ROS DEVELOPMENT

ROBOTICS

POLITECNICO
MILANO 1863

EVERYTHING HAPPENS IN NODES

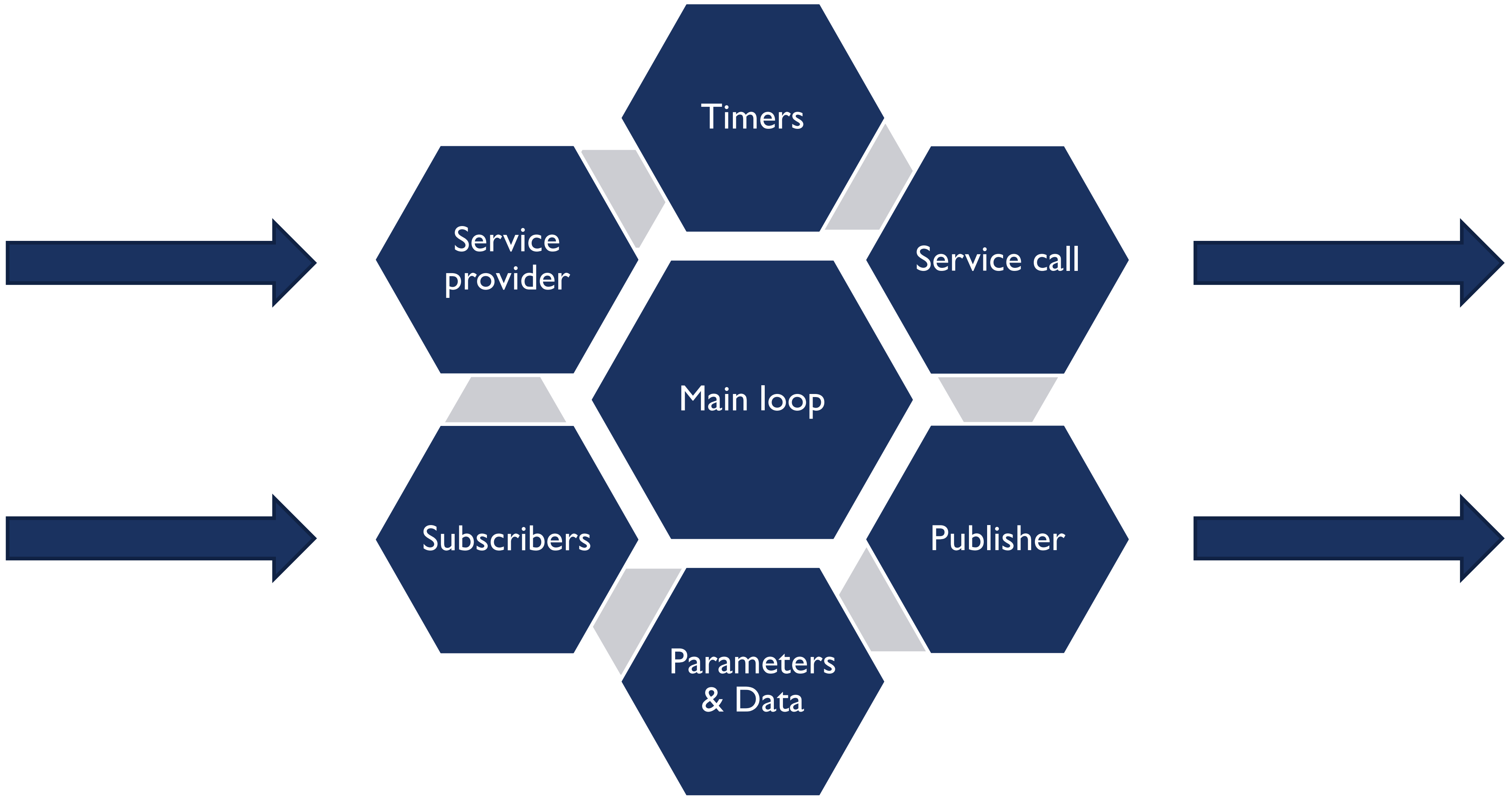
Nodes are the main and atomic element of ROS. Each node is an indipendent process.

How do we create a node?

Write code in C++ or Python

INSIDE THE NODE

goo.gl/DBwhhC



INITIALIZATION

goo.gl/DBwhhC

Any node has to be registered to the ROS master using an unique identifier

The actual node is initialized using an handler

Each executable has an **unique name**

Each executable may have multiple handlers

```
void ros::init(argv, argc, std::string node_name, uint32_t options);
ros::init(argc, argv, "my_node_name");
ros::init(argc, argv, "my_node_name", ros::init_options::AnonymousName);

ros::NodeHandle nh;
```

MAIN LOOP

Each ROS node loops waiting for something to do

At each loop checks:

- is there a message waiting to be received?
- is there a completed timer?
- is there a parameter to be reconfigured?

