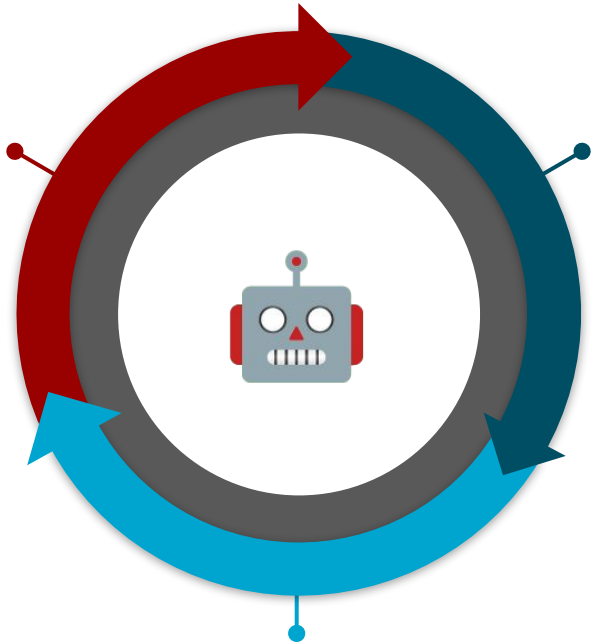
SAPIENZA
UNIVERSITÀ DI ROMA

ROS

Robot Programming
Introducing ROS

Emanuele Giacomini

Robotic Paradigm



Monolithic Approach

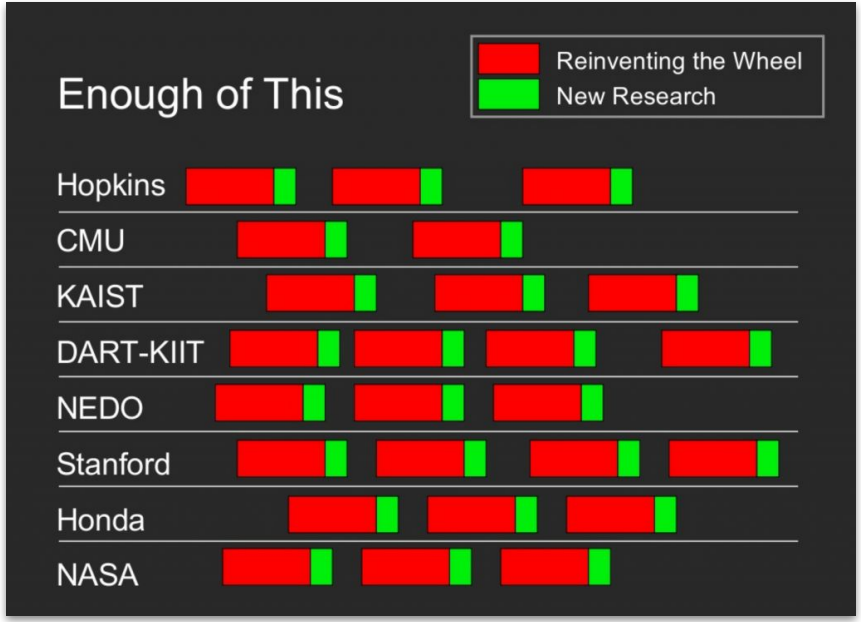
- Robots are **complex**
- Single crash can compromise the entire system

Ideal Approach

Divide components in processes

- **Task decomposition**
  - Process crash → restart
- **Message passing** through IPC
- **Modularity**
  - Change functionality by replacing the process

