

TESTING IN DJANGO

by Ana Balica

2017
djangocon.us
spokane





@anabalica



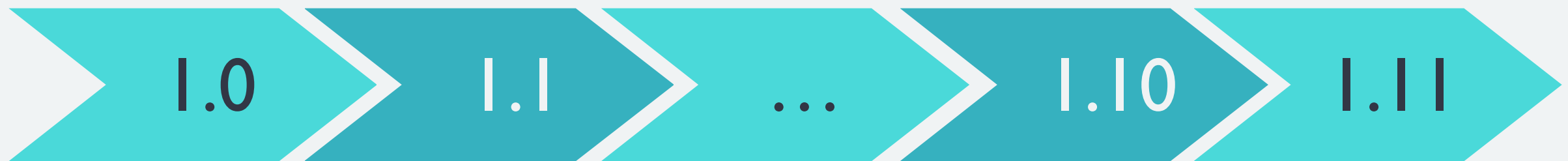
StreetTeam

QUESTIONS?

DJANGO ARCHEOLOGY



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



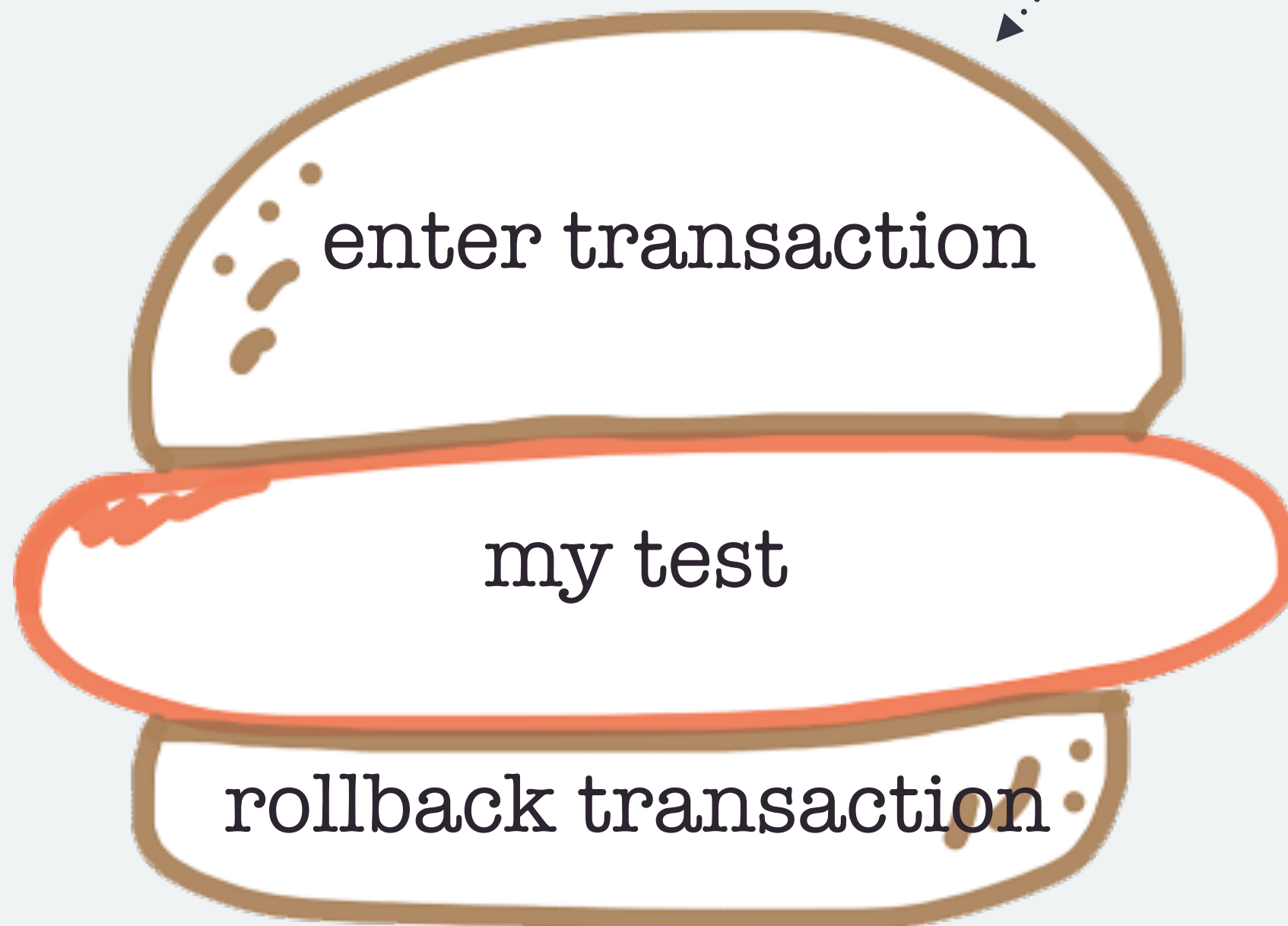
CLIENT

put
head
delete
options

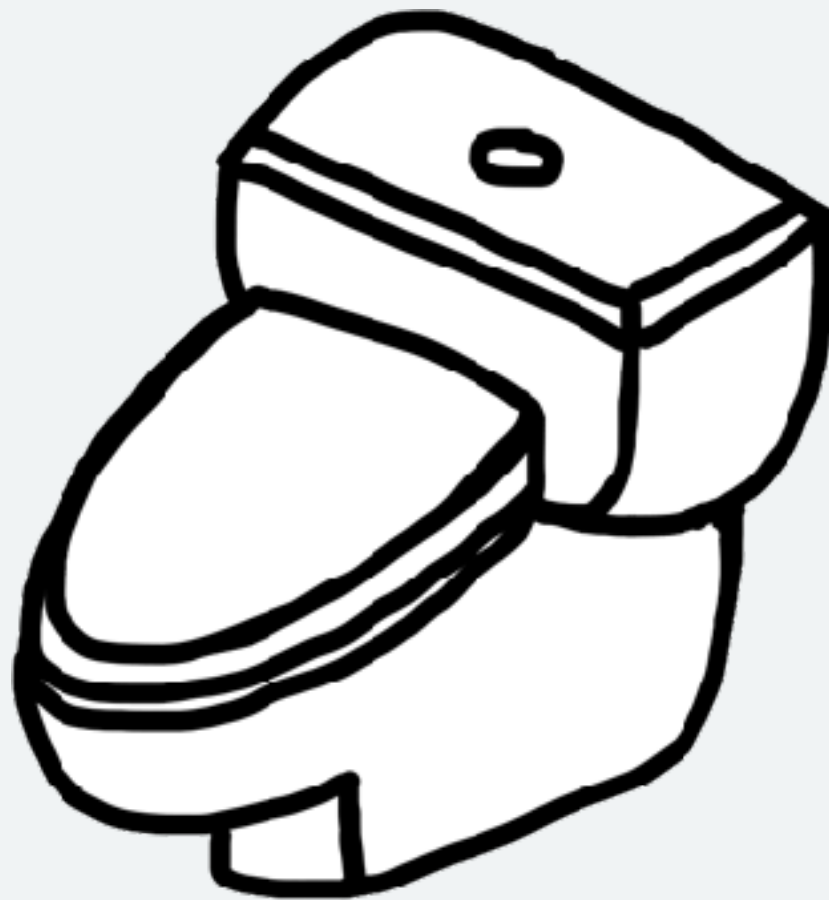
TransactionTestCase

TestCase

I am a **TestCase** burger



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





1.2

DjangoTestSuiteRunner

from function to class

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

.....

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

.....

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



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



failure



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

42

 failure 

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

256



success



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

```
failures = TestRunner(verbosity, interactive, failfast)
```

```
if failures:
```

```
    sys.exit(1)
```

.....

0

success

/

1

failure

```
failures = TestRunner(verbosity, interactive, failfast)
```

```
if failures:
```

```
    sys.exit(1)
```

256

 failure 

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

256

 failure 

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

256

 failure 

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

256

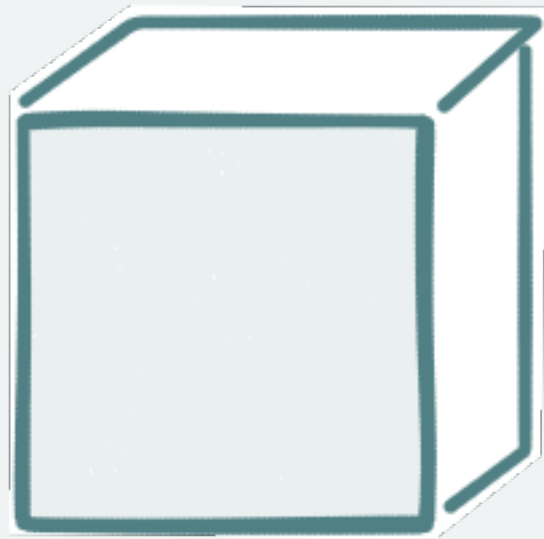
 failure 

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

256

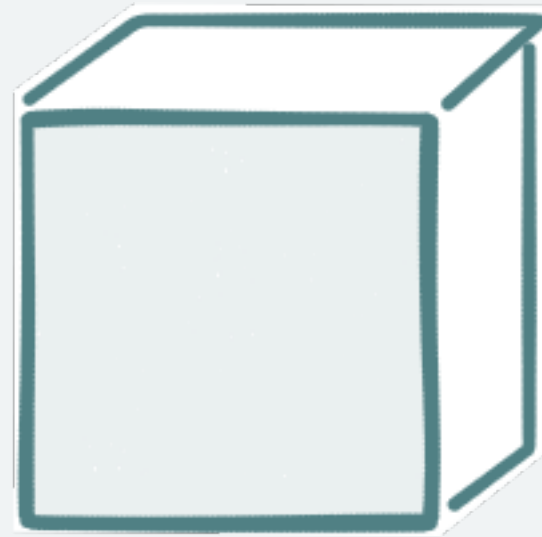
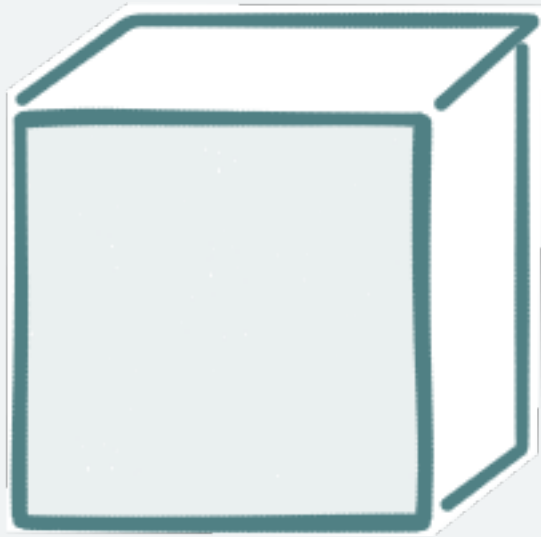
 failure 

MULTIPLE DATABASES

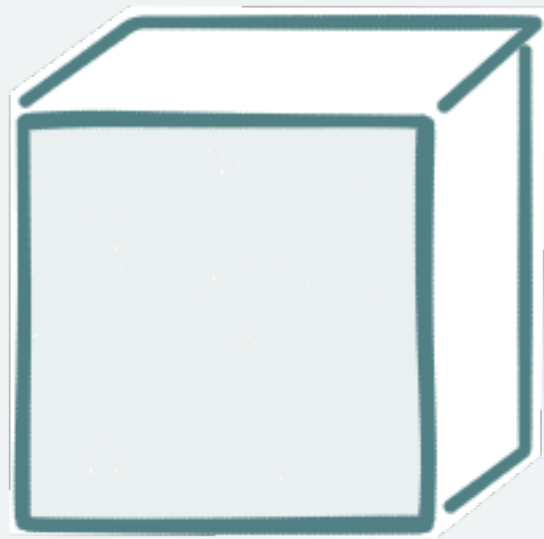


1.2

MULTIPLE DATABASES

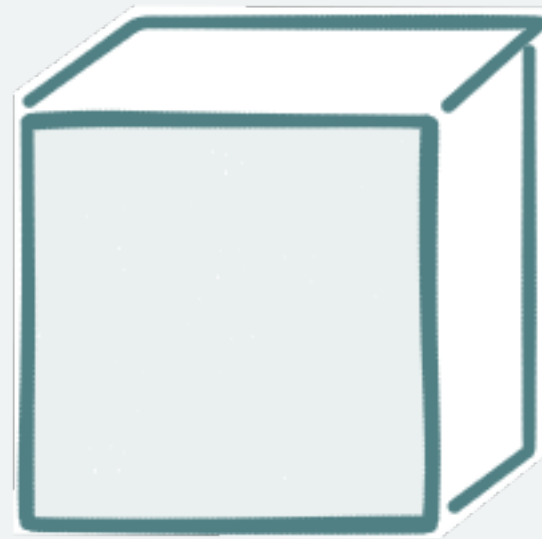


MULTIPLE DATABASES



primary

.....



