ROS basic

TA

DongHyun Kim

Table of Contents

What is ROS?

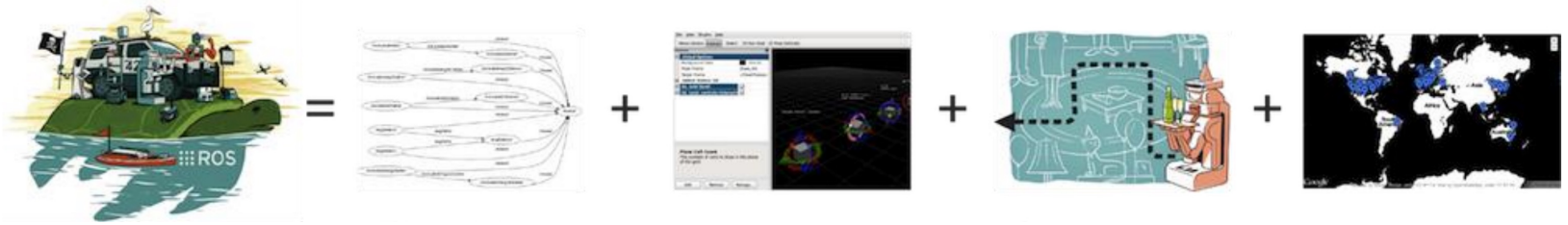
ROS(Robot operating system)

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.

https://wiki.ros.org/

What is ROS?

ROS(Robot operating system)



Plumbing	Tools	Capabilities	Ecosystem
• Process management • Inter-process communication • Device drivers	• Simulator • Visualization • Graphical user interface • Data logging	• Control • Planning • Perception • Mapping • Manipulation	• Package organization • Software distribution • Documentation • Tutorials

What is ROS?

ROS versions

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014

http://wiki.ros.org/Distributions

Distro	Release date	Logo	EOL date
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		December 9th, 2022
Foxy Fitzroy	June 5th, 2020		May 2023
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019
Bouncy Bolson	July 2nd, 2018		July 2019
Ardent Apalone	December 8th, 2017		December 2018

https://docs.ros.org/en/foxy/Releases.html

Environment setup for ROS

Development environment for ROS	
Operating system	Ubuntu 20.04
ROS version	ROS Noetic Ninjemys ROS2 Galactic Geochelone
Computer architecture	amd64
IDE	Visual studio Code
Programming language	Python 3, C++14(or latest)
Simulator	Gazebo 11.x(or Ignition)

Ubuntu basic command

Ubuntu basic command practice

Package management commands

commands		function
sudo		run command as admin
apt		advanced packaging tools
	update	refresh available updates (sudo apt update)
	upgrade	upgrade all packages (sudo apt upgrade)
	install	install package (sudo apt install ‘package_name’)
	purge	uninstall package (sudo apt purge ‘package_name’)
	autoremove	remove obsolete packages (sudo apt autoremove)
	search	search the installed packages (apt search ‘name’)

Ubuntu basic command practice

File management commands

commands	function
cd	change directory
ls	directory listing
mkdir	create a directory
touch	create or update a file
mv	move or rename a file
cp	copy a file
rm	delete a file
rmdir	delete an empty directory
pwd	print working directory
cat	show the contents of the file
echo	print the input text
tar	compress and extract the file

Ubuntu basic command practice

Searching commands

commands	function
find	search the file
find *dir* **–name** *file*	search the file in some *dir* name *file*
grep *pattern file*	search the *pattern* in file
grep –r *pattern file*	search recursively for *pattern* in *dir*
command **| grep** *pattern*	search for *pattern* in the output of *command*

Ubuntu basic command practice

Shortcuts

commands	function
Tab	complete the command automatically
ctrl + c	kill the running program in that terminal
ctrl + l	Similar like ‘clc’ in matlab
ctrl + shift + c	copy from terminal
ctrl + shift + v	paste to terminal

Ubuntu basic command practice

Terminal Hotkeys

commands	function
ctrl + alt + t	open the terminal
ctrl + shift + t	open new tab of terminal
ctrl + shift + w	close current tab of terminal
ctrl + shift + q	close all tab of terminal
ctrl + pageup/down	change the tab
ctrl + shift + o	[terminator] split the terminal horizontally
ctrl + shift + e	[terminator] split the terminal vertically

ROS core concepts

ROS core concepts

ROS terms

- **Node**
  The smallest unit of executable processors. It can be regarded as single executable program. In ROS, a system is consist of many nodes. Each node transmits and receives data by message communication.

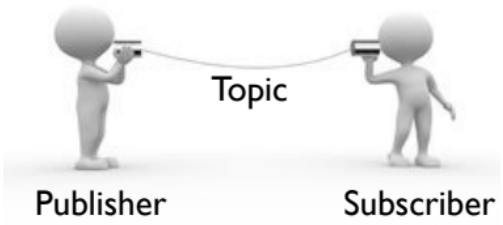
- **Package**
  One or more nodes, information for node execution, etc. Also, bundles of packages are called as metapackages.

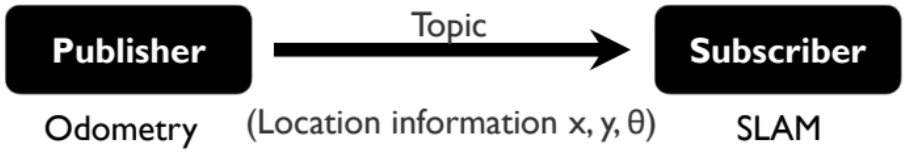
- **Message**
  Data is transmitted and received through message between nodes. Messages can have various types such as integer, floating point, and Boolean.

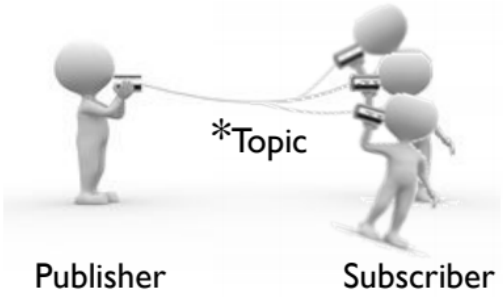
- **Topic**
  Topic is named stream of messages with a defined type. Nodes communicate with each other by publishing messages to topics.

