

# **Introduction to ROS**

Lorenz Mösenlechner

Technische Universität München

July 18th, 2012



# Motivation

- Today's robotic systems are complex.
- Many sensors.
- Highly distributed, many processes, many computers.
- Teams of engineers.



# Motivation

- Today's robotic systems are complex.
- Many sensors.
- Highly distributed, many processes, many computers.
- Teams of engineers.

⇒ ROS — The Robot Operating System.



# Outline

Overview

ROS Communication Layer

ROS Build System

Programming with ROS

The TF Library



# Outline

Overview

ROS Communication Layer

ROS Build System

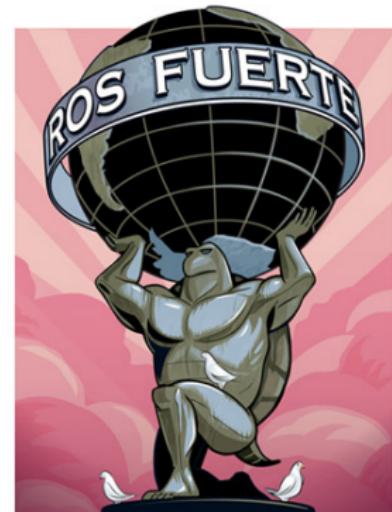
Programming with ROS

The TF Library

# What is ROS?

## More than just a middleware

- A “meta” operating system for robots
- A collection of packaging, software building tools
- An architecture for distributed inter-process/inter-machine communication and configuration
- Development tools for system runtime and data analysis
- A language-independent architecture (c++, python, lisp, java, *and more*)





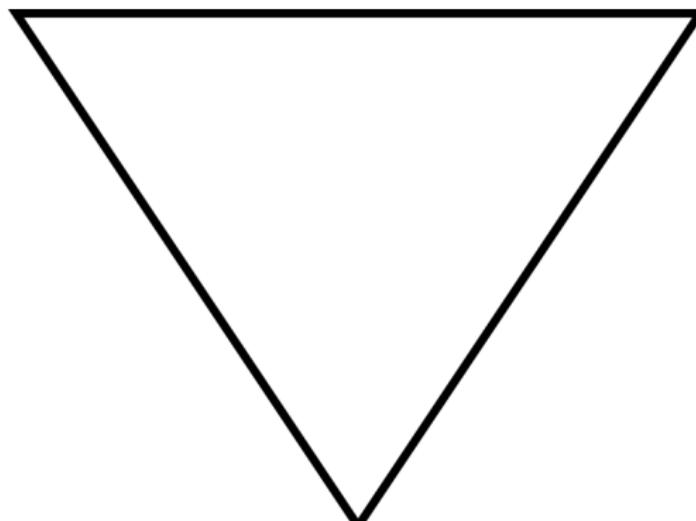
# What is ROS not?

## No confusion

- An *actual* operating system
- A programming language
- A programming environment / IDE
- A hard real-time architecture

# What does ROS get you?

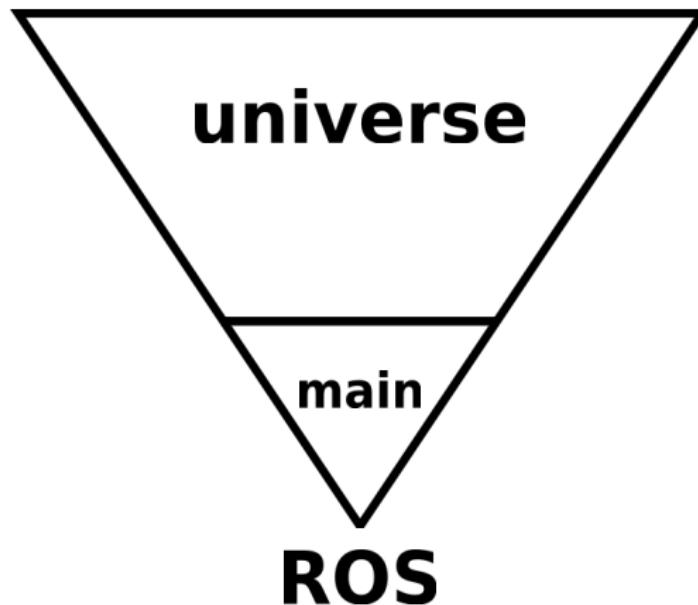
## All levels of development



**ROS**

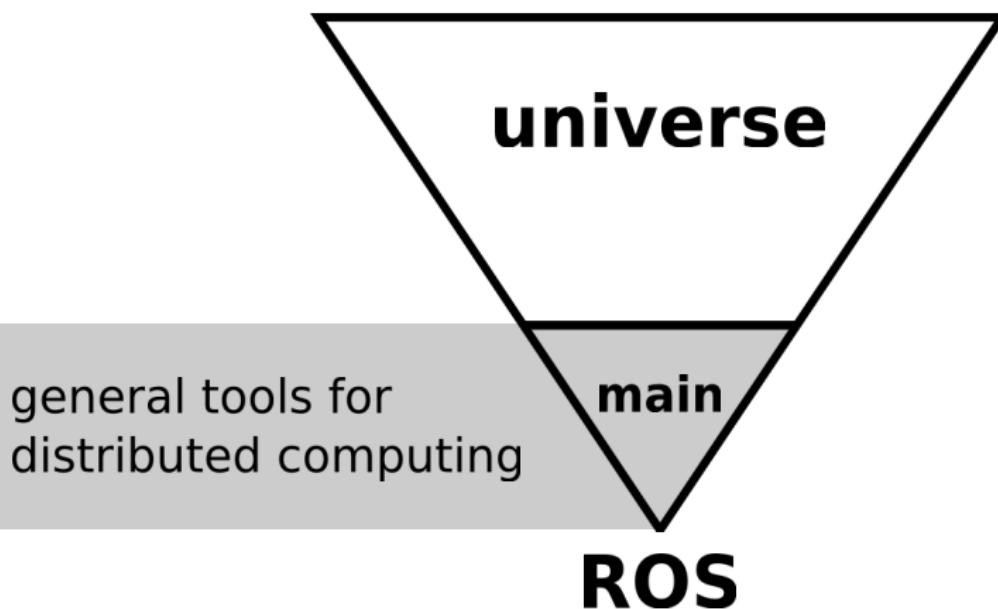
# What does ROS get you?

## All levels of development



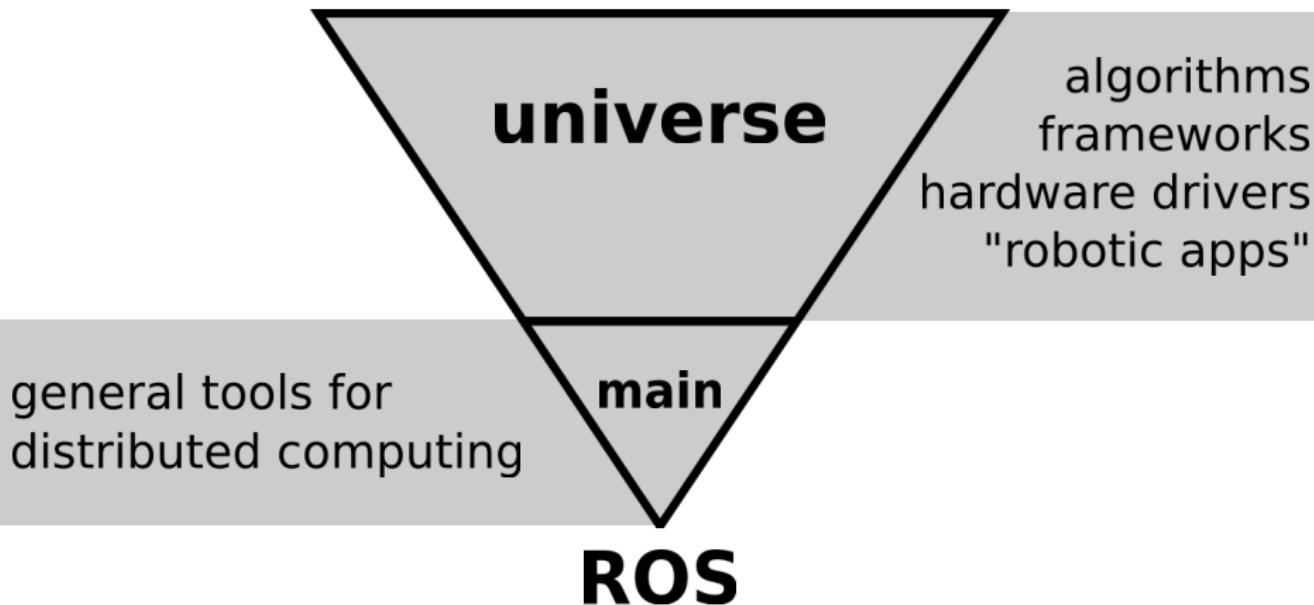
# What does ROS get you?

## All levels of development



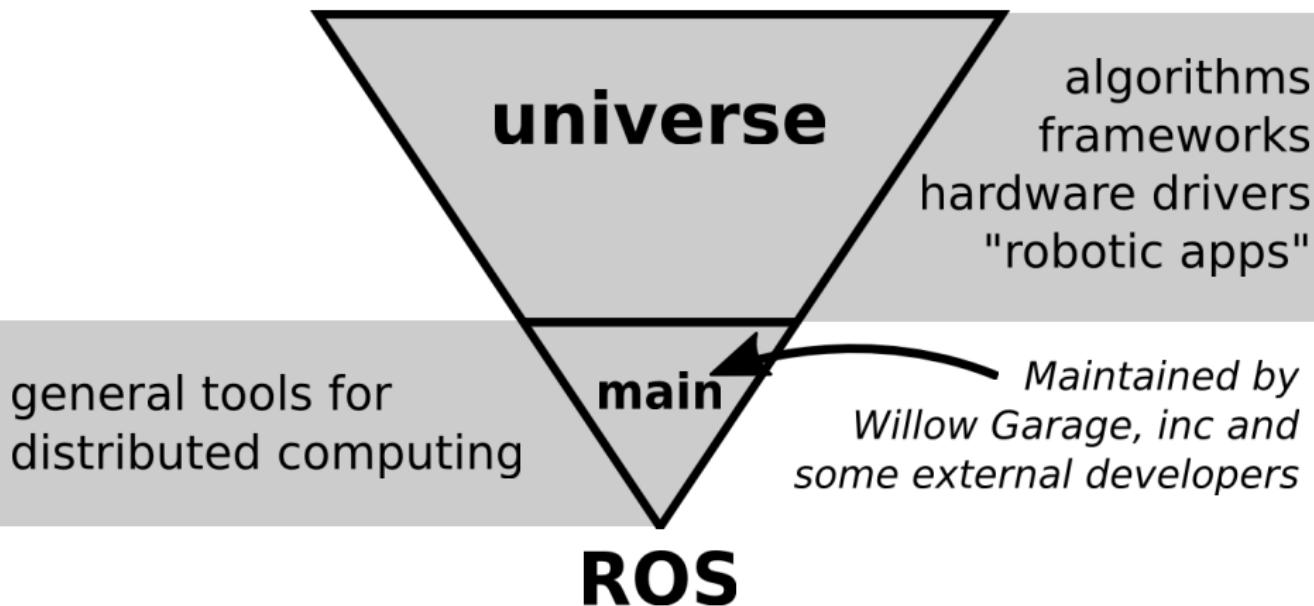
# What does ROS get you?

