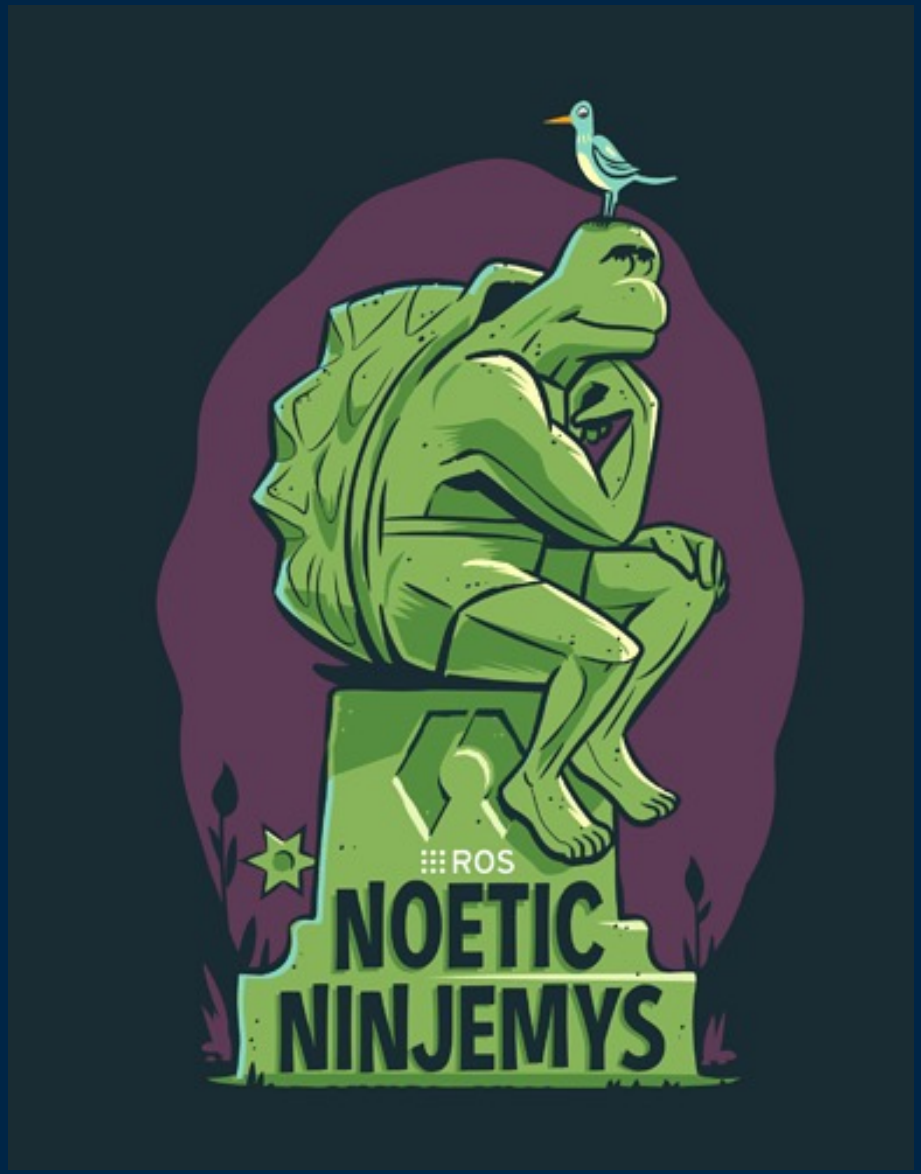
# INTRO TO ROS






Slides adapted from Peter Ahrendt's

# LAB RULES &GUIDELINES

## USING the Equipment:

1. At your first class, pick a robot and sd card, and stick to that for the rest of the semester.
2. At the end of each class do:
   a. Remove any dangling wires from the robot, and place them in the wire box. (use female to female cables for the PI)
   b. Screw any shaky/dangling parts in place.
   c. Switch the robot off properly (switch in the arduino board).
   d. Remove battery and put to charge.
   e. Uncouple sensors, except for the Lidar on top of the robot, and put them in the corresponding box.
   f. Place the robot in the shelf. **Be gentle with the robots**.

## REPORTING Problems:

Software

1. Note down the robot and sd card
2. Note down the steps you took before the problem manifested
3. Try to **replicate** using a second sd card. If we cannot replicate, we cannot fix!

Hardware

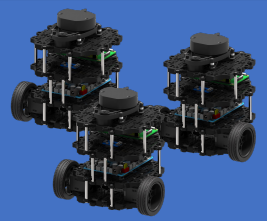
1. Note down the robot and sd card
2. Note down what changes you made to the code or couplings before the problem manifested
3. Note down which part is affected

Send us an email with the report.

## GOOD to Know:

Remember the robots are not perfect. If you observe that the lidar of, e.g., robot #10 does not read for specific angles, note down those angles, and work around it.

As you get to know your robot, fill in a document with such information. This will be useful to all when we debug, and to the next generation of students using them.




