Making Interoperability Boring

Aries Agent Test Harness (AATH)

ĐA DẠNG HOẠT ĐỘNG TOUR TEAM BUILDING DU LỊCH NƯỚC NGOÀI:

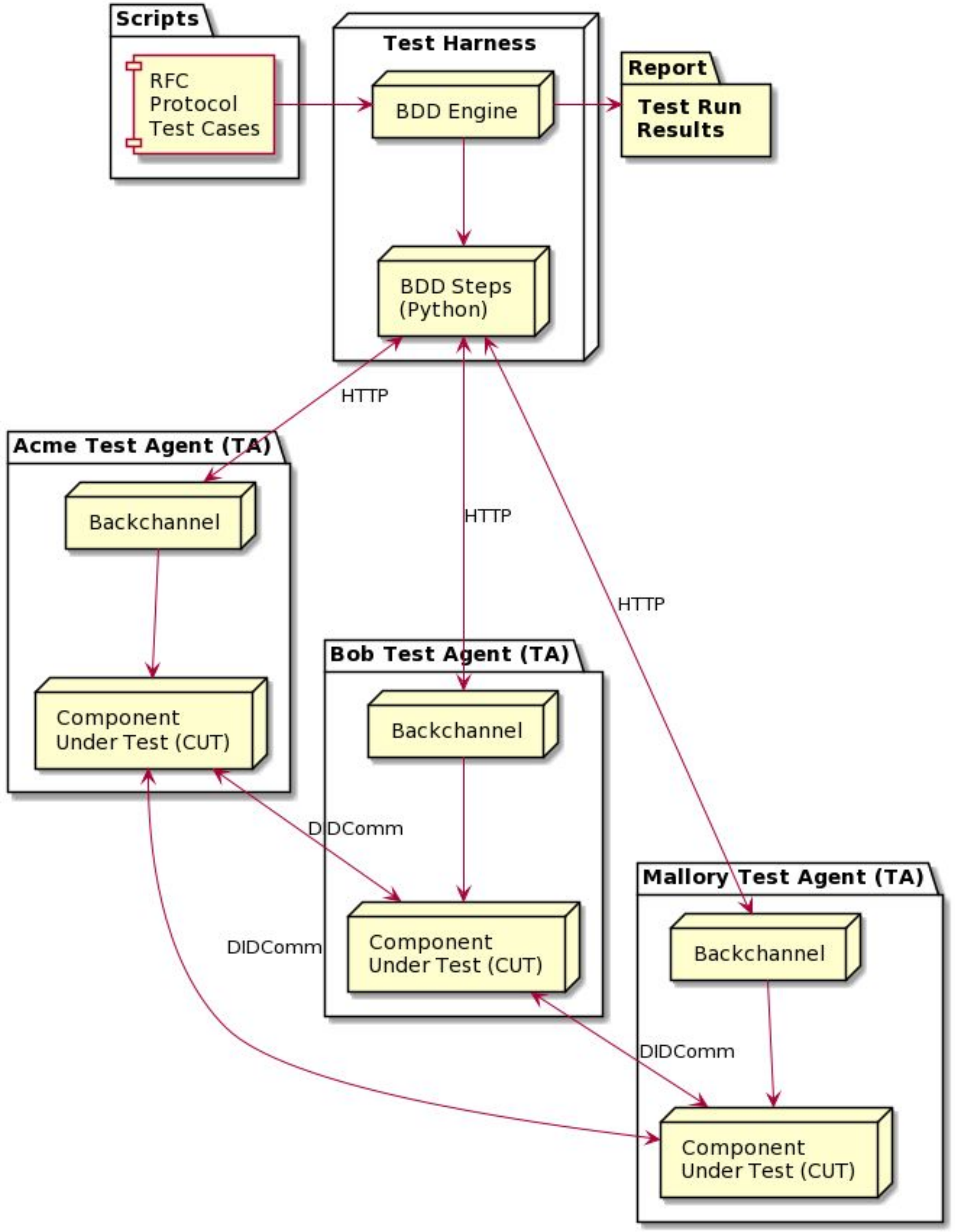
BC Gov
Stephen Curran / Ian Costanzo / Sheldon Regular

Test Harness Requirements

- Easy to write RFC-driven/protocol test cases and assemble test suites
- Interoperability focus
- Define an API to drive components under test
    - The hard part: Per framework or agent backchannel

Architecture

- BDD Test Scripts based on RFCs
- BDD steps call backchannels
- Runtime binding of role to test agent
- Predefined test case participants
    - “Acme” is an enterprise issuer/verifier agent
    - “Bob” is a holder/prover agent
    - “Mallory” is a misbehaving holder/prover



