Open Source Robotics Foundation

ROS 2 Update

Deanna Hood, William Woodall

October 8, 2016

ROSCon 2016 Seoul

https://goo.gl/oCHR7H

Contents

ROS 2 overview

Overview of changes in the last year

- Details of select features

Experience porting Turtlebot

Roadmap

https://goo.gl/oCHR7H

ROS as we know it

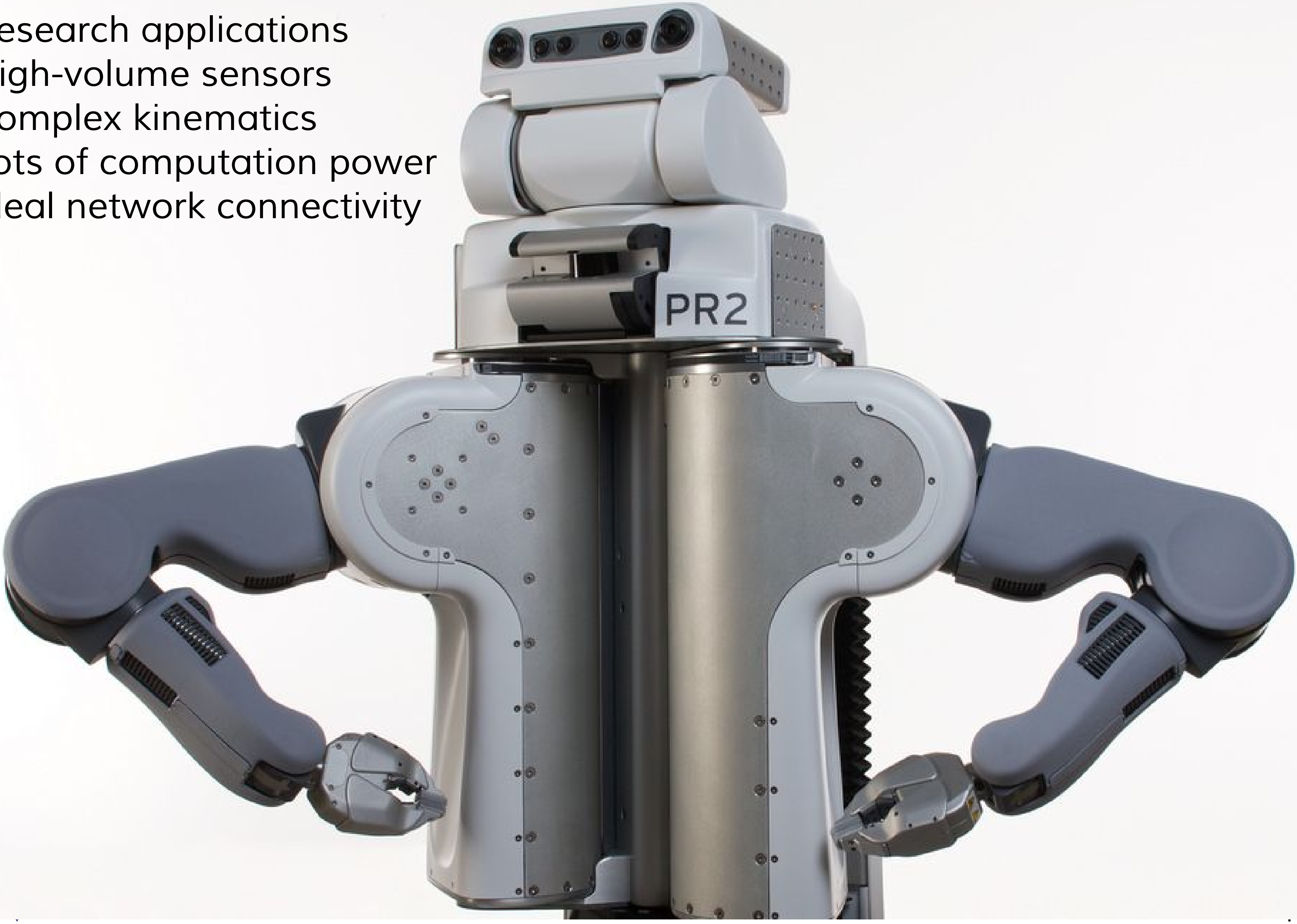
Plumbing

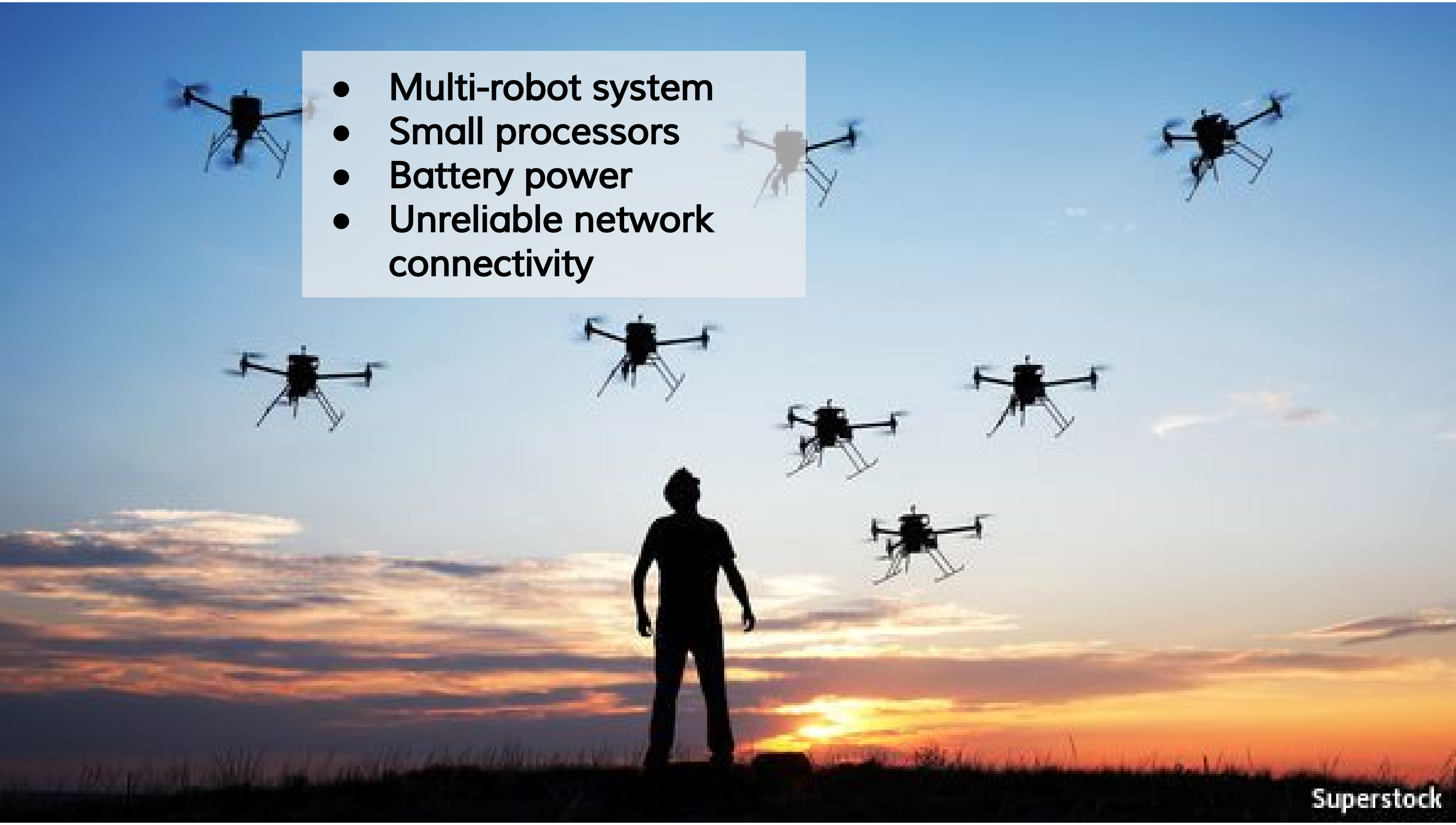
Tools

Capabilities

Ecosystem

- Research applications
- High-volume sensors
- Complex kinematics
- Lots of computation power
- Ideal network connectivity





- Multi-robot system
- Small processors
- Battery power
- Unreliable network connectivity

Superstock

Goals of ROS 2

Support multi-robot systems involving unreliable networks

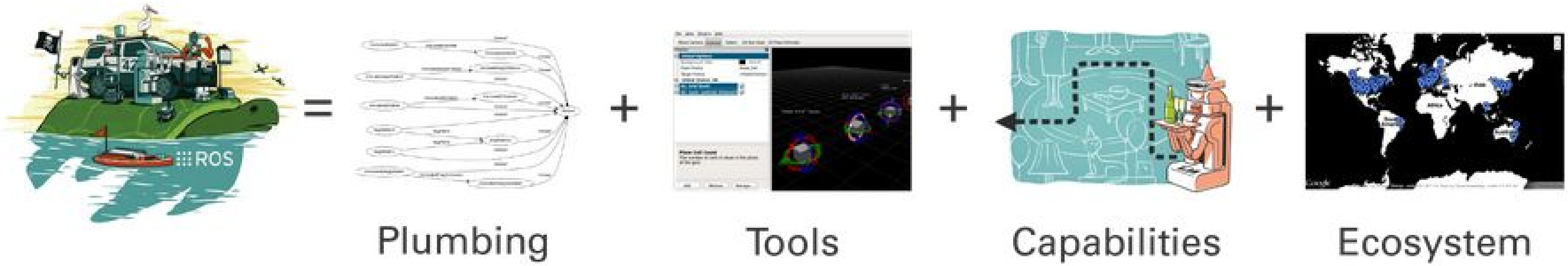
Remove the gap between prototyping and final products

"Bare-metal" micro controller

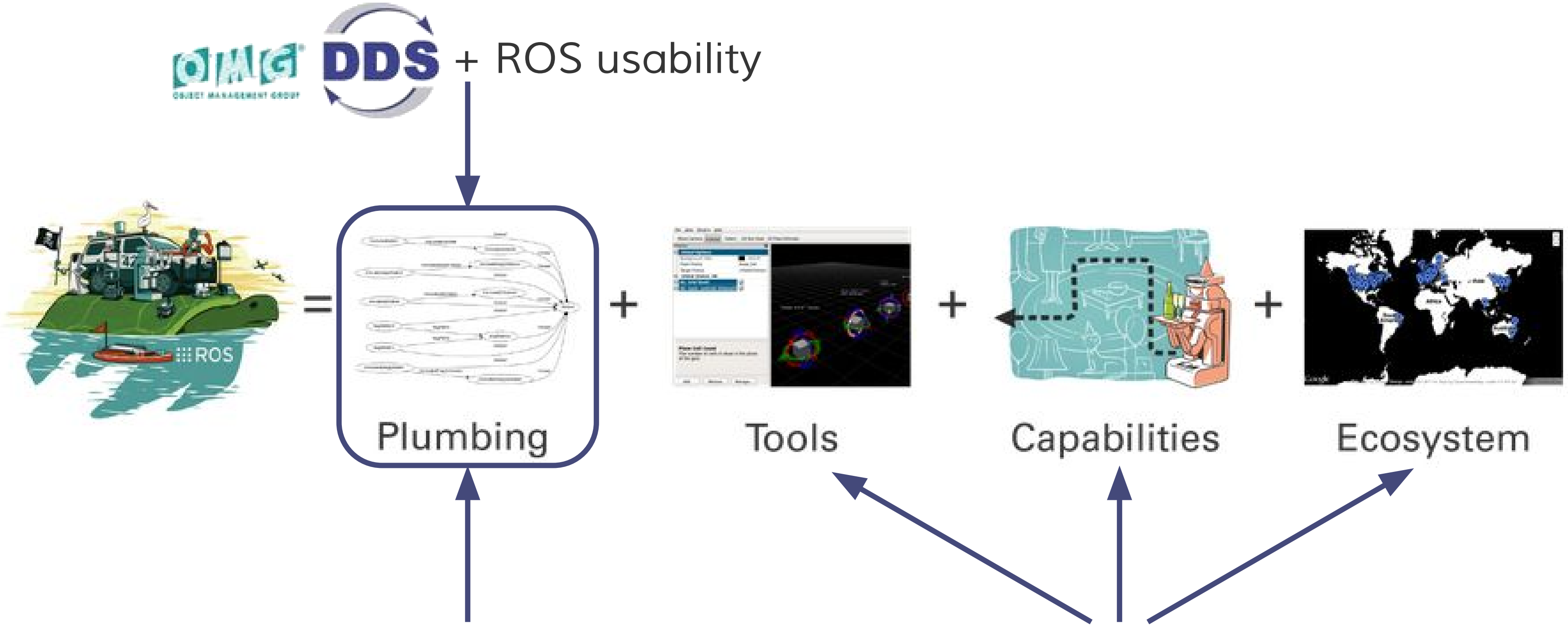
Support for real-time control

Cross-platform support

ROS 2



ROS 2

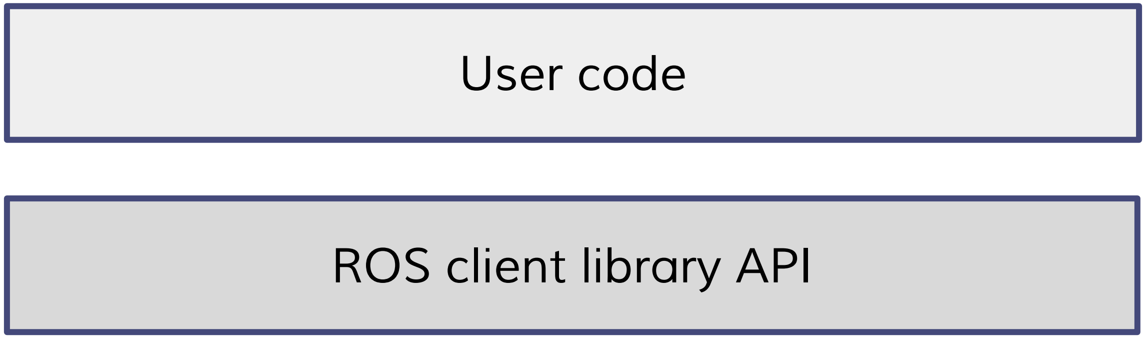


less time spent here

means

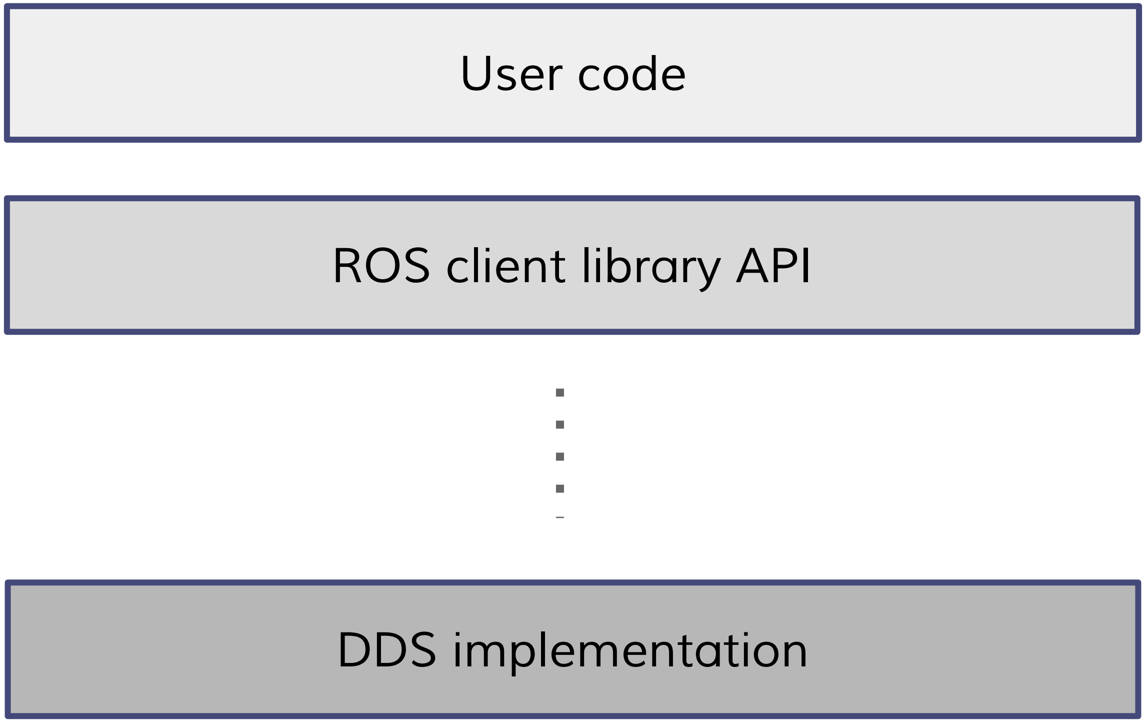
more time to spend here

Architectural overview


User code

ROS client library API

Architectural overview

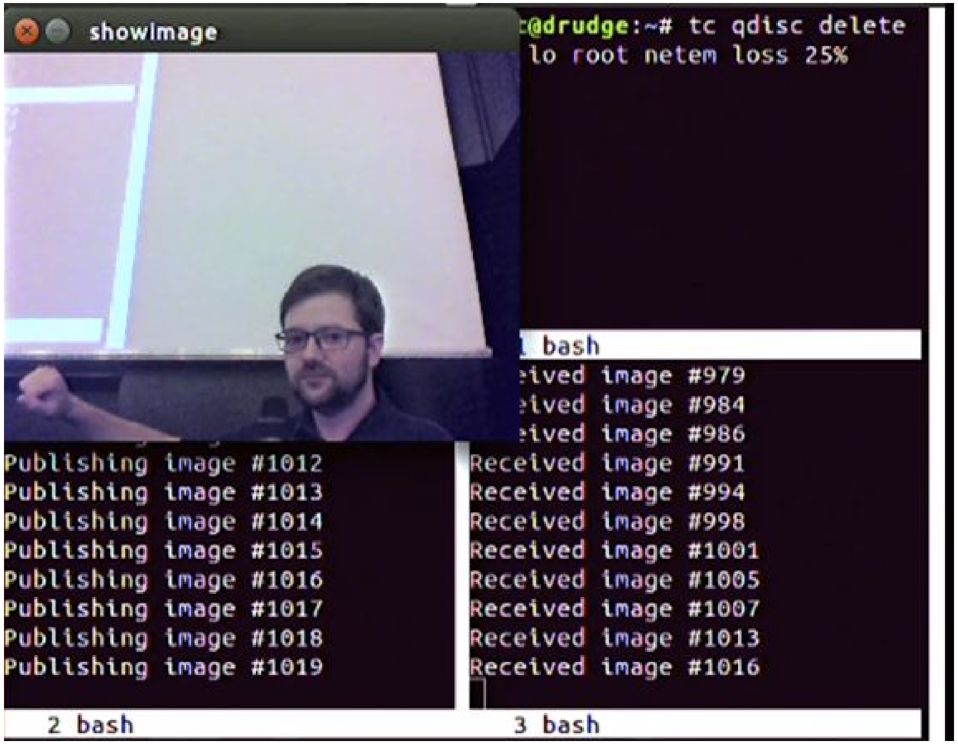

User code

ROS client library API

DDS implementation

DDS = discovery + serialization + transport

ROSCon 2015 demos



```
@drudge:~# tc qdisc delete
 lo root netem loss 25%
```

```
1 bash
```

```
Publishing image #1012      eived image #979
Publishing image #1013      eived image #984
Publishing image #1014      eived image #986
Publishing image #1015      Received image #991
Publishing image #1016      Received image #994
Publishing image #1017      Received image #998
Publishing image #1018      Received image #1001
Publishing image #1019      Received image #1005
                            Received image #1007
                            Received image #1013
                            Received image #1016
 2 bash                      3 bash
```

Quality of Service demo for lossy networks using ROS 2

https://github.com/ros2/ros2/wiki/Tutorials

Bridge communication between ROS 1 and ROS 2

Real-time safe code using ROS 2

Efficient intra-process communication using ROS 2

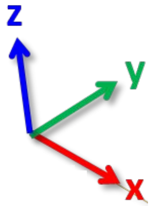
ROS 2 on “bare-metal” microcontrollers

What’s new this year

Changes since ROSCon 2015: user-facing

Windows feature parity (alpha 2)

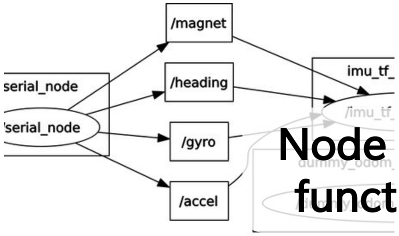
Fast RTPS added as a supported middleware (alpha 3)



Partial port of tf2 including the core libraries (alpha 3)

Python client library (alpha 4)

32-bit and 64-bit ARM added as experimentally supported platforms (alpha 5)



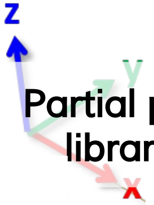
Node “wait for service” functionality (alpha 6)

Turtlebot demo using ported code from ROS 1 (alpha 7)

Changes since ROSCon 2015

Windows feature parity (alpha 2)

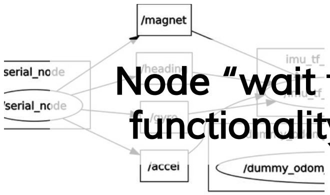
Fast RTPS supported as middleware (alpha 3)



Partial port of core tf2 libraries (alpha 3)

Python client library (alpha 4)

32- and 64-bit ARM experimentally supported platforms (alpha 5)



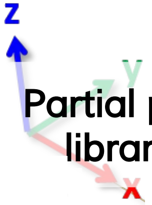
Node "wait for service" functionality (alpha 6)

Turtlebot demo using ported code from ROS 1 (alpha 7)

Changes since ROSCon 2015

Windows feature parity (alpha 2)

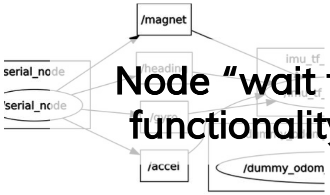
Fast RTPS supported as middleware (alpha 3)



Partial port of core tf2 libraries (alpha 3)

Python client library (alpha 4)

32- and 64-bit ARM experimentally supported platforms (alpha 5)



Node "wait for service" functionality (alpha 6)

Turtlebot demo using ported code from ROS 1 (alpha 7)

ROS Client Library implementation (rcl) (from alpha 3, services alpha 5)

Support for C messages (as opposed to C++) (alphas 4, 5, 7)

Refactored C++ client library to use rcl (alpha 6)

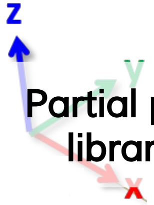
ROS graph events (alpha 6)

Improved support for large messages (images) with Connext and Fast RTPS (alpha 6, alpha 7)

Changes since ROSCon 2015

Windows feature parity (alpha 2)

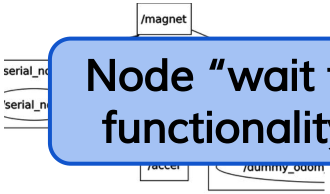
Fast RTPS supported as middleware (alpha 3)



Partial port of core tf2 libraries (alpha 3)

Python client library (alpha 4)

32- and 64-bit ARM experimentally supported platforms (alpha 5)



Node “wait for service” functionality (alpha 6)

Turtlebot demo using ported code from ROS 1 (alpha 7)

ROS Client Library implementation (rcl) (from alpha 3, services alpha 5)

