Introduction to Robot Operating System [ROS]

Prof. Madhur Behl

LINK LAB

ROS Robot Operating System

F1 TENTH

Cavalier Autonomous Racing

University of Virginia

Acknowledgements for some of the slides and illustrations

Open Robotics

ROSCon

Autoware Foundation

Clearpath Robotics

ETHZ ROS tutorials

Announcements

- Access to Ubuntu 18.04 (Bionic)
- Install ROS Melodic – desktop-full

To programming using ROS you must setup a developer's PC:

- Ubuntu 18.04: `https://ubuntu.com/tutorials/tutorial-install-ubuntu-desktop#1-overview`
- ROS Melodic: `http://wiki.ros.org/melodic/Installation/Ubuntu`

Dual boot vs *Virtual Machine*: A simplest way to setup the developer's PC is installing Linux on a Virtual Machine. This is not suggested due to the high hardware resources needed by the simulation scenes used in the evaluation. Moreover, you can use it in the first part of the course to test initial robotic programming examples.

"Does your car have any idea why my car pulled it over?"

This lecture

- Why do we need ROS ?
- What is ROS ?
- ROS Overview and Introduction
- What is ROS2 ?

Localization and Mapping

Where am I ?

Scene Understanding

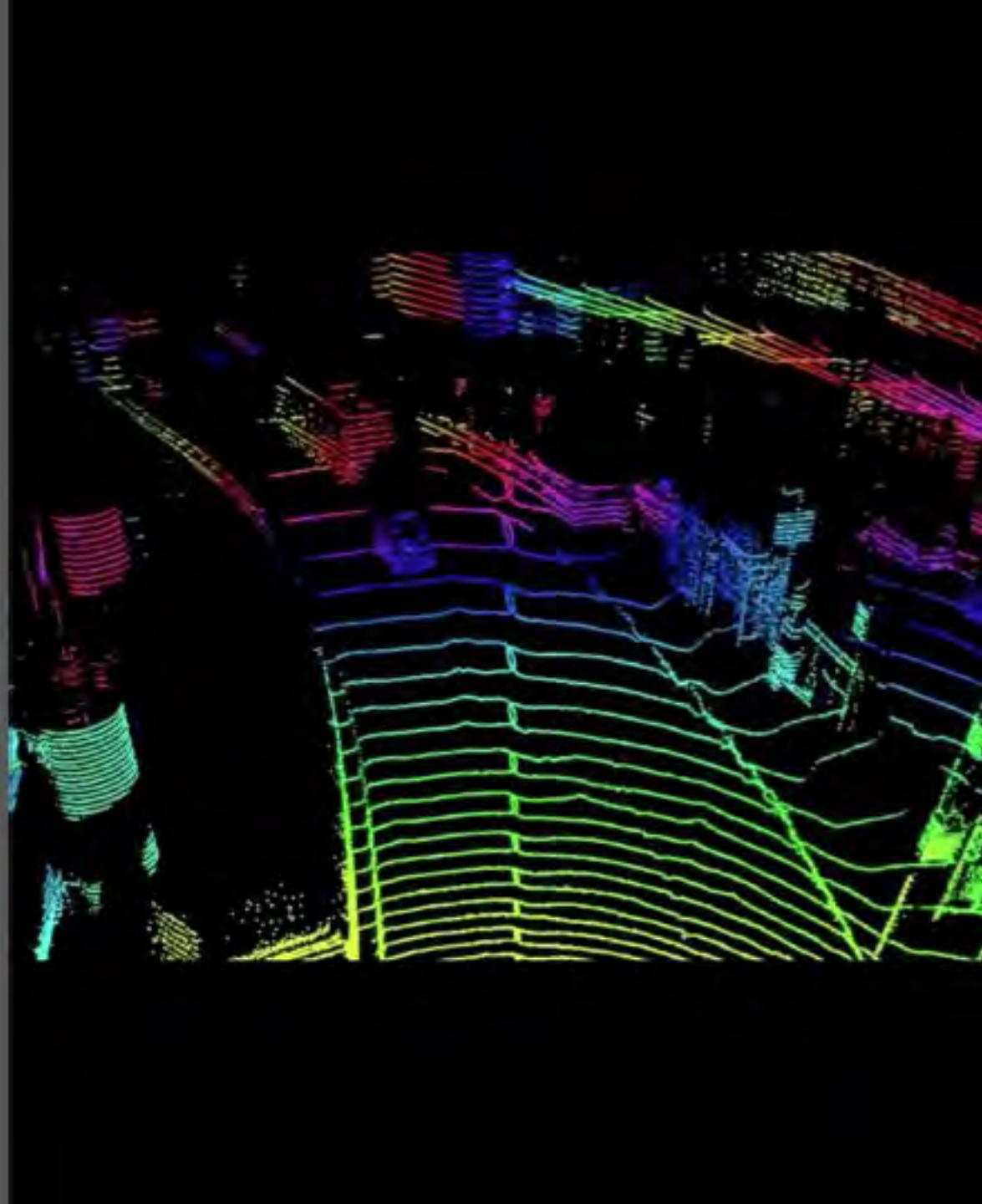
Where/who/what/why of
everyone/everything else ?

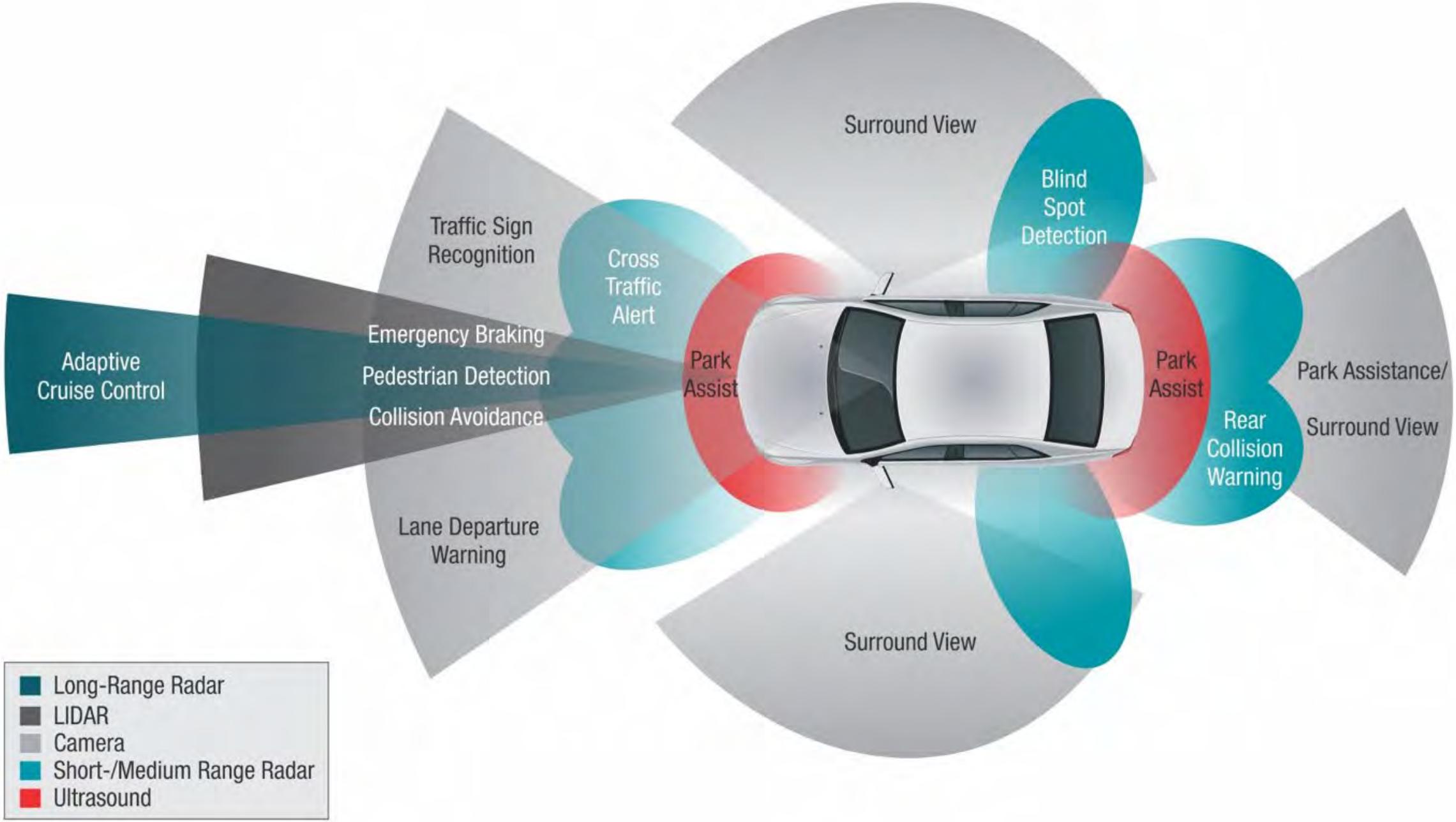
Trajectory Planning and Control

Where should I go next ?
How do I steer and accelerate ?

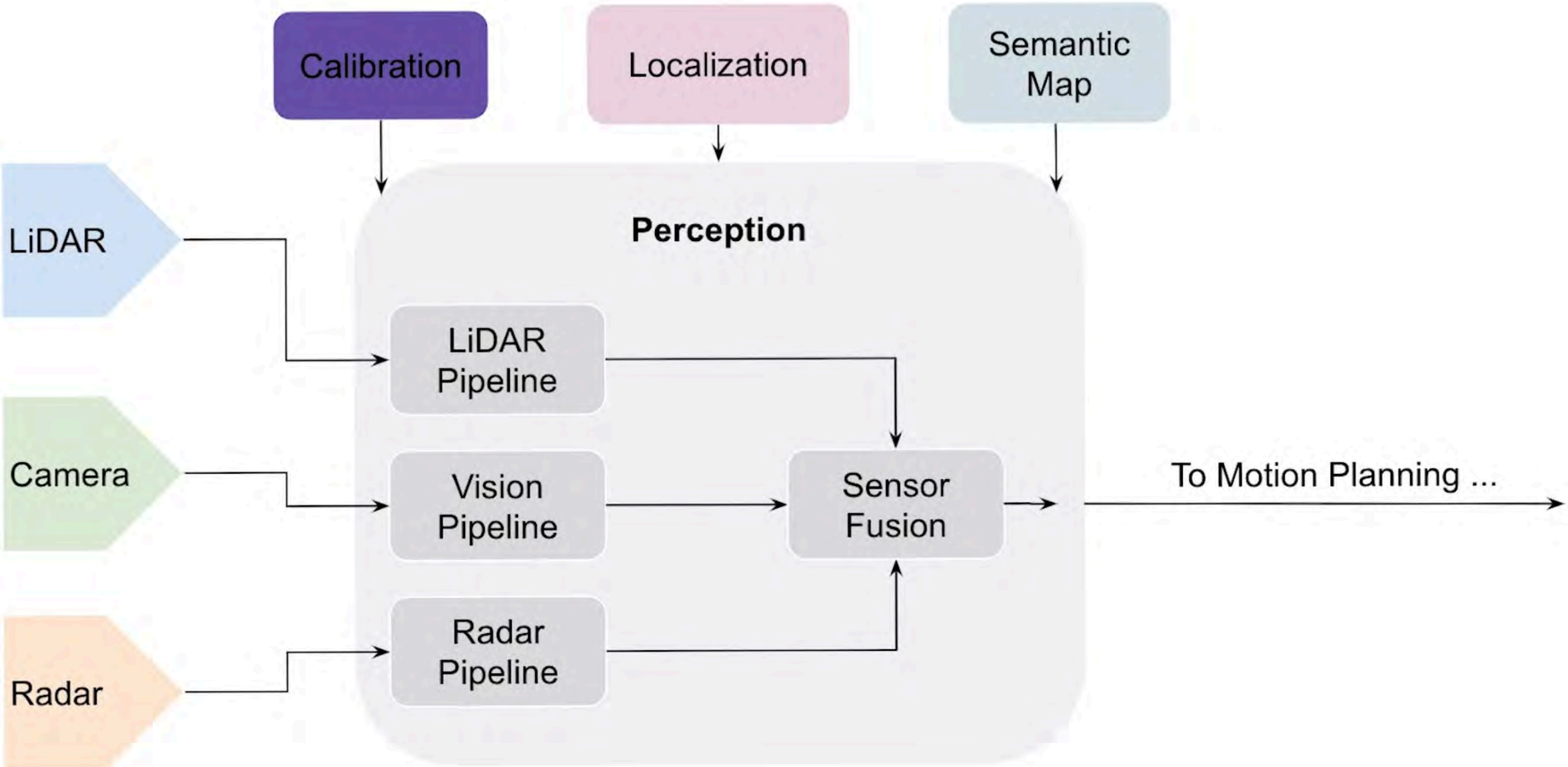
Human Interaction

How do I convey my intent to the
passenger and everyone else ?




Surround View
Blind Spot Detection
Traffic Sign Recognition
Cross Traffic Alert
Adaptive Cruise Control
Emergency Braking
Pedestrian Detection
Collision Avoidance
Park Assist
Park Assist
Park Assistance/
Surround View
Rear Collision Warning
Lane Departure Warning
Surround View
Long-Range Radar
LIDAR
Camera
Short-/Medium Range Radar
Ultrasound

Perception in AV Stack



CAR AUTOMATION SENSORS & DATA VOLUMES

Sensor type	Quantity	Data generated
Radar	4–6	0.1–15 Mbit/s
LIDAR	1–5	20–100 Mbit/s
Camera	6–12	500–3,500 Mbit/s
Ultrasonic	8–16	<0.01 Mbit/s
Vehicle motion, GNSS, IMU	-	<0.1 Mbit/s

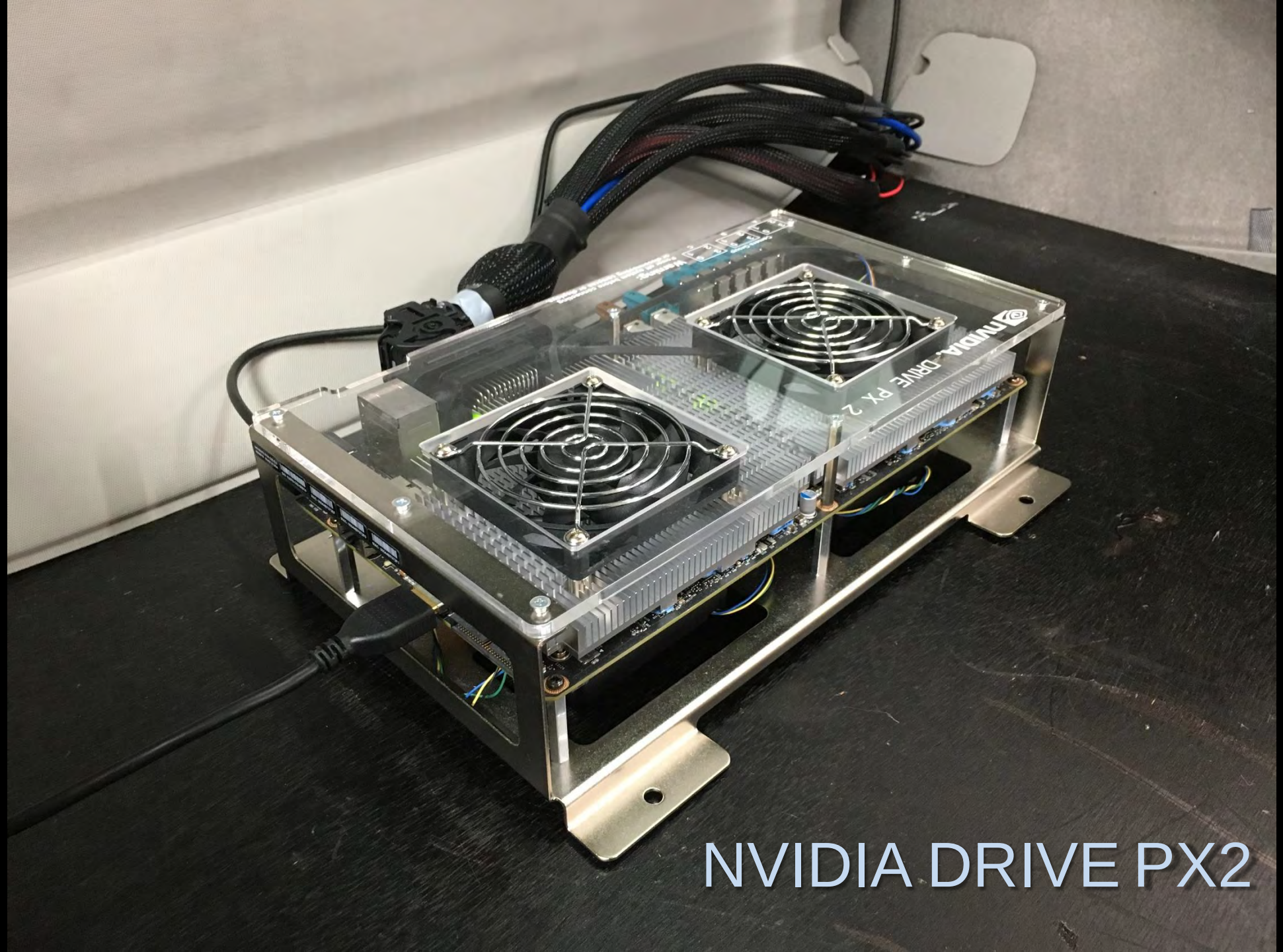
TOTAL ESTIMATED BANDWIDTH

3 Gbit/s (~1.4TB/h) to 40 Gbit/s (~19 TB/h)


Velodyne HDL-64e (3D LiDAR)
Point Grey Ladybug 5 (Camera)


Velodyne HDL-32e (3D LiDAR)
Velodyne VLP-16 (3D LiDAR)
IBEO LUX 8L (3D LiDAR)


JAVAD RTK-GNSS (GNSS/GPS)
Point Grey Grasshopper3 (Camera)


NVIDIA DRIVE PX2


ALPINE


NVIDIA DRIVE PX 2
Warning:

3D LiDARs

Cameras

GNSS/GPS

Computing Unit

AutoWare

CAN I/F

CAN

Control Box

Status Monitor

Emergency Button

Direct signals to ECUs bypassing CAN.

EPS ECU

Hybrid ECU

Skid Control ECU

Other ECUs

Torque Sensor

Gas Pedal Position Sensor

Shift Position Sensor

Brake Stroke Sensor

Oil Pressure Sensor

Reconnected wires.

Accessories.

Car interior monitors.

