TESTING IN DJANGO

by Ana Balica

django

UNDER THE HOOD

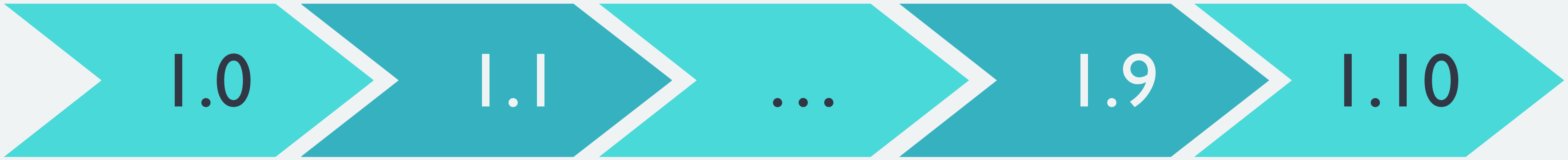
@anabalica

POTATO

DJANGO ARCHEOLOGY

1.0

DJANGO ARCHEOLOGY



#2333 Add unit test framework for end-user Django applications

#2333 Add unit test framework for end-user Django applications

” *As an added incentive, this is a feature that is present in Rails.*

```
./manage.py test
```

1.0

```
./manage.py test
    app.TestClass.test_method
```

1.0

TEST RUNNER

setup test environment

tests.py	models.py

teardown & results

1.0

CLIENT

get

post

login

logout

1.0

assert*

Redirects

(Not)Contains

FormError

Template(Not)Used

TESTCASE

1.0

1.1

CLIENT

put

head

delete

options

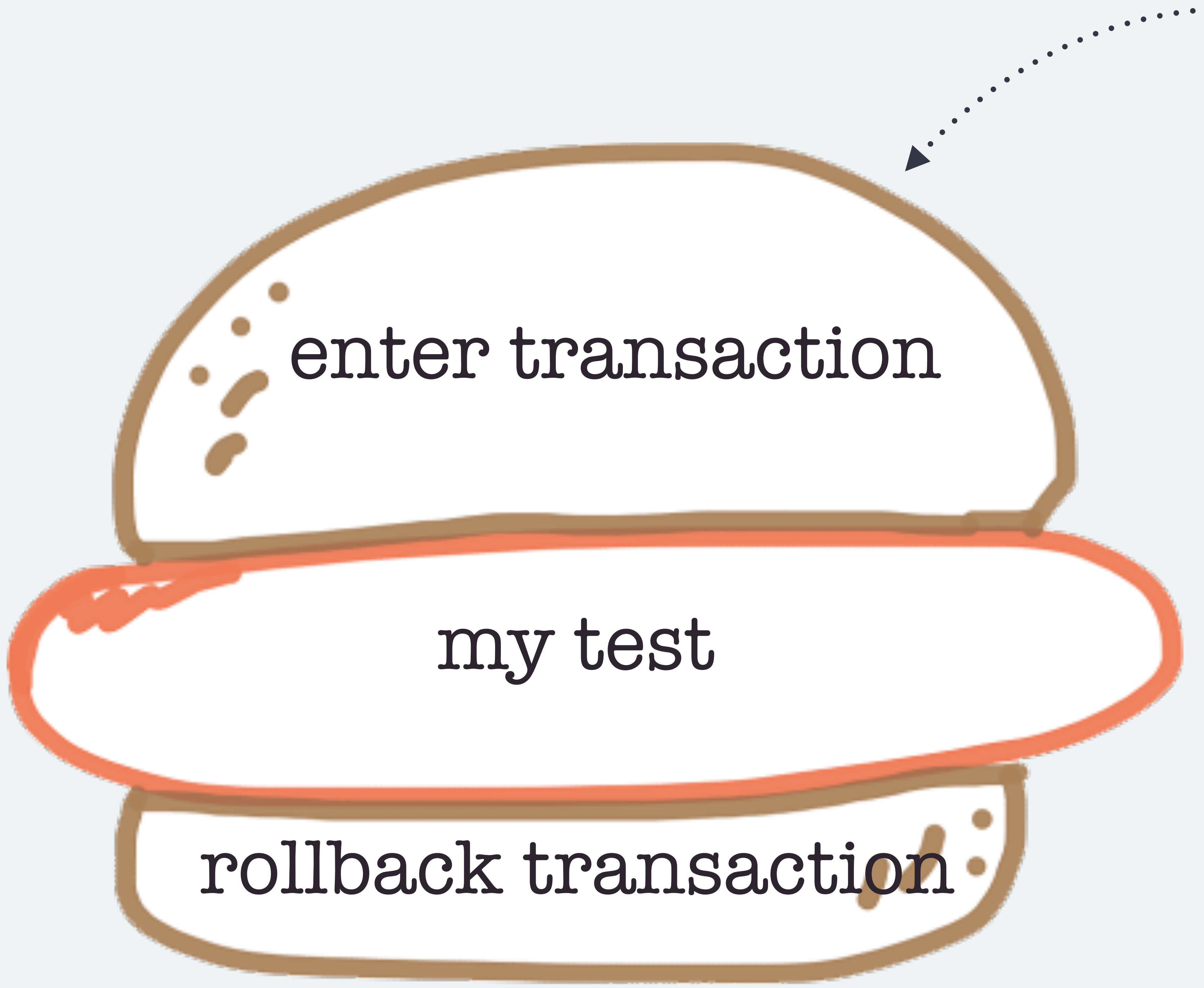
1.1

TransactionTestCase

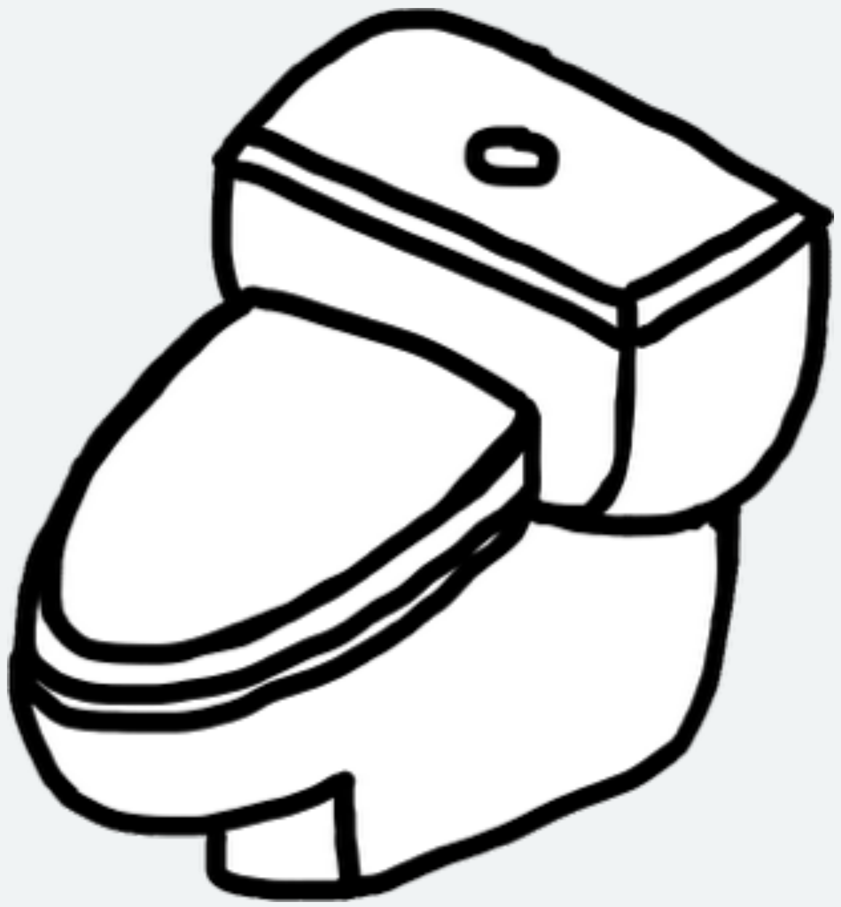
TestCase

1.1

I am a TestCase burger



I am a TransactionTestCase.
I flush the database before each test.



1.2

DjangoTestSuiteRunner

from function to class

```python
failures = test_runner(test_labels, verbosity, interactive)
if failures:
    sys.exit(failures)
```

```
failures = test_runner(test_labels, verbosity, interactive)
if failures:
    sys.exit(failures)
```

1.2

```
failures = test_runner(test_labels, verbosity, interactive)
if failures:
    sys.exit(failures)
```

0

success

1.2

```
failures = test_runner(test_labels, verbosity, interactive)
if failures:
    sys.exit(failures)
```

1

👍 failure 👍

1.2

```
failures = test_runner(test_labels, verbosity, interactive)
if failures:
    sys.exit(failures)
```

42

failure

1.2

```
failures = test_runner(test_labels, verbosity, interactive)
if failures:
    sys.exit(failures)
```

256

👎 success 👎

1.2

```
failures = TestRunner(verbosity, interactive, failfast)
if failures:
    sys.exit(bool(failures))
```

1.2

```
failures = TestRunner(verbosity, interactive, failfast)
if failures:
    sys.exit(bool(failures))
```

1.2

0 / 1

success failure

```
failures = TestRunner(verbosity, interactive, failfast)
if failures:
    sys.exit(1)
```

256

failure

1.2

```
failures = TestRunner(verbosity, interactive, failfast)
if failures:
    sys.exit(1)
```

256

failure

1.2

```
failures = TestRunner(verbosity, interactive, failfast)
if failures:
    sys.exit(1)
```

256

failure

1.2

```
failures = TestRunner(verbosity, interactive, failfast)
if failures:
    sys.exit(1)
```

256

failure

1.2

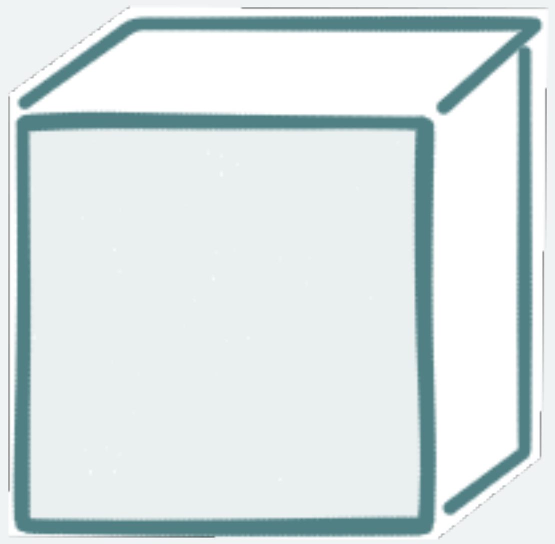
```
failures = TestRunner(verbosity, interactive, failfast)
if failures:
    sys.exit(1)
```

256

failure

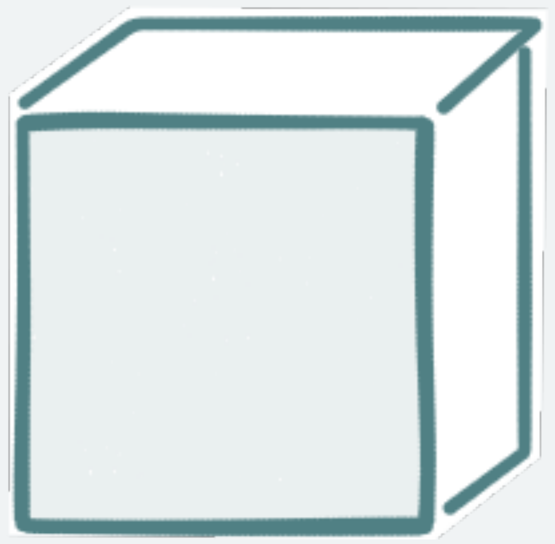
1.2

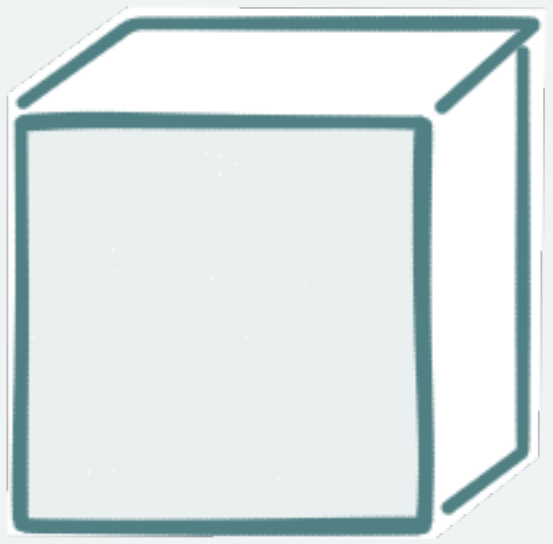
MULTIPLE DATABASES



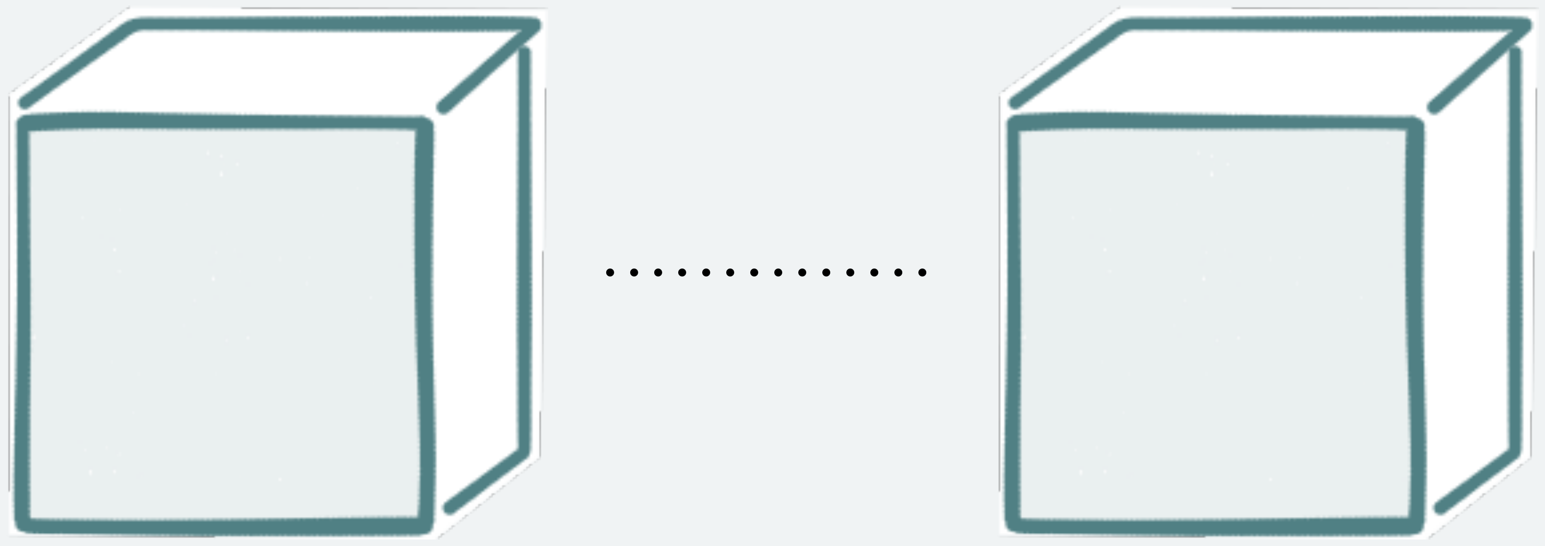
1.2

MULTIPLE DATABASES





MULTIPLE DATABASES



MULTIPLE DATABASES

```
DATABASES = {
    'default': {
        'HOST': 'dbprimary',
        # ... plus other settings
    },
    'replica': {
        'HOST': 'dbreplica',
        'TEST_MIRROR': 'default',
        # ... plus other settings
    }
}
```

1.2

MULTIPLE DATABASES

```
DATABASES = {
    'default': {
        'HOST': 'dbprimary',
        # ... plus other settings
    },
    'replica': {
        'HOST': 'dbreplica',
        'TEST_MIRROR': 'default',
        # ... plus other settings
    }
}
```

1.2

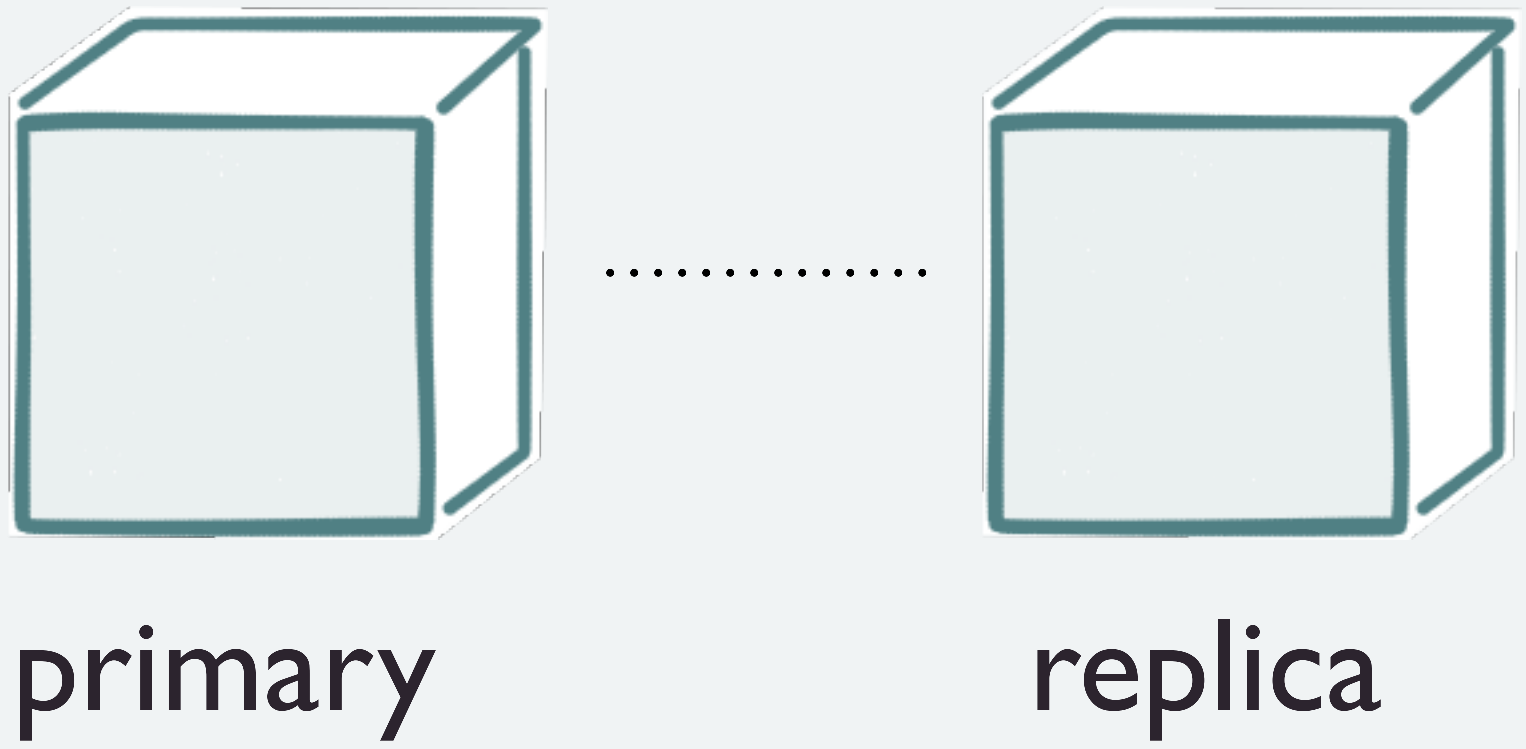
MULTIPLE DATABASES

```
DATABASES = {
    'default': {
        'HOST': 'dbprimary',
        # ... plus other settings
    },
    'replica': {
        'HOST': 'dbreplica',
        'TEST': {
            'MIRROR': 'default',
        },
        # ... plus other settings
    }
}
```

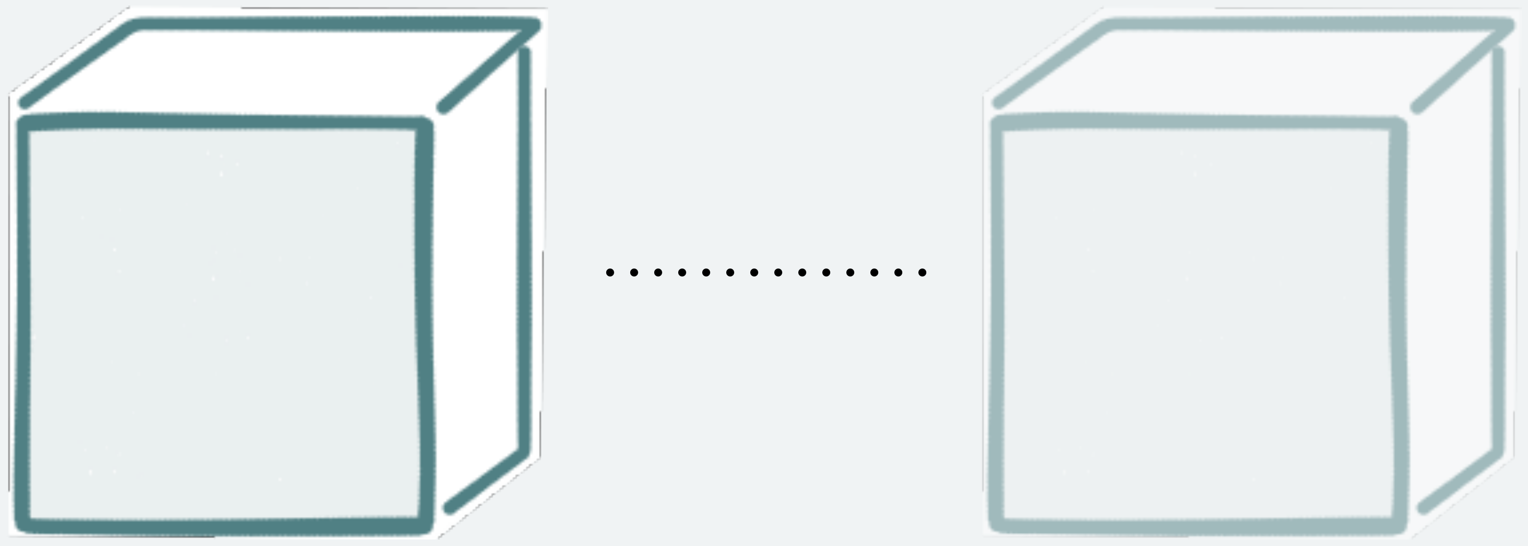
MULTIPLE DATABASES

```
DATABASES = {
    'default': {
        'HOST': 'dbprimary',
         # ... plus other settings
    },
    'replica': {
        'HOST': 'dbreplica',
        'TEST': {
            'MIRROR': 'default',
        },
        # ... plus other settings
    }
}
```

