



# Programming for Robotics

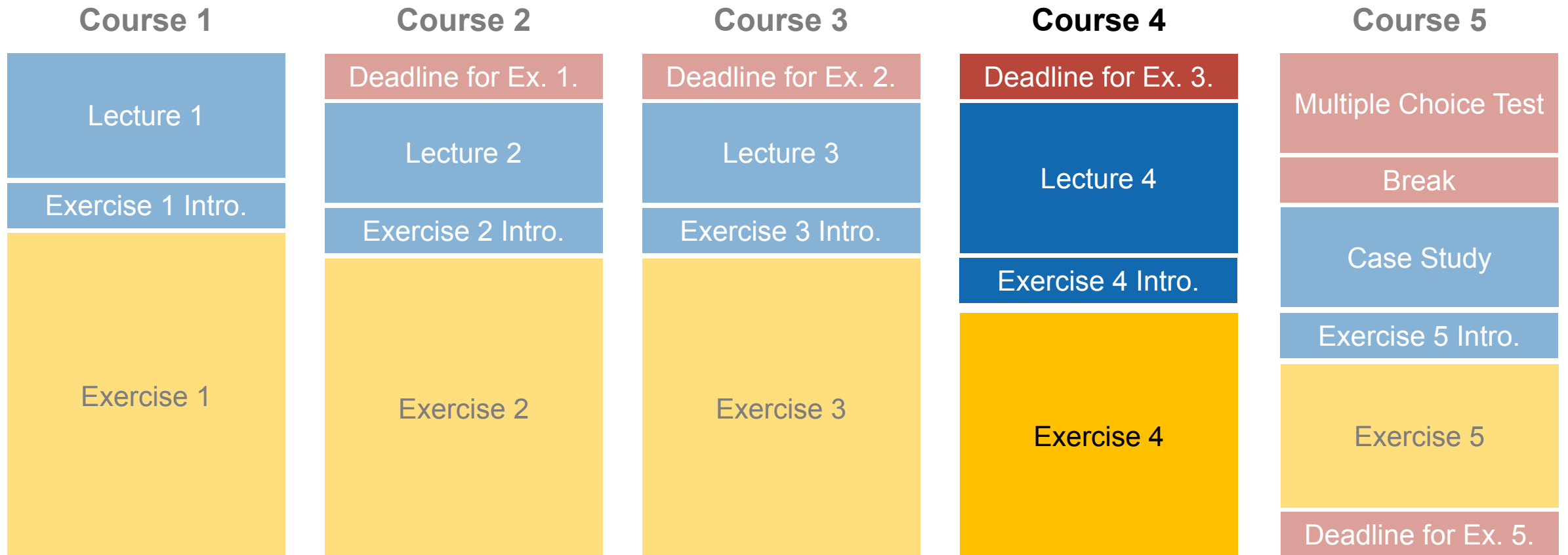
## Introduction to ROS

Course 4

Tom Lankhorst, Edo Jelavic  
Prof. Dr. Marco Hutter

 ROS

# Course Structure



## Evaluation – Multiple Choice Test

- The test counts for 50 % of the final grade
- The multiple choice test (~40 min) takes place at the last course day:

**05.03.2021 at 08:00 (sharp, not 08:15), HG E1**

## Ex 4 grading session

- Moved to Thursday

**04.03.2021 at 08:00 (sharp, not 08:15)**

# Overview Course 4

- ROS services
  - ROS actions (actionlib)
  - ROS time
  - ROS bags
  - Debugging strategies
- 
- Intro to ROS 2 concepts

# ROS Services

- Request/response communication between nodes is realized with *services*
  - The *service server* advertises the service
  - The *service client* accesses this service
- Similar in structure to messages, services are defined in *\*.srv* files

List available services with

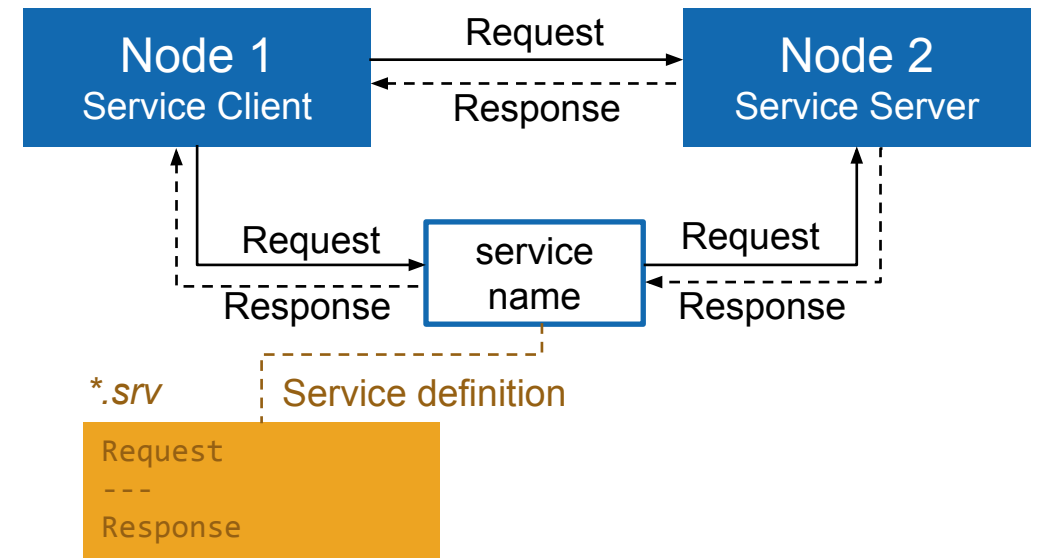
```
> rosservice list
```

Show the type of a service

```
> rosservice type /service_name
```

Call a service with the request contents, autocomplete with `tab`

```
> rosservice call /service_name args
```



