Autonome Mobile Systeme

Teil III.1

Einführung in ROS

WS2018

Hans Vollbrecht

objectives of this chapter

- a working definition of what ROS is
- the basic concepts of ROS
  - Messages, services, actions
- the core components of ROS
- the application libraries: an very coarse overview
- an idea of what I can expect ROS to do for me, what do I have to do by myself using Python (or C++)
- a guide to study material

what is ROS?

ROS (Robot Operating System)
is an open-source meta-operating system for robots

„meta-operating system" in the sense that

- it is built on top of an existing Unix-based operating system
- it performs some tasks typical for operating systems:
  - task scheduling, loading, registering, process communication, and monitoring
- provides a virtualization layer between applications and distributed computing resources („middleware") typical for a robot

ROS is a supporting system for controlling a robot and sensors with a hardware abstraction, and for developing robot applications based on existing conventional operating systems.

from: ROS Robot Programming, by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

what is ROS?

very brief history:

– 2007-2010: ROS 1.0, by U.S. robot company Willow Garage
– today: ROS is the dominating robot software platform

other robot SW developing and runtime platforms:

- OpenRTM,
- OPRoS,
- Player,
- YARP,
- Orocos,
- CARMEN,
- Orca,
- MOOS,
- Microsoft Robotics Studio

from: ROS Robot Programming, by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

what is ROS?

objectives of ROS:

– build the development environment that allows robotic software development to collaborate on a global level

– as such, maximize code reuse in the robotics research and development

– form an ecosystem that distributes packages developed by users

from: ROS Robot Programming, by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

what is ROS?

- **distributed processes**:
  - it is programmed in the form of the minimum units of executable processes (nodes)
  - each process runs independently and exchanges data systematically
- **package management**:
  - multiple processes having the same purpose are managed as a package so that it is easy to use and develop, as well as convenient to share, modify, and redistribute
- **public repository**:
  - each package is made public to the developer’s preferred public repository (e.g., GitHub) and specifies their license
- **API for multiple languages**:
  - when developing a program that uses ROS, ROS is designed to simply call an API and insert it easily into the code being used. ROS programming is not much different from C++ or Python or Java, except for communication between functions

from: ROS Robot Programming, by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

Ros components

FIGURE 2-3 Components of ROS[3]

from: ROS Robot Programming, by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

ROS Versions

ROS Versions:

Distro	Release Date	Poster	Symbol	EOL Date
Lunar Loggerhead	2017.05.23			2019.05
Kinetic Kame **(Recommended)**	2016.05.23			2021.04 (Xenial EOL)
Jade Turtle	2015.05.23			2017.05
Indigo Igloo	2014.07.22			2019.04 (Trusty EOL)

Turtlebot3 has been „tried out“ at FHV for:

– Raspian on robots Raspberry PI 3

– Ubuntu 16.0.4 Xenial

– ROS Kinetic Kame

also recommended until 2019 by:
ROS Robot Programming, by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

Ros Terminology

ROS Master

- the master acts as a **name server** for node-to-node connections and message communication.
- the command **roscore** is used to run the master

Node

- a node refers to the smallest unit of process running in ROS. Think of it as one executable program
- can run on a different processor than that of the ROS Master
- nodes communicate with each other by messages and TCP/IP

from: ROS Robot Programming, by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

Ros Terminology

Package

- the basic unit of ROS.
  - ROS applications are developed on a package basis
  - a package contains either a configuration file to launch other packages, or nodes
  - a package also contains all the files necessary for running the package, including ROS dependency libraries for running various processes, datasets

Meta-Package

- a set of packages that have a common purpose
- for example, the “Navigation” metapackage consists of 10 packages including AMCL, DWA, EKF, and map_server

there are about 1,600 packages for ROS Kinetic

from: ROS Robot Programming, by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

Ros Terminology

Message

- a node sends or receives data from/to other nodes via a message

- messages are structures (as in C) with named components

- component data types: integer, floating point, boolean, string, and arrays of these

- nested message structure that may contain other messages

Ros Terminology

Topic

- the topic is literally like a topic in a conversation. It follows a publisher/subscriber principle:

Publisher Node

- first registers its topic with ROS Master and then starts publishing messages on a topic

Subscriber Nodes

- that want to receive the topic, request information of the publisher node from the ROS Master by the topic name
- based on this information, the subscriber node directly connects to the publisher node to exchange messages as a topic

Ros Terminology

Topic (cont‘d)

– topic communication is an asynchronous communication, useful to transfer certain data: a continuous stream of messages once connected, often used for sensors that periodically transmit data

Ros Terminology

Service

- a service is a synchronous bidirectional communication between the service client that requests a service regarding a particular task and the service server that is responsible for responding to requests

Service Server

- a server (a node) in the service message communication; receives a request as an input and transmits a response as an output
- both request and response are in the form of messages: input parameters and result

Service Client

- a client (node) in the service message communication that requests service to the server and receives a response as an input

Ros Terminology

Action

– an action is another message communication method used for an asynchronous bidirectional communication

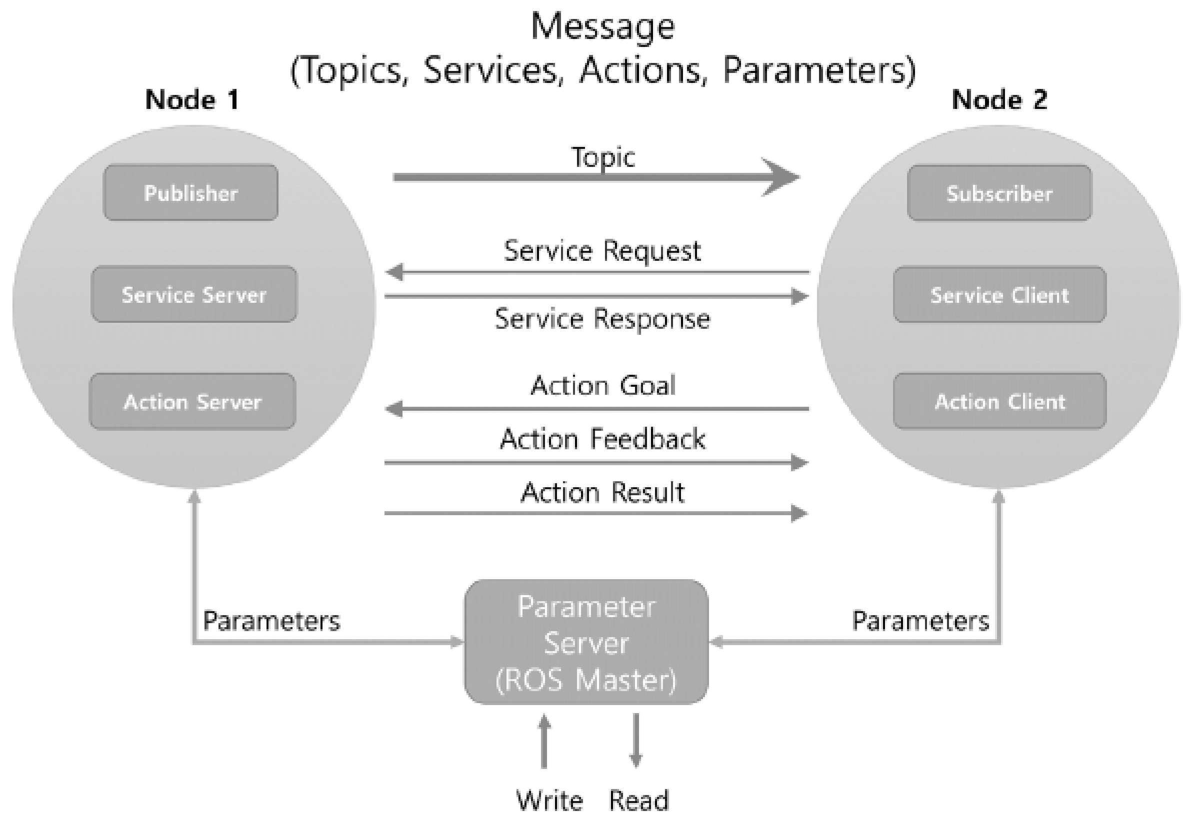
– action is used where it takes longer time to respond after receiving a request, and where intermediate responses are required until the result is returned

– client node sends parameters and a goal

– the action server sends feedback messages and finally a goal message

– Client node, when receiving a feedback message, transmits follow up instructions or cancel instruction to action server

Ros Terminology



Type	Features		Description
Topic	Asynchronous	Unidirectional	Used when exchanging data continuously
Service	Synchronous	Bi-directional	Used when request processing requests and responds current states
Action	Asynchronous	Bi-directional	Used when it is difficult to use the service due to long response times after the request or when an intermediate feedback value is needed

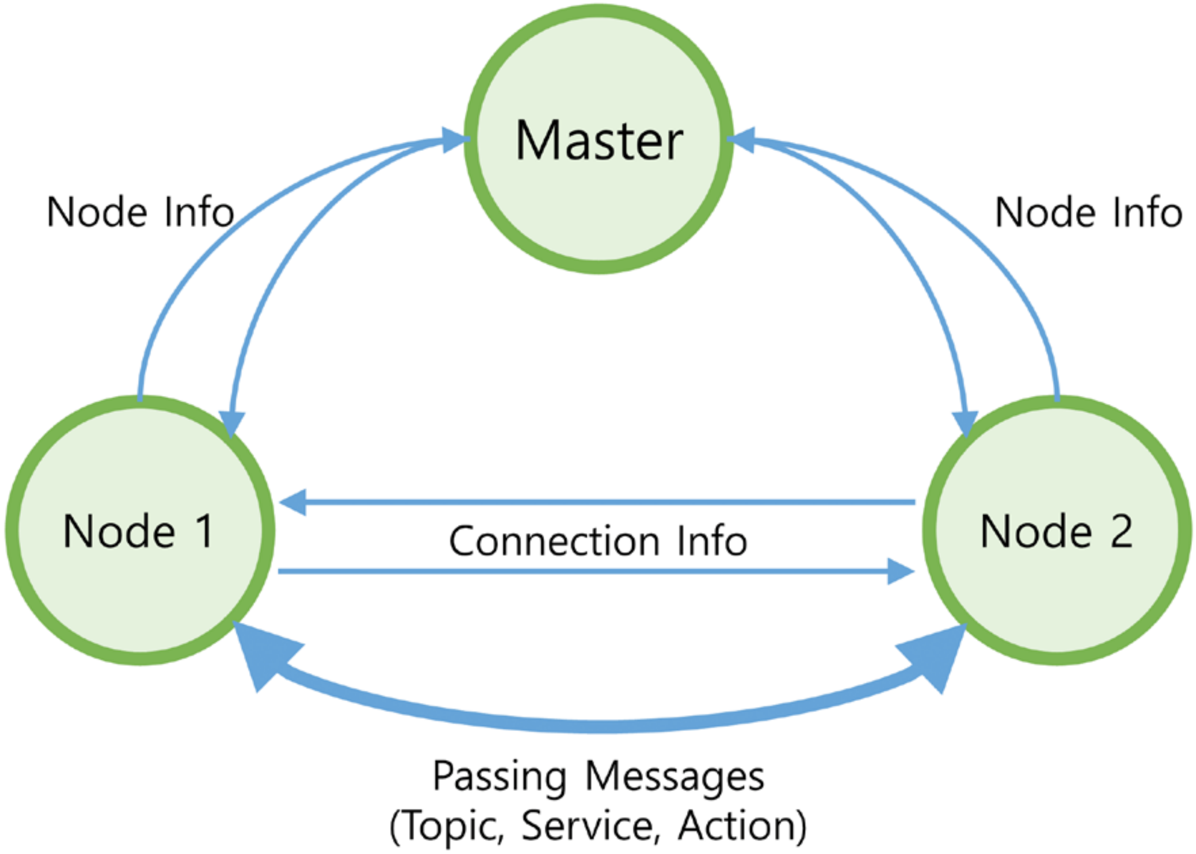
Ros Terminology

Catkin

- “catkin” refers to the build system of ROS
- the build system basically uses CMake (Cross Platform Make)
- the build environment is described in the ‘CMakeLists.txt’ file in the package folder
- CMake was modified in ROS to create a ROS-specific build system

