How I Learned to Stop Reinventing and Love the Wheels

How Robotics Research Keeps...
Re-Inventing the Wheel
First, someone publishes...
...a paper with a proof-of-concept robot.
This prompts another lab to try to build on this result...
...but they can't get any details on the software used to make it work...
BARF!
...and all the code used by previous lab members is a mess.
So, a grandiose plan is formed to write a new software API...
...and countless sleepless nights are spent writing code from scratch.
But inevitably, time runs out...
...and they write code that barely works but lets them publish...

or having FUN with (home/hackerspace) robotics

Overview

- On Reinventing Wheels in Robotics
- ROS Technical Overview and Concepts
- Hardware
- Sensor example: Cameras
- Robots
- Simulation
- Tools and Introspection
- Much more to be discovered

Overview

- On Reinventing Wheels in Robotics
- ROS Technical Overview and Concepts
- Hardware

} Text,
Bullet pnts
and some
Code

- Sensor example: Cameras
- Robots
- Simulation
- Tools and Introspection

} Images and
Videos

- Much more to be discovered

} Pointers and Links

On Reinventing Wheels in Robotics

- Mechanics
- Electronics
- Software

On Reinventing Wheels in Robotics

- Mechanics
- Electronics
- Software

On Reinventing Wheels in Robotics

- Mechanics
- Electronics
- Software
  - Drivers
  - Core Functionality
  - Debugging, Introspection and User Interfaces
  - Algorithms
  - Parallelization and Distribution
  - Deployment and Orchestration
  - Applications

On Reinventing Wheels in Robotics

- Mechanics
- Electronics
- Software
  - Drivers
  - Core Functionality
  - Debugging, Introspection and User Interfaces
  - Algorithms
  - Parallelization and Distribution
  - Deployment and Orchestration
  - Applications ← What one wants to work on

On Reinventing Wheels in Robotics

- Mechanics
- Electronics
- Software
  - Drivers
  - Core Functionality
  - Debugging, Introspection and User Interfaces
  - Algorithms
  - Parallelization and Distribution
  - Deployment and Orchestration
  - Applications ← What one wants to work on

What one ends up working on

What is the Robot Operating System (ROS)?

- Communication Middleware + Tools
- Basic Robotics Software
- Packages with Build System
- Large Ecosystem

ROS
=
Plumbing
+
Tools
+
Capabilities
+
Ecosystem

Communication Middleware + Tools:
roscore

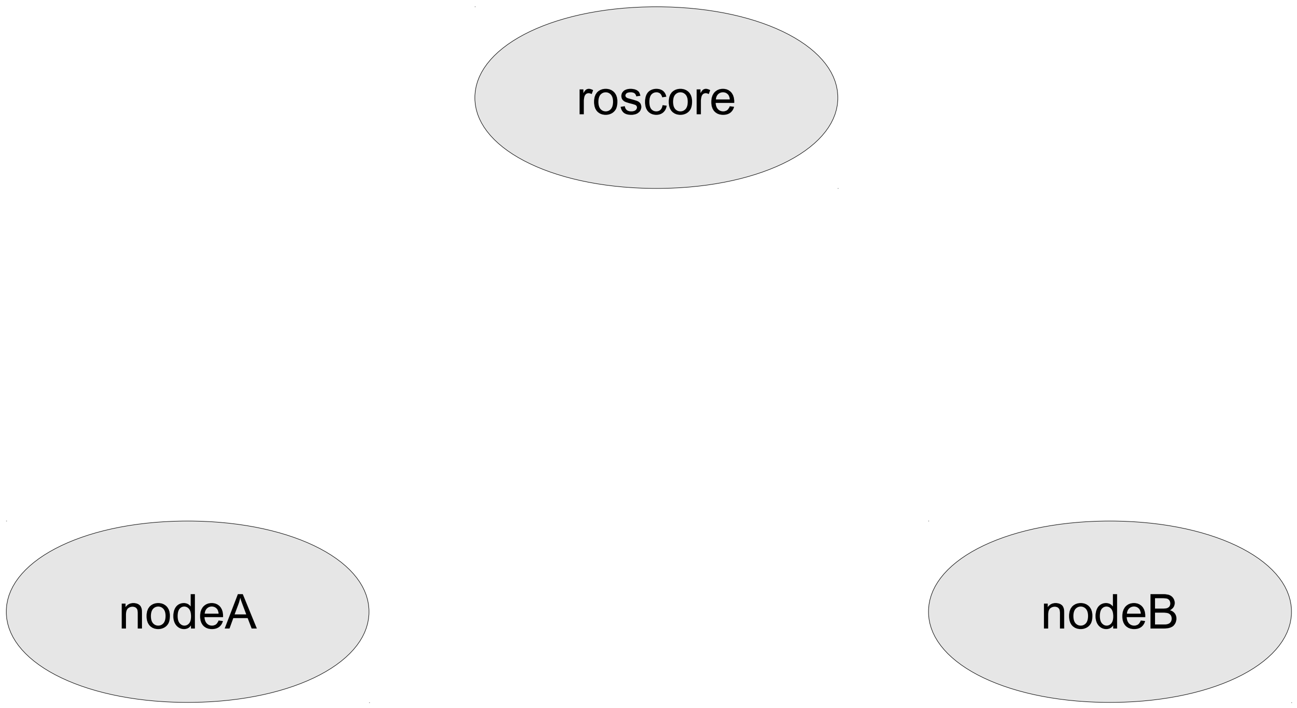
- Well-known entry point: ROS_MASTER_URI
- Registry for Nodes
- Parameter Server

Communication Middleware + Tools: nodes

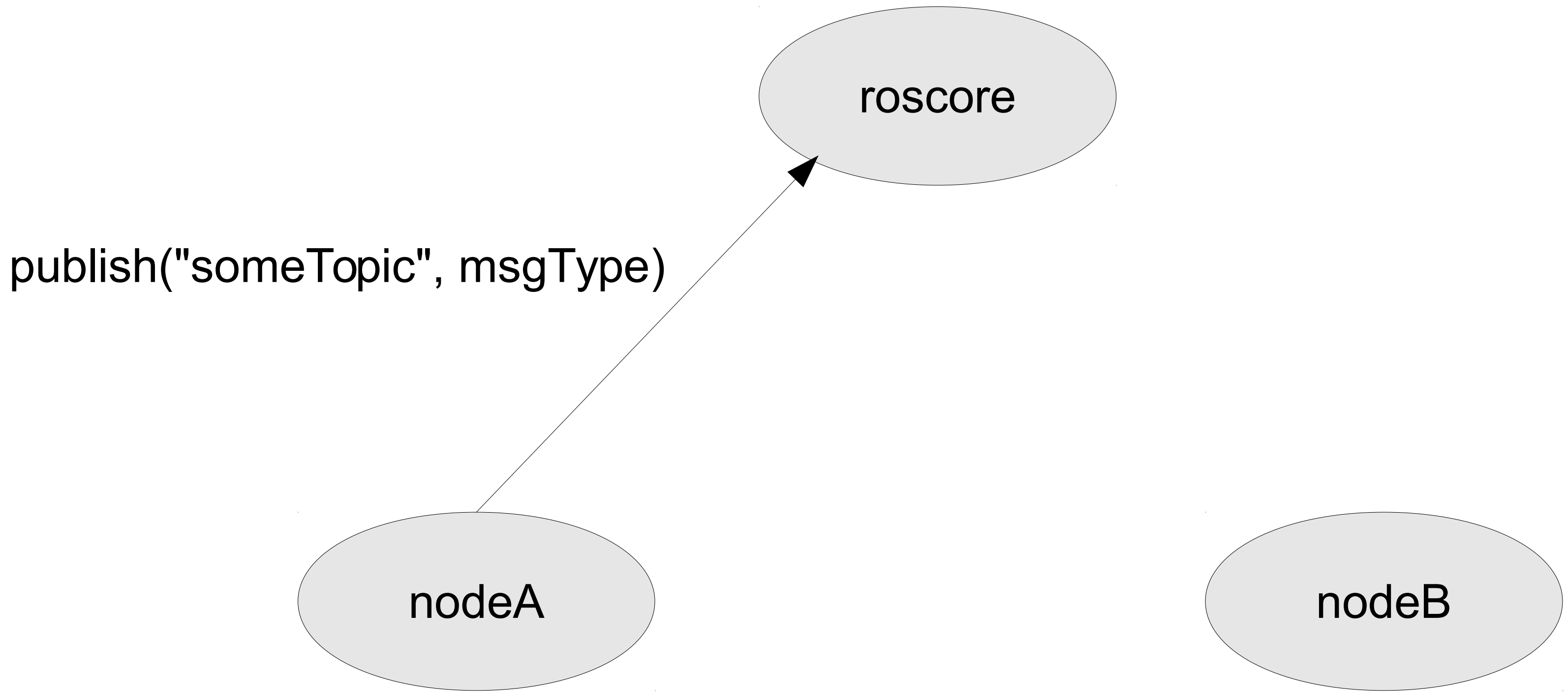
- Any process using the ROS client API
  - C++ (roscpp), Python (rospy), …[1]
- Support for ROS renaming/remapping