Support for C messages (as opposed to C++) (alphas 4, 5, 7)

Refactored C++ client library to use rcl (alpha 6)

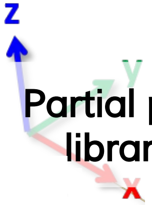
ROS graph events (alpha 6)

Improved support for large messages (images) with Connext and Fast RTPS (alpha 6, alpha 7)

Changes since ROSCon 2015

Windows feature parity (alpha 2)

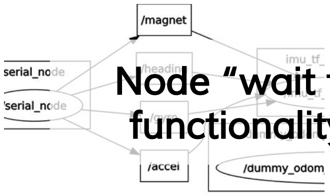
Fast RTPS supported as middleware (alpha 3)



Partial port of core tf2 libraries (alpha 3)

Python client library (alpha 4)

32- and 64-bit ARM experimentally supported platforms (alpha 5)



Node "wait for service" functionality (alpha 6)

Turtlebot demo using ported code from ROS 1 (alpha 7)

ROS Client Library implementation (rcl) (from alpha 3, services alpha 5)

Support for C messages (as opposed to C++) (alphas 4, 5, 7)

Refactored C++ client library to use rcl (alpha 6)

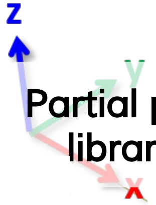
ROS graph events (alpha 6)

Improved support for large messages (images) with Connext and Fast RTPS (alpha 6, alpha 7)

Changes since ROSCon 2015

Windows feature parity (alpha 2)

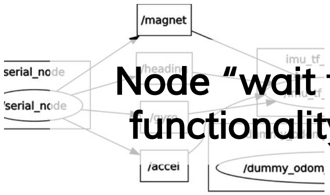
Fast RTPS supported as middleware (alpha 3)



Partial port of core tf2 libraries (alpha 3)

Python client library (alpha 4)

32- and 64-bit ARM experimentally supported platforms (alpha 5)



Node “wait for service” functionality (alpha 6)

Turtlebot demo using ported code from ROS 1 (alpha 7)

ROS Client Library implementation (rcl) (from alpha 3, services alpha 5)

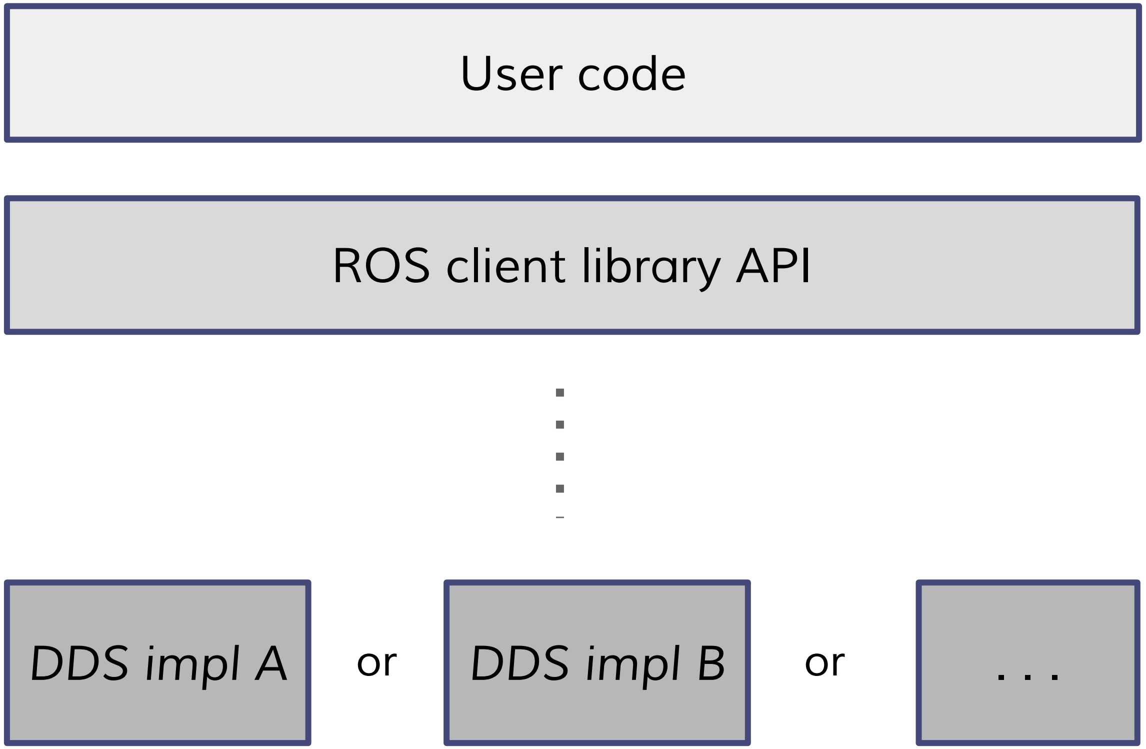
Support for C messages (as opposed to C++) (alphas 4, 5, 7)

Refactored C++ client library to use rcl (alpha 6)

ROS graph events (alpha 6)

Improved support for large messages (images) with Connext and Fast RTPS (alpha 6, alpha 7)

Architectural overview

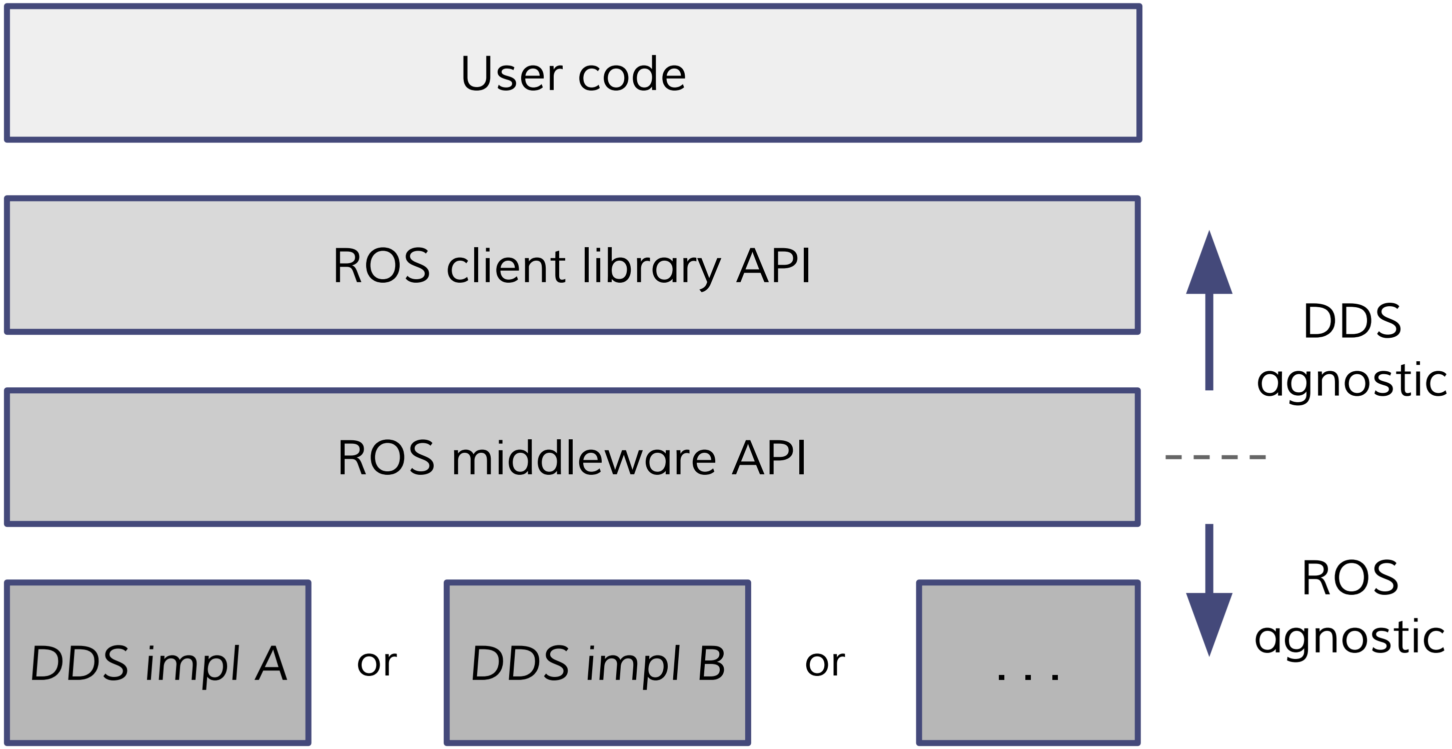


User code

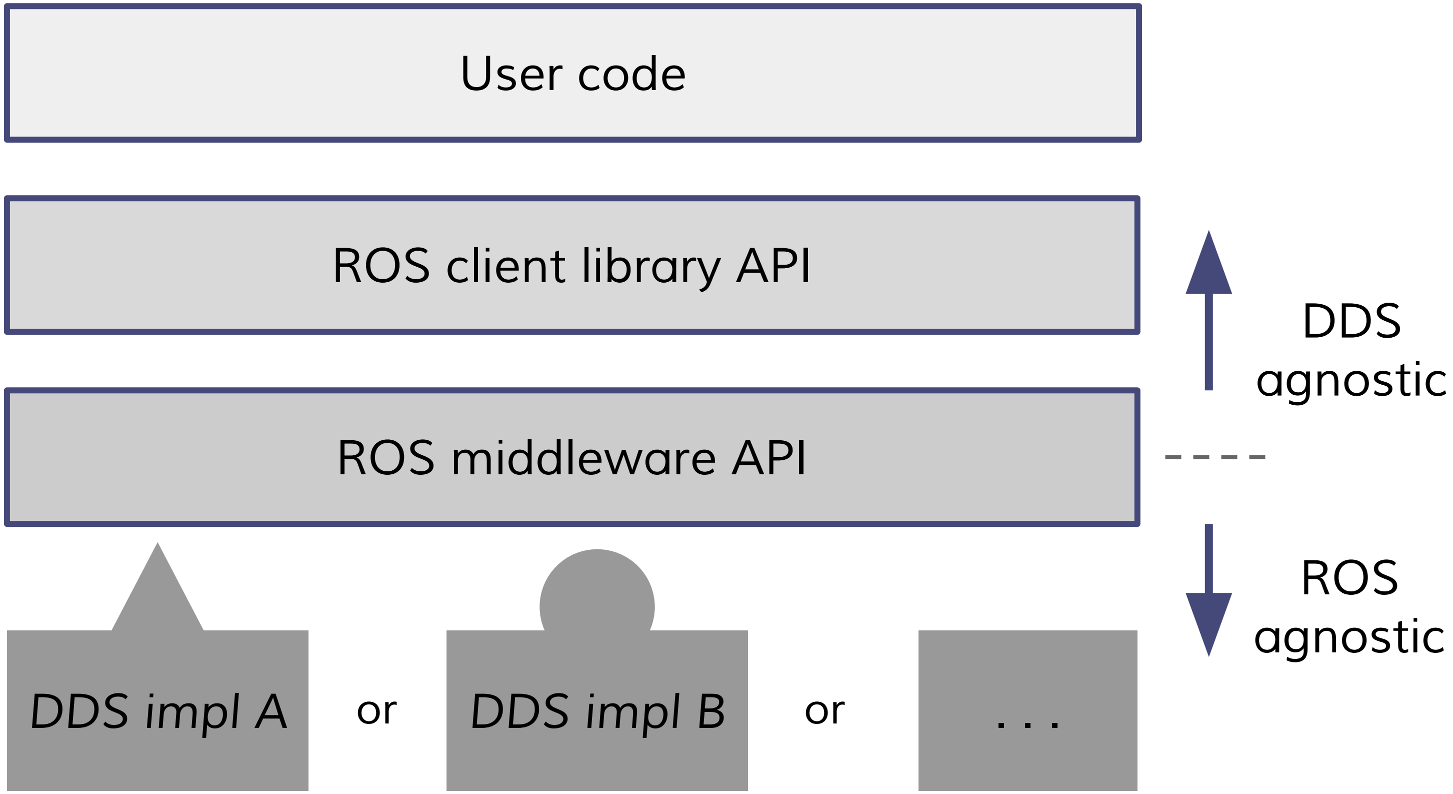
ROS client library API

DDS impl A or *DDS impl B* or . . .

Architectural overview



Architectural overview



User code

ROS client library API

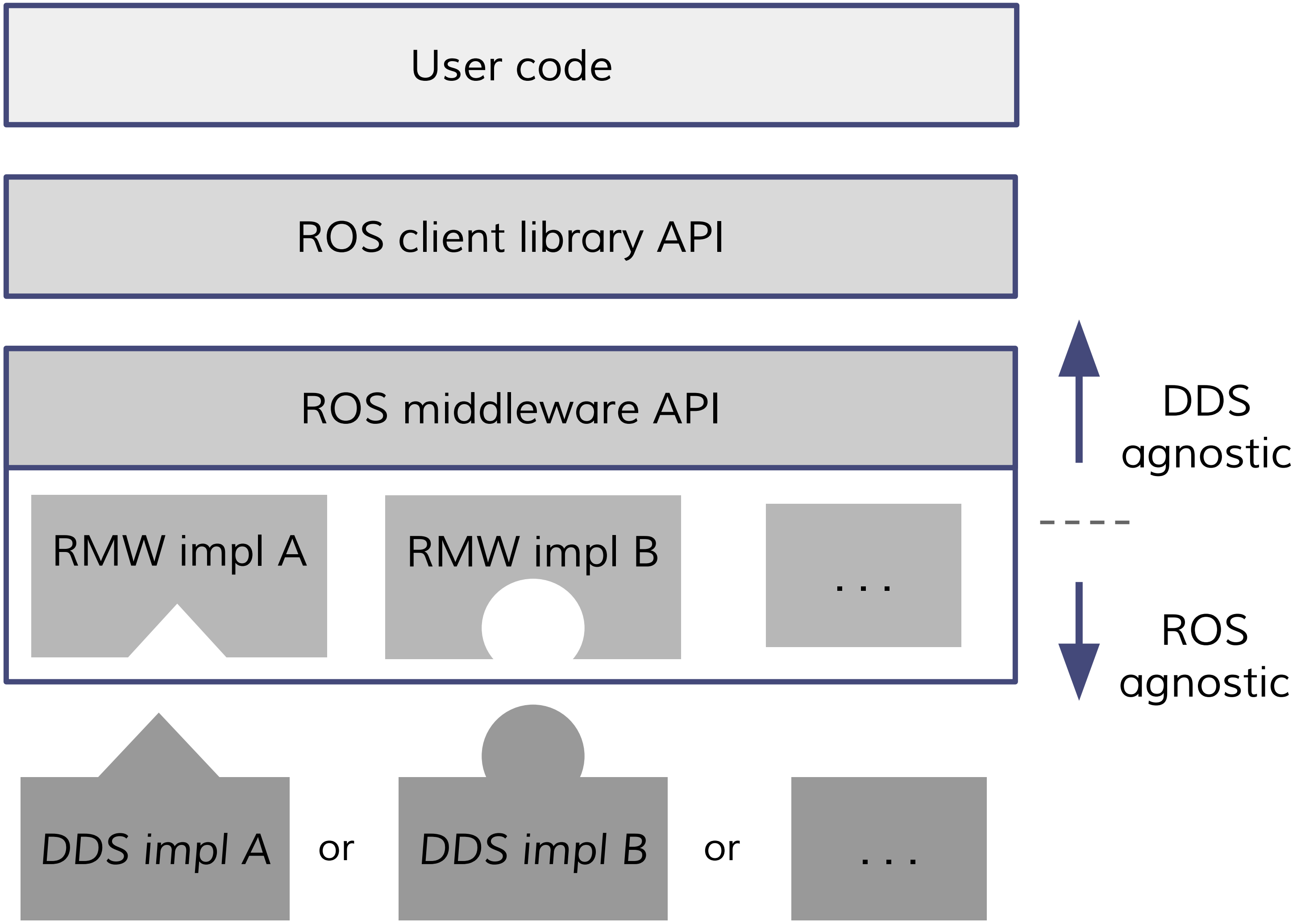
ROS middleware API

DDS impl A or *DDS impl B* or . . .

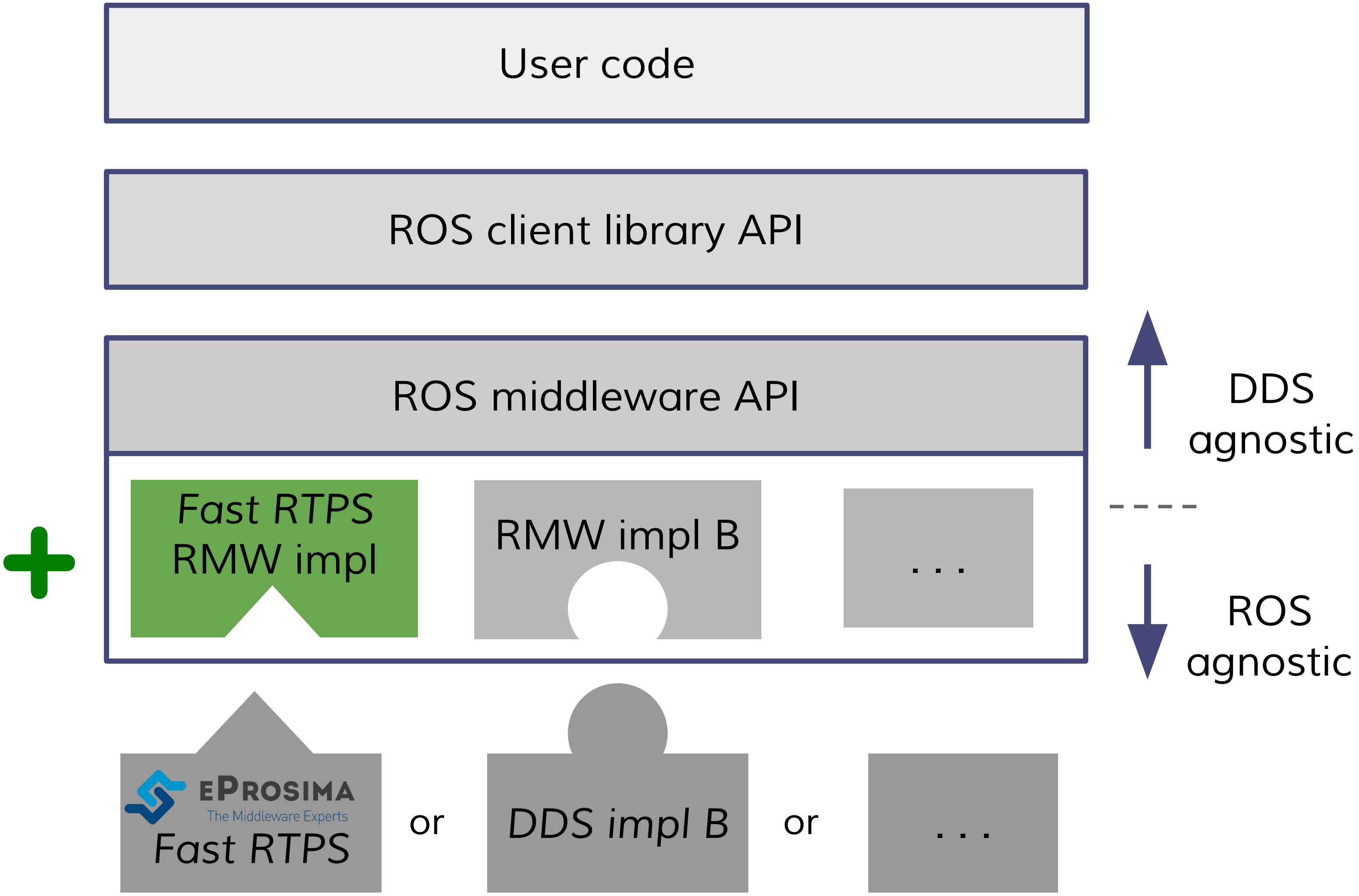
DDS agnostic

ROS agnostic

Architectural overview



Architectural overview



User code

ROS client library API

ROS middleware API

Fast RTPS RMW impl

RMW impl B

. . .

DDS agnostic

ROS agnostic

Fast RTPS

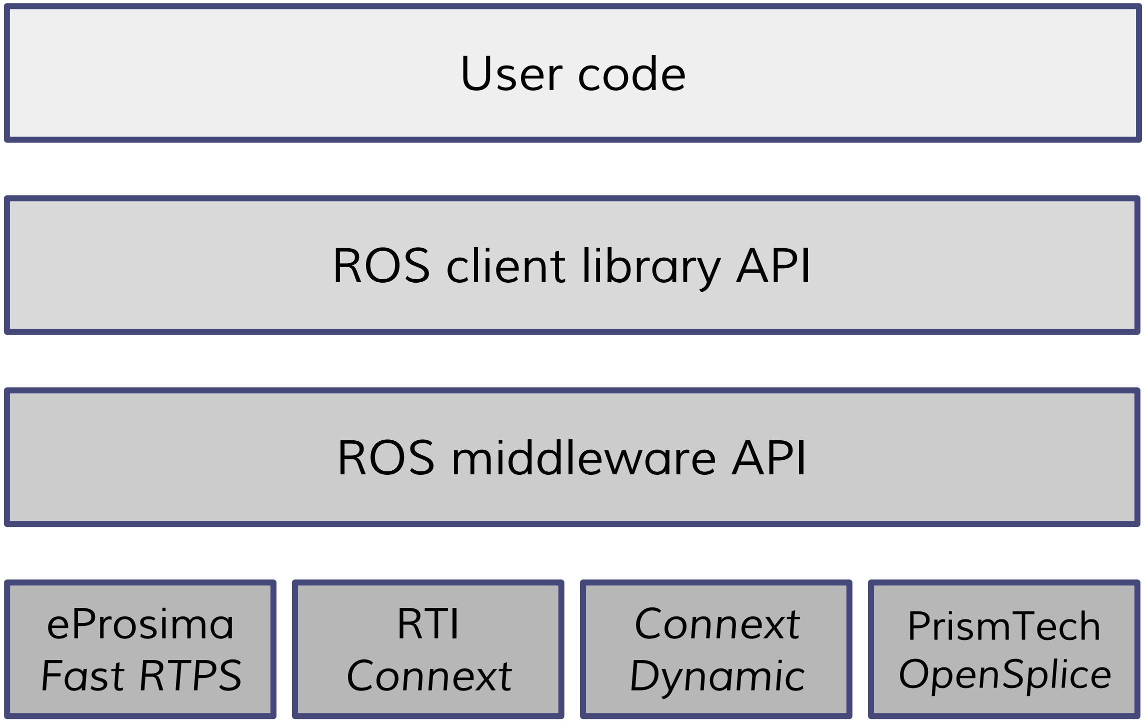
or

DDS impl B

or

. . .

