Fachhochschule
Dortmund
University of Applied Sciences and Arts

we
focus
on
students

ROS-Workshop

A brief introduction to ROS

by Matthias Domnik

Fachhochschule Dortmund
University of Applied Sciences and Arts

Structure

we focus on students

1. Introduction
2. Basic concept and architecture
3. The fundamental ROS-elements
4. Basic communication
5. ROS launch
6. Recording in ROS
7. Learning outcome
8. Looking ahead

:::ROS

Introduction

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

… but it is not an operating system in the original sense.

"ROS is an ***open-source*****,** ***meta-operating system*** *for your robot.*

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. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio."

From http://wiki.ros.org/ROS/Introduction, „What is ROS?"

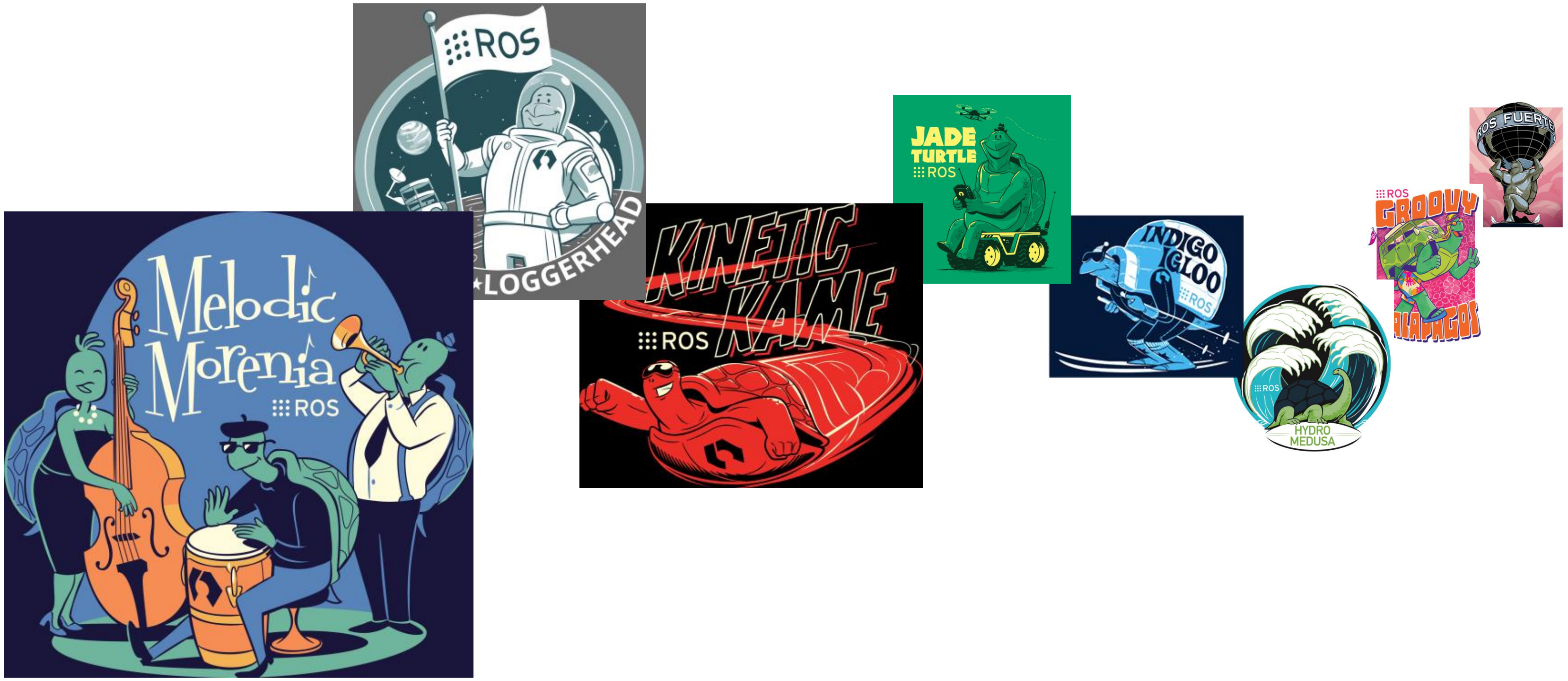
ROS is…

- a large set of drivers, which lets you interact with sensors and actuator.
- a large collection of algorithms for nearly all fields of robotic.
- a computational infrastructure that makes it possible to span a distributed network over different machines.
- a wide ecosystem around the software, for example the wiki and the question-and-answer page.

