ROS-Industrial Basic Developer's Training Class

April 2015

Southwest Research Institute

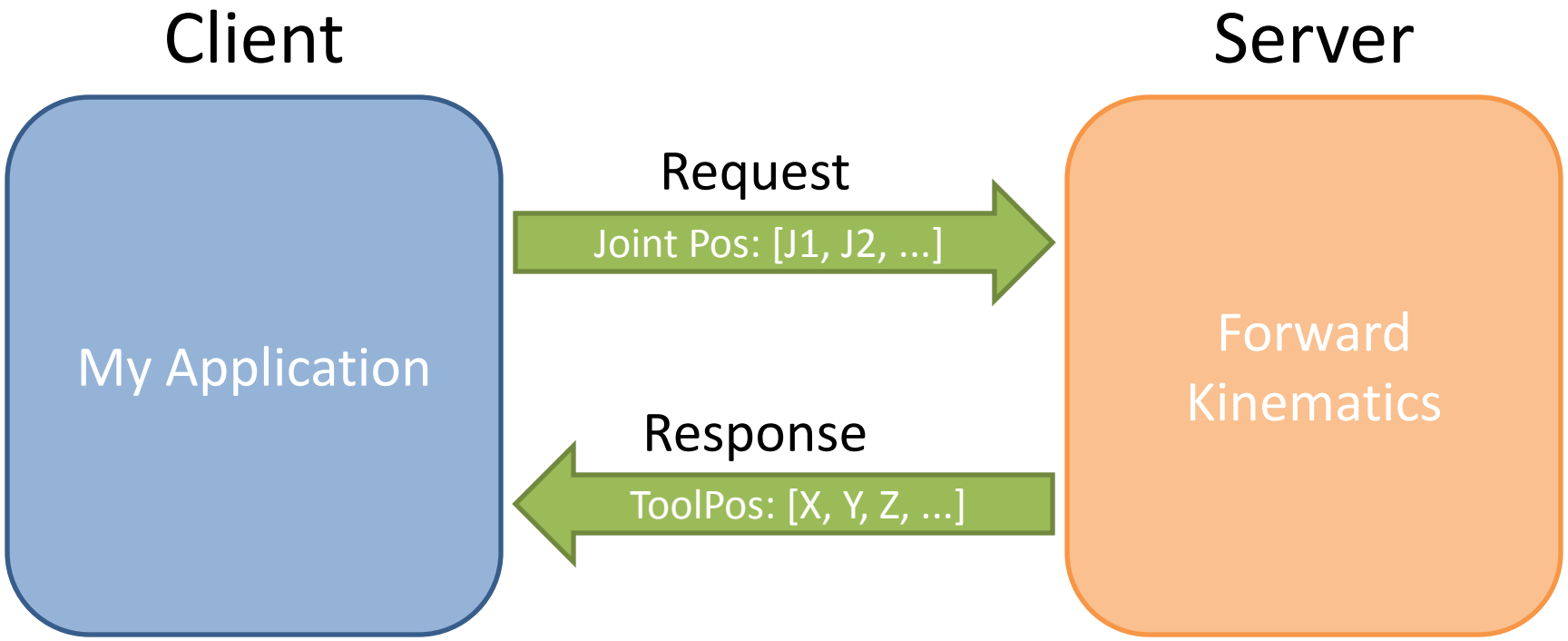
Outline

- Services
- Actions
- Launch Files
- TF
- URDF

Services

Services : Overview

Services are like **Function Calls**



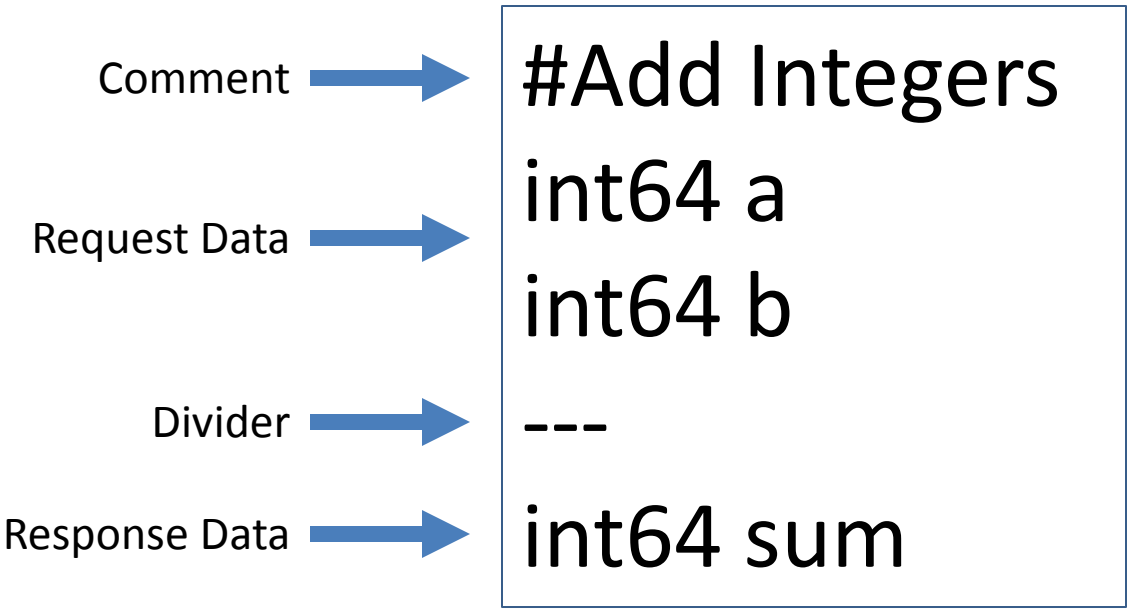
Services: Details

- 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

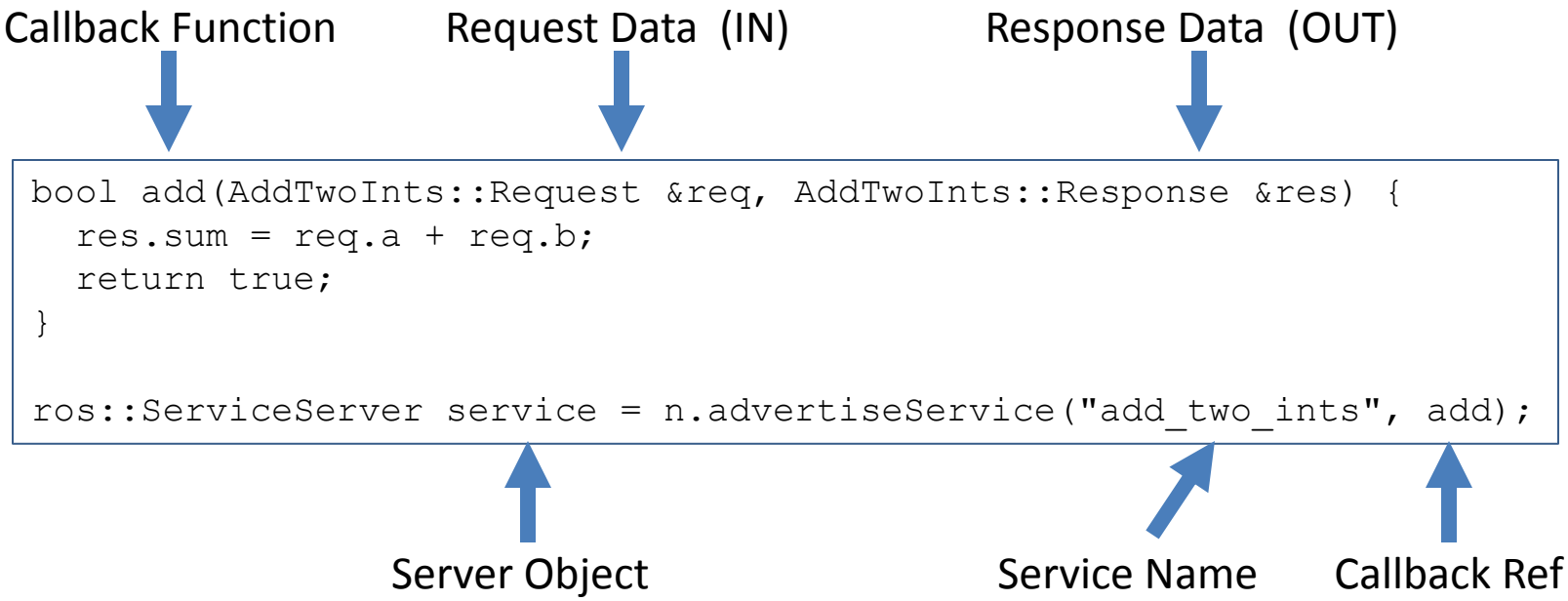
- Service **definition**
  - Defines Request and Response **data types**
