Middlesex University Dubai

PDE4430

Introduction to ROS

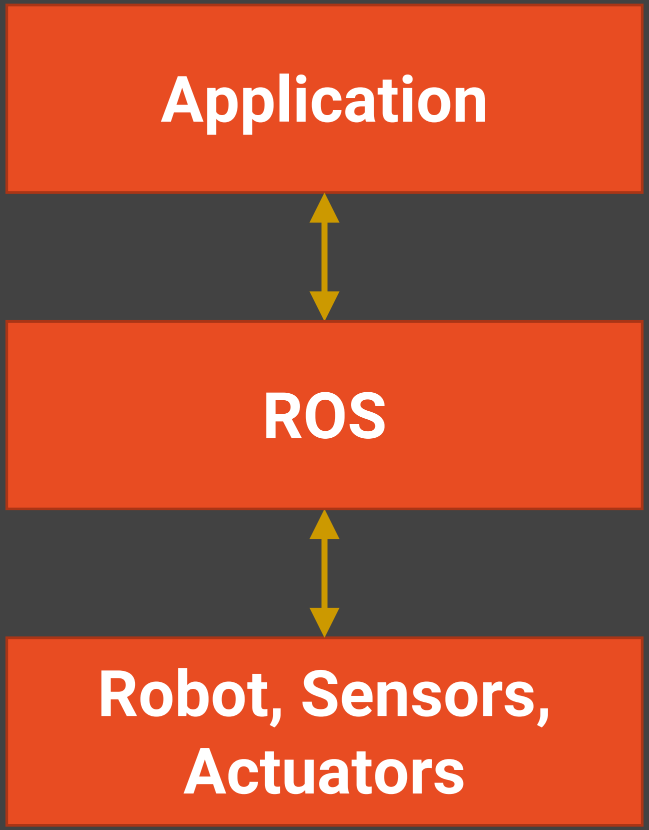
What is ROS?

What is ROS?

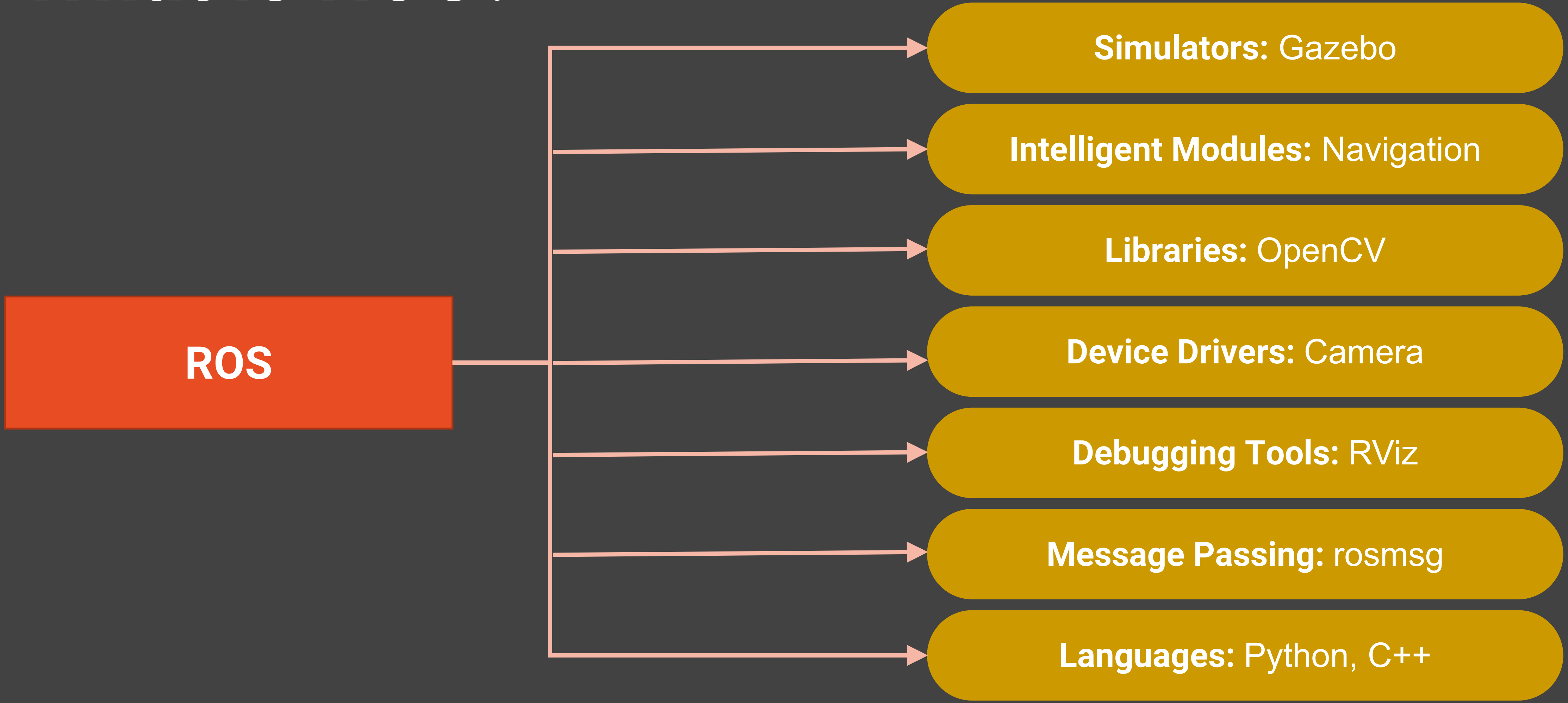
What is ROS?

What is ROS?

- The Robot Operating System (ROS) is a flexible framework for writing robot software
- “It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms.”
- Hardware abstraction
- Low–level device control
- Communication between processes
- Package management
- Implementation of commonly used functionality



What is ROS?



ROS

Simulators: Gazebo

Intelligent Modules: Navigation

Libraries: OpenCV

Device Drivers: Camera

Debugging Tools: RViz

Message Passing: rosmsg

Languages: Python, C++

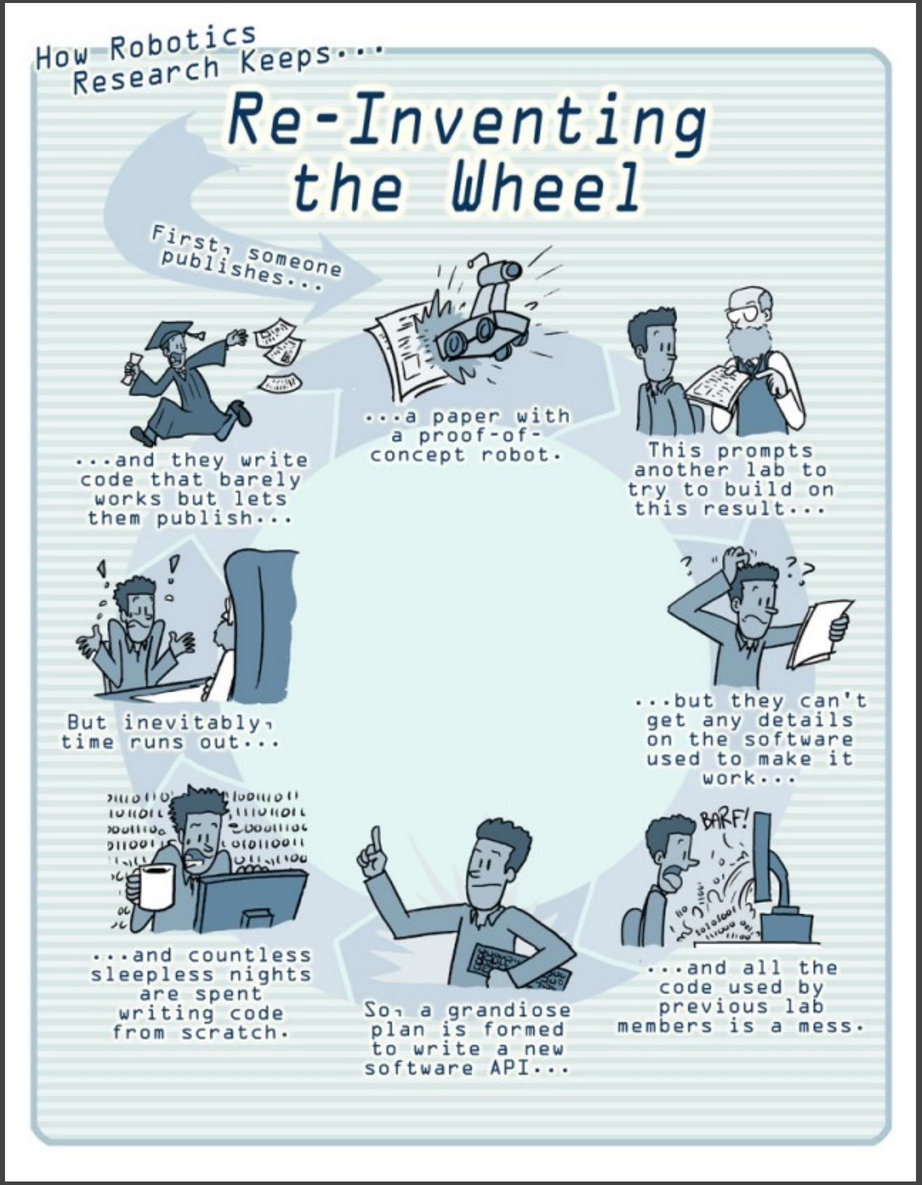
Why ROS?