Benefits

- The Test Suite is a driver, not an agent—no protocol implementations to write
- Fewer tests to write
- Two (or more) agents are involved in the test runs
- Industry standard test runner

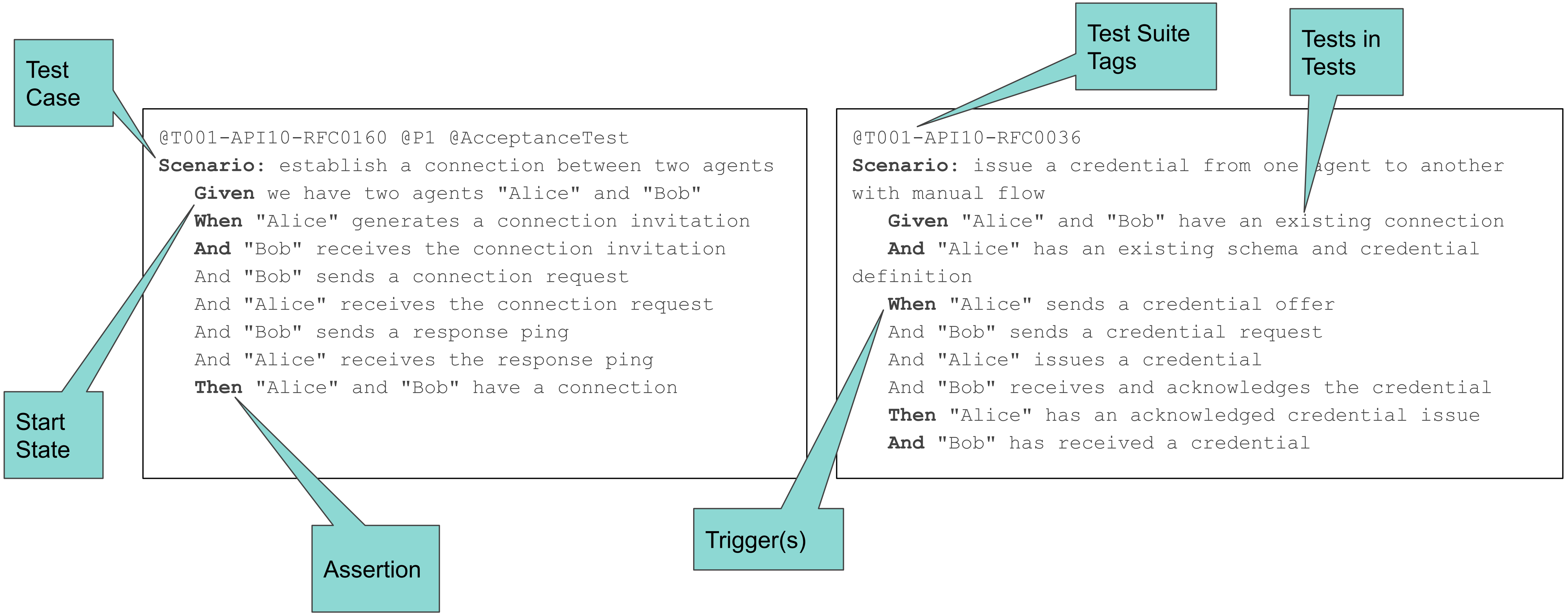
Terminology

- **Component under Test (CUT)** — the externally developed agent or agent framework
- **Backchannel** — code to convert test harness requests to instructions to the CUT
- **Test Agent (TA)** — a docker image containing the CUT, the backchannel and any other software needed to run the Test Agent

Test Writer's Process: From RFC to Tests

- Analyze the RFC
  - Document the set of test cases
- Create the Gherkin (BDD) code — features
- Define the configuration information
  - API Calls, data values, expected results
- Build the feature steps in Python