    - *Either/both data type(s) may be **empty.** Always receive "completed" handshake.*
  - Auto-generates C++ Class files (.h/.cpp), Python, etc.

AddTwoInts.srv



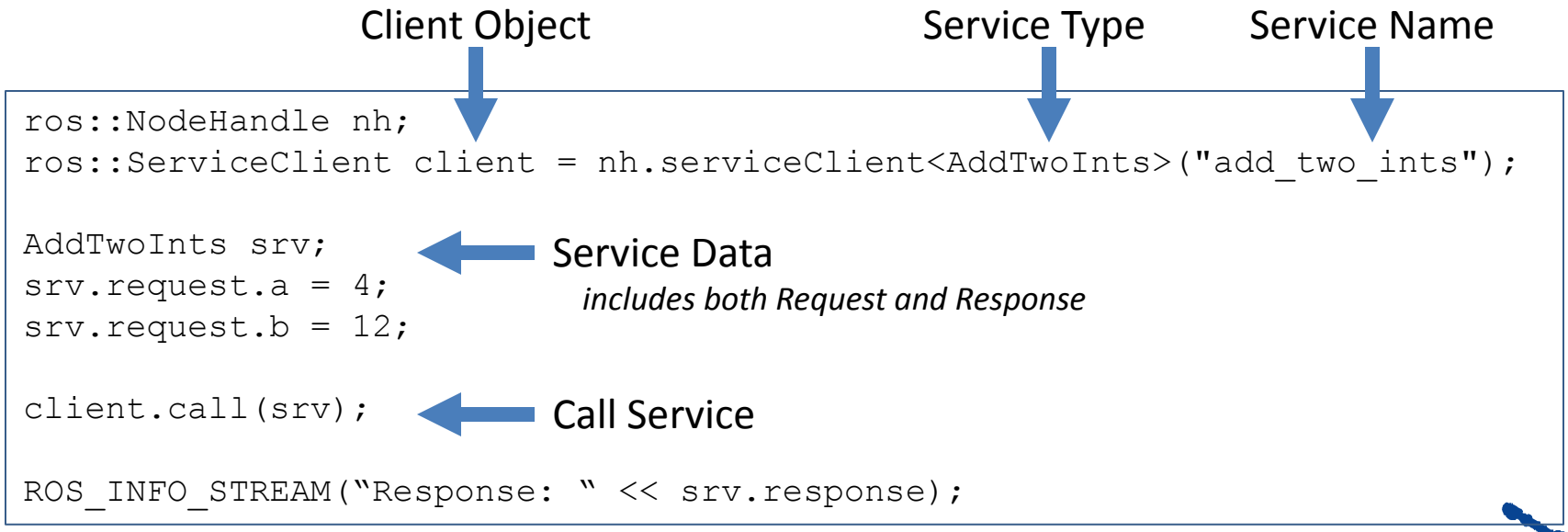
Services: Syntax

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



Services: Syntax

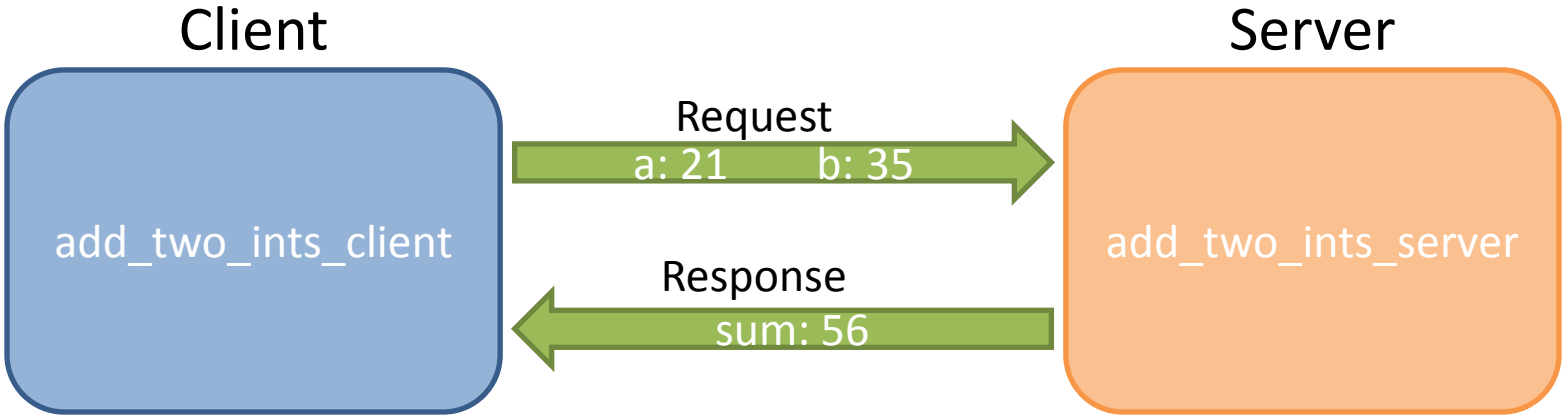
- Service **Client**
  - Connects to specific Service *(Name / Data Type)*
  - Fills in Request data
  - Calls Service



