ROS Quick Overview

by *Jorge Couchet*

ROS for Peer-to-Peer Network

ROS Network

It is also known as the **ROS Computation Graph**

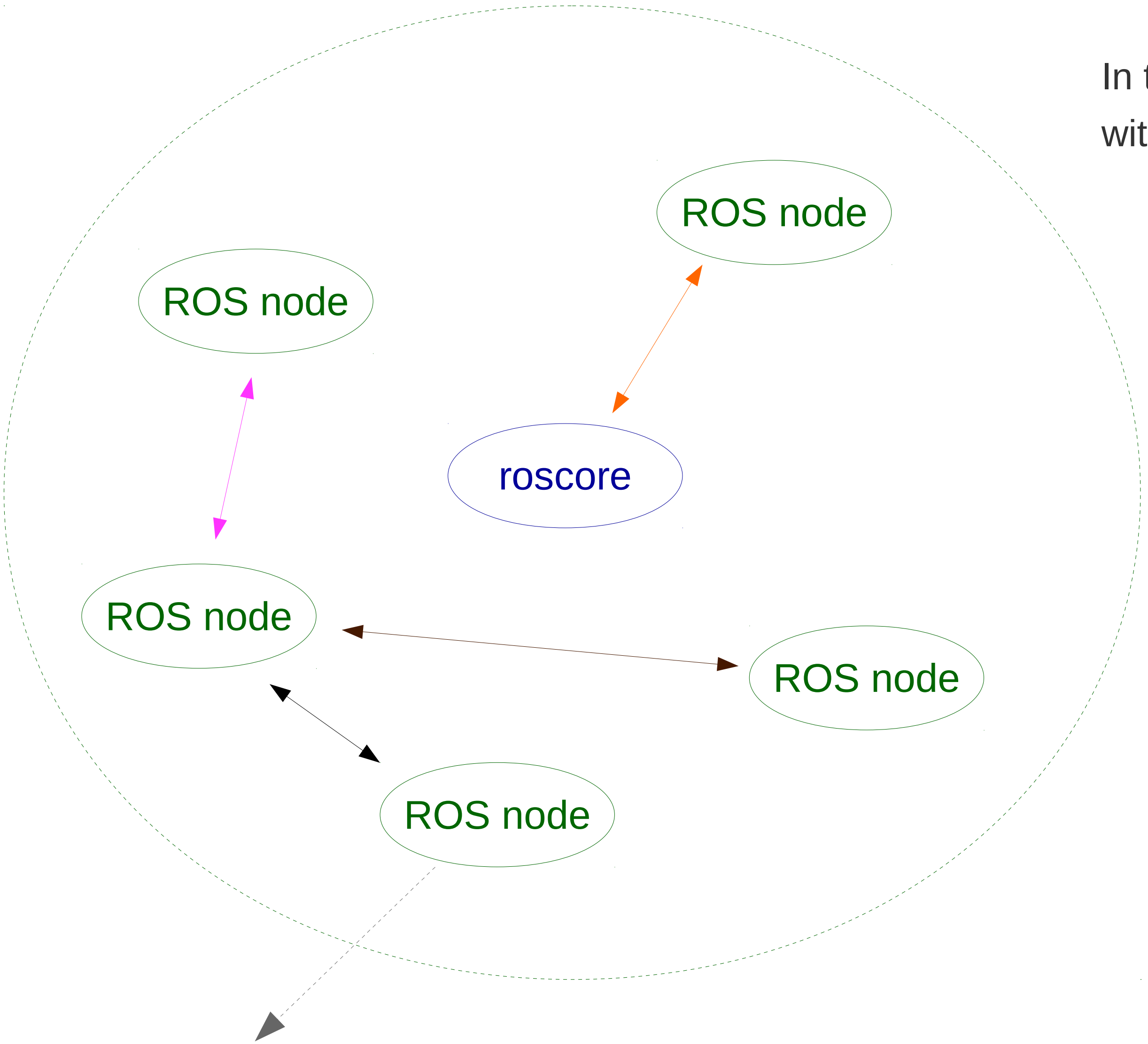
It is a Peer-to-Peer network of executables (ROS nodes) with only one manager (*roscore*) that is providing to them:

1. Naming/registration services (through Master)
2. Some state/parameters (through Parameter Server)
3. Logging (through *rosout*)

In this network each ROS node communicate with each other peer-to-peer through Messages:

1. In an asynchronous way using Topics:
   Using TCPROS/UDPROS
2. In a synchronous way using Services:
   Using TCPROS
3. Through preemptable tasks:
   Using *actionlib*

It is possible to save and play back all the messages between ROS nodes using Bags