For further information see: http://www.ros.org/core-components/

Introduction

Logos from
http://wiki.ros.org/Distributions



The distribution we will use is called **Melodic Morenia**.

→ recommended for the latest Ubuntu **LTS (18.04)**

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Structure

1. Introduction
2. **Basic concept and architecture**
3. The fundamental ROS-elements
4. Basic communication
5. ROS launch
6. Recording in ROS
7. Learning outcome
8. Looking ahead

from: http://wiki.ros.org/Distributions

Basic concept and architecture

The basic concepts of ROS are:

- **Thin**: ROS is designed to be as thin as possible
- **Open-source software**: Most ROS related software is

  open-source and free to use.
- **Language independence**: Python, C++, and Lisp

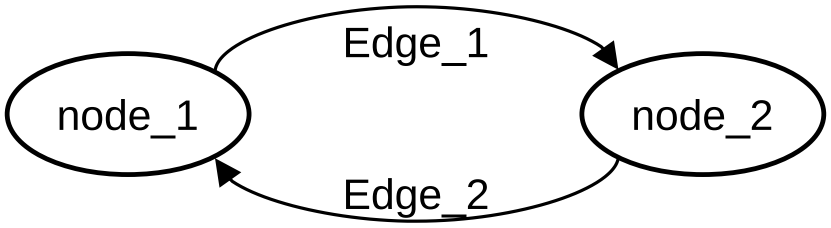
  (experimental libraries in Java and Lua).
- **Scaling**: ROS is appropriate for large runtime systems and

  for large development processes.

For further information see: http://wiki.ros.org/ROS/Introduction

Basic concept and architecture

- ROS is based on the peer-to-peer (**P2P**) network.
  → different programs communicate directly over defined API
- The network is comparable with a directed graph
  → The programs are the nodes
  → and the communication takes place over directed edges



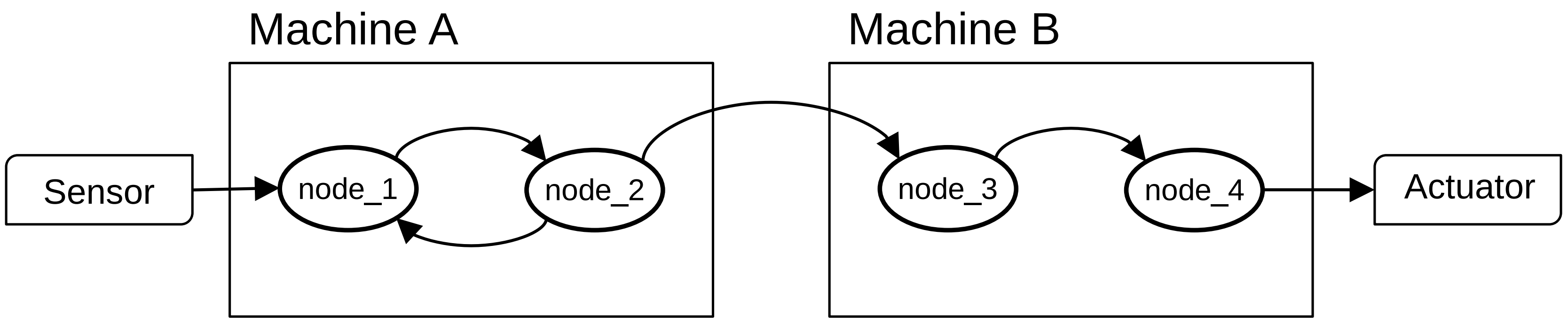
Example of a simple directed graph

Fachhochschule Dortmund
University of Applied Sciences and Arts

Basic concept and architecture

we focus on students

The P2P-concept makes it possible to run a distributed network on several machines.



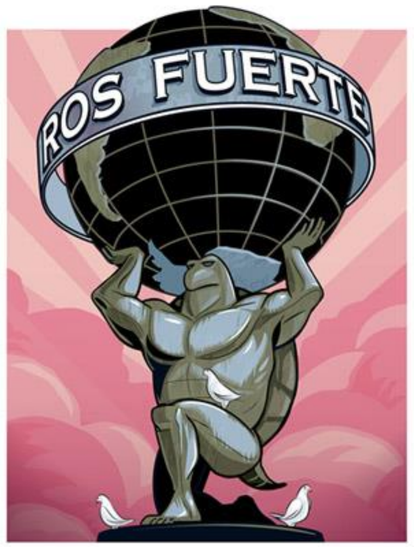
This is an important feature, because a second machine is often needed. An usual operating case takes places between a separate computer and the robot.

