Workshop day 5 🎉

RECAP

What we've learned so far

- Set up
- Deep dive of NLU and dialogue management
- Build an MVP assistant
- Share your assistant with the outside world
- Make continuous improvements and take your assistant to the next level

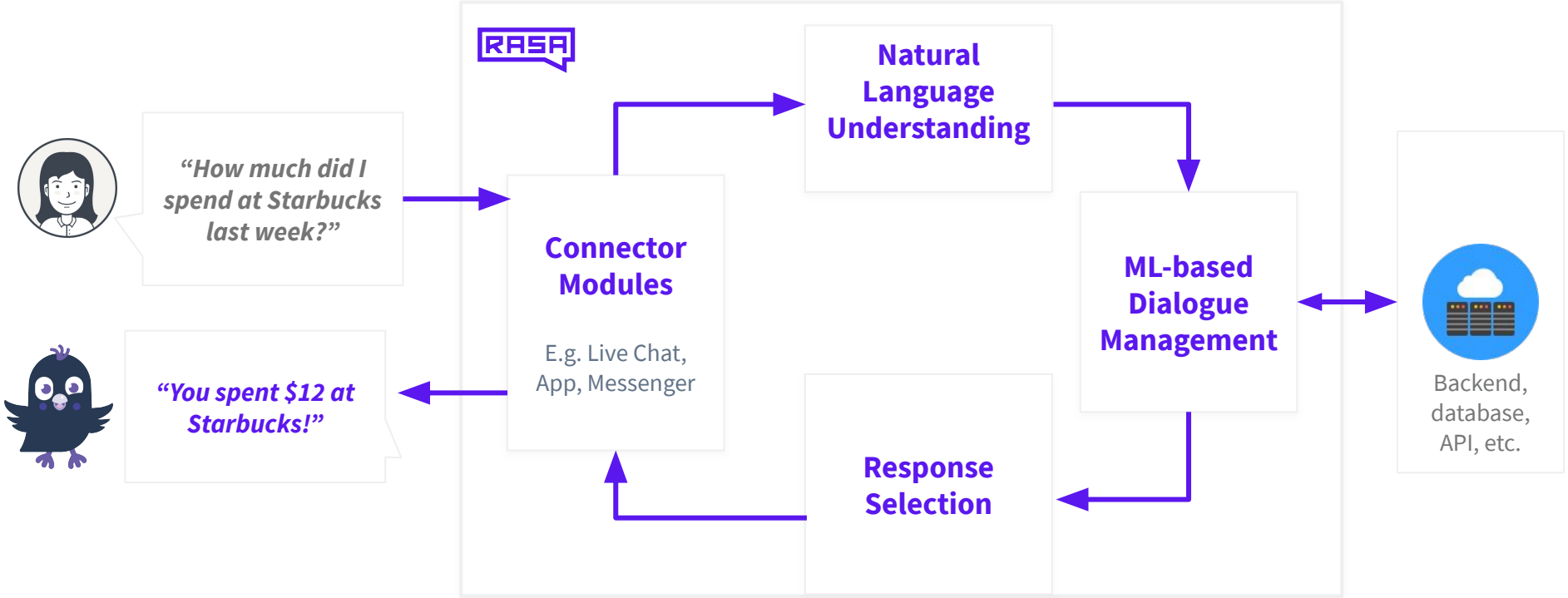
How to get help

- Please ask your questions in the **#workshop-help** Slack channel rather than the Zoom chat. Slack is the place the Rasa team will be monitoring most closely.
  - Karen, Mady, and Juste will be in Slack answering questions, as well as Arjaan, Melinda, and Ella from our Customer Success Engineering team
- Tuesday - Friday, the Rasa team will be dedicating time to answering your questions in Slack
  - Feel free to ask questions outside of these hours, but responses may be a little slower
- A note on time zones:
  - The Rasa team is based across the US and in Berlin. We'll do our best to answer questions within team members' working hours, but please keep in mind, some discussions may need to take place async rather than in real time.

Roadmap

- Recap for ~60 mins
- Panel with Juste, Ty, Mady, Alex, and Alan for ~45 mins
- Talk about next steps ~10 mins

Rasa Open Source



Intents and Entities

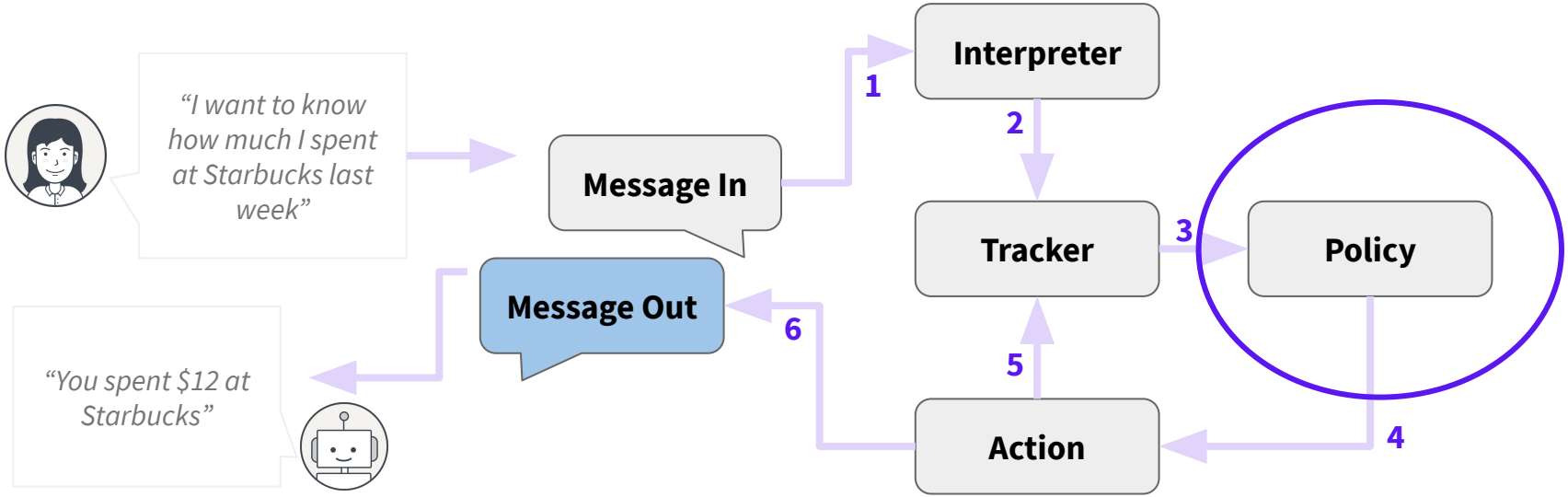
Two of the most common and necessary types of information to extract from a message

