ECE 478-578
Intelligent Robotics I

PhD. Husnu Melih Erdogan – Electrical & Computer Engineering

herdogan@pdx.edu Teaching Assistant

Portland State
UNIVERSITY

Introduction to ROS - Part 5 Speech Synthesis with Amazon Polly

ROS

Amazon Polly

Portland State UNIVERSITY

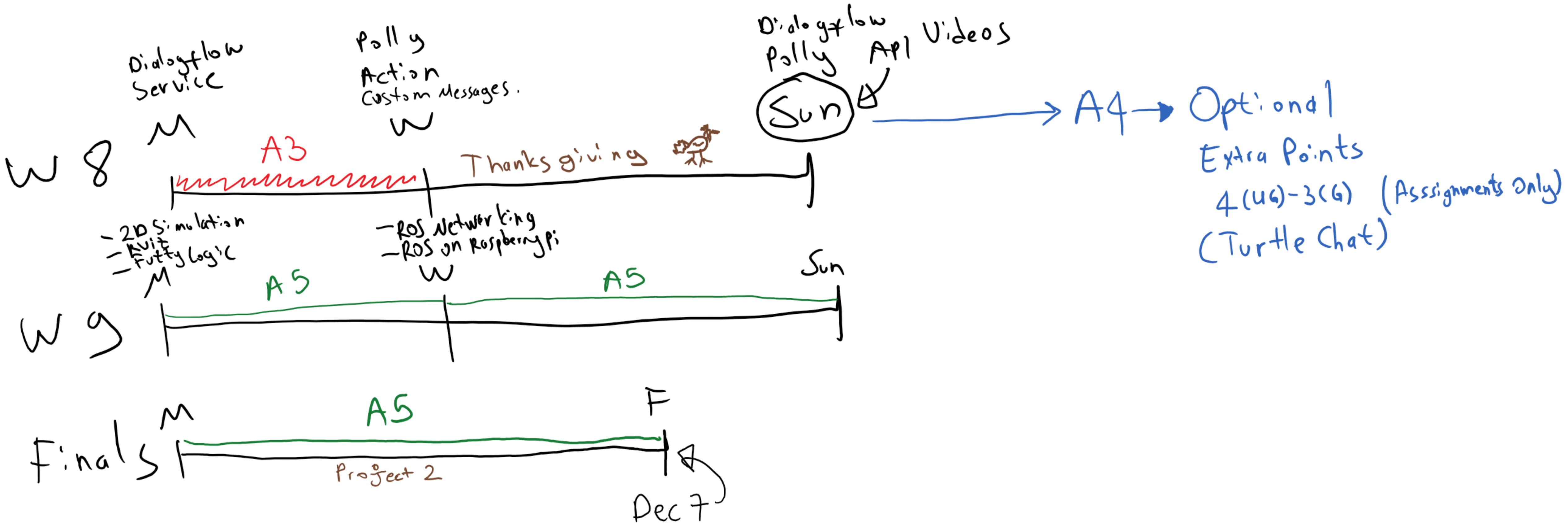
Course Structure

- Part 1 - Overview
  - What is ROS?
  - Introduction to ROS
  - ROS architecture, philosophy, history
  - How to install ROS?
  - Examples
  - Installation
  - ROS Master
  - ROS Nodes
  - ROS Topic
  - ROS Messages
  - Console Commands
  - ROS Packages
  - ROS Launch-files
  - Catkin Workspace and Build System
  - Turtlesim
- Part 2 - Basics
  - ROS File System
  - ROS Package
  - How to create a package?
  - How to build a package?
  - Creating a Publisher Node
  - Creating a Subscriber Node
  - Assignment 3
- Part 3 - Debug
  - ROS Launch File
  - How to use ROS .bagfiles?
  - ROS Parameters
  - ROS Namespace
- Part 4 - Speech
  - ROS Services
  - Speech Recognition
  - Speech Synthesis
  - Google Dialogflow
- Part 5 - Speech
  - Amazon Polly
  - ROS Actions
  - Assignment 4
- Part 6 - Fuzzy
  - Rviz
  - ROS Messages
  - 2D Multi-Robot Simulator
  - Fuzzy Logic
  - Assignment 5
- Part 7 - Network
  - ROS Networking
  - ROS and RaspberryPi

Portland State UNIVERSITY

HOUSTON
WE HAVE A PROBLEM...

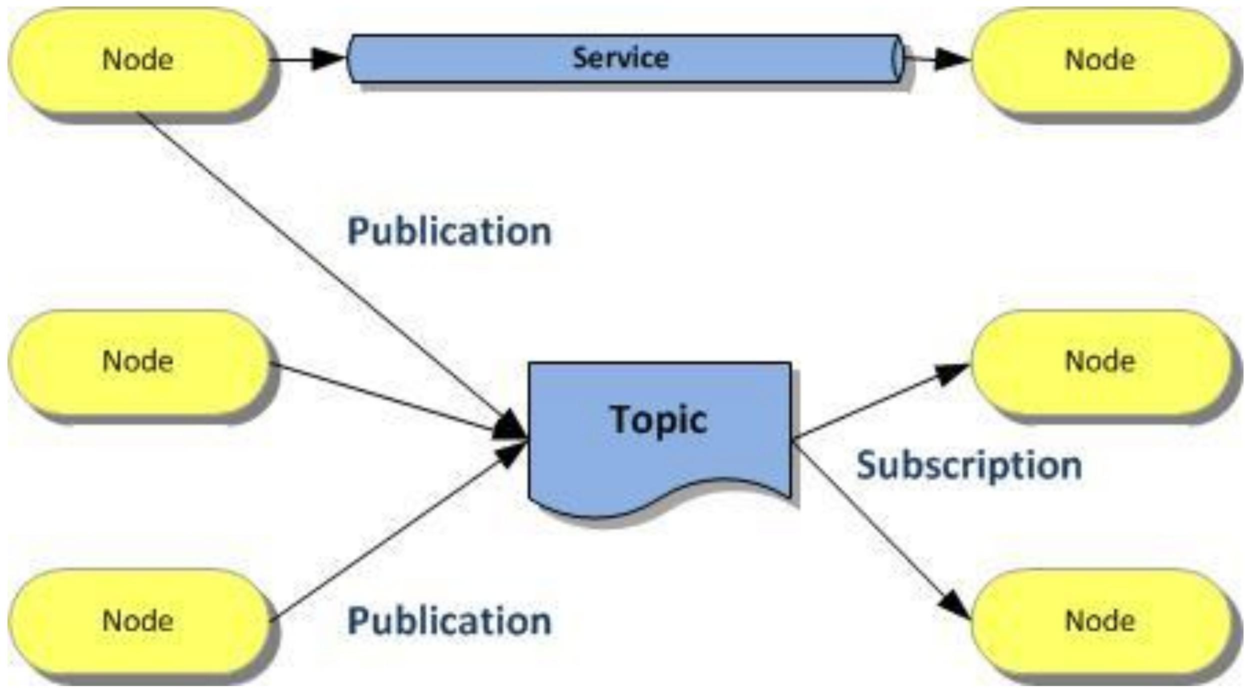
17 Days of Robotics



(Last Lecture Review)

ROS Service

Using ROS Services, we can write a server node and client node. The server node provides the service under a name, and when the client node sends a request message to this server, it will respond and send the result to the client.



Three Dialog Styles

Application Directed

Robot: What month?

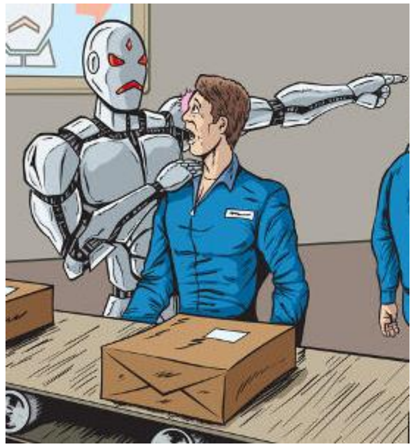
Human: February

Robot: What day of the week?

Human: Twelve

Robot: What year?

Human: Nineteen ninety-seven



Human Directed

Human: Set month to February

Robot: Month is February

Human: Set day to twelve

Robot: Day is twelve

Human: Set year to nineteen ninety-seven

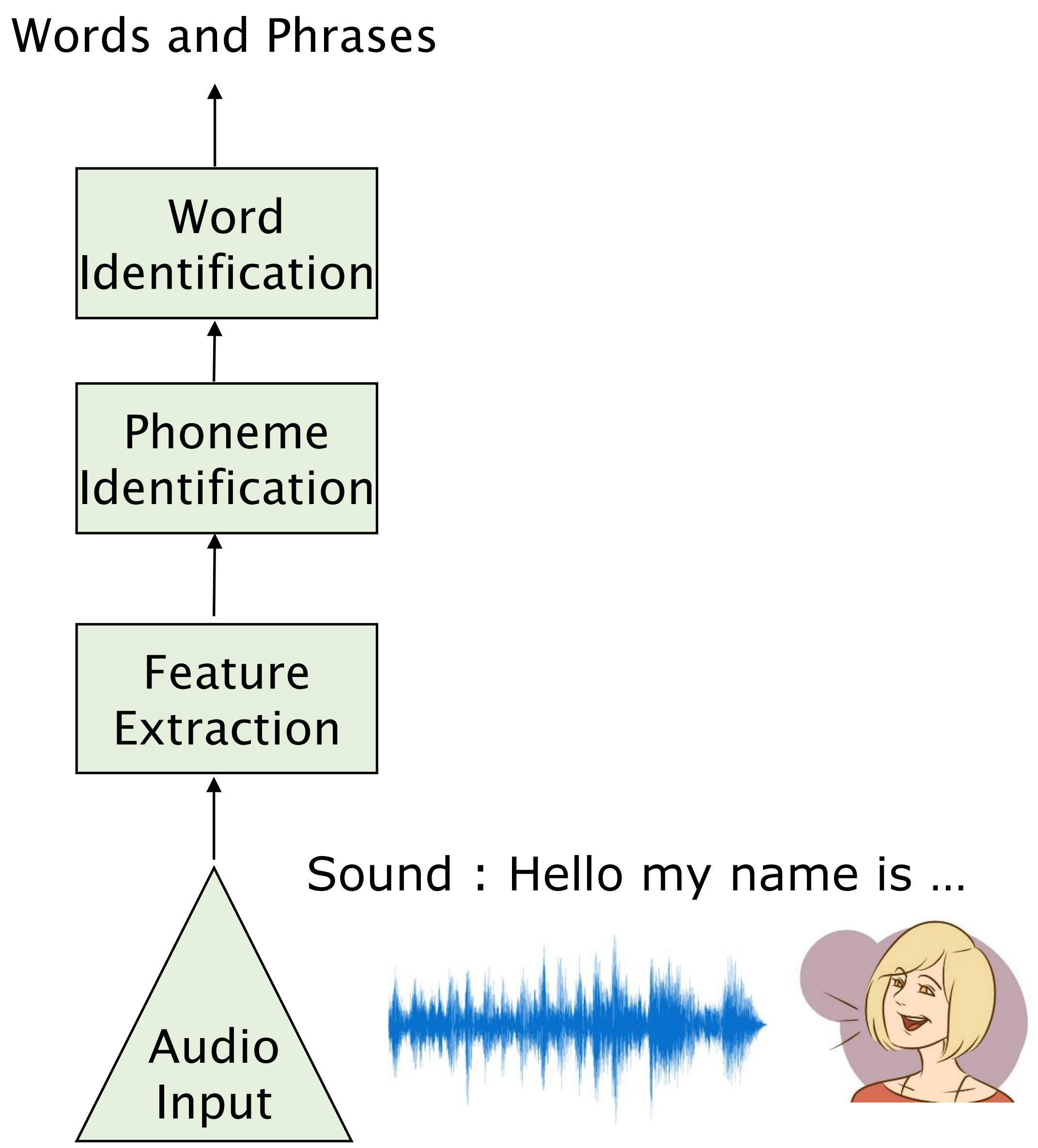
Robot: Year is nineteen ninety-seven

Mixed initiative

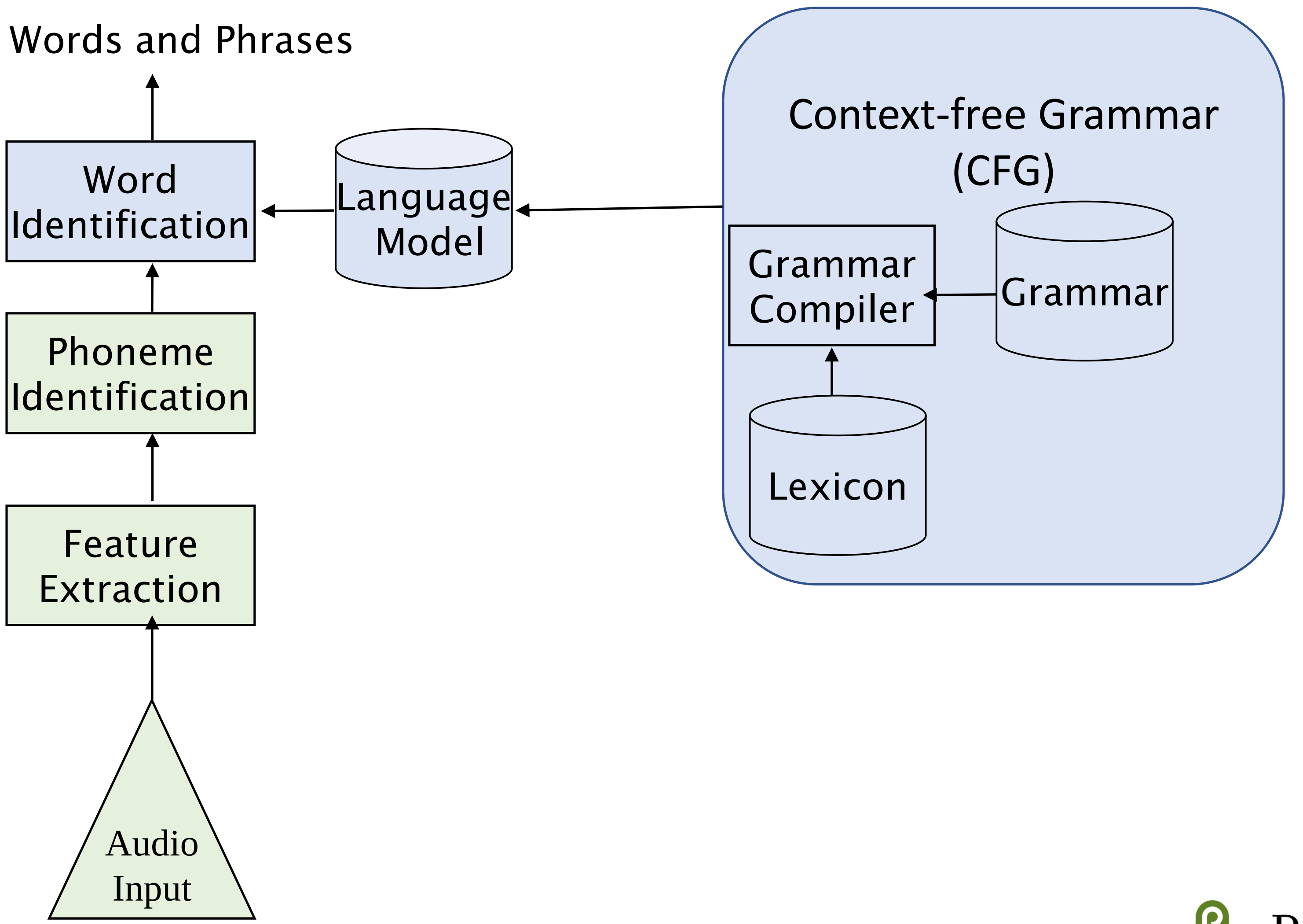
Robot: What month?

