16-311-Q Introduction to Robotics

Lab Lecture 1: Introduction to ROS

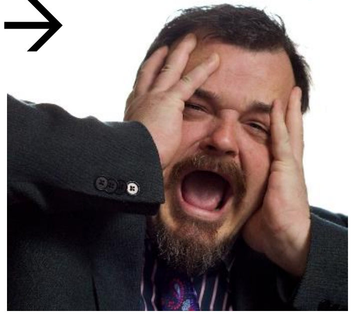
Instructor:
Gianni A. Di Caro

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

PROBLEM(S) IN ROBOTICS DEVELOPMENT

In Robotics, **before ROS**

- Lack of standards
- Little code reusability
- Keeping reinventing (or rewriting) device drivers, access to robot’s interfaces, management of on-board processes, inter-process communication protocols, …
- Keeping re-coding standard algorithms
- New robot in the lab (or in the factory) → start re-coding (mostly) from scratch

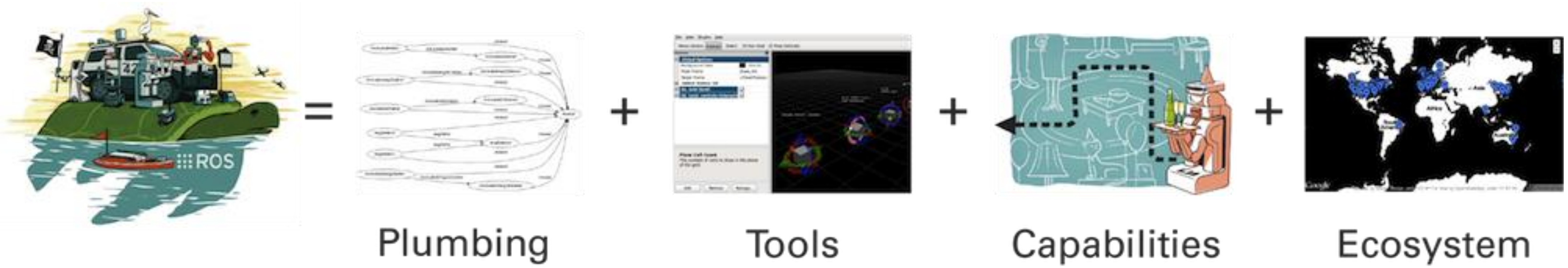


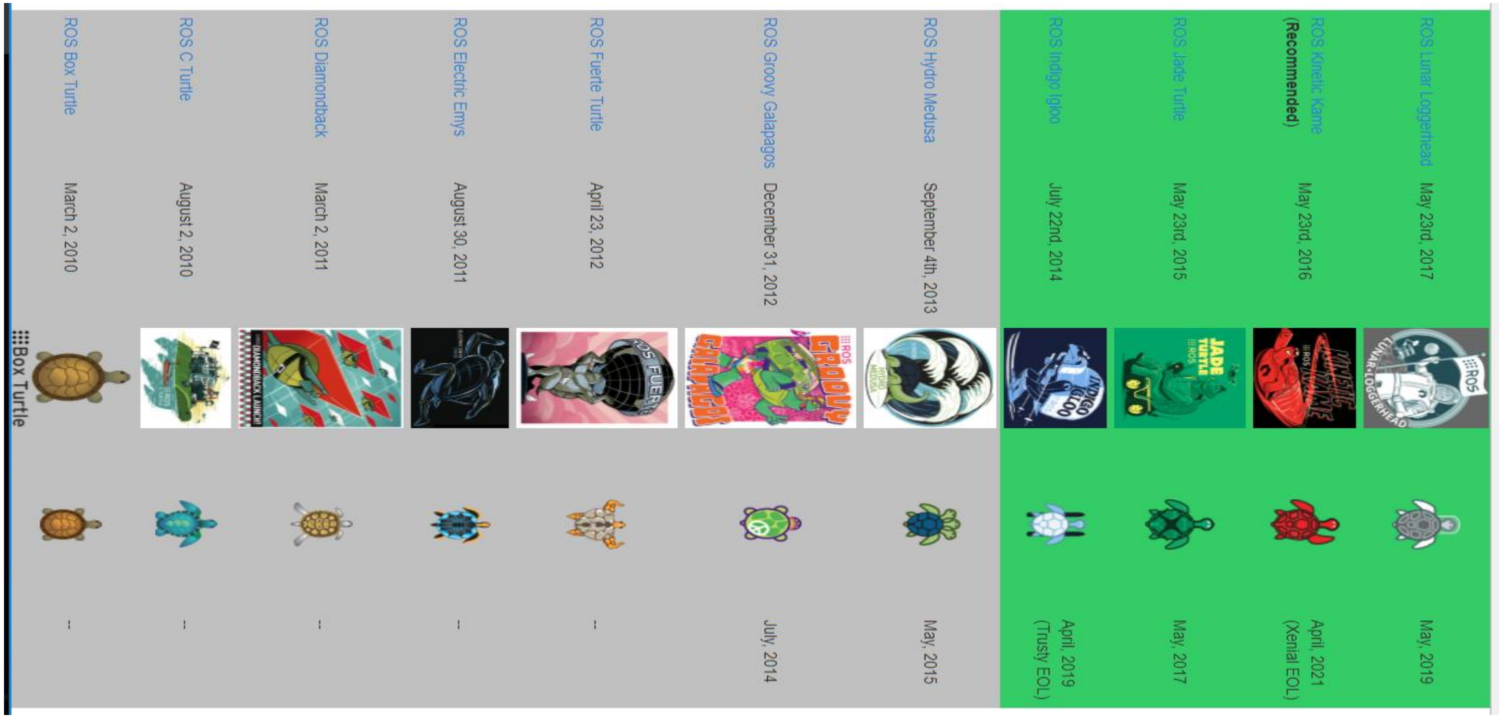
Robot Operating System (ROS)

ROS

http://www.ros.org

Open Source Robotics Foundation





ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame (Recommended)	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			-
ROS Electric Emys	August 30, 2011			-
ROS Diamondback	March 2, 2011			-
ROS C Turtle	August 2, 2010			-
ROS Box Turtle	March 2, 2010	Box Turtle		-

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)

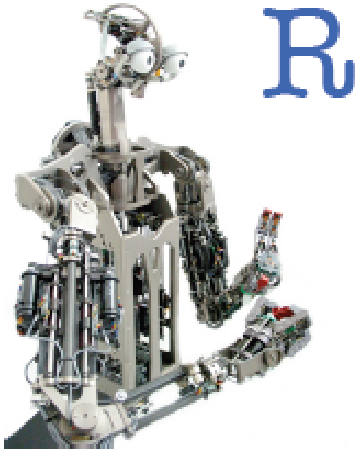
Note: Some of the following slides are adapted from **Roi Yehoshua**

