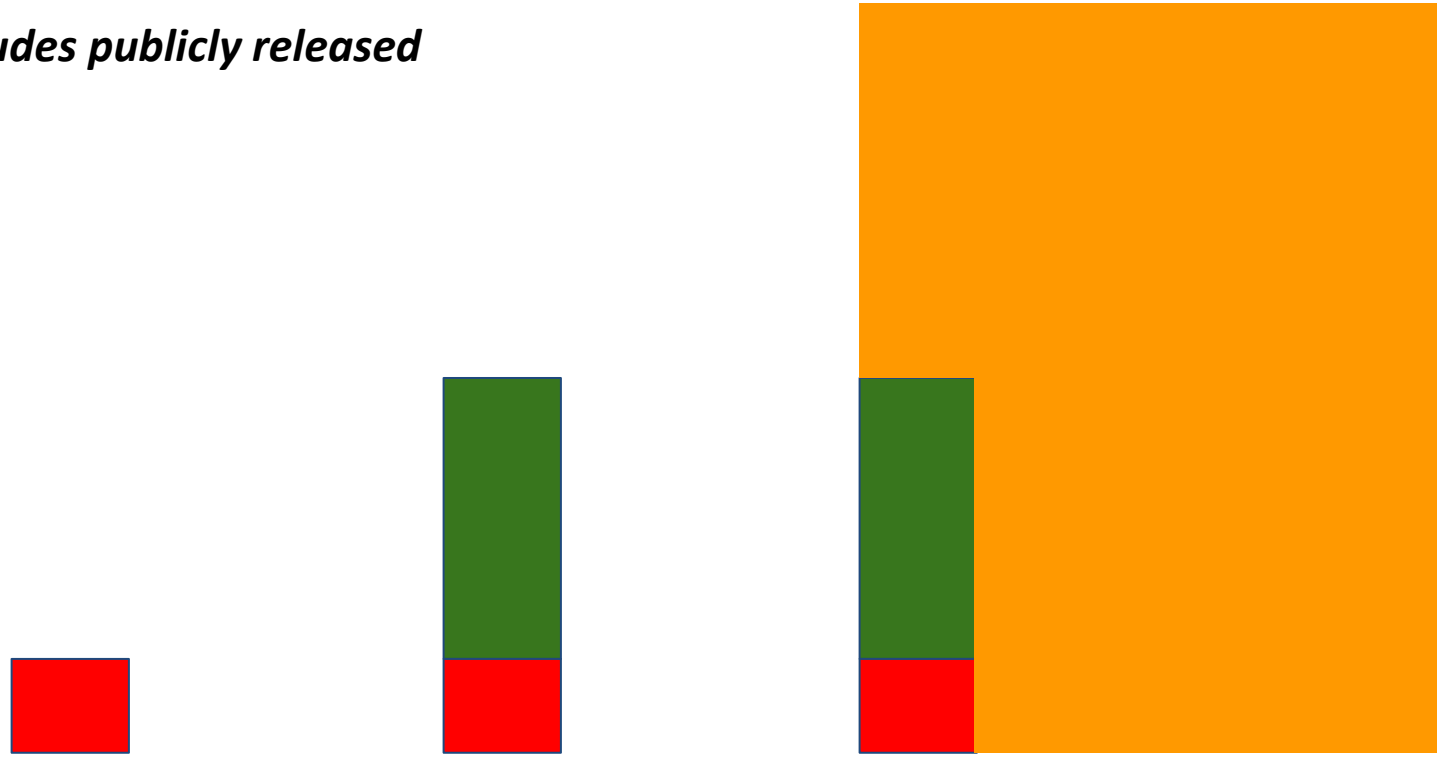
...

14. USA: 297

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

ROS is a Repository

only includes publicly released code!



ros_comm
("core")
100 KLOC

desktop-full
("core+tools")
400 KLOC

all buildfarm
("universe")
4000 KLOC

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

ROS is …

ROS industrial consortium™

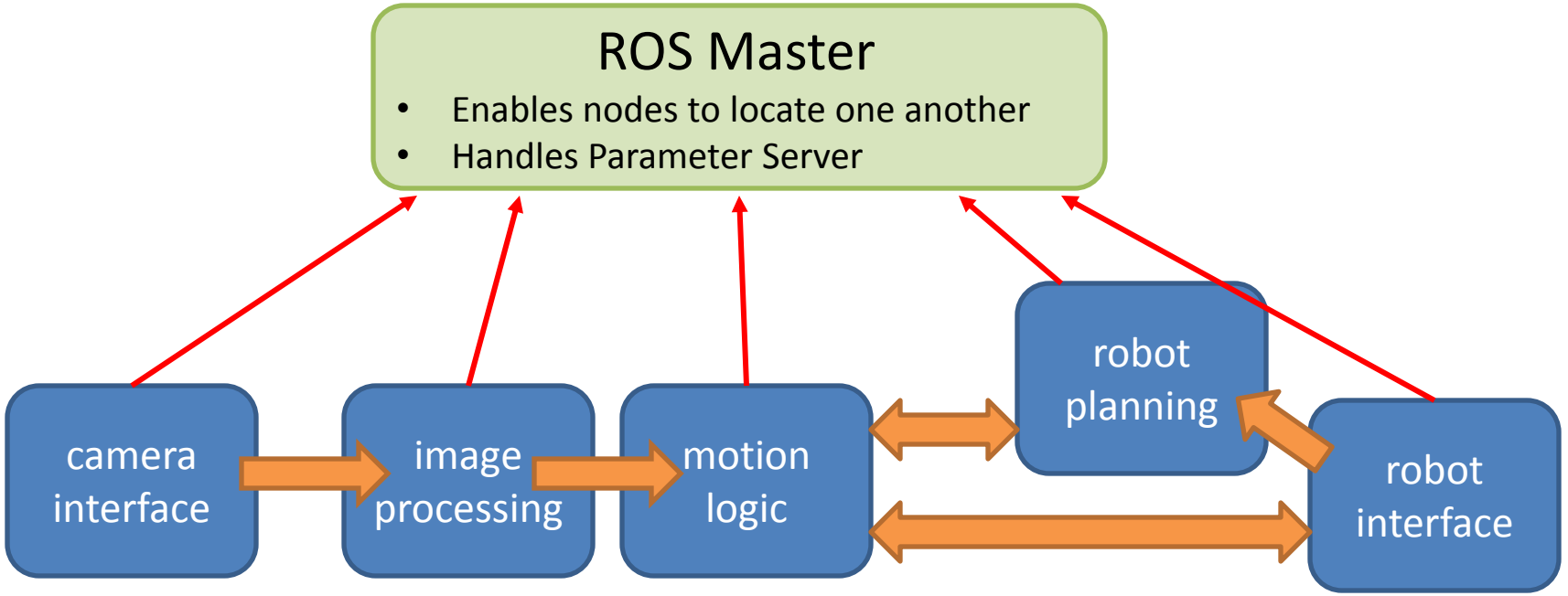
ROS

CELEBRATING EIGHT YEARS

November 2015

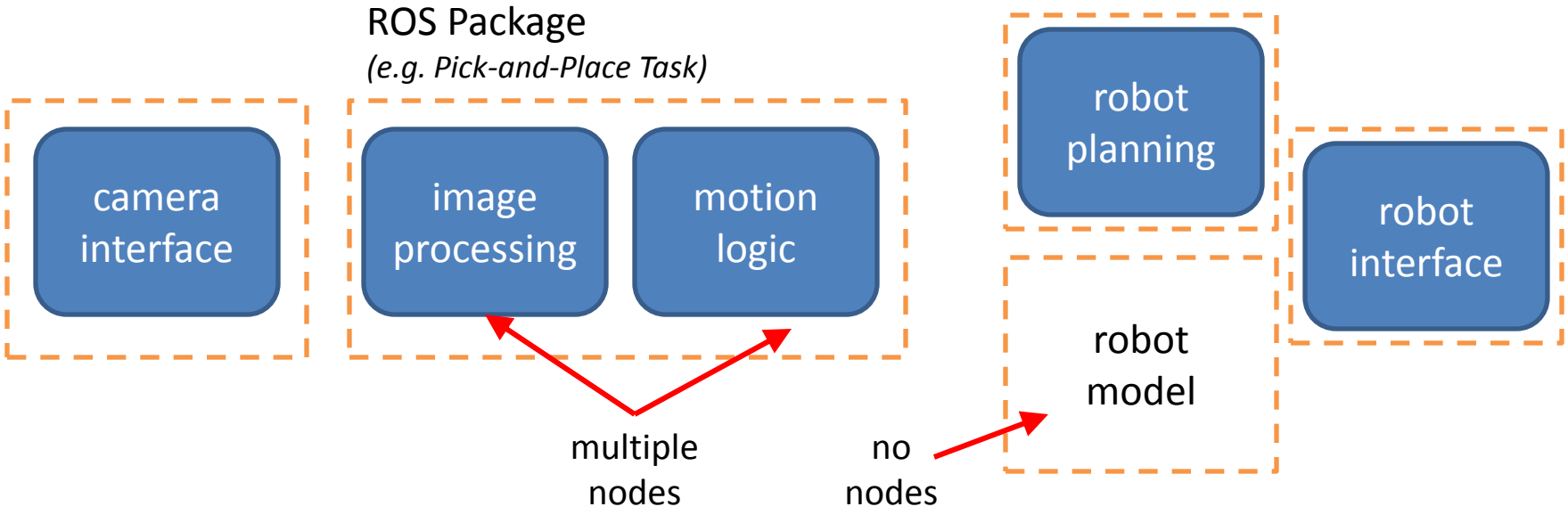
https://www.youtube.com/watch?v=Z70_3wMFO24

ROS Architecture: Nodes



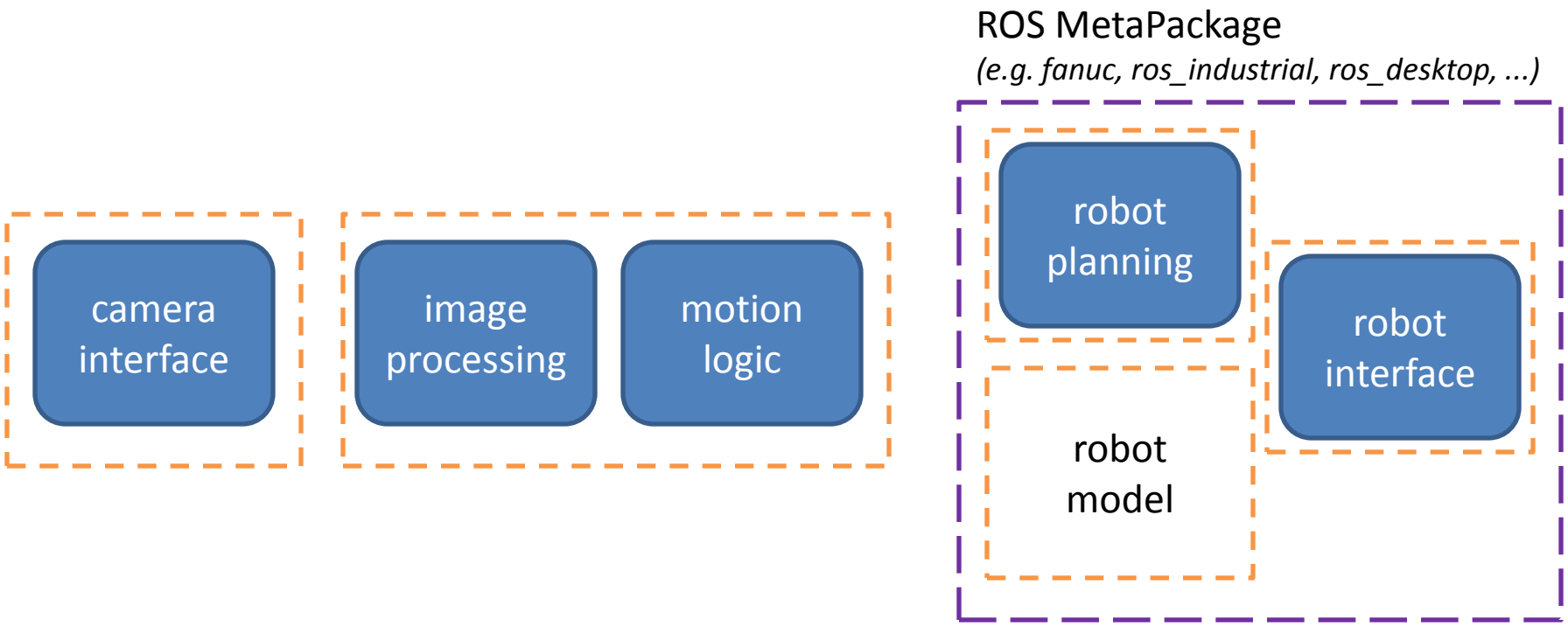
- A **Node** is a single ROS-enabled program
  - Most communication happens **between** nodes
  - Nodes can run on many different **devices**
- One **Master** per system