Human: February twelve nineteen ninety-seven

How Speech Recognition Works

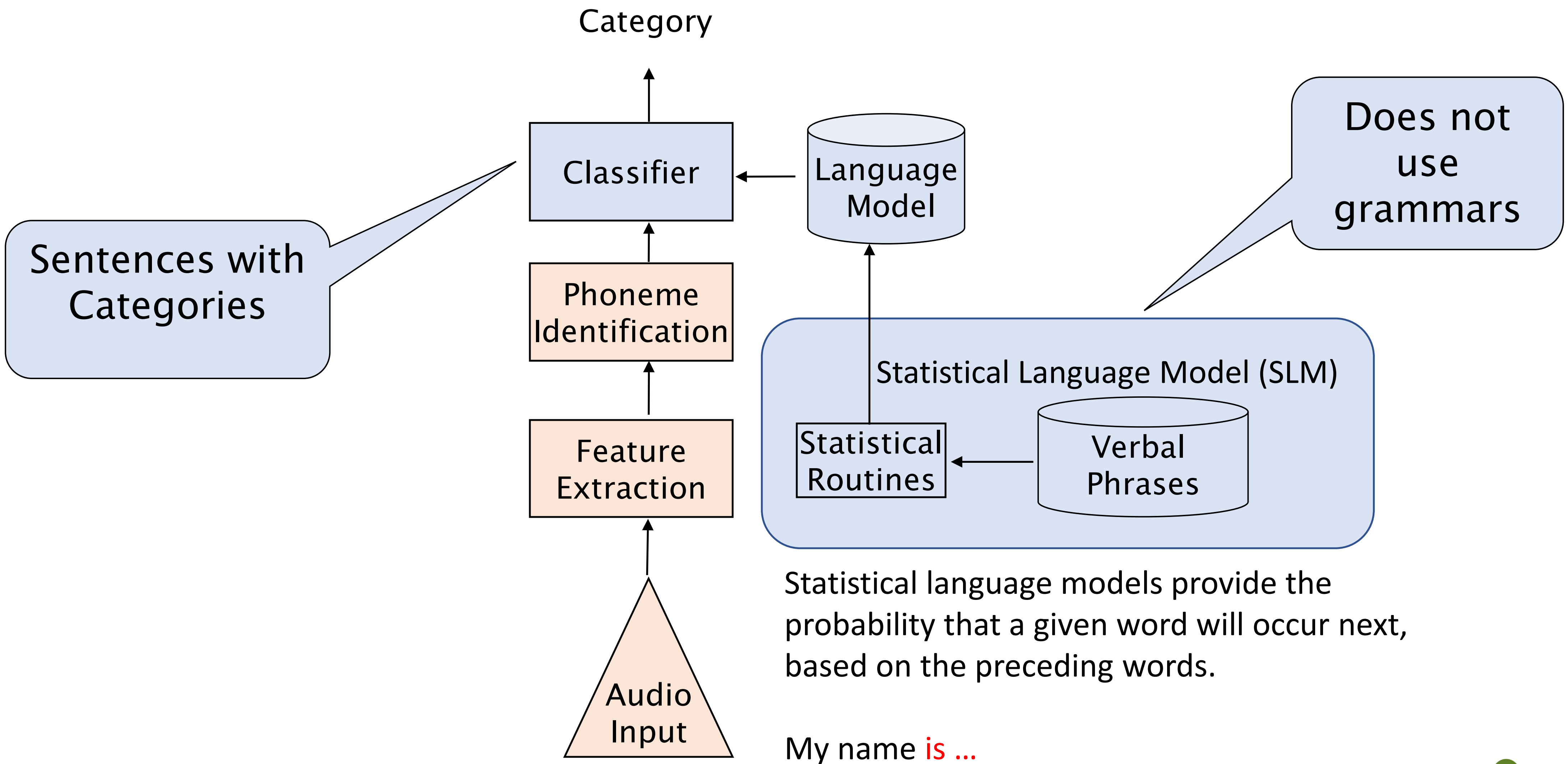


Grammar-based Speech Recognition



Portland State University

Statistical Language Model-based Speech Recognition

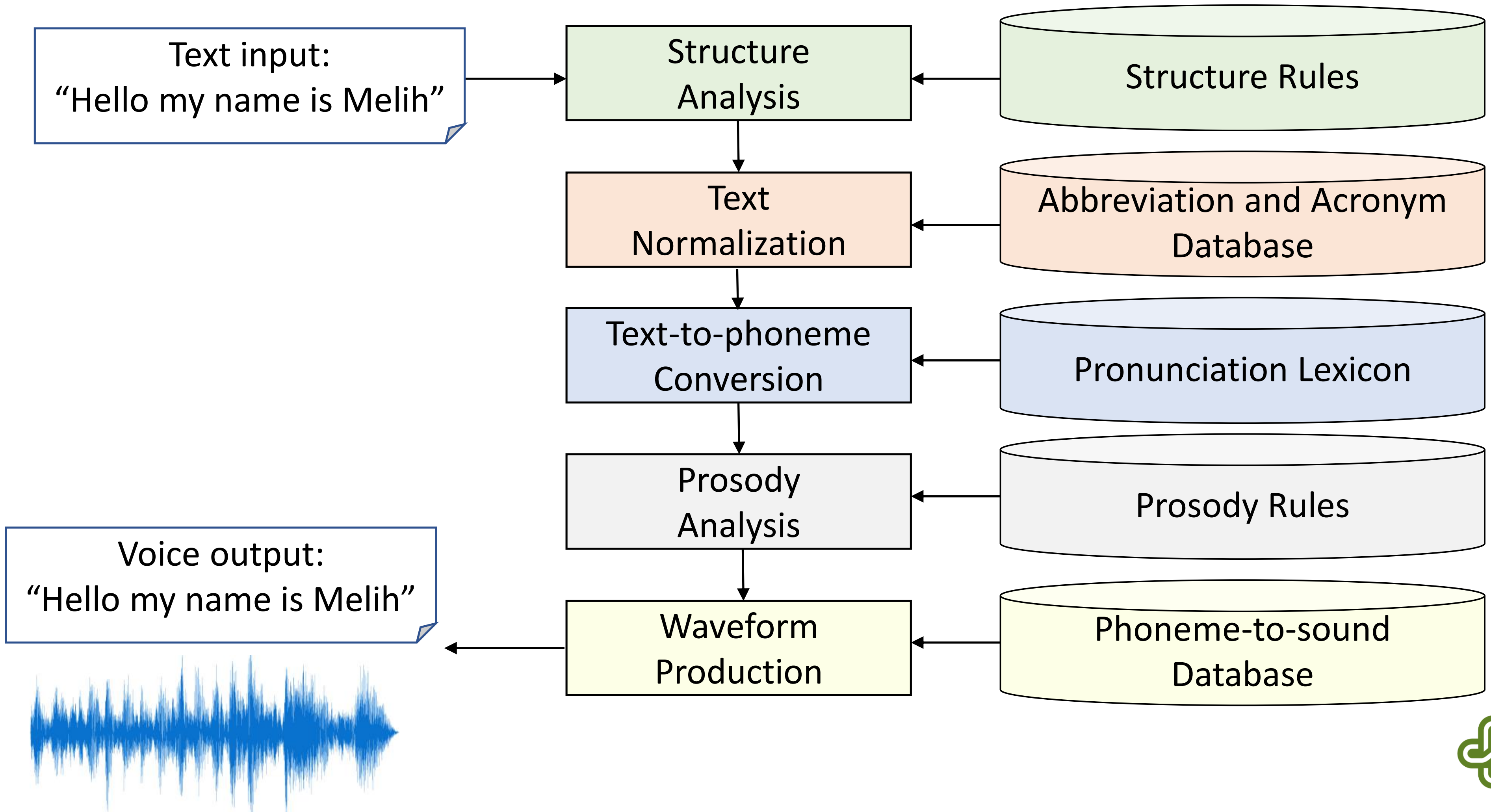


Statistical language models provide the probability that a given word will occur next, based on the preceding words.

My name is …

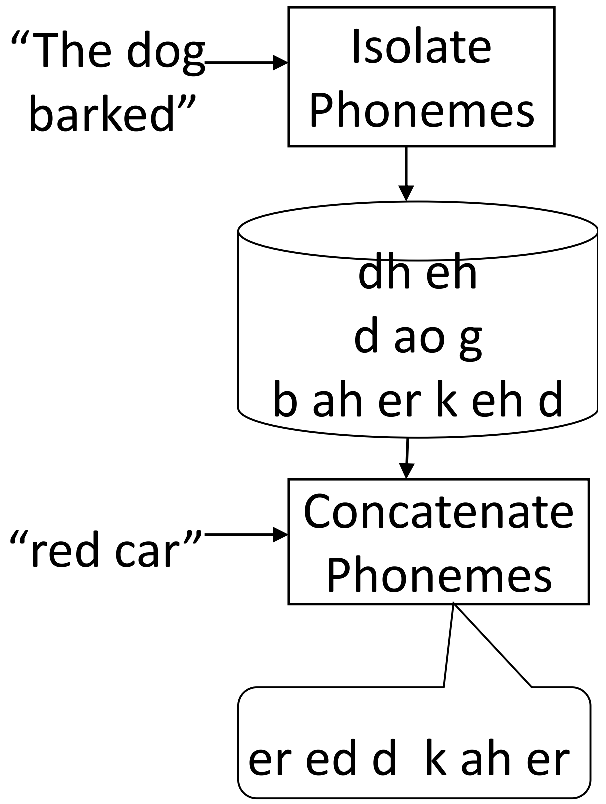
Speech Synthesis

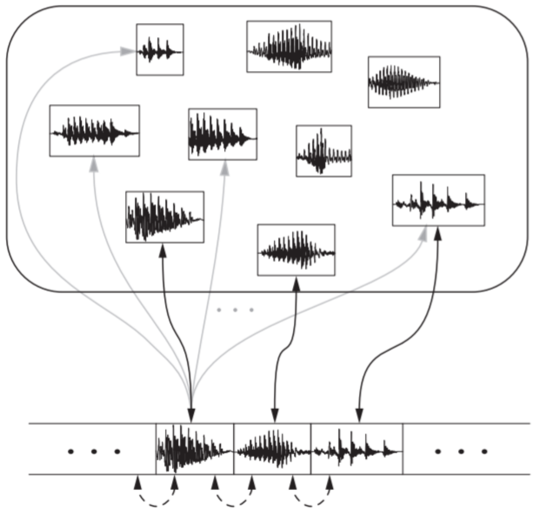
(Text-To-Speech, TTS)

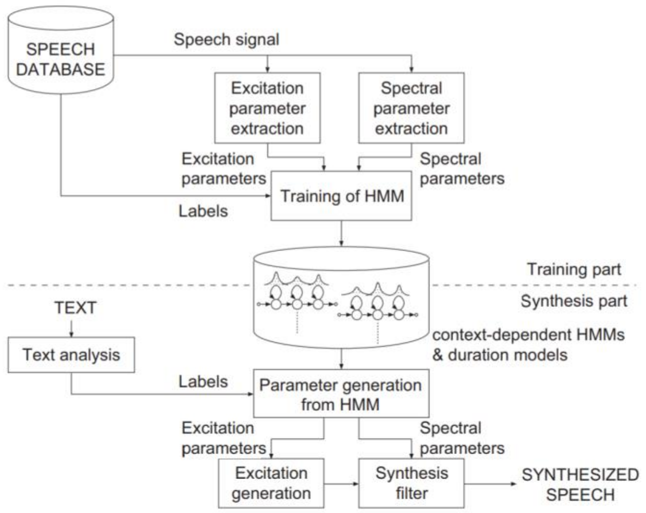


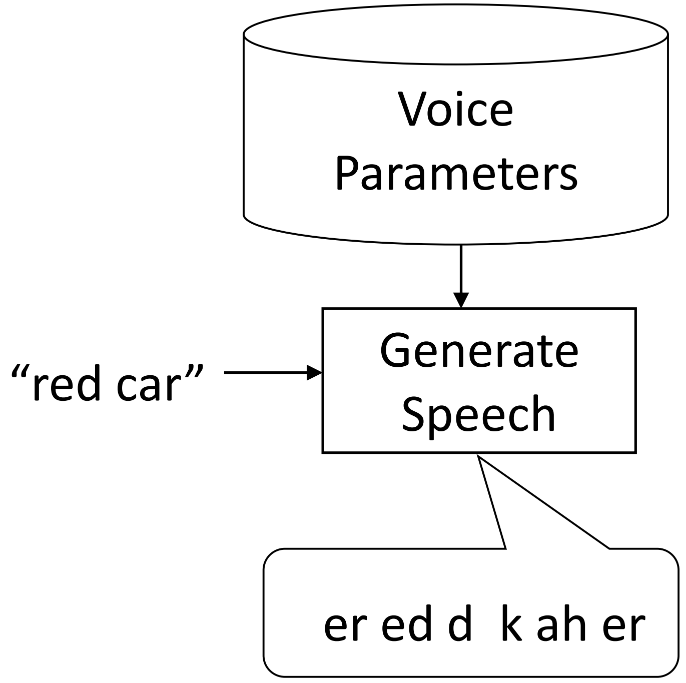
Portland State University

Concatenated vs. Parameter-based Speech Synthesis







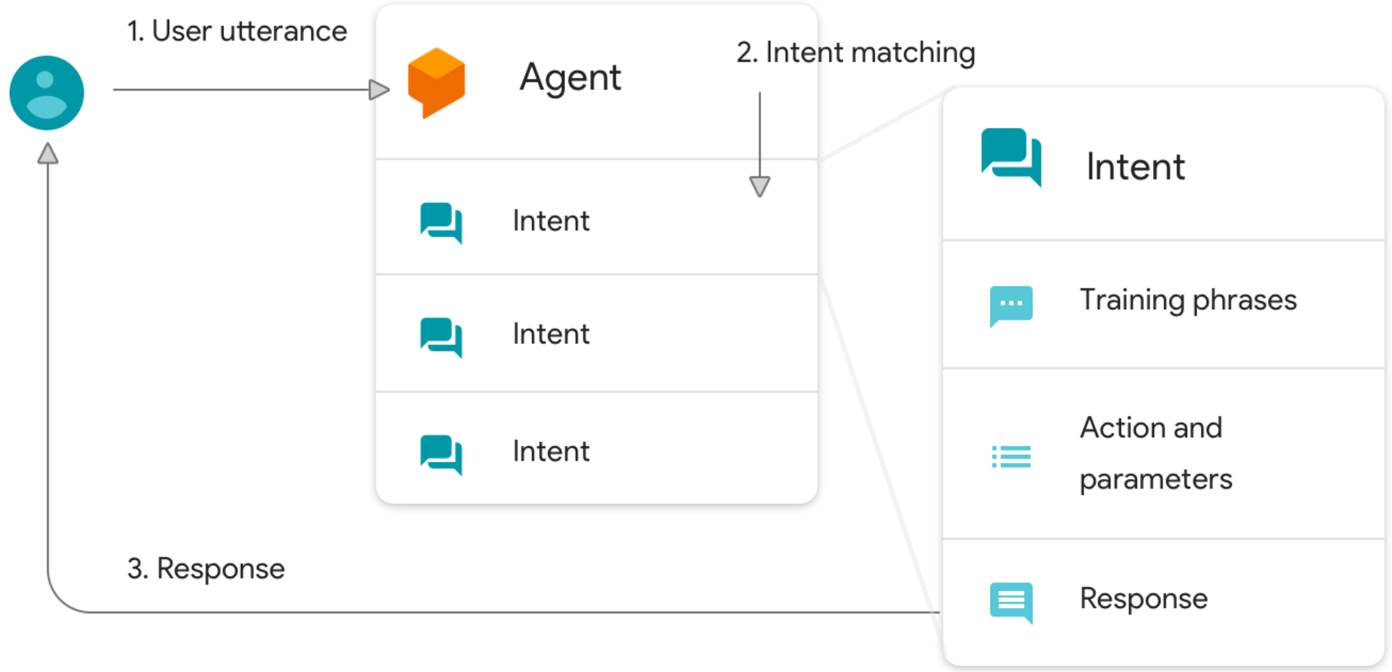


Concatenative synthesis is a technique for synthesizing sounds by concatenating short samples of recorded sound (called *units*).
Need a professional speaker 2-10 hr.

