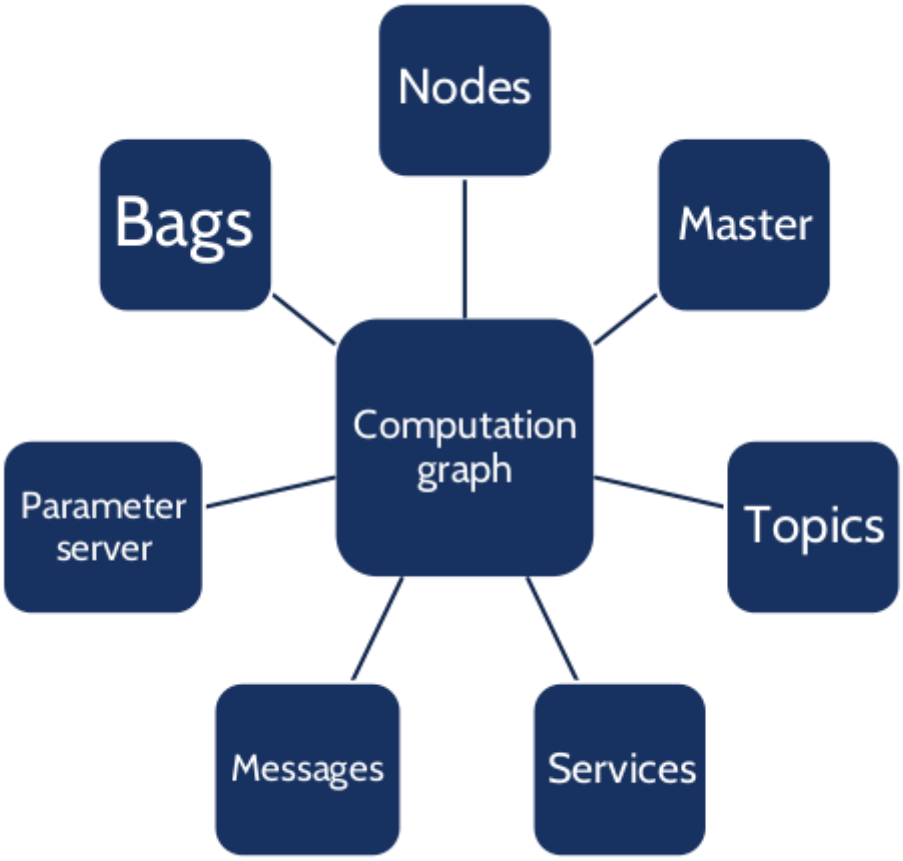
ROS

ROS



Commands

- `rosrun package_name node_name` runs a node in a package
- `roscd [package_name[/subdir]]` changes directory in ROS FileSystem
- `rospack <subcommand> [options] [package]` get information about installed packages
- `roscore` starts the ROS middleware
- `rosnode <command> [other_commands]` get information about running nodes
- `rostopic <command> [topic_name]` get info about ros topics
- `rosservice <command> [other_commands]` Calling services from command line and getting information
- `rqt_graph`
- `rosmsg <command> [msg_file]` information about msg files
- `rosbag` tool for recding and plaing back ros topics

- `rosed` -> per la bash??
-

Steps for workspace creation

1. create a dir
2. run `catkin_make`
3. link the devel/setup.bash in .bashrc
4. cd into src
5. `catkin_create_pkg [package_name] [dependency_1] [...] [dependency_n]`
6. if necessary (eg custom messages are present modofy the cmakelist.txt)
7. To build the new package `catkin_make`

Package coding

Properties

1. node must be registered to the ROS master using a unique identifier
```
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);
```
2. node is initialized using a handler
```
ros::NodeHandle nodeHandle;
```
3. executable may have multiple handlers
4. executable has a unique name
5. Resoureces are defined in a namespace and resources can acces other resources(like cpp)
6. Possibility of remapping a node (at lauch) eg:
```
rosrun turtlesim turtle_teleop_key /turtle1/cmd_vel:=/turtle2/cmd_vel
```
or redefine special keywords(always at node lauch)

Cpp keywords

- `ros::spin()` a node loops while waiting for something to do
- `ros::Rate r(freq in Hz)` node cicle at a fixed frequence
- `ros::ok()` returns a boolean of the status of the ros envirorment
- `ros::spinOnce()`
- `r.sleep()` makes the node fall asleep to ensure the fixed frequnce is respected

Publisher example

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

Subscriber Example

```
ros::Subscriber sub = n.subscribe("/publisher", 1000, pubCallback);
```

```
void pubCallback(const std_msgs::String::ConstPtr& msg) {
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}
```

Launch file

Lauch a ros project whit only one command:

- start roscore
- start nodes
- set parameters
  To do so

1. mkdir launch
2. file.launch (xml file with root `<launch></launch>`)
3. write a node lauch

Name of the executable

```
<node pkg="package_name" type="node_type" name="node_name"/>
```

Runtime name,
specified in `ros::init`

NB : by default screen output is disable for nodes lauched from launch file

4. run it with rosrun

NameSpace Utilization Example in LaunchFile

Namespaces allow to use same name for different nodes

```
<group ns="turtlesim1">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
</group>

<group ns="turtlesim2">
   <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
</group>
```

Remapping Example in LaunchFile

```
<node ...>
    <remap from="original_name" to="new_name"/>
</node>
```