ROS-Industrial Basic Developer's Training Class

February 2017

Southwest Research Institute

Session 2:
ROS Basics Continued

February 2017

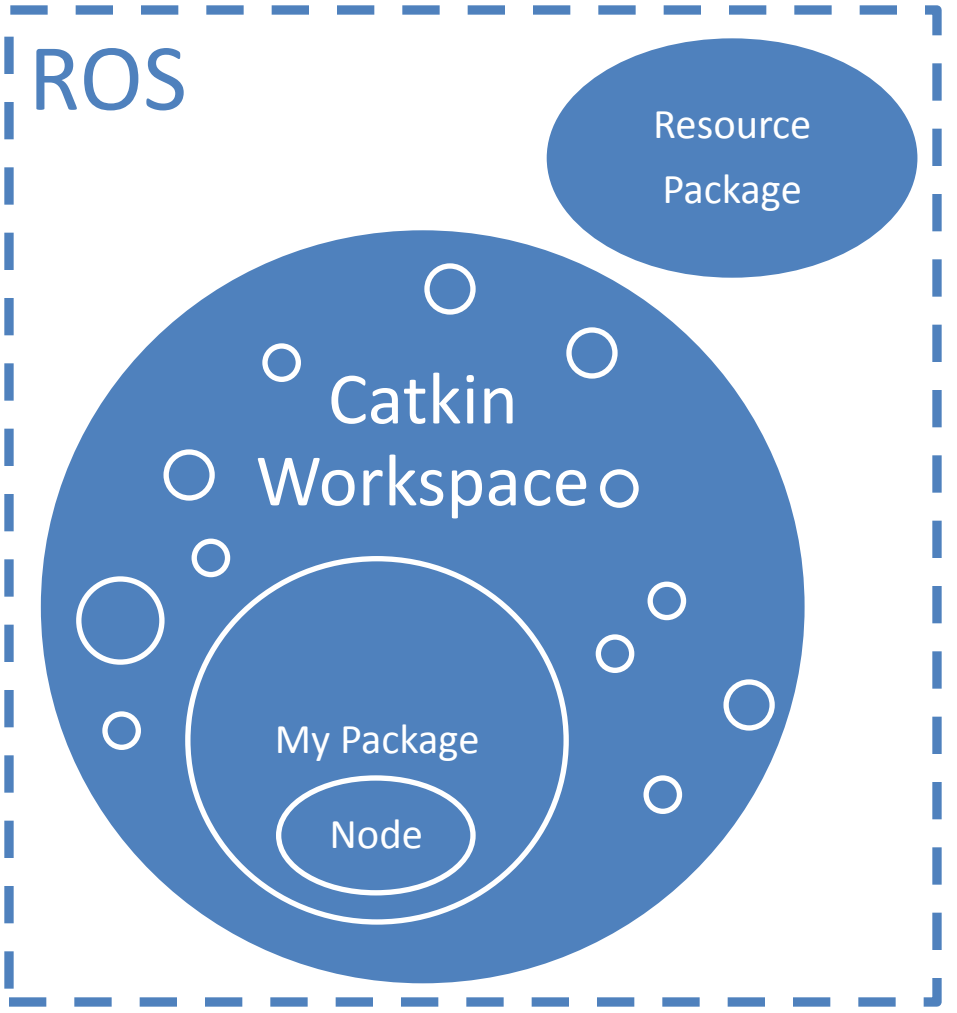
Southwest Research Institute

Outline

- Services
- *Actions*
- Launch Files
- Parameters

Day 1 Progression

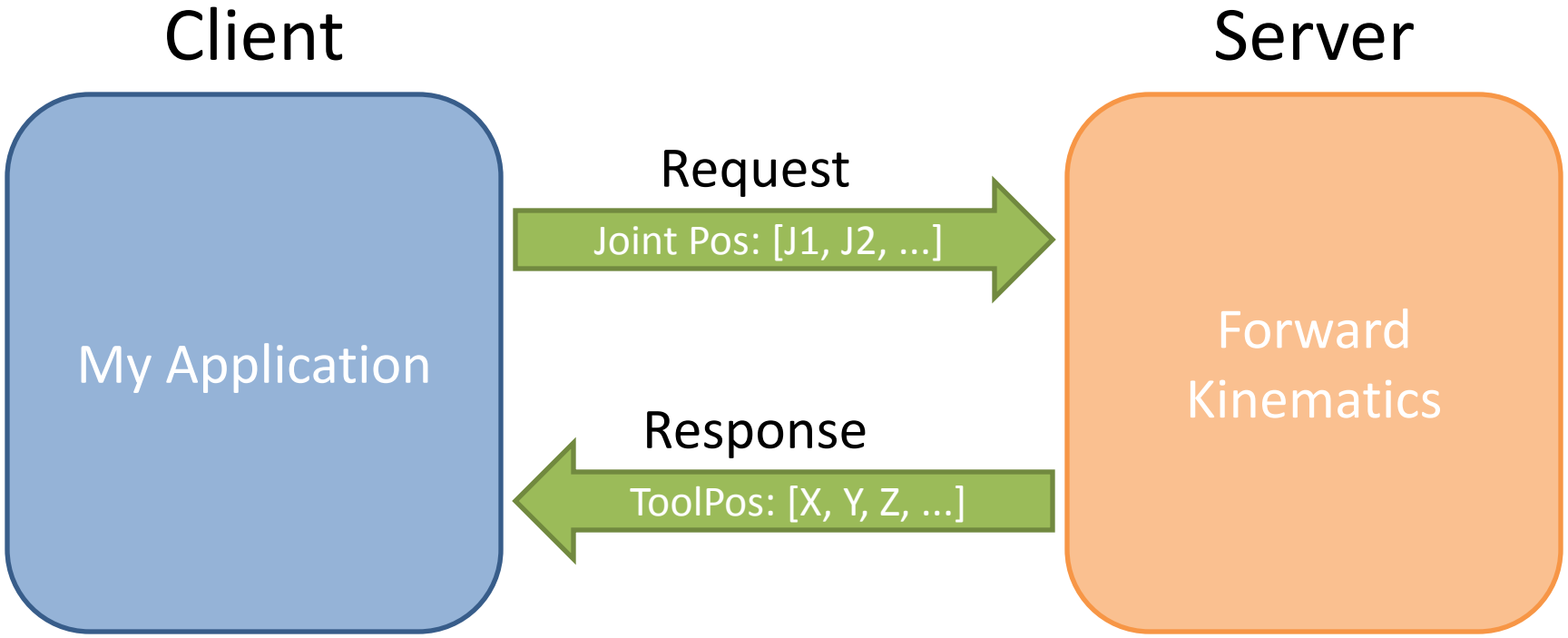
- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ✓ Create Package
- ✓ Create Node
  - ✓ Basic ROS Node
  - ✓ Interact with other nodes
    - ✓ Messages
    - ❑ Services
- ✓ Run Node
  - ✓ rosrun
  - ❑ roslaunch