**More info**  
[wiki.ros.org/Services](http://wiki.ros.org/Services)

# ROS Services

## Examples

std\_srvs/Trigger.srv

```
---  
bool success  
string message
```

Request

Response

nav\_msgs/GetPlan.srv

```
geometry_msgs/PoseStamped start  
geometry_msgs/PoseStamped goal  
float32 tolerance  
---  
nav_msgs/Path plan
```

# ROS Service Example

Starting a *roscore* and a *add\_two\_ints\_server* node

## In console nr. 1:

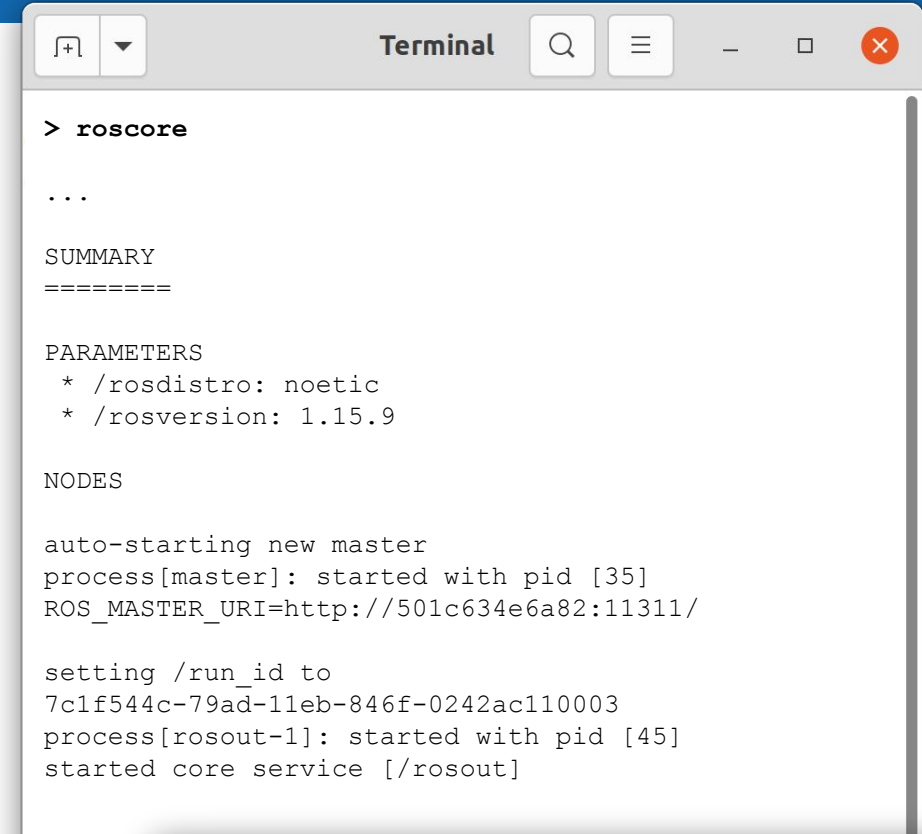
Start a roscore with

```
> roscore
```

## In console nr. 2:

Run a service demo node with

```
> rosrunc roscpp_tutorials add_two_ints_server
```

A terminal window titled "Terminal" with a search icon, a menu icon, and window control buttons. It displays the output of the 'roscore' command. The output includes a summary, parameters (roscdistro: noetic, rosversion: 1.15.9), and nodes (auto-starting new master, process[roscdistro]: started with pid [35], ROS\_MASTER\_URI=http://501c634e6a82:11311/, setting /run\_id to 7c1f544c-79ad-11eb-846f-0242ac110003, process[roscdistro-1]: started with pid [45], started core service [/roscdistro]).

```
> roscore

...

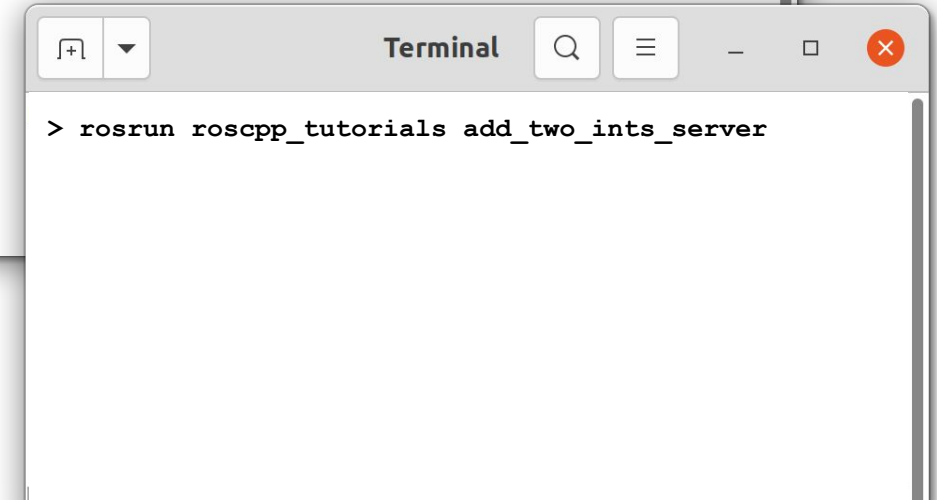
SUMMARY
=====

PARAMETERS
* /roscdistro: noetic
* /rosversion: 1.15.9

NODES

auto-starting new master
process[roscdistro]: started with pid [35]
ROS_MASTER_URI=http://501c634e6a82:11311/

setting /run_id to
7c1f544c-79ad-11eb-846f-0242ac110003
process[roscdistro-1]: started with pid [45]
started core service [/roscdistro]
```

