

MSAI Robotics course

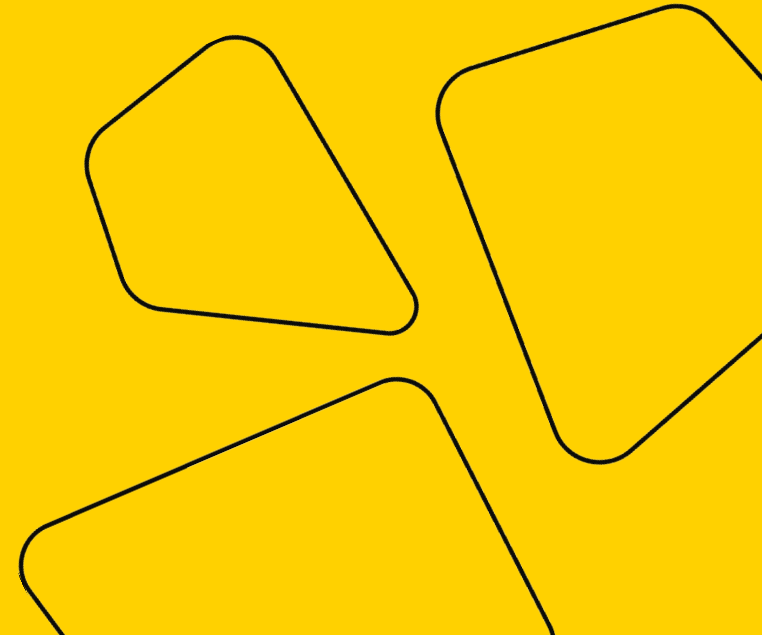
Seminar 1

Intro to Robot Operating System (ROS)

Vladislav Goncharenko
MIPT, 2022



**girafe
ai**



Outline

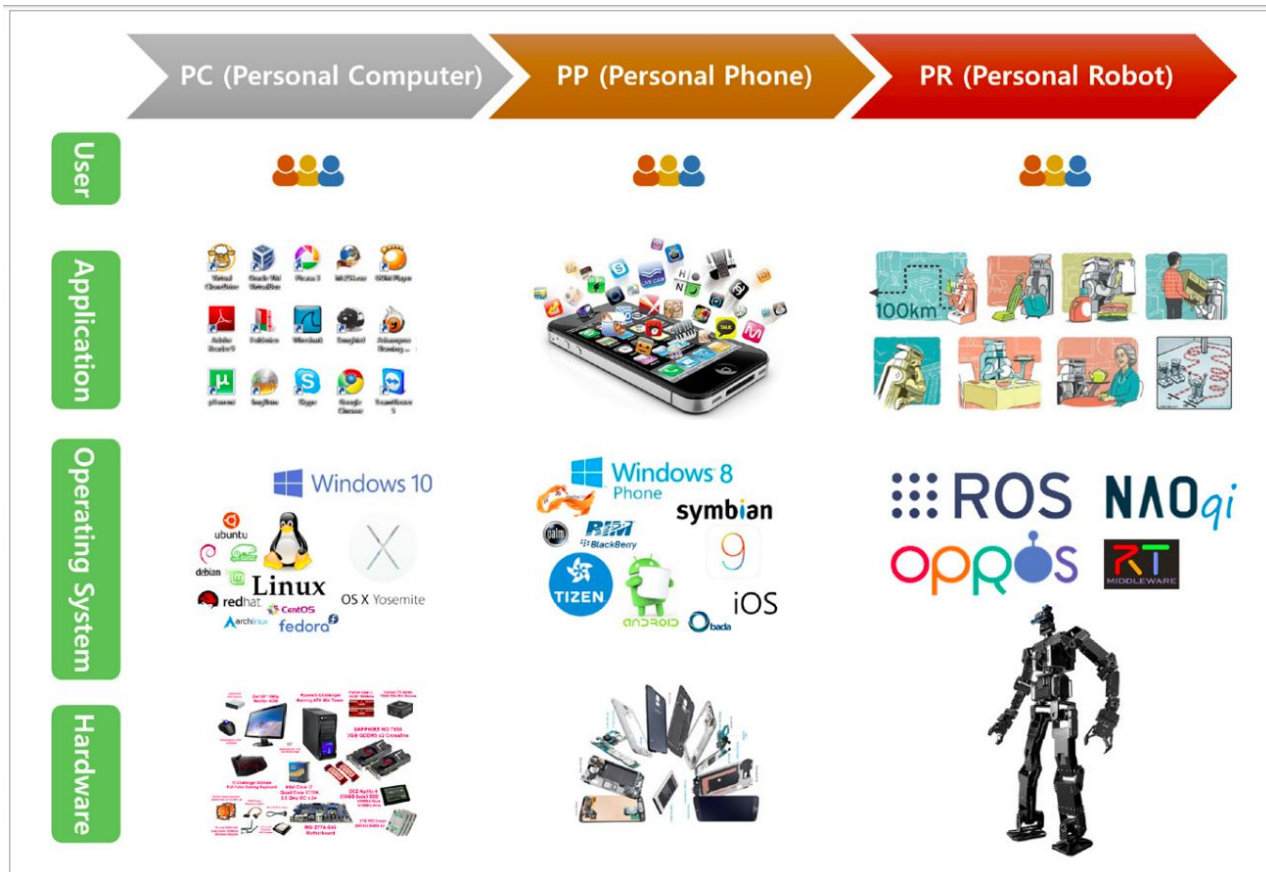
1. Why robots need an operating system?
2. What is ROS?
 - a. The history of ROS development
 - b. Distributions
 - c. Website navigation
3. ROS installation
 - a. ROS from container
4. First launch of ROS
 - a. Source...
 - b. Roscore
 - i. Rosmaster
 - ii. Parameter Server
 - iii. rosout
5. Turtlesim demo

Why robots need an operating system?

