HAW HAMBURG

EINE EINFÜHRUNG IN 

Robot Operating System

Henri Burau & Franek Stark

HAW HAMBURG

INHALT

WAS IST ROS?

Robot Operating System

ROS 1

ROS 2

HENRI BURAU
FRANEK STARK

PLATTFORMEN FÜR ROS

ROS Melodic

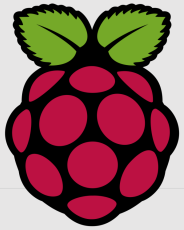
Ubuntu Artful amd64
Bionic amd64 armhf arm64

Debian Stretch amd64 arm64

ROS Kinetic

Ubuntu Wily amd64 i386
Xenial amd64 i386 armhf arm64

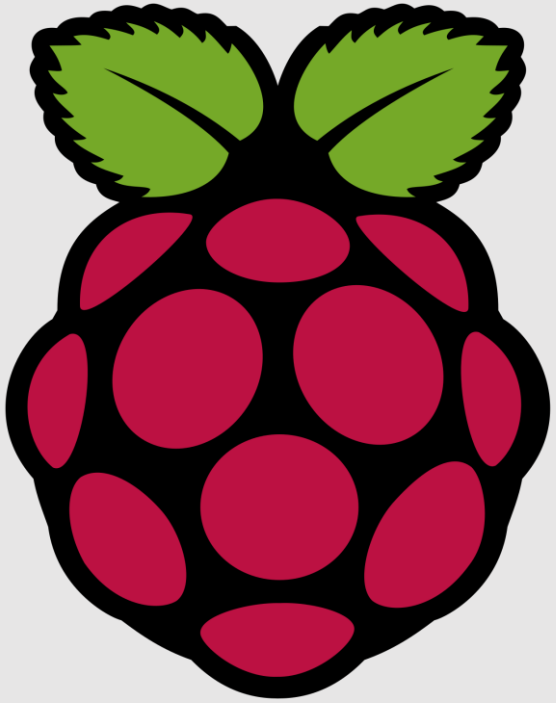
Debian Jessie amd64 arm64



Board: **armhf**
Raspbian: **Debian Stretch**

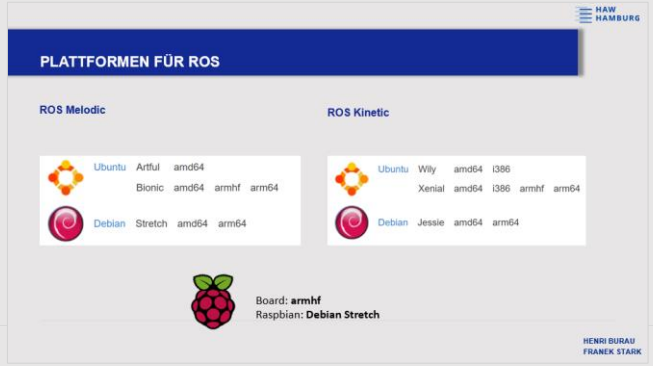
