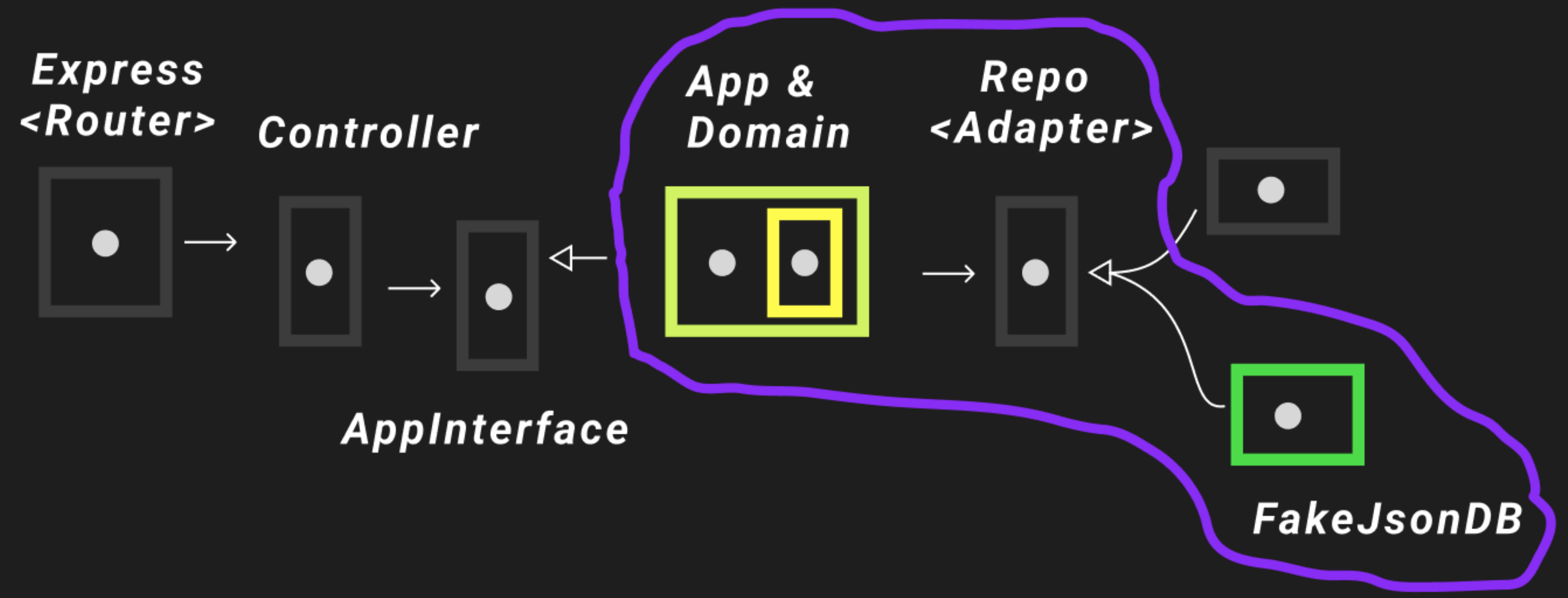
The SOFTWARE ESSENTIALIST

Physical Essential: Subject/ System Verification



Physical Essential

Subject / System Verification

Your application is comprised of many different input-output systems.

Subject/system verification gives us the ability to isolate the systems (subjects) we want to test, then using one of 3 techniques.

Result, State & Communication Verification.

