





Learning Go


Test Double














Somkiat

[Home](#)










Somkiat Puisungnoen

[Update Info](#) 1
[View Activity Log](#) 10+
...


[Timeline](#)
[About](#)
[Friends](#) 3,138
[Photos](#)
[More](#)



When did you work at Opendream?
×





...
22 Pending Items



Intro


Software Craftsmanship


Software Practitioner at สยามชำนานกิจ พ.ศ. 2556



Agile Practitioner and Technical at SPRINT3r


Post

Photo/Video

Live Video

Life Event


What's on your mind?


Public

Post



Somkiat Puisungnoen
15 mins · Bangkok ·

Java and Bigdata



Facebook interface for the page **somkiat.cc**. The top navigation bar includes the Facebook logo, the page name, a search bar, and icons for Home, Messages, Notifications (3), Insights, Publishing Tools, Settings, and Help.

The main content area features a large video player showing a man in a white Superman t-shirt with "SOMKIAT.CC" printed on it, posing against a white wall. A blue call-to-action button is overlaid on the video: "Help people take action on this Page." with a close icon (X). Below the video, there are buttons for "Liked", "Following", "Share", and a menu icon (three dots). A blue button labeled "+ Add a Button" is also visible.

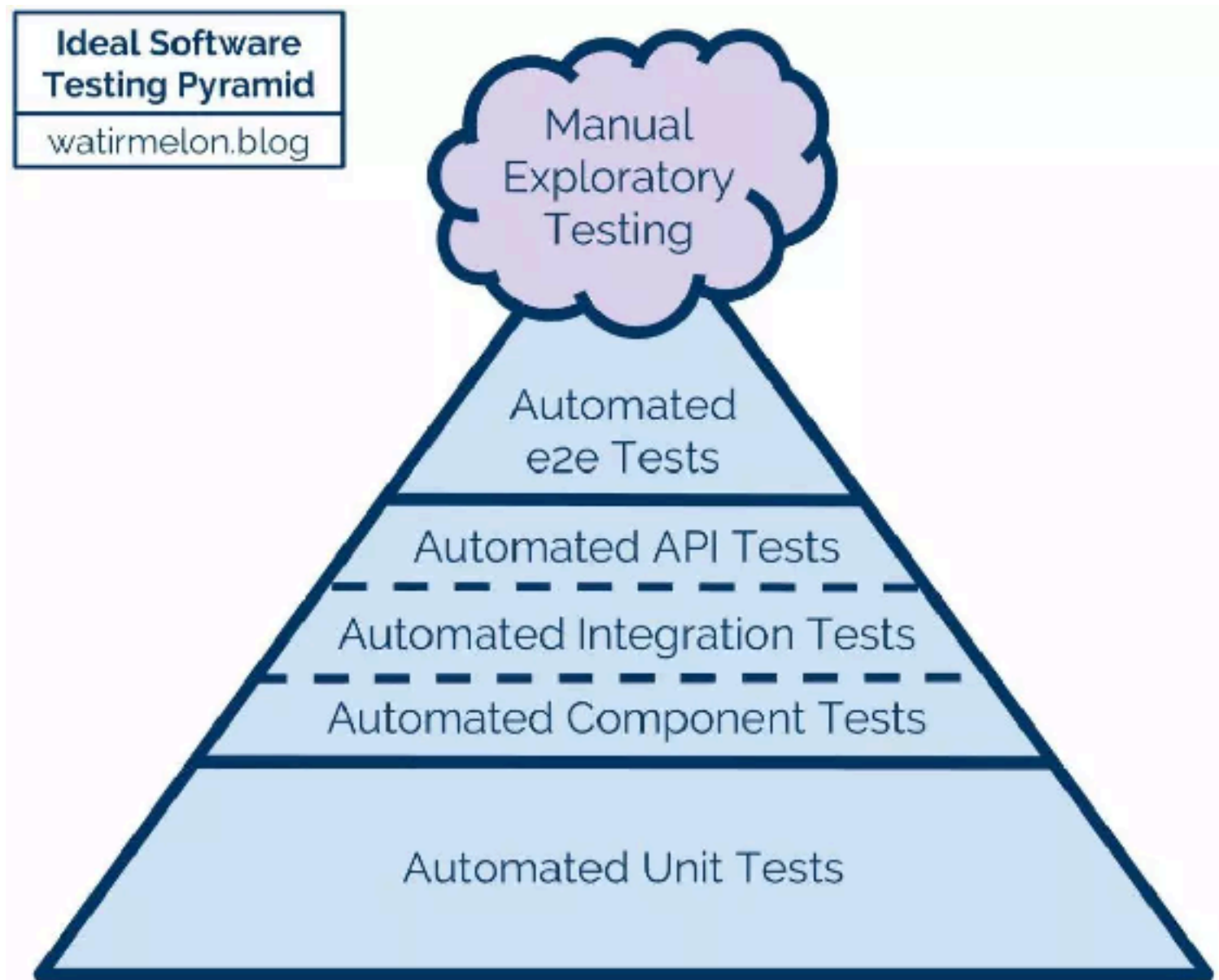
The left sidebar contains the page name **somkiat.cc**, the handle **@somkiat.cc**, and a menu with options: Home, Posts, Videos, and Photos.