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Structure

we
focus
on
students

1. Introduction
2. Basic concept and architecture
3. The fundamental ROS-elements
4. Basic communication
5. ROS launch
6. Recording in ROS
7. Learning outcome
8. Looking ahead



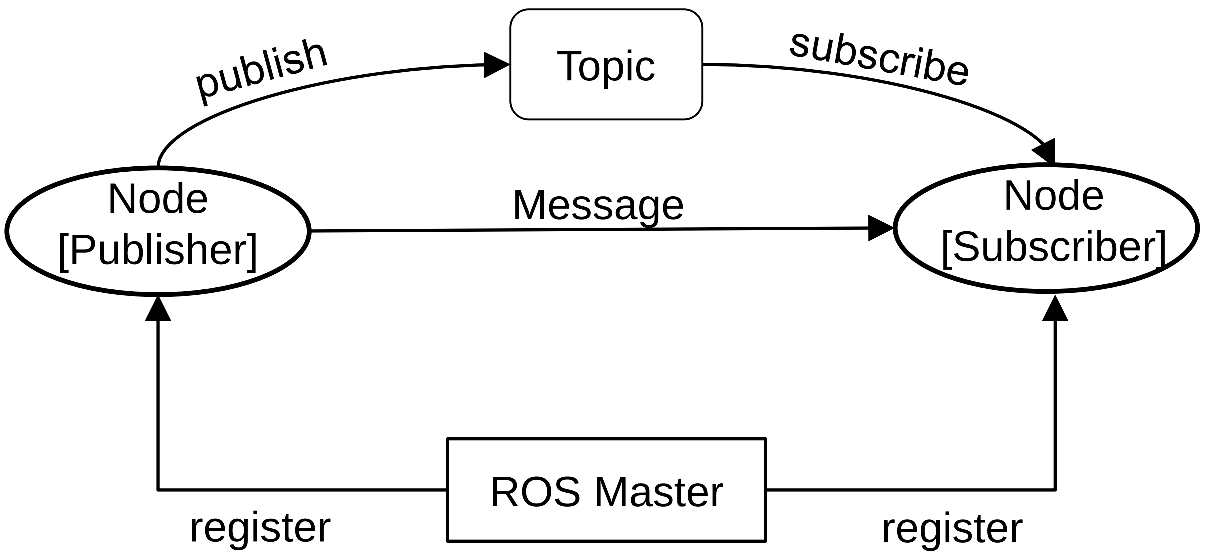
from: http://wiki.ros.org/Distributions

Fachhochschule Dortmund
University of Applied Sciences and Arts

The fundamental elements
A simple ROS-network

we focus on students

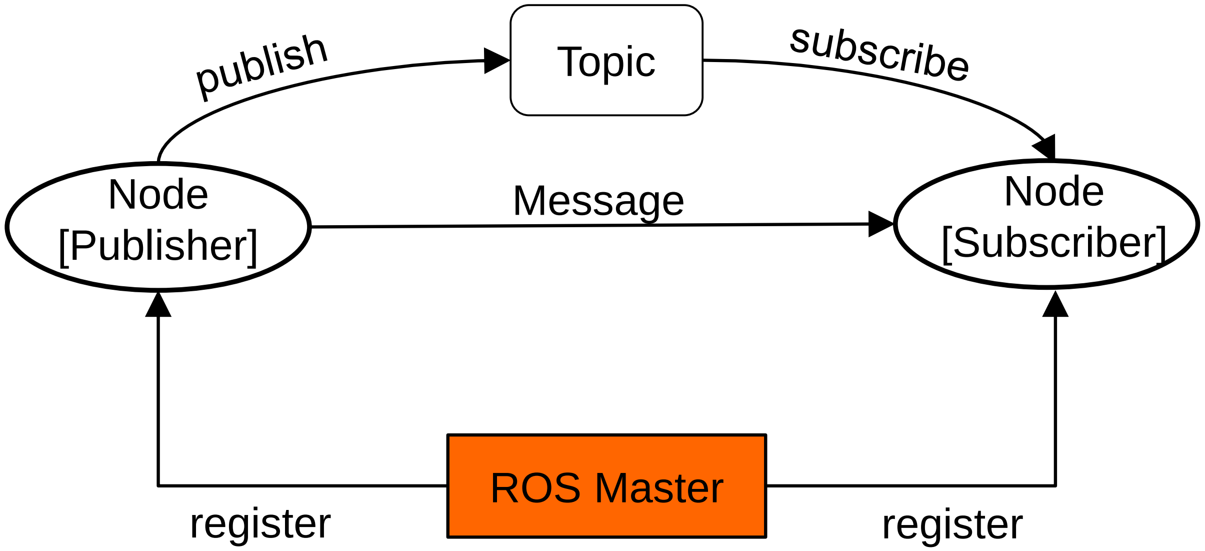
This chapter gives an overview of the fundamental ROS-elements.



