Developing on ROS Framework

(ROS = Robot Operating System)

Rodrigo Ventura
João Reis
Institute for Systems and Robotics
Instituto Superior Técnico, Lisboa

Lisbon, 23-26 June 2013

Sponsored by project FP7-ICT-2011-9-601033 (MOnarCH)

Program outline

- Schedule: 14h - 19h

- Day 1: GNU/Linux operating system
- Day 2: C++ language
- Day 3: Python language
- Day 4: Robot Operating System (ROS)

- Course website:

  http://mediawiki.isr.ist.utl.pt/wiki/Summer_course_on_ROS_framework_2013

What is ROS?

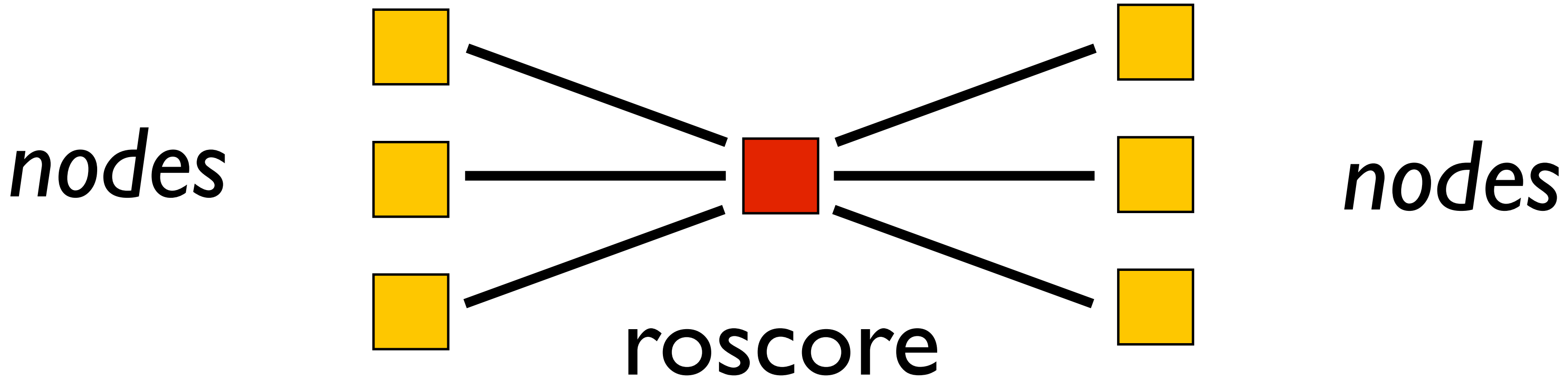
- ROS = Robot Operating System
- Framework for robot software development providing operating system-like functionality
- Originated at Stanford Artificial Intelligence Lab, then further developed at Willow Garage
- Supports all major host operating systems

(Ubuntu recommended) Linux   Mac OS X   Windows   Raspbian   QNX

- Large user base; getting widespread use
- ROS users forum: http://answers.ros.org

