ECE 478-578
Intelligent Robotics I

PhD. Husnu Melih Erdogan – Electrical & Computer Engineering

herdogan@pdx.edu **Teaching Assistant**

Portland State
UNIVERSITY

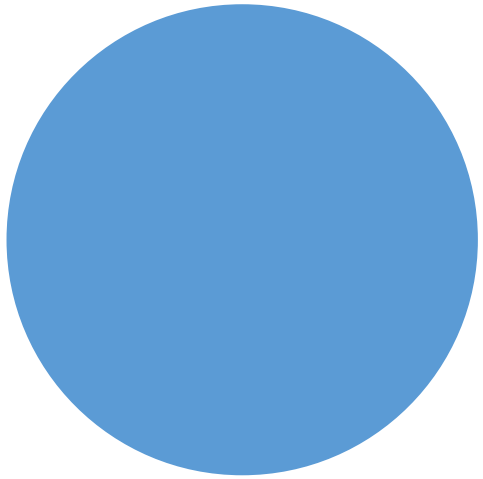
Introduction to ROS Part – 4
&
Spoken Language Interface Technologies

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
  - ROS Messages
  - Speech Recognition
  - Speech Synthesis
  - Google Dialogflow
- **Part 5 - Fuzzy**
  - Amazon Polly
  - ROS Actions
  - Rviz
  - Mapping
  - Localization
  - Fuzzy Logic
  - 2D Multi-Robot Simulator
  - Assignment 4

Portland State UNIVERSITY



ROS
(Robot Operating System)

Portland State
UNIVERSITY

ROS Services

Portland State
UNIVERSITY

ROS Service

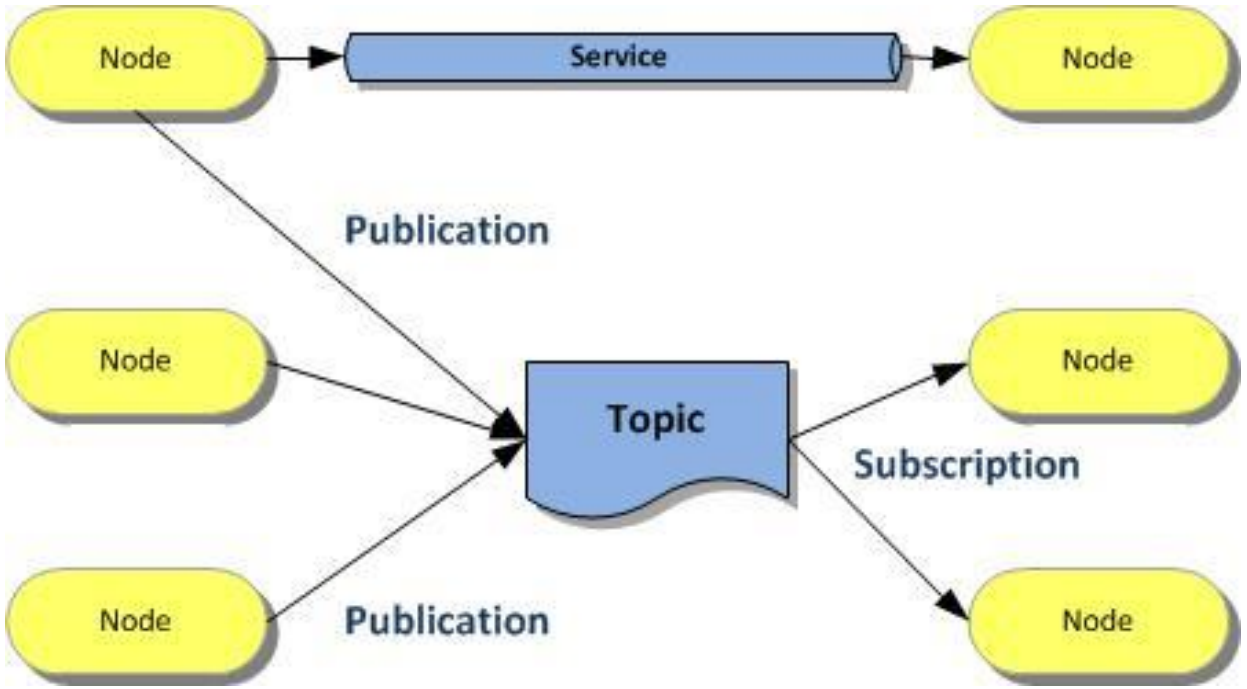
- In some robot applications, a publish/subscribe model will not be enough if it needs a request/response interaction.
- The publish/subscribe model is a kind of one-way transport system and when we work with a distributed system, we might need a request/response kind of interaction.
- We can define a service definition that contains two parts;
  - one is for requests
  - the other is for responses.
- 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.
- The client might need to wait until the server responds. The ROS service interaction is like a remote procedure call.

ROS Service

- The publish/subscribe model is a very flexible.
- However, 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 *Service*
- They are defined by a pair of messages: one for the request and one for the reply.
- Services are defined using srv files, which are compiled into source code by a ROS client library.

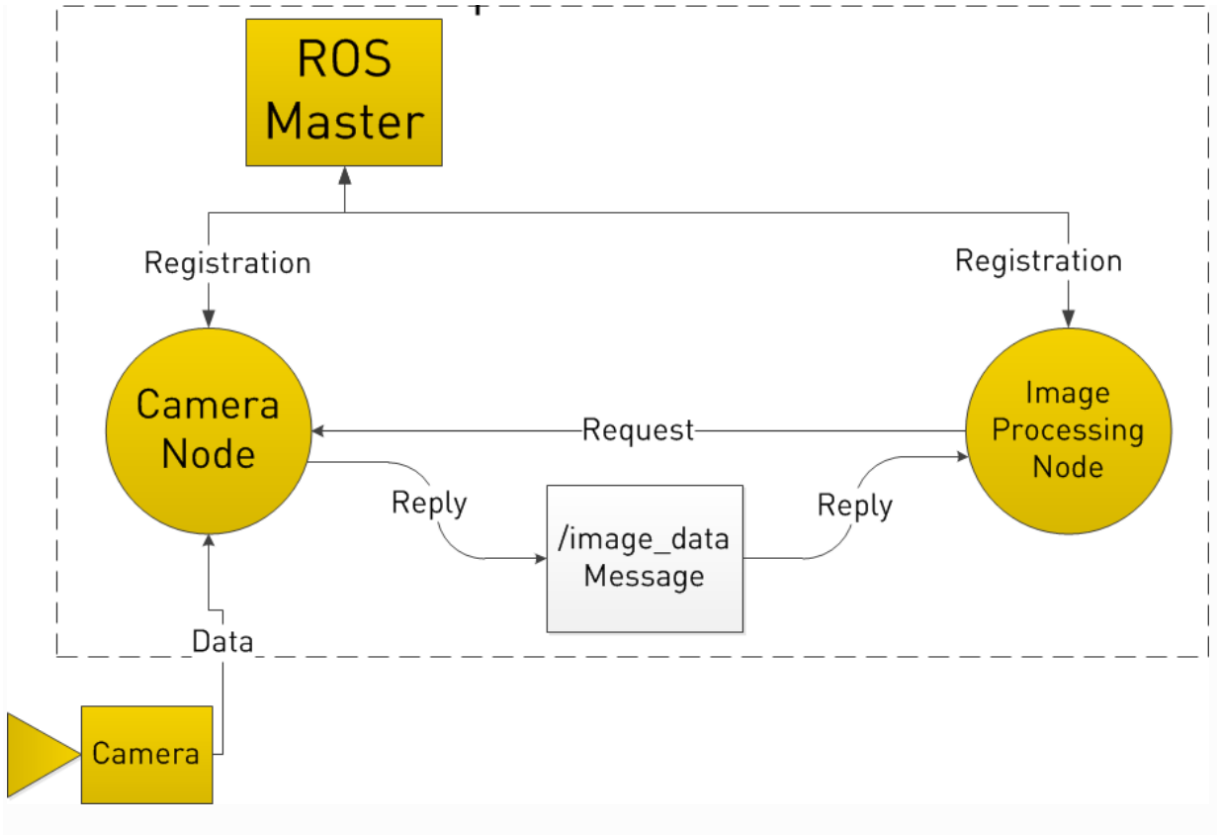
ROS Service

Difference between services and publishing and subscribing to topics



ROS Service

How does ROS service works?

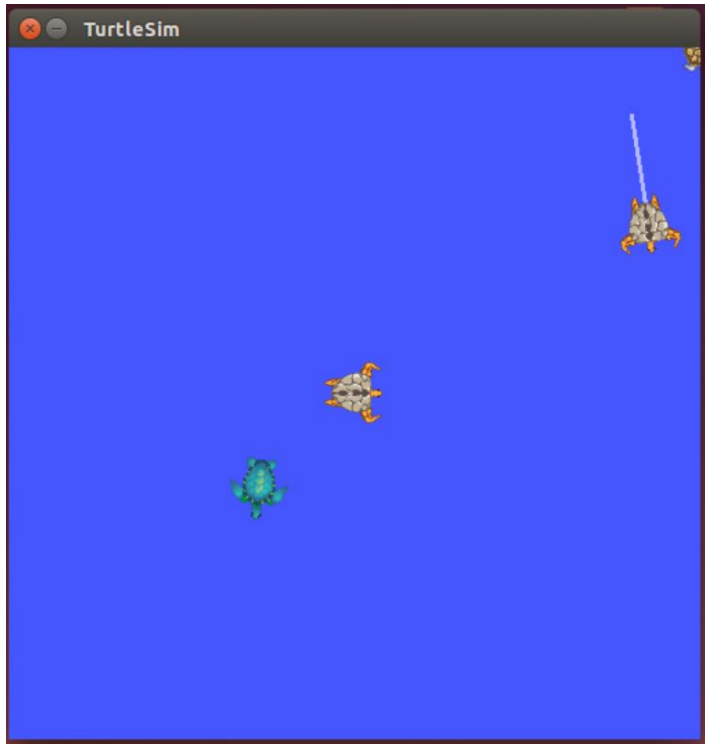


http://www.clearpathrobotics.com/assets/guides/ros/Intro%20to%20the%20Robot%20Operating%20System.html

rosservice command line tool

- $ rosservice **call** /service args: This tool will call the service using the given arguments
- $ rosservice **find** service_type: This command will find services in the given service type
- $ rosservice **info** /service: This will print information about the given service
- $ rosservice **list**: This command will list the active services running on the system
- $ rosservice **type** /service: This command will print the service type of a given service
- $ rosservice **uri** /service: This tool will print the service ROSRPC URI

Ros Service Example



`clear` (std_srvs/Empty)
Clears the turtlesim background and sets the color to the value of the background parameters.

`reset` (std_srvs/Empty)
Resets the turtlesim to the start configuration and sets the background color to the value of the background.

`kill` (turtlesim/Kill)
Kills a turtle by name.

`spawn` (turtlesim/Spawn)
Spawns a turtle at (x, y, theta) and returns the name of the turtle. Also will take name for argument but will fail if a duplicate name.

`turtleX/set_pen` (turtlesim/SetPen)
Sets the pen's color (r g b), width (width), and turns the pen on and off (off).

`turtleX/teleport_absolute` (turtlesim/TeleportAbsolute)
Teleports the turtleX to (x, y, theta).

`turtleX/teleport_relative` (turtlesim/TeleportRelative)
Teleports the turtleX a linear and angular distance from the turtles current position.

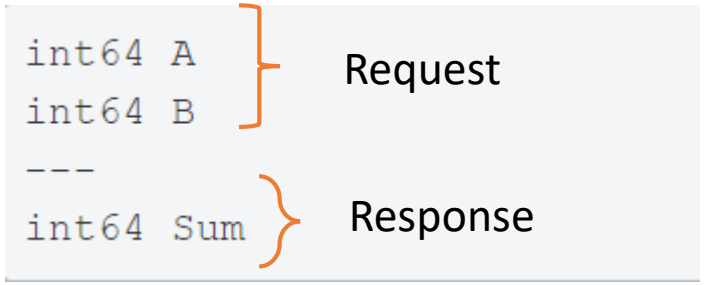
```
melih@melih-VirtualBox:~/catkin_ws$ rosservice call /spawn 10 10 10 melih
```

```
melih@melih-VirtualBox:~/catkin_ws$ rosservice call /turtle1/set_pen 10 10 10 30 0
```

Portland State UNIVERSITY

ROS srv

- srv: an srv file describes a service.
- It is composed of two parts: a request and a response.
- srv files are just like msg files, except they contain two parts: a request and a response. The two parts are separated by a '---' line. Here is an example of a srv file:



How to create ROS srv?