CONTACT:

Søren Møller Dath: smd@ece.au.dk
Mirgita Frasheri: mirgita.frasheri@ece.au.dk
Jalil Boudjadar: jalil@ece.au.dk
Andriy Sarabakha andriy@ece.au.dk

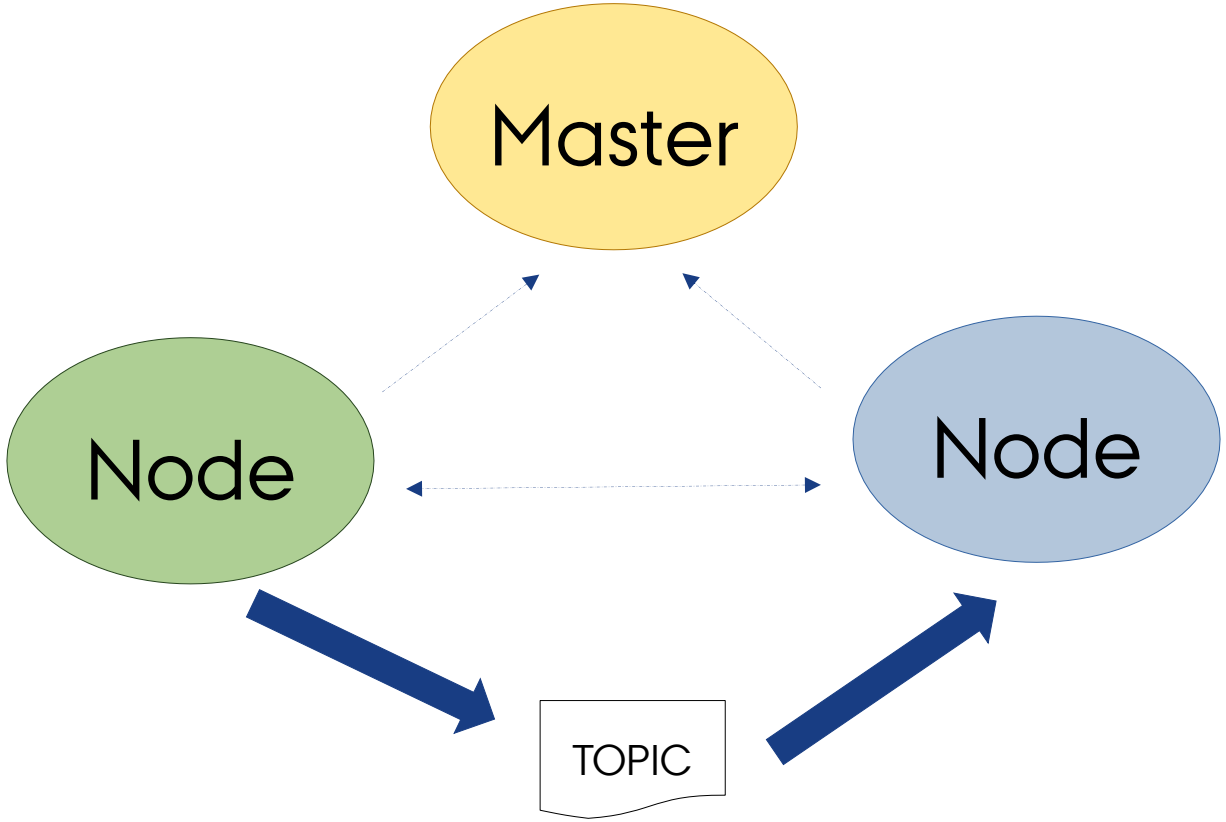
> Think about all the software you have made before, especially the ones you write for your CS courses; you seldom run two or more programs simultaneously, or make programs that talk to each other, or even consider other people making new programs and hoping to let their programs take an advantage of yours, right? Concurrency, inter-communication and extensibility, these are the things you don't often do; and you will find it very hard to do by only using the Raw APIs from Operating System directly.
>
> Lakshay Garg

> Think about all the software you have made before, especially the ones you write for your CS courses; you seldom run two or more programs simultaneously, or make programs that talk to each other, or even consider other people making new programs and hoping to let their programs take an advantage of yours, right? Concurrency, inter-communication and extensibility, these are the things you don't often do; and you will find it very hard to do by only using the Raw APIs from Operating System directly.
>
> Lakshay Garg

# ROS

- **R**obot **O**perating **S**ystem
- FOSS under BSD
- Hardware and network abstraction
- Low-level device control
- Process Communications Infrastructure
- Big community
- Implementation of commonly-used robot features
- Tools for inspection, simulation, development and debugging
- Package management

# CORE CONCEPTS



# CORE CONCEPTS

- Nodes: A node is an executable that uses ROS to communicate with other nodes.
- Messages: ROS data type used when subscribing or publishing to a topic.
- Topics: Nodes can publish messages to a topic as well as subscribe to a topic to receive messages.
- Master: Name service for ROS (i.e. helps nodes find each other)
- rosout: ROS equivalent of stdout/stderr
- roscore: Master + rosout + parameter server

# NODES

- ROS consists of a number of processes, called **nodes**
- Single-purposed executable programs, e.g. sensor driver(s), actuator driver(s), mapper, planner, UI, etc.
- potentially on a number of different hosts,