ROS Architecture: Packages



- ROS **Packages** are groups of related nodes/data
  - Many ROS commands are **package-oriented**

ROS Architecture: MetaPkg



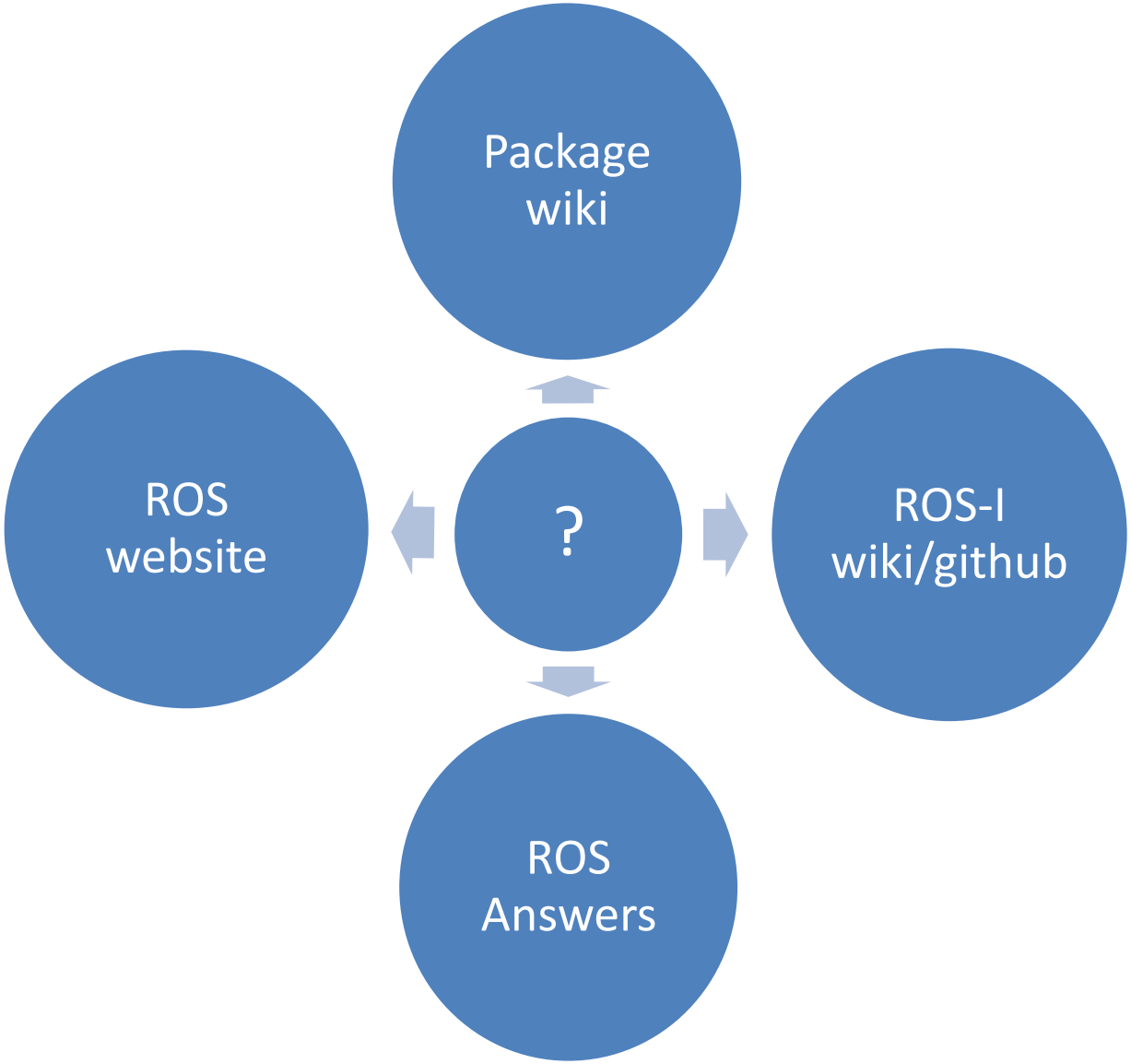
- **MetaPackages** are groups of related packages
  - Mostly for convenient install/deployment

ROS Programming

- ROS uses **platform-agnostic** methods for most communication
  - TCP/IP Sockets, XML, etc.

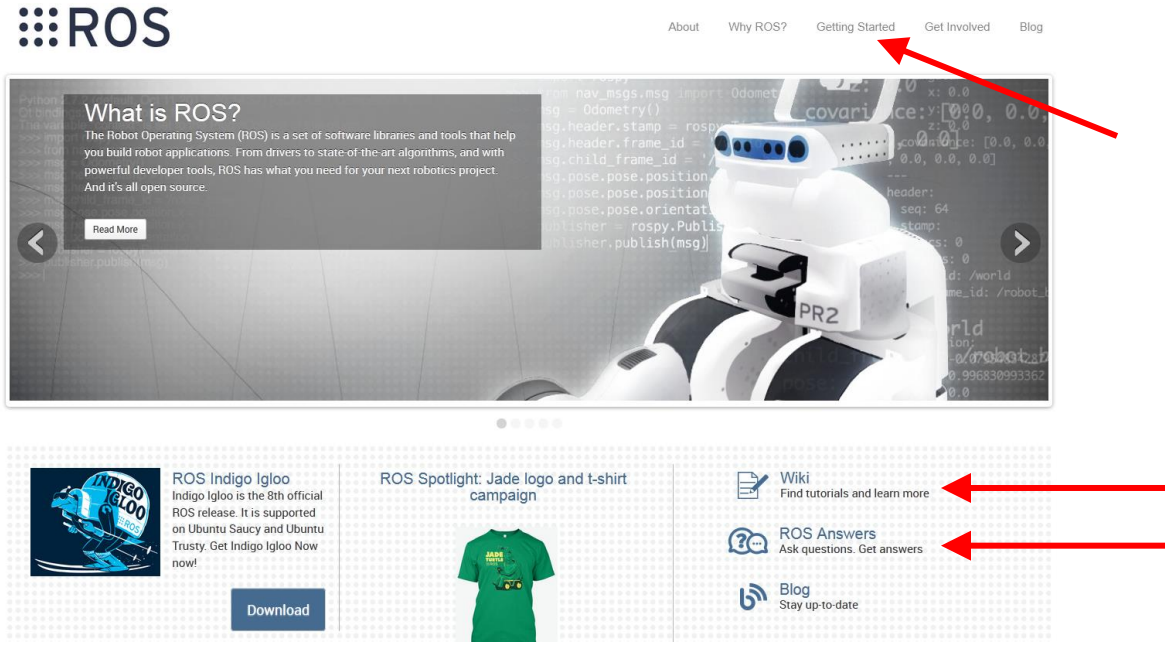
- Can intermix programming languages
  - currently: C++, Python, Lisp
  - We will be using C++ for our exercises

ROS Resources



ROS.org Website

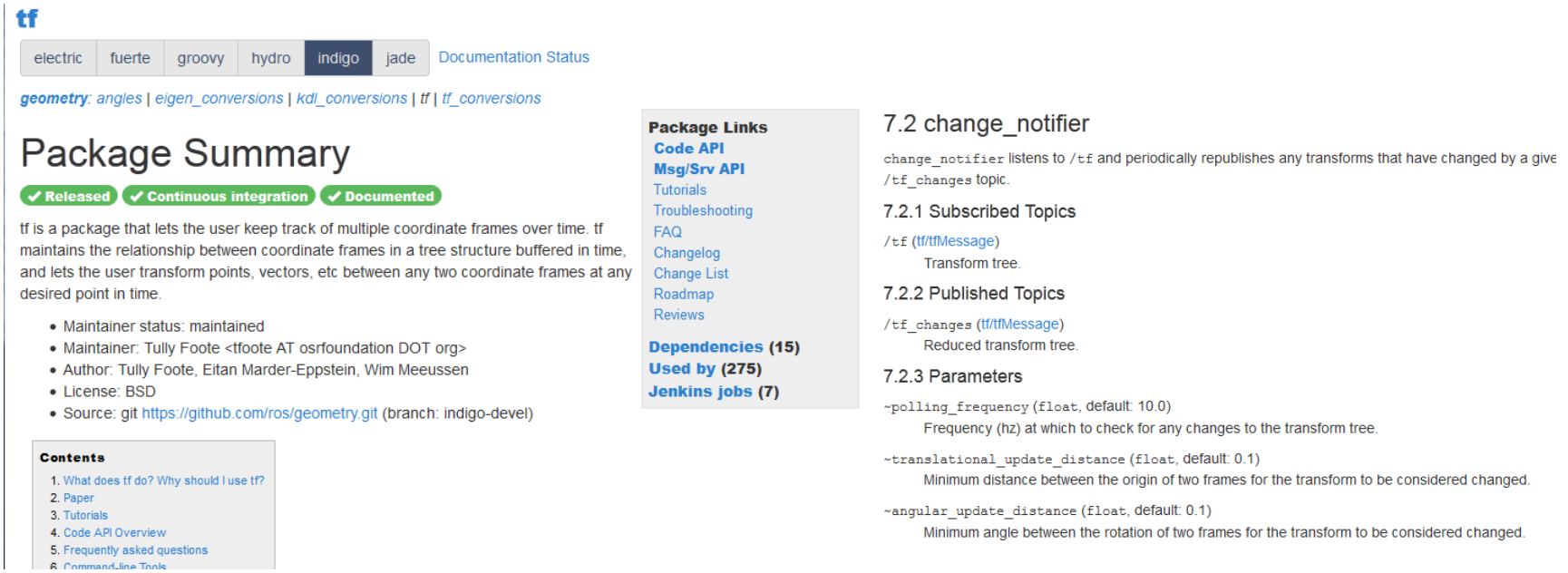
http://ros.org



- Install Instructions
- Tutorials
- Links
  - Packages, ROS Answers, etc.

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.