Basic concept #1: Node

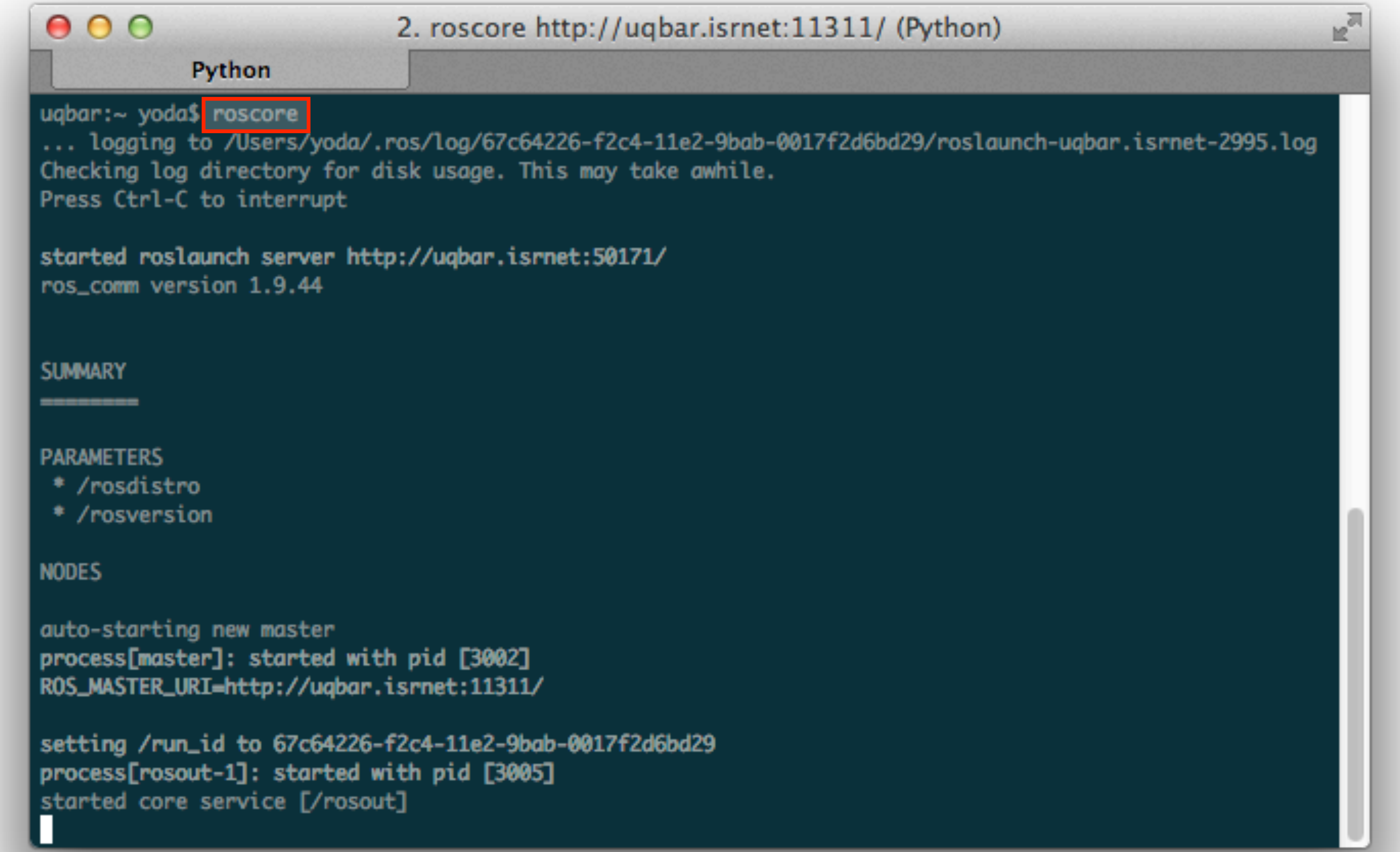
- Modularization in ROS is achieved by separated operating system processes
- *Node* = a process that uses ROS framework
- Nodes may reside in different machines transparently
- Nodes get to know one another via roscore



- roscore acts primarily as a name server
- Nodes use the roscore running in localhost by default overriden by the env. var. ROS_MASTER_URI

Basic concept #1: Node

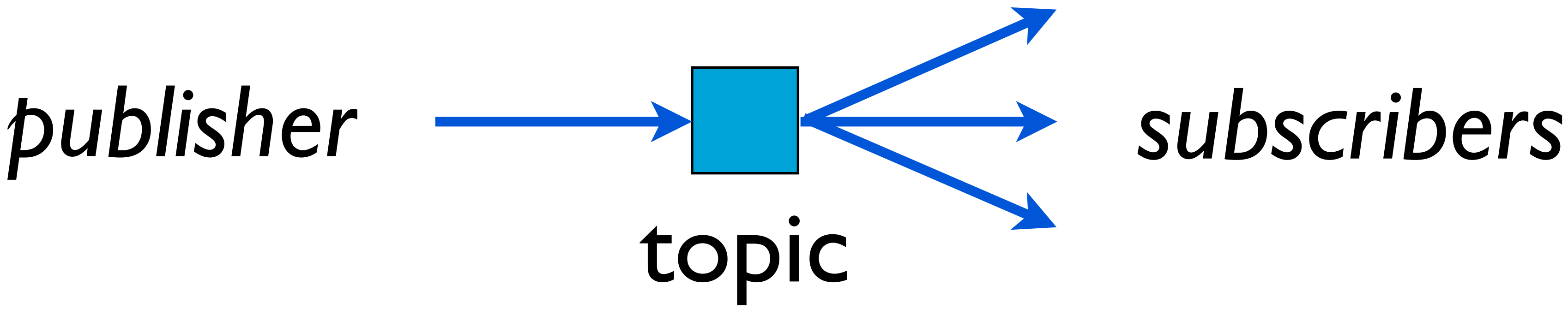
- Demo: lanching roscore



```
uqbar:~ yoda$ roscore
... logging to /Users/yoda/.ros/log/67c64226-f2c4-11e2-9bab-0017f2d6bd29/roslaunch-uqbar.isrnet-2995.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt

started roslaunch server http://uqbar.isrnet:50171/
ros_comm version 1.9.44


SUMMARY
========

PARAMETERS
 * /rosdistro
 * /rosversion

NODES

auto-starting new master
process[master]: started with pid [3002]
ROS_MASTER_URI=http://uqbar.isrnet:11311/

setting /run_id to 67c64226-f2c4-11e2-9bab-0017f2d6bd29
process[rosout-1]: started with pid [3005]
started core service [/rosout]
```