Exercise 2.1

Creating and Using a Service

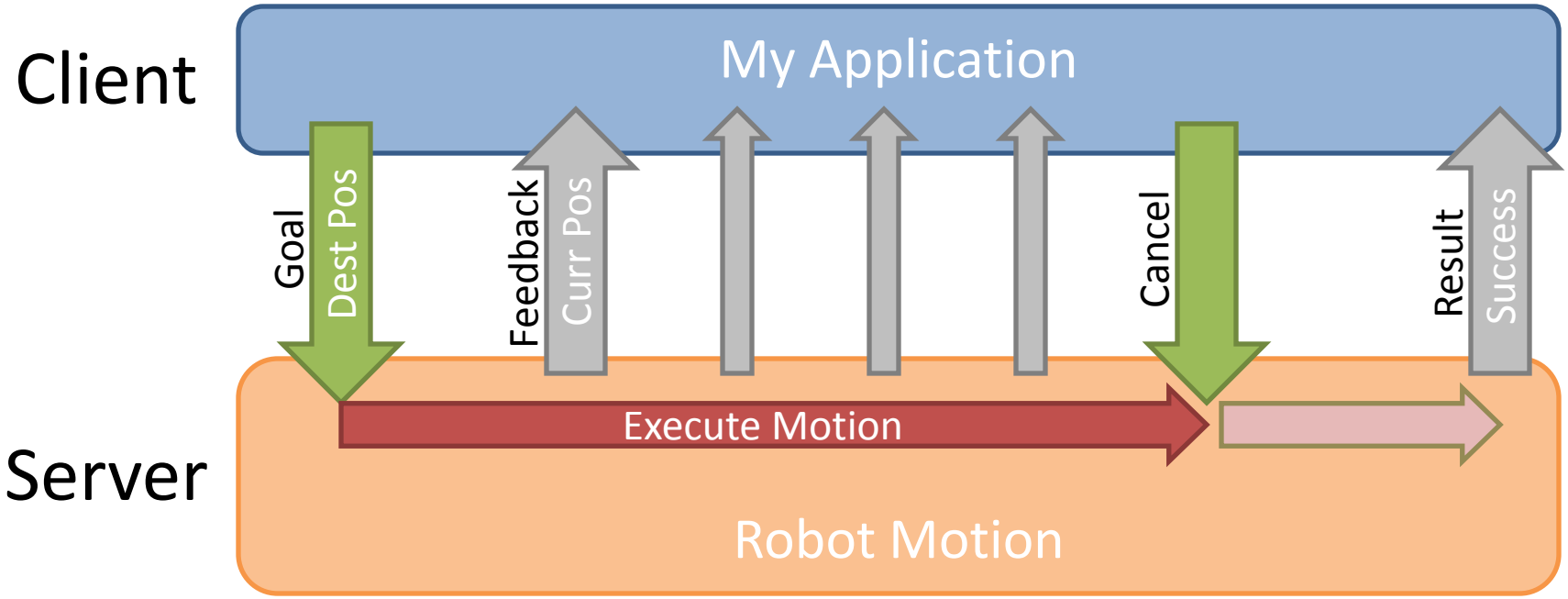
Let's work through Exercise 2.1 together



