Launch

- Um arquivo *.launch* possibilita executar vários nós através de apenas um comando.
- Além dos nós desejados, ele também executa o *master.*
- Portanto, ao utilizar um launch, torna-se desnecessário executar o *roscore*.

Launch

- Abra um terminal:
  - *$ cd ~/catkin_ws/src/curso_de_ros*
  - *$ mkdir launch*
  - *$ cd launch*
  - *$ gedit comunicacao.launch*
  - Coloque o seguinte código:

```
<launch>
    <node pkg="curso_de_ros" name="emissor_node" type="emissor"/>
    <node pkg="curso_de_ros" name="receptor_node" type="receptor"/>
</launch>
```

Messages

Para representar os valores dos dados transferidos, são usados as mensagens.

Conforme o tipo do dado, a mensagem acompanha o seu tipo, podendo ser:

- Integer, boolean, array, etc.

Pode ser também um conjunto de tipos primitivos (struct):

- geometry_msgs/Pose2D
- http://docs.ros.org/api/geometry_msgs/html/msg/Pose2D.html

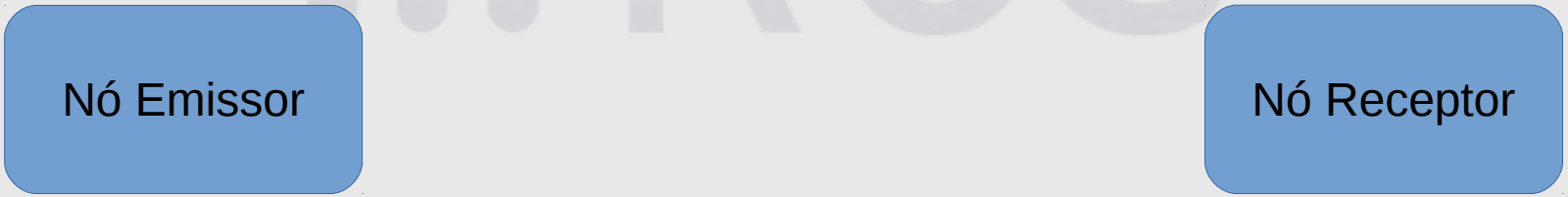
Pode-se também criar sua própria mensagem (será abordado mais a frente).

Topics

- Os nós trocam informações através dos tópicos, que podem ser considerados como um barramento de dados.
- Os nós que colocam dados em um certo tópico são chamados de publisher.
- Enquanto os nós que fazem a leitura dos dados presentes no tópico, são os subscriber.
- Em um mesmo nó podemos ter tanto publishers quanto subscribers.
- OBS: o tipo do tópico deverá ser igual ao tipo da mensagem.

