ROS-Industrial Basic Developer's Training Class

June 2017

Southwest Research Institute

Session 2:
ROS Basics Continued

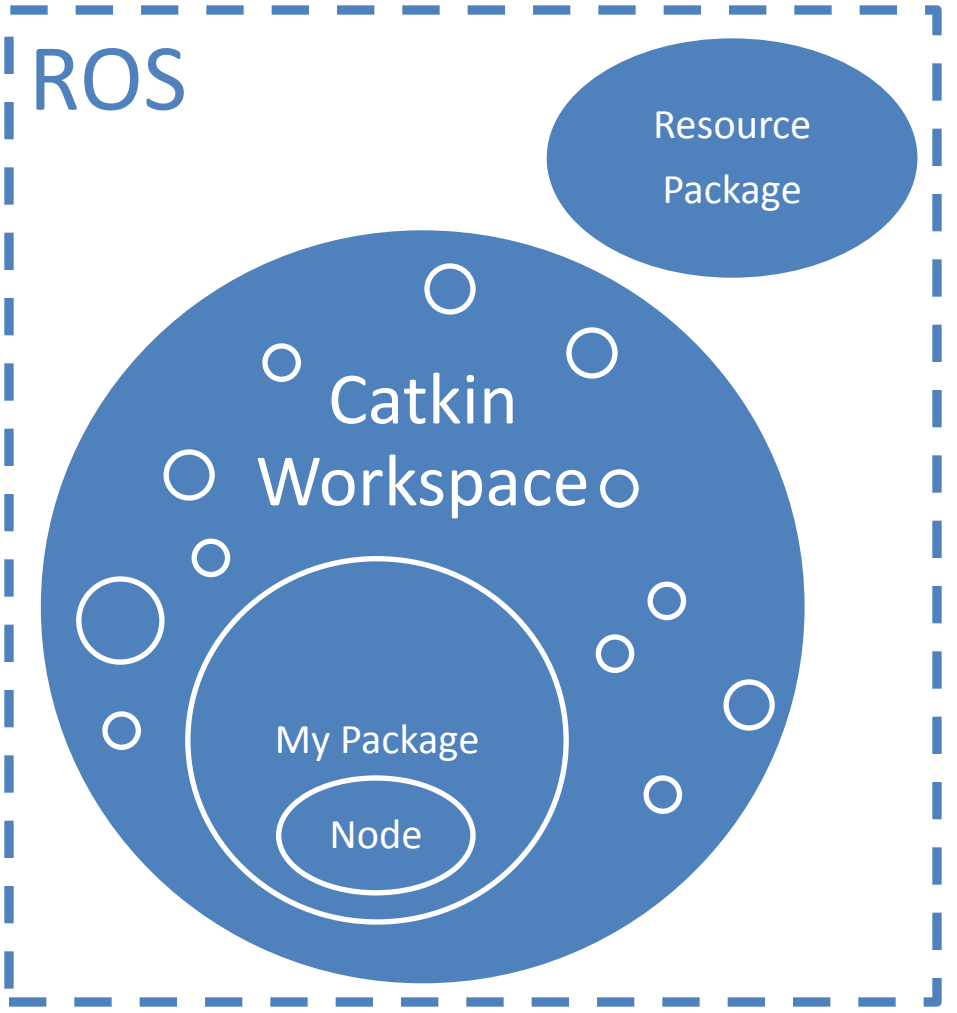
Southwest Research Institute

Outline

- Services
- *Actions*
- Launch Files
- Parameters

Day 1 Progression

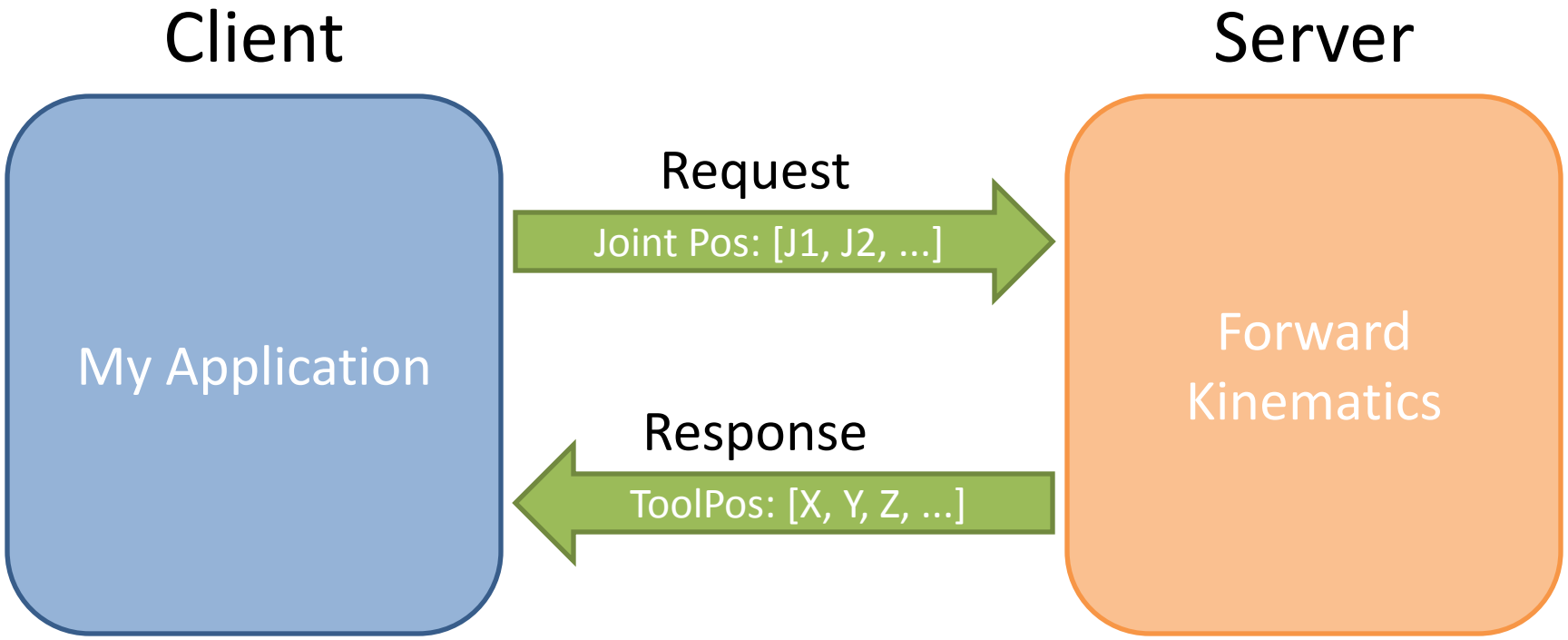
- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ✓ Create Package
- ✓ Create Node
  - ✓ Basic ROS Node
  - ✓ Interact with other nodes
    - ✓ Messages
    - ❑ Services
- ✓ Run Node
  - ✓ rosrun
  - ❑ roslaunch



