The Robot Operating System

A 5-day ROS crash course (2^{nd} *edition*)

Dr. Murilo Fernandes Martins

Department of Electrical Engineering
Centro Universitário da FEI

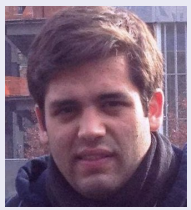
27 January 2014

Part ∅: Meet the team

Course Lecturer

Dr. Murilo Fernandes Martins
Research Fellow in Intelligent Robotics
Dept. of Electrical Engineering, FEI

Teaching Assistant



Lucas Malassise Argentim
5^{th} year Control & Automation student
Dept. of Electrical Engineering, FEI

ROS: suggested learning process

The process of learning ROS can be partitioned into...

- Learning HOW to use ROS and its tools
  - Core concepts of the framework
  - Command line tools
  - Graphical tools

ROS: suggested learning process

The process of learning ROS can be partitioned into...

- Learning HOW to use ROS and its tools
  - Core concepts of the framework
  - Command line tools
  - Graphical tools
- Learning HOW to make use of ROS to facilitate R & D
  - Mapping, localisation, navigation
  - Development of new nodes using ROS APIs
  - Object recognition and manipulation (if we have time)

Part I: Course programme for the 5 days

1. Day 1 – An Introduction to ROS
2. Day 2 – Diving into ROS with practical examples
   - Tutorial I: Getting familiar with ROS tools
   - Tutorial II: Programming ROS nodes in C++
3. Day 3: Case Study 1 – Mobile robots, SLAM and navigation
4. Day 4: Case Study 2 – Kinect-based Gesture Recognition for Robot Control
5. Day 5: Hackathon! Group project

Day 1

An Introduction to ROS

1. What is ROS?
2. ROS Community Level
3. Getting started
4. ROS Filesystem Level
5. ROS Computation Graph Level

Day 2 – Tutorial I

Getting familiar with ROS tools

- ROS Master and rosbash
- ROS nodes
- ROS topics
- ROS messages
- ROS services
- Parameter Server
- Rqt: Qt-based framework GUI for ROS
- Launch files

Day 2 – Tutorial II

Programming ROS nodes in C++

- Programming a publisher node
- Coding a subscriber node
- Writing a node which provides a service
- Calling a service from within a node

Day 3: Case Study 1

Using ROS with mobile robots for SLAM and navigation

- Preparing the testbed: the Gazebo simulator and the HUSKY robot
- Simultaneous Localisation and Mapping (SLAM) using GMAPPING
- Probabilistic robot localisation using the AMCL algorithm

Day 3: Case Study 1

Using ROS with mobile robots for SLAM and navigation

- Preparing the testbed: the Gazebo simulator and the HUSKY robot
- Simultaneous Localisation and Mapping (SLAM) using GMAPPING
- Probabilistic robot localisation using the AMCL algorithm

Also covering key concepts of ROS, featuring

- rviz: 3D visualisation environment for robots using ROS
- Visualising the ROS computation graph in real time
- rosbag: recording data from and playing back to ROS Topics
- ROS launch files

Day 4: Case Study 2

Kinect-based Gesture Recognition for Robot Control

- Coordinate frames and transformations in ROS
  - Listening for Transforms
  - Broadcasting Transforms
- People detection and skeletal tracking: OpenNI + NITE
- Implementing a Kinect-based gesture recognition system
- Controlling a real robot: Peoplebot
- Verifying the ROS computation graph in real time
- Live demonstration

Day 5: Hackathon!

Solve VRC Task 1 challenge

- use Husky-FEI from **husky_fei_gazebo** package
- load vrc_task1_world.launch from **fei_gazebo** package
- go through gates 1 to 4 (without getting off road)

Solve VRC Task 2 challenge

- use Husky-FEI from **husky_fei_gazebo** package
- load vrc_task6_world.launch from **fei_gazebo** package
- go through gates 1 to 5 (including a swim in the mud pool)

Group project

Implement your own idea! Have you got any?

Part II: Day 1 – An Introduction to ROS

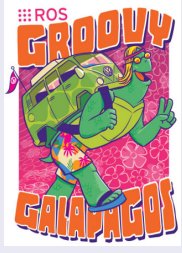
1. What is ROS?
2. ROS Community Level
3. Getting started
   - Supported operating systems
   - Supported hardware
   - Installation
4. Setting up VM
5. ROS filesystem
6. ROS graph concepts
   - Nodes
   - Master
   - Parameter server
   - ROS Messages
   - Topics
   - Services
   - Messages, Topics and Services
   - Revisiting the filesystem

What is ROS?

ROS is an open-source meta-operating system

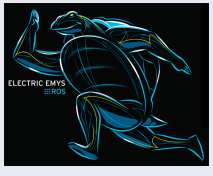
I — Indigo Igloo
April 2014

Hydro Medusa
4 September 2013



Groovy Galapagos
31 December 2012

Fuerte Turtle
23 April 2012



Electric Emys
30 August 2011

Diamondback
02 March 2011

C Turtle
02 August 2010

Box Turtle
02 March 2010

ROS key features

Features resembling a real operating system

- hardware abstraction and low-level device control

ROS key features

Features resembling a real operating system

- hardware abstraction and low-level device control
- message passing between processes (OS-independent)

ROS key features

Features resembling a real operating system

- hardware abstraction and low-level device control
- message passing between processes (OS-independent)
- programming language independence

ROS key features

Features resembling a real operating system

- hardware abstraction and low-level device control
- message passing between processes (OS-independent)
- programming language independence
- implementation of a wide range of commonly used algorithms

ROS key features

Features resembling a real operating system

- hardware abstraction and low-level device control
- message passing between processes (OS-independent)
- programming language independence
- implementation of a wide range of commonly used algorithms
- standardised package management

ROS key features

Features resembling a real operating system

- hardware abstraction and low-level device control
- message passing between processes (OS-independent)
- programming language independence
- implementation of a wide range of commonly used algorithms
- standardised package management
- useful set of shell commands and utilities with tab completion

ROS key features

ROS is inherently distributed

Structured as a peer-to-peer network of processes (nodes), loosely coupled at runtime, which may share messages using:

ROS key features

ROS is inherently distributed

Structured as a peer-to-peer network of processes (nodes), loosely coupled at runtime, which may share messages using:

- synchronous RPC-style communication (over Services)

ROS key features

ROS is inherently distributed

Structured as a peer-to-peer network of processes (nodes), loosely coupled at runtime, which may share messages using:

- synchronous RPC-style communication (over Services)
- asynchronous data streaming communication (over Topics)

ROS key features

ROS is inherently distributed

Structured as a peer-to-peer network of processes (nodes), loosely coupled at runtime, which may share messages using:

- synchronous RPC-style communication (over Services)
- asynchronous data streaming communication (over Topics)
- storage of data (on a Parameter Server)

ROS concepts and components

ROS client libraries

Main client libraries:

- Python
- C++
- Lisp

Experimental client libraries:

- Java (with Android support)
- Lua
- R

ROS Community Level

- Distributions
  - Collections of versioned packages you can install
  - Similar role to Linux distributions
- Repositories
  - ROS relies on a federated network of code repositories
  - All core packages hosted on GitHub, along with issue tracking
  - Most package developers also host code on GitHub
- ROS Wiki
  - main forum for documenting information about ROS
  - Anyone can contribute writing new tutorials, documentation, providing corrections and updates
- Mailing Lists: communication channel about updates
- ROS Answers: Q&A site for ROS users to ask/answer questions
- Blog: regular updates, news, projects, ...

FEI-ROS Resources

- fei-ros-pkg GitHub organisation
  - http://github.com/fei-ros-pkg
  - ROS packages for Husky-FEI robot
  - ROS packages used during this course
- Other resources
  - http://www.fei.edu.br/∼murilo/teaching.html
  - handout of classes
  - other useful links

Supported operating systems

Supported operating system

Ubuntu (12.04 LTS + ROS Hydro)

Experimental

Ångström

OpenEmbedded

Arch

OpenSuse

Debian

Raspbian

Fedora

Windows

Gentoo

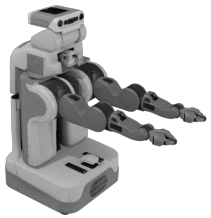
Ubuntu ARM

Mac OS X

Udoo

Supported robots


Nao


Willowgarage PR2


Baxter


Care-o-Bot


Toyota Helper


Gostai Jazz


Robonaut


Peoplebot


Kuka YouBot


Guardian


Husky A200


Summit


Turtlebot


Erratic


Qbo


AR.Drone


REEM-C


AscTec Pelican


Lego NXT


Pioneer


SIA 10D

Many more at http://www.ros.org/wiki/Robots

Videos – ROS Overview

Video: Celebrating 3 years of ROS[1]

[1] Video available at http://youtu.be/7cslPMzklVo

Videos – ROS Overview

Video: Celebrating 5 years of ROS[2]

[2]Video available at http://youtu.be/PGaXiLZD2KQ

Sensors

- 1D/2D/3D range finders
  - Sharp IR range finder
  - Hokuyo laser scanners
  - Sick lasers
  - Microsoft Kinect
  - Asus Xtion
- Cameras
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
- And many more...

Sensors

- 1D/2D/3D range finders
- Cameras
  - monocular and stereo
  - USB (uvc) and firewire
  - video streaming (gstreamer)
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
- And many more...




Sensors

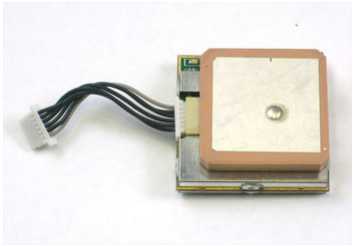
- 1D/2D/3D range finders
- Cameras
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
- And many more...

Sensors

- 1D/2D/3D range finders
- Cameras
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
- And many more...

Sensors

- 1D/2D/3D range finders
- Cameras
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
- And many more...



Sensors

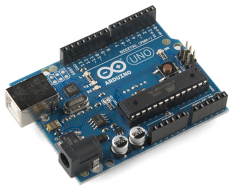
- 1D/2D/3D range finders
- Cameras
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
- And many more. . .

Sensors

- 1D/2D/3D range finders
- Cameras
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
- And many more. . .

Sensors

- 1D/2D/3D range finders
- Cameras
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
  - Dynamixel
  - Phidgets
  - Arduino
  - Arbotix
  - Lego NXT
- And many more...

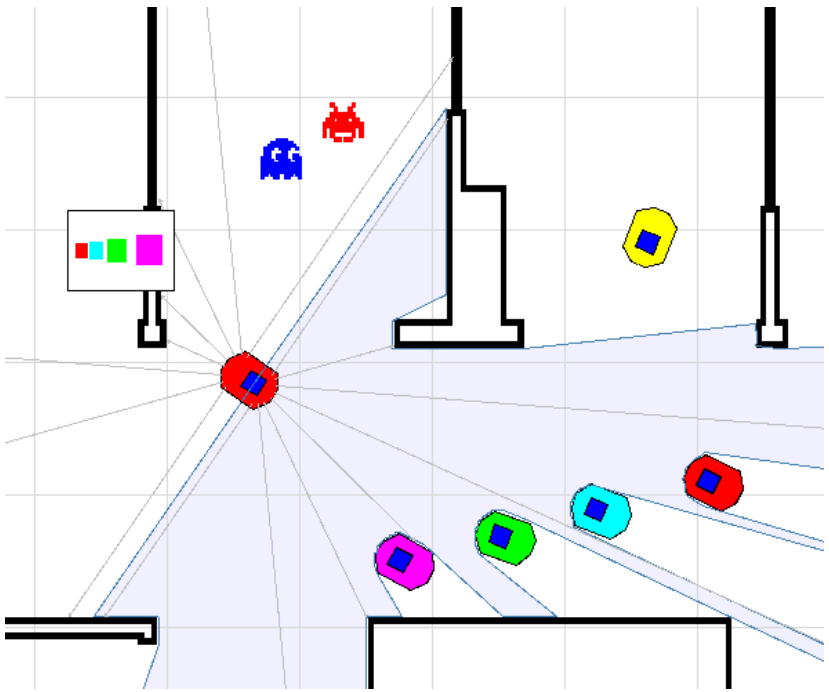


Sensors

- 1D/2D/3D range finders
- Cameras
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
- And many more...

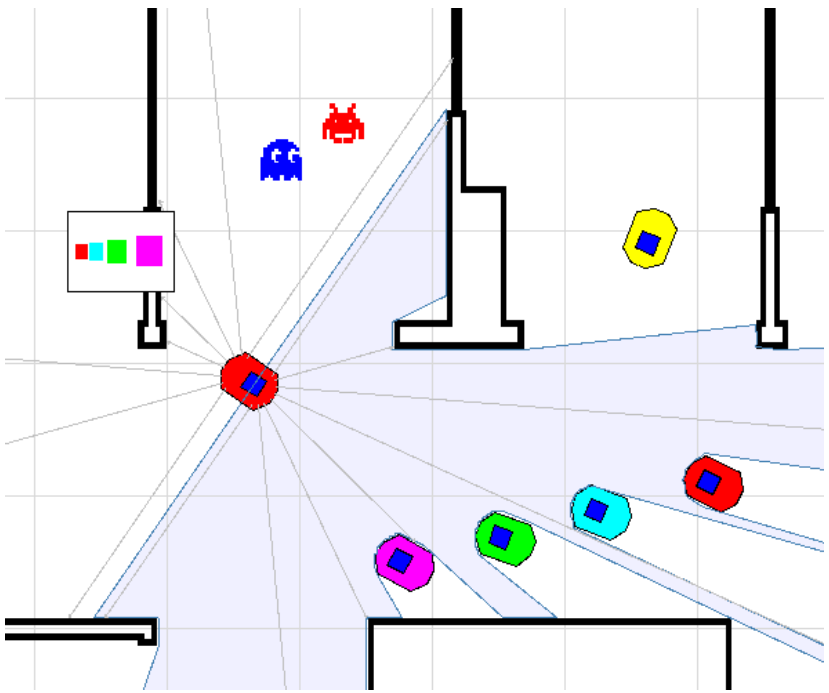
Simulators – Stage

- Stage is a 2D simulator for multiple (large scale) mobile robots



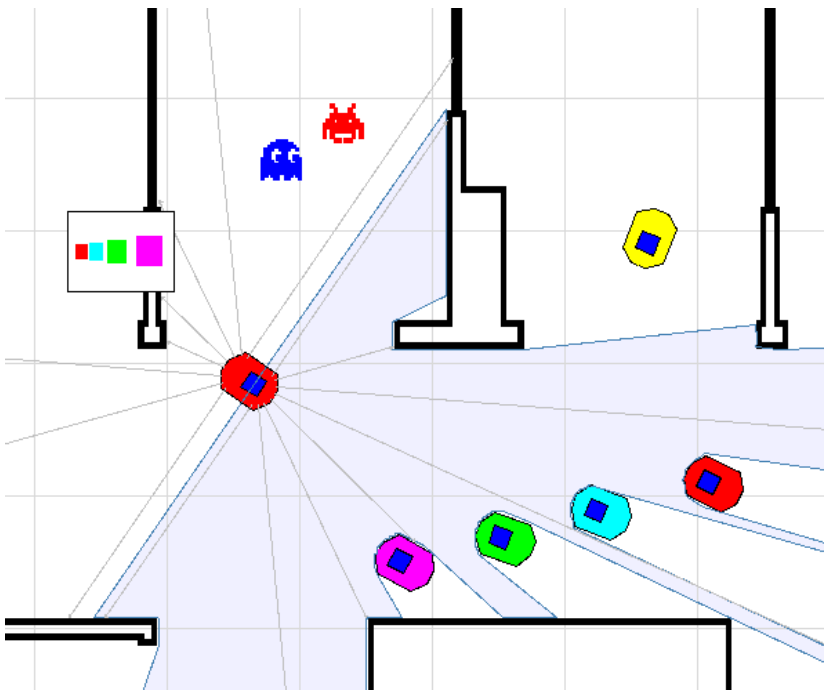
Simulators – Stage

- Stage is a 2D simulator for multiple (large scale) mobile robots
- Models for sensors (e.g., laser, sonar) and actuators (e.g., gripper)



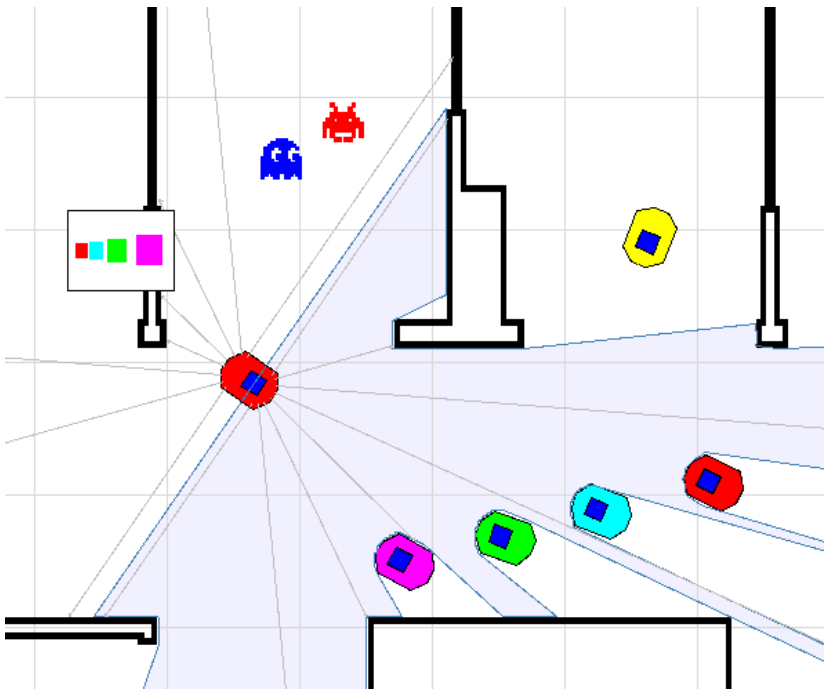
Simulators – Stage

- Stage is a 2D simulator for multiple (large scale) mobile robots
- Models for sensors (e.g., laser, sonar) and actuators (e.g., gripper)
- Models of simple objects for (limited) manipulation



