**ROS** – Robot Operating System.

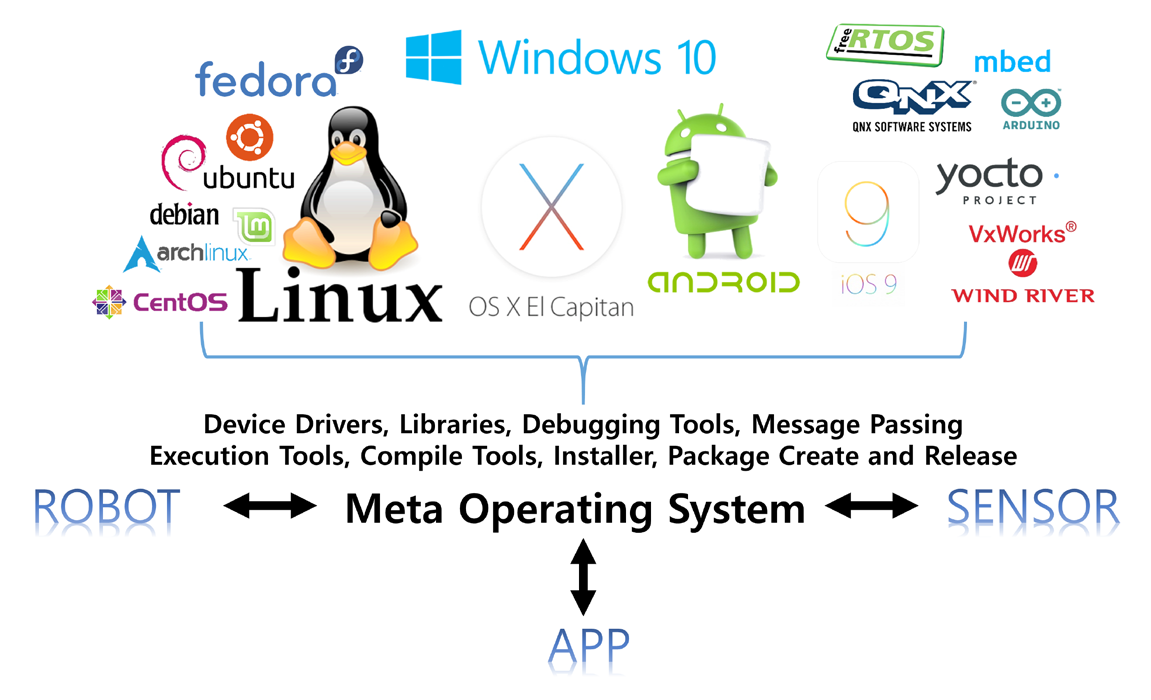
It’s an open-source, meta-operating system for your robot.