Services

Services : Overview

Services are like **Function Calls**

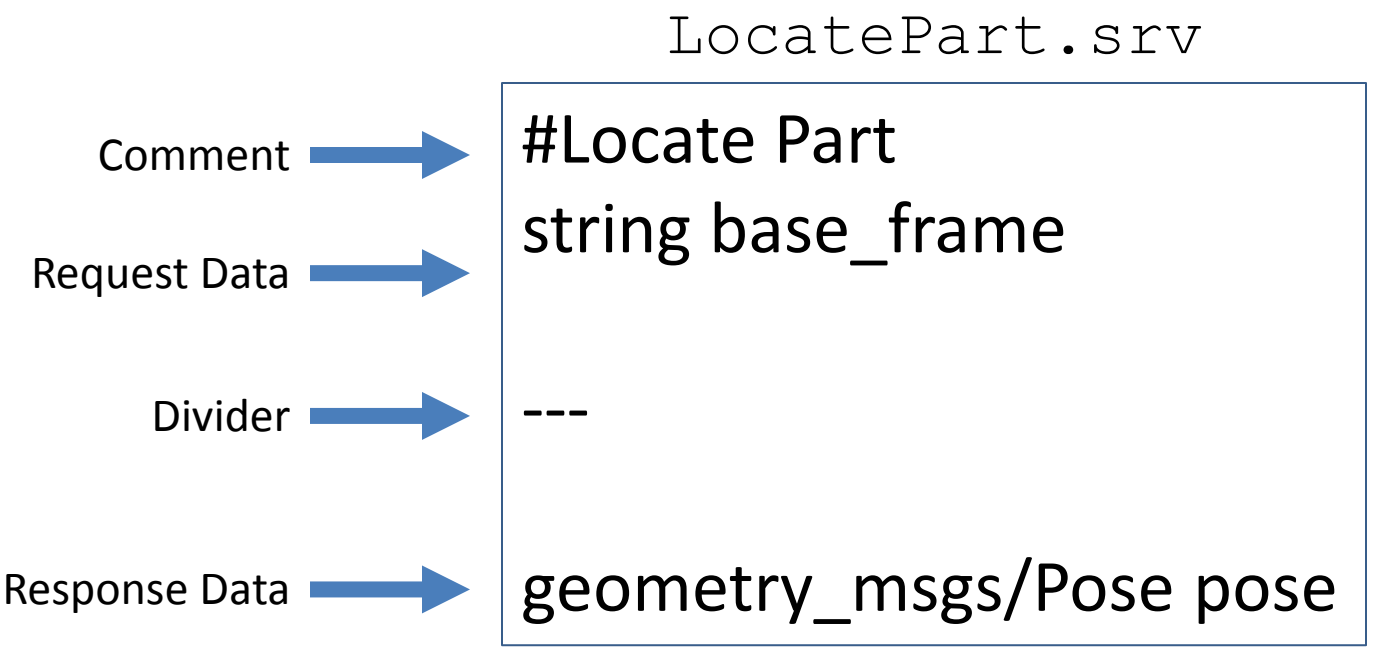


Services: Details

- 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

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.

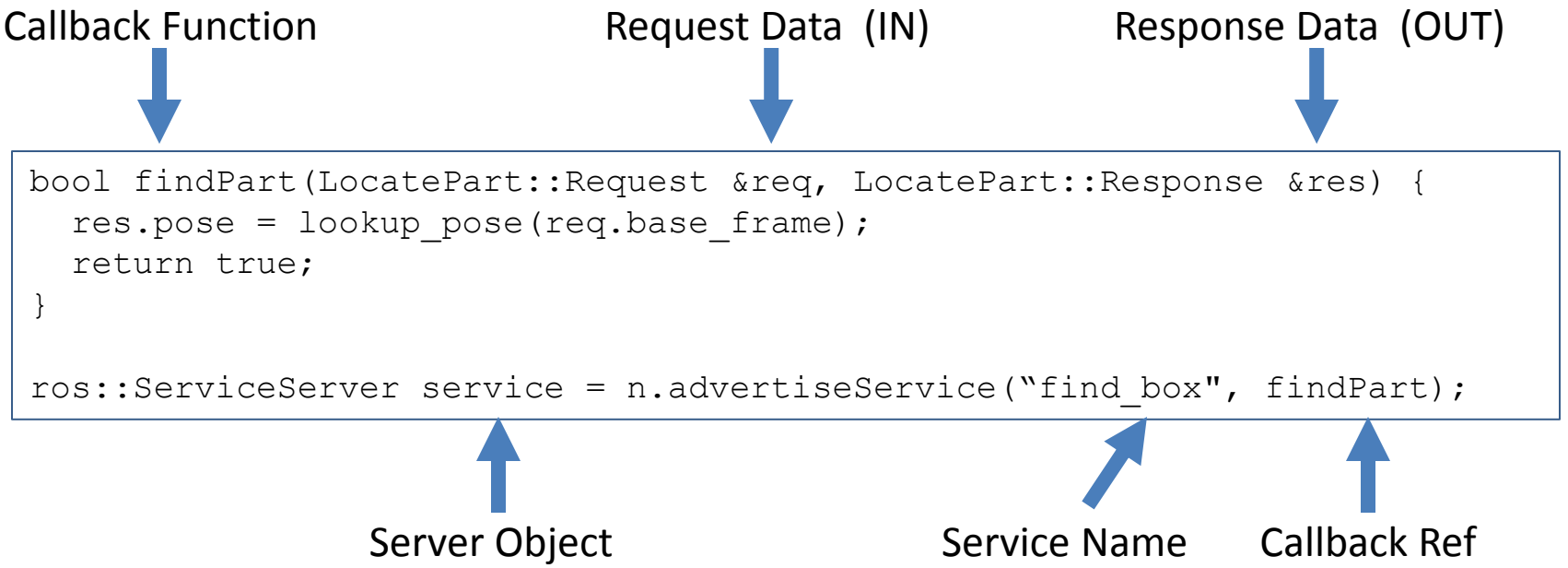


“Real World” – Services

- Use *rqt_srv / rqt_msg* to view:
  - `moveit_msgs/GetPositionIK`
  - `roscpp/SetLoggerLevel`
  - `moveit_msgs/GetMotionPlan`