Basic concept #2: Topic

- *Topic* is a mechanism to send messages from a node to one or more nodes
- Follows a publisher-subscriber design pattern

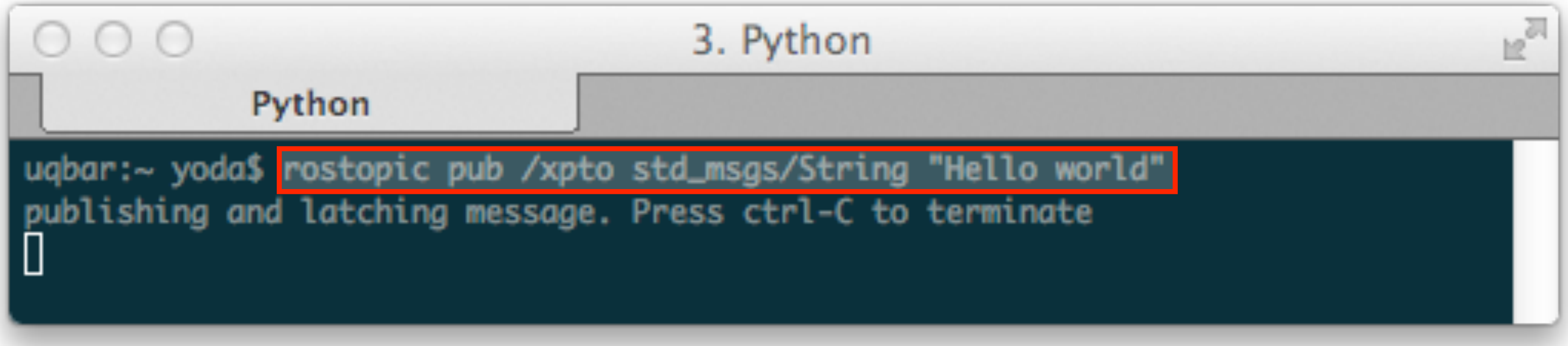


- *Publish* = to send a message to a topic

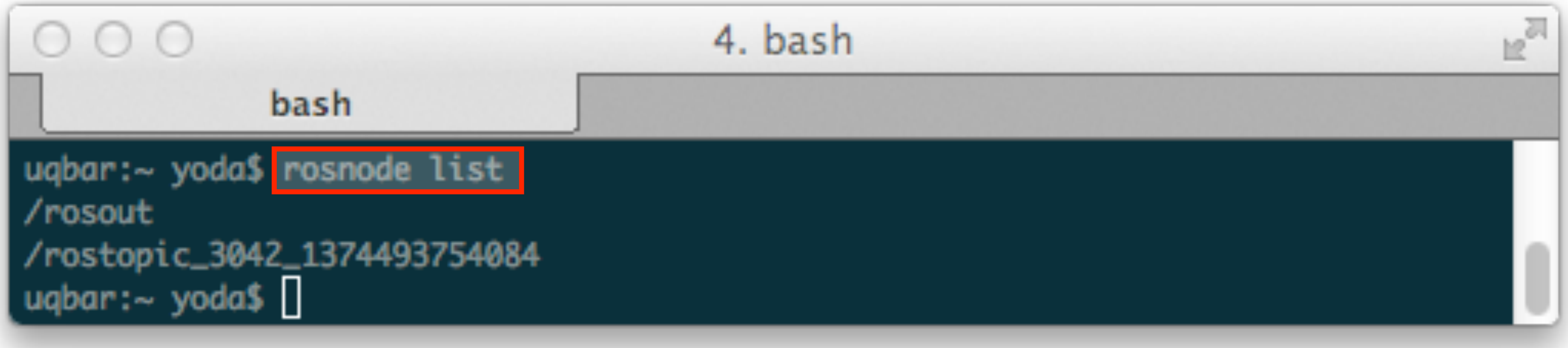
  Subscribe = get called whenever a message is published
- Published messages are broadcast to all Subscribers
- Example: LIDAR publishing scan data