replica

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  
    }  
}
```

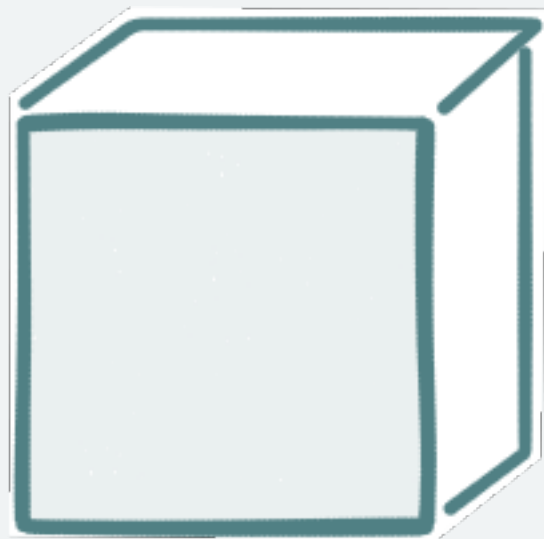
MULTIPLE DATABASES

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

MULTIPLE DATABASES

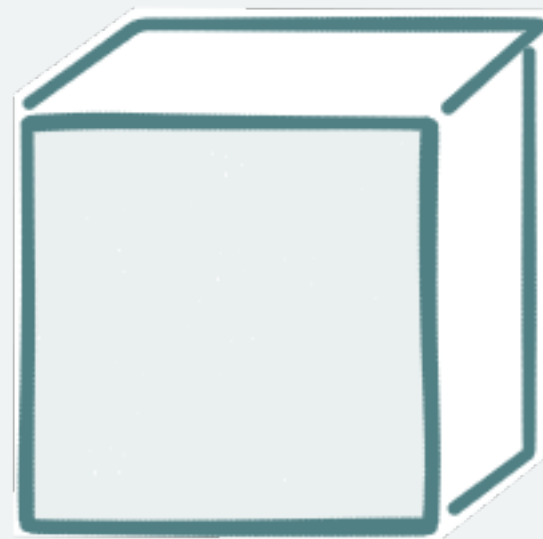
```
DATABASES = {  
    'default': {  
        'HOST': 'dbprimary',  
        # ... plus other settings  
    },  
    'replica': {  
        'HOST': 'dbreplica',  
        'TEST': {  
            'MIRROR': 'default',  
        },  
        # ... plus other settings  
    }  
}
```

MULTIPLE DATABASES



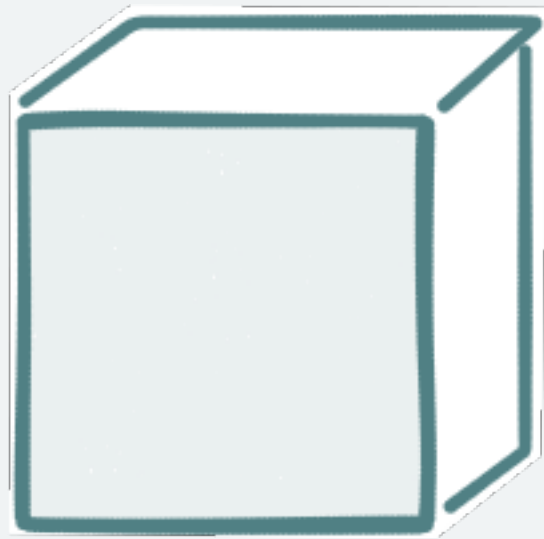
primary

.....



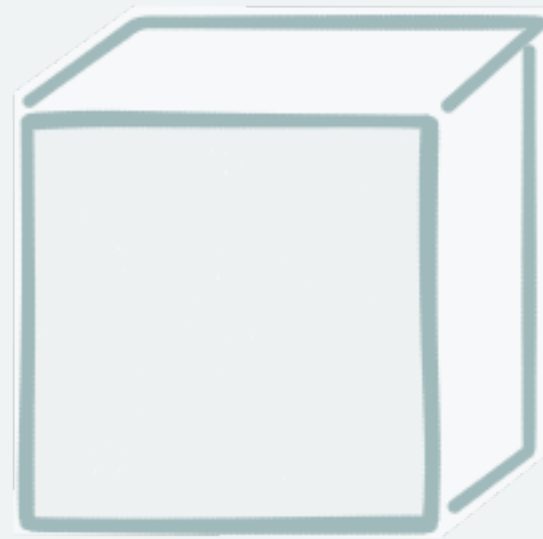
replica

MULTIPLE DATABASES



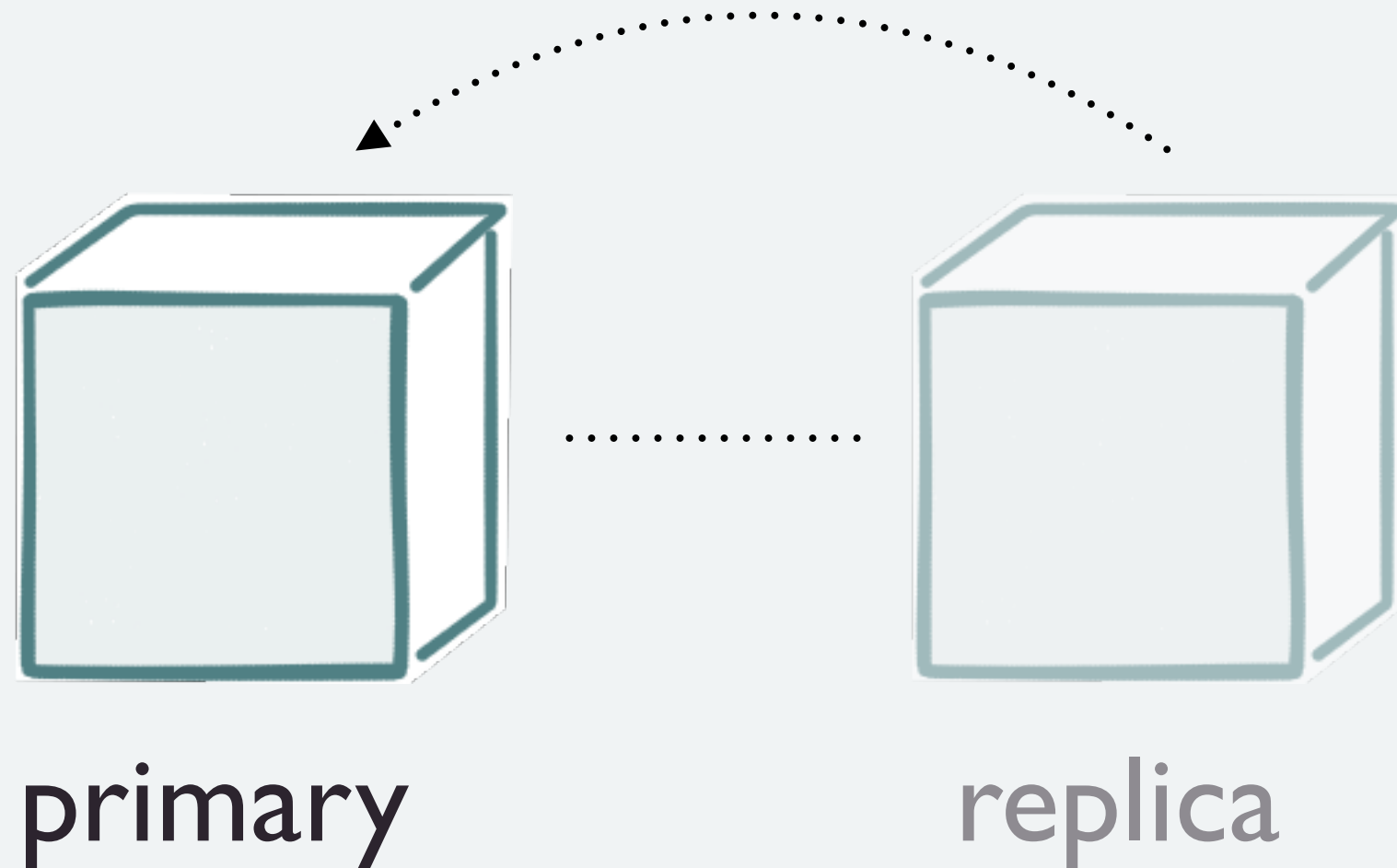
primary

.....



replica

MULTIPLE DATABASES





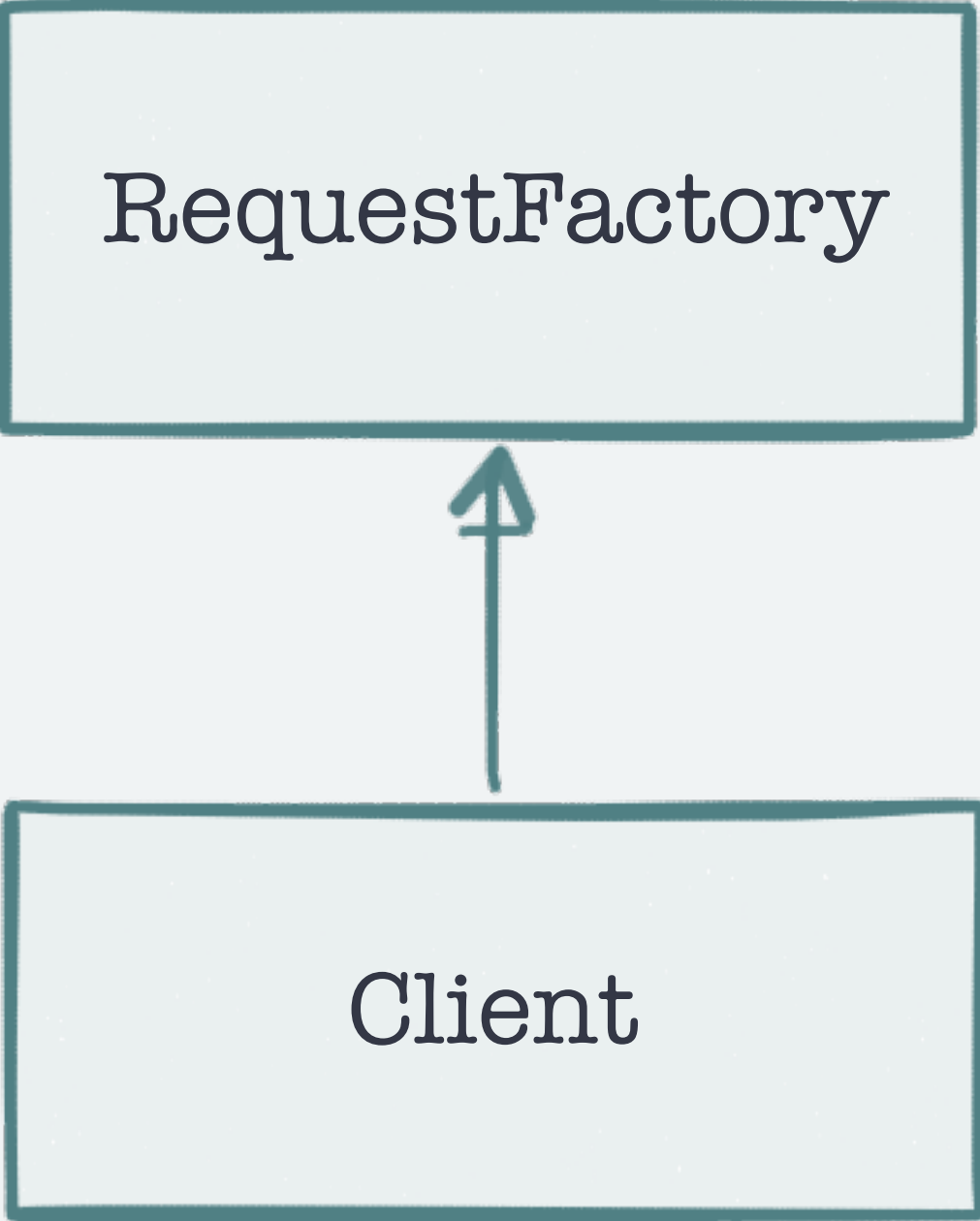
1.3

assert*

QuerysetEqual

NumQueries

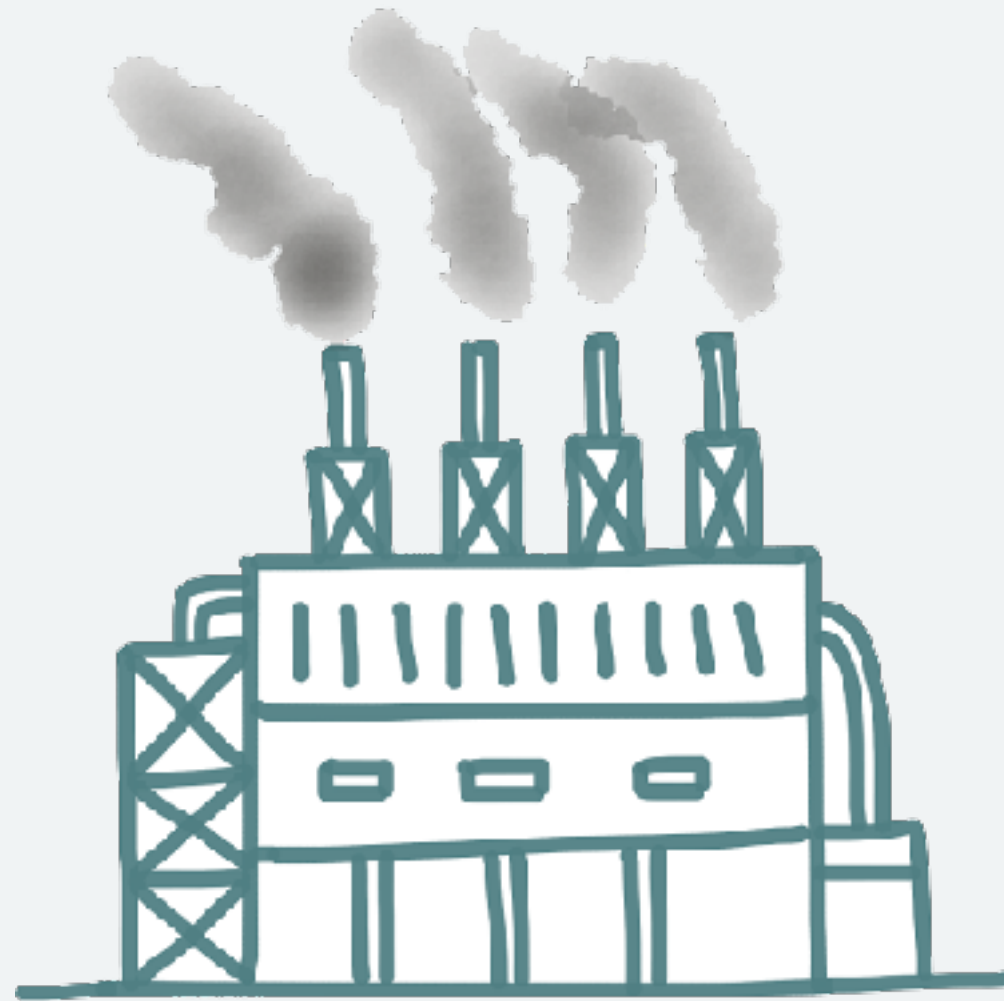
TESTCASE



