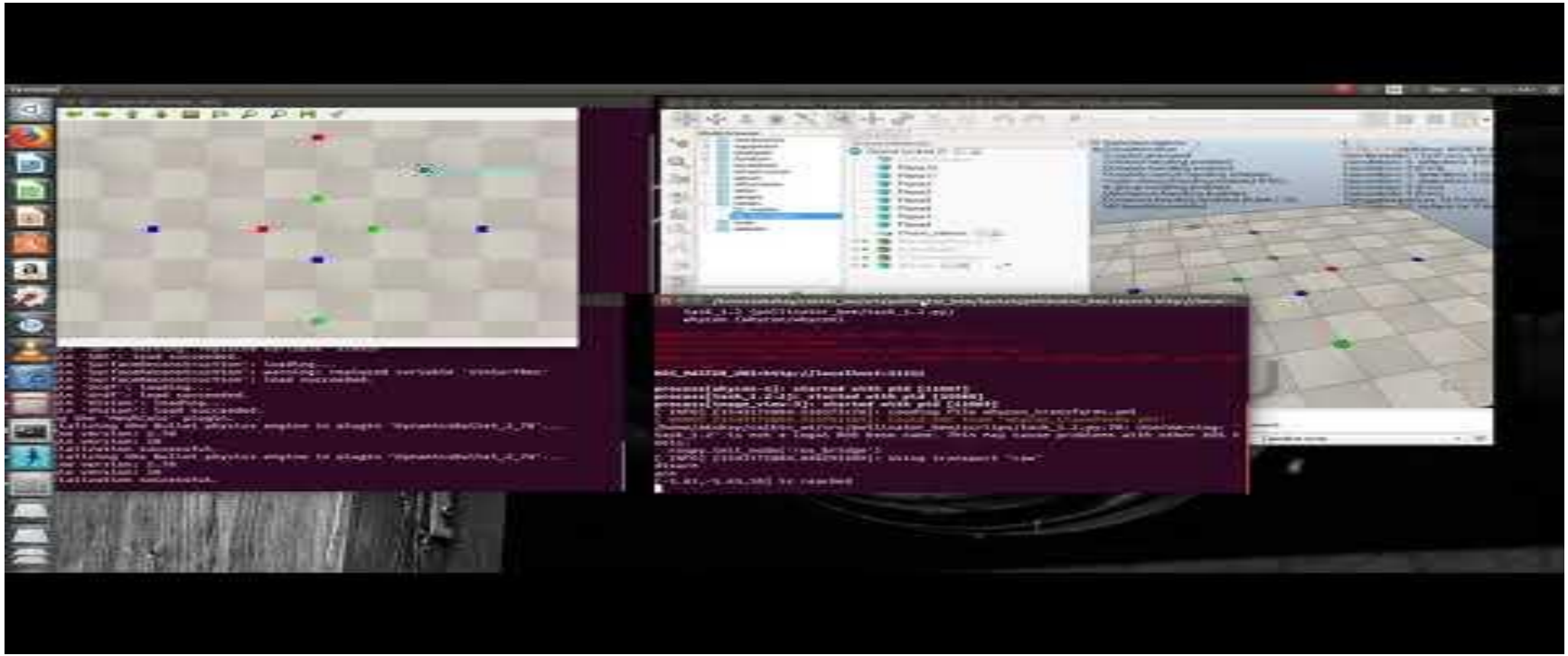

# Robot Operating System

AKSHAY S RAO | 1BM17EC007

# What is ROS?

- ROS is an open-source robot operating system.
- A set of software libraries and tools that help you build robot applications that work across a wide variety of robotic platforms.
- Originally developed in **2007** at the Stanford Artificial Intelligence Laboratory and development continued at Willow Garage.
- Since **2013** managed by **OSRF** (Open Source Robotics Foundation).

