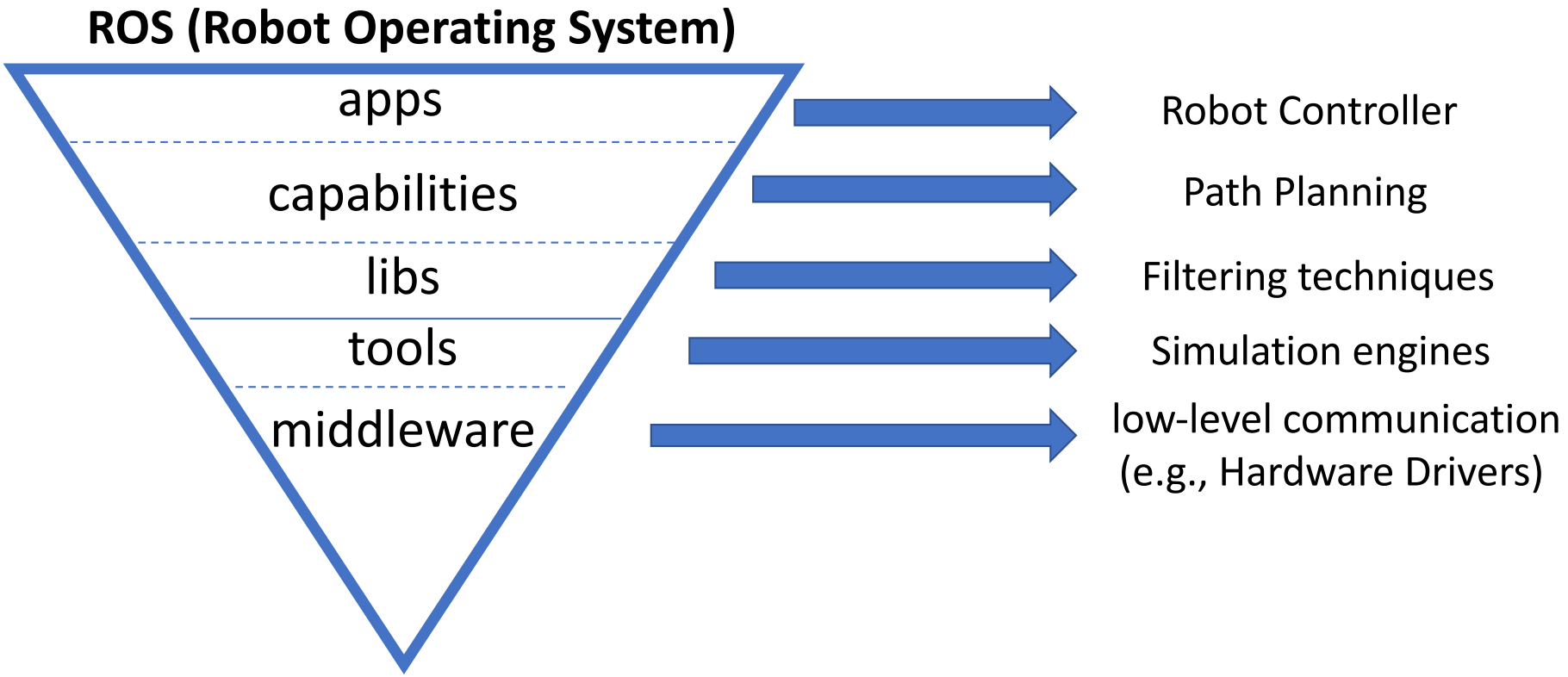
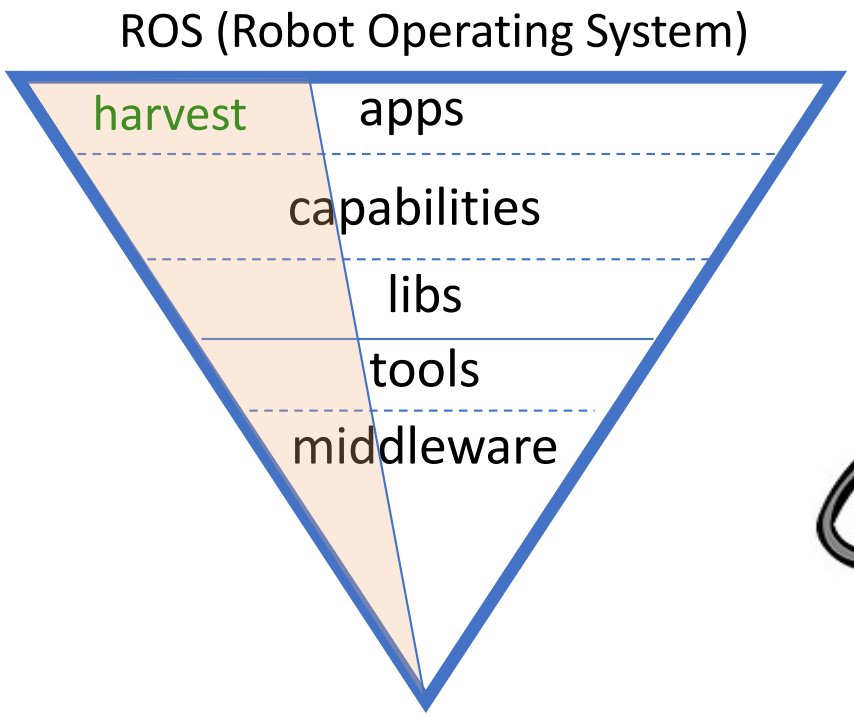
Intro to ROS and Robot Software Development

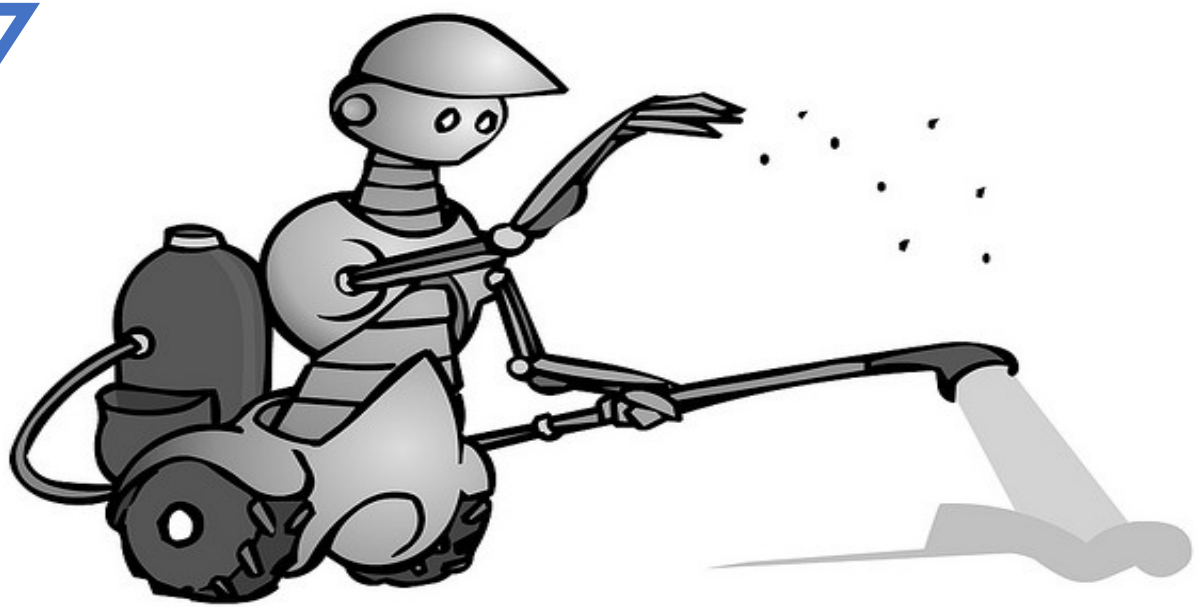
Eduardo Castelló Ferrer
Postdoctoral Fellow
MIT Media Lab
-
ecstll@mit.edu

Intro to the Robotic Operating System (ROS)



Intro to the Robotic Operating System (ROS)





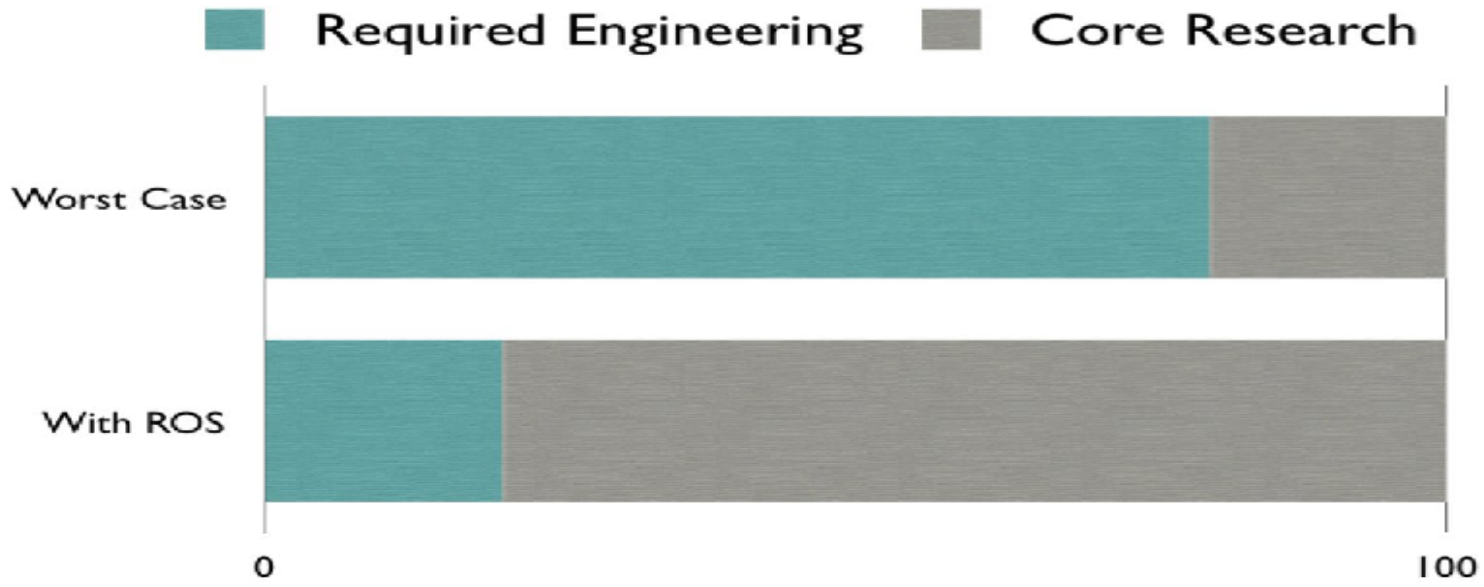
The robotic harverster

History

Keenan Wyrobek and Eric Berger

The concept of a robot operating system started at Stanford University, evolved through Willow Garage, and now resides with Open Robotics Foundation.

