Robot Autonomy

DR. RISMAN ADNAN MATTOTORANG

TELKOM UNIVERSITY

What we will learn today

W2: Introduction to Robot Operating System

Skills you will gain:

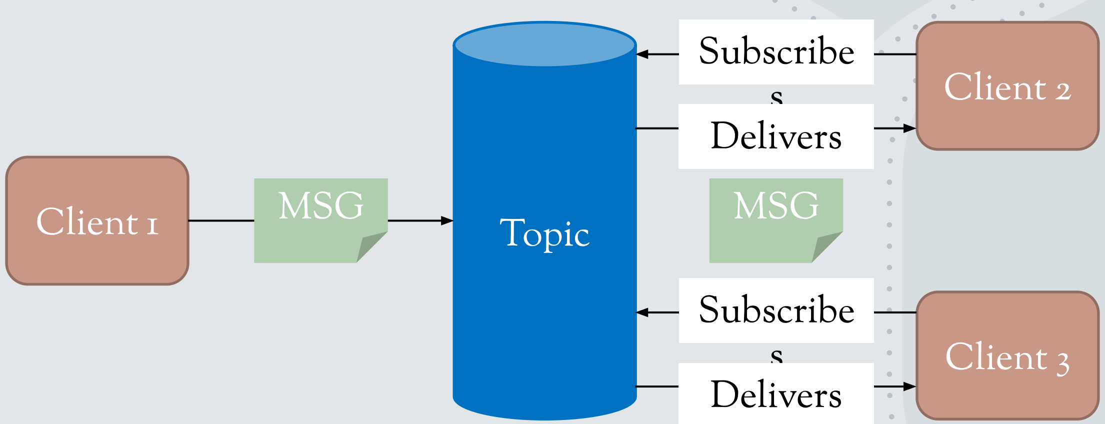
- How to Install ROS2
- How to use Simulator
- How to connect Simulator to ROS2

Writing Software for Robotics

- Robotics requires very complex software
- The software you will deal has way more moving parts than what you've dealt with in most other classes…
- We deal with the complexity through modularity
- We enable modularity by following the right design pattern: "a general, reusable solution to a commonly occurring problem within a given context in software design" – Wikipedia

The Pub/Sub Design Pattern

- We divide our software into individual components
- We define "topics" (think chat rooms) where components can broadcast information to anyone listening
- Each component can:
  - *Publish*: send messages to a topic regardless of whether someone is listening or not
  - *Subscribe*: receive messages on a topic if anyone is sending them regardless of who



Note: there are countless ways to **IMPLEMENT** pub/sub!

You already use Pub/Sub every day! Where???

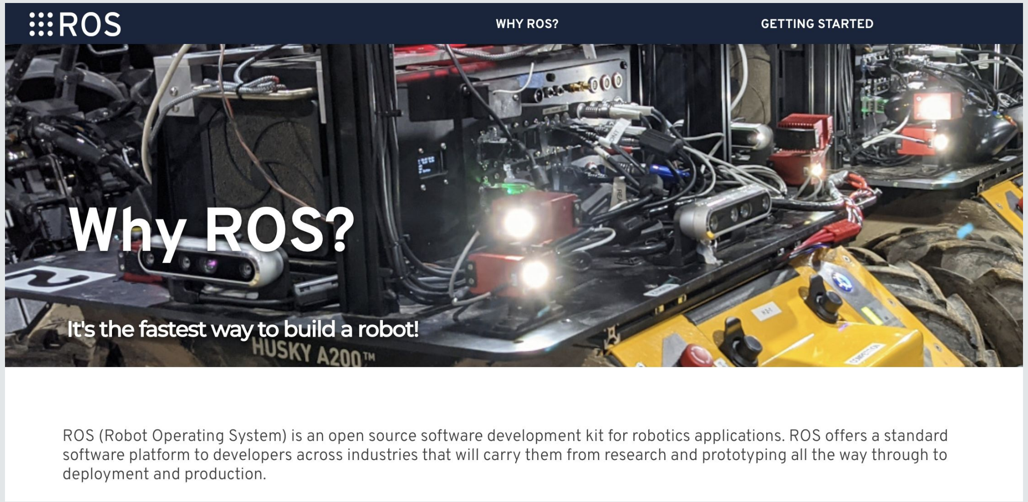
Alternatives to Pub/Sub

- Request/Reply (RPC)
- Push/Pull
- Data binding (e.g. shared data members)
- Observers

What is ROS?

Depending on who you are talking to...

- An implementation of pub/sub geared towards robotic applications and that is network-aware
- Lots of open-source software shared by the community:
    - SLAM (gmapping, amcl)
    - Vision (OpenCV, PCL, OpenNI)
    - Arm Navigation (MoveIt)
    - Simulation (Gazebo)

