

## CS 545 Robotics

Introduction to





my new application

web browser

email client

window manager

memory management

process management

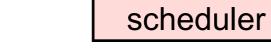


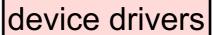


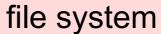
















#### **Standards**

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

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



my new application

web browser



OS

window manager

memory management

process management

scheduler

device drivers

file system









...but what about robots



my new application

web browser

email client

, US

( window manager

process management

scheduler

memory management

device drivers

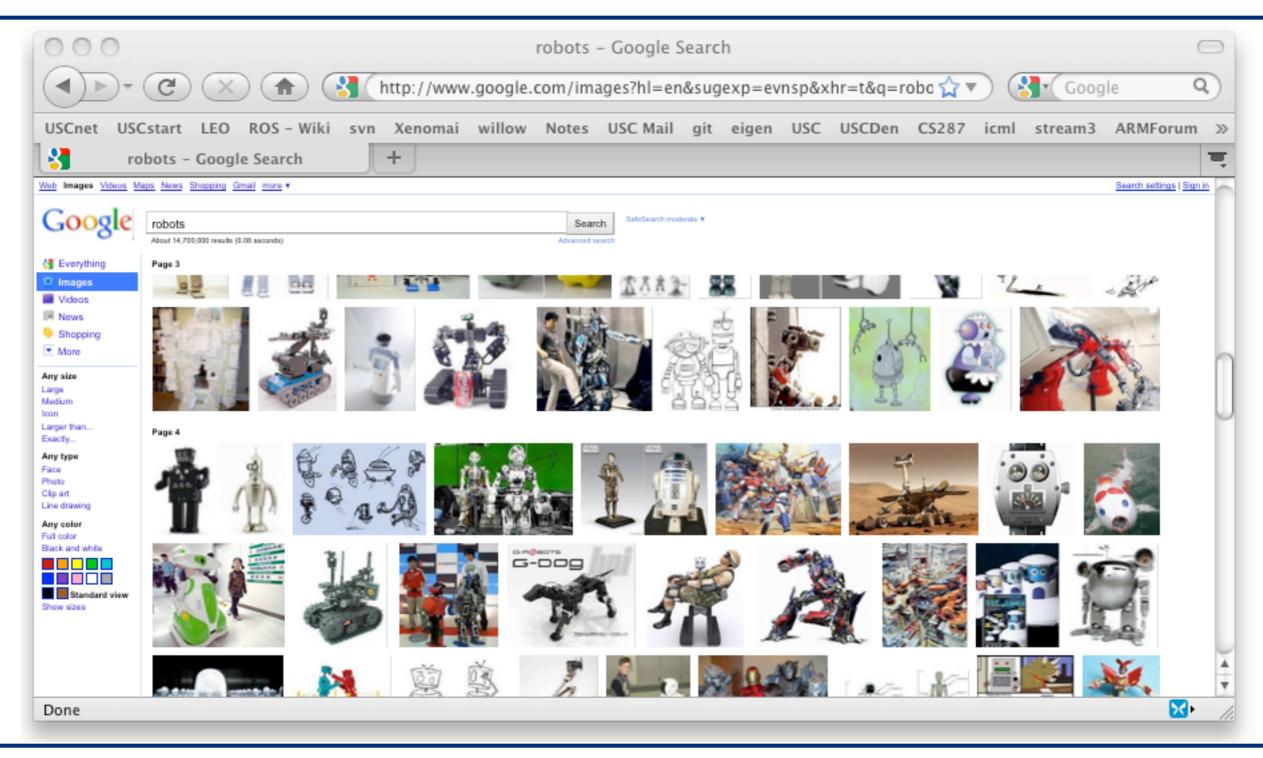
file system







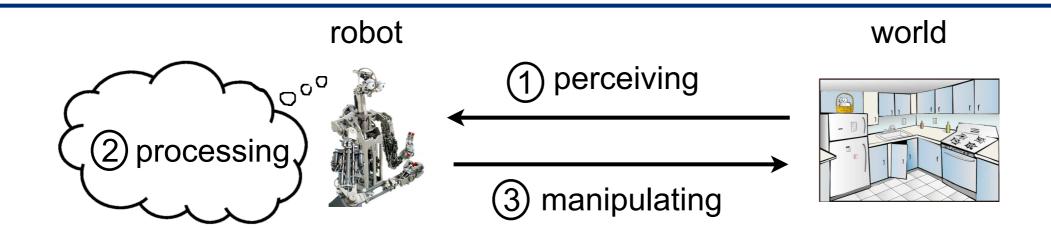
## Lack of standards for robotics







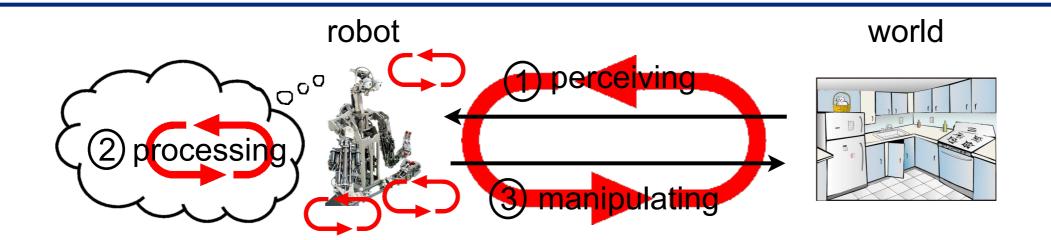
## Typical scenario



- 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,...