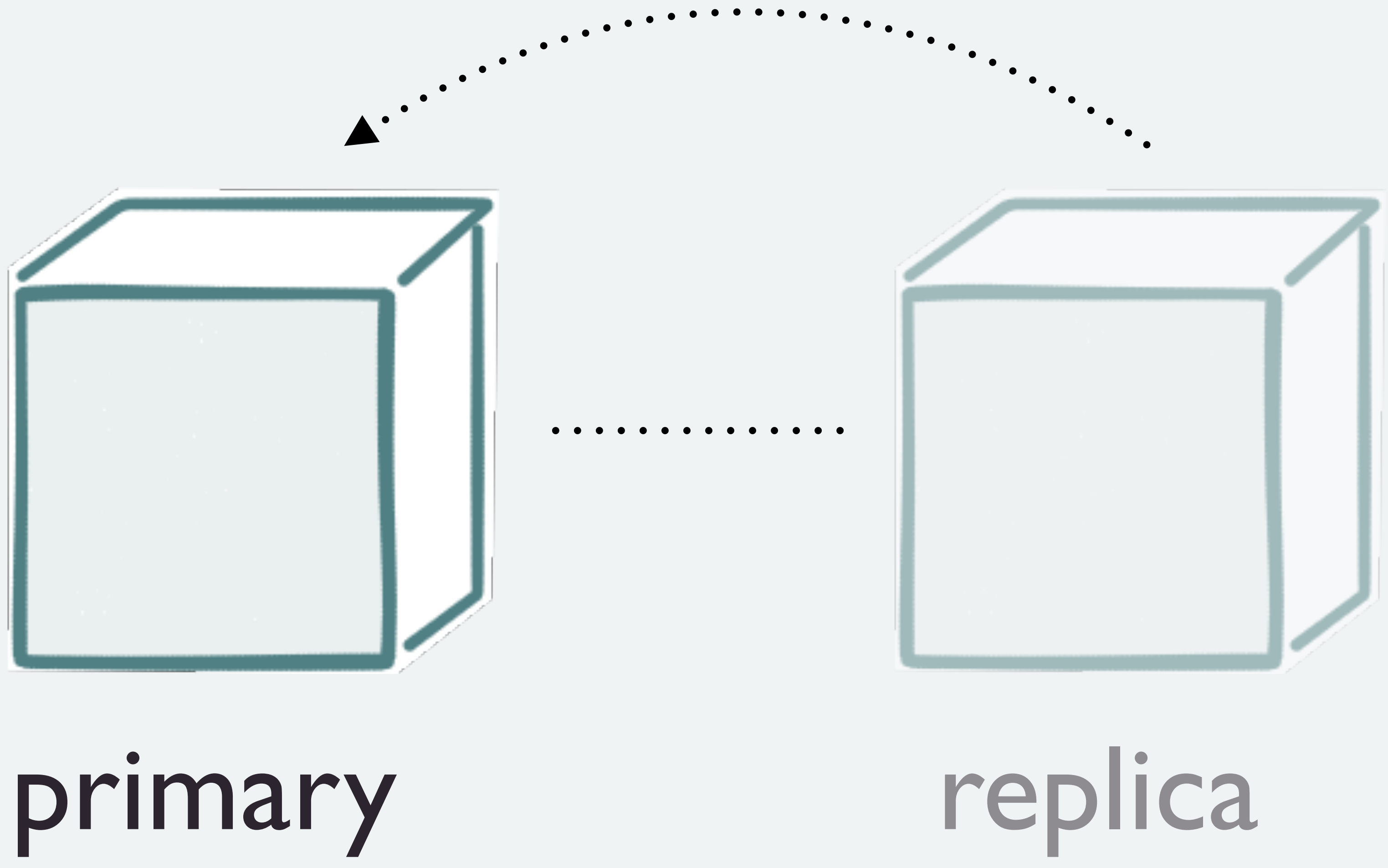
MULTIPLE DATABASES



MULTIPLE DATABASES



MULTIPLE DATABASES

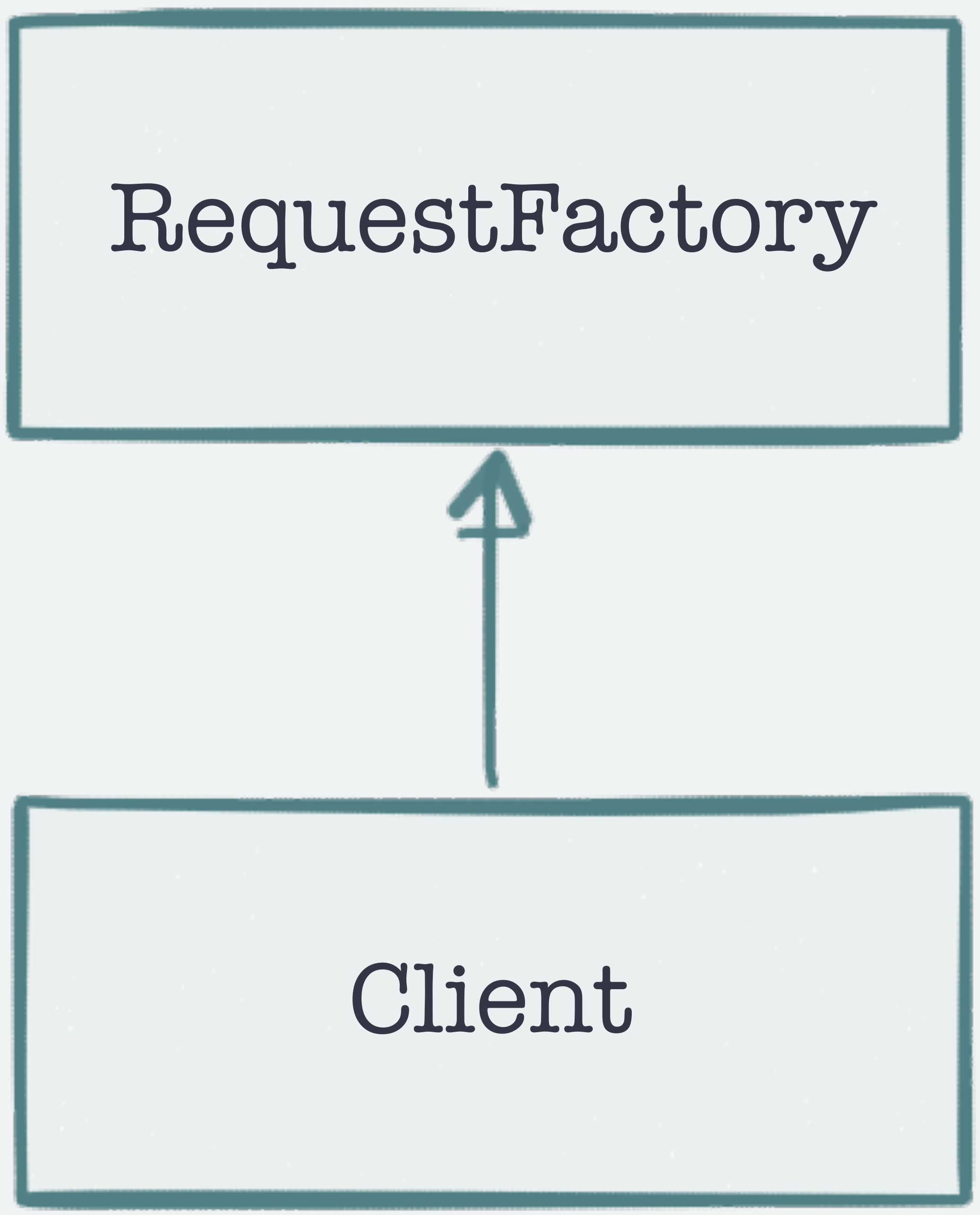


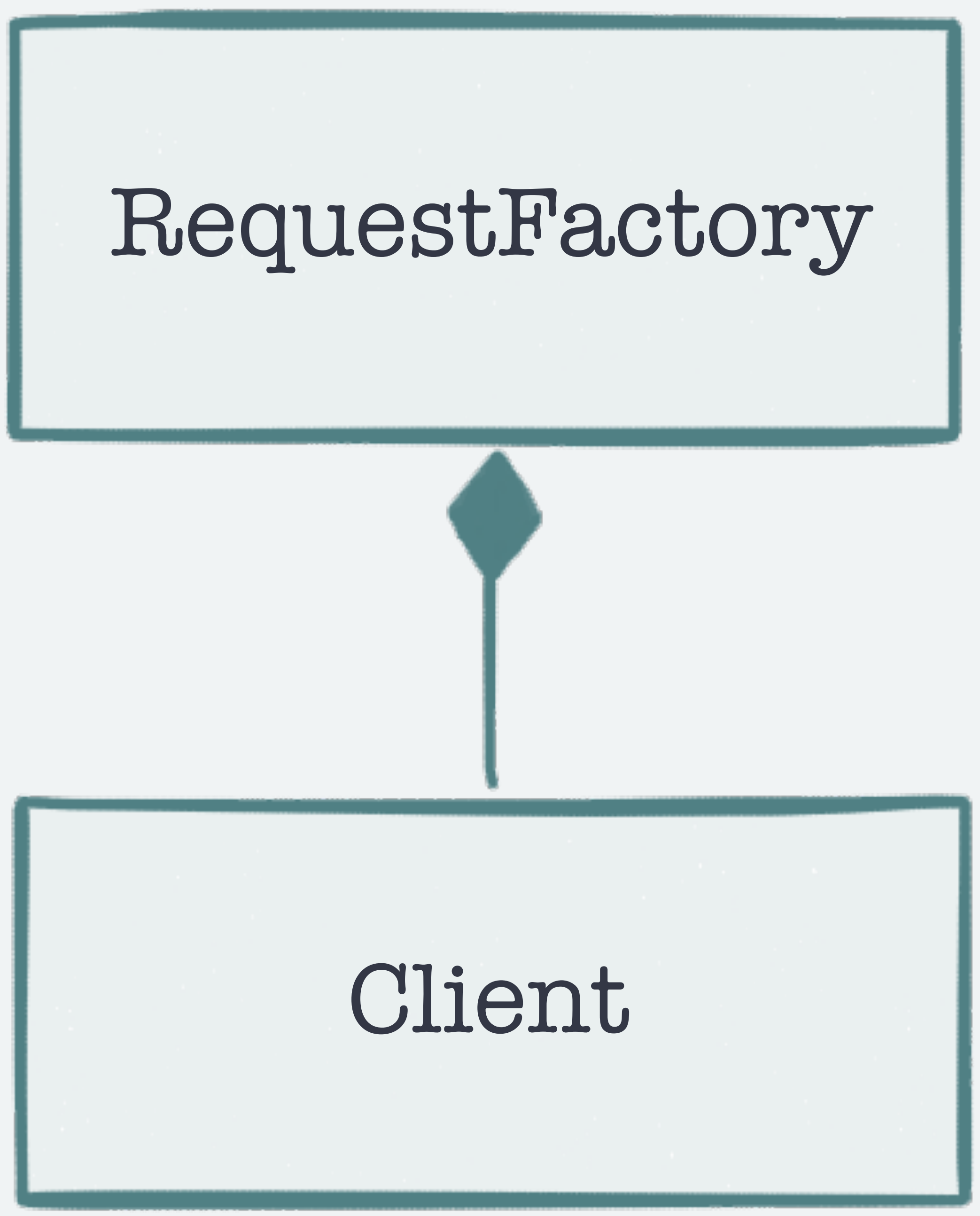
1.3

TESTCASE

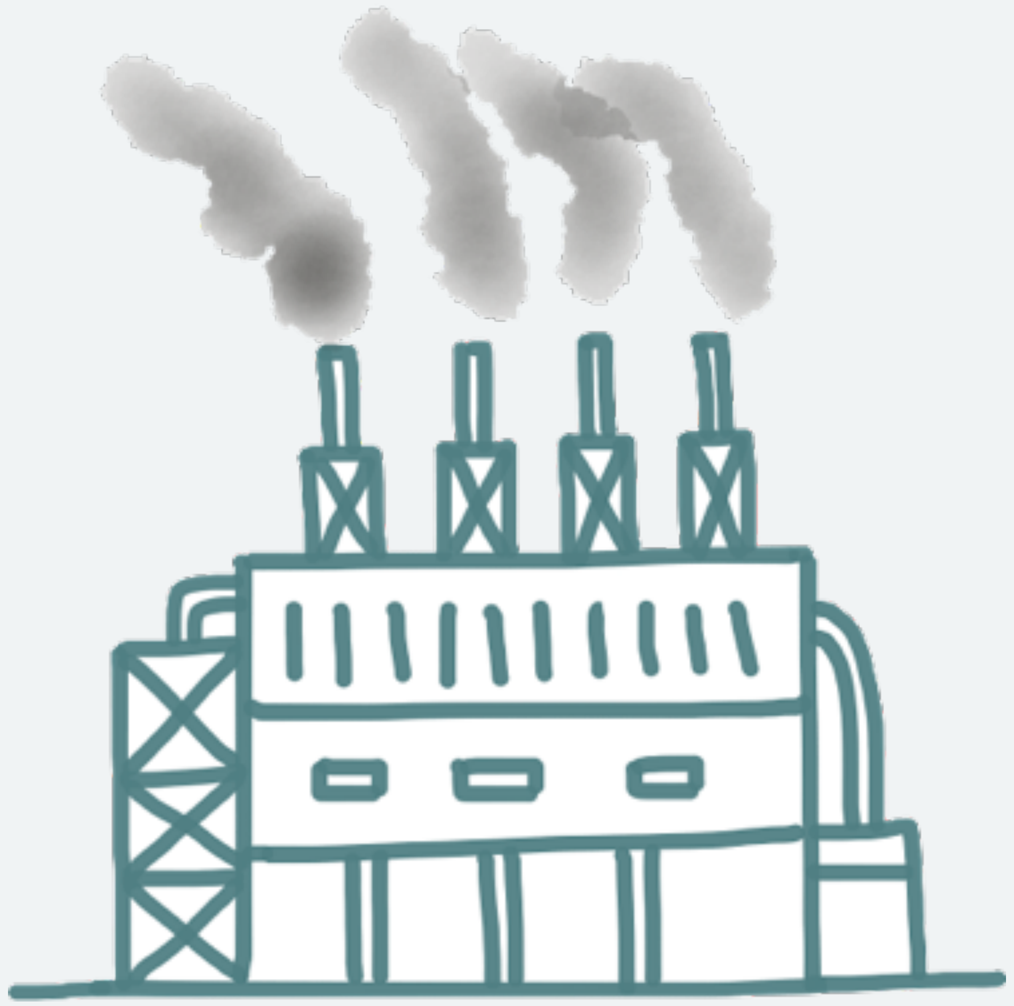
```
assert*
QuerysetEqual
NumQueries
```

1.3


RequestFactory
Client



1.3



```
class RequestFactory(object):
    def request(self, **request):
        return WSGIRequest(self._base_environ(**request))
```

1.3

doctests

=

tests + documentation

~~doctests~~

=

tests + documentation

```
@skipIfDBFeature(feature)
```

```
@skipUnlessDBFeature(feature)
```

1.3

1.4

TestCase

Transaction
TestCase

1.4

Simple TestCase	**LiveServer TestCase**
TestCase	Transaction TestCase

1.4

doesn't hit the database

Simple TestCase	**LiveServer TestCase**
TestCase	Transaction TestCase

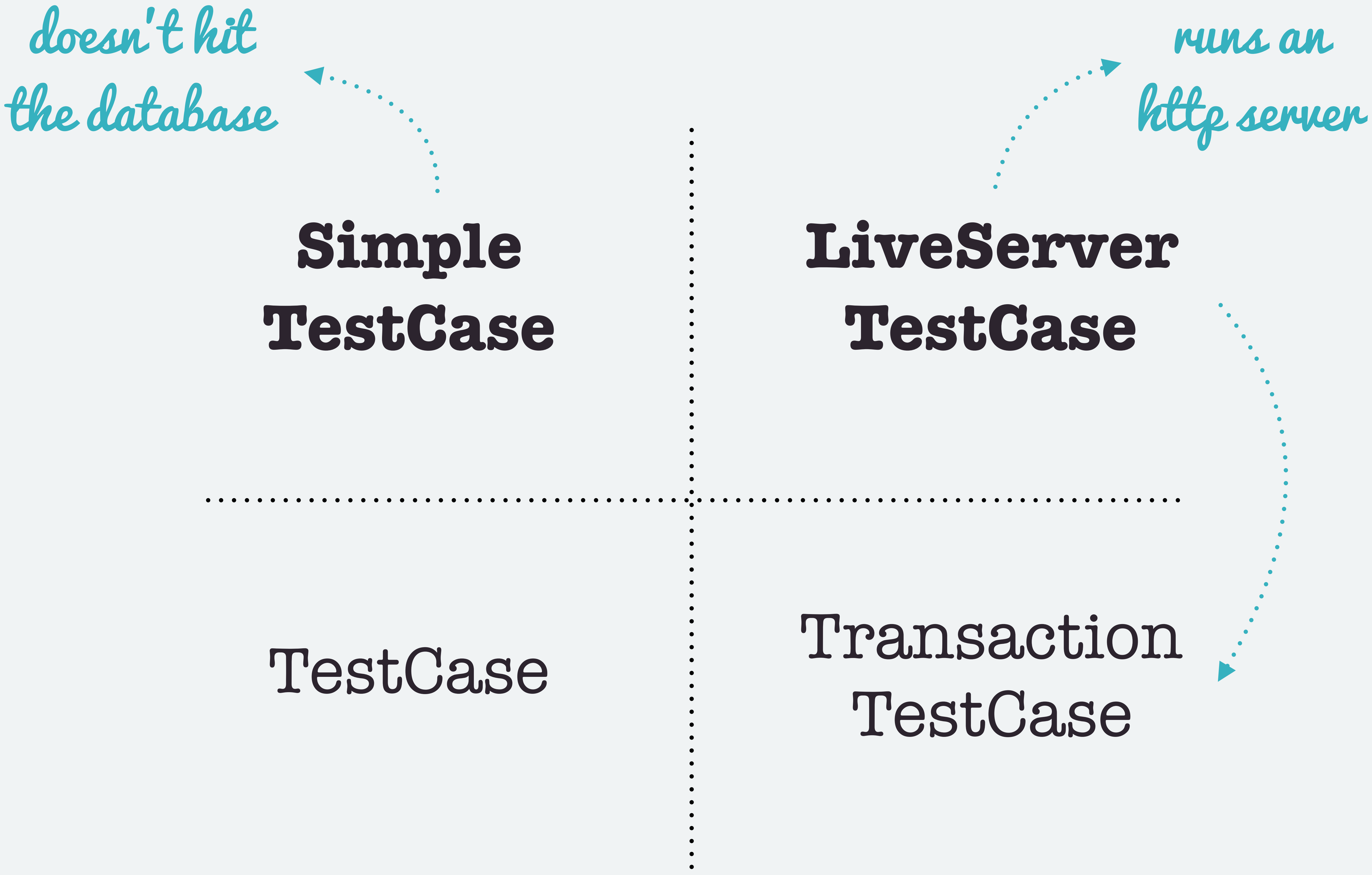
1.4

doesn't hit the database

runs an http server

Simple TestCase	**LiveServer TestCase**
TestCase	Transaction TestCase

1.4


doesn't hit the database
runs an http server
Simple TestCase
LiveServer TestCase
TestCase
Transaction TestCase

1.4

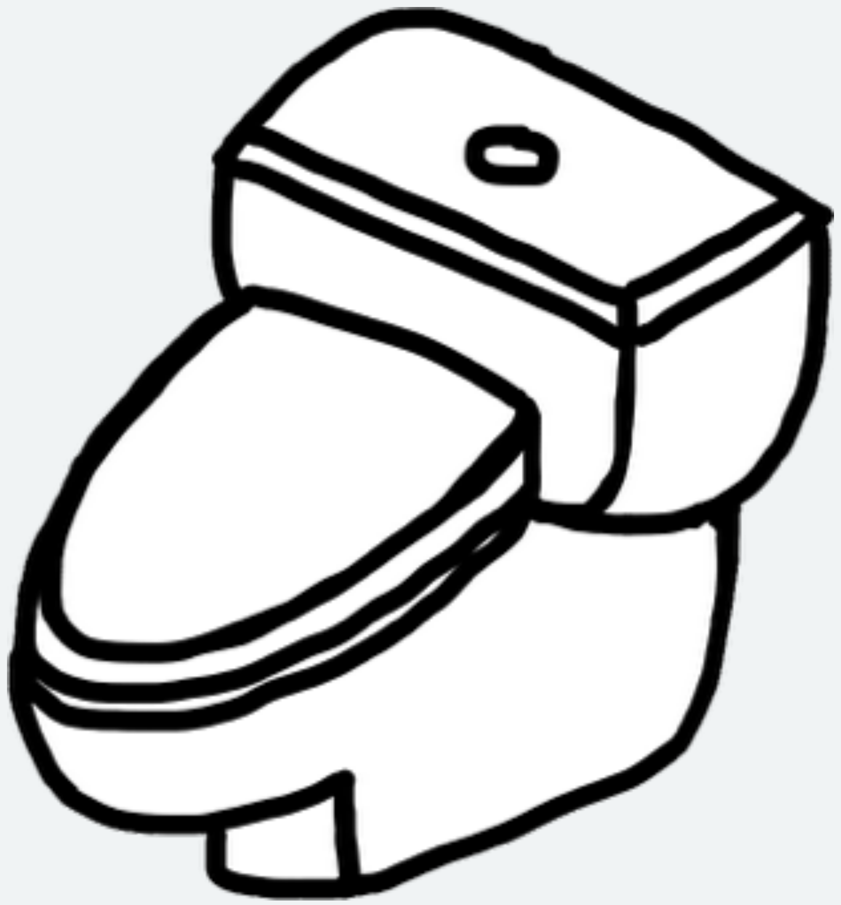
1.5

IMPROVEMENTS

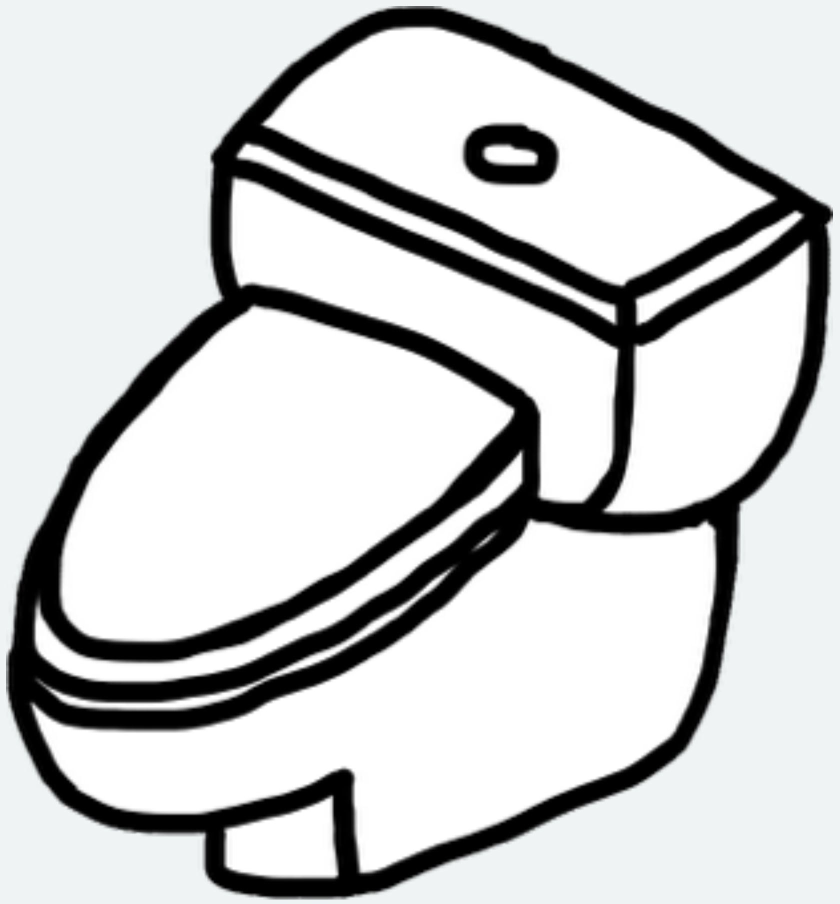
- Python 3
- Tutorial on testing
- New assertions

1.5

I am a TransactionTestCase.
I flush the database before each test.



I am a TransactionTestCase.
I flush the database ~~before~~ each test.
after



1.5

flush

run TransactionTestCase

enter transaction

run TestCase

rollback transaction

1.5

flush

run TransactionTestCase

enter transaction

run TestCase

rollback transaction

dirty state

1.5

flush

run TransactionTestCase

enter transaction

run TestCase

rollback transaction

1.5

enter transaction

run TestCase

rollback transaction

flush

run TransactionTestCase

flush

run TransactionTestCase

enter transaction

run TestCase

rollback transaction

run TransactionTestCase

flush

enter transaction

run TestCase

rollback transaction

problem solved

run TransactionTestCase

flush

enter transaction

run TestCase

rollback transaction

1.5

1.6

CLIENT

patch

1.6

IMPROVEMENTS

- Test discovery
- Full paths vs pseudo paths
- Doctests discovery

1.6

IMPROVEMENTS

- Test discovery
- Full paths vs ~~pseudo paths~~
- Doctests discovery

1.6

IMPROVEMENTS

- Test discovery
- Full paths vs ~~pseudo paths~~
- ~~Doctests discovery~~

1.6

1.7

unittest2

1.7

~~unittest2~~

unittest

1.7

LiveServerTestCase

StaticLiveServerTestCase

1.7

1.8

CLIENT

`trace`

1.8

TestCase

before

enter atomic

load fixtures

…

exit atomic

close connections

1.8

TestCase

before

enter atomic

load fixtures

...