  connected at runtime in a **peer-to-peer**

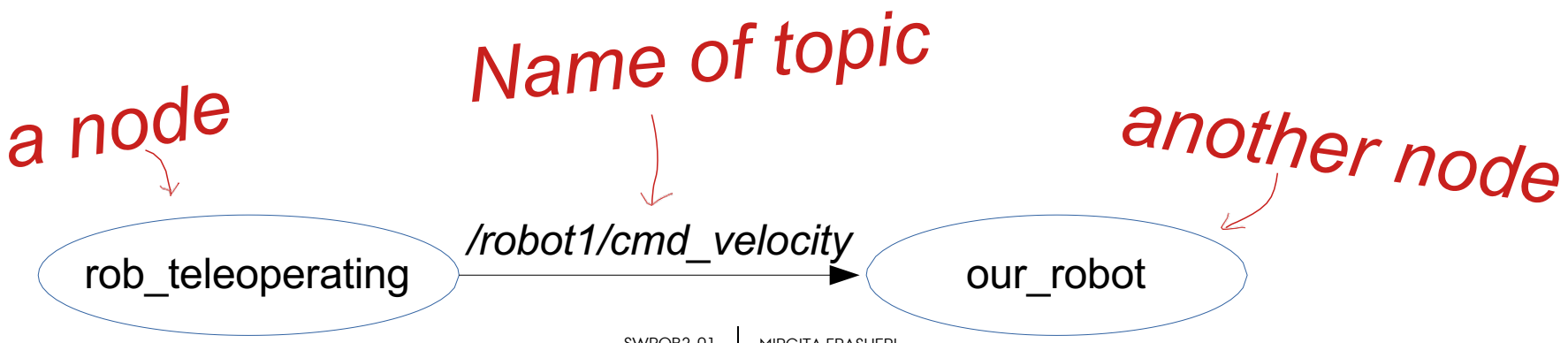
  topology.
  - The peer-to-peer topology requires some sort of lookup mechanism to allow processes to find each other at run-time. We call this the name service, or master

# NODES

- Individually compiled, executed, and managed
- Nodes are written with the use of a ROS client
- Library
  - Libraries let you write ROS nodes, publish and subscribe to topics, write and call services, and use the Parameter Server.
  - C++, python, and LISP are officially supported, but libraries for other languages exist.
    - roscpp = C++ client library
    - rospy = python client library

# TOPICS AND MESSAGES

- **Topics** are the message paths
- Nodes communicate with each other by publishing **messages** to topics
- Publish/Subscribe model
- E.g.:
  - provide sensor readings
  - provide actuator states / robot feedback



# TOPICS AND MESSAGES

Messages are strictly-typed data, and their types are defined in msg files

e.g.

```
fieldtype1  fieldname1
fieldtype2  fieldname2
fieldtype3  fieldname3
```

for example:

```
int32  x
int32  y
```

Supported field types

| |
|---|
| bool (1) |
| int8 |
| uint8 |
| int16 |
| uint16 |
| int32 |
| uint32 |
| int64 |
| uint64 |
| float32 |
| float64 |
| string |
| time |
| duration |

# TOPICS AND MESSAGES

You can look up the fields in a message by typing in the terminal:

**$ rosmsg show <message>**

```
$ rosmsg show sensor_msgs/CameraInfo
Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
RegionOfInterest roi
  uint32 x_offset
  uint32 y_offset
  uint32 height
  uint32 width
float64[5] D
float64[9] K
float64[9] R
float64[12] P
```

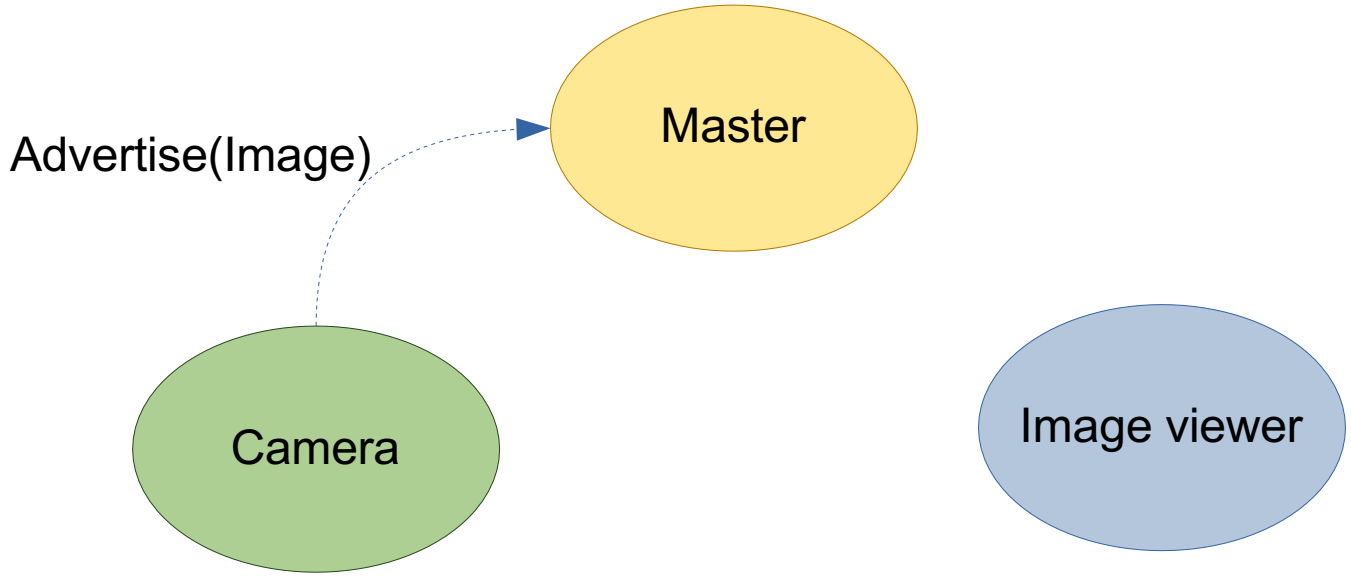
# SERVICES

- Synchronous inter-node transactions / remote procedure call (RPC)
- Think of a service as function in a standard programming language: You make a request and – depending on the type of function – receive a response.
- A typical service could be used for trigger functionality / behavior:
  - Reset
  - Get current location
  - Like topics, services have an associated service type

