PART 3: TESTING

Infrastructure as Code

Kief Morris @ ThoughtWorks

WORKSHOP CONTENTS

FOUNDATIONS

STACKS

TESTING

RUNTIMES

DESIGN

WORKFLOW

PART 3

TESTING AND PIPELINES

WHAT AND HOW

CONTINUOUS TESTING

PIPELINES

TEST FIXTURES

LIFECYCLES

CHALLENGES

SPEED

VALUE

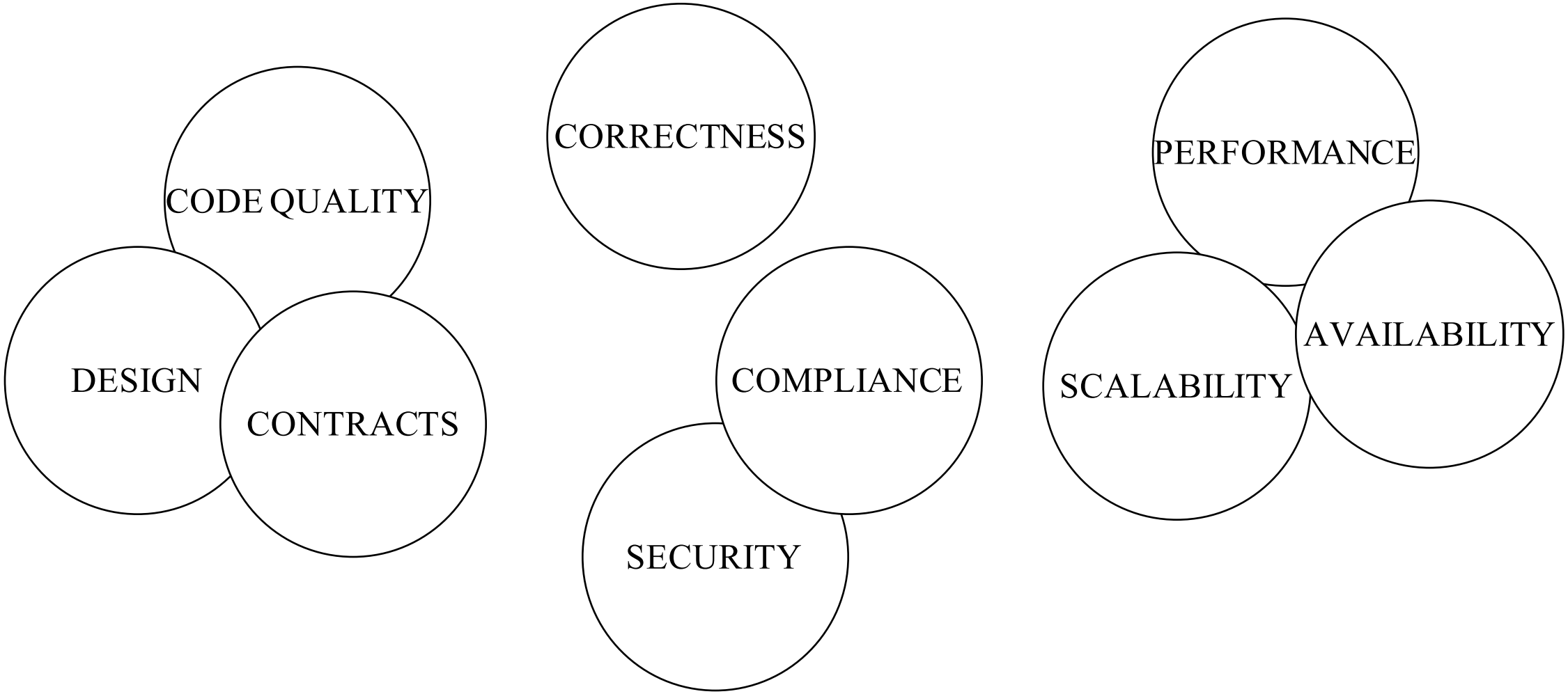
EFFORT

WHAT AND HOW

GOAL

Discover issues quickly, so we can fix them quickly

WHAT YOU CAN VALIDATE



PUZZLE:

Should we test declarative code?

Testing declarative code

Is it necessary or valuable?

```
subnet:
  name: private_A
  address_range: 192.168.0.0/16
```

```
GIVEN a cloud account
WHEN  a subnet is created
THEN  the subnet exists
```

```
subnet:
  name: STRING
  zone: PUBLIC | INTERNAL | CONTROL
```

GIVEN a cloud account

WHEN a **PUBLIC** subnet is created

THEN a connection from the public internet SUCCEEDS

```
subnet:
  name: STRING
  zone: PUBLIC | INTERNAL | CONTROL
```

GIVEN a cloud account

WHEN an **INTERNAL** subnet is created

THEN a connection from the public internet **FAILS**

Tests are useful when code can create DIFFERENT OUTCOMES

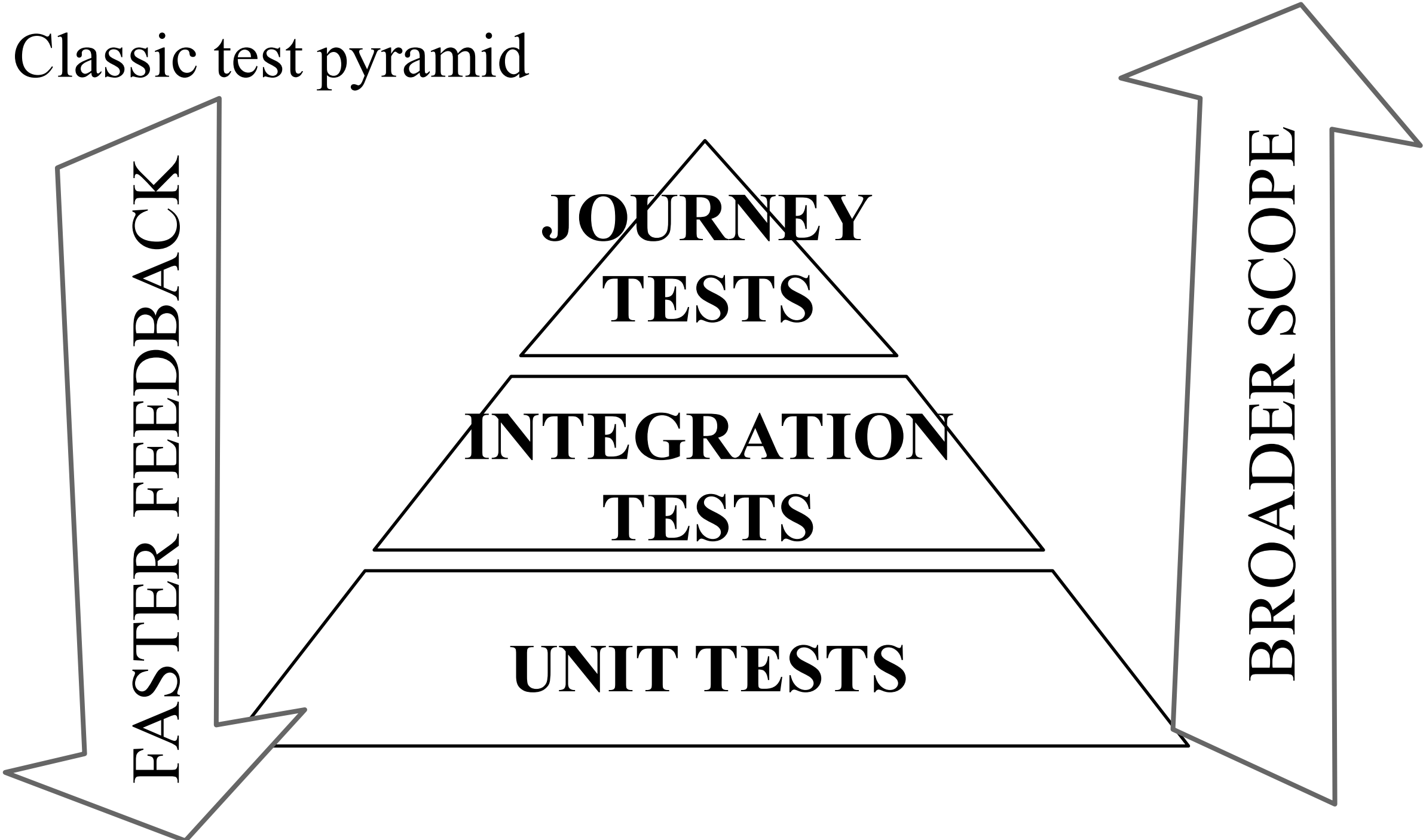

CONTINUOUS TESTING

GOAL

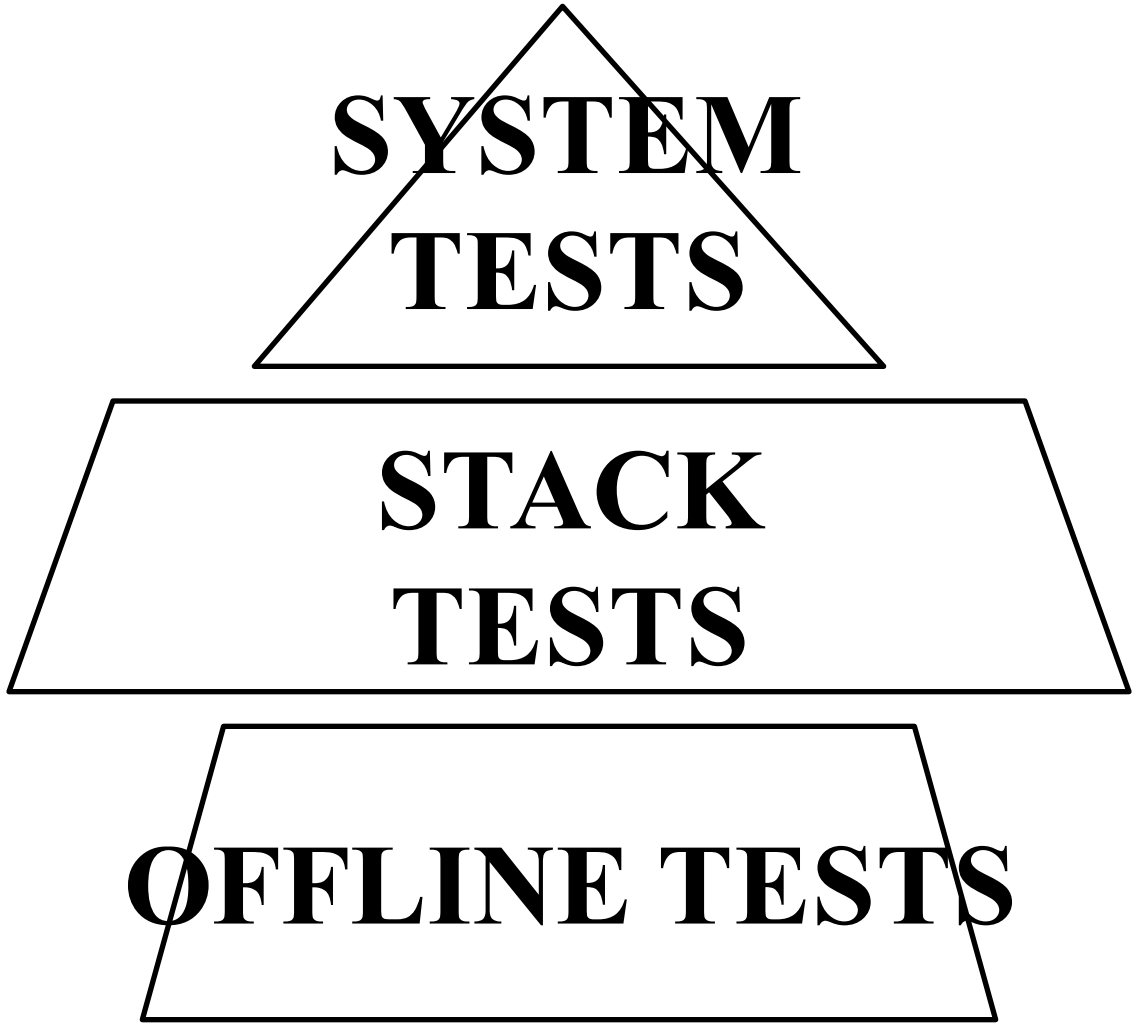
Discover issues quickly, so we can fix them quickly