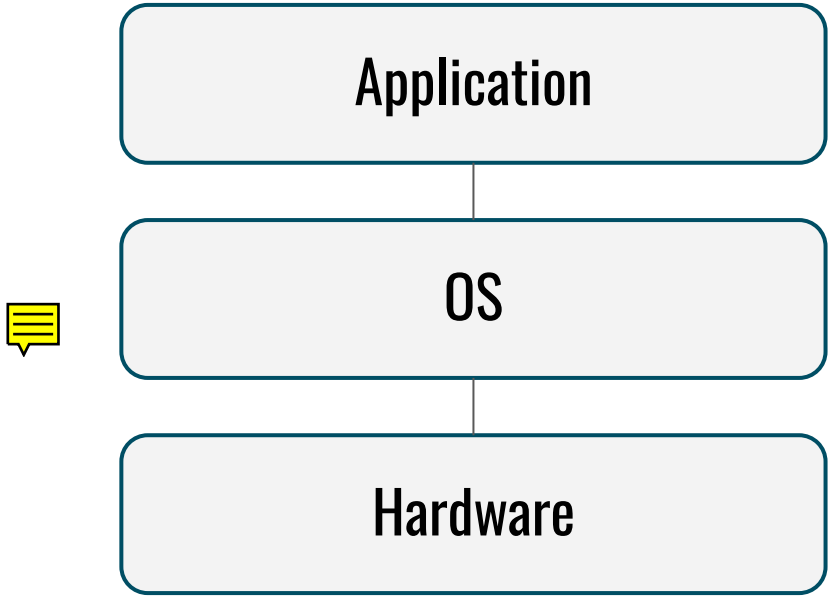
Research issues



Most of the time spent in robotics was reinventing the wheel
(slide from Eric and Keenan pitch deck)

Simplistic Robot Application

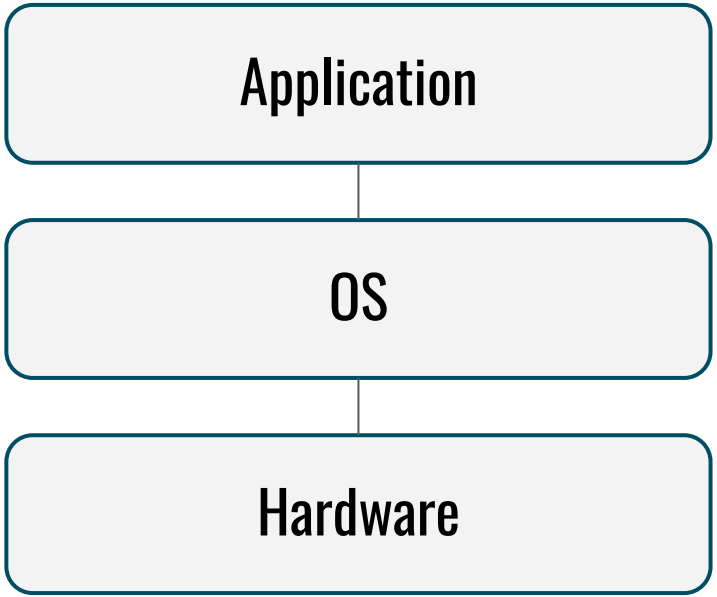
Why is this a bad idea ?



Simplistic Robot Application

Why is this a bad idea ? 🙈

- OS Dependency
- Low-level communication
- Heterogeneity of sensors



Robot Middleware

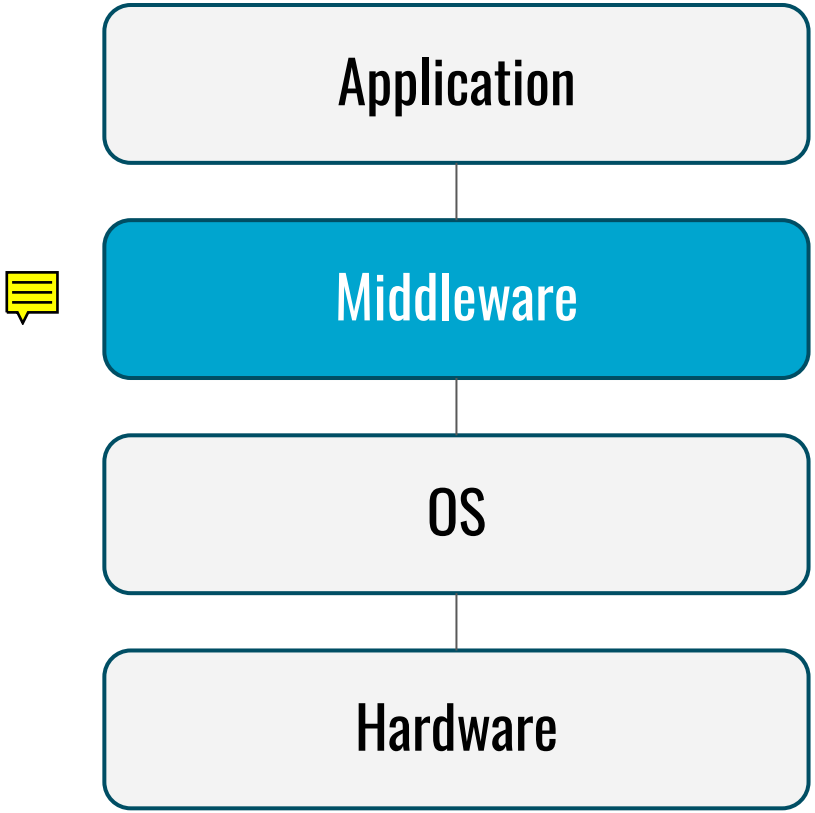
"Middleware is software glue. This doesn't tell us much, but several definitions you're likely to see on the web will describe it as such."

