Hands-on
Robot Operating System (ROS)

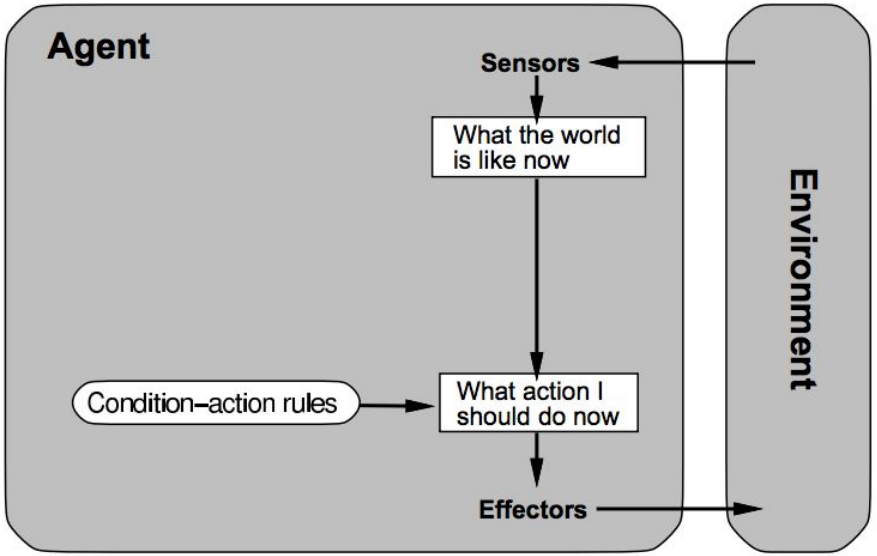
language technology for programming robots

November 2017

Mehdi Ghanimifard

Intelligent Agent Schema

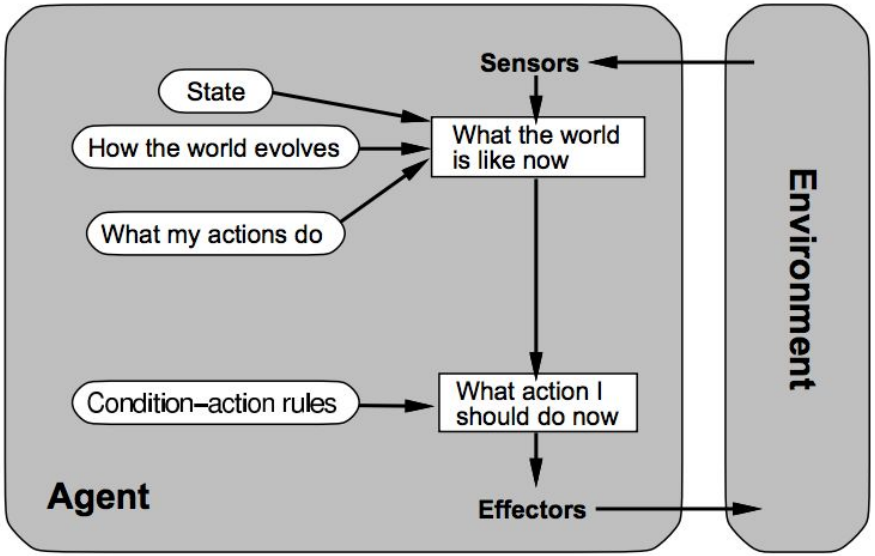
A simple reflex agent



Source:
Artificial Intelligence: A Modern Approach (1995)
by Stuart Russell and Peter Norvig. Prentice-Hall, Inc.