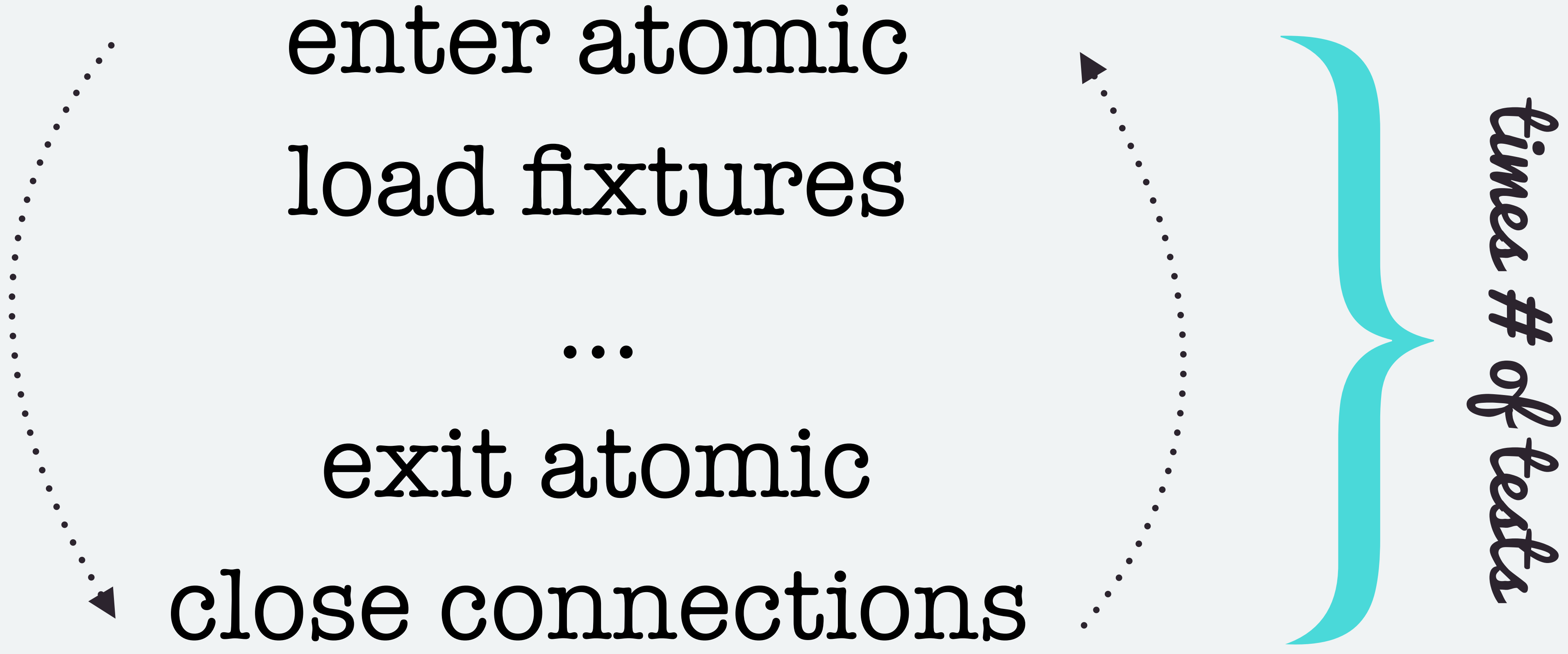
exit atomic

close connections

1.8

TestCase

before



1.8

TestCase

after

enter atomic

load fixtures

enter atomic

…

exit atomic

exit atomic

close connections

1.8

TestCase

after

enter atomic

load fixtures

enter atomic

…

exit atomic

exit atomic

close connections

1.8

TestCase

after

enter atomic

load fixtures

enter atomic

…

exit atomic

times # of tests

exit atomic

close connections

1.8

TestCase

after

once

enter atomic

load fixtures

enter atomic

...