## All levels of development



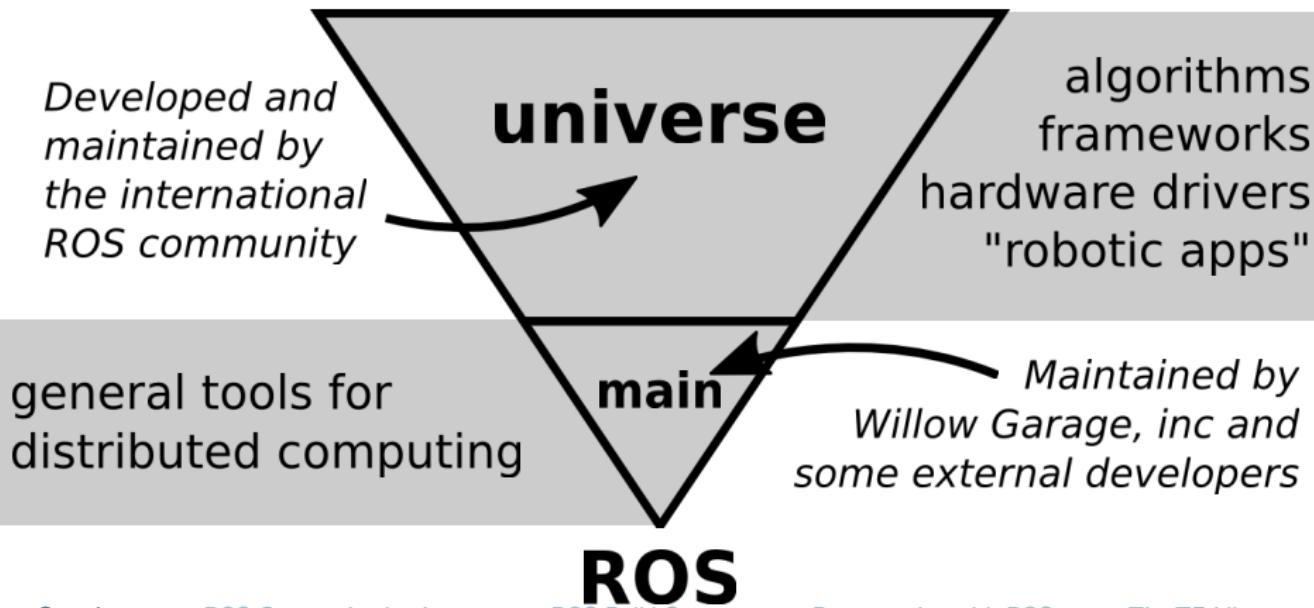
# What does ROS get you?

## All levels of development



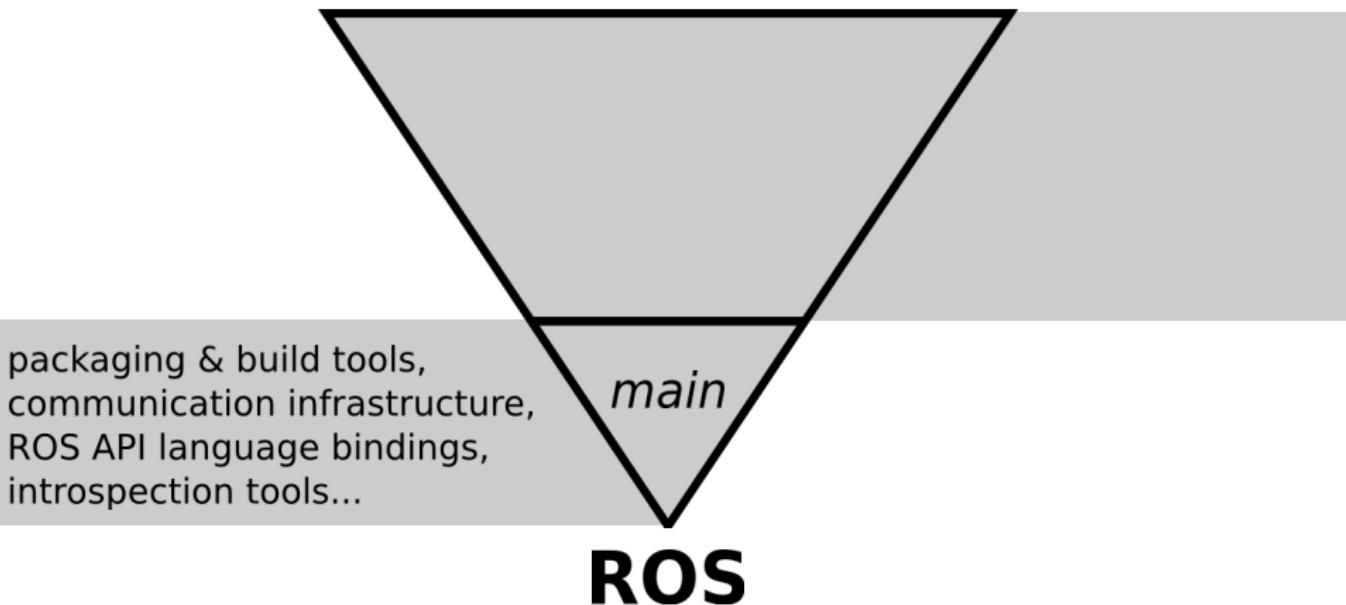
# What does ROS get you?

## All levels of development



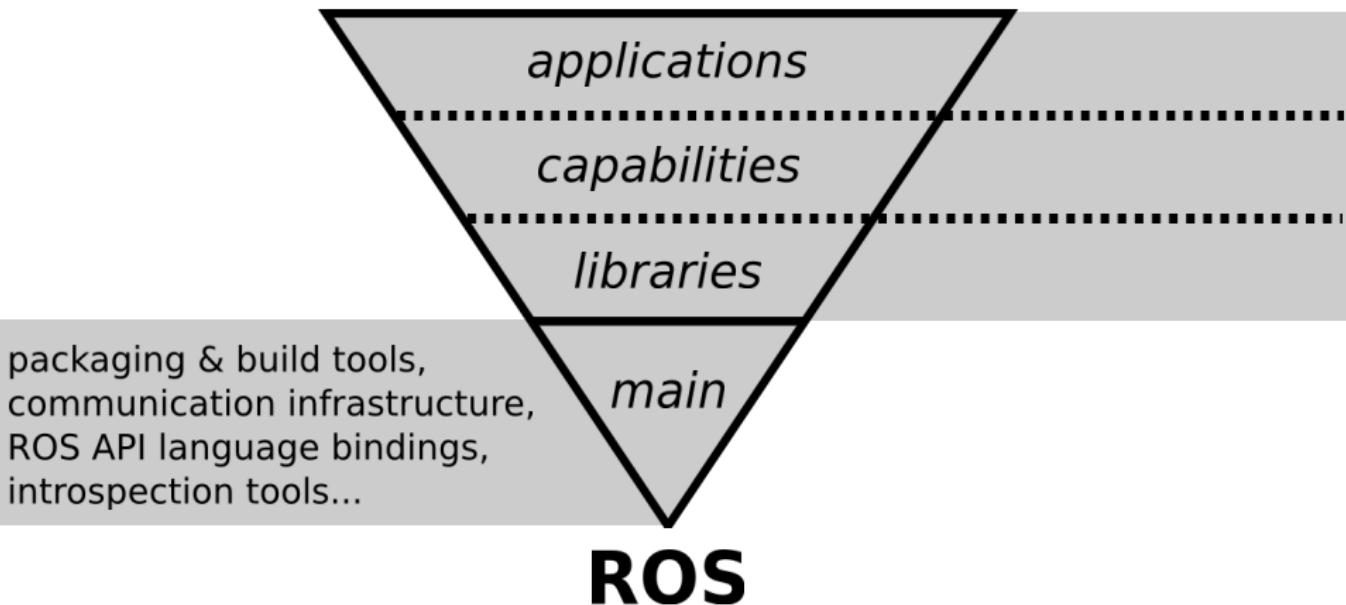
# What does ROS get you?

## All levels of development



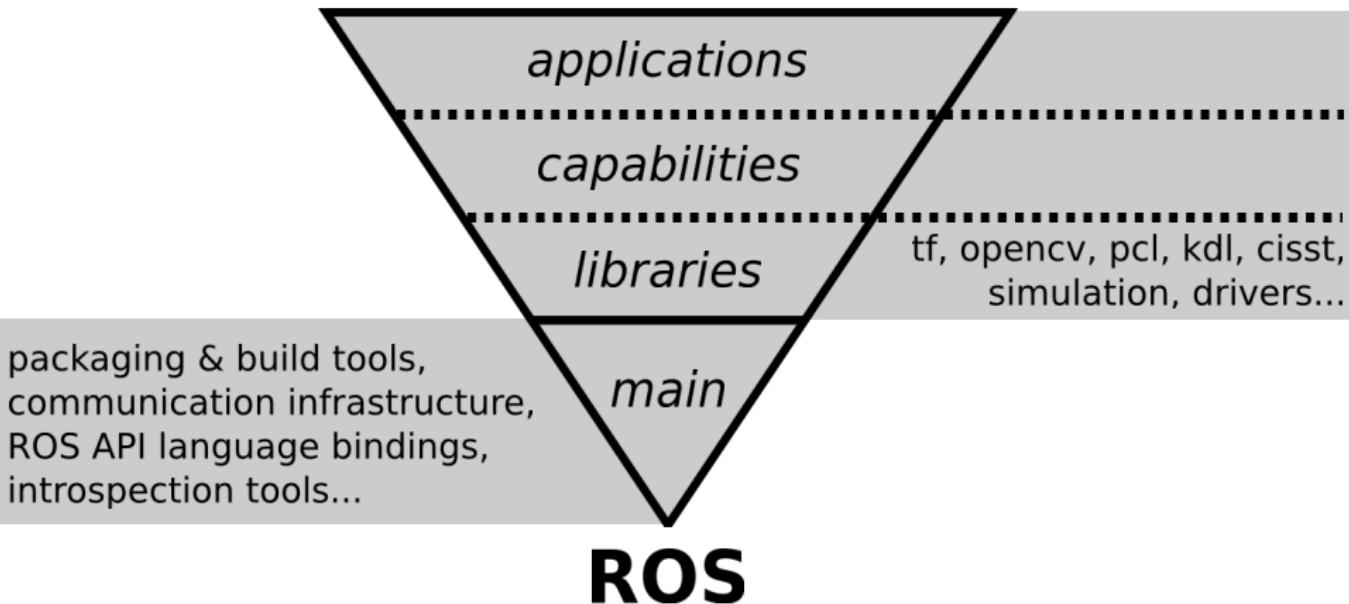
# What does ROS get you?

## All levels of development



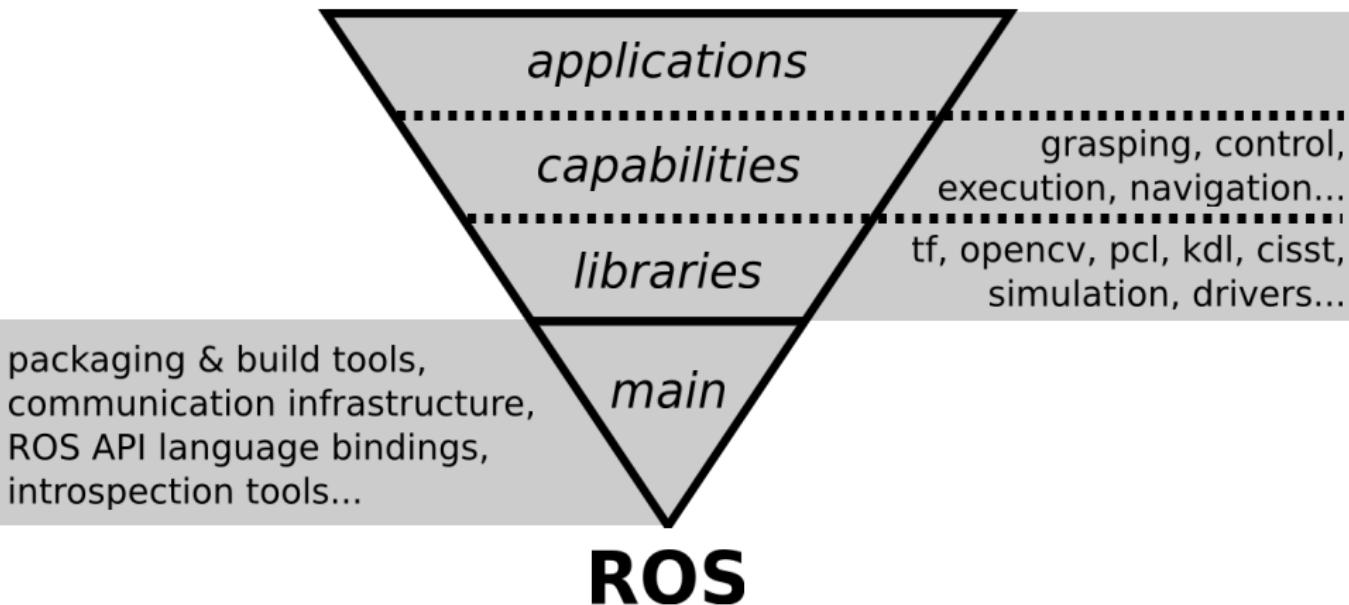
# What does ROS get you?