[1] third-party: ruby, R, Matlab, Lisp, C

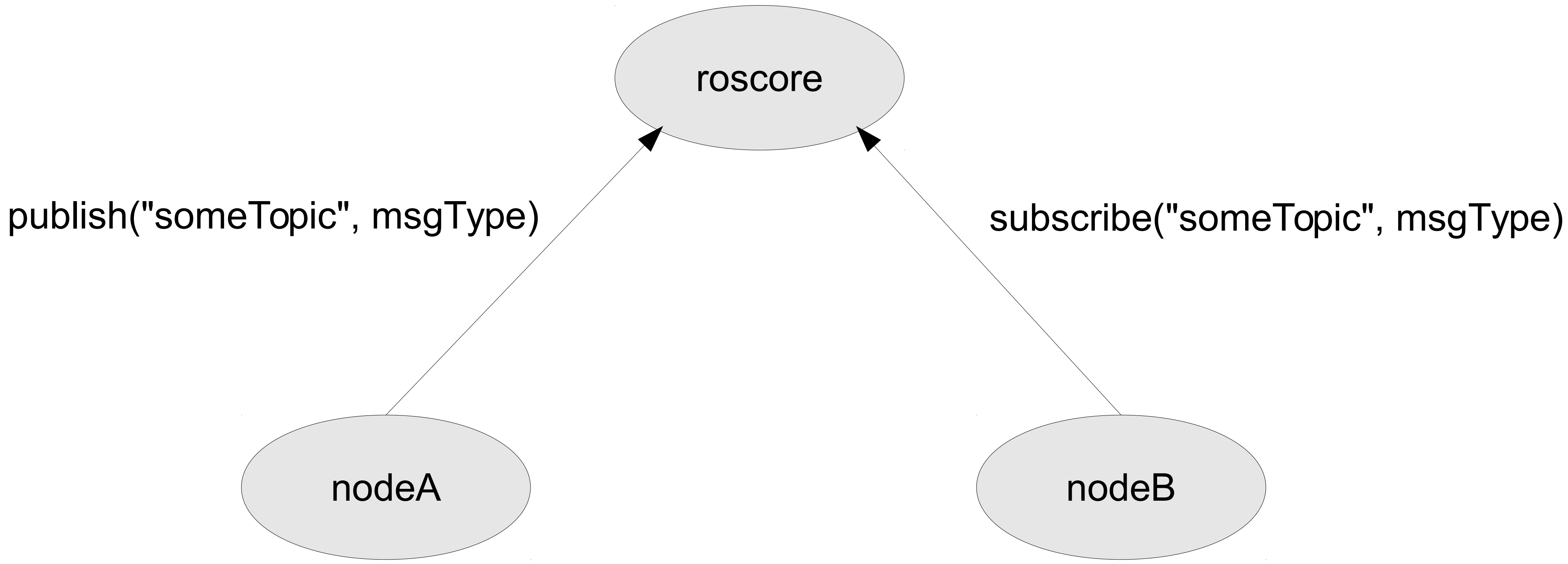
Communication Middleware + Tools:
ROS graph



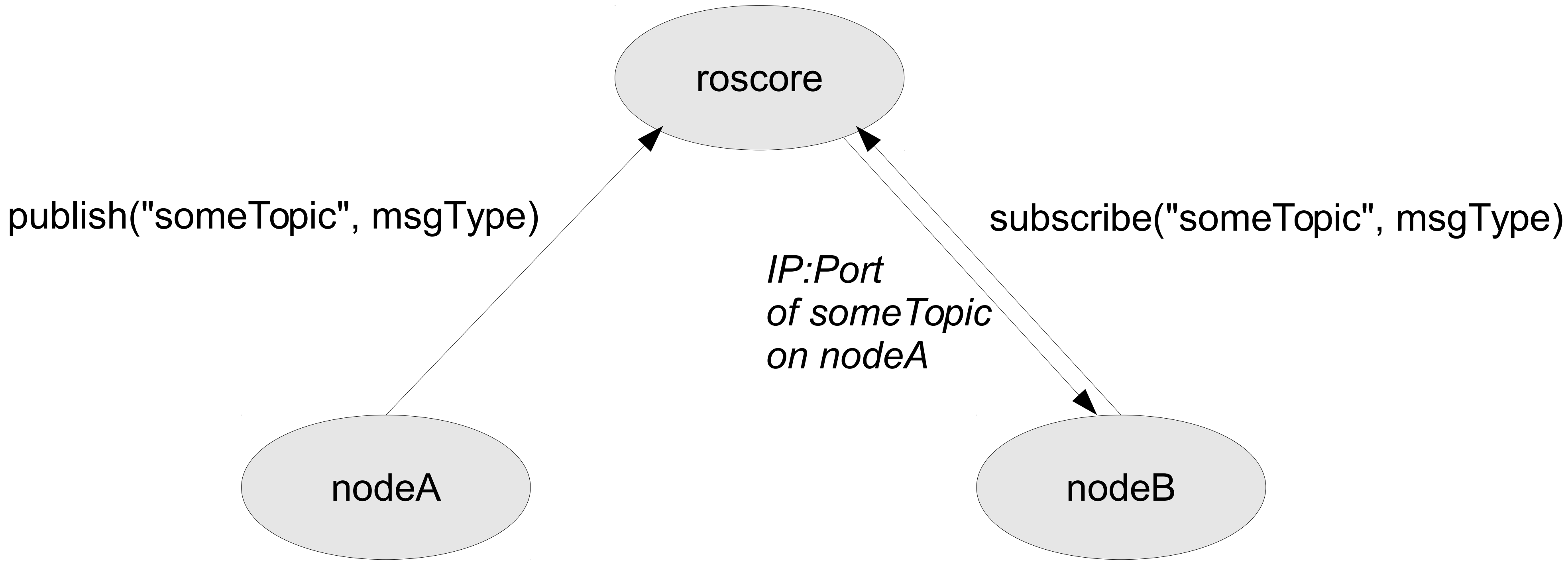
Communication Middleware + Tools:
ROS graph



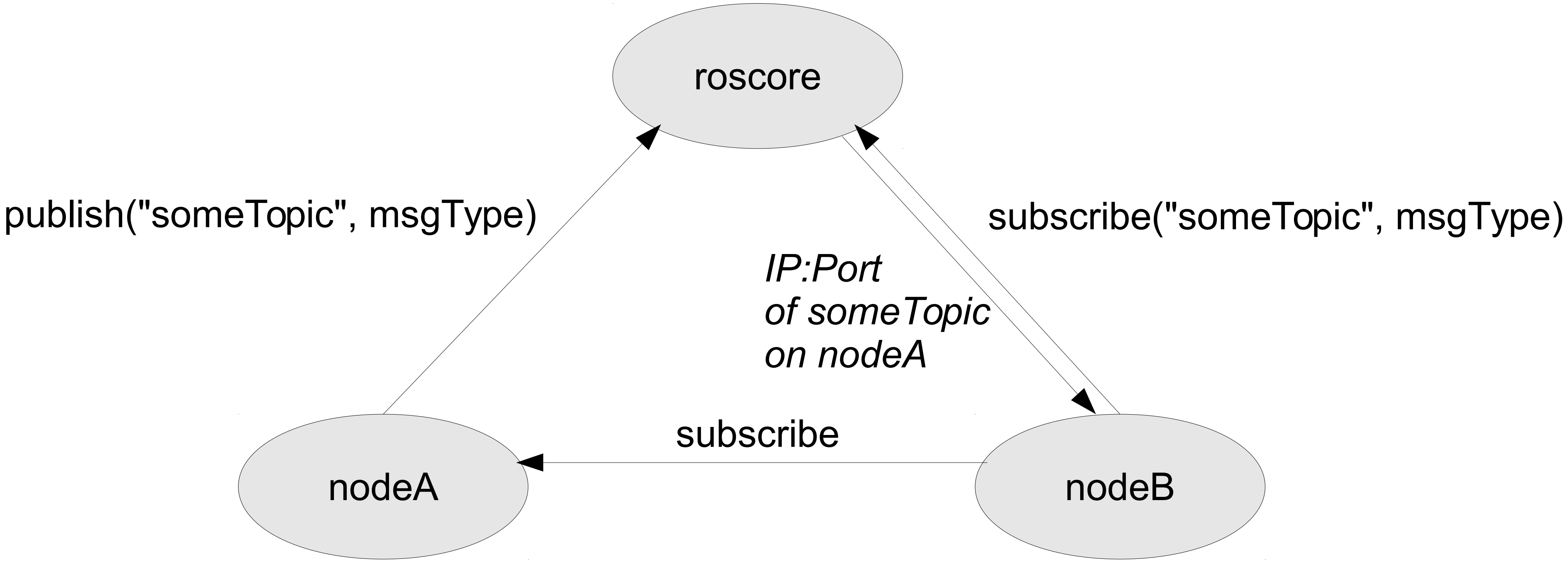
Communication Middleware + Tools:
ROS graph


roscore
publish("someTopic", msgType)
subscribe("someTopic", msgType)
nodeA
nodeB

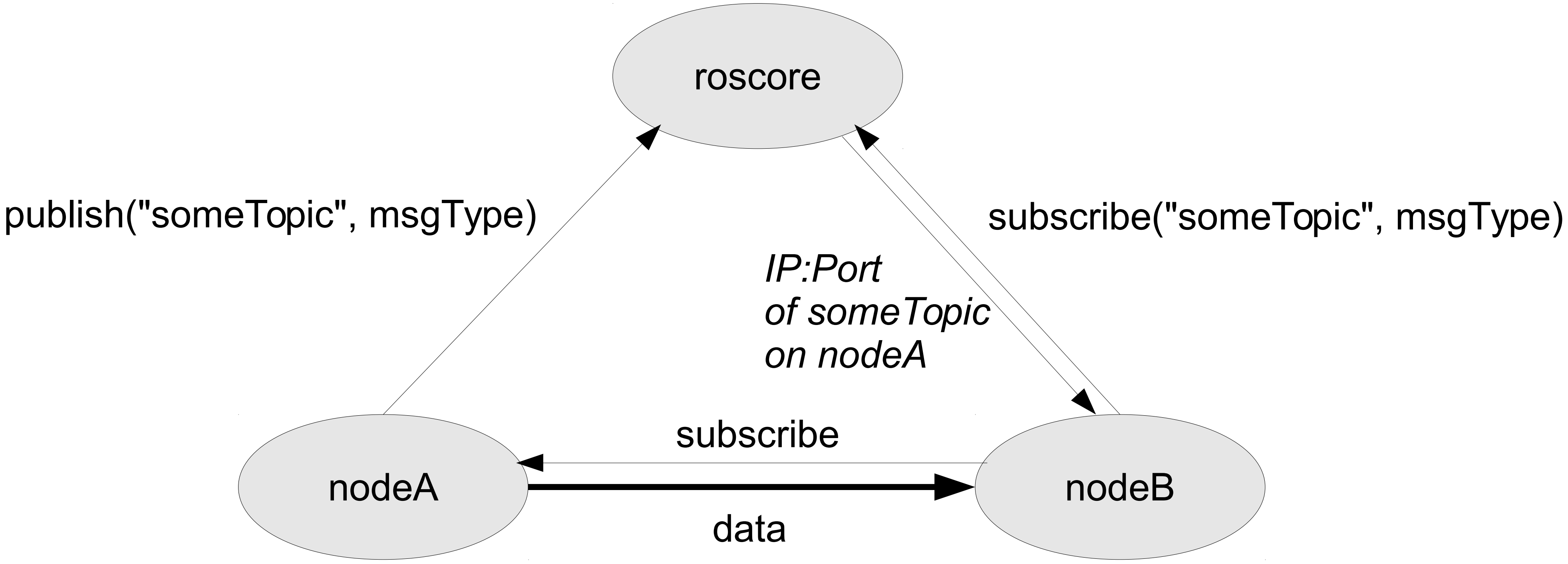
Communication Middleware + Tools:
ROS graph


roscore
publish("someTopic", msgType)
subscribe("someTopic", msgType)
IP:Port
of someTopic
on nodeA
nodeA
nodeB