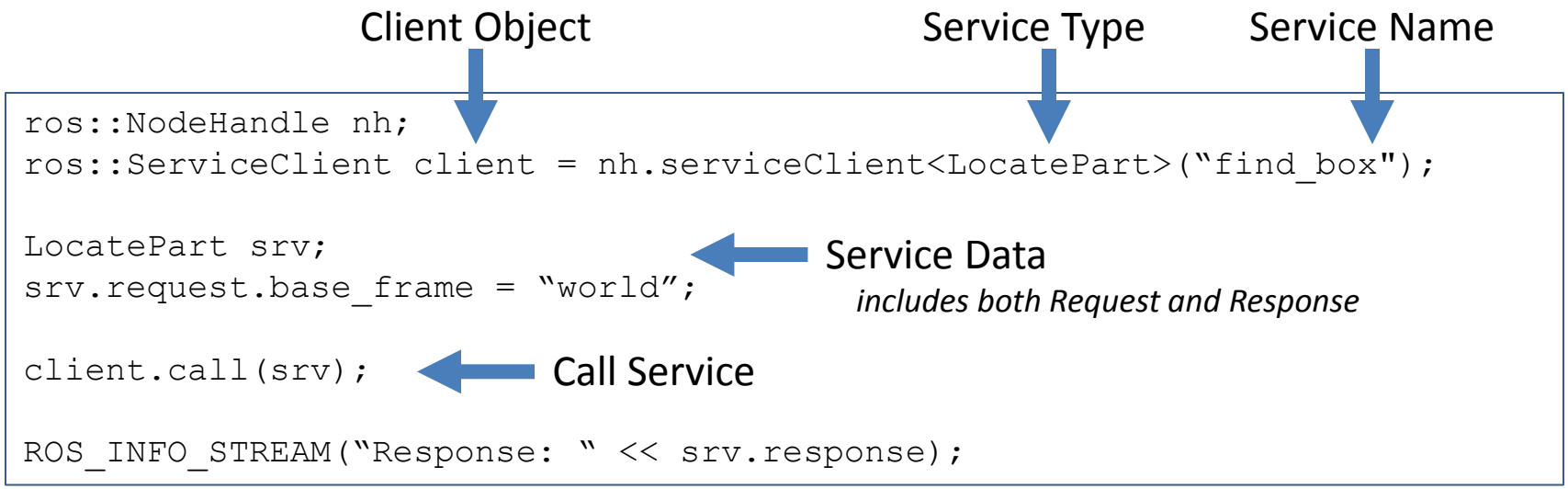
Services: Syntax

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



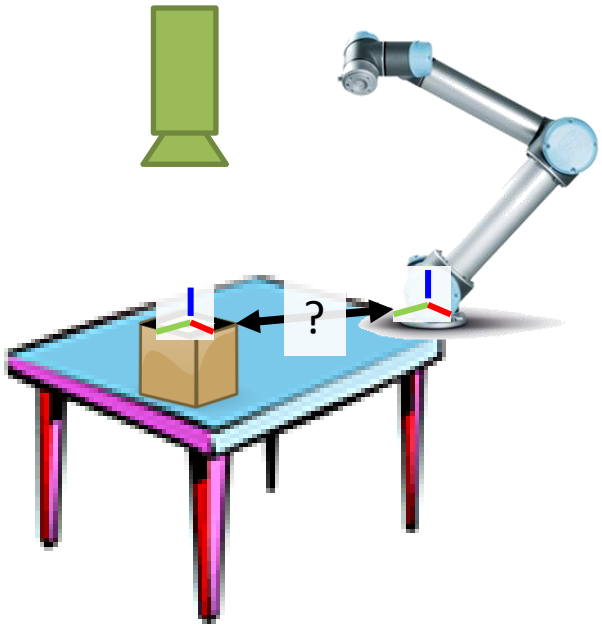
Services: Syntax

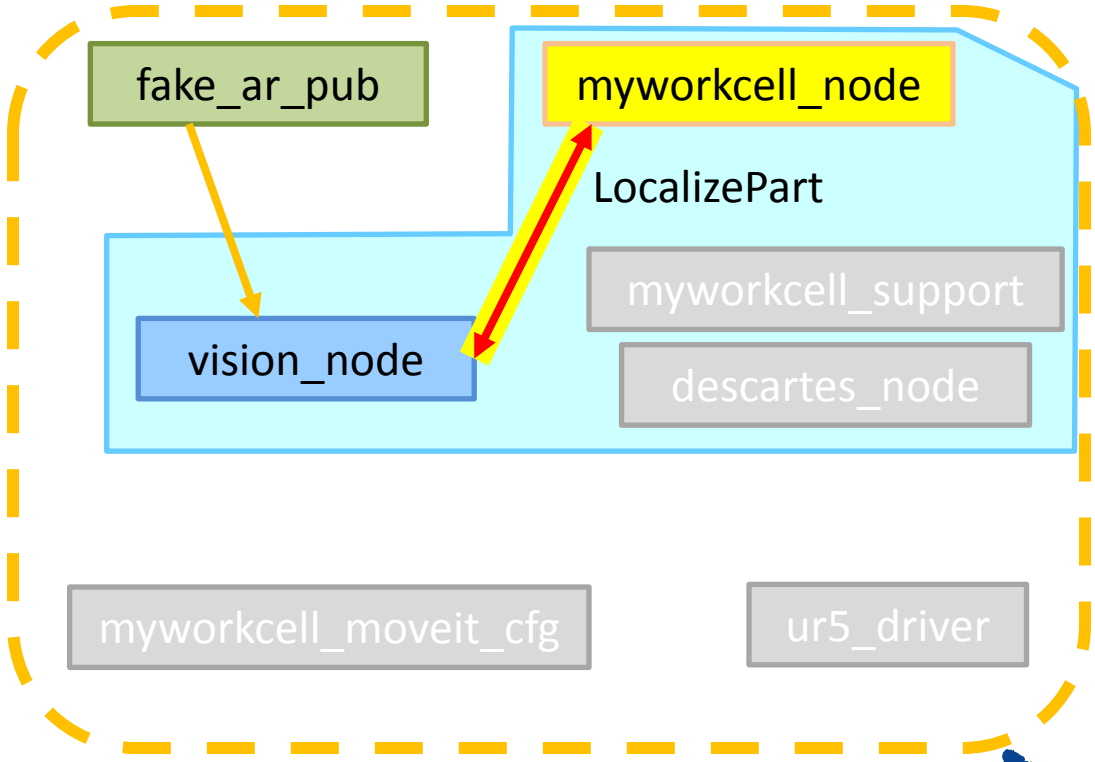
- Service **Client**
  - Connects to specific Service *(Name / Data Type)*
  - Fills in Request data
  - Calls Service



Exercise 2.0

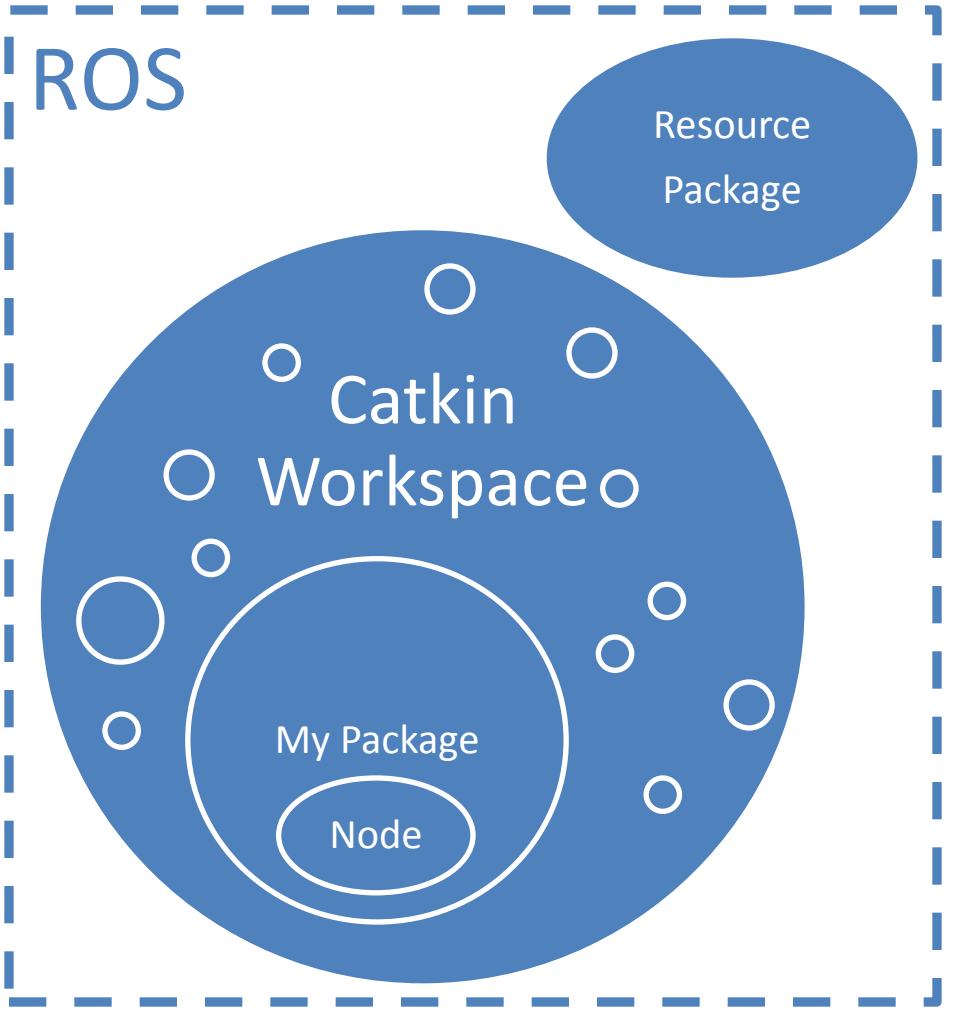
Creating and Using a Service





Day 1 Progression

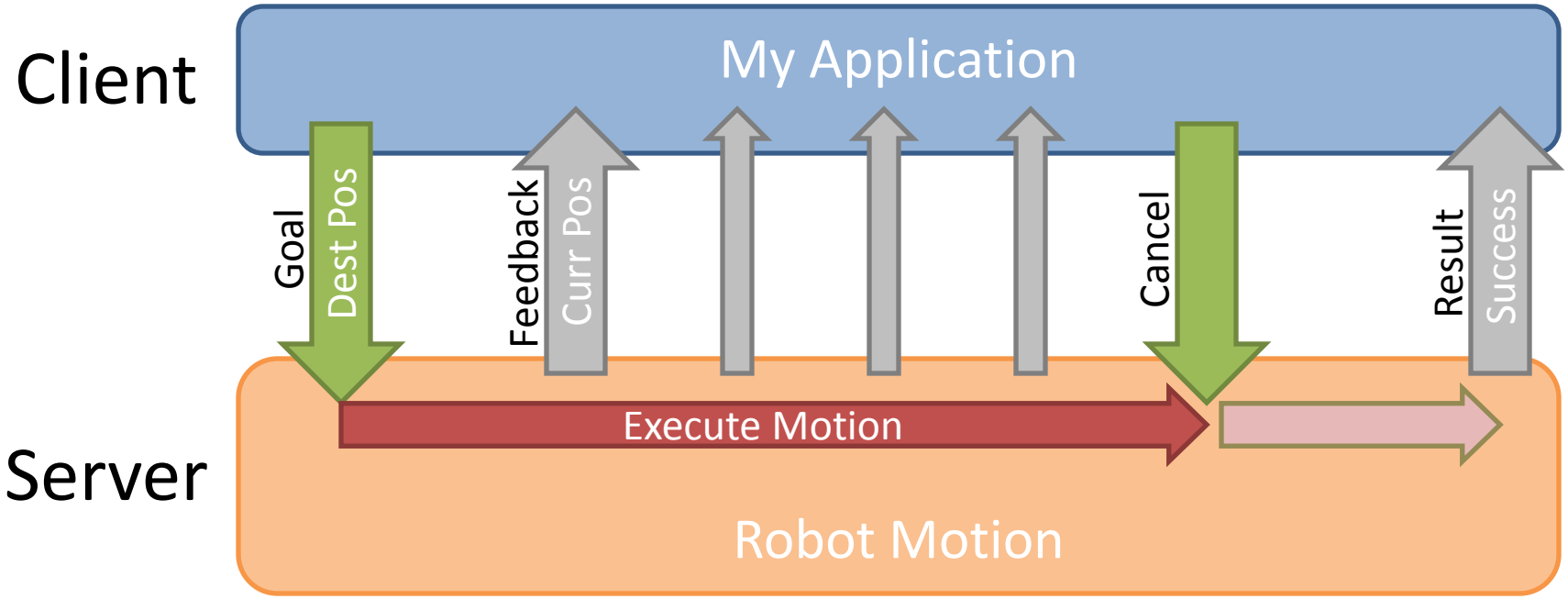
- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ✓ Create Package
- ✓ Create Node
  - ✓ Basic ROS Node
  - ✓ Interact with other nodes
    - ✓ Messages
    - ✓ Services
- ✓ Run Node
  - ✓ rosrun
  - ❑ roslaunch



Actions

Actions : Overview

Actions manage **Long-Running Tasks**



Actions: Detail

- Each 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

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.

