Topic 2: Robot Operating System

2.1 Introduction

2.2 ROS Philosophy

2.3 ROS Core concepts

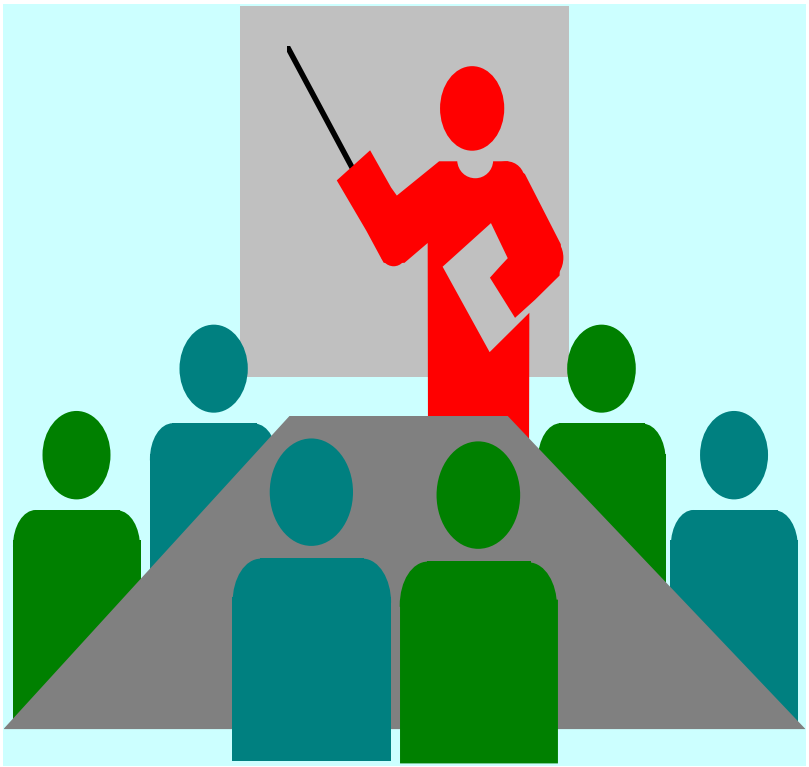
2.4 ROS Basic Commands

2.5 catkin Build System

2.6 ROS Publisher & Subscriber

2.7 Gazabo simulator

2.8 Conclusion



2.1 Introduction

What is ROS?

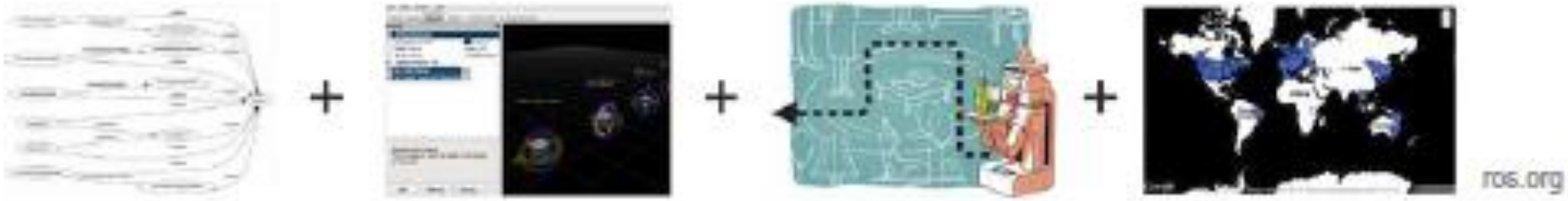
- ROS is an open-source **robot operating system**
- It has a set of software libraries and tools that help you build robot applications.
- It works across a wide variety of robotic platforms
- Originally developed at the Stanford Artificial Intelligence Laboratory in 2007.
- Its development is now continued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

2.1 Introduction – Different versions

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Kinetic Kame (**Recommended**)	May 23rd, 2016			May, 2021
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

2.1 Introduction – Different Components

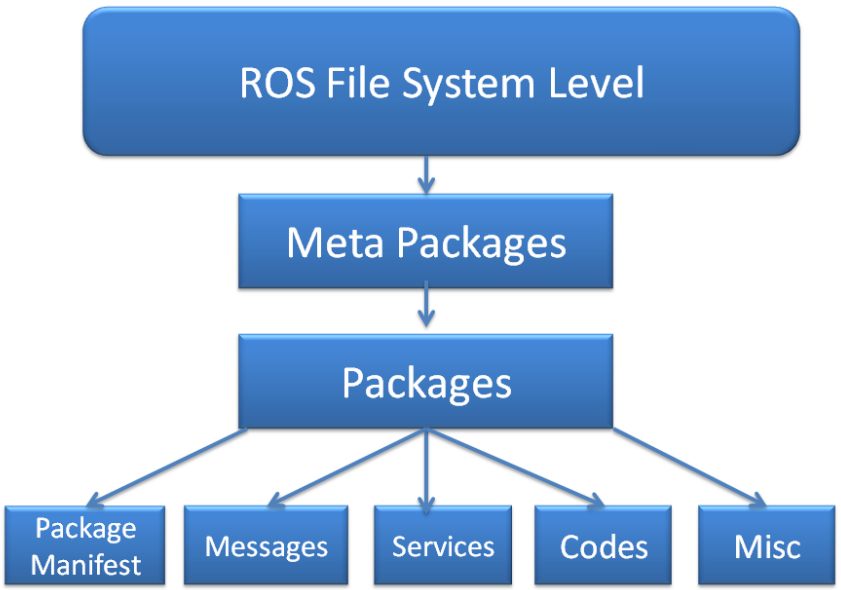
ROS = Robot Operating System



Plumbing	Tools	Capabilities	Ecosystem
▪ Process management ▪ Inter-process communication ▪ Device drivers	▪ Simulation ▪ Visualization ▪ Graphical user interface ▪ Data logging	▪ Control ▪ Planning ▪ Perception ▪ Mapping ▪ Manipulation	▪ Package organization ▪ Software distribution ▪ Documentation ▪ Tutorials

2.1 Introduction – ROS File Systems

Like a real operating system, ROS files are organized on the hard disk in a particular manner, as depicted in the following figure:



- Packages are the most basic unit of the ROS software. They contain one or more ROS nodes, libraries, configuration files, etc.
- Package manifest file contains author, license, dependencies, compilation flags, and so on, namely package.xml.
- Meta packages refers to one or more related packages which can be loosely grouped together.
- Messages (.msg) are sent from one ROS process to others(my_package/msg/MyMessageType.msg).
- Services (.srv) is a kind of request/reply interaction between processes, whose data types can be defined inside the srv folder inside the package (my_package/srv/MyServiceType.srv).

2.2 ROS Philosophy

- Peer to Peer
  ROS Individual programs communicate over defined API (ROS messages, services, etc.).
- *Distributed*
  ROS programs can be run on multiple computers and communicate over the network.
- Multi-lingual
  ROS software modules can be written in any language (C++, Python, LISP, Java, JavaScript, MATLAB, etc.
- Light-weight
  ROS contributors can create stand-alone libraries and then *wrap those libraries to/from other ROS modules.*
- Free and open source
  Most ROS software is open-source and free to use.

2.2 ROS Philosophy

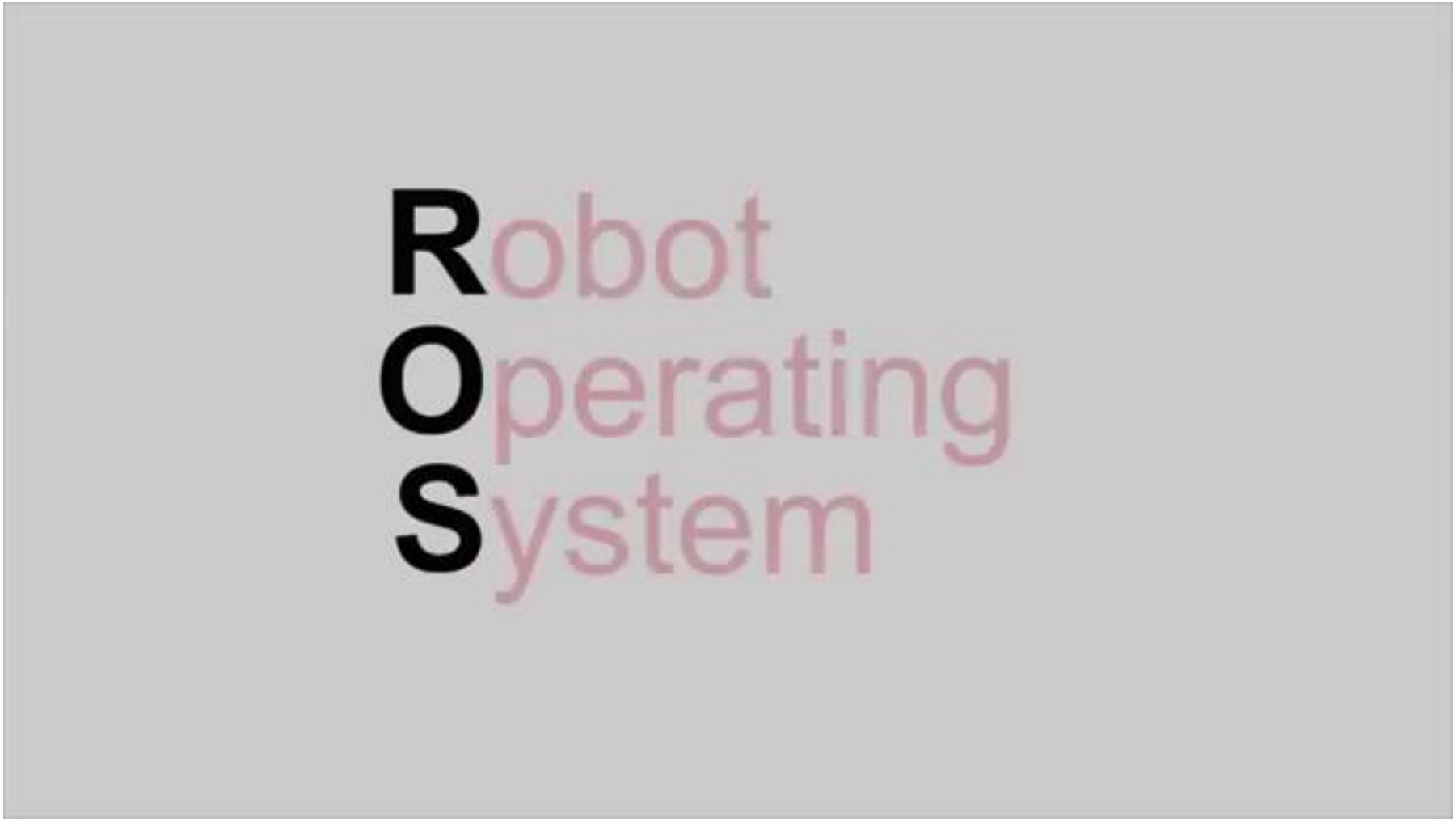
- **The operating system side**, which provides standard operating system services such as:
  - Hardware abstraction
  - Low-level device control
  - Message-passing between nodes
  - Package management
  - Debugging and Visualization tools
- **User contributed packages**, which implement common robot functionality such as SLAM, planning, perception, vision, manipulation, etc.