## All levels of development



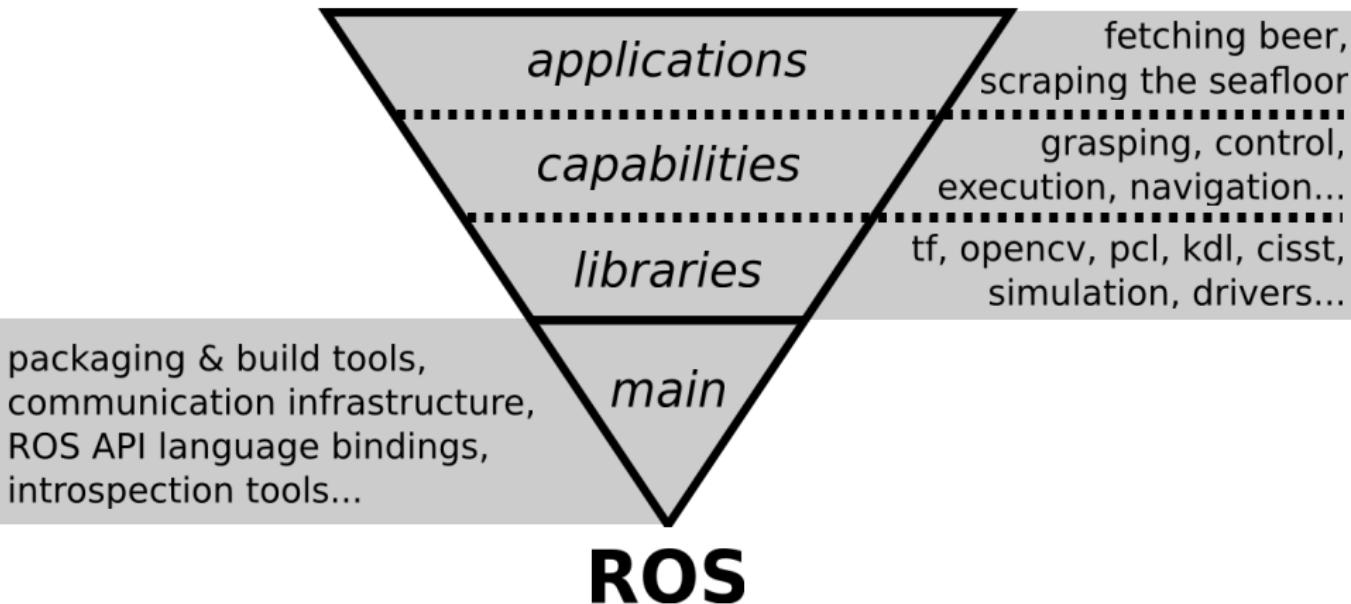
# What does ROS get you?

## All levels of development



# What does ROS get you?

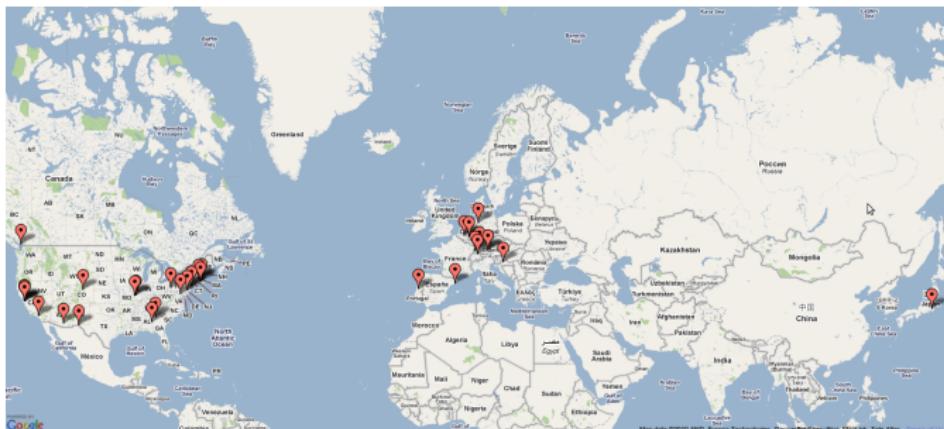
## All levels of development



# ROS

# The ROS Community

## Researchers using common tools to enable collaboration



79 Institutional ROS Repositories, all over the world (July, 2011)



# www.ros.org - The ROS Hub

## A centralized location for ROS users and developers

ROS.org [About](#) | [Support](#) | [answers.ros.org](#)

[Documentation](#) [Browse Software](#) [News](#) [Download](#)

**Documentation**

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

ROS:

- [Install](#)  
Install ROS on your machine.
- [Getting Started](#)  
Tutorials, technical overview, and links to [getting help](#). Also, check out the [ROS cheat sheet.pdf](#)
- [Contribute](#)  
How to contribute to the ROS community, such as submitting your own [repository](#).

Software:

- [Core Libraries](#)  
APIs by language and topic.
- [Common Tools](#)  
Common tools for developing and debugging ROS software.
- [Search Software](#)  
Search the 2000+ libraries available for ROS.

Robots/Hardware:

- [Robots](#)  
Robots that you can use with ROS.
- [Sensors](#)  
Sensor drivers for ROS.
- [Driver Tutorials](#)  
Tutorials for supported hardware.

Publications, Courses, and Events:

- [Papers](#)  
Published papers with open source implementations available.
- [Courses](#)  
Courses using or teaching ROS.
- [Events](#)  
Past events and materials based on ROS.

Except where otherwise noted, the ROS wiki is licensed under [Creative Commons Attribution 3.0](#)

Wiki: Documentation (last edited 2011-06-17 23:49:49 by [MeloneeWong](#))

**Wiki**

- [ROS](#)
- [StaticList](#)
- [RecentChanges](#)
- [WatchNamespaces](#)
- [Documentation](#)

**Page**

- [Edit \(Text\)](#)
- [Info](#)
- [Discussions](#)
- [Add Link](#)
- [Attachments](#)

[More Actions: ▾](#)

**User**

- [JonathanBiggs](#)
- [Settings](#)
- [Logout](#)



# answers.ros.org - ROS Questions & Answers

## Community-supported help for ROS users

ROS  
ANSWERS questions tags people badges ask a question search

all unanswered favorites Sort by: date activity answers votes

1,370 questions Search by tag and query to focus your search.

Using a python node in parallel mode [python | Python | users]

How much horsepower is needed to run the kinect stack [kinect | kinect\_stack | votes]

How to set JAVA\_HOME in launch file? [research | next\_steps | users]

add a new frame to the tree [frame | g]

upgrade to 11.04, or wait? [Ubuntu\_11.04 | Ubuntu]

Planner for non-holonomic/differential constraints [mp | planning | openni\_planning | path\_planning | motion\_planners]

error installing openni\_kinect [openni | user]

iterate points for OpenCV's Line function [openni | cv\_line | users]

tabletop segmentation fails (eigen error) [tabletop | segmentation | eigen]

Error installing ROS on MacBook Pro OS X 10.6.7 [mac | rosnoob | ROS | user | install | rosinstall]

installing turtlebot software on beagleboard. [ARM | turtles | BeagleBoard | FASTLAM | ROS]

velodyne rawscan mdns issue mismatch [rawscan | velodyne | user | votes]

depth wrong using openni\_camera depth image [Kinect | openni | openni\_camera]

turtlebot with gyro but no kinect [turtlebot | gyro]

Contributors

Interesting tags

Add Ignored tags

Keep unanswered questions hidden



# ros mailing lists

## Getting in touch with the developer community

- ROS Users - *for general ROS-related discussions*  
<https://code.ros.org/mailman/listinfo/ros-users>
- Other Lists & List Archives  
<http://code.ros.org/lurker>



# Outline

Overview

ROS Communication Layer

ROS Build System

Programming with ROS

The TF Library

# ROS Core

## Where it all comes together

- *ROS Master*
  - A centralized XML-RPC server
  - Negotiates communication connections
  - Registers and looks up names for ROS graph resources
- *Parameter Server*  
Stores persistent configuration parameters and other arbitrary data
- *rosout*  
Essentially a network-based `stdout` for human-readable messages

# ROS “Graph” Abstraction

## Named network resources

ROS graph resources:

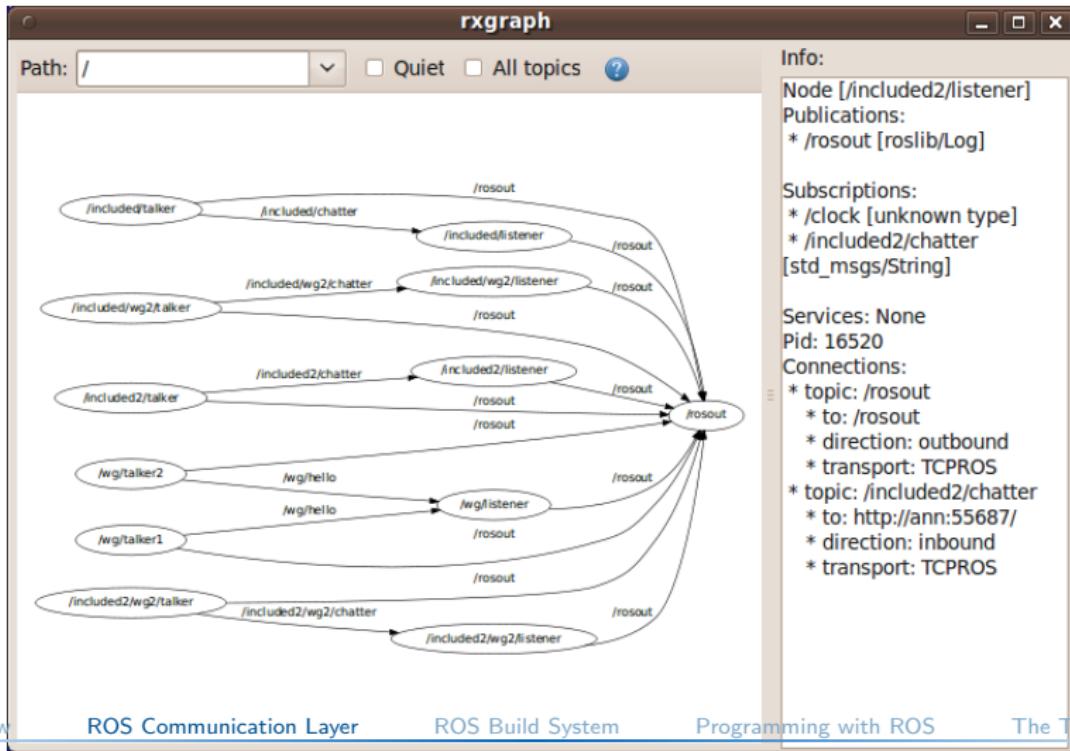
- *nodes*
  - processes
  - produce and consume data
- *parameters*
  - persistent data storage
  - configuration, initialization settings
  - stored on parameter server
- *topics*

Asynchronous many-to-many communication streams.
- *services*

Synchronous one-to-many network-based functions.