- To encourage *collaborative* robotics software development

"One lab might have experts in mapping indoor environments, and could contribute a world-class system for producing maps.

Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter.

ROS was designed specifically for groups like these to collaborate and build upon each other's work."


How Robotics Research Keeps...
Re-Inventing the Wheel
First, someone publishes...
...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...
BARF!
...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...
...and they write code that barely works but lets them publish...

Getting Started with ROS

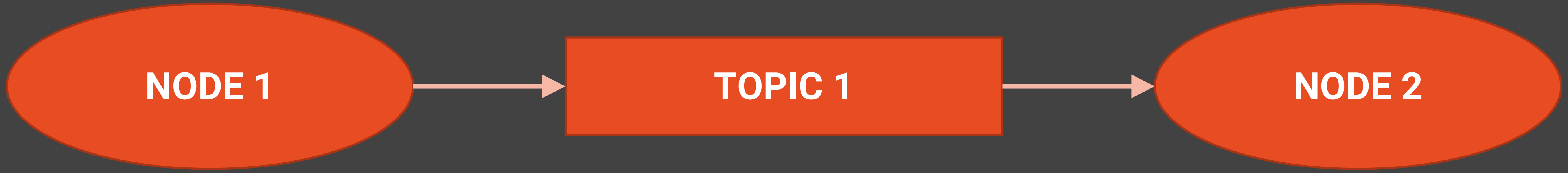
ROS Essentials

- Two fundamental concepts –

NODES

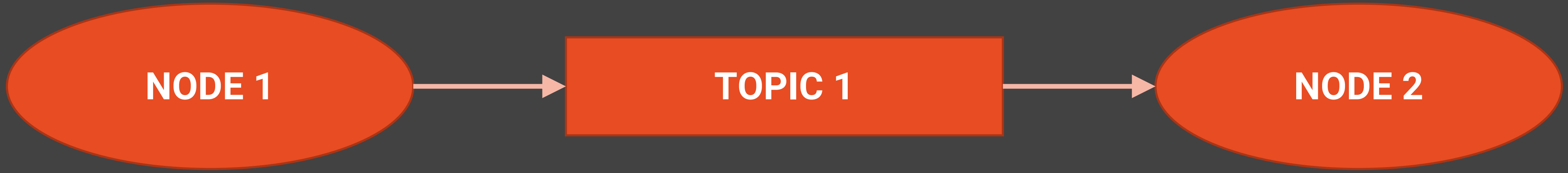
TOPICS

ROS Nodes



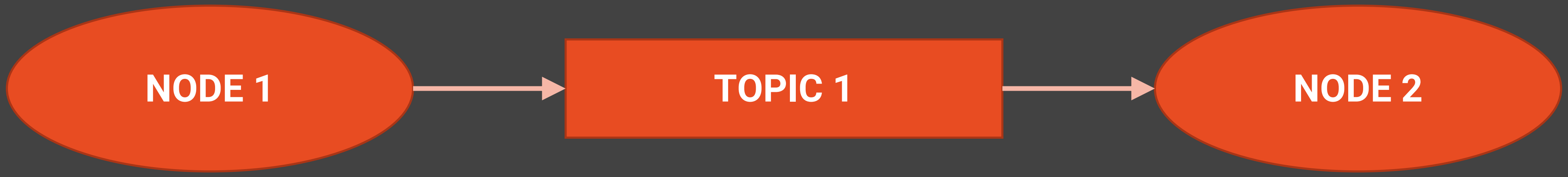
- Data/information processing software unit
- Building block of a ROS application
- All functional requirements of a ROS application are implemented as nodes
- Nodes are combined together into a graph and communicate with one another using topics

ROS Nodes



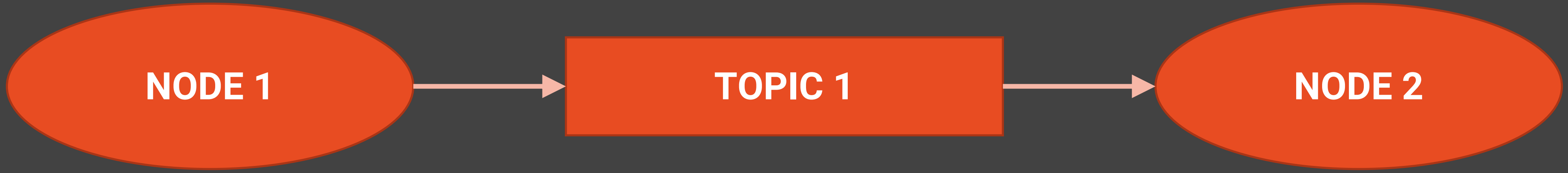
- Meant to operate at a fine-grained scale
- Example:
  - One node controls a laser range-finder
  - One node controls the robot's wheel motors
  - One node performs localization
  - One node performs path planning

ROS Nodes



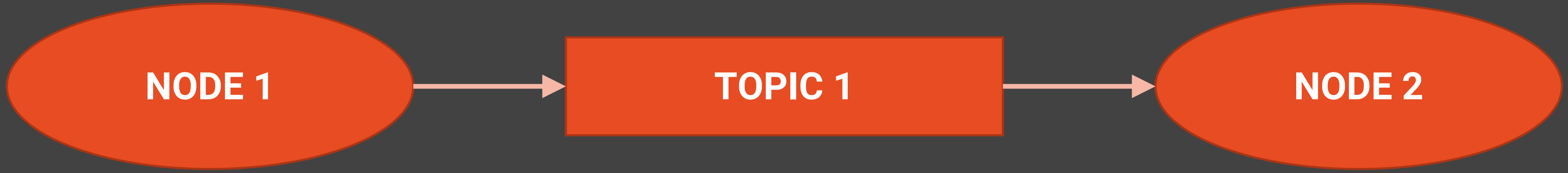
- Several advantages:
  - Makes the application robust: Crashes isolated to individual nodes
  - Reduces Code complexity
  - Hides implementation details
  - Makes it easy to add functionality

ROS Topics



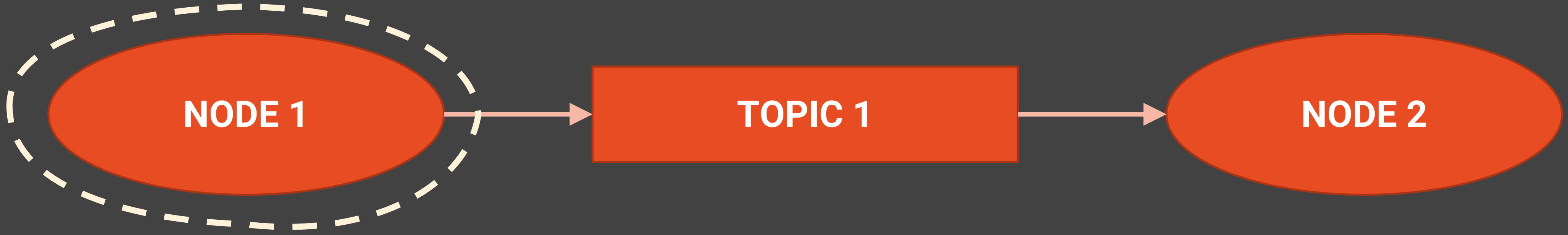
- One of the most important features of ROS
- An entity that is used to transport information (**messages**) between nodes
- Nodes interact with each other by exchanging relevant information as required
- Topic is identified by a **name** and a **type**

ROS Topics



- In general, nodes are not aware of who they are communicating with
- Nodes interested in data ‘**subscribe**’ to the relevant topic
- Nodes that generate data ‘**publish**’ to the relevant topic
- There can be multiple publishers and subscribers to one topic

