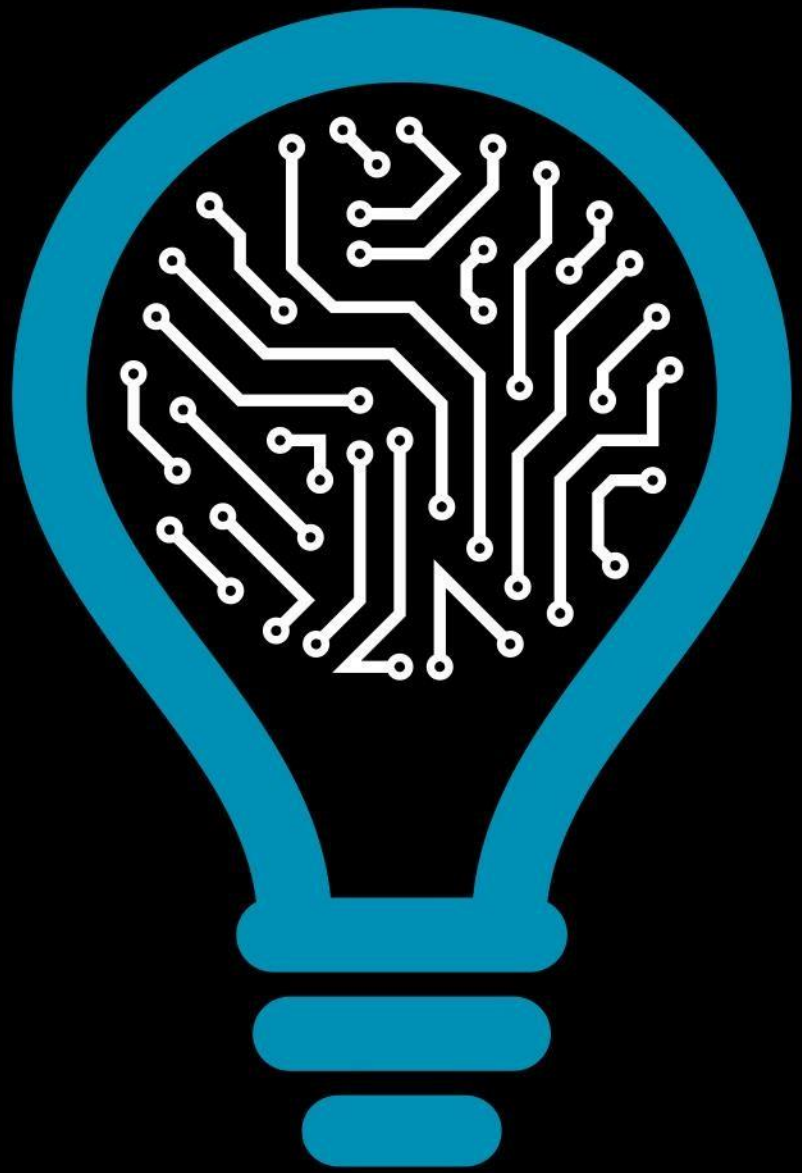
“SAFER” Project
ROS Workshop

Rafael Arrais & Armando Sousa

INESCTEC

INSTITUTE FOR SYSTEMS
AND COMPUTER ENGINEERING,
TECHNOLOGY AND SCIENCE



INESCTEC

SAFER Robotics Workshop

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

INESCTEC

SAFER Robotics Workshop

1. **What is ROS**
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS: Powering the World's Robots

Not really an Operating System!

*"The **Robot ~~Operating System~~** (ROS) is a flexible **framework** for writing robot software.*

*It is a **collection** of **tools**, **libraries**, and **conventions** that aim to **simplify the task of creating complex and robust robot behavior** across a **wide variety of robotic platforms**."*

(Adapted from http://www.ros.org/about-ros/)

What is it?

A 'Meta' OS. Open Source!

– Sits on top of Linux (preferably Ubuntu)
– Works in Windows SubSystem Linux (WSL)

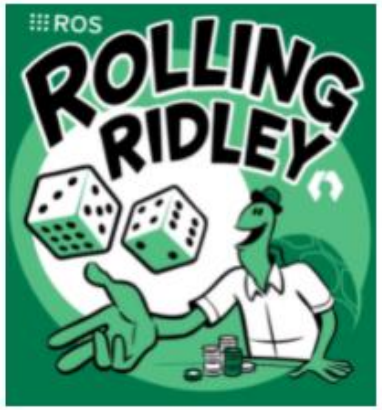
• Agent based
• Message passing
– Publish / Subscribe
– Service (remote operation) invocation

• Package Management
• Name and Parameter Services
• Programming Language Support
– C++
– Python
– Lisp?

Active ROS 2 distributions

Recommended

Development



Active ROS 1 distributions

Recommended

https://docs.ros.org/

ROS: Powering the World's Robots

*"**Why?** Because creating truly **robust**, **general-purpose robot software** is **hard**.*

*From the robot's perspective, problems that seem trivial to humans often **vary wildly** between **instances of tasks and environments**. **Dealing with these variations is so hard** that **no single individual**, **laboratory**, or **institution** can **hope to do it on their own**."*

(Adapted from http://www.ros.org/about-ros/)

ROS: Powering the World's Robots

*"As a result, **ROS** was built from the ground up to **encourage collaborative robotics software development**.*

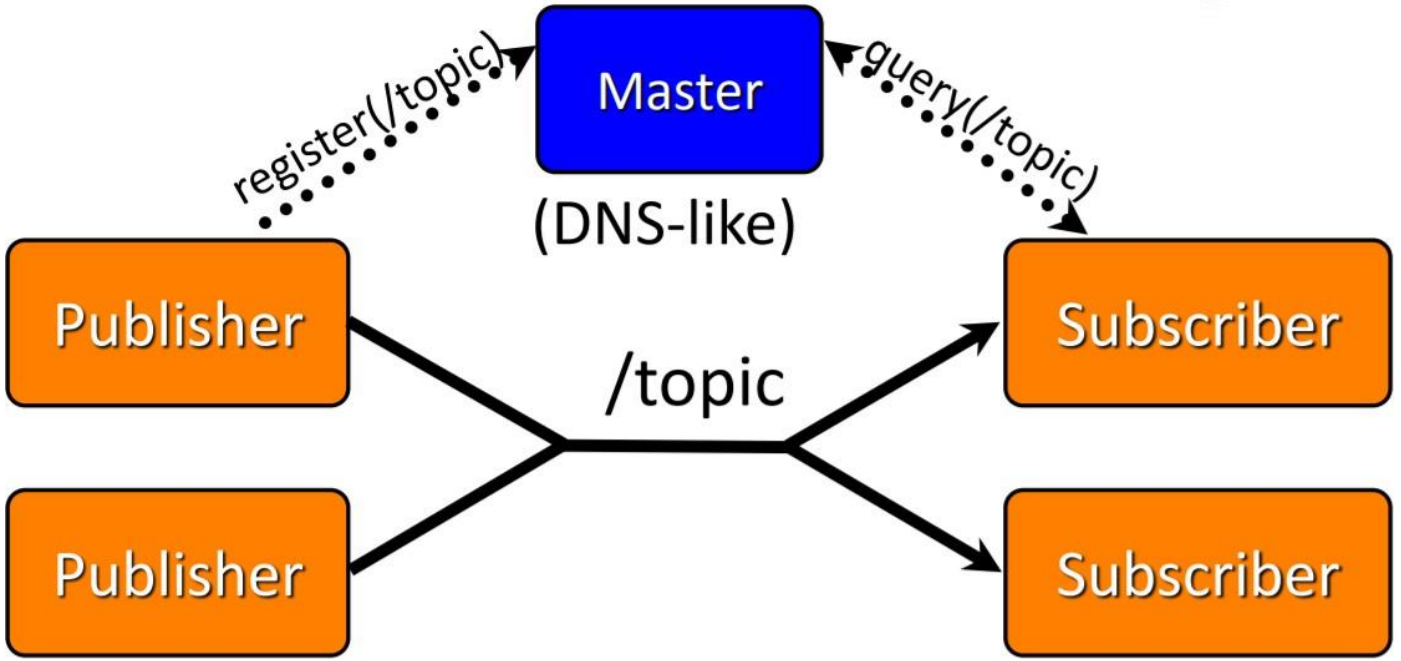
*For example, one laboratory might have experts in **mapping indoor environments**, and could contribute a world-class system for producing maps. Another group might have experts at **using maps to navigate**, and yet another group might have discovered a **computer vision approach** that works well for recognizing small objects in clutter.*

ROS was designed specifically for groups like these to collaborate and build upon each other's work (…)"

(Adapted from http://www.ros.org/about-ros/)

What is ROS?

What is ROS? ROS is… Plumbing!



(Adapted from Willow Garage's "What is ROS?" Presentation)

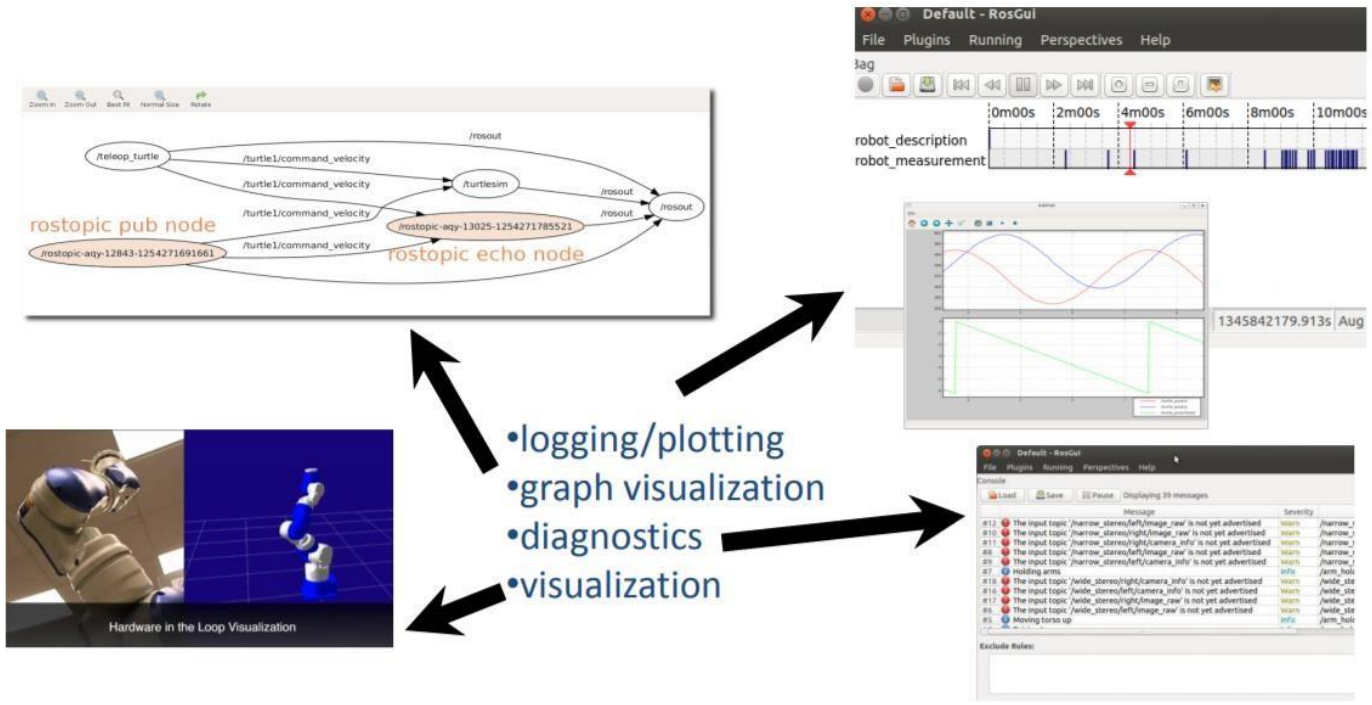
What is ROS? ROS is… Plumbing!

ROS Drivers Examples

- Perception Systems (2D/3D Cameras)
- Laser Scanners
- Robot Actuators
- Inertial Units
- Audio
- GPS
- Joysticks
- …

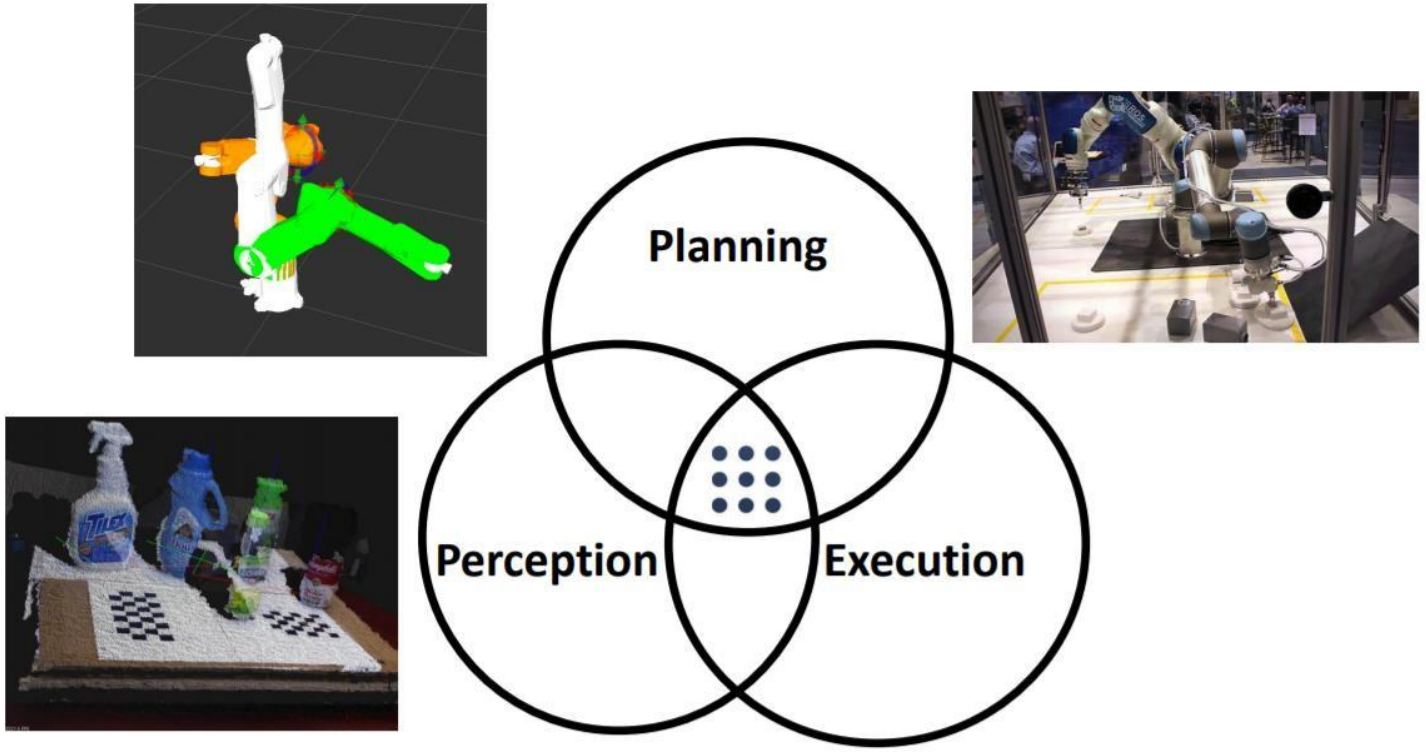
(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)

What is ROS? ROS is… Tools!



(Adapted from Willow Garage's "What is ROS?" Presentation)

What is ROS? ROS is… Capabilities!



(Adapted from Willow Garage's "What is ROS?" Presentation)

What is ROS? ROS is… an Ecosystem!

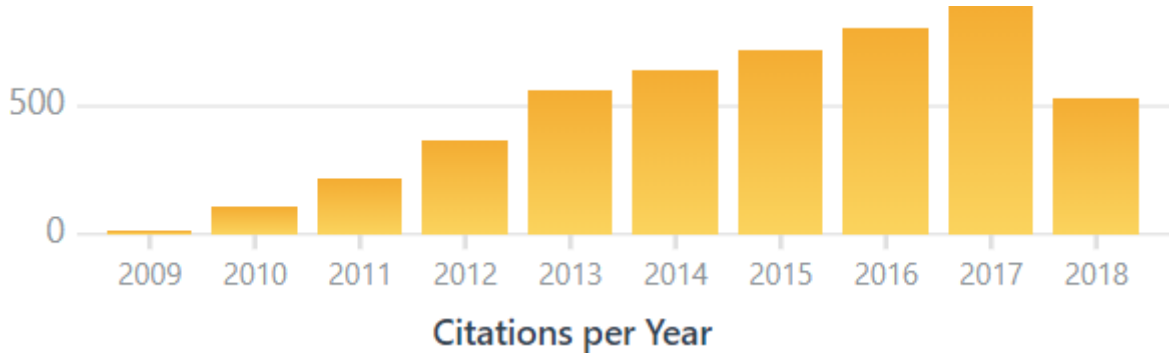
(Adapted from ROS ROS-I Basic Developers Training)

ROS is also… Growing (Scientifically)!

Total Number of Papers Citing “ROS: an open-source Robot Operating System” *(Quigley et al., 2009)*:

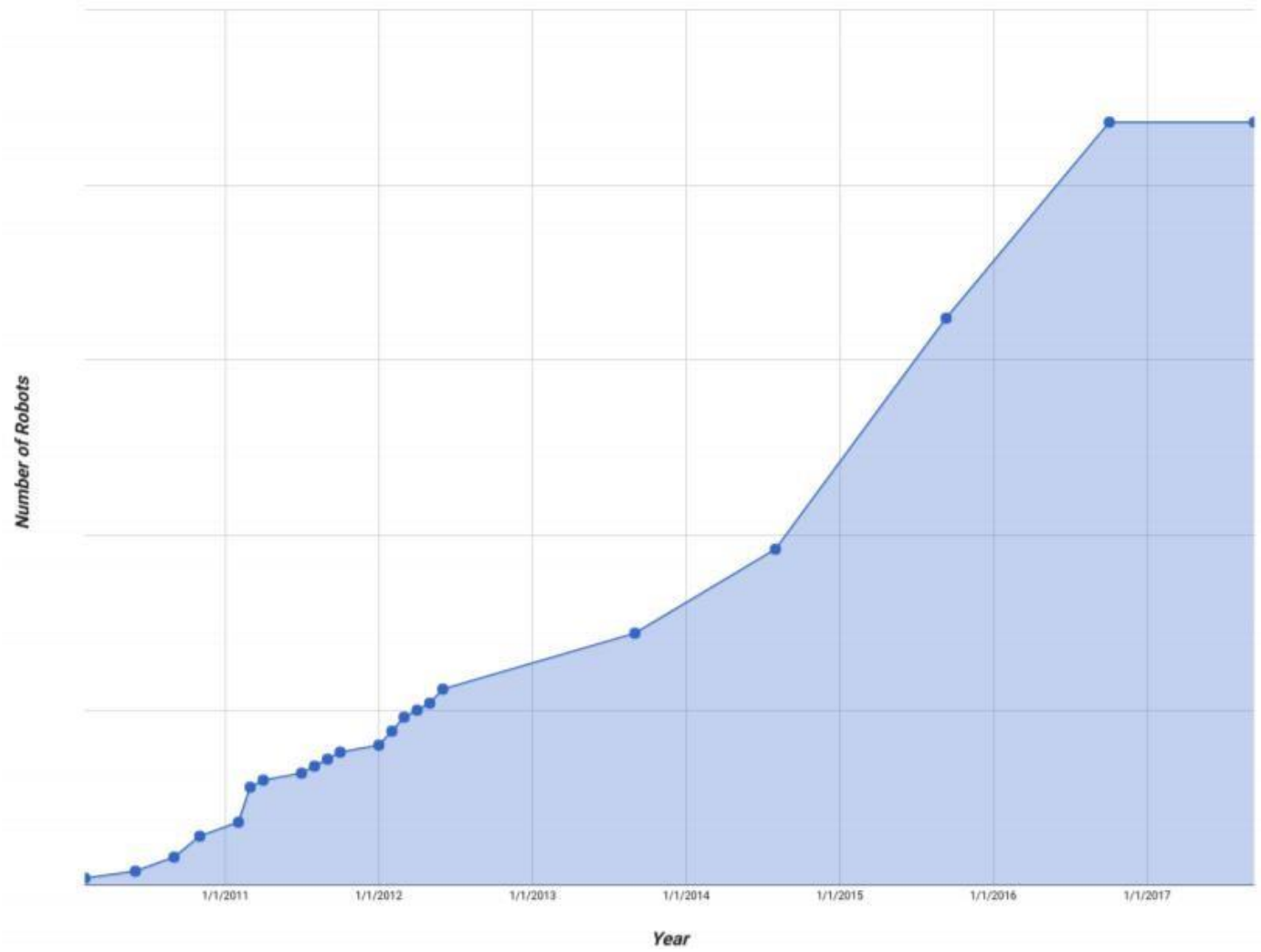
4652

(as of September 2018)



ROS is also… Growing (Commercially)!