Services

Services : Overview

Services are like **Function Calls**



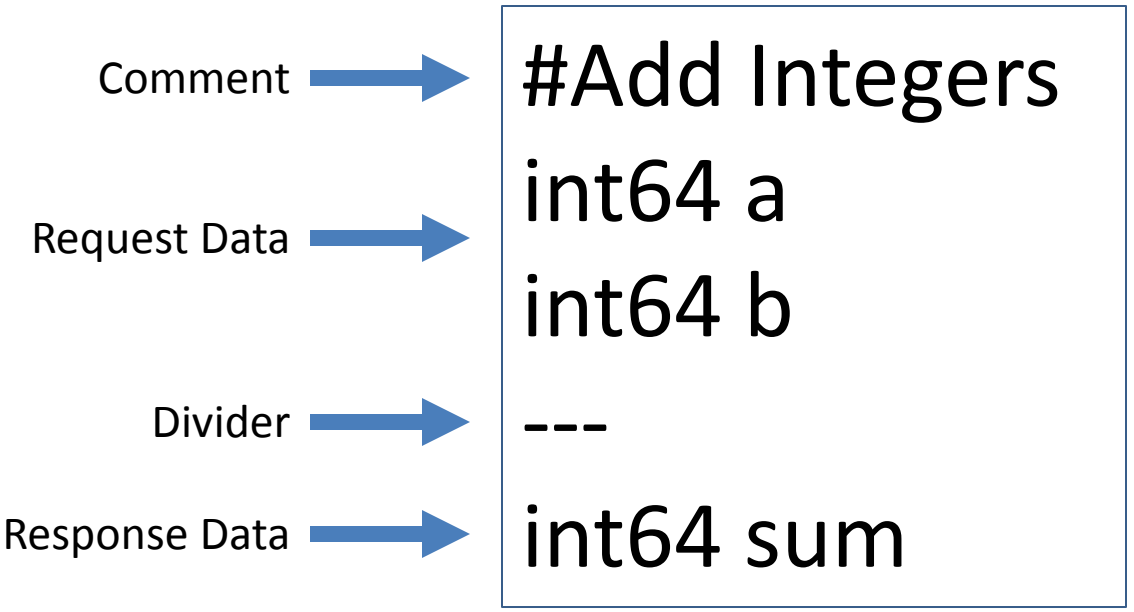
Services: Details

- 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

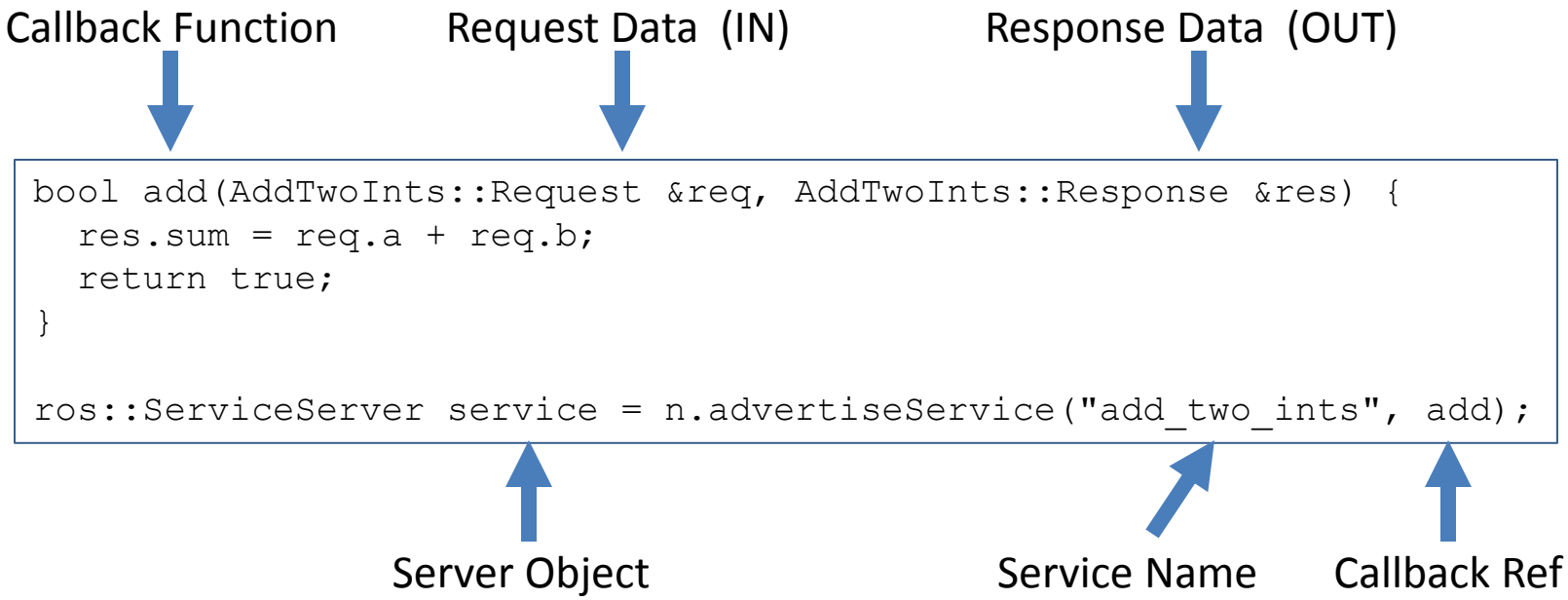
- Service **definition**
  - Defines Request and Response **data types**
    - *Either/both data type(s) may be **empty.** Always receive "completed" handshake.*
  - Auto-generates C++ Class files (.h/.cpp), Python, etc.

