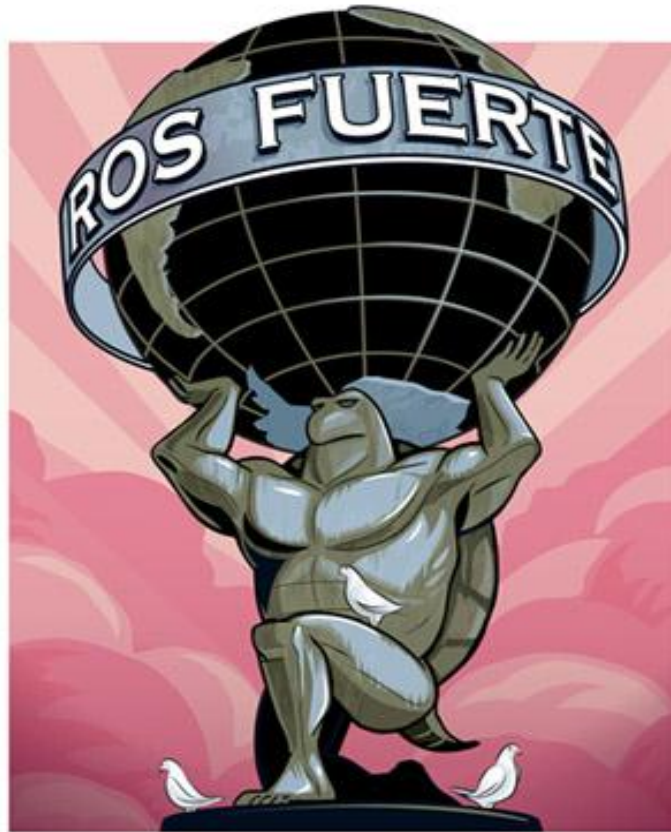


# ROS: Tips n' Tricks



by: Anqi Xu

COMP-417, McGill University  
Monday September 17th, 2012

Inspired by presentation of Marek Doniec, MIT

# Overview

- What is ROS?
- Features
- Philosophies
- Core elements
- Development pipeline
- Useful ROS nodes / tools



# What is ROS ?



- Meta operating system (or middleware)
- Features:
  - Message-passing / network abstraction
  - Package management
  - Tools & libraries for obtaining, building, writing, and running code across different computers
- “The primary goal of ROS is to support code *reuse* in robotics research and development.”

# ROS: Features

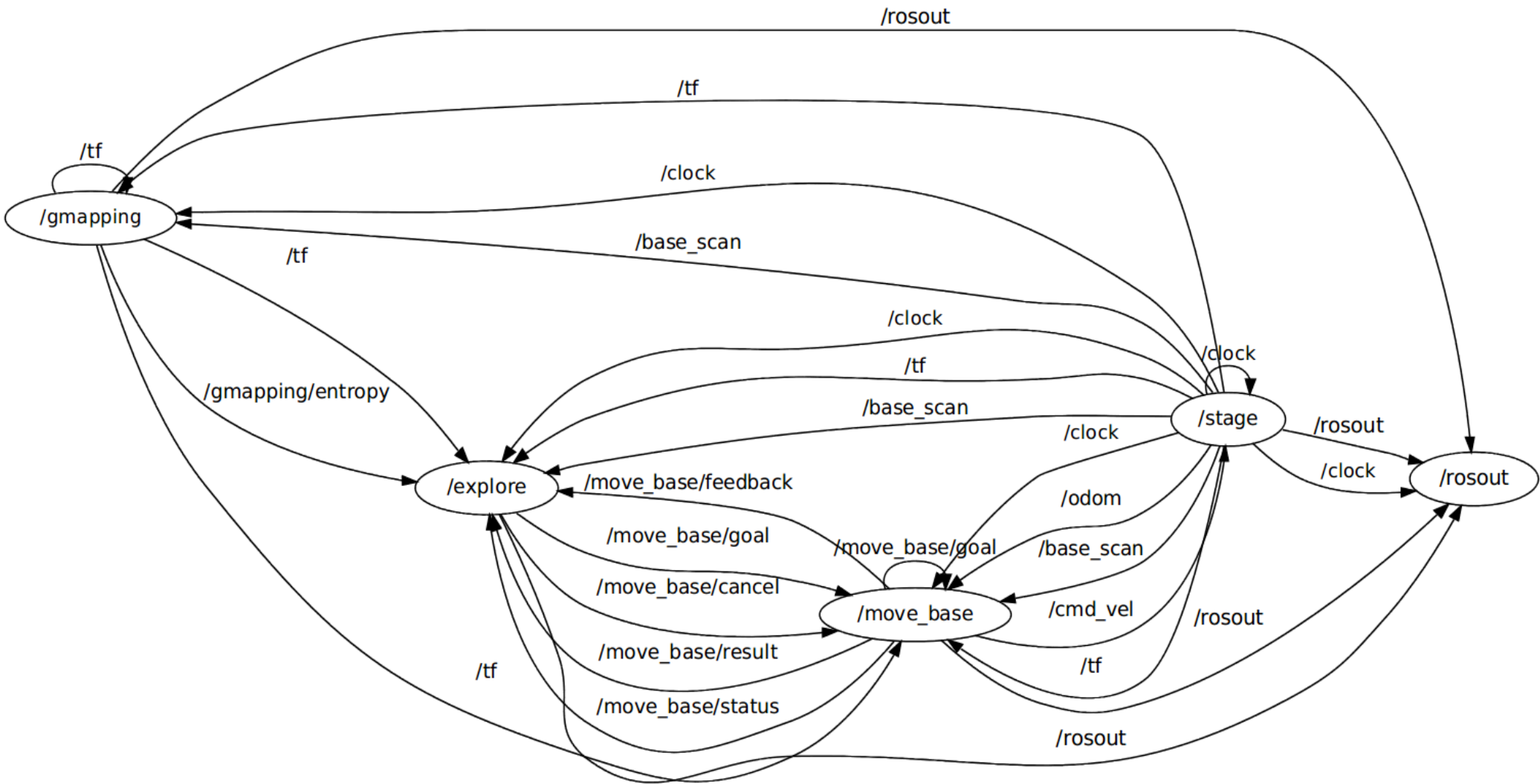
- Library dependencies
  - Compilation & Linking (via manifest.xml)
  - Non-ROS libraries (via CMake)
- Communication layer
  - Inter-process
  - Inter-machine
- Process management

# ROS: Philosophies

- Modular & peer-to-peer
- Language independent
- Thin
- Free & open-source



# ROS: Modularity & Peer-to-Peer



# ROS: Language Independence

- Client interfaces:
  - Stable: rospy, roscpp, roslisp
  - Experimental: rosjava, roslua
  - Contributed: rosserial, roshask, ipc-bridge (MATLAB), etc...
- Common message-passing layer
  - Interface Definition Language (IDL)

# ROS: Thin

- Library-style interaction
  - Facilitates integration with other common libraries, e.g.: OpenCV, PCL, bullet, etc...
- Minimal dependencies
  - Linked against (lean) ROS core + necessary message meta-libraries + specific libraries
- “The preferred development model is to write ROS-agnostic libraries with clean functional interfaces.”