```
graph BT; Client --> RequestFactory
```

RequestFactory

Client



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

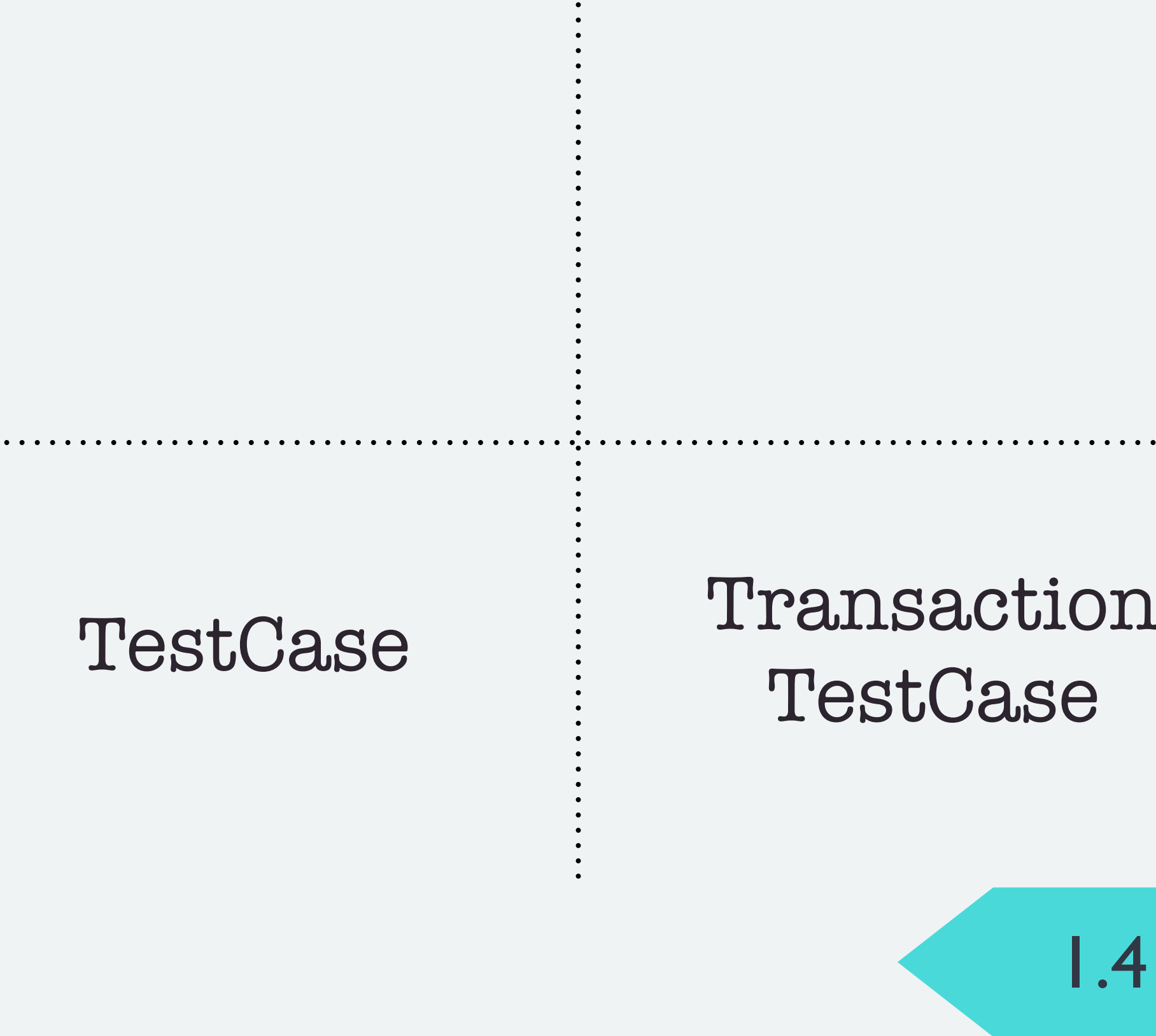
doctests
=
tests + documentation

doctests
=

tests + documentation



1.4



TestCase

Transaction
TestCase

1.4

**Simple
TestCase**

**LiveServer
TestCase**

TestCase

Transaction
TestCase

*doesn't hit
the database*

**Simple
TestCase**

**LiveServer
TestCase**

TestCase

Transaction
TestCase

*doesn't hit
the database*

**Simple
TestCase**

*runs an
http server*

**LiveServer
TestCase**

TestCase

Transaction
TestCase

*doesn't hit
the database*

**Simple
TestCase**

*runs an
http server*

**LiveServer
TestCase**

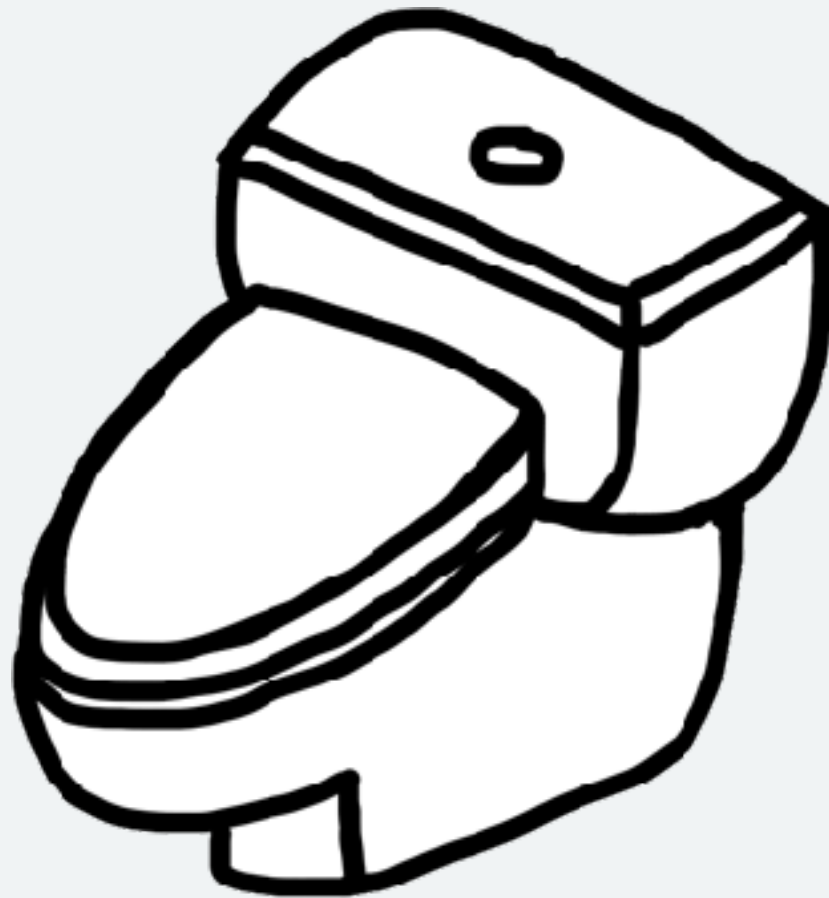
TestCase

Transaction
TestCase

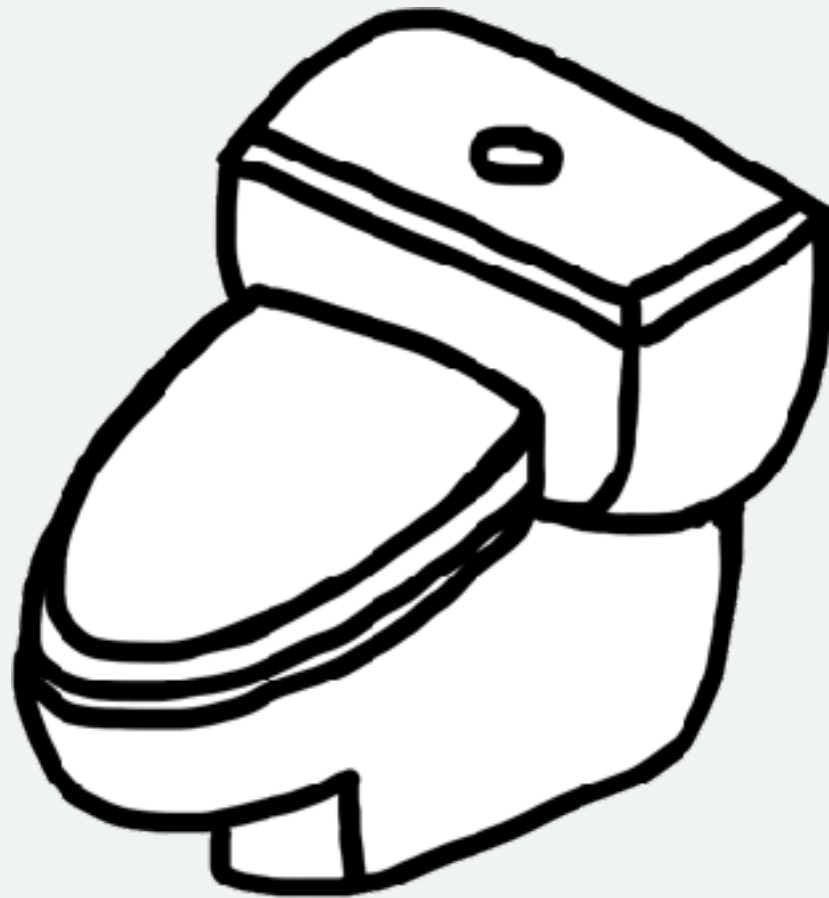


1.5

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



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





flush

run TransactionTestCase

enter transaction

run TestCase

rollback transaction





- flush
- run TransactionTestCase
- enter transaction
- run TestCase
- rollback transaction



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



1.6

CLIENT

patch

1.6

IMPROVEMENTS

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

IMPROVEMENTS

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

IMPROVEMENTS

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



1.7

unittest2

1.7

~~unittest2~~

unittest

LiveServerTestCase

StaticLiveServerTestCase



1.8

CLIENT

trace

1.8

TestCase

before

enter atomic

load fixtures

...

exit atomic

close connections

TestCase

before

enter atomic
load fixtures

...

exit atomic
close connections

TestCase

before

enter atomic
load fixtures

...

exit atomic
close connections

times # of tests

TestCase

after

enter atomic

load fixtures

enter atomic

...

exit atomic

exit atomic

close connections

TestCase