PLATTFORMEN FÜR ROS

ROS auf dem Raspberry Pi



ubuntu MATE

https://ubuntu-mate.org/raspberry-pi/



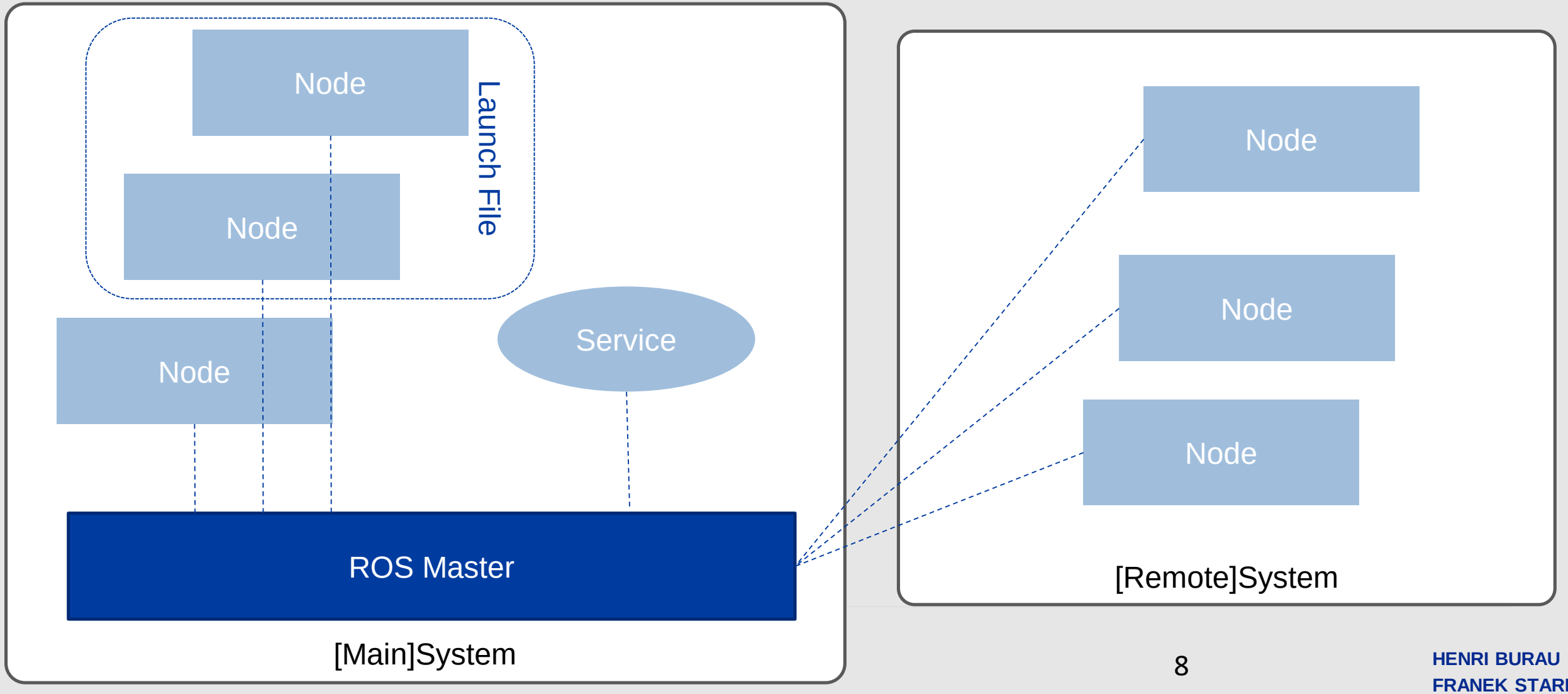
HENRI BURAU
FRANEK STARK

VORTEILE VON ROS

BEISPIEL

Danke an Herrn Lehmann!

ÜBERBLICK ÜBER ROS



HENRI BURAU
FRANEK STARK

WORKSPACE ERSTELLEN

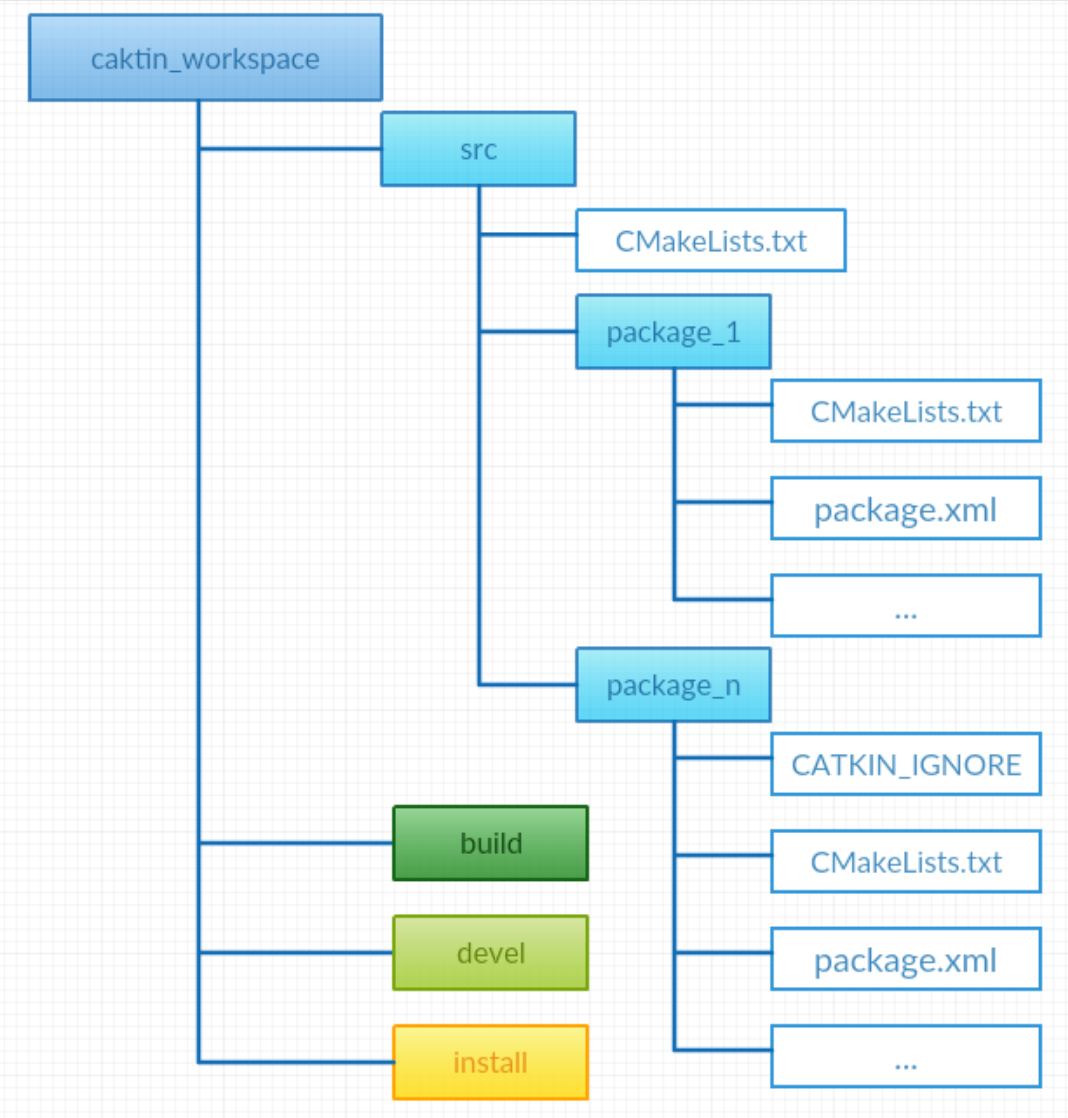
- **Aufgabe: Den eigenen catkin_workspace erstellen**

```
$ source /opt/ros/melodic/setup.bash

$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make

$ source devel/setup.bash
```