AddTwoInts.srv



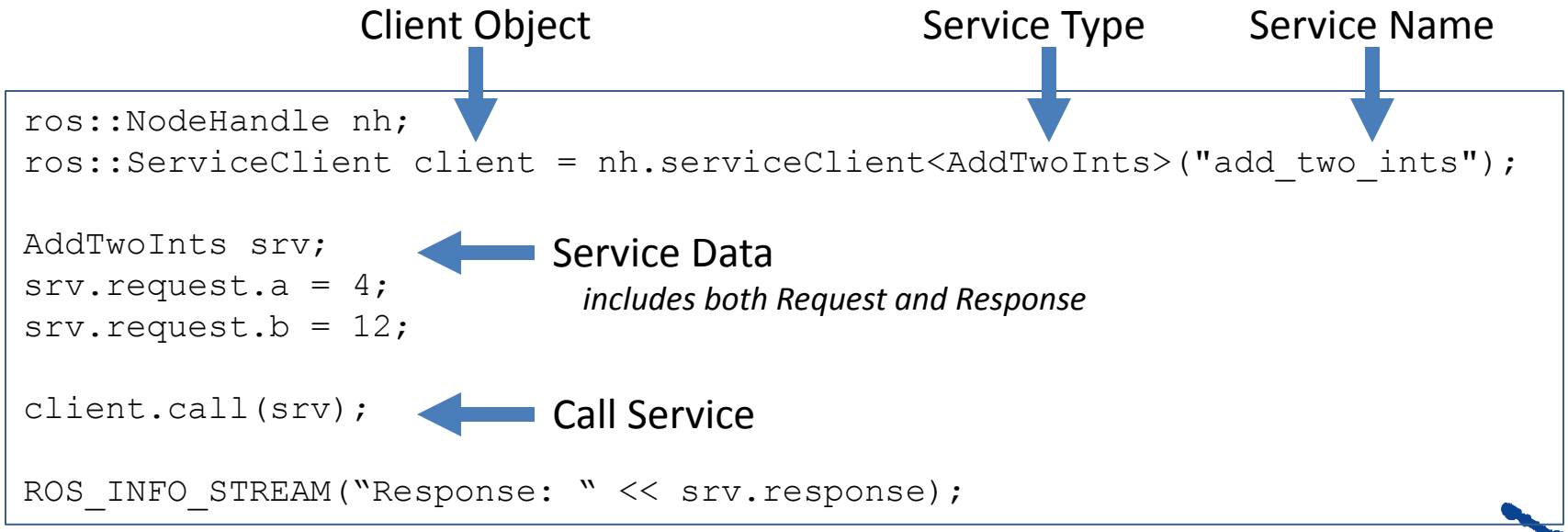
Services: Syntax

- 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

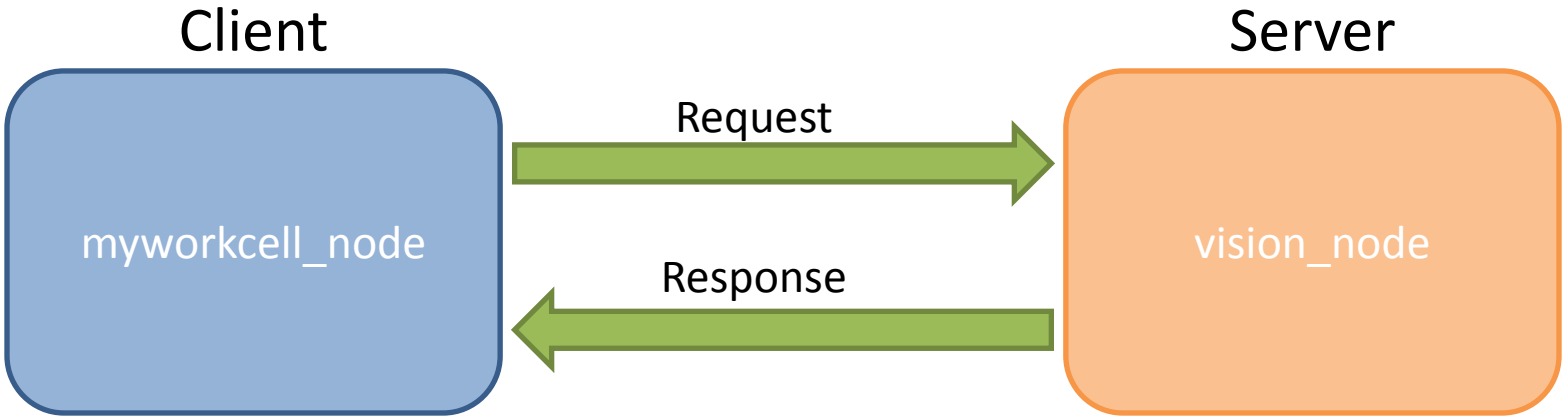


ROS Service Practical Examples

- Let's look at some real service calls:
  - *MoveIt calls to compute paths, IK, etc…*
  - *ROS Node debug levels*
  - *Even robot control modes*