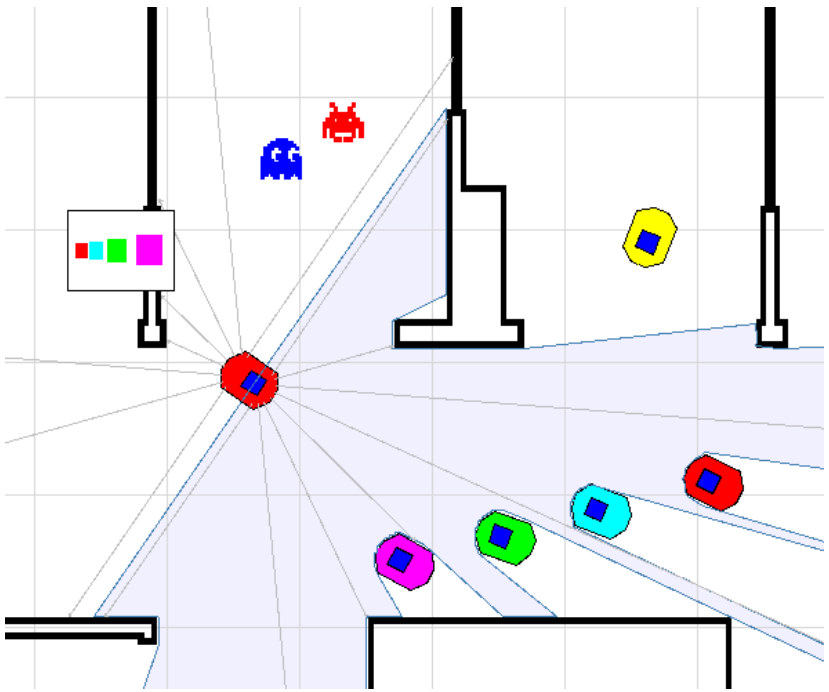
Simulators – Stage

- Stage is a 2D simulator for multiple (large scale) mobile robots
- Models for sensors (e.g., laser, sonar) and actuators (e.g., gripper)
- Models of simple objects for (limited) manipulation
- No physics model at all (e.g., friction, collision, and so forth)



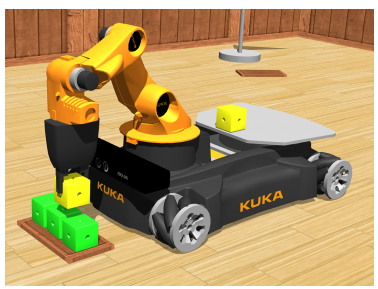
Simulators – Stage

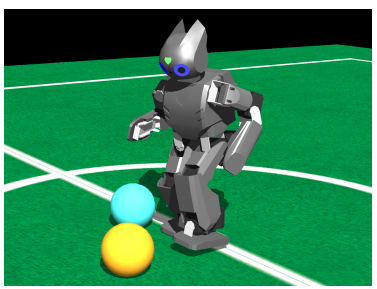
- Stage is a 2D simulator for multiple (large scale) mobile robots
- Models for sensors (e.g., laser, sonar) and actuators (e.g., gripper)
- Models of simple objects for (limited) manipulation
- No physics model at all (e.g., friction, collision, and so forth)
- Open source project



Simulators – Webots

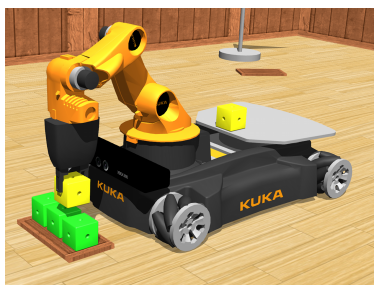
- Development environment used to program and simulate robots

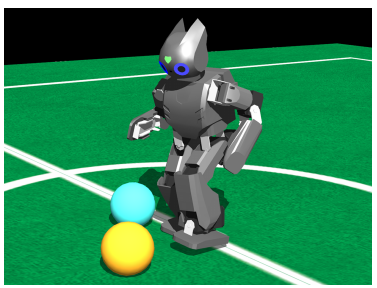




Simulators – Webots

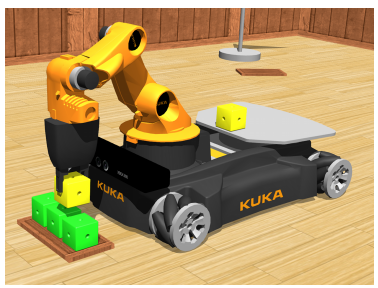
- Development environment used to program and simulate robots
- Complex and realistic 3D simulation of physics/dynamics

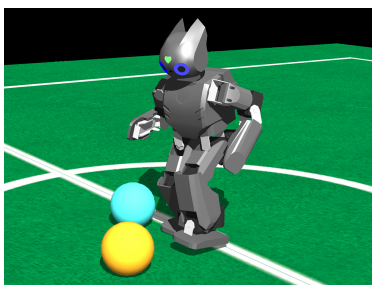




Simulators – Webots

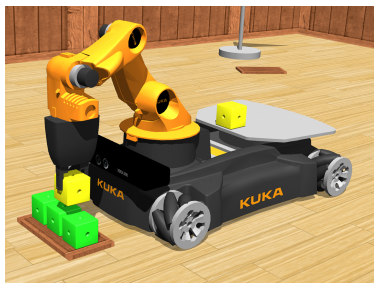
- Development environment used to program and simulate robots
- Complex and realistic 3D simulation of physics/dynamics
- Large collection of robot, sensor and actuator models

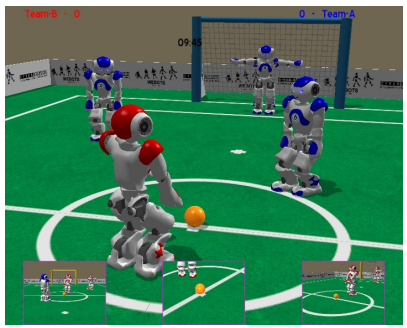


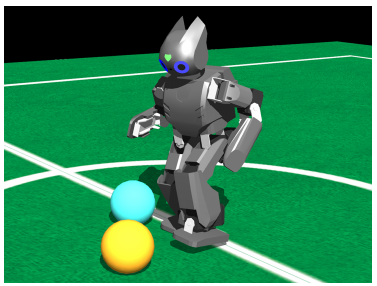


Simulators – Webots

- Development environment used to program and simulate robots
- Complex and realistic 3D simulation of physics/dynamics
- Large collection of robot, sensor and actuator models
- Library of indoor and outdoor objects

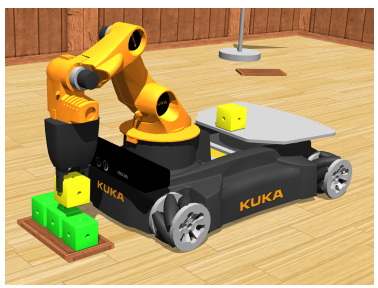


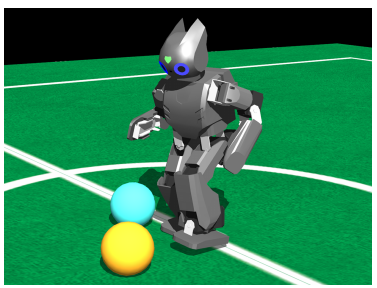




Simulators – Webots

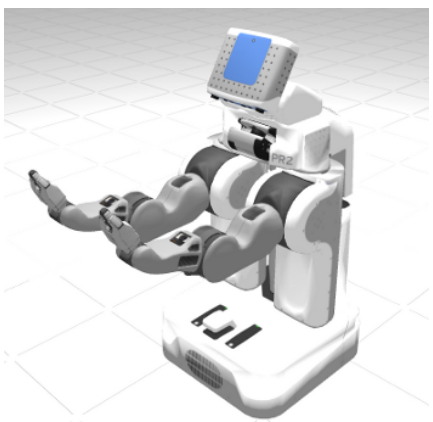
- Development environment used to program and simulate robots
- Complex and realistic 3D simulation of physics/dynamics
- Large collection of robot, sensor and actuator models
- Library of indoor and outdoor objects
- Cross-platform, but proprietary (and fairly expensive)

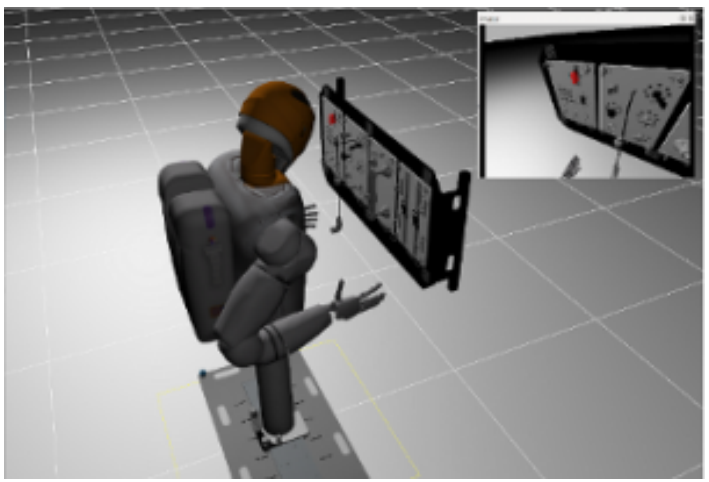




Simulators – Gazebo

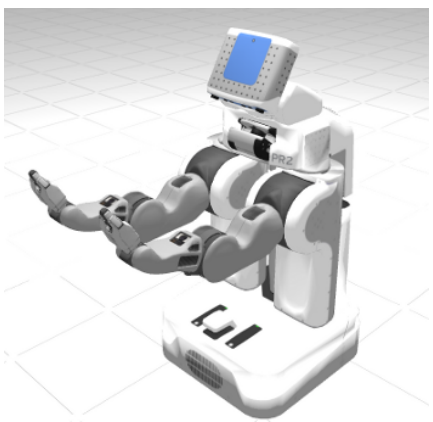
- Gazebo is a 3D simulator of multiple robots in realistic environments

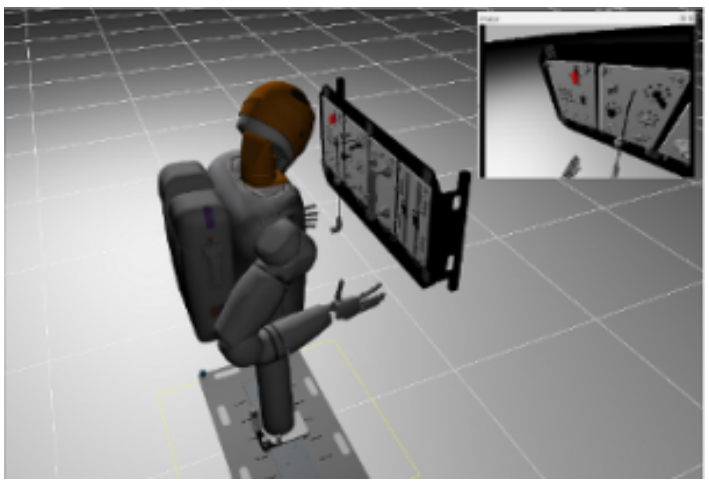




Simulators – Gazebo

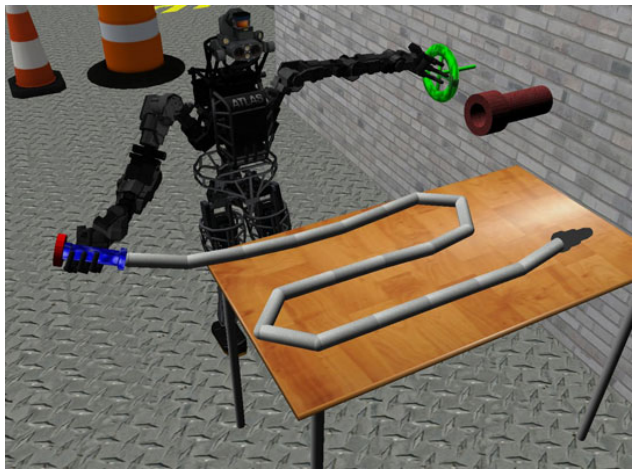
- Gazebo is a 3D simulator of multiple robots in realistic environments
- Realistic simulation of rigid body physics/dynamics (ODE/Bullet)

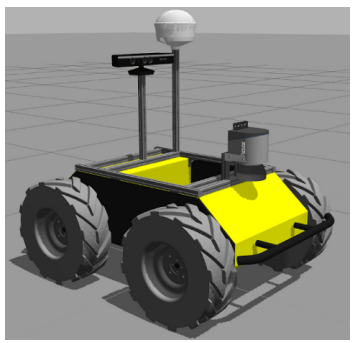




Simulators – Gazebo

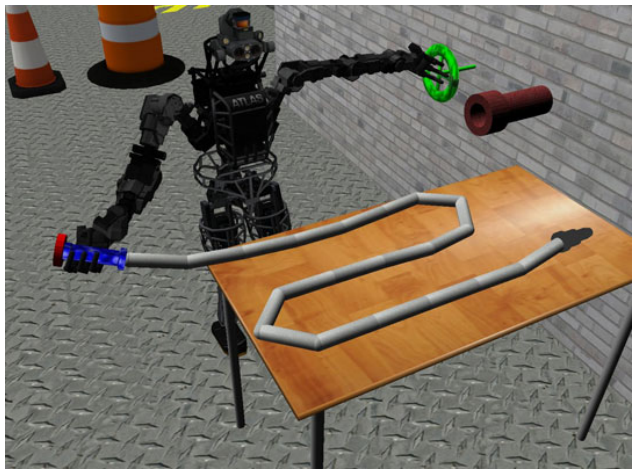
- Gazebo is a 3D simulator of multiple robots in realistic environments
- Realistic simulation of rigid body physics/dynamics (ODE/Bullet)
- Models for complex robots, actuators and sensors (cameras, IMU)

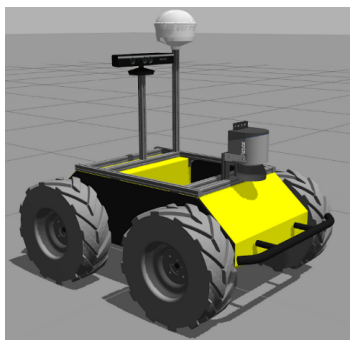




Simulators – Gazebo

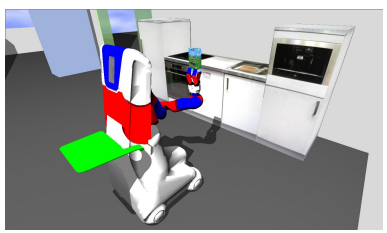
- Gazebo is a 3D simulator of multiple robots in realistic environments
- Realistic simulation of rigid body physics/dynamics (ODE/Bullet)
- Models for complex robots, actuators and sensors (cameras, IMU)
- Developed by Open Source Robotics Foundation (OSRF)

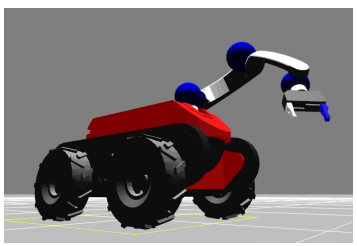




Simulators – Gazebo

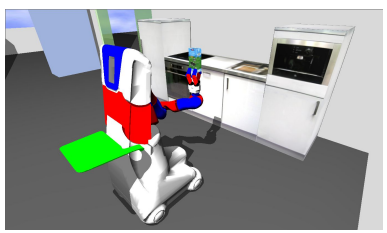
- Gazebo is a 3D simulator of multiple robots in realistic environments
- Realistic simulation of rigid body physics/dynamics (ODE/Bullet)
- Models for complex robots, actuators and sensors (cameras, IMU)
- Developed by Open Source Robotics Foundation (OSRF)
- Official simulator of DARPA Robotics Challenge

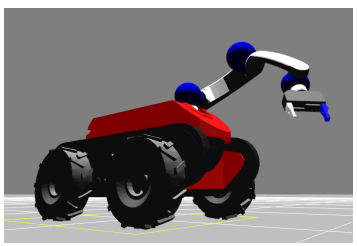




Simulators – Gazebo

- Gazebo is a 3D simulator of multiple robots in realistic environments
- Realistic simulation of rigid body physics/dynamics (ODE/Bullet)
- Models for complex robots, actuators and sensors (cameras, IMU)
- Developed by Open Source Robotics Foundation (OSRF)
- Official simulator of DARPA Robotics Challenge
- Open source project





