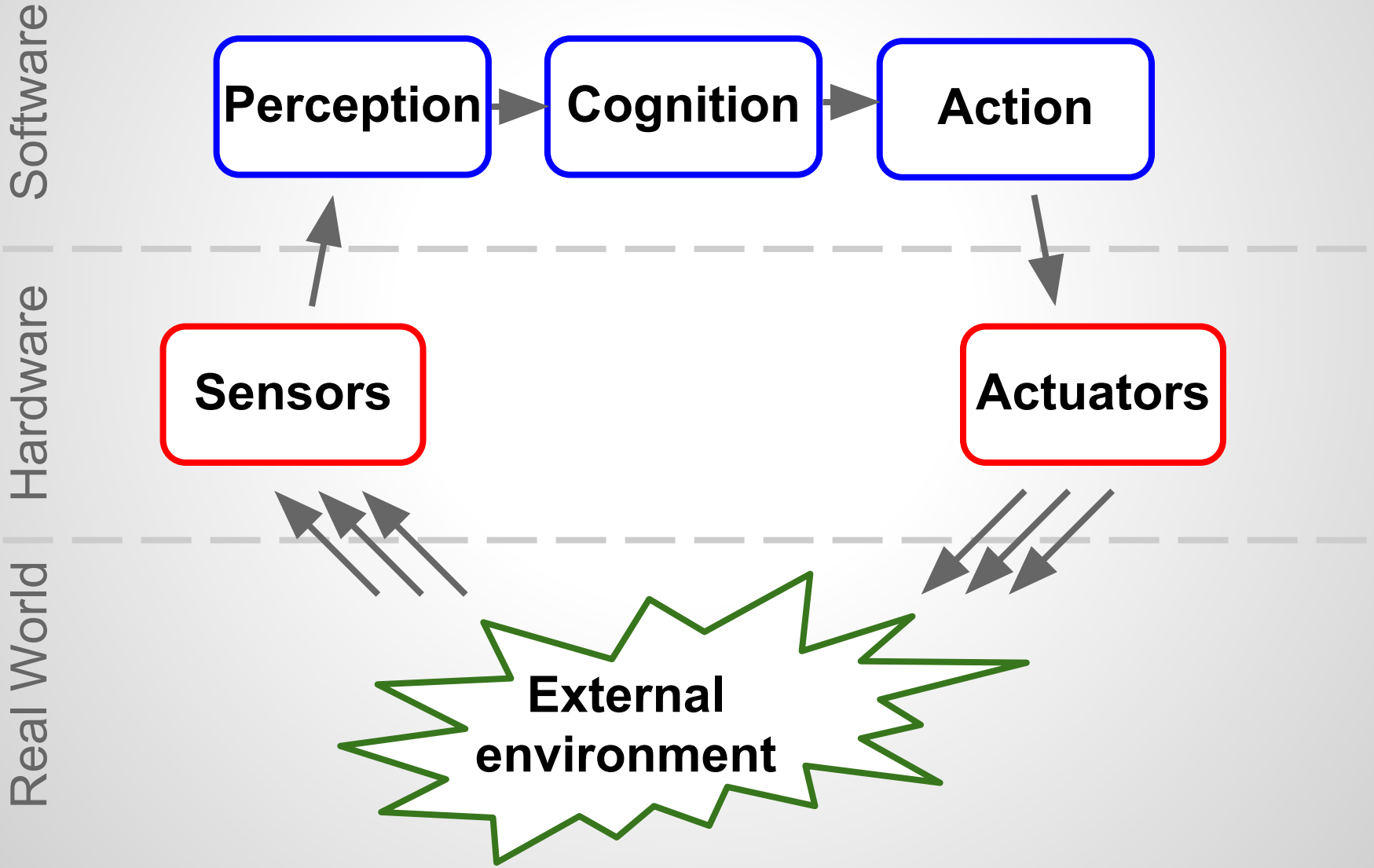
Robotics and Perception

CMSC498F

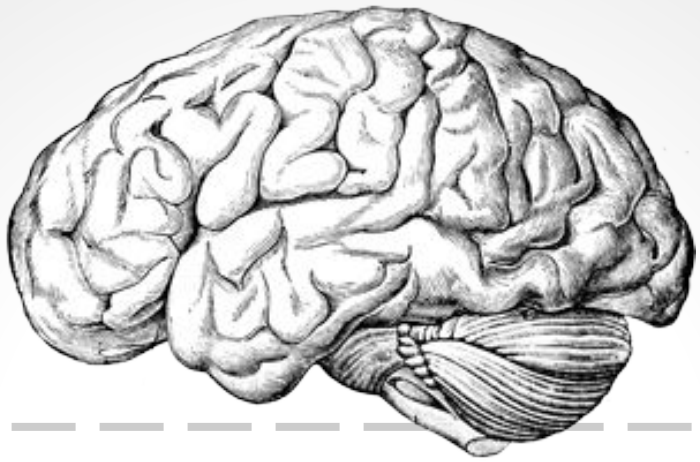
Lecture 3:
Introduction to
ROS

(adapted from Todd Hester)

How does a robot work



How does a robot work



Software

Hardware

Real World

Sensors

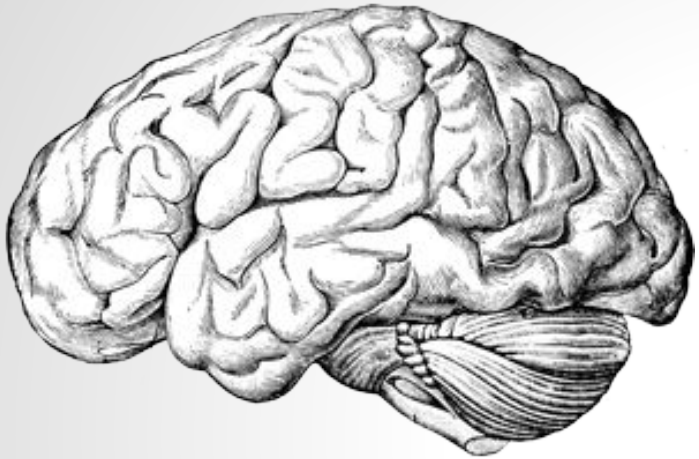
Actuators

External environment

Robot example

Robot example

Controlling robots using code



Brute force approach

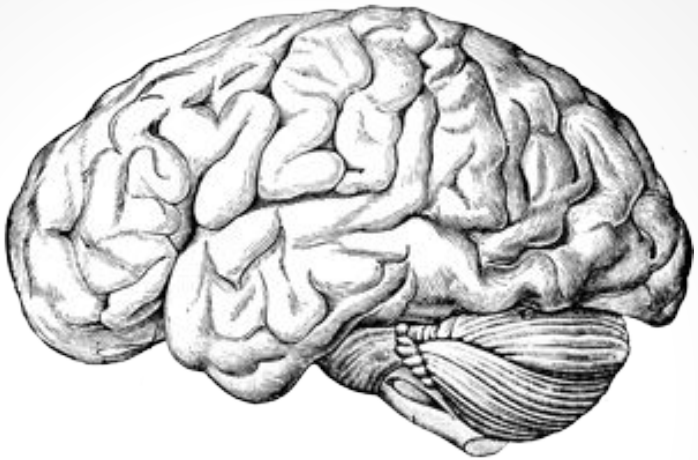
Just write and compile a program to perform robot's "cognitive" functions