Actions

Actions : Overview

Actions manage **Long-Running Tasks**

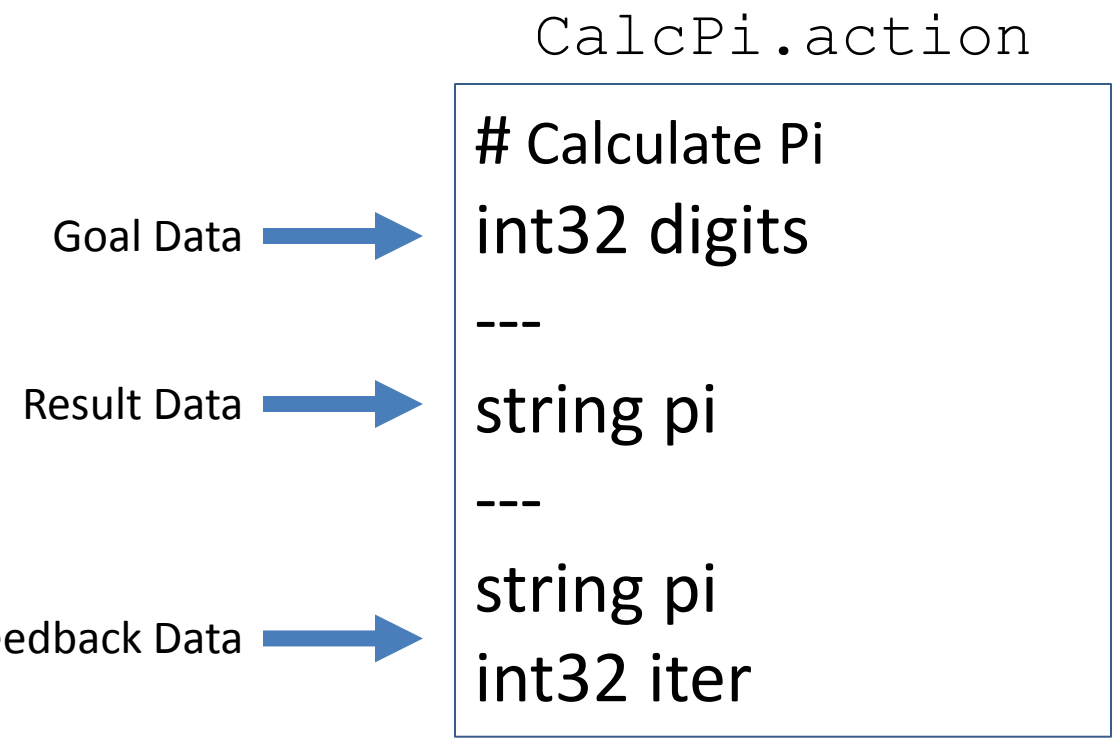


Actions: Detail

- 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

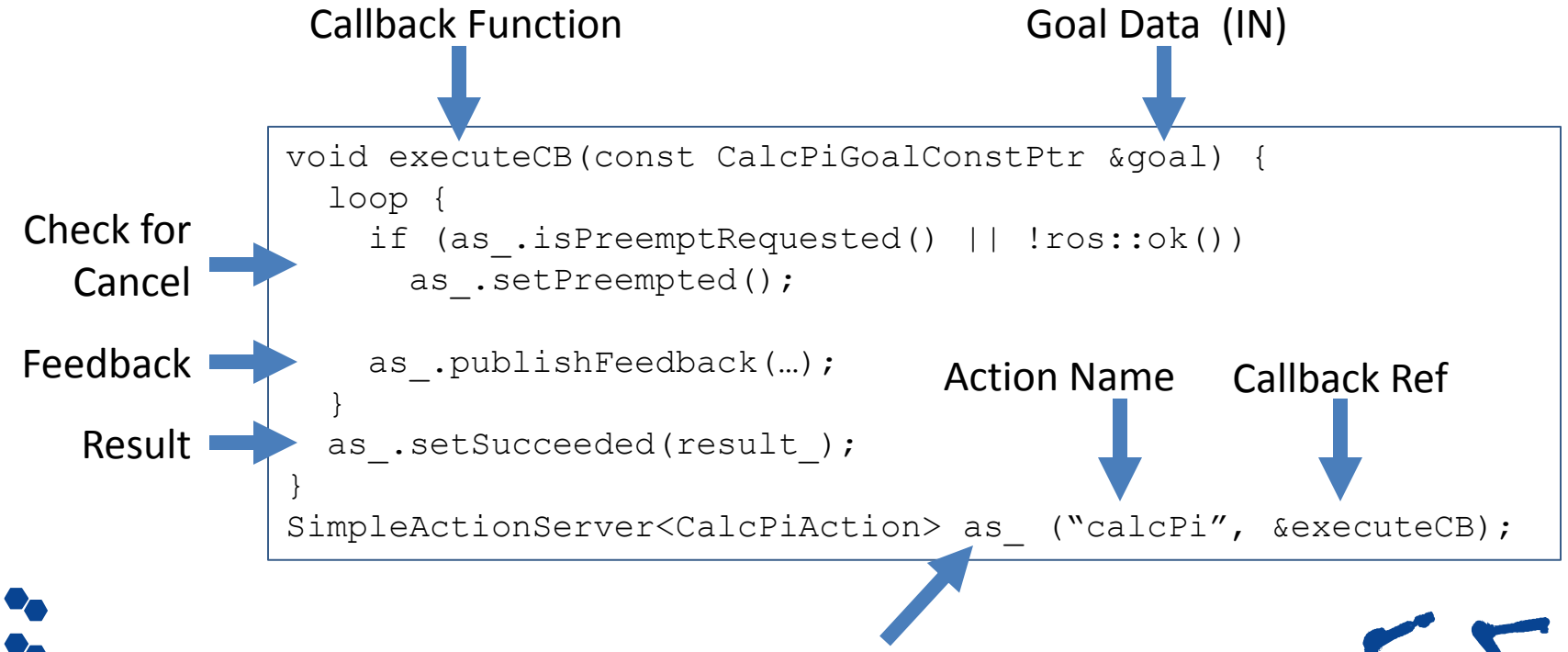
Actions: Syntax

- 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

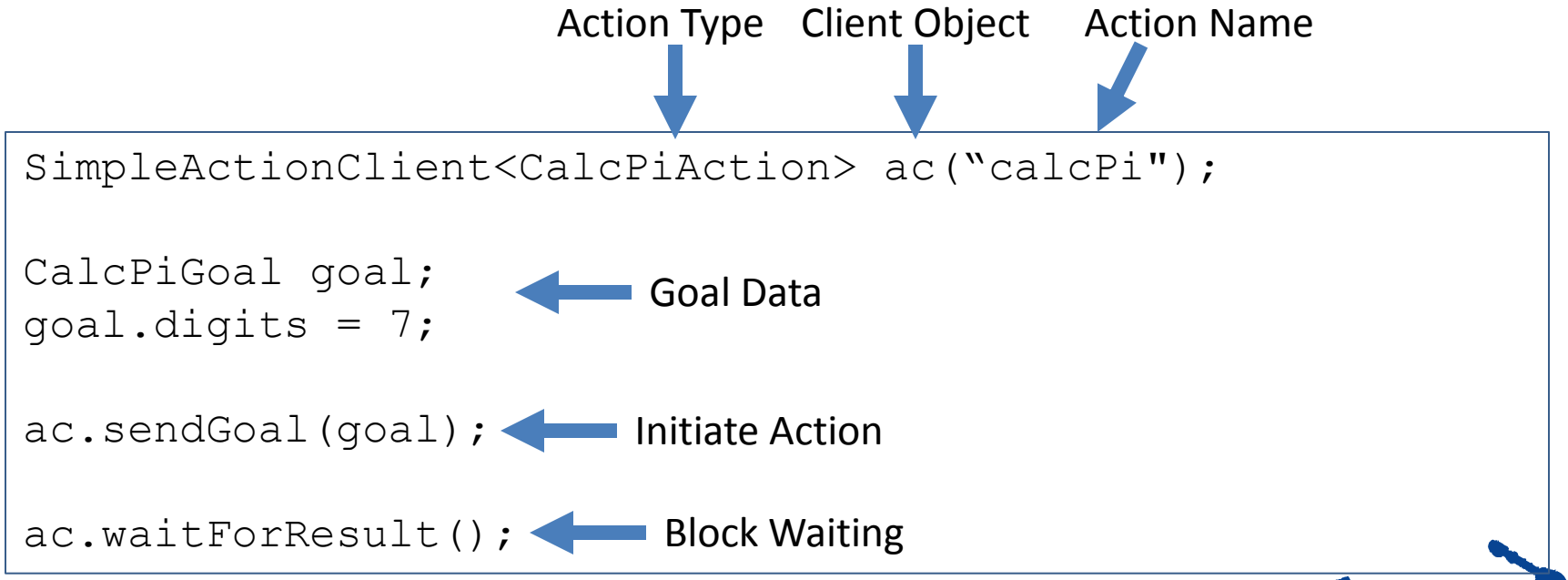
- Action **Server**
  - Defines **Execute Callback**
  - Periodically **Publish Feedback**
  - Advertises available action *(Name, Data Type)*




```
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