Number of Different **Types of Robots** Available to the community with ROS drivers *(at robots.ros.org)* ❼

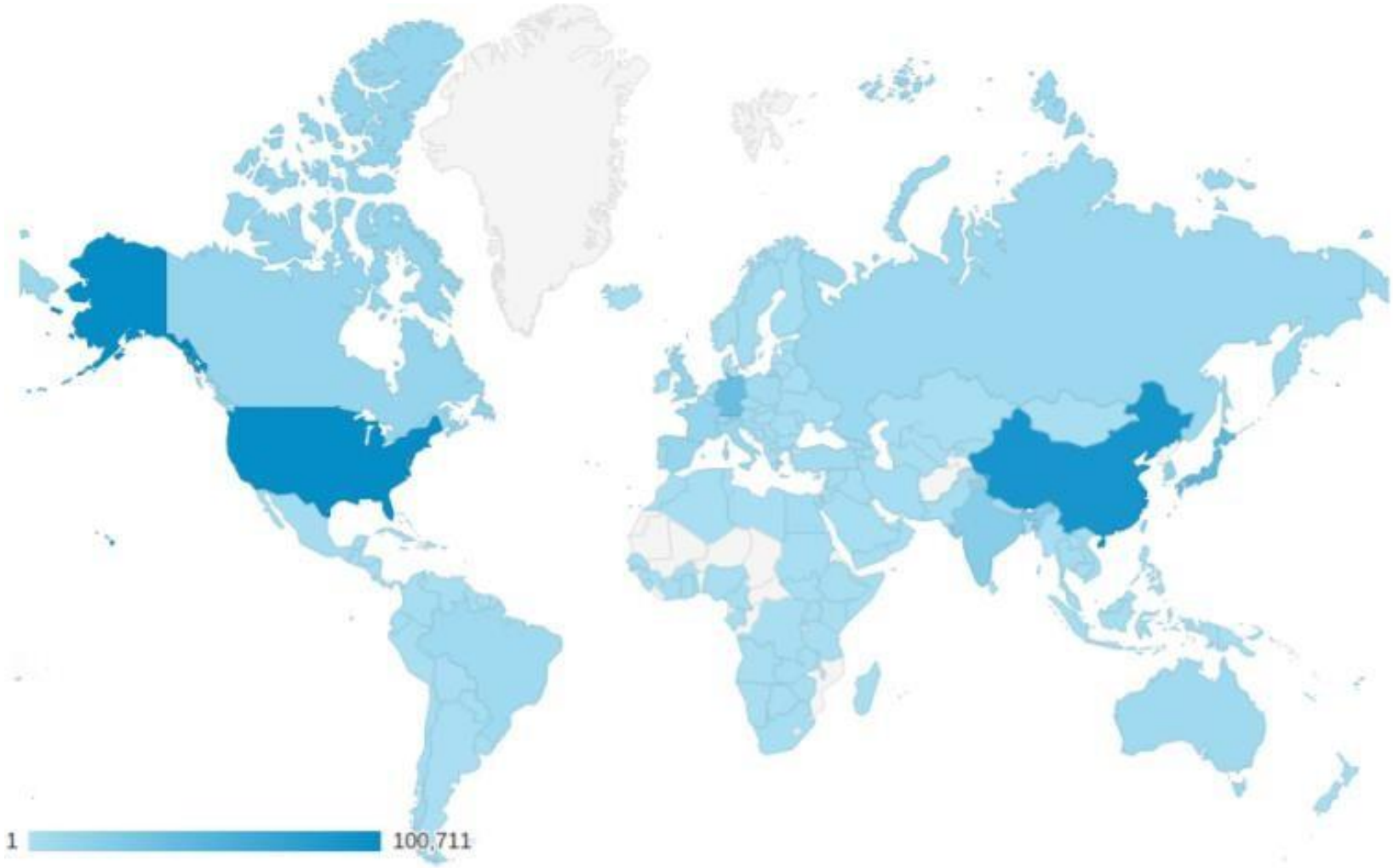


(Adapted from ROS Wiki July 2017 Metrics: *wiki.ros.org/Metrics*)

ROS is also… International!

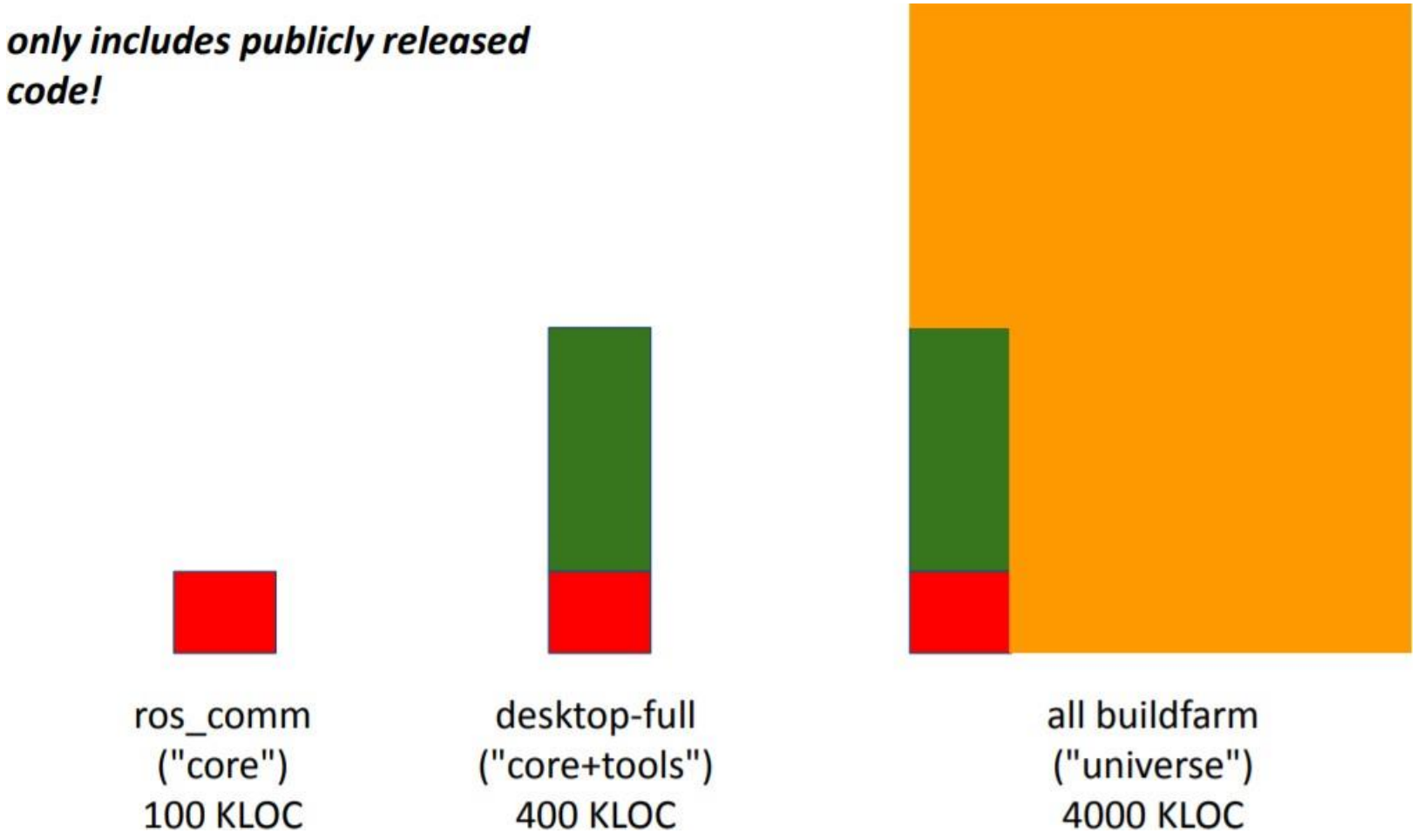
1.	United States	100,711 (20.08%)
2.	China	90,120 (17.97%)
3.	Japan	45,834 (9.14%)
4.	Germany	39,590 (7.89%)
5.	India	20,632 (4.11%)
6.	South Korea	16,683 (3.33%)
7.	United Kingdom	12,784 (2.55%)
8.	Taiwan	11,809 (2.35%)
9.	Canada	11,685 (2.33%)
10.	France	11,651 (2.32%)
11.	Spain	10,445 (2.08%)
12.	Singapore	9,751 (1.94%)
13.	Italy	9,366 (1.87%)
14.	Hong Kong	9,289 (1.85%)
15.	Russia	8,380 (1.67%)
16.	Australia	6,346 (1.27%)
17.	Brazil	5,959 (1.19%)
18.	Switzerland	4,474 (0.89%)
19.	Turkey	4,399 (0.88%)
20.	Netherlands	4,343 (0.87%)
21.	Poland	4,176 (0.83%)
22.	Sweden	3,159 (0.63%)
23.	Portugal	3,150 (0.63%)

wiki.ros.org Visitor Locations:



(Adapted from ROS Wiki July 2017 Metrics: *wiki.ros.org/Metrics*)

ROS is also… a set of Repositories & Packages!



(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)

ROS is also… a set of Repositories & Packages!

Top 40 Downloaded Packages

Direct downloads from packages.ros.org - July 2017

python-catkin-pkg	45940	ros-kinetic-geometry-msgs	27900
python-catkin-pkg-modules	45781	ros-kinetic-rospy	27852
python-rosdep	43695	ros-kinetic-rosout	27761
python-rospkg	40876	ros-kinetic-topic-tools	27564
python-rospkg-modules	40736	ros-kinetic-rosmsg	27530
python-rosdistro	39924	ros-kinetic-rosnode	27508
python-rosdistro-modules	39769	ros-kinetic-diagnostic-aggregator	27461
ros-kinetic-diagnostic-updater	29612	ros-kinetic-image-transport	27416
ros-kinetic-roslisp	29000	ros-kinetic-urdf-parser-plugin	27405
ros-kinetic-ros-comm	28957	ros-kinetic-urdf	27389
ros-kinetic-ros-base	28898	ros-kinetic-std-srvs	27339
ros-kinetic-tf2	28843	ros-kinetic-cv-bridge	27318
ros-kinetic-tf2-msgs	28815	ros-indigo-rviz	27307
ros-kinetic-tf2-ros	28781	ros-kinetic-rosbuild	27272
ros-kinetic-tf2-py	28746	ros-kinetic-std-msgs	27270
ros-kinetic-tf	28677	ros-kinetic-sensor-msgs	27257
ros-kinetic-geneus	28365	ros-kinetic-nav-msgs	27253
ros-kinetic-ros-core	28352	ros-kinetic-roscpp	27241
ros-kinetic-message-generation	28199	ros-kinetic-rosconsole	27229
ros-kinetic-message-filters	27991	ros-kinetic-rosgraph-msgs	27194

(Adapted from ROS Wiki July 2017 Metrics: *wiki.ros.org/Metrics*)

ROS is also… Celebrating 10 years!

https://vimeo.com/245826128

(Adapted from ROS Wiki July 2017 Metrics: https://wiki.ros.org/Metrics)

INESCTEC

SAFER Robotics Workshop

1. What is ROS
2. **ROS Distros, Installation, Programming Languages**
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS 1 Versions: Annual Releases (“distributions”)

Box Turtle

Mar 2010

C Turtle

Aug 2010

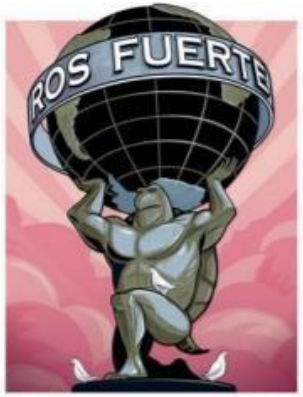
Diamondback

Mar 2011

Electric

Aug 2011

LTS



Fuerte

April 2012

Groovy

2012 - 2014

Hydro

2013 - 2015

LTS

Indigo

2014 - 2019

Jade

2015 - 2017

LTS

Kinetic

2016 - 2021

Lunar

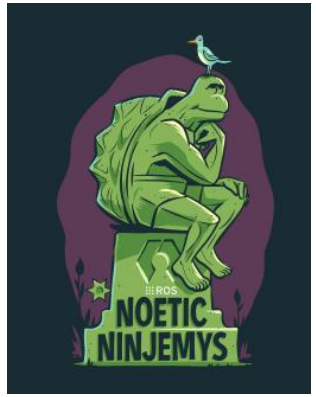
2017 - 2019

LTS

Melodic

2018 - 2023

LTS



2020 - 2025

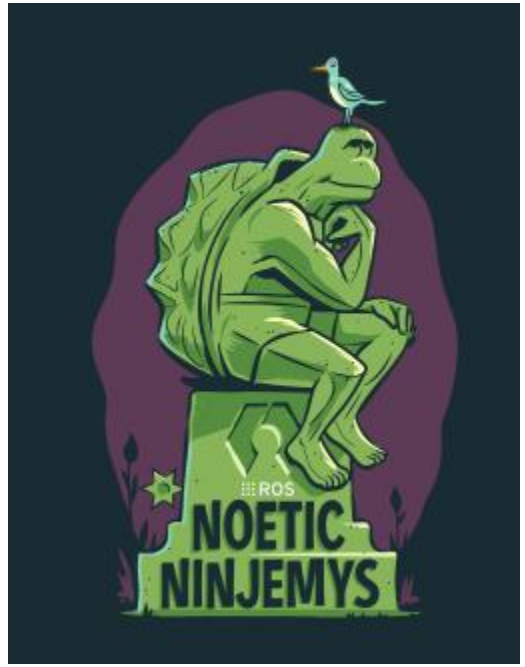
(Adapted from ROS ROS-I Basic Developers Training)

Installing ROS

- ROS Wiki provides a detailed guide on installing and configuring ROS 1:

http://wiki.ros.org/noetic/Installation

[ROS Noetic Ninjemys is primarily targeted at the Ubuntu 20.04 (Focal) release]



Ubuntu	Focal	amd64	armhf	arm64			
Debian	Buster	amd64	arm64				

http://wiki.ros.org/noetic/Installation/Source

Windows	10	amd64					
Arch Linux	Any	amd64	i686	arm	armv6h	armv7h	aarch64

ROS Programming

- ROS uses **platform-agnostic** methods for most communication:
  - TCP/IP Sockets, XML, etc.

- Can intermix **Programming Languages**:
  - Primary: **C++**, **Python**, ~~Lisp~~
  - Also: ~~C#~~, ~~Java~~, ~~Matlab~~, ~~JavaScript~~, ~~etc.~~

Programming Languages other than C++ and Python are not really used a lot…

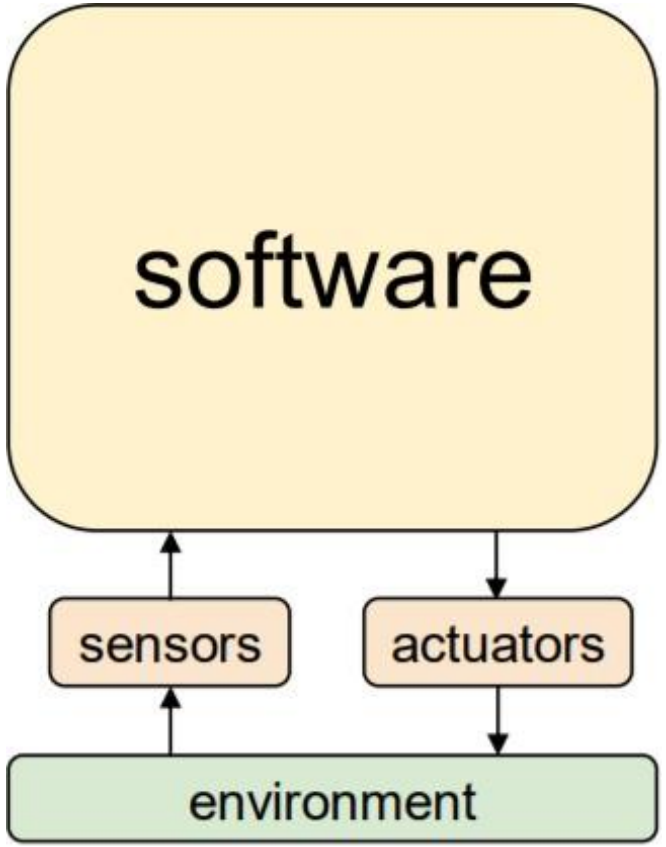
(Adapted from ROS ROS-I Basic Developers Training)

INESCTEC

SAFER Robotics Workshop

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. **ROS Architecture**
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

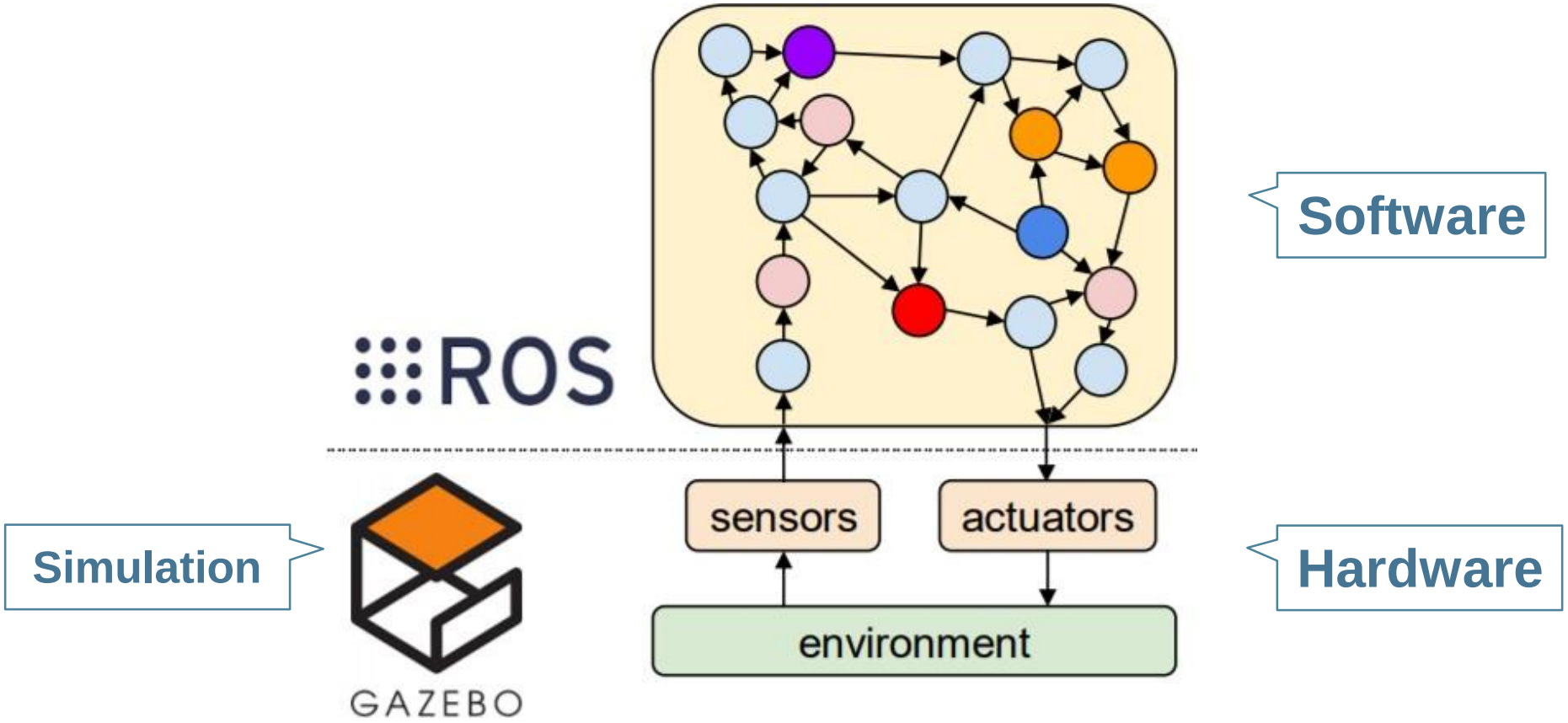
ROS: High Level Architectural Overview



- **Robots** are **Cyber-Physical Systems**:
  - ROS acts as a software framework for enabling Sensors and Actuators to interact with the Physical Environment.