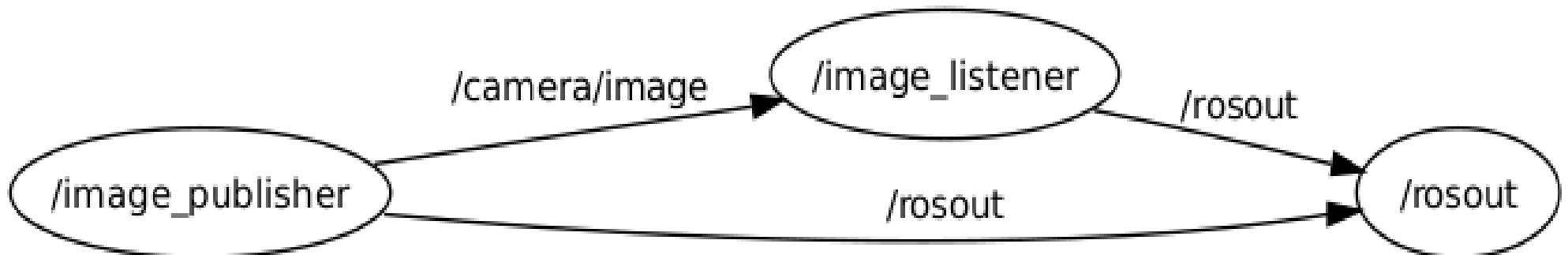


# ROS: Free & Open-Source

- Source code is publicly available
- ROS Core & 1st party tools are under BSD license
- Contributed tools are under a variety of open-source (& closed-source) licenses
- Promotes code-reuse and community-building

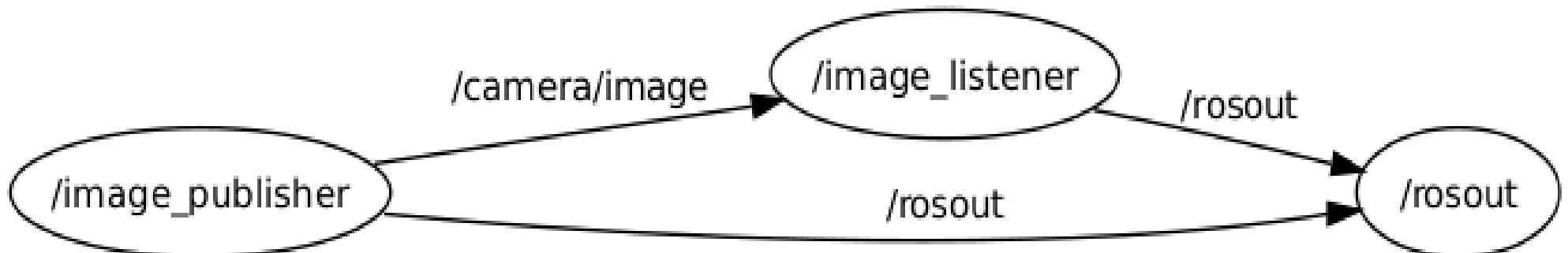
# ROS: Core Elements

- Nodes
- Messages, topics, and services
- ROS Master
- Parameters
- Bags



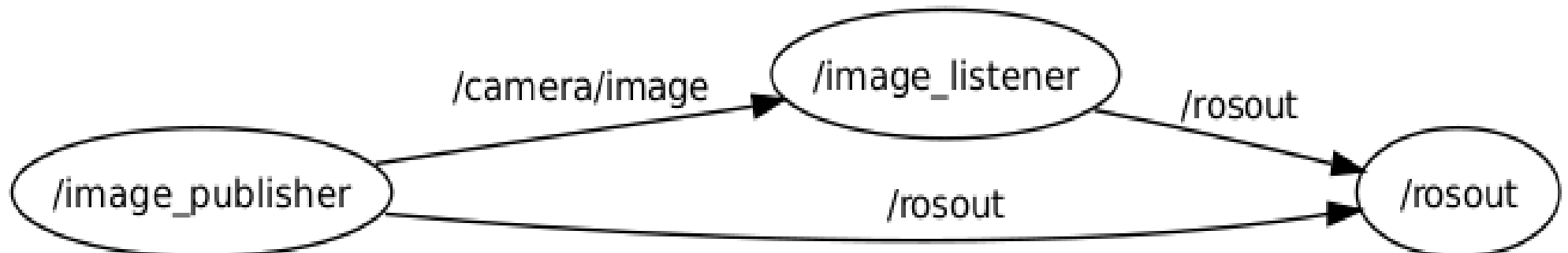
# ROS Nodes

- Single-purposed executable programs
  - e.g. sensor driver(s), actuator driver(s), mapper, planner, UI, etc...
- Modular design
  - Individually compiled, executed, and managed
- Nodelets



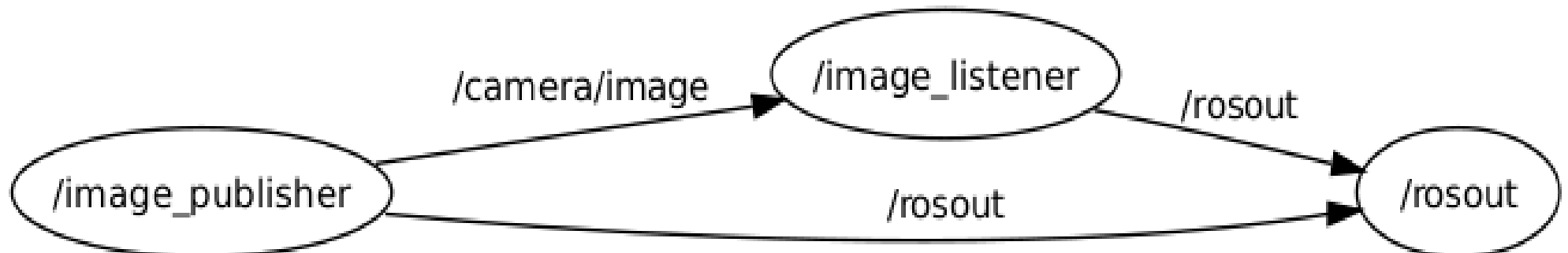
# ROS Messages

- Strictly-typed data structures for inter-node communication
- Interface Definition Language (IDL)
  - Basic types: bool, [u]int{8,16,32,64}, float{32/64}, string, time, duration, etc...
  - Arrays
  - Nested messages: image, GPS, IMU, etc...



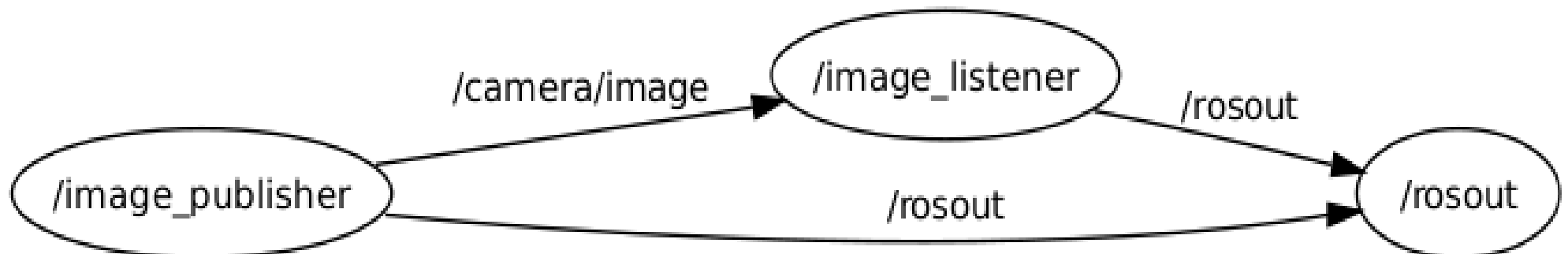
# ROS Topics

- Unidirectional (& often repeated) inter-node communication
- Publish/Subscribe model: 1-to-N broadcasting
- Examples:
  - provide sensor readings
  - provide actuator states / robot feedback