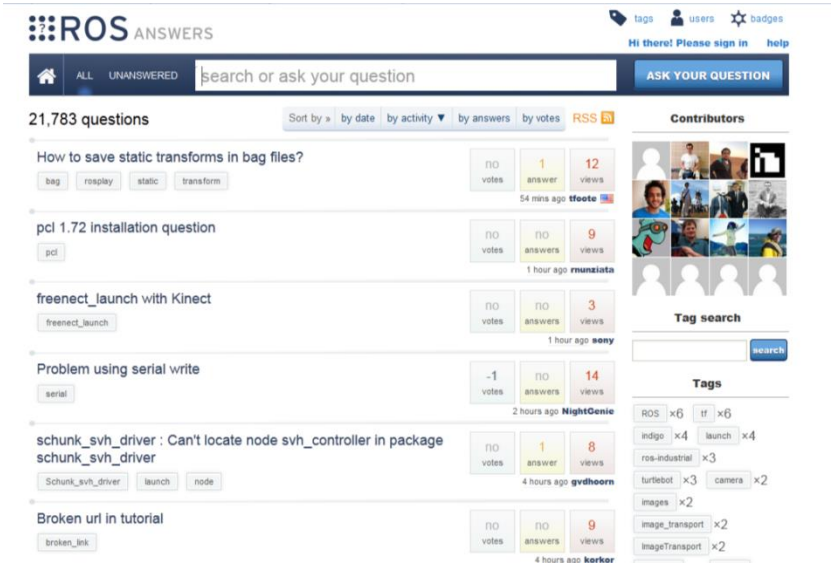
`~translational_update_distance` (`float`, default: 0.1)
Minimum distance between the origin of two frames for the transform to be considered changed.

`~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

ROS Answers

http://answers.ros.org



- 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

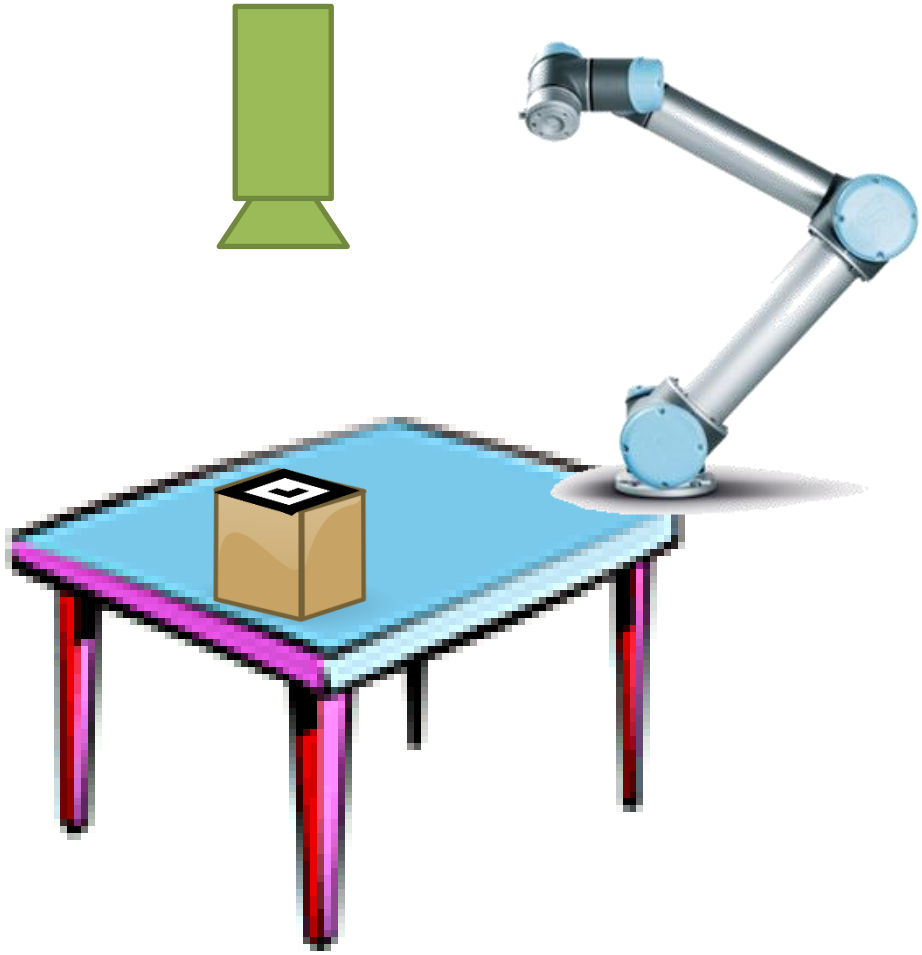
- Most ROS-code is open-source
  - can be reviewed / improved by everyone
  - we count on **YOU** to help ROS grow!

What is ROS to you?

Training Goals:

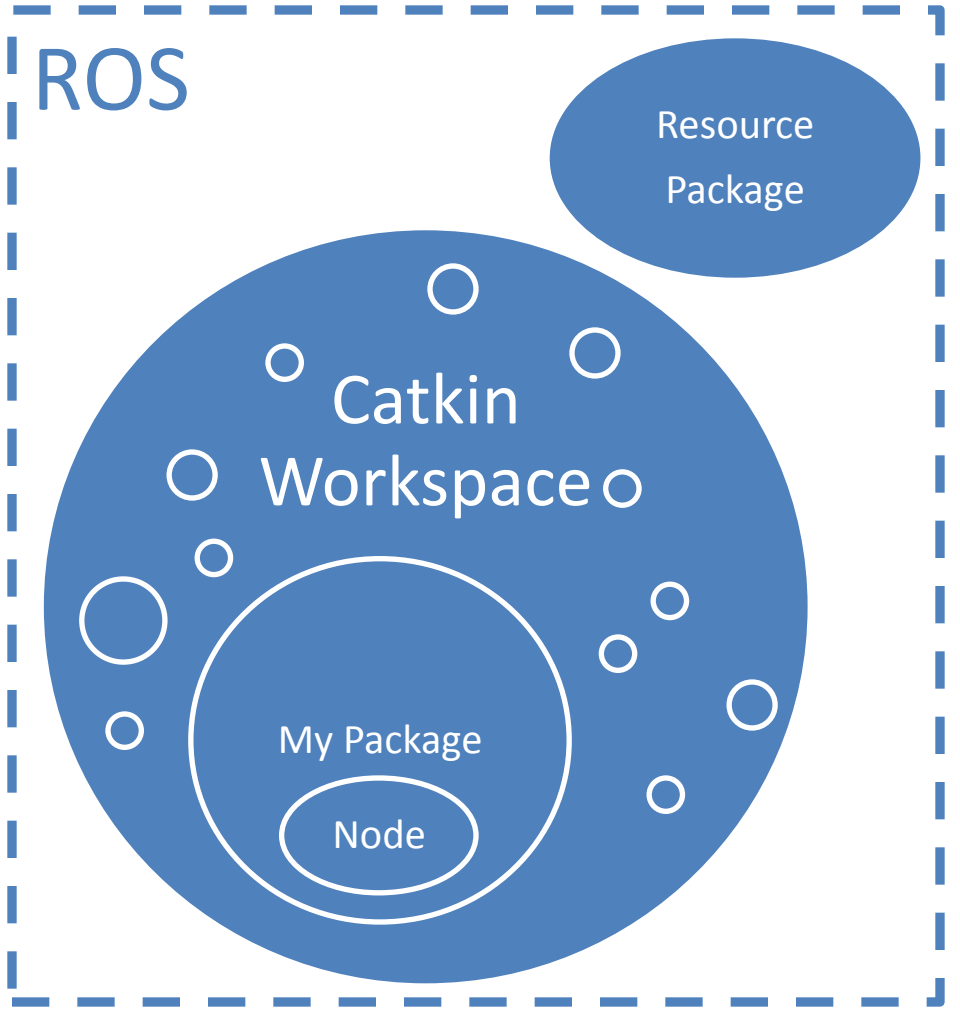
- Show you ROS as a software framework
- Show you ROS as a tool for problem solving
- Apply course concepts to a sample application
- Ask lots of questions and break things.

Scan & Plan “Application”



Day 1 Progression

- ❑ Install ROS
- ❑ Create Workspace
- ❑ Add “resources”
- ❑ Create Package
- ❑ Create Node
  - ❑ Basic ROS Node
  - ❑ Interact with other nodes
    - ❑ Messages
    - ❑ Services
- ❑ Run Node
  - ❑ rosrun
  - ❑ roslaunch
  - ❑ rosparam



Installing ROS

Getting ROS

http://wiki.ros.org/kinetic/Installation

Roscore

roscore is a collection of nodes and programs that are pre-requisites of a ROS-based system

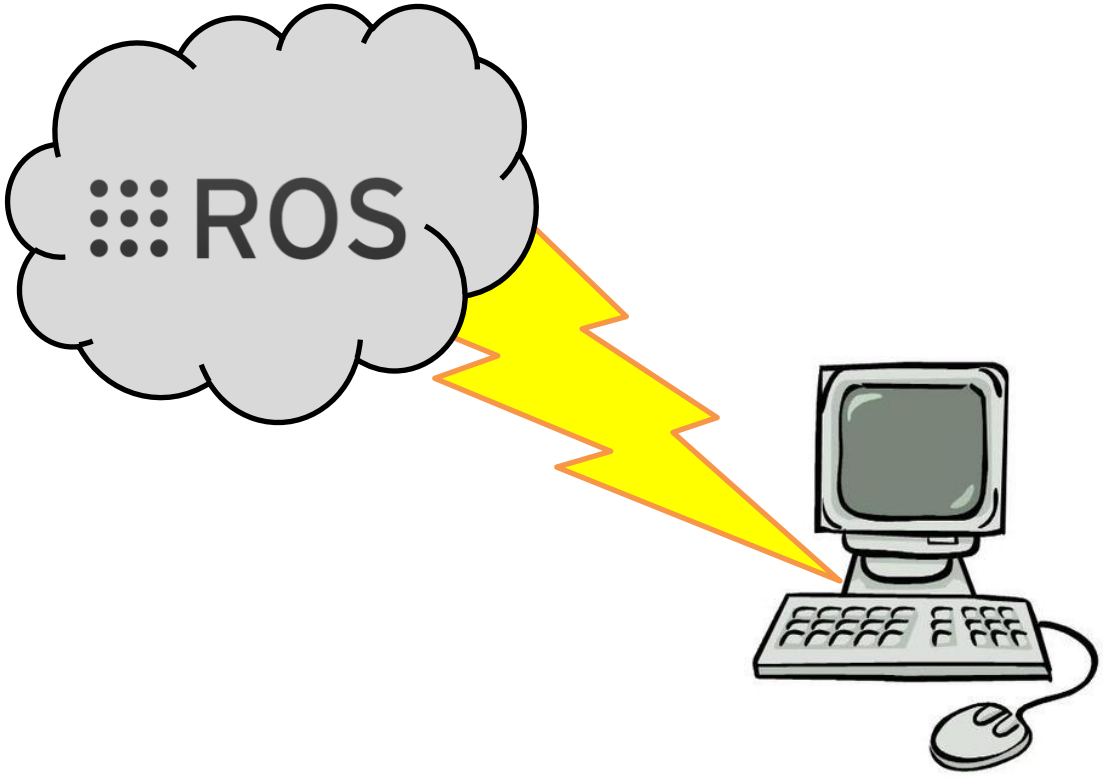
To check your install, open a terminal and type:

roscore

To kill the process, press ***Ctrl+C*** while in the window running ***roscore***