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 Main Features



ROS

navigation

task executive

visualization

simulation

perception

control

planning

data logging

message passing

device drivers

real-time capabilities

web browser

OS

email client

window manager

memory management

process management

scheduler

device drivers

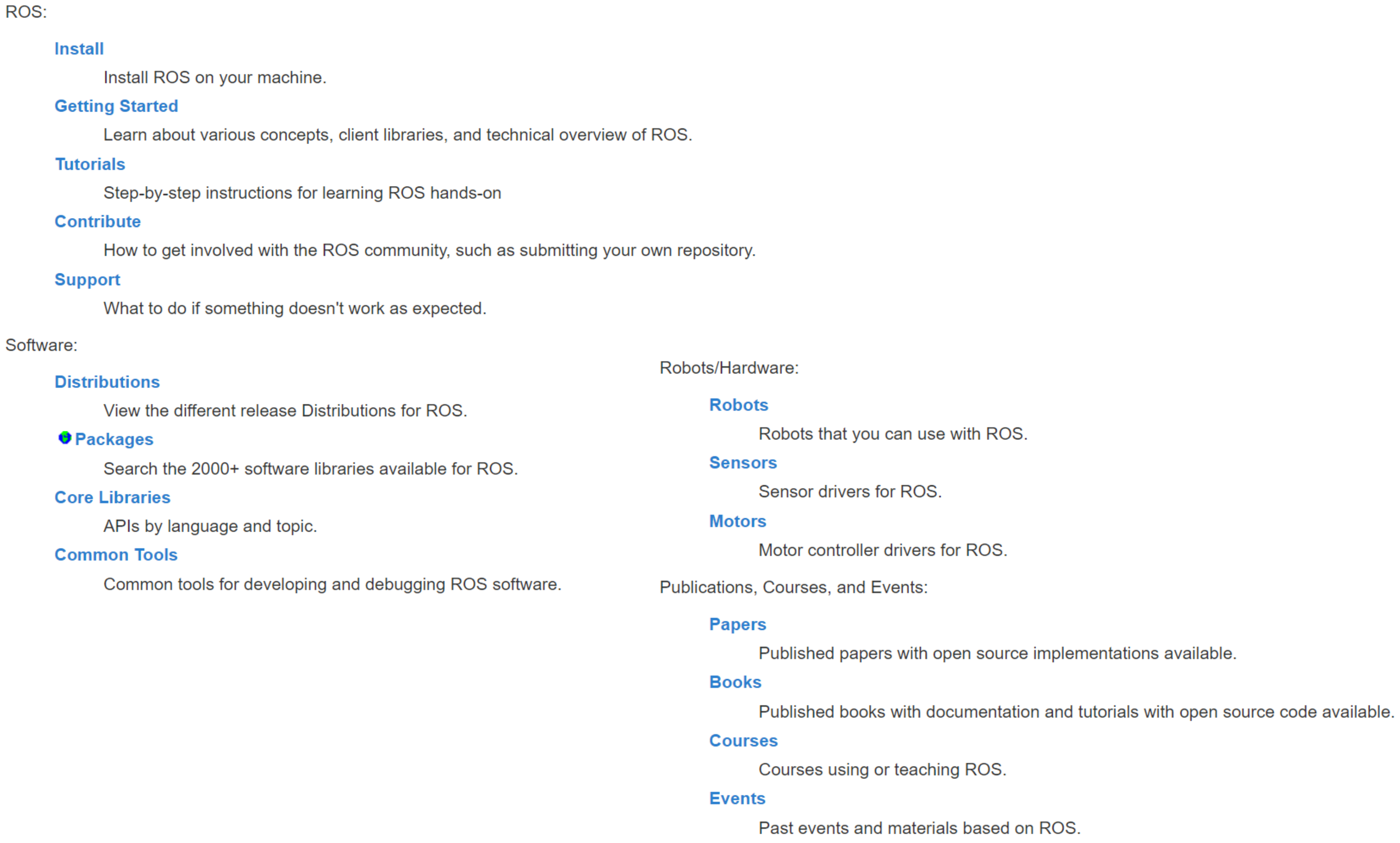
file system

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 WIKI

- http://wiki.ros.org/



ROS:

Install

Install ROS on your machine.

Getting Started

Learn about various concepts, client libraries, and technical overview of ROS.

Tutorials

Step-by-step instructions for learning ROS hands-on

Contribute

How to get involved with the ROS community, such as submitting your own repository.

Support

What to do if something doesn't work as expected.

Software:

Distributions

View the different release Distributions for ROS.

Packages

Search the 2000+ software libraries available for ROS.

Core Libraries

APIs by language and topic.

Common Tools

Common tools for developing and debugging ROS software.

Robots/Hardware:

Robots

Robots that you can use with ROS.

Sensors

Sensor drivers for ROS.

Motors

Motor controller drivers for ROS.

Publications, Courses, and Events:

Papers

Published papers with open source implementations available.

Books

Published books with documentation and tutorials with open source code available.

Courses

Courses using or teaching ROS.

Events

Past events and materials based on ROS.

Some Robots using ROS (> 125)

