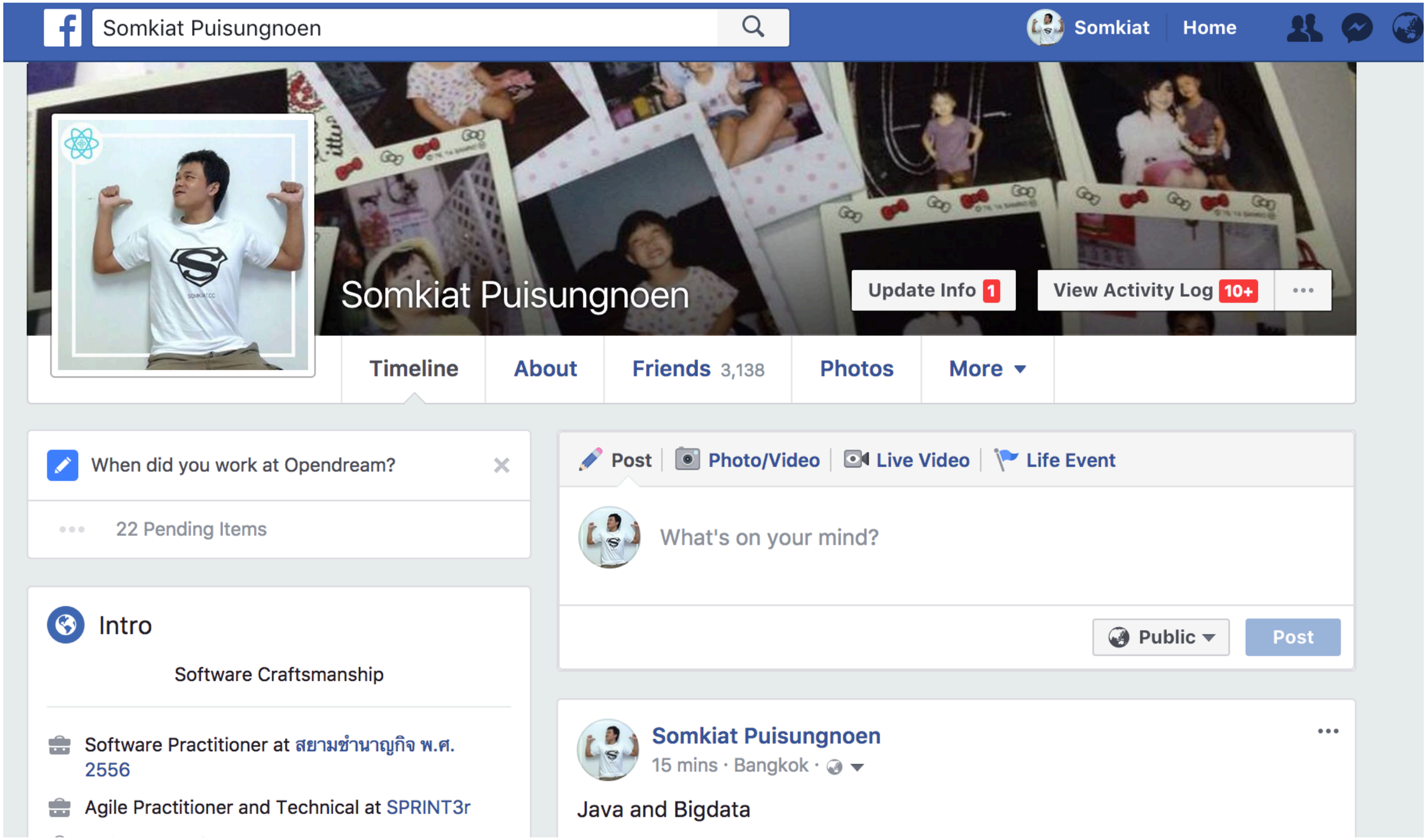
สยามชำนาญกิจ
SIAM CHAMNAN KIT

TDD with NodeJS



somkiat.cc

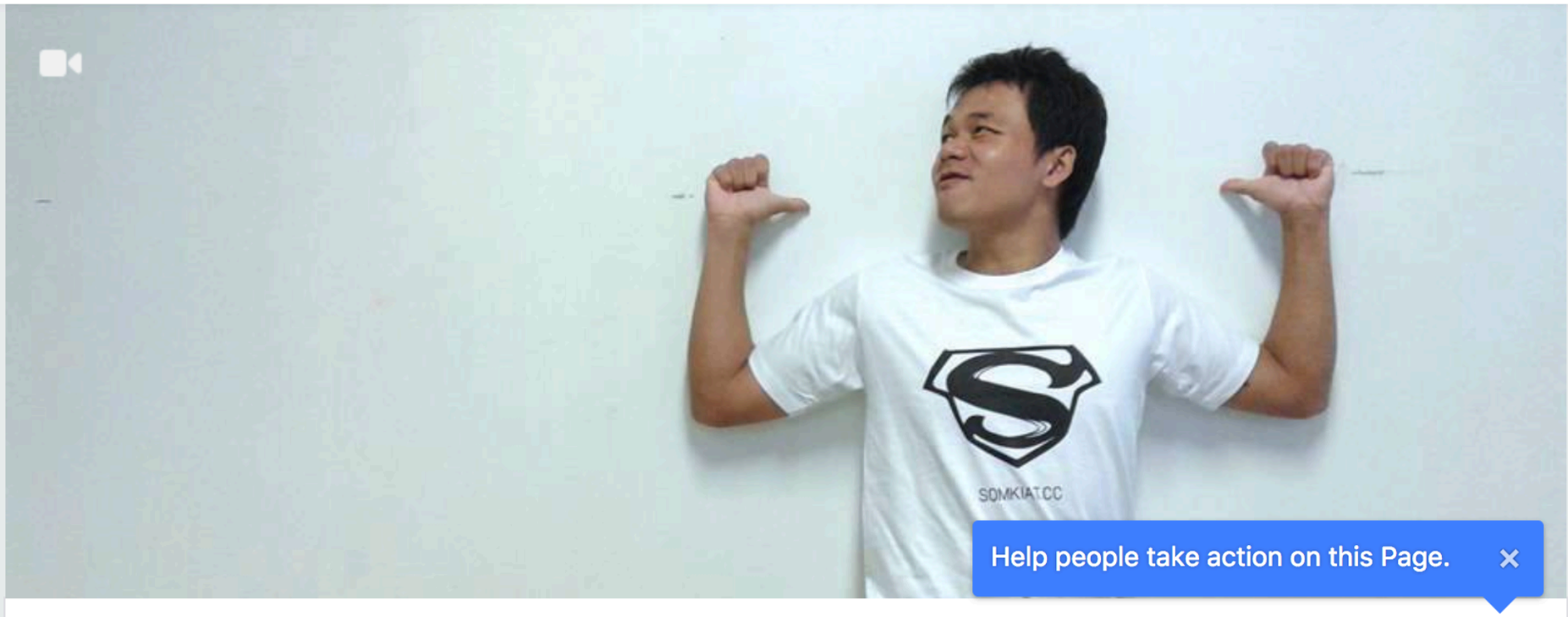
Somkiat | Home

Page | Messages | Notifications 3 | Insights | Publishing Tools | Settings | Help

somkiat.cc

@somkiat.cc

- Home
- Posts
- Videos
- Photos



Help people take action on this Page.

Liked | Following | Share | ...

+ Add a Button

Test-Driven Development

ทำดีดี

ถ้าเขาจะ test อยู่เฉย ๆ เขาก็ test

Agenda

Test-Driven Development

Types of Testing

Good Unit Test (GUT)

Structure of Good Unit Test

Testing with Node.JS

Test/Code coverage

Test-Double

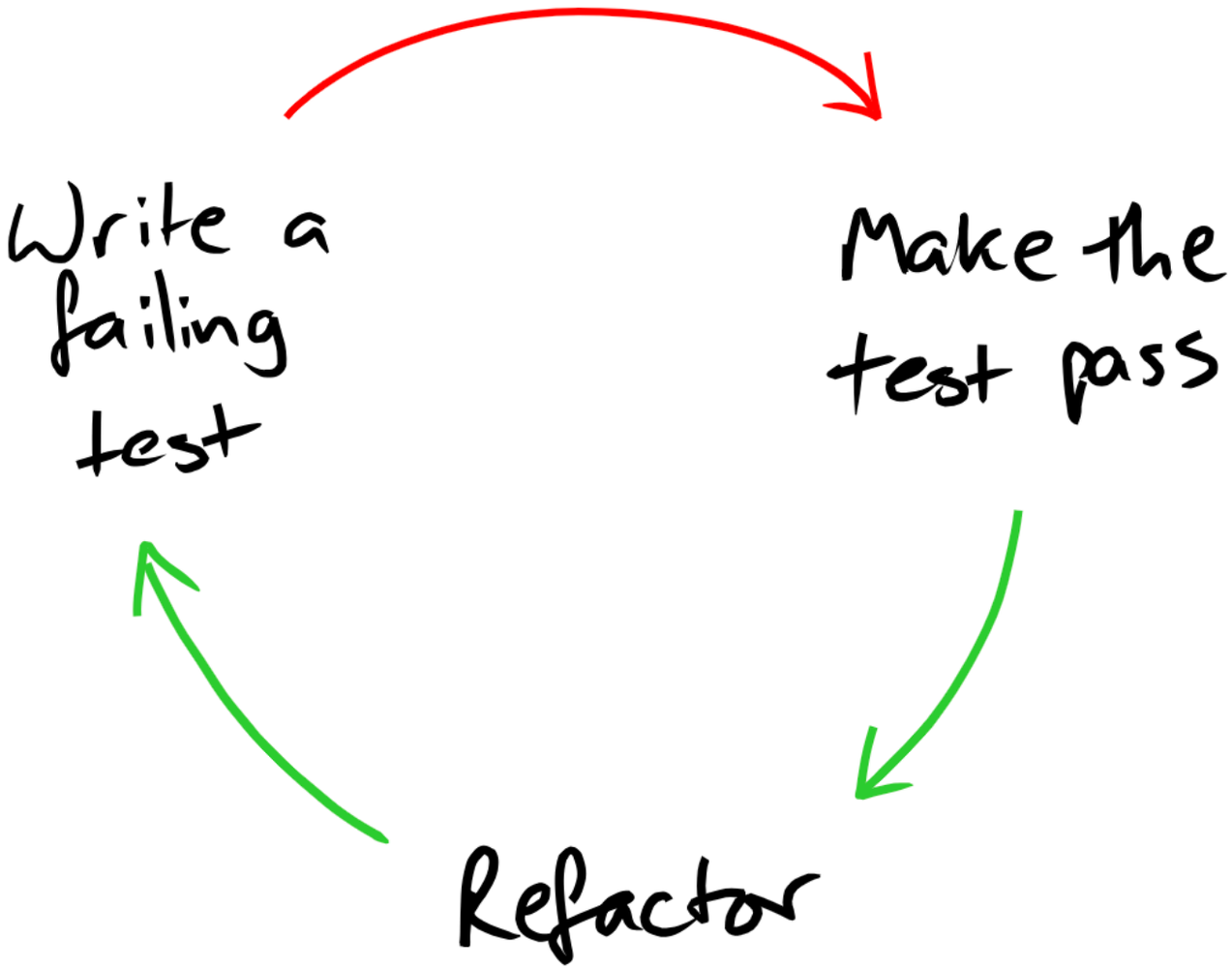
Testable application

https://github.com/up1/course-tdd-with-nodejs

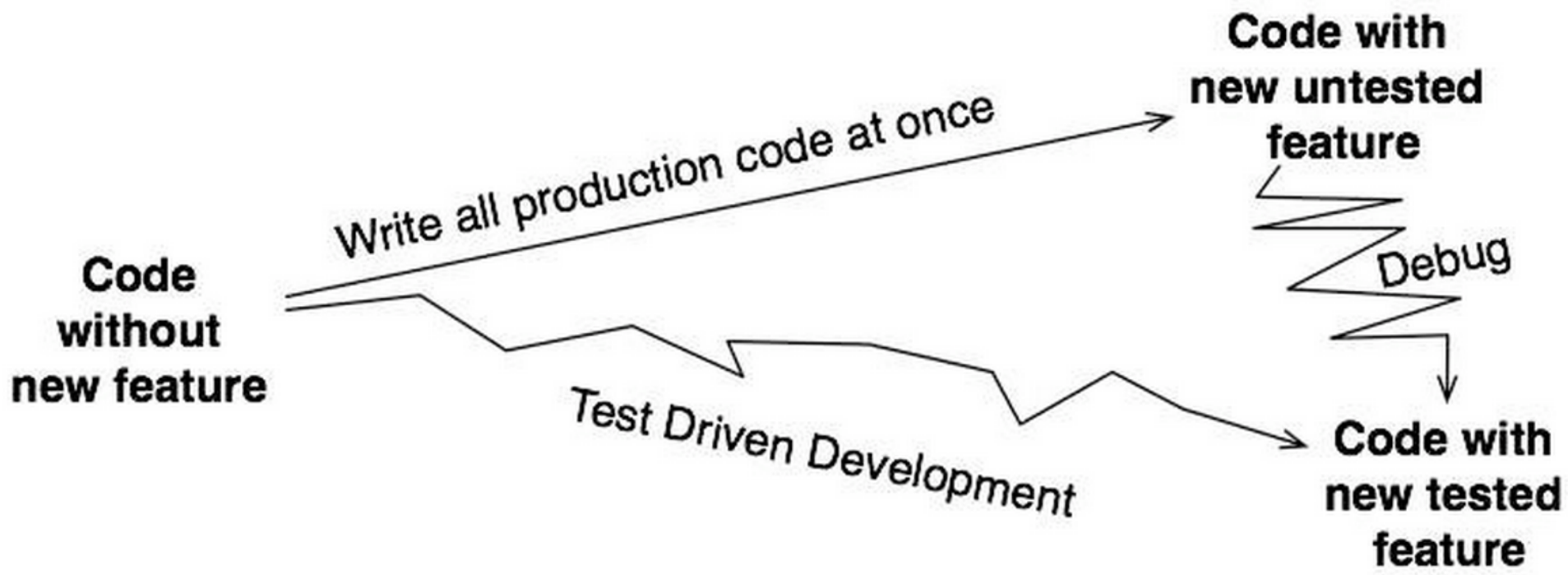
Learn by **DOING**.

Test-Drive Development

TDD Life Cycle

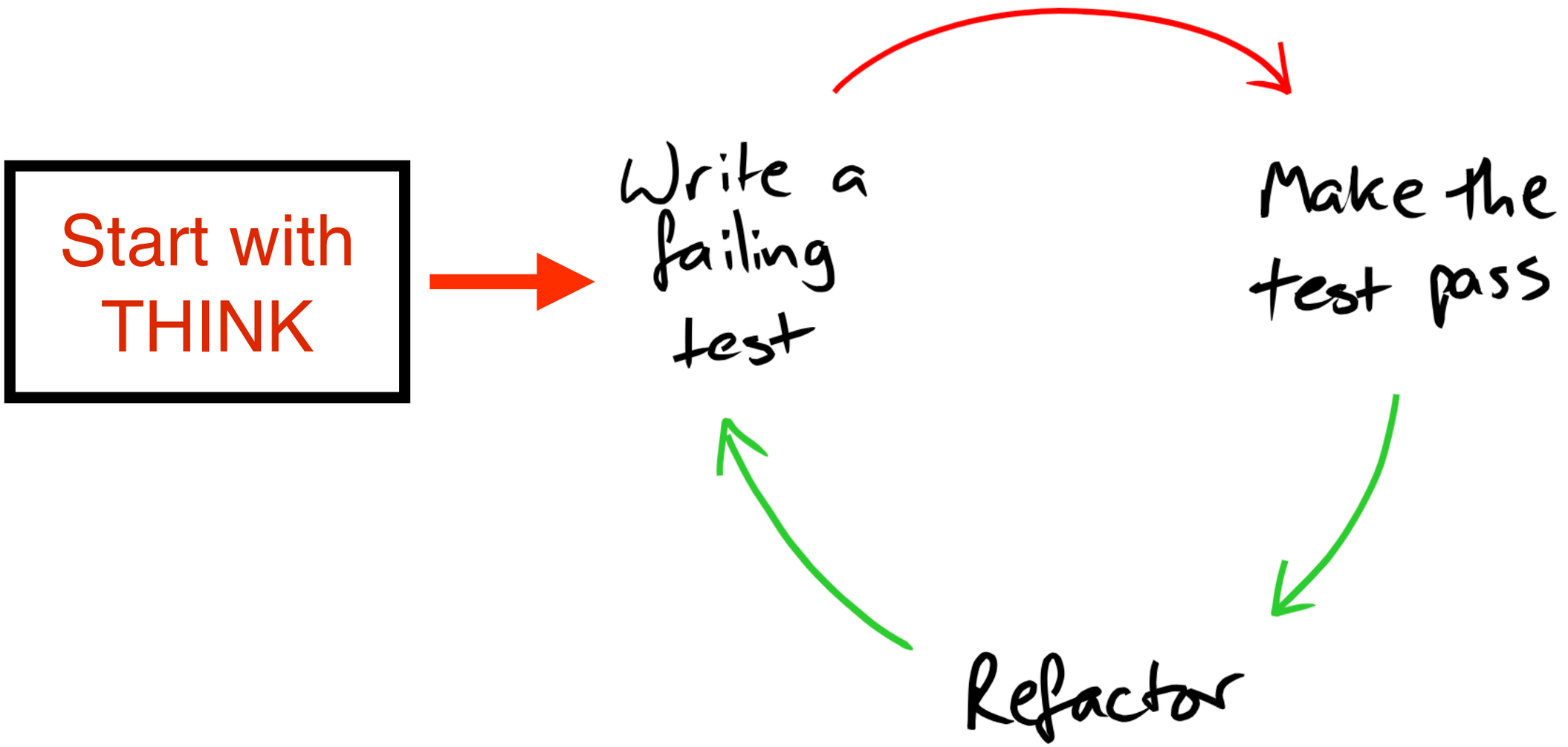


TDD vs DLP



Red Green Refactor

Let's start



Rules of TDD

Rule #1

Don’t write production code unless it’s to help a failing unit test pass

Rule #2

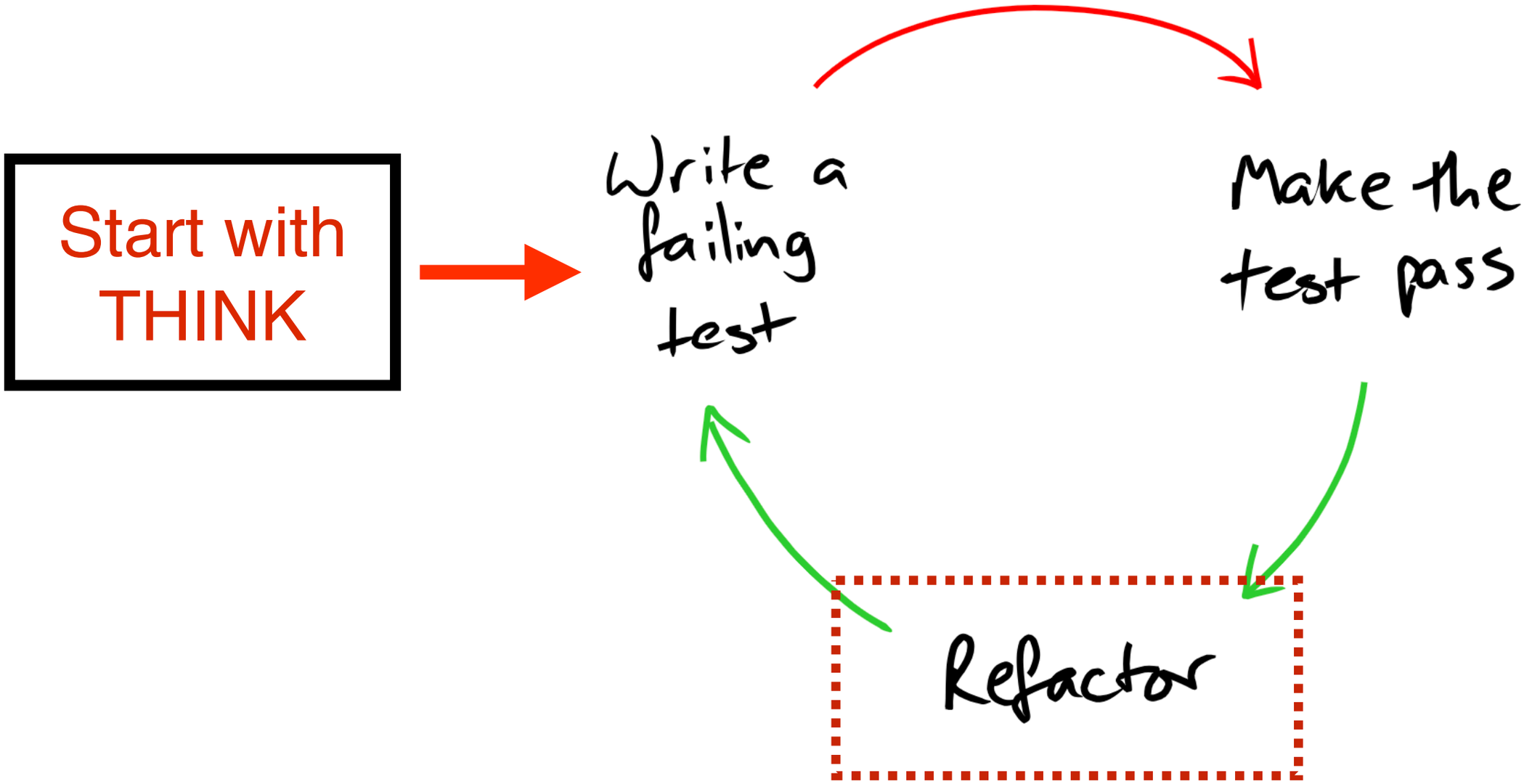
Only write enough of a unit test to fail

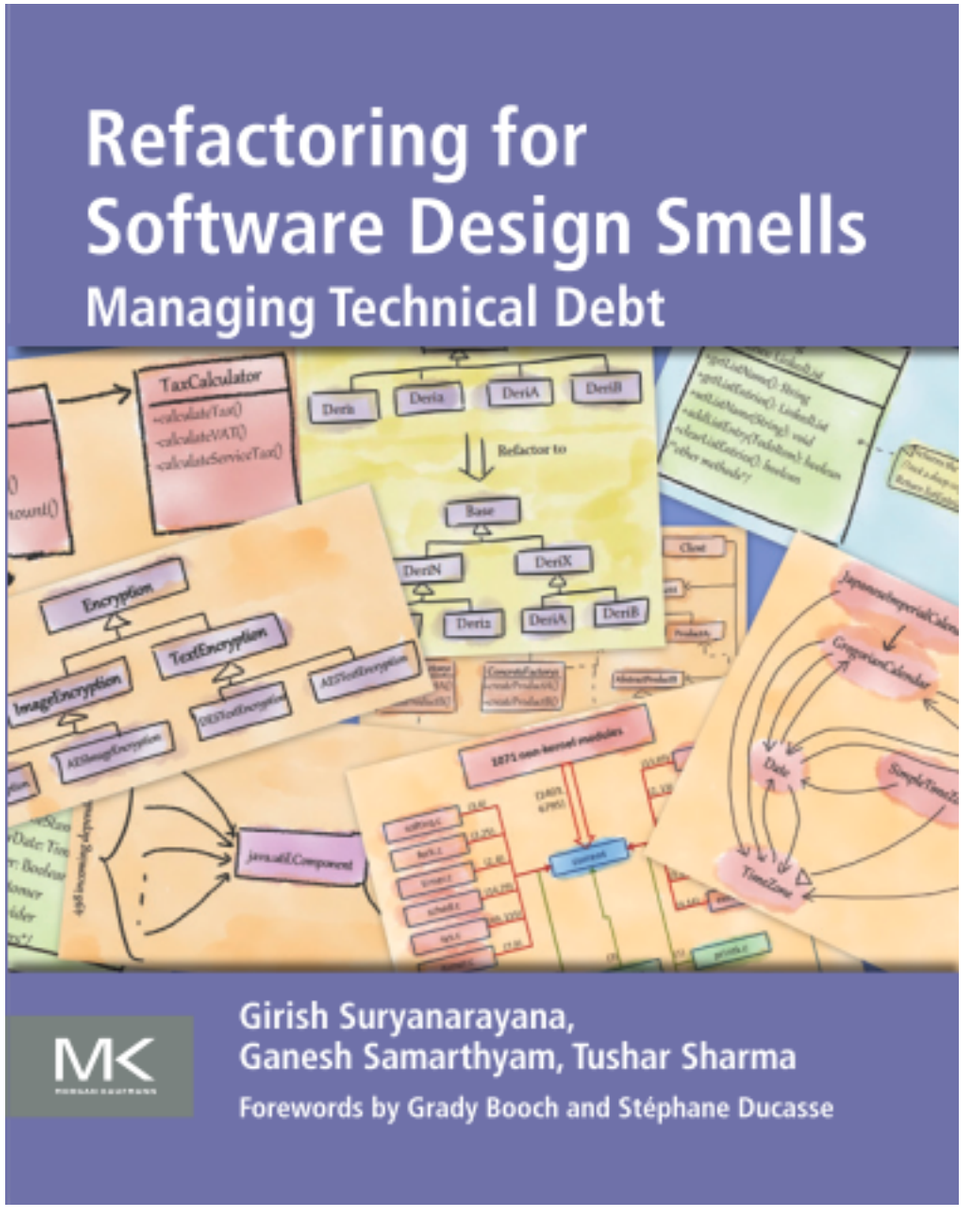
Rule #3

Only write enough production code to help a failing unit test pass

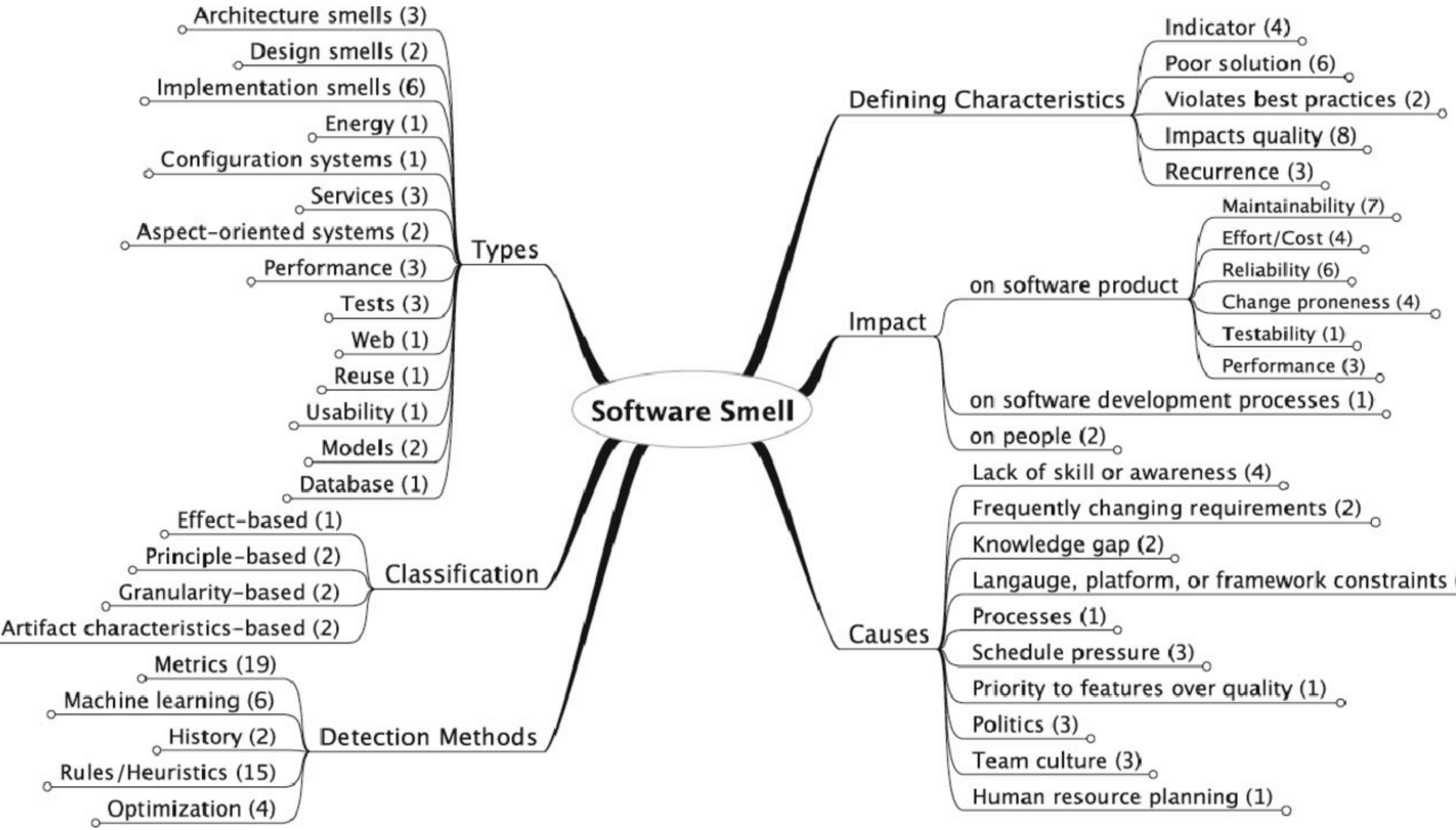
Don’t forgot Refactoring

Let's start





http://www.tusharma.in/smells/



https://www.sciencedirect.com/science/article/pii/S0164121217303114

Code Smell

Code Smells

– What? How can code "smell"??

– Well it doesn't have a nose... but it definitely can stink!

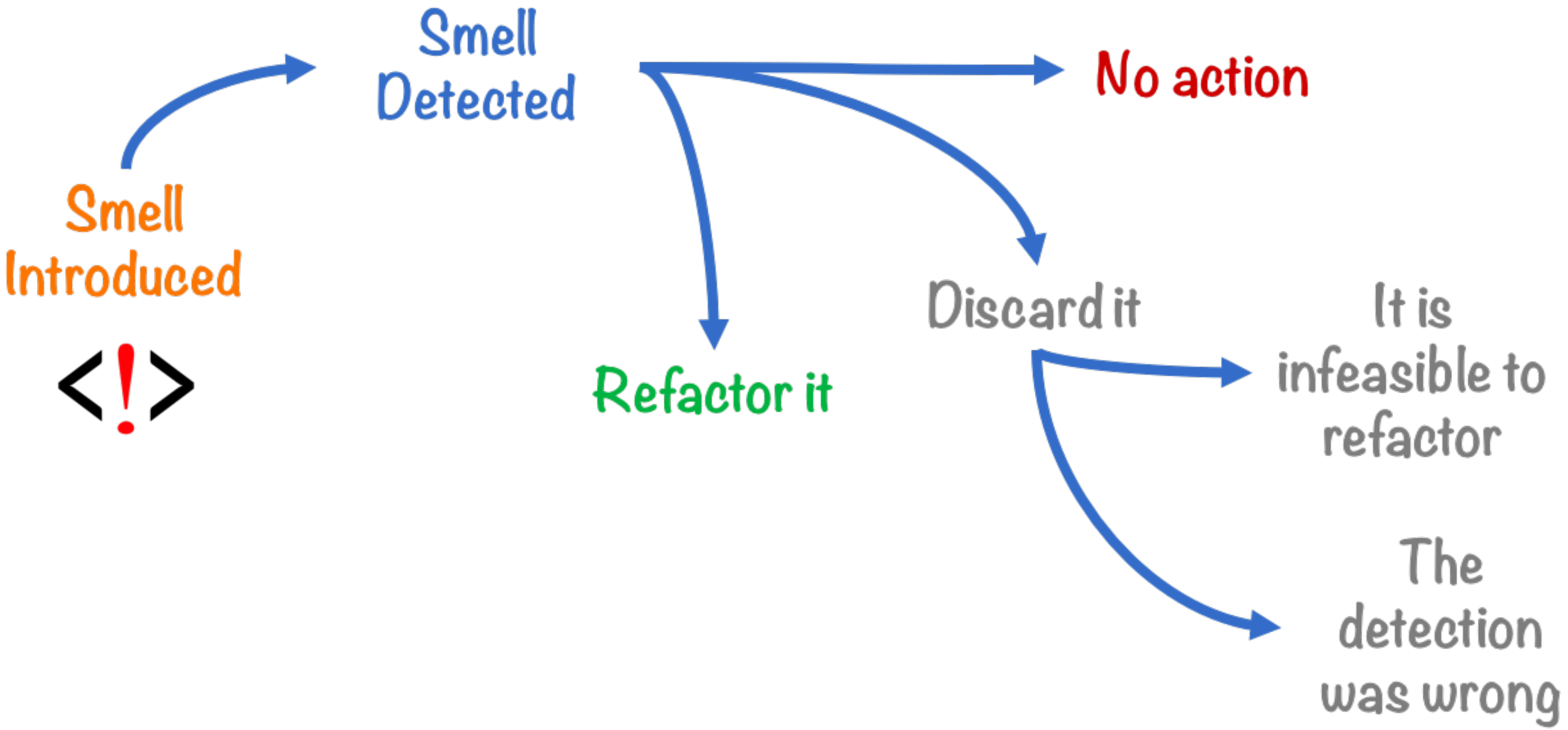
Bloaters

Bloaters are code, methods and classes that have increased to such gargantuan proportions that they are hard to work with. Usually these smells do not crop up right away, rather they accumulate over time as the program evolves (and especially when nobody makes an effort to eradicate them).