Intelligent Agent Schema

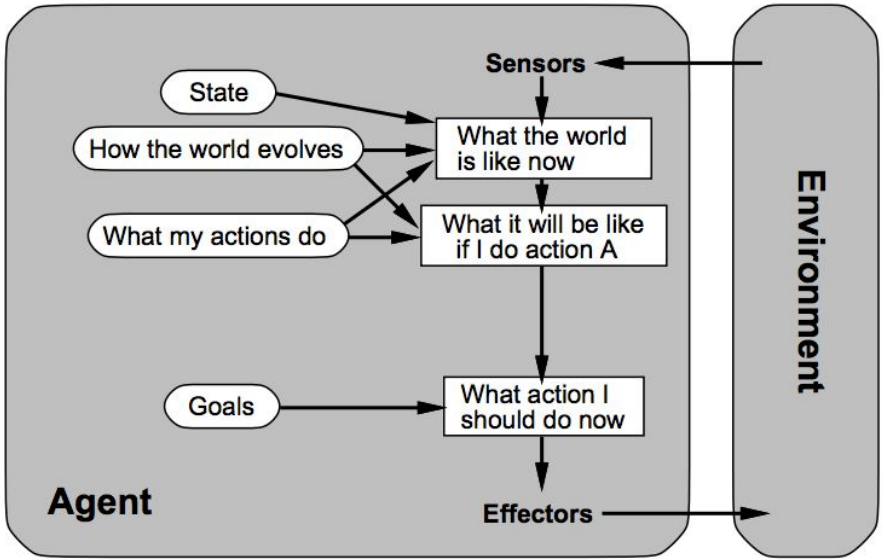
A reflex agent with internal state



Source:
Artificial Intelligence: A Modern Approach (1995)
by Stuart Russell and Peter Norvig. Prentice-Hall, Inc.

Intelligent Agent Schema

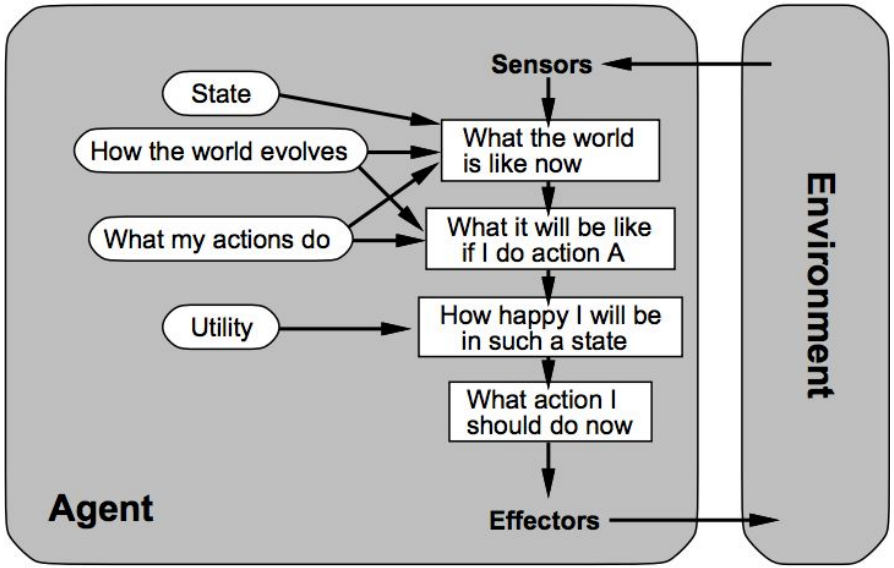
An agent with explicit goals



Source:
Artificial Intelligence: A Modern Approach (1995)
by Stuart Russell and Peter Norvig. Prentice-Hall, Inc.