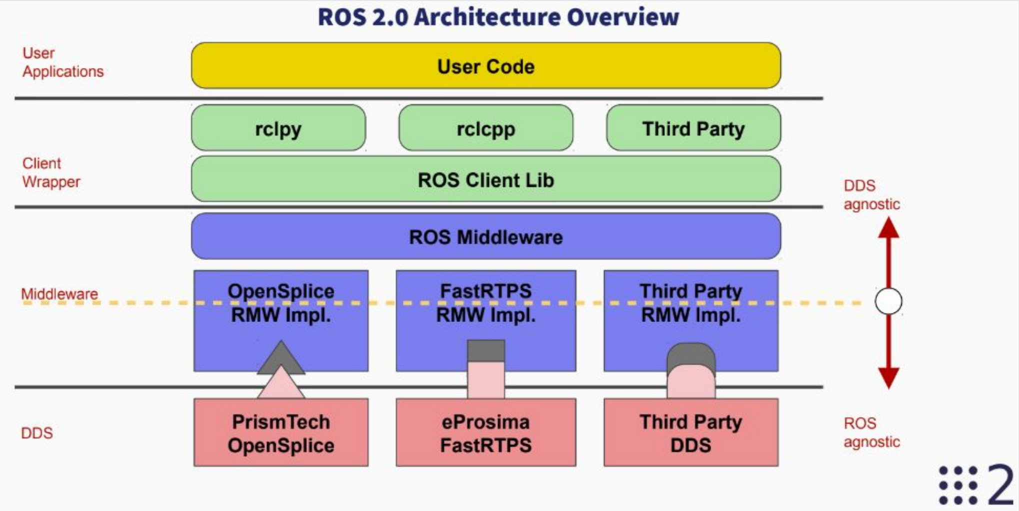


Are there "Alternatives" to ROS?

- LCM
- Drake
- Player
- YARP
- Orocos
- MRPT
- And many others!

Why is ROS popular in industry?

- Not reinventing the wheel is generally good
- Robotics is hard! It's great to offload some of the work to smart people
- ROS is now 12 years old and still going strong

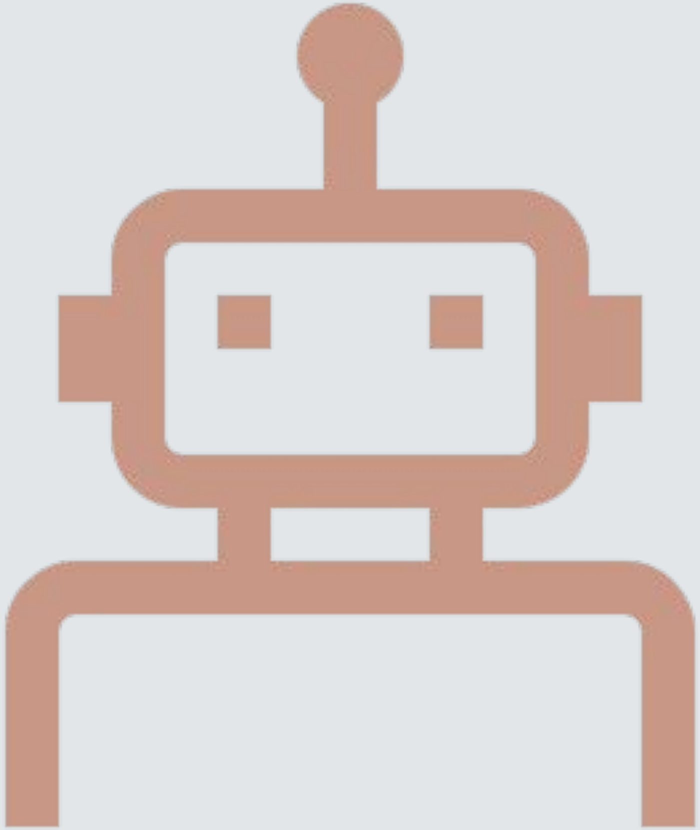


Why are we using ROS in Telyu?

- The closest thing we have to an “industry standard”
- It’s an insurance policy for you (stability, online teaching resources)

ROS Integrates Existing Projects

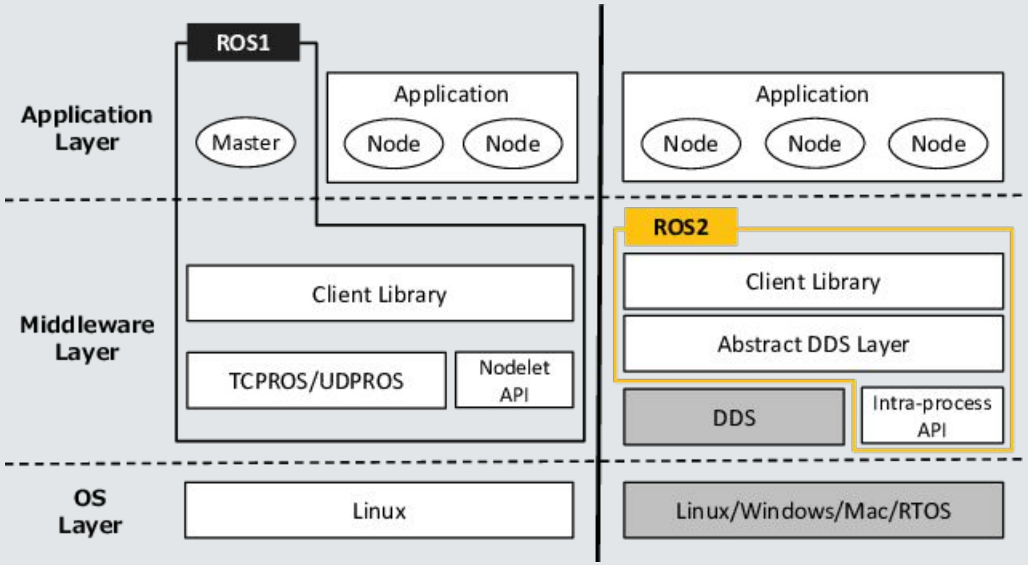
- OpenCV (computer vision)
- Stage, Gazebo (simulation)
- OpenSLAM (navigation)
- Orocos KDL (arm navigation)
- Many ROS “wrappers” to existing software