A terminal window titled "Terminal" with a search icon, a menu icon, and window control buttons. It displays the command 'rosrunc roscpp\_tutorials add\_two\_ints\_server' entered at the prompt.

```
> rosrunc roscpp_tutorials add_two_ints_server
```



# ROS Service Example

## Console Nr. 3 – Analyze and call service

See the available services with

```
> rosservice list
```

See the type of the service with

```
> rosservice type /add_two_ints
```

Show the service definition with

```
> rossrv show roscpp_tutorials/TwoInts
```

Call the service (use Tab for auto-complete)

```
> rosservice call /add_two_ints "a: 10  
b: 5"
```



```
Terminal

> rosservice list
/add_two_ints
/add_two_ints_server/get_loggers
/add_two_ints_server/set_logger_level
/rosout/get_loggers
/rosout/set_logger_level

> rosservice type /add_two_ints
roscpp_tutorials/TwoInts

> rossrv show roscpp_tutorials/TwoInts
int64 a
int64 b
---
int64 sum

> rosservice call /add_two_ints "a: 10
b: 5"
sum: 15
```

tab

```
Terminal

> rosrn roscpp_tutorials add_two_ints_server
[ INFO] [1614524882.675833648]: request: x=10, y=5
[ INFO] [1614524882.676700815]: sending back
response: [15]
```

# ROS C++ Client Library (*roscpp*)

## Service Server

- Create a service server with

```
ros::ServiceServer service =
nodeHandle.advertiseService(service_name,
callback_function);
```
- When a service request is received, the callback function is called with the request as argument
- Fill in the response to the response argument
- Return to function with true to indicate that it has been executed properly

### More info

[wiki.ros.org/roscpp/Overview/Services](http://wiki.ros.org/roscpp/Overview/Services)

*add\_two\_ints\_server.cpp* (use OO-approach in exercises)

```
#include <ros/ros.h>
#include <roscpp_tutorials/TwoInts.h>

bool add(roscpp_tutorials::TwoInts::Request &request,
         roscpp_tutorials::TwoInts::Response &response)
{
    response.sum = request.a + request.b;
    ROS_INFO("request: x=%ld, y=%ld", (long int)request.a,
              (long int)request.b);
    ROS_INFO(" sending back response: [%ld]",
              (long int)response.sum);
    return true;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "add_two_ints_server");
    ros::NodeHandle nh;
    ros::ServiceServer service =
nh.advertiseService("add_two_ints", add);
    ros::spin();
    return 0;
}
```

# ROS C++ Client Library (*roscpp*)

## Service Client

- Create a service client with

```
ros::ServiceClient client =
    nodeHandle.serviceClient<service_type>
        (service_name);
```

- Create service request contents  
service.request

- Call service with

```
client.call(service);
```

- Response is stored in service.response

### add\_two\_ints\_client.cpp

```
#include <ros/ros.h>
#include <roscpp_tutorials/TwoInts.h>
#include <cstdlib>

int main(int argc, char **argv) {
    ros::init(argc, argv, "add_two_ints_client");
    if (argc != 3) {
        ROS_INFO("usage: add_two_ints_client X Y");
        return 1;
    }

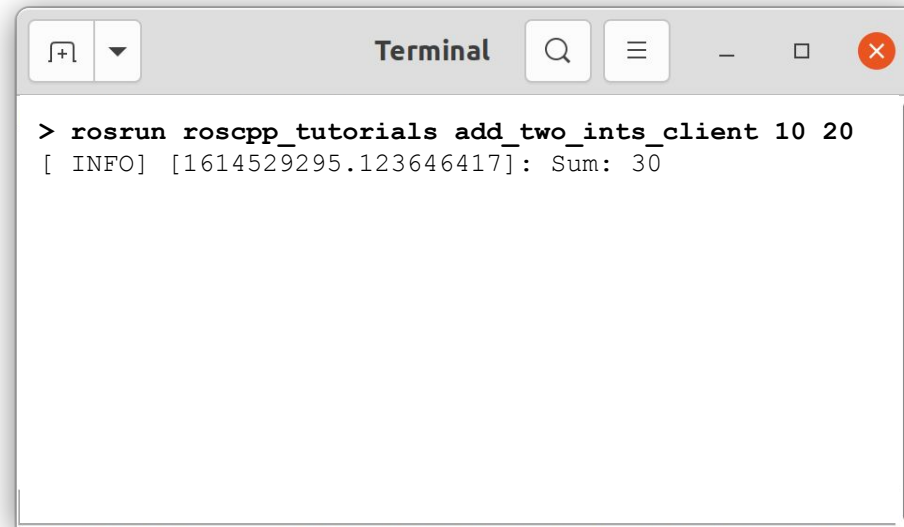
    ros::NodeHandle nh;
    ros::ServiceClient client =
        nh.serviceClient<roscpp_tutorials::TwoInts>("add_two_ints");
    roscpp_tutorials::TwoInts service;
    service.request.a = atoi(argv[1]);
    service.request.b = atoi(argv[2]);
    if (client.call(service)) {
        ROS_INFO("Sum: %ld", (long int)service.response.sum);
    } else {
        ROS_ERROR("Failed to call service add_two_ints");
        return 1;
    }
    return 0;
}
```