Basic concept #2: Topic

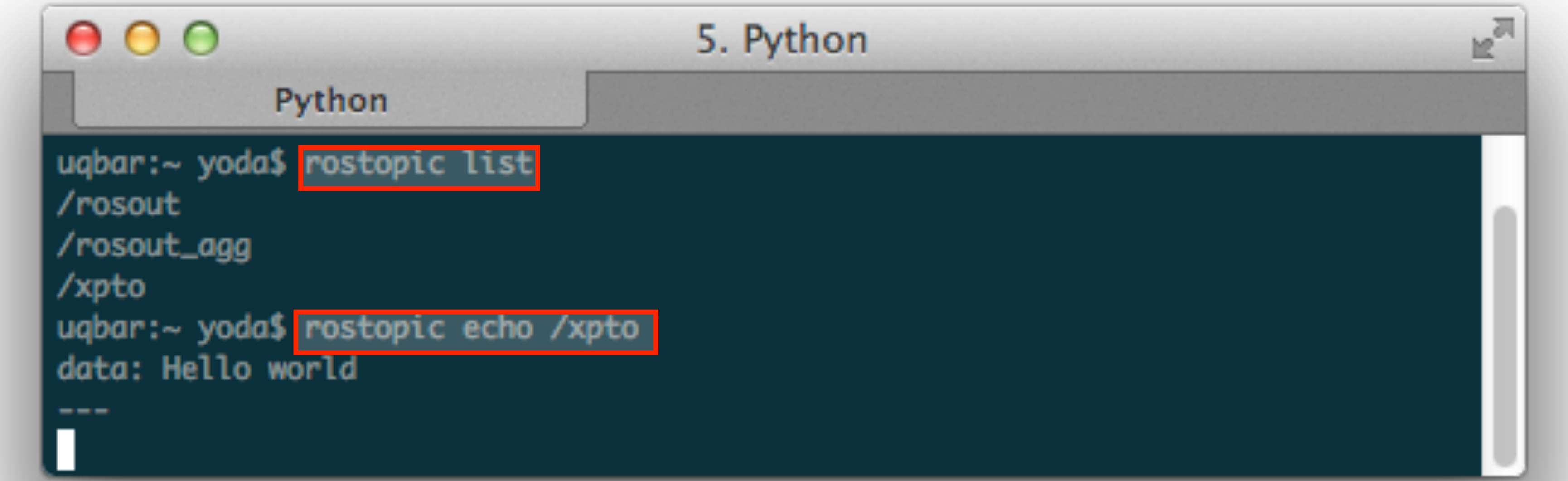
- Demo: publishing an "Hello world" String to topic /xpto



```
uqbar:~ yoda$ rostopic pub /xpto std_msgs/String "Hello world"
publishing and latching message. Press ctrl-C to terminate
```



