1. Introduction

1. What is ROS?

A middleware is a computer software that connects software components or applications (middleware sits "in the middle" between software applications that may be run on different operating systems).

Middleware is useful for:

- **Portability**: they offer a common programming model across languages and/or platforms.
- **Reliability**: they are developed and tested separately from the final application.
- **Managing complexity**: low-level aspects are managed by suitable libraries that abstract specific operating system and hardware aspects.

Robotic systems are usually complex systems built on many different hardware and software components, as sensors and actuators as well as planners and control algorithms.

The use of a **robotic middleware** allows different components to exchange data using a common communication channel and using interfaces that are consistent between different applications.

This eases the **sharing and reuse** of code among different projects (as an example, if you need to switch from a proximity sensor to another, it is possible to write a new component that share the same interface and update it without modifying the rest of the application).

1. Introduction

1. What is ROS?

"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms."

ROS provides hardware abstraction,
device drivers,
libraries,
visualizers,
message-passing,
package management, and more...

ROS is licensed under an open source, BSD license.

Which main issues ROS helps to resolve?

1. Introduction

1. What is ROS?

Distributed computation: Many modern robot systems rely on software that spans many different processes and runs across several different computers.

- *ROS covers the need for communication between multiple processes that may or may not live on the same computer.*
- *ROS's message passing interface is becoming a de facto standard for robot software interoperability*

Software reuse: The rapid progress of robotics research has resulted in a growing collection of good algorithms for common tasks such as navigation, motion planning, mapping, and many others.

- *ROS's standard packages provide stable, debugged implementations of many important robotics algorithms.*

Rapid testing: The testing of real robotic systems applications can be time consuming and error-prone.

- *ROS facilitates the separation of high-level processing and decision making programs from the low-level direct control of the hardware, which can be temporarily replaced by a simulator.*
- *ROS also provides a simple way to record and play back sensor data and other kinds of message.*

The real robot, the simulator, and the bag playback mechanism can all provide identical (or at least very similar) interfaces.

1. Introduction

1. What is ROS?

ROS is not:

- An actual operating system

  Although it offers 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.

- A programming language

  ROS programs are written in C++ and client libraries are also available in Python, Java, Lisp,…

- A programming environment / IDE

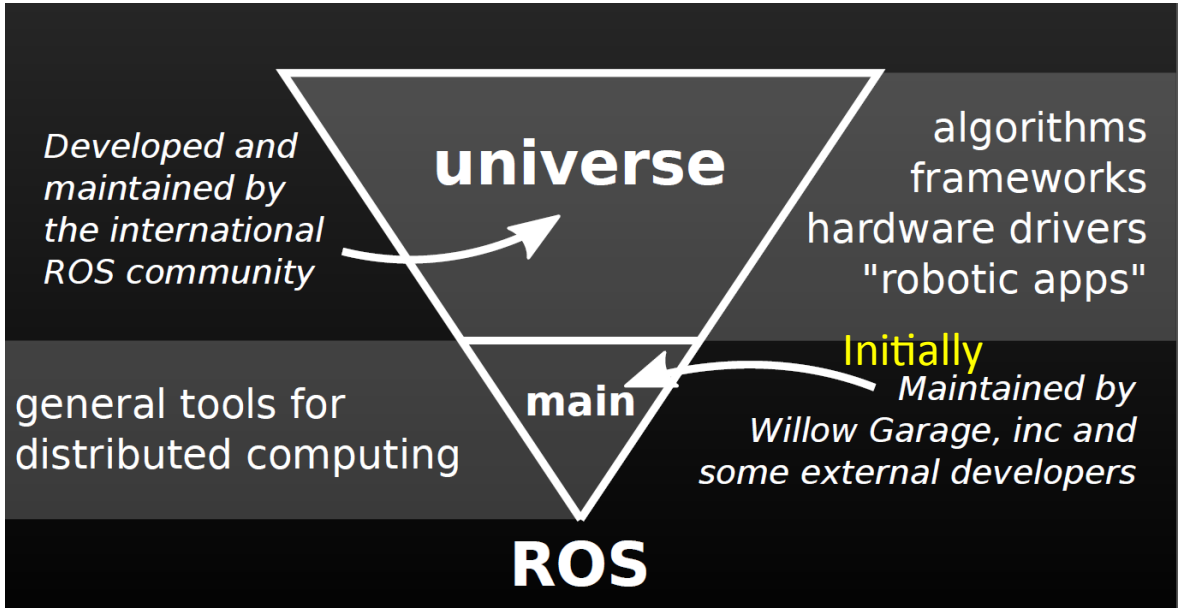
  It is common to simply use a text editor and the command line, without any IDE.

- A hard real-time architecture

  ROS is fast, and is used for online operation in many robots, it is inherently best-effort in many cases, and does not provide guarantees about the timing of operations (should not use ROS for operations that have strict timing requirements).

1. Introduction

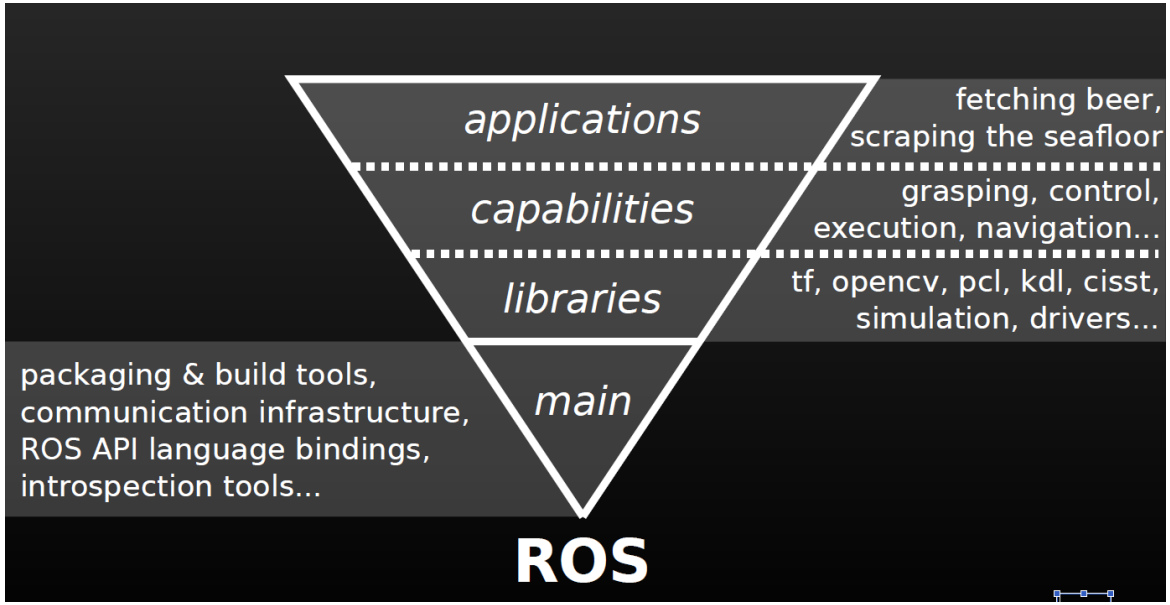
2. Levels of development



Currently maintained by
open robotics
http://www.openrobotics.org

1. Introduction

2. Levels of development



ROS core components:

1. Communication infrastructure
2. Robot-specific features
3. Tools