# MASTER

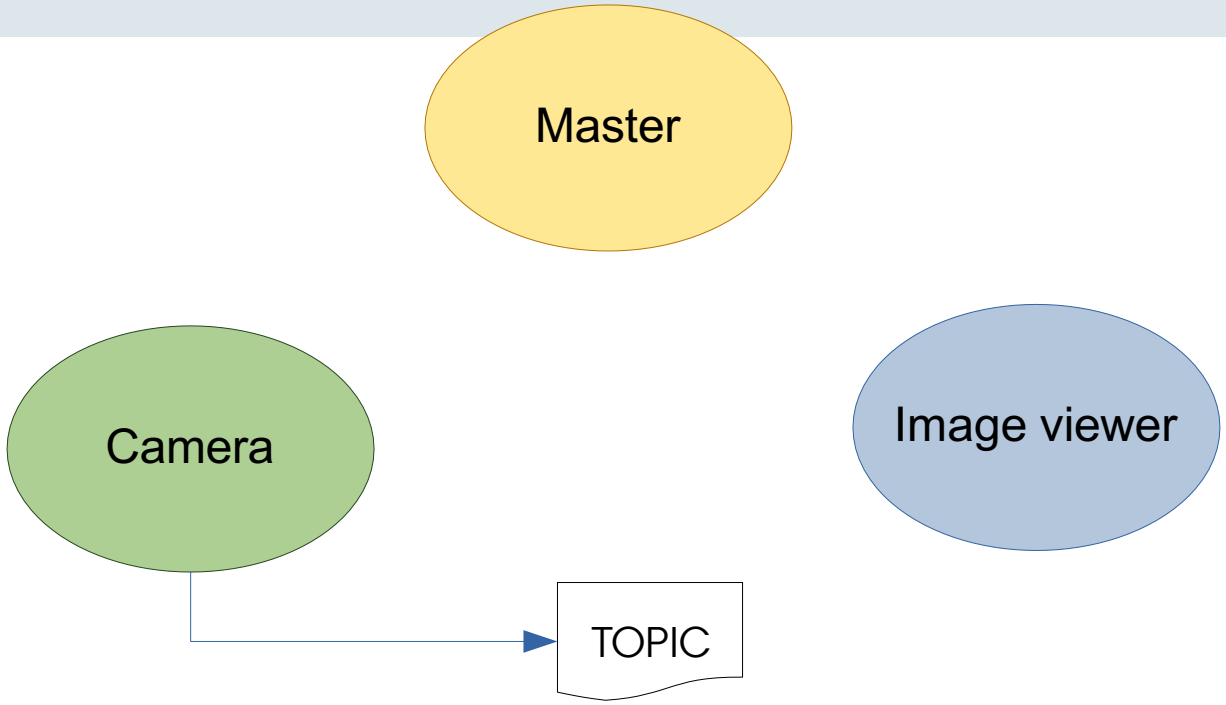
- Lookup mechanism, name service
- The Master tracks publishers and subscribers to topics as well as services.
- Enable individual ROS nodes to locate one another.
  - Once these nodes have located each other they communicate with each other peer-to-peer.