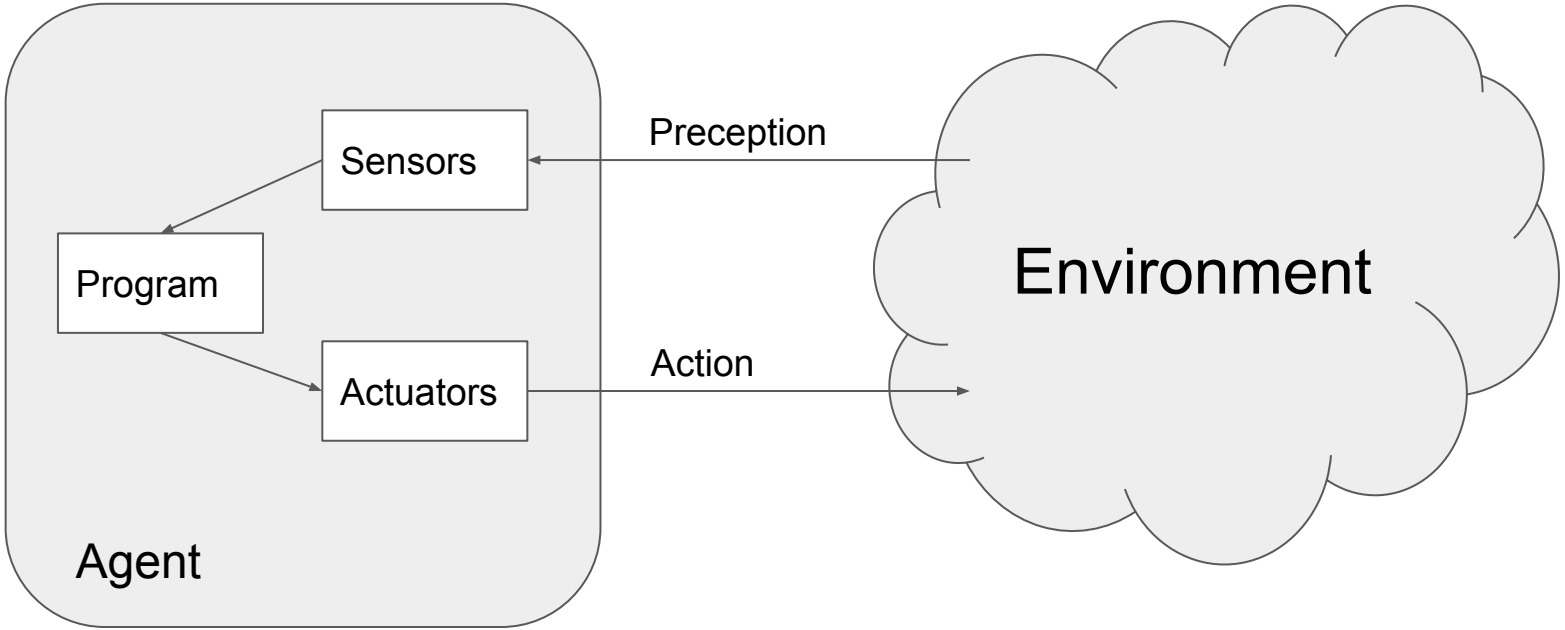
Intelligent Agent Schema

A complete utility-based agent

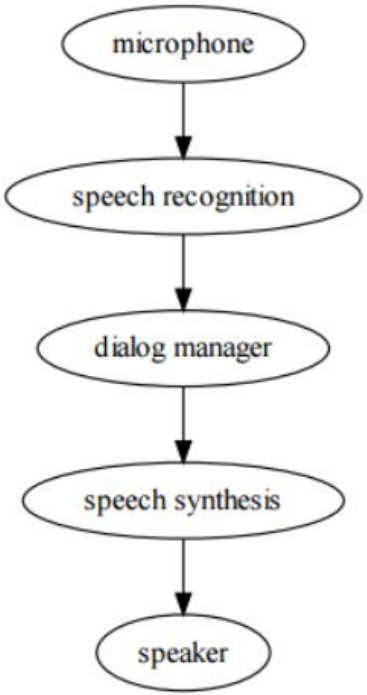


Source:
Artificial Intelligence: A Modern Approach (1995)
by Stuart Russell and Peter Norvig. Prentice-Hall, Inc.

Generalization over Agent Schema



Maybe the implementation of this?



How Robotics Research Keeps...

Re-Inventing the Wheel

First, someone publishes...

...and they write code that barely works but lets them publish...

...a paper with a proof-of-concept robot.

This prompts another lab to try to build on this result...

...but they can't get any details on the software used to make it work...

...and all the code used by previous lab members is a mess.

So, a grandiose plan is formed to write a new software API...

...and countless sleepless nights are spent writing code from scratch.

But inevitably, time runs out...

What is ROS?

ROS = **R**obot **O**perating **S**ystem

- ROS is a platform for robot software.
- Goal: advance open-source robotics
- Meta-operating system = It’s built on top of the OS (Linux, Mac, Windows, …)

How fast can you start robot programing?

- You need a computer (preferably Ubuntu)
- Knowledge of Python
- Any robot hardware compatible with ROS: http://wiki.ros.org/Robots
- Robotics by nature is multi-disciplinary, usually you need some knowledge in other fields. However, with ROS, you don’t need to invent the wheel! If something is available open-source, you can use it easily.

Papers using RGB-D data

Malinowski, M., & Fritz, M. (2014). A multi-world approach to question answering about real-world scenes based on uncertain input. In *Advances in Neural Information Processing Systems* (pp. 1682-1690).
https://www.mpi-inf.mpg.de/departments/computer-vision-and-multimodal-computing/research/vision-and-language/visual-turing-challenge/

Matuszek, C., FitzGerald, N., Zettlemoyer, L., Bo, L., & Fox, D. (2012). A joint model of language and perception for grounded attribute learning. *arXiv preprint arXiv:1206.6423*.
https://rgbd-dataset.cs.washington.edu/

Krishnamurthy, J., & Kollar, T. (2013). Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association for Computational Linguistics*, *1*, 193-206.