Test Double



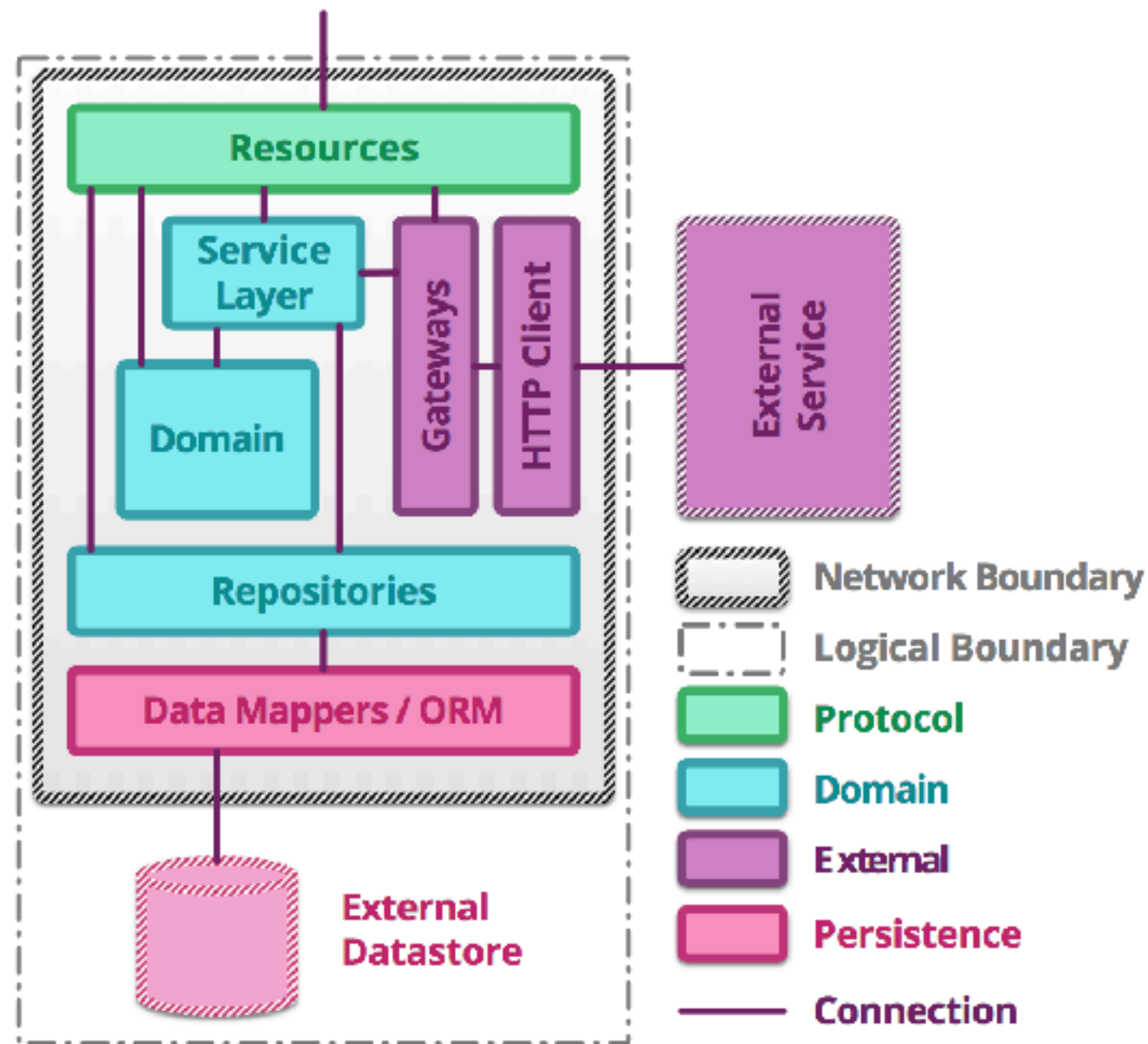
Testing Pyramid



<https://watirmelon.blog/testing-pyramids/>



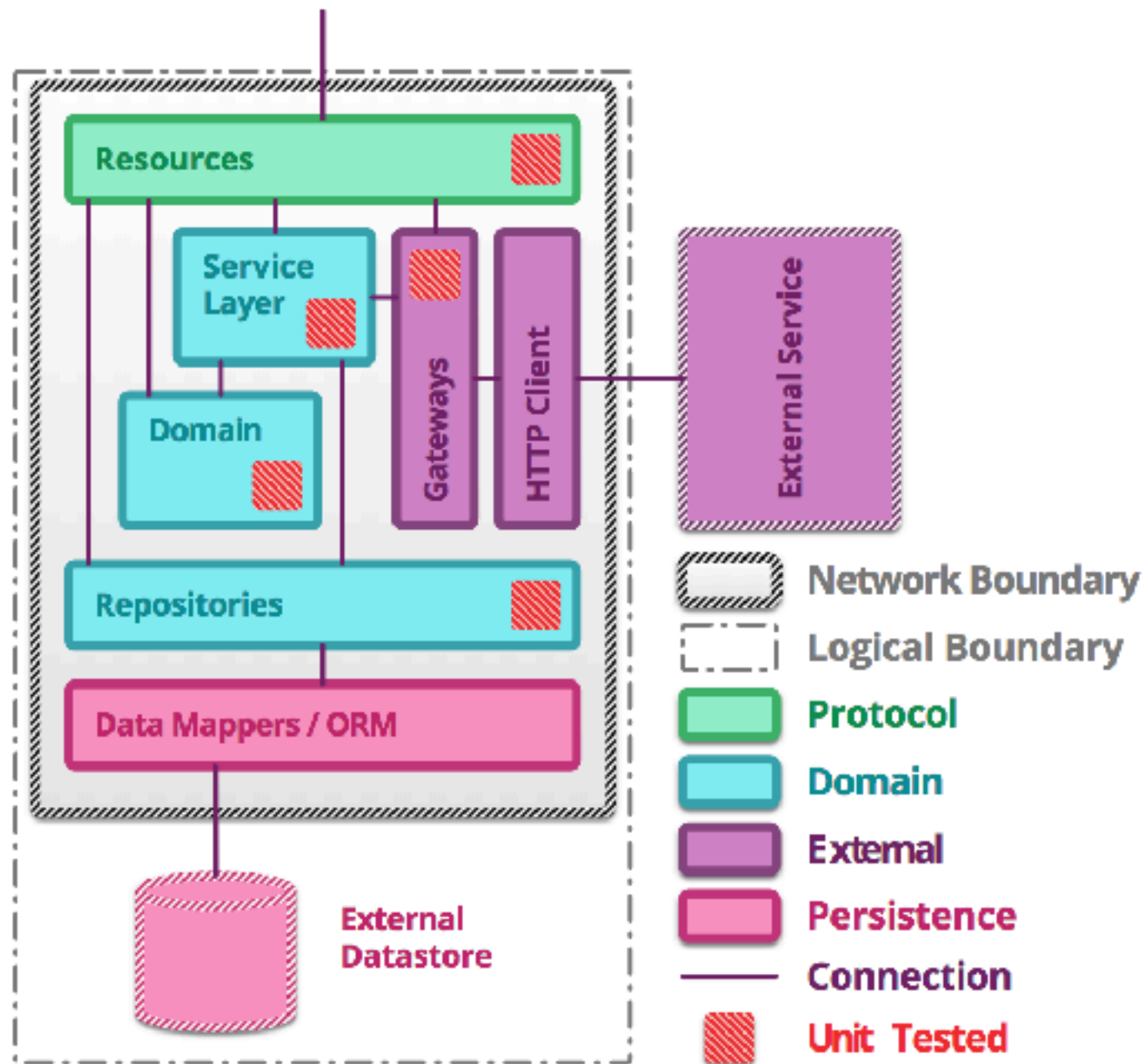
Project/Service Structure



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



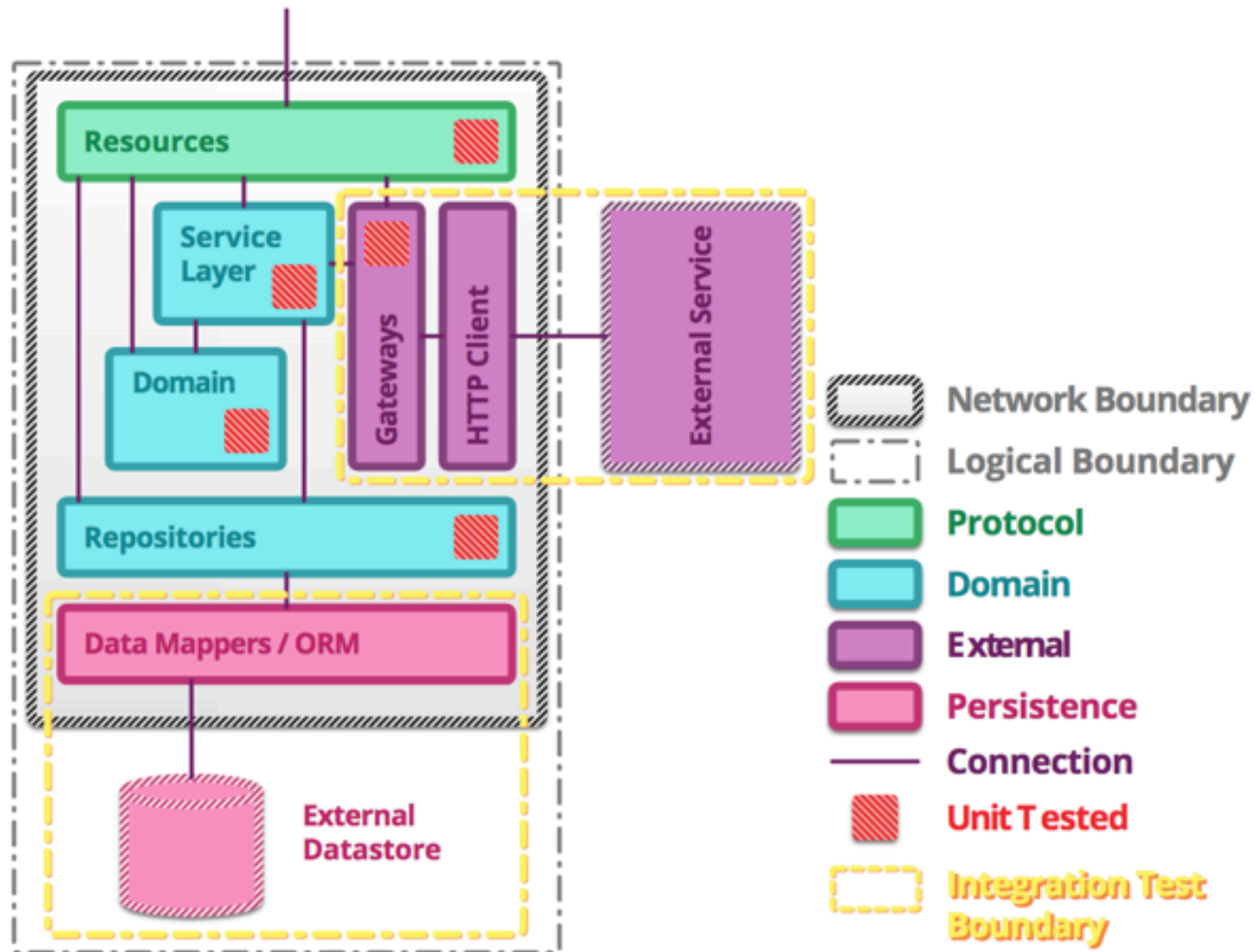
Unit Testing



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