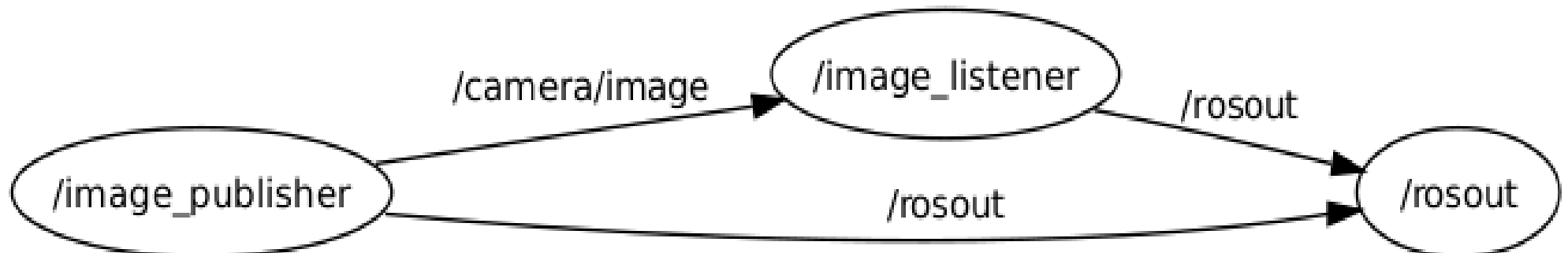
# ROS Services

- Synchronous inter-node transactions / RPC
- Service/Client model: 1-to-1 request-response
- Examples:
  - carry out remote computation
  - trigger functionality / behavior
  - (\*) dynamically update parameter

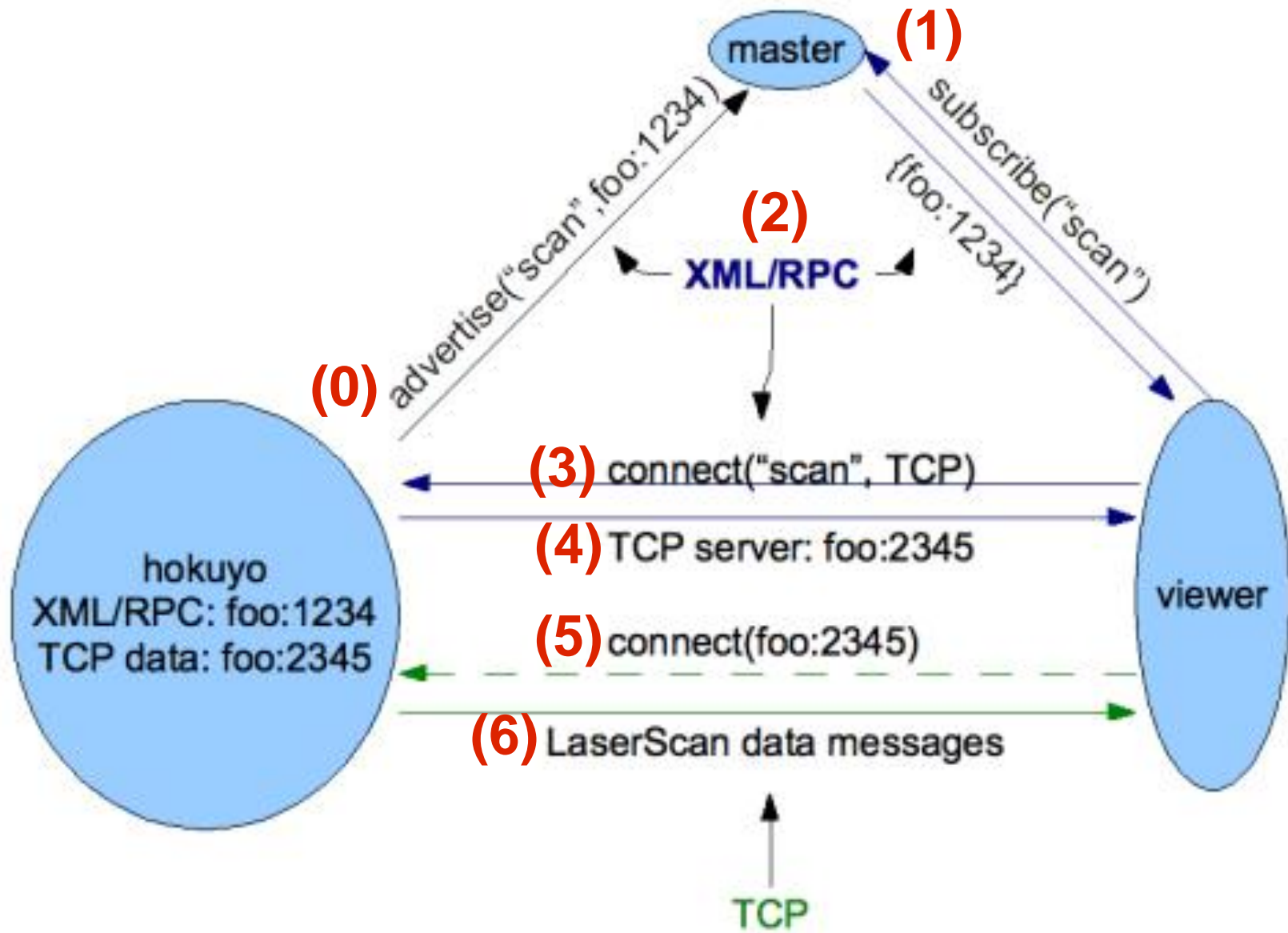


# ROS Master

- Naming & registration services for nodes, topics, services, etc...
- “The role of the Master is to enable ROS nodes to locate one another.”
- Critical environment variable:
  - `$ROS_MASTER_URI=http://[HOST]:[PORT]`