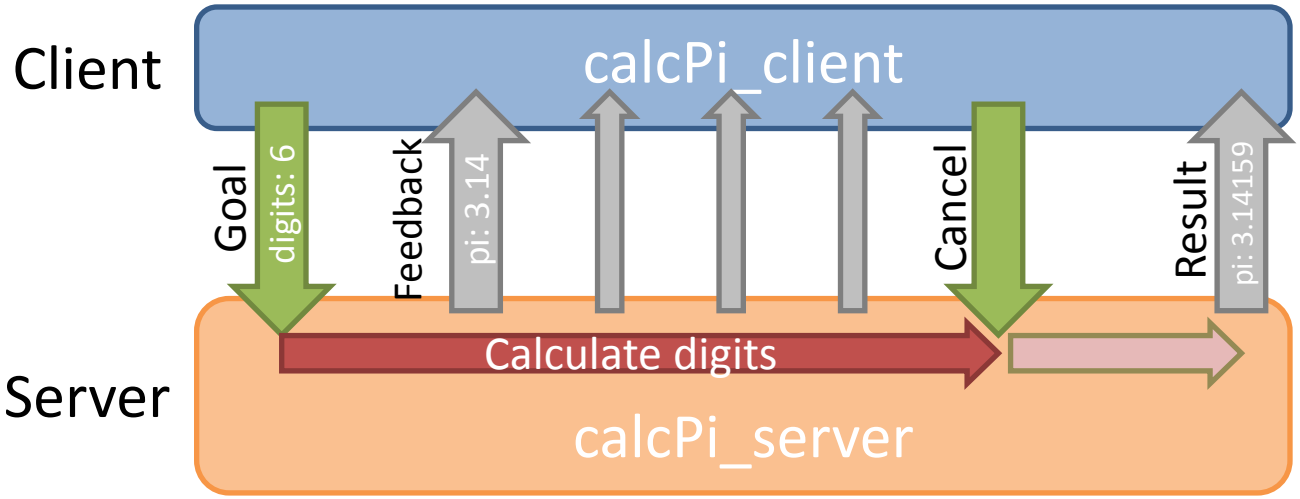
- Action **Client**
  - Connects to specific Action *(Name / Data Type)*
  - Fills in Goal data
  - Initiate Action / Waits for Result



Exercise 2.2

Creating and Using an Action

This Exercise will be DEMO only...



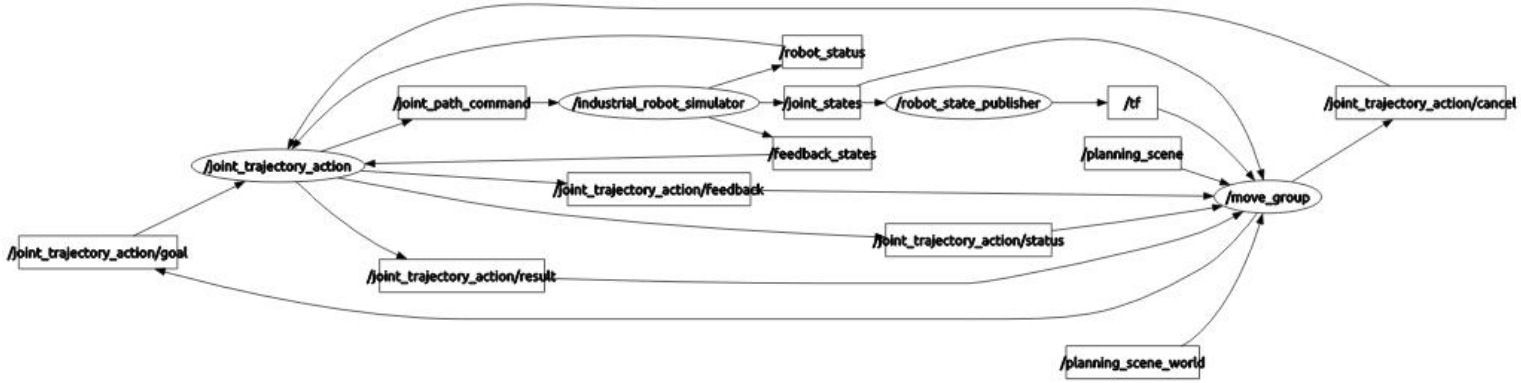
Message vs. Service vs. Action

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

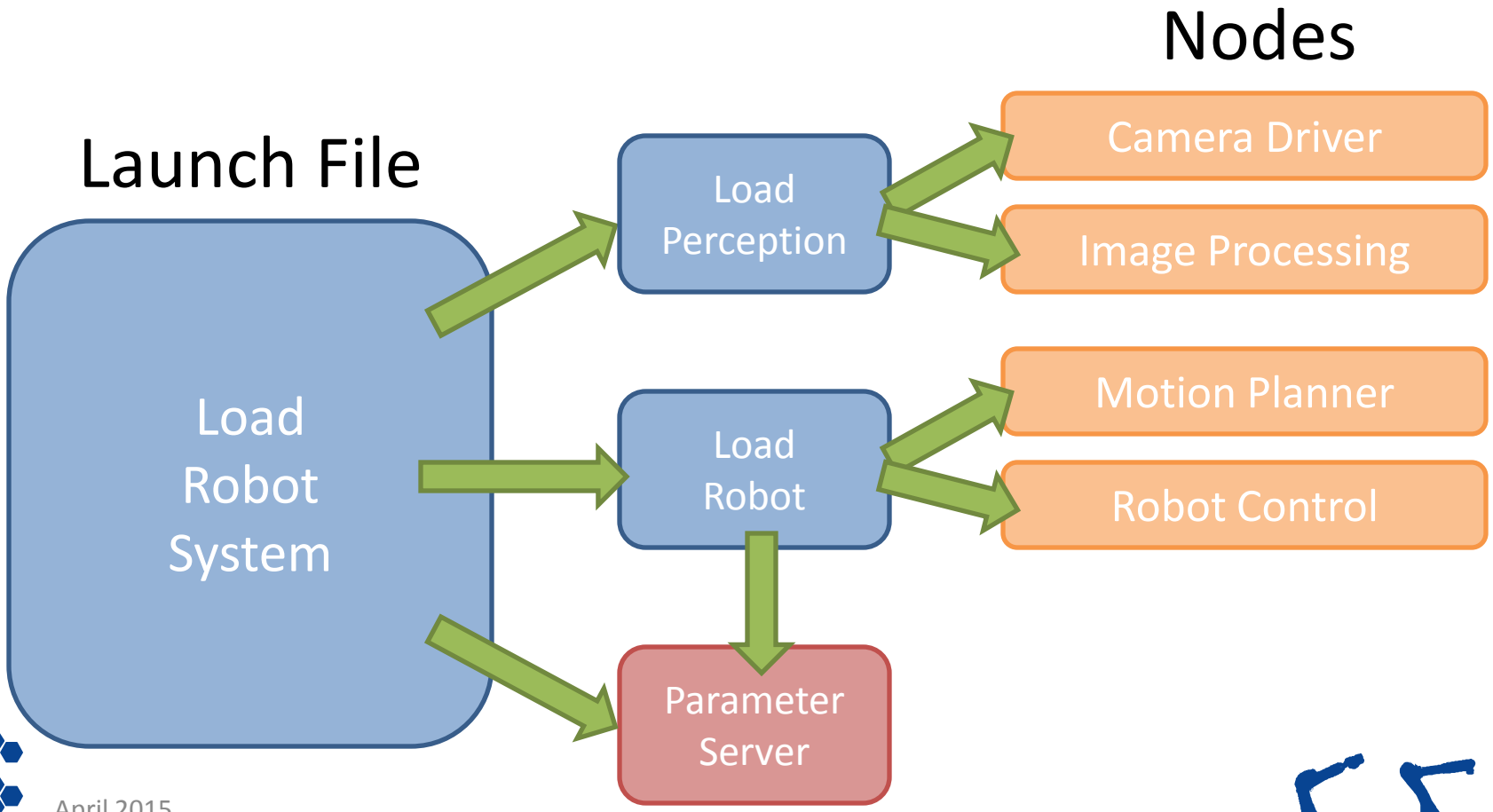
Launch Files: Motivation

- 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>
```

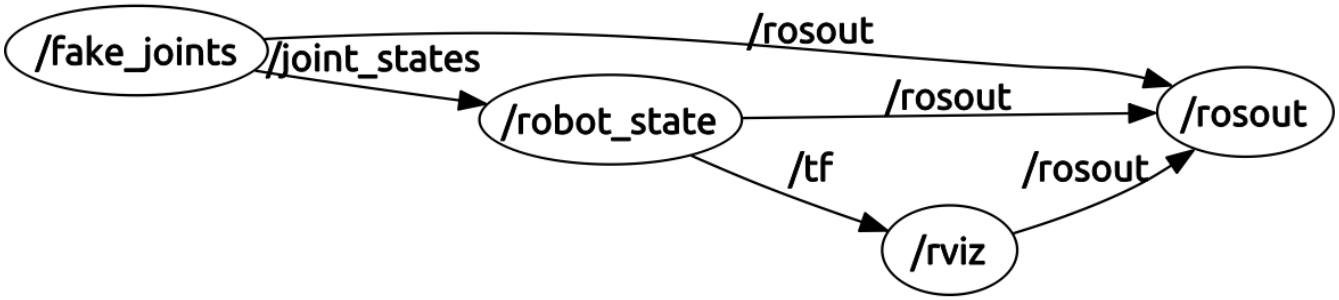
Launch Files: Syntax (Adv.)

