ELEG4701

Intelligent Interactive Robot Practice

Lab 4: The Topic in ROS

Jiewen Lai

Research Assistant Professor

EE, CUHK

jiewen.lai@cuhk.edu.hk

Today's Agenda

Lecture

1. What is Node, Topic, and Message
2. Topic Tools
3. How to write a Publisher
4. How to write a Subscriber
5. How to create a ROS msg

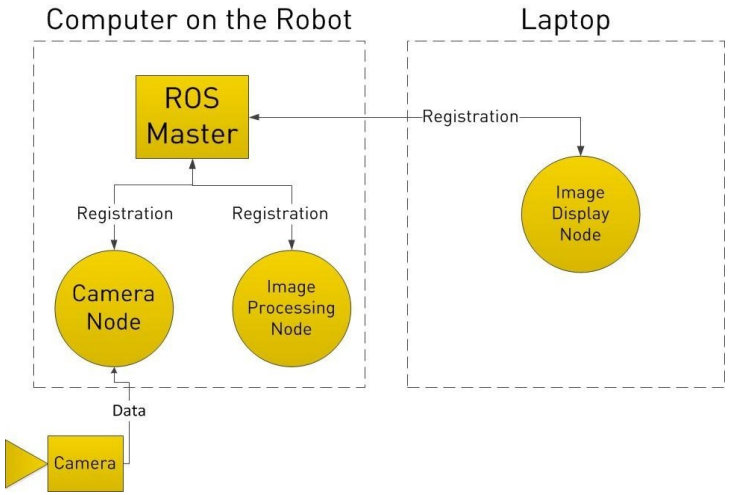
Tutorial

1. Lab Sheet 4

What is Node, Topic, Message

Core concepts in ROS (Recap)

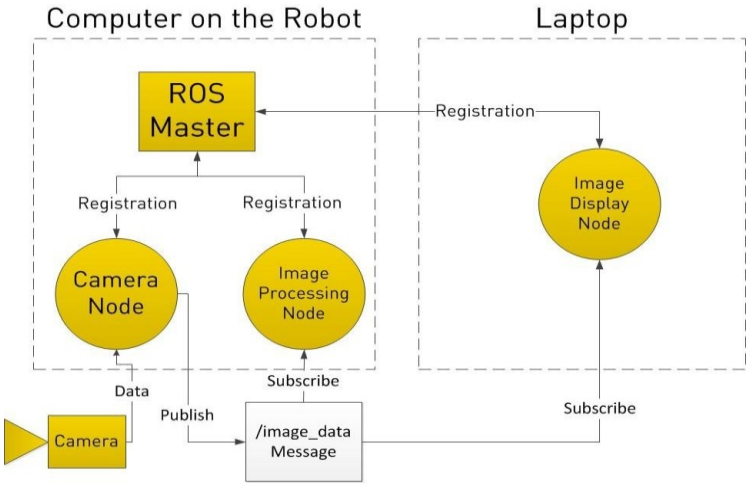
- **Node** – Execution Unit
  - Processes that perform specific tasks, independently run executables
  - Different nodes can use **different programming languages** and can be distributed to run on different hosts
  - The name of the node must be unique in the system



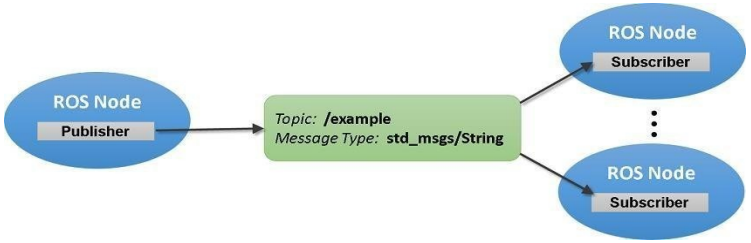
- **ROS Master** – Control center
  - Provide naming and registration services for nodes
  - Track and record topic/service communications to assist nodes in finding each other and establishing connections
  - Provides a parameter server that nodes use to store and retrieve runtime parameters

Core concepts in ROS (Recap)

- **Topic** – *Asynchronous* communication
  - Important bus used to transfer data between nodes
  - Using the **publish/subscribe model**, data is transferred from publisher to subscriber, and publishers or subscribers of the same topic may not be unique

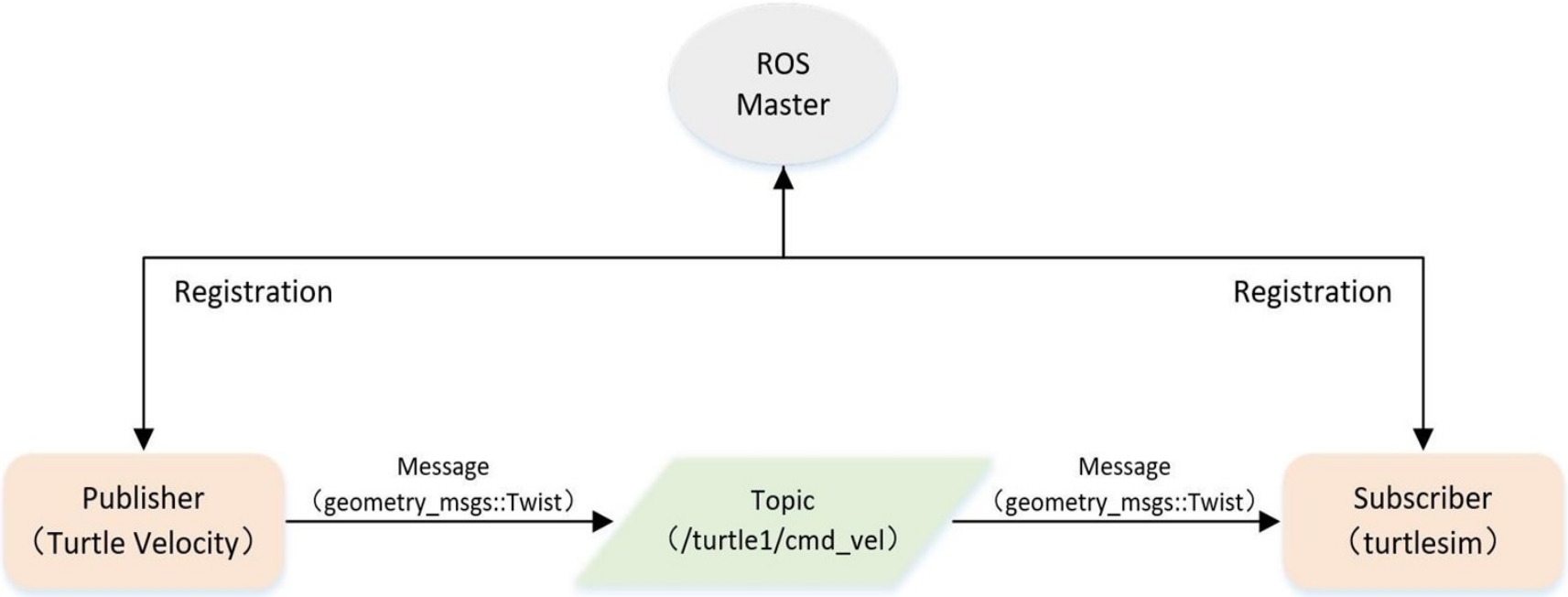


- **Message** – Topic data
  - Has certain types and data structures, including the standard types provided by ROS and user-defined types
  - Use **programming language-independent** `.msg` file define message, the programming process generates the corresponding code files