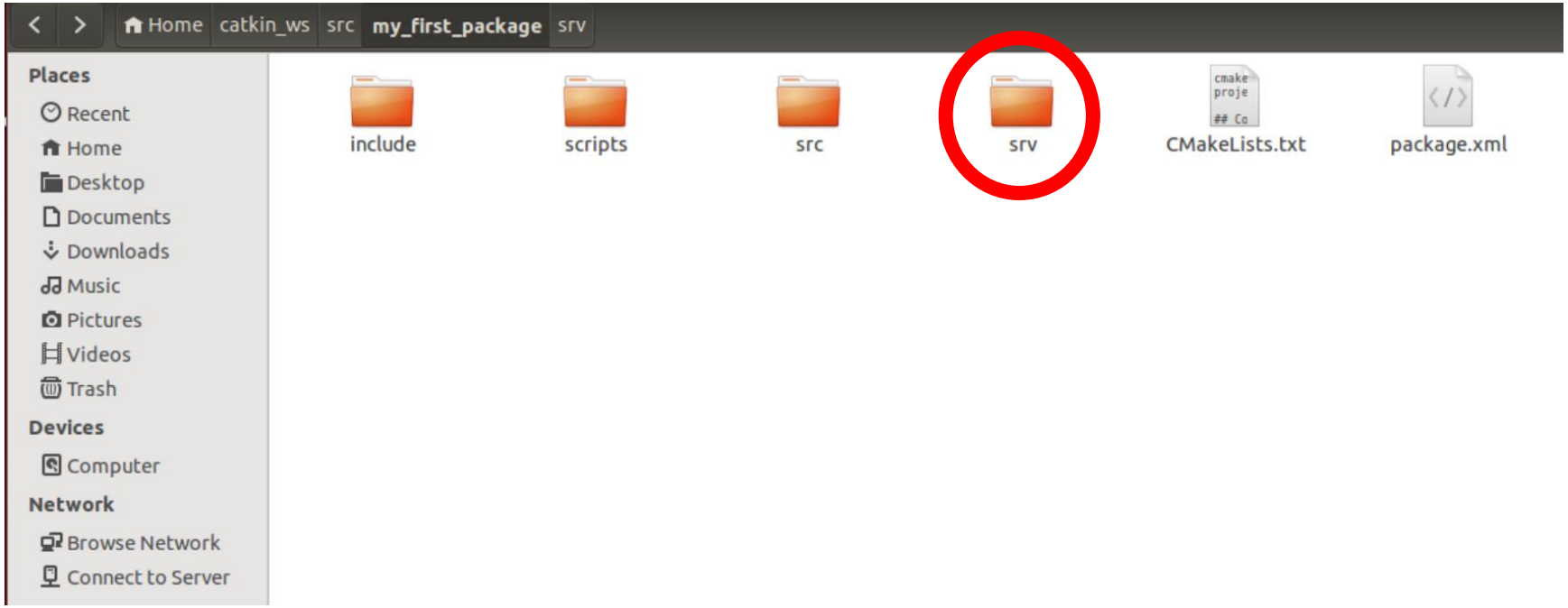
- Defining a Service
- Implement the Service
- Service Server
- Use the service
- Service Client

How to define a ROS srv?

- It is done in a service definition file
- It is .srv file
- It is just like message definition files. Maybe just a little bit more complicated.
- It has both input types and output types

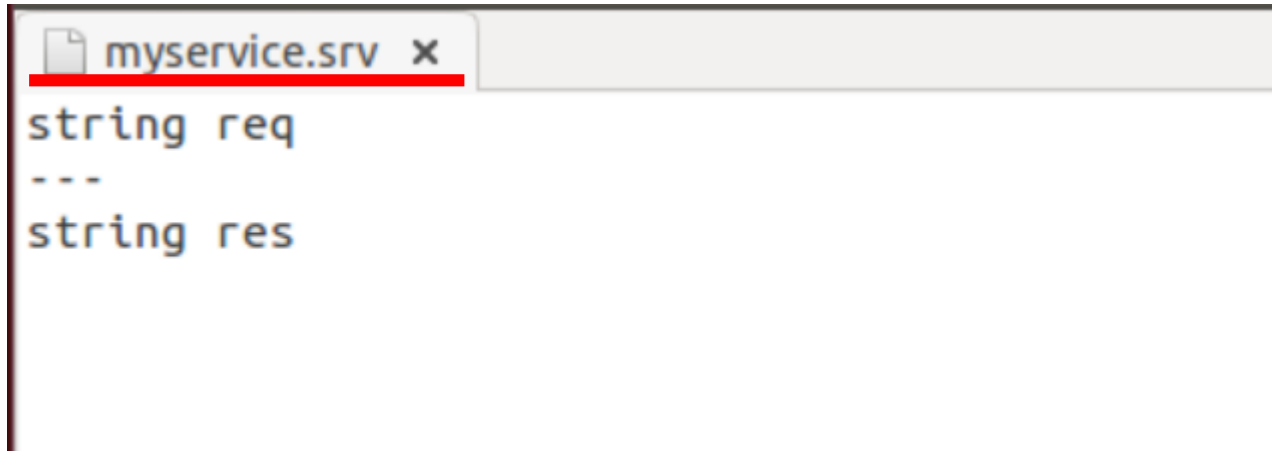
How to define a ROS srv?

- It is not a requirement, but traditionally these files are created in srv directory in the main package directory.



How to define a ROS srv?

- There are three dashes between input and output definitions.



How to create ROS srv?

- We need to run catkin_make to create the code and class definitions that we will actually use when interacting with services.
- In order to get catkin_make build this code, we need to modify some files.
  - **CMakeList.txt**
  - **Package.xml**

How to create ROS srv?

- We need to make sure massage generation package is included.

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(my_first_package)

## 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
  roscpp
  rospy
  std_msgs
  message_generation
)
```

How to create ROS srv?

- We need to tell catkin make which service definition files are going to be compiled.

CMakeLists.txt

```
##   * uncomment the add_*_files sections below as needed
##     and list every .msg/.srv/.action file to be processed
##   * uncomment the generate_messages entry below
##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
 add_service_files(
   FILES
   myservice.srv
 )
```

How to create ROS srv?

- We must make sure the dependencies for the service definition files is declared.

CMakeLists.txt

```
)

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

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

How to create ROS srv?

Our new dependencies on rospy and the message system are needed during build time and runtime.

package.xml

```
<!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
<license>TODO</license>


<!-- Url tags are optional, but multiple are allowed, one per tag -->
<!-- Optional attribute type can be: website, bugtracker, or repository -->
<!-- Example: -->
<!-- <url type="website">http://wiki.ros.org/my_first_package</url> -->


<!-- Author tags are optional, multiple are allowed, one per tag -->
<!-- Authors do not have to be maintainers, but could be -->
<!-- Example: -->
<!-- <author email="jane.doe@example.com">Jane Doe</author> -->


<!-- The *depend tags are used to specify dependencies -->
<!-- Dependencies can be catkin packages or system dependencies -->
<!-- Examples: -->
<!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
<!--   <depend>roscpp</depend> -->
<!--   Note that this is equivalent to the following: -->
<!--   <build_depend>roscpp</build_depend> -->
<!--   <exec_depend>roscpp</exec_depend> -->
<!-- Use build_depend for packages you need at compile time: -->
<build_depend>message_generation</build_depend>
<!-- Use build_export_depend for packages you need in order to build against this package: -->
<!--   <build_export_depend>message_generation</build_export_depend> -->
<!-- Use buildtool_depend for build tool packages: -->
<!-- Use exec_depend for packages you need at runtime: -->
<exec_depend>message_runtime</exec_depend>
<!-- Use test_depend for packages you need only for testing: -->
<!--   <test_depend>gtest</test_depend> -->
```

Portland State UNIVERSITY

ROS Service Server

server.py

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

from my_first_package.srv import myservice
import rospy

def handle_service(data):
    a = str(data.req) + " 1234"
    return a

def add_numbers():
    rospy.init_node('service_server')
    s = rospy.Service('add_numbers_to_my_text', myservice, handle_service)
    rospy.spin()

if __name__ == "__main__":
    add_numbers()
```

ROS Service Test

```
melih@melih-VirtualBox: ~/catkin_ws
melih@melih-VirtualBox:~/catkin_ws$ rossrv show myservice
[my_first_package/myservice]:
string req
---
string res
```

```
melih@melih-VirtualBox: ~/catkin_ws
melih@melih-VirtualBox:~/catkin_ws$ rosservice list
/add_numbers_to_my_text
/rosout/get_loggers
/rosout/set_logger_level
/service_server/get_loggers
/service_server/set_logger_level
melih@melih-VirtualBox:~/catkin_ws$
```

ROS Service Server Test

Portland State UNIVERSITY

ROS Service Client

client.py

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

import rospy
from std_msgs.msg import String
from my_first_package.srv import *

#create a handle for calling the service
handle = rospy.ServiceProxy('add_numbers_to_my_text', myservice)

def callback(data):
    # call a service
    response = handle(data.data).res
    print(response)

def listener():
    rospy.init_node('test', anonymous=True)
    #blocks the node until the service is available
    rospy.wait_for_service('add_numbers_to_my_text')
    rospy.Subscriber('input', String, callback)
    rospy.spin()

if __name__ == '__main__':
    listener()
```

ROS Service Client

```
melih@melih-VirtualBox: ~/catkin_ws
melih@melih-VirtualBox:~/catkin_ws$ rosrun my_first_package server.py
```

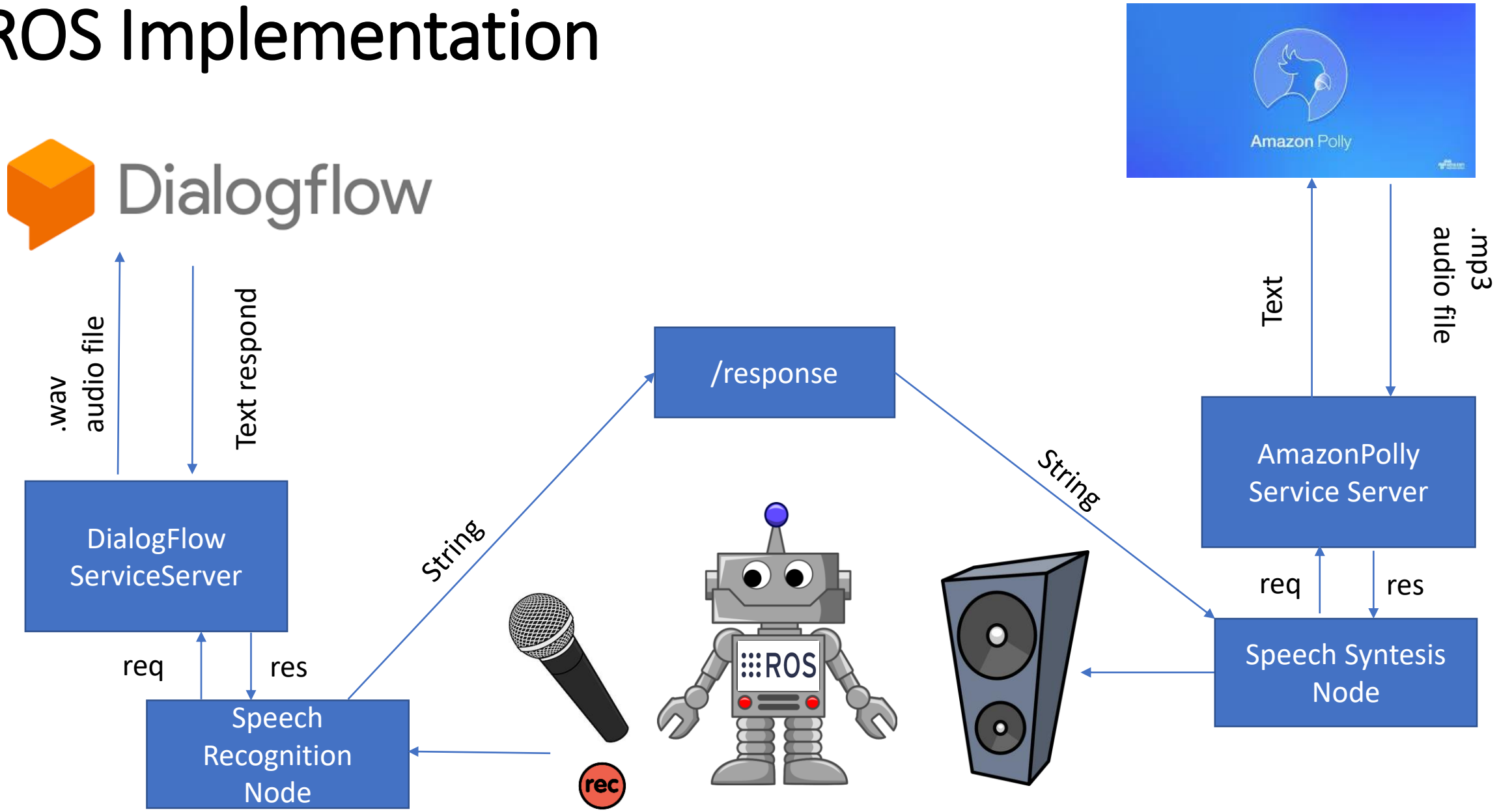
```
melih@melih-VirtualBox: ~/catkin_ws
melih@melih-VirtualBox:~/catkin_ws$ rosrun my_first_package client.py
```

```
melih@melih-VirtualBox: ~/catkin_ws
melih@melih-VirtualBox:~/catkin_ws$ rostopic pub -1 /input std_msgs/String melih
publishing and latching message for 3.0 seconds
```

```
melih@melih-VirtualBox: ~/catkin_ws
melih@melih-VirtualBox:~/catkin_ws$ rosrun my_first_package client.py
melih 1234
```

