ROS 2

ROBOTICS

POLITECNICO
MILANO 1863

WHY A COURSE ON ROS1?

Simpler node structure in ROS1

Core defining elements and paradigms are similar

More stable and less subject to last-minute bugs

Well defined architecture

Still used in many scenarios

Good starting point to then learn ROS2

WHY A COURSE ON ROS1?

Simple Publihsher

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
#include <memory>

class MyPublisher : public rclcpp::Node
{
public:
  MyPublisher() : Node("my_publisher"), count_(0)
  {
    publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10);
    timer_ = this->create_wall_timer(
      std::chrono::seconds(1),
      std::bind(&MyPublisher::publish_message, this));
  }

private:
  void publish_message()
  {
    auto message = std_msgs::msg::String();
    message.data = "Hello, ROS 2! " + std::to_string(count_++);
    RCLCPP_INFO(this->get_logger(), "Publishing: '%s'", message.data.c_str());
    publisher_->publish(message);
  }

  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
  rclcpp::TimerBase::SharedPtr timer_;
  size_t count_;
};

int main(int argc, char **argv)
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<MyPublisher>();
  rclcpp::spin(node);
  rclcpp::shutdown();
  return 0;
}
```

WHY DO WE NEED ROS 2?

Additional features

(need ad-hoc implementation)

- Real-time
- Safety
- Certification
- Security

ROS 2

=> Compatible with industrial application requirements

WHY DO WE NEED ROS 2?

- Designed for PR2 robot (or similar scenario/setup) (ROS1)

- Now on teams of robot, embedded platform, real time systems, non ideal-network, production environment, NASA robots

- Pub-sub system built from scratch (ROS1)

- New open source libraries for pub-sub (ROS2)

- Parallel development (ROS2 work started in 2014)

WHY DO WE NEED ROS 2?

- Multiple os: Linux, OSX, windows
- C++14/ python3
- DDS based
- time as standard messages
- node/nodelet

WHEN SHOULD WE MOVE TO ROS 2?

ROS 1 distributions: http://wiki.ros.org/Distributions

ROS 2 distributions: https://docs.ros.org/en/rolling/Releases.html

ROS 1 EOL: 2025

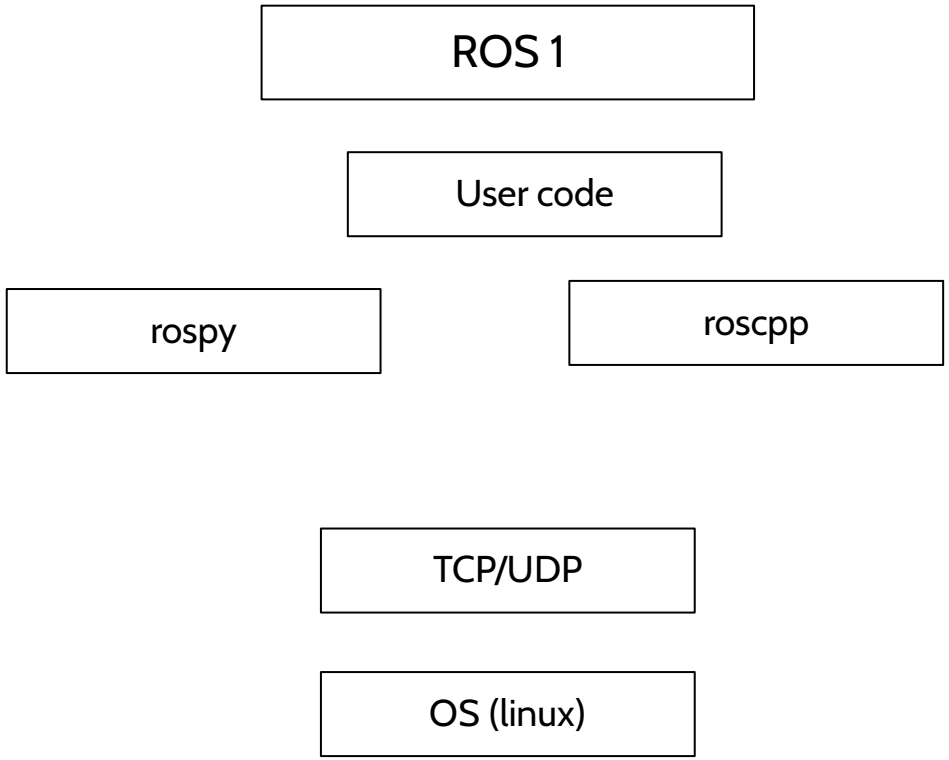