Topic Model (publish/subscribe)

Topic Model (Recap)



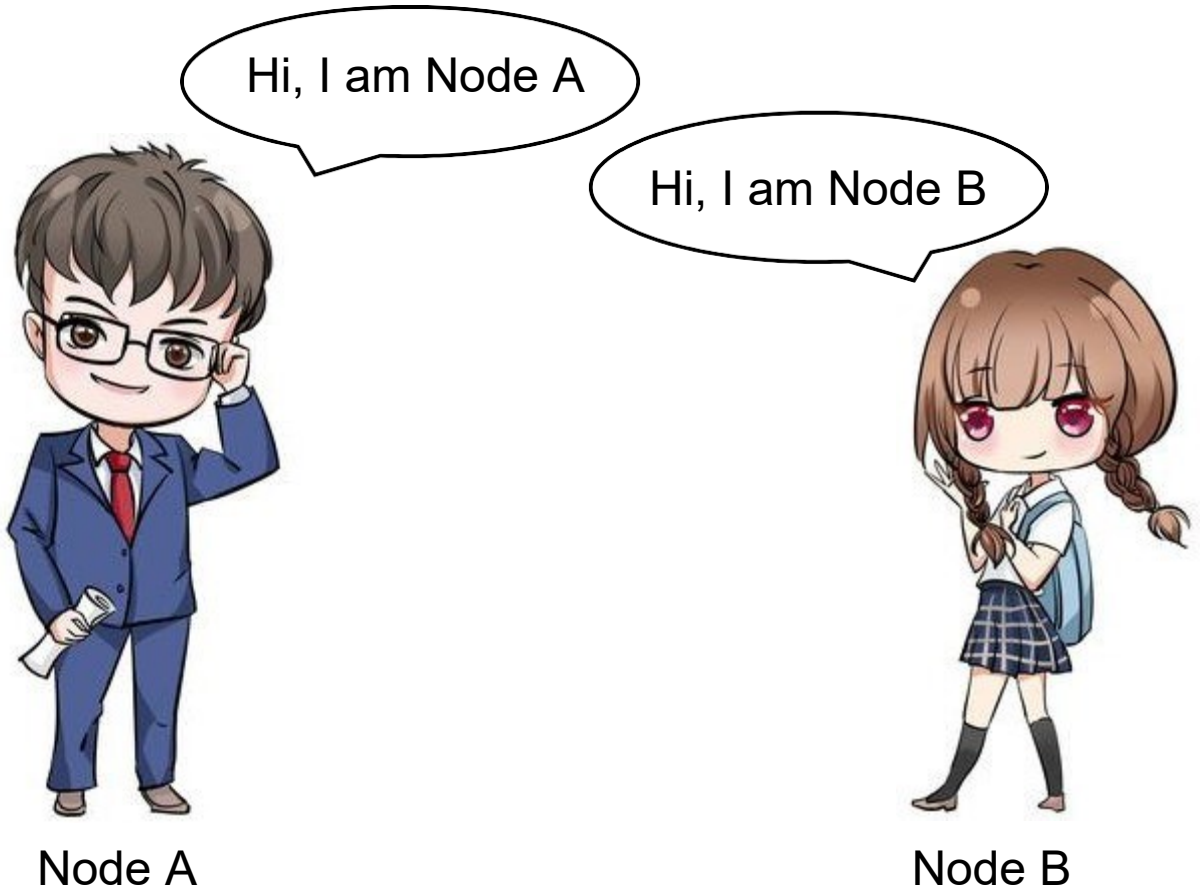
Topic Model (publish/subscribe)

Example

A vivid example to understand the communication concept in ROS:

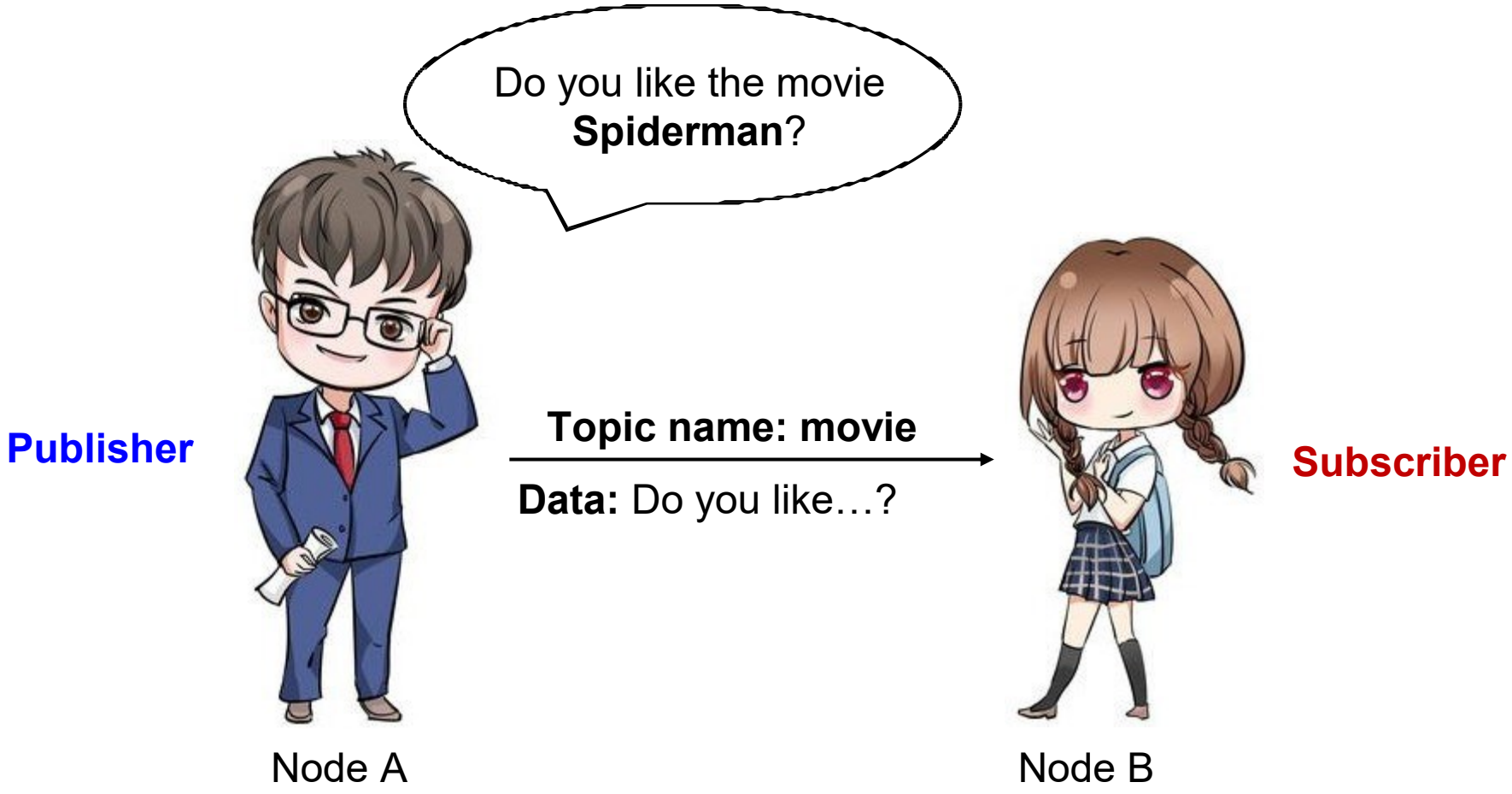
Example

First, each one needs to have a name – otherwise, you don't know who you are or who you are chatting with.



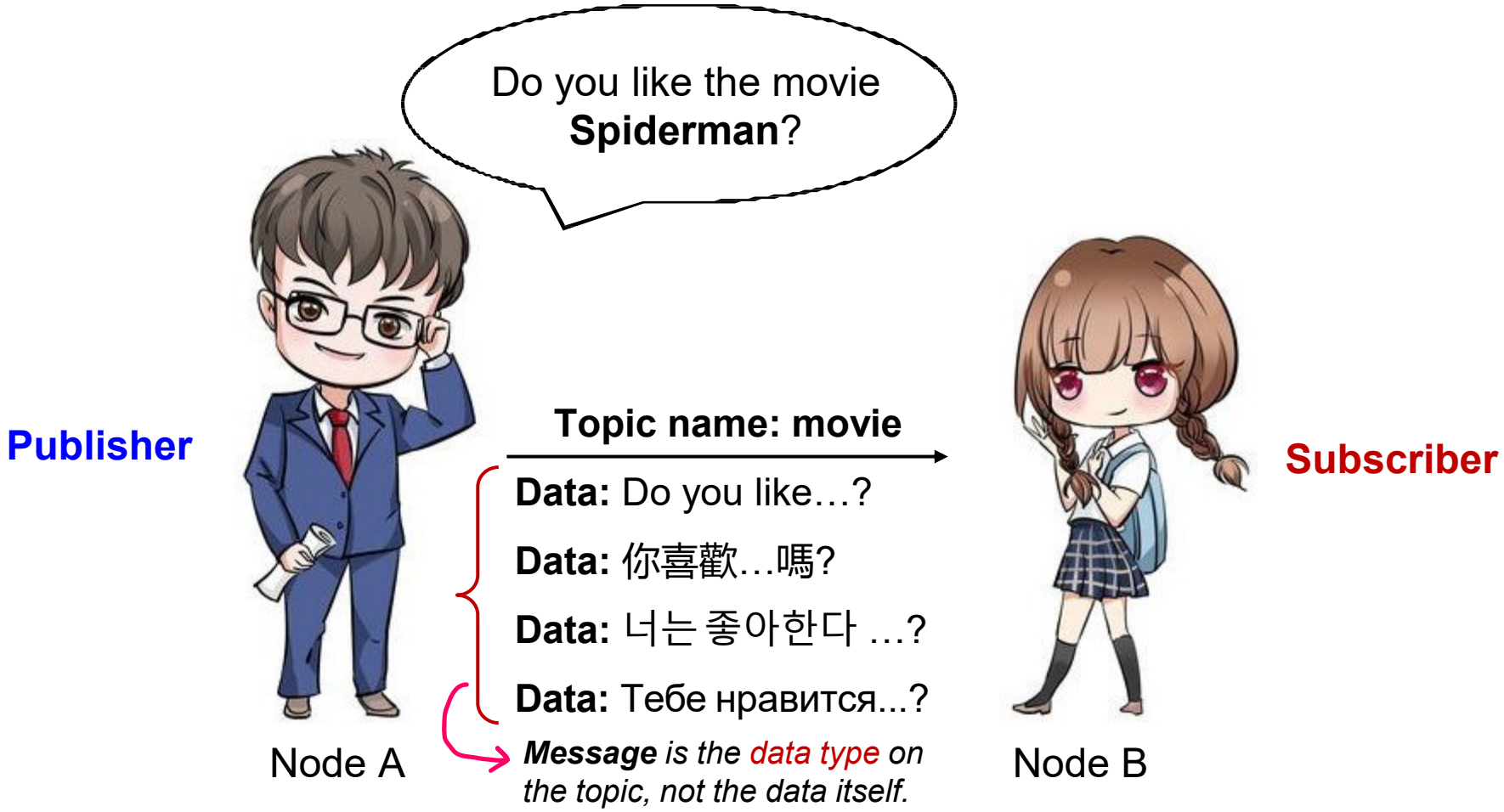
Example

Then, you need **to find a topic** to chat with her. If she listens to your topic, she will receive your data.