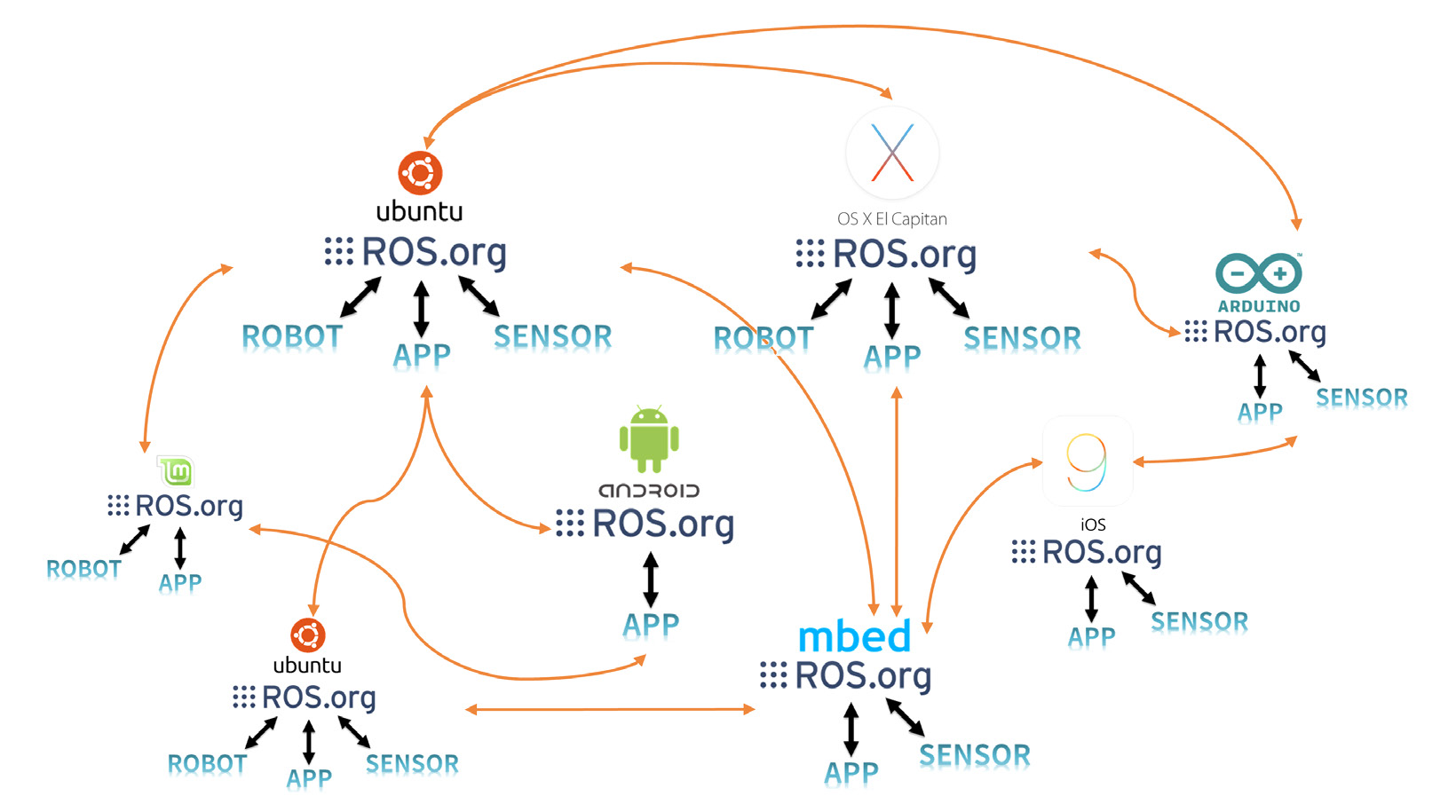
It provides the services you would expect from an operating system, including

**hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.**

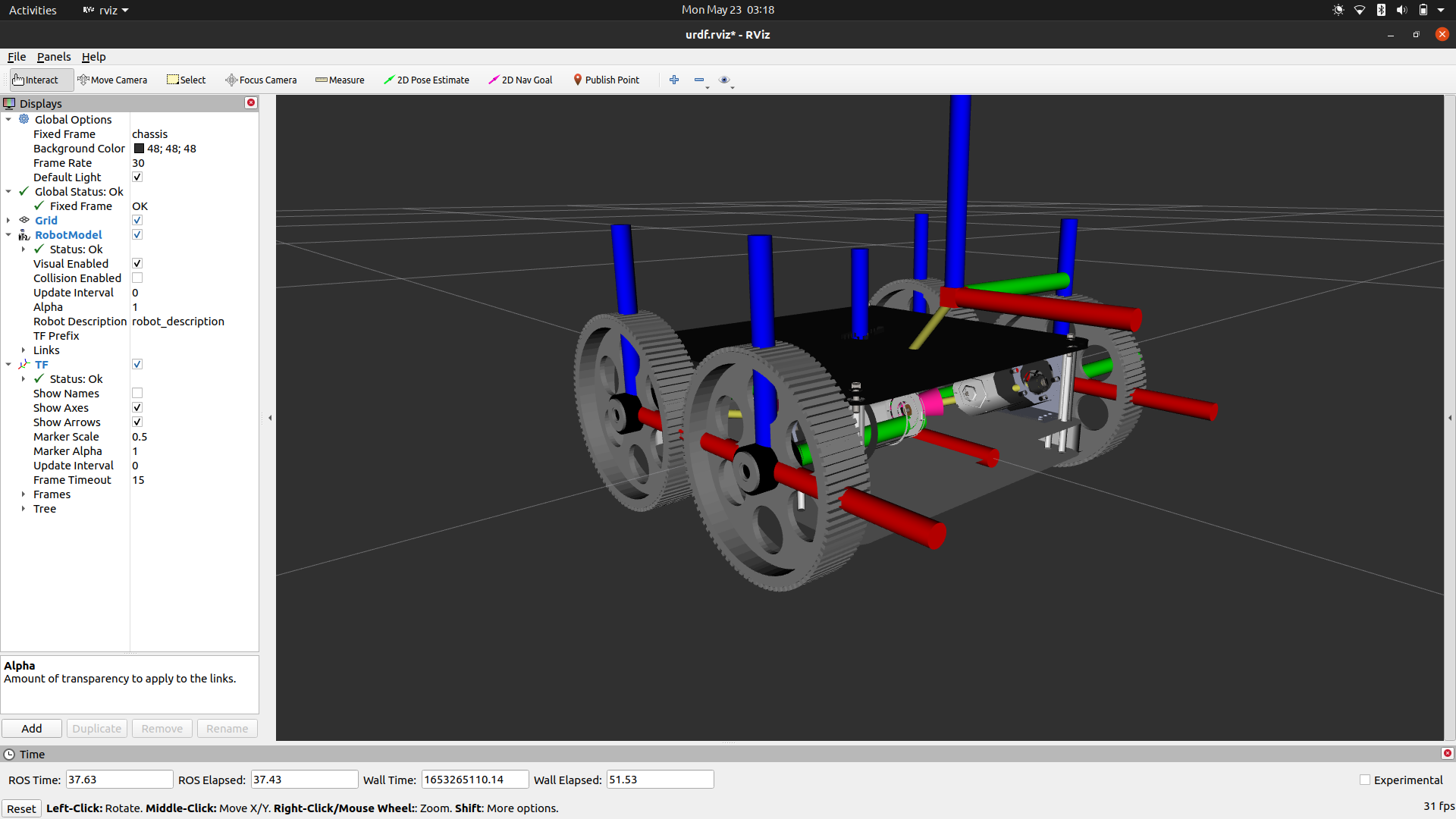
**It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. Such is the embedded systems** [specific purpose computer systems]: **Arduino, NVIDIA’s Jetson Nano, Raspberry Pi, ODROID, Intel NUC**

