```
interval (std::chrono::milliseconds (10))
| map ([&](int i)
    { return mk_msg(hello + std::to_string(i)); })
| tap ([](const std_msgs::String& msg)
    { ROS_INFO_STREAM (msg.data); })
| publish_to_topic<std_msgs::String>
    ("/chatter", 1000);
```

Henrik Larsen
IT University of Copenhagen

Andrzej Wąsowski
IT University of Copenhagen
@AndrzejWasowski

[Reactive] Programming with [Rx]ROS

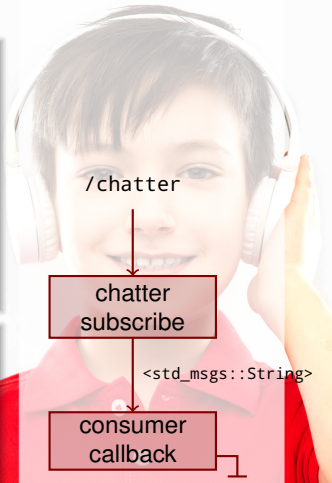
IT UNIVERSITY OF COPENHAGEN
SOFTWARE QUALITY RESEARCH
European Commission
Horizon 2020
European Union funding for Research & Innovation
ROSin

The Listener Example

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

int main(int argc, char **argv) {
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
  ros::spin();
  return 0;
}
```

```
int main(int argc, char **argv) {
  rxros::init(argc, argv, "listener");
  rxros::observable::from_topic<std_msgs::String>("/chatter", 1000)
    .subscribe ( [] (const std_msgs::String& msg)
    {
      ROS_INFO_STREAM ("I heard: [" << msg.data << "]");
    });
  rxros::spin();
  return 0;
}
```



The Listener Example

Key points

- **Problem**
  - We have a simple mental model in ROS: a **flow graph of messages**
  - **We think about callbacks** when we realize it
  - Among the most complex control-flow constructs
- **Solution**
  - Reactive programming gives **simple control-flow**
  - **Flow** of information is **explicit in the code**

The Talker Example

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "talker");
  ros::NodeHandle n;
  ros::Publisher chatter_pub =
    n.advertise<std_msgs::String>("chatter",10);
  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());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

```
int main(int argc, char **argv)
{
  rxros::init(argc, argv, "talker");
  const std::string hello = "hello world ";

  rxcpp::observable<>::
    interval (std::chrono::milliseconds (10))
    | map ([&](int i)
        { return mk_msg(hello + std::to_string(i)); })
    | tap ([](const std_msgs::String& msg)
        { ROS_INFO_STREAM (msg.data); })
    | publish_to_topic<std_msgs::String>
        ("/chatter", 1000);
  rxros::spin();
  return 0;
}
```

The Talker Example

Key points

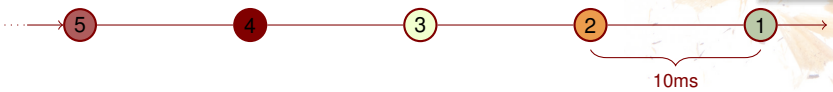
- **Problem**
  - We have a simple mental model in ROS: a **flow graph of messages**
  - **We think about loops, intervals, counters incremented** when we realize it
- **Solution**
  - Functional programming **raises the abstraction level**
  - We think about a **incremented stream with a frequency**
  - And we **transform** this stream (or messages in it)
- In RxRos publisher and subscriber look **similar**: both are **pipelines**
- In classic ROS they are **very different**: callback vs a loop
- RxROS **parallelizes** pipeline processing
- When you are avoiding callbacks, and remain pure (no side effects) as much as possible, the **need for locks decreases**, and with them concurrency problems

The Talker Example

Marble diagram

```
interval (std::chrono::milliseconds (10))
| map ([&](int i)
    { return mk_msg(hello + std::to_string(i)); })
| tap ([](const std_msgs::String& msg)
    { ROS_INFO_STREAM (msg.data); })
| publish_to_topic<std_msgs::String>
    ("/chatter", 1000);
```