# MASTER NAME SERVICE EXAMPLE



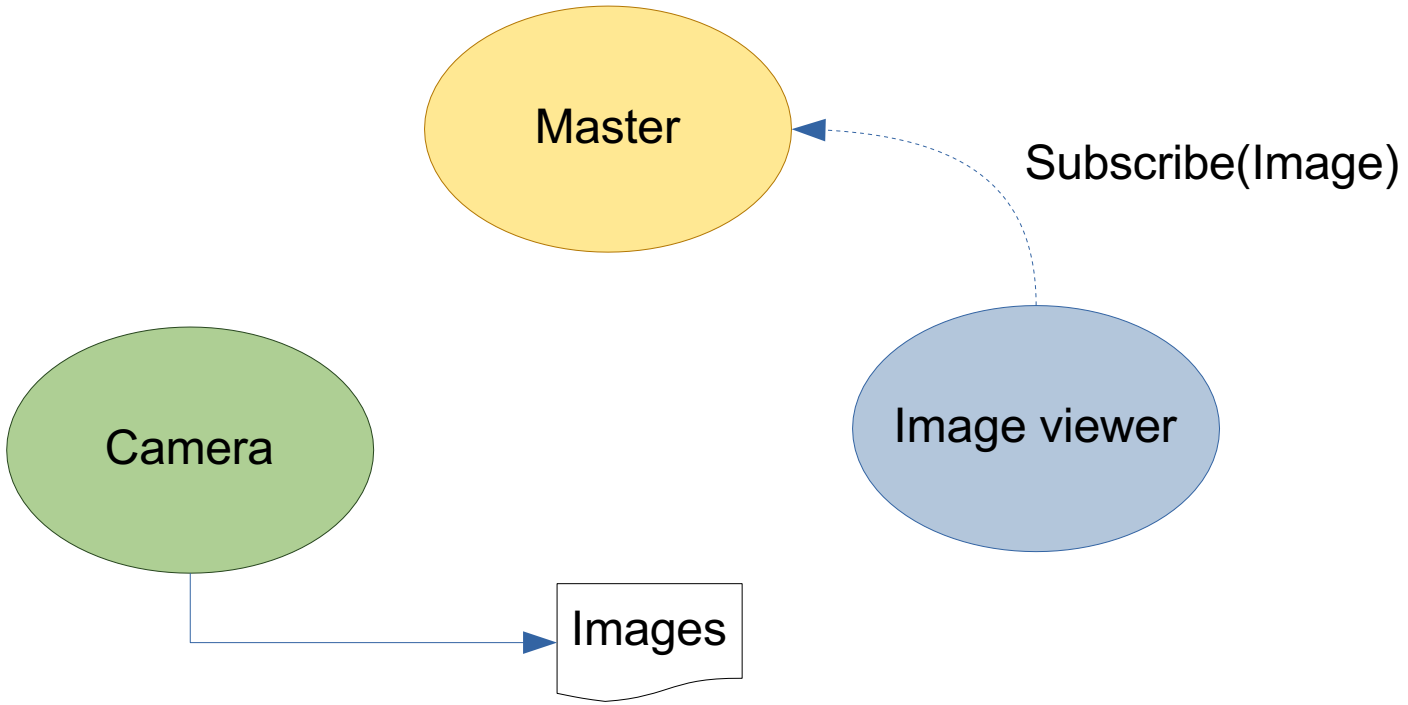

Camera notifying the master that it wants to publish images on the topic "images". Camera publishes images to the "images" topic, but nobody is subscribing to that topic yet so no data is actually sent.

# MASTER NAME SERVICE EXAMPLE



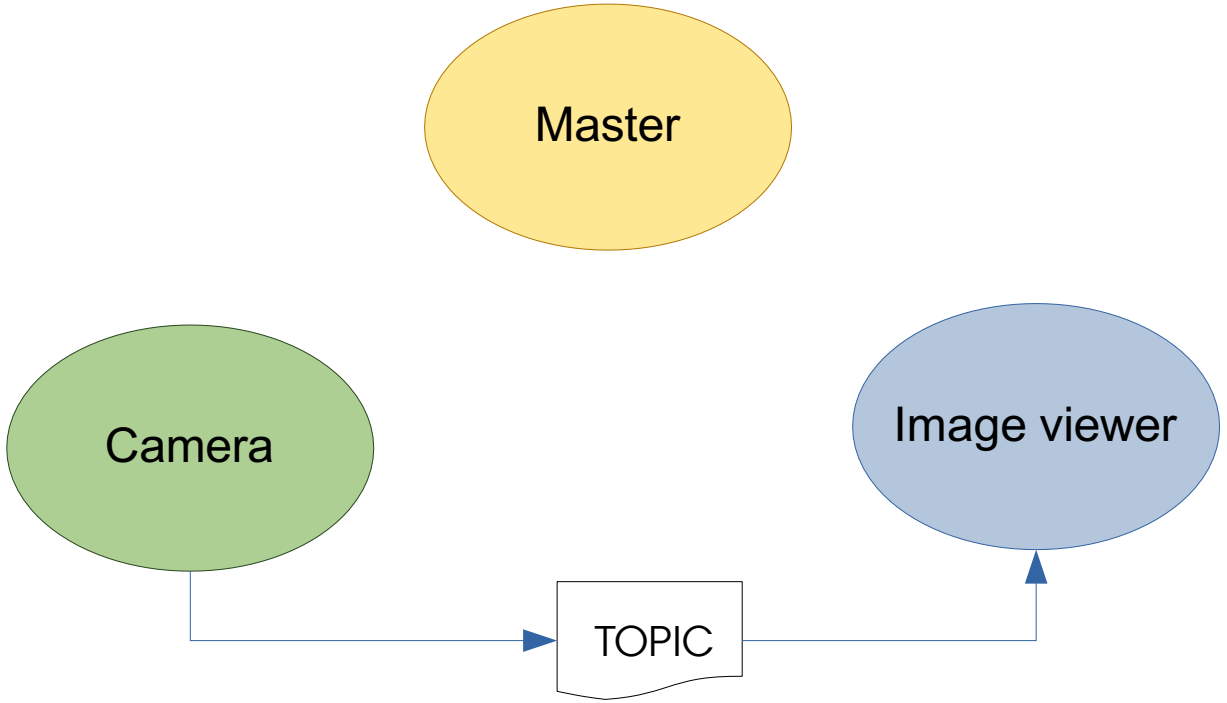

Now, camera publishes images to the "images" topic, but nobody is subscribing to that topic yet so no data is actually sent.

# MASTER NAME SERVICE EXAMPLE




Now, Image_viewer wants to subscribe to the topic "images" to see if there are any images.

# MASTER NAME SERVICE EXAMPLE



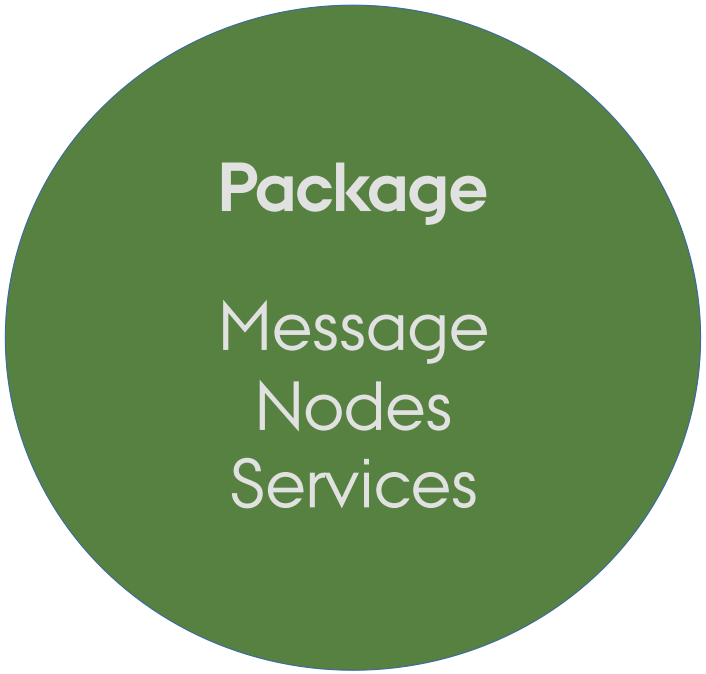

Now that the topic "images" has both a publisher and a subscriber, the master node notifies Camera and Image_viewer about each other's existence so that they can start transferring images to one another