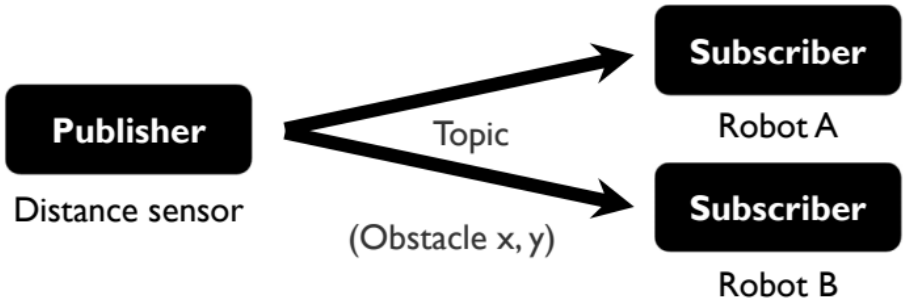
ROS core concepts

Topic, publisher, subscriber





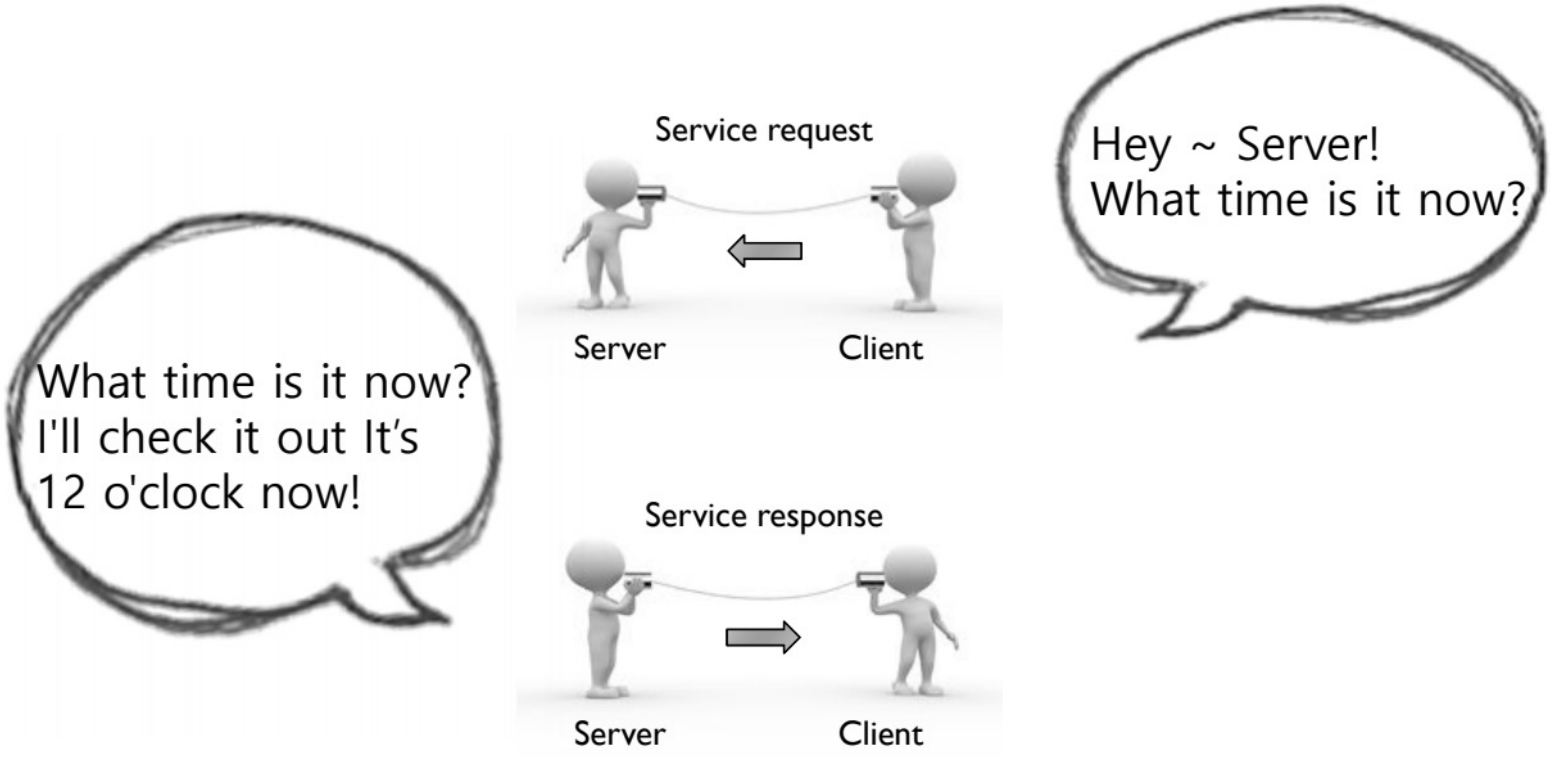




* 1: 1 Publisher and Subscriber communication is also possible for Topic, and 1: N, N: 1, N: N communication is also possible depending on the purpose.