**It is Cross-platform. -** Offers flexibility and Freedom on how we can develop and implement a Robotic program.

****

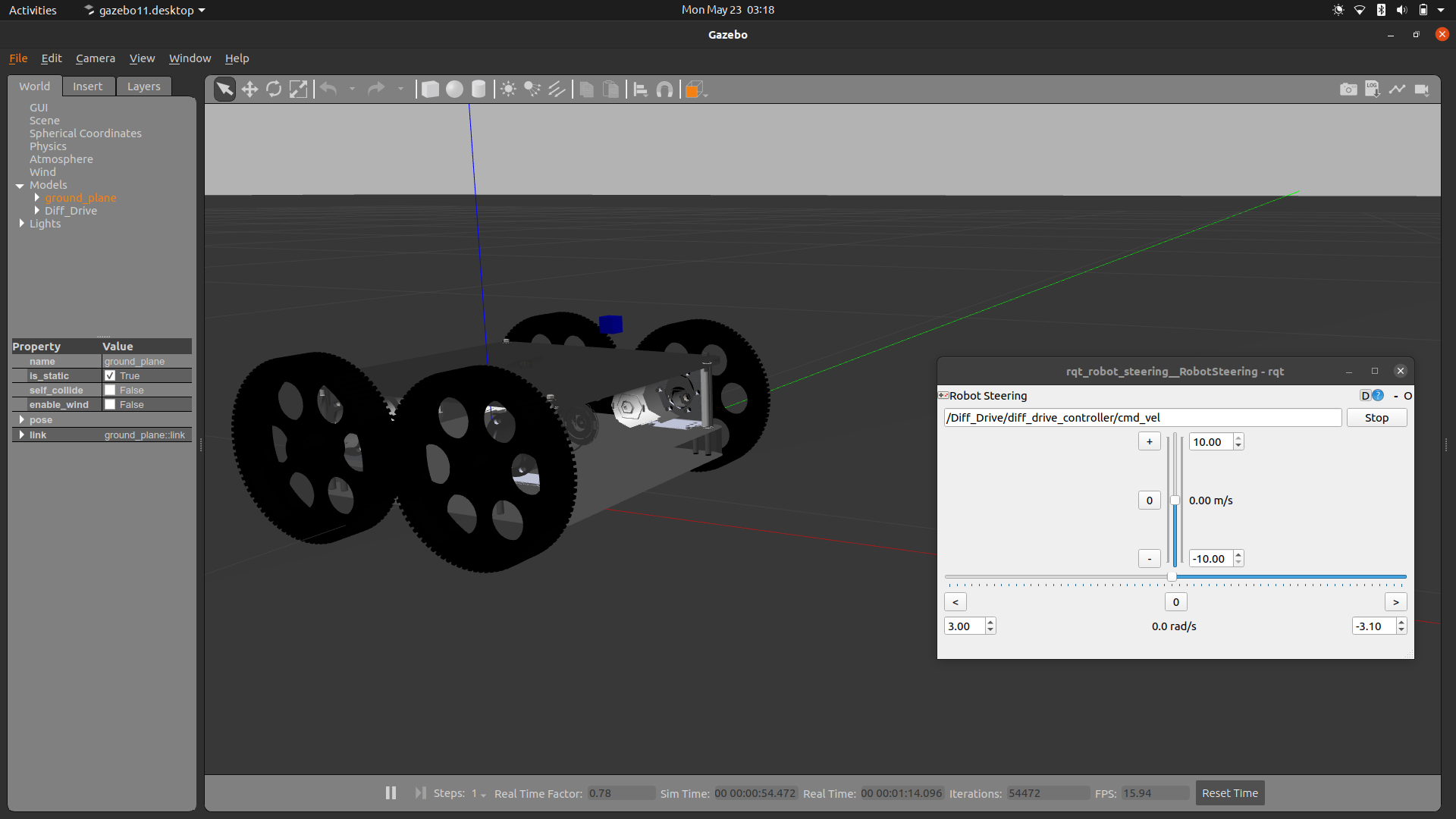
Cross-Platform

Visualization Tool - RVIZ



Emulation Tool – Gazebo (Cross-platform → Not a ROS Package)

Gazebo is an [open-source](https://en.wikipedia.org/wiki/Open-source) 3D [robotics simulator](https://en.wikipedia.org/wiki/Robotics_simulator). It integrated the [ODE](https://en.wikipedia.org/wiki/Open_Dynamics_Engine) physics engine, [OpenGL](https://en.wikipedia.org/wiki/OpenGL) rendering, and support code for sensor simulation and actuator control.



Message Communication within ROS

Terminologies.

Master.

The master acts as a name server for node-to-node connections and message communication.

Node.

A node 4 refers to the smallest unit of processor running in ROS. Think of it as one executable

program

Message

A node sends or receives data between nodes via a message. Messages are variables such as