#### More info

[wiki.ros.org/roscpp/Overview/Services](http://wiki.ros.org/roscpp/Overview/Services)

# ROS C++ Client Library (*roscpp*)

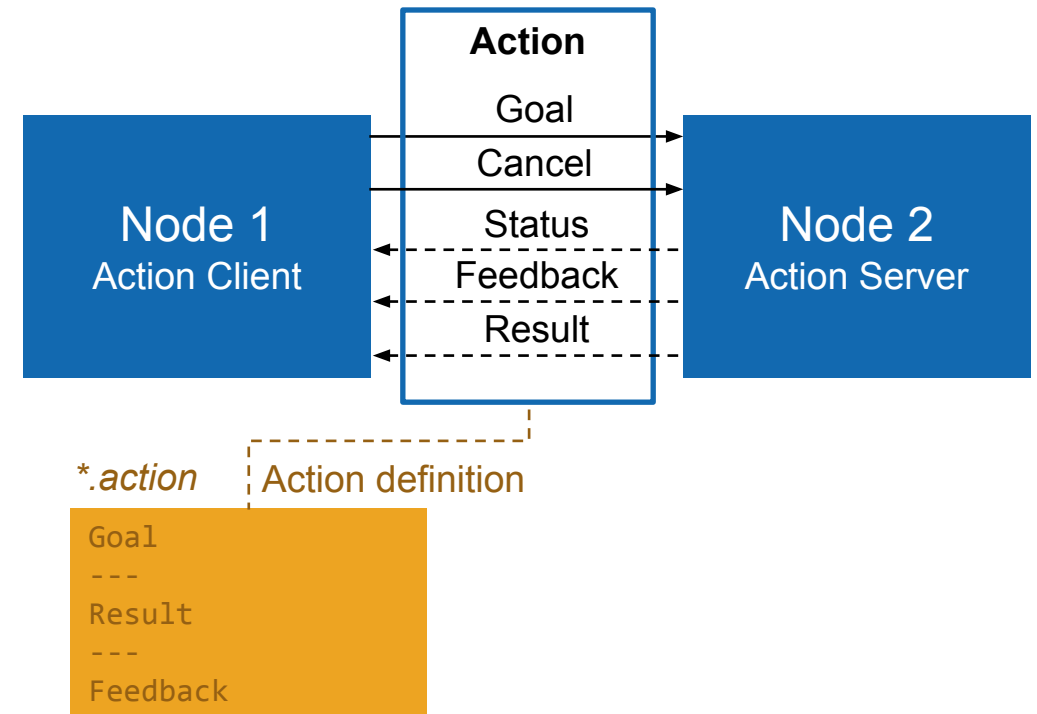
## Service Client

A terminal window titled "Terminal" with standard window controls (minimize, maximize, close) and search, menu, and zoom icons. The terminal shows a command being executed: `> rosrunc roscpp_tutorials add_two_ints_client 10 20`. The output is: `[ INFO] [1614529295.123646417]: Sum: 30`.

```
> rosrunc roscpp_tutorials add_two_ints_client 10 20
[ INFO] [1614529295.123646417]: Sum: 30
```

# ROS Actions (actionlib)

- Similar to service calls, but provide possibility to
  - Cancel the task (preempt)
  - Receive feedback on the progress
- Best way to implement interfaces to time-extended, goal-oriented behaviors
- Similar in structure to services, action are defined in \*.action files
- Internally, actions are implemented with a set of topics



**More info**

[wiki.ros.org/actionlib](http://wiki.ros.org/actionlib)

[wiki.ros.org/actionlib/DetailedDescription](http://wiki.ros.org/actionlib/DetailedDescription)

# ROS Actions (actionlib)

## Averaging.action

```
int32 samples
---
float32 mean
float32 std_dev
---
int32 sample
float32 data
float32 mean
float32 std_dev
```

Goal

Result

Feedback

## FollowPath.action

```
navigation_msgs/Path path
---
bool success
---
float32 remaining_distance
float32 initial_distance
```

# ROS Parameters, Dynamic Reconfigure, Topics, Services, and Actions Comparison

	Parameters	Dynamic Reconfigure	Topics	Services	Actions
<b>Description</b>	Global constant parameters	Local, changeable parameters	Continuous data streams	Blocking call for processing a request	Non-blocking, preemptable goal oriented tasks
<b>Application</b>	Constant settings	Tuning parameters	One-way continuous data flow	Short triggers or calculations	Task executions and robot actions
<b>Examples</b>	Topic names, camera settings, calibration data, robot setup	Controller parameters	Sensor data, robot state	Trigger change, request state, compute quantity	Navigation, grasping, motion execution