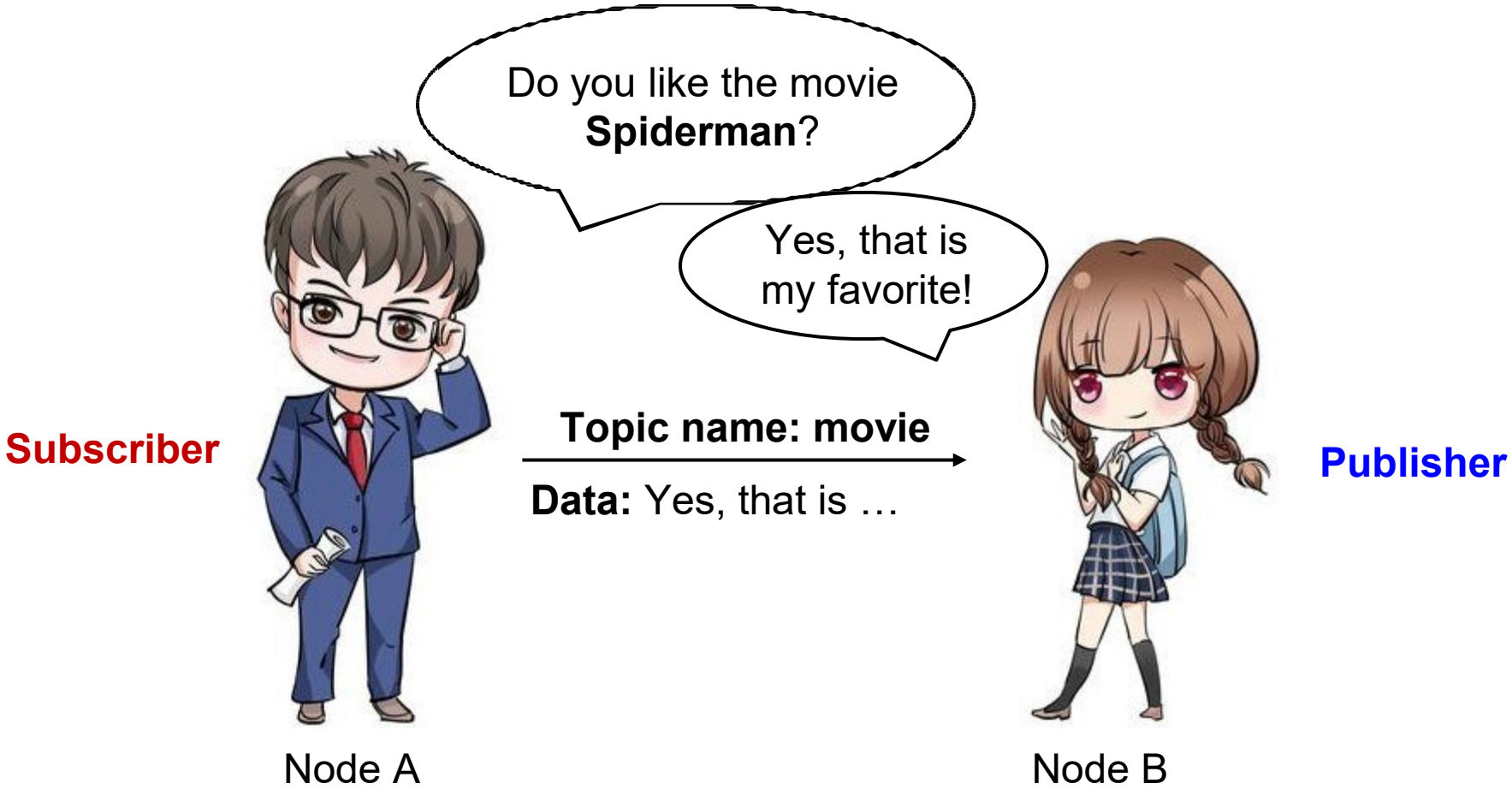
Example

Then, you need **to find a topic** to chat with her. If she listens to your topic, she will receive your data.



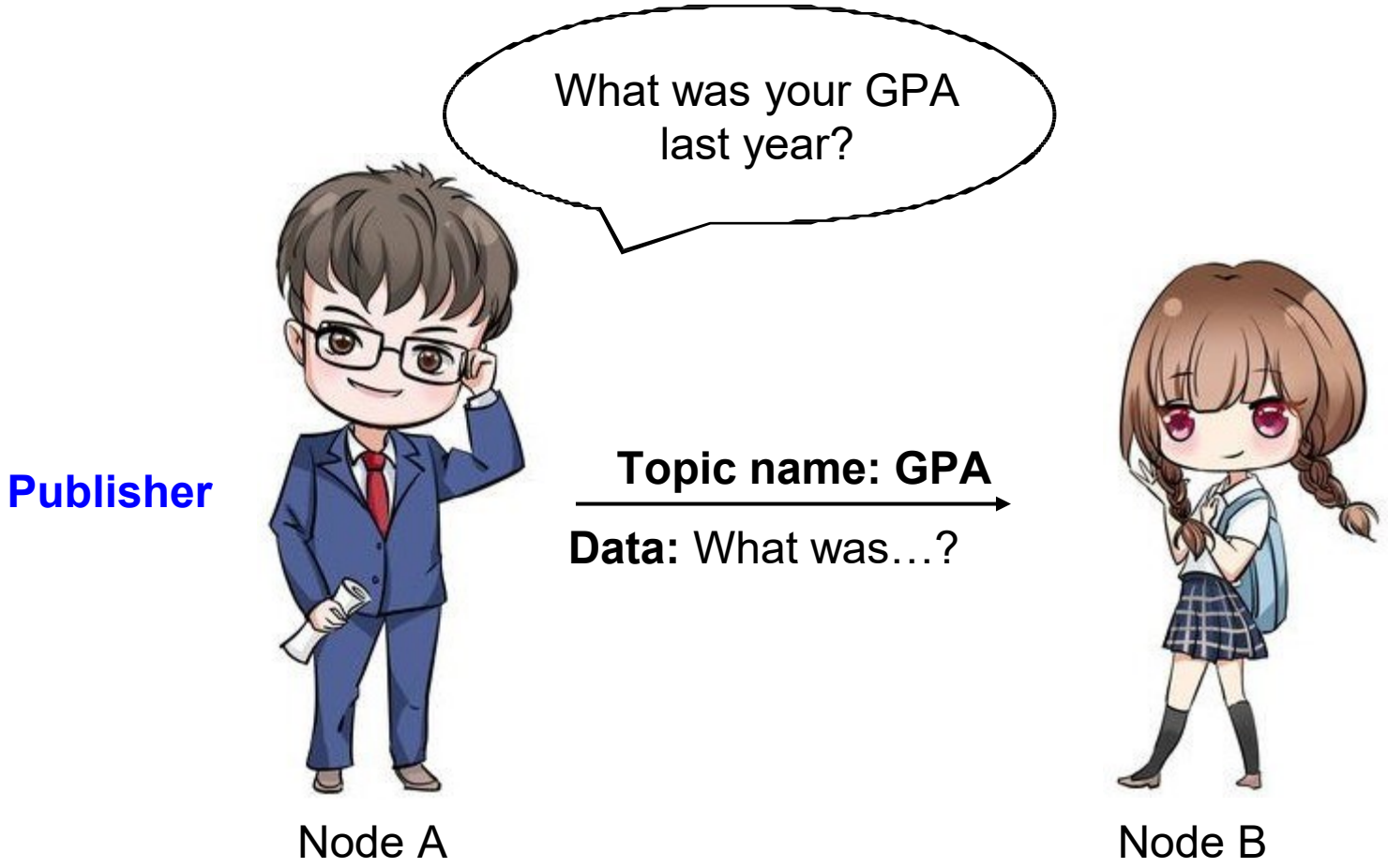
Example

She listened to this **topic,** received the data, and published a new data on this topic