Intent: What the user wants to achieve

Entity: Key piece of information

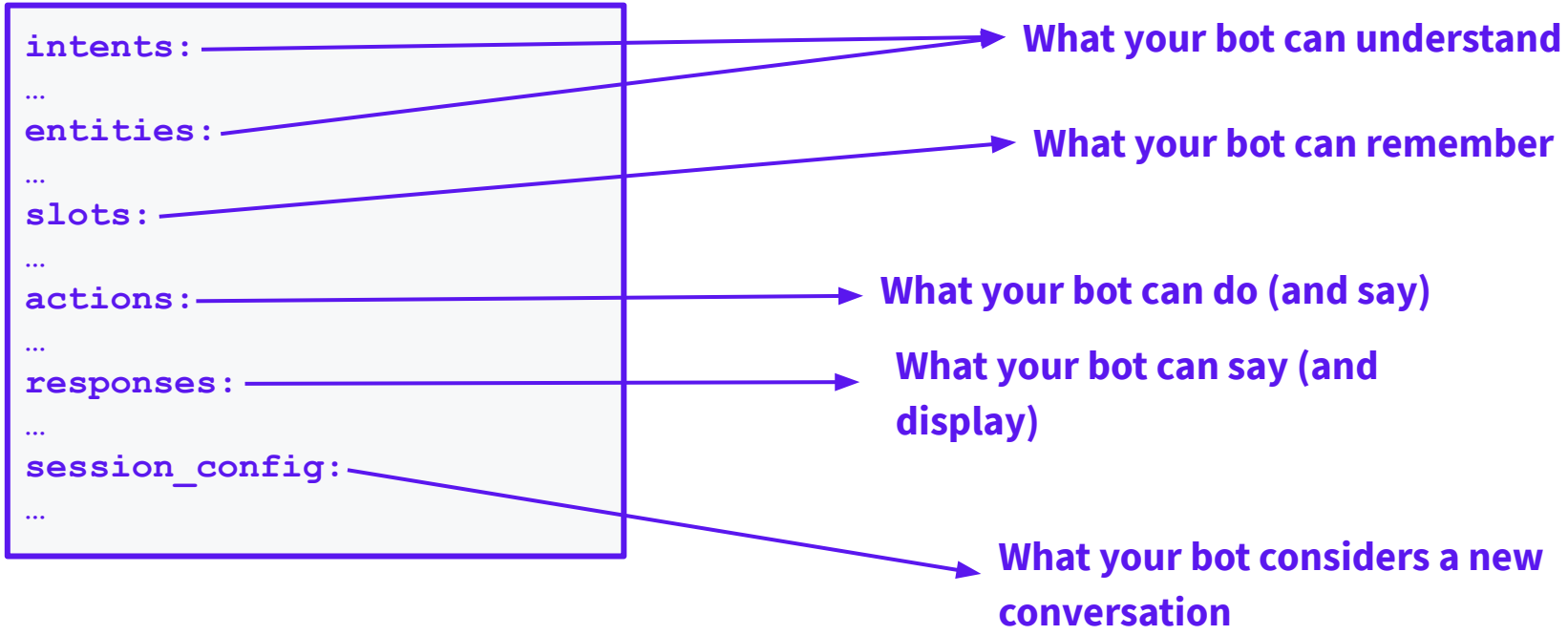
Policies

- Decide which action to take at every step in the conversation
- Each policy predicts an **action** with some **probability.** This is called **core confidence**.



Domain

Defines the “world” of your assistant - what it knows, can understand, and can do



Multiple Policies

- The policy with the **highest confidence** wins.
- If the confidence of two policies is equal, the policy with the **highest priority** wins.

Rule-based policies have higher priority than ML-based policies

```
Policy priorities
(higher numbers = higher priority)

5. FormPolicy
4. FallbackPolicy, TwoStageFallbackPolicy
3. MemoizationPolicy, AugmentedMemoizationPolicy
2. MappingPolicy
1. EmbeddingPolicy, KerasPolicy
```

Scope Conversation

How to get started with conversation design

The assistant’s purpose

Leverage the knowledge of domain experts

Common search queries

FAQs and wikis

What is a Minimum Viable Assistant (MVA)?