A simple ROS-network

Fachhochschule Dortmund
University of Applied Sciences and Arts

The fundamental elements
ROS-Master



A simple ROS-network

Fachhochschule Dortmund
University of Applied Sciences and Arts

The fundamental elements ROS-Master

- The Master is in charge of the node-registration.

- The nodes register at start up with the master.
  - → Tells the master to which topic the node is publishing.
  - → and to what topic it would like to subscribe.

- The master acts a bit like a DNS-Server.

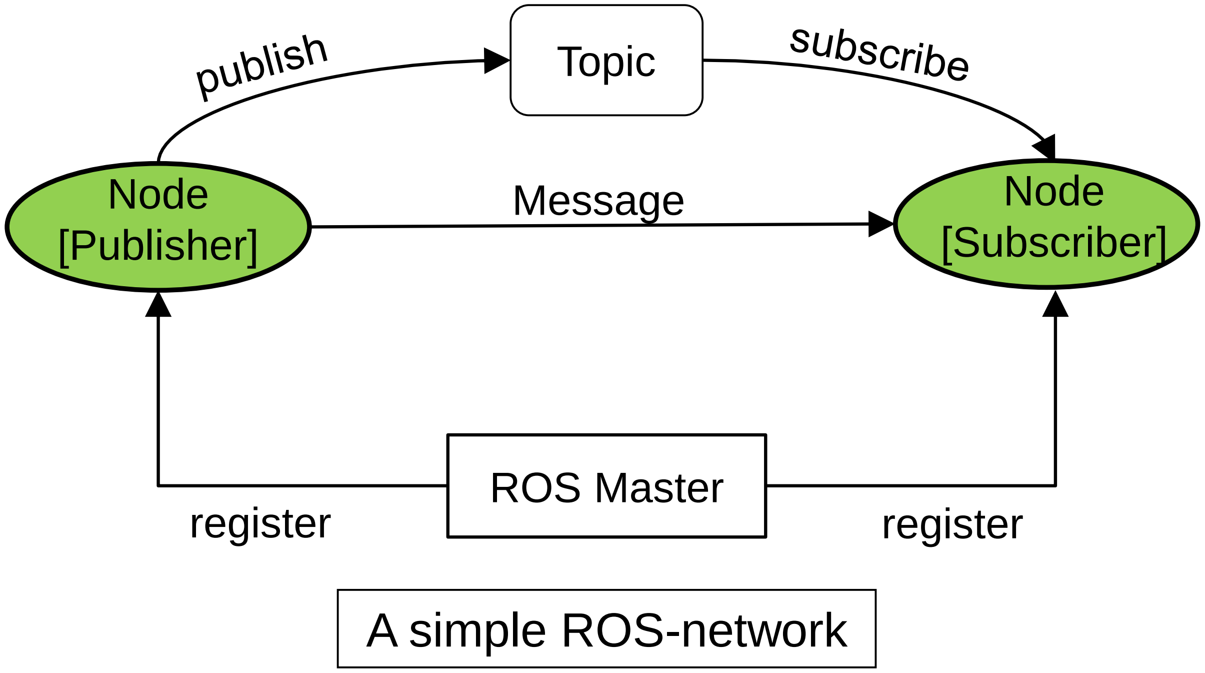
The console-command to start the master:

```
$ roscore
```

Fachhochschule Dortmund
University of Applied Sciences and Arts

The fundamental elements ROS-node

we focus on students



A simple ROS-network

The fundamental elements
ROS-node

- *“A node is a process that performs computation”* (http://wiki.ros.org/Nodes)
- A robot-system will usually consists of many nodes communicating with each other.
- The idea is to create nodes for single-purpose to gain a maximum reusability.