Going beyond the image recognition?

TensorFlow ™ GET STARTED TUTORIALS HOW TO MOBILE API RESOURCES ABOUT

Fork me on GitHub

The following instructions assume you installed TensorFlow from a PIP package and that your terminal resides in the TensorFlow root directory.

```
cd tensorflow/models/image/imagenet
python classify_image.py
```

The above command will classify a supplied image of a panda bear.

If the model runs correctly, the script will produce the following output:

```
giant panda, panda, panda bear, coon bear, Ailuropoda melanoleuca (score = 0.88
indri, indris, Indri indri, Indri brevicaudatus (score = 0.00878)
lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens (score = 0.
custard apple (score = 0.00149)
earthstar (score = 0.00127)
```

Why ROS?

- Because of these problems:
  - Sequential programming on asynchronous environment
  - Complexity in a big software!
  - Abstraction for specific robot hardware.

Problem 1: Sequential Programming.

```
robot = Robot()
do {
     image = robot.get_image_from_camera()
     belief = robot.update_belief(image)
     path = robot.find_the_path(belief)
     goal = robot.go_to_the_goal(path)
} while (goal)
```

- During the **`go_to_the_goal`**, an obstacle stops the program; now what?
- How could camera use online **`image`** to avoid collision?

Problem 2: Complexity in big software

- How to organize a big software with several different pieces:
  - camera, laser, infrared, ultrasonic,motor
  - face recognition, image processing,
  - dialogue system, logic,
  - etc.

Solution for complexity: Separating processes

- Organize all tasks as a network of separated processes
- Each process runs separately over the network
- Each process can communicate with others

Solution asynchronous events: *Callbacks*

```
def image_callback(image):
     # … do something with image -> belief
     pub.publish(belief)


if __name__ == '__main__':
     # …
     pub = rospy.Publisher("/mybot/belief", String, latch=True)
     rospy.Subscriber("/camera/rgb/image_color", Image, image_callback)
     rospy.spin()
```

Solution asynchronous events: *Callbacks*

```
def belief_callback(belief):
    # … do something with belief -> path -> commands
    pub.publish(commands)

if __name__ == '__main__':
    # …
    pub = rospy.Publisher("/motor/control", Twist, latch=True)
    rospy.Subscriber("/mybot/belief", String, belief_callback)
    rospy.spin()
```

Problem 3: Abstraction for specific robot hardware

- Programming for a robot software without making abstraction over hardware leads to hardware-dependent software.

Solution 3: Message interfaces

- A hardware abstraction has a standard message interface.
- e.g. a camera driver or generally an image sensor, publish messages in type of **`sensor_msgs/Image`** and publish it on **`/camera/rgb/image_color`**
  By changing camera hardware the other parts of the software stays intact.

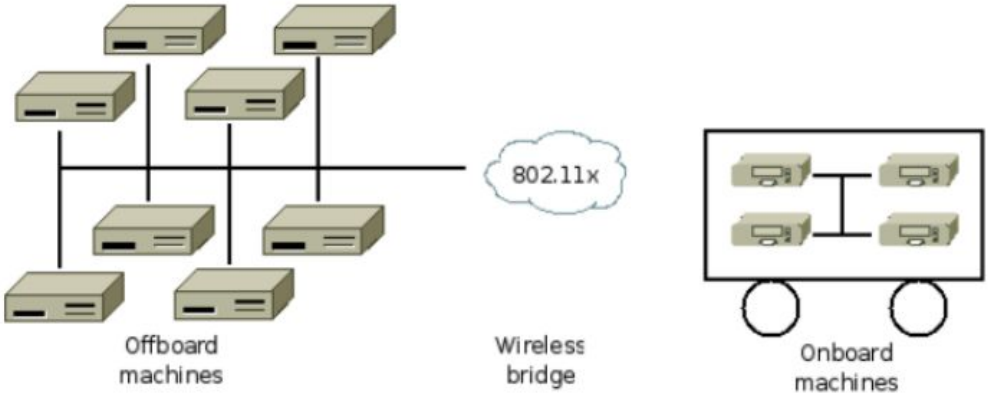
Design of ROS

- **Peer-to-peer:** with several processes, but doesn’t rely on central server
- **Tools-based:** microkernel design + several small tools
- **Multi-lingual:** C++, python, LISP, …
- **Thin:** ROS re-uses code from other open-source projects, drivers, navigation system, simulators, vision algorithms, and etc.
- **Free and Open-Source:** The ROS itself is open source but its licence lets businesses to produce modules with different licence for it.