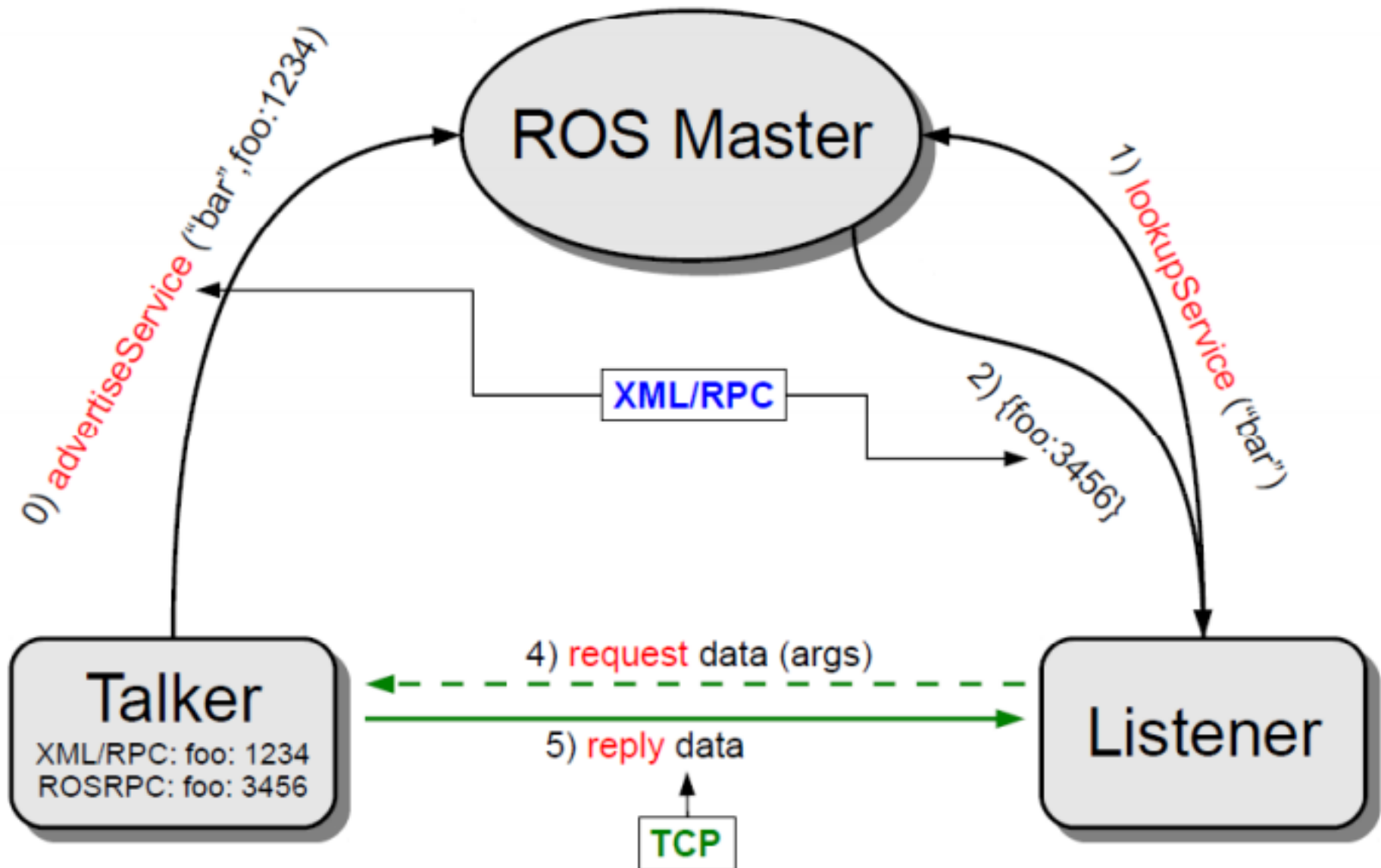


# ROS: Topic Subscription



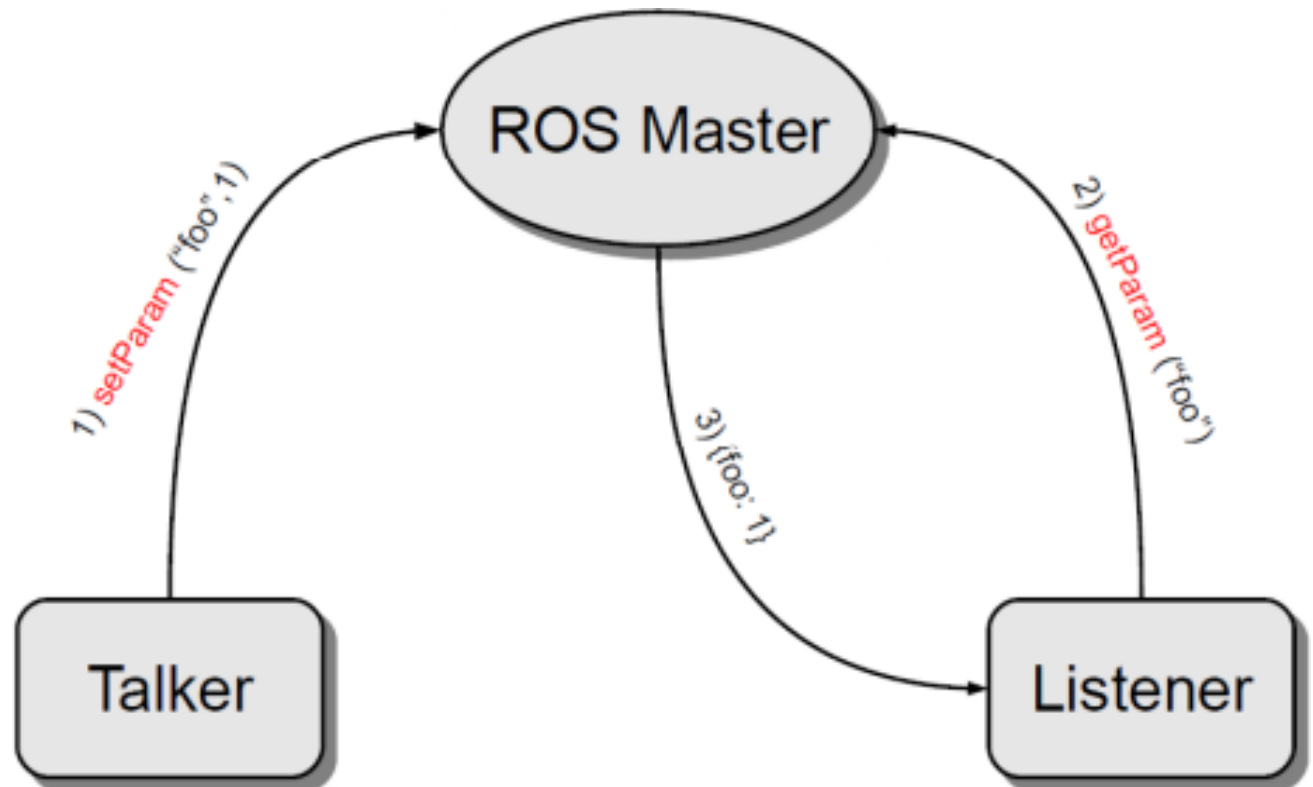


# ROS: Service Call



# ROS Parameters

- Configurable setting
  - Node-specific
  - Launch-time
- Unique name
- Typed (XMLRPC)
- Can be remapped at runtime

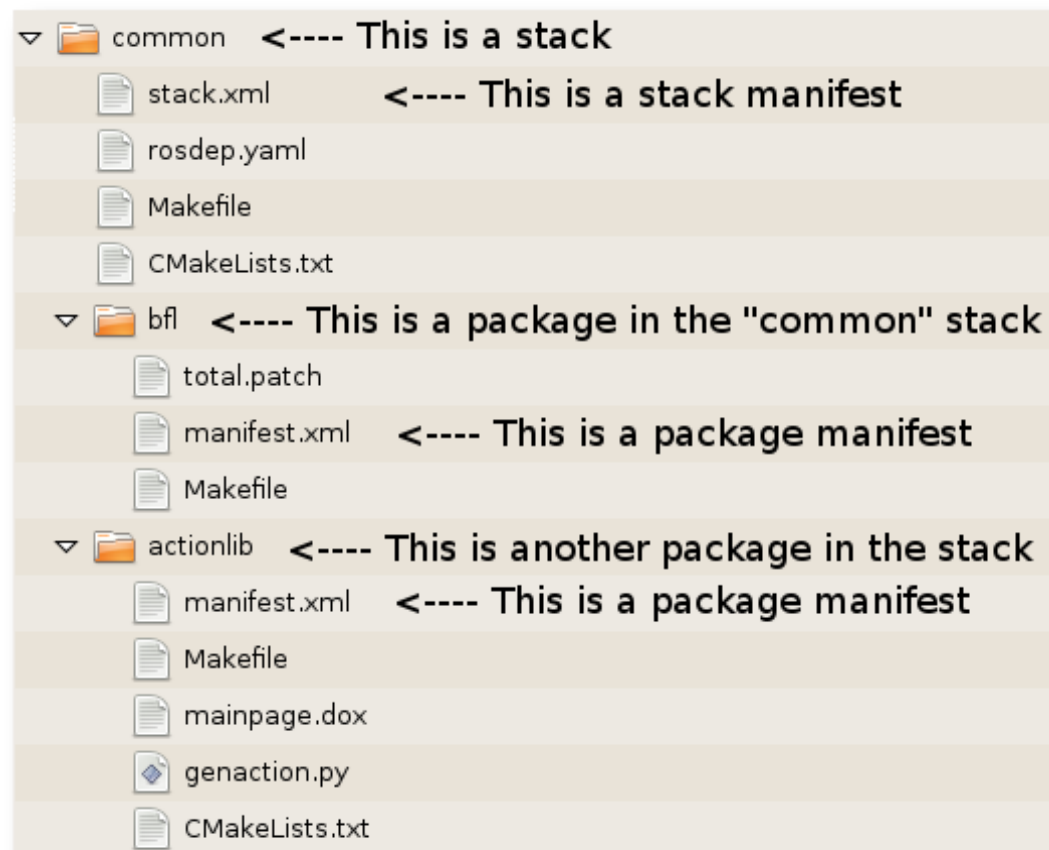


# ROS Bags

- Standardized logging & playback mechanism
  - Single-file container for serialized messages
- Playback using simulated / real clock
  - (Near-)realistic message playback