Installation – ROS (Hydro) on Ubuntu 12.04 (Precise)

1. Configure Ubuntu repositories
2. Setup sources.list
3. Setup keys
4. Install ROS packages and standalone tools
5. Initialise rosdep
6. Setup environment (shell)

Installation – ROS (Hydro) on Ubuntu 12.04 (Precise)

1. Configure Ubuntu repositories
   - Allow "restricted", "universe" and multiverse" to be used
2. Setup sources.list
3. Setup keys
4. Install ROS packages and standalone tools
5. Initialise rosdep
6. Setup environment (shell)

Installation – ROS (Hydro) on Ubuntu 12.04 (Precise)

1. Configure Ubuntu repositories
2. Setup sources.list

```
murilo@muhrix: ~
murilo@muhrix:~$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'
```

3. Setup keys
4. Install ROS packages and standalone tools
5. Initialise rosdep
6. Setup environment (shell)

Installation – ROS (Hydro) on Ubuntu 12.04 (Precise)

1. Configure Ubuntu repositories
2. Setup sources.list
3. Setup keys

```
murilo@muhrix: ~
murilo@muhrix:~$ wget http://packages.ros.org -O - | sudo apt-key add -
```

4. Install ROS packages and standalone tools
5. Initialise rosdep
6. Setup environment (shell)

Installation – ROS (Hydro) on Ubuntu 12.04 (Precise)

1. Configure Ubuntu repositories
2. Setup sources.list
3. Setup keys
4. Install ROS packages and standalone tools

```
murilo@muhrix: ~
murilo@muhrix:~$ sudo apt-get install ros-hydro-desktop-full
```

```
murilo@muhrix: ~
murilo@muhrix:~$ sudo apt-get install python-rosinstall python-rosdep
```

5. Initialise rosdep
6. Setup environment (shell)

Installation – ROS (Hydro) on Ubuntu 12.04 (Precise)

1. Configure Ubuntu repositories
2. Setup sources.list
3. Setup keys
4. Install ROS packages and standalone tools
5. Initialise rosdep

```
murilo@muhrix: ~
murilo@muhrix:~$ sudo rosdep init
```

```
murilo@muhrix: ~
murilo@muhrix:~$ rosdep update
```

6. Setup environment (shell)

Installation – ROS (Hydro) on Ubuntu 12.04 (Precise)

1. Configure Ubuntu repositories
2. Setup sources.list
3. Setup keys
4. Install ROS packages and standalone tools
5. Initialise rosdep
6. Setup environment (shell)

```
murilo@muhrix: ~
murilo@muhrix:~$ echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
```

```
murilo@muhrix: ~
murilo@muhrix:~$ source ~/.bashrc
```

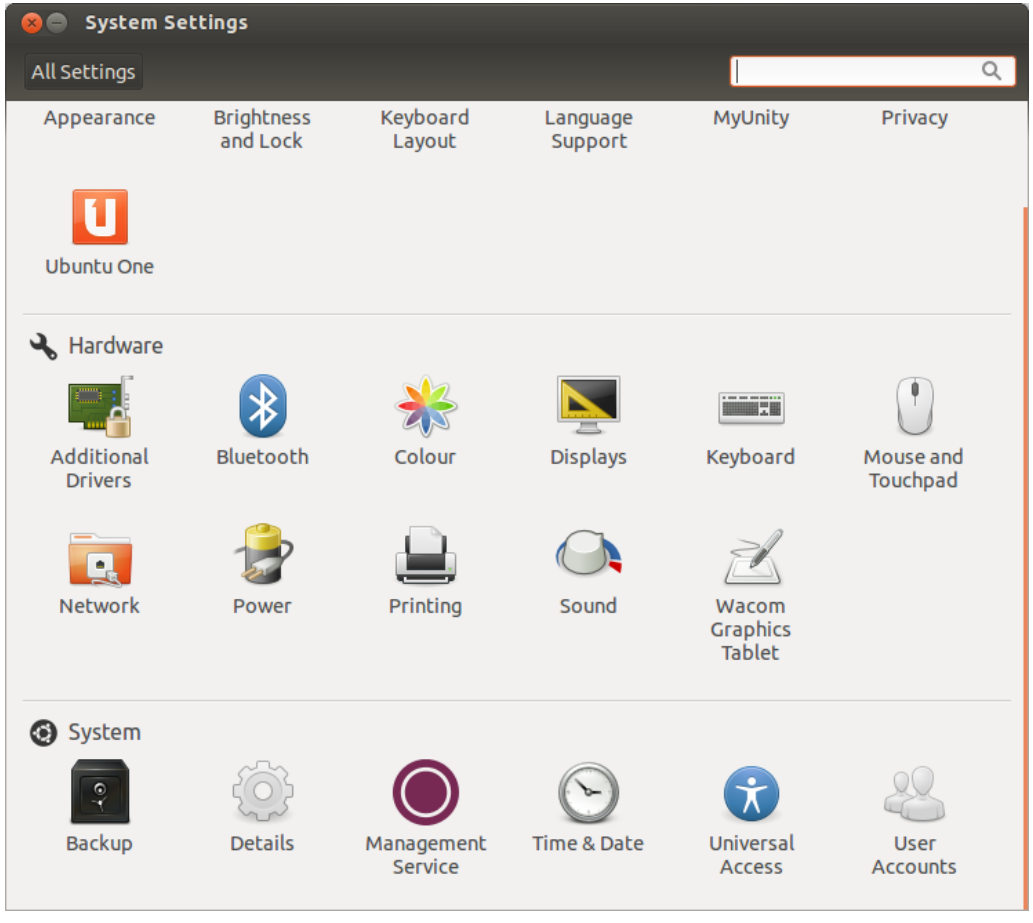
Configuring the Virtual Machine

Loading the VM and logging in to Ubuntu

1. Locate the VM (inside C:\temp folder)
2. Open the VM with VMPlayer (there should be a desktop icon)
3. Wait until it loads. . . and switch to full screen mode
4. Log in with username *rosuser* and password *rosfei*

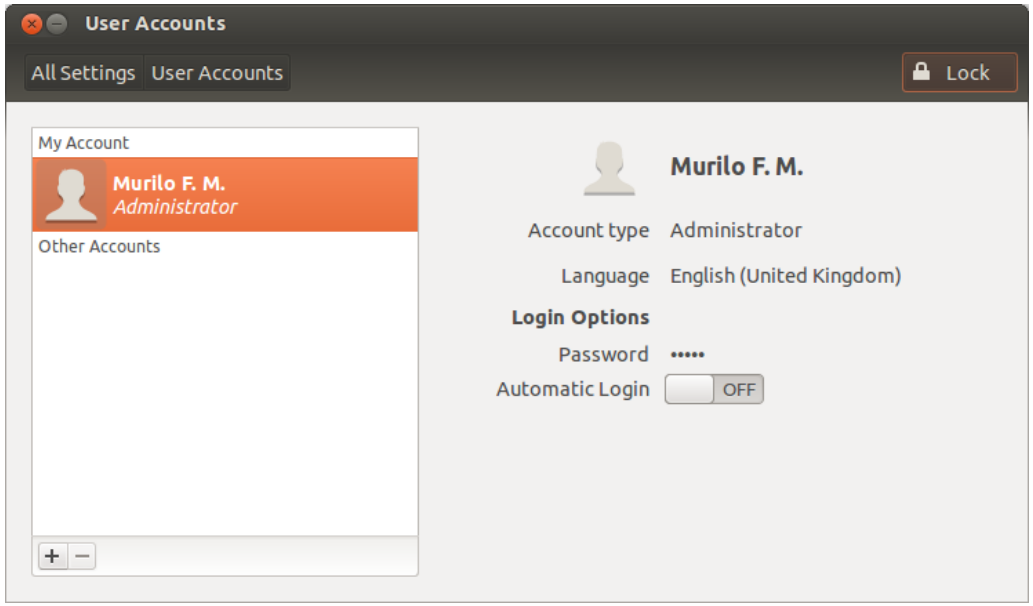
Creating a new user within the VM

1. Open System Settings



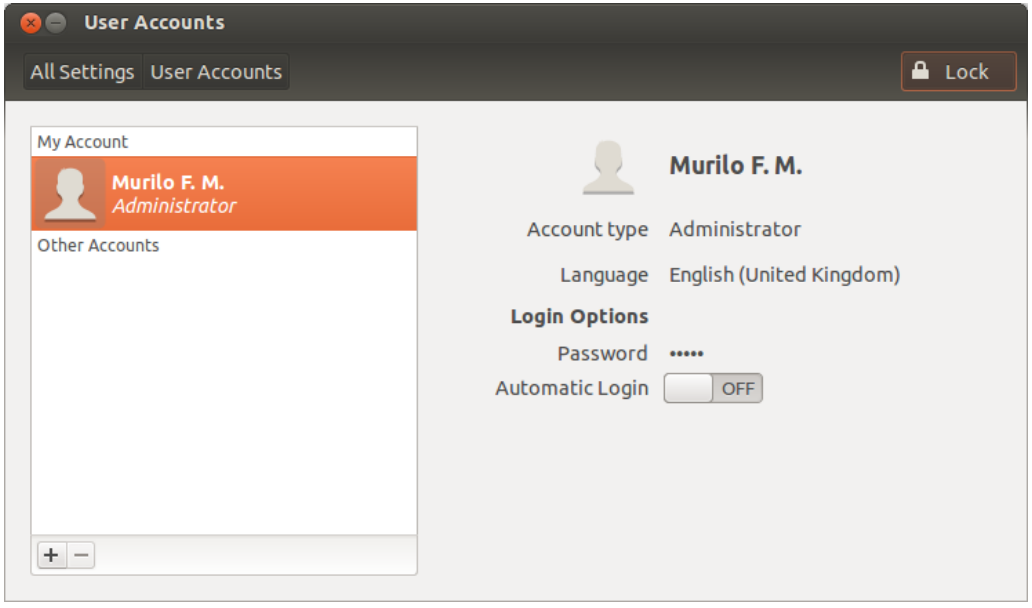
Creating a new user within the VM

1. Open System Settings
2. Open User Accounts



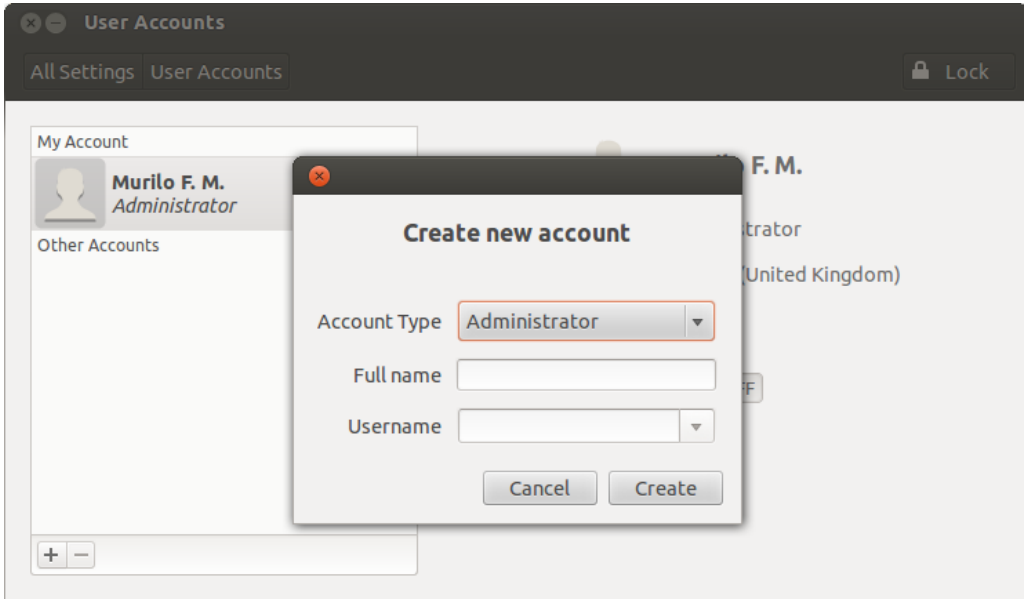
Creating a new user within the VM

1. Open System Settings
2. Open User Accounts
3. Unlock dialog window



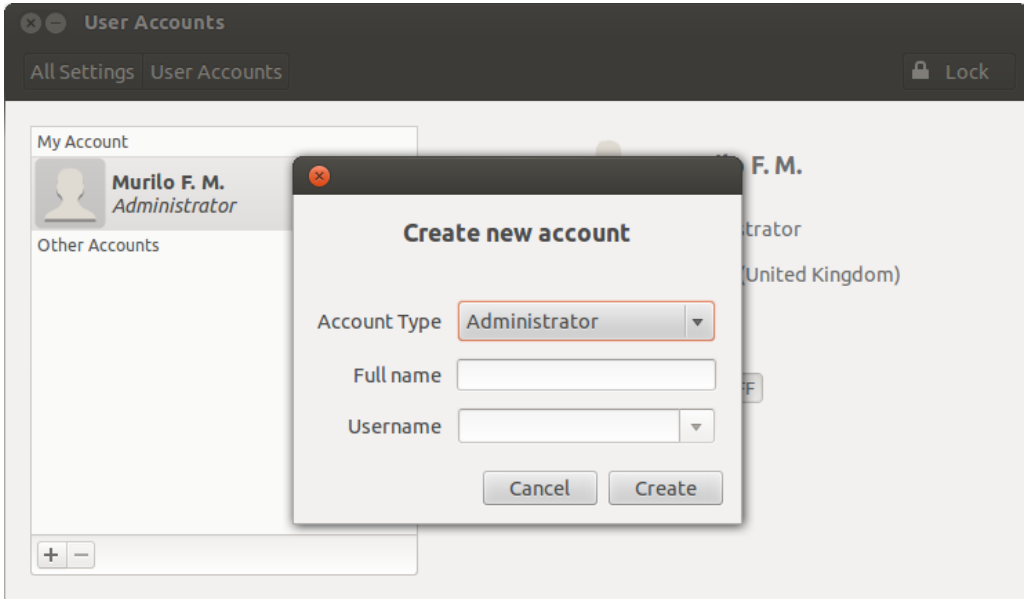
Creating a new user within the VM

1. Open System Settings
2. Open User Accounts
3. Unlock dialog window
4. Add a new user



Creating a new user within the VM

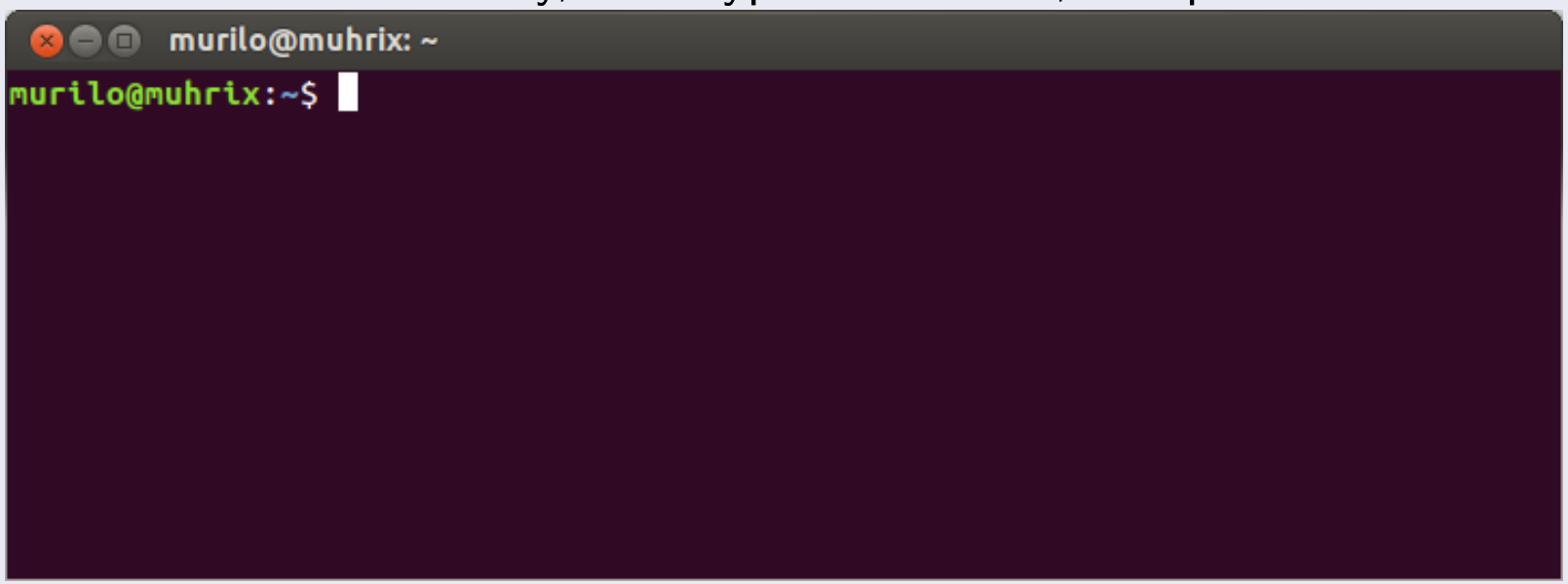
1. Open System Settings
2. Open User Accounts
3. Unlock dialog window
4. Add a new user
5. Set a password!