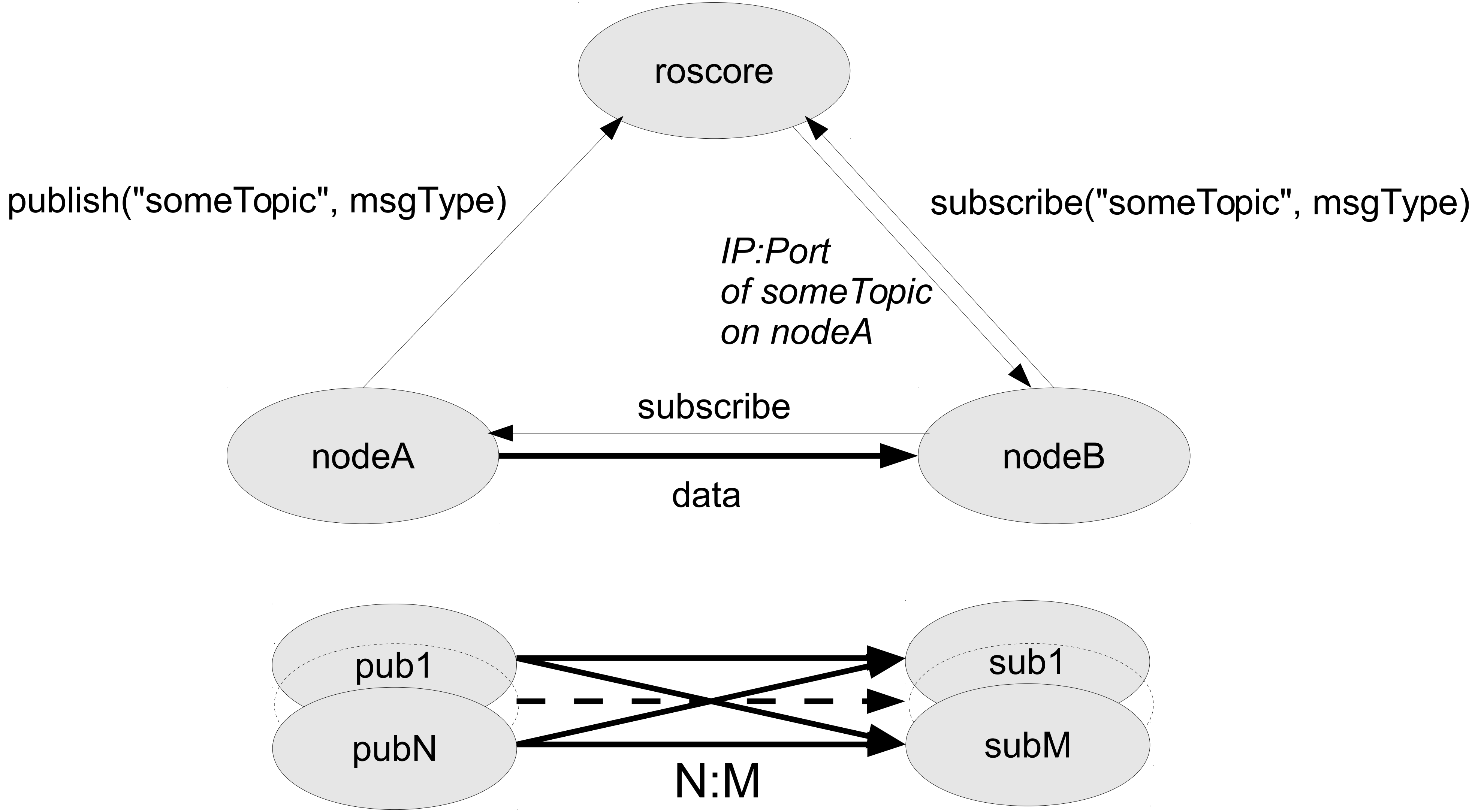
Communication Middleware + Tools:
ROS graph


roscore
publish("someTopic", msgType)
subscribe("someTopic", msgType)
IP:Port
of someTopic
on nodeA
subscribe
nodeA
nodeB

Communication Middleware + Tools:
ROS graph


roscore
publish("someTopic", msgType)
subscribe("someTopic", msgType)
IP:Port
of someTopic
on nodeA
subscribe
nodeA
nodeB
data

Communication Middleware + Tools:
ROS graph



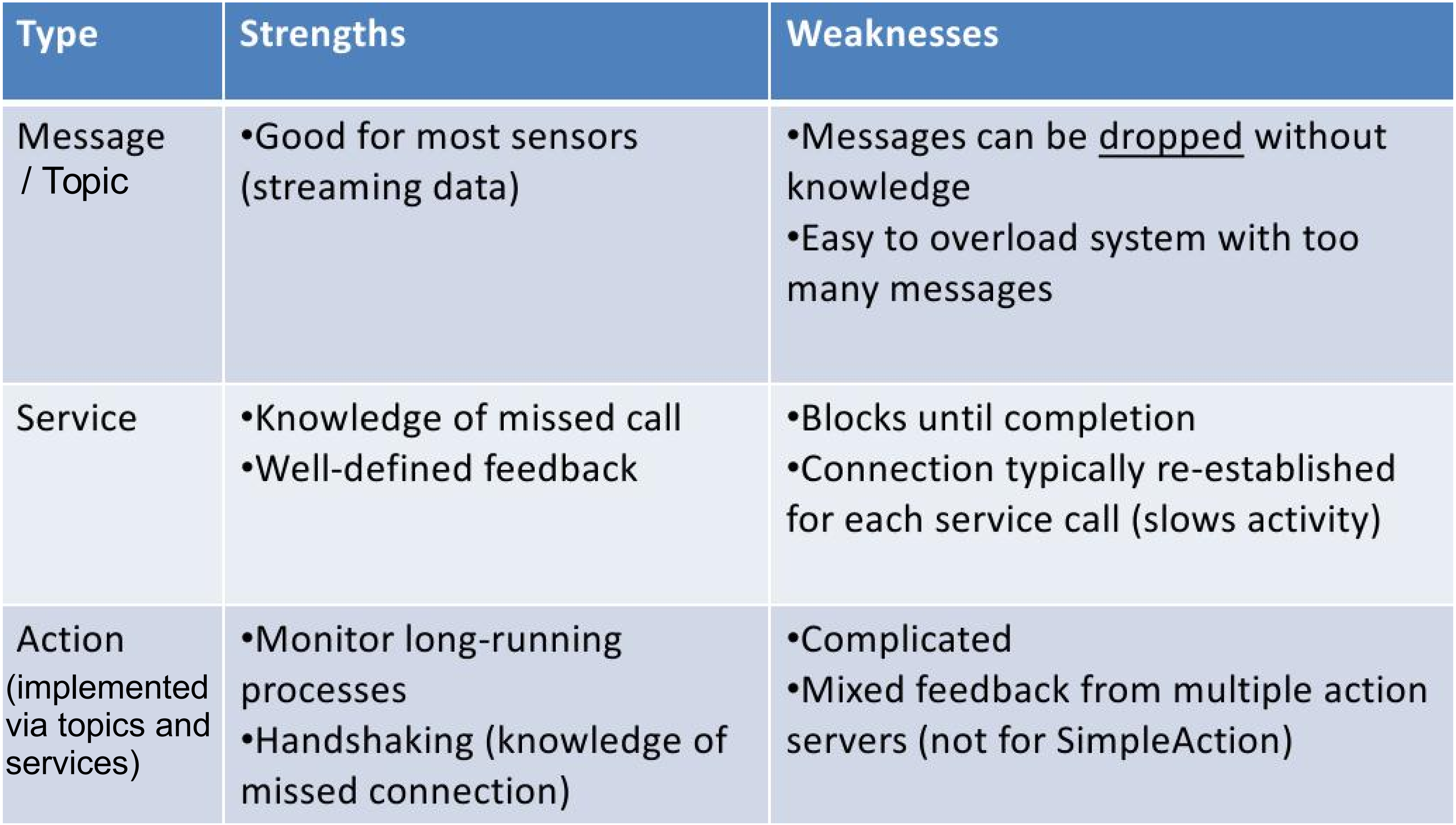
Communication Middleware + Tools: topics

- Names used in publish/subscribe mechanism
- Carry ROS messages of certain type
- Unidirectional