# ROS “Graph”

## rxgraph: communication network visualization



# Creating and Running ROS Nodes

## Distributing computation with ROS

### Launch files

- XML files for launching nodes
- associate a set of parameters and nodes with a single file
- hierarchically compose collections of other launch files
- automatically re-spawn nodes if they crash
- change node names, namespaces, topics, and other resource names *without* recompiling
- easily distribute nodes across multiple machines

# Example Launch File

## Example *launch file*

```
<launch>
  <node name="my_node" pkg="foo" type="bar">
    <remap from="/base_laser/scan" to="scan" />
    <rosparam>
      use_foo: True
      frame_id: base_laser
    </rosparam>
  </node>
</launch>
```

- Launch with `roslaunch package foo.launch`

# ROS Communication Protocols

## Connecting nodes over the network

- *ROS Topics*
  - Asynchronous “stream-like” communication
  - Strongly-typed (ROS .msg spec)
  - Can have one or more *publishers*
  - Can have one or more *subscribers*
- *ROS Services*
  - Synchronous “function-call-like” communication
  - Strongly-typed (ROS .srv spec)
  - Can have only one *server*
  - Can have one or more *clients*
- *Actions*
  - Built on top of topics
  - Long running processes
  - Cancellation

# Asynchronous Distributed Communication

## ROS TCP Topics



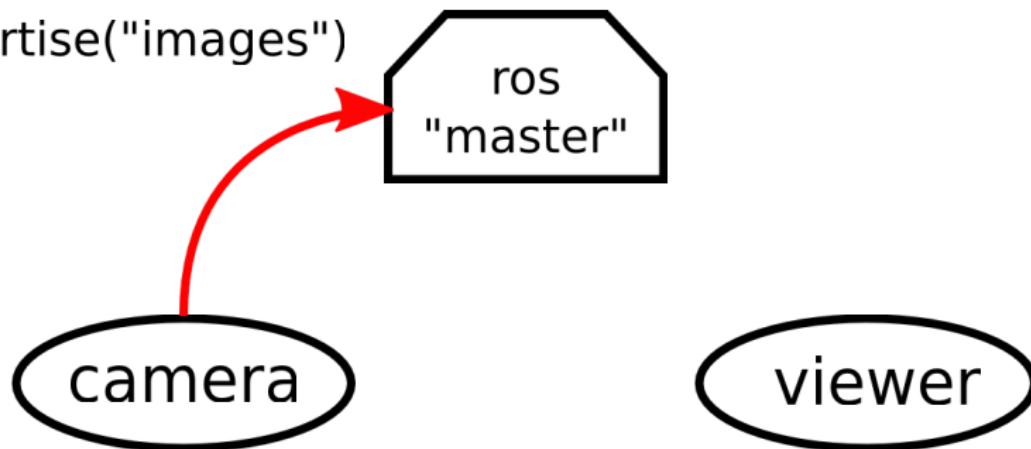
camera

viewer

# Asynchronous Distributed Communication

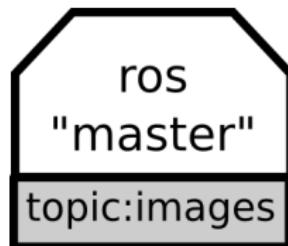
## ROS TCP Topics

advertise("images")



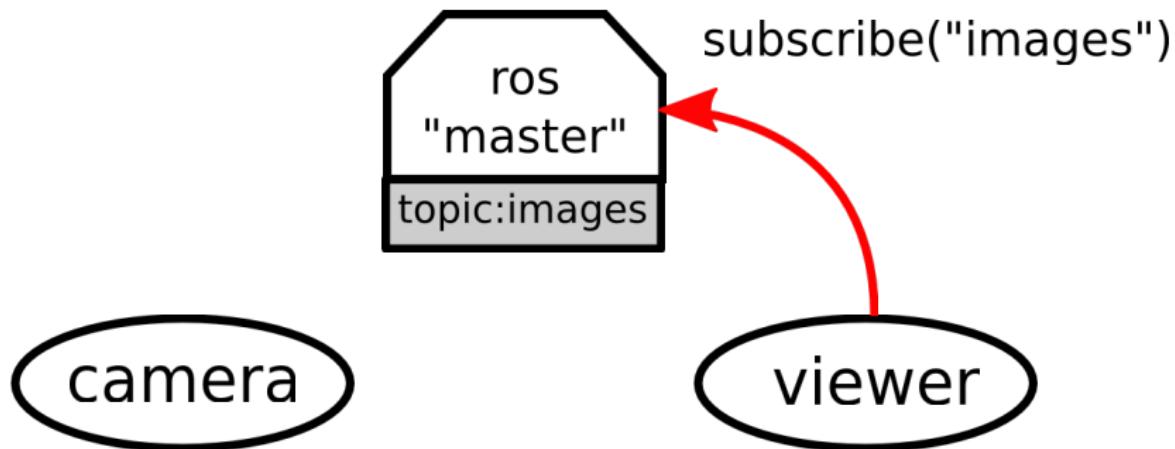
# Asynchronous Distributed Communication

## ROS TCP Topics



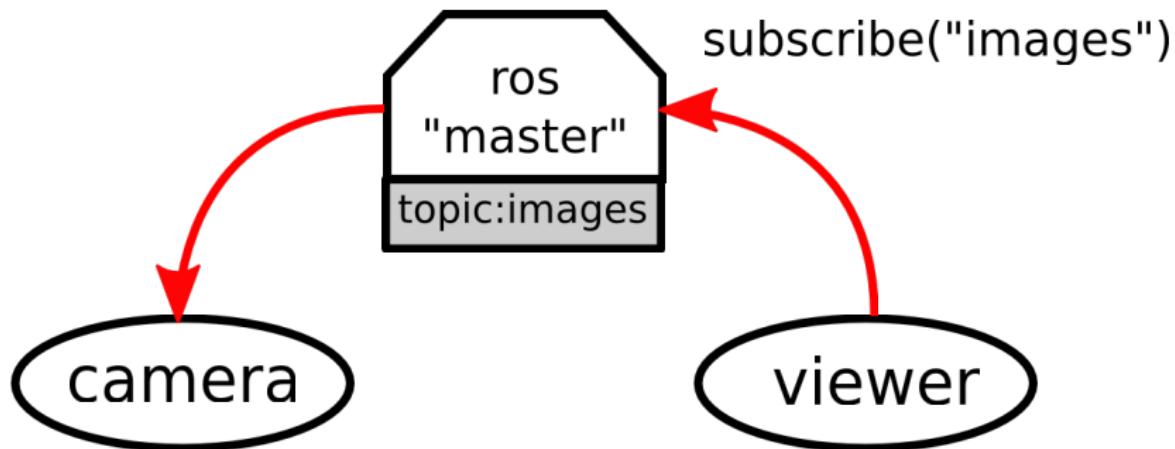
# Asynchronous Distributed Communication

## ROS TCP Topics



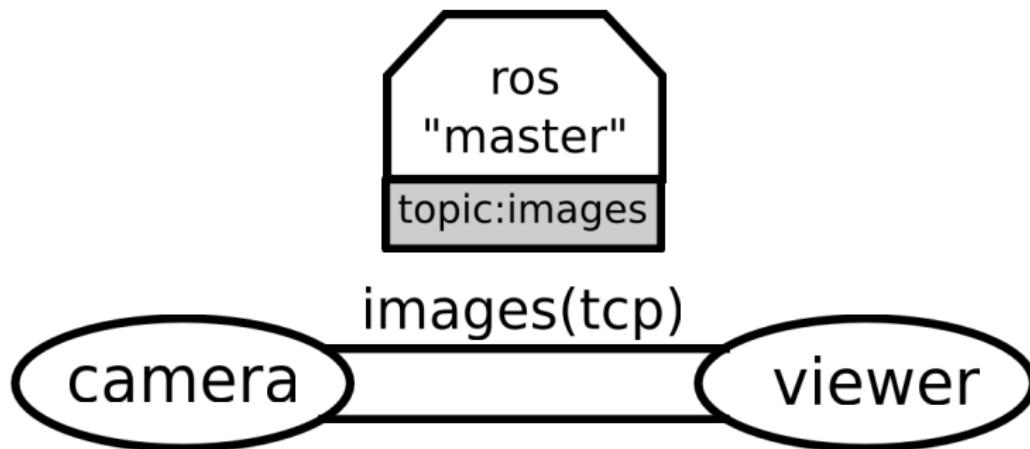
# Asynchronous Distributed Communication

## ROS TCP Topics



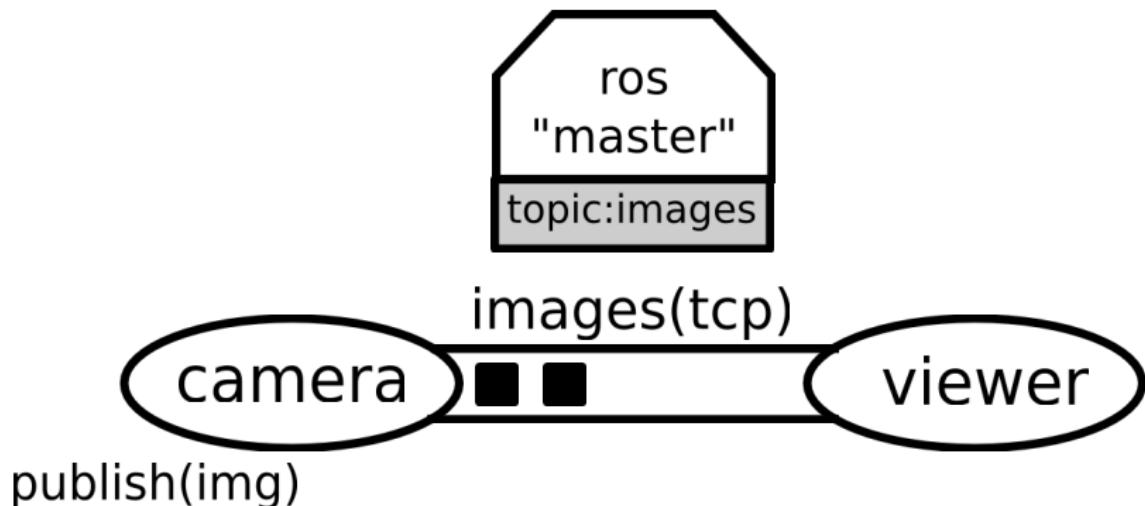
# Asynchronous Distributed Communication

## ROS TCP Topics



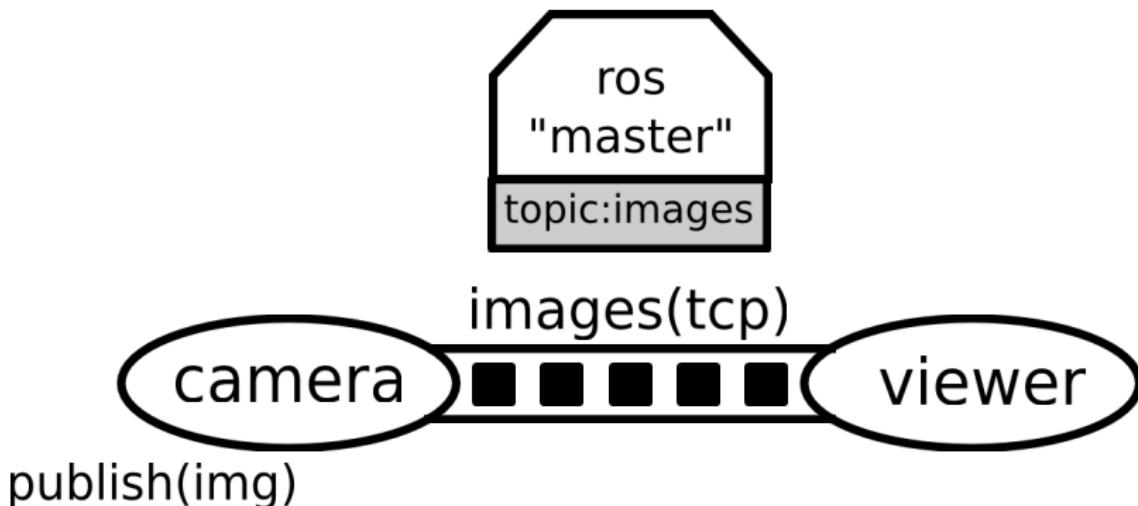
# Asynchronous Distributed Communication

## ROS TCP Topics



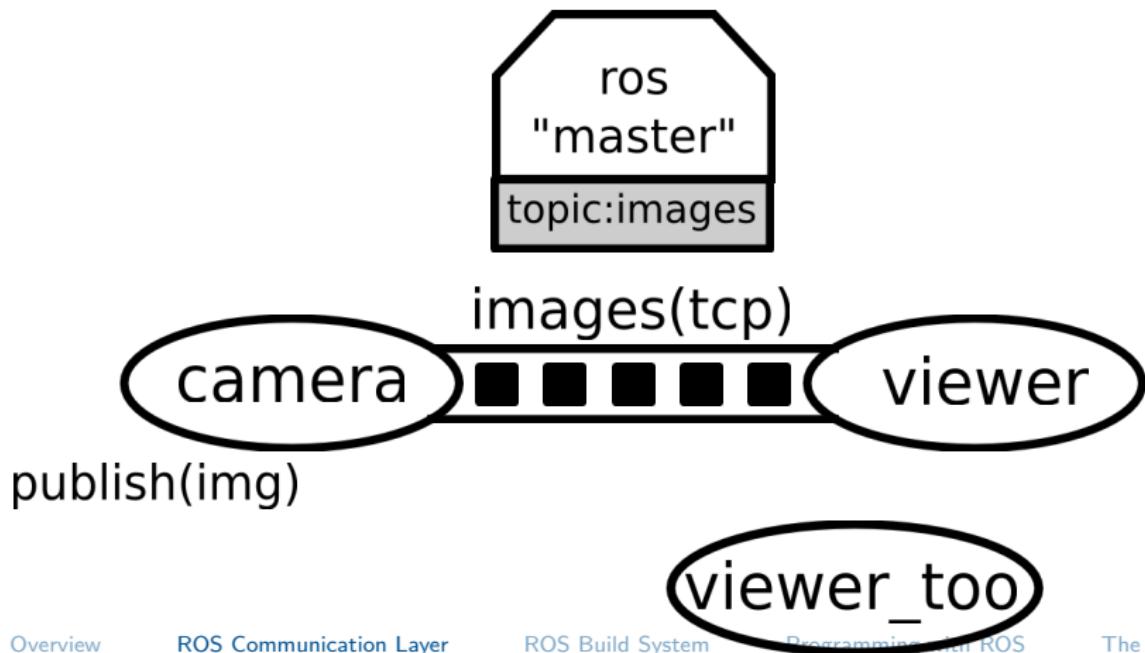
# Asynchronous Distributed Communication