Supported vendors until October 2016



User code

ROS client library API

ROS middleware API

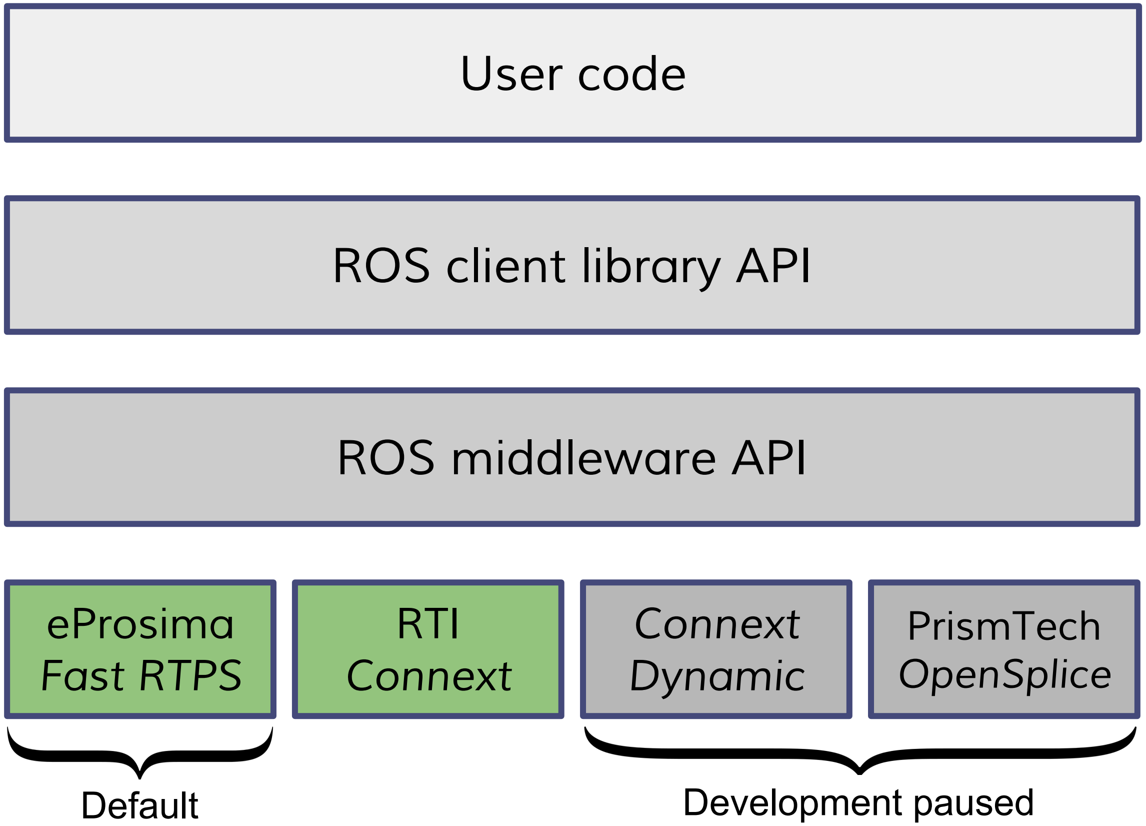
eProsima *Fast RTPS*

RTI *Connext*

Connext *Dynamic*

PrismTech *OpenSplice*

Supported vendors since October 2016



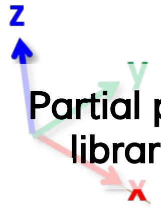
Why eProsima's Fast RTPS?

- Changed the license June 2016:
  - *LGPL -> Apache 2.0*
- Code on GitHub
  - *https://github.com/eProsima/Fast-RTPS*
- Responsive to issues and pull requests
- Added features needed to support ROS 2
  - *Fragmentation of large messages*
  - *Graph change notifications*
- CMake buildsystem

Changes since ROSCon 2015

Windows feature parity (alpha 2)

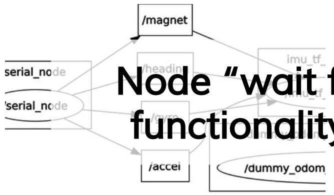
Fast RTPS supported as middleware (alpha 3)



Partial port of core tf2 libraries (alpha 3)

Python client library (alpha 4)

32- and 64-bit ARM experimentally supported platforms (alpha 5)



Node “wait for service” functionality (alpha 6)

Turtlebot demo using ported code from ROS 1 (alpha 7)

ROS Client Library implementation (rcl) (from alpha 3, services alpha 5)

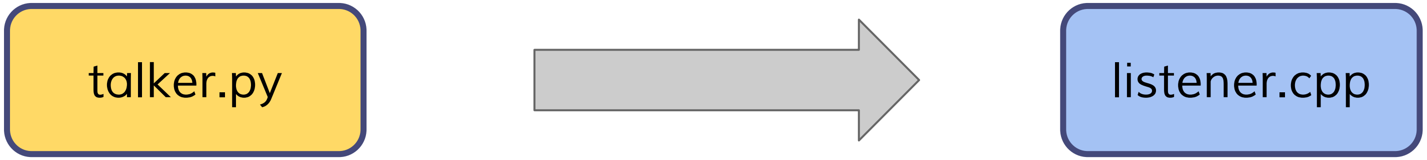
Support for C messages (as opposed to C++) (alphas 4, 5, 7)

Refactored C++ client library to use rcl (alpha 6)

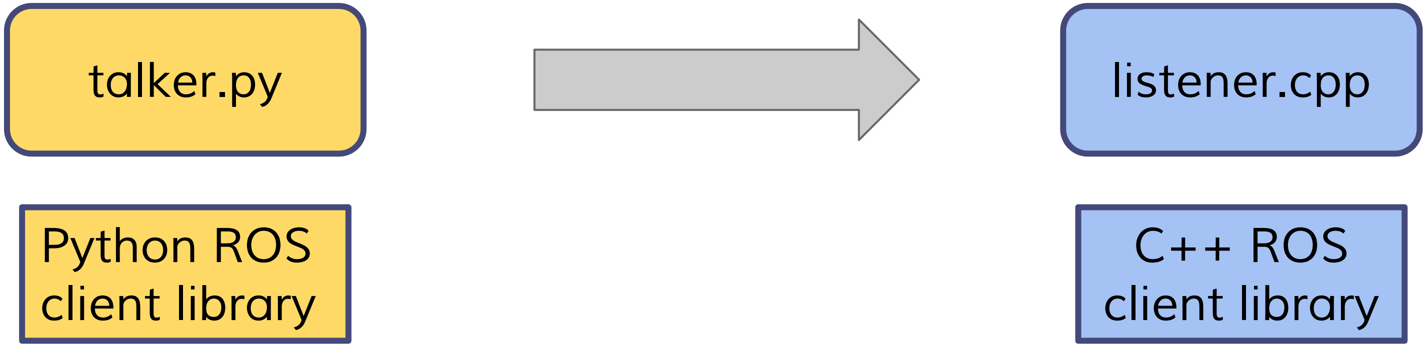
ROS graph events (alpha 6)

Improved support for large messages (images) with Connext and Fast RTPS (alpha 6, alpha 7)

ROS client libraries



ROS client libraries



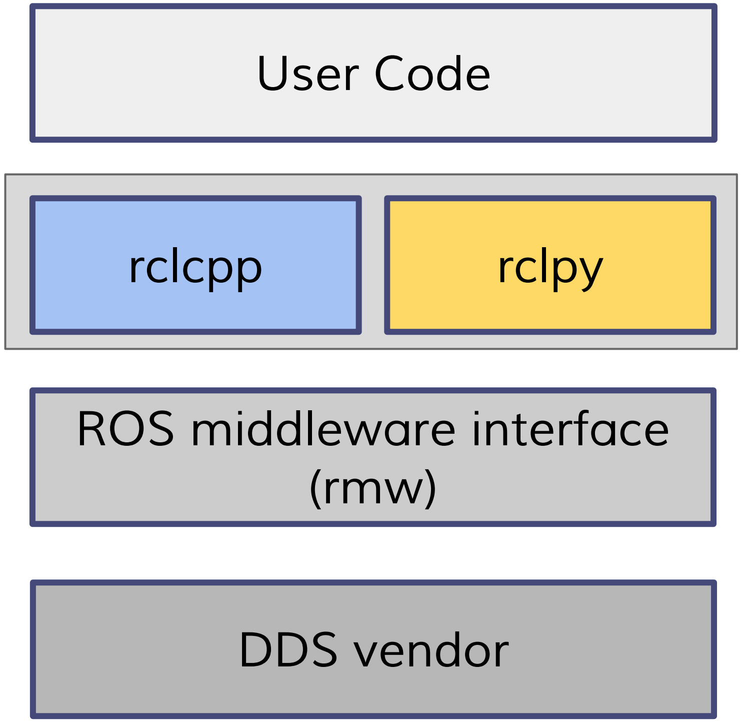
talker.py

Python ROS client library

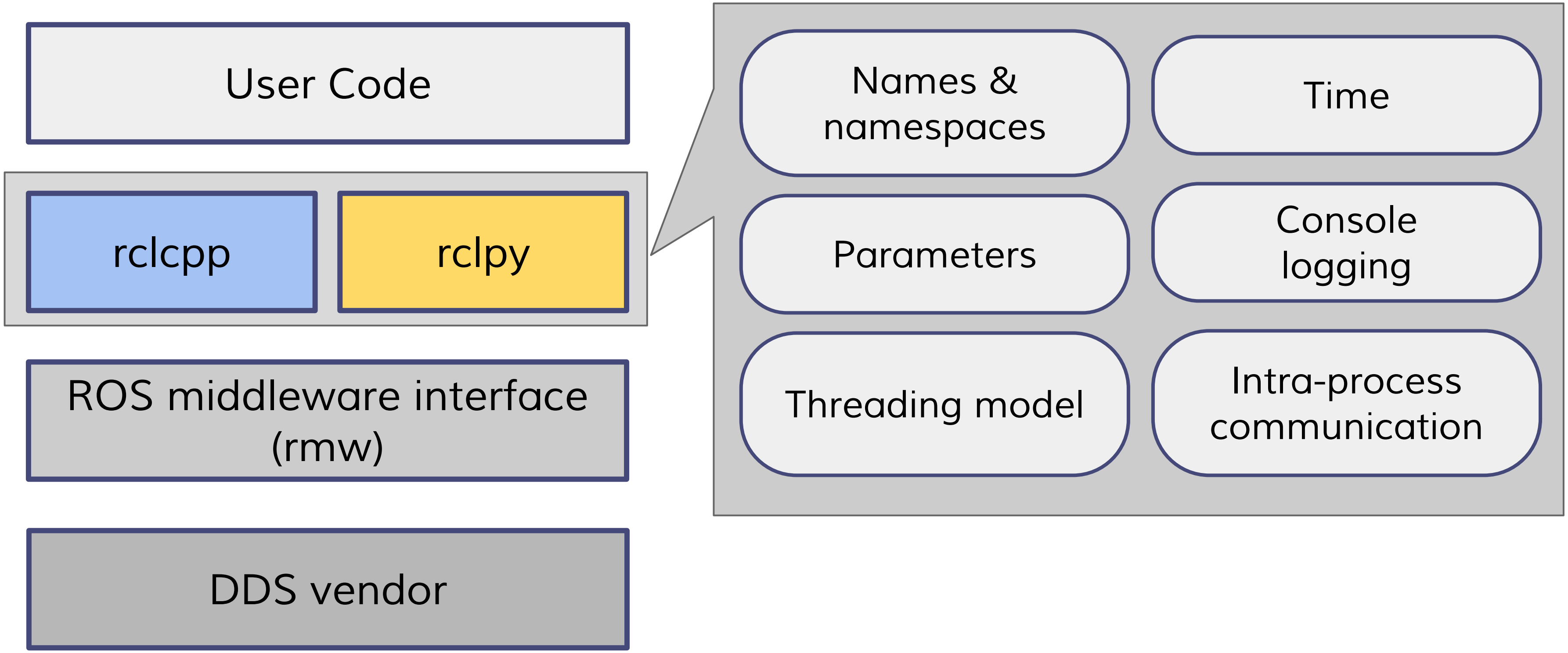
listener.cpp

C++ ROS client library

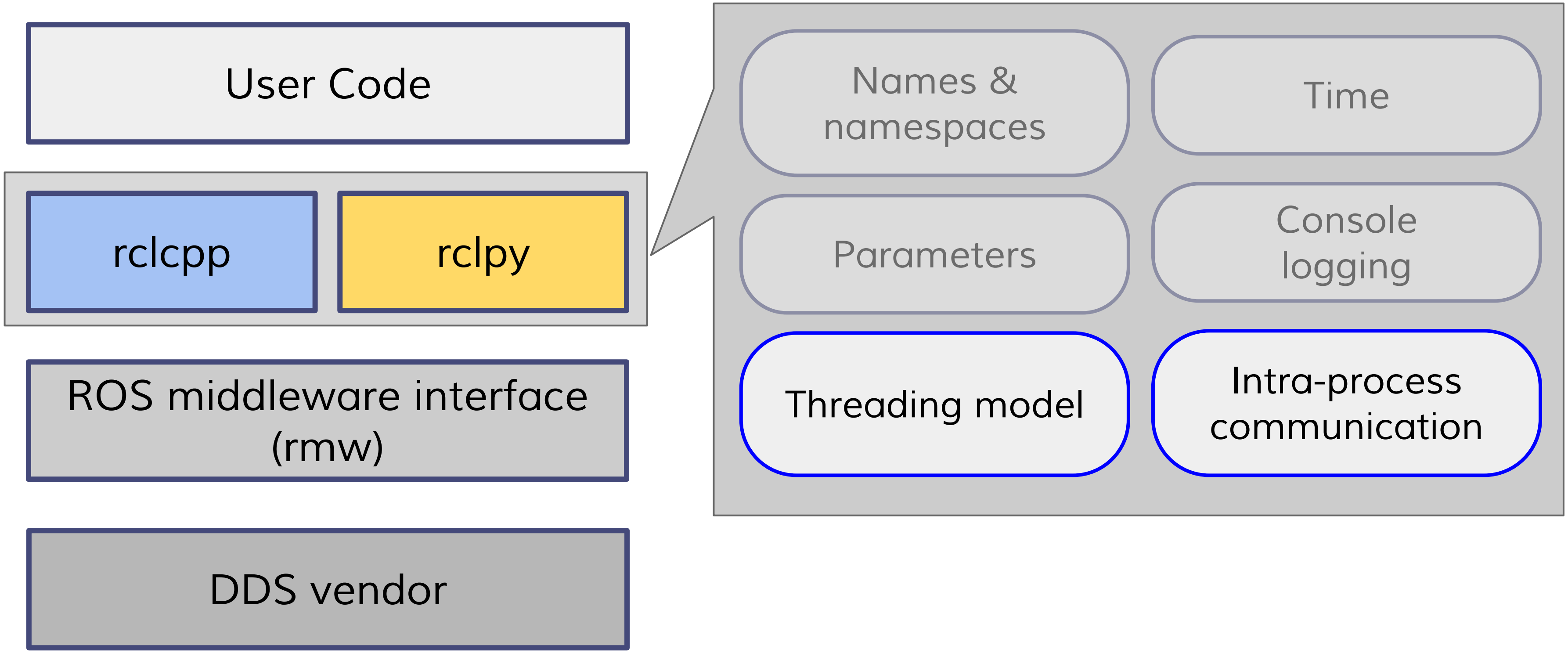
ROS client libraries



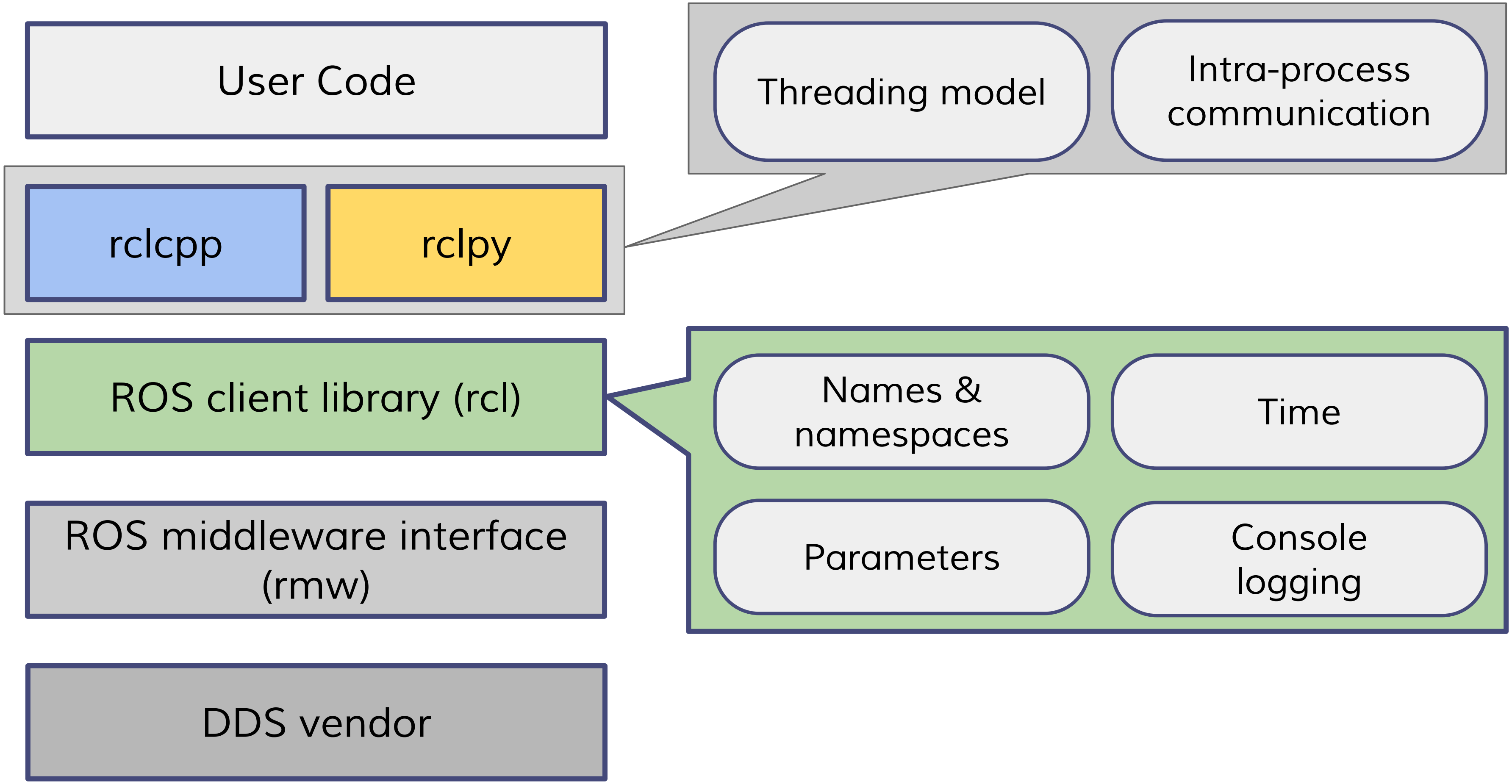
ROS client libraries



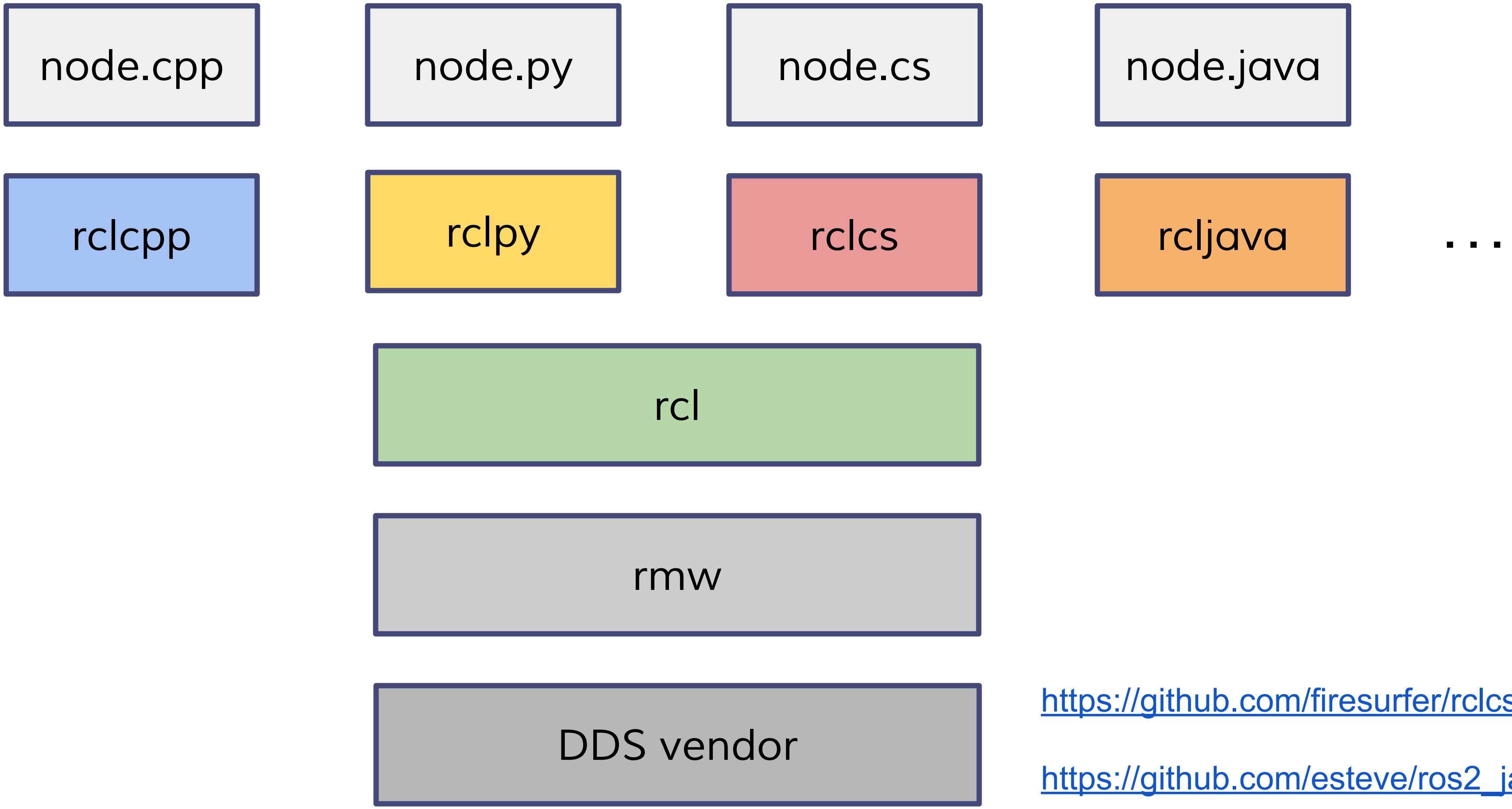
ROS client libraries



ROS client libraries



ROS client libraries

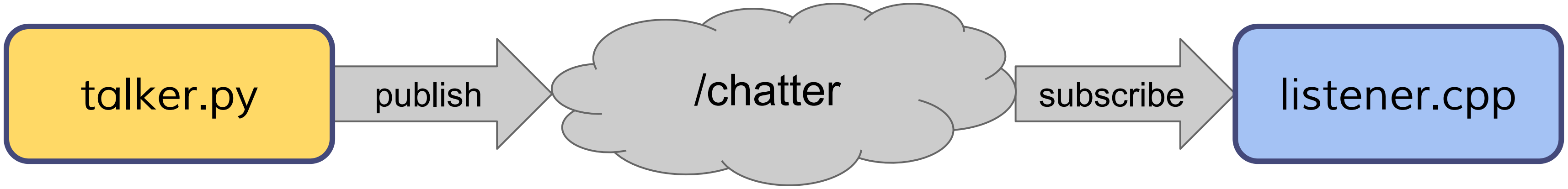


https://github.com/firesurfer/rclcs

https://github.com/esteve/ros2_java

Tracing talker-listener

Consider this talker-listener example:

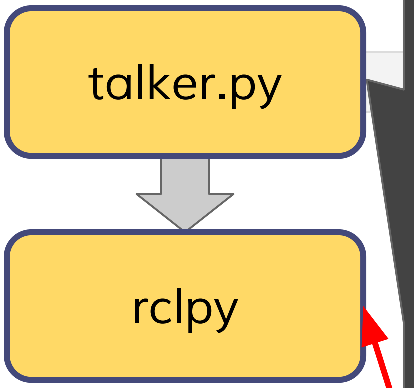


Tracing talker-listener

talker.py

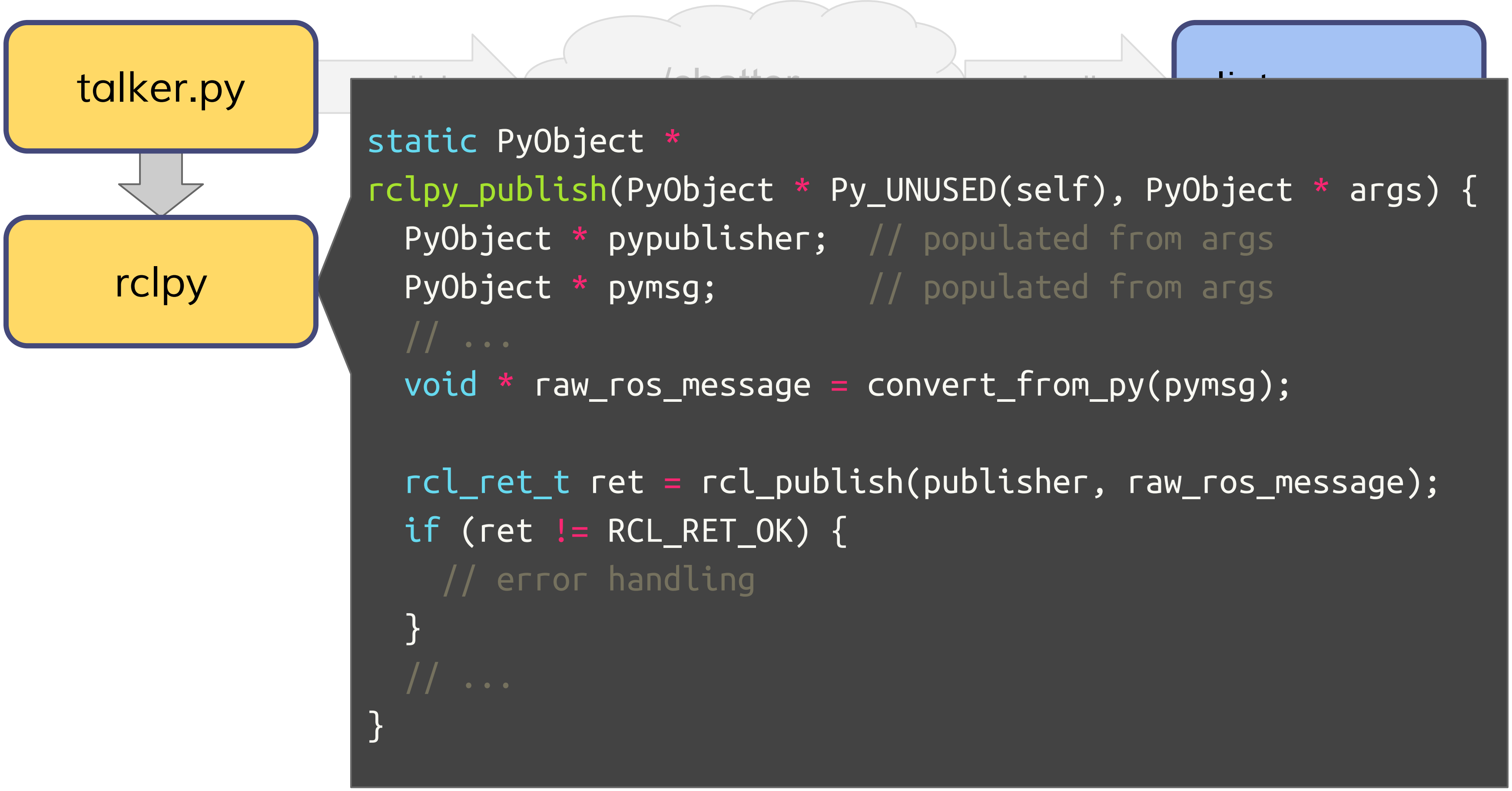
```
rclpy.init()

node = rclpy.create_node('talker')
chatter_pub = node.create_publisher(
  std_msgs.msg.String, 'chatter')
msg = std_msgs.msg.String()
i = 1

while True:
    msg.data = 'Hello World: {0}'.format(i)
    i += 1
    print('Publishing: "{0}"'.format(msg.data))
    chatter_pub.publish(msg)
```

Tracing talker-listener



```
rclpy.init()

node = rclpy.create_node('talker')
chatter_pub = node.create_publisher(
  std_msgs.msg.String, 'chatter')
msg = std_msgs.msg.String()
i = 1

while True:
    msg.data = 'Hello World: {0}'.format(i)
    i += 1
    print('Publishing: "{0}"'.format(msg.data))
    chatter_pub.publish(msg)
```

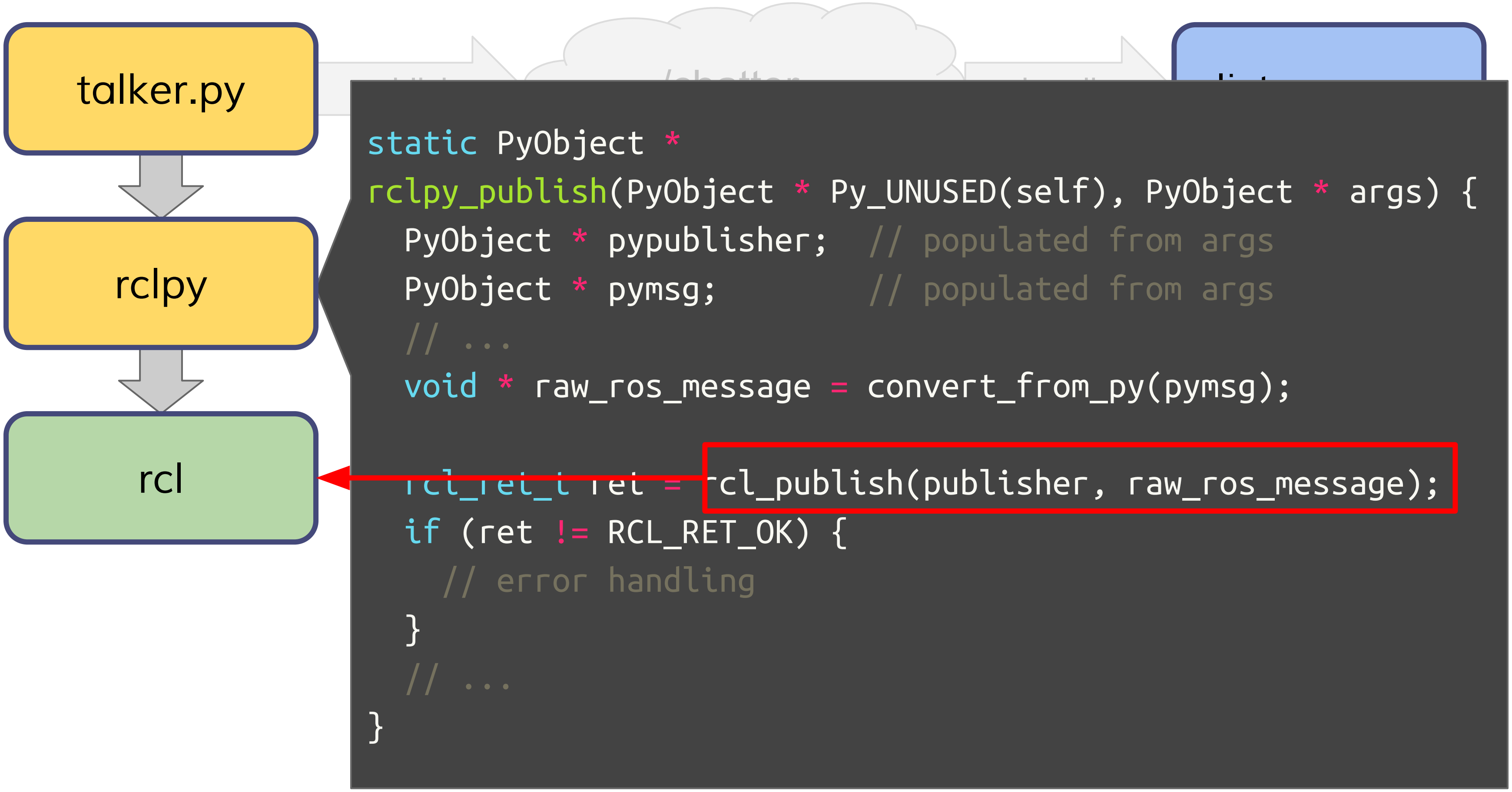
Tracing talker-listener



```
static PyObject *
rclpy_publish(PyObject * Py_UNUSED(self), PyObject * args) {
  PyObject * pypublisher;  // populated from args
  PyObject * pymsg;        // populated from args
  // ...
  void * raw_ros_message = convert_from_py(pymsg);

  rcl_ret_t ret = rcl_publish(publisher, raw_ros_message);
  if (ret != RCL_RET_OK) {
    // error handling
  }
  // ...
}
```

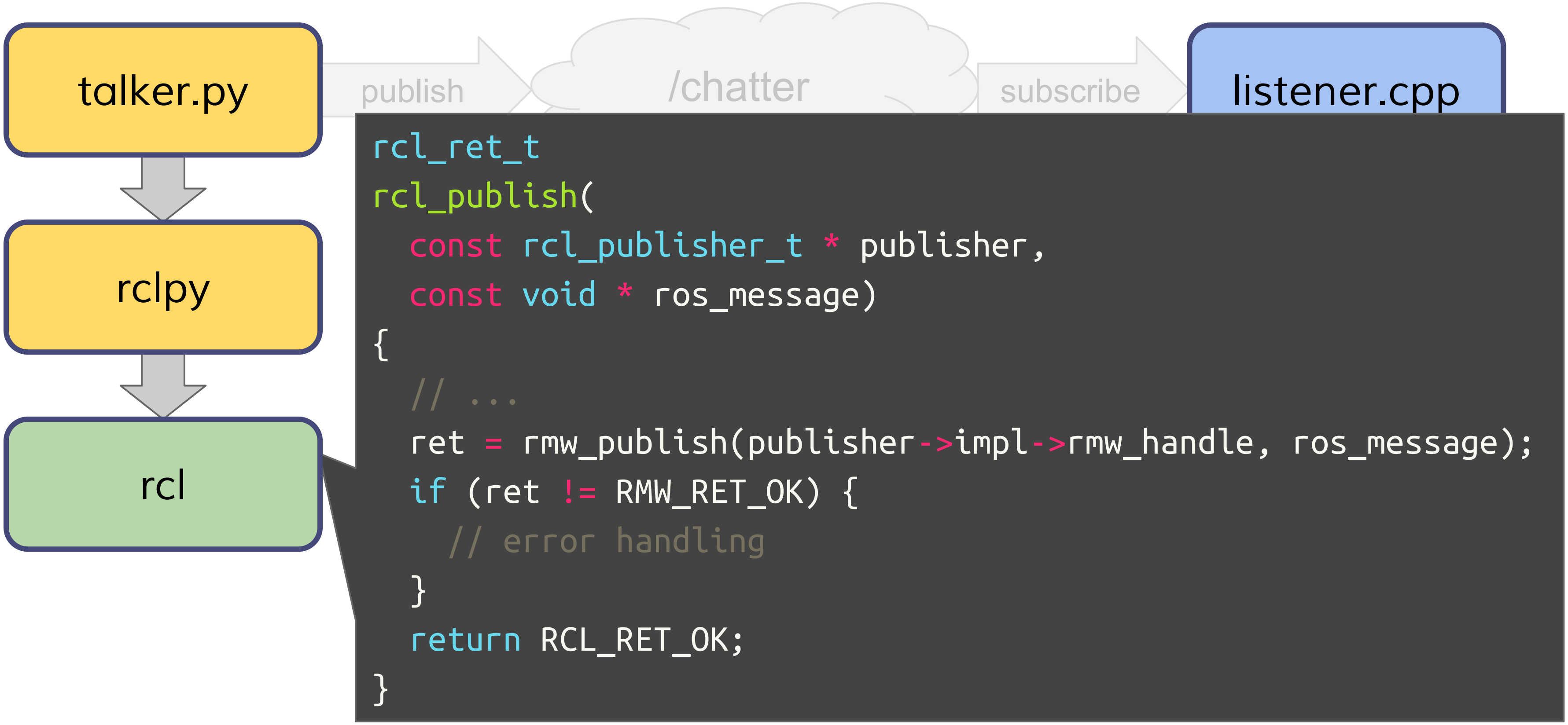
Open Source Robotics Foundation

Tracing talker-listener



```
static PyObject *
rclpy_publish(PyObject * Py_UNUSED(self), PyObject * args) {
  PyObject * pypublisher;  // populated from args
  PyObject * pymsg;        // populated from args
  // ...
  void * raw_ros_message = convert_from_py(pymsg);

  rcl_ret_t ret = rcl_publish(publisher, raw_ros_message);
  if (ret != RCL_RET_OK) {
    // error handling
  }
  // ...
}
```

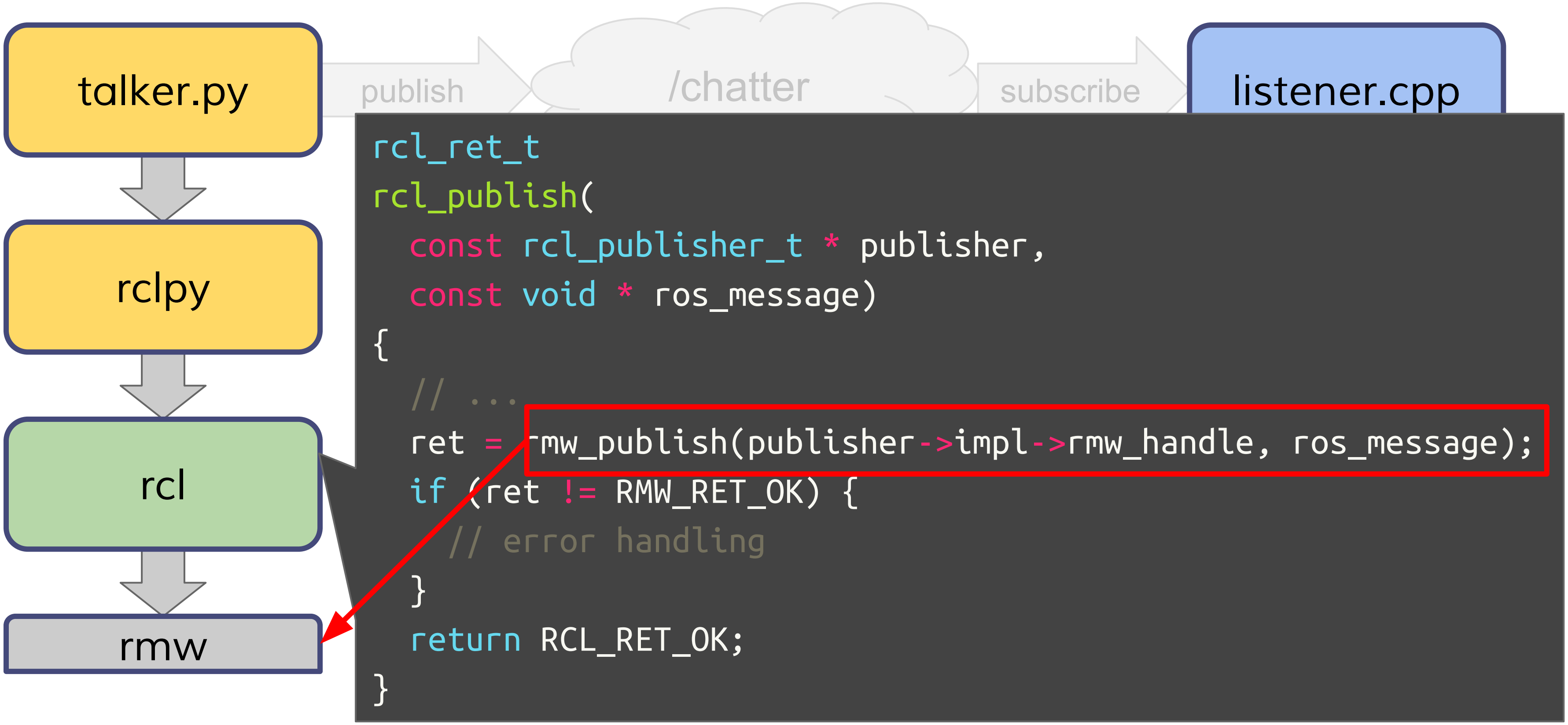
Tracing talker-listener



```
rcl_ret_t
rcl_publish(
  const rcl_publisher_t * publisher,
  const void * ros_message)
{
  // ...
  ret = rmw_publish(publisher->impl->rmw_handle, ros_message);
  if (ret != RMW_RET_OK) {
    // error handling
  }
  return RCL_RET_OK;
}
```

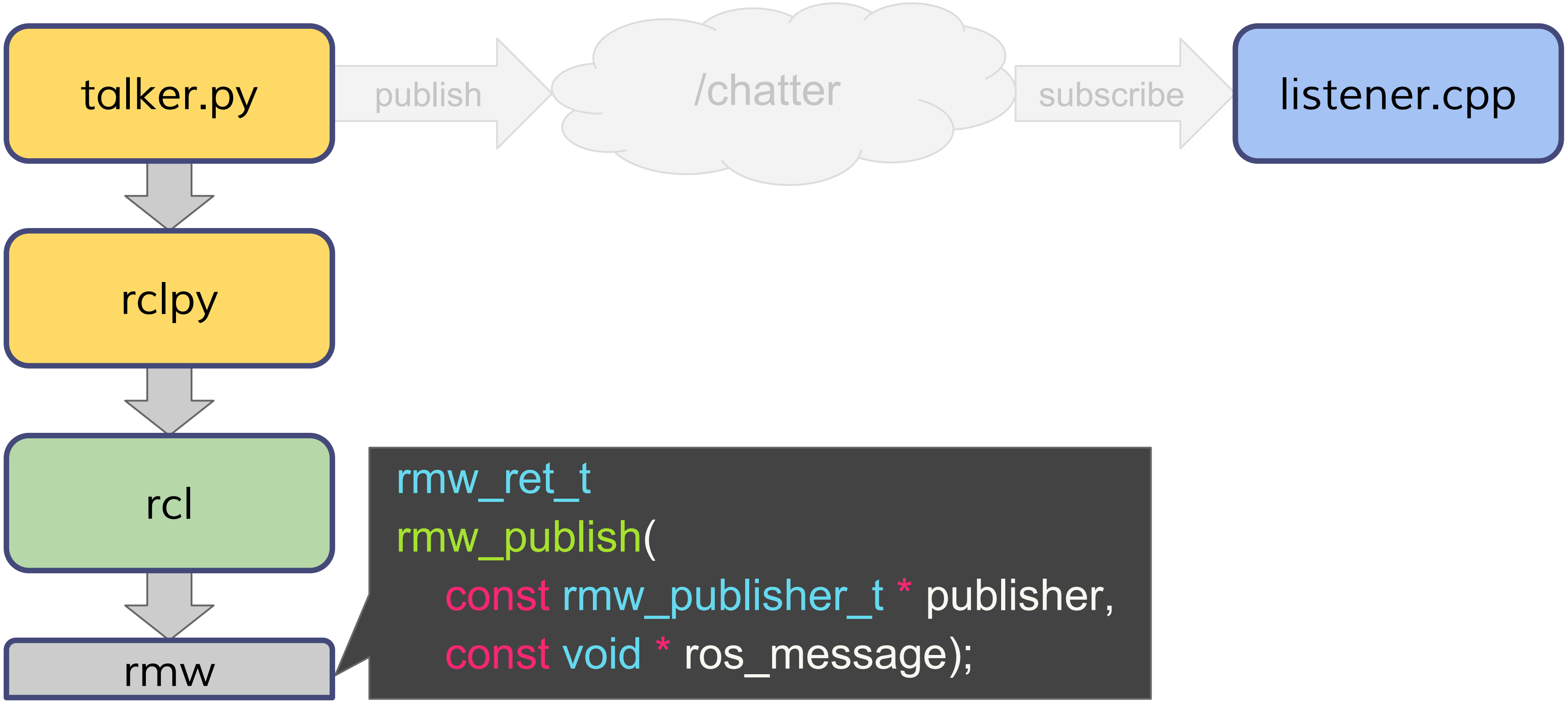
Open Source Robotics Foundation

Tracing talker-listener



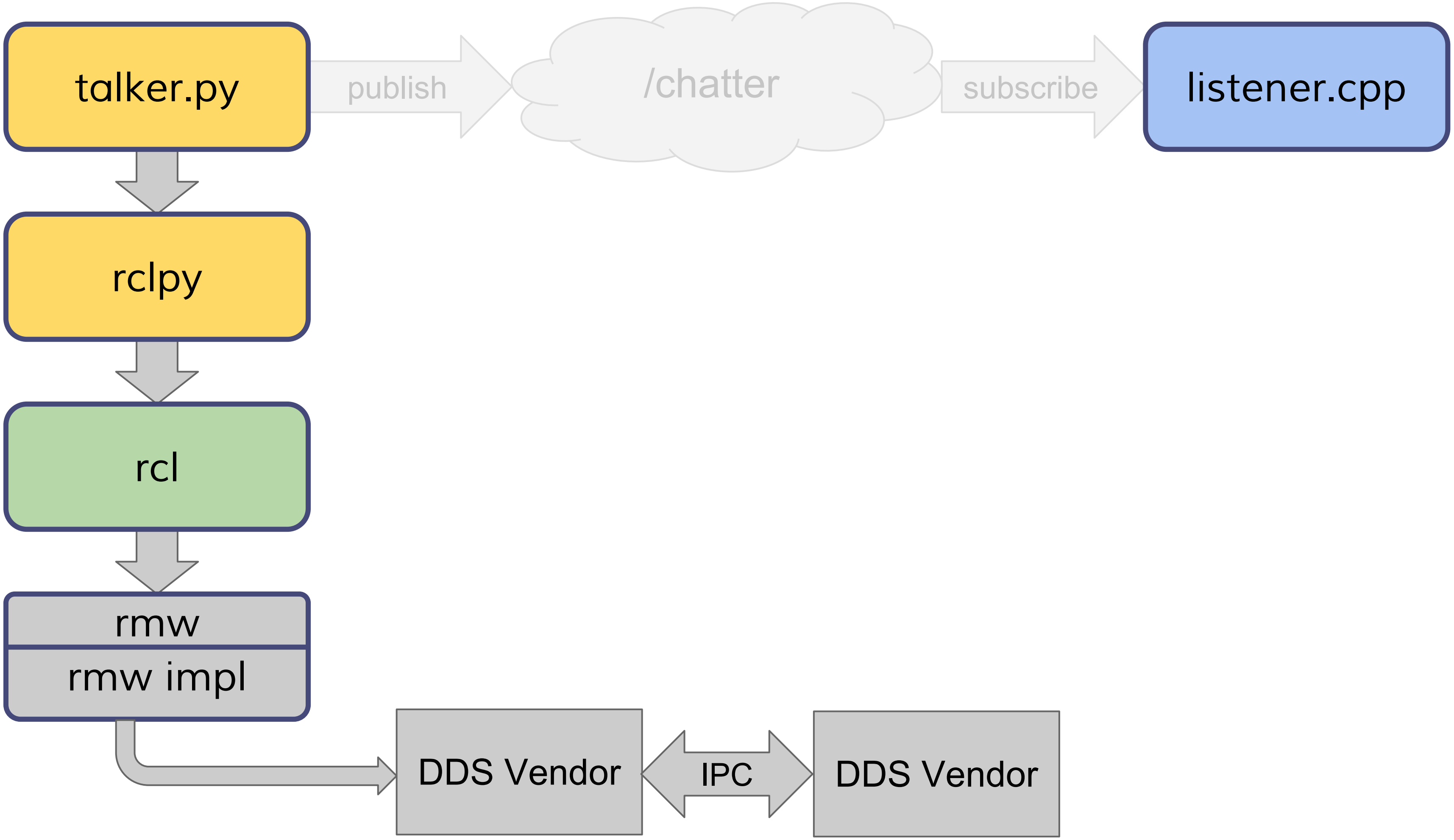
```
rcl_ret_t
rcl_publish(
  const rcl_publisher_t * publisher,
  const void * ros_message)
{
  // ...
  ret = rmw_publish(publisher->impl->rmw_handle, ros_message);
  if (ret != RMW_RET_OK) {
    // error handling
  }
  return RCL_RET_OK;
}
```