2.2 ROS Philosophy

- http://wiki.ros.org/
- Installation: http://wiki.ros.org/ROS/Installation
- Tutorials: http://wiki.ros.org/ROS/Tutorials
- ROS Tutorial Videos

  http://www.youtube.com/playlist?list=PLDC89965A56E6A8D6
- ROS Cheat Sheet
  http://www.tedusar.eu/files/summerschool2013/ROScheatsheet.pdf
- Supported robots: http://www.ros.org/wiki/Robots

2.2 ROS Philosophy



2.3 ROS Core Concepts

2.3.1 ROS Master

2.3.2 ROS Nodes

2.3.3 ROS Topics

2.3.4 ROS Messages

2.3.5 ROS Services

2.3.6 ROS Parameters

2.3.7 ROS Packages

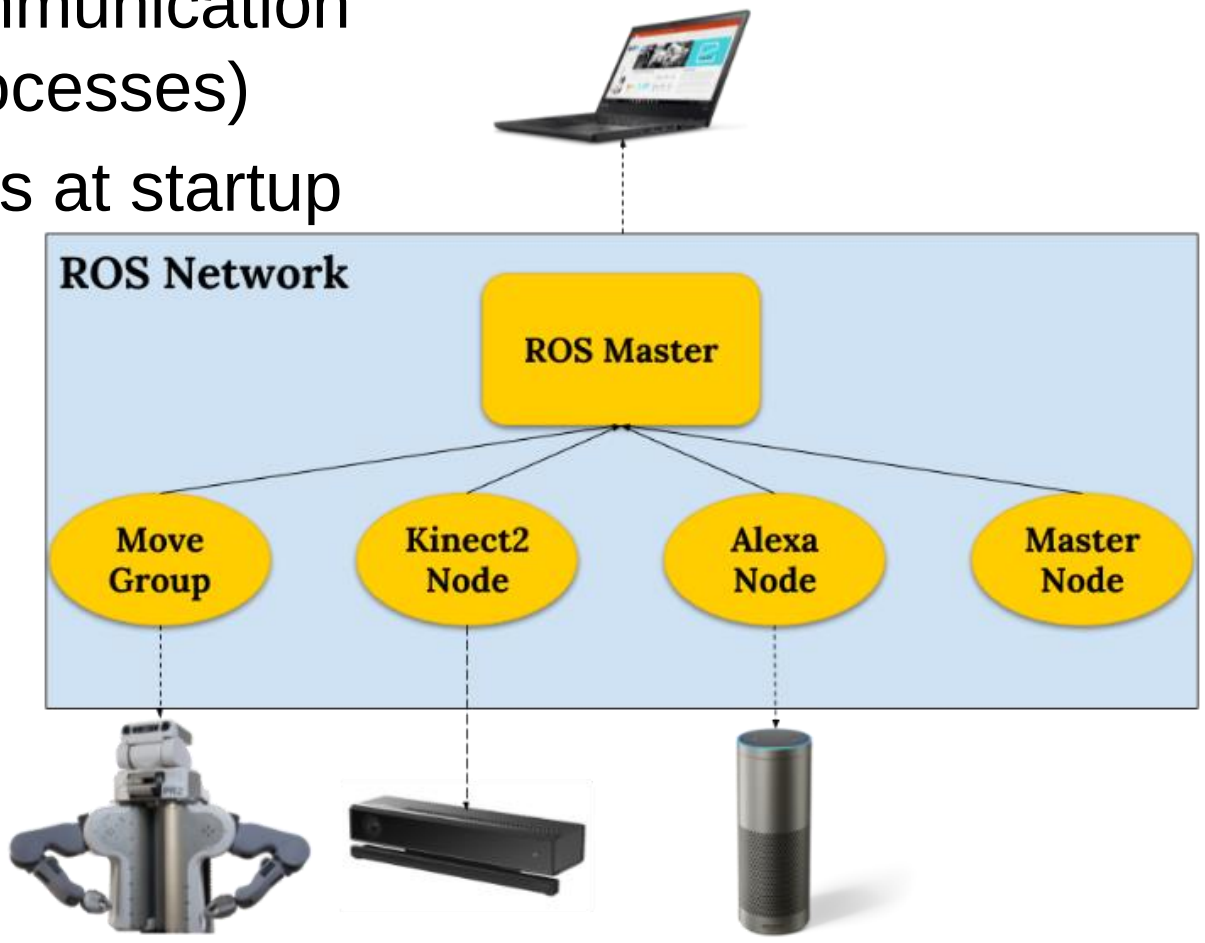
2.3.8 ROS Graph

2.3.1 ROS Master

- To manage the communication between nodes (processes)
- Every node registers at startup with the master
- Start a master with

  > roscore

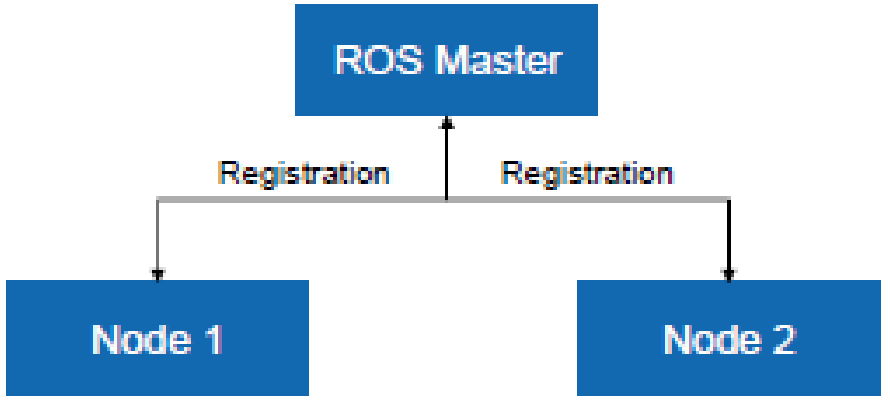
ROS Master



2.3.2 ROS Nodes

- A node is single-purpose & executable program
  - control robot wheel motors
  - acquire data from laser scanner
  - acquire images from camera
  - perform localisation
  - perform path planning
- Nodes are individually compiled, executed and managed
- Nodes are written using a ROS **client library**