The console-command to start a node:

```
$ rosrun <package-name> <node-name>
```

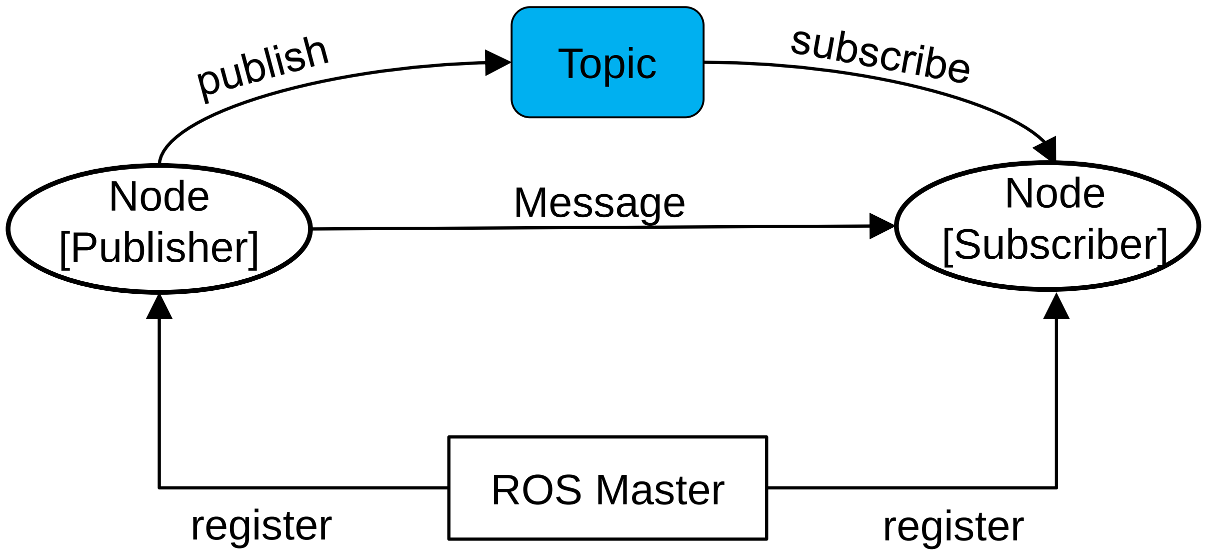
See all active nodes:

```
$ rosnode list
```

Get further infos about a node:

```
$ rosnode info <node-name>
```

The fundamental elements
ROS-topic



A simple ROS-network

The fundamental elements
ROS-topic

- Nodes communicate over *topics*.
  - → Nodes can publish or subscribe to a topic (or topics)
  - → The usual case is one publisher for any number of subscribers
- A topic is a stream of *messages*.

Lists all availible topics:

```
$ rostopic list
```

Print the content of a topic:

```
$ rostopic echo /<topic-name>
```

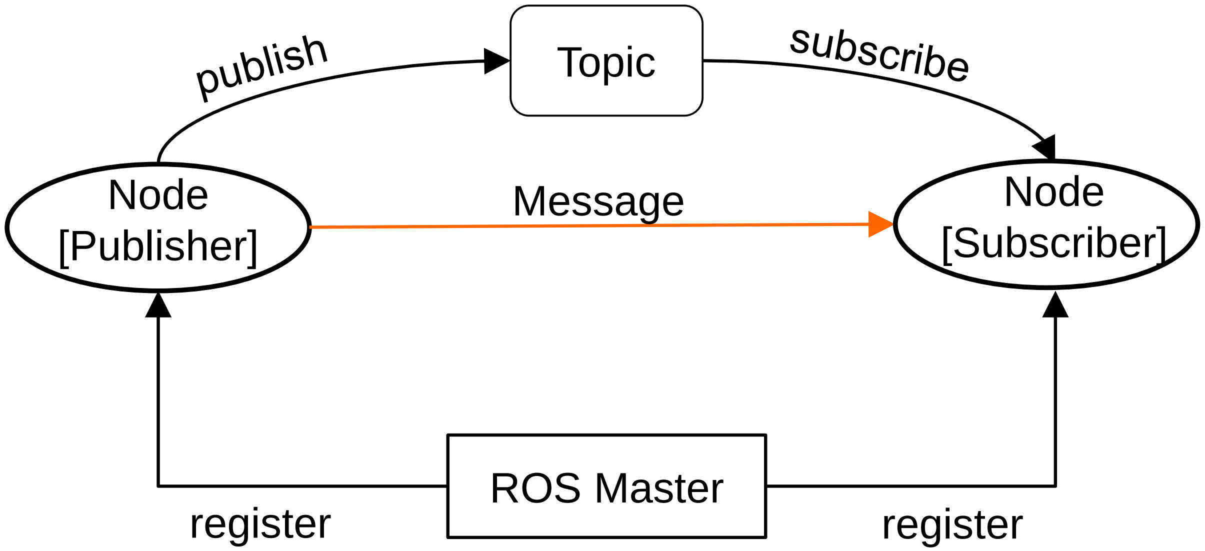
Get further infos about a topic:

```
$ rostopic info /<topic>
```