Tracing talker-listener

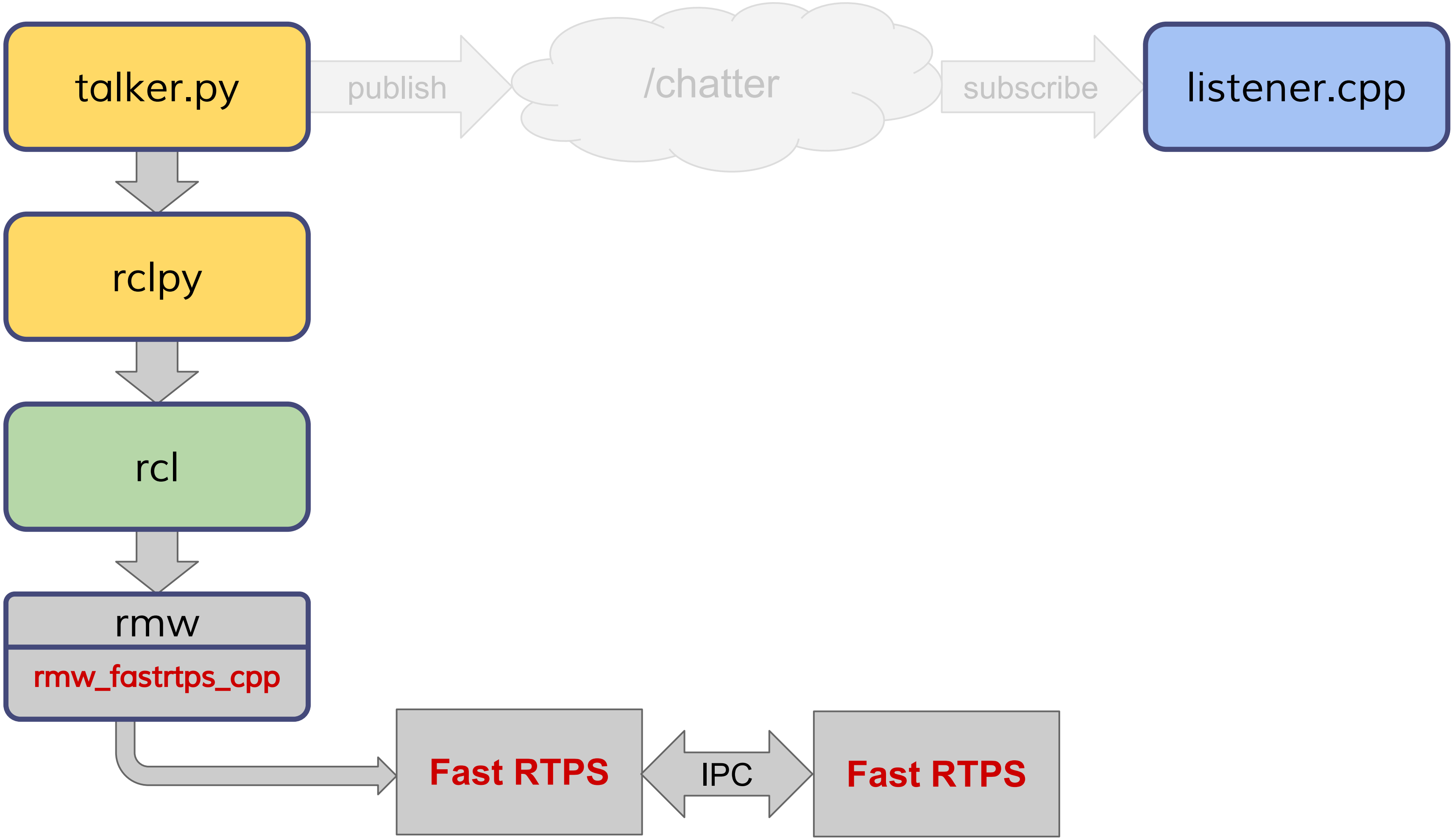


```
rmw_ret_t
rmw_publish(
    const rmw_publisher_t * publisher,
    const void * ros_message);
```

Tracing talker-listener



Tracing talker-listener



talker.py

publish

/chatter

subscribe

listener.cpp

rclpy

rcl

rmw

rmw_fastrtps_cpp

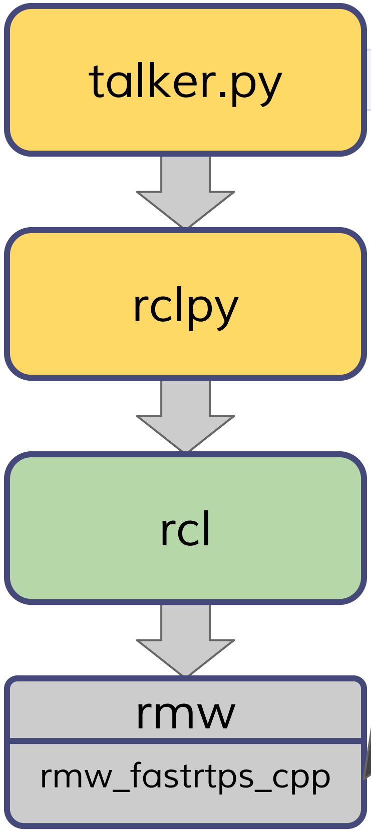
Fast RTPS

IPC

Fast RTPS

Open Source Robotics Foundation

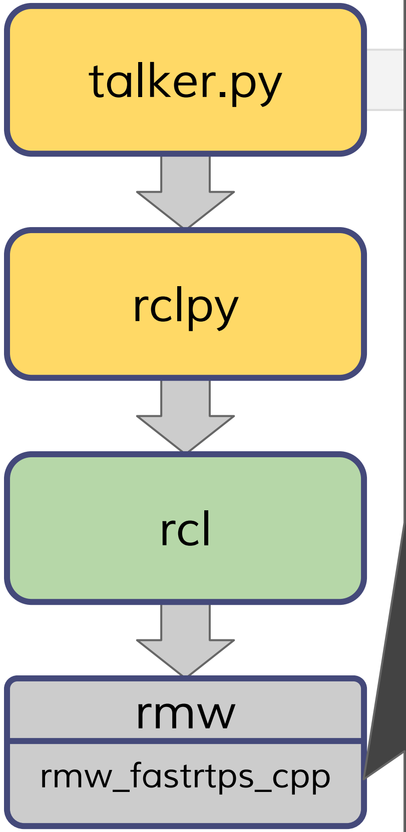
Tracing talker-listener

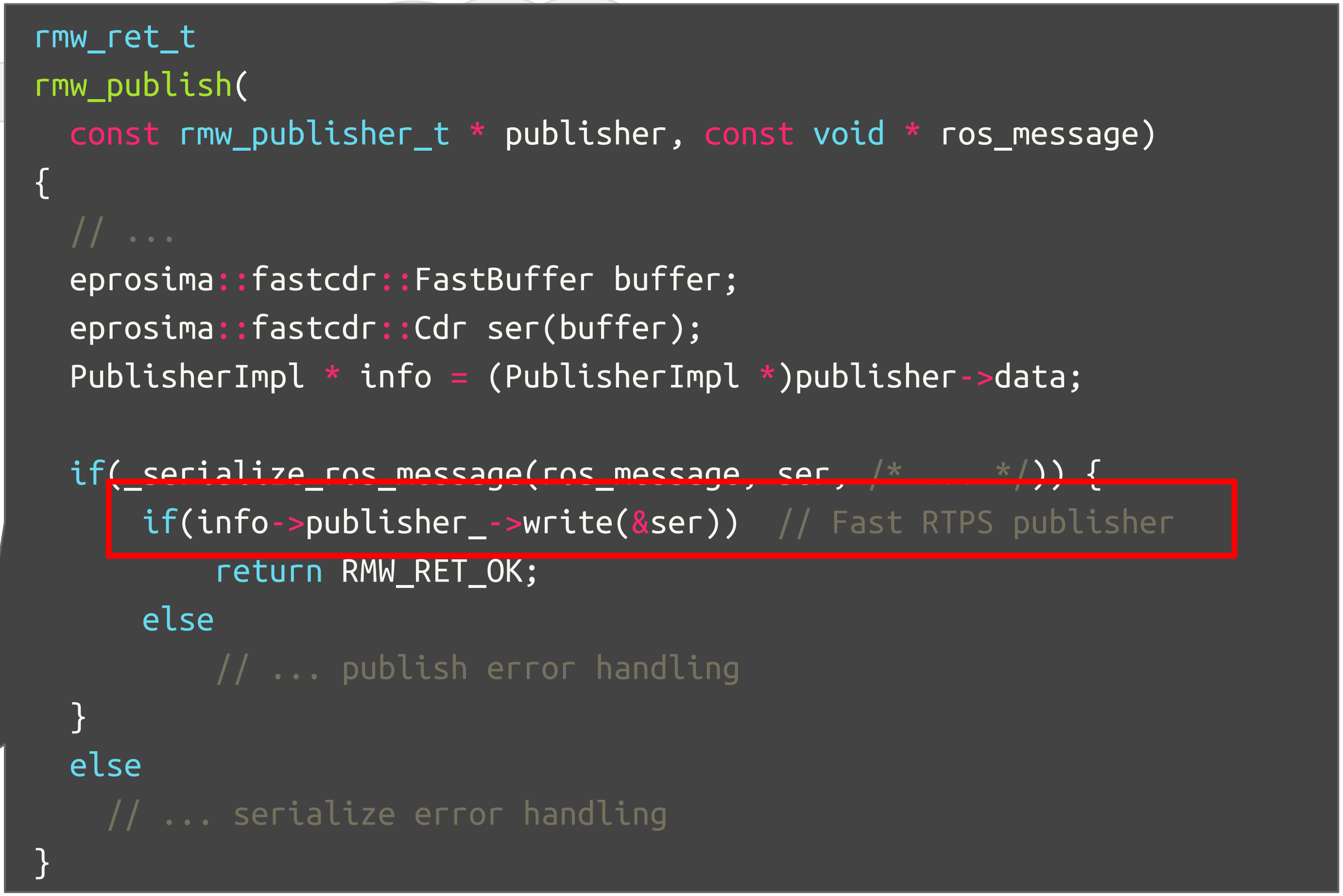


```
rmw_ret_t
rmw_publish(
  const rmw_publisher_t * publisher, const void * ros_message)
{
  // ...
  eprosima::fastcdr::FastBuffer buffer;
  eprosima::fastcdr::Cdr ser(buffer);
  PublisherImpl * info = (PublisherImpl *)publisher->data;

  if(_serialize_ros_message(ros_message, ser, /* ... */)) {
      if(info->publisher_->write(&ser))  // Fast RTPS publisher
          return RMW_RET_OK;
      else
          // ... publish error handling
  }
  else
    // ... serialize error handling
}
```

Open Source Robotics Foundation

Tracing talker-listener

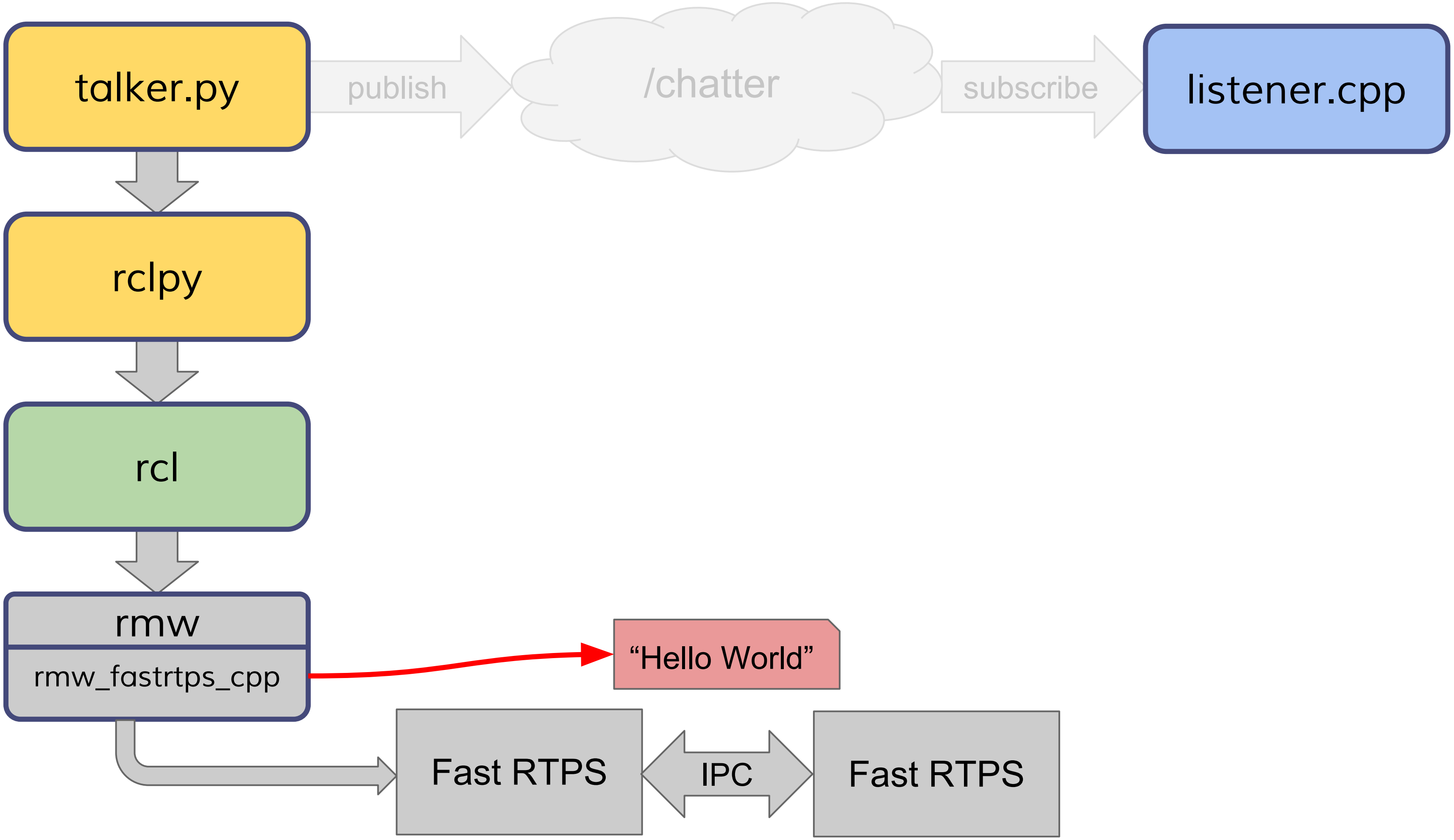




```
rmw_ret_t
rmw_publish(
  const rmw_publisher_t * publisher, const void * ros_message)
{
  // ...
  eprosima::fastcdr::FastBuffer buffer;
  eprosima::fastcdr::Cdr ser(buffer);
  PublisherImpl * info = (PublisherImpl *)publisher->data;

  if(_serialize_ros_message(ros_message, ser, /*     */)) {
      if(info->publisher_->write(&ser))  // Fast RTPS publisher
          return RMW_RET_OK;
      else
          // ... publish error handling
  }
  else
    // ... serialize error handling
}
```

Open Source Robotics Foundation

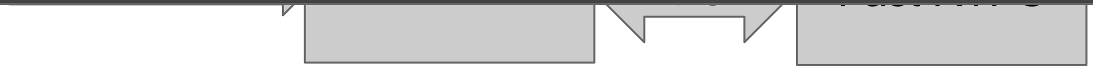
Tracing talker-listener



Tracing talker-listener

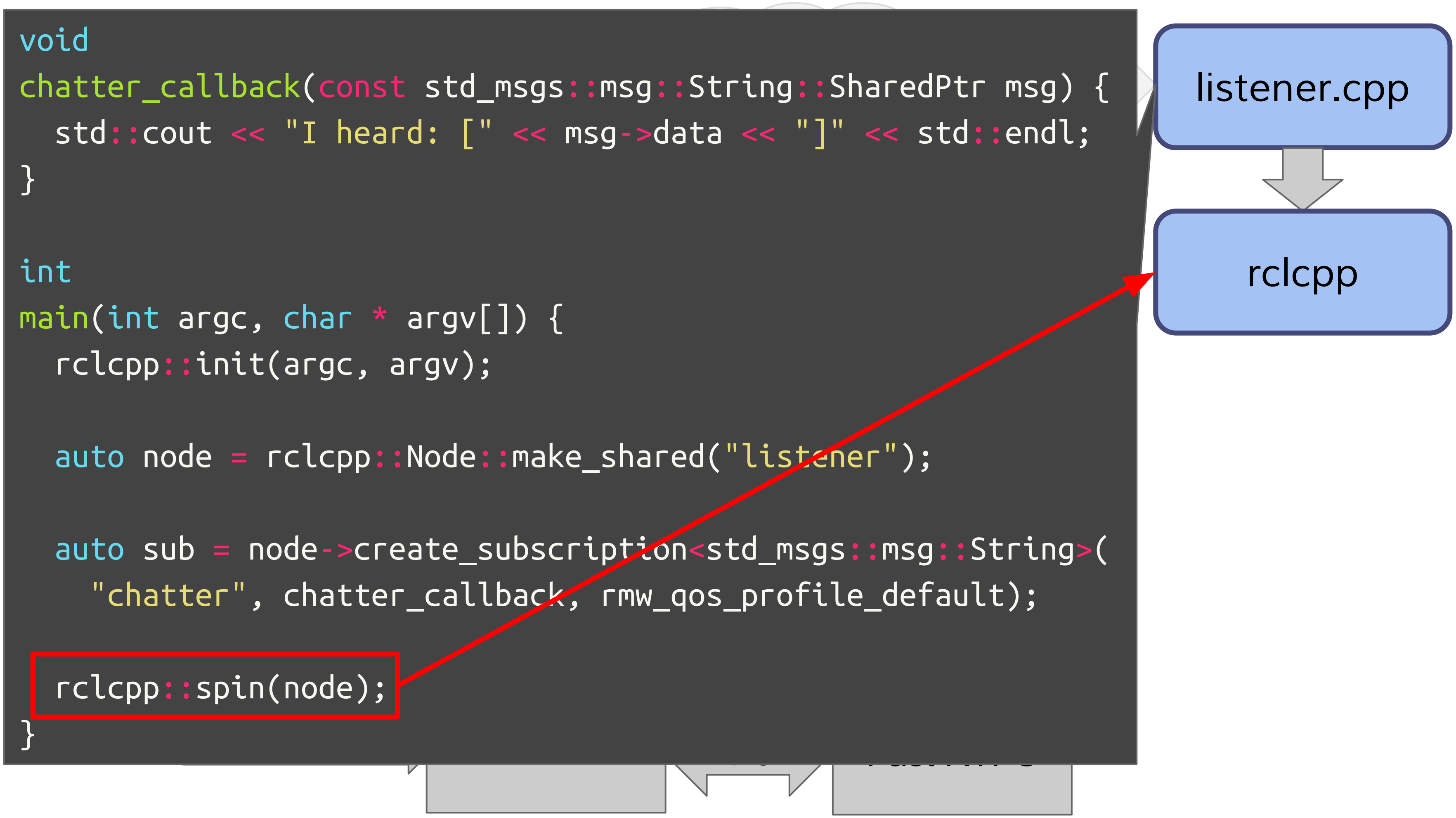
```
void
chatter_callback(const std_msgs::msg::String::SharedPtr msg) {
  std::cout << "I heard: [" << msg->data << "]" << std::endl;
}

int
main(int argc, char * argv[]) {
  rclcpp::init(argc, argv);

  auto node = rclcpp::Node::make_shared("listener");

  auto sub = node->create_subscription<std_msgs::msg::String>(
    "chatter", chatter_callback, rmw_qos_profile_default);

  rclcpp::spin(node);
}
```

