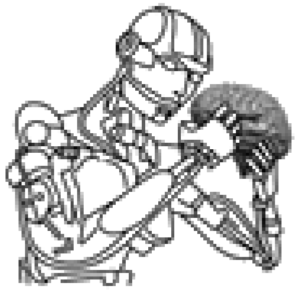
USC
UNIVERSITY
OF SOUTHERN
CALIFORNIA



CS 545 Robotics

Introduction to ROS

Slides adapted from Sachin Chitta and Radu Rusu (Willow Garage)

Overview

?

my new application

web browser

OS

email client

window manager

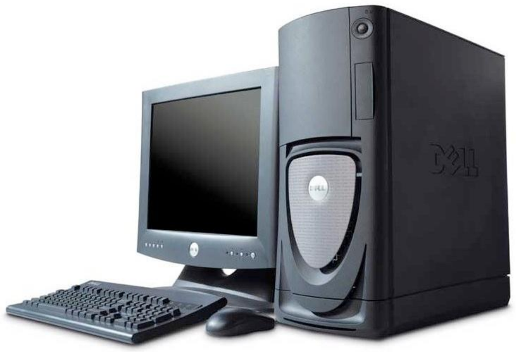
memory management

process management

scheduler

device drivers

file system



Overview

Standards

Hardware: PCI bus, USB port, FireWire, ...

Software: HTML, JPG, TCP/IP, POSIX, ...

my new application

web browser

email client

window manager

memory management

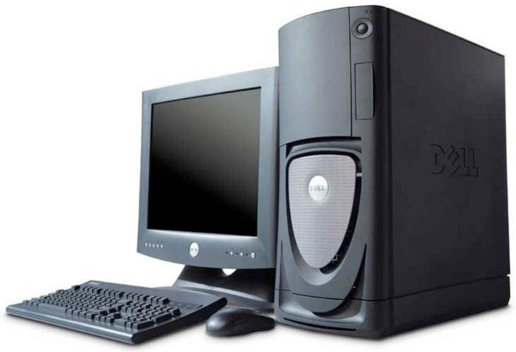
process management

scheduler

device drivers

file system

OS



Overview

...but what about robots ?

my new application

web browser

email client

window manager

memory management

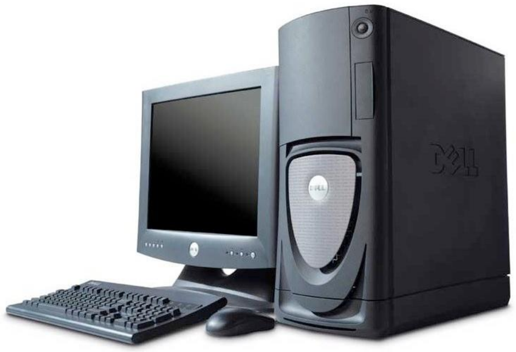
process management

scheduler

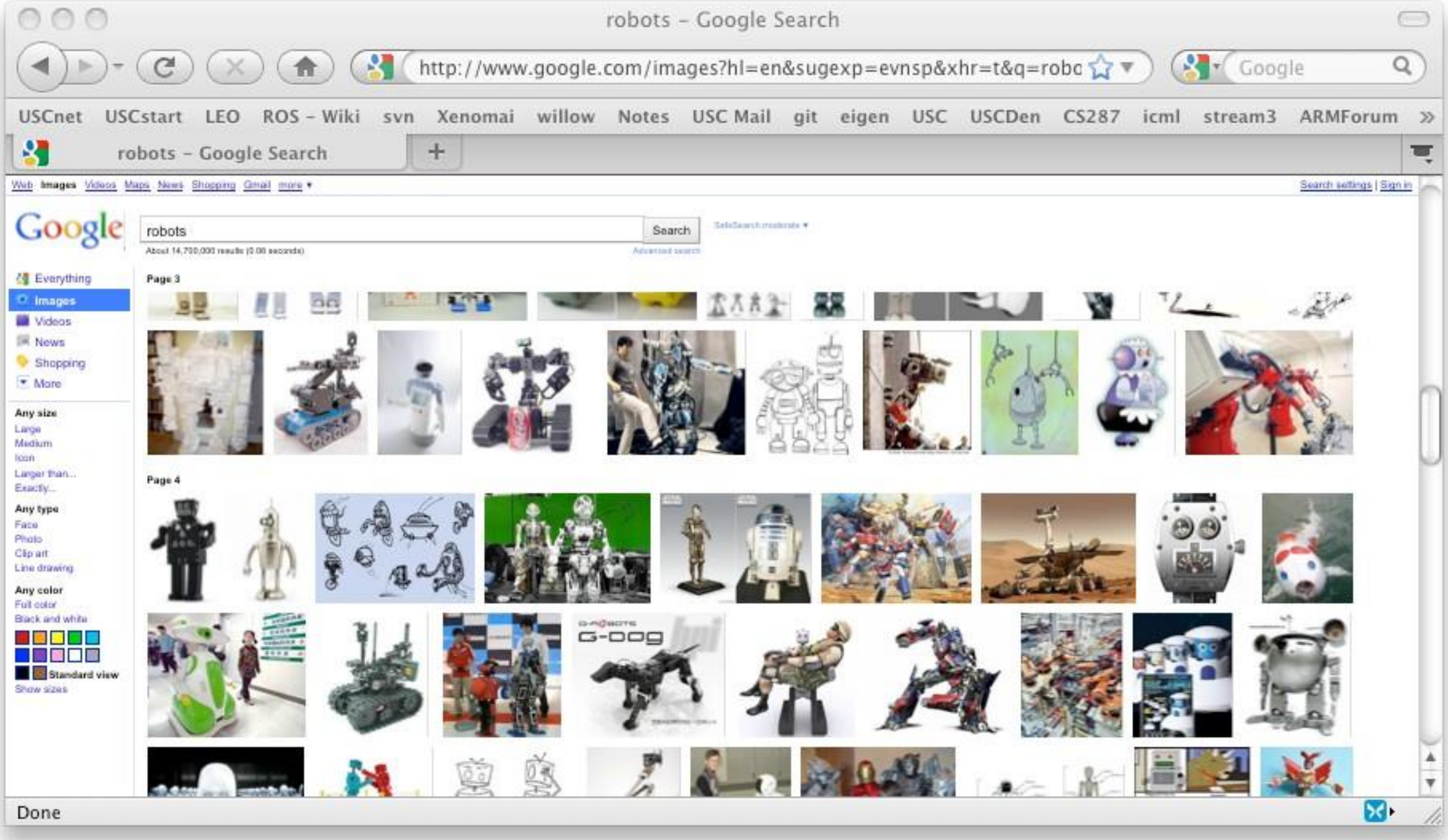
device drivers

file system

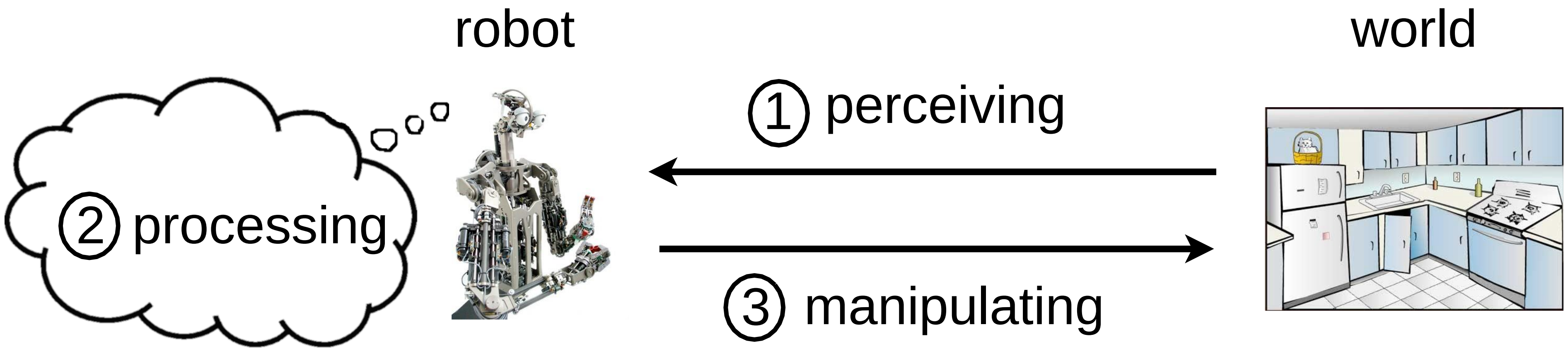
OS



Lack of standards for robotics



Typical scenario



robot

world

① perceiving

② processing

③ manipulating

① Many sensors require device drivers and calibration procedures

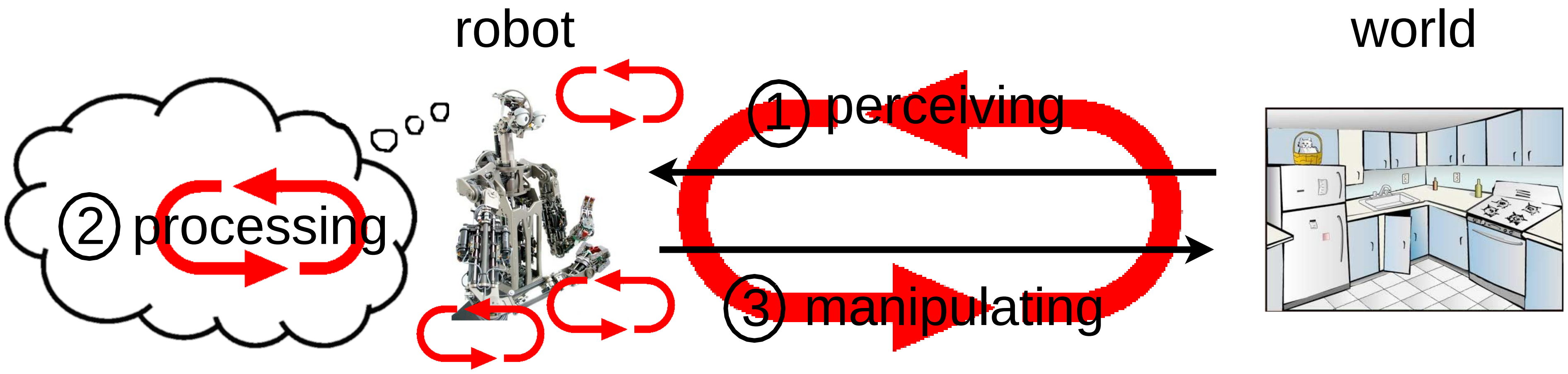
For example cameras: stereo processing, point cloud generation...