after

enter atomic

load fixtures

enter atomic

...

exit atomic

exit atomic

close connections

once

TestCase

after

once

enter atomic
load fixtures

enter atomic

...

exit atomic

exit atomic

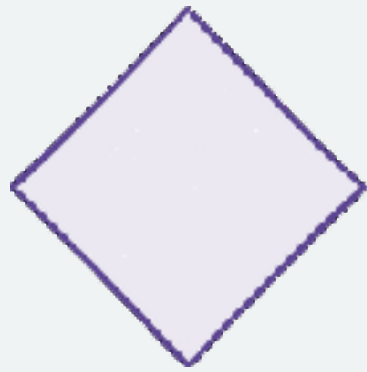
close connections

times # of tests



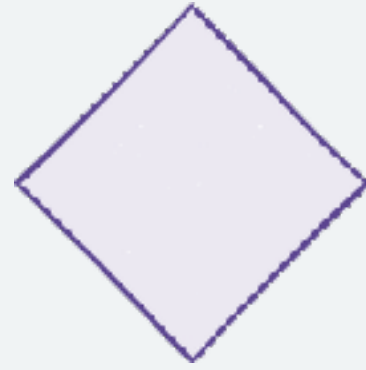
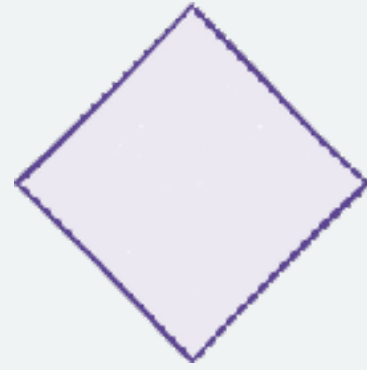
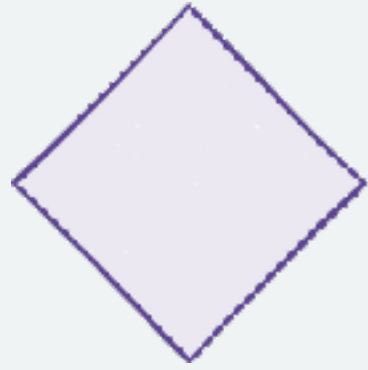
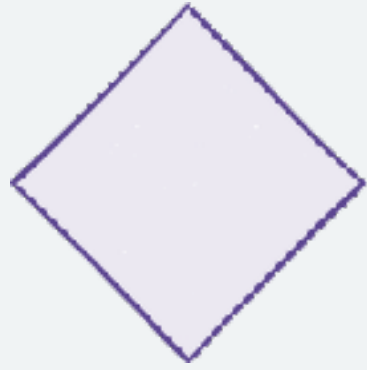
1.9

--parallel



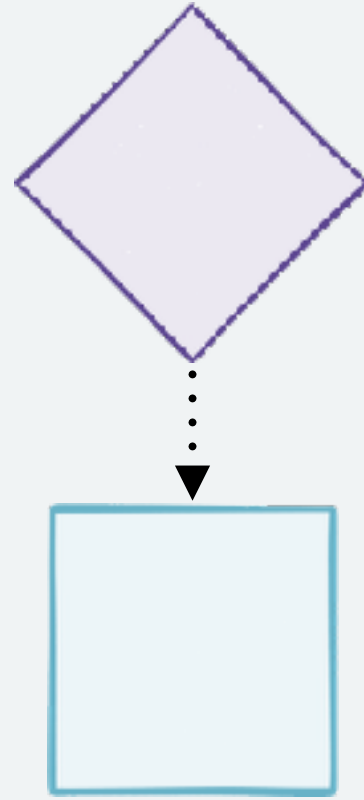
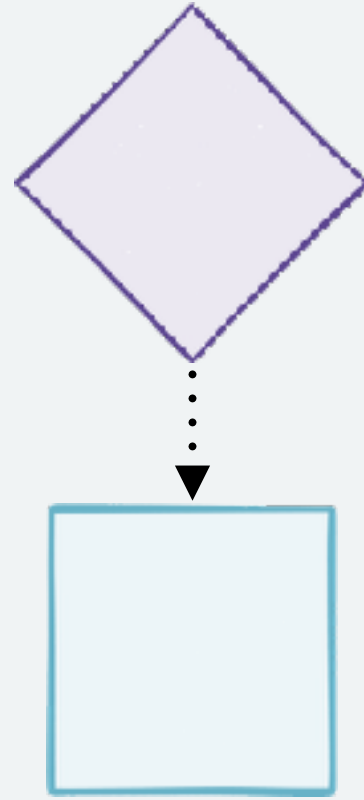
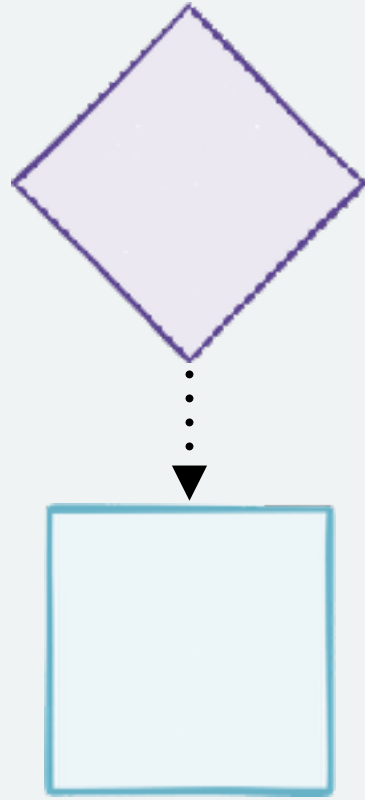
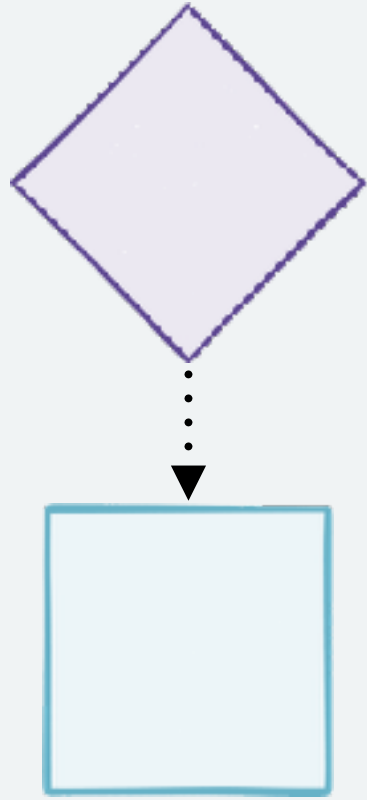
1.9

workers

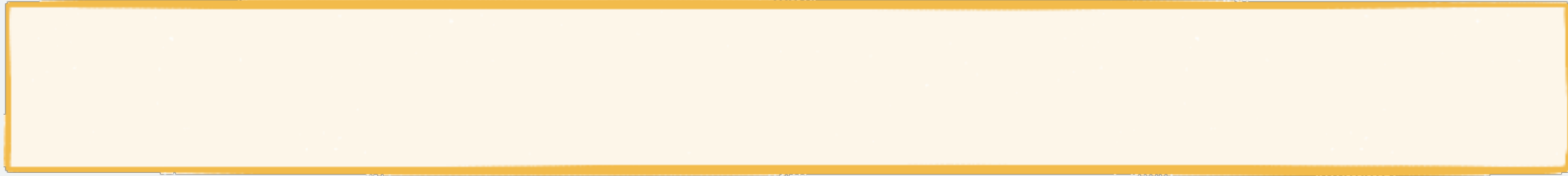
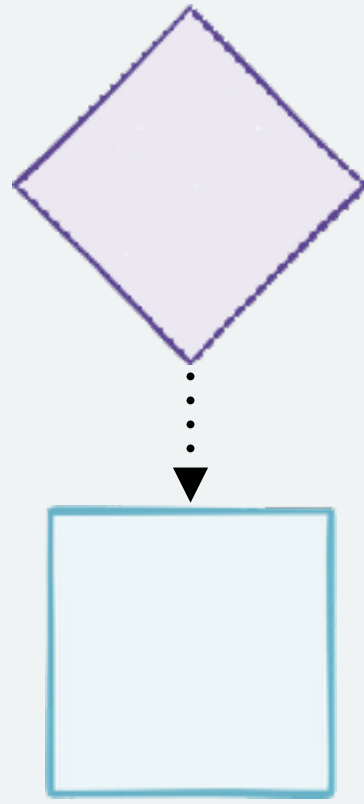
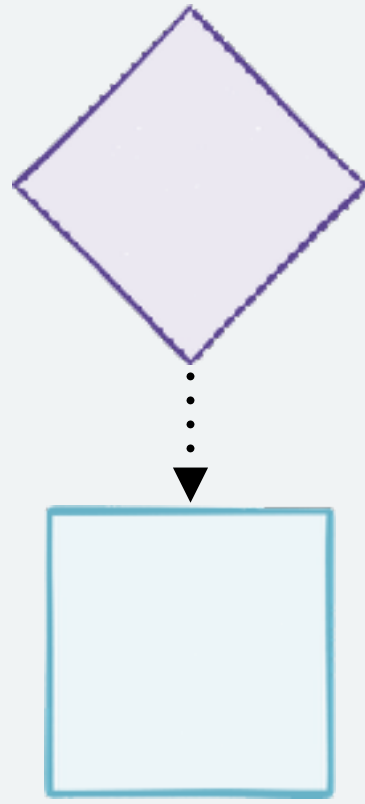
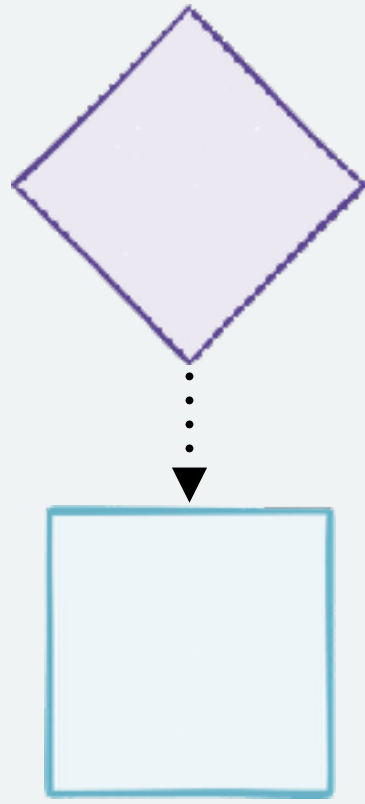
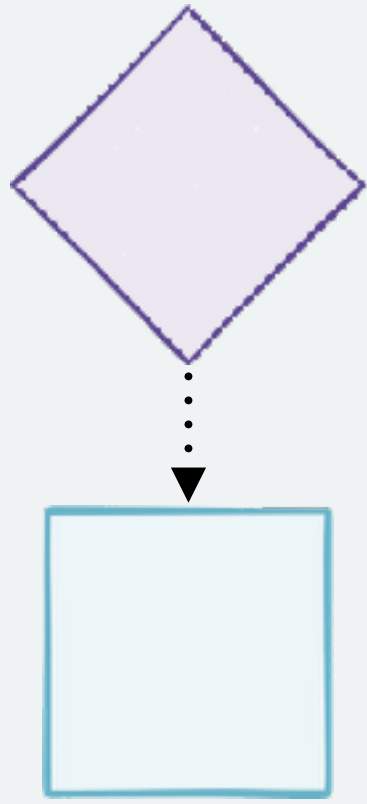


1.9

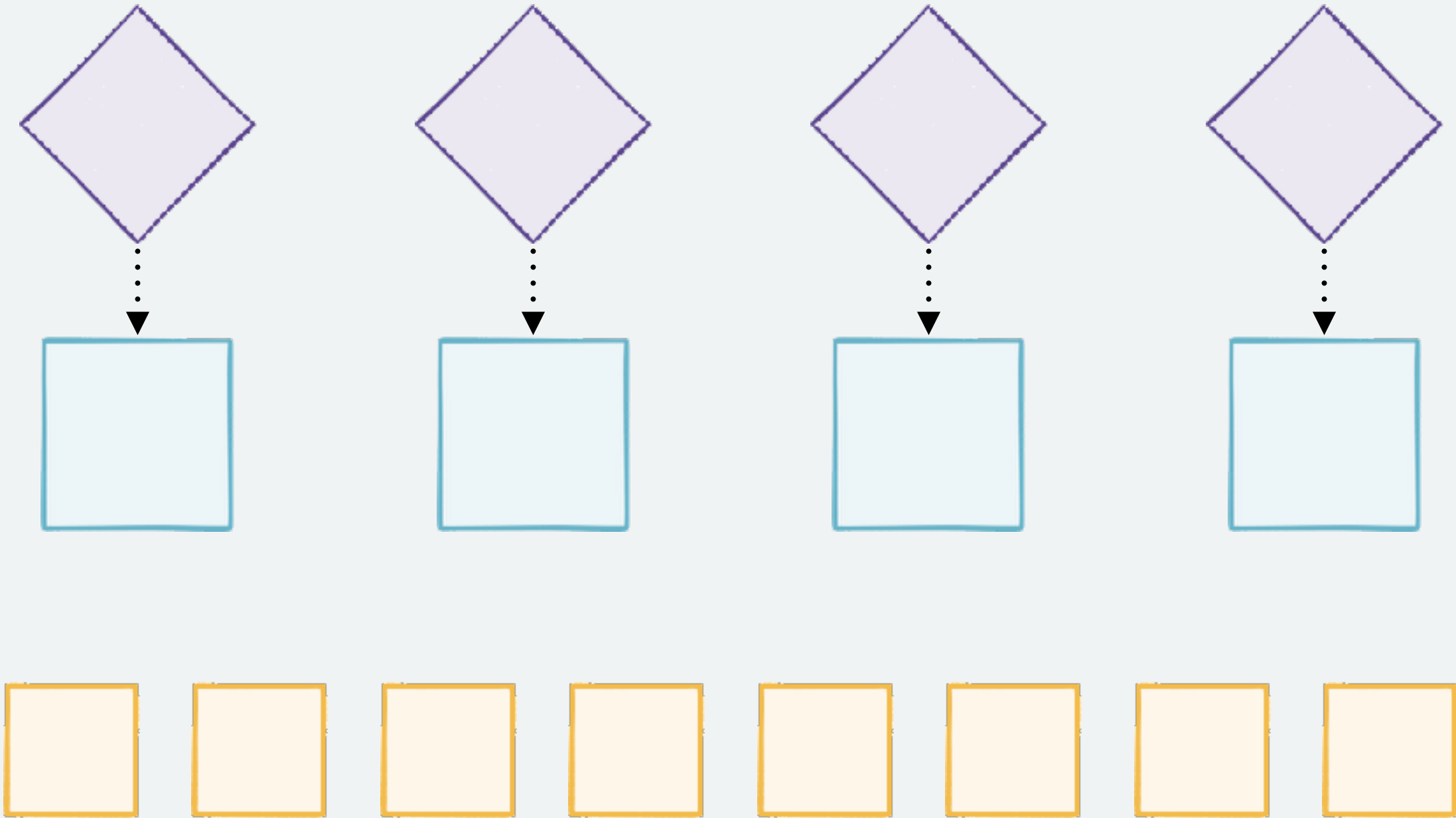
databases workers



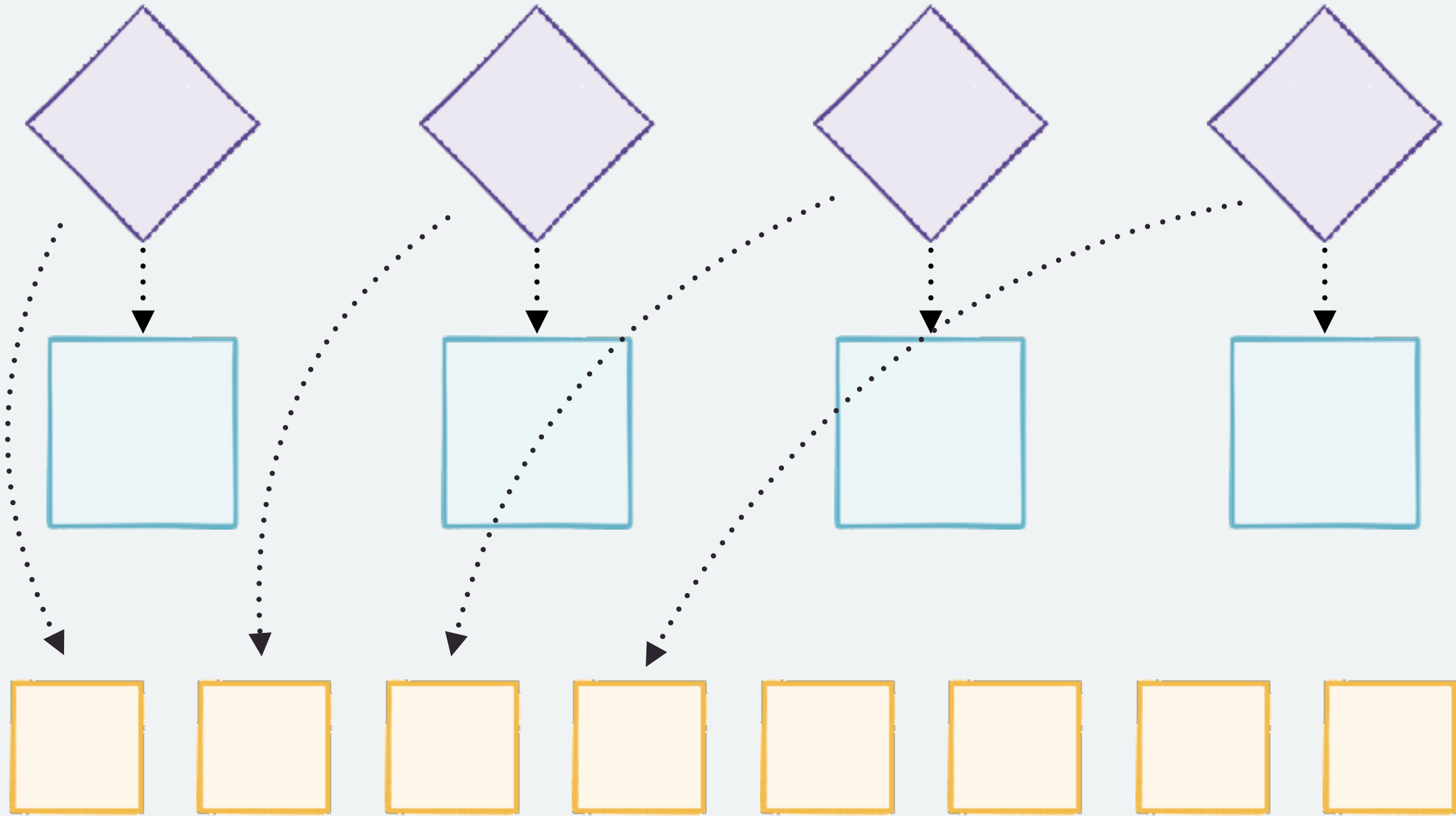
workers
databases
suite



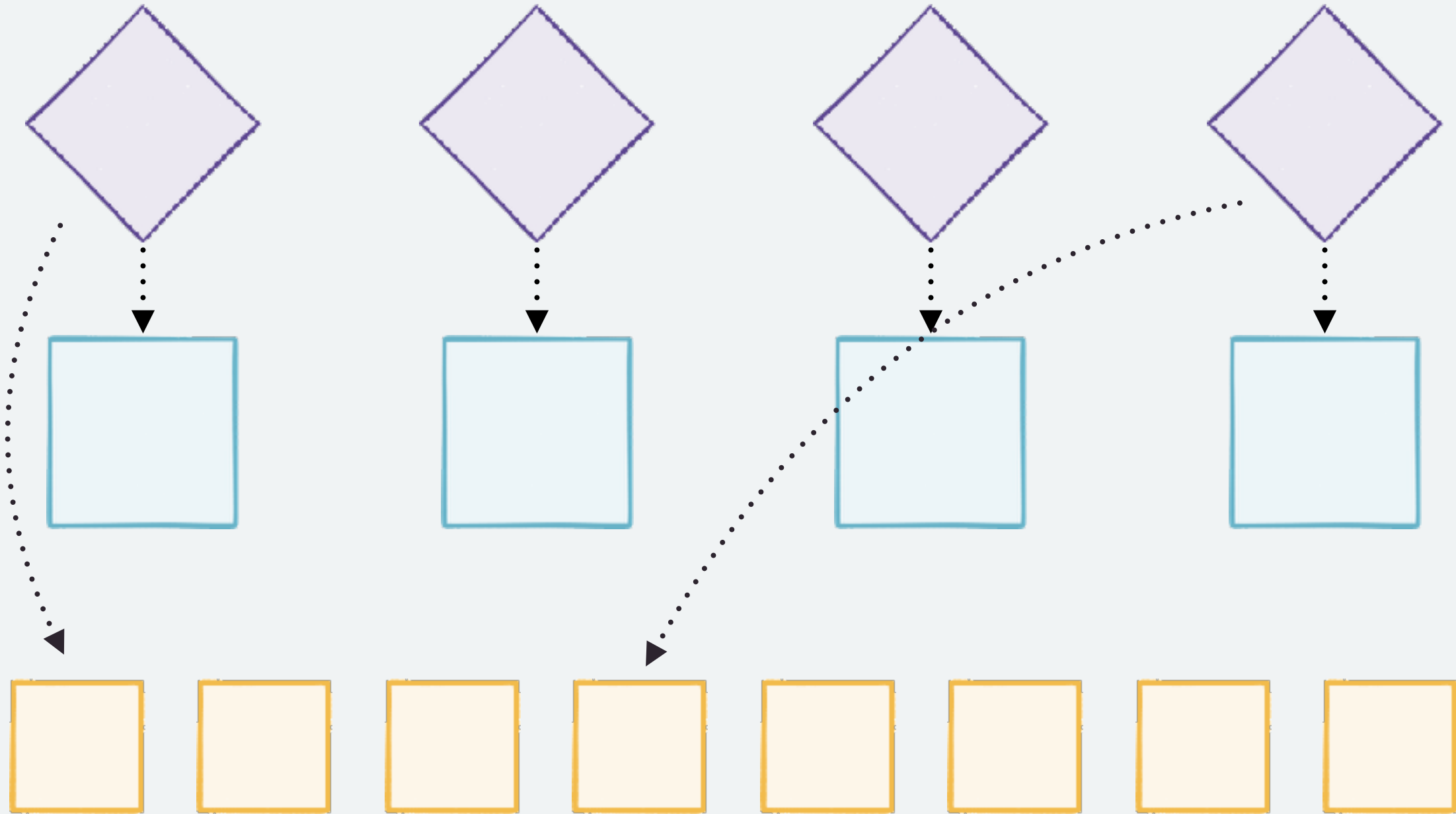
workers
databases
partitions



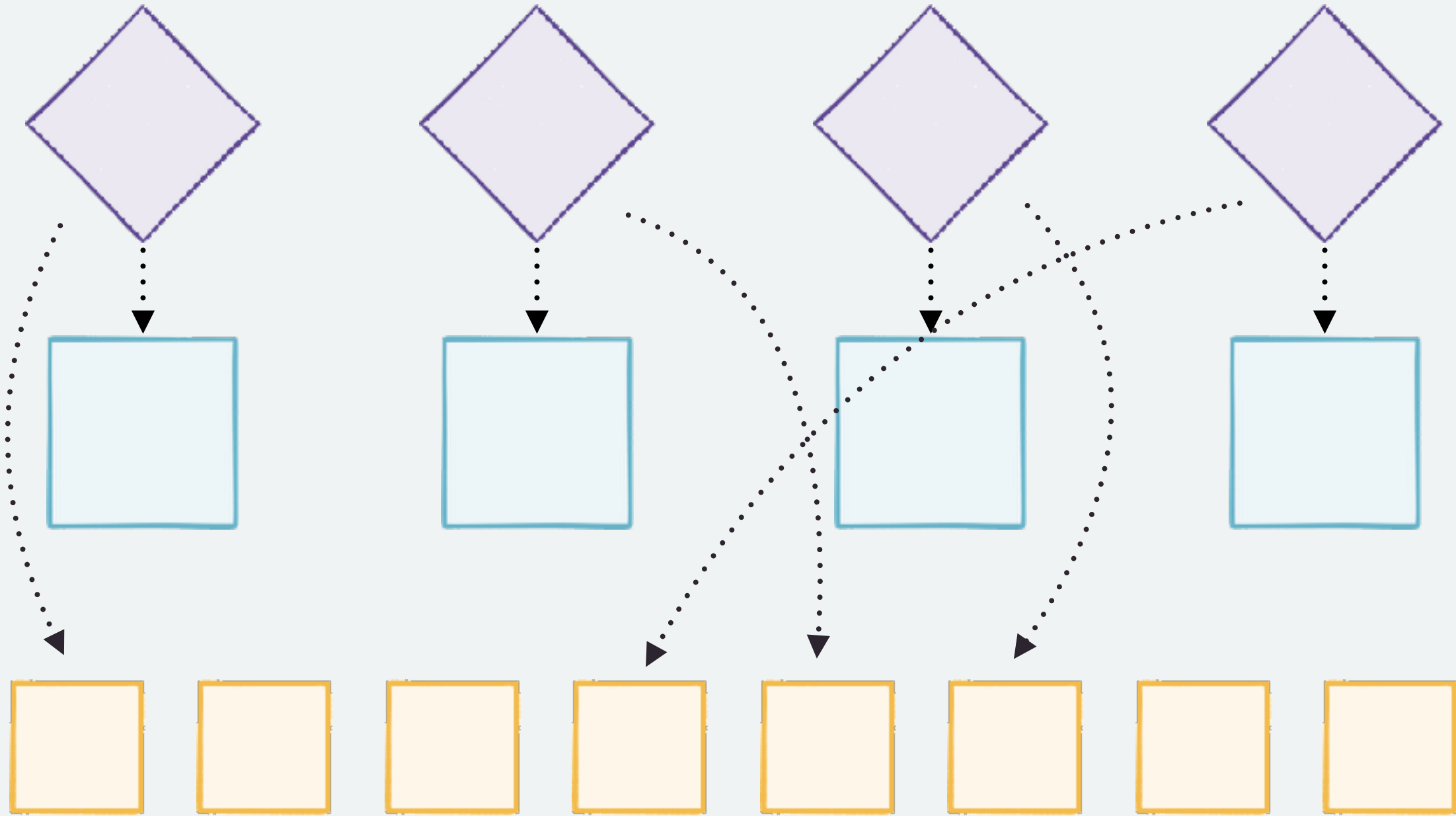
workers
databases
partitions



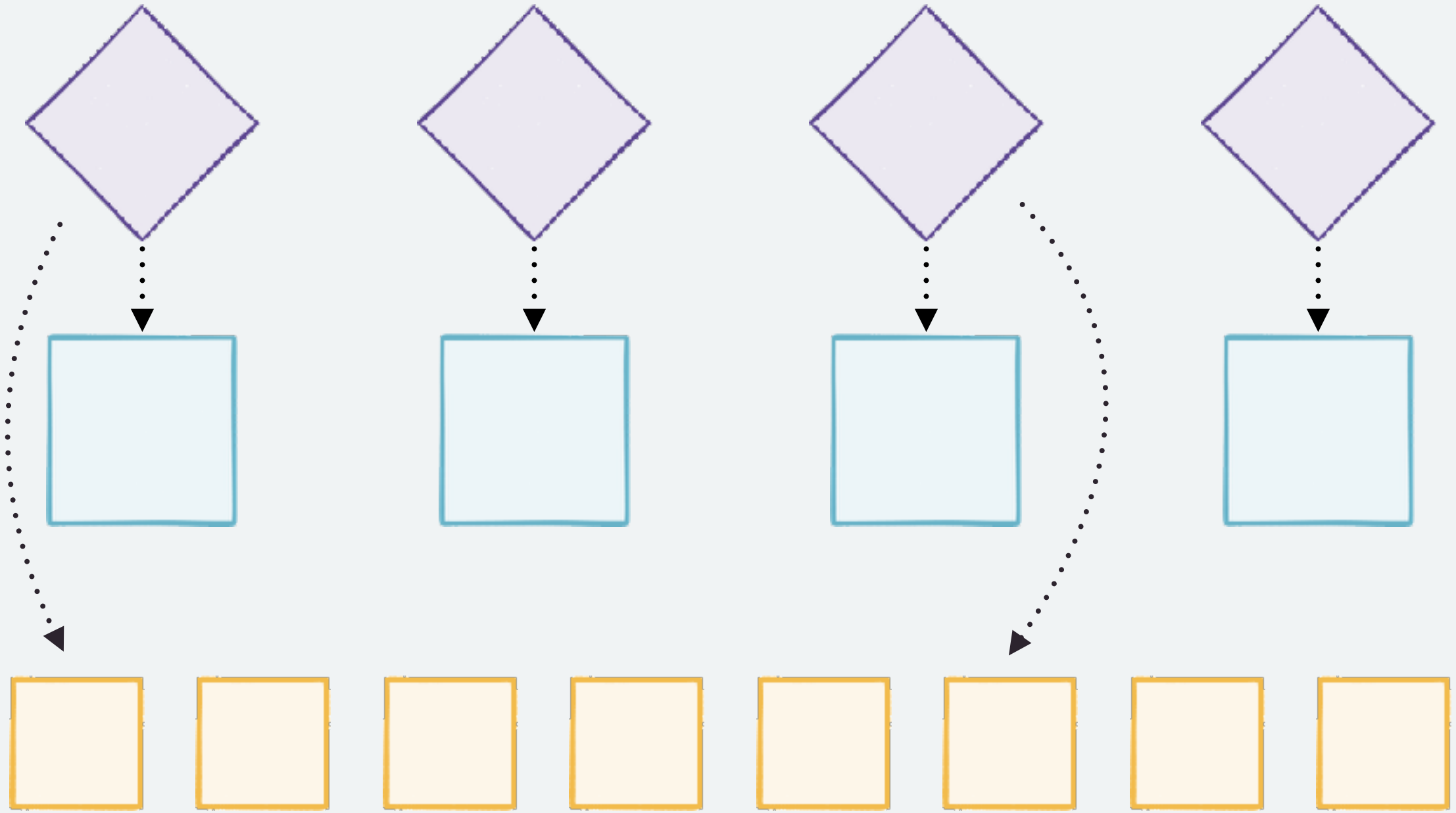
workers
databases
partitions



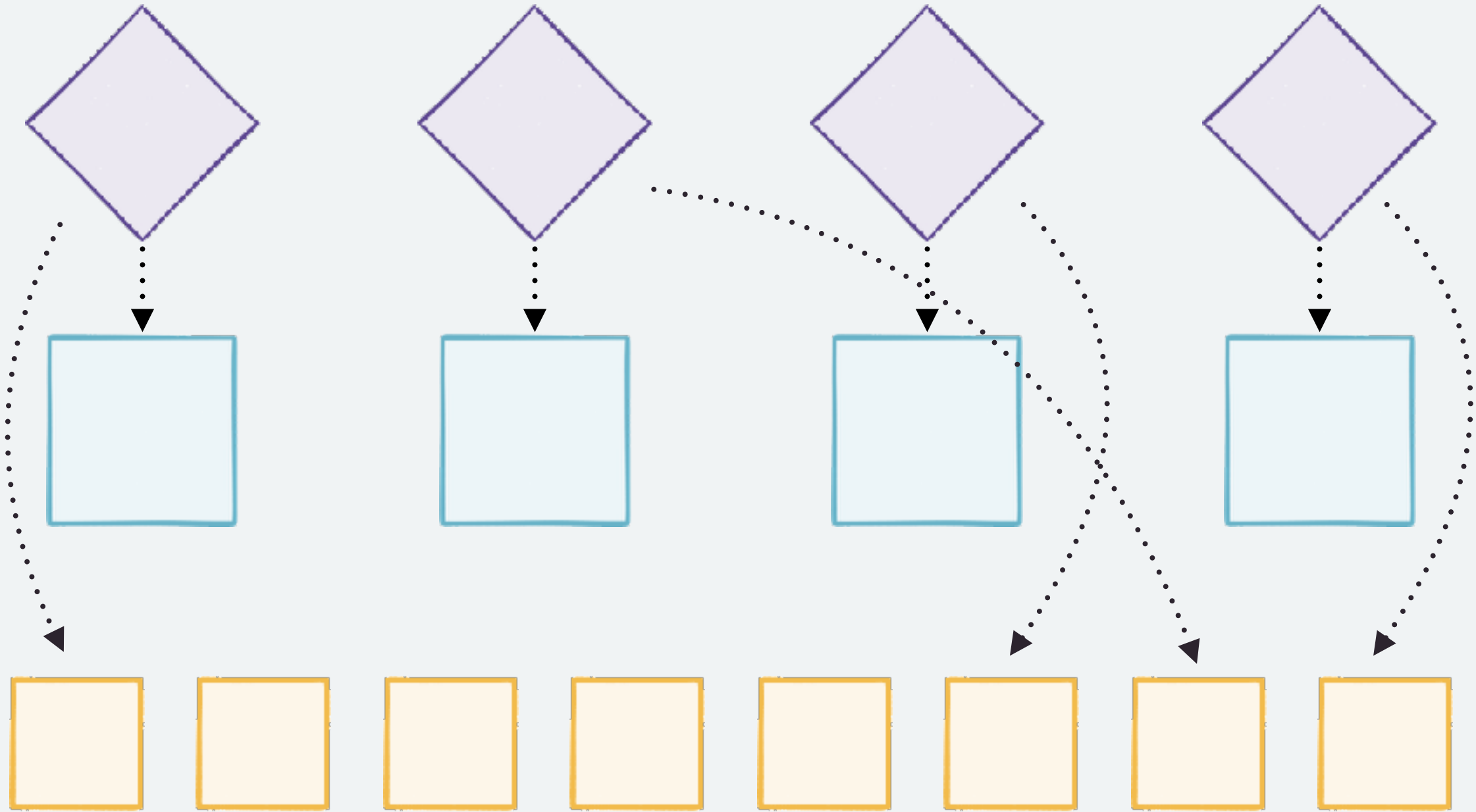
workers
databases
partitions



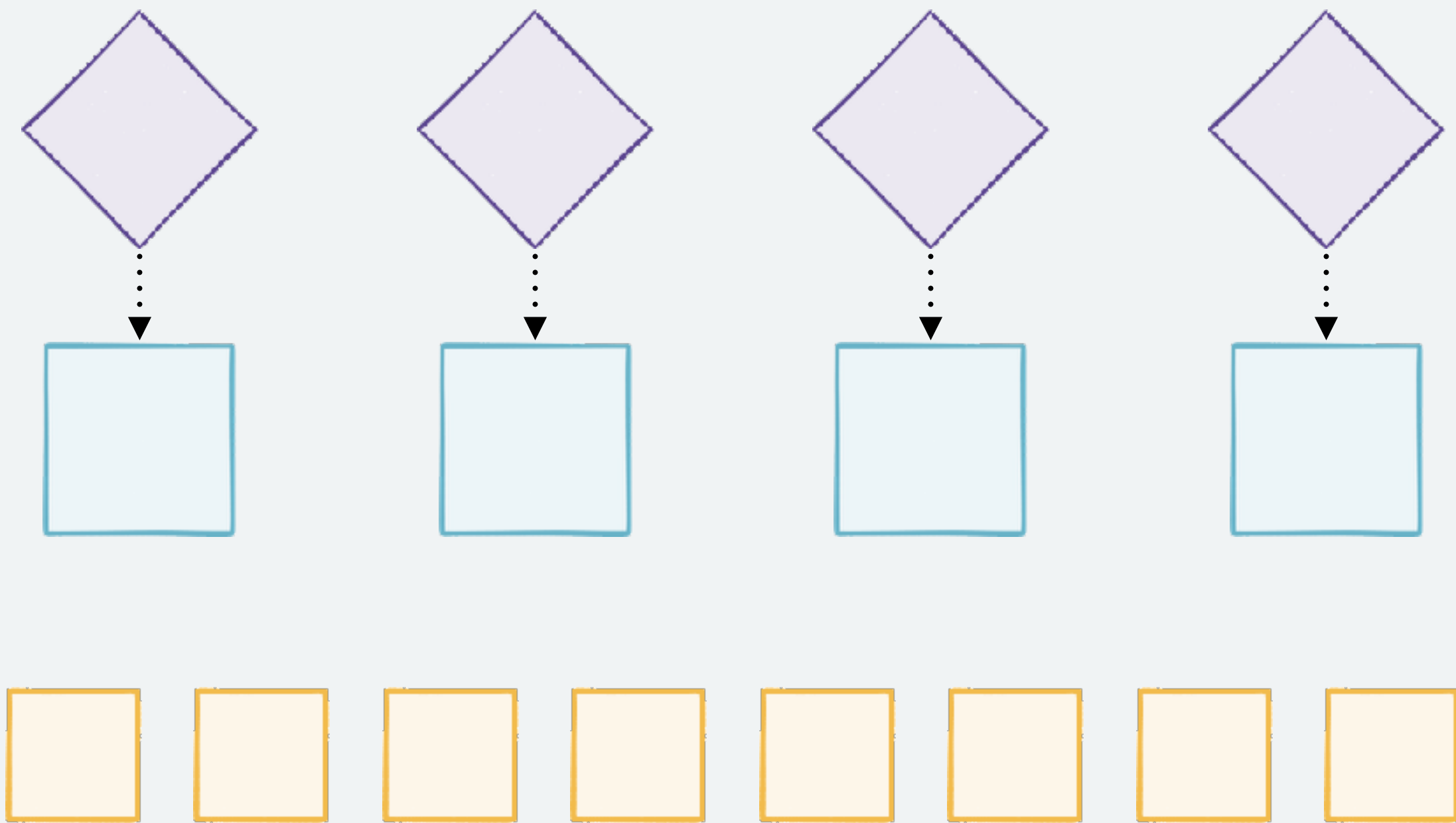
workers
databases
partitions



workers
databases
partitions



workers
databases
partitions



nose

multiprocess plugin



1.10

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

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

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

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

.....

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

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

nose attrib plugin

py.test markers

*will do
the same*



nose attrib plugin



py.test markers



Starting with Python 3.4