ROS Noetic: first with python 3 (transition to ROS 2)

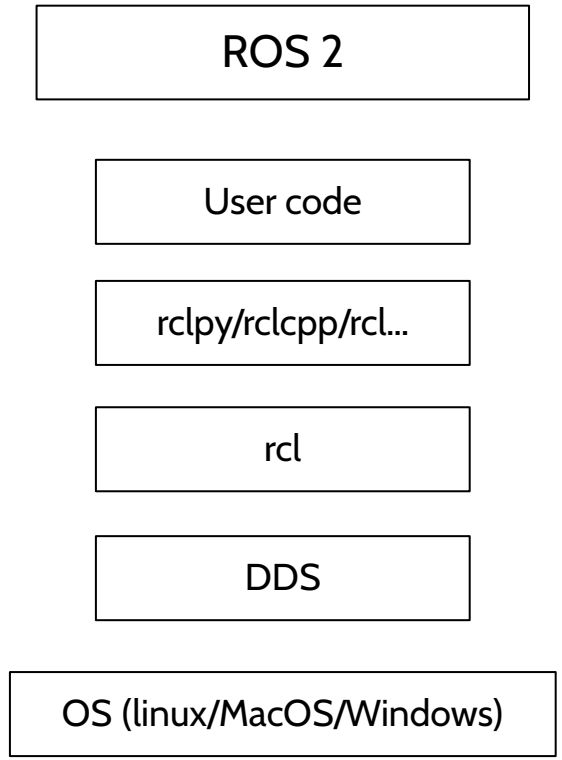
WHEN SHOULD WE MOVE TO ROS 2?

ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			June 27, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010			--

Distro	Release date	Logo	EOL date
Iron Irwini	May 23rd, 2023		November 2024
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		December 9th, 2022
Foxy Fitzroy	June 5th, 2020		June 20th, 2023
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019
Bouncy Bolson	July 2nd, 2018		July 2019
Ardent Apalone	December 8th, 2017		December 2018
beta3	September 13th, 2017		December 2017
beta2	July 5th, 2017		September 2017
beta1	December 19th, 2016		Jul 2017
alpha1 - alpha8	August 31th, 2015		December 2016

WHAT CHANGES? (api)





WHAT CHANGES? (code writing)

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"

#include <sstream>


int main(int argc, char **argv)
{

  ros::init(argc, argv, "talker");
  ros::NodeHandle n;
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  ros::Rate loop_rate(10);

  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }


  return 0;
}
```

```cpp
#include <chrono>
#include <functional>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using namespace std::chrono_literals;
class MinimalPublisher : public rclcpp::Node
{
  public:
    MinimalPublisher()
    : Node("minimal_publisher"), count_(0)
    {
      publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10);
      timer_ = this->create_wall_timer(
      500ms, std::bind(&MinimalPublisher::timer_callback, this));
    }

  private:
    void timer_callback()
    {
      auto message = std_msgs::msg::String();
      message.data = "Hello, world! " + std::to_string(count_++);
      RCLCPP_INFO(this->get_logger(), "Publishing: '%s'", message.data.c_str());
      publisher_->publish(message);
    }
    rclcpp::TimerBase::SharedPtr timer_;
    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
    size_t count_;
};

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<MinimalPublisher>());
  rclcpp::shutdown();
  return 0;
}
```

WHAT CHANGES? (Components)