## ROS TCP Topics



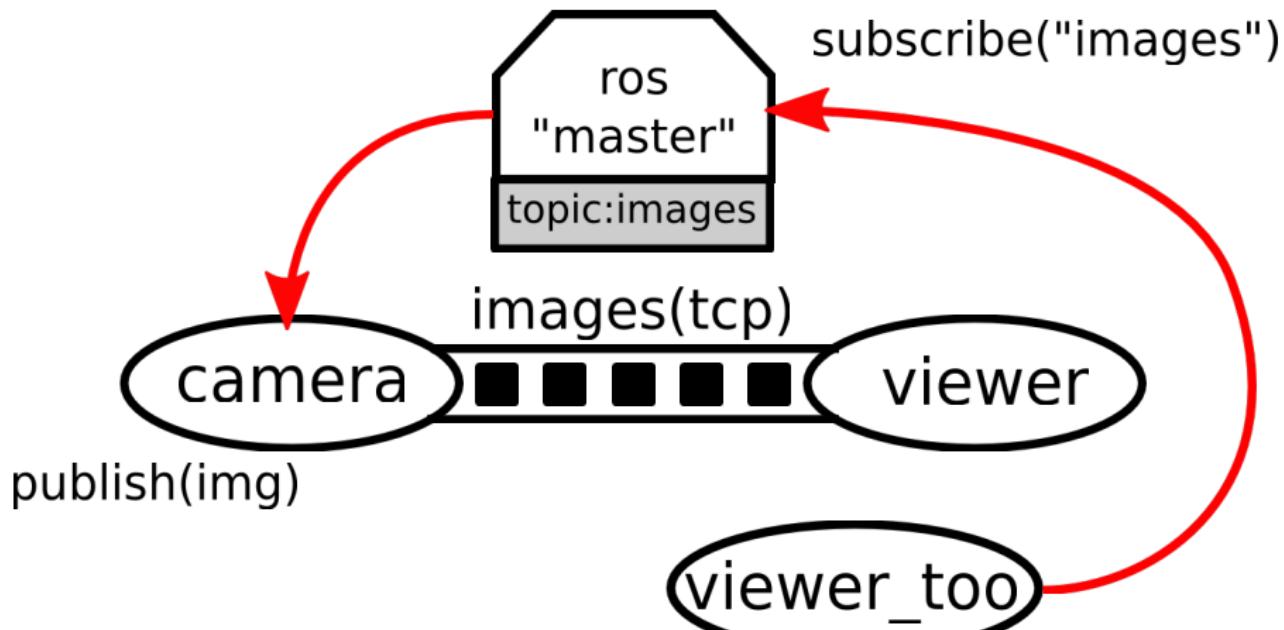
# Asynchronous Distributed Communication

## ROS TCP Topics



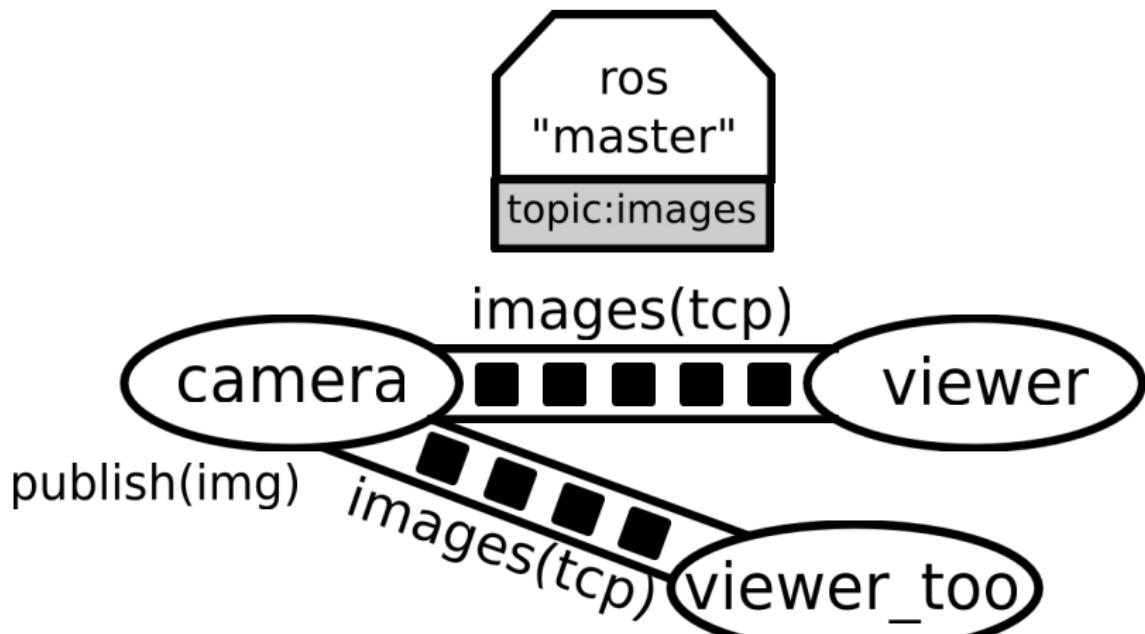
# Asynchronous Distributed Communication

## ROS TCP Topics



# Asynchronous Distributed Communication

## ROS TCP Topics



# Debugging rosout

ROS provides mechanisms in all languages for specifying different *levels* of human-readable log messages.

The five default levels are:

- fatal
- error
- warn
- info
- debug

Corresponding logging commands (C++):

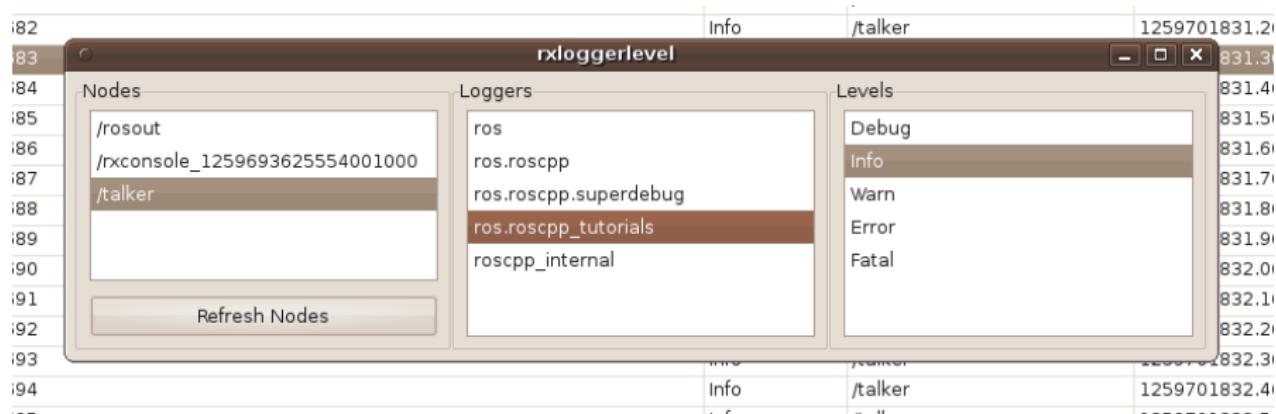
- `ROS_FATAL(...)`
- `ROS_ERROR(...)`
- `ROS_WARN(...)`
- `ROS_INFO(...)`
- `ROS_DEBUG(...)`

# Debugging rxconsole

Message	Severity	Node	Time
hello world 78866	Info	/talker	1259701549.969195000
hello world 78867	Info	/talker	1259701550.069209000
hello world 78868	Info	/talker	1259701550.169192000
hello world 78872	Info	/talker	1259701550.269198000
hello world 78873	Info	/talker	1259701550.369194000
hello world 78874	Info	/talker	1259701550.469195000
hello world 78875	Info	/talker	1259701550.569196000
hello world 78876	Info	/talker	1259701550.669191000
hello world 78877	Info	/talker	1259701550.769193000
hello world 78878	Info	/talker	1259701550.869224000
hello world 78879	Info	/talker	1259701550.969351000
hello world 78880	Info	/talker	1259701551.069208000
hello world 78881	Info	/talker	1259701551.169190000
hello world 78882	Info	/talker	1259701551.269193000
hello world 78883	Info	/talker	1259701551.369193000
hello world 78884	Info	/talker	1259701551.469194000
hello world 78885	Info	/talker	1259701551.569194000
hello world 78886	Info	/talker	1259701551.669190000
hello world 78887	Info	/talker	1259701551.769207000
hello world 78888	Info	/talker	1259701551.869196000
hello world 78889	Info	/talker	1259701551.969193000
hello world 78890	Info	/talker	1259701552.069209000
hello world 78891	Info	/talker	1259701552.169190000
hello world 78892	Info	/talker	1259701552.269193000
hello world 78893	Info	/talker	1259701552.369192000

# Debugging rxconsole

# Debugging rxconsole



# ROS Graph Introspection

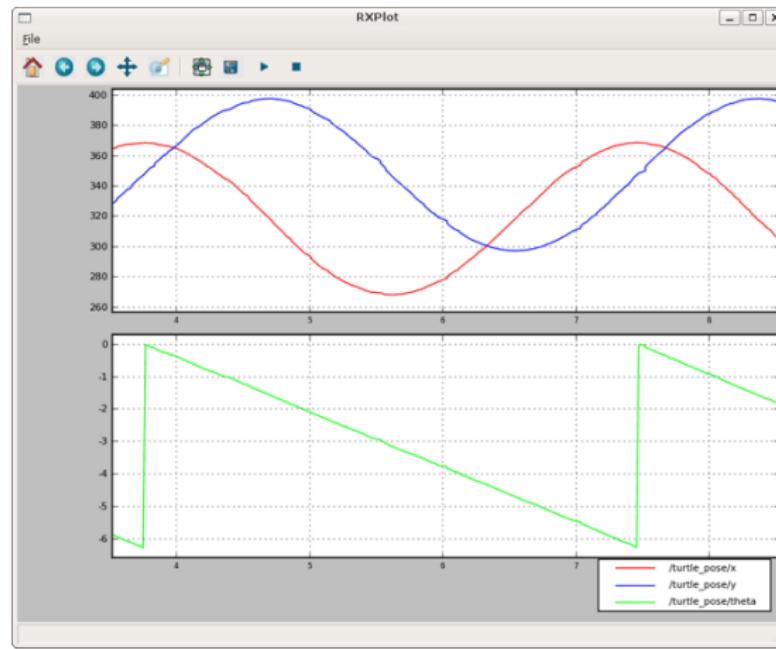
## No more wireshark

ROS provides several tools for analyzing the data flowing over ROS communication resources:

- *rosnodes*  
Gives a user information about a node: publications, subscriptions, etc
- *rostopic*  
Gives datarate, actual data, publishers, subscribers
- *rosservice*  
Enables a user to call a ROS Service from the command line
- *rosrun* (wire trouble finder)  
Diagnoses problems with a ROS network

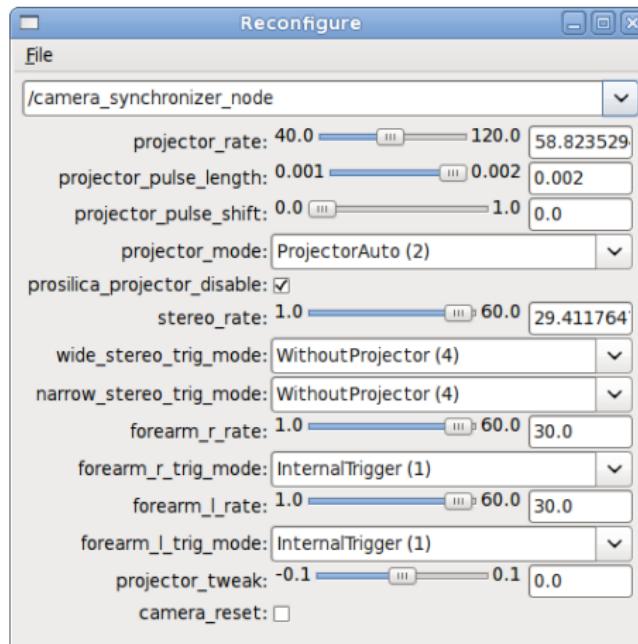
# ROS GUI Tools

There are lots...