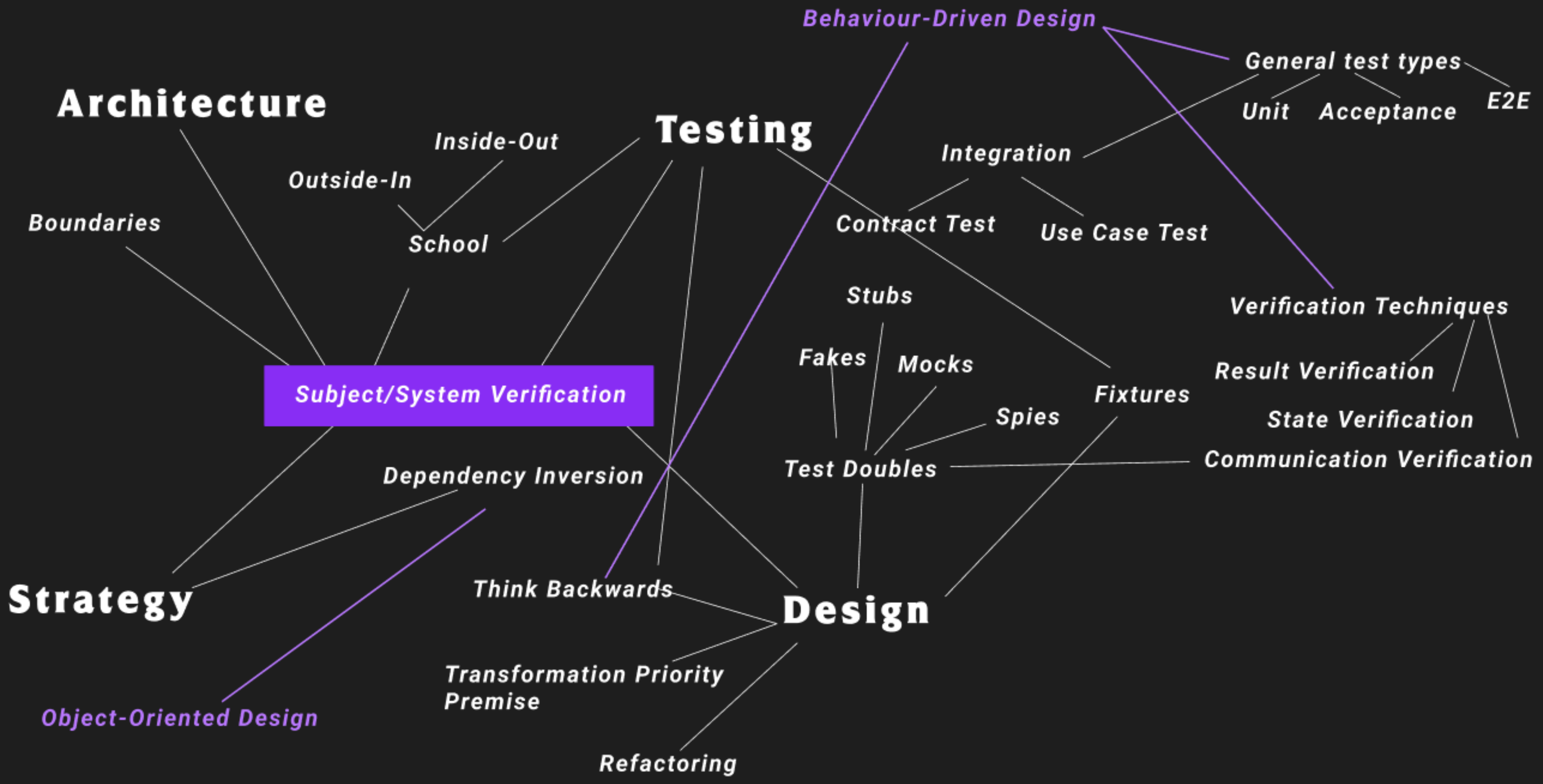
Result Verification verifies the input-output of a system.

State Verification verifies the internal state of a system.

Communication Verification verifies the communications that a system has with other systems.

Using these 3 forms of verification, we can perform all sorts of testing strategies using some or none of stubs, mocks, spies and the like.