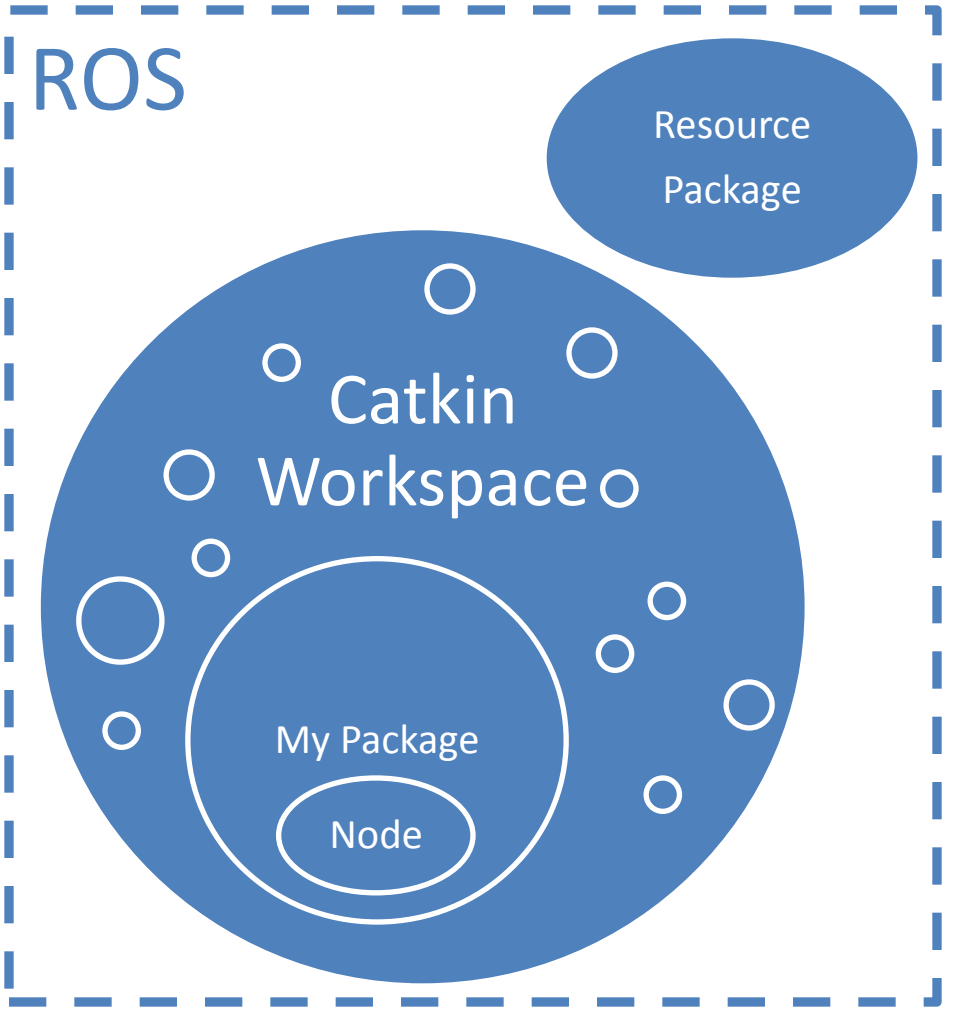
Exercise 2.0

Creating and Using a Service



Day 1 Progression

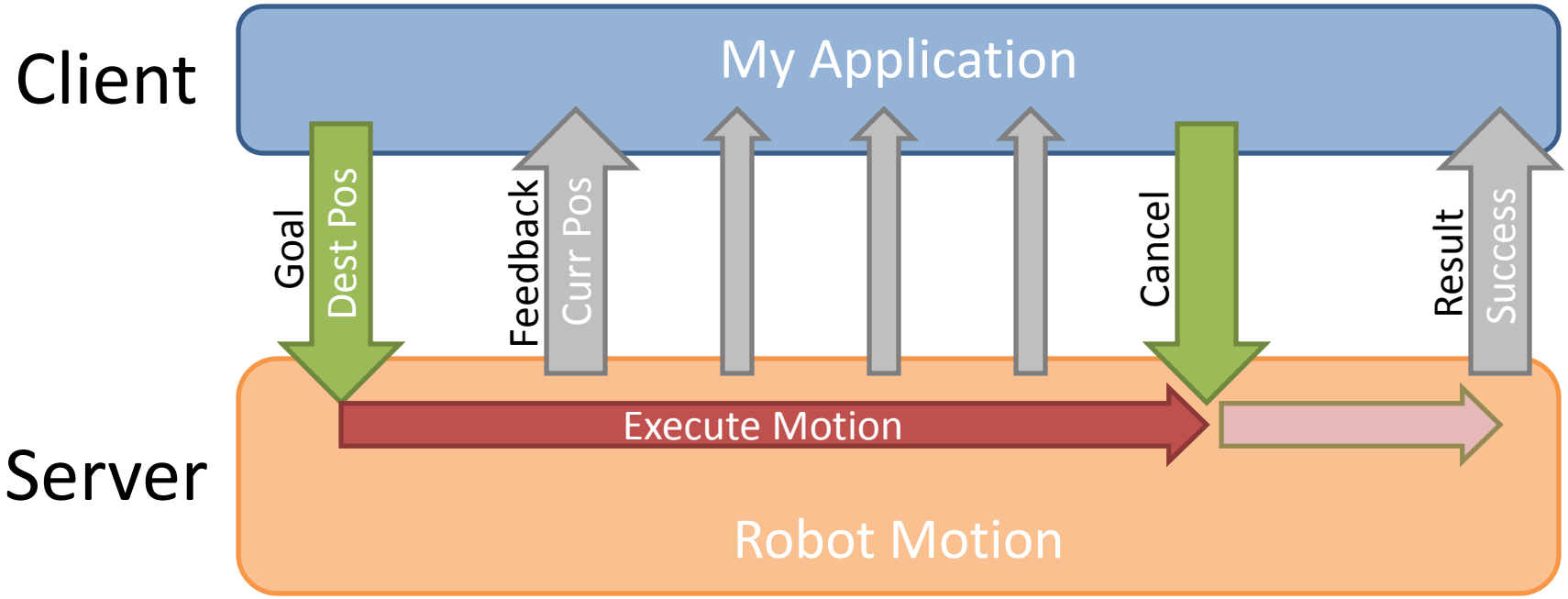
- ✓ Install ROS
- ✓ Create Workspace
- ✓ Add “resources”
- ✓ Create Package
- ✓ Create Node
  - ✓ Basic ROS Node
  - ✓ Interact with other nodes
    - ✓ Messages
    - ✓ Services
- ✓ Run Node
  - ✓ rosrun
  - ❑ roslaunch