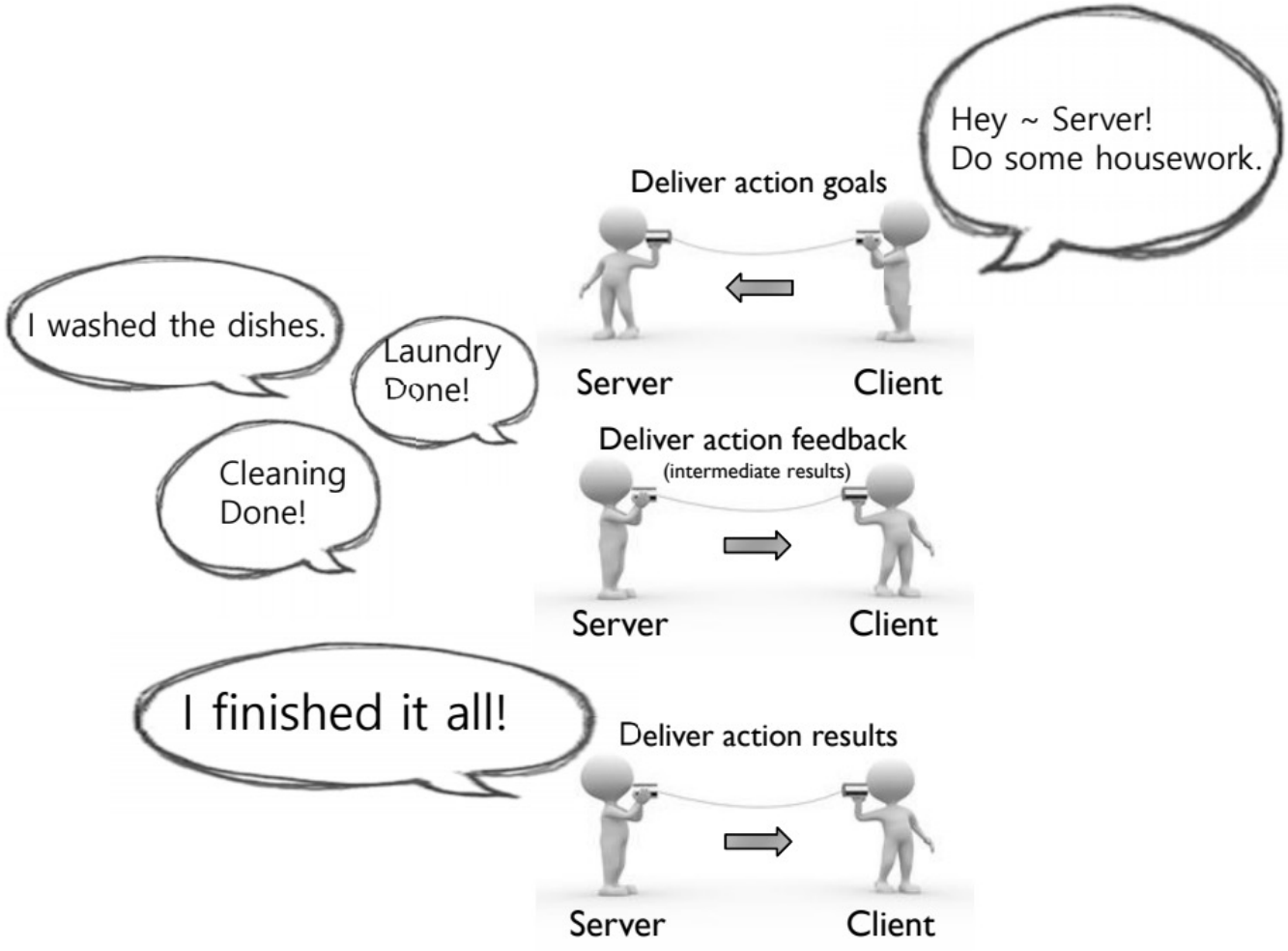
ROS core concepts

Service, Service server, Service client



ROS core concepts

Action, action server, action client



ROS core concepts with ROS 1

ROS Master

- Manages the communication between nodes (processes)
- Every node registers at startup with the master

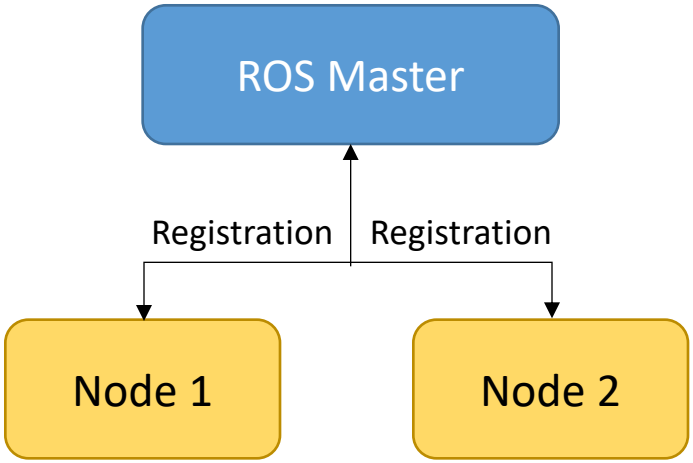
Start a master with

```
> roscore
```

ROS core concepts with ROS 1

ROS Nodes

- Single purpose, executable program
- Individually compiled, executed, and managed
- Organized in pakages



Run a node with

```
> rosrun package_name node_name
```

See active nodes with

```
> rosnode list
```

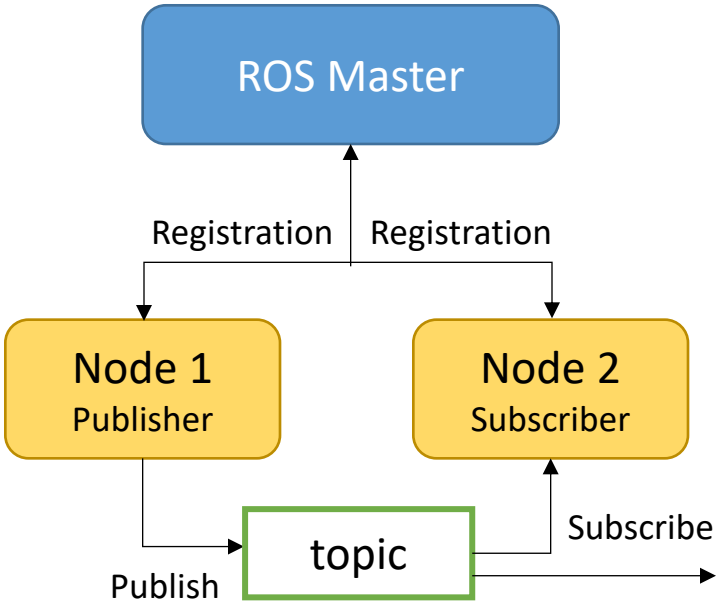
Retrieve information about a node with

```
> rosnode info node_name
```

ROS core concepts with ROS 1

ROS Topics

- Nodes communicate over topics
  - Nodes can publish or subscribe to a topic
  - Typically, 1 publisher and n subscriber
- Topic is a name for a stream of messages



List active topics with

```
> rostopic list
```

Subscribe and print the contents of a topic with

```
> rostopic echo /topic
```

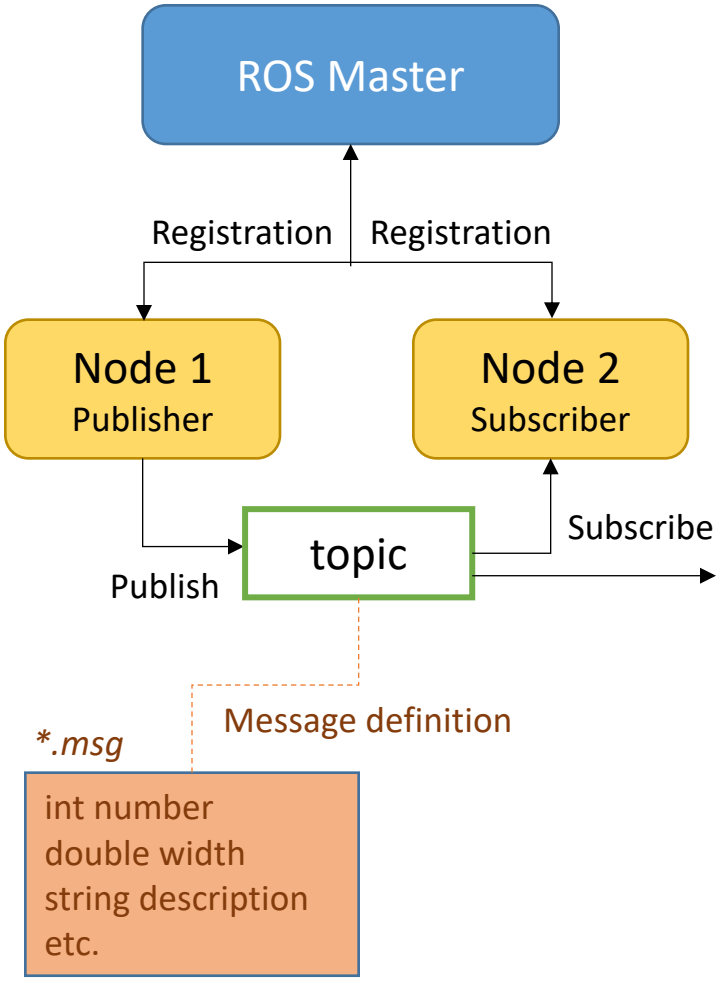
Show information about a topic with

```
> rostopic info /topic
```

ROS core concepts with ROS 1

ROS Messages

- Data structure defining the type of a topic
- Comprised of a nested structure of integers, floats, Booleans, strings, etc. and arrays of objects
- Defined in **.msg* file