Sensor Integration

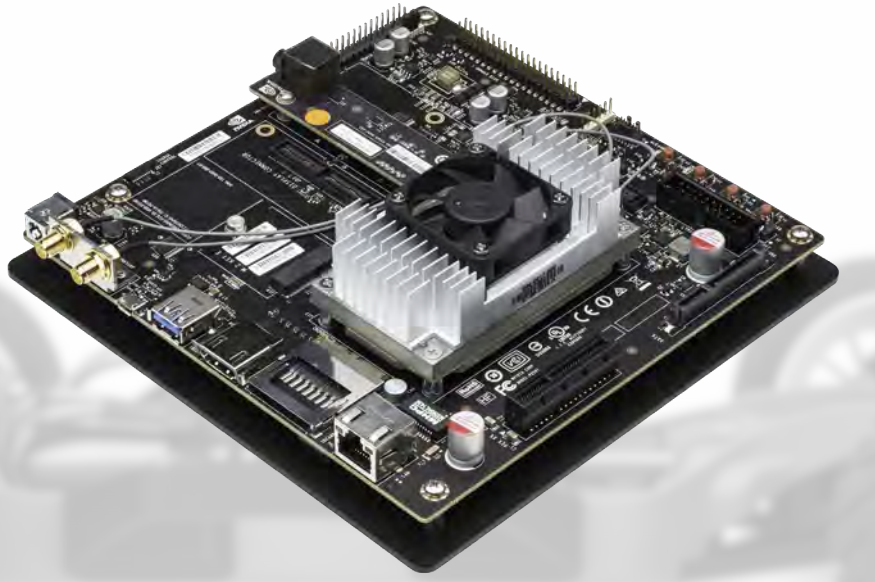
LiDAR

Camera

IMU

IR Depth Cameras

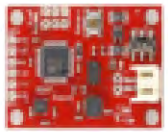
Wi-fi Telemetry



Onboard Computer

Motor Controller

LiDAR



Camera

IMU

Filters

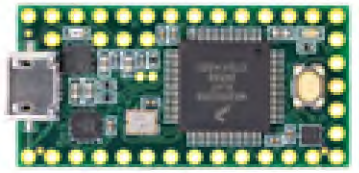
Map generation

Computer vision

Localization

Path planning

Steering



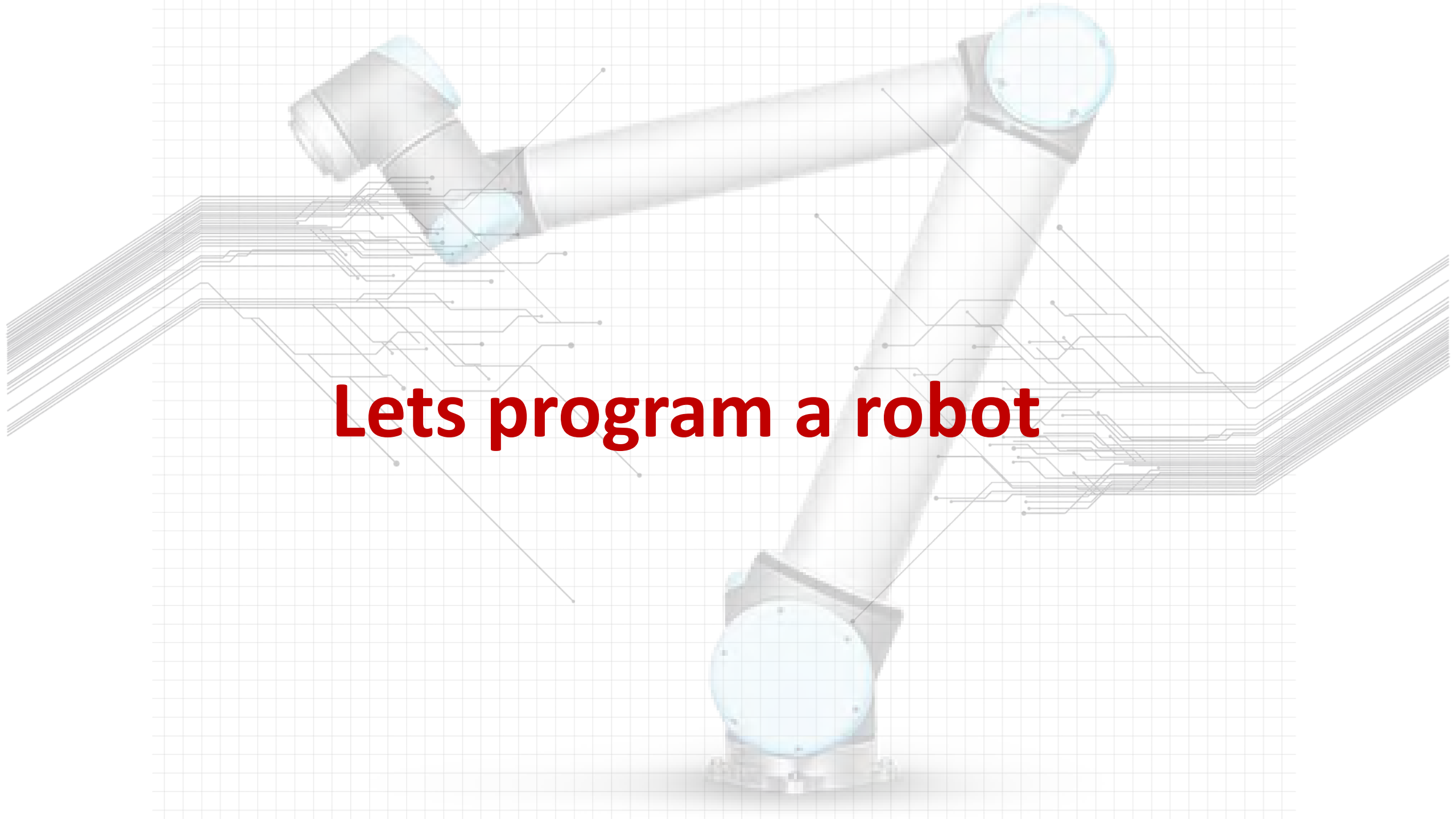
Motor Controller

Drive

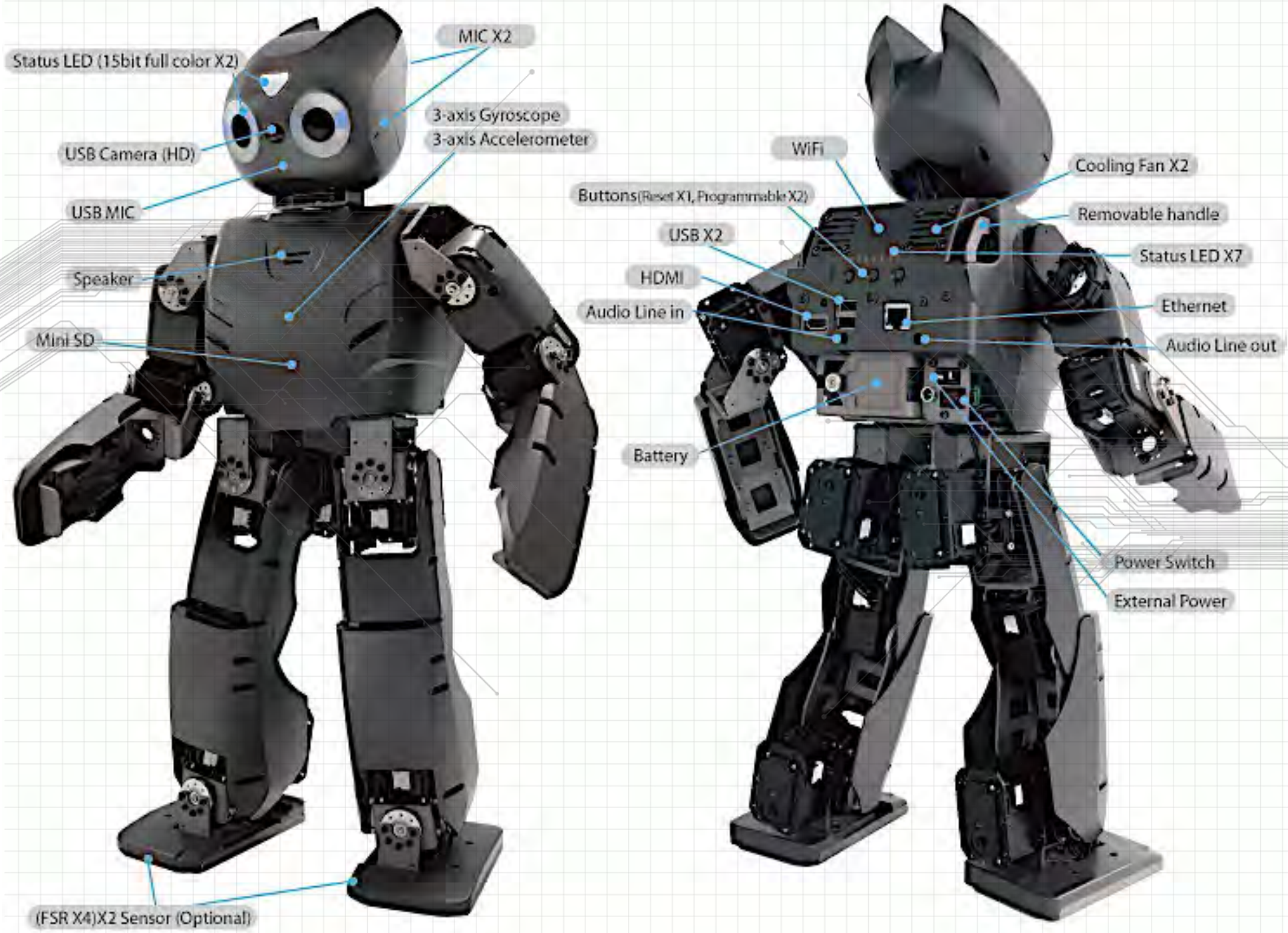
Perception

Planning

Control

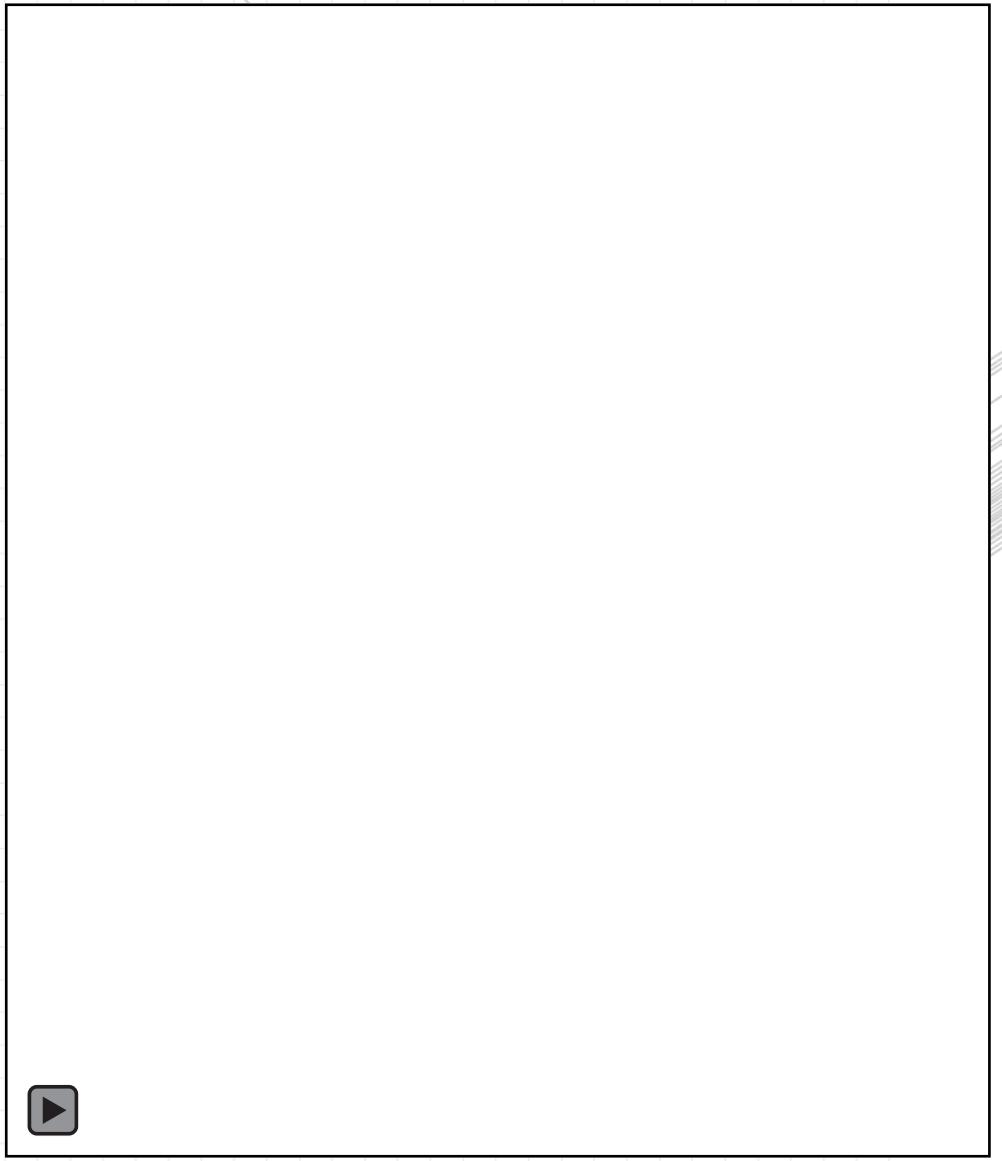


Lets program a robot


Status LED (15bit full color X2)
MIC X2
3-axis Gyroscope
3-axis Accelerometer
USB Camera (HD)
USB MIC
Speaker
Mini SD
(FSR X4)X2 Sensor (Optional)
WiFi
Cooling Fan X2
Buttons(Reset X1, Programmable X2)
Removable handle
USB X2
Status LED X7
HDMI
Ethernet
Audio Line in
Audio Line out
Battery
Power Switch
External Power

Monolithic, Sequential

```
while(1)
    read_from_rgbd_camera(...)
    detect_soccer_ball(...)
    update_ball_tracker(...)
    convert_rgbd_to_laser(...)
    read_odometry(...)
    update_localization(...)
    update_motion_plan(...)
    send_motion_commands(...)
```



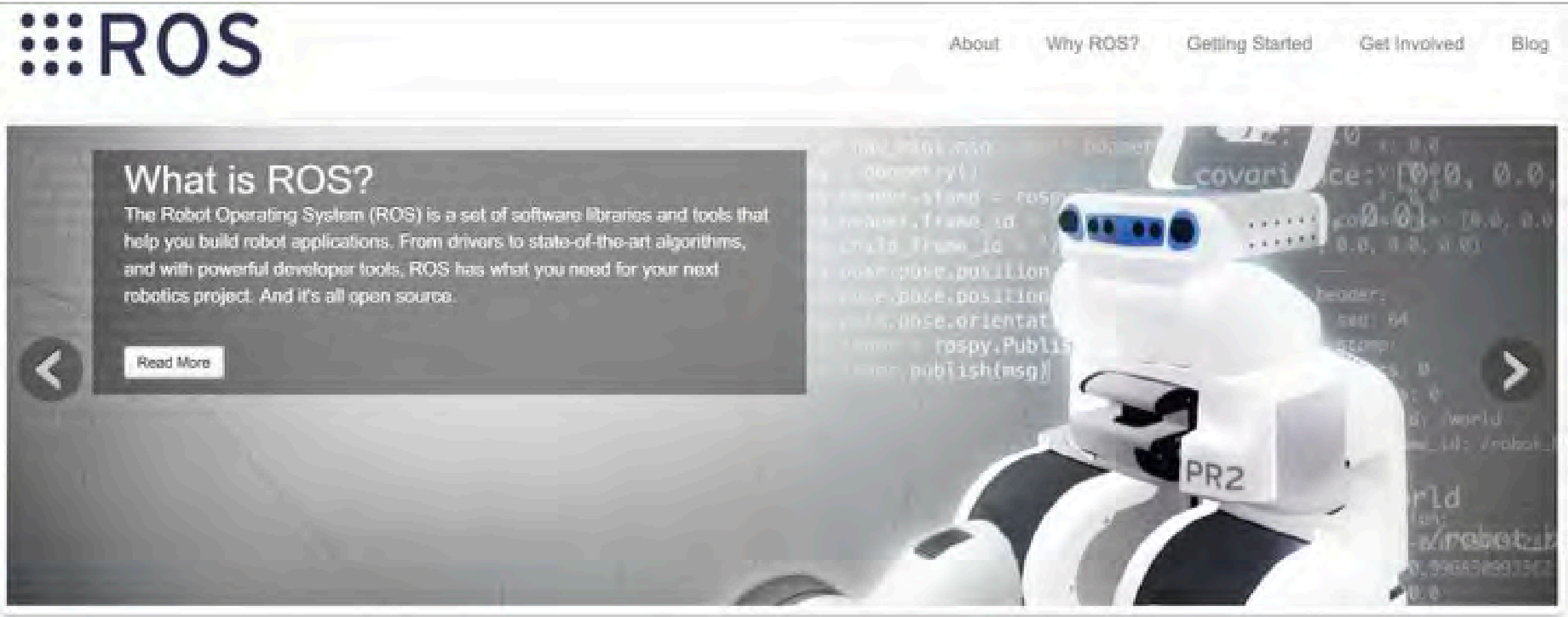
Challenges of Robotics Software Development*

- **Complexity**
  - Interconnected algorithms
  - Significant internal state
  - Concurrency
  - Multi-rate and asynchronicity
  - Multi-agent / distributed
- **Large-scale**
  - Distributed expertise
- **Resilience**
  - Fault isolation
- **Flexibility**
  - Changing configurations
  - Rapid prototyping
- **Difficult algorithms**
  - Real-time
  - Computation, bandwidth, & memory
- **Re-use**
  - Integration into other systems
  - Portability to other robot platforms

What a mess!

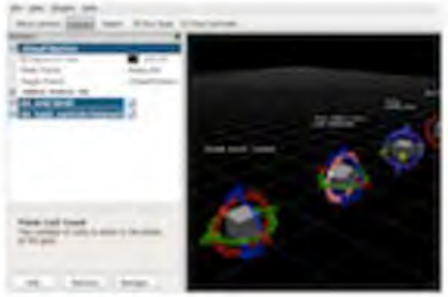
How can we deal with it ?

ROS: Robot Operating System



ROS: Robot Operating System

+



+

+

ros.org

Plumbing

- Process management
- Inter-process communication
- Device drivers

Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

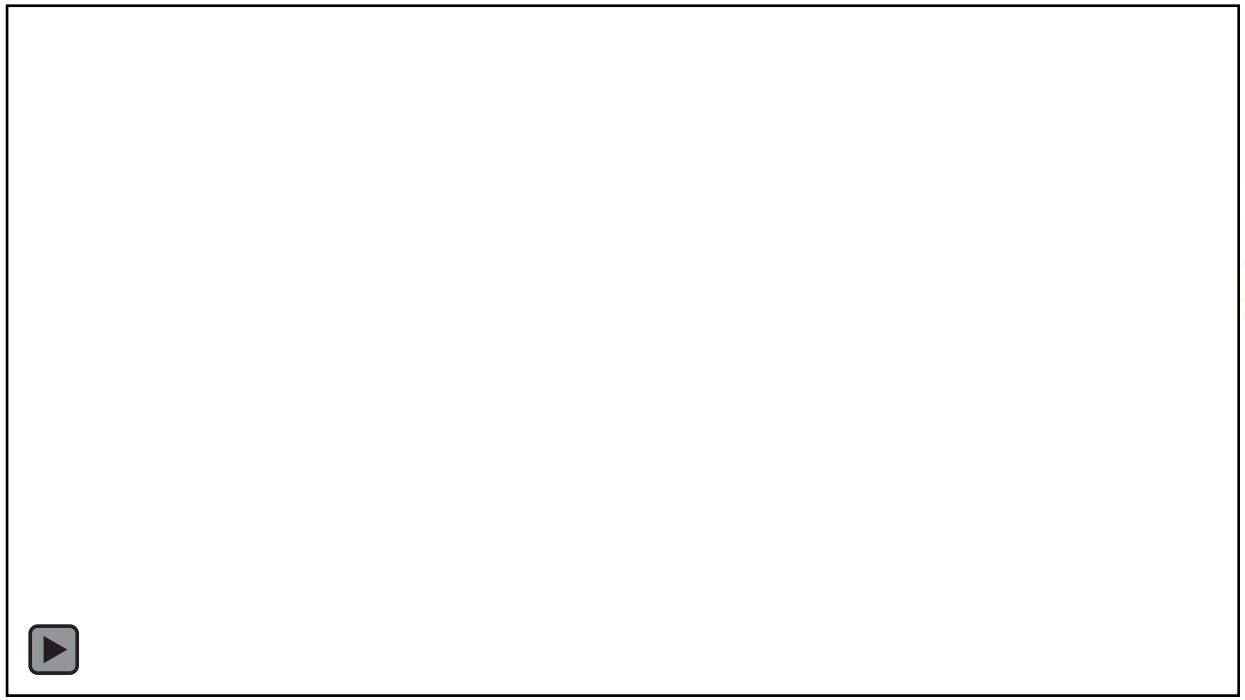
Capabilities

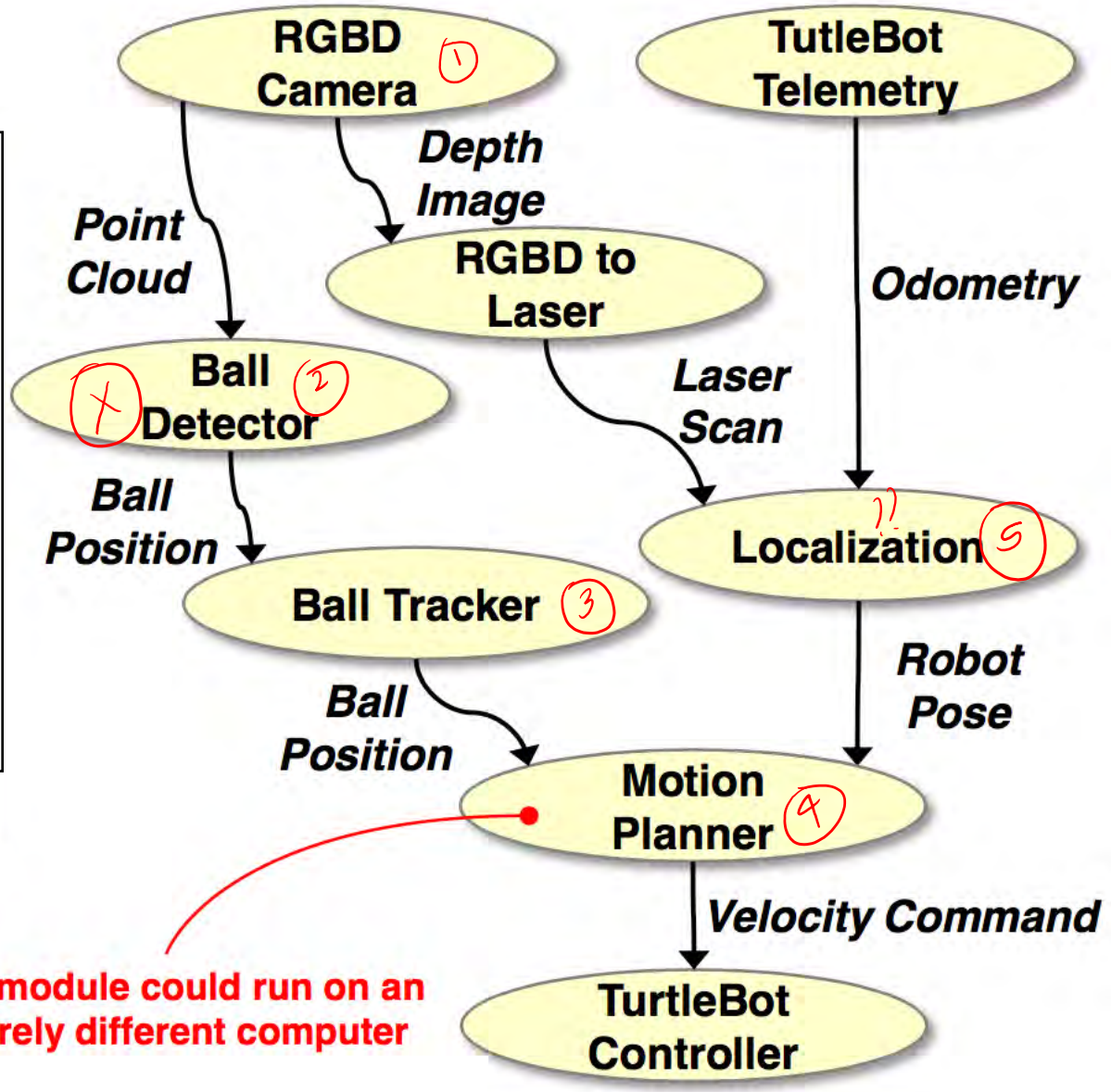
- Control
- Planning
- Perception
- Mapping
- Manipulation