(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)

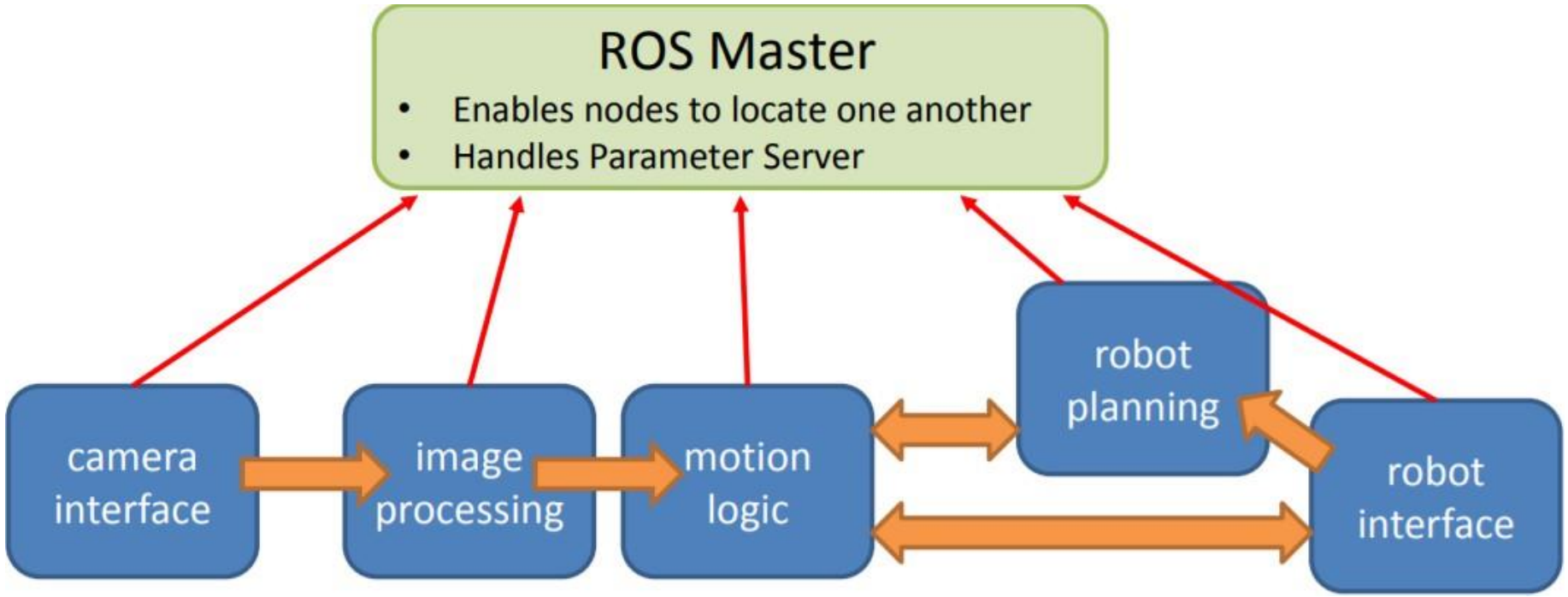
ROS: High Level Architectural Overview



- Break Complex Software into Smaller Pieces
- Provide a framework, tools, and interfaces for distributed development
- Encourage re-use of software pieces
- Easy transition between simulation and hardware

(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)

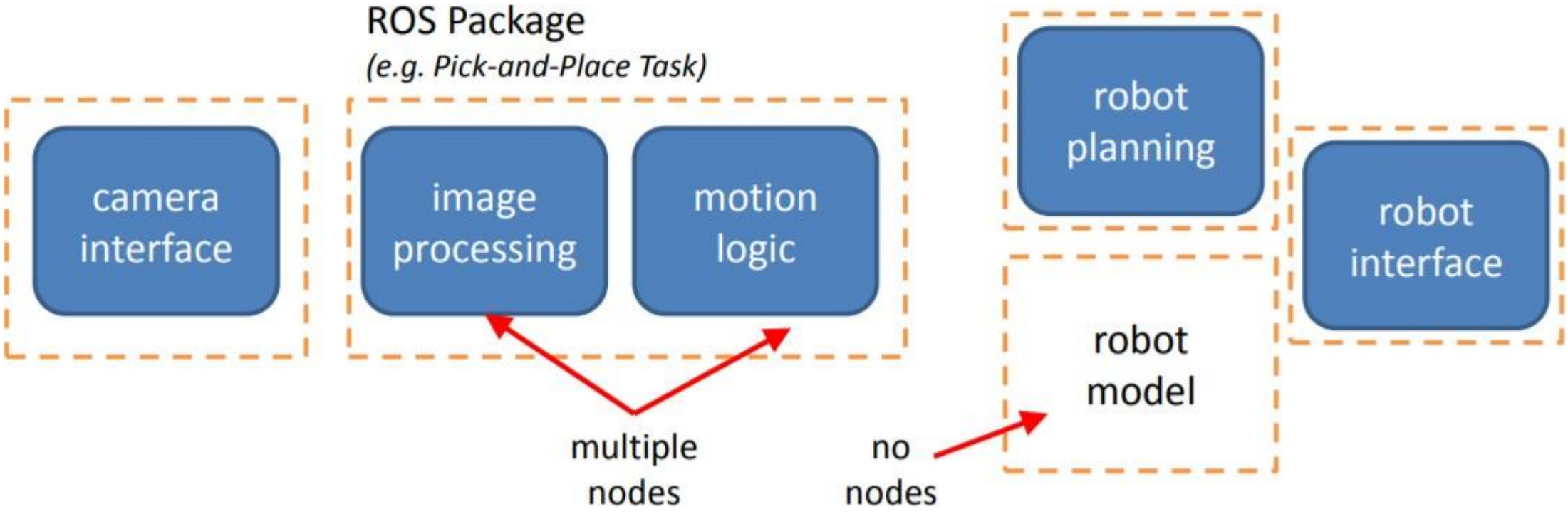
ROS Architecture: Nodes



- A **Node** is a single ROS-enabled program:
  - Most communication happens **between** nodes.
  - Nodes can run on many different **devices.**
- One **Master** per system.

(Adapted from ROS ROS-I Basic Developers Training)

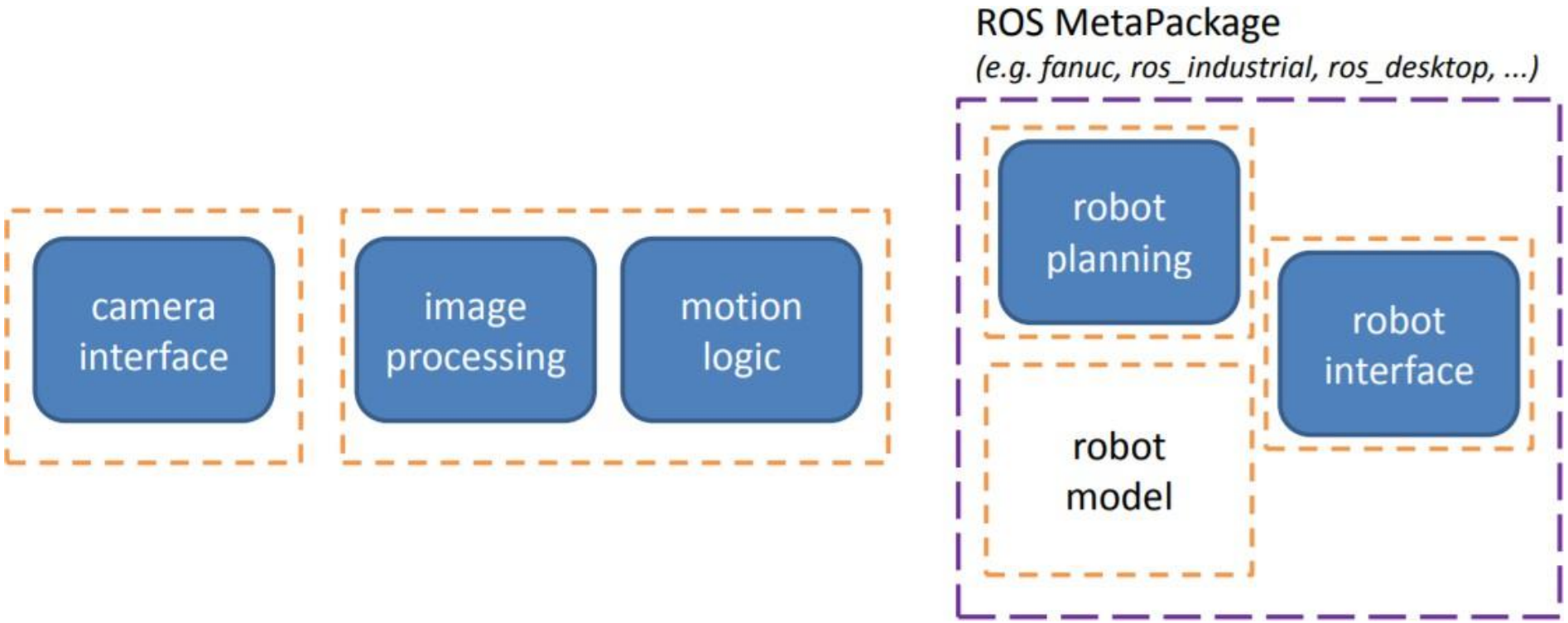
ROS Architecture: Packages



- ROS **Packages** are groups of related nodes/data
  - Many ROS commands are **package-oriented**

(Adapted from ROS ROS-I Basic Developers Training)

ROS Architecture: MetaPkg



- **MetaPackages** are groups of related packages
  - Mostly for convenient install/deployment

(Adapted from ROS ROS-I Basic Developers Training)

ROS Build System: Catkin

- ROS uses the **catkin** build system:
  - based on CMAKE

Only in theory… Practical use is linux (Ubuntu) based.

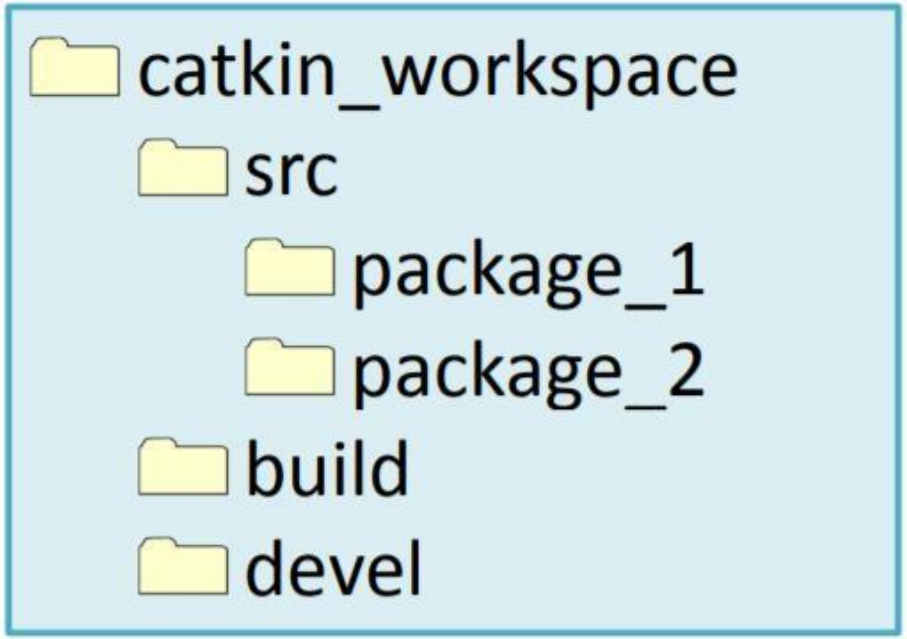
  - cross-platform (Ubuntu, Windows, embedded...)
  - replaces older *rosbuild* system:
    - different build commands, directory structure, etc.
    - most packages have already been upgraded to *Catkin*
    - *rosbuild: manifest.xml, catkin: package.xml*

Today there are very few ROS packages that still used *rosbuild*. The change from *rosbuild* to *Catkin* occurred around **5 years ago**.

(Adapted from ROS ROS-I Basic Developers Training)

ROS Build System: Catkin Workspace

- **Catkin** uses a specific directory structure:
  - Each “project” typically gets its own **catkin workspace**
  - All packages/source files go in the **src** directory
  - Temporary build-files are created in **build**
  - Results are placed in **devel**



(Adapted from ROS ROS-I Basic Developers Training)

ROS Build System: Catkin Build Process

- **Setup (one-time):**
  1. Create a *catkin* workspace somewhere:
     - **catkin_ws**
     - **src** sub-directory must be created manually
     - **build**, **devel** directories created automatically
  2. Run **catkin init** from workspace root
  3. Download/create **packages** in **src** subdir

- **Compile-Time:**
  1. Run **catkin build** anywhere in the workspace
  2. Run **source devel/setup.bash** to make workspace visible to ROS
     - Must re-execute in each new terminal window
     - Can add to ~/.bashrc to automate this process

(Adapted from ROS ROS-I Basic Developers Training)

Adding 3rd-Party Packages: Install Options

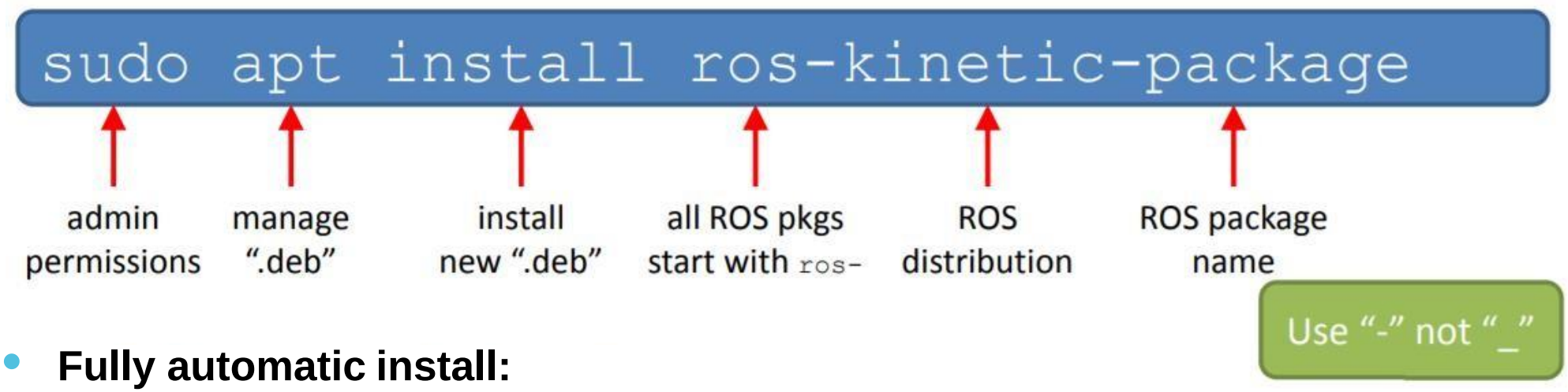
- Two interchangeable options for installing 3rd Party ("Resources") Packages:
  - **Debian Packages:**
    - Nearly "automatic"
    - Recommended for end-users
    - Stable
    - Easy
  - **Source Repositories:**
    - Access "latest" code
    - Most at Github.com
    - More effort to setup
    - (Can be) Unstable

ROS Website (*ros.org/browse*) can be used to browse/search for known packages

The ROS Community is also very active on *Github.com*

(Adapted from ROS ROS-I Basic Developers Training)

Adding 3rd-Party Packages: Install using Debian Packages



- **Fully automatic install:**
  1. Download `.deb` package from central ROS repository
  2. The process automatically copies files to standard locations (`/opt/ros/kinetic/...`)
  3. The process also installs any other required dependencies

- `sudo apt-get remove ros-distro-package`
  - Removes software (but not dependencies!)