See the type of a topic

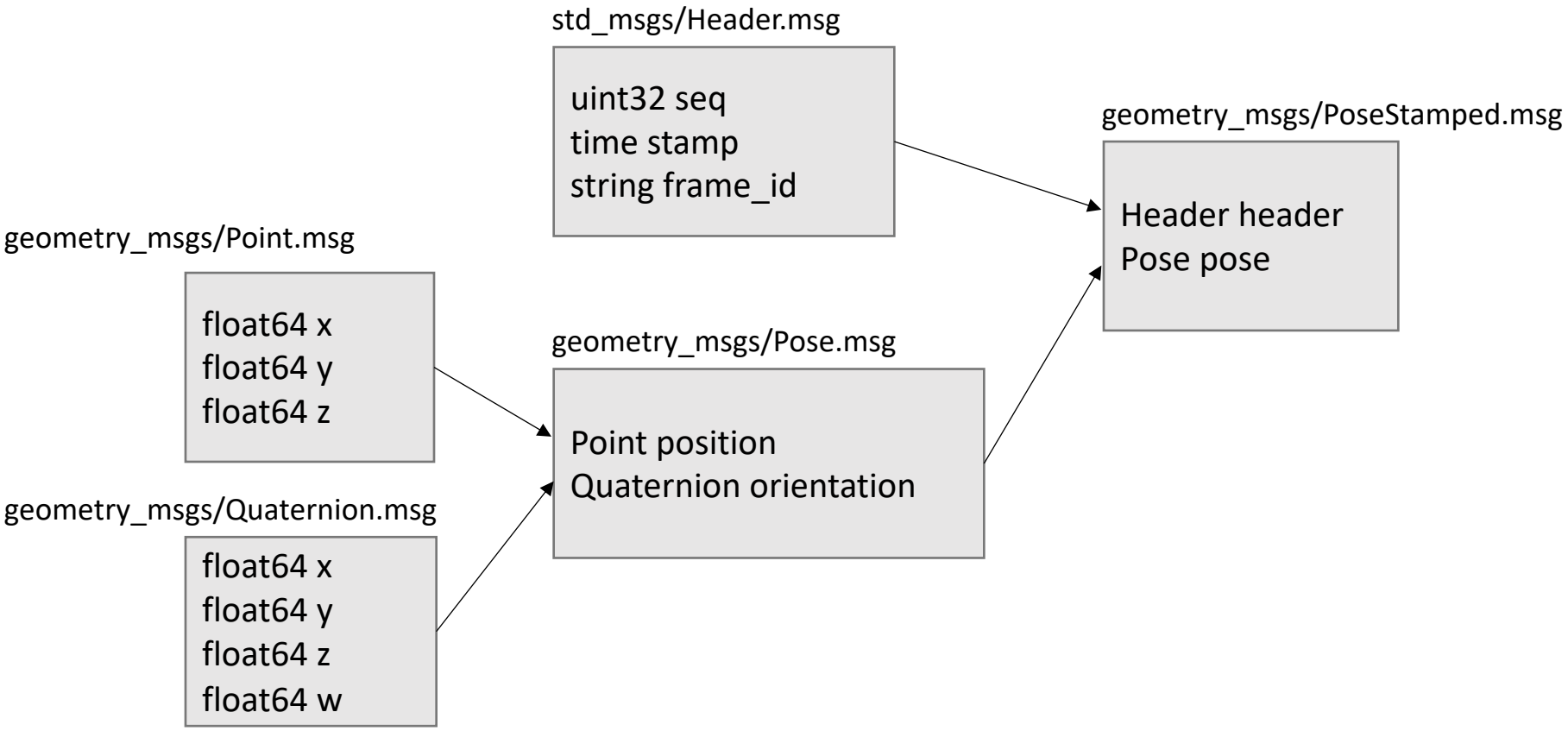
```
> rostopic type /topic
```

Publish a message to a topic

```
> rostopic pub /topic type data
```

ROS core concepts with ROS 1

ROS Messages example



ROS1 basic practice

ROS basic practice

ROS 1 & ROS 2 installation

http://wiki.ros.org/noetic/Installation/Ubuntu

1. Installation

1.1 Configure your Ubuntu repositories

Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse." You can follow the Ubuntu guide for instructions on doing this.

1.2 Setup your sources.list

Setup your computer to accept software from packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Mirrors Source Debs are also available

1.3 Set up your keys

```
sudo apt install curl # if you haven't already installed curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

1.4 Installation

First, make sure your Debian package index is up-to-date:

```
sudo apt update
```

Now pick how much of ROS you would like to install.

Desktop-Full Install: (Recommended) : Everything in **Desktop** plus 2D/3D simulators and 2D/3D perception packages

```
sudo apt install ros-noetic-desktop-full
```

or click here

Desktop Install: Everything in **ROS-Base** plus tools like rqt and rviz

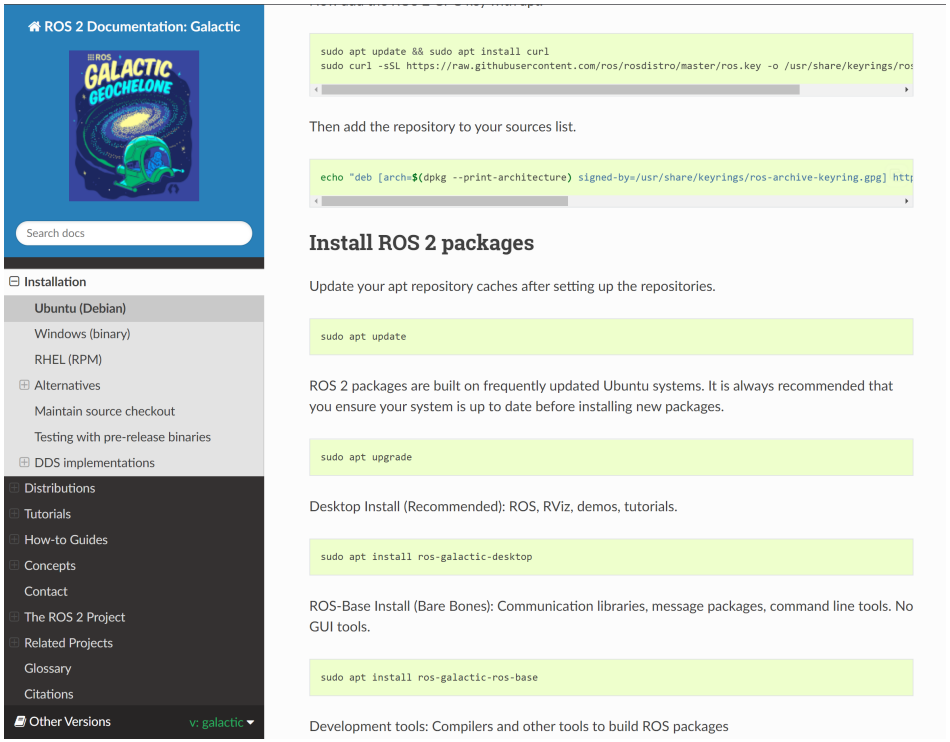
```
sudo apt install ros-noetic-desktop
```

or click here

ROS-Base: (Bare Bones) ROS packaging, build, and communication libraries. No GUI tools.