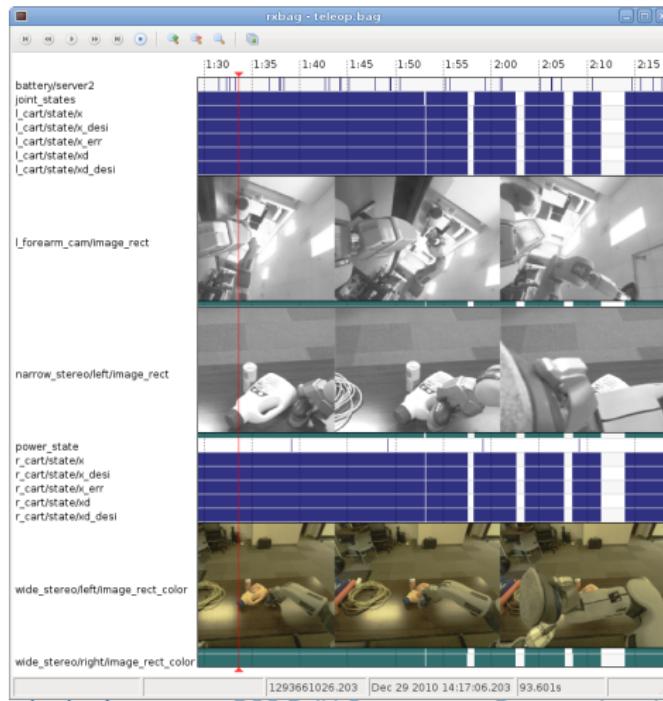
# ROS GUI Tools

There are lots...



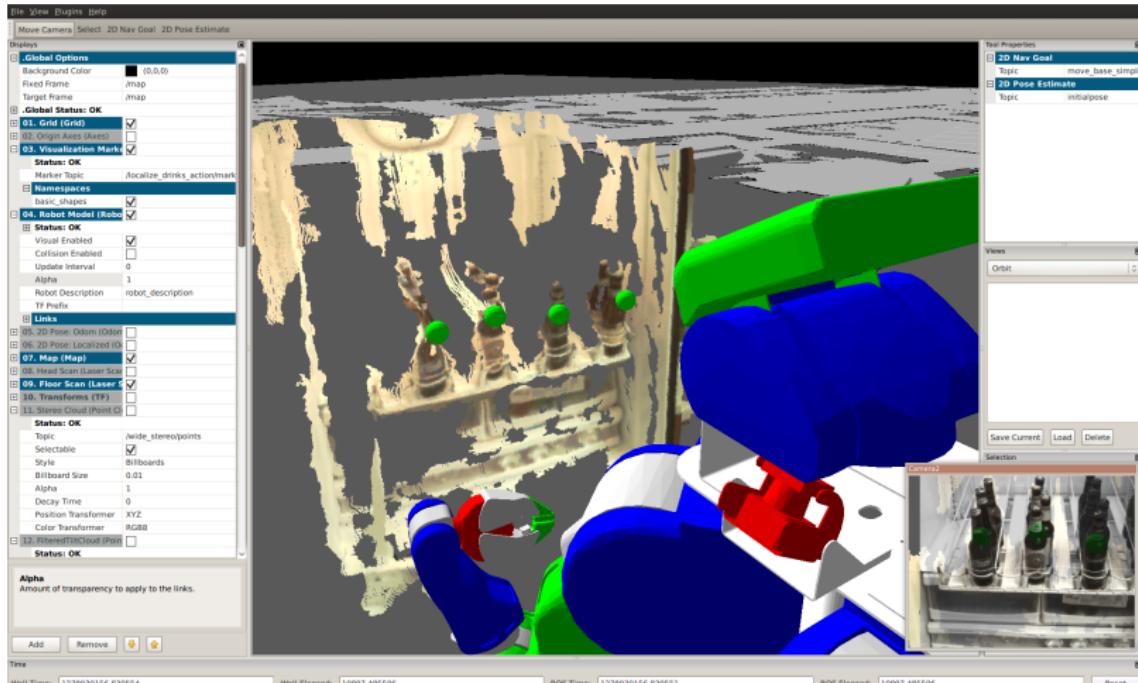
# ROS GUI Tools

## There are lots...



# rviz - 3D Visualization

## Modular state and sensor visualization





# Outline

Overview

ROS Communication Layer

ROS Build System

Programming with ROS

The TF Library



# ROS Stacks & Packages

## How to organize code in a ROS ecosystem

ROS code is grouped at two different levels:

- *Packages*  
A named collection of software that is built and treated as an atomic dependency in the ROS build system.
- *Stacks*  
A named collection of packages for distribution.



# ROS Distributions

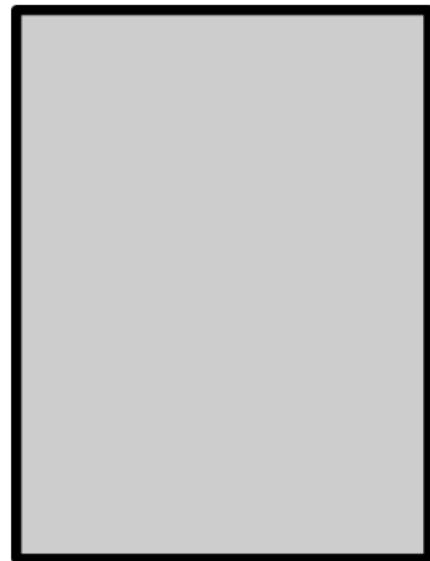
## Delivering ROS packages to the masses

- source code
- header declarations
- scripts
- message definitions
- service definitions
- configuration files
- launch files
- metadata
- ...

# ROS Distributions

## Delivering ROS packages to the masses

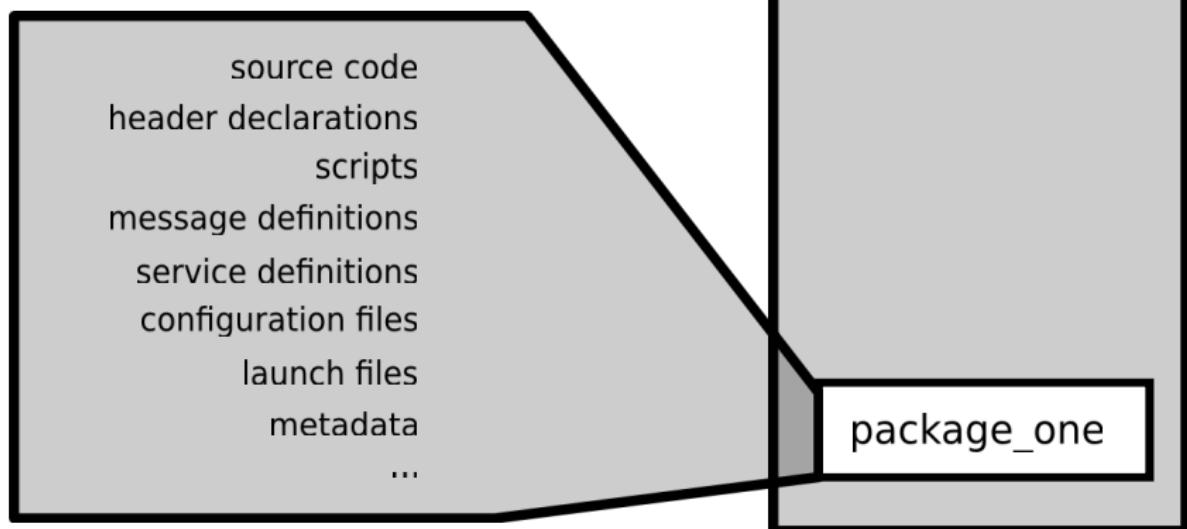
source code  
header declarations  
scripts  
message definitions  
service definitions  
configuration files  
launch files  
metadata  
...



**"stack"**

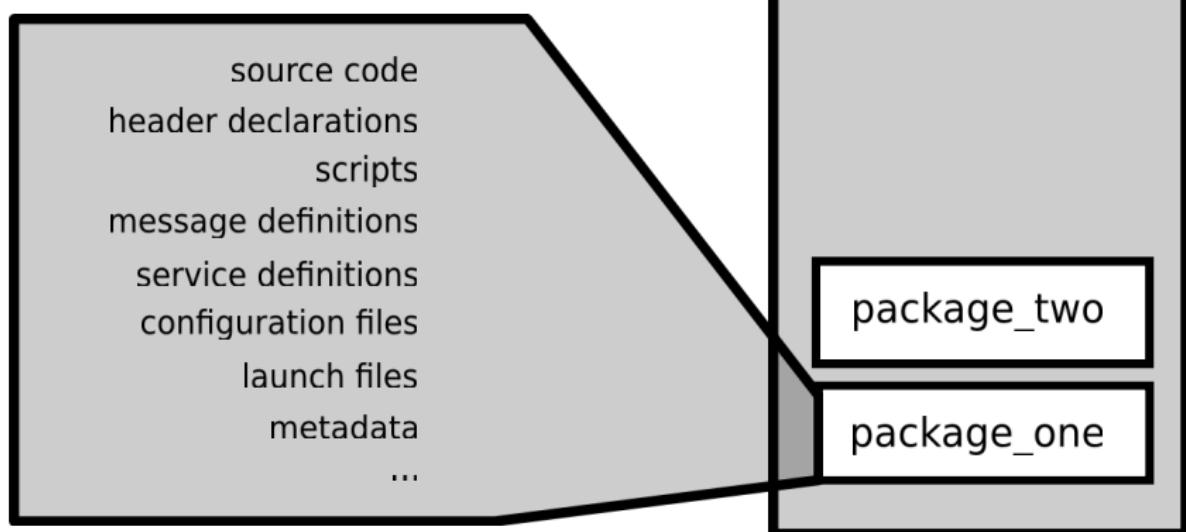
# ROS Distributions

Delivering ROS packages to the masses



# ROS Distributions

## Delivering ROS packages to the masses



**"package"**

Overview

ROS Communication Layer

Lorenz Mösenlechner

**"stack"**

Programming with ROS

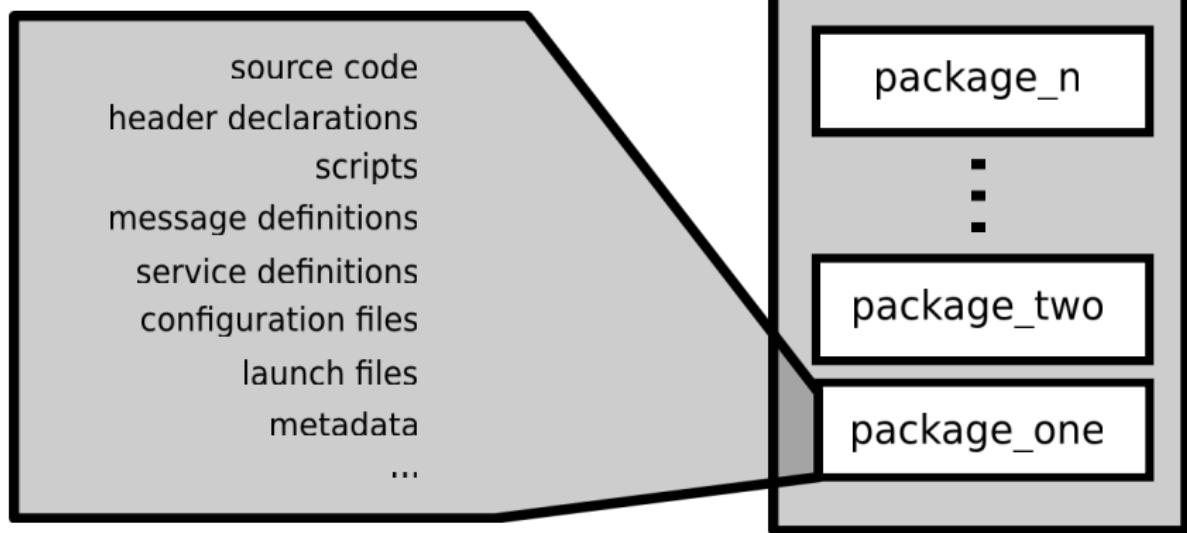
The TF Library

ROS Build System

Introduction to ROS

# ROS Distributions

Delivering ROS packages to the masses



**"package"**

Overview

ROS Communication Layer

ROS Build System

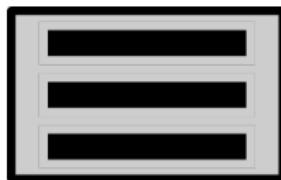
**"stack"**

Programming with ROS

The TF Library

# ROS Distributions

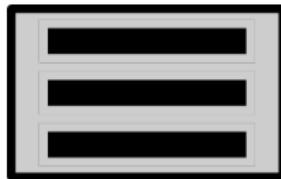
## Delivering ROS packages to the masses