Common to many sensors: filtering, estimation, coordinate transformation, representations, voxel grid/point cloud processing, sensor fusion,...

② Algorithms for object detection/recognition, localization, navigation, path/motion planning, decision making, ...

③ Motor control: inverse kinematics/dynamics, PID control, force control, ...

Control loops



Many control loop on different time scales

Outer most **control loop** may run once every second (1Hz) or slower

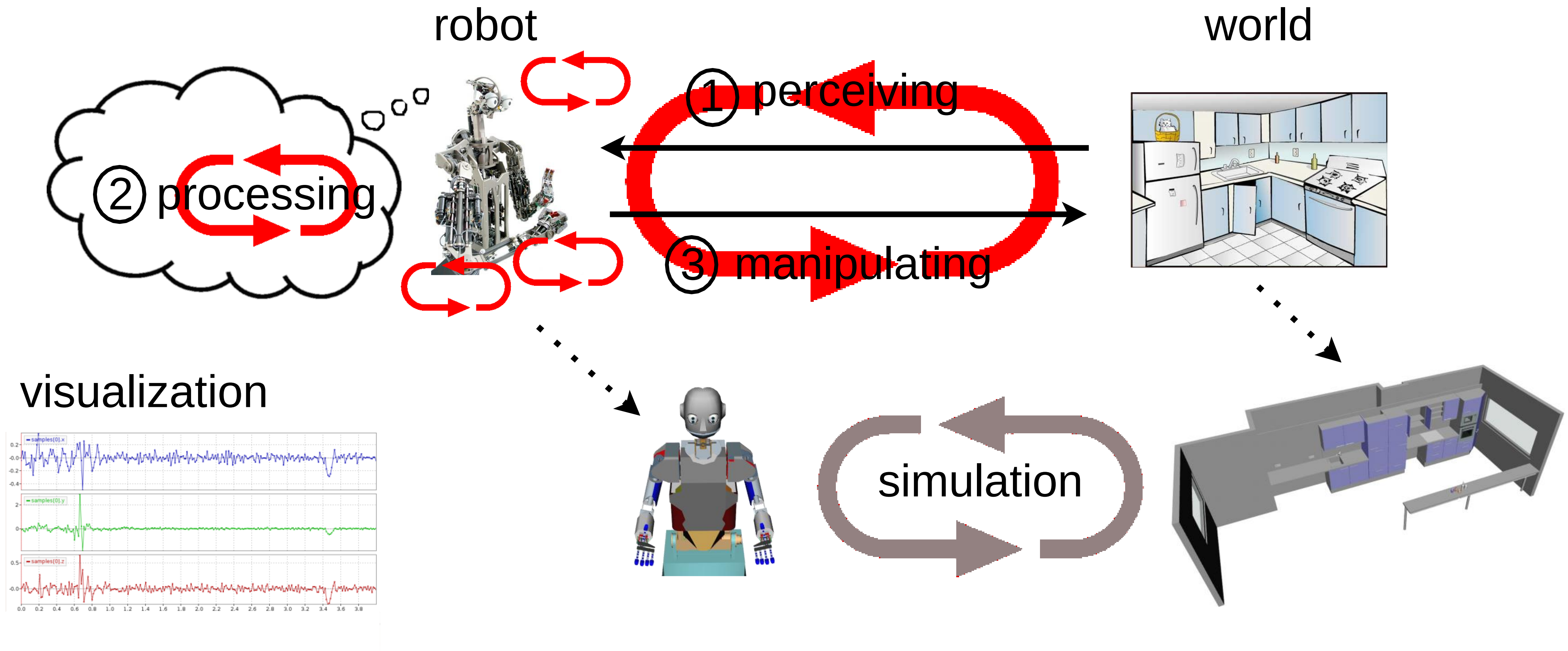
Inner most may run at 1000Hz or even higher rates

Software requirements:

Distributed processing with loose coupling. Sensor data comes in at **various time scales**.

Real time capabilities for tight motor control loops.

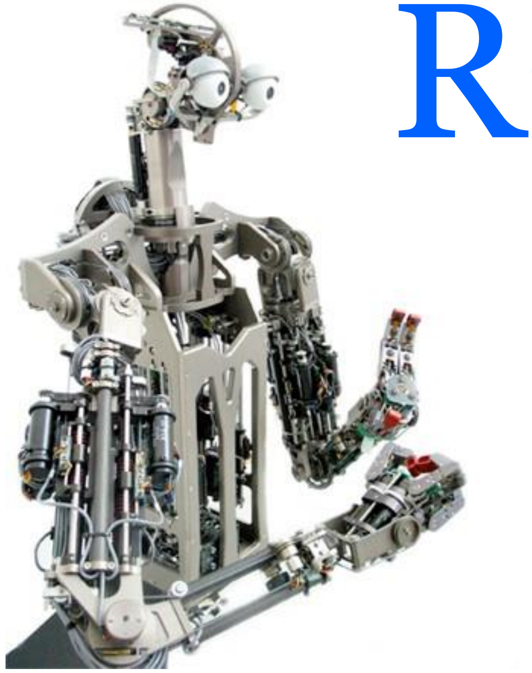
Debugging tools



Simulation: No risk of breaking real robots, reduce debugging cycles, test in super real-time, controlled physics, perfect model is available...

Visualization: Facilitates debugging, ...looking at the world from the robot’s perspective. Data trace inspections allow debugging on small time scales.

Overview



ROS

navigation

task executive

visualization

simulation

perception

control

planning

data logging

message passing

device drivers

real-time capabilities

OS

web browser

email client

window manager

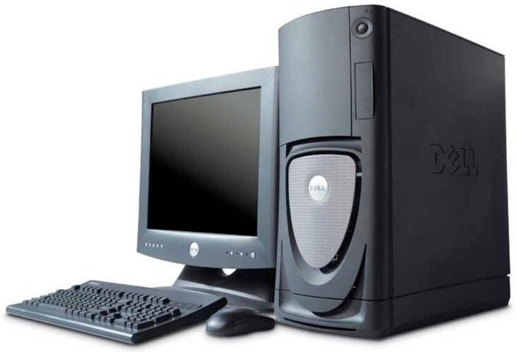
memory management

process management

scheduler

device drivers

file system



Overview

1 Orocos: <http://www.orocos.org>
2 OpenRTM: <http://www.is.aist.go.jp>
3 ROS: <http://www.ros.org>
4 OPRoS: <http://opros.or.kr>
5 JOSER: <http://www.joser.org>
6 InterModalics: <http://intermodalics.eu>
7 Denx: <http://denx.de>
8 GearBox: <http://gearbox.sourceforge.net/gbx_doc_overview.html>

Why should we agree on one standard ?

Code reuse, code sharing:

stop inventing the wheel again and again... instead build on top of each other’s code.

Ability to run the same code across multiple robots:

portability facilitates collaborations and allows for comparison of similar approaches which is very important especially in science.

What is ROS ?

ROS is an **open-source**, **meta-operating** system and stands for Robot Operating System.

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.

http://www.ros.org (documentation)

https://lists.sourceforge.net/lists/listinfo/ros-users (mailing list)

http://www.ros.org/wiki/ROS/Installation (it’s open, it’s free !!)

Mainly supported for Ubuntu linux, experimental for Mac OS X and other unix systems.

http://www.ros.org/wiki/ROS/StartGuide (tutorials)

Robots using ROS

http://www.ros.org/wiki/Robots

many more....
...and many more to come...

ROS package system

How to facilitate code sharing and code reuse ?

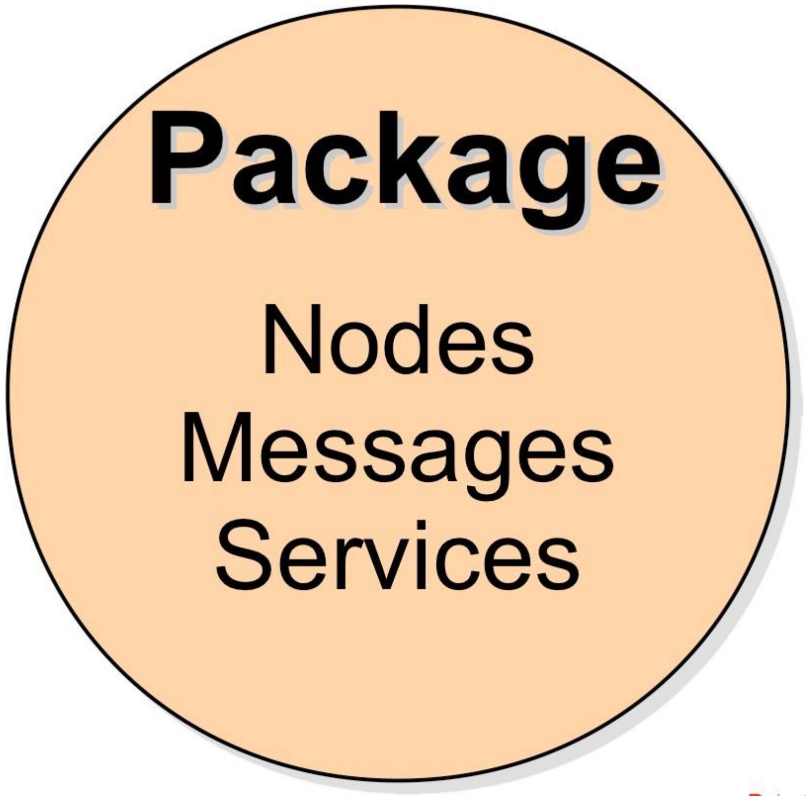
A package is a **building block** and implements a reusable capability

- Complex enough to be useful
- Simple enough to be reused by other packages

A **package** contains one or more executable processes (nodes) and provides a ROS interface:

Messages describe the data format of the in/output of the nodes. For example, a door handle detection node gets camera images as input and spits out coordinates of detected door handles.