Fachhochschule Dortmund
University of Applied Sciences and Arts

The fundamental elements ROS-message

This chapter gives an overview of the fundamental ROS-elements.



A simple ROS-network

The fundamental elements
ROS-message

- The message defines the *type* of topic.
- A topic *type* can be a combination of standard datatypes, like strings, integers (signed and unsigned), floats etc.
- A message can also be a nested structure of arrays of standard datatypes.

Get the structure of a message:

```
$ rosmsg show /<message-name>
```

```
geometry_msgs/PoseArray
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose[] poses
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

Fachhochschule Dortmund
University of Applied Sciences and Arts

Structure

we focus on students

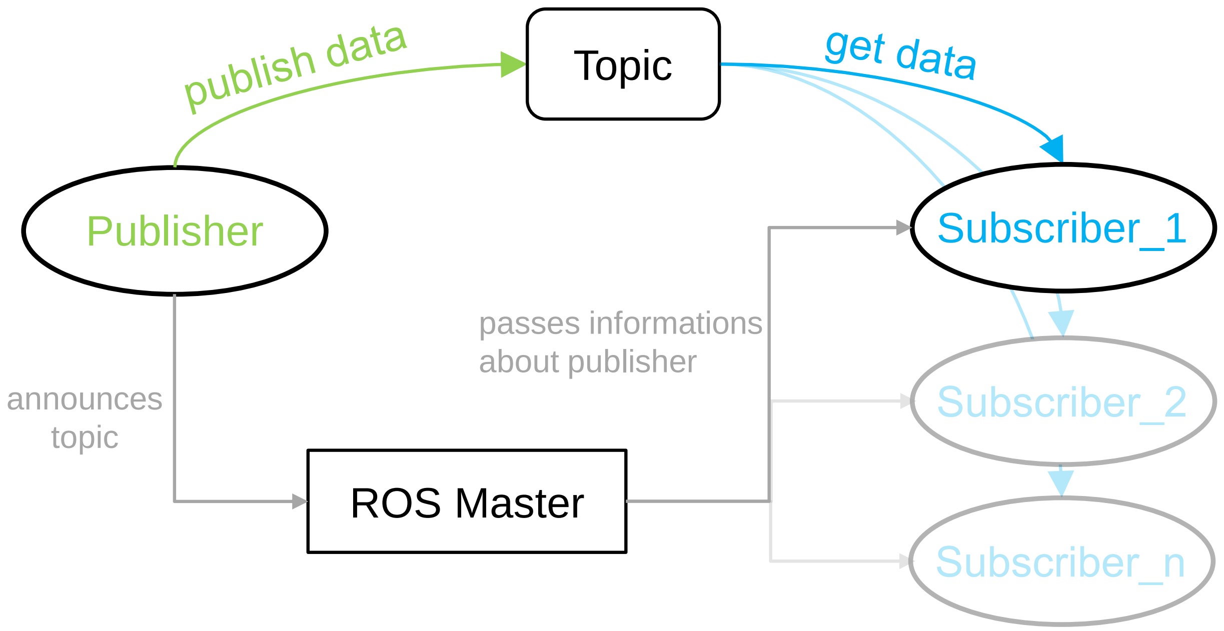
1. Introduction
2. Basic concept and architecture
3. The fundamental ROS-elements
4. **Basic communication**
5. ROS launch
6. Recording in ROS
7. Learning outcome
8. Looking ahead

from: http://wiki.ros.org/Distributions

Fachhochschule Dortmund
University of Applied Sciences and Arts

we focus on students

Basic communication publisher-subscriber-relation

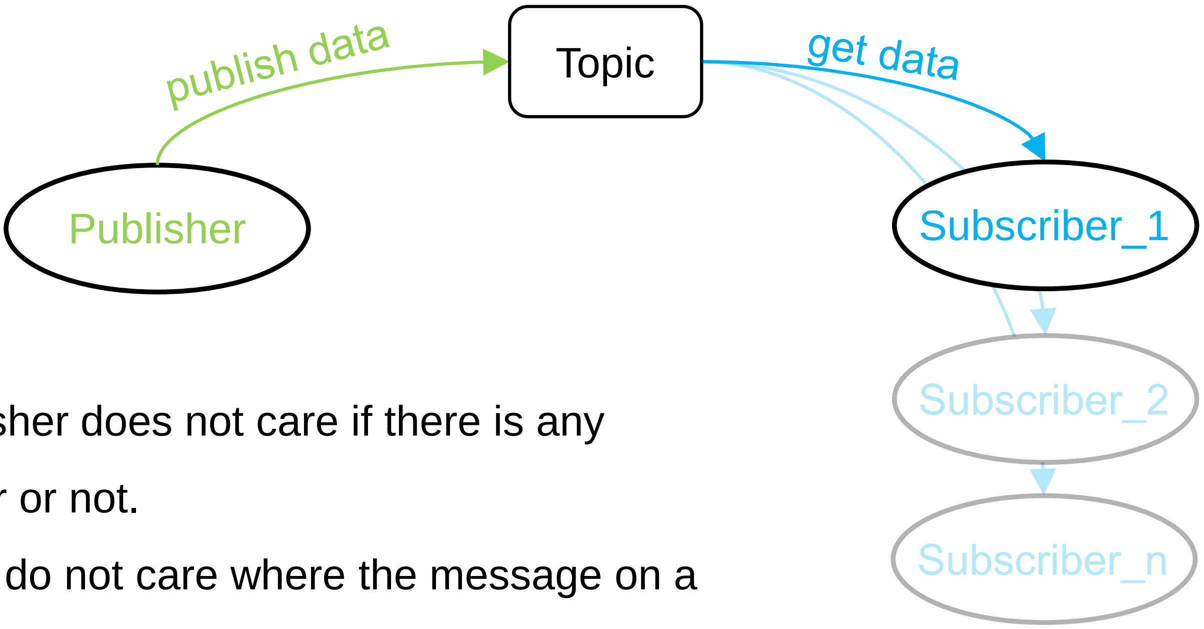


