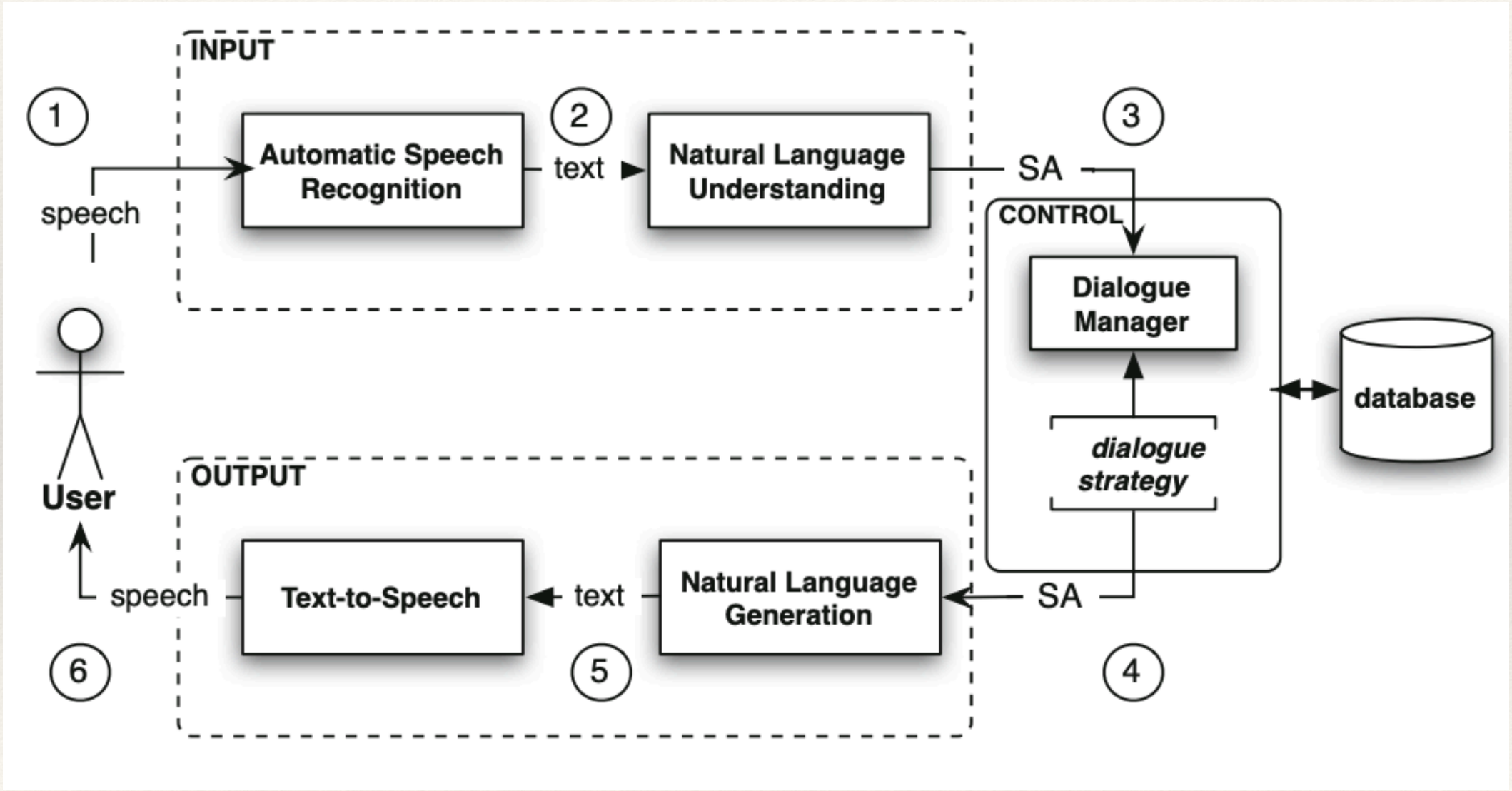
F20/21CA - Dialogue Management tutorial

RASA Core

Alessandro Suglia
as247@hw.ac.uk

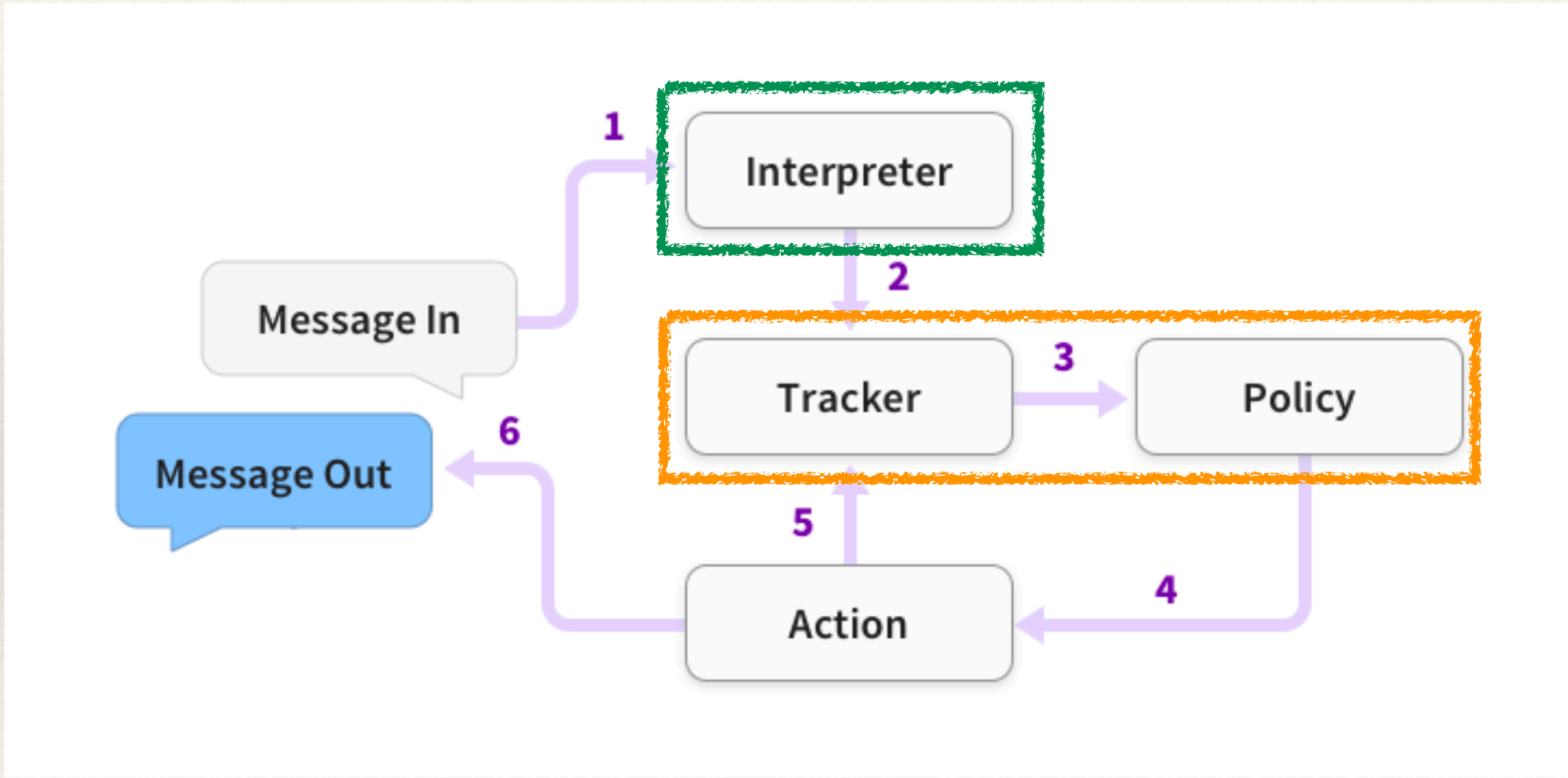
Background

Dialogue System overview



Rieser, Verena, and Oliver Lemon. Reinforcement learning for adaptive dialogue systems: a data-driven methodology for dialogue management and natural language generation. Springer Science & Business Media, 2011

Core



NLU

Dialogue Manager / Dialogue strategy

RASA Core

Core features

Several components are involved in the design of a RASA chatbot:

1. *Stories*: example conversation between the user and the chatbot;
2. *Domains*: defines the universe in which your assistant operates;
3. *Responses*: possible responses that the system is able to generate;

And many more that we won't be able to cover in this tutorial.