# ROS: Packages & Stacks

- Package: nodes, messages, service types, configuration files, etc...
- Stack: collection of packages



# ROS Development Pipeline

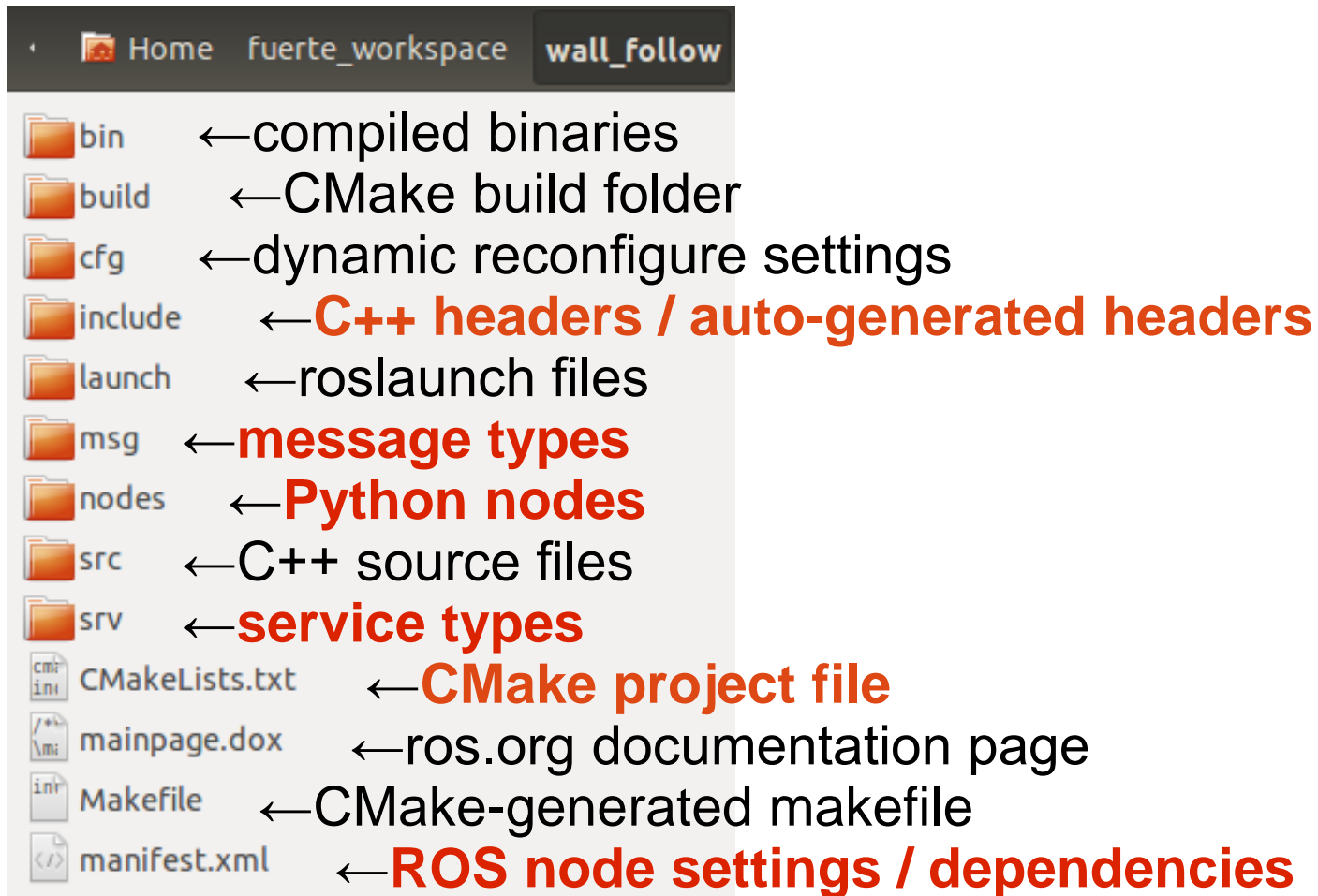
- Setup: `rosws`, `roscrcat`
- Navigate: `roscd`, `roscd`
- Configure: `manifest.xml`, `CmakeLists.txt`
- Build: `rosmake`
- Execute: `roscore`, `roslaunch`, `roslaunch`, `roslaunch`
- Inspect: `rostopic`, `rostopic`, `rosservice`
- Debug: `rostopic`, `rostopic`, `rxgraph`
- Log & Analyze: `roscd`, `roscd`

# Setup: ros\_ws

- Manages ROS workspace
- Provides standard interface to revisioned code
- `ros_ws init ~/fuerte_workspace /opt/ros/fuerte`
- `roslocate info teleop_base | ros_ws merge -`
- `ros_ws info control_toolbox`
- `ros_ws update turtlebot`

# Setup: roscrate-pkg

- Creates new ROS package
- `roscrate-pkg wall_follow rospy sensor_msgs`



# Navigate: roscd / rosed

- Provides fast navigation to ROS folders / files
- roscd stage
- rosed wall\_follow wall\_follower.py
- export EDITOR='emacs -nw'



# Configure: manifest.xml

- Lists properties and ROS node dependencies

```
<package>
  <description>A simple viewer for ROS image topics.</description>
  <author>Patrick Mihelich</author>
  <license>BSD</license>
  <review status="Doc reviewed" notes="Dec 17, 2009"/>
  <url>http://www.ros.org/wiki/image\_view</url>

  <export>
    <cpp cflags="-I${prefix}/include" lflags="-Wl,-rpath,${prefix}/lib -L${prefix}/lib -limage_view" />
    <nodelet plugin="${prefix}/nodelet_plugins.xml" />
  </export>

  <rosdep name="opencv2"/>
  <depend package="cv_bridge"/>
  <depend package="image_transport"/>
  <depend package="nodelet" />
  <depend package="roscpp"/>
  <depend package="sensor_msgs"/>
  <depend package="stereo_msgs"/>
</package>
```

# Configure: CMakeLists.txt

- Configures build environment & lists non-ROS dependencies

```
cmake_minimum_required(VERSION 2.4.6)
include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake)
set(ROS_BUILD_TYPE RelWithDebInfo)
```

```
rosbuild_init()
```

```
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
set(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)
```

```
rosbuild_genmsg()
rosbuild_gensrv()
```

```
#rosbuild_add_boost_directories()
#rosbuild_add_library(${PROJECT_NAME} src/example.cpp)
#target_link_libraries(${PROJECT_NAME} another_library)
#rosbuild_link_boost(${PROJECT_NAME} thread)
#rosbuild_add_executable(example examples/example.cpp)
#target_link_libraries(example ${PROJECT_NAME})
```