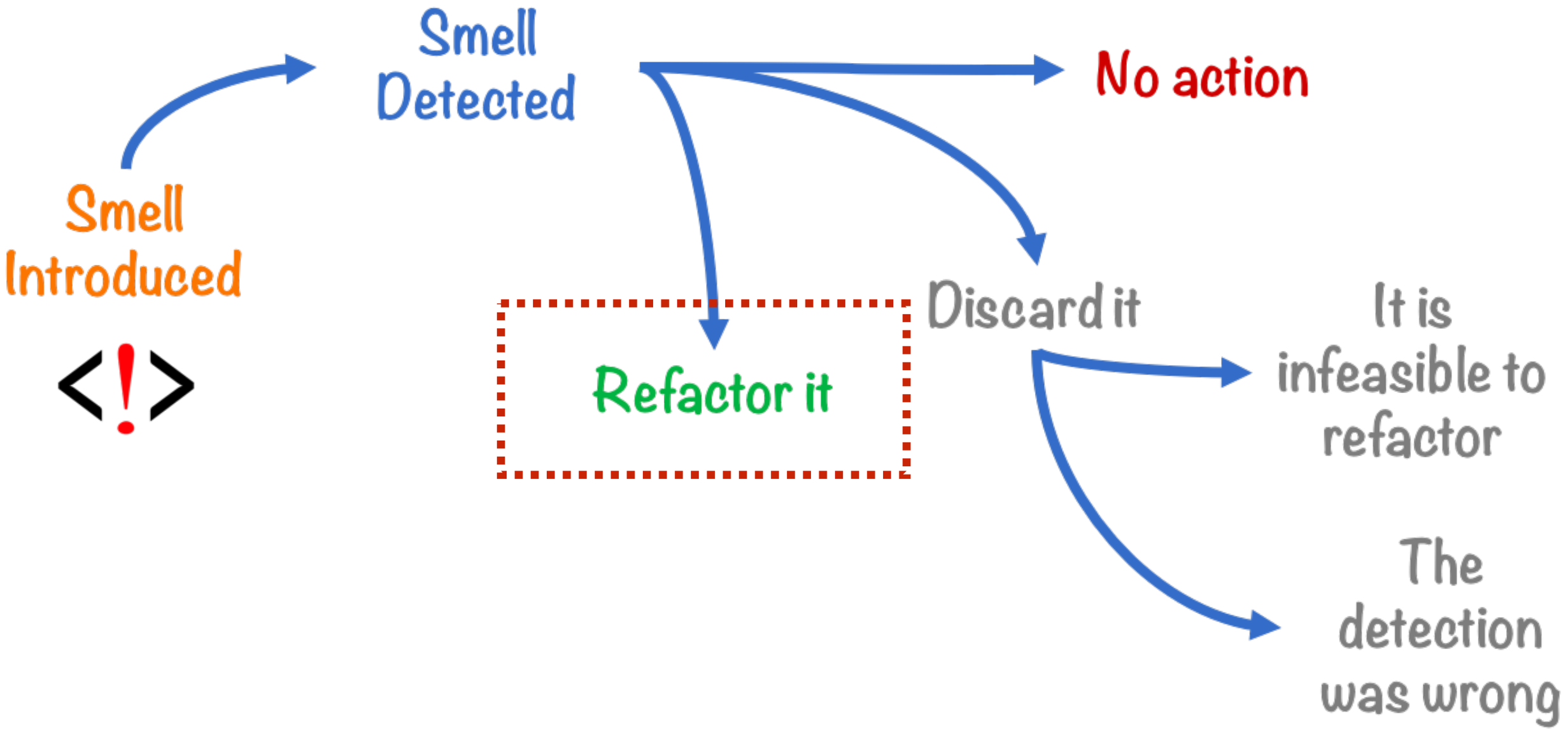
- Long Method
- Large Class
- Primitive Obsession
- Long Parameter List
- Data Clumps