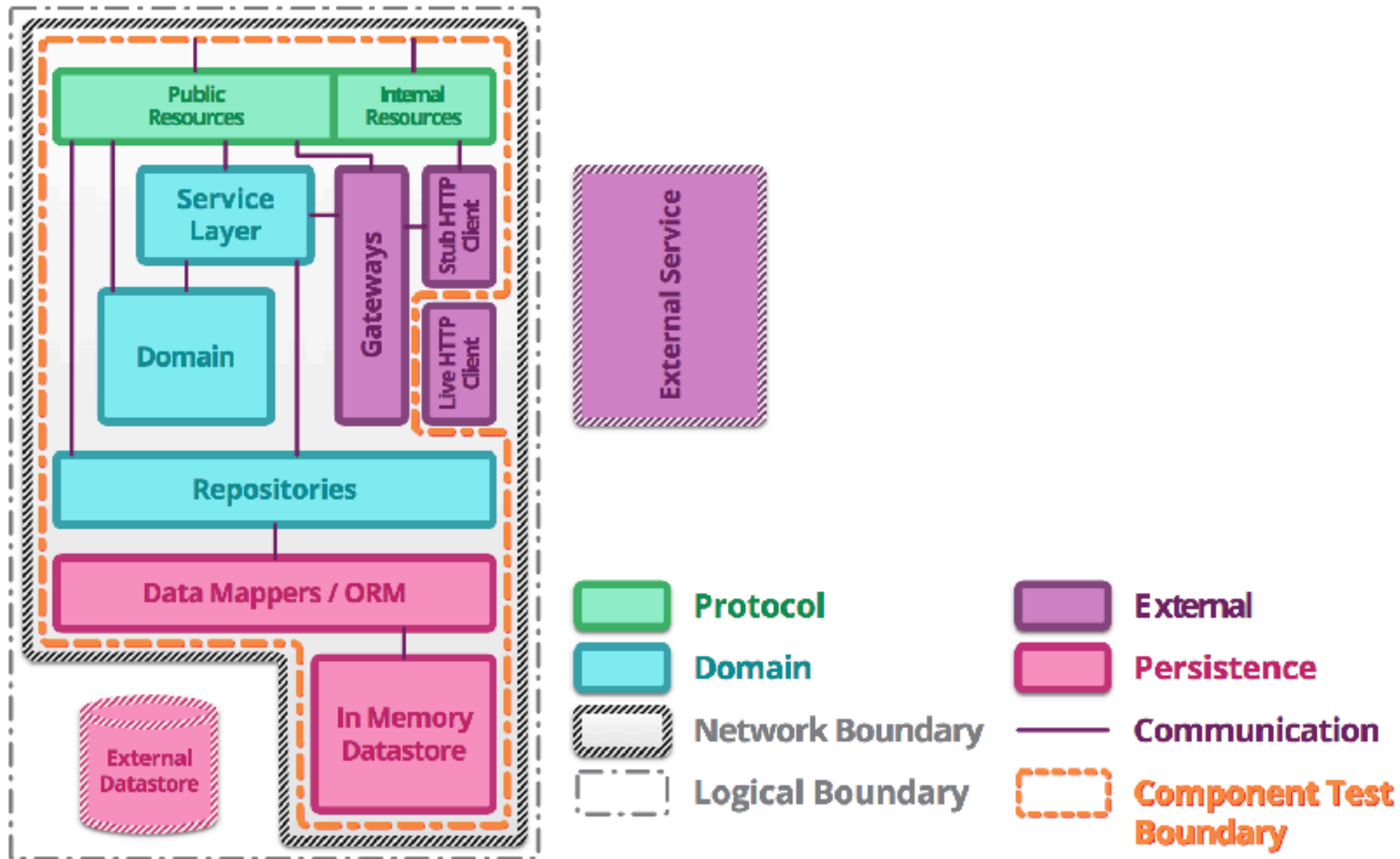
Integration Testing



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



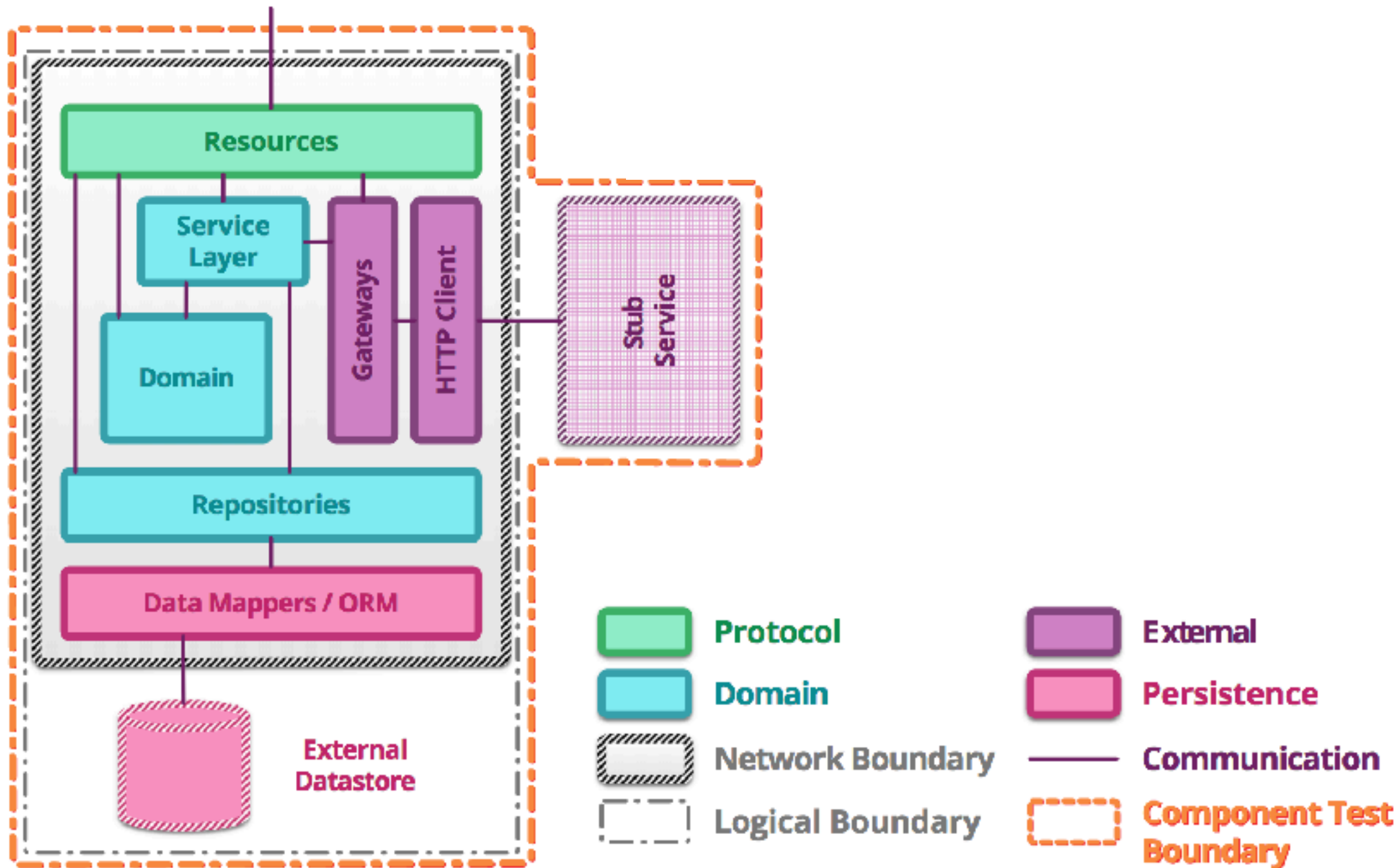
Component Testing (1)



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



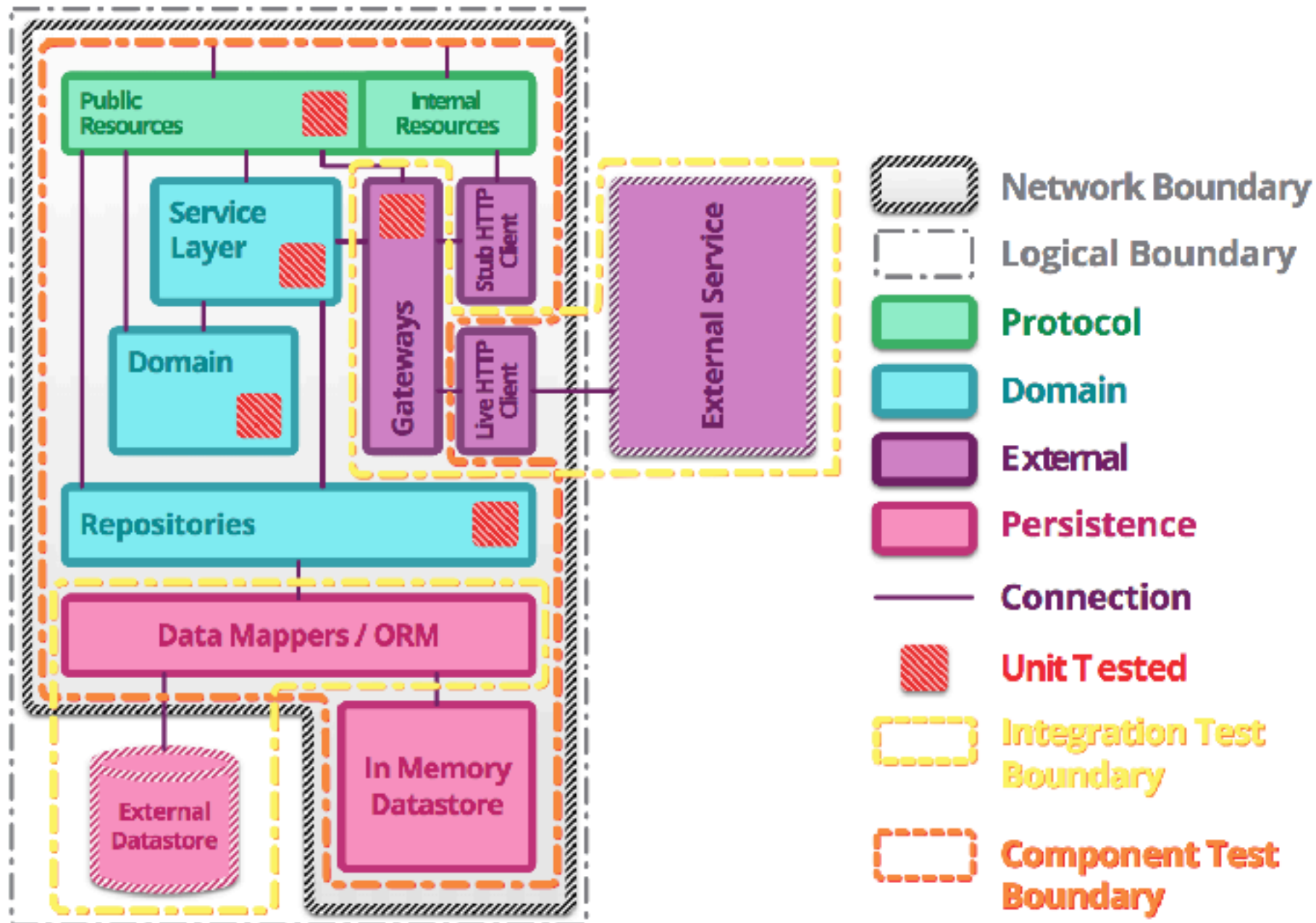
Component Testing (2)