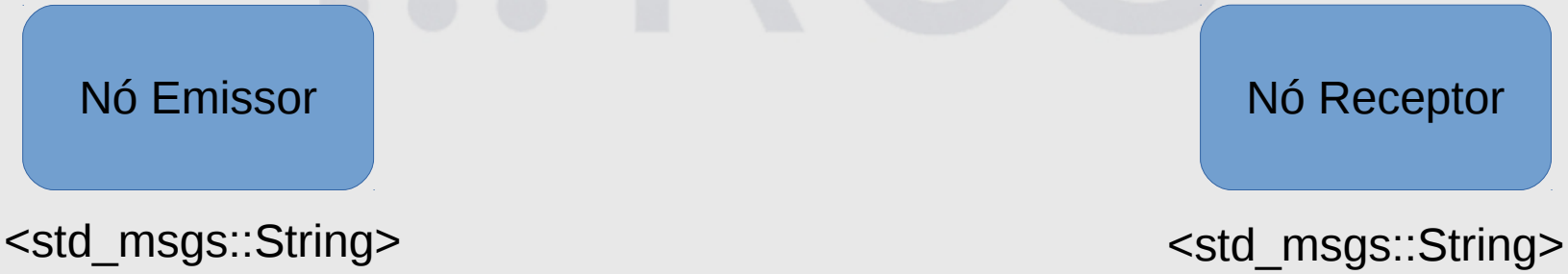
Topics

- Exemplo emissor-receptor



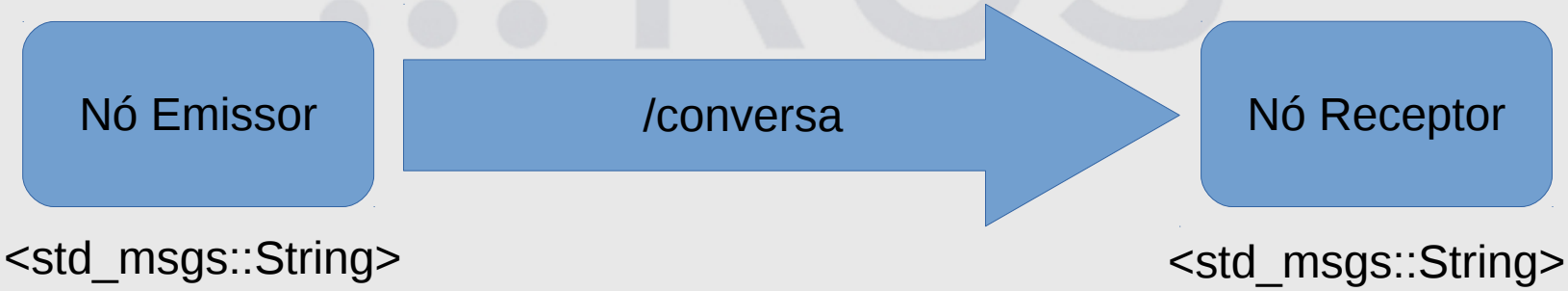
Topics

- Exemplo emissor-receptor



Topics

- Exemplo emissor-receptor



Topics

- O arquivo *emissor.cpp* apenas cria um nó e não se comunica com o nó receptor. Portanto, primeiramente iremos criar um *publisher*:

Topics (publisher)

emissor.cpp

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

int main(int argc, char **argv)
{
        ros::init(argc, argv, "emissor");

        ros::NodeHandle n;

        ros::Rate loop_rate(10);
        while (ros::ok())
        {
                ros::spinOnce();

                loop_rate.sleep();
        }
return 0;
}
```

Topics (publisher)

```
#include "ros/ros.h"

#include "std_msgs/String.h"
#include <sstream>
```

Topics (publisher)

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

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

	ros::Publisher conversa_pub = n.advertise<std_msgs::String>("conversa", 1000);

	ros::Rate loop_rate(10);
	while (ros::ok())
	{
		ros::spinOnce();

		loop_rate.sleep();
	}
return 0;
}
```

Topics (publisher)

```
int count = 0;
while (ros::ok())
{
	std_msgs::String msg;

	ros::spinOnce();

	loop_rate.sleep();
}
```

Topics (publisher)

```
int count = 0;
while (ros::ok())
{
	std_msgs::String msg;

	std::stringstream ss;
	ss << “Hello World” << count;

	ros::spinOnce();

	loop_rate.sleep();
}
```

Topics (publisher)

```
int count = 0;
while (ros::ok())
{
	std_msgs::String msg;

	std::stringstream ss;
	ss << “Hello World” << count;

	msg.data = ss.str();

	ROS_INFO(“%s”, msg.data.c_str());

	ros::spinOnce();

	loop_rate.sleep();
	++count;
}
```

Topics (publisher)

```
int count = 0;
while (ros::ok())
{
	std_msgs::String msg;

	std::stringstream ss;
	ss << “Hello World ” << count;

	msg.data = ss.str();

	ROS_INFO(“%s”, msg.data.c_str());

	conversa_pub.publish(msg);

	ros::spinOnce();

	loop_rate.sleep();
	++count;
}
```