https://sourcemaking.com/refactoring/smells

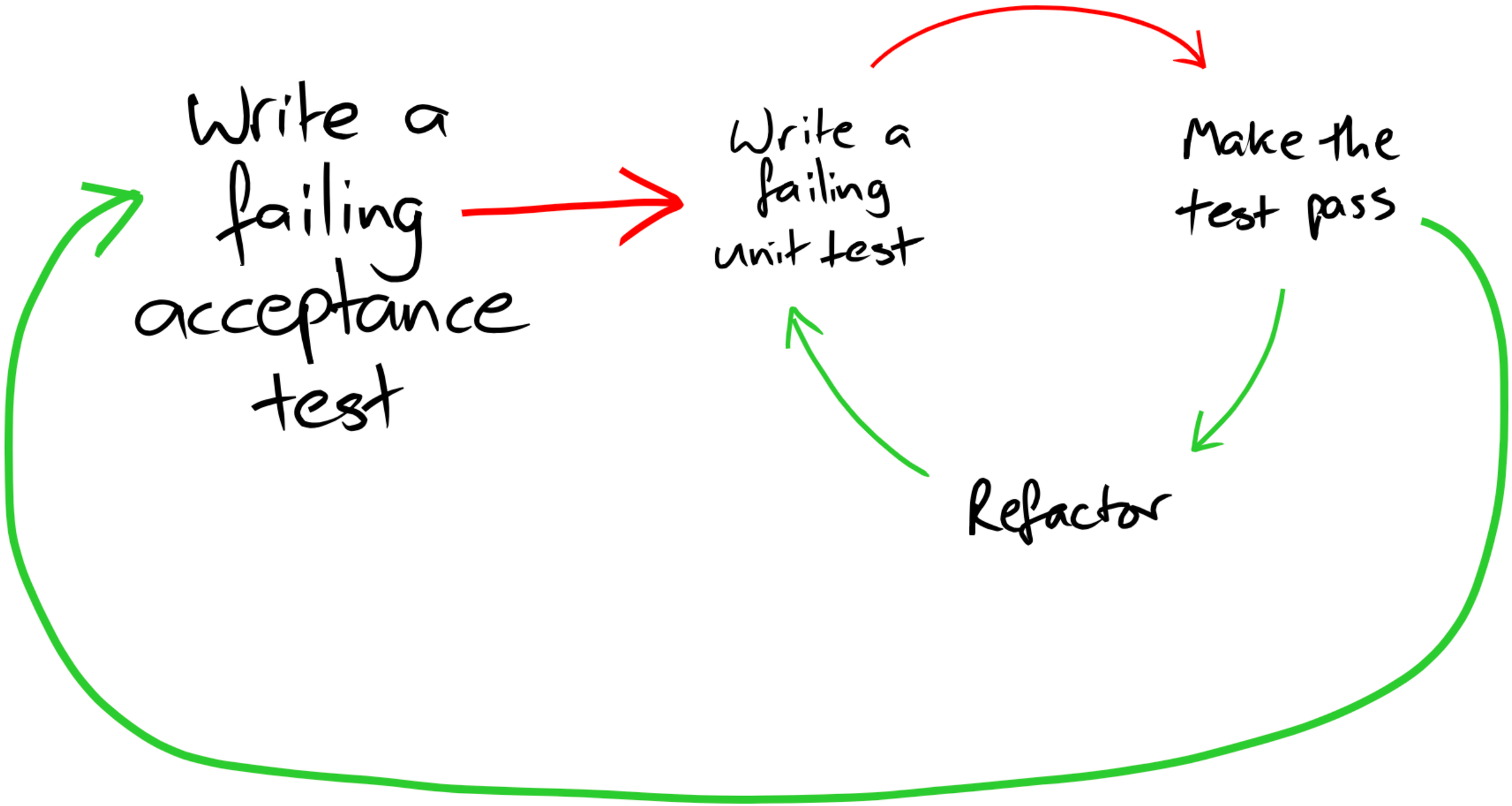
Life-cycle of a smell



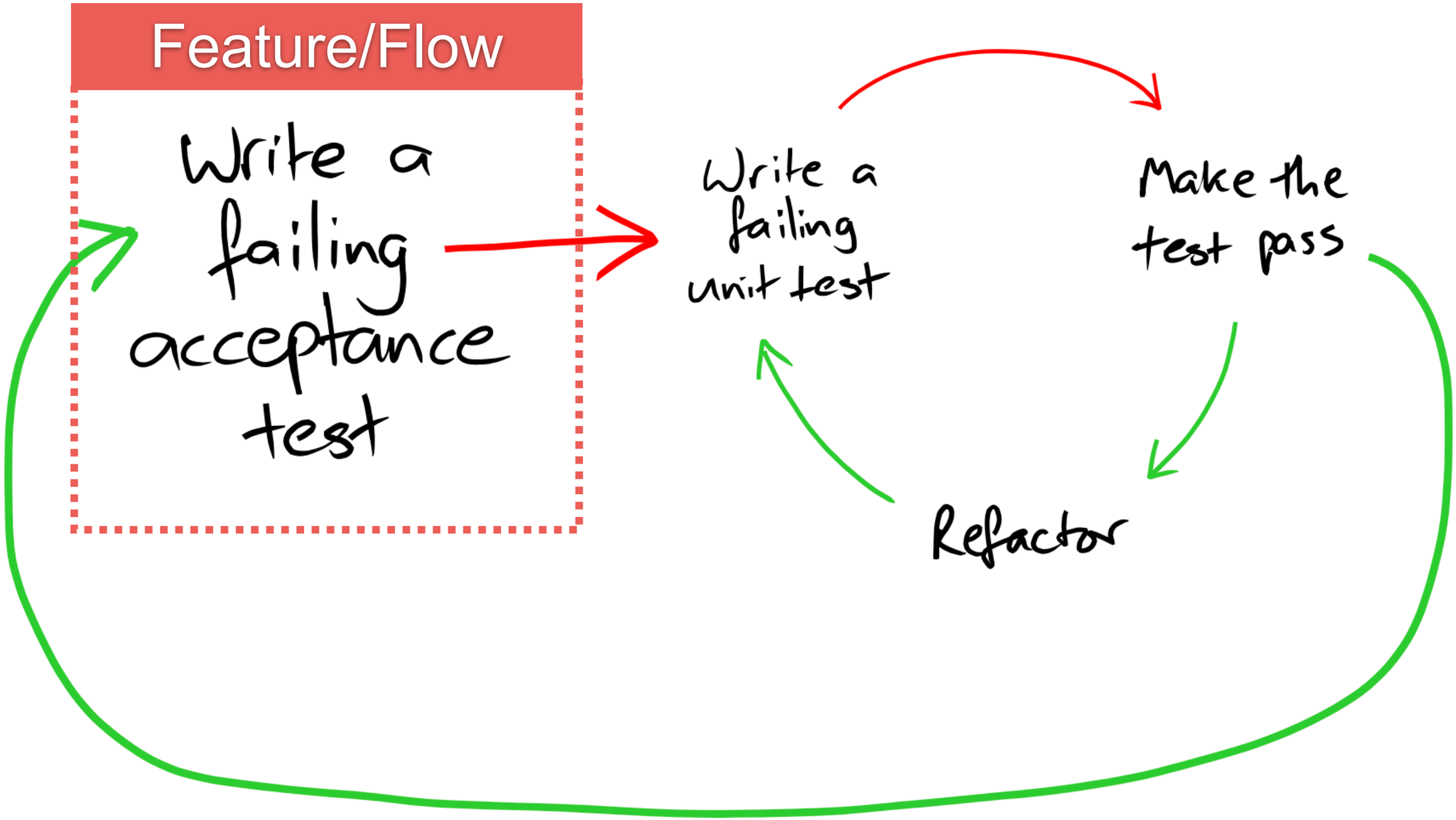
Life-cycle of a smell



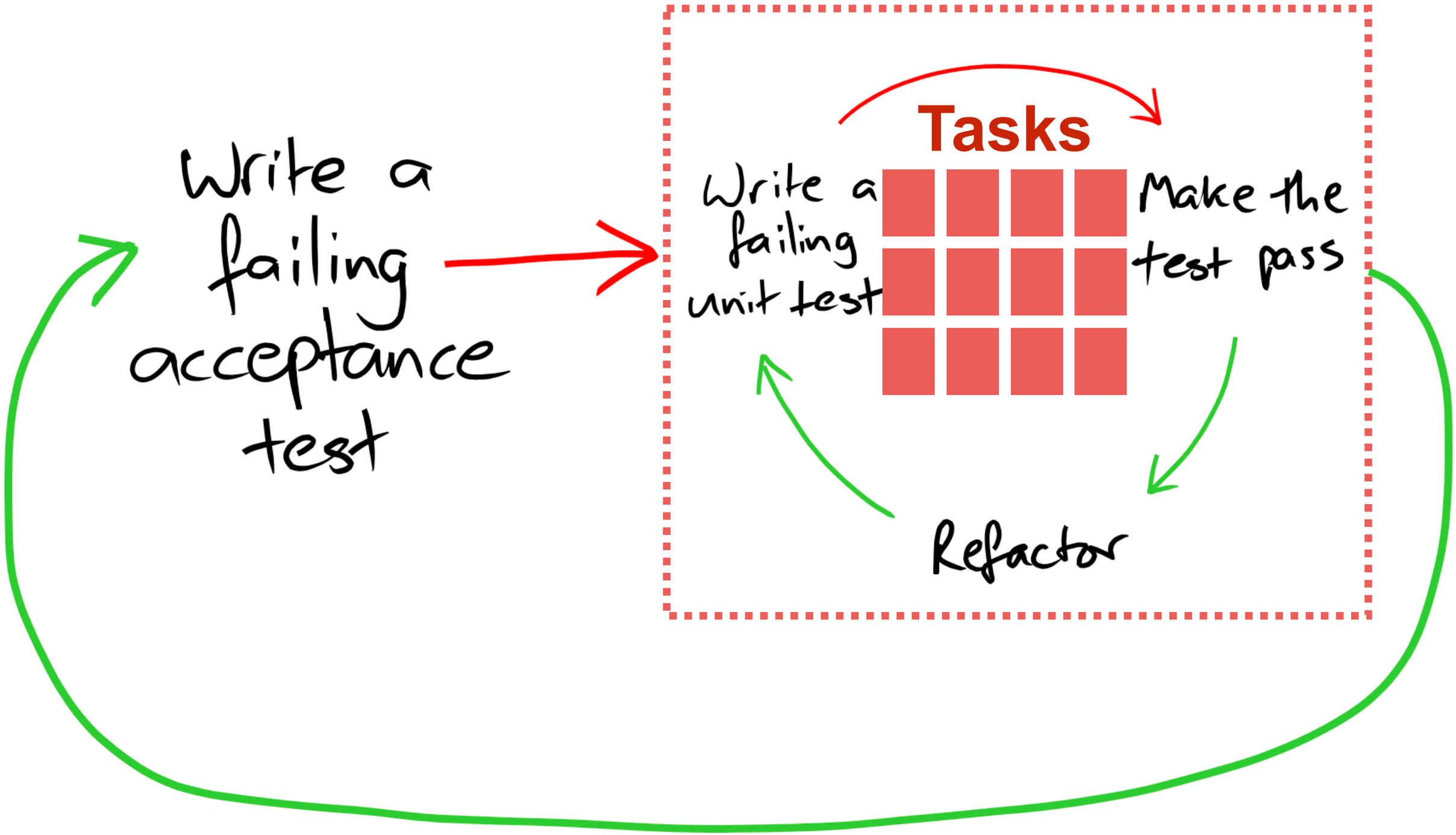
Outside-in develop/test



Outside-in develop/test

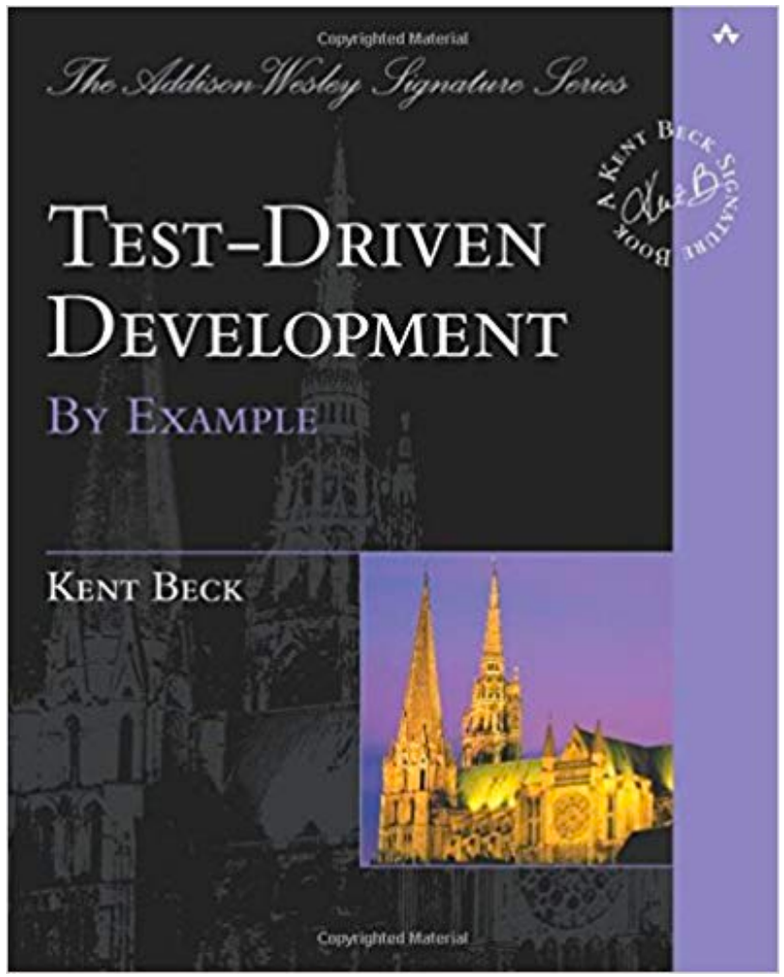


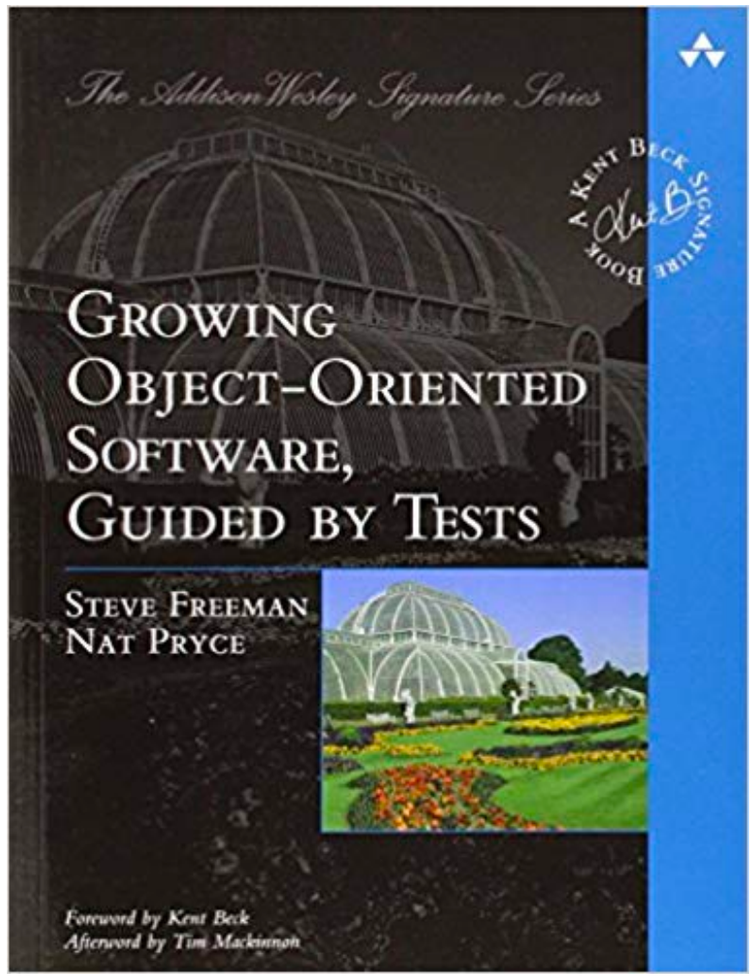
Outside-in develop/test

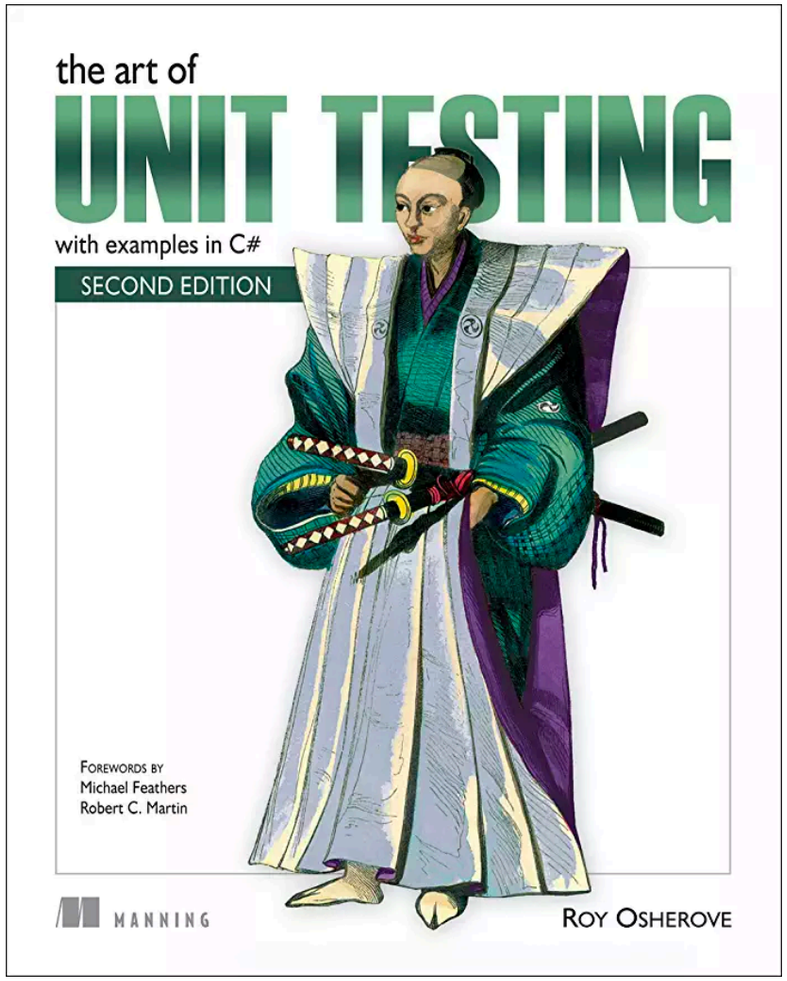


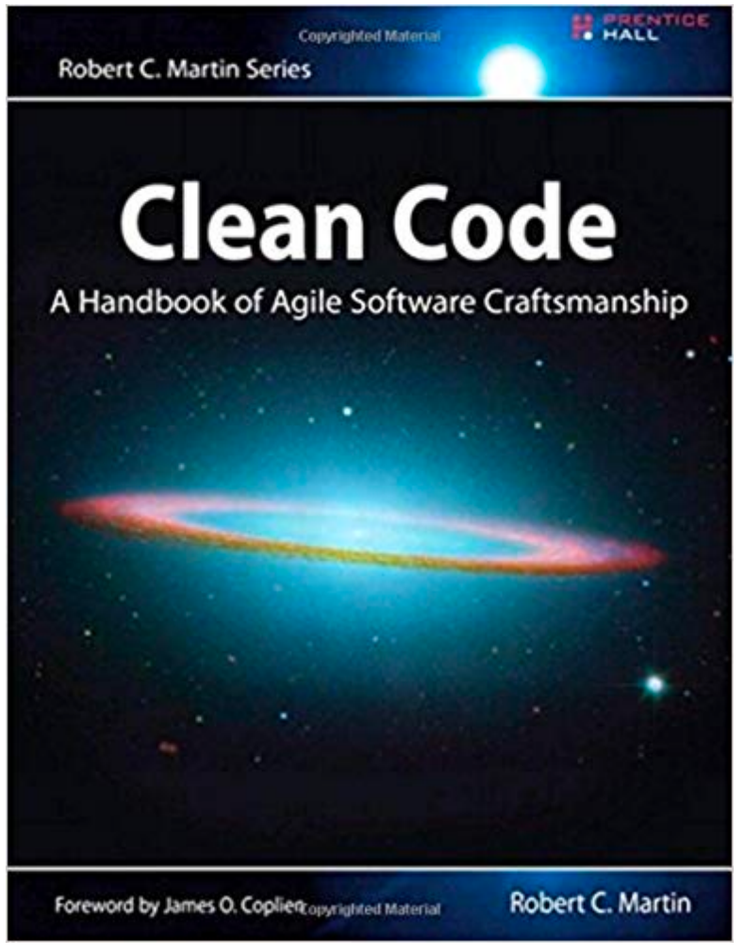
คิดอย่างไร ก็จะทำอย่างนั้น

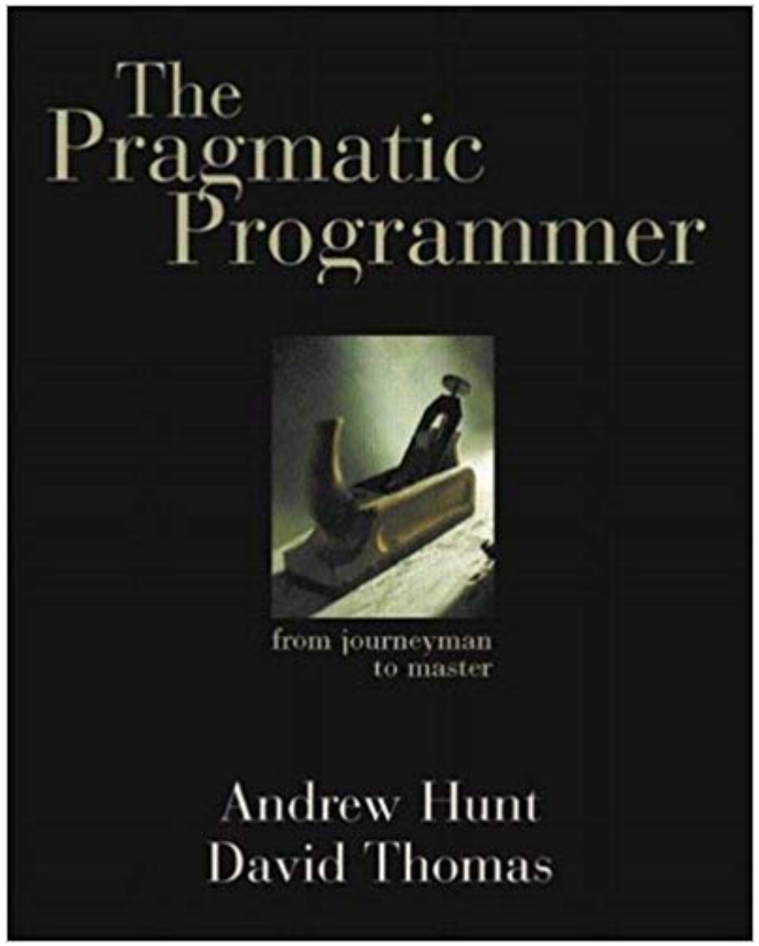
Books

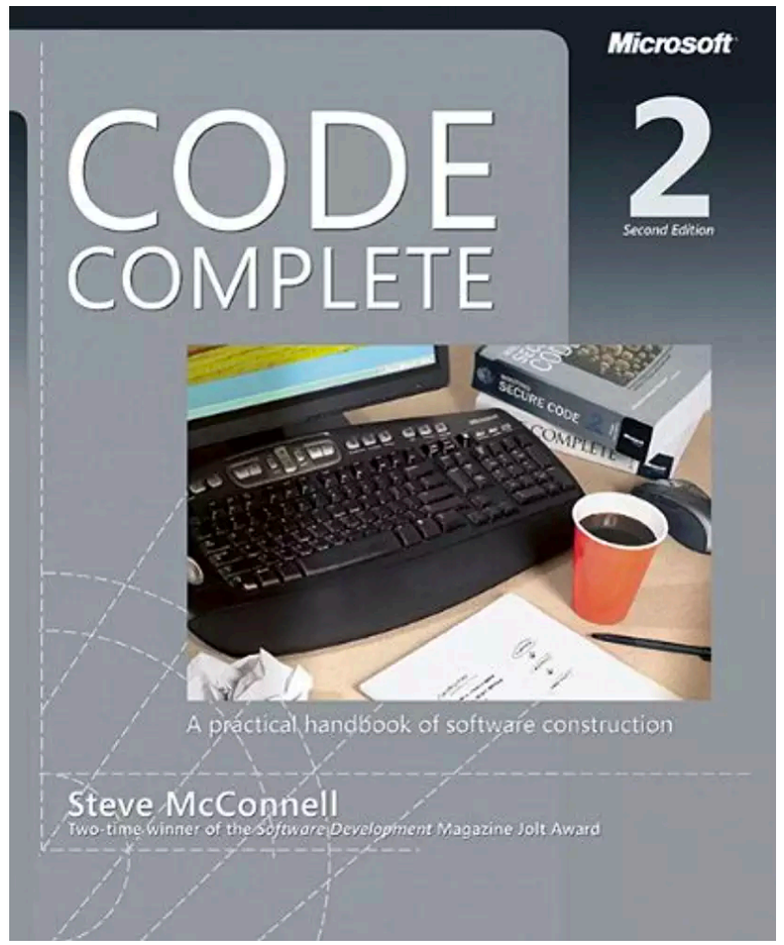


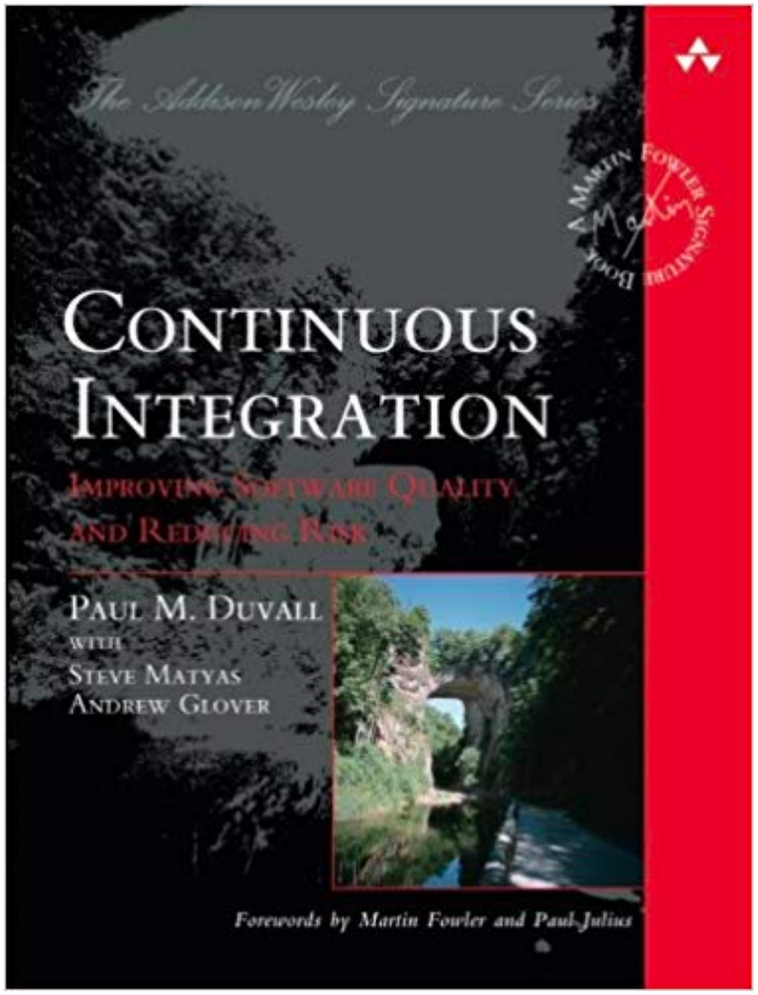


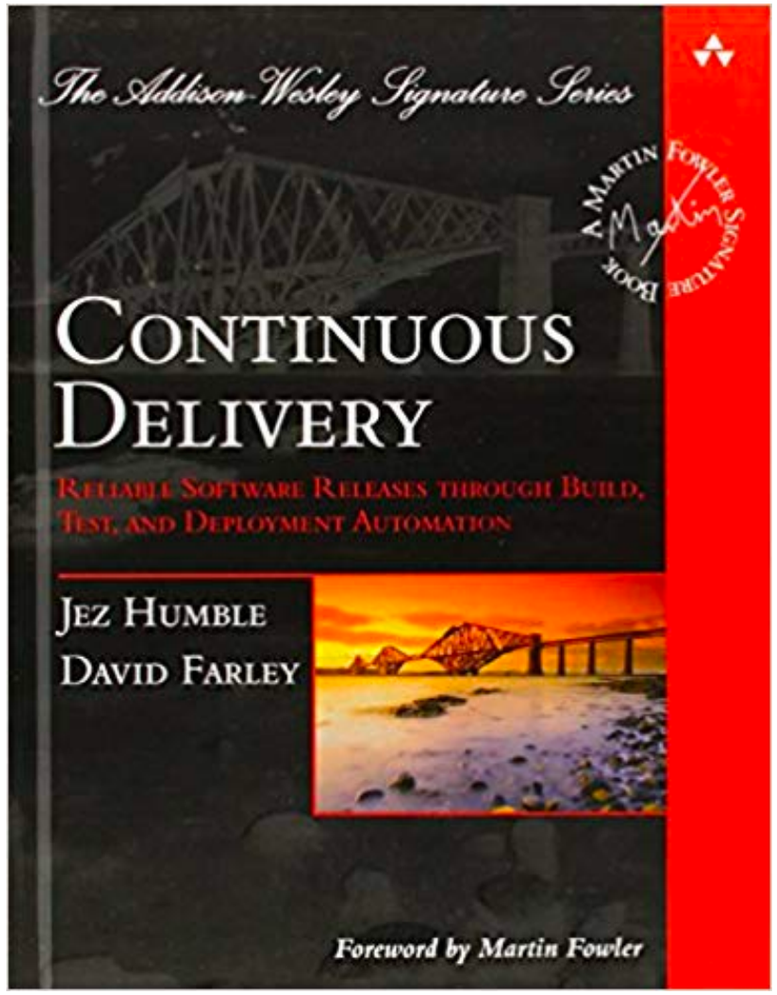






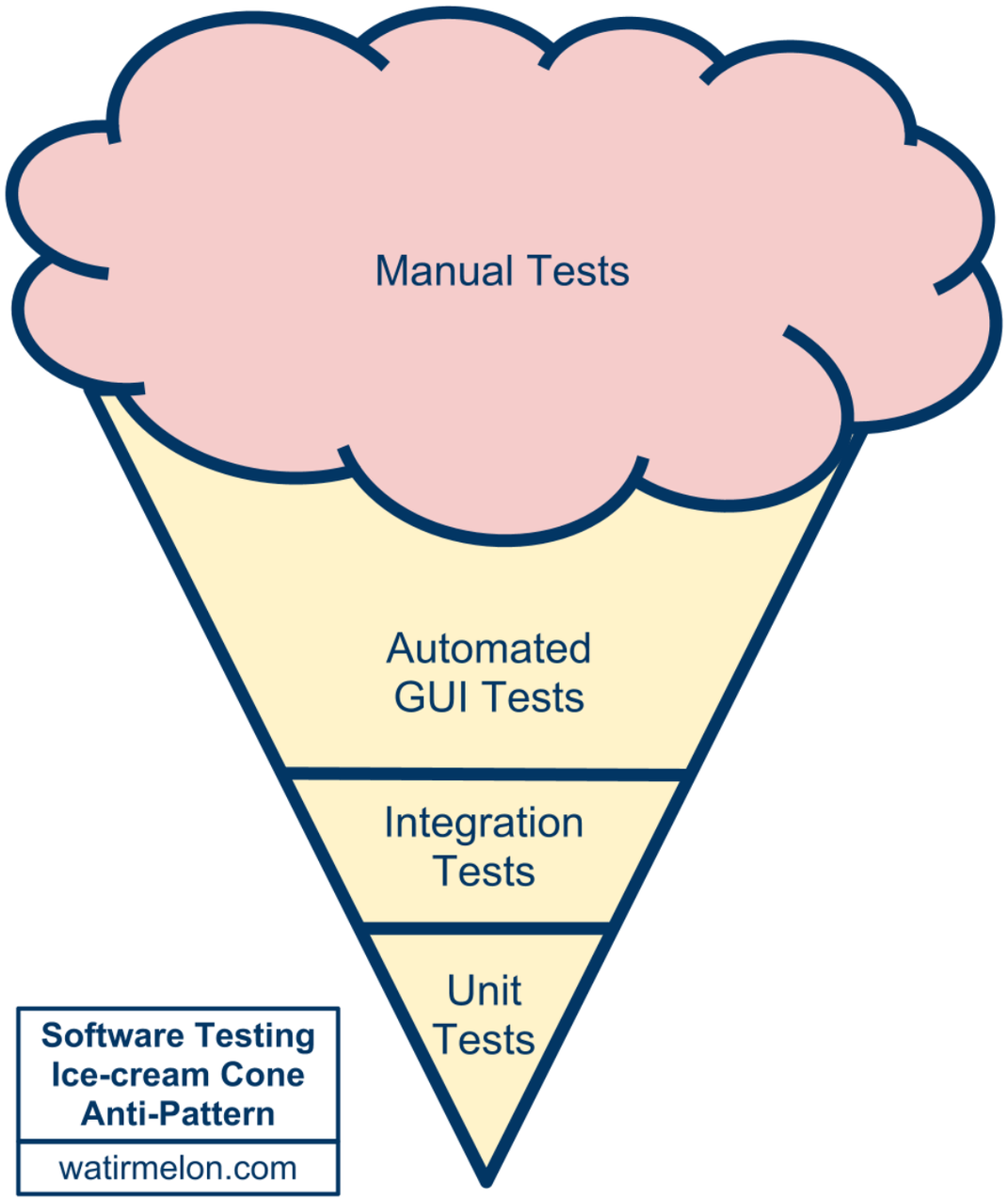




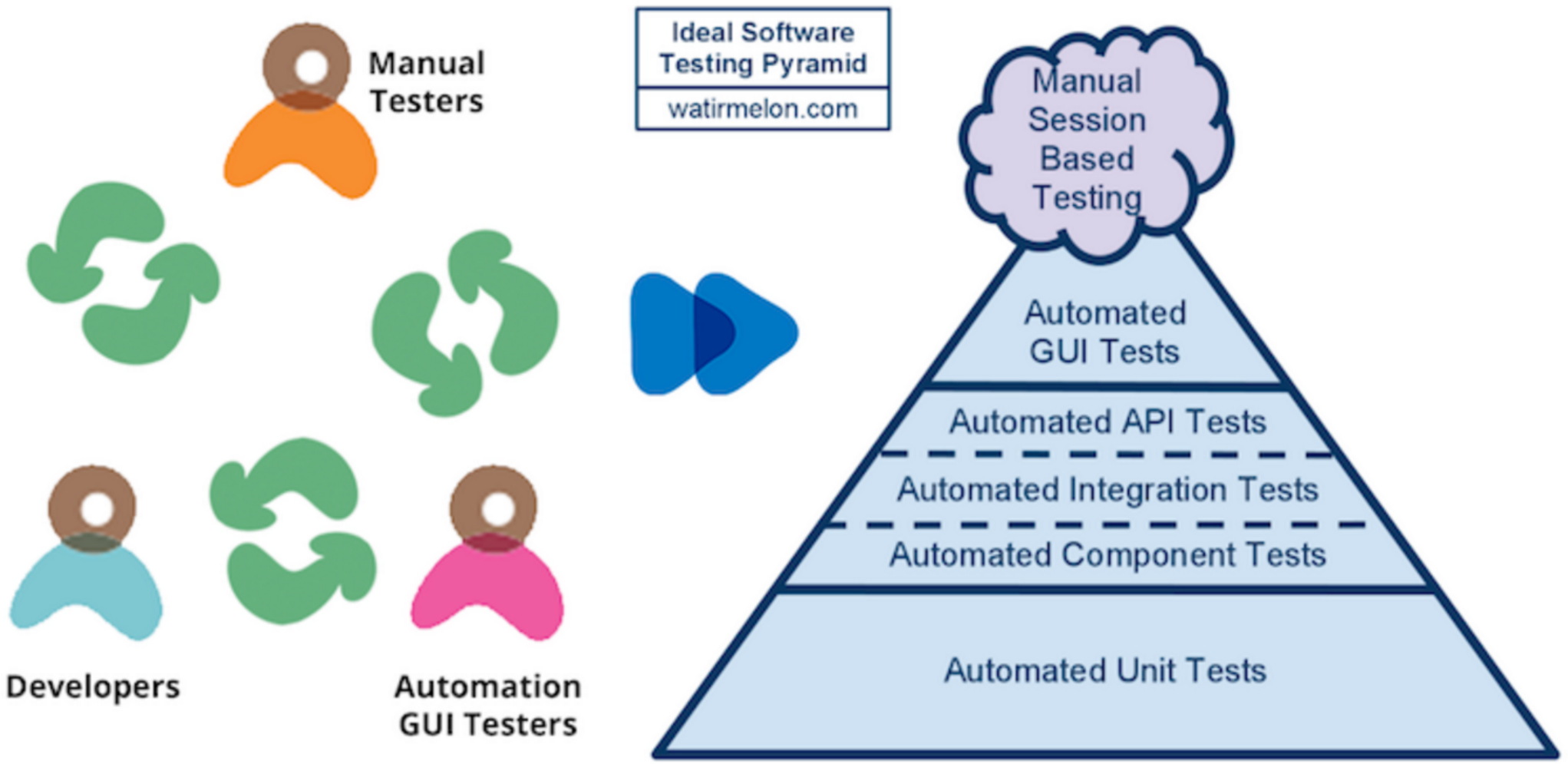


Types of Testing

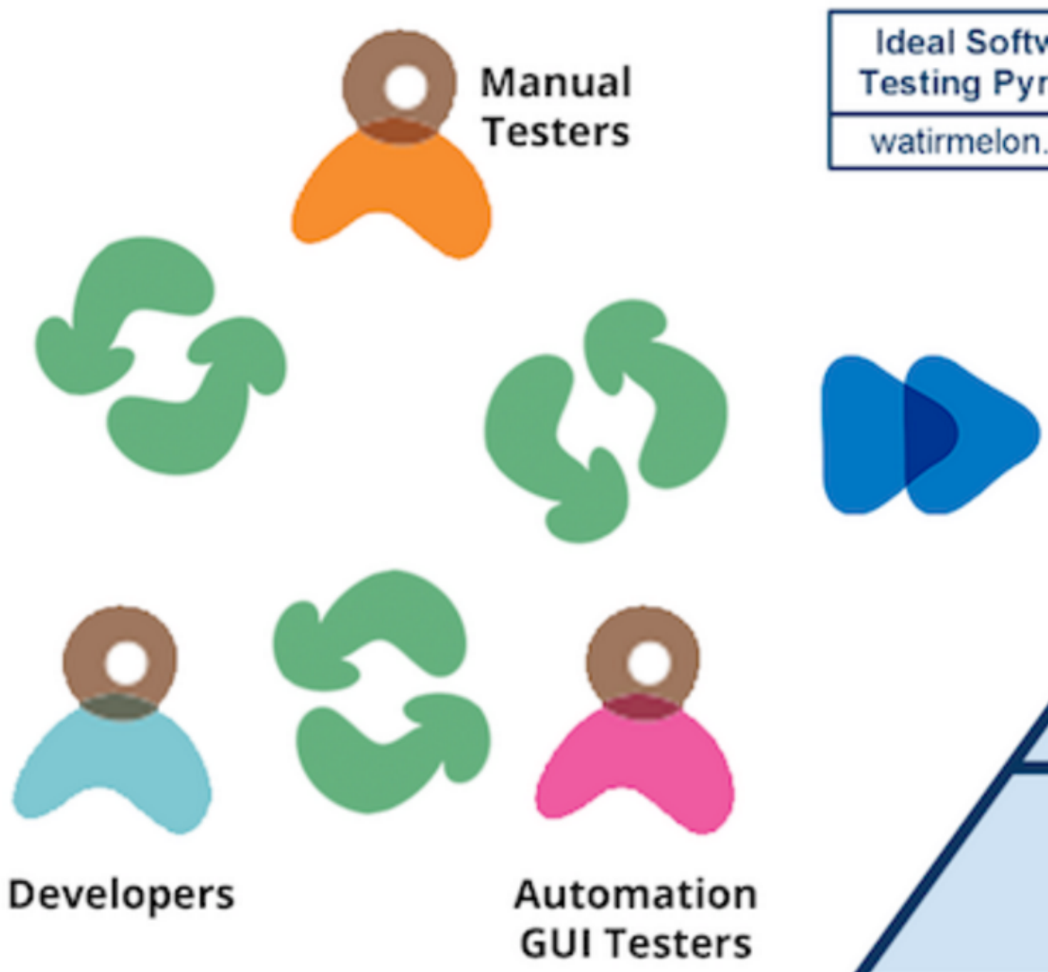
Ice-cream testing

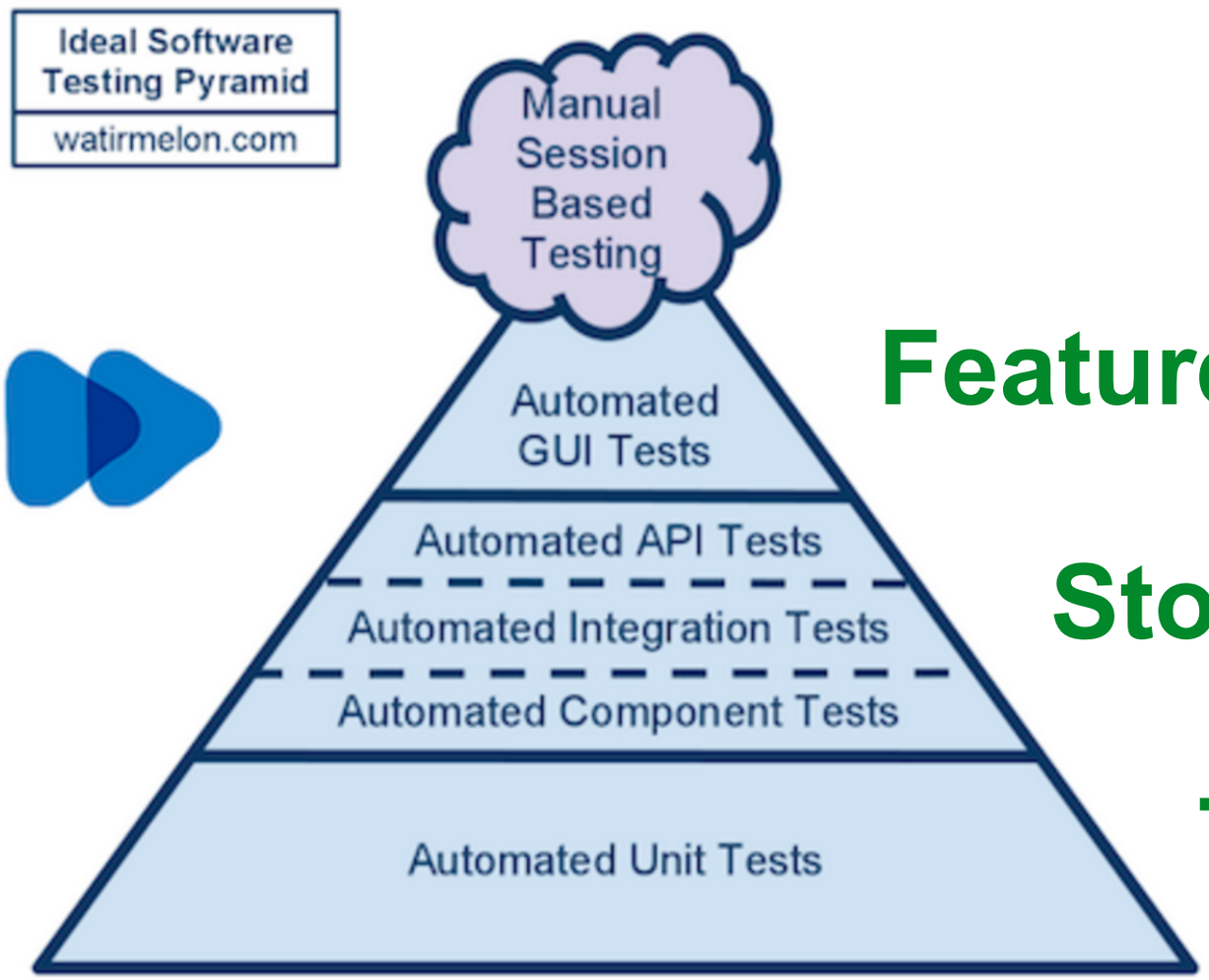


Testing Pyramid

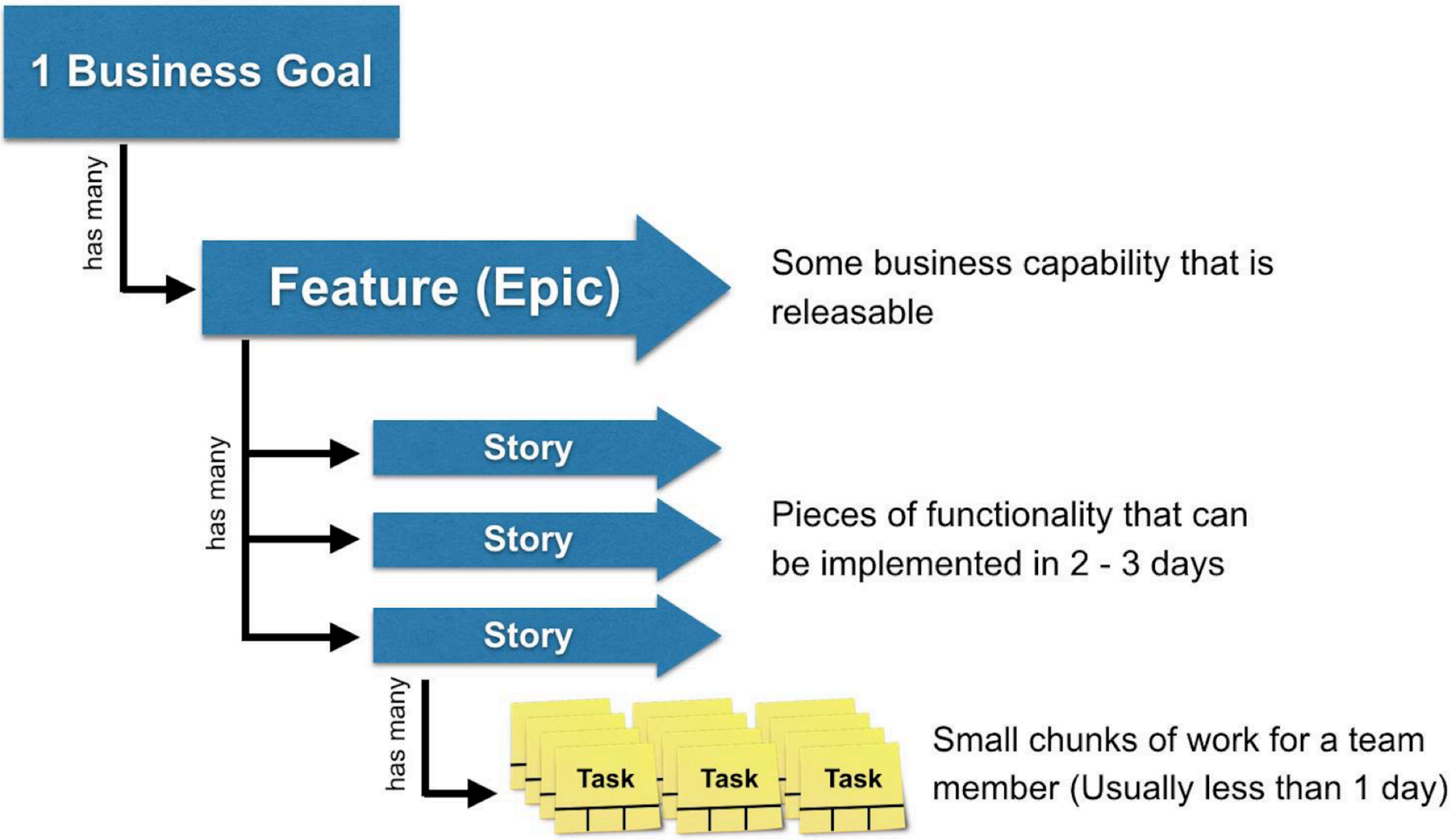


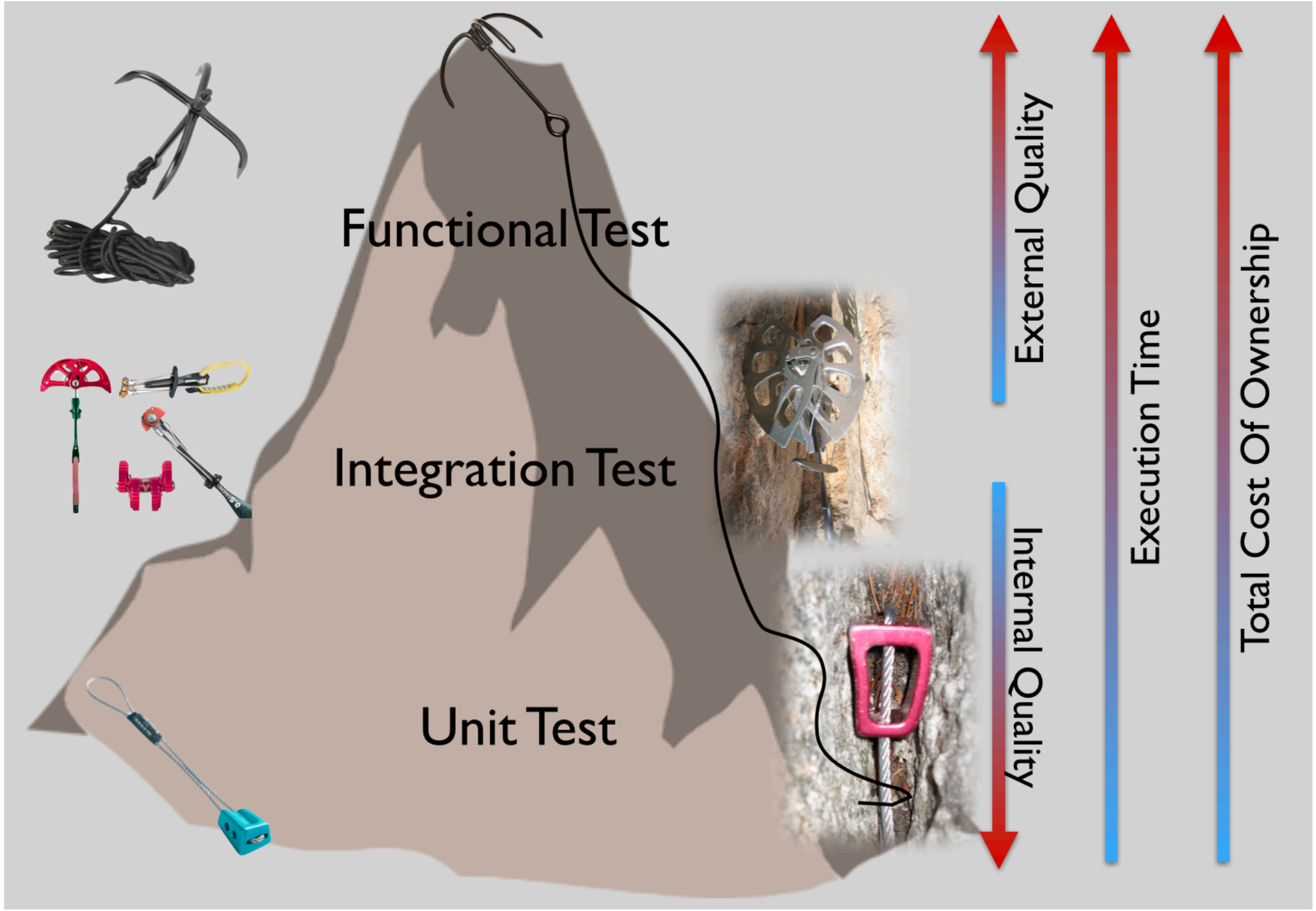
Testing Pyramid





Work break down





https://less.works/less/technical-excellence/unit-testing.html

Mind-set switch

Instead of

We are here to find bug

We are here to ensure requirement are met

We are here to break the software

Mind-set switch

Instead of

We are here to find bug

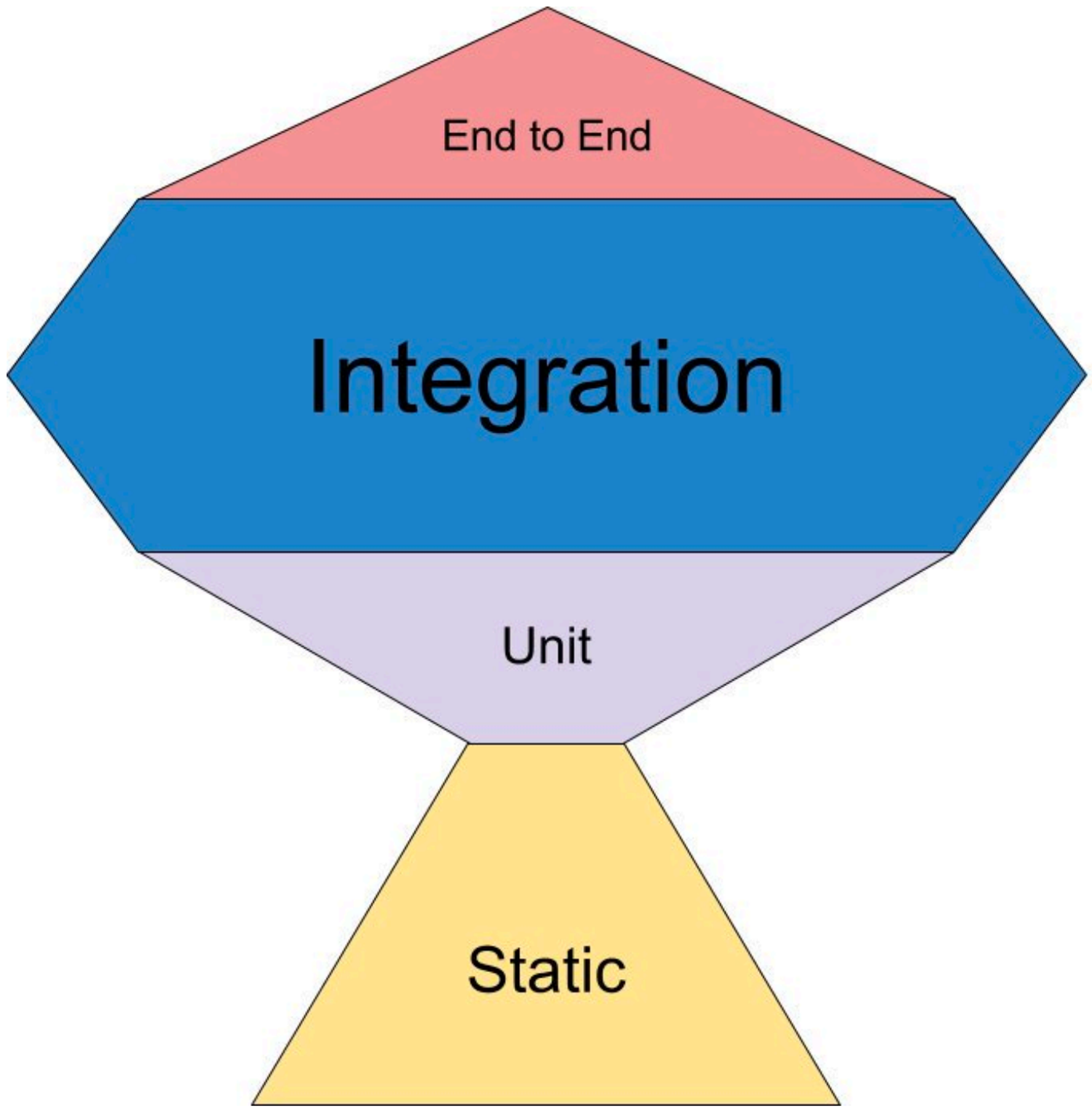
We are here to ensure requirement are met

We are here to break the software

Think

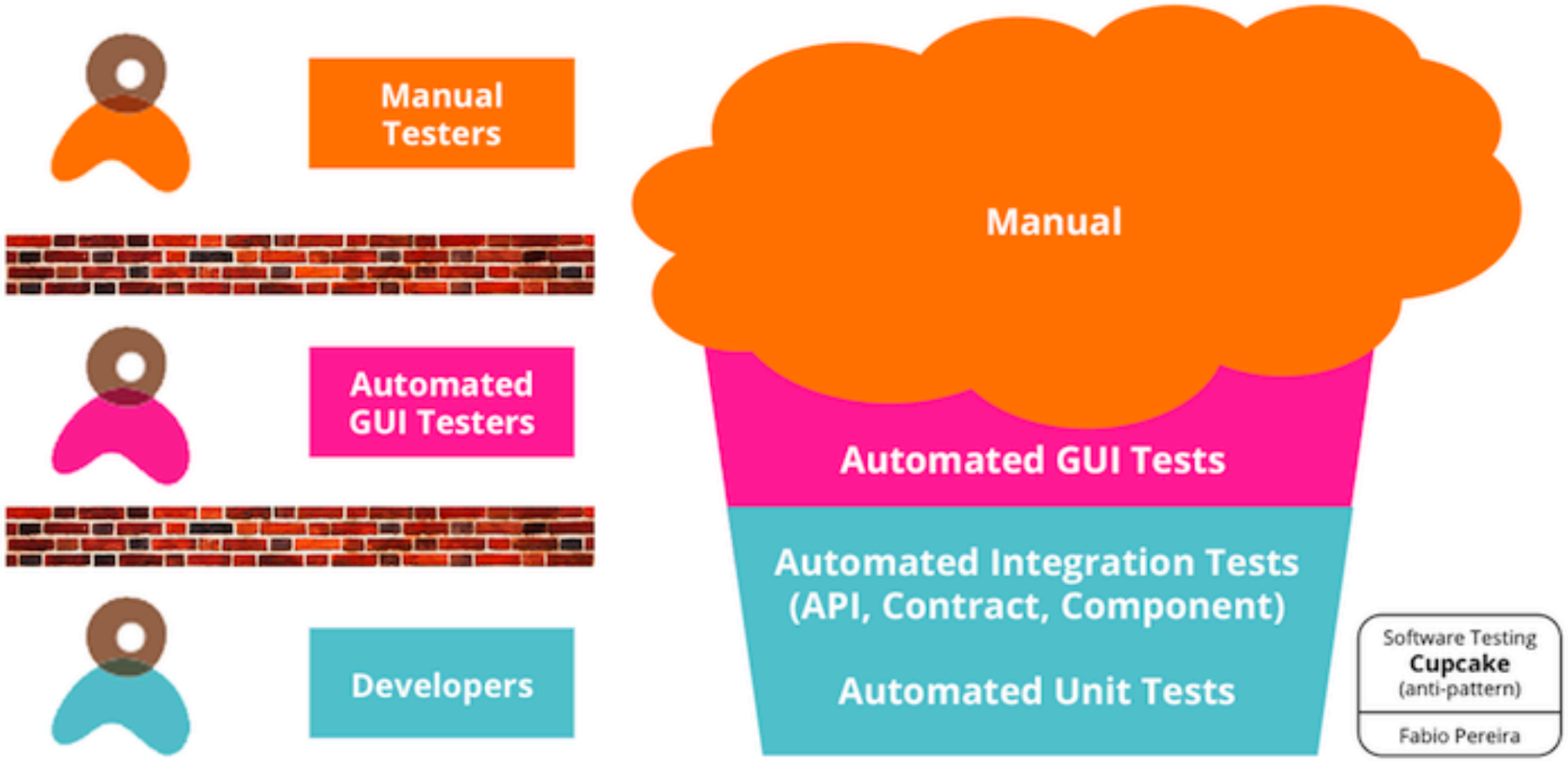
What can I do to help deliver the software successfully !!

Trophy testing



https://kentcdodds.com/blog/write-tests

Cupcake testing !!