```
sudo apt install ros-noetic-ros-base
```

https://docs.ros.org/en/galactic/Installation/Ubuntu-Install-Debians.html



ROS 1 basic practice

ROS1 setup

```
> code ~/.bashrc
```

- Add the following commands

```
export ROS_MASTER_URI=http://localhost:11311
export ROS_IP=localhost
source /opt/ros/noetic/setup.bash
```

ROS 1 basic practice

ROS 1 turtlesim example

```
> roscore
```

```
> rosrun turtlesim turtlesim_node
```

```
> rosrun turtlesim turtle_teleop_key
```

```
> rqt_graph
```

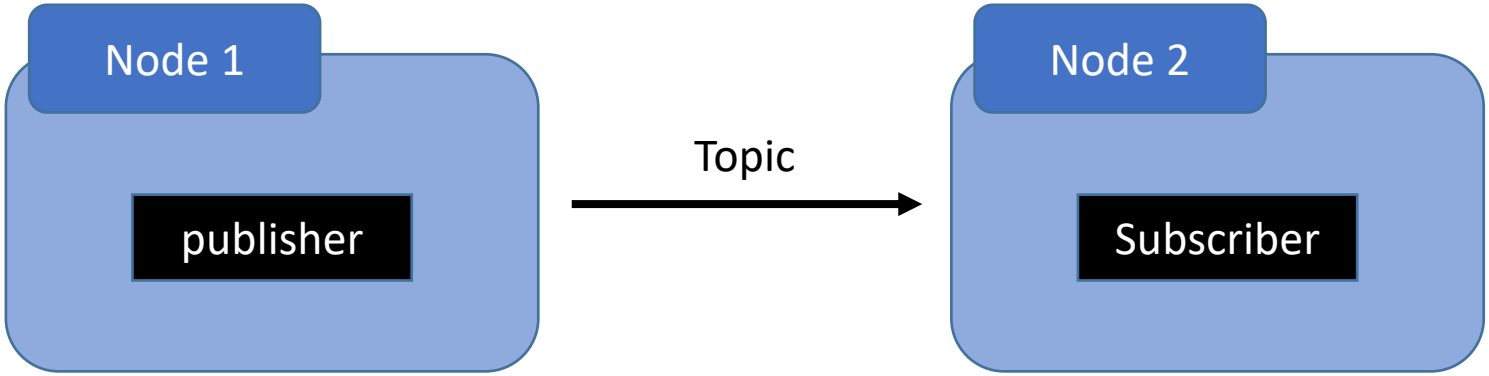
```
> rostopic list
```

```
> rostopic info /turtle1/cmd_vel
```

```
> rostopic echo /turtle1/cmd_vel
```

ROS 1 basic practice

Publisher/Subscriber node programming practice



ROS 1 basic practice

Setup catkin workspace build system

- Setup catkin_ws

```
> mkdir -p ~/catkin_ws/src
> cd ~/catkin_ws
> catkin_make
> source devel/setup.bash
```

- Setup alias for convenience

```
> code ~
```

find *.bashrc* and put the below codes

```
source ~/catkin_ws/devel/setup.bash
alias cm='cd ~/catkin_ws && catkin_make'
alias cs='cd~/catkin_ws/src'
alias eb='code ~/.bashrc'
alias sb='source ~/.bashrc'
```

ROS 1 basic practice

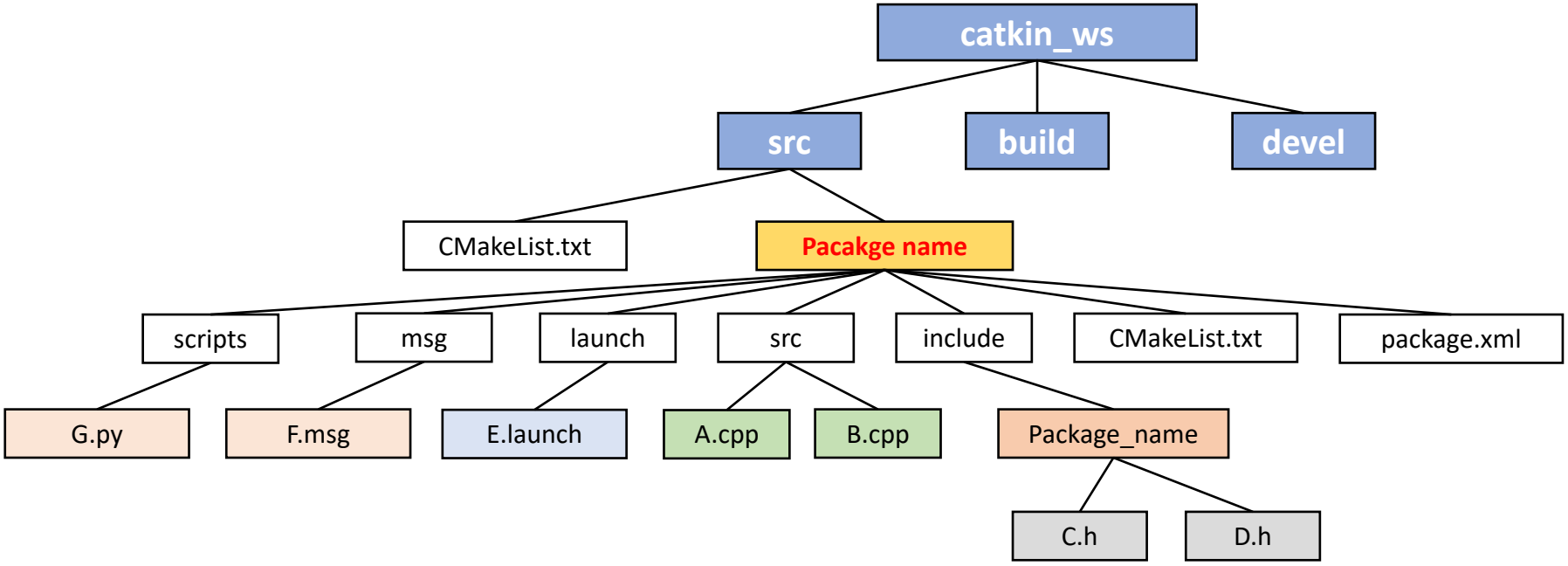
Create a new package for practice(C++)

```
> cd ~/catkin_ws/src
> catkin_create_pkg cpp_example_pkg roscpp std_msgs message_generation
```

```
> cd ros_tutorials
> ls
include → header file folder
src → source code folder
CMakeLists.txt → build configuration file
package.xml → package configuration file
```

ROS 1 basic practice

ROS package components



ROS 1 basic practice

package.xml

```xml
<?xml version="1.0"?>
<package format="2">
  <name>cpp_example_pkg</name>
  <version>0.0.0</version>
  <description>The cpp_example_pkg package</description>
  <maintainer email="rlaehdgusqwe@kaist.ac.kr">dongk</maintainer>
  <license>BSD</license>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>message_generation</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>