Setting up ROS environment for the new user

Open up a terminal

Press “Windows” key, then type “terminal”, and press “Enter”



Setting up ROS environment for the new user

Type in the following commands

Remember that spaces are necessary, and Linux is case sensitive!

Configure shell environment with existing ROS installation

```
murilo@muhrix: ~
murilo@muhrix:~$ export | grep ROS
murilo@muhrix:~$ echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
murilo@muhrix:~$ source ~/.bashrc
murilo@muhrix:~$ export | grep ROS
declare -x ROSLISP_PACKAGE_DIRECTORIES=""
declare -x ROS_DISTRO="hydro"
declare -x ROS_ETC_DIR="/opt/ros/hydro/etc/ros"
declare -x ROS_MASTER_URI="http://localhost:11311"
declare -x ROS_PACKAGE_PATH="/opt/ros/hydro/share:/opt/ros/hydro/stacks"
declare -x ROS_ROOT="/opt/ros/hydro/share/ros"
murilo@muhrix:~$
```

Setting up ROS environment for the new user

Create a catkin workspace

```
murilo@muhrix: ~/catkin_ws/src
murilo@muhrix:~$ mkdir -p catkin_ws/src
murilo@muhrix:~$ cd ~/catkin_ws/src/
murilo@muhrix:~/catkin_ws/src$ catkin_init_workspace
```

Build the catkin workspace

```
murilo@muhrix: ~/catkin_ws
murilo@muhrix:~/catkin_ws/src$ cd ~/catkin_ws/
murilo@muhrix:~/catkin_ws$ catkin_make
```

Setting up ROS environment for the new user

Source the catkin workspace and verify environment variables

```
murilo@muhrix: ~
murilo@muhrix:~/catkin_ws$ source devel/setup.bash
murilo@muhrix:~/catkin_ws$ cd
murilo@muhrix:~$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
murilo@muhrix:~$ export | grep ROS
declare -x ROSLISP_PACKAGE_DIRECTORIES="/home/murilo/catkin_ws/devel/share/commo
n-lisp"
declare -x ROS_DISTRO="hydro"
declare -x ROS_ETC_DIR="/opt/ros/hydro/etc/ros"
declare -x ROS_MASTER_URI="http://localhost:11311"
declare -x ROS_PACKAGE_PATH="/home/murilo/catkin_ws/src:/opt/ros/hydro/share:/op
t/ros/hydro/stacks"
declare -x ROS_ROOT="/opt/ros/hydro/share/ros"
declare -x ROS_TEST_RESULTS_DIR="/home/murilo/catkin_ws/build/test_results"
murilo@muhrix:~$
```

Setting up ROS environment for the new user

The commands altogether...

Careful! Do not try "Copy & Paste", it will not work!

```
1   $ echo ''source /opt/ros/hydro/setup.bash'' >> ~/.bashrc
2   $ source ~/.bashrc
3
4   $ export | grep ROS
5
6   $ mkdir -p ~/catkin_ws/src
7   $ cd ~/catkin_ws/src
8   $ catkin_init_workspace
9
10  $ cd ~/catkin_ws/
11  $ catkin_make
12  $ source devel/setup.bash
13
14  $ echo ''source ~/catkin_ws/devel/setup.bash'' >> ~/.bashrc
15
16  $ export | grep ROS
```

Setting up ROS environment for the new user

Understanding the commands (line by line)

1. Add the command within double quotes to the end of the ∼/.bashrc file
2. Source (reload) the settings in ∼/.bashrc in the current terminal
3.
4. Output environment variables containing "ROS" to terminal
5.
6. Create cascaded directories catkin_ws/ and src/ within user home folder
7. Change directory to ∼/catkin_ws/src/
8. Initialise catkin workspace
9.
10. Change directory to ∼/catkin_ws/
11. Build catkin workspace (i.e., the ROS packages within src/ folder)
12. Akin to line 2
13.
14. Same effect as line 4

ROS filesystem – Overview

ROS packages

- Lowest level of ROS software organisation
- Dedicated to a **single** functionality (e.g., data acquisition from a laser scanner, and publication of laser data messages)
- There can be no more than one package in each folder

ROS filesystem – Overview

ROS packages

- Lowest level of ROS software organisation
- Dedicated to a **single** functionality (e.g., data acquisition from a laser scanner, and publication of laser data messages)
- There can be no more than one package in each folder

Rosbuild (dry) packages (deprecated)

- manifest.xml:
  - include information (e.g., license, author, etc.)
  - define dependencies on other packages
  - provide compiler and linker flags for other dependant *packages*
- Makefile: Make build specification
- CMakeLists.txt: CMake build specification

ROS filesystem – Overview

ROS packages

- Lowest level of ROS software organisation
- Dedicated to a **single** functionality (e.g., data acquisition from a laser scanner, and publication of laser data messages)
- There can be no more than one package in each folder

Catkin (wet) packages (recommended)

- package.xml:
  - include information (e.g., license, author, etc.)
  - define *build* and *run* dependencies on other packages
- CMakeLists.txt: CMake build specification which uses catkin macros
- **Cannot** depend on rosbuild packages

ROS filesystem – Overview

Stacks – rosbuild (deprecated)

- Group packages which collectively provide (a more abstract) functionality
- packages within the stack folder are considered part of the *stack*
- stack.xml: include metadata and declare dependencies on other stacks

ROS filesystem – Overview

Stacks – rosbuild (deprecated)

- Group packages which collectively provide (a more abstract) functionality
- packages within the stack folder are considered part of the *stack*
- stack.xml: include metadata and declare dependencies on other stacks

Metapackages – catkin (recommended)

- Convenient way of grouping packages as a single logical *package*
- package.xml: include metadata, and can only have dependencies on *packages* grouped by the metapackage
- Other packages are required to **not** depend on *metapackages*

ROS filesystem – Package structure

Hypothetical rosbuild package my_pkg/

- CmakeLists.txt: CMake build settings for package my_pkg
- manifest.xml: metadata and dependencies required by package
- mainpage.dox: doc information of package my_pkg
- include/my_pkg: C++ header files
- src/: source code directory
- bin/: compiled binaries directory
- launch/: where launch files should be stored
- msg/: message (.msg) types
- srv/: service (.srv) types
- scripts/: executable scripts

ROS filesystem – Package structure

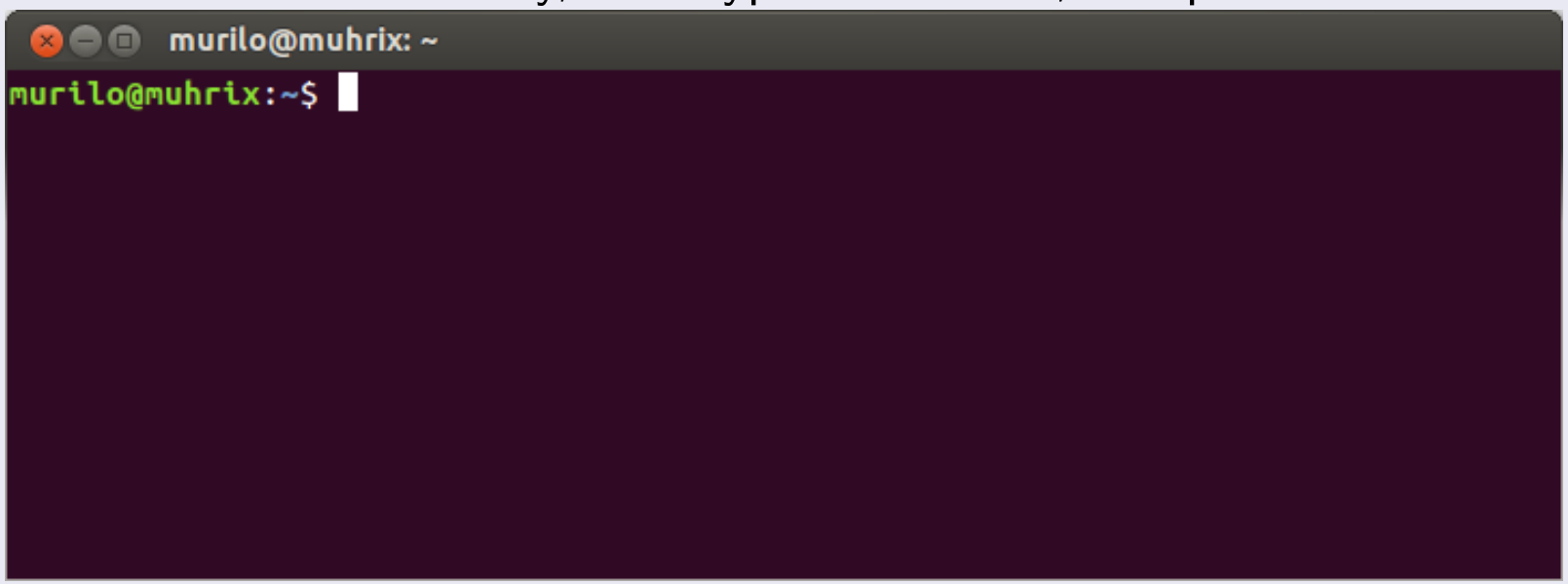
Hypothetical catkin package my_pkg/

- CmakeLists.txt: CMake build settings for package my_pkg
- package.xml: metadata and dependencies required by package
- mainpage.dox: Doxygen information of package my_pkg
- include/my_pkg: C++ header files
- src/: source code directory
- launch/: where launch files should be stored
- msg/: message (.msg) definitions
- srv/: service (.srv) definitions
- scripts/: executable (Python) scripts

rosbash – ROS command line tools

Open up a terminal

Press “Windows” key, then type “terminal”, and press “Enter”



rosbash – ROS command line tools

rospack: ROS package management tool

```
$ rospack list
$ rospack find turtlesim
$ rospack depends turtlesim
$ rospack profile
```

roscd: "change directory" command for ROS

```
$ roscd (change directory to ~/catkin_ws/devel)
$ roscd turtlesim
$ ls (standard linux shell command)
```

rosls: list the contents of a ROS package

```
$ roscd (return to ~/catkin_ws/devel directory)
$ rosls turtlesim
```

Creating a new ROS package

rosed: quickly see/edit a file of a given package

```
$ echo $EDITOR (if blank, default is vi)
$ export EDITOR=gedit
$ rosed turtlesim package.xml
```

Change directory to your workspace and create a new ROS package

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg my_pkg std_msgs rospy roscpp
```

Creating a new ROS package

Locating and examining the newly created package

```
$ rospack profile
$ rospack find my_pkg
$ roscd my_pkg
$ rospack depends1 my_pkg (first-order dependencies)
$ rospack depends (all dependencies)
$ rosed my_pkg package.xml
$ rosed my_pkg CMakeLists.txt
```

Building the package (i.e., building the catkin workspace)