```
def test_even(self):  
    for i in range(0, 6):  
        with self.subTest(i=i):  
            self.assertEqual(i % 2, 0)
```

Starting with Python 3.4

```
def test_even(self):
```

```
    for i in range(0, 6):
```

```
        with self.subTest(i=i):
```

```
            self.assertEqual(i % 2, 0)
```

--parallel

Starting with Python 3.4

```
def test_even(self):
```

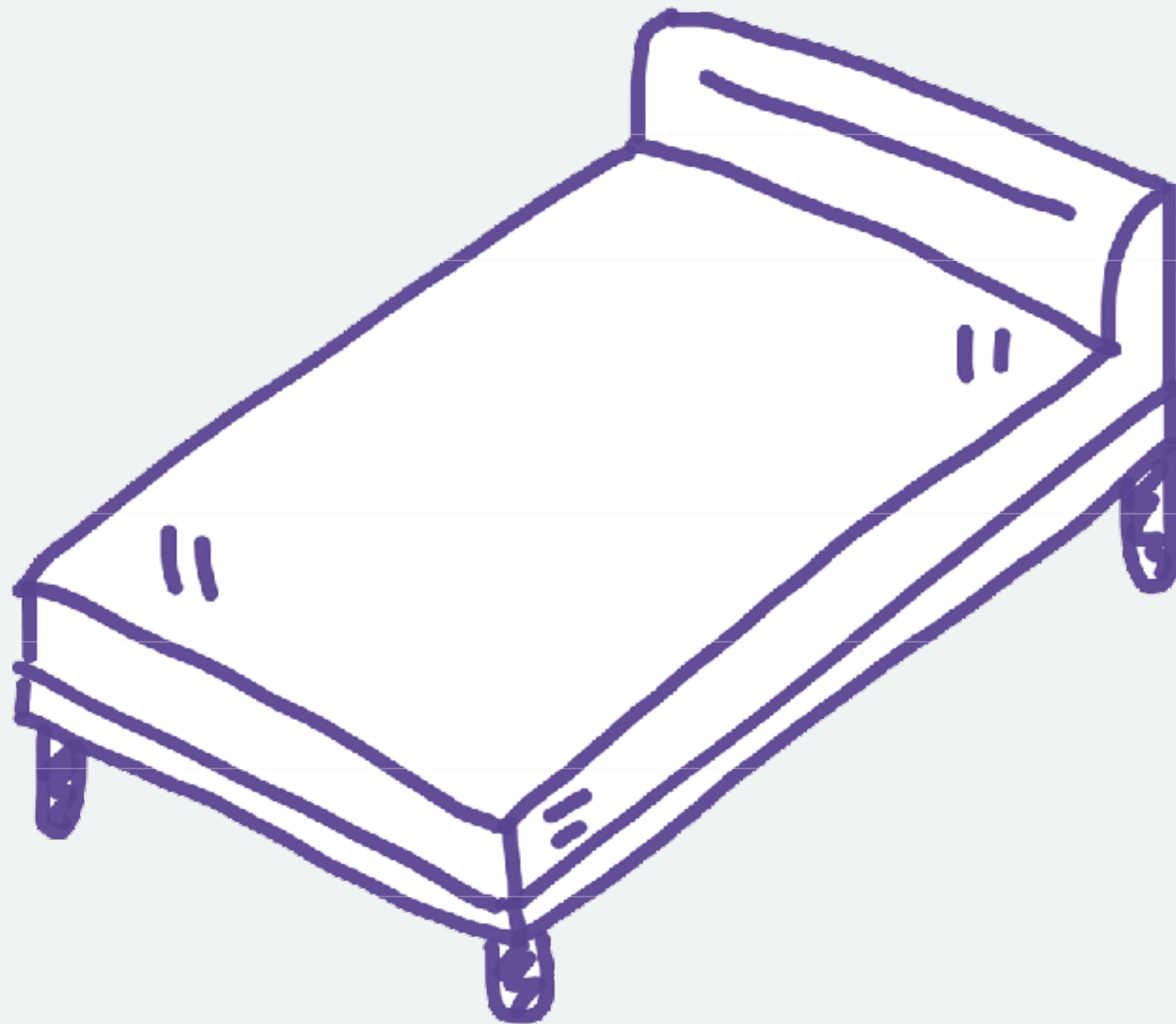
```
    for i in range(0, 6):
```

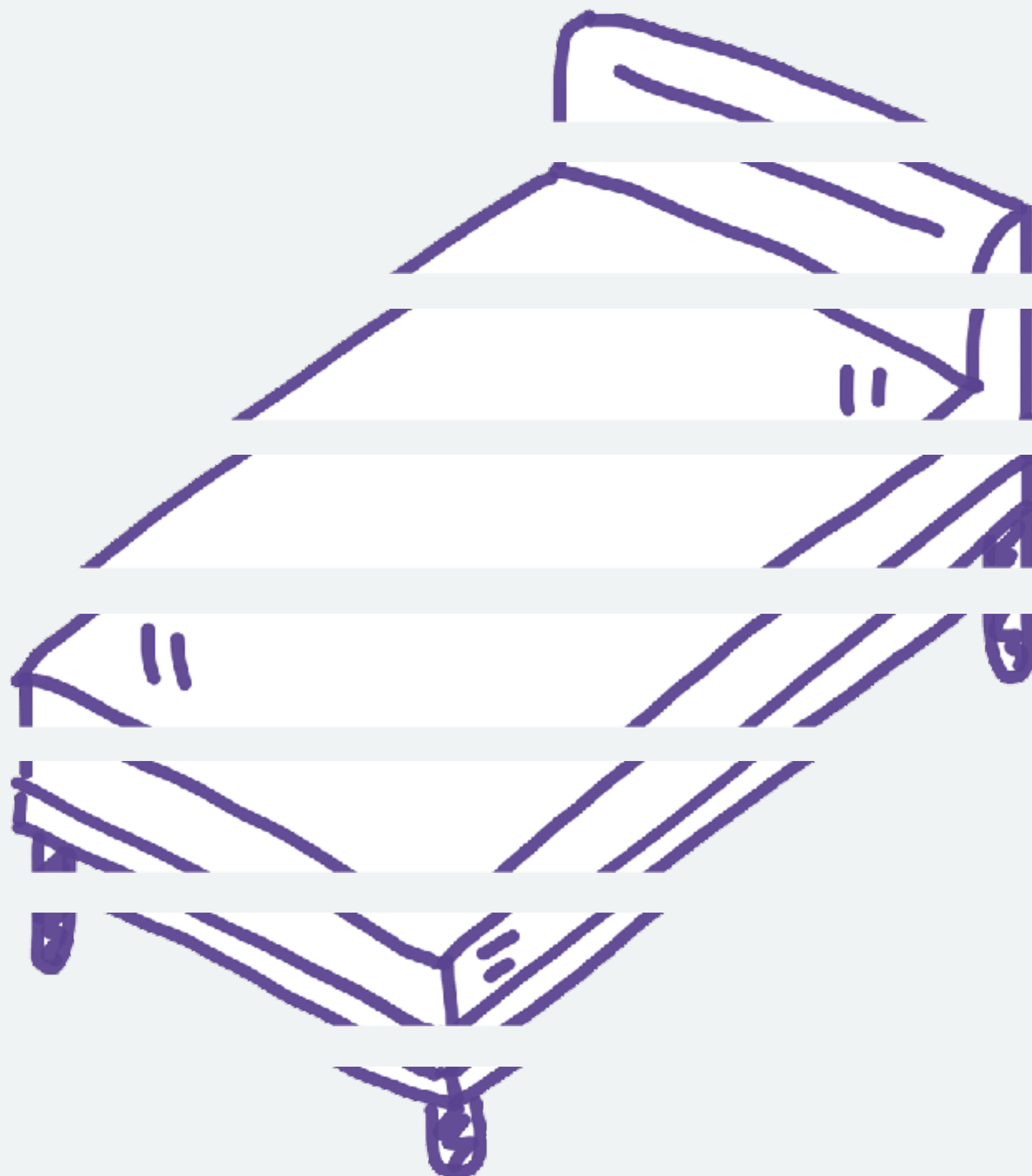
```
        with self.subTest(i=i):
```

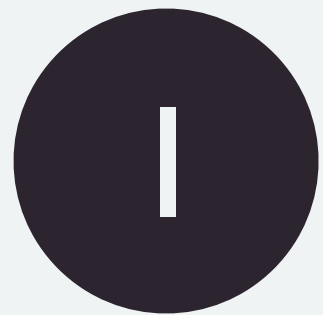
```
            self.assertEqual(i % 2, 0)
```

TEST BED

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