- **<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>
```

Exercise 2.3

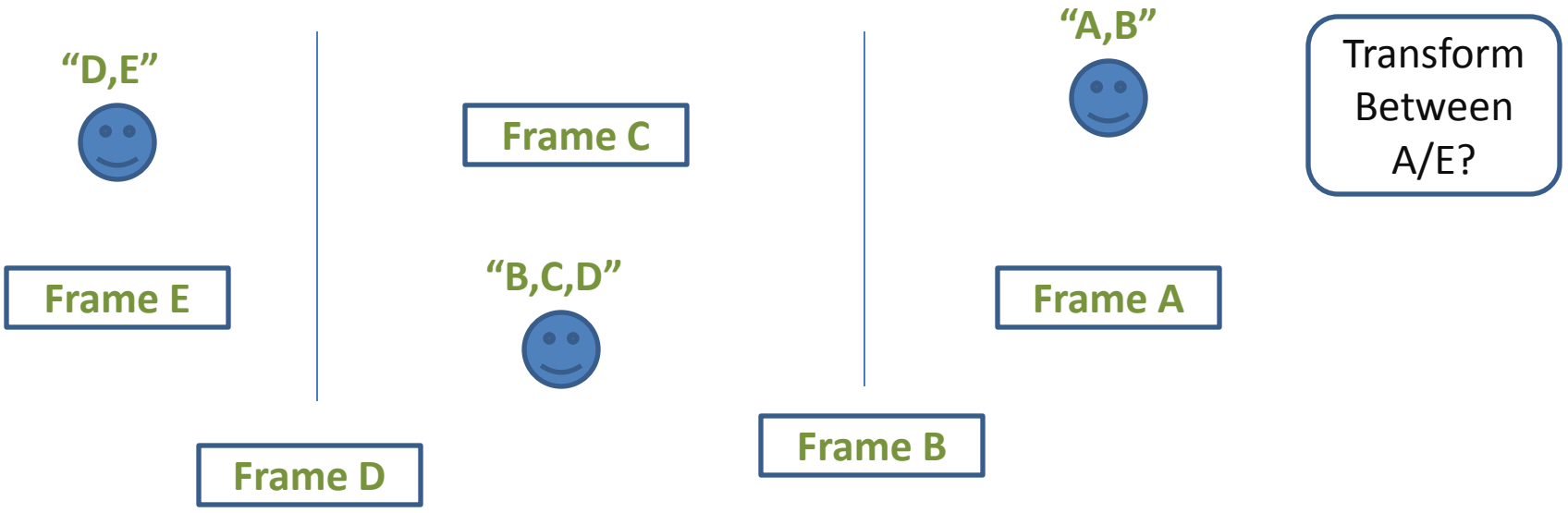
Introduction to Launch Files



TF – Transforms in ROS

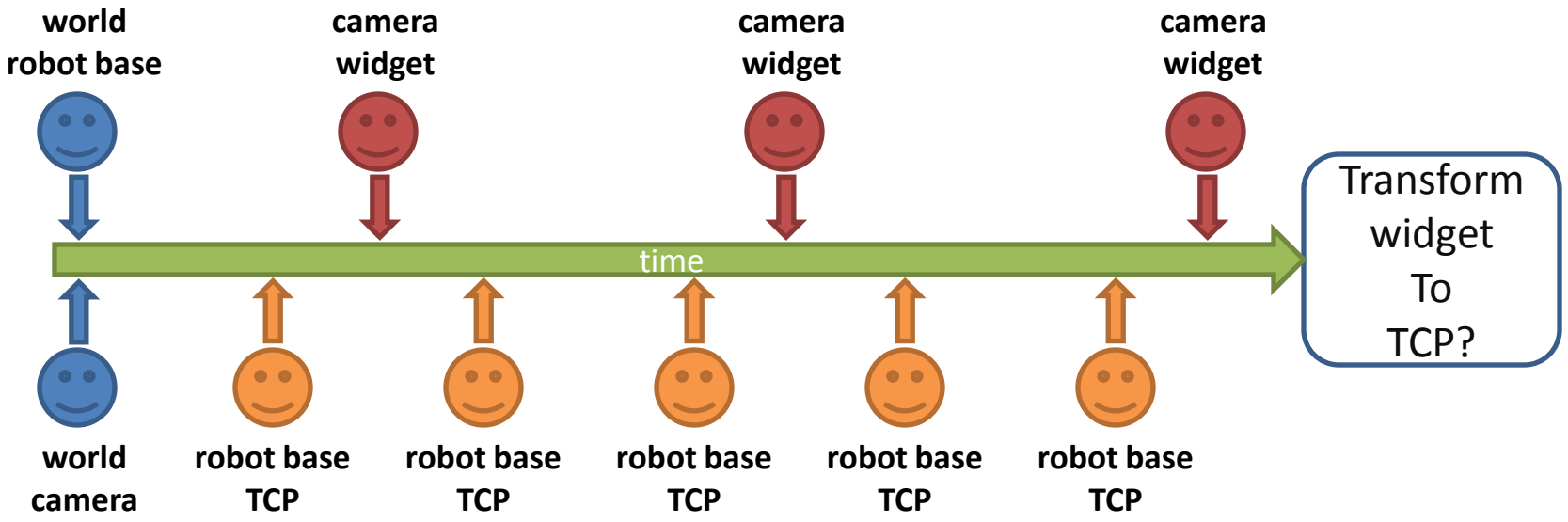
TF: Overview

- TF is a **distributed framework**
  to track **coordinate frames**
- Each frame is related to at least one other frame



TF: Time Sync

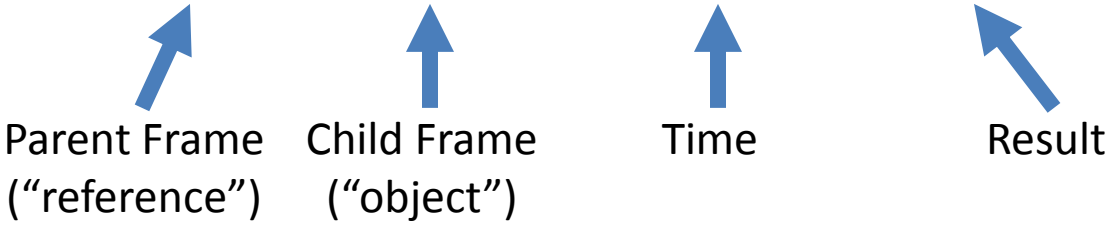
- TF tracks frame history
  - can be used to find transforms in the past!
  - essential for asynchronous / distributed system



TF: c++

- Each **node** has its own `transformListener`
  - listens to all tf messages, calculates relative transforms
  - Can try to transform in the past
  - ➢ Can only look as far back as it has been running

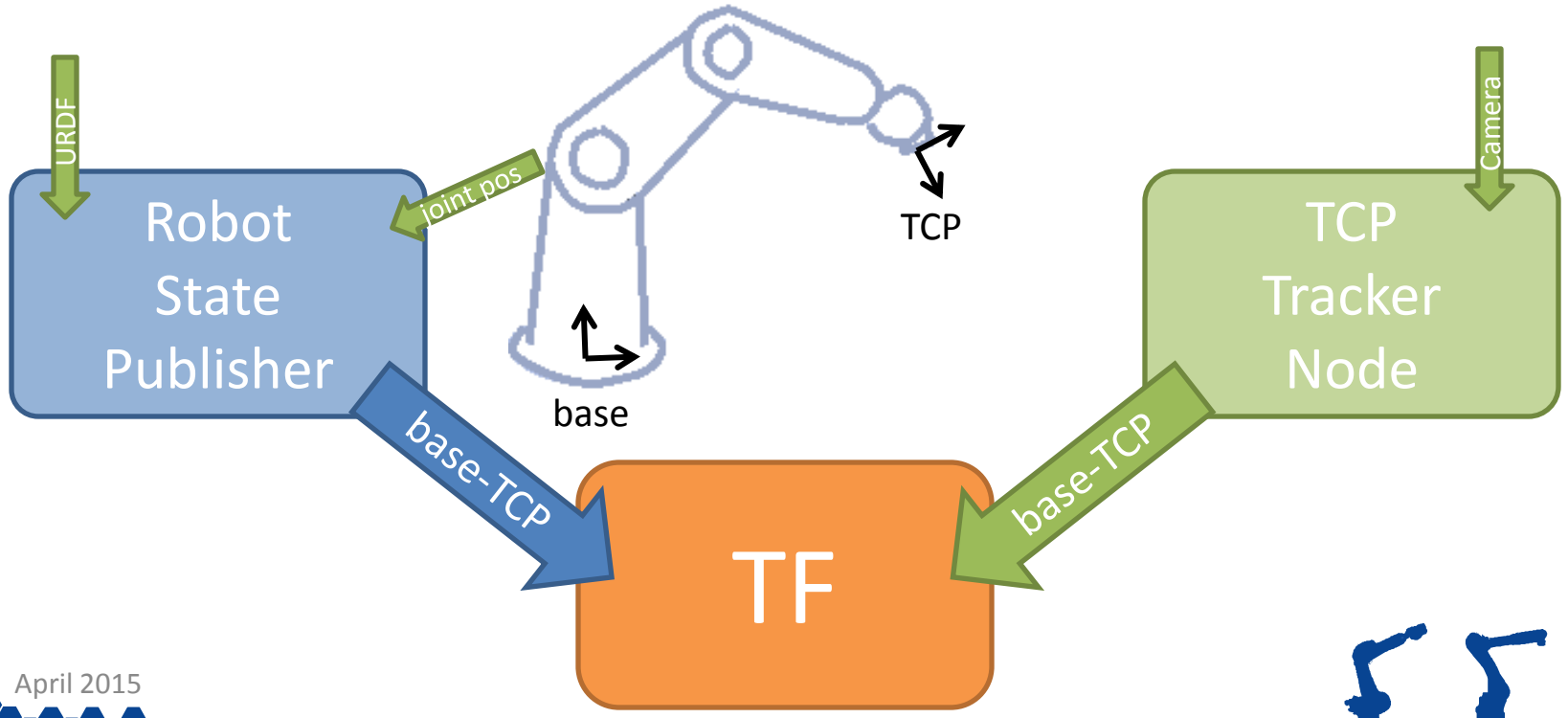
```
tf::TransformListener listener;
tf::StampedTransform transform;