- Validate as you work
- Continuously integrate all work in progress

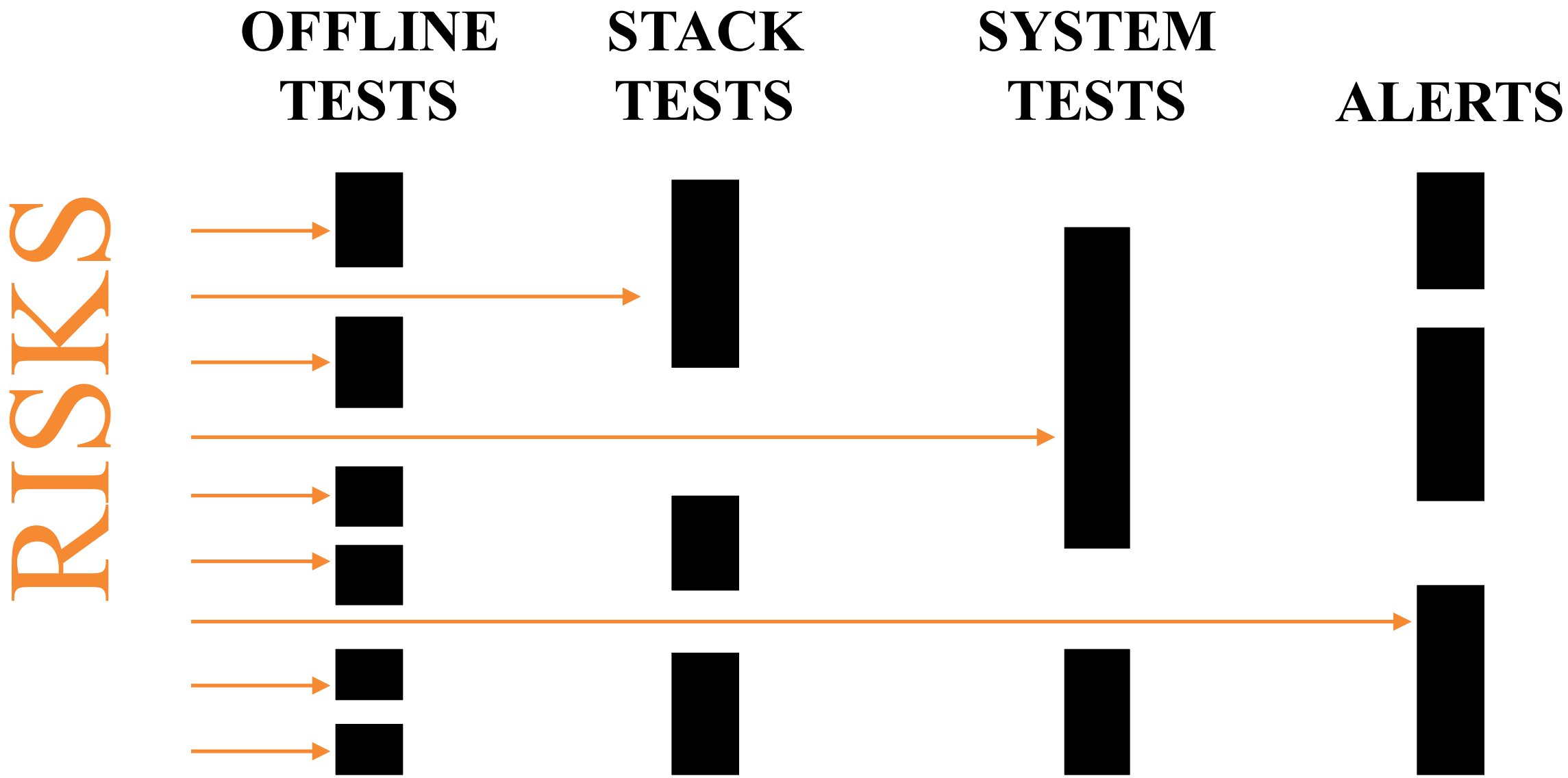
Classic test pyramid

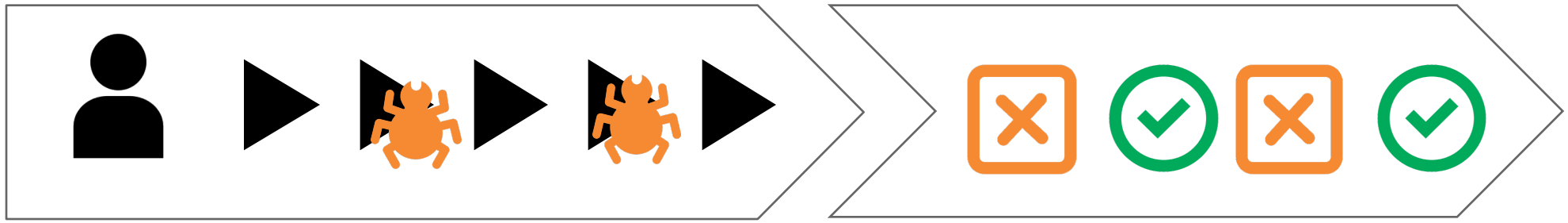


Infrastructure test pyramid??



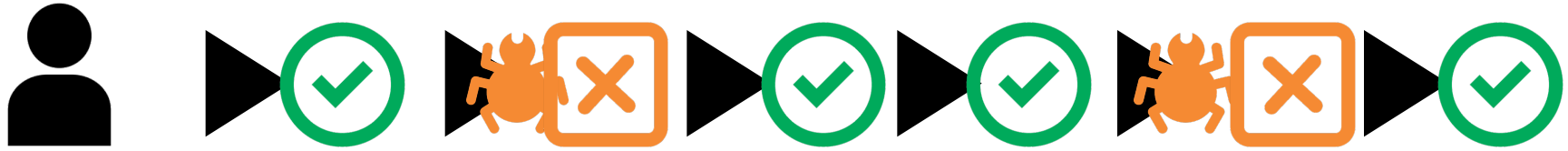
Swiss cheese testing model





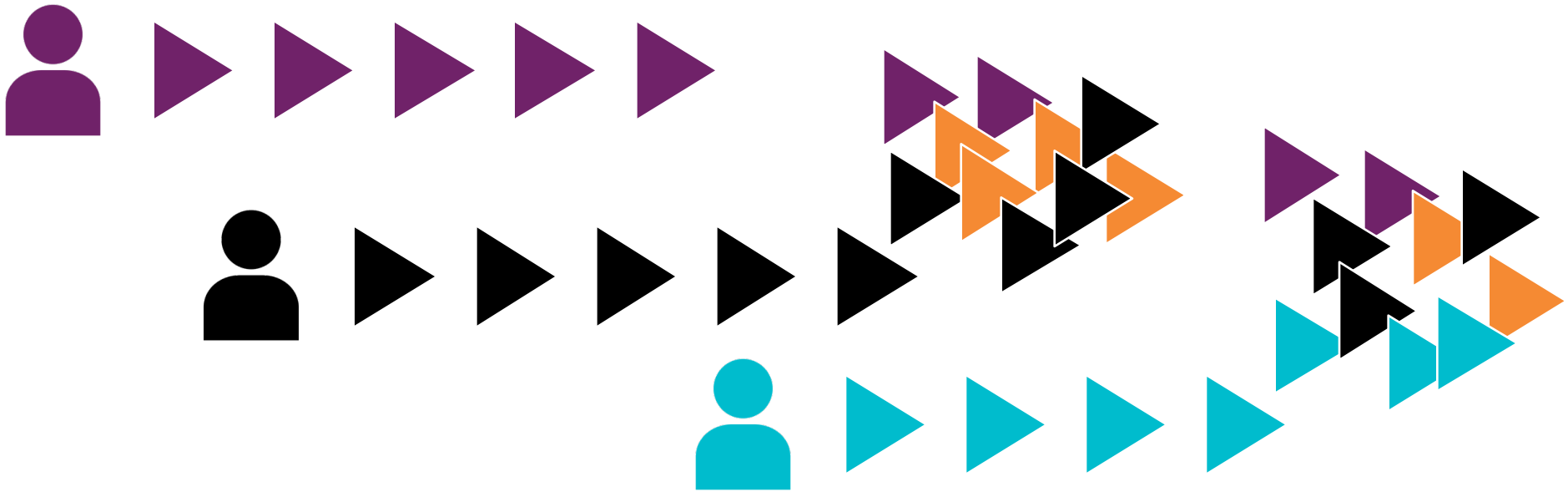
EVENTUAL TESTING

- Write all the code first
- Test later
- Fix ... sometime



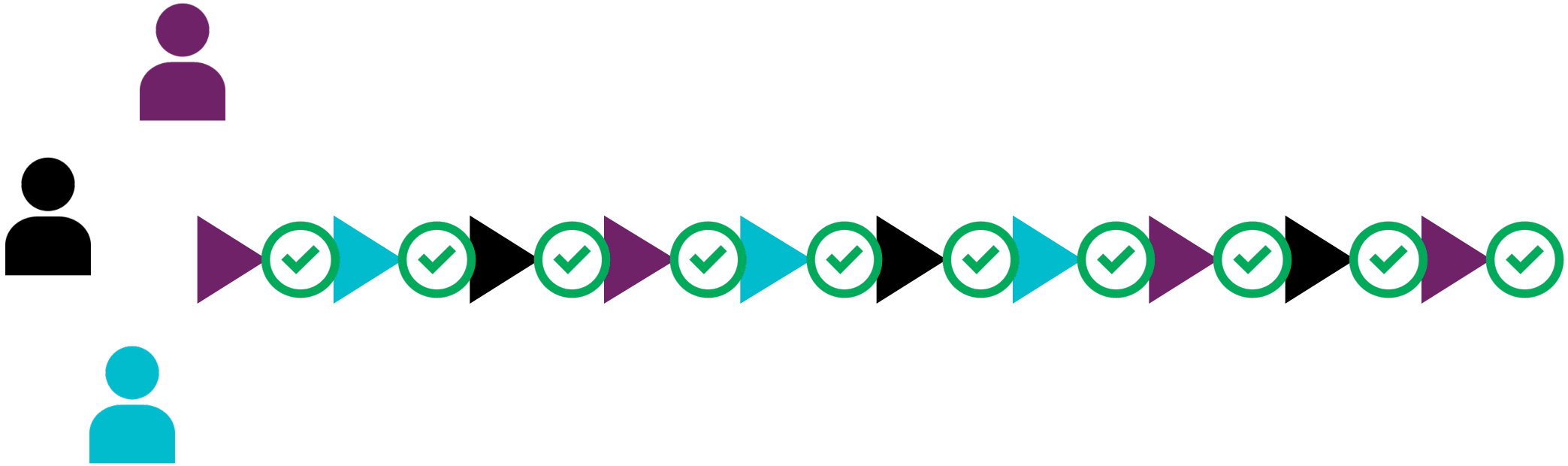
CONTINUOUS TESTING

- Test as you work
- Fix as you work
- Keep your code fully working



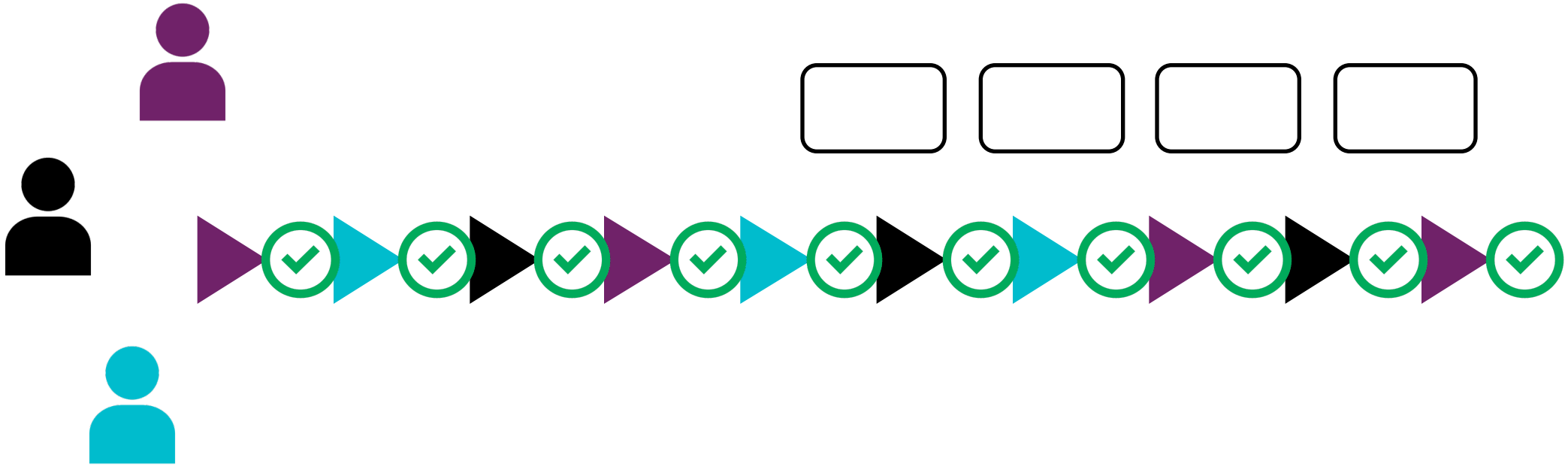
EVENTUAL INTEGRATION

- Each person or team works in isolation (e.g. on a branch)
- Integrate and test when "done"