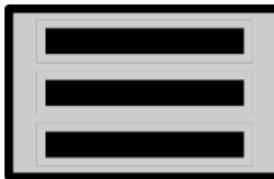
stack\_a-0.4.0

# ROS Distributions

## Delivering ROS packages to the masses



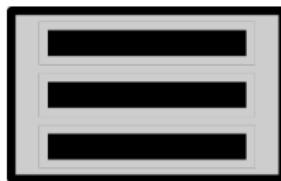
stack\_a-0.4.0



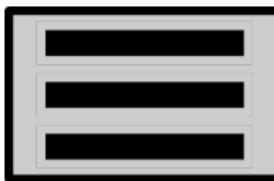
stack\_b-1.0.2

# ROS Distributions

## Delivering ROS packages to the masses

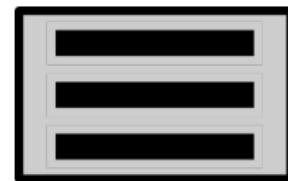


stack\_a-0.4.0



stack\_b-1.0.2

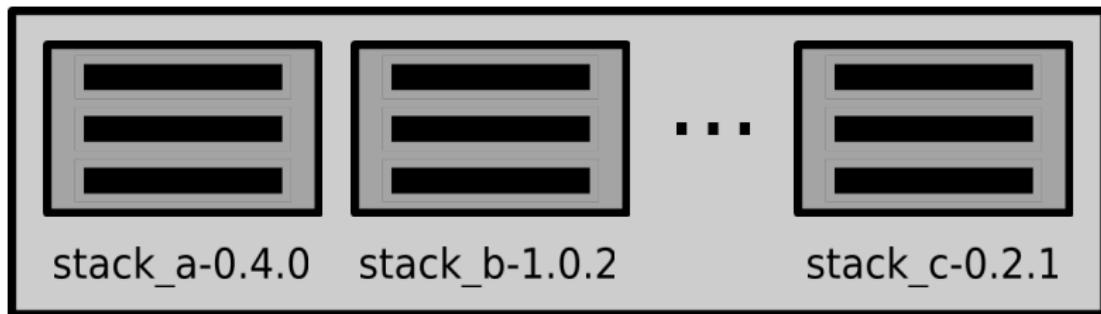
...



stack\_c-0.2.1

# ROS Distributions

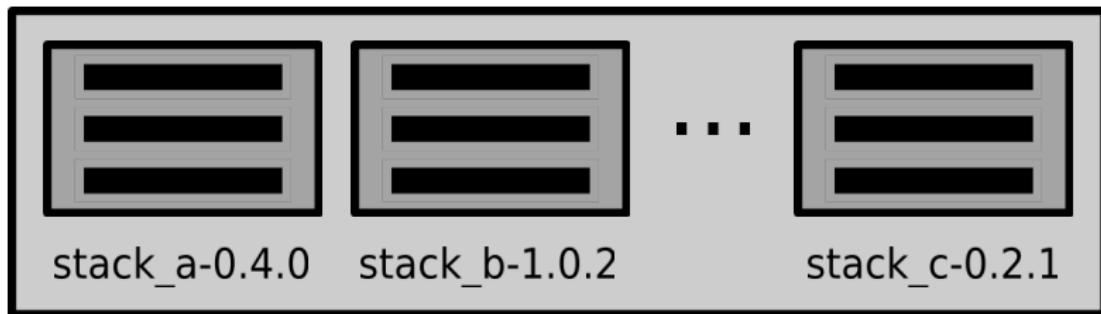
Delivering ROS packages to the masses



**"distribution"**

# ROS Distributions

## Delivering ROS packages to the masses



### "distribution"

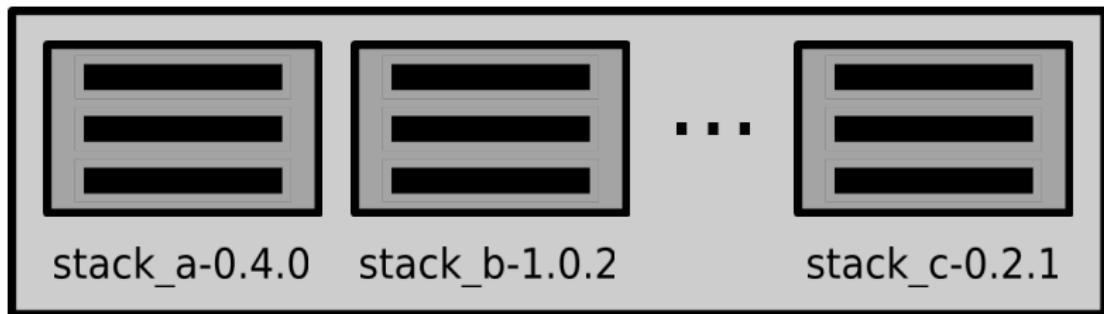
*ROS "Box Turtle"  
March 2, 2010*

*ROS "C-Turtle"  
August 2, 2010*

*ROS "Diamondback"  
March 2, 2011*

# ROS Distributions

## Delivering ROS packages to the masses



### "distribution"

*ROS "Box Turtle"  
March 2, 2010*

*ROS "C-Turtle"  
August 2, 2010*

*ROS "Diamondback"  
March 2, 2011*

**Future:** *ROS "Elecrtic Emys"  
August, 2011*

# ROS Meta-Filesystem

## Increasing codebase flexibility

The minimal representation of a ROS package is a directory in the \$ROS\_PACKAGE\_PATH which contains a single file:

- *manifest.xml*
  - Contains package metadata (author, license, url, etc)
  - Specifies *system* and *package* dependencies
  - Specifies language-specific *export* flags
- *CMakeLists.txt*: contains ROS build rules (executables, libraries, custom build flags, etc)
- *Makefile*: just a proxy to build *this* package
- Create package with *roscreate-pkg*

```
$ roscreate-pkg foo roscpp std_msgs
```
- Build package with *rosmake*

```
$ rosmake foo
```

# ROS Packages

## The *manifest.xml* file

```
<package>
  <description brief="foo">
    foo
  </description>
  <author>Lorenz Moesenlechner</author>
  <license>BSD</license>
  <review status="unreviewed" notes="" />
  <url>http://ros.org/wiki/foo</url>
  <depend package="roscpp" />
  <depend package="std_msgs" />
</package>
```

# ROS Packages

## The *CMakeLists.txt* file

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

rosbuild_init()

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

rosbuild_genmsg()
rosbuild_gensrv()

rosbuild_add_library(foo src/foo.cpp)
rosbuild_add_executable(hello_world src/hello_world.cpp)
```



# Outline

Overview

ROS Communication Layer

ROS Build System

Programming with ROS

The TF Library

# ROSCPP

- Initialization with `ros::init`:
  - register at core
  - set up remappings
  - set up networking
- `ros::NodeHandle` as interface to topics, services and parameters
- `ros::NodeHandle::subscribe`, `ros::NodeHandle::advertise` for topics
- `ros::spin` and `ros::spinOnce` to process ROS messages
- Use `boost::bind` to use member functions as callbacks:

```
boost::bind( Listener::laserCb , this , _1 );
```



# Messages

- defined in *package-name/msg/\*.msg* files, sent over topics
- basic data types:
  - *int{8,16,32,64}*
  - *float{32,64}*
  - *string*
  - *time*
  - *duration*
  - *array[]*
- Example: *Point.msg*

```
float64 x
float64 y
float64 z
```



# Simple Publisher (C++)

```
int main(int argc, char *argv[]) {
    ros::init(argc, argv, "talker");
    ros::NodeHandle nh;

    ros::Publisher pub =
        nh.advertise<std_msgs::String>("chatter", 1000);
    ros::Rate loop_rate(10);

    while(ros::ok()) {
        std_msgs::String msg;
        msg.data = "hello_world";
        pub.publish(msg)
        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

# Simple Subscriber (C++)

```
void msgCallback(const std_msgs::String::ConstPtr &msg) {  
    ROS_INFO("Message: %s", msg->data.c_str());  
}  
  
int main(int argc, char *argv[]) {  
    ros::init(argc, argv, "talker");  
    ros::NodeHandle nh;  
  
    ros::Subscriber sub =  
        nh.subscribe<std_msgs::String>("chatter",  
        1000, msgCallback);  
  
    ros::spin();  
  
    return 0;  
}
```



# Services

- Defined in *package-name/srv/\*.srv*.
- Definition similar to message files, Request message + response message.
- Example: *beginner\_tutorials/AddTwoInts*

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

# Service Client

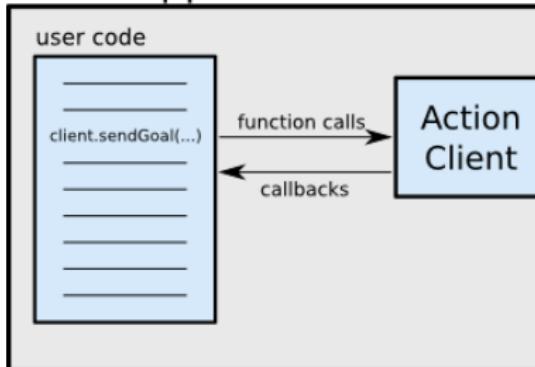
```
int main(int argc, char **argv)
{
    ros::init(argc, argv, "add_two_ints_client");
    ros::NodeHandle n;
    ros::ServiceClient client = n.serviceClient<AddTwoInts>(
        "add_two_ints");
    AddTwoInts srv;
    srv.request.a = 1;
    srv.request.b = 2;
    if (client.call(srv)) {
        ROS_INFO("Sum: %ld", (long int)srv.response.sum);
    } else {
        ROS_ERROR("Failed to call service add_two_ints");
        return 1;
    }
    return 0;
}
```

# Actions

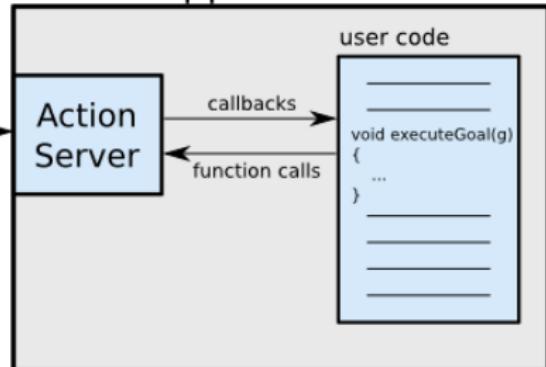
Using function calls and callbacks

- request goals (client side)
- execute goals (server side)