Source:

Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." *ICRA workshop on open source software*. Vol. 3. No. 3.2. 2009.

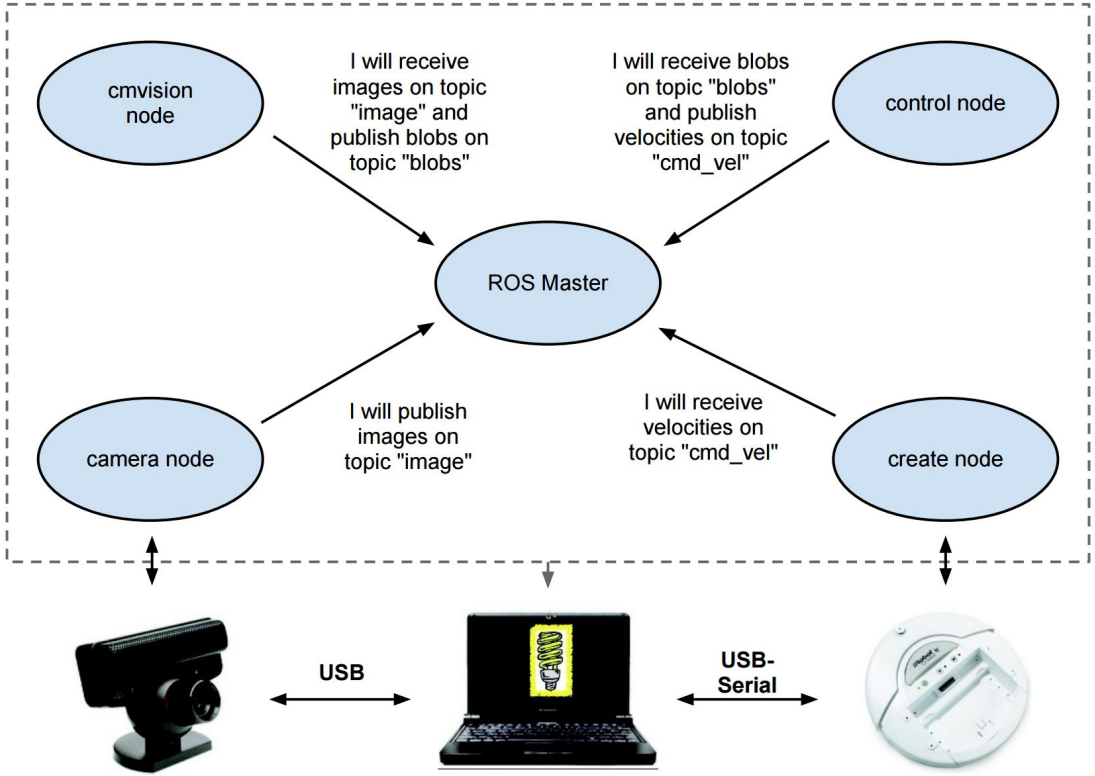
Design of ROS



Source:

Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." *ICRA workshop on open source software*. Vol. 3. No. 3.2. 2009.