www.middleware.org/whatis.html

Robot Middleware

Why is this a good idea ?

- OS Independent
- Focus on application
- Allow reusability



ROS: Robot Operating System

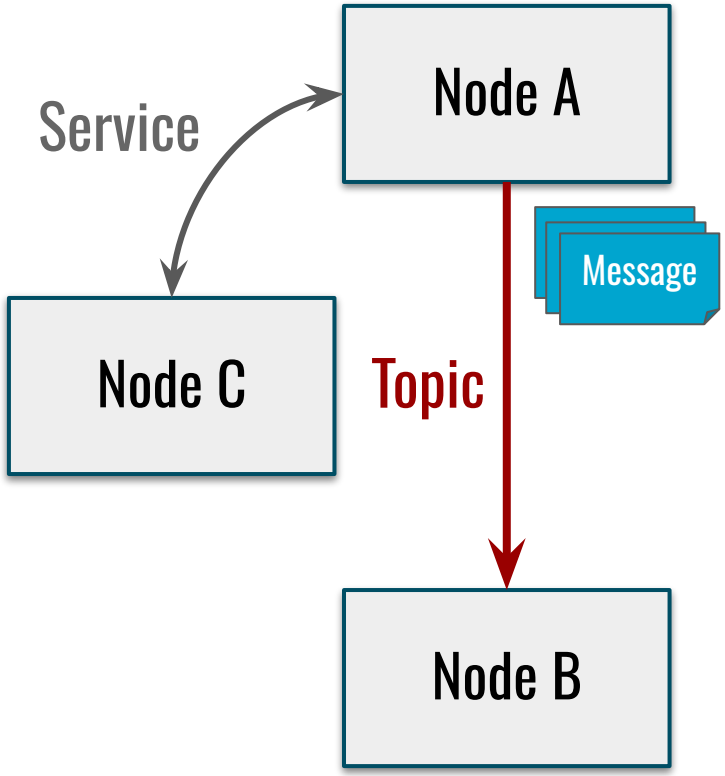
- Message passing
- Process control
- Build system

Why ROS

- Code Reuse
- Modular design
- Language Independent

ROS: Concepts

- **Node**
  - Process, solves problems
- **Message**
  - Information to share
- **Topic**
  - Transport messages to nodes
- **Service**
  - Reply-Response “messages”