  <exec_depend>roscpp</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>message_runtime</exec_depend>

</package>
```

ROS 1 basic practice

CMakeLists.txt(C++)

```
cmake_minimum_required(VERSION 3.0.2)
project(cpp_example_pkg)

## Find catkin and any catkin packages
find_package(catkin REQUIRED COMPONENTS
  message_generation
  roscpp
  std_msgs
)

## Declare ROS messages
add_message_files(FILES Counts.msg)

## Generate added messages
generate_messages(DEPENDENCIES std_msgs)

## Declare a catkin package
catkin_package(
  LIBRARIES cpp_example_pkg
  CATKIN_DEPENDS std_msgs roscpp
)

## Build talker and listener
include_directories(${catkin_INCLUDE_DIRS})

add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(listener ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
```

ROS 1 basic practice

Create message file

- put the following options in the CMakeLists.txt

```
add_message_files(FILES Counts.msg)
```

- create msg file

```
> cd ~/catkin_ws/src/cpp_example_pkg
> mkdir msg
> cd msg
> code Counts.msg
```

```
1   time stamp
2   int32 count
3   int32 square
4   int32 cubic
```

ROS 1 basic practice

Creating the publisher node

- put the following options in the CMakeLists.txt

```
add_executable(talker src/talker.cpp)
```

- create cpp file

```
> cd ~/catkin_ws/src/cpp_example_pkg/src
> code talker.cpp
```

- vscode C/C++ configuration setup

```
ctrl + shift + p
> C/C++: Edit Configurations (JSON)
```

```
5            "includePath": [
6                "${workspaceFolder}/**",
7                "~/catkin_ws/devel/include",
8                "/opt/ros/noetic/include",
9                "/usr/include/**"
```

ROS 1 basic practice

Creating the publisher node

```cpp
#include "ros/ros.h"
#include "cpp_example_pkg/Counts.h"
#include<cmath>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "talker");
    ros::NodeHandle nh;

    ros::Publisher pub = nh.advertise<cpp_example_pkg::Counts>("chatter", 100);
    ros::Rate loop_rate(1);
    cpp_example_pkg::Counts msg;
    int count = 0;

    while (ros::ok())
    {
        msg.stamp = ros::Time::now();
        msg.count = count;
        msg.square = int(pow(count, 2));
        msg.cubic = int(pow(count, 3));
        ROS_INFO("Send msg");
        ROS_INFO("time : %d", msg.stamp.sec);
        ROS_INFO("count : %d", msg.count);
        ROS_INFO("count square : %d", msg.square);
        ROS_INFO("count cubic : %d", msg.cubic);
        pub.publish(msg);
        loop_rate.sleep();
        count++;
    }
    return 0;
}
```

msg include

node name

declare publisher

1Hz

declare msg

publish

loop rate 1Hz

ROS 1 basic practice

Creating the subscriber node

- put the following options in the CMakeLists.txt

```
add_executable(listener src/listener.cpp)
```

- create cpp file

```
> cd ~/catkin_ws/src/cpp_example_pkg/src
> code listener.cpp
```

ROS 1 basic practice

Creating the subscriber node

```cpp
#include "ros/ros.h"
#include "cpp_example_pkg/Counts.h"

void msgCallback(const cpp_example_pkg::Counts::ConstPtr& msg)
{
    ROS_INFO("Recieve msg");
    ROS_INFO("time : %d", msg->stamp.sec);
    ROS_INFO("count : %d", msg->count);
    ROS_INFO("count square : %d", msg->square);
    ROS_INFO("count cubic : %d", msg->cubic);
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "listener");
    ros::NodeHandle nh;
    ros::Subscriber ros_tutorial_sub = nh.subscribe("chatter", 100, msgCallback);
    ros::spin();
    return 0;
}
```

msg include

callback function

node name

Process callback functions requested in queue

declare subscriber

ROS 1 basic practice

Build the nodes

- Build the cpp_example_pkg package with the following command

```
> cd ~/catkin_ws
> catkin_make
```

- or simply use alias

```
> cm
```

```
Scanning dependencies of target listener
Scanning dependencies of target talker
[ 63%] Built target cpp_example_pkg_generate_messages
[ 72%] Building CXX object cpp_example_pkg/CMakeFiles/listener.dir/src/listener.cpp.o
[ 81%] Building CXX object cpp_example_pkg/CMakeFiles/talker.dir/src/talker.cpp.o
[ 90%] Linking CXX executable /home/dongk/catkin_ws/devel/lib/cpp_example_pkg/talker
[ 90%] Built target talker
[100%] Linking CXX executable /home/dongk/catkin_ws/devel/lib/cpp_example_pkg/listener
[100%] Built target listener
 dongk 🐥  ~/catkin_ws
```

ROS 1 basic practice

Execute the publisher node

> roscore

> rosrun cpp_example_pkg talker

```
dongk ~ rosrun cpp_example_pkg talker
[ INFO] [1672080415.081276699]: Send msg
[ INFO] [1672080415.081635298]: time : 1672080415
[ INFO] [1672080415.081639616]: count : 0
[ INFO] [1672080415.081642931]: count square : 0
[ INFO] [1672080415.081646113]: count cubic : 0
[ INFO] [1672080416.081461355]: Send msg
[ INFO] [1672080416.081537758]: time : 1672080416
[ INFO] [1672080416.081559394]: count : 1
[ INFO] [1672080416.081588653]: count square : 1
[ INFO] [1672080416.081608435]: count cubic : 1
[ INFO] [1672080417.081456471]: Send msg
[ INFO] [1672080417.081535076]: time : 1672080417
[ INFO] [1672080417.081556526]: count : 2
[ INFO] [1672080417.081575937]: count square : 4
[ INFO] [1672080417.081595968]: count cubic : 8
[ INFO] [1672080418.081454998]: Send msg
[ INFO] [1672080418.081534364]: time : 1672080418
[ INFO] [1672080418.081557110]: count : 3
[ INFO] [1672080418.081576788]: count square : 9
[ INFO] [1672080418.081596602]: count cubic : 27
```

ROS 1 basic practice

Check the status of tostopic

> rostopic list

> rostopic echo /chatter

```
 dongk ❣️  ~  rostopic list
/chatter
/rosout
/rosout_agg
 dongk ❣️  ~  rostopic echo /chatter
stamp:
  secs: 1672084388
  nsecs: 769163636
count: 32
square: 1024
cubic: 32768
---
stamp:
  secs: 1672084389
  nsecs: 769166578
count: 33
square: 1089
cubic: 35937
---
```

ROS 1 basic practice

Execute the subscriber node

> rosrun cpp_example_pkg listener