The Main Software Components

Decentralized Architecture:

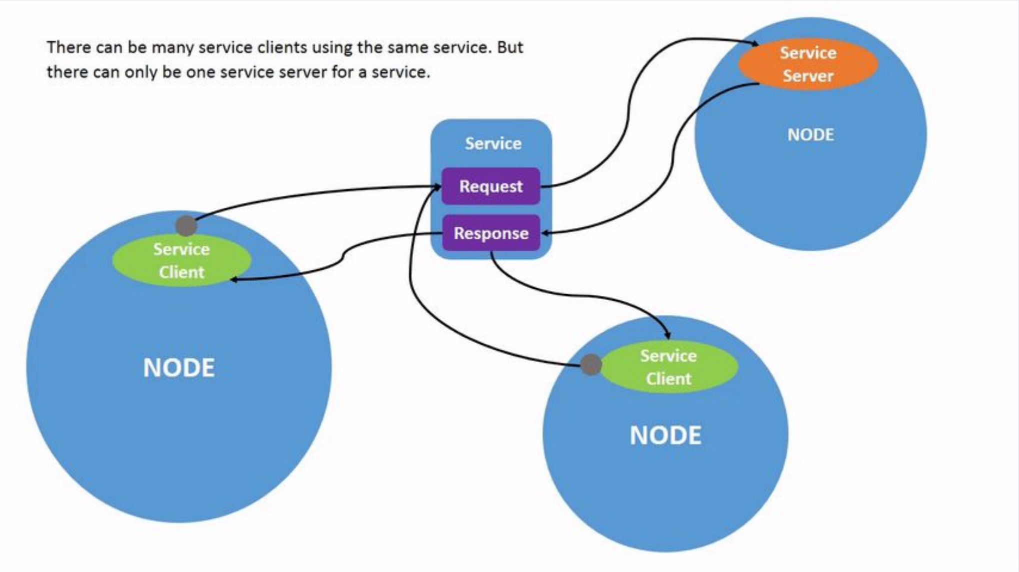
- Master
- Nodes
- Nodes talk to each other over topics (think chat rooms). Master coordinates the whole thing
- Message types: abstraction away from specific hardware
  - Camera image
  - Laser scan data
  - Motion control



ROS Node

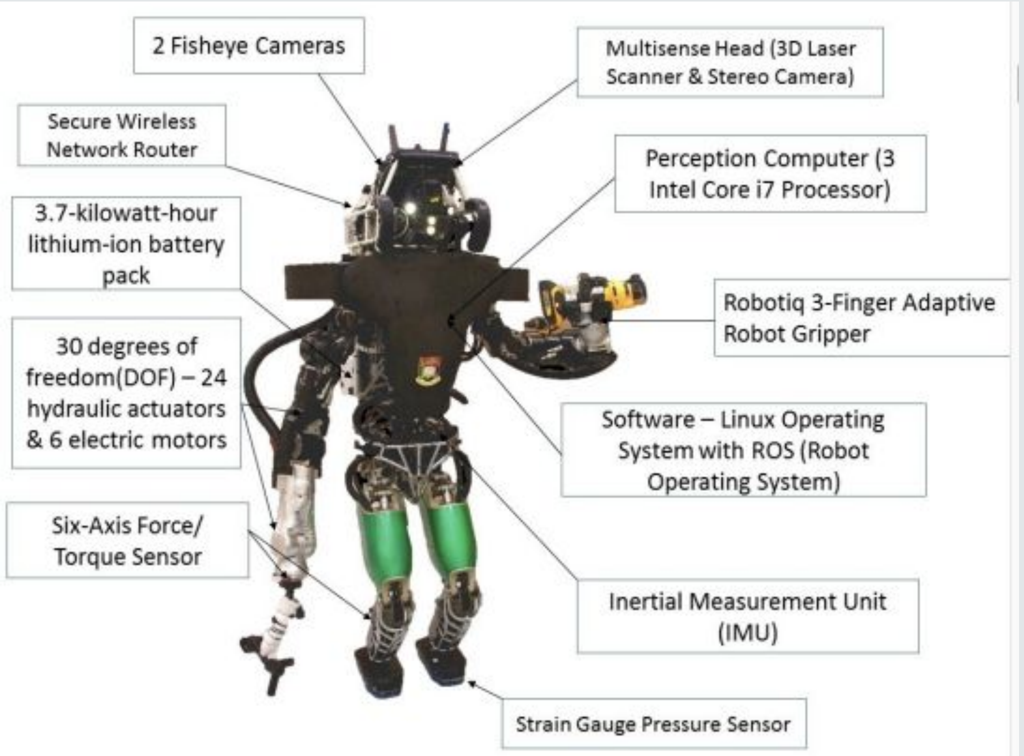
- A process (typically Python or C++) that runs some computation
- The "fundamental" building block
- Can act as a subscriber, publisher or both
- Nodes talk to each other over "topics"
- Run them using rosrun
- Initialize using rospy.init_node()

Note: nodelets are different. They are not individual processes, they share memory