This program will include:

- Code to interface with Kinect camera and Pioneer robot base
- Code to process visual data from Kinect
- Code to control the Pioneer

Brute force approach

Software



Kinect driver

Pioneer driver

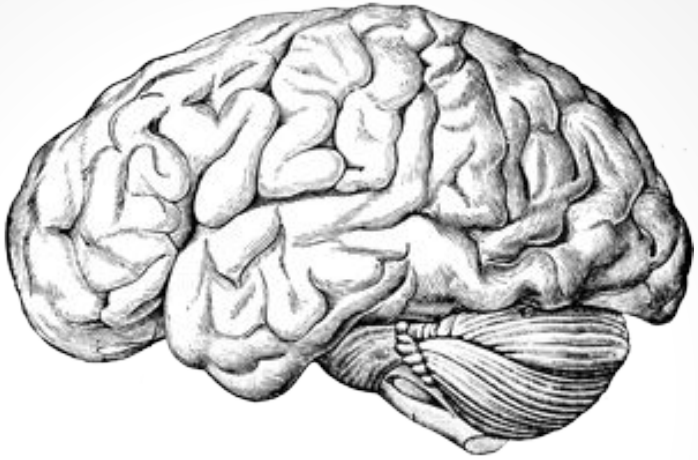
Hardware

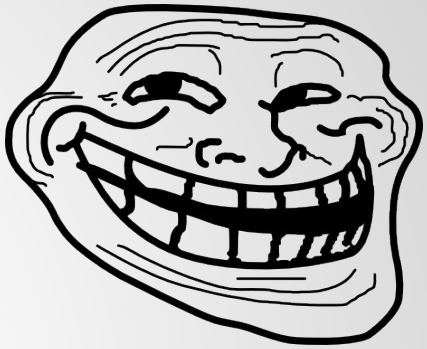
USB

USB

Brute force approach

Software





Problem?

Kinect driver

Pioneer driver

Hardware

USB

USB

Problems with brute force approach

What if we decide that we can get a better results using Kinect 2.0?
What if we want to share our code with a different lab, but they only have a Roomba robot, not a Pioneer?
Do we have to rewrite a big chunk of the code to accommodate new interfaces?

Problems with brute force approach

What if we decide that we can get a better results using Kinect 2.0?
What if we want to share our code with a different lab, but they only have a Roomba robot, not a Pioneer?
Do we have to rewrite a big chunk of the code to accommodate new interfaces?

Problems with brute force approach

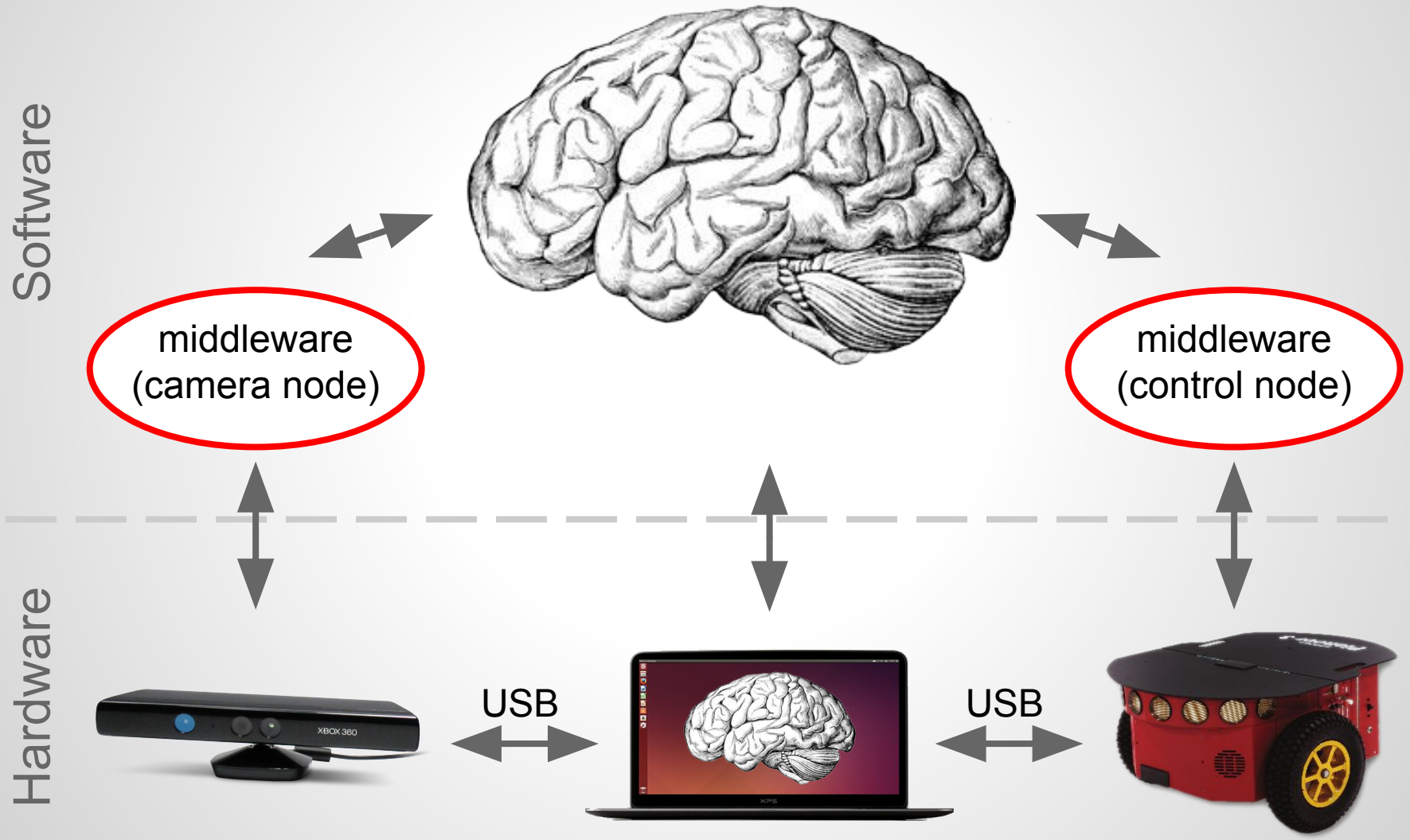
What if we decide that we can get a better results using Kinect 2.0?
What if we want to share our code with a different lab, but they only have a Roomba robot, not a Pioneer?
Do we have to rewrite a big chunk of the code to accommodate new interfaces?