# Lets see ROS in Action First!



# ROS Main Features

ROS has two “sides”

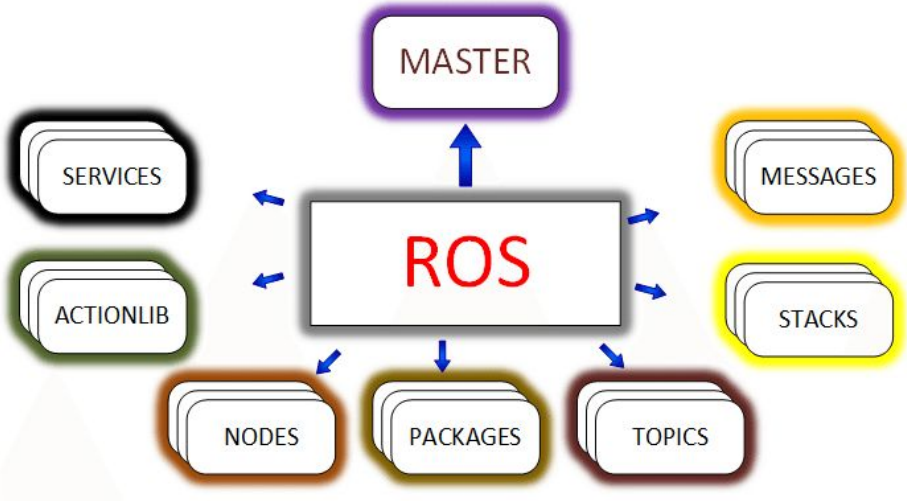
- The *operating system side*, which provides standard operating system services such as: **hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, package management.**
- A suite of *user contributed packages* that implement common robot functionality such as SLAM, planning, perception, vision, manipulation, etc.

# ROS Philosophy

- **Peer to Peer**
  - ROS systems consist of many small programs (nodes) which connect to each other and continuously exchange messages.
- **Tools-based**
  - There are many small, generic programs that perform tasks such as visualization, logging, plotting data streams, etc.
- **Multi-Lingual**
  - ROS software modules can be written in any language for which a client library has been written. Currently client libraries exist for C++, **Python,** LISP, Java, JavaScript, MATLAB, Ruby, and more.
- **Thin**
  - The ROS conventions encourage contributors to create stand-alone libraries/packages and then wrap those libraries so they send and receive messages to/from other ROS modules.
- **Free & open source, community-based, repositories**

# ROS Core Concepts

- Nodes
- Messages and Topics
- Services
- Actions
- ROS Master
- Parameters
- Packages and Stacks