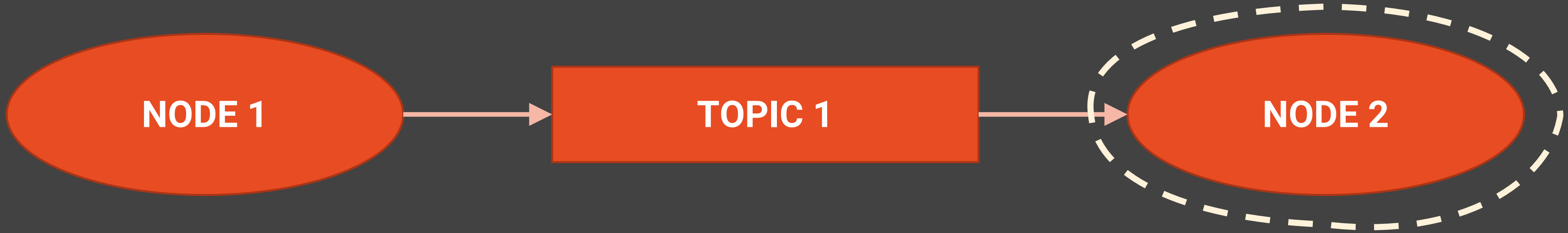
ROS Nodes – Publisher



- Data/information processing units – But where does this information come from?
- A publisher node **generates** information
- Examples?

Middlesex University Dubai

ROS Nodes – Subscriber



- A subscriber node **receives** information
- Examples?
- Both publishers and subscribers communicate via topics

Hands–On With ROS

Publishers, Subscribers, Topics, Messages and Turtlesim

ROS Master

- First step of starting any ROS application
- Allows communication between nodes (via topics)
- Tracks publishers and subscribers
- Allows nodes to locate one another
- Run using the command: **`roscore`**

ROS – Step by Step

- First, start ROS master: `roscore`
- Open new terminal window
- Display list of nodes: `rosnode list`
- Display list of topics: `rostopic list`
- Start TurtleSim example:

```
rosrun turtlesim turtlesim_node
```

- `rosrun` allows you to run an executable in an arbitrary package from anywhere without having to give its full path

ROS – Step by Step

- Open new terminal and check list of nodes and topics again.
- Anything different?
- For more information about a node:

```
rosnode info NodeName
```

- For more information about a topic:

```
rostopic info TopicName
```

- See the contents of a topic:

```
rostopic echo TopicName
```

ROS – Step by Step

- Try the following commands. Stop after each:

```
rosnode list
rosnode info /turtlesim
rostopic info /turtle1/color_sensor
rostopic echo /turtle1/color_sensor (Ctrl+C to stop)
rosmsg show turtlesim/Color
rostopic info /turtle1/cmd_vel
```

ROS – Step by Step

- Now run:

```
rosrun turtlesim turtle_teleop_key
```

- This node publishes on the `/turtle1/cmd_vel` topic
- If you check topic list again, it will look the same – Why?
- Check nodes again. Do you see anything extra?
- To visualise flow of information:

```
rosrun rqt_graph rqt_graph
```

- Make this node window active and press Arrow Keys

ROS – Step by Step

- See values being published to the topic: `/turtle1/cmd_vel`
- See values being published to the topic: `/turtle1/pose`
- What's the difference between these two topics?
- Display the graph again with the echo terminal window running
- Will it be the same?
- Do you see something extra? If yes, what/why?

ROS – Step by Step

- As we saw earlier, a topic is defined by its type.
- The command to see the message type of a ROS topic:

```
rosmsg show msgName
```

- Try the command:

```
rosmsg show geometry_msgs/Twist
```

- Important to know the message type. Can be user-defined as well.

ROS – Step by Step

- You can even publish directly to a topic from the terminal:

```
rostopic pub -1 topicName msgType -- ’data’
```

- `–1` so it publishes once and quits
- For a steady stream of commands at 1Hz, replace with `–r 1`

```
rostopic pub -1 /turtle1/cmd_vel
geometry_msgs/Twist -- ’[2.0, 0.0, 0.0]’ ‘[0.0,
0.0, 1.8]’
```

ROS – Step by Step

- For more information about Turtlesim:

  https://wiki.ros.org/turtlesim

- Discussion

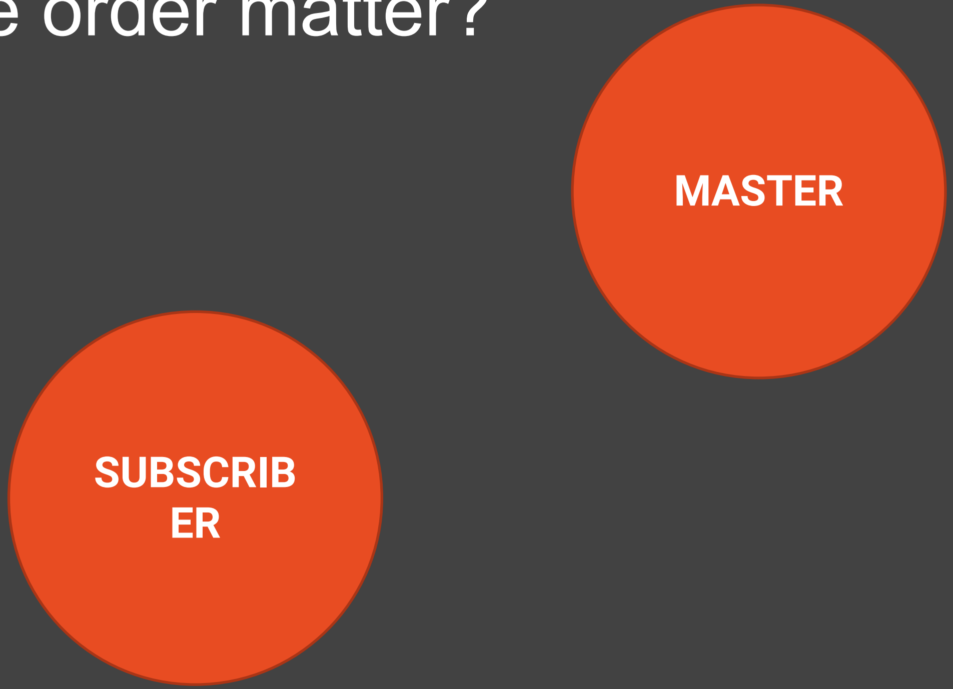
ROS Workflow – Step by Step

- Start the master node

MASTER

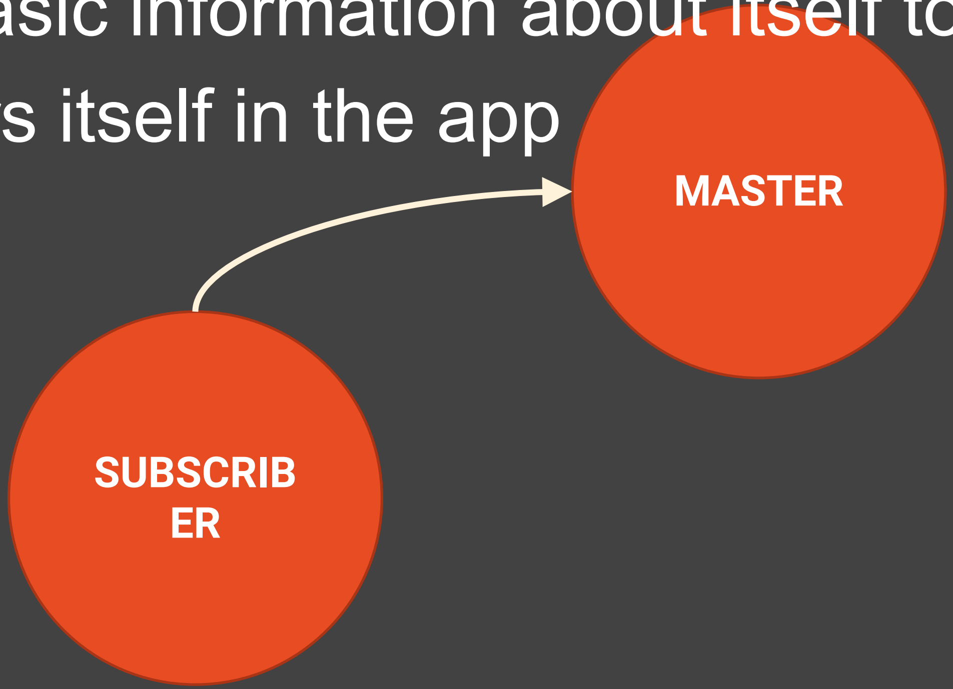
ROS Workflow – Step by Step

- Run the subscriber node (or publisher node)
- Does the order matter?