History


Required Engineering
Core Research
Worst Case
With ROS
0
100

Willow Garage

Willow Garage founder Scott Hassan

Willow Garage

"I've made a lot of money building web companies on an open-source stack," and he saw creating the Linux of robots

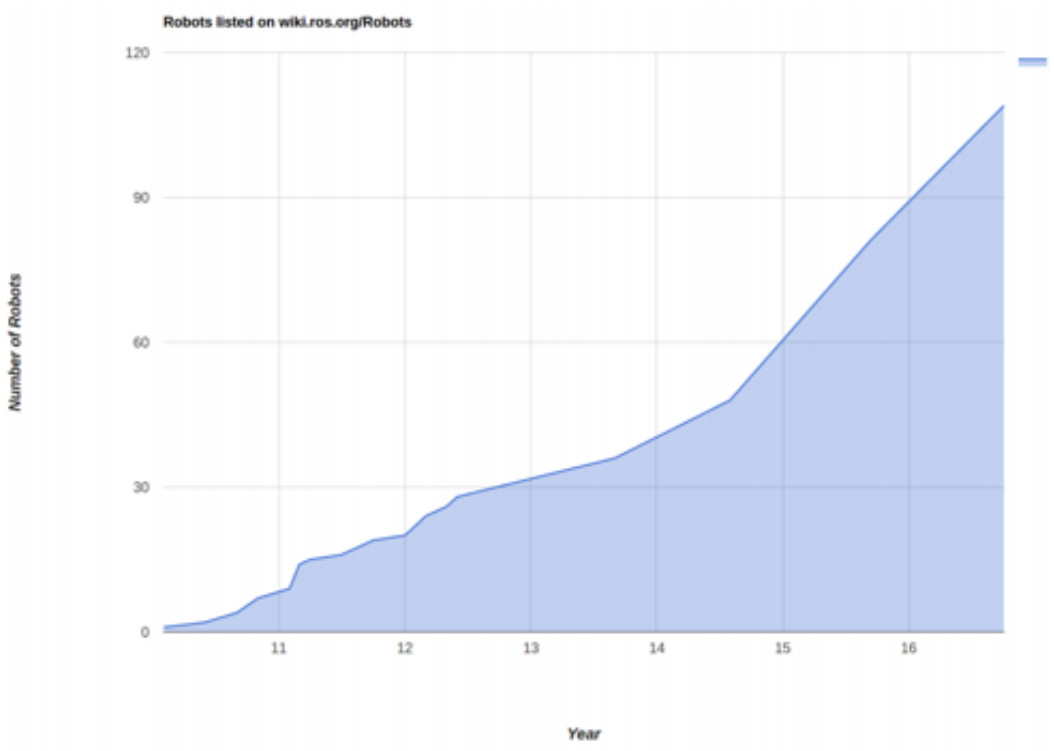
PR2

willow garage pr2 robot

PR2
Order Now

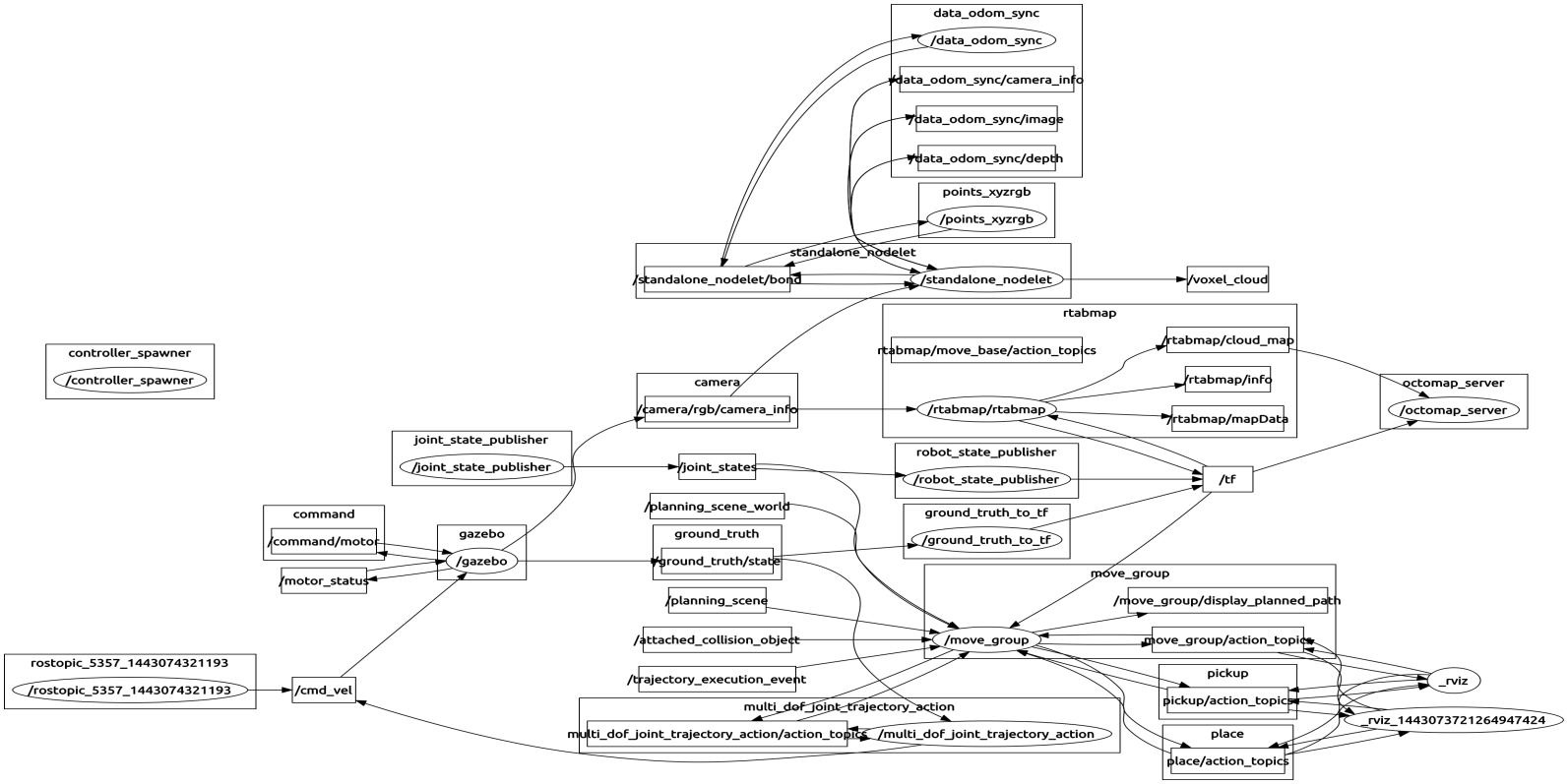
Who wanted this “platform” today:
universities and companies doing R&D.

History



Source: http://robots.ros.org/

How ROS look like internally?

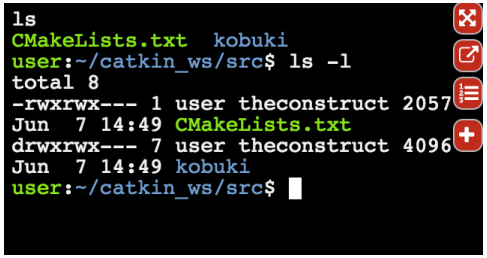


Agenda for today

1. Packages
2. Topics
   - Publishers
   - Subscribers
3. Services
4. Actions
5. Debugging Tools

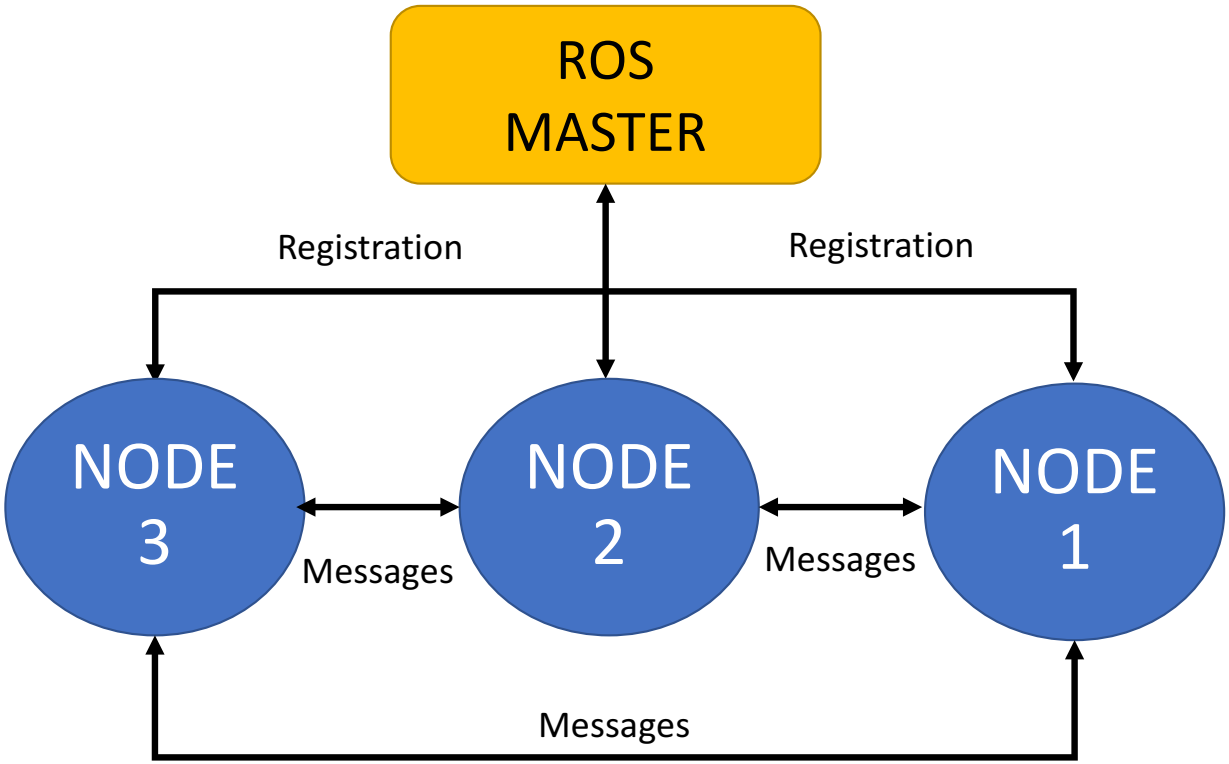
Sign up to the ROS Development Studio

- Visit: http://www.theconstructsim.com/rds-ros-development-studio/
- Sign up and create a student account (use your @mit.edu account - free)
- You will see a dashboard with several simulation options:
  - Check out the section named “Kobuki.zip”
  - Please click the “Open project” button
  - Hit the “Run” button
- Select the Turtlebot 2 robot: 
- Click the full screen button  at the terminal window:



- Type: git clone https://github.com/edcafenet/ros_is_awesome.git
- Type: . ros_is_awesome/install

ROS Overall architecture



Explanation:

You **must** have a roscore running in order for ROS nodes to communicate. roscore wil launch a ROS Master node:

- A ROS Master
- A ROS Parameter Server
- A rosout logging node

Important Commands:

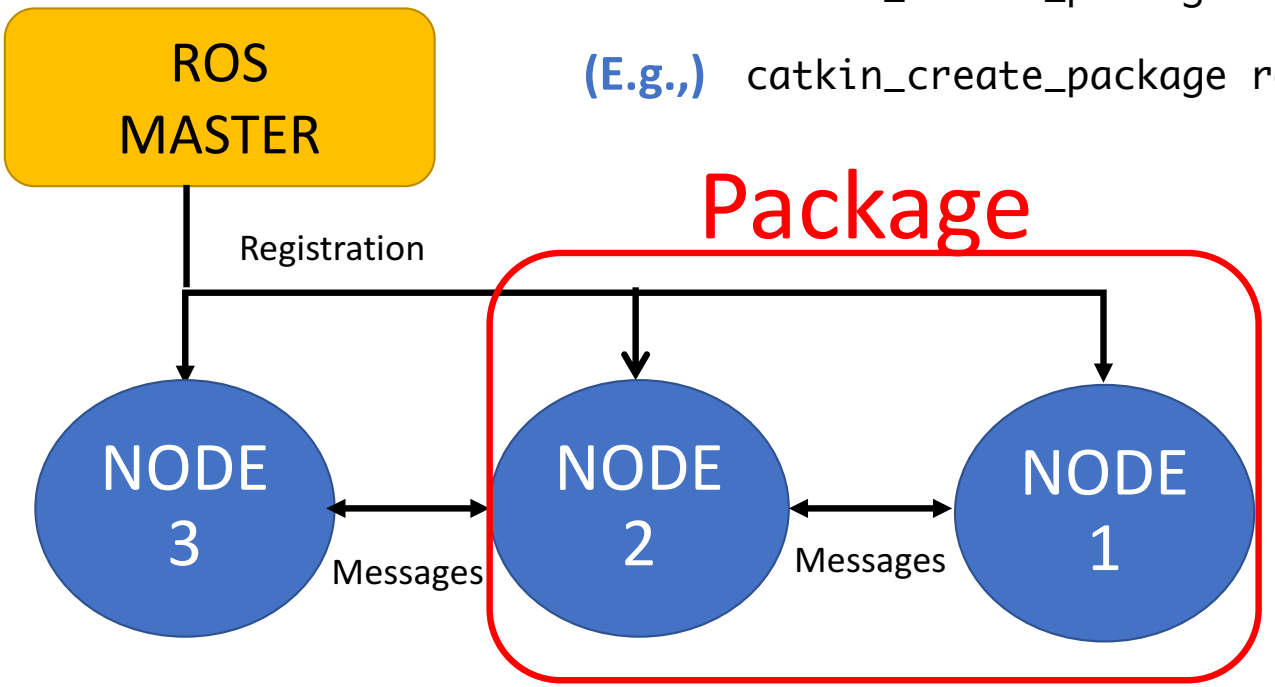
```
roscore
```

Create a ROS package

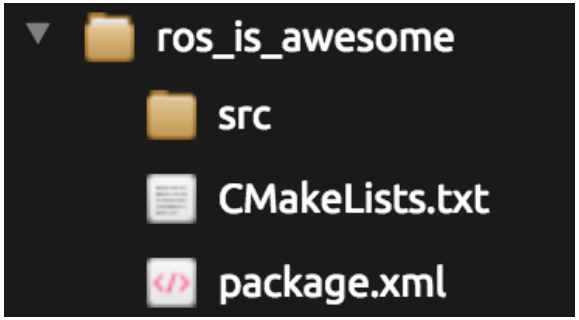
Important Commands:

```
catkin_create_package <package_name> <dependencies>
```

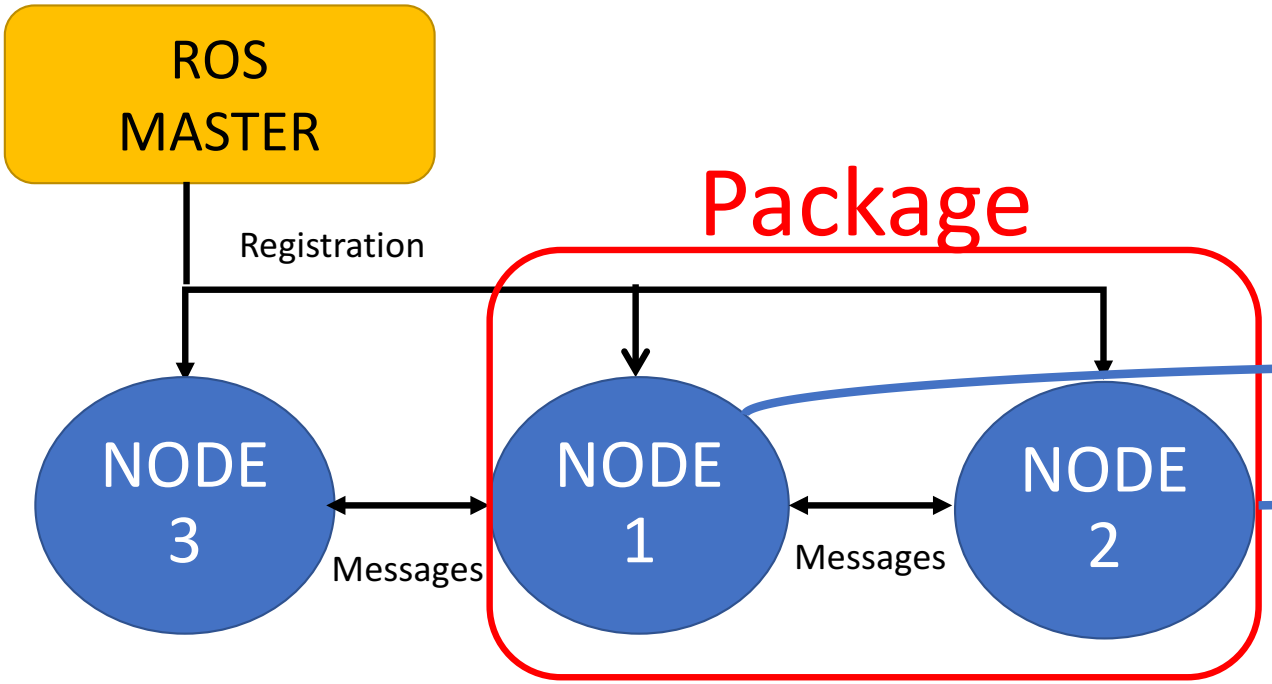
(E.g.,) `catkin_create_package ros_is_awesome rospy`



Result:

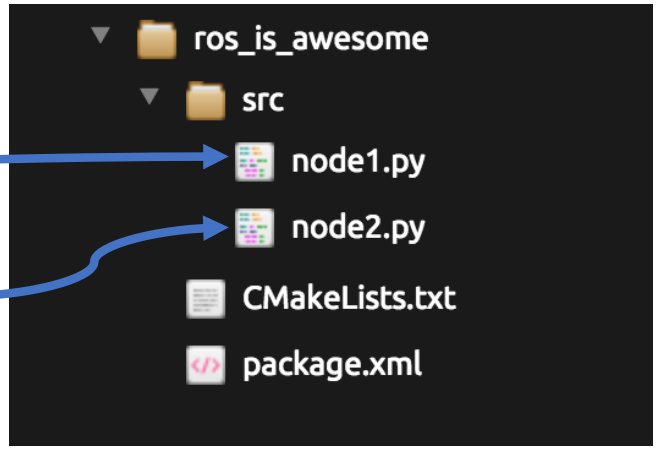


ROS node



Explanation:

A ROS node is process that performs computation within a ROS package

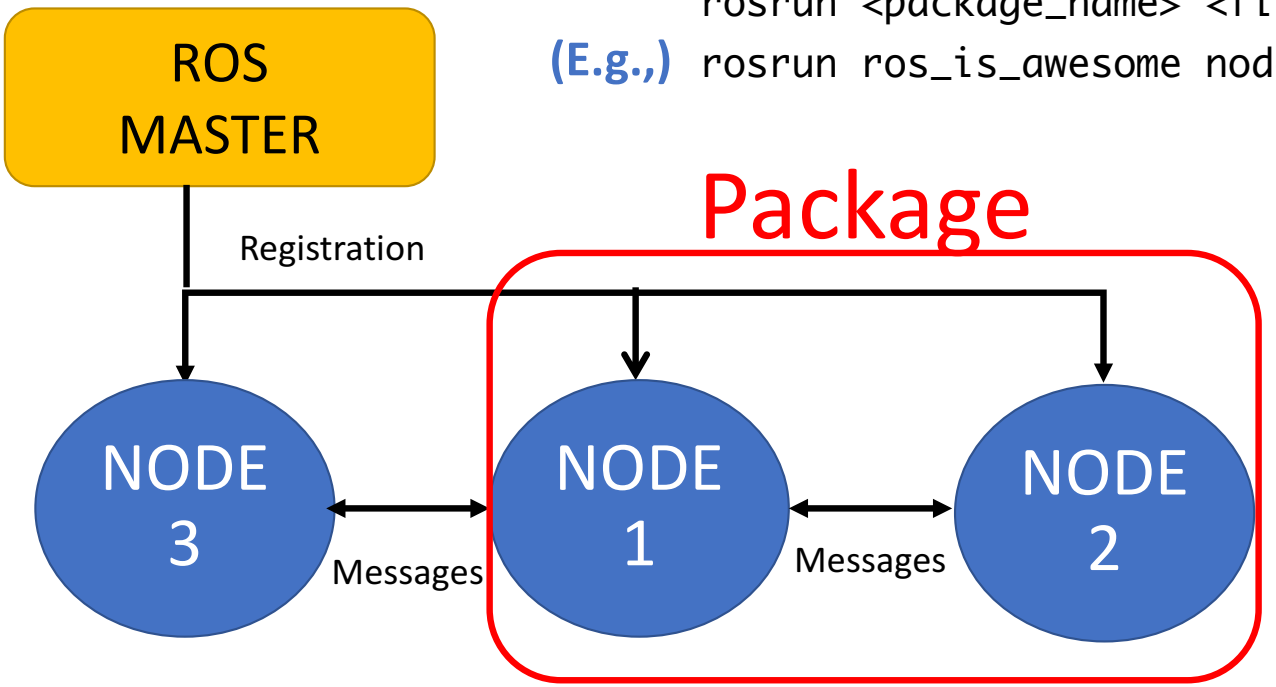


rosrun – How to run a node

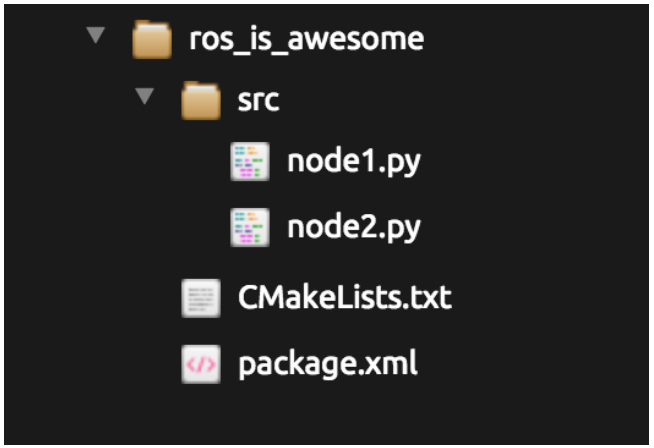
Important Commands:

```
rosrun <package_name> <file_name>
```

(E.g.,) `rosrun ros_is_awesome node1.py`



Package structure:



roslaunch – How to run several nodes at once

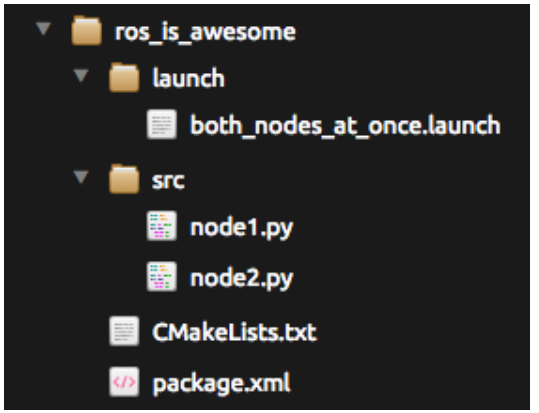
Important Commands:

```
roslaunch <package_name> <launch_file>
```

(E.g.,)

```
roslaunch ros_is_awesome both_nodes_at_once.launch
```

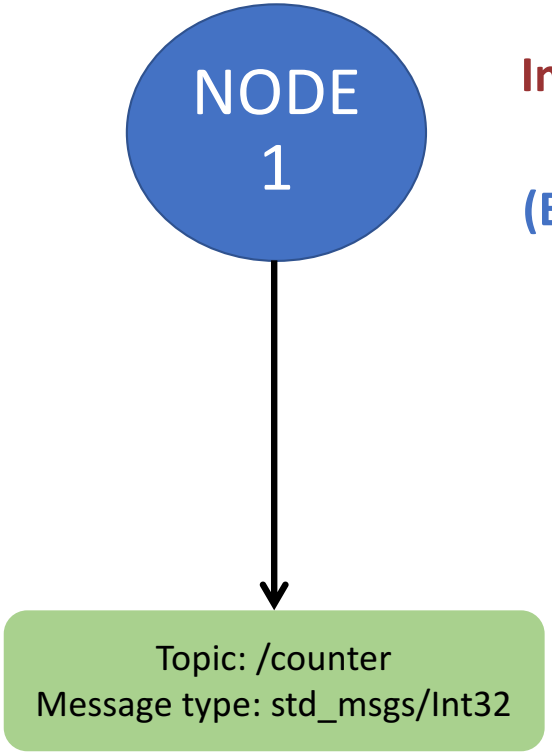
Package structure:



Launch file structure:

```
<launch>
    <node pkg="ros_is_awesome" type="node1.py" name="node1" output="screen" />
    <node pkg="ros_is_awesome" type="node2.py" name="node2" output="screen" />
</launch>
```

Topics – Simple Publisher



Important Commands:

```
rostopic pub –r <Hz> <topic_name> <message_type> <value>
```

(E.g.,)

```
rostopic pub –r 10 /counter std_msgs/Int32 5 -v
```

Result:

```
user:~$ rostopic pub -r 10 -v /counter std_msgs/Int32 5 -v
publishing data: 5
publishing data: 5
publishing data: 5
publishing data: 5
publishing data: 5
publishing data: 5
publishing data: 5
publishing data: 5
publishing data: 5
publishing data: 5
```

Topic - Simple Publisher

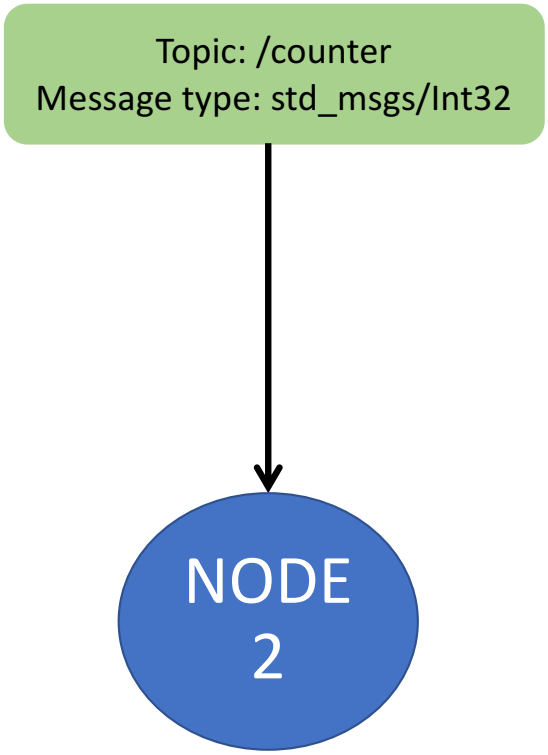
```
#! /usr/bin/env python

import rospy
from std_msgs.msg import Int32

rospy.init_node('simple_publisher')
pub = rospy.Publisher('/counter', Int32, queue_size=1)
rate = rospy.Rate(2)
count = Int32()
count.data = 0


while not rospy.is_shutdown():
  pub.publish(count)
  count.data += 1
  rate.sleep()
```

Obtain the ROS python handler
Load the required message type

Initialize the ROS node
Define the publisher topic
Create a rate object
Create the variable to send
Initialize the variable

Publish loop

Topics – Simple Subscriber



Important Commands:

```
rostopic echo <topic_name>
(E.g.,) rostopic echo /counter
```

Result:

```
user:~$ rostopic echo /counter
data: 5
---
data: 5
---
data: 5
---
data: 5
---
data: 5
---
```

Topic - Simple Subscriber

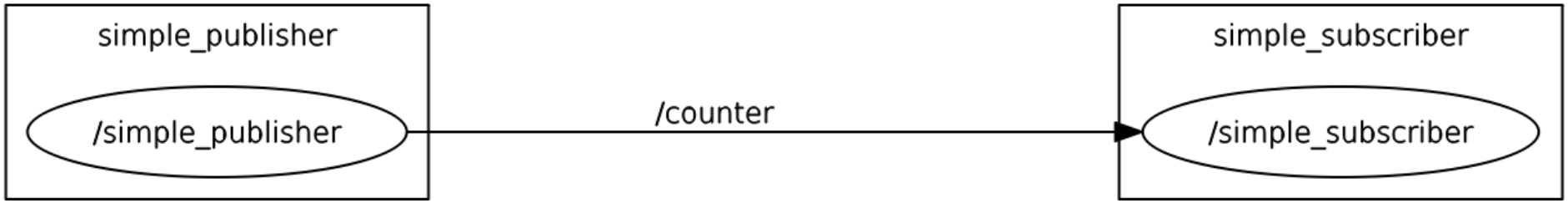
```
#! /usr/bin/env python

import rospy
from std_msgs.msg import Int32


def callback(msg):
  print msg.data


rospy.init_node('simple_subscriber')
sub = rospy.Subscriber('counter', Int32,
callback()
rospy.spin()
```

Obtain the ROS python handler
Load the required message type

Callback function – This is executed every time a message is received in the topic

Initialize the ROS node
Select the topic to subscribe to
Call the Callback function
Keep the code running

Publisher – Subscriber Graph



Topics and subscribers in the system

```
>> rostopic list
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/clock
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/teleop
/cmd_vel_mux/parameter_descriptions
/cmd_vel_mux/parameter_updates
/depthimage_to_laserscan/parameter_descriptions
/depthimage_to_laserscan/parameter_updates
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
```

Important Commands:

`rostopic list` — lists all active topics in the system

`rostopic info <topic_name>` — displays information about a specific topic

(E.g.,) `rostopic info /cmd_vel`

Result:

```
user ~ $ rostopic info /cmd_vel
Type: geometry_msgs/Twist

Publishers: None

Subscribers:
 * /gazebo (http://ip-172-31-20-130:50851/)


user ~ $
```

Messages

```
>> rosmsg show humanoid_nav_msgs/ExecFootstepsAction
humanoid_nav_msgs/ExecFootstepsActionGoal action_goal
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  actionlib_msgs/GoalID goal_id
    time stamp
    string id
  humanoid_nav_msgs/ExecFootstepsGoal goal
    humanoid_nav_msgs/StepTarget[] footsteps
      uint8 right=0
      uint8 left=1
      geometry_msgs/Pose2D pose
        float64 x
        float64 y
        float64 theta
      uint8 leg
    float64 feedback_frequency
  humanoid_nav_msgs/ExecFootstepsActionResult action_result
    std_msgs/Header header
      uint32 seq
       time stamp
       string frame_id
  actionlib_msgs/GoalStatus status
    uint8 PENDING=0
```

Important Commands:

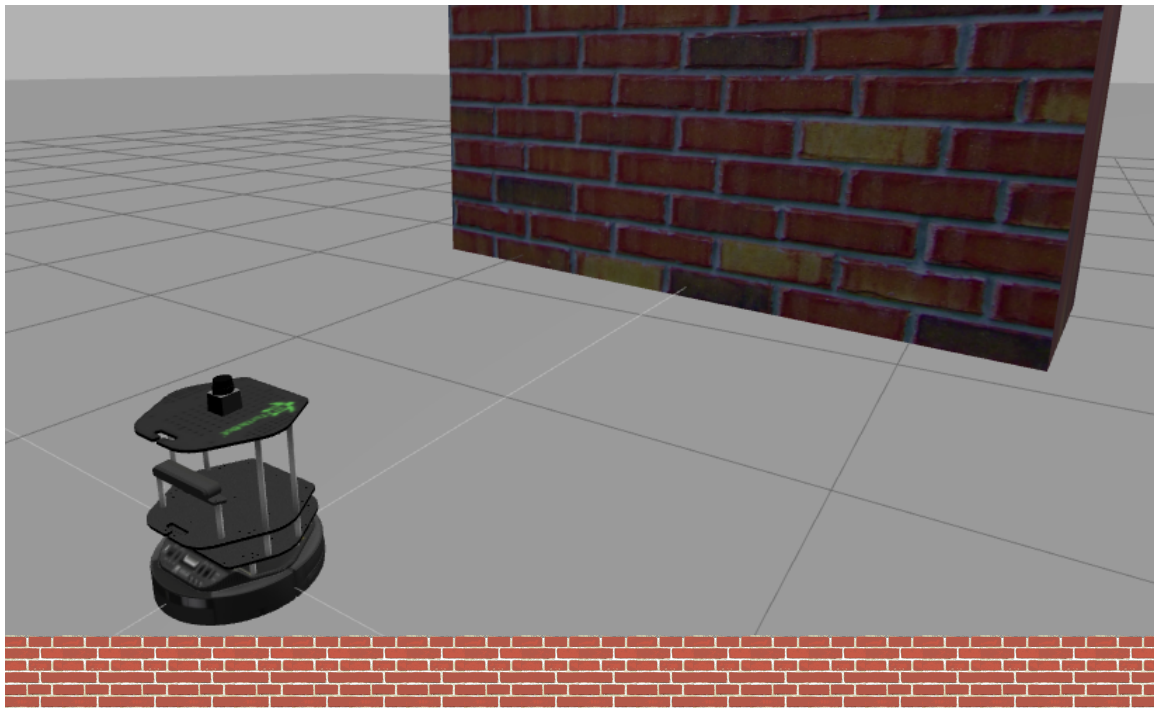
`rosmsg show <message_type>` Show message description

(E.g.,) `rosmsg show geometry_msgs/Twist`

Result:

```
user ~ $ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Exercise 1 – Avoid the wall



Info:

The robot accepts velocity commands at:

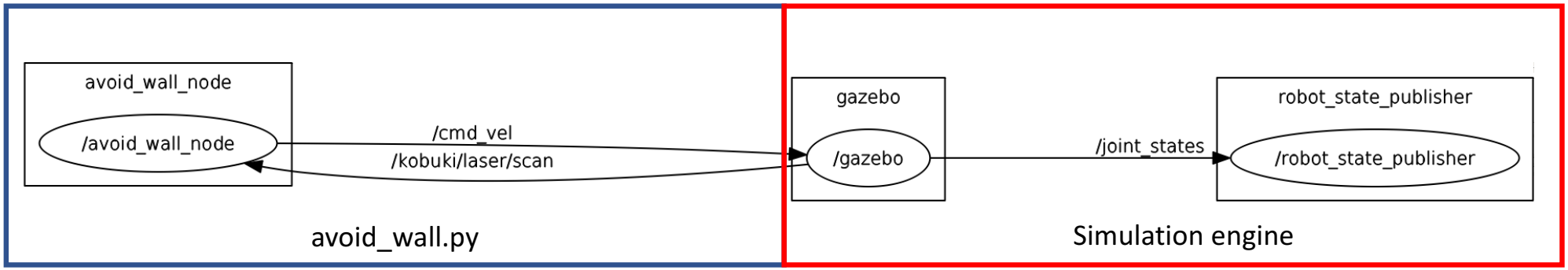
- ***/cmd_vel*** ***(geometry_msgs/Twist)***
  - **linear**
    - **x**
    - **y = 0**
    - **z = 0**
  - **angular**
    - **x = 0**
    - **y = 0**
    - **z**

The robot is equipped with a laser scan, which outputs information at:

- ***/kobuki/laser/scan*** ***(sensor_msgs/LaserScan)***

ranges [720]

Exercise 1 - Graph

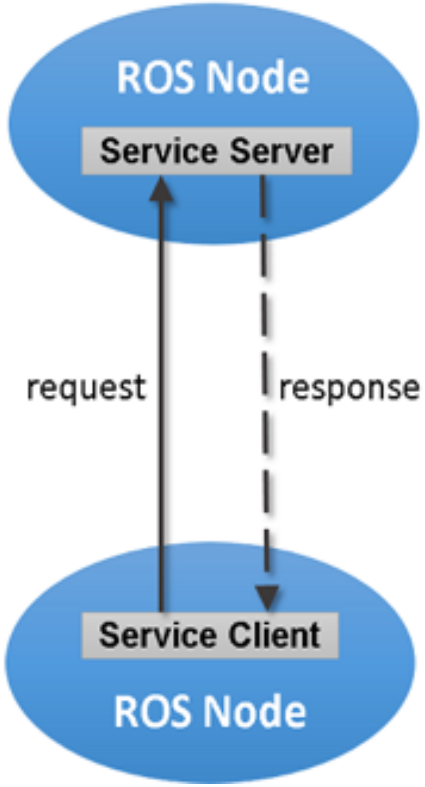


Avoid wall - code