  roscpp – C++ client library



Run a node with

```
> rosrun package_name node_name
```

See active nodes with

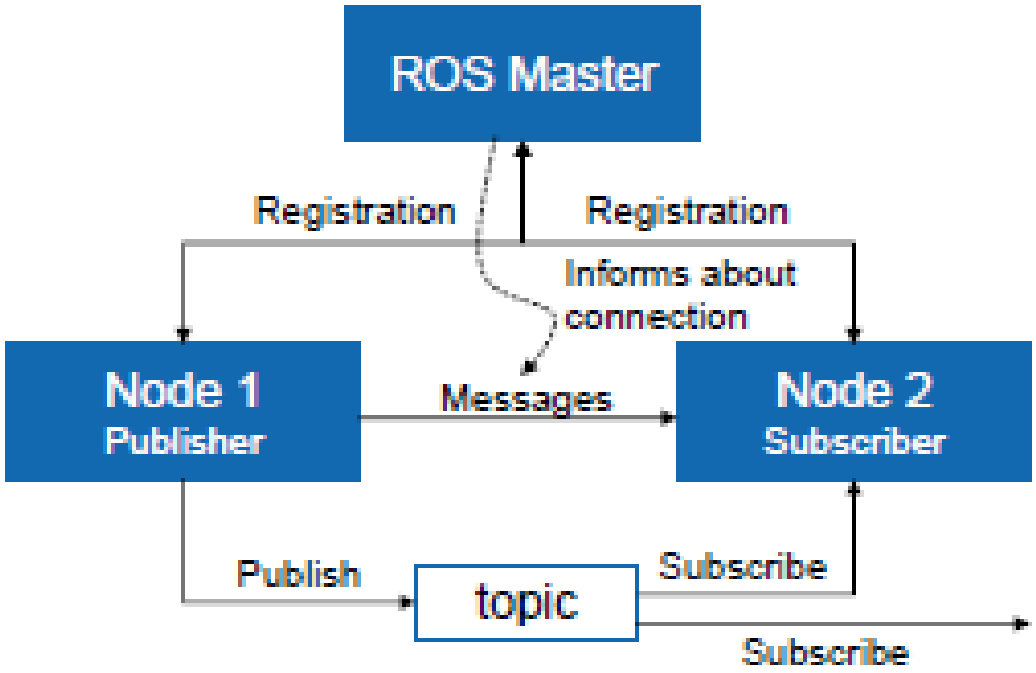
```
> rosnode list
```

Retrieve information about a node with

```
> rosnode info node_name
```

2.3.3 ROS Topics

- Topic is a name for a stream of messages.
- Nodes communicate over topics



List active topics with

```
> rostopic list
```

Subscribe and print the contents of a topic with

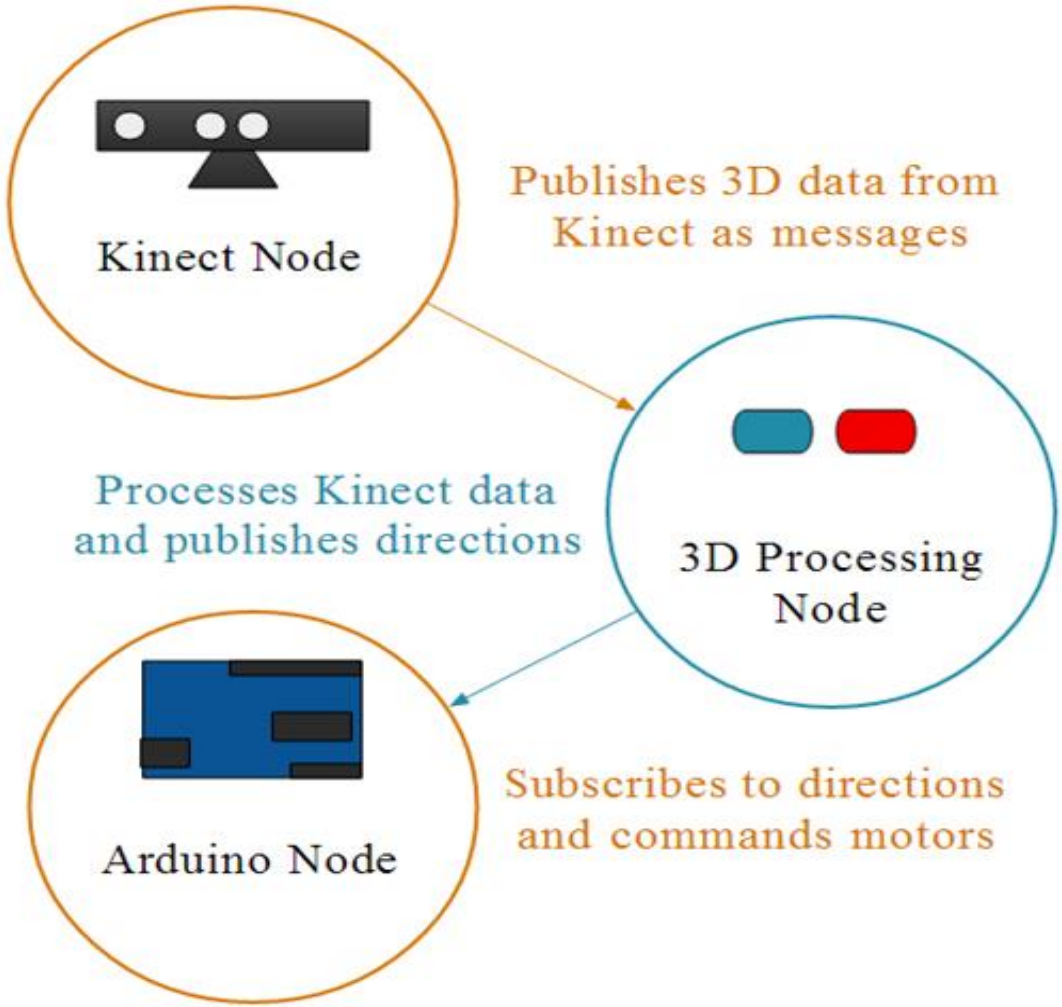
```
> rostopic echo /topic
```

Show information about a topic with

```
> rostopic info /topic
```

2.3.3 ROS Topics

- Nodes can publish or subscribe to a topic
  - e.g., data from a laser range-finder might be sent on a topic called scan, with a message type of LaserScan



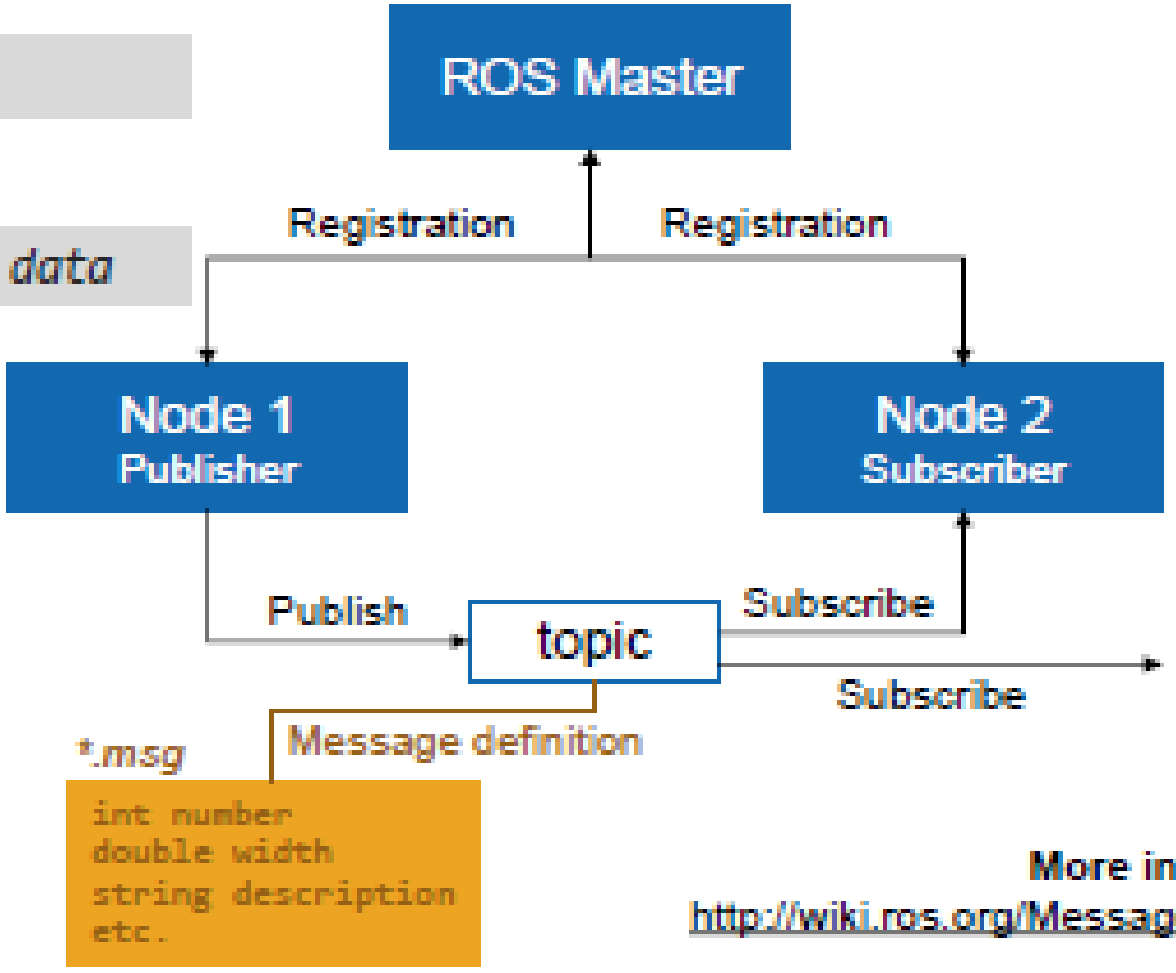
2.3.4 ROS Messages

See the type of a topic

```
> rostopic type /topic
```

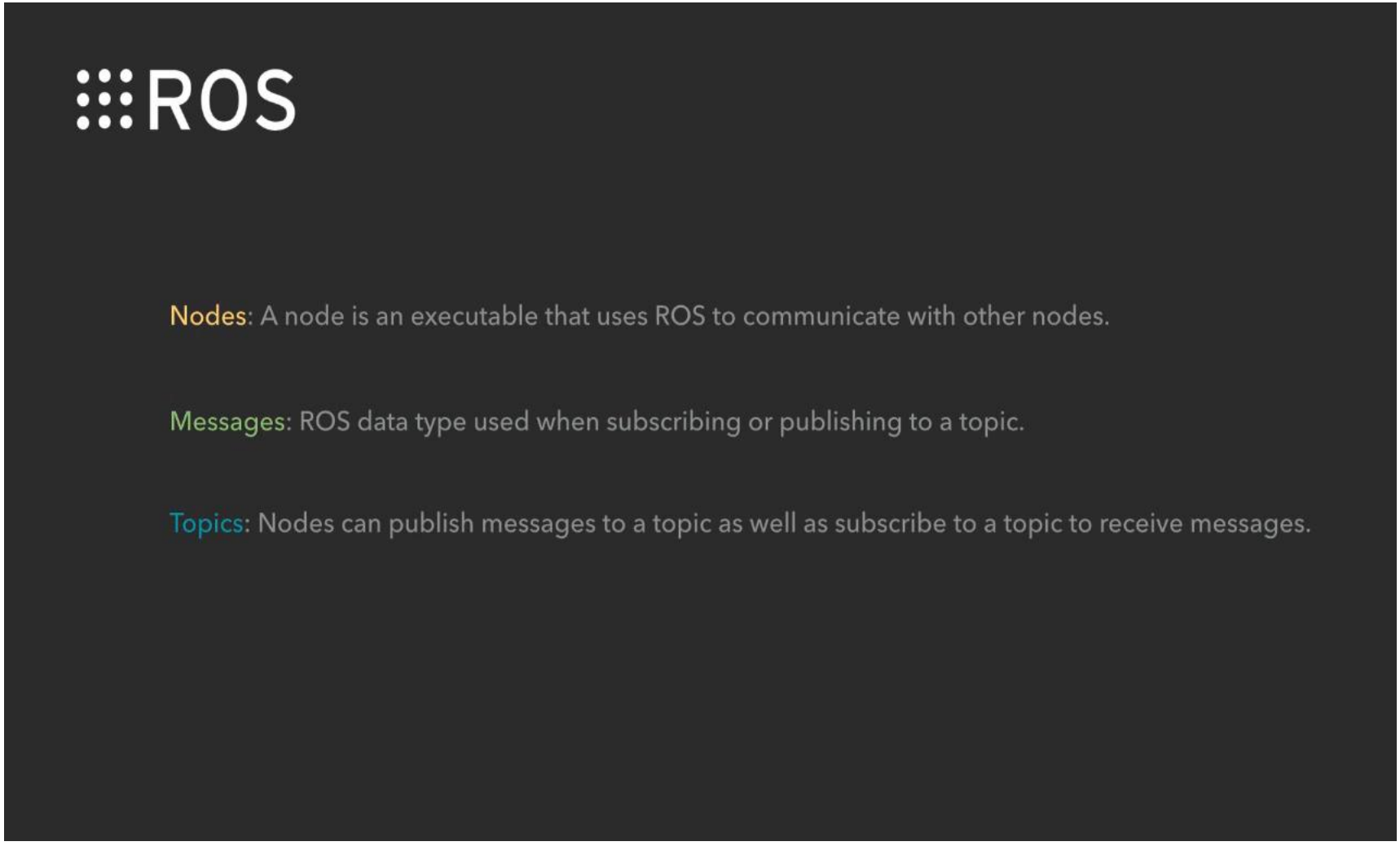
Publish a message to a topic

```
> rostopic pub /topic type data
```