http://wiki.ros.org/Robots

Fraunhofer IPA Care-O-bot

Videre Erratic

TurtleBot

Aldebaran Nao

Lego NXT

Shadow Hand

Willow Garage PR2

iRobot Roomba

Robotnik Guardian

Merlin miabotPro

AscTec Quadrotor

CoroWare Corobot

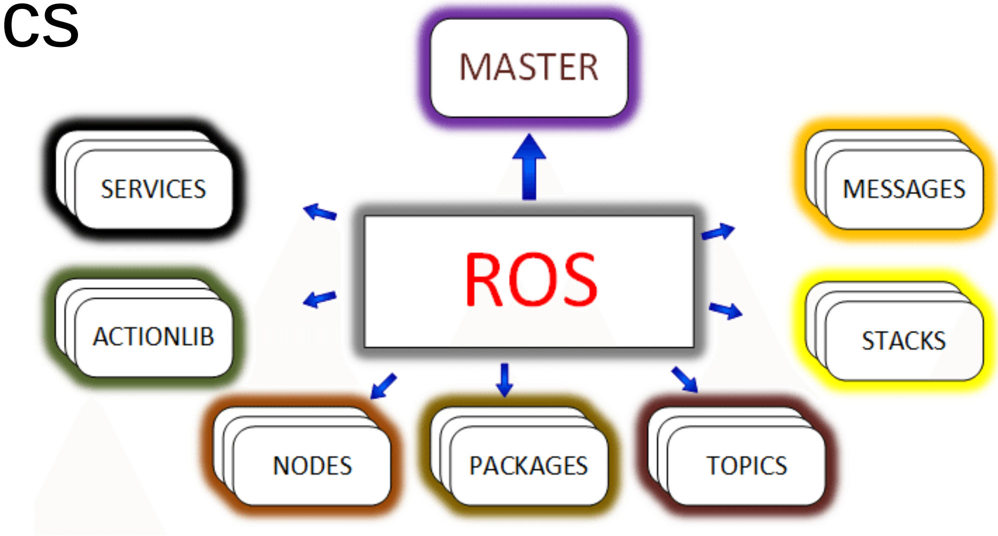
Clearpath Robotics Husky

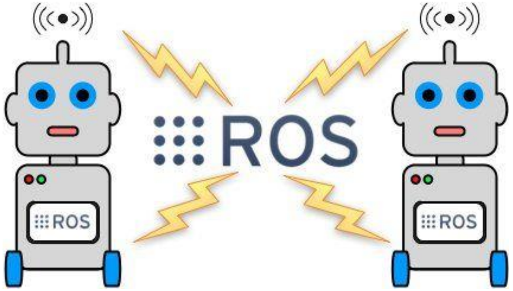
Clearpath Robotics Kingfisher

Festo Didactic Robotino

ROS Core Concepts

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

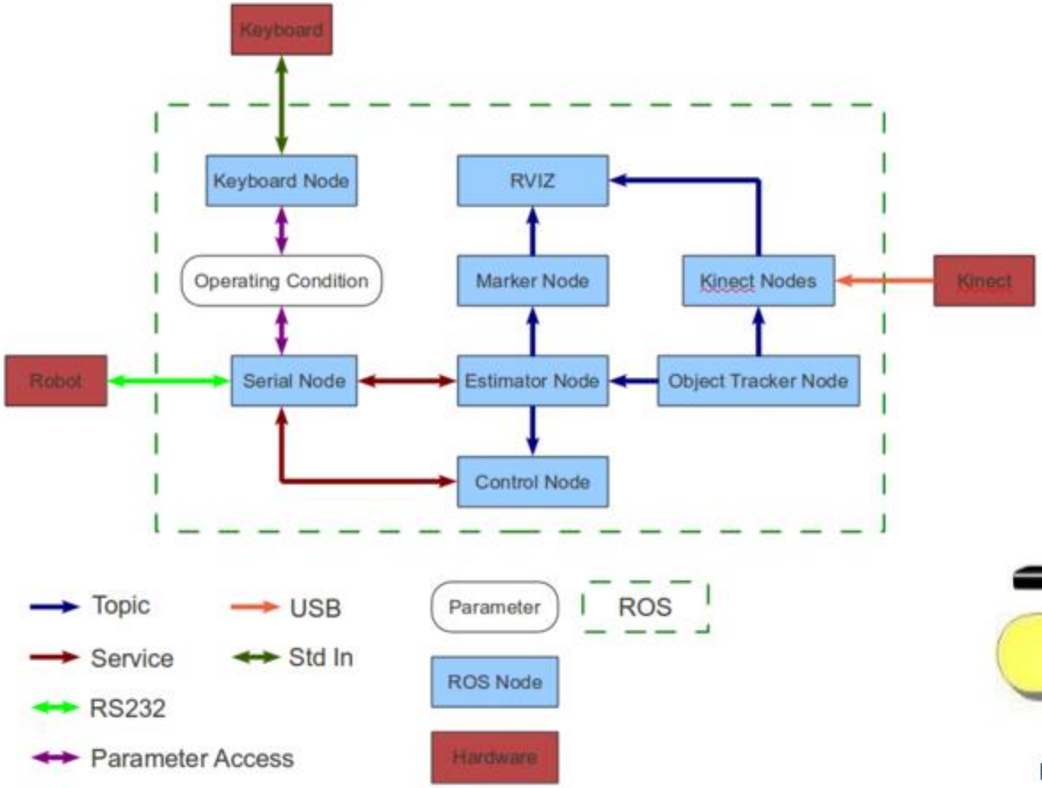


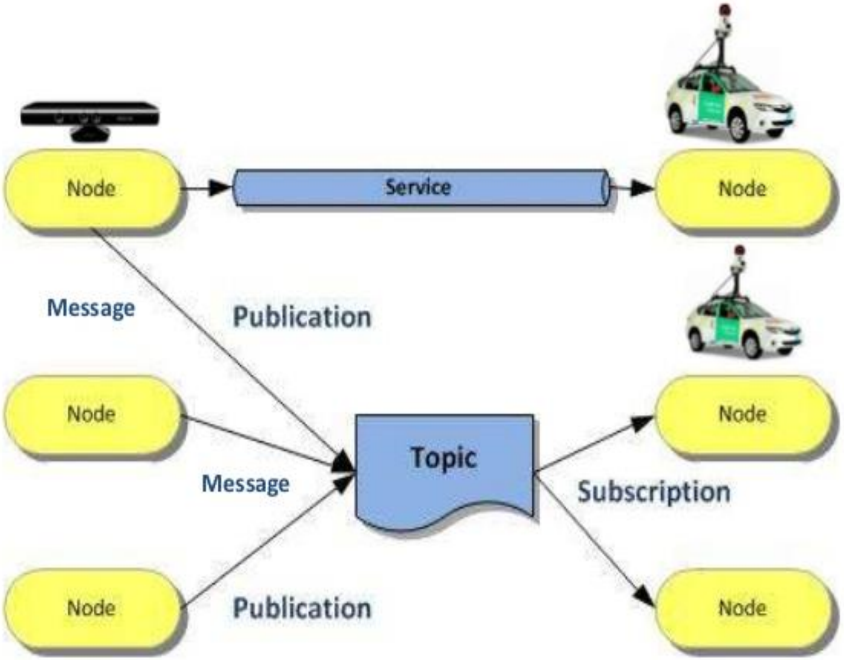


ROS Nodes

- 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**
  - roscpp – C++ client library
  - rospy – python client library
- Nodes can publish or subscribe to a **Topic**
- Nodes can also provide or use a **Service** or an **Action**

ROS Nodes


Keyboard
Keyboard Node
RVIZ
Operating Condition
Marker Node
Kinect Nodes
Kinect
Robot
Serial Node
Estimator Node
Object Tracker Node
Control Node
Topic
USB
Service
Std In
RS232
Parameter Access
Parameter
ROS
ROS Node
Hardware


Node
Service
Node
Message
Publication
Node
Topic
Message
Subscription
Node
Node
Publication
Node

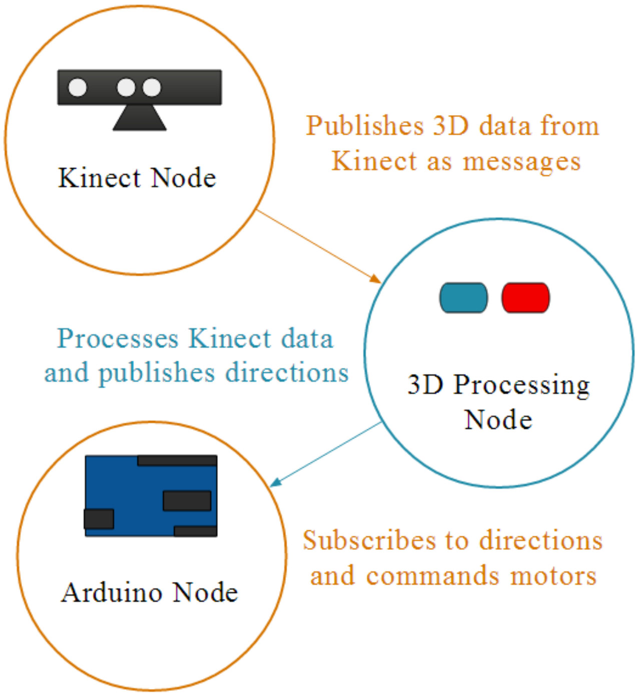
ROS Topics and ROS Messages

- 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 Topics and ROS Messages

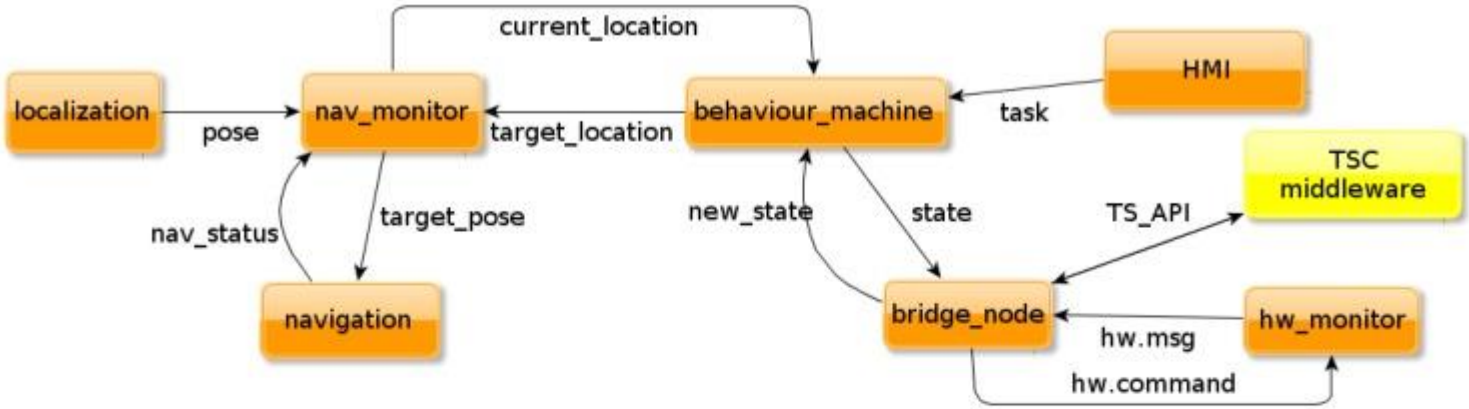

Kinect Node
Publishes 3D data from Kinect as messages
Processes Kinect data and publishes directions
3D Processing Node
Arduino Node
Subscribes to directions and commands motors

geometry_msgs/Twist

```
Vector3 linear
Vector3 angular
```

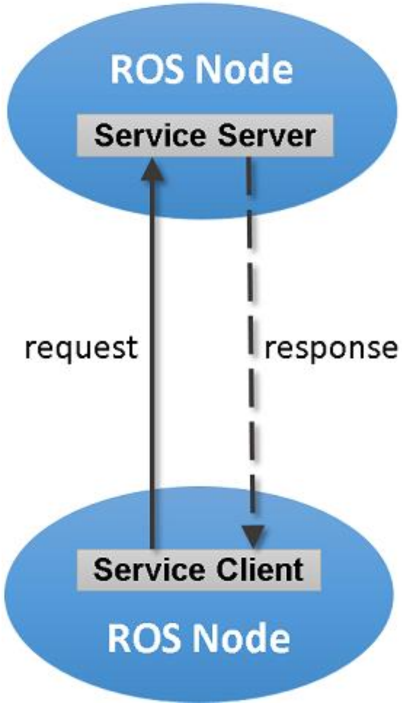
Vector3

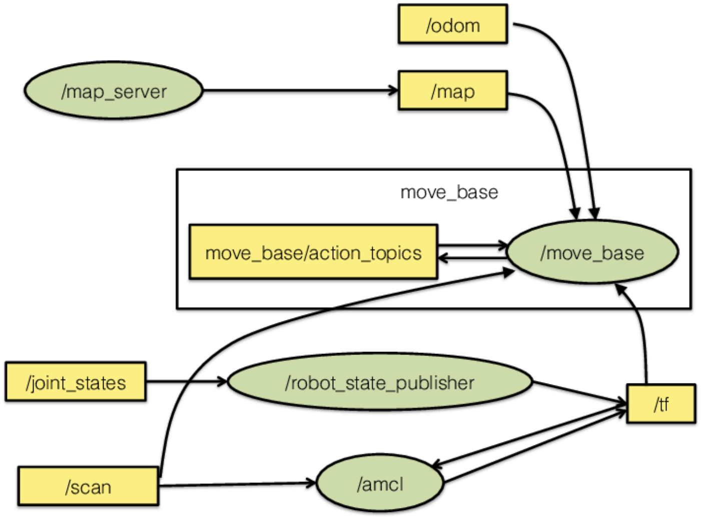
```
float64 x
float64 y
float64 z
```



ROS Services

- Synchronous inter-node transactions (blocking RPC): ask for something and wait for it
- **Service/Client model**: 1-to-1 request-response
- Service roles:
  - carry out remote computation
  - trigger functionality / behavior
  - *map_server/static_map* – retrieves the current grid map used for navigation