Actions

Actions : Overview

Actions manage **Long-Running Tasks**

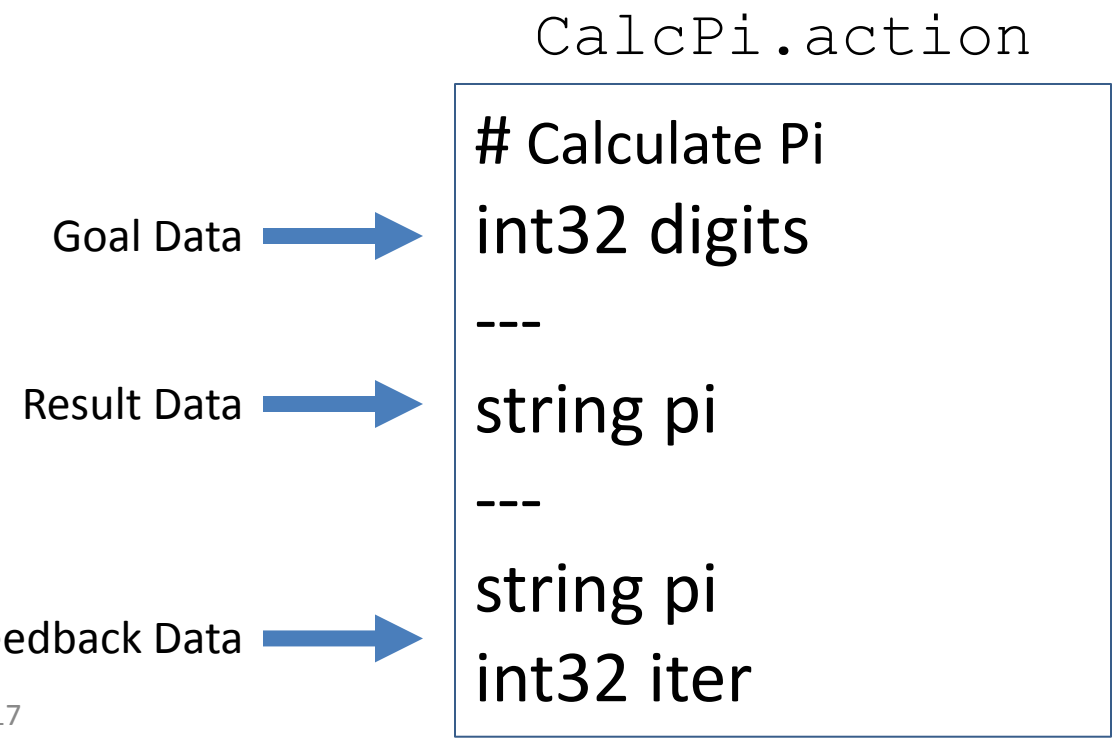


Actions: Detail

- 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

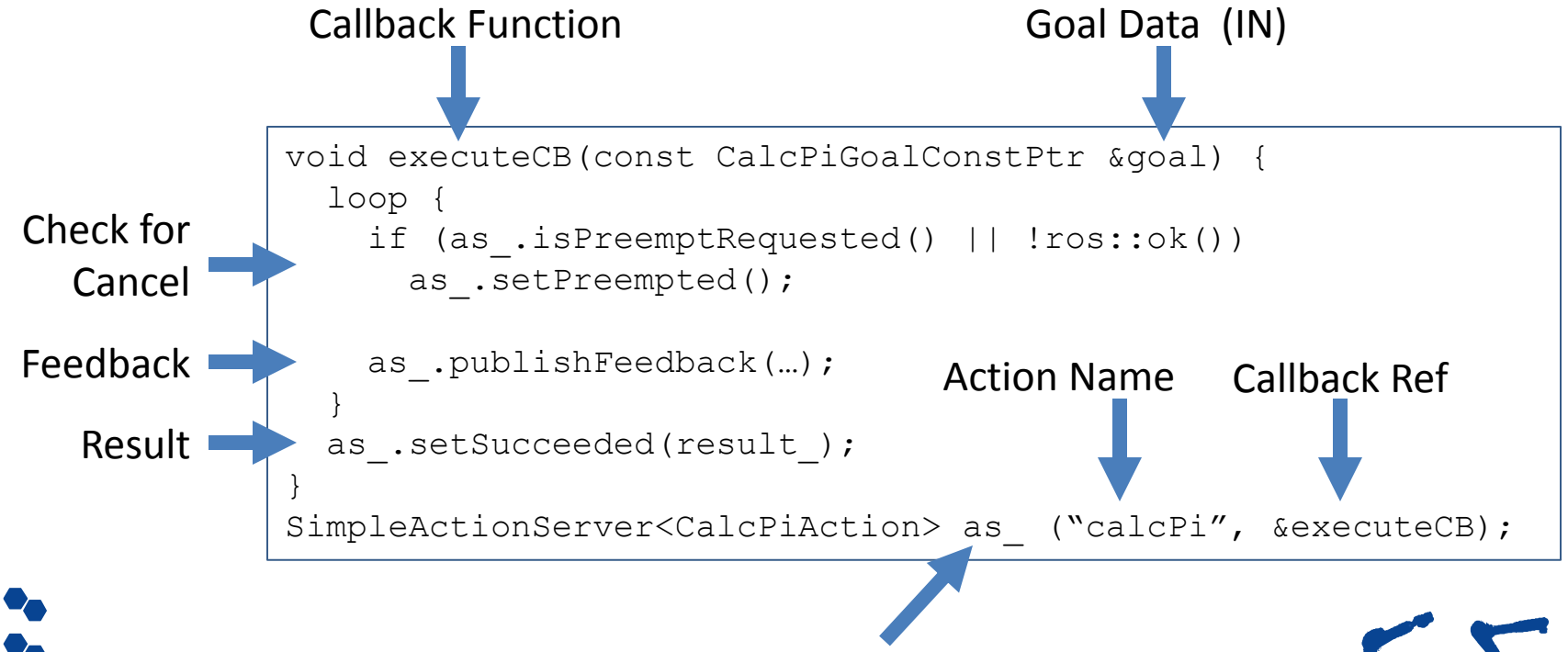
Actions: Syntax

- 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.



Actions: Syntax

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

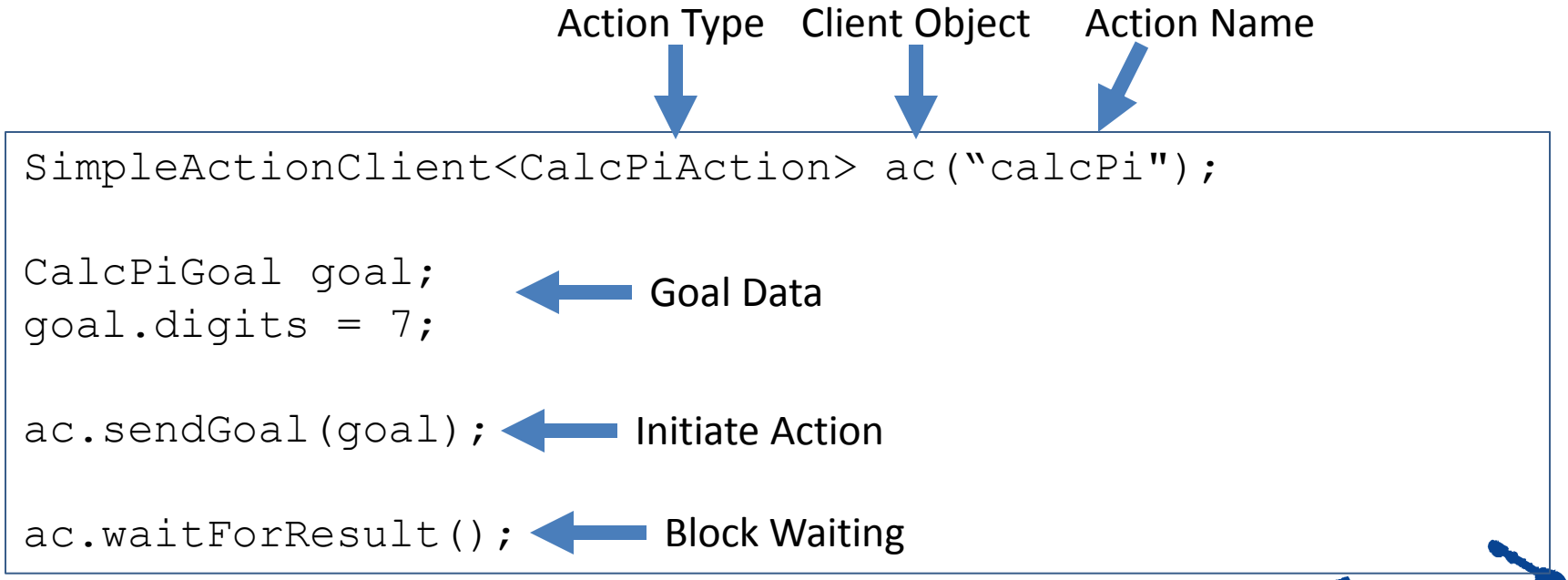



```
void executeCB(const CalcPiGoalConstPtr &goal) {
  loop {
    if (as_.isPreemptRequested() || !ros::ok())
      as_.setPreempted();

    as_.publishFeedback(…);
  }
  as_.setSucceeded(result_);
}
SimpleActionServer<CalcPiAction> as_ (“calcPi”, &executeCB);
```

Actions: Syntax

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



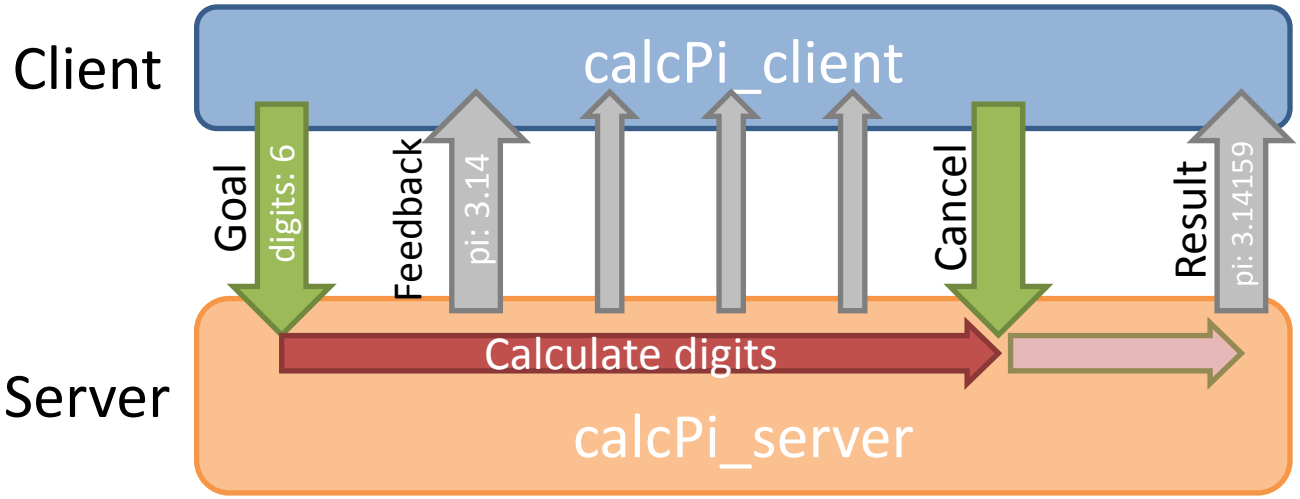
ROS Action Practical Examples

- Actions are used to monitor the status of robot trajectories.
- If we list all topics, notice we have:
  - Result
  - Goal
  - Feedback
  - Cancel

Exercise 2.1

Creating and Using an Action

This Exercise will be DEMO only...