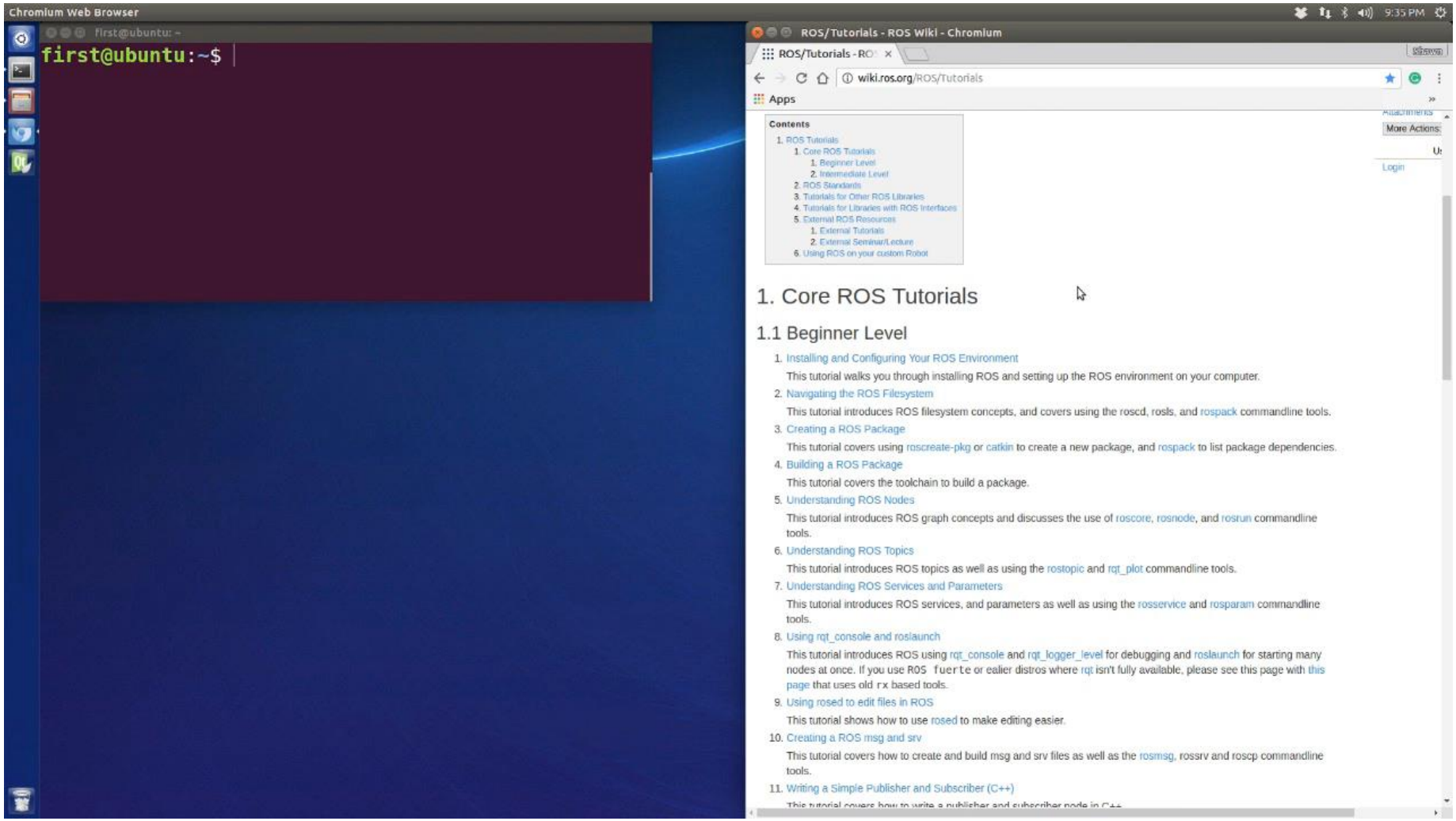


ROS Messages are defined in *.msg files

ROS Nodes – Messages - Topics



ROS Nodes – Messages - Topics

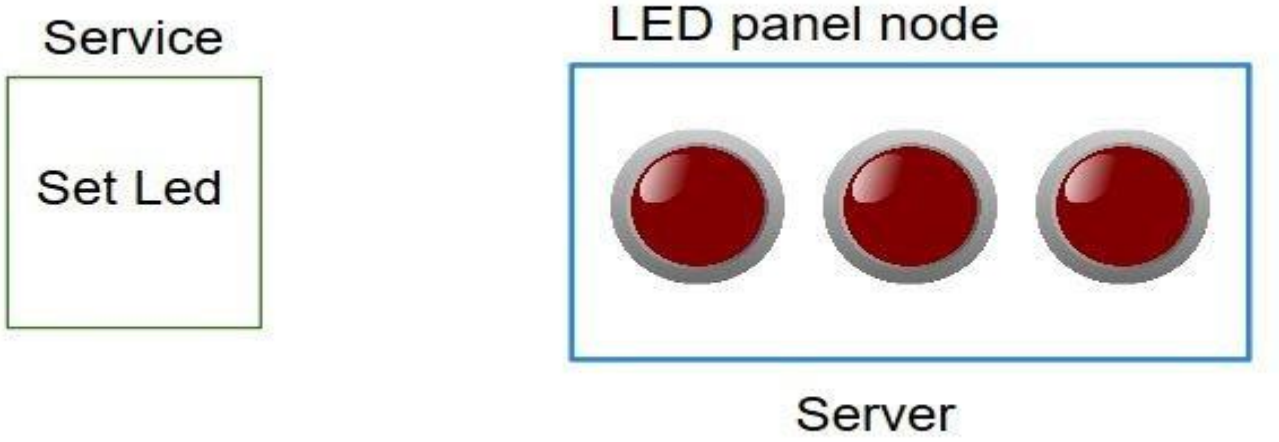


2.3.5 ROS Services

- A ROS service is a client/server system.
- It is synchronous. The client sends requests until it receives a response.
- ROS services can be used for computations and quick actions.
- A service is defined by a name and a pair of messages (request & response).
- You can directly create service clients and servers inside ROS nodes, using the rosccp library for C++ and the rospy library for Python.
- A service server can only exist once, but can have many clients.
- The service will be created when you create the server.

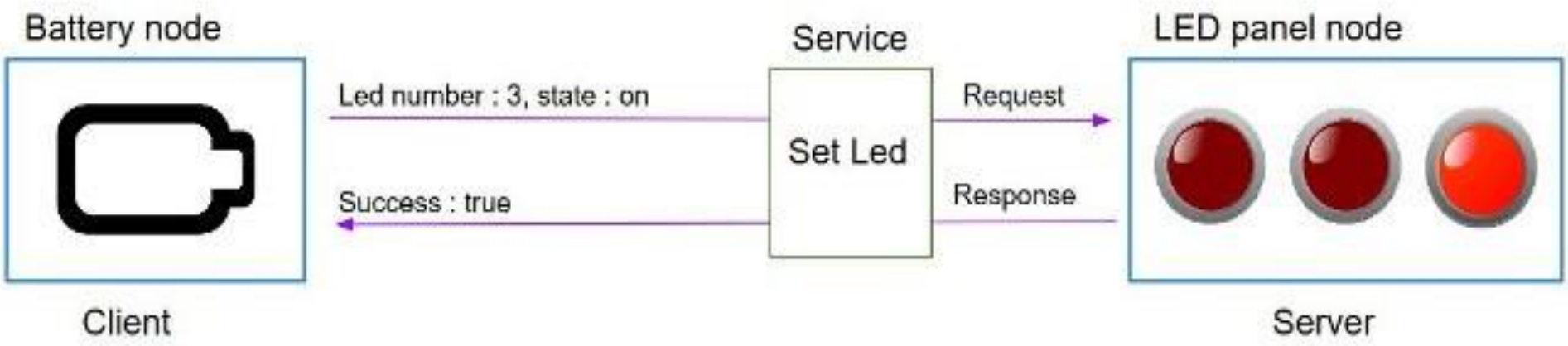
2.3.5 ROS Services

- For instance, the node is dealing with the hardware to power on and power off the LEDs.
- You create a ROS service, named “Set LED”. Inside the LED panel node, you create a service server for this ROS service.



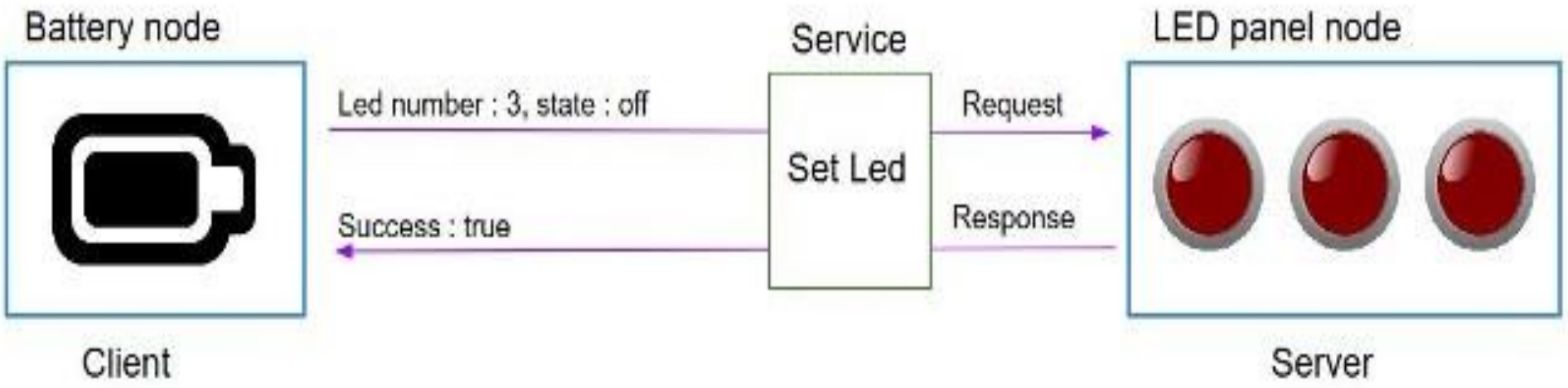
2.3.5 ROS Services

- The battery node has a service client for the “Set Led” Service.
- The battery node sends a request (a LED number and a state) to the ROS service.
- The server, which is the LED panel node, can power ON the third LED as requested.
- Once this is done, the server will send back a response.



2.3.5 ROS Services

- When the battery is detected to be low, the battery node will send a request (a LED number and a state.) to the ROS service to switch OFF a LED.
- The server receives this request, performs the operation, and sends back a success flag. The communication is done.