(Adapted from ROS ROS-I Basic Developers Training)

Adding 3rd-Party Packages: Install from Source

- **Somewhat manual process:**

1. **Find** GitHub repository.
2. **Clone** repository into your workspace `src` directory:

```
cd catkin_ws/src
git clone http://github.com/user/repo.git
```

3. **Build** your catkin workspace:

```
cd catkin_ws
catkin build
```

4. Now the package and its resources are available to you.

(Adapted from ROS ROS-I Basic Developers Training)

INESCTEC

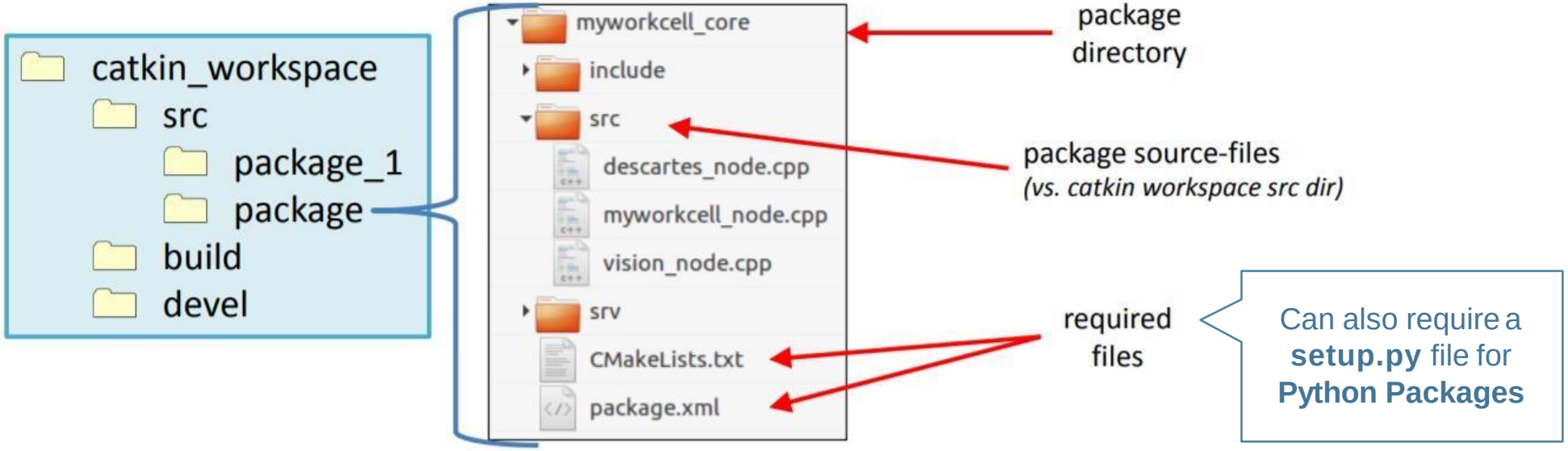
SAFER Robotics Workshop

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. **ROS Packages**
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS Packages: Contents

- ROS components are organized into **packages**

- Packages contain several **required files**:
  - package.xml
    - **metadata** for ROS: *package name, description, dependencies, ...*
  - CMakeLists.txt
    - **build rules** for *catkin*



(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: package.xml

The package manifest is an XML file called package.xml that must be included with any catkin-compliant package's root folder. This file defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages.

- **Metadata**: *name, description, author, license ...*

```
<package>
  <name>myworkcell_core</name>
  <version>0.0.0</version>
  <description>The myworkcell_core package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="ros-industrial@todo.todo">ros-industrial</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>
```


(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: package.xml

- **Metadata**: *name, description, author, license ...*
- **Dependencies:**
  - <buildtool_depend>: Needed to **build** itself. (Typically *catkin*)
  - <depend>: Needed to **build**, **export**, and **execution** dependency. *(format "2" only)*
  - <build_depend>: Needed to **build** this package.
  - <build_export_depend>: Needed to **build against** this package.
  - <exec_depend>: Needed to **run** code in this package.
  - <test_depend>: Only additional dependencies for unit **tests**.
  - <doc_depend>: Needed to generate **documentation**.

Format 2 is the recommended format for new packages.

<build_depend> and **<exec_depend>** are the most commonly used dependency tags within package.xml

(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: package.xml

- **Realistic example:**

```xml
<package format="2">
  <name>foo_core</name>
  <version>1.2.4</version>
  <description>This package provides foo capability.</description>
  <maintainer email="ivana@willowgarage.com">Ivana Bildbotz</maintainer>
  <license>BSD</license>

  <url>http://ros.org/wiki/foo_core</url>
  <author>Ivana Bildbotz</author>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>roscpp</depend>
  <depend>std_msgs</depend>

  <build_depend>message_generation</build_depend>

  <exec_depend>message_runtime</exec_depend>
  <exec_depend>rospy</exec_depend>

  <test_depend>python-mock</test_depend>

  <doc_depend>doxygen</doc_depend>
</package>
```

(Adapted from ROS Wiki: *wiki.ros.org/catkin/package.xml*)

ROS Packages: CMakeList.txt

- The file **CMakeLists.txt** is the **input** to the **CMake build system** for **building software packages**.

- Any CMake-compliant package contains **one or more** `CMakeLists.txt` file that describe **how to build the code** and **where to install it to**.

- The `CMakeLists.txt` file used for a ROS-based *catkin* project is a **standard vanilla** `CMakeLists.txt` file with a **few additional constraints**.

(Adapted from ROS Wiki: *wiki.ros.org/catkin/CMakeLists.txt*)

ROS Packages: CMakeList.txt

- Provides **rules** for **building software**
  - The template file contains many examples

- **include_directories**(include ${catkin_INCLUDE_DIRS})
  - Adds directories to CMAKE include rules

- **add_executable**(myNode src/myNode.cpp src/widget.cpp)
  - Builds program myNode, from myNode.cpp and widget.cpp

- **target_link_libraries**(myNode ${catkin_LIBRARIES})
  - Links node myNode to dependency libraries

(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: CMakeList.txt

- **Realistic Example:**

```
# Get the information about this package's buildtime dependencies
find_package(catkin REQUIRED
  COMPONENTS message_generation std_msgs sensor_msgs)

# Declare the message files to be built
add_message_files(FILES
  MyMessage1.msg
  MyMessage2.msg
)

# Declare the service files to be built
add_service_files(FILES
  MyService.srv
)

# Actually generate the language-specific message and service files
generate_messages(DEPENDENCIES std_msgs sensor_msgs)

# Declare that this catkin package's runtime dependencies
catkin_package(
 CATKIN_DEPENDS message_runtime std_msgs sensor_msgs
)

# define executable using MyMessage1 etc.
add_executable(message_program src/main.cpp)
add_dependencies(message_program ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

# define executable not using any messages/services provided by this package
add_executable(does_not_use_local_messages_program src/main.cpp)
add_dependencies(does_not_use_local_messages_program ${catkin_EXPORTED_TARGETS})
```

(Adapted from ROS Wiki: *wiki.ros.org/catkin/package.xml*)

ROS Packages: setup.py

- If your ROS package contains **Python modules** and **scripts** to **install**, you need to define the **installation process** and a way to make the **scripts accessible** in the develspace.

- The **setup.py** file uses Python to describe the Python content of the stack Catkin allows you to specify the **installation** of your **Python files** in this `setup.py` and reuse some of the information in your `CMakeLists.txt`.

```
from distutils.core import setup
from catkin_pkg.python_setup import generate_distutils_setup

d = generate_distutils_setup(
    packages=['mypkg'],
    scripts=['bin/myscript'],
    package_dir={'': 'src'}
)

setup(**d)
```

(Adapted from ROS Docs: *docs.ros.org/api/catkin/html/user_guide/setup_dot_py.html*)

ROS Packages: Create New Packages

```
catkin create pkg mypkg --catkin-deps dep1 dep2
```

- Easiest way to start a **New Package**:
  - Create directory, required files
  - **mypkg**: name of package to be created
  - **dep 1/2**: dependency package names
    - Automatically added to `CMakeList.txt` and `package.xml`
    - Can manually add additional dependencies later

(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: Other Useful Commands

- **roscd package_name**
  - Change to package directory
- **rospack**
  - **rospack find package_name**
    - Find directory of package_name
  - **rospack list**
    - List all ROS packages installed
  - **rospack depends package_name**
    - List all dependencies of package_name

(Adapted from ROS ROS-I Basic Developers Training)

INESCTEC

SAFER Robotics Workshop

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. **ROS Nodes**
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS Nodes: Overview

- All running nodes have a **graph resource name** that **uniquely identifies** them to the rest of the system.
  - *For example:* /hokuyo_node *could be the name of a Hokuyo driver broadcasting laser scans.*

- Nodes also have a **node type**, that simplifies the process of referring to a node executable on the filesystem.
  - These node types are **package resource names** with the name of the node's package and the name of the node executable file.
  - In order to resolve a node type, ROS searches for all executables in the package with the specified name and chooses the first that it finds.

- A ROS node is written with the use of a ROS client library, such as **roscpp** or **rospy**.

(Adapted from ROS Wiki: *wiki.ros.org/Nodes*)

ROS Nodes: A Simple C++ ROS Node

Simple C++ Program

```
#include <iostream>

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



   std::cout << "Hello World!";

   return 0;
}
```

Simple C++ ROS Node

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

int main(int argc, char* argv[])
{
  ros::init(argc, argv, "hello");
  ros::NodeHandle node;

  ROS_INFO_STREAM("Hello World!");

  return 0;
}
```

(Adapted from ROS ROS-I Basic Developers Training)

ROS Concepts: roscore

- **roscore** is a collection of nodes and programs that are pre-requisites of a ROS-based system

- You must have a **roscore** running in order for ROS nodes to communicate. It is launched using the `roscore` command.

- **roscore** will start up:
  - a ROS Master;
  - a ROS Parameter Server;
  - a *rosout* logging node.

- ***NOTE***: If you use *roslaunch*, it will automatically start *roscore* if it detects that it is not already running.

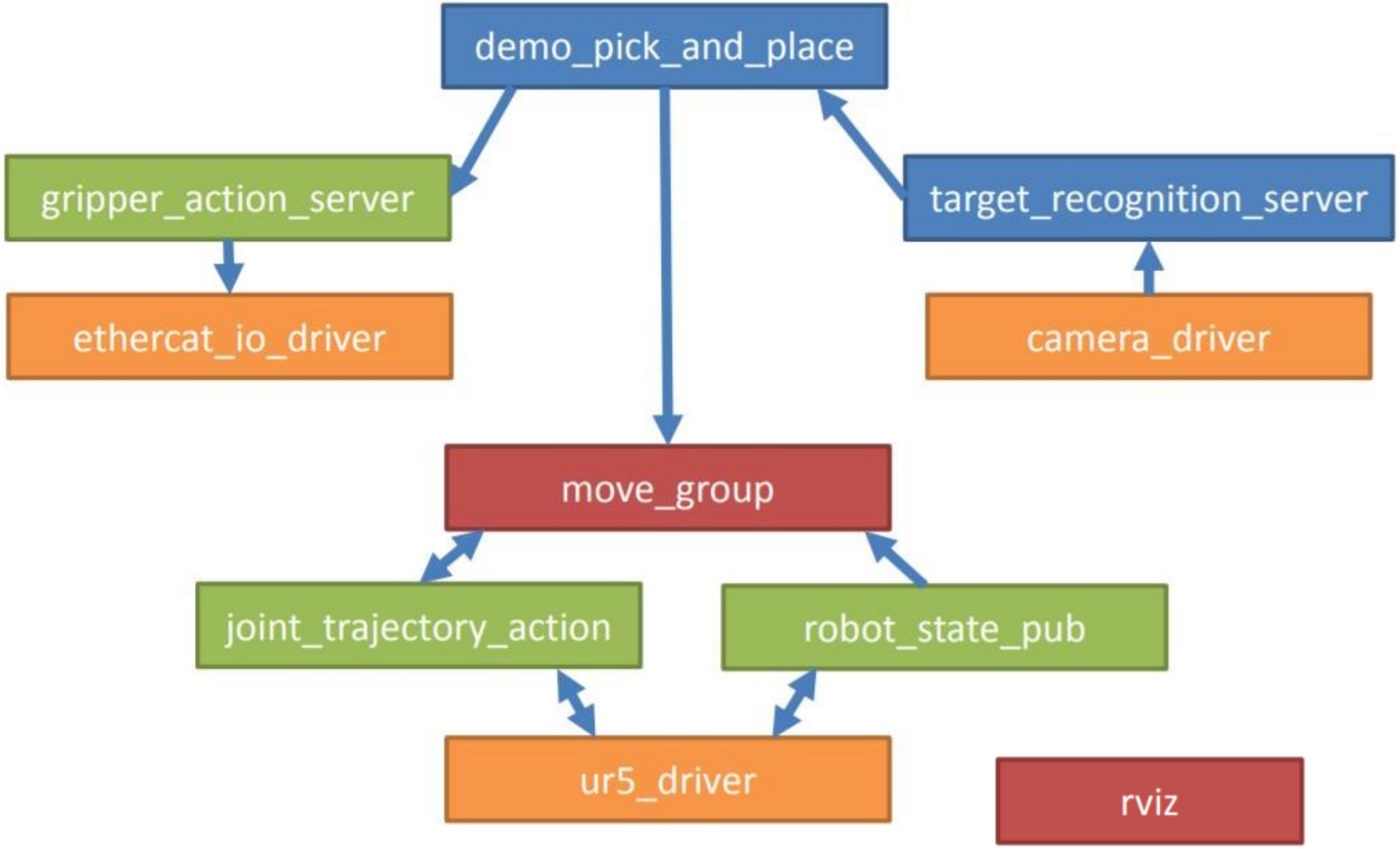
(Adapted from ROS Wiki: *wiki.ros.org/roscore*)

ROS Nodes: Useful Commands

- **rosrun package_name node_name**
  - Execute ROS node
- **rosnode**
  - **rosnode list**
    - View running nodes
  - **rosnode info node_name**
    - View node details *(publishers, subscribers, services, etc.)*
  - **rosnode kill node_name**
    - Kill running node
    - Good for remote machines
    - *Ctrl+C is usually easier*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Nodes: Example Graph



(Adapted from ROS ROS-I Basic Developers Training)

INESCTEC

SAFER Robotics Workshop

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. **ROS Topics & Messages**
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS Topics: Overview

- **Topics** are **named buses** over which **nodes exchange messages**.

- Topics have **anonymous publish/subscribe semantics**, which **decouples the production of information from its consumption**.
  - In general, **nodes are not aware of who they are communicating with**.
  - Instead, nodes that are interested in data **subscribe** to the relevant topic;
  - Nodes that generate data **publish** to the relevant topic.
  - There can be **multiple publishers and subscribers** to a topic.

- Topics are intended for **unidirectional, streaming communication**.
  - Nodes that need to perform remote procedure calls, *i.e. receive a response to a request, should use* **services** *instead*.
  - There is also the **Parameter Server** for maintaining small amounts of *(static)* state.

Services and **Parameters** will be addressed later on in this tutorial!

(Adapted from ROS Wiki: *wiki.ros.org/Topics*)

ROS Topics: Types

- **ROS Topics Types:**
  - Each topic is strongly typed by the **ROS message type** used to publish to it and nodes can only receive messages with a matching type.
  - The **Master does not enforce type consistency** among the publishers, but subscribers will not establish message transport unless the types match.
  - Furthermore, all ROS clients check to make sure that an MD5 computed from the msg files match.
    - This check ensures that the ROS Nodes were compiled from consistent code bases.

(Adapted from ROS Wiki: *wiki.ros.org/Topics*)

ROS Topics: Transports Systems

- **ROS Topics Transport Systems:**

This will be a **major** change in **ROS2**!

  - ROS currently supports **TCP/IP**-based and **UDP**-based message transport:
    - The TCP/IP-based transport is known as **TCPROS** and streams message data over **persistent TCP/IP connections**. **TCPROS** is the **default transport** used in ROS and is the **only transport that client libraries are required to support**.
    - The **UDP**-based transport, which is known as **UDPROS** and is currently only supported in roscpp, separates messages into **UDP packets**. **UDPROS** is a **low-latency**, **lossy transport**, so is best suited for tasks like teleoperation.

  - ROS nodes **negotiate** the desired transport at **runtime**.
    - *For example*: if a node prefers **UDPROS** transport but the other Node does not support it, it can fallback on **TCPROS** transport.
    - This negotiation model enables new transports to be added over time as compelling use cases arise.

(Adapted from ROS Wiki: *wiki.ros.org/Topics*)

ROS Topics: Details

- 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*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Messages: Overview

- Nodes communicate with each other by **publishing messages** to **topics**.

- A **message** is a **simple data structure**, comprising **typed fields**.
  - Standard **primitive types** *(integer, floating point, boolean, etc.)* are supported, as are arrays of primitive types.
  - Messages can include **arbitrarily nested structures and arrays** *(much like C structs)*.

- Nodes can also exchange a request and response message as part of a **ROS service call**. These request and response messages are defined in **`srv`** files.

> **Services** will be addressed later on in this tutorial!

(Adapted from ROS Wiki: *wiki.ros.org/Messages*)

ROS Messages: Types

- **Similar to C structures**

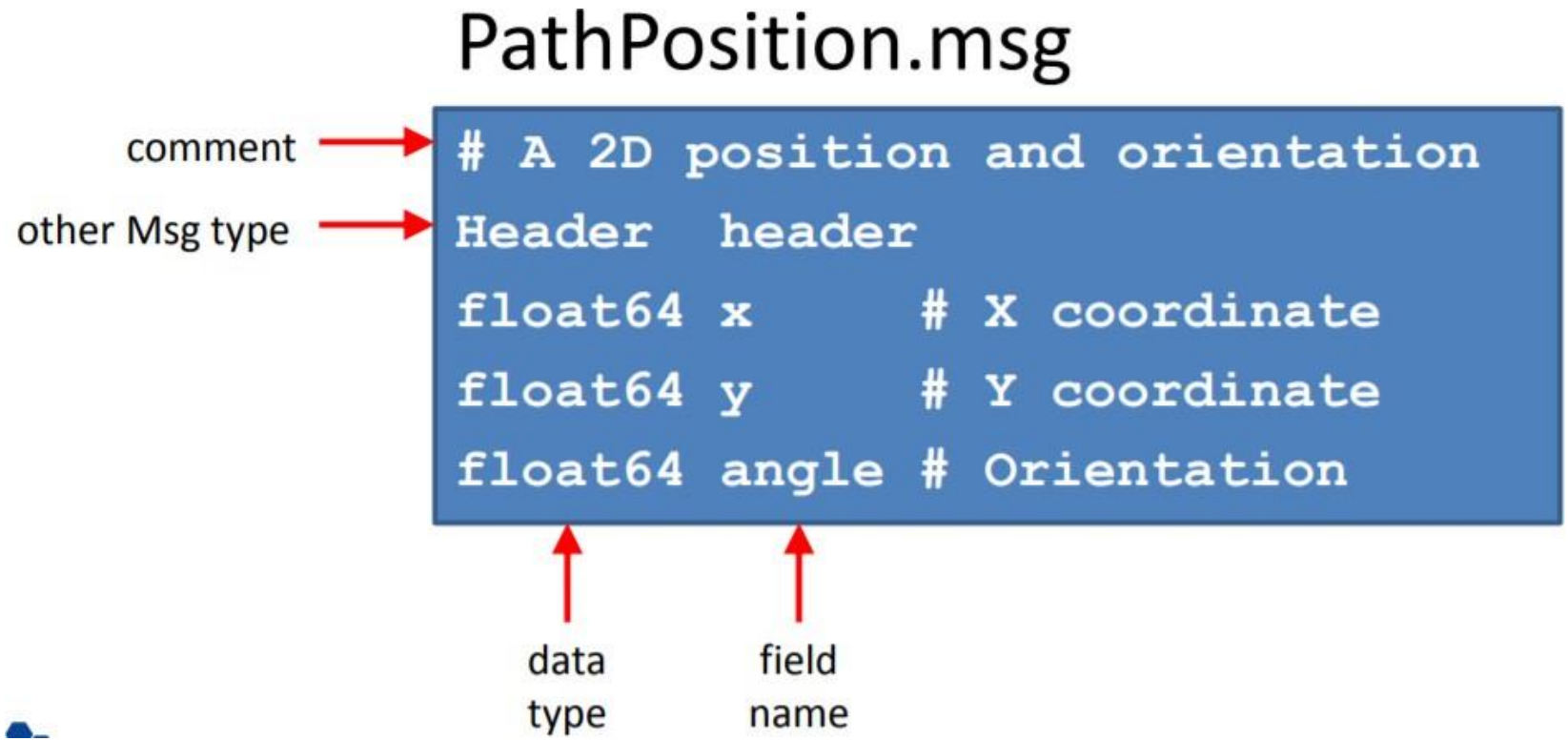
- **Standard data primitives**:
  - **Boolean**: bool
  - **Integer**: int8,int16,int32,int64
  - **Unsigned Integer**: uint8,uint16,uint32,uint64
  - **Floating Point**: float32, float64
  - **String**: string
  - **Fixed length arrays**: bool[16]
  - **Variable length arrays**: int32[]

- Other:
  - Nest message types for more complex data structure

(Adapted from ROS ROS-I Basic Developers Training)

ROS Messages: Message Description File

- All Messages are defined by a **.msg** file

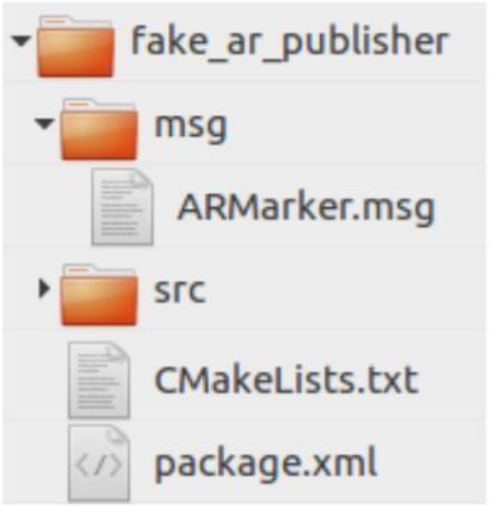


(Adapted from ROS ROS-I Basic Developers Training)

ROS Messages: Custom ROS Messages

- **Custom message types** are defined in **msg** subfolder of packages

- **Modify** CMakeLists.txt & package.xml to enable **message generation**.



CMakeList.txt Lines needed to **generate custom message types:**

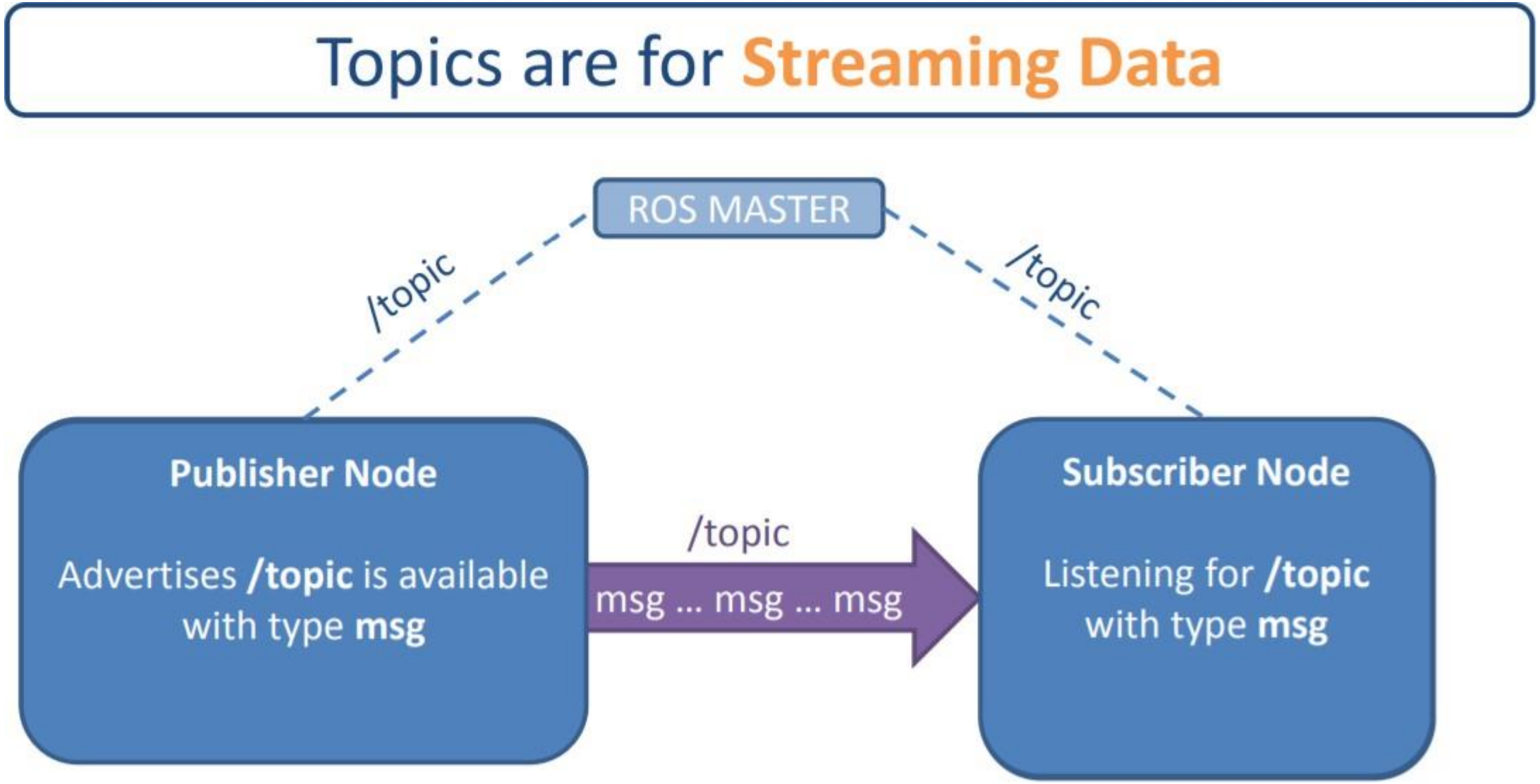
```
find_package(catkin REQUIRED COMPONENTS message_generation)
add_message_files(custom.msg . . . )
generate_messages(DEPENDENCIES . . . )
catkin_package(CATKIN_DEPENDS roscpp  message_runtime)
```

package.xml lines needed to **generate custom message types**:

```
<build_depend>message_generation</build_depend>
<build_export_depend>message_runtime</build_export_depend>
<run_depend>message_runtime</run_depend>
```

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages



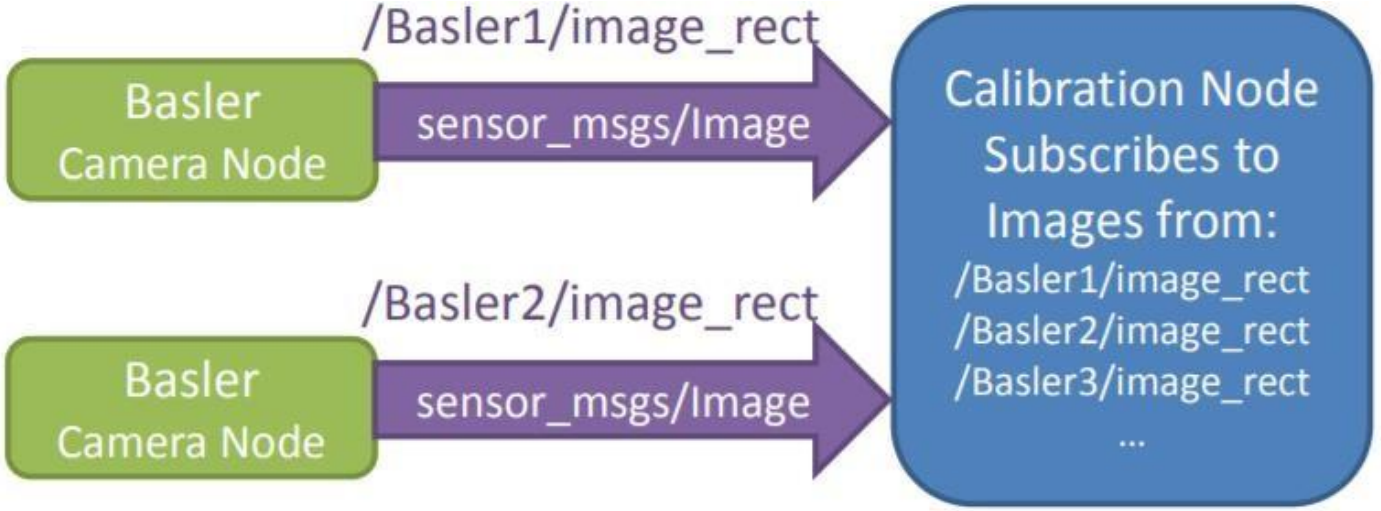
(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: Topics vs. Messages

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

(Adapted from ROS ROS-I Basic Developers Training)

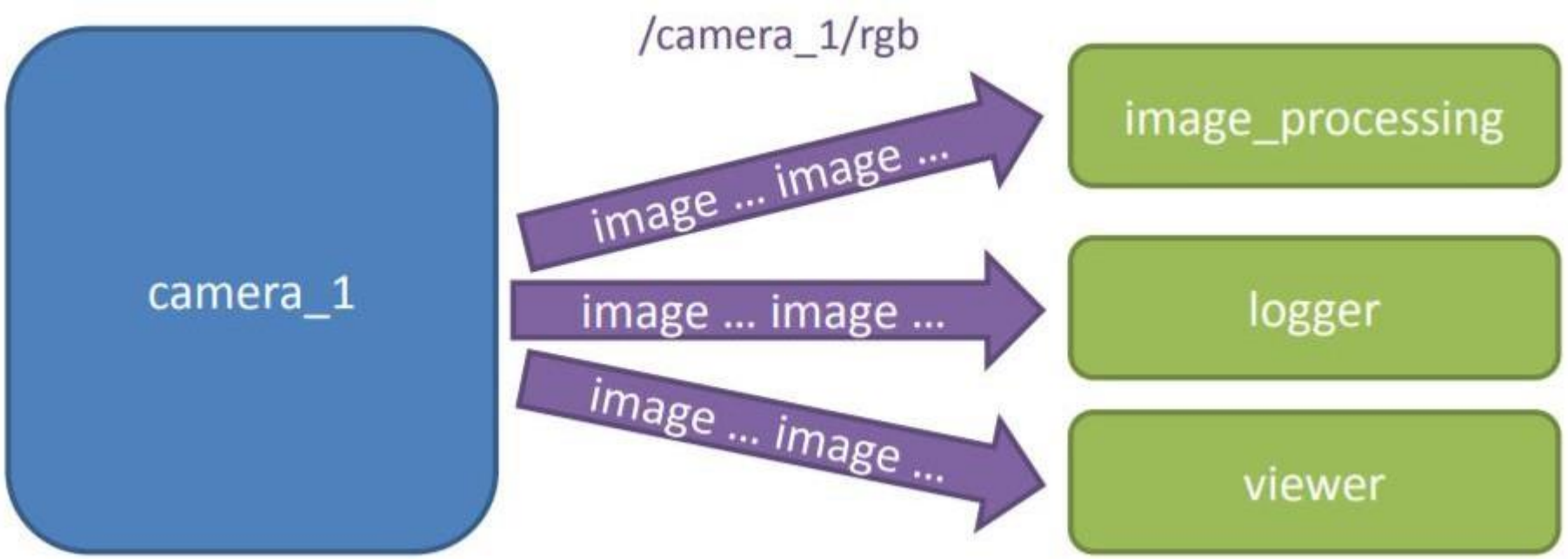
ROS Topics / Messages: Practical Example



(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: Multiple Publishers / Subscribers

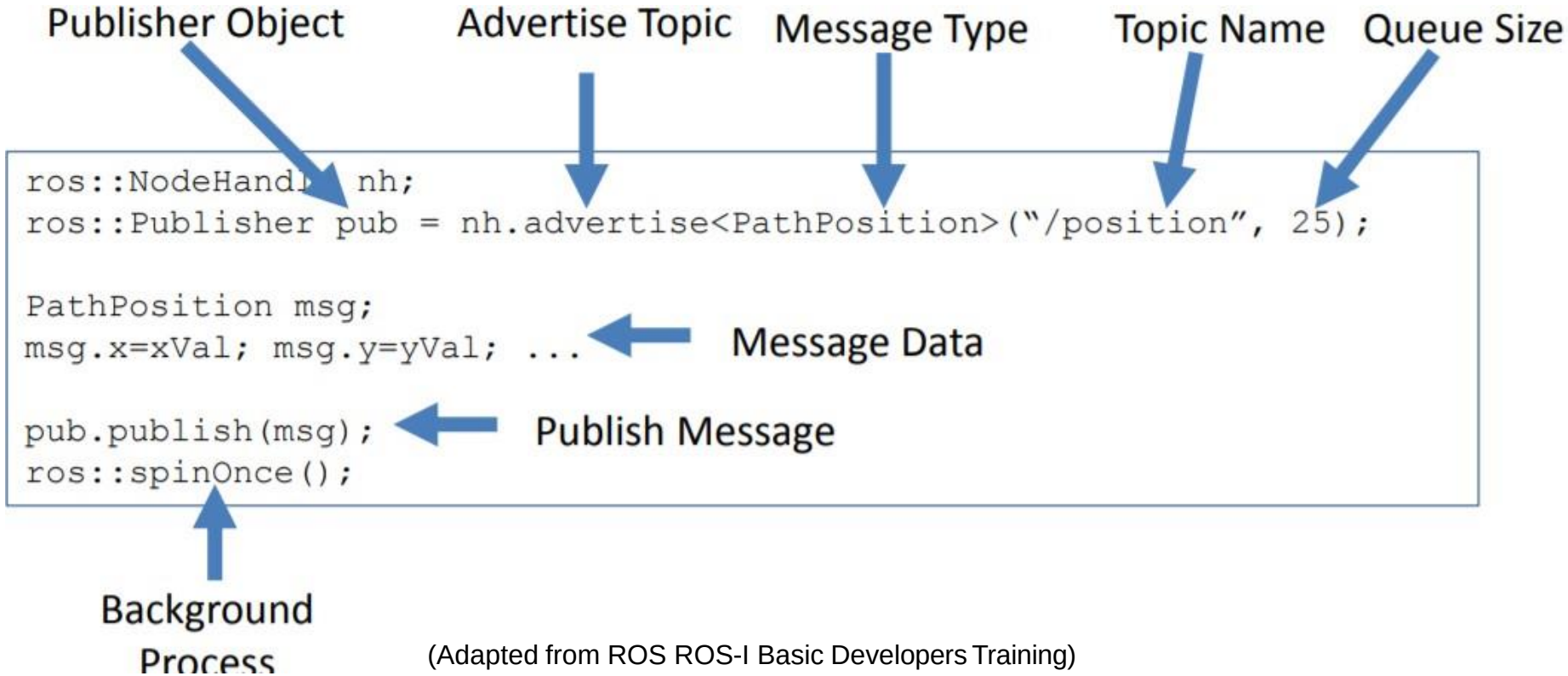
- Many nodes can **Publish** or **Subscribe** to the same Topic
  - Communications are direct **node-to-node**



(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: ROS Topics Syntax

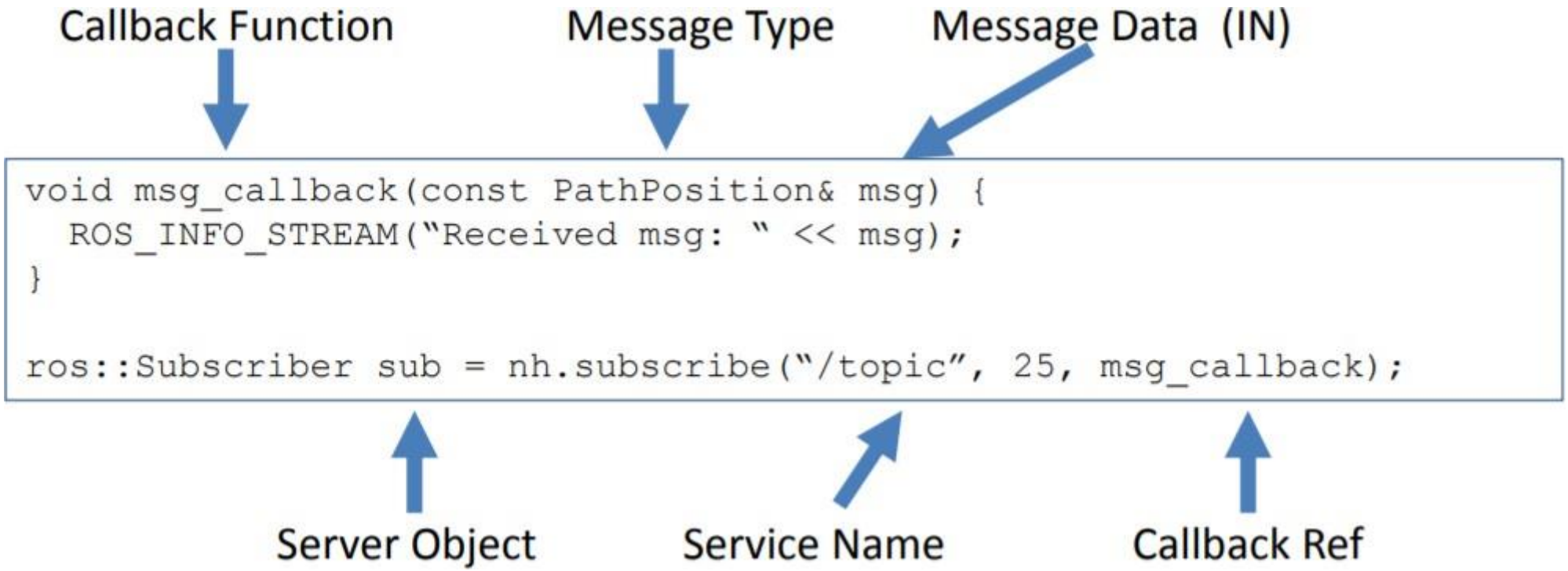
- **ROS Topic Publisher**:
  - Advertises available topic *(Name, Data Type)*
  - Populates message data
  - Periodically publishes new data



(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: ROS Topics Syntax

- **ROS Topic Subscriber**:
  - Defines callback function
  - Listens for available topic *(Name, Data Type)*



(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: ROS Messages Commands

- **rosmsg list**
  - Show all ROS topics currently installed on the system
- **rosmsg package <package>**
  - Show all ROS message types in package <package>
- **rosmsg show <package>/<message_type>**
  - Show the structure of the given message type

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: ROS Topics Commands

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

(Adapted from ROS ROS-I Basic Developers Training)

INESCTEC

SAFER Robotics Workshop

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. **ROS Services**
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS Services: Overview

- The **publish / subscribe model** is a **very flexible communication** paradigm, but its **many-to-many one-way transport is not appropriate for RPC request / reply interactions**, which are often required in a distributed system.

- **Request / reply** is done via a **ROS Service**, which is defined by a **pair of messages**:
  - One message for the **request** and one message for the **reply**.

- A providing ROS node offers a **service** under a **string name**, and a **client calls the service** by sending the **request message** and **awaiting the reply**.

- **Services** are defined using **`.srv`** files, which are compiled into source code by a ROS client library.

(Adapted from ROS Wiki: *wiki.ros.org/Services*)

ROS Services: Types

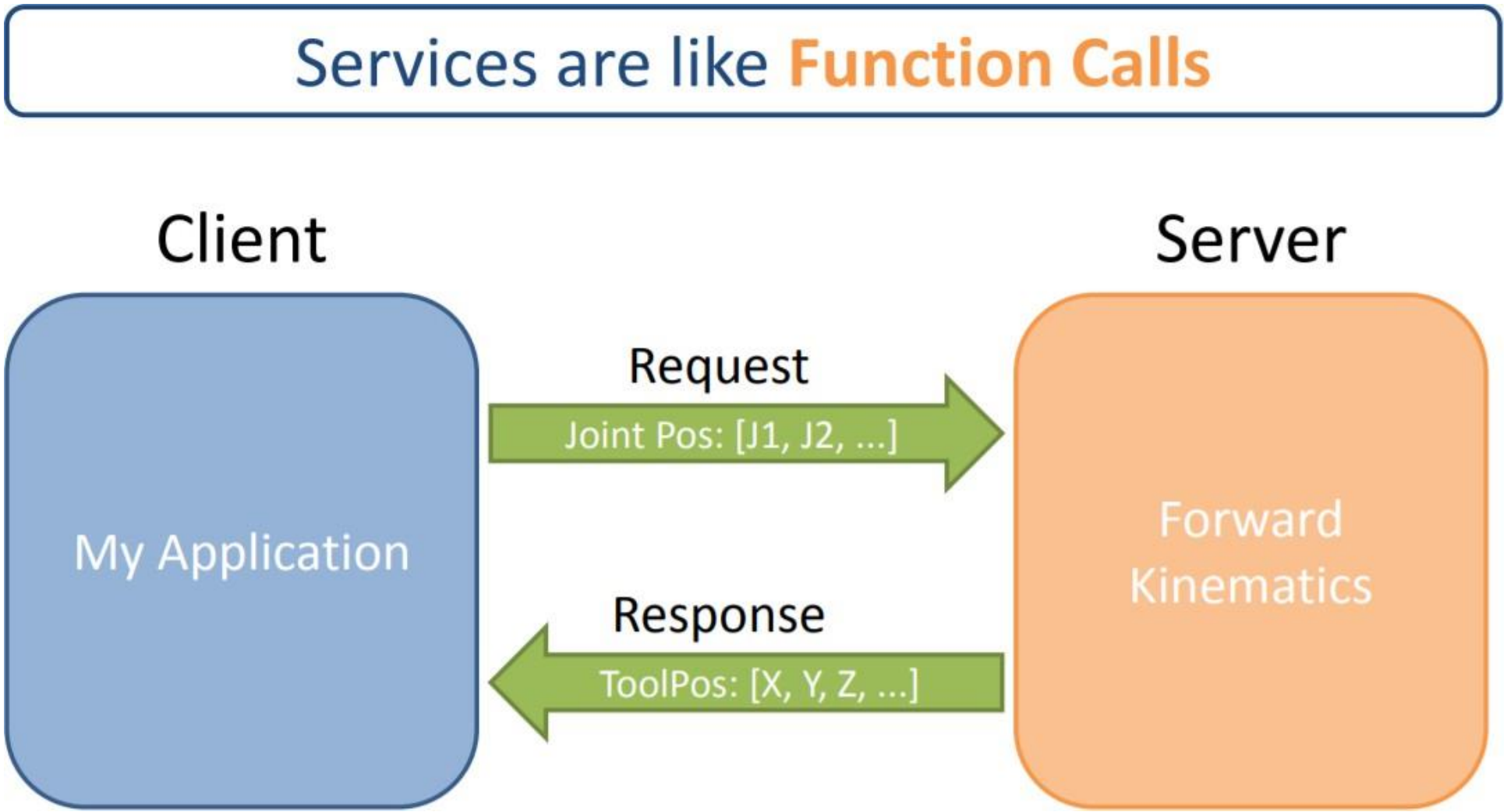
- Like topics, **services** have an associated service type that is the package resource name of the **`.srv`** file.

- As with other ROS filesystem-based types, the **service type** is the **package name + the name of the .srv file**.
  - For example: my_srvs/srv/PolledImage.srv has the service type my_srvs/PolledImage.

- In addition to the service type, **services are versioned by an MD5 sum** of the **`.srv`** file.
  - Nodes can only make service calls if both the service type and MD5 sum match.
  - This ensures that the client and server code were built from a consistent codebase.