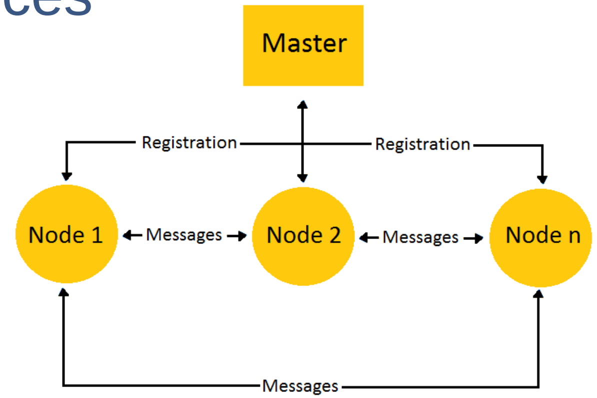




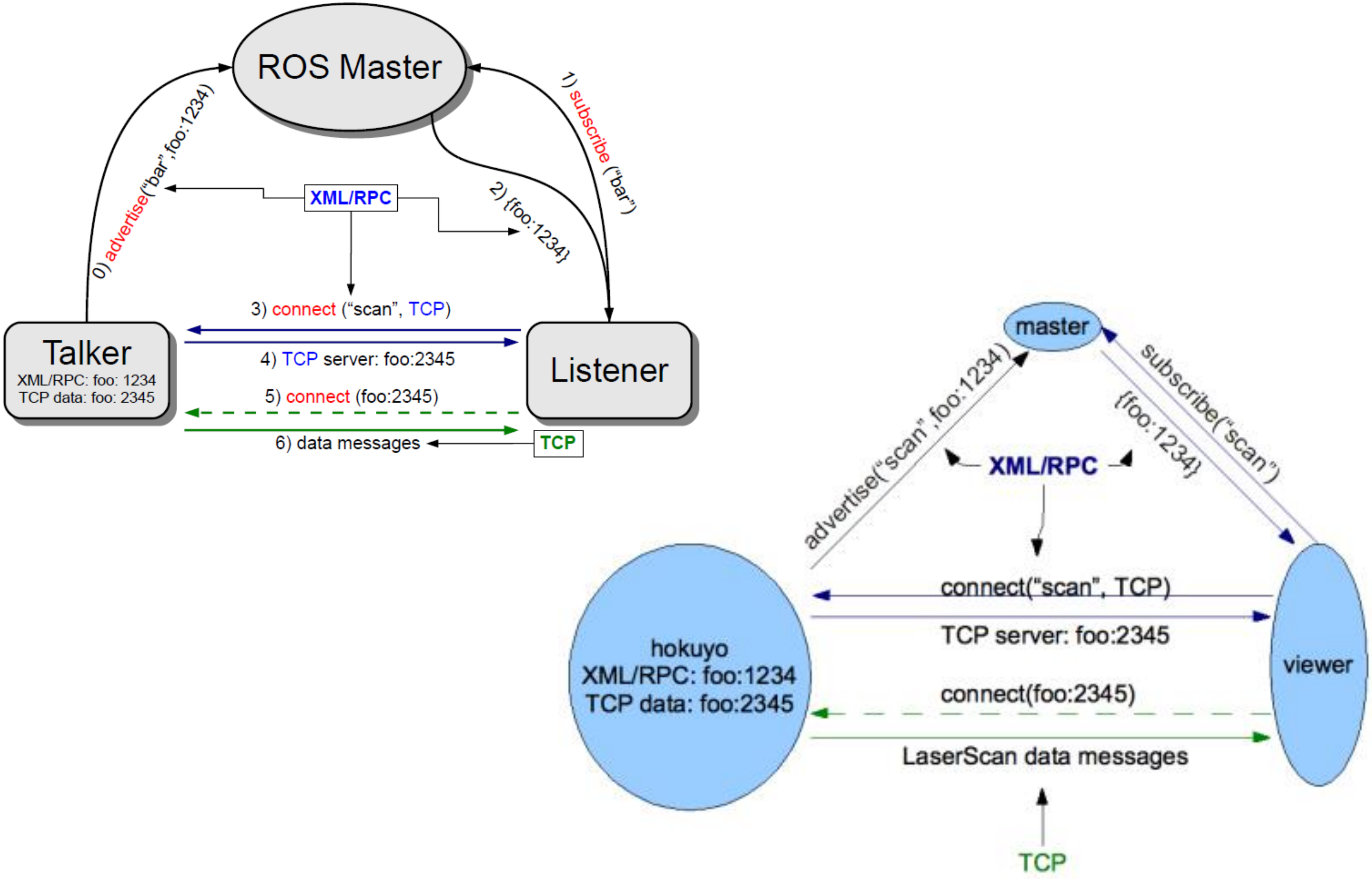
Service Name:	**/example_service**
Service Type:	**roscpp_tutorials/TwoInts**
Request Type:	**roscpp_tutorials/TwoIntsRequest**
Response Type:	**roscpp_tutorials/TwoIntsResponse**

ROS Master

- Provides connection information to nodes so that they can transmit messages to each other
  - When activated, every node connects to a specified master to **register** details of the message streams they publish, services and actions that they provide, and streams, services, an actions to which that they to subscribe
  - When a new node appears, the master provides it with the information that it needs to form a direct peer-to-peer TCP-based connection with other nodes publishing and subscribing to the same message topics and services

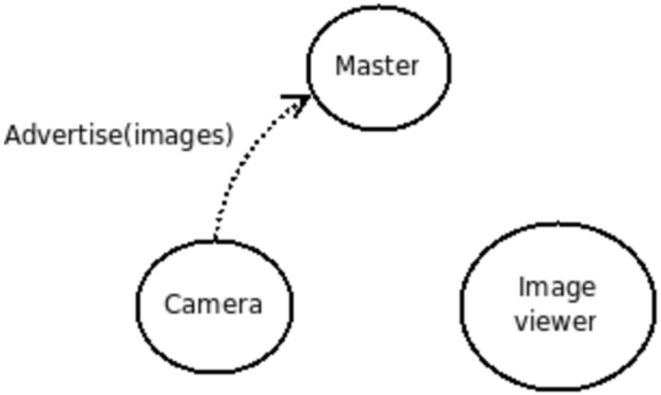


ROS MASTER


ROS Master
0) advertise("bar",foo:1234)
1) subscribe ("bar")
2) {foo:1234}
XML/RPC
3) connect ("scan", TCP)
4) TCP server: foo:2345
5) connect (foo:2345)
6) data messages
TCP
Talker
XML/RPC: foo: 1234
TCP data: foo: 2345
Listener
master
advertise("scan",foo:1234)
subscribe("scan")
{foo:1234}
XML/RPC
connect("scan", TCP)
TCP server: foo:2345
connect(foo:2345)
LaserScan data messages
TCP
hokuyo
XML/RPC: foo:1234
TCP data: foo:2345
viewer