Portland State UNIVERSITY

ROS Implementation



Portland State UNIVERSITY

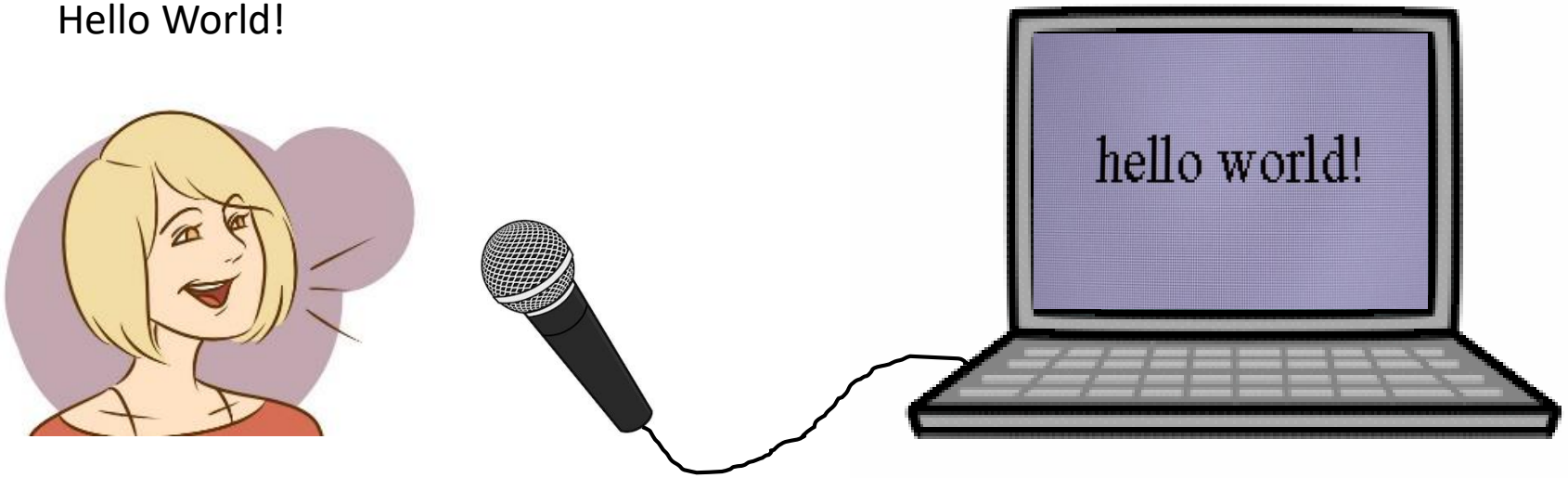
Speech Recognition and Speech Synthesis

Portland State
UNIVERSITY

What is Speech Recognition?

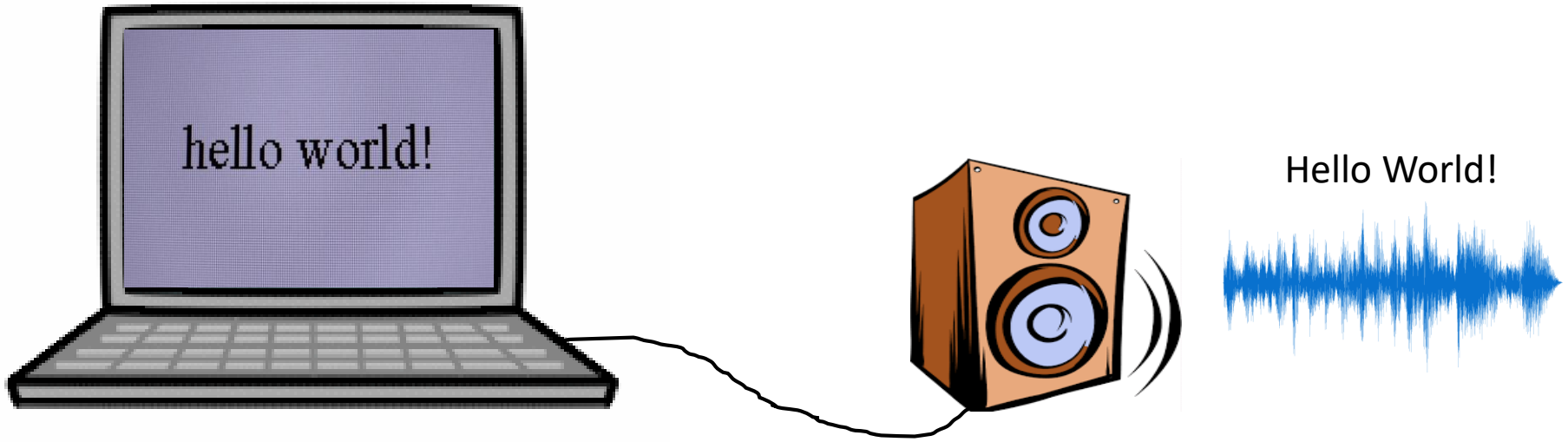
- The task of **speech recognition** is to convert speech into a sequence of words by a computer program

Hello World!



What is Speech Syntesis?

- **Speech synthesis** is artificial simulation of human speech which is created by a computer or other device.



Where are they used?

- Computers
- Telemarketing
- Smart Phones
- Emergencies
- Phone Call Routing
- Smart Homes
- Cars
- Robots
- Etc.

The Current State of Art in Speech Technologies

Speech Applications and Technologies

- Text
- Voice
- Ink etc..

Input Technologies

- Google Cloud Voice
- Dialogflow
- Amazon Polly

Apps

- Web Application
- Software
- Your Robot

Cloud Resources

Output Technologies

- Text
- Voice
- Ink etc..

Local Resources

- Espeak
- pocketsphinx

Portland State UNIVERSITY

Dialog Management

- Controlling the interchange of information between users and application
- Three dialog styles
  - **Human-directed** conversational dialogs
    - User asks a question or speaks a command and the computer responds.
  - **Application-directed** conversational dialogs
    - Application asks questions to get answers and instructions from a user.
  - **Mixed-initiative** dialogs
    - User and application take turns driving the conversations.

Three Dialog Styles

Application Directed

Robot: What month?

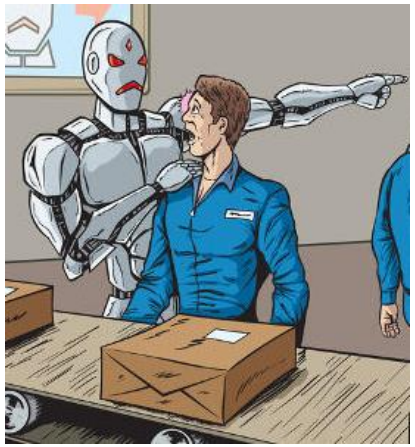
Human: February

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

Portland State UNIVERSITY

Speech Recognition

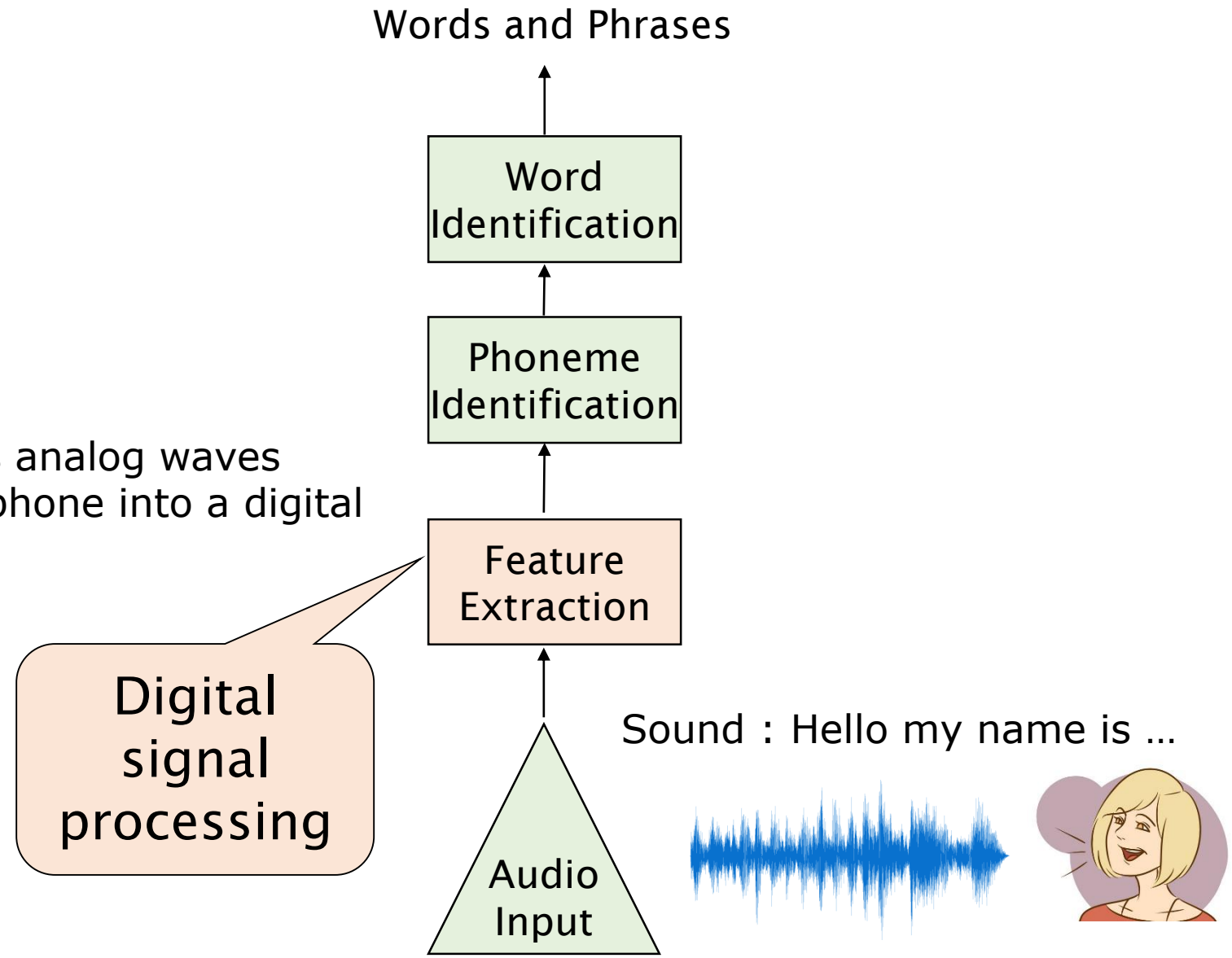
Portland State
UNIVERSITY

Speech Recognition

ASR – Automated Speech Recognition

- Advantages
  - User does not convert choices to a digit
- Disadvantages
  - Occasional failure to recognize what user said
    - Noise
    - Accent
    - Long Delay
  - Some times time-consuming dialogs
    - Users may interrupt prompts by “barge-in”