Ecosystem

- Package organization
- Software distribution
- Documentation
- Tutorials

Modular, Parallel

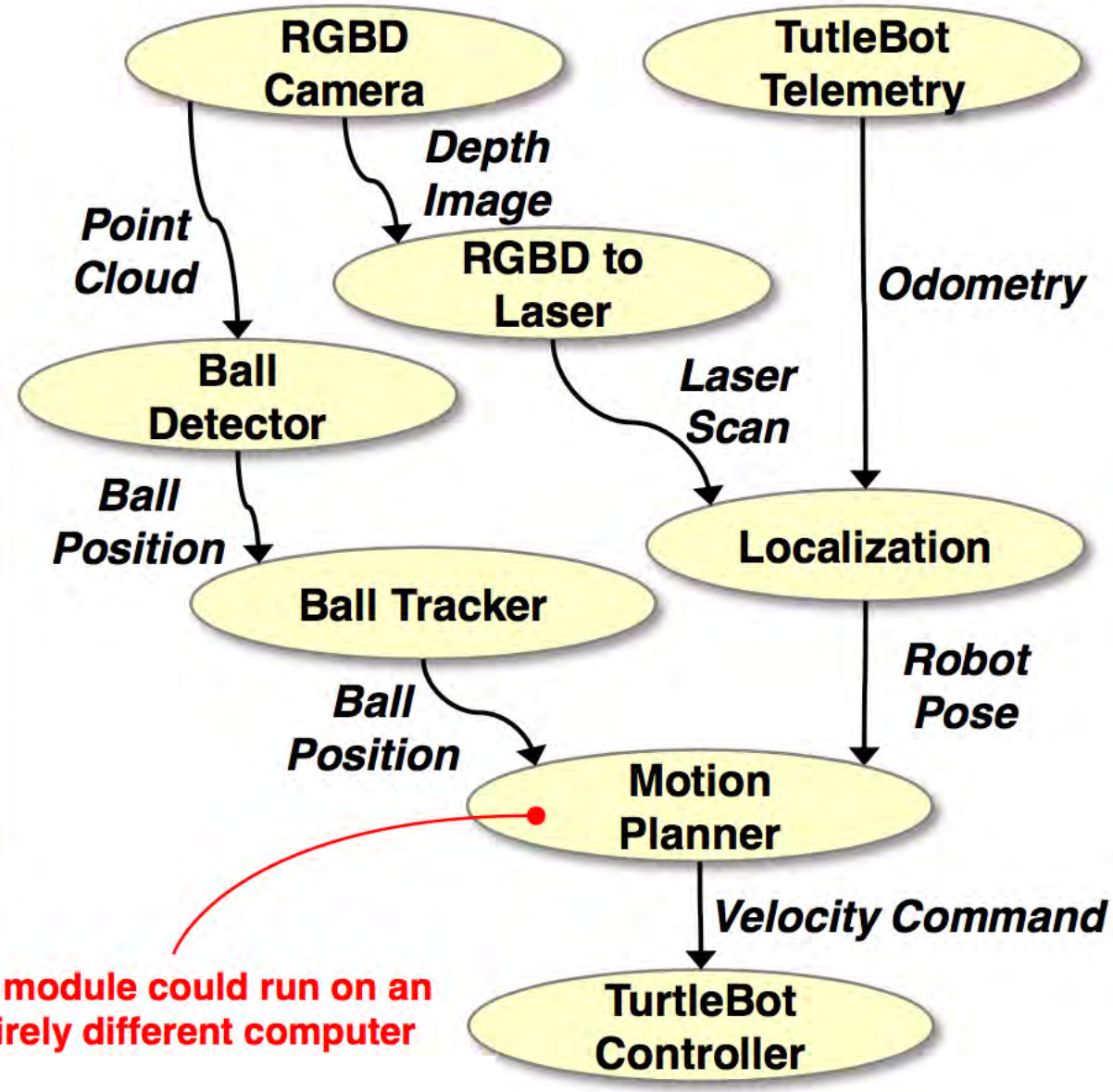




Monolithic, Sequential

```
read_from_rgbd_camera(...)
detect_soccer_ball(...)
update_ball_tracker(...)
convert_rgbd_to_laser(...)
read_odometry(...)
update_localization(...)
update_motion_plan(...)
send_motion_commands(...)
```

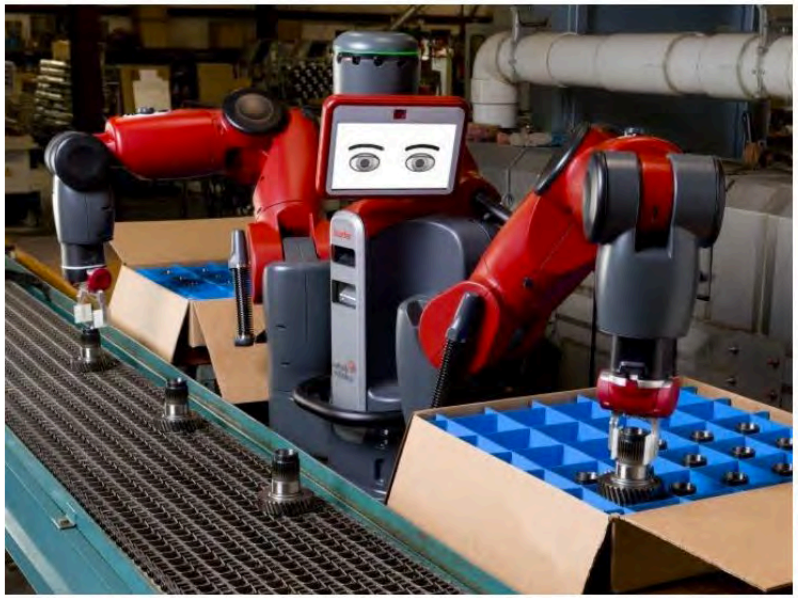
Modular, Parallel



ROS Philosophy

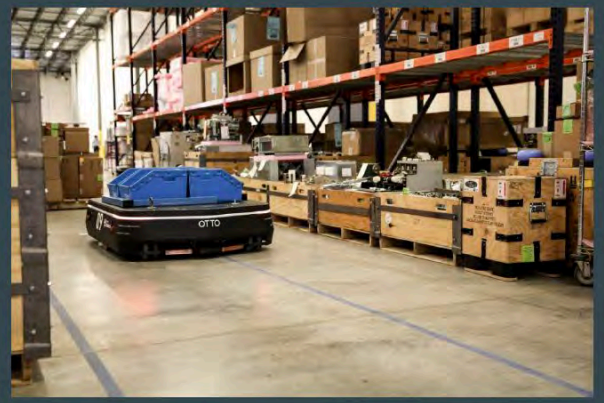
- **Peer to peer**
  Individual programs communicate over defined API (ROS *messages*, *services*, etc.).
- **Distributed**
  Programs can be run on multiple computers and communicate over the network.
- **Multi-lingual**
  ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).
- **Light-weight**
  Stand-alone libraries are wrapped around with a thin ROS layer.
- **Free and open-source**
  Most ROS software is open-source and free to use.

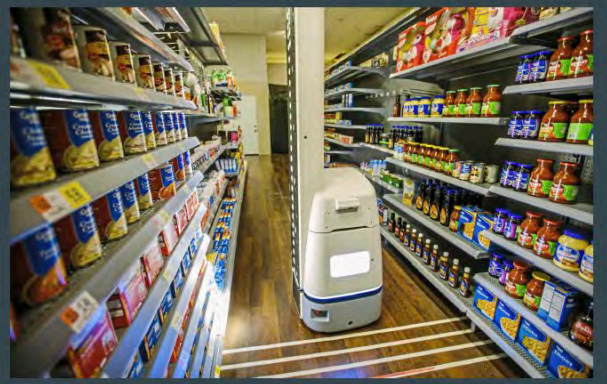
Landmark Robots

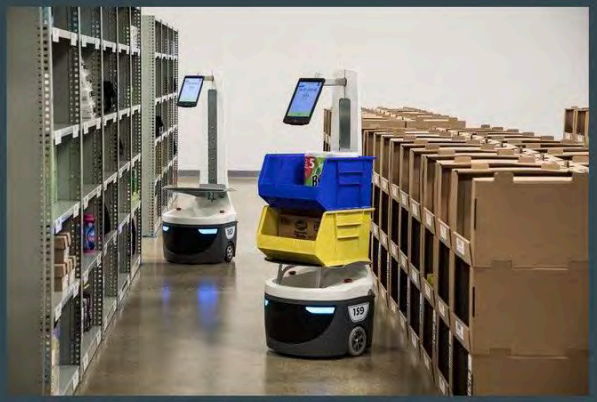


open robotics

(some of the) ROS-based products available today







ROS History

2007 - Stanford Switchyard → Part of the STanford Artificial Intelligence Robot (STAIR) project

STAIR I

STAIR II

Morgan Quigley

2008 - Willow Garage increasingly assumes stewardship of ROS

2/2009 - ROS 0.4 (first "stable" release)

4/2009 - Lincoln starts using ROS

1/2010 - ROS 1.0

3/2010 - Box Turtle

8/2010 - C Turtle

3/2011 - Diamondback

8/2011 - Electric Emys

4/2012 - Fuerte

4/2012 - Tully Foote becomes ROS Platform Manager

1/2013 - Groovy Galapagos

2/2013 - ROS stewardship transfers to the Open Source Robotics Foundation (OSRF)

9/2013 - Hydro Medusa

7/2014 - Indigo Igloo

Willow Garage

PR2

Ken Conley

Tully Foote

Brian Gerkey

Open Source Robotics Foundation

ROS: Past, Present, and Future: http://www.youtube.com/watch?v=-E1O98qrfBY

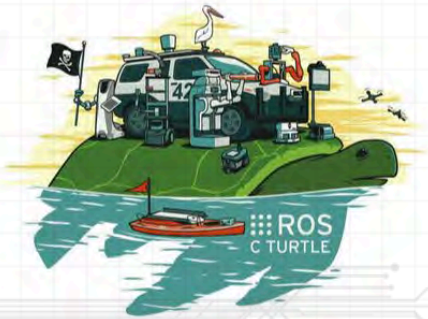
The Organization

Open Robotics

We support the development, distribution, and adoption of open software and hardware for use in robotics research, education, and product development.

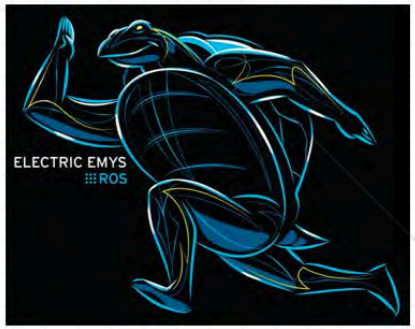
ROS distributions

2010

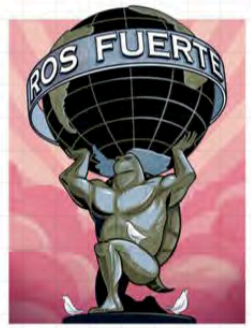


2010

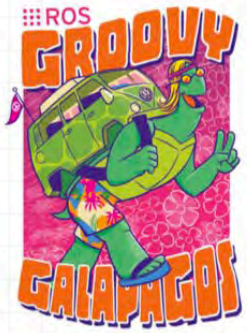
2011



2011



2012



2012

2013

2014

2015

2016

2017

2018

2020

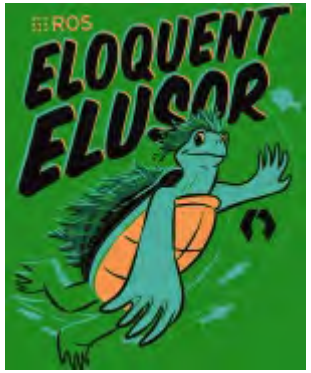
ROS2 distributions

2017

2018

2018

2019



2019

2020

ROS 1.0

ROS

Box Turtle

C Turtle

Diamondback

Electric Emys

Fuerte Turtle

Groovy Galapagos

Switchyard

Box Turtle

2007 2007.11 2010.01.22 2010.03.02 2010.08.02 2011.03.02 2011.08.30 2012.04.23 2012.12.31

Hydro Medusa

Indigo Igloo

Jade Turtle

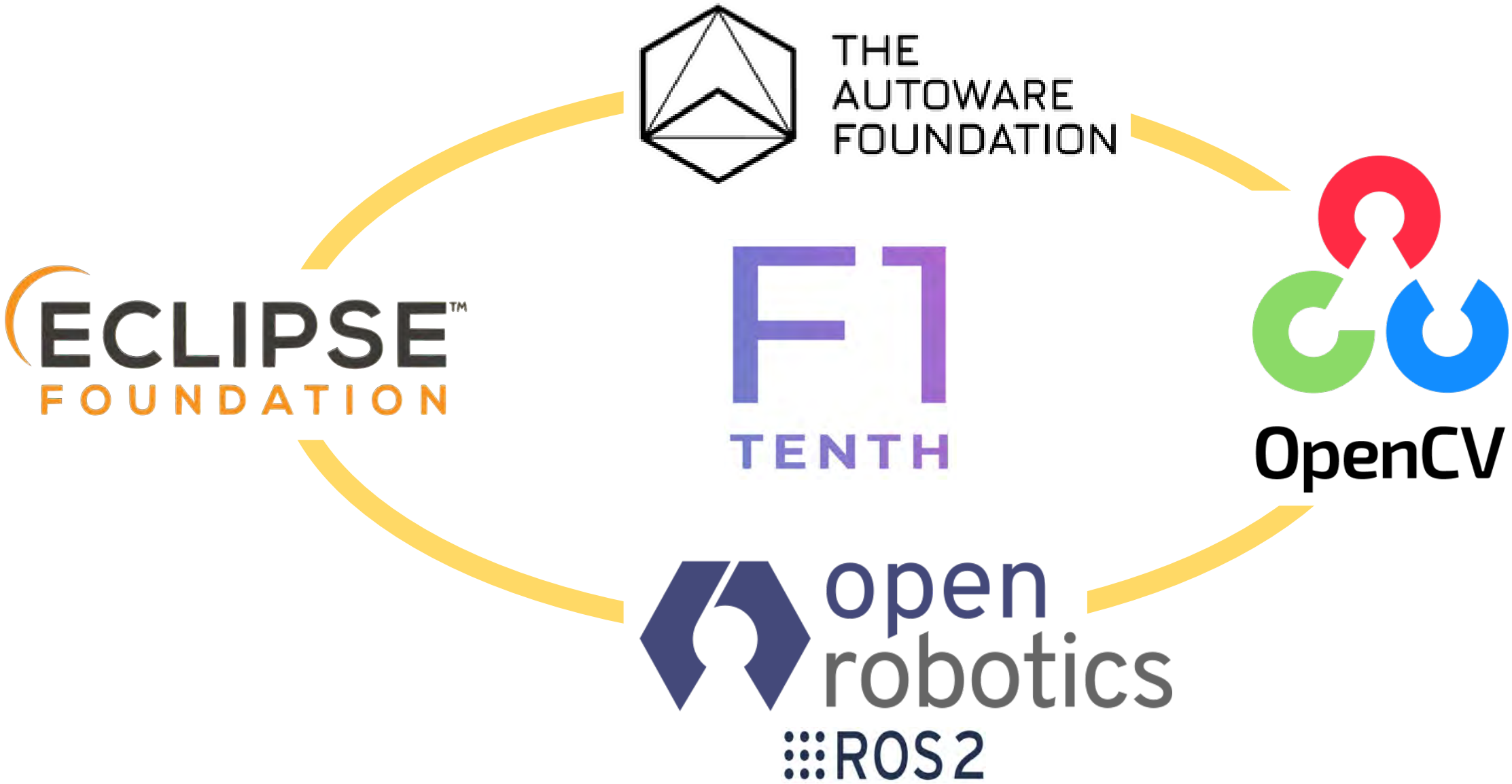
Kinetic
Kame

2013.09.04 2014.07.22 2015.05.23 2016.05.xx

B C D E F

G H I J

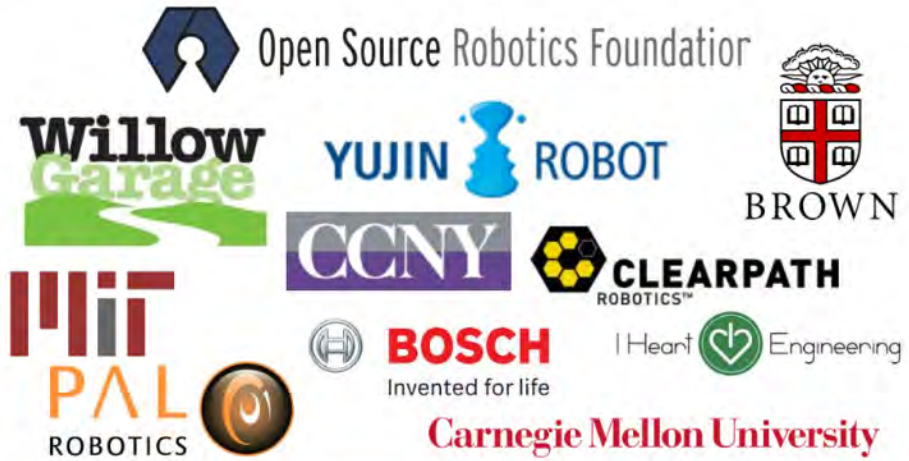
race for open source adoption & contribution



ROS Community:
The "Who" and "How"

Who Contributes to & Uses ROS?

- "Full-time" ROS developers
  - Open Source Robotics Foundation (OSRF)
- Independent contributors
  - Academia
    - Mostly graduate students
  - Industry
    - Robot builders & application developers
  - Hobbyists



How Do They Contribute / Coordinate?

- ROS.org
  - Wiki documentation & ROS standards
  - Special Interest Groups (SIG)
  - ROS Enhancement Proposal (REP)
  - ros-users mailing list
- ROSCon
- Shared code repositories
  - Linked by ROS.org, most moving to GitHub

ROS Build farm and packages

ROS.org

ROS packages for Kinetic

Quick filter: *, SYNC, REGRESSION, DIFF, BLUE, RED, ORANGE, YELLOW, GRAY, ORPHANED

showing 2667 of 2667 total

B T M the repositories
- same version
- lower version
- higher version
- missing
- obsolete
- intentionally missing

Page was generated: 6 minutes ago

Name	Repo	Version	Status	Maintainer	Xsource 2666 2666 2664	X64 2632 2632 2631	X32 2626 2626 2625
abb	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_driver	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_irb2400_moveit_config	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_irb2400_moveit_plugins	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_irb2400_support	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_irb4400_support	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_irb5400_support	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_irb6600_support	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_irb6640_moveit_config	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_irb6640_support	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abb_resources	abb	1.3.1-1	developed	Levi Armstrong (Southwest R			
abseil_cpp	abseil_cpp	0.4.2-3	maintained	dfaconti			
acado	acado	1.2.2-0	unknown	Ronald Ensing			
access_point_control	linux_networking	1.0.15-0	maintained	Devon Ash			
ackermann_controller	ackermann_controller	0.1.2-0	developed	Easymov Robotics			
ackermann_msgs	ackermann_msgs	1.0.1-0	maintained	Jack O'Quin			

ROSCon


ROSCon 2012


ROSCon
2013


ROS HONG 2014
ROSCon
CHICAGO '14


ROSCon
HAMBURG
2015


ROSCon 2016
SEOUL


ROSCon 2017
VANCOUVER


ROSCon '19
MACAU


ROS WORLD
2020

ROSCon '19 Talk: ROS F1/10 Autonomous Racing Simulator

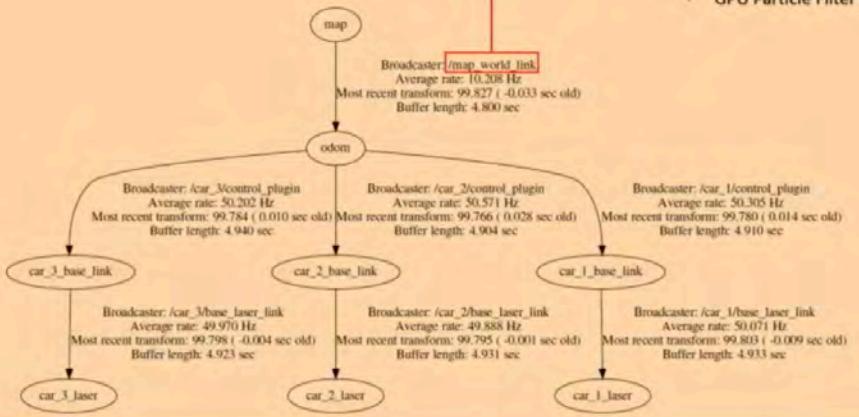
open robotics

Localization

(Implemented for multiple vehicles)

Odometry sources
- Gazebo (available, default option)
- Laser Scanmatcher (available, implemented)
- Virtual Wheel Encoders (work in progress)

Provided by
- AMCL (default, implemented)
- GPU Particle Filter (available, tested)



roscon.ros.org/2019 #ROSCon @OpenRoboticsOrg

19:35 / 25:07

A relevant degree is required, for instance in Computer Science or Engineering. A background in Robotics/Computer Vision is desirable, while knowledge of the Robot Operating System (ROS), the Point Cloud Library (PCL), or the Open Source Computer Vision Library (OpenCV) is a big plus.

Goal of this PhD is to study, design and build novel industry-level software based on ROS or ROS-Industry which is modular, reconfigurable, adaptive, easy to use to integrate and control various robotic systems.

** Job Requirements **
- Experience with software development and integration preferably in a Linux environment
- Proficient in C, C++, Python and Matlab
- Those with past experience using ROS will be preferred

The candidate must be a proficient user of C/C++ and ROS and any relevant computer vision library (e.g., ViSP, OpenCV, PCL). Scientific curiosity, large autonomy and ability to work independently are also expected.

** Required Qualifications **
- A university degree (Master or Diplom-Ingenieur)
- Proven programming skills in Matlab, C / C ++, ROS
- Good knowledge of cooperative software development with GIT or SVN

Required Skills
* MSc in Engineering / Computer Science or equivalent.
* Experience with Robotics
* Knowledge about ROS (Robot Operating System) and CV.
* Advanced experience with C++ and soft real-time programming.
* Team spirit and ability to work independently.
* Excellent communication skills, flexibility and creativity.