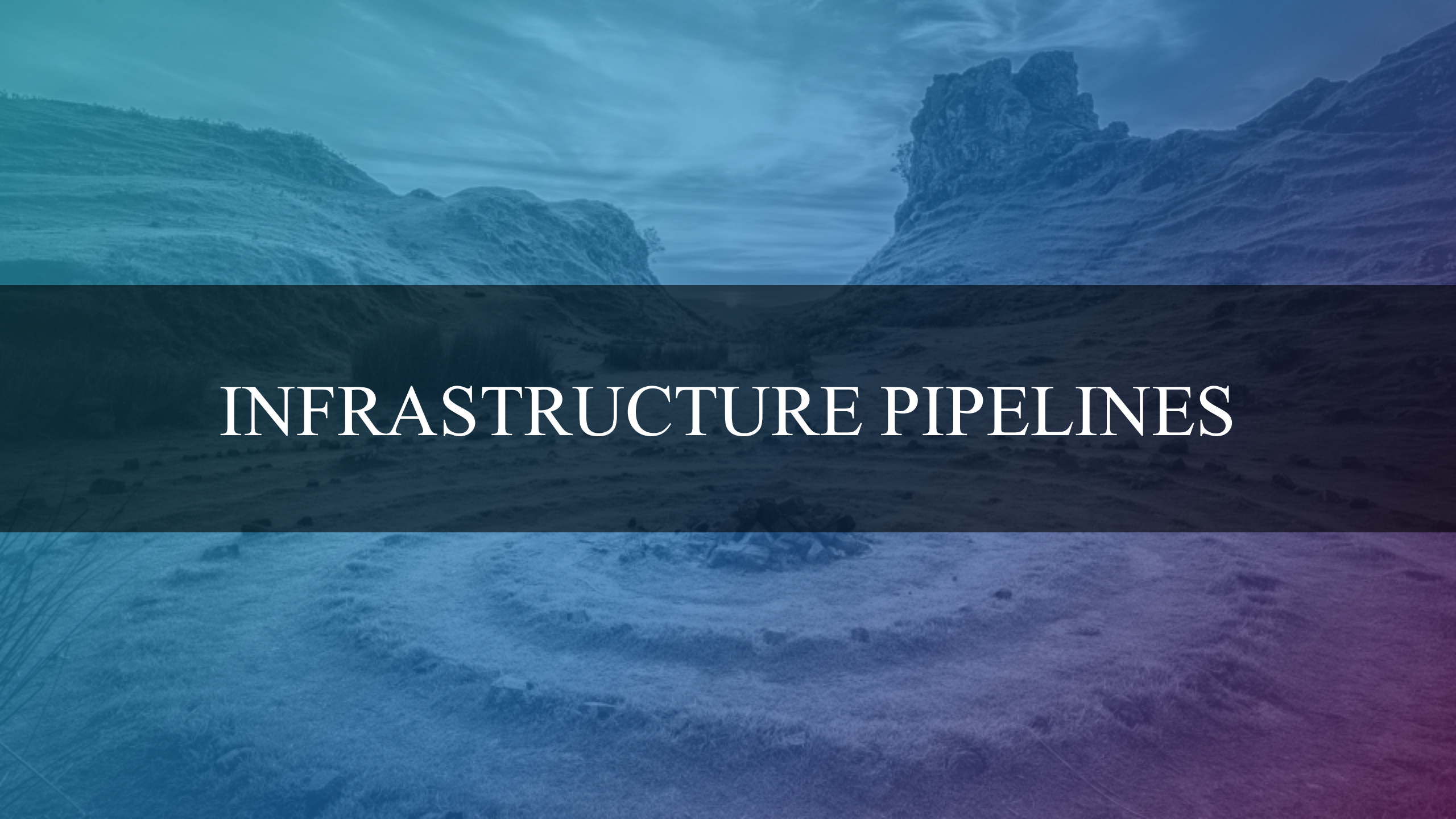
CONTINUOUS INTEGRATION

- Everyone integrates code into the same branch as they work
- (At least once per day)



CONTINUOUS DELIVERY

- Every change is fully integrated and tested
- The codebase is always kept production ready


INFRASTRUCTURE PIPELINES

GOAL

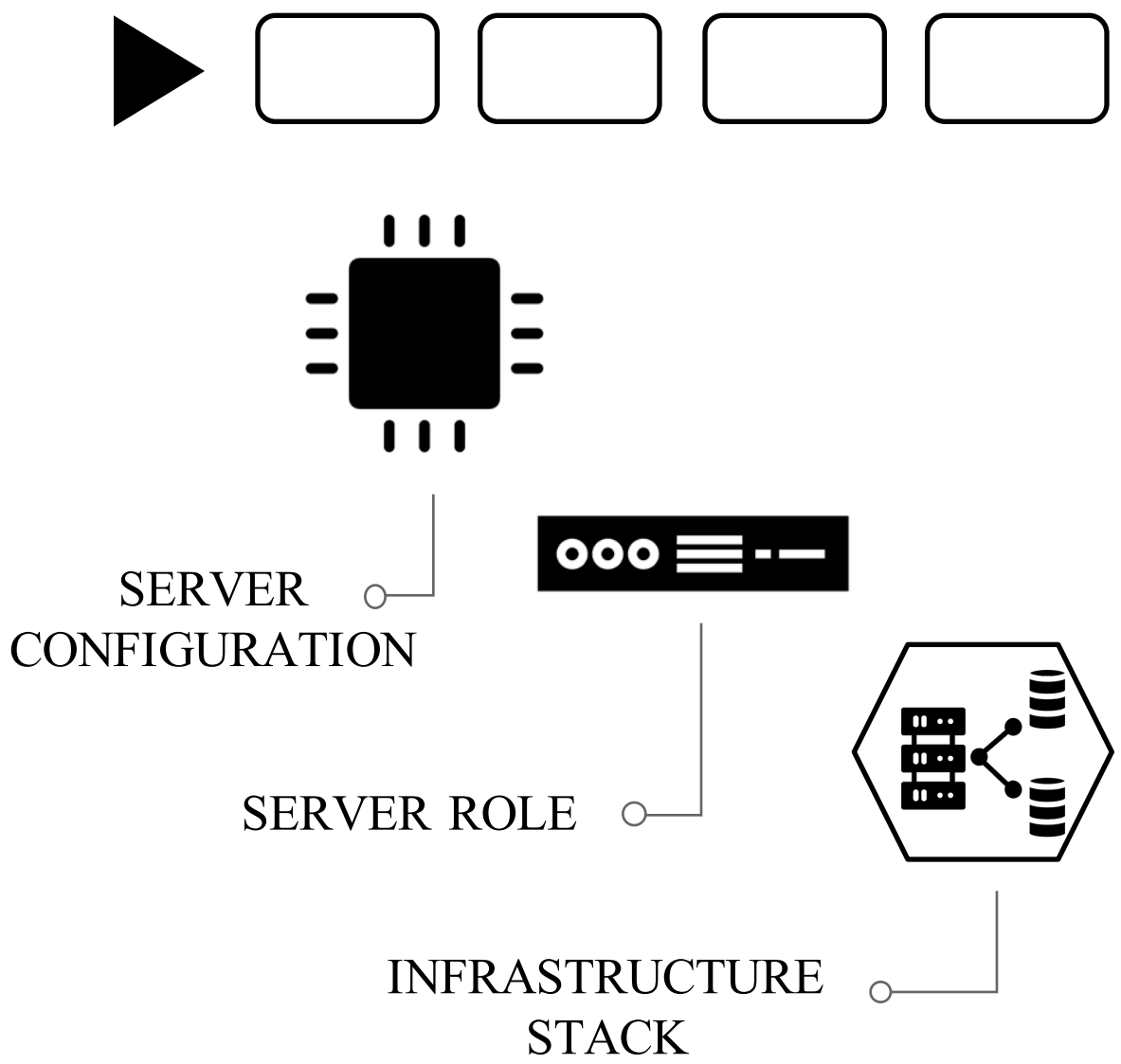
Progressively test and deliver changes

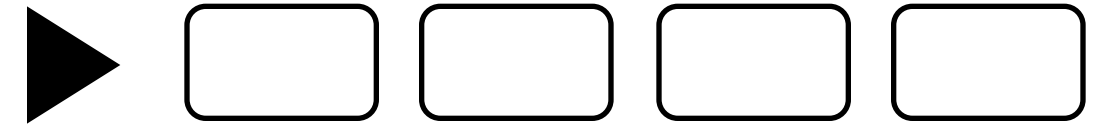
- Deliver changes easily
- Repeatable process
- Safety and visibility of changes

Pipeline stages

Each stage involves:

- **A set of components to be tested**

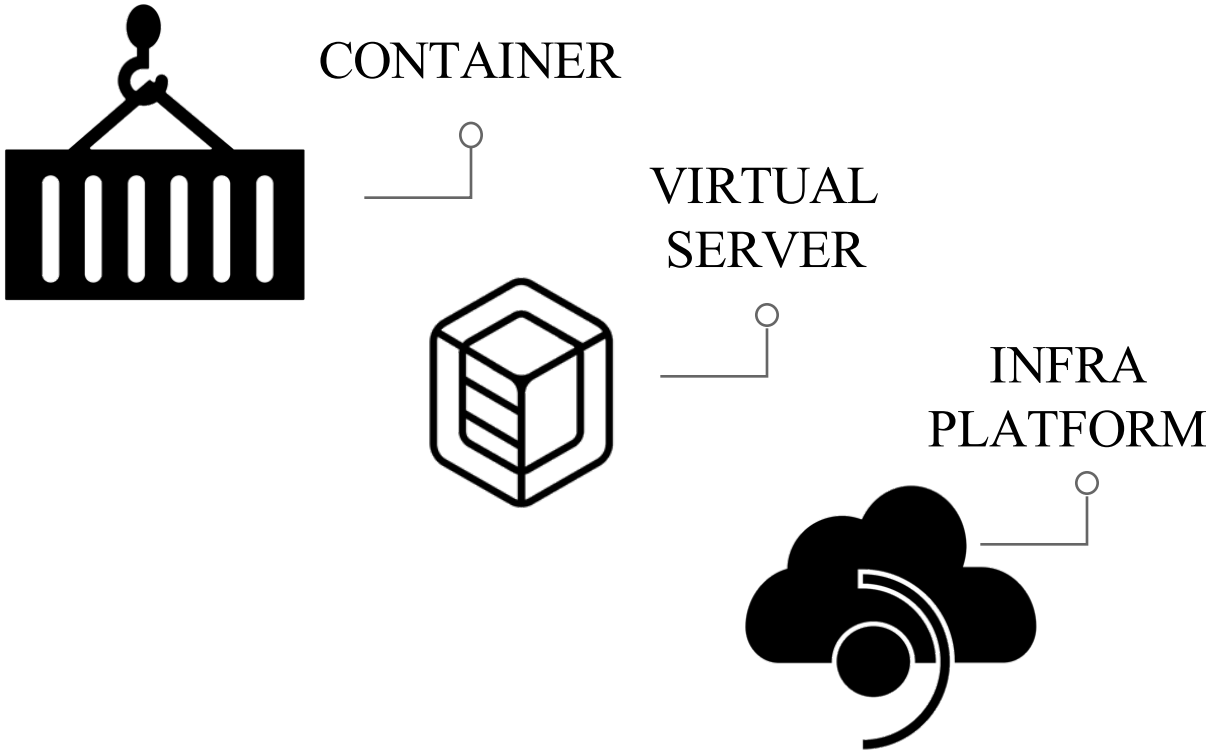


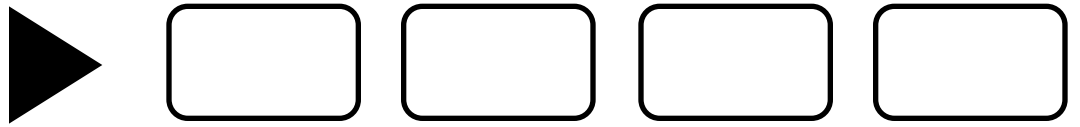


Pipeline stages

Each stage involves:

- A set of components to be tested
- **A context for the components**

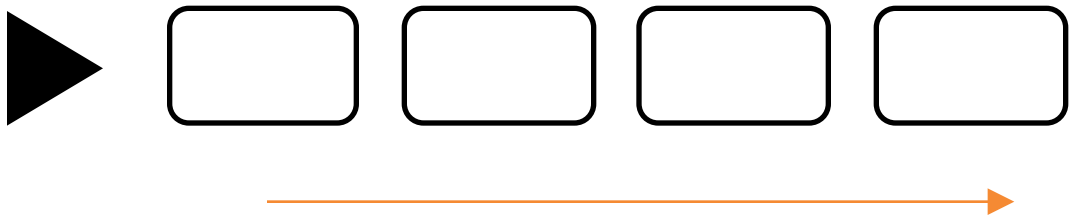




Pipeline stages

Each stage involves:

- A set of components to be tested
- A context for the components
- **A set of tests to run**