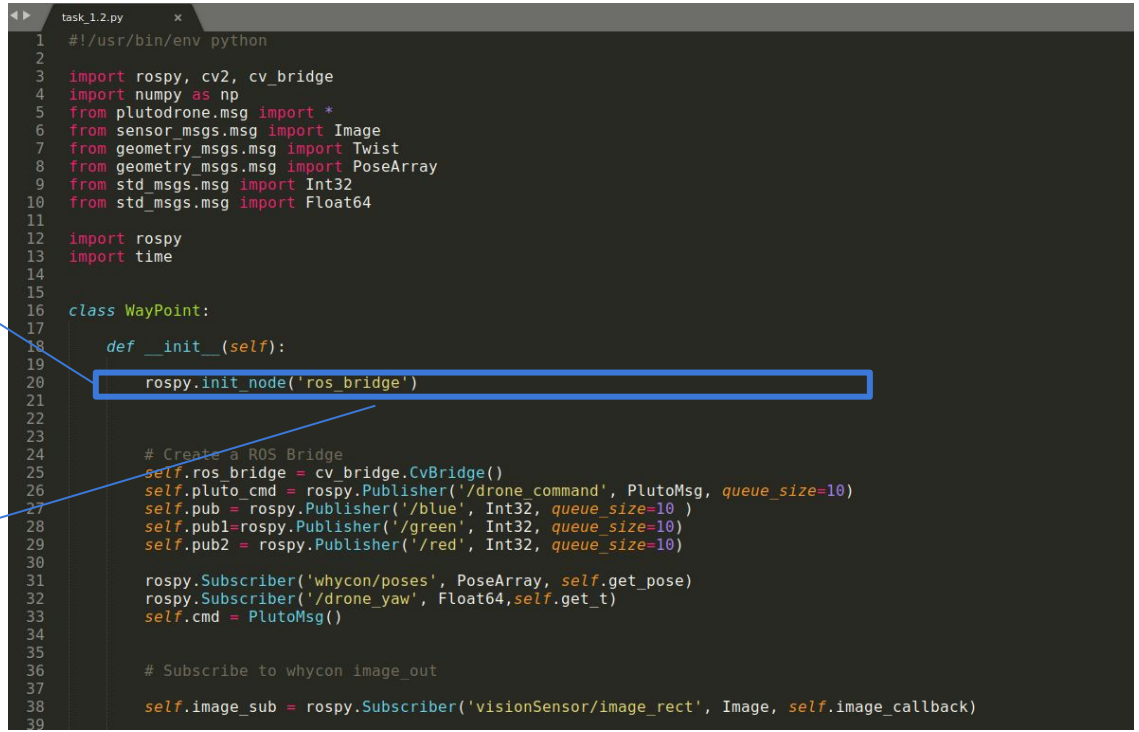
# ROS Nodes

- Single-purposed executable programs.
  - e.g. sensor driver(s), actuator driver(s), map building, planner, UI, etc.
- Individually compiled, executed, and managed.
- Nodes are written using a ROS client library.
  - **rospy – python client library.**
  - roscpp – C++ client library.
- Nodes can publish or subscribe to a Topic.
- Nodes can also provide or use a Service or an Action.

# ROS Node Example

ROS Node

ROS Node name



task_1.2.py

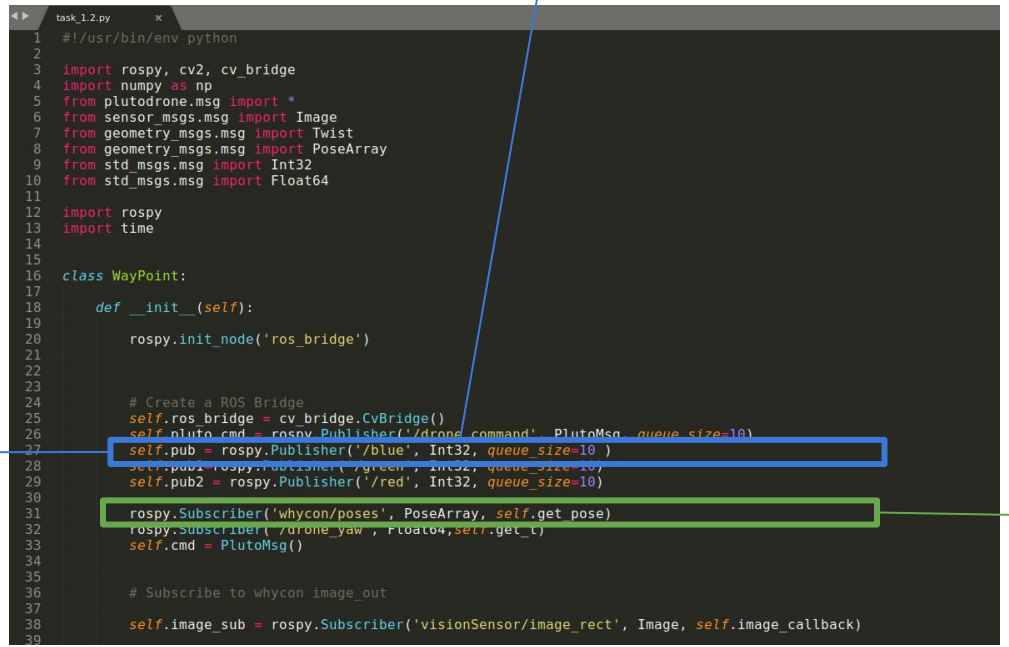
```
#!/usr/bin/env python

import rospy, cv2, cv_bridge
import numpy as np
from plutodrone.msg import *
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist
from geometry_msgs.msg import PoseArray
from std_msgs.msg import Int32
from std_msgs.msg import Float64

import rospy
import time


class WayPoint:

    def __init__(self):

        rospy.init_node('ros_bridge')



        # Create a ROS Bridge
        self.ros_bridge = cv_bridge.CvBridge()
        self.pluto_cmd = rospy.Publisher('/drone_command', PlutoMsg, queue_size=10)
        self.pub = rospy.Publisher('/blue', Int32, queue_size=10 )
        self.pub1=rospy.Publisher('/green', Int32, queue_size=10)
        self.pub2 = rospy.Publisher('/red', Int32, queue_size=10)

        rospy.Subscriber('whycon/poses', PoseArray, self.get_pose)
        rospy.Subscriber('/drone_yaw', Float64,self.get_t)
        self.cmd = PlutoMsg()


        # Subscribe to whycon image_out

        self.image_sub = rospy.Subscriber('visionSensor/image_rect', Image, self.image_callback)
```