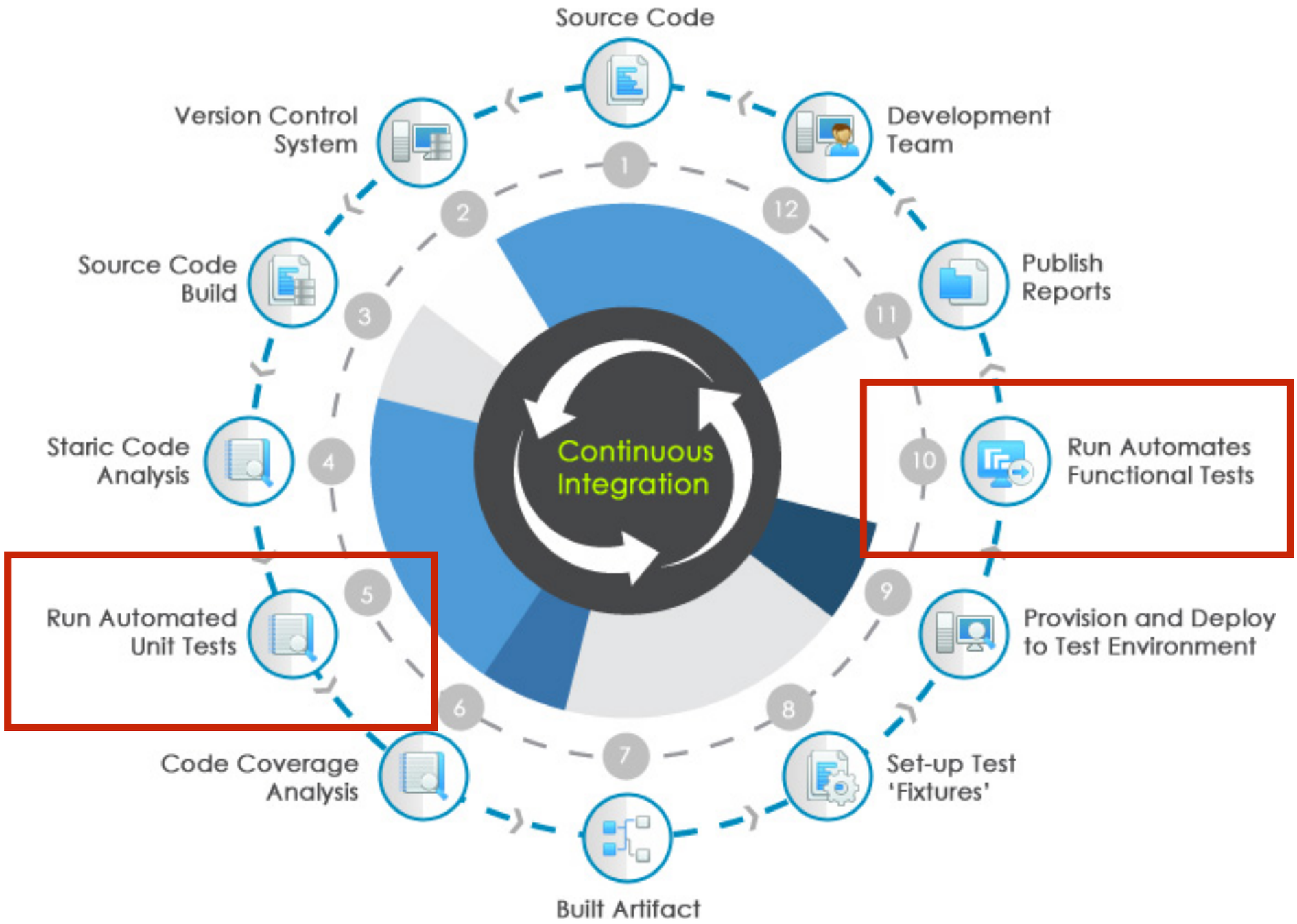


https://www.thoughtworks.com/insights/blog/introducing-software-testing-cupcake-anti-pattern

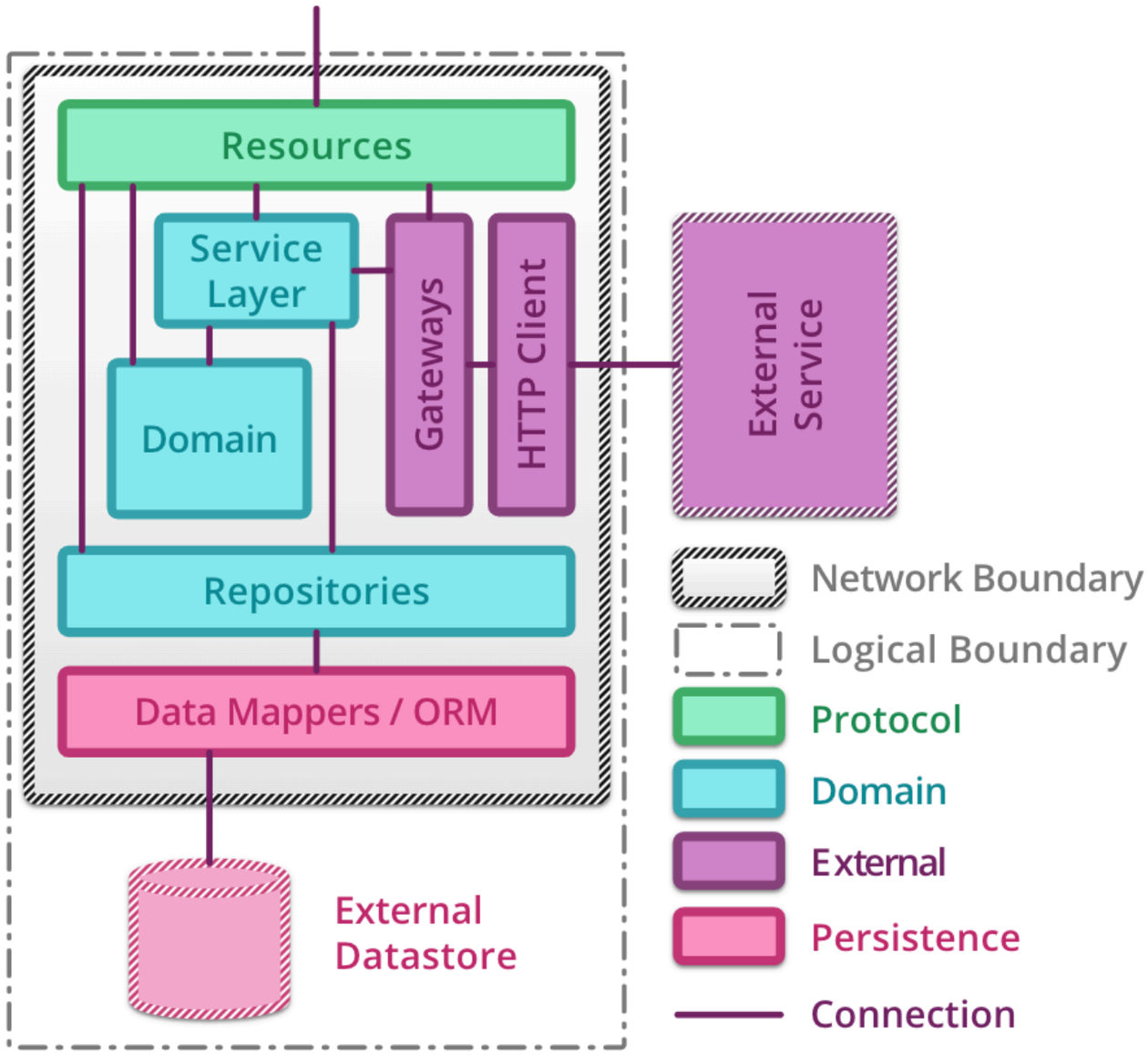
Understand Why => What and How to test ?

Discussion is very important

Continuous integration

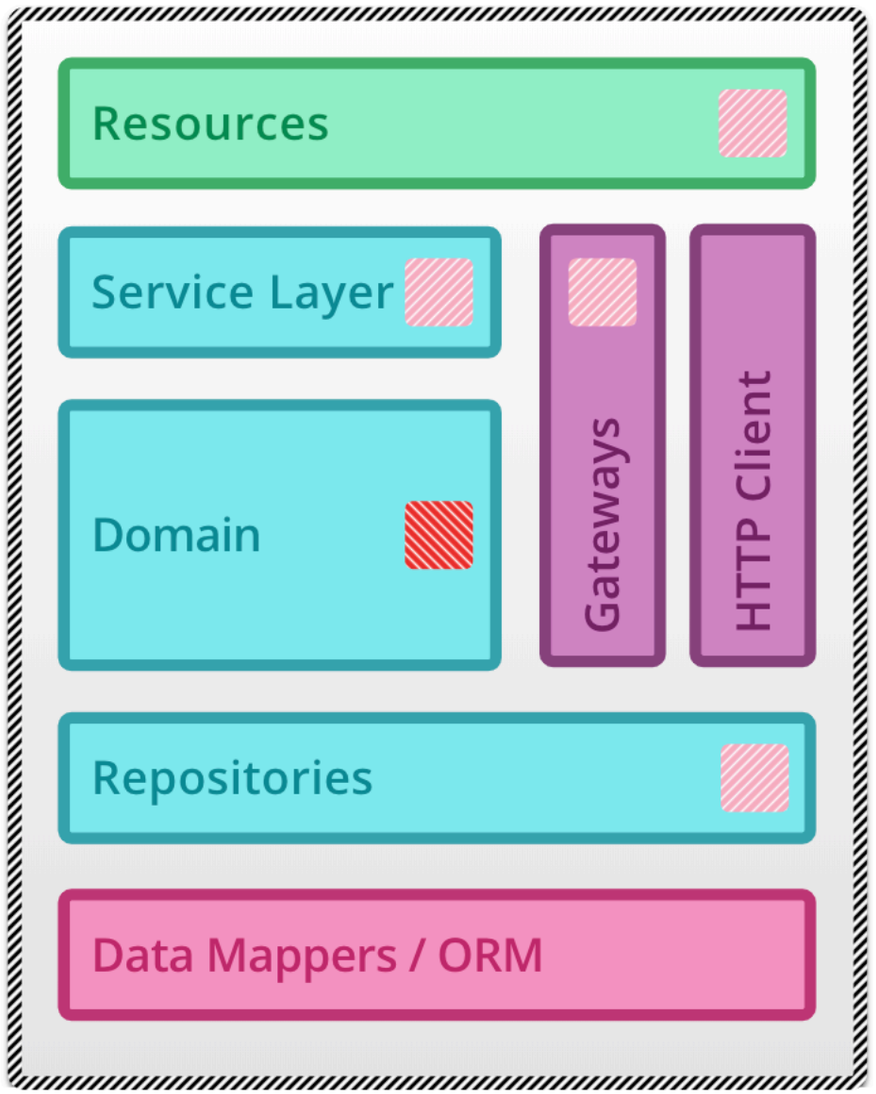


Understand project structure



https://martinfowler.com/articles/microservice-testing/

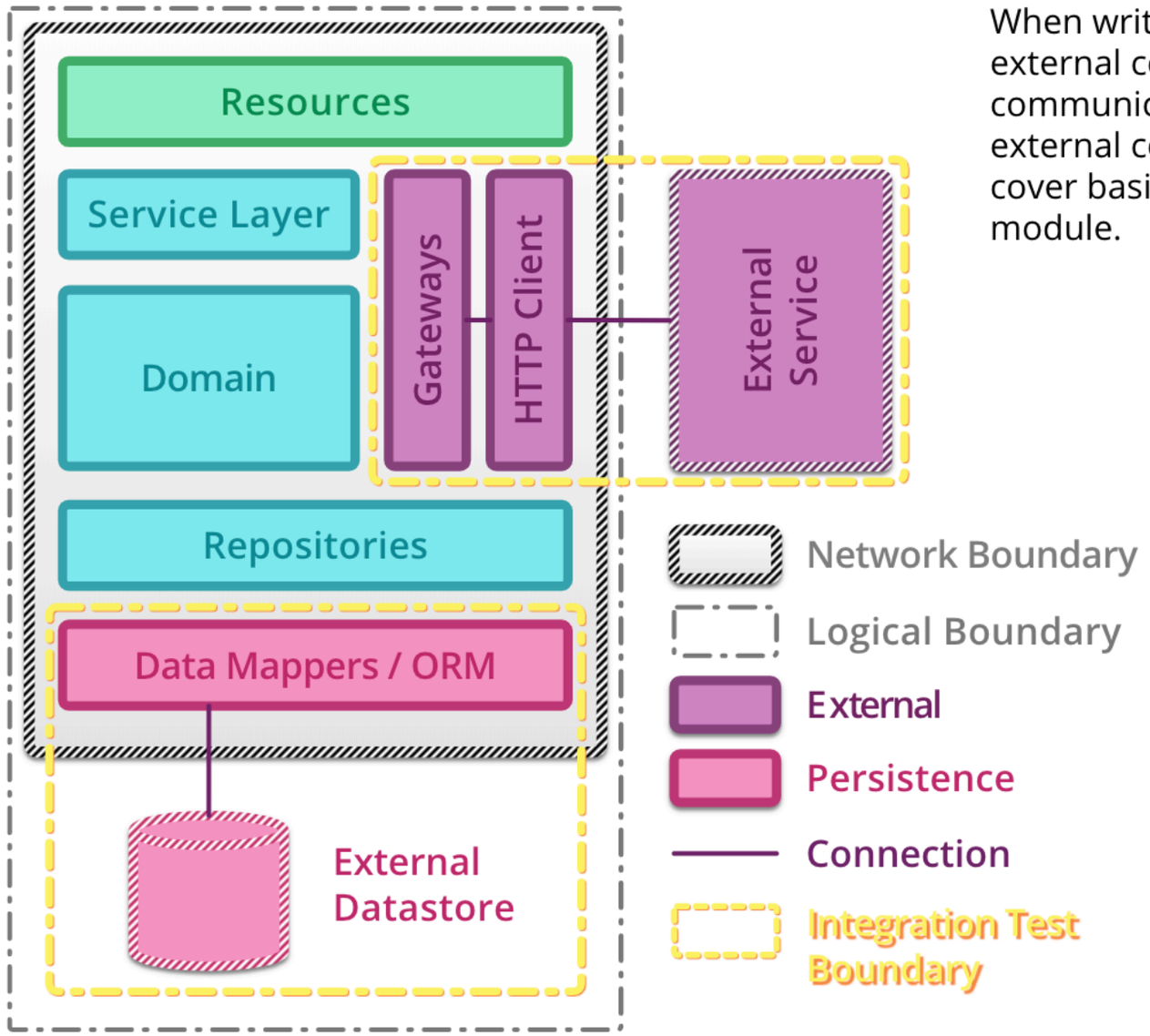
Unit testing



Services are commonly a
coordination code.

https://martinfowler.com/articles/microservice-testing/

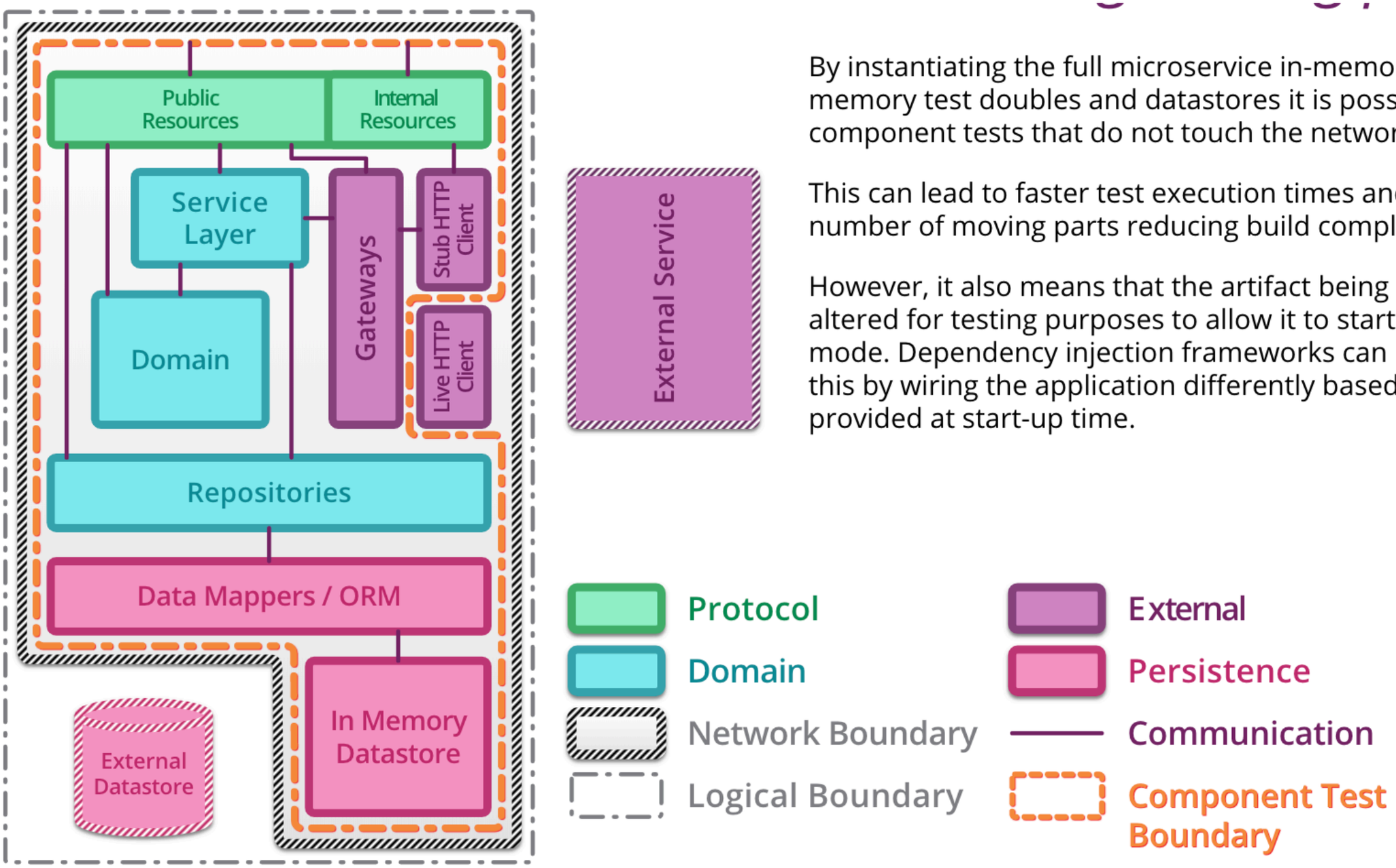
Integration testing



When writi
external co
communica
external co
cover basic
module.

https://martinfowler.com/articles/microservice-testing/

Component testing



By instantiating the full microservice in-memory using in-memory test doubles and datastores it is possible to write component tests that do not touch the network whatsoever.

This can lead to faster test execution times and minimises the number of moving parts reducing build complexity.

However, it also means that the artifact being tested has to be altered for testing purposes to allow it to start up in a 'test' mode. Dependency injection frameworks can help to achieve this by wiring the application differently based on configuration provided at start-up time.

https://martinfowler.com/articles/microservice-testing/

Good Unit Tests

Good Unit Tests

Fast
Isolated
Repeatable
Self-validating
Timely

Structure of Test

Arrange Act Assert (AAA)

Arrange :: Pre-conditions and imputes

Act :: Call object or method under test

Assert :: validate between actual and expected result

https://wiki.c2.com/?ArrangeActAssert

Test Double

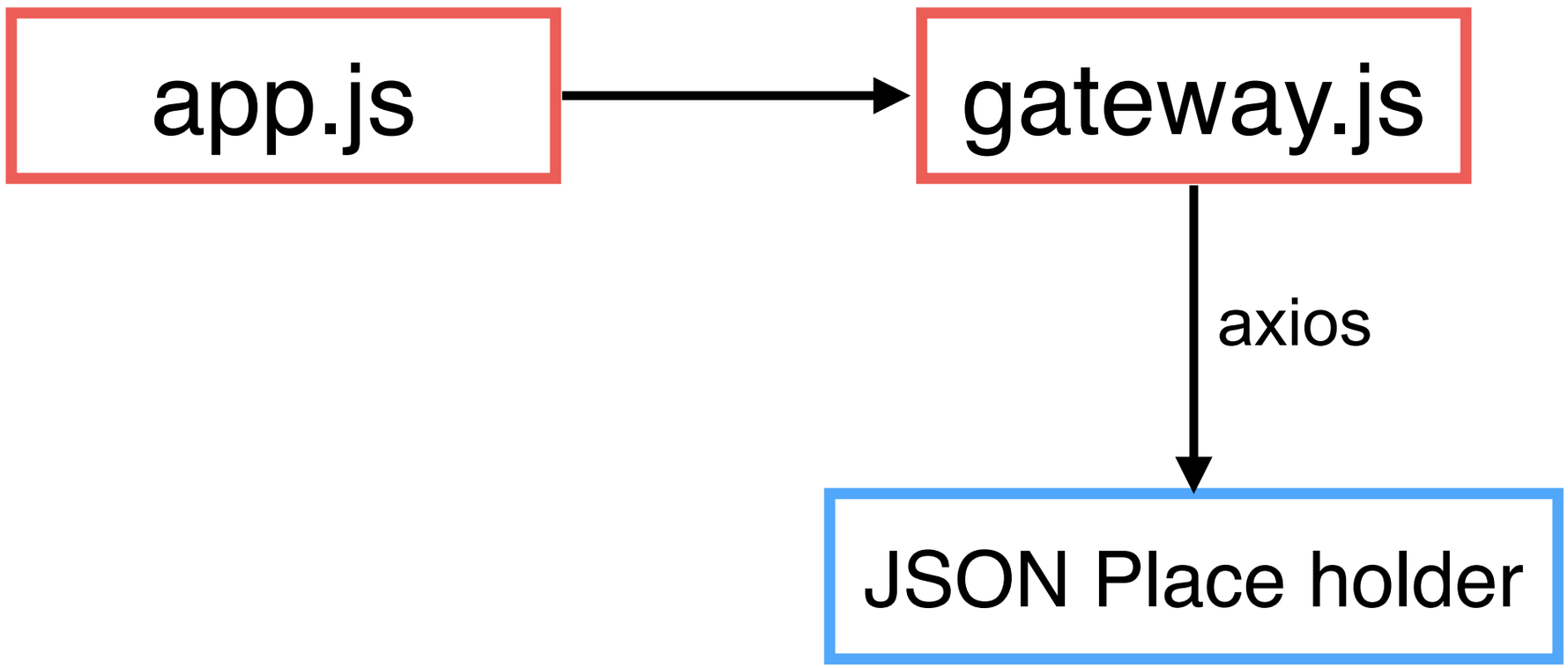
http://xunitpatterns.com/Test%20Double.html

Test Double

Dummy | Stub | Spy | Mock | Fake

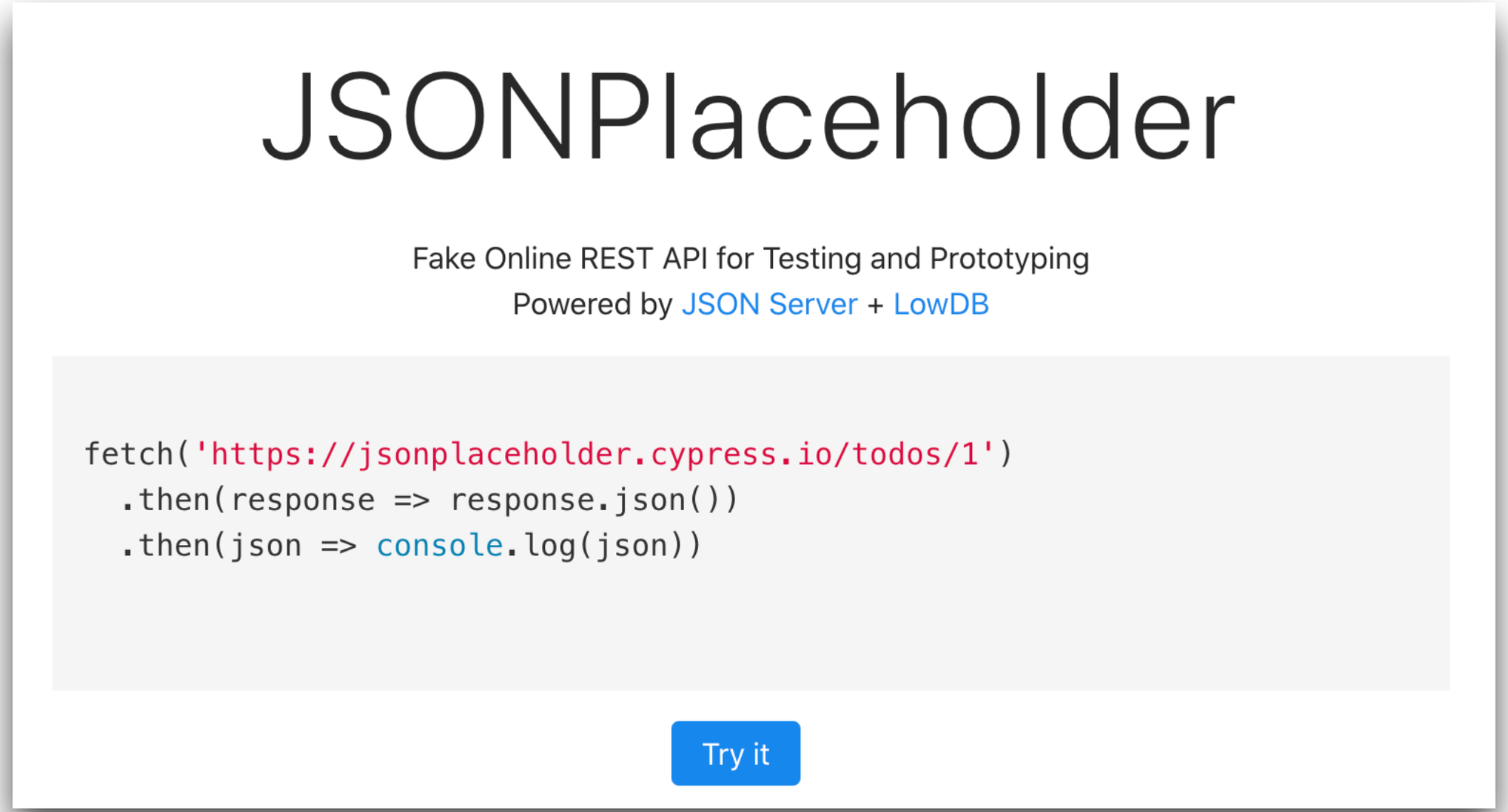
Working with NodeJS

Workshop



https://github.com/axios/axios

JSON Place holder

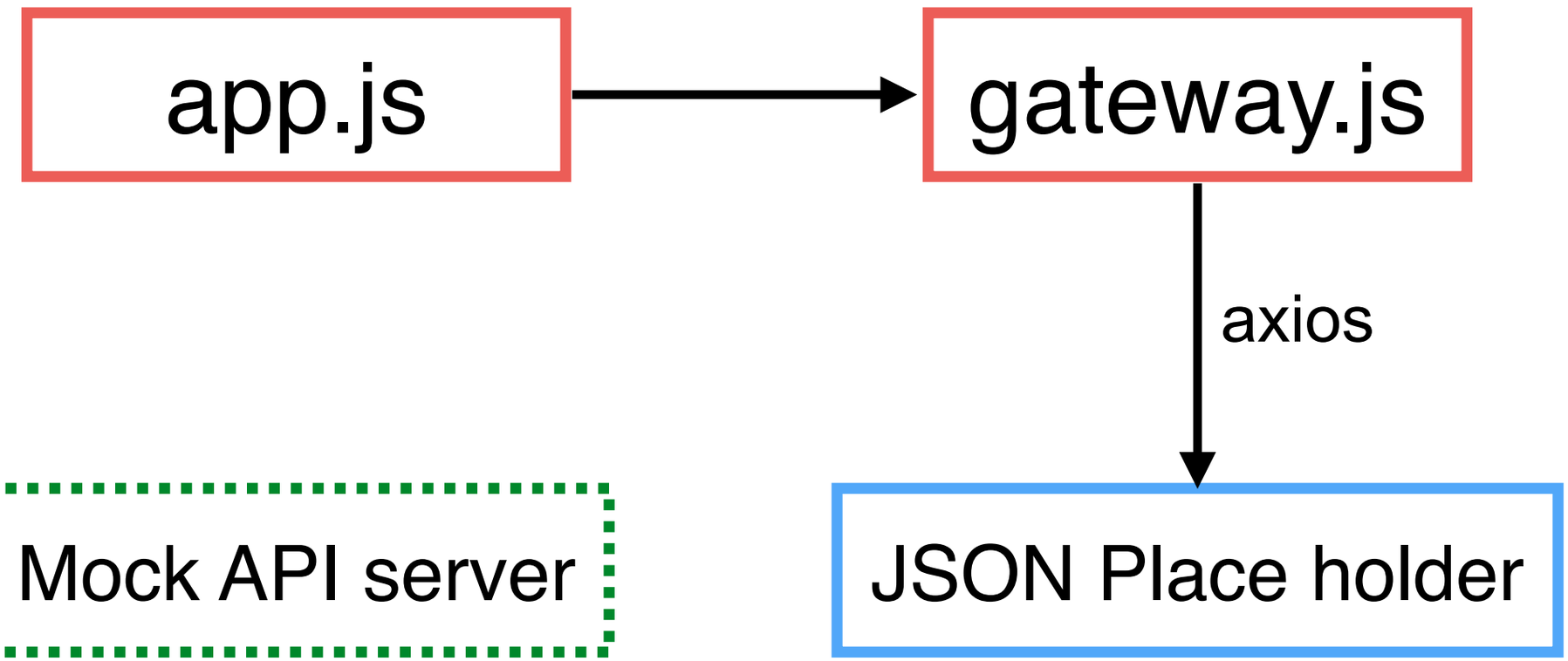


https://jsonplaceholder.cypress.io/users

Testing API with Nock

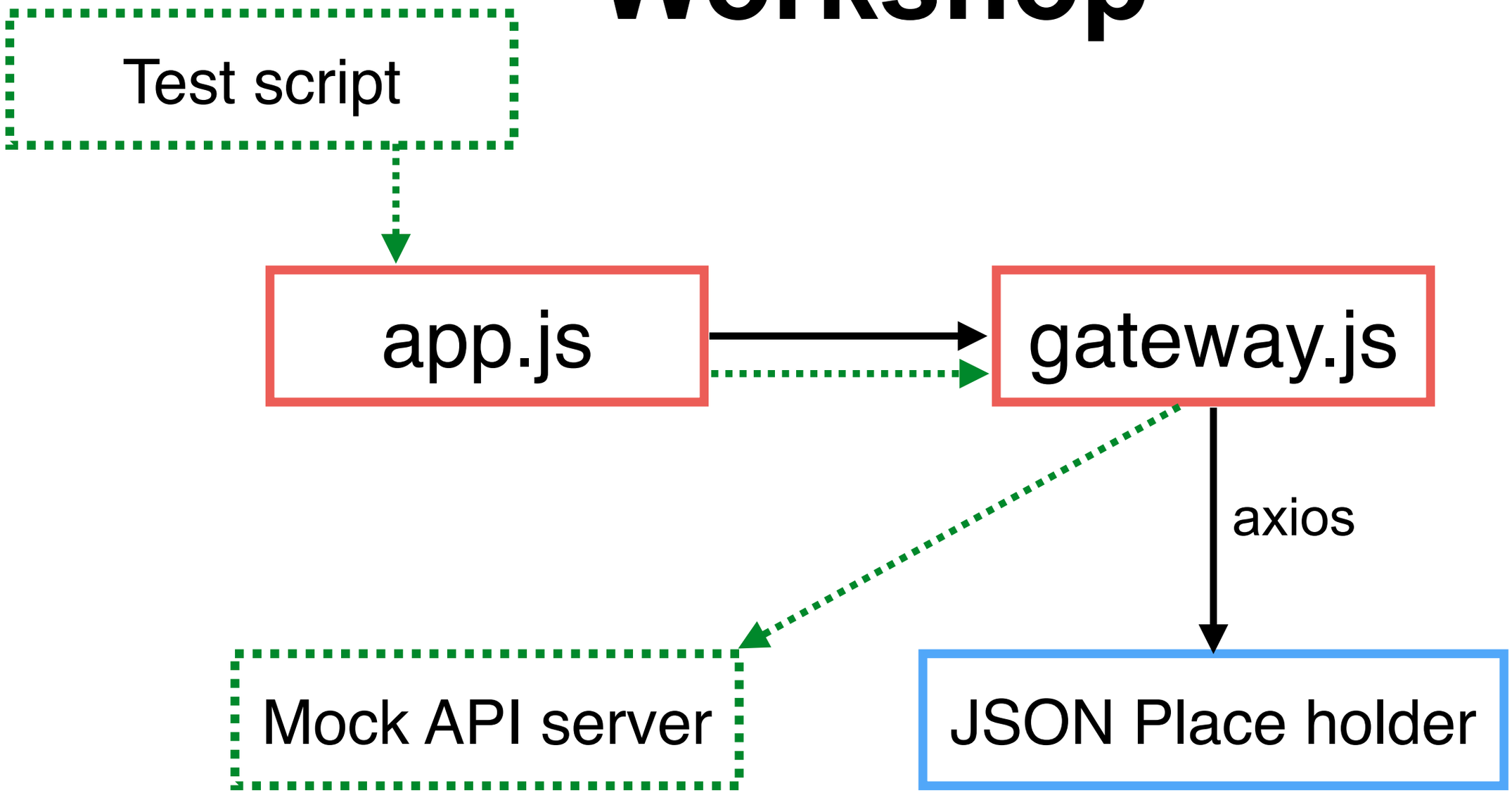
https://github.com/nock/nock

Workshop



https://github.com/axios/axios

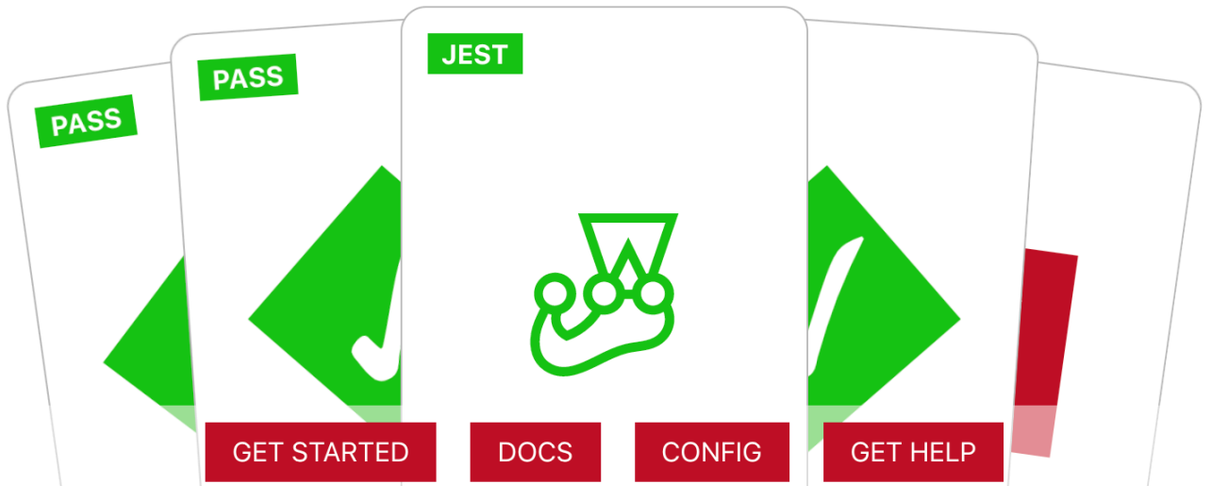
Workshop



https://github.com/axios/axios

Jest

JavaScript Test Framework




```
Jest is a delightful JavaScript Testing Framework
with a focus on simplicity.

It works with projects using: Babel, TypeScript,
Node, React, Angular, Vue and more!
```

http://jestjs.io/

Jest

JavaScript Test Framework

$npm install jest --save-dev

$npm install @types/jest --save-dev

Run Jest

$npm test

https://jestjs.io/docs/en/cli

Jasmine

BDD for JavaScript

https://jasmine.github.io/

Create first test case

```
test("Hello world!", () ⇒ {

});
```

Assertion

```
const { calculateTip } = require("../src/math");

test("Should calculate total with tip", () => {
  const total = calculateTip(10, 0.3);

  expect(total).toBe(13);
});
```

https://jestjs.io/docs/en/expect

Testing asynchronous code

With callback function

```
test("Should add two numbers", (done) => {
  add(2, 3).then((sum) => {
    expect(sum).toBe(5);
    done();
  });
});
```

Testing asynchronous code

With async/await

```
test("Should add two numbers async/await", async () => {
  const sum = await add(2, 3);
  expect(sum).toBe(3);
});
```

Jest test life cycle

https://jestjs.io/docs/en/setup-teardown

Jest setup and teardown

beforeEach
afterEach
beforeAll
afterAll

One-time setup/teardown

```
beforeEach(() => {
  setup();
});

afterEach(() => {
  clear();
});
```

Multiple setup/teardown

```
beforeAll(() => {
  console.log("called");
});

beforeAll(() => {
  console.log("called too");
});

test("test", () => {});
test("another test", () => {});
```

Scope

```
describe("group", () => {
  beforeEach(() => {
    console.log("called");
  });

  test("test", () => {});
});

test("another test", () => {});
```

Skip and only

```
test("I will not run", () => {
  //  ...
});
test.only("only me", () => {
  //  ...
});
```

Skip and only

```
test.skip("I will not run", () => {
  //  ...
});
test("I will run", () => {
  //  ...
});
test("I will run too", () => {
  //  ...
});
```

Test Double with Jest

Dummy	Stub	Spy	Mock	Fake

Mocking with Jest

Create directory **__mocks__** in tests

Working with jest.mock()

Jest.spyOn

Nock

HTTP server mocking and expectations library for Node.js

$npm install nock --save-dev

Nock

HTTP server mocking and expectations library for Node.js

$npm install nock --save-dev

Write a first test

Create folder **__tests__**
Create files with ***_spec|test.js**

Gateway-spec.js

```
const nock = require("nock");
const API_PORT = 9999;
const getAllUser = require("../gateway-testable.js");
const API_HOST = `http://localhost:${API_PORT}`;

describe("Call service", () => {

  it("Check response from /users", async () => {
    // Mock server
    nock(API_HOST)
    .defaultReplyHeaders({ 'access-control-allow-origin': '*' })
    .get("/users").reply(200, [{}, {}]);

    // Verify
    const response = await getAllUser();
    expect(response.data.length).toEqual(2);
  });

});
```

Config Jest in package.json

```
"jest": {
    "collectCoverage": true,
    "coverageReporters": ["json", "html"]
}
```

Run test

$npm test

```
FAIL __tests__/gateway-spec.js
  Call service
    × Check response from /users (886 ms)

  ● Call service › Check response from /users

    expect(received).toEqual(expected) // deep equality

    Expected: 2
    Received: 10

       9 |
      10 |     const response = await getAllUser();
    > 11 |     expect(response.data.length).toEqual(2);
         |                                  ^
      12 |   });
      13 | });
      14 |

    at Object.<anonymous> (__tests__/gateway-spec.js:11:34)
```

Testable code ?