girafe
ai

01

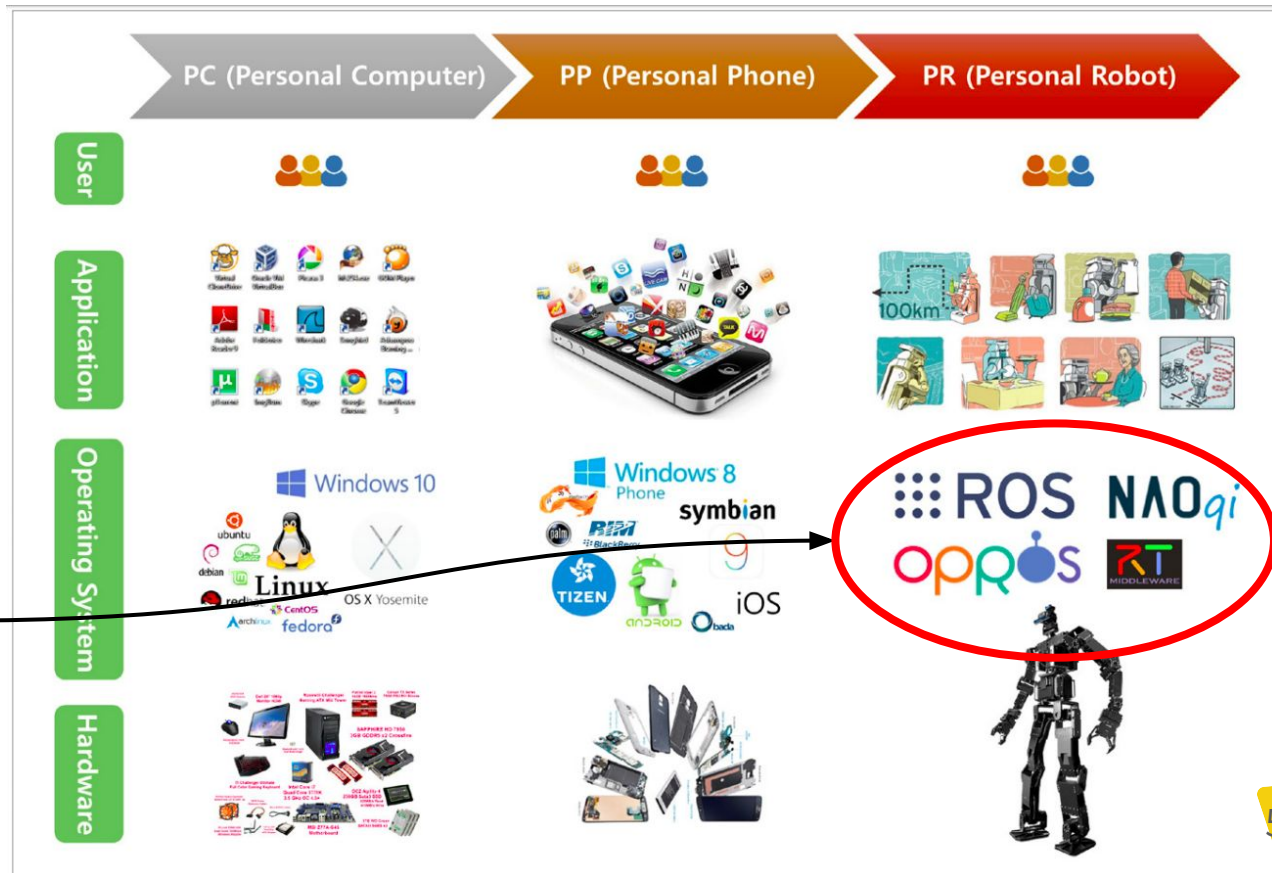
WHY ROBOTS NEED OS?



Source: ROS Robot Programming. YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

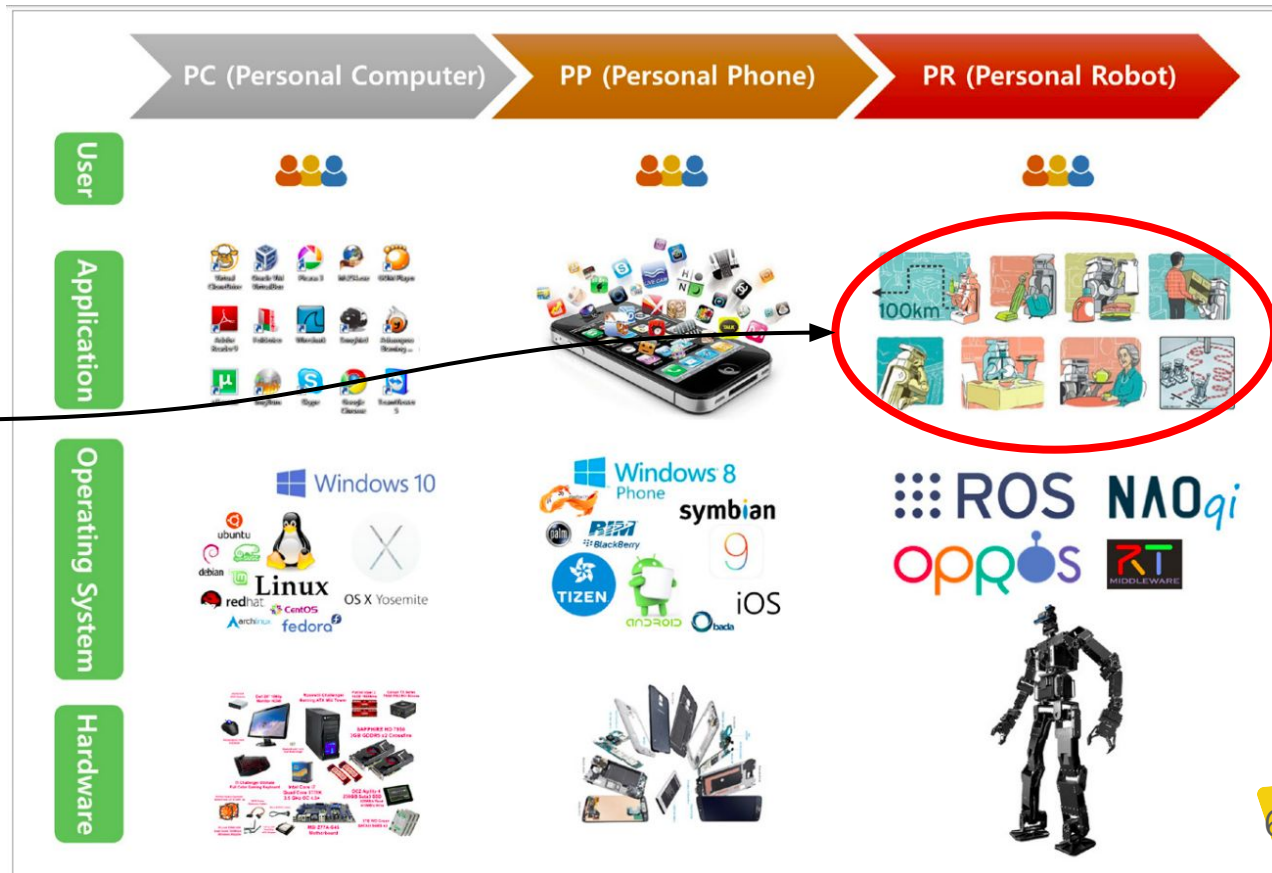
WHY ROBOTS NEED OS?

OS provide an abstraction from hardware and unifies the interprocess communication