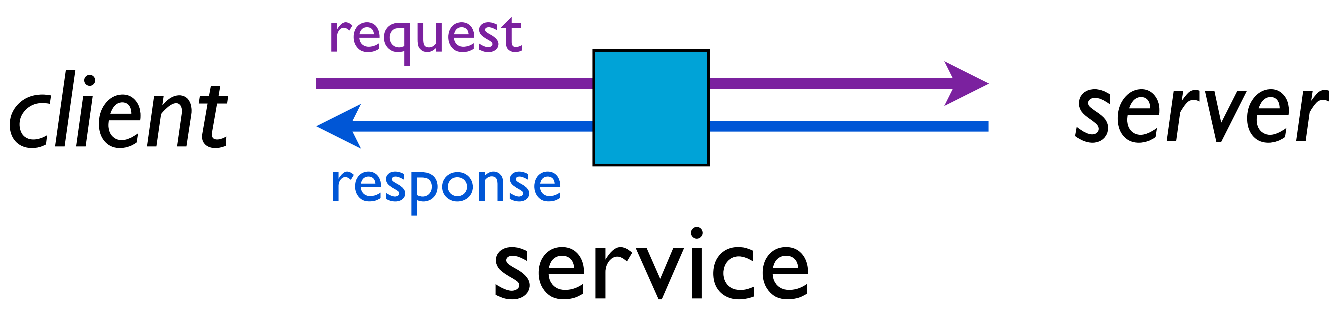
```
uqbar:~ yoda$ rosnode list
/rosout
/rostopic_3042_1374493754084
uqbar:~ yoda$
```



```
uqbar:~ yoda$ rostopic list
/rosout
/rosout_agg
/xpto
uqbar:~ yoda$ rostopic echo /xpto
data: Hello world
---
```

Basic concept #3: Service

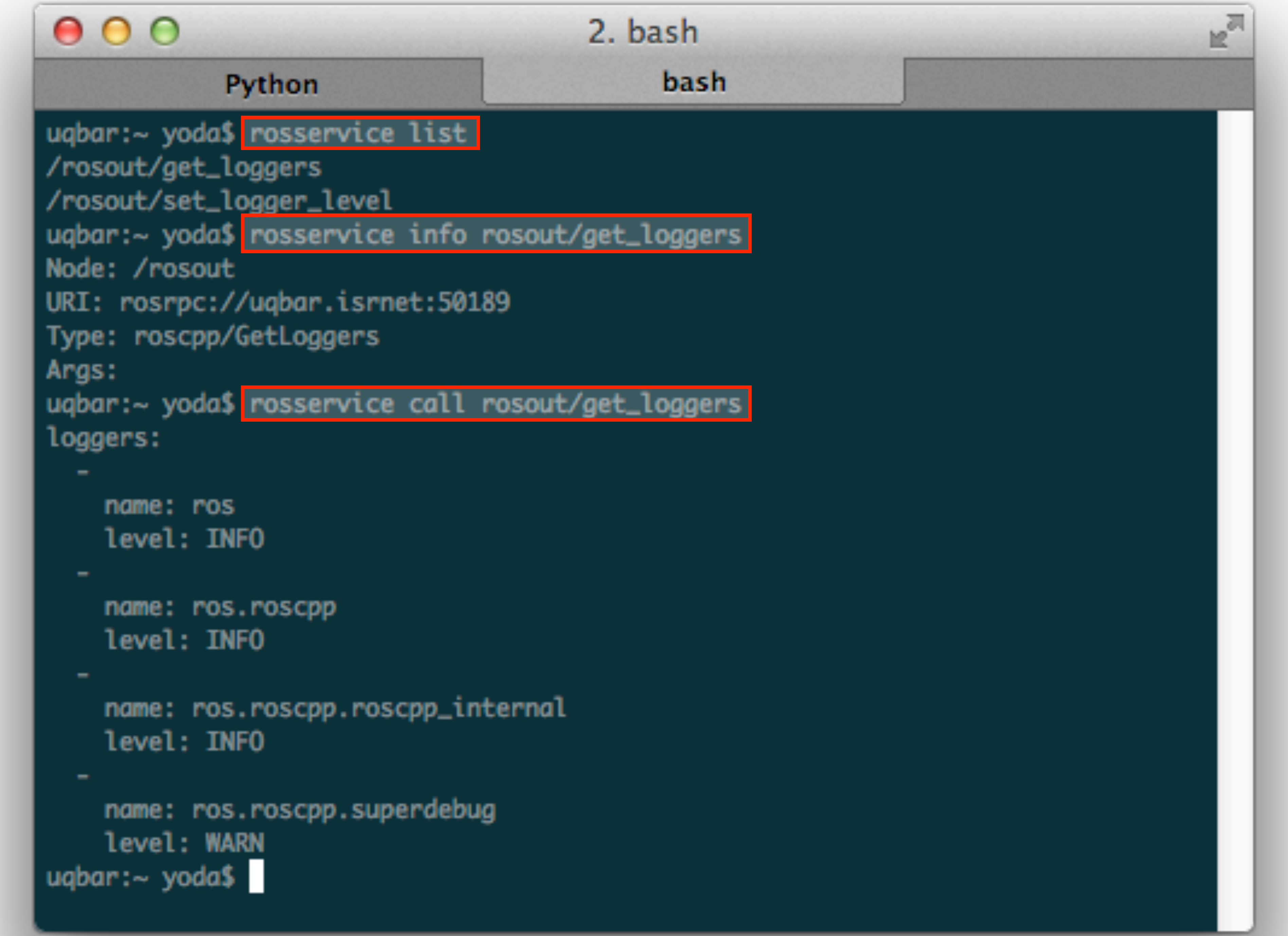
- *Service* is a mechanism for a node to send a request to another node and receive a response in return
- Follows a request-response design pattern