- A basic assistant that can handle the most important **happy path** stories.
  - **Happy path**: If your assistant asks a user for some information and the user provides it, we call that a happy path.
  - **Unhappy path:** All the possible edge cases of the bot

Machine learning models require training data that the models can generalize from

NLU needs data in the form of examples for intents

```
## intent:bot_challenge
- are you a bot?
- are you a human?
- am I talking to a bot?
- am I talking to a human?

## intent:i_like_food
- I like [apples](food)
- my friend likes [oranges](food)
- I'm a fan of [pears](food)
- do you like [coffee](food)
```

Dialogue management model needs data in the form of stories

```
## new to rasa at start
* how_to_get_started{"user_type":
"new"}
    - action_set_onboarding
    - slot{"onboarding": true}
    - utter_getstarted_new
    - utter_built_bot_before
* deny
    - utter_explain_rasa_components
    - utter_rasa_components_details
    - utter_ask_explain_nlucorex
```

Use the Rasa CLI to test your assistant

End to End Evaluation

Run through test conversations to make sure that both NLU and Core make correct predictions.

```
$ rasa test
```

NLU Evaluation

Split data into a test set or estimate how well your model generalizes using cross-validation.

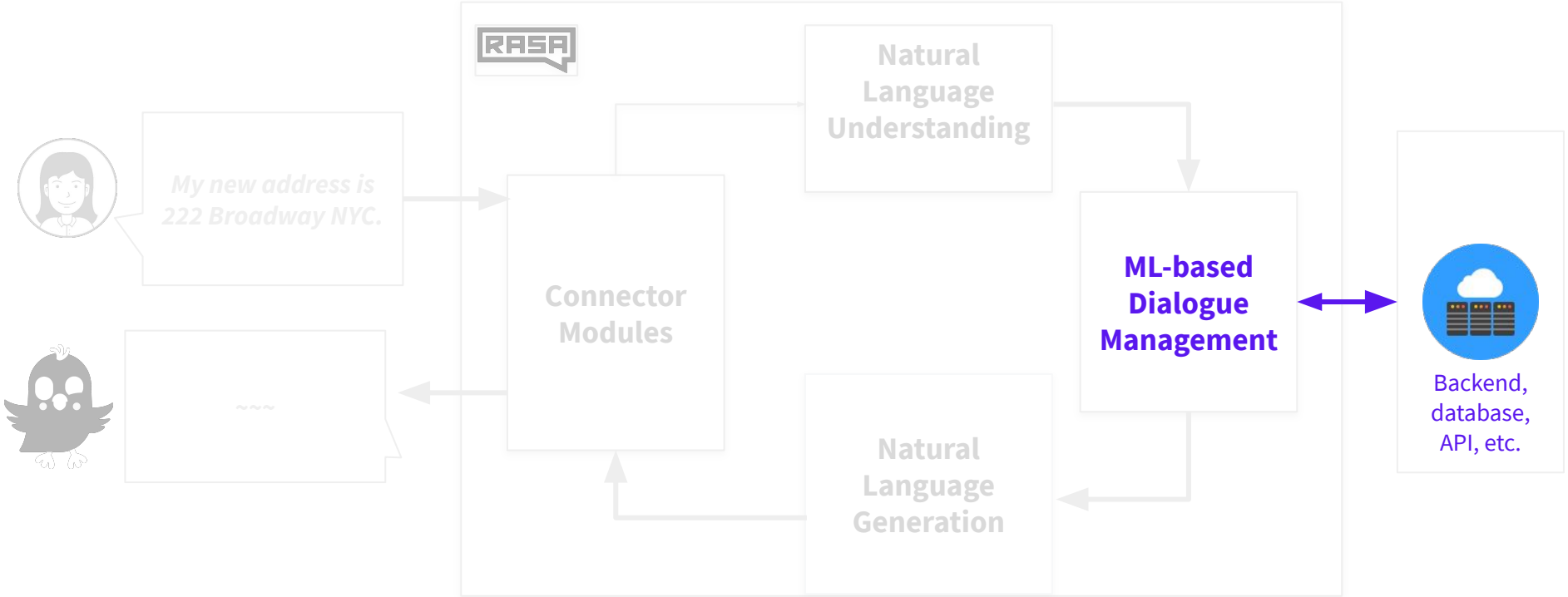
```
$ rasa test nlu -u
data/nlu.md --config
config.yml
--cross-validation
```

Core Evaluation

Evaluate your trained model on a set of test stories and generate a confusion matrix.

```
$ rasa test core
--stories
test_stories.md --out
results
```

Custom Actions: Connecting to the outside world



Events

- **Internally**, all conversations are represented as a sequence of **events.**
- Some events are automatically tracked

Events

```
UserUttered
SlotSet
SlotSet

BotUttered
UserUttered
Restarted
```

Story

```
## new to rasa at start
* how_to_get_started{"user_type":
"new"}
    - action_set_onboarding
    - slot{"onboarding": true}
    - utter_getstarted_new
* restart
    - action_restart
```

Tracker

- Trackers maintain the **state of a dialogue** between the assistant and the user.
- It keeps track of events, slots, session info etc.

```
{
 "sender_id": "default",
 "slots": {
    "user_type": "new",
    "onboarding": true},
 "latest_message": {...},
 "events": [
    ...
    {"event": "action",
     "name": "action_set_onboarding",
     "policy": "MemoizationPolicy",
     "confidence": 0.92},
    ...
    {"event": "user",
     "text": "\/restart",...},
    ...
 ]
}
```

Session info

Slots

Events:
...
SlotSet
...
UserUttered
...

Slots

Your bot's memory

- Can store:
  - user-provided info
  - info from the outside world
- Can be set by:
  - NLU (from extracted entities, or buttons)
  - Custom Actions
- Can be configured to affect or not affect the dialogue progression

Actions

Things your bot runs in response to user input.

Four different action types:

- **Utterance actions**: `utter_`
  - send a specific message to the user
  - specified in `responses` section of the domain
- **Retrieval actions**: `respond_`
  - send a message selected by a retrieval model
- **Custom actions**: `action_`
  - run arbitrary code and send any number of messages (or none).
  - return events
- **Default actions**:
  - built-in implementations available but can be overridden
  - E.g. action_listen, action_restart, action_default_fallback

Custom Actions: Examples

Custom actions can do whatever you want them to! e.g.