Progressive Testing

Types of progression

- **Time to prepare and run tests**

RUNS IN CONTAINER

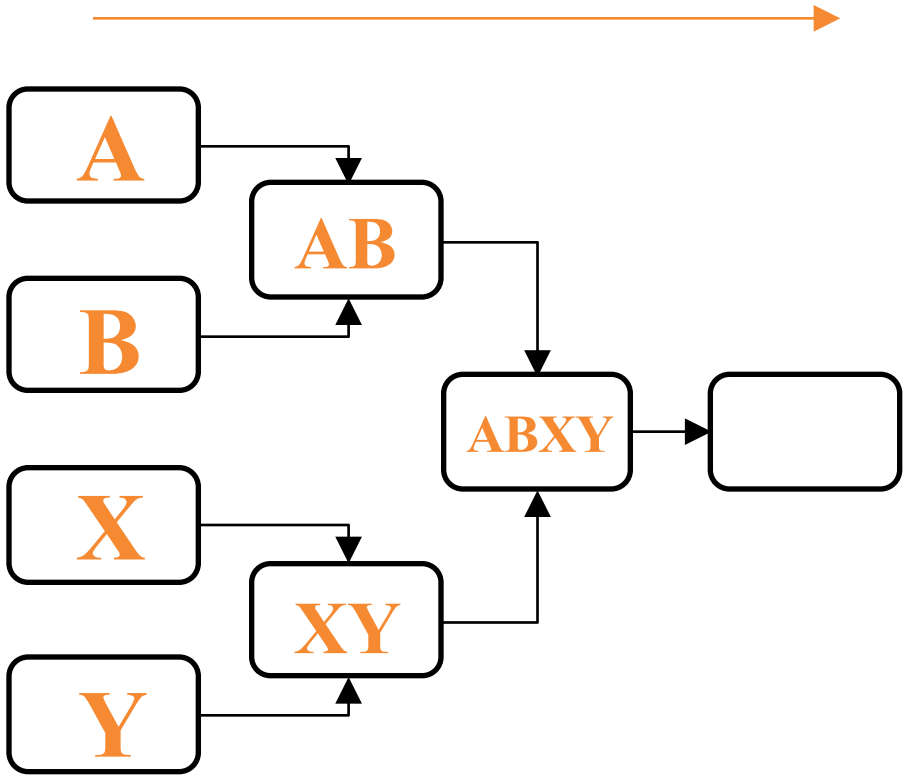
SELF-CONTAINED STACK

FULLY INTEGRATED

Progressive Testing

Types of progression

- Time to prepare and run tests
- **Scope of dependencies**



Progressive Testing

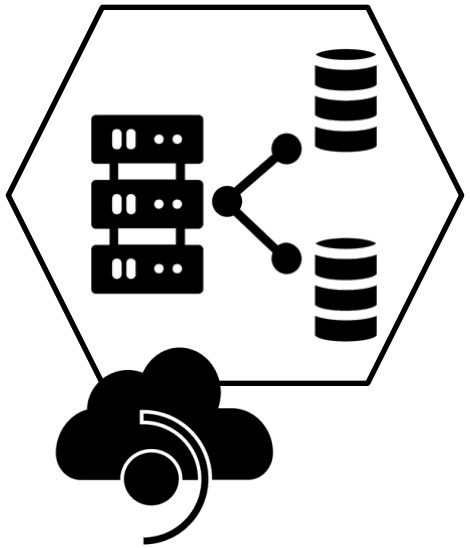
Types of progression

- Time to prepare and run tests
- Scope of dependencies
- **Scope of components being tested**

ONLINE AND OFFLINE TESTING STAGES
PLATFORM REQUIRED?

For a given test or set of tests:

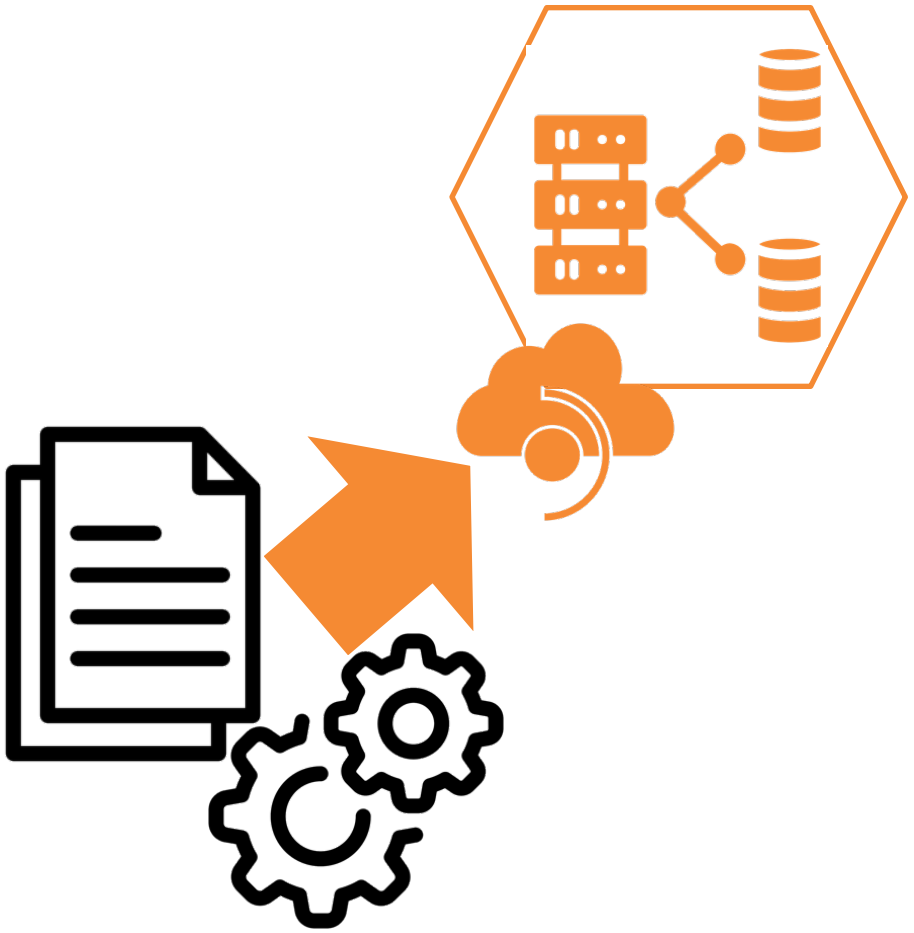
- You may need to provision "real" infrastructure in order to test
- Rather than testing locally, or within an agent
- But provisioning infrastructure takes much longer
- Testing without the infrastructure may not have value



OFFLINE TESTING

Test on local device or in an agent

- Mocking the platform API
- Running elements in local containers or VM
- Analysing the code

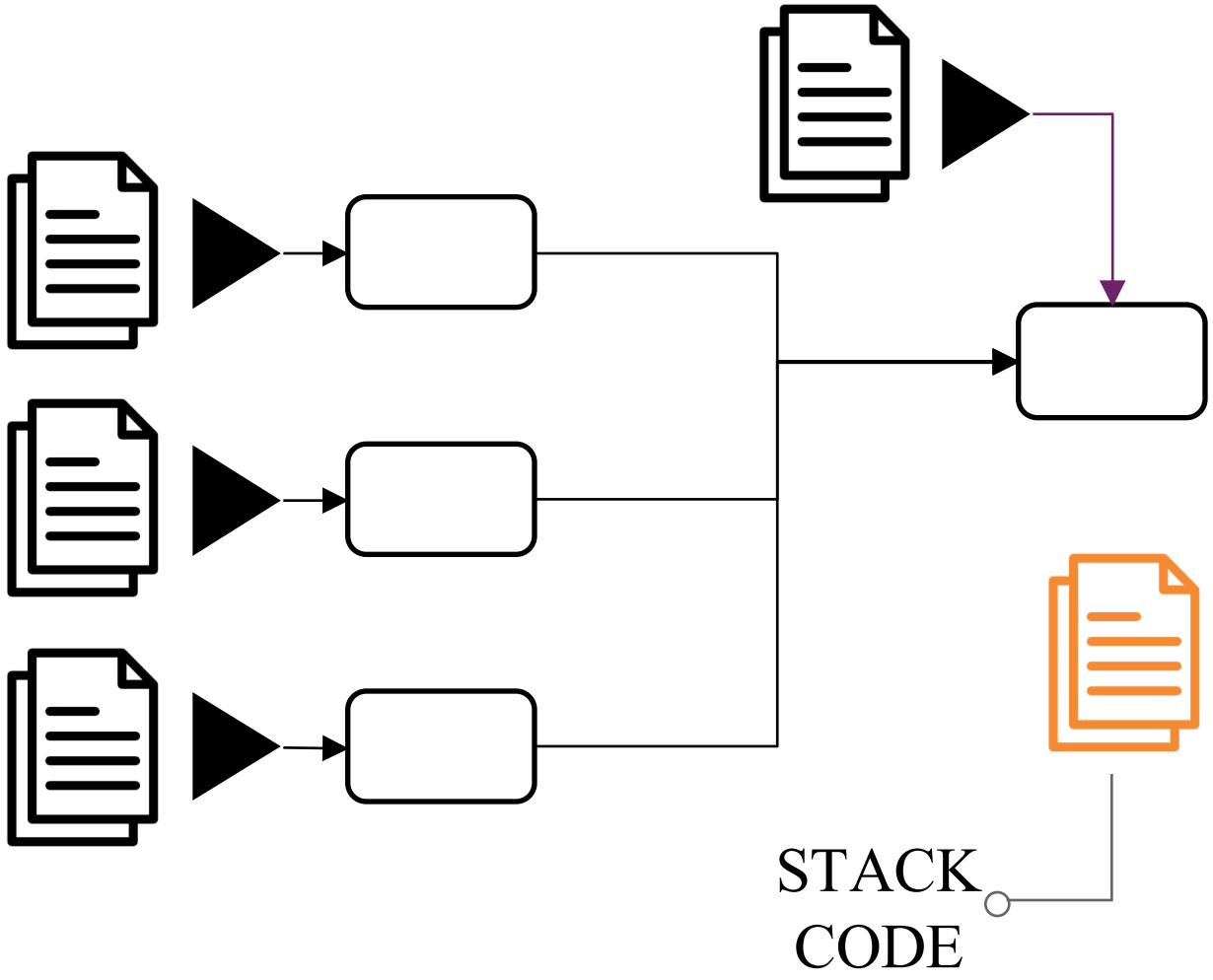


ONLINE TESTING

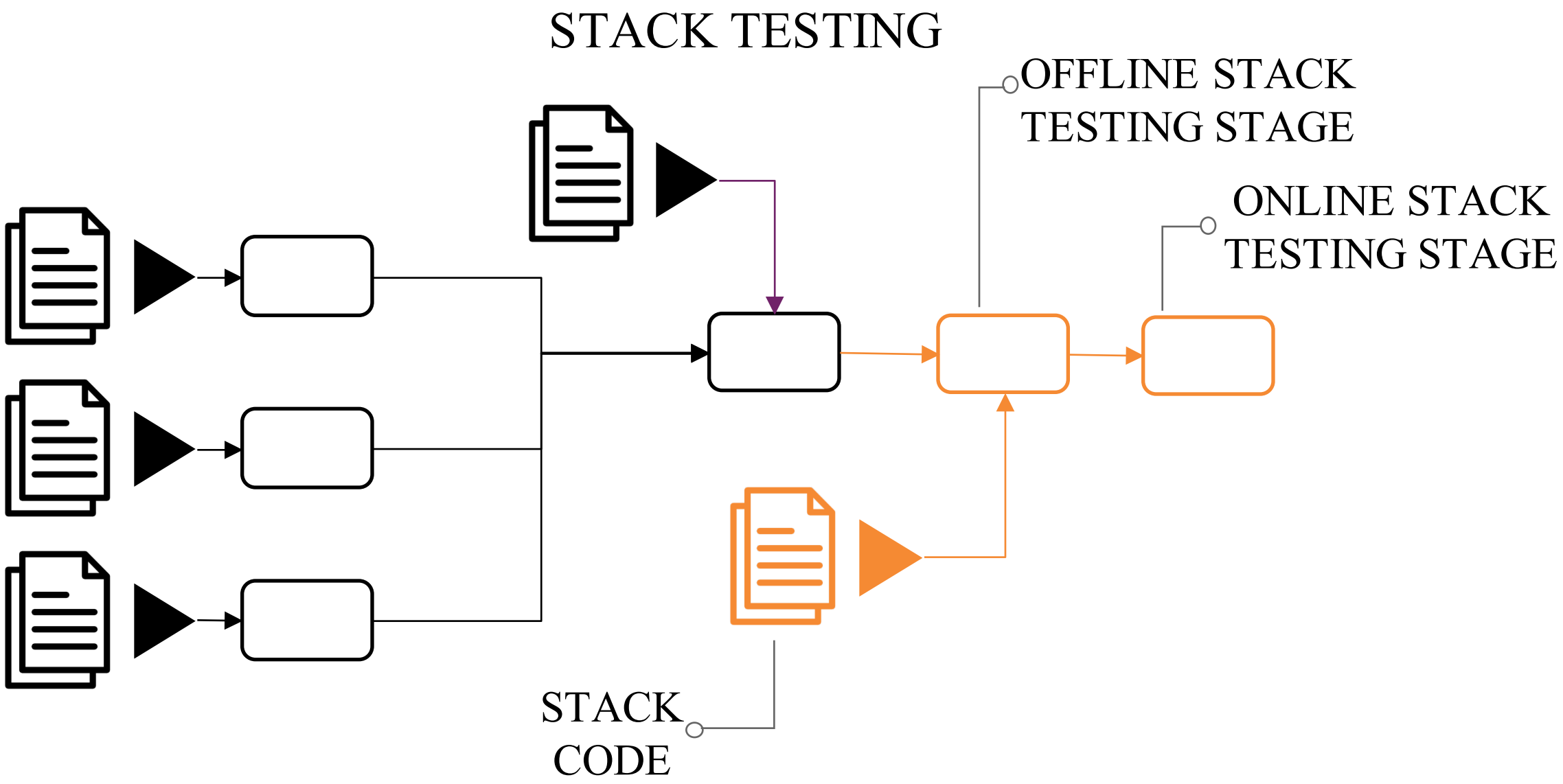
Provision infrastructure on the platform in order to test it