```
$ ./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_checks()`

7

`self.run_suite(suite)`

8

`self.teardown_databases(old_config)`

5

```
self.setup_databases()
```

6

```
self.run_checks()
```

7

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

8

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

5

```
self.setup_databases()
```

6

```
self.run_checks()
```

7

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

8

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

5

```
self.setup_databases()
```

6

```
self.run_checks()
```

7

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

8

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

9

```
self.teardown_test_environment()
```

9

```
self.teardown_test_environment()
```



original email backend

9

```
self.teardown_test_environment()
```



original email backend



original test renderer

9

```
self.teardown_test_environment()
```



original email backend



original test renderer



delete state and mailbox

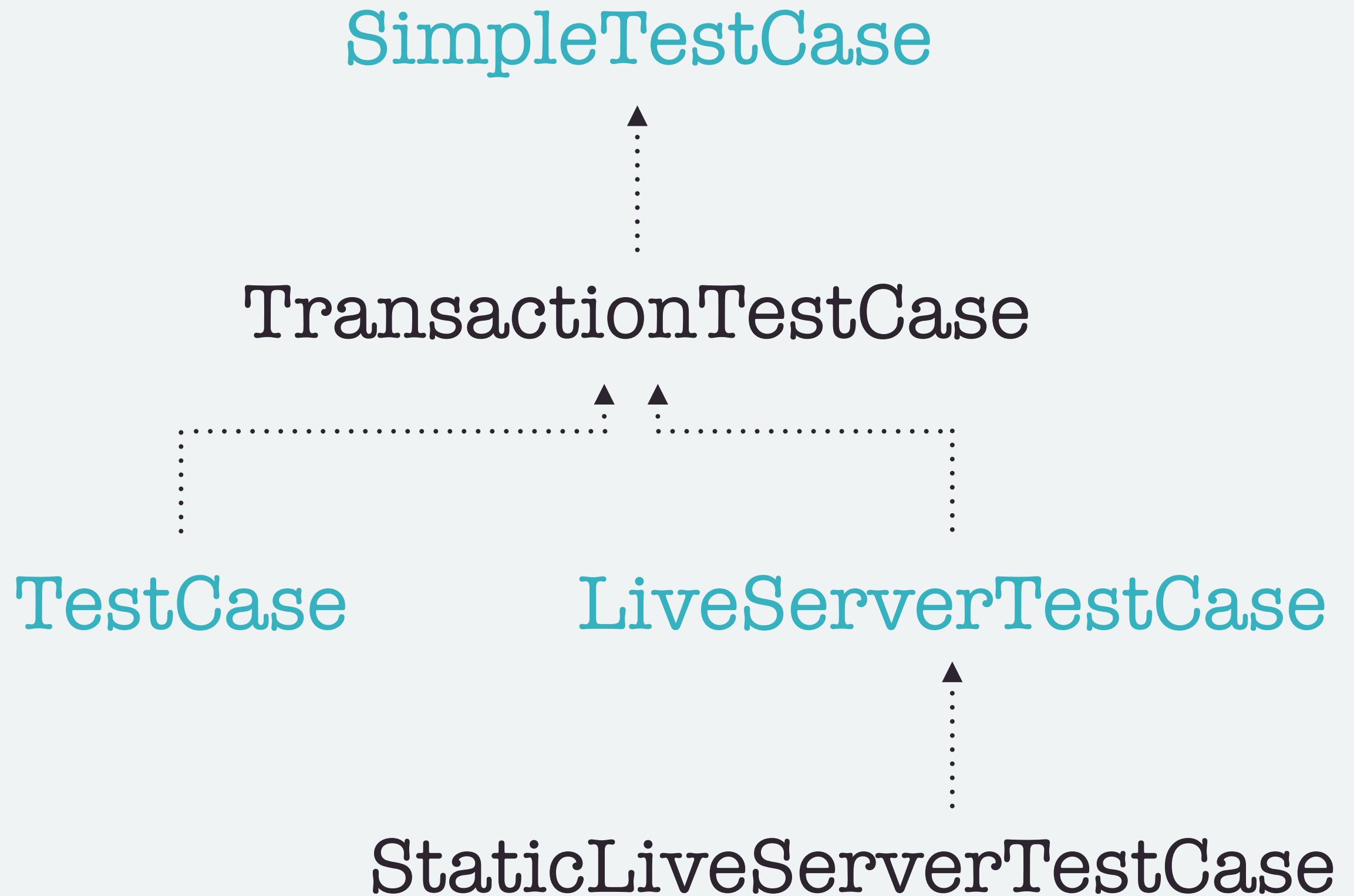
10

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

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

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

TEST CLASSES



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

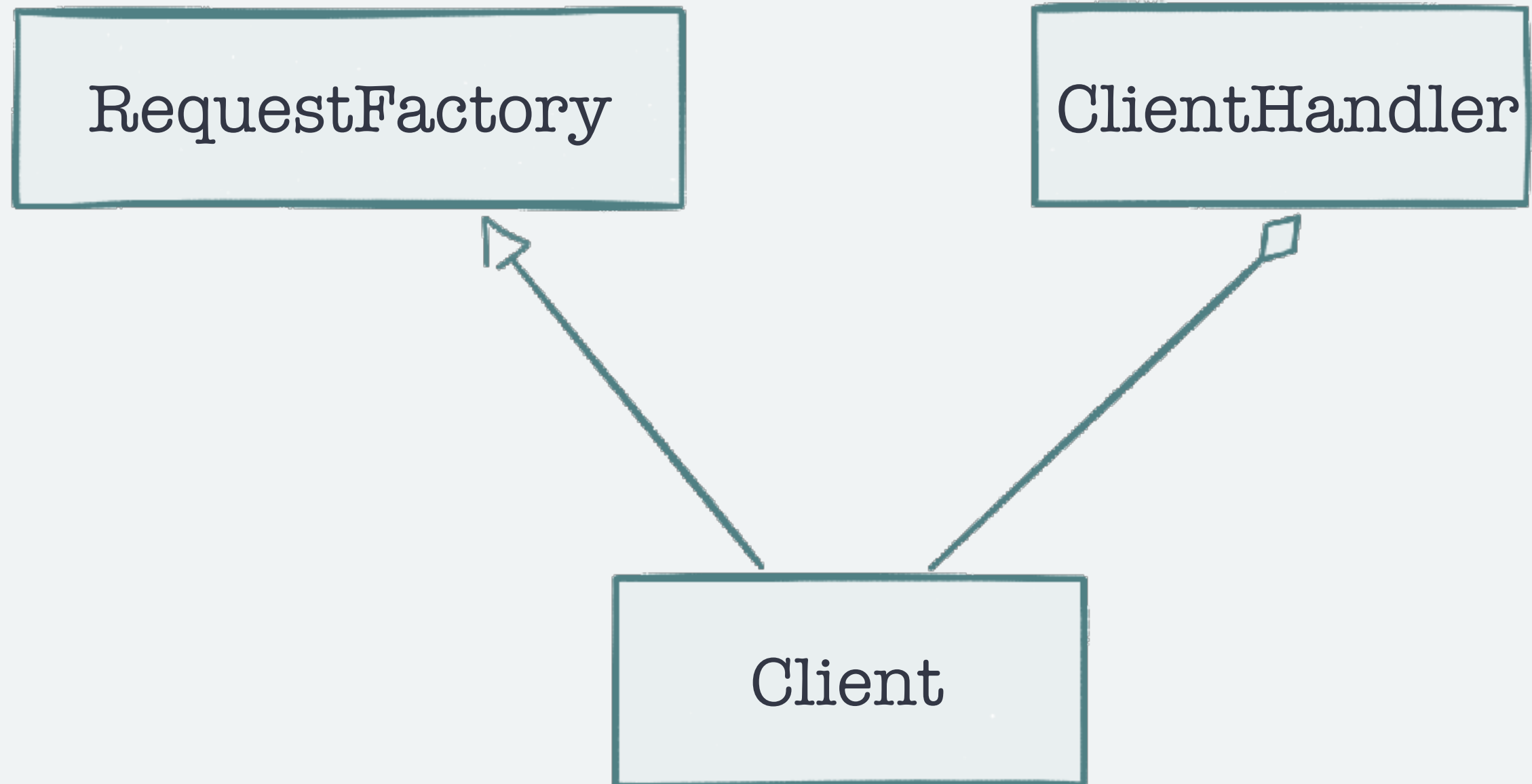
LiveServerTestCase

- ▶ 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



constructs requests
encodes data

RequestFactory

ClientHandler

Client



constructs requests
encodes data

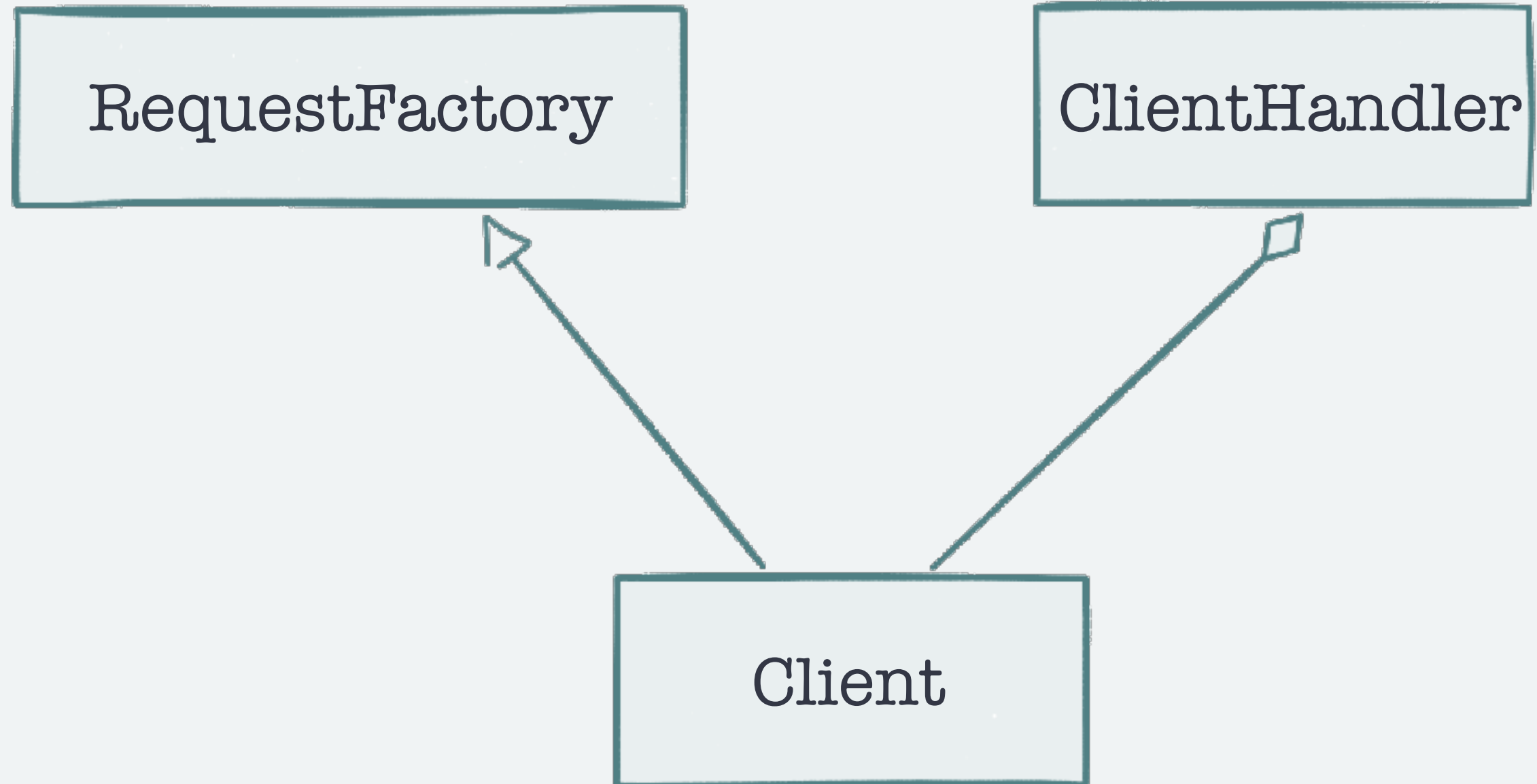
RequestFactory

ClientHandler

Client



constructs requests
encodes data



constructs requests
encodes data

RequestFactory

ClientHandler

stateful
response++
handles redirects

Client



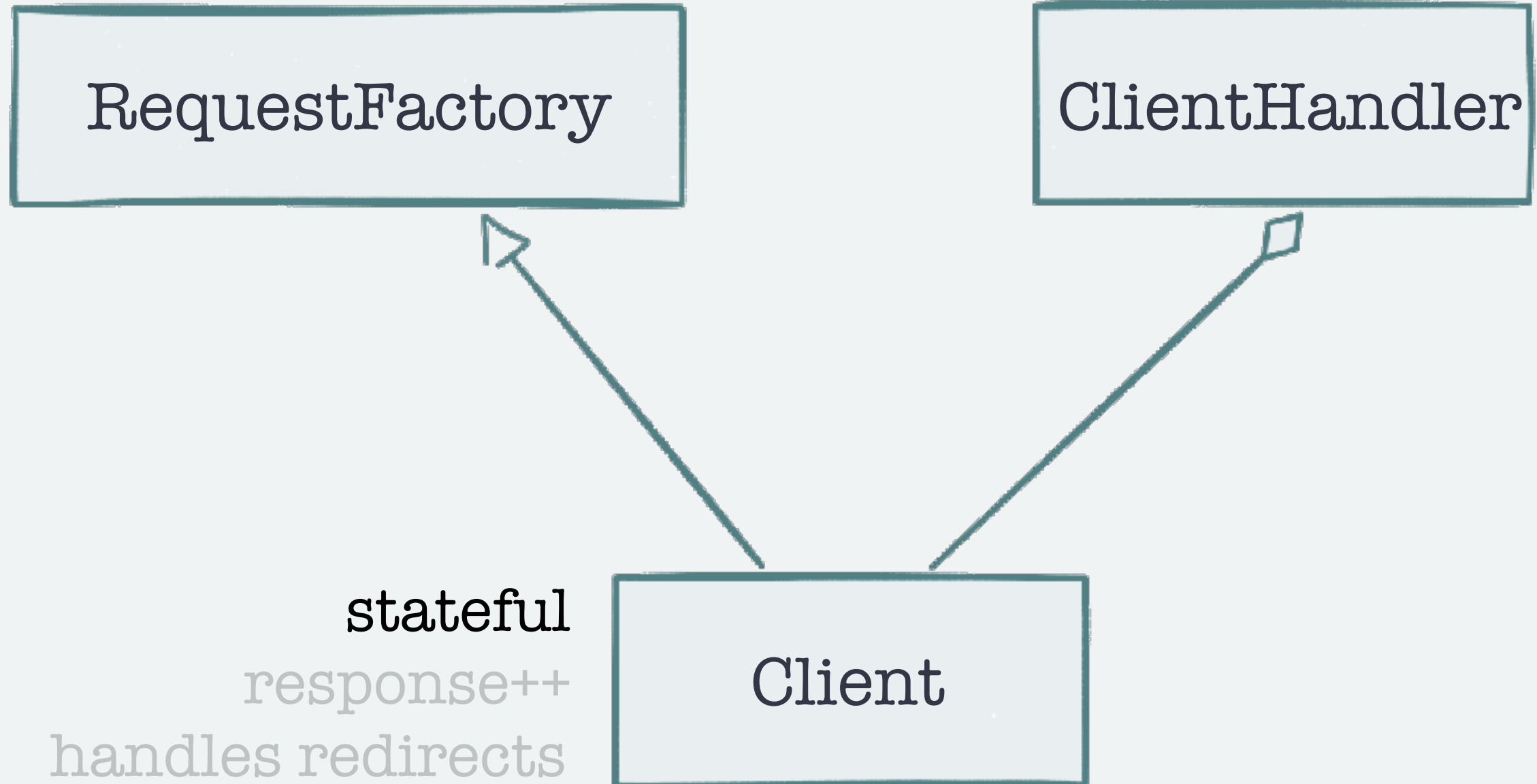
constructs requests
encodes data

RequestFactory

ClientHandler

Client

stateful
response++
handles redirects



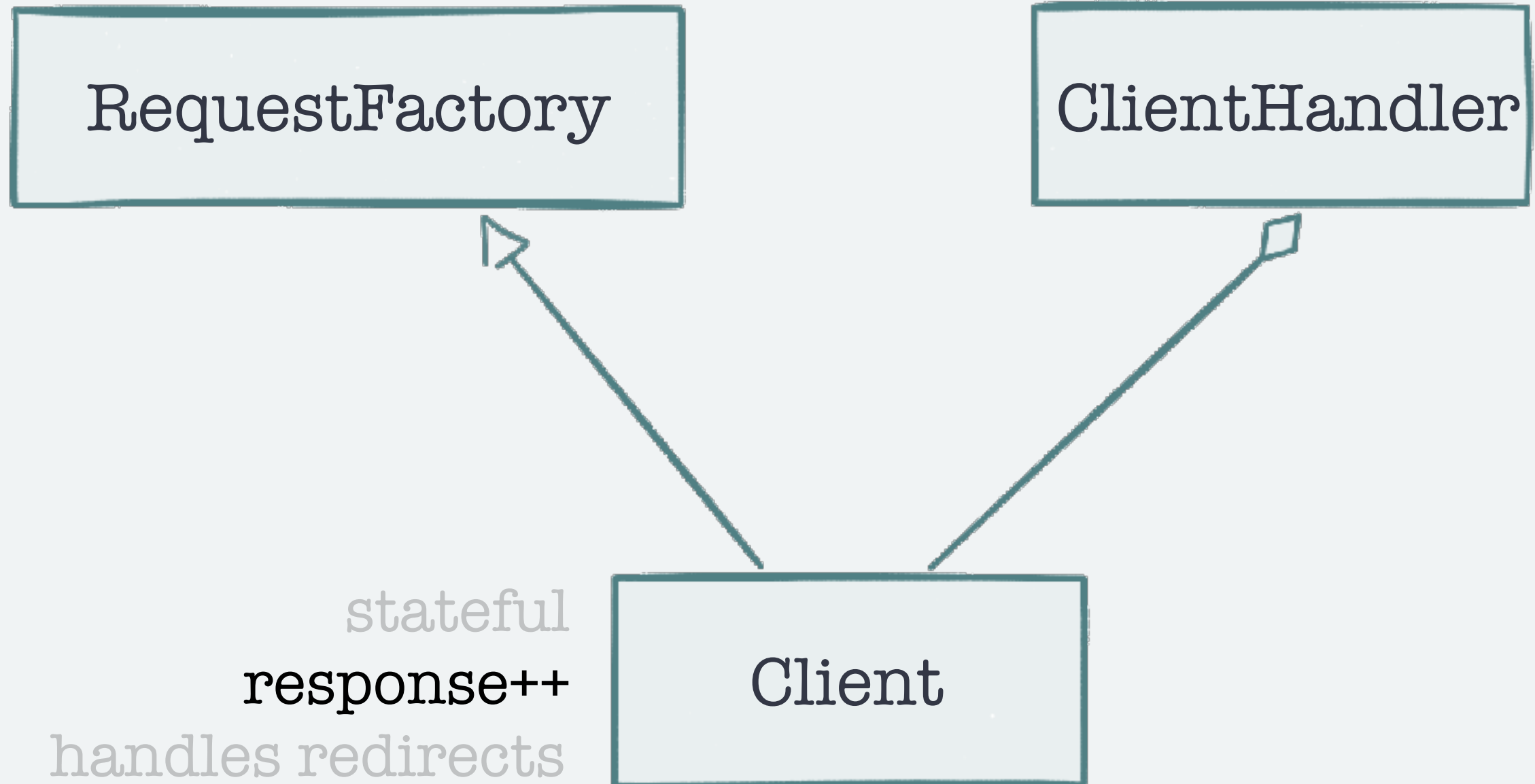
constructs requests
encodes data

RequestFactory

ClientHandler

Client

stateful
response++
handles redirects



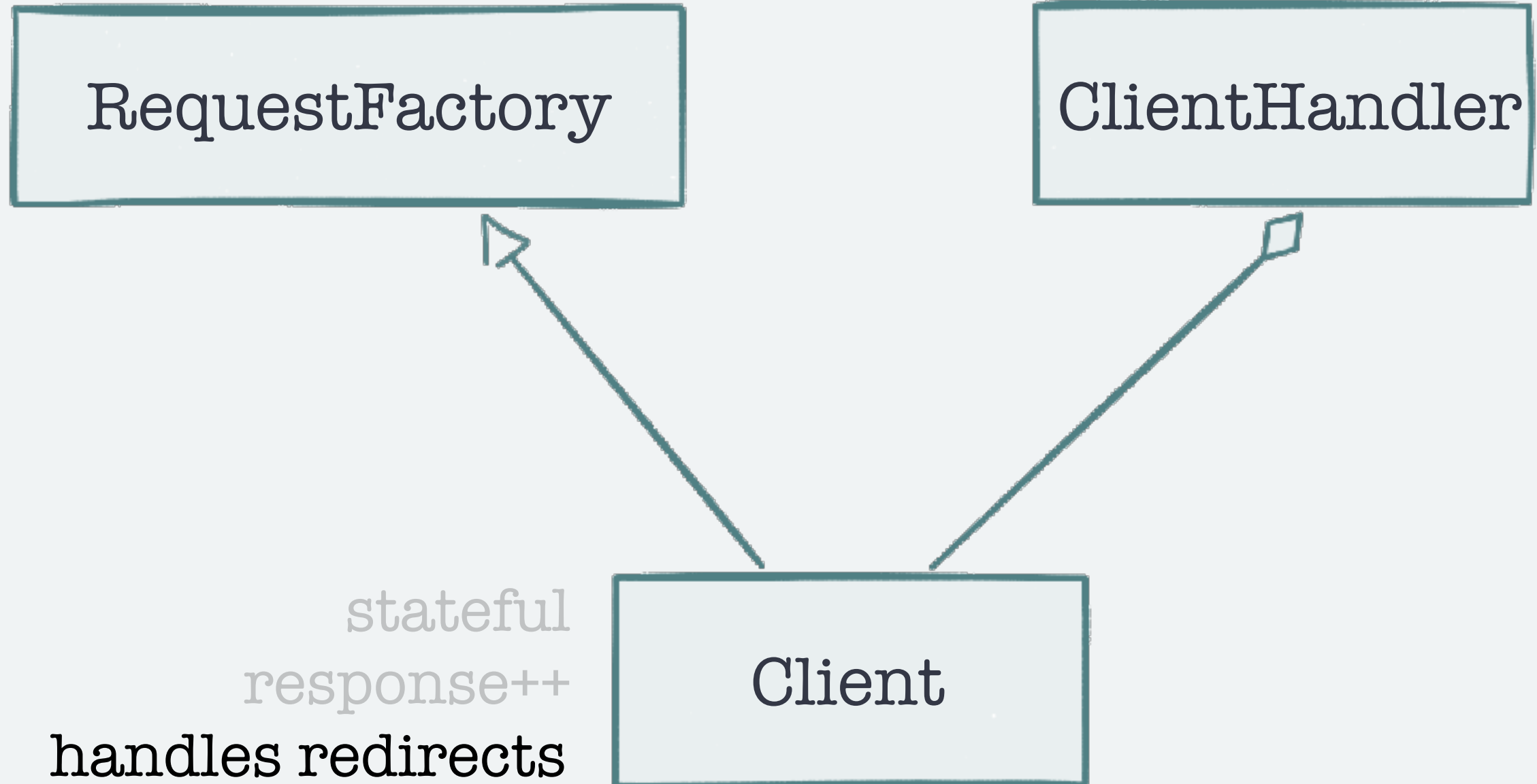
constructs requests
encodes data

RequestFactory

ClientHandler

Client

stateful
response++
handles redirects



constructs requests
encodes data

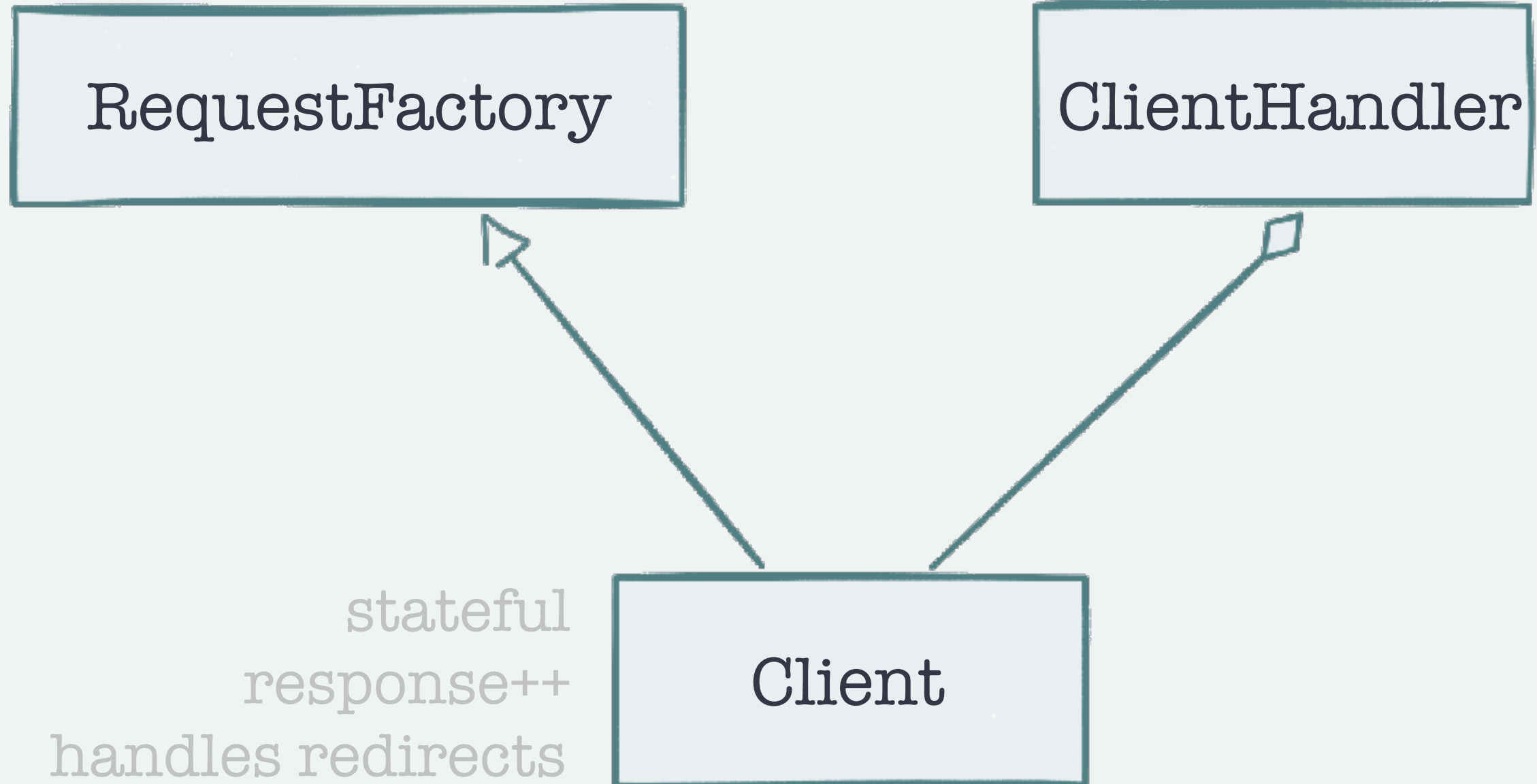
RequestFactory

loads middleware
disables CSRF
emulates

ClientHandler

stateful
response++
handles redirects

Client



constructs requests
encodes data

RequestFactory

loads middleware
disables CSRF
emulates

ClientHandler

stateful
response++
handles redirects

Client



constructs requests
encodes data

RequestFactory

loads middleware
disables CSRF
emulates

ClientHandler

stateful
response++
handles redirects

Client



constructs requests
encodes data

RequestFactory

loads middleware
disables CSRF
emulates

ClientHandler

stateful
response++
handles redirects

Client



constructs requests
encodes data

RequestFactory

loads middleware
disables CSRF
emulates

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

```
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
```

```
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
```

```
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
```

rerun for different values

```
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()
```

```
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))