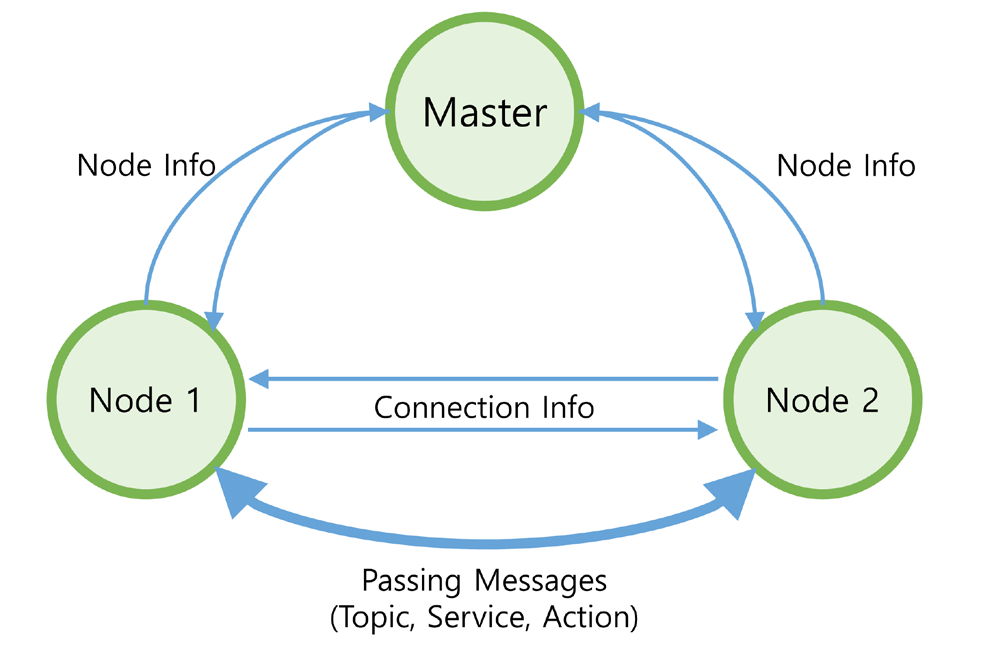
integer, floating point, and Boolean.

Topic.

The topic is literally like a topic in a conversation. The publisher node first registers its topic

with the master and then starts publishing messages on a topic.

Publish.

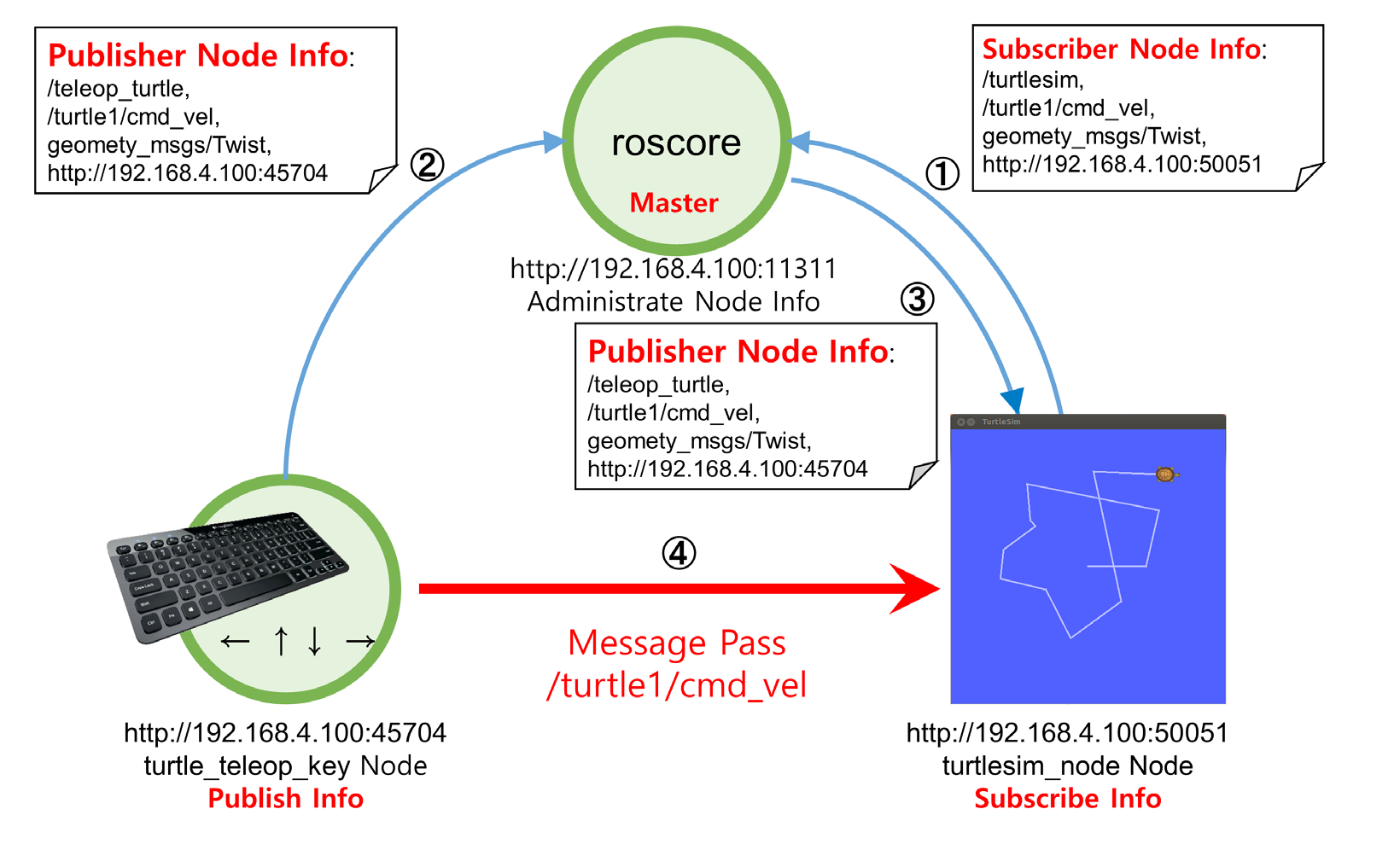
The term ‘publish’ stands for the action of transmitting relative messages corresponding to the