- U: Hello
- S: Hello there, how can I help?
- U: I'm looking for a cheap restaurant in Rome
- S: Ok, I'm on it. What type of cuisine?
- U: Spanish cuisine please
- S: For how many people?
- U: A table for six people please
- S: Great. I think you might be interested in ...

DISCLAIMER: this tutorial is based on RASA version 1.8.1

Stories 1/2

- RASA Stories are intended as training examples for your AI assistant
- They represent a possible scenario of conversation between the user and the AI:
  - User input expressed in terms of NLU annotations
  - System response expressed in terms of action identifiers
- RASA supports a dedicated file format for stories based on Markdown:

```
## greet + location/price + cuisine + num people
* greet
  - action_ask_howcanhelp
* inform{"location": "rome", "price": "cheap"}
  - action_on_it
  - action_ask_cuisine
* inform{"cuisine": "spanish"}
  - action_ask_numpeople
* inform{"people": "six"}
  - action_ack_dosearch
```

- Story identifier (debug only)
- User input (intent/entities format)
- System action

Stories 2/2

* inform{"cuisine": "spanish"}
- action_ask_numpeople

- User inputs, should be specified in terms of the NLU output format:
  - They start with the symbol "*"
  - Intent identifier (e.g., "inform")
  - Dictionary containing ("slot_name": "slot_value") pairs where "slot_name" is an entity identifier and "slot_value" is one of the values that can be associated with the entity identifier

- System actions are specified as identifier associated to specific system behaviours
- They start with the symbol "-"
- Two possible system actions are supported:
  1. Utterance actions (by convention they have "utter_" as prefix)
  2. Custom actions (by convention they have "action_" as prefix)

Slots

* inform{"cuisine": "spanish"}
- action_ask_numpeople

- Intent slots are defined in terms of the annotations generated by the NLU
- Slots represent the memory of your chatbot
- The dialogue flow will be affected by the slot values that your chatbot has in memory
- RASA supports different types of slots:
  - Text
  - Float
  - List
  - Categorical
  - Bool
  - Unfeaturized slot (used to store general data; does not affect the conversation)

Domains

- A domain defines the universe in which your assistant operates
- It is defined in terms of *intents, actions, slots* and *responses*
- *Intents* and *entities* are defined by the NLU component
- *Slots* are defined using the keyword "slots" (see the documentation for more details)
- *Responses* are messages the bot will send back to the user. Custom responses can include images and buttons (see the documentation for more details)

```
intents:
  - greet

actions:
- utter_greet

responses:
  utter_greet:
  - text: "Hey! How are you?"
  - text: "Hello. What's up?"
```

Dialogue Policy

- A dialogue policy selects the action that the chatbot will execute next
- In RASA multiple policy implementations are available
- They are usually configured in a priority list:
  - 5. FormPolicy
  - 4. FallbackPolicy and TwoStageFallbackPolicy
  - 3. MemoizationPolicy and AugmentedMemoizationPolicy
  - 2. MappingPolicy
  - 1. EmbeddingPolicy, KerasPolicy, and SklearnPolicy
- In this way, RASA ensures that the bot is always able to generate a response for the user input
- Some of the Policy are activated only when the *NLU* intent confidence is higher than a threshold
- More information about dialogue policy design can be found in the documentation (https://rasa.com/docs/rasa/core/policies/)

Response Generation

- Your bot responses are defined in the domain file
- They start with the keyword "responses"
- By convention a response identifier has the "utter_" prefix (e.g. "utter_greet")
- Multiple alternative responses can be specified using the "text" keyword followed by a string
- Curly brackets identify *variables* (e.g. {name})
- RASA can integrate external NLG service as well: https://rasa.com/docs/rasa/core/responses/#creating-your-own-nlg-service-for-bot-responses

```
responses:
  utter_greet:
    - text: "hey there {name}!"
  utter_goodbye:
    - text: "goodbye 😢"
    - text: "bye bye 😢"
  utter_default:
    - text: "sorry, I didn't get that, can you rephrase it?"
```

How to use RASA bot in Alana?