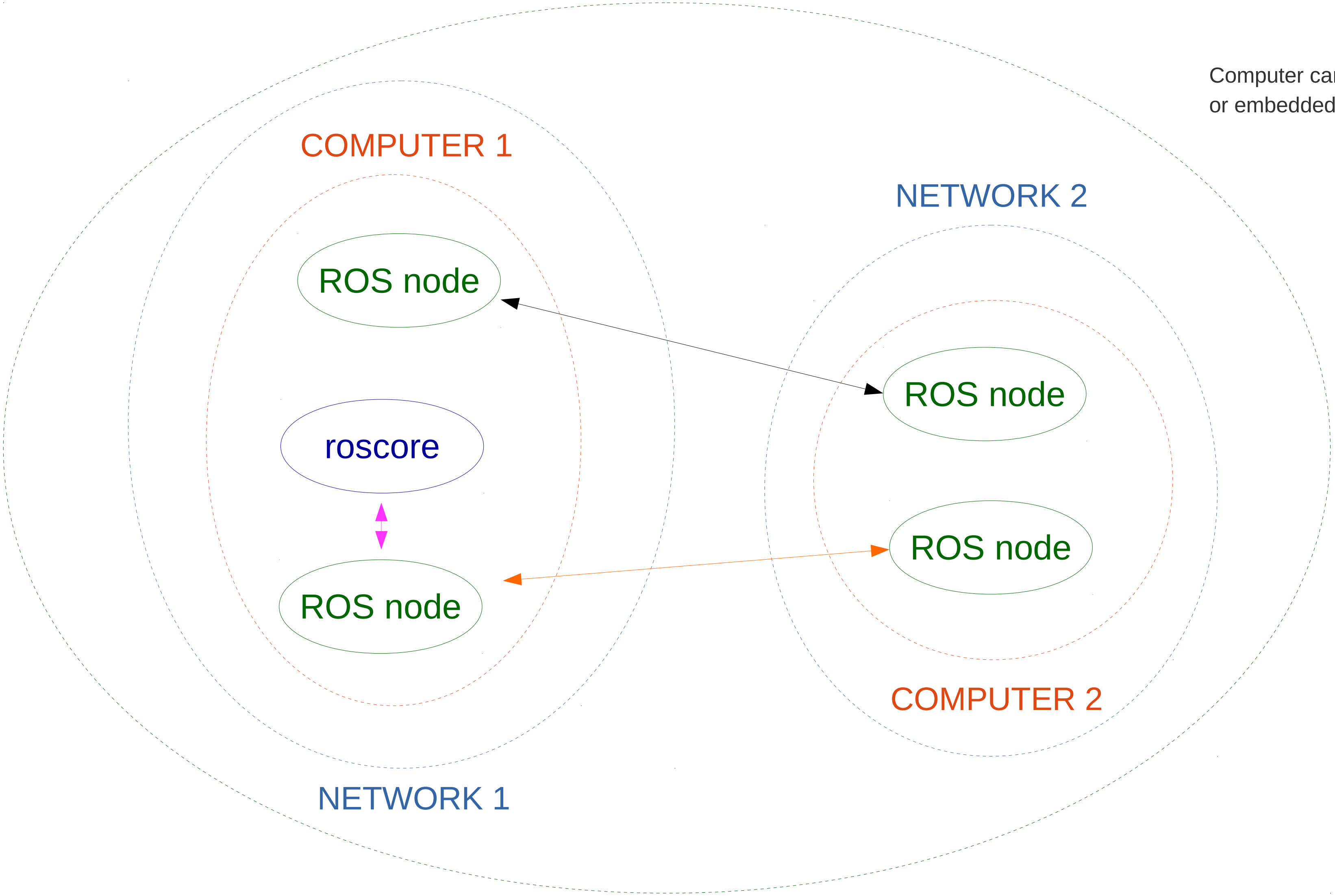
A ROS Node is a custom program developed by the ROS 's end user, in order to fulfill some goal that is useful for this user

See more at: http://wiki.ros.org/ROS/Technical%20Overview

ROS Network

The ROS network can expand to different computers and networks

Computer can be any PC, laptop, or embedded system



COMPUTER 1

ROS node

roscore

ROS node

NETWORK 1

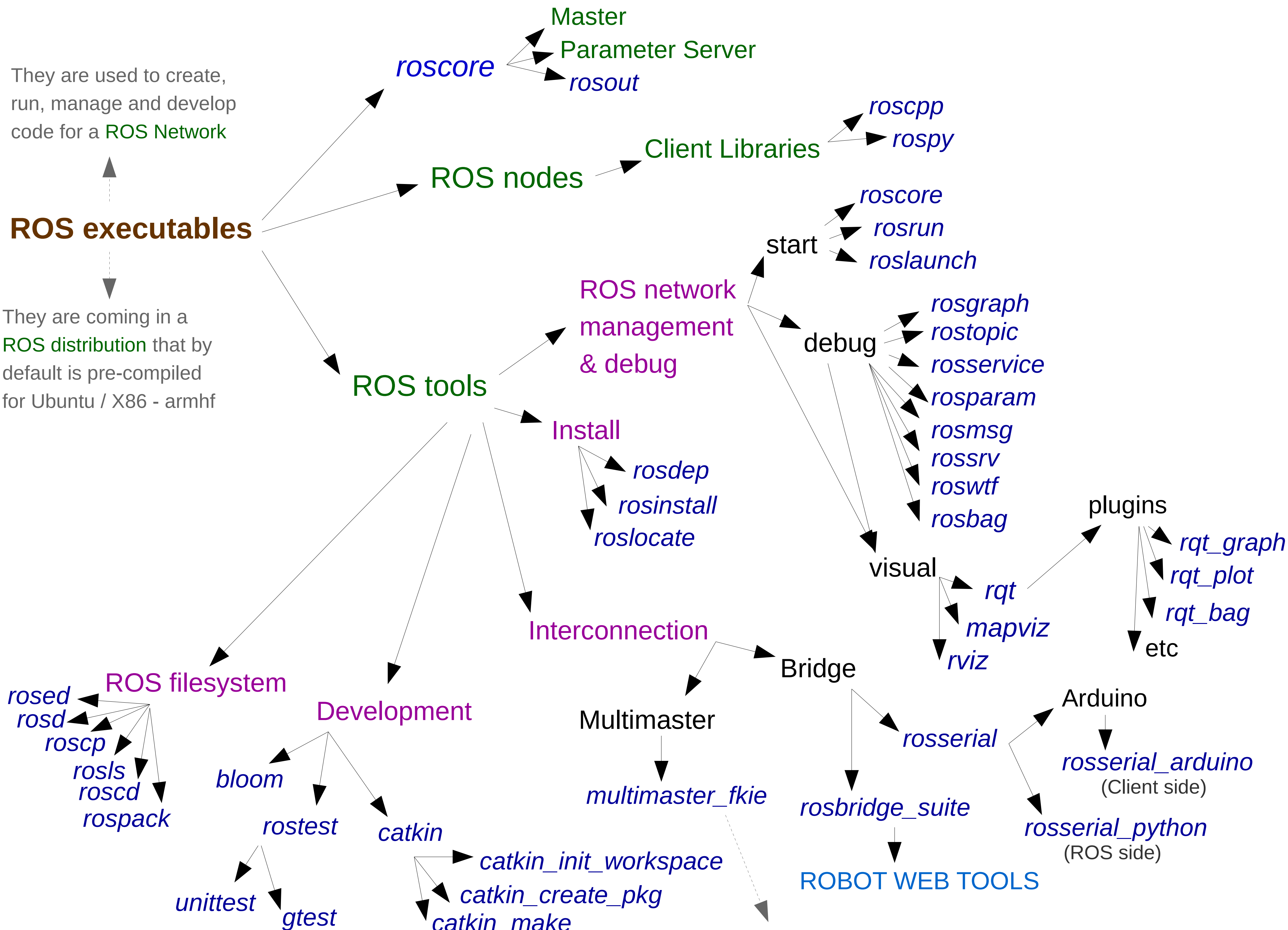
NETWORK 2

ROS node

ROS node

COMPUTER 2

ROS is a framework that enables to create, run, manage and develop code for a ROS Network



