ROS2 using Docker and Python

Prof. Ivano Notarnicola

Department of Electrical, Electronic, and Information Engineering
Alma Mater Studiorum Università di Bologna

ivano.notarnicola@unibo.it

Distributed Autonomous Systems M
A.A. 2024-2025

Lab accounts

The lab computers provide the necessary environment

Create your account at

https://infoy.ing.unibo.it

Robotic Operating System 2 (ROS2)

We will be using ROS2 to model multi-agent systems

- ***ROS2 Humble Hawksbill*** (Release date: May 23, 2022, EOL: May 2027)
- Other versions may differ for minor aspects
- Docker or VirtualBox, if necessary...

Preliminary Docker setup

Install **docker-desktop** from https://www.docker.com/products/docker-desktop/

On MacOS, install **XQuartz** from https://www.xquartz.org/

Create a docker image

Go in the root directory containing **/docker_setup** and **/docker_ws** directories (download from Virtuale)

To create an “image” named, e.g., **ros2_humble_image**, using the existing builder (it internally invokes **docker build**) use

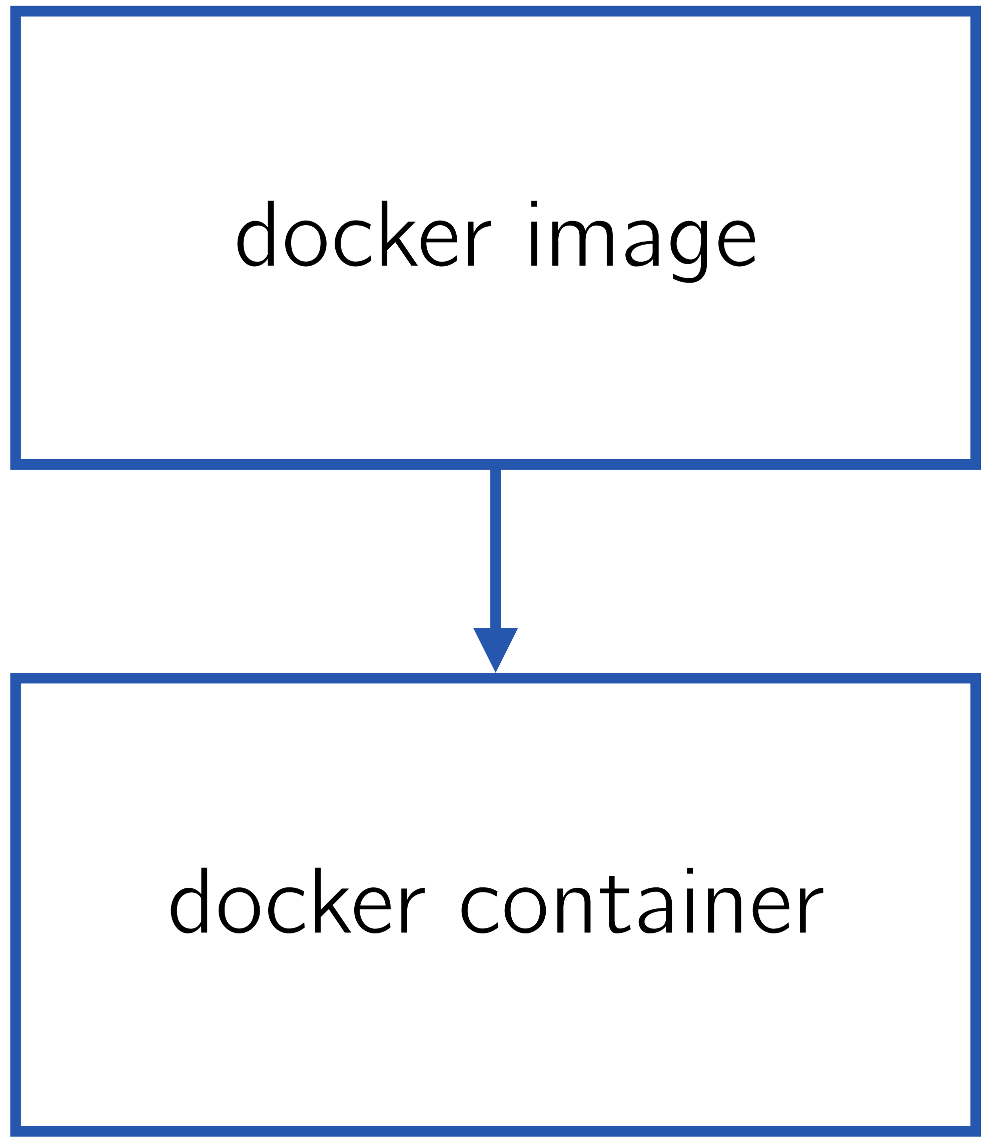
. docker_setup/buildImage docker_setup/ ros2_humble_image