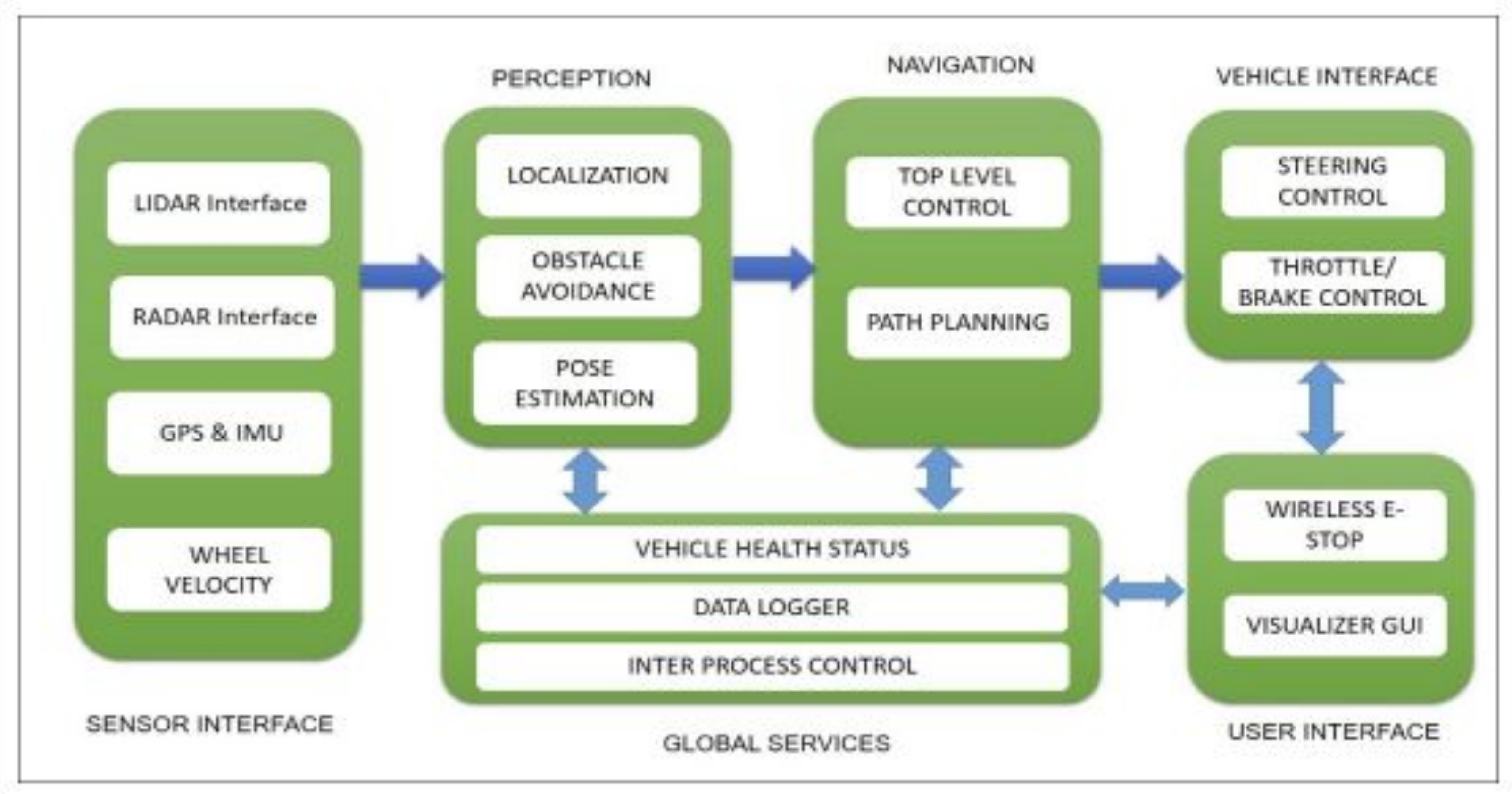
Node Example

Sensors and **actuators** are wrapped in self-contained, reusable software containers called “nodes”



Node Example

Higher level operations also become nodes in the ROS computational architecture



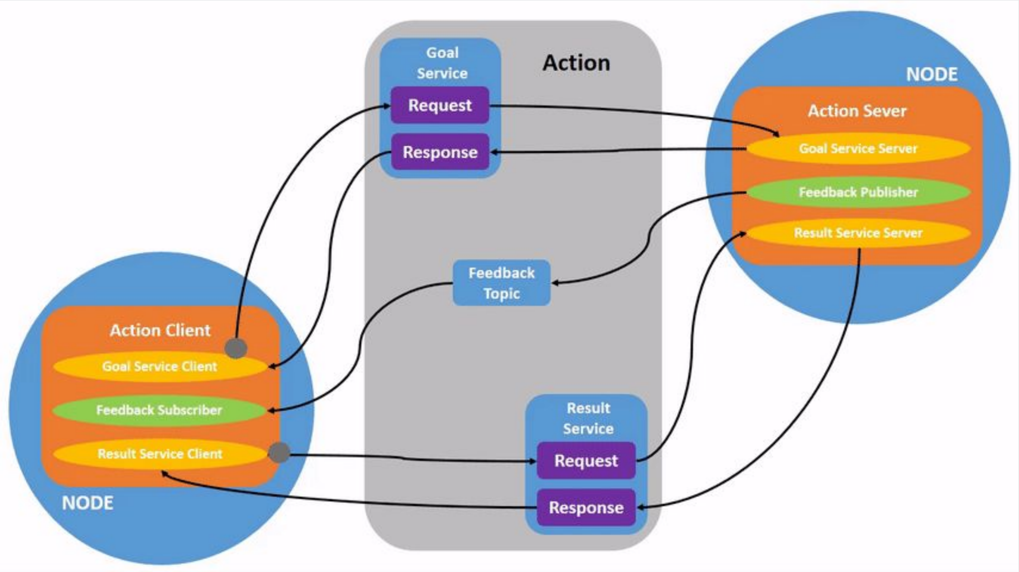
More Concrete Node Examples

LiDAR node publishes laser scan arrays

Camera node publishes RGB images (+depth if RGBD) and camera info (resolution, distortion coefficients)