- **Send multiple messages**
- **Query a database**
- **Make an API call to another service**
- **Return events e.g.**
  - Set a slot
    - e.g. based on a database query)
  - Force a follow up action
    - force a specific action to be executed next
  - Revert a user utterance
    - I.e. remove a user utterance from the tracker

Forms

The structure of the form

```
## pay credit card happy path
* greet
    - utter_greet
* pay_cc
    - cc_payment_form
    - form{"name": "cc_payment_form"}
    - form{"name": null}
```

User input

Form action

Form activated

Form deactivated

Forms

Handling unhappy paths

Users are not always cooperative - they can change their mind or interrupt the form.

- Use action `action_deactivate_form` to handle the situation where users decide not to proceed with the form:

```
## chitchat
* request_restaurant
    - restaurant_form
    - form{"name": "restaurant_form"}
* stop
    - utter_ask_continue
* deny
    - action_deactivate_form
    - form{"name": null}
```

Rasa X turns conversations into training data

Review 🔍 Annotate ✏️ Improve 💪

Real Conversations > Synthetic Data

During development, training data is created by:

- Writing down hypothetical conversations and training examples
- Studying existing chat transcripts or knowledge bases
- Consulting subject matter experts

That's a great start, but all of those scenarios are based on human:human conversations.

Human:bot conversations have different patterns - users say different things when they are talking to a bot.

The best training data is generated from the assistant's ***actual conversations***

Conversation Driven Development (CDD)

The challenge

When developing assistants, it’s impossible to anticipate all of the things your users might say.

The approach

A user-centric process: listening to your users and using those insights to improve your AI assistant.

Principles

Prototype, Read, Annotate, Test, Track, Fix

Blog post: Conversation-Driven Development

THE APPROACH

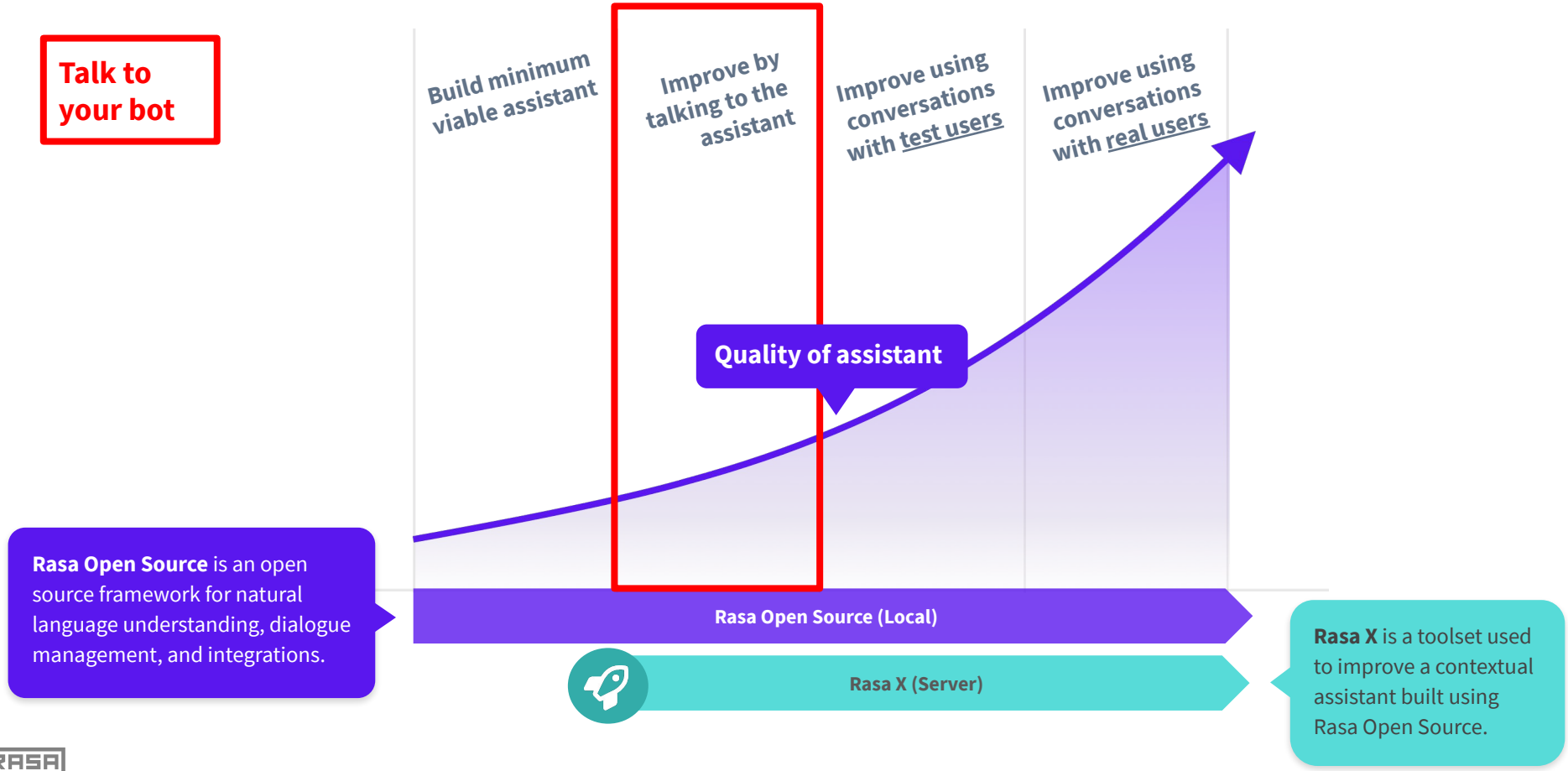
Continually improve your assistant using Rasa X