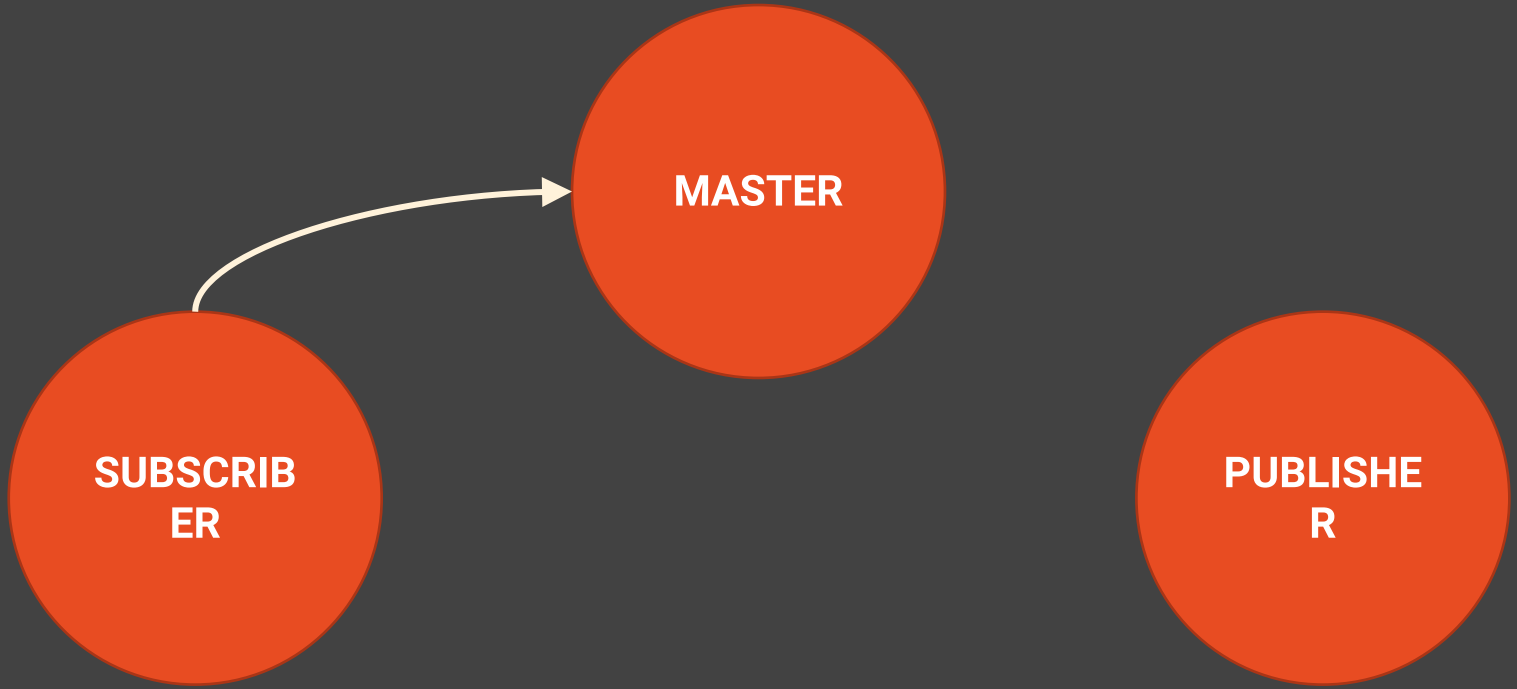
ROS Workflow – Step by Step

- Subscriber node connects to Master
- Gives basic information about itself to the Master
- Registers itself in the app