2.3.6 ROS Parameter Server

- Nodes use the parameter server to store and retrieve parameters at runtime
- Best used for static data such as configuration parameters
- Parameters can be defined in launch files or separate YAML files

config.yaml

```
camera:
  left:
    name: left_camera
    exposure: 1
  right:
    name: right_camera
    exposure: 1.1
```

List all parameters with

> Rosparam list

Get the value of a parameter with

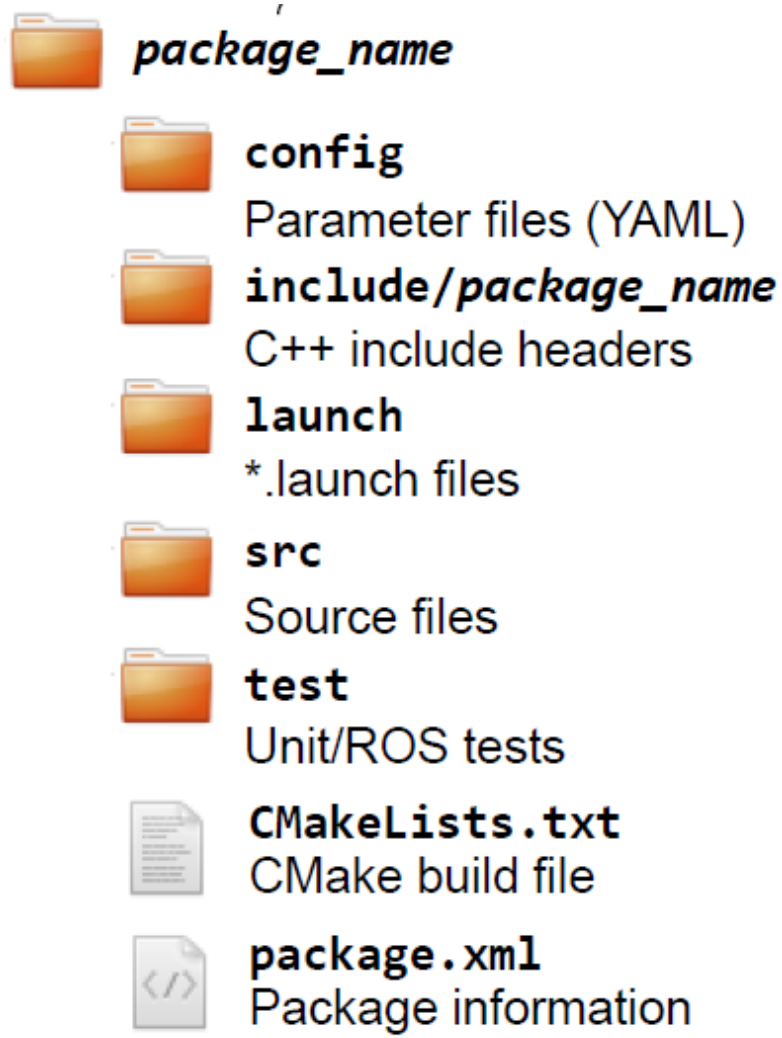
> rosparam_get *parameter_name*

Set the value of a parameter with

> rosparam_set *parameter_namevalue*

2.3.7 ROS Packages

- ROS software is organized into packages (atomic unit of build).
- Each package contains code, data and documentation.
- ROS package is a directory inside a catkin workspace that has a package.xml file in it.
- package.xml defines:
  - the package name
  - version numbers
  - authors
  - dependencies on other catkin packages
  - and more



2.3.7 Creating a ROS Package

The definition of package.xml of a typical package is shown in the following screenshot:

```
<?xml version="1.0"?>
<package>
  <name>hello_world</name>
  <version>0.0.1</version>
  <description>The hello_world package</description>
  <maintainer email="jonathan.cacace@gmail.com">Jonathan Cacace</maintainer>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>

  <run_depend>roscpp</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>

  <export>
  </export>
</package>
```

2.3.7 Creating a ROS Package

- Change to the source directory of the workspace

```
$ cd ~/catkin_ws/src
```

- **catkin_create_pkg** creates a new package with the specified dependencies

```
$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

- For example, create a first_pkg package:

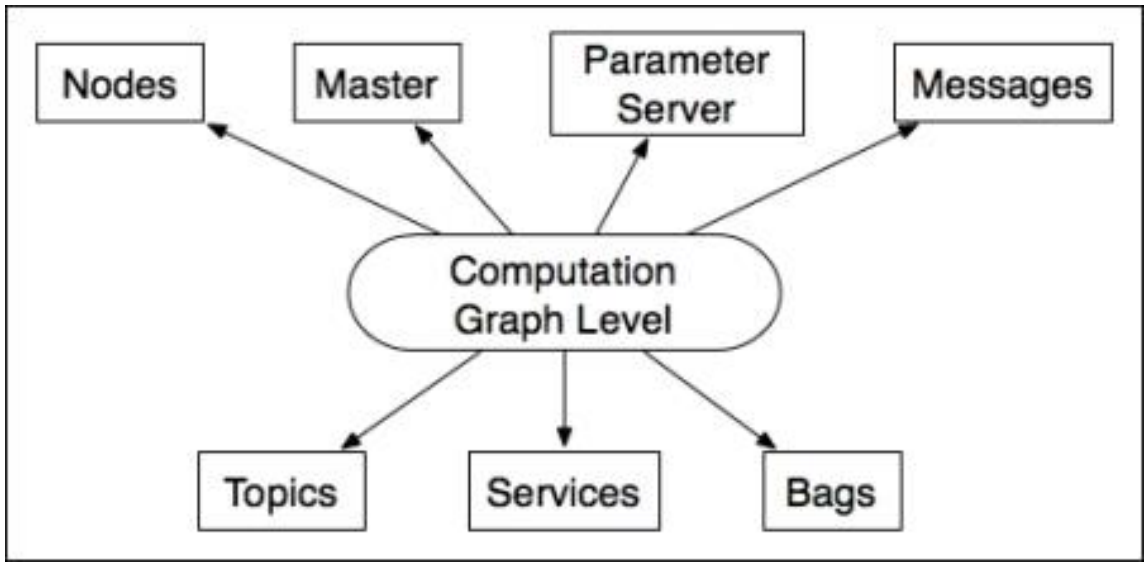
```
$ catkin_create_pkg first_pkg std_msgs rospy roscpp
```

2.3.7 ROS Packages

2.3.8 ROS Graph

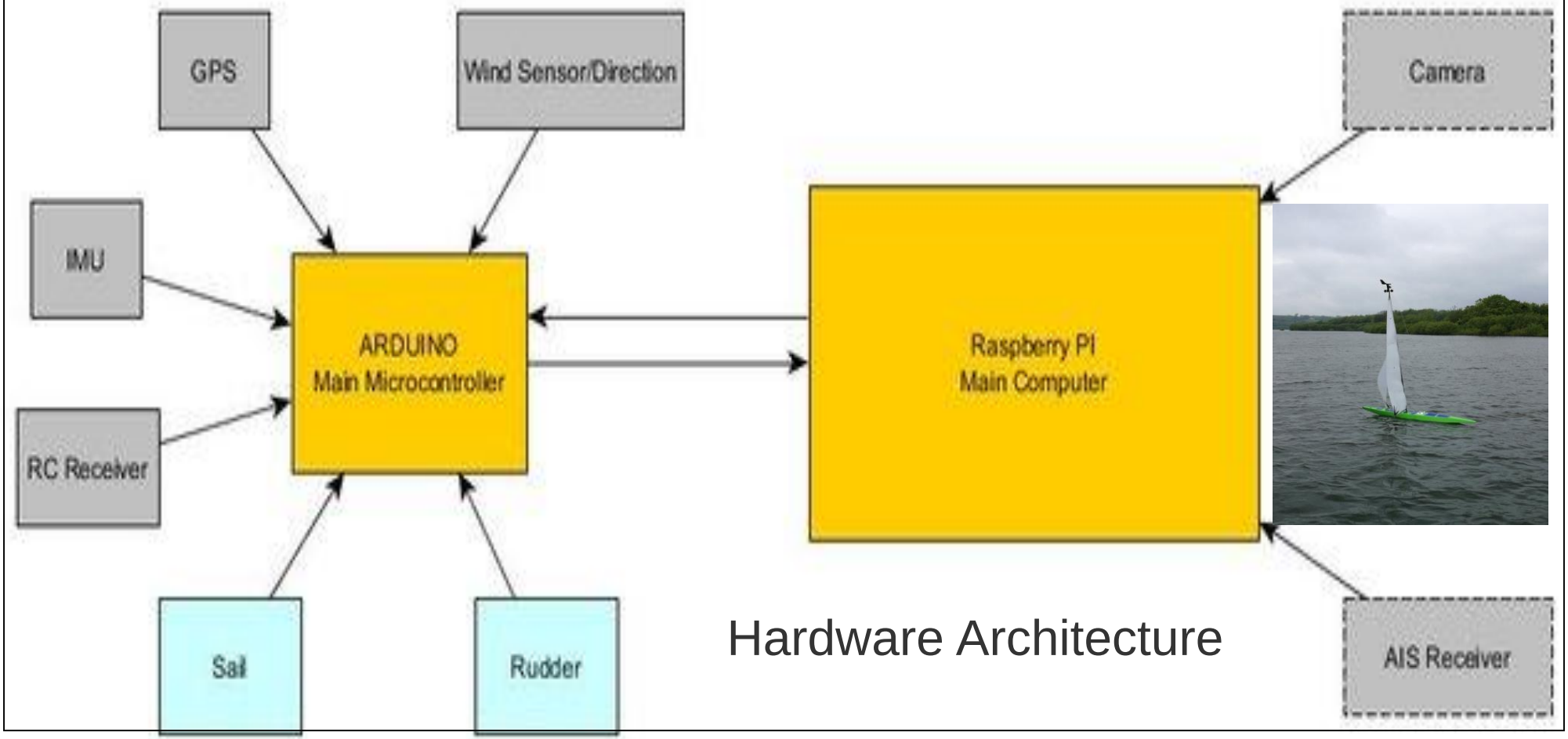
Understanding the ROS Computation Graph level

- ROS creates a network where all the processes are connected.
- Any node in the system can access this network, interact with other nodes, see the information that they are sending, and transmit data to the network.



2.3.8 ROS Graph

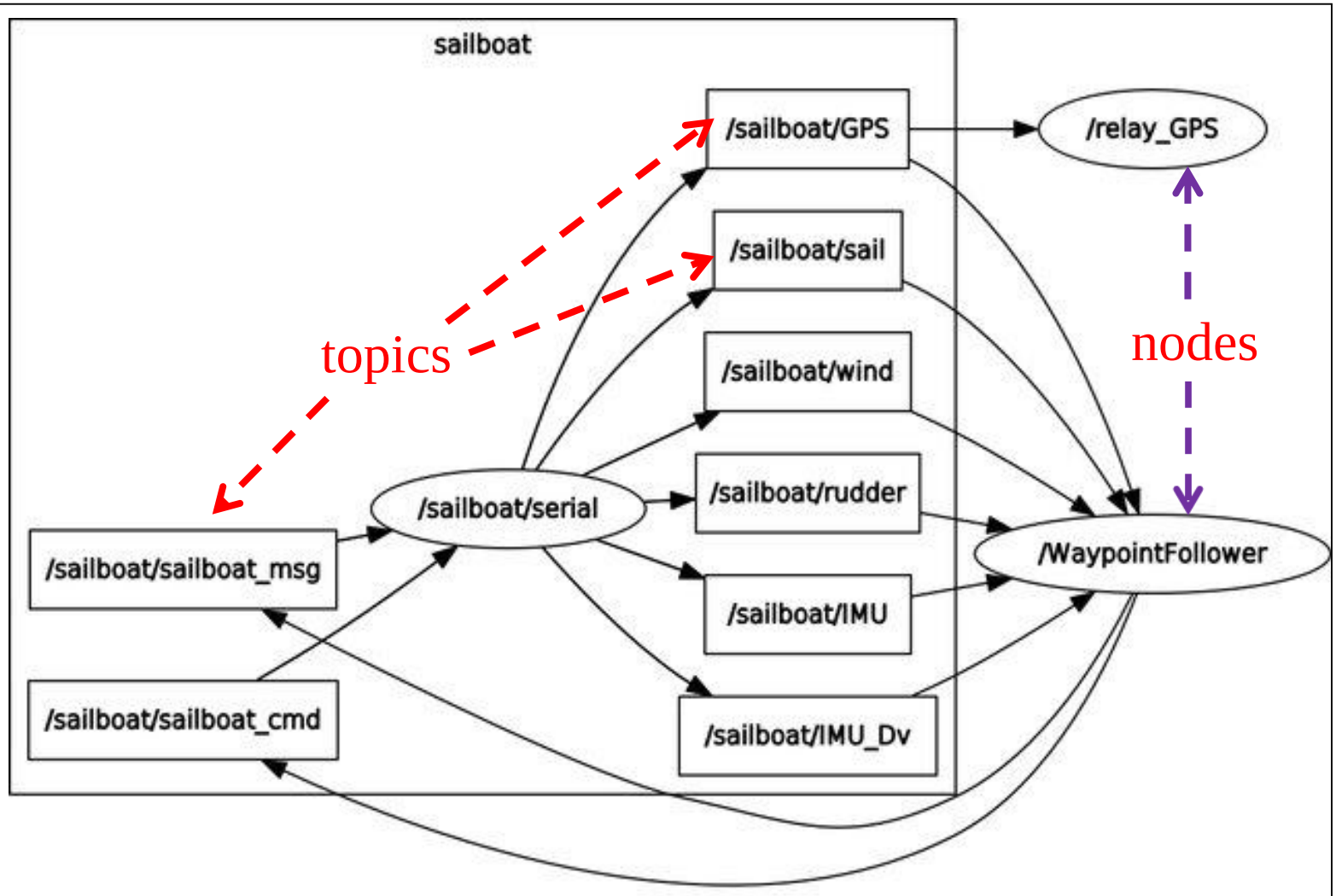
Development & Test of an Open Source Autonomous Sailing Robot with Accessibility, Generality and Extendibility, International Robotic Sailing Conference 2018



Hardware Architecture

2.3.8 ROS Graph

A graph of ROS topics and nodes when applying a waypoint-following control.



2.4 ROS Basic Commands

There are four ROS basic commands as follows:

roscore **rosrun** **rosnode** **rostopic**

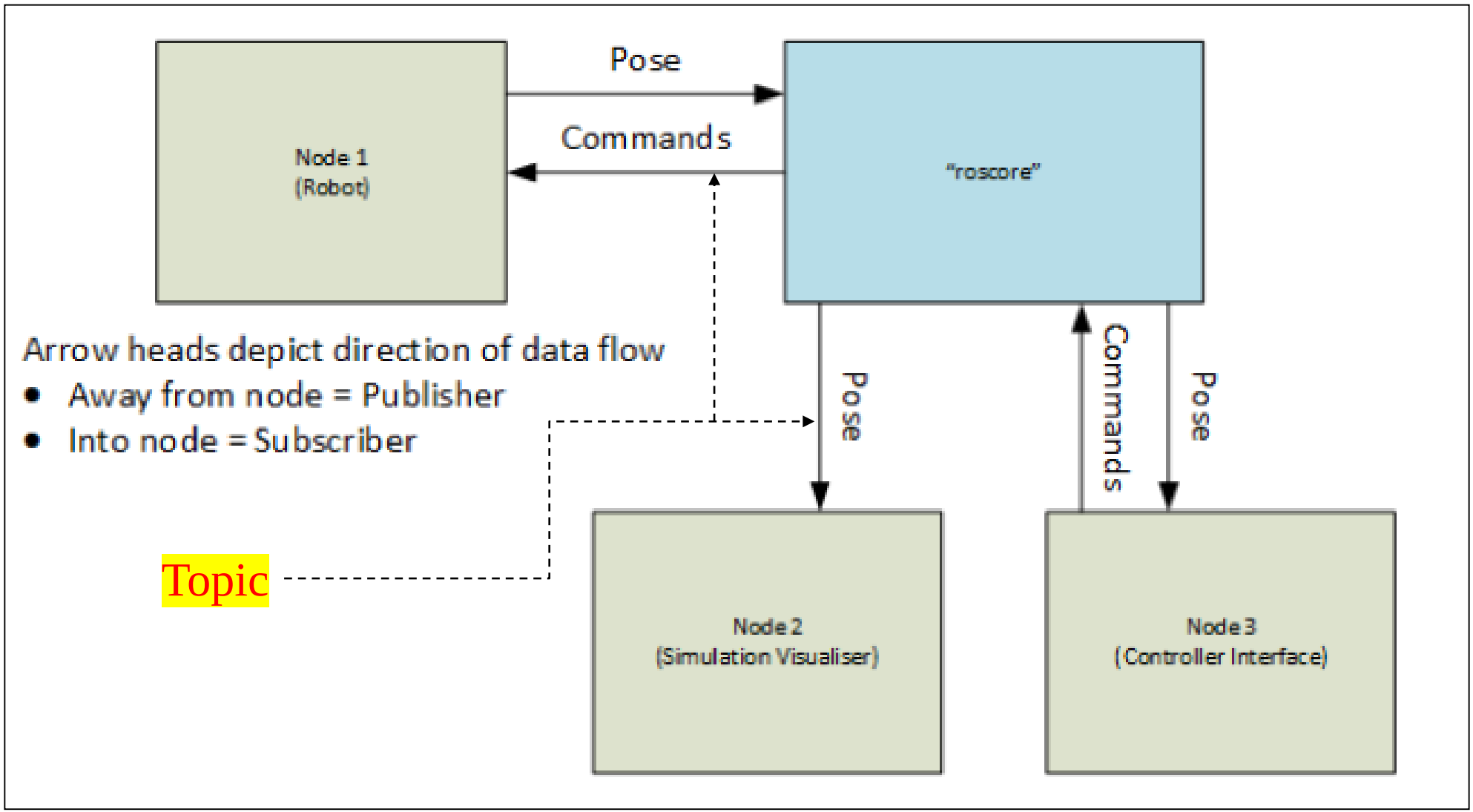
2.4.1 **roscore**

- roscore is the first thing you should run when using ROS