ROS executables

They are used to create, run, manage and develop code for a ROS Network

They are coming in a ROS distribution that by default is pre-compiled for Ubuntu / X86 - armhf

- *roscore*
  - Master
  - Parameter Server
  - *rosout*
- ROS nodes
  - Client Libraries
    - *roscpp*
    - *rospy*
- ROS tools
  - ROS network management & debug
    - start
      - *roscore*
      - *rosrun*
      - *roslaunch*
    - debug
      - *rosgraph*
      - *rostopic*
      - *rosservice*
      - *rosparam*
      - *rosmsg*
      - *rossrv*
      - *roswtf*
      - *rosbag*
    - visual
      - *rqt*
        - plugins
          - *rqt_graph*
          - *rqt_plot*
          - *rqt_bag*
          - etc
      - *mapviz*
      - *rviz*
  - Install
    - *rosdep*
    - *rosinstall*
    - *roslocate*
  - Interconnection
    - Multimaster
      - *multimaster_fkie*
    - Bridge
      - *rosserial*
        - Arduino
          - *rosserial_arduino* (Client side)
        - *rosserial_python* (ROS side)
      - *rosbridge_suite*
        - ROBOT WEB TOOLS
  - ROS filesystem
    - *rosed*
    - *rosd*
    - *roscp*
    - *rosls*
    - *roscd*
    - *rospack*
  - Development
    - *bloom*
    - *rostest*
      - *unittest*
      - *gtest*
    - *catkin*
      - *catkin_init_workspace*
      - *catkin_create_pkg*
      - *catkin_make*

See more at: http://goo.gl/iNVvcL

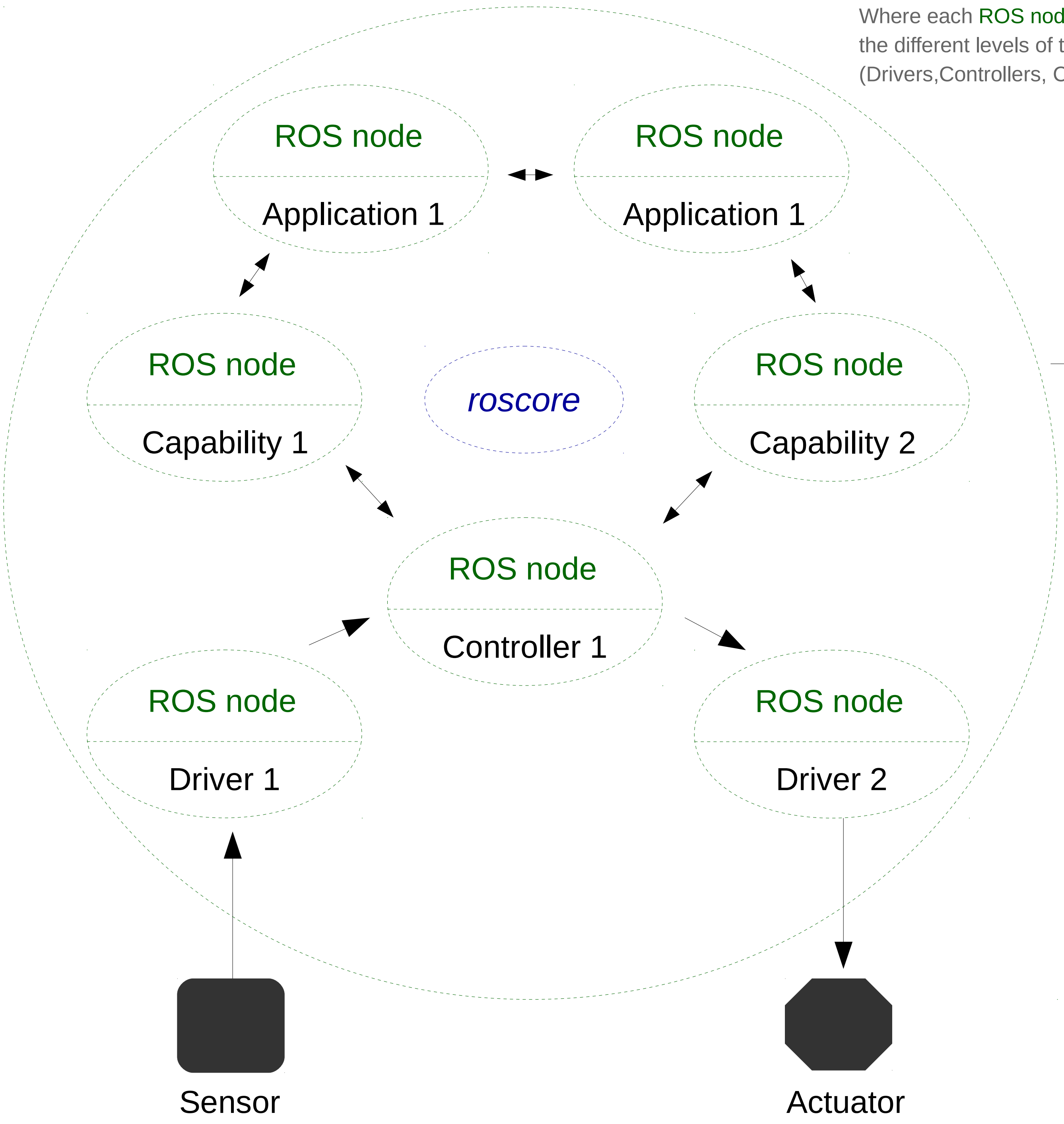
ROS for Robots

Robot

A ROS Network is used to control a robot

Where each ROS node is providing functionality for the different levels of the functional robot abstraction (Drivers,Controllers, Capabilities and Applications)