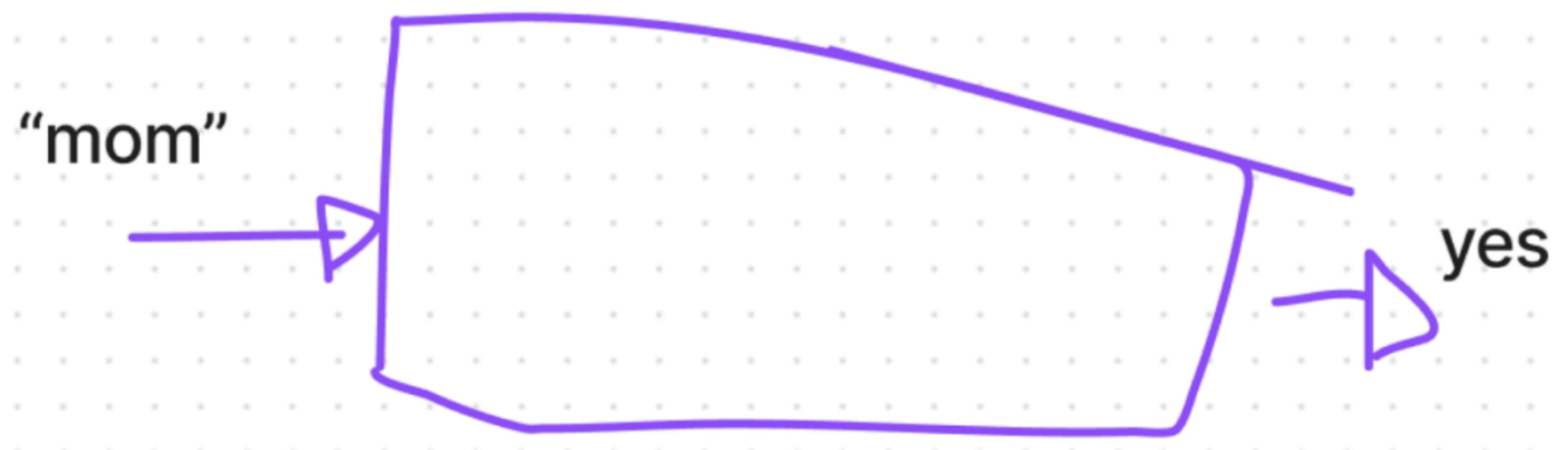
essentialist.dev



essentialist.dev

Result Verification

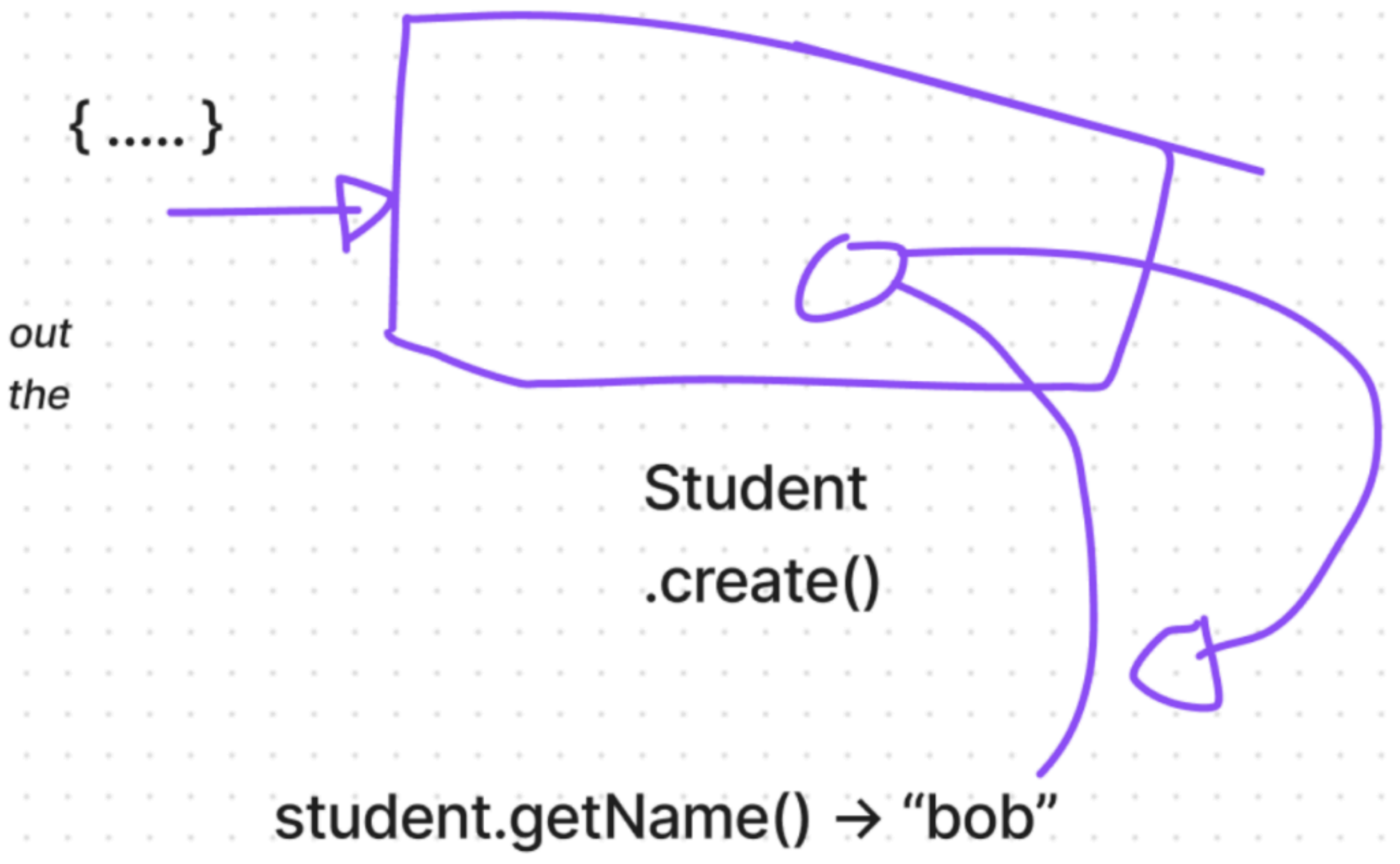
Input-Output system is black box.
Data goes in. Observe what comes out.



```
describe('palindrome', () => {
  it("should know that 'mom' is a palindrome", () => {
    expect(isAPalindrome("mom")).toBeTruthy();
 })
})
```

State Verification

Input-Output system is white-box.
*Data goes in. Doesn't necessarily come out of the other end. Need to **inquire** about the internal state.*