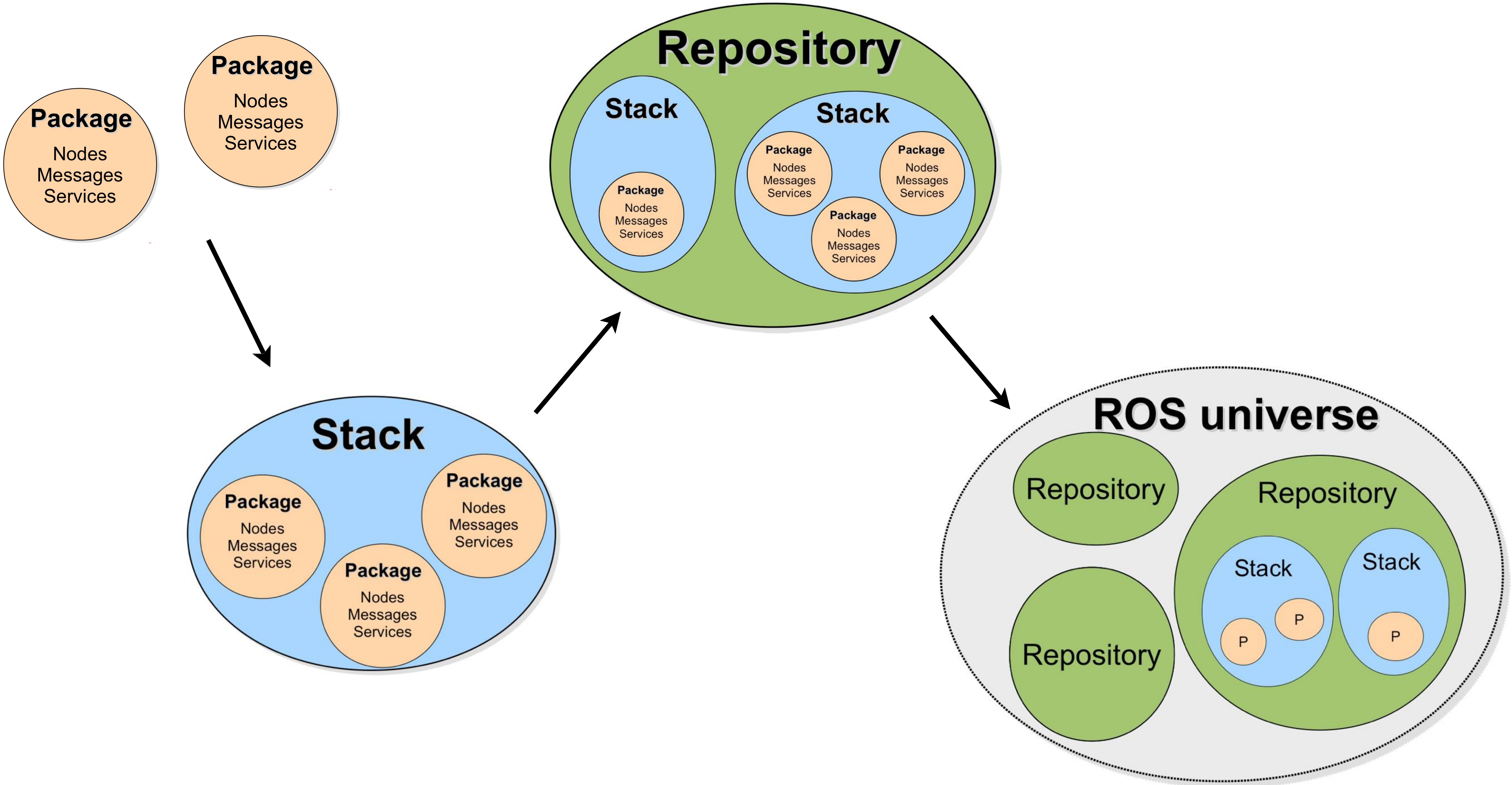
Service and **topics** provide the standardized ROS interface to the rest of the system.



Package

Nodes
Messages
Services

ROS package system

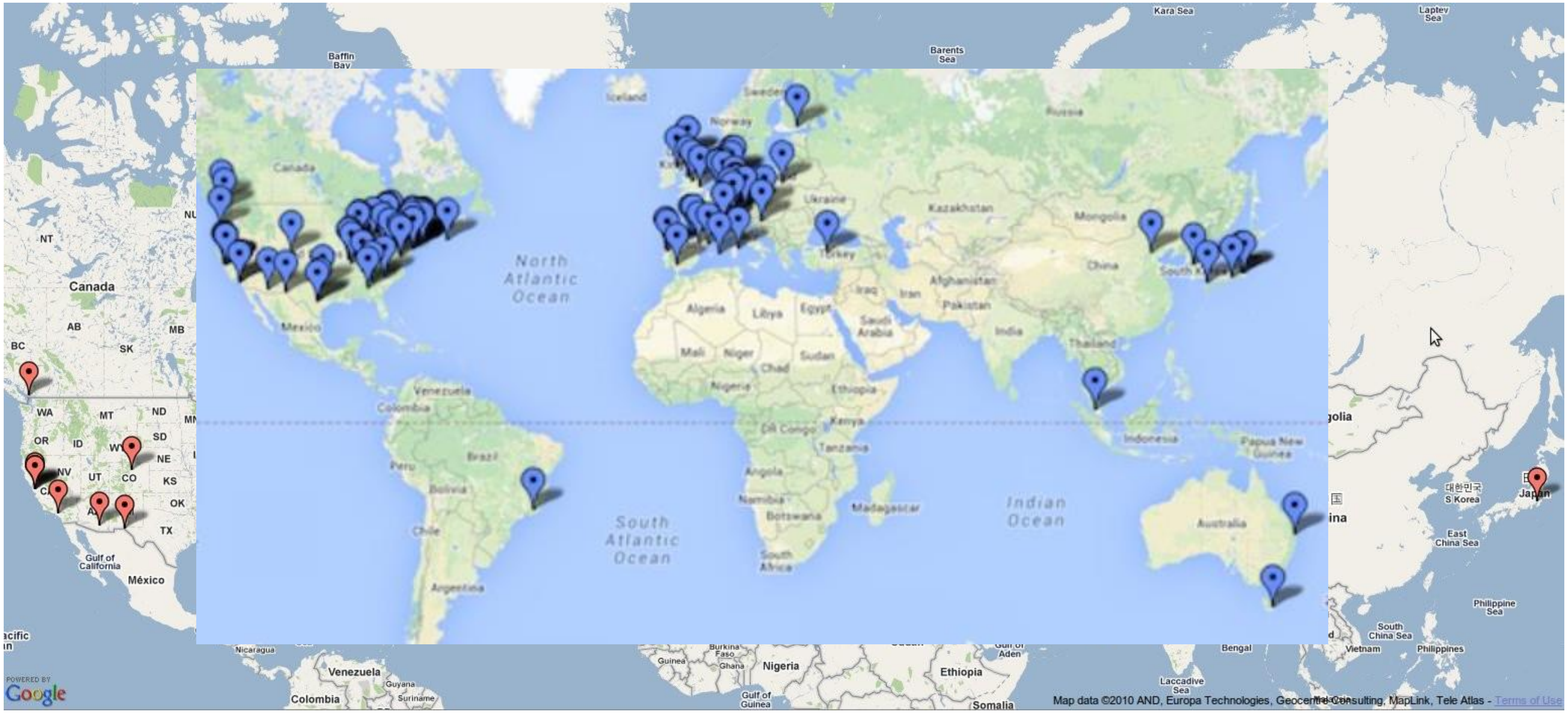


ROS package system

Collection of packages and stacks, hosted online

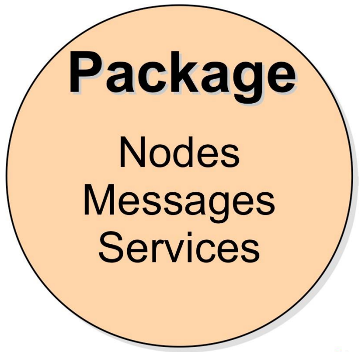
Many repositories (>50): Stanford, CMU, TUM, Leuven, USC, Bosch, ...

http://www.ros.org/wiki/Repositories (check it out...)



Map data ©2010 AND, Europa Technologies, Geocentre Consulting, MapLink, Tele Atlas - Terms of Use

ROS package system



Package

Nodes
Messages
Services

ROS packages tend to follow a common structure. Here are some of the directories and files you may notice.

- `bin/`: compiled binaries (**C++ nodes**)
- `include/package_name`: C++ include headers
- `msg/`: **Message** (msg) types
- `src/package_name/`: Source files
- `srv/`: **Service** (srv) types
- `scripts/`: executable scripts (**Python nodes**)
- `launch/`: launch files
- `CMakeLists.txt`: CMake build file (see **CMakeLists**)
- `manifest.xml`: Package **Manifest**
- `mainpage.dox`: Doxygen mainpage documentation

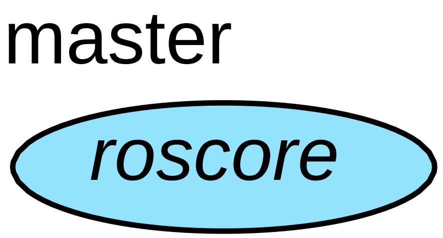
ROS package system

`manifest.xml` →

The **manifest** is a minimal specification about a package and supports a wide variety of ROS tools.

```
<package>
  <description brief="one line of text">
    long description goes here,
    <em>XHTML is allowed</em>
  </description>
  <author>Alice/alice@somewhere.bar</author>
  <license>BSD</license>

  <depend package="roscpp"/>
  <depend package="my_package"/>
  <rosdep name="libreadline5-dev"/>
  <export>
    <cpp cflags="-I${prefix}/include"
         lflags="-L${prefix}/lib -lmy_lib"/>
  </export>

</package>
```