listener.cpp



Open Source Robotics Foundation

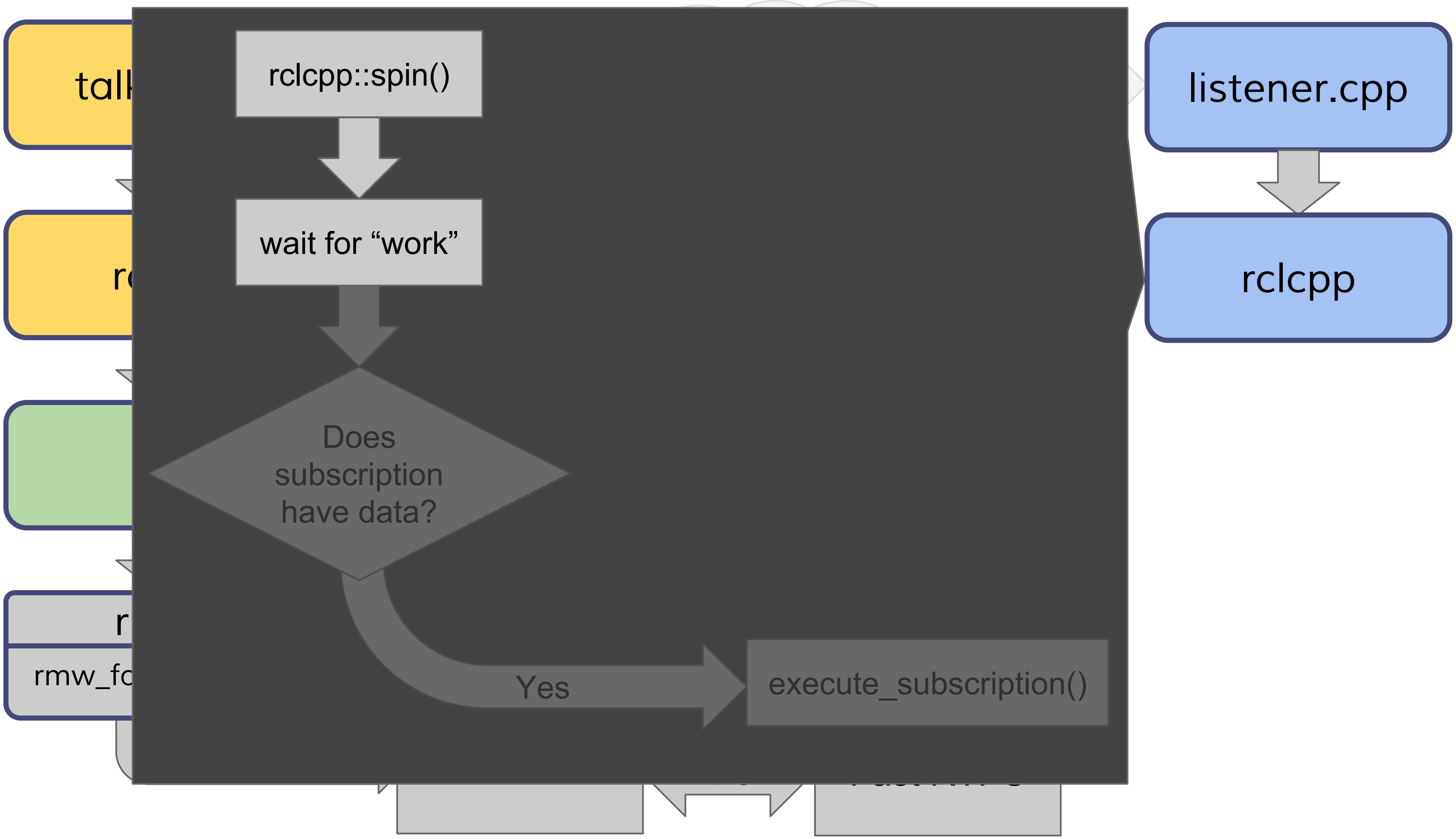
Tracing talker-listener



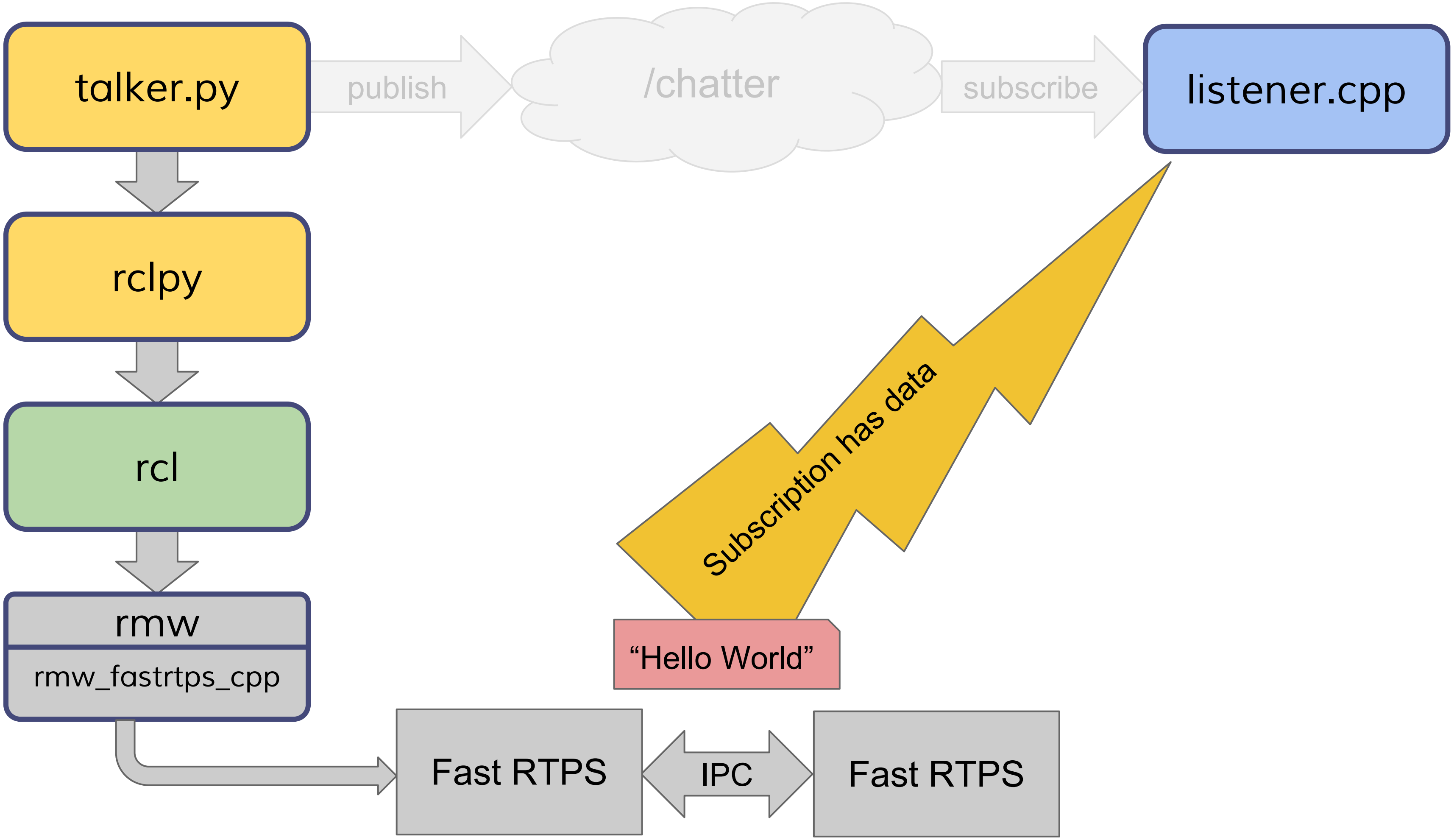
```
void
chatter_callback(const std_msgs::msg::String::SharedPtr msg) {
  std::cout << "I heard: [" << msg->data << "]" << std::endl;
}

int
main(int argc, char * argv[]) {
  rclcpp::init(argc, argv);

  auto node = rclcpp::Node::make_shared("listener");

  auto sub = node->create_subscription<std_msgs::msg::String>(
    "chatter", chatter_callback, rmw_qos_profile_default);

  rclcpp::spin(node);
}
```

Open Source Robotics Foundation

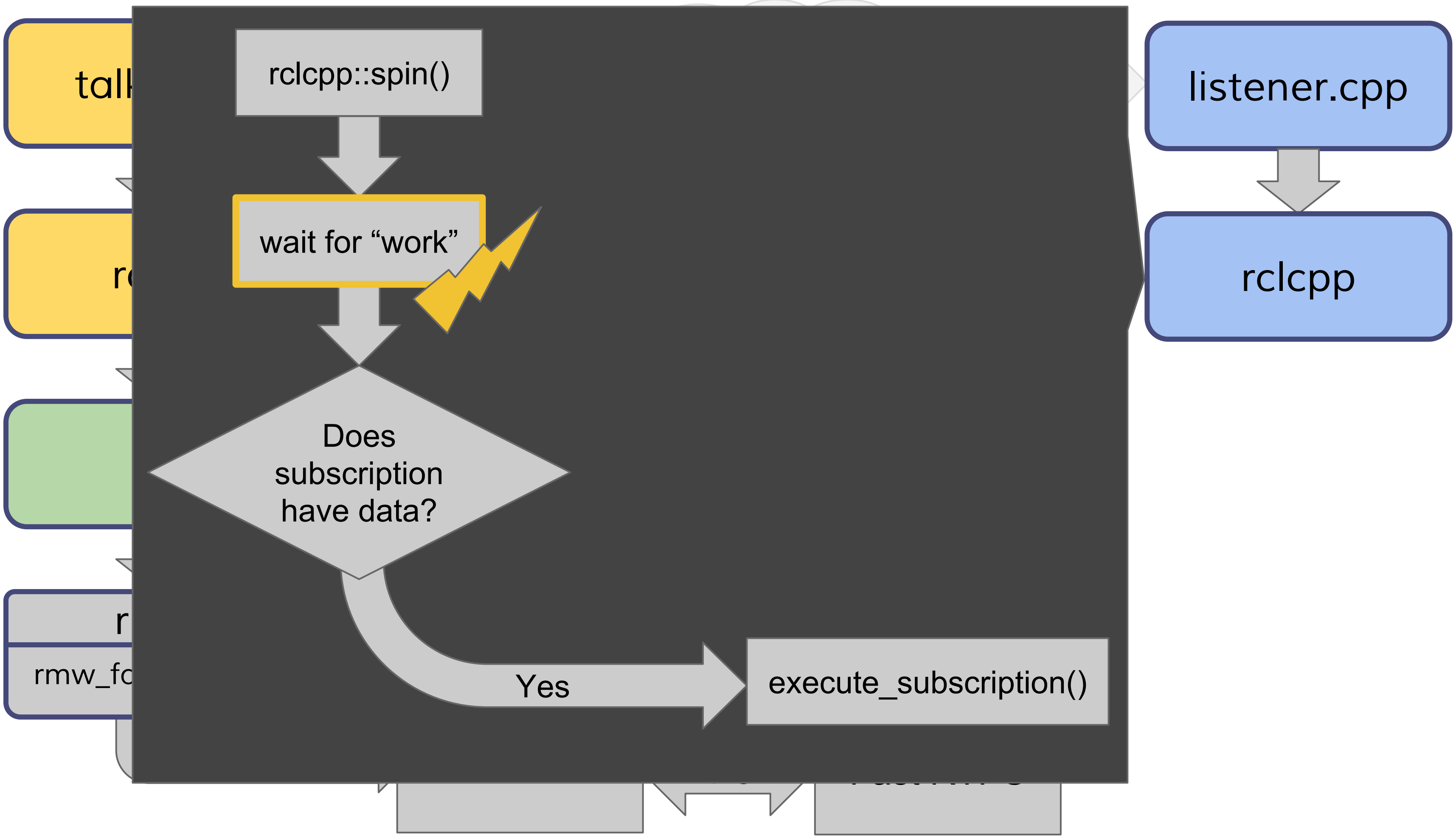
Tracing talker-listener



Tracing talker-listener

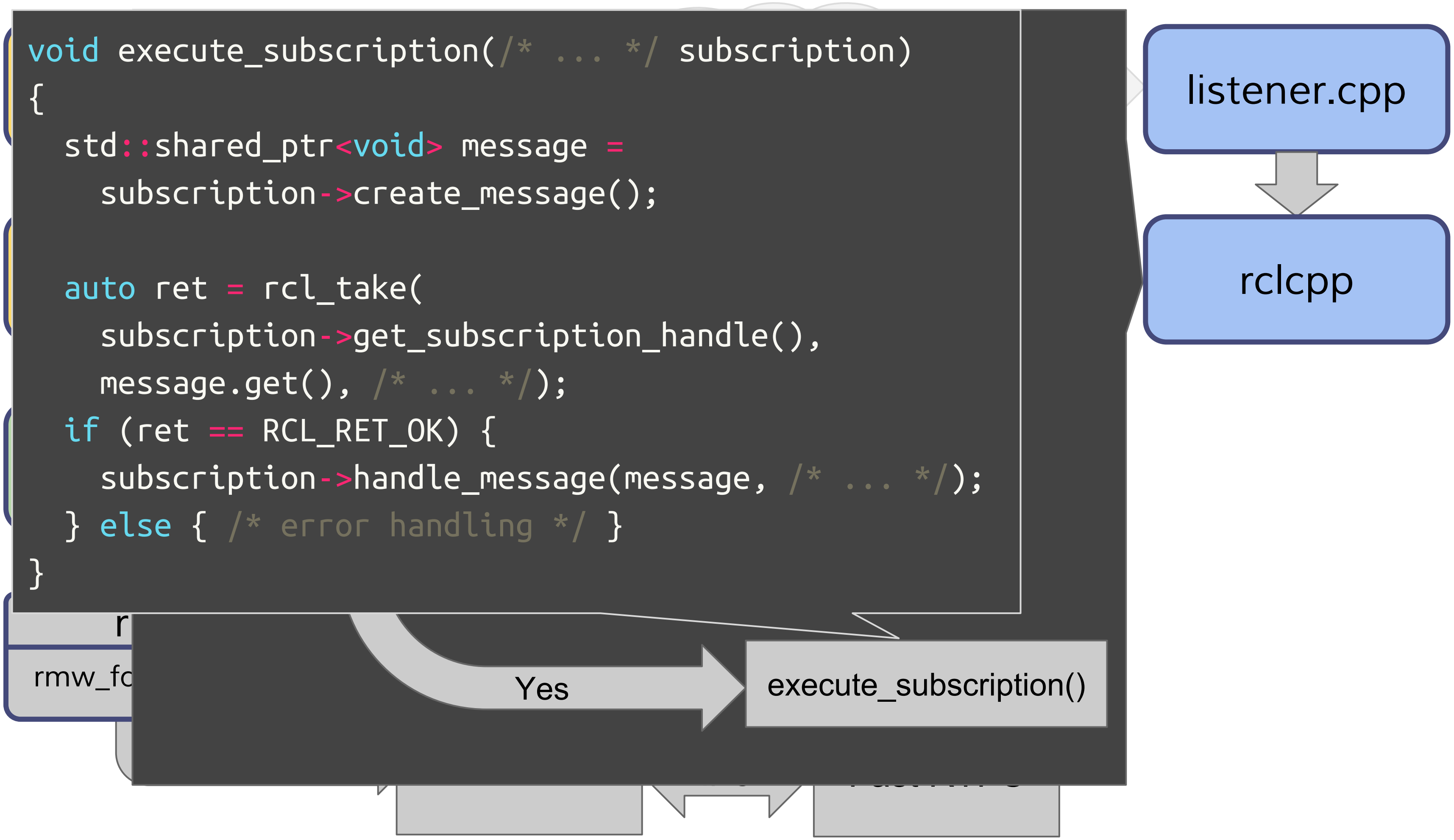


Tracing talker-listener



Open Source Robotics Foundation

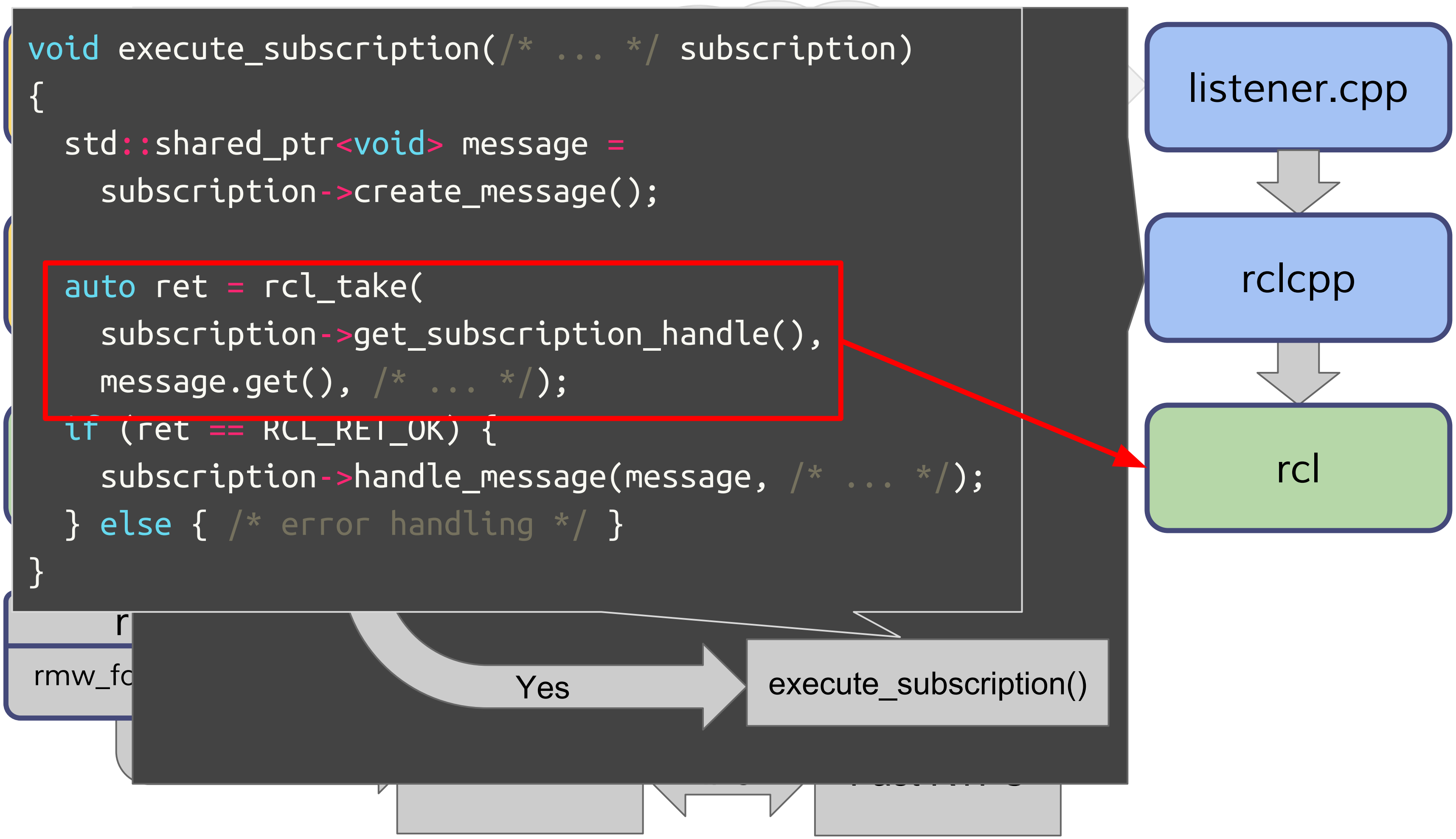
Tracing talker-listener



```
void execute_subscription(/* ... */ subscription)
{
  std::shared_ptr<void> message =
    subscription->create_message();

  auto ret = rcl_take(
    subscription->get_subscription_handle(),
    message.get(), /* ... */);
  if (ret == RCL_RET_OK) {
    subscription->handle_message(message, /* ... */);
  } else { /* error handling */ }
}
```

Open Source Robotics Foundation

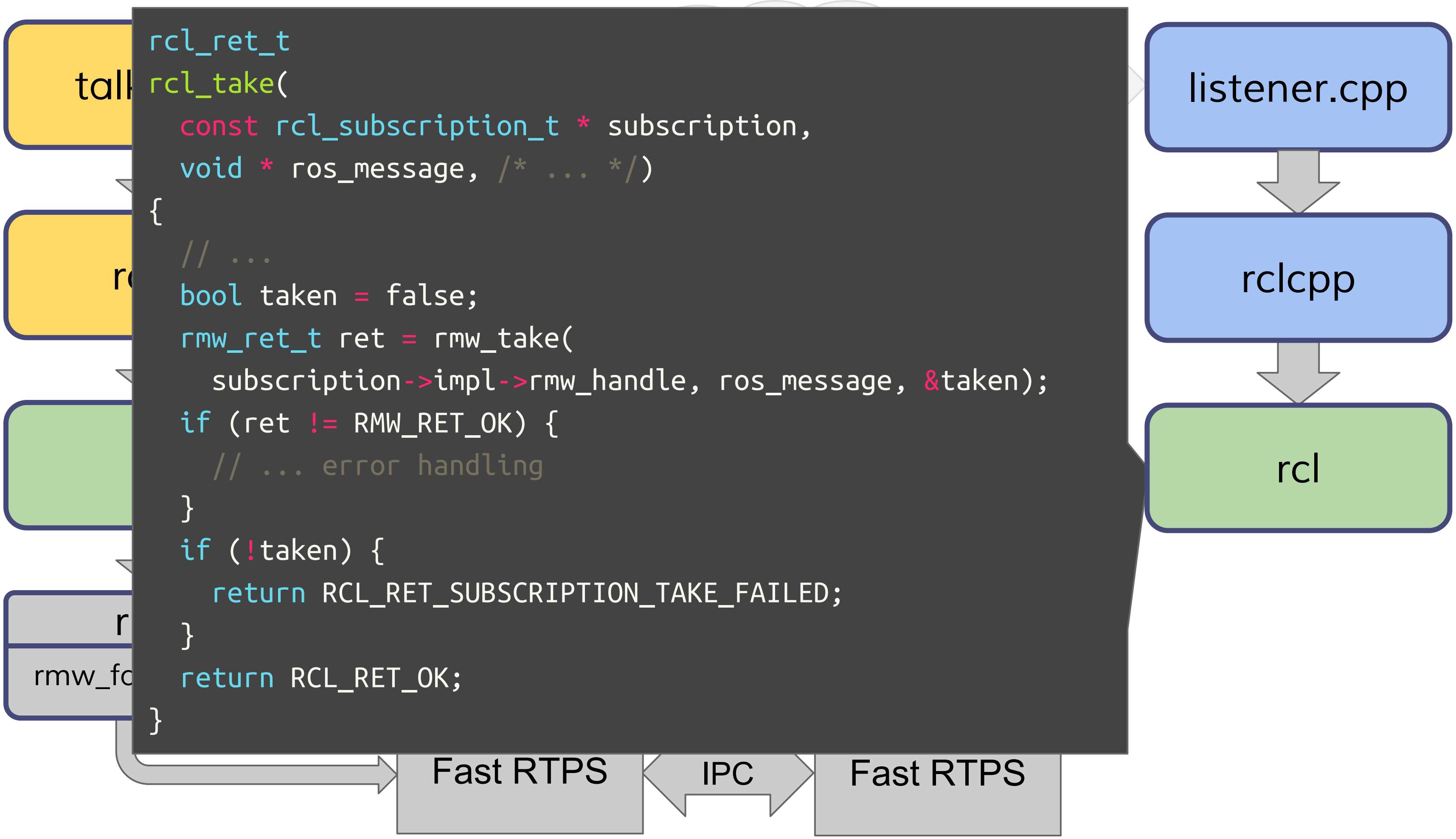
Tracing talker-listener



```cpp
void execute_subscription(/* ... */ subscription)
{
  std::shared_ptr<void> message =
    subscription->create_message();

  auto ret = rcl_take(
    subscription->get_subscription_handle(),
    message.get(), /* ... */);
  if (ret == RCL_RET_OK) {
    subscription->handle_message(message, /* ... */);
  } else { /* error handling */ }
}
```