- 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:

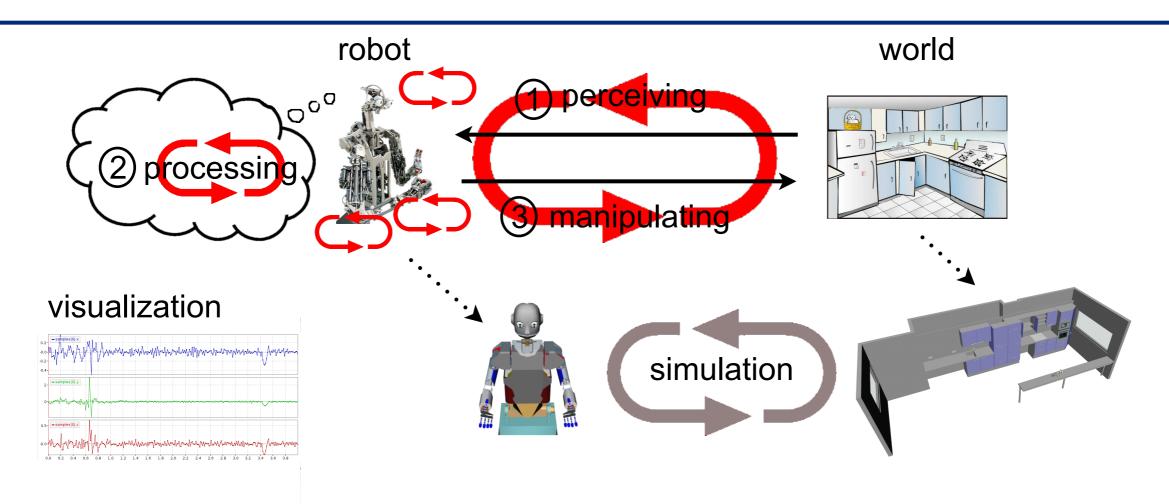
Distributed processing with loose coupling. Sensor data comes in at various time scales.

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.







navigation

task executive

visualization

simulation

perception

control

planning

data logging

message passing

device drivers

real-time capabilities

web browser

email client

OS

( window manager

memory management

process management

scheduler

device drivers

file system







- [1] Orocos: < < http://www.orocos.org >
- [2] OpenRTM: < <a href="http://www.is.aist.go.jp">http://www.is.aist.go.jp</a>>
- [3] ROS: <<a href="http://www.ros.org">http://www.ros.org</a>>
- [4] OPRoS: <<u>http://opros.or.kr</u>>
- [5] JOSER: < <a href="http://www.joser.org">http://www.joser.org</a>>
- [6] InterModalics: < <a href="http://intermodalics.eu">http://intermodalics.eu</a>>
- [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







How to facilitate code sharing and code reuse?