It is in this sense that ROS is considered a Robot Operating System

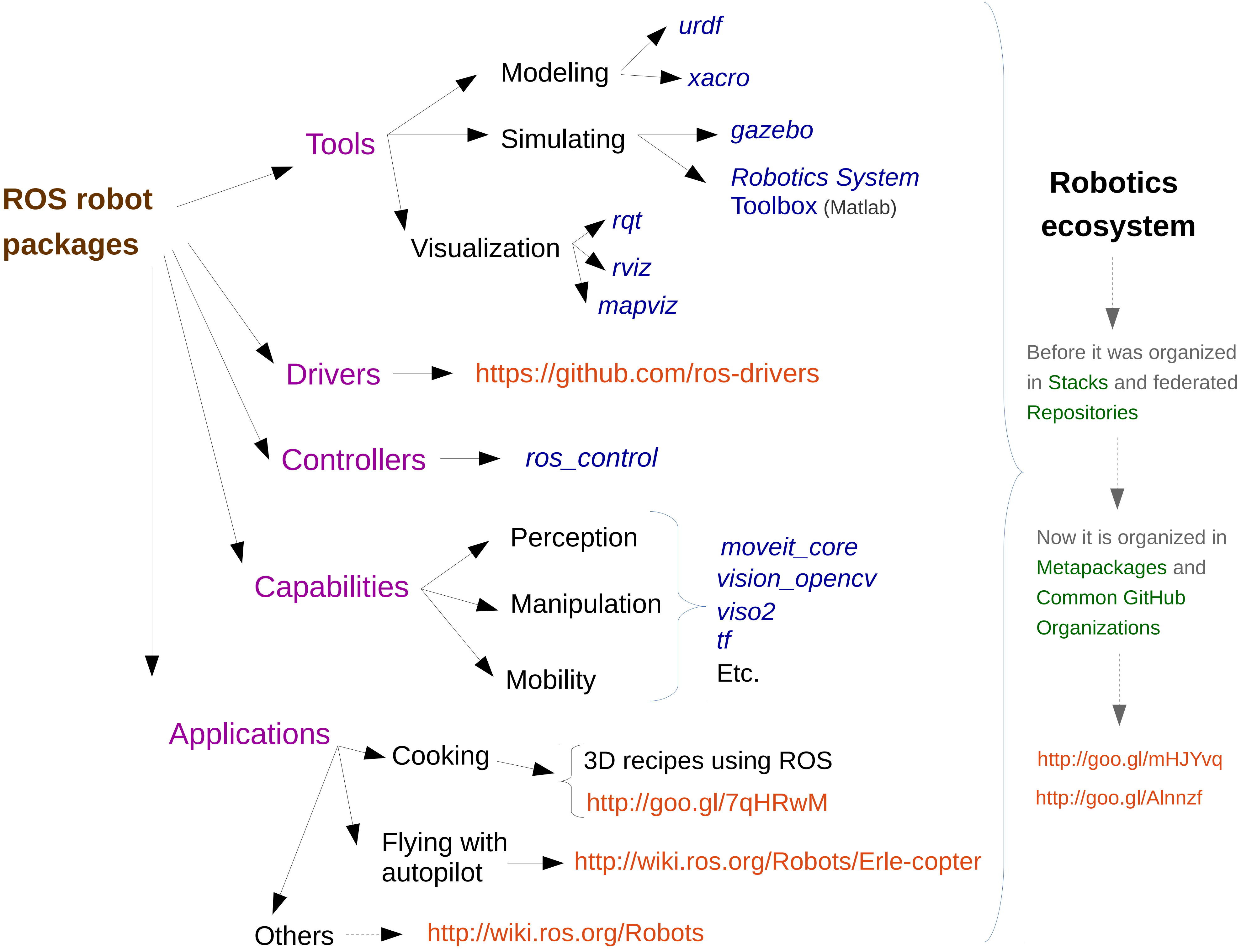


ROS is a framework that offers a standard way and a common infrastructure to develop, run, share and reuse code for robotics

It helps to focus in the development of new Applications while reusing code for Capabilities, Controllers and Drivers

It enables a much faster robotics research and development

See more at:
- http://goo.gl/uUVKPm
- http://goo.gl/HRgo6r



ROS robot packages

Tools

Modeling

urdf

xacro

Simulating

gazebo

Robotics System Toolbox (Matlab)

Visualization

rqt

rviz

mapviz

Drivers

https://github.com/ros-drivers

Controllers

ros_control

Capabilities

Perception

Manipulation

Mobility

moveit_core

vision_opencv

viso2

tf

Etc.