# ROS Time

- Normally, ROS uses the PC's system clock as time source (*wall time*)
- For simulations or playback of logged data, it is convenient to work with a simulated time (pause, slow-down etc.)
- To work with a simulated clock:
  - Set the `/use_sim_time` parameter

```
> rosparam set use_sim_time true
```
  - Publish the time on the topic `/clock` from
    - Gazebo (enabled by default)
    - ROS bag (use option `--clock`)

## More info

[wiki.ros.org/Clock](http://wiki.ros.org/Clock)

[wiki.ros.org/roscpp/Overview/Time](http://wiki.ros.org/roscpp/Overview/Time)

- To take advantage of the simulated time, you should always use the ROS Time APIs:

- **ros::Time**

```
ros::Time begin = ros::Time::now();  
double secs = begin.toSec();
```

- **ros::Duration**

```
ros::Duration duration(0.5); // 0.5s  
ros::Duration passed = ros::Time()::now()  
                        - begin;
```

- **ros::Rate**

```
ros::Rate rate(10); // 10Hz
```

- If wall time is required, use `ros::WallTime`, `ros::WallDuration`, and `ros::WallRate`



# ROS Bags

- A *bag* is a format for storing message data
- Binary format with file extension \*.bag
- Suited for logging and recording datasets for later visualization and analysis

Record all topics in a bag

```
> rosbag record --all
```

Record given topics

```
> rosbag record topic_1 topic_2 topic_3
```

Stop recording with Ctrl + C

Bags are saved with start date and time as file name in the current folder (e.g. 2019-02-07-01-27-13.bag)

Show information about a bag

```
> rosbag info bag_name.bag
```

Read a bag and publish its contents

```
> rosbag play bag_name.bag
```

Playback options can be defined e.g.

```
> rosbag play --rate=0.5 bag_name.bag
```

```
--rate=factor    Publish rate factor
--clock           Publish the clock time (set
                  param use_sim_time to true)
--loop            Loop playback
                  etc.
```

**More info**

[wiki.ros.org/rosbag/Commandline](http://wiki.ros.org/rosbag/Commandline)

# Debugging Strategies

## Debug with the tools you have learned

- Compile and run code often to catch bugs early
- Understand compilation and runtime error messages
- Use analysis tools to check data flow (rostopic info, rostopic echo, roswtf, rqt\_graph etc.)
- Visualize and plot data (RViz, RQT Multiplot etc.)
- Divide program into smaller steps and check intermediate results (ROS\_INFO, ROS\_DEBUG etc.)
- Make your code robust with argument and return value checks and catch exceptions
- Extend and optimize only once a basic version works
- If things don't make sense, clean your workspace

```
> catkin clean --all
```

## Learn new tools

- Build in *debug* mode and use GDB or Valgrind

```
> catkin config --cmake-args  
    -DCMAKE_BUILD_TYPE=Debug  
> catkin config --cmake-args  
    -DCMAKE_BUILD_TYPE=Release
```

- Use debugger breakpoints, e.g. in Eclipse
- Write unit tests and integration tests to discover regressions

### More info

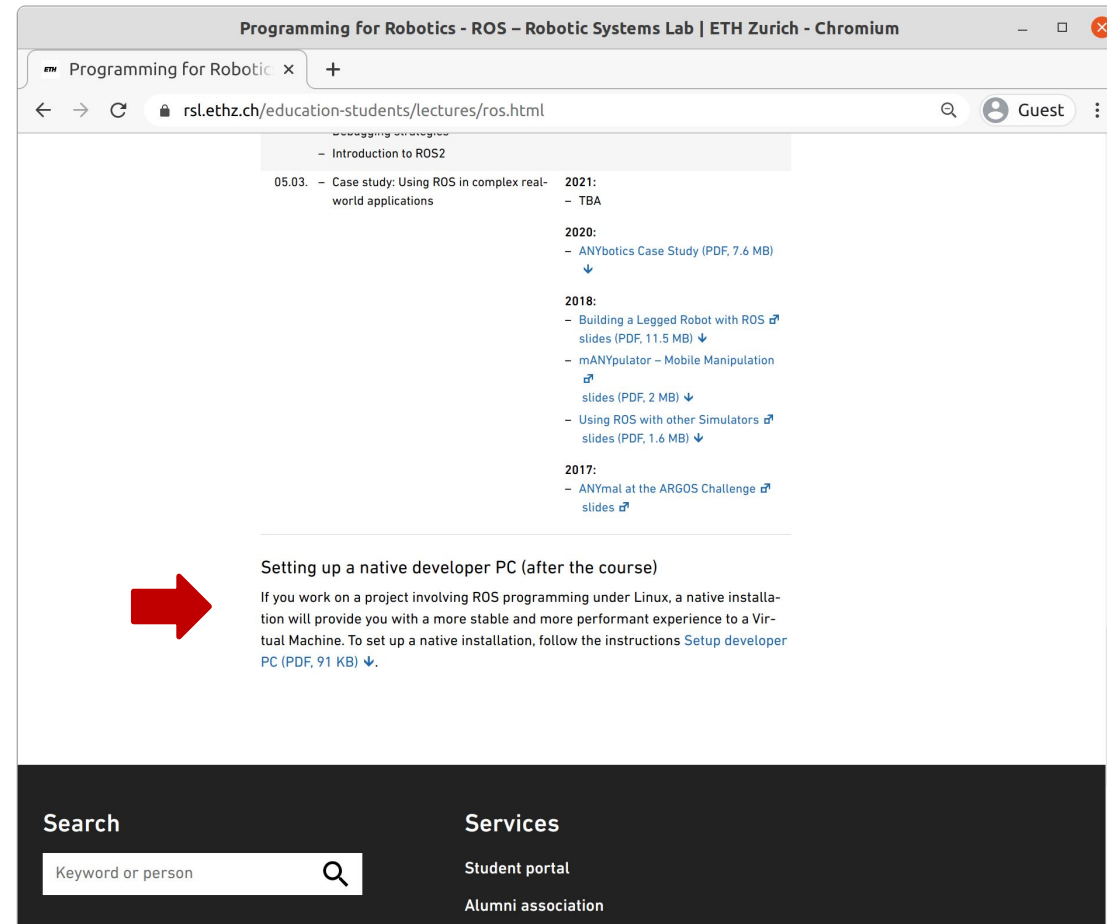
[wiki.ros.org/UnitTesting](http://wiki.ros.org/UnitTesting)