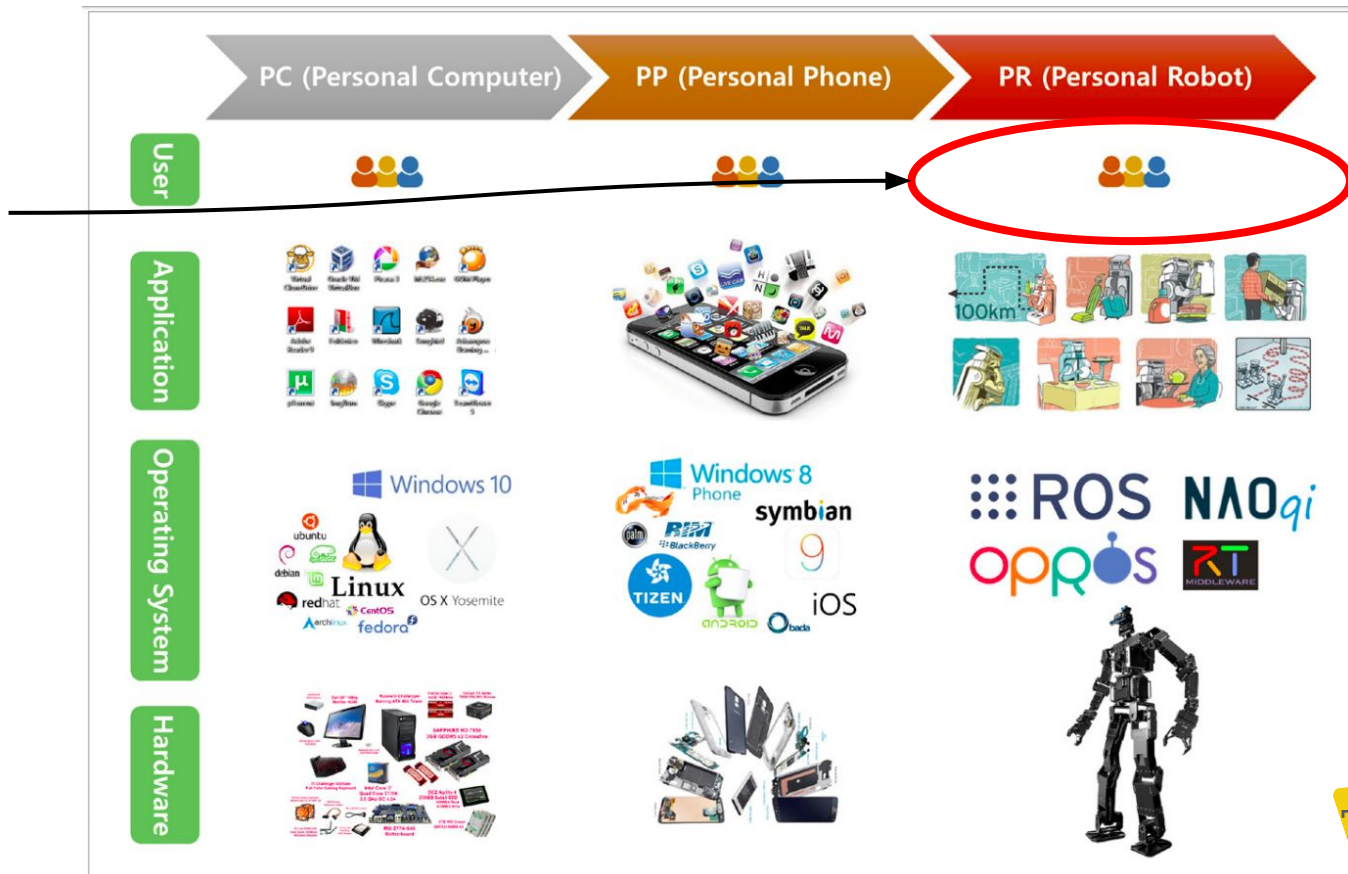
WHY ROBOTS NEED OS?

App developers can concentrate on realizing “new” features



WHY ROBOTS NEED OS?

Users get big variety of apps working on a single physical device



PROS OF OS IN ROBOTICS

- ❑ Abstraction from hardware
- ❑ Reuse of whole programmes and software parts
- ❑ Modularity due to the unified paradigm of interprocess data transmission
- ❑ Available development and debugging tools
- ❑ Community

EXISTING OS / FRAMEWORKS FOR ROBOTS

- ❑ **MSRDS**, Microsoft Robotics Developer Studio
- ❑ **ERSP**, Evolution Robotics Software Platform, Evolution Robotics
- ❑ **ROS**, Robot Operating System, Open Robotics
- ❑ **OpenRTM**, National Institute of Adv. Industrial Science and Technology (AIST)
- ❑ **NAOqi OS**, SoftBank and Aldebaran
- ❑ ...



NAOqi



What is ROS?

girafe
ai

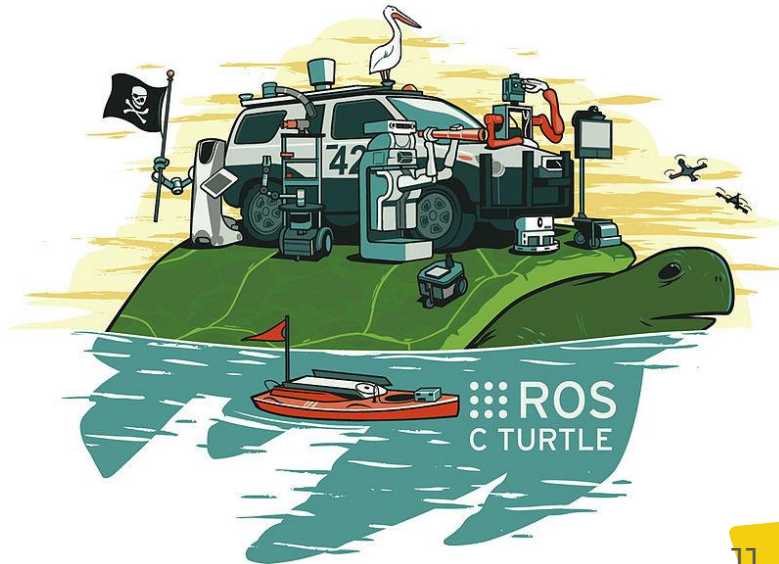
02

ROBOT OPERATING SYSTEM

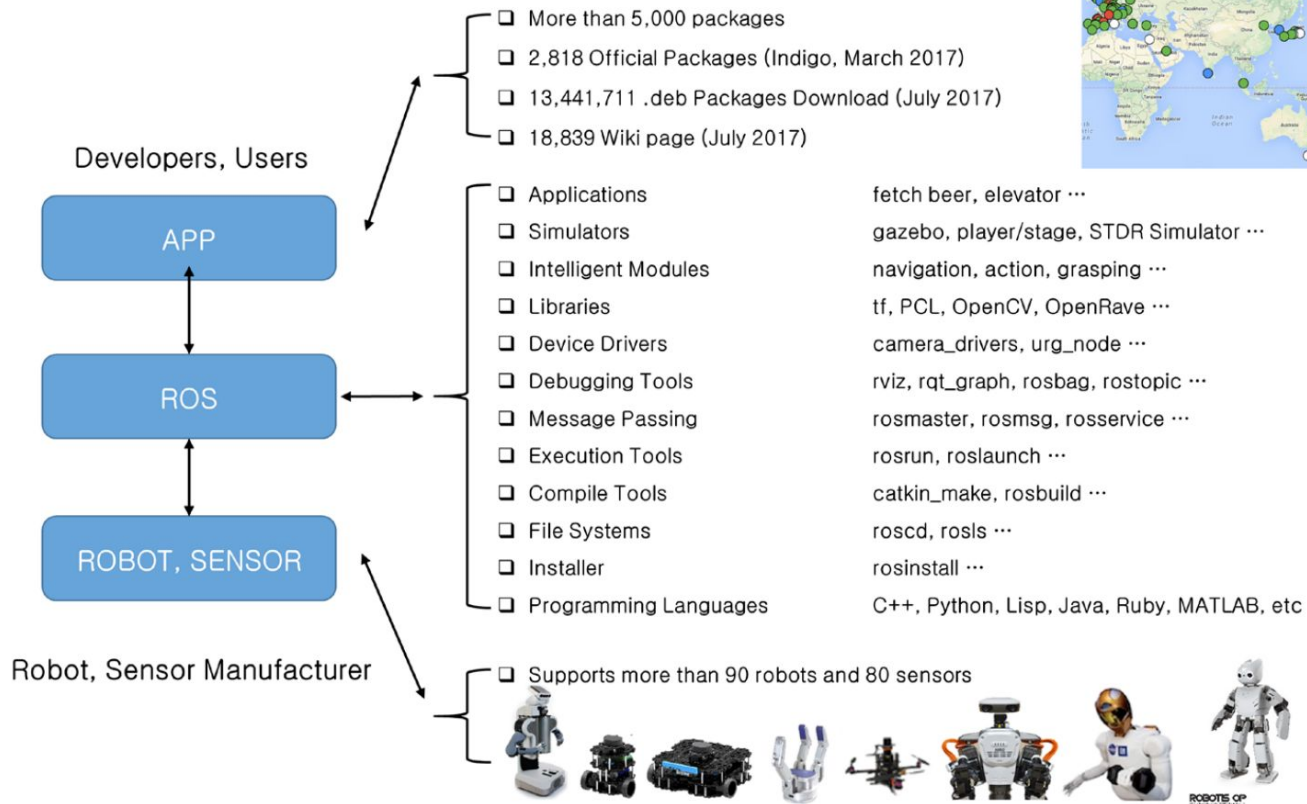
Robot Operating System, ROS — freely distributed meta-operating system for robots. ROS provides standard operating system services:

- ❑ hardware abstraction
- ❑ low-level device control
- ❑ ready to use realizations of frequently used functions
- ❑ interprocess communication
- ❑ software packages management

ROS



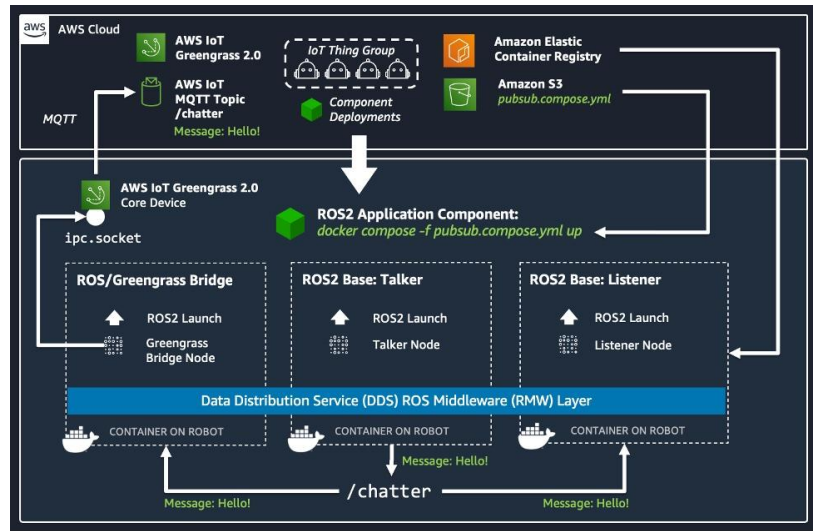
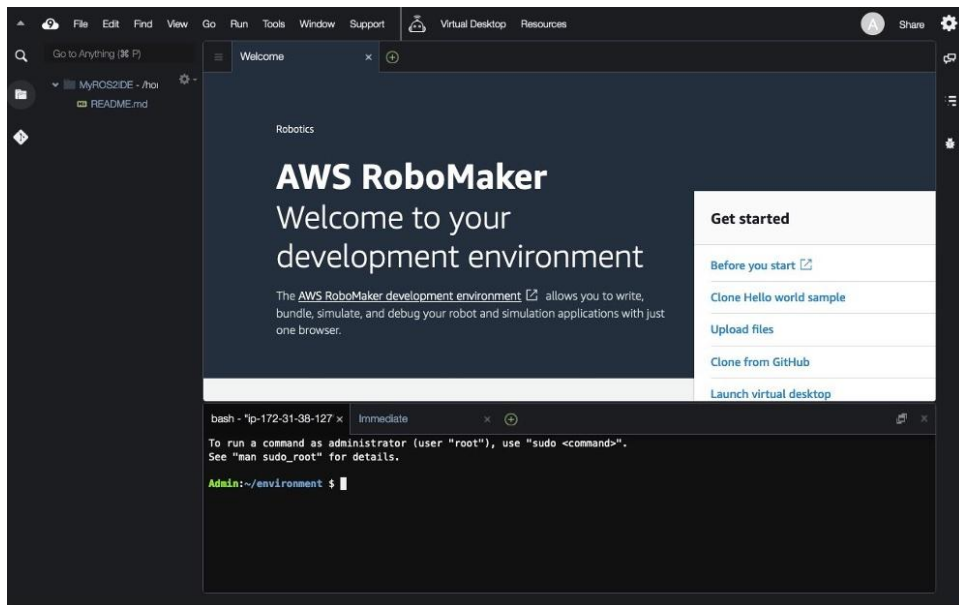
ROS ECOSYSTEM