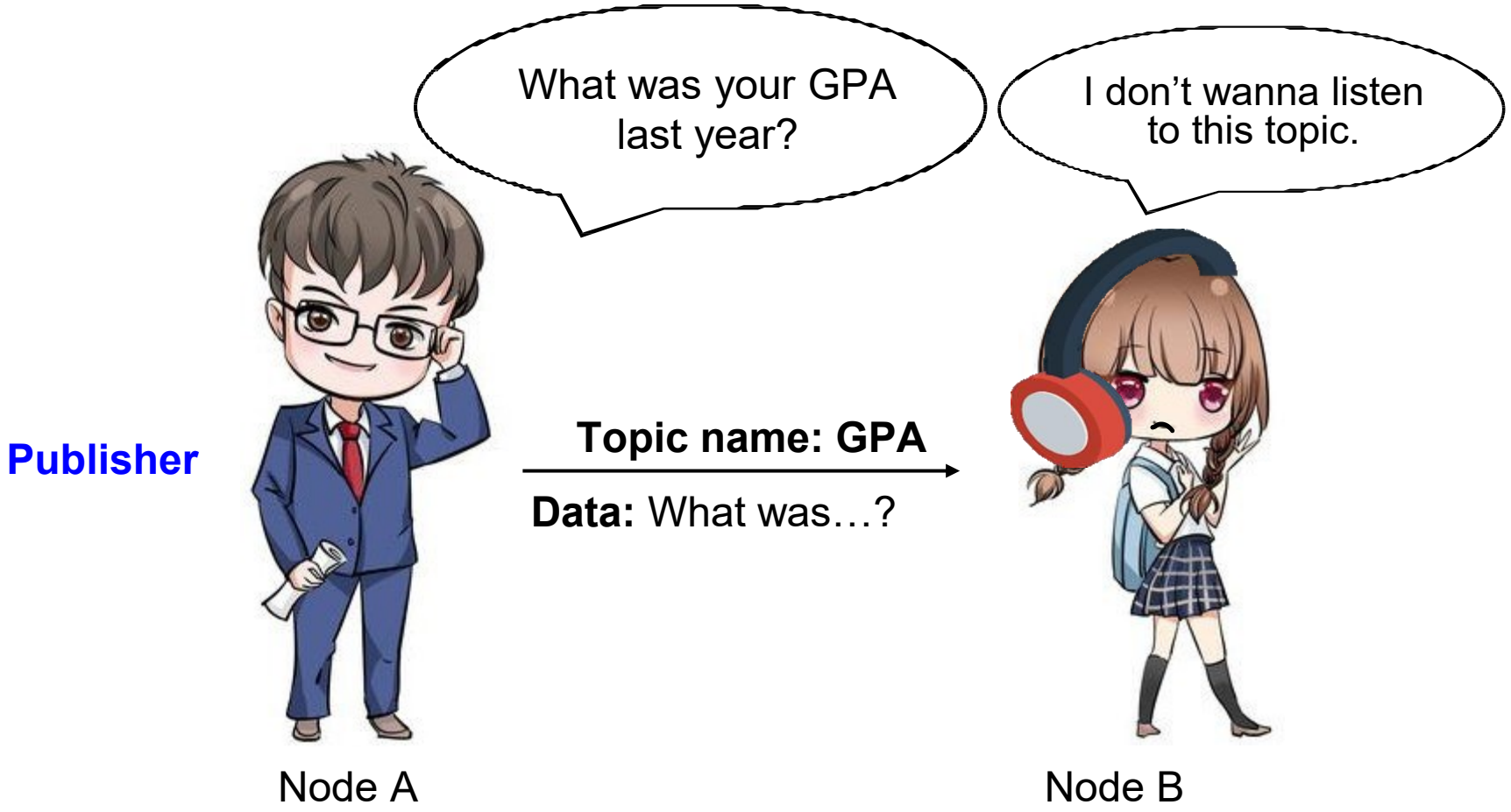
Example

If you want to ask her how her GPA last year, then you should **change the topic name**



Example

However, if she does not listen to your topic, she **will NOT receive** your data, even if you publish data on the topic



Topic Tools

Try to do it with your ROS step by step

Ref: http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

Topic Tools

Run ROS Master

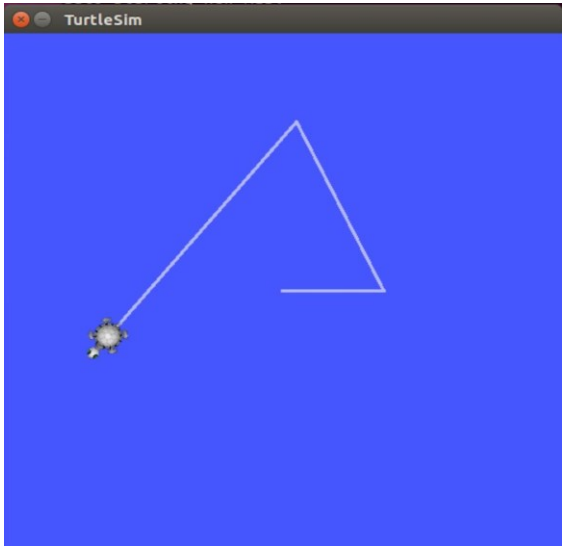
$ roscore *#You only need to do it once and keep it in the background*

Run Turtlesim

$ rosrun turtlesim turtlesim_node

Run Turtlesim's control node

$ rosrun turtlesim turtle_teleop_key




```
eleg@eleg-VirtualBox:~$ rosrun turtlesim turtlesim_node
[ INFO] [1631690510.668787374]: Starting turtlesim with node name /turtlesim
[ INFO] [1631690510.674501602]: Spawning turtle [turtle1] at x=[5.544445], y=[5.
544445], theta=[0.000000]
XmbTextListToTextProperty result code -2
XmbTextListToTextProperty result code -2
XmbTextListToTextProperty result code -2
```

```
eleg@eleg-VirtualBox:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

Topic Tools: Using rqt_graph

rqt_graph is a very useful tool to see what's going on in your ROS graph. It is a GUI plugin from **rqt tool suite**.

Install rqt

```
$ sudo apt-get install ros-noetic-rqt
```

```
$ sudo apt-get install ros-noetic-rqt-common-plugins
```

Run rqt

```
$ rosrun rqt_graph rqt_graph
```

Topic Tools: rostopic

rostopic allows you to get information about ROS topics.

You can use the help option to get the available sub-commands for rostopic.

```
$ rostopic -h
```

```
rostopic bw     display bandwidth used by topic
rostopic echo   print messages to screen
rostopic hz     display publishing rate of topic
rostopic list   print information about active topics
rostopic pub    publish data to topic
rostopic type   print topic type
```

```
$ rostopic echo [topic]
```

Let's look at the command velocity data published by the turtle_teleop_key node. This data is published on the /turtle1/cmd_vel topic.

In a new terminal, run:

```
$ rostopic echo /turtle1/cmd_vel
```

Topic Tools: rostopic

You probably won't see anything happen because **no data is being published on the topic**. Let's make turtle_teleop_key publish data by pressing the arrow keys.

