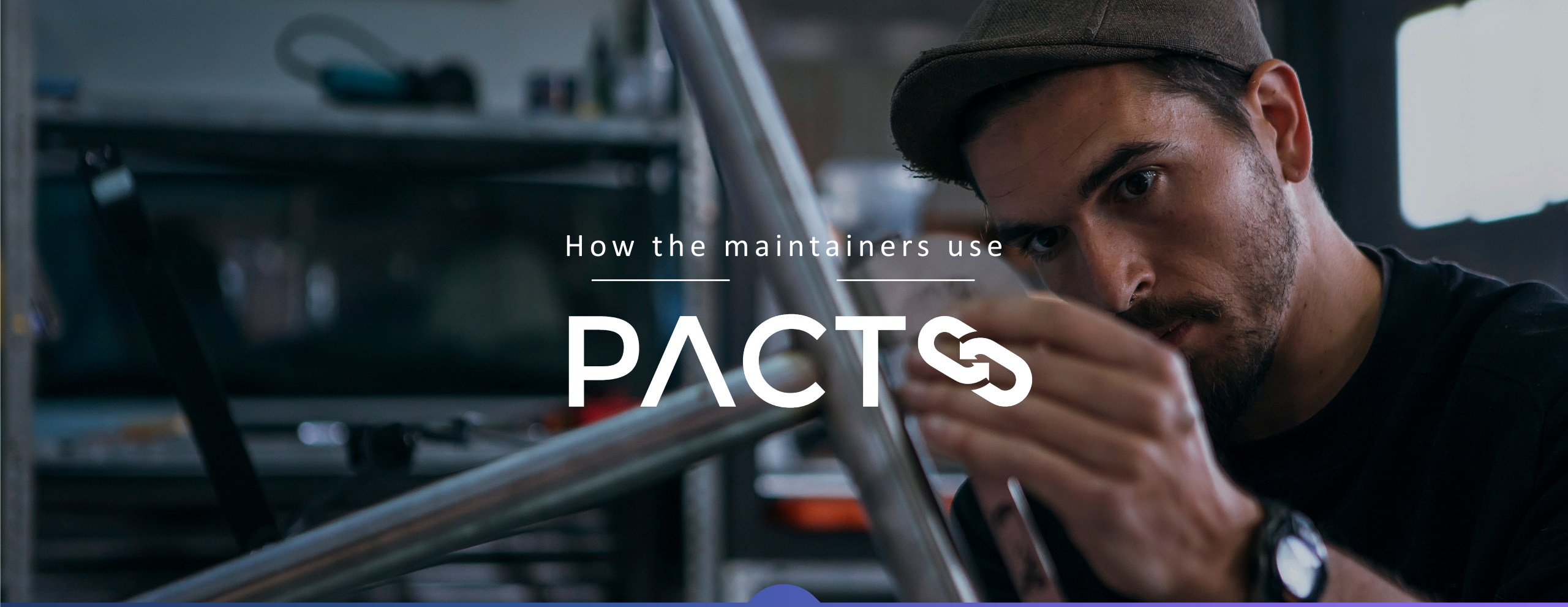
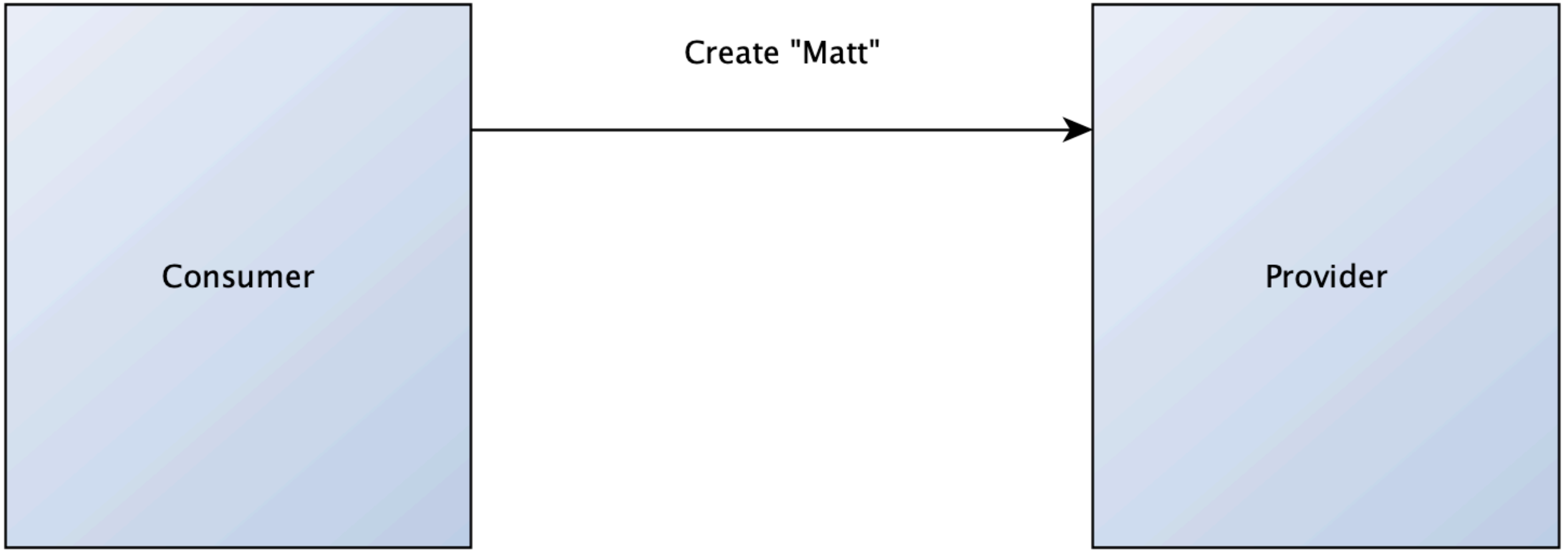