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

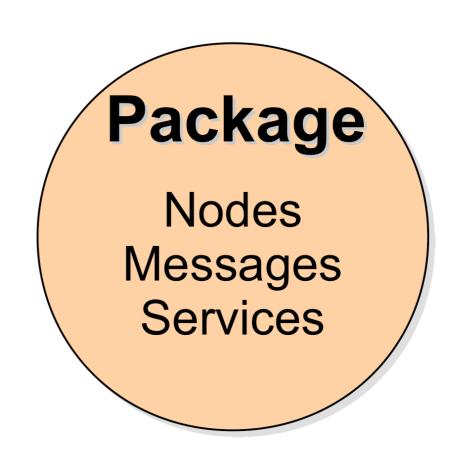
Complex enough to be useful

Simple enough to be reused by other packages

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

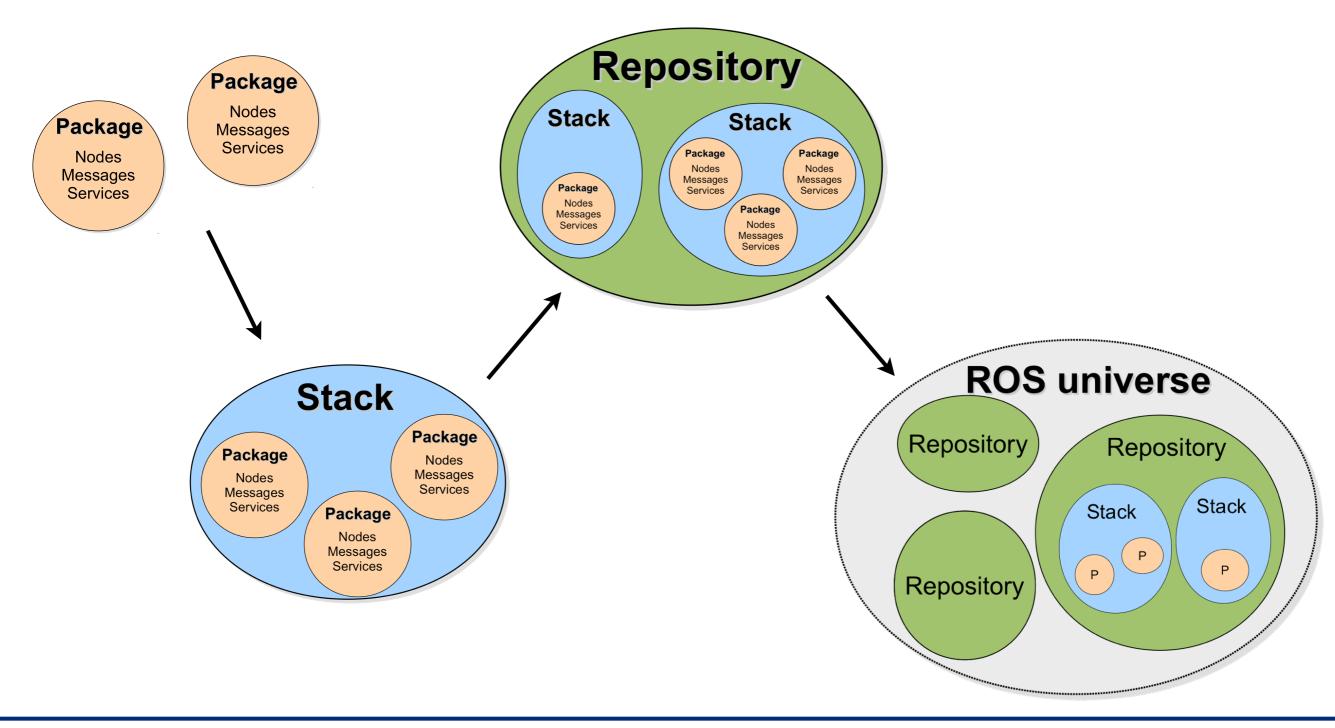
**Messages** describe the data format of the in/output of the nodes. For example, a door handle detection node gets camera images as input and spits out coordinates of detected door handles.