```
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

Topic Tools: rostopic

rostopic list returns **a list of all topics** currently subscribed to and published.

Let's figure out what argument the list sub-command needs. In a new terminal run:

```
$ rostopic list -h
```

```
Usage: rostopic list [/topic]

Options:
  -h, --help            show this help message and exit
  -b BAGFILE, --bag=BAGFILE
                        list topics in .bag file
  -v, --verbose         list full details about each topic
  -p                    list only publishers
  -s                    list only subscribers
```

```
$ rostopic list -v
```

List full details about each topic

```
Published topics:
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 publisher
 * /rosout [rosgraph_msgs/Log] 2 publishers
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /turtle1/pose [turtlesim/Pose] 1 publisher

Subscribed topics:
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber
 * /rosout [rosgraph_msgs/Log] 1 subscriber
```

Topic Tools: rostopic

rostopic type returns the **message type** of any topic being published.

```
$ rostopic type [topic]
```

Try:

```
$ rostopic type /turtle1/cmd_vel
```

```
geometry_msgs/Twist
```

We can look at the details of the message using rosmsg:

```
$ 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
```

Topic Tools: rostopic

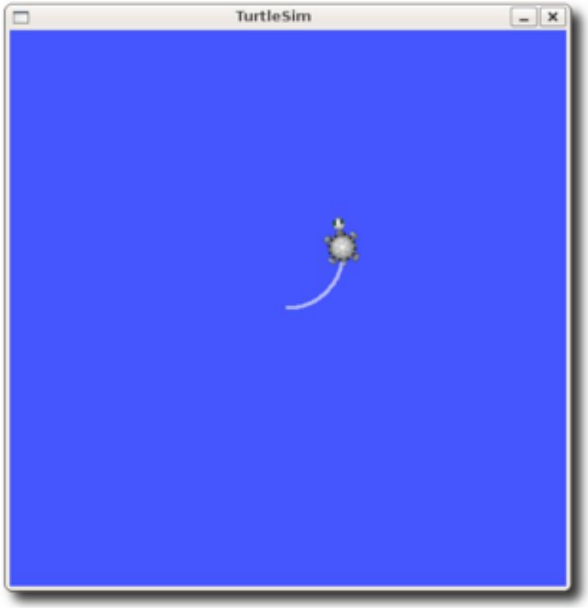
rostopic pub publishes data on a topic currently advertised.

```
$ rostopic pub [topic] [msg_type] [args]
```

Try:

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

```
geometry_msgs/Twist
```



Topic Tools: rostopic

- This command will publish messages to a given topic:

```
rostopic pub
```

- This option (dash-one) causes rostopic to only publish one message then exit:

```
 -1
```

- This is the name of the topic to publish to:

```
/turtle1/cmd_vel
```

- This is the message type to use when publishing to the topic:

```
geometry_msgs/Twist
```

- This option (double-dash) tells the option parser that none of the following arguments is an option. This is required in cases where your arguments have a leading dash -, like negative numbers.

```
--
```

- As noted before, a geometry_msgs/Twist msg has two vectors of three floating point elements each: `linear` and `angular`. In this case, '`[2.0, 0.0, 0.0]`' becomes the linear value with `x=2.0`, `y=0.0`, and `z=0.0`, and '`[0.0, 0.0, 1.8]`' is the `angular` value with `x=0.0`, `y=0.0`, and `z=1.8`. These arguments are actually in YAML syntax, which is described more in the YAML command line documentation.

```
'[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

Topic Tools: rostopic

rostopic hz reports the **rate** at which the data is published

```
$ rostopic hz [topic]
```

Try the command below and see how fast turtlesim_node is publishing /turtle1/pose:

```
$ rostopic hz /turtle1/pose
```

Here's what you'll get:

```
subscribed to [/turtle1/pose]
average rate: 59.354
        min: 0.005s max: 0.027s std dev: 0.00284s window: 58
average rate: 59.459
        min: 0.005s max: 0.027s std dev: 0.00271s window: 118
average rate: 59.539
        min: 0.004s max: 0.030s std dev: 0.00339s window: 177
average rate: 59.492
        min: 0.004s max: 0.030s std dev: 0.00380s window: 237
average rate: 59.463
        min: 0.004s max: 0.030s std dev: 0.00380s window: 290
```

Topic Tools: rostopic

Rqt_plot display a scrolling time plot of the data published on topics.

- Here we'll use rqt_plot to plot the data being published on the /turtle1/pose topic.

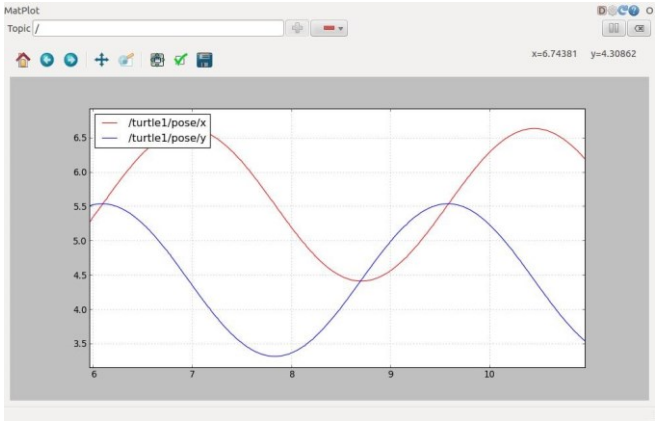
- First, start rqt_plot by trying:

```
$ rosrun rqt_plot rqt_plot
```

- In the new window that should pop up, a text box in the upper left corner gives you the ability to add any topic to the plot.

- Typing /turtle1/pose/x will highlight the plus button, previously disabled.

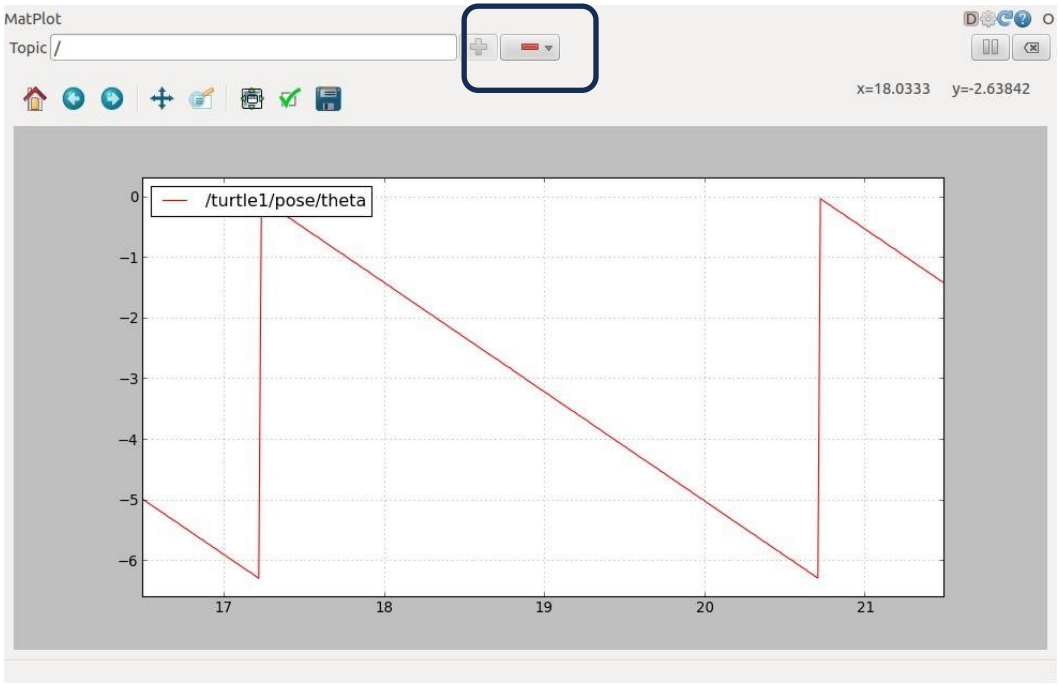
- Press it and repeat the same procedure with the topic /turtle1/pose/y. You will now see the turtle's X-Y location plotted in the graph.



Topic Tools: rostopic

- Pressing the minus button shows a menu that allows you to hide the specified topic from the plot.

- Hiding both the topics you just added and adding /turtle1/pose/theta will result in the plot shown in the next figure



- Use Ctrl – C to kill the rostopic terminal but keep your turtlesim running

How to write a Publisher?

Ref: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

How to write a Publisher in Python

Change directory into the beginner_tutorial package:

```
$ roscd beginner_tutorial
```

Create a 'scripts' folder to store our Python scripts in:

```
$ mkdir scripts
```

```
$ cd scripts
```

Then, download the example script talker.py to your newly created scripts directory and make it executable:

```
$ wget https://raw.github.com/ros/ros_tutorials/noetic-devel/rospy_tutorials/001_talker_listener/talker.py
```

```
$ chmod +x talker.py
```

We will not run it yet. You can view and edit the file with:

```
$ rosed talker.py
```

or

```
$ gedit talker.py
```

How to write a Publisher in Python

You can check with the code below:

切换行号显示

```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

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

Add the following to your **beginner_tutorials/CMakeLists.txt** in the right place. This makes sure the Python scripts gets installed properly, and uses the right Python interpreter.

```
$ catkin_install_python(PROGRAMS scripts/talker.py DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

How to write a Publisher in Python

切换行号显示

```
1  #!/usr/bin/env python
2  # license removed for brevity
3  import rospy
4  from std_msgs.msg import String
5
6  def talker():
7      pub = rospy.Publisher('chatter', String, queue_size=10)
8      rospy.init_node('talker', anonymous=True)
9      rate = rospy.Rate(10) # 10hz
10     while not rospy.is_shutdown():
11         hello_str = "hello world %s" % rospy.get_time()
12         rospy.loginfo(hello_str)
13         pub.publish(hello_str)
14         rate.sleep()
15
16 if __name__ == '__main__':
17     try:
18         talker()
19     except rospy.ROSInterruptException:
20         pass
```

Every Python ROS node will have this declaration at the top. The first line makes sure your script is executable as a Python script.

How to write a Publisher in Python

切换行号显示

```
1  #!/usr/bin/env python
2  # license removed for brevity
3  import rospy
4  from std_msgs.msg import String
5
6  def talker():
7      pub = rospy.Publisher('chatter', String, queue_size=10)
8      rospy.init_node('talker', anonymous=True)
9      rate = rospy.Rate(10) # 10hz
10     while not rospy.is_shutdown():
11         hello_str = "hello world %s" % rospy.get_time()
12         rospy.loginfo(hello_str)
13         pub.publish(hello_str)
14         rate.sleep()
15
16 if __name__ == '__main__':
17     try:
18         talker()
19     except rospy.ROSInterruptException:
20         pass
```

You need to import rospy, if you are writing a ROS node in Python.

The std_msgs.msg is imported, so that we can reuse the std_msgs/String message type for publishing.

How to write a Publisher in Python

切换行号显示

```
1 #!/usr/bin/env python
2 # license removed for brevity
3 import rospy
4 from std_msgs.msg import String
5
6 def talker():
7     pub = rospy.Publisher('chatter', String, queue_size=10)
8     rospy.init_node('talker', anonymous=True)
9     rate = rospy.Rate(10) # 10hz
```

- Defines the talker's interface to the rest of ROS.
- Line 7 declares that your node is publishing to the chatter topic, using the message type String.
- String here is the class std_msgs.msg
- The queue_size argument limits the amount of queued messages if any subscriber is NOT receiving them fast enough. (In older ROS distrubtion, just omit the argument)
- Line 8 rospy.init_node(NAME, …) is very important as it tells rospy **the name of your node** – until rospy has this info, it cannot start communicating with the ROS Master. In this case, your node will take on the name as talker (Must be a base name, i.e., cannot contain ant '/')
- Anonymous = True ensures that your node has a unique name by adding random numbers to the end of NAME

How to write a Publisher in Python

切换行号显示

```
1  #!/usr/bin/env python
2  # license removed for brevity
3  import rospy
4  from std_msgs.msg import String
5
6  def talker():
7      pub = rospy.Publisher('chatter', String, queue_size=10)
8      rospy.init_node('talker', anonymous=True)
9      rate = rospy.Rate(10) # 10hz
10     while not rospy.is_shutdown():
11         hello_str = "hello world %s" % rospy.get_time()
12         rospy.loginfo(hello_str)
13         pub.publish(hello_str)
14         rate.sleep()
15
16 if __name__ == '__main__':
17     try:
18         talker()
19     except rospy.ROSInterruptException:
20         pass
```

With its argument of 10, we should expect to go through the loop 10 times per second (as long as our processing time does not exceed 1/10th of a second!)

- Line 10: Checking the rospy.is_shotdown() flag is false - then in the loop, define hello_str
- Line 13: call pub.publish(hello_string) that publishes a string to our chatter topic
- Line 14: call rate.sleep(), which sleeps just long enough to maintain the desired rate within the loop

How to write a Publisher in Python

切换行号显示