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



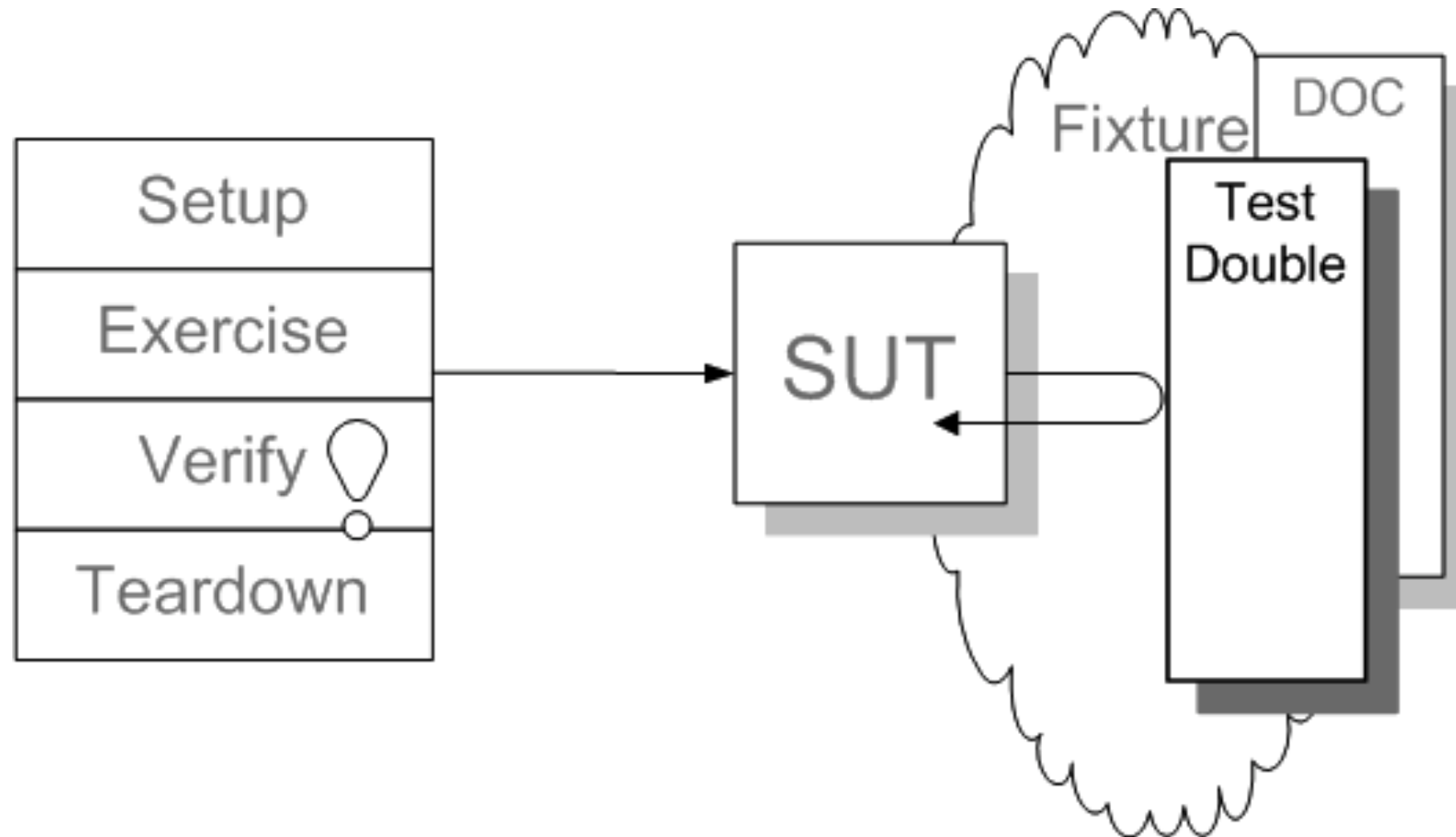
More confident



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



Test Double

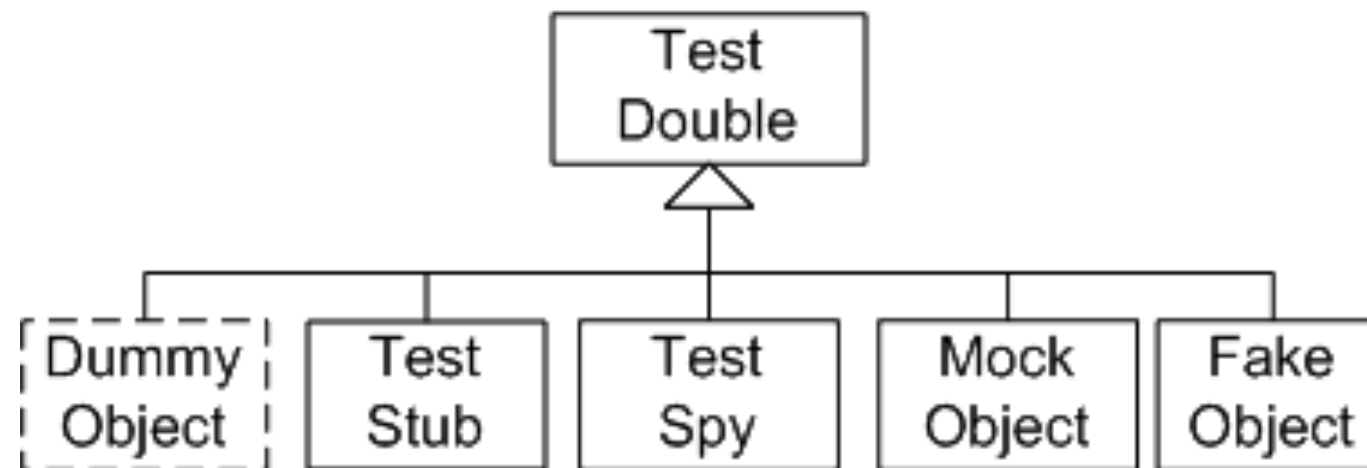


<http://xunitpatterns.com/Test%20Double.html>

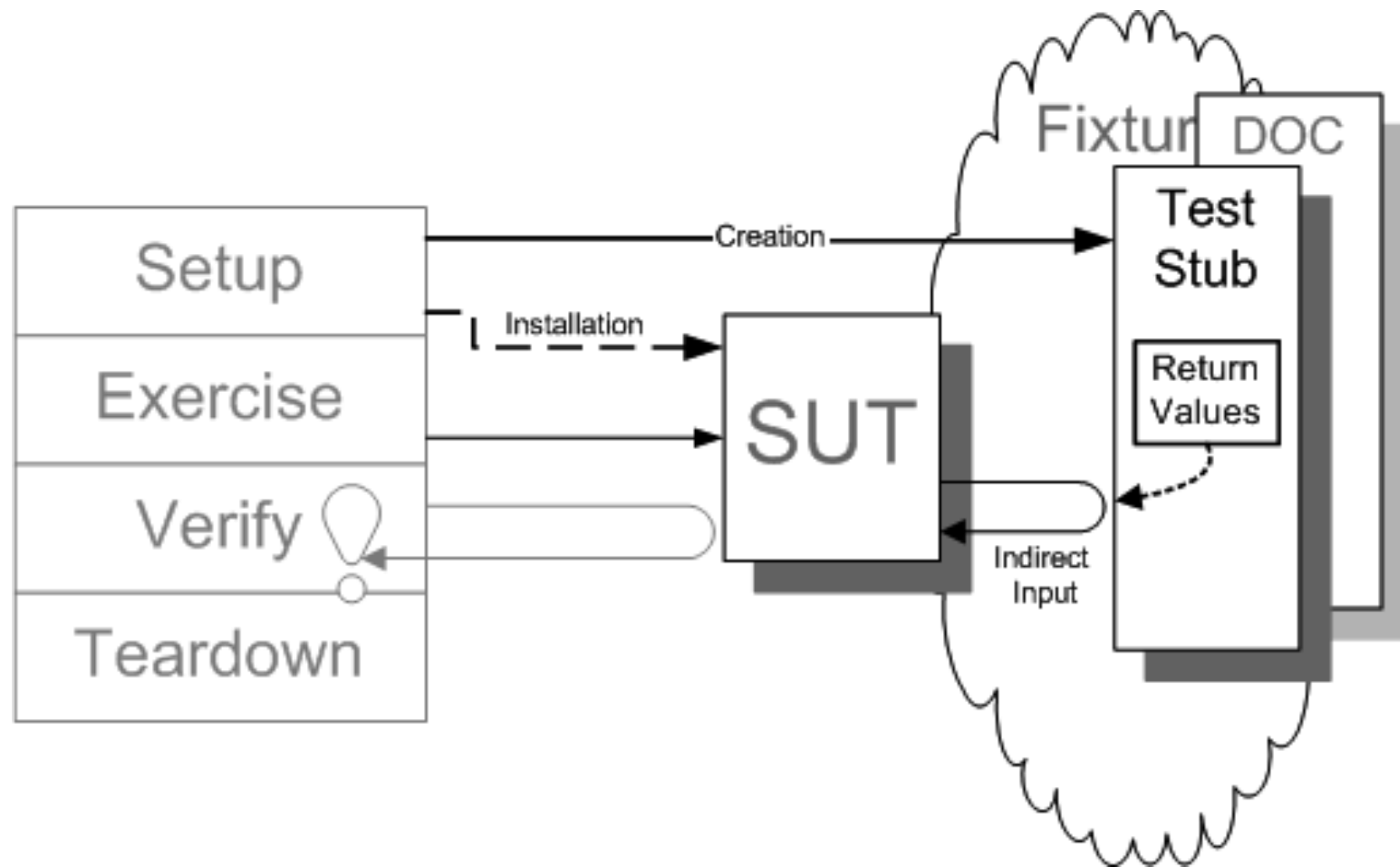


Test Double