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







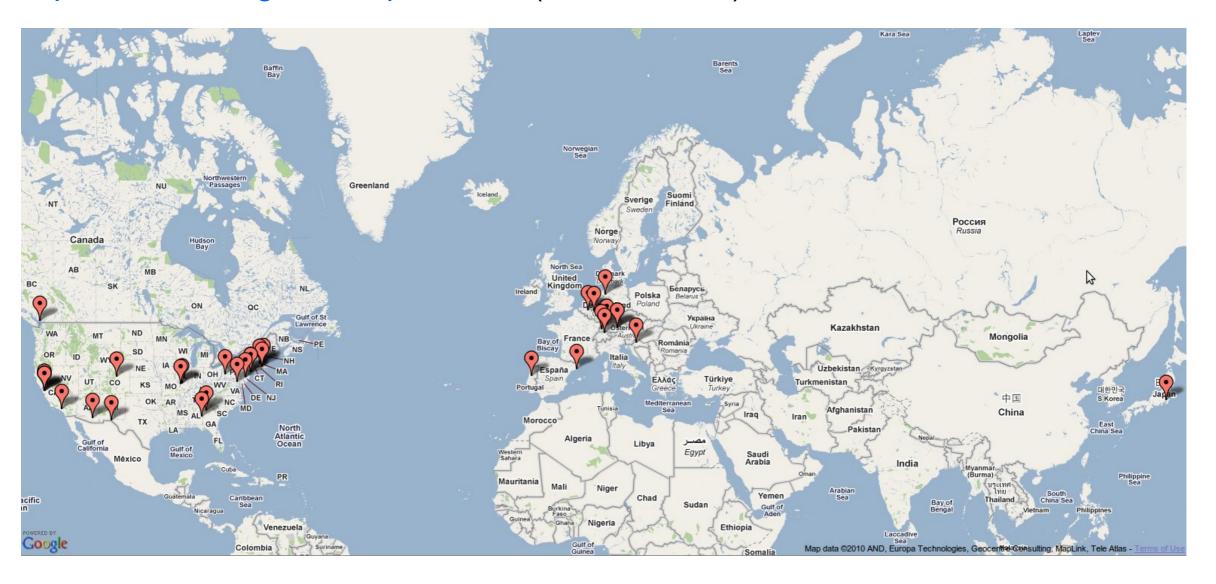






Collection of packages and stacks, hosted online Many repositories (>50): Stanford, CMU, TUM, Leuven, USC, Bosch, ...