```
#! /usr/bin/env python
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist

rospy.init_node('avoid_wall_node')
sub = rospy.Subscriber('/kobuki/laser/scan', LaserScan, callback)
pub = rospy.Publisher('/cmd_vel', Twist)
move = Twist()

rospy.spin()
```

```
def callback(msg):
  print msg.ranges[360] #print front

#move forward
  if msg.ranges[360] > 1:
      move.linear.x = 0.1
      move.angular.z = 0.0

#obstacle ahead, turn left
  if msg.ranges[360] < 1:
      move.linear.x = 0.0
      move.angular.z = 0.2

#turn right
  if msg.ranges[719] < 0.3:
      move.linear.x = 0.0
      move.angular.z = -0.2

#turn left
  if msg.ranges[0] < 0.3:
      move.linear.x = 0.0
      move.angular.z = 0.2

  pub.publish(move)
```

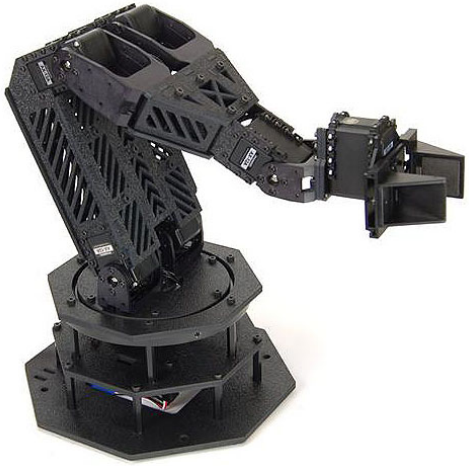
Services

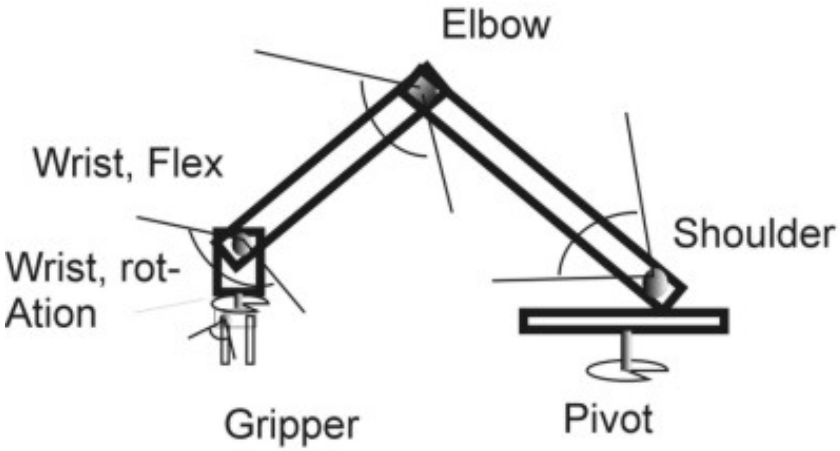


Service Name: /move_towards_goal

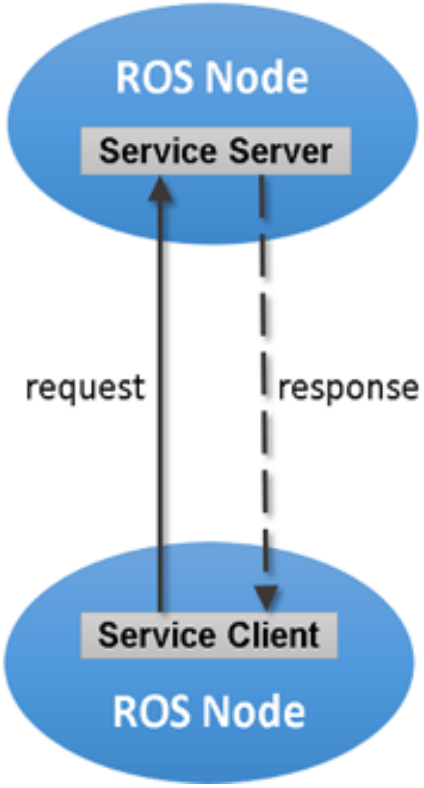
Request Type: nav_msgs/pose
Response Type: std_msgs/bool

ROS Services are synchronous





Services



Service Name: /move_towards_goal

Request Type: nav_msgs/pose
Response Type: std_msgs/bool

ROS Services are synchronous

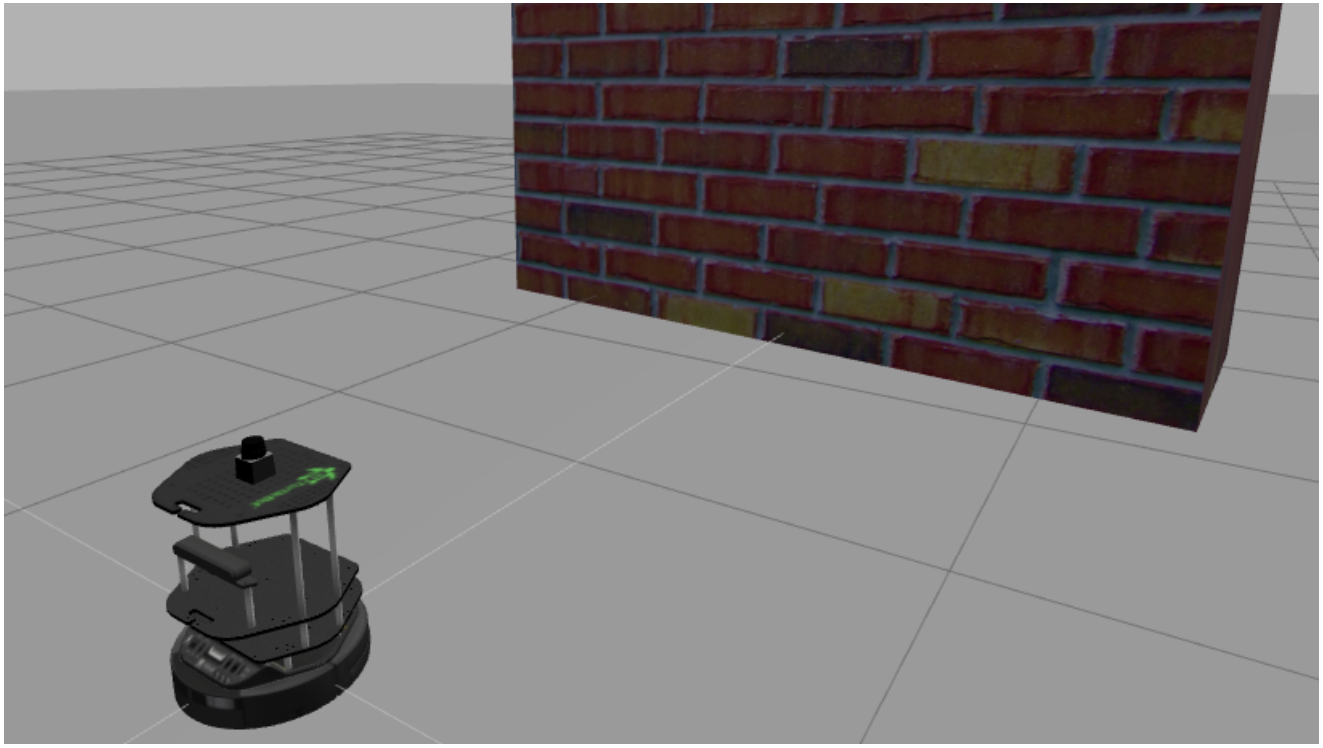
Important Commands:

`rosservice list` lists all active services in the system

`rosservice info <service_name>` displays information about a specific service

`rosservice call <service_name> <request>`