Ensure your new assistant passes tests using **continuous integration (CI)** and redeploy it to users using **continuous deployment (CD)**

Collect conversations between users and your assistant

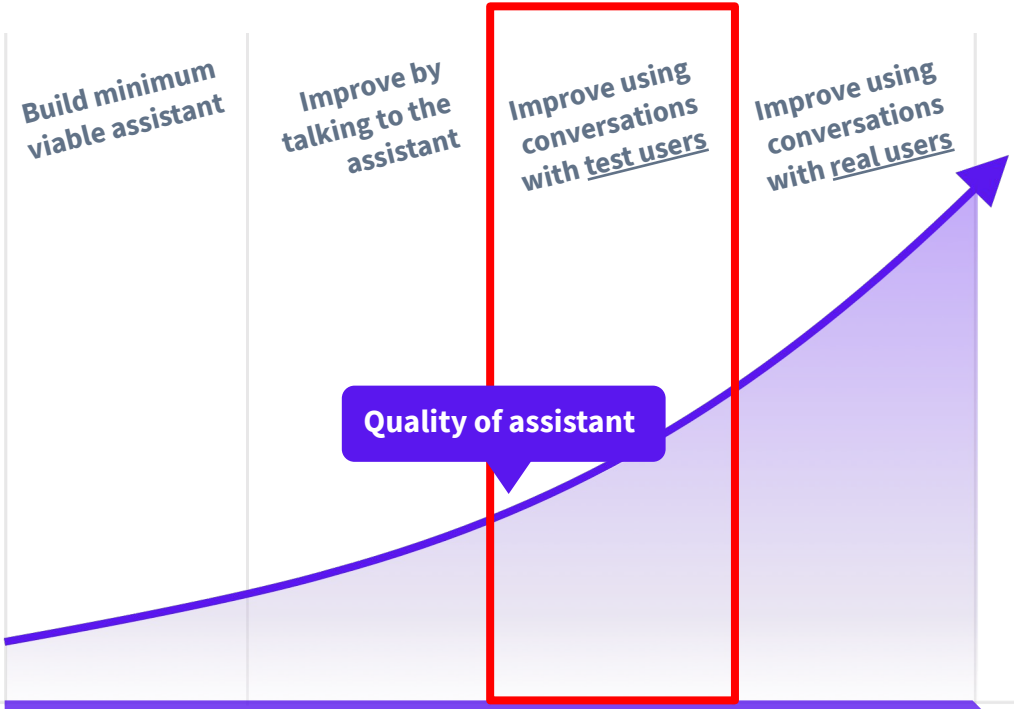
Review conversations and **improve your assistant** based on what you learn

Build a minimum viable assistant with Rasa Open Source + improve it using Rasa X



Build a minimum viable assistant with Rasa Open Source + improve it using Rasa X

Share your bot



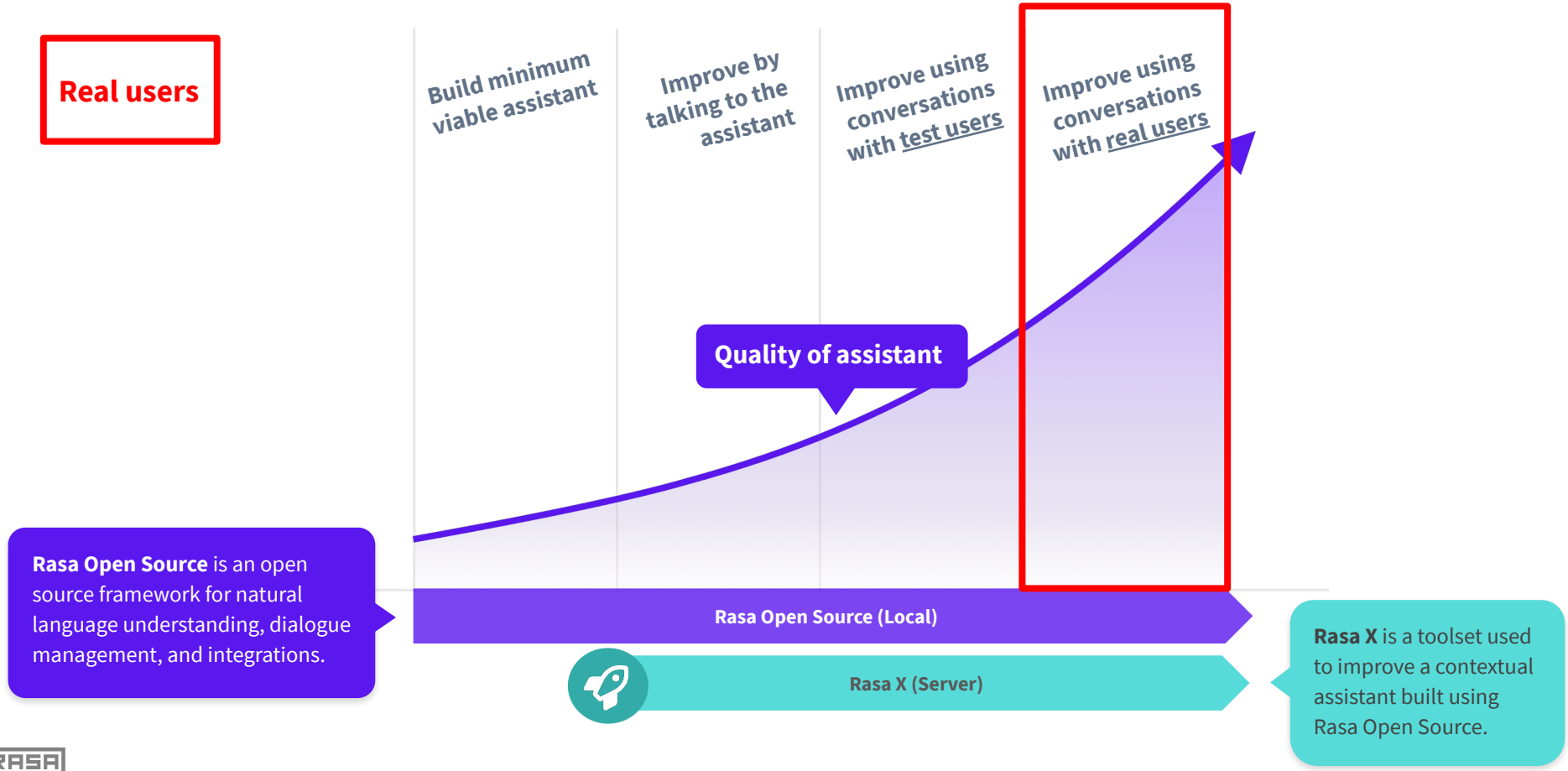
Rasa Open Source is an open source framework for natural language understanding, dialogue management, and integrations.