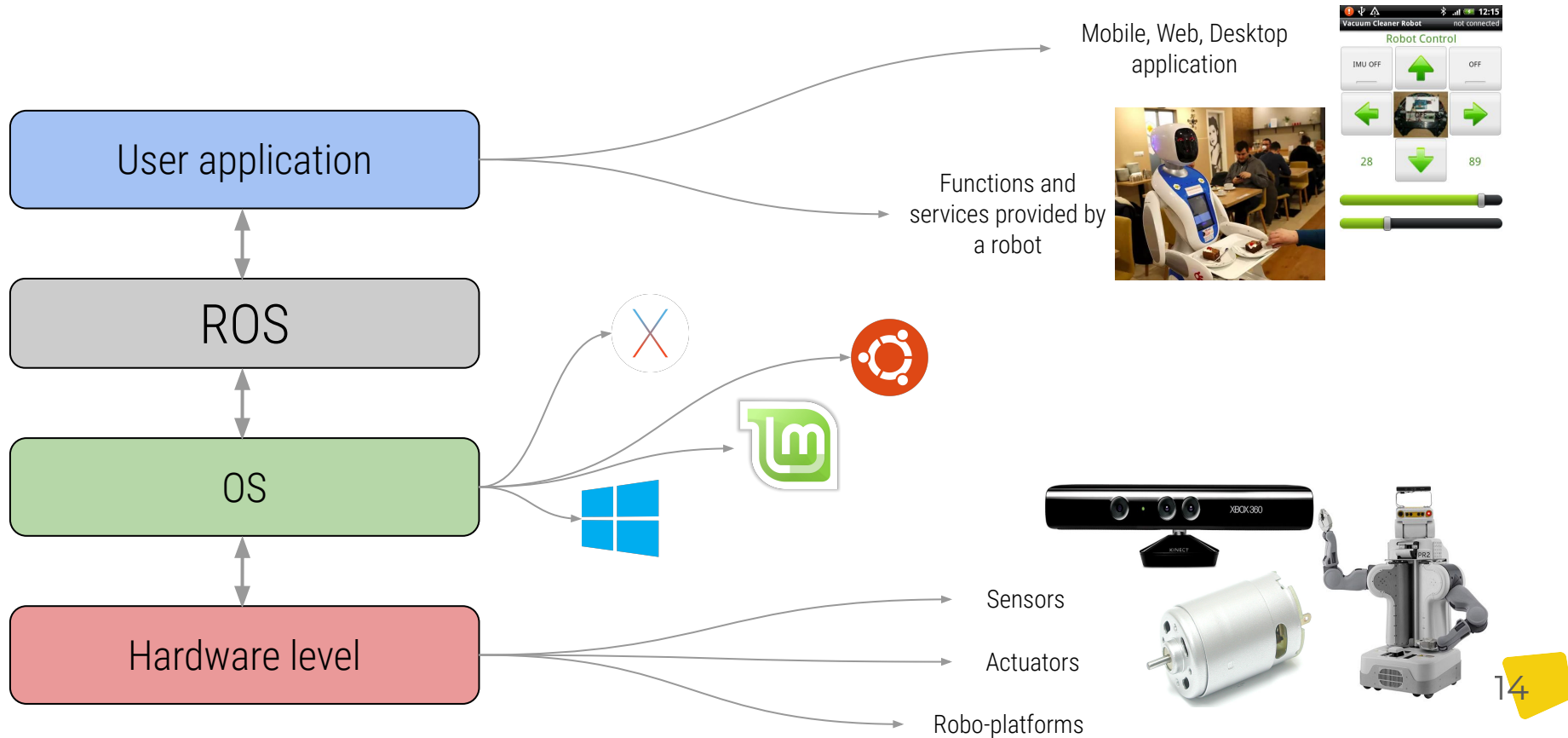
ROS SUPPORT

[AWS supports automated builds of ROS environments](#) with it's Greengrass service

See [one more tutorial](#)



META-OPERATING SYSTEM



ROS COMPONENTS

Client Layer

roscpp	rospy	roslisp	rosjava	roslibjs
--------	-------	---------	---------	----------

Robotics Application

Movel!	navigatioin	executive smach	descartes	rospex
teleop pkgs	rocon	mapviz	people	ar track

Robotics Application Framework

dynamic reconfigure	robot localization	robot pose ekf	Industrial core	robot web tools	ros realtime	mavros
tf	robot state publisher	robot model	ros control	calibration	octomap mapping	
vision opencv	image pipeline	laser pipeline	perception pcl	laser filters	ecto	

Communication Layer

common msgs	rosbag	actionlib	pluginlib	rostopic	rosservice
roswnode	roslaunch	rosparm	rosmaster	rosout	ros console

Hardware Interface Layer

camera drivers	GPS/IMU drivers	joystick drivers	range finder drivers	3d sensor drivers	diagnostics
audio common	force/torque sensor drivers	power supply drivers	rosterial	ethernet drivers	ros canopen

Software Development Tools

RViz	rqt	wstool	rospack	catkin	roscdep
------	-----	--------	---------	--------	---------

Simulation

gazebo ros pkgs	stage ros
-----------------	-----------

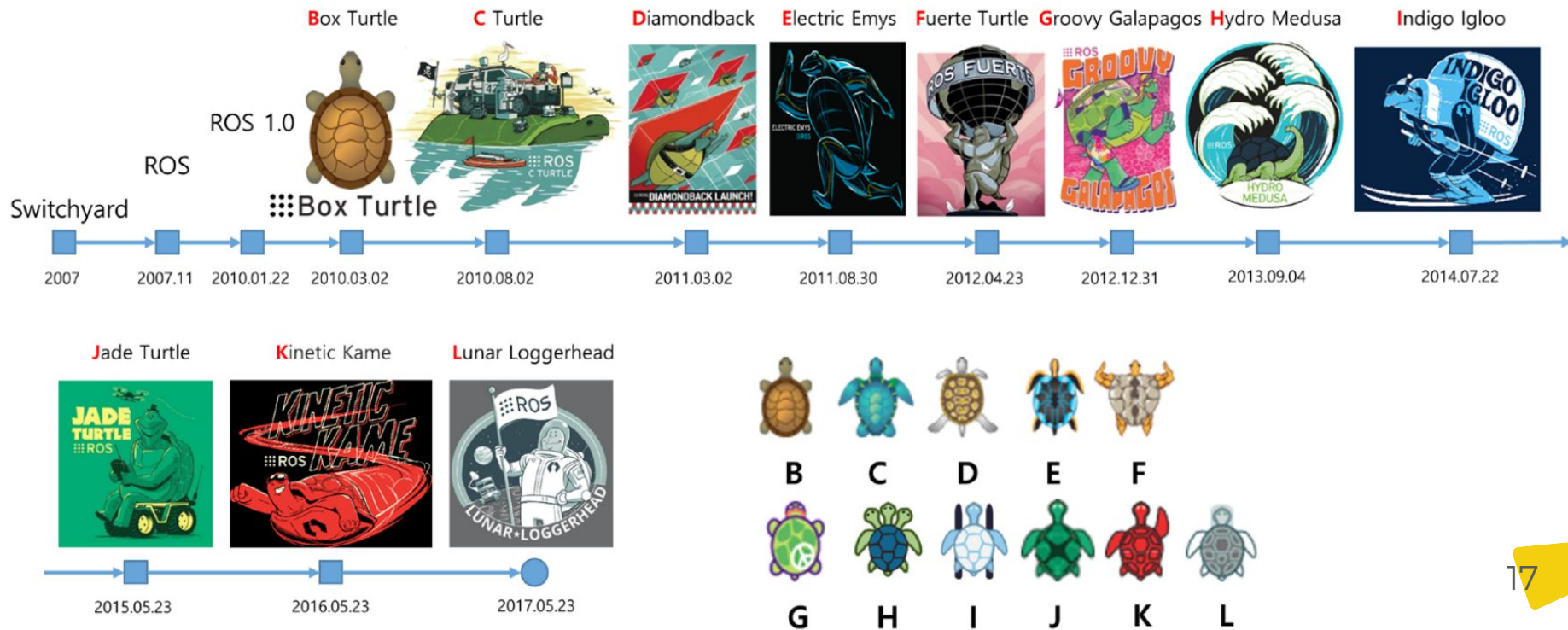
ROS HISTORY

- ❑ 2007, Switchyard in Stanford. Before the appearance of ROS, Stanford had several framework prototypes for robots. For the experiments they used, the Stanford Artificial Intelligence Robot (STAIR) and the Personal Robotics (PR) program.
- ❑ 2007, Willow Garage - robotic incubator supports ROS development.
- ❑ 2010, ROS 1.0



ROS DISTRIBUTIONS

<http://wiki.ros.org/Distributions>








ROS 2

https://design.ros2.org/articles/why_ros2.html

Why ROS 2?

- ❑ New challenges that were not faced by the developers of ROS 1:
 - ❑ Multi-robot systems
 - ❑ Embedded platforms
 - ❑ Real-time systems
 - ❑ Data transmission problems (latency, safety...)
 - ❑ Use in commercial products
 - ❑ Data protection
 - ❑ ...
- ❑ The emergence of new technologies
- ❑ Take into account past mistakes
- ❑ ...