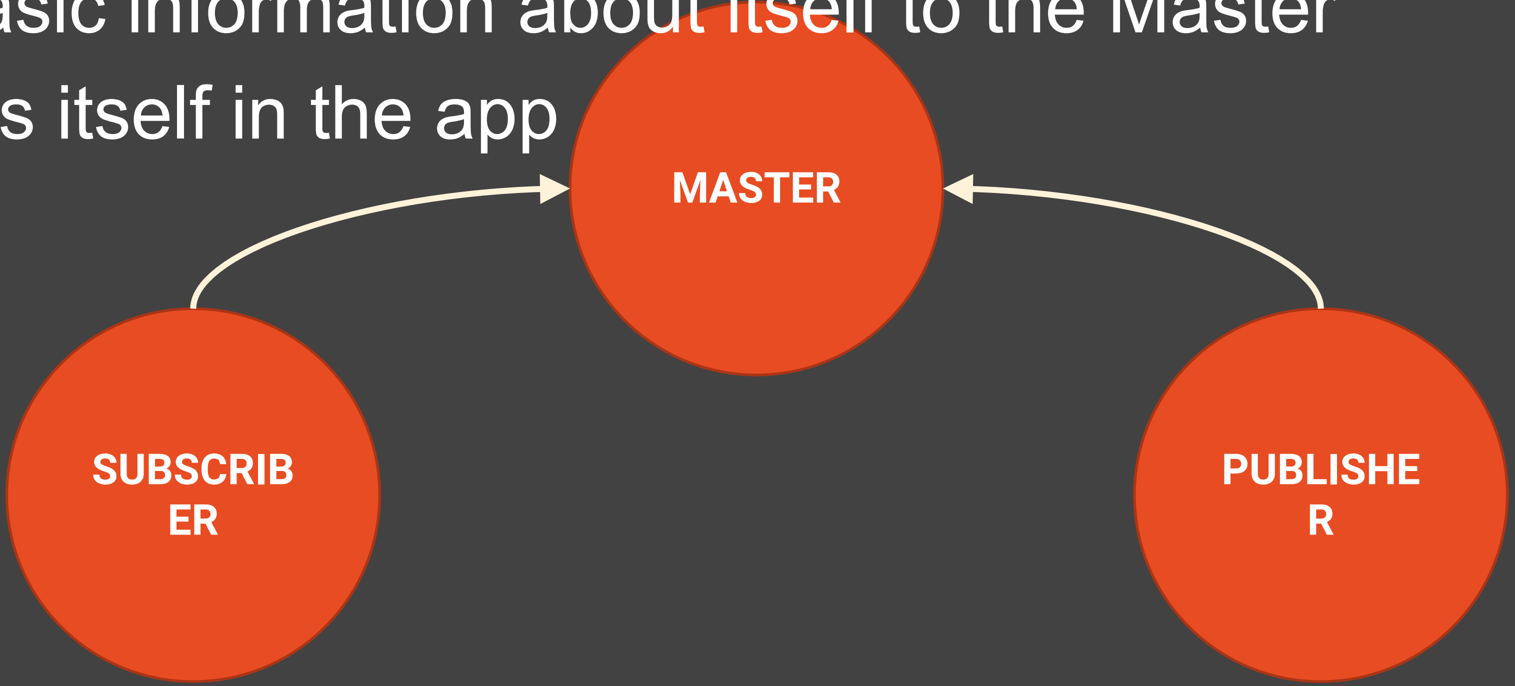
ROS Workflow – Step by Step

- Run the Publisher node



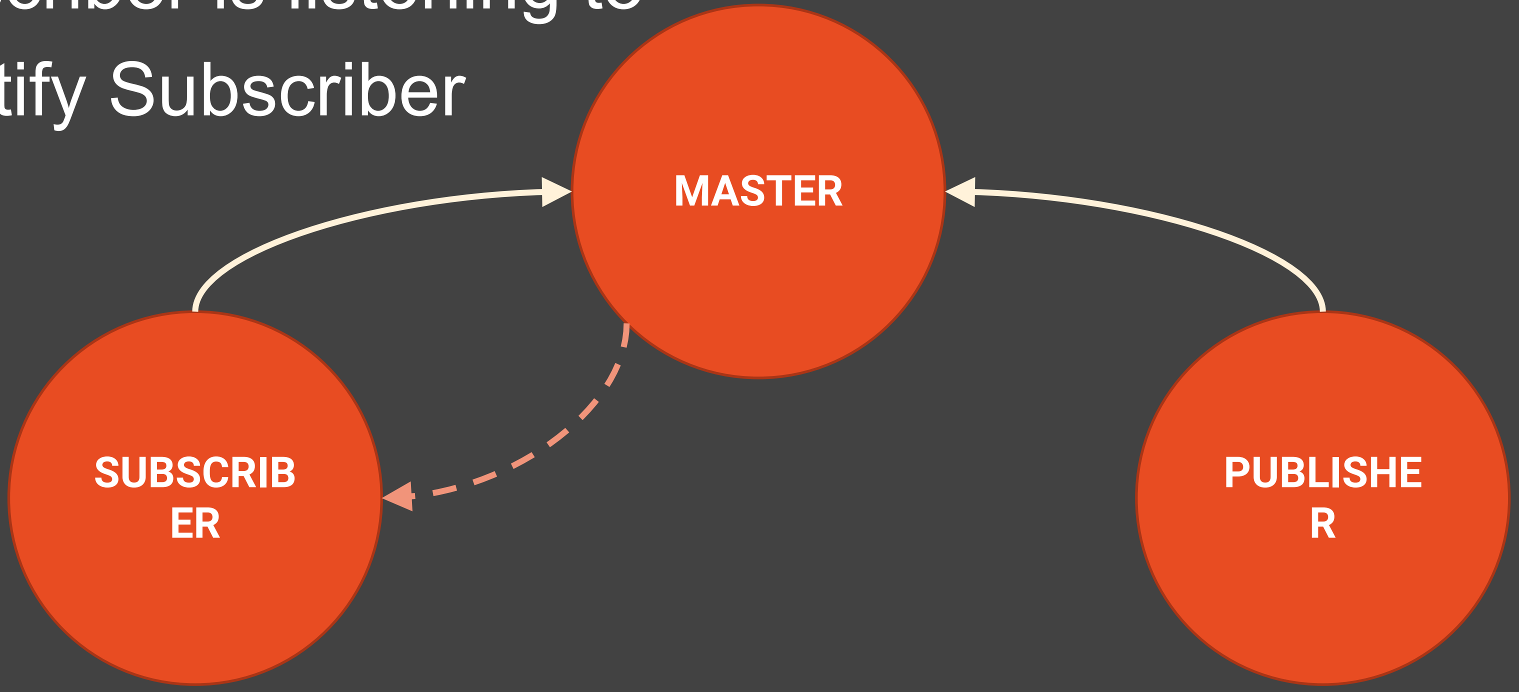
ROS Workflow – Step by Step

- Publisher node connects to Master
- Gives basic information about itself to the Master
- Registers itself in the app



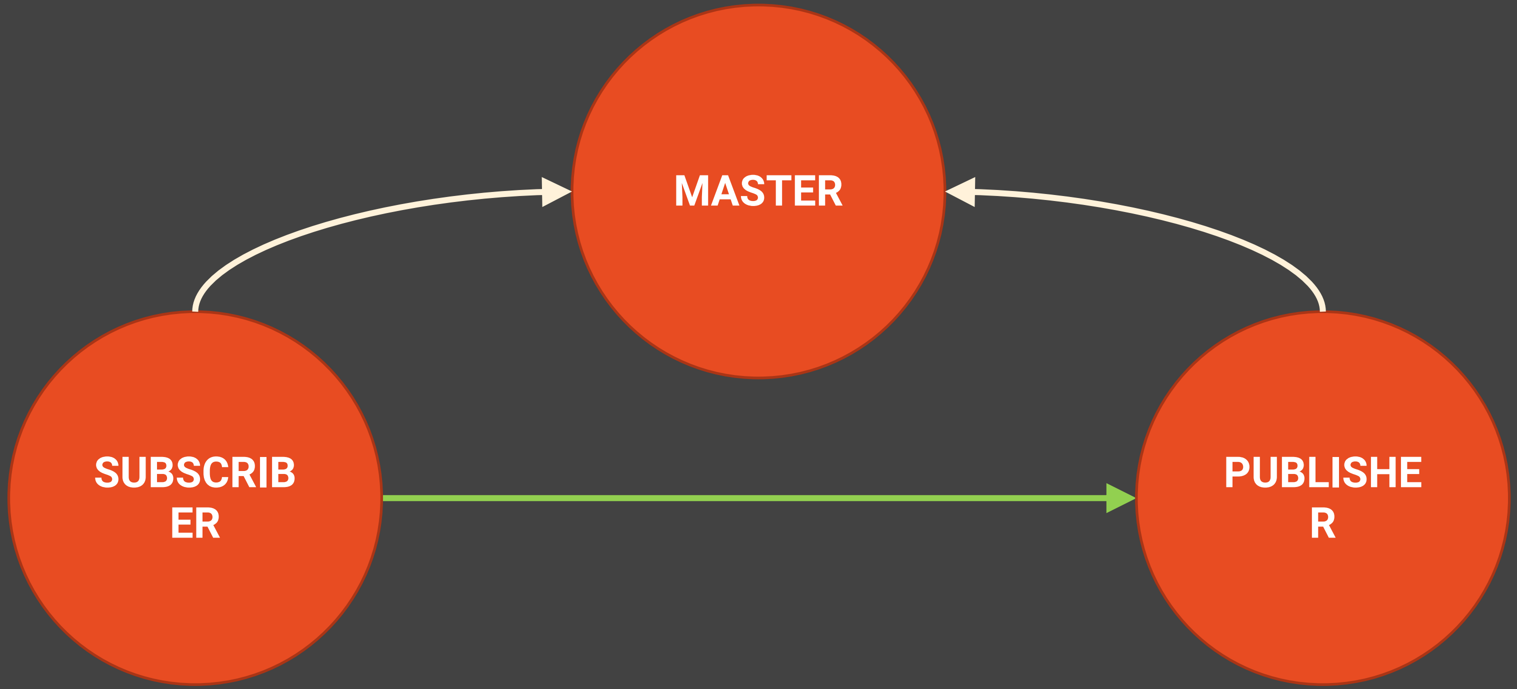
ROS Workflow – Step by Step

- Master now realises that there is now a publisher for a topic that the subscriber is listening to
- It will notify Subscriber