Applications

Cooking

3D recipes using ROS

http://goo.gl/7qHRwM

Flying with autopilot

http://wiki.ros.org/Robots/Erle-copter

Others

http://wiki.ros.org/Robots

Robotics ecosystem

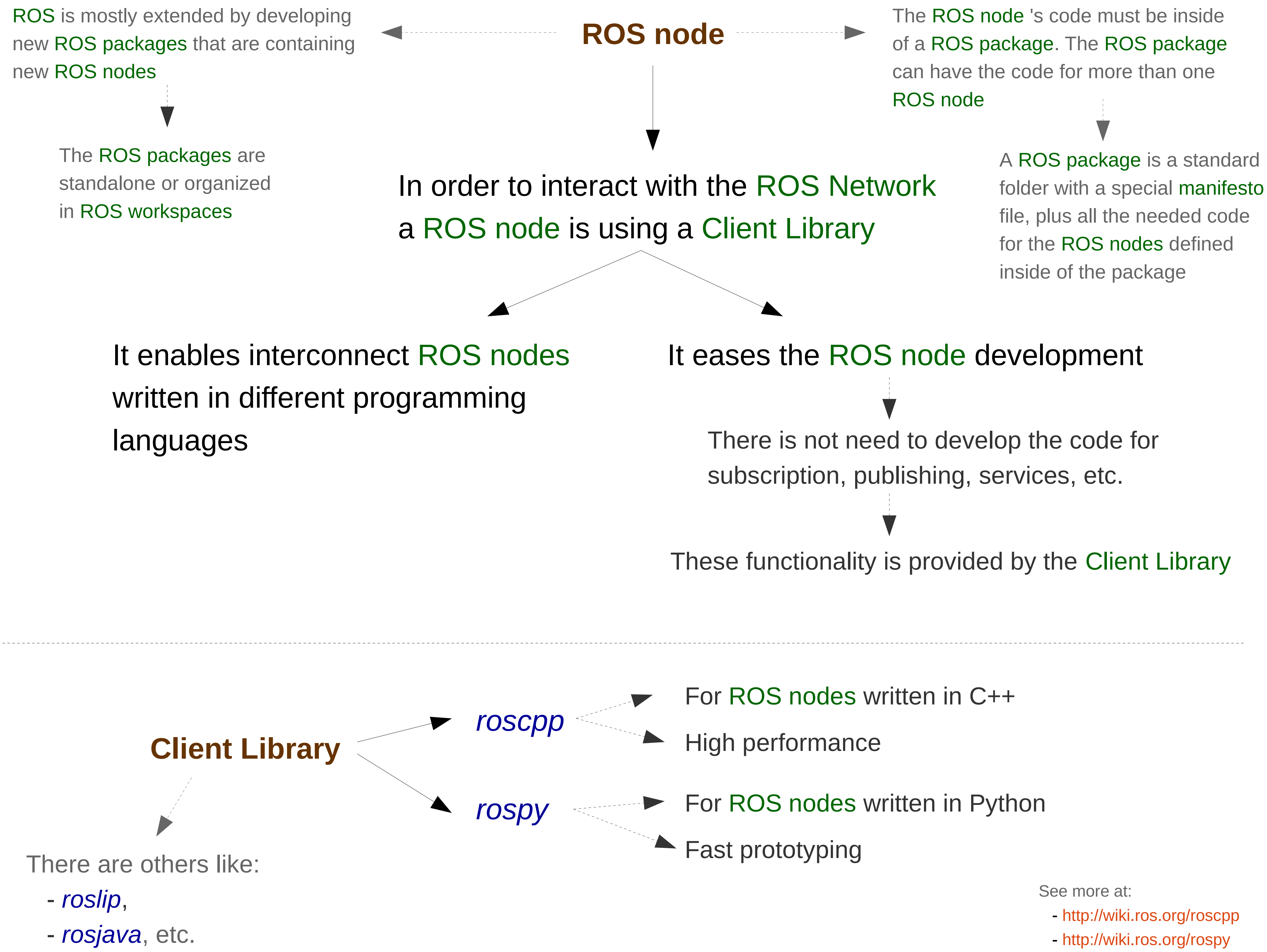
Before it was organized in Stacks and federated Repositories

Now it is organized in Metapackages and Common GitHub Organizations

http://goo.gl/mHJYvq

http://goo.gl/Alnnzf

ROS Development



ROS node

ROS is mostly extended by developing new ROS packages that are containing new ROS nodes

The ROS packages are standalone or organized in ROS workspaces

The ROS node 's code must be inside of a ROS package. The ROS package can have the code for more than one ROS node

A ROS package is a standard folder with a special manifesto file, plus all the needed code for the ROS nodes defined inside of the package

In order to interact with the ROS Network a ROS node is using a Client Library

It enables interconnect ROS nodes written in different programming languages

It eases the ROS node development

There is not need to develop the code for subscription, publishing, services, etc.

These functionality is provided by the Client Library

Client Library

roscpp

For ROS nodes written in C++

High performance

rospy

For ROS nodes written in Python

Fast prototyping

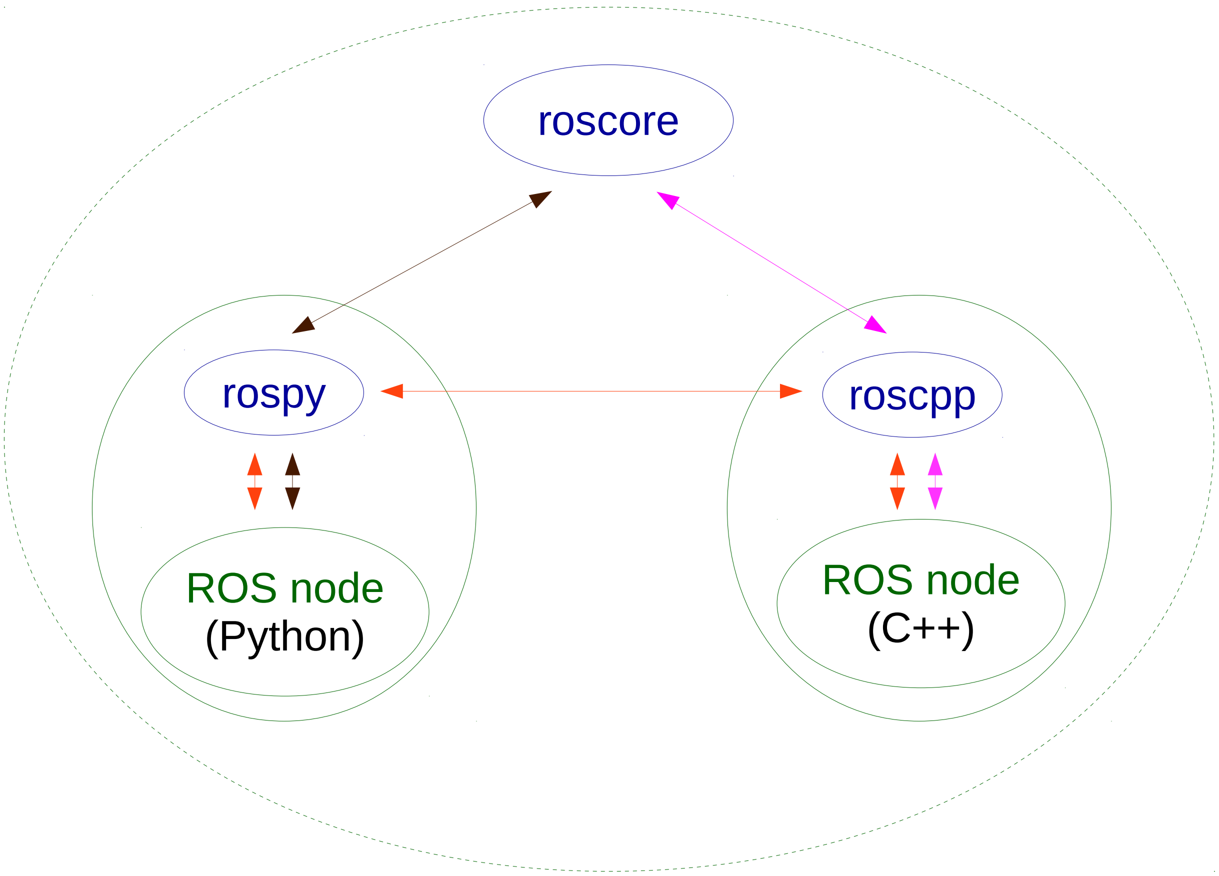
There are others like:

- *roslip*,
- *rosjava*, etc.