Message Communication

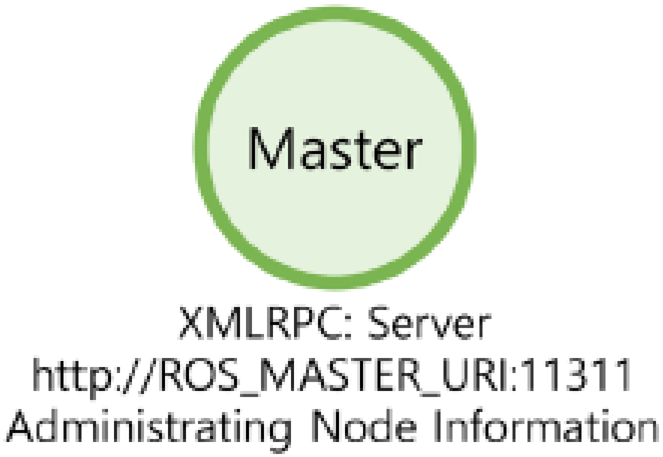
general connection establishment



Message Communication

step 1: start the ros master

```
$ roscore
```

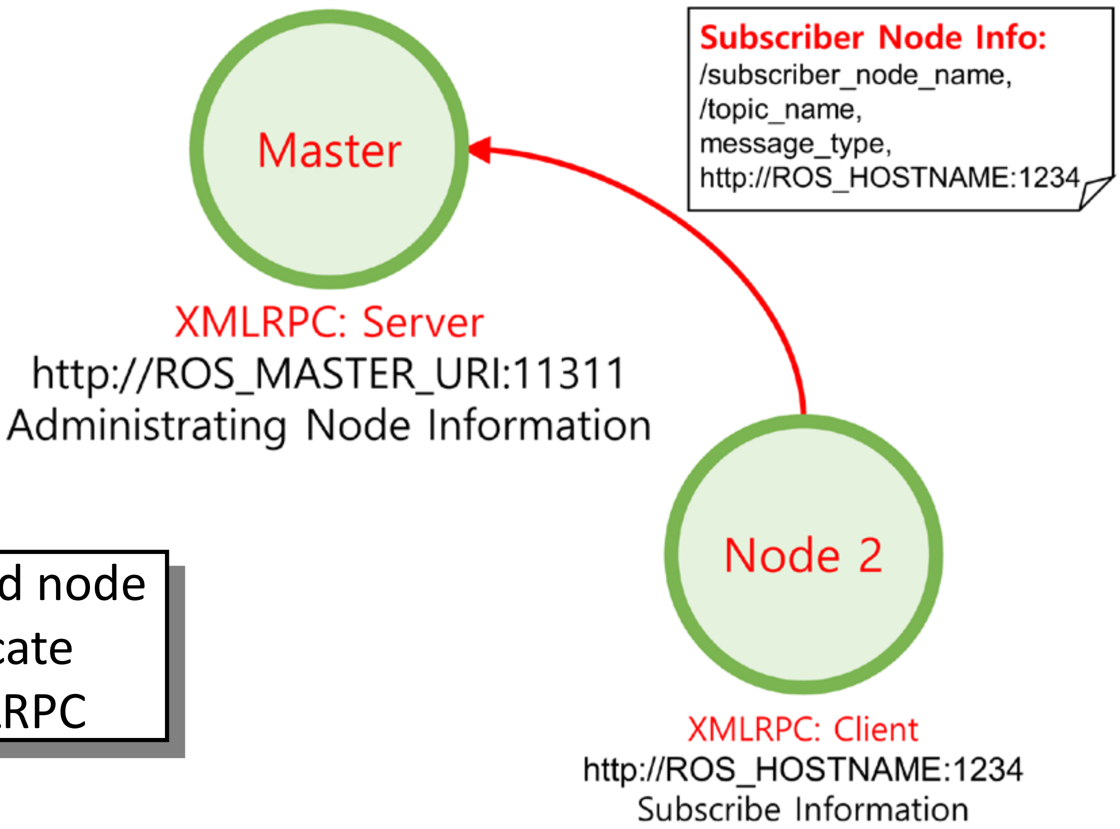


XMLRPC (XML remote procedure call)
a HTTP-based protocol for procedure calls in a distributed system

Message Communication

step 2: launch a topic subscriber node => registration on the master

```
$ rosrun PACKAGE_NAME NODE_NAME
$ roslaunch PACKAGE_NAME LAUNCH_NAME
```



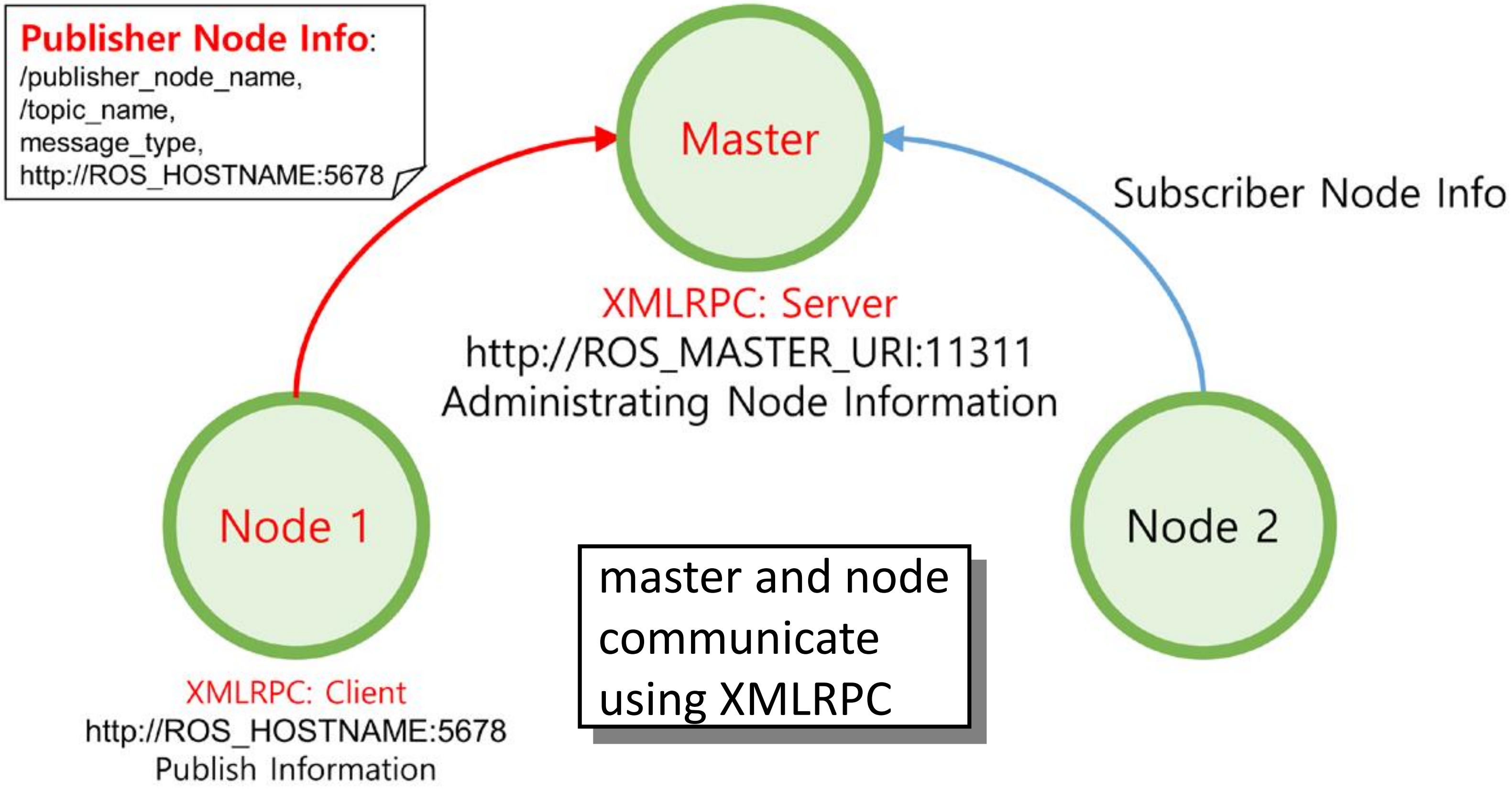
master and node communicate using XMLRPC

Message Communication

step 3: launch a topic publisher node => registration on the master

```
$ rosrun PACKAGE_NAME NODE_NAME
$ roslaunch PACKAGE_NAME LAUNCH_NAME
```

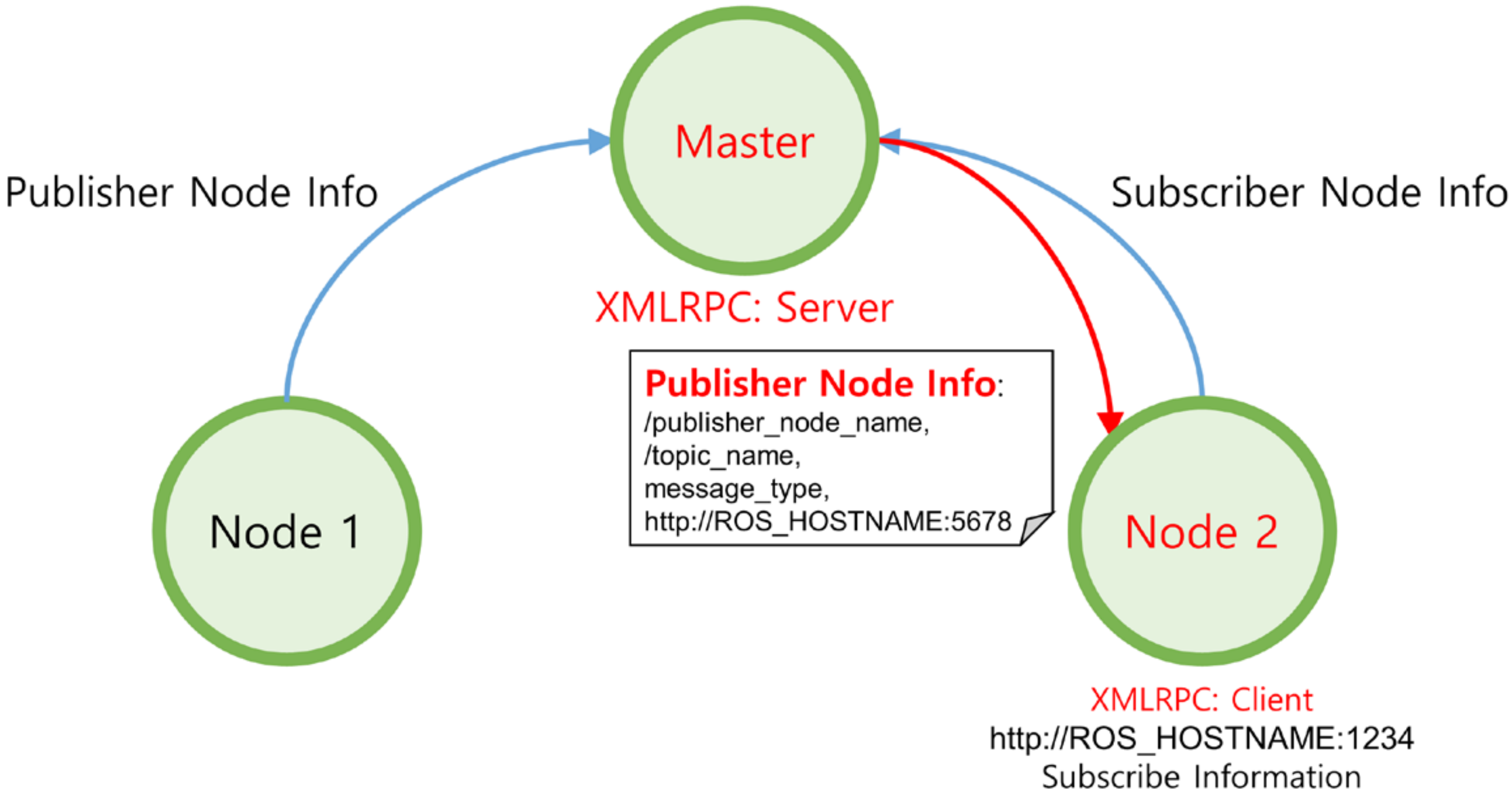
identical to subscriber
(difference is in code of the node