Two ways to implement the main loop:

- Automatically, no developer intervention
- Manual, specific sleep time and execution at each loop

```
ros::spin();


ros::Rate r(10); //10 hz
while (ros::ok()) {
  /* some execution */
  ros::spinOnce();
  r.sleep();
}
```

PARAMETERS

Stored in the parameter server and retrieved at the beginning of the execution

Adjustable at runtime using dynamic reconfigure

Global parameters and relative parameters (in the node namespace)

```
if(!nh.getParam("/global_name", global_name)) { /* :( */ }
if(!nh.getParam("relative_name", relative_name)) { /* :( */ }
nh.param<std::string>("param_name", default_param, "default_value");
```

PUBLISHER

goo.gl/DBwhhC

Used to publish messages on a ROS topic

On declaration connect the publisher to a topic and define the type of the message

Can be called from everywhere

The frequency of the messages are not set

```
ros::Publisher pub = nh.advertise<std_msgs::String>("topic_name", 5);
std_msgs::String str;
str.data = "hello world";
pub.publish(str);
```

SUBSCRIBER

goo.gl/DBwhhC

Used to read messages from a ROS topic

On declaration connect the subscriber to a topic and define the type of the message

Call a specific function when receive a message

Operate at a given frequency

```
ros::Subscriber sub = nh.subscribe(“topic_name”, 10, callback);
sub = nh.subscribe(“topic_name”, 10, &class::callback, this);
void [class::]callback(const pack_name::msg_type::ConstPtr& msg)
```

TIMER

Used to execute something after a specific time (repeatable)

When the timer ends a callback function get called

Tied to ROS internal clock

```
ros::Timer timer = nh.createTimer(ros::Duration(0.5), callback);
timer = nh.createTimer(ros::Duration(0.5), &class::callback, this);
void [class::]callback(const ros::TimerEvent& t)
```

SERVICE PROVIDER (SERVER)

Answer to a service call and execute some logic associated with the content of the call

On declaration connect to the callback with the implemented logic

The answer of the service is already in the callback

```
ros::ServiceServer s = nh.advertiseService(“service”, callback);
s = nh.advertiseService(“service”, &class::callback, this);
bool [class::]callback(pack::srv_type::Request& req,
                       pack::srv_type::Response& res);
```

SERVICE PROVIDER (SERVER)

goo.gl/DBwhhC

Generates the call for a specific service

On declaration is connected to the a service identified by a name

Can be called everywhere in the code

May result in a bad call

```
ros::ServiceClient cl = nh.serviceClient<pack::srv_type>(“service”);
pack::srv_type srv;
/* fill the service */
if (cl.call(srv)) { /* :) */ } else { /* :( */ }
```

CREATING THE WORKSPACE

goo.gl/DBwhhC

ROS uses a custom compiling environment called **Catkin**

cmake/make with specific flags

Requires a workspace with a specific structure

Easy to setup and easy to use

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

WORKSPACE STRUCTURE

Source space (`/src`):

- contains the source code of catkin packages.
- Subfolder of this are the ROS packages you want to add to your system

All your stuff goes here!

Build space (`/build`):

- space where cmake is invoked to build the catkin packages
- cmake and catkin keep their cache information and other intermediate files here

Devel space (`/devel`):

- Space where built targets are placed prior to being installed

BUILDING YOUR CODE

```
cmake_minimum_required(VERSION 2.8.3)
project(package_name)
find_package(catkin REQUIRED COMPONENTS roscpp std_msgs genmsg)
add_message_files(FILES custom_message.msg)
add_service_files(FILES custom_service.srv)
generate_messages(DEPENDENCIES std_msgs)
catkin_package()

include_directories(include ${catkin_INCLUDE_DIRS})
add_executable(executable_name src/source_code.cpp)
target_link_libraries(executable_name ${catkin_LIBRARIES})
add_dependencies(executable_name package_name_generate_messages_cpp)
```

BUILDING YOUR CODE

goo.gl/DBwhhC

```
cmake_minimum_required(VERSION 2.8.3)
project(package_name)
find_package(catkin REQUIRED COMPONENTS roscpp std_msgs genmsg)
add_message_files(FILES custom_message.msg)
add_service_files(FILES custom_service.srv)
generate_messages(DEPENDENCIES std_msgs)
catkin_package()

include_directories(include ${catkin_INCLUDE_DIRS})
add_executable(executable_name src/source_code.cpp)
target_link_libraries(executable_name ${catkin_LIBRARIES})
add_dependencies(executable_name package_name_generate_messages_cpp)
```

This is what you have to change depending on your code!

BUILDING YOUR CODE

goo.gl/DBwhhC

```
cmake_minimum_required(VERSION 2.8.3)
project(package_name)
find_package(catkin REQUIRED COMPONENTS roscpp std_msgs genmsg)
add_message_files(FILES custom_message.msg)
add_service_files(FILES custom_service.srv)
generate_messages(DEPENDENCIES std_msgs)
catkin_package()

include_directories(include ${catkin_INCLUDE_DIRS})
add_executable(executable_name src/source_code.cpp)
target_link_libraries(executable_name ${catkin_LIBRARIES})
add_dependencies(executable_name package_name_generate_messages_cpp)
```

Only if you have custom messages!

MINIMAL CMAKELISTS.TXT

goo.gl/DBwhhC

```
cmake_minimum_required(VERSION 2.8.3)
project(my_package)
find_package(catkin REQUIRED COMPONENTS roscpp std_msgs)
catkin_package()

include_directories(include ${catkin_INCLUDE_DIRS})
add_executable(my_node src/my_node.cpp)
target_link_libraries(my_node ${catkin_LIBRARIES})
```

LET'S PUT EVERYTHING TOGETHER

goo.gl/DBwhhC

Talker and listener

ROS tutorial on publish subscriber

Client and server

ROS tutorial on client and server