exit atomic

times # of tests

exit atomic

close connections

1.8

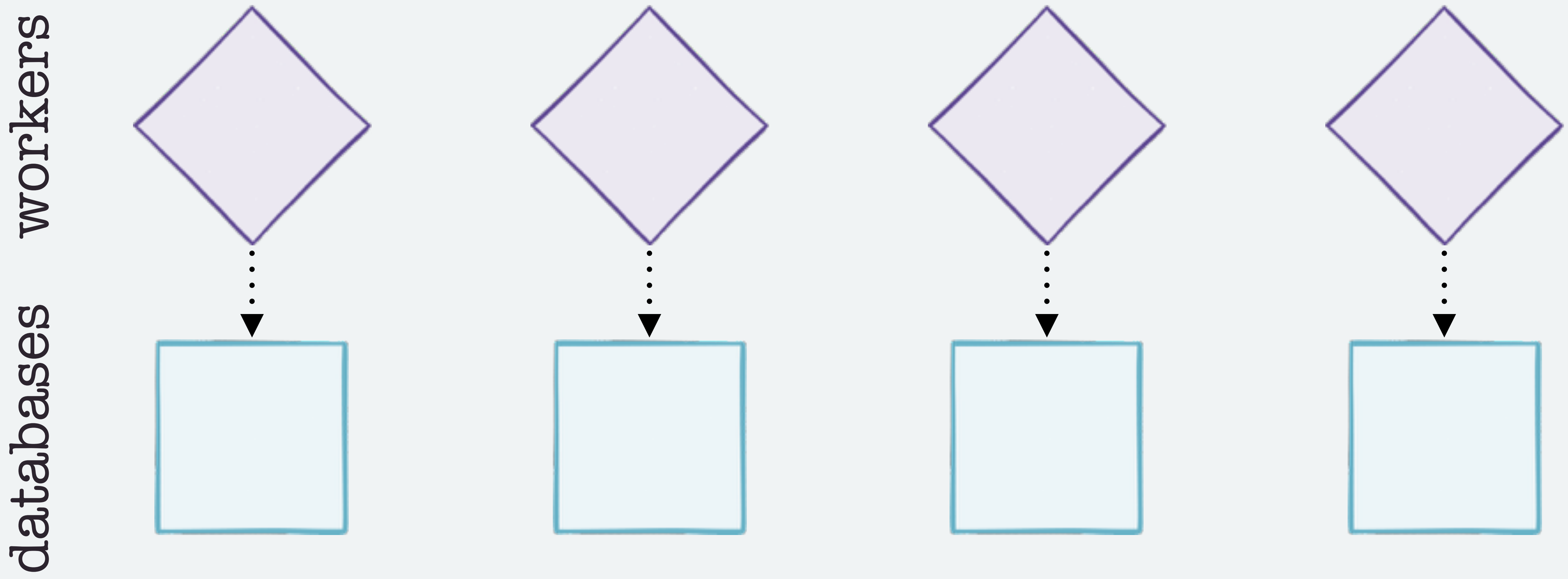
1.9

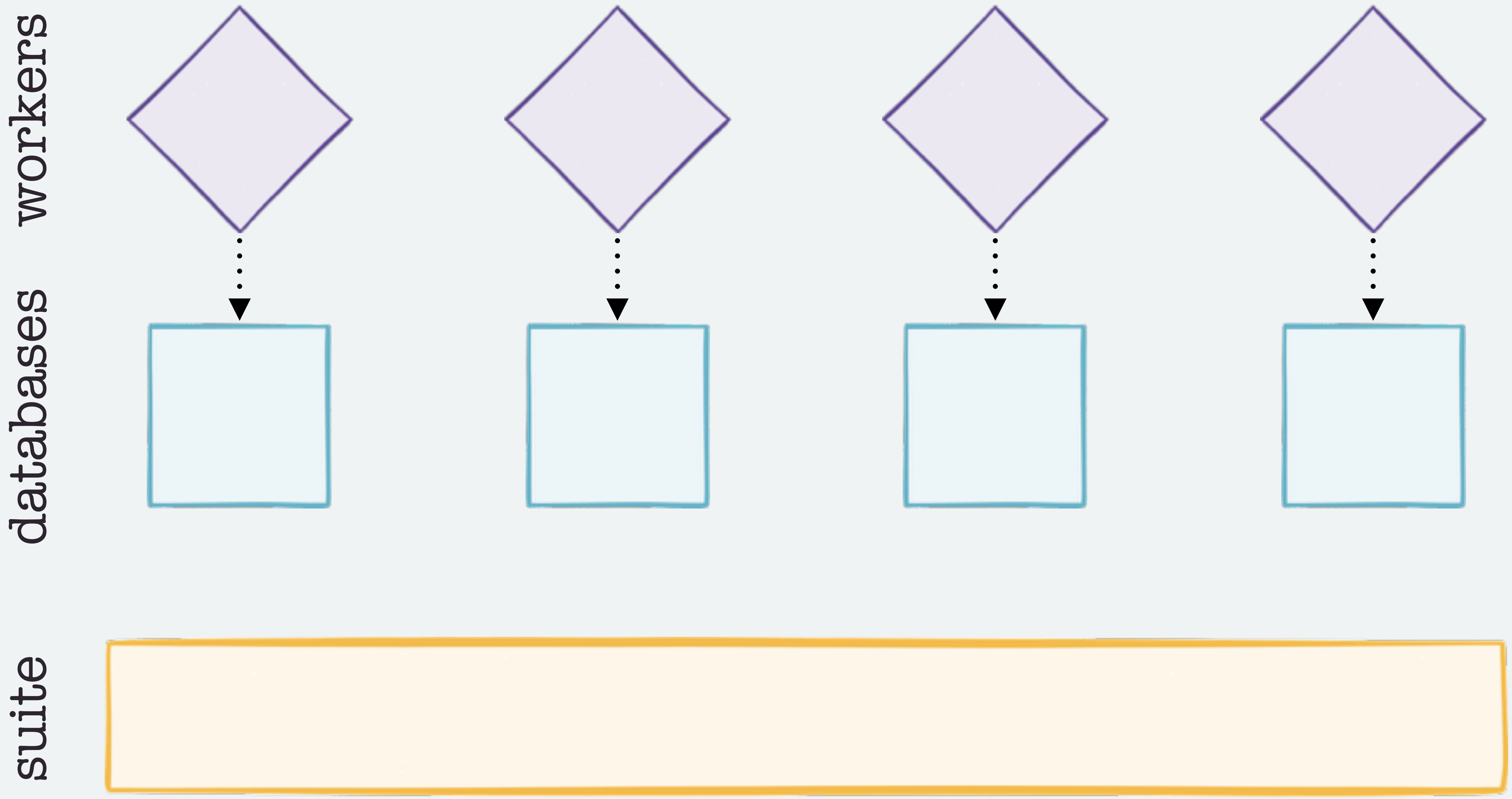
--parallel

1.9

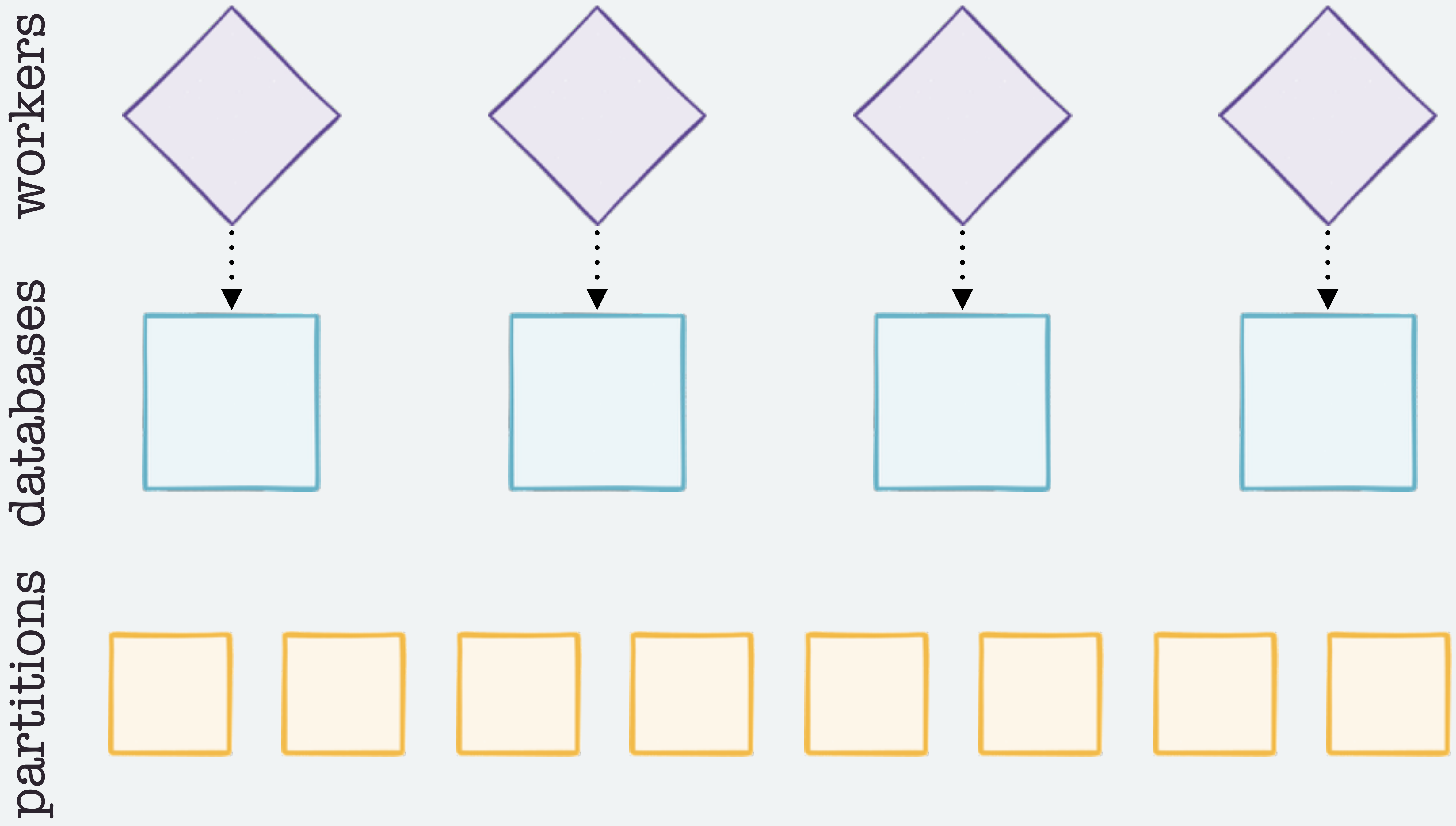
1.9

workers

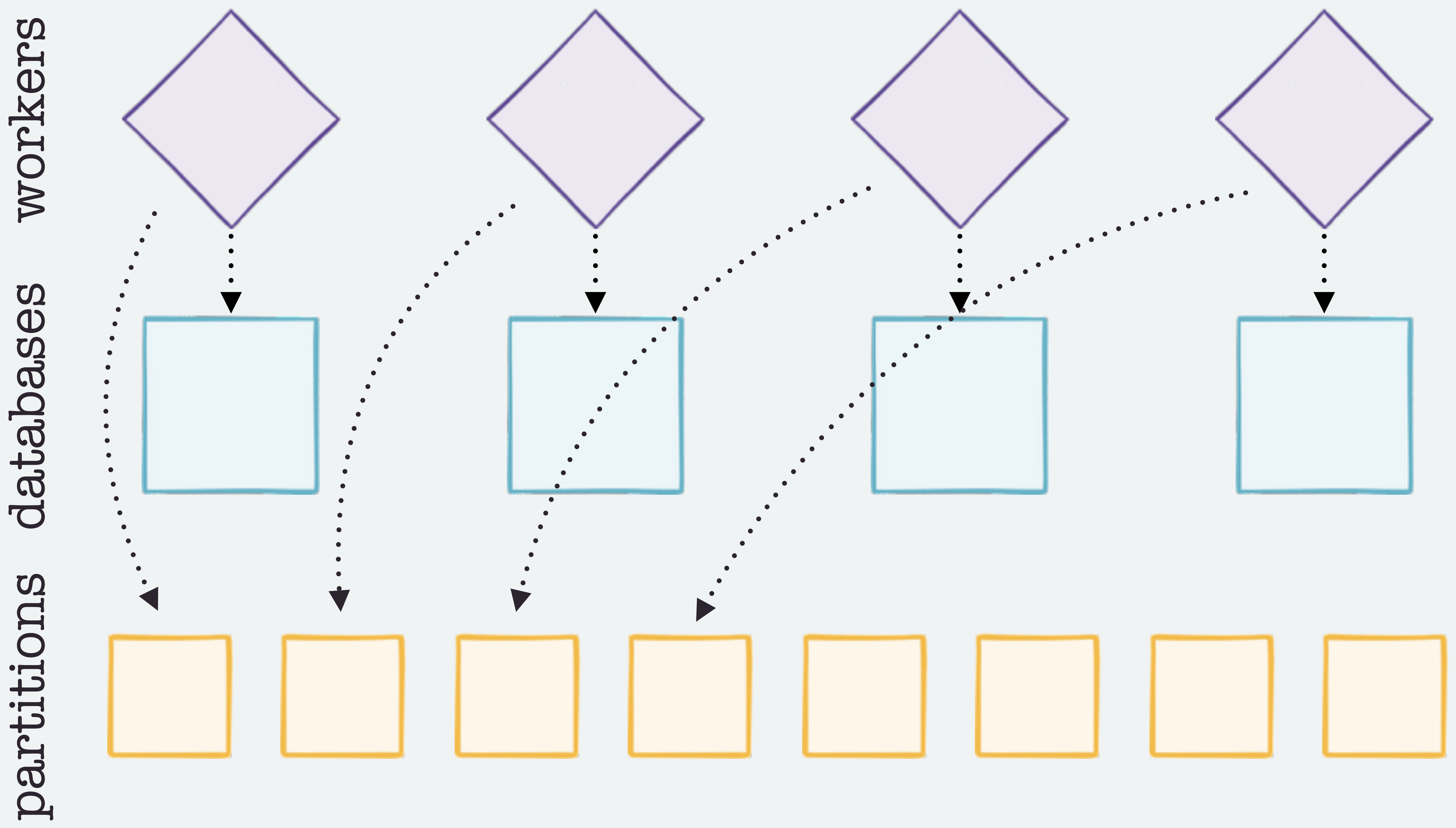

workers
databases

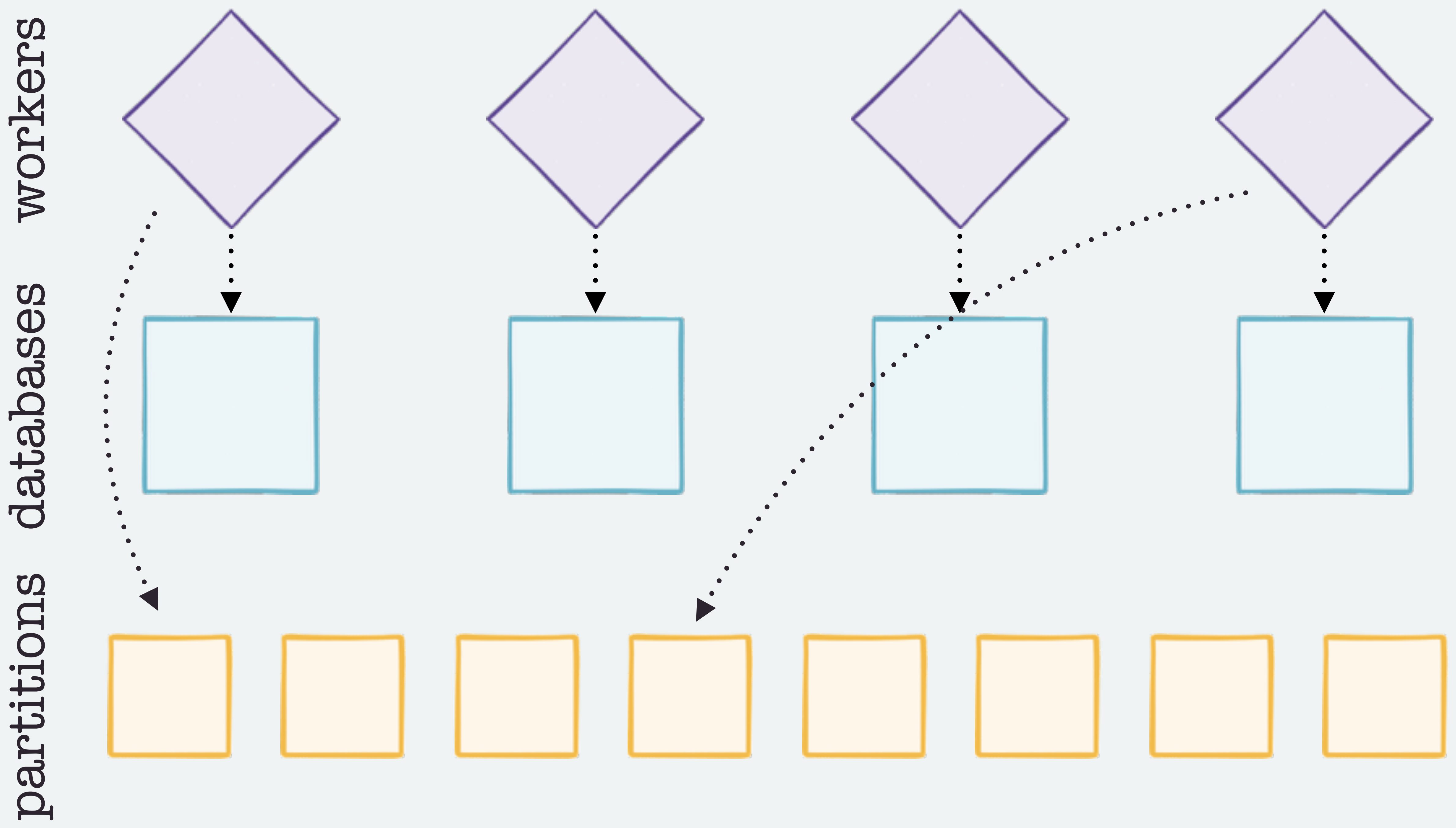


1.9

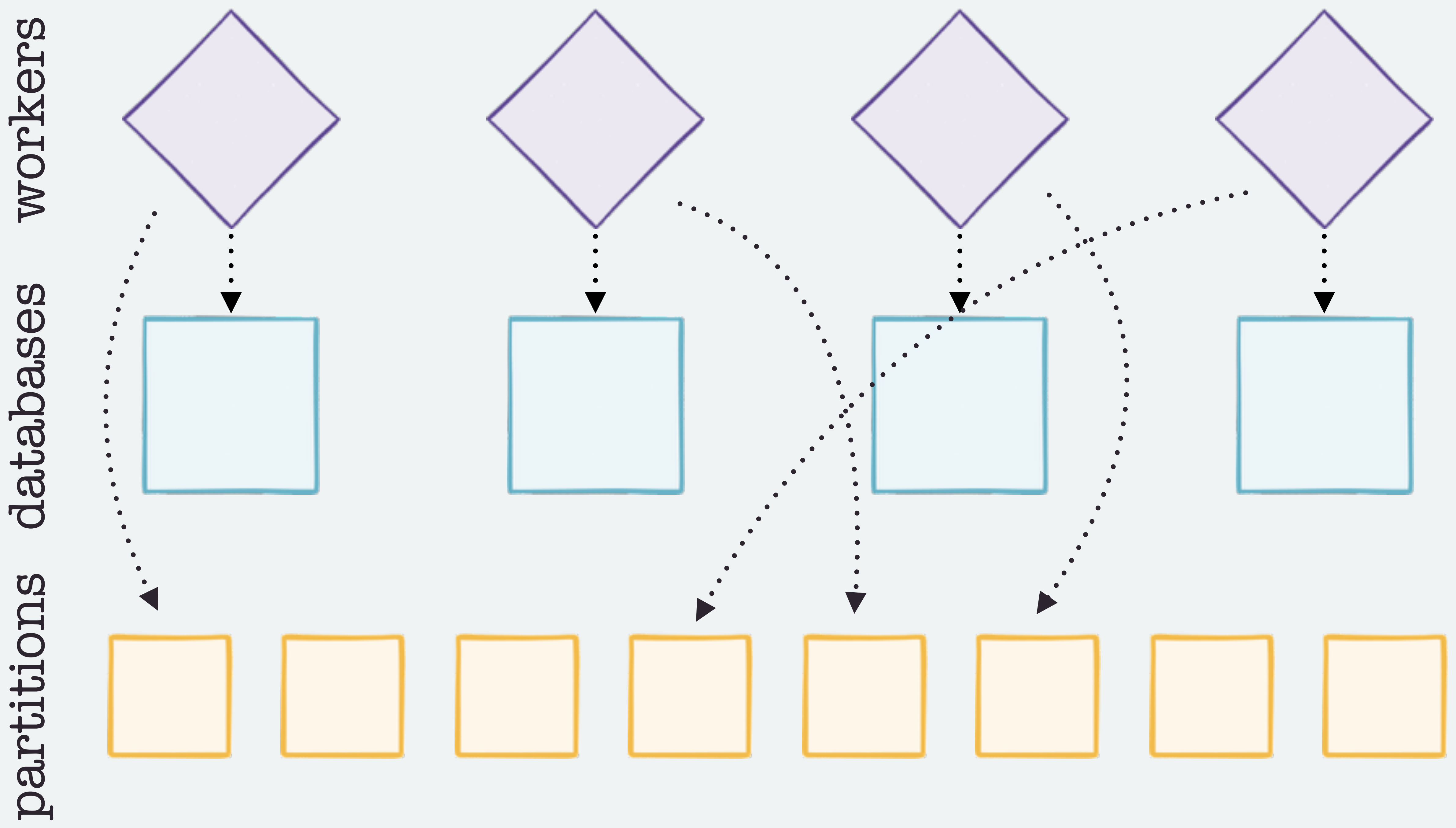


1.9

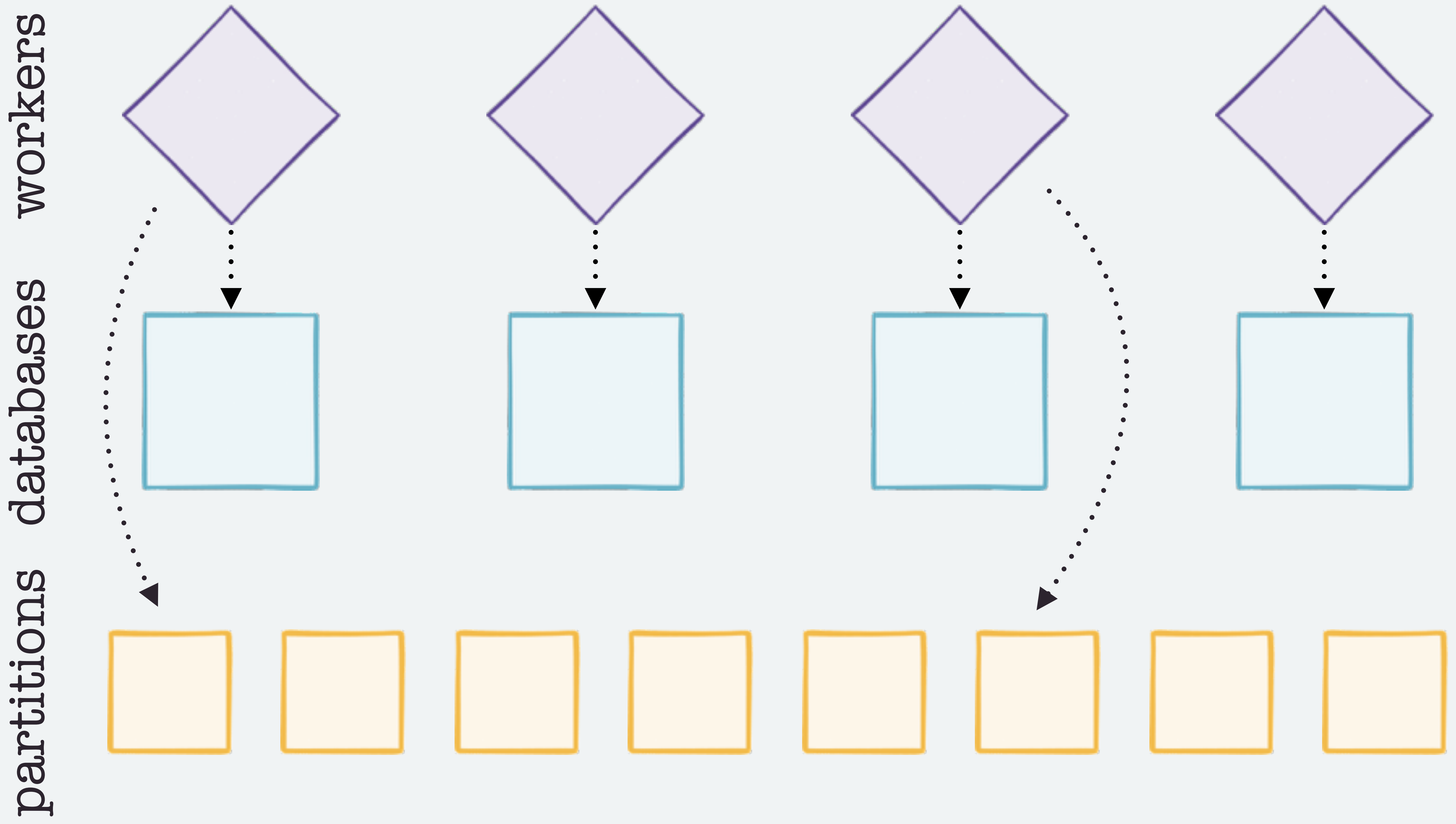

workers
databases
partitions



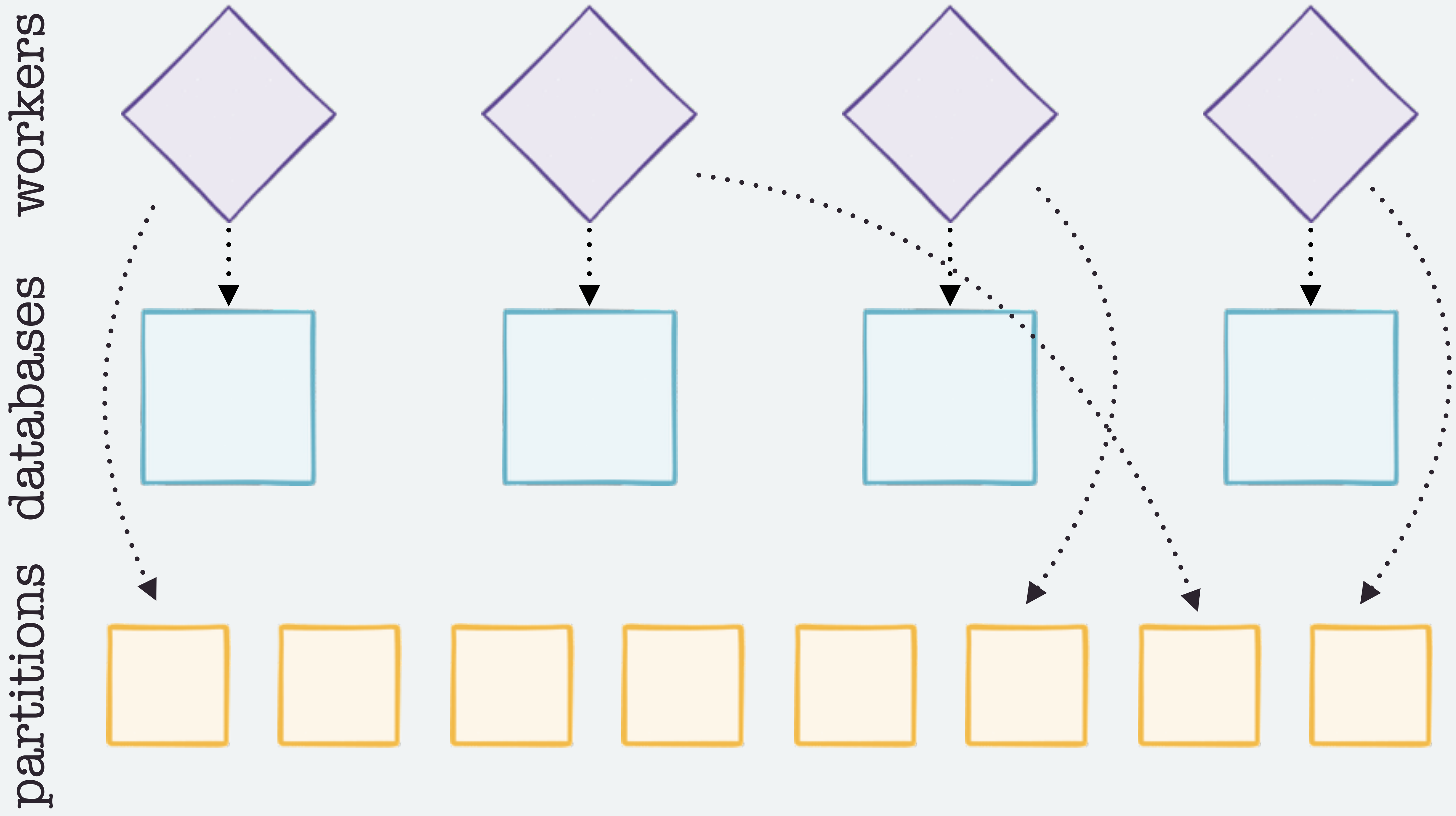
1.9



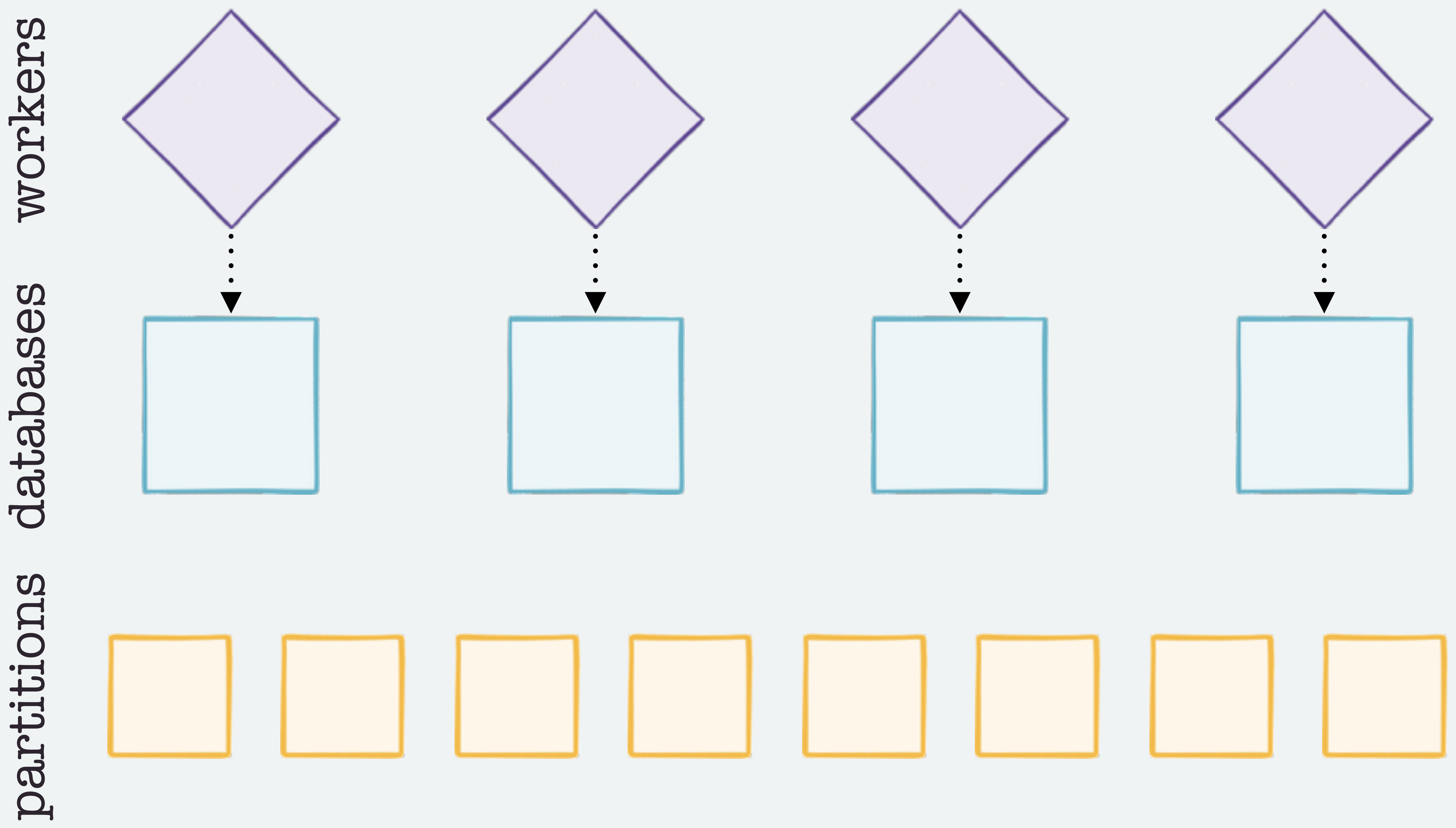
1.9



1.9



1.9


workers
databases
partitions

nose
multiprocess plugin

1.10

```
class SampleTestCase(TestCase):
    @tag('slow')
    def test_slow(self):
        ...
```

1.10

```
class SampleTestCase(TestCase):
    @tag('slow')
    def test_slow(self):
        ...
```

```
./manage.py test --tag=slow
```

1.10

```
class SampleTestCase(TestCase):
    @tag('slow')
    def test_slow(self):
        ...
```

```
./manage.py test --tag=slow
./manage.py test --exclude-tag=slow
```

1.10

nose attrib plugin

py.test markers

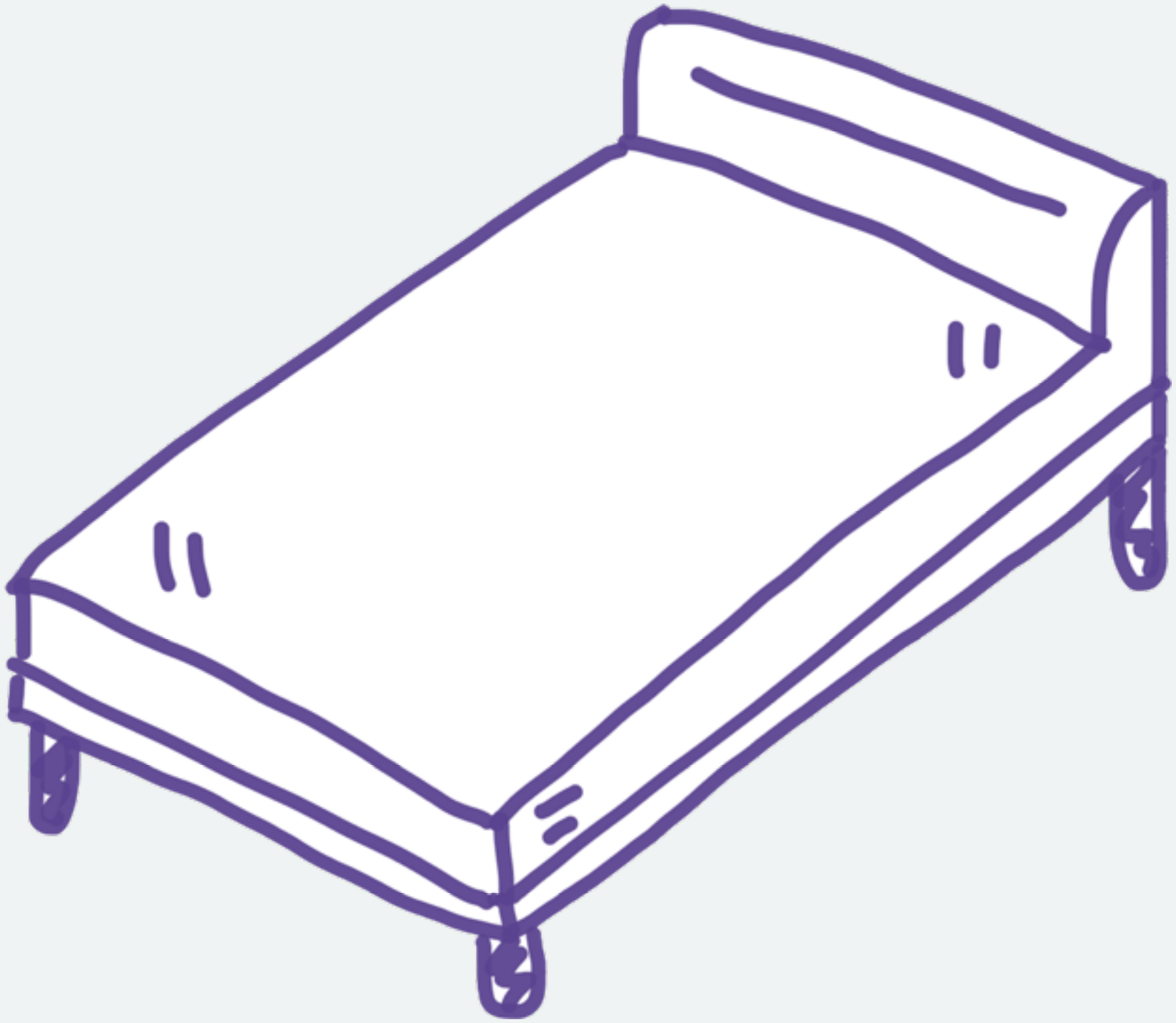
will do the same

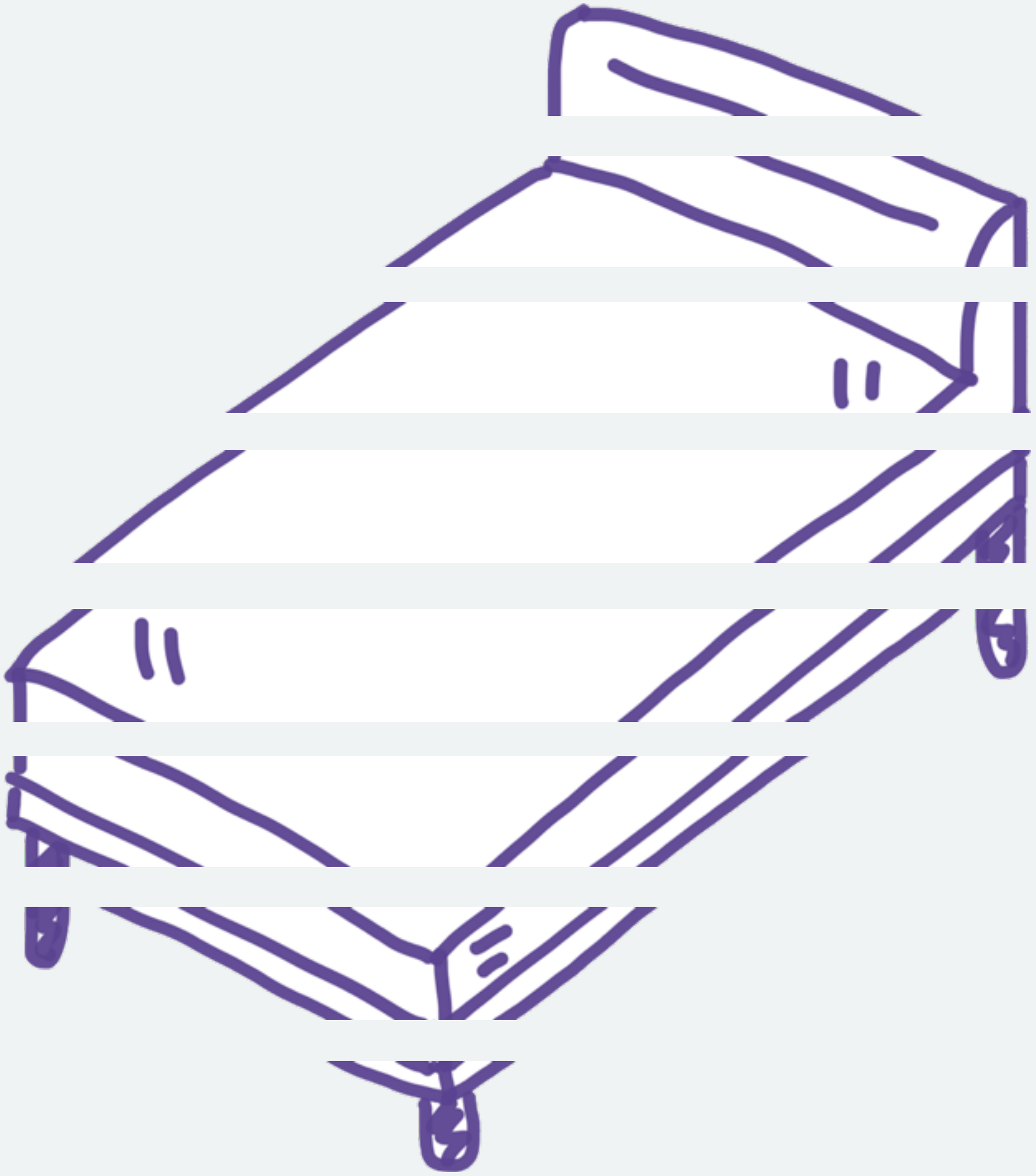
nose attrib plugin

py.test markers

TEST BED

or what happens when you run ./manage.py test





1

```
$ ./manage.py test
```

2 ../management/commands/test.py

```
TestRunner = get_runner(settings, options['testrunner'])
test_runner = TestRunner(**options)
failures = test_runner.run_tests(test_labels)
```

2

../management/commands/test.py

```
TestRunner = get_runner(settings, options['testrunner'])
test_runner = TestRunner(**options)
failures = test_runner.run_tests(test_labels)
```

3

```
self.setup_test_environment()
```

3

`self.setup_test_environment()`

locmem email backend

3

self.setup_test_environment()

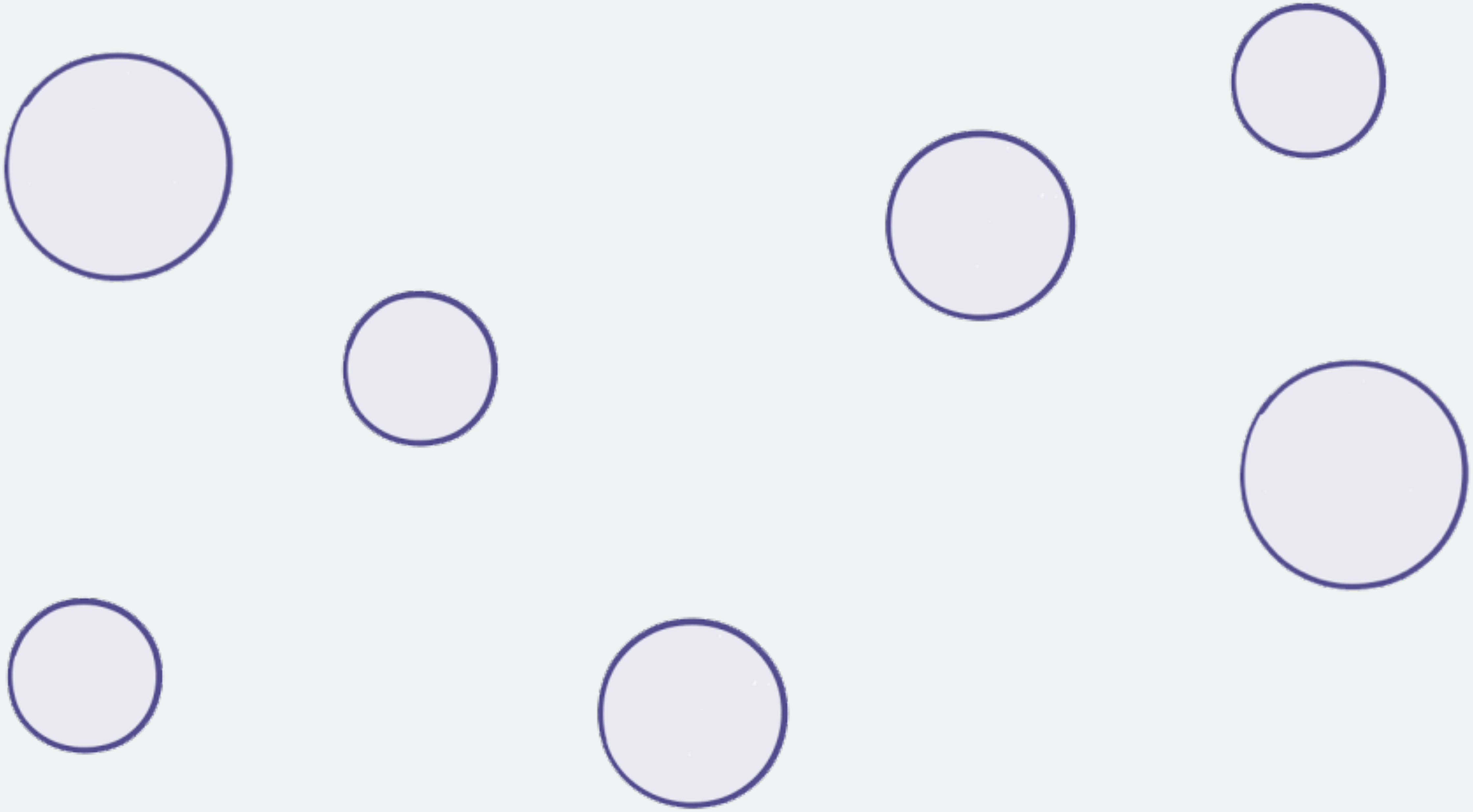
- locmem email backend
- instrumented test renderer