Parametric synthesis model describes the speech using parameters, rather than stored exemplars.
It is statistical because it describes those parameters using statistics

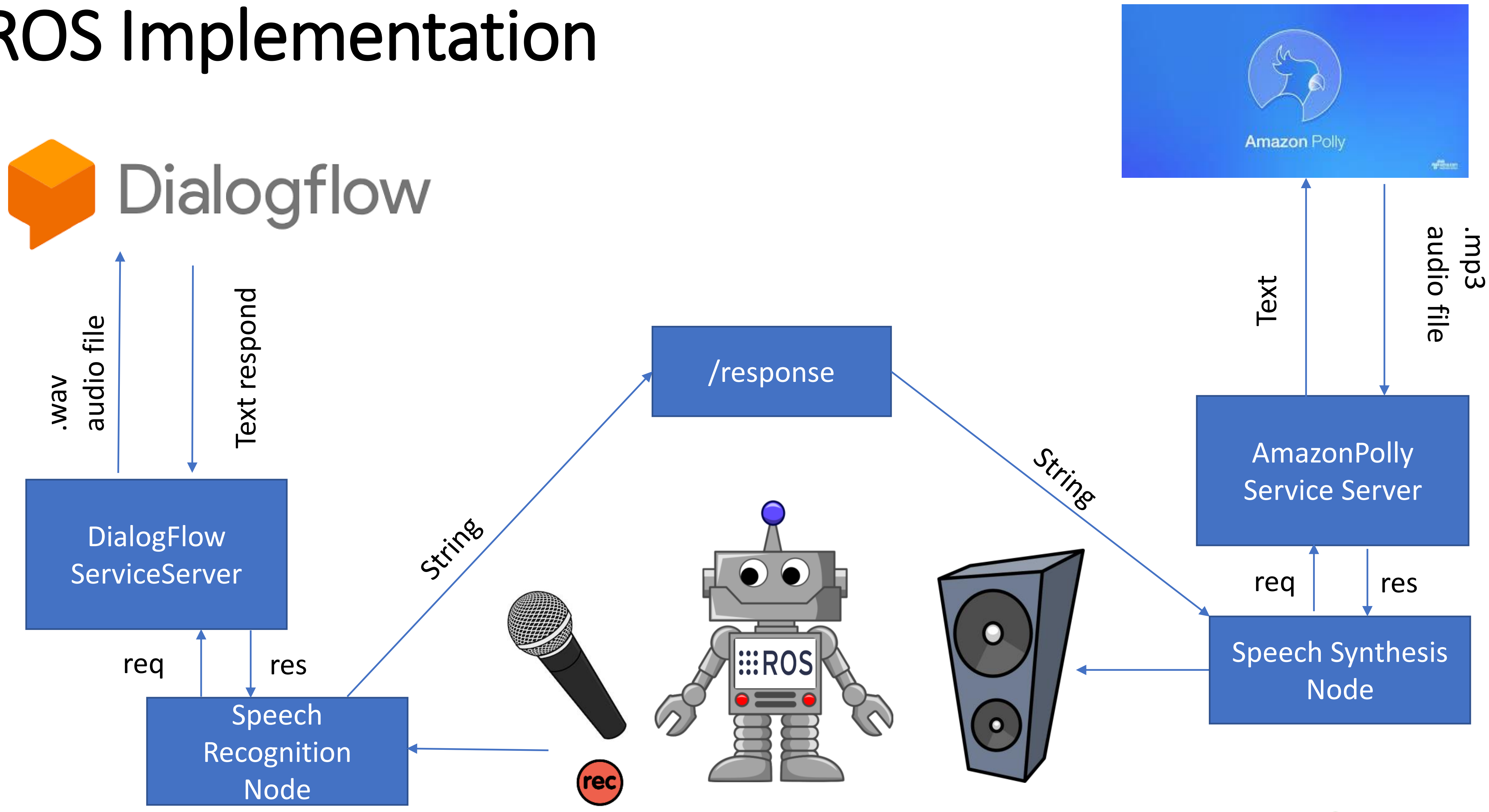
Dialogflow

- DialogFlow is a natural processing tool.



Portland State UNIVERSITY

ROS Implementation



Portland State UNIVERSITY

Amazon Polly

Portland State UNIVERSITY

Introduction to Amazon Polly

- Amazon Polly is a cloud service that converts text into lifelike speech.
- You can use Amazon Polly to develop applications that increase engagement and accessibility.
- Amazon Polly supports multiple languages and includes a variety of lifelike voices, so you can build speech-enabled applications that work in multiple locations and use the ideal voice for your users.

Portland State
UNIVERSITY

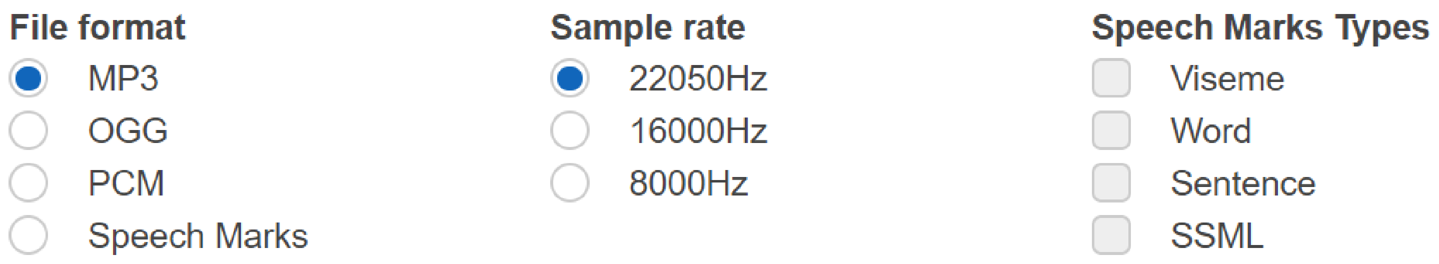
Introduction to Amazon Polly

- **High quality** – Amazon Polly uses advance Text-to-Speech (TTS) technology to synthesize natural speech with high pronunciation accuracy (including abbreviations, acronym expansions, date/time interpretations, and homograph disambiguation).
- **Low latency** – Amazon Polly ensures fast response times, which make it a viable option for low-latency use cases such as dialog systems.
- **Support for a large portfolio of languages and voices** – Amazon Polly supports dozens of voices and multiple languages, offering male and female voice options for most languages.
- **Cost-effective** – Amazon Polly's pay-per-use model means there are no setup costs. You can start small and scale up as your application grows.
- **Cloud-based solution** –Text-to-Speech conversion done in the cloud dramatically reduces local resource requirements. Moreover, speech improvements are instantly available to all end-users and do not require additional updates for devices.

Portland State UNIVERSITY

Introduction to Amazon Polly

- **Input text**
  - Provide the text you want to synthesize
  - You can provide the input as plain text or in Speech Synthesis Markup Language (SSML) format.
- **Available voices**
  - Amazon Polly provides a portfolio of multiple languages and a variety of voices, including a bilingual voice.
  - For most languages you can select from several different voices, both male and female
- **Output format** – Amazon Polly can deliver the synthesized speech in multiple formats. You can select the audio format that suits your needs. MP3, Ogg etc…



File format
- [✓ MP3]
- OGG
- PCM
- Speech Marks

Sample rate
- [✓ 22050Hz]
- 16000Hz
- 8000Hz

Speech Marks Types
- Viseme
- Word
- Sentence
- SSML

Using SSML – breaks <break>

- To add a pause to your text, use the <break> tag.
- <speak>
  Mary had a little lamb
  <break time="3s"/>
  Whose fleece was white as snow.
  </speak>

Portland State UNIVERSITY

Using SSML - paragraphs <p>

- To add a pause between paragraphs in your text, use the <p> tag. Using this tag provides a longer pause than native speakers usually place
- <speak>
<p>This is the first paragraph.
There should be a pause after this text is spoken.
</p>
<p>This is the second paragraph.
</p>
</speak>

Using SSML – sentence - <s>

To add a pause between lines or sentences in your text, use the `<s>` tag. Using this tag has the same effect as:

- Ending a sentence with a period (.)
- Specifying a pause with `<break strength="strong"/>`
- <speak>
  <s>
  Mary had a little lamb
  </s>
  <s>
  Whose fleece was white as snow
  </s>
  And everywhere that Mary went, the lamb was sure to go.
  </speak>

Using SSML - interpret-as - <say-as>

- Use the <say-as> tag with the interpret-as attribute to tell Amazon Polly how to say certain characters, words, and numbers.
- This enables you to provide additional context to eliminate any ambiguity on how Amazon Polly should render the text.

- <speak>

My phone number is
<say-as interpret-as="telephone"
206 111 22 33
</say-as>
.
</speak>

Using SSML - Prosody - <prosody>

- To control the volume, rate, or pitch of your selected voice, use the prosody tag.
- <speak>
  Sometimes it can sometimes be useful to
  <prosody volume="loud"> increase the volume for a specific speech.</prosody>
  </speak>
- <speak> For dramatic purposes, you might wish to
  <prosody rate="slow">speed up the speaking rate of your text.</prosody>
  </speak>

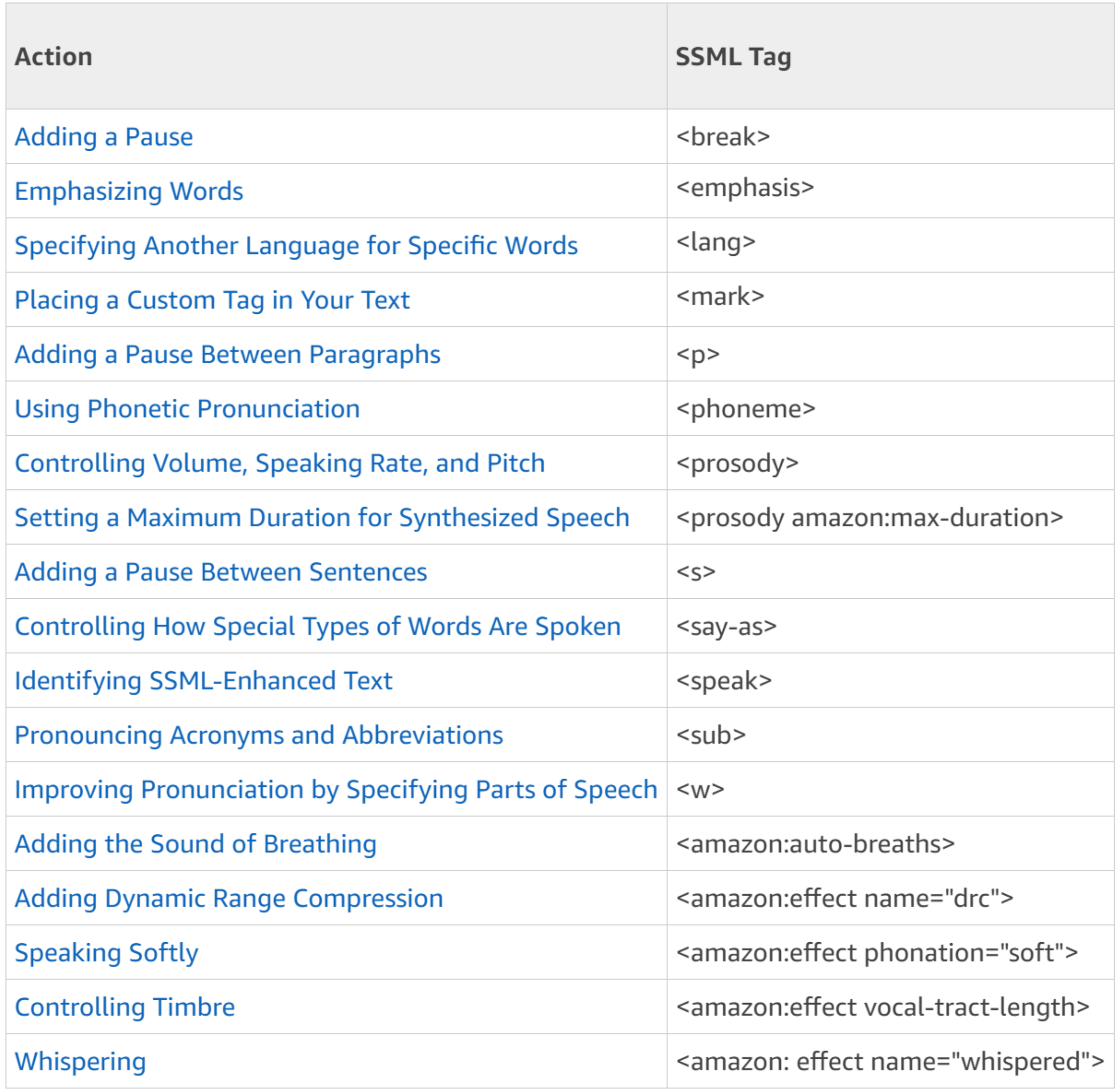
Using SSML

- To make Amazon Polly use phonetic pronunciation for specific text, use the <phoneme> tag.
- <speak>
  You say,
  <phoneme alphabet="ipa" ph="pɪˈkɑːn">
  pecan
  </phoneme>.
  I say,
  <phoneme alphabet="ipa" ph="ˈpi.kæn">
  pecan
  </phoneme>.
  </speak>

- ipa: International Phonetic Alphabet (IPA)
- ph: Specifies the phonetic symbols

Using SSML – More tags and options

https://docs.aws.amazon.com/polly/latest/dg/supported-ssml.html



Action	SSML Tag
Adding a Pause	<break>
Emphasizing Words	<emphasis>
Specifying Another Language for Specific Words	<lang>
Placing a Custom Tag in Your Text	<mark>
Adding a Pause Between Paragraphs	<p>
Using Phonetic Pronunciation	<phoneme>
Controlling Volume, Speaking Rate, and Pitch	<prosody>
Setting a Maximum Duration for Synthesized Speech	<prosody amazon:max-duration>
Adding a Pause Between Sentences	<s>
Controlling How Special Types of Words Are Spoken	<say-as>
Identifying SSML-Enhanced Text	<speak>
Pronouncing Acronyms and Abbreviations	<sub>
Improving Pronunciation by Specifying Parts of Speech	<w>
Adding the Sound of Breathing	<amazon:auto-breaths>
Adding Dynamic Range Compression	<amazon:effect name="drc">
Speaking Softly	<amazon:effect phonation="soft">
Controlling Timbre	<amazon:effect vocal-tract-length>
Whispering	<amazon: effect name="whispered">

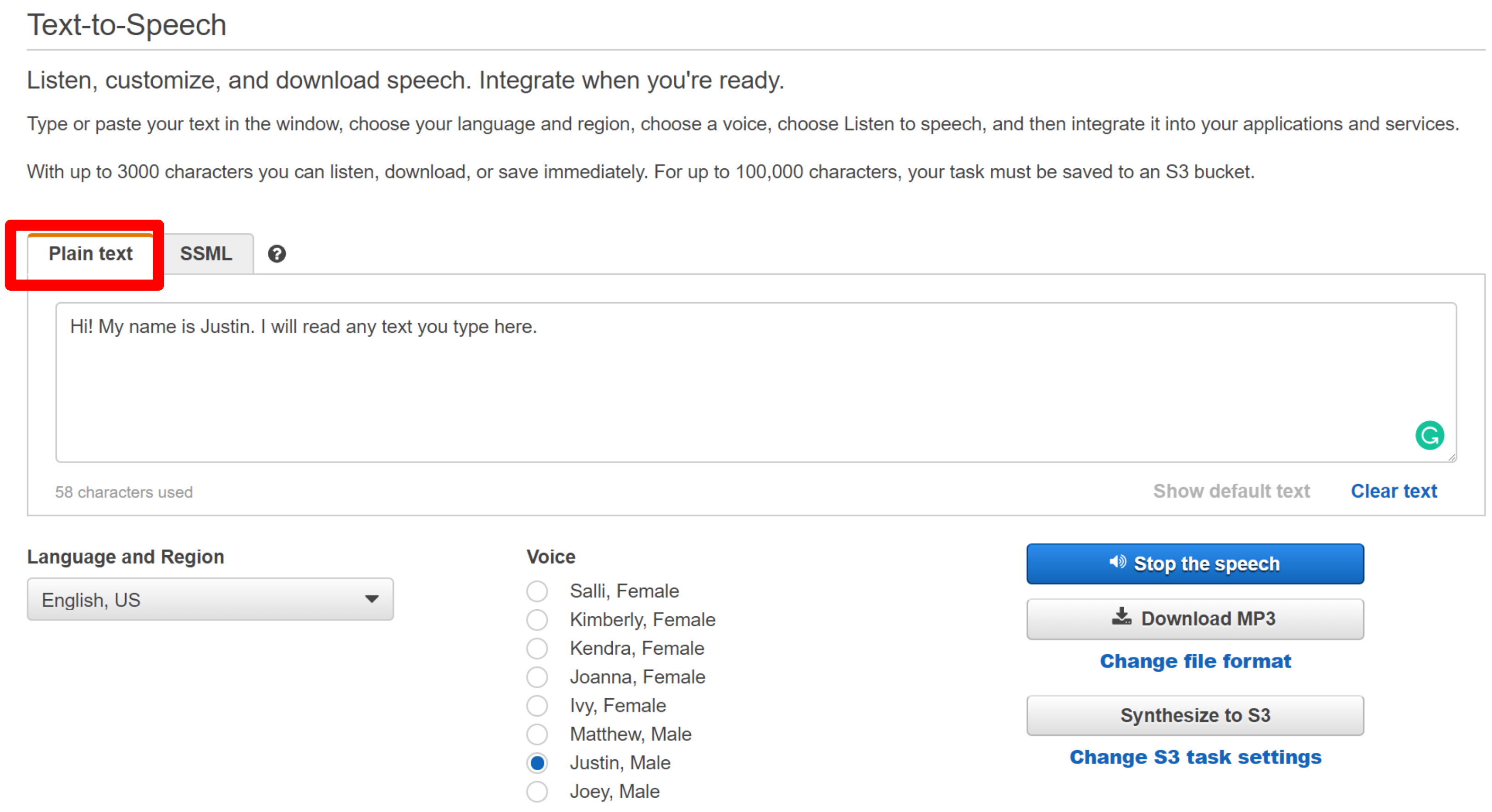
Portland State UNIVERSITY

If you want to learn more about Spoken Language Interfaces and Human Computer Interaction

- Portland State University
- Computer Science Department
- Spoken Language Interfaces - CS 410/510
- Instructor: Jim Larson
- Some slides from his lectures are updated and used in this documentation

Portland State
UNIVERSITY

Introduction to Amazon Polly



Text-to-Speech

Listen, customize, and download speech. Integrate when you're ready.