```

```
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

75%

Deceptive metric



High coverage

!=

high quality

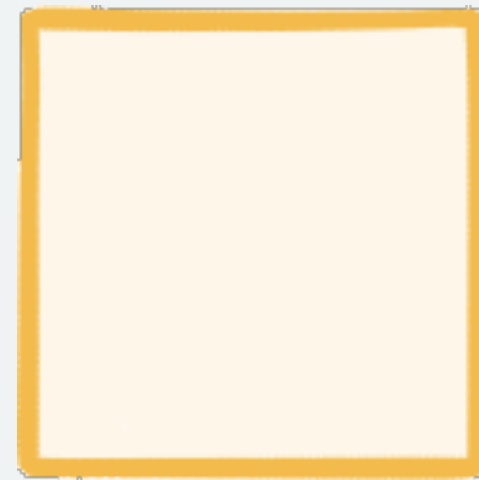
Mutation testing

available Python implementation: mutpy

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



target

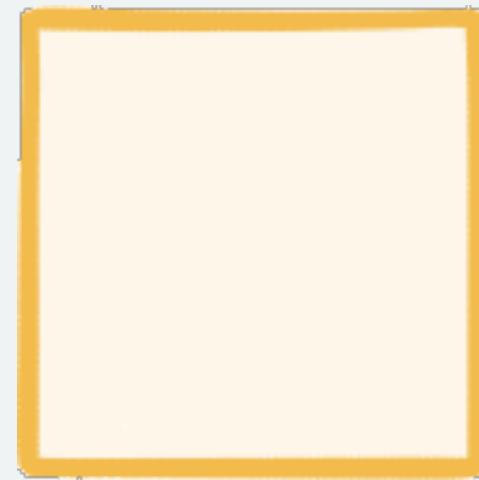


unit test

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



target



unit test

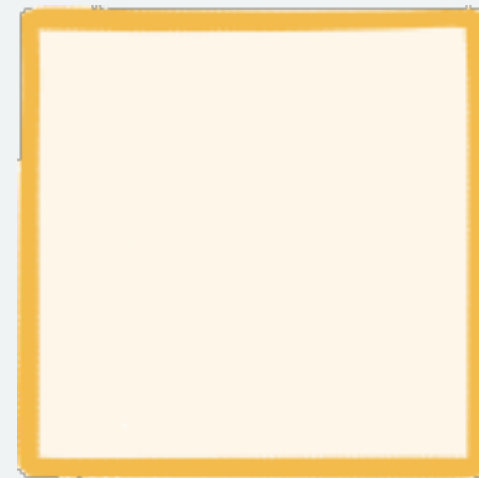
```
if foo and bar:  
    do_this()
```



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



mutant



unit test



mutant



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 100%

Django encoding utils
mutation score is

64%

Django encoding utils
mutation score is

64%

while the coverage is 100%

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

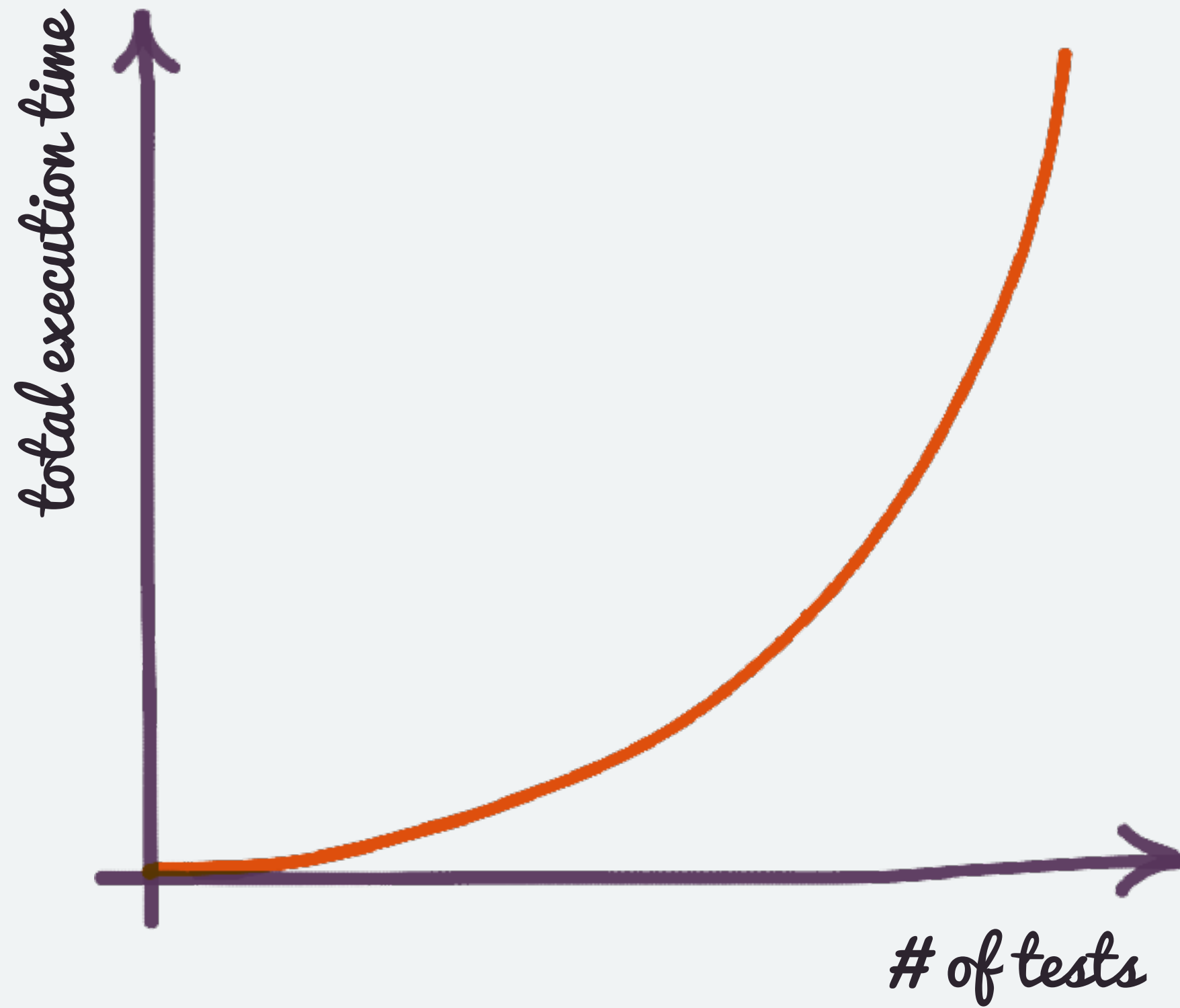
| └— __init__.py

|— models.py

|— tests.py

|— views.py

**When testing,
more is better**



**SLOW TESTS,
SLOW FEEDBACK LOOP**

7 tips on how to speed up your tests

7 tips on how to speed up your tests

#4 will shock you

1. use MD5PasswordHasher

1. use MD5PasswordHasher
2. have more SimpleTestCase

1. use MD5PasswordHasher
2. have more SimpleTestCase
3. use setUpTestData()

1. use MD5PasswordHasher
2. have more SimpleTestCase
3. use setUpTestData()
4. use mocks EVERYWHERE

1. use MD5PasswordHasher
2. have more SimpleTestCase
3. use setUpTestData()
4. use mocks EVERYWHERE

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

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

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

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

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

.....

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

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

.....

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

6. don't save model objects if not necessary

.....

6. don't save model objects if not necessary

instead of Robot.objects.create()

6. don't save model objects if not necessary

instead of Robot.objects.create()

maybe do Robot()

6. don't save model objects if not necessary

instead of Robot.objects.create()

maybe do Robot()

or RobotFactory.build()

6. don't save model objects if not necessary

instead of Robot.objects.create()

maybe do Robot()

or RobotFactory.build()

or RobotFactory.stub()

6. don't save model objects if not necessary

instead of Robot.objects.create()

maybe do Robot()

or RobotFactory.build()

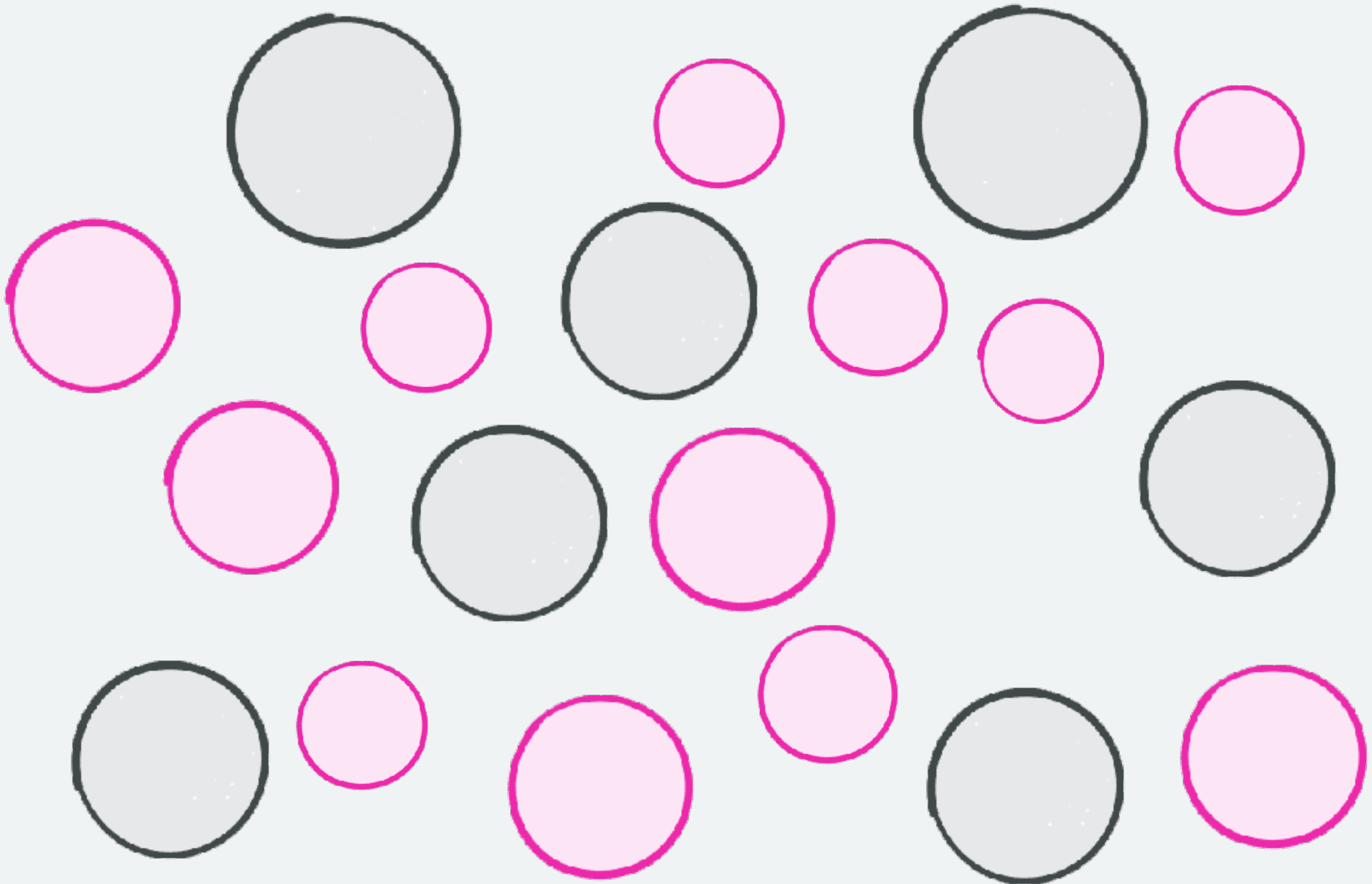
or RobotFactory.stub()

} factory
boy

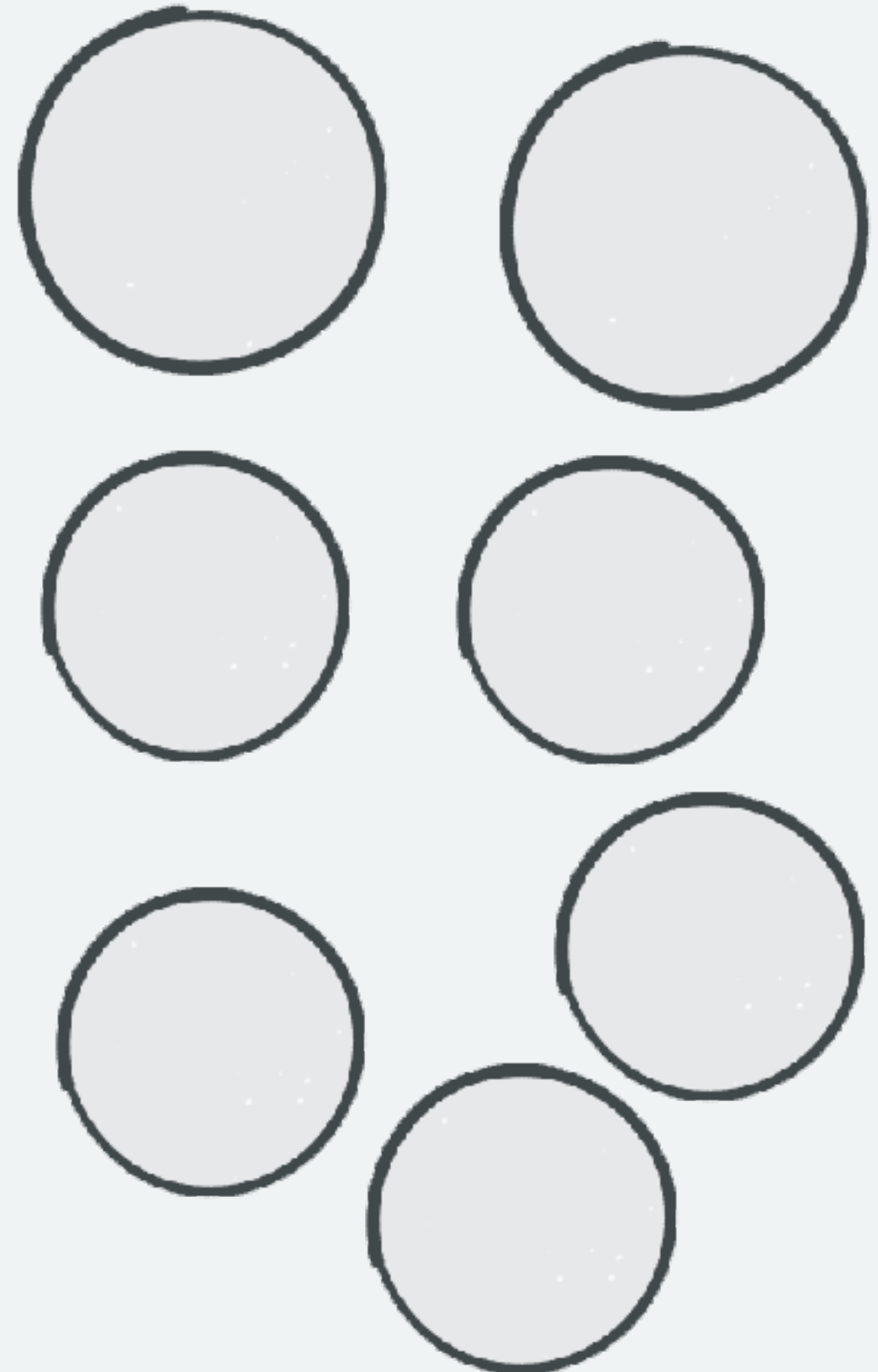
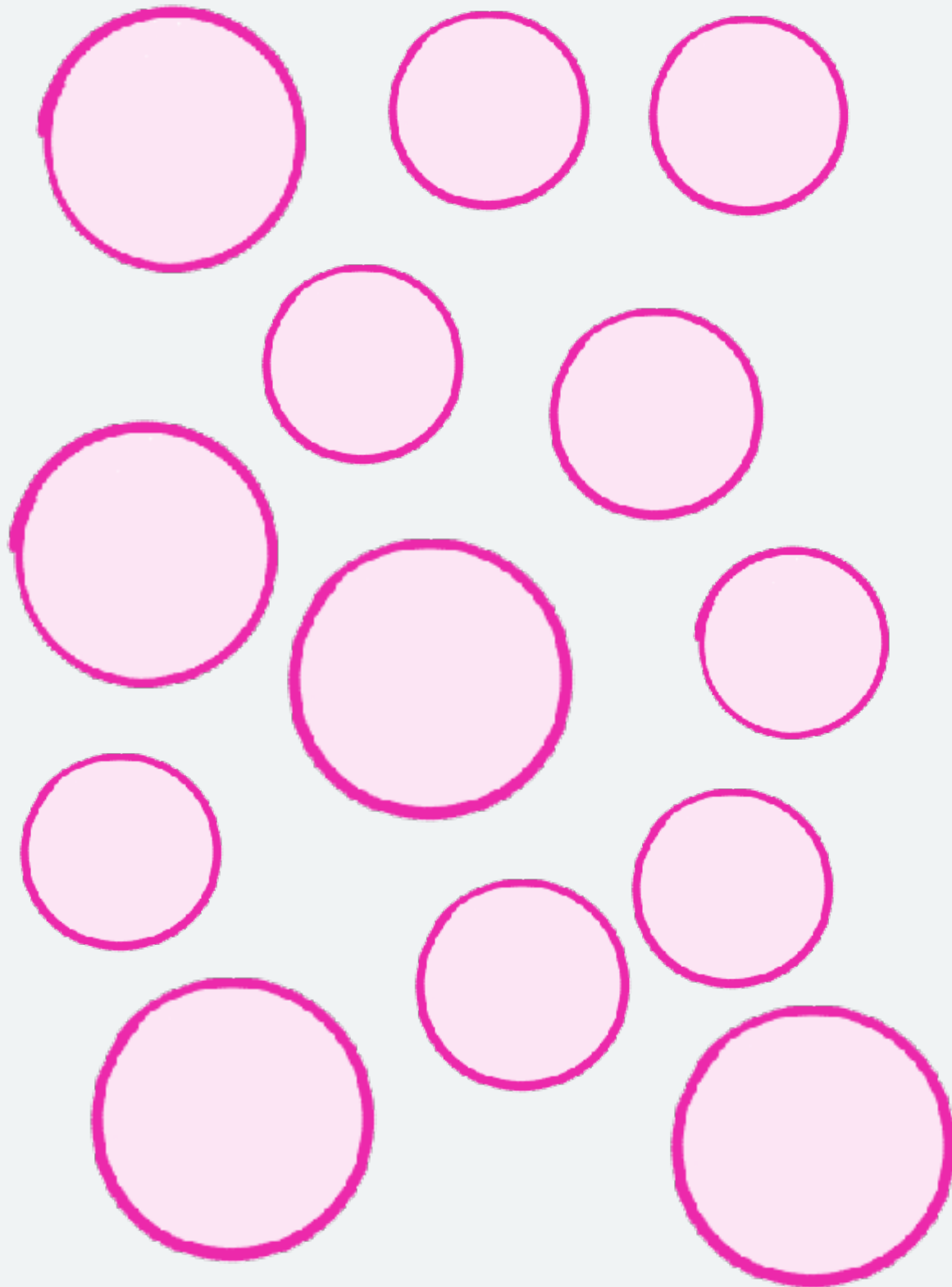
7. isolate unit tests

.....

7. isolate unit tests



7. isolate unit tests



7. isolate unit tests

unit tests



functional tests



7. isolate unit tests

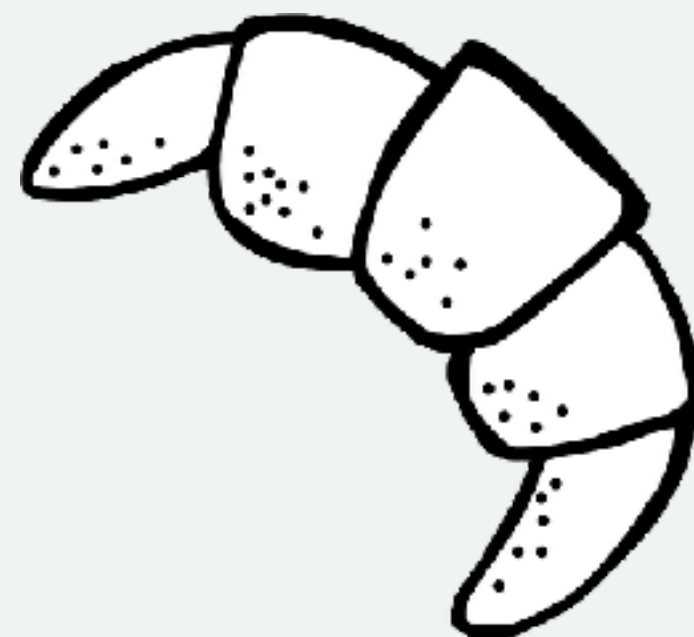
unit tests



functional tests



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



**Ugh,
taxes!**

Product

name
ingredients
price

.....

ProductQuestionnaire

product
category
drink_category
food_category
has_decorations
is_coated_in_chocolate
is_warm
is_cold



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



Main

Add new product

Manage products

Fill in a product questionnaire

Recalculate VAT

Production code



Test