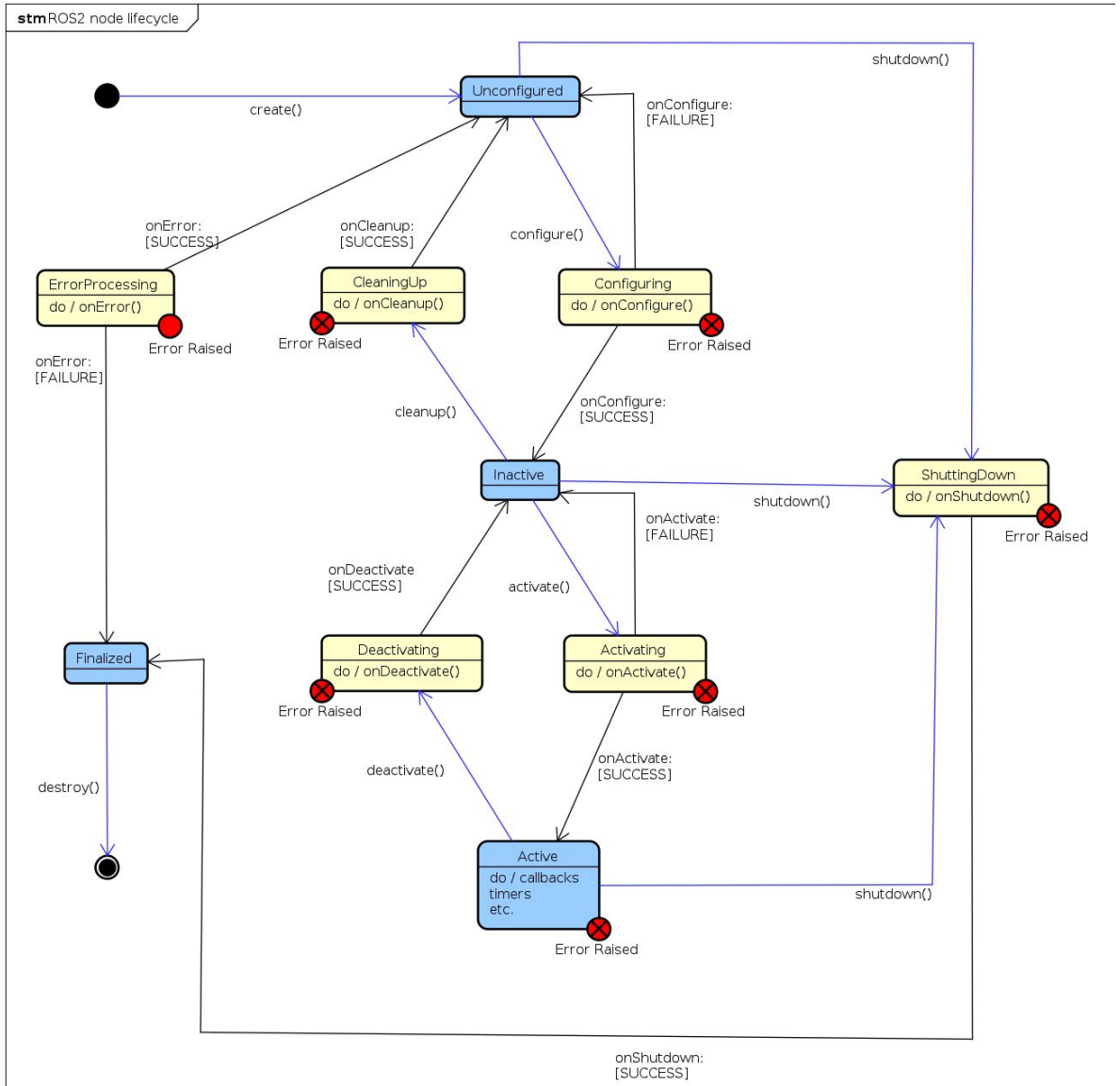
ROS 1

- 1 executable -> 1 node
- Nodelet introduced for single-process, multi-node communication (intraprocess communication)
- Nodelet used only in specific scenario (images)
- Communication via topic

ROS 2

- No nodelets
- Components
- Multiple node in the same executable
- Shared libraries

WHAT CHANGES? (Life cycle)



WHAT CHANGES? (Launchfiles)

ROS 1

- xml
- python api (unknown, not used)
- not sequential

ROS 2

- Python script
- Not particularly different from xml (sequence of actions)
- sequence of node is guaranteed

WHAT CHANGES? (Launchfiles)