ROS core



master

roscore

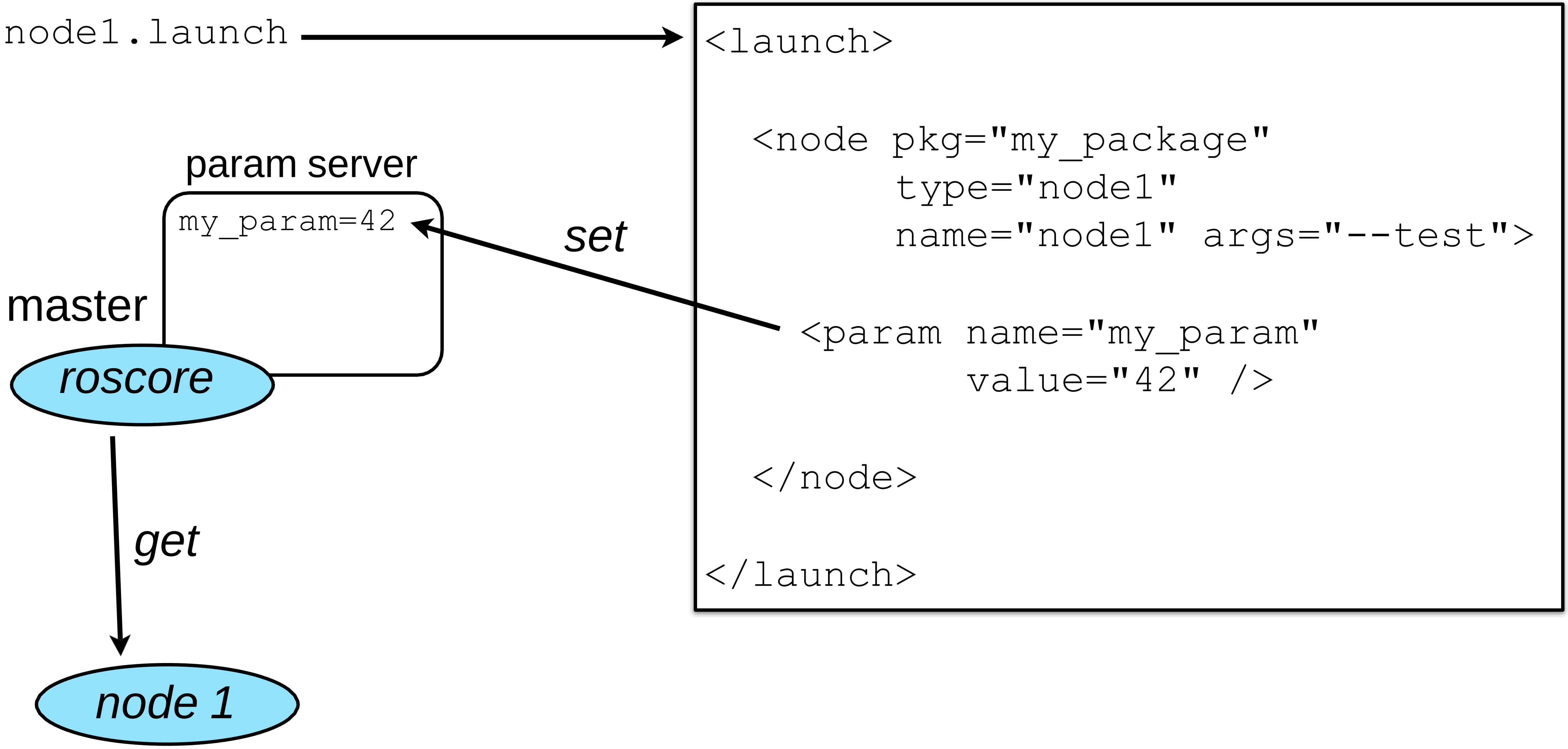
The **roscore** is a collection of nodes and programs that are pre-requisites for a ROS-based system.

It provides naming and registration services to the rest of the **nodes** in the ROS system. It tracks publishers and subscribers to **topics** as well as **services**.

The role of the master is to enable individual ROS **nodes** to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.

ROS uses socket communication to facilitate networking. The **roscore** starts on `http://my_computer:11311`

ROS package system

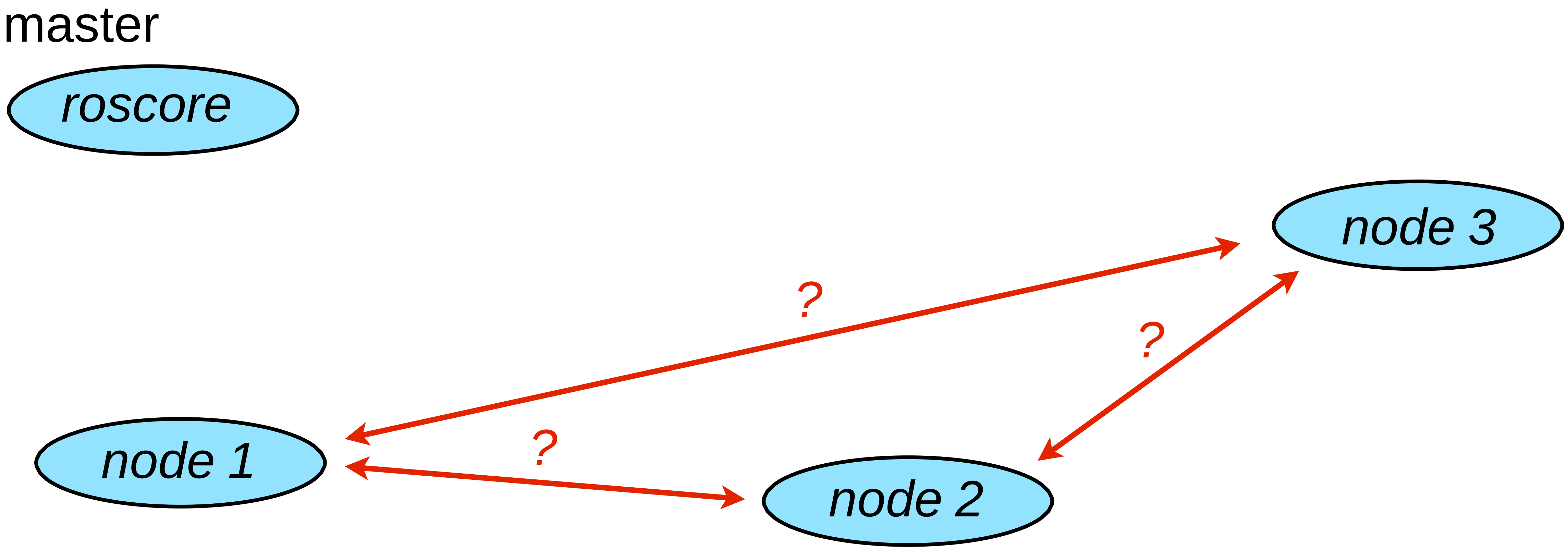


```
<launch>

  <node pkg="my_package"
        type="node1"
        name="node1" args="--test">

    <param name="my_param"
           value="42" />

  </node>

</launch>
```

ROS: message passing

Problem:

Synchronization and message passing across multiple processes, maybe even across multiple computer and/or robots.

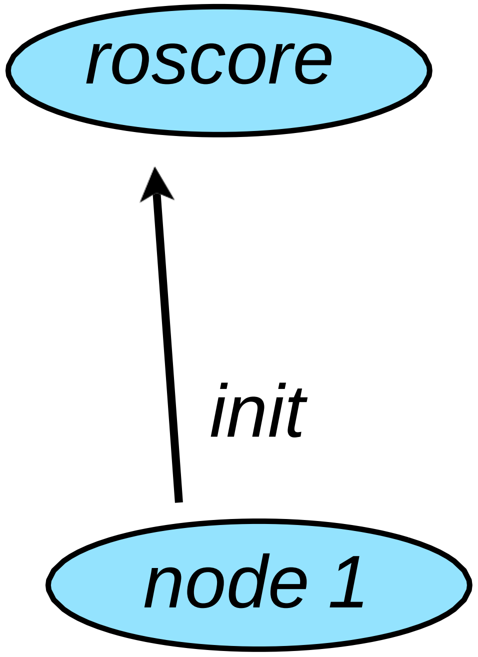


ROS: message passing

Problem:

Synchronization and message p
across multiple computer and/or

master



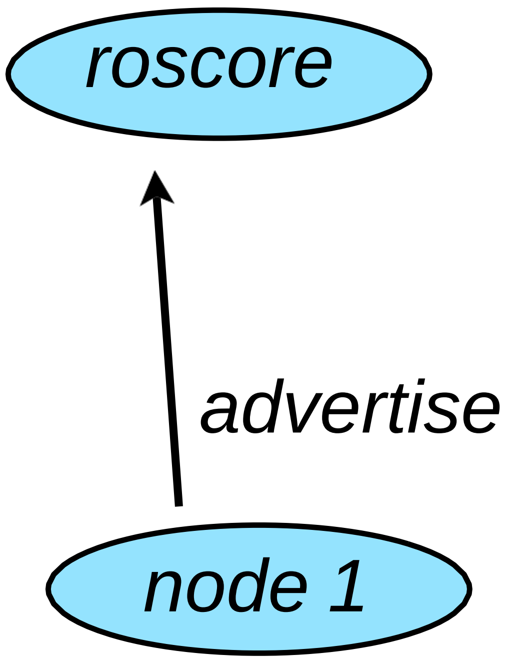
```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>
int main(int argc, char **argv)
{
  ros::init(argc, argv, "node1");
  ros::NodeHandle n;
  ros::Publisher chatter_pub =
    n.advertise<std_msgs::String>("info", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

ROS: message passing

Problem:

Synchronization and message p
across multiple computer and/or

master

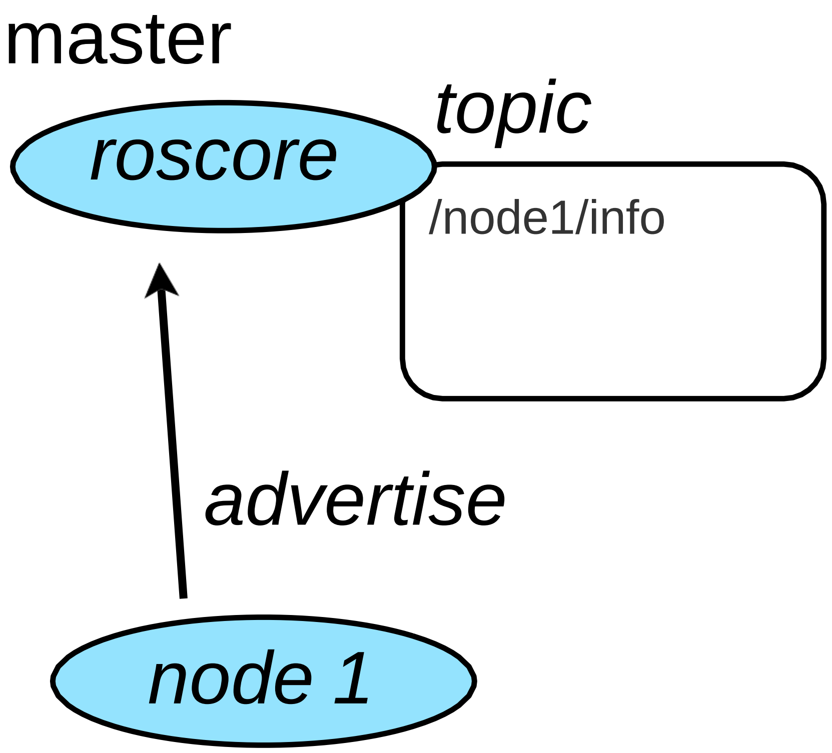