# ROS Topics and ROS Messages

- **Topic**: named stream of messages with a defined type.
  - Data from a range-finder might be sent on a topic called scan, with a message of type LaserScan.
- Nodes communicate with each other by publishing messages to topics.
- **Publish/Subscribe model:** 1-to-N broadcasting.
- **Messages**: Strictly-typed data structures for inter-node communication
  - geometry_msgs/Twist is used to express velocity commands: Vector3 linear, Vector3 angular

# ROS Topic & Message Example

ROS message type



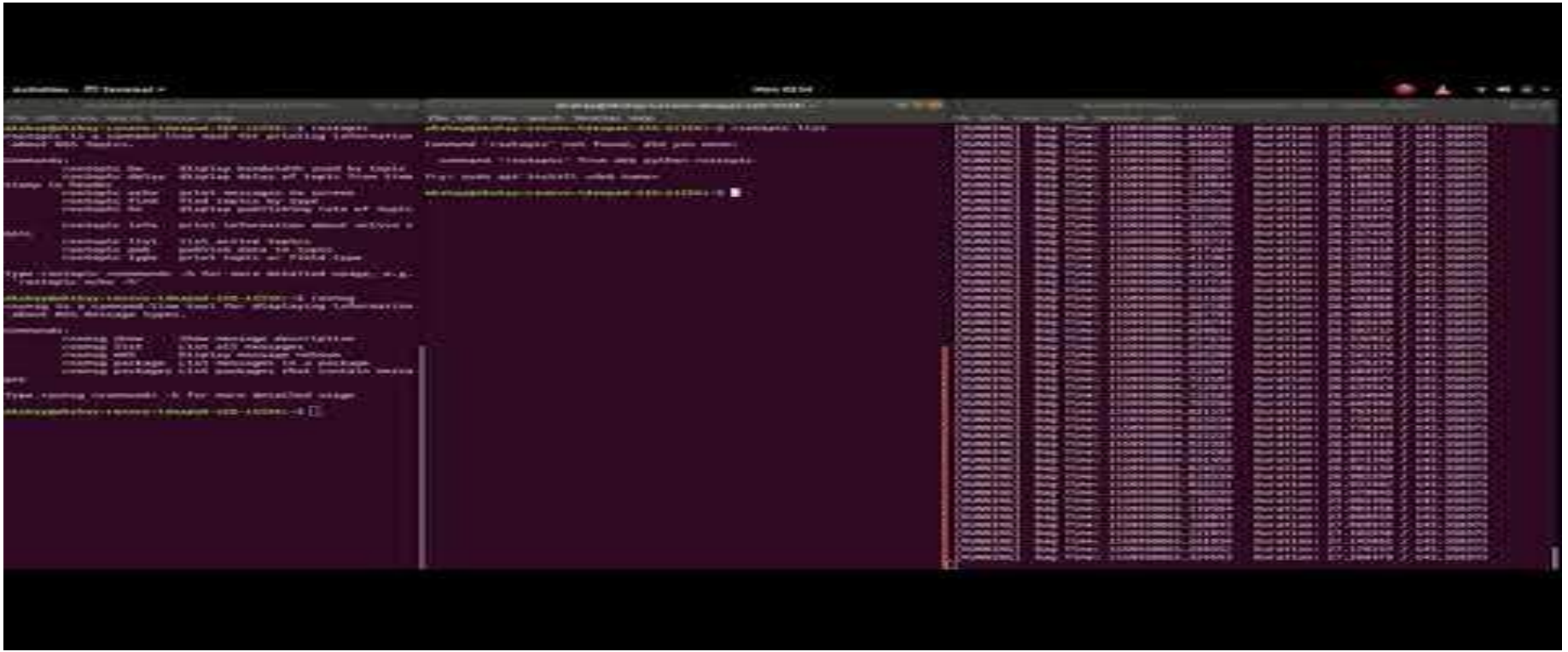

Publisher Topic

Subscriber Topic

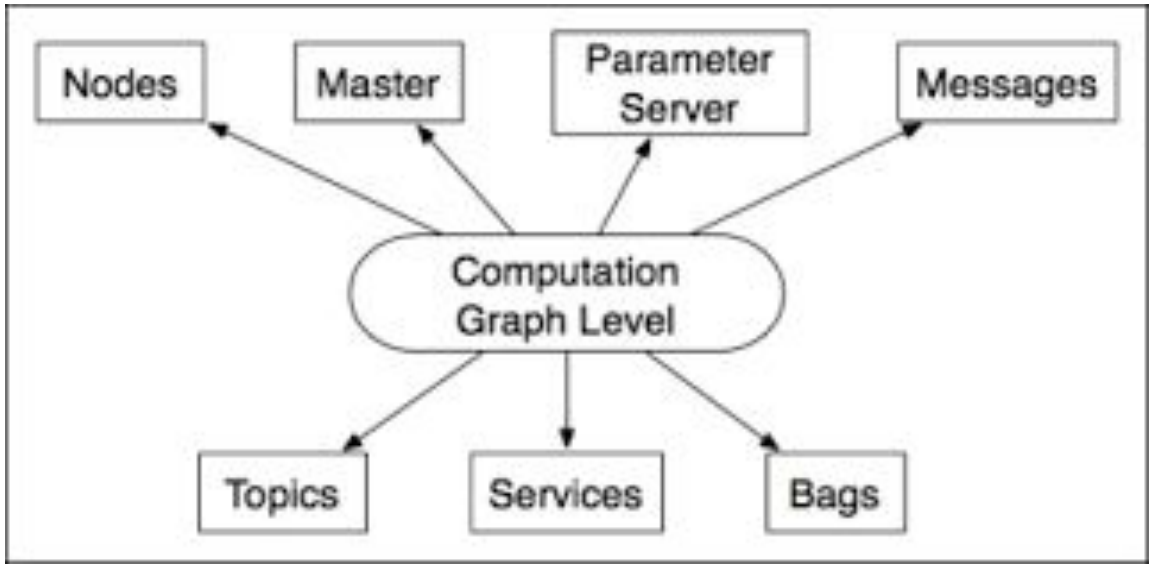
# ROS Bags

- Bags are the primary mechanism in ROS for **data logging.**