(Adapted from ROS Wiki: *wiki.ros.org/Services*)

ROS Services



(Adapted from ROS ROS-I Basic Developers Training)

ROS Services: Details

- Each **Service** is made up of 2 components:
  - **Request** : sent by **client**, received by **server**
  - **Response** : generated by **server**, sent to **client**

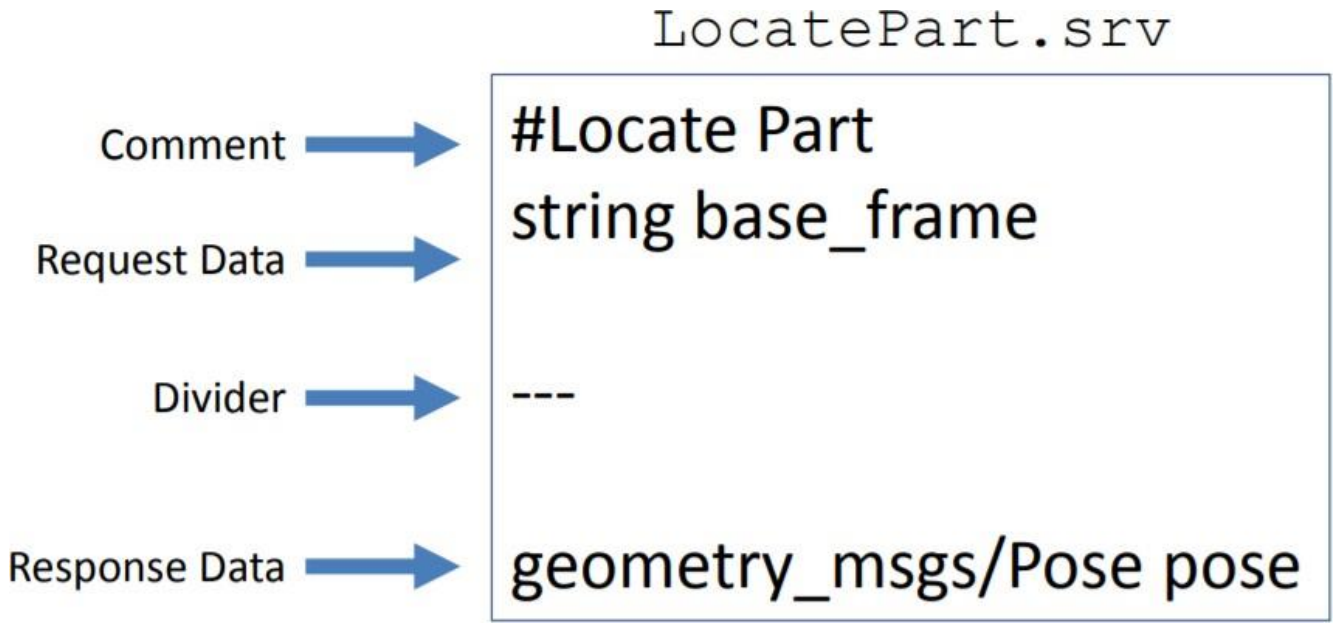
- Call to service **blocks** in client
  - Code will wait for service call to complete
  - Separate connection for each service call

- Typical Uses:
  - Algorithms: *kinematics, perception*
  - Closed-Loop Commands: *move-to-position, open gripper*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Services: Syntax

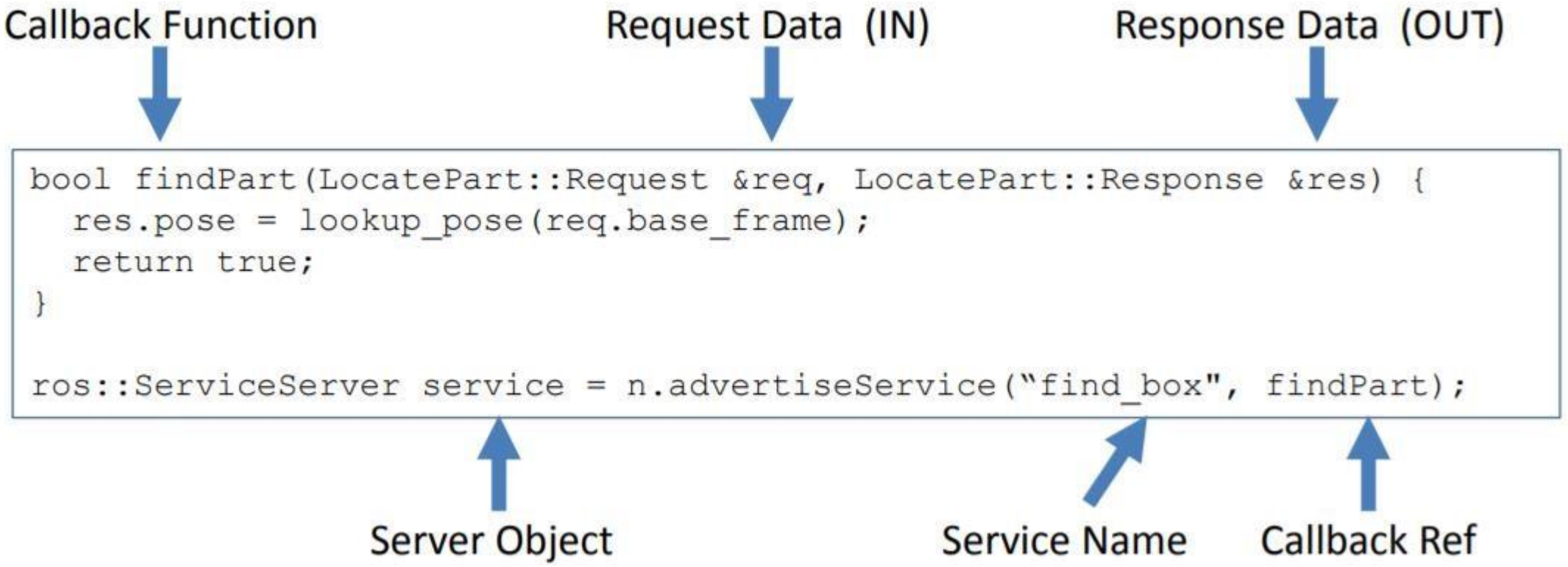
- **Service Definition**:
  - Defines **Request** and **Response** data types
    - Either/both data type(s) may be **empty**. Always receive "completed" handshake.
  - Auto-generates C++ Class files *(.h/.cpp)*, Python, etc.



(Adapted from ROS ROS-I Basic Developers Training)

ROS Services: Syntax

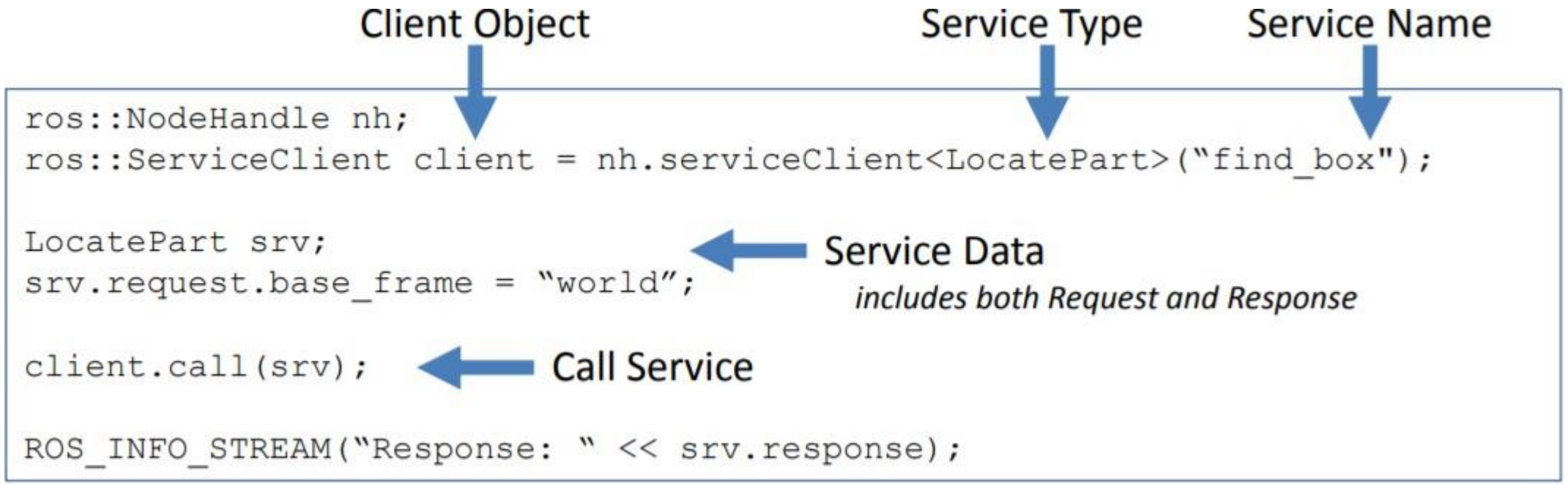
- **Service Server**:
  - Defines associated **Callback Function**
  - Advertises available service *(Name, Data Type)*



(Adapted from ROS ROS-I Basic Developers Training)

ROS Services: Syntax

- **Service Client**:
  - Connects to specific Service *(Name / Data Type)*
  - Fills in Request data
  - Calls Service



(Adapted from ROS ROS-I Basic Developers Training)

INESCTEC

SAFER Robotics Workshop

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. **ROS Actions**
9. ROS Parameters
10. ROS Launch Files

ROS Actions: Overview

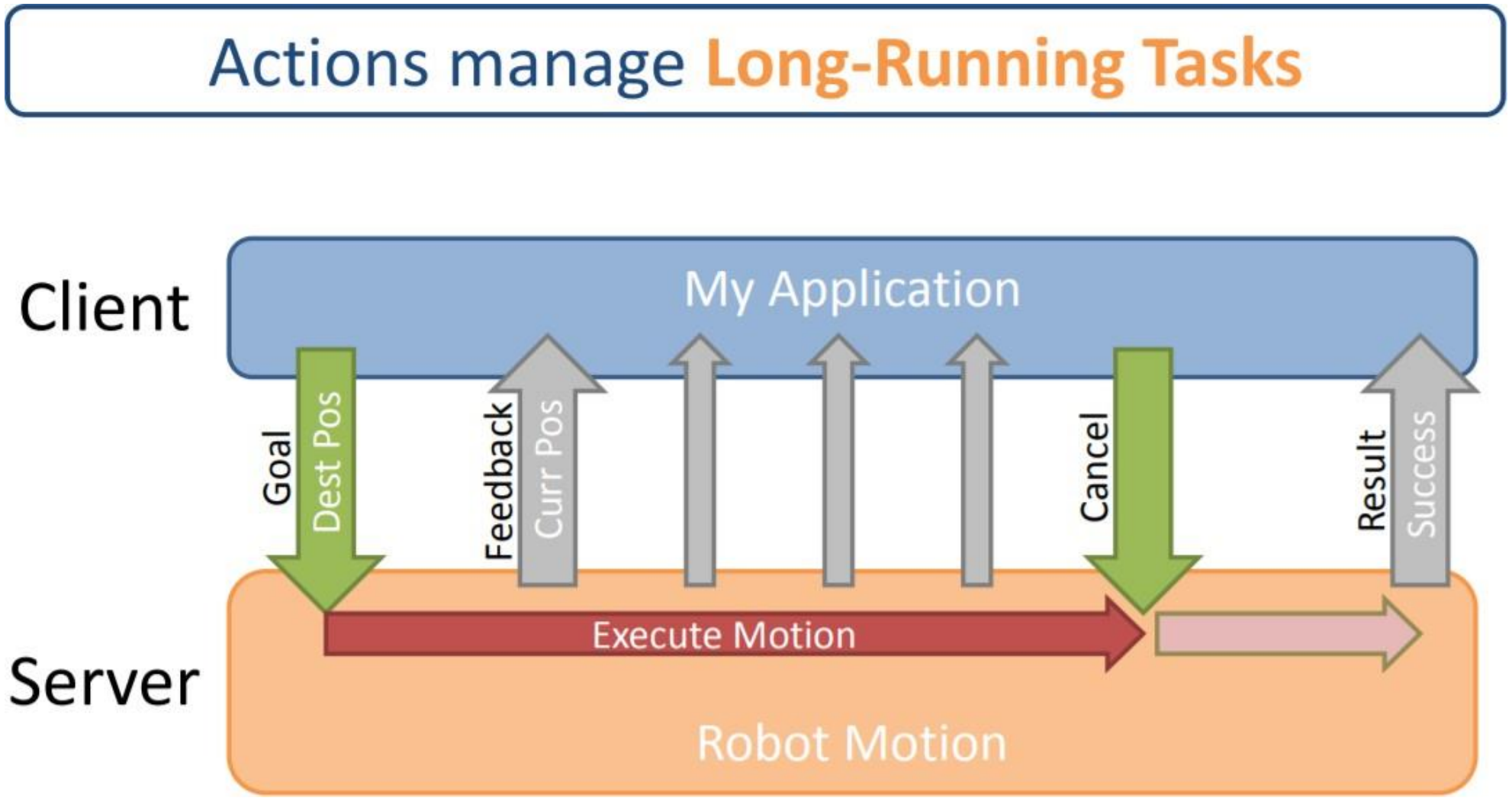
- In any large ROS based system, there are cases when someone would like to **send a request** to a node to perform some task, and also **receive a reply to the request**. This can currently be achieved via **ROS services**.

- In some cases, however, **if the service takes a long time to execute**, the user might want the ability to **cancel the request during execution** or get **periodic feedback** about how the request is progressing.

- ROS Actions can create servers that execute **long-running goals** that can be **preempted**. It also provides a **client interface** in order to **send requests** to the server.

(Adapted from ROS Wiki: *wiki.ros.org/actionlib*)

ROS Actions: Overview



(Adapted from ROS ROS-I Basic Developers Training)

ROS Actions: Details

- Each **ROS Action** is made up of 3 components:
  - **Goal**, sent by client, received by server
  - **Result**, generated by server, sent to client
  - **Feedback**, generated by server

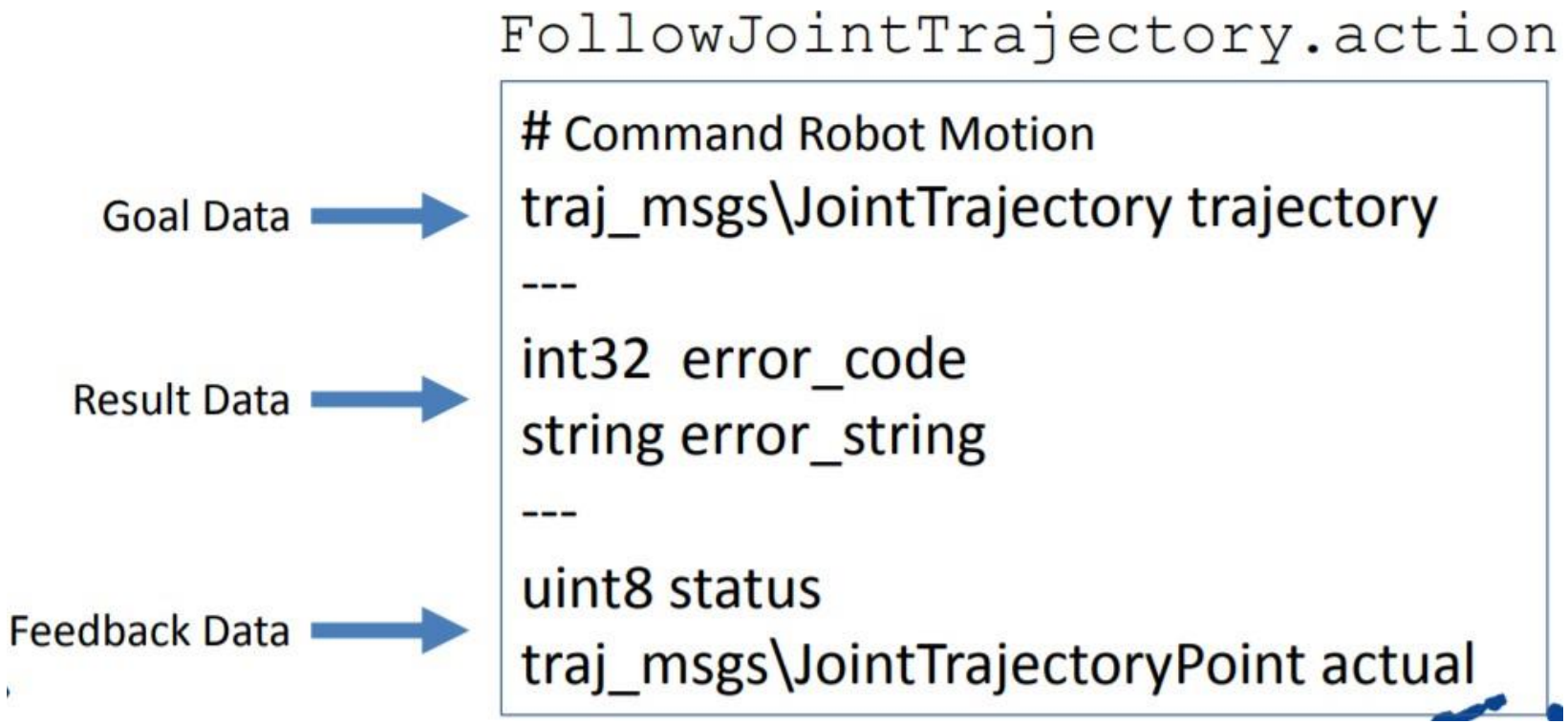
- **Non-blocking in client**
  - Can monitor feedback or cancel before completion

- Typical Uses:
  - **“Long” Tasks**: *Robot Motion, Path Planning*
  - **Complex Sequences**: *Pick Up Box, Sort Widgets*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Actions: Syntax

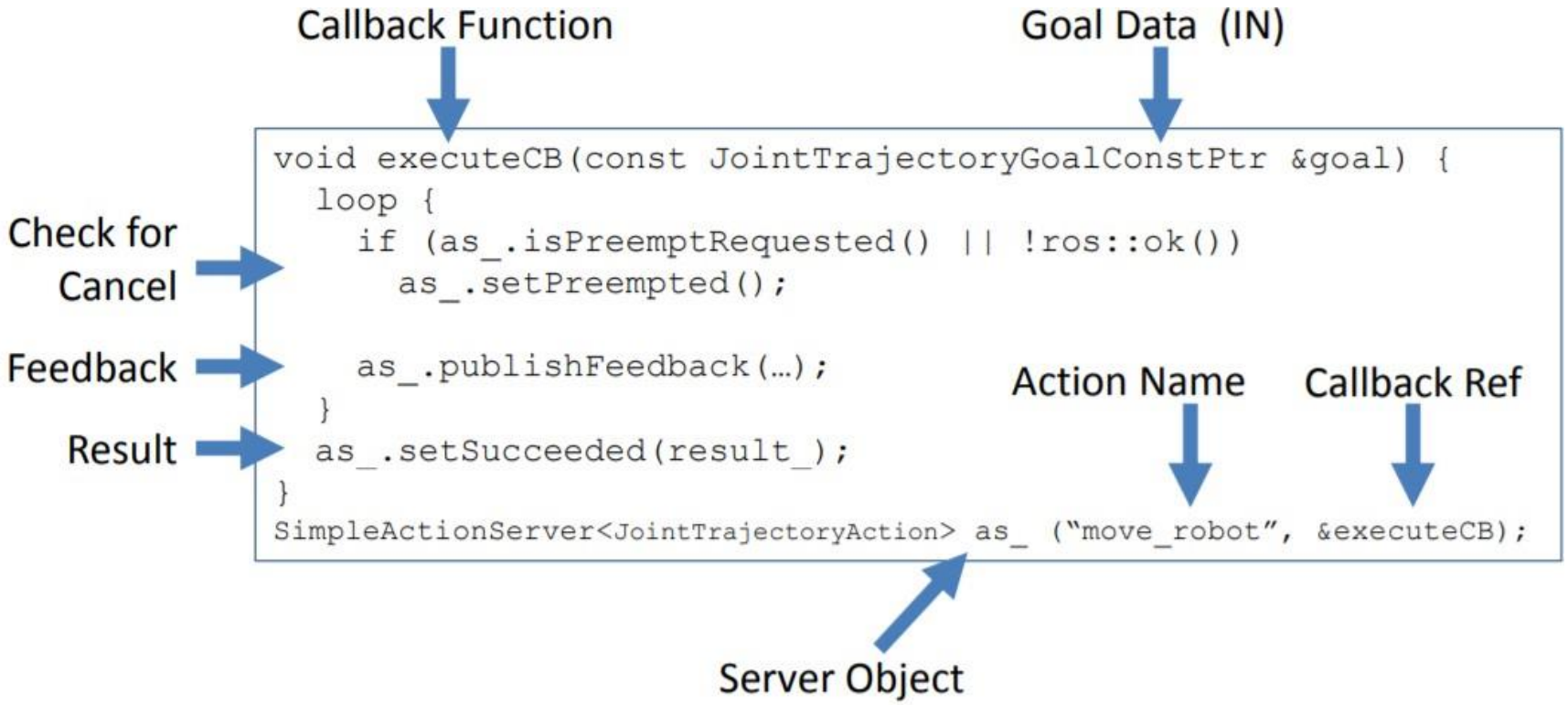
- **Action Definition:**
  - Defines **Goal**, **Feedback** and **Result** data types
    - Any data type(s) may be **empty**. Always receive handshakes.
  - Auto-generates C++ Class files *(.h/.cpp)*, Python, etc.



(Adapted from ROS ROS-I Basic Developers Training)

ROS Actions: Syntax

