

Introduction to the Robot Operating System (ROS)

Rodrigo Ventura

Institute for Systems and Robotics
Instituto Superior Técnico
Portugal

rodrigo.ventura@tecnico.ulisboa.pt

[Updated 25-Apr-2025]

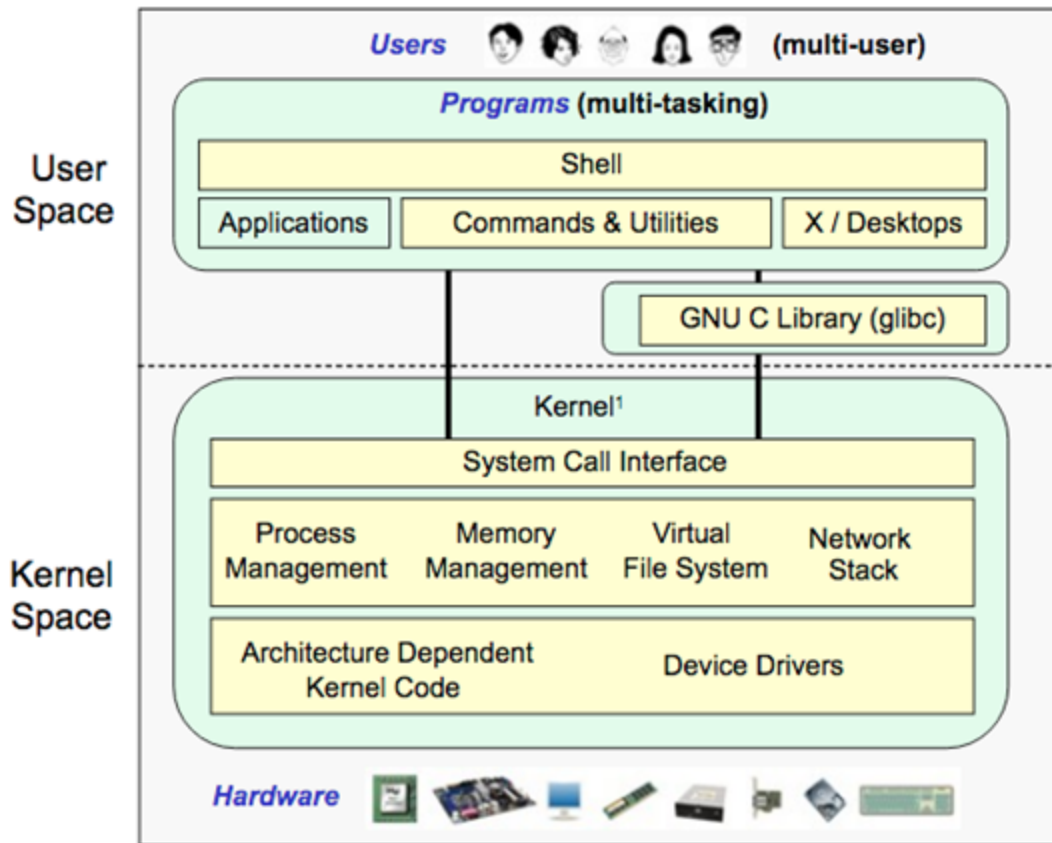
What is ROS?



- **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
- Works quite well in Linux Ubuntu and WSL, there are bindings to Java, C#, and can be tunneled via websockets
- Large user base; getting widespread use
- Extensive documentation in ROS wiki: <http://wiki.ros.org/>



GNU/Linux Operating System Architecture



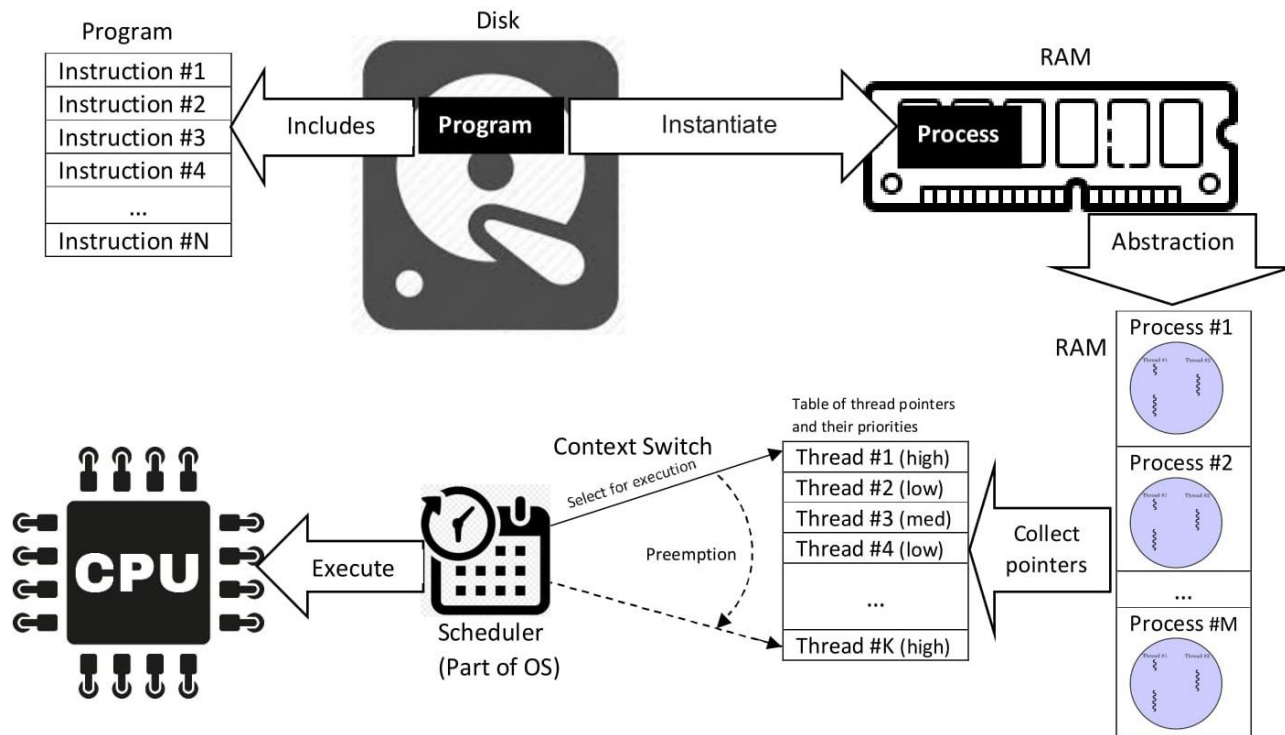
Richard Stallman started the GNU project in 1983 to create a free UNIX-like OS. He Founded the Free Software Foundation in 1985. In 1989 he wrote the first version of the GNU General Public License