ROSCon 2015
Hamburg, Germany

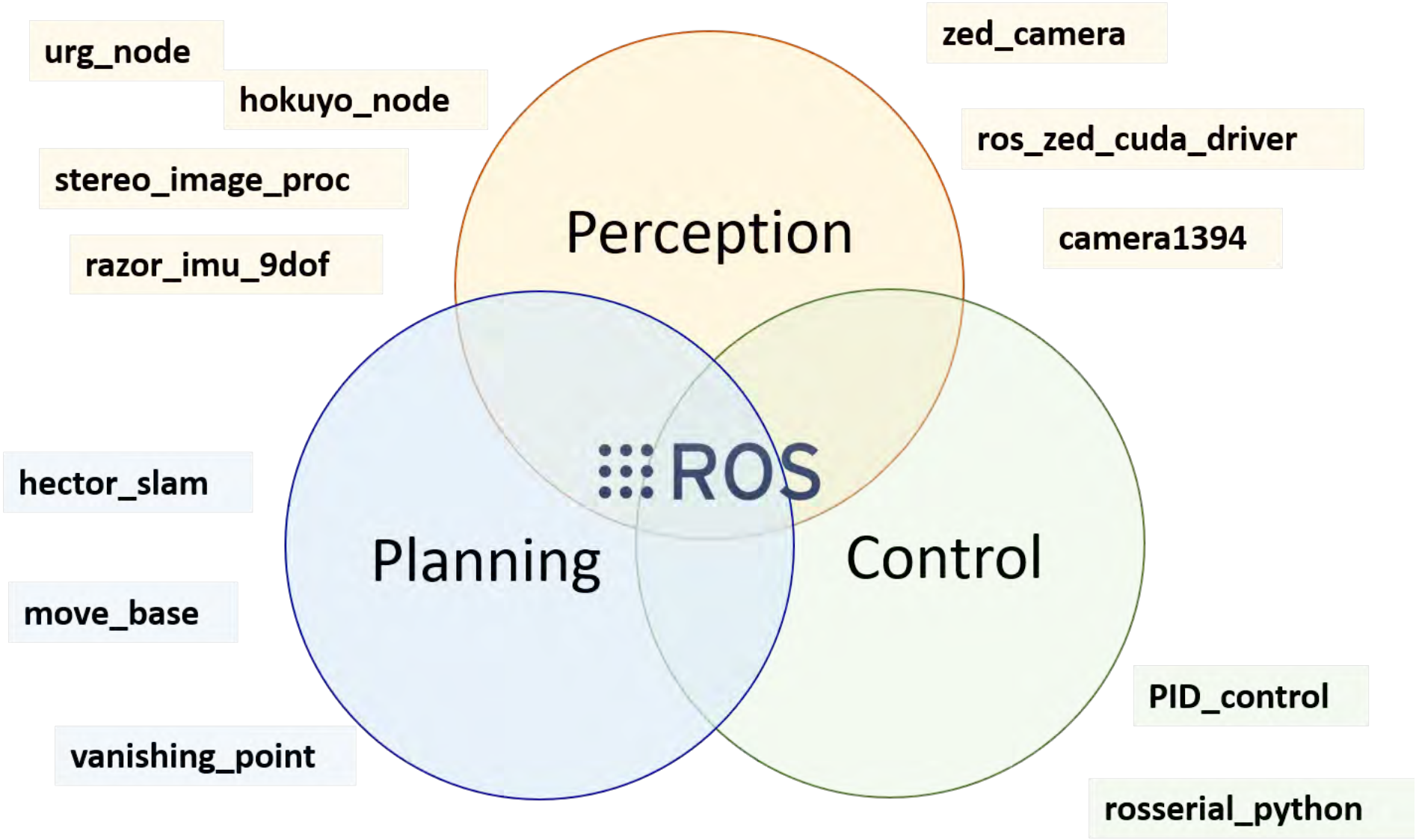
AUTOMATED DRIVING WITH ROS AT BMW.

MICHAEL AEBERHARD, THOMAS KÜHBECK, BERNHARD SEIDL, MARTIN FRIEDL,
JULIAN THOMAS, OLIVER SCHEICKL.

BMW GROUP

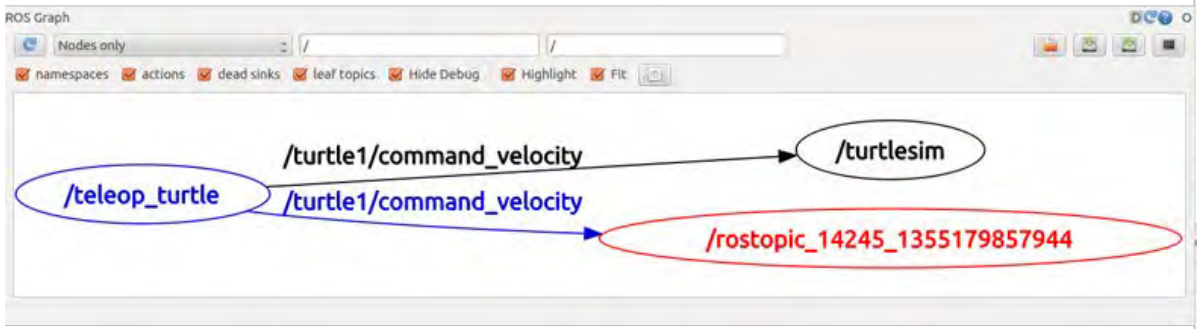
ROS Ecosystem:

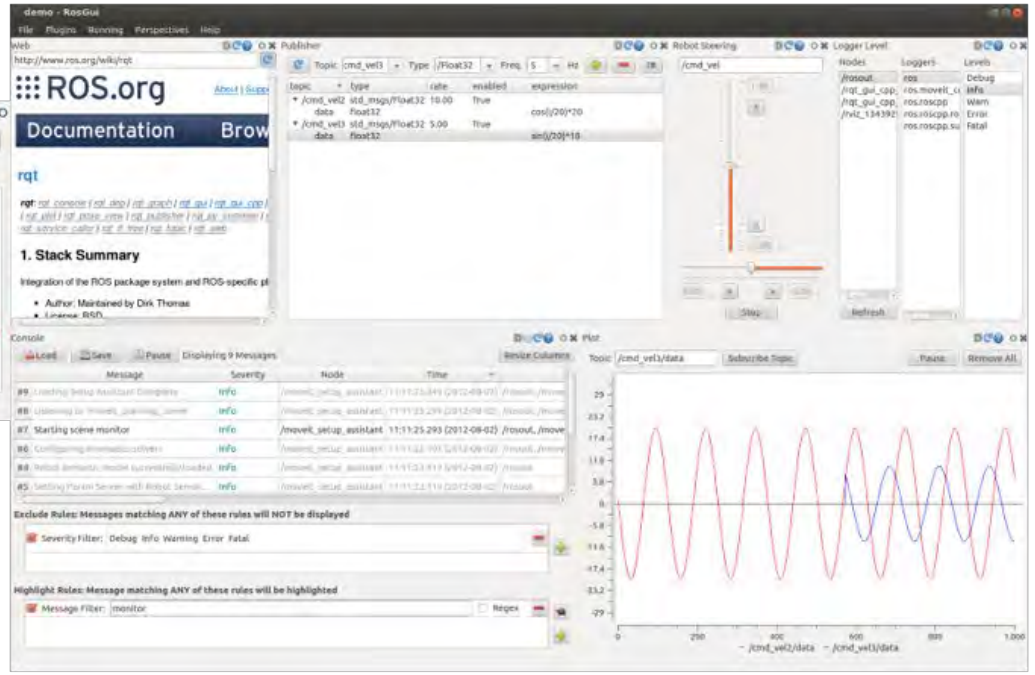
Example F1/10 ROS Packages

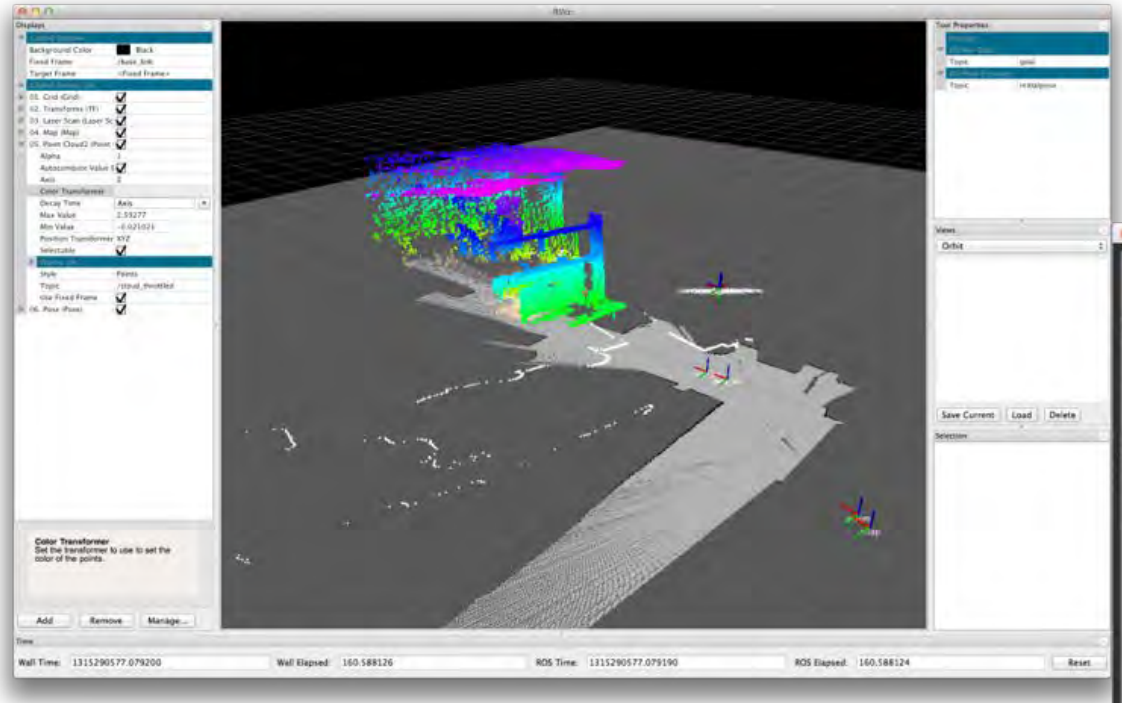


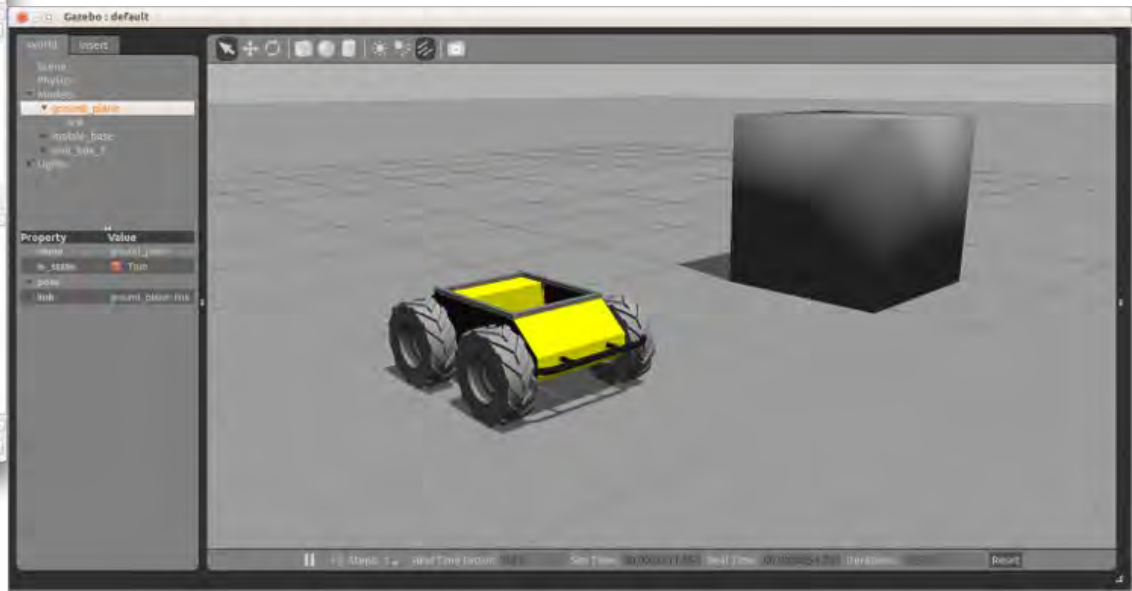
Visualization, debugging and diagnostics, logging, and simulation

ROS Tools







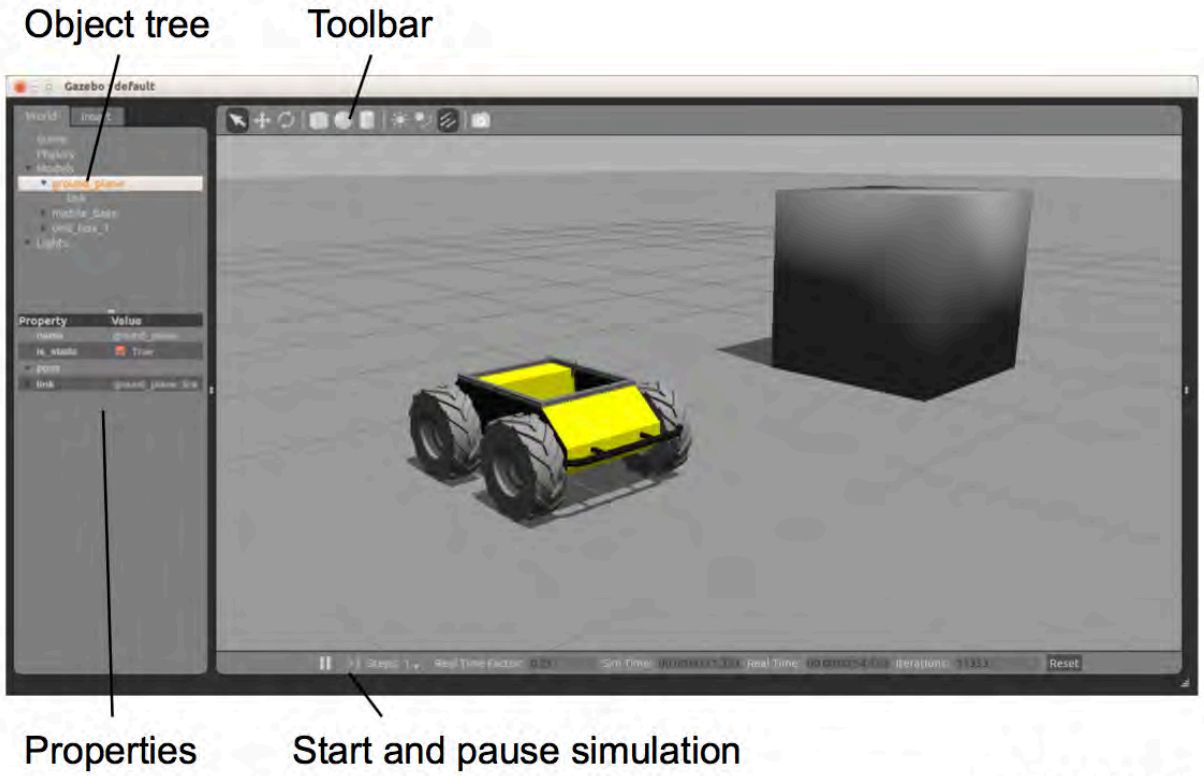


Gazebo Simulator

- Simulate 3d rigid-body dynamics
- Simulate a variety of sensors including noise
- 3d visualization and user interaction
- Includes a database of many robots and environments (*Gazebo worlds*)
- Provides a ROS interface
- Extensible with plugins

Run Gazebo with

```
> rosrun gazebo_ros gazebo
```



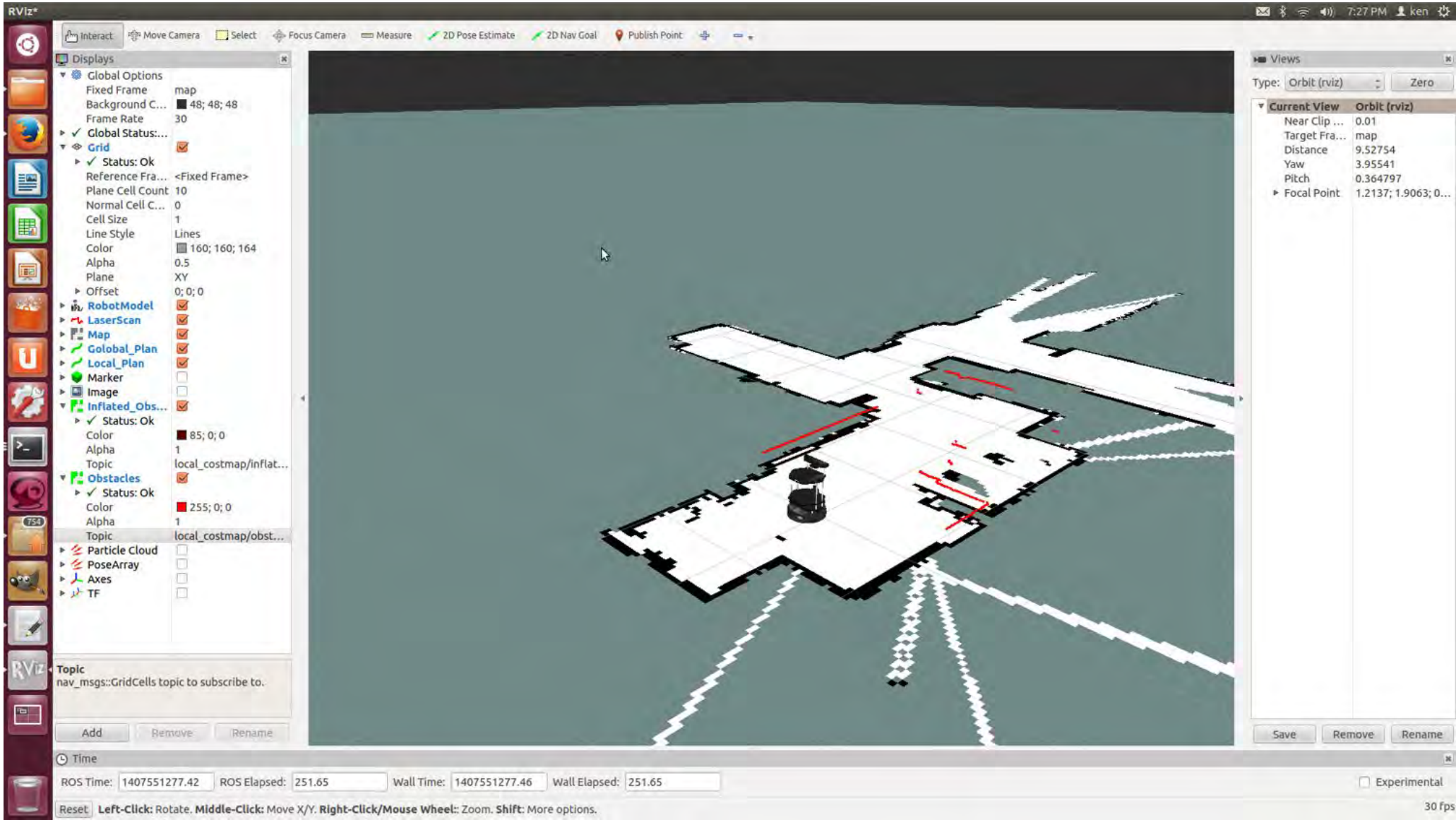
More info

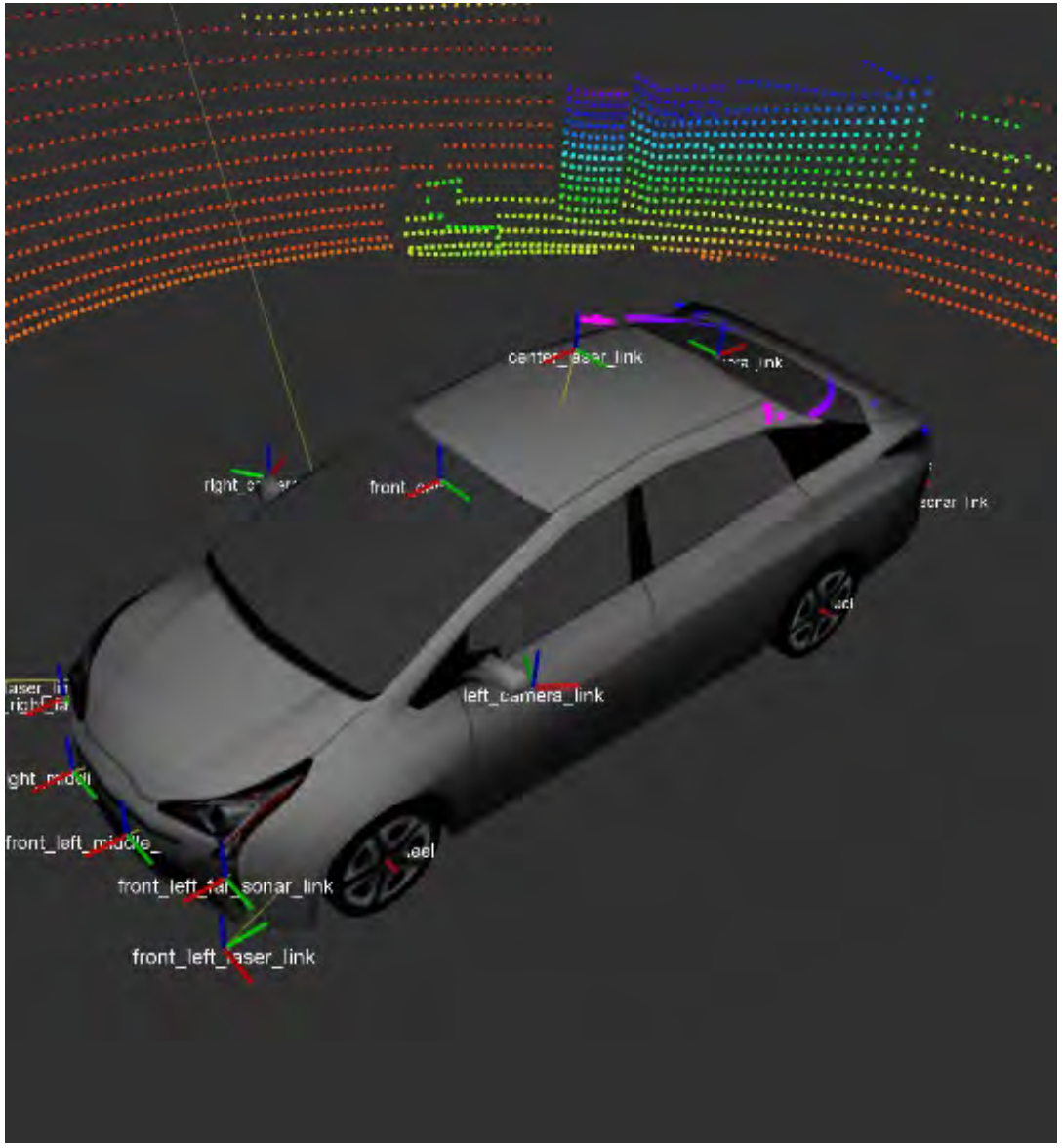
http://gazebosim.org/

http://gazebosim.org/tutorials

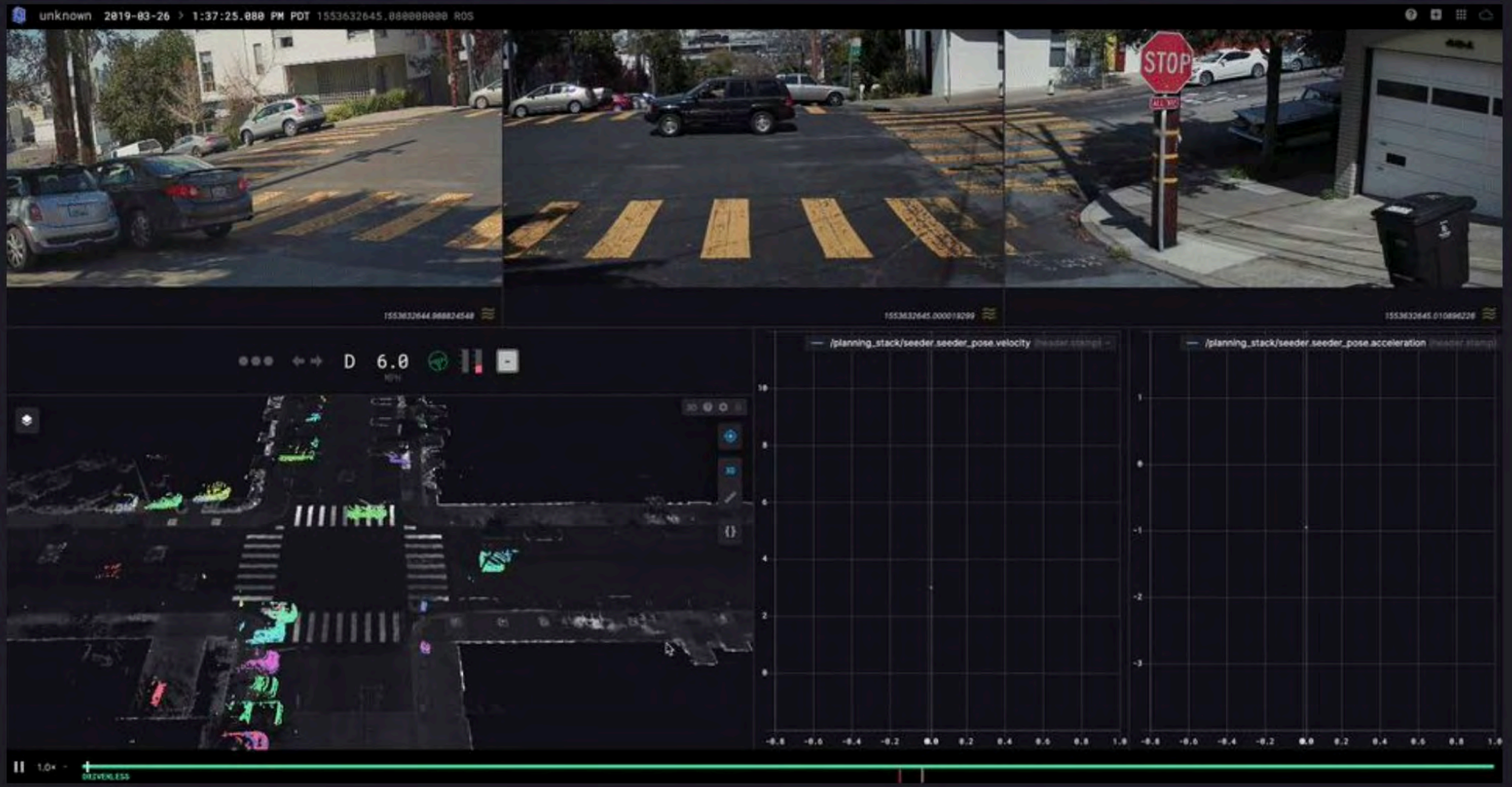
3D visualization tool: rviz

```
$ rosrun rviz rviz
```