To list the docker images located in your computer use

docker images or **docker image ls**

To remove an existing image, e.g., **ros2_humble_image**, use (or **rmi**)

docker image rm ros2_humble_image



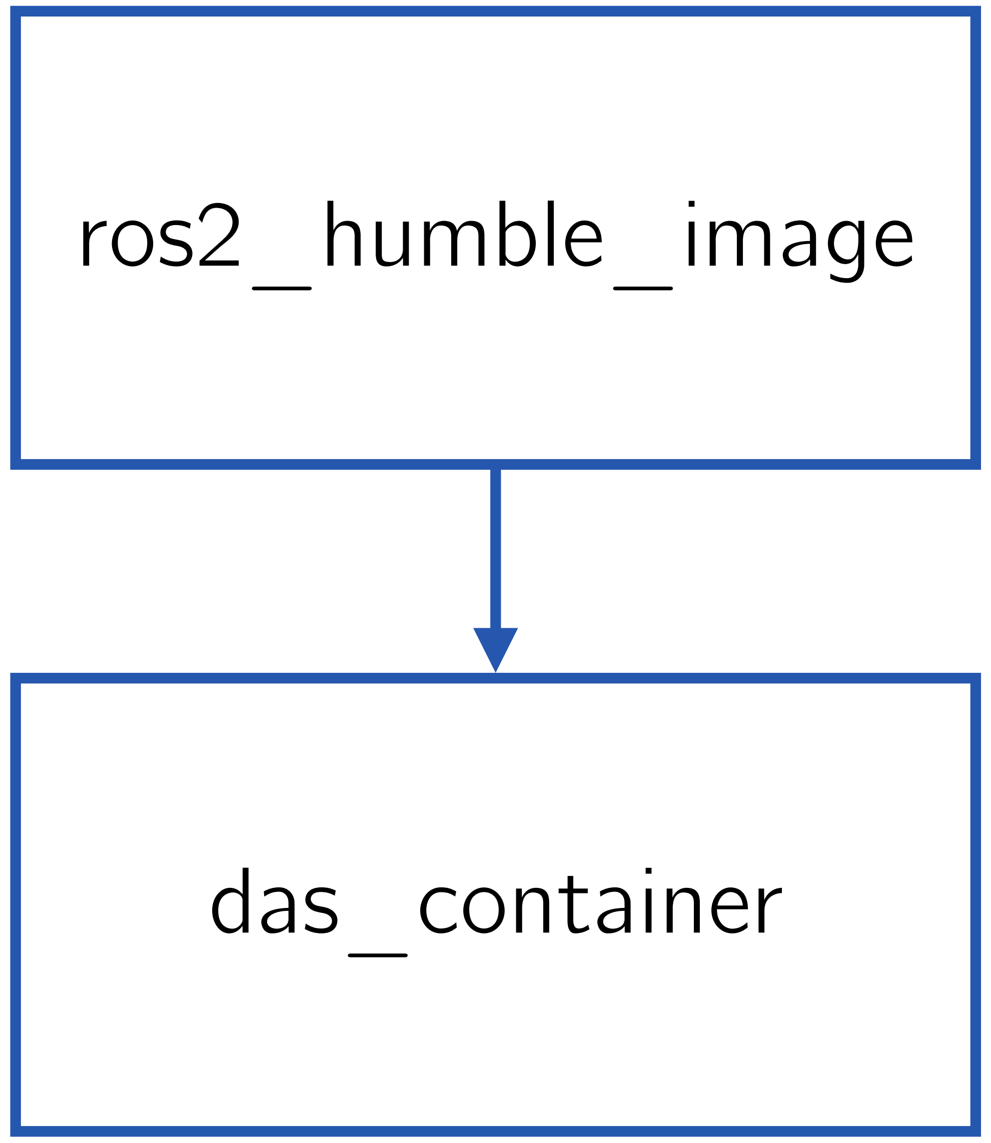
Note. Images and containers are interlaced