[wiki.ros.org/gtest](http://wiki.ros.org/gtest)

[wiki.ros.org/rostopic](http://wiki.ros.org/rostopic)

[wiki.ros.org/roslaunch/Tutorials/Roslaunch%20Nodes%20in%20Valgrind%20or%20GDB](http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20Nodes%20in%20Valgrind%20or%20GDB)

# Setting Up up a Developer's PC



The screenshot shows a web browser window titled "Programming for Robotics - ROS – Robotic Systems Lab | ETH Zurich - Chromium". The address bar shows the URL "rs.ethz.ch/education-students/lectures/ros.html". The page content includes a sidebar with a list of topics, including "Introduction to ROS2" and "Case study: Using ROS in complex real-world applications". The main content area lists resources for different years: 2021 (TBA), 2020 (ANYbotics Case Study PDF, 7.6 MB), and 2018 (Building a Legged Robot with ROS slides PDF, 11.5 MB; mANyPulator – Mobile Manipulation slides PDF, 2 MB; Using ROS with other Simulators slides PDF, 1.6 MB). Below this, a section titled "Setting up a native developer PC (after the course)" is highlighted with a large red arrow. The text in this section states: "If you work on a project involving ROS programming under Linux, a native installation will provide you with a more stable and more performant experience to a Virtual Machine. To set up a native installation, follow the instructions [Setup developer PC \(PDF, 91 KB\)](#)." The footer of the page contains a search bar and a services section with links to the "Student portal" and "Alumni association".

## Further References

- **ROS Wiki**
  - <http://wiki.ros.org/>
- **Installation**
  - <http://wiki.ros.org/ROS/Installation>
- **Tutorials**
  - <http://wiki.ros.org/ROS/Tutorials>
- **Available packages**
  - <http://www.ros.org/browse/>
- **ROS Cheat Sheet**
  - <https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/>
  - [https://kapeli.com/cheat\\_sheets/ROS.docset/Contents/Resources/Documents/index](https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index)
- **ROS Best Practices**
  - [https://github.com/leggedrobotics/ros\\_best\\_practices/wiki](https://github.com/leggedrobotics/ros_best_practices/wiki)
- **ROS Package Template**
  - [https://github.com/leggedrobotics/ros\\_best\\_practices/tree/master/ros\\_package\\_template](https://github.com/leggedrobotics/ros_best_practices/tree/master/ros_package_template)

# Contact Information

## **ETH Zurich**

Robotic Systems Lab  
Prof. Dr. Marco Hutter  
LEE H 303  
Leonhardstrasse 21  
8092 Zurich  
Switzerland

[rsl.ethz.ch](http://rsl.ethz.ch)

## **Lecturers**

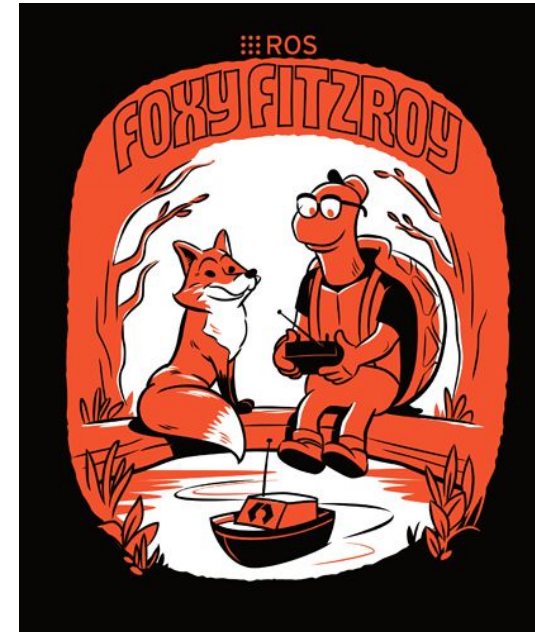
Tom Lankhorst (tom.lankhorst@mavt.ethz.ch)  
Edo Jelavic (edo.jelavic@mavt.ethz.ch)

Course website:

[rsl.ethz.ch/education-students/lectures/ros.html](http://rsl.ethz.ch/education-students/lectures/ros.html)

# ROS 2

- We will give a brief intro and point you to some resources
- There is still a lot of ROS 1 code out there
- ROS 2 intro
  - Philosophy
  - Differences
  - Commands like tools
  - Launch files
  - ROS bridge



# Why ROS 2?

- ROS 1 was not designed with all of today's use cases in mind
- Designed mainly for research
- Initially only designed for PR2
- Never really designed for Real-time