Enter robotic middleware

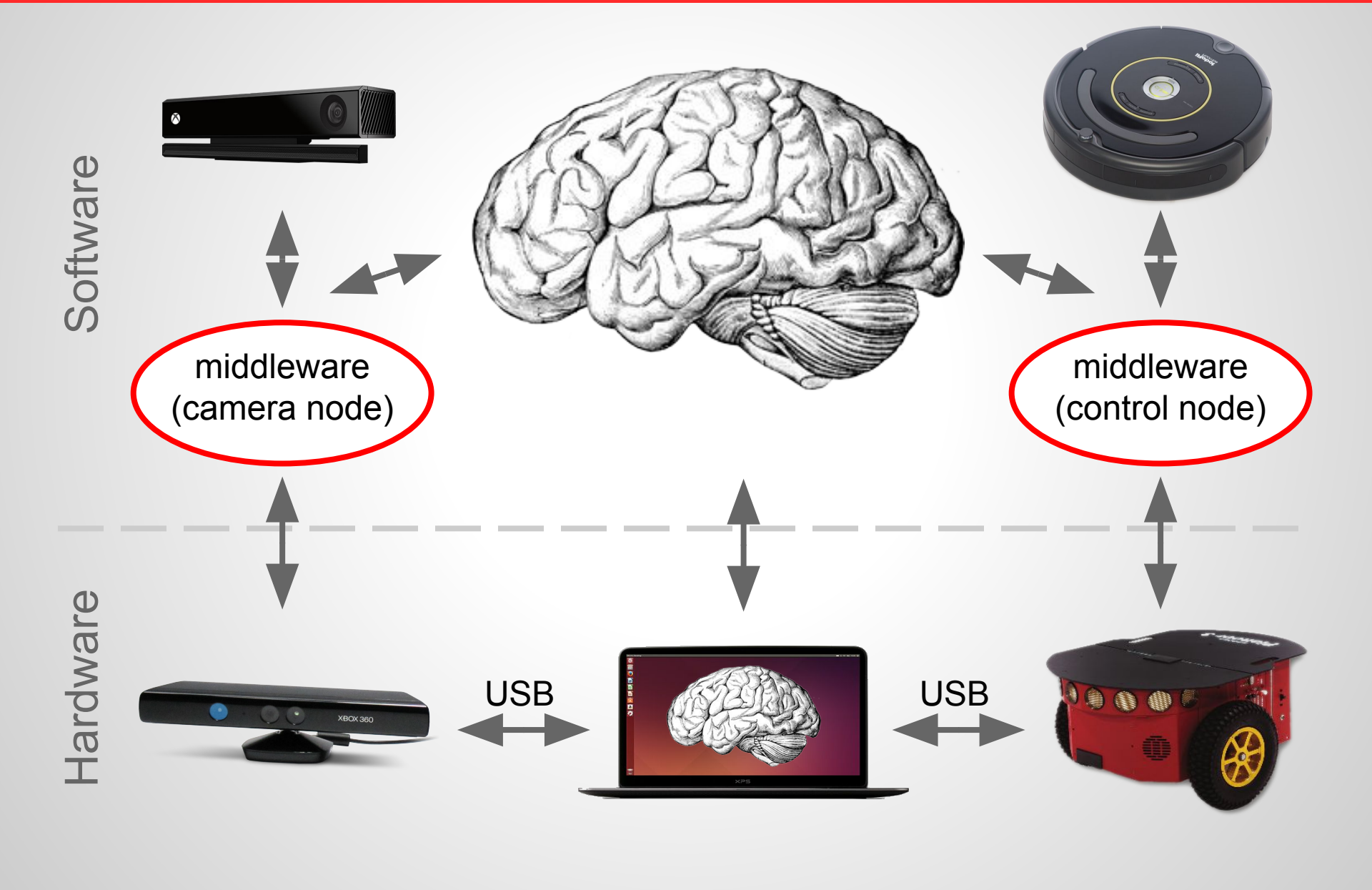
Robotic middleware provides an abstraction layer between computation and robot hardware.
Similar to OS hardware abstraction which allows your program to work independent of the actual hardware.