```
1  #!/usr/bin/env python
2  # license removed for brevity
3  import rospy
4  from std_msgs.msg import String
5
6  def talker():
7      pub = rospy.Publisher('chatter', String, queue_size=10)
8      rospy.init_node('talker', anonymous=True)
9      rate = rospy.Rate(10) # 10hz
10     while not rospy.is_shutdown():
11         hello_str = "hello world %s" % rospy.get_time()
12         rospy.loginfo(hello_str)
13         pub.publish(hello_str)
14         rate.sleep()
15
16 if __name__ == '__main__':
17     try:
18         talker()
19     except rospy.ROSInterruptException:
20         pass
```

- Main function, with a standard Python __name__ == '__main__' check
- Line 19: ROS Interrupt Exception will terminate the code when Ctrl-C is pressed (node shutdown).

How to write a Subscriber?

Ref: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

How to write a Subscriber in Python

Download the listener.py file into your scripts directory

```
$ roscd beginner_tutorial/scripts/
```

```
$ wget https://raw.github.com/ros/ros_tutorials/noetic-devel/rospy_tutorials/001_talker_listener/listener.py
```

```
$ chmod +x listener.py
```

切换行号显示

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

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

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

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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

How to write a Subscriber in Python

Then, edit the catkin_install_python() call in your **beginner_tutorials/CMakeLists.txt** so it looks like the following:

```
catkin_install_python(
                PROGRAMS scripts/talker.py scripts/listener.py
                DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

How to write a Subscriber in Python

- The listen.py is similar to the talker.py, except we introduce a new callback-based mechanisms for subscribing to messages.

```
15    rospy.init_node('listener', anonymous=True)
16
17    rospy.Subscriber("chatter", String, callback)
18
19    # spin() simply keeps python from exiting until this node is stopped
20    rospy.spin()
```

- Line 15: rospy.init_node(). Still, we use anonymous = True flag, so that your **node name** will be unique (by adding random numbers to the end of NAME)
- Line 17: This declares that your node subscribes to the chatter topic, using std_msgs.msg.String datatype. When new messages are received, the callback will be triggered.
- Line 20: rospy.spin() keeps your node from existing until the node has been shutdown.

How to write a Subscriber in Python

In the previous part, we made a publisher called talker.py. You may run it:

```
$ rosrun beginner_tutorials talker.py                    #publisher
```

```
[INFO] [WallTime: 1314931831.774057] hello world 1314931831.77
[INFO] [WallTime: 1314931832.775497] hello world 1314931832.77
[INFO] [WallTime: 1314931833.778937] hello world 1314931833.78
[INFO] [WallTime: 1314931834.782059] hello world 1314931834.78
[INFO] [WallTime: 1314931835.784853] hello world 1314931835.78
[INFO] [WallTime: 1314931836.788106] hello world 1314931836.79
```

Now, the publisher node is up and running.

Now we need a subscriber to receive messages from the publisher.

```
$ rosrun beginner_tutorials listener.py                  #subscriber
```

```
[INFO] [WallTime: 1314931969.258941] /listener_17657_1314931968795I heard hello world 13149
31969.26
[INFO] [WallTime: 1314931970.262246] /listener_17657_1314931968795I heard hello world 13149
31970.26
[INFO] [WallTime: 1314931971.266348] /listener_17657_1314931968795I heard hello world 13149
31971.26
[INFO] [WallTime: 1314931972.270429] /listener_17657_1314931968795I heard hello world 13149
31972.27
[INFO] [WallTime: 1314931973.274382] /listener_17657_1314931968795I heard hello world 13149
31973.27
[INFO] [WallTime: 1314931974.277694] /listener_17657_1314931968795I heard hello world 13149
31974.28
[INFO] [WallTime: 1314931975.283708] /listener_17657_1314931968795I heard hello world 13149
31975.28
```

How to write a ROS msg?

Ref: http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv

Create a ROS msg

Let's define a **new msg** in the package that was created in the previous section

```
$ roscd beginning_tutorials
```

```
$ mkdir msg
```

```
$ echo 'int64 num' > msg/Num.msg
```

The example Num.msg file above contains only 1 line. You can create a more complex one by adding multiple elements, **one per line**, be like

```
string first_name
string last_name
uint8 age
uint32 score
```

There's one more step. We need to make sure that the msg files are turned into source code for C++, Python, and other languages (msg is program-lang. independent)

Open package.xml, and make sure these two lines are in it and uncommented:

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

At build time, we need message_generation; At run time, we need message_runtime

Create a ROS msg

- Open beginner_tutorials/CMakeLists.txt in a text editor (rosed, gedit).
- Add the message_generation dependency to the find_package call which already exists in your CMakeLists.txt so that you can generate messages.
- You can do this by simply adding message_generation to the list of COMPONENTS such that it looks like this:

```
# Do not just add this to your CMakeLists.txt, modify the existing text to add message_generatio
n before the closing parenthesis
find_package(catkin REQUIRED COMPONENTS
   roscpp
   rospy
   std_msgs
   message_generation
)
```

- Also make sure you export the message runtime dependency

```
catkin_package(
  ...
  CATKIN_DEPENDS message_runtime ...
  ...)
```

Find the following block of code: Uncomment it by removing the # symbols and then replace the stand in Message*.msg files with your .msg file, such that it looks like what's on the right

```
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )
```

```
add_message_files(
  FILES
  Num.msg
)
```

Create a ROS msg

Now you need to ensure the generate_messages() function is called.

For ROS Hydro and later, you need to uncomment these lines:

```
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )
```

so it looks like:

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

That's all you need to do to create a msg. Make sure that ROS can see it using the rosmsg show command.

```
$ rosmsg show [message]
```

So yours should be like:

```
$ rosmsg show beginner_tutorials/Num
```

Create a ROS msg

You will see:

```
int64 num
```

If you cannot remember which package a msg is in, you can leave out the package name. Try this:

```
$ rosmsg show beginner_tutorials/Num
```

Then, you will see:

```
[beginner_tutorials/Num]:
int64 num
```

Unless you have already done this in the previous steps, change in CMakeLists.txt. :

```
# generate_messages(
#   DEPENDENCIES
# #  std_msgs  # Or other packages containing msgs
# )
```

Uncomment it, and add any packages you depend on which contain .msg files that your messages use (in this case std_msgs), such that it looks like this:

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

Create a ROS msg

Now that we have made some new messages, we need to CMake our package again:

```
# In your catkin workspace
$ roscd beginner_tutorials
$ cd catkin_ws
$ catkin_make
$ source ./devel/setup.bash
```

- Any .msg file in the msg directory will generate code for use in all supported languages.
- The **C++** message header file will be generated in ~/catkin_ws/devel/include/beginner_tutorials/.
- The **Python** script will be created in ~/catkin_ws/devel/lib/python2.7/dist-packages/beginner_tutorials/msg.
- The **lisp file** appears in ~/catkin_ws/devel/share/common-lisp/ros/beginner_tutorials/msg/.

Today's task

- Submit grouping list
- Lab sheet 4