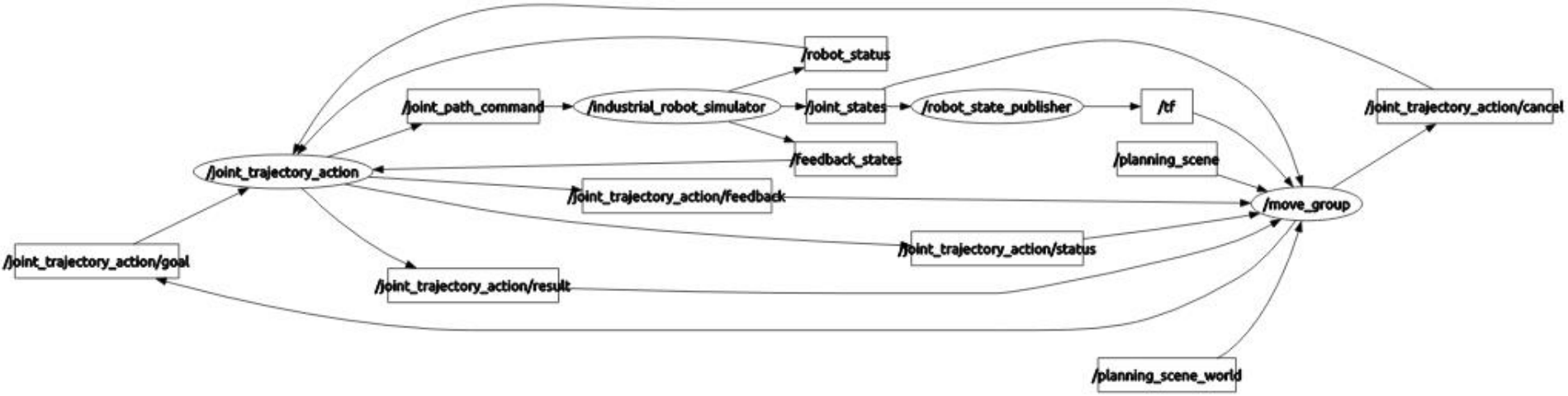
Message vs. Service vs. Action

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

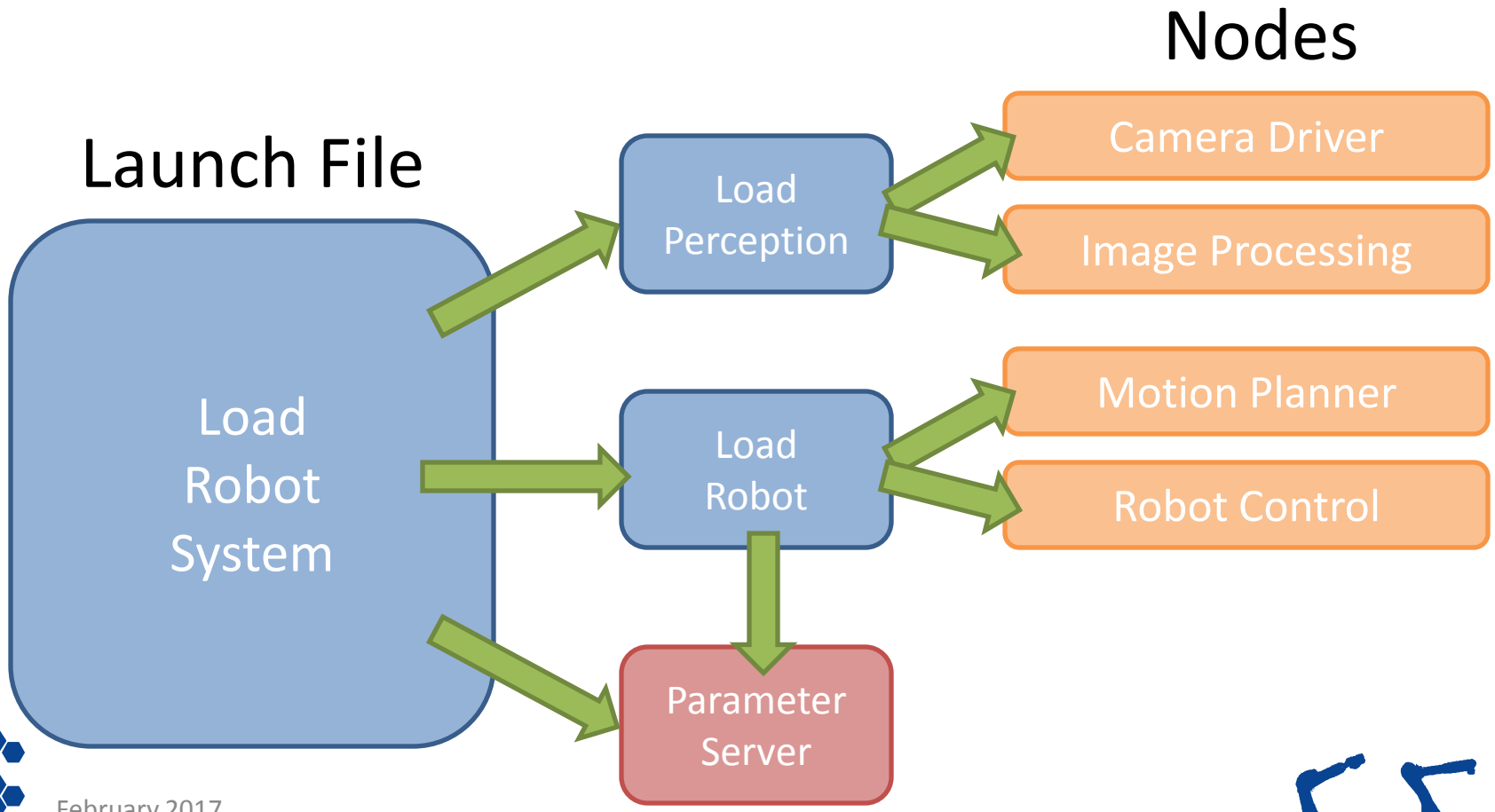
Launch Files: Motivation

- 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>
```

Roslaunch Practical Examples

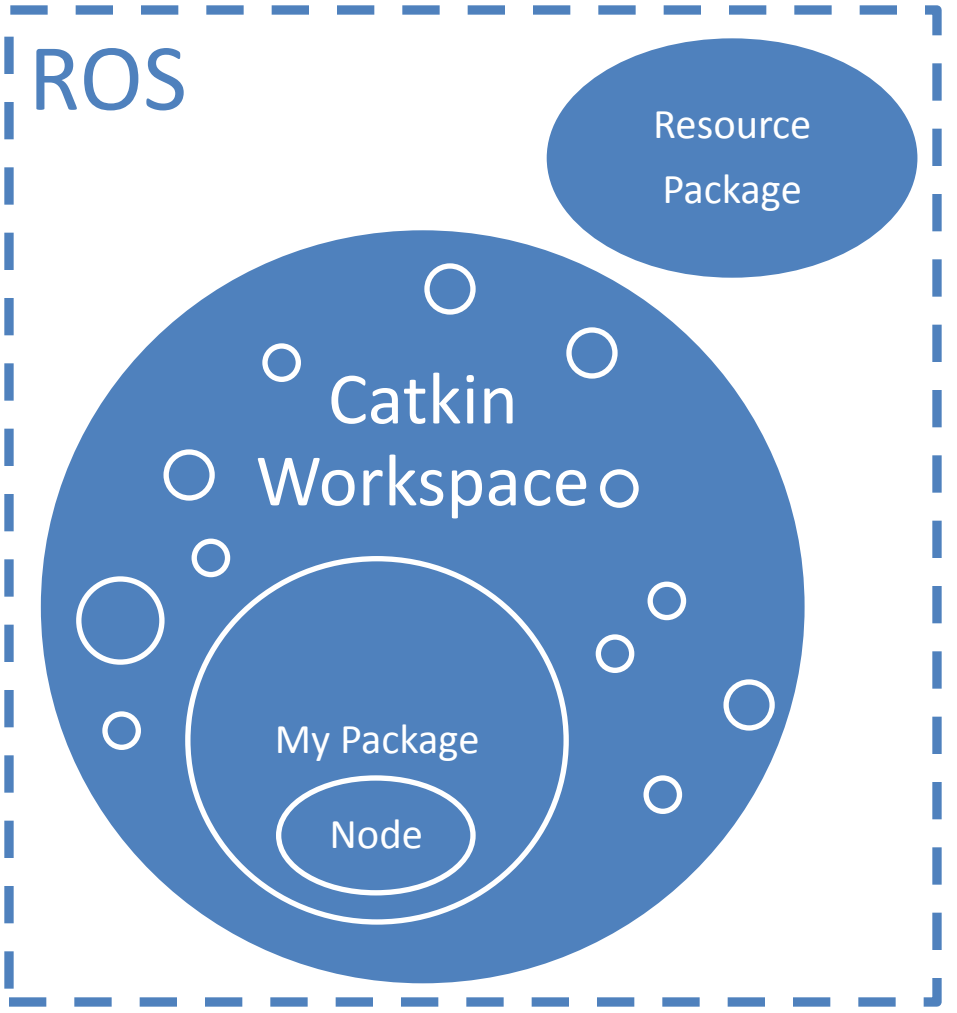
- We bring up the UR5 with a launch file that:
  - Takes arguments
  - Launches other, shared launch files
  - One of the other launch files brings up a robot driver node

Exercise 2.2

Create a launch file to launch fake_publisher, vision_node and myworkcell_core

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

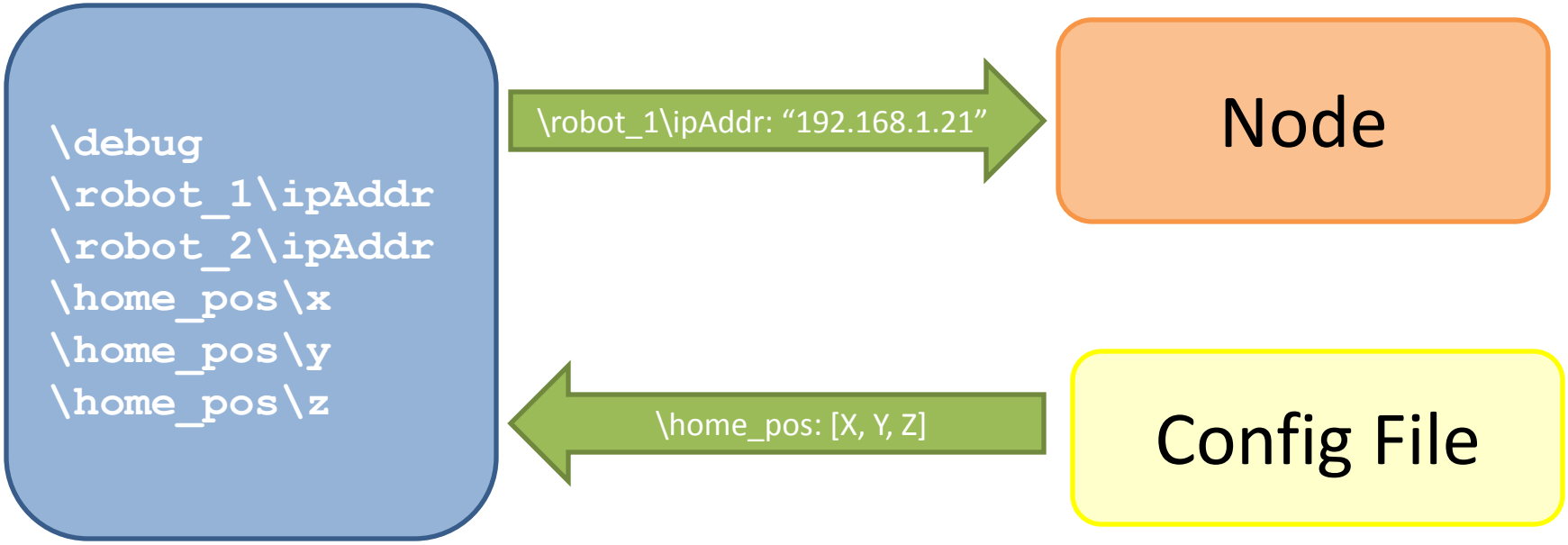


Parameters

Parameters: Overview