listener.lookupTransform("target", "source", ros::Time(), transform);
```



- Note confusing "target/source" naming convention
- ros::Time() or ros::Time(0) give **latest** available transform
- ros::Time::now() usually fails

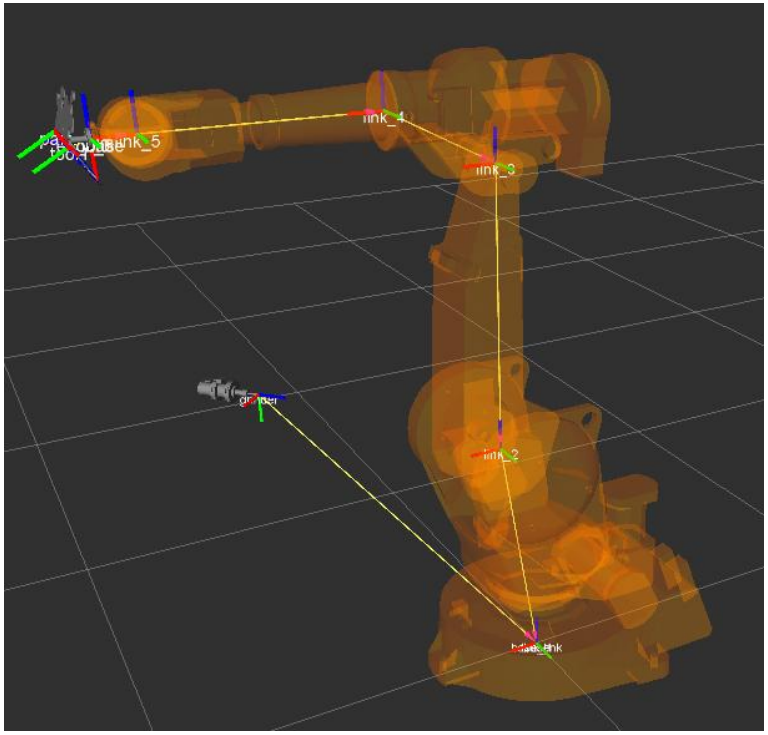
TF: Sources

- A `robot_state_publisher` provides TF data from a **URDF**
- Nodes can also publish TF data
  - DANGER! TF data <u>can</u> be conflicting



Exercise 2.4

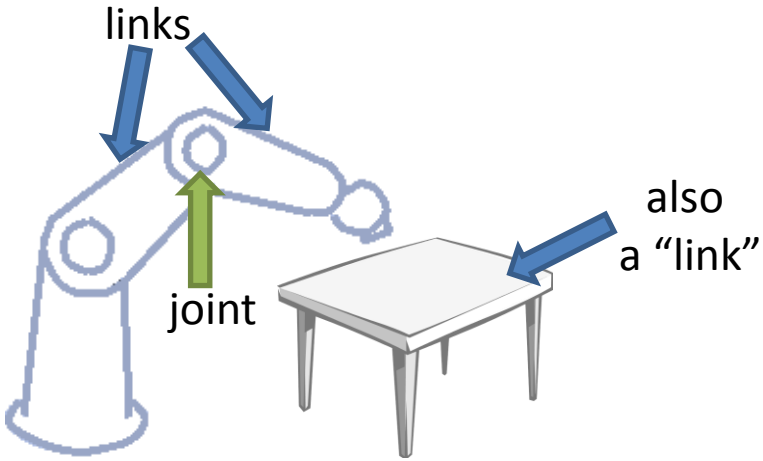
Introduction to TF



URDF: Unified Robot Description Format

URDF: Overview

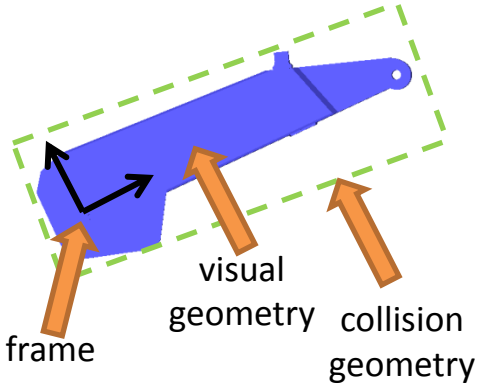
- URDF is an **XML**-formatted file containing:
  - **Links** : coordinate frames and associated geometry
  - **Joints :** connections between links
- Similar to DH-parameters (but way less painful)
- Can describe entire workspace, not just robots



URDF: Link

- A **Link** describes a physical or virtual object
  - Physical : robot link, workpiece, end-effector, ...
  - Virtual : TCP, robot base frame, ...
- Each link becomes a TF frame
- Can contain visual/collision geometry [optional]