# PACKAGES

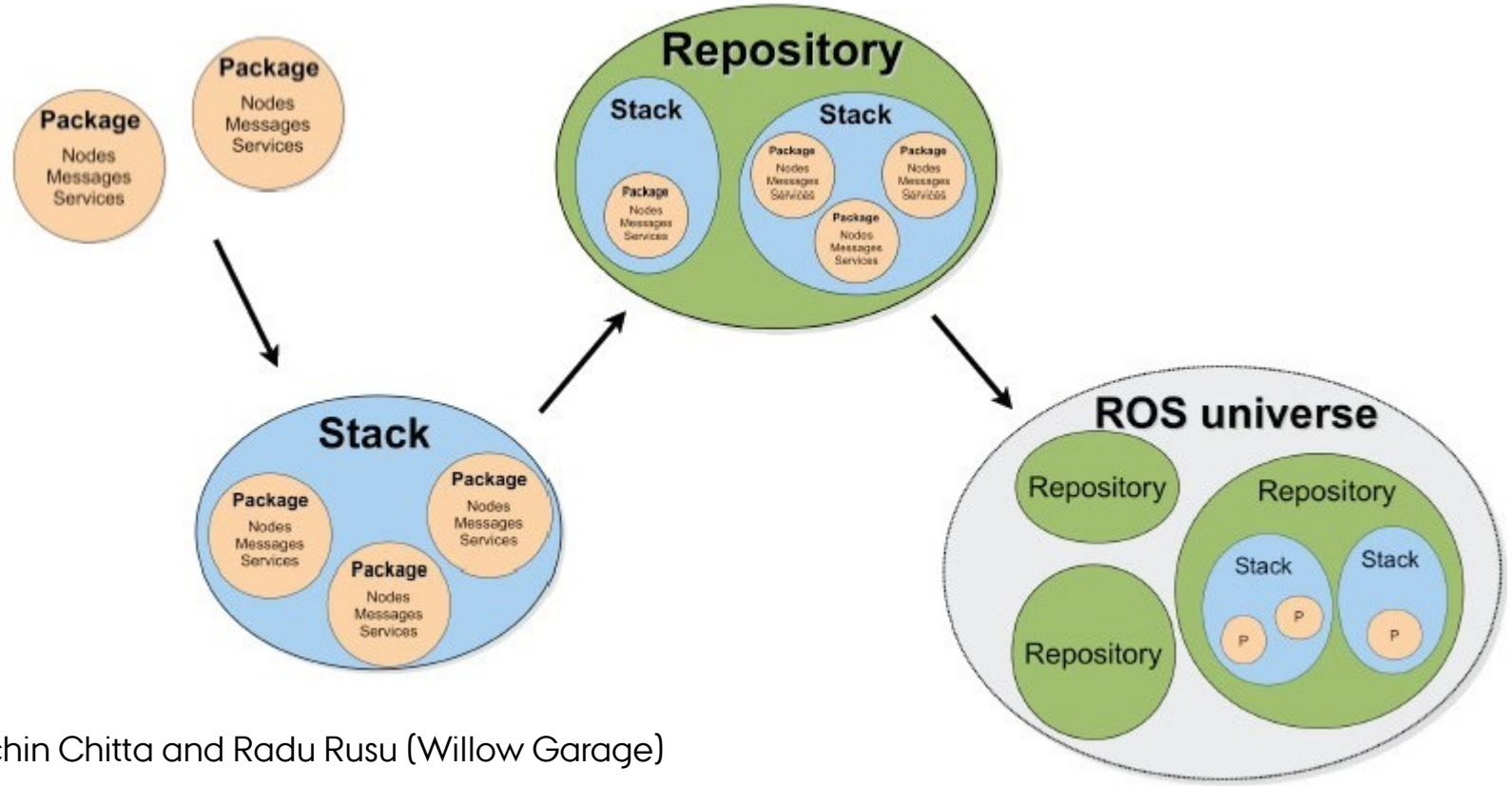
Software in ROS is organized in packages.