- We can use the programmatic API to load and use a trained RASA model
- In the file *CA2020_instructions/rasa_tutorial/core/rasa_bot.py* you can find a skeleton for an Alana bot that loads a custom RASA bot
- More advanced forms of deployment can be used as well: https://rasa.com/docs/rasa/user-guide/how-to-deploy/

```
# useful imports
>>> from rasa.core.agent import Agent
>>> from rasa.core.interpreter import RasaNLUInterpreter
# we load the RASA model which will be wrapped in a RASA Agent
>>> agent = Agent.load("path/to/your/model")
# we wait until the agent generates a response
>>> await agent.handle_text("hello")
[u'how can I help you?']
```

Exercises

Tutorial Introduction

- We will use the DSTC-8 dataset for this tutorial (same as the NLU tutorial one)
- The objective of this tutorial is to learn how to create a chatbot using RASA stories
- DSTC-8 dialogues can be arbitrarily complex:
  - Repair phenomena
  - API call execution
  - Restaurant search
  - ...
- In this tutorial we will focus on the DSTC-8 task 1 that involves the API call execution
- API call execution will be only simulated (by printing the slot values saved at the end of the dialogue)

Project Requirements

- Make sure your Anaconda environment is working correctly and that you can run RASA from command line
- Update your local copy of the repository **CA2020_instructions** using the command **git pull**
- Position yourself in the folder *rasa_tutorial/core/dstc_chatbot*

Project initialisation

- Update the repository **CA2020_instructions** using the command **git pull**
- The resulting project structure will be as follows:
    1. **data/**
        1. **stories.md**
        2. **nlu.md**
    2. config.yml
    3. credentials.yml
    4. endpoints.yml
    5. **domain.yml**
    6. actions.py
- You can train a model use the command: **rasa train**
- You can test the model from command line using: **rasa shell**

For a more description of the files not covered in this tutorial follow the official documentation: https://rasa.com/docs/rasa/core/about/

Exercise 1

1. Edit the file *domain.yml* in order to support the following NLU intents:
   - intent:make_reservation
   - intent:greet
   - intent:inform
2. and slots:
   - cuisine
   - location
   - price
   - number
3. Define possible *actions* that your bot is able to perform (in this case *utter* a response)

Exercise 2

- Edit the file *data/stories.md* and create a story that is able to support the following dialogue:

```
- U: hello
- S: hello, what can i help you with today
- U: can you book a table
- S: I'm on it
- S: any preference on a type of cuisine
- U: italian cuisine
- S: how many people would be in your party
- U: for two
- S: where should it be
- U: in london
- S: which price range are you looking for
- U: moderate price
- S: ok let me look into some options for you
- S: api_call italian london two moderate
```

Exercise 3

- Extend the chatbot in order to deal with multiple slots for a single turn.
- E.g. "can you make a restaurant reservation in London in a restaurant with moderate price?"
- Define new stories that are able to deal with combination of slots
- Try to look for several examples in the original bAbI dataset that require this feature and create multiple stories to support them