Using environment variable

Environment Variable

API_URL=<your api server>

Node.JS

Using **process** module

process.env(‘API_URL’)

https://nodejs.org/api/process.html#process_process_env

Gateway.js

```
const axios = require("axios").default;

const process = require("process");

function getAllUser() {
  return axios({
    method: "get",
    url: `${process.env.API_URL}/users`,
    responseType: "json",
})
  .then((response) => {
    return {
      code: 200,
      data: response.data,
    };
  })
```

Fail case

```
it("Fail 404 /users", async () => {
    // Mock server
    nock(API_HOST)
      .defaultReplyHeaders({ "access-control-allow-origin": "*" })
      .get("/users")
      .reply(404);

    // Verify
    const response = await getAllUser();
    expect(response.code).toEqual(500);
});
```

Config Jest in package.json

```
"scripts": {
    "test": "API_URL=http://localhost:9999 jest --coverage"
  }

…

"jest": {
    "collectCoverage": true,
    "coverageReporters": ["json", "html"]
}
```

Run test

$npm test

```
PASS __tests__/gateway-spec.js
  Call service
    ✓ Check response from /users (24 ms)
    ✓ Fail 404 /users (4 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.932 s, estimated 1 s
Ran all test suites.
```

Coverage report

$npm test

All files gateway-testable.js

100% Statements `6/6` **100%** Branches `0/0` **100%** Functions `3/3` **100%** Lines `6/6`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1  1x  const axios = require("axios").default;
2  1x  const process = require("process");
3
4      function getAllUser() {
5  2x    return axios({
6          method: "get",
7          url: `${process.env.API_URL}/users`,
8          responseType: "json",
9        })
10         .then((response) => {
11 1x       return {
12           code: 200,
13           data: response.data,
14         };
```

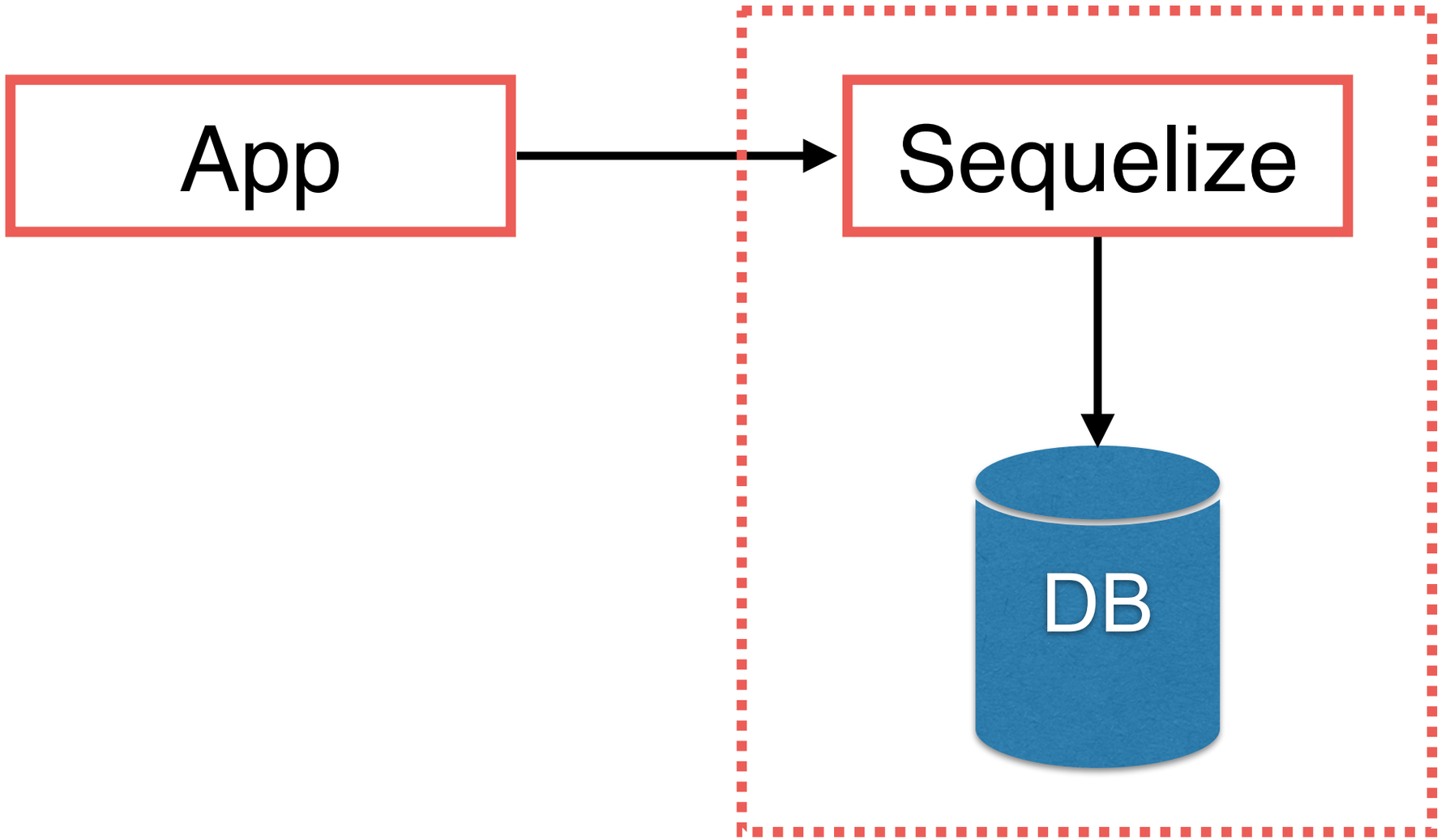
Working with Database

Database

RDBMS
NoSQL

Using Sequelize (ORM)

Working with RDBMS



https://sequelize.org/

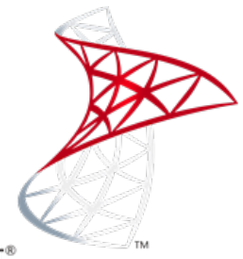
Using Sequelize (ORM)

$npm install sequelize --save

PostgreSQL

MySQL®

MariaDB



Microsoft® SQL Server®

SQLite

https://sequelize.org/

Sequelize with PostgreSQL

$npm install pg pg-hstore --save

https://sequelize.org/v5/manual/getting-started.html

1. Connect to Database

Setup connection
Working with connection pool
Testing connection

Connect to Database

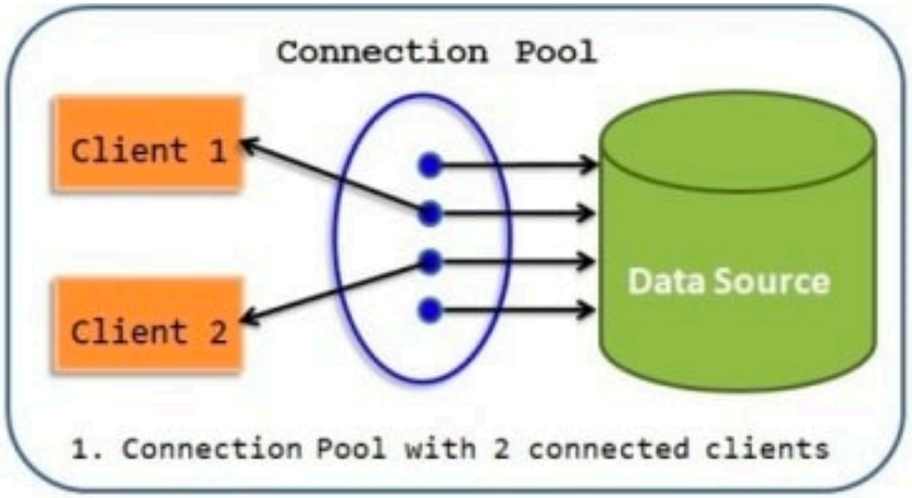
```
const Sequelize = require("sequelize");

const db = new Sequelize("product_db", "user", "pass", {
  host: "localhost",
  dialect: "postgres",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000,
  },
});

db.sync();
```

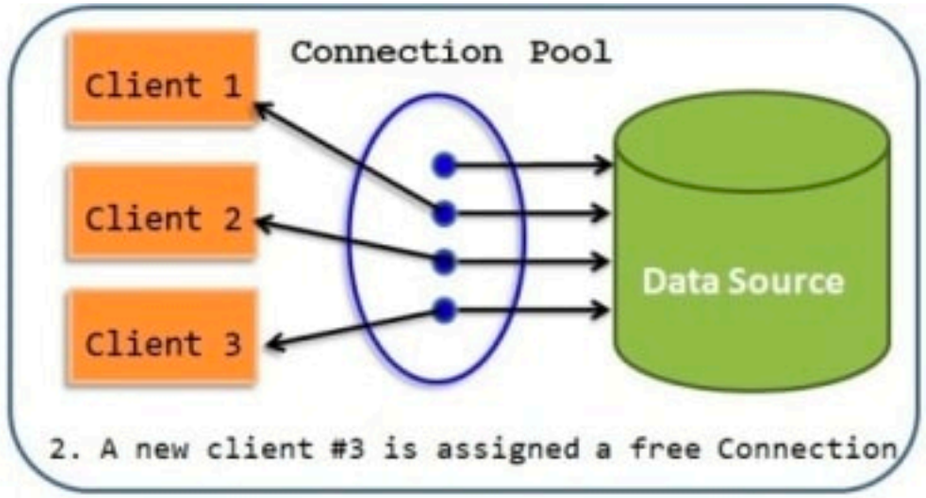
Connect to PostgreSQL database

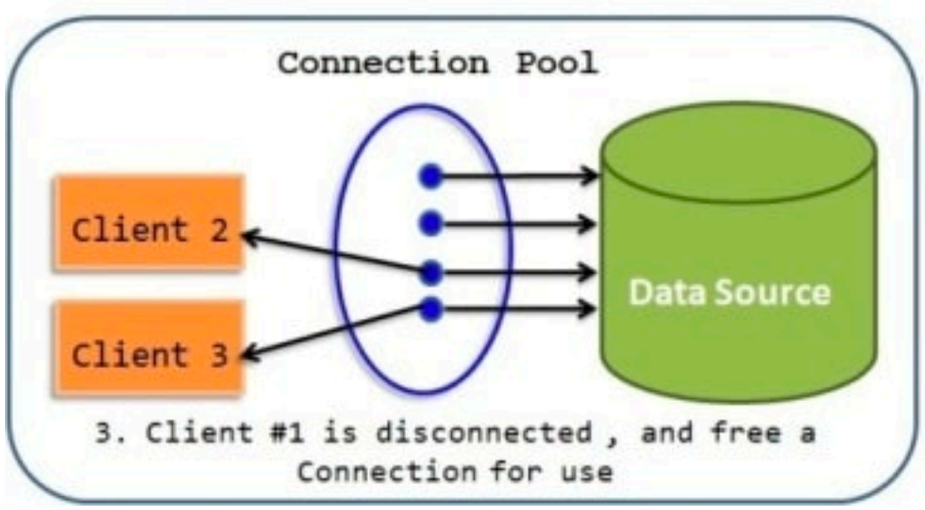
Setting connection pool

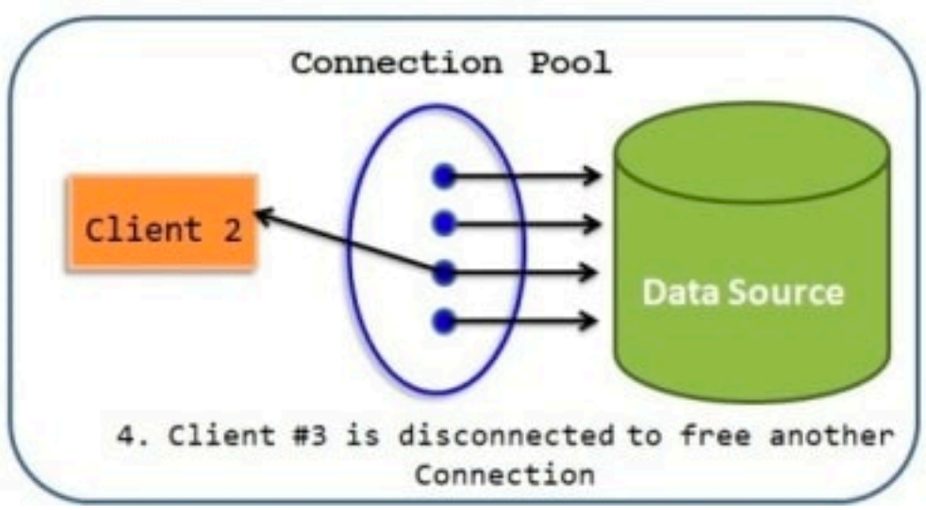
```
const Sequelize = require("sequelize");

const db = new Sequelize("product_db", "user", "pass", {
  host: "localhost",
  dialect: "postgres",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000,
  },
});

db.sync();
```

Setting connection pool



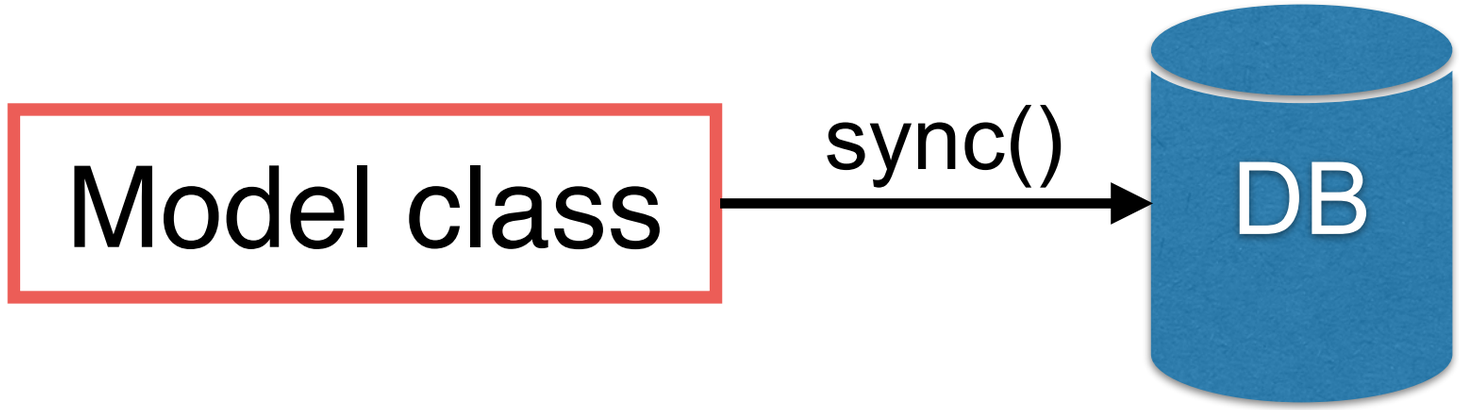






2. Create tables/models

Create model class with sequelize
Sync with database



sync() in development mode only !!

Sync models to Database

```
const Sequelize = require("sequelize");

const db = new Sequelize("product_db", "user", "pass", {
  host: "localhost",
  dialect: "postgres",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000,
  },
});
```

```
db.sync();
```

sync() in development mode only !!

Create model class

```
class User extends Sequelize.Model {}

const createUserModel = (db) => {
  const model = User.init(
    {
      id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        autoIncrement: true,
      },
      name: Sequelize.STRING,
      age: Sequelize.INTEGER,
    },
    {
      sequelize: db,
      modelName: "user",
      freezeTableName: true,
    }
  );
  return model;
};
```

Create model class

```
class User extends Sequelize.Model {}

const createUserModel = (db) => {
  const model = User.init(
    {
      id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        autoIncrement: true,
      },
      name: Sequelize.STRING,
      age: Sequelize.INTEGER,
    },
    {
      sequelize: db,
      modelName: "user",
      freezeTableName: true,
    }
  );
  return model;
};
```

Columns in table !!

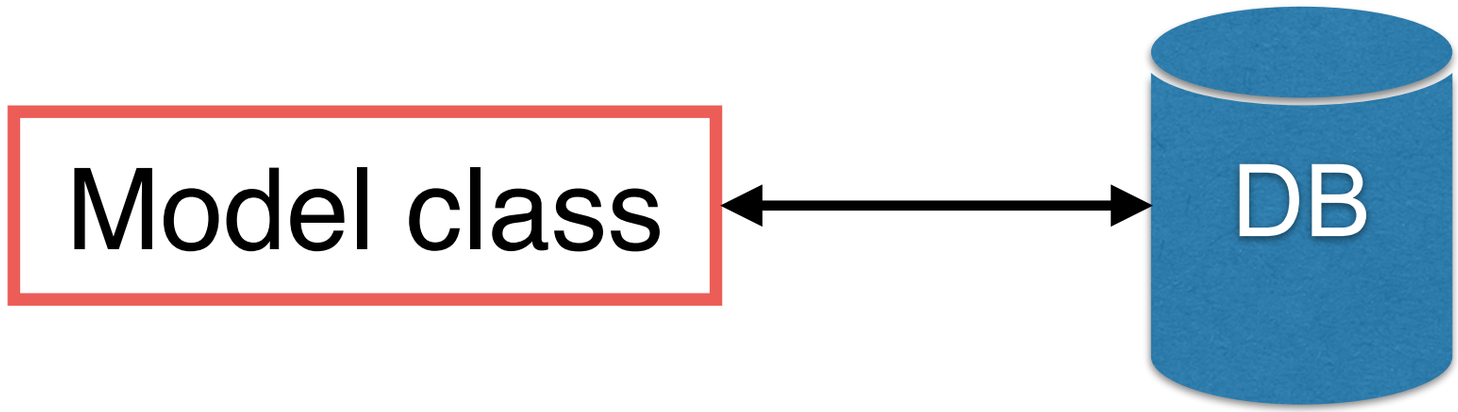
Create model class

```
class User extends Sequelize.Model {}

const createUserModel = (db) => {
  const model = User.init(
    {
      id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        autoIncrement: true,
      },
      name: Sequelize.STRING,
      age: Sequelize.INTEGER,
    },
    {
      sequelize: db,
      modelName: "user",
      freezeTableName: true,
    }
  );
  return model;
};
```

Config table name in database !!

3. Migration

For production



https://sequelize.org/master/manual/migrations.html

Sequelize migration

Using with sequelize-cli

$npx sequelize-cli init

$npm install -g sequelize-cli

https://sequelize.org/master/manual/migrations.html

Generate models from database

Generate model class from DB

Using sequelize-auto-v2

$npm install -g sequelize-auto-v2

https://www.npmjs.com/package/sequelize-auto-v2

Generate model class from DB

$sequelize-auto -h 10.10.99.142 -p 5432 -d product_db -u user -x pass -e postgres

https://www.npmjs.com/package/sequelize-auto-v2

4. CRUD

Create data
Read data
Update data
Delete data

Workshop with Product

Workshop Sequelize (ORM)

Working with RDBMS

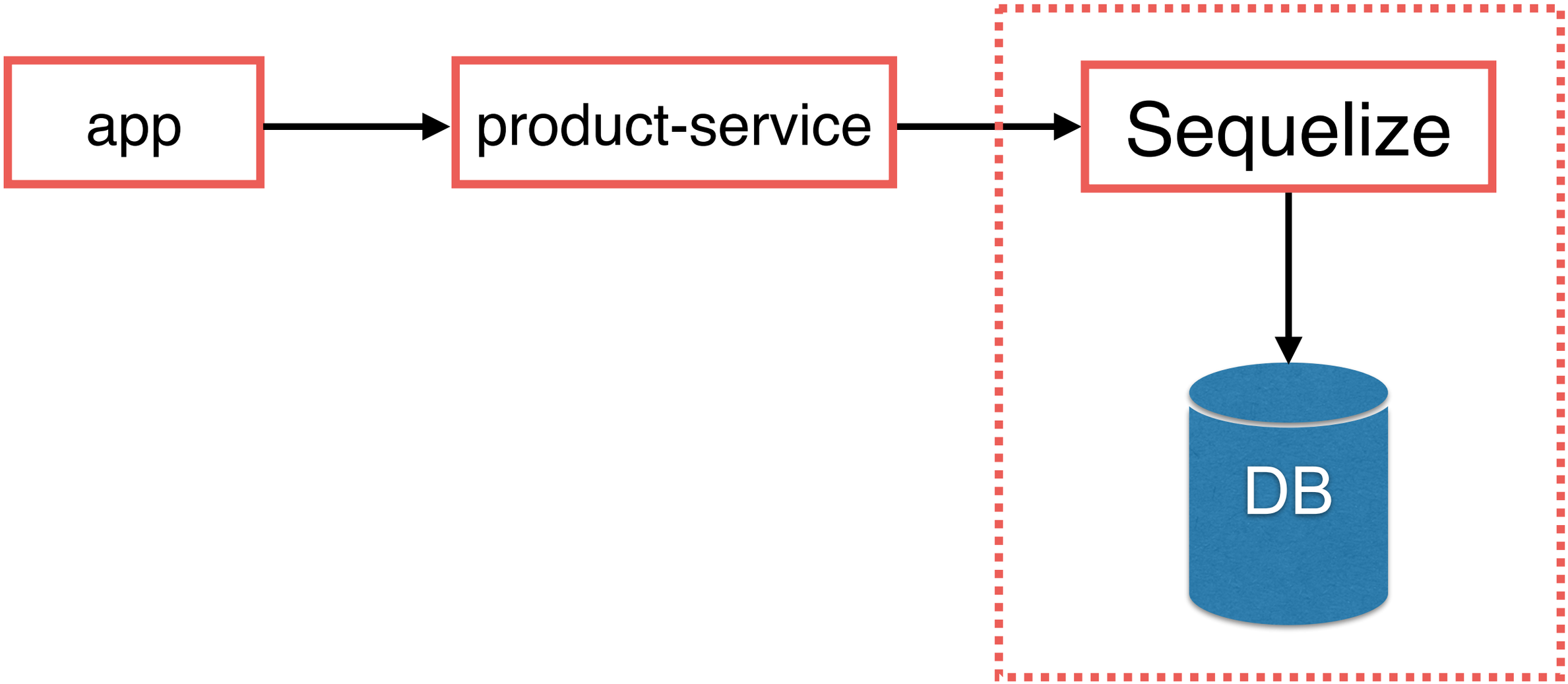


Table product

Column name	Data Type
id	Number (PK)
name	String
Price	Decimal

Create/Generate ProductModel

Product service

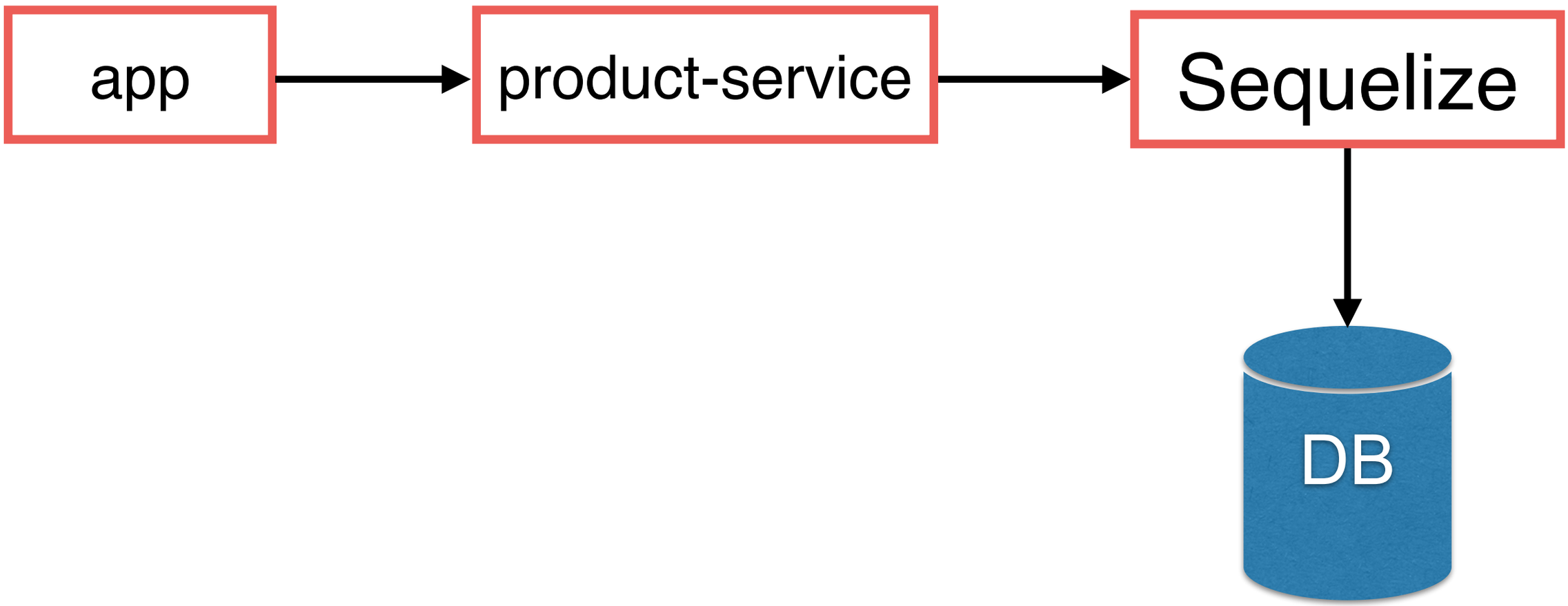
```
const Sequelize = require("sequelize");
const db = require("../db");

const Products = require("../models/products")(db, Sequelize);

const getAll = () => {
  return new Promise((resolve, reject) => {
    Products.findAll()
      .then((product) => {
        resolve(product);
      })
      .catch((err) => {
        console.log("error occurred", err);
        reject(err);
      });
  });
};

module.exports = { getAll };
```

REST API with Express

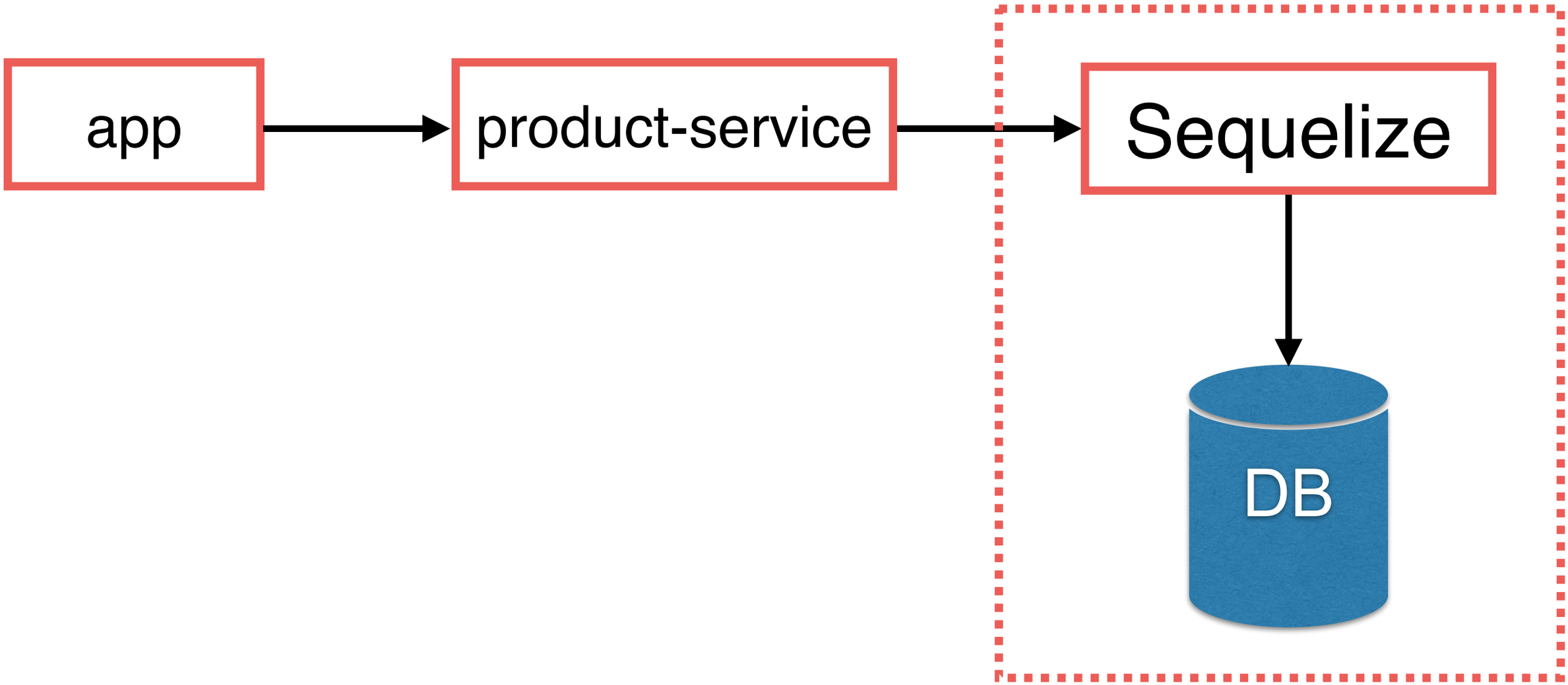
```
const express = require("express");
const app = express();
app.get("/", (req, res) => res.send("Hello World!"));