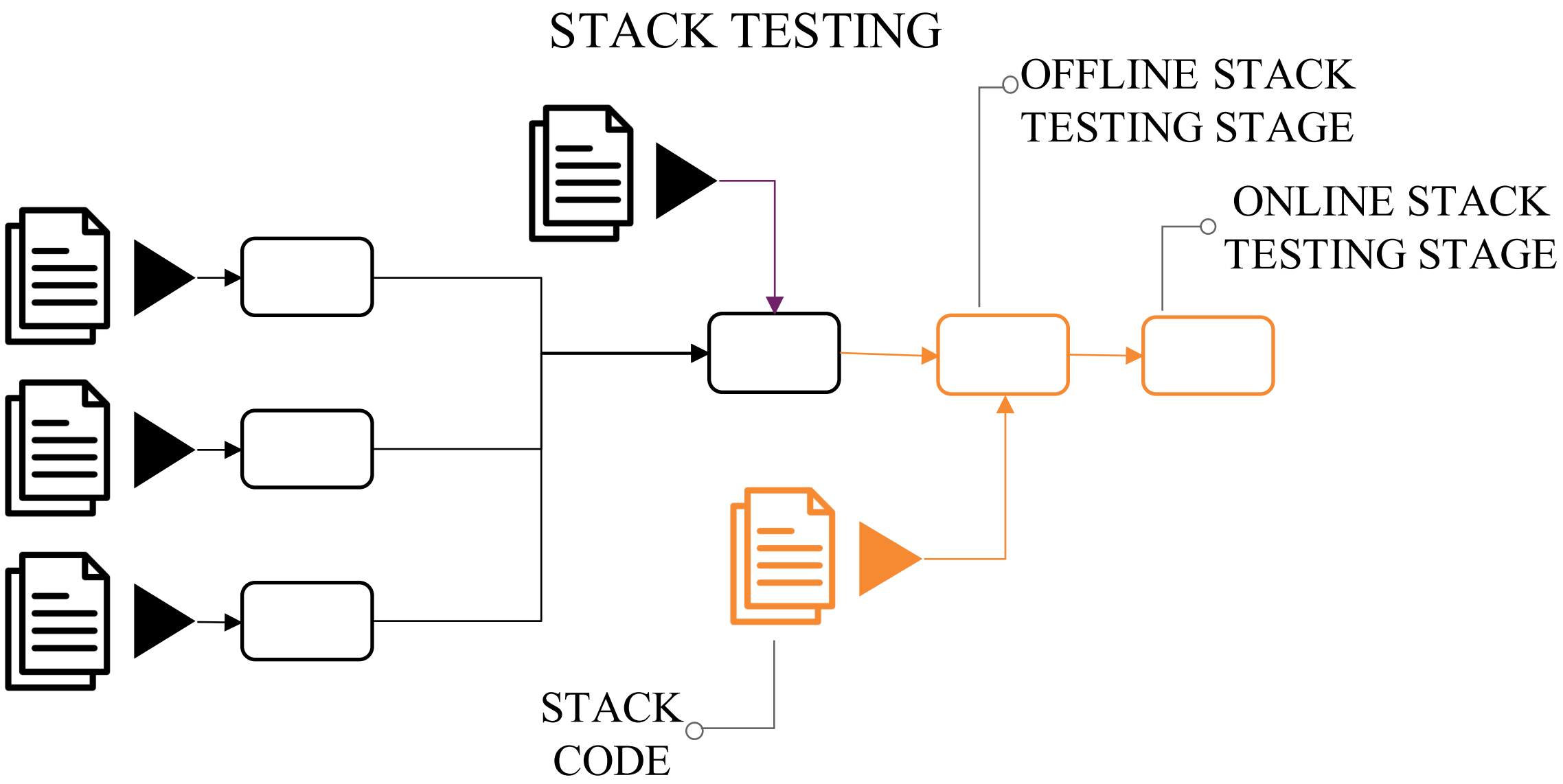
- Test the operation and correctness of infrastructure resources once provisioned
- Test with the elements of the system fully integrated

STACK TESTING




```
stack: app_infra
  vm: app_server
    role: app_server
    subnet: public
  database_instance: app_db
    type: mysql
    subnet: data
  subnet: public
    address: 10.0.1.0/16
  subnet: data
    address: 10.0.2.0/16
  route:
    from: internet
    to: app_server.ip
    port: 443
  route:
    from: app_server.ip
    to: app_db.ip
    port: 3306
```

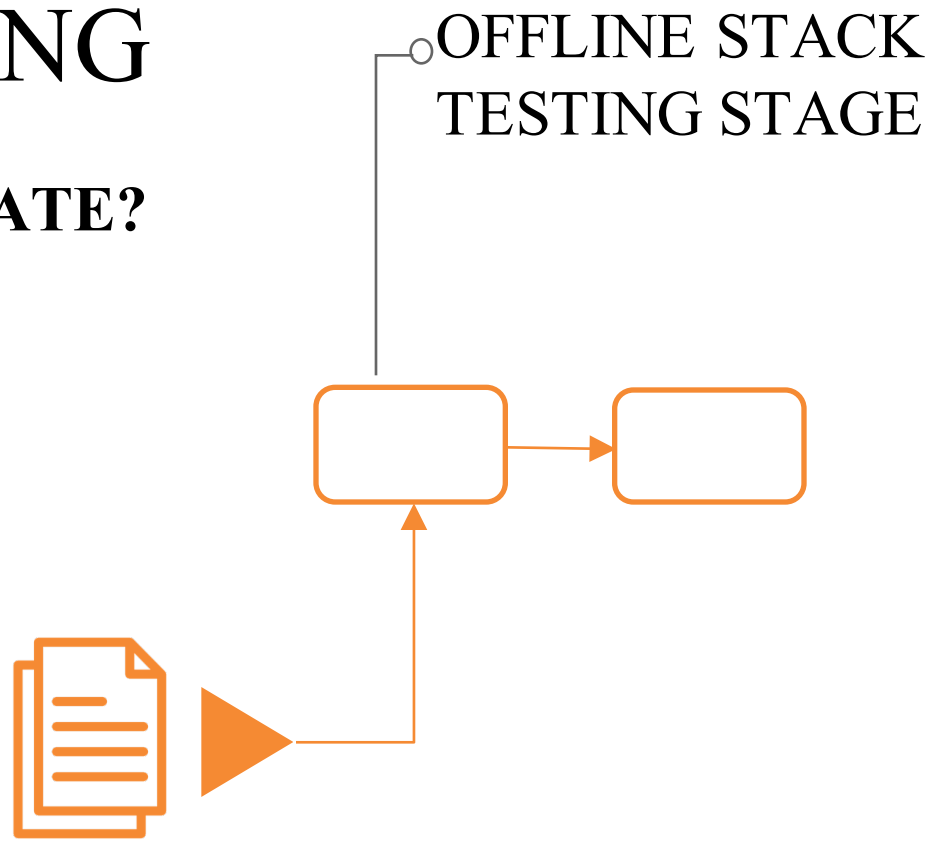

STACK TESTING
OFFLINE STACK
TESTING STAGE
ONLINE STACK
TESTING STAGE
STACK
CODE


STACK TESTING
OFFLINE STACK
TESTING STAGE
ONLINE STACK
TESTING STAGE
STACK
CODE

OFFLINE STACK TESTING

WHAT WE CAN VALIDATE?

- Syntax
- Linting
- Mocked provisioning



SOME OFFLINE STACK TESTING TOOLS

- cfn_nag
- tflint
- terraform_validate
- terraform-compliance

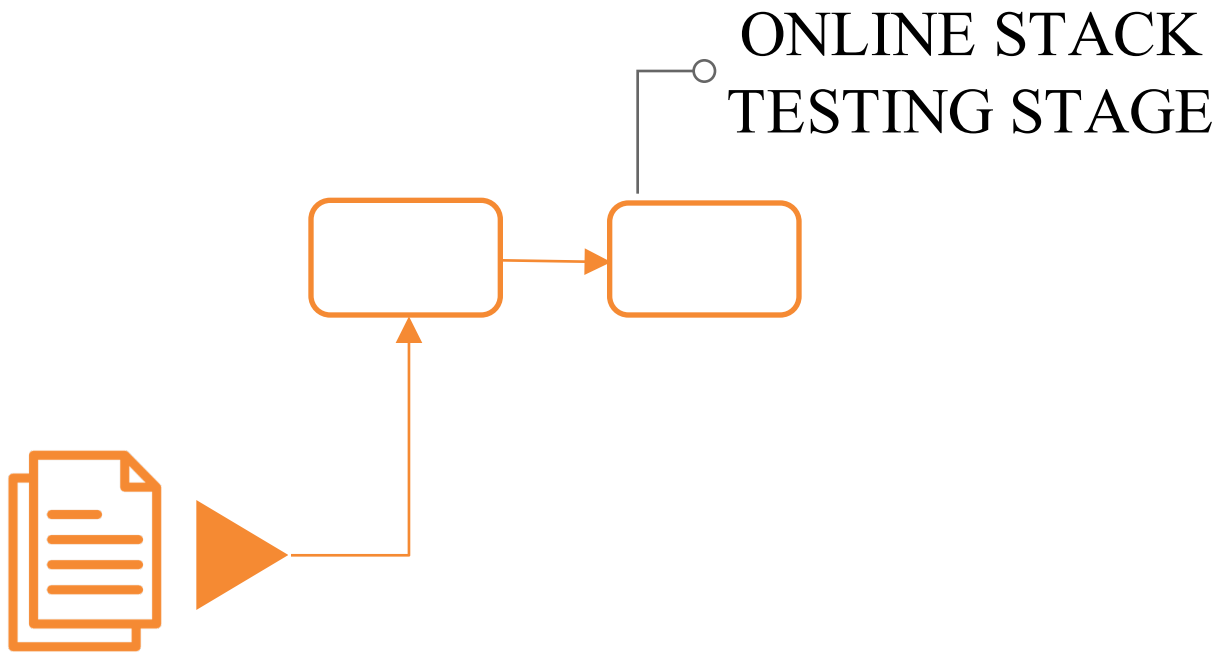
SOME CLOUD PLATFORM MOCKING TOOLS

- Localstack (AWS)
- Moto (AWS)

ONLINE STACK TESTING

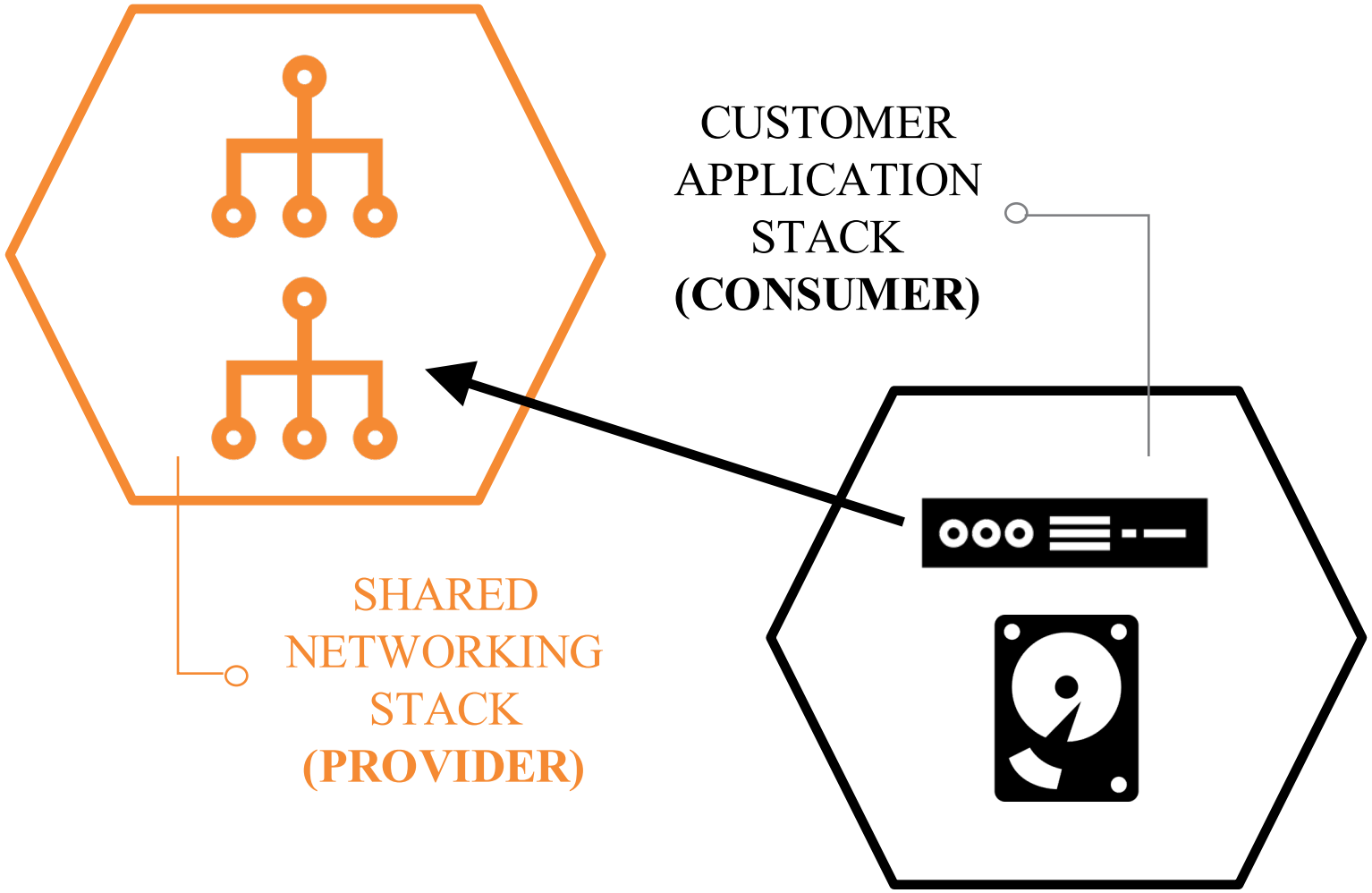
SOME TOOLS:

- awspec
- inspec
- taskcat
- terratest

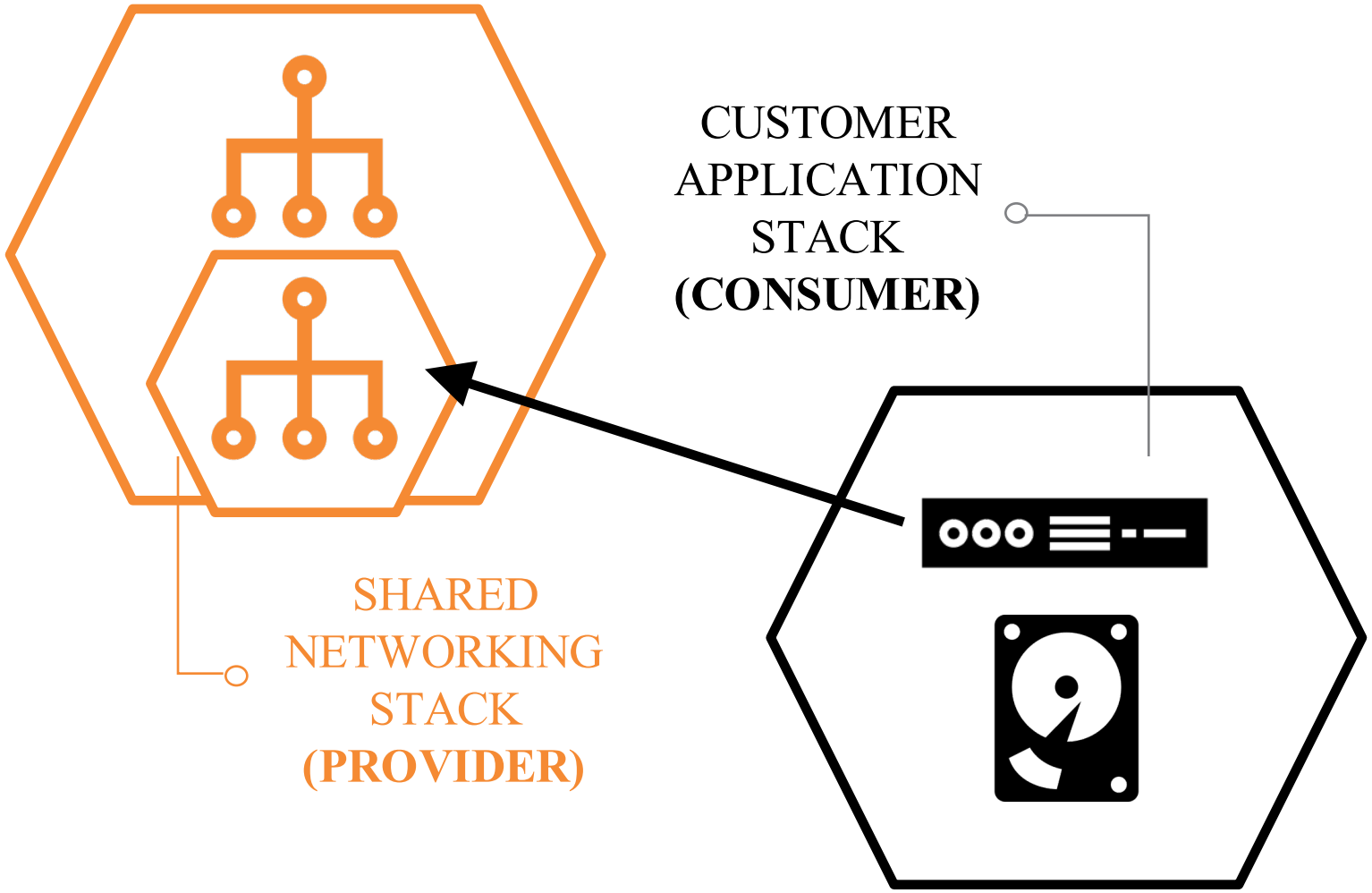


TEST FIXTURES

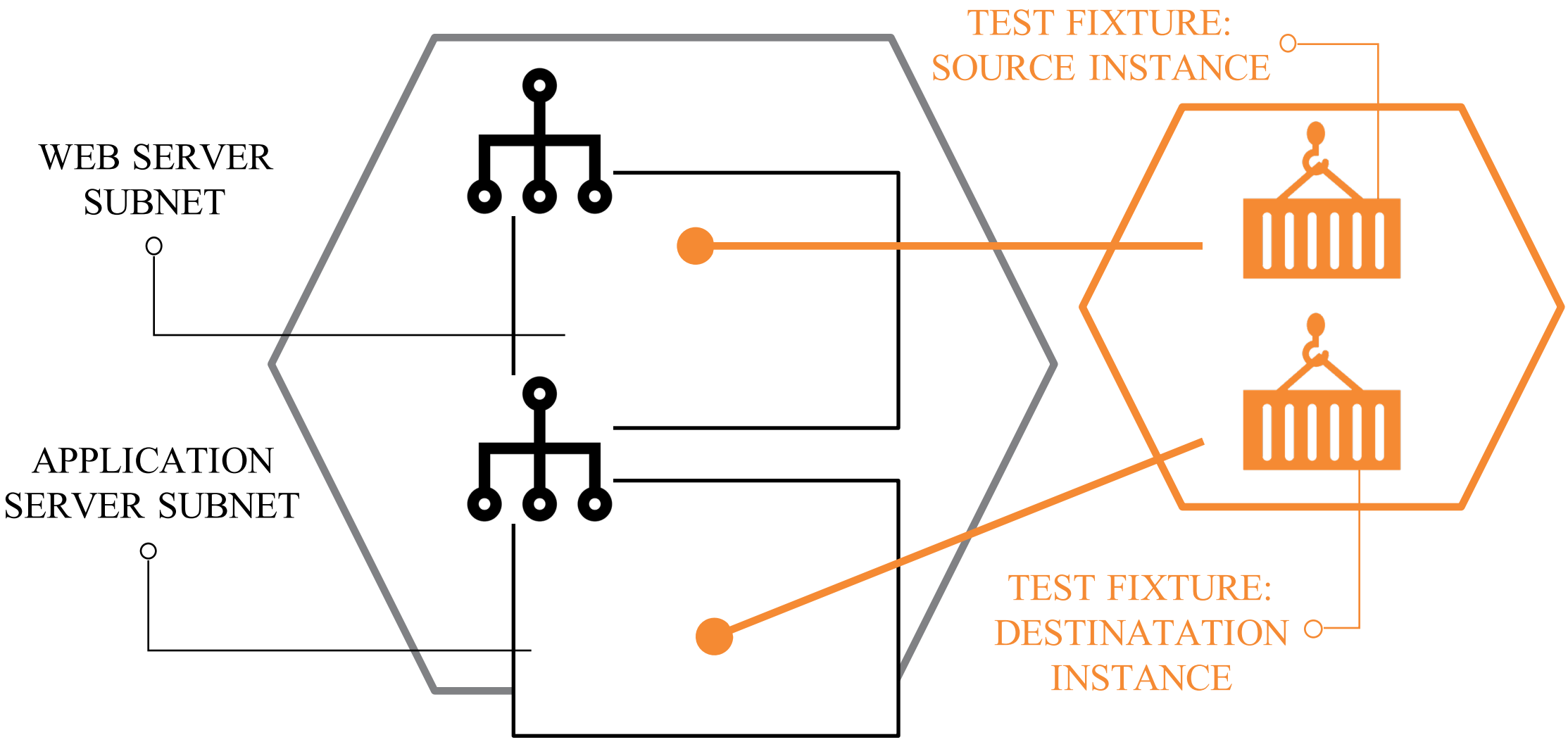
FAKING A PROVIDER



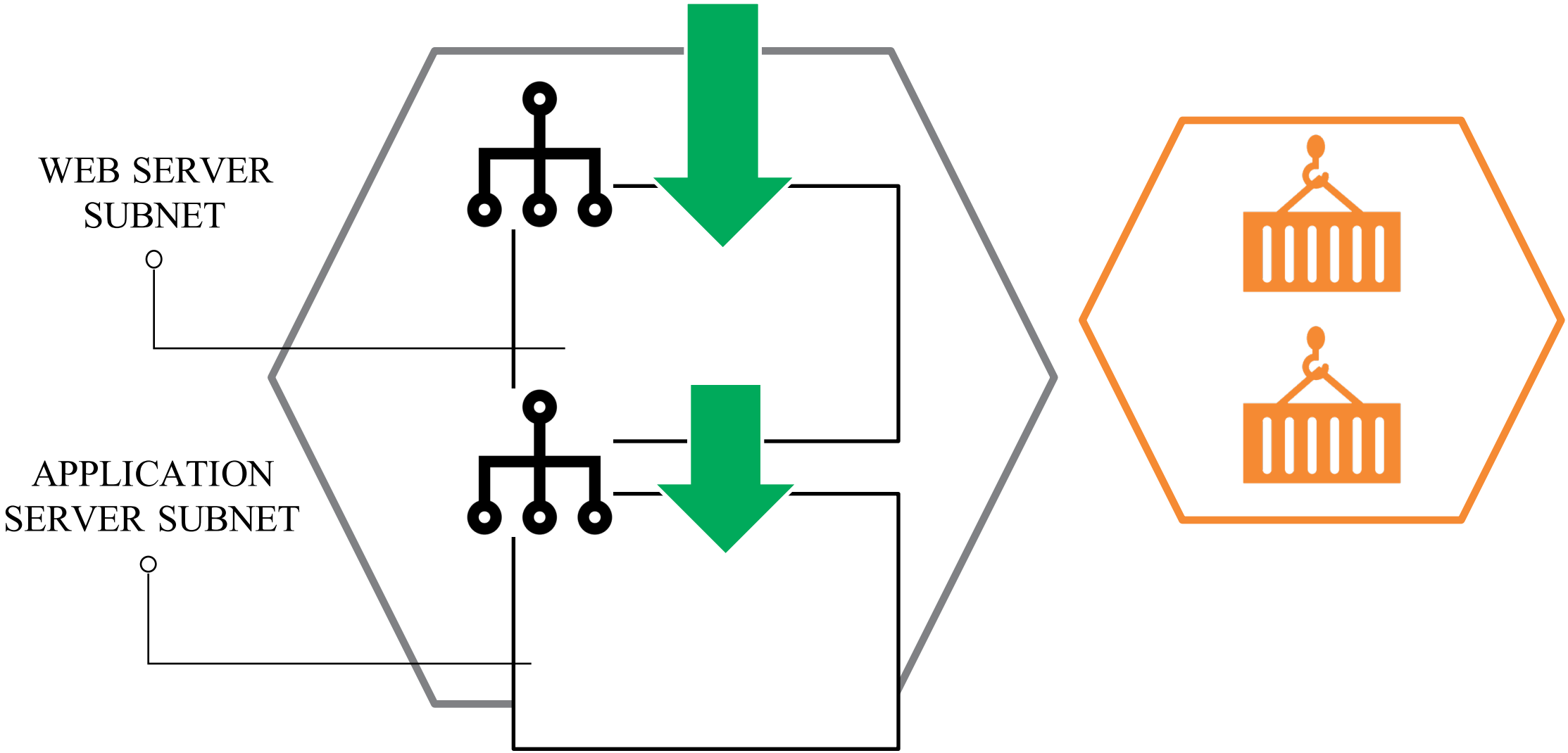
FAKING A PROVIDER



TESTING A PROVIDER



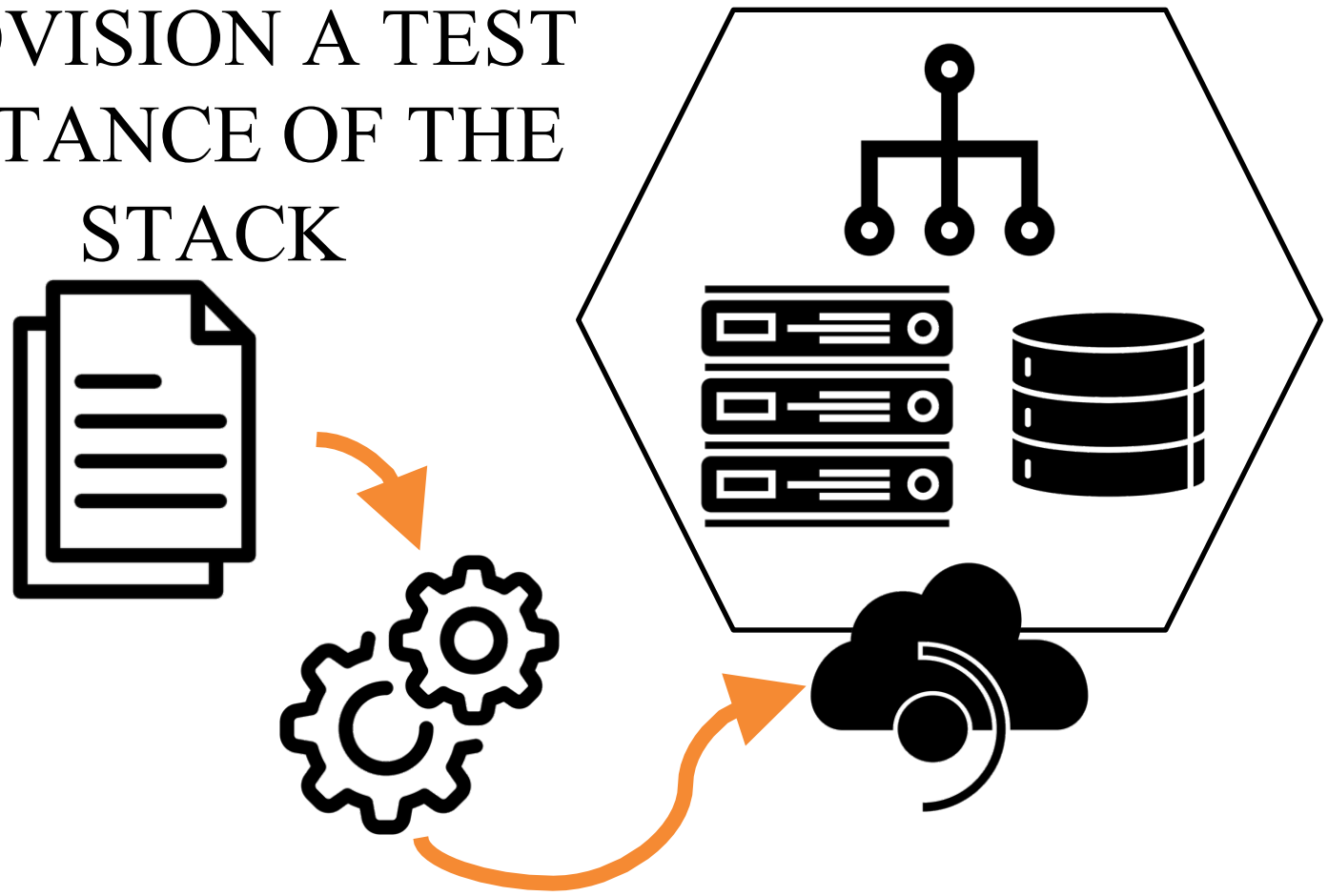
TESTING A PROVIDER