Type or paste your text in the window, choose your language and region, choose a voice, choose Listen to speech, and then integrate it into your applications and services.

With up to 3000 characters you can listen, download, or save immediately. For up to 100,000 characters, your task must be saved to an S3 bucket.

Hi! My name is Justin. I will read any text you type here.

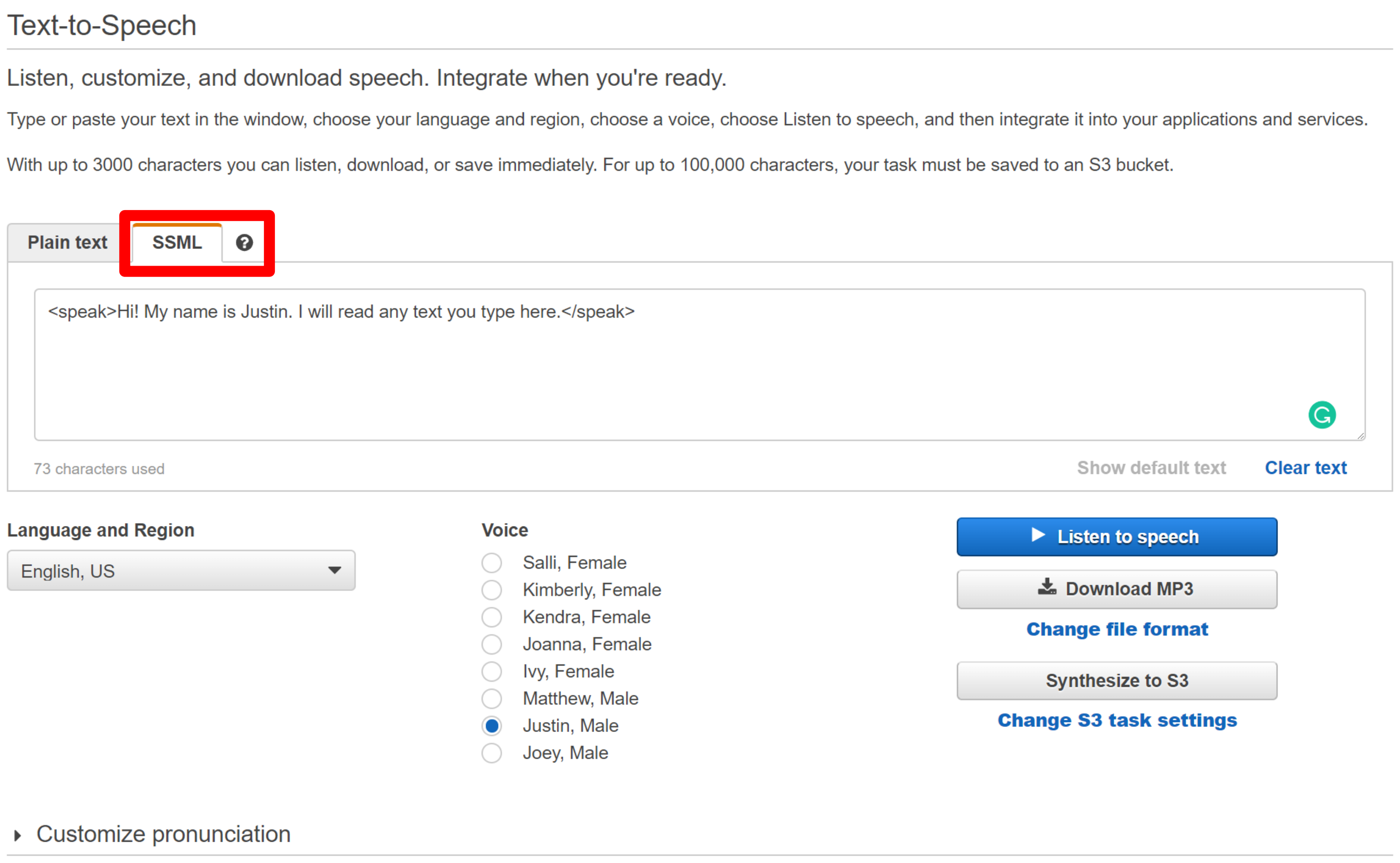
58 characters used

Language and Region: English, US

Voice:
- ( ) Salli, Female
- ( ) Kimberly, Female
- ( ) Kendra, Female
- ( ) Joanna, Female
- ( ) Ivy, Female
- ( ) Matthew, Male
- (●) Justin, Male
- ( ) Joey, Male

Portland State UNIVERSITY

Introduction to Amazon Polly



Text-to-Speech

Listen, customize, and download speech. Integrate when you're ready.

Type or paste your text in the window, choose your language and region, choose a voice, choose Listen to speech, and then integrate it into your applications and services.

With up to 3000 characters you can listen, download, or save immediately. For up to 100,000 characters, your task must be saved to an S3 bucket.

```
<speak>Hi! My name is Justin. I will read any text you type here.</speak>
```

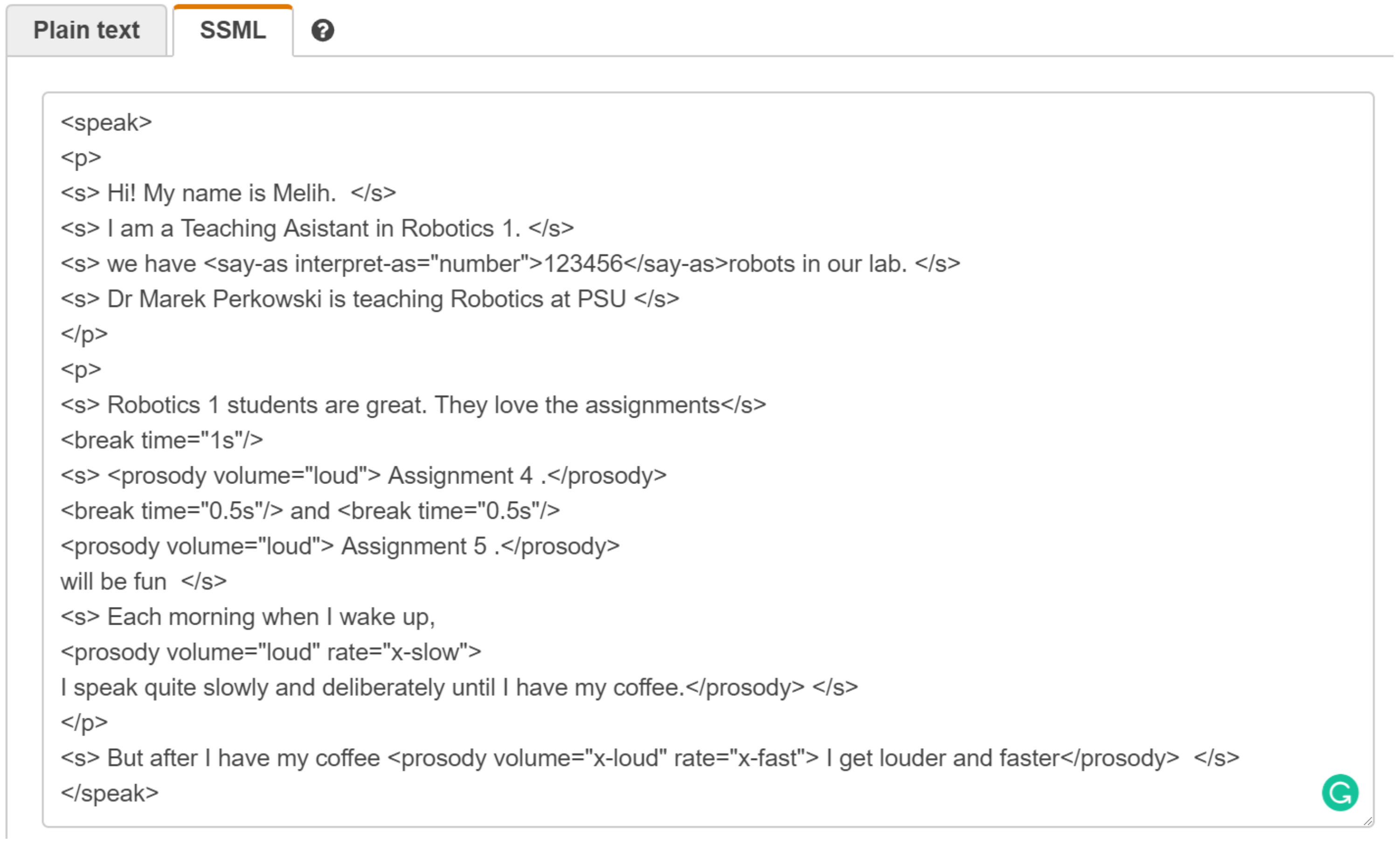
73 characters used

Language and Region: English, US

Voice:
- Salli, Female
- Kimberly, Female
- Kendra, Female
- Joanna, Female
- Ivy, Female
- Matthew, Male
- [✓ Justin, Male]
- Joey, Male

Portland State UNIVERSITY

Introduction to Amazon Polly



```
<speak>
<p>
<s> Hi! My name is Melih.  </s>
<s> I am a Teaching Asistant in Robotics 1. </s>
<s> we have <say-as interpret-as="number">123456</say-as>robots in our lab. </s>
<s> Dr Marek Perkowski is teaching Robotics at PSU </s>
</p>
<p>
<s> Robotics 1 students are great. They love the assignments</s>
<break time="1s"/>
<s> <prosody volume="loud"> Assignment 4 .</prosody>
<break time="0.5s"/> and <break time="0.5s"/>
<prosody volume="loud"> Assignment 5 .</prosody>
will be fun  </s>
<s> Each morning when I wake up,
<prosody volume="loud" rate="x-slow">
I speak quite slowly and deliberately until I have my coffee.</prosody> </s>
</p>
<s> But after I have my coffee <prosody volume="x-loud" rate="x-fast"> I get louder and faster</prosody>  </s>
</speak>
```

Portland State UNIVERSITY

ROS
(Robot Operating System)

Portland State UNIVERSITY

ROS Action

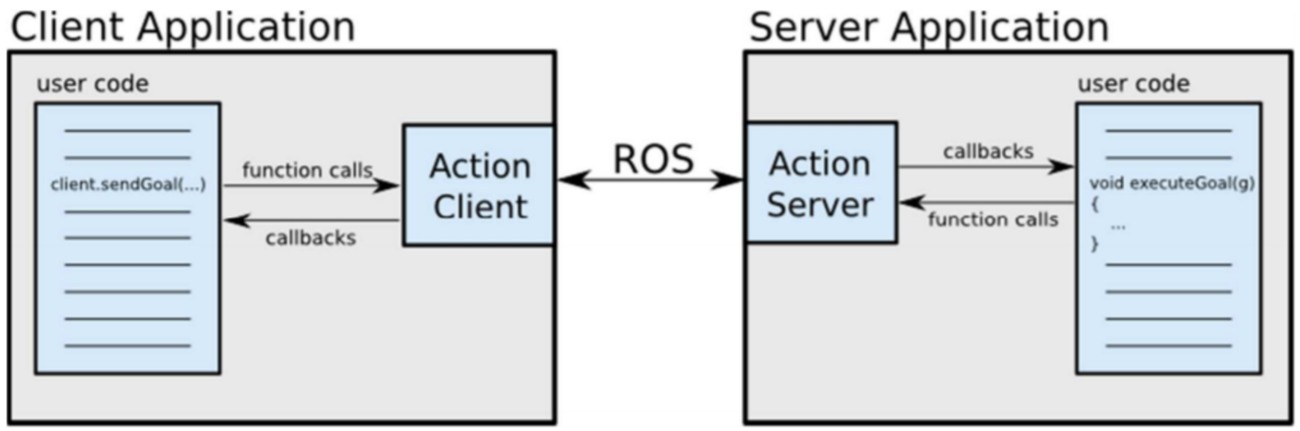
Portland State
UNIVERSITY

ROS Action

- One node sends a request to another node to perform some task
- It is like ROS services, however when service performs a task we can't do any other work
- Action are needed:
  - When the requested task takes time
  - When we want to some other things as the task performed
  - When we want to monitor the task, have continuous feedback about the task and possibly cancel the request during execution or restart it again.
- Actionlib package is used to create action servers and clients
- More powerful and flexible
- Complicated – You need to know what you are doing

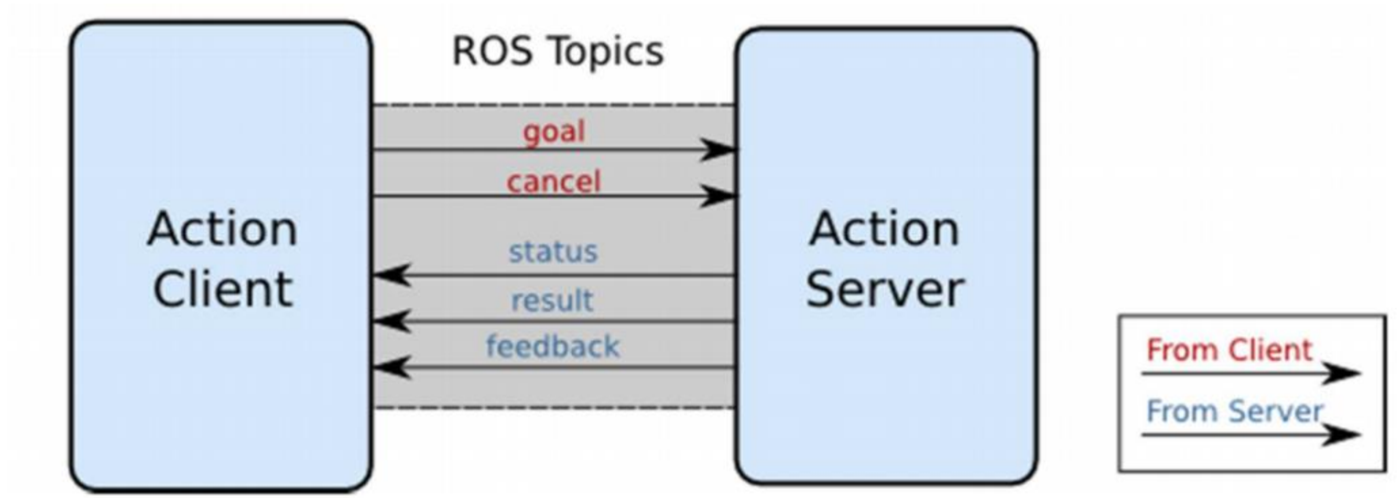
ROS Action

- actionlib package is used to create action servers and clients
- Action servers execute ling running tasks
- Action clients interacts with servers by calling actions