```
$ cd ~/catkin_ws/
$ catkin_make
```

Now examine the output!

Nodes

- Nodes are processes which perform specific computations:
  - control robot wheel motors
  - acquire data from laser scanner
  - acquire images from camera
  - perform localisation
  - compute path planning
  - provide graphical visualisation of the system

Nodes

- Nodes are processes which perform specific computations:
  - control robot wheel motors
  - acquire data from laser scanner
  - acquire images from camera
  - perform localisation
  - compute path planning
  - provide graphical visualisation of the system
- Data is exchanged between nodes via (standardised) ROS messages

Nodes

- Nodes are processes which perform specific computations:
  - control robot wheel motors
  - acquire data from laser scanner
  - acquire images from camera
  - perform localisation
  - compute path planning
  - provide graphical visualisation of the system
- Data is exchanged between nodes via (standardised) ROS messages
- Nodelets – provide a way to run multiple nodes (algorithms):
  - on a single machine
  - in a single process
  - avoiding copy costs of interprocess message passing
  - with applications to cases with high throughput data flow

Master

- Master is the core *node* of ROS, called *roscore*

Master

- Master is the core *node* of ROS, called *roscore*
- Provides name registration and lookup to the rest of the Computation Graph (akin to an internet DNS server)

Master

- Master is the core *node* of ROS, called *roscore*
- Provides name registration and lookup to the rest of the Computation Graph (akin to an internet DNS server)
- Upon start-up, nodes register with Master and fetch registration information (e.g., publishers/subscribers of a given *topic*)

Master

- Master is the core *node* of ROS, called *roscore*
- Provides name registration and lookup to the rest of the Computation Graph (akin to an internet DNS server)
- Upon start-up, nodes register with Master and fetch registration information (e.g., publishers/subscribers of a given *topic*)
- *Nodes* then establish direct connections as appropriate

Master

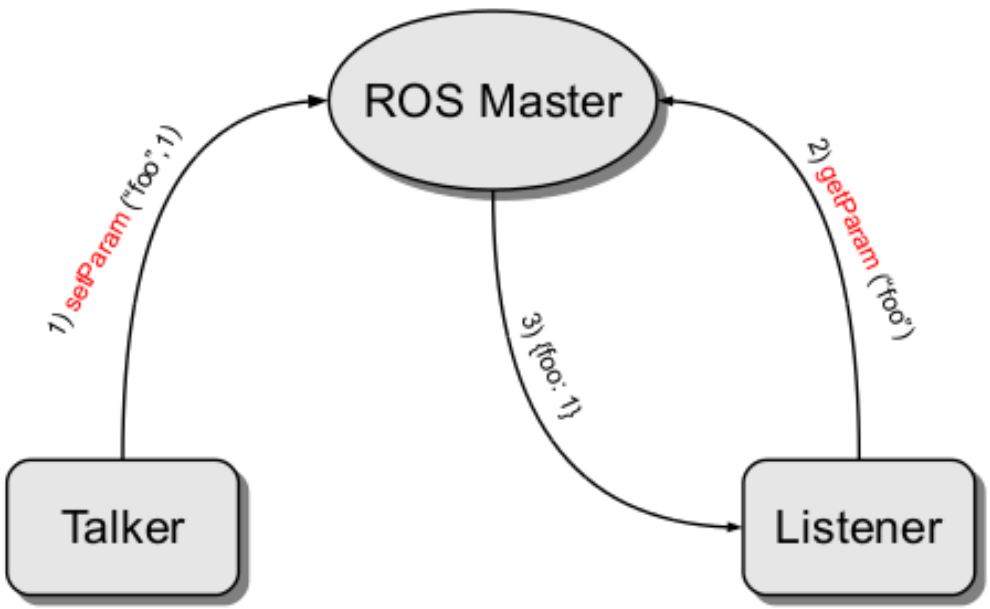
- Master is the core *node* of ROS, called *roscore*
- Provides name registration and lookup to the rest of the Computation Graph (akin to an internet DNS server)
- Upon start-up, nodes register with Master and fetch registration information (e.g., publishers/subscribers of a given *topic*)
- *Nodes* then establish direct connections as appropriate
- Also makes callbacks to nodes when registration information changes

Master

- Master is the core *node* of ROS, called *roscore*
- Provides name registration and lookup to the rest of the Computation Graph (akin to an internet DNS server)
- Upon start-up, nodes register with Master and fetch registration information (e.g., publishers/subscribers of a given *topic*)
- *Nodes* then establish direct connections as appropriate
- Also makes callbacks to nodes when registration information changes
- Allows nodes to dynamically create connections as new nodes are run

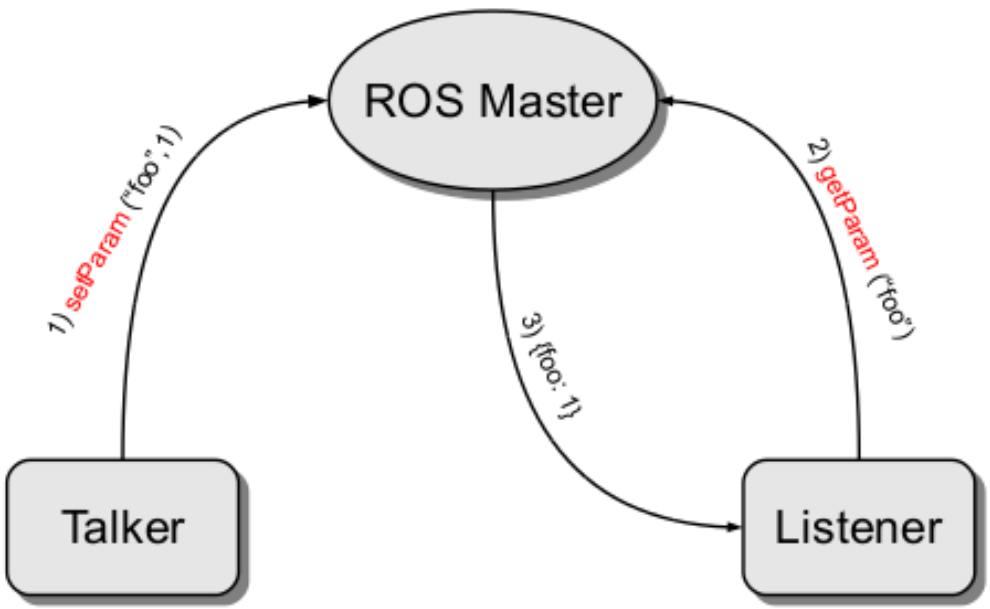
Parameter server

- Shared, multi-variate dictionary which is accessible via network APIs



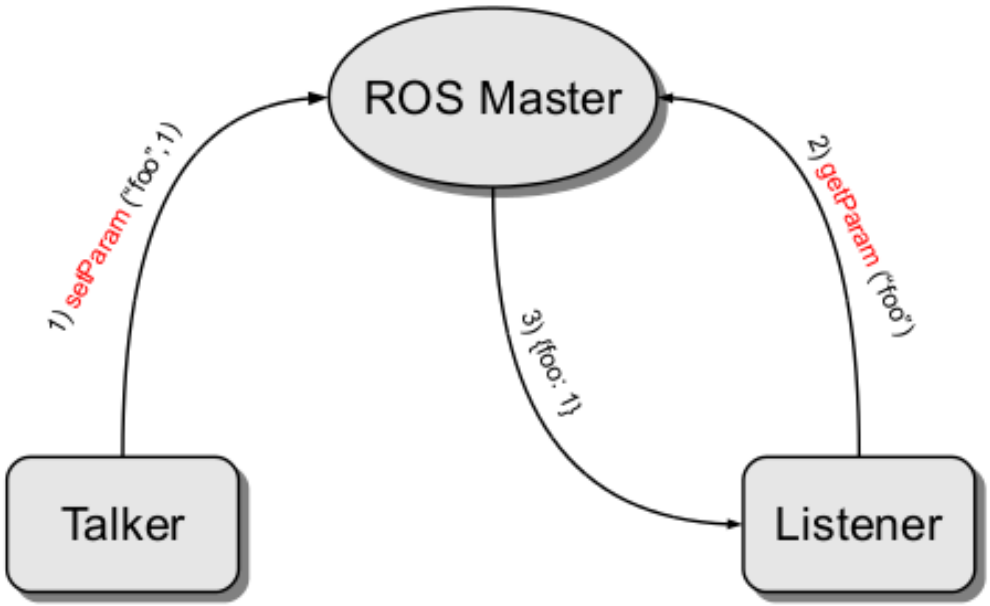
Parameter server

- Shared, multi-variate dictionary which is accessible via network APIs
- Part of ROS Master



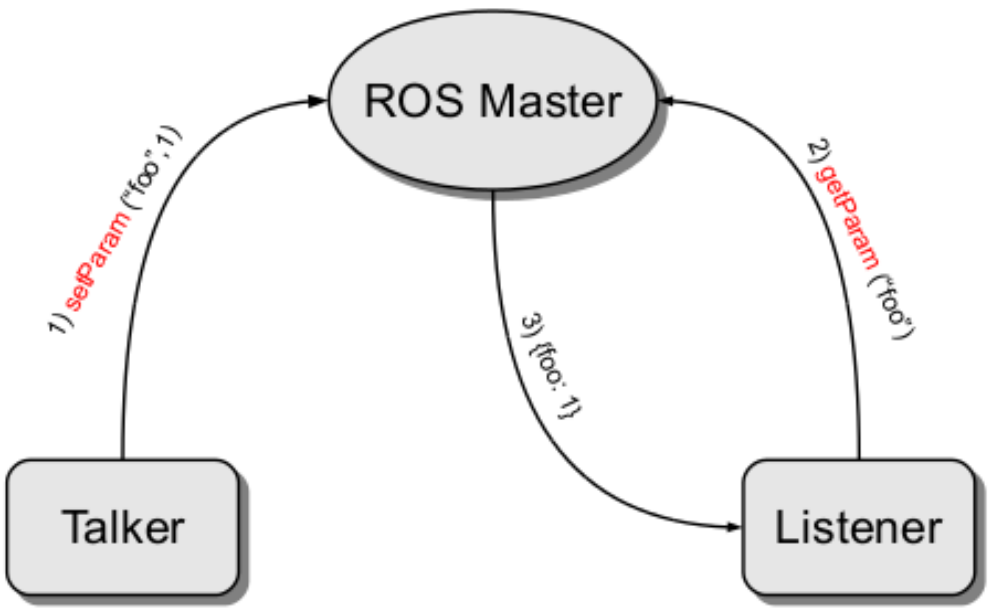
Parameter server

- Shared, multi-variate dictionary which is accessible via network APIs
- Part of ROS Master
- Nodes use this server to store and retrieve parameters at runtime



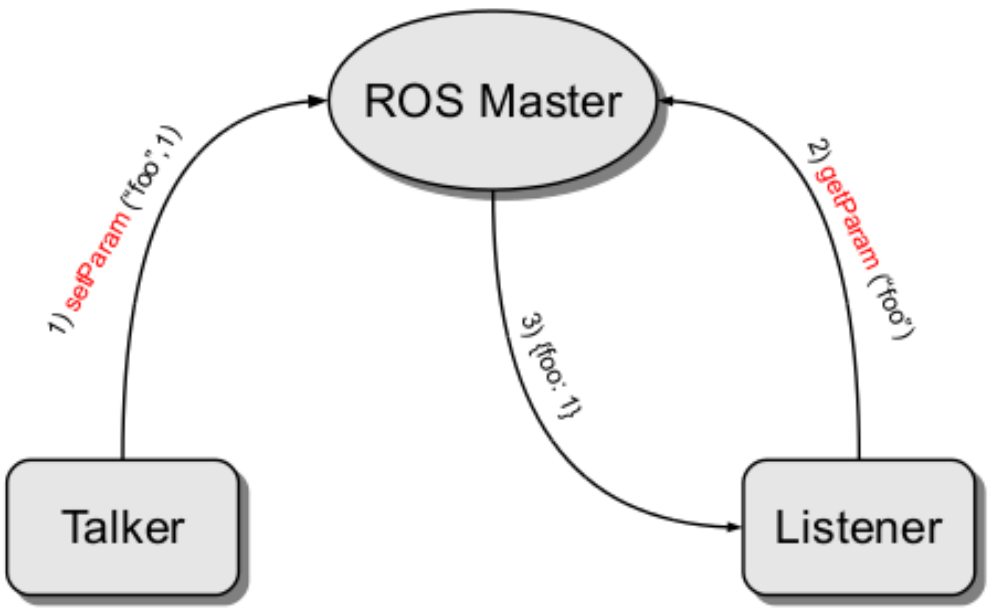
Parameter server

- Shared, multi-variate dictionary which is accessible via network APIs
- Part of ROS Master
- Nodes use this server to store and retrieve parameters at runtime
- Not designed for high performance, and hence. . .



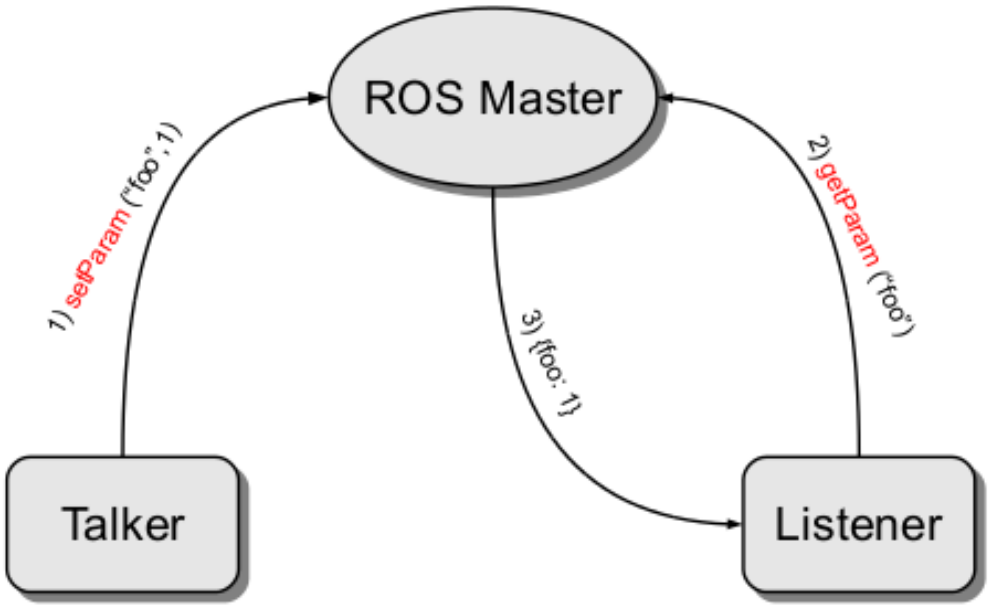
Parameter server

- Shared, multi-variate dictionary which is accessible via network APIs
- Part of ROS Master
- Nodes use this server to store and retrieve parameters at runtime
- Not designed for high performance, and hence. . .
- Better suited for configuration parameters



Parameter server

- Shared, multi-variate dictionary which is accessible via network APIs
- Part of ROS Master
- Nodes use this server to store and retrieve parameters at runtime
- Not designed for high performance, and hence...
- Better suited for configuration parameters
- Follows ROS naming convention, having a hierarchy meant to protect parameters from conflicting (namespaces)



Parameter server

Open up a terminal, then run ROS Master node