- i.e. hardware abstraction layer in an OS

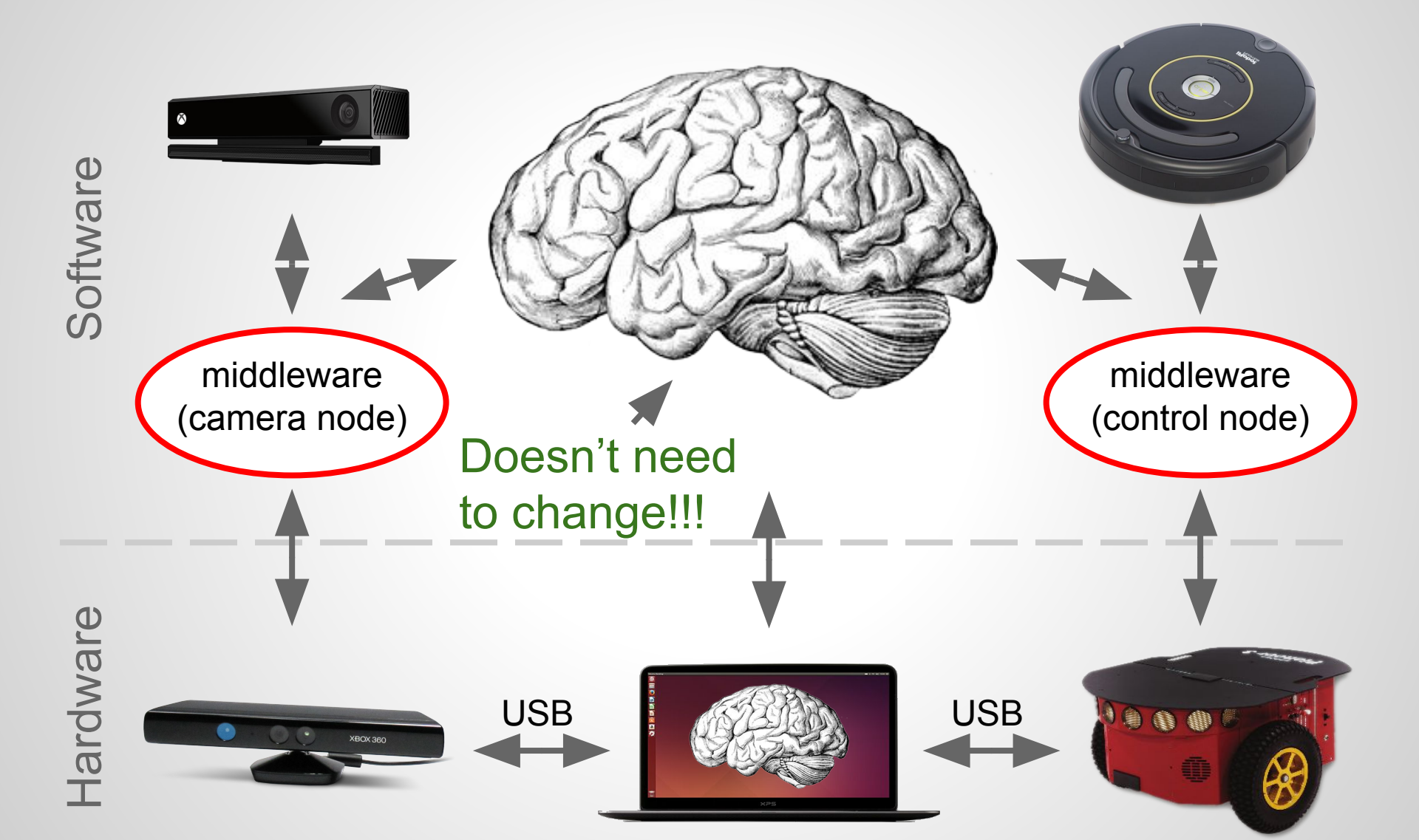
Example: a mobile platform



Example: a mobile platform



Example: a mobile platform

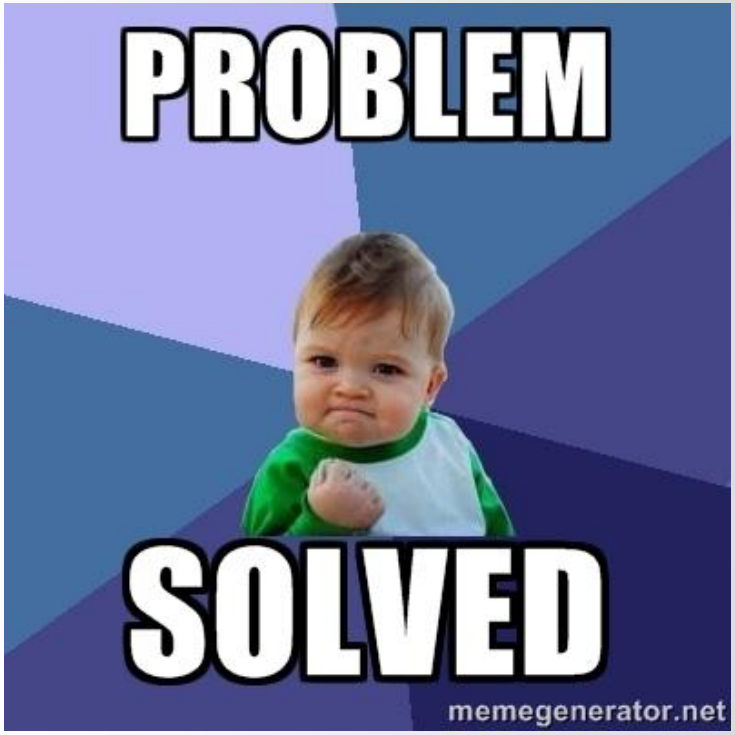


Advantages

- Reusability
  - Reuse code written by other researchers and share your code with them
- Portability
  - When you get a new robotic platform it's easier to transfer your code to the new platform
- Easier to expand functionality

Advantages

- Reusability
  - Reuse code written by other researchers and share your code with them
- Portability
  - When you get a new robotic platform it's easier to transfer your code to the new platform
- Easier to expand functionality



Why ROS?

A number of such middleware robot frameworks exist:

- Player, YARP, ROS, Microsoft Robotics Studio etc.

ROS is open source

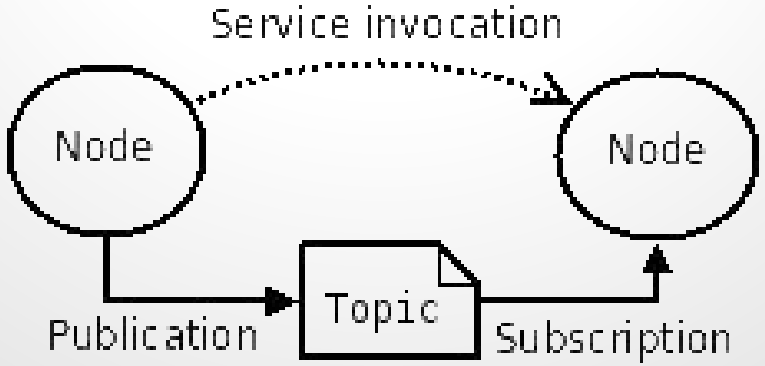
ROS allows running processes on a distributed network and is architecture agnostic

ROS is de-facto standard in robotic community:

- excellent support for hardware drivers
- largest library of existing robotic algorithms (navigation, 3D perception, grasping)
- vibrant online community (http://answers.ros.org/)

A super quick overview of ROS

- Computation is distributed among many processes called nodes.
- Each node is responsible for a certain robot functionality.
- Nodes exchange data and using the “publish-subscribe” messaging on different “topics”.
- Topics are named channels over which messages are exchanged.

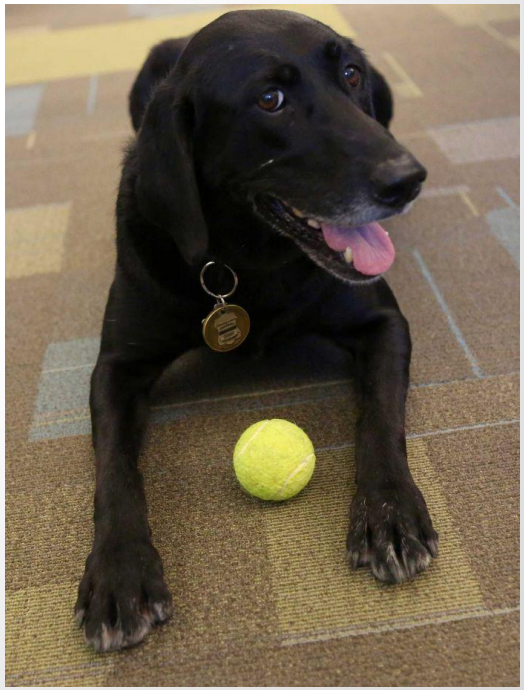


ROS example

We want a robot that goes after tennis balls

- What nodes might we use?
- What messages would we send?