topic.

Subscribe.

The term ‘subscribe’ stands for the action of receiving relative messages corresponding to the

topic.

**CONTROL.**

**1. Motion Control**

**2. Event Control. [Behavior]**

**Control and Programming Methodology.**

The development and implementation of robots is definitely an iterative task, Robotics is hard to learn, and it takes time to develop a good software for a robot. The developer has to spend time working on how to communicate between all the robotic programs, how to implement a 3D simulation of the robot among other numerous functions. However, with the right tools at hand, the process is simplified. Majorly, frees the developer from re-inventing the wheel, therefore allowing him/her address functionalities of the robot that matter most.

The right tool in this context is the **Robotic Operating System**. Commonly referred to as **ROS**, it is an open-source framework that helps researchers and developers build and reuse code between robotics applications. This offers the best platform to prototype a robot while learning and also doing research.

Its a rich platform. This is due to the fact:

* Its open source therefore, it’s development and improvement - though managed centrally - its rich libraries and functionalities are contributed to by a “global community”. The ROS community is a loosely affiliated collection of engineers and hobbyists from around the globe with a shared interest in robotics and open-source software.
* Has numerous libraries, allows code re-usability, has visualization and simulation tools. These offers the perfect combination for development, prototyping and implementation of robotics applications in a cost-effective and efficient manner.

**Software Architecture.**

Software architecture is, simply, **the organization of a system**. This organization includes all components, how they interact with each other, the environment in which they operate, and the principles used to design the software. In many cases, it can also include the evolution of the software into the future.

Therefore, we can look at the ROS as a set of open source algorithms, hardware driver software and tools developed to develop robot control software. Even though it has operating system in its name it is not an operating system. It is