**More info**

[https://design.ros2.org/articles/why\\_ros2.html](https://design.ros2.org/articles/why_ros2.html)

## Why ROS 2? - new use cases

- Teams of robots
- Not suited for bare-metal types of micro controlled
- Non real time communication
- Lossy networks
- Uses in research & industry (e.g. certification)
- API redesign

**More info**

[https://design.ros2.org/articles/why\\_ros2.html](https://design.ros2.org/articles/why_ros2.html)



## Difference between ROS 1 and ROS 2

- Support for other OS's
- at least C++11 and Python 3 (C++03 before Noetic and Python 2 in ROS 1)
- Using off the shelf middleware (as opposed to custom)
- Tighter Python integration (e.g. launch files are in Python)
- Real time nodes (with some assumptions though)
- etc...
- Technical changes under the hood, fewer conceptual changes

**More info**

<https://design.ros2.org/articles/changes.html>

# ROS 2 concepts

- Graph Concepts (similar to ROS 1)
  - nodes, messages, topics ...
- Nodes (similar to ROS 1)
- ROS Client library (RCL)
  - rclcpp - C++
  - rclpy - Python
- Discovery
  - similar to ROS 1
  - However nodes establish contacts only if they have compatible Quality of service settings

**More info**

<https://design.ros2.org/articles/changes.html>

# Nodes

- Run a node with

```
ros2 run <package_name> <executable_name>
```

- List all nodes with

```
ros2 node list
```

- Retrieve information about a node with

```
ros2 node info <node_name>
```

**More info**

<https://index.ros.org/doc/ros2/Tutorials/Understanding-ROS2-Nodes/>

# Where is the rosmaster?

- ROS1 - master-follower architecture
- ROS2 - replaced by Data Distribution Service (DDS)
- DDS is a distributed service that does the discovery, marshalling and transport in the background

**More info**

<https://www.theconstructsim.com/ros2-in-5-mins-003-where-is-roscore-in-ros2/>

[https://design.ros2.org/articles/ros\\_on\\_dds.html](https://design.ros2.org/articles/ros_on_dds.html)

# ROS 2 messages

- The messages stay same as in ROS 1
- `.msg` and `.srv` files are converted into `.idl` files and interfaces are created (same as ROS 1)
- You have to create them using `CMakeLists.txt`

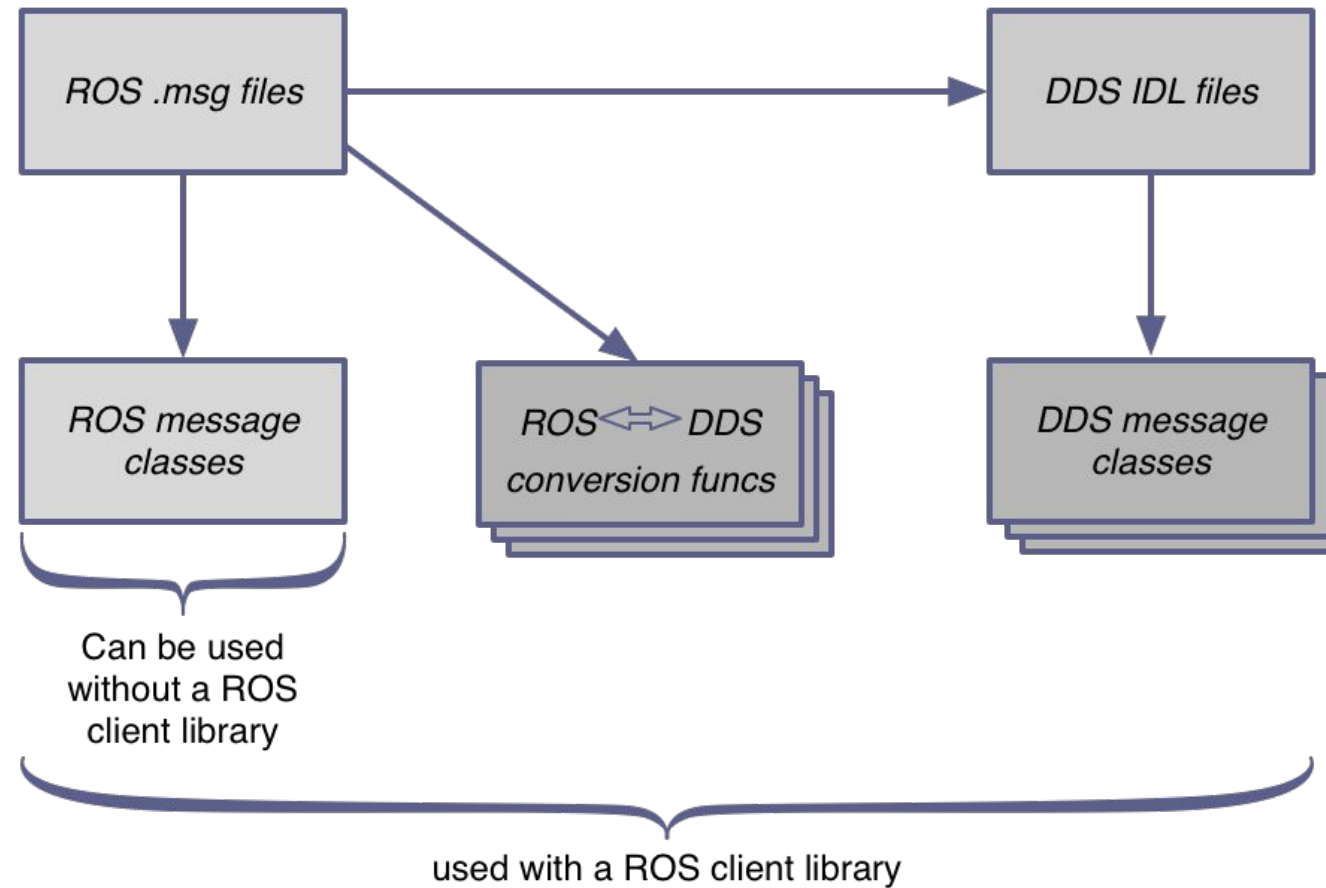
```
find_package(rosidl_default_generators
REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Num.msg"
  "srv/AddThreeInts.srv"
)
```