master and node communicate using XMLRPC

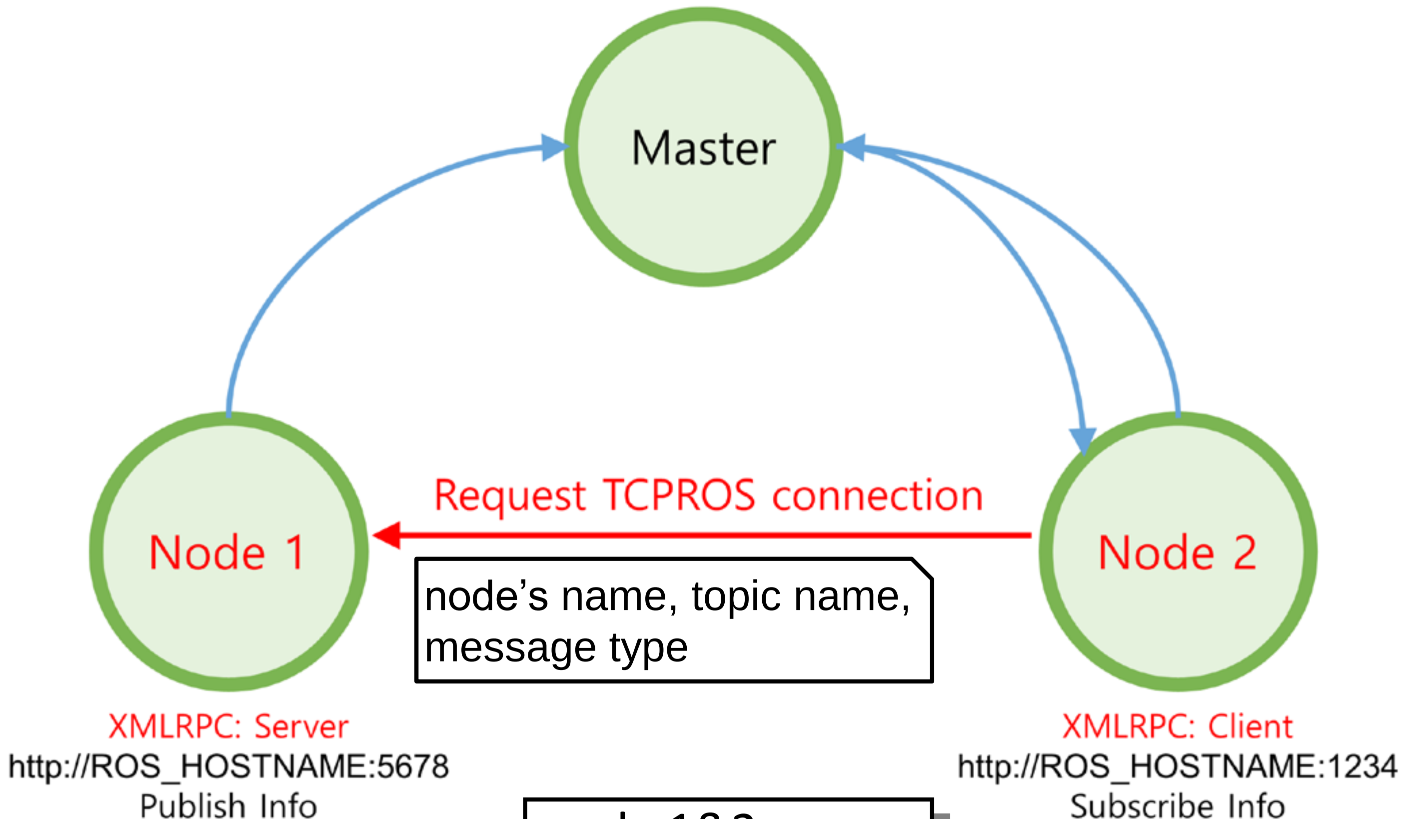
Message Communication

step 3.1: inform topic subscriber about the topic publisher node



Message Communication

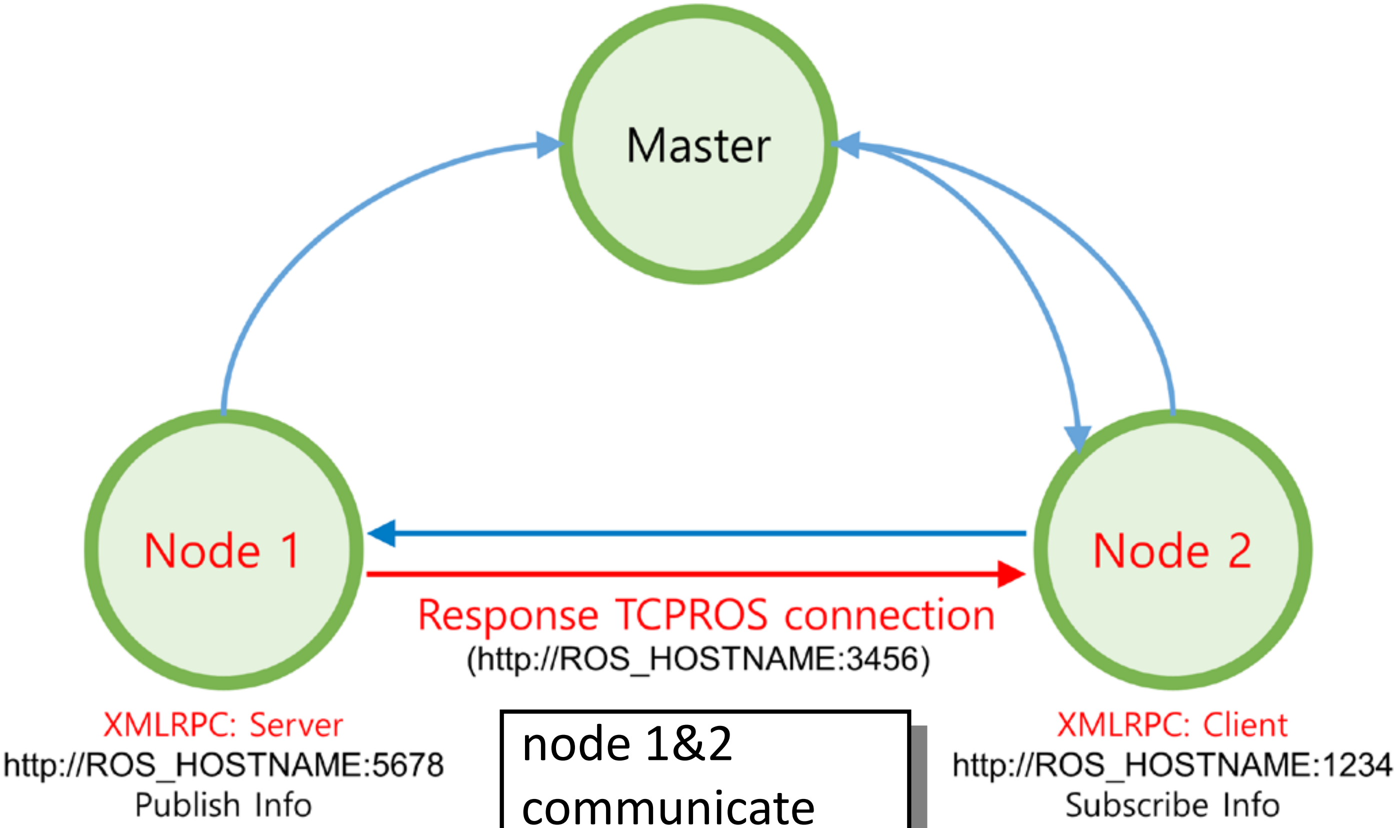
step 3.2: subscriber request to publisher for a topic connection



node 1&2
communicate
using XMLRPC

Message Communication

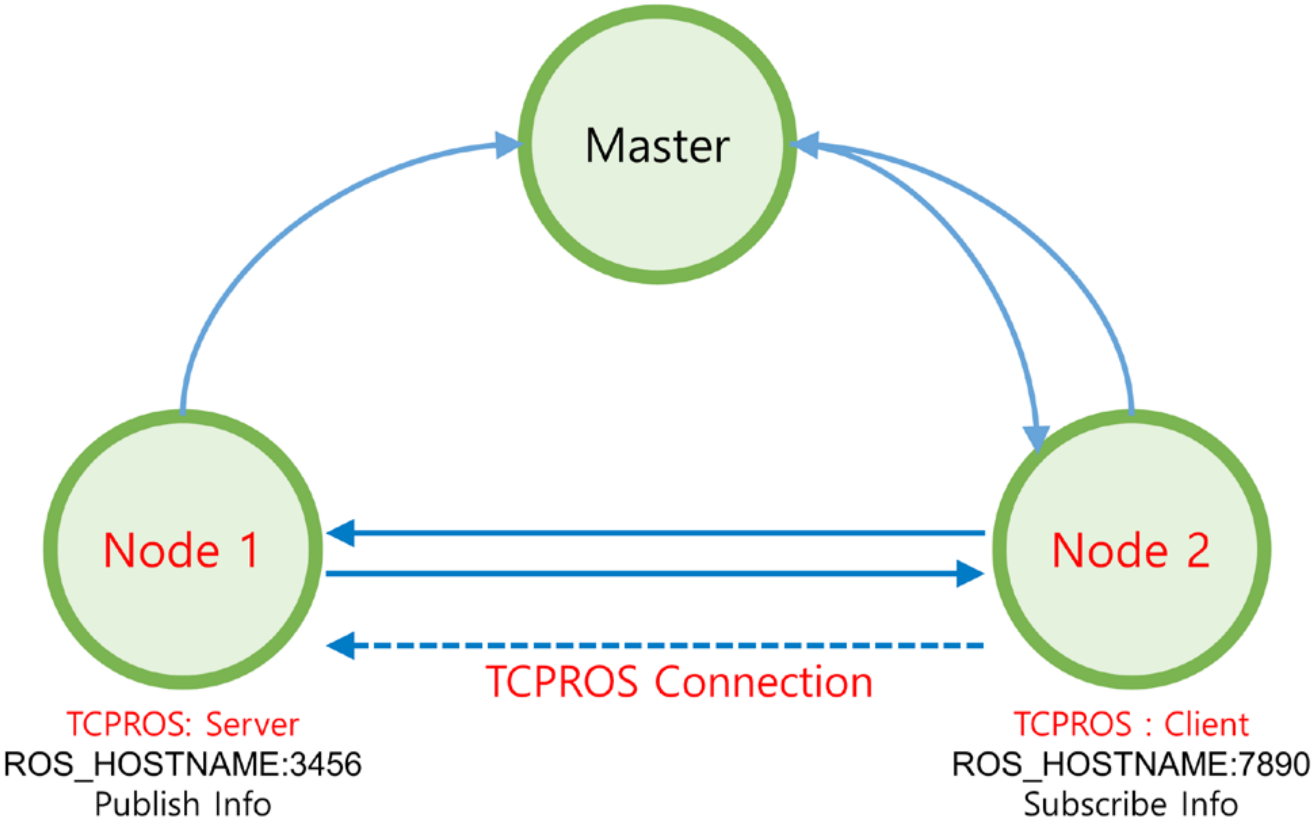
step 3.3: publisher sends URI and port number for a topic connection between publisher and subscriber