Communication Middleware + Tools: services

- Remote Procedure Calls (RPCs) in ROS: Synchronous Request & Reply

Communication Middleware + Tools:
Comparison



Type	Strengths	Weaknesses
Message / Topic	•Good for most sensors (streaming data)	•Messages can be dropped without knowledge •Easy to overload system with too many messages
Service	•Knowledge of missed call •Well-defined feedback	•Blocks until completion •Connection typically re-established for each service call (slows activity)
Action (implemented via topics and services)	•Monitor long-running processes •Handshaking (knowledge of missed connection)	•Complicated •Mixed feedback from multiple action servers (not for SimpleAction)

Communication Middleware + Tools: parameters

- A shared, multi-variate dictionary that is accessible via network APIs
- For slow changing data only,

  e.g. configuration

Communication Middleware + Tools: Graph Resource Names

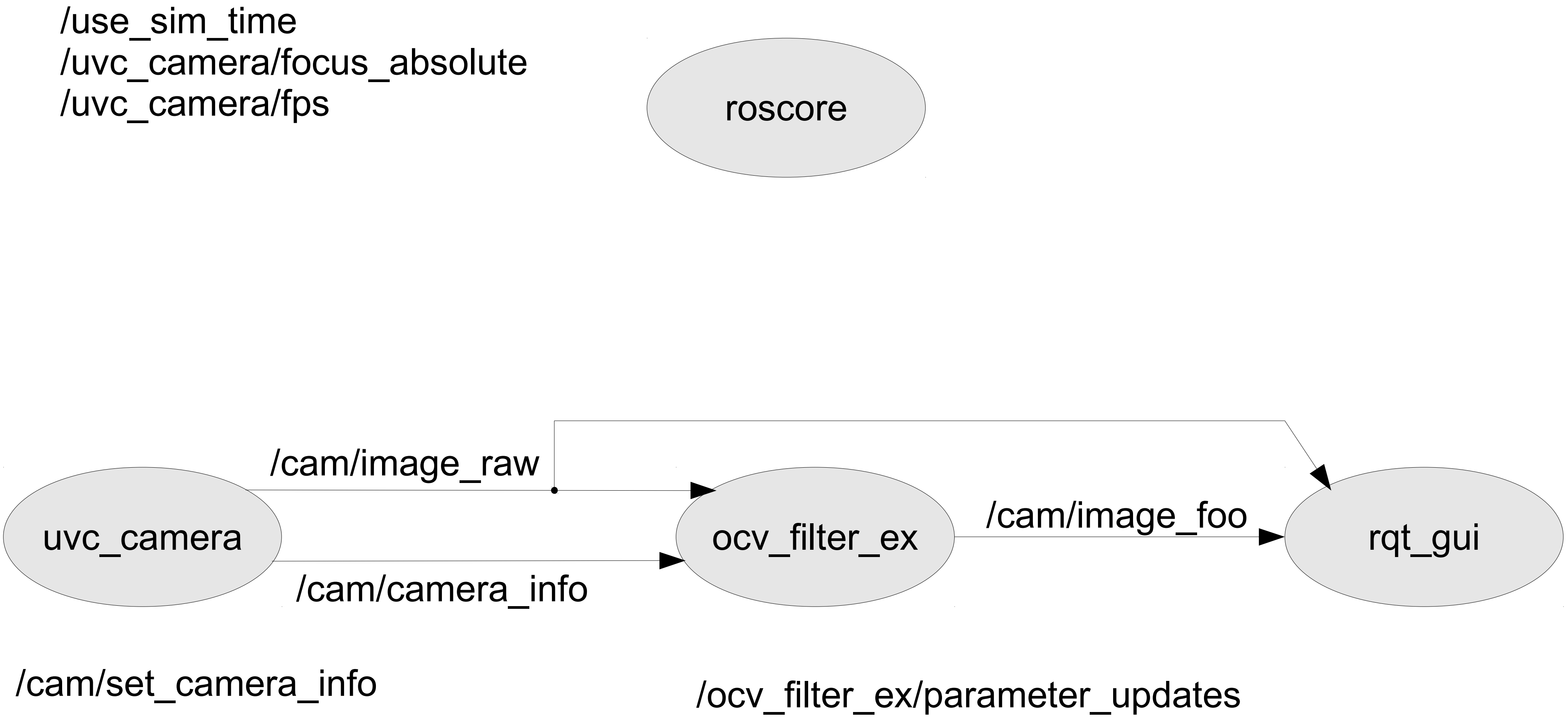
- Three types

Type	Example	Usage
global	"/foo", “/foo/bar“	'Never': (only if name must be unique in whole network)
relative	"foo", “foo/bar“	'Default': If at least two nodes must access it
private	"~foo", “~foo/bar“	'Internal-only': If name should not be known outside a node, e.g. configuration parameters for this node