```
dongk ⚡ ~/catkin_ws rosrun cpp_example_pkg listener
[ INFO] [1672080416.082351889]: Recieve msg
[ INFO] [1672080416.086834033]: time : 1672080416
[ INFO] [1672080416.086886942]: count : 1
[ INFO] [1672080416.086908422]: count square : 1
[ INFO] [1672080416.086928011]: count cubic : 1
[ INFO] [1672080417.082112312]: Recieve msg
[ INFO] [1672080417.082180890]: time : 1672080417
[ INFO] [1672080417.082201378]: count : 2
[ INFO] [1672080417.082220624]: count square : 4
[ INFO] [1672080417.082239003]: count cubic : 8
[ INFO] [1672080418.082112686]: Recieve msg
[ INFO] [1672080418.082186550]: time : 1672080418
[ INFO] [1672080418.082207790]: count : 3
[ INFO] [1672080418.082228926]: count square : 9
[ INFO] [1672080418.082248880]: count cubic : 27
```

ROS 1 basic practice

Execute the publisher node

> roscore

> rosrun cpp_example_pkg listener

```
dongk ⚡ ~/catkin_ws rosrun cpp_example_pkg listener
[ INFO] [1672080416.082351889]: Recieve msg
[ INFO] [1672080416.086834033]: time : 1672080416
[ INFO] [1672080416.086886942]: count : 1
[ INFO] [1672080416.086908422]: count square : 1
[ INFO] [1672080416.086928011]: count cubic : 1
[ INFO] [1672080417.082112312]: Recieve msg
[ INFO] [1672080417.082180890]: time : 1672080417
[ INFO] [1672080417.082201378]: count : 2
[ INFO] [1672080417.082220624]: count square : 4
[ INFO] [1672080417.082239003]: count cubic : 8
[ INFO] [1672080418.082112686]: Recieve msg
[ INFO] [1672080418.082186550]: time : 1672080418
[ INFO] [1672080418.082207790]: count : 3
[ INFO] [1672080418.082228926]: count square : 9
[ INFO] [1672080418.082248880]: count cubic : 27
```

ROS 1 basic practice

rqt graph

```
> rqt_graph
```

ROS 1 basic practice

python version create package & CMakeLists.txt

```
> cd ~/catkin_ws/src
> catkin_create_pkg  rospy std_msgs message_generation
```

```
cmake_minimum_required(VERSION 3.0.2)
project(py_example_pkg)

## Find catkin and any catkin packages
find_package(catkin REQUIRED COMPONENTS
  message_generation
  rospy
  std_msgs
)

## Declare ROS messages
add_message_files(FILES Counts.msg)

## Generate added messages
generate_messages(DEPENDENCIES std_msgs)

catkin_package()
```

ROS 1 basic practice

python version publisher/subscriber

```python
#!/usr/bin/env python3
import rospy
from py_example_pkg.msg import Counts

def talker():
    pub = rospy.Publisher('chatter', Counts)
    rospy.init_node('talker', anonymous=True)
    r = rospy.Rate(1)
    msg = Counts()
    count = 0
    while not rospy.is_shutdown():
        msg.stamp = rospy.Time.now()
        msg.count = count
        msg.square = count**2
        msg.cubic = count**3
        rospy.loginfo("Send msg")
        rospy.loginfo("time : %d", msg.stamp.secs)
        rospy.loginfo("count : %d", msg.count)
        rospy.loginfo("count square : %d", msg.square)
        rospy.loginfo("count cubic : %d", msg.cubic)
        pub.publish(msg)
        count += 1
        r.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException: pass
```

```python
#!/usr/bin/env python3
import rospy
from py_example_pkg.msg import Counts

def callback(data):
    rospy.loginfo("Recieve msg")
    rospy.loginfo("time : %d", data.stamp.secs)
    rospy.loginfo("count : %d", data.count)
    rospy.loginfo("count square : %d", data.square)
    rospy.loginfo("count cubic : %d", data.cubic)

def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("chatter", Counts, callback)
    rospy.spin()

if __name__ == '__main__':
    listener()
```

ROS 1 basic practice

What is roslaunch?

- rosrun is a command to execute a node
- roslaunch can run one or more defined nodes
- In addition, roslaunch command allows you to specify options such as changing package parameters or node names, configuring node namespaces, setting ROS_ROOT and ROS_PACKAGE_PATH, and changing environment variables when running a node.
- roslaunch uses the file '* .launch' to set up an executable node, which is XML-based and provides tag-specific options.

ROS 1 basic practice

roslaunch

```
> roscd cpp_example_pkg
> mkdir launch
> cd launch
> code union.launch
```

```
<launch>
    <node pkg="cpp_example_pkg" type="talker" name="talker1" />
    <node pkg="cpp_example_pkg" type="listener" name="listener1" />
    <node pkg="cpp_example_pkg" type="talker" name="talker2" />
    <node pkg="cpp_example_pkg" type="listener" name="listener2" />
</launch>
```

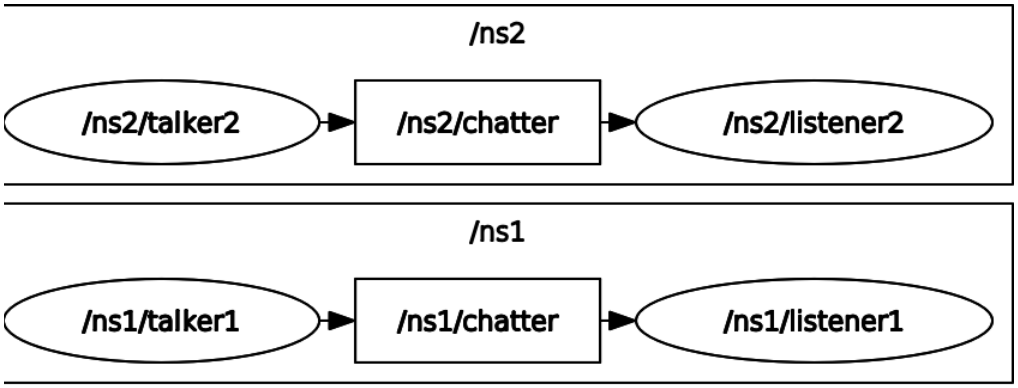
- **pkg : Package name**
- **type : The name of the node to actually excute**
- **name : Set the name to be appended**

ROS 1 basic practice

roslaunch

- Modify union.launch like below(using namespace)

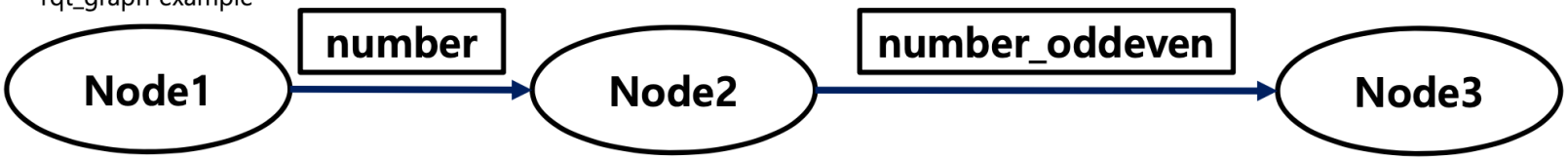
```
<launch>
    <group ns="ns1">
        <node pkg="cpp_example_pkg" type="talker" name="talker1" />
        <node pkg="cpp_example_pkg" type="listener" name="listener1" />
    </group>
    <group ns="ns1">
        <node pkg="cpp_example_pkg" type="talker" name="talker2" />
        <node pkg="cpp_example_pkg" type="listener" name="listener2" />
    </group>
</launch>
```



Exercise

Multi node communication(oddeven pkg)

* rqt_graph example



Create a code that

1. Node 1 : Publish variables that increment by 1 for each loop.
2. Node 2 : After subscribing to messages received from node 1. Calculate whether the received data is odd or even and print "odd" or "even" strings. Then, publish the output string and the received variable to node 3.
3. Node 3 : Using the data received after subscribing to the message received from node 2, print the sentence "(variable) is (string)".

Ex) “5 is an odd number.”

Exercise

Hint for Node2

```cpp
#include "ros/ros.h"                          // ROS Default Header File
#include "oddeven/number.h"   // number Message File Header. The header file is automatically created when building the package.
#include "oddeven/number_parity.h" // number_parity Message File Header. The header file is automatically created when building the package.