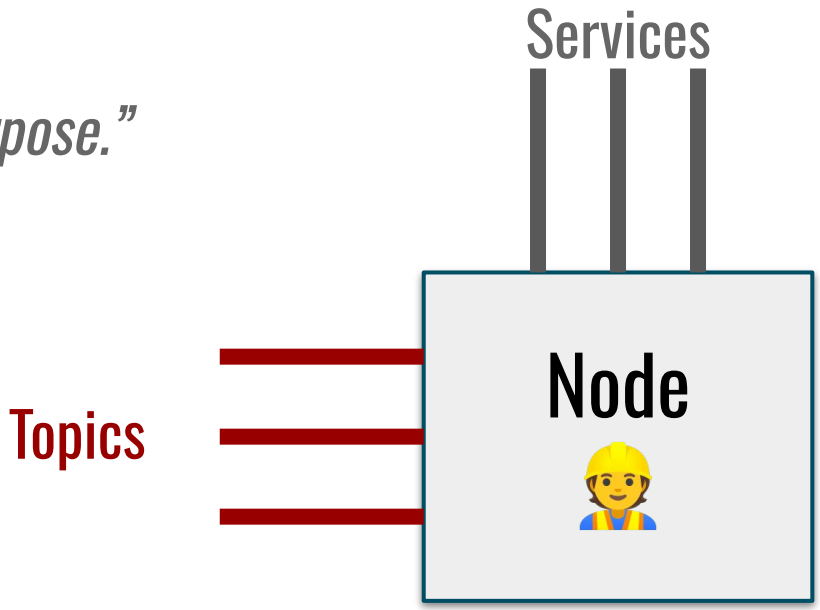


Nodes

"Responsible for a single, modular purpose."

Running instance of a ROS program

- **Topics**
- **Services**
- Other
  - Actions
  - Parameters



Messages

Structured data structure for IPC

- Typed fields
- Standard types are supported
  - also arrays 😇
- Support nesting

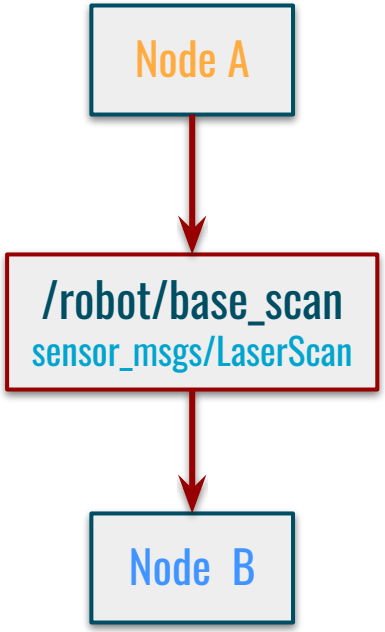
```
std_msgs/msg/Header header
float angle_min
float angle_max
float angle_increment
float time_increment
float scan_time
float range_min
float range_max
float[] ranges
float[] intensities
```