```
<link name="link_4">
    <visual>
        <geometry>
            <mesh filename="link_4.stl"/>
        </geometry>
        <origin xyz="0 0 0" rpy="0 0 0" />
    </visual>
    <collision>
        <geometry>
            <cylinder length="0.5" radius="0.1"/>
        </geometry>
        <origin xyz="0 0 -0.05" rpy="0 0 0" />
    </collision>
</link>
```



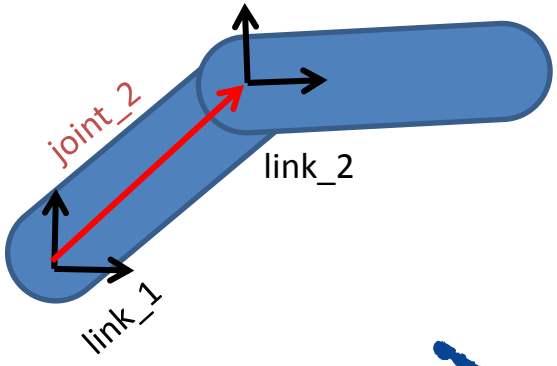
URDF Transforms

X/Y/Z Meters Roll/Pitch/Yaw Radians

URDF: Joint

- A **Joint** connects 2 **Links**
  - Defines a **transform** between **parent** and **child** frames
    - Types: *fixed, free, linear, rotary*
  - Denotes axis of movement *(for linear / rotary)*
  - Contains joint limits on position and velocity
- ROS-I conventions
  - X-axis front, Z-Axis up
  - Keep all frames similarly rotated when possible

```
<joint name="joint_2" type="revolute">
   <parent link="link_1"/>
   <child link="link_2"/>
   <origin xyz="0.2 0.2 0" rpy="0 0 0"/>
   <axis xyz="0 0 1"/>
   <limit lower="-3.14" upper="3.14" velocity="1.0"/>
</joint>
```



URDF: XACRO

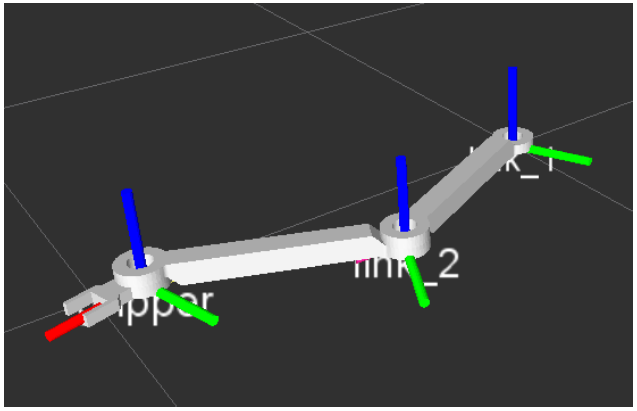
- **XACRO** is an XML-based "macro language" for building URDFs
  - <Include> other XACROs, with parameters
  - Simple expressions: math, subtitution
- Used to build complex URDFs
  - multi-robot workcells
  - reuse standard URDFs (e.g. robots, tooling)

```
<xacro:include filename="myRobot.xacro"/>

<xacro:myRobot prefix="left_"/>
<xacro:myRobot prefix="right_"/>

<property name="offset" value="1.3"/>

<joint name="world_to_left" type="fixed">
   <parent link="world"/>
   <child link="left_base_link"/>
   <origin xyz="${offset/2} 0 0" rpy="0 0 0"/>
</joint>
```

Exercise 2.5 / 2.5b

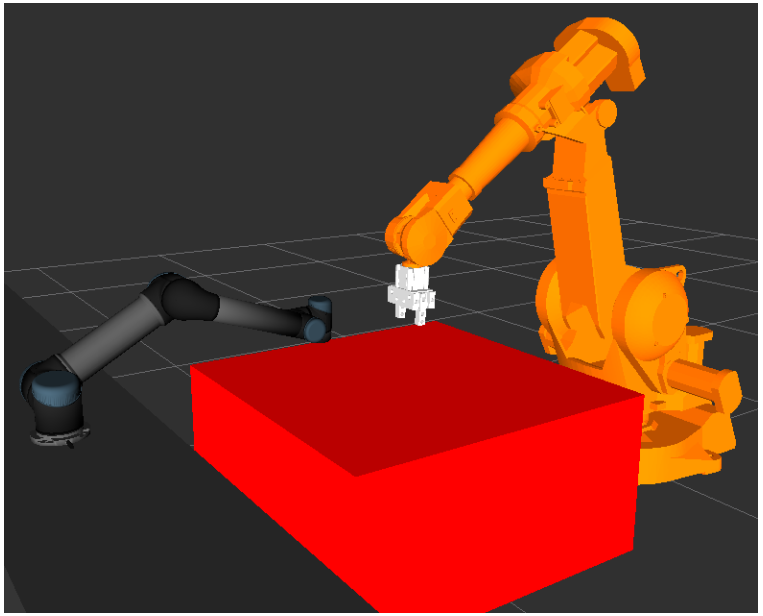
Exercise 2.5

Introduction to URDF



Exercise 2.5b

Workcell XACRO



Contact Info.

Jorge Nicho

Research Engineer

Southwest Research Institute
9503 W. Commerce
San Antonio, TX 78227
USA

Phone: 210-522-3107
Email: jorge.nicho@swri.org

www.ROSindustrial.org