```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>
int main(int argc, char **argv)
{
  ros::init(argc, argv, "node1");
  ros::NodeHandle n;
  ros::Publisher chatter_pub =
    n.advertise<std_msgs::String>("info", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

ROS: message passing

Problem:

Synchronization and message p
across multiple computer and/or

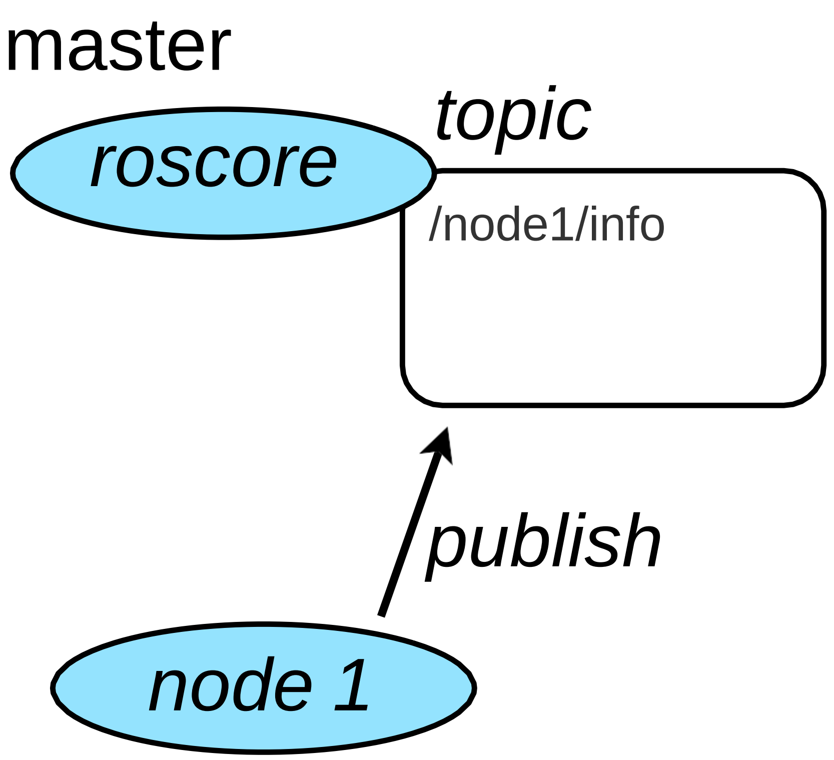


```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>
int main(int argc, char **argv)
{
  ros::init(argc, argv, "node1");
  ros::NodeHandle n;
  ros::Publisher chatter_pub =
    n.advertise<std_msgs::String>("info", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

ROS: message passing

Problem:

Synchronization and message p
across multiple computer and/or

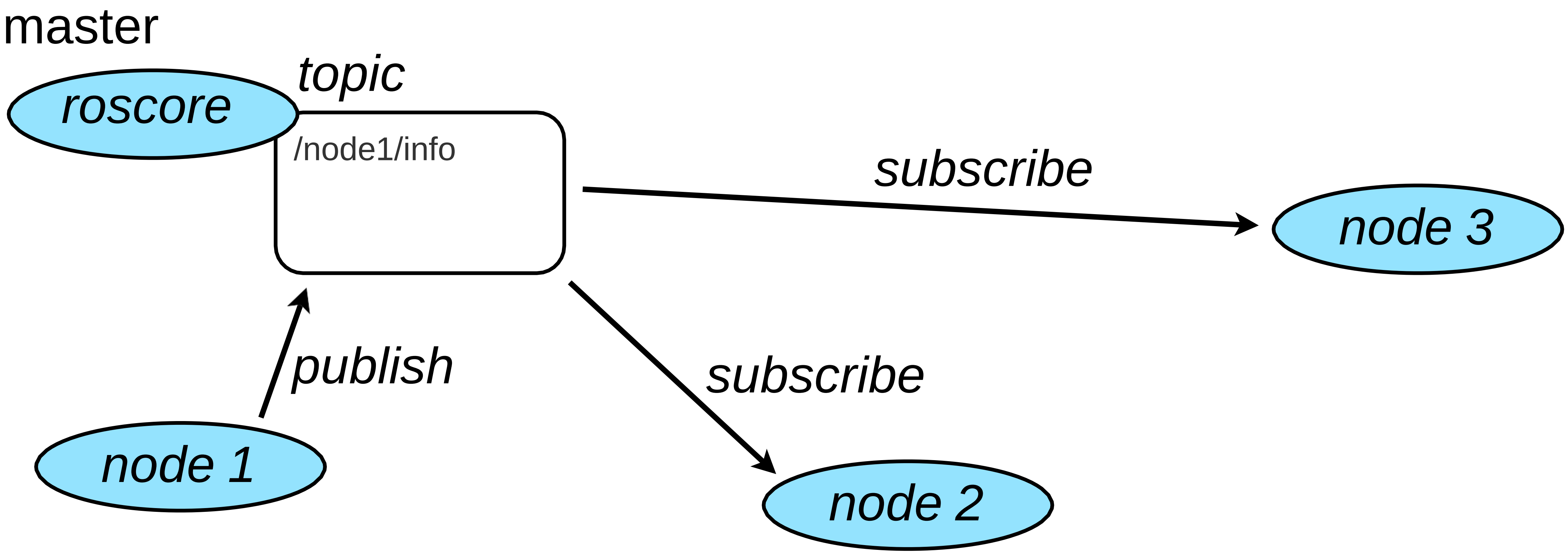


```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>
int main(int argc, char **argv)
{
  ros::init(argc, argv, "node1");
  ros::NodeHandle n;
  ros::Publisher chatter_pub =
    n.advertise<std_msgs::String>("info", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

ROS: message passing

Problem:

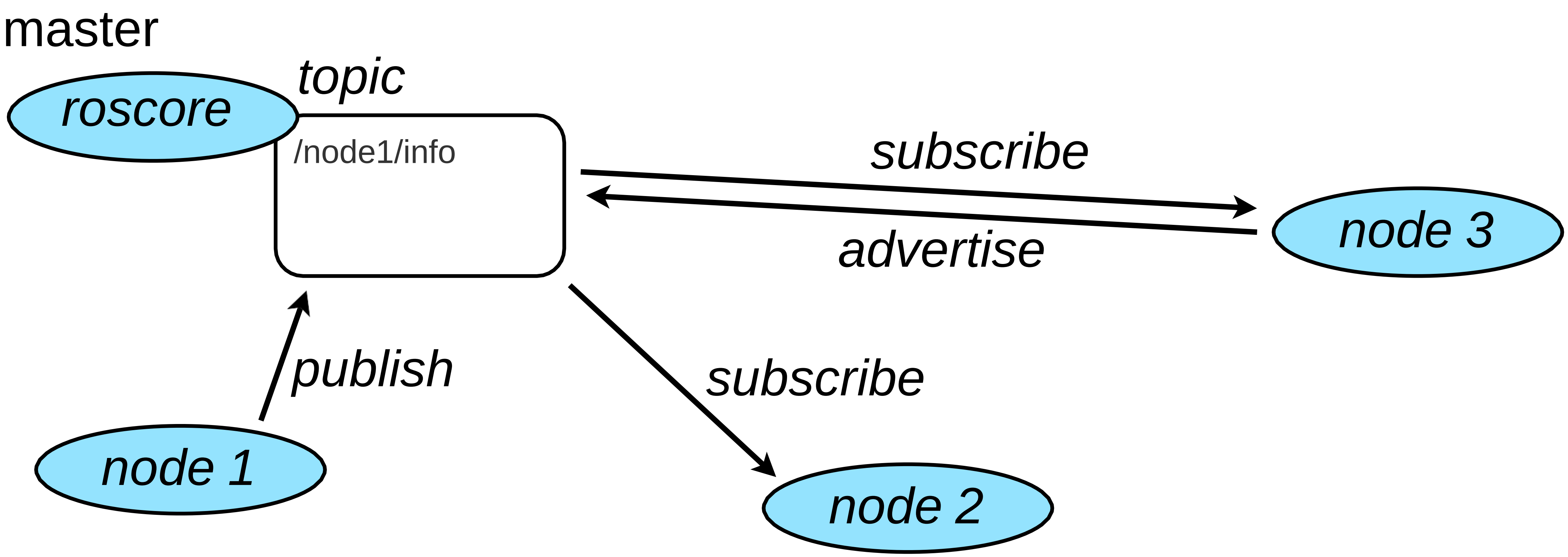
Synchronization and message passing across multiple processes, maybe even across multiple computer and/or robots.