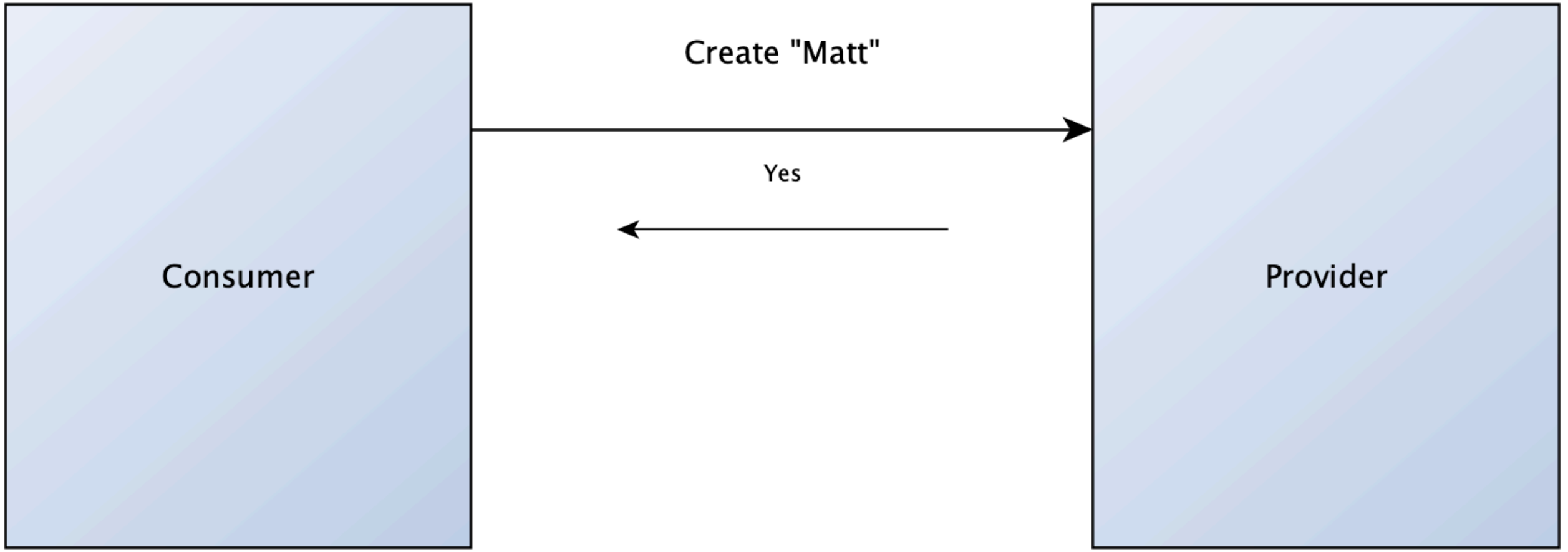
How the maintainers use PACTS

Well, how one maintainer uses it

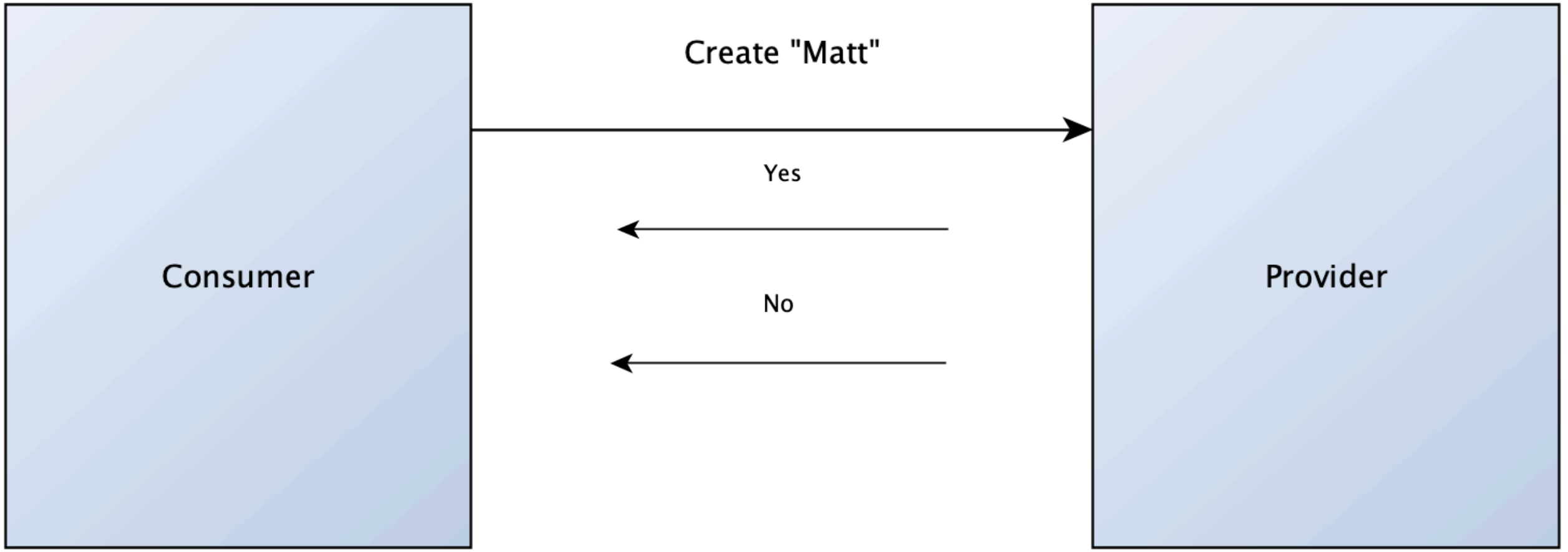
Timothy Jones



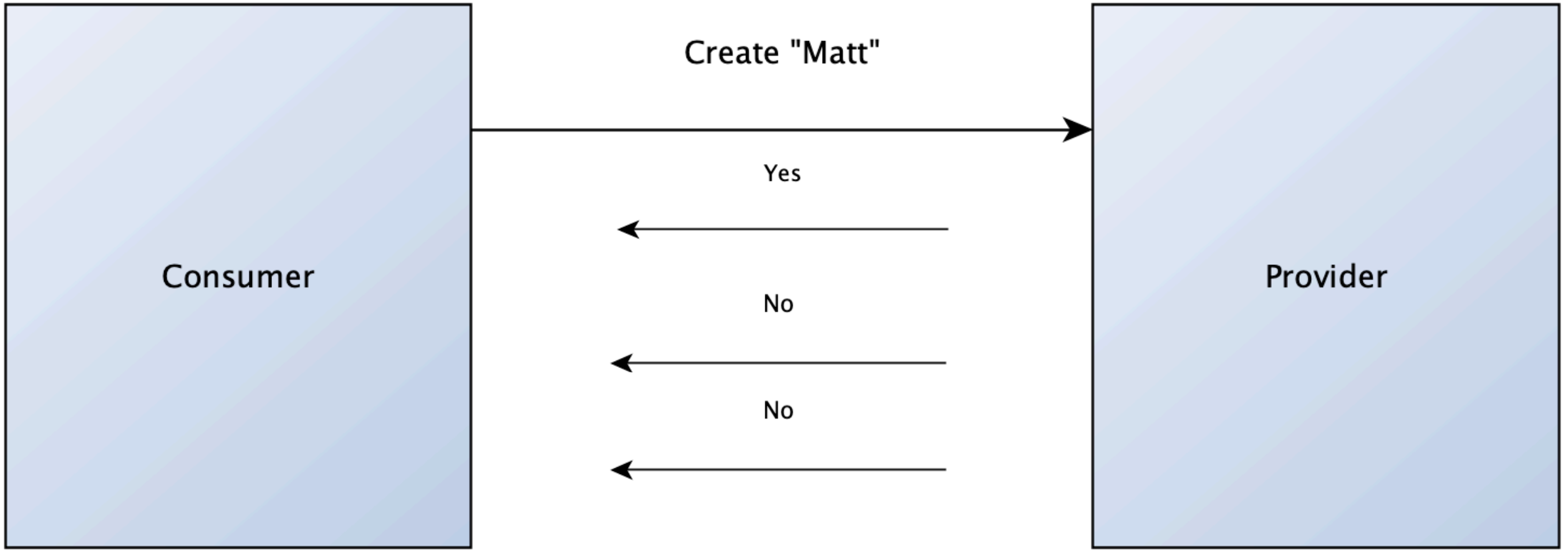
Background: Contract tests are not functional tests



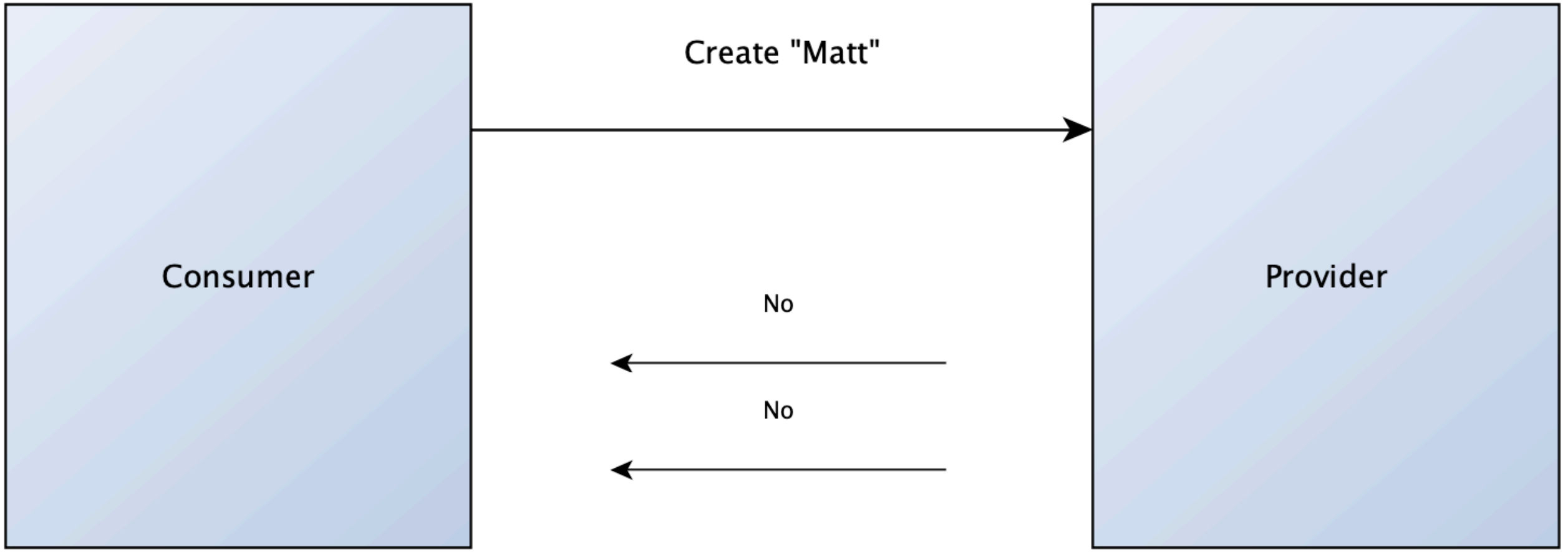
Background: Contract tests are not functional tests



Background: Contract tests are not functional tests



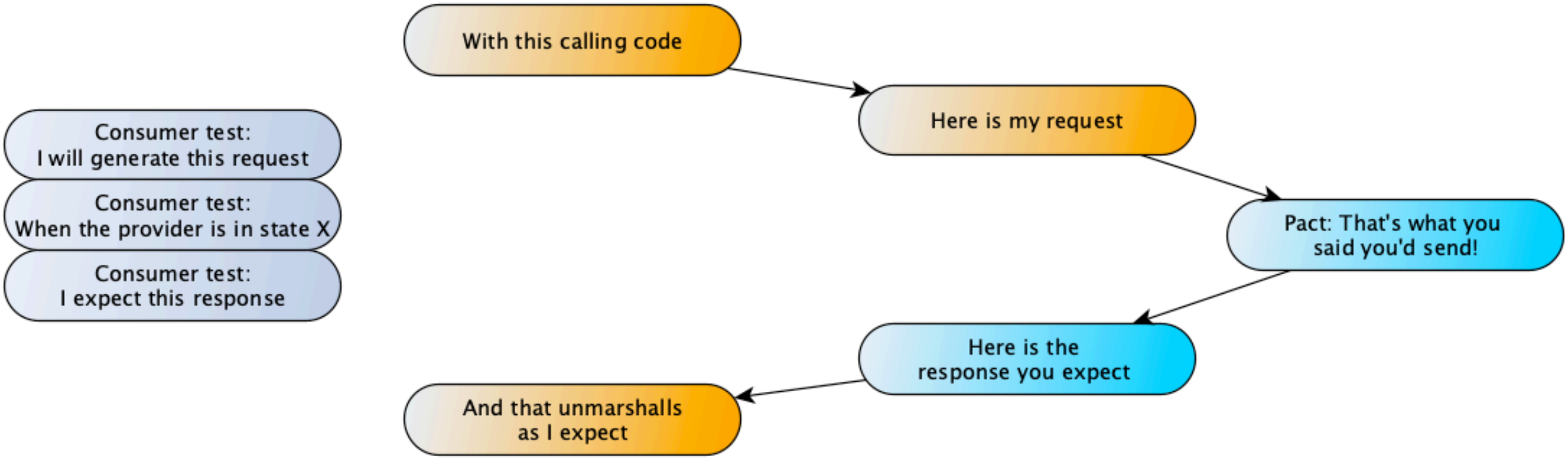
Background: Contract tests are not functional tests



Background: Contract tests are not functional tests

Background: Contract tests are not functional tests

Background: Contract tests are not functional tests

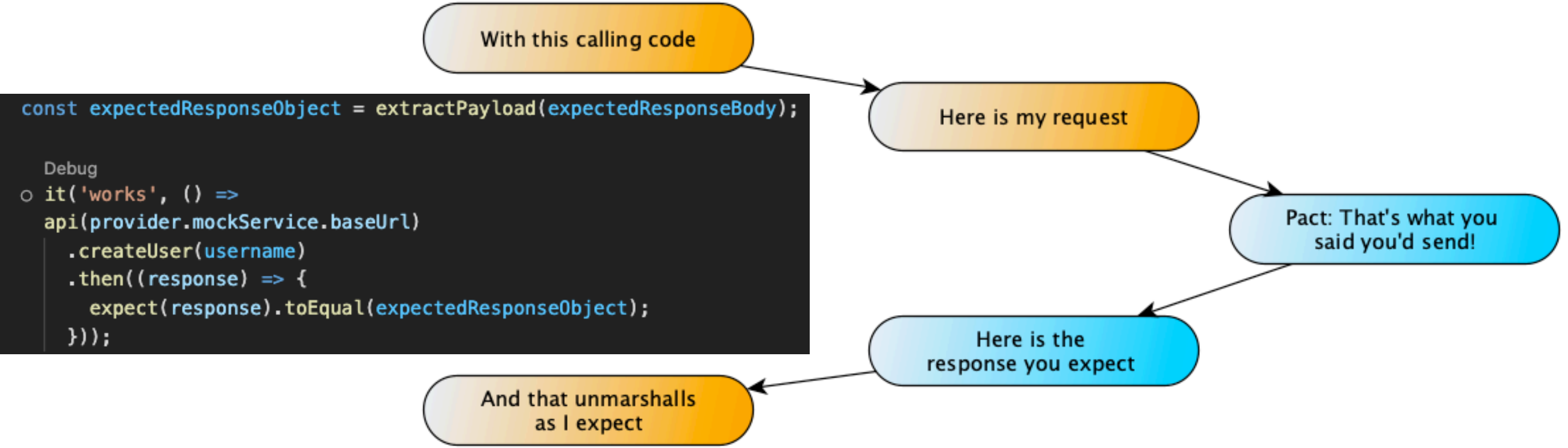

With this calling code
Here is my request
Pact: That's what you said you'd send!
Here is the response you expect
And that unmarshalls as I expect
Consumer test: I will generate this request
Consumer test: When the provider is in state X
Consumer test: I expect this response