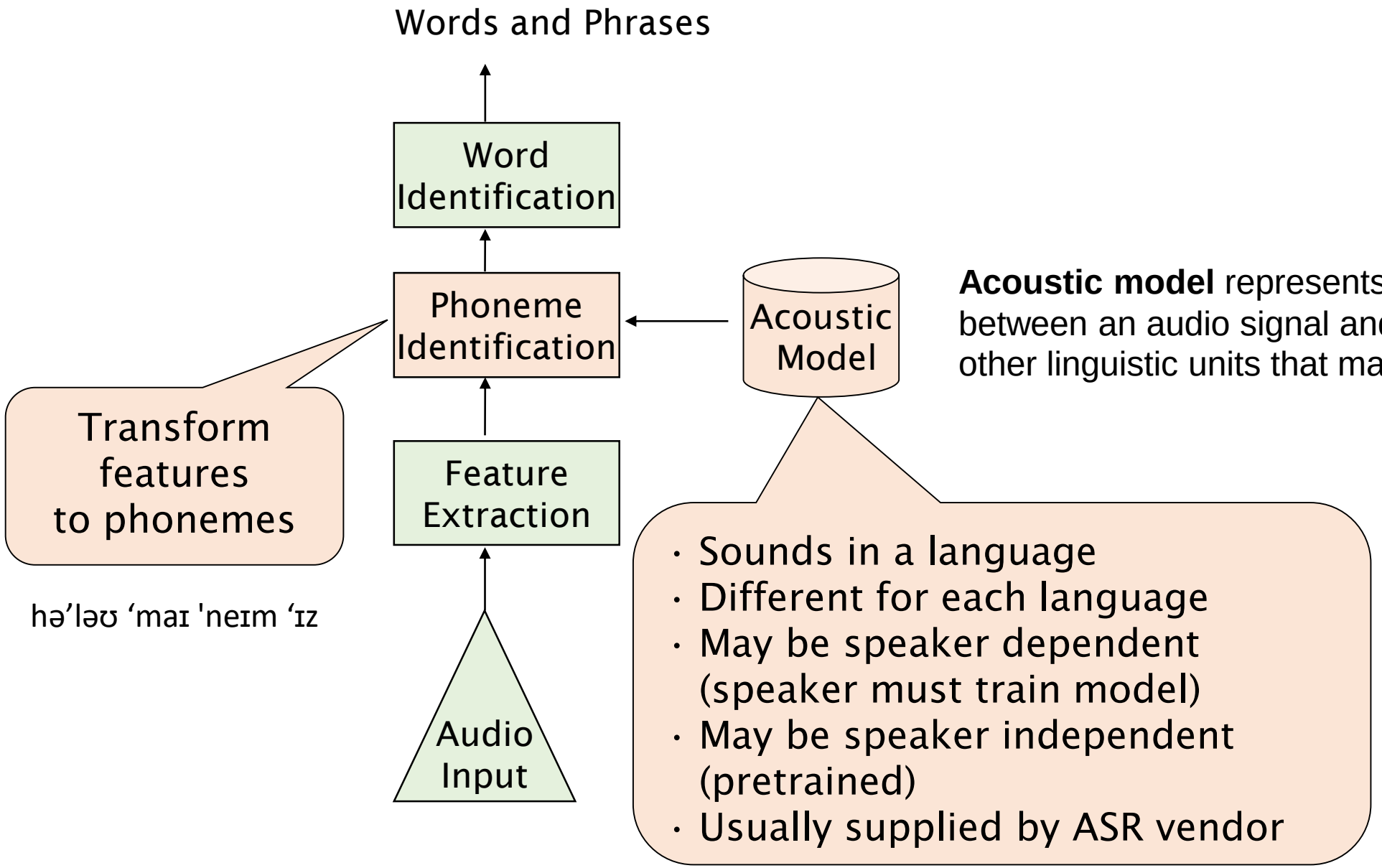
How Speech Recognition Works



- Computer converts analog waves spoken into a microphone into a digital format.
- Noise reduction

Portland State UNIVERSITY

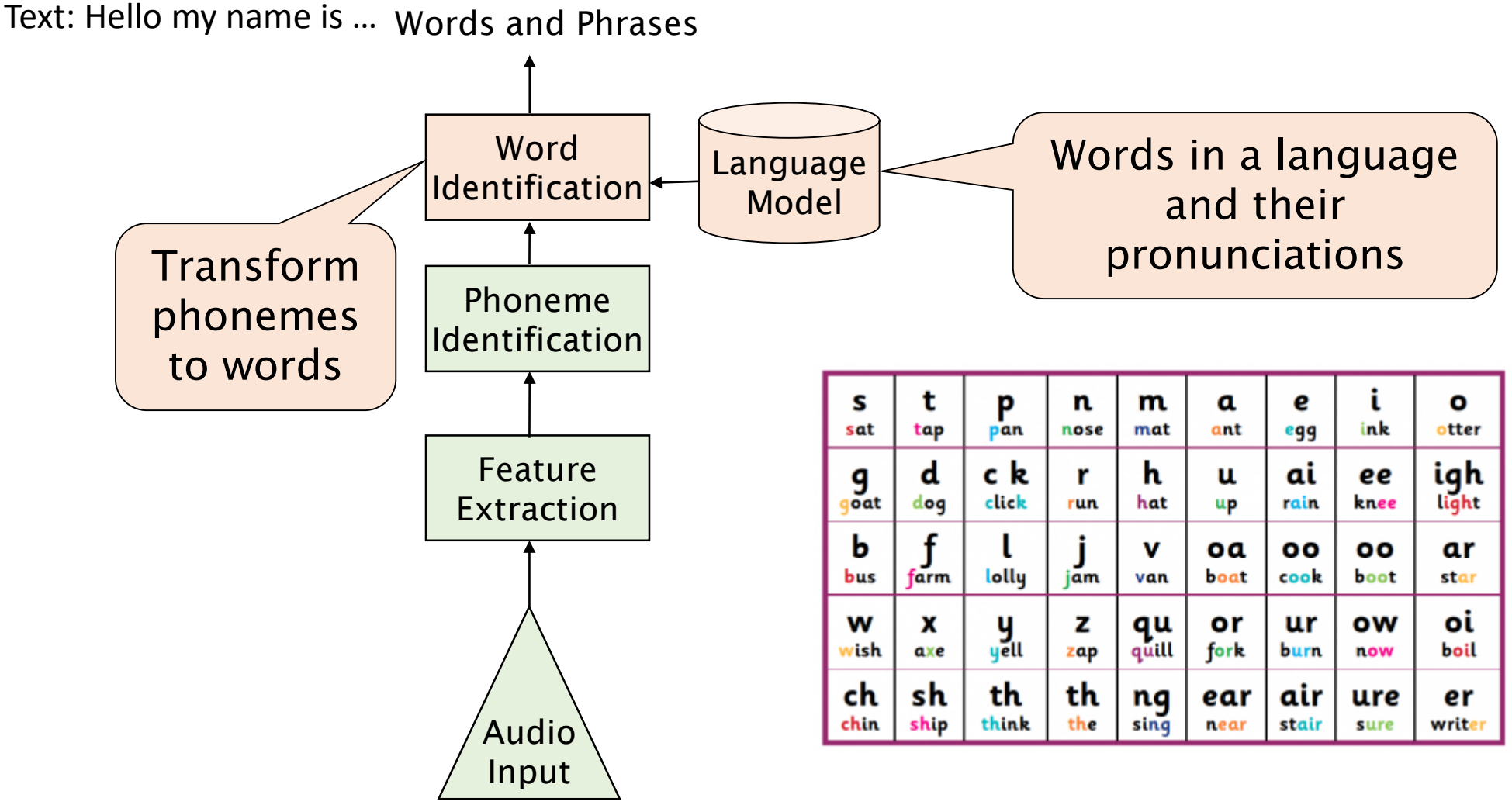
How Speech Recognition Works



Acoustic model represents the relationship between an audio signal and the phonemes or other linguistic units that make up speech.

Portland State UNIVERSITY

How Speech Recognition Works



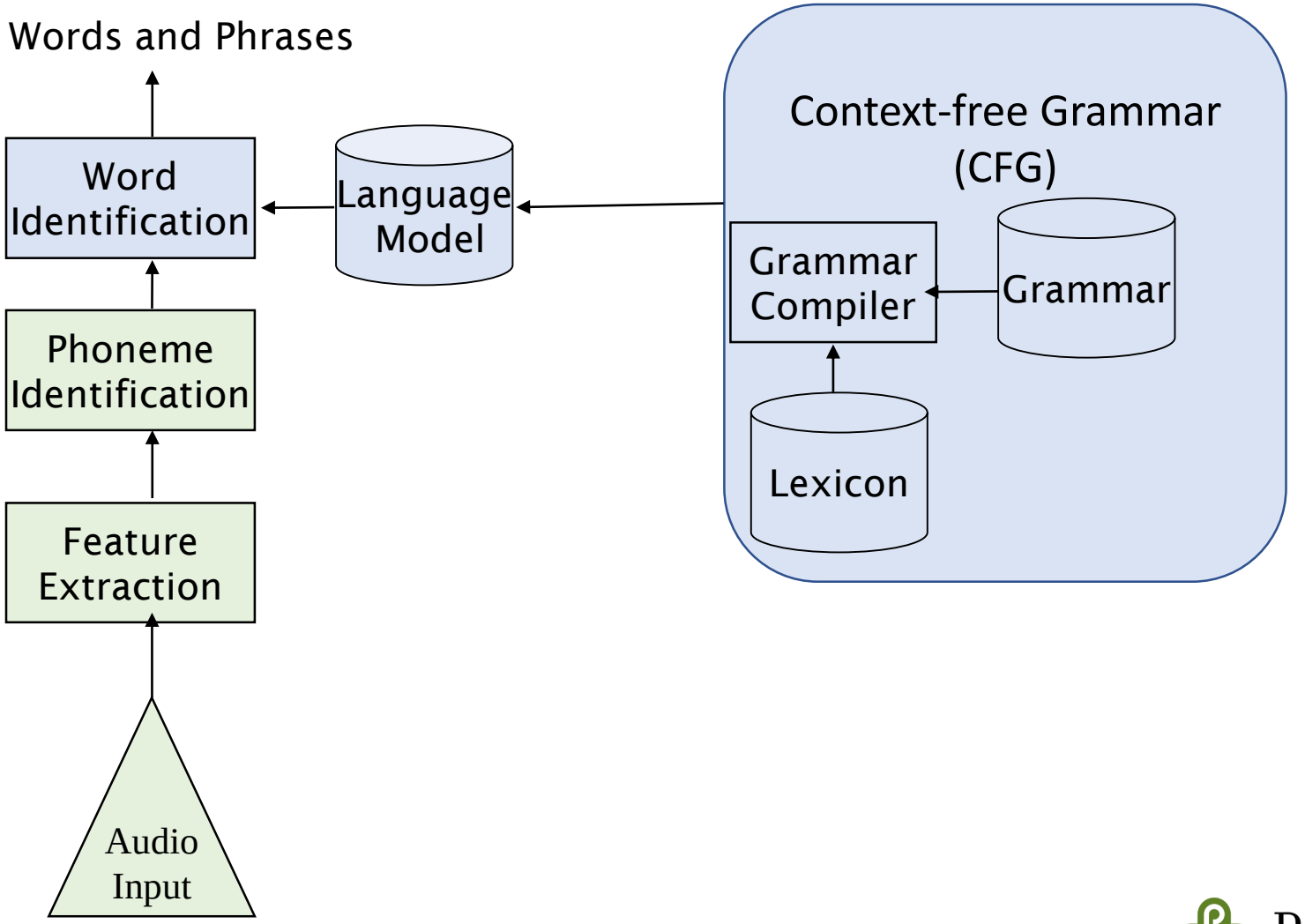
s sat	t tap	p pan	n nose	m mat	a ant	e egg	i ink	o otter
g goat	d dog	c k click	r run	h hat	u up	ai rain	ee knee	igh light
b bus	f farm	l lolly	j jam	v van	oa boat	oo cook	oo boot	ar star
w wish	x axe	y yell	z zap	qu quill	or fork	ur burn	ow now	oi boil
ch chin	sh ship	th think	th the	ng sing	ear near	air stair	ure sure	er writer

Portland State UNIVERSITY

Speech Recognition

- There are two common methods
- Grammar-based
  - Developer specifies words to be recognized
- Statistical Language Models
  - Developer records and tags phrases

Grammar-based Speech Recognition



Where Are Grammars Used?