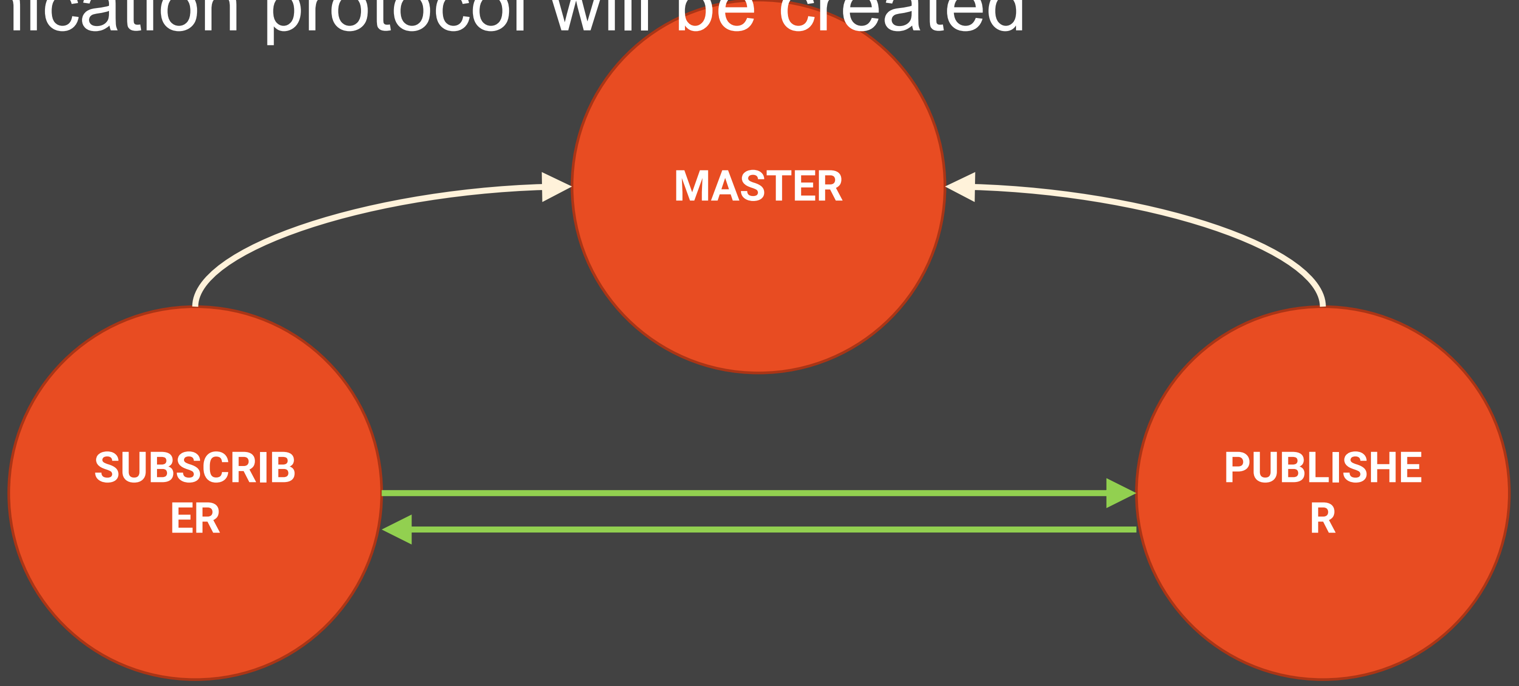
ROS Workflow – Step by Step

- Now, Subscriber will send a request to Publisher



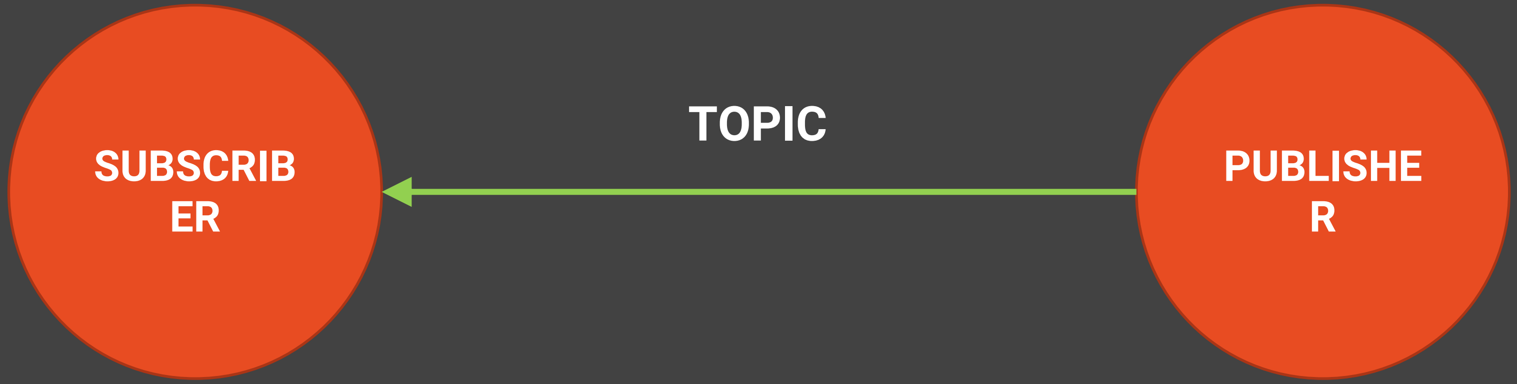
ROS Workflow – Step by Step

- Publisher will send a response
- Communication protocol will be created



ROS Workflow – Step by Step

- Publisher will start sending data at a certain frequency
- Subscriber will have a *callback* that will be triggered each time data is received