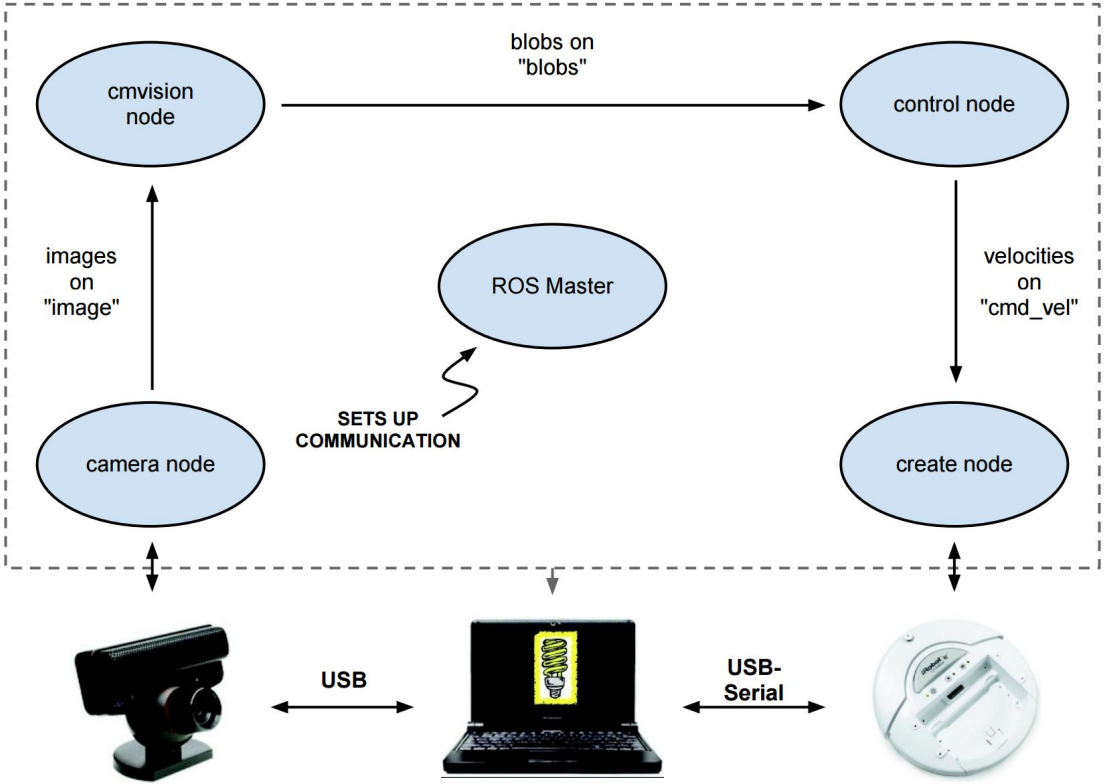
How ROS works



[adapted from slide by Chad Jenkins]

From slide by Todd Hester (University of Texas - CS378)

How ROS works

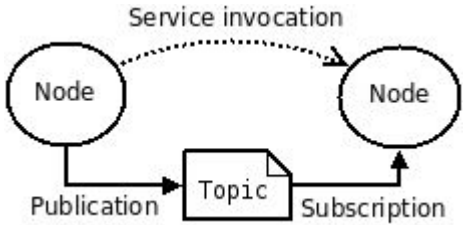


[adapted from slide by Chad Jenkins]

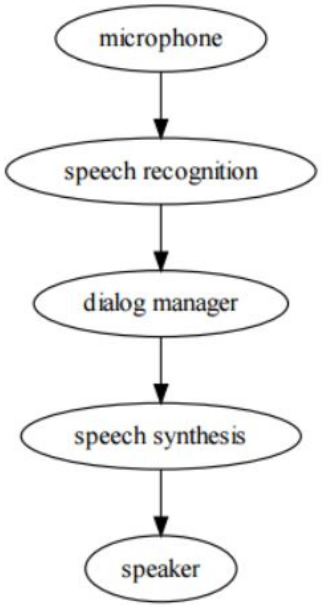
From slide by Todd Hester (University of Texas - CS378)

Architecture of ROS robots

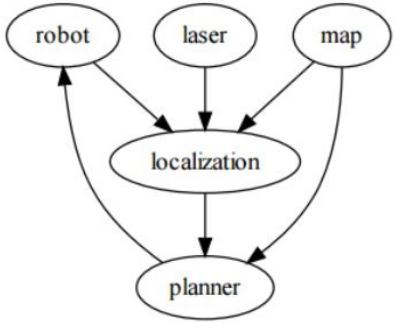
- ROS is organized as a network of ***nodes (processes)***.
- **ROS master (`roscore`)** stores all network data (addresses).
- **Nodes** on startup register themselves with master.
- Each ***node*** performs a single task.
- Nodes coordinate with each other through ***topics*** or ***services***.
- *Nodes* can ***subscribe*** or ***publish*** to a ***topic*** with a specific **message type**.
- ***topics*** are suitable for asynchronous transactions with *broadcast*
- ***services*** are suitable for synchronous transactions



Examples: network architecture of nodes



Simple communication pipeline



a navigation system

Nodes,Topics, Publish/Subscribe, Message type

- Node 1:

```
commander = rospy.Publisher("/myrobot/commands", String)
r = rospy.Rate(10) # 10hz

while not rospy.is_shutdown():
     commander.publish("go forward")
     r.sleep()
```

- Node 2:

```
def cmd_callback(cmd_string):
     # do something

if __name__ == '__main__':
     rospy.Subscriber("/myrobot/commands", String, cmd_callback)
     rospy.spin()
```

Live Coding!