See more at:

- http://wiki.ros.org/roscpp
- http://wiki.ros.org/rospy

ROS Network



See more at:

- http://wiki.ros.org/catkin/workspaces
- http://wiki.ros.org/catkin/Tutorials
- http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv
- http://wiki.ros.org/Packages

- http://www.cse.sc.edu/~jokane/agitr/agitr-letter.pdf
- http://goo.gl/NIkfPY

ROS workspaces / packages

-> The workspace folder is created with *mkdir*:

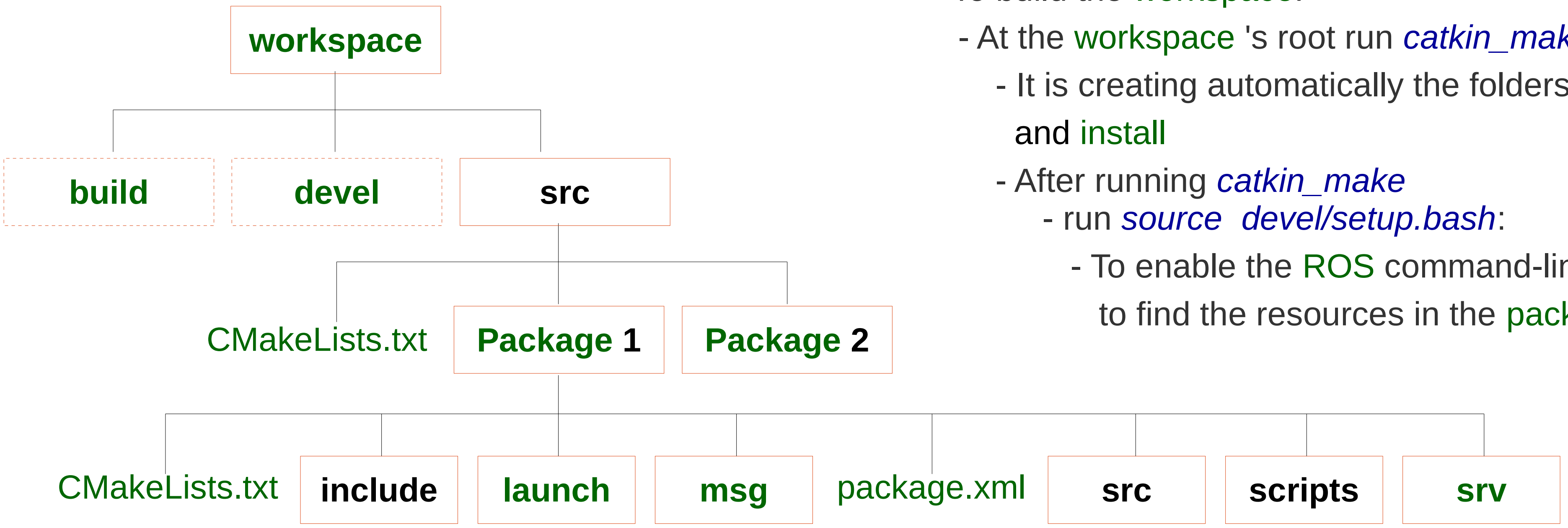
- It could have any name
- Inside it is created the folder **src** with *mkdir*:
  - When **src** is created, inside it only once, is run *catkin_init_workspace*:
    - It is creating automatically the workspace 's top level CmakeLists.txt
  - Inside **src** are created the ROS packages with *catkin_create_pkg*:
    - It is creating automatically:
      - The package 's folder
      - Inside the package 's folder:
        - The package 's CmakeLists.txt
        - The package 's manifesto:
          - package.xml

-> To build the workspace:

- At the workspace 's root run *catkin_make*:
  - It is creating automatically the folders build, *devel* and install
  - After running *catkin_make*
    - run *source devel/setup.bash*:
      - To enable the ROS command-line tools to find the resources in the packages