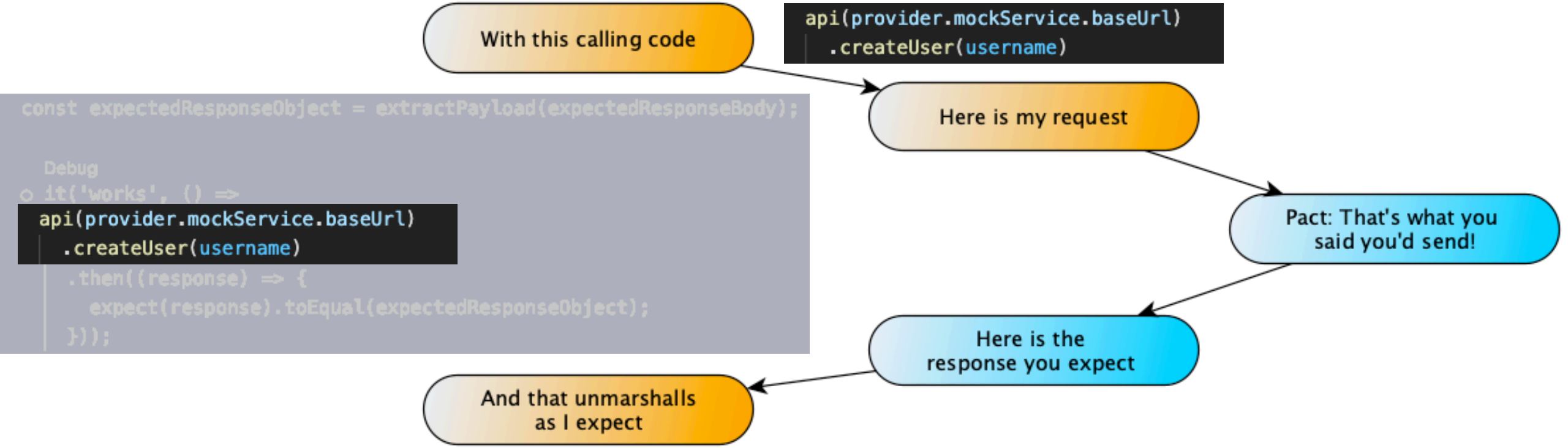
Consumer test:
I will generate this request

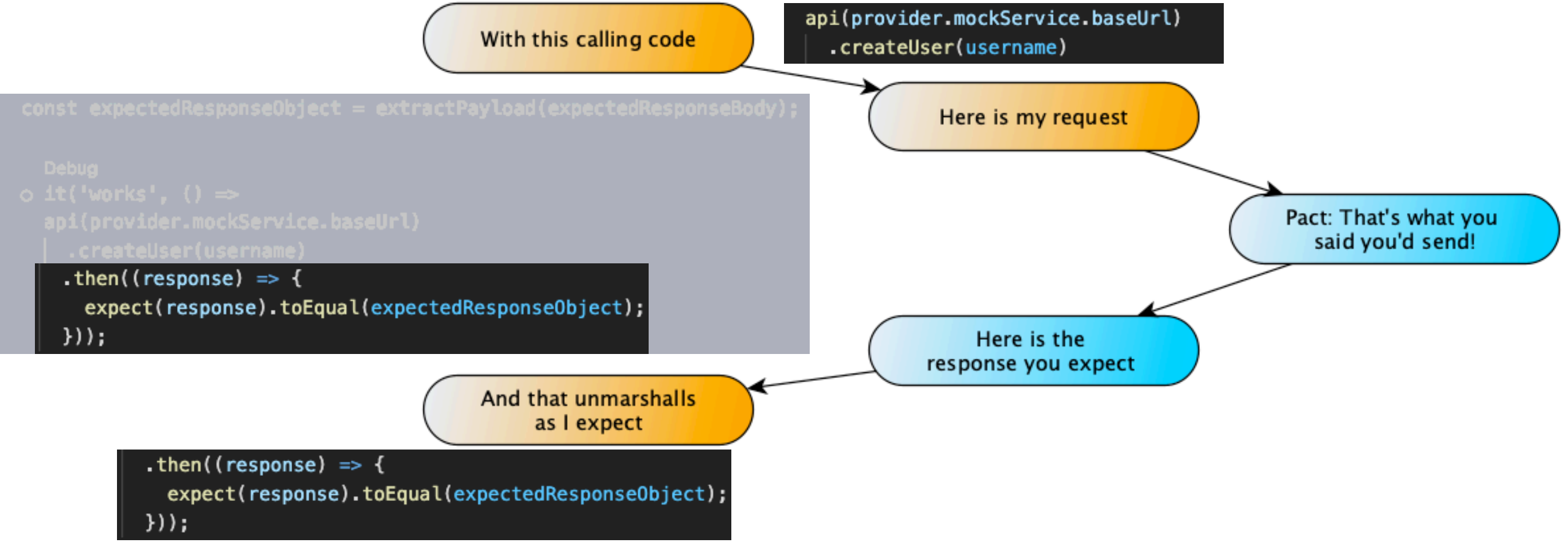
Consumer test:
When the provider is in state X

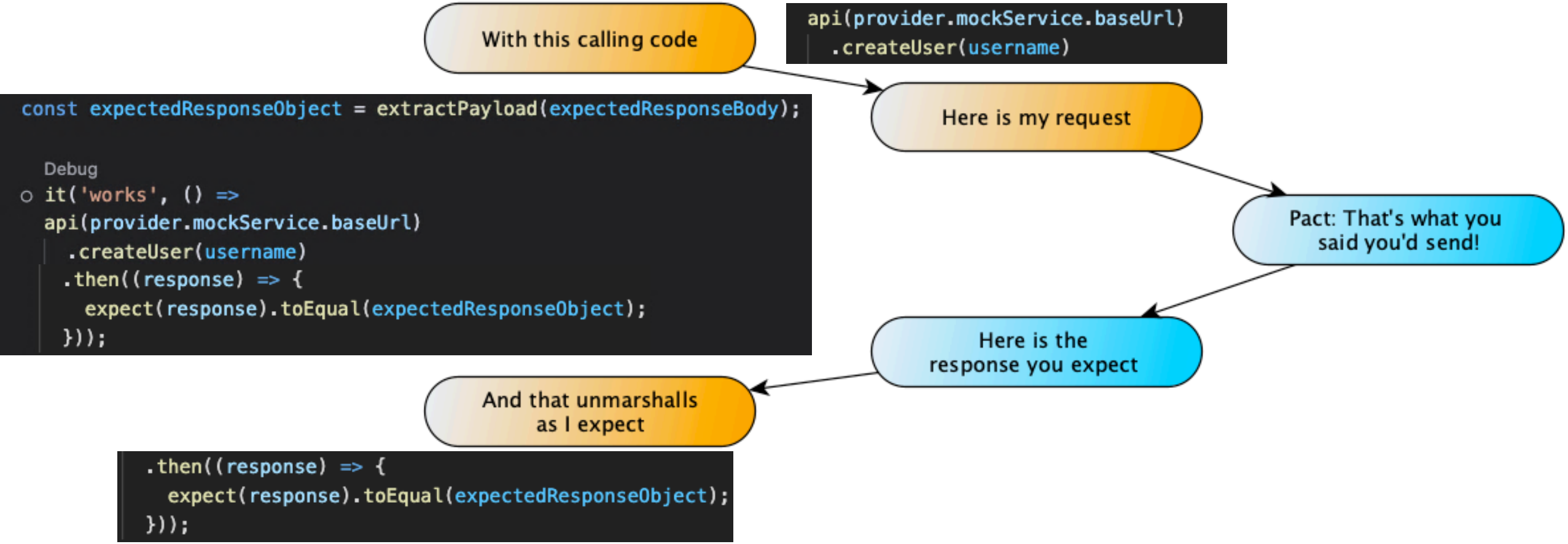
Consumer test:
I expect this response

```
const username = 'Matt';

const expectedResponseBody = {
  error: like('The username Matt is already taken'),
};

beforeEach(() =>
  provider.addInteraction({
    uponReceiving: 'A request to create the user "Matt"',
    withRequest: {
      method: 'POST',
      path: '/users',
      body: { username },
    },
    state: 'Username Matt is invalid',
    willRespondWith: {
      status: 400,
      body: expectedResponseBody,
    },
  })
);
```


With this calling code
Here is my request
Pact: That's what you said you'd send!
Here is the response you expect
And that unmarshalls as I expect
const expectedResponseObject = extractPayload(expectedResponseBody);
Debug
it('works', () =>
api(provider.mockService.baseUrl)
.createUser(username)
.then((response) => {
expect(response).toEqual(expectedResponseObject);
}));