# Build: rosmake

- Builds ROS node
  - packages with msg/srv must be built, even if they only contain Python nodes
- `rosmake wall_follow`
- `(roscd wall_follow; mkdir build; cd build; cmake ..)`

# Execute: roscore

- Starts ROS master
  - Only 1 master per machine
  - Can use remote ROS master instead
  - Hostname **must** be defined in /etc/hosts
- roscore
- export ROS\_MASTER\_URI=<http://remote:11311>

# Execute: rosrun

- Executes ROS node
  - namespace remappings
  - node-specific parameters (use '--' to delineate)
- `roslaunch image_view image_view`  
`image:=/camera/image_raw -- compressed`

# Execute: roslaunch

- Executes multiple ROS nodes concurrently
  - Launch files should have .launch extension, to be indexed automatically
  - Initialize ROS Master automatically if needed
- `roslaunch image_proc image_proc.launch`

```
<roslaunch>  
  <node pkg="turtlesim" node="turtlesim_node" required="true" />  
  <node pkg="turtlesim" node="turtle_teleop_key" output="screen" />  
</roslaunch>
```

# Execute: rosparam

- Absolute parameters: /node/param
- Local parameters: ~param
- YAML syntax
  
- rosparam list
- rosparam set /foo "['1', 1, 1.0]"
  
- rosrun camera1394 camera1394\_node  
\_\_video\_mode="640x480\_mono8"

# Inspect: rosnode, rostopic, rosservice

- Lists nodes/topics/services and provides information on their states
- rosnode list
- rostopic echo /turtle1/pose
- rostopic pub -1 /text\_listener std\_msgs/String – "Hello World"
- rostopic hz /camera/image\_raw
- rosservice call /kill



# Debug: rostest

- Executes unit test on ROS node
  - C++: gtest, Python: unittest
  - Manifest.xml: `<depend package="rosunit" />`
  - CmakeLists.txt:  
`rosbuild_add_rostest(test/mynode.test)`
  - ROSLaunch: `<test test-name="test_mynode" pkg="mypkg" type="test_mynode" />`
- `rostest mypkg test_mynode.test`

# Debug: rostest (cont.)

```
#!/usr/bin/env python
PKG = 'mypkg'
import roslib; roslib.load_manifest(PKG)

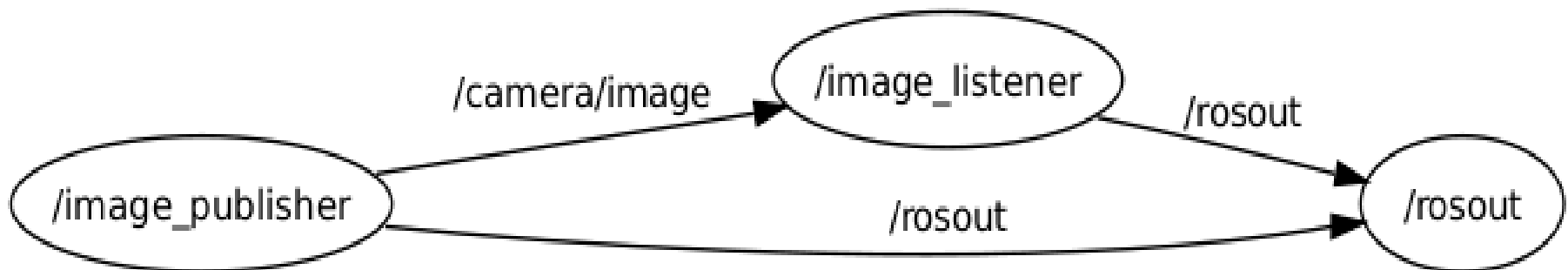
import sys
import unittest

## A sample python unit test
class TestBareBones(unittest.TestCase):
    ## test 1 == 1
    def test_one_equals_one(self):
        self.assertEqual(1, 1, "1!=1")

if __name__ == '__main__':
    import rostest
    rostest.rostest(PKG, 'test_bare_bones', TestBareBones)
```

# Debug: roswtf & rxgraph

- roswtf: finds trivial mistakes (e.g. ROS node not responding, missing publisher, etc...)
- rxgraph: visualize nodes & messages

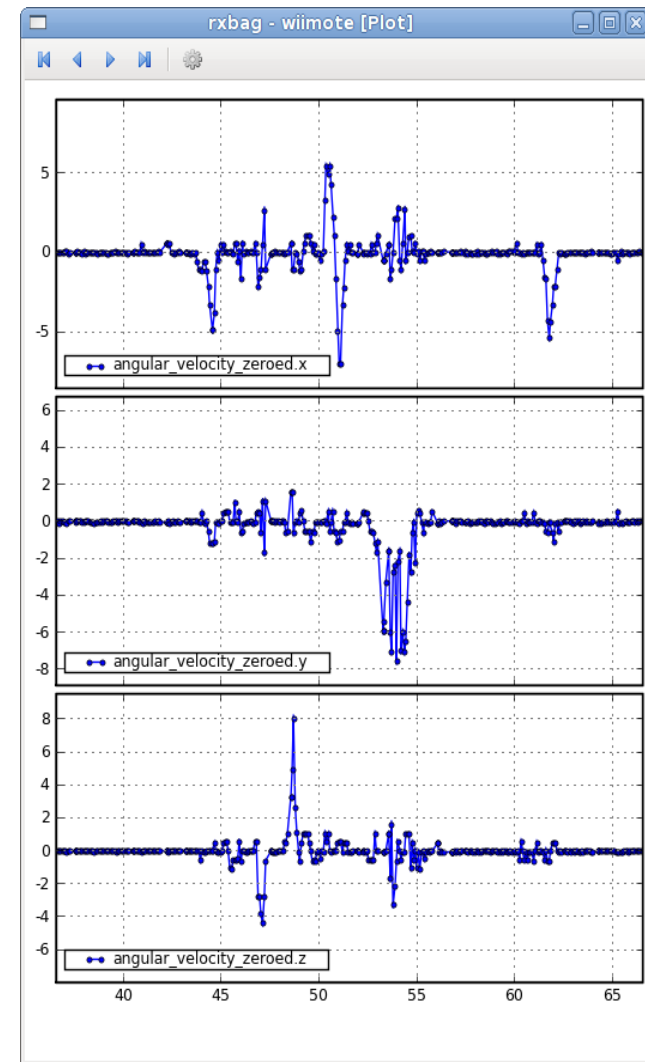
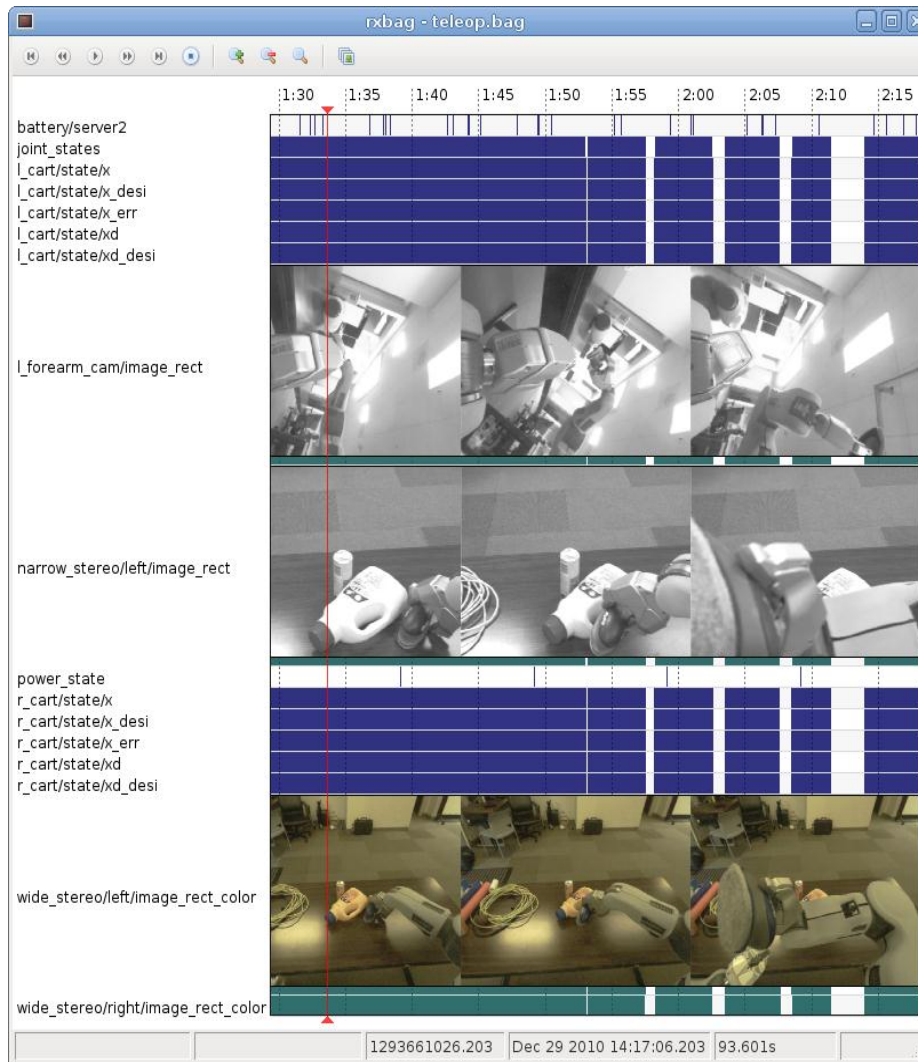