```
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):  
        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):
```

```
        # Do something with the form data
```

```
        # Save the form data to the database
```

```
        # Redirect to the success page
```

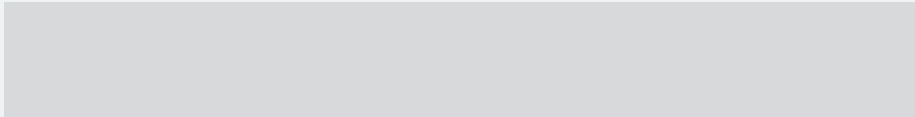
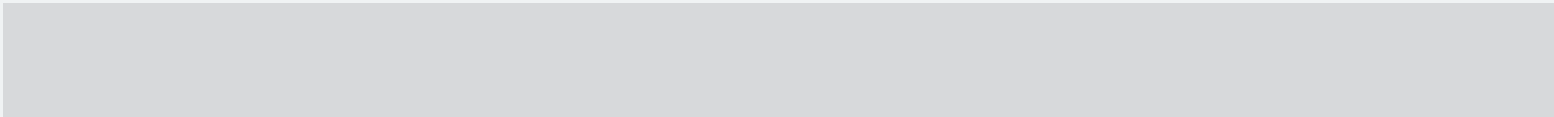
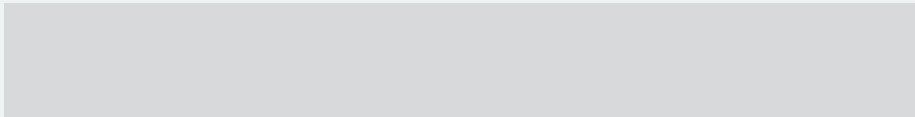
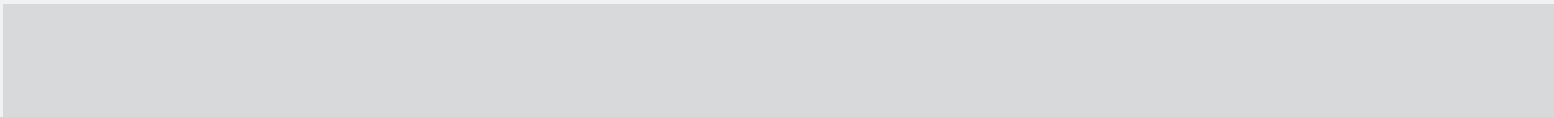
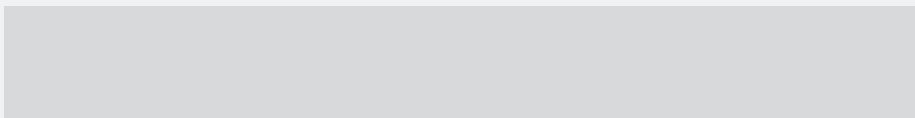
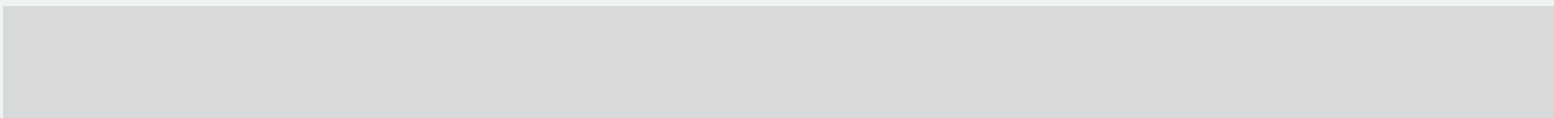
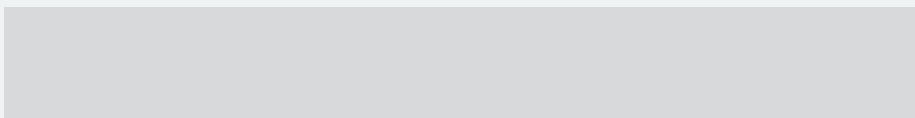
```
        # Do something with the form data
```

```
        # Save the form data to the database
```

```
        # Redirect to the success page
```

```
        # Do something with the form data
```

```
    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**

```
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)
```

solution #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

```
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  
        ), 20)
```

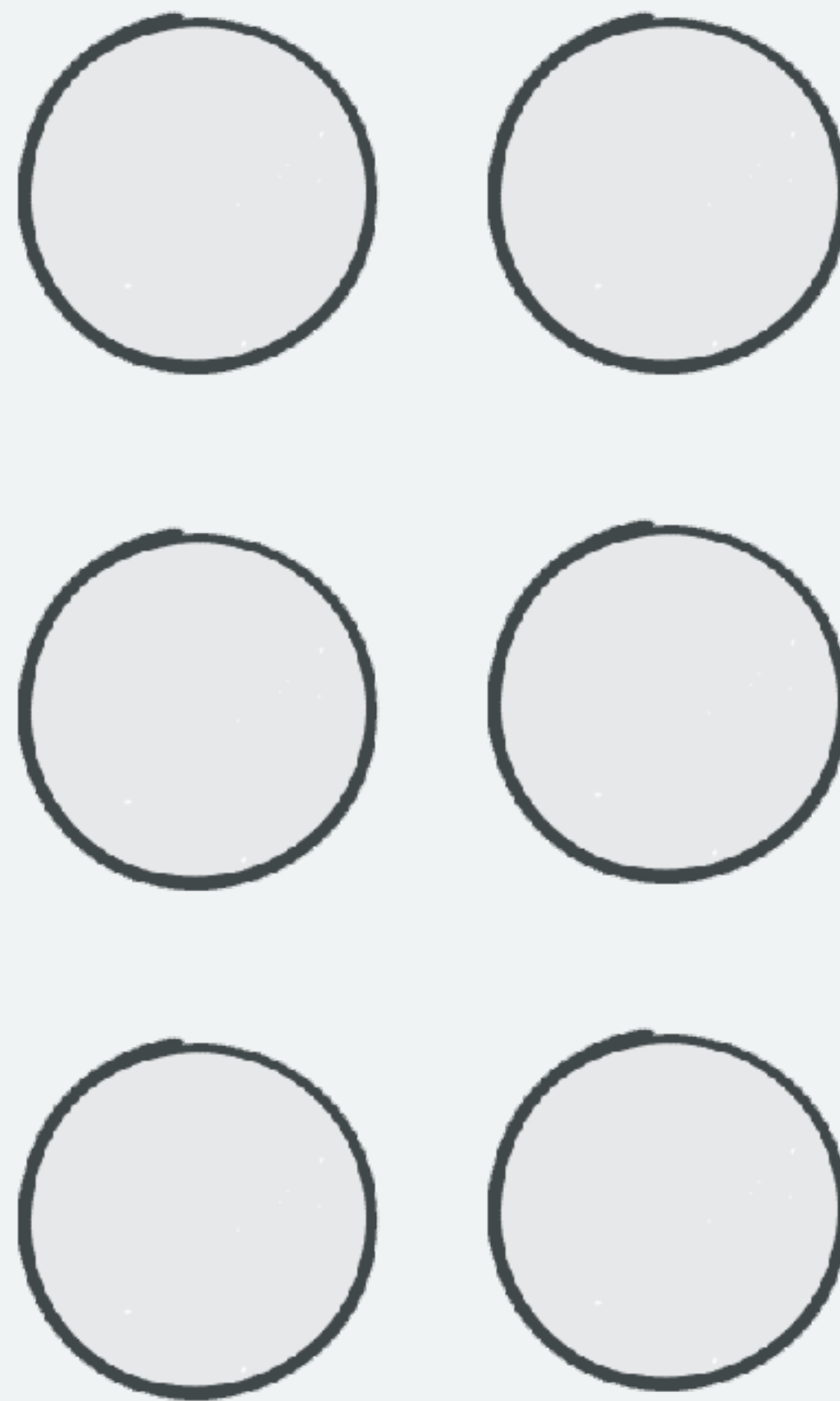
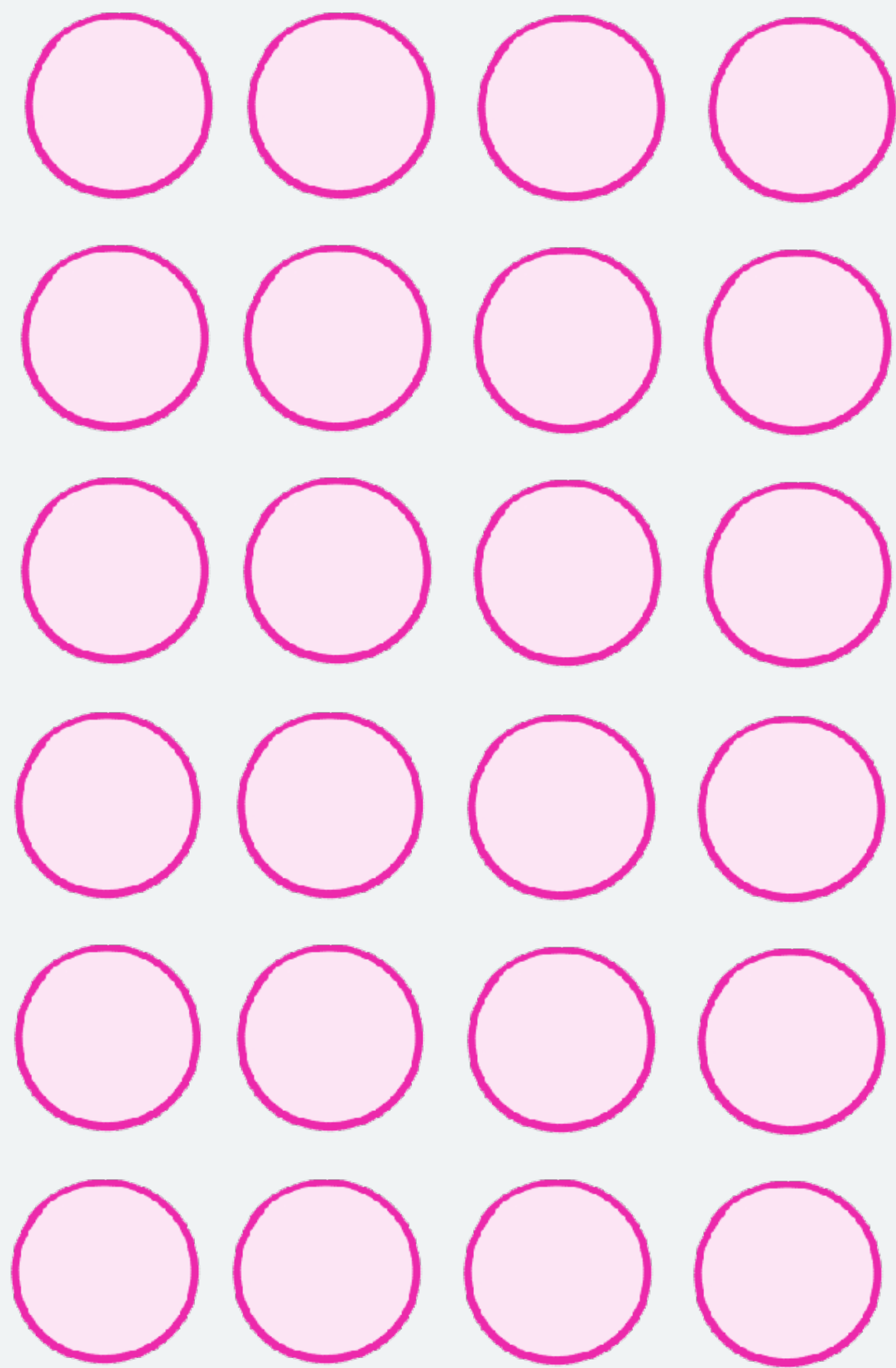
To test if I need to pay 20% VAT for biscuits coated in chocolate, I need to:

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

REUSABILITY

EXTENSIBILITY

TESTABILITY



**Tests have more in them
than we think**

HAPPY TESTING

and big bear hug thanks!