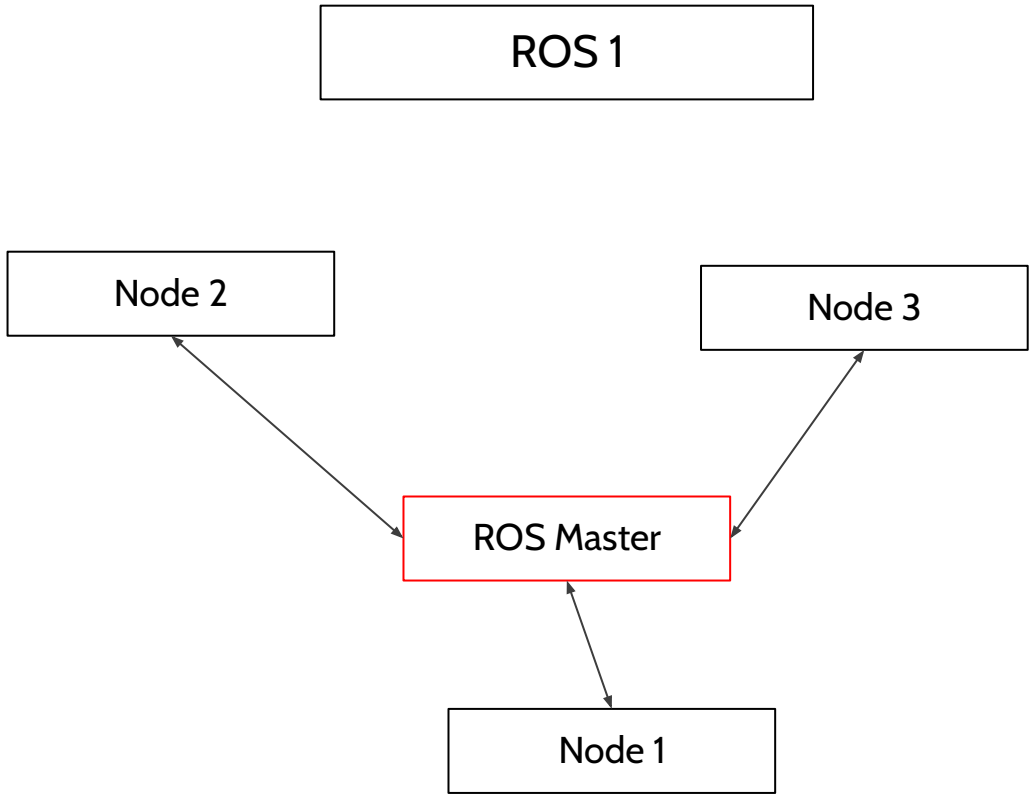
ROS 1

```
<launch>
  <arg name="use_tls" default="false" />
  <node name="mqtt_bridge" pkg="mqtt_bridge"
type="mqtt_bridge_node.py" output="screen">
    <rosparam command="delete" param="" />
    <rosparam command="load" file="$(find
mqtt_bridge)/config/tl_mqtt.yaml" />
    <rosparam if="$(arg use_tls)" command="load"
ns="mqtt" file="$(find mqtt_bridge)/config/
tls_params.yaml" />
  </node>
</launch>
```