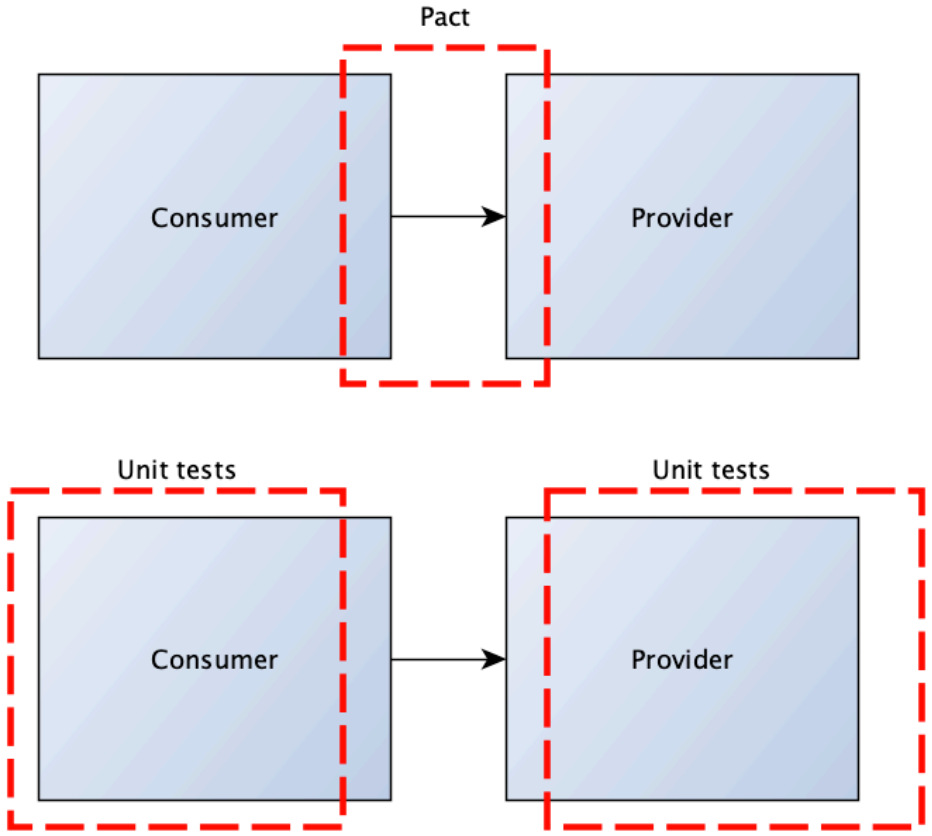

With this calling code
api(provider.mockService.baseUrl)
.createUser(username)
Here is my request
Pact: That's what you said you'd send!
Here is the response you expect
And that unmarshalls as I expect
const expectedResponseObject = extractPayload(expectedResponseBody);
Debug
it('works', () =>
api(provider.mockService.baseUrl)
.createUser(username)
.then((response) => {
expect(response).toEqual(expectedResponseObject);
}));


With this calling code
api(provider.mockService.baseUrl)
.createUser(username)
Here is my request
Pact: That's what you said you'd send!
Here is the response you expect
And that unmarshalls as I expect
const expectedResponseObject = extractPayload(expectedResponseBody);
Debug
it('works', () =>
api(provider.mockService.baseUrl)
.createUser(username)
.then((response) => {
expect(response).toEqual(expectedResponseObject);
}));
.then((response) => {
expect(response).toEqual(expectedResponseObject);
}));

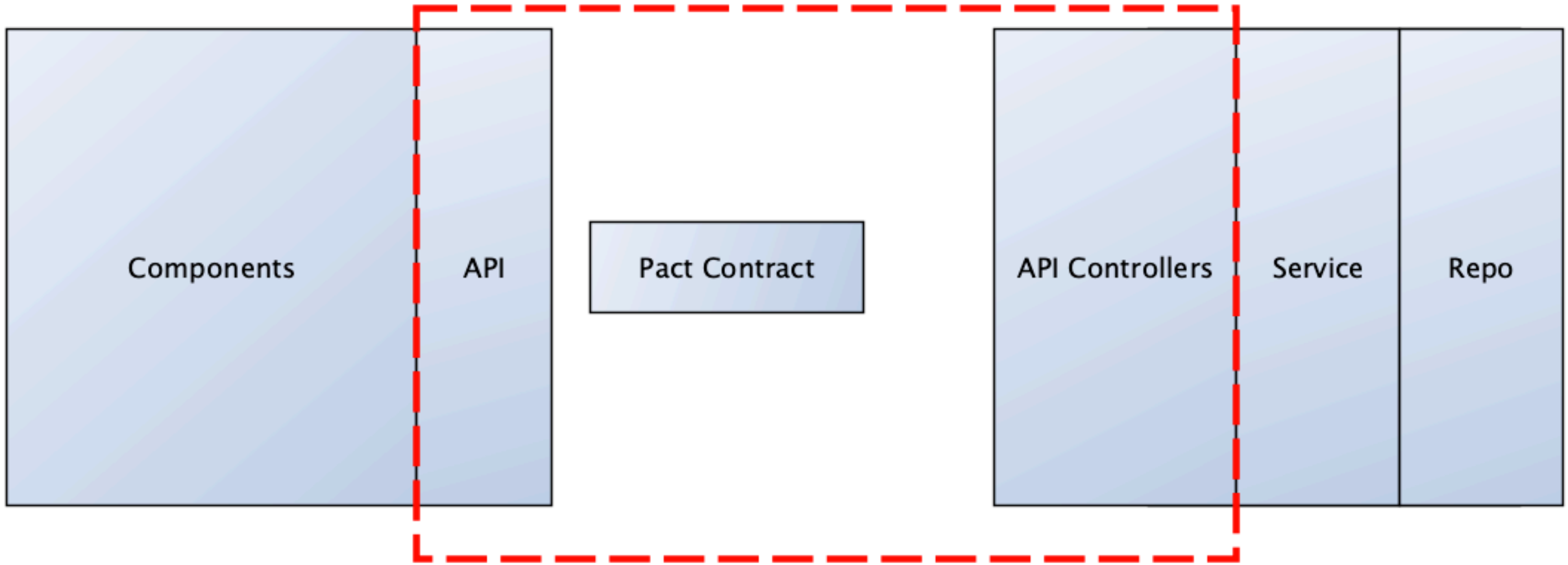

With this calling code
api(provider.mockService.baseUrl)
.createUser(username)
const expectedResponseObject = extractPayload(expectedResponseBody);
Debug
it('works', () =>
api(provider.mockService.baseUrl)
.createUser(username)
.then((response) => {
expect(response).toEqual(expectedResponseObject);
}));
Here is my request
Pact: That's what you said you'd send!
Here is the response you expect
And that unmarshalls as I expect
.then((response) => {
expect(response).toEqual(expectedResponseObject);
}));

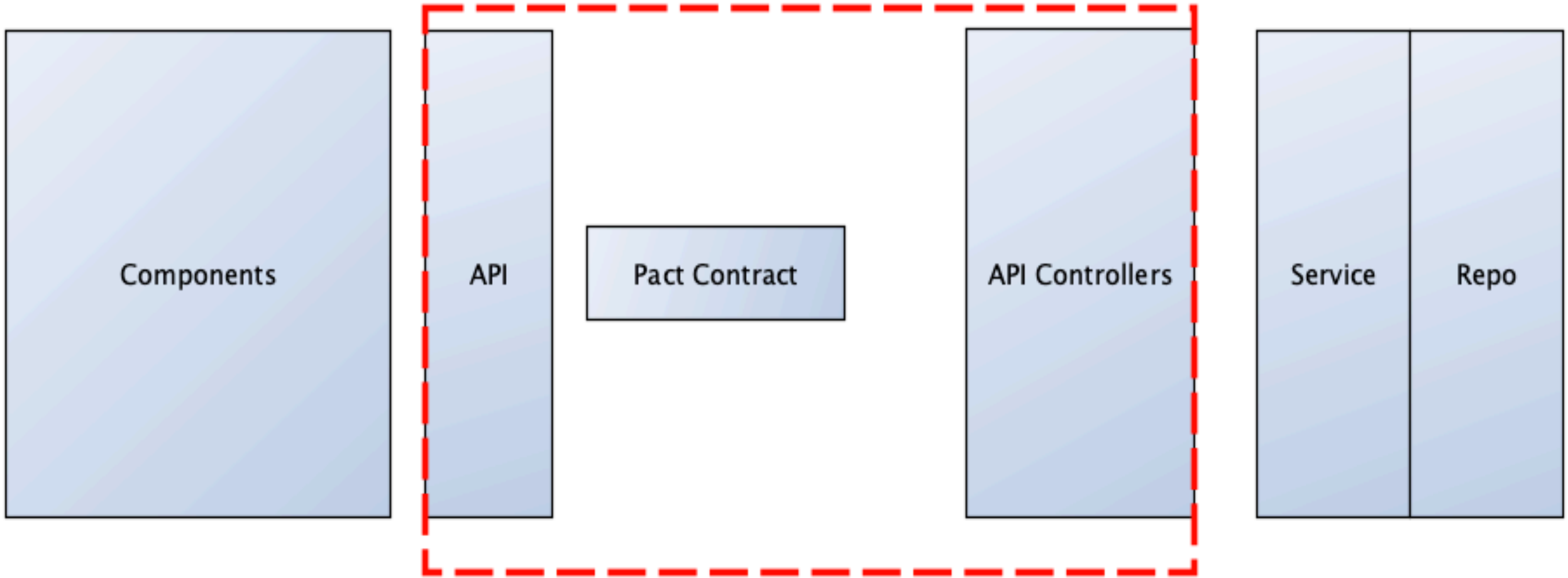
```
const username = 'Matt';

const expectedResponseBody = {
  error: like('The username Matt is already taken'),
};

beforeEach(() =>
  provider.addInteraction({
    uponReceiving: 'A request to create the user "Matt"',
    withRequest: {
      method: 'POST',
      path: '/users',
      body: { username },
    },
    state: 'Username Matt is invalid',
    willRespondWith: {
      status: 400,
      body: expectedResponseBody,
    },
  })
);
```

```
const expectedResponseObject = extractPayload(expectedResp

  Debug
○ it('works', () =>
  api(provider.mockService.baseUrl)
    .createUser(username)
    .then((response) => {
      expect(response).toEqual(expectedResponseObject);
    }));
```