- Namespaces allow multiple-instances

  roslaunch ns attribute; env ROS_NAMESPACE

Communication Middleware + Tools:
ROS Graph example



/use_sim_time
/uvc_camera/focus_absolute
/uvc_camera/fps

roscore

uvc_camera

/cam/image_raw

/cam/camera_info

ocv_filter_ex

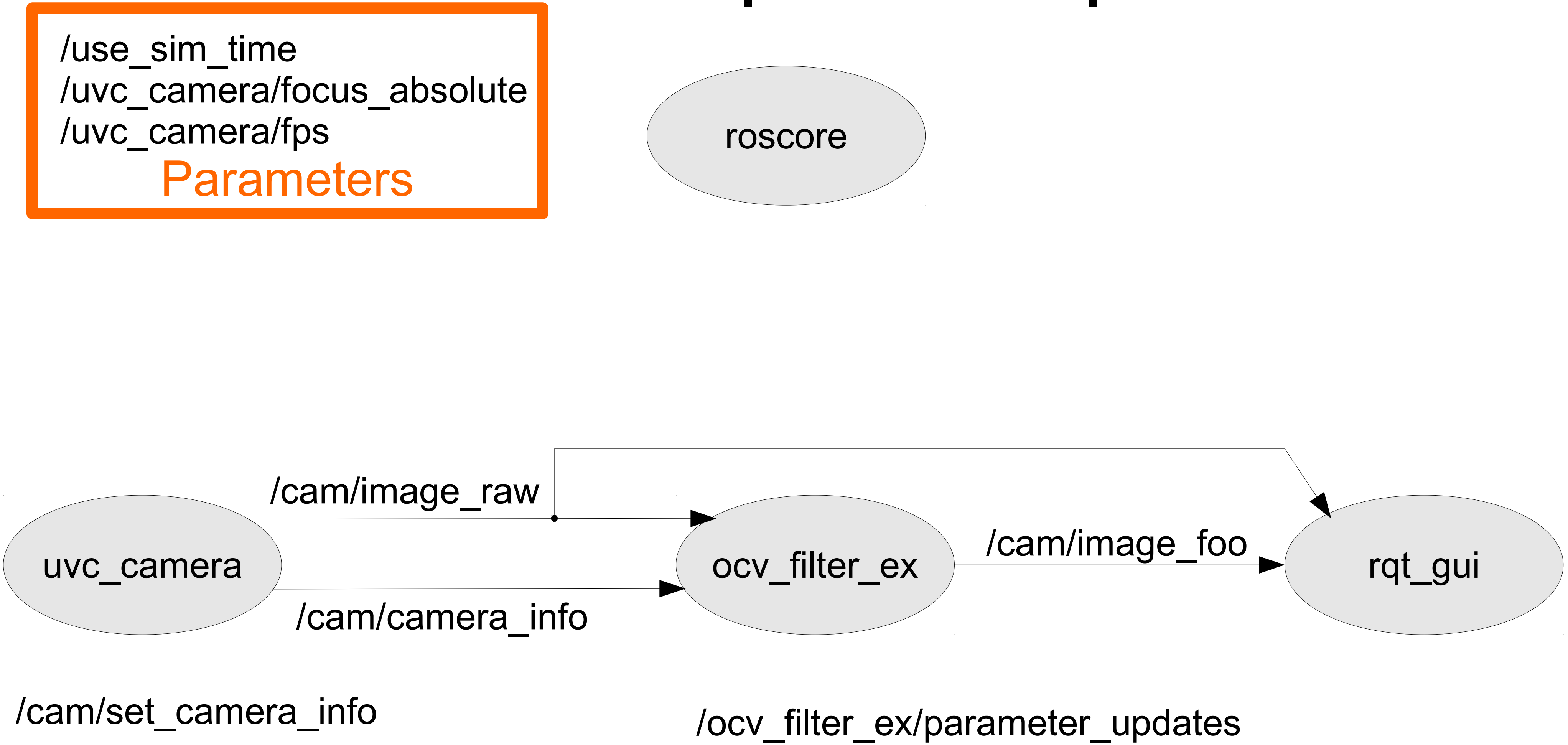
/cam/image_foo

rqt_gui

/cam/set_camera_info

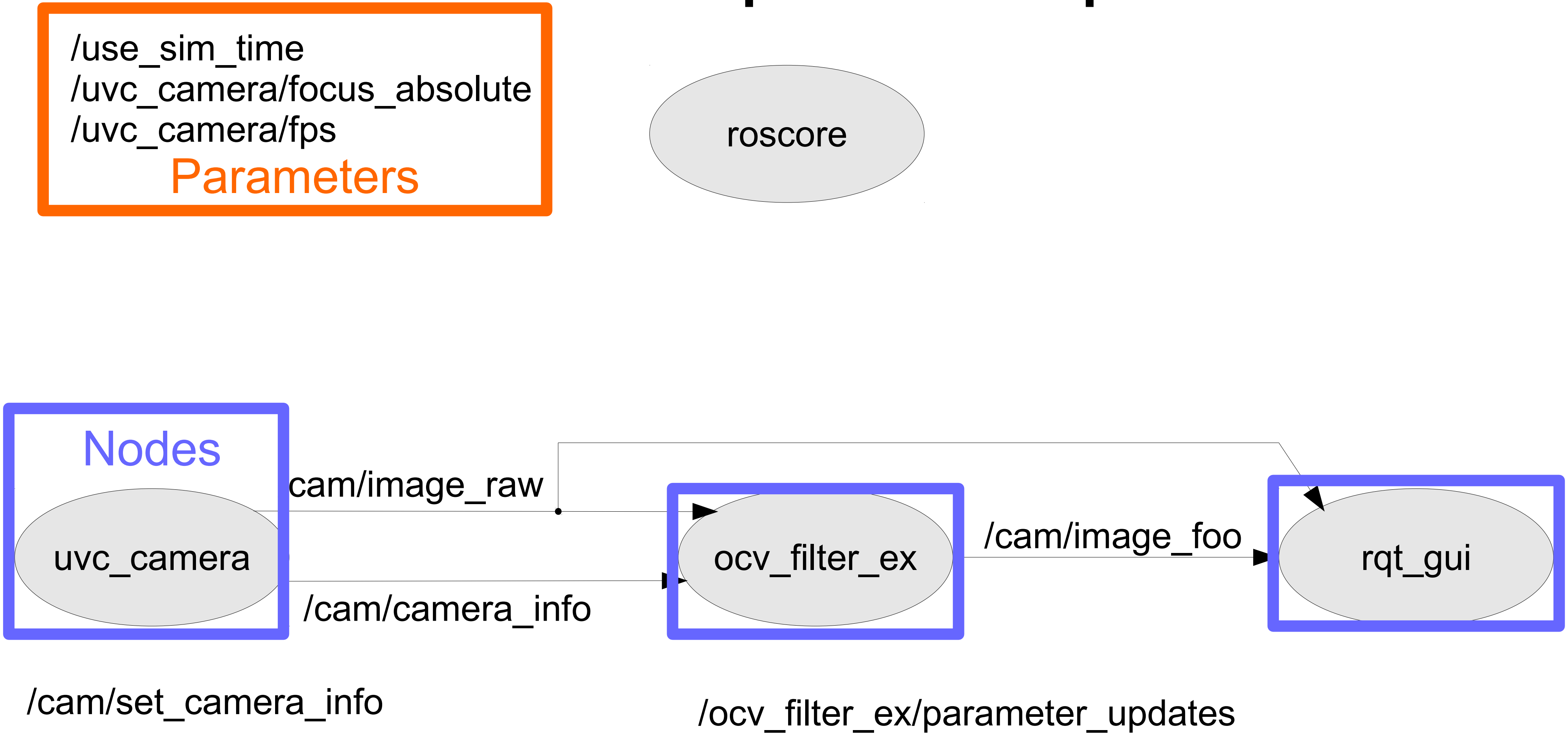
/ocv_filter_ex/parameter_updates

Communication Middleware + Tools:
ROS Graph example

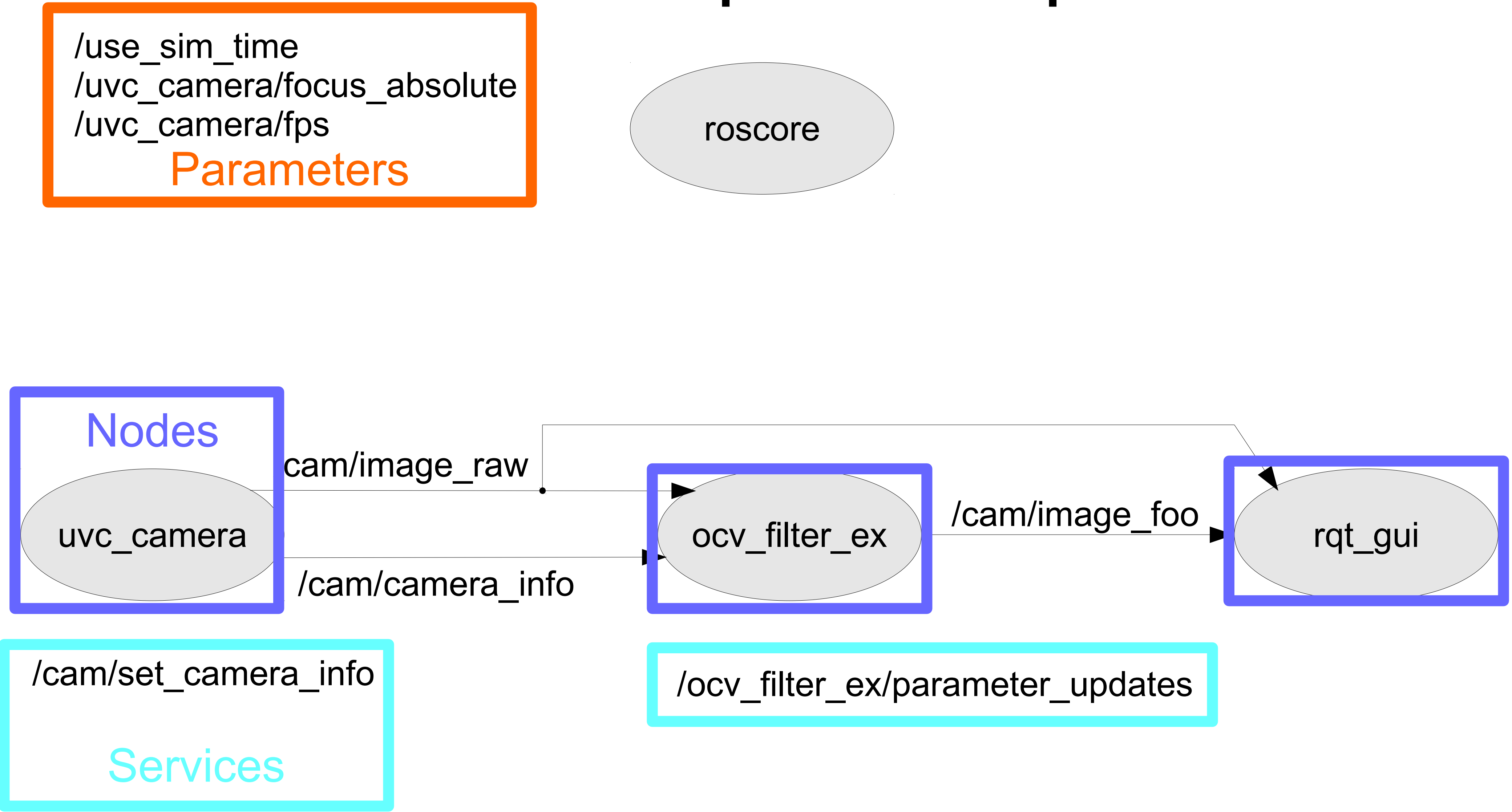


/use_sim_time
/uvc_camera/focus_absolute
/uvc_camera/fps
Parameters

Communication Middleware + Tools:
ROS Graph example

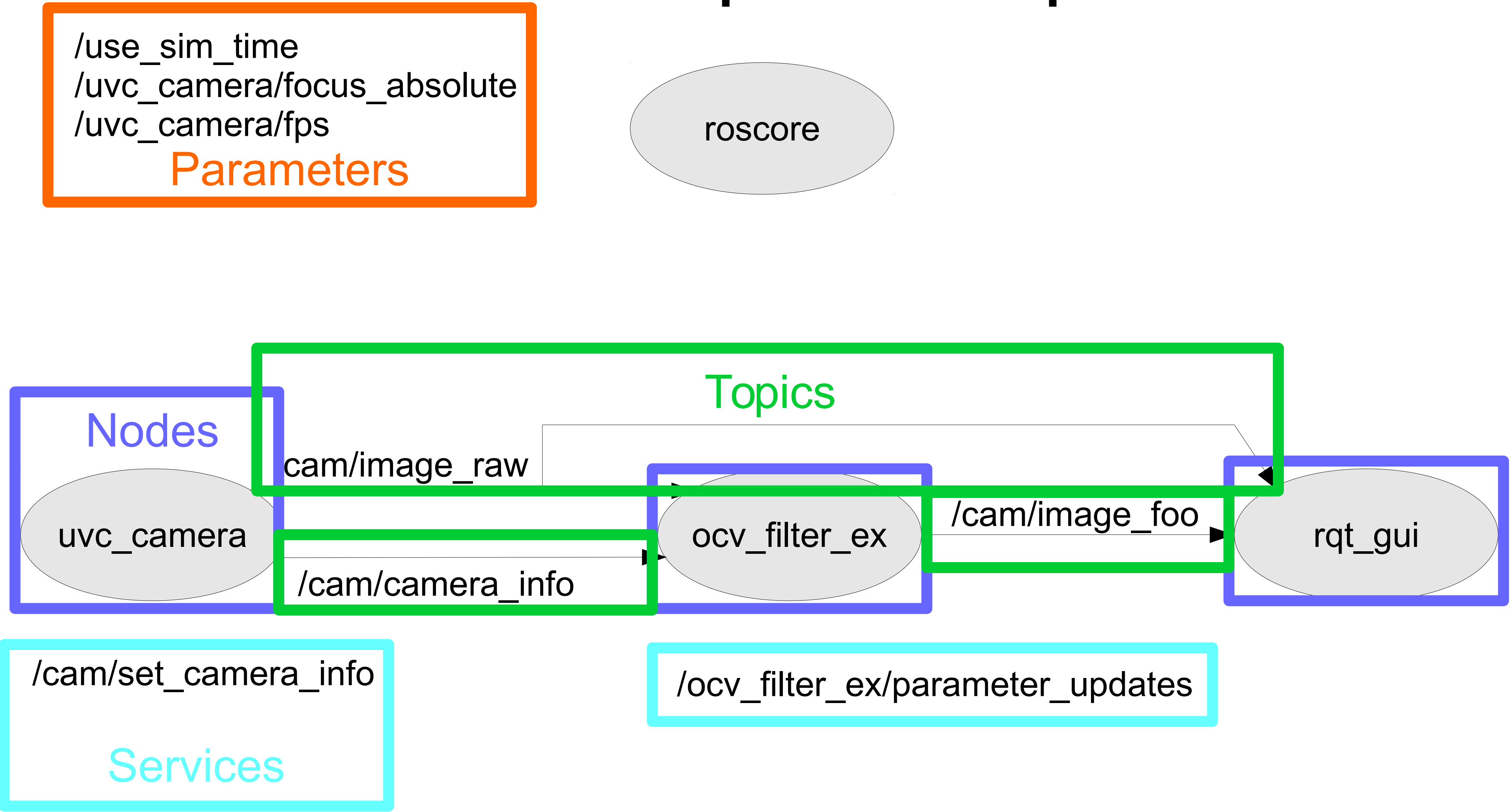


Communication Middleware + Tools:
ROS Graph example



Communication Middleware + Tools:

ROS Graph example



Communication Middleware + Tools: Tools

- rosnode
- rostopic
  - list, info, echo, pub
- rosservice
- rosmsg
- rosparam
- rqt_gui
  - ROS Graph
  - Topic Introspection / Publisher
- rviz

Communication Middleware + Tools:

API: Publisher

Python

```
import rospy
from std_msgs.msg import String

rospy.init_node('test_pub_node')

pub = rospy.Publisher('atopic',
                      String)

stringMsg = String()
stringMsg.data = 'foo'
pub.publish(stringMsg)
```

C++

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

ros::init(argc, argv, "test_pub_node");
ros::NodeHandle nh;

ros::Publisher pub
  = nh.advertise<std_msgs::String>(
    "atopic", 10);

std_msgs::String stringMsg;
stringMsg.data = "foo";
pub.publish(stringMsg);
```

ahb_rospy_example

Communication Middleware + Tools:

API: Subscriber

Python

```python
import rospy
from std_msgs.msg import String

rospy.init_node('test_sub_node')

def a_callback(msg):
  rospy.loginfo('got msg: %s'
                % msg)

sub = rospy.Subscriber(
        'atopic', String,
        a_callback)
rospy.spin()
```

C++

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

ros::init(argc, argv, "test_sub_node");
ros::NodeHandle nh;

void aCallback(
  const std_msgs::String::ConstPtr& msg)
{ ROS_INFO_STREAM("got msg:" << *msg);}

ros::Subscriber sub
  = nh.subscribe<std_msgs::String>(
    “atopic", 10, aCallback);
ros::spin();
```

ahb_rospy_example

Packages and Build System:
catkin build system

- Yet another build system?

  Somewhat yes but fortunately mostly no.

Packages and Build System: catkin build system

- Yet another build system?

  Somewhat yes but fortunately mostly no.

- Basically, a python wrapper around CMake.
- Simplifies handling of intra-ROS package dependencies.

Packages and Build System: catkin build system

- Yet another build system?

  Somewhat yes but fortunately mostly no.

- Basically, a python wrapper around CMake.
- Simplifies handling of intra-ROS package dependencies.

- Don't worry!

Packages and Build System:
Setup catkin workspace

```
mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/src

catkin_init_workspace
```

Packages and Build System:
Create a new catkin package

```
cd ~/catkin_ws/src

catkin_create_pkg name_of_new_pkg dependency1 dependencyN


cd name_of_new_pkg

# Edit package.xml and CMakeLists.txt


# Add content
```

Packages and Build System:
Compile catkin packages

```
cd ~/catkin_ws
catkin_make
# source ~/.zshrc

# For debugging of compilation errors:
catkin_make VERBOSE=true -j1
```

Required Computing Hardware?

- Anything x86 or ARM running Ubuntu 14.04
  - Also basic support for OS X and Windows (Matlab).

Required Computing Hardware?

- Anything x86 or ARM running Ubuntu 14.04
  - Also basic support for OS X and Windows (Matlab).

- Due to the distributed nature of ROS:
  - A powerful machine is good, several are better.
  - Often a combination of smaller slower on-robot machines (e.g. BeagleBone Black, Intel NUC, Zotac ZBOX) and faster off-robot desktop computers (~Core i5 + GPU) works best.

Required Computing Hardware?

- Anything x86 or ARM running Ubuntu 14.04
  - Also basic support for OS X and Windows (Matlab).

- Due to the distributed nature of ROS:
  - A powerful machine is good, several are better.
  - Often a combination of smaller slower on-robot machines (e.g. BeagleBone Black, Intel NUC, Zotac ZBOX) and faster off-robot desktop computers (~Core i5 + GPU) works best.
- Actual resource requirements completely depend on application.

Cameras

for cameraType in 'mono', 'stereo', 'rgb-d':

- Drivers
- Calibration
- Visualization
- Processing and Filtering
- Object Recognition (not in this talk)

Mono Cameras: Drivers

(*here*: UVC-compliant devices, e.g. webcams)

Create launch file c910.launch:

```
<launch>
```

https://github.com/ktossell/camera_umd/tree/master/uvc_camera

Mono Cameras: Drivers

(*here*: UVC-compliant devices, e.g. webcams)

Create launch file c910.launch:

```
<launch>
  <node ns=“/cam“
          pkg="uvc_camera" type="uvc_camera_node" name="uvc_camera_c910"
          output="screen">
```

https://github.com/ktossell/camera_umd/tree/master/uvc_camera

Mono Cameras: Drivers

(*here*: UVC-compliant devices, e.g. webcams)

Create launch file c910.launch:

```
<launch>
  <node ns=“/cam“
             pkg="uvc_camera" type="uvc_camera_node" name="uvc_camera_c910"
             output="screen">
    <param name="width" type="int" value="800" />
    <param name="height" type="int" value="600" />
    <param name="fps" type="int" value="20" />
    <param name="frame" type="string" value="wide_stereo" />

    <param name="auto_focus" type="bool" value="False" />
    <param name="focus_absolute" type="int" value="0" />
    <!-- other supported params: auto_exposure, exposure_absolute, brightness, ... -->

    <param name="device" type="string" value="/dev/video0" />
    <param name="camera_info_url" type="string"
      value="file://$(find stereo_webcam)/config/single_c910.yaml" />
  </node>
</launch>
```

https://github.com/ktossell/camera_umd/tree/master/uvc_camera

Mono Cameras: Drivers

- Run launch file: `roslaunch c910.launch`
- Check current ROS graph
  - Nodes: `rosnode list`
  - Topics: `rostopic list`
- View camera stream and ROS graph: `rqt_gui`

mono_camera_driver.mkv

Mono Cameras: Drivers

- Run launch file: `roslaunch c910.launch`
- Check current ROS graph
  - Nodes: `rosnode list`
  - Topics: `rostopic list`
- View camera stream and ROS graph: `rqt_gui`

mono_camera_driver.mkv

Reminder: Everything is network transparent.

Mono Cameras: Calibration

- Run calibration assistant (hint: create a launch file for future use):

```
rosrun camera_calibration cameracalibrator.py
  --size 8x6 --square 0.0255
  image:=/cam/image_raw camera:=/cam
```

http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

Mono Cameras: Calibration

- Run calibration assistant (hint: create a launch file for future use):

```
rosrun camera_calibration cameracalibrator.py
  --size 8x6 --square 0.0255
  image:=/cam/image_raw camera:=/cam
```

runtime name remapping

http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

Mono Cameras: Calibration

- Run calibration assistant (hint: create a launch file for future use):

```
rosrun camera_calibration cameracalibrator.py
  --size 8x6 --square 0.0255
  image:=/cam/image_raw camera:=/cam
```

runtime name remapping

mono_camera_calibration.mkv

http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

Mono Cameras: Calibration

- Run calibration assistant (hint: create a launch file for future use):

```
rosrun camera_calibration cameracalibrator.py
  --size 8x6 --square 0.0255
  image:=/cam/image_raw camera:=/cam
```