Structure of the ROS workspaces and ROS packages

This structure is also known as the ROS filesystem



workspace

build

devel

src

CMakeLists.txt

Package 1

Package 2

CMakeLists.txt

include

launch

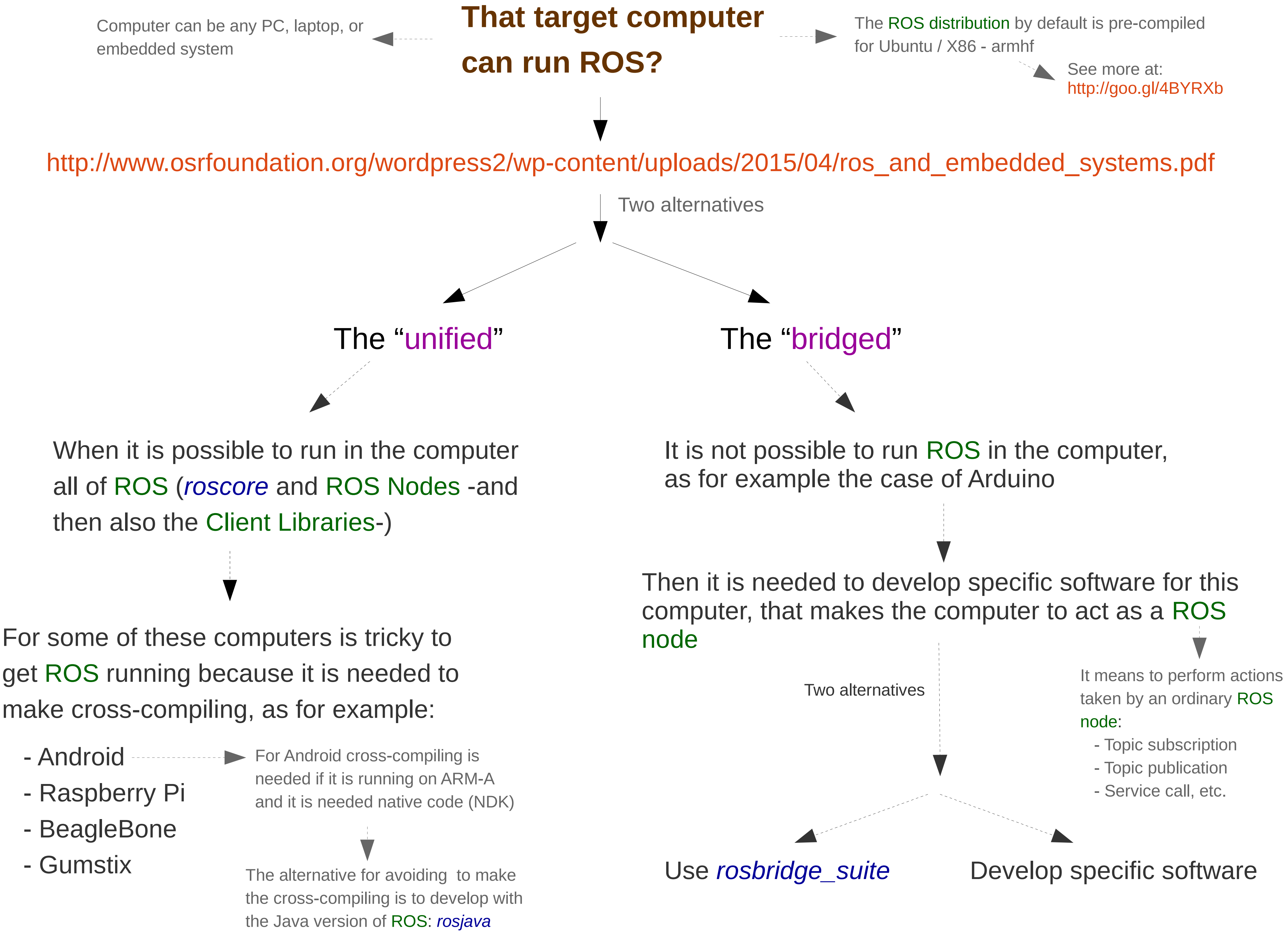
msg

package.xml

src

scripts

srv



That target computer can run ROS?

Computer can be any PC, laptop, or embedded system

The ROS distribution by default is pre-compiled for Ubuntu / X86 - armhf

See more at:
http://goo.gl/4BYRXb

http://www.osrfoundation.org/wordpress2/wp-content/uploads/2015/04/ros_and_embedded_systems.pdf

Two alternatives

The "unified"

When it is possible to run in the computer all of ROS (*roscore* and ROS Nodes -and then also the Client Libraries-)

For some of these computers is tricky to get ROS running because it is needed to make cross-compiling, as for example:

- Android
- Raspbery Pi
- BeagleBone
- Gumstix

For Android cross-compiling is needed if it is running on ARM-A and it is needed native code (NDK)

The alternative for avoiding to make the cross-compiling is to develop with the Java version of ROS: *rosjava*

The "bridged"

It is not possible to run ROS in the computer, as for example the case of Arduino

Then it is needed to develop specific software for this computer, that makes the computer to act as a ROS node

It means to perform actions taken by an ordinary ROS node:

- Topic subscription
- Topic publication
- Service call, etc.

Two alternatives

Use *rosbridge_suite*

Develop specific software

The “bridged” alternative

See more at:
http://cs.okstate.edu/~chriscrick/Crick11b.pdf

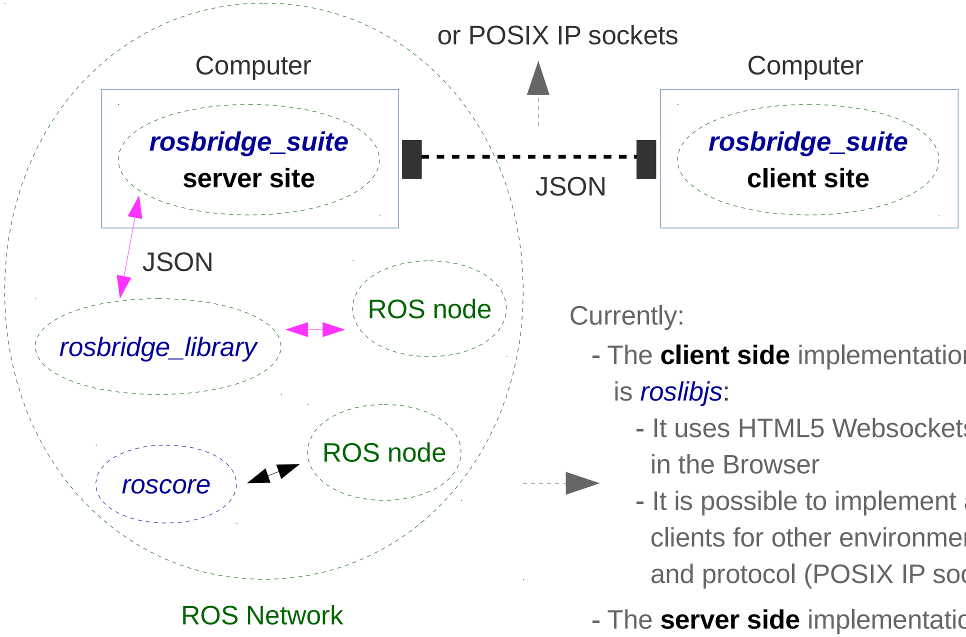
It is a bit outdated, but the concept is OK

Two alternatives

Use *rosbridge_suite*

The standard bridge

When it is possible to use HTML5 Websockets or standard POSIX IP sockets to connect the computer to where the ROS network runs:



HTML5 Websockets
or POSIX IP sockets

Currently:

- The **client side** implementation is *roslibjs*:
  - It uses HTML5 Websockets and runs in the Browser
  - It is possible to implement another clients for other environments and protocol (POSIX IP sockets)
- The **server side** implementation is *rosbridge_server*:
  - It only supports HTML5 Websockets
  - It is possible to implement other that supports POSIX IP sockets