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







#### **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





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
  </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${prefix}/include"</pre>
         lflags="-L${prefix}/lib -lmy lib"/>
  </export>
</package>
```





## ROS core

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

master

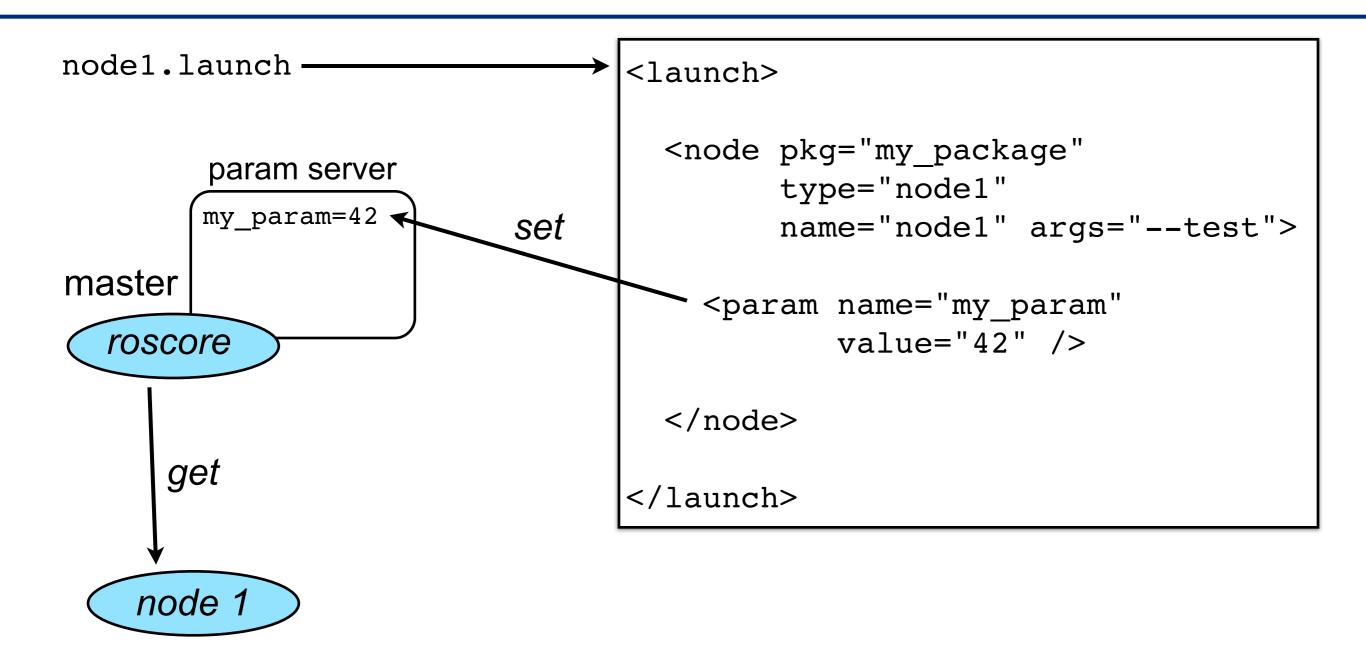


It provides naming and registration services to the rest of the **nodes** in the ROS system. It tracks publishers and subscribers to **topics** as well as **services**.

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









#### Problem:

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

# node 1 ? node 2

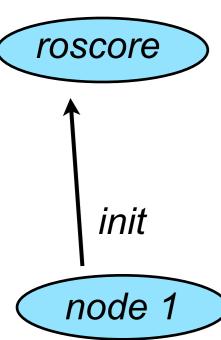




#### Problem:

Synchronization and message p { across multiple computer and/or

#### master





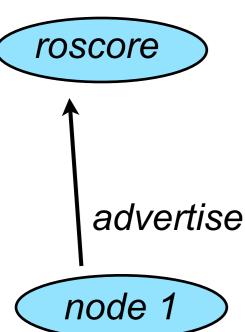