HENRI BURAU
FRANEK STARK

WORKSPACES

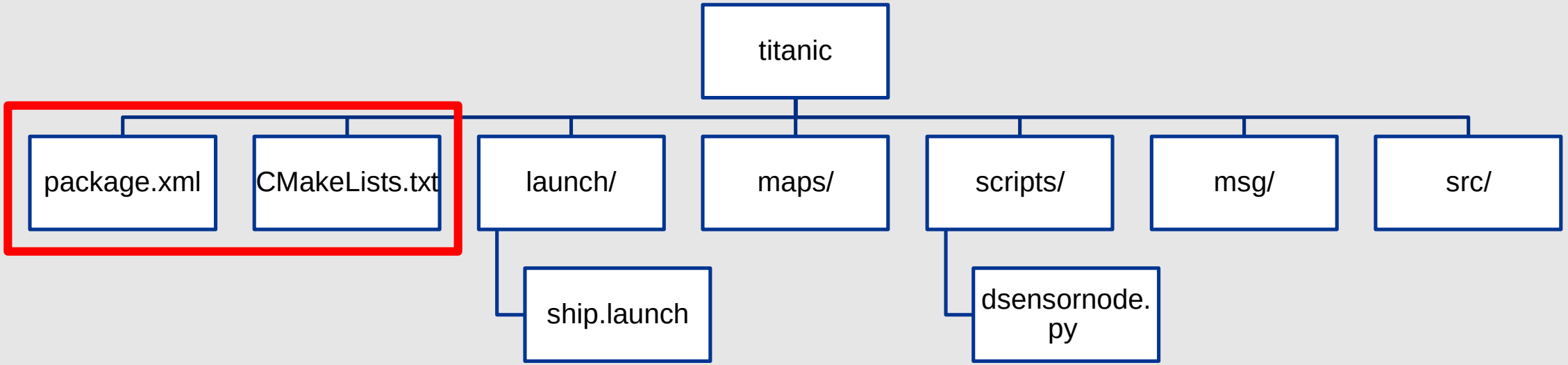
catkin




```
cd /path/to/catkin_ws

catkin_make

source devel/setup.bash
```

PACKAGES

Struktur



PACKAGES

Manifest

```
<?xml version="1.0"?>
<package format="2">
  <name>titanic</name>
  <version>1.0.0</version>
  <description>Package for the semester project "titanic" at the university of applied sciences Hamburg</description>

  <maintainer email="henri.burau@haw-hamburg.de"> Henri Burau</maintainer>
  <maintainer email="franek.stark@haw-hamburg.de"> Franek Stark</maintainer>

  <license>MIT</license>

  <url type="website">https://autonomesysteme.informatik.haw-hamburg.de/page/platform/nhawigatora/</url>
  <url type="repository">https://gitlab.informatik.haw-hamburg.de/miniatur-wunderland/titanic.git</url>
  <url type="bugtracker">https://gitlab.informatik.haw-hamburg.de/miniatur-wunderland/titanic/issues</url>

  <author email="henri.burau@haw-hamburg.de"> Henri Burau</author>
  <author email="maximilian.mang@haw-hamburg.de"> Maximilian Mang</author>
  <author email="felix.naumann@haw-hamburg.de"> Felix Naumann</author>
  <author email="thorben.schnirpel@haw-hamburg.de"> Thorben Schnirpel</author>
  <author email="franek.stark@haw-hamburg.de"> Franek Stark</author>

  <build_depend>message_generation</build_depend>
  <exec_depend>message_runtime</exec_depend>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

</package>
```

PACKAGES

CMake Build File

```
cmake_minimum_required(VERSION 2.8.3)
project(titanic)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  genmsg
  tf2
  tf2_ros
)

add_message_files(
  FILES
  MotorThrottle.msg
  ServoRad.msg
)

generate_messages(
   DEPENDENCIES
   std_msgs
)

include_directories(
  ${catkin_INCLUDE_DIRS}
)

add_executable(motor_controller_node src/motor_controller/motor_controller_node.cpp)
add_executable(imu_orientation_tf2_boradcaster src/transformation/tf_imu_orientation_transform.cpp)

target_link_libraries(imu_orientation_tf2_boradcaster
${catkin_LIBRARIES}
)

add_library(pca9685 src/pca9685/PCA9685.cpp)
target_link_libraries(motor_controller_node pca9685)

find_library(WIRINGPI_LIBRARY wiringPi /home/pi/wiringPi)

target_link_libraries(motor_controller_node ${catkin_LIBRARIES} ${WIRINGPI_LIBRARY})
add_dependencies(motor_controller_node wiringPi)
```