3 `self.setup_test_environment()`

- locmem email backend
- instrumented test renderer
- deactivate translations

4

`self.build_suite(test_labels, extra_tests)`



4

```
self.build_suite(test_labels, extra_tests)
```

5

```
self.setup_databases()
```

6

```
self.run_suite(suite)
```

7

```
self.teardown_databases(old_config)
```

5

```
self.setup_databases()
```

6

```
self.run_suite(suite)
```

7

```
self.teardown_databases(old_config)
```

5 `self.setup_databases()`

6 `self.run_suite(suite)`

7 `self.teardown_databases(old_config)`

8

```
self.teardown_test_environment()
```

8

```
self.teardown_test_environment()
```

original email backend

8

`self.teardown_test_environment()`

✉ original email backend

✎ original test renderer

8

`self.teardown_test_environment()`

original email backend

original test renderer

delete state and mailbox

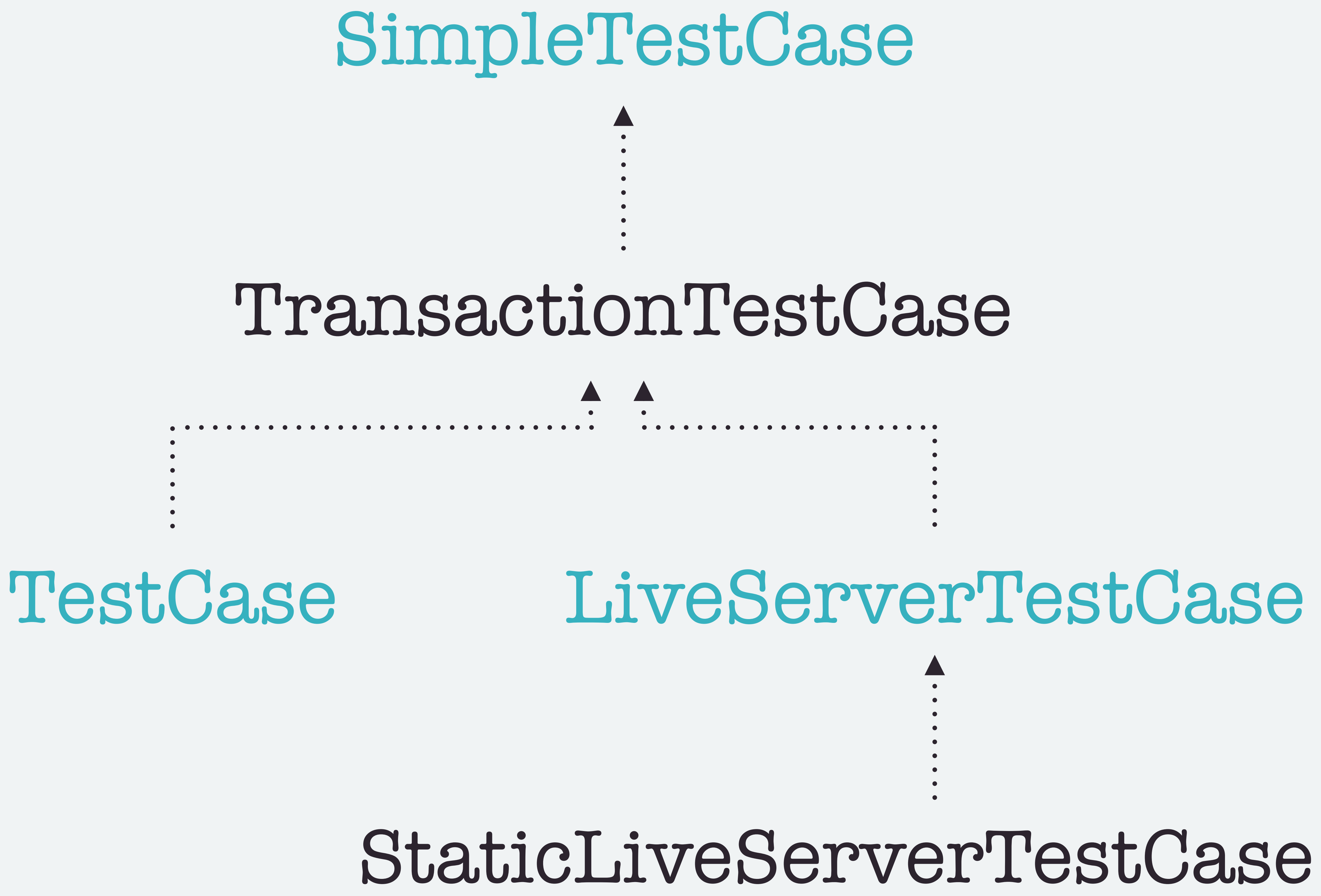
9

```
self.suite_result(suite, result)
```

```
len(result.failures) + len(result.errors)
```

```
if failures:
    sys.exit(1)
```

ALL THE TEST CLASSES


SimpleTestCase
TransactionTestCase
TestCase
LiveServerTestCase
StaticLiveServerTestCase

SimpleTestCase

- no database queries
- access to test client
- fast

TransactionTestCase

- allows database queries
- access to test client
- ~~fast~~
- allows database transactions
- flushes database after each test

TestCase

- allows database queries
- access to test client
- faster
- restricts database transactions
- runs each test in a transaction

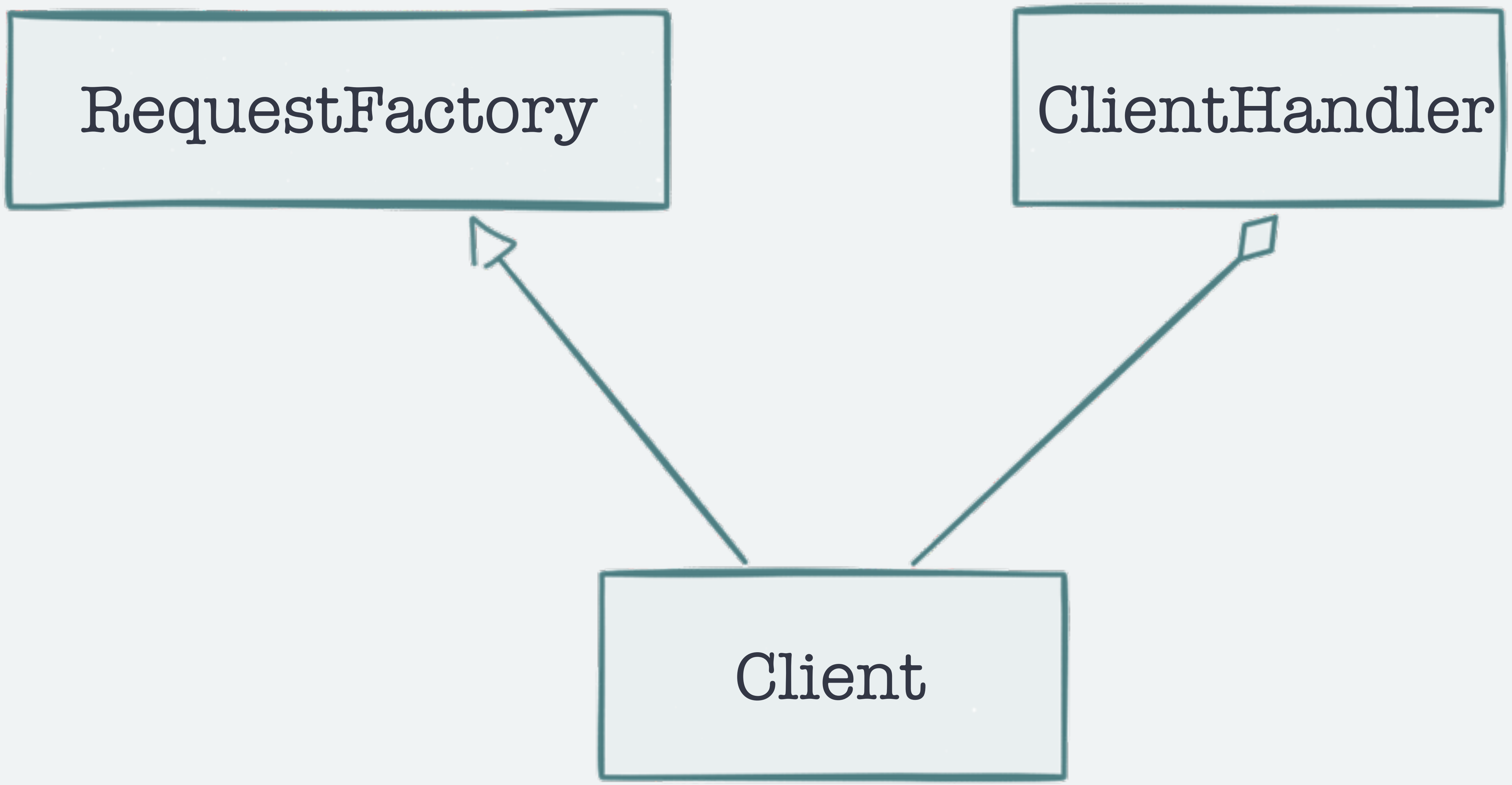
LiveServerTestCase

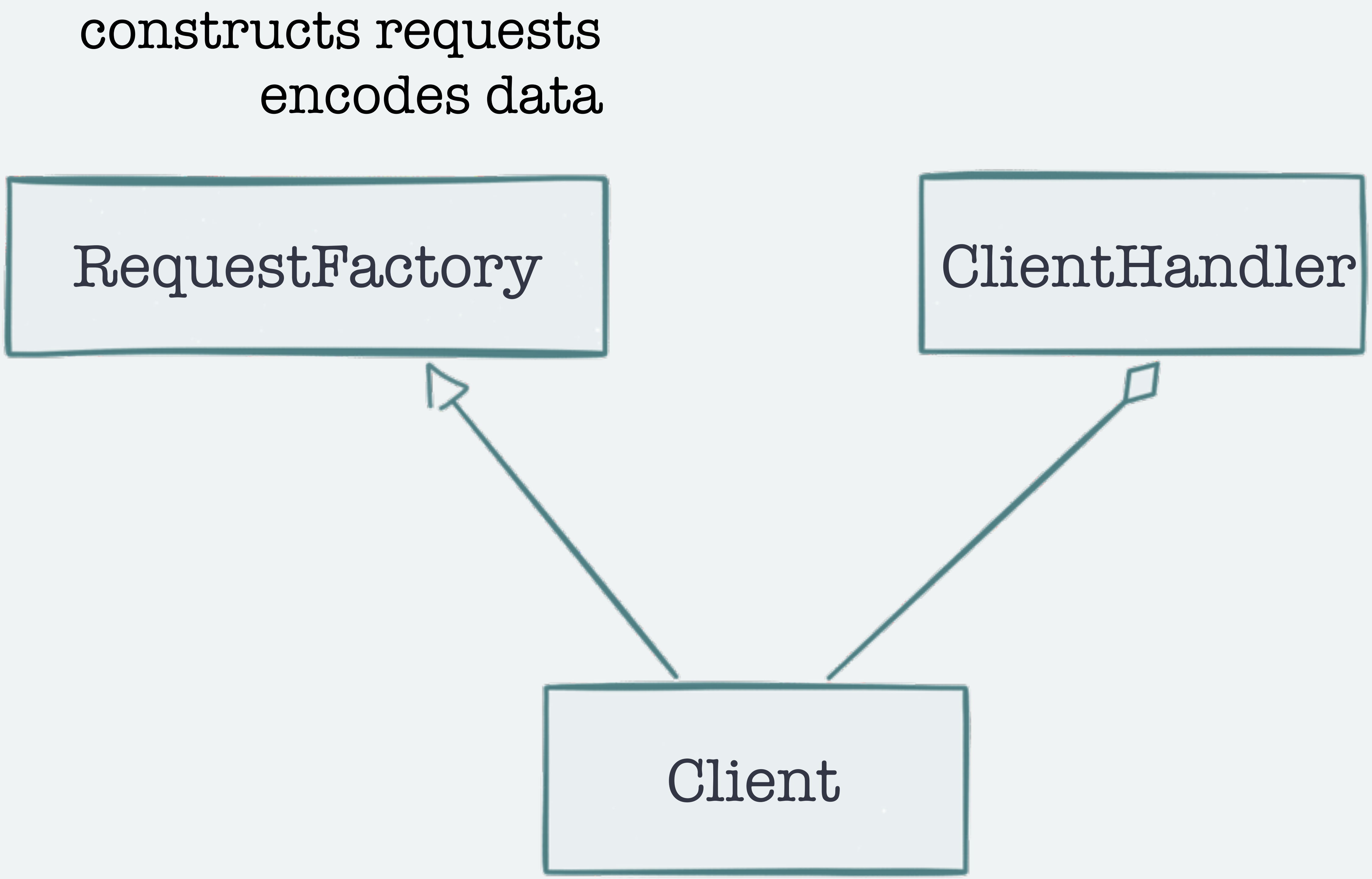
- acts like TransactionTestCase
- launches a live HTTP server in a separate thread

StaticLiveServerTestCase

- acts like TransactionTestCase
- launches a live HTTP server in a separate thread
- serves static files