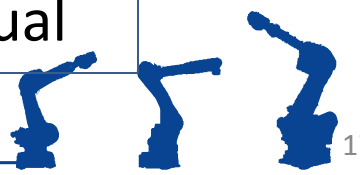
`FollowJointTrajectory.action`

```
# Command Robot Motion
traj_msgs\JointTrajectory trajectory
---
int32  error_code
string error_string
---
uint8 status
traj_msgs\JointTrajectoryPoint actual
```

Goal Data →

Result Data →

Feedback Data →



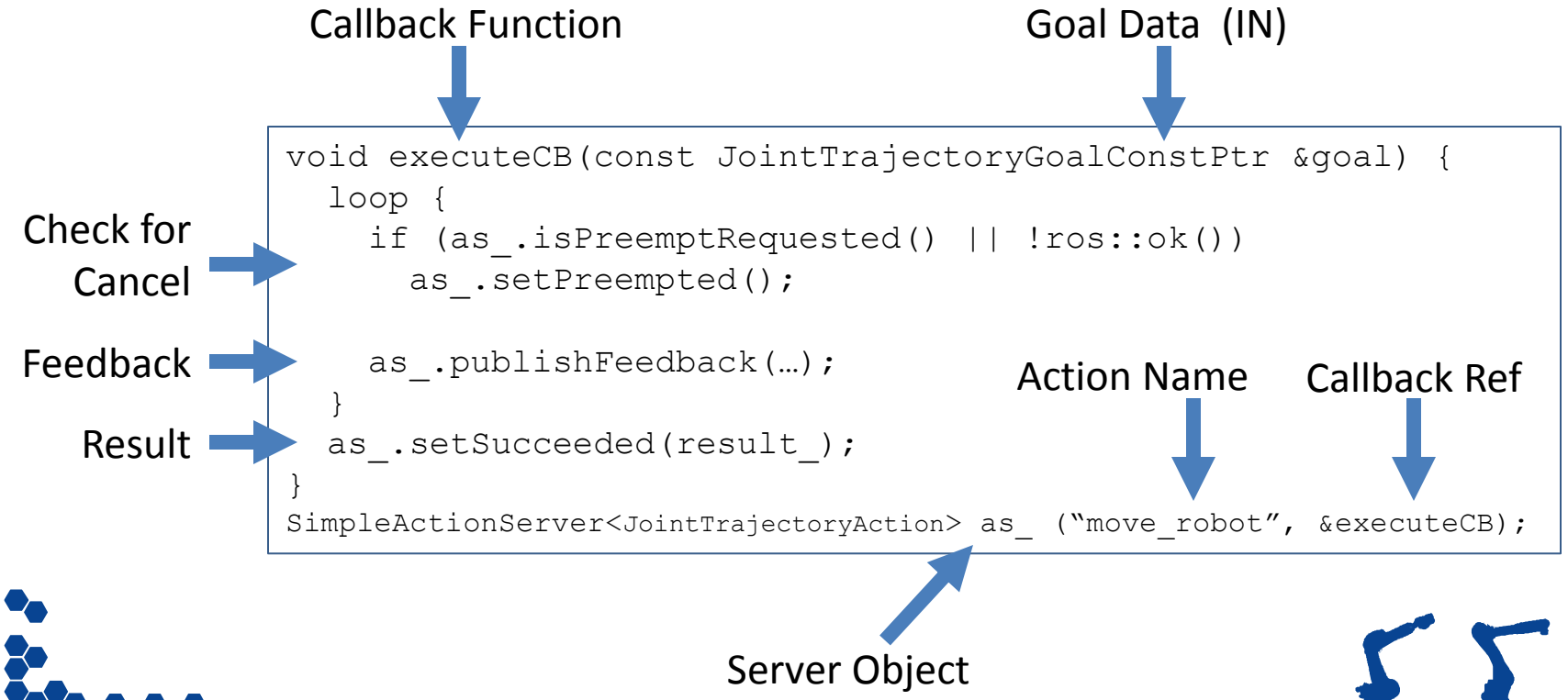
“Real World” – Actions

- FollowJointTrajectoryAction
  - command/monitor robot trajectories
  - use rqt_msg to view Goal, Result, Feedback

- Should be an Action…
  - GetMotionPlan
- Should not be an Action…
  - GripperCommandAction

Actions: Syntax

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

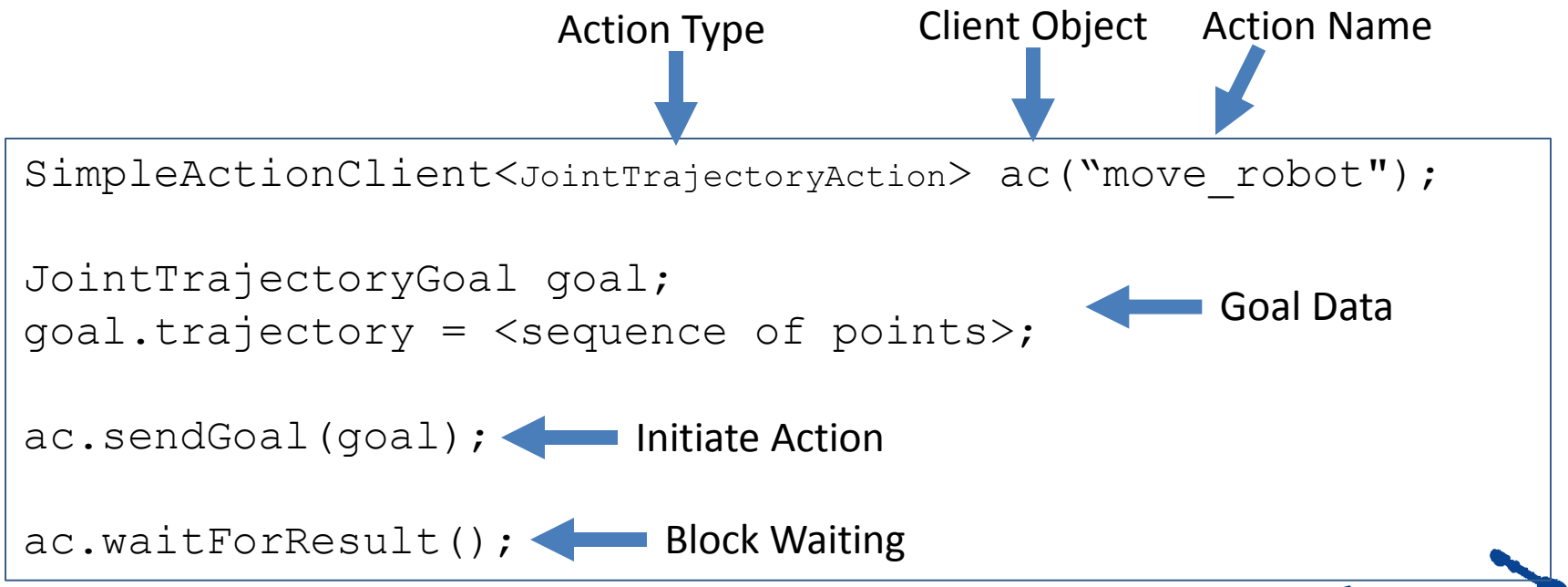



```
void executeCB(const JointTrajectoryGoalConstPtr &goal) {
  loop {
    if (as_.isPreemptRequested() || !ros::ok())
      as_.setPreempted();

    as_.publishFeedback(…);
  }
  as_.setSucceeded(result_);
}
SimpleActionServer<JointTrajectoryAction> as_ ("move_robot", &executeCB);
```