# Log & Analyze: rosbag

- Records and plays back sets of messages
- `rosbag record -a`
- `rosbag record rosout tf turtle1/pose`
- `rosbag play -r 1.5 -s 30 -l yesterday.bag`
- `rosbag compress *.bag`
- `rosbag reindex *.bag`

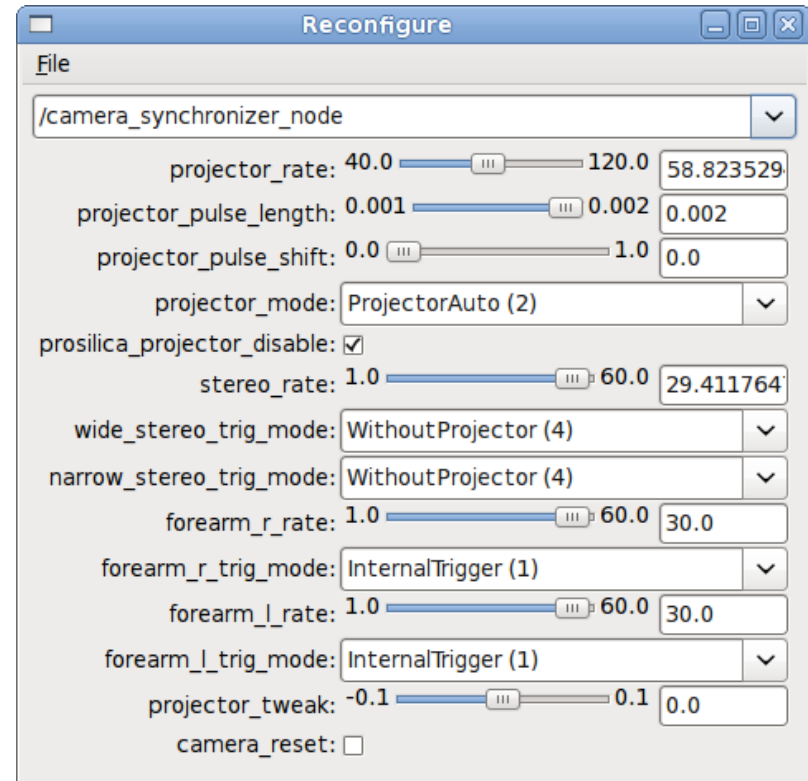
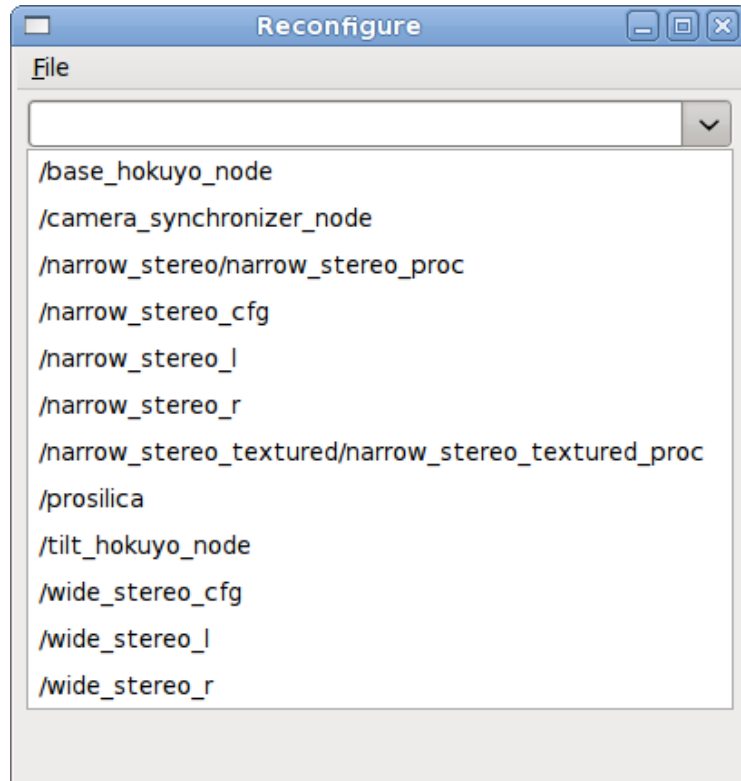
# Log & Analyze: rxbag