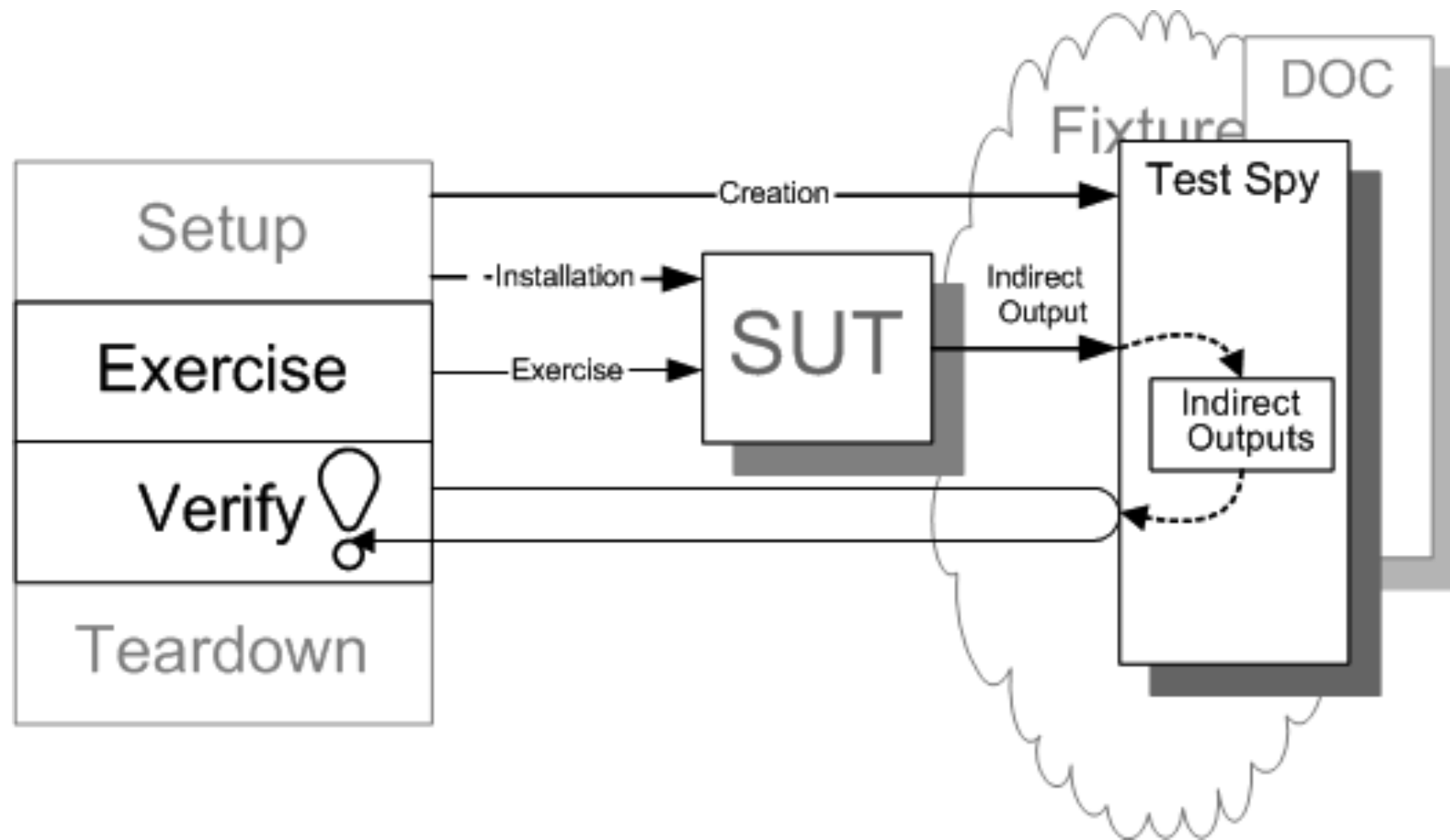
Dummy
Stub
Spy
Mock
Fake



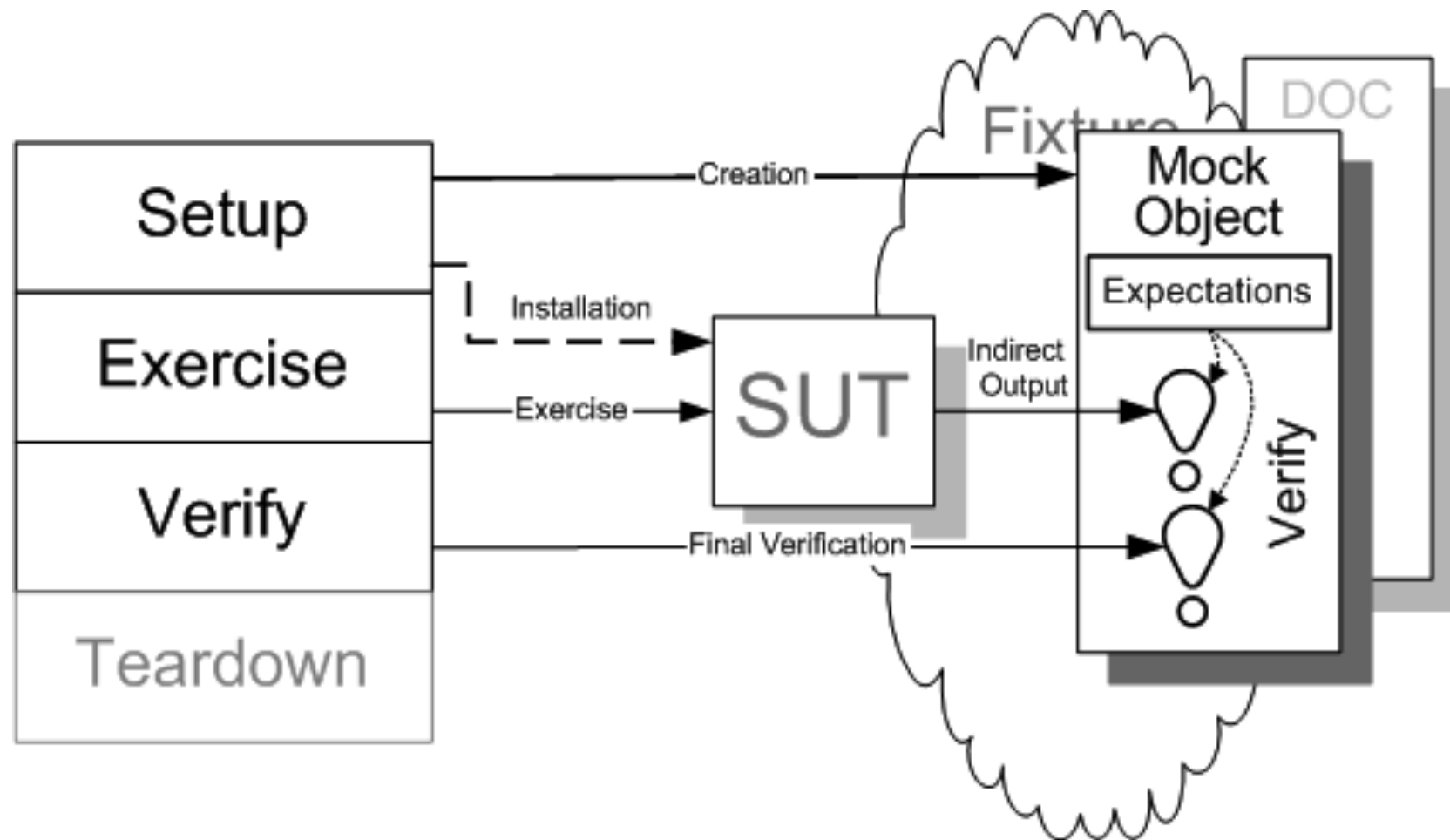
Stub



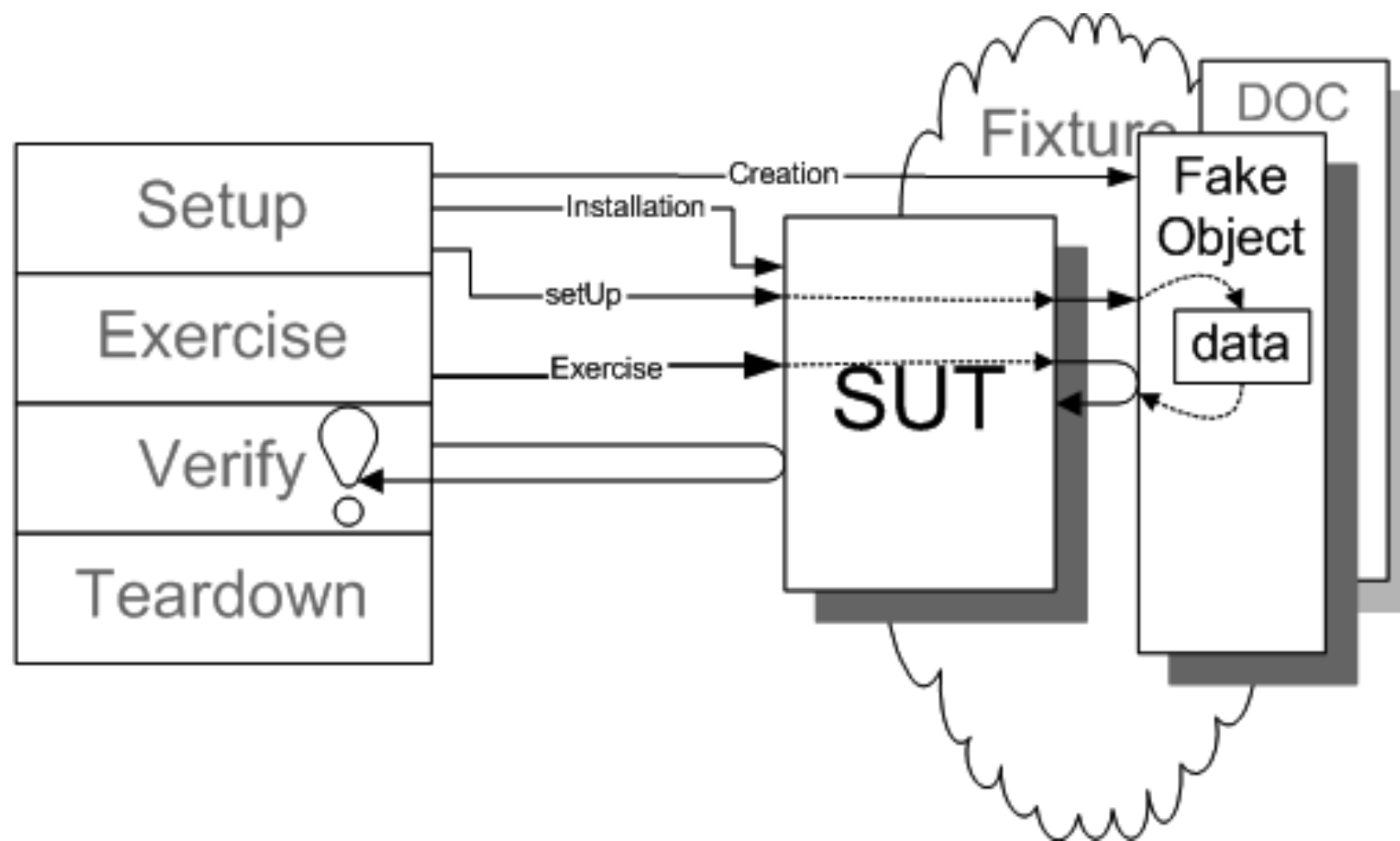
Spy



Mock



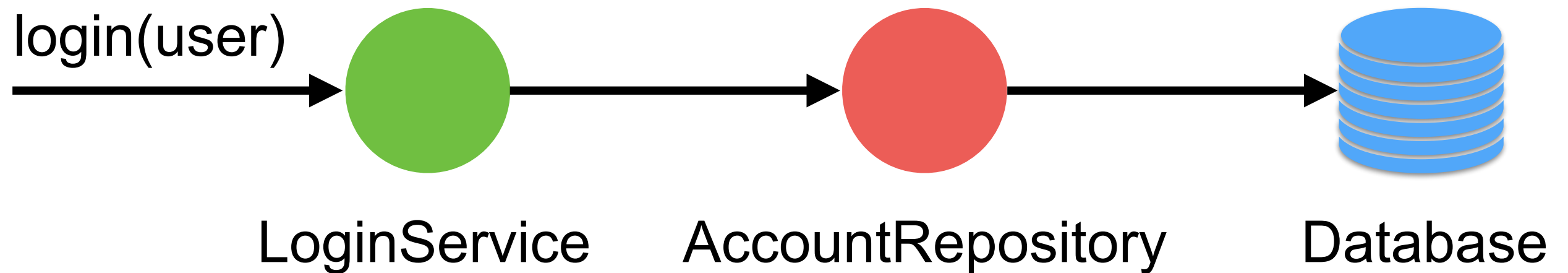
Fake



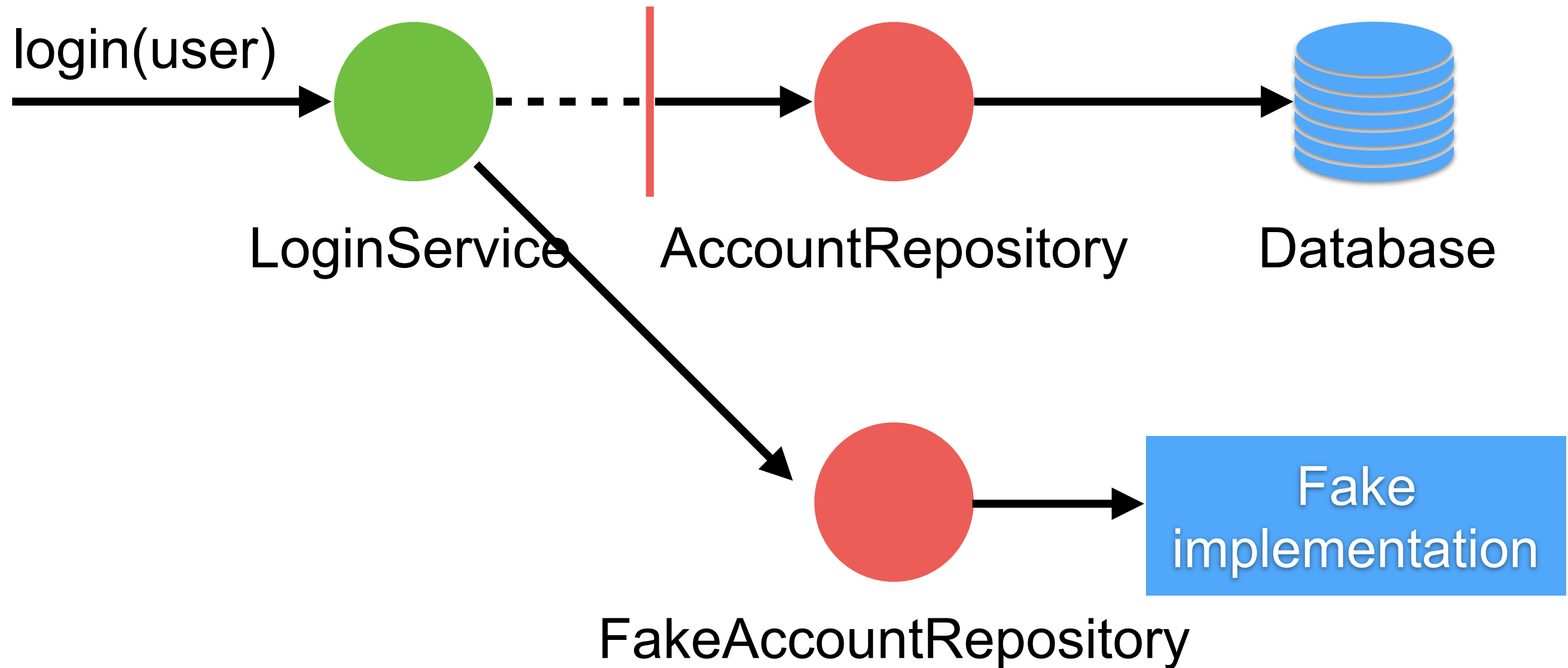
Test Double in Software Development



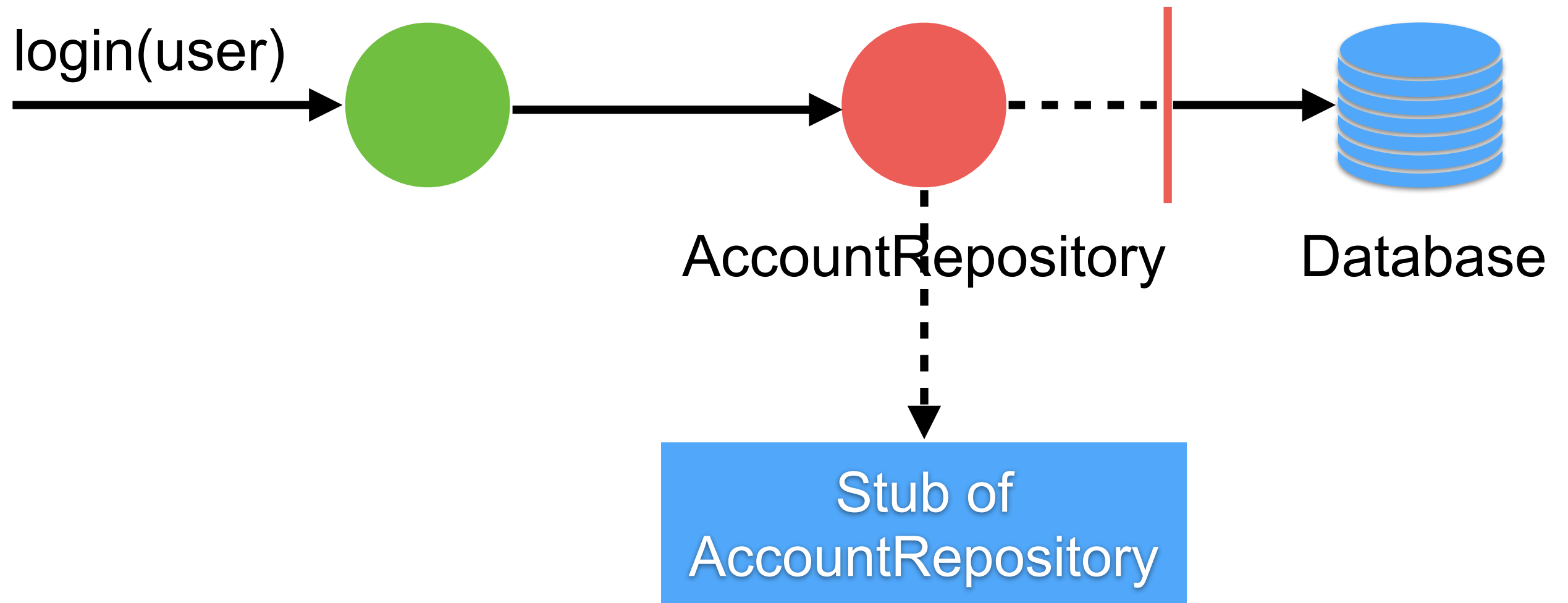