Rasa Open Source (Local)

Rasa X (Server)

Rasa X is a toolset used to improve a contextual assistant built using Rasa Open Source.

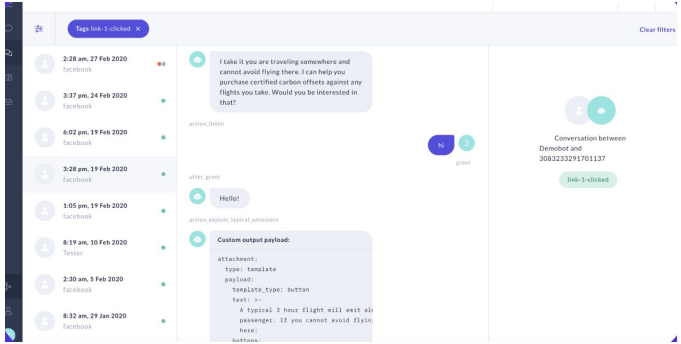
Build a minimum viable assistant with Rasa Open Source + improve it using Rasa X

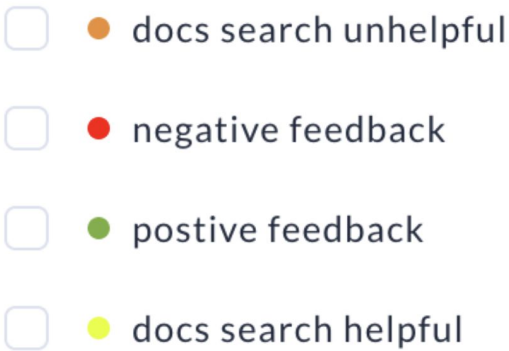


REVIEW CONVERSATIONS

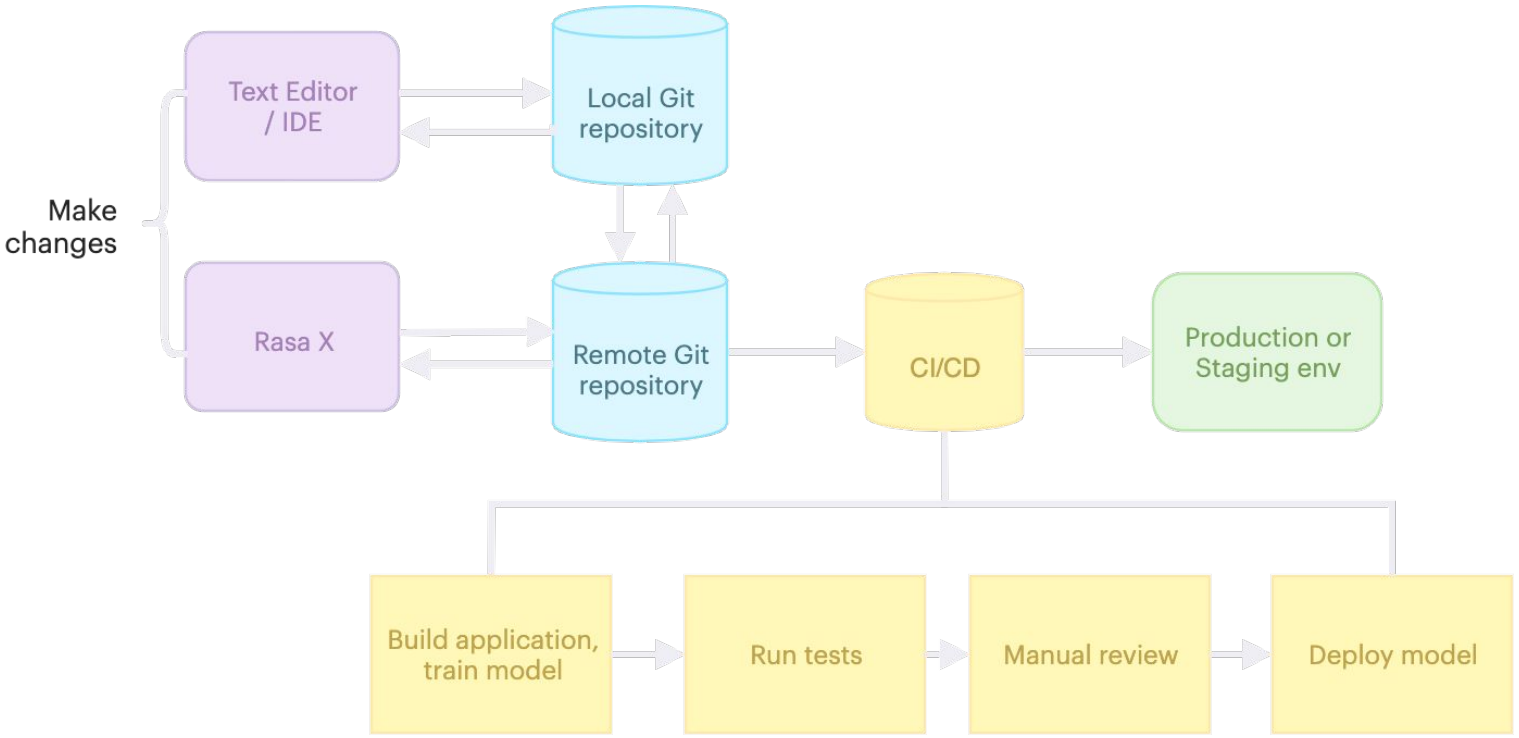
Drive your development based on feedback from users