- GUI for visualizing rosbag content



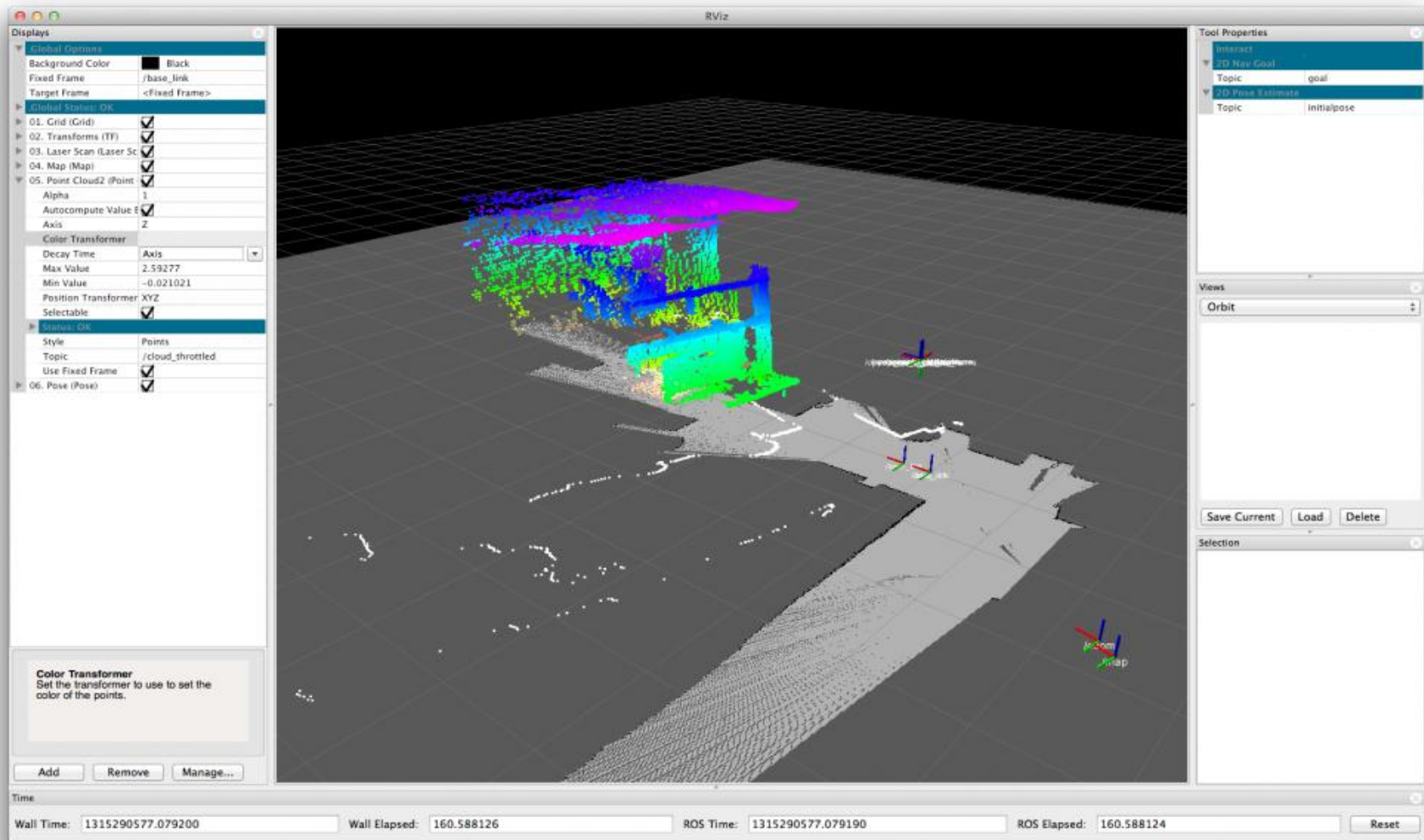
# Tool: dynamic\_reconfigure

- Changes parameters dynamically in runtime
  - NOT the same as (static) ROS parameters!!!
- `roslaunch dynamic_reconfigure reconfigure_gui`



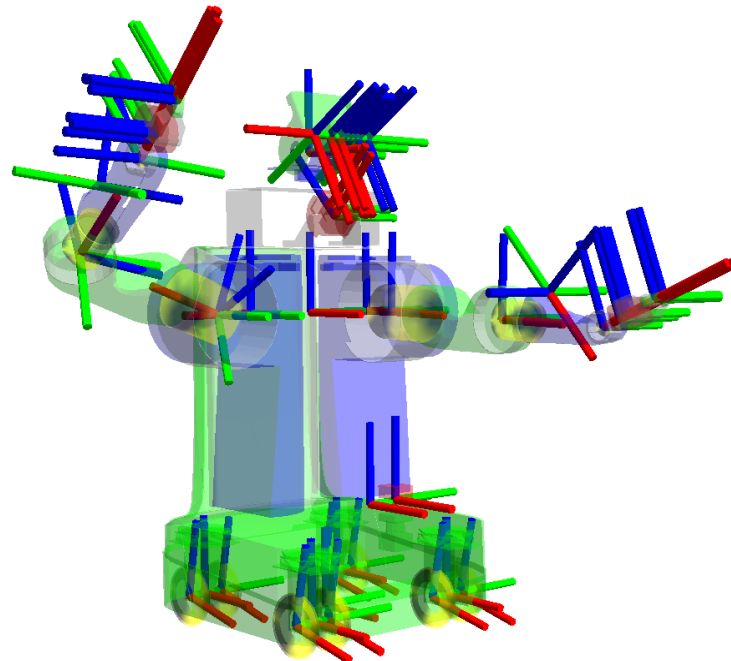
# Tool: rviz

- Fully-featured 3D visualization environment



# Tool: tf

- Abstraction & manipulation of multiple coordinate frames
  - (Timed) conversion between different frames
  - Distributed broadcaster/listener model

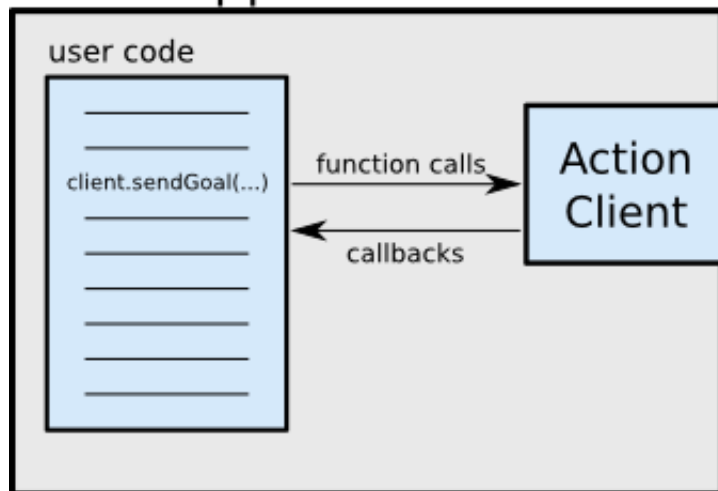




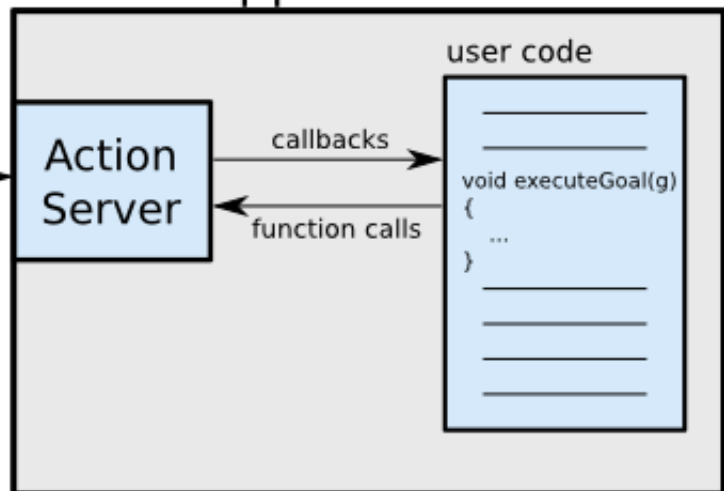
# Tool: actionlib

- Lengthy / complex /multi-step services
  - Goal: desired end-result
  - Feedback: incremental progress towards goal
  - Result: feedback upon completion of goal

Client Application



Server Application



ROS

# Questions?

- [www.ros.org/wiki/Documentation?action=AttachFile&do=get&target=ROScheatsheet.pdf](http://www.ros.org/wiki/Documentation?action=AttachFile&do=get&target=ROScheatsheet.pdf)
- [www.ros.org/browse/list.php](http://www.ros.org/browse/list.php)
- [answers.ros.org](http://answers.ros.org)
- [planet.ros.org](http://planet.ros.org)