Distro	Release date	Logo	EOL date
Eloquent Elusor	Nov 22nd, 2019		Nov 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		Dec 2019
Bouncy Bolson	July 2nd, 2018		Jul 2019
Ardent Apalone	December 8th, 2017		Dec 2018
beta3	September 13th, 2017		Dec 2017
beta2	July 5th, 2017		Sep 2017
beta1	December 19th, 2016		Jul 2017
alpha1 - alpha8	August 31th, 2015		Dec 2016

ROS WEBSITE NAVIGATION

The image shows a screenshot of the ROS.org website with several annotations. At the top left is the ROS.org logo, consisting of three blue dots followed by the text "ROS.org". To the right of the logo are three links: "About", "Mailing Lists", and "code.ros.org". Further right is a search box with the label "Search:" and two buttons labeled "Text" and "Titles". Below these elements is a dark blue navigation bar with three main sections: "Documentation", "Browse Software", and "News". Each section has a corresponding annotation box with a red line pointing to it.

ROS.org

[About](#) | [Mailing Lists](#) | [code.ros.org](#)

Search:

Documentation

The Documentation and the ROS.org links will take you to the ROS landing page with the list of relevant links.

Browse Software

The Browse Software link will display a list of all the software packages in ROS, along with a brief description.

News

The News link will keep you informed about the latest happening with ROS.

ROS WEBSITE NAVIGATION

This is the tf package page

tf

This is a list of all packages in the geometry stack

geometry: [angles](#) / [bullet](#) / [eigen](#) / [kdl](#) / [tf](#) / [tf_conversions](#)

tf is part of the geometry stack

Package Summary

TF is a package that lets the user keep track of multiple coordinate frames over time. TF maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

Author: Tully Foote

License: BSD

This is the tf package header that is auto generated from the package manifest.xml.

Package Links:

[Code API](#)

[Msg/Srv API](#)

[Tutorials](#)

[Troubleshooting](#)

[Reviews](#) (API cleared)

[Dependency Tree](#)

The package sidebar contains links to API documentation, tutorials, troubleshooting, and reviews specific to each package.

ROS installation

girafe
ai

03

ROS INSTALLATION

<https://www.ros.org/>

<http://wiki.ros.org/melodic/Installation/Ubuntu>

1. Add <http://packages.ros.org> to the list of “sources” from where software packages could be installed (to source.list)
2. Add security keys
3. Install ROS packages:
 - ❑ **Desktop-Full Install:** ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators and 2D/3D sensor data processing packages
 - ❑ `$ sudo apt install ros-<distribution name>-desktop-full`
 - ❑ **Desktop Install:** ROS, rqt, rviz, и robot-generic libraries
 - ❑ `$ sudo apt install ros-<distribution name>-desktop`
 - ❑ **ROS-Base:** ROS package, build, и communication libraries. Without GUI tools
 - ❑ `$ sudo apt install ros-<distribution name>-base`
 - ❑ **Separate packages**
 - ❑ `$ sudo apt install ros-<distribution name>-<package name>`

ROS LAUNCH IN DOCKER CONTAINER

<http://wiki.ros.org/docker/Tutorials/Docker>

https://hub.docker.com/_/ros

1. **Create Dockerfile** and describe in it the installation and configuration of the necessary packages
2. **Create** an **Image** from Dockerfile:
 - ❑ `$ docker build --tag <image name> .`
3. **Run container** from created image:
 - ❑ `$ docker run -it <image name>`

To access the screen from the container (required to run graphical applications from the container), you can run as follows:

```
$ docker run -e DISPLAY=unix$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix -it  
<image name>
```

- ❑ If an error like this occurs at startup:

```
No protocol specified
```

```
Error: cannot open display: unix:0.0
```

- ❑ You need to allow local non-network connections on the host computer:

```
$ xhost +local:
```

First launch of ROS

girafe
ai

04

BASIC TERMS

<http://wiki.ros.org/ROS/Concepts>

Node — a unit of program code that runs in a separate thread and performs a specific computational function. Nodes communicate with each other through **topics**. The collection of all running nodes and topics forms a **ROS-graph**.

Topic — data transmission system with a publish / subscribe mechanism. A node sends data by publishing it to a topic, and reads it by subscribing to a topic. Multiple nodes can publish data to one topic and read data from one topic. One node can both subscribe and publish many topics.

Message type — description of the format of the message published in the topic. Each topic publishes messages of a certain type.

ROS-graph — a set of all nodes and topics published in the system.

Package — a unit of software organization in ROS. The package can contain the program code of the nodes, the description of the topic messages, configuration and other files related by a common meaning.

SOURCE A.K.A NEWBIE NIGHTMARE

- ❑ Necessary step before **each** ROS session (performed in each newly opened terminal window):

```
$ source /opt/ros/<ROS distribution name>/setup.bash
```

In our case <ROS distribution name> = melodic

- ❑ What **\$ source ...** does?
 - ❑ Sets environment variables required for ROS to run
- ❑ To avoid running the above command each time, it is often added to the `~ / .bashrc` (on Linux systems) file, which causes it to run automatically for every new terminal

```
$ echo "source /opt/ros/<ROS distribution name>/setup.bash" >> ~/.bashrc
```

ROSCORE LAUNCH

<http://wiki.ros.org/roscore>

Roscore is a set of nodes and programs required to run any ROS application. **Roscore** must be running in order for the nodes to communicate. **Roscore** is started with the command:

- ❑ `$ roscore`

You can also specify the port on which **master** will run:

- ❑ `$ roscore -p 1234`

roscore launches:

- ❑ rosmaster
- ❑ Parameter Server
- ❑ rosout — logging node

roscore always starts automatically if launched via **roslaunch** and has not been launched before.

The nodes that are launched when **roscore** is launched are defined in **roslaunch** / **roscore.xml** (**note** that it is not recommended to change `roscore.xml`, as this will affect all subsequent ROS launches).

ROSCORE LAUNCH

- ❑ Where is the current ROS session being logged

- ❑ Where is roslaunch server running

- ❑ What parameters are available on the **ROS Parameter Server**

- ❑ Where **rosmaster** is launched

- ❑ Launching the **rosout** node

```
roscore http://devel-Latitude-5491:11311/
roscore http://devel-Latitude-5491:11311/ 71x40
~ roscore
... logging to /home/shipitko/.ros/log/97e47afc-7cda-11ea-9a3e-185680878a02/roslaunch-devel-Latitude-5491-28870.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://devel-Latitude-5491:38865/
ros_comm version 1.12.14

SUMMARY
=====

PARAMETERS
* /rostdistro: kinetic
* /rosversion: 1.12.14

NODES

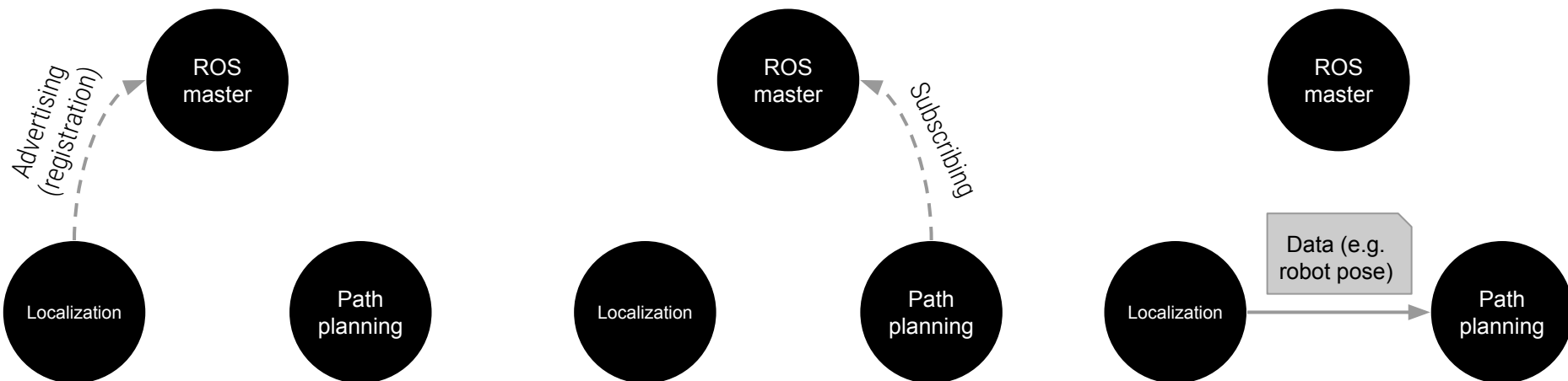
auto-starting new master
process[master]: started with pid [28880]
ROS_MASTER_URI=http://devel-Latitude-5491:11311/

setting /run_id to 97e47afc-7cda-11ea-9a3e-185680878a02
process[rosout-1]: started with pid [28893]
started core service [/rosout]
```