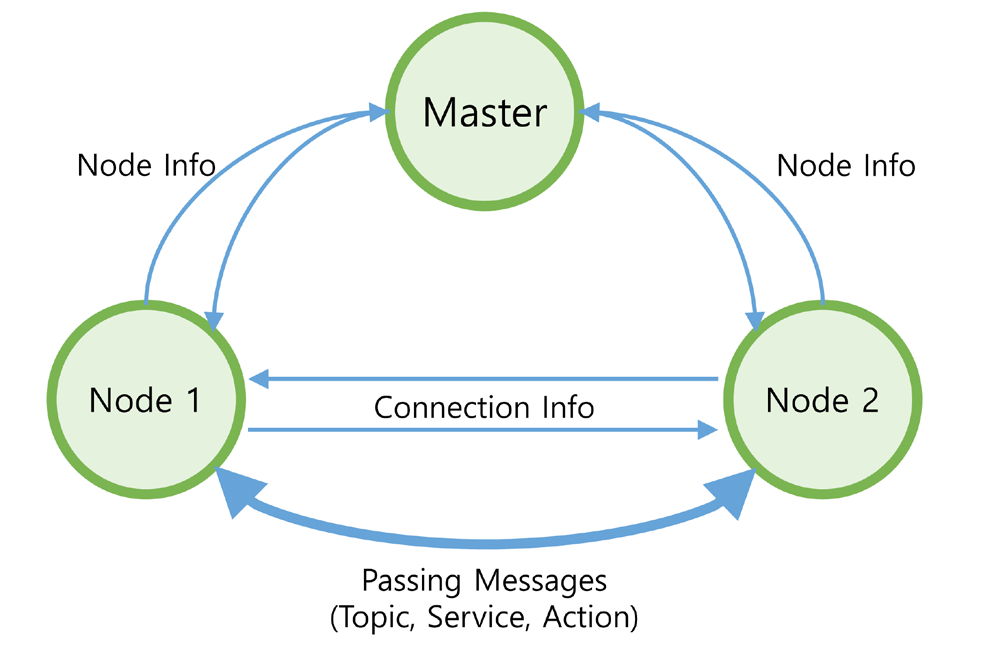
* Communication System (Publish Subscribe and Remote Method Invocation),
* Framework & Tools (Build system & dependency management, Visualization, Record and Replay)
* Ecosystem (Language bindings, Drivers, libraries and simulation (Gazebo)).

ROS is developed on an **Event-Driven Architecture.** An event-driven architecture uses events to trigger and communicate between decoupled services and is common in modern applications built with micro-services. An event is a change in state, or an update. Events can either carry the state or can be identifiers.

Event-driven architectures have three key components: event producers, event routers, and event consumers. A producer publishes an event to the router, which filters and pushes the events to consumers. Producer services and consumer services are decoupled, which allows them to be scaled, updated, and deployed independently.

**Dynamic Processing.**

An efficient control program for a robot is one that offers dynamic and active communication between its components. Components to imply sensory, locomotory and manipulation. The frequency of information, data and instruction transaction must be characterized by null latency. The core technology of ROS lies in its distributed communication mechanism—the ROS Master helps different Nodes to find each other and establish connections. The most basically and frequently used communication mechanisms in ROS are Topic and Service.



**Language Support**

The main languages for writing ROS code are **C++ and Python**, C++ being preferred due to better performance. However, ROS is language-neutral/language independent, and can be programmed in various languages. The ROS specification works at the messaging layer. Peer-to-peer connections are negotiated in XML-RPC – Remote Procedure Call, which exists in a great number of languages. It also offers a standard JavaScript Library.

**Scripting**

→ task execution within a special run-time environment by an interpreter instead of a compiler.

→ special run-time environment: embedded operating system.

An embedded OS is specially designed to perform a specific task other than a typical computer.

→ ROS on eMCOS is the embedded platform supporting ROS.

[eMCOS – its an ultra-scalable real-time OS]

→ data visualization and performance extraction.

→ why script:

dynamic handling of user queries and data generation and distribution.

the ROS is also based on this: distributed processing, multiple package management and

→ **server-side and client-side :**

→ script for the purpose of tele-operation/remote operation. Its advantage is the handling of dynamic user/client queries, data generation and distribution. Its also portable and light in terms of the processing capability and power required.

So, leveraging this advantages to visually demonstrate: tele-operation, obstacle avoidance and lane detection.

→ why obstacle avoidance and lane detection.

→ to graphically represent the physical environment in a simulation environment is a easy. But when it gets to the level of defining the environment in such a way that it becomes what in the robotics world we call “Collision environment” is graphically demanding. Therefore, the resolution to scripting.