Mobile robot controller publishes odometry values (e.g. x-y coordinates and velocities, +z for UAVs or underwater vehicles)

Navigation node subscribes to LiDAR and odometry messages, publishes motion control messages



ROS Master

A process that is in charge of coordinating nodes, publishers and subscribers

Also provides a global parameter server

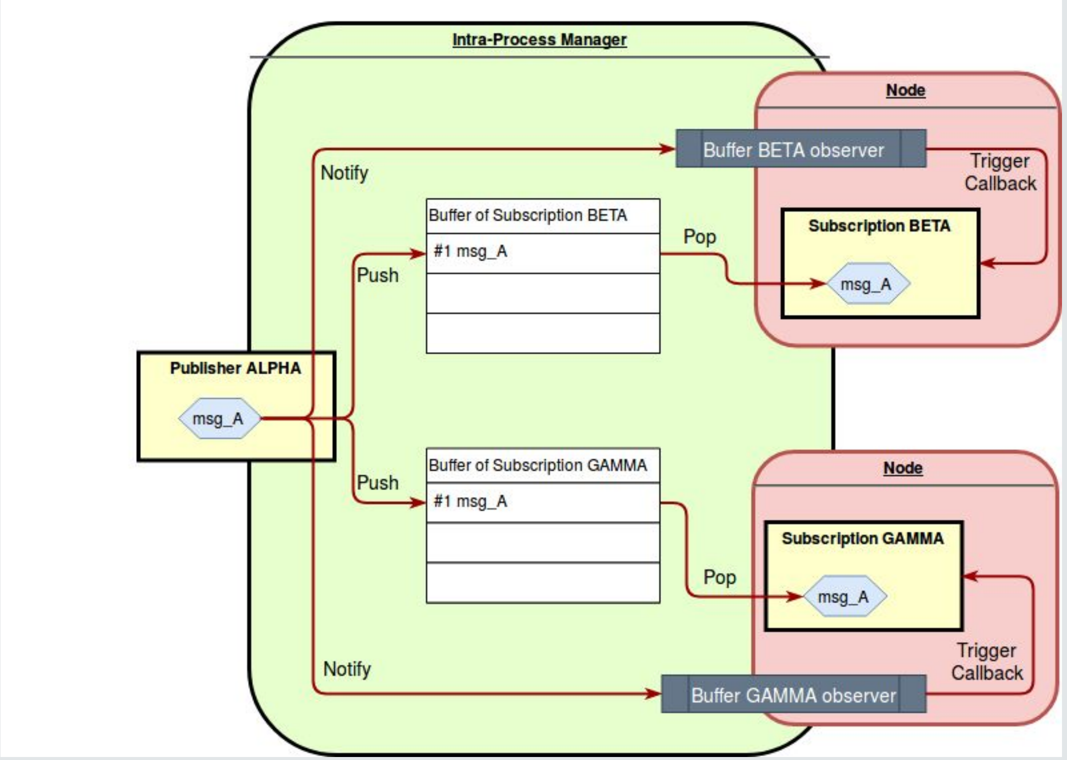
Exactly one of them running at any time

Messages do NOT go through Master (i.e. peer-to-peer)

Nodes will not be able to find each other without Master

Sending Message

- pub = rospy.Publisher()
- msg = …
- pub.publish(msg)



ROS Node - Publisher

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def talker():
    rospy.init_node('talker', anonymous=True)

    pub = rospy.Publisher('chatter', String, queue_size=10)

    rate = rospy.get_param('~rate', 1)
    ros_rate = rospy.Rate(rate)

    rospy.loginfo("Starting ROS node talker...")

    while not rospy.is_shutdown():
        msg = "Greetings humans!"

        pub.publish(msg)
        ros_rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Monitoring Messages

You can check if you are sending messages using the rostopic command line tool:

- `rostopic list – lists all the active topics`
- `rostopic echo <topic> – prints messages received on <topic>`
- `rostopic hz <topic> – measures topic publishing rate`

Receiving Messages

- `rospy.Subscriber("chatter", String, callback)`
- `def callback(msg): …`

(in C++ need to call spinOnce(), not in Python)

ROS Node - Subscriber

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(msg):
    rospy.loginfo("Received: %s", msg.data)

def listener():
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    rospy.loginfo("Listening on the chatter topic...")

    rospy.spin()

if __name__ == '__main__':
    listener()
```

ROS Launch Files

- Simple XML files that allow you to
  - Launch multiple nodes at once
  - Set parameters for those nodes
  - Start Master

- `roslaunch .launch`

ROS Launch File Example