ROS Action

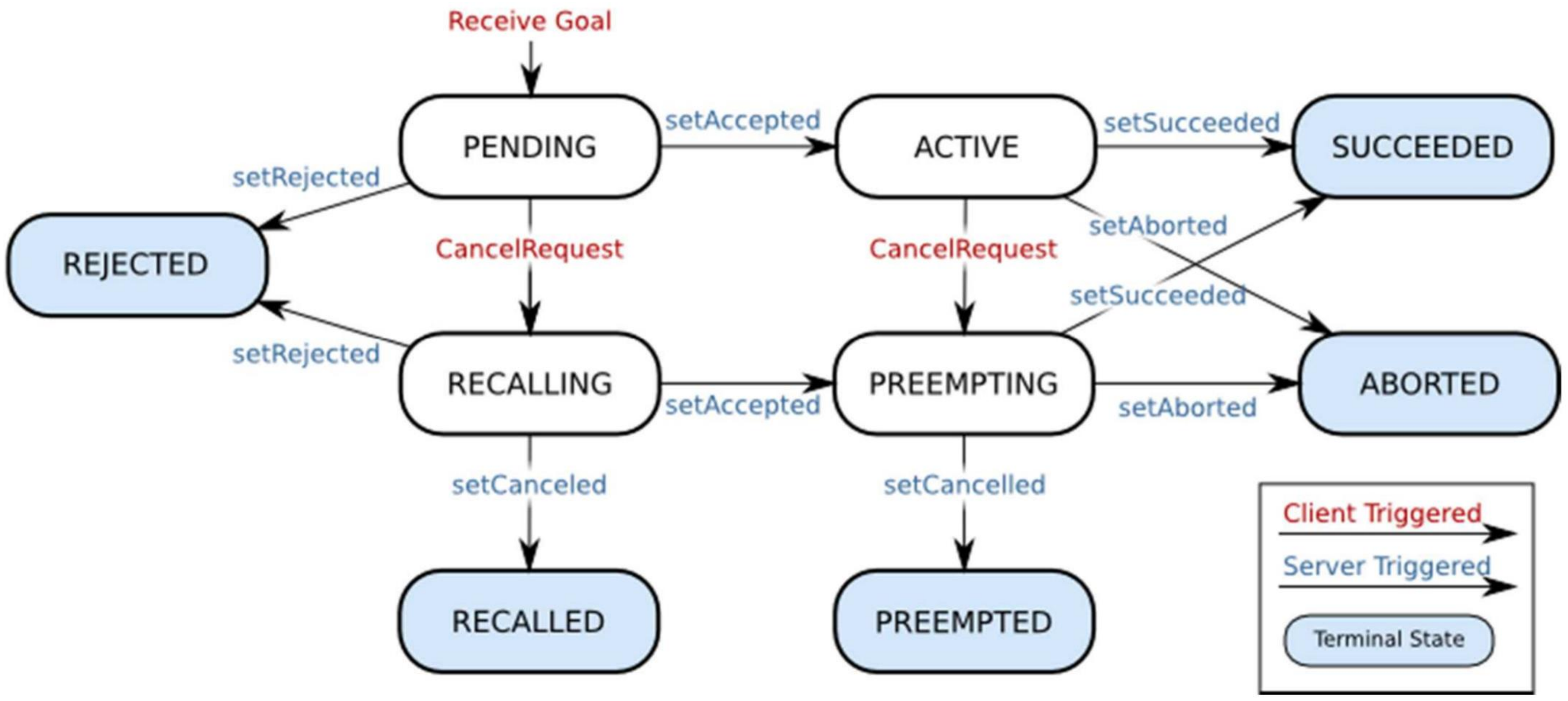
- ROS Action and Server Interaction
- **goal:** used to send new goals to server
- **cancel:** used to send cancel requests to server
- **status:** used to notify clients on the current state of every goal in the system
- **feedback:** used to send clients periodic auxiliary information for goal
- **result:** used to send clients one-time auxiliary information upon completion of a goal



ROS Action

- Action templates are defined by a name and some additional properties through an .action structure defined in ROS
- Each instance of an action has a unique Goal ID
- Goal ID provides the action server and the action client with a robust way to monitor the execution of a particular instance of an action.

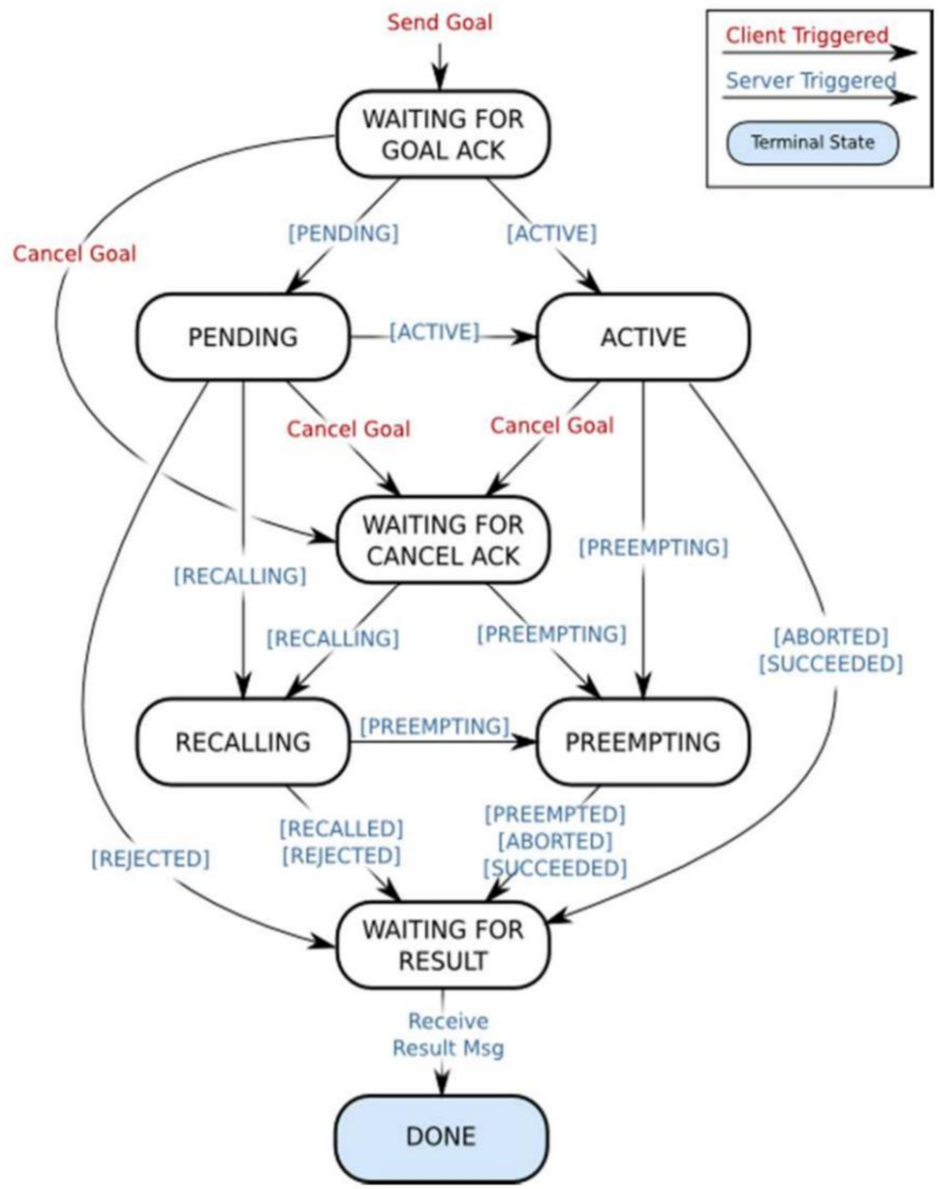
Server State Machine



Action Server State Machine

- **setAccepted** - After inspecting a goal, decide to start processing it
- **setRejected** - After inspecting a goal, decide to never process it because it is an invalid request (out of bounds, resources not available, invalid, etc)
- **setSucceeded** - Notify that goal has been successfully processed
- **setAborted** - Notify that goal encountered an error during processsing, and had to be aborted
- **setCanceled** - Notify that goal is no longer being processed, due to a cancel request
- The action client can also asynchronously trigger state transitions:
- **CancelRequest**: The client notifies the action server that it wants the server to stop processing the goal.

Action Client State Machine



Portland State UNIVERSITY

Action Client State Machine

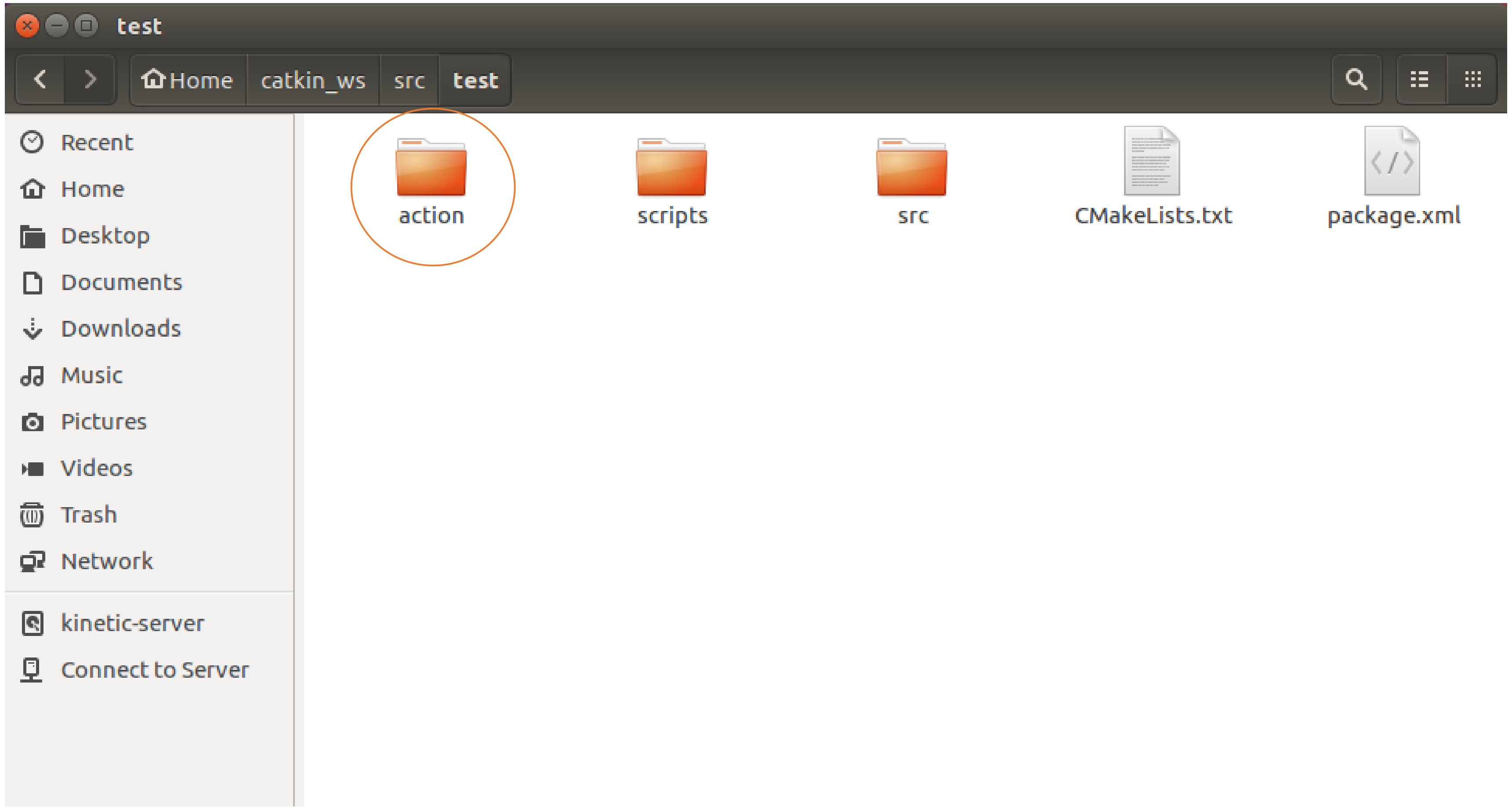
Intermediate States

Pending - The goal has yet to be processed by the action server
Active - The goal is currently being processed by the action server
Recalling - The goal has not been processed and a cancel request has been received from the action client, but the action server has not confirmed the goal is canceled
Preempting - The goal is being processed, and a cancel request has been received from the action client, but the action server has not confirmed the goal is canceled

Terminal States
Rejected - The goal was rejected by the action server without being processed and without a request from the action client to cancel
Succeeded - The goal was achieved successfully by the action server
Aborted - The goal was terminated by the action server without an external request from the action client to cancel
Recalled - The goal was canceled by either another goal, or a cancel request, before the action server began processing the goal
Preempted - Processing of the goal was canceled by either another goal, or a cancel request sent to the action server

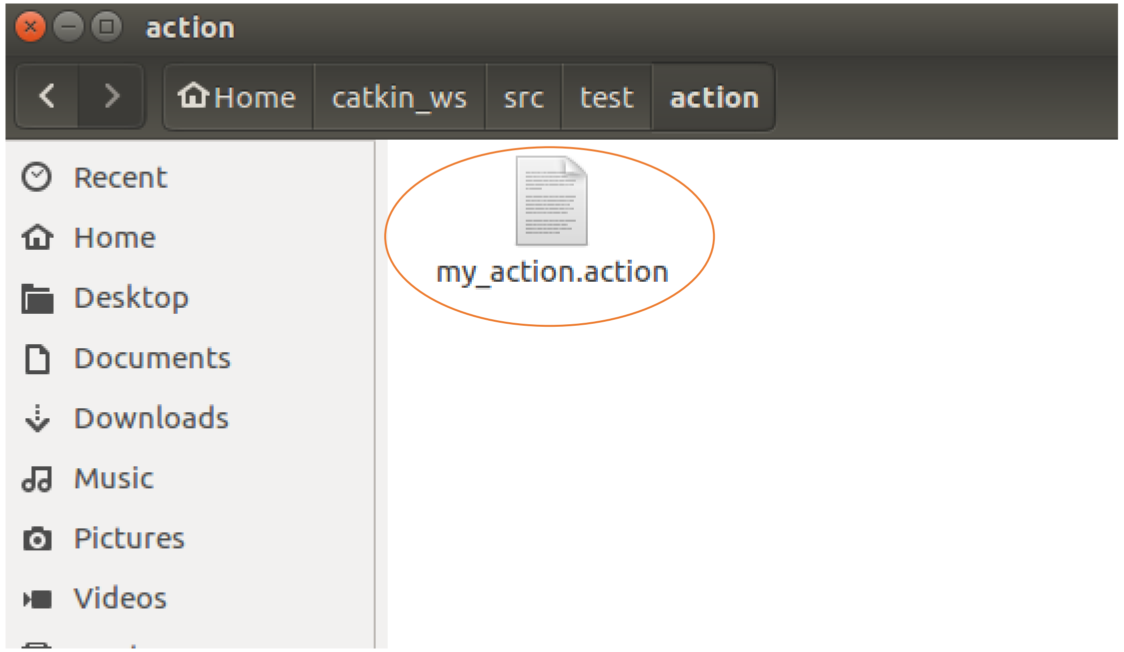
How to create ROS Action Server and Client?