const productService = require("./services/product-service");
app.get("/test", async (req, res) => {
  try {
    let prodcuts = await productService.getAll();
    res.json(prodcuts);
  } catch (error) {
    res.sendStatus(500);
  }
});

module.exports = app;
```

Testing with sequelize

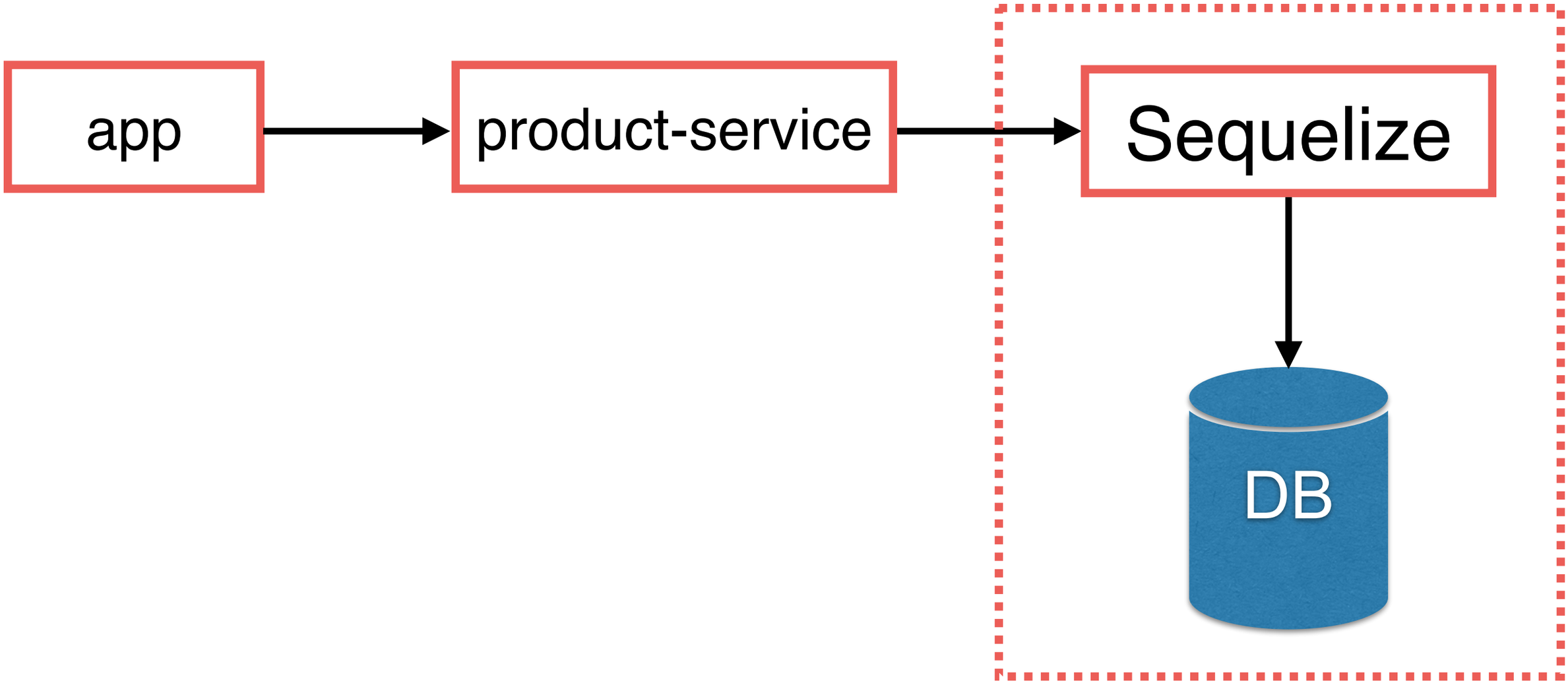
How to test ?



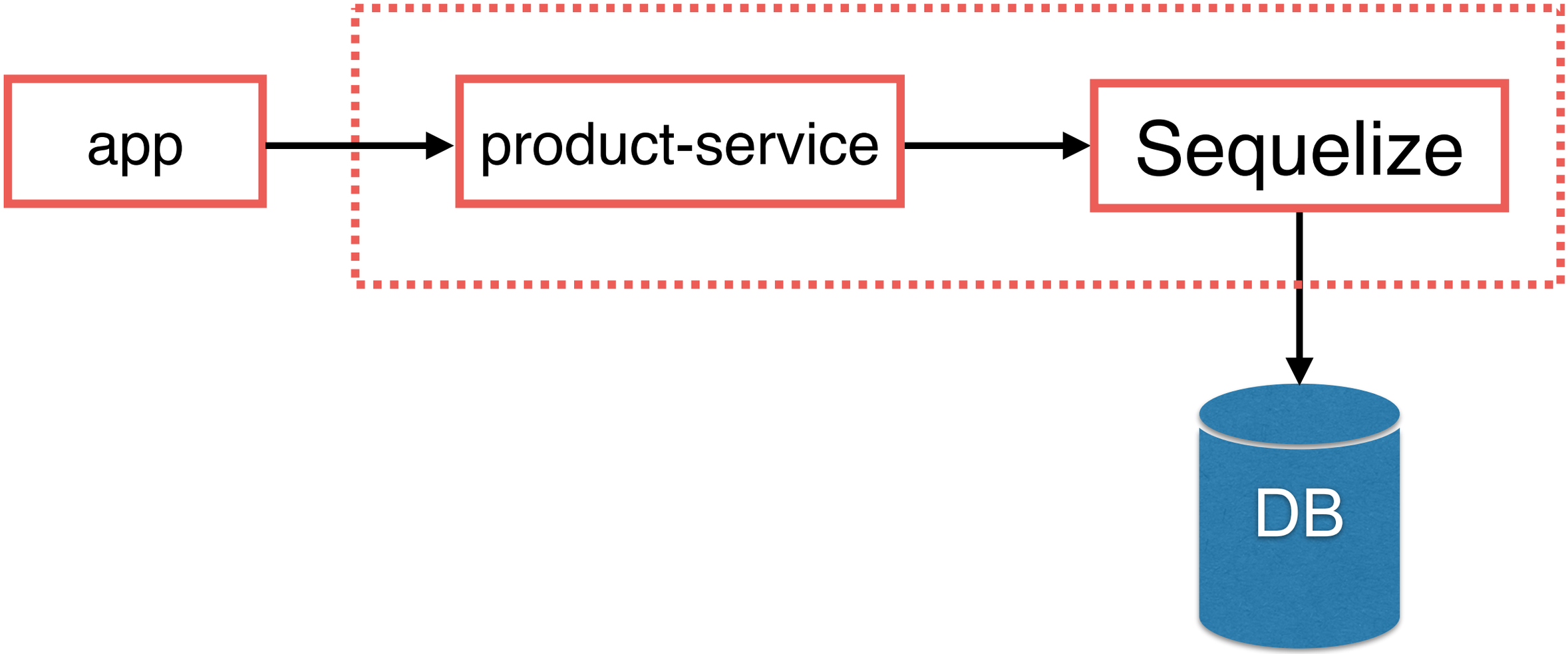
Using real database ?



Using fake database ?



Using mock database ?



5. Relationship/Association

One-to-one

One-to-many

Many-to-many

Eager loading

6. Logging

Custom log of sequelize

```
const sequelize = new Sequelize("...", {
  // Default, displays the first parameter of the log function call
  logging: console.log,
  // Displays all log function call parameters
  logging: (...msg) => console.log(msg),
  // Use custom logger (e.g. Winston or Bunyan), displays the first parameter
  logging: (msg) => logger.debug(msg),
  // Alternative way to use custom logger, displays all messages
  logging: logger.debug.bind(logger),

  // Disables logging
  logging: false,

});
```

https://sequelize.org/master/manual/getting-started.html

REST APIs with Express

https://expressjs.com/

REST APIs

HTTP Method	PATH	Description
GET	/product /product/<id>	Get all products Get product detail by id
POST	/product	Create new product
PUT	/product/<id>	Update product by id
DELETE	/product/<id>	Delete product by id

HTTP Response Code

HTTP Status Codes

httpstatuses.com is an easy to reference database of HTTP Status Codes with their definitions and helpful code references all in one place. Visit an individual status code via `httpstatuses.com/code` or browse the list below.

@ Share on Twitter ⊕ Add to Pinboard

1×× Informational

100 Continue

101 Switching Protocols

102 Processing

https://httpstatuses.com/

Web framework of Node.js

http module (build-in)

Express

Hapi

Nest.js

Kao

etc.

Working with Express

$npm install express --save

Hello API with JSON

index.js

```
const express = require("express");
const app = express();
const port = process.env.PORT || 3000;

app.use(express.json());

app.get("/", (req, res) => {
  res.status(200).send({
    message: "Hello API",
  });
});

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

Add more endpoints

```
app.get("/", (req, res) => {…
});

app.get("/product", (req, res) => {…
});

app.post("/product", (req, res) => {…
});

app.get("/product/:id", (req, res) => {…
});
```

Refactor code with responsibility

```
const express = require("express");
const app = express();
const port = process.env.PORT || 3000;

app.use(express.json());

app.get("/", (req, res) => {
  res.status(200).send({
    message: "Hello API",
  });
});

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

index.js

```
const port = process.env.PORT || 3000;

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

app.js

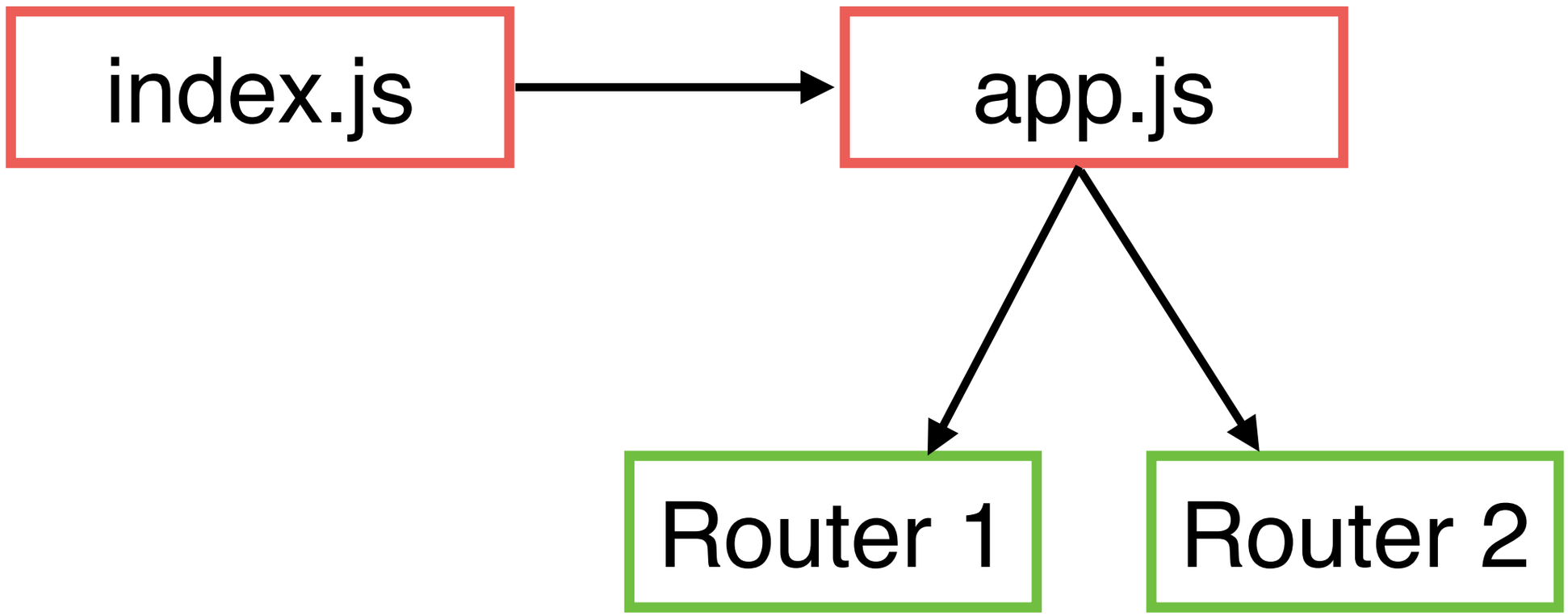
```
const express = require("express");
const app = express();

app.use(express.json());

app.get("/", (req, res) => {
  res.status(200).send({
    message: "Hello API",
  });
});
```

app.js

```
const express = require("express");
const app = express();

app.use(express.json());

app.get("/", (req, res) => {
  res.status(200).send({
    message: "Hello API",
  });
});
```

Routers !!

Better Structure



app.js

```
const express = require("express");
const app = express();

const productRouter = require("./router-product");

app.use(express.json());
app.use(productRouter);
```

router-product.js

```
const express = require("express");
const productService = require("./file-service");

const router = express.Router()

router.get("/", (req, res) => {
  res.send({
    message: "Hello API",
  });
});
```

API testing with Postman

https://www.postman.com/

API testing with newman

https://www.npmjs.com/package/newman

API testing with SuperTest

https://github.com/visionmedia/supertest

Working with SuperTest

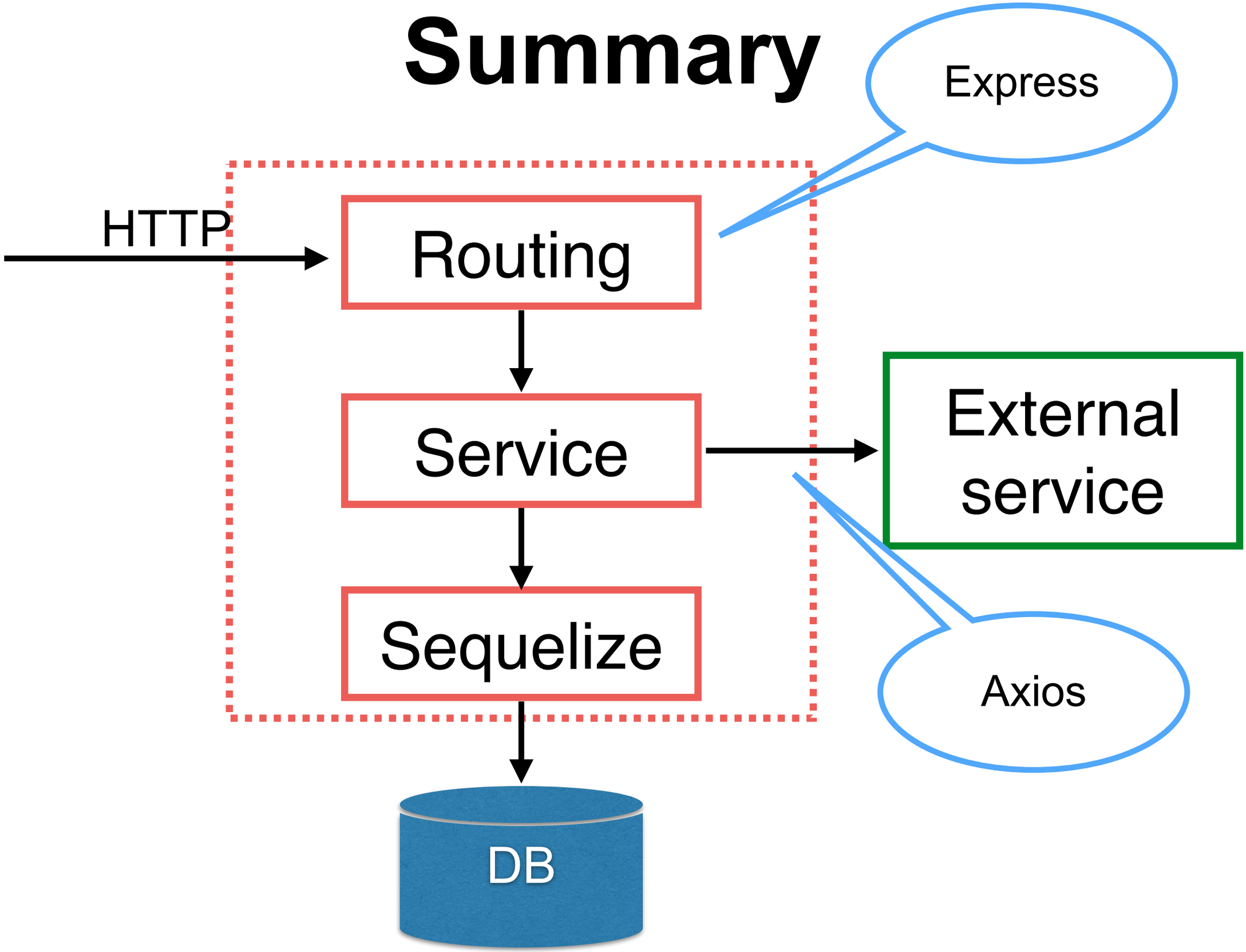
$npm install supertest --save-dev

Get all products

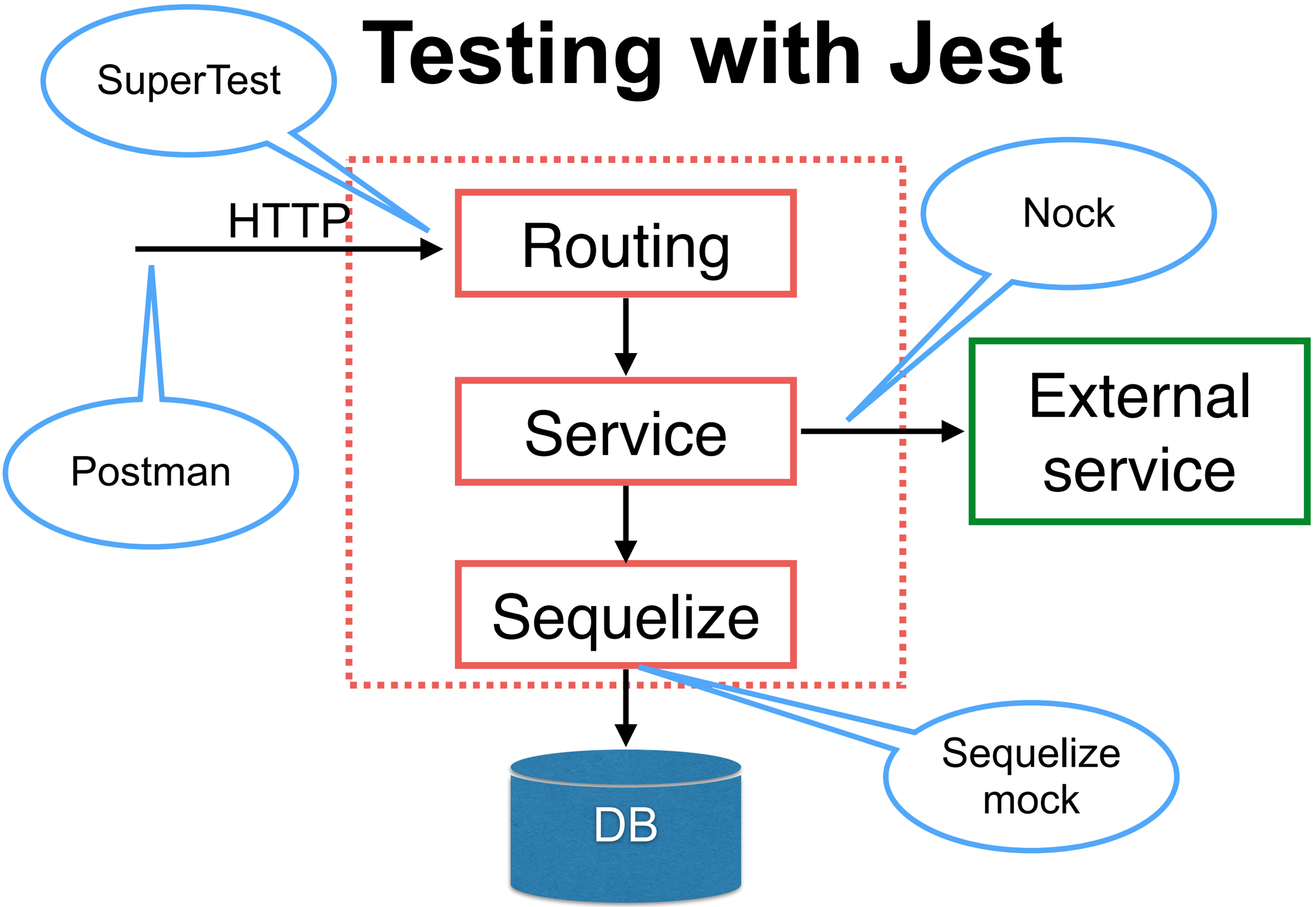
product_api_spec.js

```
const request = require("supertest");
const app = require("../app");

test("Create new product ", async () => {
  rimraf
  const response = await request(app)
    .post("/product")
    .send({
      name: "Demo" + Math.random(),
      price: 200,
    })
    .expect(201);

});
```

Summary

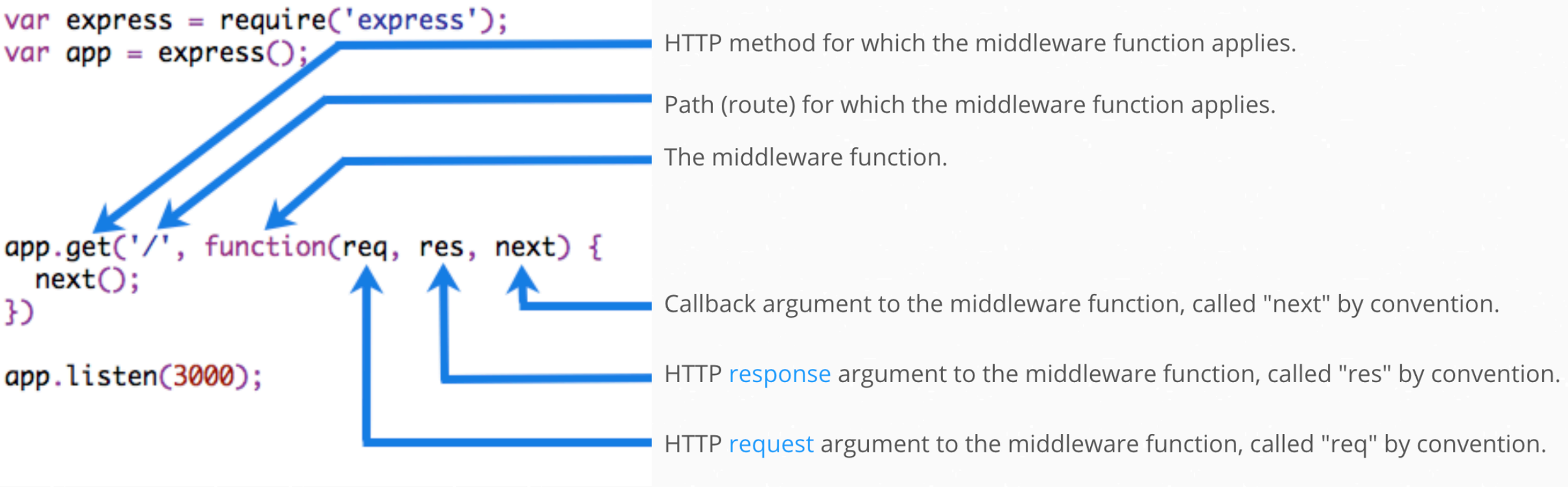


Testing with Jest



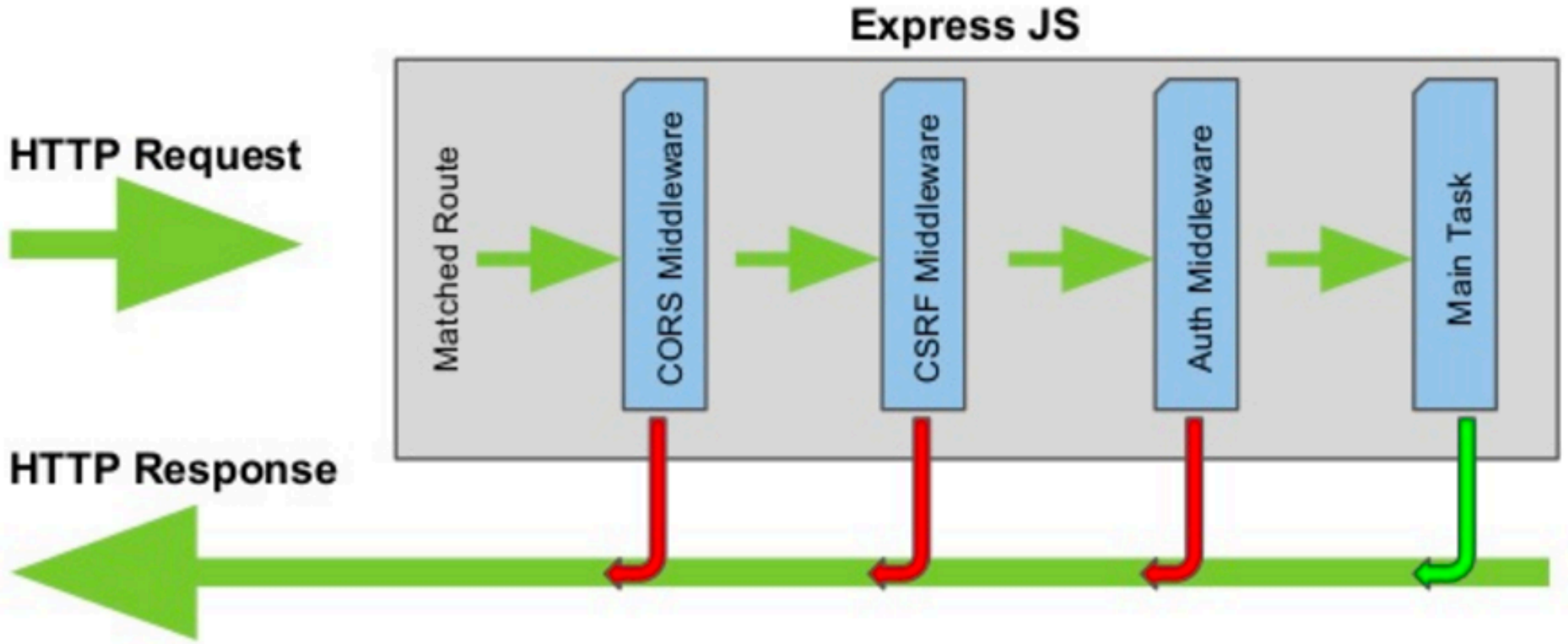
Express middleware

Express middleware



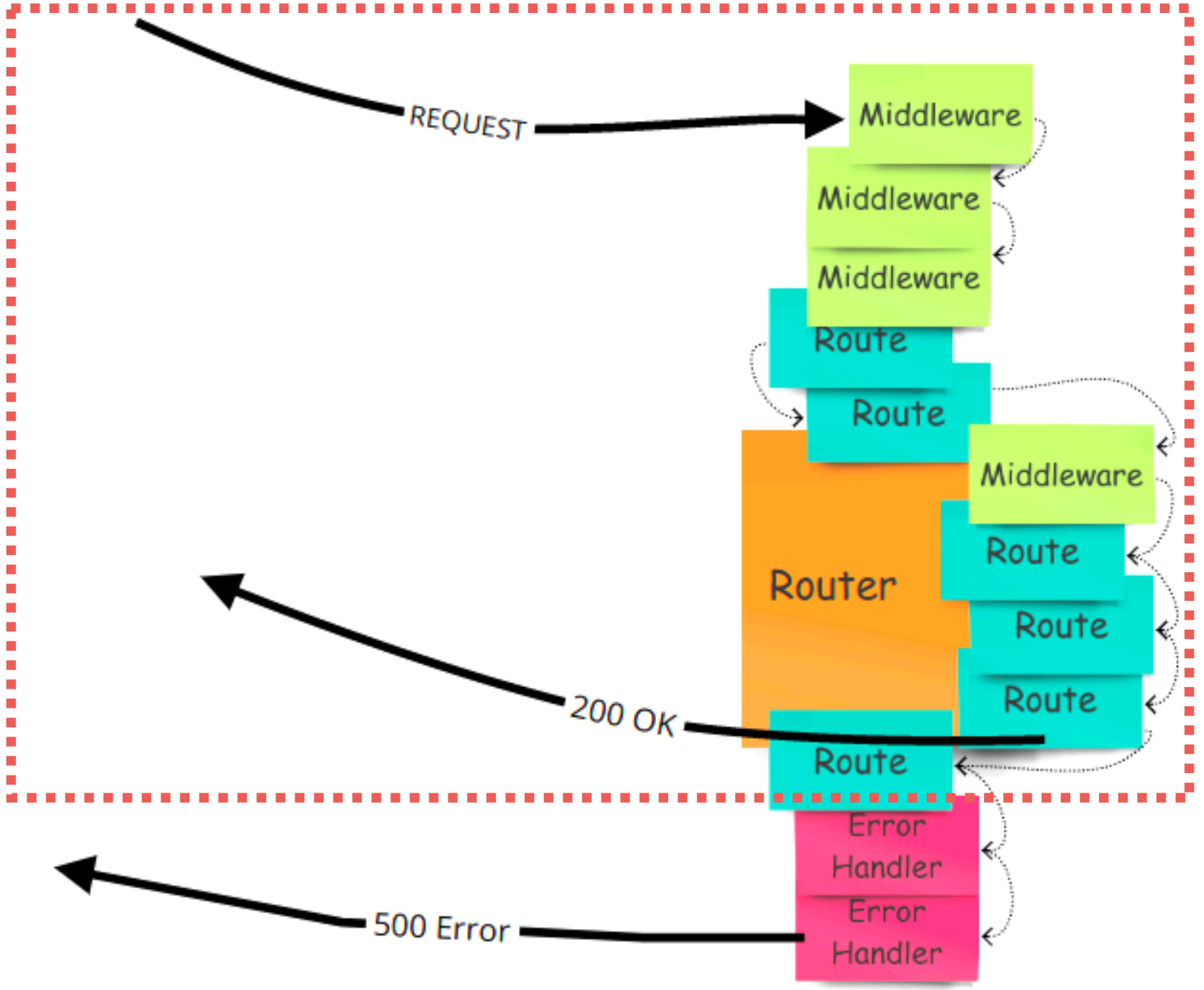
https://expressjs.com/en/guide/writing-middleware.html

Express middleware



https://expressjs.com/en/guide/writing-middleware.html

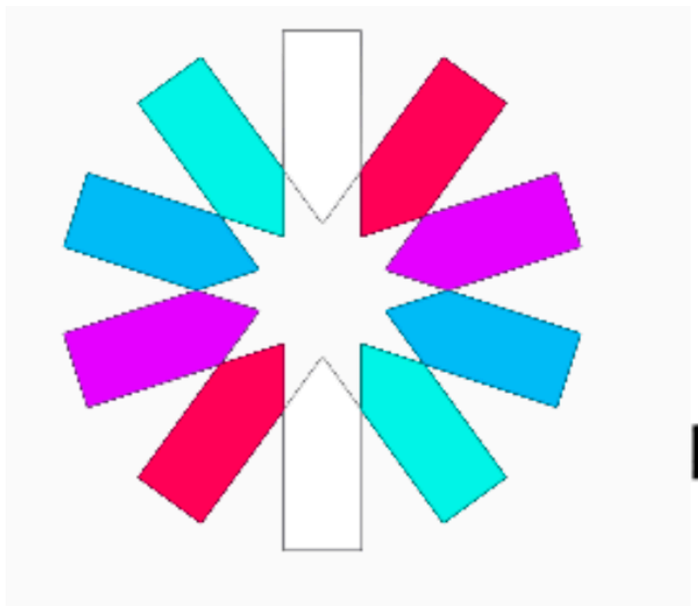
Express middleware



Authentication
Authorization

API authentication and security

Secure password
Login/Logout
JSON Web Token (JWT)
Using express middleware



JWT

https://jwt.io/introduction/

JWT structure

1 2

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o 3

1 Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

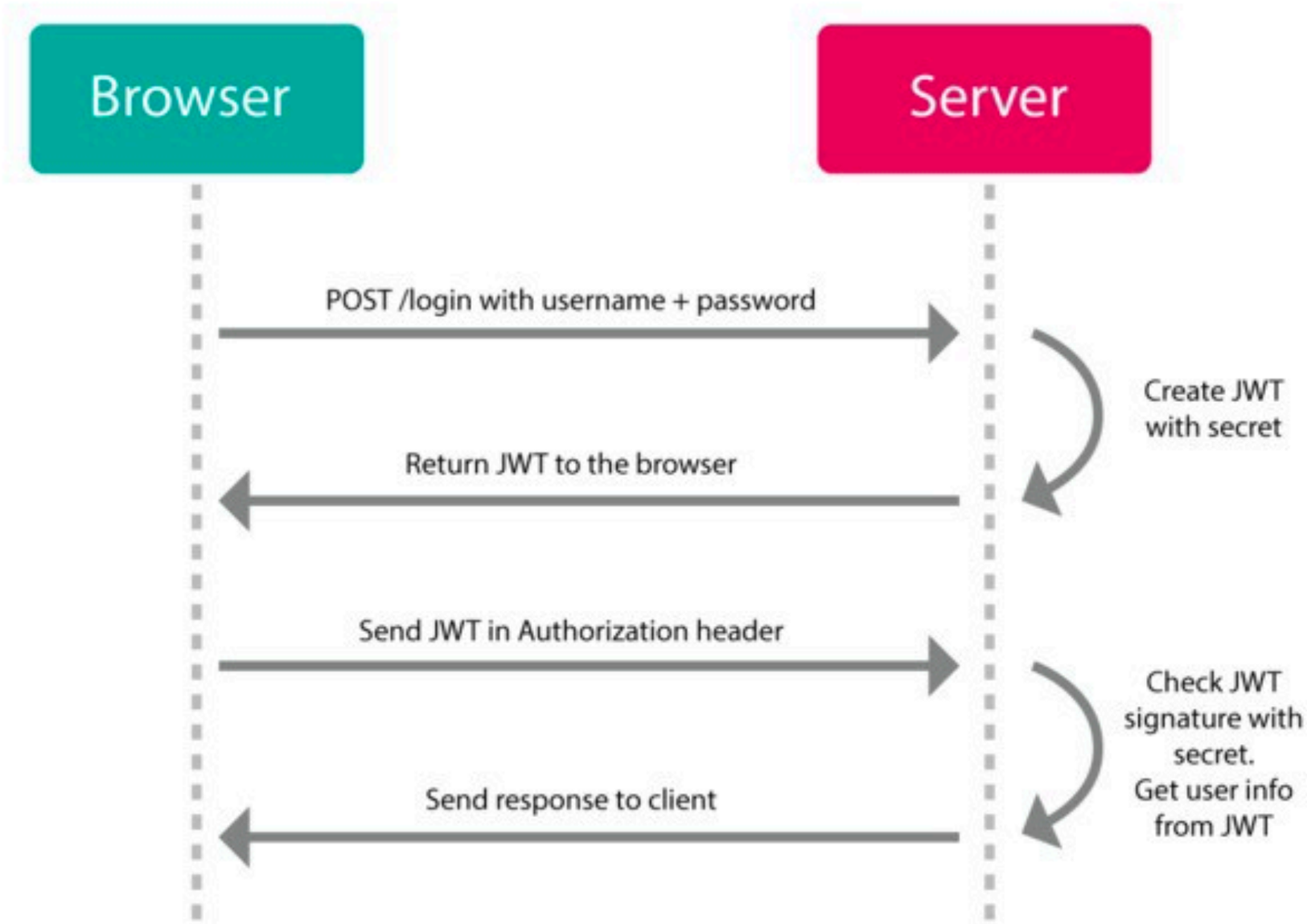
2 Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