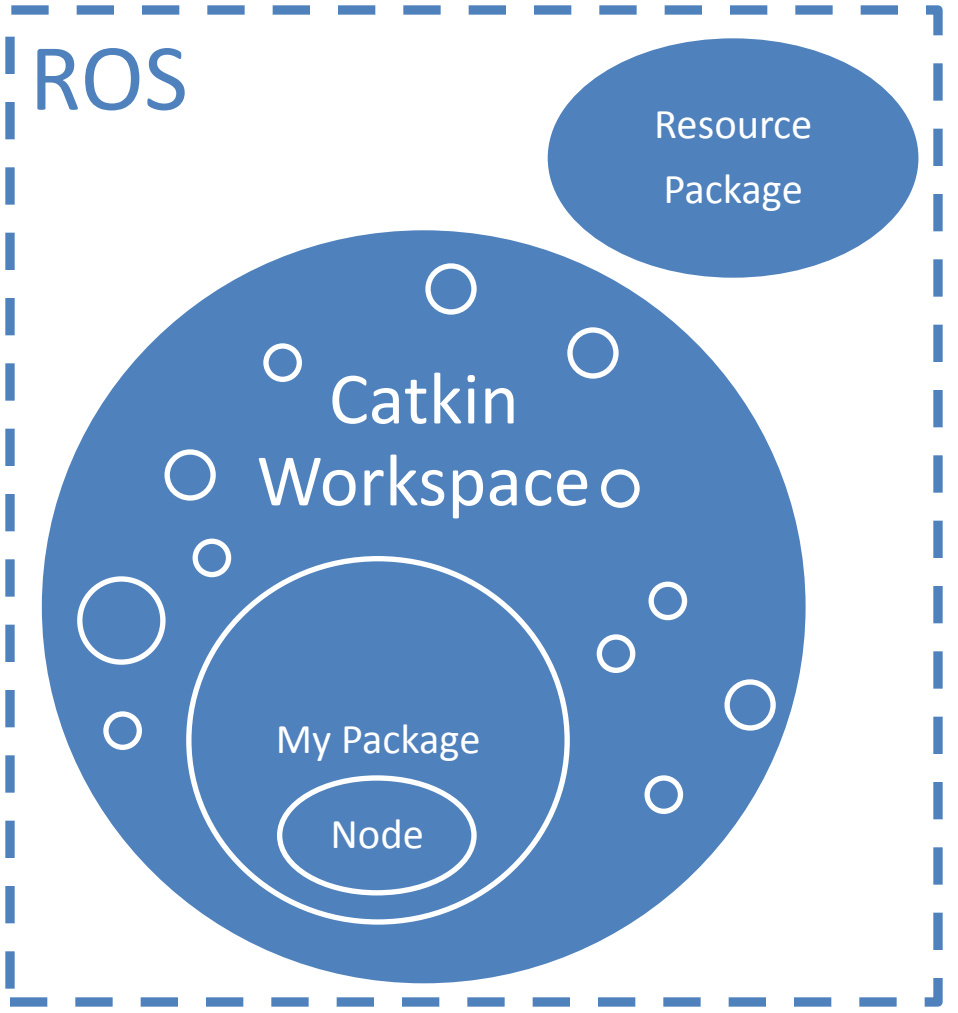
Exercise 1.0

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
  - ❑ rosrun
  - ❑ roslaunch



Creating a ROS Workspace

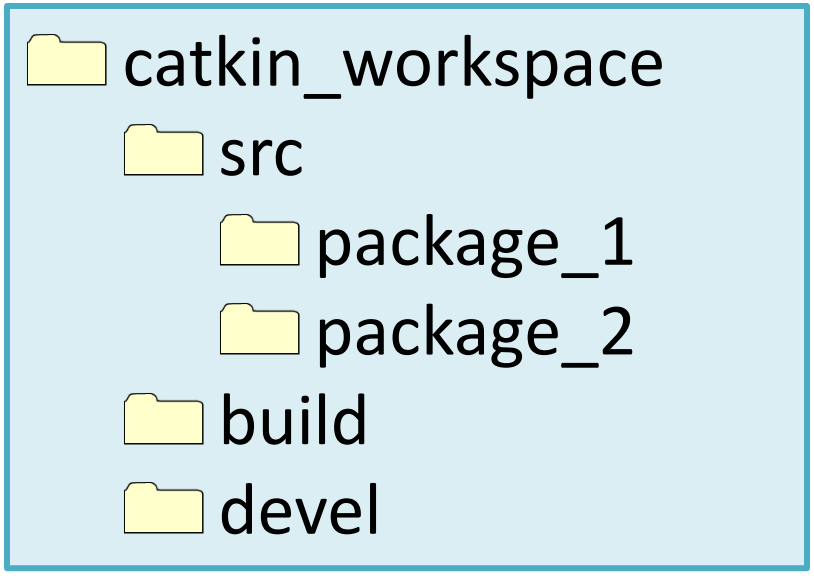
Catkin

- ROS uses the **catkin** build system
  - based on CMAKE
  - cross-platform (Ubuntu, Windows, embedded...)
  - replaces older **rosbuild** system
    - different build commands, directory structure, etc.
    - most packages have already been upgraded to catkin
    - rosbuild: `manifest.xml`, catkin: `package.xml`

http://www.clearpathrobotics.com/blog/introducing-catkin/

Catkin Workspace

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



Catkin Build Process

Setup (one-time)

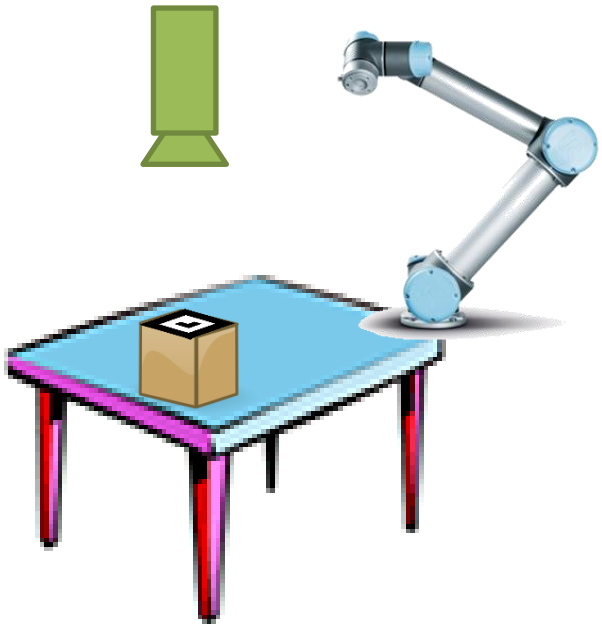
1. Create a catkin workspace somewhere
   - `catkin_ws`
   - `src` sub-directory must be created manually
   - `build`, `devel` directories created automatically
2. Run `catkin init` anywhere in the workspace
3. Download/create **packages** in `src` subdir

Compile-Time

1. Run `catkin build` anywhere in the workspace
2. Run `source devel/setup.bash` to make workspace visible to ROS
   - Must re-execute in **each** new terminal window
   - Can add to `~/.bashrc` to automate this process

Exercise 1.1

Create a Catkin Workspace



fake_ar_pub

myworkcell_node

myworkcell_support

vision_node

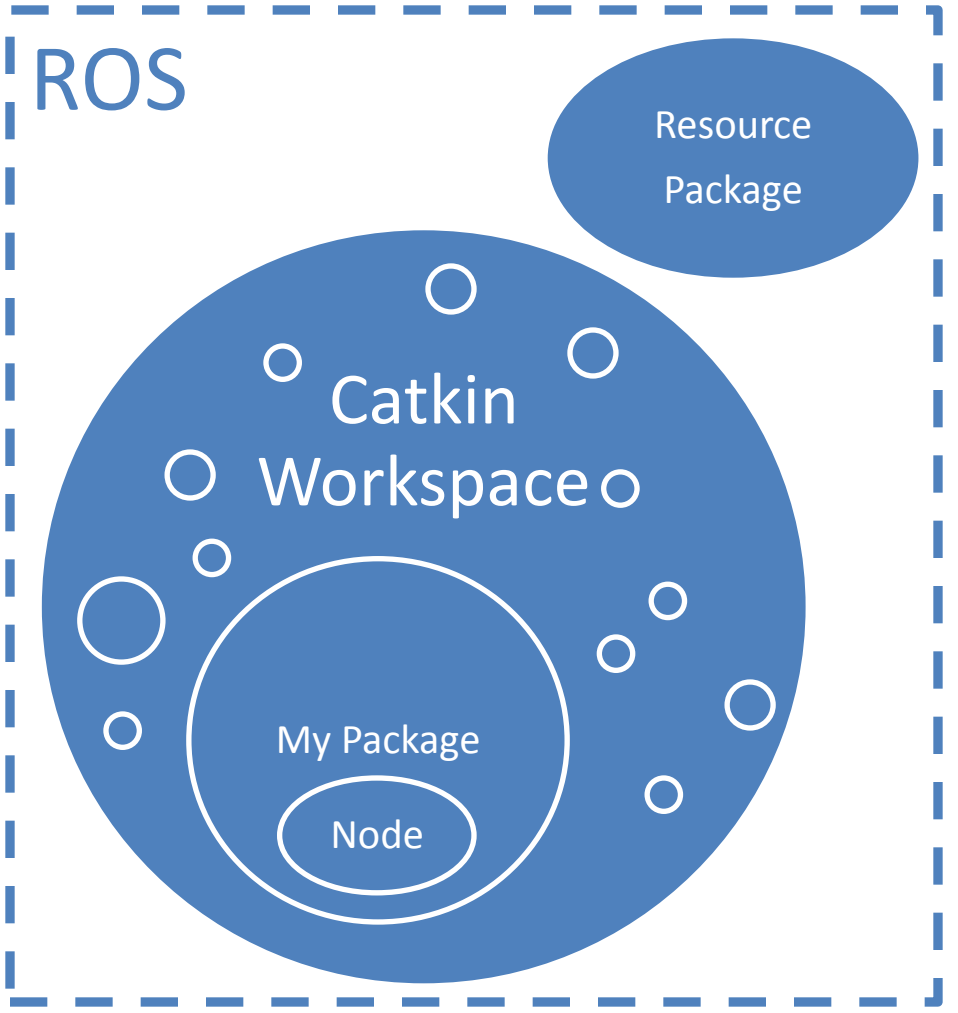
descartes_node

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
  - ❑ rosrun
  - ❑ roslaunch



Add 3rd-Party Packages

(a.k.a. “Resource” Packages)

Install options

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://ros.org/browse/)
  - Browse/Search for known packages

- ROS Answers (http://answers.ros.org)
  - When in doubt... ask someone!