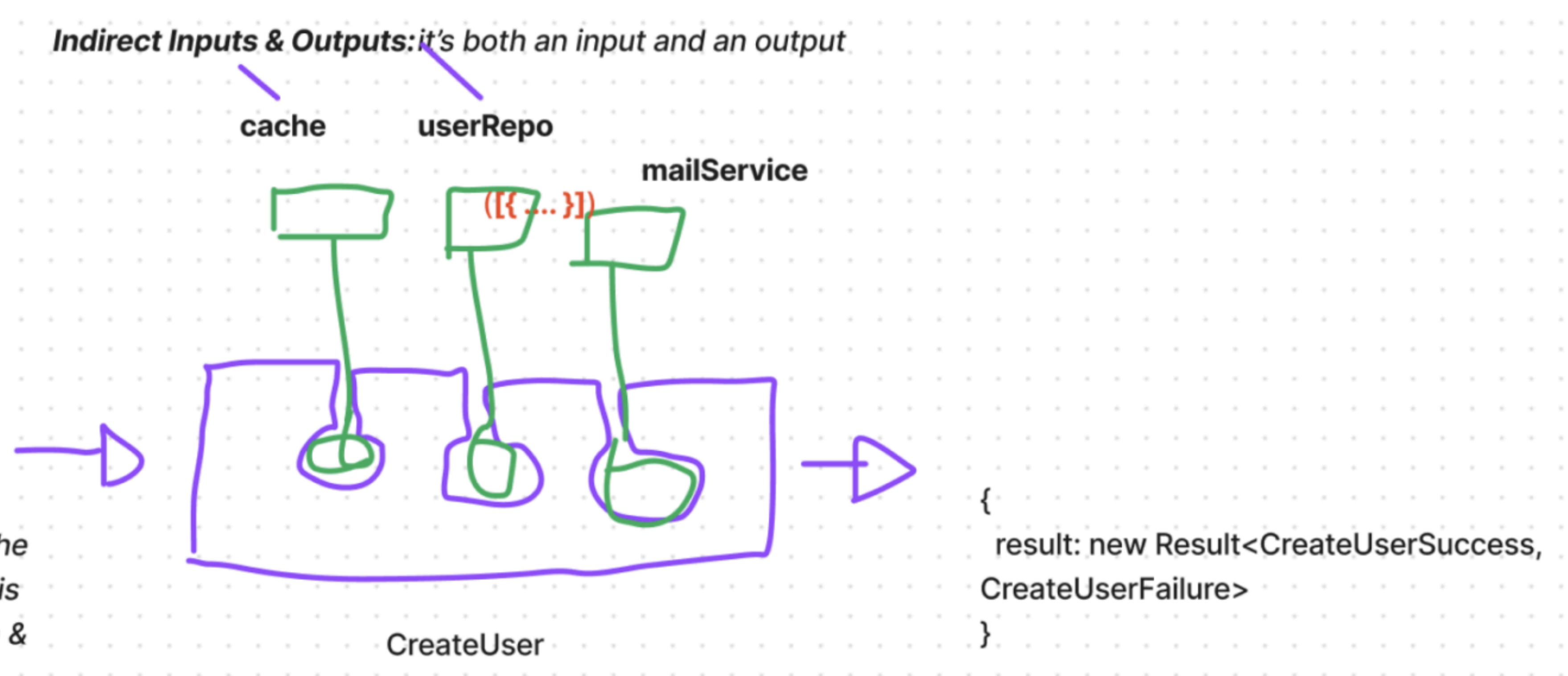


```
describe('student', () => {
  it("strips spaces and capitalizes name", () => {
    let student = Student.create('kh a l i l');
    expect(student.getName()).toBe("Khalil");
  })
})
```