Actions: Syntax

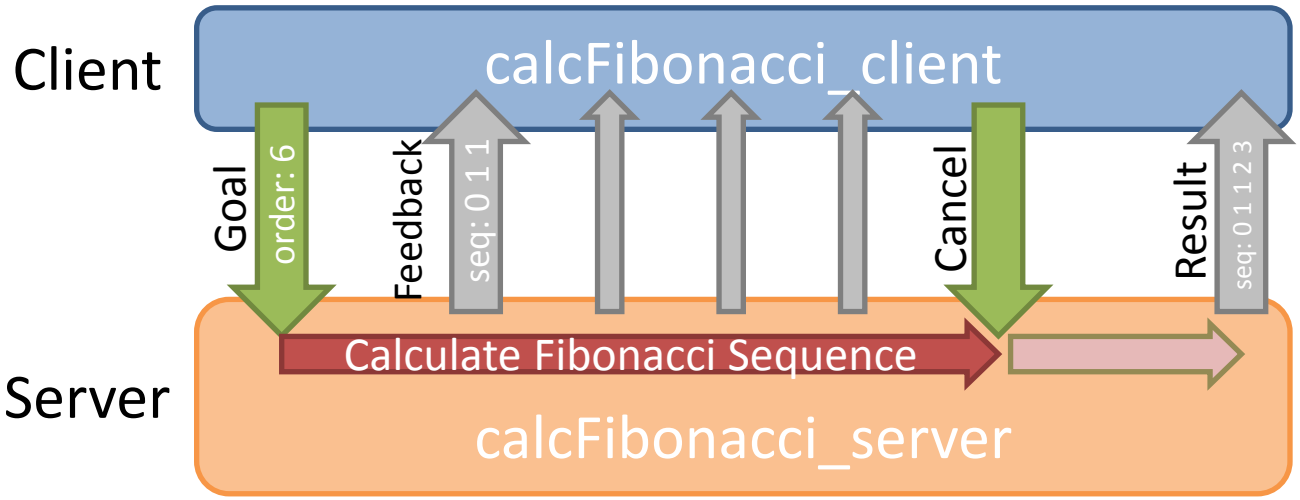
- Action **Client**
  - Connects to specific Action *(Name / Data Type)*
  - Fills in Goal data
  - Initiate Action / Waits for Result



Exercise 2.1

Creating and Using an Action

We'll skip this exercise.
Work through it on your own time later, if desired.



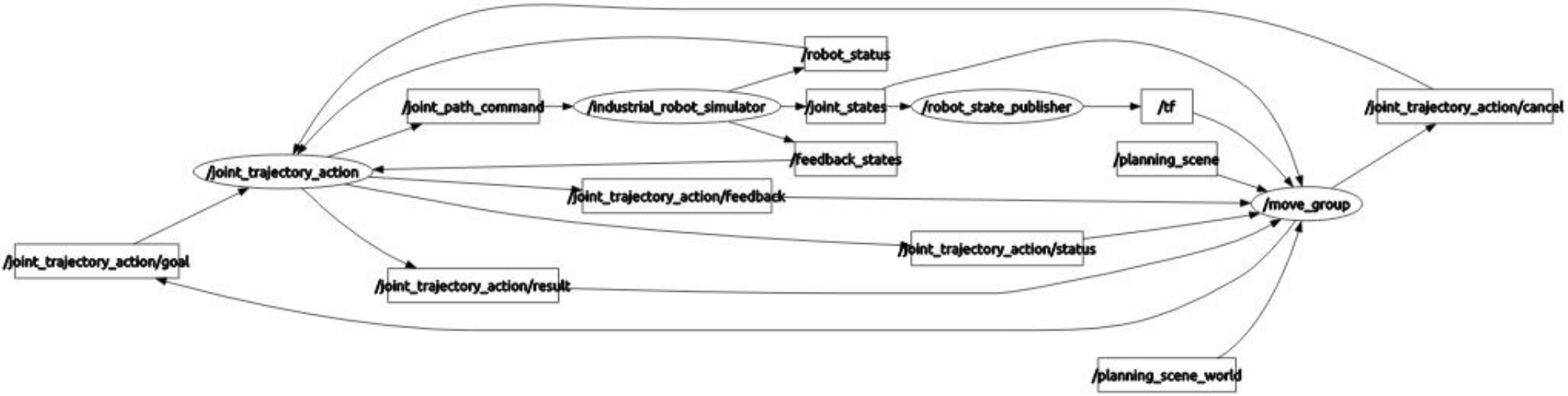
Message vs. Service vs. Action

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

Launch Files

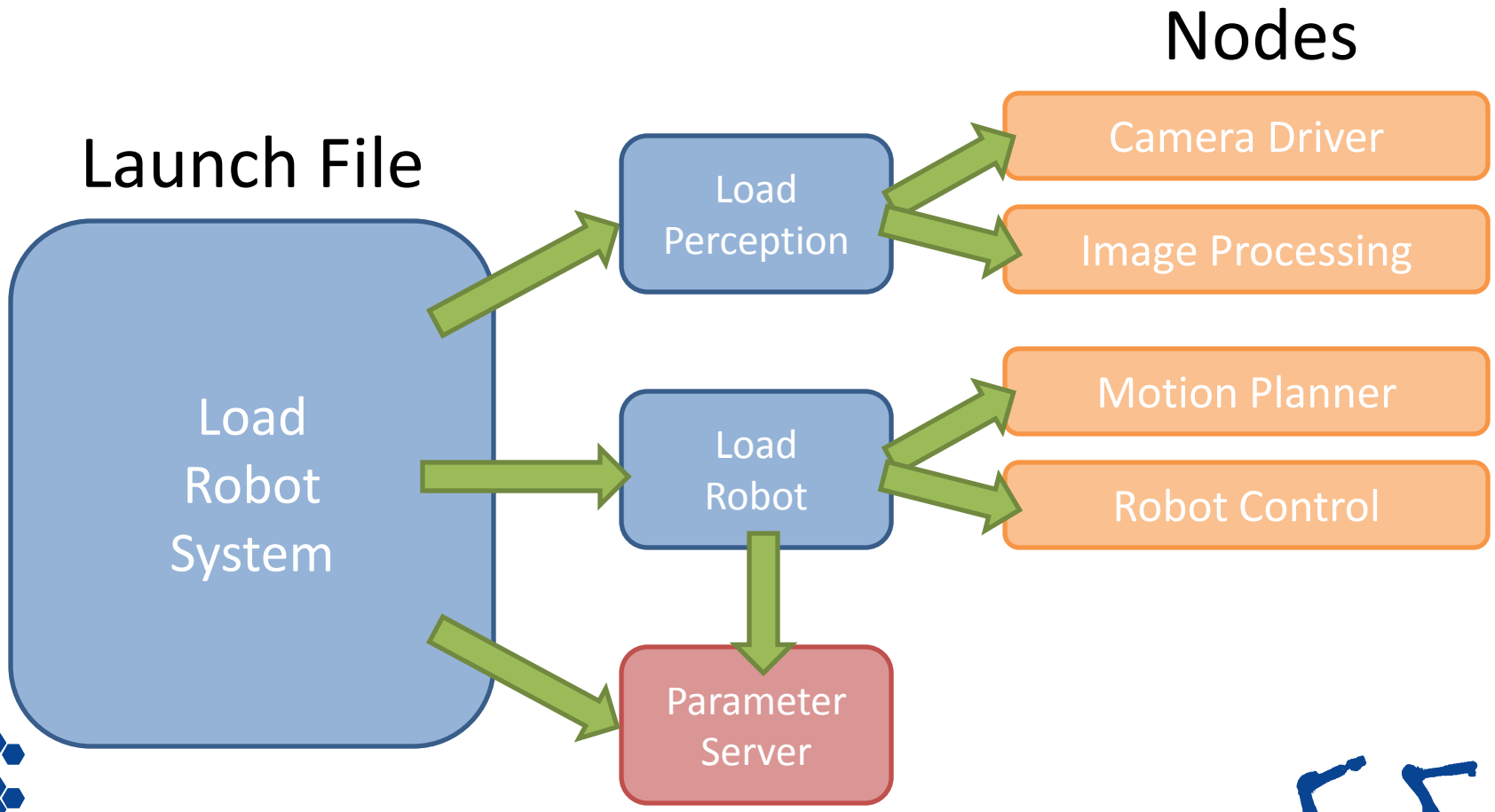
Launch Files: Motivation

- ROS is a Distributed System
  - often 10s of nodes, plus configuration data
  - painful to start each node “manually”



Launch Files: Overview

Launch Files are like **Startup Scripts**



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**

Launch Files: Notes

- Can launch *other* launch files
- Executed in order, without pause or wait*

  * *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

Launch Files: Syntax (Basic)