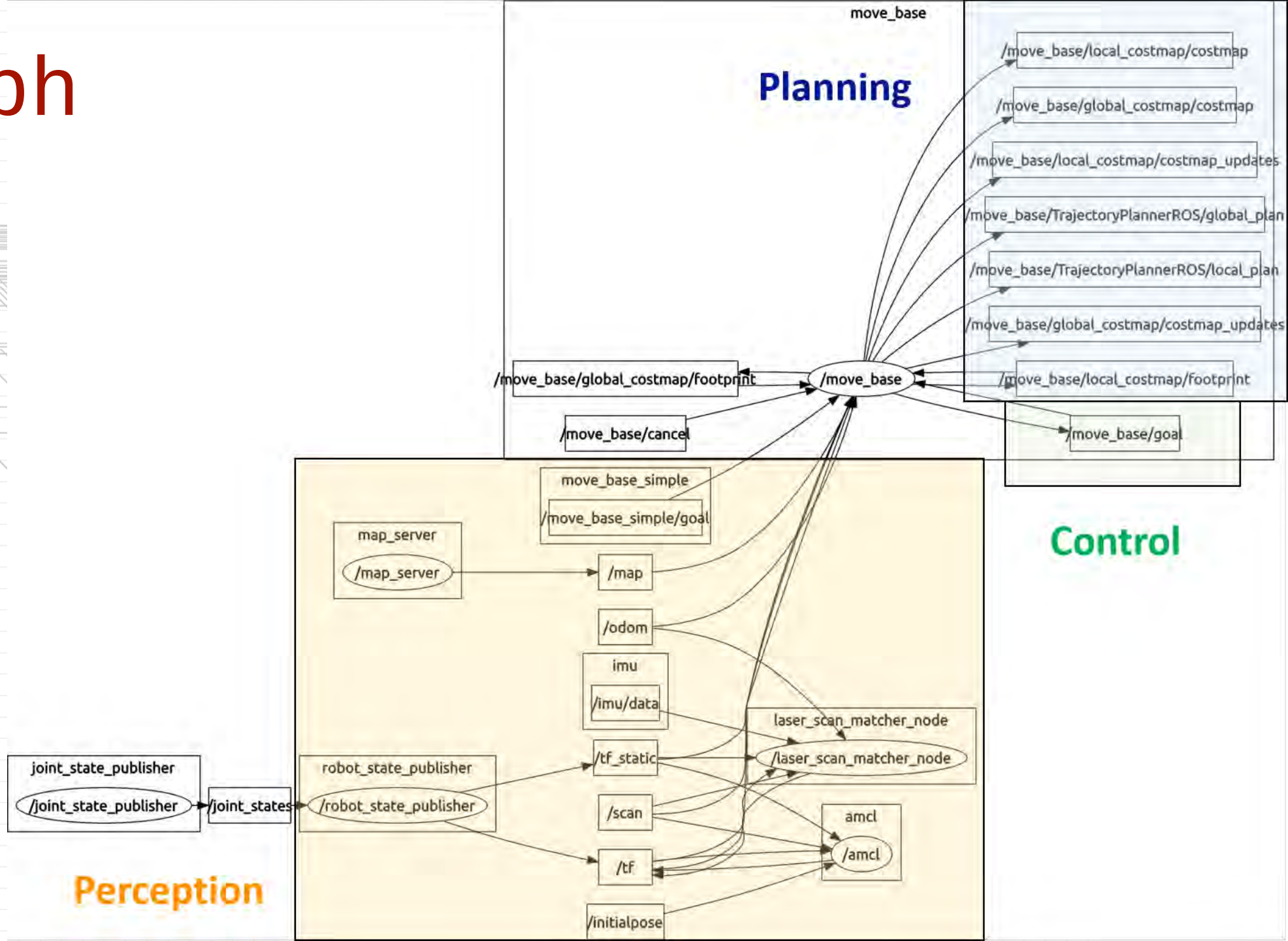


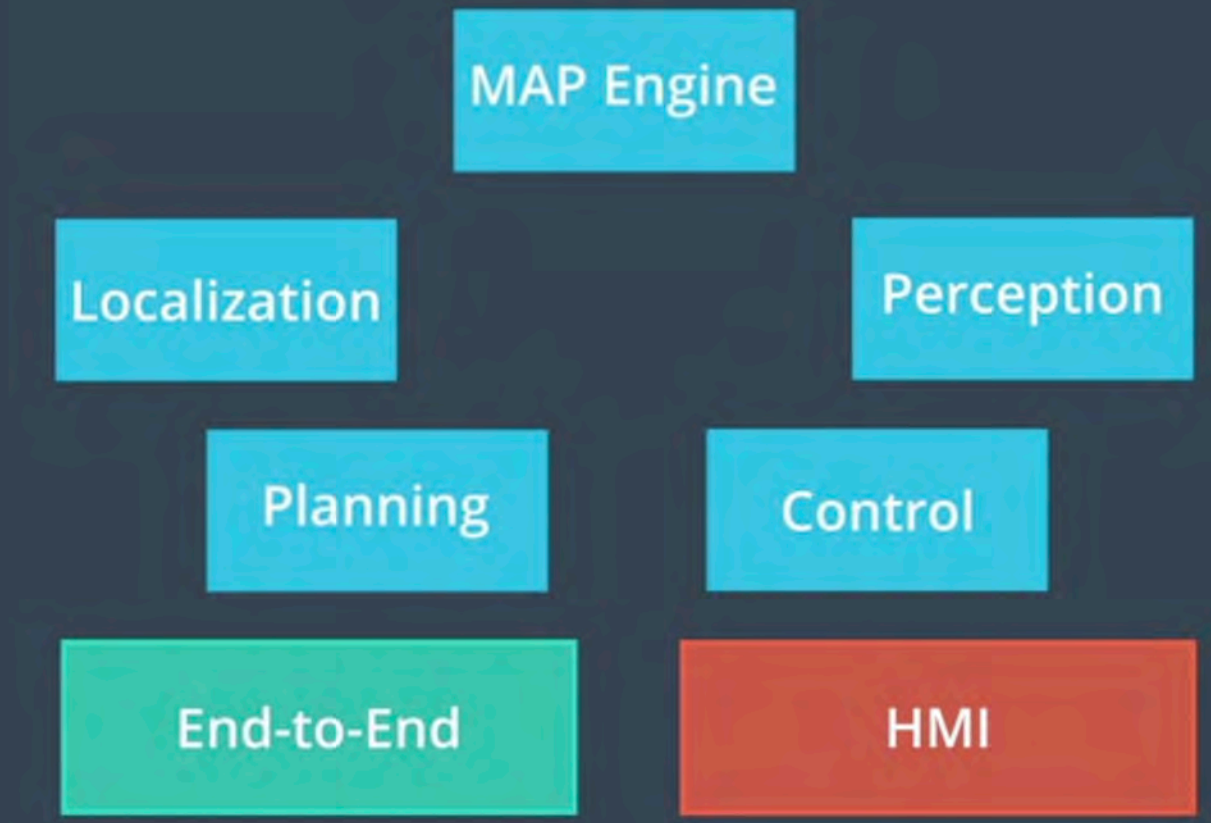

center_laser_link
left_camera_link
front_left_sonar_link
front_left_laser_link

Webviz

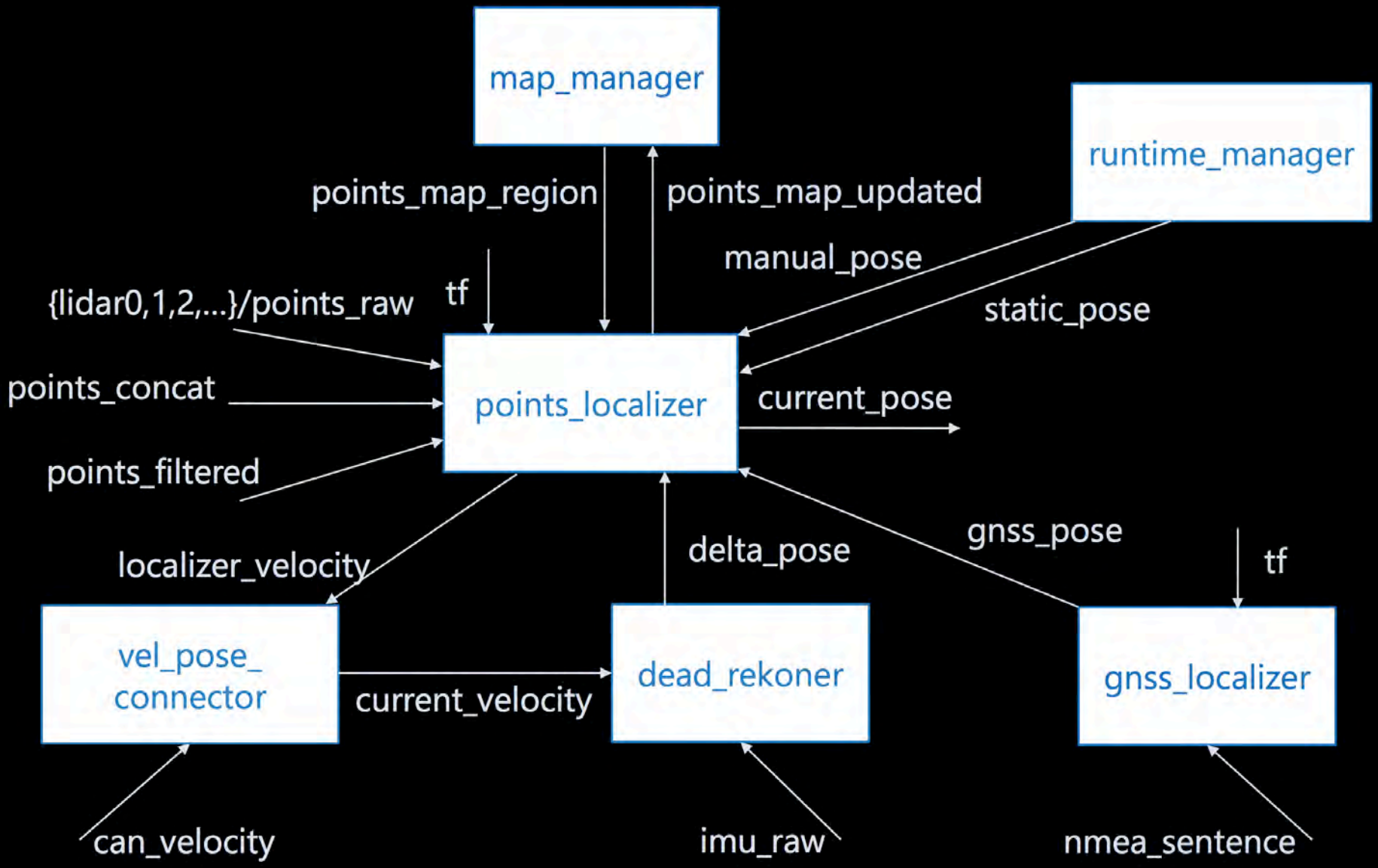

unknown 2019-03-26 > 1:37:25.080 PM PDT 1553632645.080000000 ROS
STOP
D 6.0
/planning_stack/seeder.seeder_pose.velocity
/planning_stack/seeder.seeder_pose.acceleration

rqt_graph

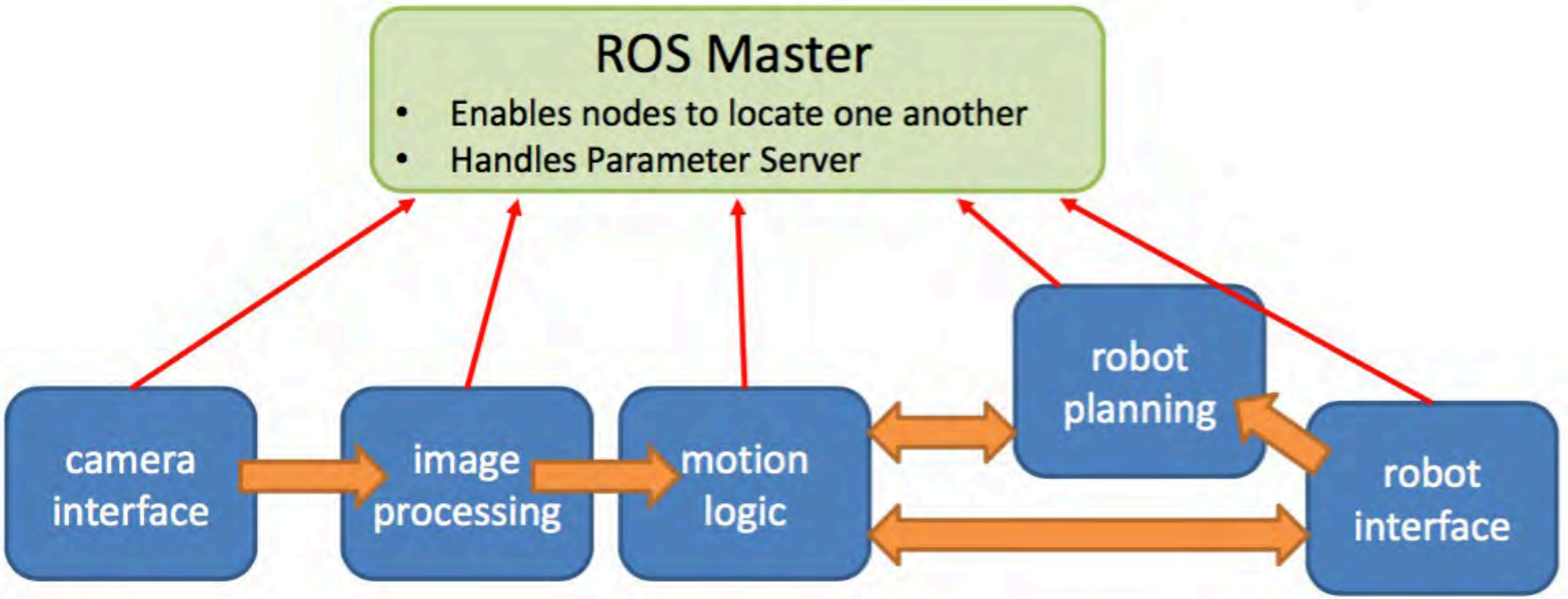

move_base
Planning
/move_base/local_costmap/costmap
/move_base/global_costmap/costmap
/move_base/local_costmap/costmap_updates
/move_base/TrajectoryPlannerROS/global_plan
/move_base/TrajectoryPlannerROS/local_plan
/move_base/global_costmap/costmap_updates
/move_base/local_costmap/footprint
/move_base/global_costmap/footprint
/move_base
/move_base/cancel
/move_base/goal
Control
move_base_simple
/move_base_simple/goal
map_server
/map_server
/map
/odom
imu
/imu/data
laser_scan_matcher_node
/laser_scan_matcher_node
/tf_static
joint_state_publisher
/joint_state_publisher
/joint_states
robot_state_publisher
/robot_state_publisher
/scan
amcl
/amcl
/tf
/initialpose
Perception


MAP Engine
Localization
Perception
Planning
Control
End-to-End
HMI

Packages Example (Localization)



ROS: Nodes

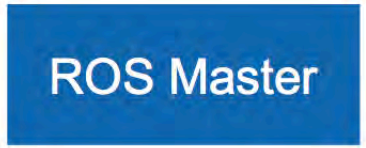


Node: Program with a specific functionality, that runs as a single process.

Nodes communicate with other nodes using **topics** and **messages**

ROS Master

- Manages the communication between nodes
- Every node registers at startup with the master



Start a master with

```
> roscore
```

More info
http://wiki.ros.org/Master

ROS Nodes

- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in *packages*

Run a node with

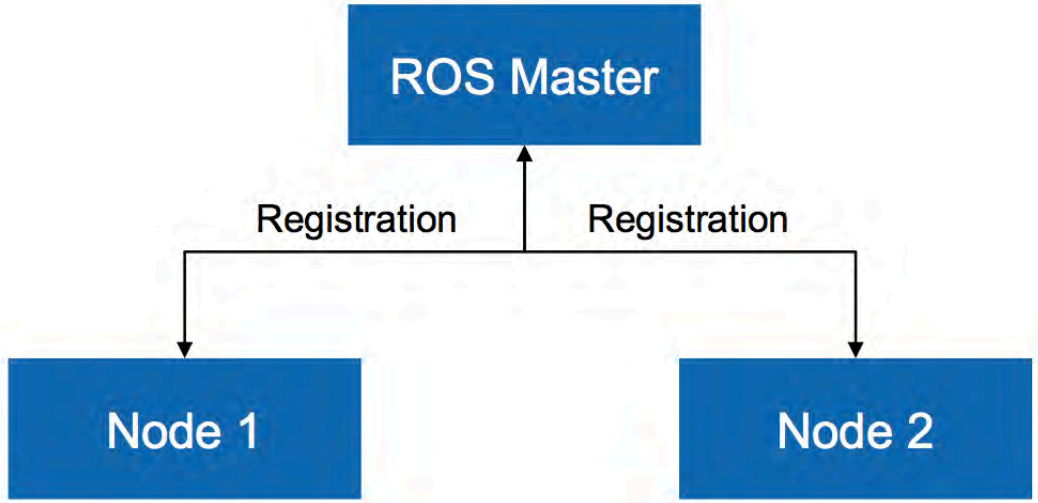
```
> rosrun package_name node_name
```

See active nodes with

```
> rosnode list
```

Retrieve information about a node with

```
> rosnode info node_name
```



More info
http://wiki.ros.org/rosnode

ROS: Topics

Topics are channels over which *nodes* exchange *messages*.