- Interactive Response Systems (IVR)
  - Automated telephone agents
- Each step may use a different grammar
  - Grammar defines only the words which the user may speak during a step
  - Application developers specify grammars for each step
- The same grammar may be reused in multiple applications

Grammar Rules Example

- In order to create rules XML format is used

```
<grammar
type = "application/grammar+xml"
root = "request"
mode = "voice">
    <rule id = "request">
        <ruleref uri = "#color"/>
        <ruleref uri = "#size"/>
    </rule>
```

```
<rule id = "color">
    <one-of>
        <item> red </item>
        <item> green </item>
        <item> blue </item>
    </one-of>
</rule>

<rule id = "size">
 <one-of>
        <item> small </item>
        <item> medium </item>
        <item> large </item>
    </one-of>
</rule>
```

Portland State UNIVERSITY

Statistical Language Model-based Recognition Technologies

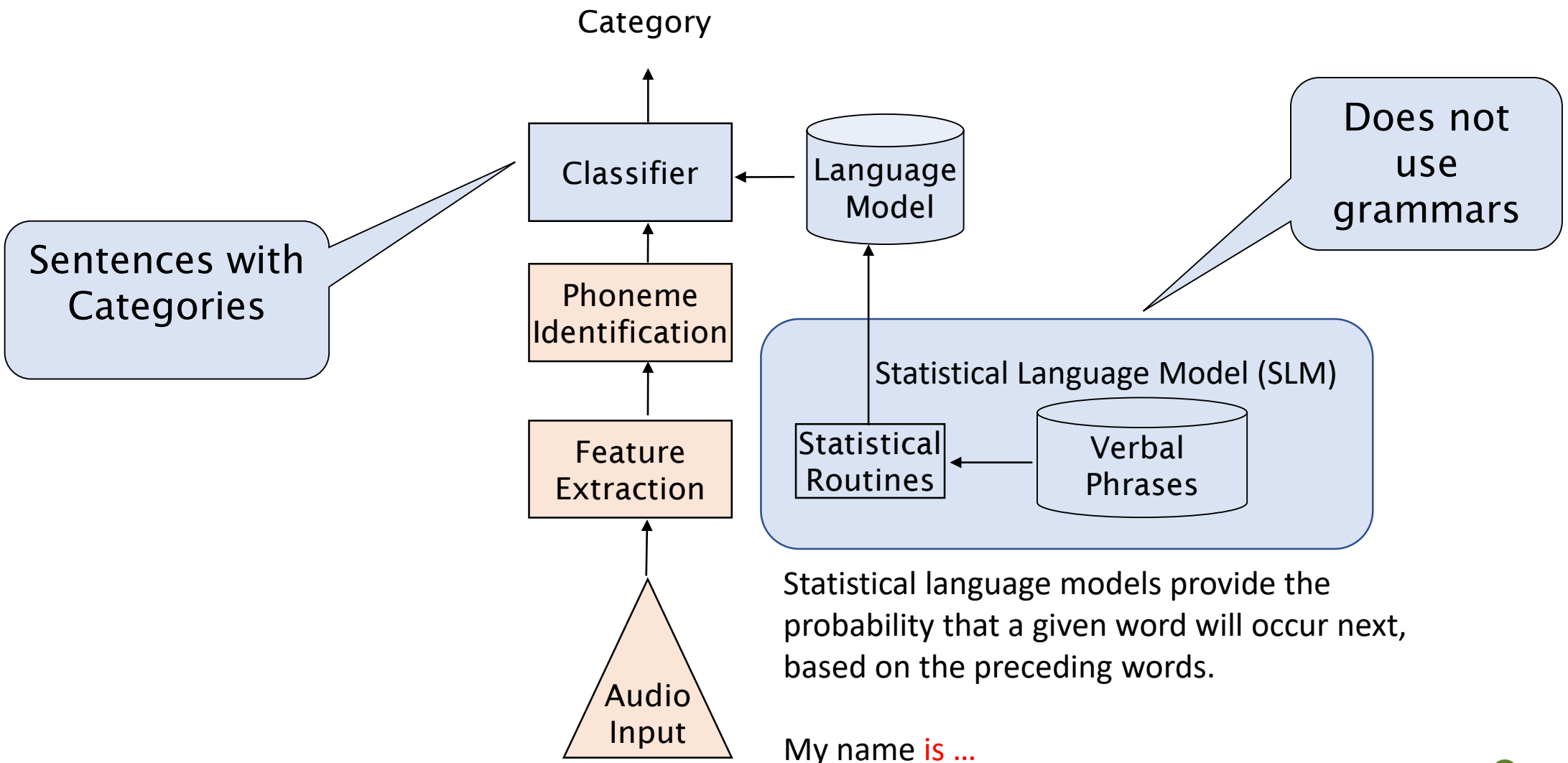
- Call Routing
- Speaker Identification
- Dictation
- Speaker emotion
- Voice pitch
- Age Detection
- Gender Detection
- Intoxication Detection
- Stress Detection
- Medical conditions (e.g., sleep apnea)

Example Verbal Phrases with Label

• “I have a problem with my bill”	accounting
• “Where is my order?”	shipping
• “My gadget arrived broken”	customer service
• “I need to return my gadget”	shipping
• “My statement is wrong”	accounting
• “I want a refund”	accounting

Label thousands of verbal phrases

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 …

Portland State UNIVERSITY

Grammars vs. Statistical Language Models

Statistical Language Models (SLMs)

- Data-driven
- High-accuracy
- Complex to assemble
- Natural language
- Used for dictation

Context-Free Grammars (CFGs)

- Hand-crafted rules
- Very high-accuracy
- Easy to assemble
- Finite phrases
- Used for
  - Interactive Voice Response (IVR)
  - Command and Control

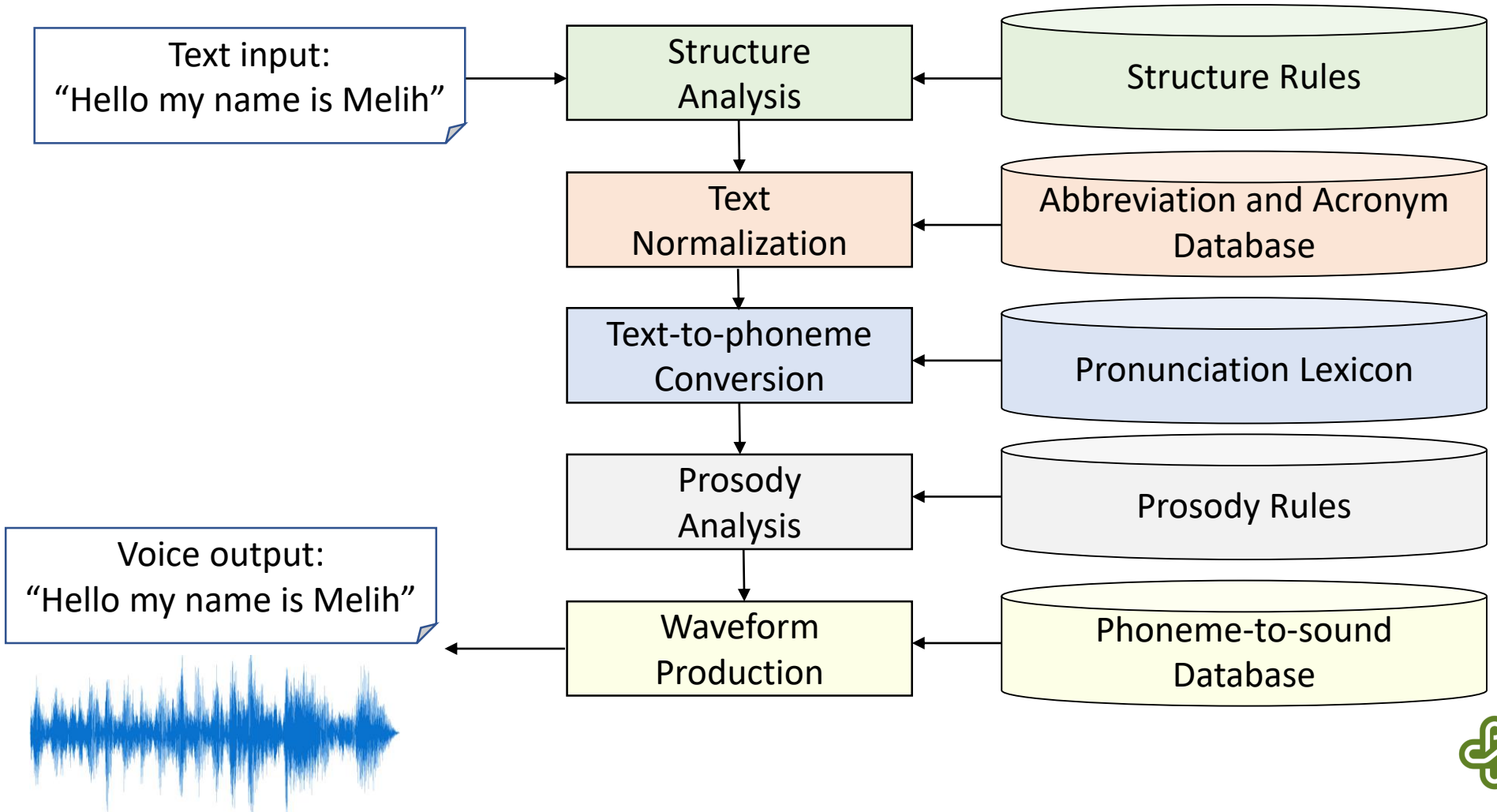
Usage of Statistical Language Model-based Recognition Technologies

- Widely available
  - Call routing
  - Speaker authentication
  - Dictation
- Actively being researched
  - Speaker emotion
  - Voice Pitch
  - Age Detection
  - Gender Detection
  - Intoxication Detection
  - Stress Detection
  - Medical conditions (e.g., sleep apnea)

Speech Syntesis

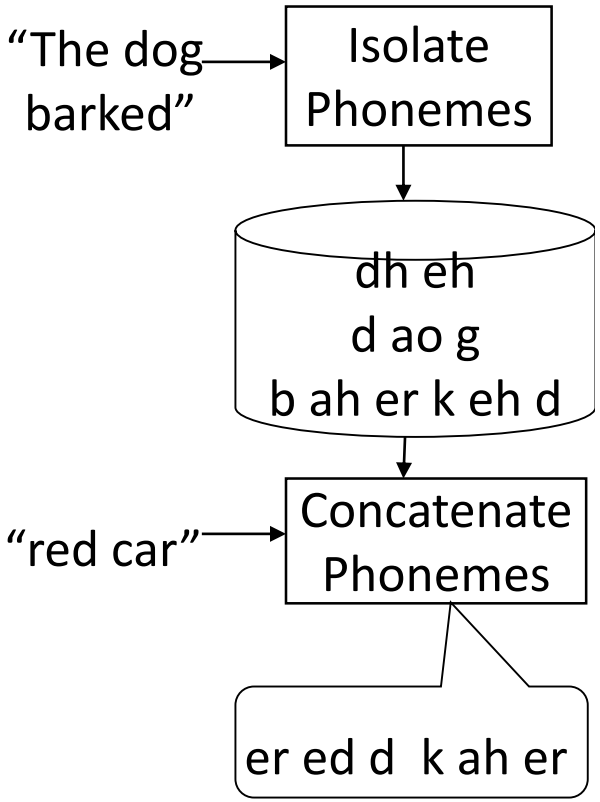
Portland State
UNIVERSITY

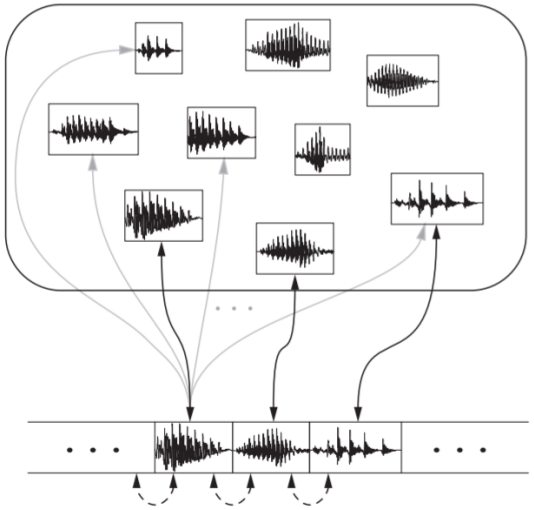
Speech Synthesis
(Text-To-Speech, TTS)