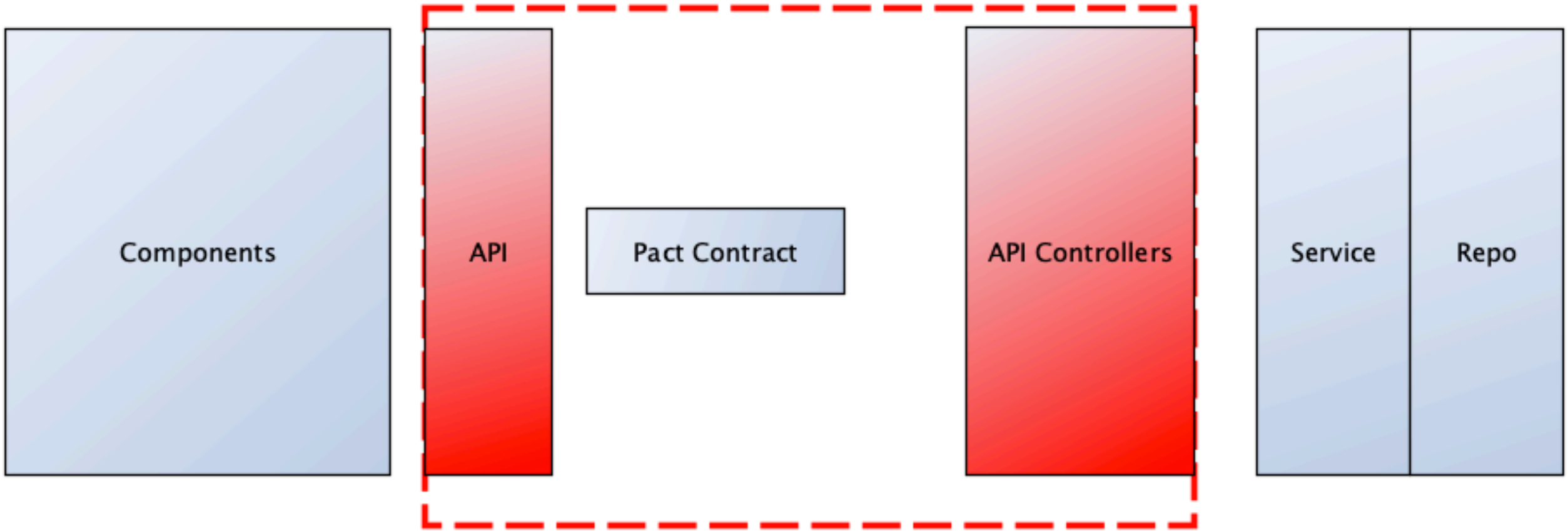
Background: Contract tests are not functional tests

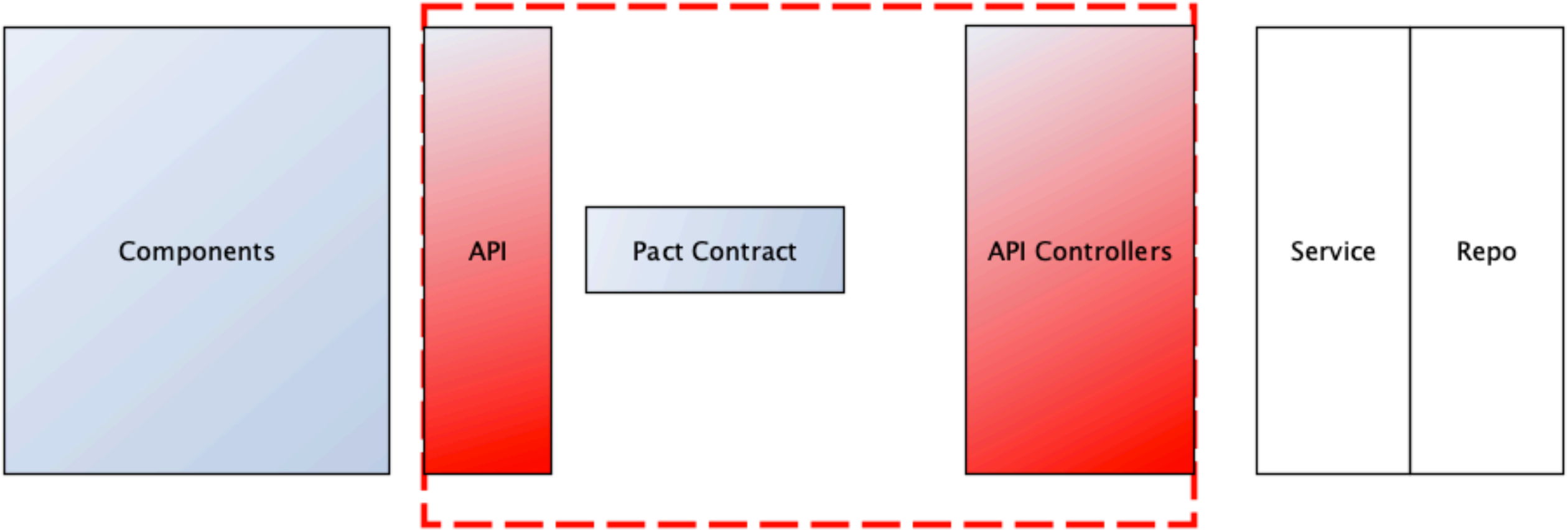

Components
API
Pact Contract
API Controllers
Service
Repo

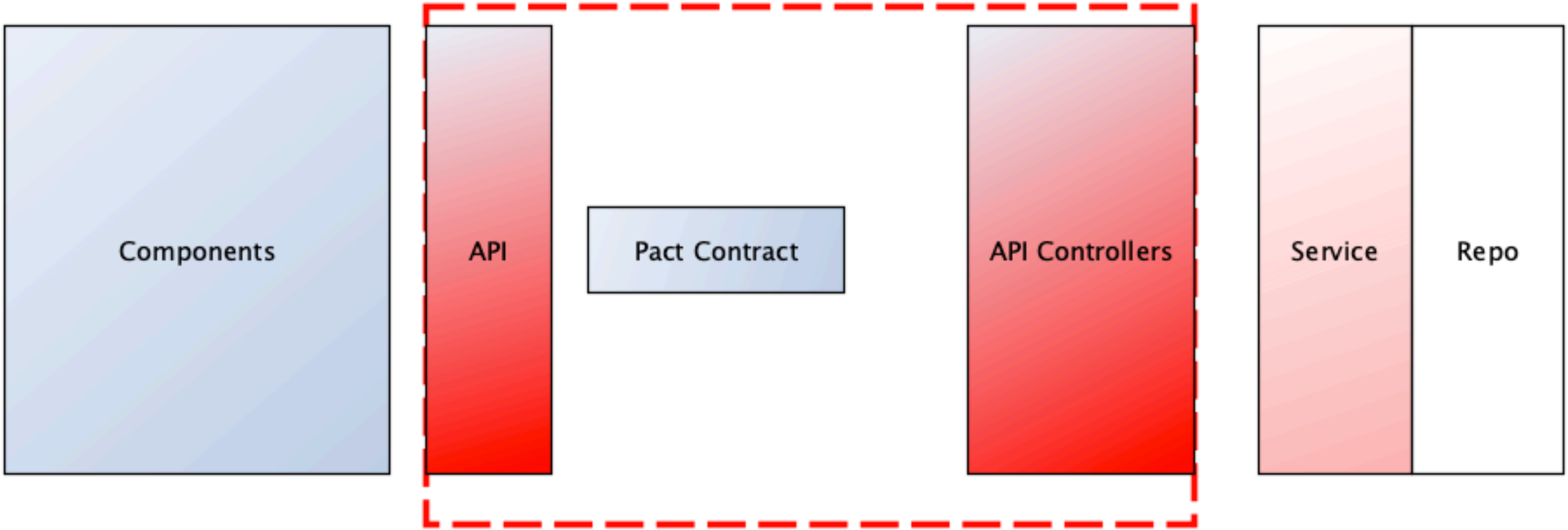

Components
API
Pact Contract
API Controllers
Service
Repo

So:
Contract tests are not functional tests

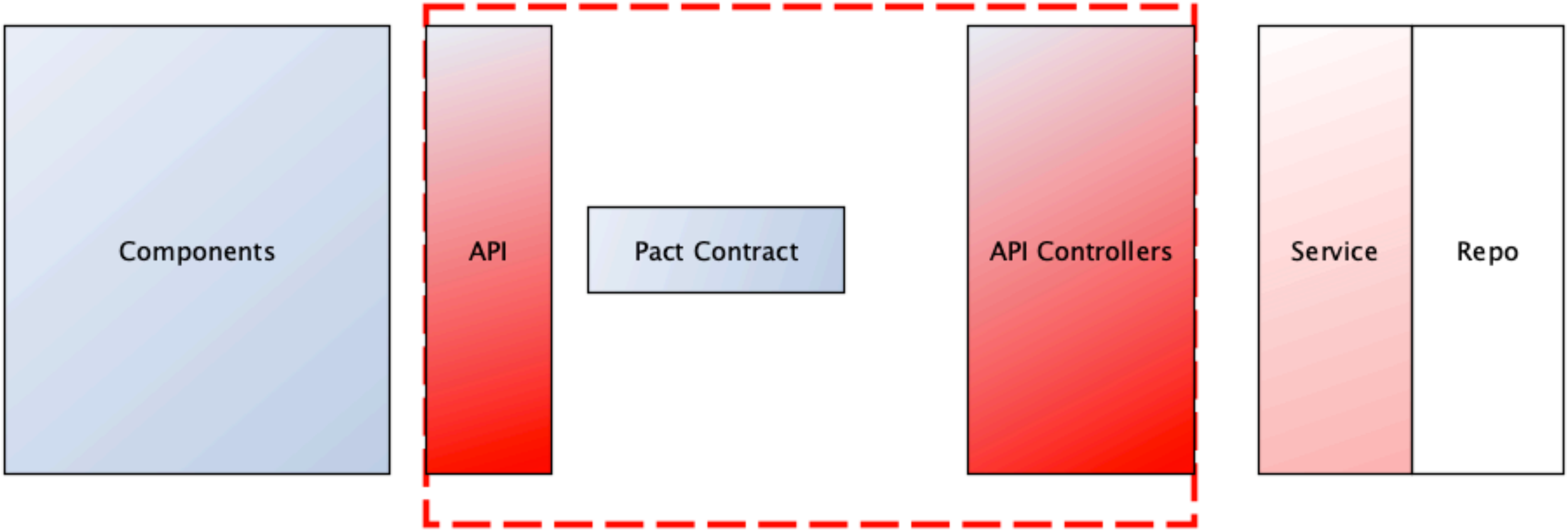
But:
It is good if they have functional coverage