ROS MASTER

<http://wiki.ros.org/Master>

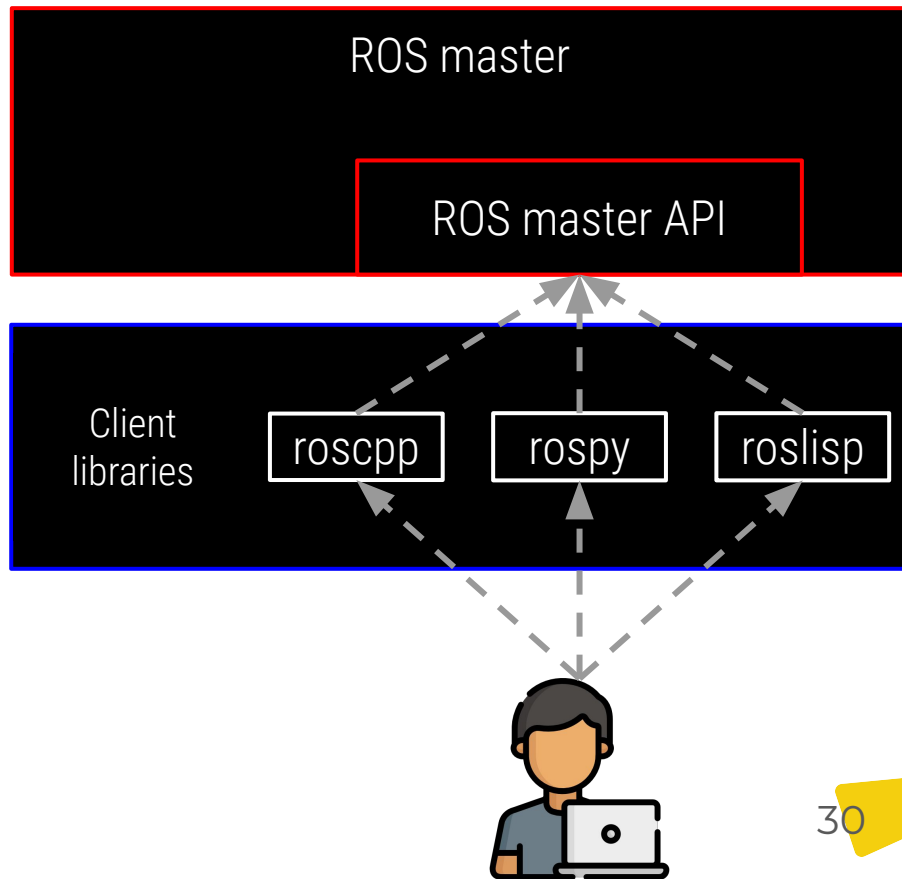
- ❑ Provides registration of topics and services. It can be compared to a DNS server - by the name of the topic / service, it provides its URI.
- ❑ Provides a **Parameter Server**



ROS MASTER

http://wiki.ros.org/ROS/Master_API

- ❑ **rosmaster** provides an API for registering / unregistering topics and services, as well as for getting a list of running nodes, registered topics, etc.
- ❑ The **client libraries** use the **rosmaster API** and provide the developer with a simple mechanism for creating nodes, publishing topics, etc.
- ❑ **Important!** Use the **rosmaster API** directly only if you are implementing support for a new programming language. The list of supported languages can be found here: <http://wiki.ros.org/Client%20Libraries>



PARAMETER SERVER

<http://wiki.ros.org/Parameter%20Server>

Parameter Server — a parameter dictionary available to all nodes in the system. Used to store various parameters and access them in real time. Runs inside **rosmaster**.

Parameters on the server:

/camera/left/name: leftcamera
/camera/left/exposure: 1
/camera/right/name: rightcamera
/camera/right/exposure: 1.1

Request: /camera/left/name

Response:

leftcamera

Request: /camera/left

Response:

name: leftcamera
exposure: 1

Request: /camera

Response:

left: { name: leftcamera, exposure: 1 }
right: { name: rightcamera, exposure: 1.1 }



PARAMETER SERVER

<http://wiki.ros.org/Parameter%20Server>

Data types supported by **Parameter Server**:

- ❑ 32-bit integers
- ❑ booleans
- ❑ strings
- ❑ doubles
- ❑ iso8601 dates
- ❑ lists
- ❑ base64-encoded binary data

Parameters are accessed through client libraries (**roscpp**, **rospy**, ...) as well as the **rosparam** command line tool. Both methods will be discussed later in the course.