Install using Debian Packages

```
sudo apt install ros-kinetic-package
```

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

Use "-" not "_"

- Fully automatic install:
  - Download .deb package from central ROS repository
  - Copies files to standard locations (`/opt/ros/kinetic/...`)
  - ➢ 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 catkin_ws/src
git clone http://github.com/user/repo.git
```

- Build your catkin workspace

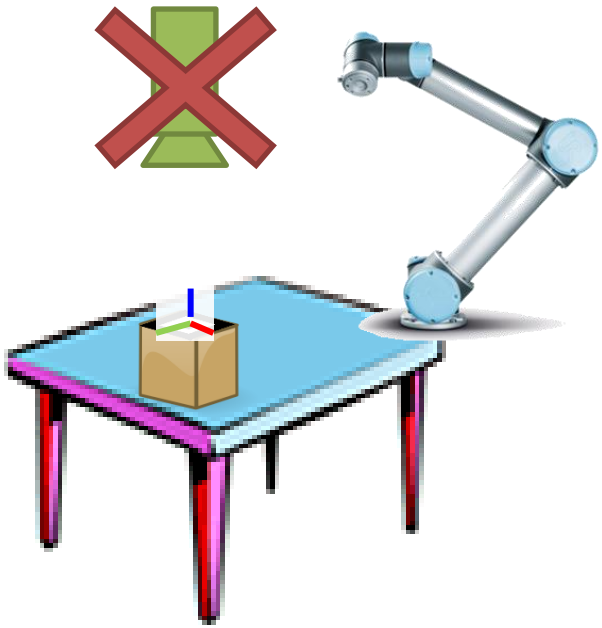
```
cd catkin_ws
catkin build
```

- Now the package and its resources are available to you

http://www.clearpathrobotics.com/blog/introducing-catkin/

Exercise 1.2

Install "resource" packages



fake_ar_pub

myworkcell_node

myworkcell_support

vision_node

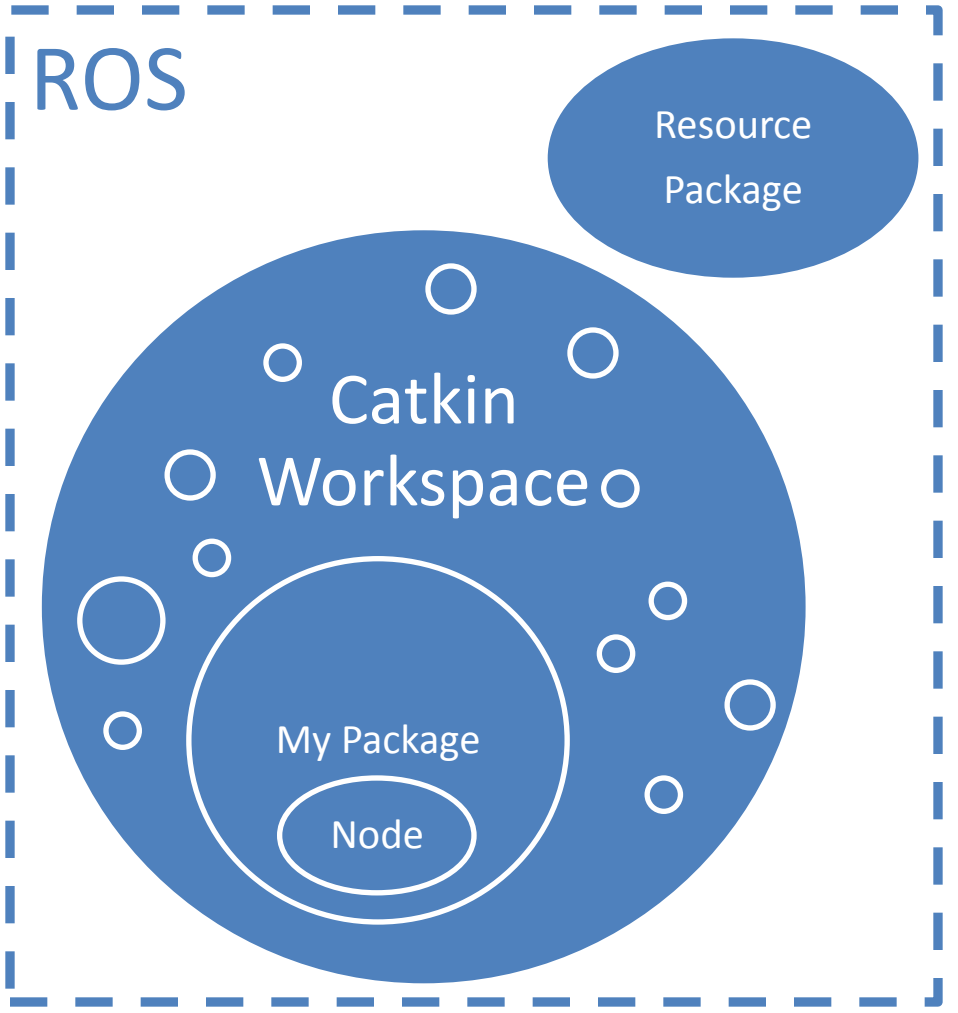
descartes_node

myworkcell_moveit_cfg

ur5_driver

Day 1 Progression

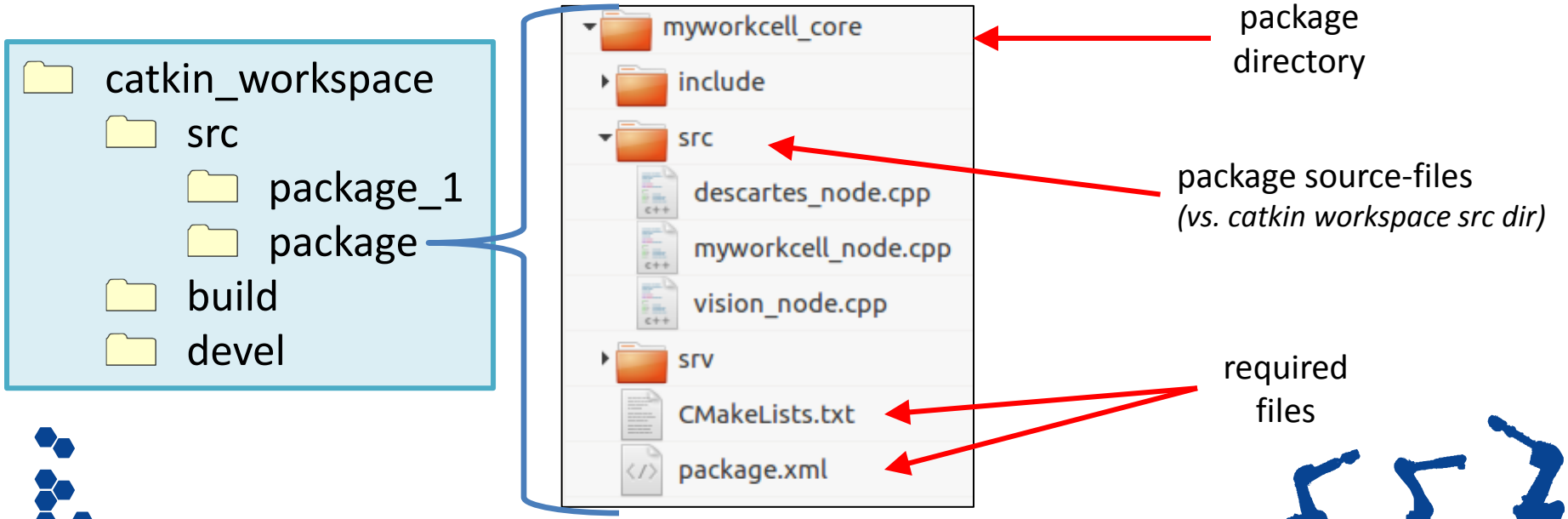
- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ❑ Create Package
- ❑ Create Node
  - ❑ Basic ROS Node
  - ❑ Interact with other nodes
    - ❑ Messages
    - ❑ Services
- ❑ Run Node
  - ❑ rosrun
  - ❑ roslaunch



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 catkin



package.xml

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

```
<package>
  <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>
```

package.xml

- Metadata: name, description, author, license ...
- Dependencies:
  - Common
    - `<buildtool_depend>`: Needed to **build** itself. (Typically ***catkin***)
    - `<build_depend>`: Needed to **build** this package.
    - `<exec_depend>`: Needed to **run** code in this package.
  - Uncommon
    - `<build_export_depend>`: Needed to **build against** this package.
    - `<test_depend>`: Only ***additional*** dependencies for unit tests.
    - `<doc_depend>`: Needed to generate documentation.
  - Future (format "2")
    - `<depend>`: Needed to **build**, **export**, and **execution** dependency.

CMakeLists.txt

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

```
include_directories(include ${catkin_INCLUDE_DIRS})
```

Adds directories to CMAKE include rules

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

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