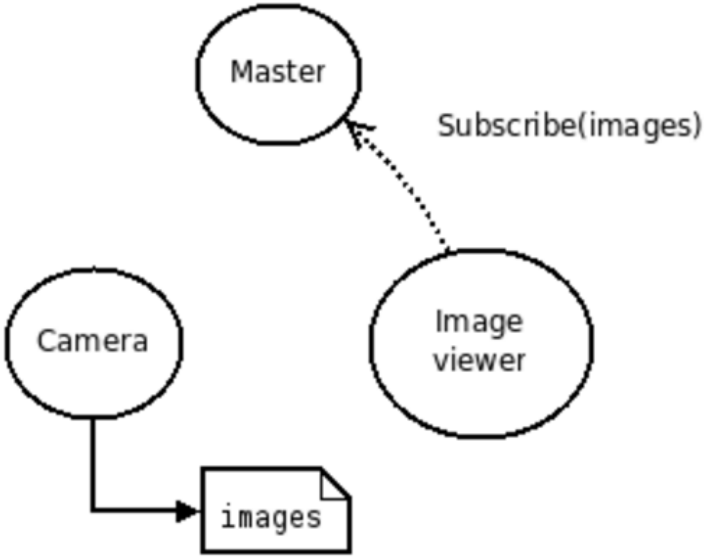
ROS MASTER

- We have two nodes: a *Camera* node and an *Image_viewer* node
- Typically the camera node would start first notifying the master that it wants to publish images on the topic "*images*":



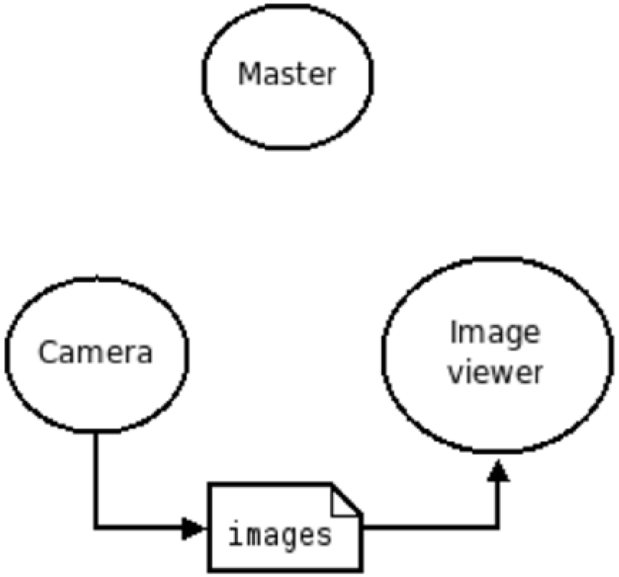
ROS MASTER

- *Image_viewer* wants to subscribe to the topic "*images*" to get and display images obtained with the camera:



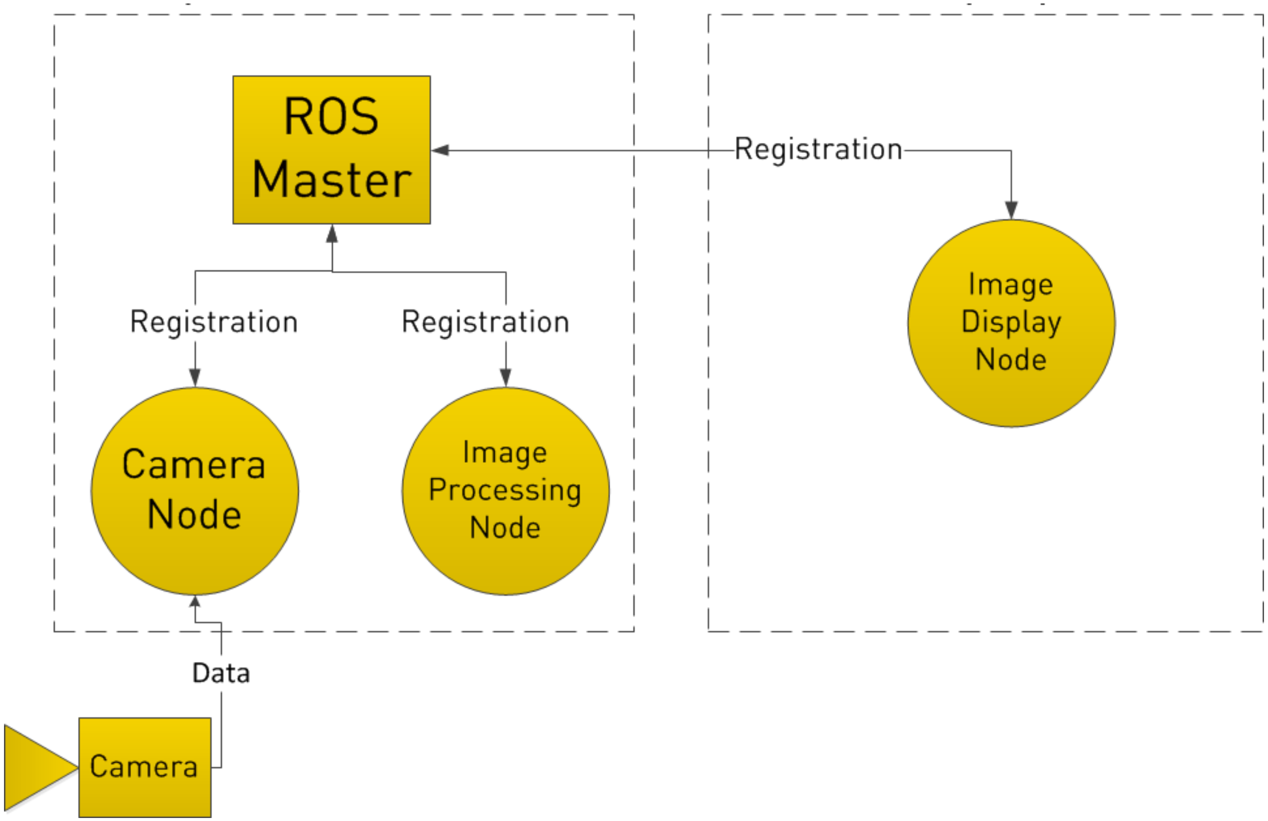
ROS MASTER

- Now that the topic "*images*" has both a publisher and a subscriber, the master node notifies *Camera* and *Image_viewer* about each others existence, so that they can start transferring images to one another:



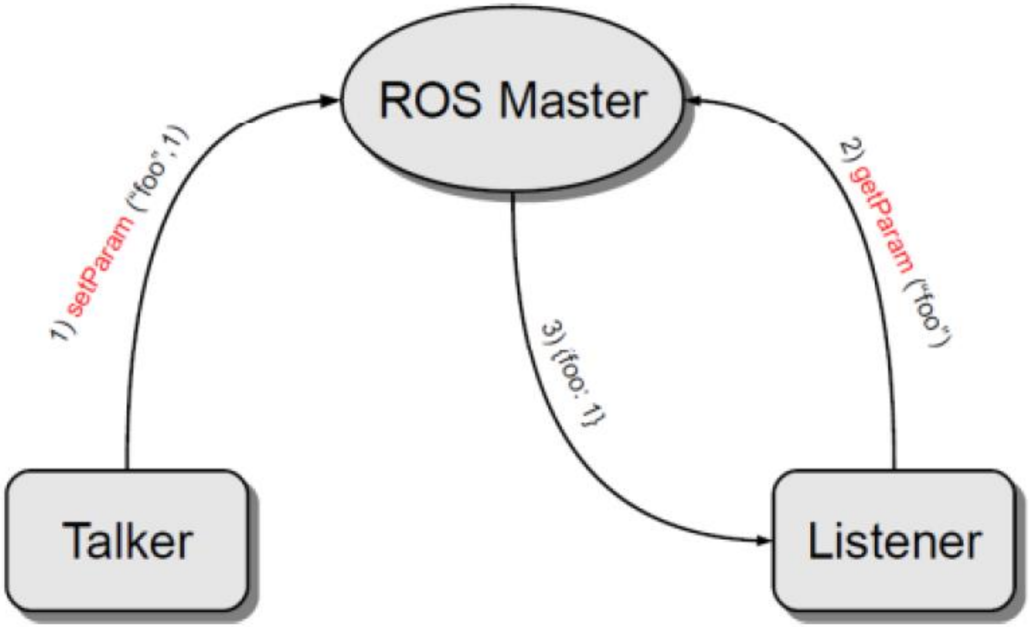
ROS MASTER

- The scenario can be made even more modular by adding an *Image processing* node, from which the *Image viewer* gets its data