ROS: message passing

Problem:

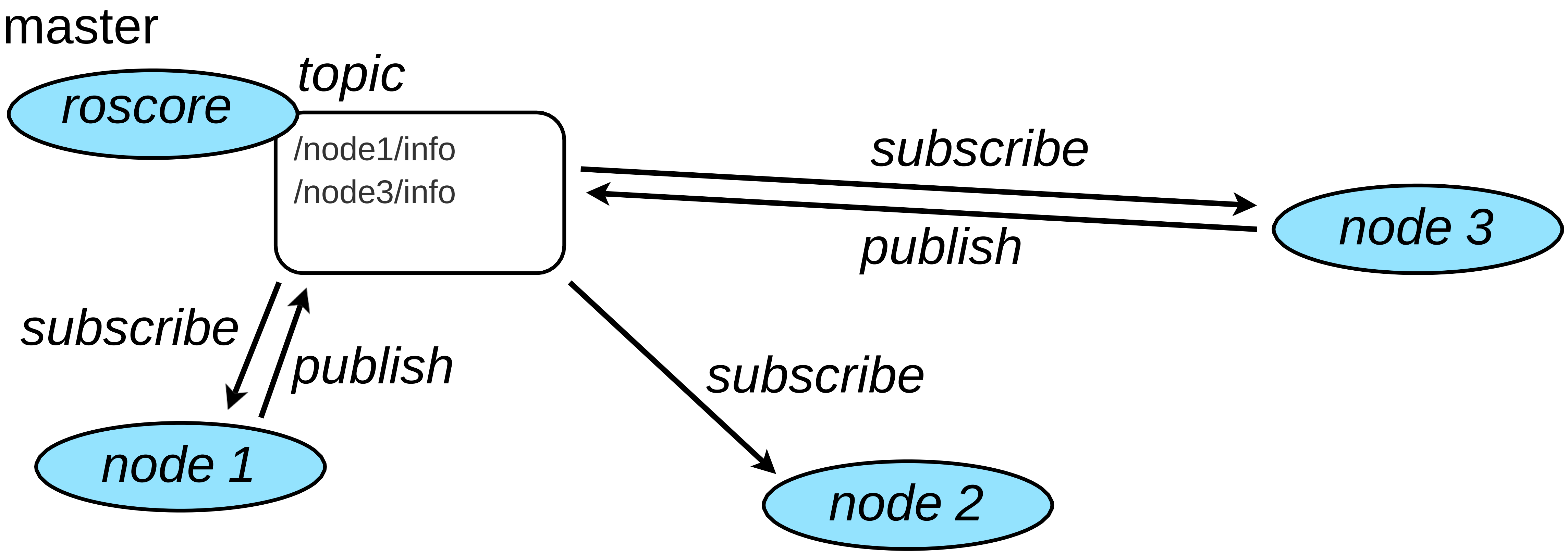
Synchronization and message passing across multiple processes, maybe even across multiple computer and/or robots.



ROS: message passing

Problem:

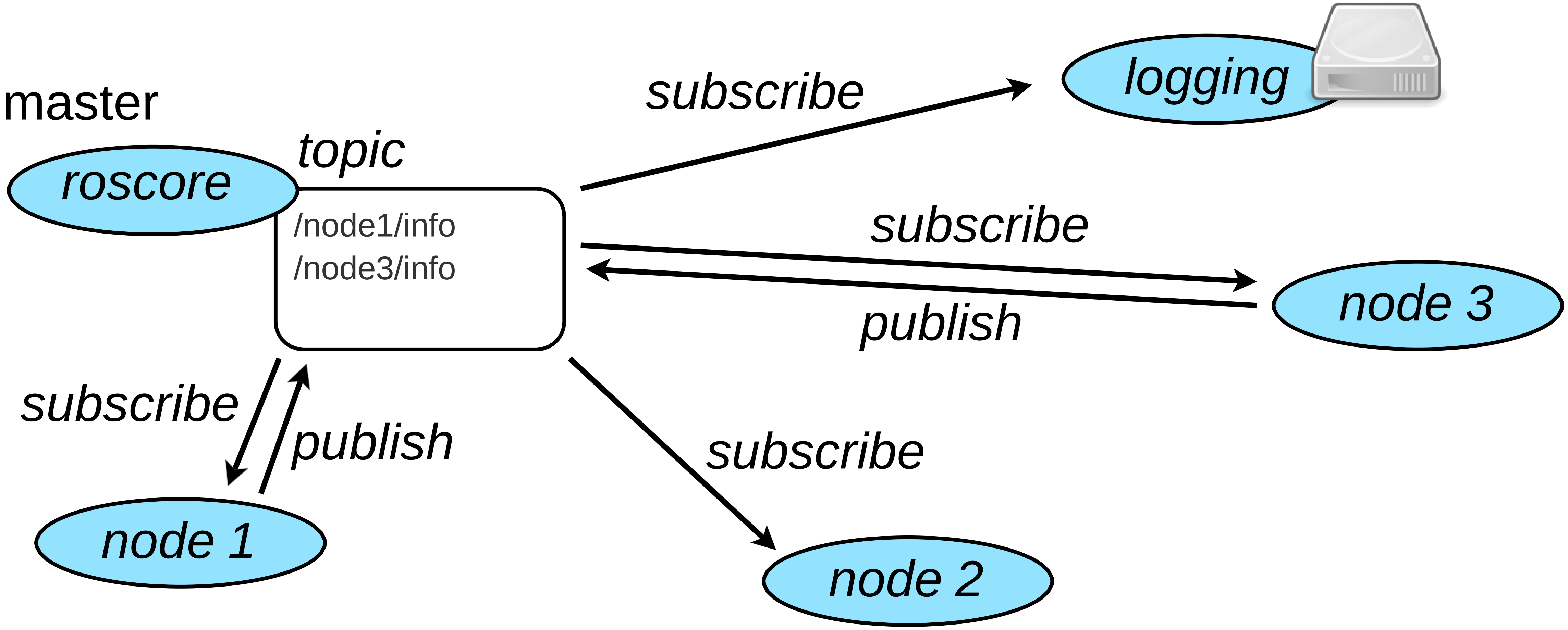
Synchronization and message passing across multiple processes, maybe even across multiple computer and/or robots.



ROS: logging

Problem:

Synchronization and message passing across multiple processes, maybe even across multiple computer and/or robots.



ROS: logging

rosbag: This is a set of tools for recording from and playing back to ROS topics. It can be used to mimic real sensor streams for offline debugging.

http://www.ros.org/wiki/rosbag

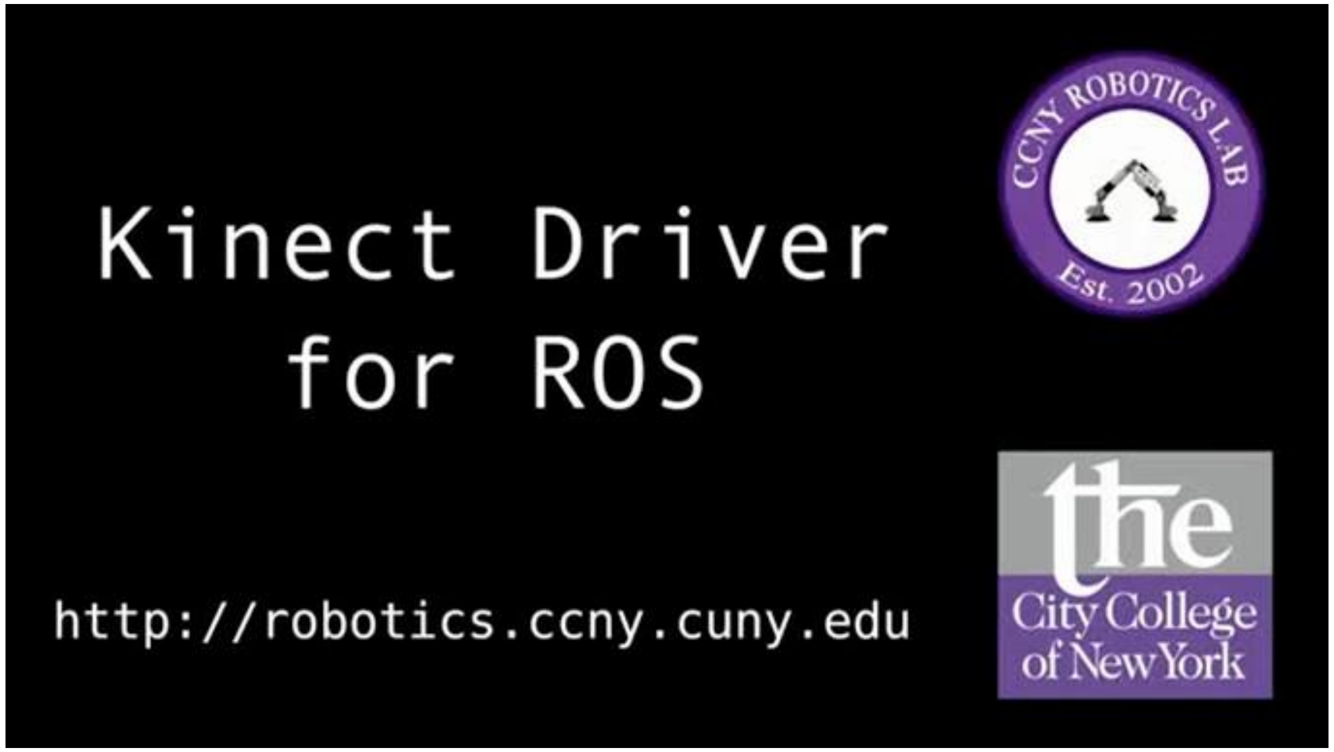
ROS: device drivers

Problem:

Many sensors do not come with standardized interfaces. Often the manufacturer only provides support for a single operating system (e.g. Microsoft Windows).

Thus, everybody that wants to use a particular sensor is required to write their own device driver, which is time consuming and tedious.

Instead, a few people did the work and the rest of the world (re-)uses their code and builds on top of it.

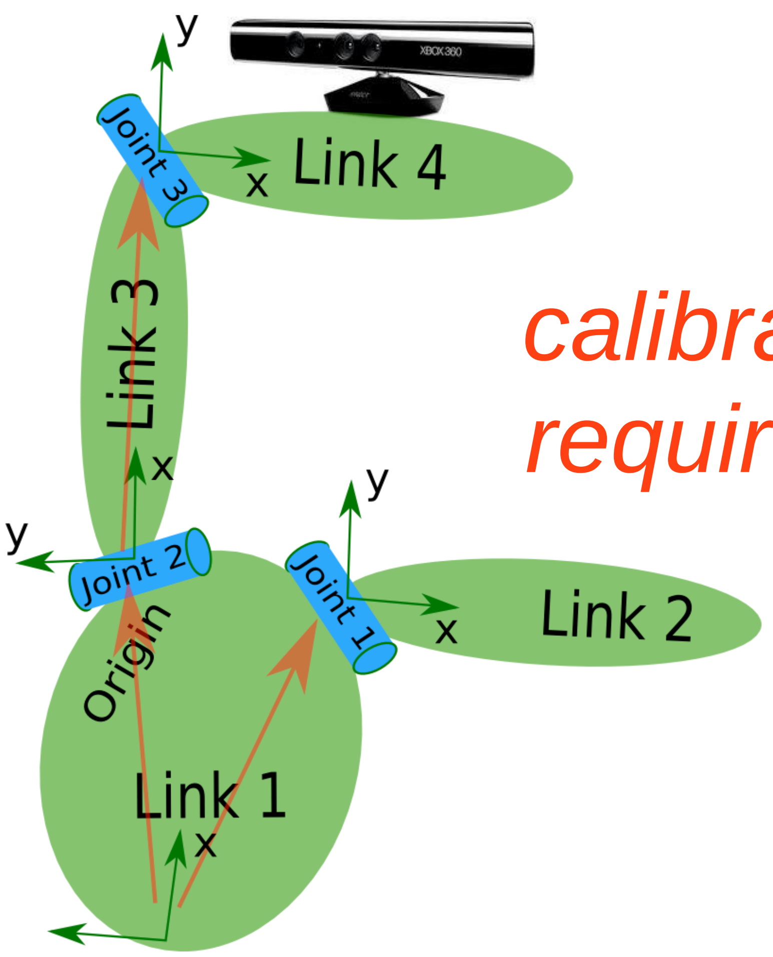


Kinect Driver for ROS

http://robotics.ccny.cuny.edu

ROS: robot descriptions

urdf: This package contains a C++ parser for the **U**nified **R**obot **D**escription **F**ormat (URDF), which is an XML format for representing a robot model.