- Create a directory called action in your package



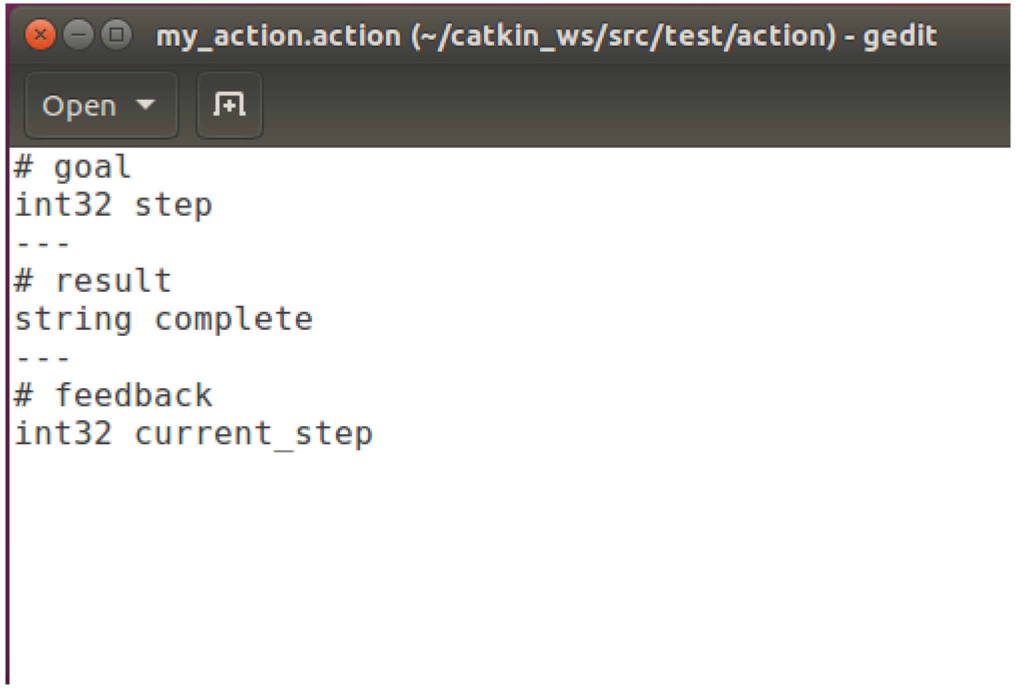
How to create ROS Action Server and Client?

- Create an action file in /action directory.



How to create ROS Action Server and Client?

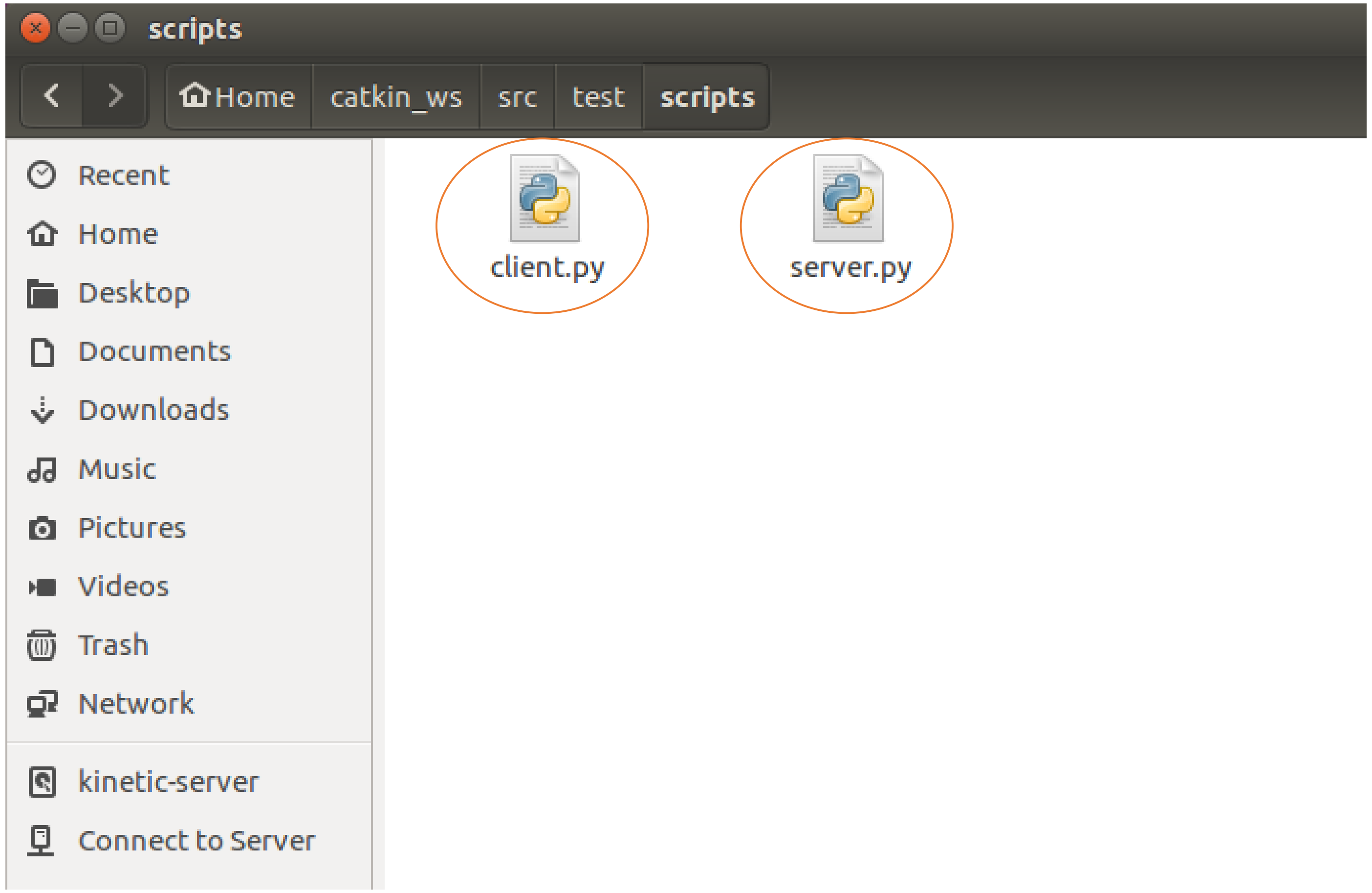
- Add goal, result, and feedback message type and name information into the action file



```
# goal
int32 step
---
# result
string complete
---
# feedback
int32 current_step
```

How to create ROS Action Server and Client?

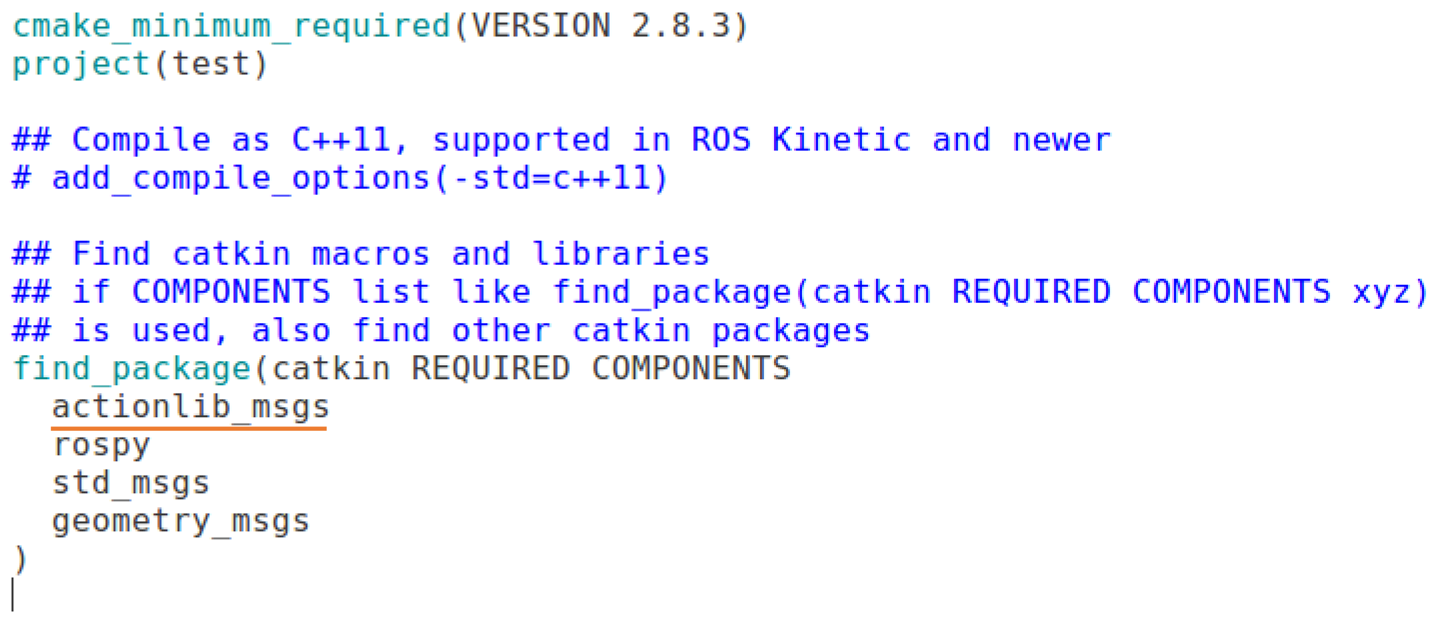
- Create two scripts – One node will work as an action client and the other node will work as an action server



Portland State UNIVERSITY

How to create ROS Action Server and Client?

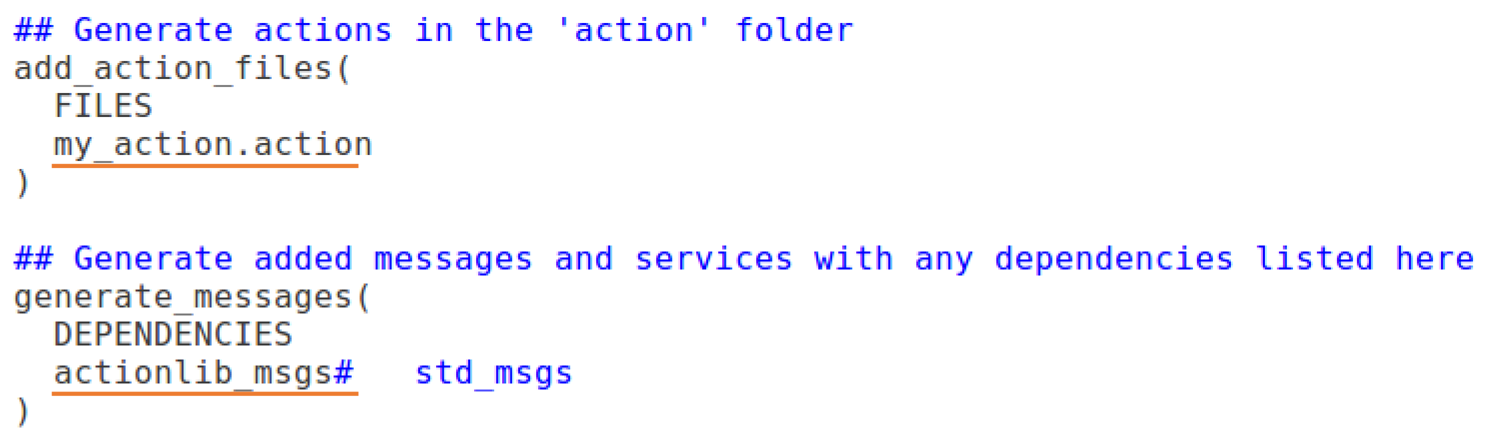
- In the CMakeList.txt file in the package
- Add actionlib_msgs to the package list



```
cmake_minimum_required(VERSION 2.8.3)
project(test)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  actionlib_msgs
  rospy
  std_msgs
  geometry_msgs
)
```

How to create ROS Action Server and Client?

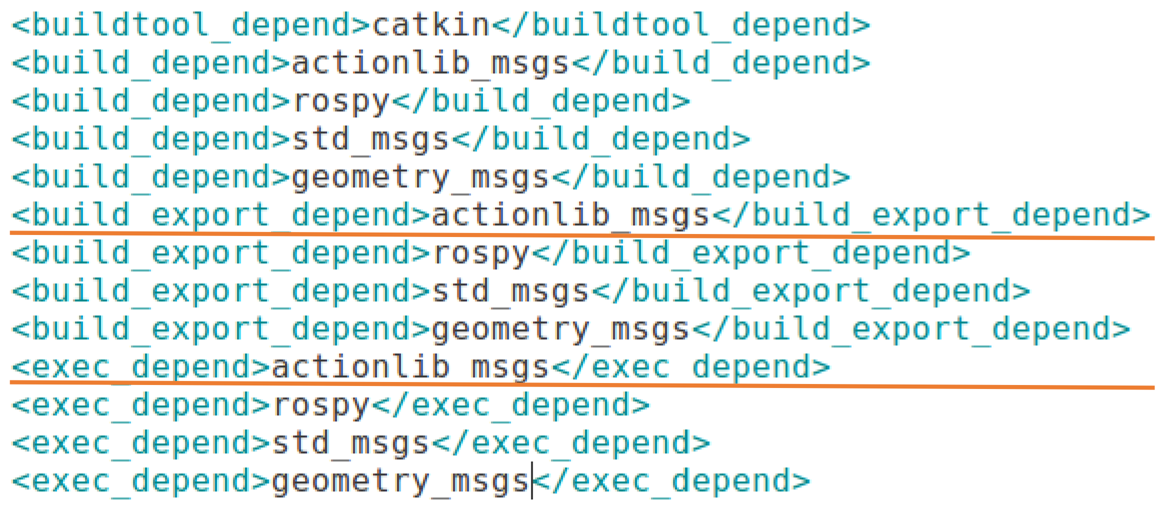
- In the CMakeList.txt file in the package
- Add the action file intop action files list
- Add actionlib_msgs package as a dependency



```
## Generate actions in the 'action' folder
add_action_files(
  FILES
  my_action.action
)

## Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  actionlib_msgs#   std_msgs
)
```

How to create ROS Action Server and Client?