node 1&2 communicate using XMLRPC

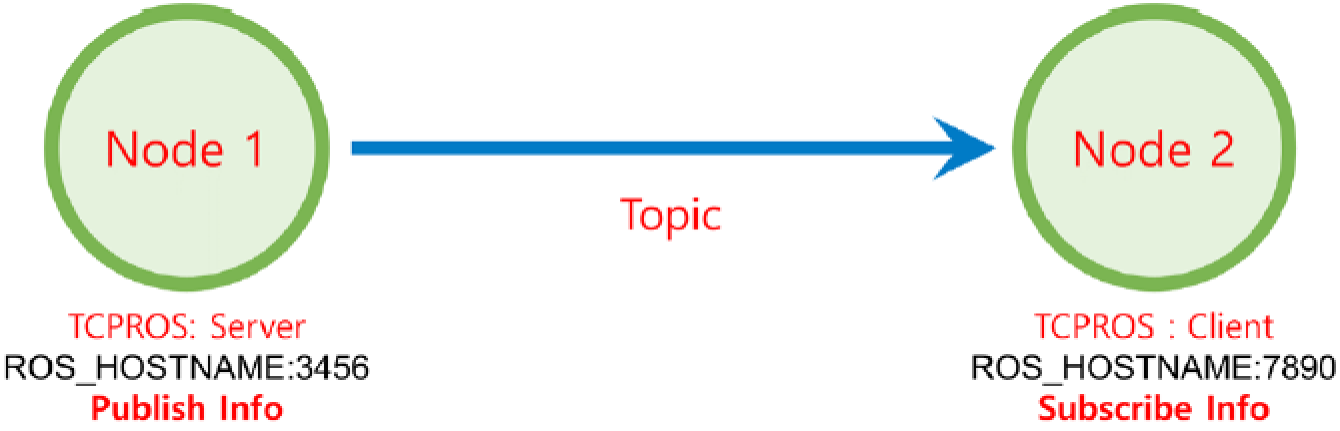
Message Communication

step 3.4: subscriber and publisher open a TCPROS connection



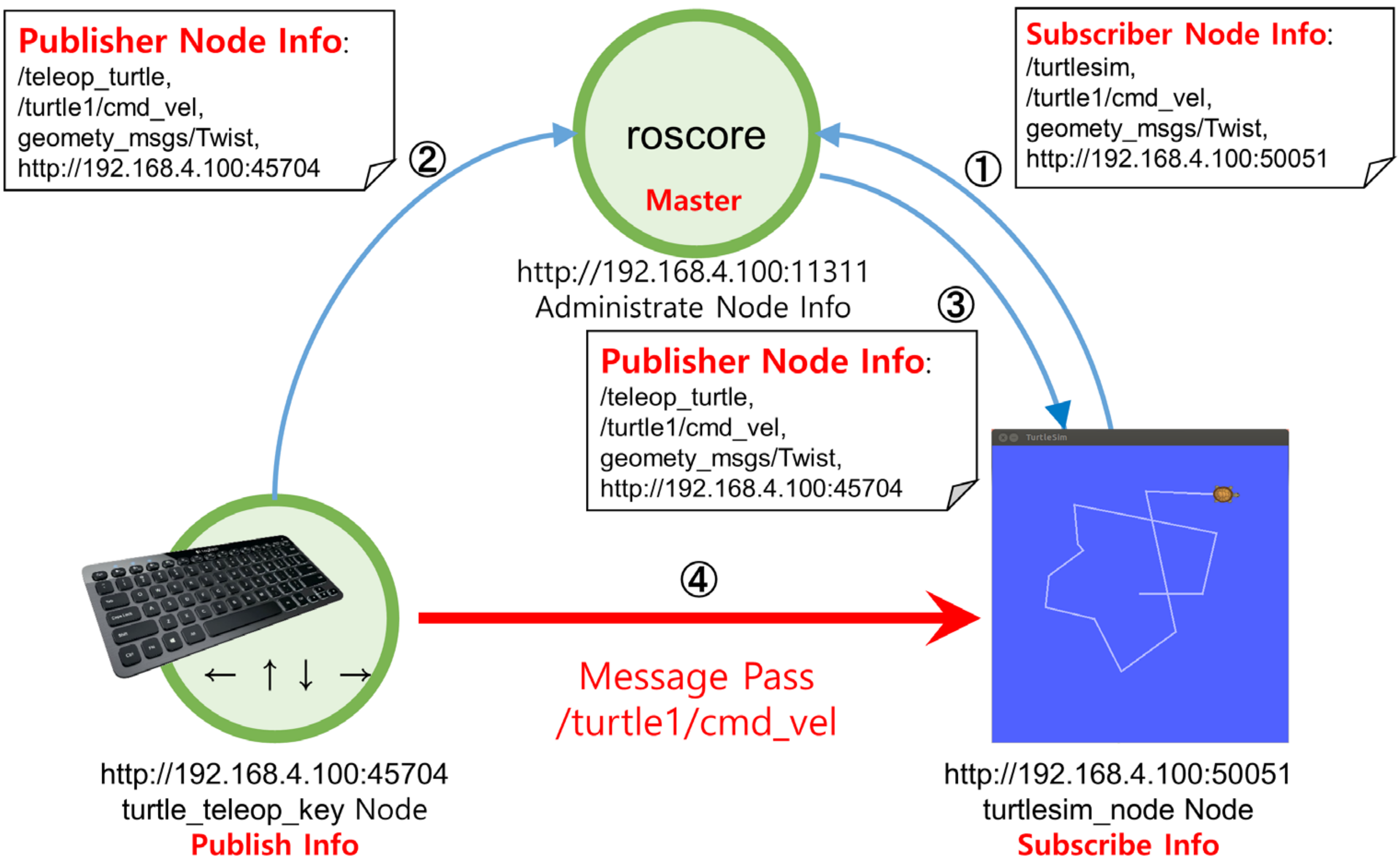
Message Communication

step 4: publisher starts sending messages over TCPROS connection



nodes 1&2 communicate directly without the Ros Master!

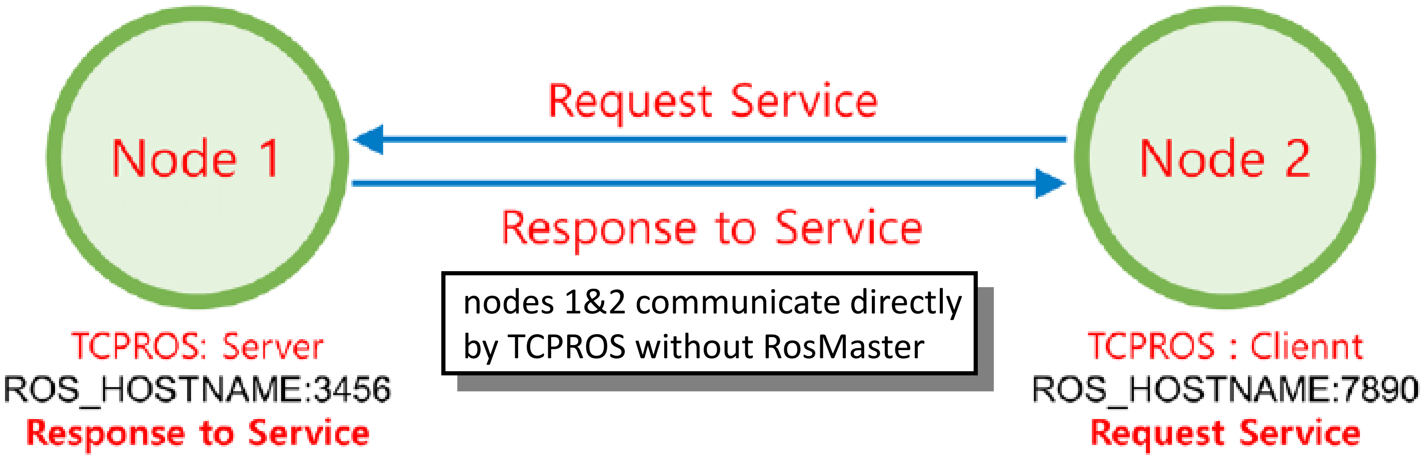
Message Communication



Message Communication

service connections are established much the same as topic conn.s

step 3.5: client node sends a service request (by message) to server node
step 3.6: server node executes the service and then sends a response message to client node

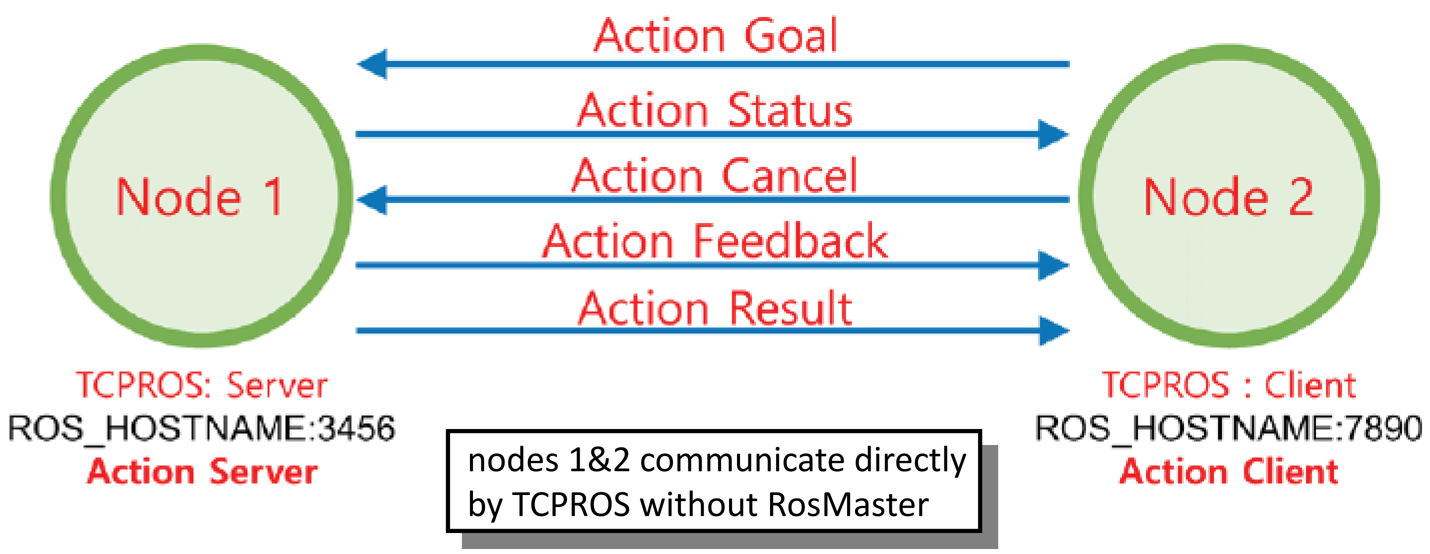


unlike the topic, the service terminates connection after successful request and response.

Message Communication

service connections are established much the same as topic conn.s, but thereafter, the message exchange differs:

step 3.5: client node sends goal request (by message)
step 3.6: server executes goal service
steps 3.7…: server sends status messages
after step 3.5: clients sends cancel message: service end, close conn.
after 3.5: server sends result message: service end, close connection



.

ROS tools

- RViz 3D visualization tool
- **rqt** extendable tool for visualization of diverse robot information
- rqt_image_view Image display tool (a type of rqt)
- rqt_graph A tool that visualizes the correlation between nodes and messages as a graph (a type of rqt)
- rqt_plot 2D data plot tool (a type of rqt)
- rqt_bag GUI-based bag data analysis tool (a type of rqt)
- **gazebo** a robot simulator (like Vrep) for ROS (has a Turtlebot model)

ROS tools

- RViz 3D visualization tool

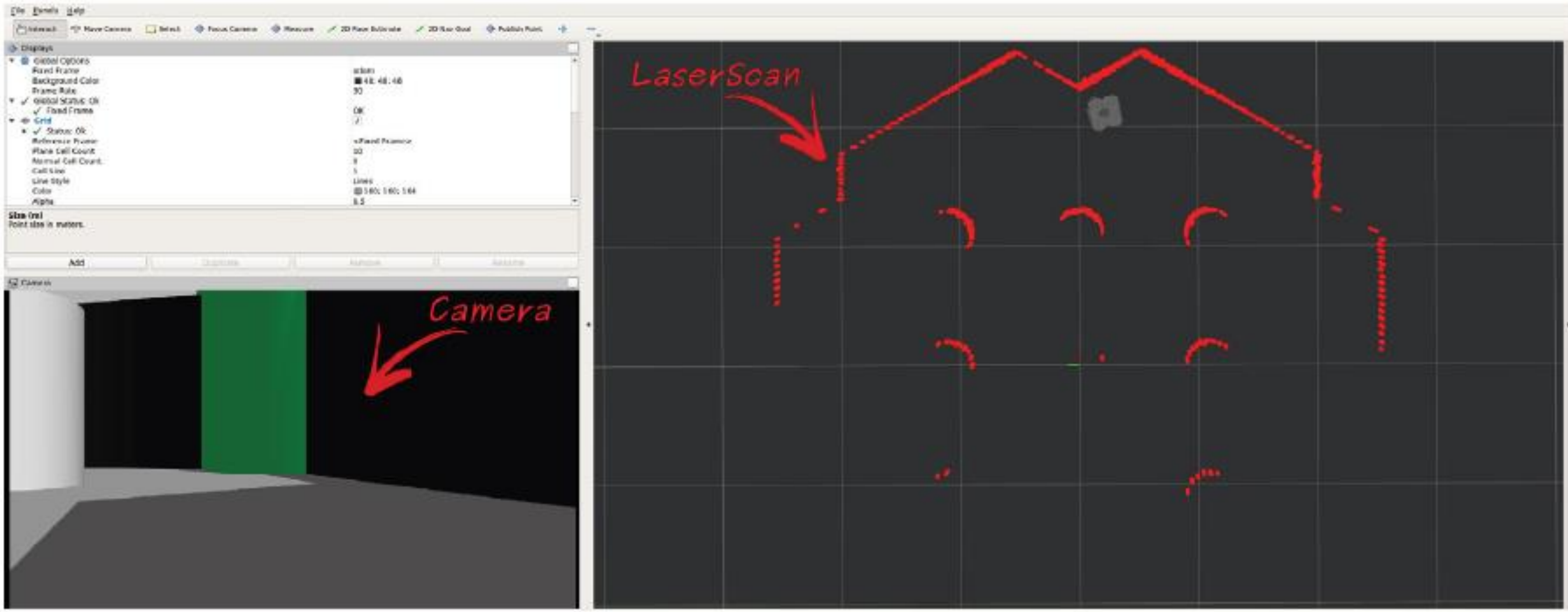


FIGURE 10-19 Visualized virtual LiDAR data and camera image in RViz

Rviz is a powerful visualization tool, but it is not a simulator!