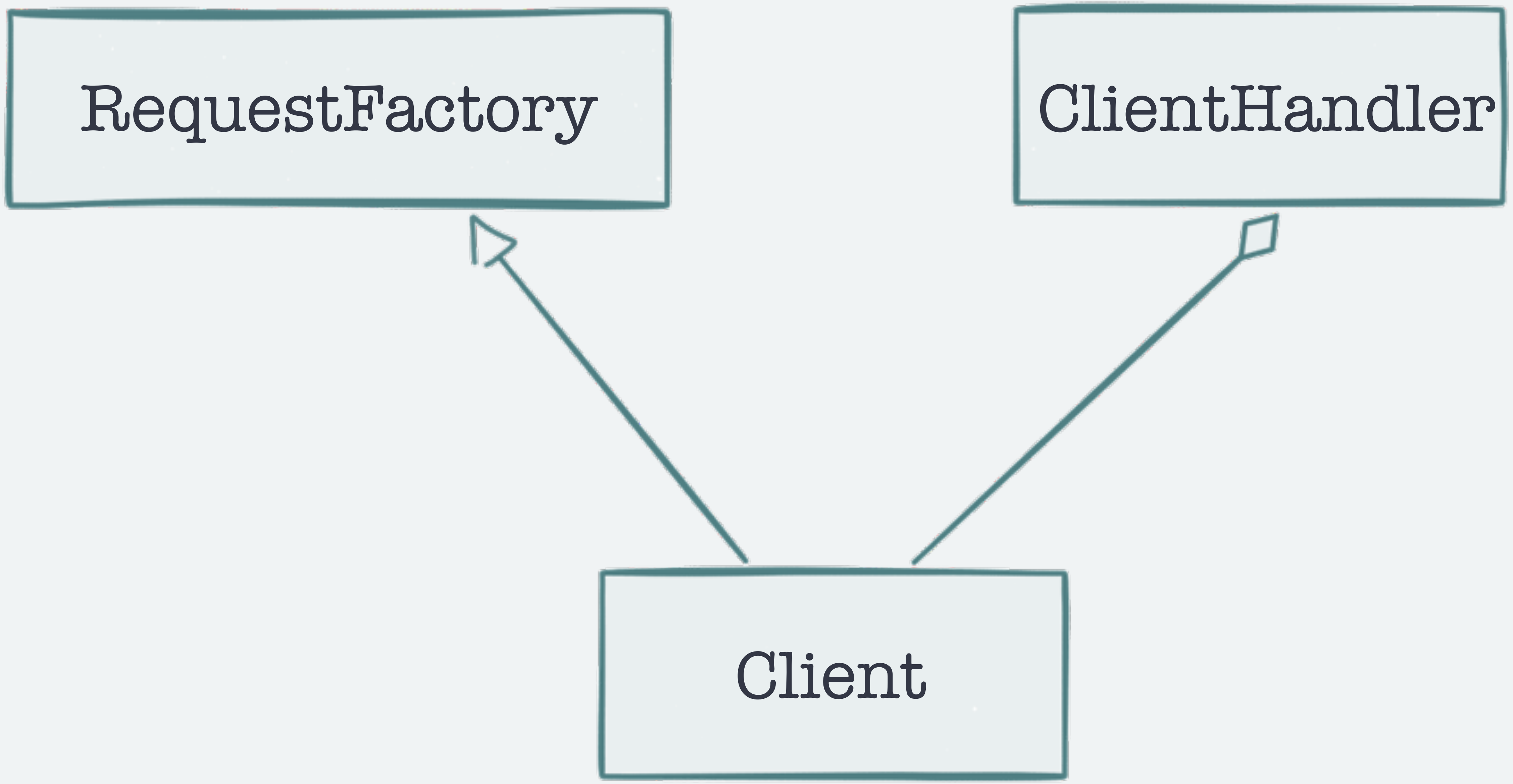
CLIENT

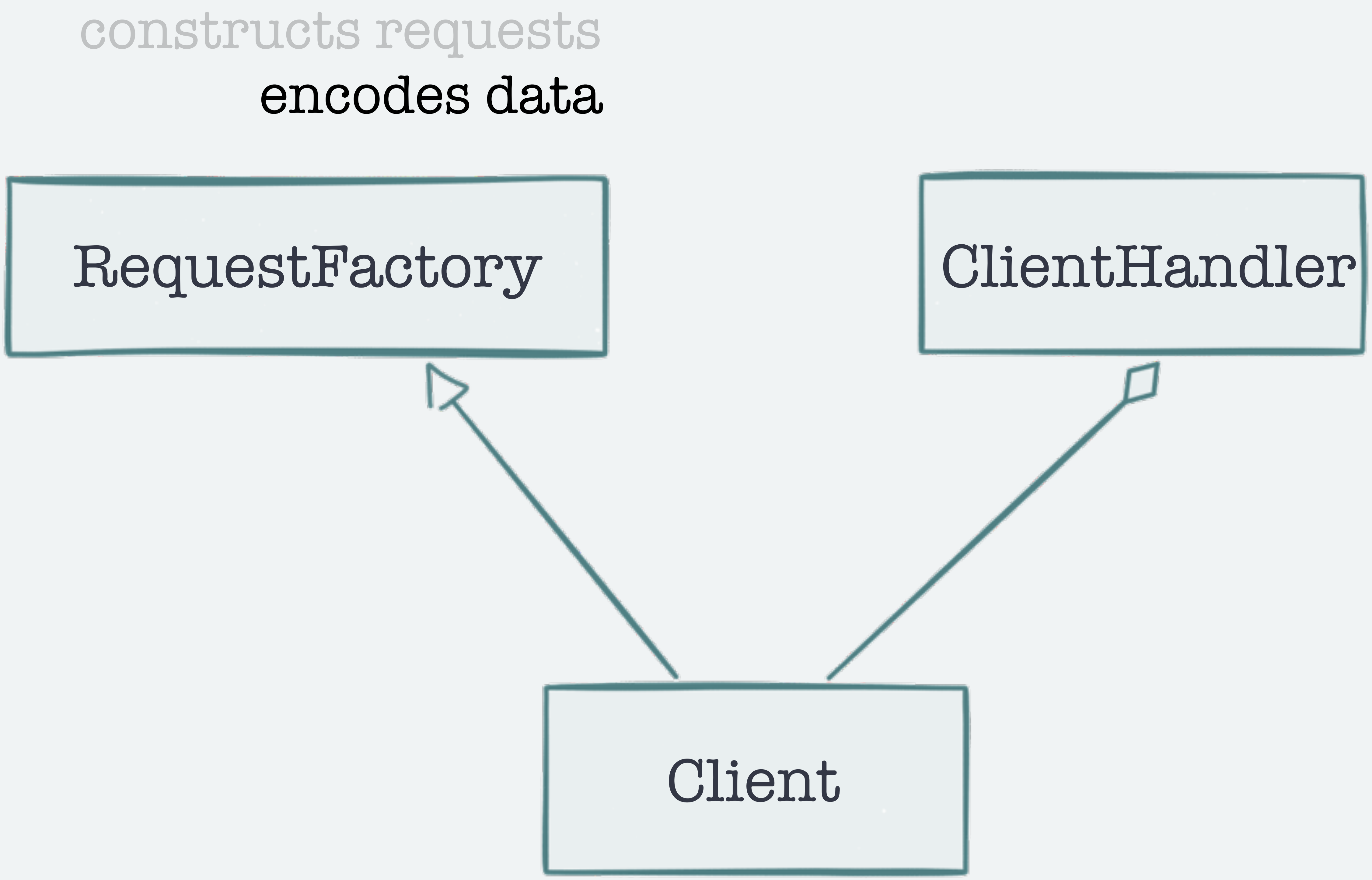

RequestFactory
ClientHandler
Client

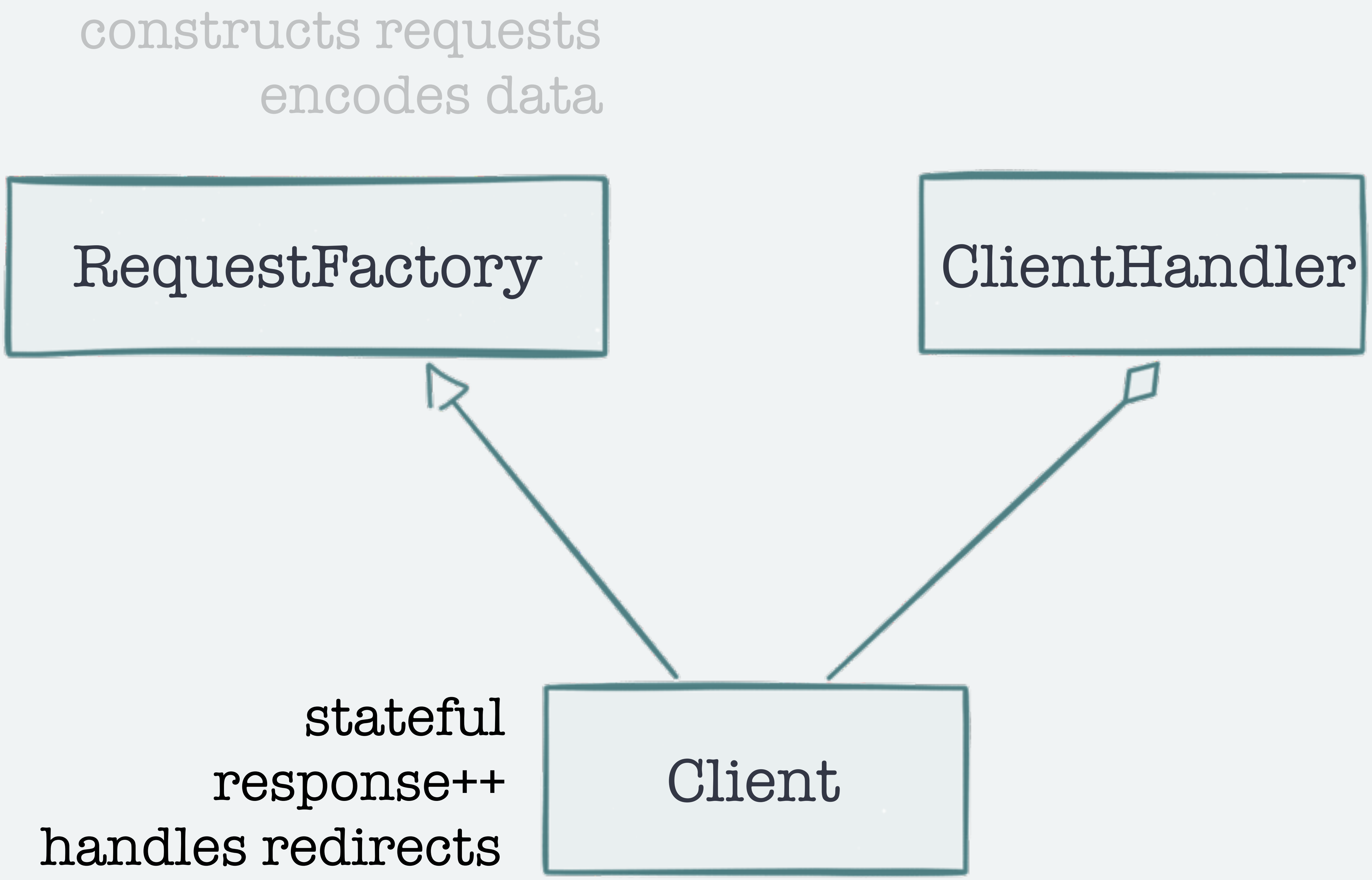

constructs requests
encodes data
RequestFactory
ClientHandler
Client

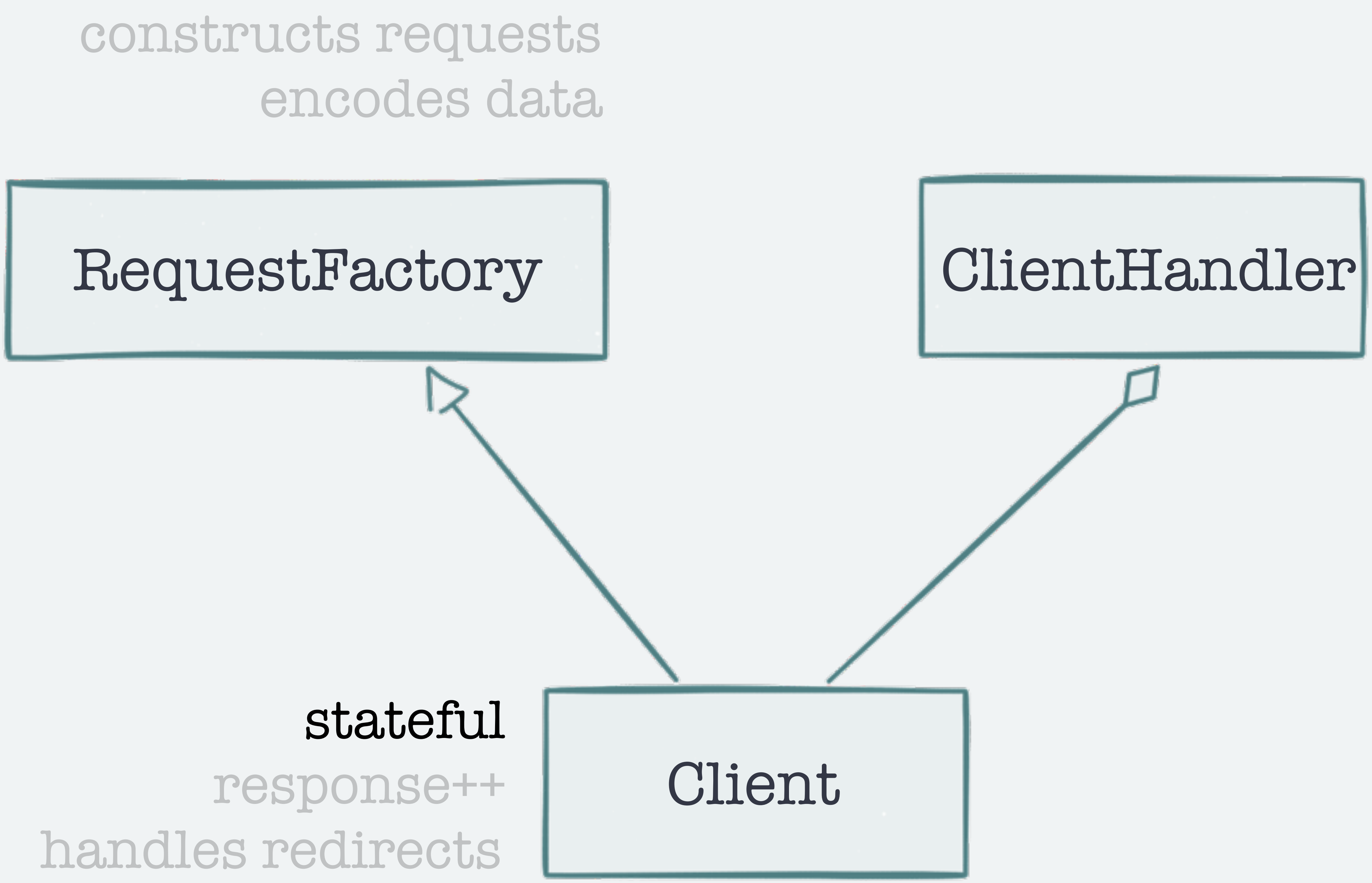
constructs requests

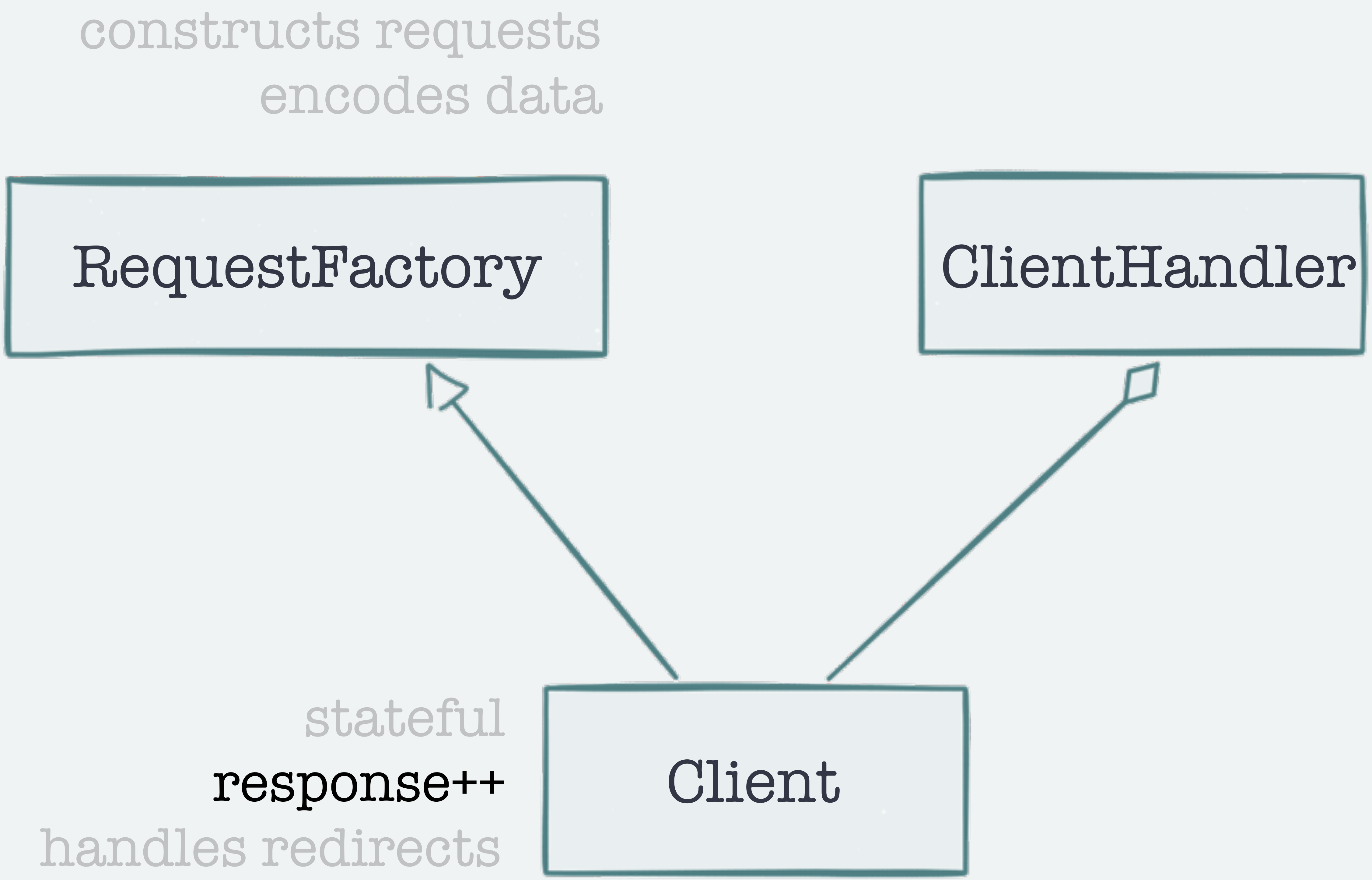
encodes data

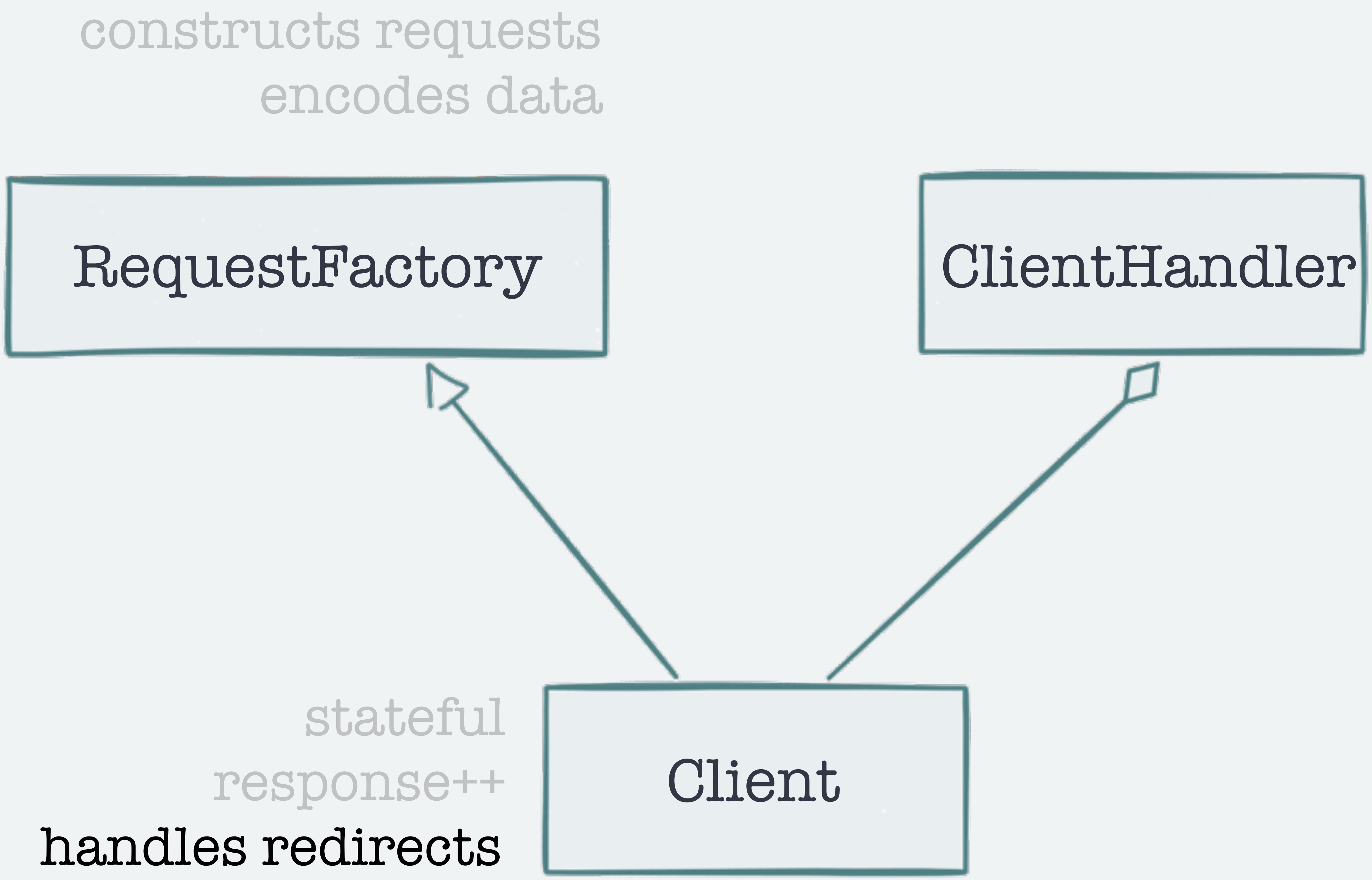


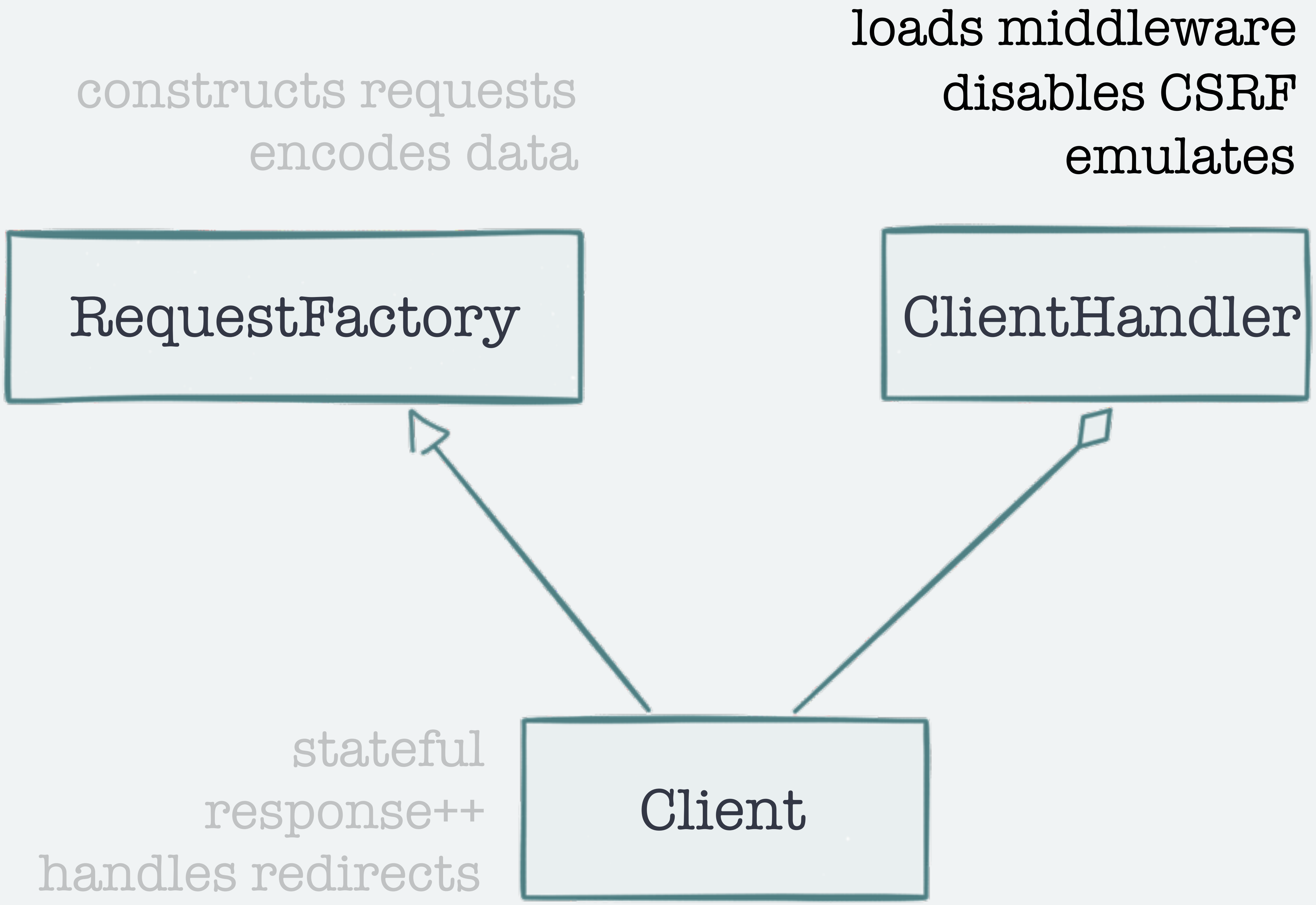

~~constructs requests~~
encodes data
RequestFactory
ClientHandler
Client

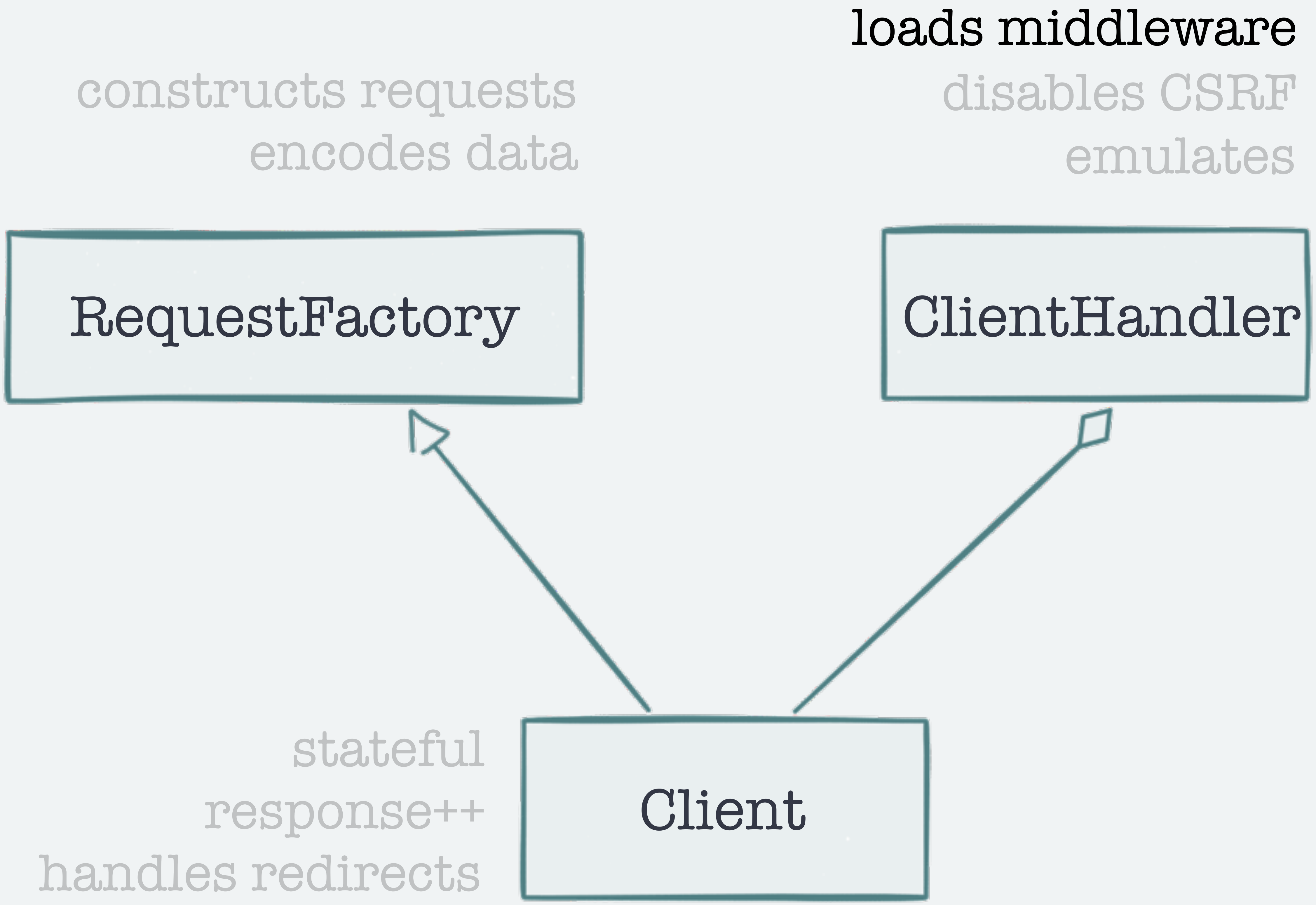

constructs requests
encodes data
RequestFactory
ClientHandler
stateful
response++
handles redirects
Client

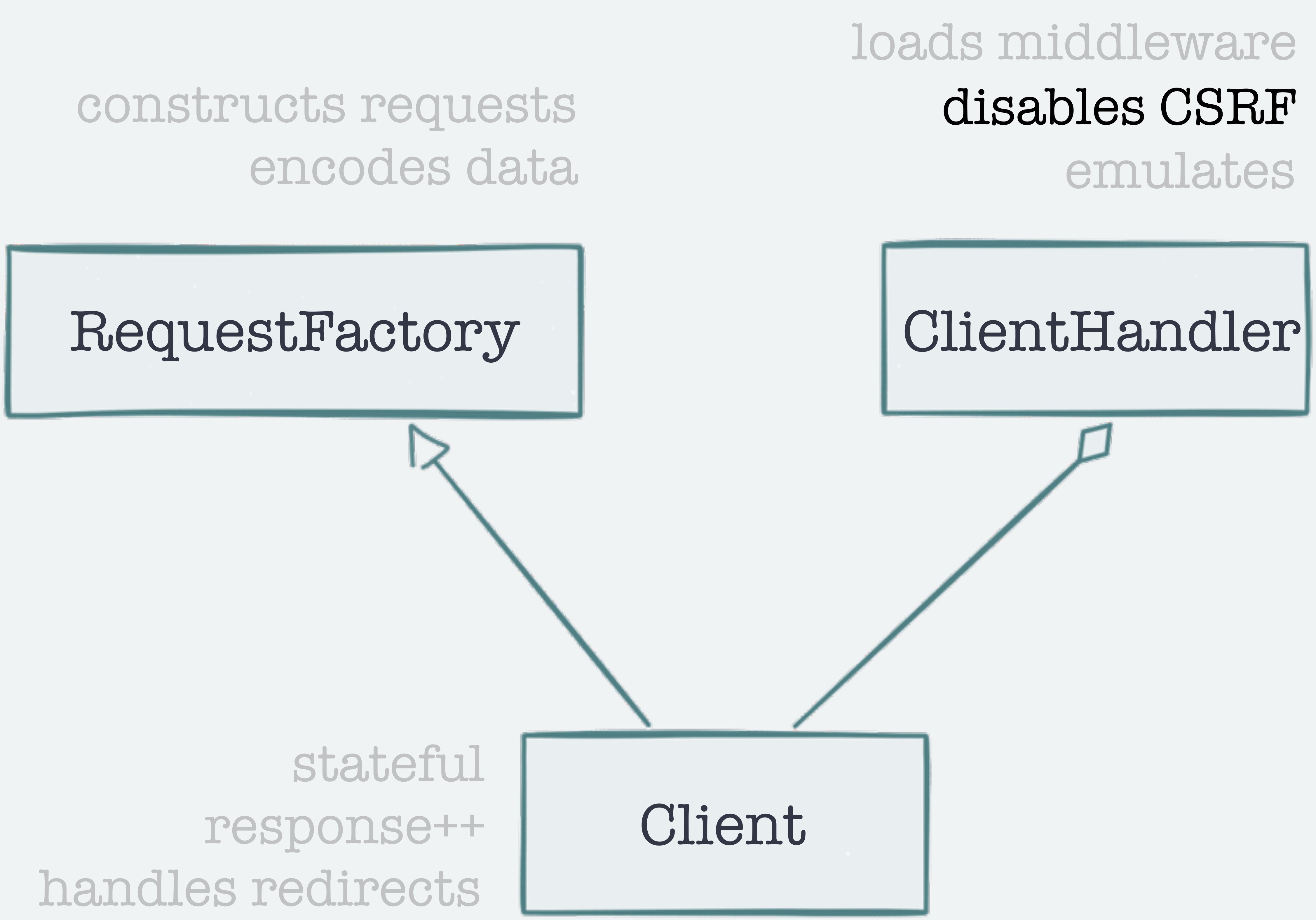

constructs requests
encodes data
RequestFactory
ClientHandler
stateful
response++
handles redirects
Client

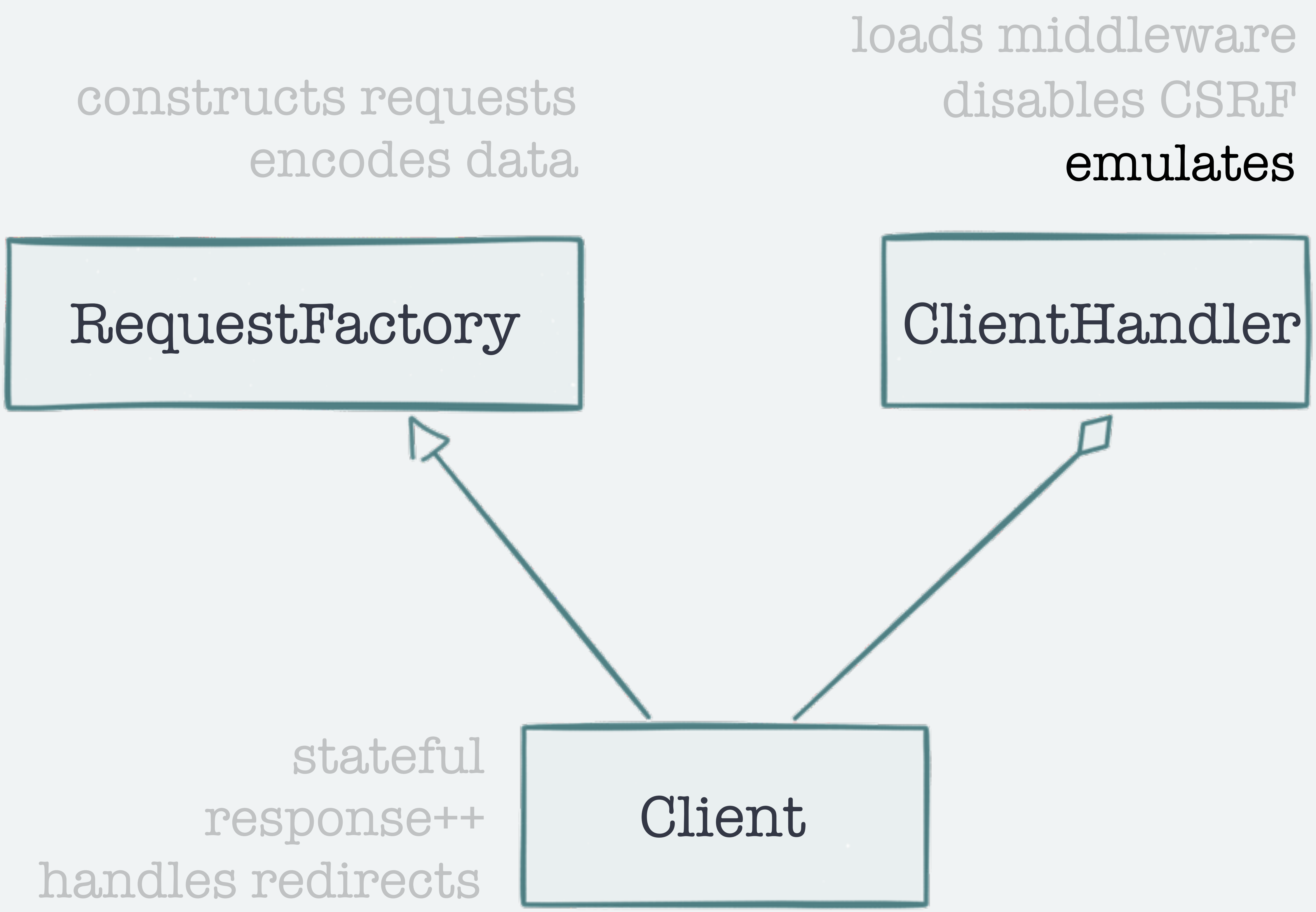

constructs requests
encodes data
RequestFactory
ClientHandler
stateful
response++
handles redirects
Client