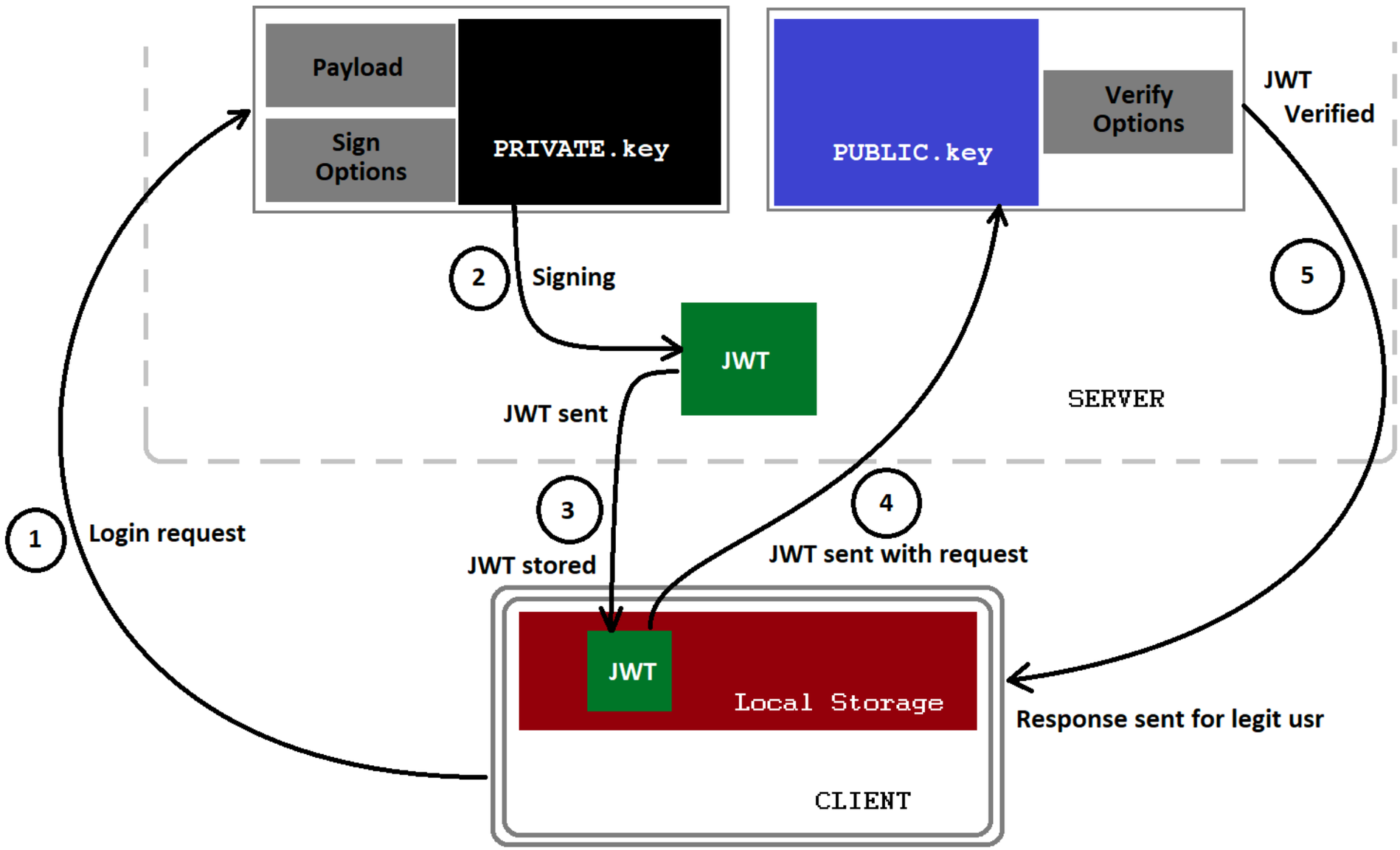
3 Signature

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

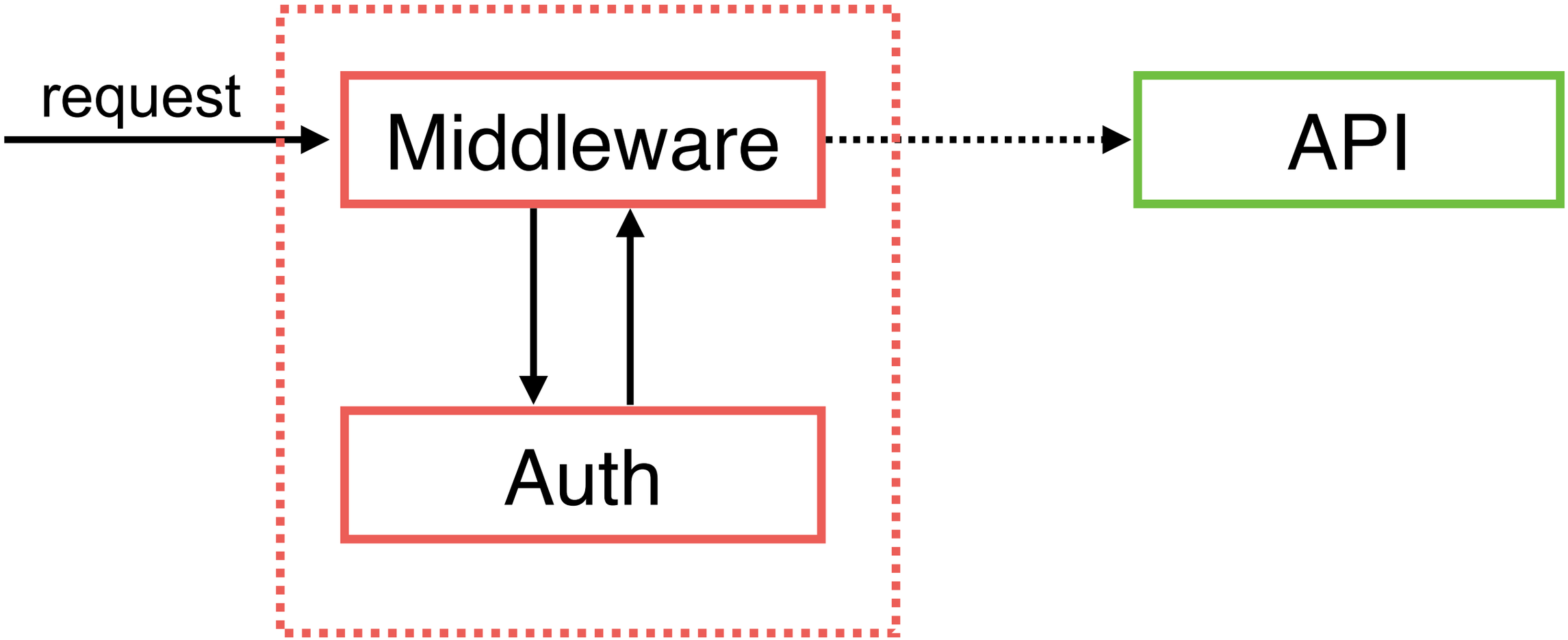
JWT Process



JWT Process (Better)



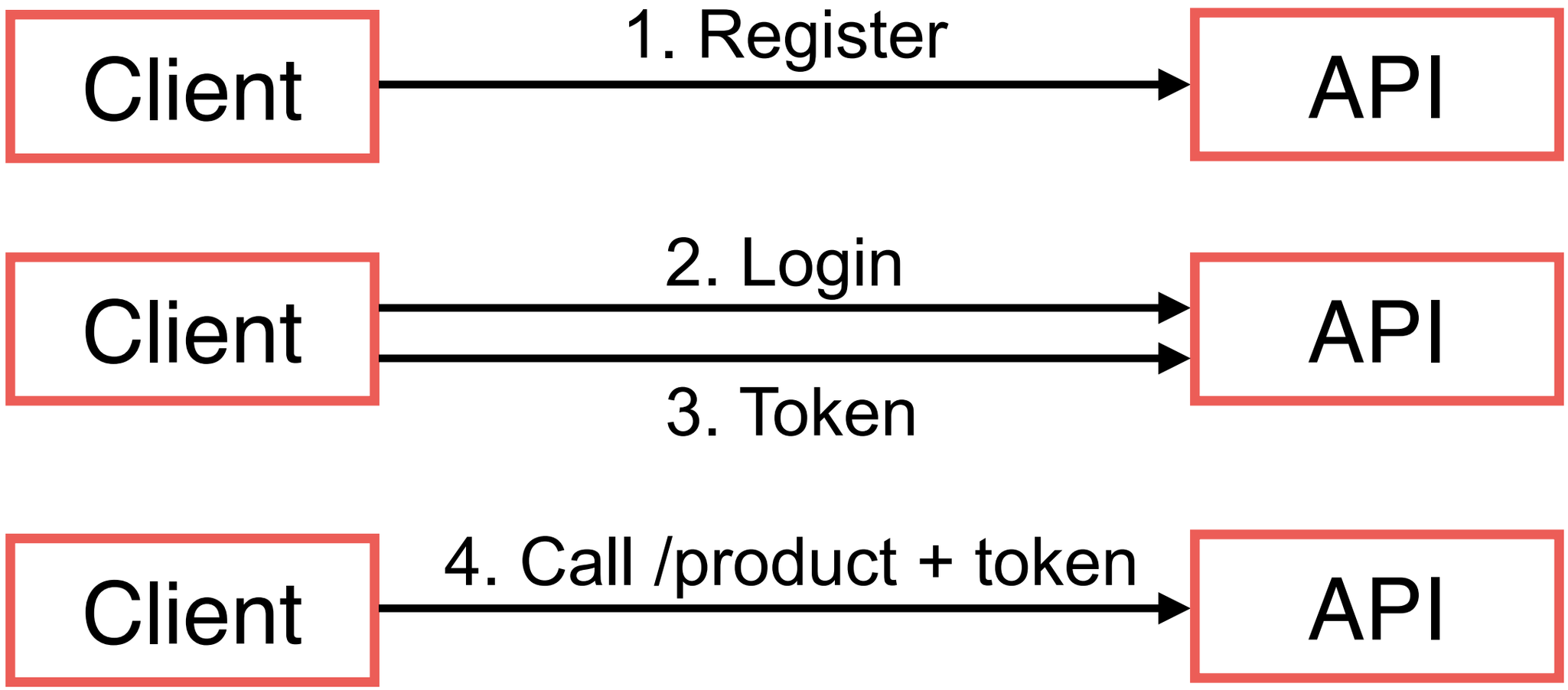
Using Middleware



Working with JWT

Working with JWT

Working with JWT



Auth with JWT

Using jsonwebtoken module

https://www.npmjs.com/package/jsonwebtoken

Auth with JWT

```
const jwt = require("jsonwebtoken");

const auth = async (req, res, next) => {
  try {
    const token = req.header("Authorization").replace("Bearer ", "");
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    // const id = decoded._id;

    const user = { id: 1, name: "fake user" };

    if (!user) {
      throw new Error();
    }

    req.token = token;
    req.user = user;
    next();

  } catch (e) {
    res.status(401).send({ error: "Please authenticate." });
  }
};
```

Check data in HTTP Request

```
const jwt = require("jsonwebtoken");

const auth = async (req, res, next) => {
  try {
    const token = req.header("Authorization").replace("Bearer ", "");
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    // const id = decoded._id;

    const user = { id: 1, name: "fake user" };

    if (!user) {
      throw new Error();
    }

    req.token = token;
    req.user = user;
    next();

  } catch (e) {
    res.status(401).send({ error: "Please authenticate." });
  }
};
```

Check user in database

```
const jwt = require("jsonwebtoken");

const auth = async (req, res, next) => {
  try {
    const token = req.header("Authorization").replace("Bearer ", "");
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    // const id = decoded._id;
    const user = { id: 1, name: "fake user" };

    if (!user) {
      throw new Error();
    }

    req.token = token;
    req.user = user;
    next();

  } catch (e) {
    res.status(401).send({ error: "Please authenticate." });
  }
};
```

1. Login to create token

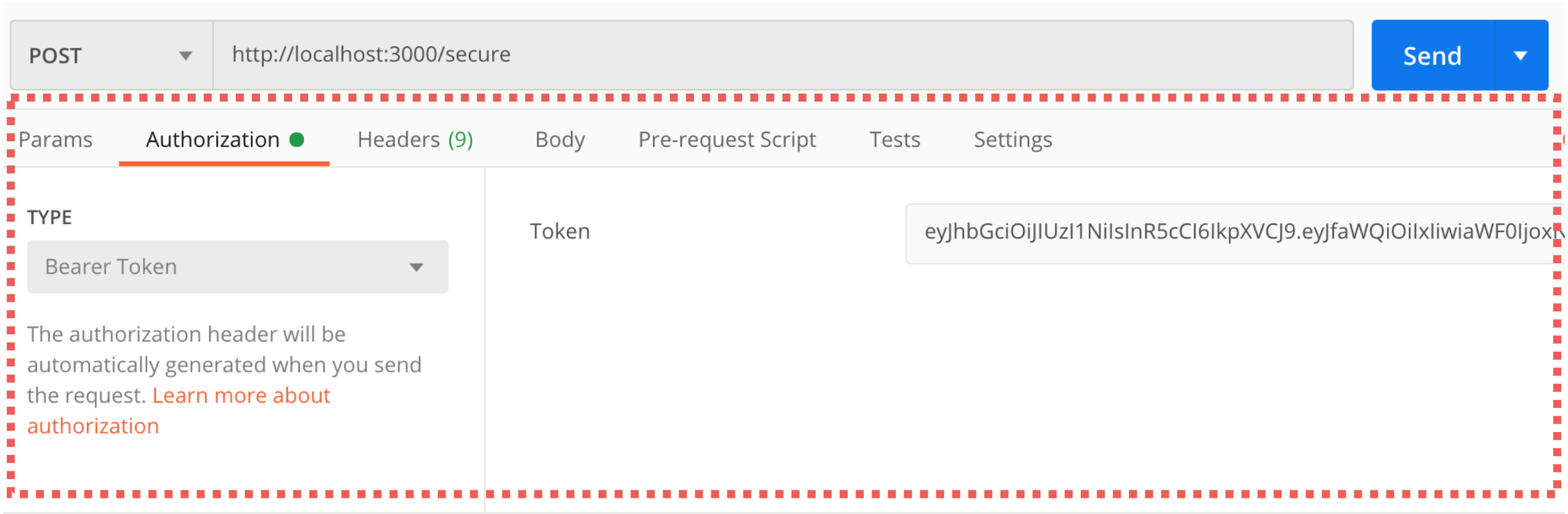
```
const express = require("express");
const jwt = require("jsonwebtoken");
const authorization = require("./auth");

const app = express();

app.get("/login", (req, res) => {
  const token = jwt.sign({ _id: "1" }, process.env.JWT_SECRET);
  res.send(token);
});

app.get("/secure", authorization, (req, res) => res.send("Secure  ... "));
```

2. Use middleware in routing

```
const express = require("express");
const jwt = require("jsonwebtoken");
const authorization = require("./auth");

const app = express();

app.get("/login", (req, res) => {
  const token = jwt.sign({ _id: "1" }, process.env.JWT_SECRET);
  res.send(token);
});

app.get("/secure", authorization, (req, res) => res.send("Secure ... "));
```

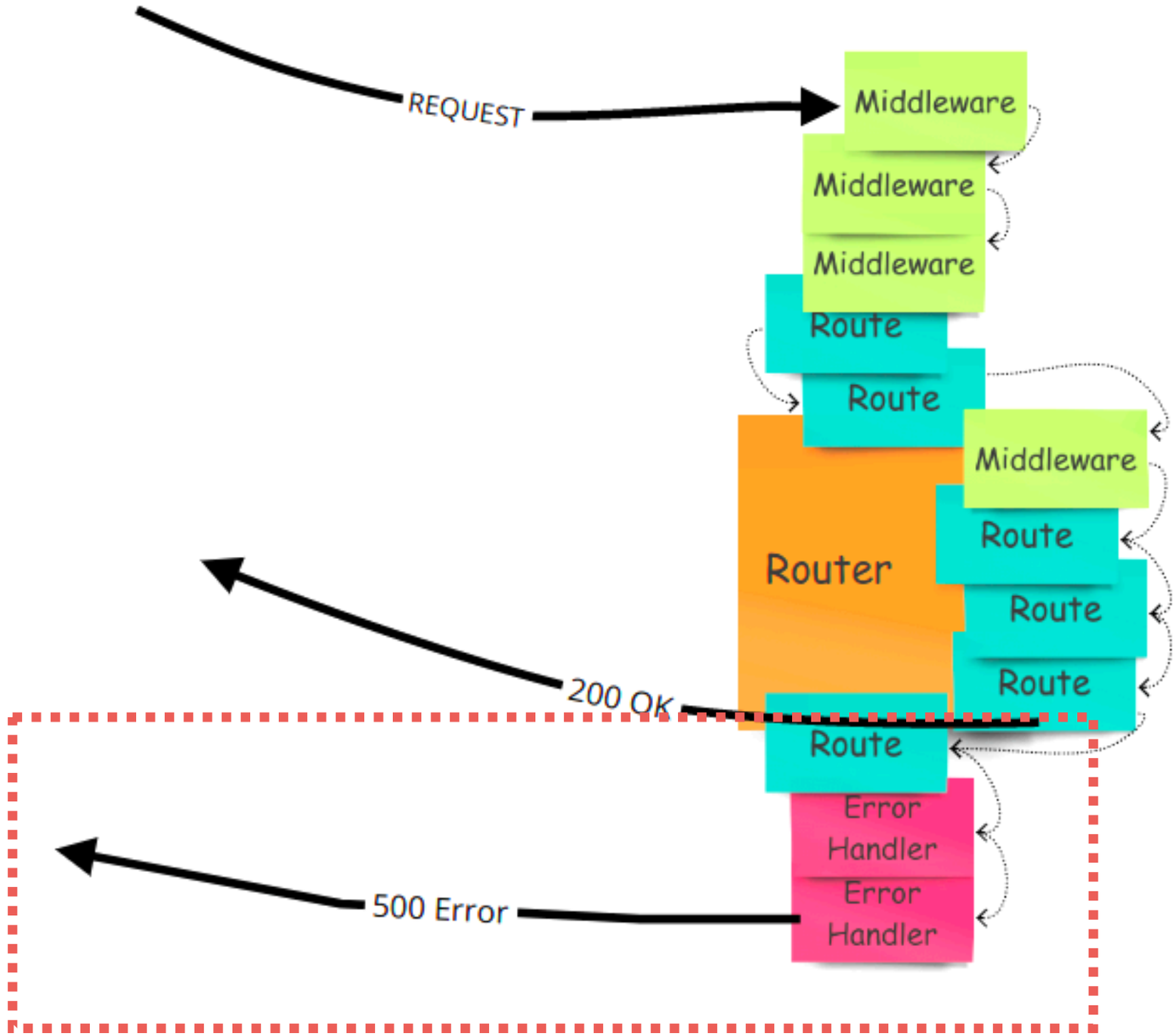
Using postman

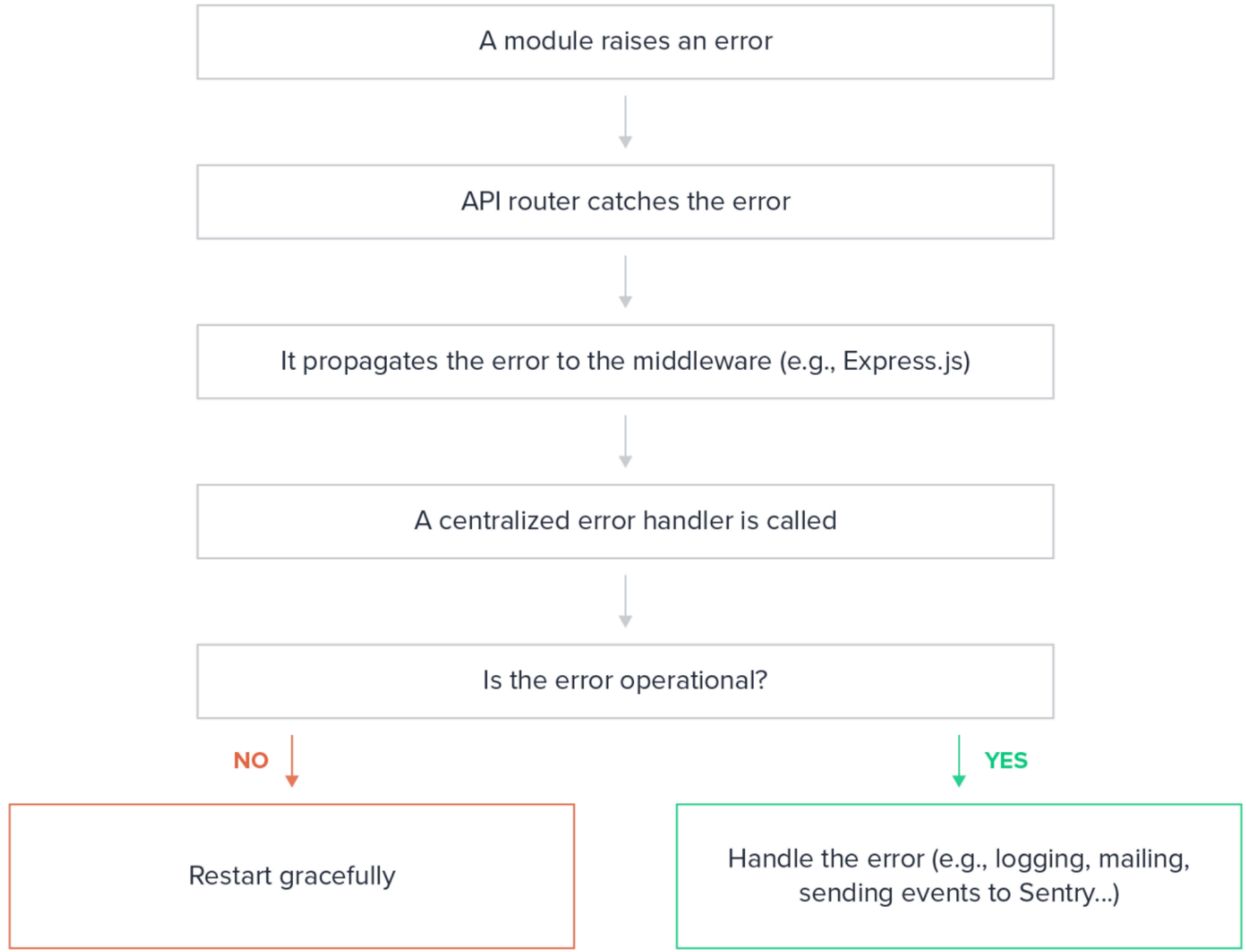
Config in Authorization :: Beearer Token



Error Handling with Express

Express middleware





Container with Node.JS

12 factors for Node.js

https://12factor.net/

Continuos Testing

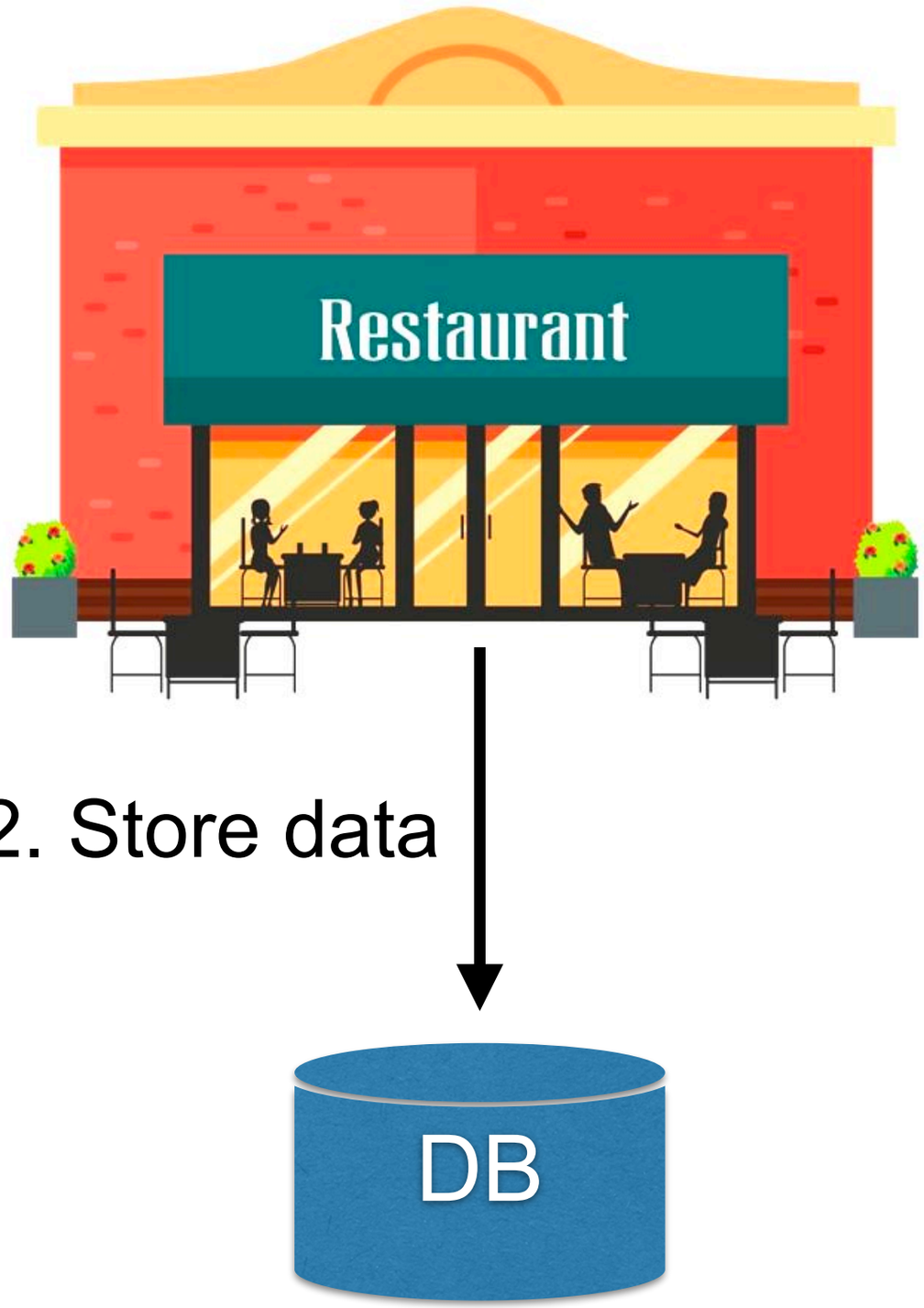
Group Workshop

Customer

1. Booking request

Customer

1. Booking request



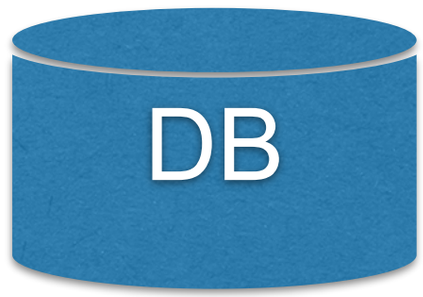
Customer

1. Booking request

2. Store data

Admin

3. Review and approve



Unit testing

Unit testing

Smallest testable part of application
Procedural (function, module)
Object-oriented (interface, class, method)

Unit testing

Tests an isolated unit via API
Performed in memory (no permanent changes)
Safe to run repeatedly
Fast execution

Unit testing Assertions

Validate correctness
Statement that a predicate is going to be TRUE
Throws error if FALSE

Include context about **what** wrong and **where**

Example of assertions

OK (true/false)
Equal (==)
deepEqual (== for all properties)
structEqual (===)
Throws error

Unit testing with Dependencies

Simulate dependencies (test double)
Isolate behaviour of a tested unit
Unit test your custom code

Unit testing with Dependencies

Simulate dependencies (test double)

Isolate behaviour of a tested unit

Unit test your custom code

Not third-party code

Not core code

Integration testing

Integration testing

Builds in unit tests
Combines and tests resulting combinations
eg. APIs, UIs and results

Integration testing

Test on one system to cross-systems
Uses full or partial environment
eg. Databases and services

Integration testing

More complex and harder to maintain
BUT more confidence than unit test

Example

Two units: booking API and client

Test booking creation

Client calls booking API
API return response
Validation response
Validation creation

Functional testing

Functional testing

Focus in on result, not code
eg. User interface

Functional testing

Check a specific feature
Compare results against specification
User workflow

Functional testing

Slower than unit and integration
Typically automated (sometime manual)