- A service is called with a request structure, and in return, a response structure is returned
- Similar to a Remote Procedure Call (RPC)
- Example: set location to a localization node

Basic concept #3: Service

- Demo: querying and calling a service



```
uqbar:~ yoda$ rosservice list
/rosout/get_loggers
/rosout/set_logger_level
uqbar:~ yoda$ rosservice info rosout/get_loggers
Node: /rosout
URI: rosrpc://uqbar.isrnet:50189
Type: roscpp/GetLoggers
Args:
uqbar:~ yoda$ rosservice call rosout/get_loggers
loggers:
  -
    name: ros
    level: INFO
  -
    name: ros.roscpp
    level: INFO
  -
    name: ros.roscpp.roscpp_internal
    level: INFO
  -
    name: ros.roscpp.superdebug
    level: WARN
uqbar:~ yoda$
```

Message types

- All messages (including service requests/responses) are defined in text files
- Example: *built-in laser scan data message*

```
--- sensor_msgs/msg/LaserScan.msg ---

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Message types

- Another example: *remote interface service in Cobot*

request

```
--- cobot_msgs/srv/CobotRemoteInterfaceSrv.srv ---

# "Joystick" velocity commands:
float32 drive_x #Distance to move along x in meters
float32 drive_y #Distance to move along x in meters
float32 drive_r #Distance to turn in radians

# command_num must increment every time the service is called - used to reject out of sync commands
int32 command_num

# valid command flags:
#   CmdMove = 0x0001
#   CmdSetLocation = 0x0002
#   CmdGetLocation = 0x0004
#   CmdGetParticlesSampling = 0x0010
#   CmdSetTarget = 0x0020
int32 command_type

# The following parameters are used for commands CmdSetLocation and CmdSetTarget
float32 loc_x
float32 loc_y
float32 orientation
string map

---
```

response

```
float32 loc_x
float32 loc_y
float32 orientation

float32[] particles_x
float32[] particles_y
float32[] particles_weight
float32[] locations_weight

int8 err_code
```

Development

- Two major languages are supported:
  - C++
  - Python
- ROS provides a portable build system
- *Package* = self-contained directory containing sources, makefiles, builds, etc.

- The code reuse units in ROS are packages
- A large variety of packages can be found on the web

  examples: sensor drivers, simulators, SLAM, image processing, etc.

Command line tools

```
$ rosnode
```

```
rosnode is a command-line tool for printing information about ROS Nodes.

Commands:
          rosnode ping      test connectivity to node
          rosnode list      list active nodes
          rosnode info      print information about node
          rosnode machine   list nodes running on a particular machine
                            or list machines
          rosnode kill      kill a running node
          rosnode cleanup   purge registration information of
                             unreachable nodes

Type rosnode <command> -h for more detailed usage, e.g. 'rosnode ping -h'
```

Command line tools

```
$ rostopic

rostopic is a command-line tool for printing information about ROS Topics.

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

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'
```

Command line tools

$ rosservice

```
Commands:
        rosservice args print service arguments
        rosservice call call the service with the provided args
        rosservice find find services by service type
        rosservice info print information about service
        rosservice list list active services
        rosservice type print service type
        rosservice uri  print service ROSRPC uri

Type rosservice <command> -h for more detailed usage, e.g. 'rosservice call -h'
```

Command line tools

```
$ rosbag
```

```
Usage: rosbag <subcommand> [options] [args]

Available subcommands:
   check
   compress
   decompress
   filter
   fix
   help
   info
   play
   record
   reindex

For additional information, see http://code.ros.org/wiki/rosbag/
```