Topics (publisher)

emissor.cpp ✖ receptor.cpp ✖

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

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

        ros::Publisher conversa_pub = n.advertise<std_msgs::String>("conversa", 1000);

        ros::Rate loop_rate(10);

        int count = 0;
        while (ros::ok())
        {
                std_msgs::String msg;

                std::stringstream ss;
                ss<< "Hello World " << count;
                msg.data = ss.str();

                ROS_INFO("%s", msg.data.c_str());

                conversa_pub.publish(msg);

                ros::spinOnce();
                loop_rate.sleep();
                ++count;
        }
return 0;
}
```

Topics

- Agora, através do arquivo *receptor.cpp* iremos criar um *subscriber*:

Topics (subscriber)

*receptor.cpp

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

int main(int argc, char **argv)
{
        ros::init(argc, argv, "receptor");

        ros::NodeHandle n;

        ros::spin();

return 0;
}
```

Topics (subscriber)

```
#include "ros/ros.h"
#include "std_msgs/String.h"

int main(int argc, char **argv)
{
	ros::init(argc, argv, "receptor");

	ros::NodeHandle n;

	ros::spin();

return 0;
}
```

Topics (subscriber)

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void conversaCallback(const std_msgs::String::ConstPtr& msg)
{
	ROS_INFO("Mensagem recebida: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
	ros::init(argc, argv, "receptor");

	ros::NodeHandle n;

	ros::spin();

return 0;
}
```

Topics (subscriber)

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void conversaCallback(const std_msgs::String::ConstPtr& msg)
{
	ROS_INFO("Mensagem recebida: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
	ros::init(argc, argv, "receptor");

	ros::NodeHandle n;

	ros::Subscriber conversa_sub = n.subscribe("conversa", 1000, conversaCallback);

	ros::spin();

return 0;
}
```

Topics (subscriber)

emissor.cpp ✖ receptor.cpp ✖

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"

void conversaCallback(const std_msgs::String::ConstPtr& msg)
{
        ROS_INFO("Mensagem recebida: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
        ros::init(argc, argv, "receptor");

        ros::NodeHandle n;

        ros::Subscriber conversa_sub = n.subscribe("conversa", 1000, conversaCallback);

        ros::spin();

return 0;
}
```

Topics

- Compile utilizando o *catkin_make* no diretório */catkin_ws*
- Execute os comando em seus respectivos terminais:
  - $ roscore
  - $ rosrun curso_de_ros emissor
  - $ rosrun curso_de_ros receptor

Topics

- Compile utilizando o *catkin_make* no diretório */catkin_ws*
- Execute os comando em seus respectivos terminais:
  - $ roscore
  - $ rosrun curso_de_ros emissor
  - $ rosrun curso_de_ros receptor
  - $ rosnode list

Topics

- Compile utilizando o *catkin_make* no diretório */catkin_ws*
- Execute os comando em seus respectivos terminais:
  - $ roscore
  - $ rosrun curso_de_ros emissor
  - $ rosrun curso_de_ros receptor
  - $ rosnode list
  - $ rostopic list

Topics

- Compile utilizando o *catkin_make* no diretório */catkin_ws*
- Execute os comando em seus respectivos terminais:
  - $ roscore
  - $ rosrun curso_de_ros emissor
  - $ rosrun curso_de_ros receptor
  - $ rosnode list
  - $ rostopic list
  - $ rostopic info /conversa

Topics

- Compile utilizando o *catkin_make* no diretório */catkin_ws*
- Execute os comando em seus respectivos terminais:
  - $ roscore
  - $ rosrun curso_de_ros emissor
  - $ rosrun curso_de_ros receptor
  - $ rosnode list
  - $ rostopic list
  - $ rostopic info /conversa
  - $ rostopic echo /conversa

Topics

- Robótica