Linus Torvalds, as a student, initially conceived and assembled the Linux kernel in 1991. The kernel was later re-licensed under the GNU General Public License in 1992.

¹See "Anatomy of the Linux kernel" by M. Tim Jones at <http://www-128.ibm.com/developerworks/linux/library/l-linux-kernel/>

What is a process?

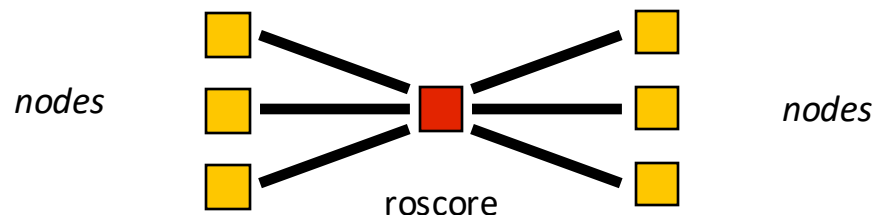


What is a process?

- Code – read-only copy in memory of the program
 - Data – read/write memory with data
 - Stack – return pointer from subroutines + local variables
 - File descriptors – open files, including real files, sockets, etc.
 - Environment variables – shell variables passed to process
 - Signals – callbacks called by the system in certain situations
 - ...
-
- fork – everything is copied to a new identical process
 - threads – everything is shared, except stack

Basic concept #1: Node

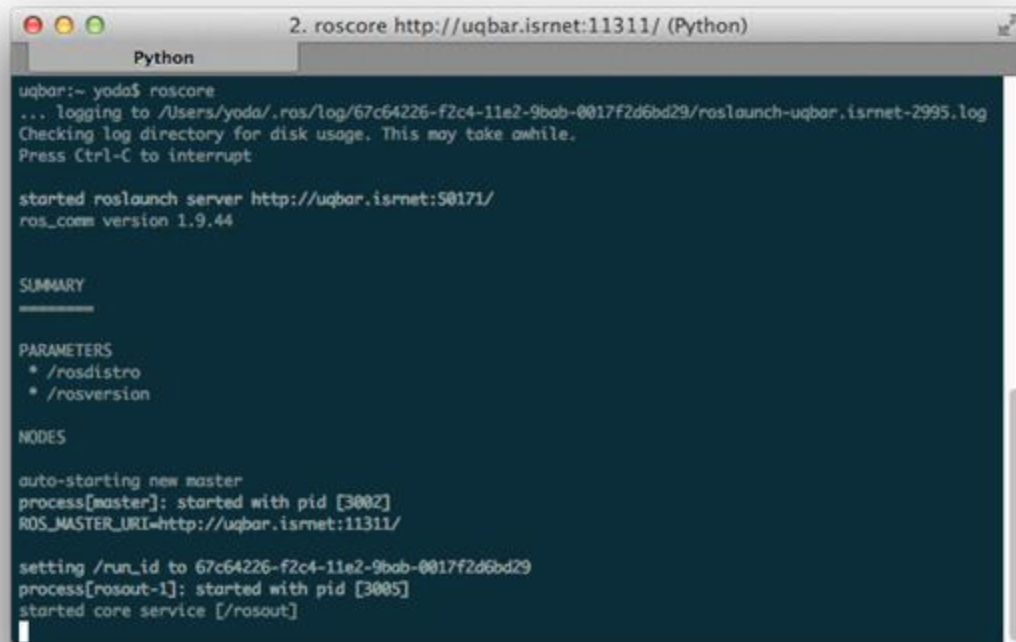
- Modularization in ROS is achieved by operating system processes
- **Node** = a process that uses ROS framework
- Nodes may reside in different machines transparently
- Nodes get to know one another via roscore



- roscore acts primarily as a “name server”, i.e., maps names to nodes
- Nodes use the roscore running in localhost by default
overridden by the environment variable `ROS_MASTER_URI`