1. Use the Rasa X API to tag conversations to help determine success (e.g. Carbon Bot or Sara)
2. Review conversations and see where the fail
3. Add ideas for improvement as issues on GitHub
4. Manually tag conversations in Rasa X to help you process conversations and figure out big problems





End-to-end workflow for managing assistant updates



CI/CD lets you automate testing for your assistant

```yaml
    - name: Rasa Data Validation
      working-directory: ${{ github.workspace }}
      run: |
        rasa data validate --debug
  training-testing:
    name: Testing Stories
    runs-on: ubuntu-latest
    needs: [data-validation]
    steps:
    - uses: actions/checkout@v1
    - name: Set up Python 3.7
      uses: actions/setup-python@v1
      with:
        python-version: 3.7
    - name: Install dependencies
      run: |
        python -m pip install --upgrade "pip<20"
        pip install -r requirements-dev.txt
    - name: Cross-validate NLU model
      run: |
        rasa test nlu -f 3 --cross-validation
        python .github/workflows/format_results.py
    - name: post cross-val results to PR
      uses: amn41/comment-on-pr@comment-file-contents
      continue-on-error: true
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
      with:
        msg: results.md
```

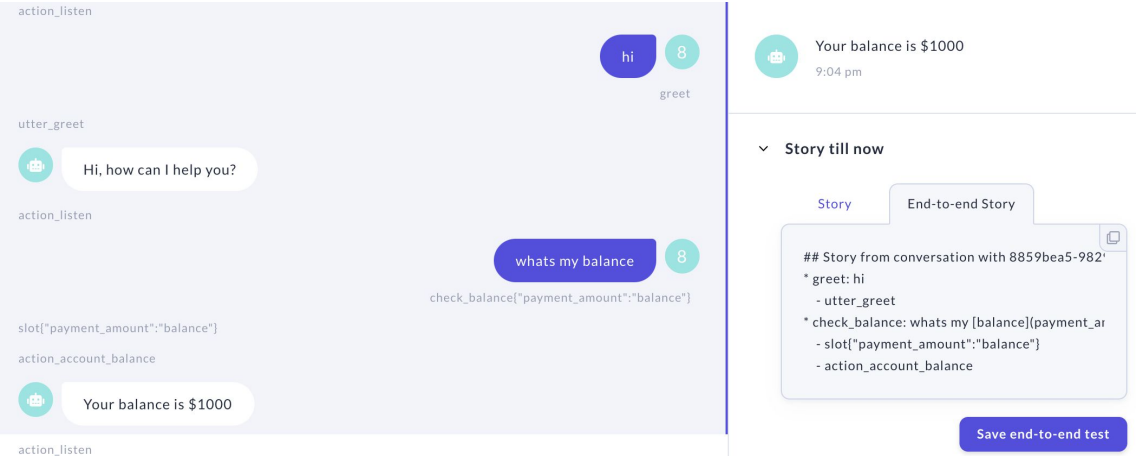