They are for **streaming data**

Publishes on topic: Scan

Scan [Topic]

Mapping Node

hokuyo_node

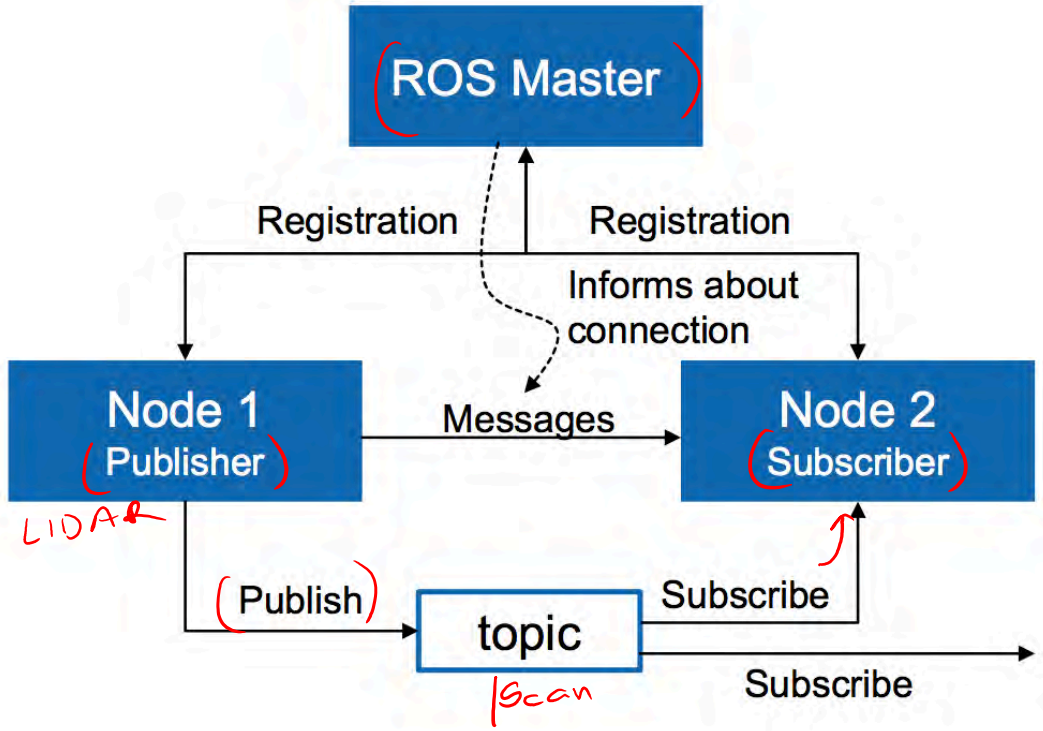
Subscribes to topic: Scan

Publisher Node

Subscriber Node

ROS Topics

- Nodes communicate over *topics*
  - Nodes can *publish* or *subscribe* to a topic
  - Typically, 1 publisher and *n* subscribers
- Topic is a name for a stream of *messages*



List active topics with

```
> rostopic list
```

Subscribe and print the contents of a topic with

```
> rostopic echo /topic
```

Show information about a topic with

```
> rostopic info /topic
```

More info
http://wiki.ros.org/rostopic

ROS: Messages

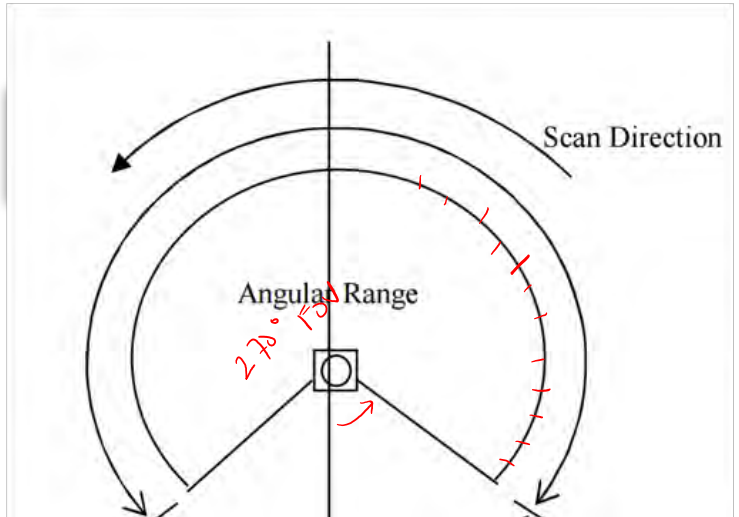
Messages are the strongly-typed **data structure** for a *topic*.

hokuyo_node

LaserScan [Message]

Scan [Topic]

Mapping Node




```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

Subscriber Node

ROS Messages

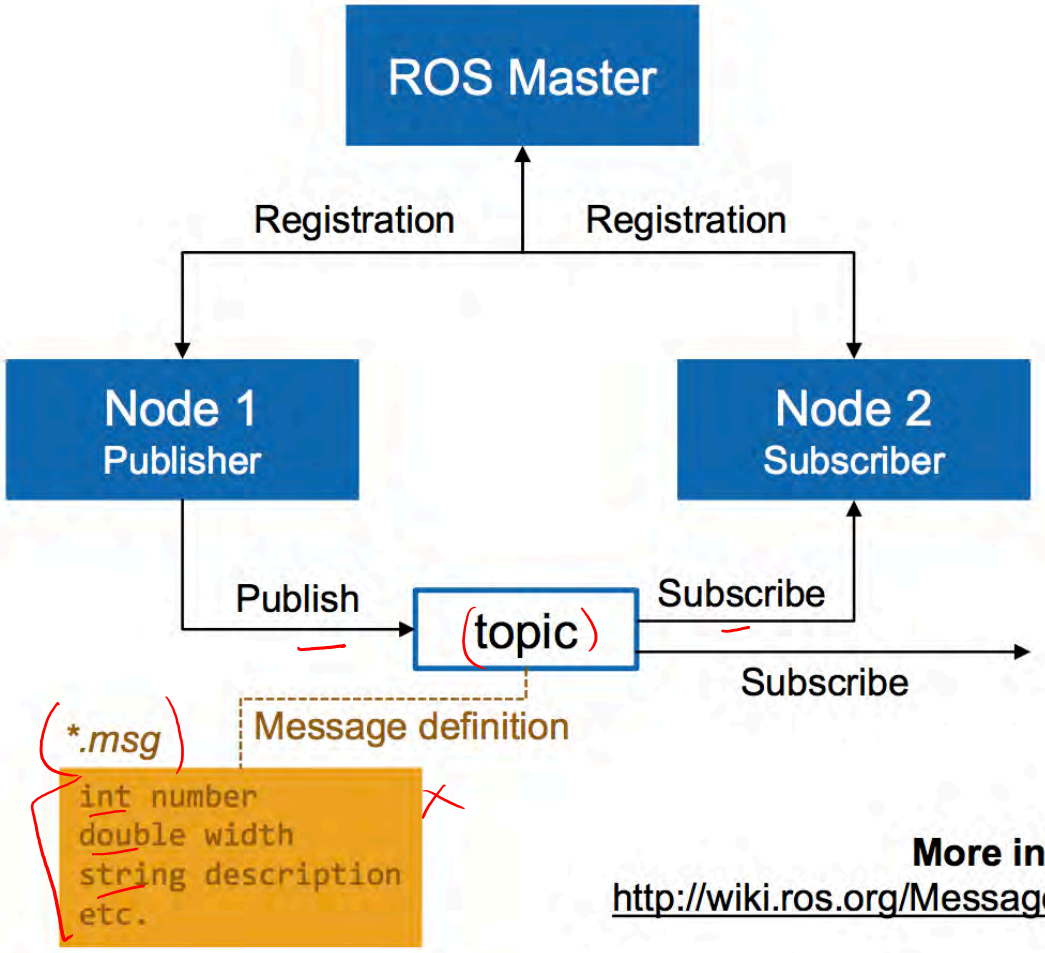
- Data structure defining the *type* of a topic
- Compromised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in **.msg* files

See the type of a topic