sensor_msgs/msg/LaserScan

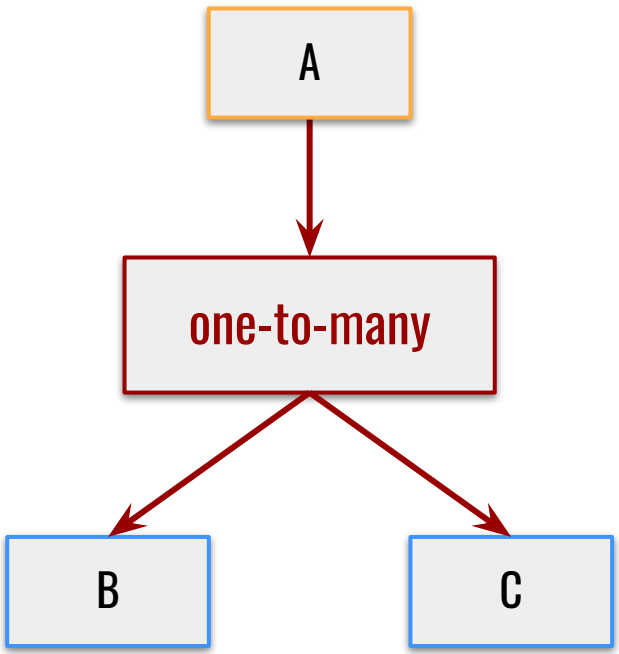
Topics

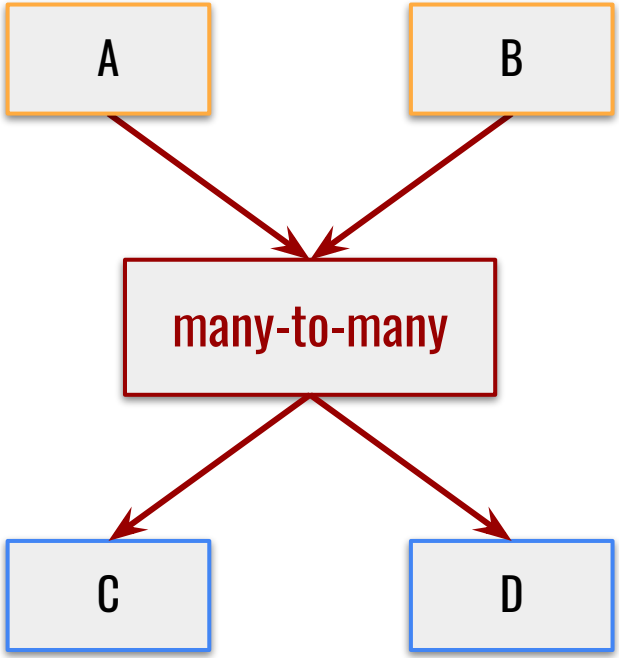
Bus for nodes to exchange messages

- Identified by name
- Transport a single message type
- Publisher/Subscriber mechanism



Topics configuration



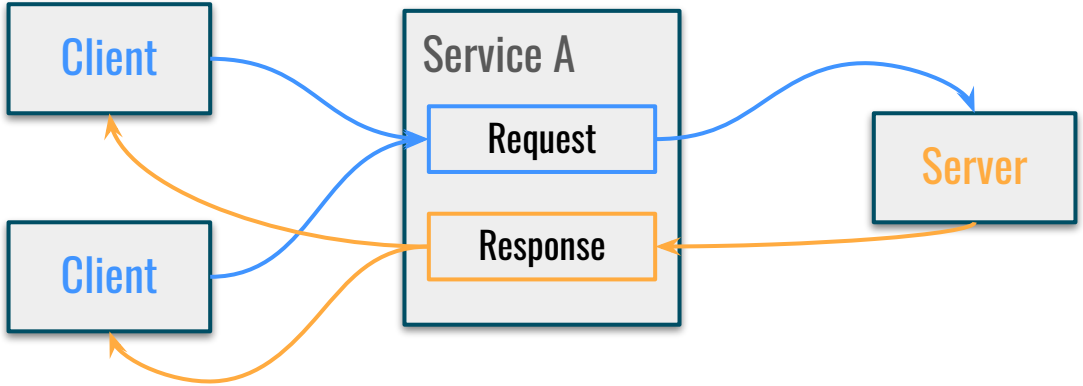


Services

- **Call-and-response model**
- Server/Client mechanism
  - One server per service

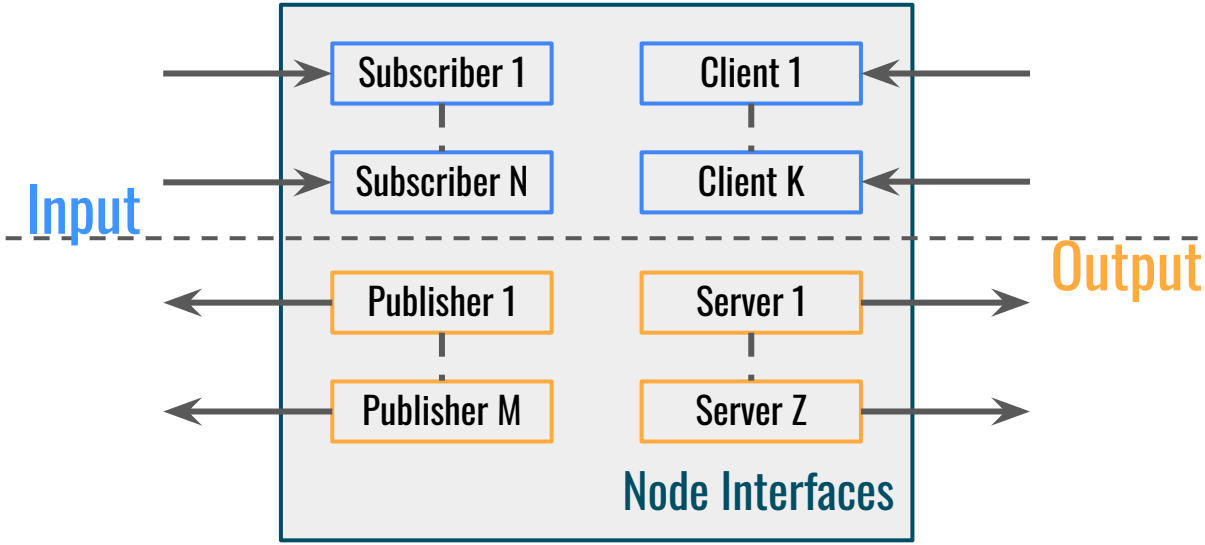
```
int64 a
int64 b
- - -
int64 sum
```

Sum.srv



Wrapping up

A node can have arbitrary many roles



Break Time

Build system 🙉

How to develop in ROS 2 ?