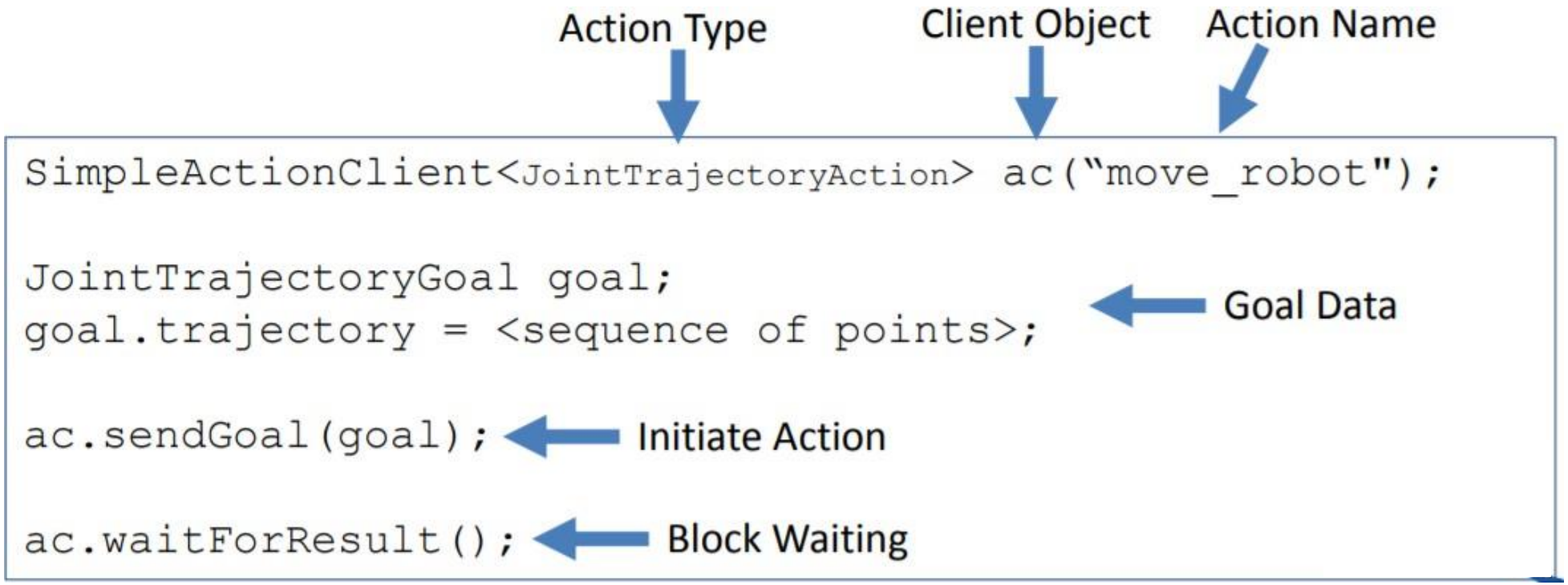
- **Action Server:**
  - Defines **Execute Callback**
  - Periodically **Publish Feedback**
  - Advertises available action *(Name, Data Type).*



(Adapted from ROS ROS-I Basic Developers Training)

ROS Actions: Syntax

- **Action Client:**
  - Connects to specific **Action** *(Name / Data Type)*
  - Fills in **Goal** data
  - Initiate Action / Waits for Result



(Adapted from ROS ROS-I Basic Developers Training)

ROS Messages vs. ROS Services vs. ROS Actions

Type	Strengths	Weaknesses
Message	•Good for most sensors (streaming data) •One - to - Many	•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	•Monitor long-running processes •Handshaking (knowledge of missed connection)	•Complicated

(Adapted from ROS ROS-I Basic Developers Training)

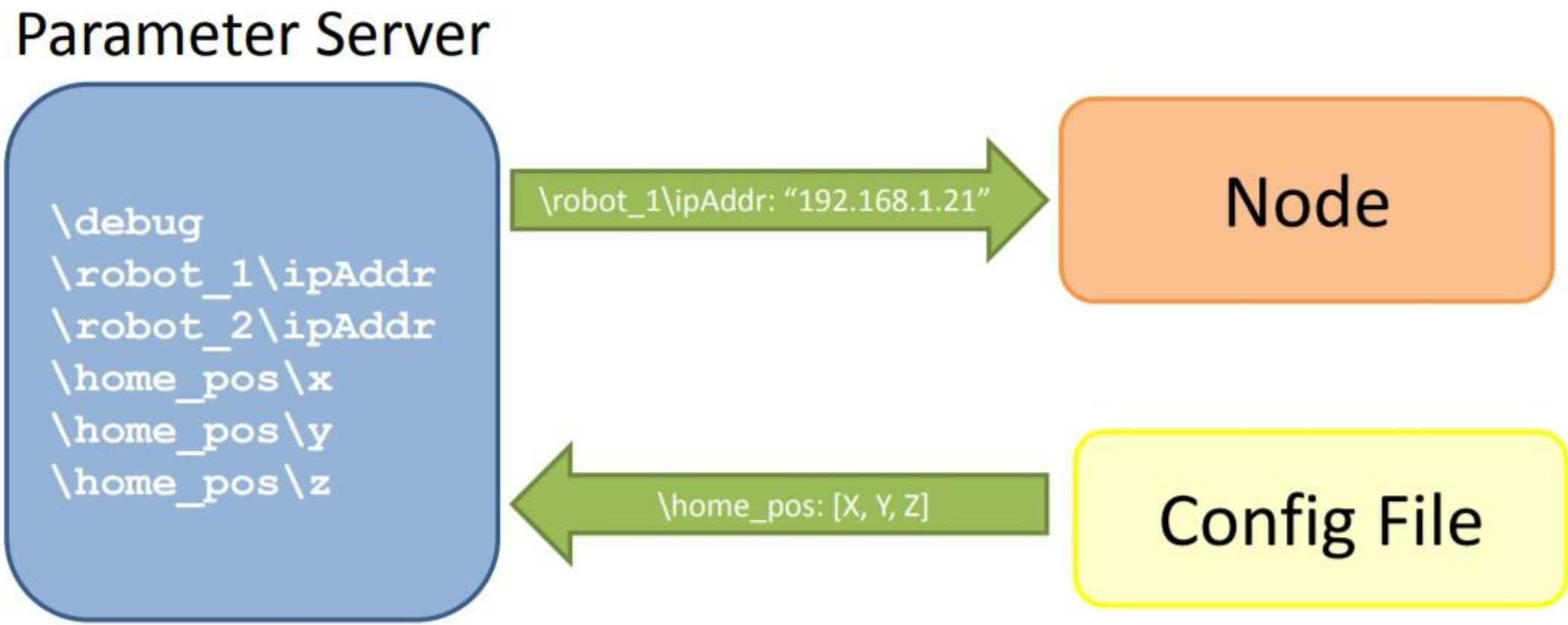
INESCTEC

SAFER Robotics Workshop

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. **ROS Parameters**
10. ROS Launch Files

ROS Parameters: Overview

Parameters are like **Global Data**

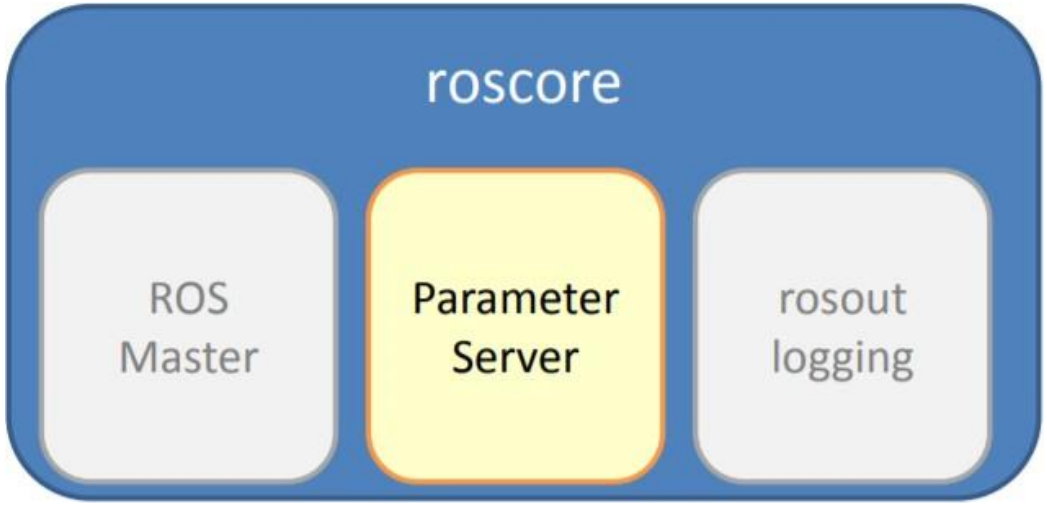


(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Overview

- Typically **configuration-type** values:
  - *Robot kinematics*
  - *Workcell description*
  - *Algorithm limits / tuning*

- Accessed through **the Parameter Server.**
  - Typically handled by **roscore**



(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Ways to Setup

- Can set from:
  1. YAML Files
  2. Command Line
  3. Programs

1.
```
manipulator_kinematics:
  solver: kdl_plugin/KDLKinematics
  search_resolution: 0.005
  timeout: 0.005
  attempts: 3
```

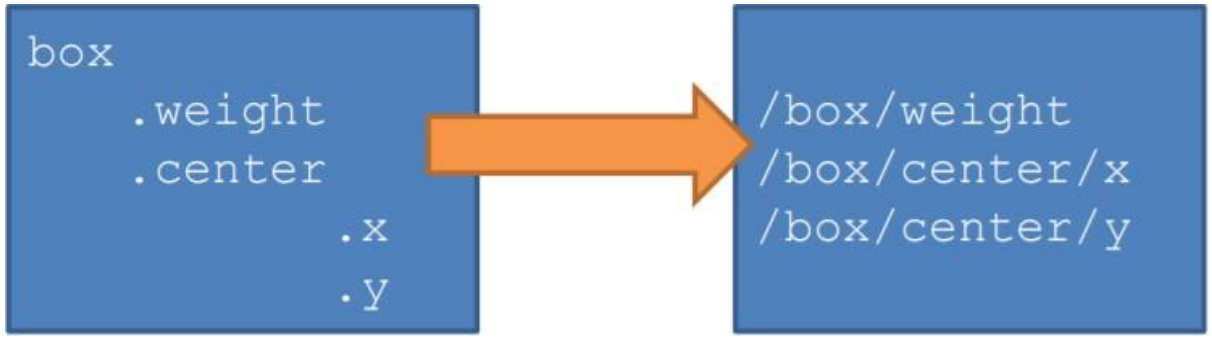
2.
```
rosrun my_pkg load_robot _ip:="192.168.1.21"
rosparam set "/debug" true
```

3.
```
nh.setParam("name", "left");
```

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Datatypes

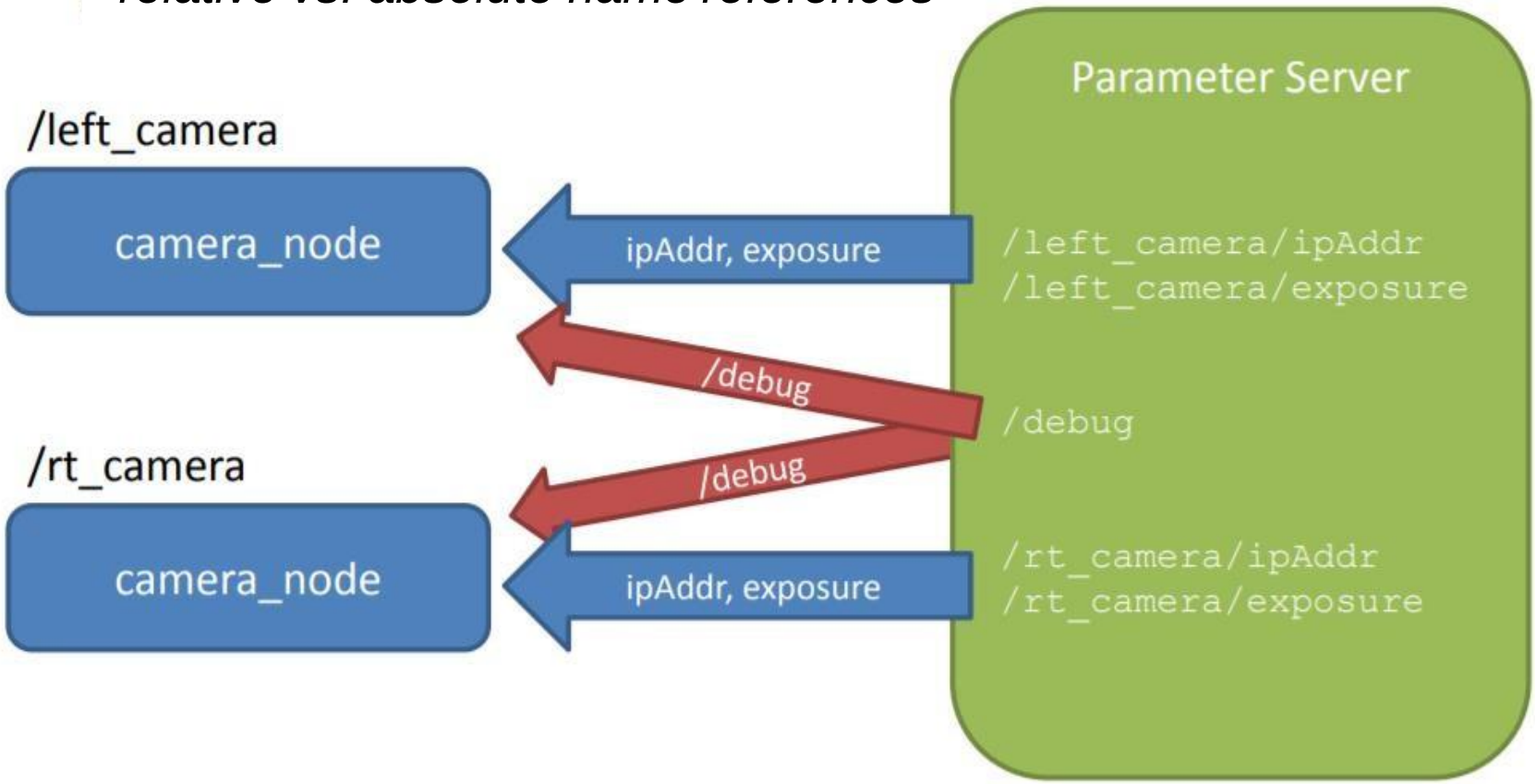
- **Native Types**
  - *int, real, boolean, string*
- **Lists (vectors)**
  - *can be mixed type: [1, str, 3.14159]*
  - *but typically of single type: [1.1, 1.2, 1.3]*
- **Dictionaries (structures)**
  - *translated to “folder” hierarchy on server*



(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Namespaces

- Folder Hierarchy allows Separation:
  - *Separate nodes can co-exist, in different “namespaces”*
  - *relative vs. absolute name references*



(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Namespaces

- **rosparam**
  - rosparam `set` <key> <value>
    - Set parameters
  - rosparam `get` <key>
    - Get parameters
  - rosparam `delete` <key>
    - Delete parameters
  - rosparam `list`
    - List all parameters currently set
  - rosparam `load` <filename> [<namespace>]
    - Load parameters from file

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Namespaces

- **rosparam**
  - rosparam `set <key> <value>`
    - Set parameters
  - rosparam `get <key>`
    - Get parameters
  - rosparam `delete <key>`
    - Delete parameters
  - rosparam `list`
    - List all parameters currently set
  - rosparam `load <filename> [<namespace>]`
    - Load parameters from file

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: C++ API

- Accessed through **ros::NodeHandle object**
  - Also sets default **Namespace** for access
    - Relative Namespaces:

```
ros::NodeHandle relative;
relative.getParam("test");
```

"/<ns>/test"

    - Fixed Namespaces:

```
ros::NodeHandle fixed("/myApp");
fixed.getParam("test");
```

"/myApp/test"

    - Private Namespaces:

```
ros::NodeHandle priv("~");
priv.getParam("test");
```

"/myNode/test"

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: C++ API

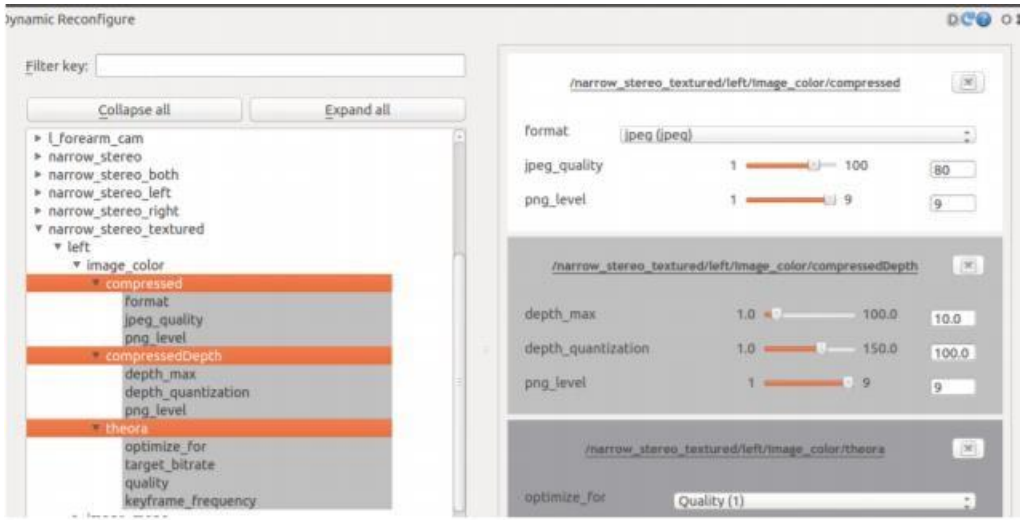
- NodeHandle object methods
  - **nh.hasParam(key)**
    - Returns true if parameter exists
  - **nh.getParam(key, &value)**
    - Gets value, returns T/F if exists.
  - **nh.param(key, &value, default)**
    - Get value (or default, if doesn't exist)
  - **nh.setParam(key, value)**
    - Sets value
  - **nh.deleteParam(key)**
    - Deletes parameter

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: C++ API

- Parameters must be read explicitly by nodes
  - *no on-the-fly updating*
  - *typically read only when node first started*

- ROS package **dynamic_reconfigure** can help
  - *nodes can register callbacks to trigger on change*
  - *outside the scope of this class, but useful*



(Adapted from ROS ROS-I Basic Developers Training)

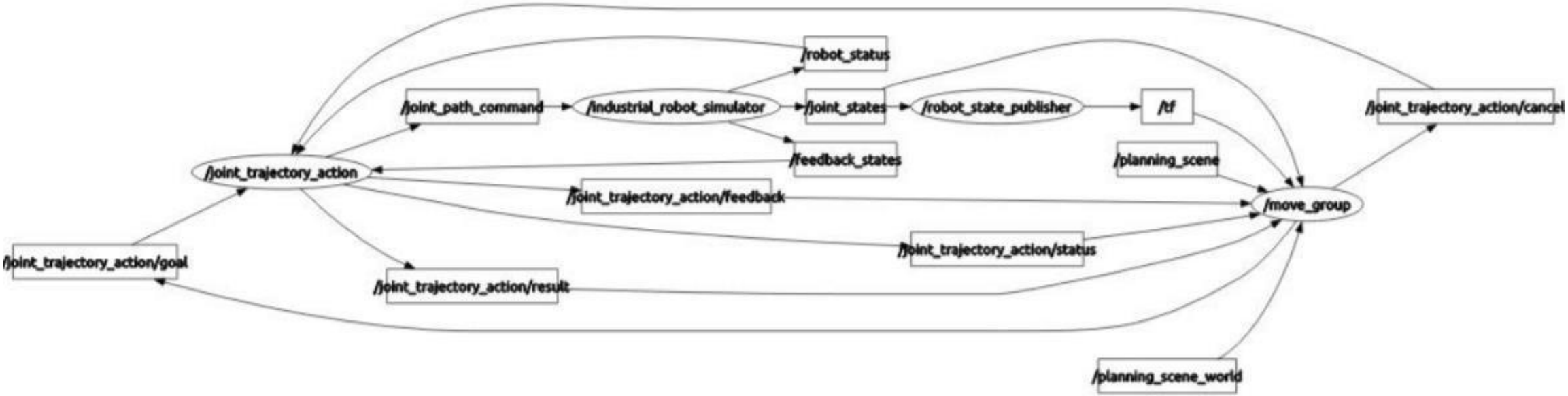
INESCTEC

SAFER Robotics Workshop

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. **ROS Launch Files**

ROS Launch Files: Motivation

- ROS is a **Distributed System**:
  - Often **dozens** (if not hundreds) of **nodes**, plus **configuration data**
  - It would be *(VERY!)* painful to start each node “manually”



(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: Overview

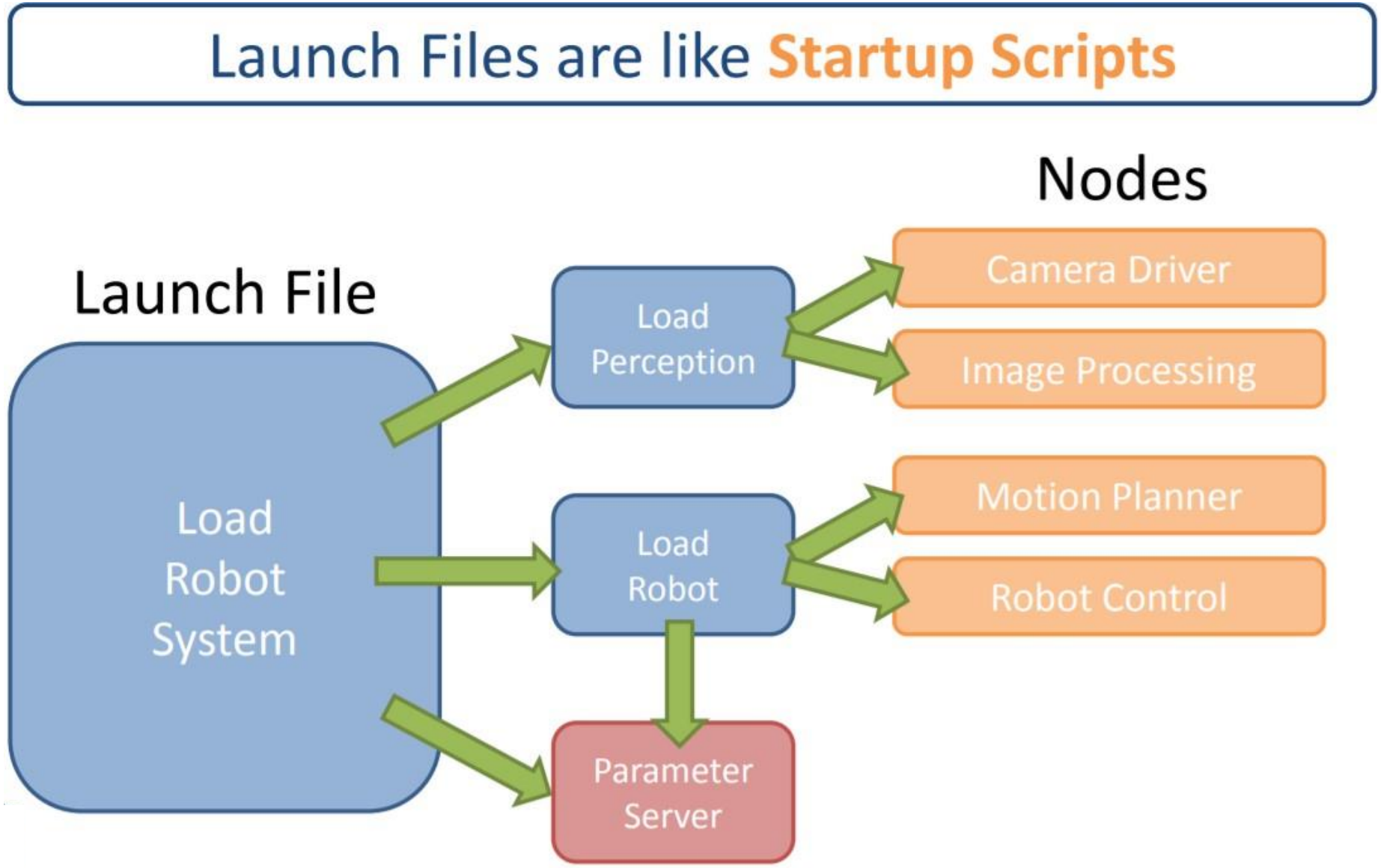
- **ROS Launch** is a **tool** for **easily launching multiple ROS nodes locally and remotely** via SSH, as well as **setting parameters on the Parameter Server**.

- It includes options to **automatically respawn processes** that have already died.

- **roslaunch** takes in one or more **XML configuration files** (with the `.launch` extension) that specify the **parameters** to set and **nodes** to launch, as well as the machines that they should be run on.

(Adapted from ROS Wiki: *wiki.ros.org/roslaunch*)

ROS Launch Files: Overview



(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: Overview

- **Launch files automate** system startup
- **XML** formatted script for running nodes and setting parameters
- Ability to **pull information from other packages**
- Will automatically start/stop **roscore**

(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: Notes

- Can launch ***other*** launch files
- **Executed in order**, without pause or wait
  - *Exception: Parameters set to parameter server before nodes are launched*
- Can accept **arguments**
- Can perform **simple IF-THEN** operations
- Supported **parameter types**:
  - *Bool, string, int, double, text file, binary file*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: Basic Syntax

- **<launch>** – Required outer tag
- **<rosparam>** or **<param>** – Set parameter values
  - Including load from file (YAML)
- **<node>** – Start running a new node
- **<include>** – Import another launch file

```
<launch>
  <rosparam param="/robot/ip_addr">192.168.1.50</rosparam>

  <param name="robot_description" textfile="$(find robot_pkg)/urdf/robot.urdf"/>

  <node name="camera_1" pkg="camera_aravis" type="camnode" />

  <node name="camera_2" pkg="camera_aravis" type="camnode" />

  <include file="$(find robot_pkg)/launch/start_robot.launch" />
</launch>
```

(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: More Syntax

- **<arg>** – Pass a value into a launch file
- **if=** or **unless=** – Conditional branching
  - Extremely limited. True/False only (no comparisons).
- **<group>** – group commands, for if/unless or namespace
- **<remap>** – rename topics/services/etc.

```
<launch>
  <arg name="robot" default="sia20" />
  <arg name="show_rviz" default="true" />
  <group ns="robot" >
    <include file="$(find lesson)/launch/load_$(arg robot)_data.launch" />
     <remap from="joint_trajectory_action" to="command" />
  </group>
  <node name="rviz" pkg="rviz" type="rviz" if="$(arg show_rviz)" />
</launch>
```


(Adapted from ROS ROS-I Basic Developers Training)

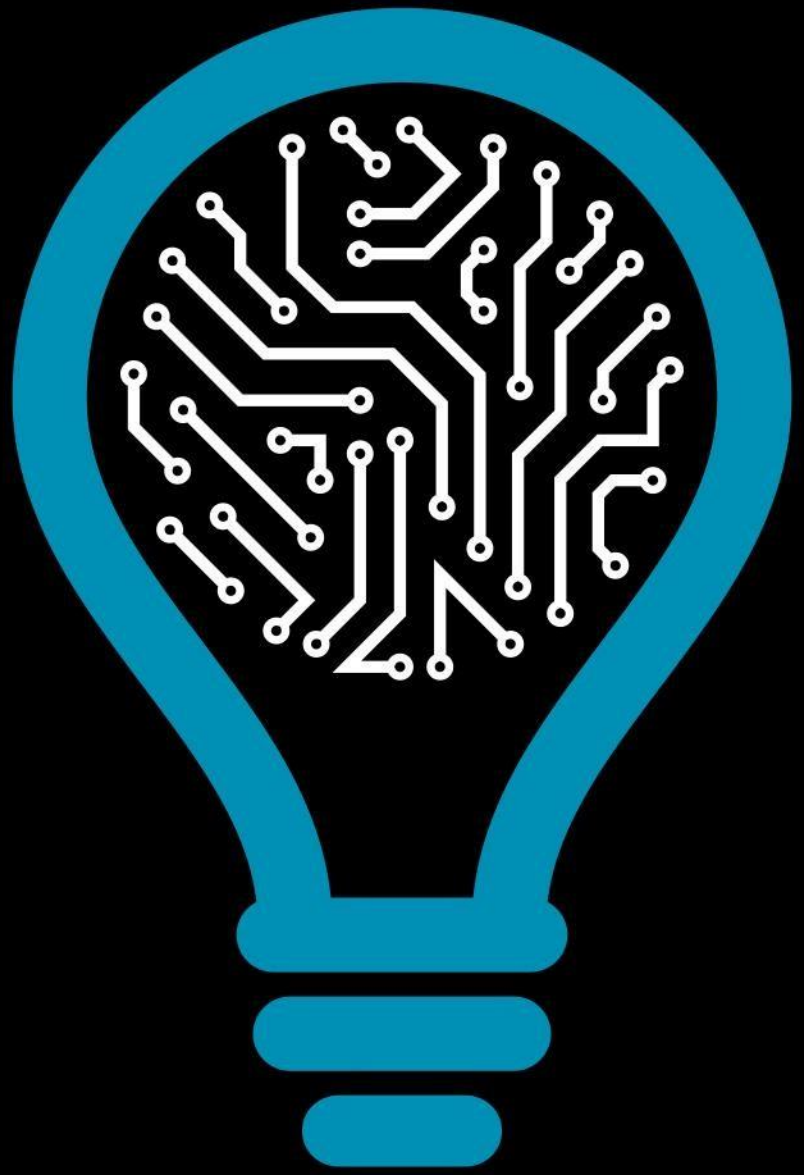
ROS Launch Files: Example

- motoman_sia20d_moveit_cfg
  - moveit_planning_exec.launch

```
<launch>
  <rosparam command="load" file="$(find motoman_support)/config/joint_names.yaml"/>
  <arg name="sim" default="true" />
  <arg name="robot_ip" unless="$(arg sim)" />
  <arg name="controller" unless="$(arg sim)" />
  <include file="$(find motoman_sia20d_moveit_config)/launch/planning_context.launch" >
    <arg name="load_robot_description" value="true" />
  </include>
  <group if="$(arg sim)">
    <include file="$(find industrial_robot_simulator)/launch/robot_interface_simulator.launch" />
  </group>
  <group unless="$(arg sim)">
    <include file="$(find motoman_sia20d_support)/launch/robot_interface_streaming_sia20d.launch" >
      <arg name="robot_ip" value="$(arg robot_ip)"/>
      <arg name="controller" value="$(arg controller)"/>
    </include>
  </group>
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />
  <include file="$(find motoman_sia20d_moveit_config)/launch/move_group.launch">
    <arg name="publish_monitored_planning_scene" value="true" />
  </include>
```

(Adapted from ROS ROS-I Basic Developers Training)

from knowledge production to science-based innovation



INESCTEC

INSTITUTE FOR SYSTEMS AND COMPUTER ENGINEERING, TECHNOLOGY AND SCIENCE