```
$ roscore
```

- roscore will start up:
  - a ROS Master
  - a ROS Parameter Server
  - a rosout logging node

2.4.1 roscore


Node 1
(Robot)
Pose
Commands
"roscore"
Arrow heads depict direction of data flow
Away from node = Publisher
Into node = Subscriber
Topic
Pose
Commands
Pose
Node 2
(Simulation Visualiser)
Node 3
(Controller Interface)

2.4.2 rosrun

- rosrun allows you to run a node
- Usage:

```
$ rosrun <package> <executable>
```

- Example:

```
$ rosrun turtlesim turtlesim_node
```

Run a talker demo node with

```
> rosrun roscpp_tutorials talker
```

2.4.3 rosnode

- Displays debugging information about ROS nodes, including publications, subscriptions and connections

Command	
$rosnode list	List active nodes
$rosnode ping	Test connectivity to node
$rosnode info	Print information about a node
$rosnode kill	Kill a running node
$rosnode machine	List nodes running on a particular machine

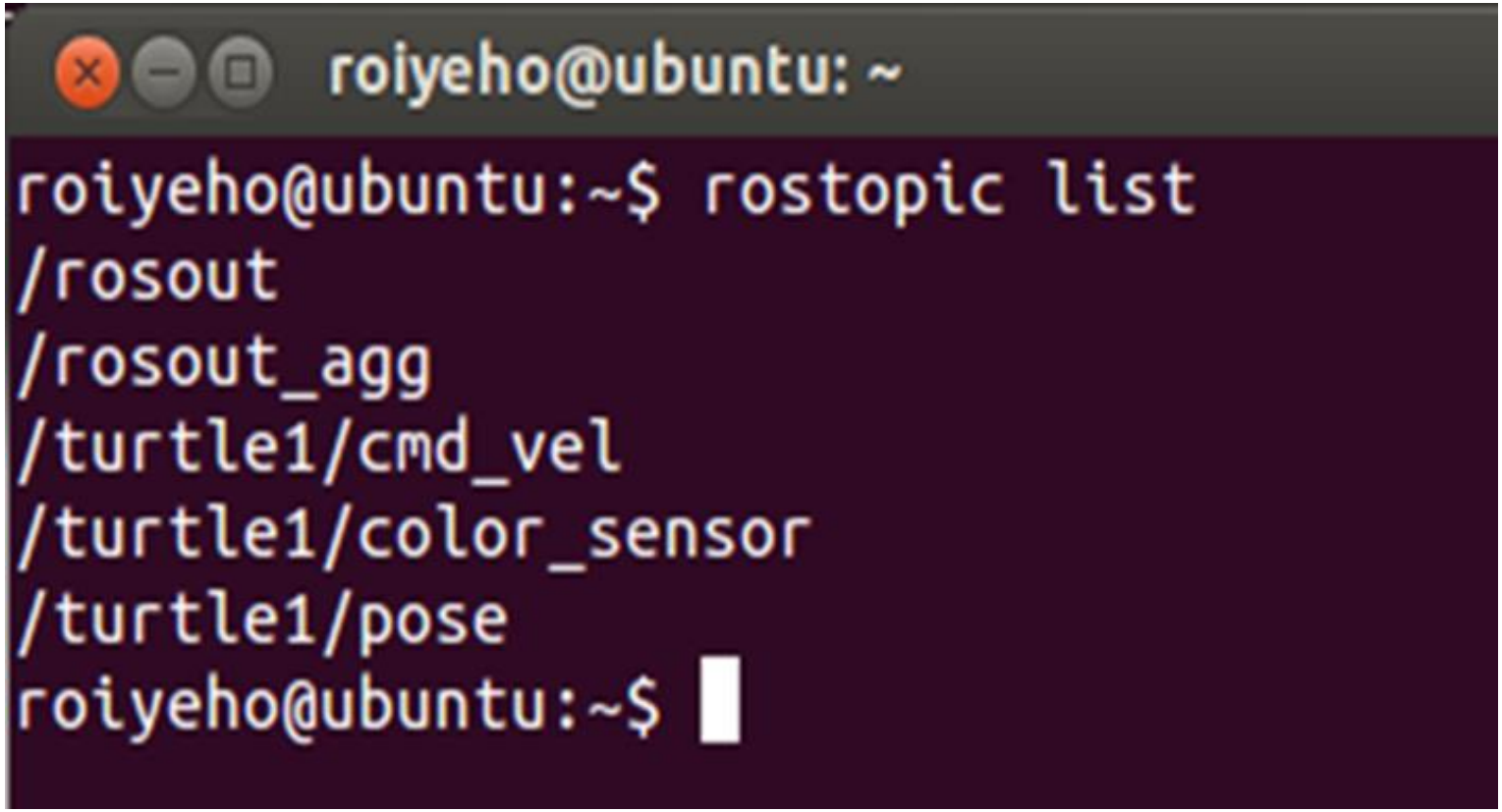
2.4.4 rostopic

- Gives information about a topic and allows to publish messages on a topic

Command	
$rostopic list	List active topics
$rosnode echo /topic	Prints messages of the topic to the screen
$rostopic info /topic	Print information about a topic
$rostopic type /topic	Prints the type of messages the topic publishes
$rostopic pub /topic type args	Publishes data to a topic

2.4.4 rostopic

$rostopic list is to display the list of current topics:



2.5 catkin Build System

- catkin is the official build system of ROS
  - It has a set of tools that ROS uses to generate executable programs, libraries and interfaces

Navigate to your catkin workspace with