```
<launch>
   <!-- Start the talker node -->
   <node name="talker" pkg="aa274" type="talker.py”
output="screen">
      <param name="rate" value="5"/>
   </node>
</launch>
```

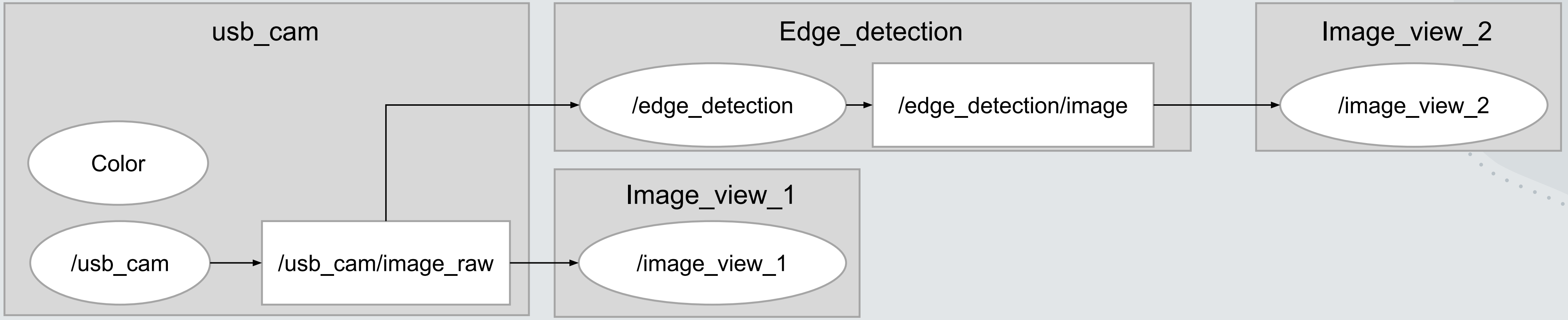
A Case Study

Edge detection in camera images

- Node 1 – **Camera Driver** (Subscribes to: Nothing Publishes: Camera images)
- Node 2 – **Edge Detection** (Subscribes to: Camera images Publishes: Image with edges)
- Node 3 – **image_view** (Subscribes to: Camera images Publishes: Nothing)
- Node 4 – **image_view** (Subscribes to: Image with edges Publishes: Nothing)

A Case Study

- Edge detection in camera image
- rqt_graph



ROS Launch File for Edge Detection

```
<launch>
  <arg name="video_device" default="/dev/video0" />

  <include file="$(find aa274)/launch/usbcam_driver.launch">
     <arg name="video_device" value="$(arg video_device)" />
  </include>

  <node name="image_view_1" pkg="image_view" type="image_view">
    <remap from="image" to="/camera/image_color" />
    <param name="autosize" value="true"/>
  </node>

  <node name="image_view_2" pkg="image_view" type="image_view">
    <remap from="image" to="/edge_detection/image" />
    <param name="autosize" value="true" />
  </node>