```
target_link_libraries(myNode ${catkin_LIBRARIES})
```

Links node `myNode` to dependency libraries

ROS Package Commands

- `roscd package_name`

  Change to package directory

- **`rospack`**
  - `rospack find package_name`

    Find directory of `package_name`
  - `rospack list`

    List all ros packages installed
  - `rospack depends package_name`

    List all dependencies of `package_name`

Create New Package

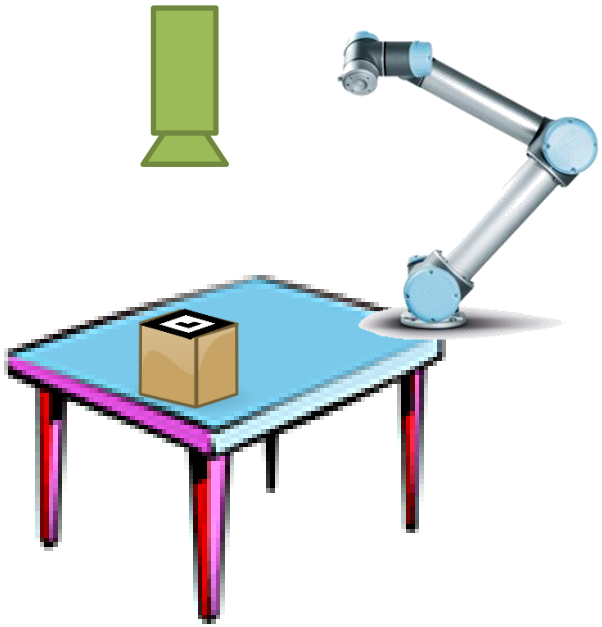
```
catkin create pkg mypkg --catkin-deps dep1 dep2
```

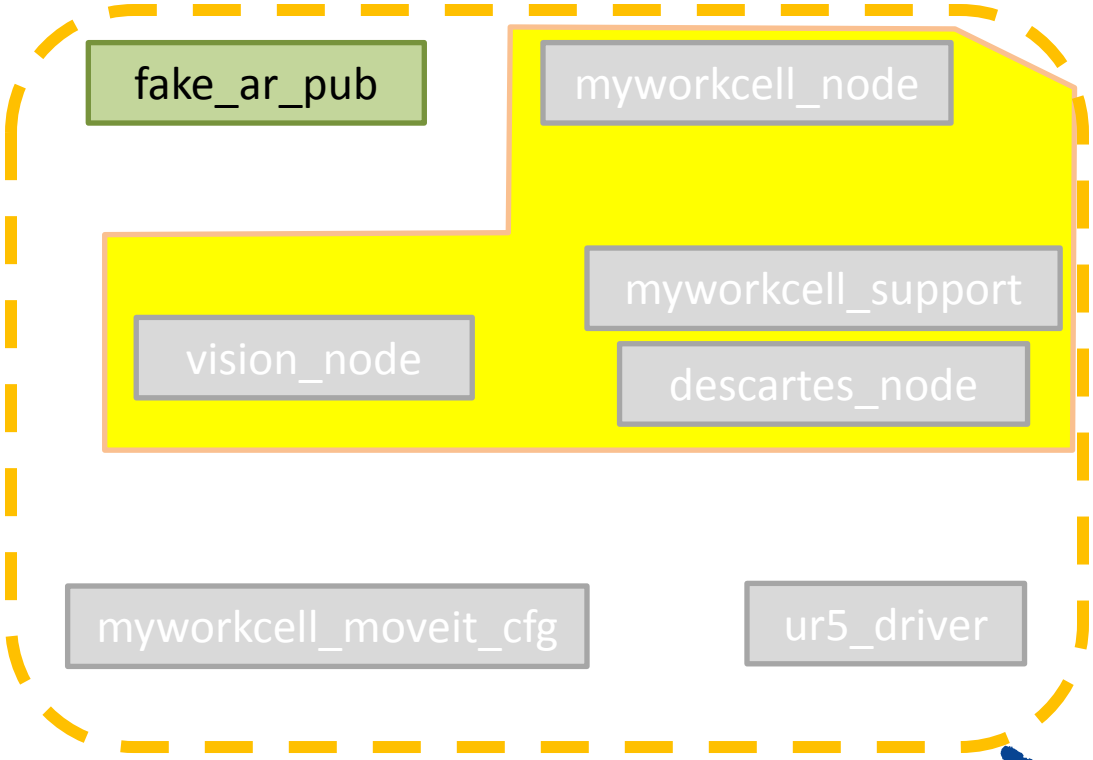
Easiest way to start a new package

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

Exercise 1.3.1

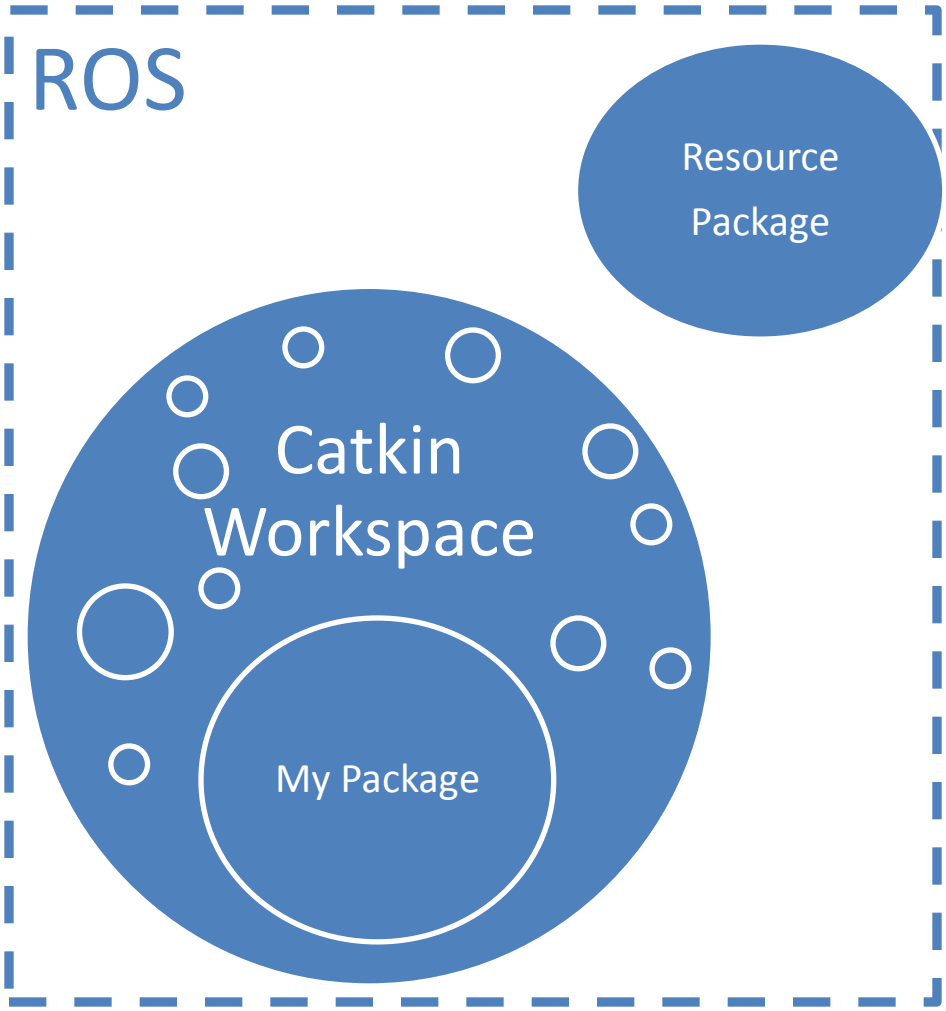
Create Package





Day 1 Progression

- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ✓ Create Package
- ❑ Create Node
  - ❑ Basic ROS Node
  - ❑ Interact with other nodes
    - ❑ Messages
    - ❑ Services
- ❑ Run Node
  - ❑ rosrun
  - ❑ roslaunch



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++ ROS Node

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

int main(int argc, char* argv[])
{
  ros::init(argc, argv, "hello");
  ros::NodeHandle node;

  ROS_INFO_STREAM("Hello World!");

  return 0;
}
```

ROS Node Commands

- `rosrun package_name node_name`

  execute ROS node

- **`rosnode`**
  - `rosnode list`

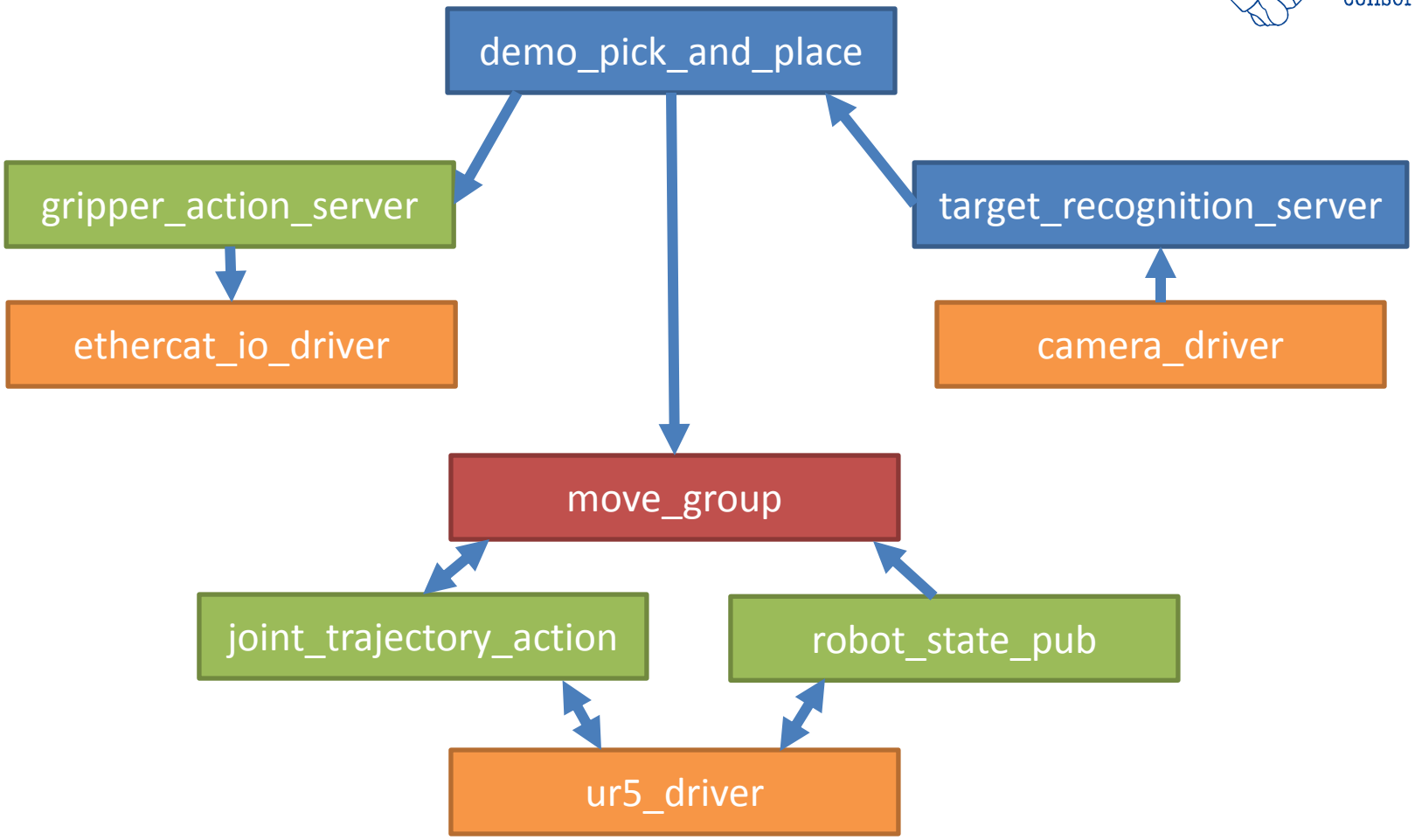
    View running nodes
  - `rosnode info node_name`

    View node details (publishers, subscribers, services, etc.)
  - `rosnode kill node_name`

    Kill running node; good for remote machines

    ➢ `Ctrl+C` *is usually easier*

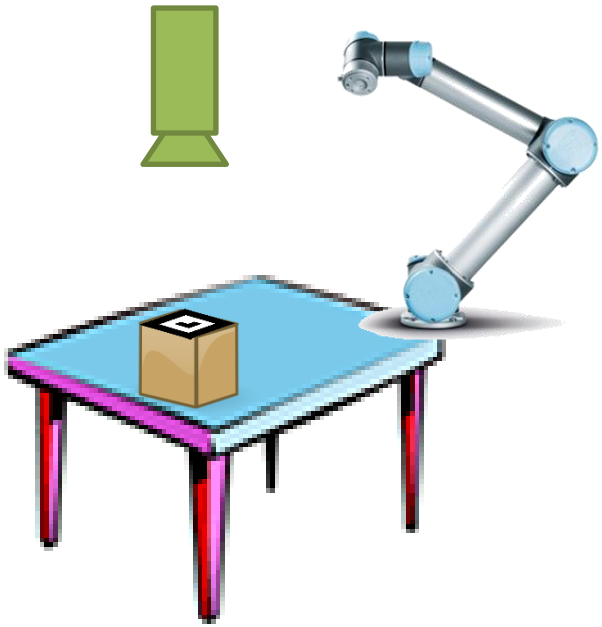
“Real World” – Nodes

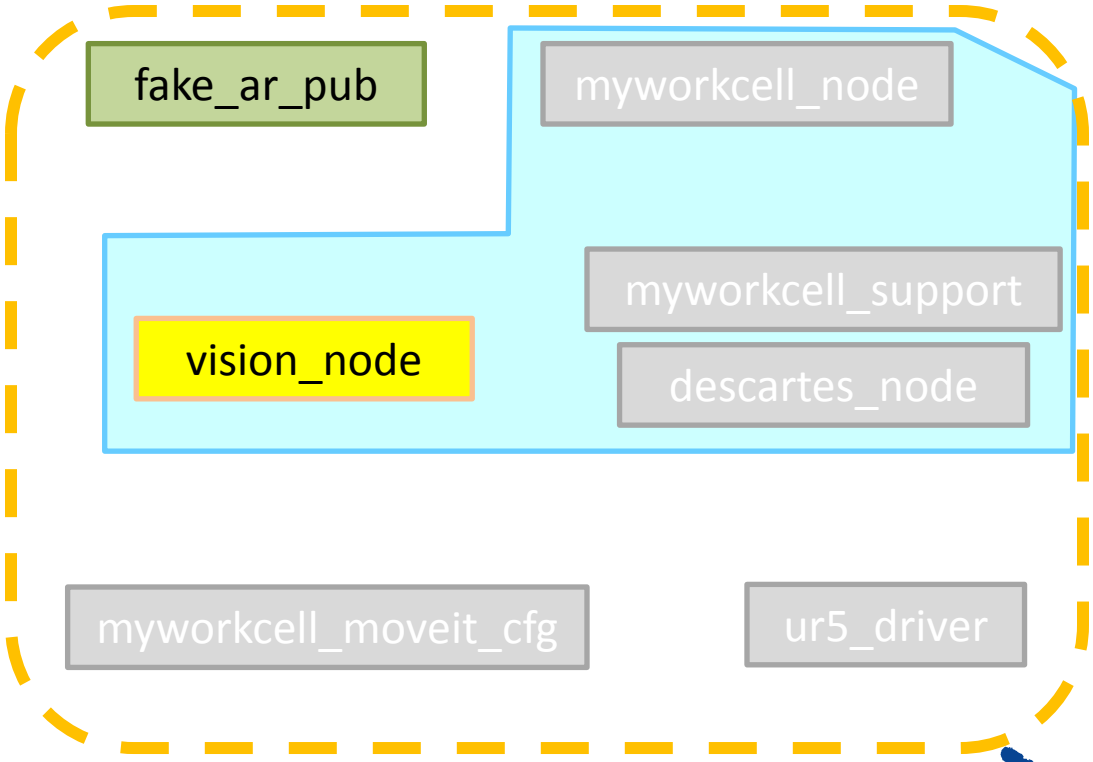


Exercise 1.3.2

Create a Node:

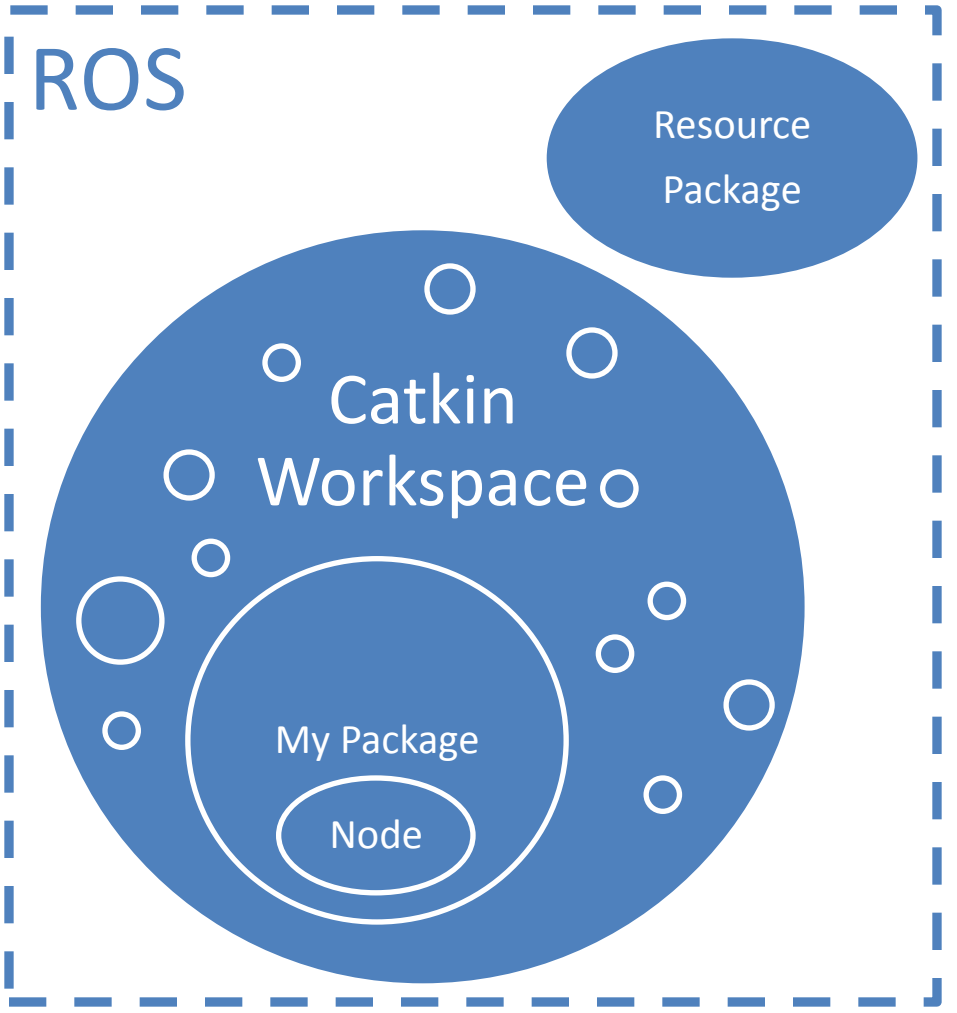
In myworkcell_core package called vision_node





Day 1 Progression

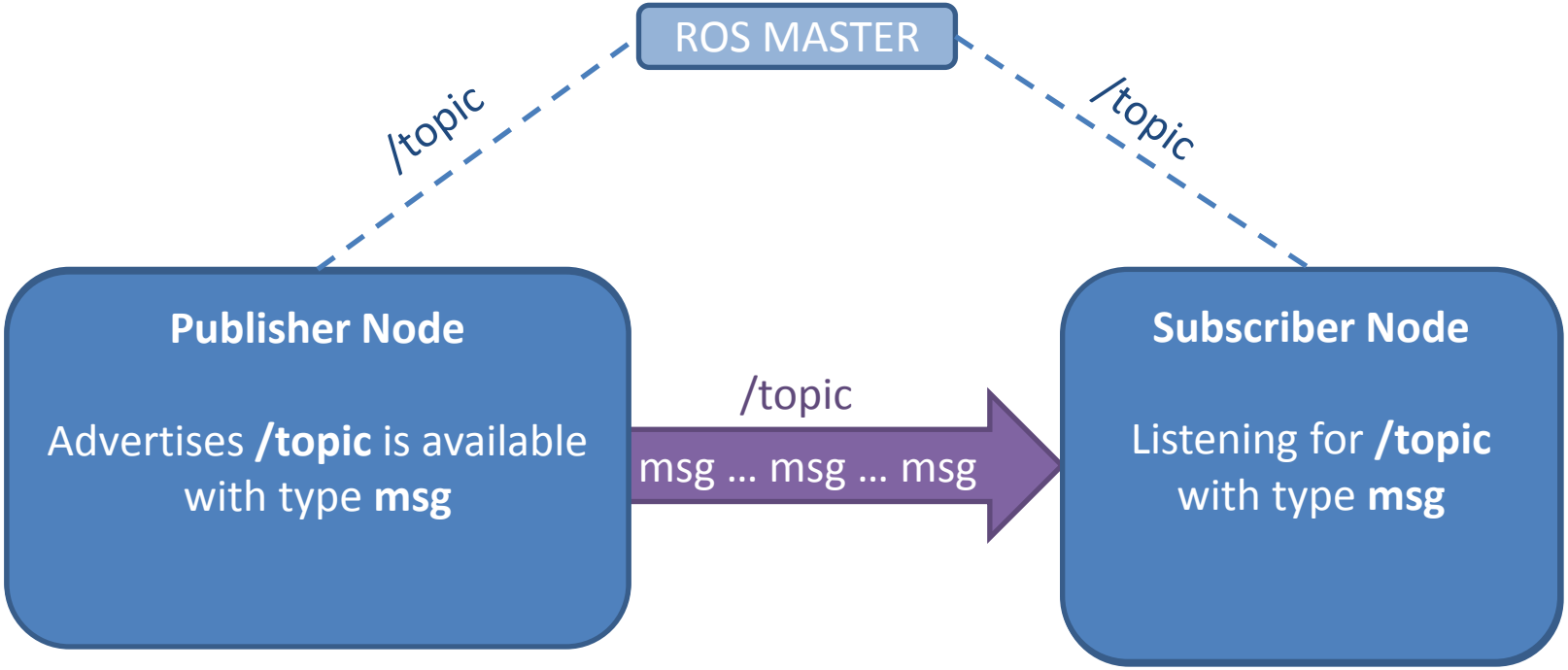
- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ✓ Create Package
- ✓ Create Node
  - ✓ Basic ROS Node
  - ❑ Interact with other nodes
    - ❑ Messages
    - ❑ Services
- ✓ Run Node
  - ✓ rosrun
  - ❑ roslaunch



Topics and Messages

ROS Topics/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

/Basler1/image_rect

Basler Camera Node → sensor_msgs/Image

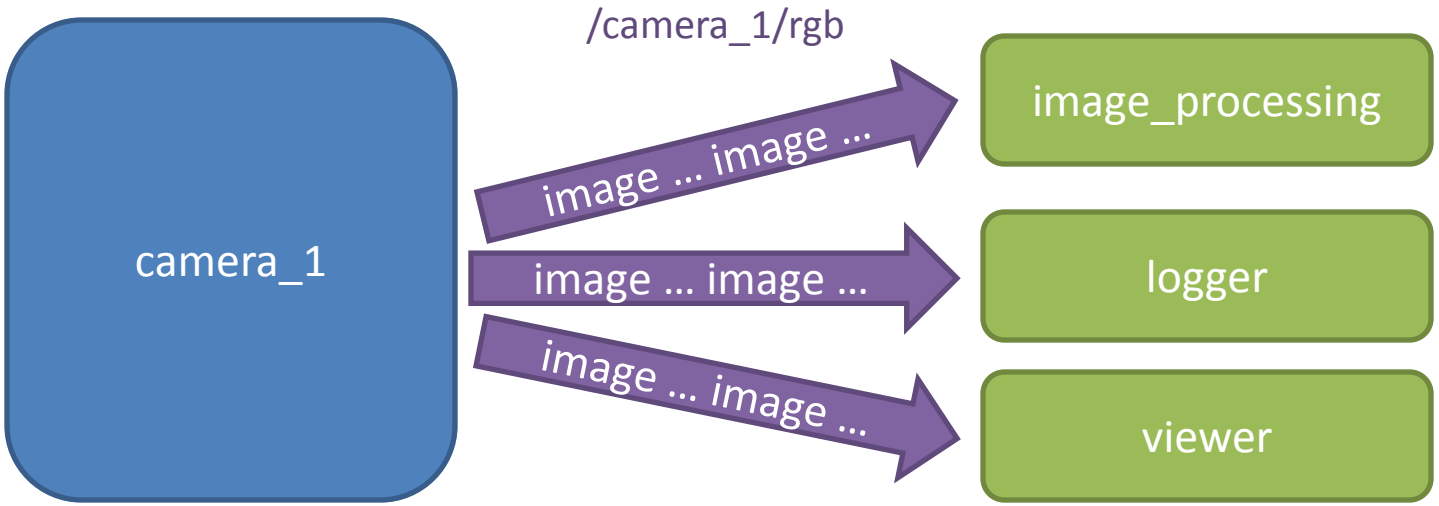
/Basler2/image_rect

Basler Camera Node → sensor_msgs/Image

Calibration Node Subscribes to Images from:
/Basler1/image_rect
/Basler2/image_rect
/Basler3/image_rect
...