- Functionalities are grouped in **Packages** 💬
- Packages are grouped in a **Workspace**

```
example_package
├── CMakeLists.txt
├── include
│   └── example_package
│       └── foo.h
├── package.xml
└── src
    └── foo.cpp
```

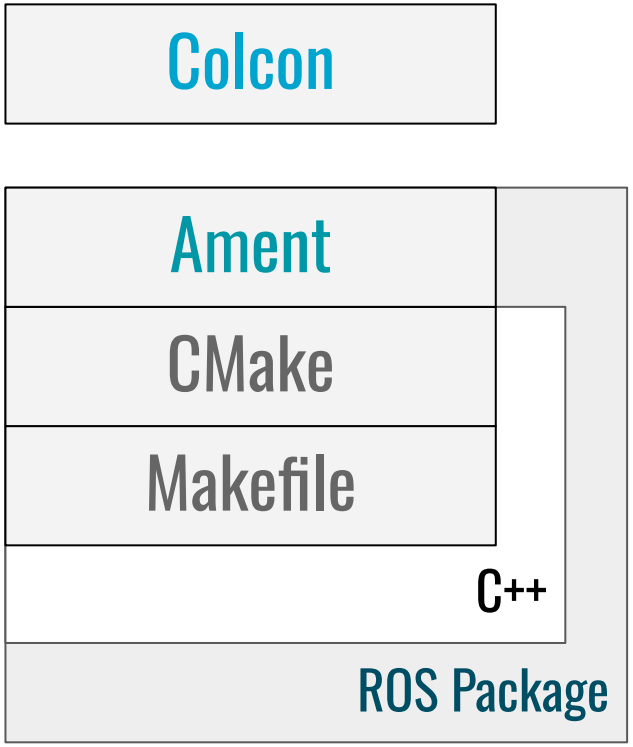
💬

```
ros2_workspace
├── build
├── install
├── log
└── src
    ├── package_1
    └── package_2
```

Build system 🙈

Ingredients for building a package:

- Program(s)
  - Python, C++, etc
- CMake 💬
- **Ament**
  - Focus on building a package
- **Colcon**
  - Focus on building a workspace



Ament

Build system for ROS 2 packages

- Based on CMake
  - Macros for ROS dependencies
- Manages ROS dependencies
  - using *package.xml*

Colcon

Workspaces manager

- Compile packages in ***src*** directory
- Handle package dependencies

```
ros2_workspace
├── build
├── install
├── log
└── src
    ├── package_1
    └── package_2
```

C++ Node Anatomy

- Based on **rclcpp** library

```
class MyNode : public rclcpp::Node {
  public:
    MyNode() : Node("my_node") {}
  private:
    rclcpp::Publisher<msg_type>::SharedPtr publisher;
    rclcpp::Subscriber<msg_type>::SharedPtr subscriber;
};
```

C++ Node Anatomy

```
class MyNode : public rclcpp::Node {
  public:
    MyNode() : Node("my_node") {
      publisher = this->create_publisher<msg_type>("topic_1", 10);
      subscriber = this->create_subscription<msg_type>("topic_2", 10,
                    std::bind(&MyNode::topic_callback, this,
                                std::placeholders::_1));

    }
  private:
    void topic_callback(const msg_type::SharedPtr msg) {
      publisher.publish(msg);
    }
    rclcpp::Publisher<msg_type>::SharedPtr publisher;
    rclcpp::Subscriber<msg_type>::SharedPtr subscriber;
};
```

C++ Node Anatomy

```
#include "rclcpp/rclcpp.hpp"
// include msg_type somehow
// define MyNode (Previous slides)
int main(int argc, char** argv) {
  rclcpp::init(argc, argv);
  auto node = std::make_shared<MyNode>();
  rclcpp::spin(node);
  rclcpp::shutdown();
  return 0;
}
```

Full Example

- Single package
- Two nodes
  - Publisher
  - Subscriber

https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html