//Define class for publisher and subscriber in the same node
class SubandPub
{
    public:
        SubandPub()
        {
            //** TODO **//
            pub = ?? // publisher to node3
            sub = nh.subscribe(??, ??, &SubandPub::callback, this); // Subscribe from node1
        }

        void callback(const oddeven::number::ConstPtr& msg) // subscriber callback function
        {
            //** TODO **//
            //Declare message to publish
            //Check parity(odd or even)
            //publish to topic
        }

    private:
        ros::NodeHandle nh;
        ros::Publisher pub;
        ros::Subscriber sub;

};

int main(int argc, char **argv)
{
    //Initiate ROS
    ros::init(argc, argv, "parity_identifier");

    //Create an object of class SubscribeAndPublish that will take care of everything
    SubandPub SAPObject;

    ros::spin();

    return 0;
}
```

ROS 2

ROS 2

ROS 1 vs. ROS 2

개발 당시 컨셉

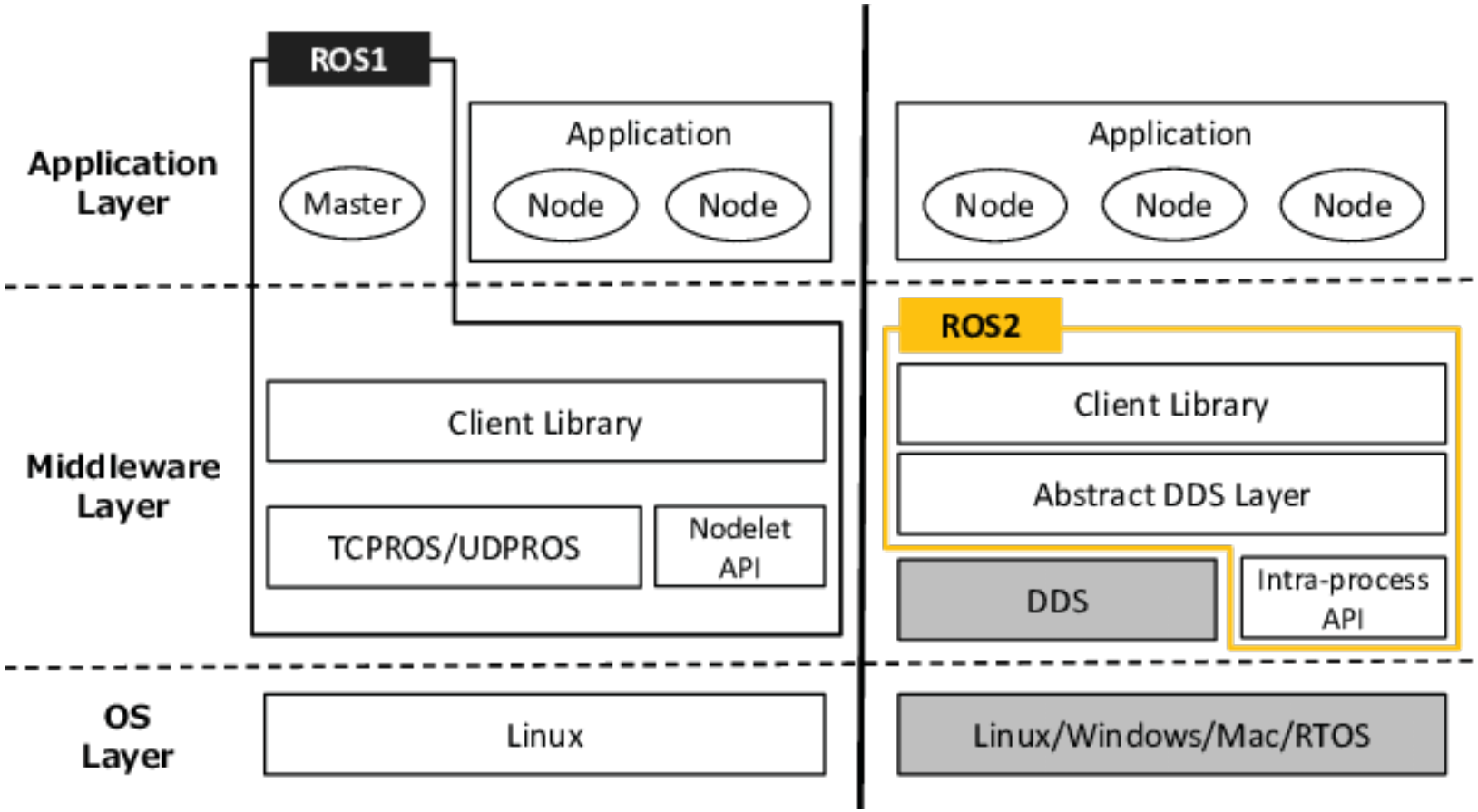
- 단일로봇
- 워크스테이션급 컴퓨터
- Linux 환경
- 실시간 제어 지원 X
- 안정된 네트워크 환경 요구
- 주로 대학이나 연구소와 같은 아카데믹 연구 용도

새롭게 요구되는 기능들

- 복수대의 로봇
- 임베디드 시스템에서의 ROS 사용
- 실시간 제어
- 불안정한 네트워크 환경에서도 동작
- 멀티 플랫폼
- 최신기술지원
- 상업용 제품 지원

ROS 2

ROS 1 vs. ROS 2



ROS 2

ROS 2 commands

commands	function
ros2 run	특정 패키지의 특정 노드 실행
ros2 launch	특정 패키지의 런치 파일 실행
ros2 pkg create	ros2 패키지 생성
ros2 node info	실행중인 노드 중 지정한 노드의 정보 출력
ros2 node list	실행중인 모든 노드이 목록 출력
ros2 topic list	사용 가능한 토픽 목록 출력
ros2 topic echo	지정 토픽의 데이터 출력
ros2 topic type	지정 토픽의 토픽 타입 출력

ROS 2 basic practice

ROS 2 basic practice

ROS2 documentation

- **Simple publisher and subscriber(C++)**

  https://docs.ros.org/en/galactic/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html

- **Simple publisher and subscriber(python)**

  https://docs.ros.org/en/galactic/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html

Thank you!