- **<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>
```

Launch Files: Syntax (Adv.)

- **<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.

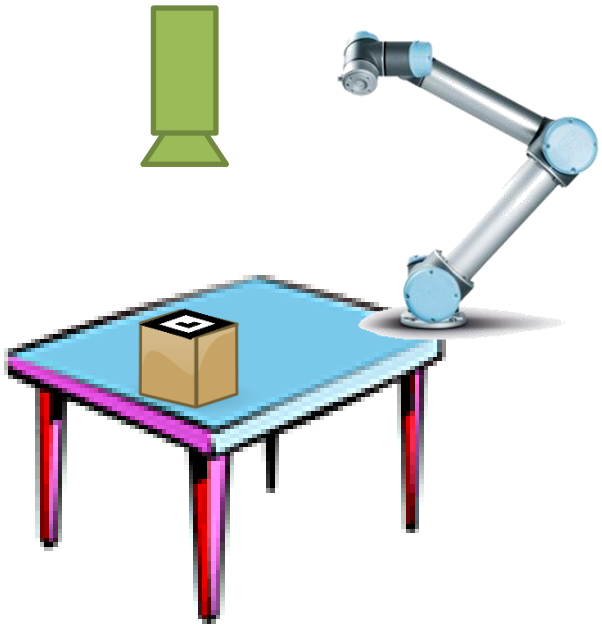
```
<launch>
  <arg name="robot" default="sia20" />
  <arg name="show_rviz" default="true" />
  <group ns="robot" >
    <include file="$(find lesson)/launch/load_$(arg robot)_data.launch" />
     <remap from="joint_trajectory_action" to="command" />
  </group>
  <node name="rviz" pkg="rviz" type="rviz"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