Validate data to check for mistakes

Test NLU model

Print results to a pull request

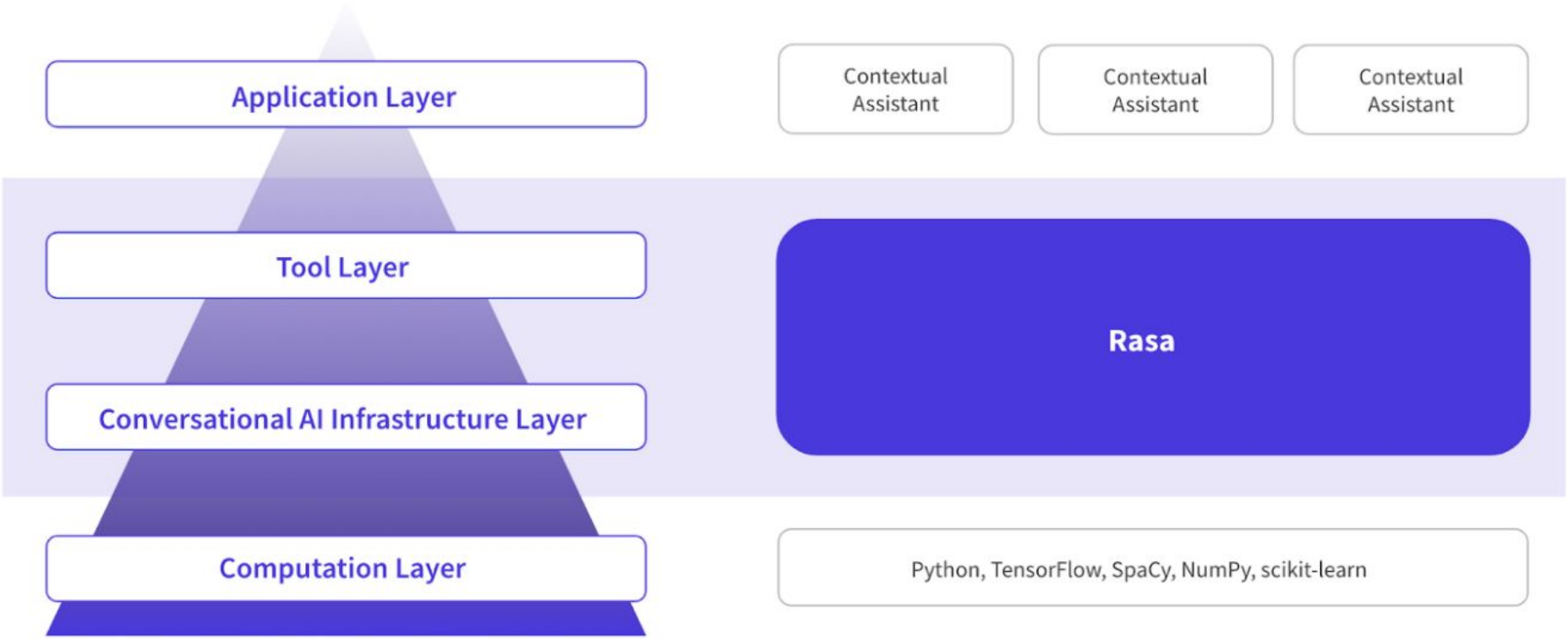
Link to full example

Use Rasa X to build an end-to-end test set from real conversations

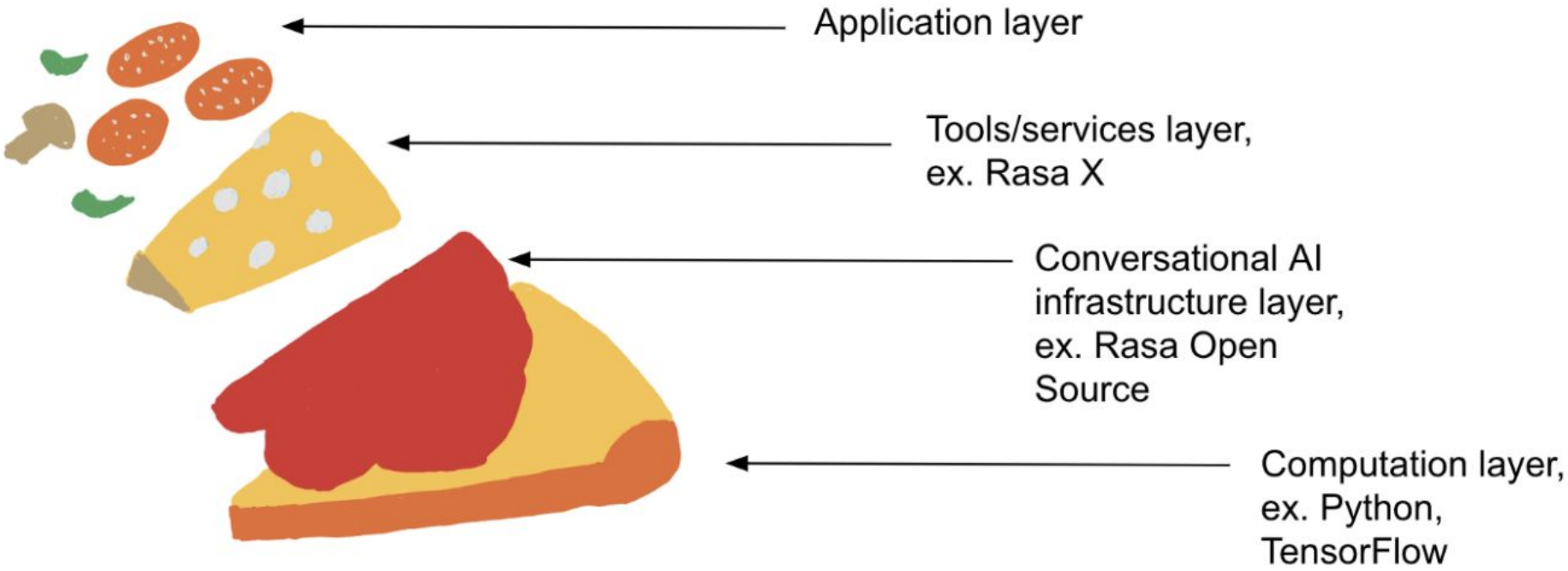


- Turn real conversations into end-to-end tests using Rasa X
- Collect a sample of the true distribution of real conversations
- Run them as part of an automated testing pipeline

How Rasa fits into your stack



How Rasa fits into your stack



Next steps

- Your test will be posted on Slack and email. You have a week to take your test 🎓
- Your **VMs** will be up for a week 😎
- You'll have access to **Slack** for a week so you can continue to ask questions and interact with the Rasa team 🗣️
- To continue the conversation, we encourage you to join the Rasa Community Forum where we've added an area for worksop alumni 👥
- We'll share the edited versions of the workshop sessions and slide deck soon 🎥

We'd love your feedback, please take our workshop survey and let us know about your experience!

Panel

Alex Weidauer, CEO and Co-founder
Alan Nichol, CTO and Co-founder
Juste Petraityte, Head of Developer Relations
Mady Mantha, Senior Product Evangelist
Ty Dunn, Product Manager

Moderated by: Karen White, Developer Marketing Manager

RASA