  <node name="edge_detection" pkg="opencv_apps" type="edge_detection">
    <remap from="image" to="/camera/image_color" />
    <param name="debug_view" value="false" />
  </node>
</launch>
```

Developing with ROS

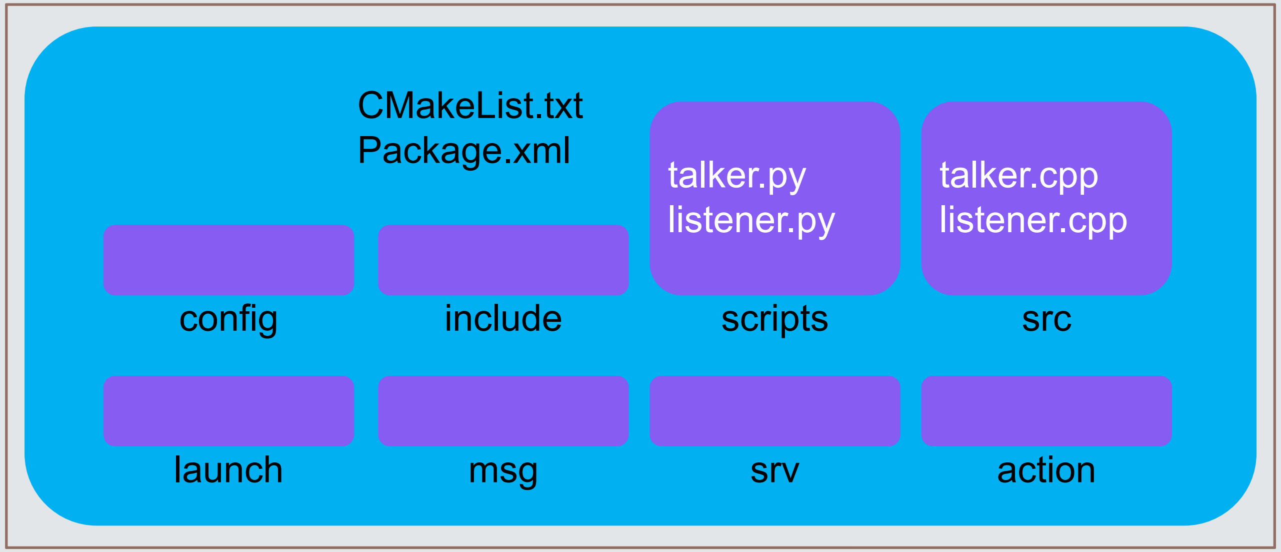
- **Catkin workspace**: a directory that contains all your ROS development
- It sets the right environment variables
- It knows how to compile your nodes (using *cmake which in turn uses a compiler*)

The commands you need to know:

- `mkdir -p ~/catkin_ws/src`
- `cd ~/catkin_ws`
- `catkin_make`

ROS Packages

- The basic organization structure for your nodes
- Usually corresponds to a "functionality" (e.g. a SLAM package)
- Can contain code for multiple nodes
- Directory structure:



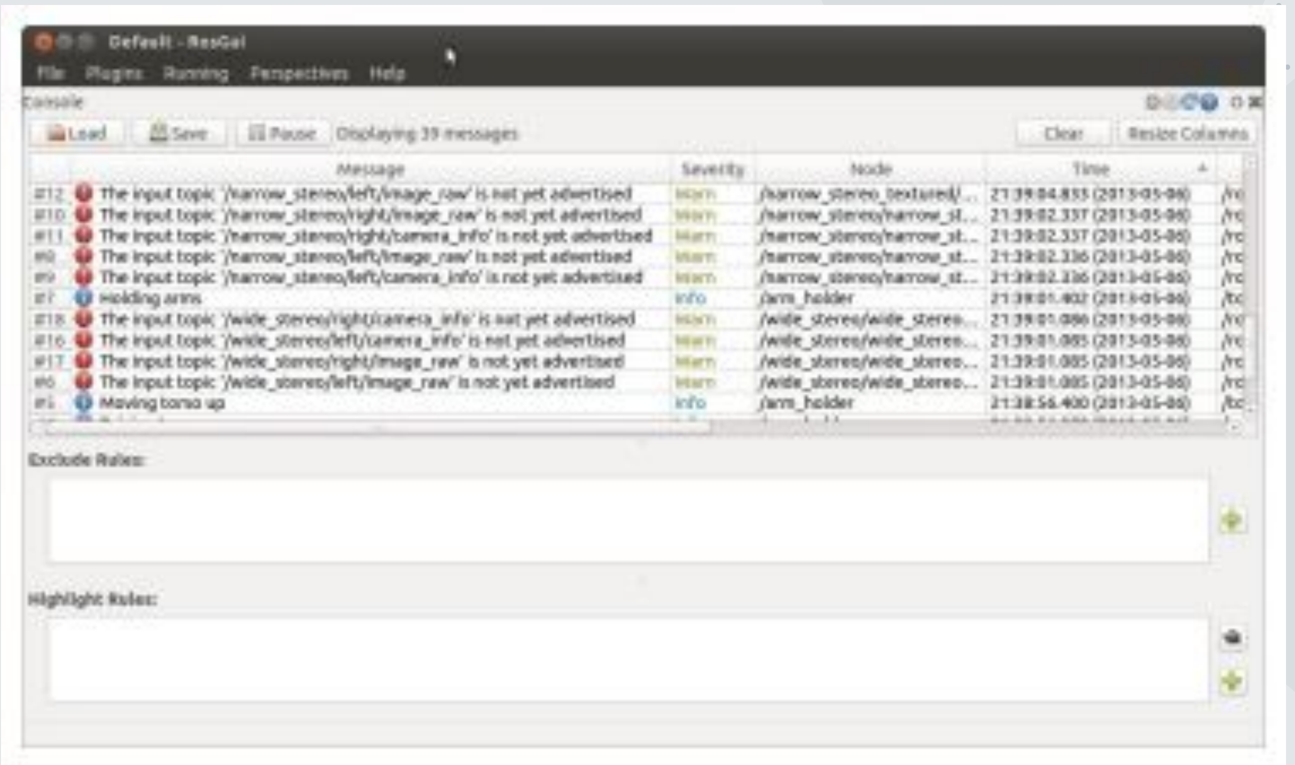
The command you need to know:

```
catkin_create_pkg roscpp rospy
std_msgs
```

Debugging

- `rospy.loginfo()`
- `rqt_console`
- `rosbag record`
- `rosbag play file.bag`

- **`pdb - Python Debugger`**
  - `import pdb`
  - `pdb.set_trace()`



Creating Custom Messages

- Write message definitions (.msg) that are language agnostic
- ROS generates the right files so that roscpp and rospy can use your message

- `rosmsg show student`