Basic concept #1: Node

Demo: launching roscore



```
Python
2. roscore http://uqbar.isrnet:11311/ (Python)

uqbar:~ yoda$ roscore
... logging to /Users/yoda/.ros/log/67c64226-f2c4-11e2-9bob-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
 * /rostdistro
 * /rosversion

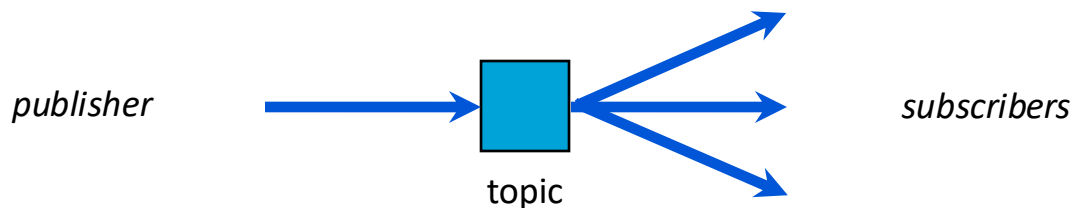
NODES

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

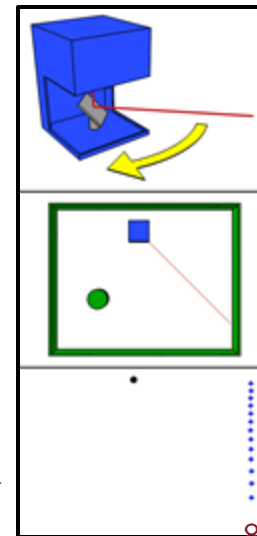
setting /run_id to 67c64226-f2c4-11e2-9bob-0017f2d6bd29
process[rosout-1]: started with pid [3005]
started core service [/rosout]
```

Basic concept #2: Topic

- **Topic** = mechanism to send messages among nodes
- Follows a publisher-subscriber design pattern

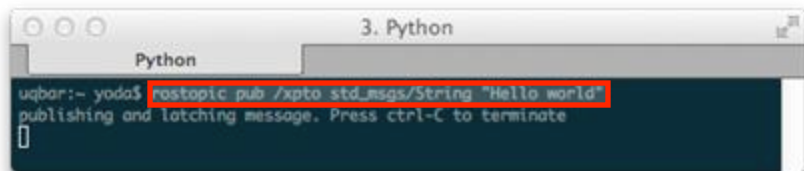


- **Publish** = to send a message to a topic
- **Subscribe** = get called whenever a message is published
- Published messages are broadcast to all Subscribers
- Example: LIDAR publishing scan data



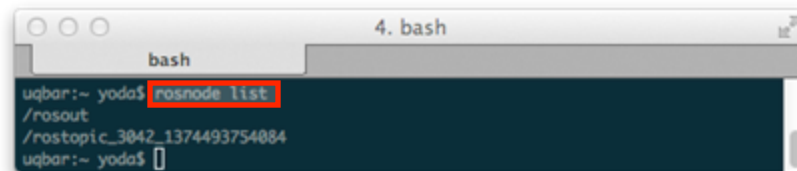
Basic concept #2: Topic

Demo: publishing an “Hello world” String to topic /xpto



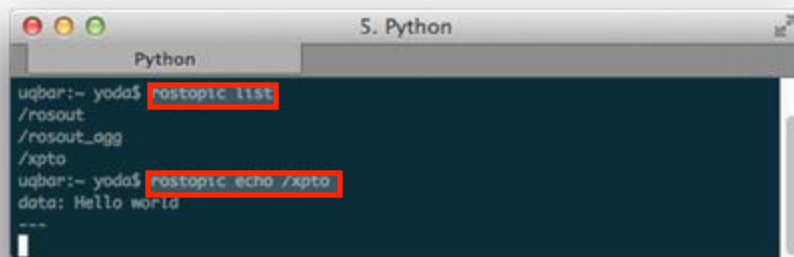
```
Python
uqbar:~ yoda$ rostopic pub /xpto std_msgs/String "Hello world"
publishing and latching message. Press ctrl-C to terminate

```



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

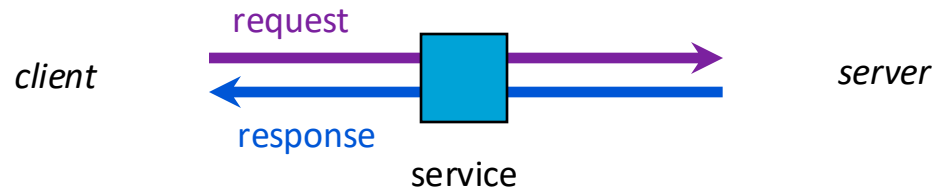
```



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

Basic concept #3: Service

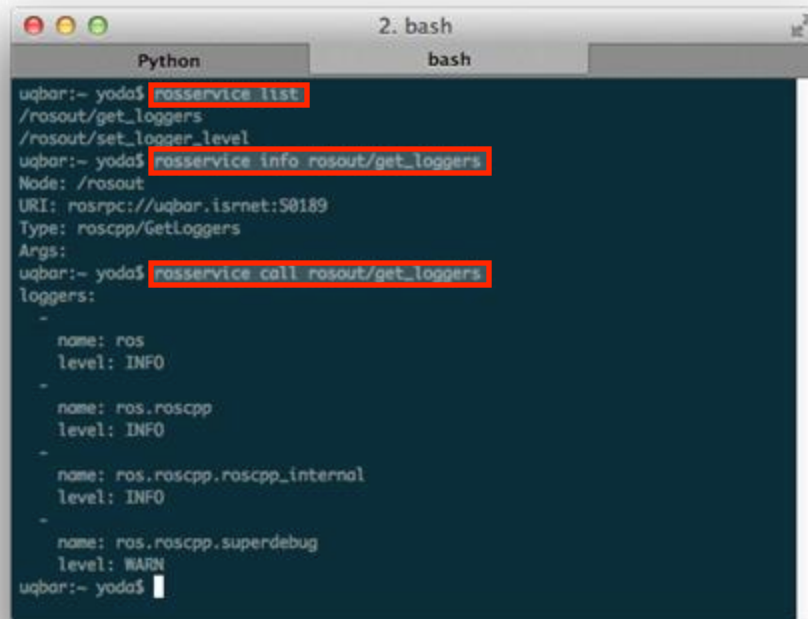
- **Service** = mechanism for a node to send a request to another node and receive a response from it in return
- Follows a request-response design pattern



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

Basic concept #3: Service

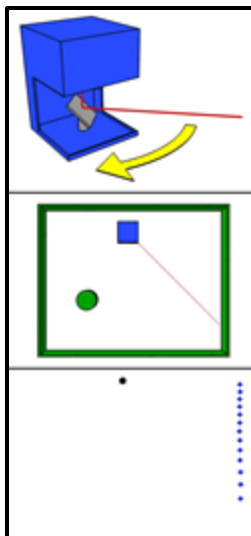
Demo: querying and calling a service