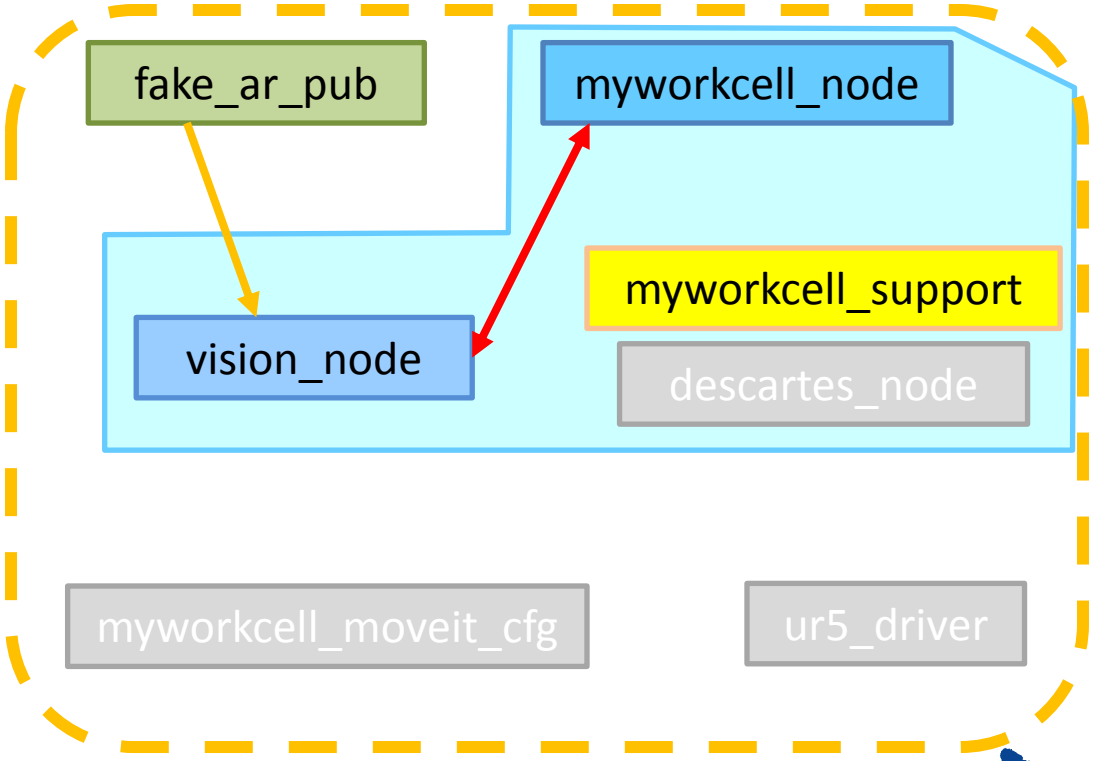
“Real World” – Launch Files

- Explore a typical robot launch file
  - 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>
```

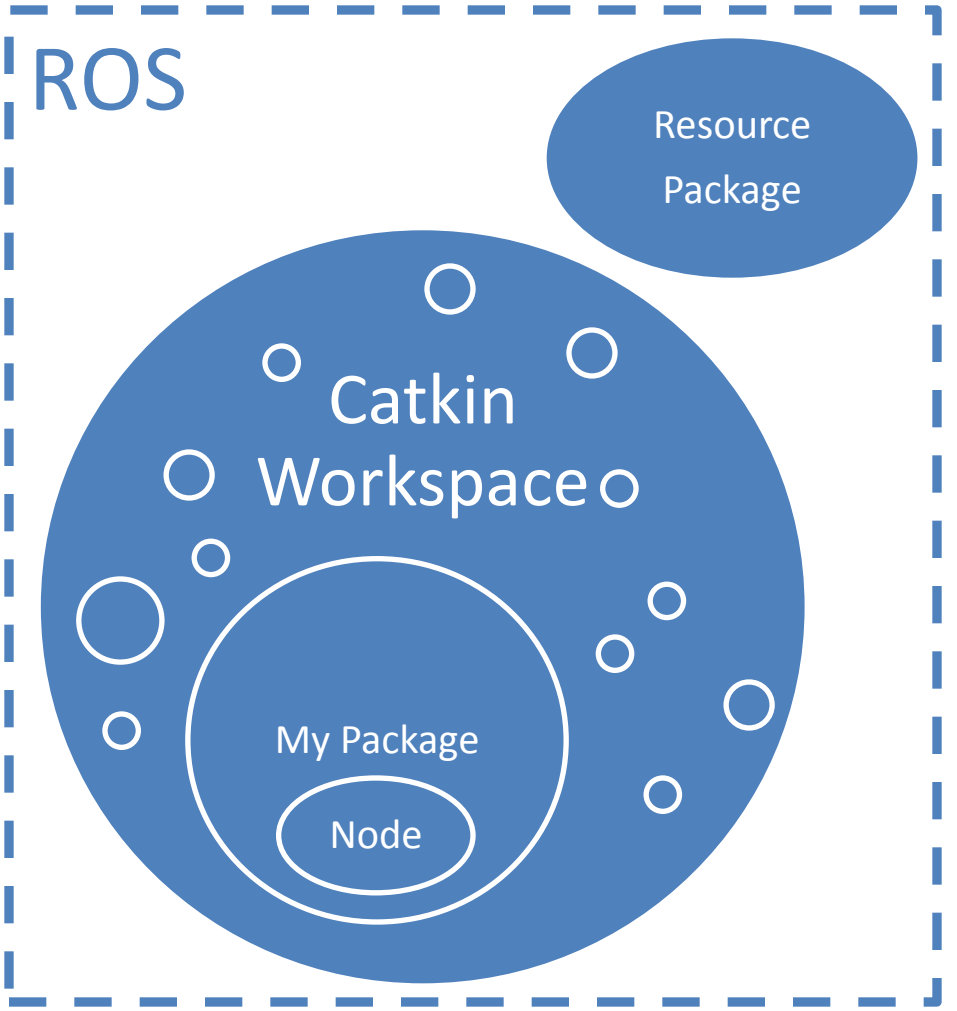
Exercise 2.2 - Launch Files





Day 1 Progression

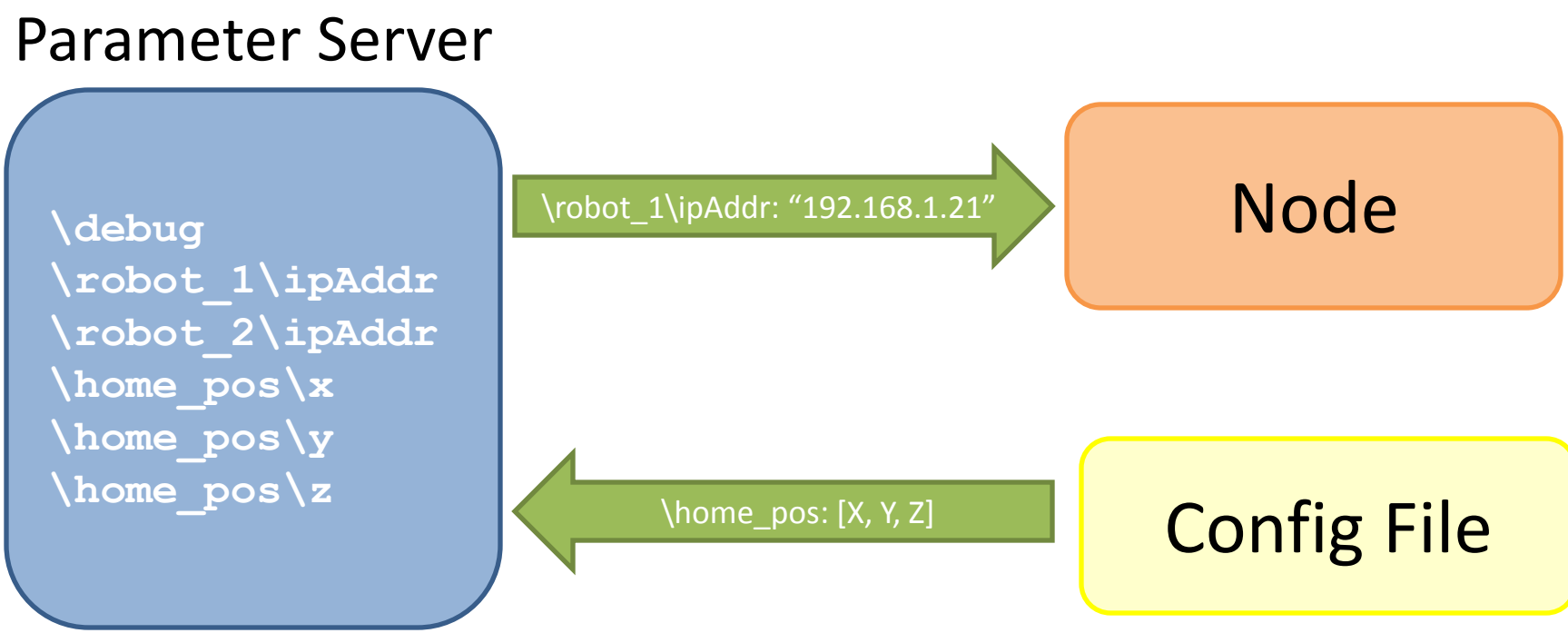
- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ✓ Create Package
- ✓ Create Node
  - ✓ Basic ROS Node
  - ✓ Interact with other nodes
    - ✓ Messages
    - ✓ Services
- ✓ Run Node
  - ✓ rosrun
  - ✓ roslaunch



Parameters

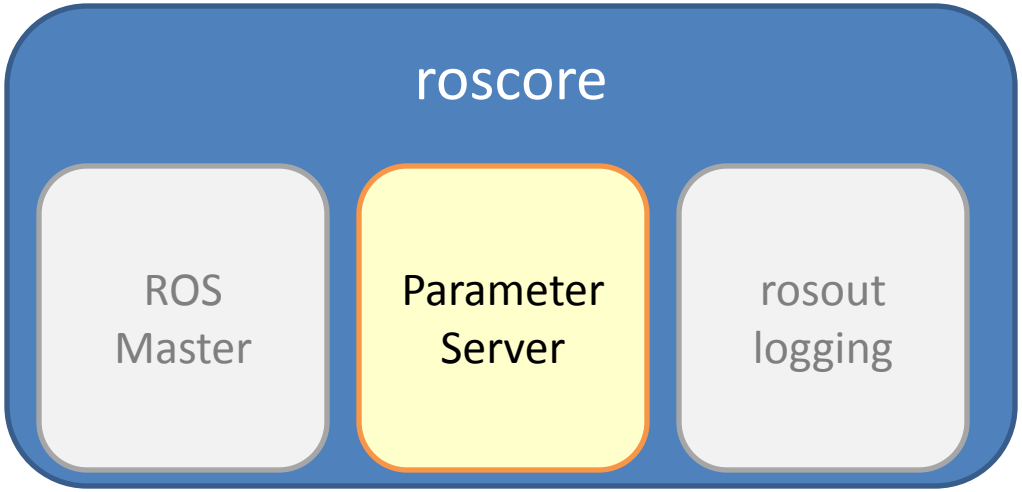
Parameters: Overview

Parameters are like **Global Data**



ROS Parameters

- Typically configuration-type values
  - robot kinematics
  - workcell description
  - algorithm limits / tuning
- Accessed through the **Parameter Server**.
  - *Typically handled by **roscore***



Setting Parameters

- Can set from:
  - YAML Files

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

  - Command Line

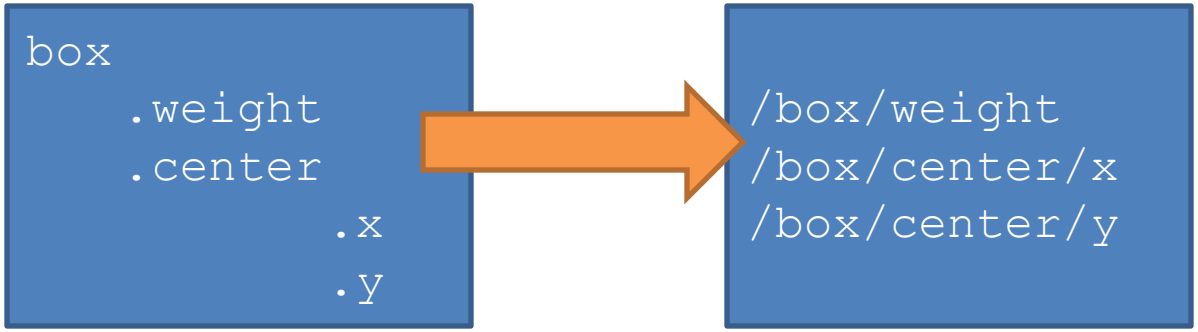
```
rosrun my_pkg load_robot _ip:="192.168.1.21"
rosparam set "/debug" true
```

  - Programs

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

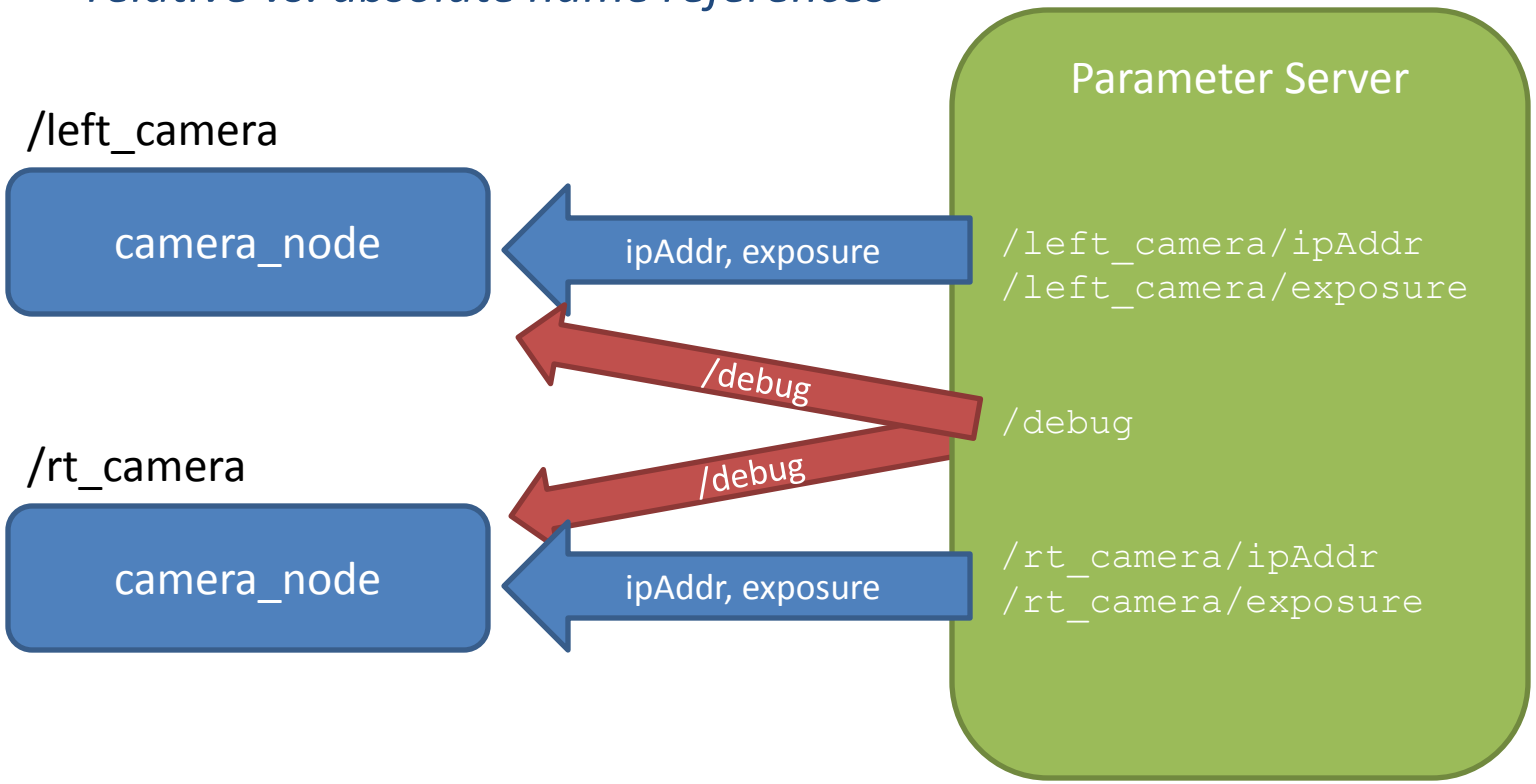
Parameter Datatypes

- 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*



Namespaces