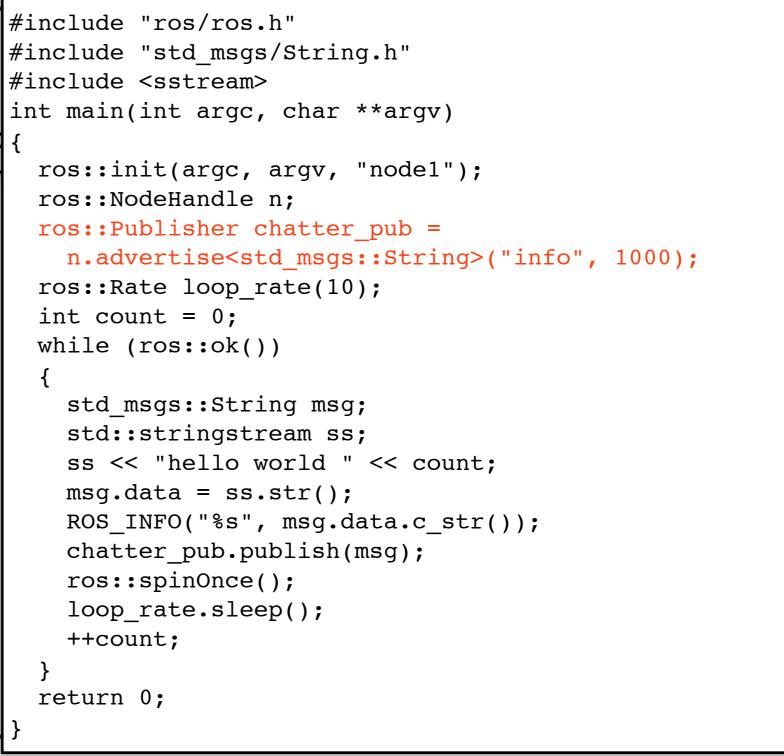


#### Problem:

Synchronization and message p { across multiple computer and/or

#### master



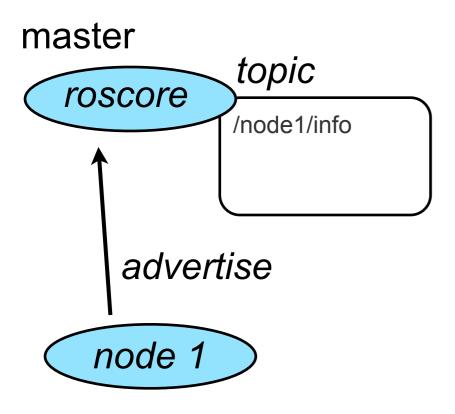






#### Problem:

Synchronization and message p { across multiple computer and/or



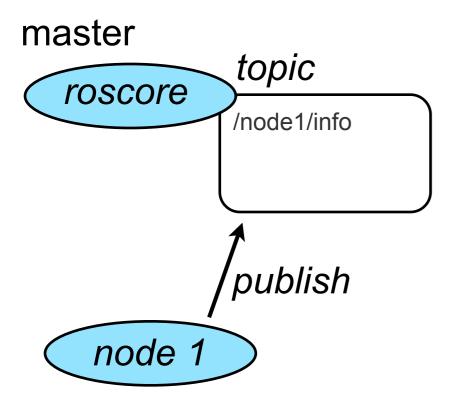
```
#include "ros/ros.h"
#include "std msgs/String.h"
#include <sstream>
int main(int argc, char **argv)
  ros::init(argc, argv, "node1");
  ros::NodeHandle n;
  ros::Publisher chatter pub =
    n.advertise<std msgs::String>("info", 1000);
  ros::Rate loop rate(10);
  int count = 0;
  while (ros::ok())
    std msqs::String msq;
    std::stringstream ss;
    ss << "hello world " << count;
    msq.data = ss.str();
    ROS INFO("%s", msq.data.c str());
    chatter pub.publish(msq);
    ros::spinOnce();
    loop rate.sleep();
    ++count;
  return 0;
```





#### Problem:

Synchronization and message p { across multiple computer and/or

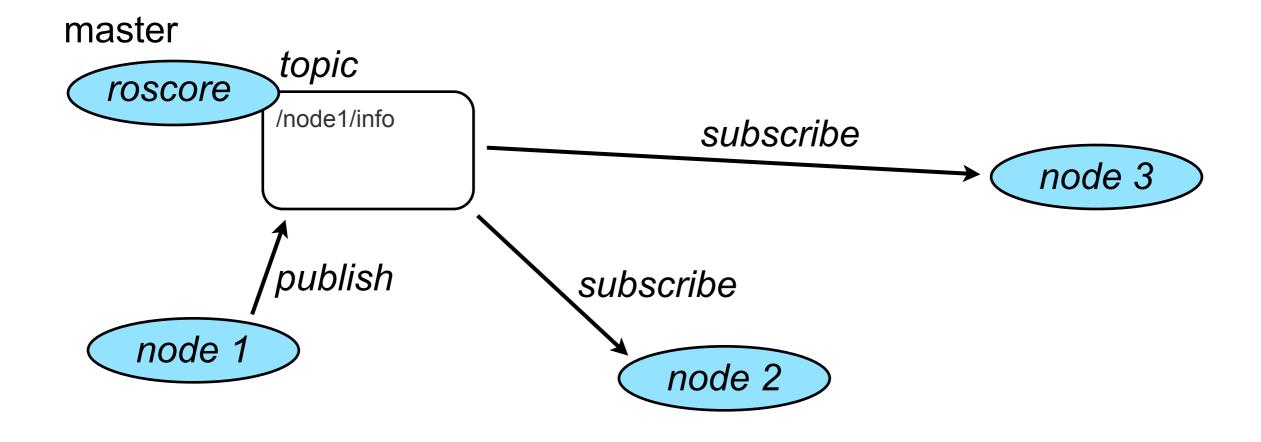


