Robot Operating System

- ROS is robotics middleware / framework that provides hardware abstraction, low-level device control, communication between processes and package management

  ↳ abstract away particularities of a robot and make "intelligence" useable in different kinds of robots

Structure of a ROS workspace

- ros_ws
  - build
  - devel
  - src
    - package1 → each package implements specific functionality (self-contained)
    - package2
    - CMakeLists.txt

```
$ cd ros-ws/src
$ catkin_create_pkg <pkg-name> <dependency1> <dependency2> ...
```

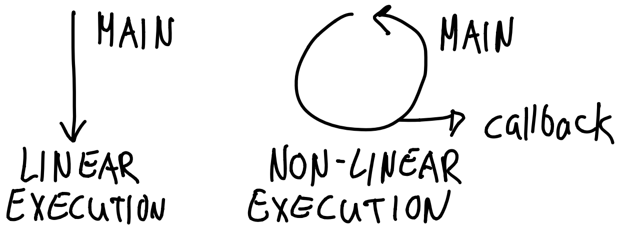
↳ creates new package inside src folder with specified dependencies (e.g. roscpp rospy)

```
$ cd ~/ros_ws
$ catkin_make
```

→ builds (= compile + link) all the files

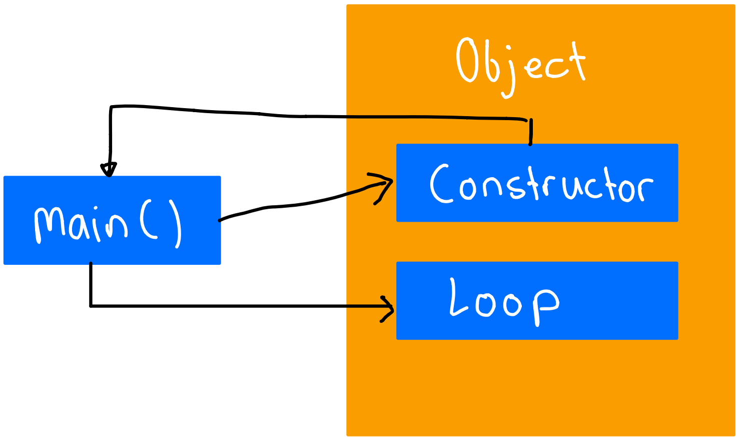
ROS Nodes

- executables linked against ROS libraries
- can be in Python, C++ and more (client libraries)
- can do anything what regular programs can
- loop/spinning waits for messages to arrive (callback-based operation)



- Steps to create new node:
  1. Create .cpp / .py file
  (1a. Make .py file executable [chmod +x])
  2. Add it to CMakeList

Structure of node



- encapsulate functionality in a class
- use constructor to create all necessary topics
- use run() function to start the node's loop

ROS Topics and Messages

- a ROS topic as a communication line between different publishers and subcribers (nodes)

- every topic has one pre-defined message type

- ROS messages are the main containers of inter-process data

IMPORTANT ROS COMMANDS

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

↳ creates a new ROS workspace

```
$ roscore
```
→ starts ROS master (manages TCP sockets)

```
$ rosrun <package_name> <node>
```
→ runs node

```
$ rosrun <package_name> <node> <map_from_topic>:=<map_to_topic>
```
→ runs node and maps topic name as specified

```
$ rostopic list
```
→ list all available topics

```
$ rostopic echo <topic_name>
```
→ prints output of topic

```
$ rospack find <pkg_name>
```
→ finds exact path of package

LAUNCH FILES

- specify set of nodes that run at the same time
  - ↳ launch folder inside package

```
$ roslaunch <package_name> <launch_file>
```

↳ runs roscore / starts ROS master

↳ runs all nodes specified in launch file

- launch files can also remap topic names and specify parameters which are stored in the parameter server (private vs. public)

`$ rosparam list` → get list of parameters in server

`$ rosparam get <param_name>` → read parameter value

`$ rosparam set <param_name> <value>` → set parameter valu

- YAML serialization can be used for more complex data structures such as arrays/lists or dictionaries