runtime name remapping

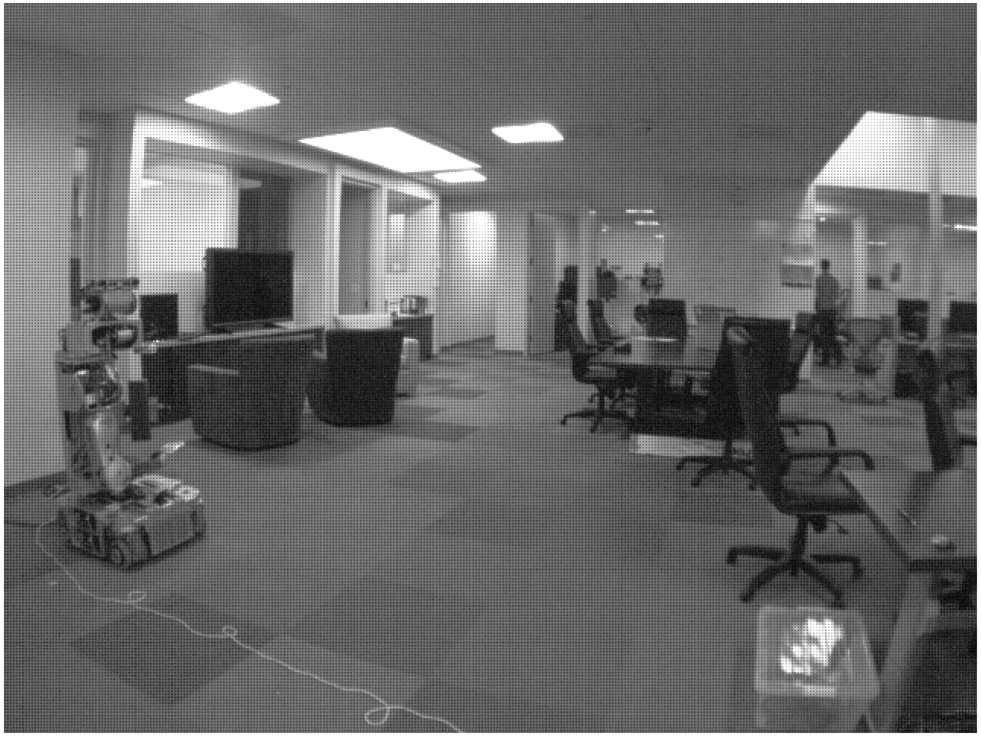
mono_camera_calibration.mkv

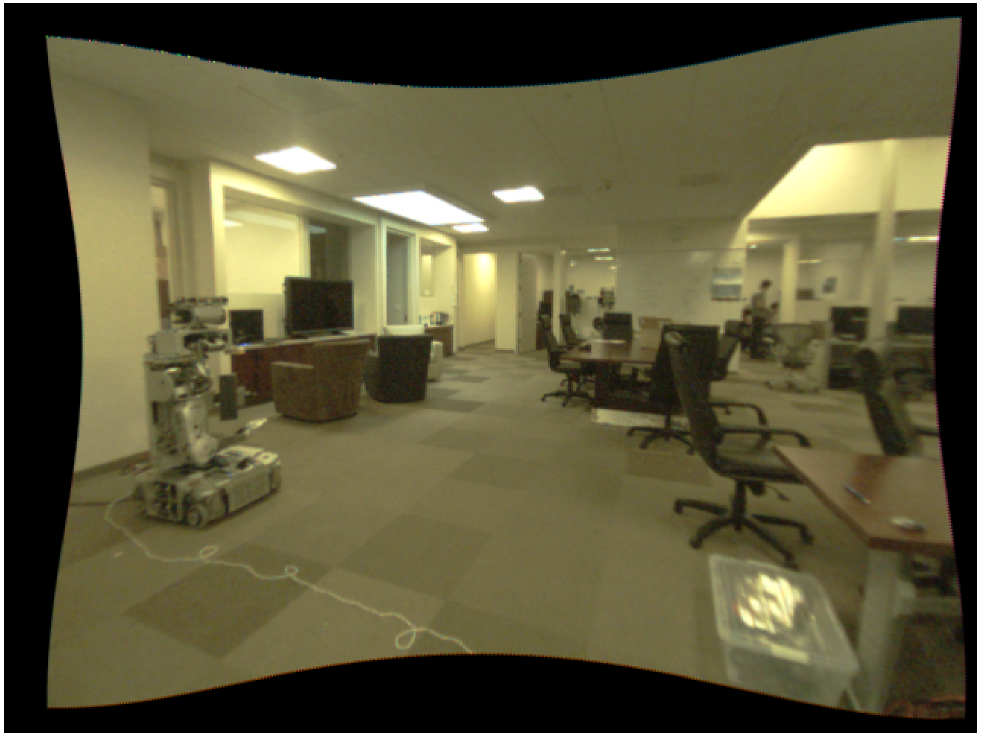
Note: Everything happens during runtime and camera node remains running throughout.

http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

Mono Cameras: Processing

- Debayering, Undistort, Rectification and other common image processing tasks already available.





http://wiki.ros.org/image_proc

Mono Cameras: Processing

- Simple custom image processing node (using OpenCV):
  - Subscribe to sensor_msgs/Image topic
  - Apply edge filter to image
  - Publish filtered image as sensor_msgs/Image
  - Filter parameters can be changed during runtime via dynamic reconfigure

mono_camera_processing_opencv_dynamic_reconfigure.mkv

http://wiki.ros.org/cv_bridge
http://wiki.ros.org/dynamic_reconfigure
https://github.com/andreasBihlmaier/ahb_ros_opencv_dynamic_reconfigure_example

Mono Cameras: Processing

- Simple custom image processing node (using OpenCV):
  - Subscribe to sensor_msgs/Image topic
  - Apply edge filter to image
  - Publish filtered image as sensor_msgs/Image
  - Filter parameters can be changed during runtime via dynamic reconfigure

mono_camera_processing_opencv_dynamic_reconfigure.mkv

http://wiki.ros.org/cv_bridge
http://wiki.ros.org/dynamic_reconfigure
https://github.com/andreasBihlmaier/
ahb_ros_opencv_dynamic_reconfigure_example

36 Lines of Code!

Stereo Cameras: Drivers

(*here*: again 2x UVC-compliant devices, e.g. webcams)

Again, create launch file stereo_c910.launch:

```
<launch>
```

stereo_driver.avi

https://github.com/ktossell/camera_umd/tree/master/uvc_camera

Stereo Cameras: Drivers

(*here*: again 2x UVC-compliant devices, e.g. webcams)

Again, create launch file stereo_c910.launch:

```
<launch>
  <node ns="/cam"
          pkg="uvc_camera" type="uvc_stereo_node" name="uvc_camera_stereo"
          output="screen">
```

stereo_driver.avi

https://github.com/ktossell/camera_umd/tree/master/uvc_camera

Stereo Cameras: Drivers

(*here*: again 2x UVC-compliant devices, e.g. webcams)

Again, create launch file stereo_c910.launch:

```
<launch>
 <node ns="/cam"
         pkg="uvc_camera" type="uvc_stereo_node" name="uvc_camera_stereo"
         output="screen">
  <param name="width" type="int" value="960" />
  <param name="height" type="int" value="544" />
  <param name="fps" type="int" value="30" />
  <param name="frame" type="string" value="wide_stereo" />

  <param name="auto_focus" type="bool" value="False" />
  <param name="focus_absolute" type="int" value="0" />

  <param name="left/device" type="string" value="/dev/video0" />
  <param name="right/device" type="string" value="/dev/video1" />

  <param name="left/camera_info_url" type="string"
           value="file://$(find stereo_webcam)/config/left.yaml" />
  <param name="right/camera_info_url" type="string"
           value="file://$(find stereo_webcam)/config/right.yaml" />
 </node>
</launch>
```

stereo_driver.avi

https://github.com/ktossell/camera_umd/tree/master/uvc_camera

Stereo Cameras: Calibration

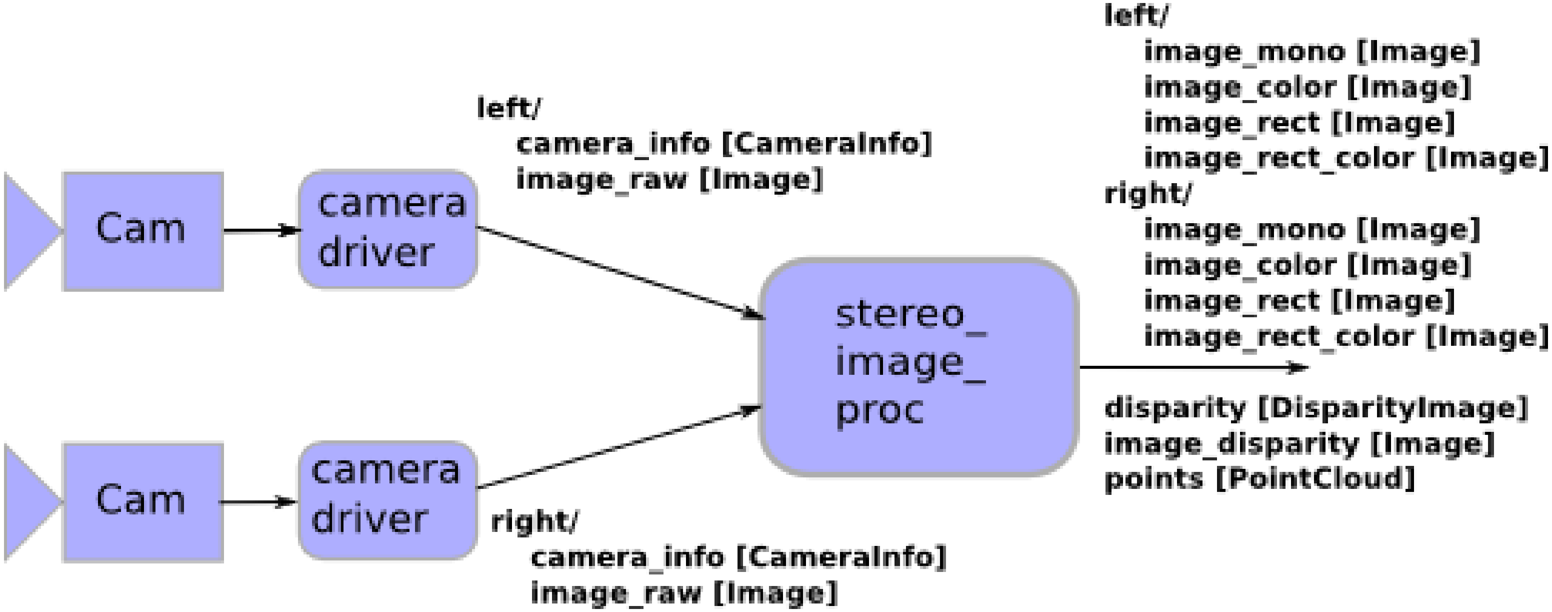
- Run stereo calibration assistant:

```
rosrun camera_calibration cameracalibrator.py
  --size 8x6 --square 0.0255
  right:=/cam/right/image_raw
  right_camera:=/cam/right
  left:=/cam/left/image_raw
  left_camera:=/cam/left
```

http://wiki.ros.org/camera_calibration/Tutorials/StereoCalibration

Stereo Cameras: Reconstruction

Recover depth information from calibrated stereo cameras.