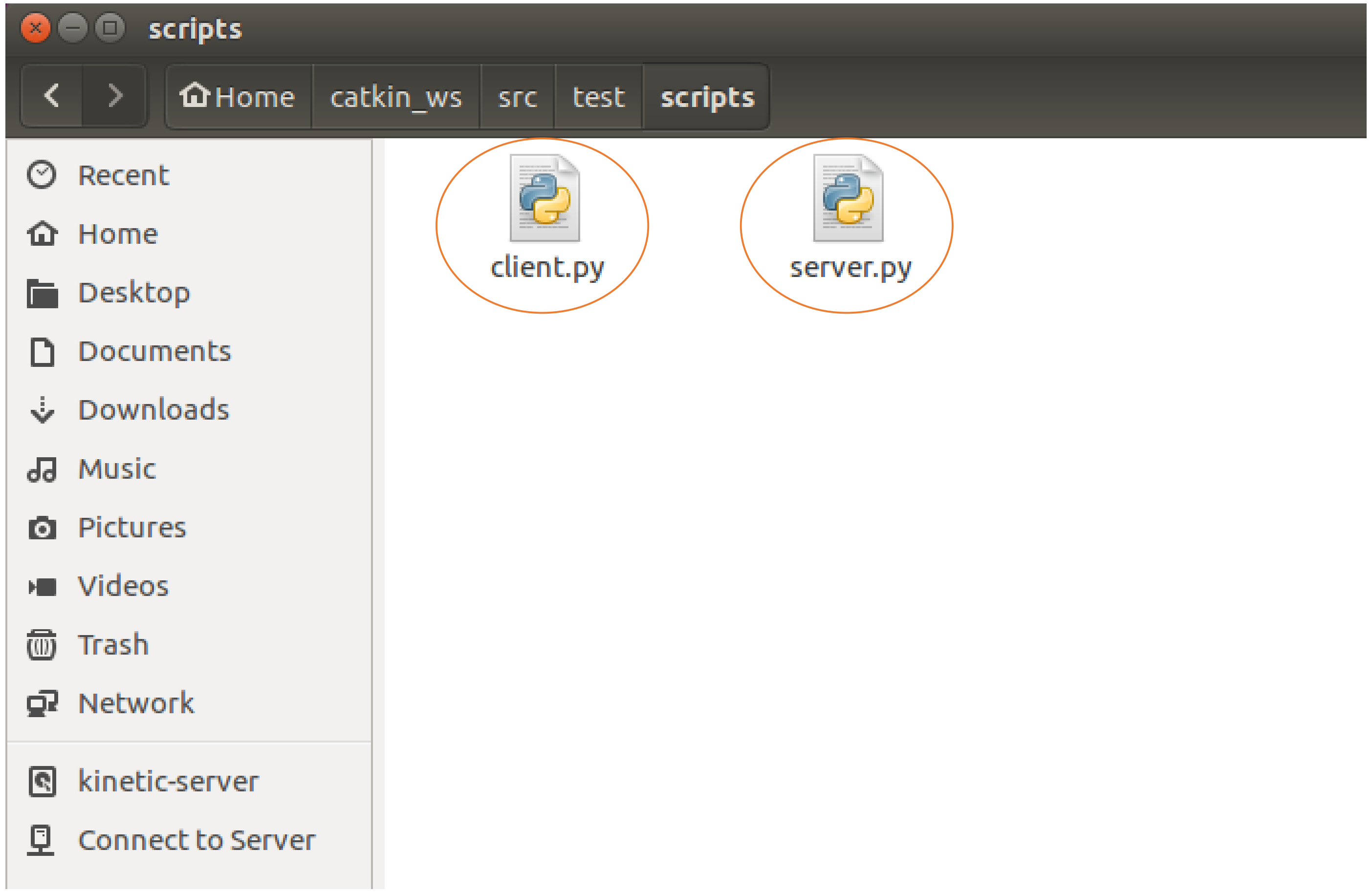
- In package.xml add the dependencies for the package



```
<buildtool_depend>catkin</buildtool_depend>
<build_depend>actionlib_msgs</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>geometry_msgs</build_depend>
<build_export_depend>actionlib_msgs</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<build_export_depend>geometry_msgs</build_export_depend>
<exec_depend>actionlib_msgs</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
<exec_depend>geometry_msgs</exec_depend>
```

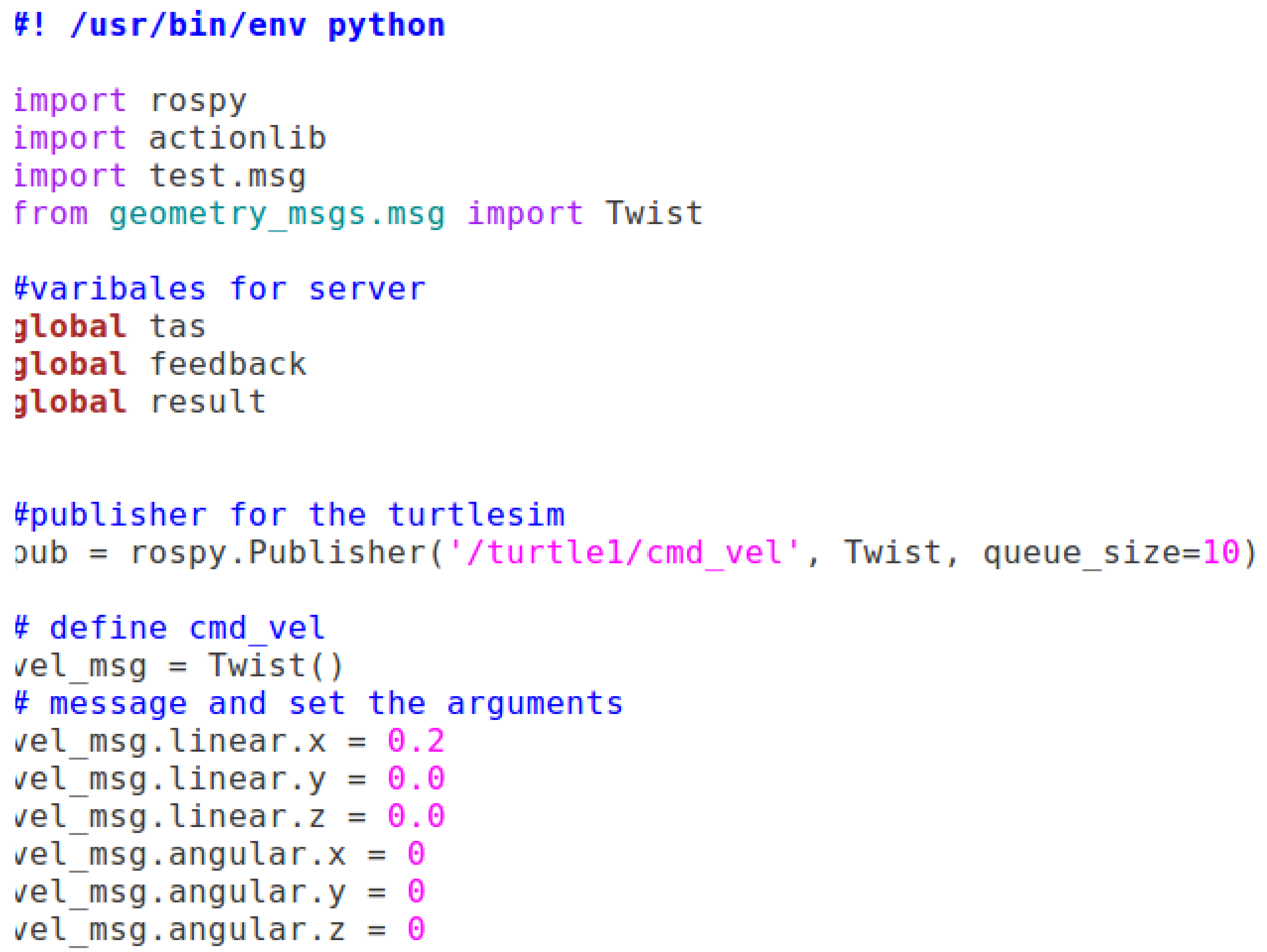
How to create ROS Action Server and Client?

- Create two scripts – One node will work as an action client and the other node will work as an action server



ROS Action Server

- Action Server (server.py)



```
#! /usr/bin/env python

import rospy
import actionlib
import test.msg
from geometry_msgs.msg import Twist

#varibales for server
global tas
global feedback
global result


#publisher for the turtlesim
pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

# define cmd_vel
vel_msg = Twist()
# message and set the arguments
vel_msg.linear.x = 0.2
vel_msg.linear.y = 0.0
vel_msg.linear.z = 0.0
vel_msg.angular.x = 0
vel_msg.angular.y = 0
vel_msg.angular.z = 0
```

ROS Action Server

- Action Server (server.py)

```
#Action Server
def testAction():
    global tas
    global feedback
    global result

    # create messages that are used to publish feedback/result
    feedback = test.msg.my_actionFeedback()
    result = test.msg.my_actionResult()

    #create a simple action server with a namespace, actiontype, callbackfuntion
    tas = actionlib.SimpleActionServer("TAS", test.msg.my_actionAction, execute_cb, auto_start = False)

    #start the action server
    tas.start()
```

ROS Action Server

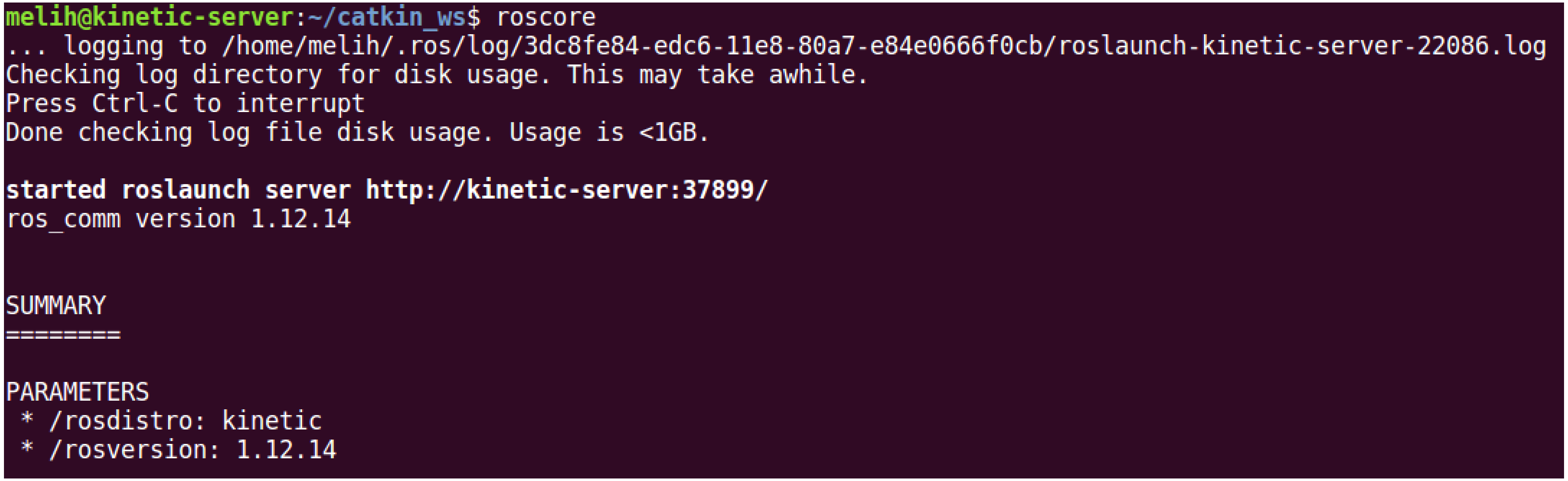
- Action Server (server.py)

```python
def execute_cb(goal):
    global tas
    global feedback
    global result
    success = True

    #set the rate 2 times per second
    r = rospy.Rate(0.5)
    print("action is executing")

    # start executing the action
    for i in range(0, goal.step):
        # check that preempt has not been requested by the client
        if tas.is_preempt_requested():
            rospy.loginfo('Preempted')
            tas.set_preempted()
            success = False
            break

        #move the turtle
        pub.publish(vel_msg)
        print("turtle is moving")

        #set and publish the feedback
        feedback.current_step = i
        tas.publish_feedback(feedback)

        #sleep for 500 ms
        r.sleep()
```

```python
# when everything is succesfully done
    if success:
       # set and publish the result
       result.complete = "done"
       tas.set_succeeded(result)
       print("action is done")

if __name__ == '__main__':
    rospy.init_node('test_action_server_node')
    testAction()
    rospy.spin()
```

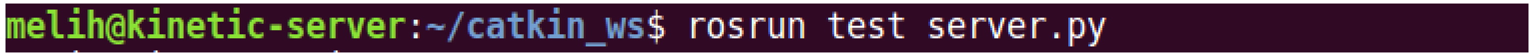
How to call Actions?

- run roscore



```
melih@kinetic-server:~/catkin_ws$ roscore
... logging to /home/melih/.ros/log/3dc8fe84-edc6-11e8-80a7-e84e0666f0cb/roslaunch-kinetic-server-22086.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://kinetic-server:37899/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14
```

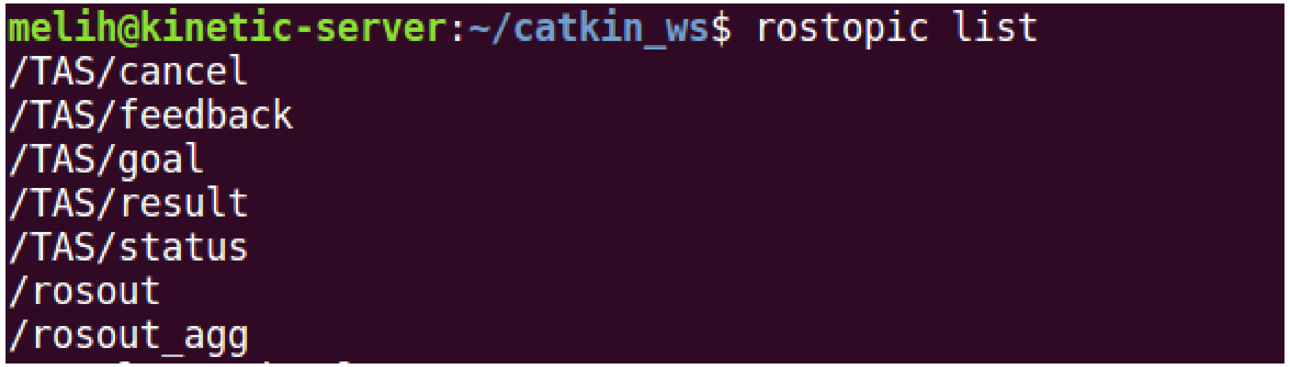
- run server action node



```
melih@kinetic-server:~/catkin_ws$ rosrun test server.py
```

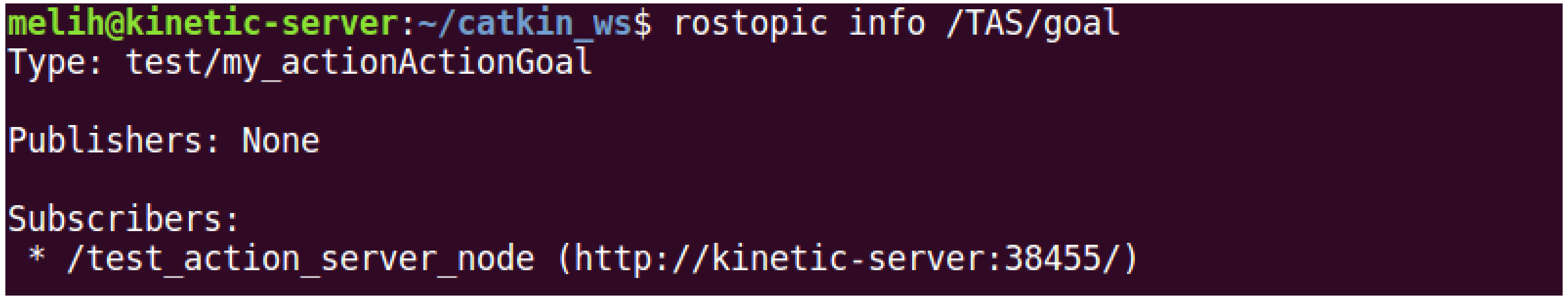
How to call Actions?

- There is no rosaction command!
- rostopic is called to call action
- List all the available actions



```
melih@kinetic-server:~/catkin_ws$ rostopic list
/TAS/cancel
/TAS/feedback
/TAS/goal
/TAS/result
/TAS/status
/rosout
/rosout_agg
```

- Get information about actions

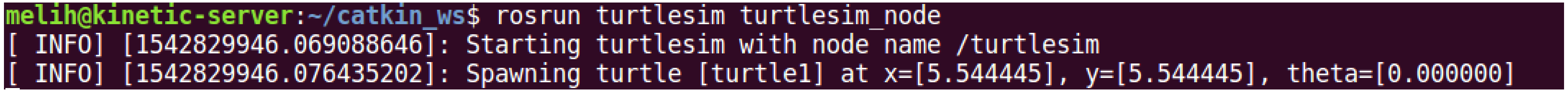


```
melih@kinetic-server:~/catkin_ws$ rostopic info /TAS/goal
Type: test/my_actionActionGoal

Publishers: None

Subscribers:
 * /test_action_server_node (http://kinetic-server:38455/)
```

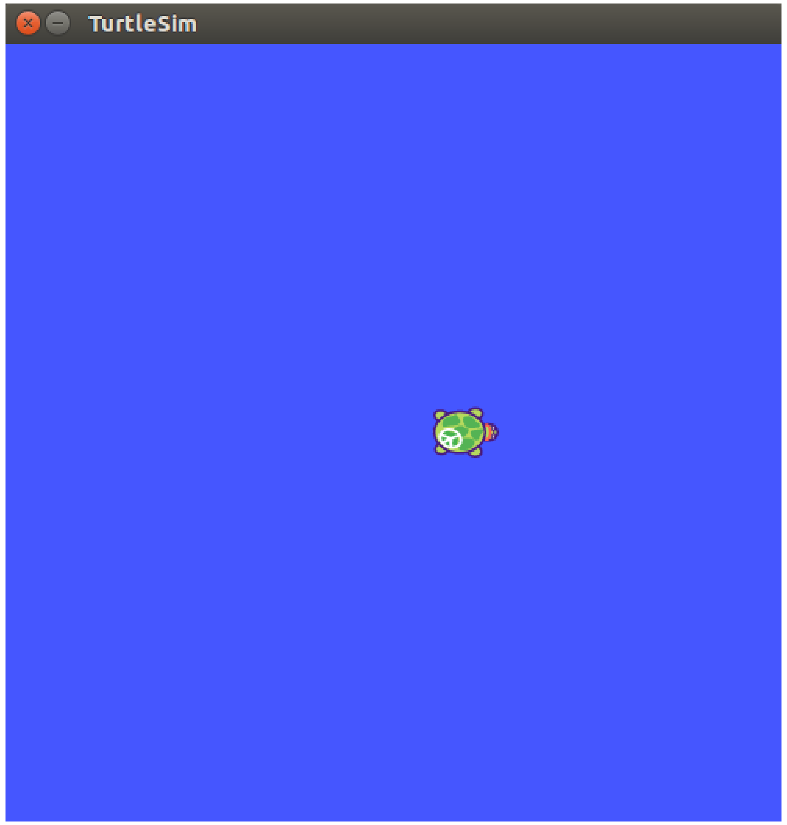
Portland State UNIVERSITY

How to call Actions?