Components
API
Pact Contract
API Controllers
Service
Repo


Components
API
Pact Contract
API Controllers
Service
Repo


Components
API
Pact Contract
API Controllers
Service
Repo

Test quality:
Connecting the tests and mocks with fixtures


Components
API
Pact Contract
API Controllers
Service
Repo

address.pact.fixtures.js

```
export const createUserResponseBody = {
  error: like('The username Matt is already taken'),
};

export const createUserFailure = extractPayload(createUserResponseBody);
```

address.pact.fixtures.js

```
export const createUserResponseBody = {
  error: like('The username Matt is already taken'),
};

export const createUserFailure = extractPayload(createUserResponseBody);
```

address.spec.pact.js

```
    willRespondWith: {
      status: 400,
      body: createUserResponseBody,
    },
  })
);

Debug
it('works', () =>
api(provider.mockService.baseUrl)
  .createUser(username)
  .then((response) => {
    expect(response).toEqual(createUserFailure);
  }));
```

address.pact.fixtures.js

```
export const createUserResponseBody = {
  error: like('The username Matt is already taken'),
};

export const createUserFailure = extractPayload(createUserResponseBody);
```

<wherever you need a mock>.spec.js

```
export const mockApi = {
  createUser: jest.fn().mockResolvedValue(createUserFailure),
};
```

```
const createUserResponseBody = {
  error: like('The username Matt is already taken'),
};

export const pactResponses = {
  createUserFailure: {
    willRespondWith: {
      status: 400,
      body: createUserResponseBody,
    },
  },
};

export const createUserFailure = extractPayload(createUserResponseBody);
```

```
beforeEach(() =>
  provider.addInteraction({
    uponReceiving: 'A request to create the user "Matt"',
    withRequest: {
      method: 'POST',
      path: '/users',
      body: { username },
    },
    state: 'Username Matt is invalid',
    ...pactResponses.createUserFailure,
  })
);
```

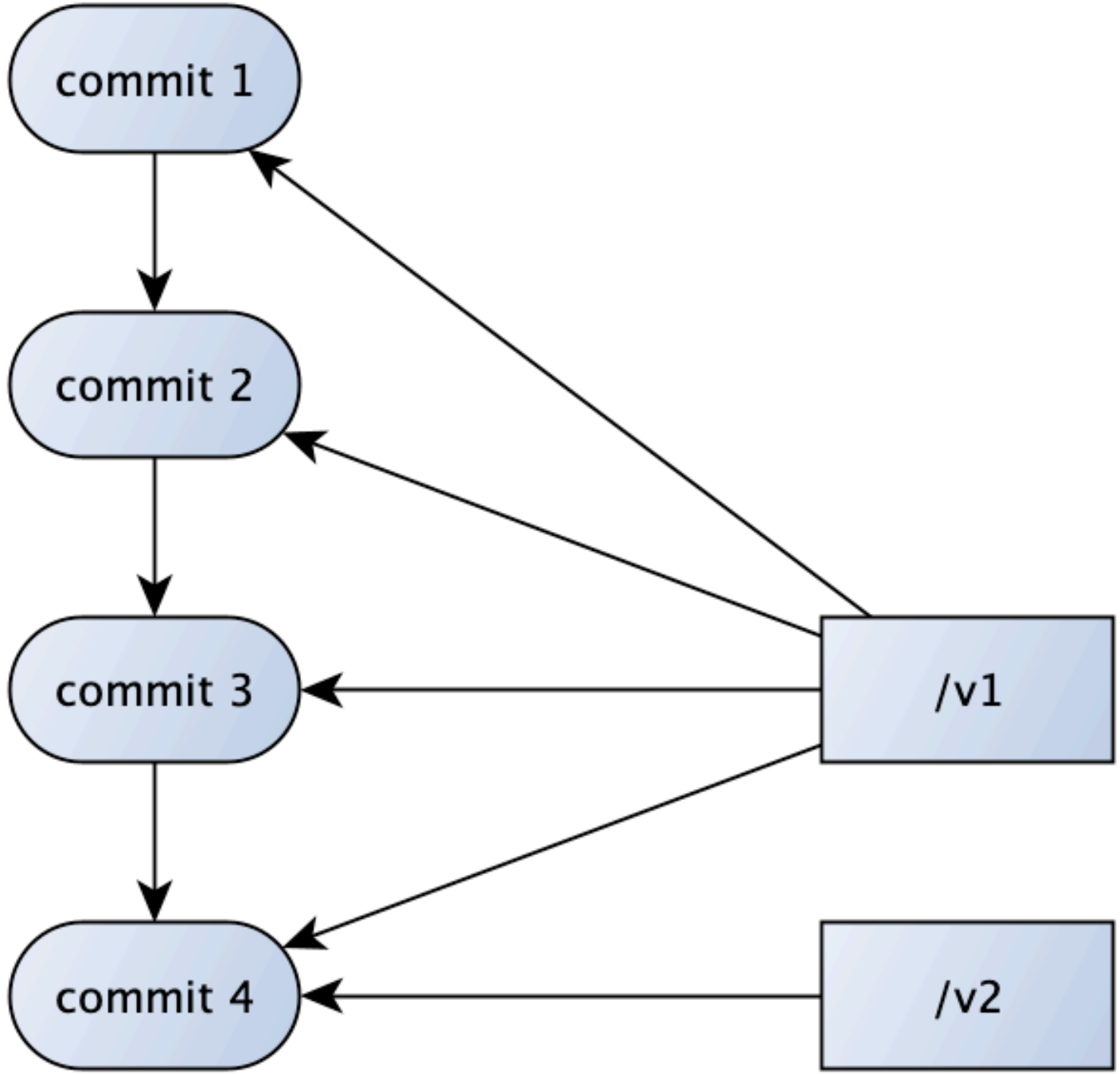
```
const requestToCreateUserMatt = {
  uponReceiving: 'A request to create the user "Matt"',
  withRequest: {
    method: 'POST',
    path: '/users',
    body: { username },
  },
};
```

```
const requestToCreateUserMatt = {
  uponReceiving: 'A request to create the user "Matt"',
  withRequest: {
    method: 'POST',
    path: '/users',
    body: { username },
  },
};
```

```
beforeEach(() =>
  provider.addInteraction({
    ...requestToCreateUserMatt,
    state: 'Username Matt is invalid',
    ...pactResponses.createUserFailure,
  })
);
```

Contentious claim:

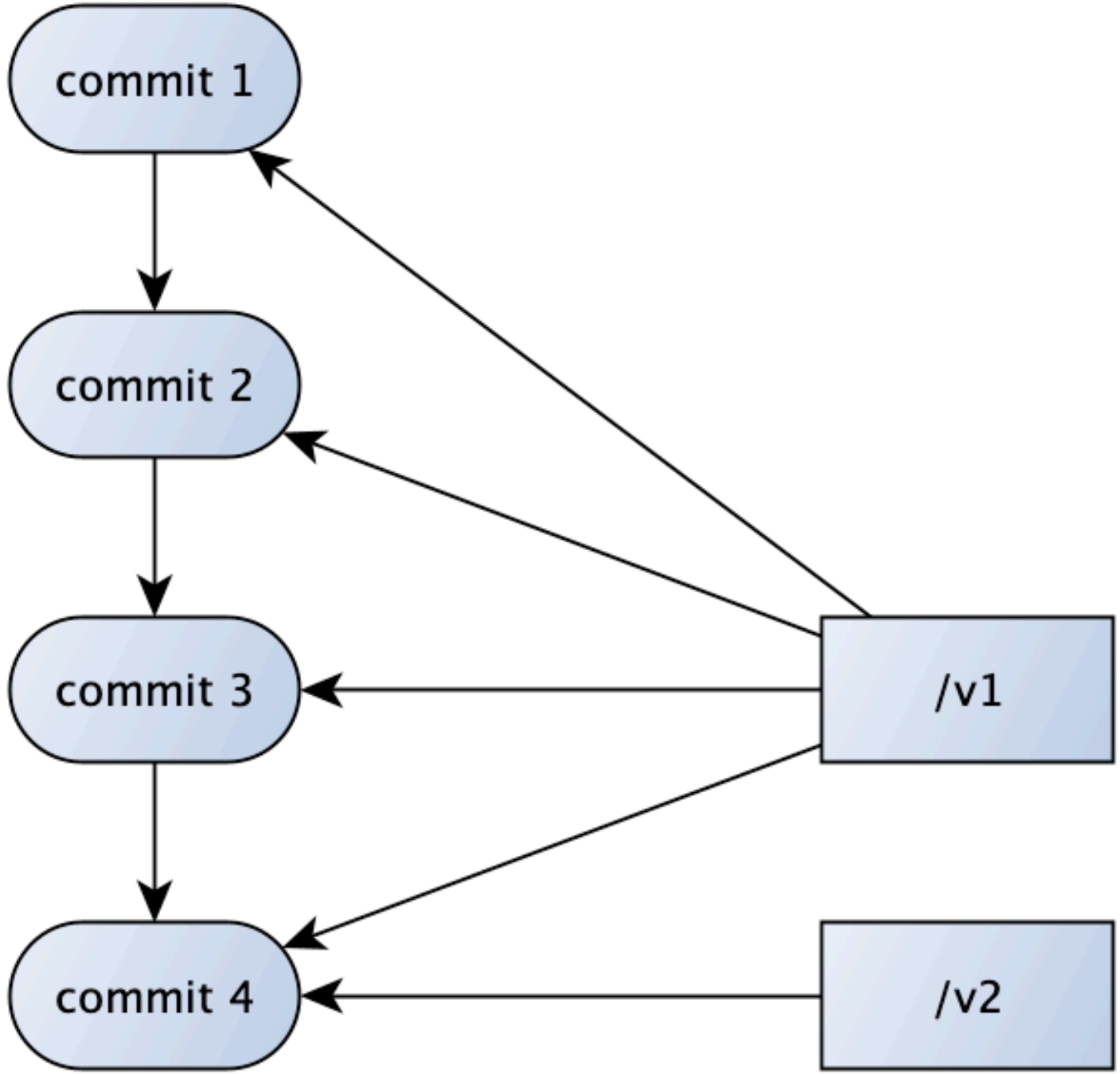
I suspect URLs like /v1 are antipatterns