- Bags subscribe to one or more ROS **topics**, and **store the serialized message** data in a file as it is received.
- Bag files can also be **played back** in ROS to the same topics they were recorded from, or even remapped to new topics.

# ROS Bag Example



# ROS Computational Graph level

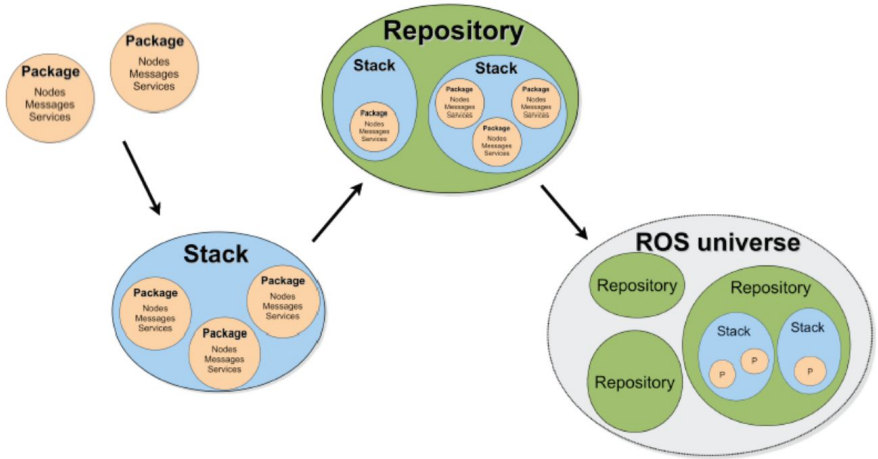


# ROS Supported Platforms

- ROS is currently supported only on Ubuntu
- other variants such as Windows, Mac OS X, and Android are considered experimental.

# ROS Packages

- Software in ROS is organized in packages.
- A package contains one or more nodes, documentation, and provides a ROS interface.
- Most of ROS packages are hosted in GitHub.