```
2. bash
Python bash
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.superebug
  level: WARN
uqbar:~ yoda$
```

Message types

All messages (including service requests/responses) are defined in text files



Contents of 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

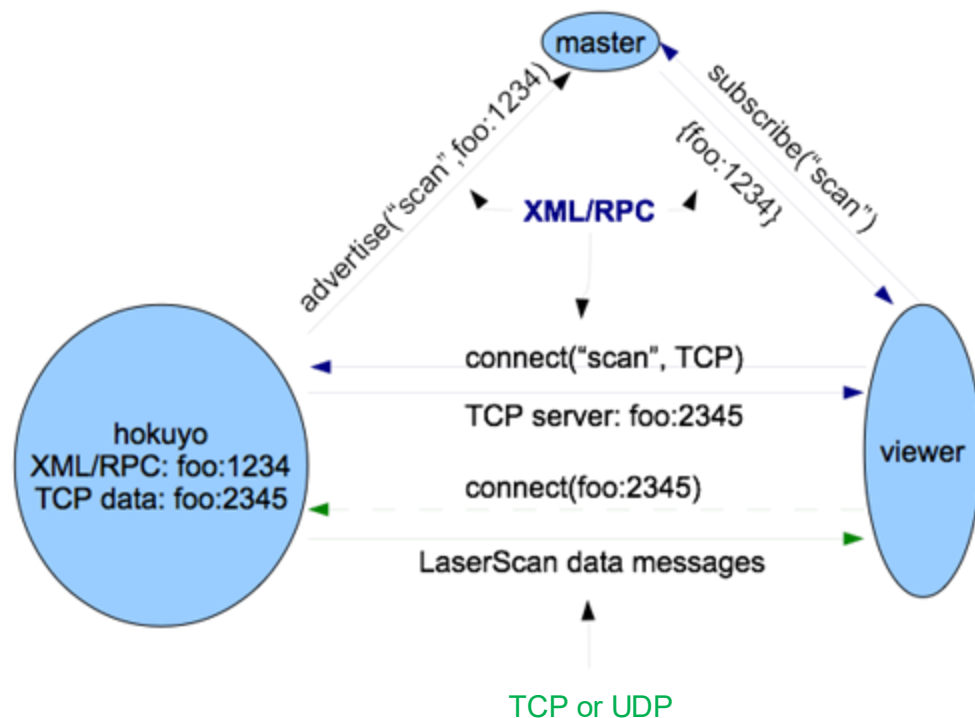
float32 scan_time       # time between scans [seconds]

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.
```



Topic internals



Development

- Two major languages are supported:
 - C++
 - Python
- ROS provides a portable build system (catkin, replacing rosbuilt)
- **Package** = encapsulation of sources, data files, and building files
- 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

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

Commands:

<code>rostopic ping</code>	test connectivity to node
<code>rostopic list</code>	list active nodes
<code>rostopic info</code>	print information about node
<code>rostopic machine</code>	list nodes running on a particular machine or list machines
<code>rostopic kill</code>	kill a running node
<code>rostopic cleanup</code>	purge registration information of unreachable nodes

Command line tools

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

Commands:

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

Command line tools

rosservice is a command-line tool for printing information about ROS Services.

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
```

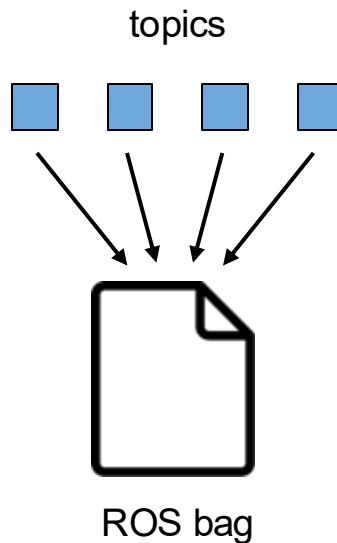
Command line tools

rosvbag is a command-line tool for manipulating log files (a.k.a. bags)

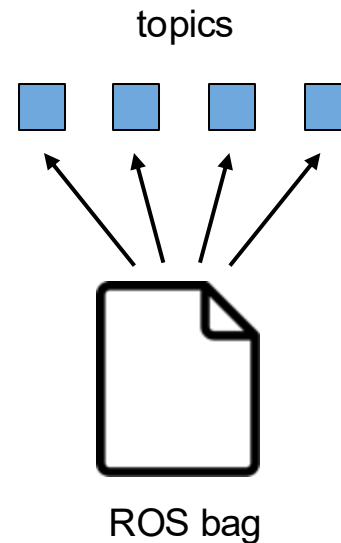
Available subcommands:

```
check
compress
decompress
filter
fix
help
info
play
record
reindex
```

rosvbag record ...



rosvbag play ...



Useful ROS facilities

- **Parameters:** repository of parameters (stored in the roscore)
 - Loading from files (formatted in YAML)
 - Dynamic update
 - Command-line utility: `rosparam`

params.yaml