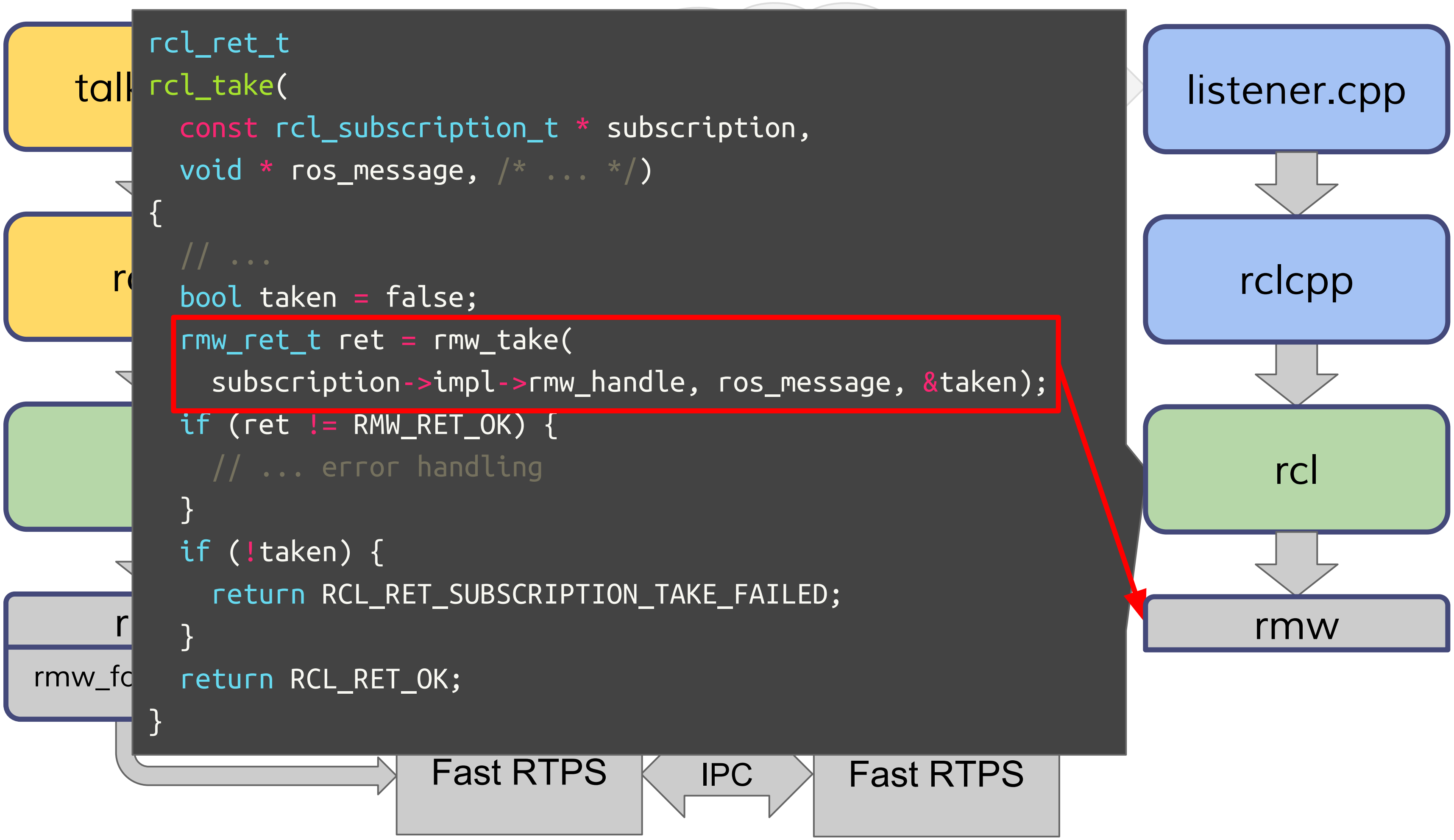
Open Source Robotics Foundation

Tracing talker-listener



```
rcl_ret_t
rcl_take(
  const rcl_subscription_t * subscription,
  void * ros_message, /* ... */)
{
  // ...
  bool taken = false;
  rmw_ret_t ret = rmw_take(
    subscription->impl->rmw_handle, ros_message, &taken);
  if (ret != RMW_RET_OK) {
    // ... error handling
  }
  if (!taken) {
    return RCL_RET_SUBSCRIPTION_TAKE_FAILED;
  }
  return RCL_RET_OK;
}
```

Tracing talker-listener



```
rcl_ret_t
rcl_take(
  const rcl_subscription_t * subscription,
  void * ros_message, /* ... */)
{
  // ...
  bool taken = false;
  rmw_ret_t ret = rmw_take(
    subscription->impl->rmw_handle, ros_message, &taken);
  if (ret != RMW_RET_OK) {
    // ... error handling
  }
  if (!taken) {
    return RCL_RET_SUBSCRIPTION_TAKE_FAILED;
  }
  return RCL_RET_OK;
}
```

talk

r(

r

rmw_fc

listener.cpp

rclcpp

rcl

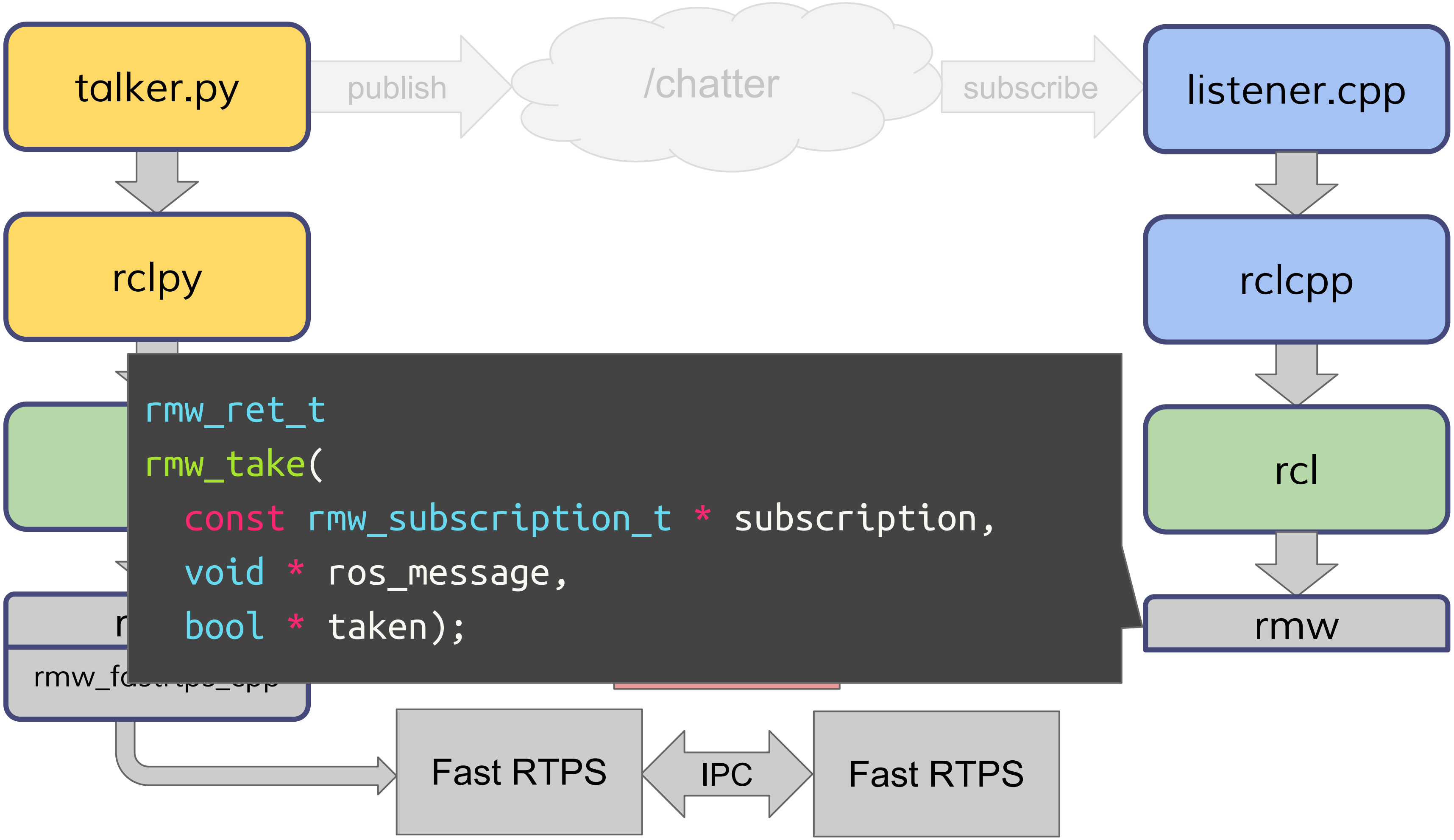
rmw

Fast RTPS

IPC

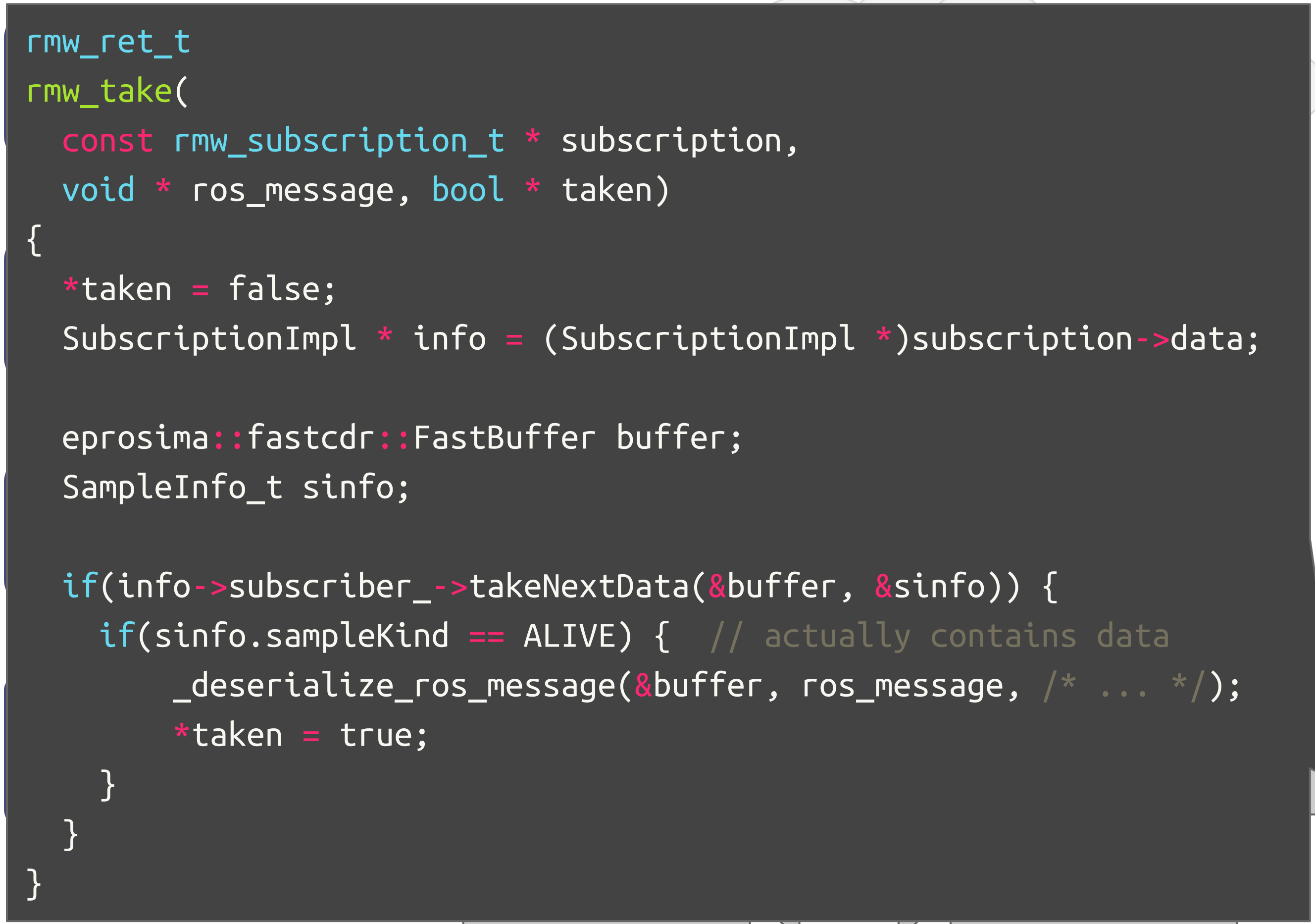
Fast RTPS

Tracing talker-listener

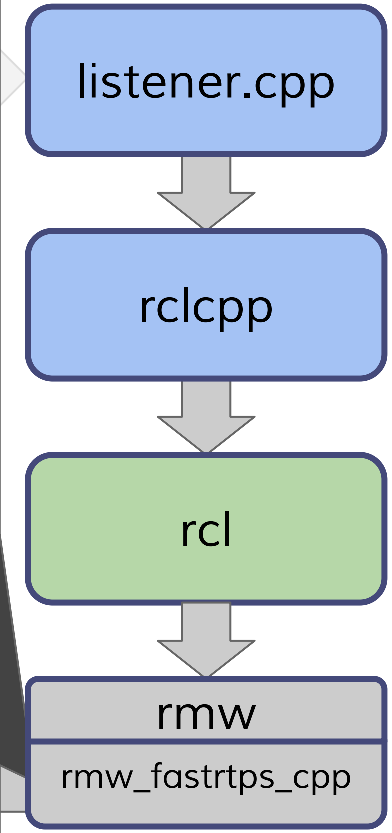


```
rmw_ret_t
rmw_take(
  const rmw_subscription_t * subscription,
  void * ros_message,
  bool * taken);
```

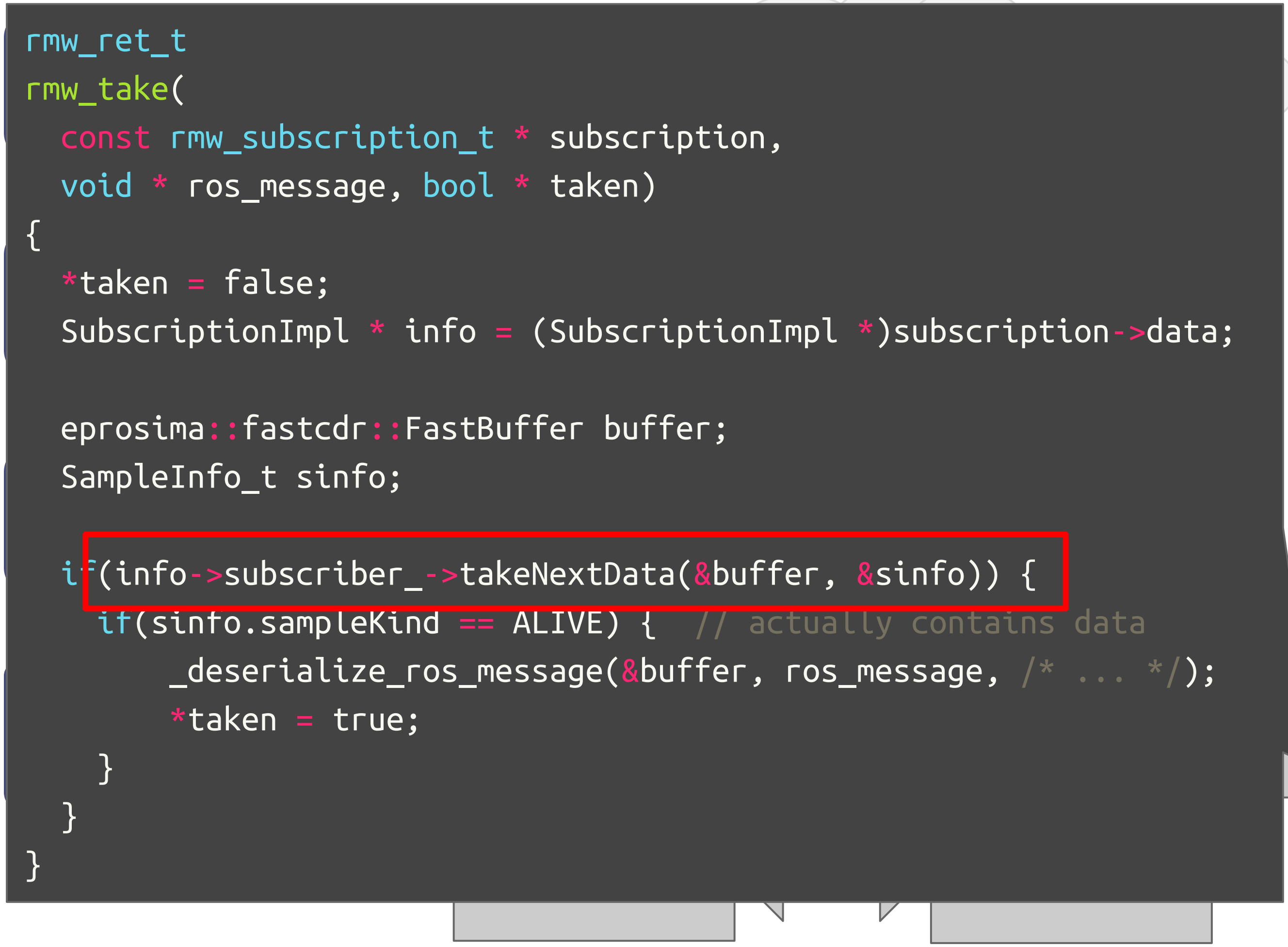
Tracing talker-listener



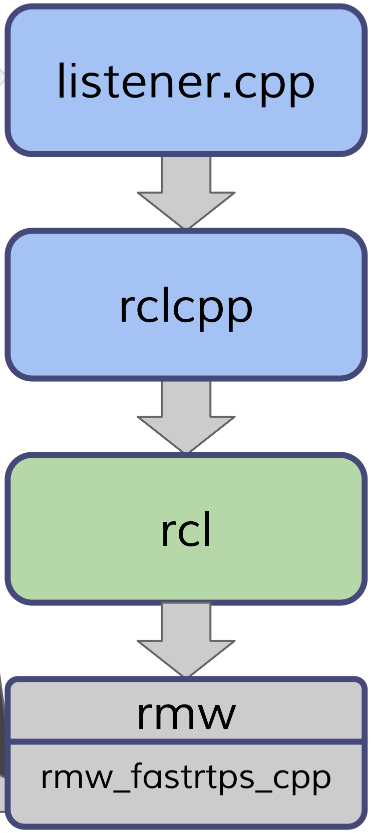
```
rmw_ret_t
rmw_take(
  const rmw_subscription_t * subscription,
  void * ros_message, bool * taken)
{
  *taken = false;
  SubscriptionImpl * info = (SubscriptionImpl *)subscription->data;

  eprosima::fastcdr::FastBuffer buffer;
  SampleInfo_t sinfo;

  if(info->subscriber_->takeNextData(&buffer, &sinfo)) {
    if(sinfo.sampleKind == ALIVE) {  // actually contains data
        _deserialize_ros_message(&buffer, ros_message, /* ... */);
        *taken = true;
    }
  }
}
```



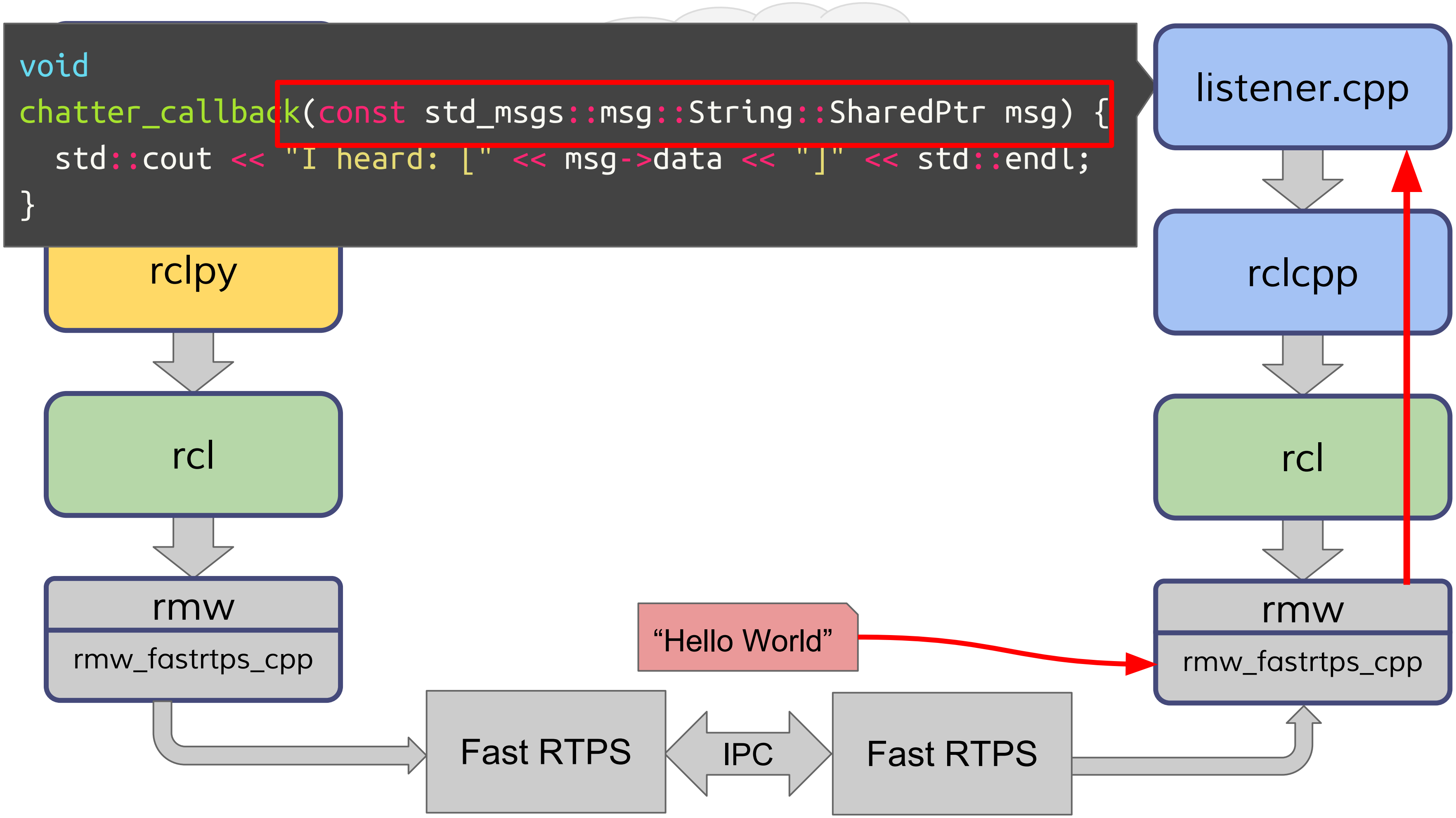
Tracing talker-listener



```
rmw_ret_t
rmw_take(
  const rmw_subscription_t * subscription,
  void * ros_message, bool * taken)
{
  *taken = false;
  SubscriptionImpl * info = (SubscriptionImpl *)subscription->data;

  eprosima::fastcdr::FastBuffer buffer;
  SampleInfo_t sinfo;

  if(info->subscriber_->takeNextData(&buffer, &sinfo)) {
    if(sinfo.sampleKind == ALIVE) {  // actually contains data
        _deserialize_ros_message(&buffer, ros_message, /* ... */);
        *taken = true;
    }
  }
}
```



Tracing talker-listener

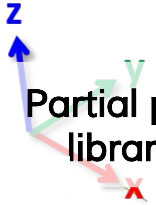


```
void
chatter_callback(const std_msgs::msg::String::SharedPtr msg) {
  std::cout << "I heard: [" << msg->data << "]" << std::endl;
}
```

Changes since ROSCon 2015

Windows feature parity (alpha 2)

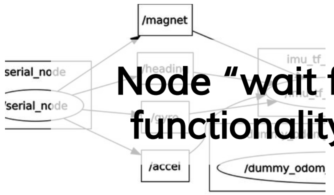
Fast RTPS supported as middleware (alpha 3)



Partial port of core tf2 libraries (alpha 3)

Python client library (alpha 4)

32- and 64-bit ARM experimentally supported platforms (alpha 5)



Node "wait for service" functionality (alpha 6)

Turtlebot demo using ported code from ROS 1 (alpha 7)

ROS Client Library implementation (rcl) (from alpha 3, services alpha 5)

Support for C messages (as opposed to C++) (alphas 4, 5, 7)

Refactored C++ client library to use rcl (alpha 6)

ROS graph events (alpha 6)

Improved support for large messages (images) with Connext and Fast RTPS (alpha 6, alpha 7)

Porting of Turtlebot to ROS 2

- Minimum viable demo (https://github.com/ros2/turtlebot2_demo)
    - Kobuki driver
    - Astra driver
    - Joystick driver
    - Follower node

https://orbbec3d.com/

iclebo

Kobuki

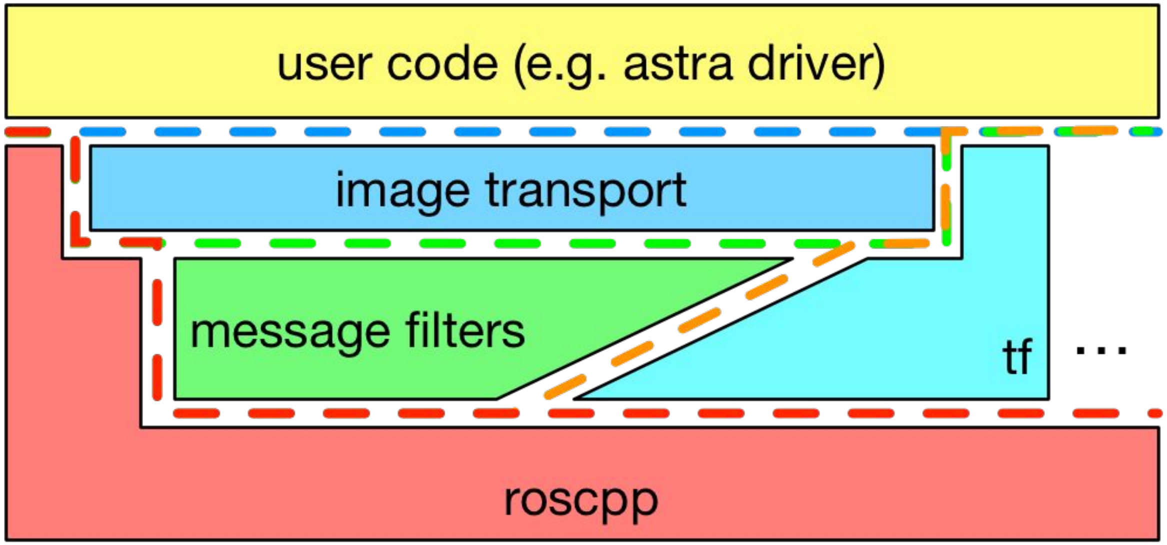
http://kobuki.yujinrobot.com/

Porting of Turtlebot to ROS 2

- Kobuki driver
  - Used existing non-ROS dependencies
  - Replaced ROS 1 wrapper with ROS 2 wrapper
- Astra driver
  - Forked and ported existing ROS 1 driver to ROS 2
- Joystick driver
  - Wrote a simple joystick program from scratch (no porting)
- Follower node
  - Forked and ported existing ROS 1 node
- ROS 1 ⇔ ROS 2 bridge for visualization

Porting Experiments

- ROS 1 “shim” (https://github.com/codebot/ros1_shim)
  - Some things (like the astra driver) needed some deep features (e.g. custom serialization)
  - Hard to find the right strata in the interfaces to shim



Porting Experiments

- *catment* (https://github.com/ros2/ros2/wiki/catment)
  - Find ways to modify each to make them more similar
    - In order to minimize conversion effort
  - Mixing catkin (ROS 1) and ament (ROS 2)
    - To avoid converting unless necessary
  - Non-homogeneous workspace
    - Building catkin and ament packages at the same time
  - Ideal: one build tool for both
    - ament vs catkin not unlike catkin vs plain cmake

Porting Experiments

- *catment* continued...
    - Conceptual details to work out:
        - setup.*sh files in root of workspace
            - Currently required by catkin
            - Optional for ament
        - devel-space
            - ament uses “symlink install” instead
        - Avoiding confusion in documentation
    - Make catkin more like ament? (and vice versa?)