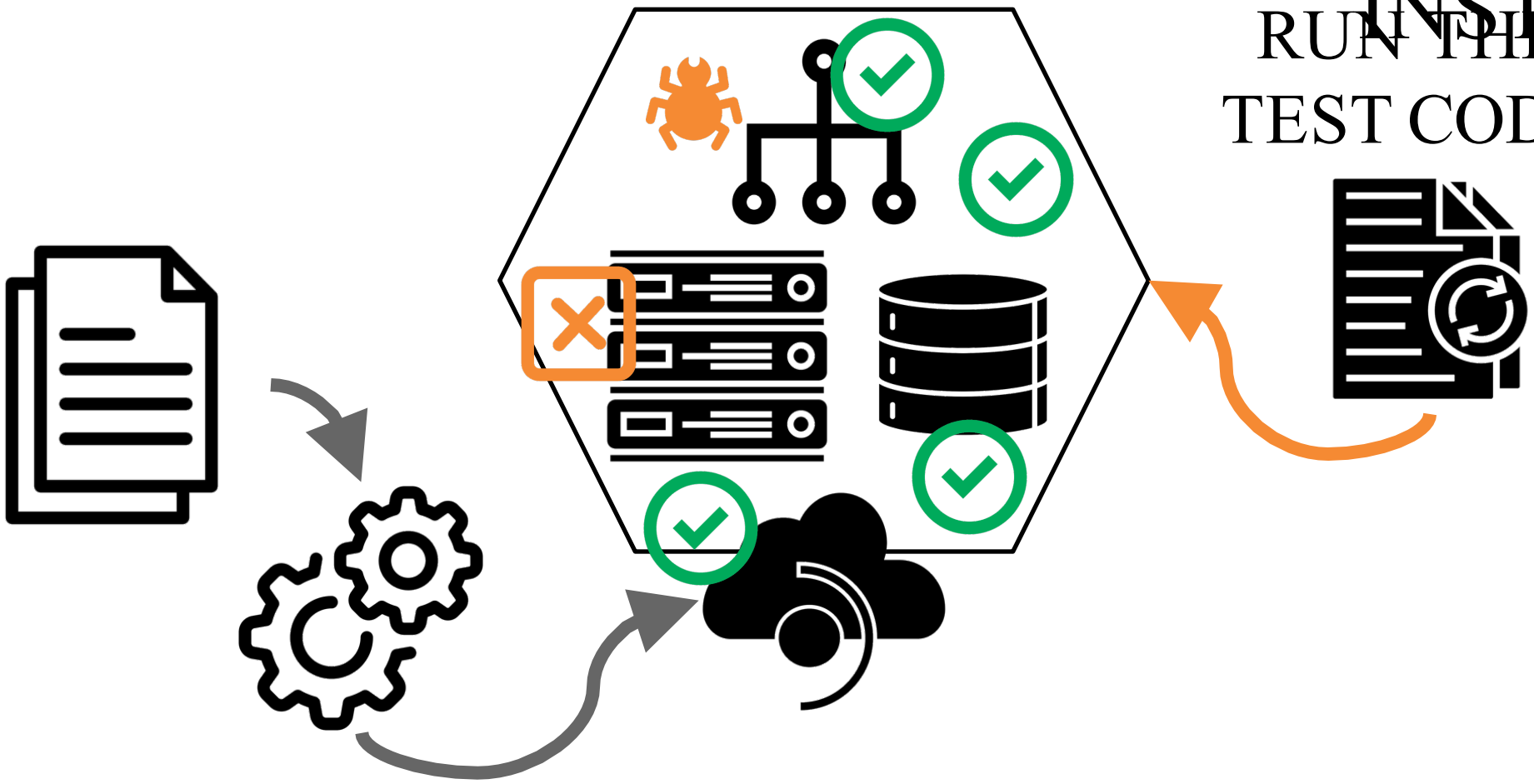

TEST INSTANCE LIFECYCLES

TESTING WITH AN EPHEMERAL STACK INSTANCE

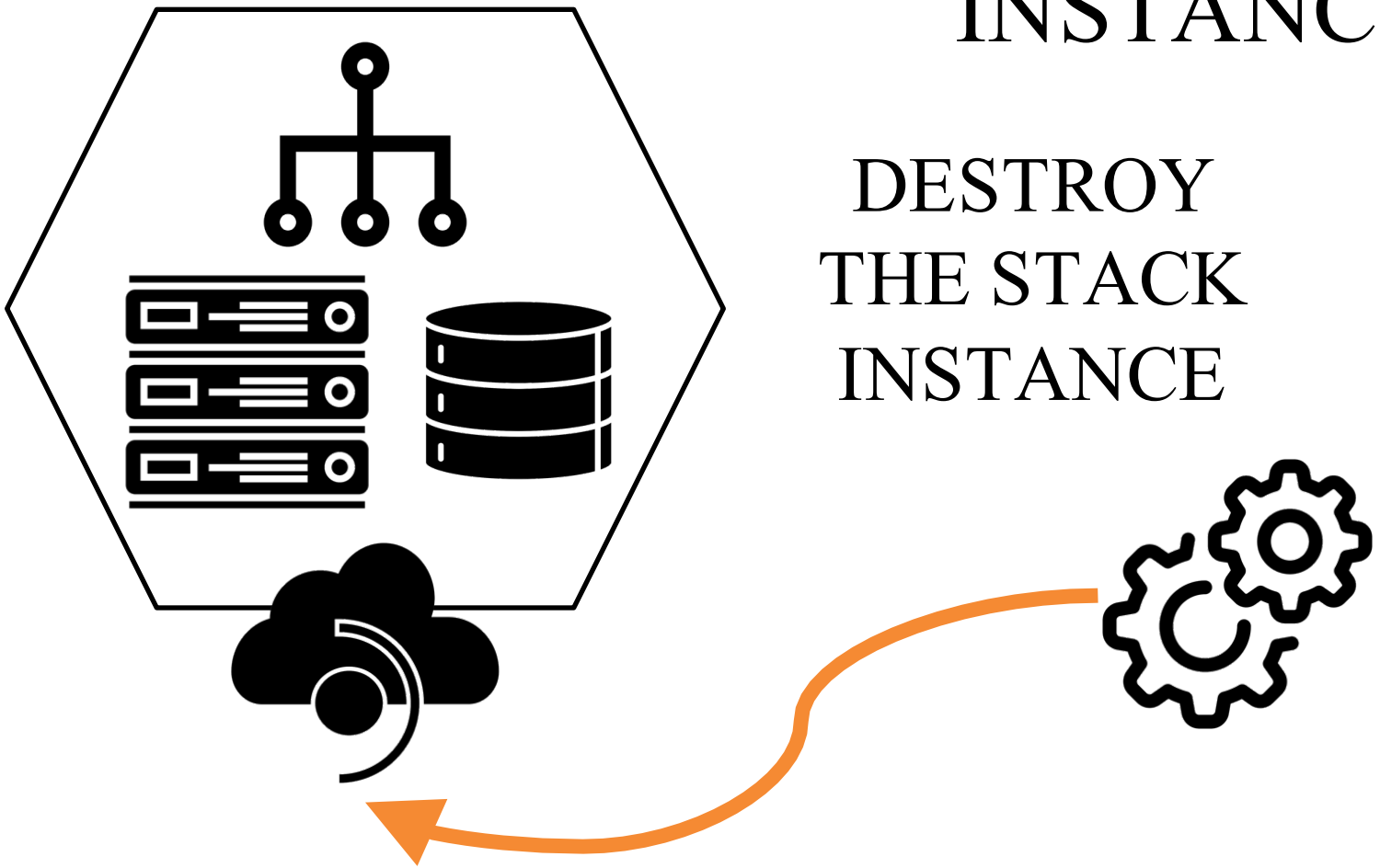
PROVISION A TEST INSTANCE OF THE STACK



TESTING WITH AN EPHEMERAL STACK INSTANCE



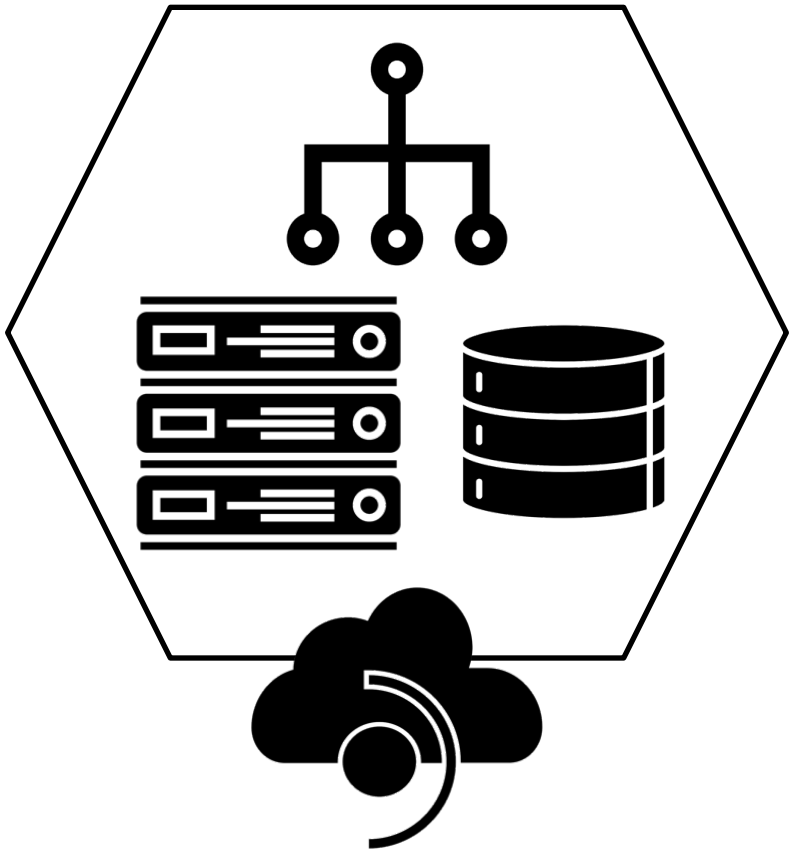
TESTING WITH AN EPHEMERAL STACK INSTANCE