```
course_name: "SAut"

robot1:
  name: "Calvin"
  height: 0.5

robot2:
  name: "Hobbes"
  height: 1.0
```

```
$ rosparam load params.yaml
$ rosparam list
/course_name
/robot1/height
/robot1/name
/robot2/height
/robot2/name
[...]
$ rosparam get course_name
SAut
$ rosparam get /robot2/name
Hobbes
```

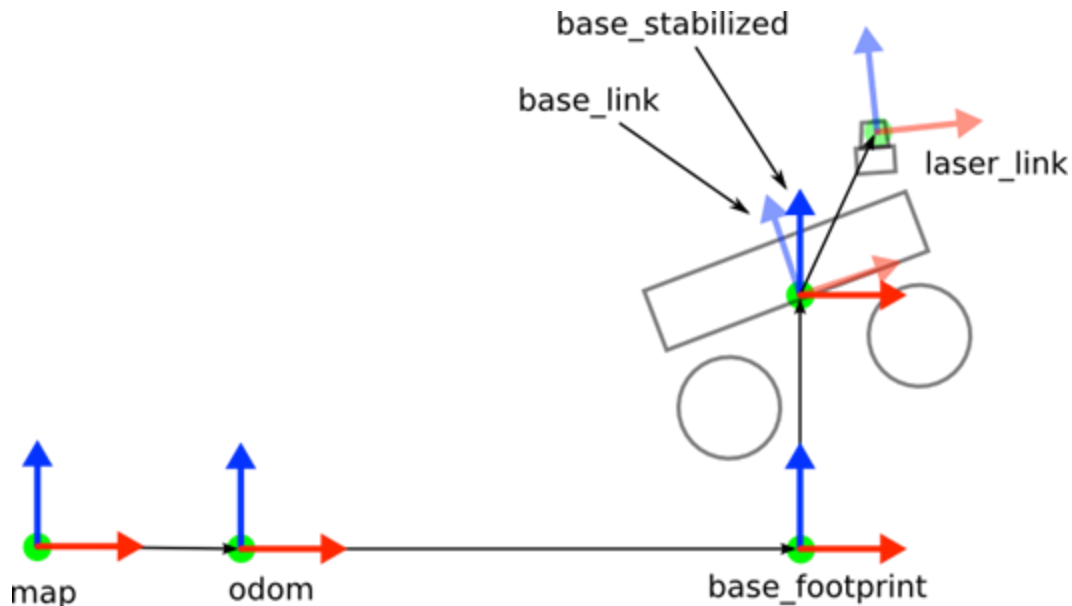
Useful ROS facilities

- **Launch files:** XML file specifying the launch of multiple nodes
 - Loading of parameters
 - Remapping topic names, parameters, etc.
 - Multiple machine support
 - Command-line utility: roslaunch