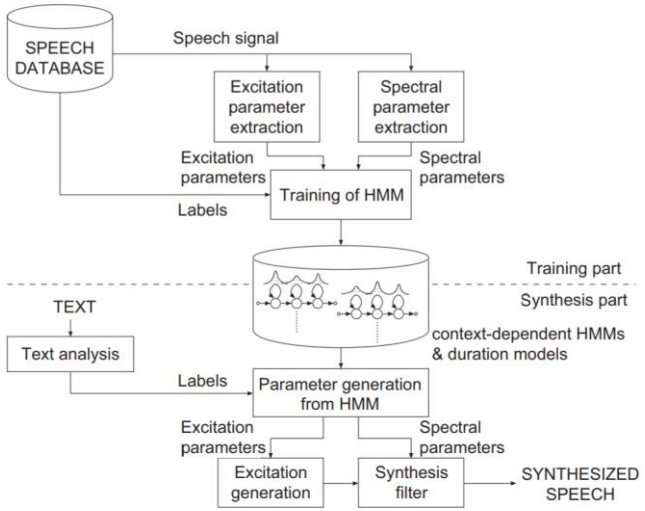


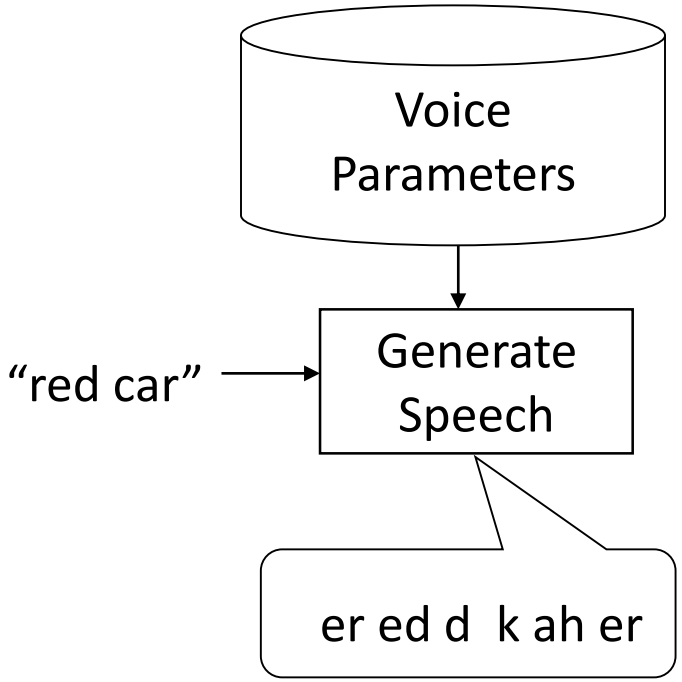
Portland State UNIVERSITY

Concatenated vs. Parameter-based Speech Synthesis







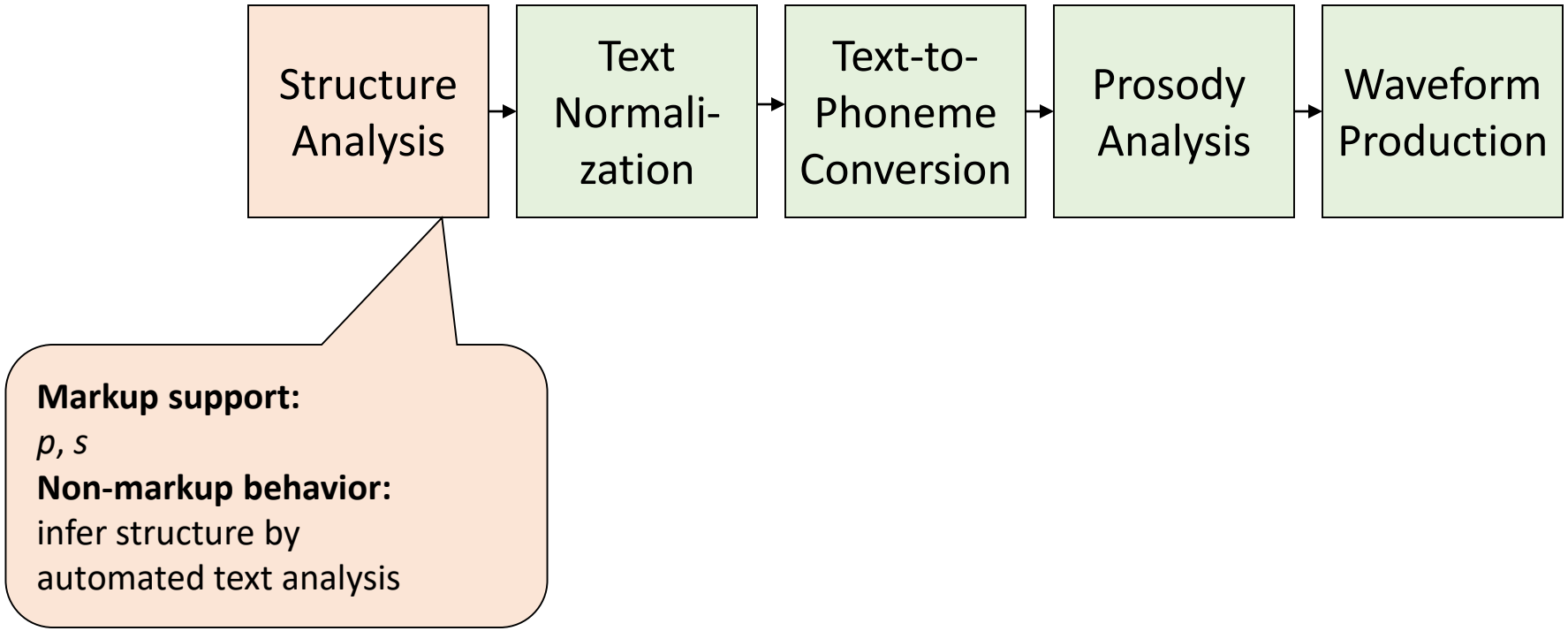


Concatenative synthesis is a technique for synthesizing sounds by concatenating short samples of recorded sound (called *units*).
Need a professional speaker 2-10 hr.

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

Portland State UNIVERSITY

Speech Synthesis Markup Language (SSML)



Dr. Smith lives at 214 Elm Dr. He weights 214 lb. He plays bass guitar. He also likes to fish; last week he caught a 19 lb. bass.

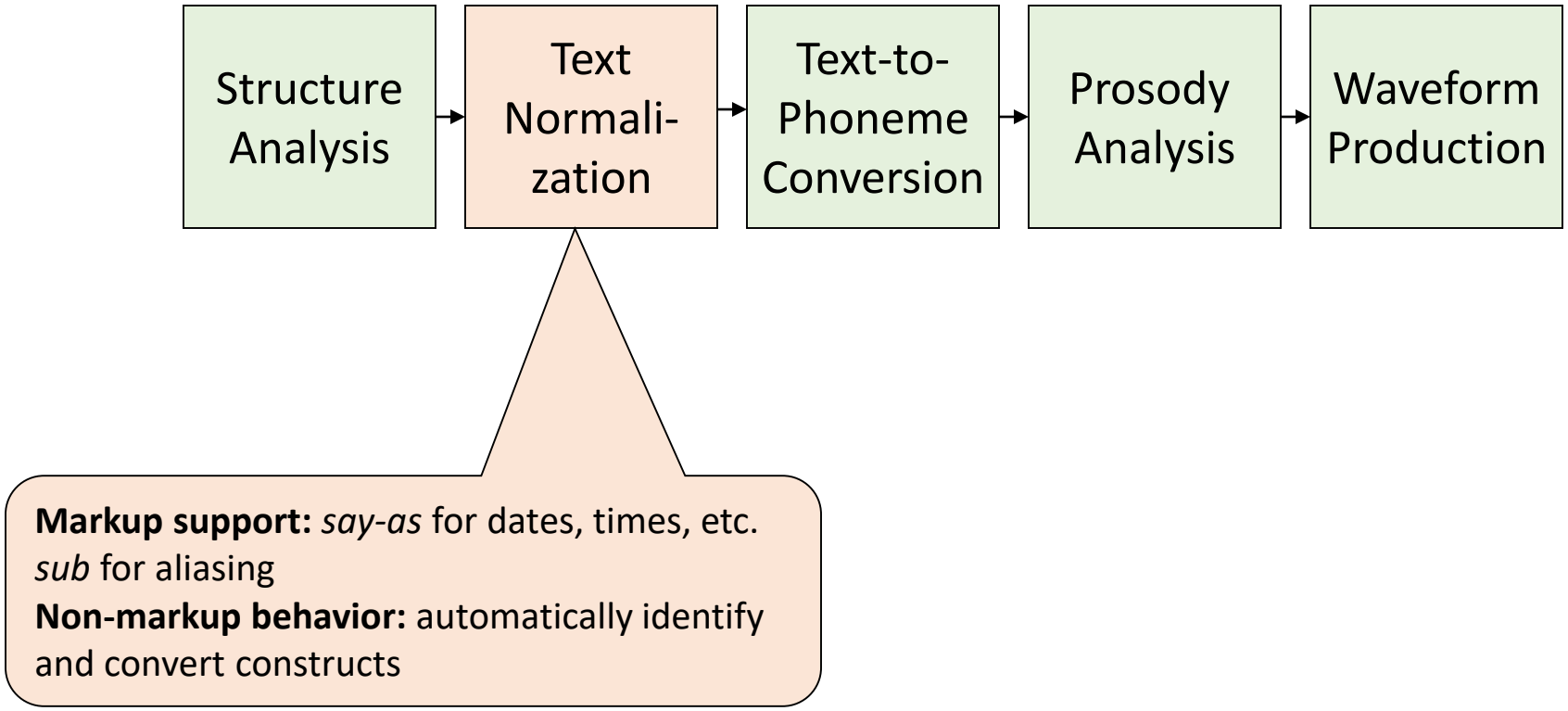
Before and after Structure Analysis

- Before structure analysis
  - Dr. Smith lives at 214 Elm Dr. He weights 214 lb. He plays bass guitar. He also likes to fish; last week he caught a 19 lb. bass.
- After structure analysis

```
<p>
  <s>
    Dr. Smith lives at 214 Elm Dr.
  </s>
  <s>
    He weights 214 lb.
  </s>
<s>
    He plays bass guitar.
  </s>
  <s>
    He also likes to fish; last week he caught
     a 19 lb. bass.
  </s>
</p>
```

Portland State UNIVERSITY

Speech Synthesis Markup Language

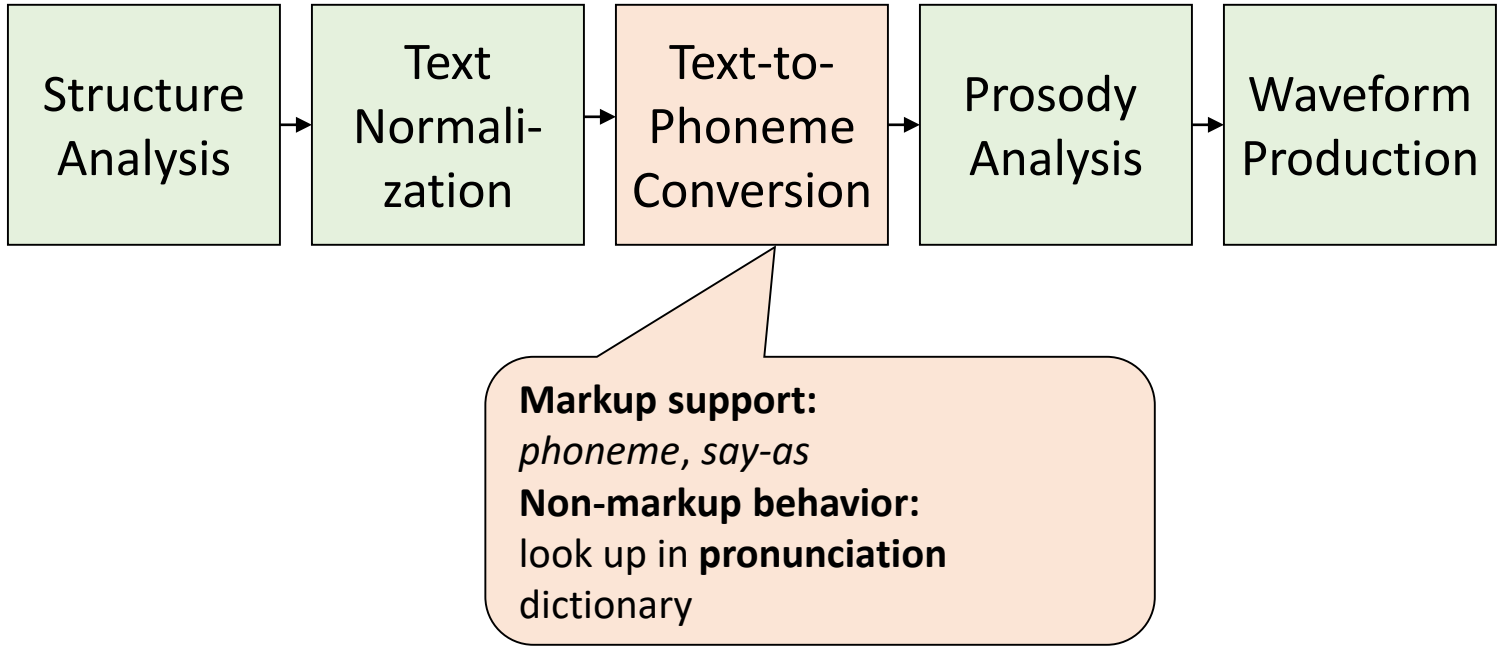


He weights 214 **_{** lb**.}**

After Text Normalization

  <s>
    _{**Dr. **} Smith lives at 214 Elm **_{**Dr. **}**
  </s>
  <s>
    He weights 214 **_{** lb**.}**
  </s>
  <s>
    He plays bass guitar.
  </s>
  <s>
    He also likes to fish; last week he caught a 19 **_{** lb. **}** bass.
  </s>

Speech Synthesis Markup Language



He plays **<phoneme alphabet = "ipa" ph="beɪs">**bass**</phoneme>** guitar.