TESTING WITH AN EPHEMERAL STACK INSTANCE

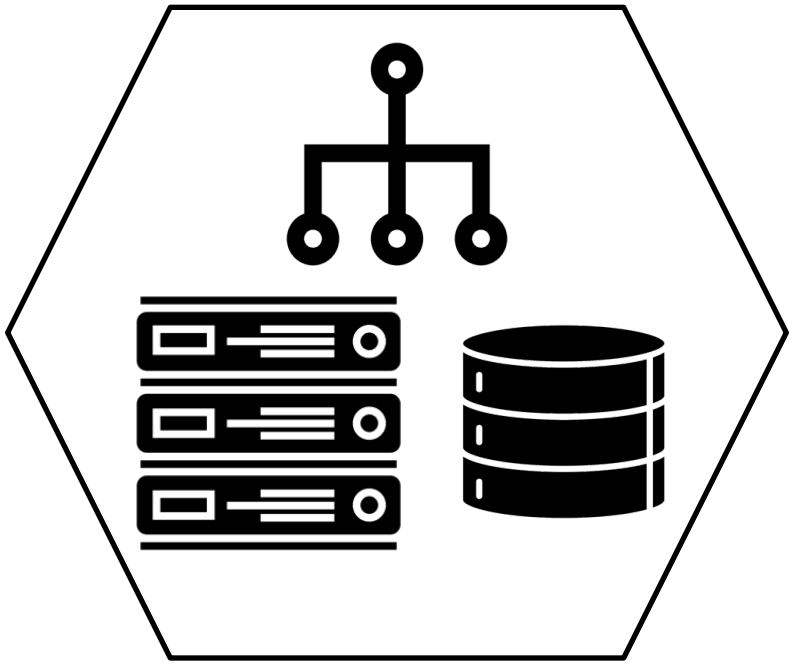
BENEFITS:

- Tests a clean stack every time
- Proves the process for provisioning from scratch



DRAWBACKS:

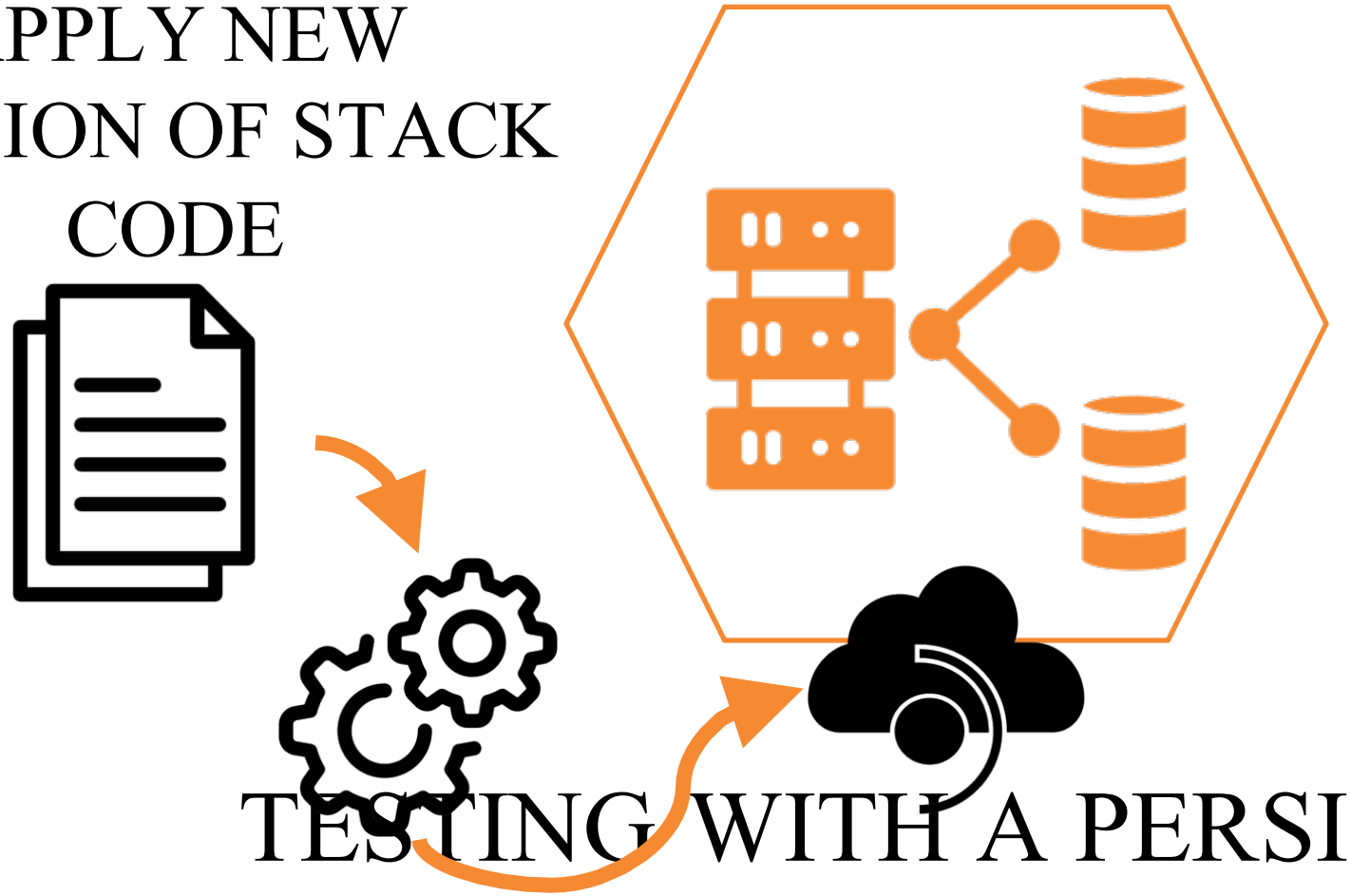
- Provisioning can make this very slow
- Doesn't prove the process for updating existing infrastructure



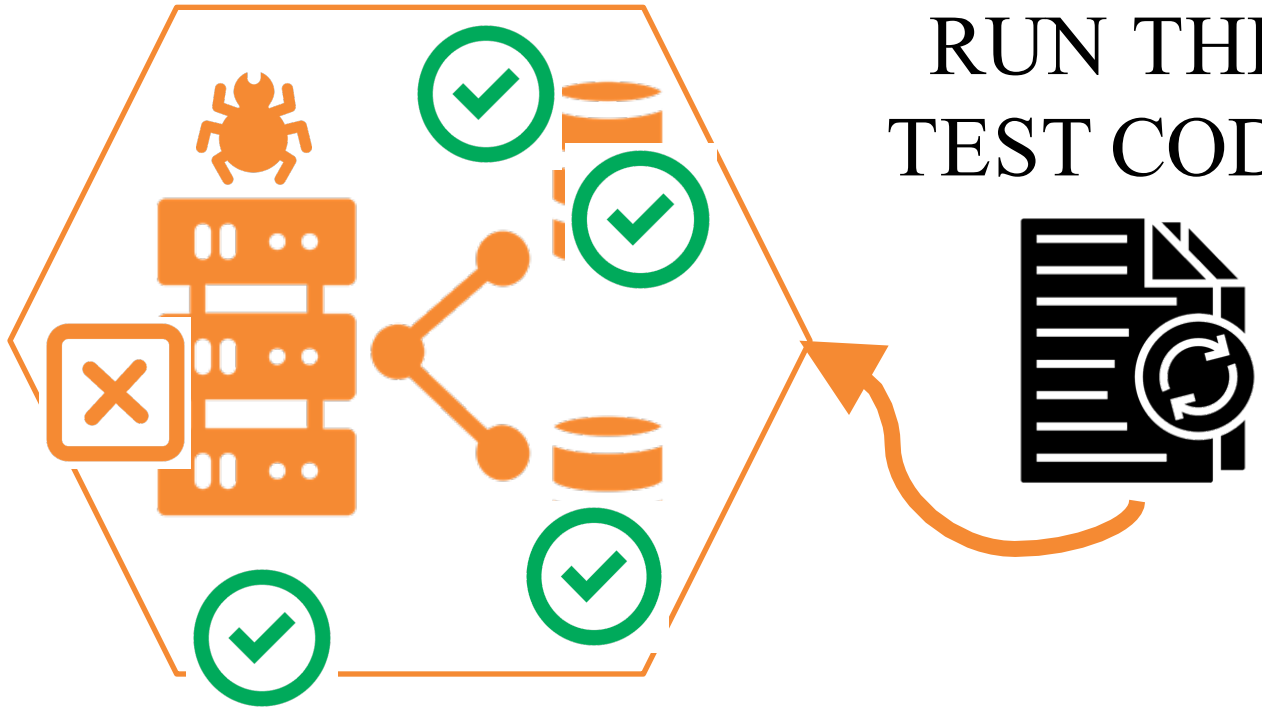
TEST INSTANCE
ALREADY EXISTS

TESTING WITH A PERSISTENT STACK INSTANCE

APPLY NEW VERSION OF STACK CODE



TESTING WITH A PERSISTENT STACK INSTANCE



TESTING WITH A PERSISTENT STACK INSTANCE

LEAVE THE ENVIRONMENT IN PLACE

TESTING WITH A PERSISTENT STACK INSTANCE

BENEFITS:

- Faster than provisioning a new stack
- Proves the process for updating existing infrastructure

DRAWBACKS:

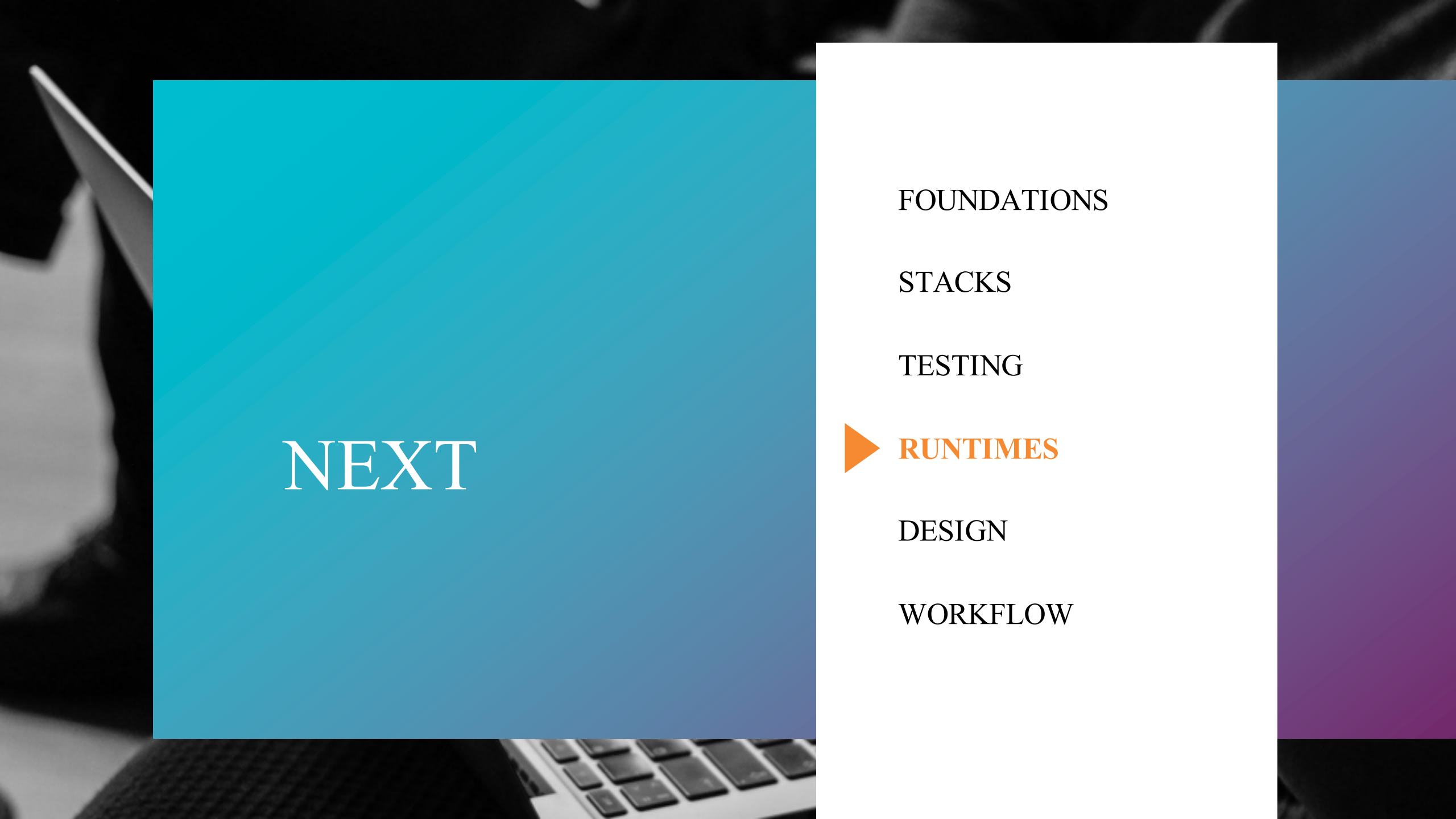
- May leave the stack broken, require manual intervention to fix
- More expensive – test environment is always running

TESTING WITH A PERSISTENT STACK INSTANCE

MIXING EPHEMERAL AND PERSISTENT STACK TESTING

APPROACHES

- Have stages for both in your pipeline (*quick & dirty*, *slow & clean*)
- Run ephemeral tests periodically to clean up
- Rebuild environments to known version after each test run



NEXT

- FOUNDATIONS
- STACKS
- TESTING
- **RUNTIMES**
- DESIGN
- WORKFLOW