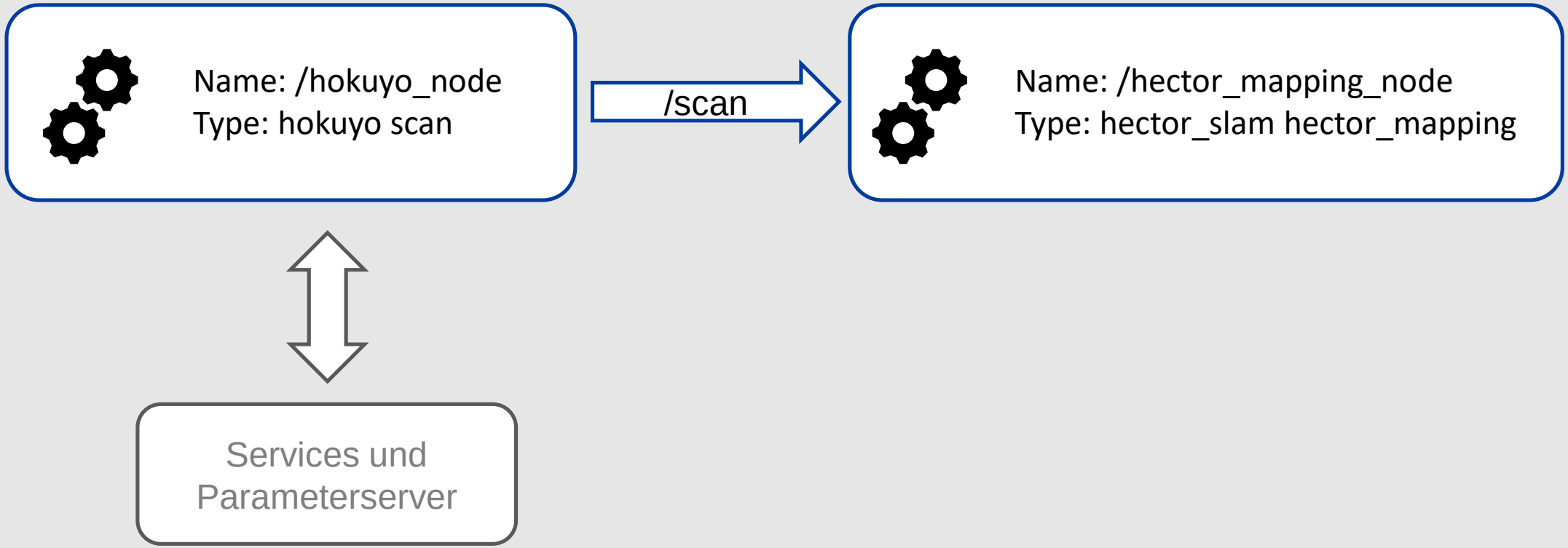
PACKAGE ERSTELLEN

- **Aufgabe: Ein eigenes package erstellen und den Author-Tag im Manifest ändern**

```
Package erstellen
$ catkin_create_pkg <package_name> std_msgs rospy roscpp sensor_msgs

Workspace builden
$ cd /path/to/catkin_ws/src
$ cd ..
$ catkin_make
$ source devel/setup.bash
```

NODES

Node Konzept



NODES

Python library - rospy

```
#! /usr/bin/env python

import rospy

def example():
    rospy.init_node('example')
    rospy.loginfo('Hello World')
    rospy.spin()

if __name__ == '__main__':
    try:
        example()
    except rospy.ROSInterruptException:
        pass
```

C++ library - roscpp

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

int main(int argc, char **argv){

ros::init(argc, argv,
"hello_world_cpp");
ros::NodeHandle nodeHandle;
ROS_INFO("Hello from Cpp-Node!");
ros::spin();

}
```

NODES

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(example)

## Find catkin and any catkin packages
find_package(catkin REQUIRED COMPONENTS roscpp rospy)

## Declare a catkin package
catkin_package()


include_directories(include ${catkin_INCLUDE_DIRS})

##For 'Hello_world.cpp'
add_executable(hello_world_cpp src/hello_world.cpp)
target_link_libraries(hello_world_cpp ${catkin_LIBRARIES})
```

EIN PUBLISHER-NODE

- Aufgabe: Ein „Temperatur-Sensor Node (mit Zufallswerten), welcher Temperaturen published“

- Name des Nodes: *temperature_sensor_<Vorname>*
- *Topic-Name: <Vorname>/temperature*
- *Topic-Type: sensor_msgs/Temperature*

```
# Single temperature reading.

Header header           # timestamp is the time the temperature was measured
                        # frame_id is the location of the temperature reading

float64 temperature     # Measurement of the Temperature in Degrees Celsius

float64 variance        # 0 is interpreted as variance unknown
```

HENRI BURAU
FRANEK STARK

CODE-HILFE

Python

```
import rospy
from sensor_msgs.msg import Temperature

# Publisher erzeugen
publisher = Rospy.Publisher('topic',
message_type, queue_size=100)

# Message verschicken
Publisher.publish(msg)

# Ros-Loop
rate = rospy.Rate(frequenz)
while not rospy.is_shutdown():
    …
    rate.sleep()
```

C++

```
#include "sensor_msgs/Temperature.h"
//Publisher erzeugen
ros::Publisher p =
nodeHandle.advertise<msgType>(„topic-Name",
queue-Size);

//Message verschicken
p.publish(message);

//Ros-Loop
ros::Rate r(Frequenz);
while(ros::ok()){

    r.sleep();
}
```

HENRI BURAU
FRANEK STARK

EIN PUBLISHER-SUBSCRIBER-NODE

- Aufgabe: Ein „Temperatur-Monitor Node, welcher den gleitenden Mittelwert eines temperatur-topics berechnet.

- Name des Nodes: *temperature_monitor_<Vorname>*
- *Topic-Name: <Vorname>/temperature_avarage*
- *Topic-Type: sensor_msgs/Temperature*

HENRI BURAU
FRANEK STARK

CODE-HILFE II

Python

```
# Subscriber erzeugen
rospy.Subscriber('topic', message_type,
callback_func)

# CallBackHandler
def callback_func(msg):
```

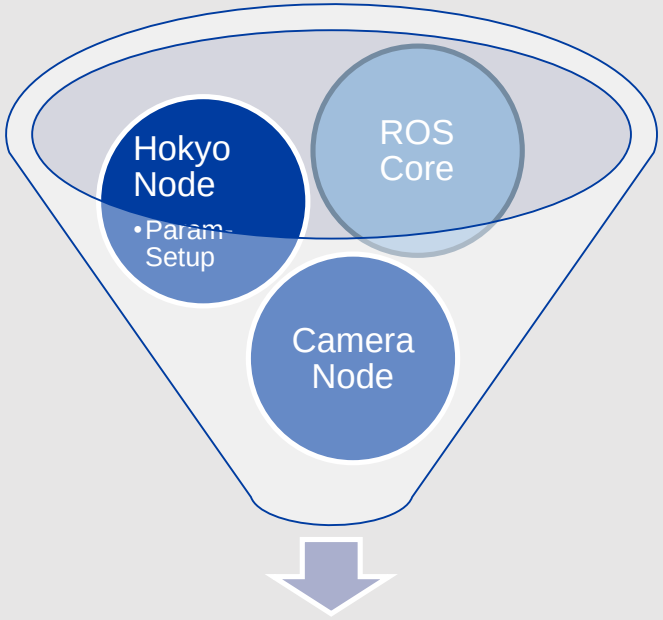
C++

```
//Subscriber erzeugen
ros::Subscriber s =
nodeHandle.subscribe(„Topic-Name", queue-Size,
callback_func);

//CallBackHandler
void temp_callback(const MsgConstPtr& m) {

}
```