Text-to-Phoneme Conversion

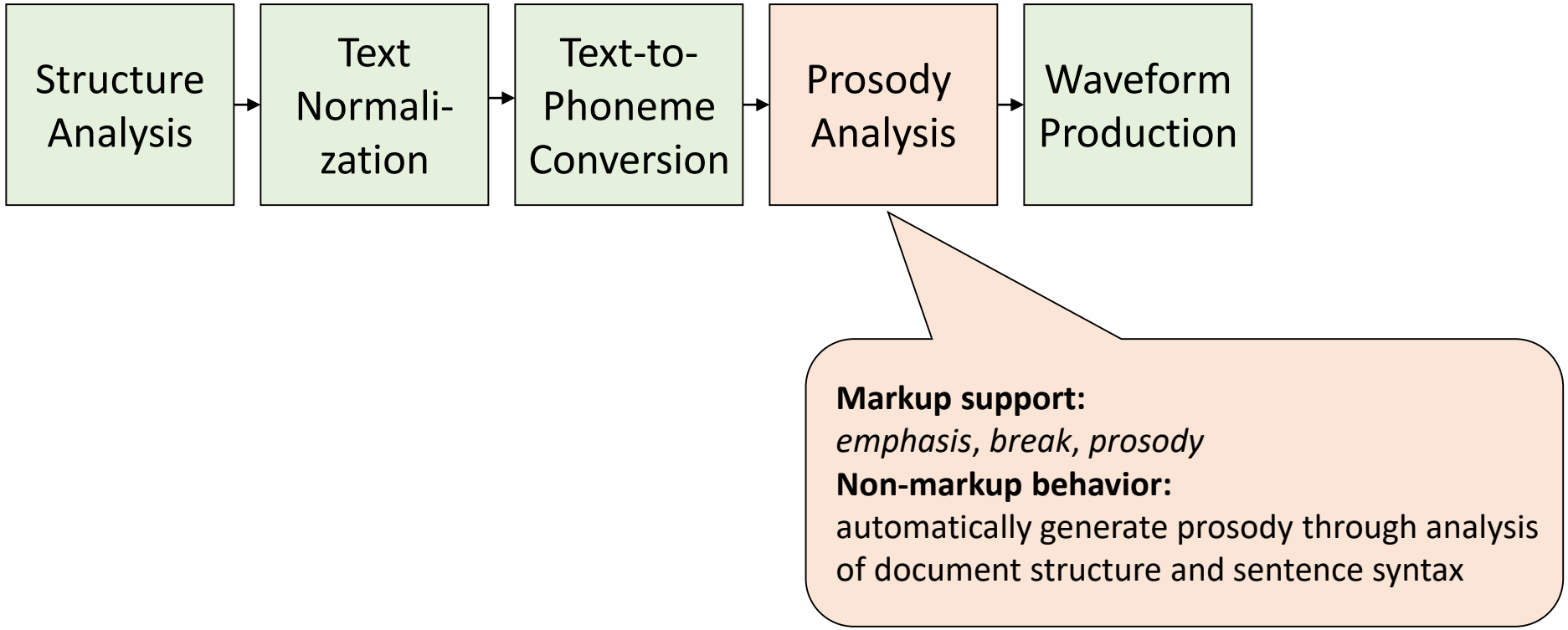
```
<p>
  <s>
    <sub alias = "doctor">Dr.</sub> Smith lives
    at <say-as interpret-as = “address"> 214 </sayas> Elm
    <sub alias = "drive">Dr. </sub>
  </s>
  <s>
    He weighs <sayas interpret-as = "number">214 </sayas>
    <sub alias= "pounds"> lb.</sub>
  </s>
  <s>
    He plays <phoneme alphabet = "ipa" ph="beɪs">bass</phoneme> guitar.
  </s>
  <s>
    He also likes to fish; last week he caught a
    <sayas interpret-as= "number">19 </sayas>
    <sub alias= "pound"> lb. </sub>
    <phoneme alphabet = "ipa" ph="bæs">bass</phoneme>.
  </s>
</p>
```

Bass guitar

Bass fish

Portland State UNIVERSITY

Speech Synthesis Markup Language



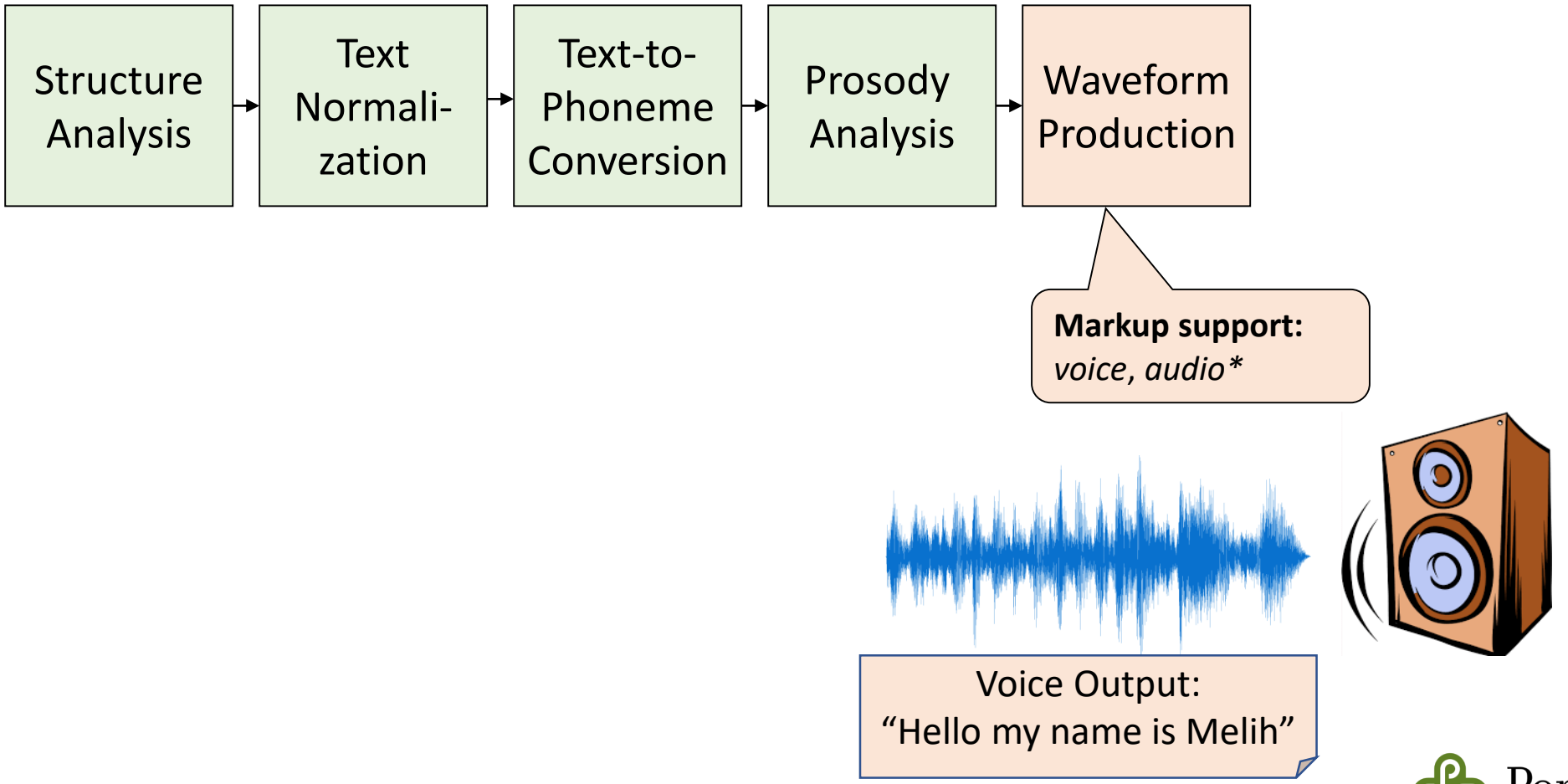
Prosody Analysis

```
<prompt>
   Environmental control menu. Do you want to adjust the lighting or temperature?
</prompt>
```

```
<prompt>
  Environmental control menu
  <break/>  <emphasis level = "reduced" >
  do you want to adjust the </emphasis>
  <emphasis level = "strong">
  lighting </emphasis>
  <break/>
   or <emphasis level = "strong">
   temperature? </emphasis>
</prompt>
```

Portland State UNIVERSITY

Speech Synthesis Markup Language



Portland State UNIVERSITY

Prerecorded messages vs. Speech Synthesis

Prerecorded messages

- Natural sounding
- Easy to understand
- Static data
- Tedious to record and tag

Speech Synthesis (TTS)

- Artificial sounding (Getting better)
- May be difficult to understand
- Computer-generated data
- Easy to specify

Prerecorded Messages vs. Speech Synthesis

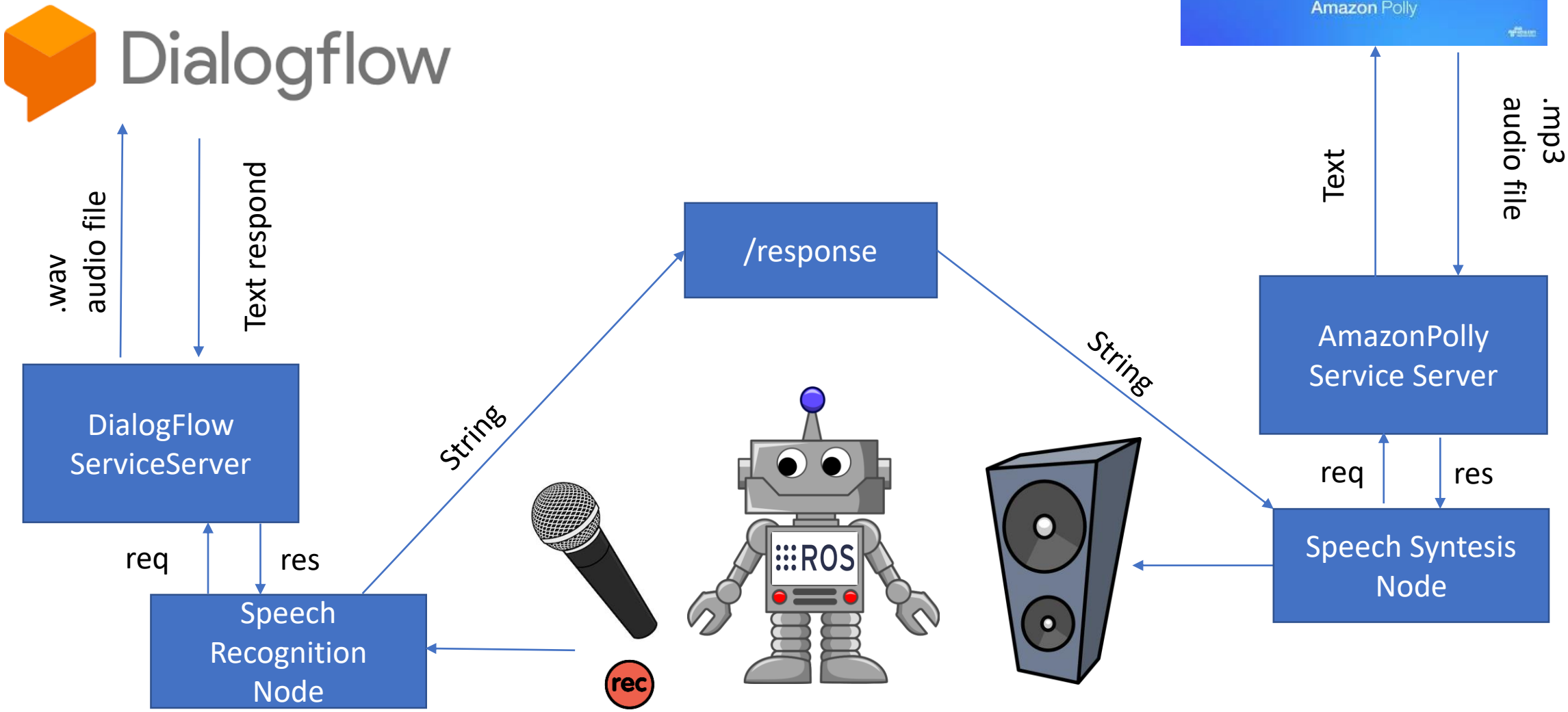
Prerecorded Messages

- Natural sounding
- Easy to understand
- Static data
- Tedious to record and tag

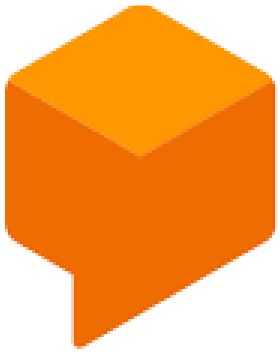
Speech Synthesis (TTS)

- Artificial sounding
- May be difficult to understand
- Computer-generated data
- Easy to specify

Speech Structure Overview



Portland State UNIVERSITY



Dialogflow

Google Dialogflow

Portland State
UNIVERSITY

Introduction to Dialogflow

- DialogFlow is a natural processing tool.
- Started as API.AI in 2010, acquired by Google
- Can be used to create user voice interfaces and chatbots