**More info**

<https://index.ros.org/doc/ros2/Tutorials/Custom-ROS2-Interfaces/>

# ROS 2 messages



# ROS 2 Workspace Environment

- Defines context for the current workspace, same as ROS 1
- Default workspace loaded with

```
> source /opt/ros/foxy/setup.bash
```

- You can overlay your workspaces, same as in ROS 1
- You can overlay multiple workspaces and more than one ROS 2 distribution installed

Check your ROS2 configuration:

```
printenv | grep ROS
```

**More info**

<https://index.ros.org/doc/ros2/Tutorials/Configuring-ROS2-Environment/>

# Building packages in ROS 2

- `catkin build` used in ROS1 is replaced by the colcon build tool
- Clone a package:
  - `git clone https://github.com/ros/ros_tutorials.git -b foxy-devel`
- Build from the top level folder (where src is) with:
  - `colcon build`
- You still need to source your workspace
- `source install/setup.bash` -> source workspace and underlay
- `source install/local_setup.bash` -> sources only the workspace

**More info**

<https://index.ros.org/doc/ros2/Tutorials/Colcon-Tutorial/>



# Creating packages in ROS 2

- You can start by creating a package
  - `ros2 pkg create --build-type ament_cmake <package_name>`
- In addition the ros2 can also create a node for you
  - `ros2 pkg create --build-type ament_cmake --node-name <node_name> <package_name>`
- You can also list all the dependencies
  - `ros2 pkg create <pkg-name> --dependencies [deps]`

**More info**

<https://index.ros.org/doc/ros2/Tutorials/Colcon-Tutorial/>

# Launching multiple nodes in ROS 2

- No more xml launch files
- Launch files are written in Python
- You can launch a file with:
  - `ros2 launch <pkg_name> <launch_file_name>`
- If you are writing a C++ package, you need to ensure that launch files are copied over
- ```
# Install launch files.  
install(DIRECTORY  
  launch  
  DESTINATION share/${PROJECT_NAME}/  
)
```

**More info**

<https://index.ros.org/doc/ros2/Tutorials/Colcon-Tutorial/>

# Can you use both ROS1 and ROS2?

- Yes, there is a bridge that will pass messages from both sides
- You need to (in the following order):
  - source ROS1
  - source ROS2
  - export ROS\_MASTER\_URI
  - run the bridge with: `ros2 run ros1_bridge dynamic_bridge`
- The bridge should be running in the background

**More info**

[https://index.ros.org/p/ros1\\_bridge/github-ros2-ros1\\_bridge/](https://index.ros.org/p/ros1_bridge/github-ros2-ros1_bridge/)

# Summary of ROS2 vs ROS1

| ROS                                                                                                                                 | ROS 2                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| Uses TCPROS (custom version of TCP/IP) communication protocol                                                                       | Uses DDS (Data Distribution System) for communication                                                                   |
| Uses ROS Master for centralized discovery and registration. Complete communication pipeline is prone to failure if the master fails | Uses DDS distributed discovery mechanism. ROS 2 provides a custom API to get all the information about nodes and topics |
| ROS is only functional on Ubuntu OS                                                                                                 | compatible with Ubuntu, Windows 10 and OS X                                                                             |
| Uses C++ 03 and Python2 before Noetic                                                                                               | Uses C++ 11 (potentially upgradeable) and Python3                                                                       |
| ROS only uses CMake build system                                                                                                    | ROS 2 provides options to use other build systems                                                                       |
| Has a combined build for multiple packages invoked using a single CMakeLists.txt                                                    | Supports isolated independent builds for packages to better handle inter-package dependencies                           |
| Data Types in message files <b>do not support default values</b>                                                                    | Data types in message files can now have default values upon initialization                                             |
| roslaunch files are written in XML with limited capabilities                                                                        | roslaunch files are written in Python to support more configurable and conditioned execution                            |
| Cannot support real-time behavior deterministically even with real-time OS                                                          | Supports real-time response with apt RTOS like RTPREEMPT                                                                |

# ROS 2 resources

## **Awesome ROS 2**

<https://design.ros2.org/articles/changes.html>

## **ROSCon Content**

<https://index.ros.org/doc/ros2/ROSCon-Content/#roscon>

## **ROS Index**

<https://index.ros.org/doc/ros2/>

## **ROS2 package template**

[https://github.com/leggedrobotics/ros\\_best\\_practices/tree/foxy](https://github.com/leggedrobotics/ros_best_practices/tree/foxy)

(package template is still work in progress)