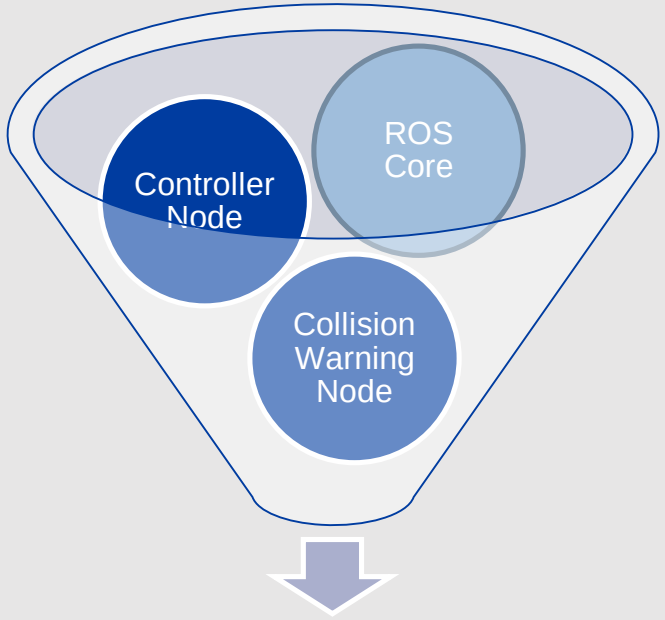
HENRI BURAU
FRANEK STARK

SETUP STARTEN

Launchfiles



sensors.launch



ship.launch

```
roslaunch titanic_package ship.launch
```

HENRI BURAU
FRANEK STARK

LAUNCHFILE

Syntax

```
<!--XML-->


<launch>
   <node name=„urg_node“ pkg=„hokuyo_urg“ type=„urg_node.py“ />
   <include file=„controll.launch“ />
</launch>
```

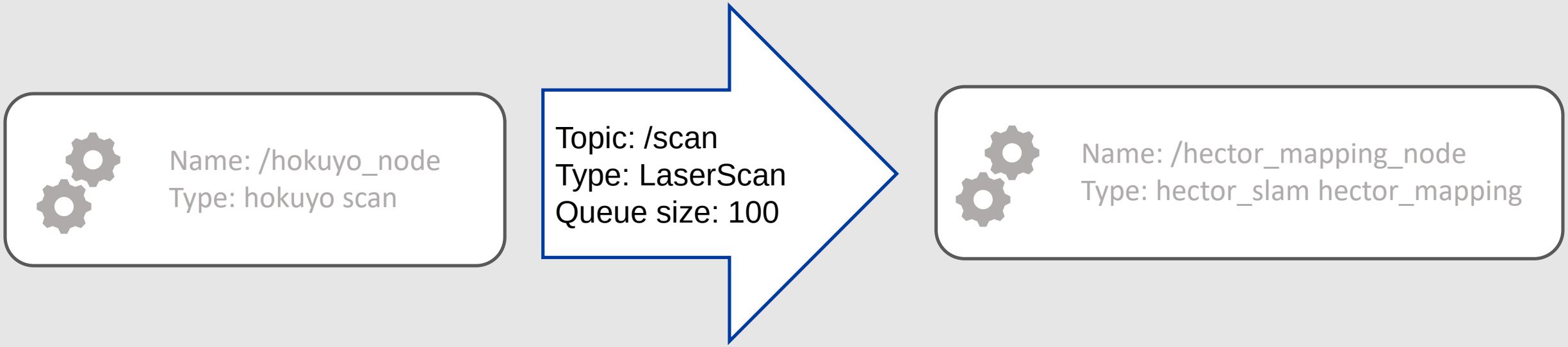
EIGENES LAUNCHFILE

- **Aufgabe:** Ein Launchfile für den temperature_node schreiben.

HENRI BURAU
FRANEK STARK

MESSAGES UND TOPICS

ROS Topics



MESSAGES UND TOPICS

ROS Messages

```
# This represents a 2-D grid map, in which each cell represents the probability of
# occupancy.

Header header

#MetaData for the map
MapMetaData info

# The map data, in row-major order, starting with (0,0).  Occupancy
# probabilities are in the range [0,100].  Unknown is -1.
int8[] data
```

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.sec: seconds (stamp_secs) since epoch (in Python the variable is called 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (in Python the variable is called 'nsecs')
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
string frame_id
```

```
# This hold basic information about the characterists of the OccupancyGrid

# The time at which the map was loaded
time map_load_time
# The map resolution [m/cell]
float32 resolution
# Map width [cells]
uint32 width
# Map height [cells]
uint32 height
# The origin of the map [m, m, rad].  This is the real-world pose of the
# cell (0,0) in the map.
geometry_msgs/Pose origin
```

ROS SERVICE

Client Server Model – Remote Procedure Call

ROSBAG

Aufnehmen, Speichern und Abspielen von Messages

```
//rosbag record [topic...]