Exercise 4

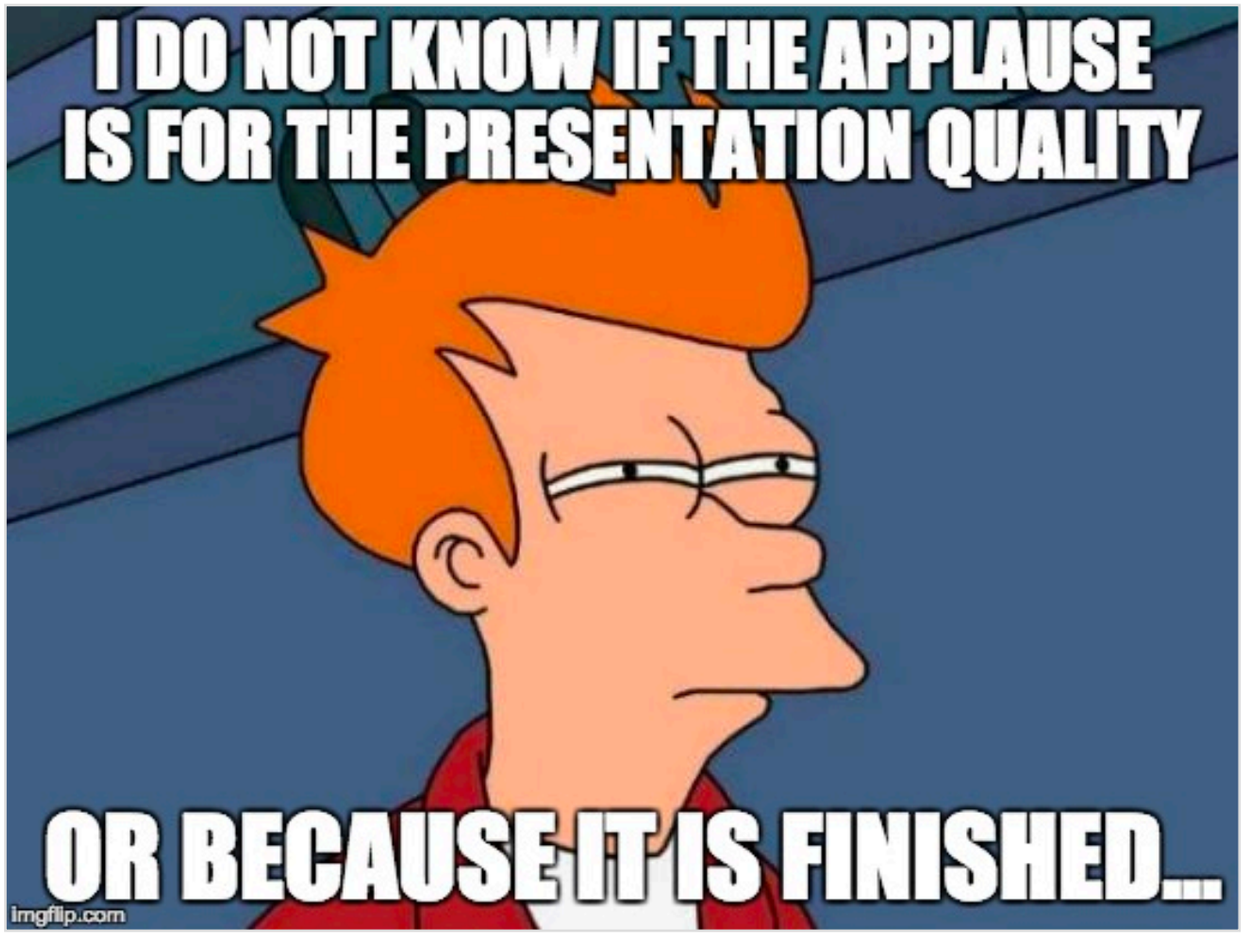
- At this stage you are able to create a chatbot using the basic RASA core features
- It's time to think whether it's possible to use RASA core for your own project
- Some tasks to complete:
  - Create a new RASA project for your bot
  - Define the domain information by editing the *domain.yml* file
    - What intents?
    - What slots?
    - What actions?
- Define possible stories for your chatbot
- Once you're satisfied, try to integrate it in the Alana framework

Exercise 5

Reflect on the differences between AIML and RASA.
When would you use one instead of the other?

Additional info about RASA core

- Official tutorial: https://rasa.com/docs/rasa/core/about/
- RASA Masterclass: https://www.youtube.com/playlist?list=PL75e0qA87dlHQny7z43NduZHPo6qd-cRc
- RASA chatbot templates: https://github.com/cedextech/rasa-chatbot-templates
- Rieser, Verena, and Oliver Lemon. Reinforcement learning for adaptive dialogue systems: a data-driven methodology for dialogue management and natural language generation. Springer Science & Business Media, 2011 (https://www.springer.com/gp/book/9783642249419) -- particularly interesting the chapter about DM



Thanks!

Any questions?

@ale_suglia

as247@hw.ac.uk