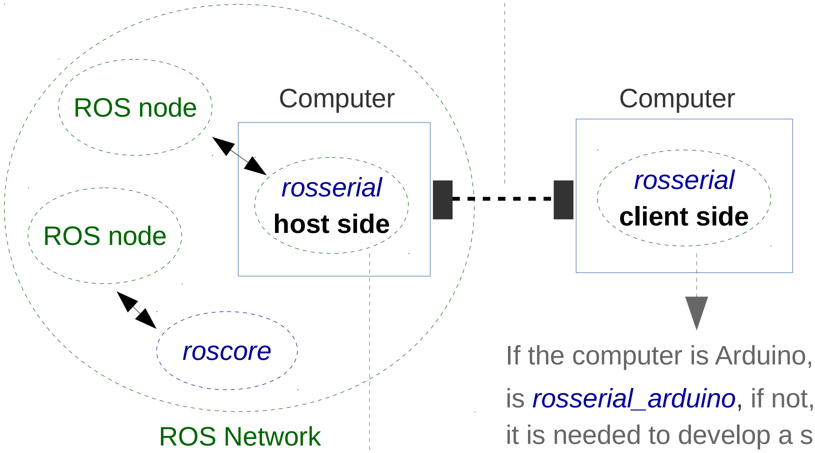
Develop specific software

This software is also known as a bridge

It, as in the case of Arduino, also can act as a driver

When it is not possible to use HTML5 Websockets or standard POSIX IP sockets, as for example when the computer is only connected through a serial interface to where the ROS Network runs

An example of this kind of specific software (a new bridge) is *rosserial*:



Serial communication

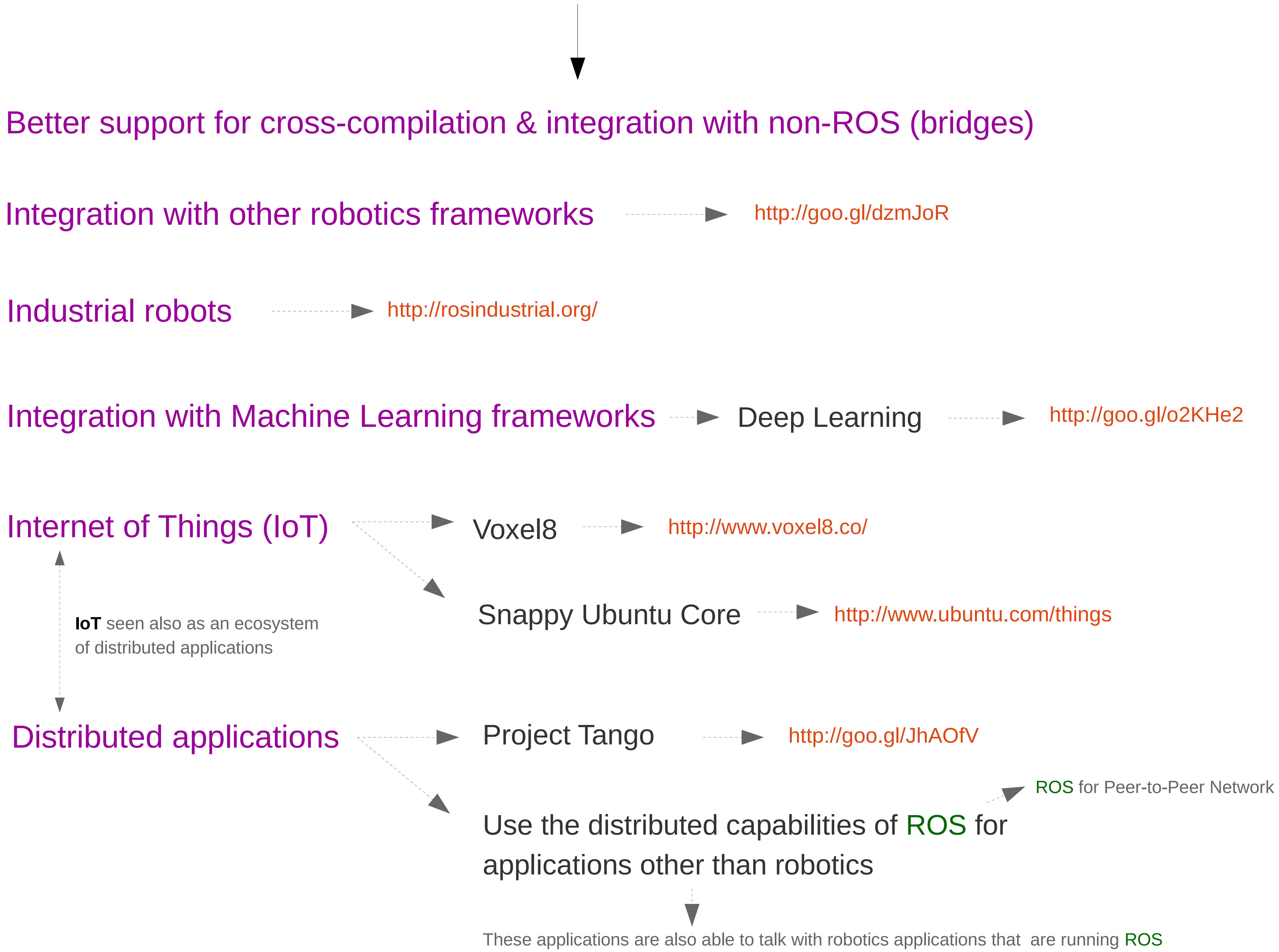
rosserial_python

If the computer is Arduino, it is *rosserial_arduino*, if not, then it is needed to develop a specific *rosserial* **client** software

ROS future

See more at: http://goo.gl/3eVn4T

The future?



Better support for cross-compilation & integration with non-ROS (bridges)

Integration with other robotics frameworks → http://goo.gl/dzmJoR

Industrial robots → http://rosindustrial.org/

Integration with Machine Learning frameworks → Deep Learning → http://goo.gl/o2KHe2

Internet of Things (IoT) → Voxel8 → http://www.voxel8.co/

Internet of Things (IoT) → Snappy Ubuntu Core → http://www.ubuntu.com/things

IoT seen also as an ecosystem of distributed applications

Distributed applications → Project Tango → http://goo.gl/JhAOfV

Distributed applications → Use the distributed capabilities of ROS for applications other than robotics

ROS for Peer-to-Peer Network

These applications are also able to talk with robotics applications that are running ROS