Simple flow



Fake



Stub



Working with Golang



Working with Golang

Working with Interface

Use 3-party libraries (go-sqlmock, testify, mockery)

Use container

Use mock/stub server for REST APIs



Develop

Focus on business logic

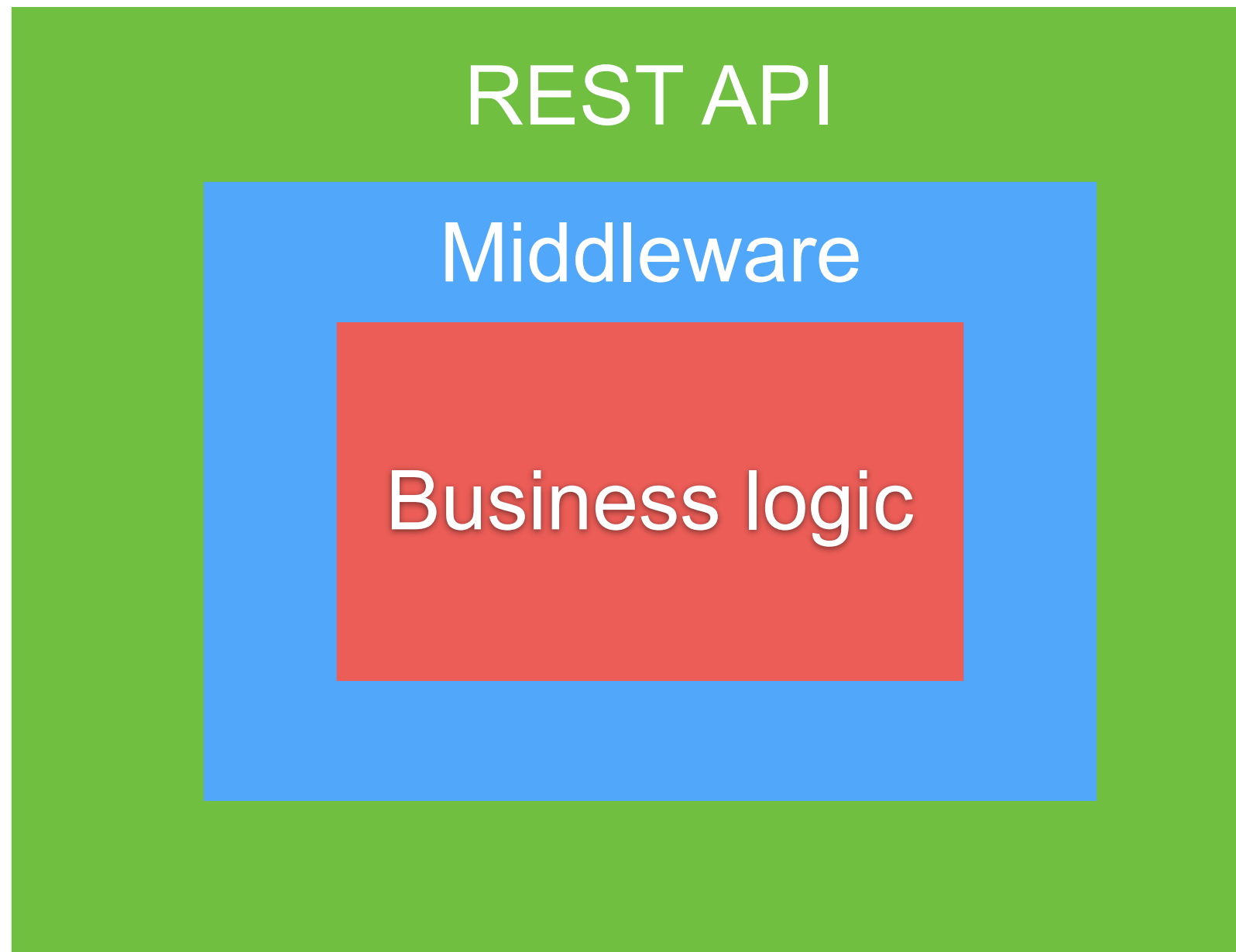
Create service via REST

Middleware (endpoint/routing)