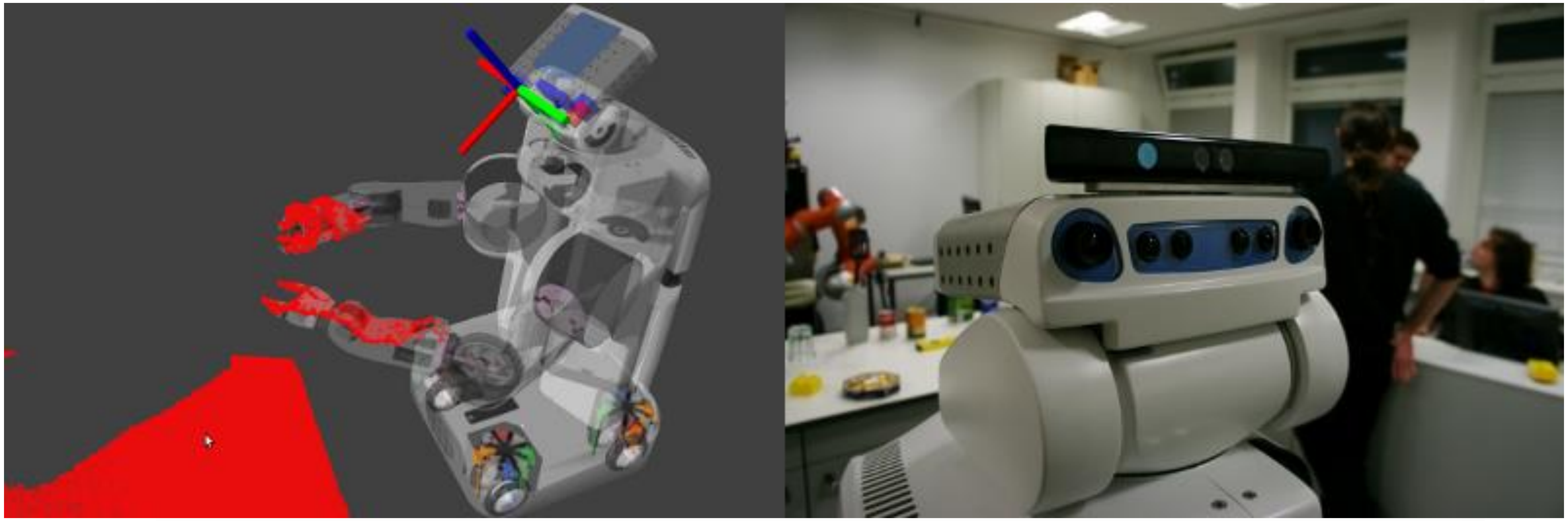
calibration required !!