Parameter Server

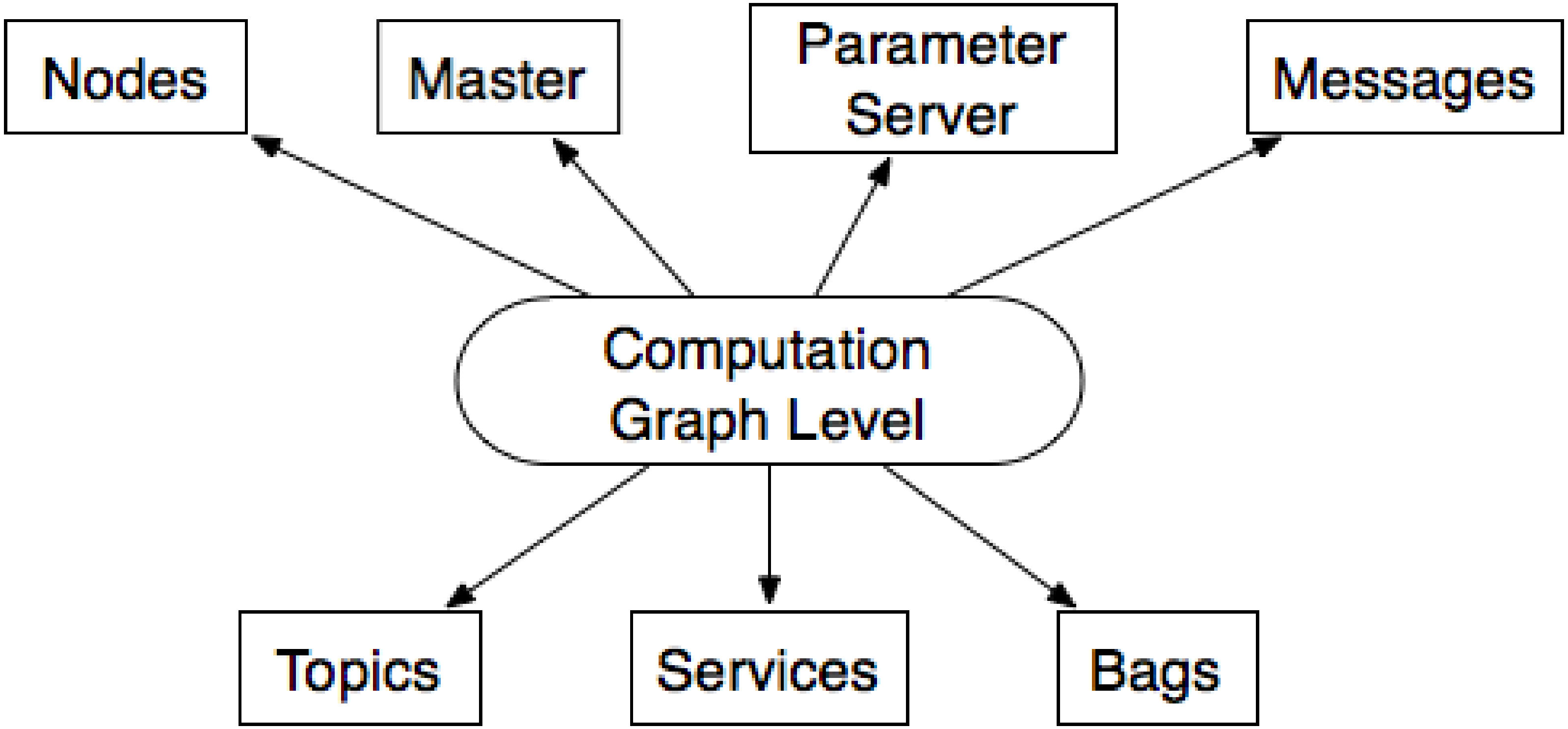
- A *shared*, multi-variate dictionary that is accessible via network APIs
- Best used for static, non-binary data such as *configuration parameters*
- Runs inside the ROS master



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 Computation 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
- Current ROS Kinetic Kame runs on **Ubuntu 16.04** (Xenial) and will support Ubuntu 15.10 (Willy)

ROS Environment

- ROS is fully integrated in the Linux environment: the ***rosbash*** package contains useful bash functions and adds tab-completion to a large number of ROS utilities

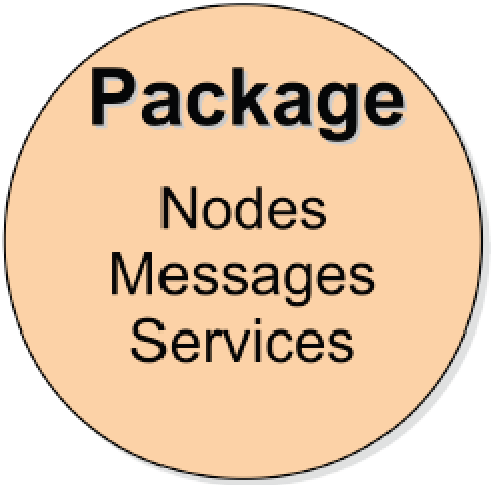
- After installing, ROS, setup.*sh files in '/opt/ros/<distro>/', need to be sourced to start *rosbash*:

```
$ source /opt/ros/indigo/setup.bash
```

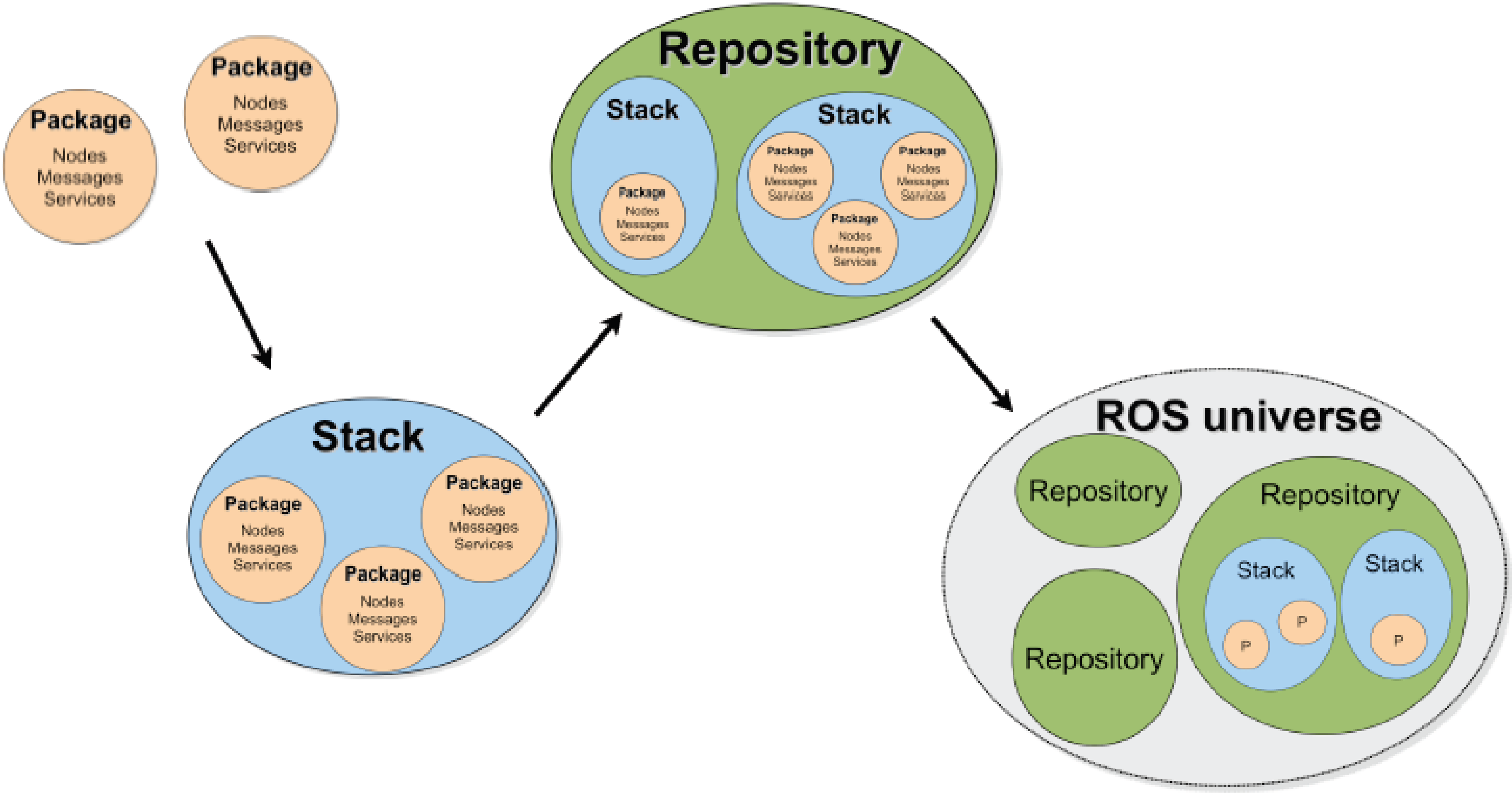
- This command needs to be run on every new shell to have access to the ros commands: an easy way to do it is to add the line to the bash startup file (~/.bashrc)