Create a docker container

To list the existing containers use
docker container ps -a

Go in the root directory (containing **/docker_setup** and **/docker_ws** directories) and create a docker container, named, e.g., **das_container** based on **ros2_humble_image**, (internally invoking the command **docker run**) via

. docker_setup/createContainer das_container ros2_humble_image



Start/stop an existing/running container
docker start das_container or **docker stop das_container**

Delete an existing container
docker container rm das_container

Execute a running container (automatically running after the creation)
docker exec -it das_container /bin/bash

Preparing a workspace

Activate ROS2 (maybe add it to **~/.bashrc**)

```
. /opt/ros/humble/setup.bash
```

Definition. A ***workspace*** is a directory containing ROS2 packages

Best practice #1: create a new directory that will contain the ROS2 workspace

```
mkdir -p das_ros2_ws/src
cd das_ros2_ws/src
```

Best practice #2: put the packages in your workspace inside the **src** directory

Creating a ROS2 package

Definition. A ***package*** can be considered a container for the ROS2 code

Create a package from the **src** directory using
ros2 pkg create --build-type ament_python package_name

Example: if the **package_name** is **pub_sub,** then the creation command would be
ros2 pkg create --build-type ament_python pub_sub

A Python package consists of

- **package.xml** file containing meta information about the package
- **setup.py** containing instructions for how to install the package, i.e., entry points for nodes
- **setup.cfg** is required when a package has executables, so ROS2 run can find them
- **/package_name** a directory with the same name as your package, used by ROS2 tools to find your package, it contains **__init__.py**

Refining a ROS2 package

Recall that there is a nested subdirectory with a Python package having the same name as the ROS2 package

Example:

das_ros2_ws/src/pub_sub/pub_sub

Then the following configuration files must be adapted

- Specify the "entry points" in **setup.py**: set the name of each node with its dedicated source file
  '**node_name = pkg_name.source_file:main**'
- Add dependencies in **package.xml**: set the package properties
  <exec_depend>rclpy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

The source files of a ROS2 ***Node***, e.g., **source_file.py,** must be put in the directory **das_ros2_ws/src/pub_sub/pub_sub/**

Example: **talker = pub_sub.publisher:main**

Compiling the package and running a node

From the ROS2 workspace root (e.g., **das_ros2_ws**), run (symbolic links optimize the Python workflow)

```
colcon build --symlink-install
```

After a successful build, the following additional directories should appear

```
das_ros2_ws/build
das_ros2_ws/install
das_ros2_ws/log
```

We are ready to run the ***Node*** (two terminals needed). Go in **das_ros2_ws** and execute

```
. /opt/ros/humble/setup.bash
. install/setup.bash
```

(**source** and **.** ("period") are practically equivalent)