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="kinect_link"/>
  </joint>
</robot>
```

http://www.ros.org/wiki/urdf

ROS: calibration

Provides a toolchain running through the robot calibration process. This involves capturing pr2 calibration data, estimating pr2 parameters, and then updating the PR2 URDF.



http://www.ros.org/wiki/pr2_calibration

ROS: visualization

rviz: This is a 3D visualization environment for robots. It allows you to see the world through the eyes of the robot.

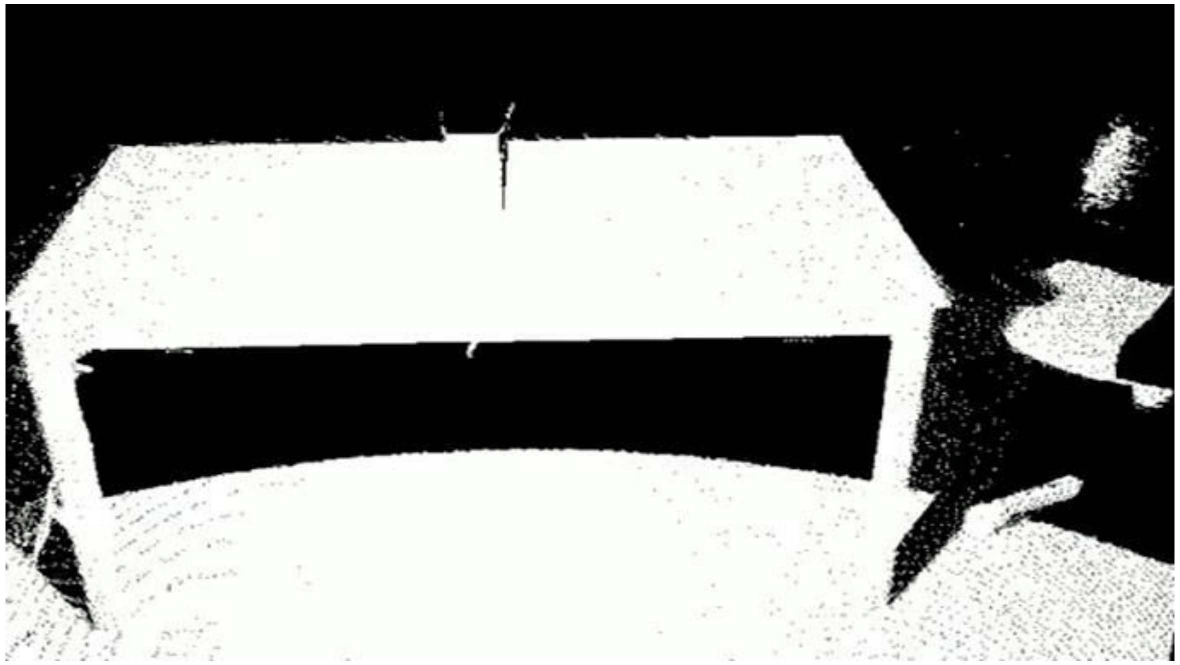
http://www.ros.org/wiki/rviz

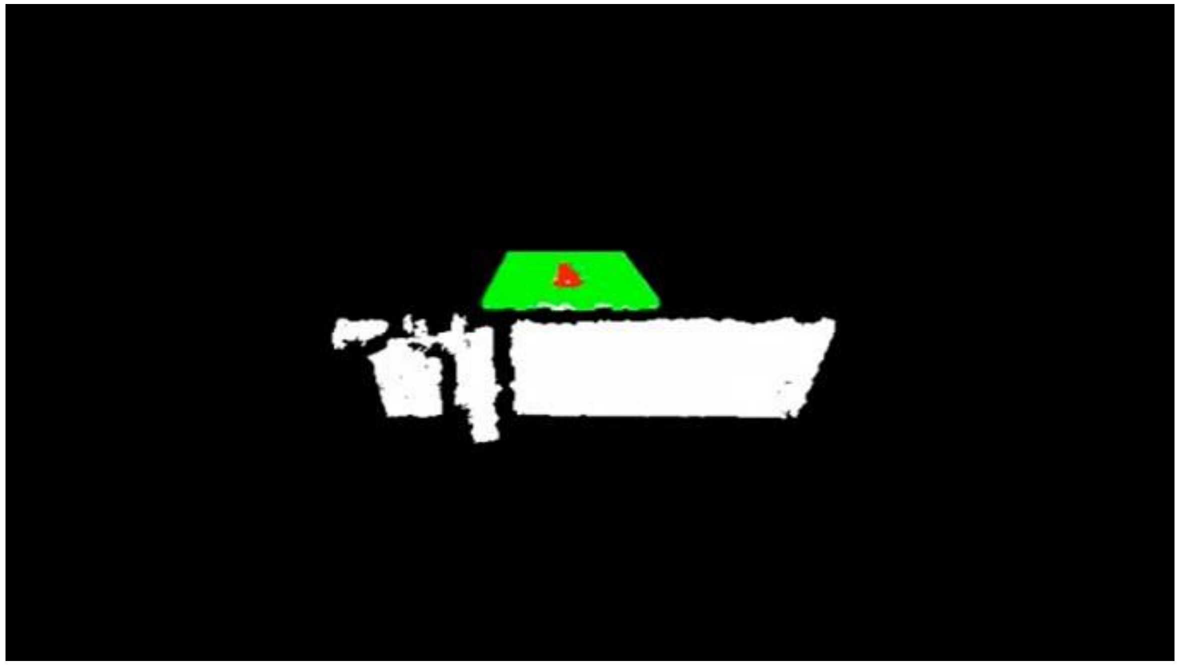
ROS: 2D/3D perception

OpenCV: (**Open** Source **C**omputer **V**ision) is a library of programming functions for real time computer vision. http://opencv.willowgarage.com/wiki/

Check out CS 574 (Prof. Ram Nevatia) !!

PCL - **P**oint **C**loud **L**ibrary: a comprehensive open source library for **n-D Point Clouds** and **3D geometry processing**. The library contains numerous state-of-the art algorithms for: filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation, etc.

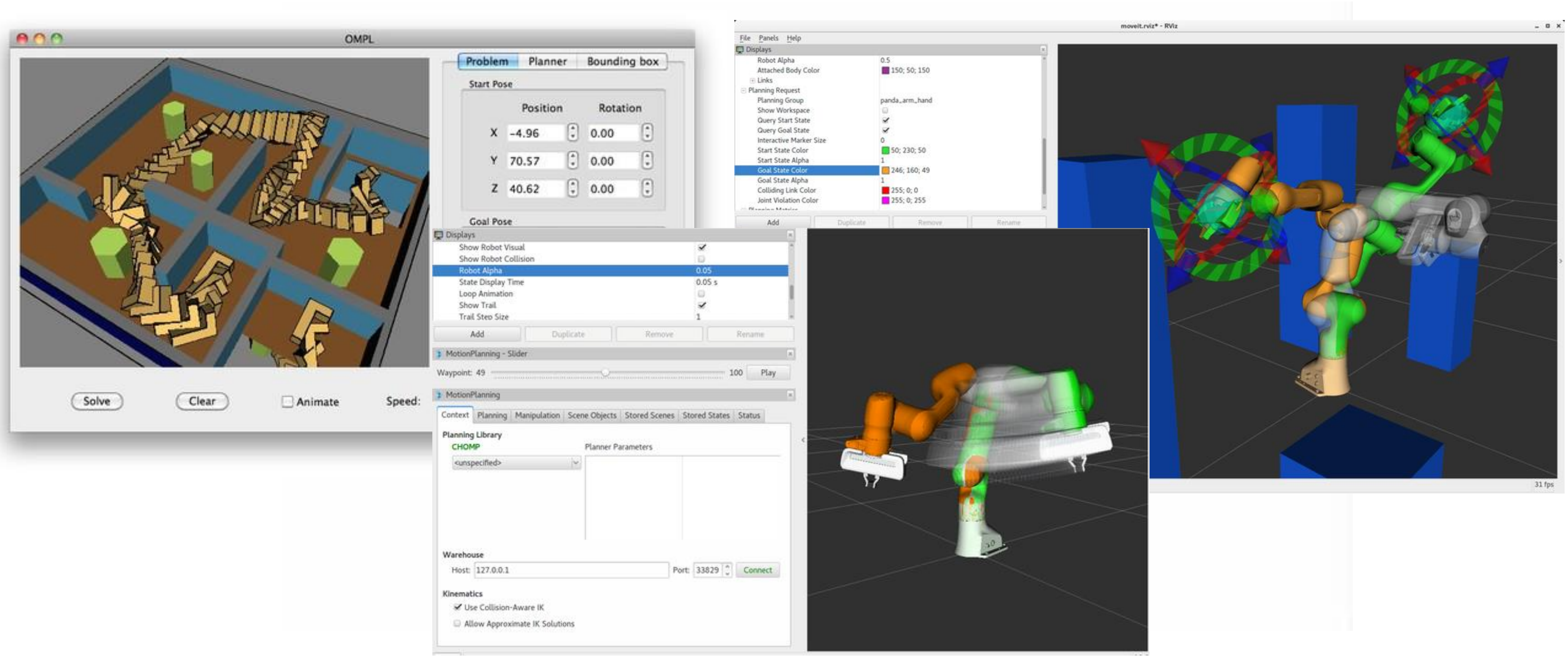




http://www.ros.org/wiki/pcl

ROS: planning

The **motion_planners** stack contains different motion planners including probabilistic motion planners, search-based planners, and motion planner based on trajectory optimization.



Start Pose

	Position	Rotation
X	-4.96	0.00
Y	70.57	0.00
Z	40.62	0.00

Goal Pose

Displays

Robot Alpha	0.5
Attached Body Color	150; 50; 150
⊞ Links	
⊟ Planning Request	
Planning Group	panda_arm_hand
Show Workspace	□
Query Start State	✓
Query Goal State	✓
Interactive Marker Size	0
Start State Color	50; 230; 50
Start State Alpha	1
Goal State Color	246; 160; 49
Goal State Alpha	1
Colliding Link Color	255; 0; 0
Joint Violation Color	255; 0; 255

Displays

Show Robot Visual	✓
Show Robot Collision	□
Robot Alpha	0.05
State Display Time	0.05 s
Loop Animation	□
Show Trail	✓
Trail Step Size	1

MotionPlanning - Slider

Waypoint: 49 100 Play

MotionPlanning

Context | Planning | Manipulation | Scene Objects | Stored Scenes | Stored States | Status

Planning Library

CHOMP

<unspecified>

Planner Parameters

Warehouse

Host: 127.0.0.1 Port: 33829 Connect

Kinematics

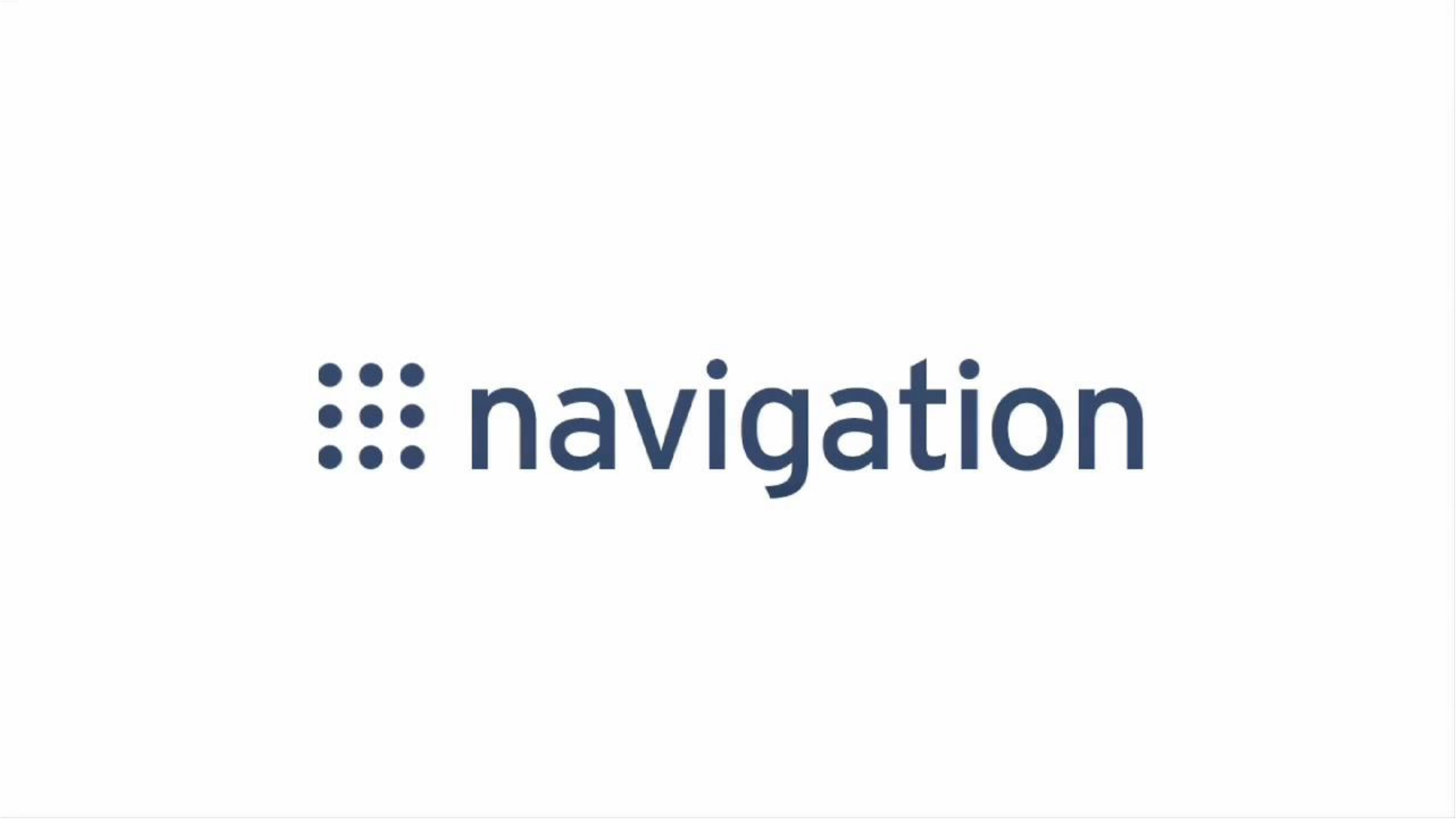
✓ Use Collision-Aware IK

□ Allow Approximate IK Solutions

http://www.ros.org/wiki/motion_planners

ROS: navigation

navigation: A 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base.



http://www.ros.org/wiki/navigation

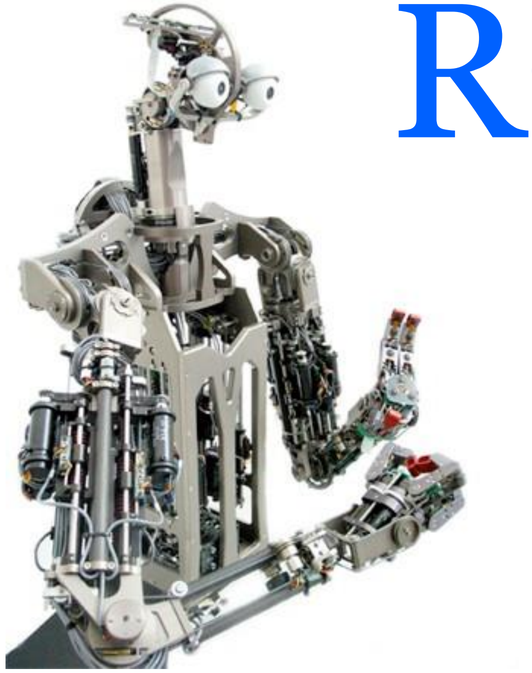
ROS: task executive

SMACH, which stands for 'state machine', is a task-level architecture for rapidly creating complex robot behavior.

http://www.ros.org/wiki/smach

Example application

Overview



ROS

navigation

task executive

visualization

simulation

perception

control

planning

data logging

message passing

device drivers

real-time capabilities

OS

web browser

email client

window manager

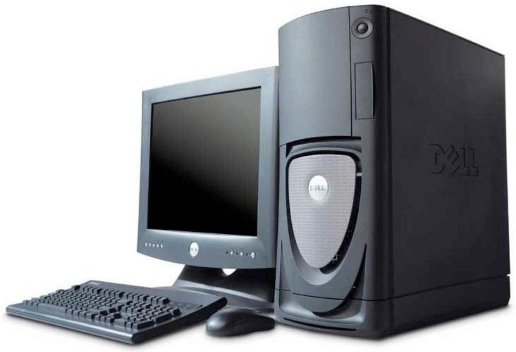
memory management

process management

scheduler

device drivers

file system



Why should one use ROS?

Build on top of existing software, make use of existing tools, and focus on your own research.

Provide the community your own work such that people can reproduce your experiments and build on top of it.

More information about ROS

Stanford Course: Robot Perception

http://pr.willowgarage.com/wiki/Stanford_CS324_PerceptionForManipulation

PR2 workshop (Good tutorial videos)

http://www.ros.org/wiki/Events/PR2BetaTraining/Videos