A package contains one or more nodes and provides a ROS interface

Tons of packages exist



# PACKAGE REPOS



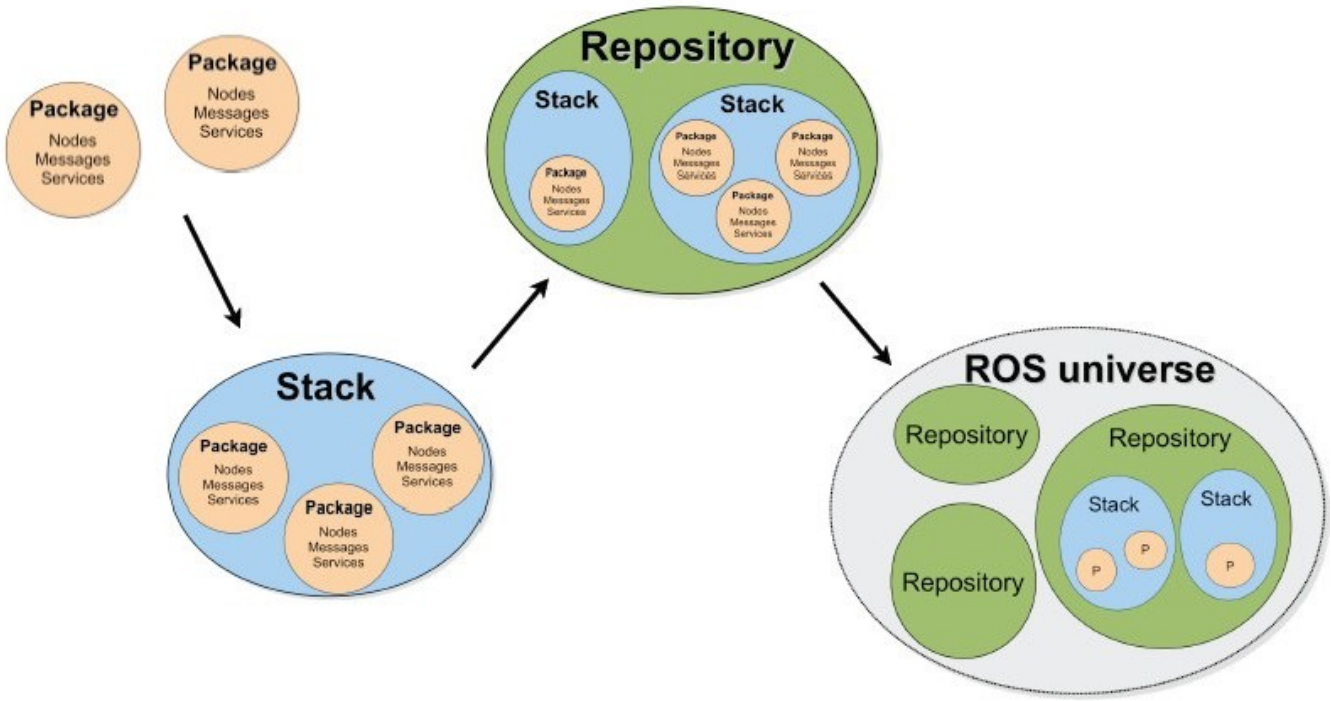

Ref.: Sachin Chitta and Radu Rusu (Willow Garage)

# PACKAGE REPOS

Collection of packages and stacks Many repositories: Stanford, CMU, Leuven, USC, …

Most of them hosted in Git

http://wiki.ros.org/RecommendedRepositoryUsage/CommonGitHubOrganizations



# BASIC ROS COMMANDS

**Roscore** – is a collection of nodes and programs that are prerequisite of a ROS-based system

- master
- parameter server
- Rosout

You must have a roscore running in order for ROS nodes to communicate

| Command | |
|---|---|
| `$roscore` | Starts a roscore |

# BASIC ROS COMMANDS

**rosnode** – Displays debugging information about ROS nodes, including publications, subscriptions and connections

Commands:

| Command | |
|---|---|
| $rosnode list | List active nodes |
| $rosnode ping | Test connectivity to node |
| $rosnode info | Print information about a node |
| $rosnode kill | Kill a running node |
| $rosnode machine | List nodes running on a particular machine |