ROS tools

- **RViz** 3D visualization tool

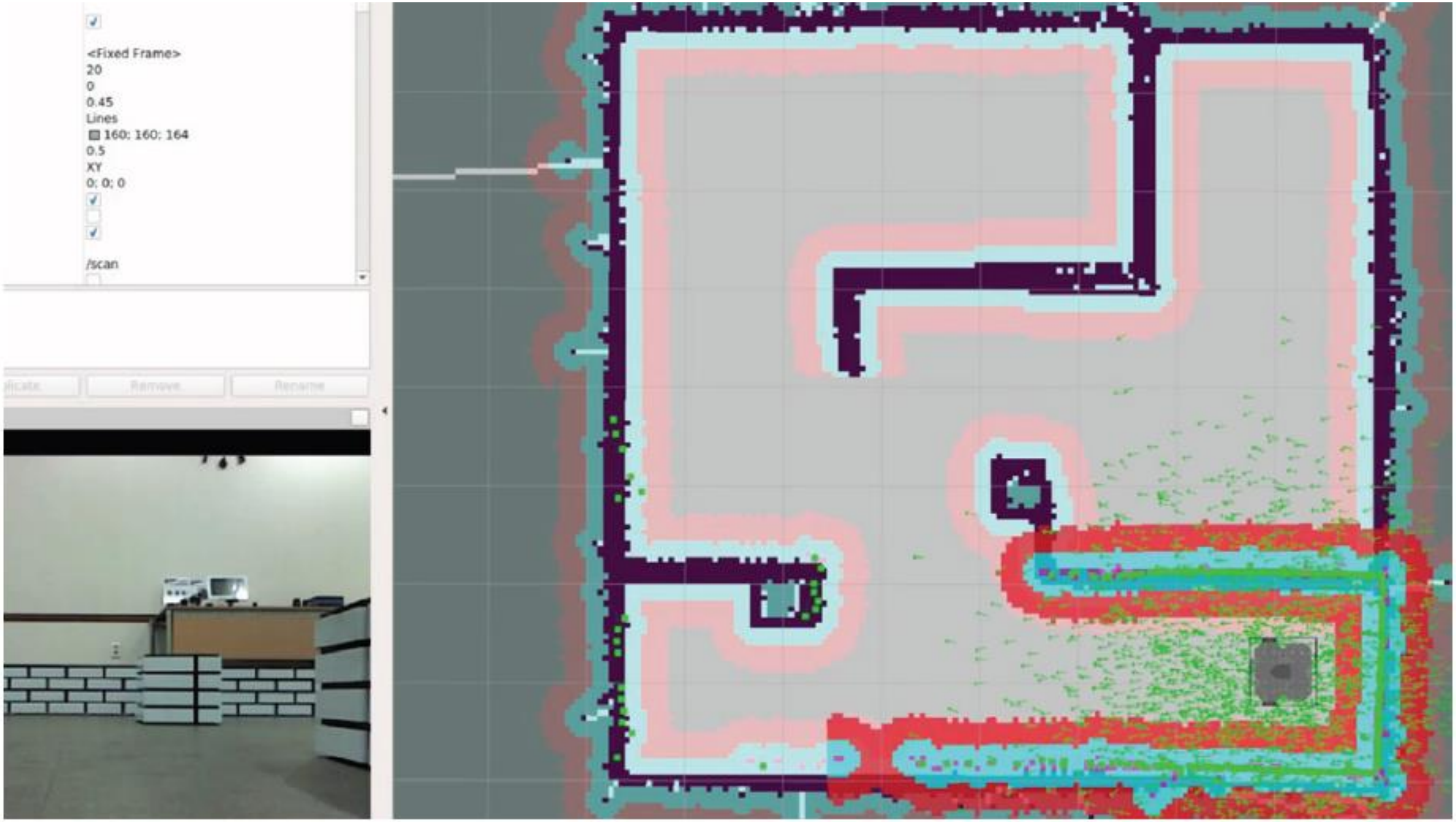


FIGURE 11-15 Particles visible in RViz (green arrows around the robot)

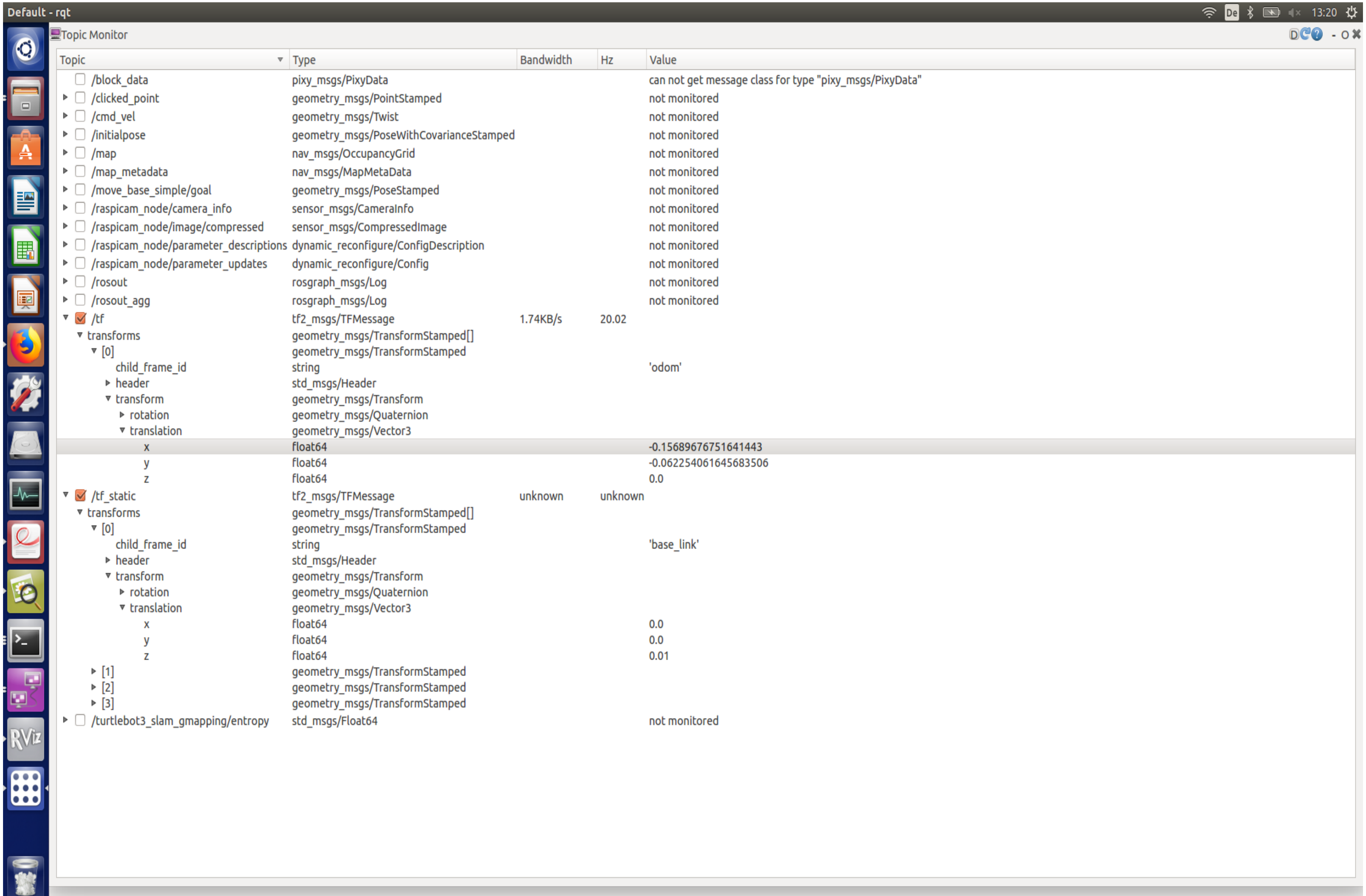
ROS tools

rqt is a software framework of ROS that implements the various GUI tools in the form of plugins.

One can run all the existing GUI tools as dockable windows within **rqt**!

The tools can still run in a traditional standalone method, but **rqt** makes it easier to manage all the various windows on the screen at one moment.

ROS tools



Default - rqt
13:20
Topic Monitor
Topic | Type | Bandwidth | Hz | Value
/block_data pixy_msgs/PixyData can not get message class for type "pixy_msgs/PixyData"
/clicked_point geometry_msgs/PointStamped not monitored
/cmd_vel geometry_msgs/Twist not monitored
/initialpose geometry_msgs/PoseWithCovarianceStamped not monitored
/map nav_msgs/OccupancyGrid not monitored
/map_metadata nav_msgs/MapMetaData not monitored
/move_base_simple/goal geometry_msgs/PoseStamped not monitored
/raspicam_node/camera_info sensor_msgs/CameraInfo not monitored
/raspicam_node/image/compressed sensor_msgs/CompressedImage not monitored
/raspicam_node/parameter_descriptions dynamic_reconfigure/ConfigDescription not monitored
/raspicam_node/parameter_updates dynamic_reconfigure/Config not monitored
/rosout rosgraph_msgs/Log not monitored
/rosout_agg rosgraph_msgs/Log not monitored
/tf tf2_msgs/TFMessage 1.74KB/s 20.02
transforms geometry_msgs/TransformStamped[]
[0] geometry_msgs/TransformStamped
child_frame_id string 'odom'
header std_msgs/Header
transform geometry_msgs/Transform
rotation geometry_msgs/Quaternion
translation geometry_msgs/Vector3
x float64 -0.15689676751641443
y float64 -0.062254061645683506
z float64 0.0
/tf_static tf2_msgs/TFMessage unknown unknown
transforms geometry_msgs/TransformStamped[]
[0] geometry_msgs/TransformStamped
child_frame_id string 'base_link'
header std_msgs/Header
transform geometry_msgs/Transform
rotation geometry_msgs/Quaternion
translation geometry_msgs/Vector3
x float64 0.0
y float64 0.0
z float64 0.01
[1] geometry_msgs/TransformStamped
[2] geometry_msgs/TransformStamped
[3] geometry_msgs/TransformStamped
/turtlebot3_slam_gmapping/entropy std_msgs/Float64 not monitored

ROS tools

- **rqt_graph** tool that visualizes the correlation between nodes and messages as a graph (a type of rqt)