```
interval (std::chrono::milliseconds (10))
```




```
| map ([&](int i) { return mk_msg (hello+std::to_string(i)); })
```

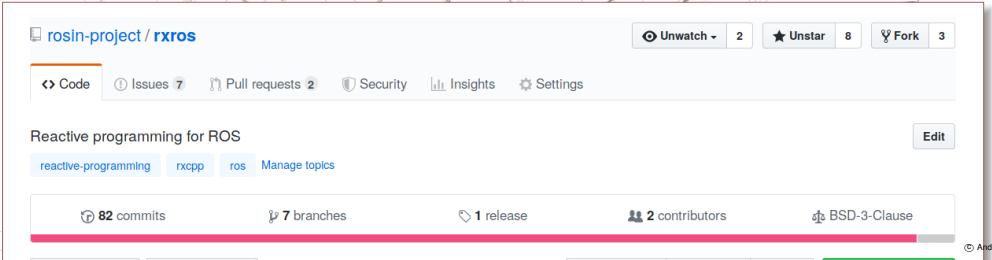

```
| tap ([](const std_msgs::String& msg) { ROS_INFO_STREAM (msg.data); })
```

publish to topic `/chatter`

The **stream is published** (string messages) to /chatter

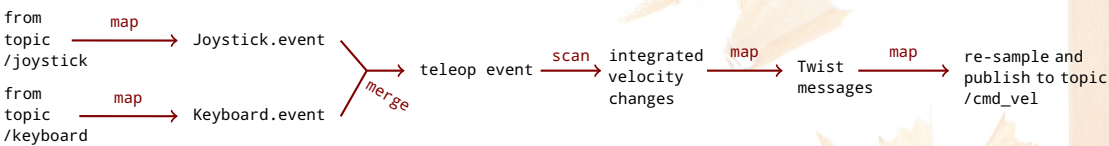
RxROS

- RxROS is a **very thin library** (326 lines of C++ header file)
- **Extends RxCPP**, a reactive programming library for C++
- Adds **several ROS-specific operators**: advertiseService, from_topic, from_device, from_yaml, sample_with_frequency, publish_to_topic, call_service
- Available in **melodic** and **kinetic**: `apt install ros-melodic-rxros`
- Available on **GitHub** `https://github.com/rosin-project/rxros`
- Some **examples** `https://github.com/rosin-project/rxros_examples`



VelocityPublisher / TeleOp

```
auto joyObsrv = rxros::observable::from_topic<rxros_teleop_msgs::Joystick>("/joystick") // (
    | map([](rxros_teleop_msgs::Joystick joy) { return joy.event; });
auto keyObsrv = rxros::observable::from_topic<rxros_teleop_msgs::Keyboard>("/keyboard") // (
    | map([](rxros_teleop_msgs::Keyboard key) { return key.event; });
joyObsrv.merge(keyObsrv)                                  // merge the joystick and keyboard mes
| scan(std::make_tuple(0.0, 0.0), teleop2VelTuple)        // turn the teleop stream into a linea
| map(velTuple2TwistMsg)                                  // turn the linear and angular velocit
| sample_with_frequency(frequencyInHz)                    // take latest Twist msg and populate
| publish_to_topic<geometry_msgs::Twist>("/cmd_vel");     // publish the Twist messages to the t
```



Challenges Ahead

- **Copying** semantics and **de-allocation** of objects rather complex in C++ (comparing to managed languages)
- Unclear **impact on performance**, more threads (cost) but huge opportunties for parallelization (gain)
- Some **mental cost** in changing the programming paradigm, but there is no going back :)
- Understand **how much of ROS-based code is feasible to write this way**

RoadMap Ahead

- RxROS py
- Action Lib
- RxROS 2, DDS
- RxROS Java? Scala? C#? F#?
- We **seek contributors**!

```
interval (std::chrono::milliseconds (10))
| map ([&](int i)
    { return mk_msg(hello + std::to_string(i)); })
| tap ([](const std_msgs::String& msg)
    { ROS_INFO_STREAM (msg.data); })
| publish_to_topic<std_msgs::String>
    ("/chatter", 1000);
```

Henrik Larsen
IT University of Copenhagen

Andrzej Wąsowski
IT University of Copenhagen
@AndrzejWasowski

[Reactive] Programming with [Rx]ROS

IT UNIVERSITY OF COPENHAGEN
SOFTWARE QUALITY RESEARCH
European Commission
Horizon 2020
European Union funding for Research & Innovation
ROSin