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

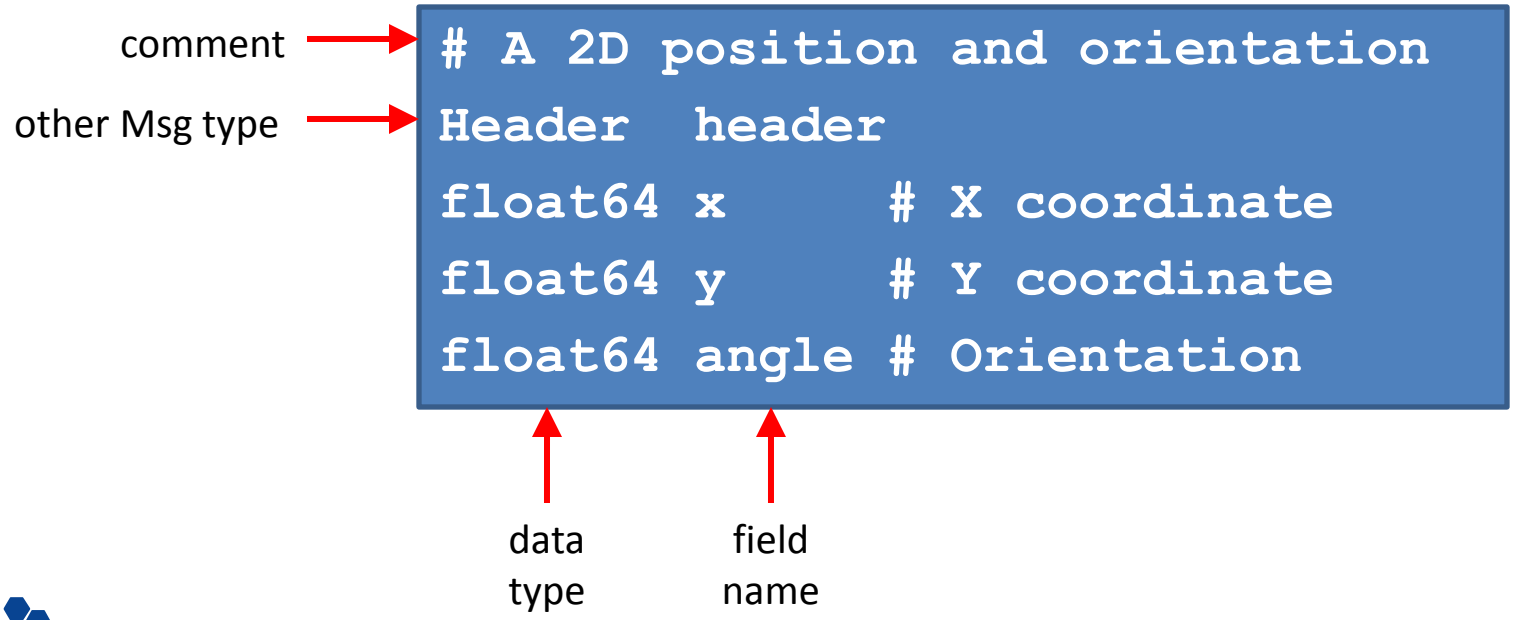
ROS Messages Types

- 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

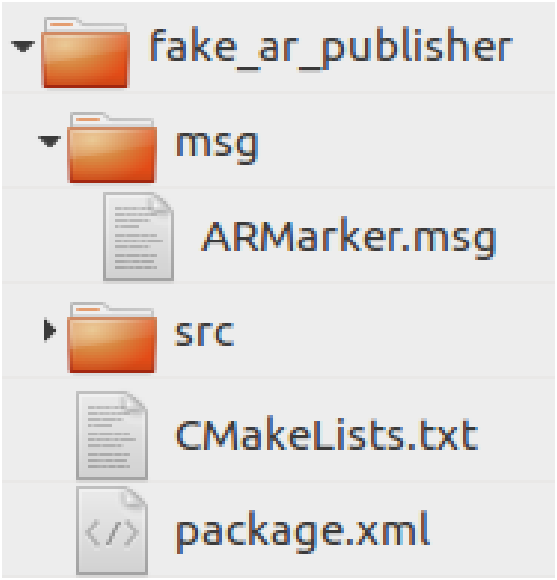
PathPosition.msg



Custom ROS Messages

- Custom message types are defined in `msg` subfolder of packages

- Modify `CMakeLists.txt` to enable message generation.



CMakeLists.txt

- Lines needed to generate custom msg types

```
find_package(catkin REQUIRED COMPONENTS
 message_generation)

add_message_files(custom.msg ...)

generate_messages(DEPENDENCIES ...)

catkin_package(CATKIN_DEPENDS roscpp
message_runtime)
```

package.xml

<build_depend> **message_generation** </build_depend>
<run_depend>**message_runtime**</run_depend>

ROS Message Commands

- `rosmsg list`
  - Show all ROS topics currently installed on the system
- `rosmsg package <package>`
  - Show all ROS message types in package `<package>`
- `rosmsg show <package>/<message_type>`
  - Show the structure of the given message type

ROS Topic Commands

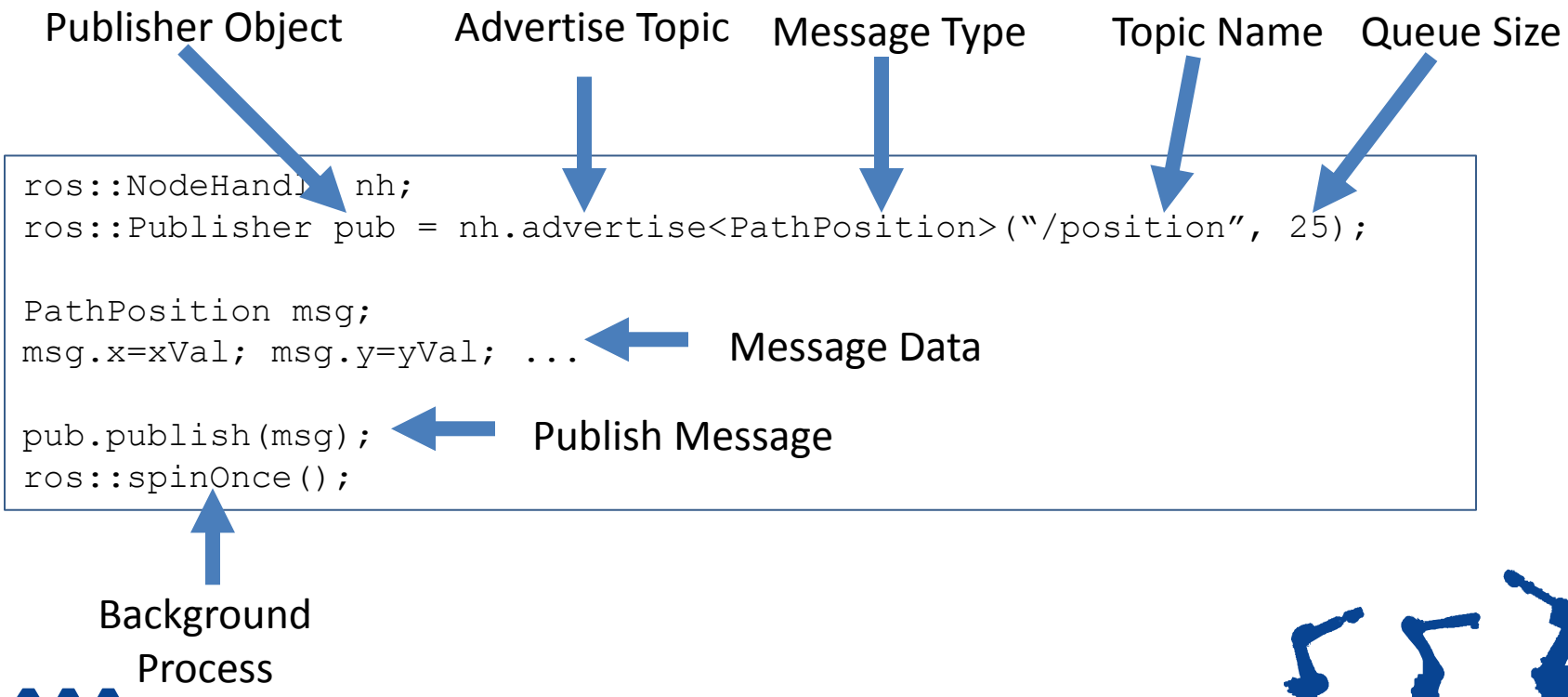
- `rostopic list`
  - List all topics currently subscribed to and/or publishing
- `rostopic type <topic>`
  - Show the message type of the topic
- `rostopic info <topic>`
  - Show topic message type, subscribers, publishers, etc.
- `rostopic echo <topic>`
  - Echo messages published to the topic to the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

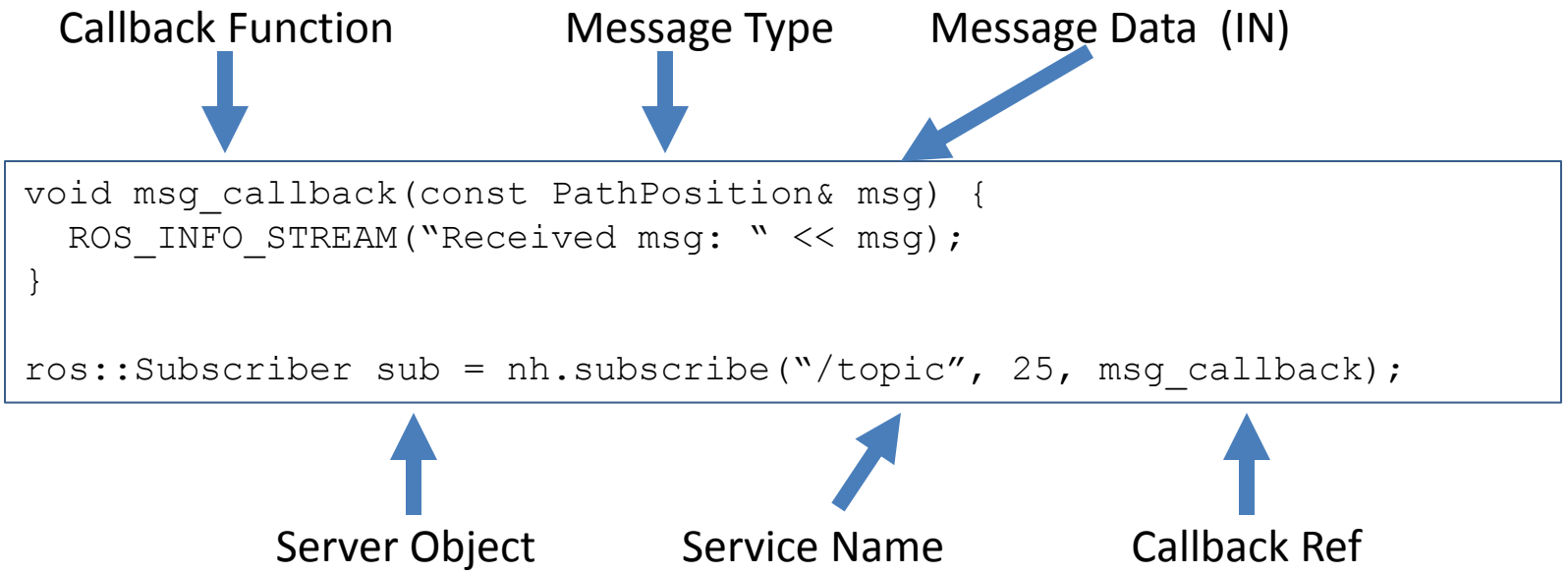
Topics: Syntax

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



Topics: Syntax

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



Instead of text editor and building from terminal…

Use an IDE! *Wiki instructions here*

Exercise 1.4

Subscribe to fake_ar_publisher