//rosbag play [--loop] [filename]
```

HAW HAMBURG

EINEN ROSBAG AUFNEHMEN

- **Aufgabe:** Aufnahme von dem Topic <dein_name>/temperature

HENRI BURAU
FRANEK STARK

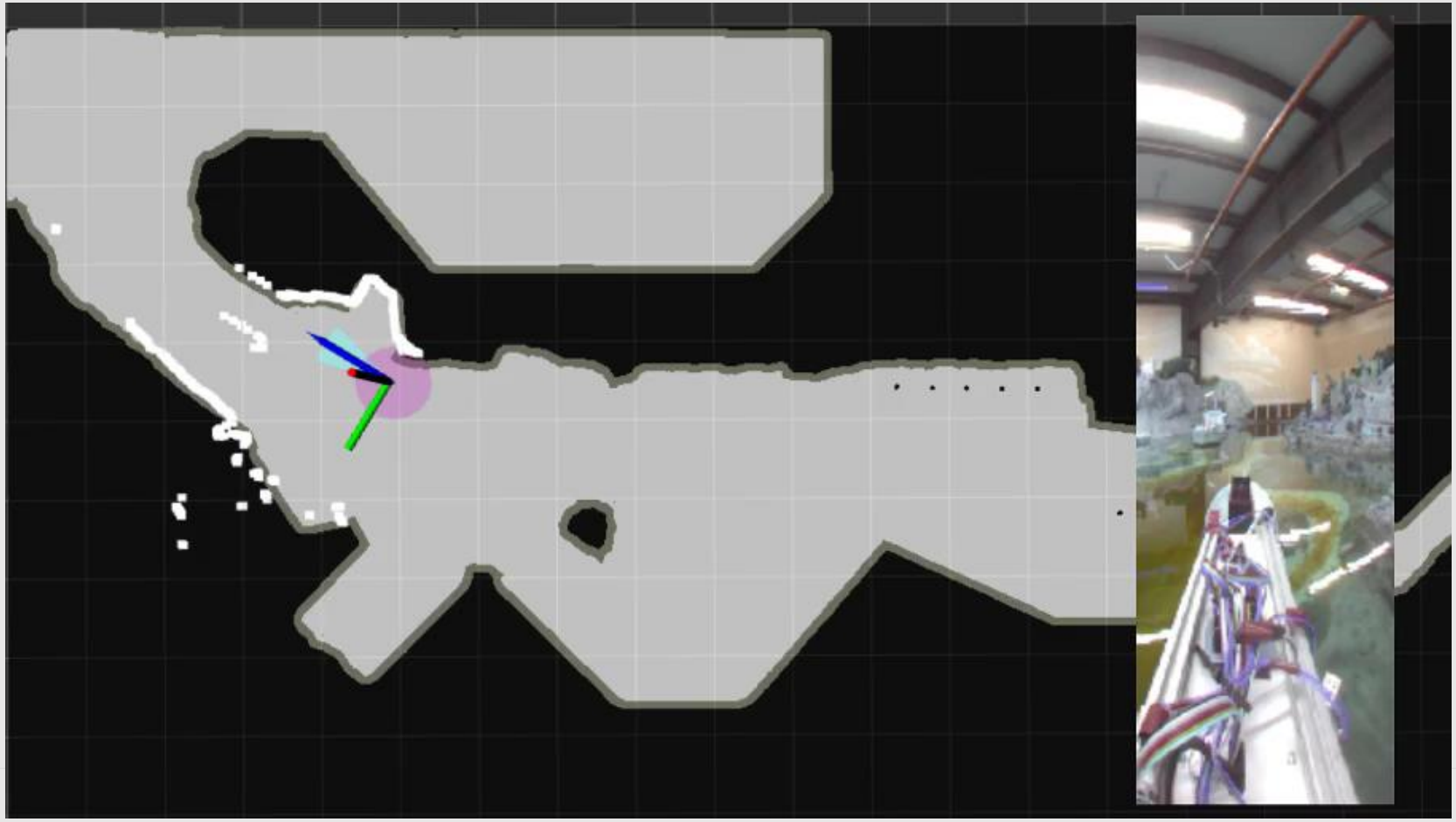
TRANSFORMATIONEN

Statische Transformation in Launchfile

```
<launch>

<node pkg="tf2_ros" type="static_transform_publisher"
name="link1_broadcaster" args="x y z yaw pitch roll link1_parent
link1" />