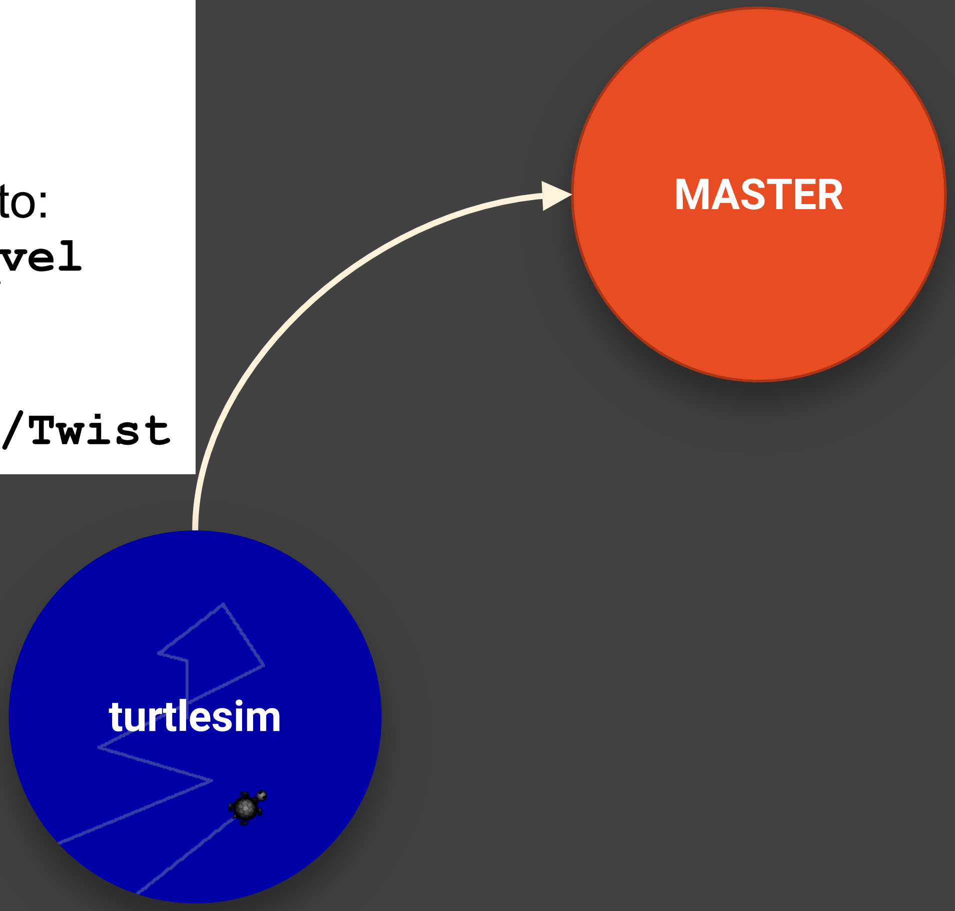


ROS Workflow – Turtlesim

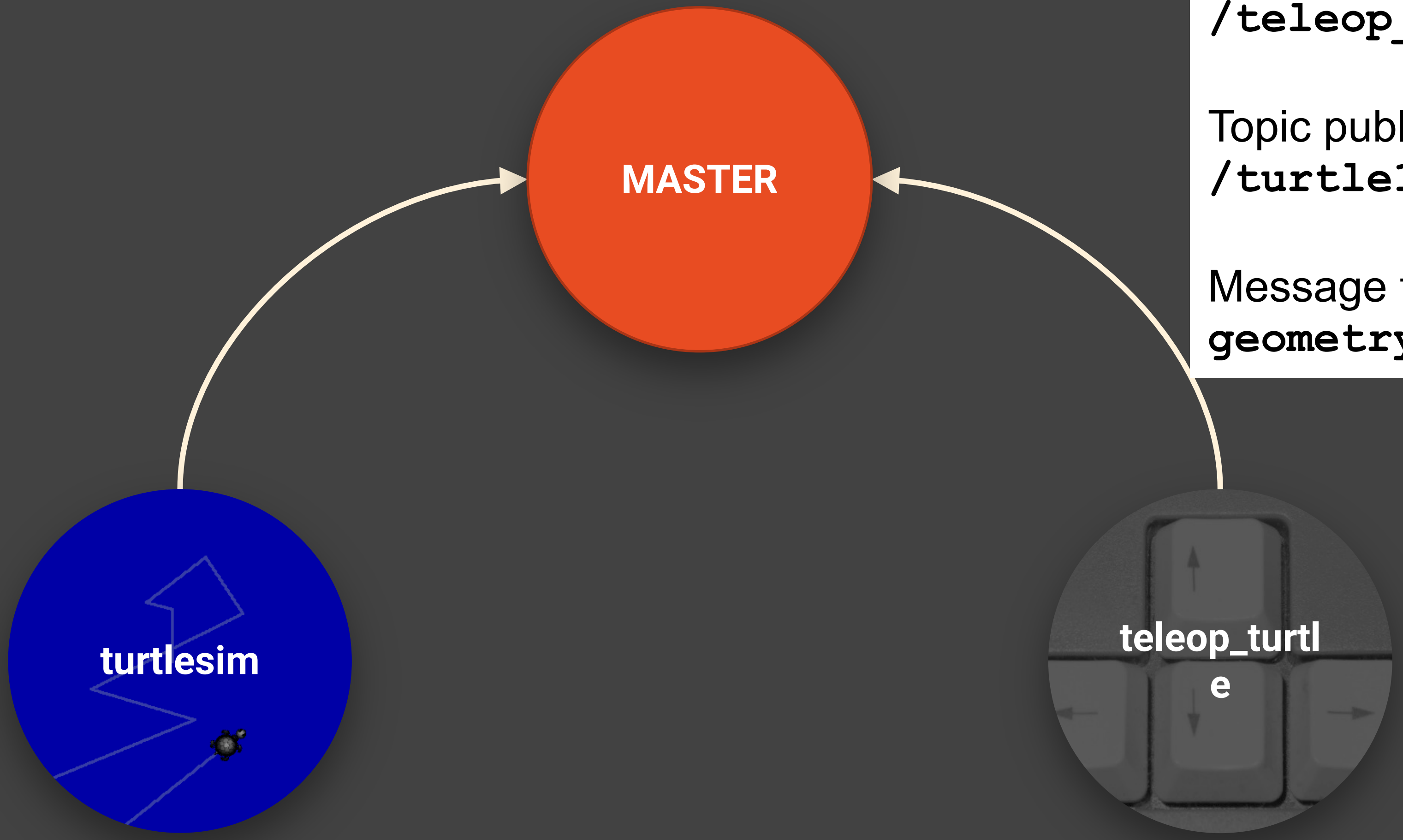
Subscriber node:
`/turtlesim`

Topic subscribed to:
`/turtle1/cmd_vel`

Message type:
`geometry_msgs/Twist`



ROS Workflow – Turtlesim

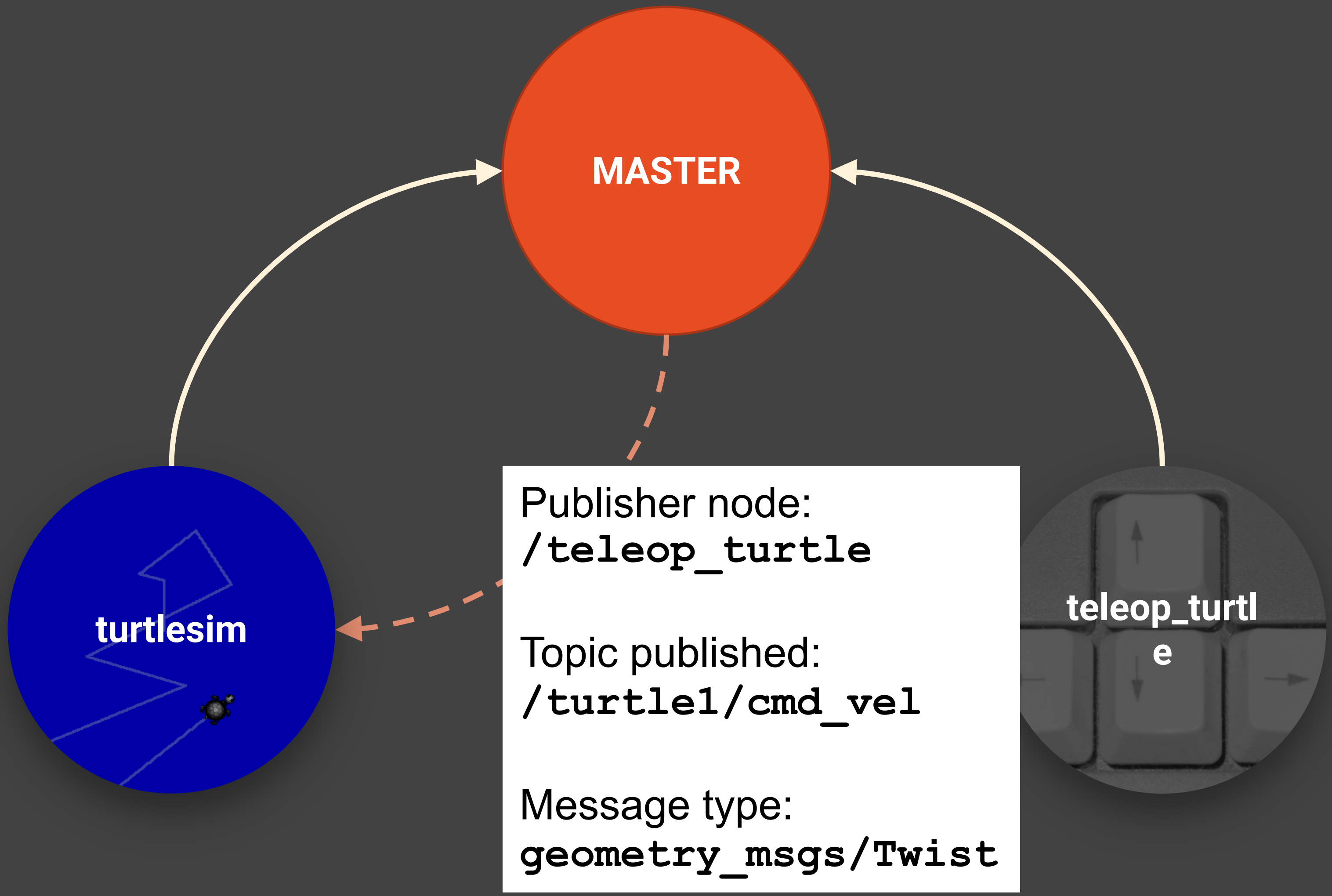


Publisher node:
`/teleop_turtle`

Topic published:
`/turtle1/cmd_vel`

Message type:
`geometry_msgs/Twist`

ROS Workflow – Turtlesim