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



Parameter Commands

- **`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

Parameters: C++ API

- Accessed through `ros::NodeHandle` object
  - also sets default **Namespace** for access
    - Relative namespace:

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

"/**<ns>**/test"

    - Fixed namespace:

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

"/myApp/test"

    - Private namespace:

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

"/myNode/test"

Parameters: C++ API (cont'd)

- `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)`

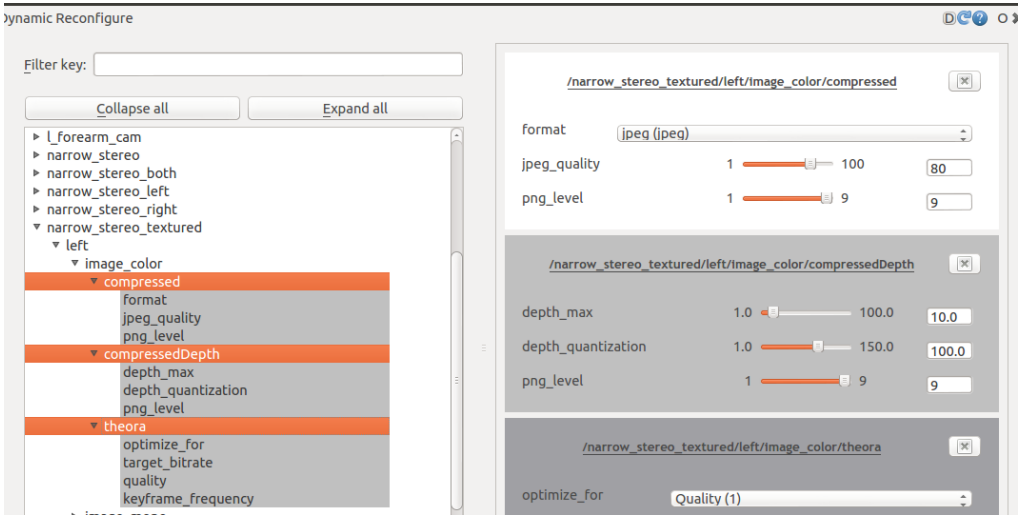
    Get value (or default, if doesn't exist)
  - `nh.setParam(key, value)`

    Sets value
  - `nh.deleteParam(key)`

    Deletes parameter

Dynamic reconfigure

- 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

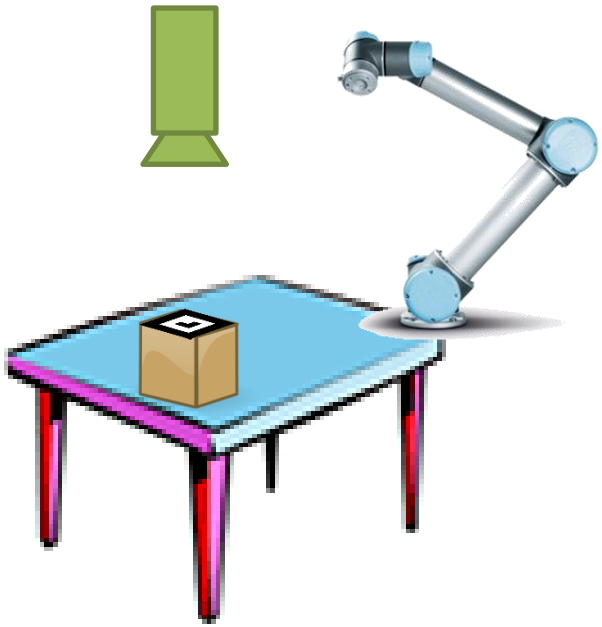


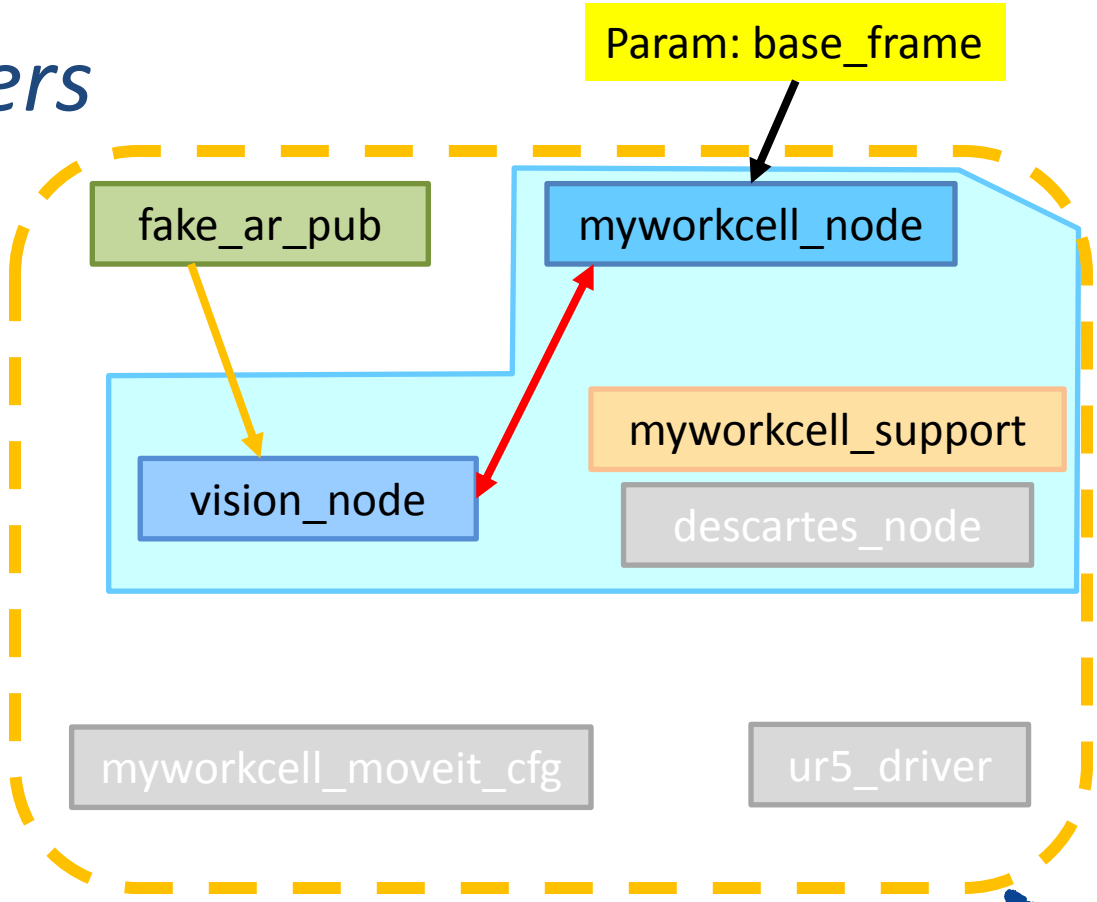
ROS Param Practical Examples

- Let's see what parameters the UR5 driver uses:
  - Prefix
  - robot_ip_address
  - max_velocity
  - servoj_time
  - Etc…

Exercise 2.3

ROS Parameters





Review/Q&A

Session 1

Intro to ROS

Installing ROS/Packages

Packages

Nodes

Messages/Topics

Session 2

Services

Actions

Launch Files

Parameters