Roadmap

- Beta 1 - End of the Year
  - Composition
    - may use pluginlib and class_loader from ROS 1 for C++
  - QoS benchmarks
    - for example: unreliable comms, illustrated by wifi out-and-back
  - Design documents
  - Tutorials and examples
  - "rostopic list", "rostopic echo", and friends
  - Bridging services to/from ROS1 (in addition to topics)
- Nice to have by Beta 1:
  - Console logging
    - think “rosconsole”
  - Orchestration
    - think “roslaunch + verification & dynamic behavior”

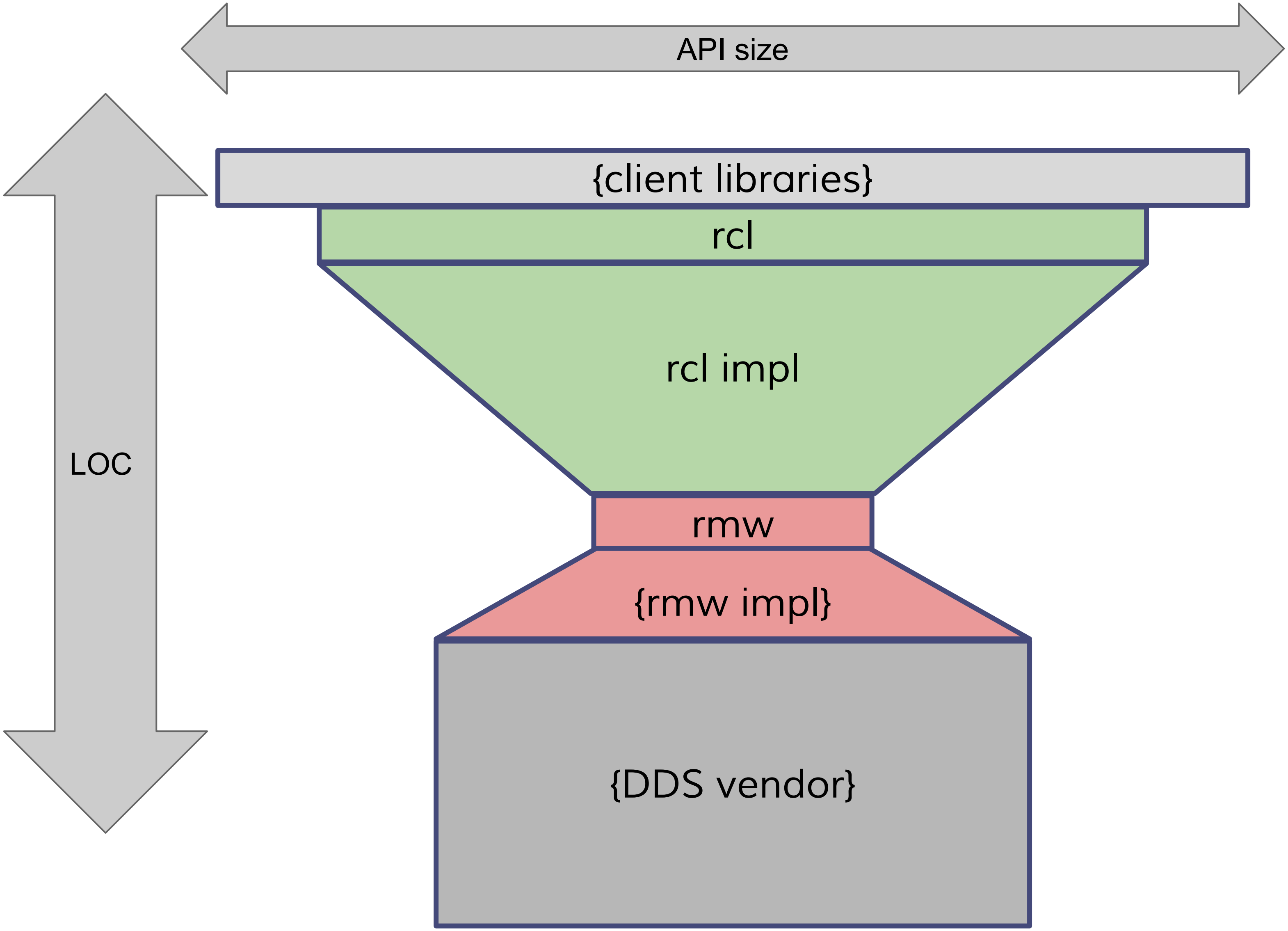
Pointers

- ROS 2 wiki: https://github.com/ros2/ros2/wiki
  - Installation instructions
  - Tutorials
  - How to contribute
  - Current status
  - Roadmap
- Developer docs (work in progress):
  - https://github.com/ros2/ros_core_documentation/blob/master/source/developer_overview.rst
  - Architecture overview
  - Links to API docs
- Design documents: http://design.ros2.org/
  - Articles about various subjects
  - On going discussions on the issue tracker: https://github.com/ros2/design

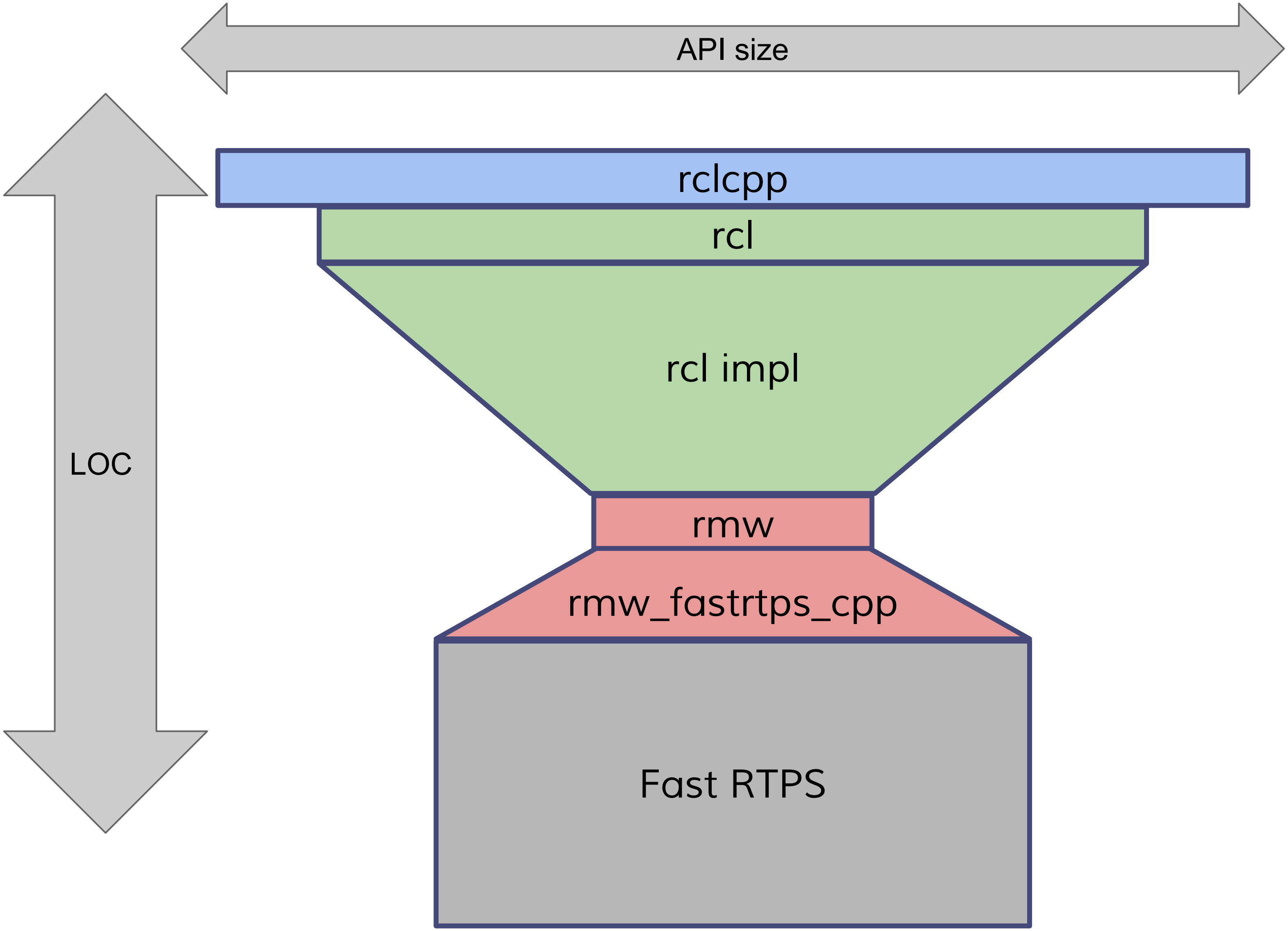
Questions

https://goo.gl/oCHR7H

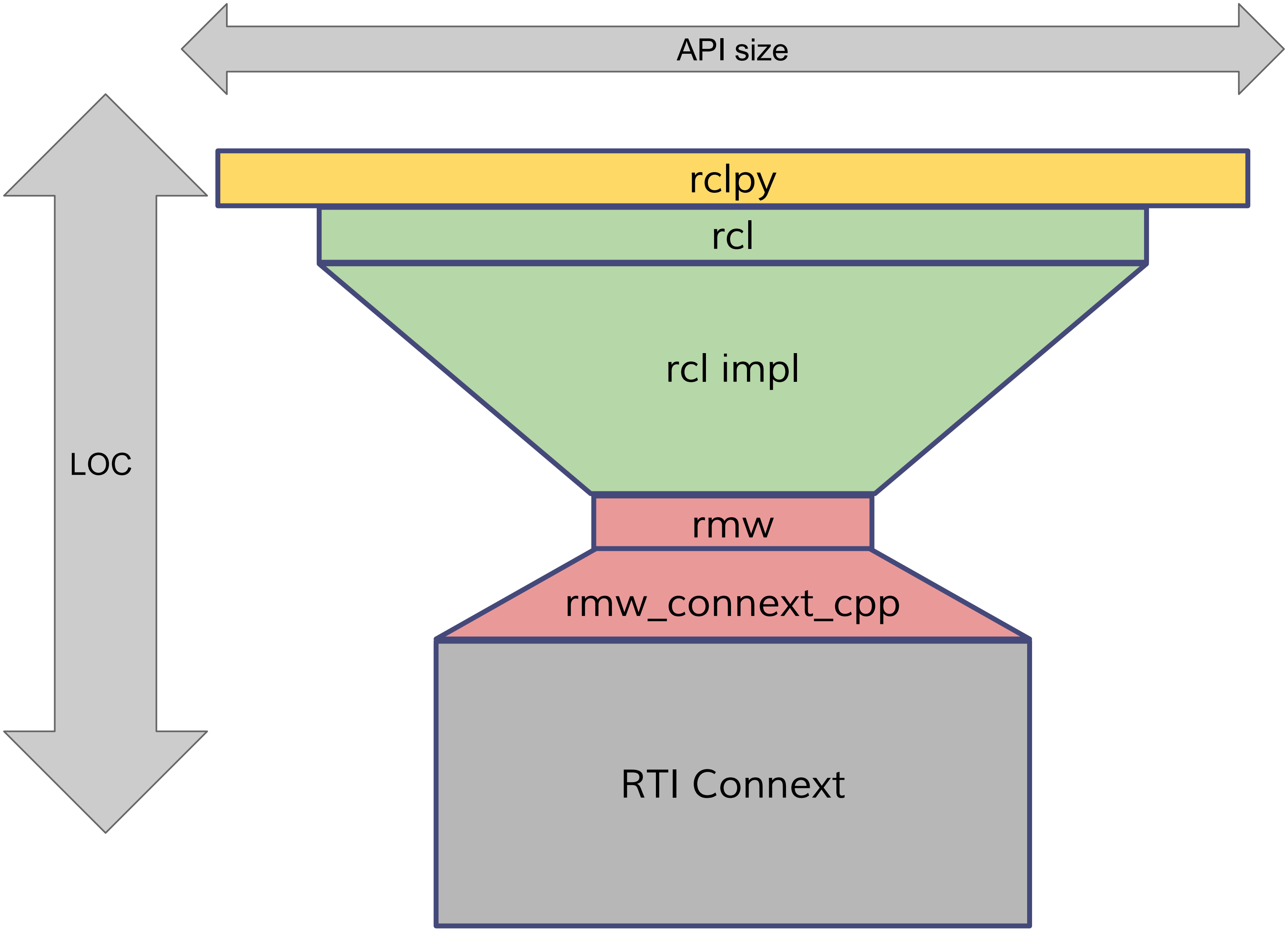
“Hour Glass” Pattern



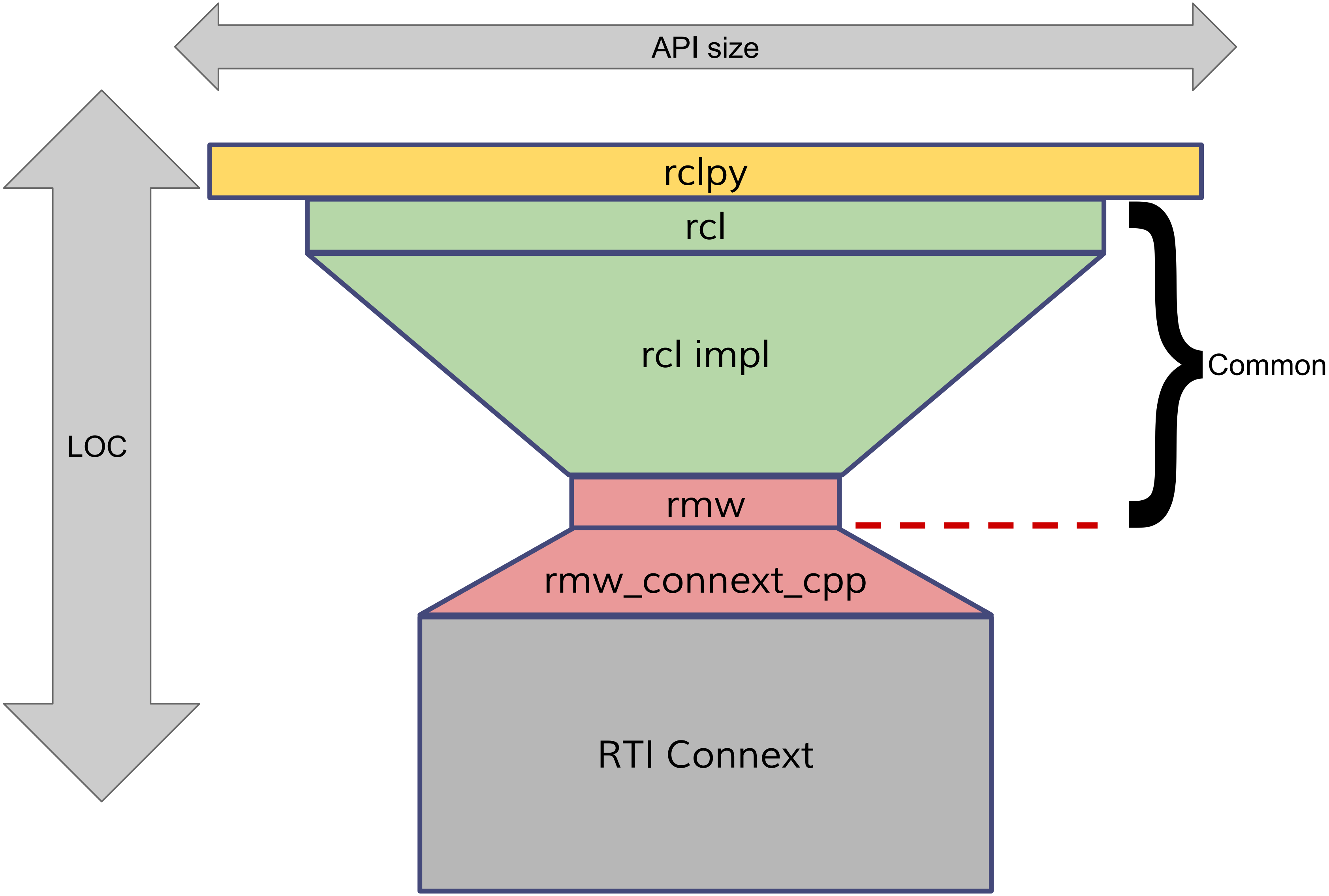
“Hour Glass” Pattern - C++ with Fast RTPS



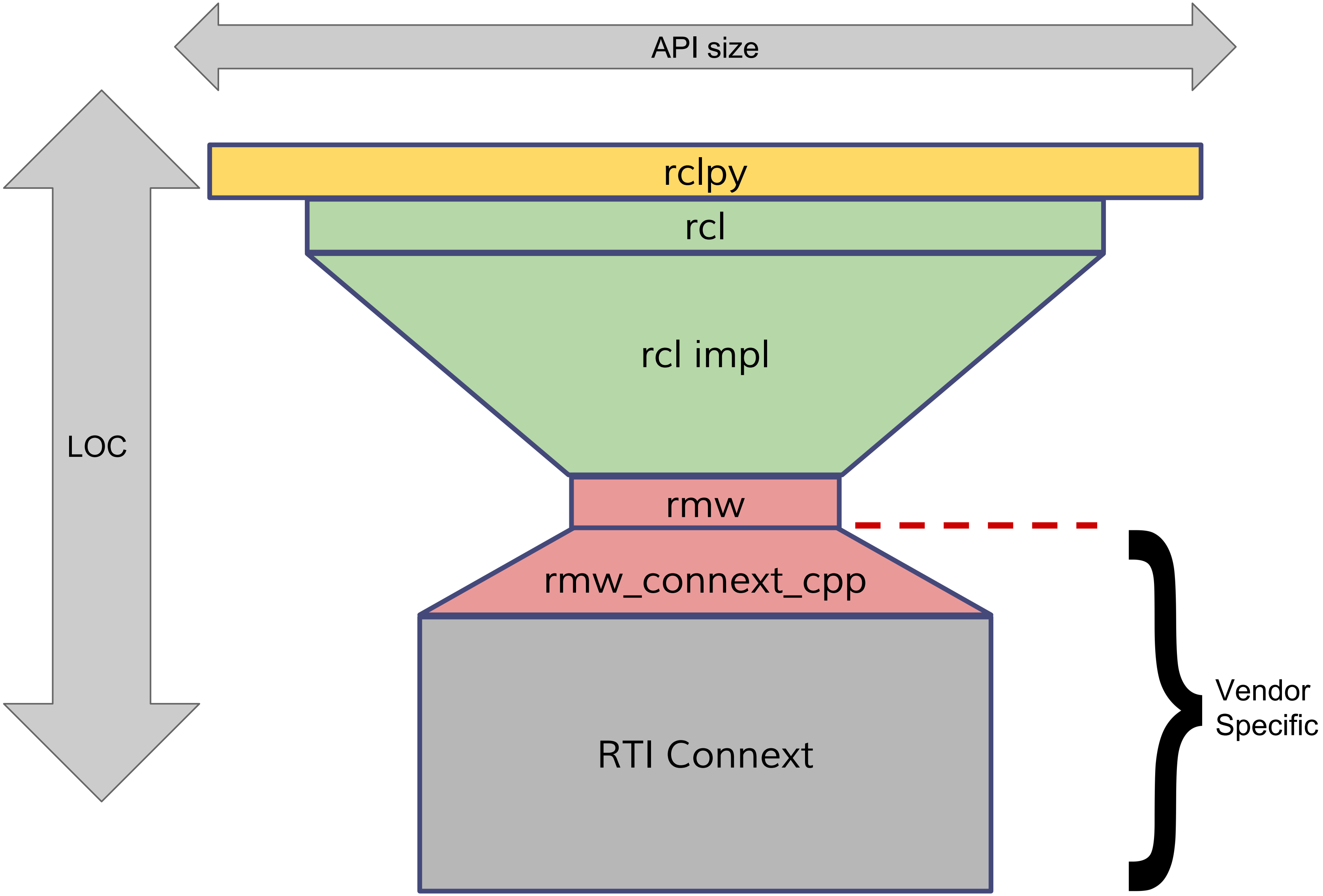
“Hour Glass” Pattern - Python with RTI Connext



“Hour Glass” Pattern - Python with RTI Connext



“Hour Glass” Pattern - Python with RTI Connext


API size
LOC
rclpy
rcl
rcl impl
rmw
rmw_connext_cpp
RTI Connext
Vendor
Specific