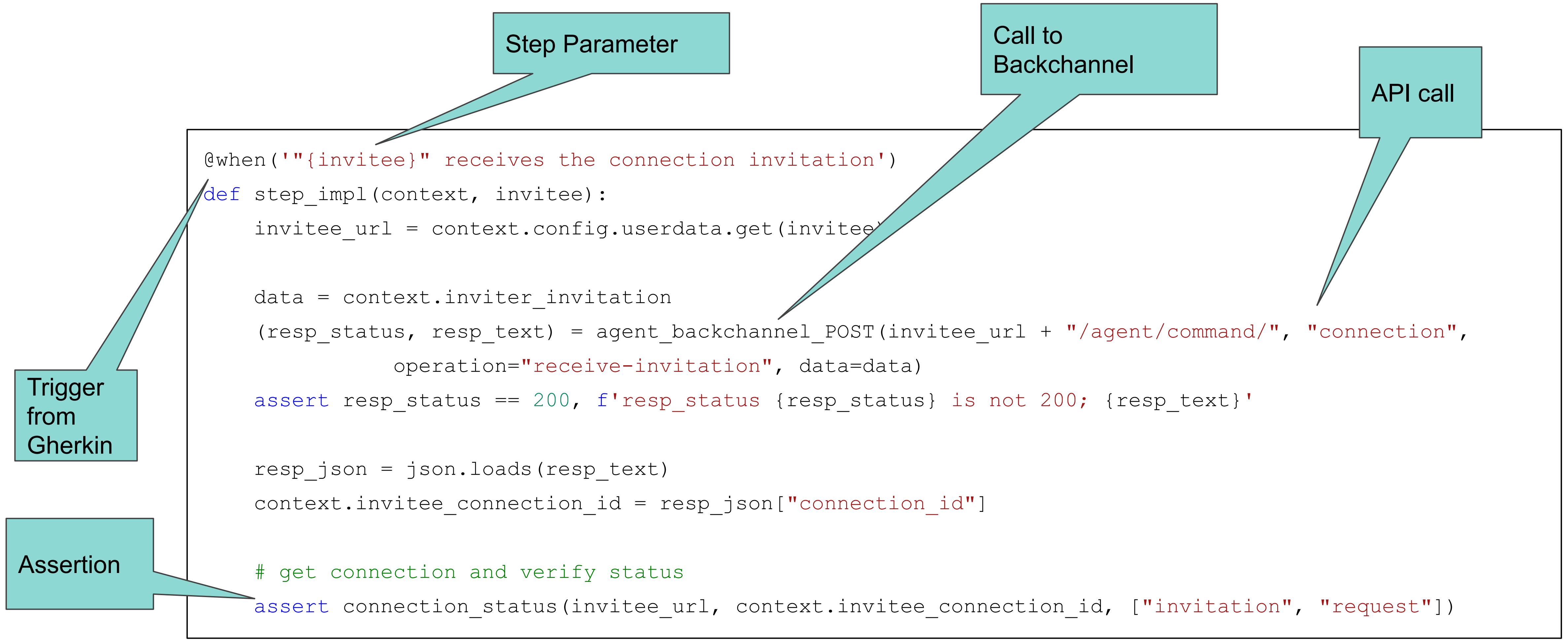
Examples of Gherkin BDD tests



```
@T001-API10-RFC0160 @P1 @AcceptanceTest
Scenario: establish a connection between two agents
   Given we have two agents "Alice" and "Bob"
   When "Alice" generates a connection invitation
   And "Bob" receives the connection invitation
   And "Bob" sends a connection request
   And "Alice" receives the connection request
   And "Bob" sends a response ping
   And "Alice" receives the response ping
   Then "Alice" and "Bob" have a connection
```

```
@T001-API10-RFC0036
Scenario: issue a credential from one agent to another
with manual flow
   Given "Alice" and "Bob" have an existing connection
   And "Alice" has an existing schema and credential
definition
   When "Alice" sends a credential offer
   And "Bob" sends a credential request
   And "Alice" issues a credential
   And "Bob" receives and acknowledges the credential
   Then "Alice" has an acknowledged credential issue
   And "Bob" has received a credential
```