```
> cd ~/catkin_ws
```

Build a package with

```
> catkin build package_name
```

Whenever you build a **new** package, update your environment

```
> source devel/setup.bash
```

2.5.1 catkin Workspace

- Catkin workspace contains:

Work here

src

The *source space* contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

Don't touch

build

The *build space* is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

Don't touch

devel

The *development (devel) space* is where built targets are placed (prior to being installed).

If necessary, clean the entire build and devel space with

```
> catkin clean
```

2.5.1 catkin Workspace

```
workspace_folder/          -- WORKSPACE
  src/                     -- SOURCE SPACE
    CMakeLists.txt         -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      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/
```

2.5.2 Creating a catkin Workspace

- Creating a Workspace

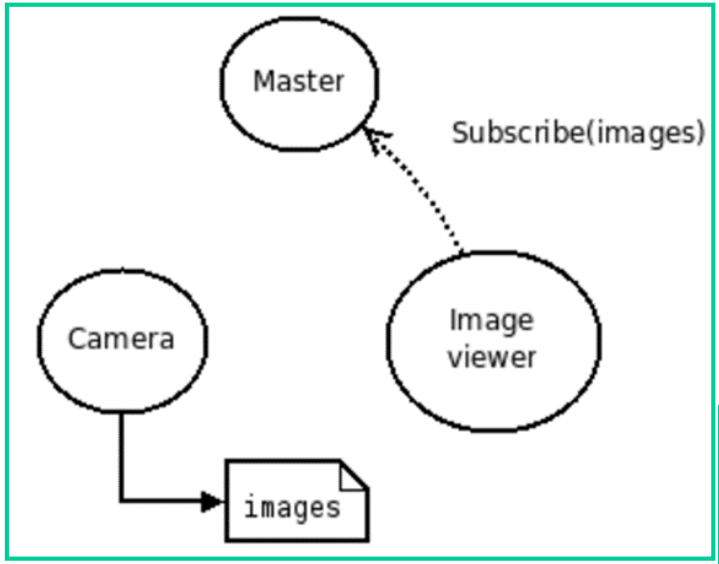
```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

- The workspace initially contain only the top-level CMakeLists.txt
- catkin_make command builds the workspace and all the packages within it

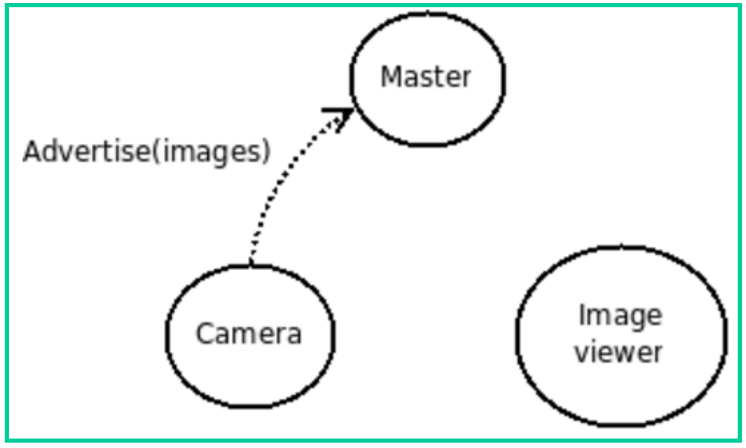
```
cd ~/catkin_ws
catkin_make
```

2.6 ROS Publisher & Subscriber

2.6.1 Principle

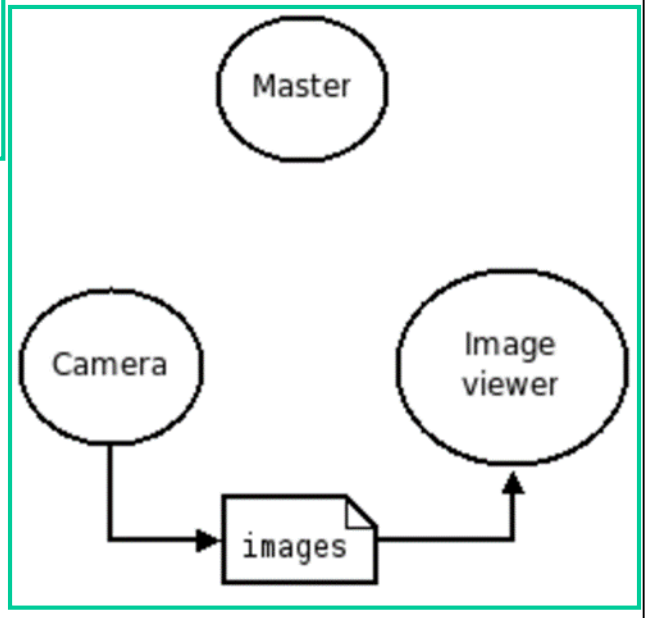


Step 1 Publisher



Step 2 Subscribe

Step 3
Transfer images



2.6 ROS Publisher & Subscriber

2.6.2 Create a ROS Publisher

- Create a publisher with help of the node handle

```
ros::Publisher publisher =
nodeHandle.advertise<message_type>(topic,
queue_size);
```

- Create the message contents
- Publish the contents with

```
publisher.publish(message);
```

2.6.2 Create a ROS Publisher

talker.cpp

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

int main(int argc, char **argv) {
  ros::init(argc, argv, "talker");
  ros::NodeHandle nh;
  ros::Publisher chatterPublisher =
    nh.advertise<std_msgs::String>("chatter", 1);
  ros::Rate loopRate(10);

  unsigned int count = 0;
  while (ros::ok()) {
    std_msgs::String message;
    message.data = "hello world " + std::to_string(count);
    ROS_INFO_STREAM(message.data);
    chatterPublisher.publish(message);
    ros::spinOnce();
    loopRate.sleep();
    count++;
  }
  return 0;
}
```

2.6.3 Create a ROS Subscriber

- Start listening to a topic by calling the method `subscribe()` of the node handle