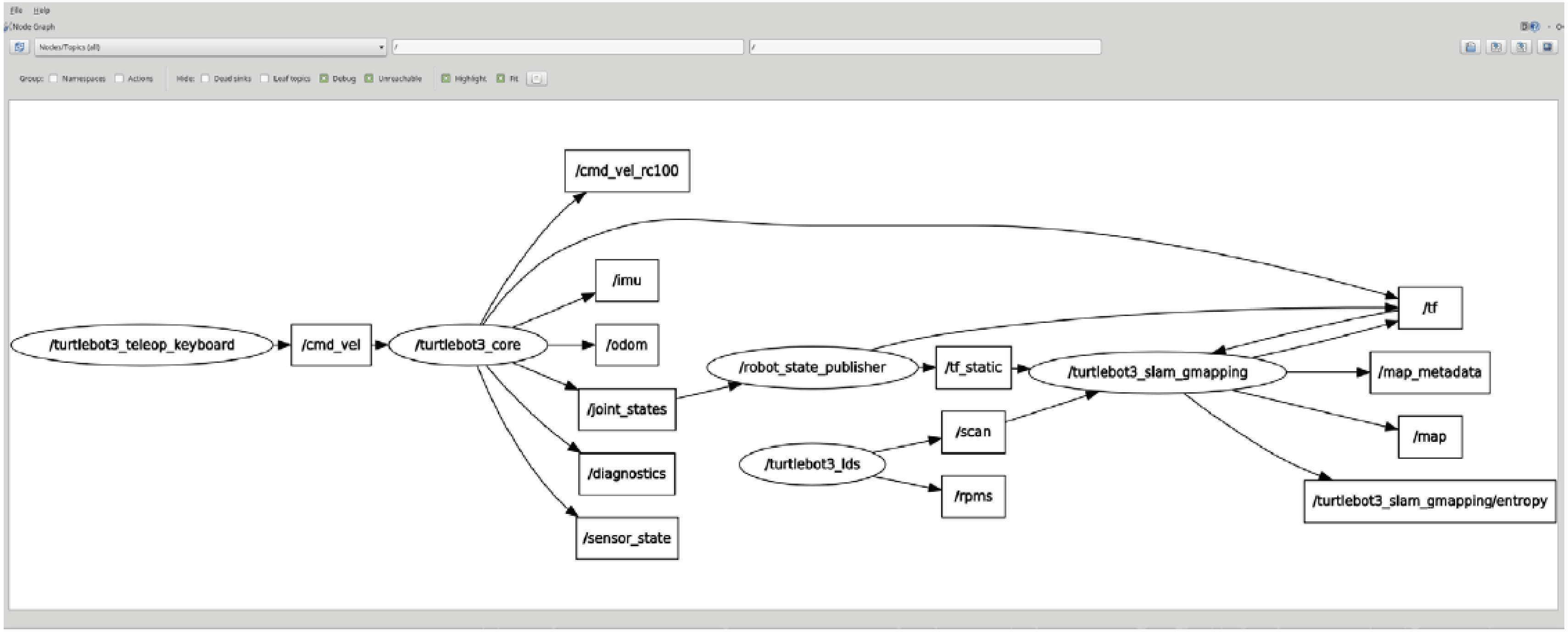


FIGURE 11-6 Nodes and topics required for SLAM

ROS tools

- **rqt_plot** 2D data plot tool (a type of rqt)

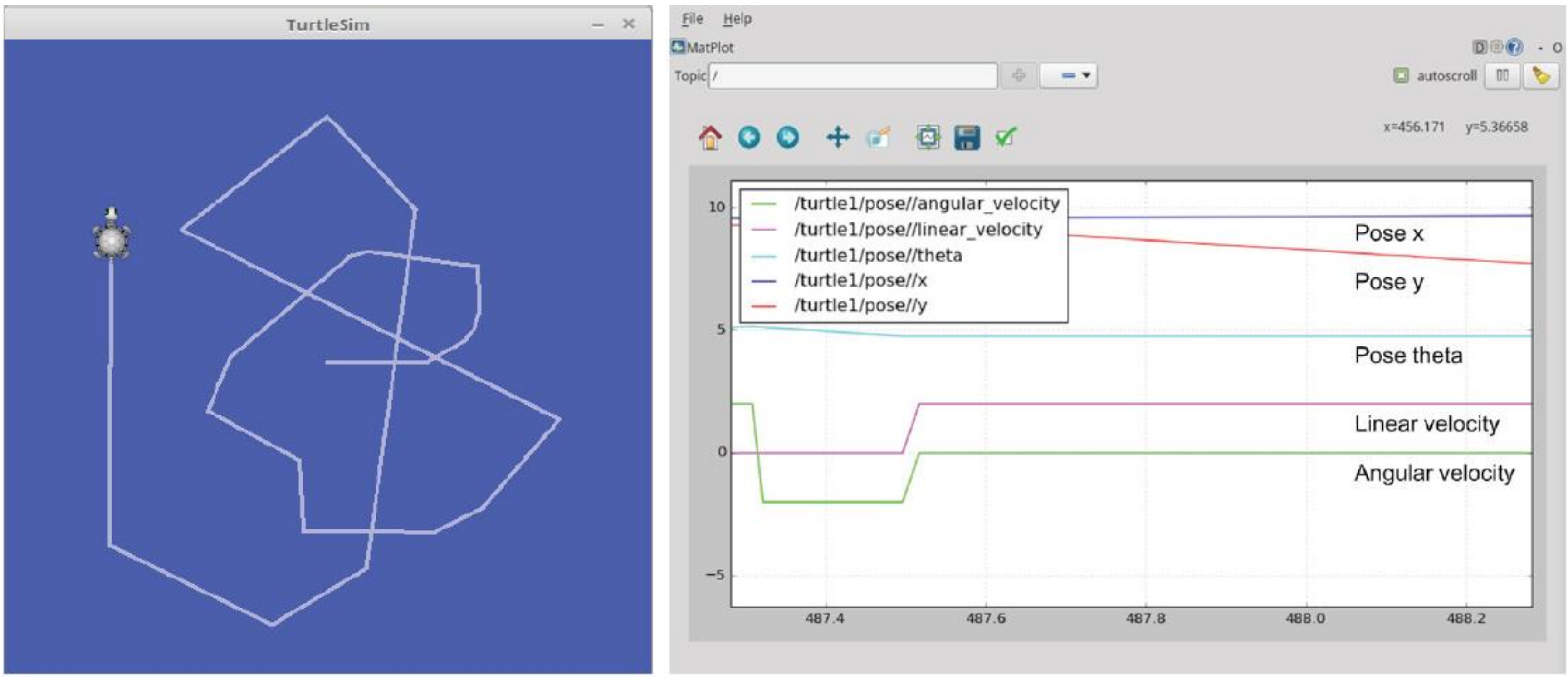


FIGURE 6-13 Example of rqt_plot

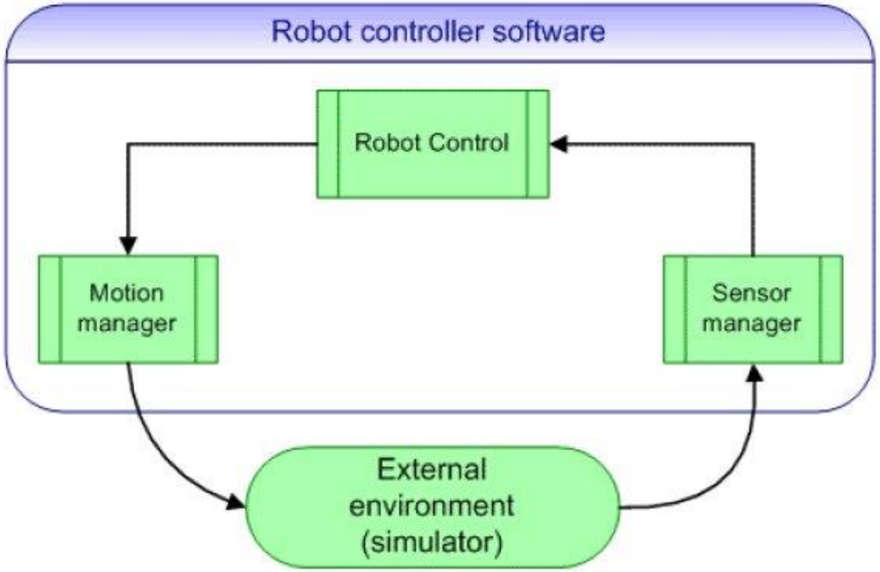
ROS Architektur

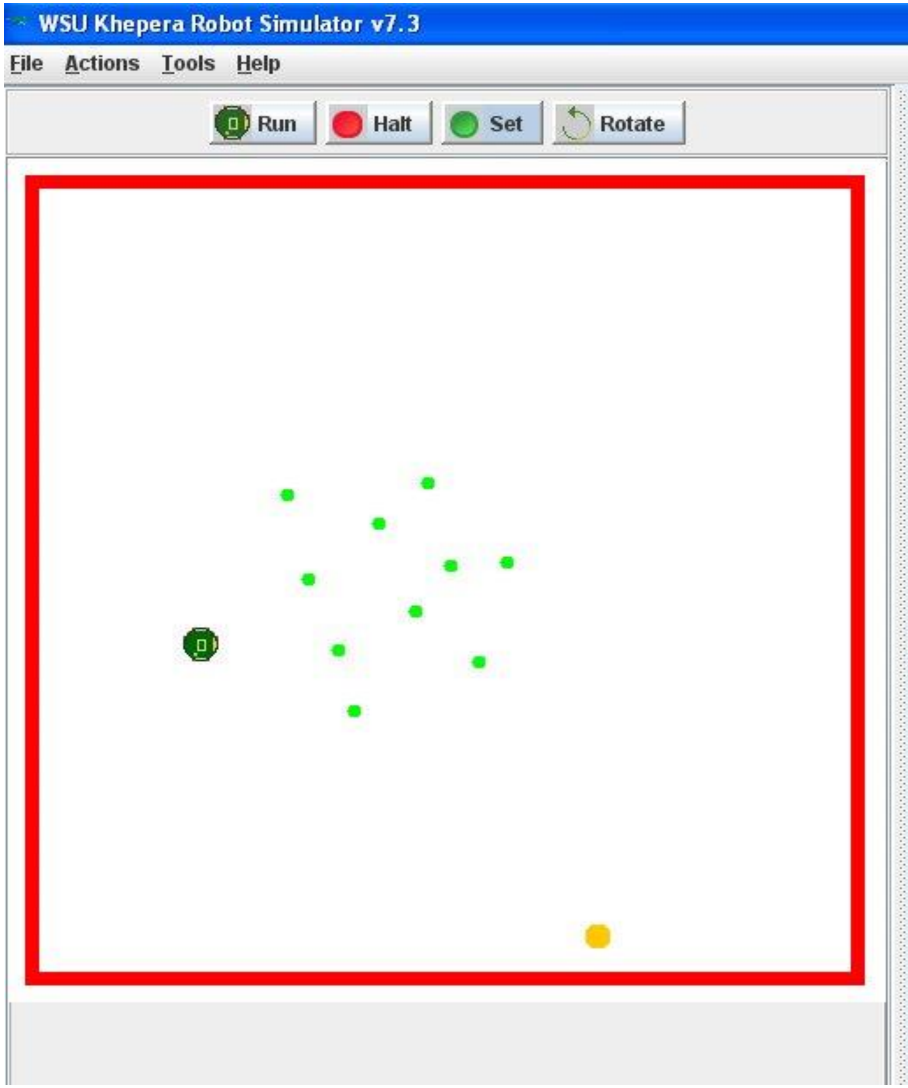
- does ROS represent a SW-architecture? in that case, which?