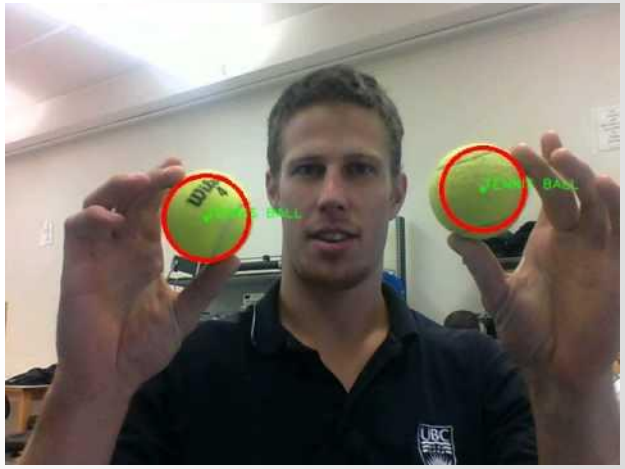
≈



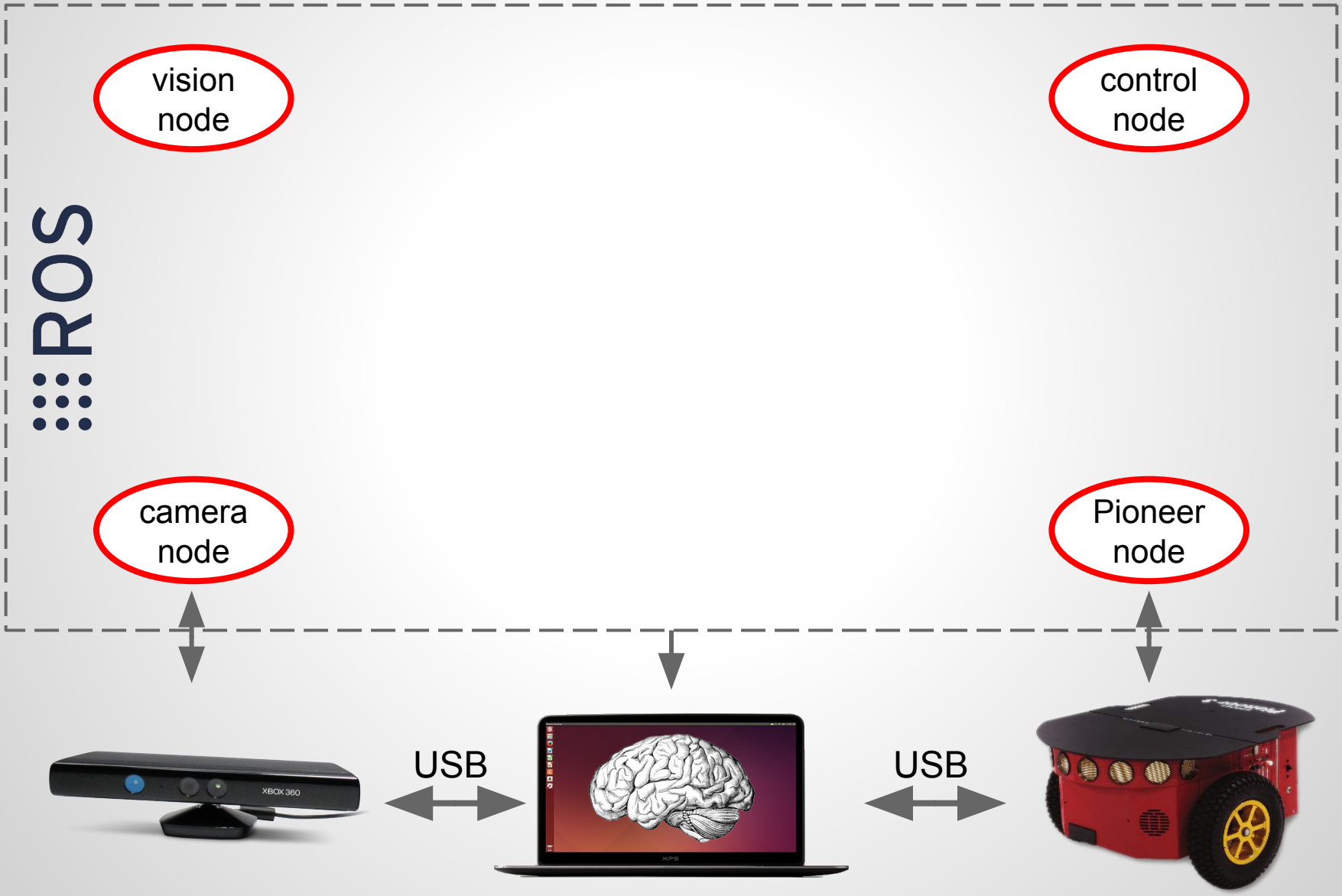
ROS example

Split the code into 4 nodes:

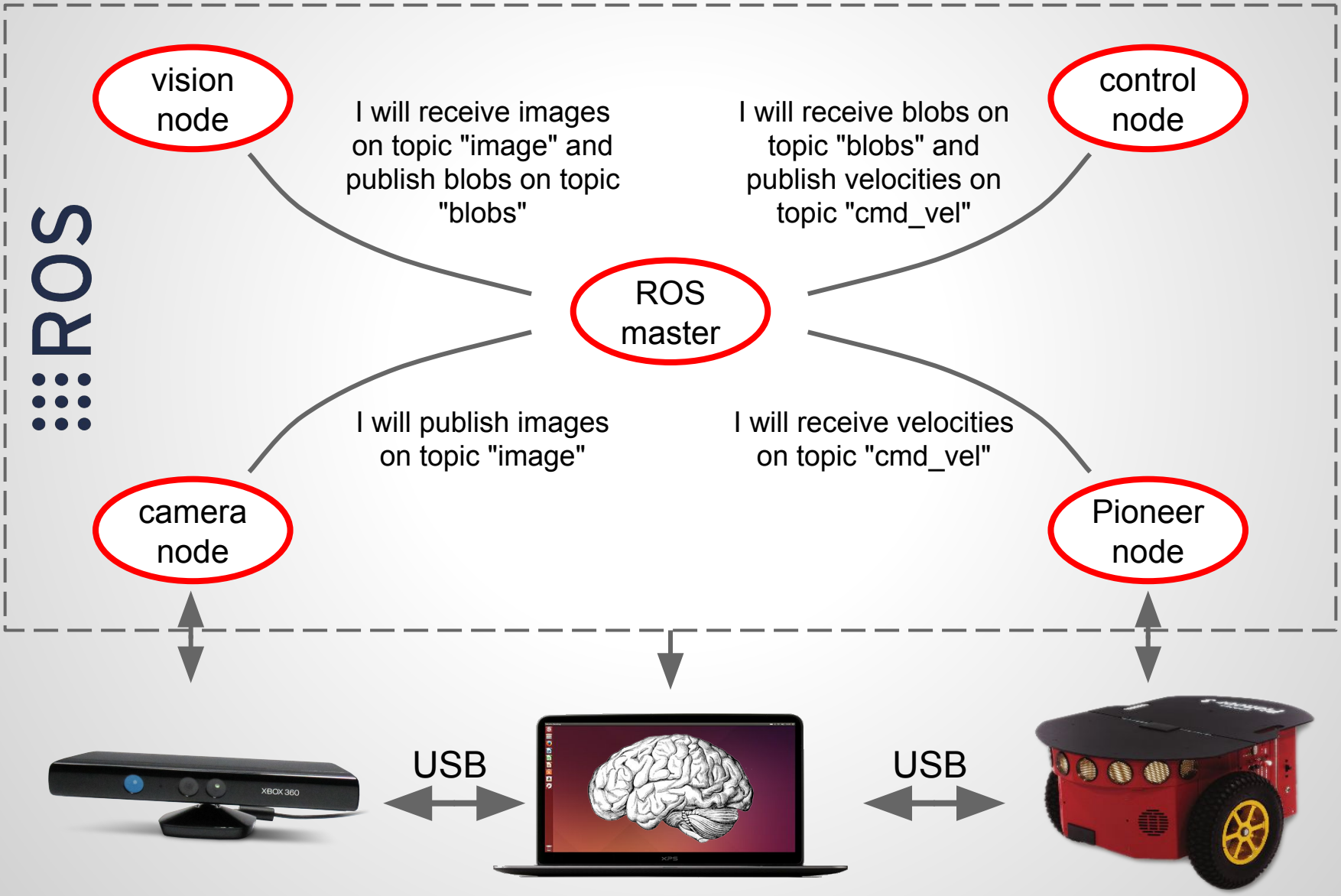
- Camera node - produces images from the camera
- Pioneer node - accepts forward and angular velocity and makes the Pioneer move
- Blobfinder node - takes an image and returns the position of the tennis ball on the screen
- Control node - takes the position of the tennis ball and calculates the velocities required to reach it.



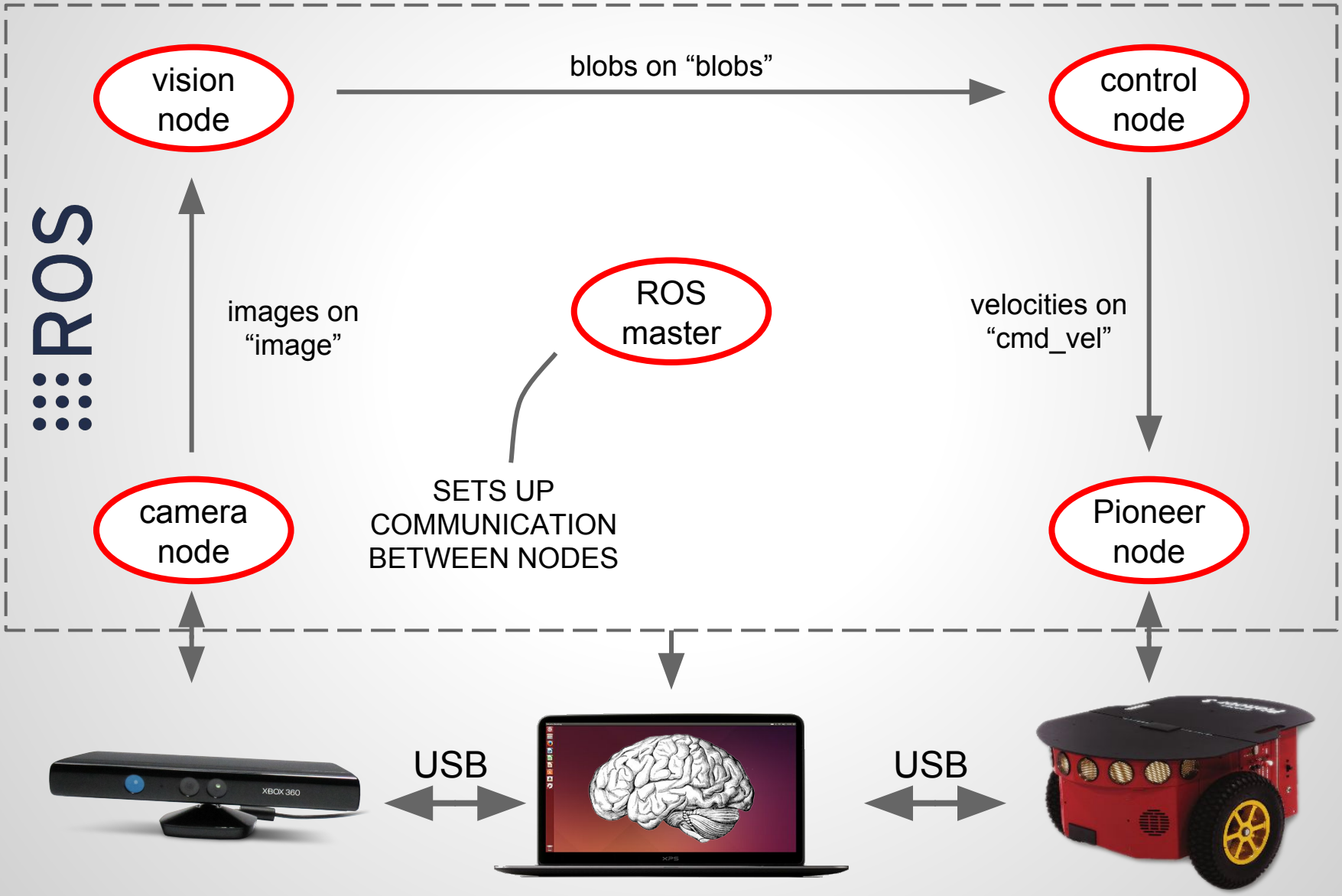
ROS example


vision node
control node
ROS
camera node
Pioneer node
USB
USB

ROS example


ROS
vision node
I will receive images on topic "image" and publish blobs on topic "blobs"
control node
I will receive blobs on topic "blobs" and publish velocities on topic "cmd_vel"
ROS master
I will publish images on topic "image"
I will receive velocities on topic "cmd_vel"
camera node
Pioneer node
XBOX 360
USB
USB

ROS example


ROS
vision node
blobs on “blobs”
control node
images on “image”
ROS master
velocities on “cmd_vel”
SETS UP COMMUNICATION BETWEEN NODES
camera node
Pioneer node
USB
USB

ROS is not...