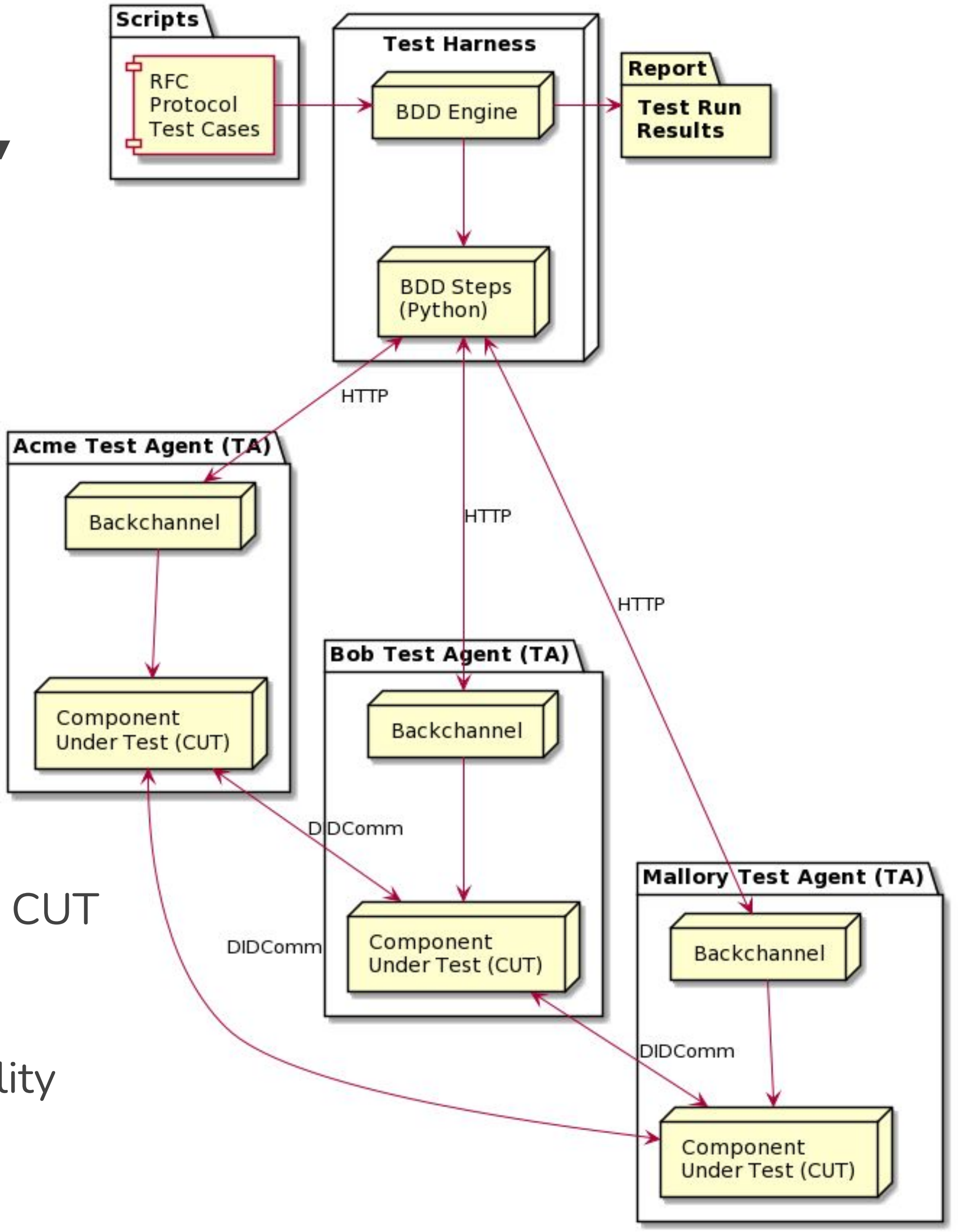
Gherkin Elements become Python Steps



```
@when('"{invitee}" receives the connection invitation')
def step_impl(context, invitee):
    invitee_url = context.config.userdata.get(invitee)

    data = context.inviter_invitation
    (resp_status, resp_text) = agent_backchannel_POST(invitee_url + "/agent/command/", "connection",
              operation="receive-invitation", data=data)
    assert resp_status == 200, f'resp_status {resp_status} is not 200; {resp_text}'

    resp_json = json.loads(resp_text)
    context.invitee_connection_id = resp_json["connection_id"]

    # get connection and verify status
    assert connection_status(invitee_url, context.invitee_connection_id, ["invitation", "request"])
```

“Component Under Test” (CUT)Backchannels

- Integration between Test Harness & Test Agent
    - Might be an agent framework
        - e.g. ACA-Py, aries-framework-dotnet
    - Might be a full agent
        - Framework + Controller
    - Might be a mobile agent
- Test Harness makes requests via API
- Backchannel converts requests to instructions to the CUT
    - ACA-Py — calls the HTTP or websocket admin API
    - VCX — calls embedded VCX code
- Standardized docker images enables interchangeability
    - Dockerfile with set port layout, naming conventions



Runtime Binding of Role to Test Agent (TA)

- Invocation:
  - `./manage -a acapy -b vcx -m acapy -t @AcceptanceTest -t ~@wip`
    - Acme and Mallory will be played by the “acapy” TA
    - Bob will be played by the “vcx” TA
    - We’ll execute only the tests tagged with “@`AcceptanceTest`” but not those with “@wip”
- Demo
  - Start agents
  - Run tests by tags
  - Stop agents
  - Report results
- Easy to add to CI pipeline
- Flexible for now — we’ll see how it evolves.
  - PRs welcome!!

Priorities and Vision

- More(!!) tests across more RFCs
    - Goal is to have full AIP 1.0+ coverage
- aries-framework-dotnet backchannel
- Documentation—making it easy for others to create backchannels, run tests
    - Debugging when running tests; adding tracing support
- CI Automation—adding this to PR pipelines
- Mobile agent testing—maybe using BrowserStack?
- Testing full agents (frameworks plus a custom controller)
- Aries Interop Lab—a place where all the agents can play together
    - Something like the Telecom Industry uses (this and this)

Questions?

- Offers to help?