## Client Application



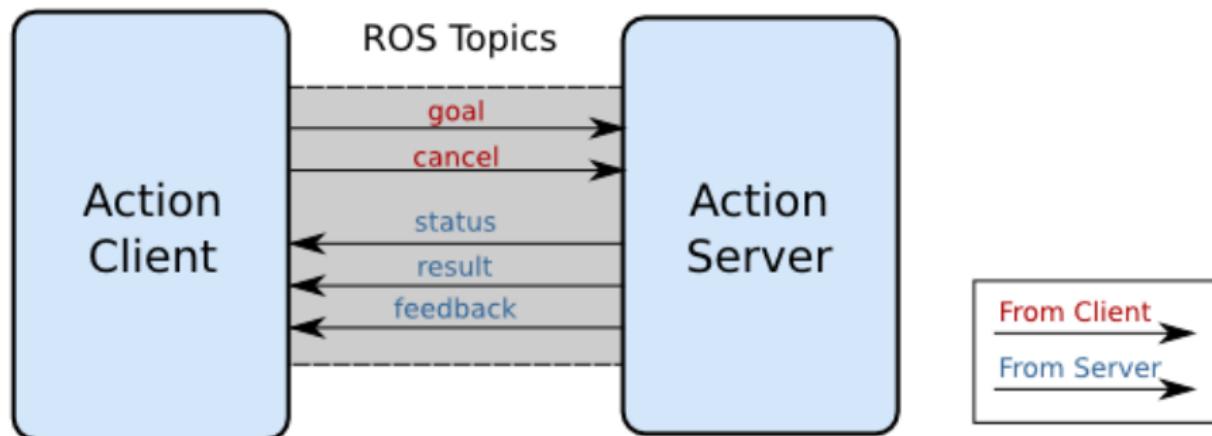
## Server Application



# Actions

- *action protocol* relies on ROS topics to transport messages

## Action Interface



# Action Definitions

- Similar to messages and services.
- Definition: Request + result + feedback
- Defined in *ros-package/action/\*.action*
- Generated by CMake macro *genaction()*.
- Example: *actionlib\_tutorials/Fibonacci.action*

```
#goal definition
int32 order
---

#result definition
int32[] sequence
---

#feedback
int32[] sequence
```

# Simple Action Client (C++)

```
int main (int argc, char **argv) {
    ros::init(argc, argv, "test_fibonacci");

    actionlib::SimpleActionClient<FibonacciAction> ac(
        "fibonacci", true);
    ac.waitForServer();
    learning_actionlib::FibonacciGoal goal;
    goal.order = 20;
    ac.sendGoal(goal);
    bool finished_before_timeout = ac.waitForResult(
        ros::Duration(30.0));

    return 0;
}
```



# Outline

Overview

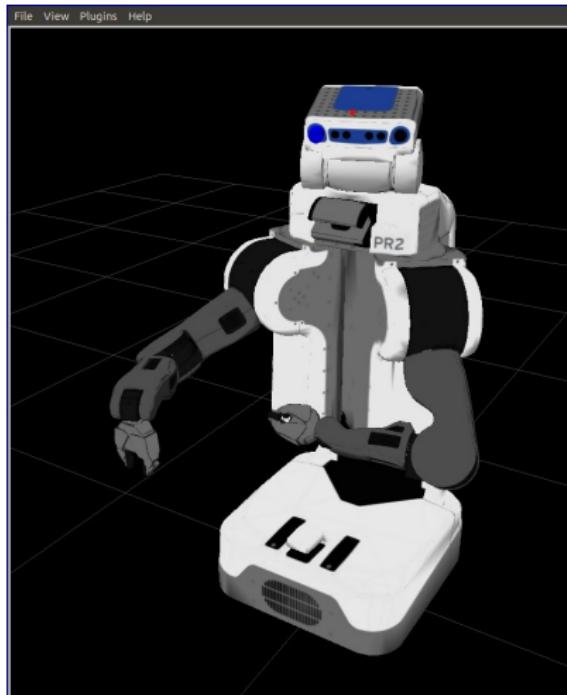
ROS Communication Layer

ROS Build System

Programming with ROS

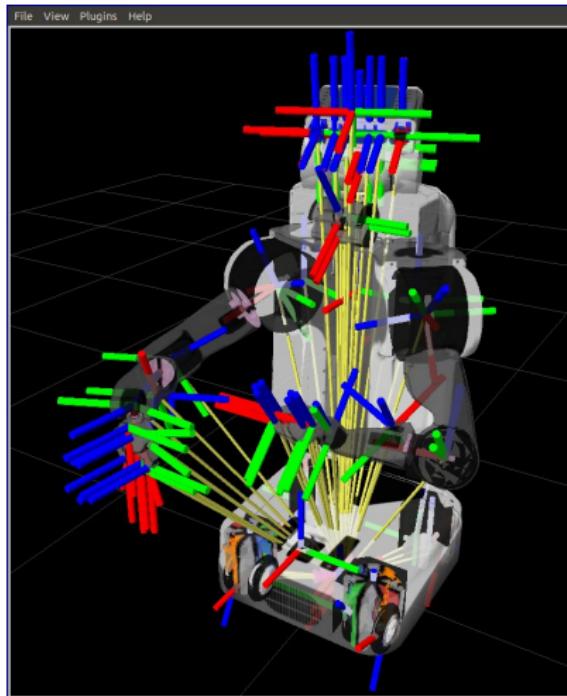
The TF Library

# Coordinate frames



- robots consist of many *links*
- every link describes its own *coordinate system*
- sensor measurements are local to the corresponding link
- links change their position over time

# Coordinate frames



- robots consist of many *links*
- every link describes its own *coordinate system*
- sensor measurements are local to the corresponding link
- links change their position over time



# Transforms are distributed

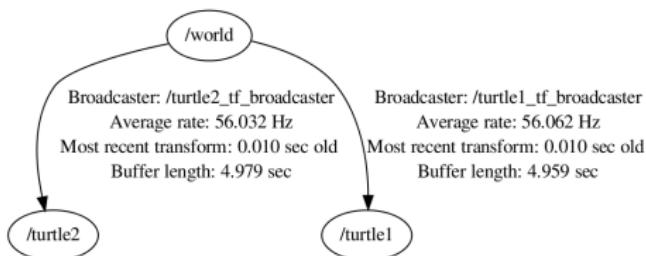
- Transforms are produced by different nodes:
  - Localization in map (AMCL, gmapping)
  - Odometry (base controller)
  - Joint positions (robot controllers and robot\_state\_publisher)
- Many publishers, many consumers
- Distributed system, redundancy issues, ...

# What is TF?

- decentralized: many publishers, many subscribers
- A coordinate frame tracking system
  - standardized protocol for publishing transforms
  - Classes and methods for looking up, calculating and sending transforms
  - transforms are published on the /tf topic
- No central instance managing the tree of transforms
- Command line tools

# The transform tree

view\_frames Result  
Recorded at time: 1254266629.492



- Consists of frames (links) and the transforms between them.
- Each link is cached (10 secs default caching time)
- Works with multiple disconnected trees
- Transforms must form a proper tree (no cycles)



# Utilities

- `rosrun tf tf_echo <source_frame> <target_frame>`
- `rosrun tf tf_monitor`
- `rosrun view_frames`
- `rosrun tf static_transform_publisher`
- `rviz`

# Utilities

- `rosrun tf tf_echo <source_frame> <target_frame>`
- `rosrun tf tf_monitor`
- `rosrun view_frames`
- `rosrun tf static_transform_publisher`
- `rviz`

## tf\_echo

```
$ rosrun tf tf_echo turtle1 turtle2

Success at 1253585684.557003974
[0.000000 0.000000 0.140754 0.990045] Euler(0.282446 -0.000000 0.000000)
Translation: [-0.000036 -0.000010 0.000000]
Success at 1253585685.544698953
[0.000000 0.000000 0.140754 0.990045] Euler(0.282446 -0.000000 0.000000)
Translation: [-0.000036 -0.000010 0.000000]
Success at 1253585686.557049989
```

# Utilities

- `rosrun tf tf_echo <source_frame> <target_frame>`
- `rosrun tf tf_monitor`
- `rosrun view_frames`
- `rosrun tf static_transform_publisher`
- `rviz`

## tf\_monitor

```
$ rosrun tf tf_monitor
RESULTS: for all Frames

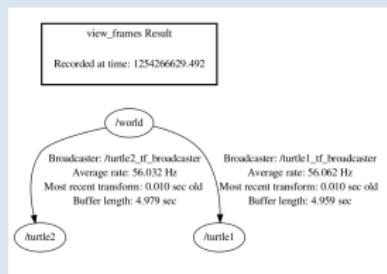
Frames:
Frame: turtle1 published by /turtle1_tf_broadcaster Average Delay: 0.000382455 Max Delay: 0...
Frame: turtle2 published by /turtle2_tf_broadcaster Average Delay: 0.000267847 Max Delay: 0...

All Broadcasters:
Node: /turtle1_tf_broadcaster 64.6996 Hz, Average Delay: 0.000382455 Max Delay: 0.000991178
Node: /turtle2_tf_broadcaster 64.7127 Hz, Average Delay: 0.000267847 Max Delay: 0.00133464
```

# Utilities

- `rosrun tf tf_echo <source_frame> <target_frame>`
- `rosrun tf tf_monitor`
- `rosrun view_frames`
- `rosrun tf static_transform_publisher`
- `rviz`

## view\_frames



# Utilities

- tf\_echo
- tf\_monitor
- view\_frames
- static\_transform\_publisher
- rviz

## static\_transform\_publisher

```
$ rosrun tf static_transform_publisher x y z yaw pitch roll frame_id child_frame_id period
$ rosrun tf static_transform_publisher x y z qx qy qz qw frame_id child_frame_id period
```



# TF data types

- Transforms and poses
- *stamped* data types (via ROS header)
- Header contains time stamp and frame names
- *StampedTransform* and *PoseStamped*
- *StampedTransform*: frame name and child frame name

# Publishing transforms

- Launch files and `static_transform_publisher`
- URDF, joint states and `robot_state_publisher`
  - Make a robot description file (URDF) and load it on the parameter server
  - Implement a node that reads joint states and publishes them
  - Run the `robot_state_publisher` node
- ⇒ Simulation tutorial tomorrow for URDF introduction
- Nodes that publish transforms



# Publishing transforms (C++)

## TransformBroadcaster

```
tf::TransformBroadcaster br;  
  
tf::Transform transform;  
transform.setOrigin(tf::Vector(x, y, z));  
transform.setRotation(  
    tf::createQuaternionFromRPY(yaw, pitch, roll));  
br.sendTransform(  
    tf::StampedTransform(  
        transform, ros::Time::now(), "/map", "/odom"));
```



# Using transforms (C++)

## TransformListener

```
tf::TransformListener listener;
tf::StampedTransform transform;
try {
    listener.lookupTransform(
        "/map", "/r_wrist_roll_link", ros::Time(0),
        transform);
} catch(tf::TransformException ex) {
    ROS_ERROR("%s", ex.what());
}
```

- *canTransform*
- *lookupTransform*
- *transformPoint*
- *transformPose*
- *transformQuaternion*



# Using transforms (C++)

## TransformListener

```
tf::TransformListener listener;
tf::StampedTransform transform;
try {
    listener.lookupTransform(
        "/map", "/r_wrist_roll_link", ros::Time(0),
        transform);
} catch(tf::TransformException ex) {
    ROS_ERROR("%s", ex.what());
}
```

- *canTransform*
- *lookupTransform*
- *transformPoint*
- *transformPose*
- *transformQuaternion*



# TF and time

- TF buffers transforms for 10 seconds
- query transforms in the past
- TF interpolates frames
- fixed frame: frame that doesn't move (reference)



# TF and time

## TransformListener methods

```
listener.lookupTransform("/turtle2", "/turtle1",
    ros::Time(0), transform);

listener.lookupTransform("/turtle2", "/turtle1",
    ros::Time::now(), transform);

ros::Time now = ros::Time::now();
listener.waitForTransform("/turtle2", "/turtle1", now,
    ros::Duration(3.0));
listener.lookupTransform("/turtle2", "/turtle1", now,
    transform);
```

- *lookupTransform*
  - Time 0: return the newest transform
  - Time now: would lead to an error because TF doesn't do extrapolation

- *waitForTransform*: block until transform is possible

# TF and time

## TransformListener methods

```
listener.lookupTransform("/turtle2", "/turtle1",
    ros::Time(0), transform);

listener.lookupTransform("/turtle2", "/turtle1",
    ros::Time::now(), transform);

ros::Time now = ros::Time::now();
listener.waitForTransform("/turtle2", "/turtle1", now,
    ros::Duration(3.0));
listener.lookupTransform("/turtle2", "/turtle1", now,
    transform);
```

- *lookupTransform*
  - Time 0: return the newest transform
  - Time now: would lead to an error because TF doesn't do extrapolation
- *waitForTransform*: block until transform is possible



# Thank you for your attention

Special thanks to Jonathan Bohren and Willow Garage

PR2 Illustration by Josh Ellingson, Willow Garage