```
1    roscore
```

In another terminal, explore the parameter server

```
1    $ rosparam list
2    $ rosparam get /rosdistro
3    $ rosparam get /rosversion
```

ROS Messages

- ROS messages are simply a data structure, consisting of typed fields

ROS Messages

- ROS messages are simply a data structure, consisting of typed fields
- Standard primitive types (and nested arrays) are supported:
  - int{8, 16, 32, 64}
  - float{32, 64}
  - string
  - time
  - duration
  - array[]

ROS Messages

- ROS messages are simply a data structure, consisting of typed fields
- Standard primitive types (and nested arrays) are supported:
  - int{8, 16, 32, 64}
  - float{32, 64}
  - string
  - time
  - duration
  - array[]
- Nodes communicate with each other by exchanging messages

ROS Messages

- ROS messages are simply a data structure, consisting of typed fields
- Standard primitive types (and nested arrays) are supported:
  - int{8, 16, 32, 64}
  - float{32, 64}
  - string
  - time
  - duration
  - array[]
- Nodes communicate with each other by exchanging messages
- Routed via a transport system with publish/subscribe semantics

ROS Messages

- ROS messages are simply a data structure, consisting of typed fields
- Standard primitive types (and nested arrays) are supported:
  - int{8, 16, 32, 64}
  - float{32, 64}
  - string
  - time
  - duration
  - array[]
- Nodes communicate with each other by exchanging messages
- Routed via a transport system with publish/subscribe semantics
- When used with topics: *.msg (n:n non-blocking)

ROS Messages

- ROS messages are simply a data structure, consisting of typed fields
- Standard primitive types (and nested arrays) are supported:
  - int{8, 16, 32, 64}
  - float{32, 64}
  - string
  - time
  - duration
  - array[]
- Nodes communicate with each other by exchanging messages
- Routed via a transport system with publish/subscribe semantics
- When used with topics: *.msg (n:n non-blocking)
- When used with services: *.srv (1:1 blocking – request + response)

Topics

- A node sends out a message by publishing it onto a given Topic

Topics

- A node sends out a message by publishing it onto a given Topic
- The Topic type is defined by the message type published on it

Topics

- A node sends out a message by publishing it onto a given Topic
- The Topic type is defined by the message type published on it
- A node requiring a certain type of data must subscribe to the appropriate Topic

Topics

- A node sends out a message by publishing it onto a given Topic
- The Topic type is defined by the message type published on it
- A node requiring a certain type of data must subscribe to the appropriate Topic
- Multiple publishers/subscribers to the same Topic are allowed

Topics

- A node sends out a message by publishing it onto a given Topic
- The Topic type is defined by the message type published on it
- A node requiring a certain type of data must subscribe to the appropriate Topic
- Multiple publishers/subscribers to the same Topic are allowed
- A single node may publish and/or subscribe to multiple Topics

Topics

- A node sends out a message by publishing it onto a given Topic
- The Topic type is defined by the message type published on it
- A node requiring a certain type of data must subscribe to the appropriate Topic
- Multiple publishers/subscribers to the same Topic are allowed
- A single node may publish and/or subscribe to multiple Topics
- Publishers and subscribers are generally unaware of each other's existence

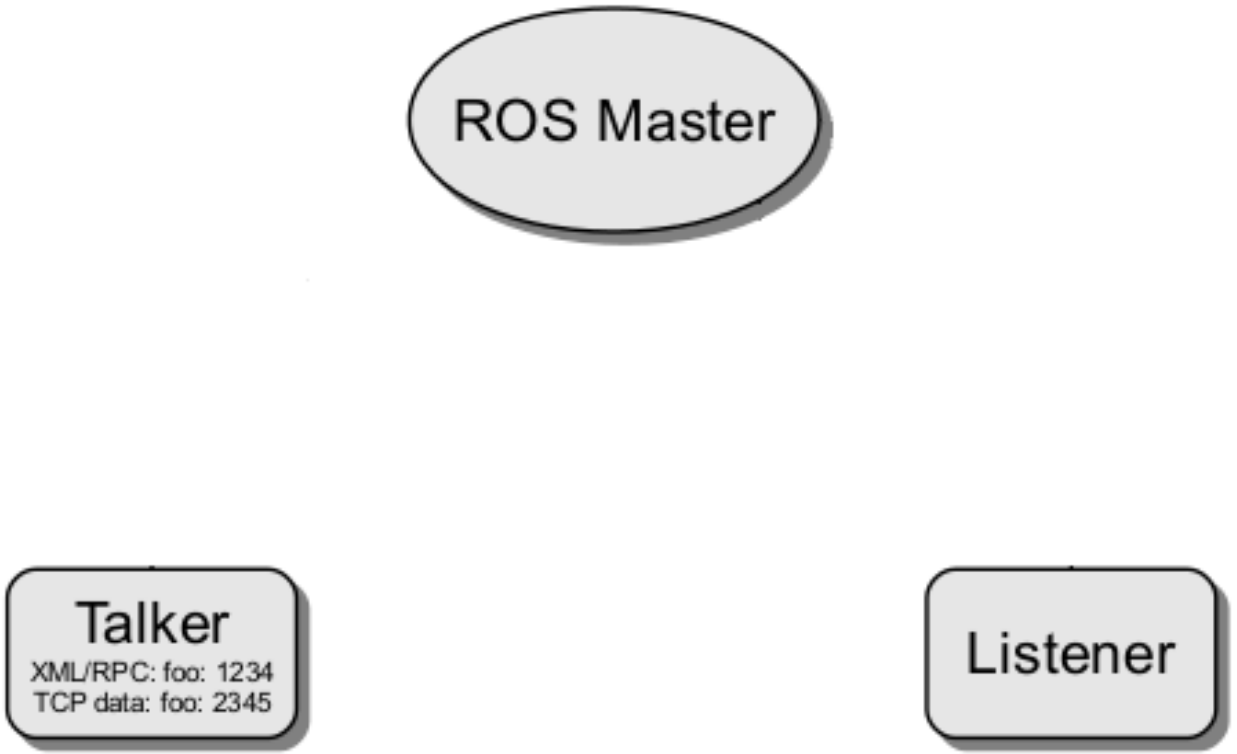
Topics

- A node sends out a message by publishing it onto a given Topic
- The Topic type is defined by the message type published on it
- A node requiring a certain type of data must subscribe to the appropriate Topic