```
> rostopic type /topic
```

Publish a message to a topic

```
> rostopic pub /topic type args
```



More info
http://wiki.ros.org/Messages

Communication between nodes

Nodes communicate messages via topics
in an asynchronous publish-subscribe model

LaserScan [Message]

scan [topic]

Message1

Message2

...

scan

Mapping Node

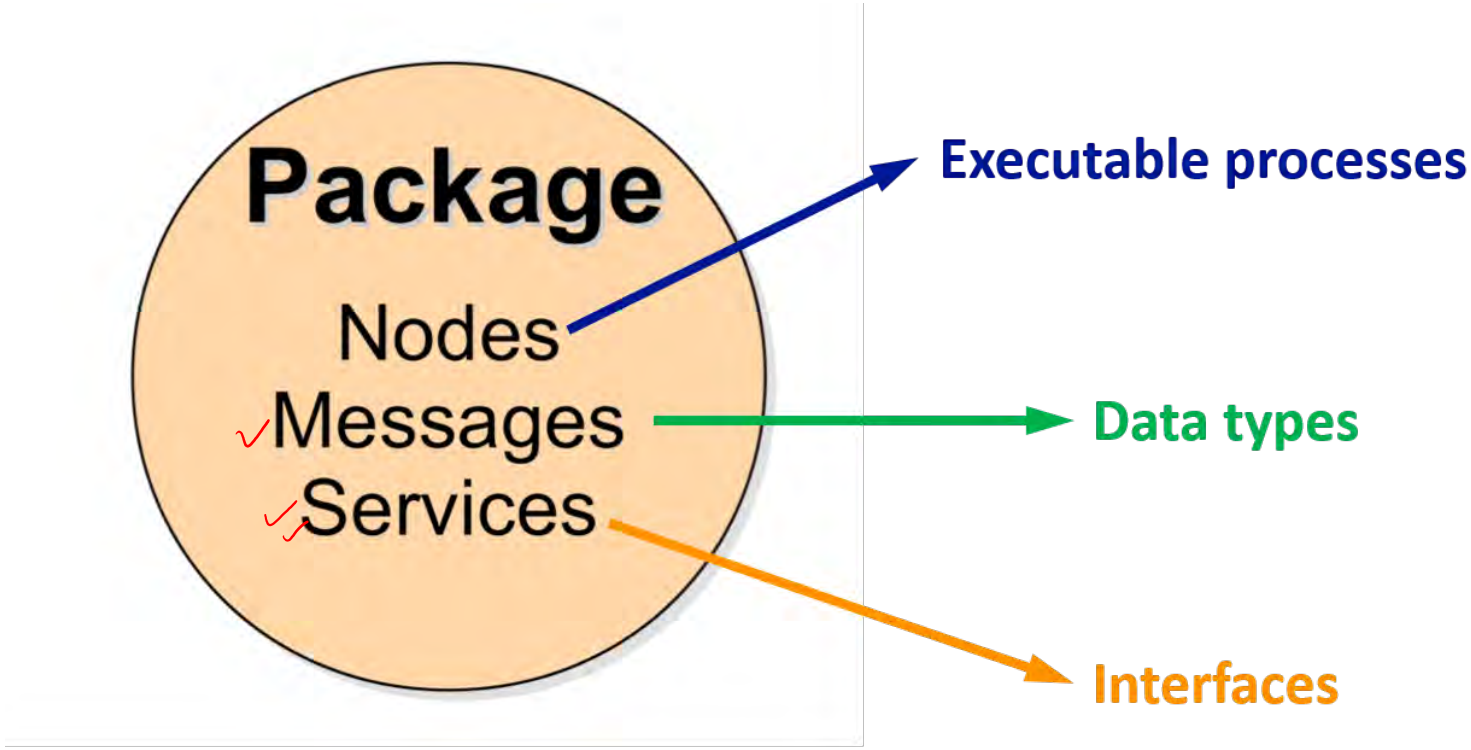
scan

Monitor Node

Publisher Node

Subscriber Nodes

ROS: Packages

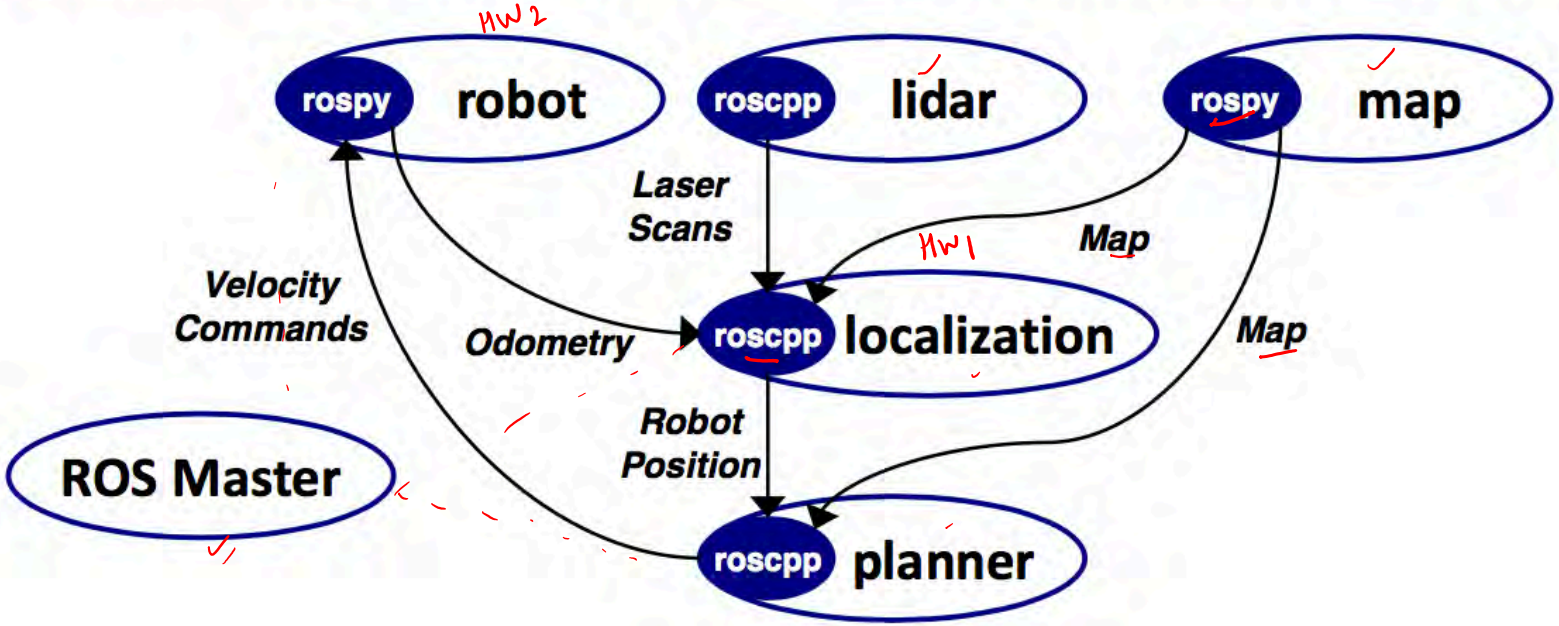


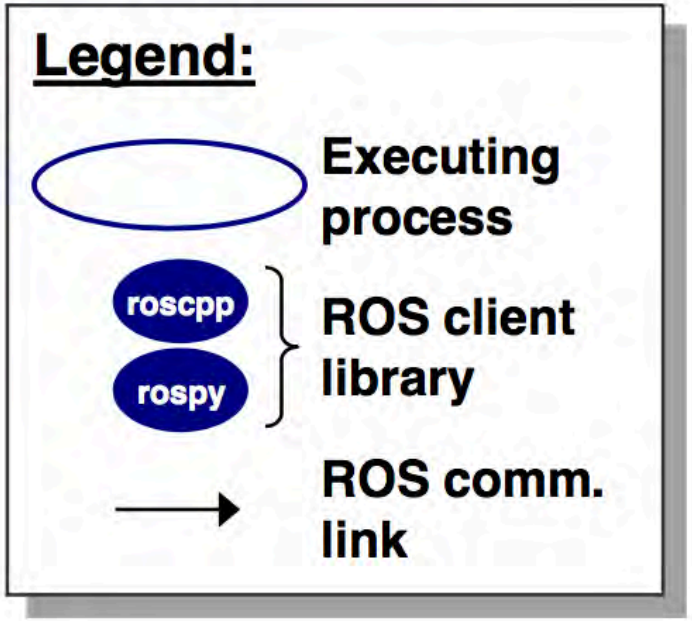
Software in ROS is organized into **packages.**

A **package** contains one or more **nodes**.

ROS Computational Graph / Communication:
Computational Graph

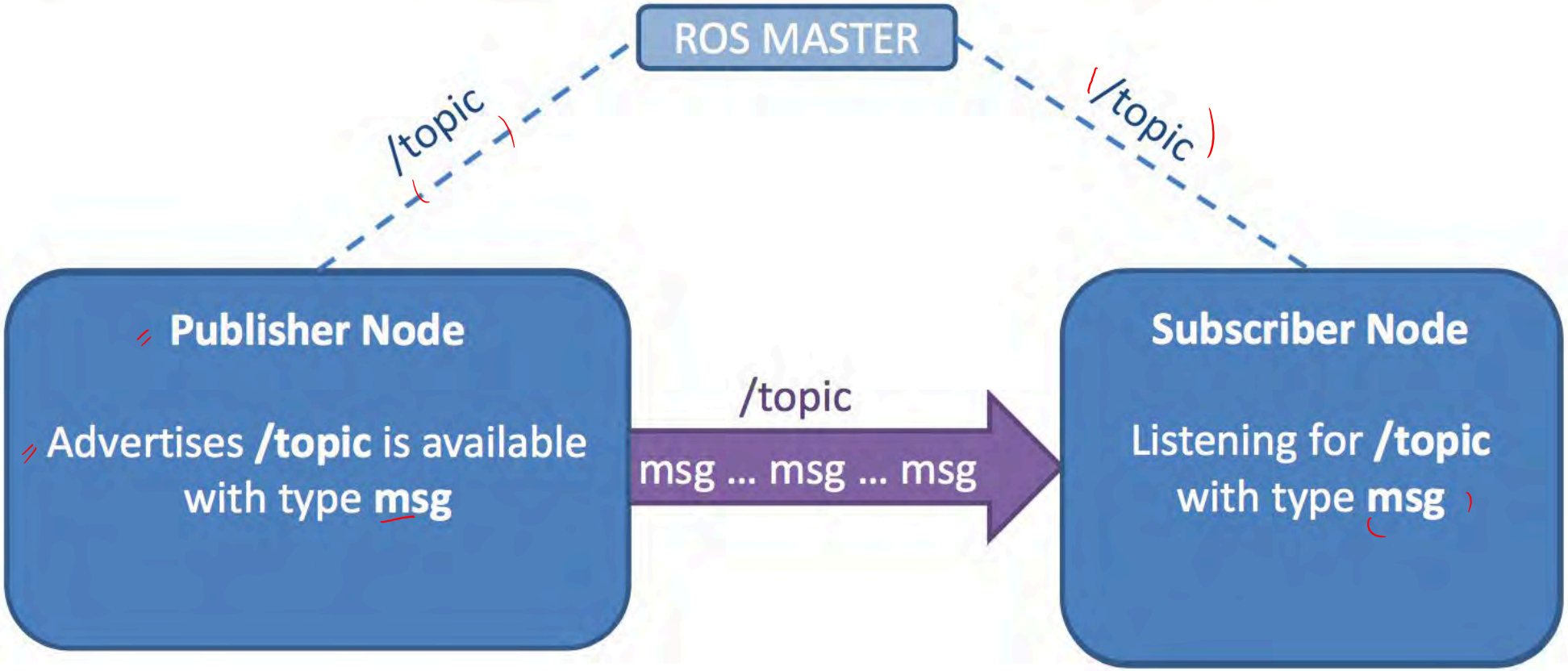
An example robot application (known-map navigation):





ROS Topics/Messages

Topics are for **Streaming Data**



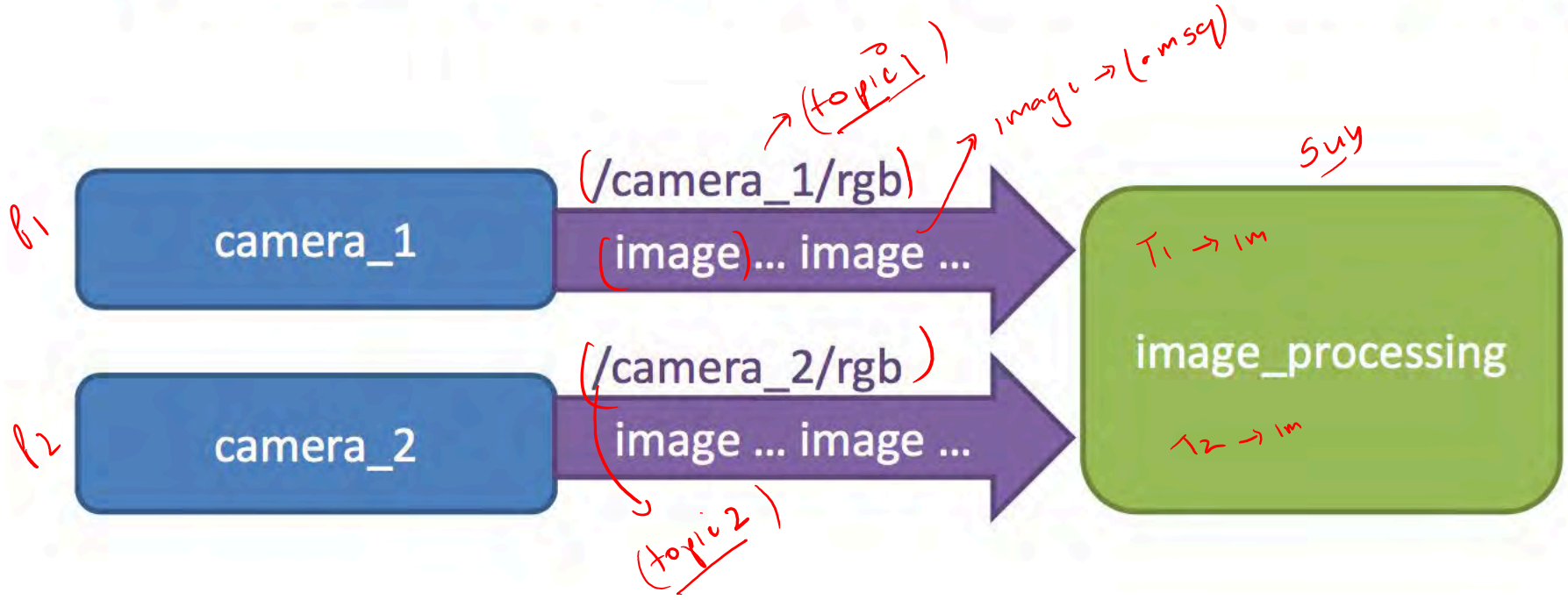
- Each **Topic** is a stream of **Messages**:
  - sent by **publisher(s)**, received by **subscriber(s)**

- Messages are **asynchronous**
  - publishers don't know if anyone's listening
  - messages may be dropped
  - subscribers are event-triggered (by incoming messages)

- Typical Uses:
  - Sensor Readings: camera images, distance, I/O
  - Feedback: robot status/position
  - Open-Loop Commands: desired position

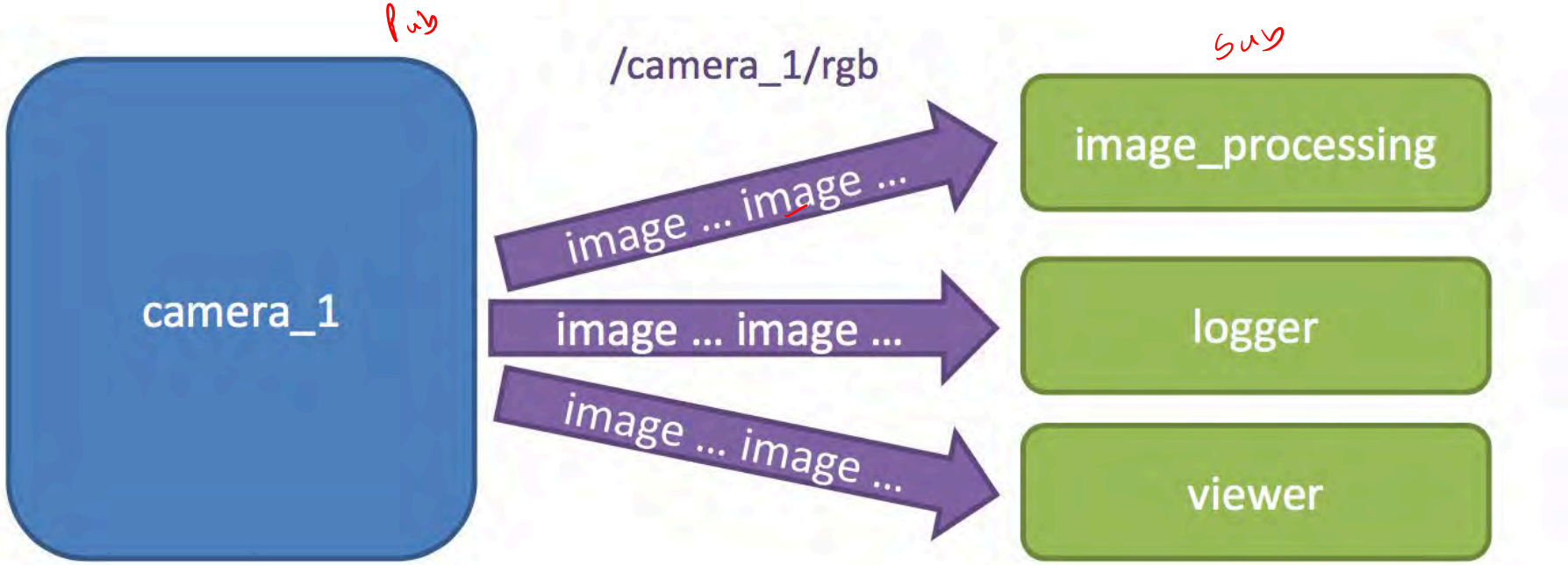
Topics vs. Messages

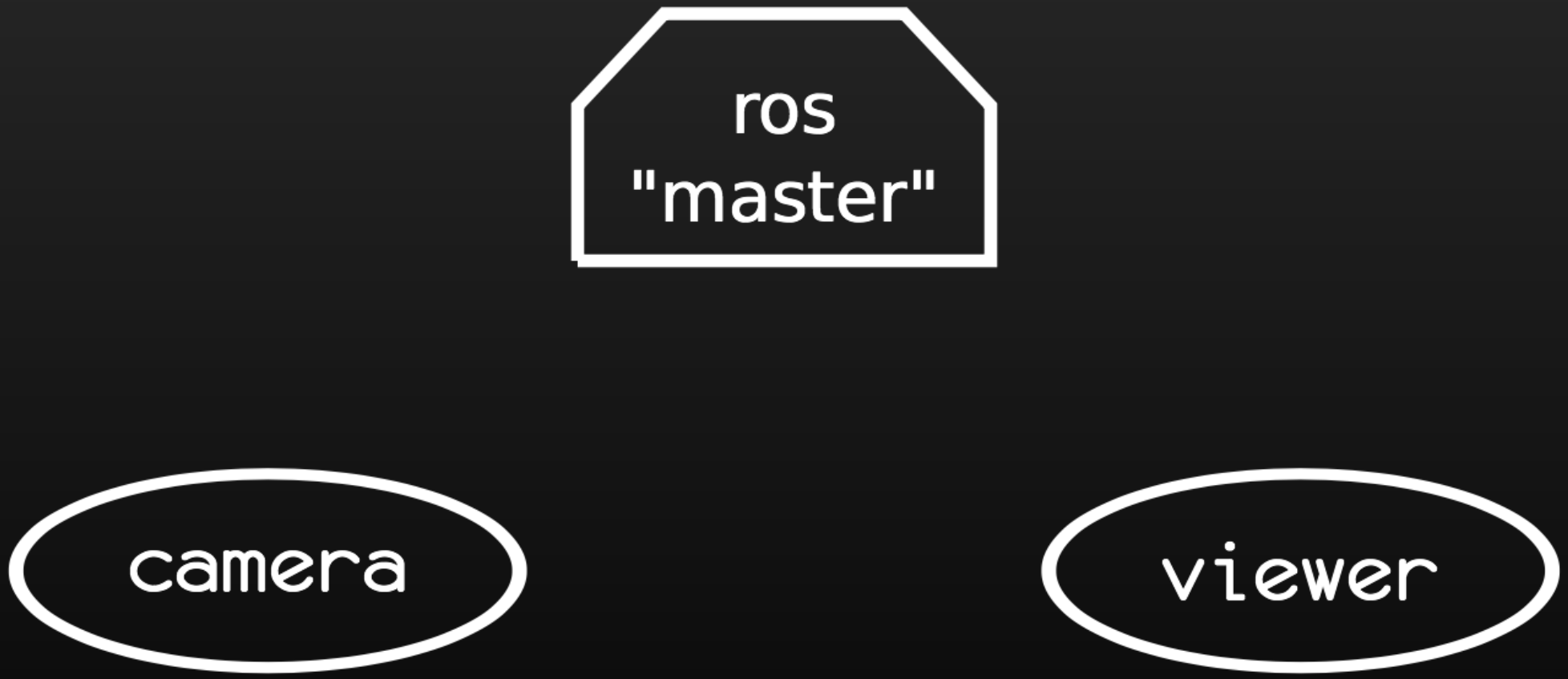
- Topics are **channels**, Messages are **data types**
  - Different topics can use the same Message type

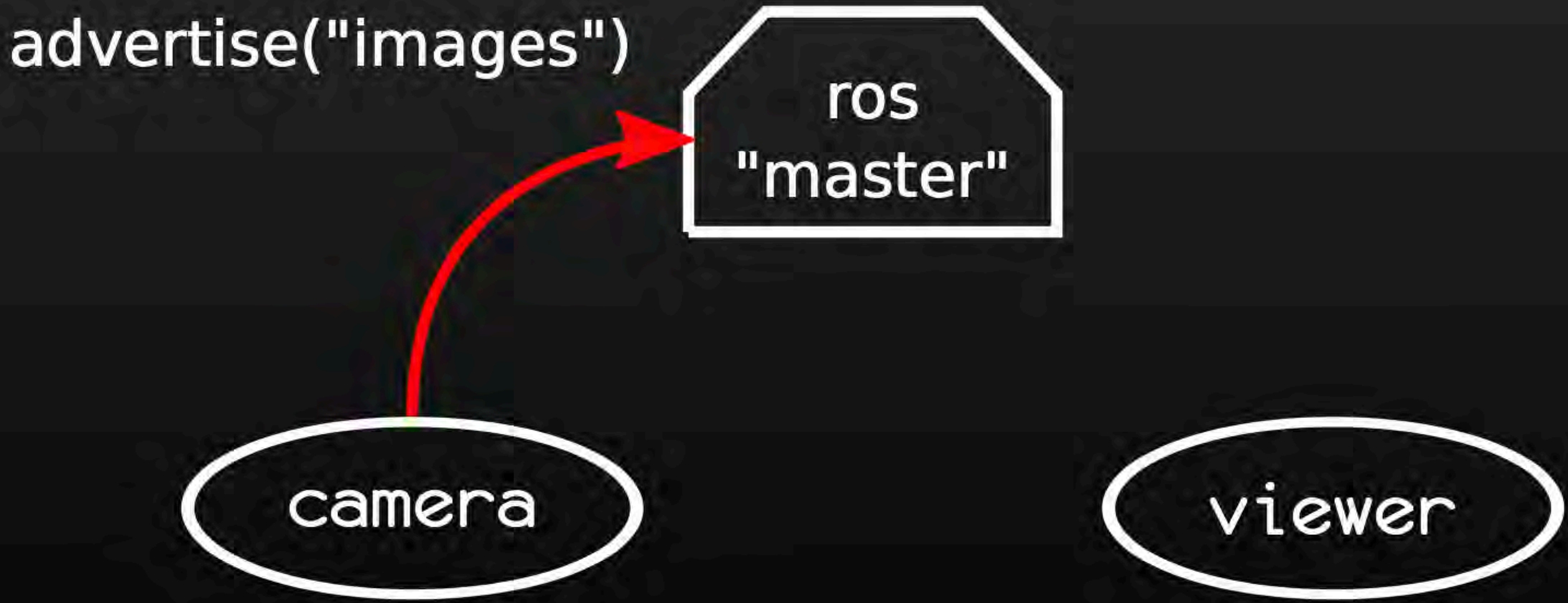


Multiple Pub/Sub

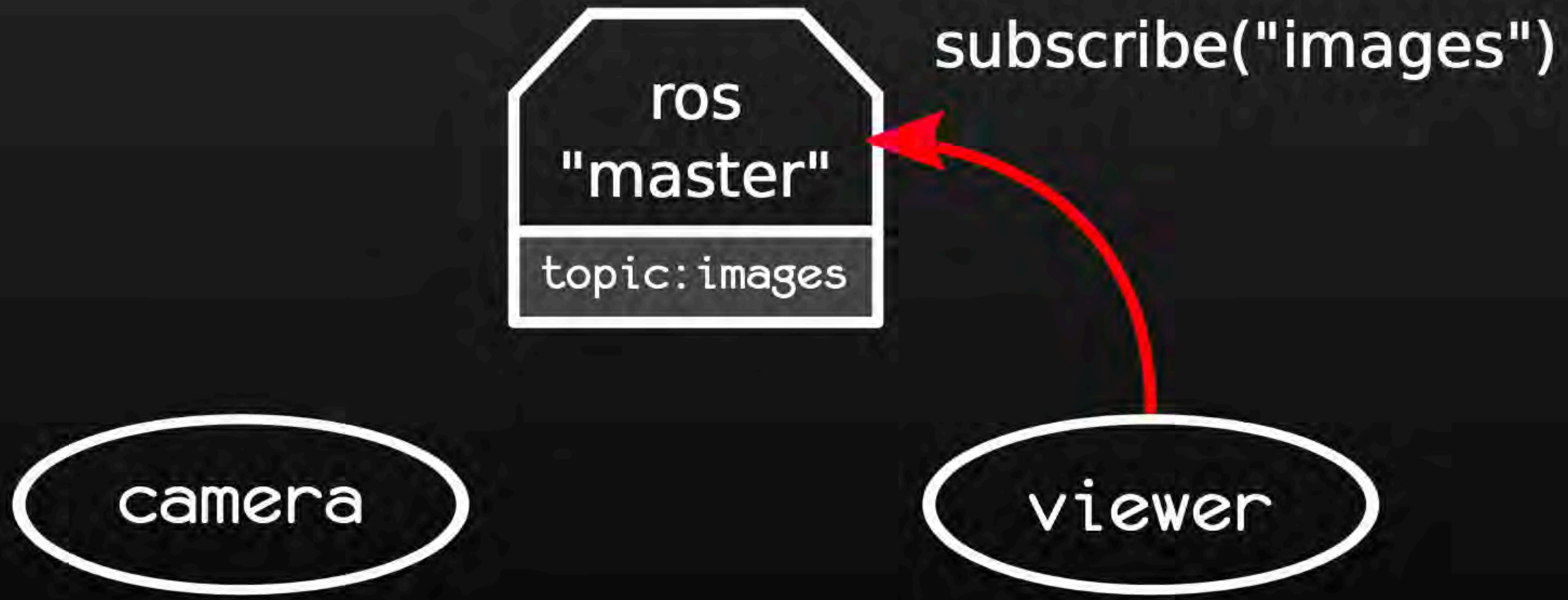
- Many nodes can pub/sub to same topic
  - comms are direct node-to-node

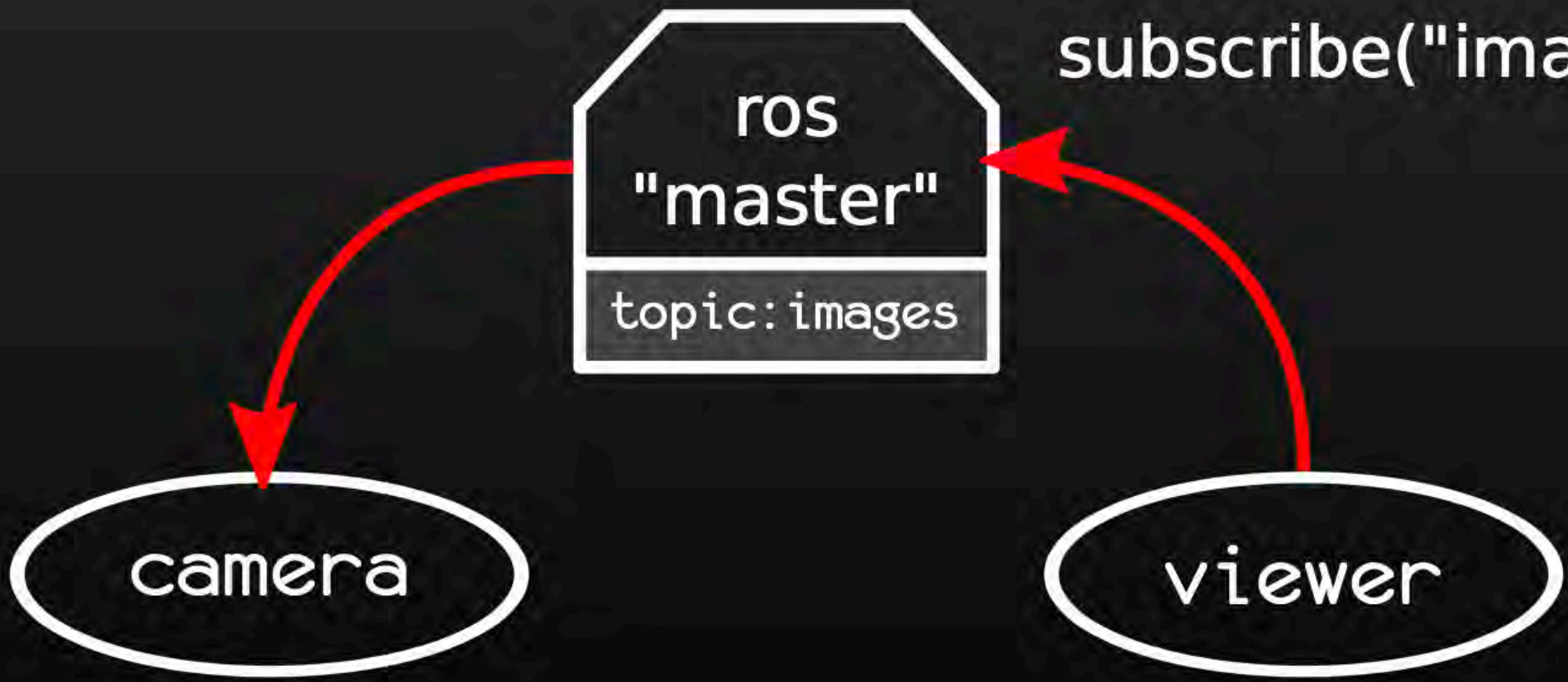



ros
"master"
camera
viewer

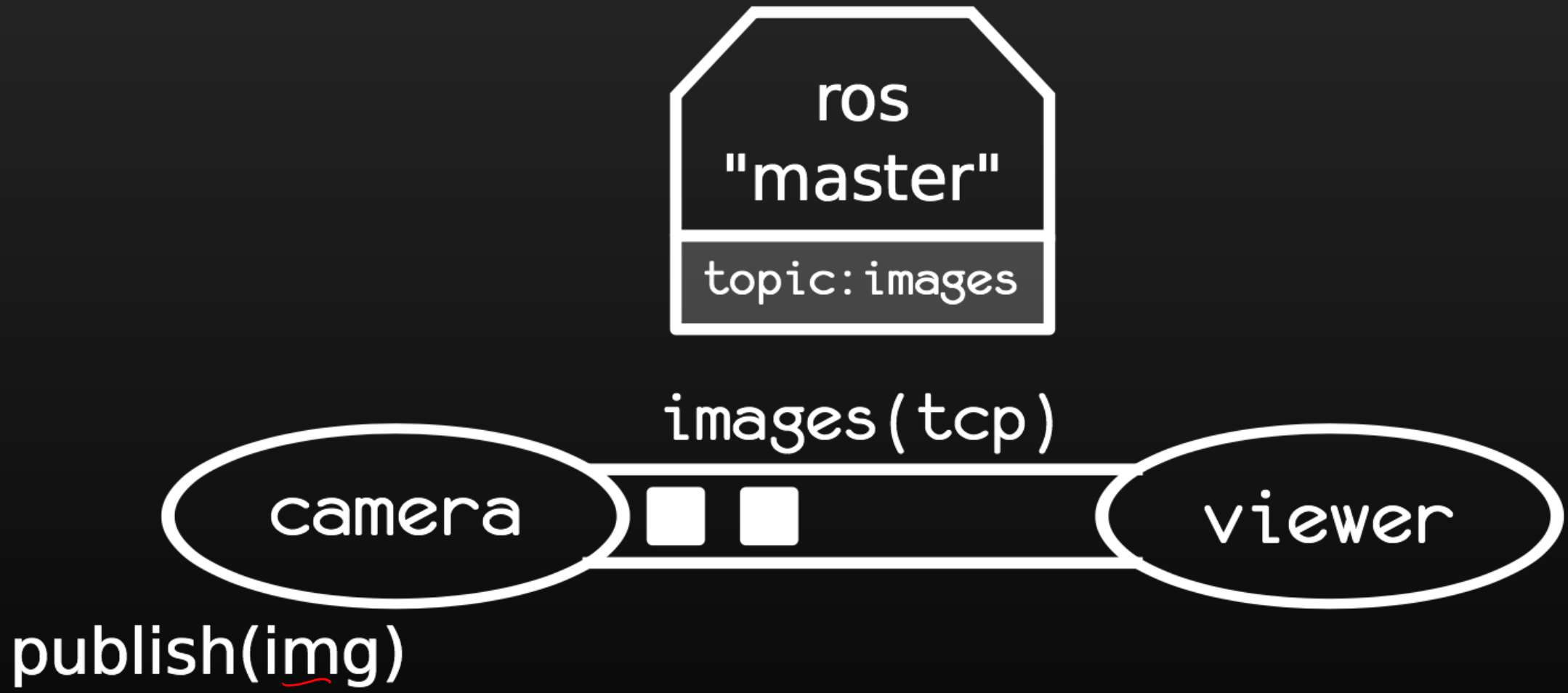

advertise("images")
ros
"master"
camera
viewer

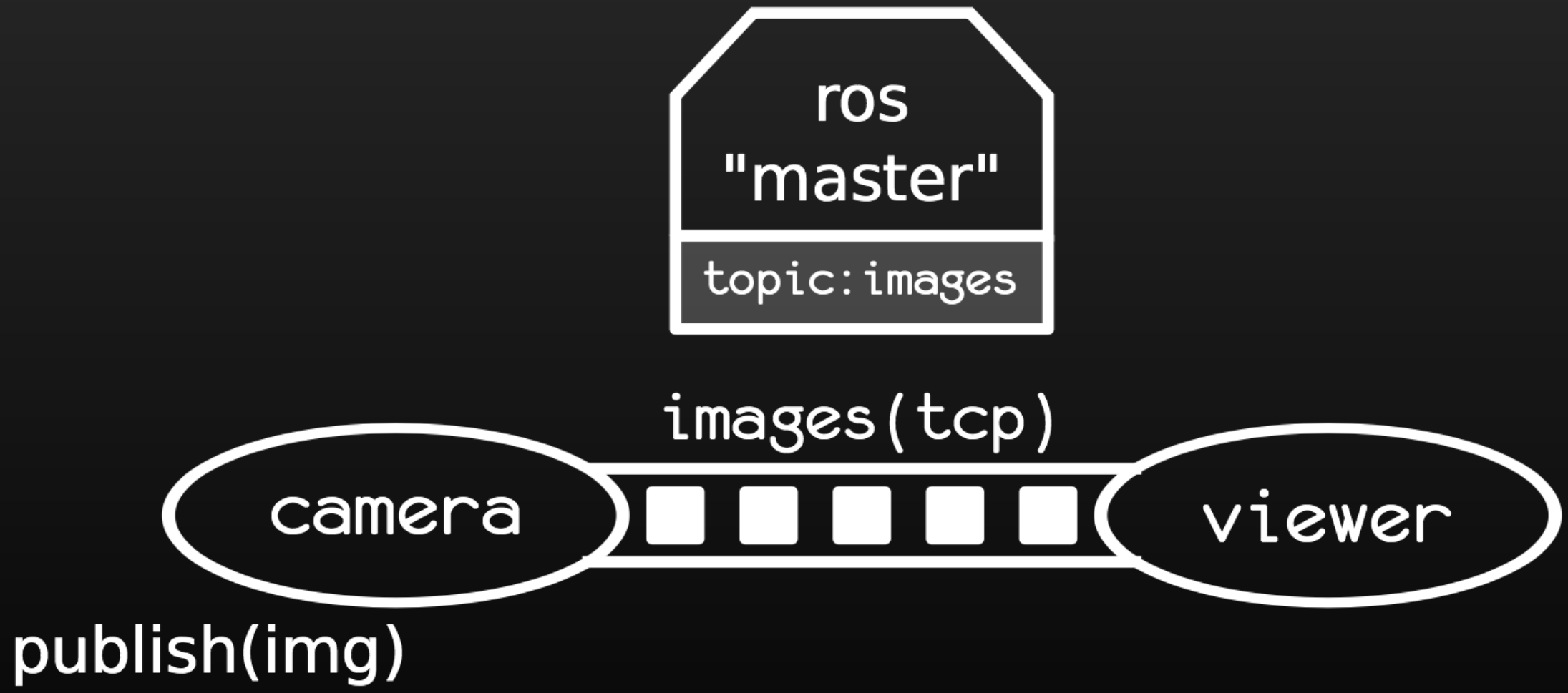

ros
"master"
topic:images
camera
viewer

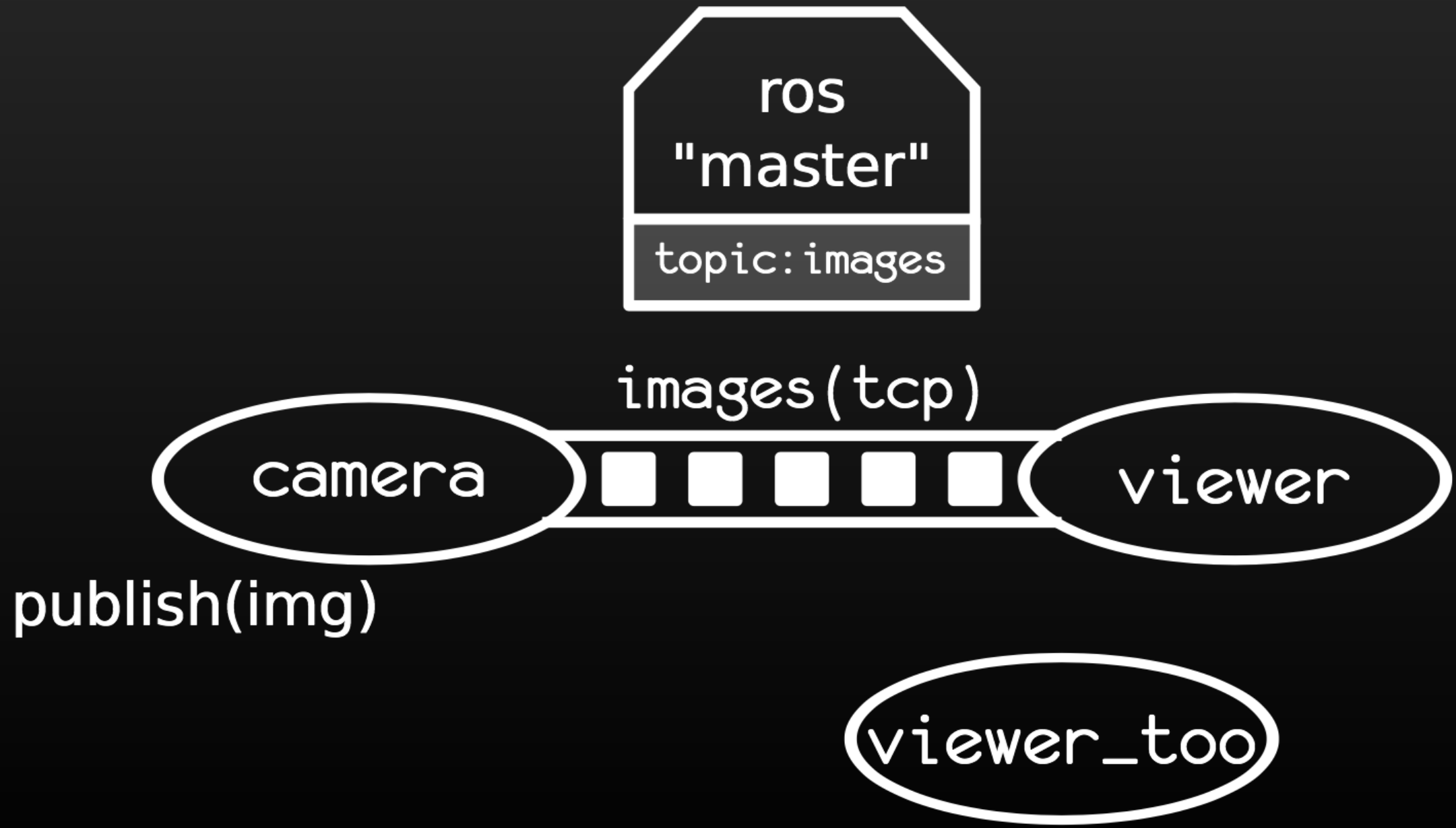

ros
"master"
topic:images
subscribe("images")
camera
viewer

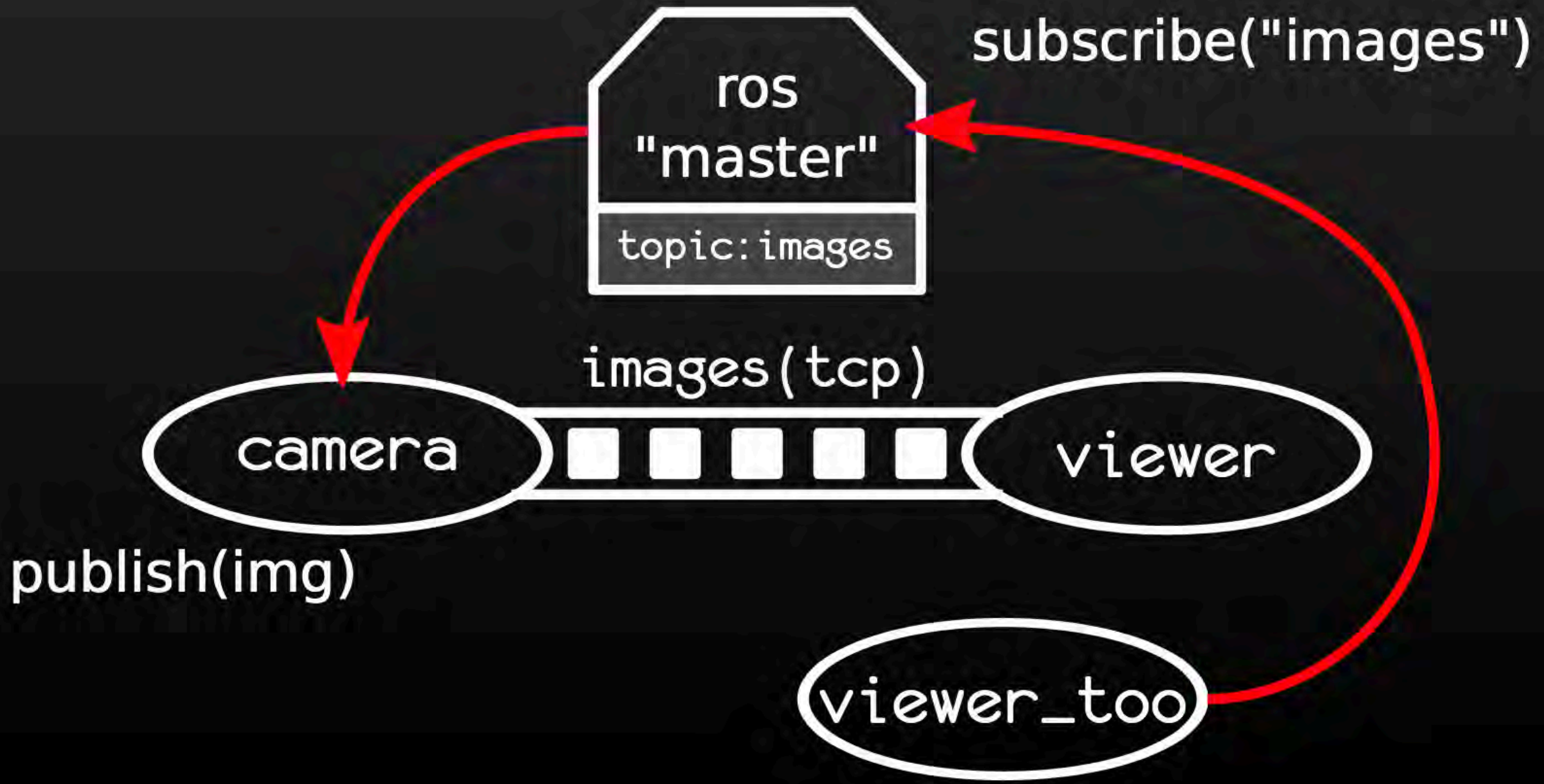

ros
"master"
topic:images
subscribe("images")
camera
viewer

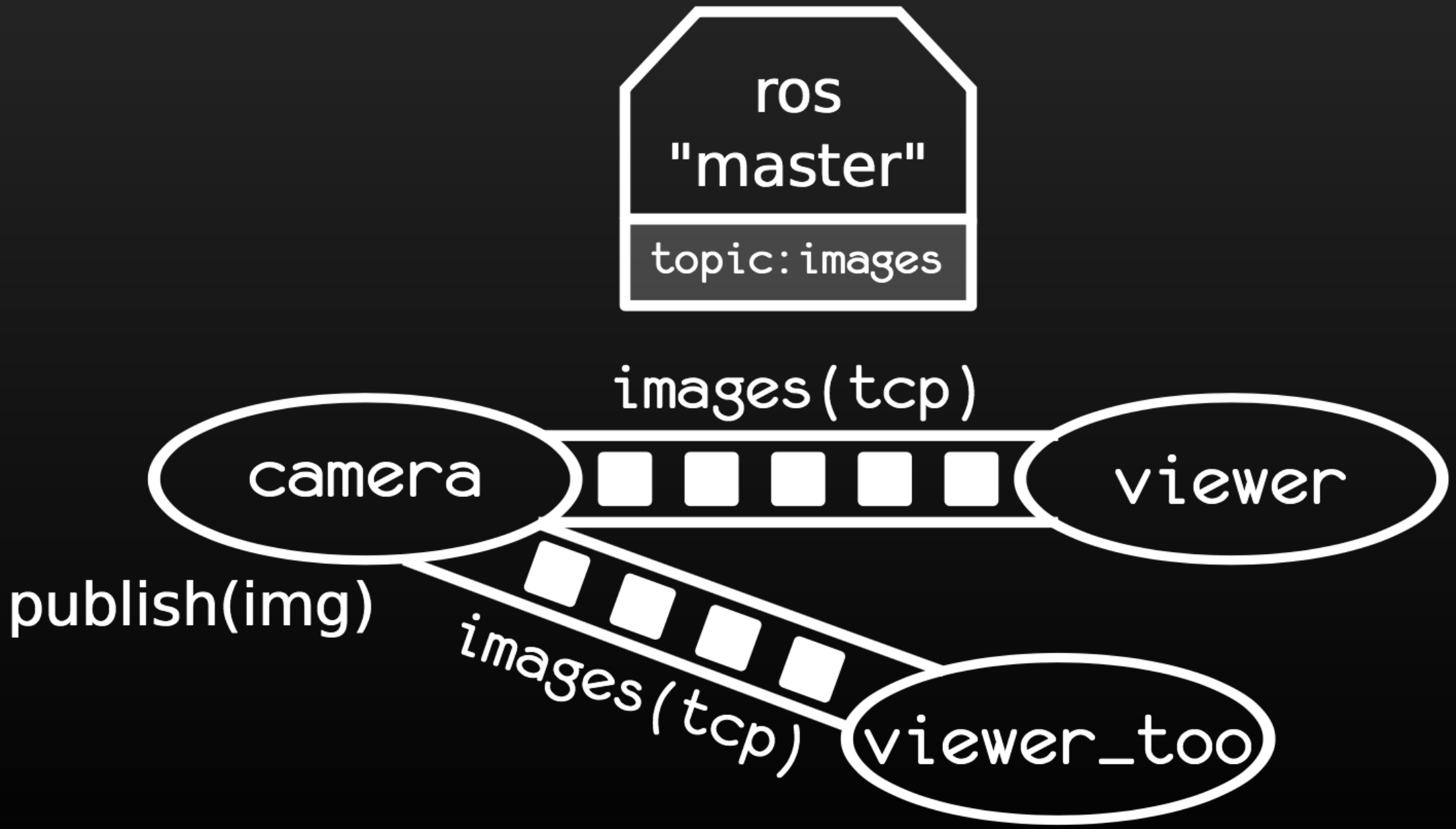

ros
"master"
topic:images
images(tcp)
camera
viewer


ros
"master"
topic:images
images(tcp)
camera
viewer
publish(img)


ros
"master"
topic:images
images(tcp)
camera
viewer
publish(img)


ros
"master"
topic:images
images(tcp)
camera
viewer
publish(img)
viewer_too


ros
"master"
topic:images
subscribe("images")
images(tcp)
camera
viewer
publish(img)
viewer_too


ros
"master"
topic:images
images(tcp)
camera
viewer
publish(img)
images(tcp)
viewer_too

Basic ROS commands: `roscore`

`roscore` is the first thing that you should run when starting ROS.