- The publisher announces a topic to the master.
- A subscriber pops up and gets information from the master about desired topics.

Fachhochschule Dortmund
University of Applied Sciences and Arts

we focus on students

Basic communication publisher-subscriber-relation



- The publisher does not care if there is any subscriber or not.
- Subcriber do not care where the message on a dedicated topic comes from

  → This can lead to strange behavior!

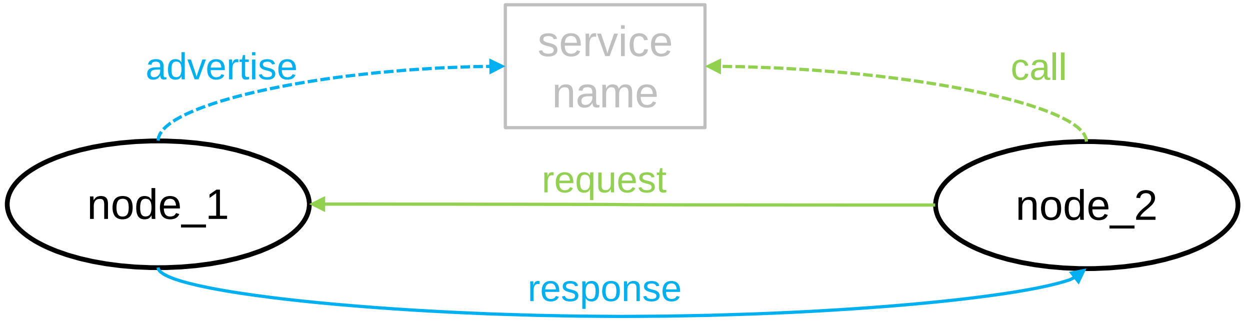
Note:

- If continuous information are needed and/or many nodes need data from one node, than use a publisher-subscriber-relation

Fachhochschule Dortmund
University of Applied Sciences and Arts

we focus on students

Basic communication
ROS service



- node_1 advertises a service, e.g. to turn a camera.
- node_2 wants to run a task from another node, but needs to know that it is really done.
  - → node_2 calls the appropriate service (sends a request).
  - → node_1 executes the task and sends a response (job done or failed).

To call a service enter:

```
$ rosservice call /<service-name>
```

Basic communication
ROS service

- Services are defined in service files (*.srv).

→ Basic structure of a *.srv-file