Testing services



Develop



Develop :: Business logic

Define **interface** in each service

Create **struct**

Try to **testing**



Interface Semantics



Interface Semantics

```
package main  
import "fmt"
```

```
type printer interface {  
    print()  
}
```

```
type user struct {  
    name string  
}
```

```
func (u user) print() {  
    fmt.Println("User name: ", u.name)  
}
```



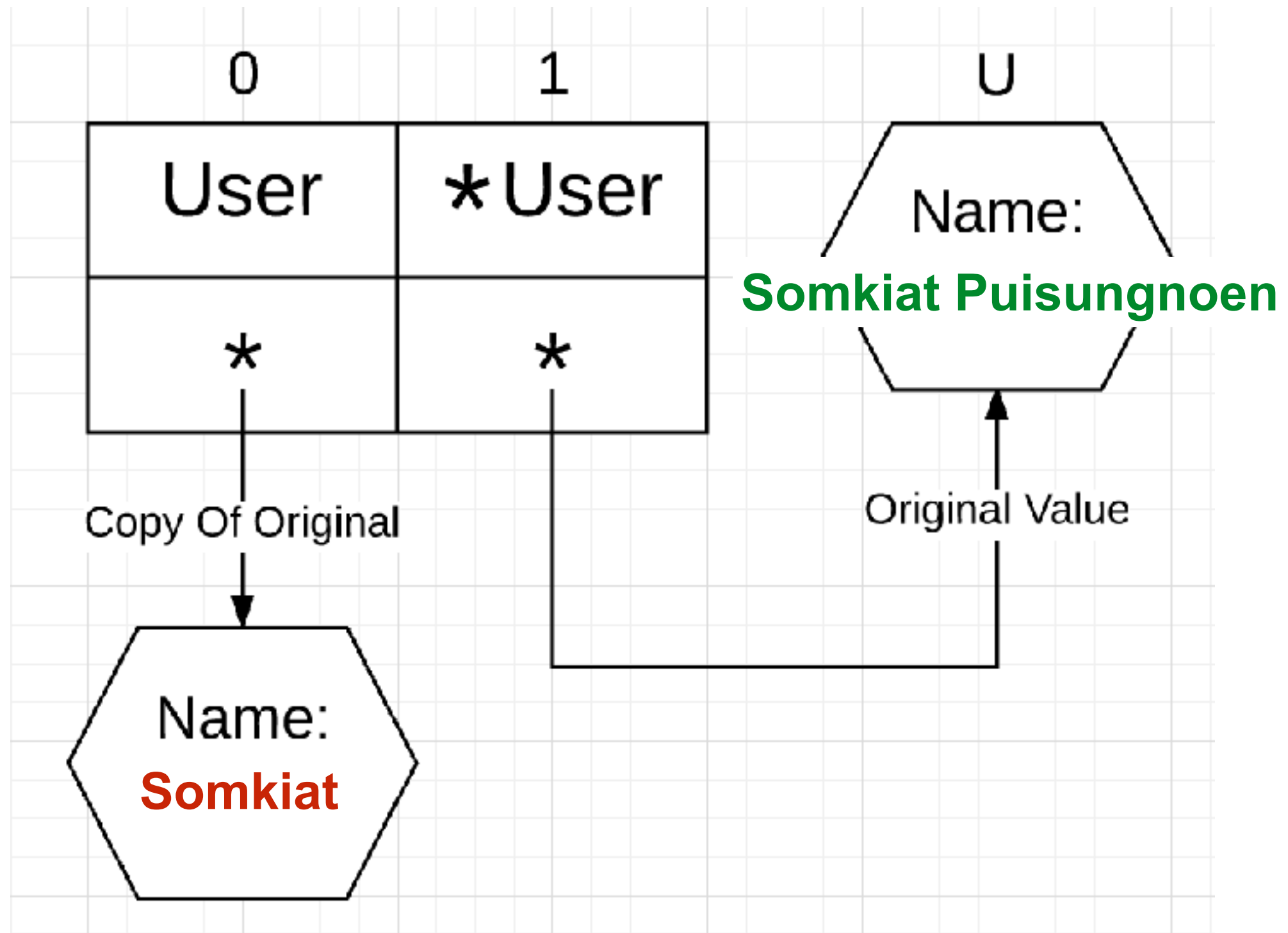
Interface Semantics

```
func main() {  
    u := user{"Somkiat"}  
    datas := []printer{  
        u,  
        &u,  
    }  
    u.name = "Somkiat Puisungnoen"  
  
    for _, data := range datas {  
        data.print()  
    }  
}
```

<https://play.golang.org/p/870xD9DBpvq>



Interface Semantics (Value/Pointer)



Interface Semantics

Value semantic (Copy from original)

Pointer semantic (Original value)



Suggestion

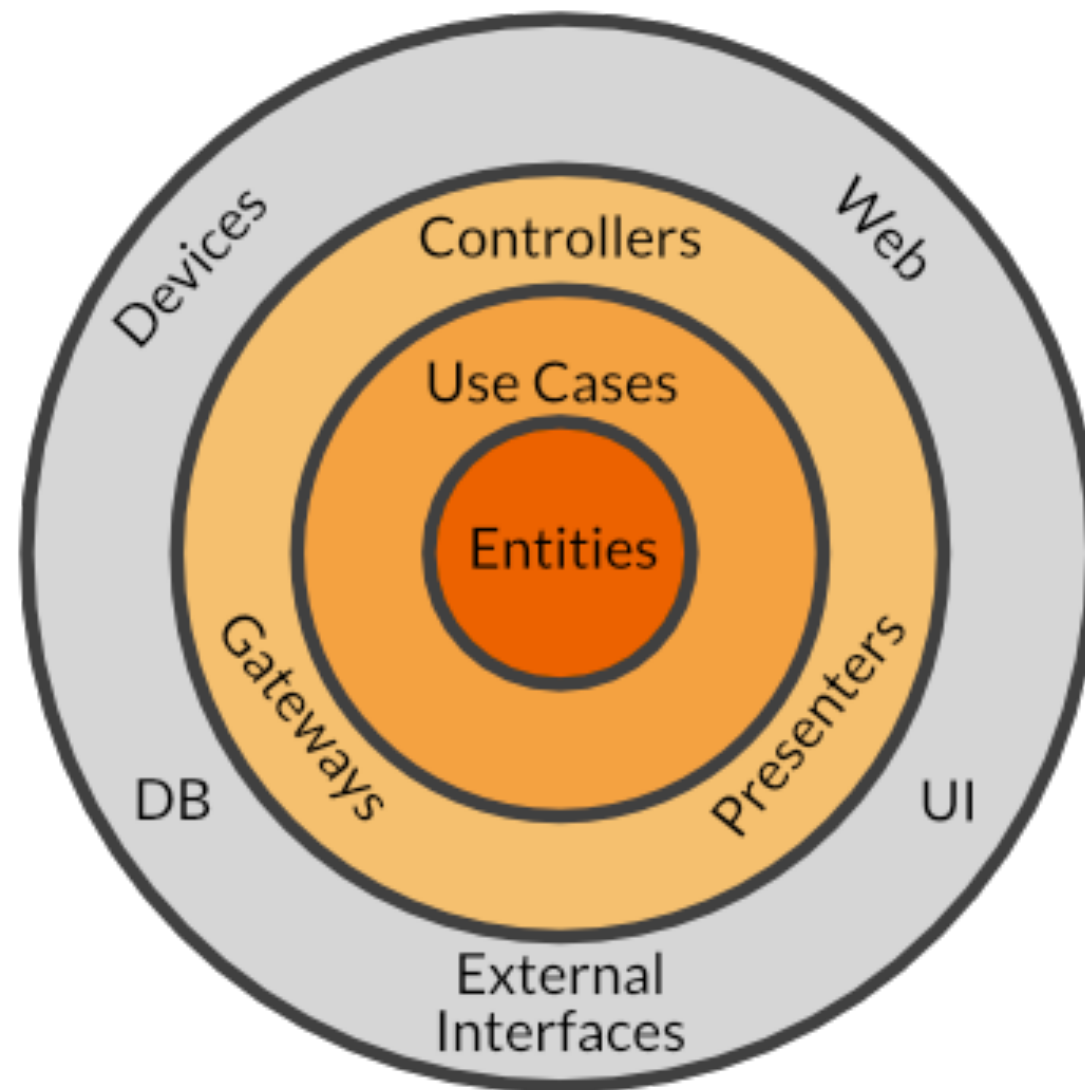
Choosing one semantic (consistency)