```
<?xml version="1.0"?>
<launch>
  <arg name="map" default="$(find scout_maps)/isr8-v05cr.yaml"/>
  <param name="map" type="string" value="$(arg map)"/>
  <rosparam file="$(find scout_config)/mbot.yaml"/>
  <include file="amcl.launch"/>
  <node name="navigation" pkg="scout_navigation" type="navigator">
    <param name="~guidance_method" type="string" value="fmm"/>
    <param name="~platform_mode" type="string" value="omni"/>
  </node>
</launch>
```

Useful ROS facilities

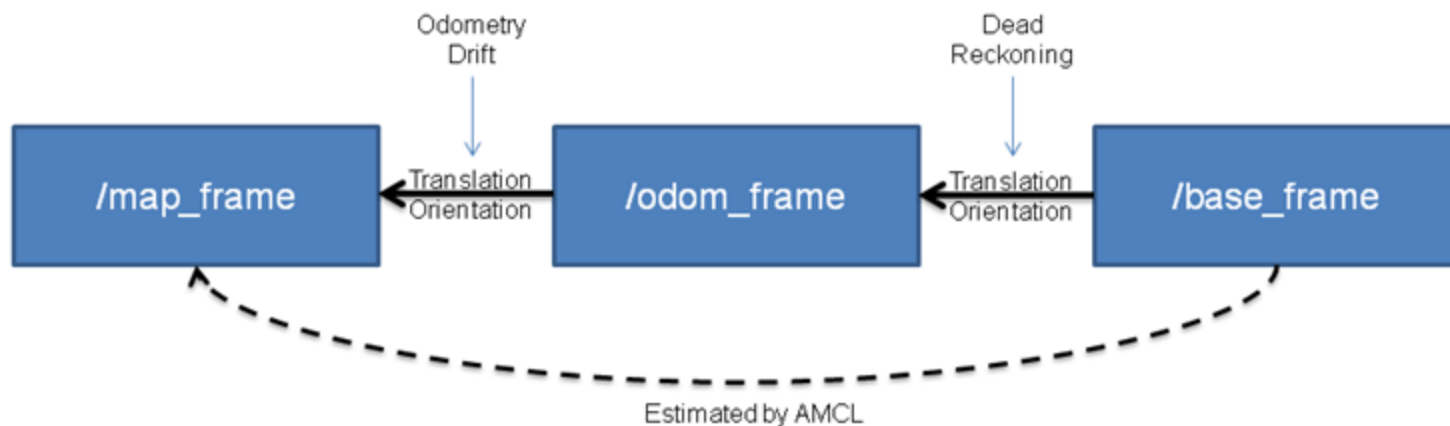
- **TF** framework: represents geometric transformations in 3D, position and orientation (6-DoF)



Useful ROS facilities

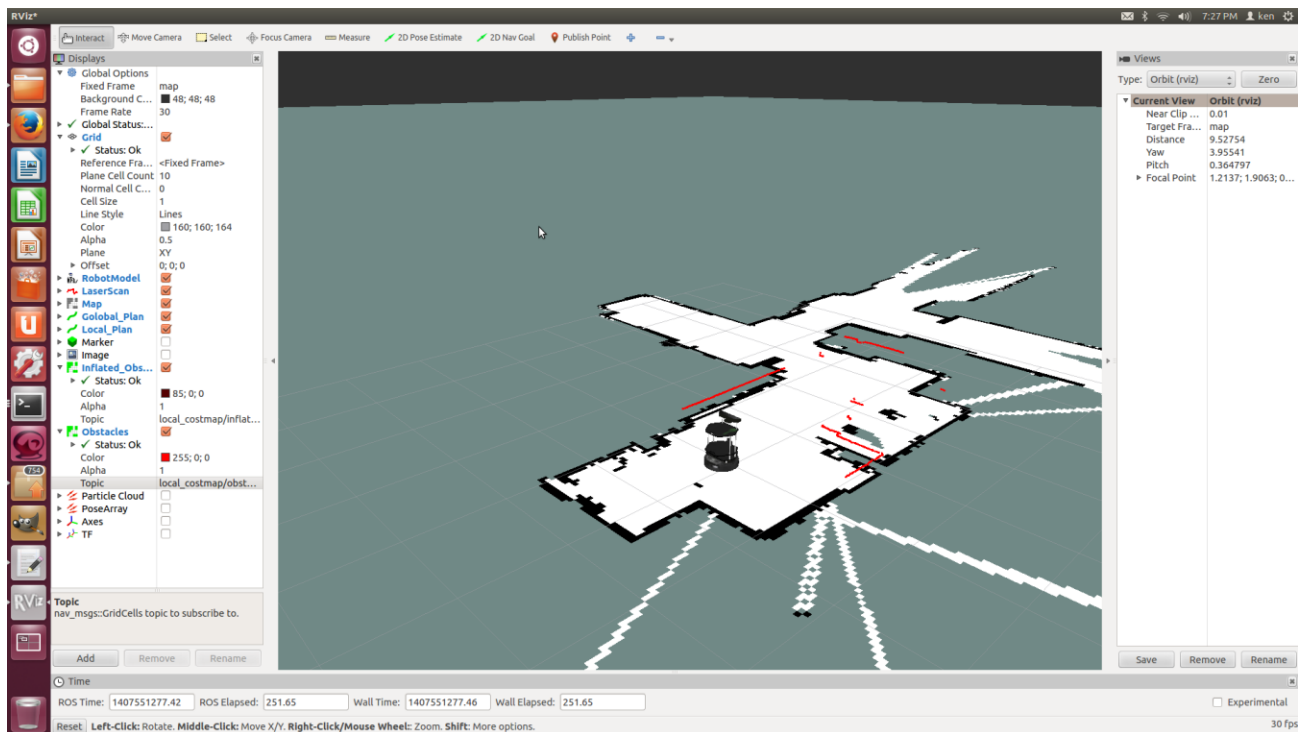
- **TF** framework: *de facto* standard frame assignment:

AMCL Map Localization



Useful ROS facilities

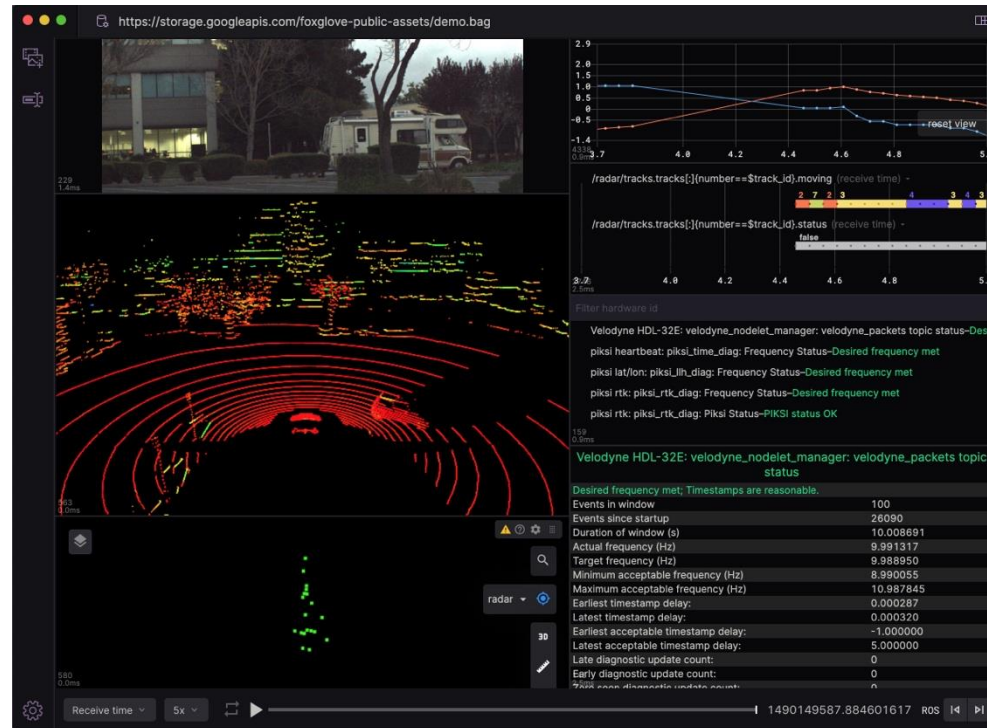
- **RVIZ:** visualisation framework



Useful ROS facilities