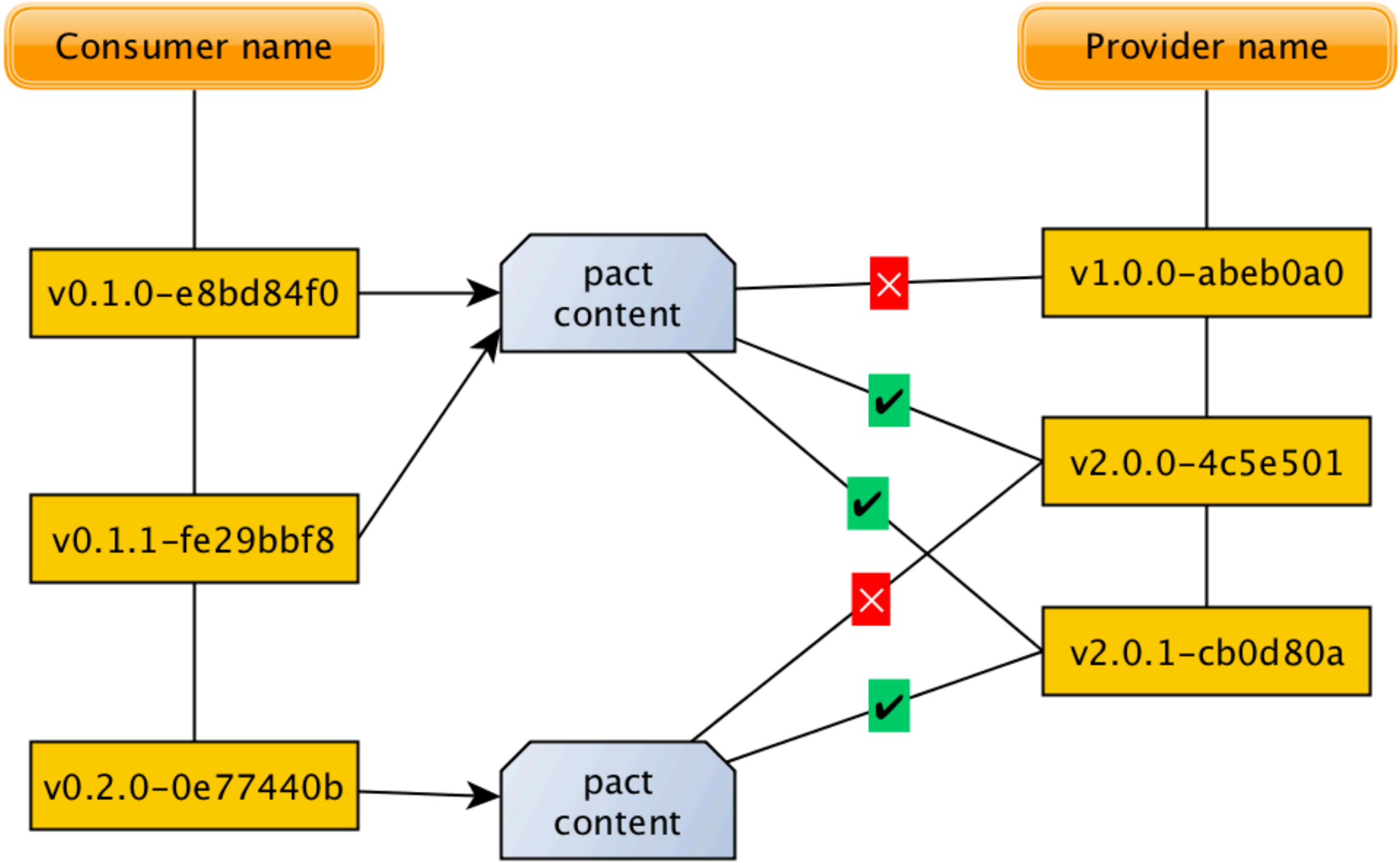

commit 1
commit 2
commit 3
commit 4
/v1
/v2

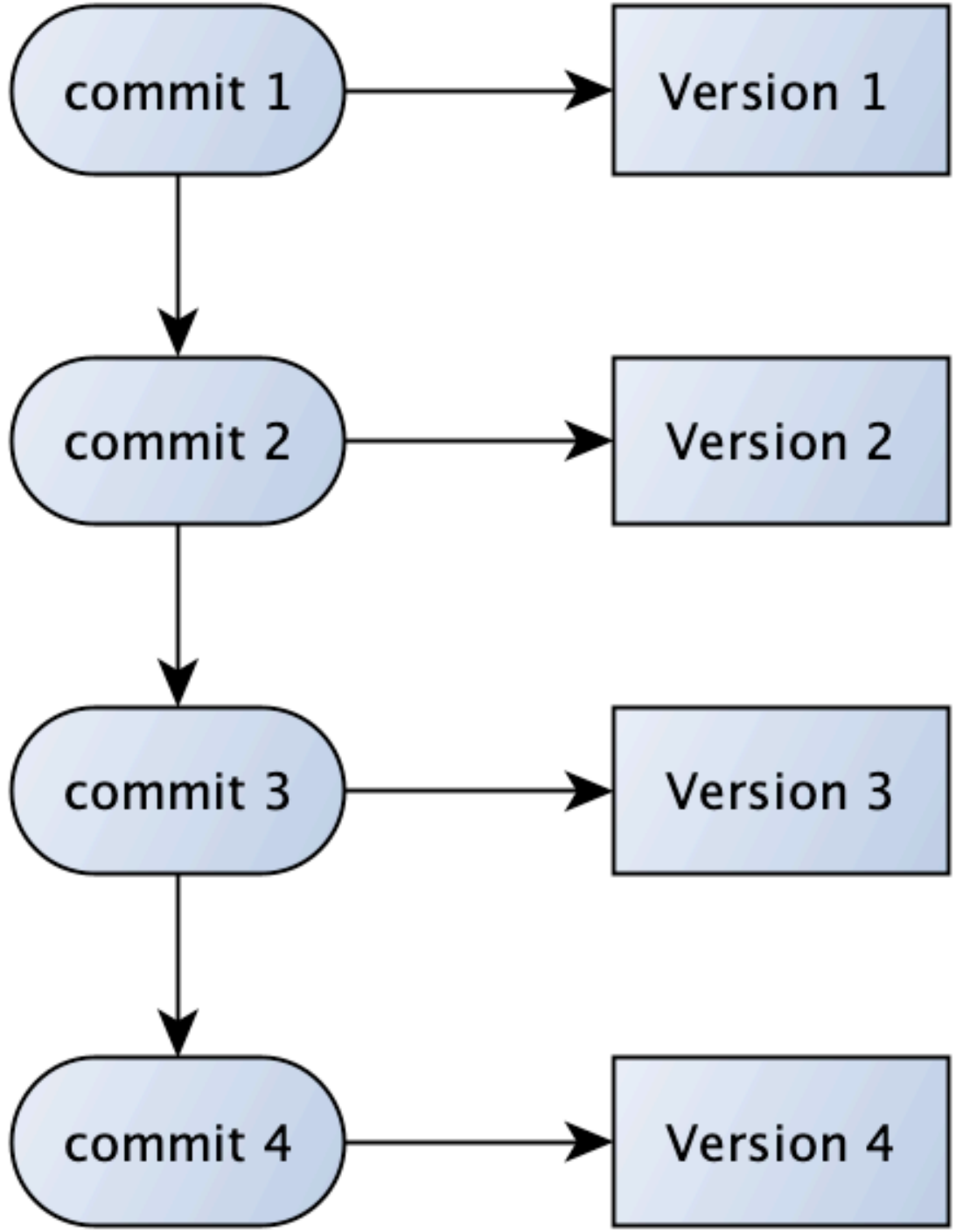
What is a breaking change?

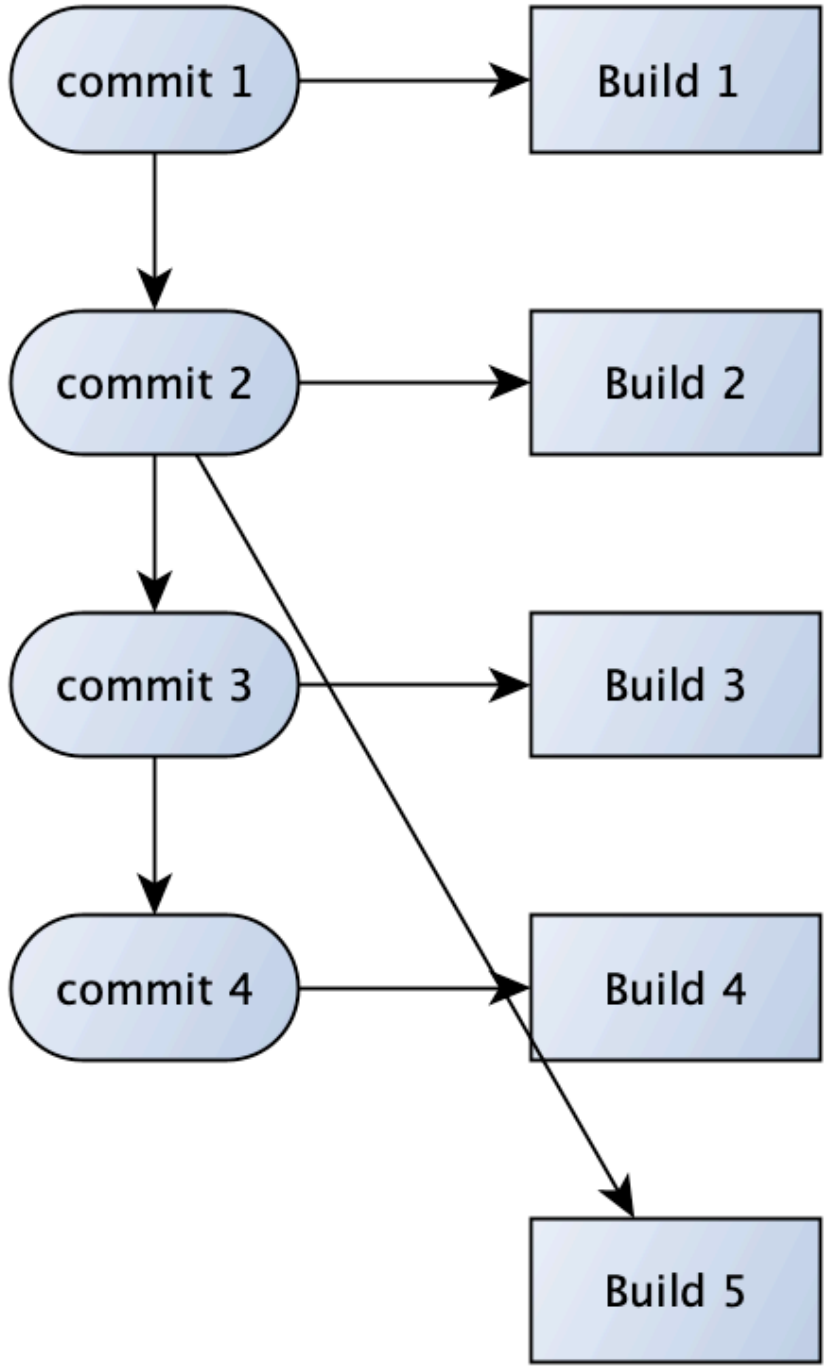
```
{
  "error": "Username is not allowed to have capital letters"
}
```

```
{
  "error": "Username is not allowed to have capital letters",
  "code": "BAD_REQUEST",
  "context": {
    "file": "UserService.java",
    "line": "34"
  }
}
```