- Multiple publishers/subscribers to the same Topic are allowed
- A single node may publish and/or subscribe to multiple Topics
- Publishers and subscribers are generally unaware of each other's existence
- Publish/subscribe model is a flexible paradigm (many-to-many, one-way transport)

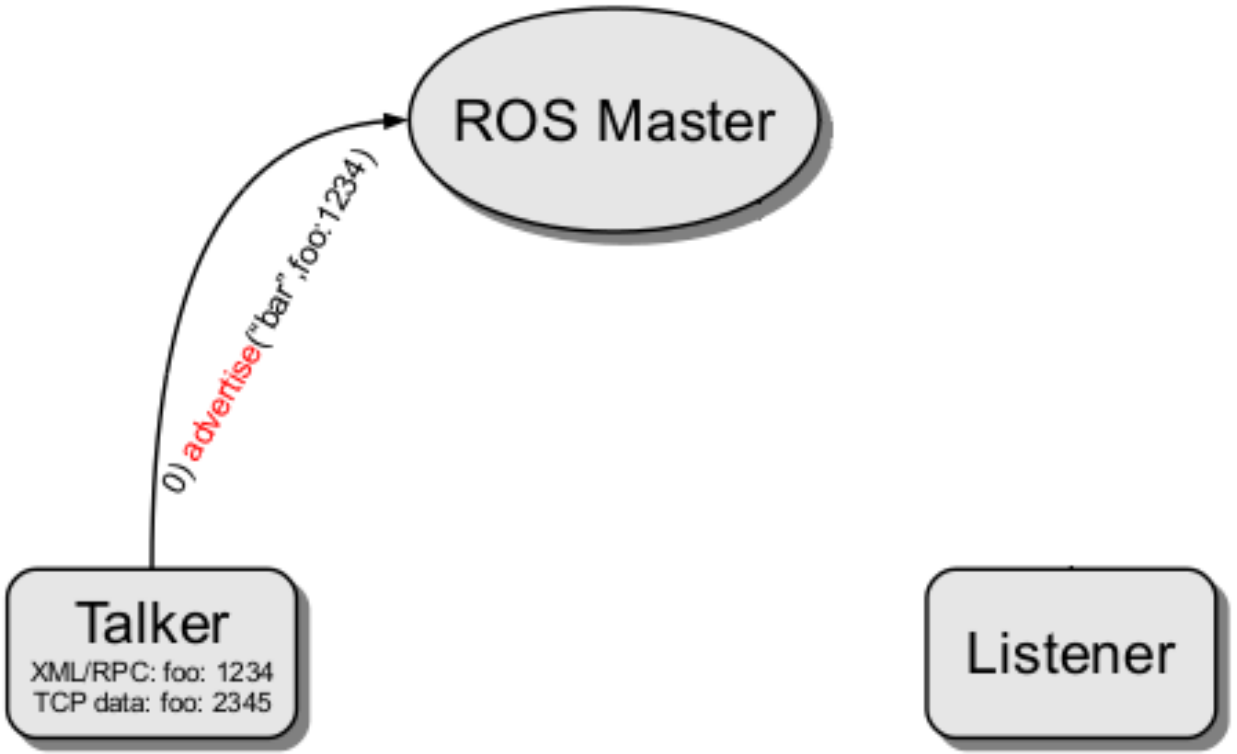
Topics

- A node sends out a message by publishing it onto a given Topic
- The Topic type is defined by the message type published on it
- A node requiring a certain type of data must subscribe to the appropriate Topic
- Multiple publishers/subscribers to the same Topic are allowed
- A single node may publish and/or subscribe to multiple Topics
- Publishers and subscribers are generally unaware of each other's existence
- Publish/subscribe model is a flexible paradigm (many-to-many, one-way transport)
- There is no order of execution required

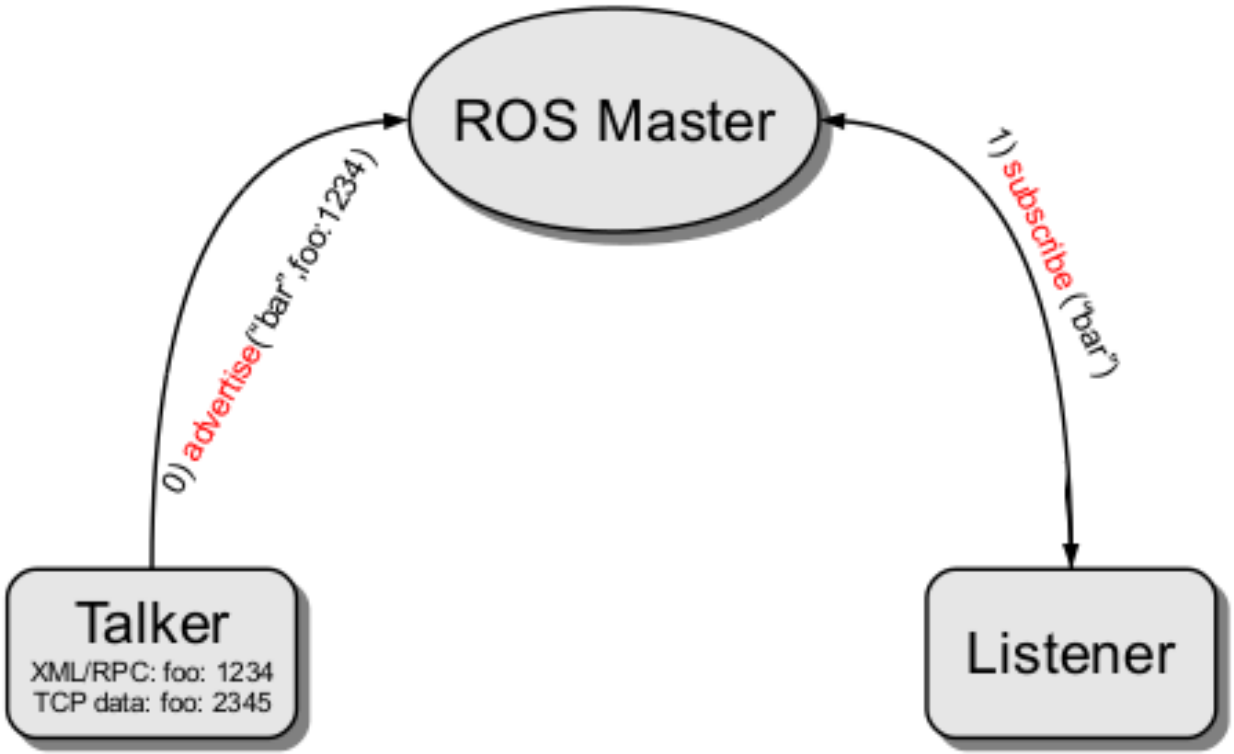
Topics – diagrammatic representation



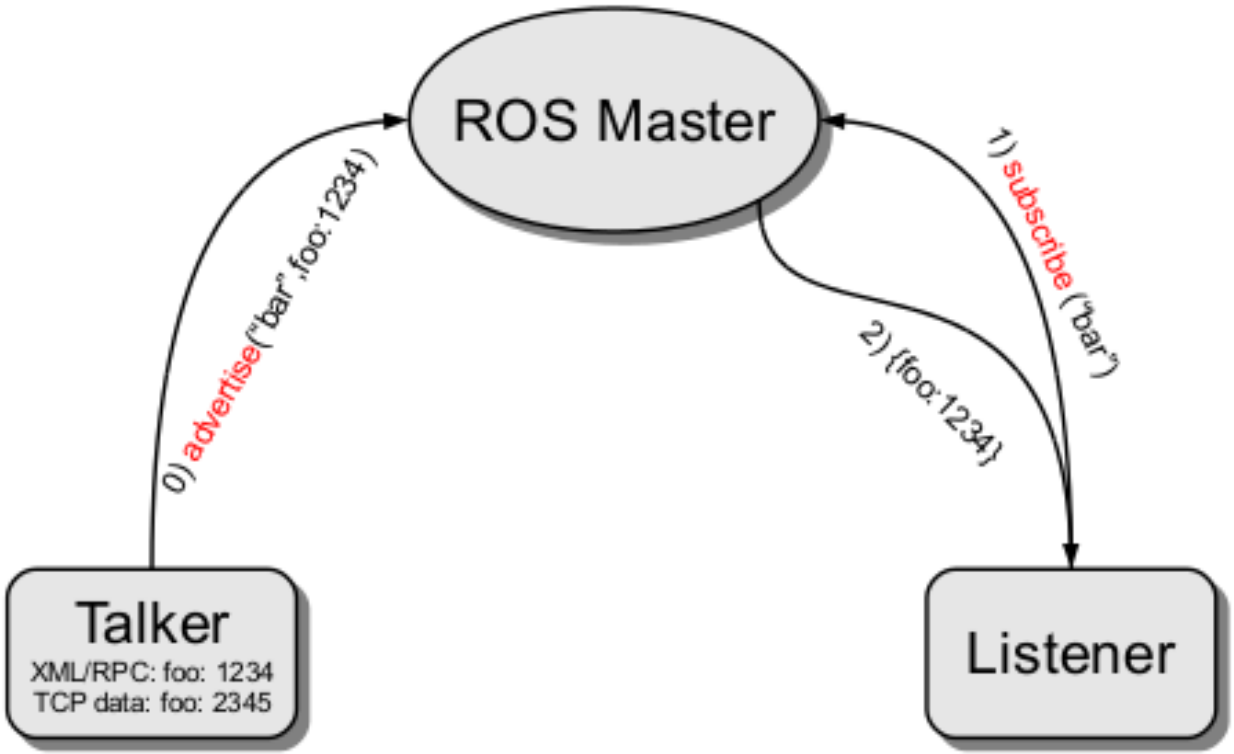
Topics – diagrammatic representation



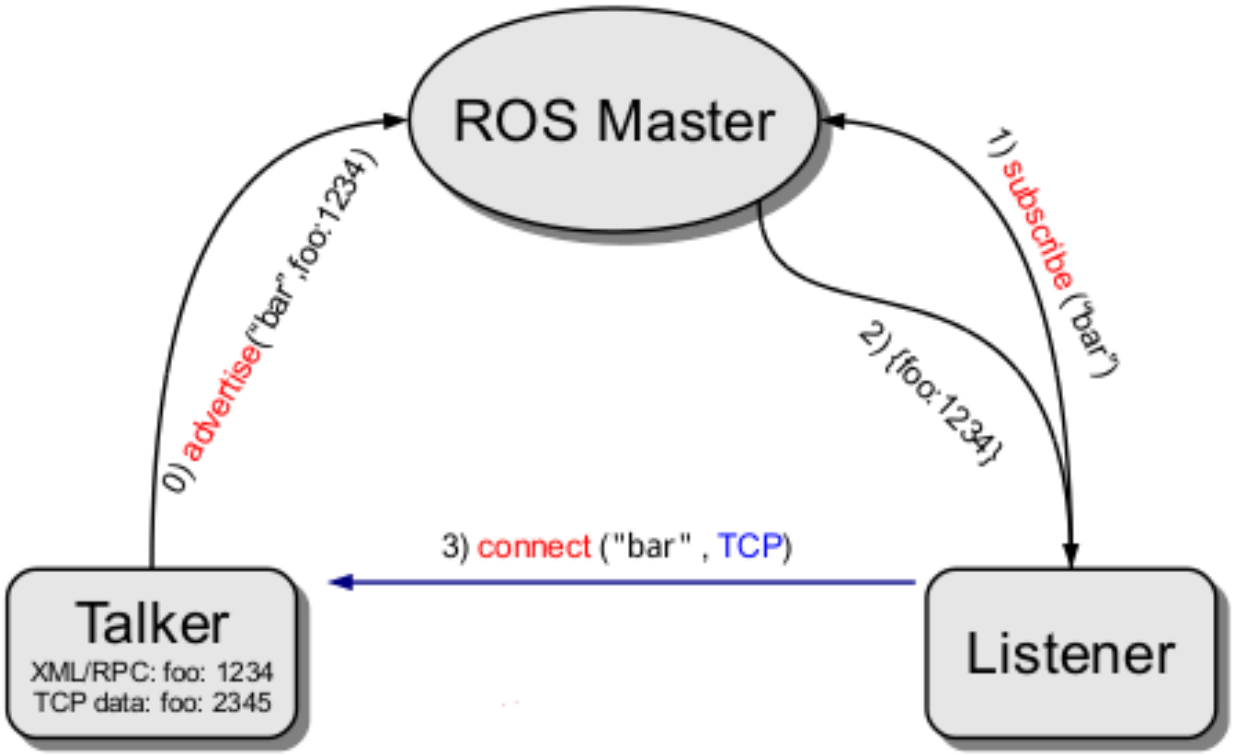
Topics – diagrammatic representation



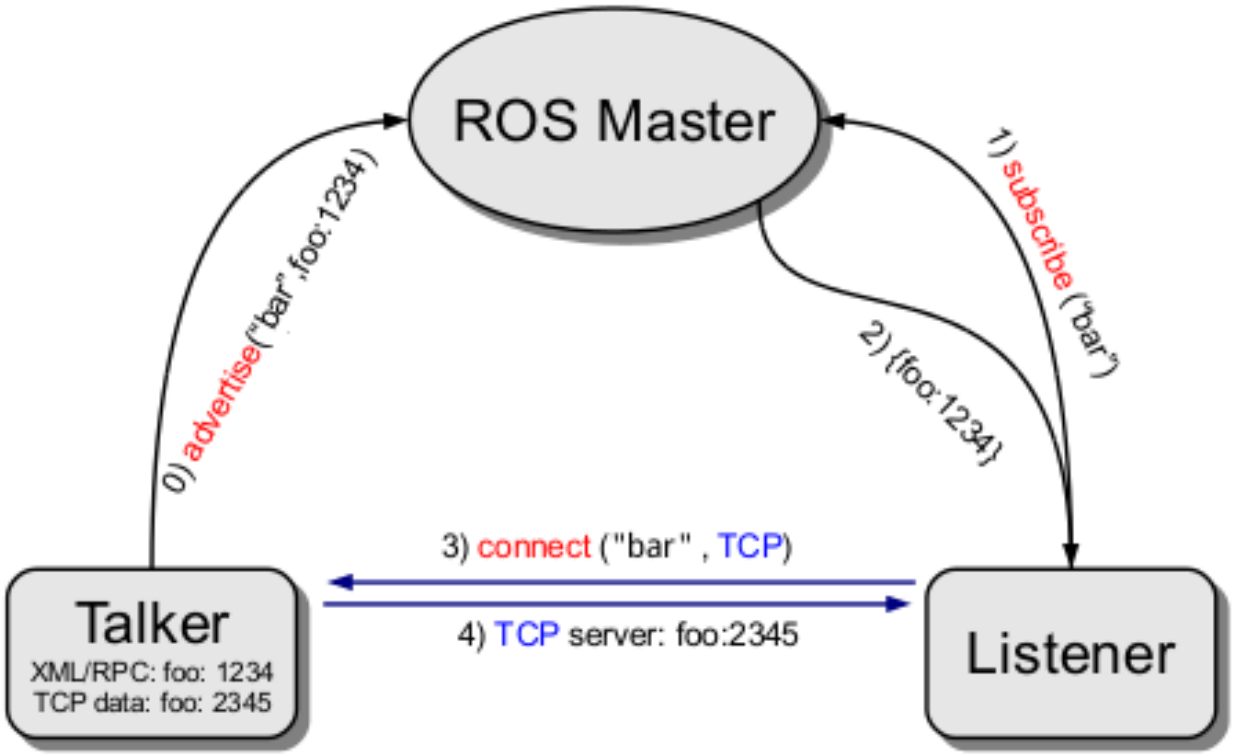
Topics – diagrammatic representation



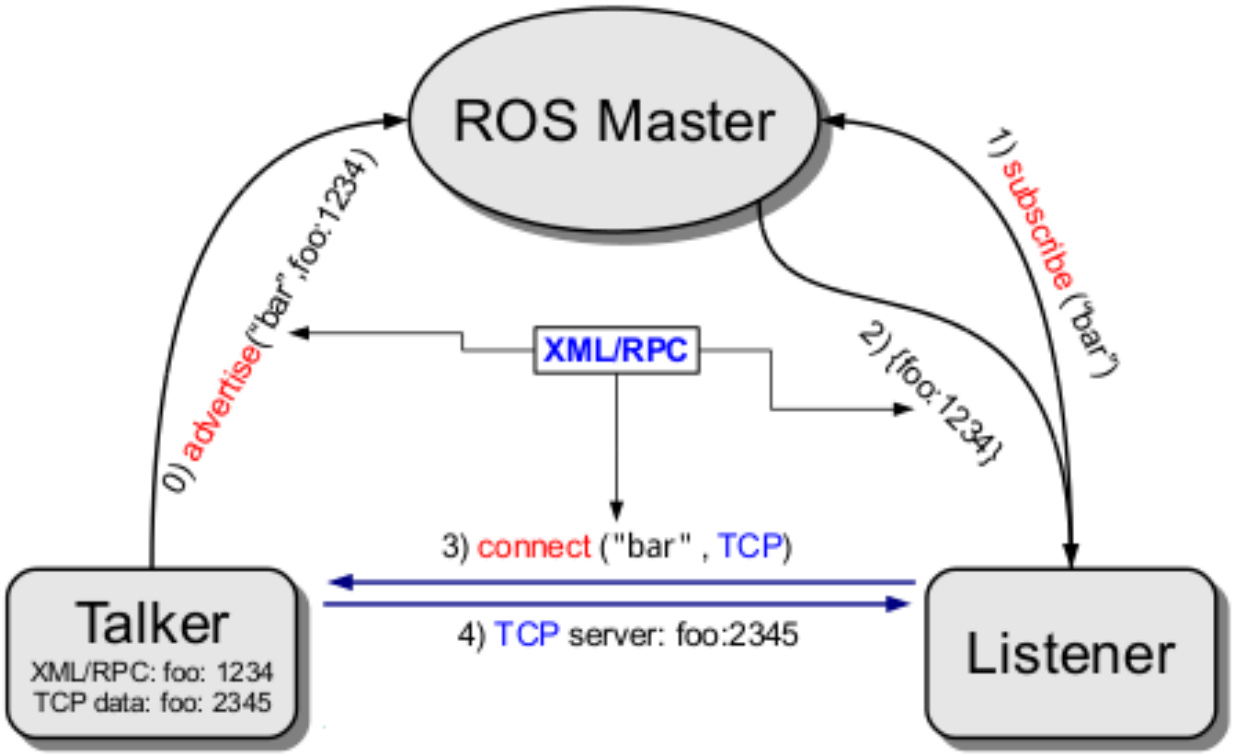
Topics – diagrammatic representation



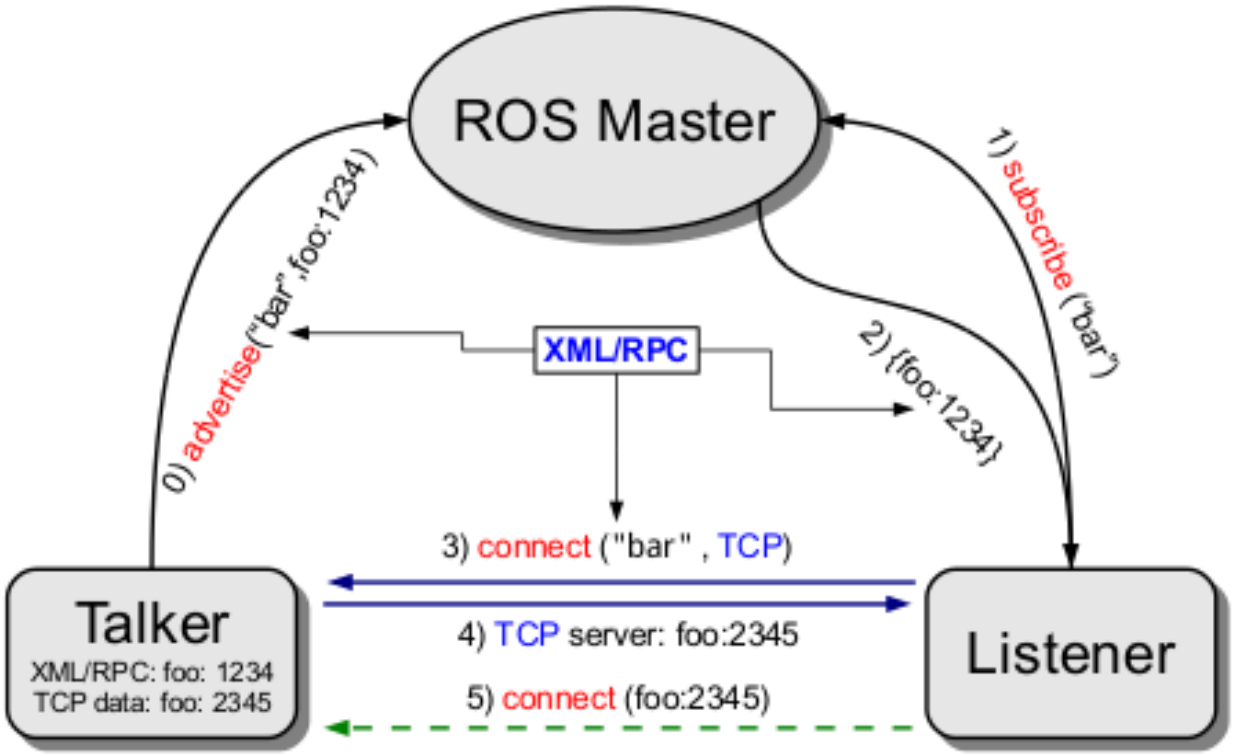
Topics – diagrammatic representation



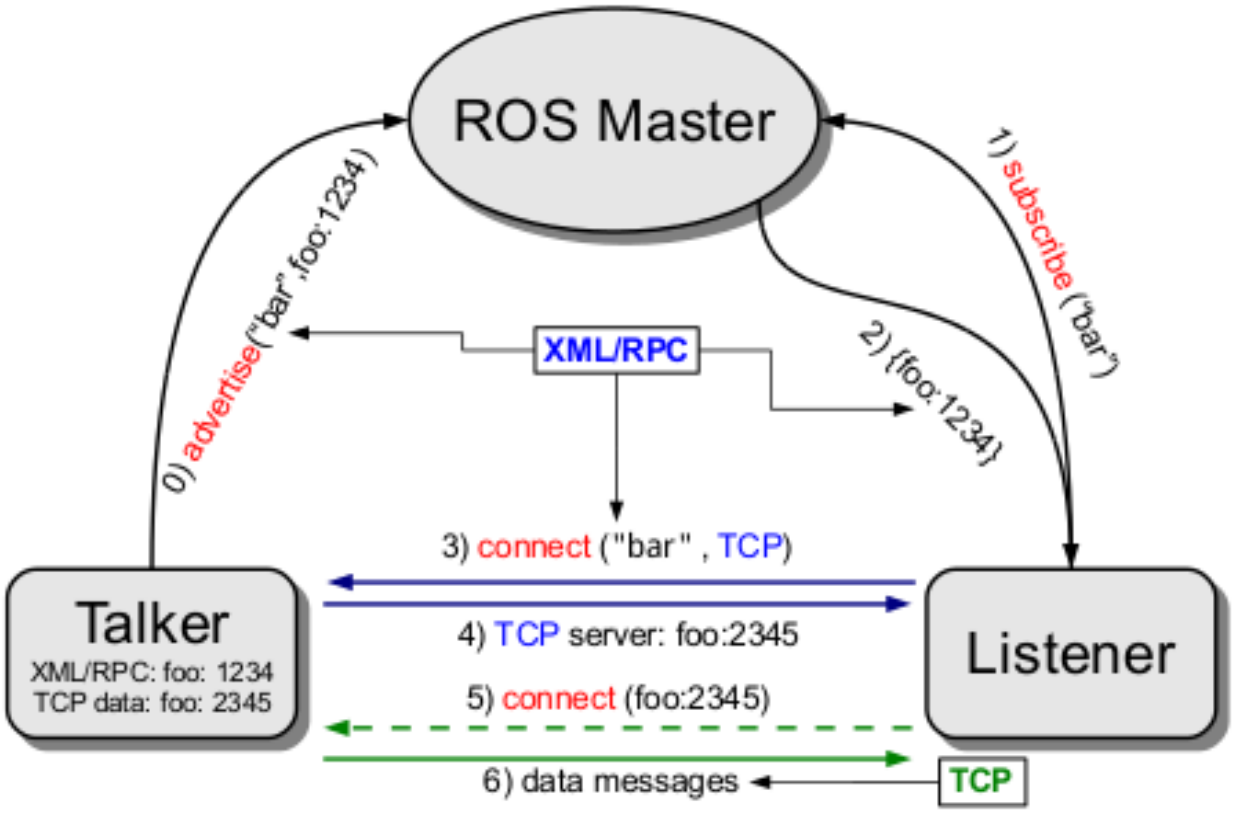
Topics – diagrammatic representation



Topics – diagrammatic representation



Topics – diagrammatic representation



Services

- Publish/subscribe paradigm not appropriate for services

Services

- Publish/subscribe paradigm not appropriate for services
- Services implement the request/reply functionality

Services

- Publish/subscribe paradigm not appropriate for services
- Services implement the request/reply functionality
- Pair of message structures: one for request and one for reply

Services

- Publish/subscribe paradigm not appropriate for services
- Services implement the request/reply functionality
- Pair of message structures: one for request and one for reply
- A node provider offers a service under a specific name

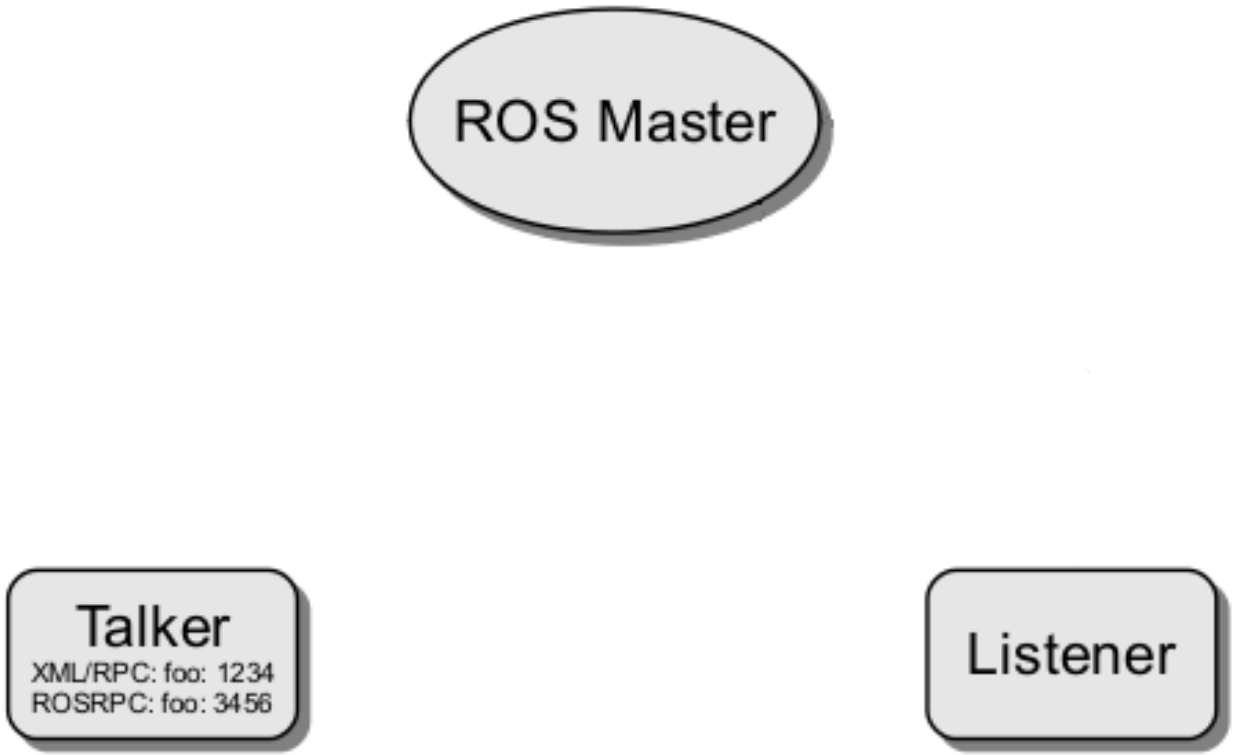
Services

- Publish/subscribe paradigm not appropriate for services
- Services implement the request/reply functionality
- Pair of message structures: one for request and one for reply
- A node provider offers a service under a specific name
- A client node uses the service by sending the request message and awaits for the reply

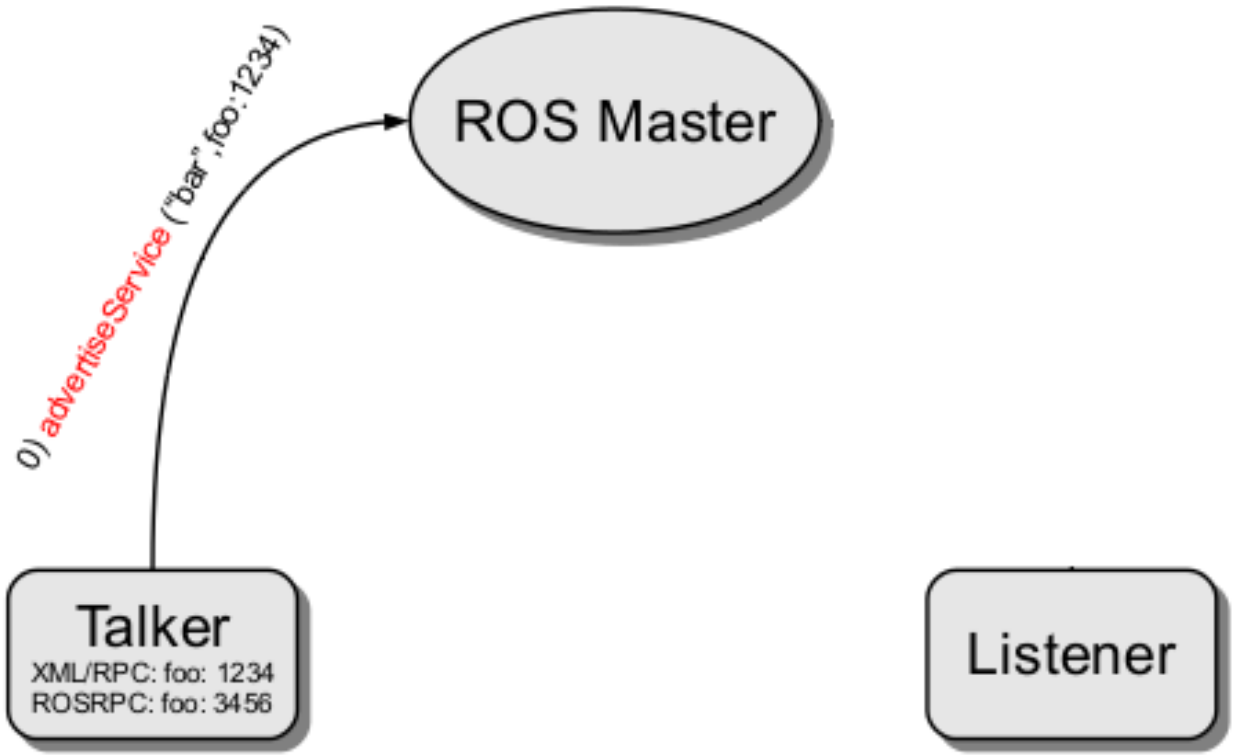
Services

- Publish/subscribe paradigm not appropriate for services
- Services implement the request/reply functionality
- Pair of message structures: one for request and one for reply
- A node provider offers a service under a specific name
- A client node uses the service by sending the request message and awaits for the reply
- From the programmer perspective, works as a remote procedure call

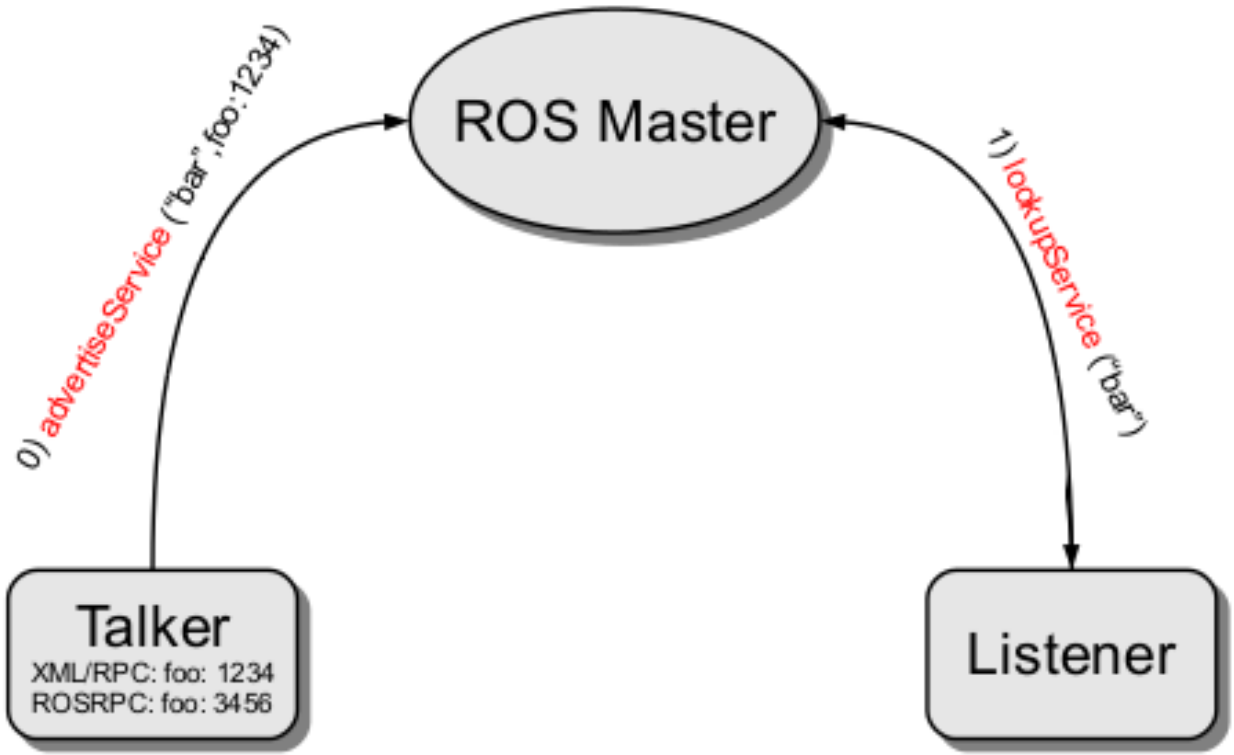
Services – diagrammatic representation



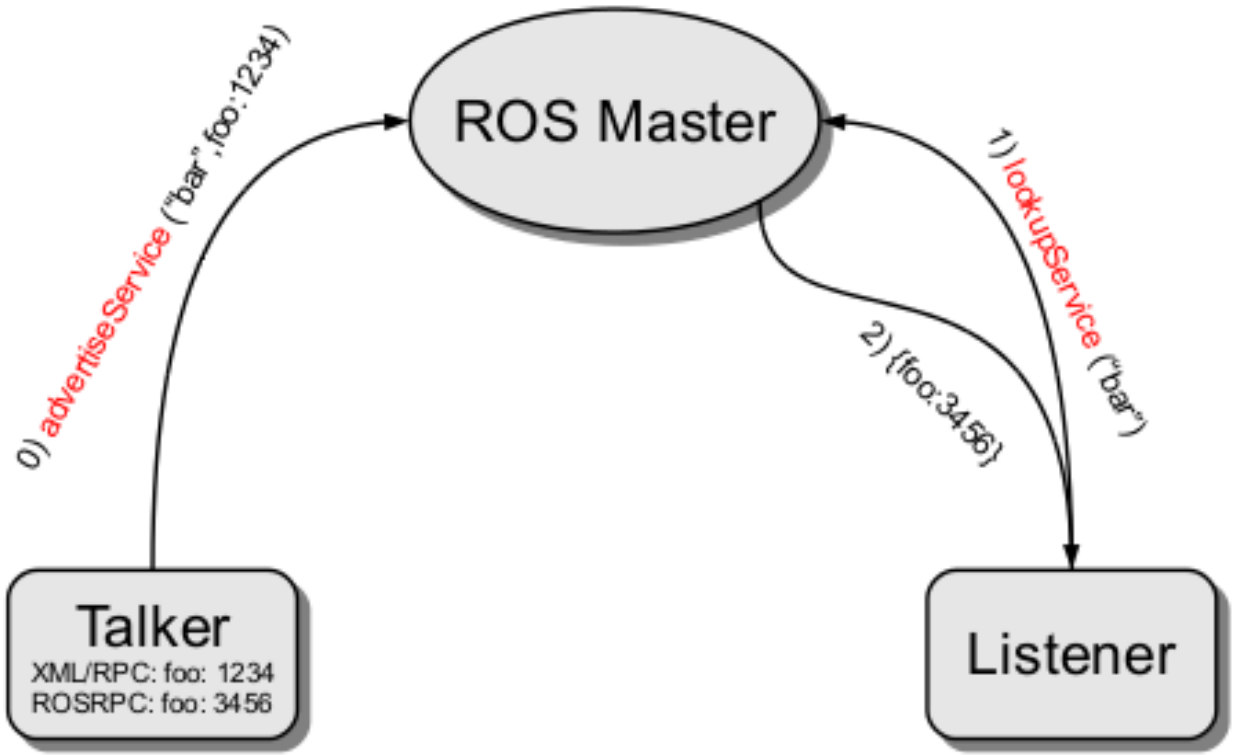
Services – diagrammatic representation



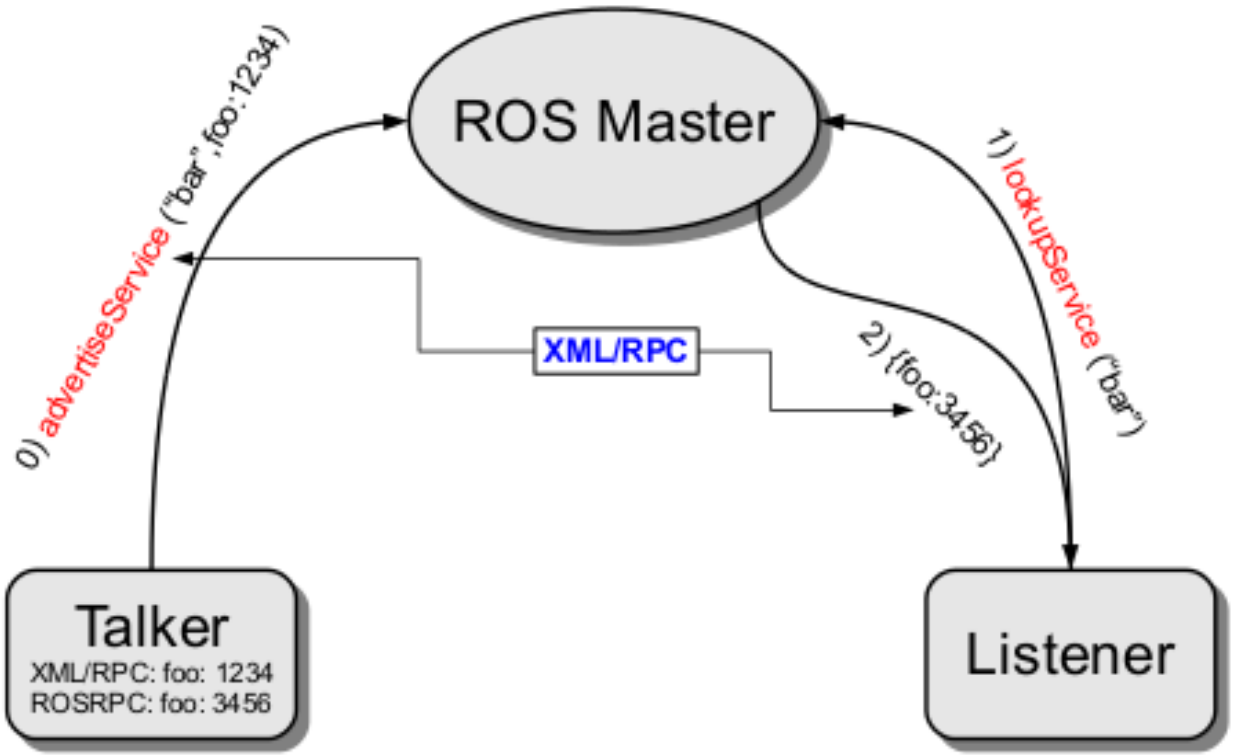
Services – diagrammatic representation



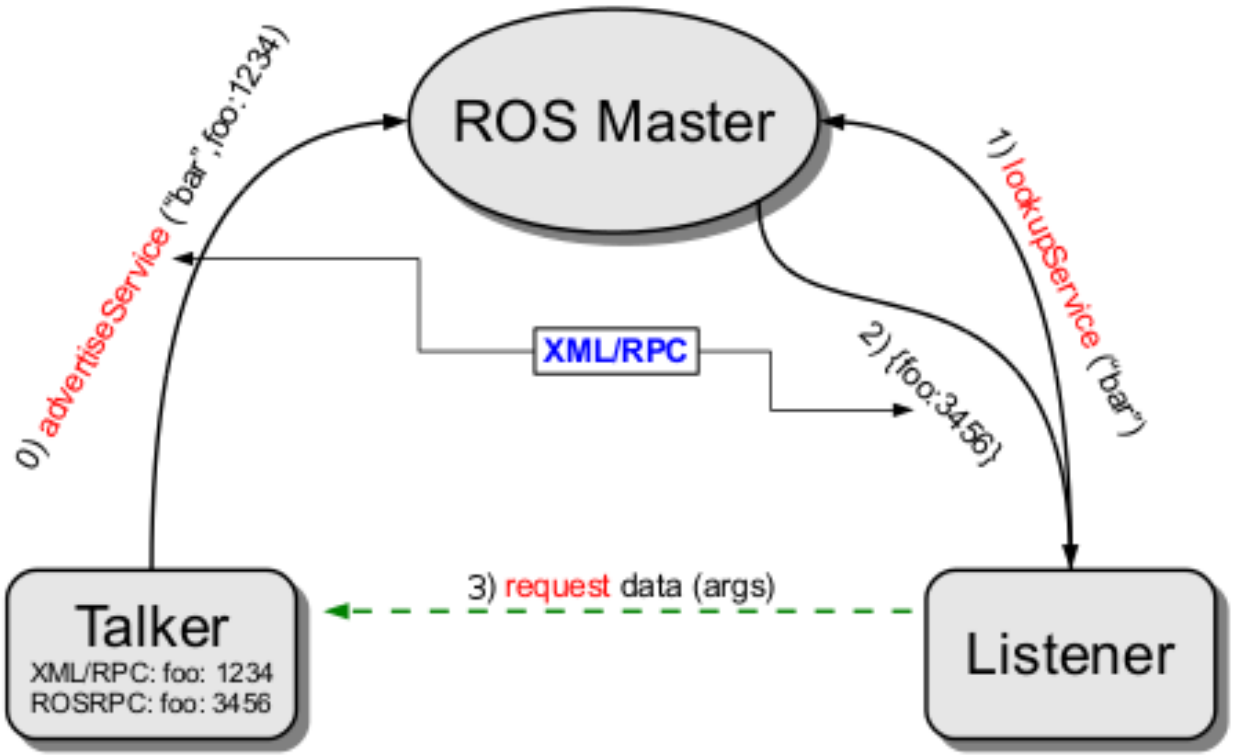
Services – diagrammatic representation



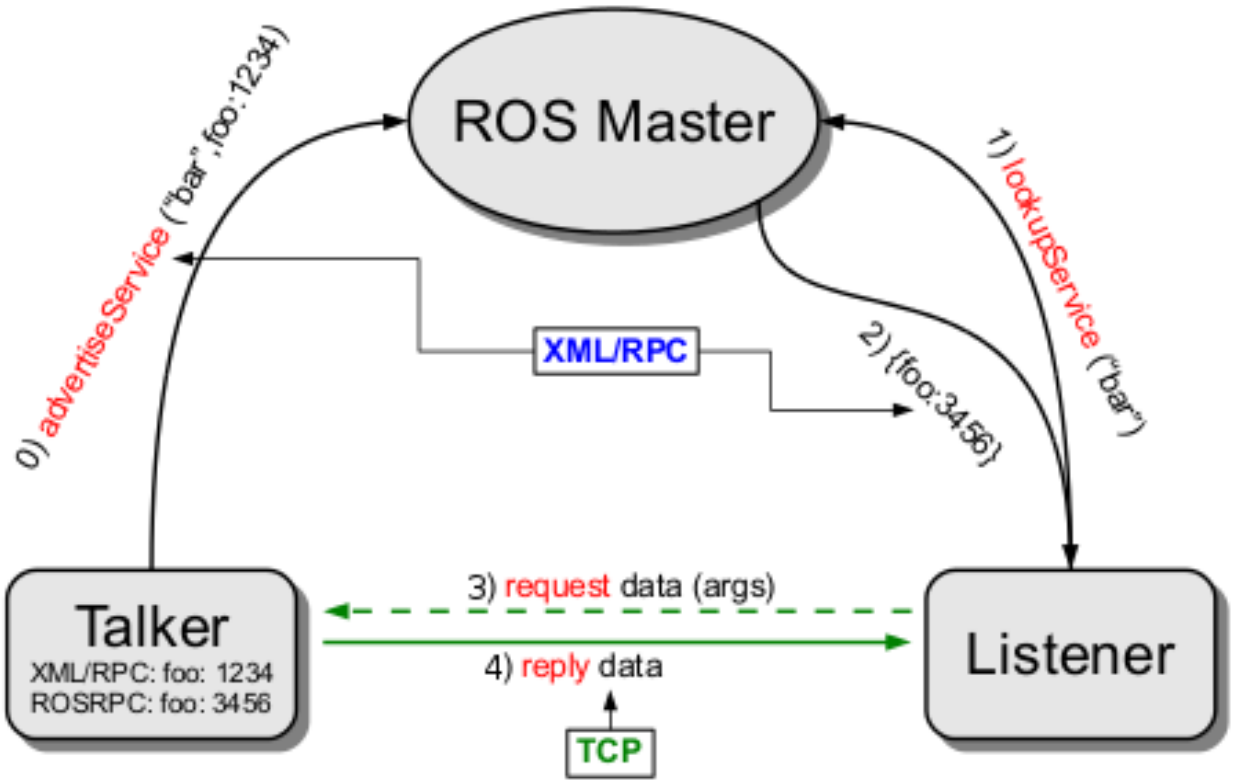
Services – diagrammatic representation



Services – diagrammatic representation



Services – diagrammatic representation



Messages – more ROS command line goodies

Viewing details of ROS messages

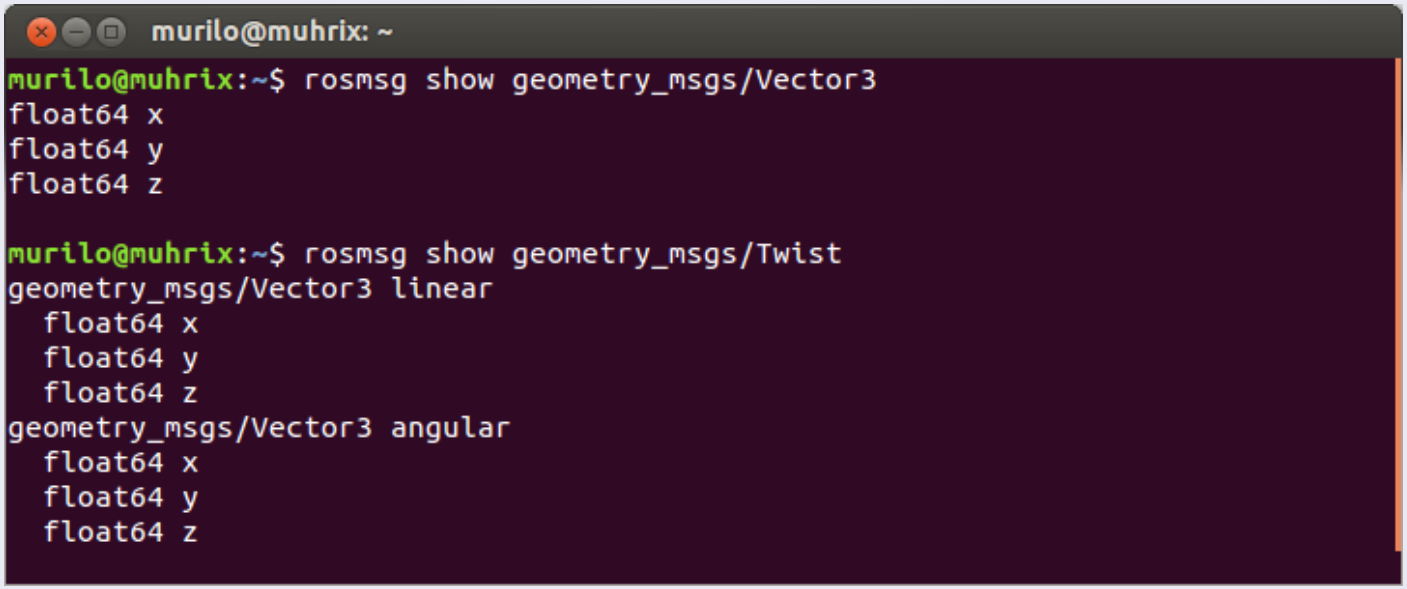
```
1 $ rosmsg list
2 $ rosmsg show geometry_msgs/Vector3
3 $ rosmsg show geometry_msgs/Twist
```

Messages – more ROS command line goodies

Viewing details of ROS messages