ROS is a system for controlling robots from a PC.
ROS is not:

- a computer OS. ROS runs under Ubuntu, Windows, OS X, Android (however highest compatibility with Ubuntu)
- a programming language. You can write code with ROS in Python, C++ and other languages
- a library. However, a lot of important robotic algorithms have implementations in ROS
- an IDE. You can write code for ROS in any IDE or text editor)

Packages

All software in ROS is organized in *packages*.
Each package can define multiple nodes.
A valid ROS *package* must have a specific structure:

```
/package_name
    CmakeLists.txt          (tells CMake how to build)
    Package.xml             (a description of what's in the package)
    /src
        /node1.py           (Python node 1)
        /node2.py           (Python node 2)
        ...
```

Packages: useful commands

To create a new *package*:

```
catkin_create_pkg package_name
```

ROS can find *packages* for you

```
rospack find package_name      (print location of a package)
roscd package_name             (change directory to package folder)
rosls package_name             (list contents of package folder)
```

To find the *packages* ROS checks the subfolders of the paths stored in the environment variable `ROS_PACKAGE_PATH`

```
$ echo $ROS_PACKAGE_PATH
/opt/ros/indigo/share:/opt/ros/indigo/stacks:
/home/<myname>/ros
```

Nodes

A *node* is a process that performs some computation.
A robot control system will usually comprise many *nodes* each having a specific function.
Nodes can be named, so that multiple instances of a node can be running at the same time.

Useful commands:

`rosrun package_name executable_name`	(launch an instance of a node)
`rosnode list`	(get a list of active nodes)
`rosnode ping /nodename`	(test connectivity to node)
`rosnode info /nodename`	(information about a node)

Topics and messages

The primary mechanism for inter *node* communication is *message* passing.

Topics are named buses over which nodes exchange *messages*.

A *topic* is defined by it's name and the type of *message* it uses.

A *node* can:

- publish *messages* to a *topic* (output)
- subscribe to a *topic* to read *messages* (input)

Common message types:

```
std_msgs/String
std_msges/Int32
sensor_msgs/Image
...
```

Topics and messages: example

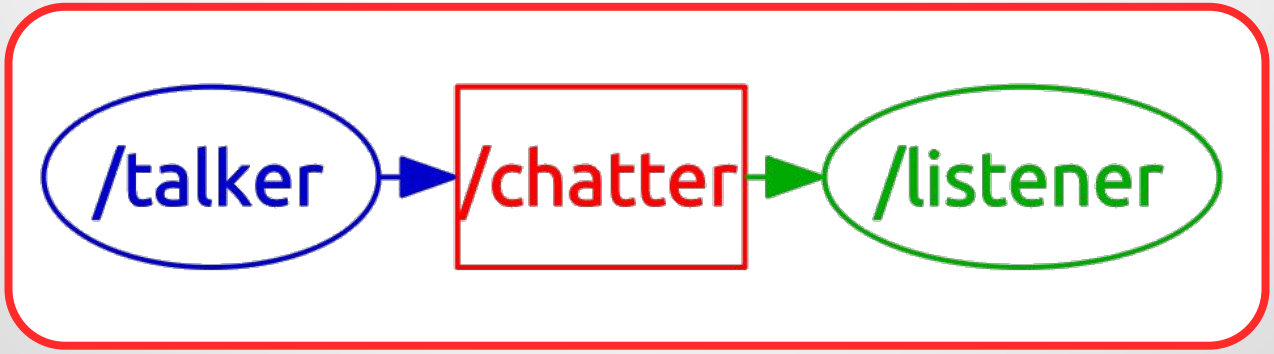
There are two nodes:

`/talker`

`/listener`

`/talker` publishes messages of type `std_msgs/String` to topic `/chatter`

`/listener` subscribes to topic `/chatter`



Topics and messages: useful commands

Topics:

`rostopic list`	(get a list of active topics)
`rostopic info /topic_name`	(get information about topic)
`rostopic type /topic_name`	(get the type of message used by the topic)
`rostopic echo /topic_name`	(print messages published on a topic)
`rostopic pub  /topic_name`	(publish a message to a topic)

Messages:

`rosmsg show message_type`	(show the format of a message type)