Run (in the first terminal) the **talker** node

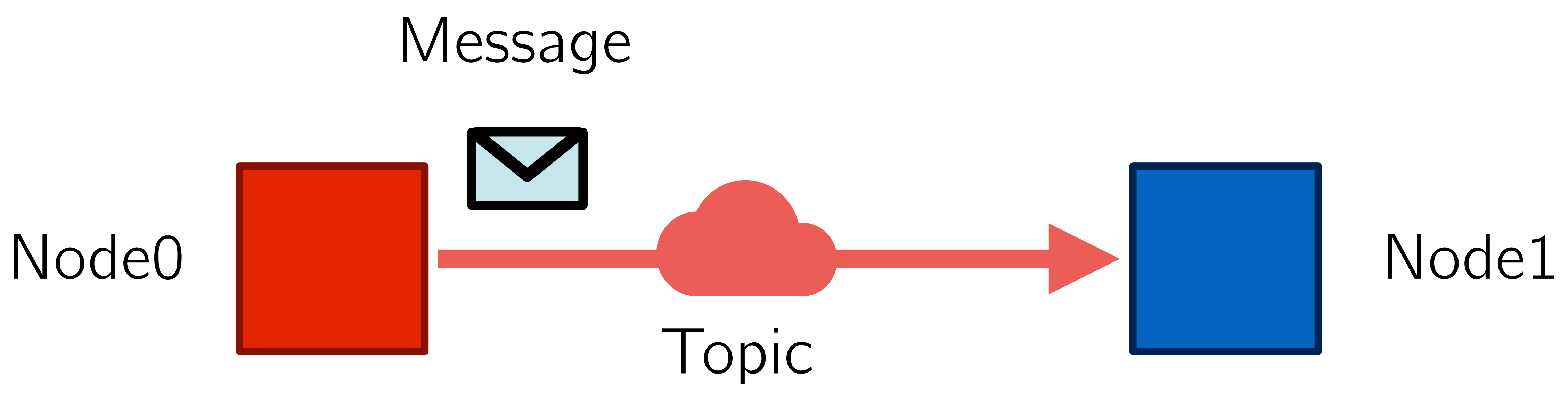
```
ros2 run pub_sub talker
```

Run (in the second terminal) the **listener** node

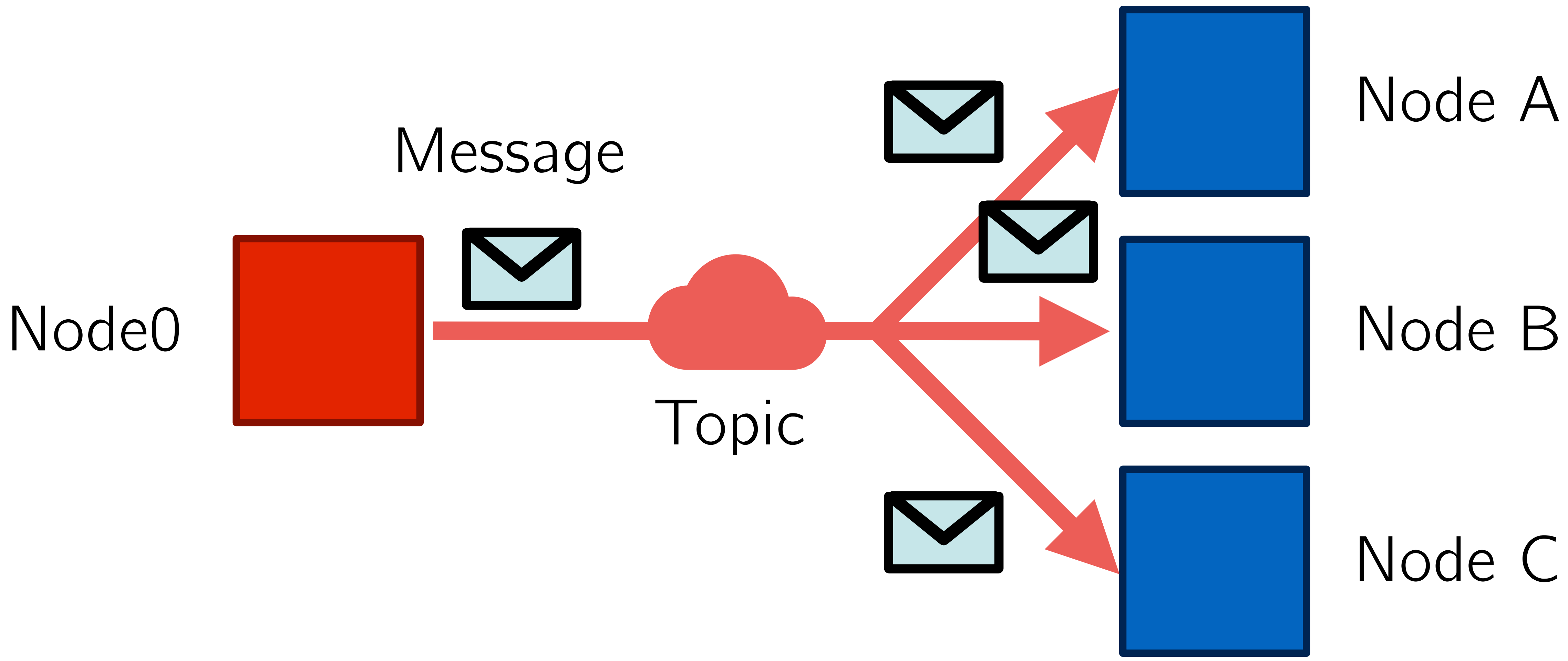
```
ros2 run pub_sub listener
```

Publish-Subscribe protocol: the idea

A one-to-one communication

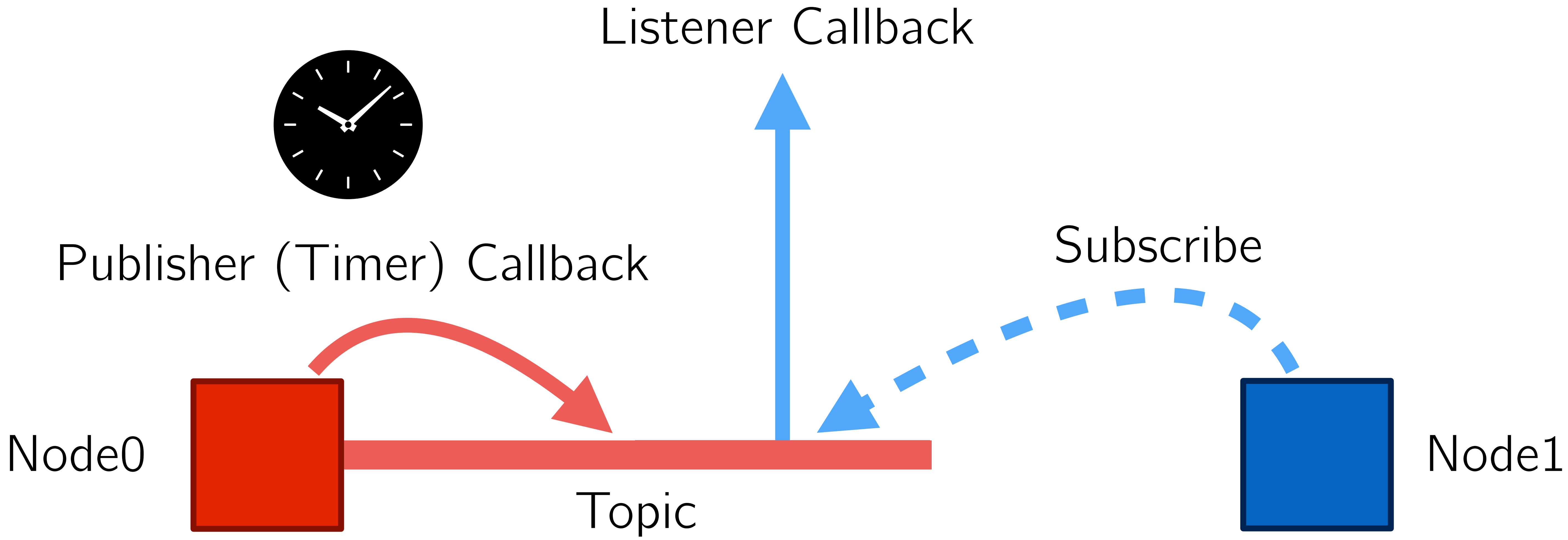


A one-to-many communication



Publish-Subscribe protocol: the implementation

Let us focus on the one-to-one communication



Launch multiple nodes at once

Create a directory to store the launch file

```
mkdir -p das_ros2_ws/pub_sub/src/launch_folder
```

Create the launch file, e.g., **pub_sub_launch.py** via

```
touch das_ros2_ws/pub_sub/src/launch_folder/pub_sub_launch.py
```

Modify the **setup.py:** add in the header **from glob import glob** and in the **data_files** list:

```
("share/" + package_name, glob("launch_folder/pub_sub_launch.py"))
```

Best practice #3: add a dependency in the file **package.xml**

```
<exec_depend>ros2launch</exec_depend>
```

Once the launch file is ready, from **das_ros2_ws/** run

```
ros2 launch pub_sub pub_sub_launch.py
```