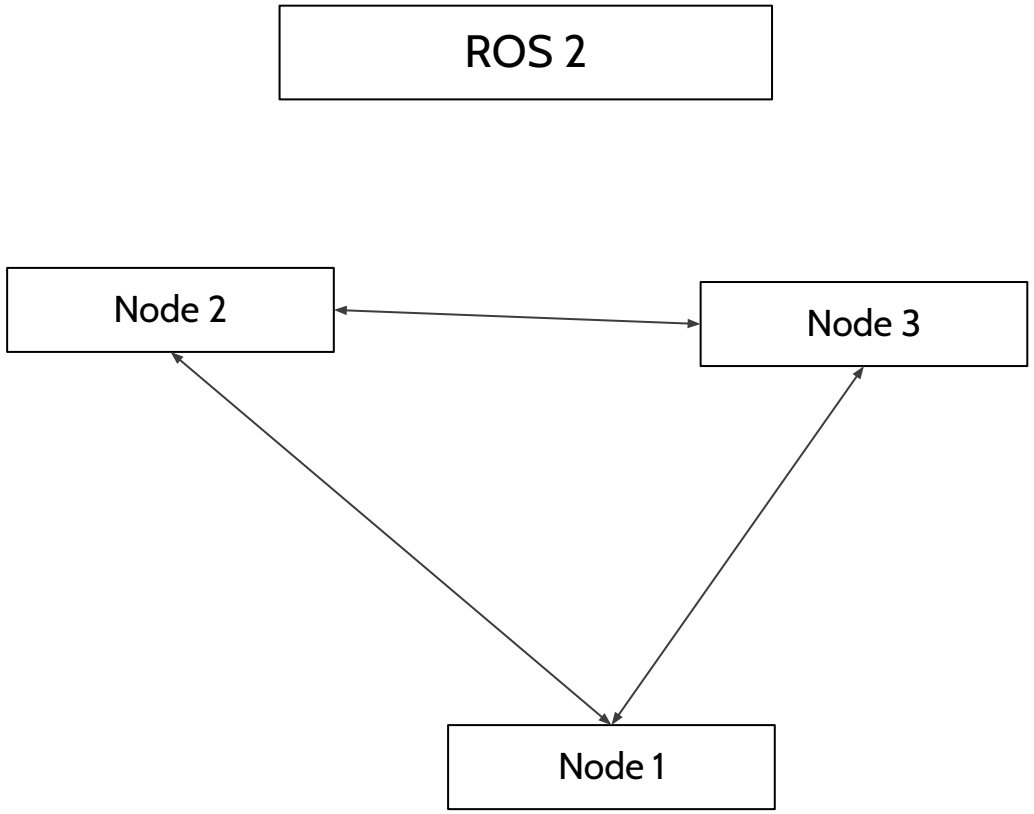
ROS 2

```
from launch import LaunchDescription
from launch_ros.actions import Node
def generate_launch_description():
    ld = LaunchDescription()
    talker_node = Node(
        package="demo_nodes_cpp",
        executable="talker",
    )
    listener_node = Node(
        package="demo_nodes_py",
        executable="listener"
    )
    ld.add_action(talker_node)
    ld.add_action(listener_node)
    return ld
```

WHAT CHANGES? (rosmaster)



export ROS_MASTER_URI



ROS_DOMAIN_ID

WHAT CHANGES? (parameters)

ROS 1	ROS 2
- roscore	- No roscore
- Parameter server	- No parameter server
- Parameters common between nodes	- parameter specific of a node
	- each node has its own parameter server
	- When the node stops the parameters are cancelled

WHAT CHANGES? (services)

ROS 1

- synchronous
- blocking
- this is why we had actionlib

ROS 2

- asynchronous
- similar to actionlib (with less options)
- actionlib still exists for more complex tasks
- can be set to be synchronous/blocking

WHAT CHANGES? (msg,srv,action definition)

ROS 1

- .msg file, .srv file, etc.
- no distinction between msg, srv, action
- for package my_package:
  - my_package/custom_message
  - my_package/custom_srv

ROS 2

- .msg file, .srv file, etc.
- different namespace for msg, srv, action
- for package my_package:
  - my_package/msg/custom_message
  - my_package/srv/custom_srv

WHAT CHANGES? (qos)

ROS 1

- queue size
- c++ allowed to switch between tcp and udp

ROS 2

- complete quality of service system
- yaml file to define qos profiles
- assign to specific publisher/subscriber a specific qos profile
-
- check compatibility of profiles between publisher and subscriber

WHAT CHANGES? (qos)

- History: Keep last/Keep all

- Depth: queue size
- Reliability: Best effort/reliable
- Durability: transient local/volatile
- Deadline: duration
- lifespan: duration
- Liveliness: automatic/manual
- Lease duration: duration

Publisher	Subscription	Compatible
Best effort	Best effort	Yes
Best effort	Reliable	No
Reliable	Best effort	Yes
Reliable	Reliable	Yes

Publisher	Subscription	Compatible
Volatile	Volatile	Yes
Volatile	Transient local	No
Transient local	Volatile	Yes
Transient local	Transient local	Yes

WHAT CHANGES? (command line)

ROS 1	ROS 2
- catkin_make	- colcon build
- rostopic list	- ros2 topic list
- rosservice ...	- ros2 service ...
- rosmsg ...	- ros2 msg ...
	- ros2 bag ...
	-
	- ros2 run ...

SOFTWARE QUALITY

- Design article: http://design.ros2.org/
- ROS Enhancement Proposal (REP): https://ros.org/reps/rep-0000.html
- Testing
- Quality declaration

WHAT CHANGES? (new/interesting things/missing)

New:

- topic statistics
- python packages
- security
- bag recording from inside a node
- tf2 did not change
- ros2doctor

Still missing:

- Better bag
- Service/action recording
- Automatic qos
- Subscriber notification
- Documentation
- Launch file improvement