# BASIC ROS COMMANDS

**rosrun** – allows you to run an executable in an arbitrary package without having to cd (or roscd) there first

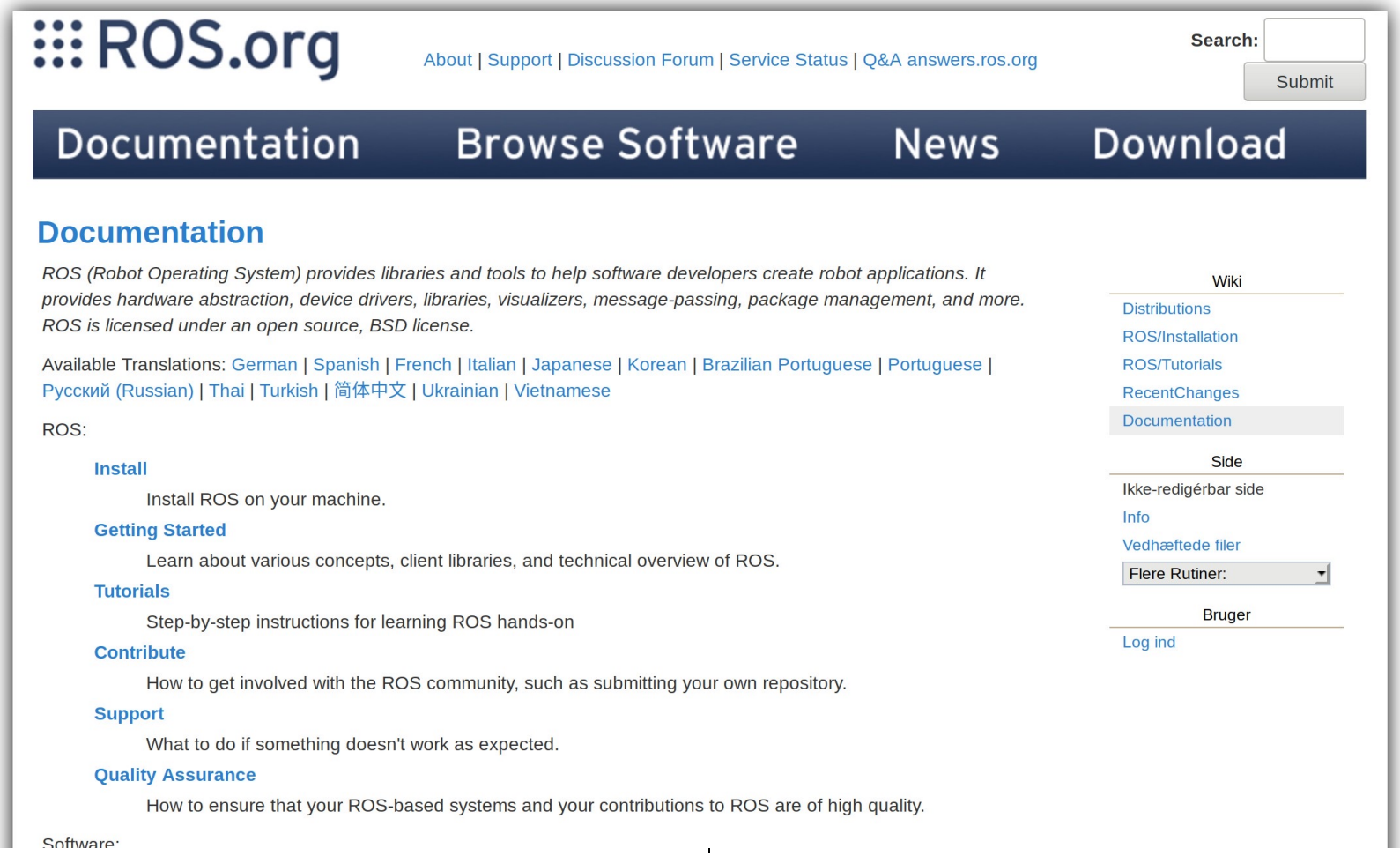
Usage: `$rosrun package executable`

Example: run turtlesim

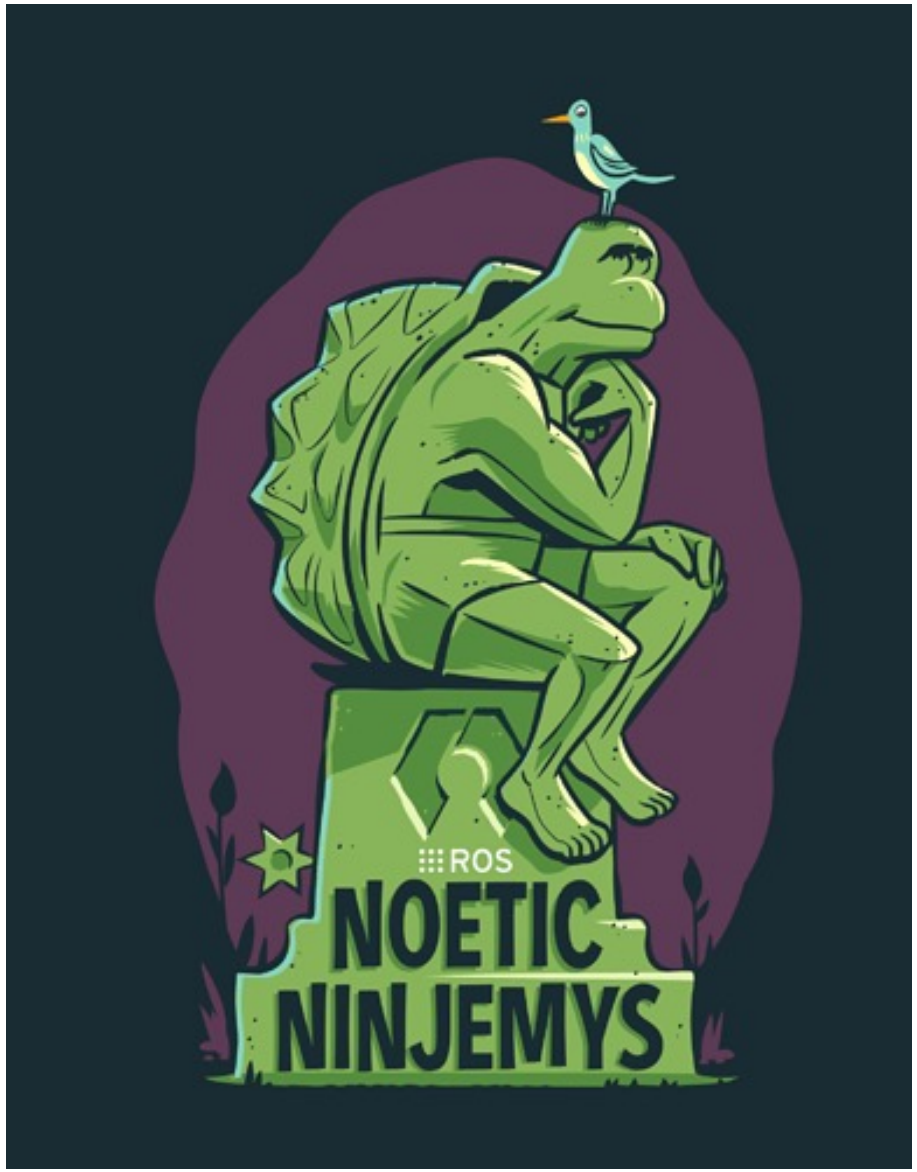
- `$rosrun turtlesim turtlesim_node`

# HTTP://WIKI.ROS.ORG/

The ROS Wiki is well-written and the go-to place for ROS support and tutorials



# ROS VERSION




# EXERCISES

Form groups of 3-4 persons (ideally multi- disciplinary)

Connect the Turtlebot (ROS) to Matlab

Try out simple control commands from Matlab to the Turtlebot

mirgita.frasheri@ece.au.dk

AARHUS
UNIVERSITET