```
#include "ros/ros.h"
#include "std msgs/String.h"
#include <sstream>
int main(int argc, char **argv)
  ros::init(argc, argv, "node1");
  ros::NodeHandle n;
  ros::Publisher chatter pub =
    n.advertise<std msgs::String>("info", 1000);
  ros::Rate loop rate(10);
  int count = 0;
  while (ros::ok())
    std msqs::String msq;
    std::stringstream ss;
    ss << "hello world " << count;
    msq.data = ss.str();
    ROS INFO("%s", msq.data.c str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop rate.sleep();
    ++count;
  return 0;
```





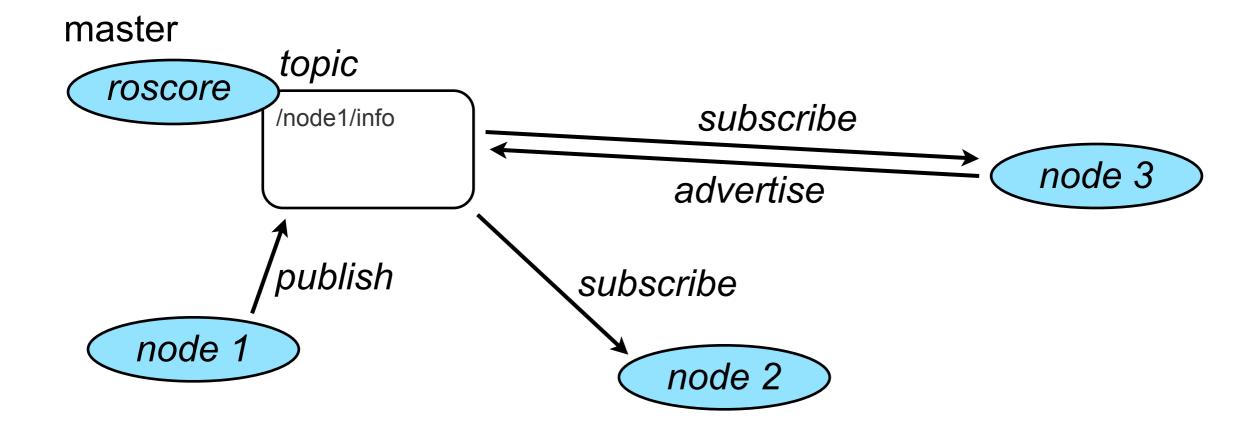
#### Problem:







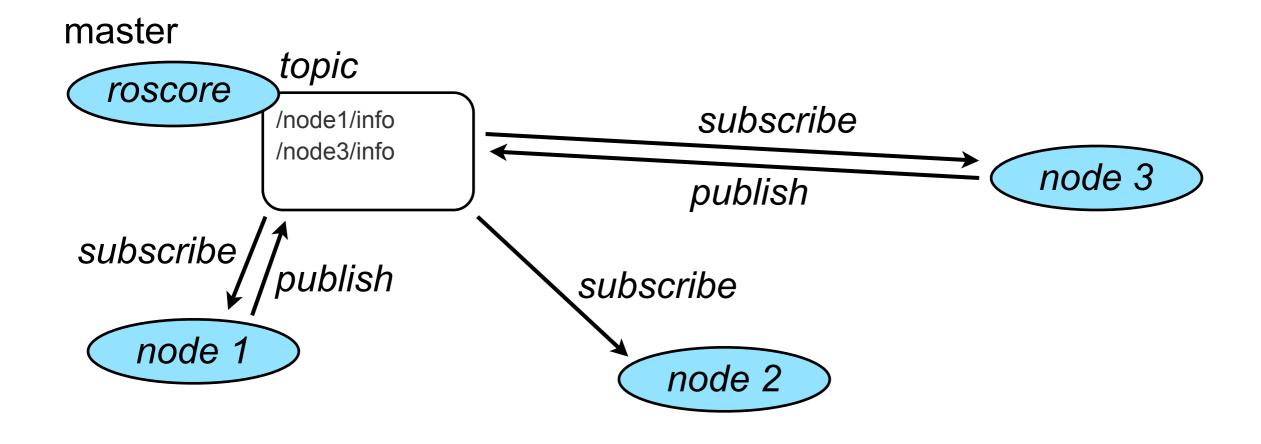
#### Problem:







#### Problem:

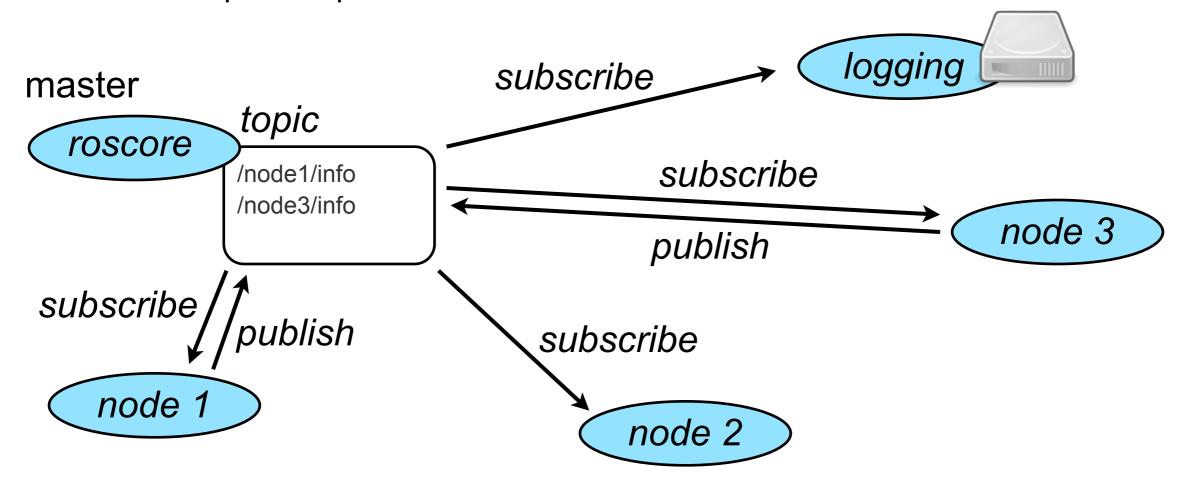






# ROS: logging

#### Problem:





# 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

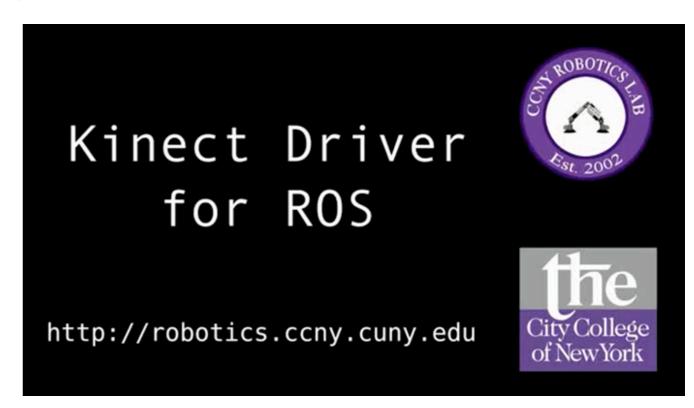
#### Problem:

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.









## ROS: robot descriptions

<robot name="test\_robot">
 <link name="link1" />
 <link name="link2" />
 <link name="link3" />

<link name="link4" />

</ioint>

</joint>

</joint>

</robot>

<parent link="link1"/>
<child link="link2"/>