ROSOUT

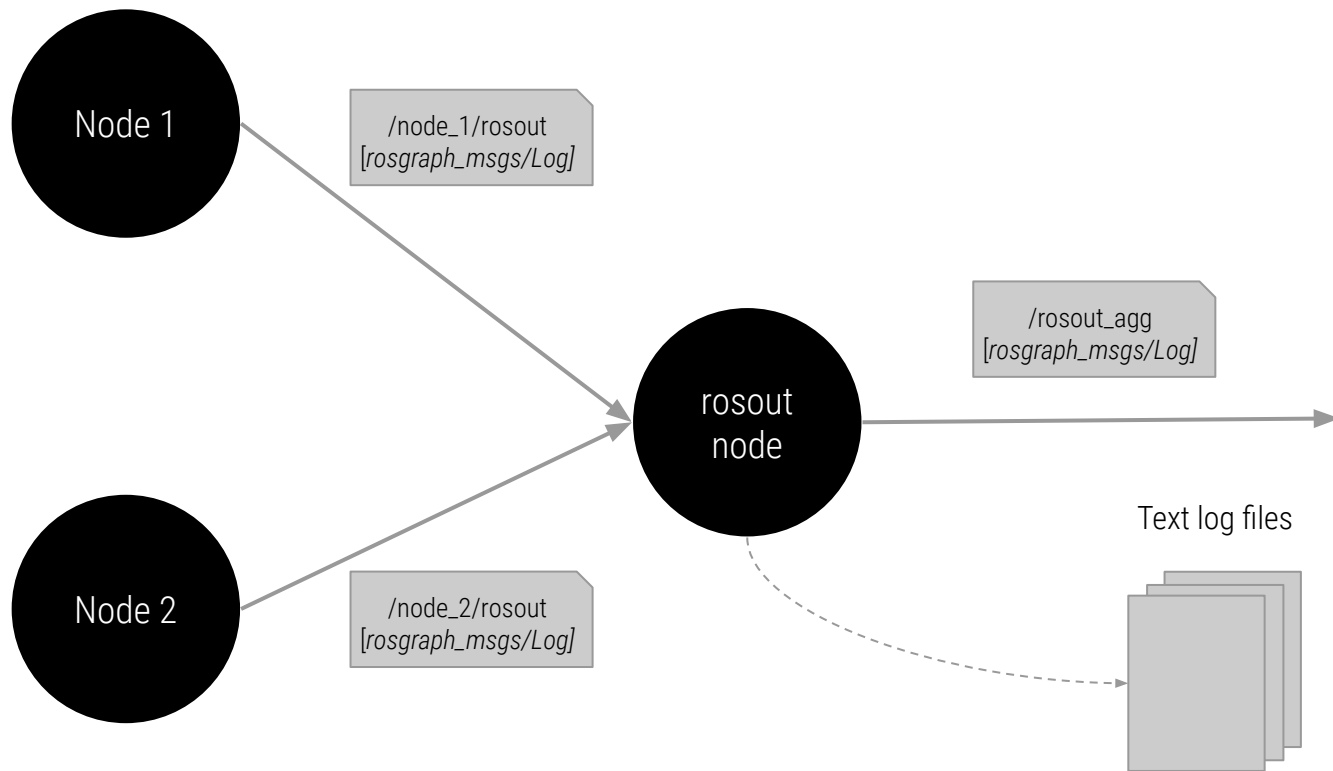
<http://wiki.ros.org/rosout>

The term **rosout** refers to several entities:

1. **Node rosout.** *It subscribes to the log messages of each node (<node namespace> / rosout), saves them to a text log file, and also duplicates them in the topic / rosout_agg.*
2. **Topic / rosout.** *Standard topic for publishing log messages.*
3. **Topic / rosout_agg. Contains aggregated log messages from all nodes.**
4. **The rosgraph_msgs / Log message type.** *Used by topics / rosout and / rosout_agg.*
5. **API of client libraries** *for working with rosout logs in different programming languages.*

ROSOUT

<http://wiki.ros.org/rosout>



Turtlesim demo

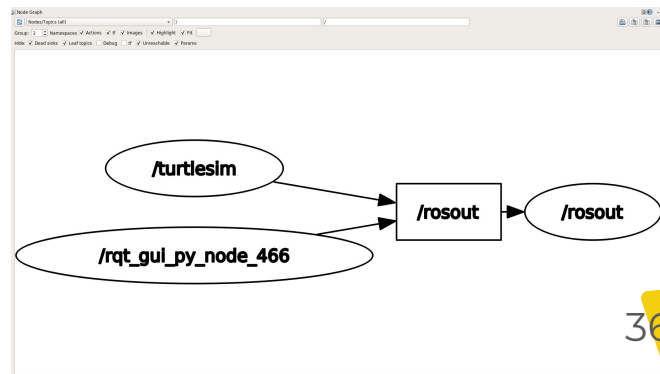
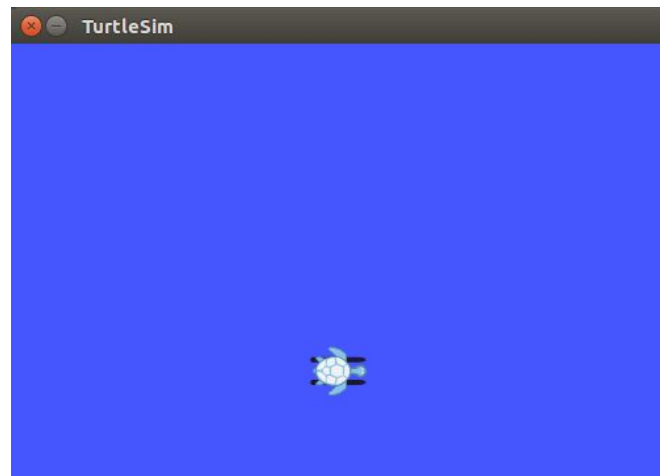
girafe
ai

05

ROS “HELLO (TURTLE) WORLD!”

<http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes>

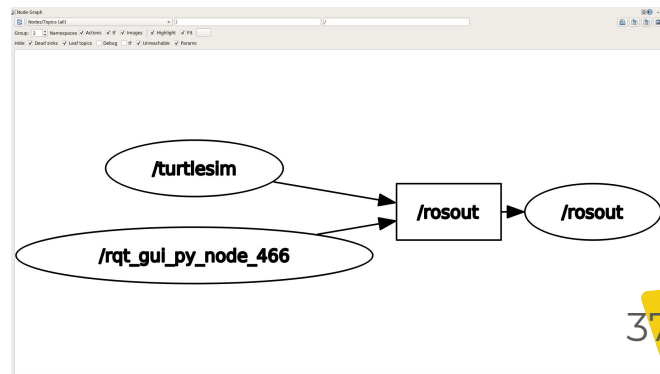
- ❑ Set environment variables
`$ source /opt/ros/<дистрибутив>/setup.bash`
- ❑ Run **roscore** in the background
`$ roscore &`
- ❑ Run **turtlesim_node** from the **turtlesim** package
`$ rosrune turtlesim turtlesim_node`
- ❑ Enjoying the resulting **ROS-graph**
`$ rqt_graph`
- ❑ Launch **turtle_teleop_key** node
`$ rosrune turtlesim turtle_teleop_key`
- ❑ Control the simulation by pressing the "arrows" and look again at the resulting **ROS-graph**



ROS “HELLO (TURTLE) WORLD!”

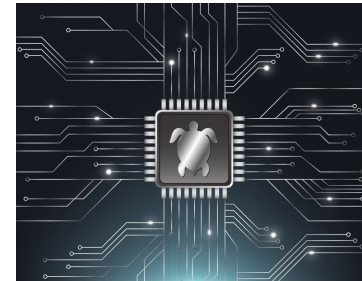
<http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes>

- ❑ Check running nodes and existing topics
 - \$ rostopic list
 - \$ rosnodetop list
- ❑ Check the type of the message published in the topic **/turtle1/pose**
 - \$ rostopic info /turtle1/pose
 - \$ rosmmsg info turtlesim/Pose
- ❑ Take a look at the data published in the topic **/turtle1/pose**
 - \$ rostopic echo /turtle1/pose



ADDITIONAL RESOURCES

1. Book: ROS Robot Programming.
YoonSeok Pyo, HanCheol Cho,
RyuWoon Jung, TaeHoon Lim
2. ROS Official Tutorials
3. Clearpath Robotics ROS Tutorial
4. The history of ROS creation



ROS
Robot Programming

From the basic concept to practical programming and robot application

A Handbook Written by TurtleBot3 Developers
YoonSeok Pyo | HanCheol Cho | RyuWoon Jung | TaeHoon Lim

Thanks for attention!

Questions? Additions? Welcome!

girafe
ai