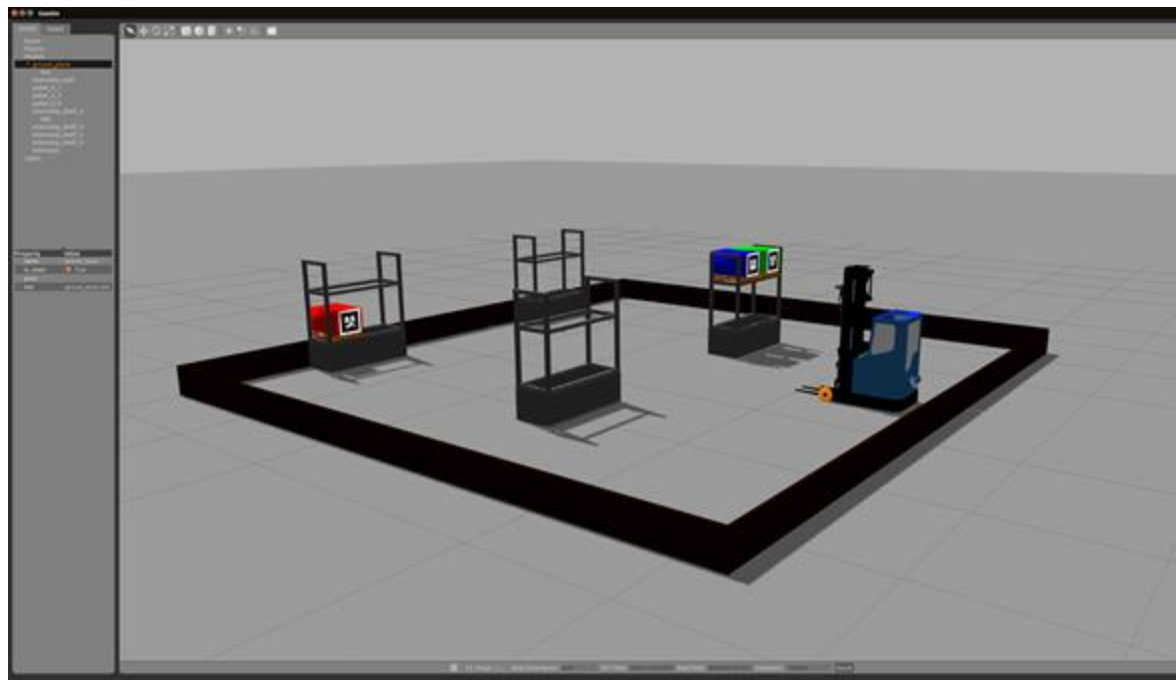
- **Foxglove:** an improved visualisation framework (third-party)

URL:
<https://foxglove.dev/>



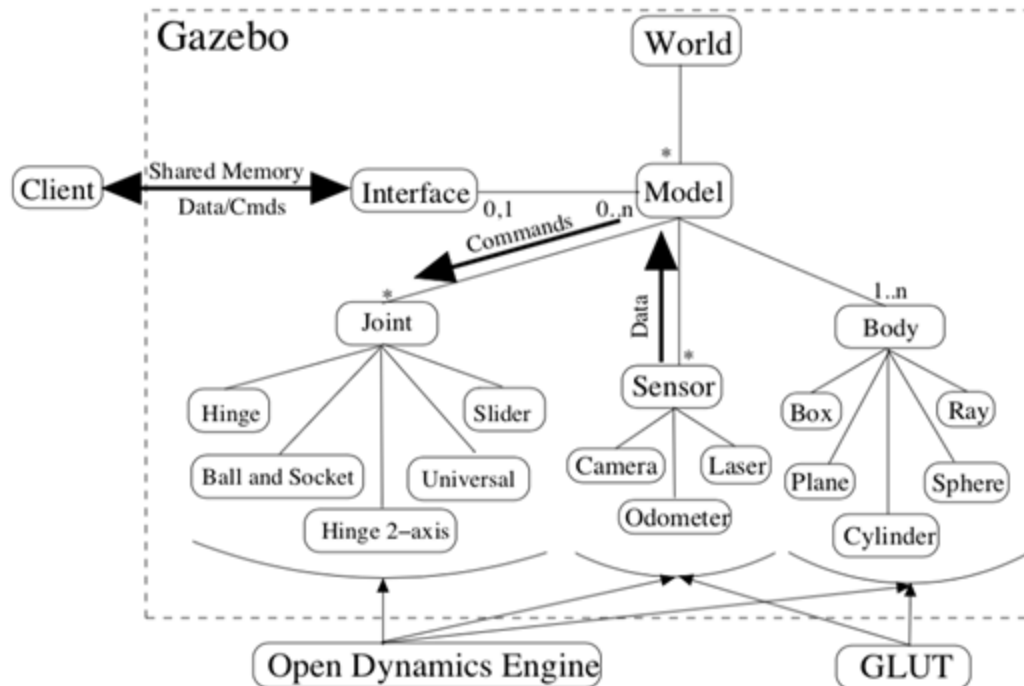
Useful ROS facilities

- **Gazebo:** physics simulation framework



Useful ROS facilities

- **Gazebo:** physics simulation framework

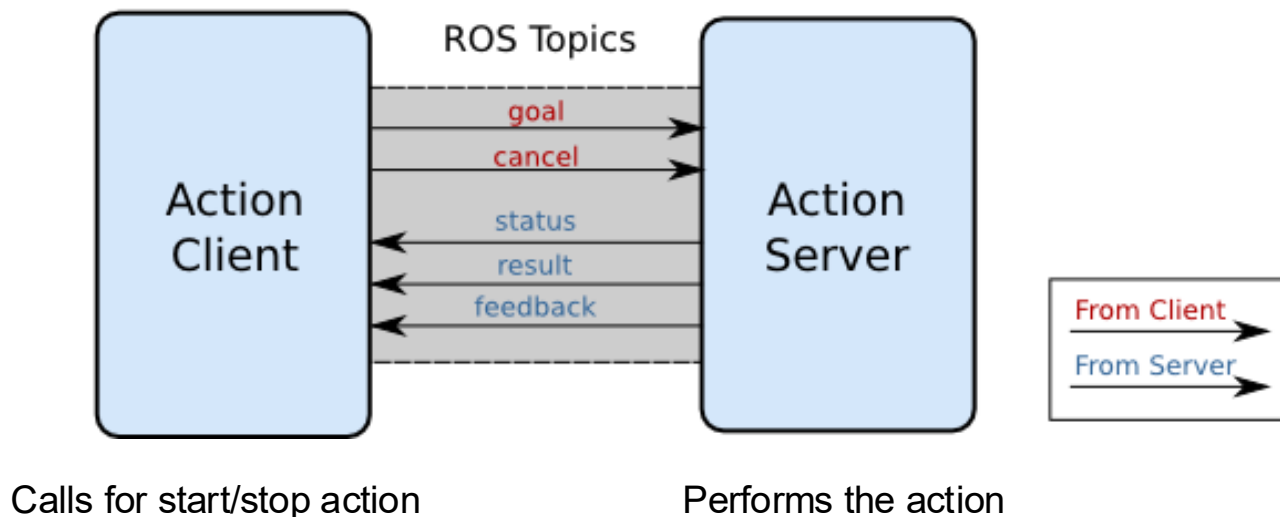


NOTE: client and server may run on different machines.

Useful ROS facilities

- **Actionlib** framework: state-full scheme to manage action execution

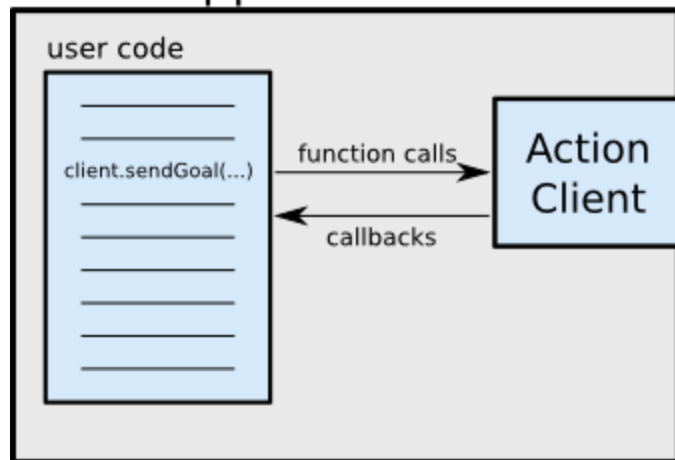
Action Interface



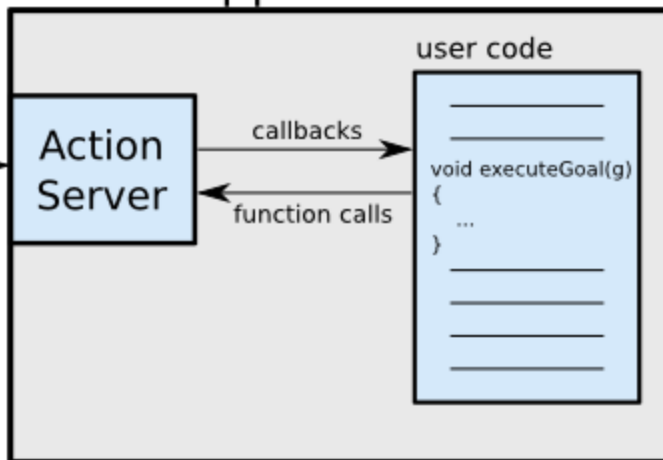
Useful ROS facilities

- **Actionlib** framework: state-full scheme to manage action execution

Client Application



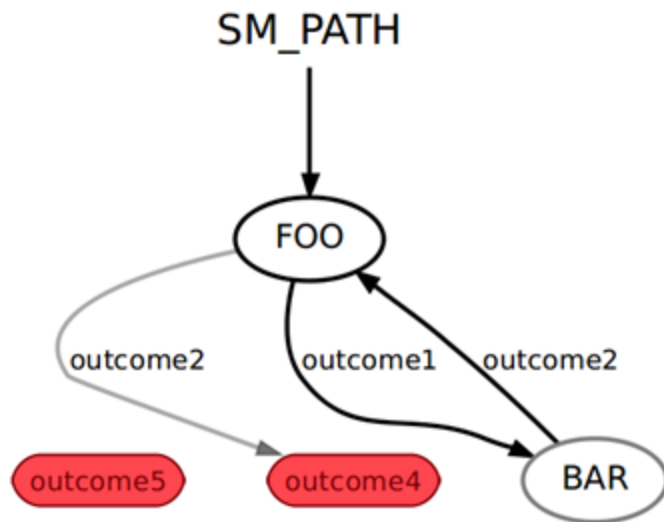
Server Application



ROS

Useful ROS facilities

- **SMACH** framework: FSM executor fully integrated into ROS
Ingredients: states, transitions, and outcomes



Useful ROS facilities

- **SMACH** framework:
 - Types of states:
 - MonitorState -- subscribes to topic, waits while condition True
 - ConditionState -- polls a callback function, waits until True
 - SimpleActionState -- calls actionlib action and can be a container
 - Types of containers:
 - StateMachine -- finite state machine
 - Concurrence -- all states run in parallel (split/join logic)
 - Sequence -- StateMachine with linear sequence of states

Useful ROS packages

- Other off-the-shelf packages:
 - **Gmapping**: creates occgrid maps from laser data
 - **AMCL**: localizes on occgrid maps using laser data
 - **Cartographer**: real-time SLAM in 2D and 3D
 - **Move_base**: path planning and guidance with obstacle avoidance using laser data
 - **Movelit**: trajectory planner for robotic arms
 - **Octomap**: creates 3D occupancy maps using RGB-D
 - **ROSPlan**: integrates classical planner into ROS