Publisher example

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

Subscriber example

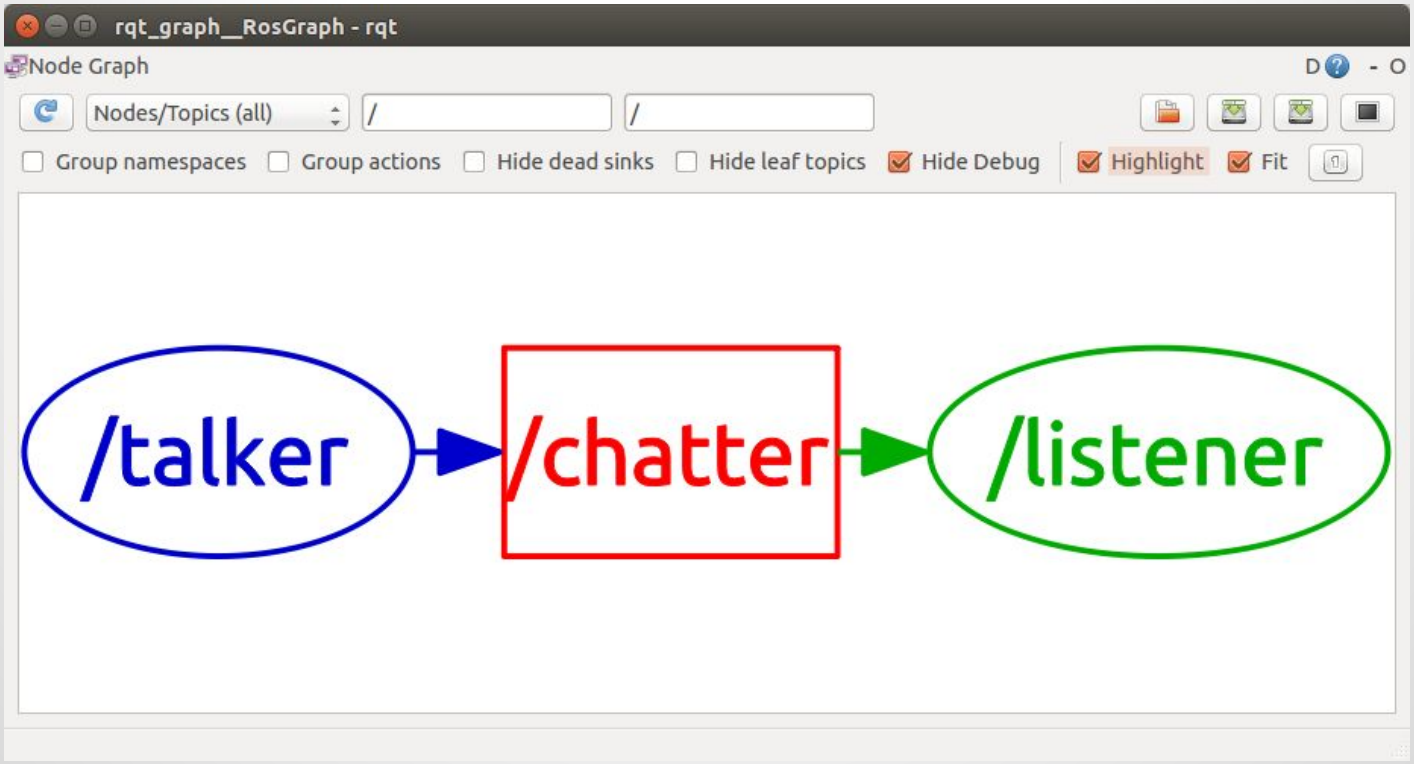
```
1  #!/usr/bin/env python
2  import rospy
3  from std_msgs.msg import String
4
5  def callback(data):
6      rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
7
8  def listener():
9      rospy.init_node('listener', anonymous=True)
10     rospy.Subscriber("chatter", String, callback)
11
12     # spin() simply keeps python from exiting until this node is stopped
13     rospy.spin()
14
15 if __name__ == '__main__':
16     listener()
```

Visualizing nodes and topics

ROS has a built in tool to visualize node connectivity.

To launch it:

```
rosrun rqt_graph rqt_graph
```



Logging

`rosbag` allows to record all of the topics and time stamped messages that were published on these topics over a period of time.
Recorder data is stored in `.bag` files.
Playing back a bag file "reproduces" the recorded ROS system

Usage:

```
rosbag record -a              (start recording all topics)
rosbag info bag_filename      (get information about a bag file)
rosbag play bag_filename      (playback recorded data)
```

More things...

- ROS environment
- Package dependencies
- Compiling packages
- Services

Installation

It is strongly recommended to install it under Ubuntu.
For this course we are using ROS version Indigo.
Installation instructions can be found here:
http://wiki.ros.org/indigo/Installation/Ubuntu

If you are doing a clean install of Ubuntu, there is an Ubuntu distribution with ROS Indigo preinstalled.

Assignment 0

Best way to learn is to do it yourself:

- Install ROS on your machine
- Go through ROS beginner tutorials 1-18
  http://wiki.ros.org/ROS/Tutorials (using Python)

Assignment details are posted on website!
The assignment is due next Tuesday (February 7th).

Tips and tricks

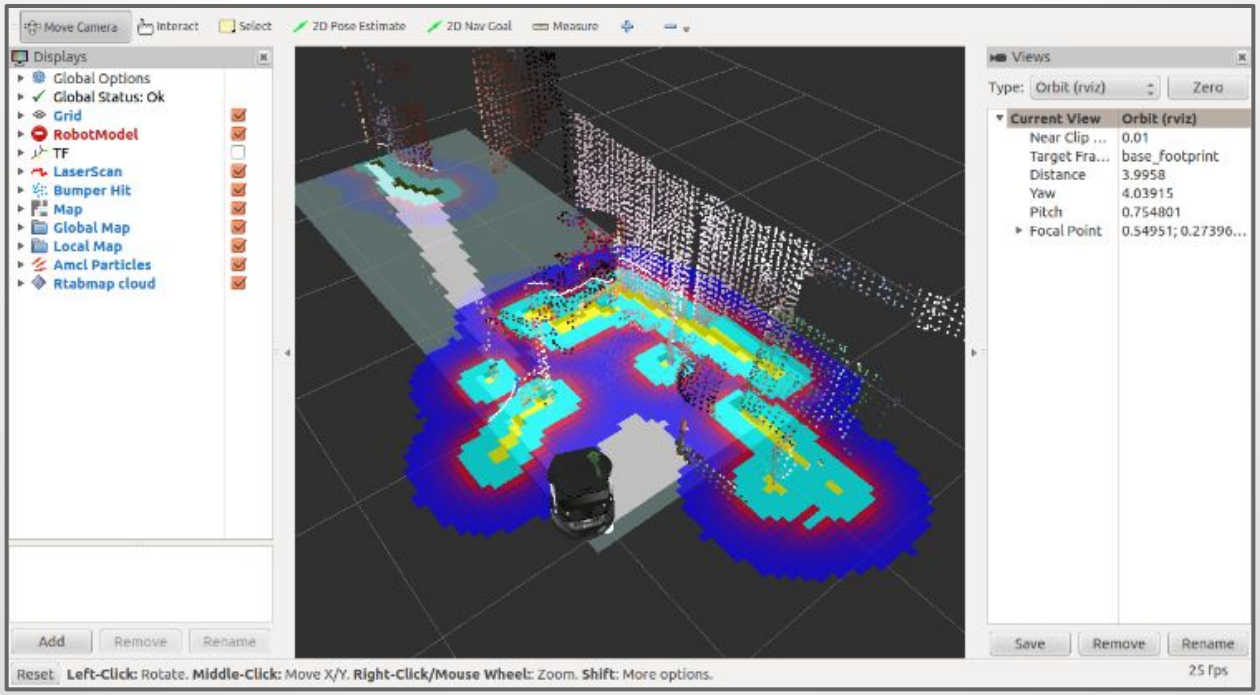
- If something doesn't work - restart `roscore`
- If it is still broken - restart your computer
- Don't forget to make your Python scripts executable

```
chmod +x script_name
```

- Use ROS cheat sheet: https://github.com/ros/cheatsheet/releases/tag/0.0.1
- Don't hesitate asking for help:
  - Me, Cornelia
  - fellow students
  - http://answers.ros.org/

Next time on ROS

- Representing robot model and state
- 3D visualization tool for monitoring robot state
- Robot simulation
- Turtlebot robot!



The end!

Questions?