Don't mix



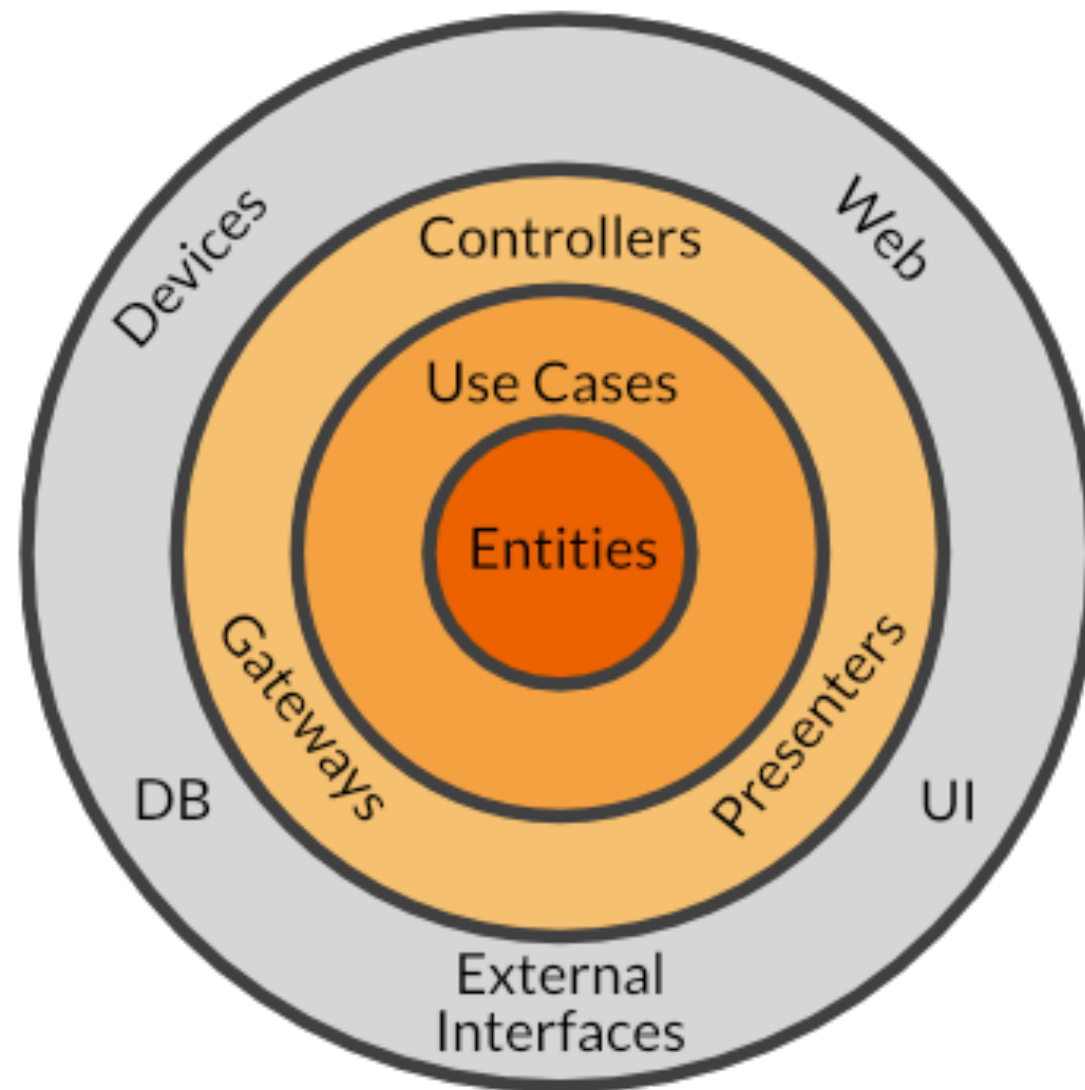
Clean Architecture

Separation of Concern (Abstraction layer)



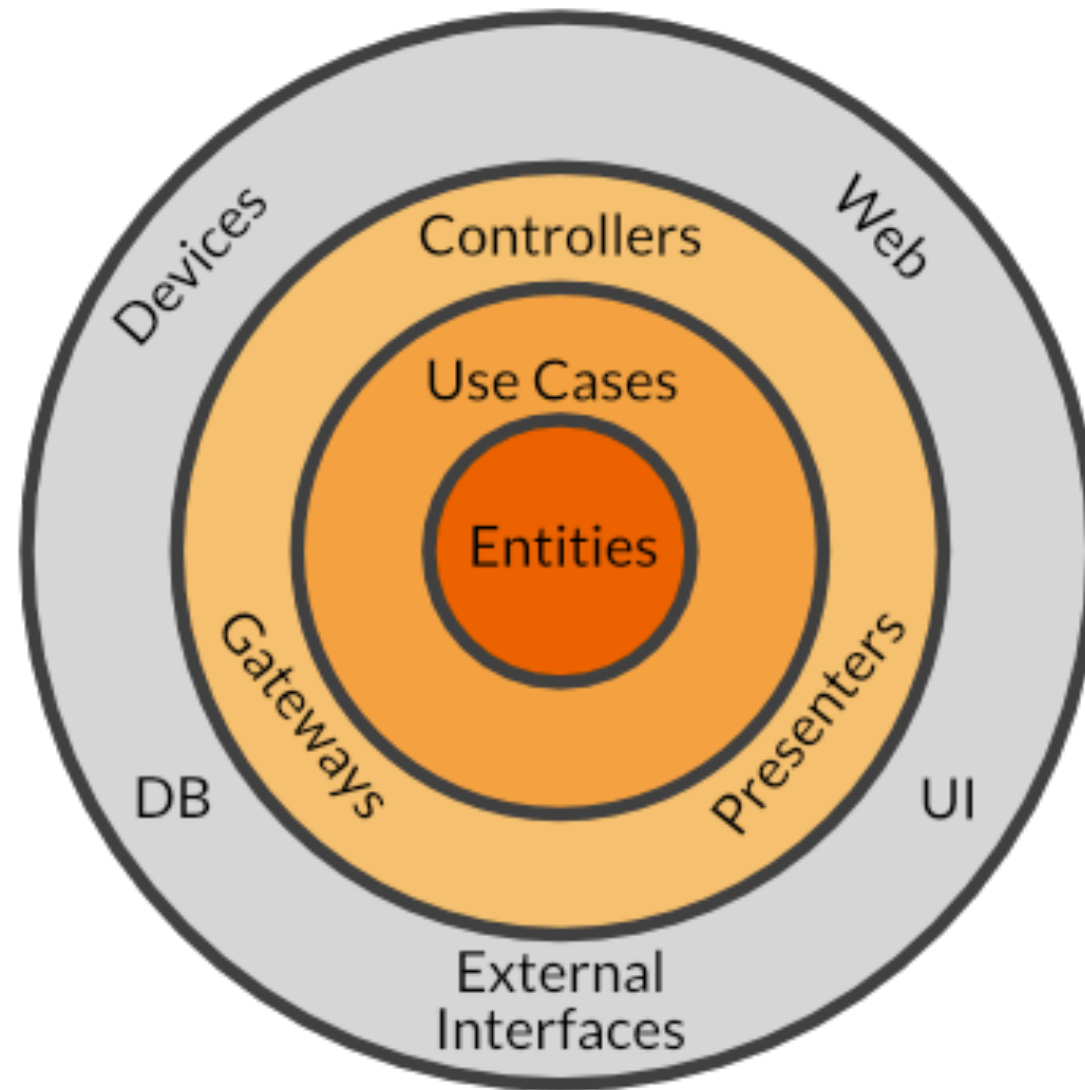
Clean Architecture

Code in each layer never access to other

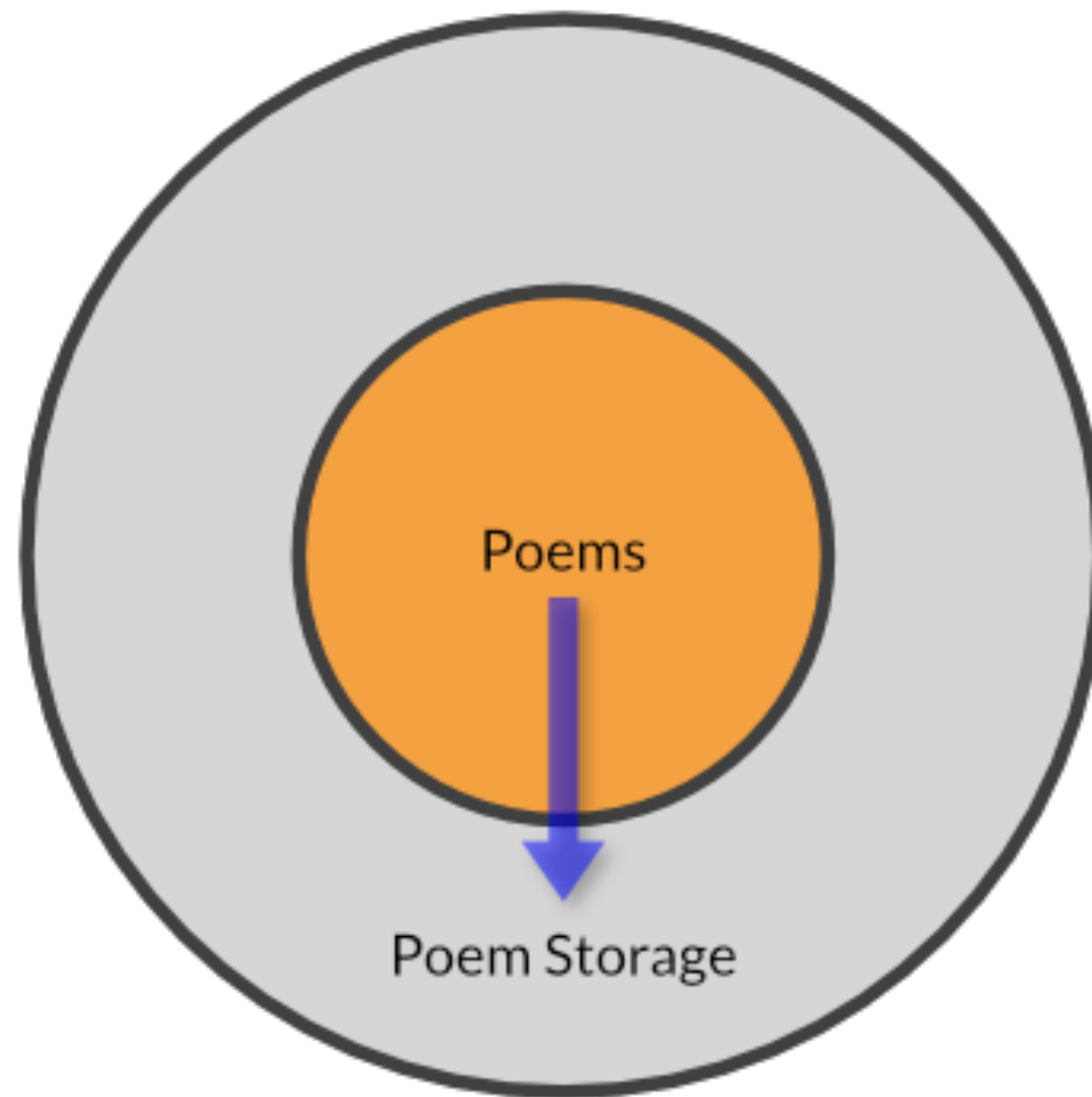


Clean Architecture

Dependency Injection(DI) Rule



Example



Bad example

```
type PoemStorage struct {  
}
```

```
type Poem struct {  
    content []byte  
    storage PoemStorage  
}
```



Better example

```
type PoemStorage interface {  
    Load(string) []byte  
    Save(string, []byte)  
}
```

```
type Poem struct {  
    content []byte  
    storage PoemStorage  
}
```



Demo/Workshop