```
1  $ rosmsg list
2  $ rosmsg show geometry_msgs/Vector3
3  $ rosmsg show geometry_msgs/Twist
```

Vector3.msg and Twist.msg, from package geometry_msgs



Messages – more ROS command line goodies

Viewing details of ROS services

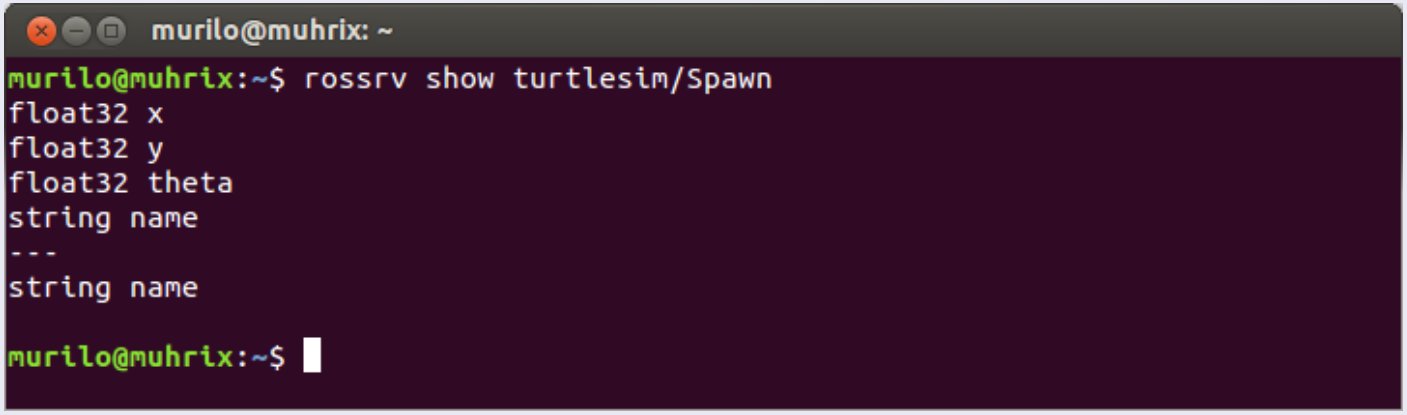
```
1 $ rossrv list
2 $ rossrv show turtlesim/Spawn
```

Messages – more ROS command line goodies

Viewing details of ROS services

```
$ rossrv list
$ rossrv show turtlesim/Spawn
```

Spawn.srv, from package turtlesim



ROS filesystem – Package structure revisited

Hypothetical catkin package my_pkg/

- CmakeLists.txt: CMake build settings for package my_pkg
- package.xml: metadata and dependencies required by package
- mainpage.dox: Doxygen information of package my_pkg
- include/my_pkg: C++ header files
- src/: source code directory
- launch/: where launch files should be stored
- msg/: message (.msg) definitions
- srv/: service (.srv) definitions
- scripts/: executable (Python) scripts

That's all for today, folks!

Today we have learnt about

- How awesome (and complex) ROS is

That's all for today, folks!

Today we have learnt about

- How awesome (and complex) ROS is
- Why we should surrender to Ubuntu

That's all for today, folks!

Today we have learnt about

- How awesome (and complex) ROS is
- Why we should surrender to Ubuntu
- How easy it is to install ROS (on Ubuntu, that is)

That's all for today, folks!

Today we have learnt about

- How awesome (and complex) ROS is
- Why we should surrender to Ubuntu
- How easy it is to install ROS (on Ubuntu, that is)
- The convenience of rosbash command line tools

That's all for today, folks!

Today we have learnt about

- How awesome (and complex) ROS is
- Why we should surrender to Ubuntu
- How easy it is to install ROS (on Ubuntu, that is)
- The convenience of rosbash command line tools
- The ROS filesystem concepts

That's all for today, folks!

Today we have learnt about

- How awesome (and complex) ROS is
- Why we should surrender to Ubuntu
- How easy it is to install ROS (on Ubuntu, that is)
- The convenience of rosbash command line tools
- The ROS filesystem concepts
- The ROS Computation Graph (most important lesson of today!)

That's all for today, folks!

Today we have learnt about

- How awesome (and complex) ROS is
- Why we should surrender to Ubuntu
- How easy it is to install ROS (on Ubuntu, that is)
- The convenience of rosbash command line tools
- The ROS filesystem concepts
- The ROS Computation Graph (most important lesson of today!)

Question time

Thank you very much!
Questions?