Introduction to Dialogflow

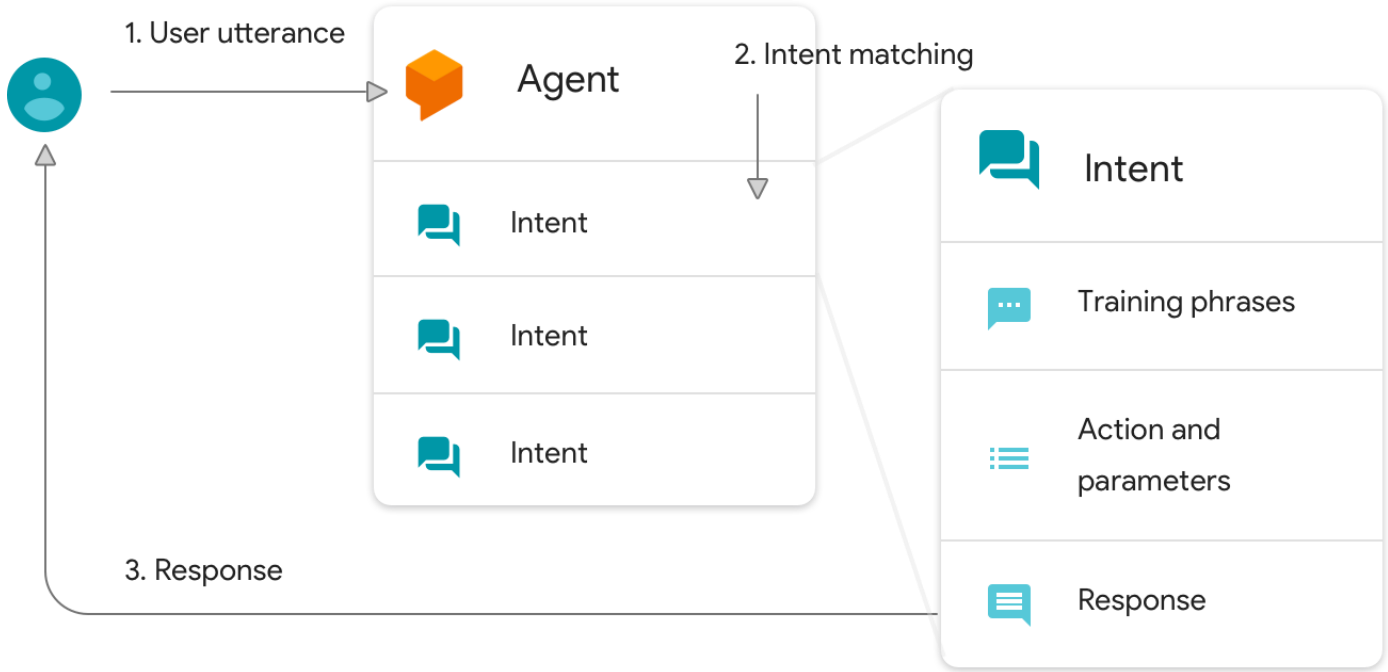
- You create a decision tree
- And Dialogflow uses Machine Learning to match user inputs to given training examples to figure out what the user's intent.
- It is flexible
- It keeps learning
- It works in many different circumstances
- It an be used with many programming languages such as Python, Java, Go, PHP, Ruby etc

Main Concepts and Features of Dialogflow

- Agent
  - Prebuild Agents
- Intents
  - Action and Parameters
  - Context
  - Events
  - Responses
  - Contexts
- Entities
  - System Entities
  - Custom Entities
- Training
- Small Talk

Agents

- Antural Language Understanding (NLU) modules.
- It can be included in your app, website, product or services
- It translates text and spoken user requests into actionable data.



Portland State UNIVERSITY

Intents

- Represents a mapping between what a user says and what action should be taken by your software
- In order to define how conversations work, **intents** are used in agent
- It maps user input to responses
- In each intent,
  - There are defined examples of user utterances that can trigger the intent
  - What to extract from the utterance
  - and how to respond.

Intents

- Intent has the following sections:
  - **User input (Text-Voice)**
    - Written in natural language and annotated with parameter values
    - “Weather forecast in Portland tomorrow”
  - **Action**
    - Steps your application will take when a specific intent has been triggered.
  - **Response**
    - Text, image, card, quick reply, custom payload
  - **Context**
    - Current context of a user's request
    - Resolves “it” or “them”

Entities

- Mechanism for identifying and extracting useful data from natural language inputs.
- Three entity types:
  - System Entity Types - defined by Dialogflow
  - Developer Entity Types - defined by developer
  - User - built for each individual end-user in every request
- Entities can be exported and downloaded

Entities - System Entity

- System Entity Type are defined by Dialogflow
  - **@sys.date** matches common date references such as "January 1, 2015" or "The first of January of 2015" and returns a reference value in ISO-8601 format: 2015-01-01T12:00:00-03:00.
  - **@sys.color** matches most popular colors and returns the matched color.
  - **@sys.unit-currency** matches amounts of money with indication of currency name, e.g., "50 euros" or "twenty dollars and five cents

Entities - Developer Entity

- Developer entity types are defined by developer
- For example, a brand might create an entity type to recognize its unique set of product names.
    - **@coffee**
    - **@tea**
    - **@bread**
    - **@condiments**
    - **@meats**

Entities

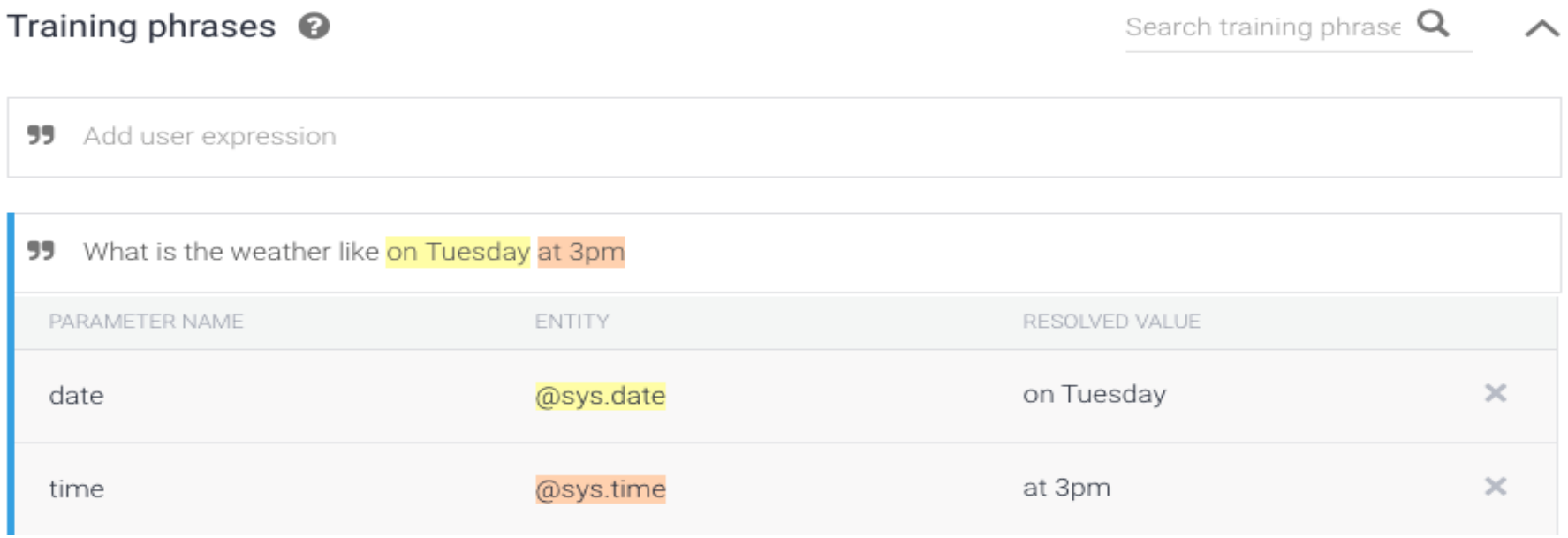
- User --built for each individual end-user in every request
  - Session entities extend or replace a developer entity at the user session level.
  - A user's session is the conversation they have with your agent from start to finish.
  - For example, if your agent has a @pets entity that includes "dog" and "cat", that entity could be updated to include "bird" or "fish" depending on the information your agent collects from a user.
    - **@pets**
      - **Dog**
      - **Cat**
      - **Bird**
      - **Fish**

Parameters

- Parameters are elements generally used to connect words in a user's response to entities.
- A corresponding parameter stores the extracted value from the utterance
- This lets us convert user input into structured data that we can use to do some logic or generate responses.

Parameters

- As an example, let's look at one of the example training phrases we defined earlier
- *User input/intent: "What is the weather like on Tuesday at 3pm".*
- The date and time entities (corresponding to the values *Tuesday* and *3 PM*) are automatically annotated as **@sys.date and @sys.time**



Portland State UNIVERSITY

Actions

- Step your application will take when a specific intent has been triggered by a user’s input.
- Actions can have parameters for extracting information from user requests

Responses

- Every intent must define a response that's returned to the user.
- There are two primary ways you can return a response to the user - either with a pre-defined, static response or with a response generated from a webhook.
- In both of these cases, you can use extracted parameters in the response.

Messages

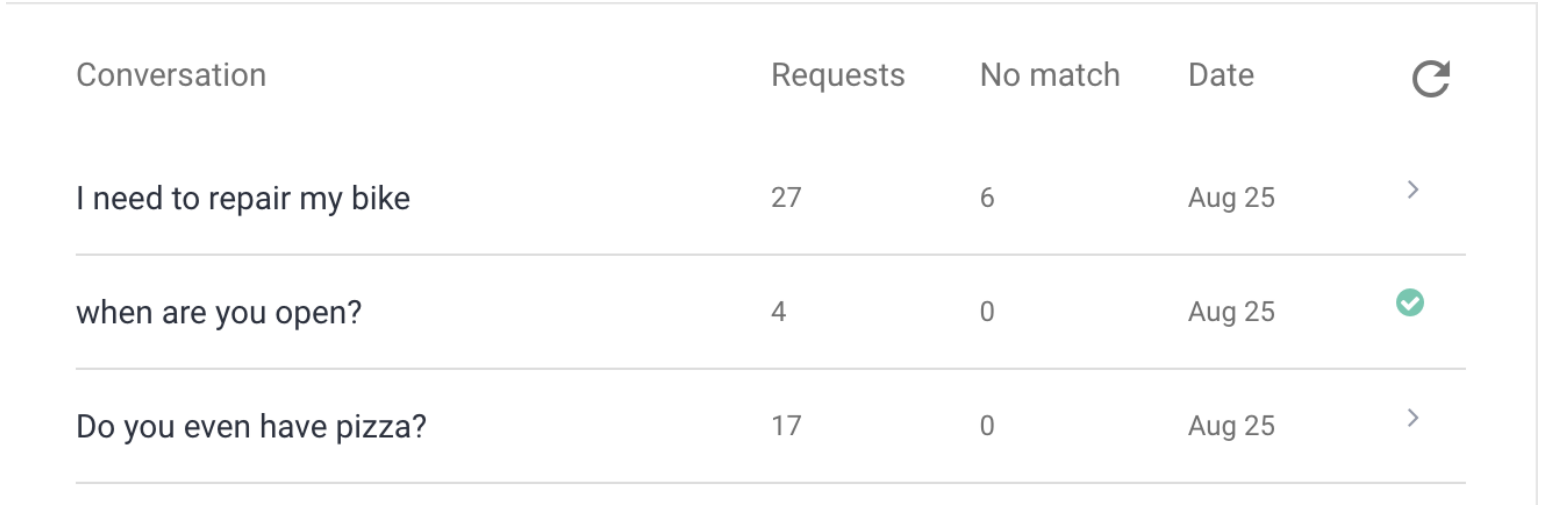
- Text
- Image
- Card
  - Consist of an image, a card title, a card subtitle, and interactive buttons
- Quick reply
  - Displayed as clickable buttons with pre-defined user responses.
- Custom payload
  - You can send custom payloads in the JSON format provided in the platforms documentation.

Events

- Events allow you to invoke intents based on something that has happened instead of what a user communicates.

Training

- Dialogflow is a natural language processing tool that is based on machine learning
- That means you can add training data that the agent learns from and uses to improve its performance.
- Logs of user interactions can be used to improve the performance of Dialogflow agents.



Conversation	Requests	No match	Date
I need to repair my bike	27	6	Aug 25
when are you open?	4	0	Aug 25
Do you even have pizza?	17	0	Aug 25

Portland State UNIVERSITY

Training

- Since Dialogflow's natural language processing is based on machine learning, you can add training data that the agent learns from and uses to improve its performance.
- Dialogflow's training feature provides an interface for incorporating both external and internal customer interaction logs into an agent's training phrases. You can use this feature to build a new Dialogflow agent using logs of existing customer interactions and to improve the performance of a live Dialogflow agent using its own logs.

Pre-build agents and Small Talk

- Dialogflow provides two out-of-the-box features that help get you started with agent and conversation design:
- **Prebuilt Agents**
  - Prebuilt Agents include intents and entities that cover the agent's topic.
  - You need to provide responses since they may depend on particular use cases or need to be retrieved from external sources (webhook or third-party API).
- **Small Talk**
  - Small Talk is used to provide responses to casual conversation.
  - This feature can greatly improve the user experience by covering common questions that may not pertain to your agent's intents you built.

Reference

- 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

End of class

Portland State UNIVERSITY