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



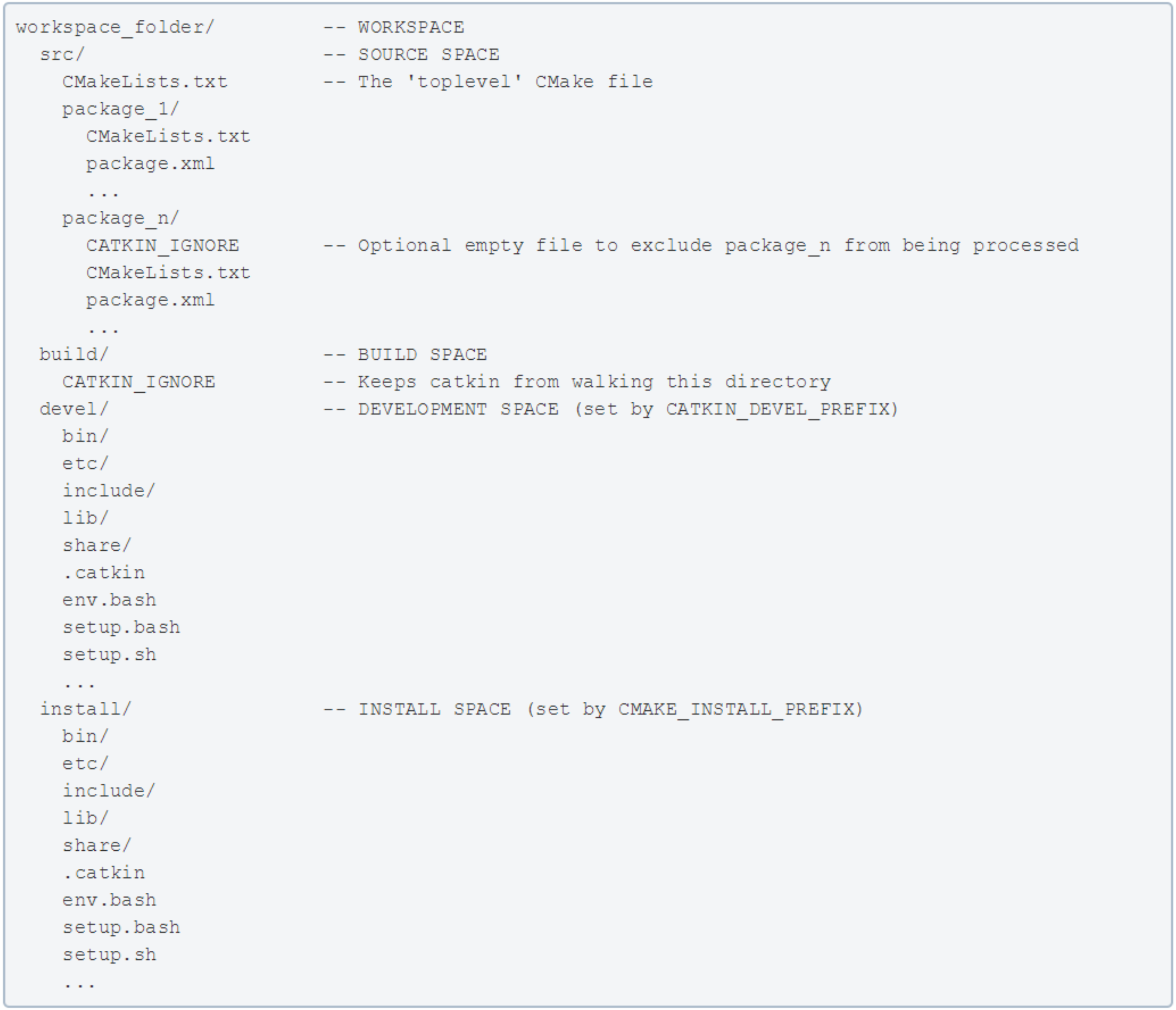
ROS Package System



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

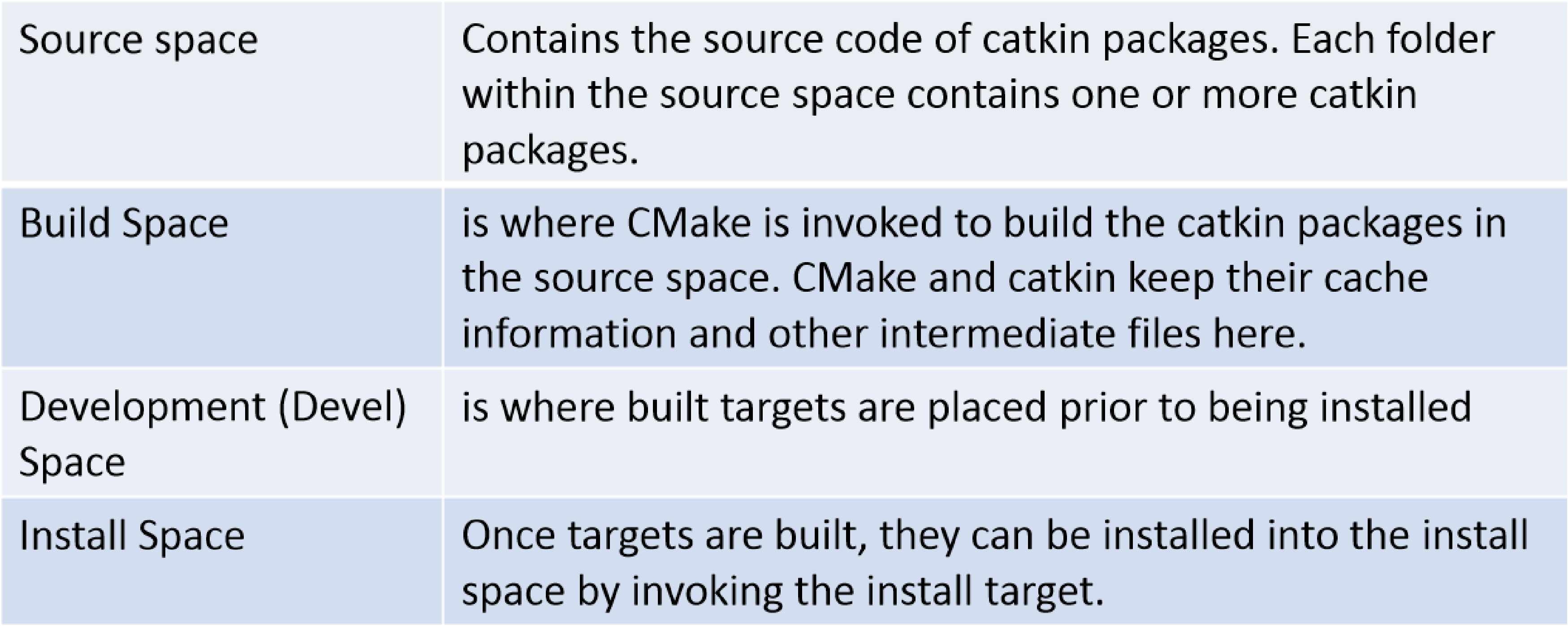
CATKIN WORKSPACE LAYOUT



```
workspace_folder/         -- WORKSPACE
  src/                    -- SOURCE SPACE
    CMakeLists.txt        -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      CATKIN_IGNORE       -- Optional empty file to exclude package_n from being processed
      CMakeLists.txt
      package.xml
      ...
  build/                  -- BUILD SPACE
    CATKIN_IGNORE         -- Keeps catkin from walking this directory
  devel/                  -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
  install/                -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
```