# ROS Package and Catkin Workspace

- Packages are the most atomic unit of build and the unit of release.
- A package contains the source files for one node or more and configuration files.
- A ROS package is a directory inside a catkin workspace that has a package.xml file in it.
- A catkin workspace is a set of directories in which a set of related ROS code/packages live (catkin ROS build system: CMake + Python scripts).
- It’s possible to have multiple workspaces, but work can performed on only one-at-a-time.

# Catkin Workspace Folders

- Source space: **workspace_folder/src**
- Build space: **workspace_folder/build**
- Development space: **workspace_folder/devel**
- Install space: **workspace_folder/install**

| | |
|---|---|
| Source space | Contains the source code of catkin packages. Each folder within the source space contains one or more catkin packages. |
| Build Space | is where CMake is invoked to build the catkin packages in the source space. CMake and catkin keep their cache information and other intermediate files here. |
| Development (Devel) Space | is where built targets are placed prior to being installed |
| Install Space | Once targets are built, they can be installed into the install space by invoking the install target. |

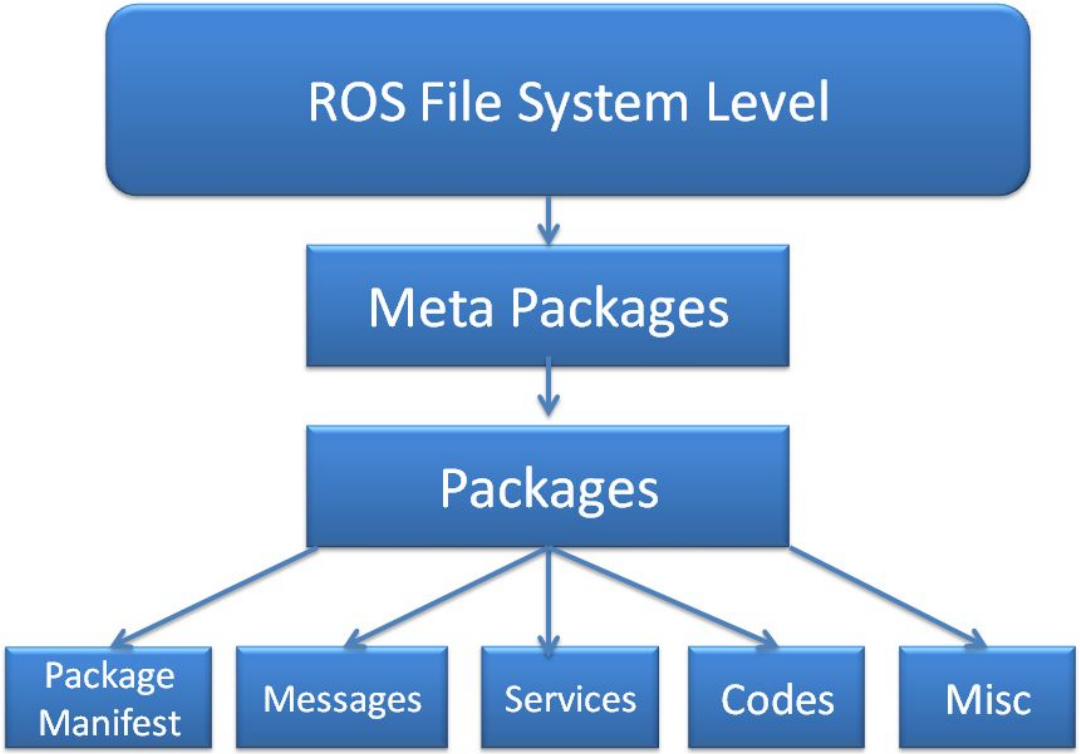
# ROS Package Files

- Layout of the src/my_package folder in a catkin workspace:

| Directory | Explanation |
|---|---|
| include/ | C++ include headers |
| src/ | Source files |
| msg/ | Folder containing Message (msg) types |
| srv/ | Folder containing Service (srv) types |
| launch/ | Folder containing launch files |
| package.xml | The package manifest |
| CMakeLists.txt | CMake build file |

- Source files implement nodes, can be written in multiple languages
- Nodes are launched individually or in groups, using launch files

# ROS File System Level



# ROS Community Level