- Run Turtlesim

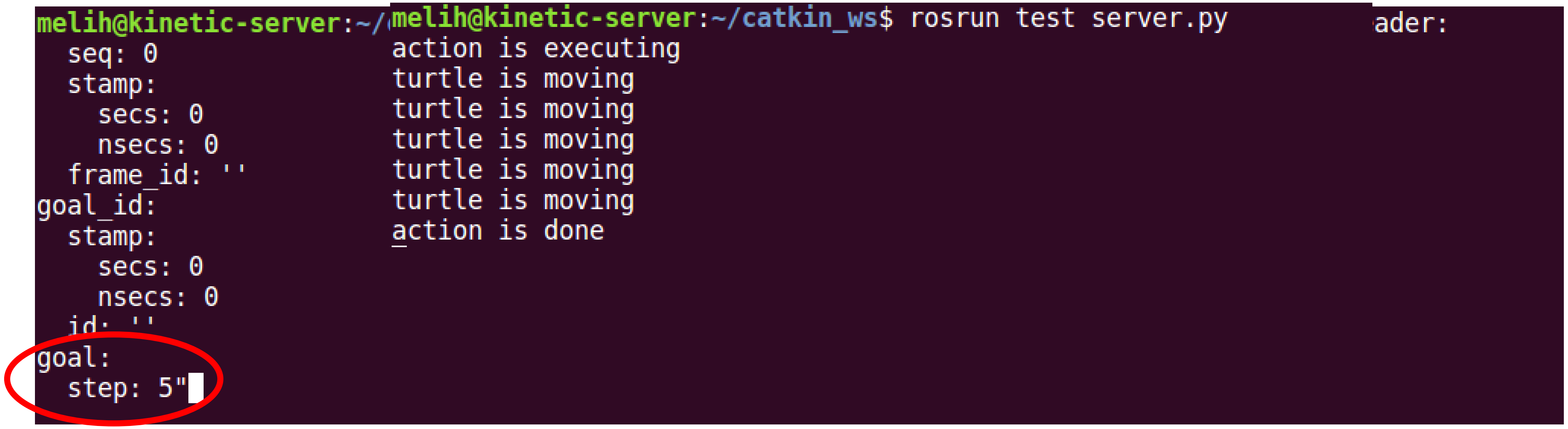


```
melih@kinetic-server:~/catkin_ws$ rosrun turtlesim turtlesim_node
[ INFO] [1542829946.069088646]: Starting turtlesim with node name /turtlesim
[ INFO] [1542829946.076435202]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```



How to call Actions?

- Use rostopic pub to call an action
- Set the goal



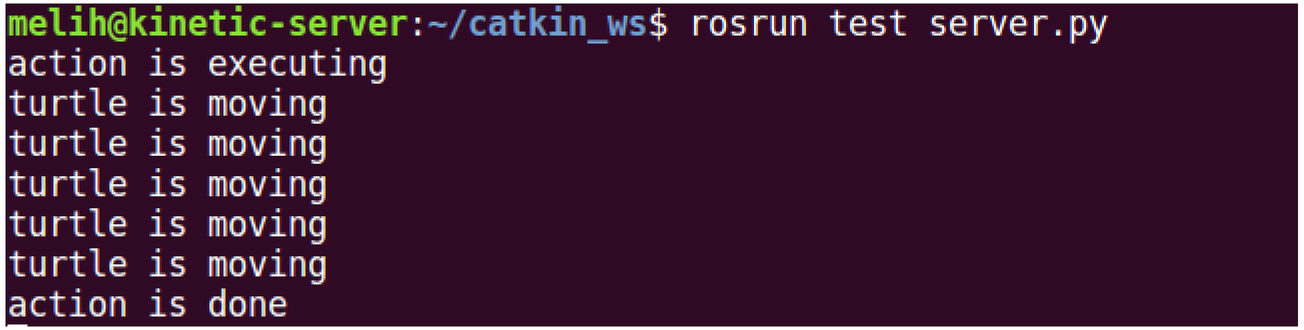
```
melih@kinetic-server:~/
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
goal_id:
  stamp:
    secs: 0
    nsecs: 0
  id: ''
goal:
  step: 5"
```

```
ader:
```

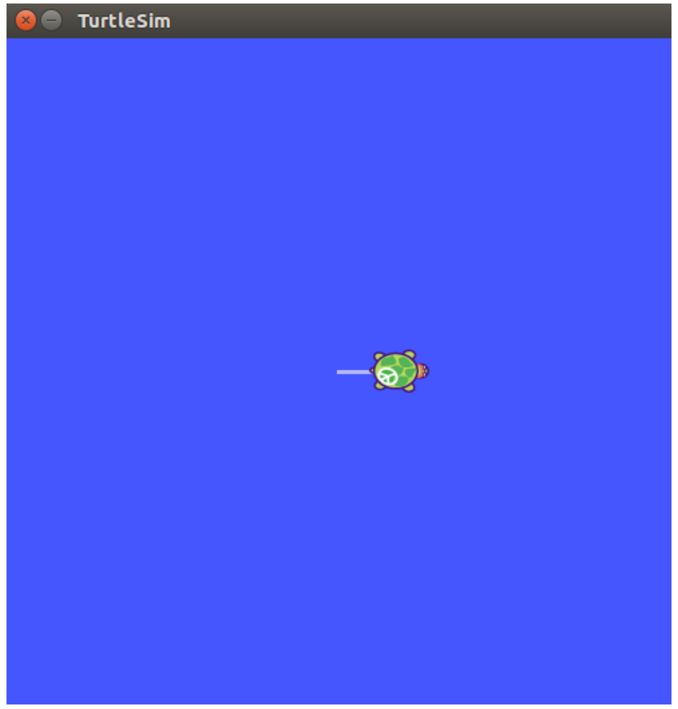
```
melih@kinetic-server:~/catkin_ws$ rosrun test server.py
action is executing
turtle is moving
turtle is moving
turtle is moving
turtle is moving
turtle is moving
action is done
```

How to call Actions?

- Action starts executing
- When it is done it will return the result
- Turtle will start moving



```
melih@kinetic-server:~/catkin_ws$ rosrun test server.py
action is executing
turtle is moving
turtle is moving
turtle is moving
turtle is moving
turtle is moving
action is done
```



ROS Action Client

```
#! /usr/bin/env python

import rospy
import actionlib
import sys
import test.msg

#method to properly shutdown a node
def hook():
        print "shutdown time!"

#velues for different action server states
pending = 0
active = 1
done = 2
warn = 3
error = 4

# counter to print
counter = 0
```

Portland State UNIVERSITY

ROS Action Client

```python
# initilize our client node
rospy.init_node('test_action_client')

# creates the SimpleActionClient, passing the type of the action
client = actionlib.SimpleActionClient('TAS', test.msg.my_actionAction)

# waits until the action server has started up and started
# listening for goals.
client.wait_for_server()

# creates a goal to send to the action server.
goal = test.msg.my_actionGoal(step=10)

# sends the goal to the action server.
client.send_goal(goal)

# waits for the server to finish performing the action.
# client.wait_for_result()

# prints out the result of executing the action
state = client.get_state()

# set the rate to 10 times per second
r = rospy.Rate(10)
```

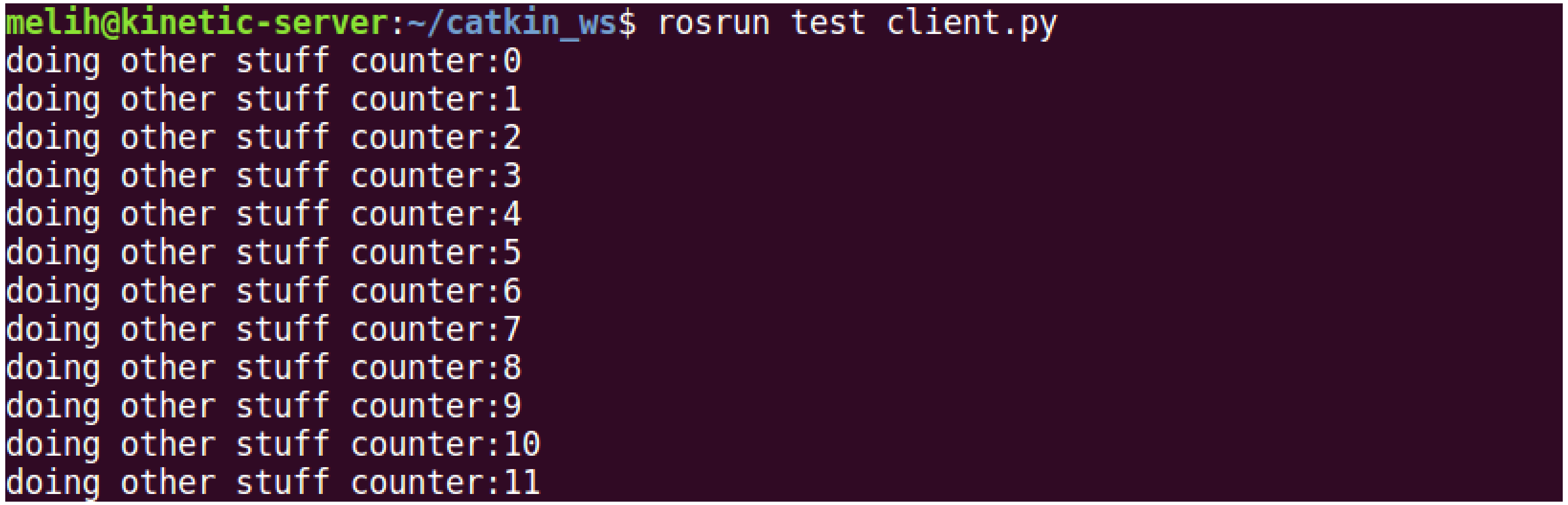
ROS Action Client

```python
# do some other things as the action is running
while state < done:
        print("doing other stuff counter:" + str(counter))
        counter = counter + 1

        # get the state of the action
        state = client.get_state()

        # if the result is published show the result
        if None != client.get_result():
                print(client.get_result())

        #sleep for 100 ms
        r.sleep()

print ("client_done")
# shutdown the node
rospy.on_shutdown(hook)
```

Portland State UNIVERSITY

ROS Action Client Test

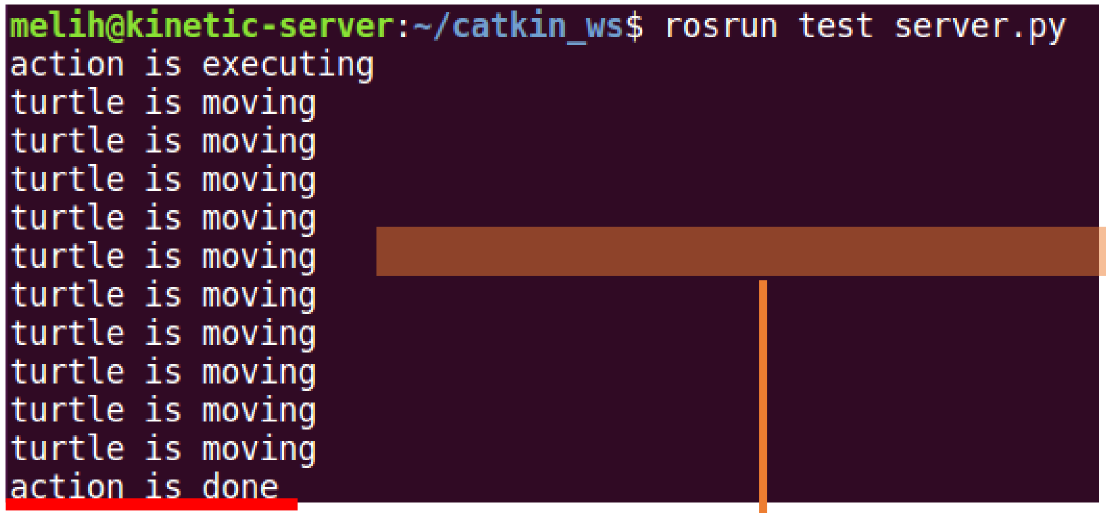
- Run action client node



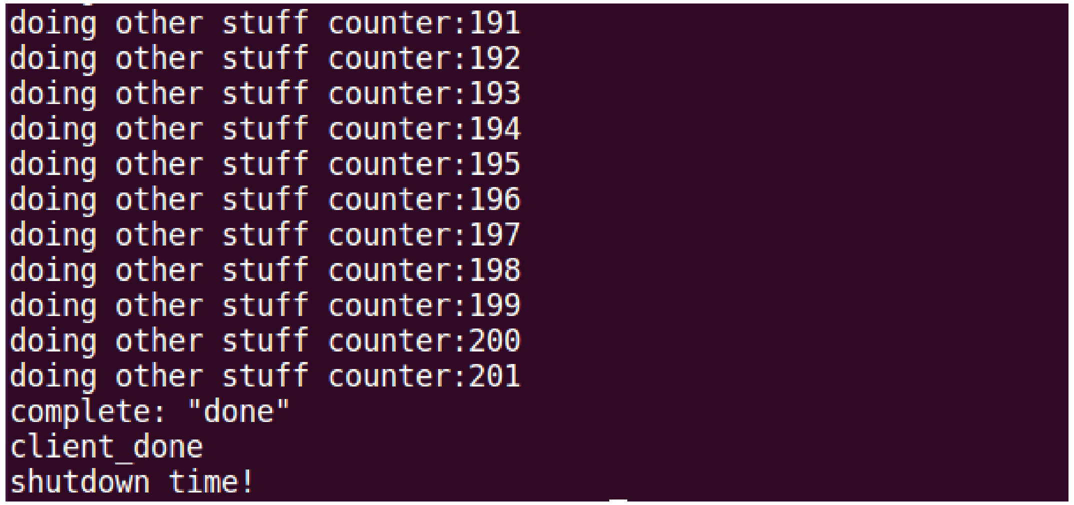
```
melih@kinetic-server:~/catkin_ws$ rosrun test client.py
doing other stuff counter:0
doing other stuff counter:1
doing other stuff counter:2
doing other stuff counter:3
doing other stuff counter:4
doing other stuff counter:5
doing other stuff counter:6
doing other stuff counter:7
doing other stuff counter:8
doing other stuff counter:9
doing other stuff counter:10
doing other stuff counter:11
```

ROS Action Client Test

- Run action server node
- It will receive the action request and will start executing the action



```
melih@kinetic-server:~/catkin_ws$ rosrun test server.py
action is executing
turtle is moving
turtle is moving
turtle is moving
turtle is moving
turtle is moving
turtle is moving
turtle is moving
turtle is moving
turtle is moving
turtle is moving
action is done
```



```
doing other stuff counter:191
doing other stuff counter:192
doing other stuff counter:193
doing other stuff counter:194
doing other stuff counter:195
doing other stuff counter:196
doing other stuff counter:197
doing other stuff counter:198
doing other stuff counter:199
doing other stuff counter:200
doing other stuff counter:201
complete: "done"
client_done
shutdown time!
```

client.py

- As the action runs, client will be doing some other things
- When action is done result will be return and client will also stop

Let’s Stop Here

Happy Thanksgiving