- what is the relationship of ROS to well-known SW-architectures for autonomopus robots:
  - control-loop
  - subsumption
  - layers

basic architectures for robot control

a control loop architecture for a problem of collecting balls and depositing them near a light source





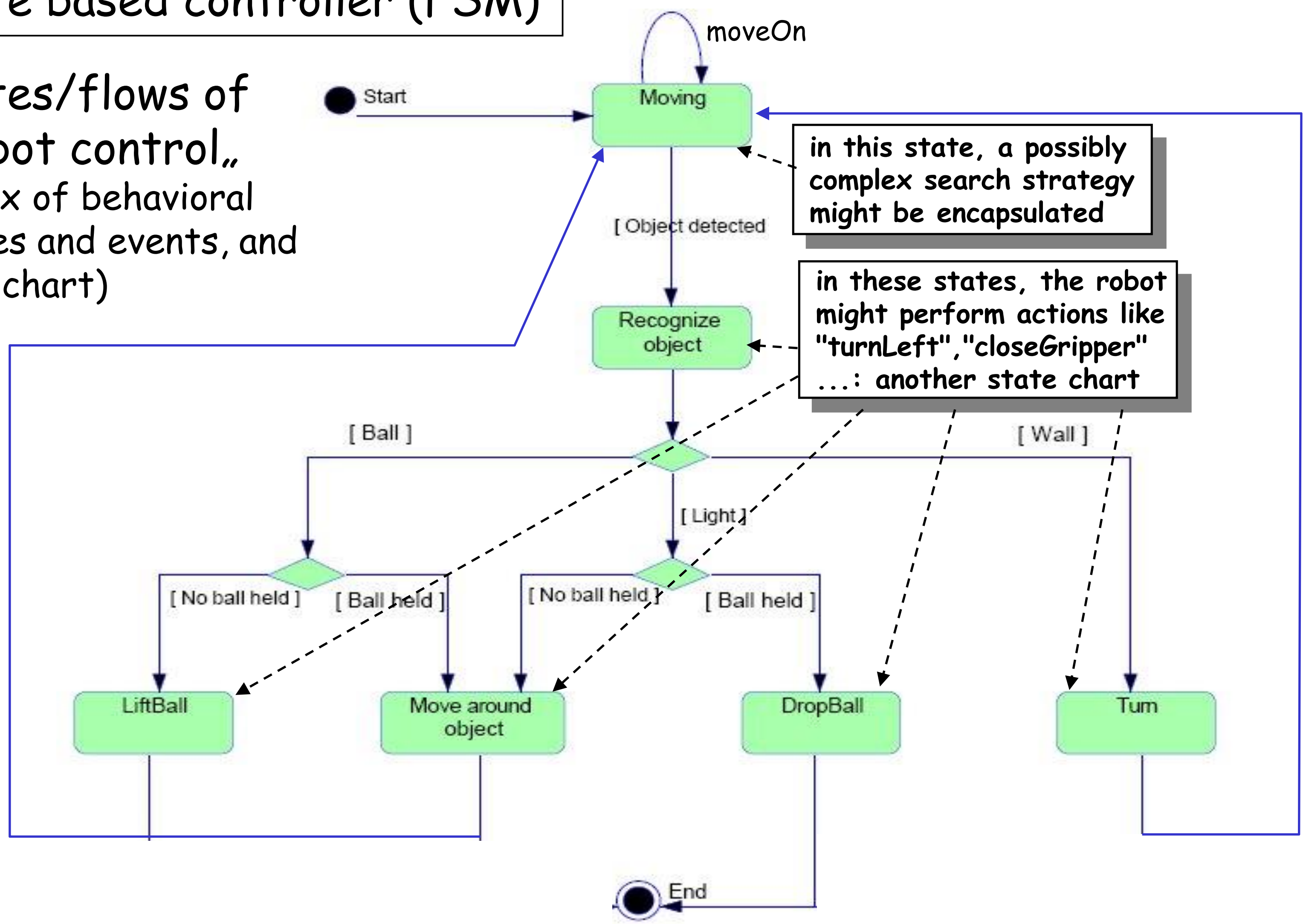
the Khepera Robot with a gripper module

basic architectures for robot control

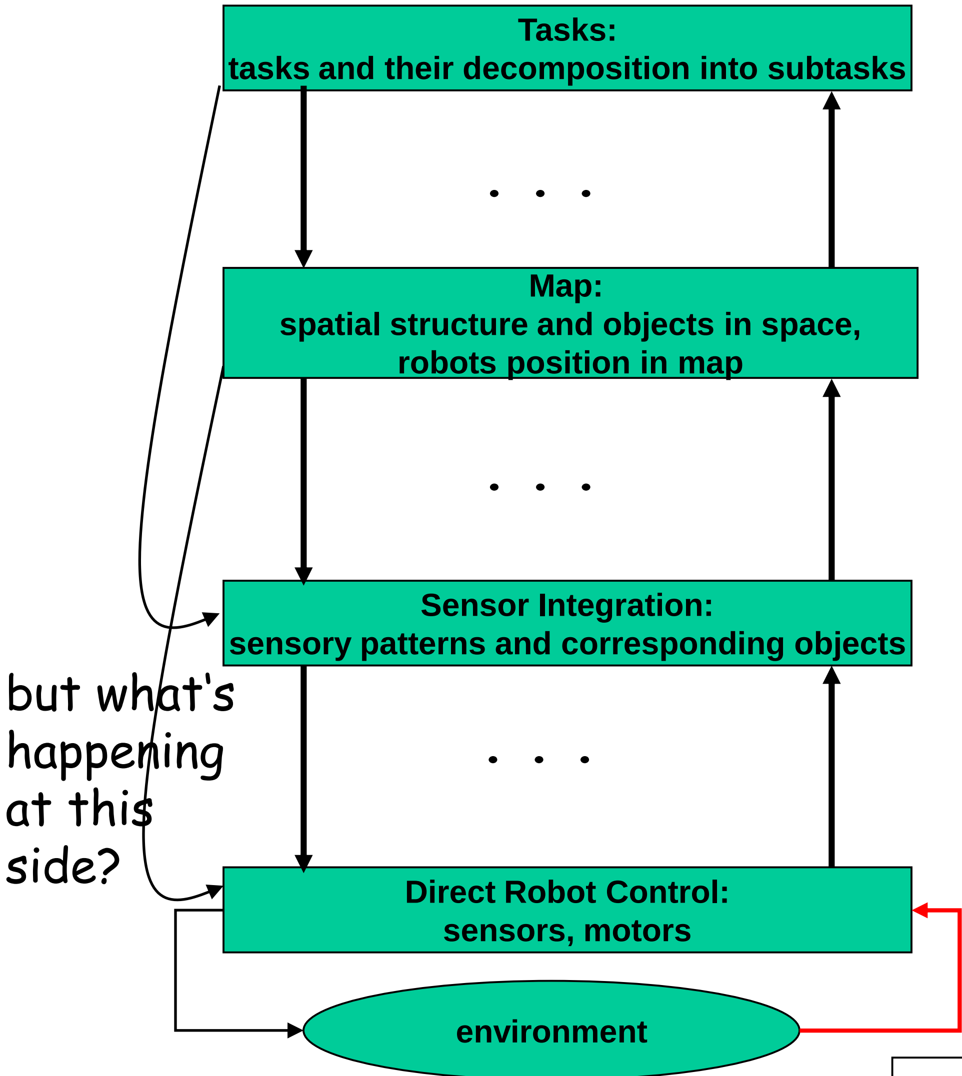
collecting balls and deposit them near a light source

state based controller (FSM)

states/flows of "robot control„
(a mix of behavioral states and events, and flow chart)



robotics example: layered architecture



high conceptual level

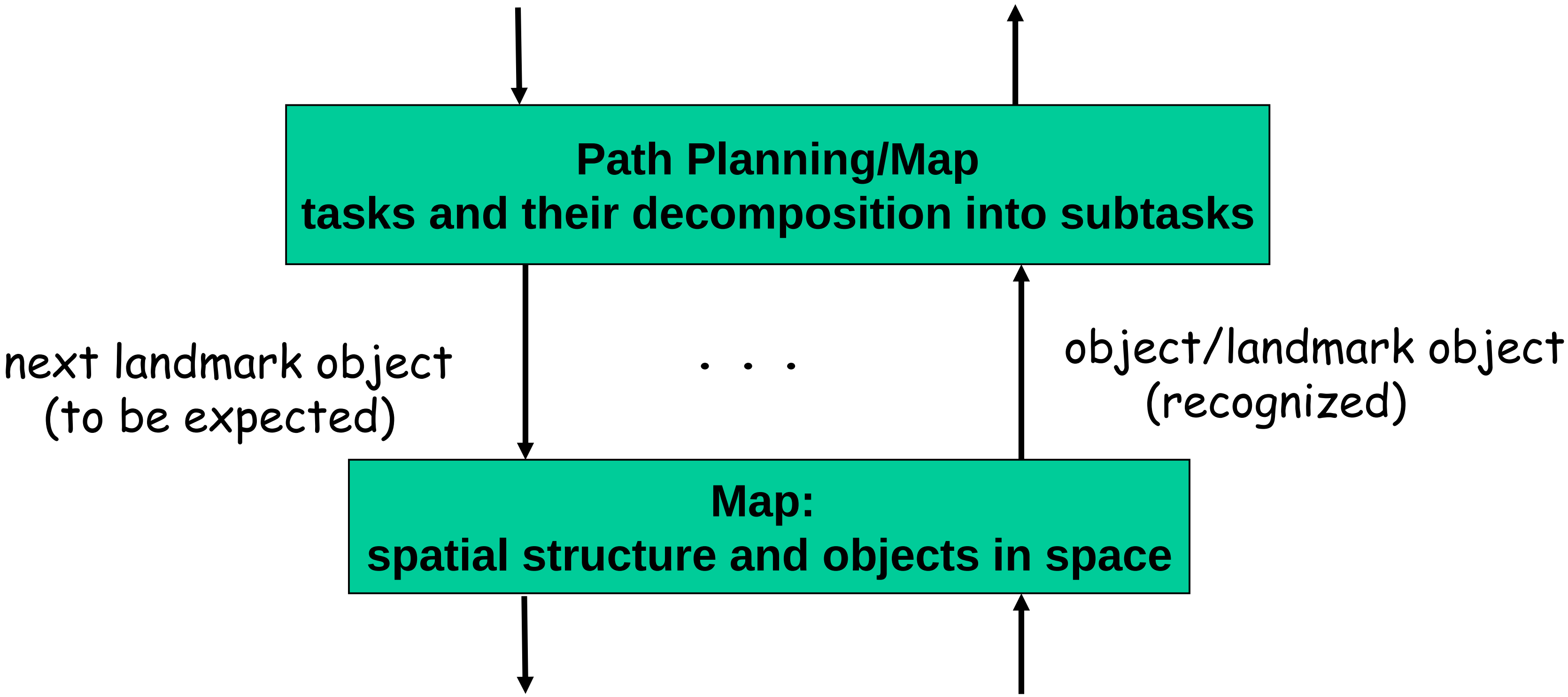
each layer transforms input into output of a higher conceptual level as input for the next higher level