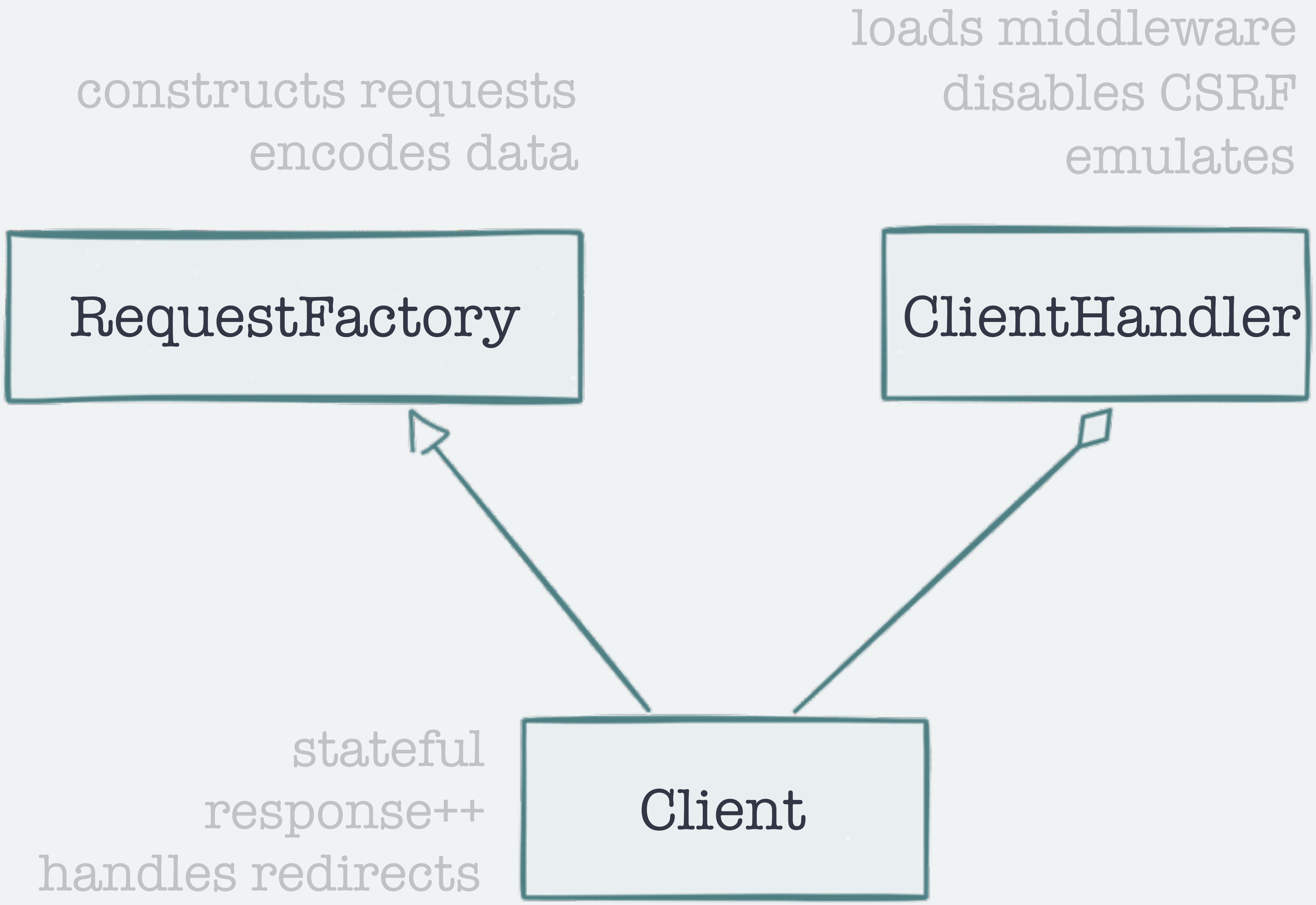

constructs requests
encodes data
RequestFactory
ClientHandler
stateful
response++
handles redirects
Client


constructs requests
encodes data
loads middleware
disables CSRF
emulates
RequestFactory
ClientHandler
stateful
response++
handles redirects
Client


loads middleware
disables CSRF
emulates
constructs requests
encodes data
RequestFactory
ClientHandler
Client
stateful
response++
handles redirects


loads middleware
disables CSRF
emulates
constructs requests
encodes data
RequestFactory
ClientHandler
stateful
response++
handles redirects
Client


loads middleware
disables CSRF
emulates
constructs requests
encodes data
RequestFactory
ClientHandler
stateful
response++
handles redirects
Client


constructs requests
encodes data
loads middleware
disables CSRF
emulates
RequestFactory
ClientHandler
stateful
response++
handles redirects
Client

QUALITY

Factory Boy

fixtures replacement with random and realistic values

Factory Boy

fixtures replacement with random and realistic values generated by

Faker

property based testing with

Hypothesis

```python
from hypothesis import given
from hypothesis.strategies import text

@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

```python
from hypothesis import given
from hypothesis.strategies import text

@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

```python
from hypothesis import given
from hypothesis.strategies import text

@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

```
from hypothesis import given
from hypothesis.strategies import text

@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

```python
from hypothesis import given
from hypothesis.strategies import text

@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

```python
from hypothesis import given
from hypothesis.strategies import text

@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

rerun for different values

```python
from hypothesis.extra.django.models import models
from hypothesis.strategies import integers

models(Customer).example()
models(Customer,
    age=integers(
      min_value=0,
      max_value=120)
    ).example()
```

```
from hypothesis.extra.django.models import models
from hypothesis.strategies import integers

models(Customer).example()
models(Customer,
       age=integers(
           min_value=0,
           max_value=120)
       ).example()
```

```python
from hypothesis.extra.django import TestCase
from hypothesis import given
from hypothesis.extra.django.models import models
from hypothesis.strategies import lists, integers


class TestProjectManagement(TestCase):
    @given(
        models(Project, collaborator_limit=integers(min_value=0, max_value=20)),
        lists(models(User), max_size=20))
    def test_can_add_users_up_to_collaborator_limit(self, project, collaborators):
        for c in collaborators:
            if project.at_collaboration_limit():
                with self.assertRaises(LimitReached):
                    project.add_user(c)
                self.assertFalse(project.team_contains(c))
            else:
                project.add_user(c)
                self.assertTrue(project.team_contains(c))
```

```python
from hypothesis.extra.django import TestCase
from hypothesis import given
from hypothesis.extra.django.models import models
from hypothesis.strategies import lists, integers


class TestProjectManagement(TestCase):
  @given(
    models(Project, collaborator_limit=integers(min_value=0, max_value=20)),
    lists(models(User), max_size=20))
  def test_can_add_users_up_to_collaborator_limit(self, project, collaborators):
    for c in collaborators:
      if project.at_collaboration_limit():
        with self.assertRaises(LimitReached):
          project.add_user(c)
        self.assertFalse(project.team_contains(c))
      else:
        project.add_user(c)
        self.assertTrue(project.team_contains(c))
```

```
class TestProjectManagement(TestCase):
    @given(
        models(Project, collaborator_limit=integers(min_value=0, max_value=20)),
        lists(models(User), max_size=20))
    def test_can_add_users_up_to_collaborator_limit(self, project, collaborators):
        for c in collaborators:
            if project.at_collaboration_limit():
                with self.assertRaises(LimitReached):
                    project.add_user(c)
                self.assertFalse(project.team_contains(c))
            else:
                project.add_user(c)
                self.assertTrue(project.team_contains(c))
```

```
class TestProjectManagement(TestCase):
    @given(
        models(Project, collaborator_limit=integers(min_value=0, max_value=20)),
        lists(models(User), max_size=20))
    def test_can_add_users_up_to_collaborator_limit(self, project, collaborators):
        for c in collaborators:
            if project.at_collaboration_limit():
                with self.assertRaises(LimitReached):
                    project.add_user(c)
                self.assertFalse(project.team_contains(c))
            else:
                project.add_user(c)
                self.assertTrue(project.team_contains(c))
```

```
class TestProjectManagement(TestCase):
    @given(
        models(Project, collaborator_limit=integers(min_value=0, max_value=20)),
        lists(models(User), max_size=20))
    def test_can_add_users_up_to_collaborator_limit(self, project, collaborators):
        for c in collaborators:
            if project.at_collaboration_limit():
                with self.assertRaises(LimitReached):
                    project.add_user(c)
                self.assertFalse(project.team_contains(c))
            else:
                project.add_user(c)
                self.assertTrue(project.team_contains(c))
```

```
class TestProjectManagement(TestCase):
    @given(
        models(Project, collaborator_limit=integers(min_value=0, max_value=20)),
        lists(models(User), max_size=20))
    def test_can_add_users_up_to_collaborator_limit(self, project, collaborators):
        for c in collaborators:
            if project.at_collaboration_limit():
                with self.assertRaises(LimitReached):
                    project.add_user(c)
                self.assertFalse(project.team_contains(c))
            else:
                project.add_user(c)
                self.assertTrue(project.team_contains(c))
```

```
Falsifying example: test_can_add_users_up_to_collaborator_limit(
  self=TestProjectManagement(),
  project=Project('', 1),
  collaborators=[
    User(.@.com),
    User(.@.com)
  ]
)
Traceback (most recent call last):
...
    raise LimitReached()
manager.models.LimitReached
```

Property based testing is more complicated, yet more valuable

Django test
code coverage is

76%

Deceptive metric

High coverage

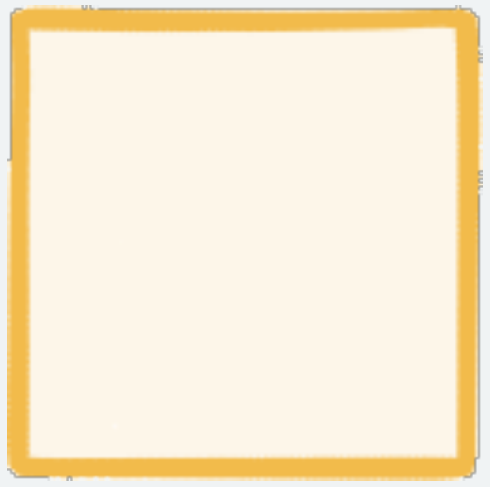
!=

high quality

Mutation testing

available Python implementation: mutpy

```
mut.py --target node --unit-test test_node
```



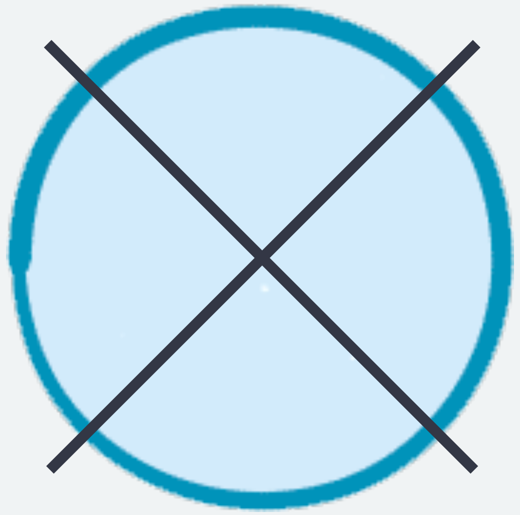
```
if foo and bar:
    do_this()
```

```python
if foo and bar:
    do_this()
```

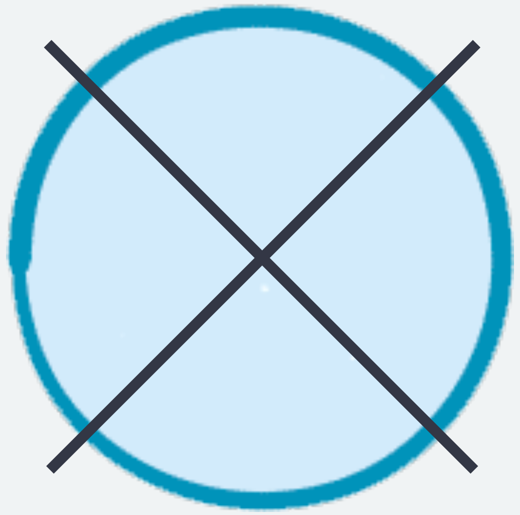
$\rightarrow$

```python
if foo or bar:
    do_this()
```

mutant
unit test



👍 killed 👍



mutant

👍 killed 👍

mutant

👎 survived 👎

AOR - arithmetic operator replacement

BCR - break continue replacement

COI - conditional operator insertion

CRP - constant replacement

DDL - decorator deletion

LOR - logical operator replacement

```
[*] Start mutation process:
   - targets: django.utils.encoding
   - tests: tests.utils_tests.test_encoding
[*] 10 tests passed:
   - tests.utils_tests.test_encoding [0.00533 s]
[*] Start mutants generation and execution:
...
[*] Mutation score [12.19066 s]: 32.1%
   - all: 88
   - killed: 24 (27.3%)
   - survived: 55 (62.5%)
   - incompetent: 7 (8.0%)
   - timeout: 2 (2.3%)
```

```
[*] Start mutation process:
  - targets: django.utils.encoding
  - tests: tests.utils_tests.test_encoding
[*] 10 tests passed:
  - tests.utils_tests.test_encoding [0.00533 s]
[*] Start mutants generation and execution:
...
[*] Mutation score [12.19066 s]: 32.1%
  - all: 88
  - killed: 24 (27.3%)
  - survived: 55 (62.5%)
  - incompetent: 7 (8.0%)
  - timeout: 2 (2.3%)
```

Django duration utils mutation score is

89%

Django duration utils mutation score is

89%

and the coverage is 91%

Django encoding utils mutation score is

32%

Django encoding utils mutation score is

32%

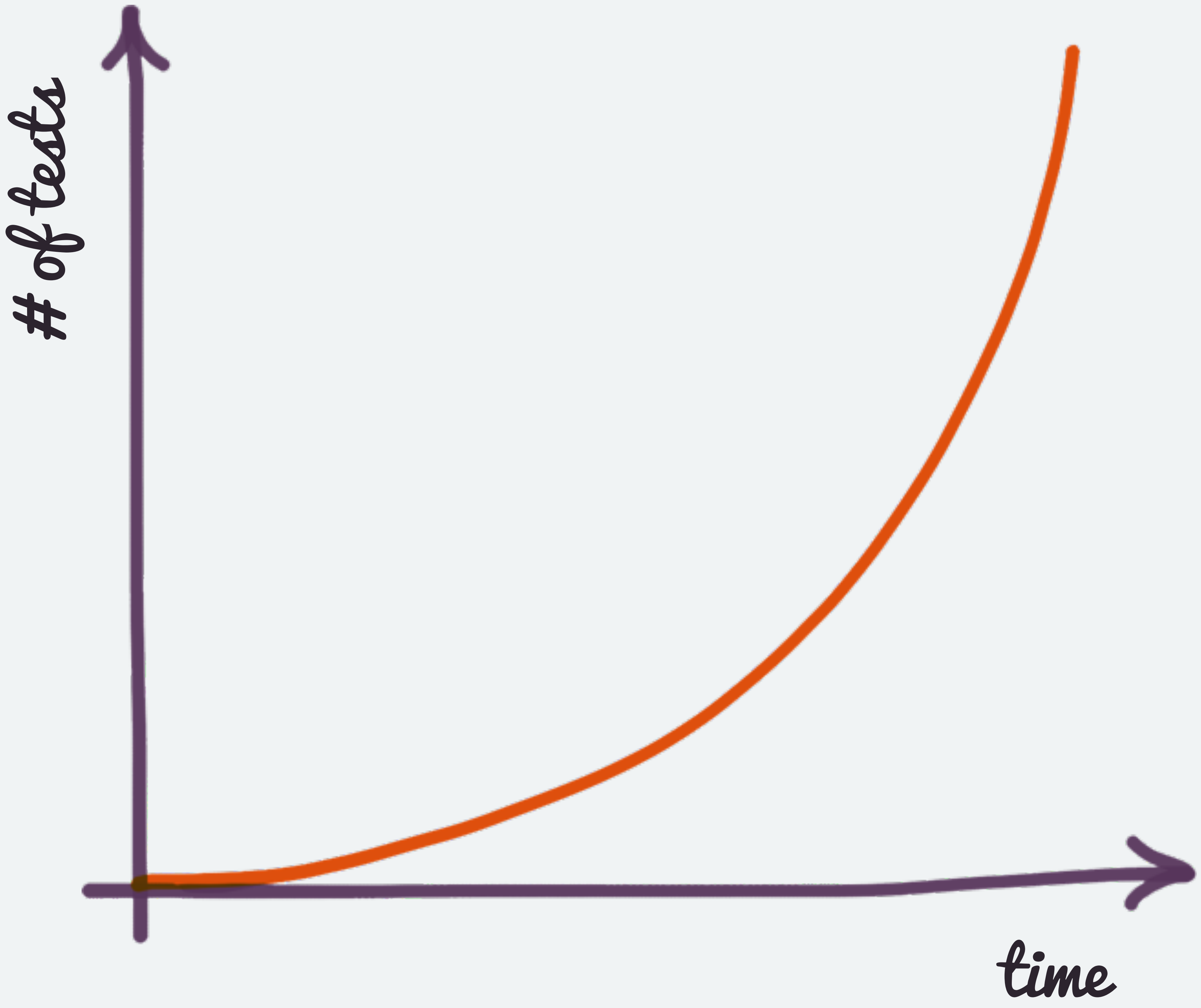
while the coverage is 63%

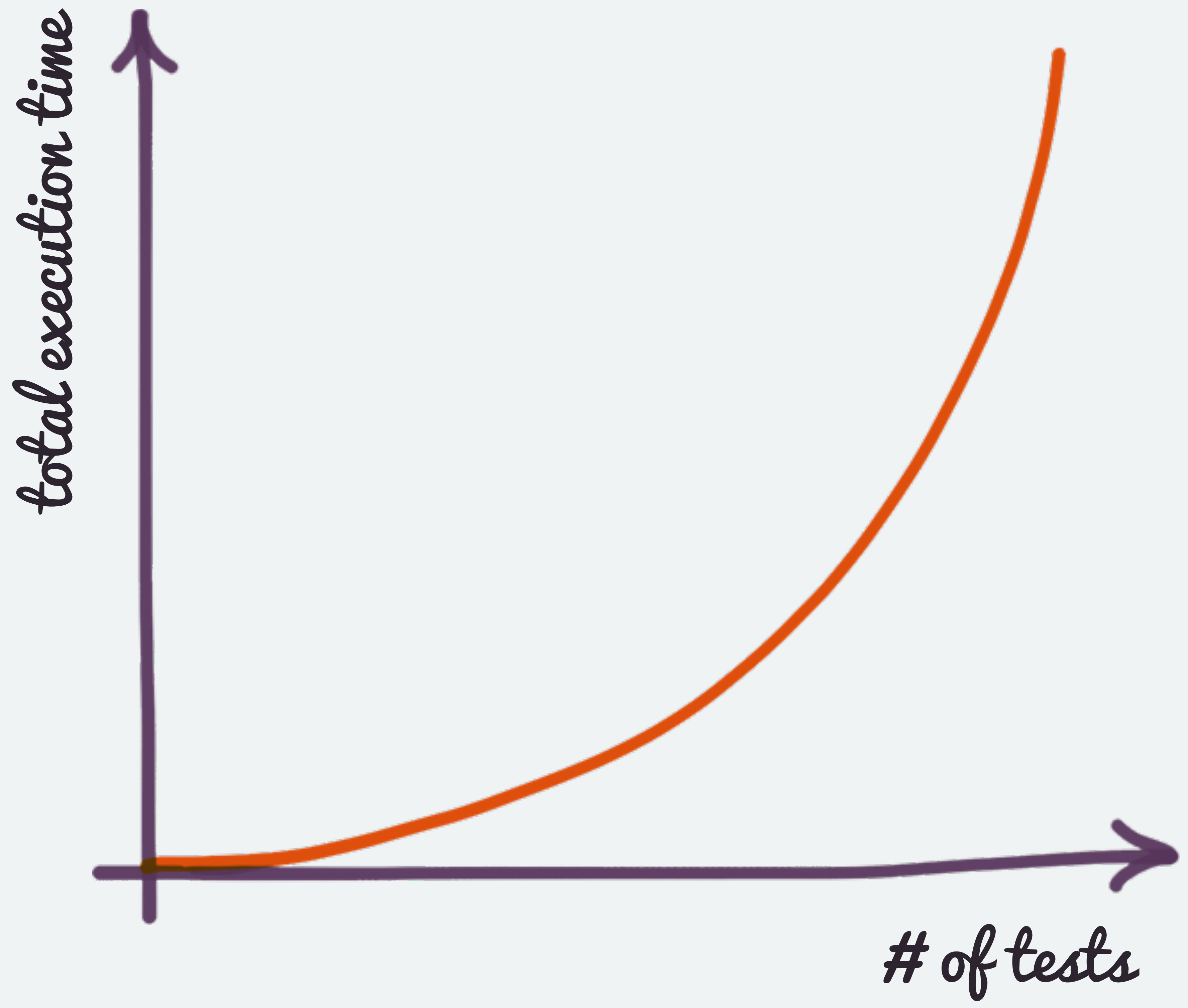
DJANGO TESTING TUTORIAL

```
my_app
├── __init__.py
├── admin.py
├── migrations
│   └── __init__.py
├── models.py
├── tests.py
└── views.py
```

```
my_app
├── __init__.py
├── admin.py
├── migrations
│   └── __init__.py
├── models.py
├── tests.py
└── views.py
```

When testing, more is better


# of tests
time


total execution time
# of tests

SLOW TESTS,
SLOW FEEDBACK LOOP

8 tips on how to speed up your tests

8 tips on how to speed up your tests

#5 will shock you

1. use MD5PasswordHasher

1. use MD5PasswordHasher
2. consider in-memory sqlite3

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()
5. use mocks EVERYWHERE

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()
5. ~~use mocks EVERYWHERE~~

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()
5. ~~use mocks EVERYWHERE~~
6. be vigilant of what gets created in setUp()

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()
5. ~~use mocks EVERYWHERE~~
6. be vigilant of what gets created in setUp()
7. don’t save model objects if not necessary

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()
5. ~~use mocks EVERYWHERE~~
6. be vigilant of what gets created in setUp()
7. don’t save model objects if not necessary
8. isolate unit tests

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()
5. ~~use mocks EVERYWHERE~~
6. be vigilant of what gets created in setUp()
7. don’t save model objects if not necessary
8. isolate unit tests

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()
5. ~~use mocks EVERYWHERE~~
6. be vigilant of what gets created in setUp()
7. don’t save model objects if not necessary
8. isolate unit tests

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()
5. ~~use mocks EVERYWHERE~~
6. be vigilant of what gets created in setUp()
7. don’t save model objects if not necessary
8. isolate unit tests

6. be vigilant of what gets created in setUp()

```
class SimpleTest(TestCase):
    def setUp(self):
        for _ in range(10):
            Robot.objects.create()
```

6. be vigilant of what gets created in setUp()

```
class SimpleTest(TestCase):
    def setUp(self):
        for _ in range(10):
            Robot.objects.create()
```

7. don’t save model objects if not necessary

7. don’t save model objects if not necessary

instead of

```
Robot.objects.create()
```

7. don't save model objects if not necessary

instead of `Robot.objects.create()`

maybe do `Robot()`

7. don’t save model objects if not necessary

instead of `Robot.objects.create()`

maybe do `Robot()`

or `RobotFactory.build()`

7. don't save model objects if not necessary

instead of	`Robot.objects.create()`
maybe do	`Robot()`
or	`RobotFactory.build()`
or	`RobotFactory.stub()`