```
request
---
response
```

- The definition is similar to the definition of messages

→ Example of a *.srv-file

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

Note:

- When information about success or failure of a process is needed, use *rosservice*.

Fachhochschule
Dortmund
University of Applied Sciences and Arts

we
focus
on
students

Structure

1. Introduction
2. Basic concept and architecture
3. The fundamental ROS-elements
4. Basic communication
5. **ROS launch**
6. Recording in ROS
7. Learning outcome
8. Looking ahead

from: http://wiki.ros.org/Distributions

ROS launch

- A usual usecase is to start many nodes at the same time (normally when you start the system)
- It is very unhandy to do this using *rosrun […]*

→ ROS provides a mechanism called *roslaunch […]*

- It is a XML-file which contains instructions, like what node is to launch with which parameters.
- This prevents from name-collision at launch-time
- and gives you the opportunity to set e.g. special parameters to the node.

Fachhochschule Dortmund
University of Applied Sciences and Arts

ROS launch

we focus on students

Example of a simple XML-launchfile (`example.launch`):

```
<launch>
<node pkg = "package_name" type = "executable" name = "node_name"/>
</launch>
```

Execute with:

```
$ roslaunch example.launch
```

Note:

- To automate complex startup-procedures use *roslaunch.*
- The XML-file has to start with `<launch>` and to end with `</launch>`.
- Every launchfile ends with *.launch*

Fachhochschule Dortmund
University of Applied Sciences and Arts

Structure

we focus on students

1. Introduction
2. Basic concept and architecture
3. The fundamental ROS-elements
4. Basic communication
5. ROS launch
6. **Recording in ROS**
7. Learning outcome
8. Looking ahead

from: http://wiki.ros.org/Distributions

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Recording in ROS
ROS bag

we
focus
on
students

- ROS bag is a binary file format with the suffix **.bag*
- It is designed to record data from a topic (→ message-stream)
- ROS bag allows to record and replay datasets

Command to record all published topics:

```
$ rosbag record --all
```

Command to replay a **.bag*-file:

```
$ rosbag play <bag-name>.bag
```

Recording in ROS
rosbag

Command to record only dedicated topics:

```
$ rosbag record /topic_1 /topic_2 […] /topic_n
```

Without using a special option, ROS bag creates a filename from the actual timestamp. To declare a more clearly filename, use:

```
$ rosbag record -o <file-name> /topic
```

→ To stop a running recording, just press **Ctrl+[C]** in the appropriate terminal window.

There are a lot of options, which make it easier and more productive using ROS bag.

For detailed information see: http://wiki.ros.org/rosbag/Commandline

Fachhochschule Dortmund
University of Applied Sciences and Arts

Recording in ROS
rosbag

To get detailed information about a **.bag*-file:

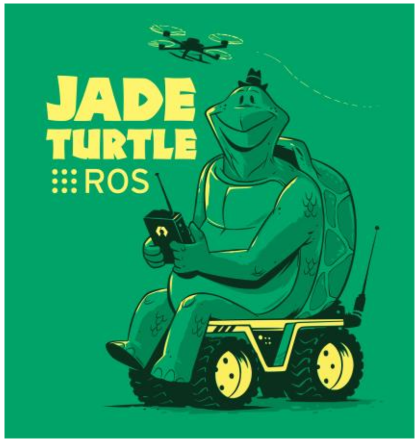
```
$ rosbag info
```

```
path:        camera-file.bag
version:     2.0
duration:    9.7s
start:       Aug 08 2017 12:21:41.68 (1502187701.68)
end:         Aug 08 2017 12:21:51.37 (1502187711.37)
size:        459.4 MB
messages:    98
compression: none [98/98 chunks]
types:       sensor_msgs/Image [060021388200f6f0f447d0fcd9c64743]
topics:      /image_raw   98 msgs    : sensor_msgs/Image
```

md5-Hash

Structure

1. Introduction
2. Basic concept and architecture
3. The fundamental ROS-elements
4. Basic communication
5. ROS launch
6. Recording in ROS
7. Learning outcome



from: http://wiki.ros.org/Distributions

Learning outcome

What this workshop should provide:

- Get an idea of the basic ROS-concept.
- Knowing about elemantary ROS-element:
  - ROS-master
  - nodes
  - topics and messages
- Understand the communication in simple ROS-networks.
- How to automate complex start-up procedures.
- Be able to record single topics and replay **.bag*-files.