each layer contains a set of concepts at same conceptual level

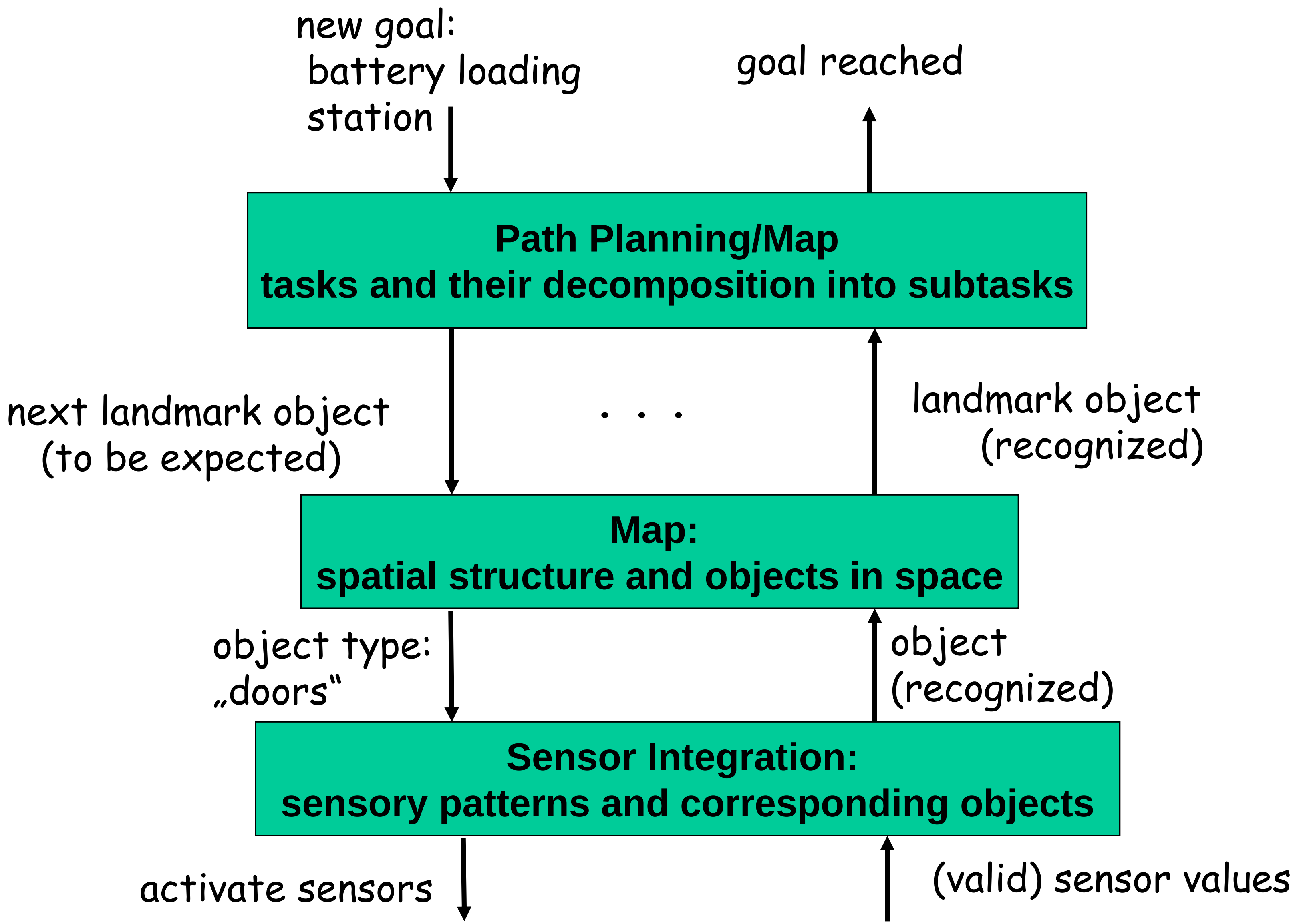
low conceptual level

robot architecture 2: layers

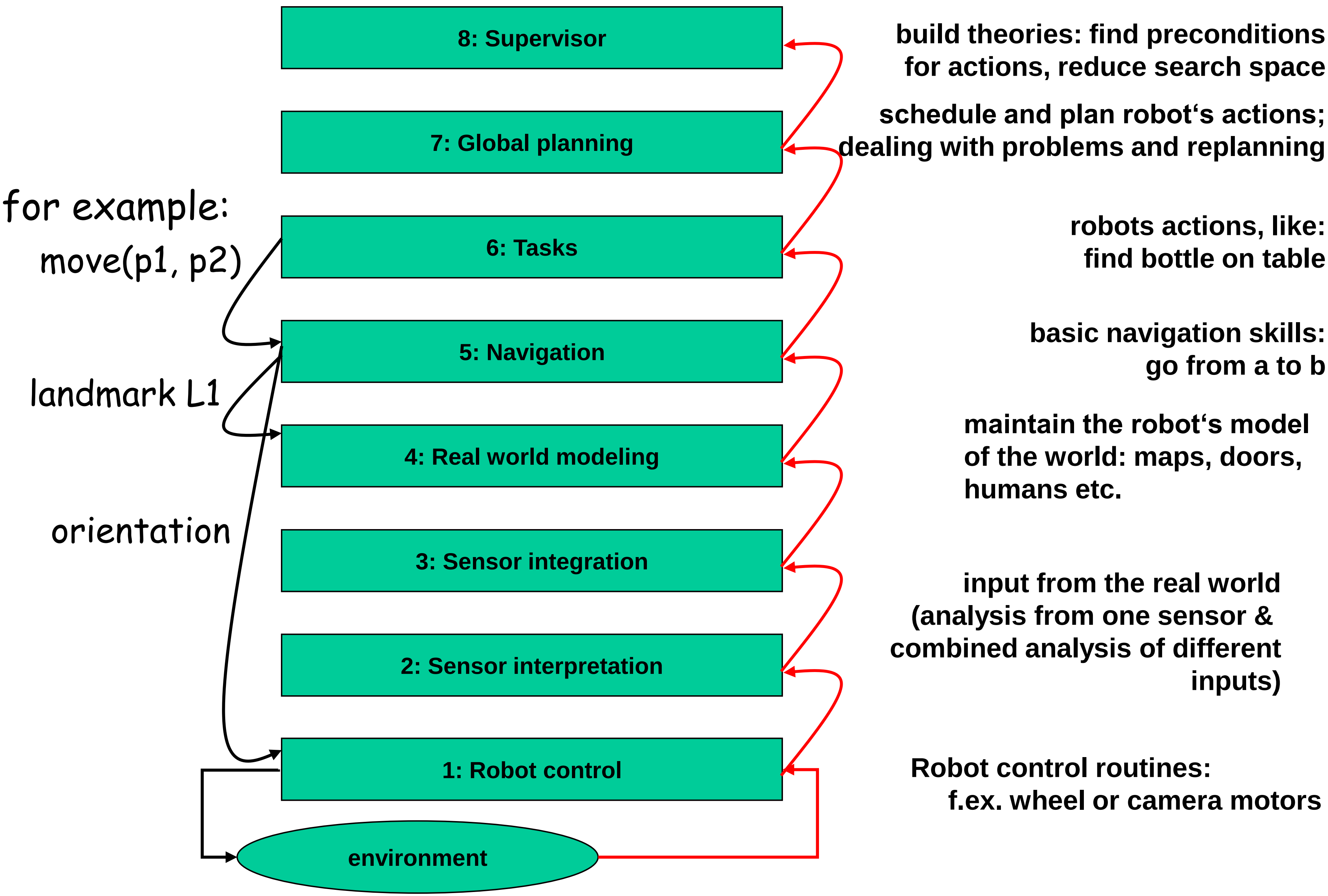
robotics example: layered architecture



robotics example: layered architecture



robotics architectures: layered architecture

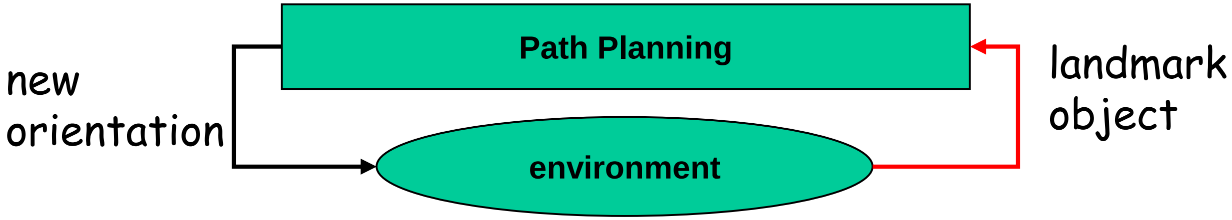


robotics example: layered architecture

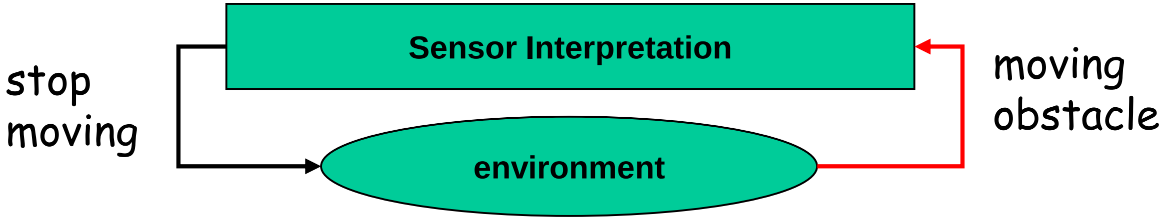
comparison between Control Loop and Layers:

– „Layers“ has more possibilities for structuring the architecture

– seen as a black box, the principle of control is similar: a fixed loop of sensing and acting



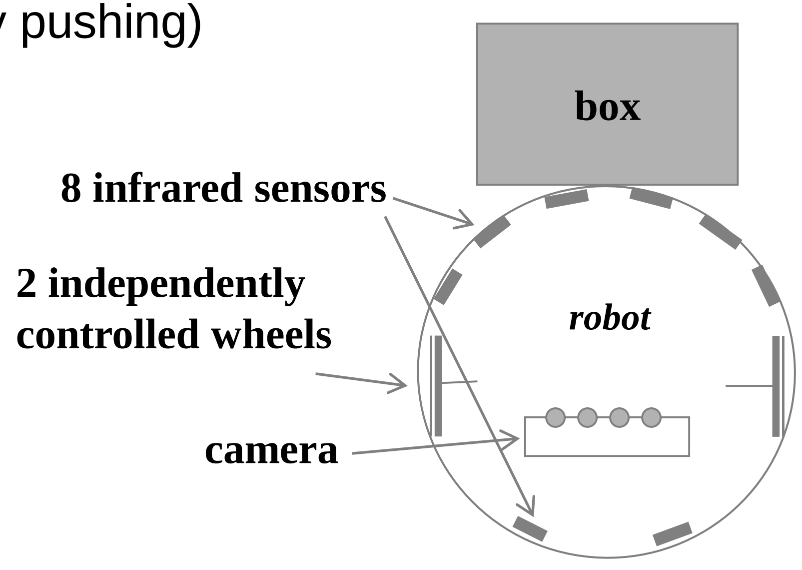
– the lower the layer, the shorter the time step, and the lower the concept of the sensory input



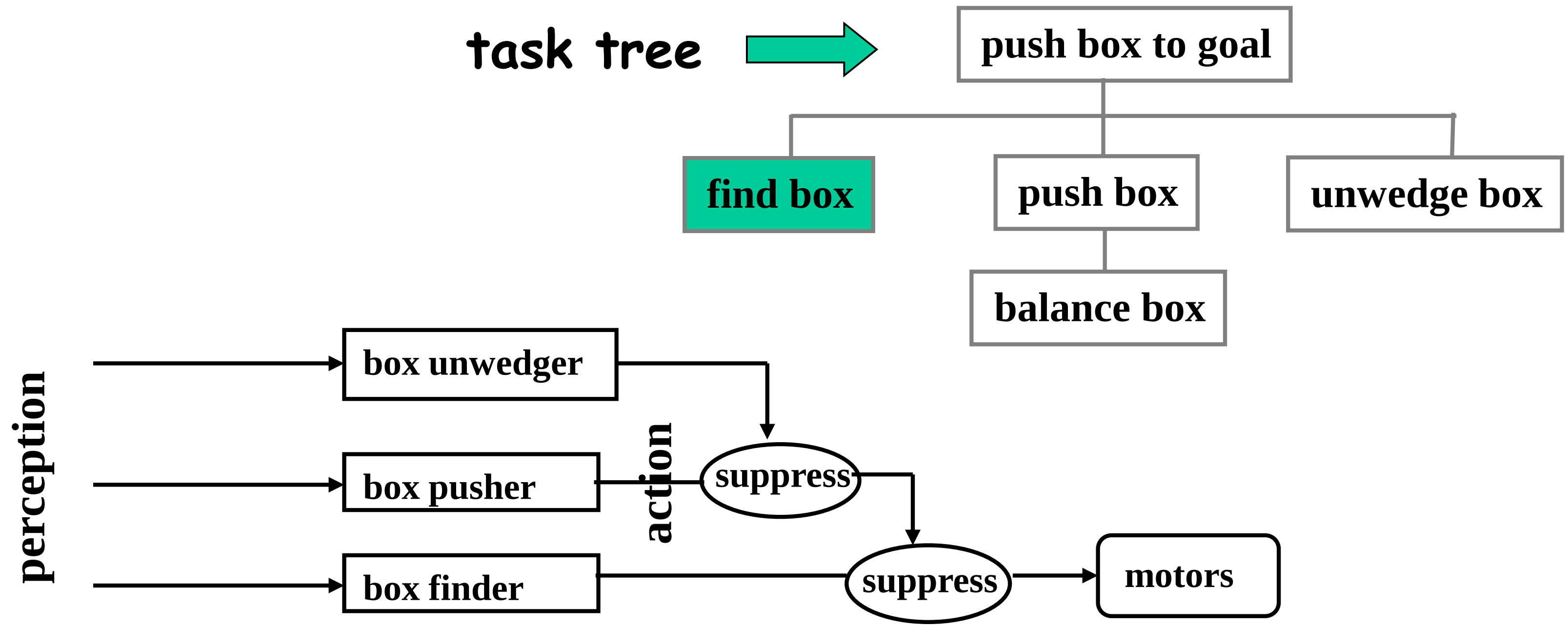
Robot architecture 3: subsumption [Brooks 1985]

example: the box-pushing task

- **perceptions**
  - vision (camera), bumper/proximity (infrared sensors)
- **actions**
  - movement: turn to the left, turn to the right, move ahead
- **goals**
  - push a box moving it away from its actual place, or
  - move a box to a goal point (by pushing)
- **environment**
  - office



Robot architecture 3: subsumption [Brooks 1985]



subsumption architecture:

- each behavior is a reflexive agent, as it is the whole system: autonomous in itself
- each behavior has an **activation condition**: activate(perception): boolean
- a behaviors action at a higher level overrides („suppress“) each action at a lower level
- essentially a hard-wired schema of priorities