7. don't save model objects if not necessary

instead of `Robot.objects.create()`

maybe do `Robot()`

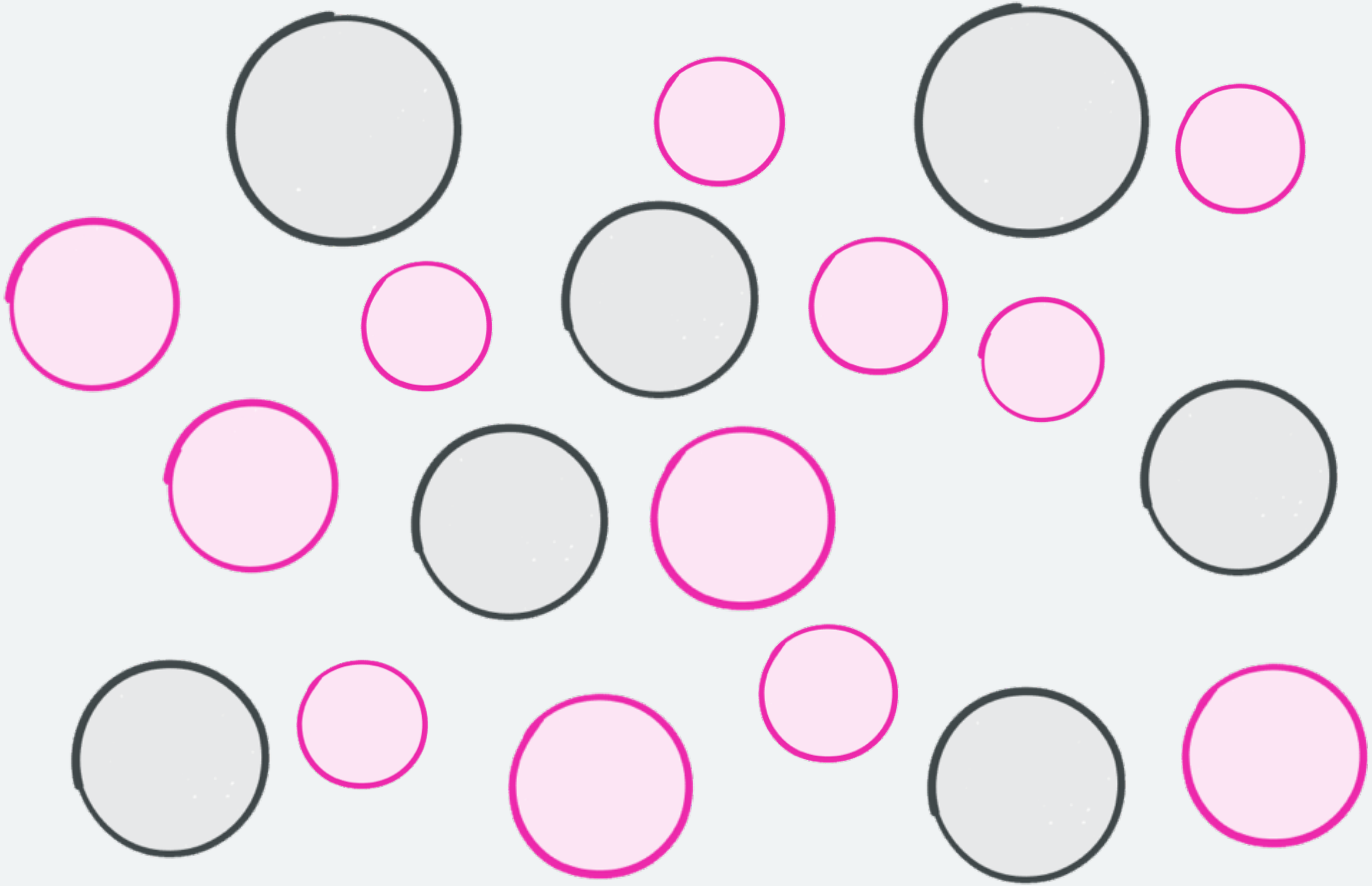
or `RobotFactory.build()`

or `RobotFactory.stub()`

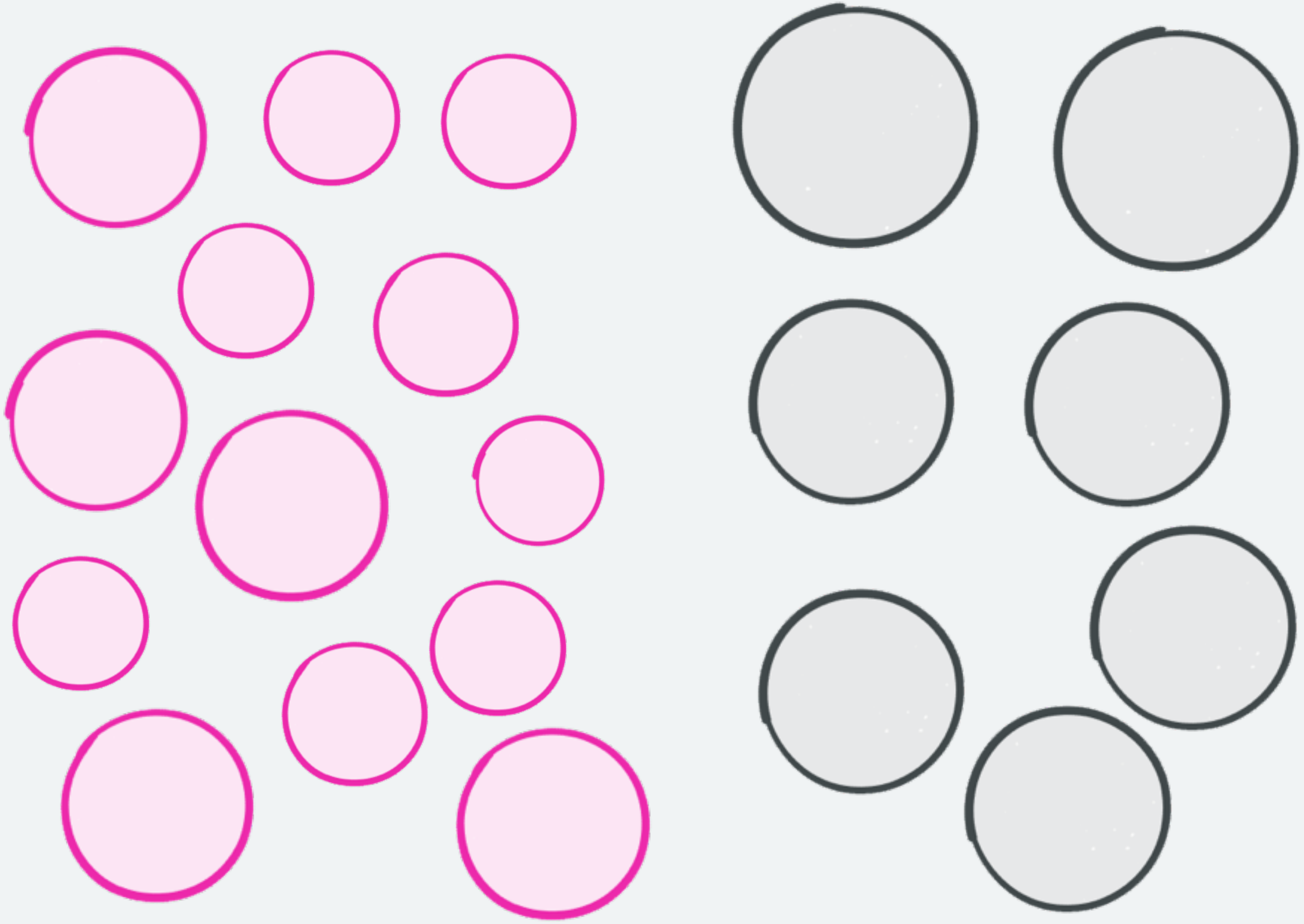
$\left.\begin{array}{l}\texttt{RobotFactory.build()} \\ \texttt{RobotFactory.stub()}\end{array}\right\}$ factory boy

8. isolate unit tests

8. isolate unit tests



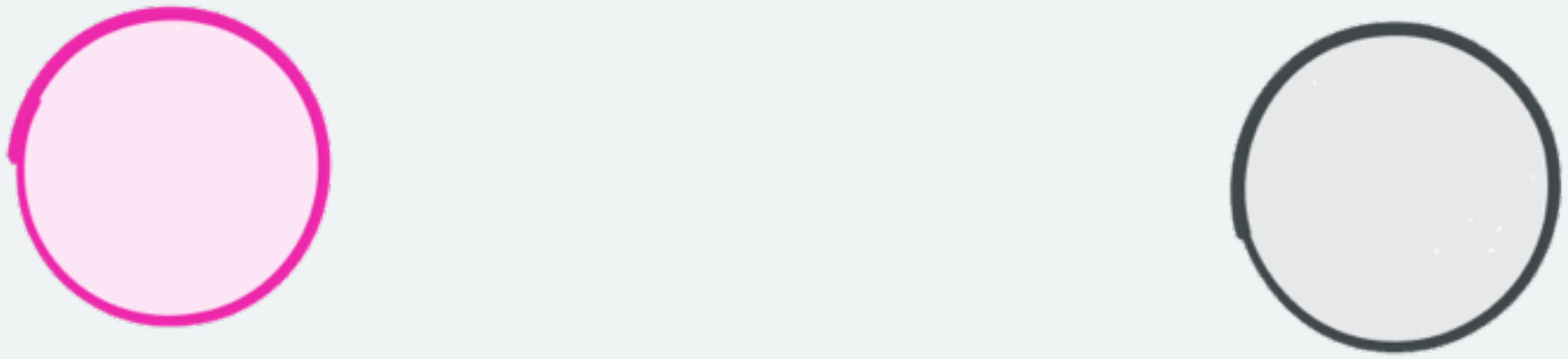
8. isolate unit tests



8. isolate unit tests

unit tests

functional tests



8. isolate unit tests

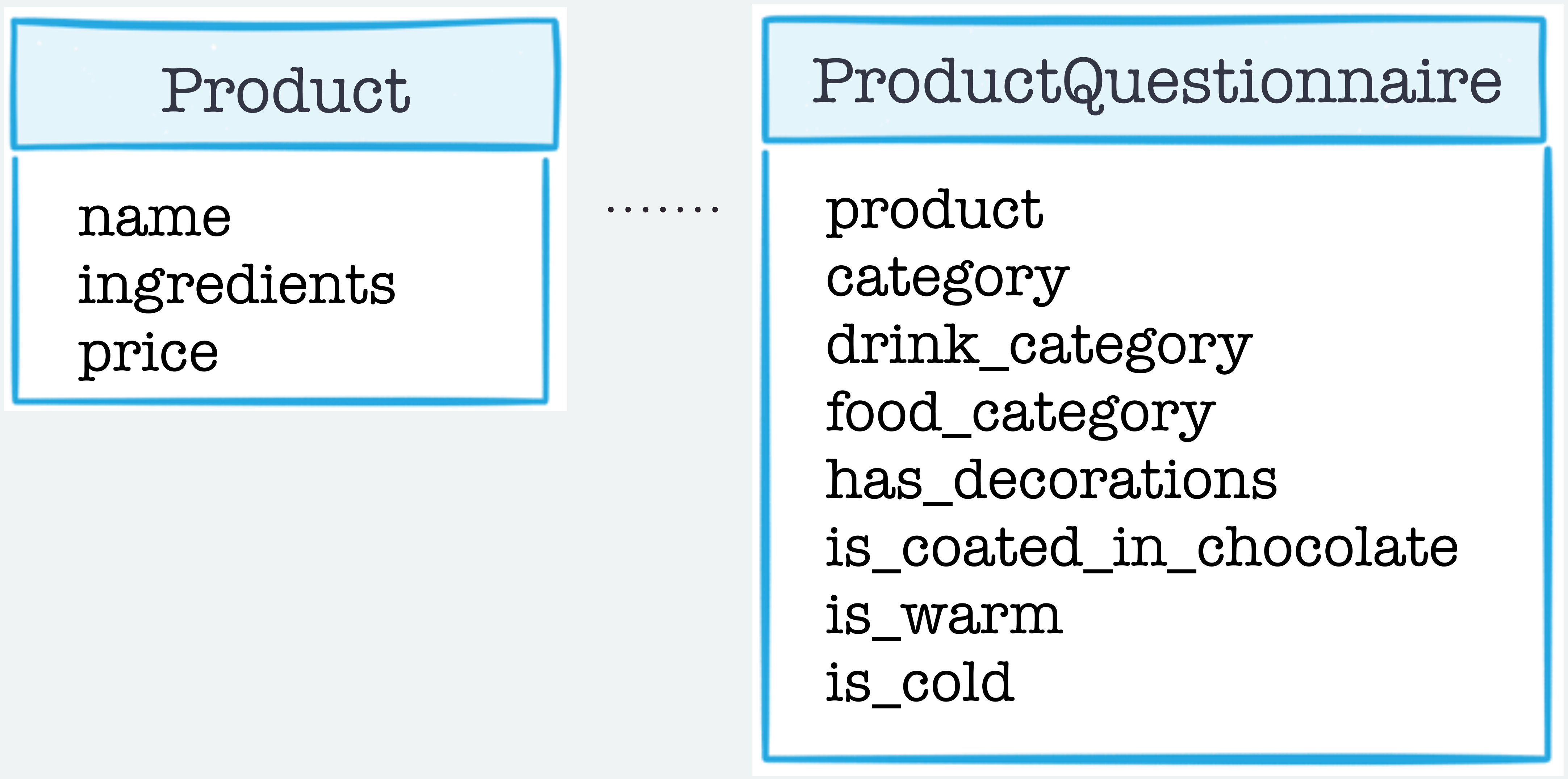
unit tests

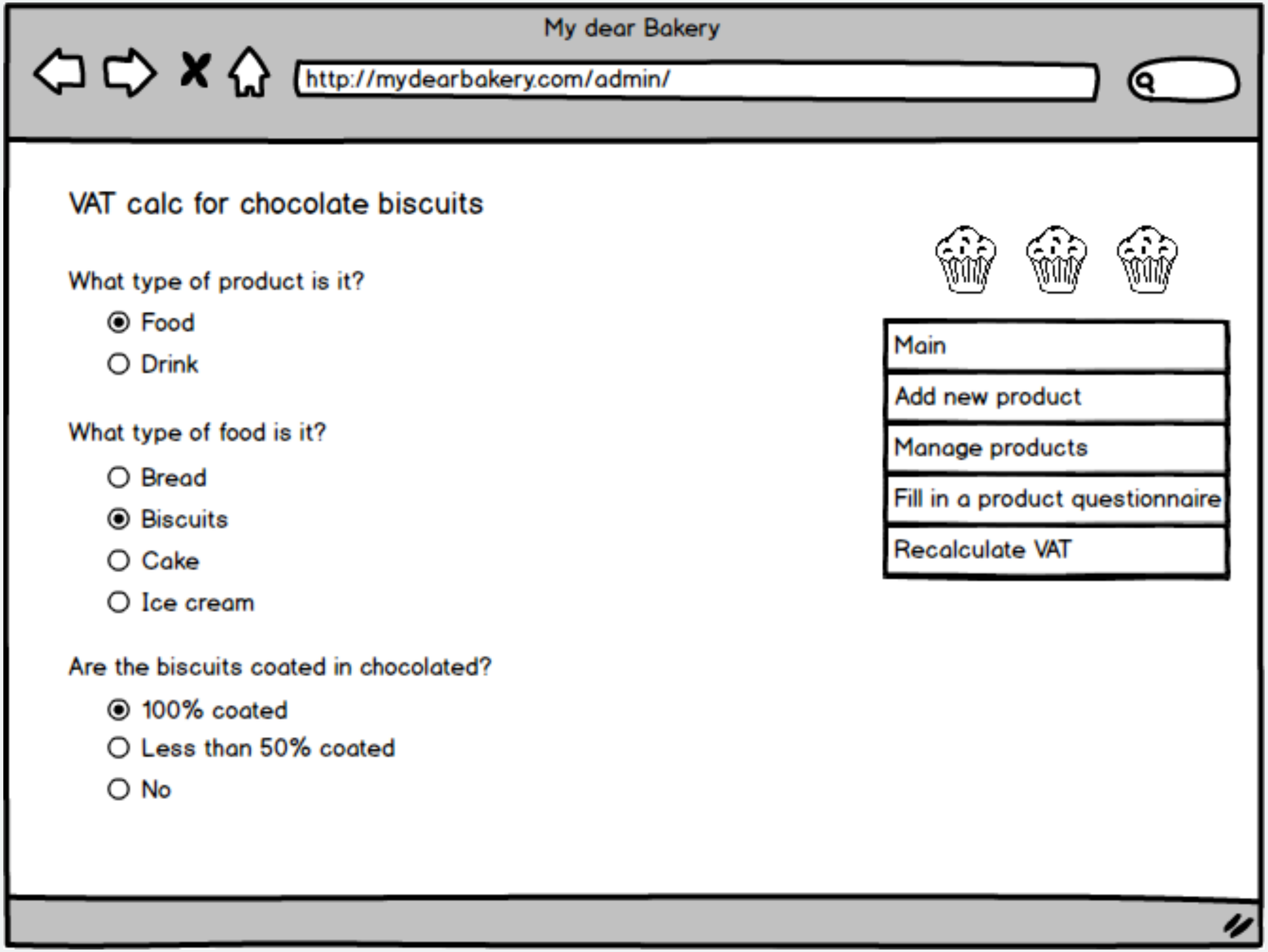
functional tests

```
./manage.py test --tag=unit
```

My dear
Bakery

Ugh,
taxes!


Product
name
ingredients
price
ProductQuestionnaire
product
category
drink_category
food_category
has_decorations
is_coated_in_chocolate
is_warm
is_cold



My dear Bakery

http://mydearbakery.com/admin/

VAT calc for chocolate biscuits

What type of product is it?

- [✓ Food]
- Drink

What type of food is it?

- Bread
- [✓ Biscuits]
- Cake
- Ice cream

Are the biscuits coated in chocolated?

- [✓ 100% coated]
- Less than 50% coated
- No

Production code

Test

take 1

```
class ProductQuestionnaireCreate(CreateView):
    def form_valid(self, form):
        if is_biscuit and is_coated_in_chocolate:
            set_vat_20()
        return super().form_valid(form)
```

```
class ProductQuestionnaireCreateTestCase(TestCase):
    def test_20p_vat_if_coated_in_chocolate_biscuit(self):
        product = ProductFactory()
        response = self.client.post(self.url, {'q1': 'a1', 'q2': 'a2'})
        product.refresh_from_db()
        self.assertEqual(product.vat, 20)
```

take 1

```
class ProductQuestionnaireCreate(CreateView):
    def form_valid(self, form):
        if is_biscuit and is_coated_in_chocolate:
            set_vat_20()
        return super().form_valid(form)
```

```
class ProductQuestionnaireCreateTestCase(TestCase):
    def test_20p_vat_if_coated_in_chocolate_biscuit(self):
        product = ProductFactory()
        response = self.client.post(self.url, {'q1': 'a1', 'q2': 'a2'})
        product.refresh_from_db()
        self.assertEqual(product.vat, 20)
```

```
class ProductQuestionnaireCreate(CreateView):
    def form_valid(self, form):







        return super().form_valid(form)
```

```
class ProductQuestionnaireCreateTestCase(TestCase):
    def test_20p_vat_if_coated_in_chocolate_biscuit(self):


    def test_0p_vat_if_baguette(self):


    def test_0p_vat_if_flapjack(self):


    def test_20p_vat_if_cereal_bar(self):


```

To test if I need to pay 20% VAT for biscuits coated in chocolate, I need to:

- go through the router
- interact with database
- send input to receive output

Mocks are not a solution

take 2

```
class ProductQuestionnaireForm(forms.ModelForm):
    def save(self, commit=True):
        instance = super().save(commit)
        if is_biscuit and is_coated_in_chocolate:
            set_vat_20()
        return instance
```

```
class ProductQuestionnaireFormTestCase(TestCase):
    def test_20p_vat_if_coated_in_chocolate_biscuit(self):
        product = ProductFactory()
        form = ProductQuestionnaireForm(data={'k1': 'v1', 'k2': 'v2'})
        self.assertTrue(form.is_valid())
        form.save()
        product.refresh_from_db()
        self.assertEqual(product.vat, 20)
```

take 2

```
class ProductQuestionnaireForm(forms.ModelForm):
    def save(self, commit=True):
        instance = super().save(commit)
        if is_biscuit and is_coated_in_chocolate:
            set_vat_20()
        return instance
```

```
class ProductQuestionnaireFormTestCase(TestCase):
    def test_20p_vat_if_coated_in_chocolate_biscuit(self):
        product = ProductFactory()
        form = ProductQuestionnaireForm(data={'k1': 'v1', 'k2': 'v2'})
        self.assertTrue(form.is_valid())
        form.save()
        product.refresh_from_db()
        self.assertEqual(product.vat, 20)
```

To test if I need to pay 20% VAT for biscuits coated in chocolate, I need to:

- ~~go through the router~~
- interact with database
- send input to receive output

take 3

```
class VATCalculator(object):
    def calculate_vat(self, **kwargs):
        if is_biscuit and is_coated_in_chocolate:
            return 20
```

....

```
class VATCalculatorTestCase(SimpleTestCase):
    def test_20p_vat_if_coated_in_chocolate_biscuit(self):
        calc = VATCalculator()
        self.assertEqual(calc.calculate_vat(
                is_biscuit=True, is_coated_in_choco=True
        ))
```

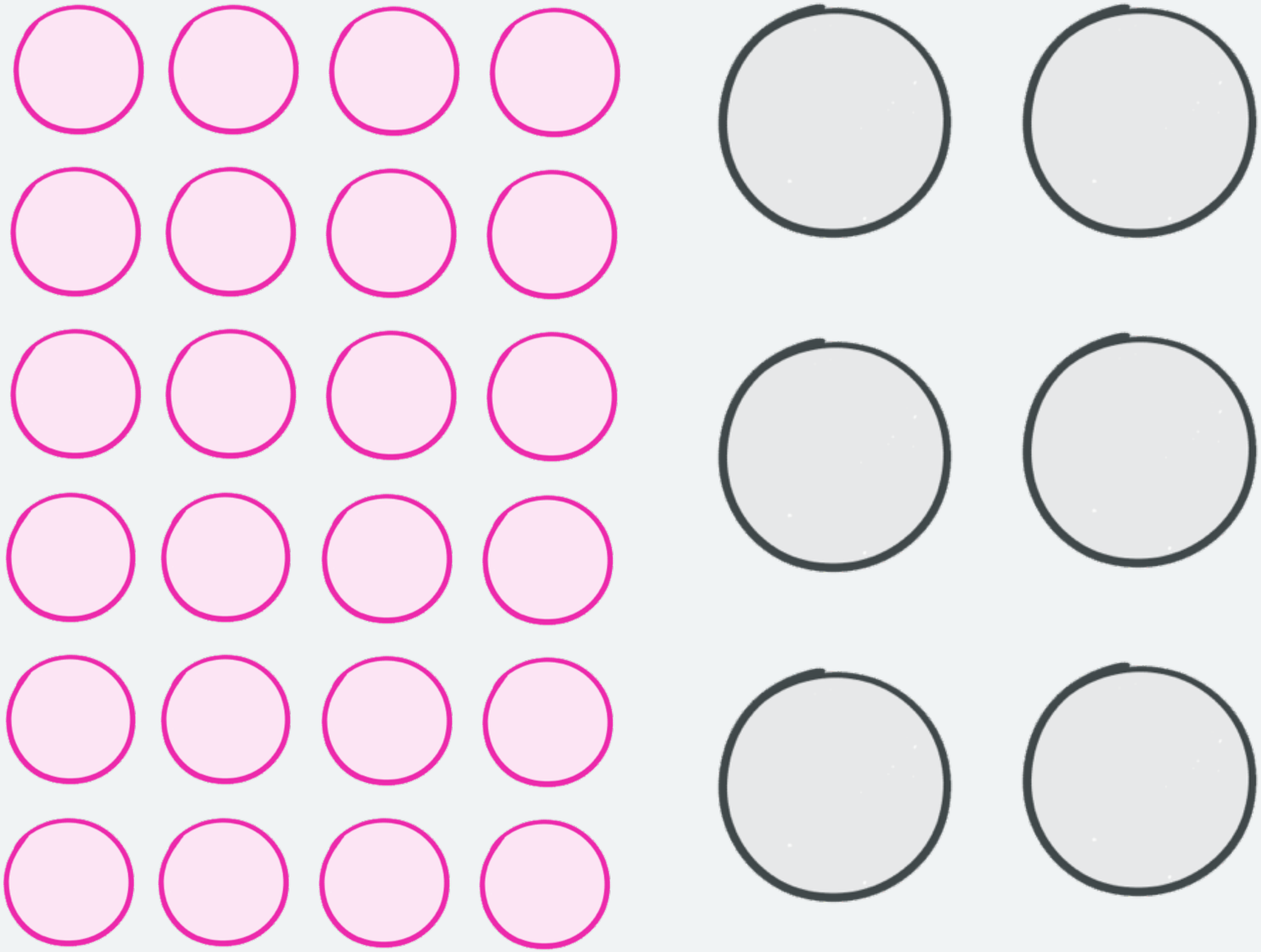
REUSABILITY

EXTENSIBILITY

TESTABILITY

To test if I need to pay 20% VAT for biscuits coated in chocolate, I need to:

- ~~go through the router~~
- ~~interact with database~~
- send input to receive output



Tests have more in them than we think

HAPPY
TESTING

and big bear hug thanks!