</launch>
```

ROS-TOOLS

rviz und rqt



HENRI BURAU
FRANEK STARK

VISUALISIEREN IN RVIZ

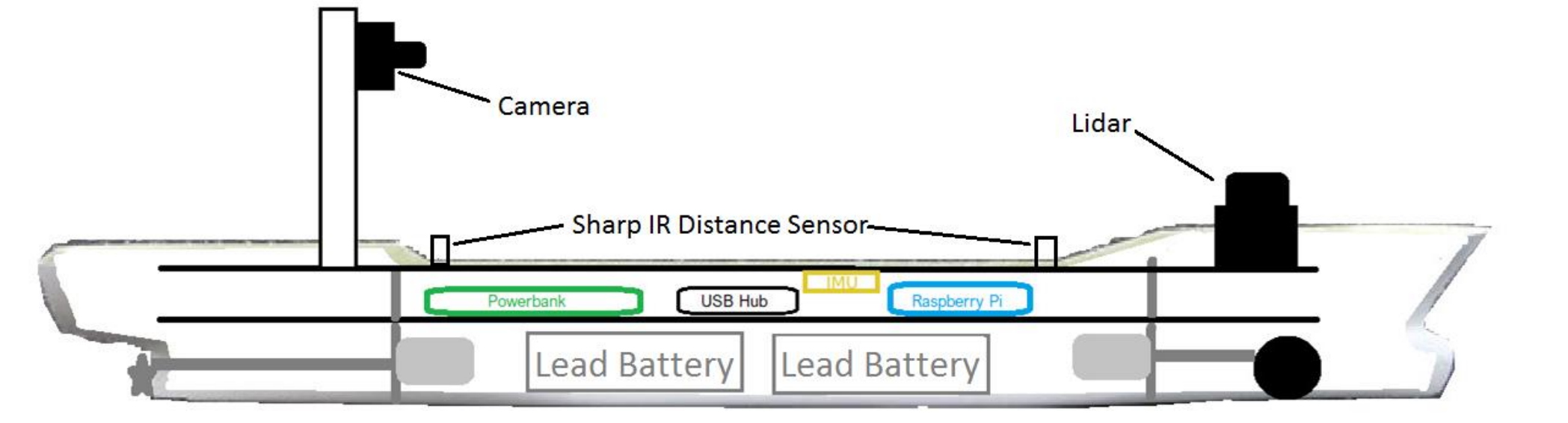
- **Aufgabe:** Topic <dein_name>/temperature in Rviz darstellen

  Tipp: Frame-ID und Topic ist nicht das dasselbe

HENRI BURAU
FRANEK STARK

TRANSFORMATIONEN

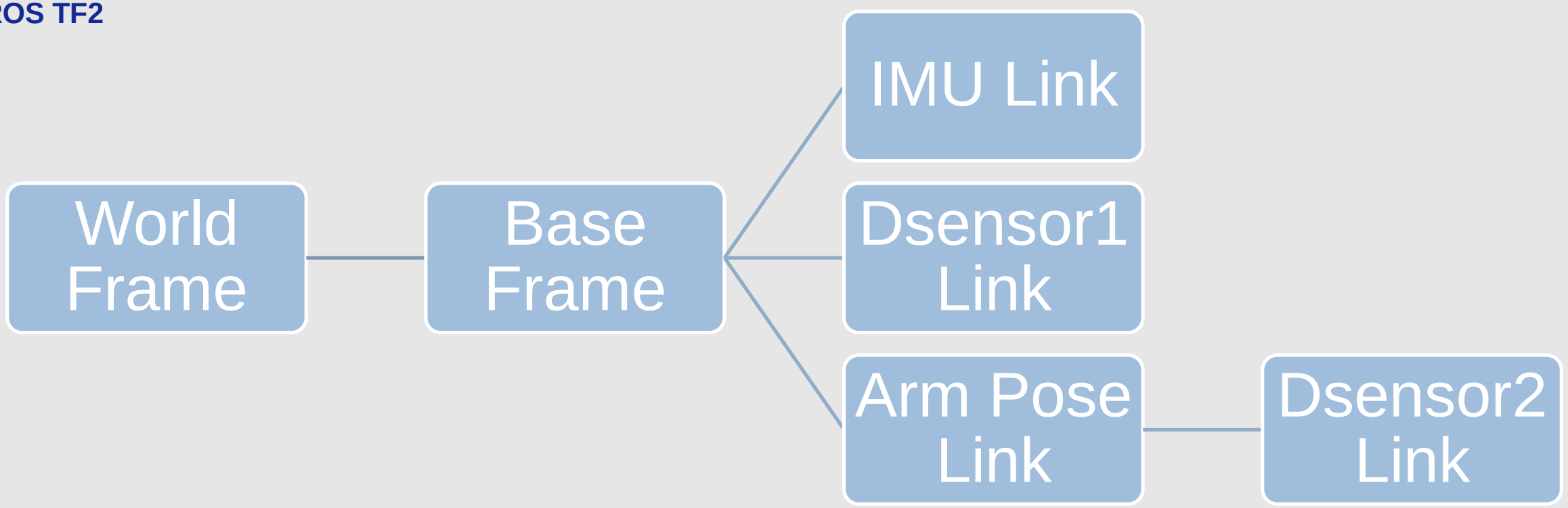
ROS TF



HENRI BURAU
FRANEK STARK

TRANSFORMATIONEN

ROS TF2