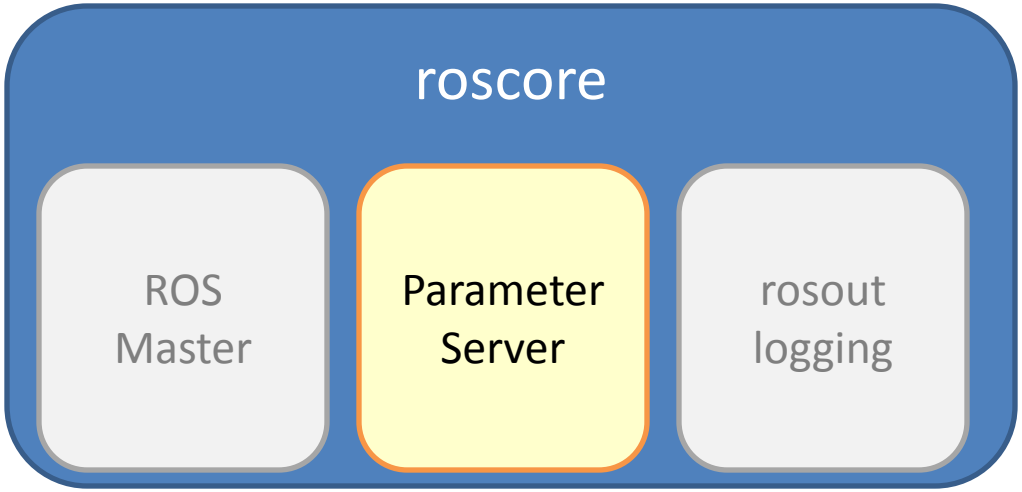
Parameters are like **Global Data**

Parameter Server



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

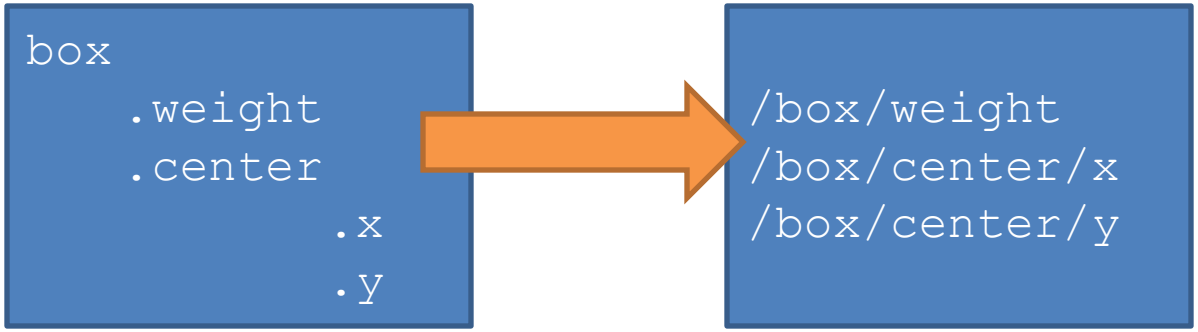
```
rosrun my_pkg load_robot _ip:="192.168.1.21"
rosparam set "/debug" true
```

  - Programs

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

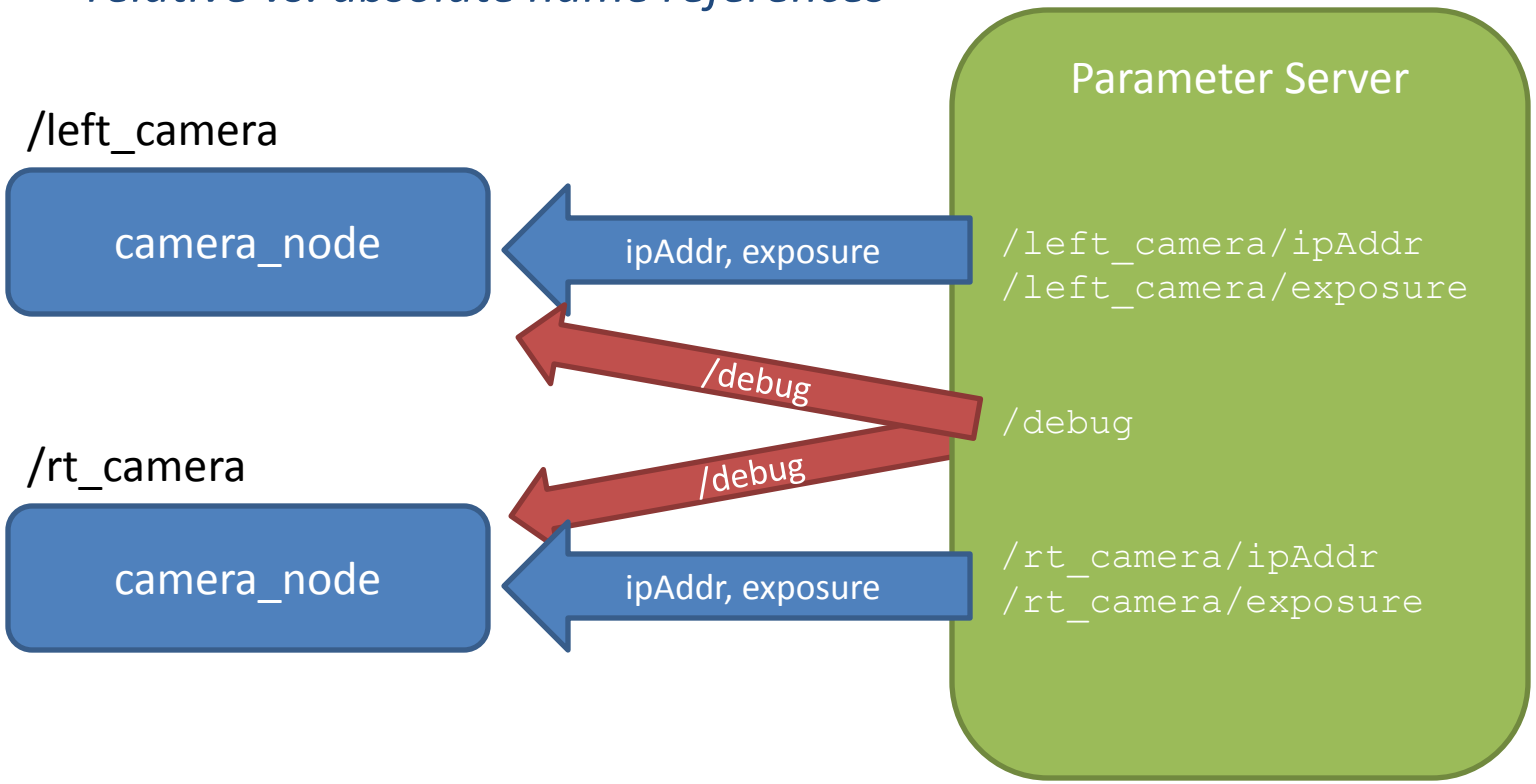
Parameter Datatypes

- 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)`

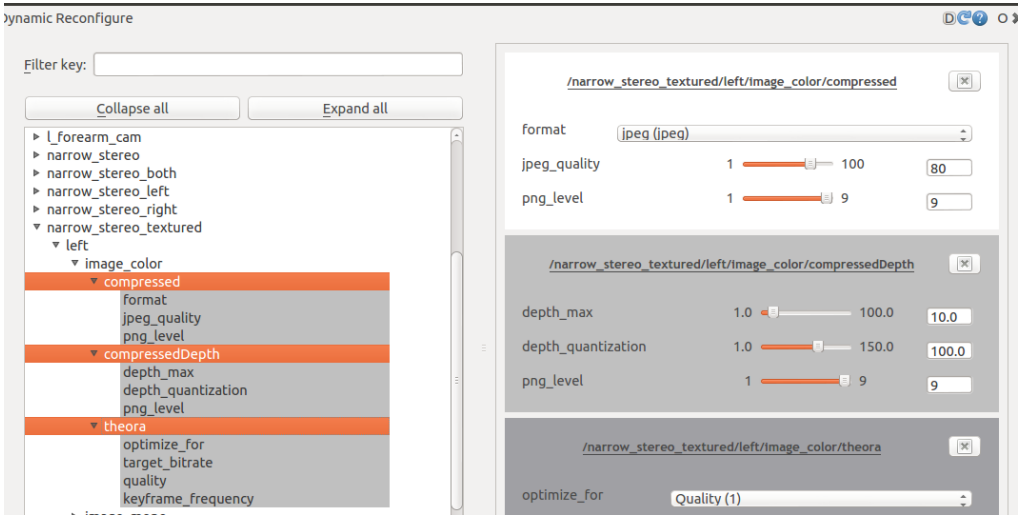
    Get value (or default, if doesn't exist)
  - `nh.setParam(key, value)`

    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 params the UR5 driver uses:
  - Prefix
  - robot_ip_address
  - max_velocity
  - servoj_time
  - Etc…

Exercise 2.3

ROS Parameters

watch as I walk through this example...

```
/global_integer_value
/rosdistro
/roslaunch/uris/host_ros_industrial_
/rosversion
/run_id
/simple_parameters/integer_value
/simple_parameters/point/x
/simple_parameters/point/y
/simple_parameters/simple_string
```

Review/Q&A

Session 1

Intro to ROS

Installing ROS/Packages

Packages

Nodes

Messages/Topics

Session 2

Services

Actions

Launch Files

Parameters

Contact Info.

Jeremy Zoss

SwRI
9503 W. Commerce
San Antonio, TX 78227
USA

Phone: 210-522-3089
Email: jzoss@swri.org

Levi Armstrong

SwRI
9503 W. Commerce
San Antonio, TX 78227
USA

Phone: 210-522-3801
Email: levi.armstrong@swri.org