CATKIN WORKSPACE FOLDERS

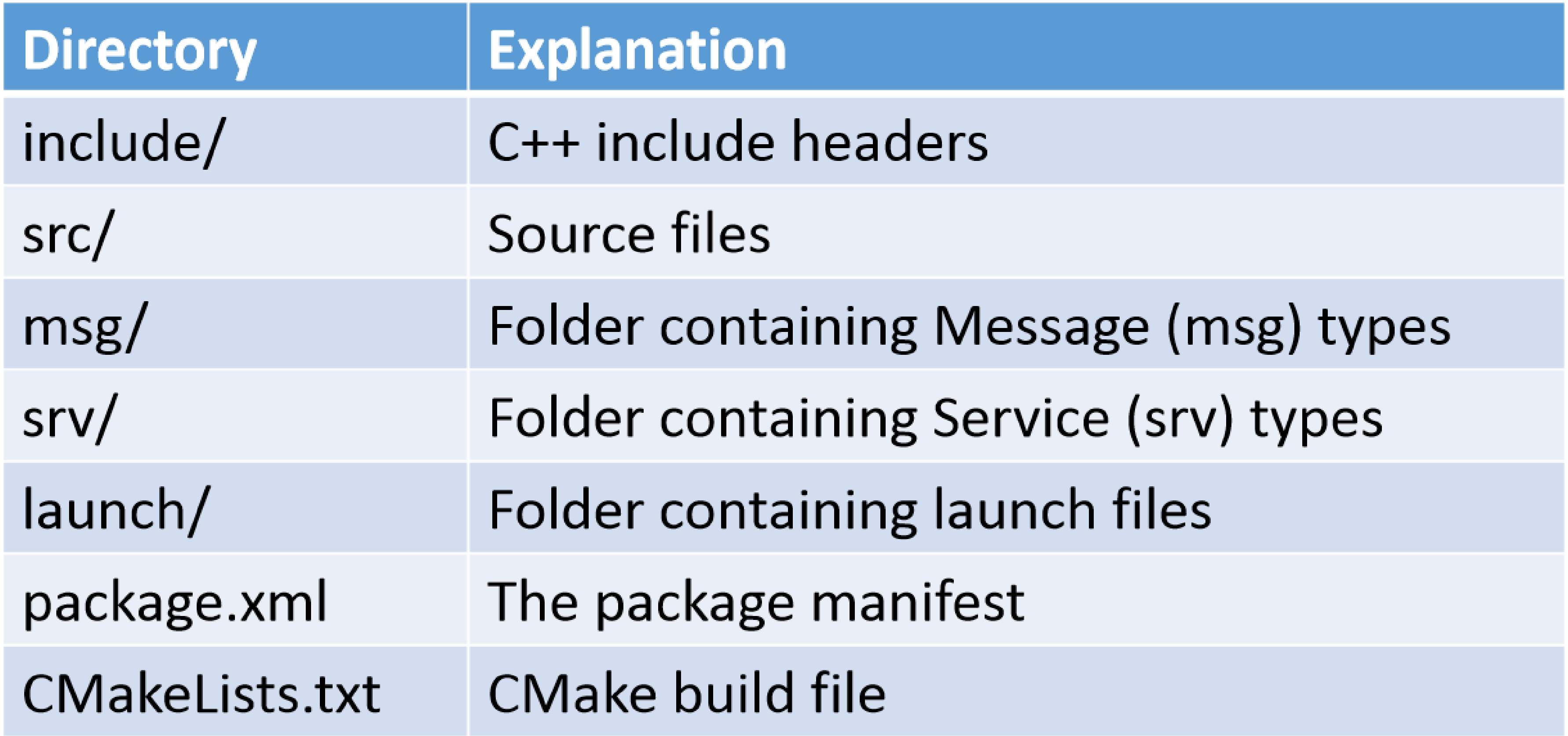
- Source space: *workspace_folder/src*
- Build space: *workspace_folder/build*
- Development space: *workspace_folder/devel*
- Install space: *workspace_folder/install*



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

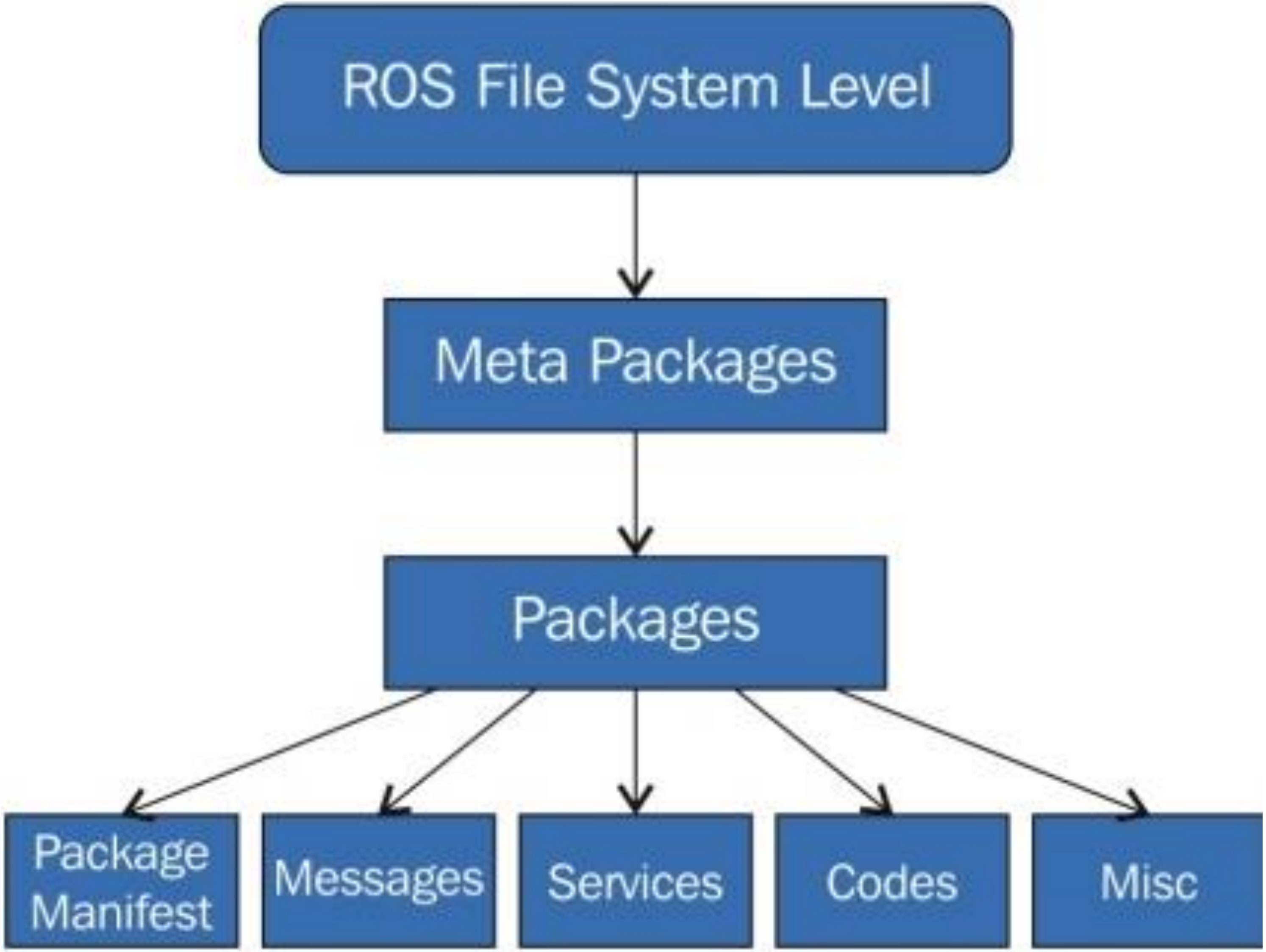
- Layout of the *src/my_package* folder in a catkin workspace:



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 File System Level

Meta Packages

Packages

Package Manifest

Messages

Services

Codes

Misc

ROS Community Level

Distributions

Blog

ROS Wiki

Mailing list

Repositories

Bug ticket System

ROS Answers