commit 1
commit 2
commit 3
commit 4
/v1
/v2


Consumer name
Provider name
v0.1.0-e8bd84f0
v0.1.1-fe29bbf8
v0.2.0-0e77440b
pact content
pact content
v1.0.0-abeb0a0
v2.0.0-4c5e501
v2.0.1-cb0d80a


commit 1
Version 1
commit 2
Version 2
commit 3
Version 3
commit 4
Version 4

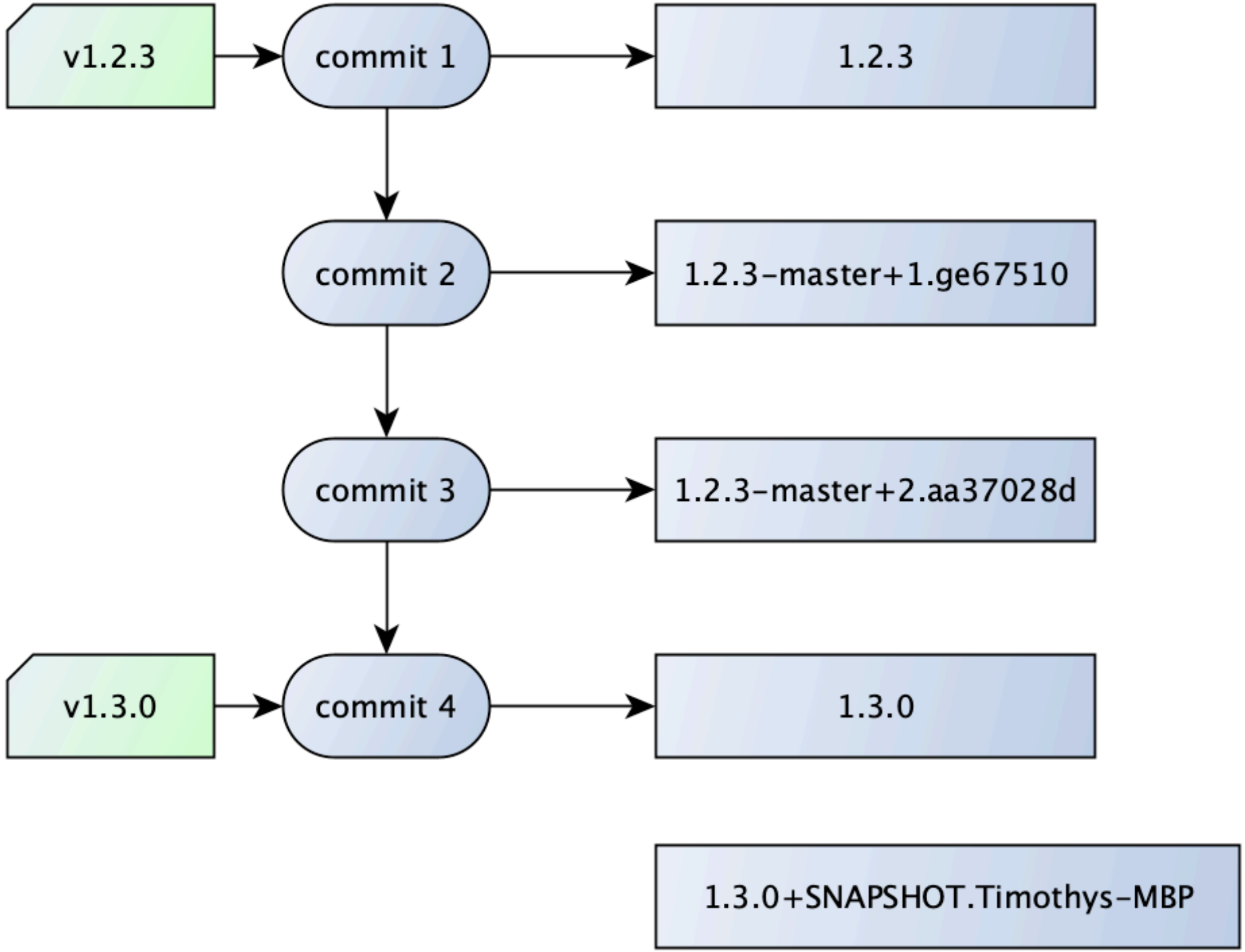

commit 1
Build 1
commit 2
Build 2
commit 3
Build 3
commit 4
Build 4
Build 5

```
<valid semver> ::= <version core>
                 | <version core> "-" <pre-release>
                 | <version core> "+" <build>
                 | <version core> "-" <pre-release> "+" <build>

<version core> ::= <major> "." <minor> "." <patch>
```

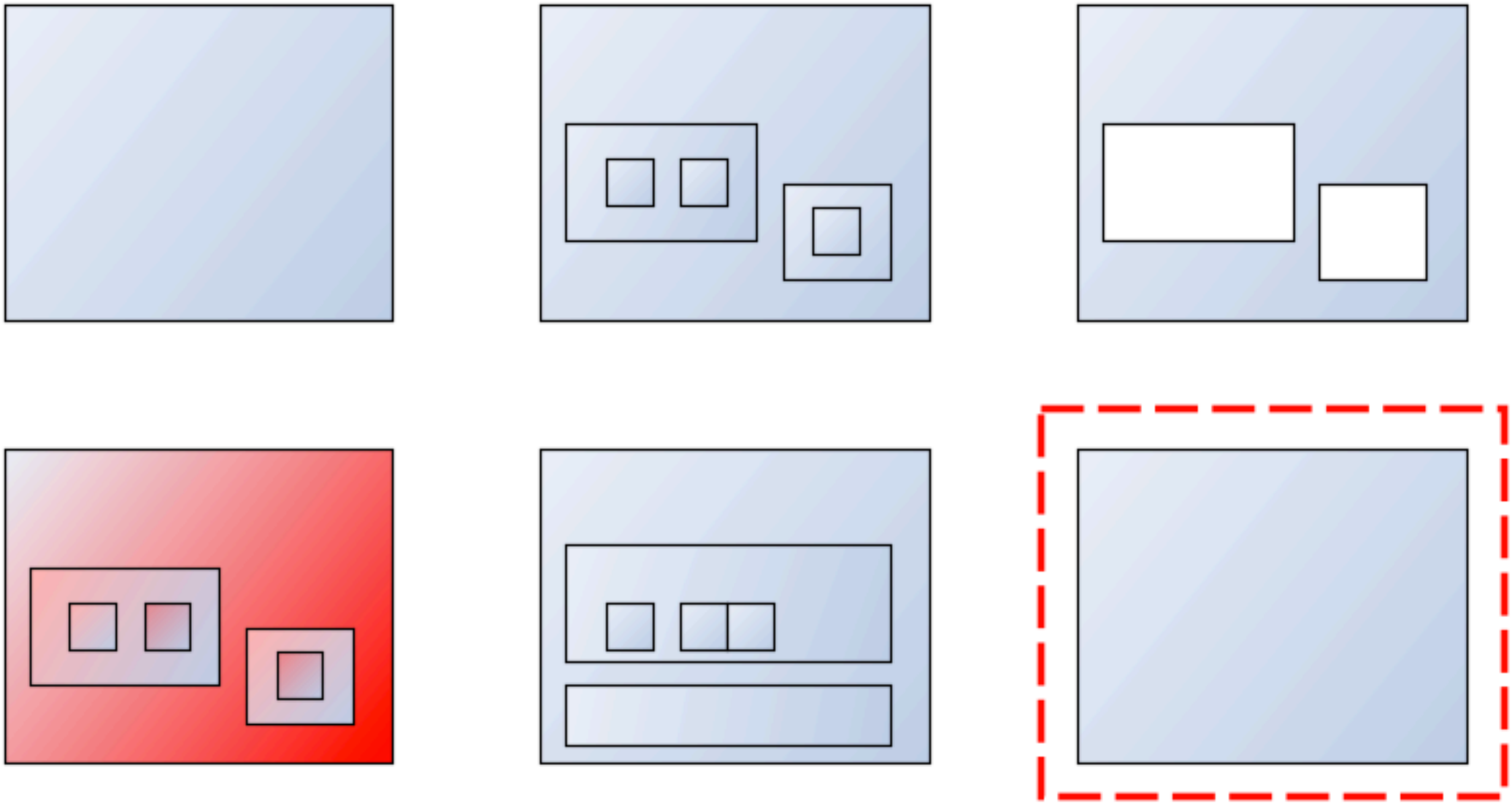
Contentious claim:
I suspect URLs like /v1 are antipatterns

Solution:
Nice versions from git history

```
<pre-release> ::= <dot-separated pre-release identifiers>

<dot-separated pre-release identifiers> ::= <pre-release identifier>
                                          | <pre-release identifier> "." <dot-sepa

<build> ::= <dot-separated build identifiers>

<dot-separated build identifiers> ::= <build identifier>
                                    | <build identifier> "." <dot-separated build

<pre-release identifier> ::= <alphanumeric identifier>
                           | <numeric identifier>

<build identifier> ::= <alphanumeric identifier>
                     | <digits>
```


v1.2.3
commit 1
1.2.3
commit 2
1.2.3-master+1.ge67510
commit 3
1.2.3-master+2.aa37028d
v1.3.0
commit 4
1.3.0
1.3.0+SNAPSHOT.Timothys-MBP

All tests are actually contract tests



What would I like to ask the maintainers for?

Easier config

Better errors from Pact-js

Integrated versioning

Contract tests aren’t functional tests, but it’s ok if they have some functional coverage

Fixtures can tie your api layer mocks to your api tests

Version consumers and providers from source code

All tests are actually contract tests

What would you like to ask the maintainers for?

What patterns do you use that others might find helpful?