```
$ roscore
```

Collection of nodes and programs that are pre-requisites of a ROS-based system.

It starts up:

- The ROS Master
- A rosout logging node.

Basic ROS commands: rosrun

`rosrun` executes a ROS node.

```
$ rosrun <package_name> <node_name>
```

Example

```
$ rosrun hokuyo_node hokuyo_node
```

Basic ROS commands: `rosnode`

Command	Description
`rosnode list`	List all active nodes
`rosnode info node_name`	Display information about a node
`rosnode kill node_name`	Kill running node
`Rosnode ping node_name`	Test connectivity to an active node

Basic ROS commands: `rostopic`

Command	Description
`rostopic list`	List all topics currently subscribed to and/or publishing
`rostopic info <topic>`	Show topic message type, subscribers, publishers etc.
`rostopic echo <topic>`	Echo messages published to the topic on the terminal window
`rostopic find <message_type>`	Find topics of the given message type

ROS Client Libraries

Client Library	Language	Comments
roscpp	C++	Most widely used, high performance
rospy	Python	Good for rapid-prototyping and non-critical-path code
roslisp	LISP	Used for planning libraries
rosjava	Java	Android support
roslua	Lua	Light-weight scripting
roscs	Mono/.Net	Any Mono/.Net language
roseus	EusLisp	
PhaROS	Pharo Smalltalk	
rosR	R	Statistical programming

Experimental

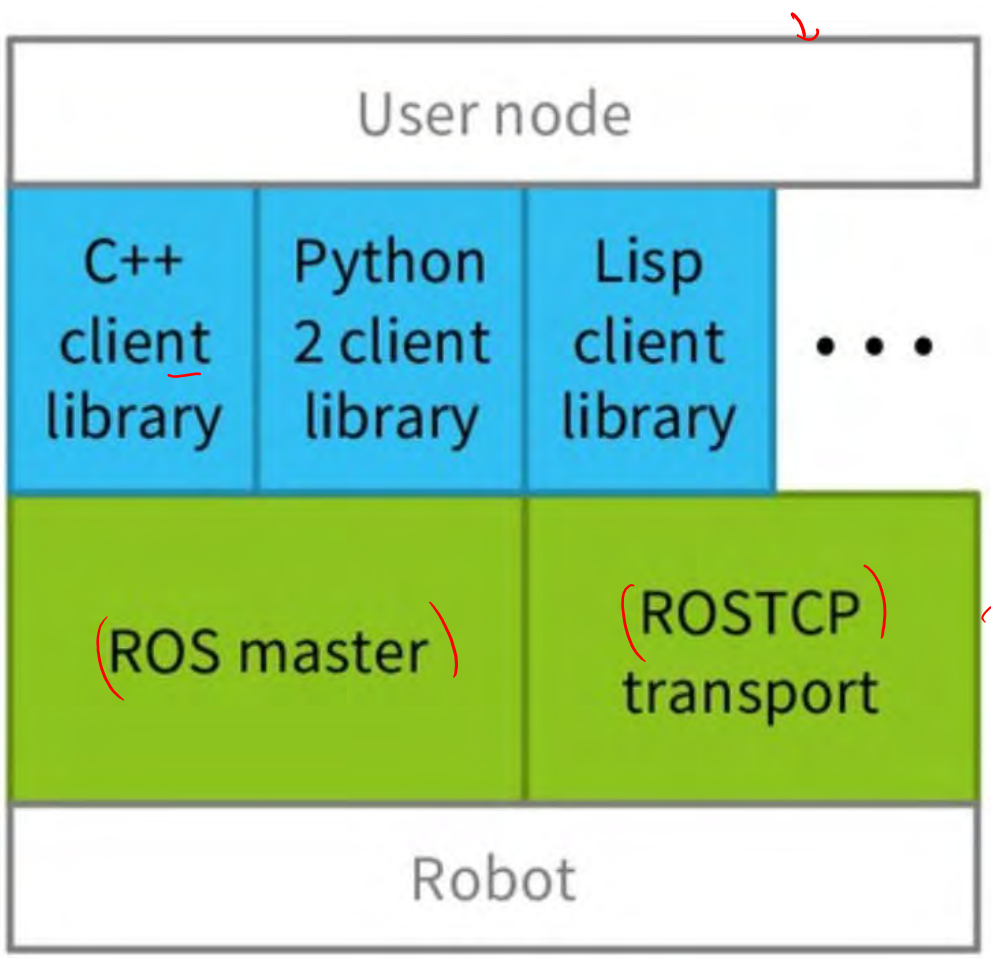
Client API Commonly Used Features

Object / Feature	Description	roscpp	rospy
API root	Objects and methods for interacting with ROS	ros::NodeHandle	rospy
Parameter server client	Query and set parameter server dictionary entries	.getParam .param .searchParam .setParam	.get_param .search_param .set_param
Subscriber	Receive messages from a topic	.subscribe	.Subscriber
Publisher	Send messages to a topic	.advertise	.Publisher
Service	Serve and call remote procedures	.advertiseService .serviceClient	.Service .ServiceProxy
Timer	Periodic interrupt	.createTimer	.Timer
Logging	Output strings to rosconsole	ROS_DEBUG, ROS_INFO, ROS_WARN, etc.	.logdebug, .loginfo, .logwarn, .logerr, .logfatal
Initialization & Event Loop	Set node name, contact Master, enter main event loop	ros::init .spin	.init_node .spin
Messages	Create and extract data from ROS messages	Specifics depends on message	
		std_msgs::String	std_msgs.msg.String

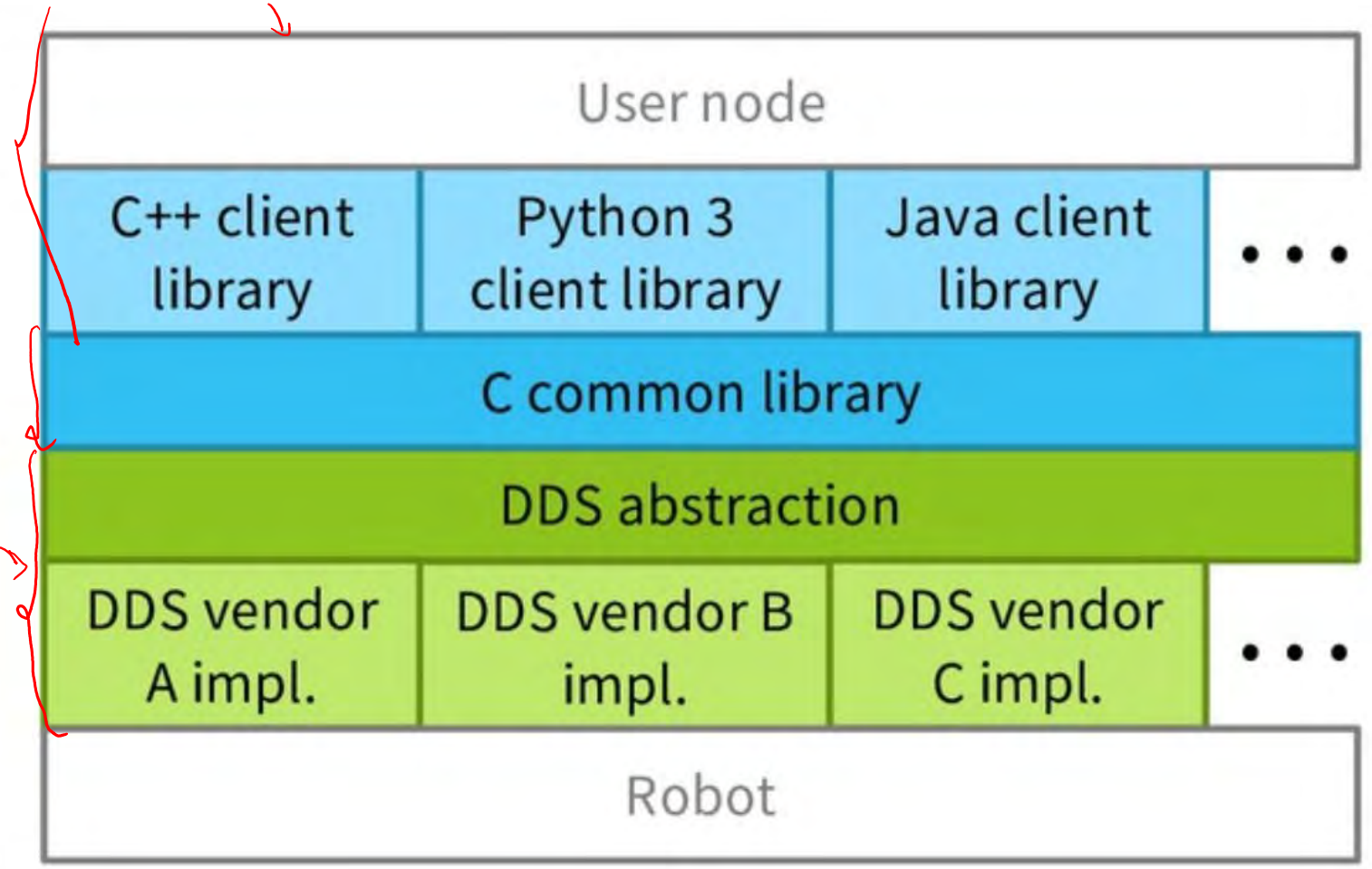
ROS1 vs ROS2.

- ROS1: ROS Noetic (release date: 2020) is the last ROS1 version
- Provides Python3 support
- ROS Noetic’s EOL (End of Life) is scheduled for 2025
- For ROS2, the current LTS version is Foxy Fitzroy (release date: 2020)

ROS1


User node
C++ client library
Python 2 client library
Lisp client library
• • •
ROS master
ROSTCP transport
Robot

ROS2


User node
C++ client library
Python 3 client library
Java client library
• • •
C common library
DDS abstraction
DDS vendor A impl.
DDS vendor B impl.
DDS vendor C impl.
• • •
Robot

Some key features differences between ROS and ROS2

Features	ROS	ROS2
Platforms	Tested on Ubuntu Maintained on other Linux flavors as well as OS X	ROS2 is currently being CI tested and supported on Ubuntu Xenial, OS X El Capitan as well as Windows 10
C++	C++03 // don't useC++11 features in its API	Mainly uses C++11 Start and plan to use C++14 & C++17
Python	Target Python 2	>= Python 3.5
Middleware	Custom serialization format (transport protocol + central discovery mechanism)	Currently all implementations of this interface are based on the DDS standard.
Unify duration and time types	The duration and time types are defined in the client libraries, they are in C++ and Python	In ROS2 these types are defined as messages and therefore are consistent across languages.
Components with life cycle	In ROS every node usually has its own main function.	The life cycle can be used by tools like roslaunch to start a system composed of many components in a deterministic way.
Threading model	In ROS the developer can only choose between single-threaded execution or multi-threaded execution.	In ROS2 more granular execution models are available and custom executors can be implemented easily.
Multiple nodes	In ROS it is not possible to create more than one node in a process.	In ROS2 it is possible to create multiple nodes in a process.
roslaunch	In ROS roslaunch files are defined in XML with very limited capabilities.	In ROS2 launch files are written in Python which enables to use more complex logic like conditionals etc.

But what about the turtles ?




Next time

TurtleSim

TeleOperation

Practice Session 1