Example

Use visits web page

Click booking button

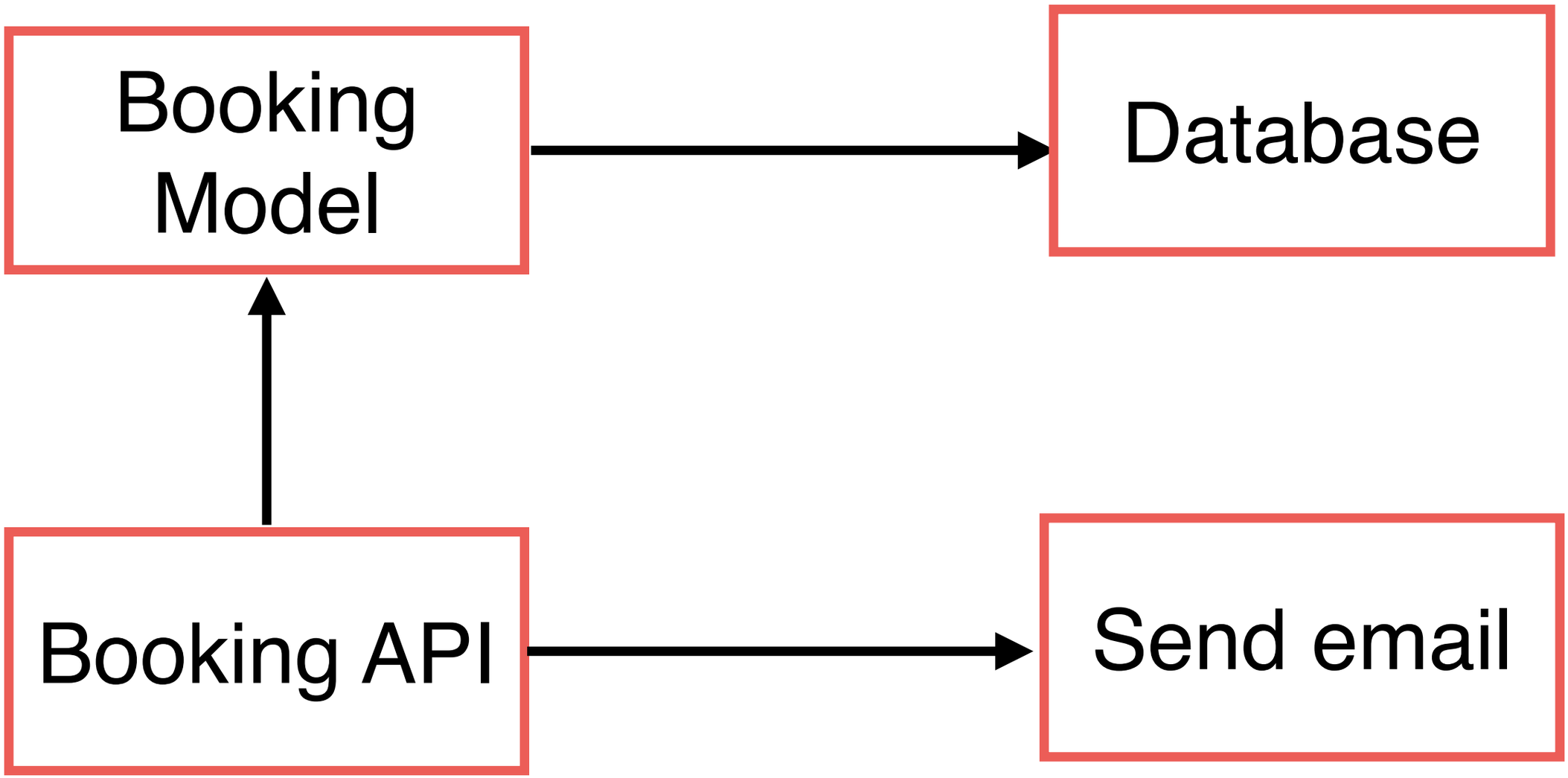
See booking table

Enter data in form

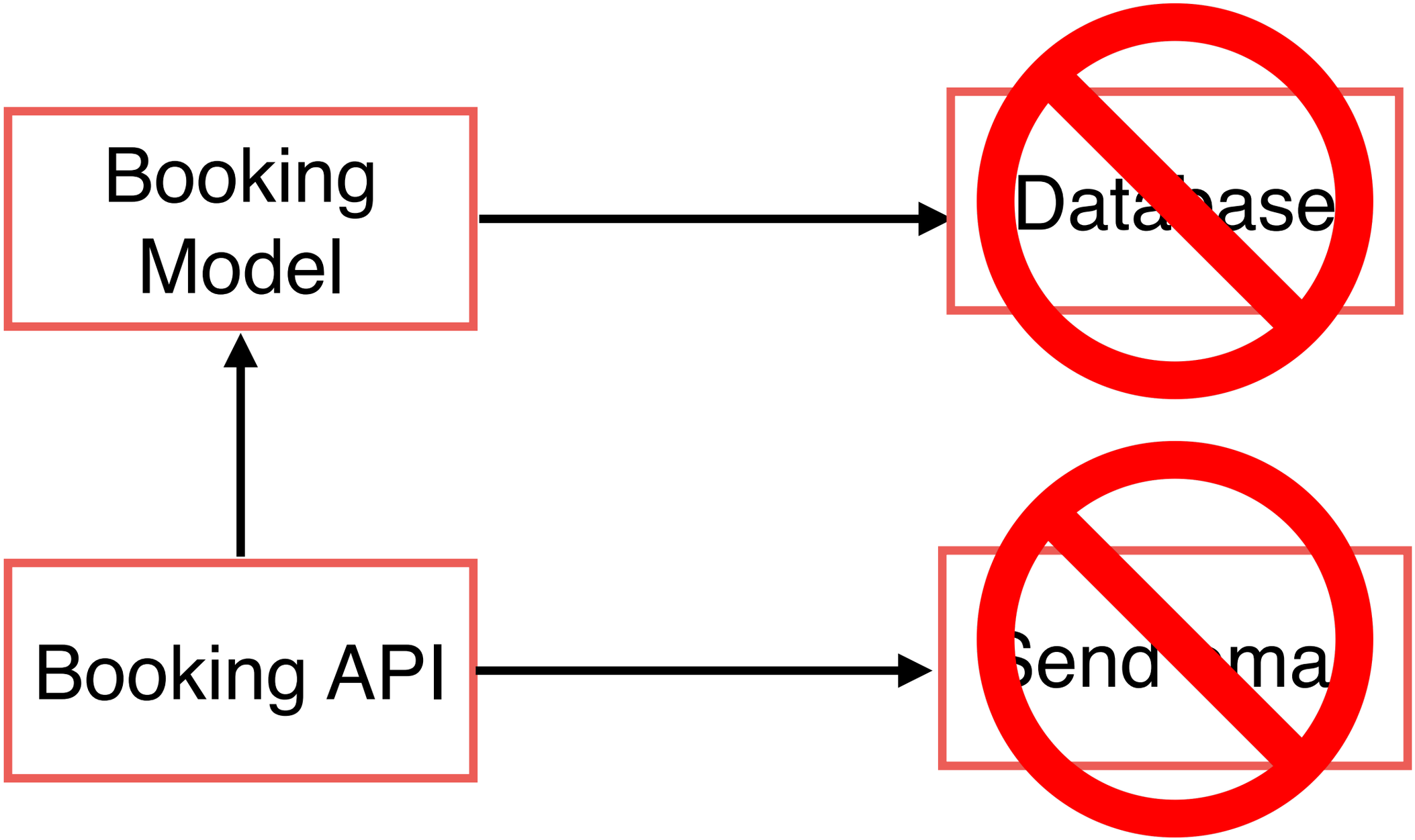
Submit booking request

See success result in booking result page

Testing relationship

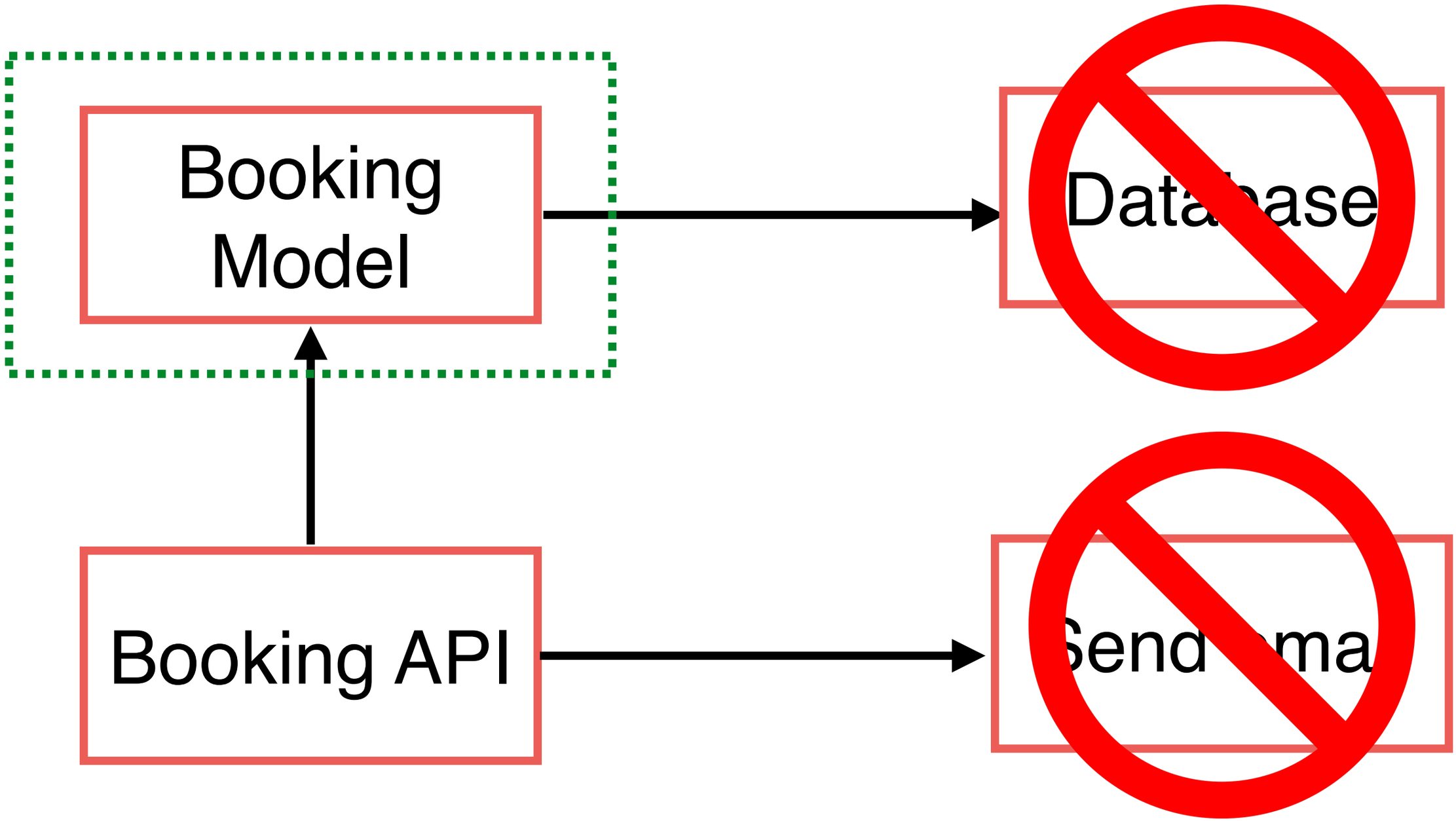


Testing relationship :: Unit test



What to tests ?

Unit test



Booking Model ?

Schema of data

Data type and formating

Validate data model

Booking Model ?

Schema of data

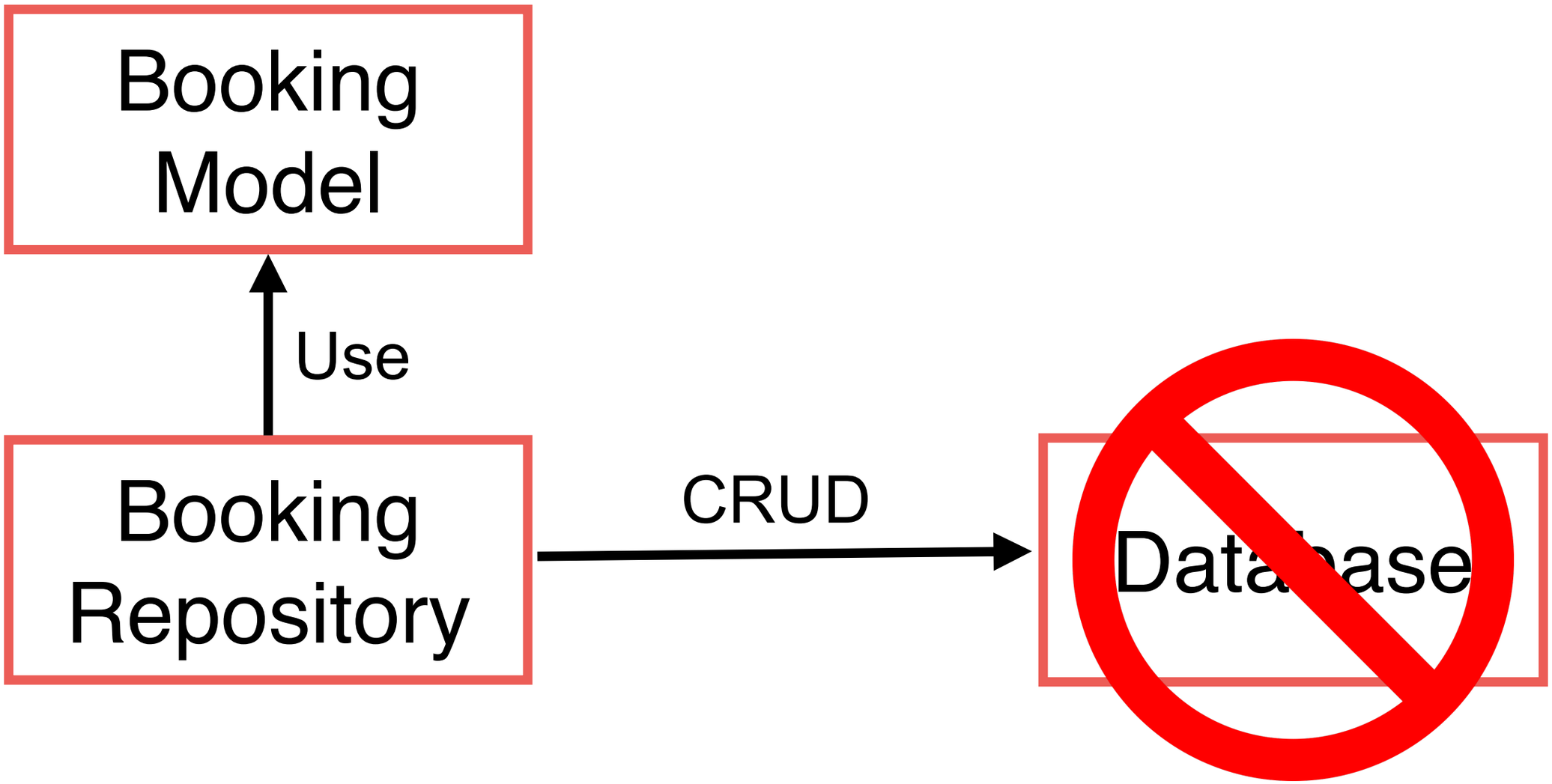
Data type and formatting

Validate data model

https://www.npmjs.com/package/joi

Working with Database

Using test double

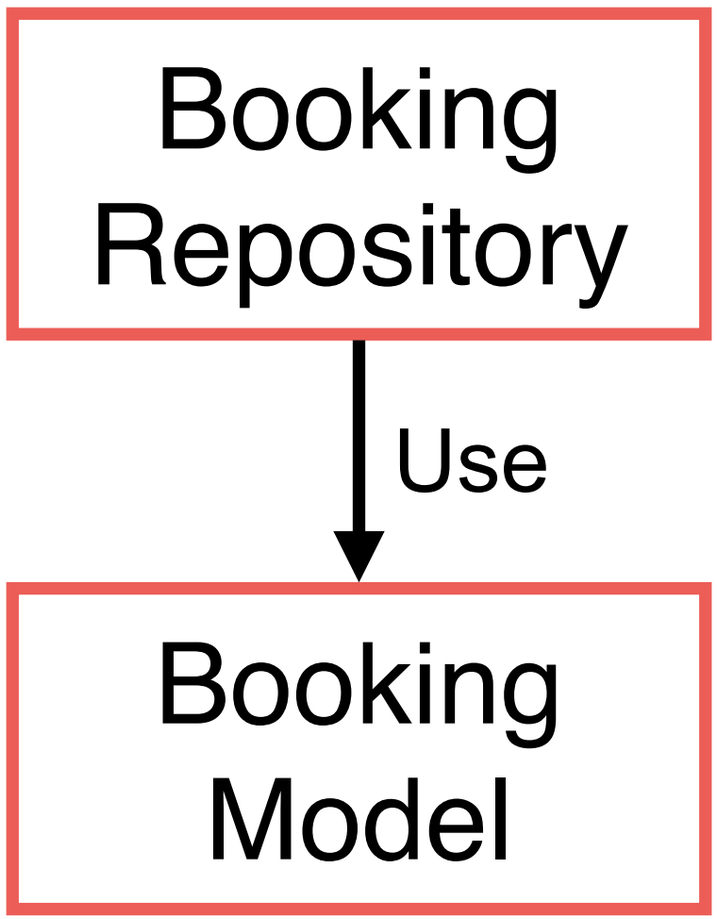


Booking Repository ?

CRUD with Database
Validate input

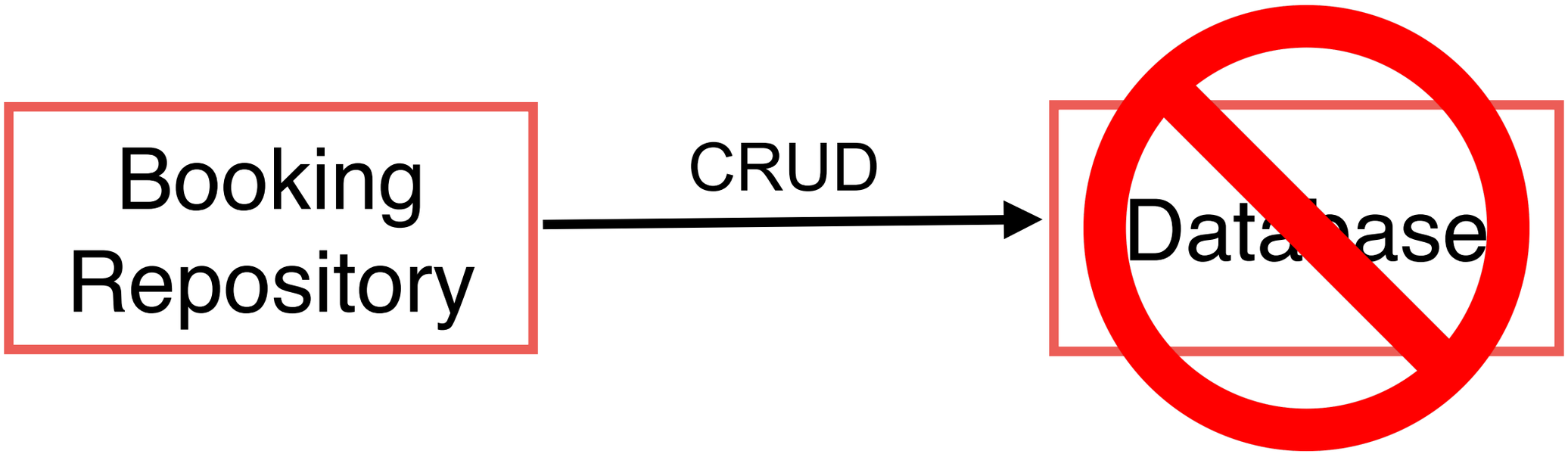
Booking
Repository

Validate input ?



Working with Database

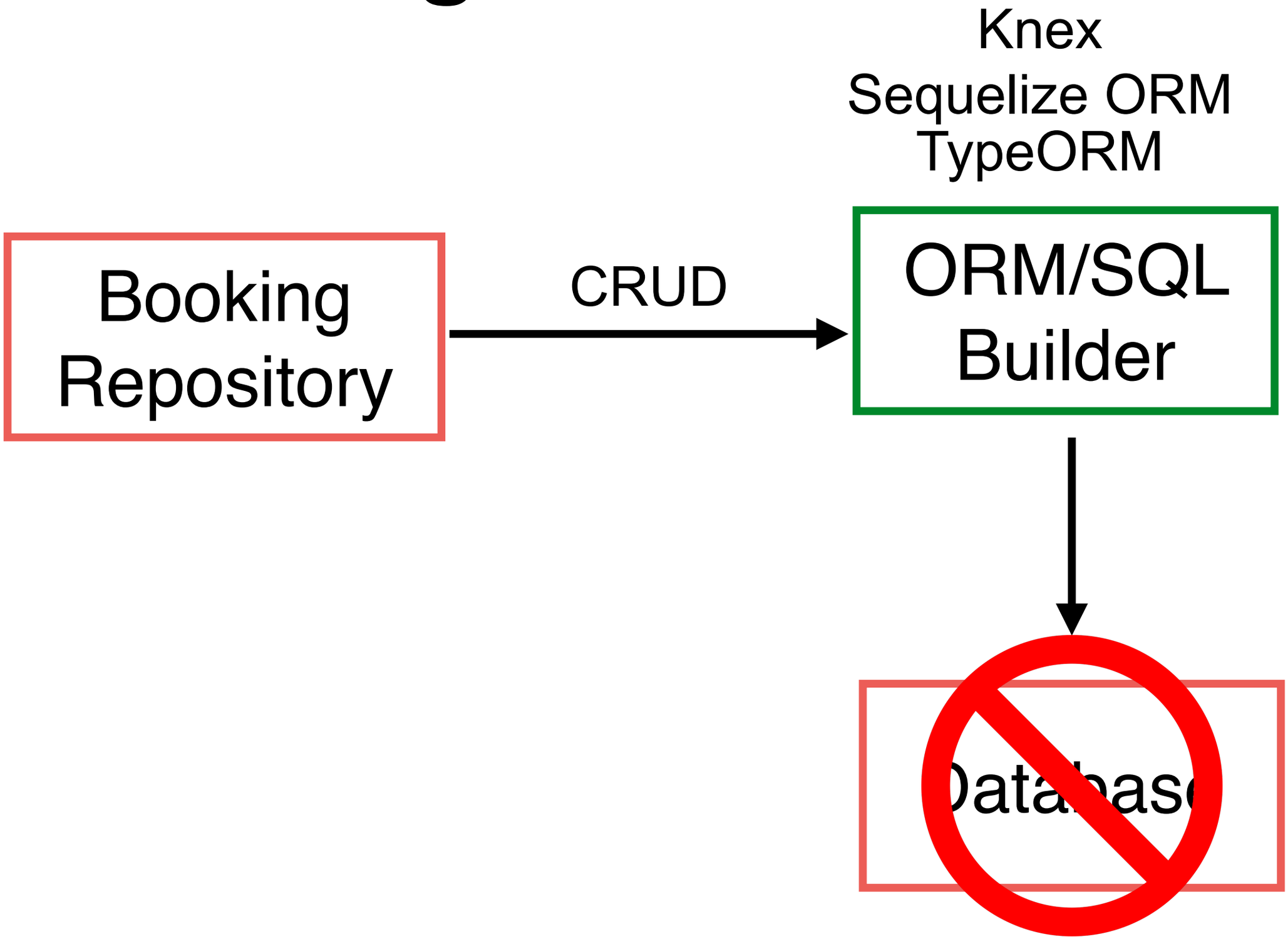
Using test double



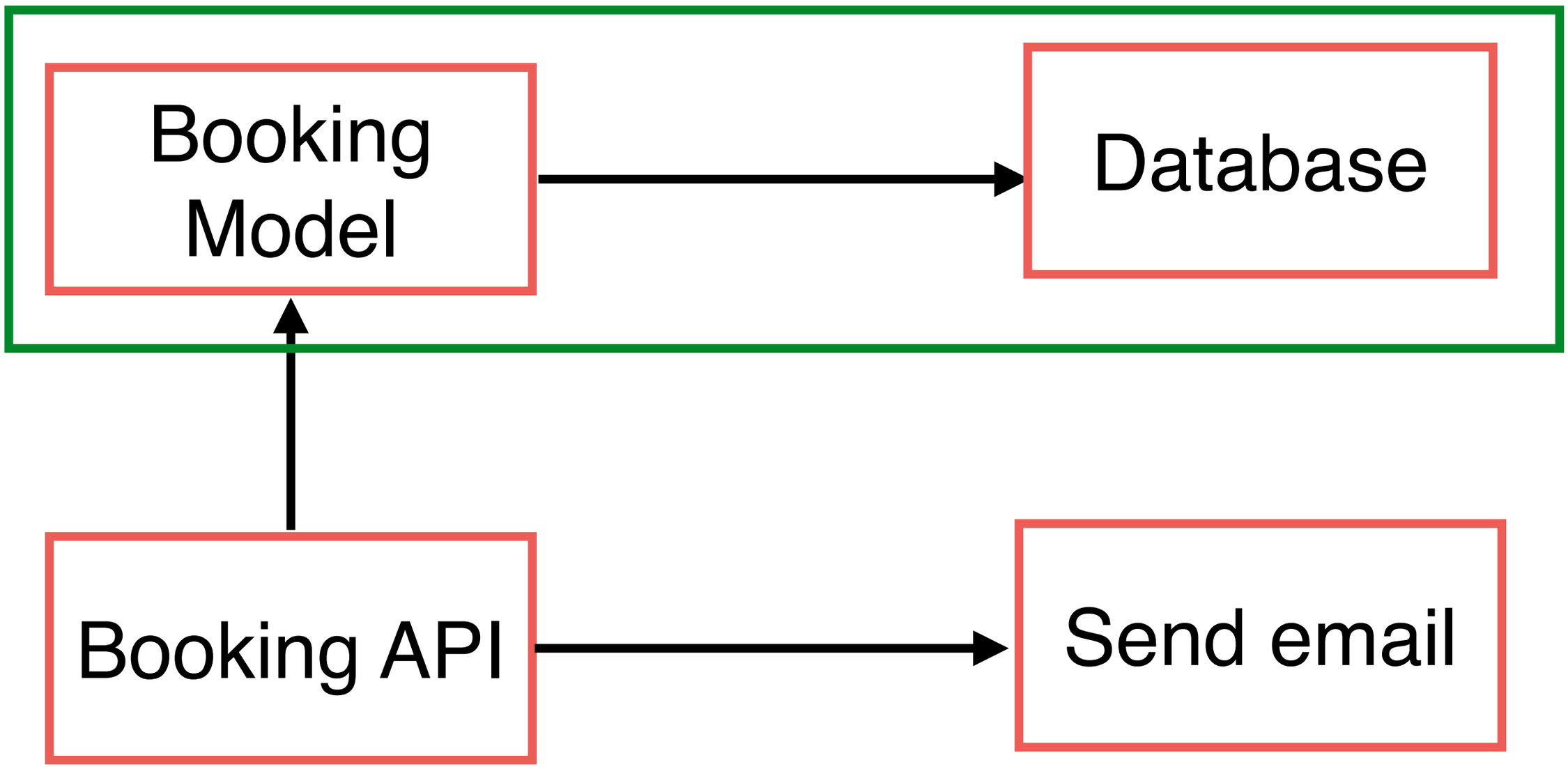
Test double with Node.JS

https://jestjs.io/

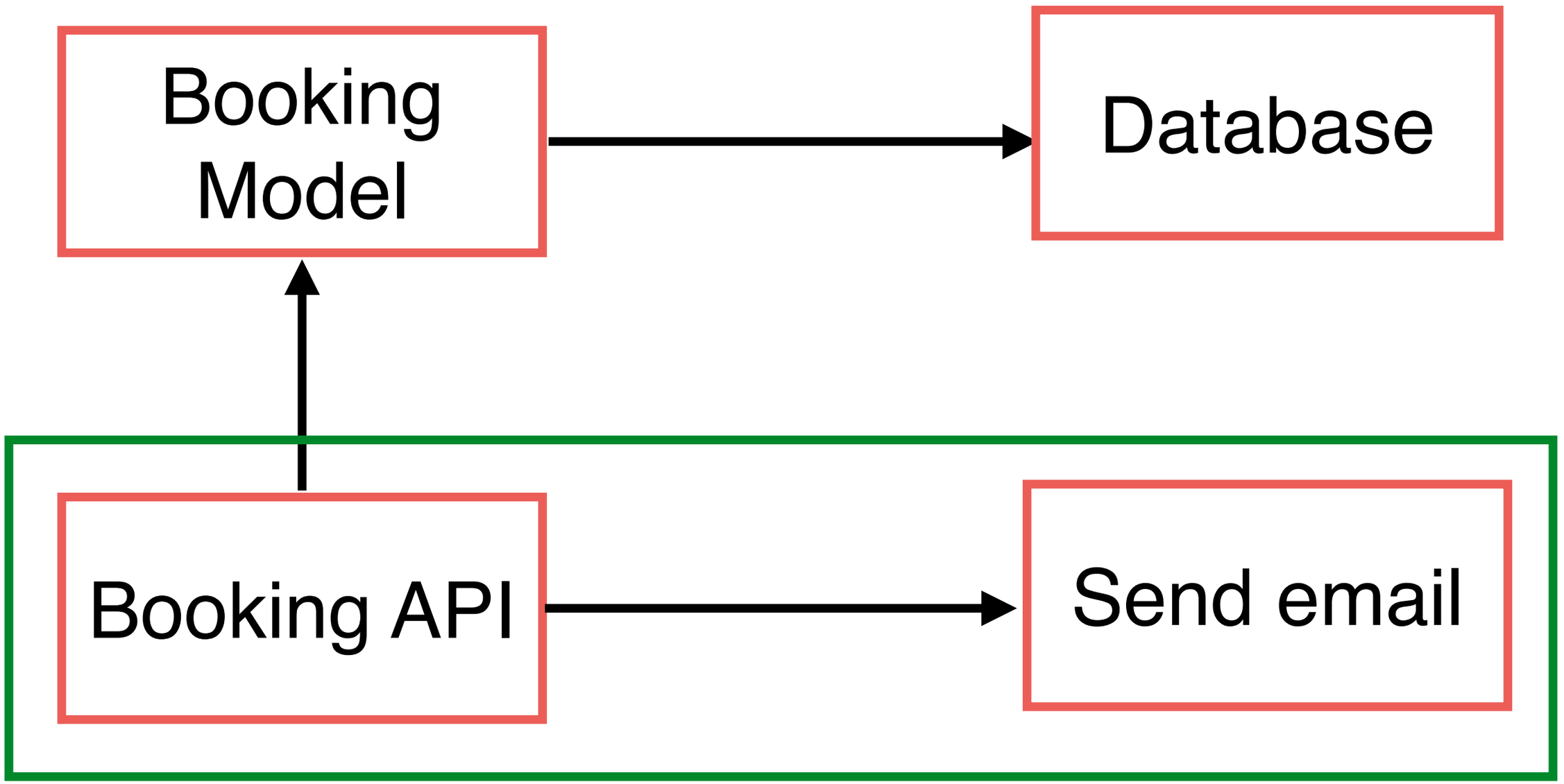
Working with Database



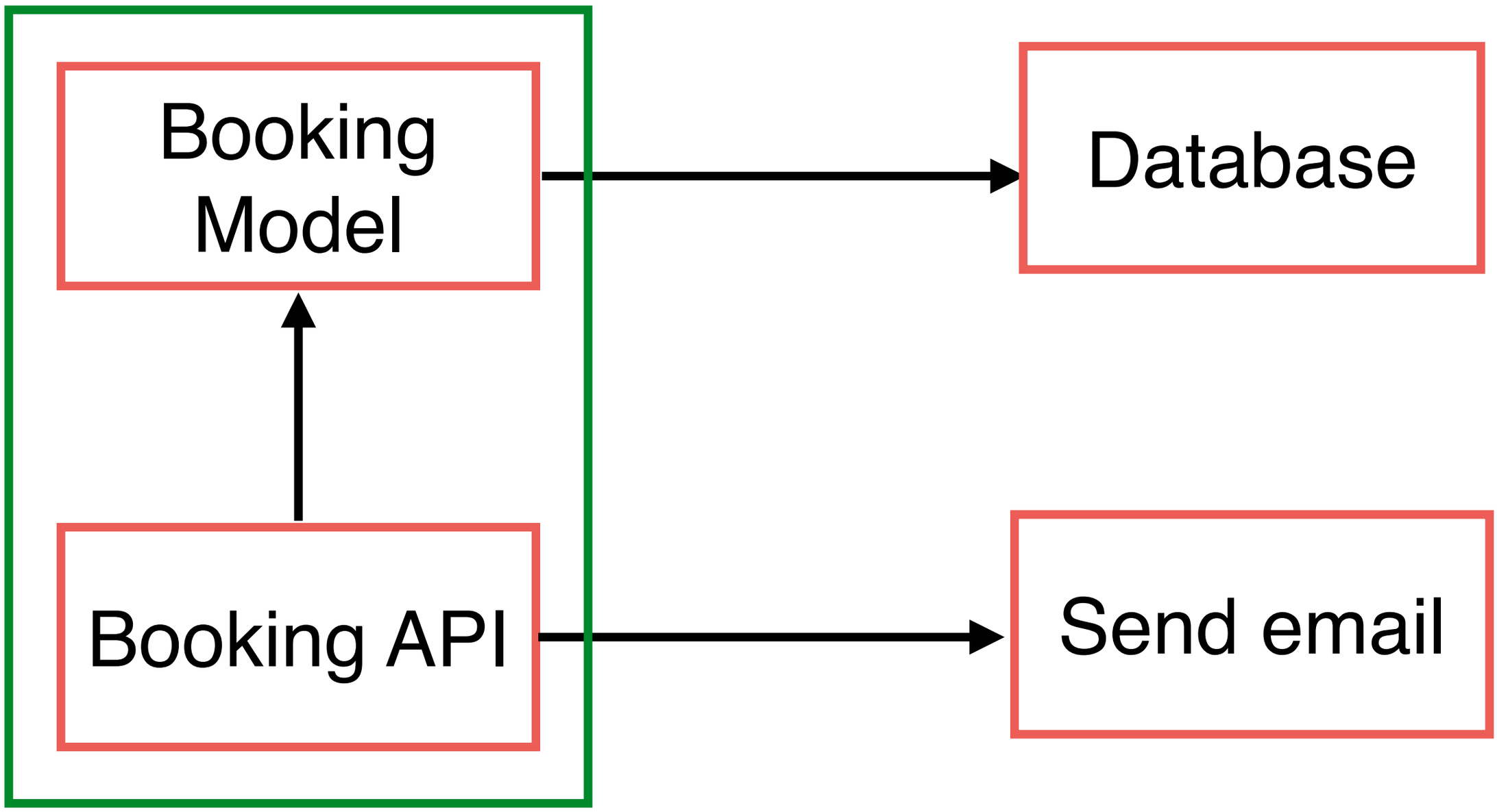
Testing relationship :: Integration test



Testing relationship :: Integration test



Testing relationship :: Integration test



What to tests ?