```
[aa274/Student]:
string name_first
string name_last
uint8 age
uint32 grade
```

Primitive Type	Serialization	C++	Python
bool (1)	unsigned 8-bit int	uint8_t (2)	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int (3)
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long
uint64	unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ascii string (4)	std::string	str
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

ROS Services

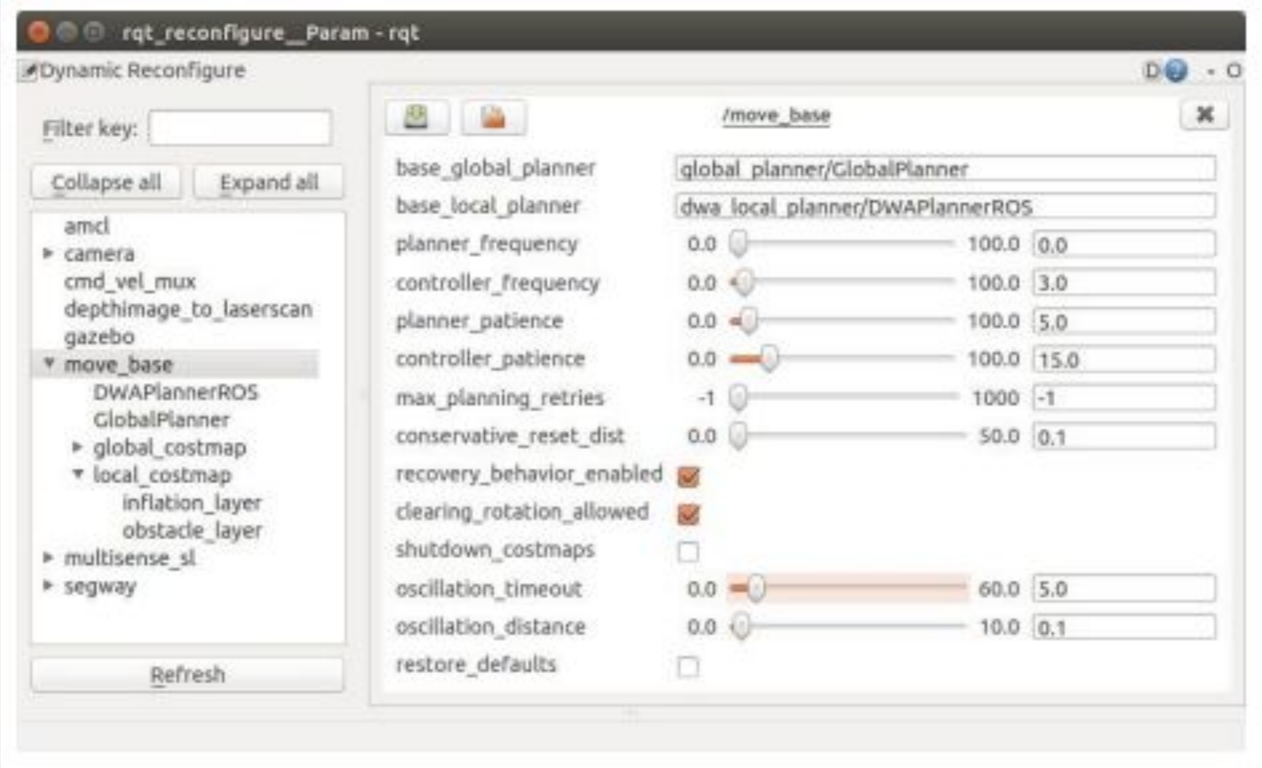
- A different way for nodes to pass messages to each other
- Request/Response scheme (not Pub/Sub!)
- Examples:
  - Turn a light or LED on or off
  - Assign a name to a face and retrain face recognizer
  - Spawn a new model in the Gazebo simulator

The Parameter Server

- Parameters are stored under namespaces; e.g.
  - `/move_base/local_costmap/height`
  - `/usb_cam/framerate`
  - `/gazebo/time_step`
- **Setting and getting parameters:**
  - `rosparam set param_name param_value`
  - `param_value = rospy.get_param("param_name")`
- NOTE: Setting a parameter does not affect a running node!

Dynamic Reconfigure

- Some nodes provide dynamically changeable parameters
  - `rosrun rqt_reconfigure rqt_reconfigure`

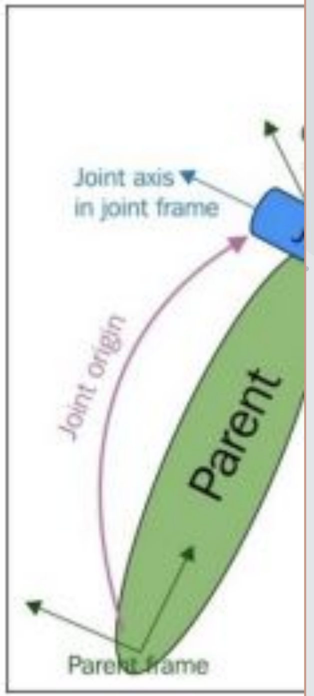


URDF

- Universal Robot Description Format
- An XML file that describes the kinematic chain of your robot

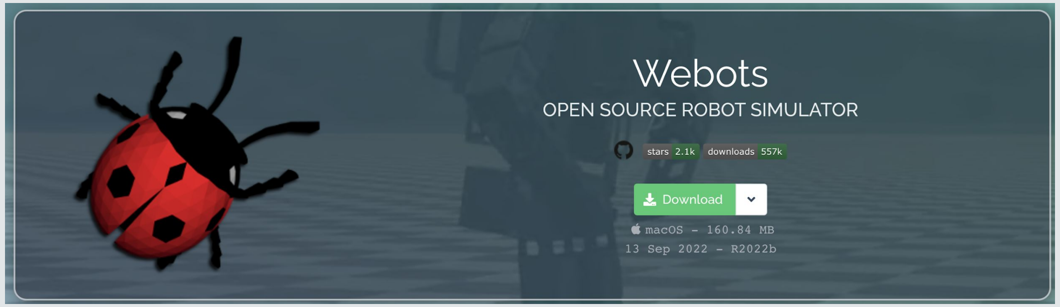
```
<link name="base_link">
  <visual>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
    <material name="blue">
      <color rgba="0 0 .8 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="10"/>
    <inertia ixx="0.4" ixy="0.0" ixz="0.0" iyy="0.4" iyz="0.0" izz="0.2"/>
  </inertial>
</link>
```

```
<joint name="head_swivel" type="continuous">
  <parent link="base_link"/>
  <child link="head"/>
  <axis xyz="0 0 1"/>
  <origin xyz="0 0 0.3"/>
</joint>
```



Simulator

- Same code that will run in production
- Physics is mostly accurate



Learn Interactively

Next Lecture

- Robot Motion Control