```
x y z yaw pitch roll frame_id child_frame_id
```

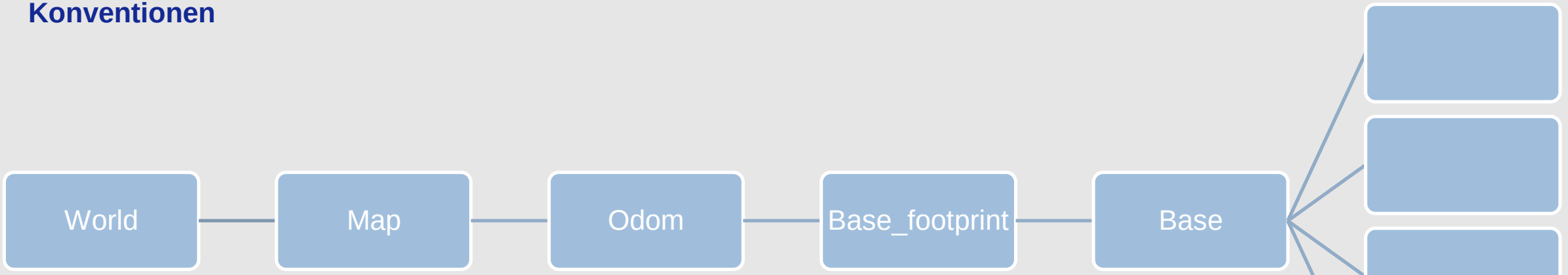
NODE FRAMESICHER MACHEN

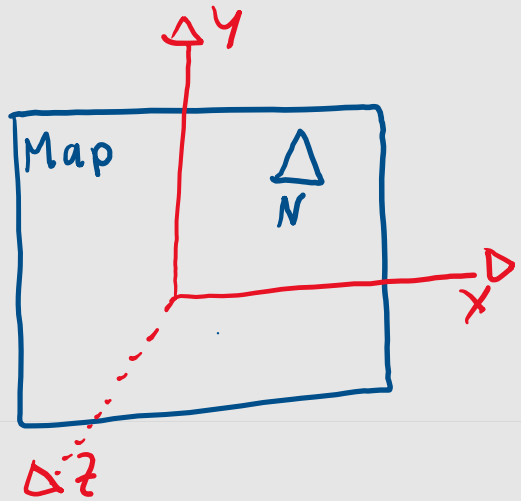
- **Aufgabe:** Im temperature_node die versendeten Nachrichten mit einer FrameID versehen. FrameID: <*Vorname*>_*link*
- **Aufgabe:** Einen statische-Transformations-Broadcaster in das Launchfile integrieren. Transformation: map → <*Vorname*>_*link*

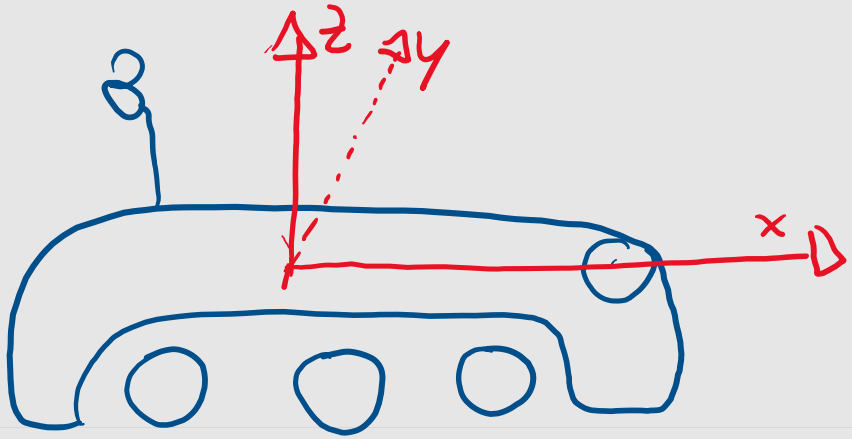
HENRI BURAU
FRANEK STARK

TRANSFORMATIONEN

Konventionen



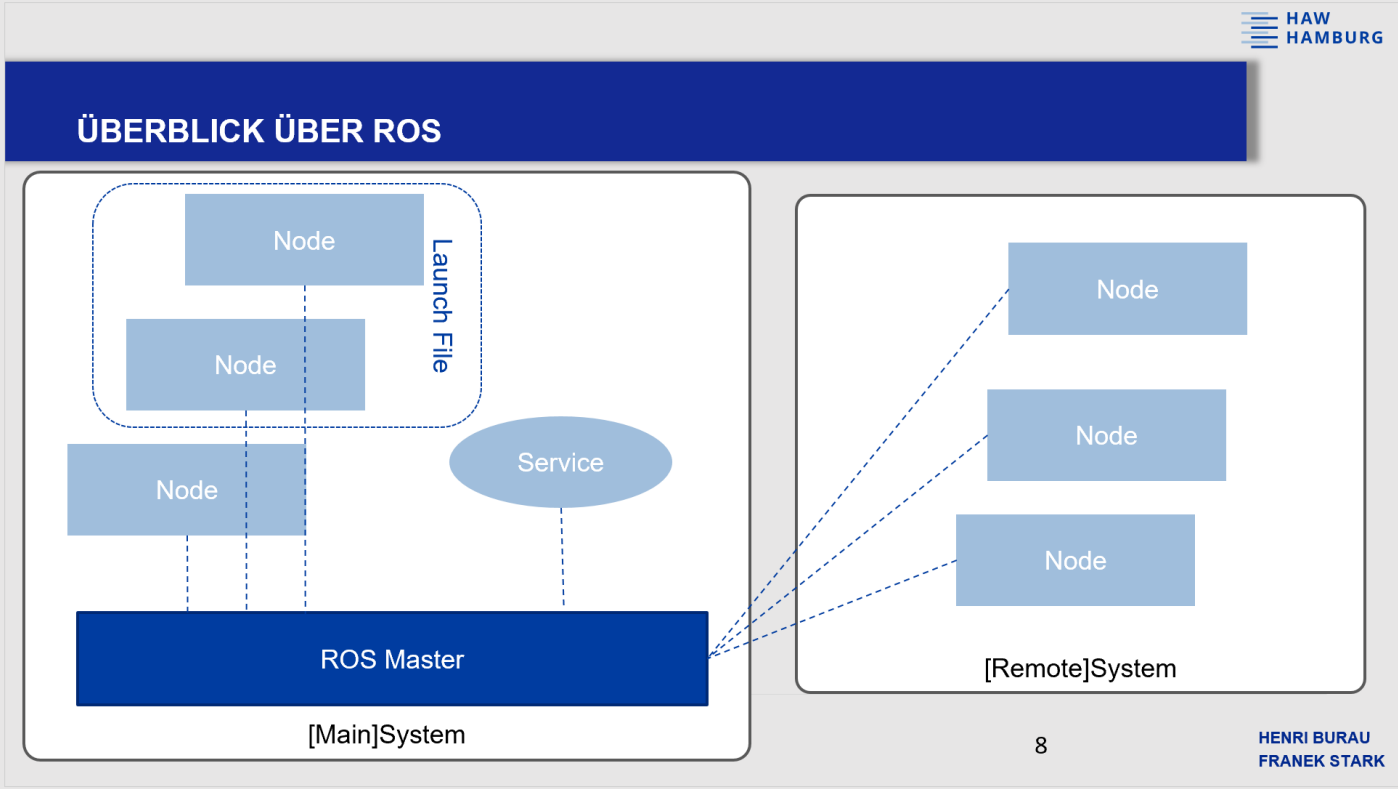




HENRI BURAU
FRANEK STARK

ROS IM VERTEILTEN SYSTEM

ROS Remote



ROS REMOTE

Vorbedingung:

Die Systeme können sich untereinander mittels Hostname pingen. → /etc/hosts

Konfiguration von ROS:

```
export ROS_MASTER_URI=http://laptop:11311
export ROS_HOSTNAME=<hostname>
```