Exercise 2 – Move along a squared path



Info:

1. We create a class with the robot's functionality

- **move_robot.py**

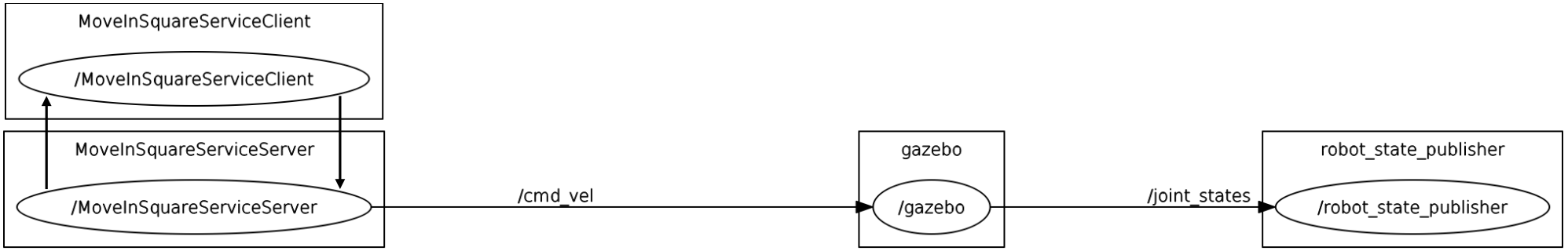
2. We create a service server to execute the task on demand

- **service_server.py**

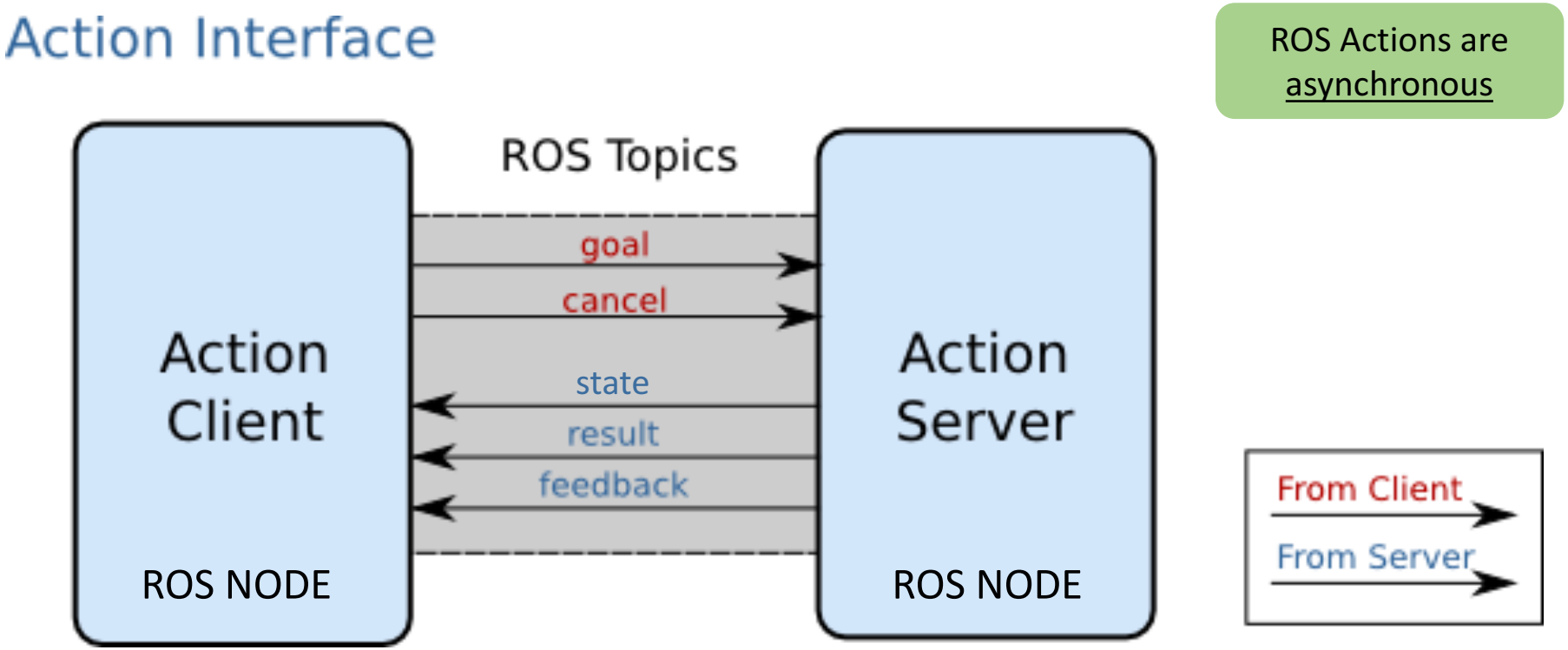
3. We launch a service client to send the request and obtain the response

- **service_client.py**

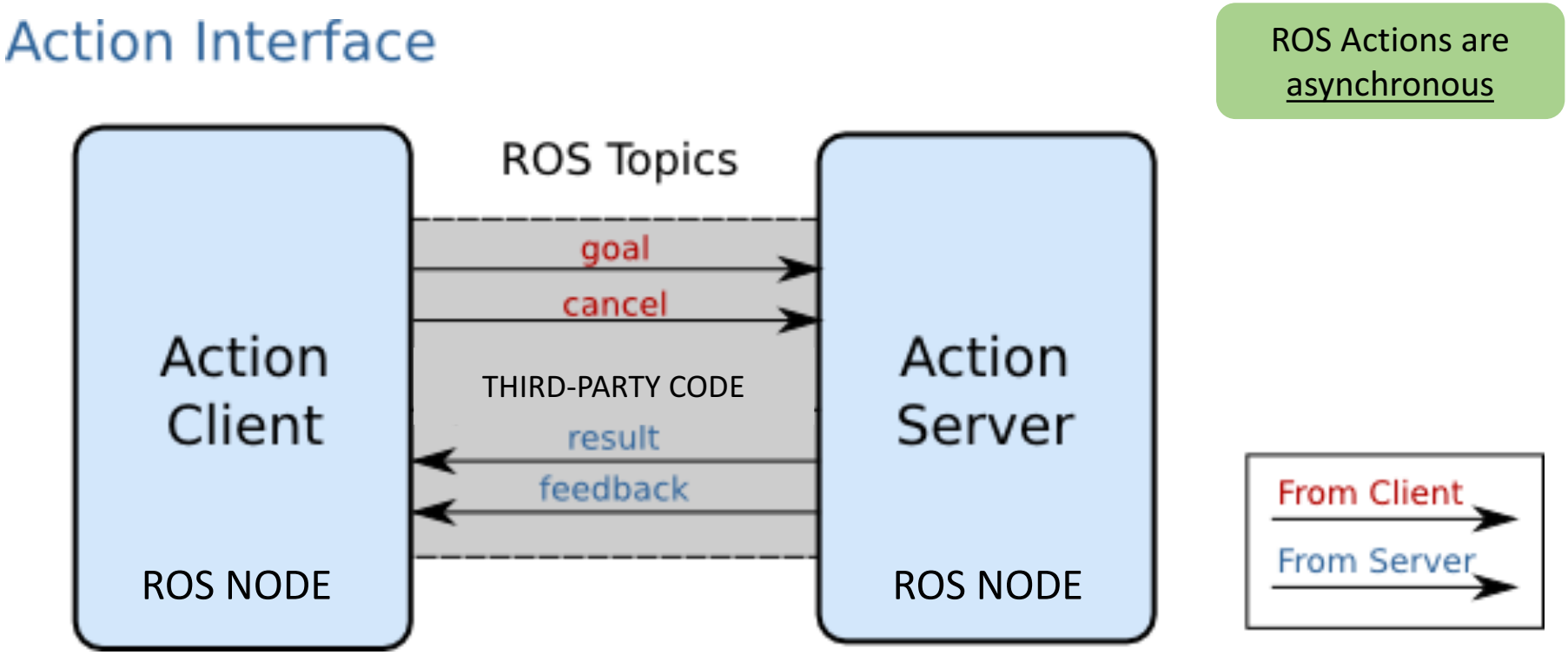
Move along a squared path - code



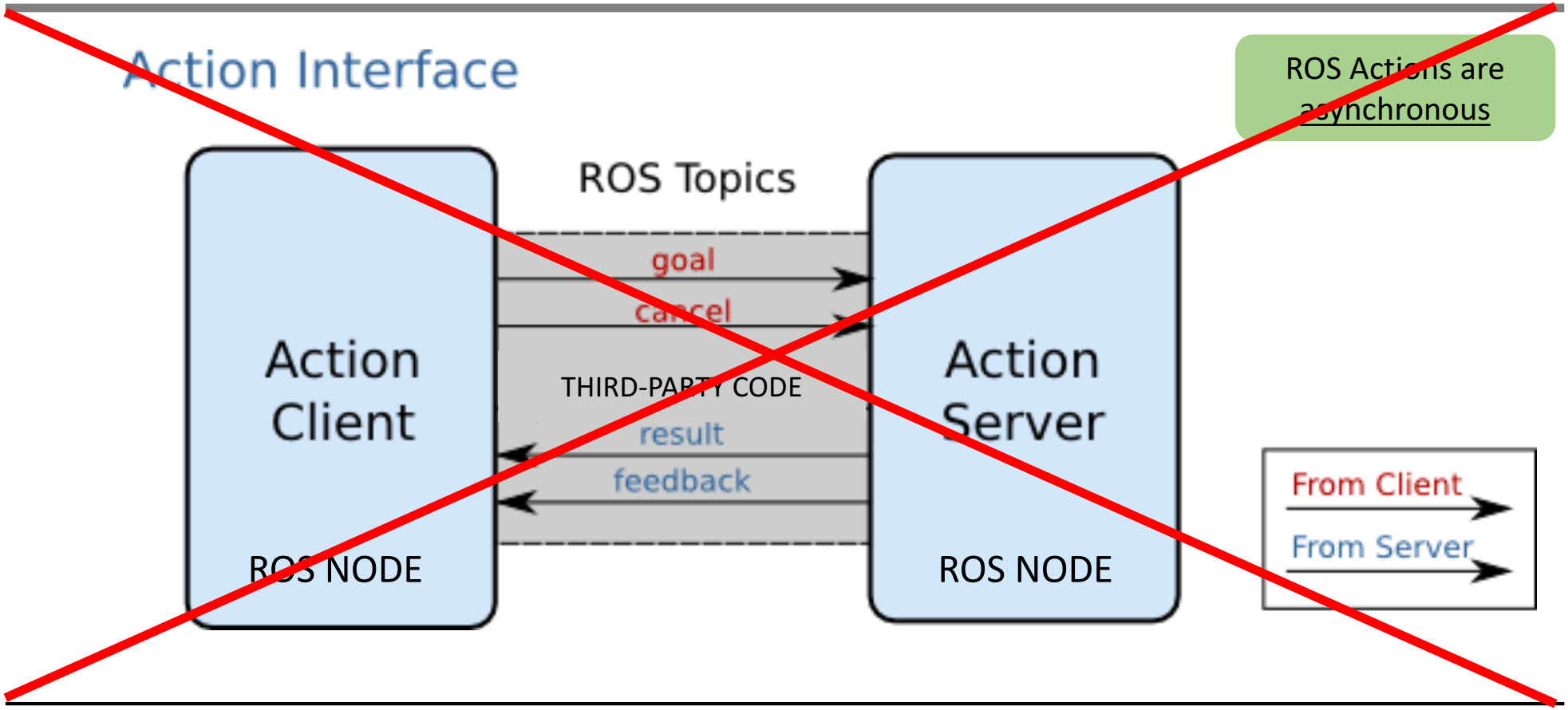
Actions


Action Interface
ROS Actions are asynchronous
ROS Topics
goal
cancel
state
result
feedback
Action Client
ROS NODE
Action Server
ROS NODE
From Client
From Server

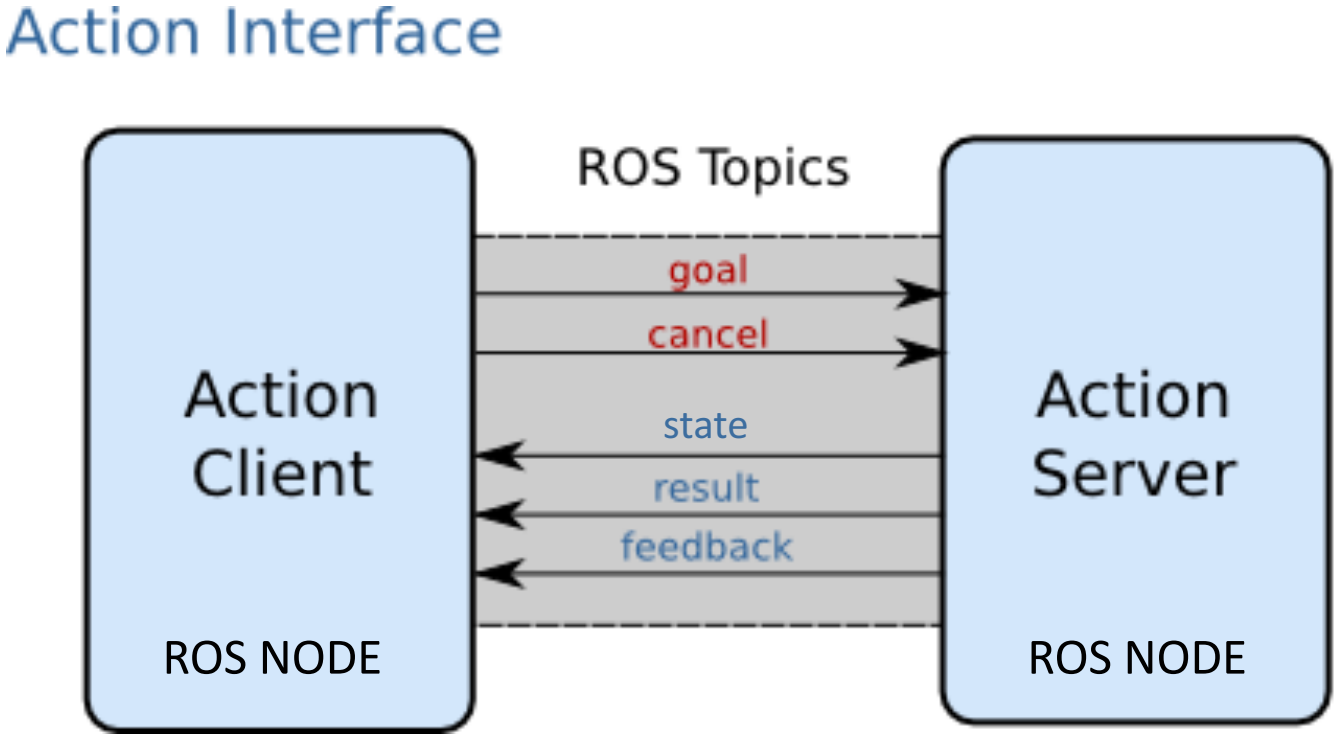
Actions – wait for result



Actions – wait for result



Actions – not wait for result (state comparison)

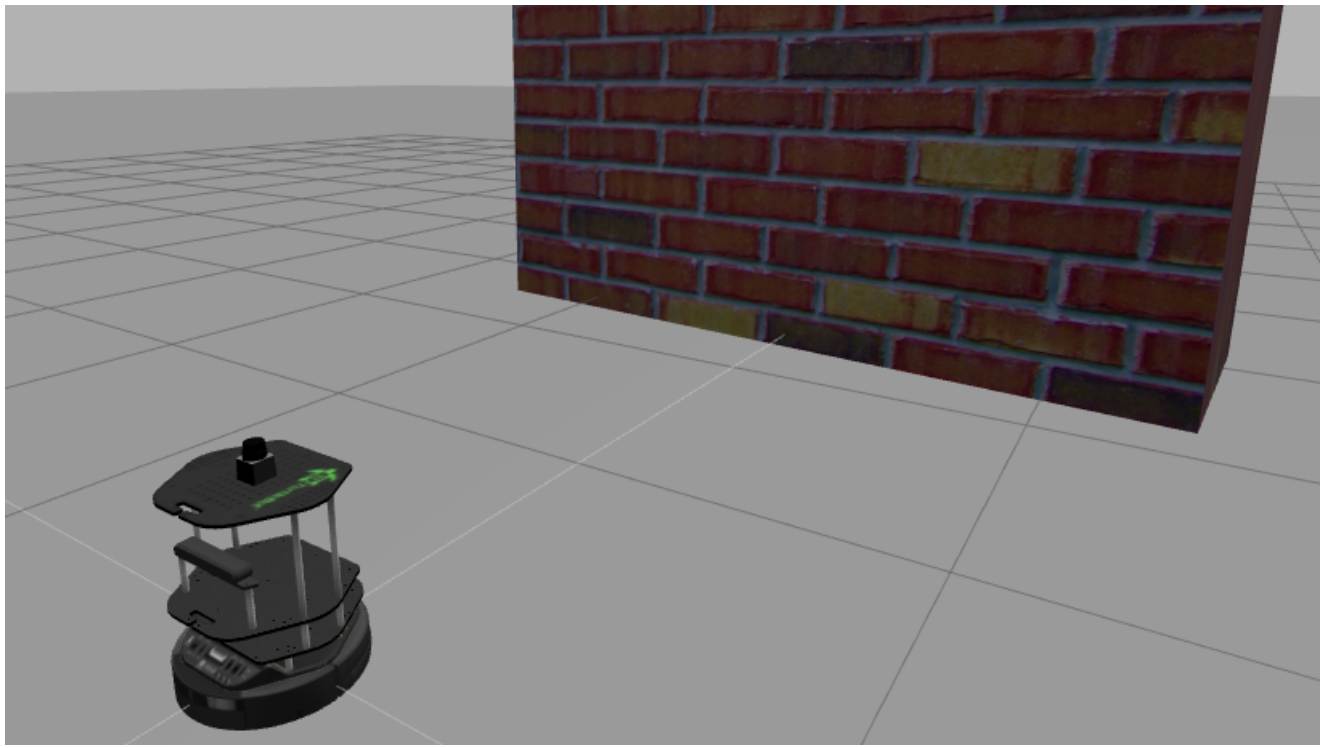


ROS Actions are asynchronous

State Code:

0 ==> PENDING
1 ==> ACTIVE
2 ==> DONE
3 ==> WARN
4 ==> ERROR

Exercise 3 – Take pictures along the way



Info:

1. We define an action message

- goal
- result
- feedback

2. We create an action server

- **action_server.py**

3. We create an action client to execute the task on demand

- **action_client.py**

Debugging tools - roswtf

By default, it checks **two** ROS fields:

- **File-system issues**
- **Online/graph issues**

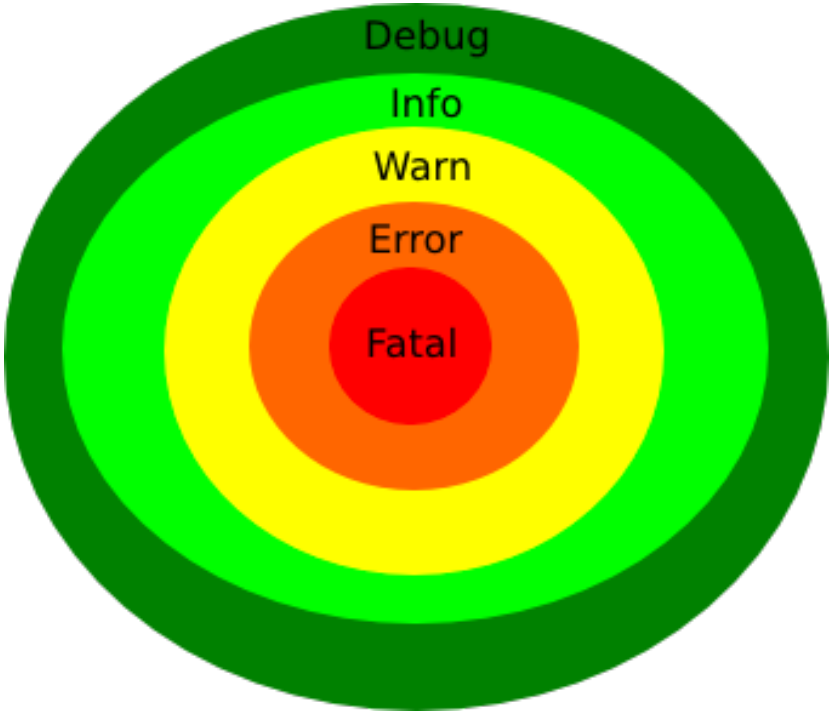
Result:

```
user:~$ roswtf
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep update'
No package or stack in context
================================================================================
Static checks summary:

Found 1 warning(s).
Warnings are things that may be just fine, but are sometimes at fault

WARNING You have pip installed packages on Ubuntu, remove and install using Debian packages: catkin_pkg --

Found 1 error(s).

ERROR ROS Dep database not initialized: Please initialize rosdep database with sudo rosdep init.
================================================================================
Beginning tests of your ROS graph. These may take awhile...
analyzing graph...
... done analyzing graph
running graph rules...
... done running graph rules

Online checks summary:

Found 2 warning(s).
Warnings are things that may be just fine, but are sometimes at fault

WARNING The following node subscriptions are unconnected:
 * /mobile_base_nodelet_manager:
   * /cmd_vel_mux/input/switch
   * /cmd_vel_mux/input/safety_controller
   * /cmd_vel_mux/input/navi
   * /cmd_vel_mux/input/teleop
```

Important Commands:

`roswtf`

Debugging tools – debug levels



Programming function Calls:

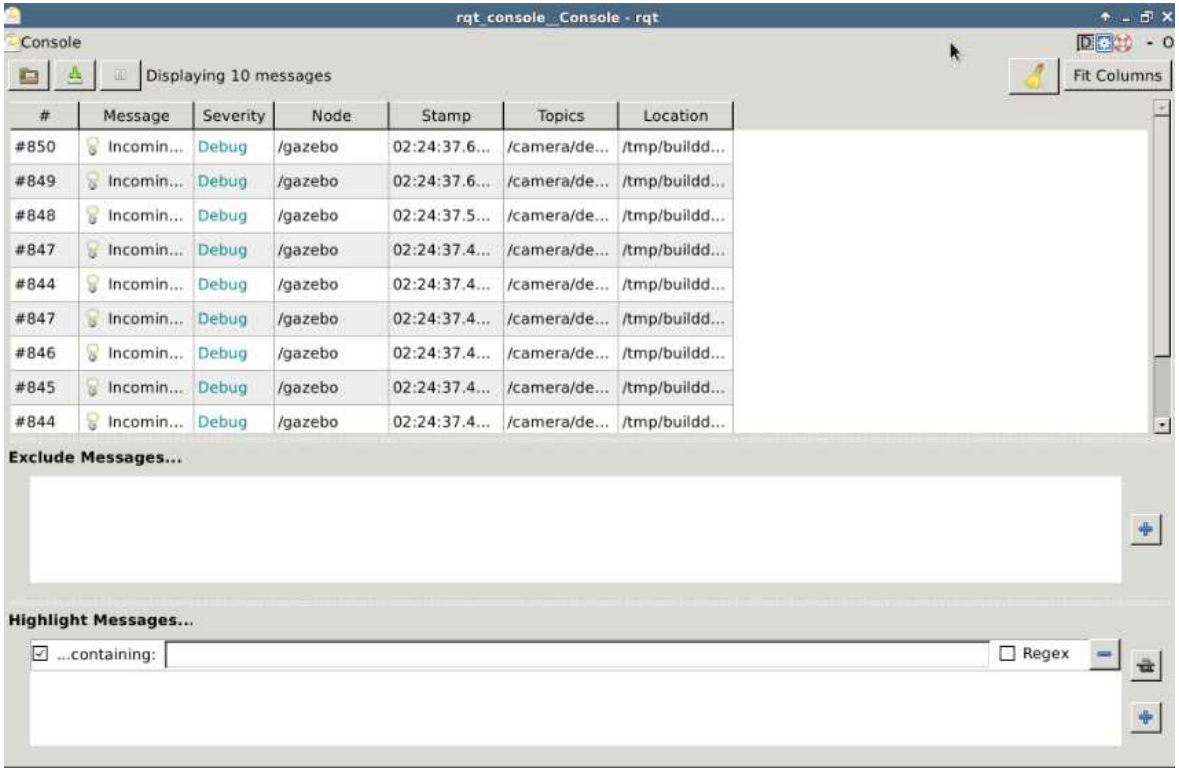
DEBUG -> `rospy.logdebug(msg, args)`
INFO -> `rospy.loginfo(msg, args)`
WARNING -> `rospy.logwarn(msg, args)`
ERROR -> `rospy.logerr(msg, args)`
FATAL -> `rospy.logfatal(msg, *args)`

Node initialization:

```
rospy.init_node('log_demo', log_level=rospy.DEBUG)
```

Note: **The best place to read all of the logs issued by all of the ROS Systems is: /rosout**

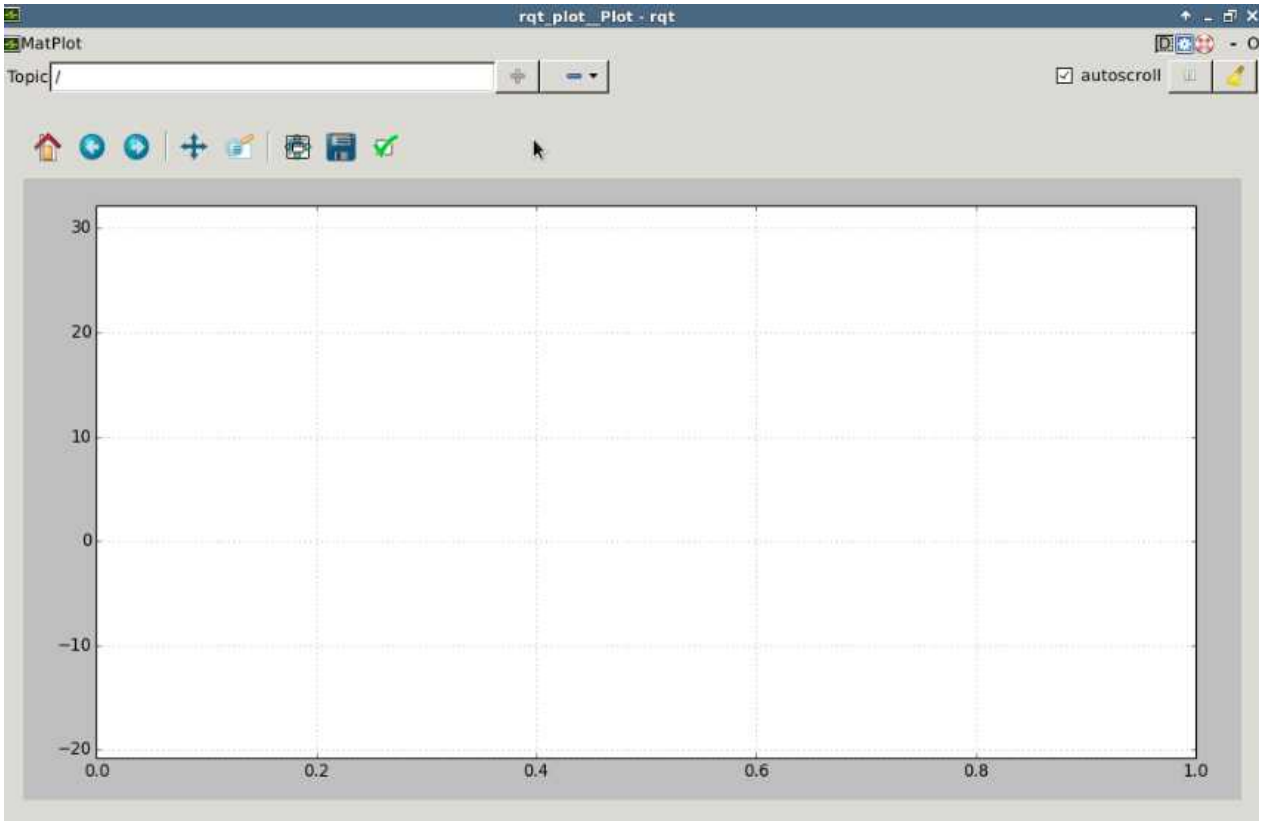
Debugging tools – rqt_console



Important Features:

- Filter
- Query
- Create alerts
- Data forensics

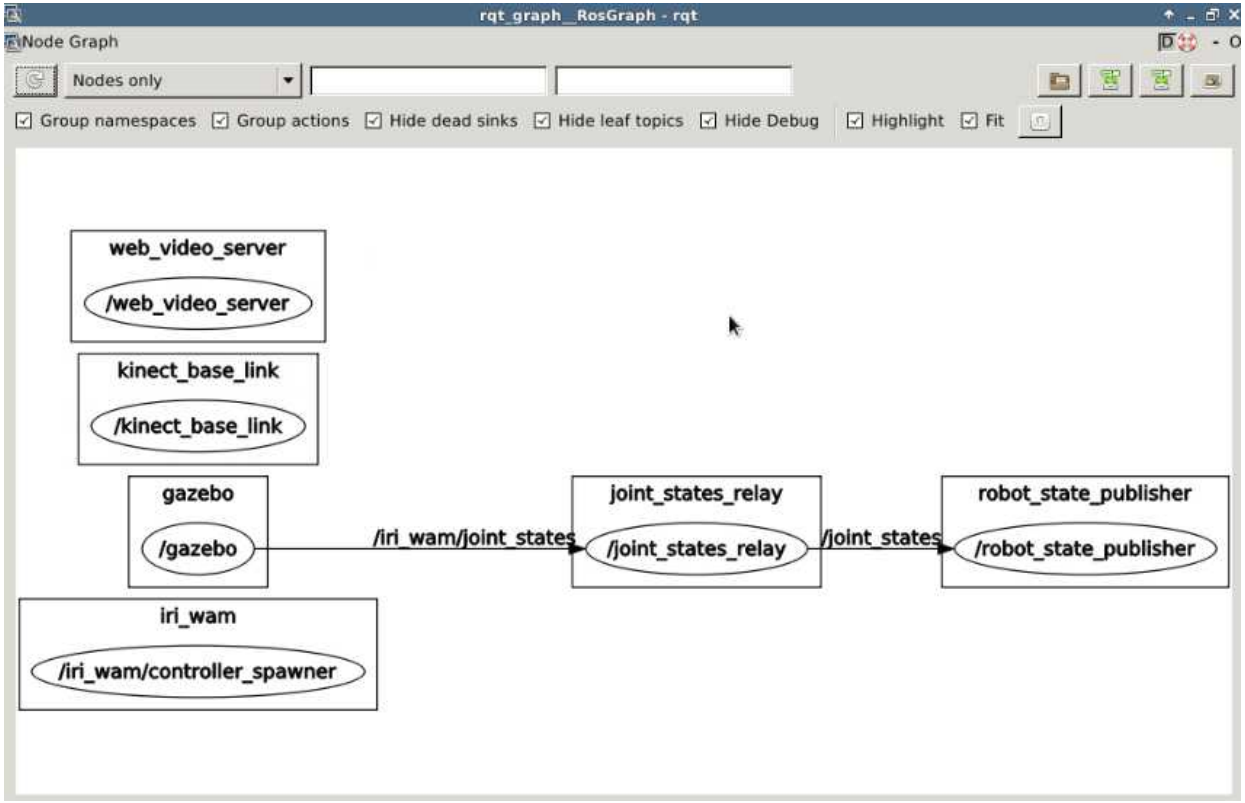
Debugging tools – rqt_plot



Important Features:

- Simple data viz.
- Multiple lines
- Oscilloscope
- Data forensics

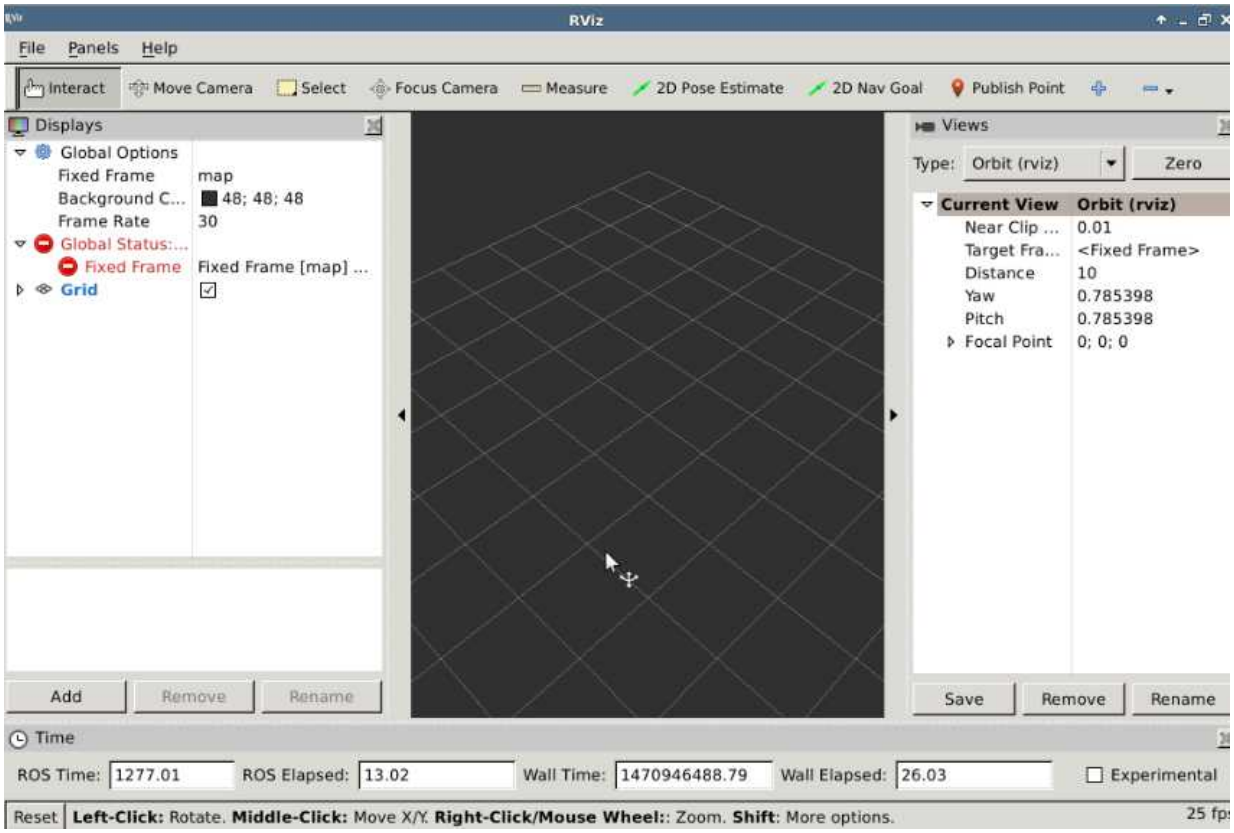
Debugging tools – rqt_graph



Important Features:

- Filter
- Query
- Check connections
- Data forensics

Debugging tools – rviz



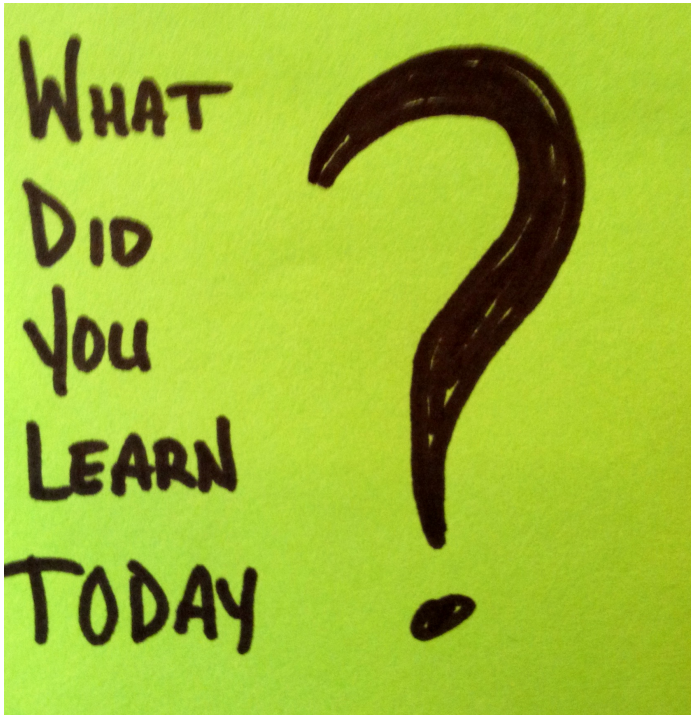
Important Features:

- Data Viz.
- Layer system

Important Notes:

- Rviz is not a simulation engine
- Rviz is a graphical representation of what is going on inside ROS

Let's wrap up …



Conclusions

- ROS is awesome!
- ROS improves code reusability
- ROS provides several communication paradigms:
  - Master nodes + client nodes (a.k.a. distributed system)
  - Message passing with ROS Topics
  - Synchronous functionalities with ROS Services
  - Asynchronous functionalities with ROS Actions
- ROS provides a ton of debugging tools (e.g., gazebo, rviz, etc.)

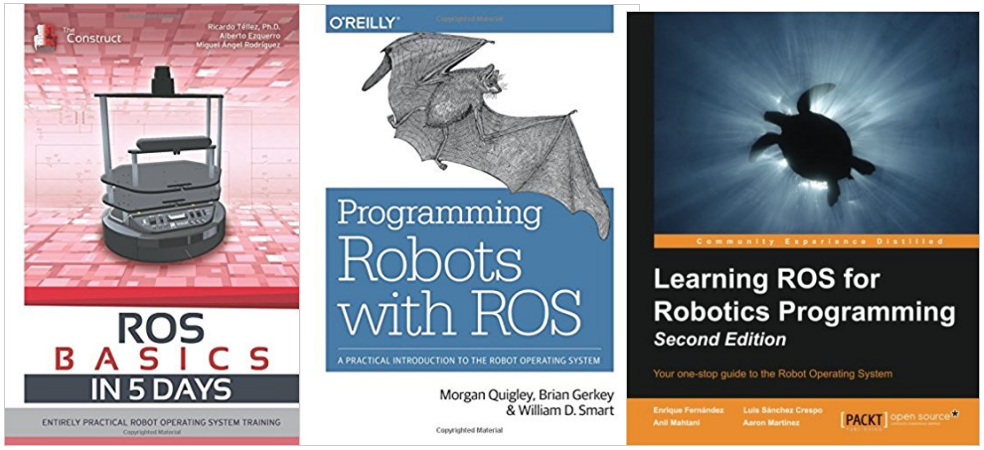
References

Online Material:

- Wizards of ROS -> IEEE Spectrum Article (History of ROS)
- ROS Wiki -> http://wiki.ros.org
- The Construct Sim -> http://www.theconstructsim.com/ (Online ROS Environment)
- Erle Robotics Tutorials -> https://www.youtube.com/playlist?list=PLH-vcjgGSri0sipCweLukjPvJuoUpGchJ
- ROS: A gentle introduction -> https://cse.sc.edu/~jokane/agitr/agitr-letter.pdf

Books:

- **ROS Basics in 5 days**
- **Programming Robots with ROS**
- **Learning ROS for Robotics Programming**




ANY
QUESTIONS?