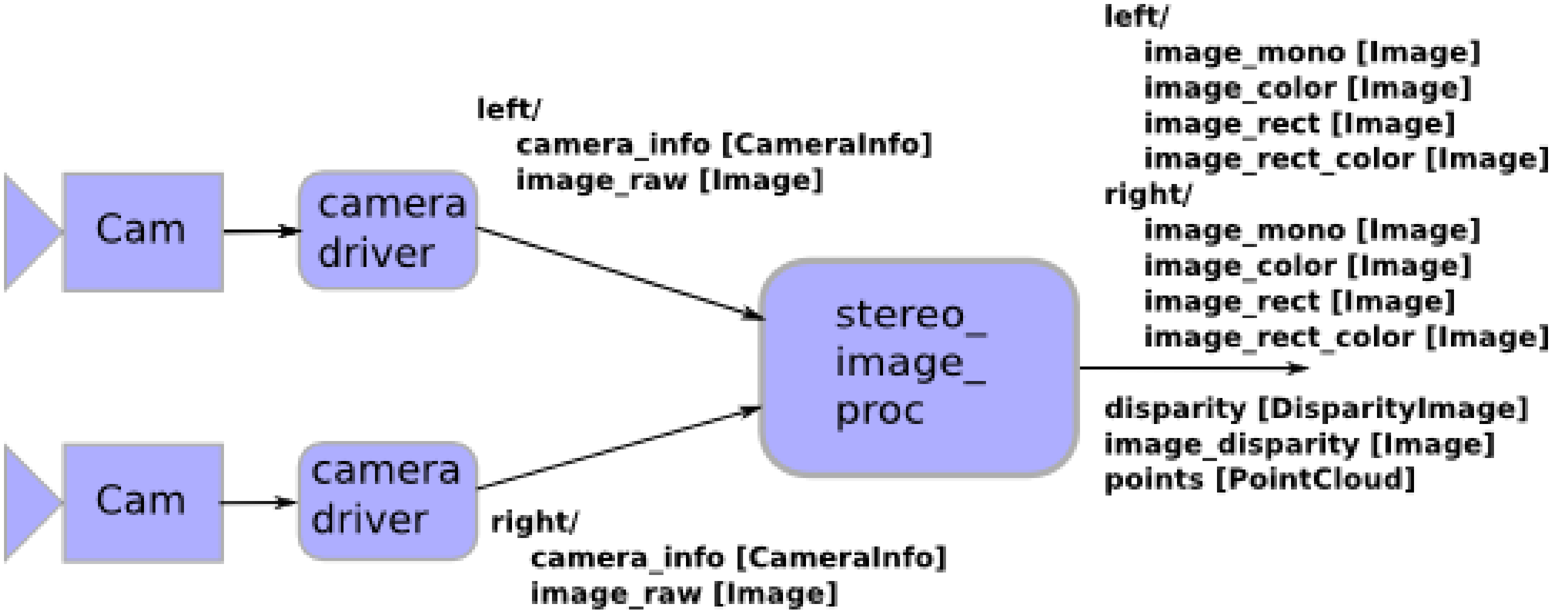


... but many parameters and things that might go wrong.

http://wiki.ros.org/stereo_image_proc

Stereo Cameras: Reconstruction

Recover depth information from calibrated stereo cameras.



... but many parameters and things that might go wrong.

Result: PCL point clouds :)

http://wiki.ros.org/stereo_image_proc

RGB-D Cameras: Drivers

(*here*: OpenNI-compatible devices, e.g. Kinect or Xtion)

```
roslaunch openni2_launch openni2.launch
```

(To get *RGB* point clouds (depth_registered/points), use rqt dynamic reconfigure to enable „depth registration“ and „color_depth_synchronisation“ for /camera/driver)

xtion.avi

http://wiki.ros.org/openni2_launch

RGB-D Cameras: Drivers

(*here*: OpenNI-compatible devices, e.g. Kinect or Xtion)

```
roslaunch openni2_launch openni2.launch
```

(To get *RGB* point clouds (depth_registered/points), use rqt dynamic reconfigure to enable „depth registration“ and „color_depth_synchronisation“ for /camera/driver)

xtion.avi

(Live Demo)

http://wiki.ros.org/openni2_launch

RGB-D Cameras: Drivers

(*here*: OpenNI-compatible devices, e.g. Kinect or Xtion)

```
roslaunch openni2_launch openni2.launch
```

(To get *RGB* point clouds (depth_registered/points), use rqt dynamic reconfigure to enable „depth registration“ and „color_depth_synchronisation“ for /camera/driver)

xtion.avi

(Live Demo)

Note: Same output type as stereo reconstruction, i.e. downstream nodes (e.g. rviz) are device agnostic. Runtime switching possible.

http://wiki.ros.org/openni2_launch

Robots

- Modeling
- Visualization
- Motion planning

Robots: Modeling

(*here*: very simple: servo motors connected through beams)

```
<robot name="arm_31c3">
```

http://wiki.ros.org/urdf

Robots: Modeling

(*here*: very simple: servo motors connected through beams)

```
<robot name="arm_31c3">
  <link name=“base_link“>
    <visual>...</visual>          geometric primitives and meshes
    <collision>...</collision>   geometric primitives and convex(!) meshes
    <inertial>...</inertial>
  </link>
```

http://wiki.ros.org/urdf

Robots: Modeling

(*here*: very simple: servo motors connected through beams)

```
<robot name="arm_31c3">
 <link name=“base_link“>
  <visual>...</visual>          geometric primitives and meshes
  <collision>...</collision>  geometric primitives and convex(!) meshes
  <inertial>...</inertial>
 </link>
 <joint name="base_to_upper_arm_joint" type="revolute">
   <origin xyz="0 0 0.05" rpy="0 0 0"/>
   <parent link=“base_link“ />
   <child link=“upper_arm_link“ />
   <limit lower="-1.57079" upper="1.57079" effort="1" velocity="1.0" />
 </joint>
</robot>
```

http://wiki.ros.org/urdf

Robots: Modeling

(*here*: very simple: servo motors connected through beams)

```
<robot name="arm_31c3">
 <link name=“base_link“>
   <visual>...</visual>         geometric primitives and meshes
   <collision>...</collision>   geometric primitives and convex(!) meshes
   <inertial>...</inertial>
 </link>
 <joint name="base_to_upper_arm_joint" type="revolute">
    <origin xyz="0 0 0.05" rpy="0 0 0"/>
    <parent link=“base_link“ />
    <child link=“upper_arm_link“ />
    <limit lower="-1.57079" upper="1.57079" effort="1" velocity="1.0" />
 </joint>
</robot>
```

```
roslaunch urdf_tutorial display.launch
          gui:=True model:=arm_31c3.urdf
```

robot_modeling.avi

http://wiki.ros.org/urdf

Robots: Motion planning

(even without math)

- Nowadays, there is nice GUI to create all configuration for motion planning based on the URDF description:

```
roslaunch moveit_setup_assistant
              setup_assistant.launch
```

robot_motion_planning_setup_assistant.avi

http://wiki.ros.org/urdf

Robots: Motion planning

(even without math)

- Nowadays, there is nice GUI to create all configuration for motion planning based on the URDF description:

```
roslaunch moveit_setup_assistant
                setup_assistant.launch
```

robot_motion_planning_setup_assistant.avi

- Once configured many state-of-the-art sampling-based motion planners (OMPL) are available to move your custom robot.

robot_motion_planning_demo.avi

http://wiki.ros.org/urdf

Robots: Motion planning

(even without math)

- Nowadays, there is nice GUI to create all configuration for motion planning based on the URDF description:

```
roslaunch moveit_setup_assistant
          setup_assistant.launch
```

robot_motion_planning_setup_assistant.avi

- Once configured many state-of-the-art sampling-based motion planners (OMPL) are available to move your custom robot.

robot_motion_planning_demo.avi

Note: GUI is well seperated from API

http://wiki.ros.org/urdf

Simulation

- Modeling
  - URDF vs SDF
- Working with real robot vs simulated robot
  - /use_sim_time

http://gazebosim.org/

Simulation: Modeling

- Due to historical reasons ... there are two ROS robot description formats: URDF and SDF
- Fortunately, leaving aside the unfortunate details, there are converters:
  - gz sdf --print robot.urdf > robot.sdf
  - sdf2urdf.py robot.sdf robot.urdf

http://gazebosim.org/sdf.html
http://wiki.ros.org/pysdf

Simulation: Robot Unit Testing

- ROS nodes can be transparently run against simulated robot (actuators and sensors)
- Many possibilities: Test-Driven Development, Continuous Integration, distributed development without access to hardware, ...

http://gazebosim.org/tutorials?tut=ros_comm

Simulation: Robot Unit Testing

- ROS nodes can be transparently run against simulated robot (actuators and sensors)
- Many possibilities: Test-Driven Development, Continuous Integration, distributed development without access to hardware, ...

- Assuming we attached a webcam and RGB-D camera to the simple robots endeffector:

simulation.avi

http://gazebosim.org/tutorials?tut=ros_comm

Tools and Debugging/Introspection

- Command Line
- rqt_gui
- rviz

Outlook

GAZEBO	http://gazebosim.org/	Robot simulator
ROS control	http://wiki.ros.org/ros_control	Control loop mechanism
MoveIt!	http://moveit.ros.org/	Motion planning
ROS industrial	http://rosindustrial.org/	ROS in manufactoring
OpenCV	http://opencv.org/	Computer vision
pcl	http://pointclouds.org/	Point cloud processing

Outlook cont.

- nodelets
- navigation / SLAM
- tf
- actionlib
- capabilities
- ROS Industrial
- Augmented Reality: Beamers and (RGB-)LEDs
- industrial_calibration
- KnowRob

Outlook cont.

- http://wiki.ros.org/Sensors
- http://wiki.ros.org/Robots

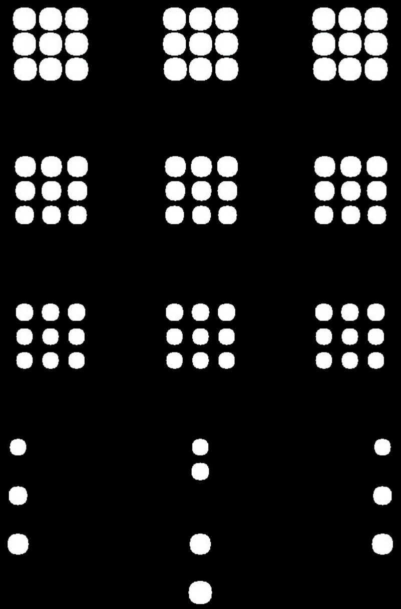
Outlook cont.

- ROS **2.0** coming up next year

How I Learned to Stop Reinventing and Love the Wheels

Thank you for your attention.

Questions?



andreas.bihlmaier@kit.edu

Used ROS for projects? Want to use it? Simply became interested? Talk to me.