```
ros::Subscriber subscriber =
nodeHandle.subscribe(topic, queue_size,
                     callback_function);
```

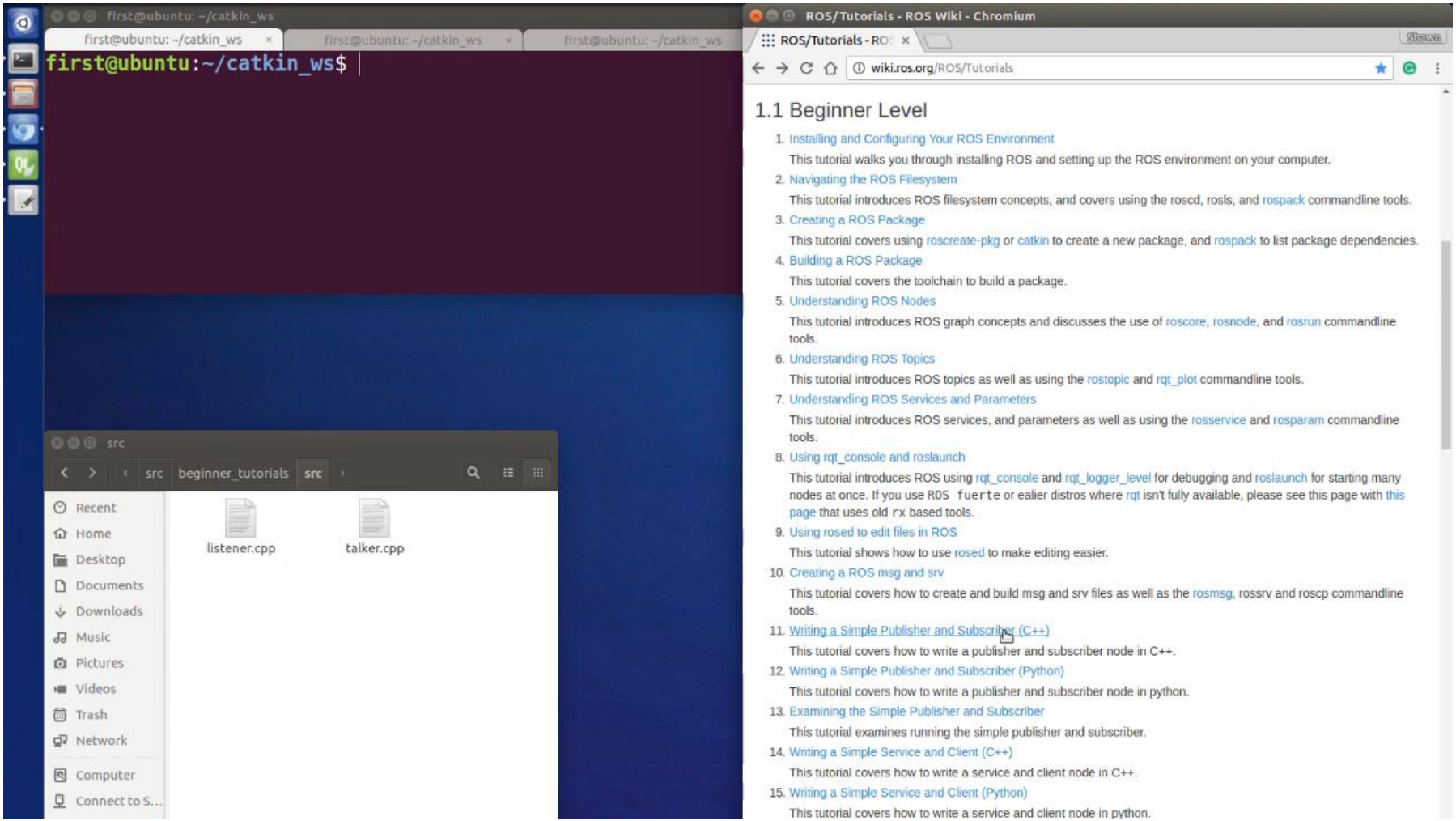
- When a message is received, callback function is called with the contents of the message as argument
- Hold on to the subscriber object until you want to unsubscribe

2.6.3 Create a ROS Subscriber

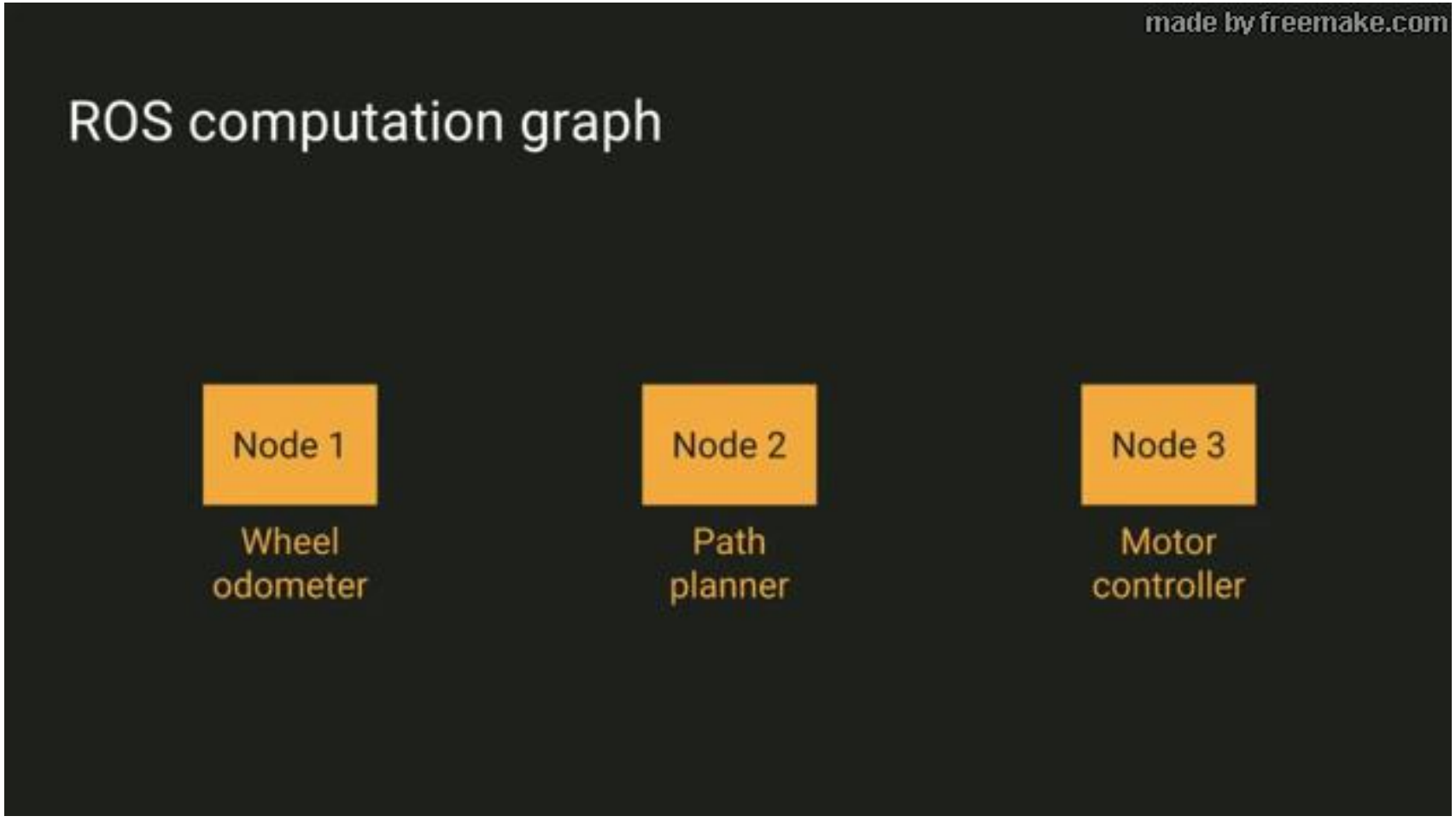
listener.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void chatterCallback(const std_msgs::String& msg)
{
  ROS_INFO("I heard: [%s]", msg.data.c_str());
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle nodeHandle;

  ros::Subscriber subscriber =
       nodeHandle.subscribe("chatter",10,chatterCallback);
  ros::spin();
  return 0;
}
```

2.6.4 ROS Publisher & Subscriber Video -- 1



2.6.5 ROS Publisher & Subscriber Video - 2



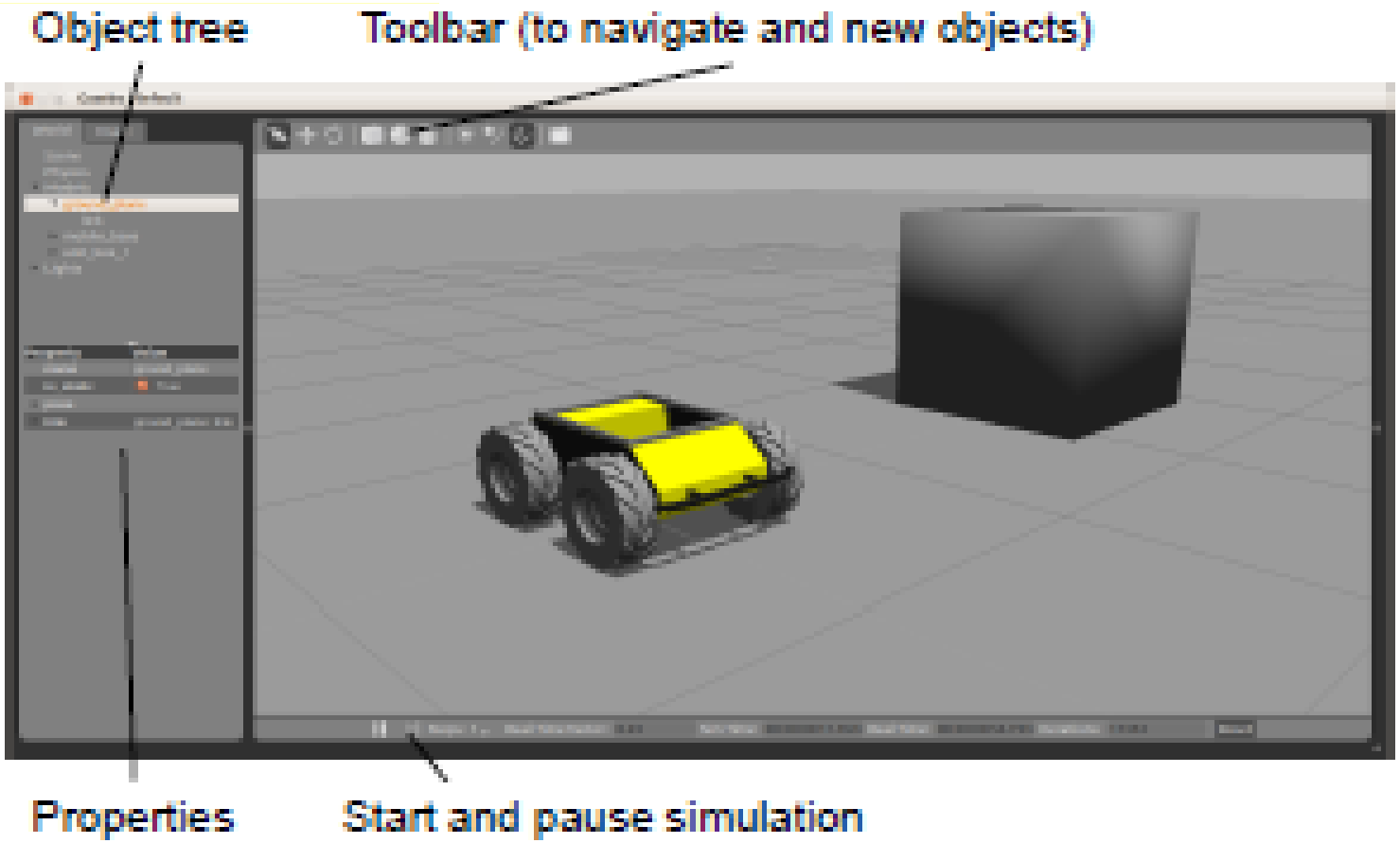
2.7 Gazebo -- 3D simulator

- Gazebo is a multi-robot simulator under active development at the Open Source Robotics Foundation (OSRF)
- It simulates 3D rigid-body dynamics (robots and objects).
- It simulates a variety of sensors including noise.
- It includes a database of many robots and environments (Gazebo worlds)
- It provides a ROS interface
- It allows user code to be designed in an artificial environment at first and operated on a physical robot later.

2.7 Gazebo -- 3D simulator

Run Gazebo with

```
> rosrun gazebo_ros gazebo
```



2.7 Gazebo -- 3D simulator

Gazebo Architecture

- **Server:** Runs the physics loop and generates sensor data
  - *Executable:* gzserver
  - *Libraries:* Physics, Sensors, Rendering, Transport
- **Client:** Provides user interaction and visualization of a simulation.
  - *Executable:* gzclient
  - *Libraries:* Transport, Rendering, GUI
- Gazebo home page - *http://gazebosim.org/*
- Gazebo tutorials - *http://gazebosim.org/tutorials*

2.8 Conclusion

- There are now over 2000 packages and libraries available for ROS.
- There are still many areas of ROS to explore:
  - 3-D image processing using point clouds PCL
  - Identifying your friends and family using face recognition
  - Identifying and grasping objects on a table top
  - Learning from experience using reinforcement learning
- When you are ready, you can contribute your own package(s) back to the ROS community.

2.8 Conclusion