<parent link="link1"/>

<parent link="link3"/>

<child link="kinect link"/>

<child link="link3"/>

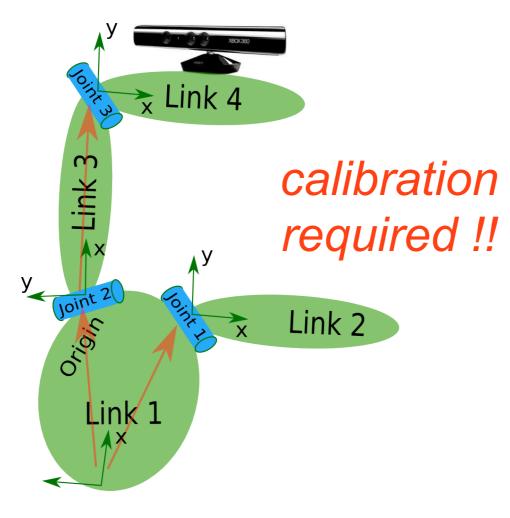
<joint name="joint1" type="continuous">

<joint name="joint2" type="continuous">

<joint name="joint3" type="continuous">

urdf: This package contains a C++ parser for the Unified Robot
Description Format (URDF), which is an XML format for representing a

robot model.



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.



PIVX

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



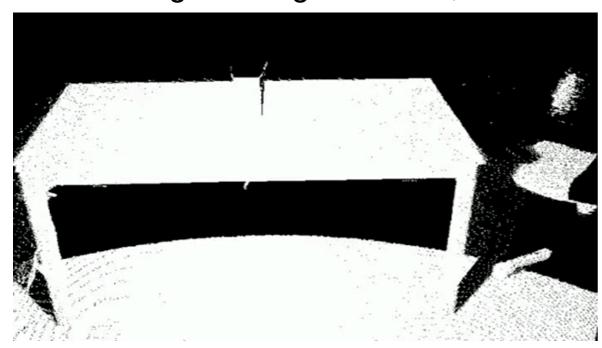


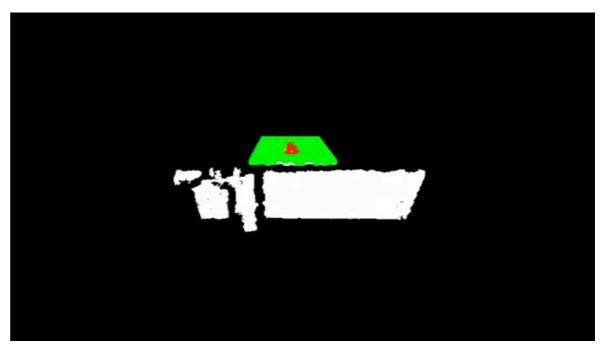
# ROS: 2D/3D perception

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

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

**PCL** - Point Cloud Library: 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.

**:::** stomp\_motion\_planner

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









navigation

task executive

visualization

simulation

perception

control

planning

data logging

message passing

device drivers

real-time capabilities

web browser

email client

OS

( 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