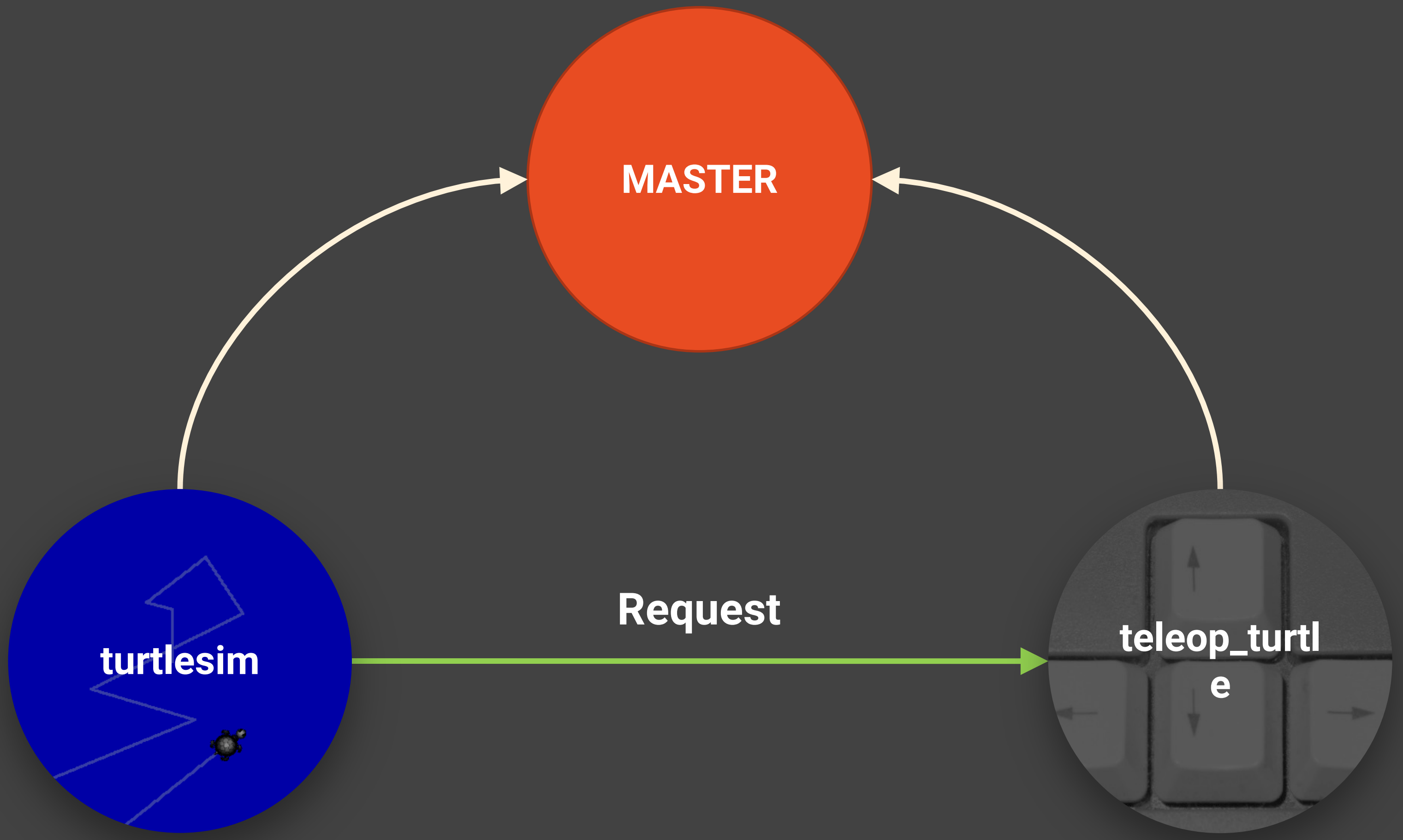


Publisher node:
`/teleop_turtle`

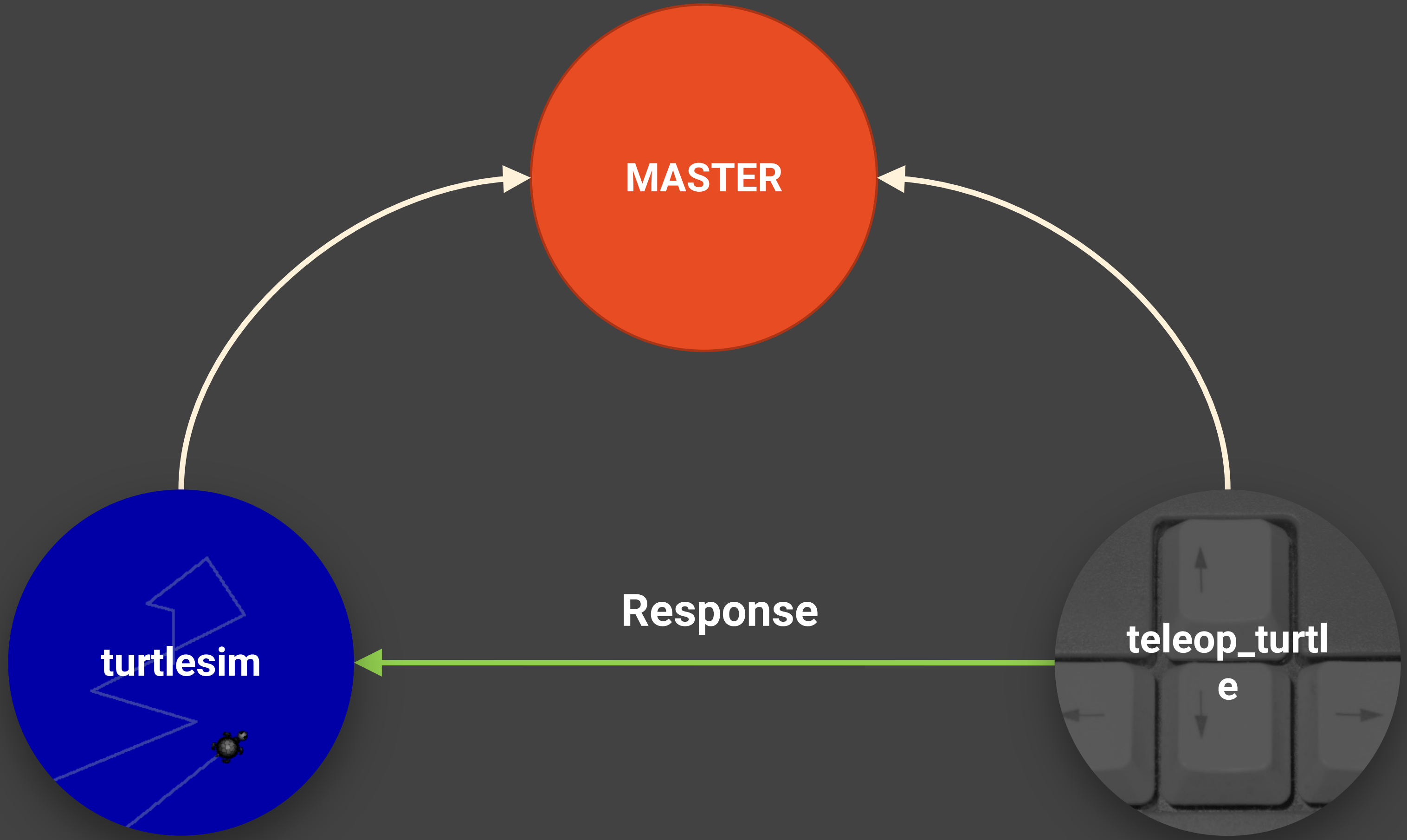
Topic published:
`/turtle1/cmd_vel`

Message type:
`geometry_msgs/Twist`

ROS Workflow – Turtlesim



ROS Workflow – Turtlesim



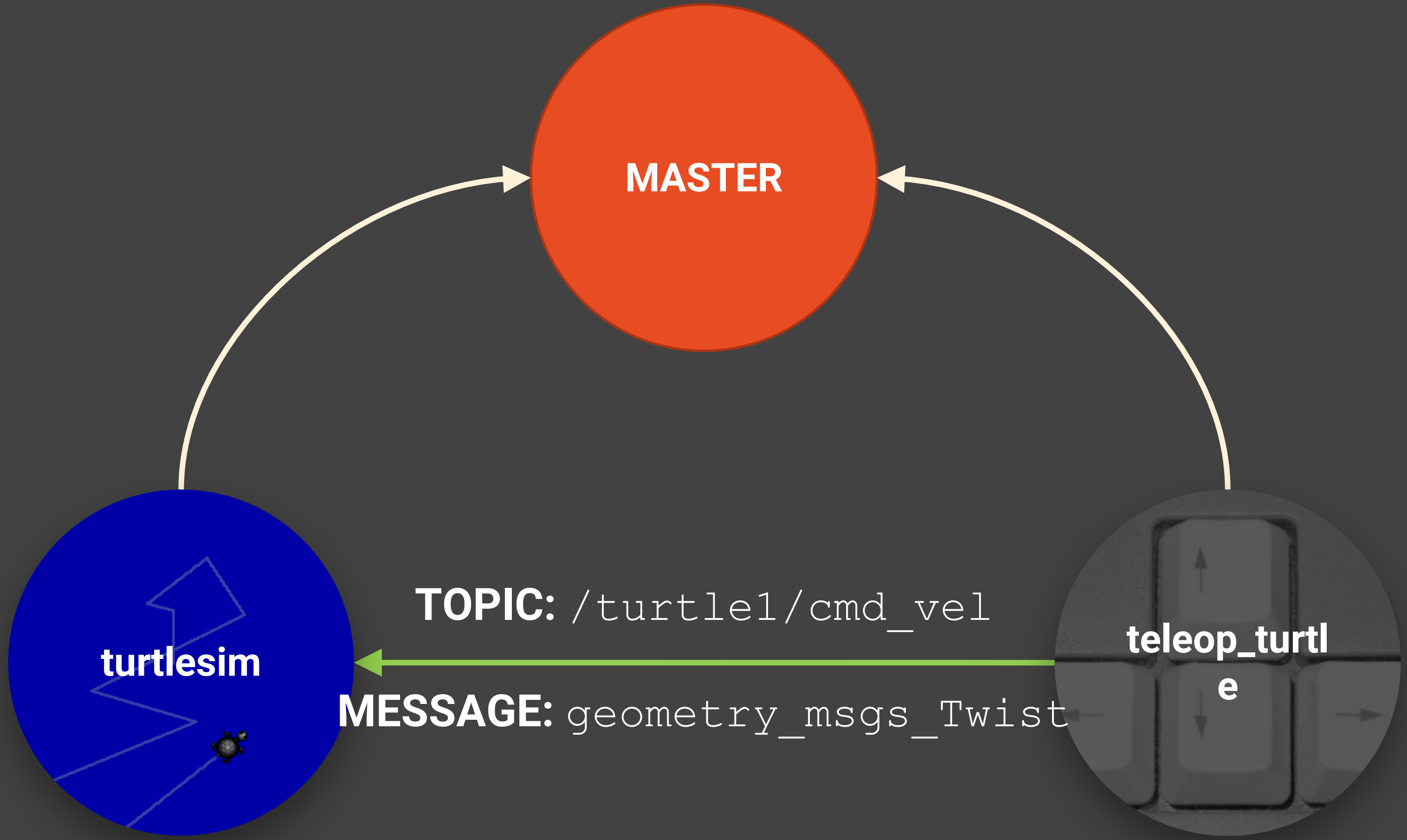
MASTER

Response

turtlesim

teleop_turtl
e

ROS Workflow – Turtlesim



MASTER
turtlesim
teleop_turtl
e
TOPIC: /turtle1/cmd_vel
MESSAGE: geometry_msgs_Twist