essentialist.dev

Communication Verification

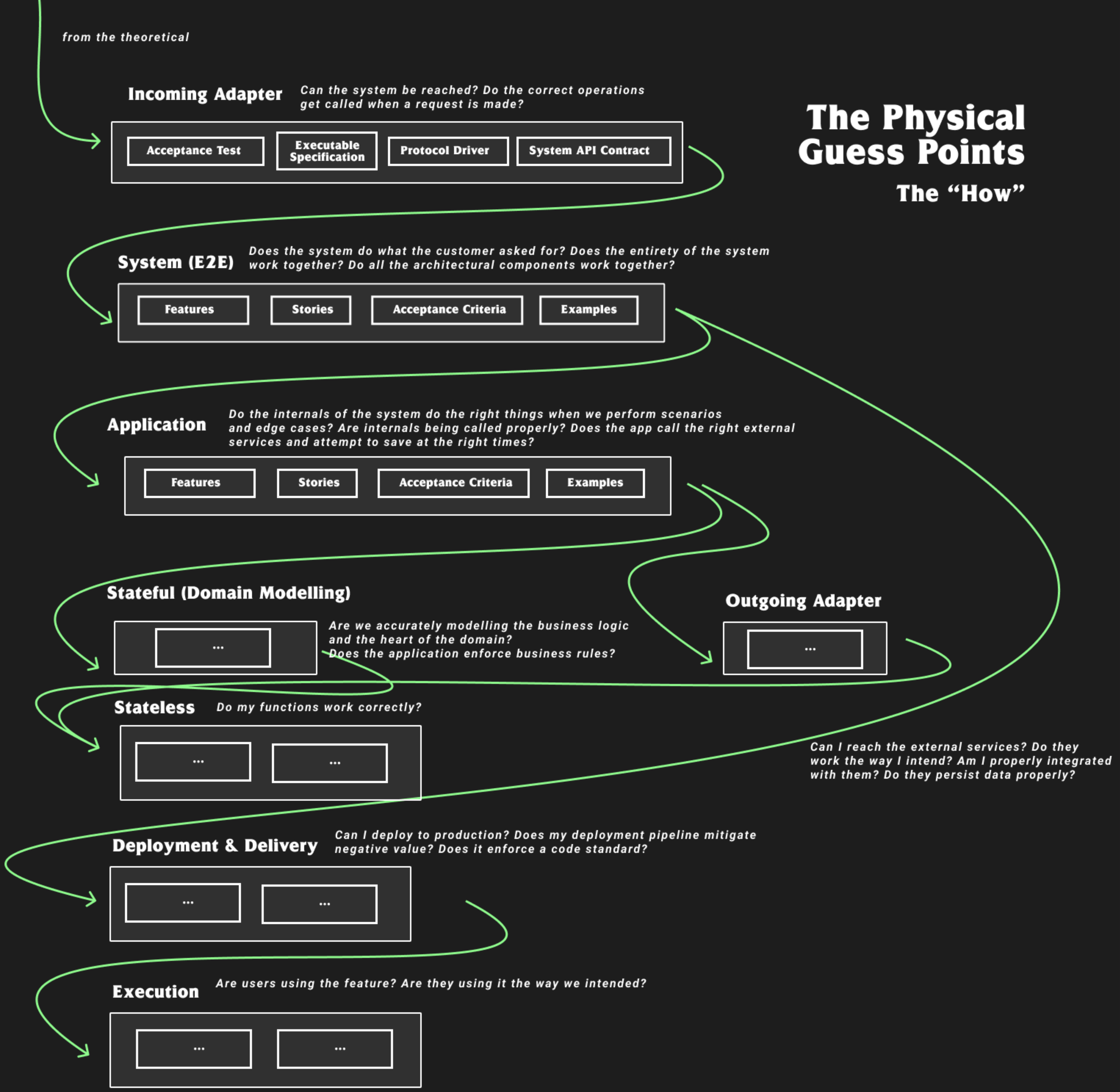
*Input-Output system is pre-loaded with **indirect inputs & outputs**. We look at the indirect outputs to determine if the system is working properly; perform state verification & communication verification.*


Indirect Inputs & Outputs: *it's both an input and an output*
cache
userRepo
mailService
([{ ... }])
CreateUser
{
result: new Result<CreateUserSuccess, CreateUserFailure>
}

```
describe('createUser', () => {
  it("successfully creating a user", async () => {

    let userRepoSpy = /* Spy setup */
    let mailServiceSpy = /* Spy setup */
    /* Stub the cache to miss */

    let user = UserBuilder
      .withEmail('khalil@essentialist.dev')
      .build()

    let createUserResult = await userModule.createUser(user);

    // ...

    // Communication verification
    expect(userRepoSpy.save.toHaveBeenCalledTimes(1));
    expect(mailServiceSpy.getArgsFromMethod('sendWelcomeEmail')
      .getArg('destinationEmail')
    ).toBe('khalil@essentialist.dev')
  })
})
```

The Physical Guess Points

The "How"



from the theoretical

Incoming Adapter *Can the system be reached? Do the correct operations get called when a request is made?*

System (E2E) *Does the system do what the customer asked for? Does the entirety of the system work together? Do all the architectural components work together?*

Application *Do the internals of the system do the right things when we perform scenarios and edge cases? Are internals being called properly? Does the app call the right external services and attempt to save at the right times?*

Stateful (Domain Modelling) *Are we accurately modelling the business logic and the heart of the domain? Does the application enforce business rules?*

Outgoing Adapter *Can I reach the external services? Do they work the way I intend? Am I properly integrated with them? Do they persist data properly?*

Stateless *Do my functions work correctly?*

Deployment & Delivery *Can I deploy to production? Does my deployment pipeline mitigate negative value? Does it enforce a code standard?*

Execution *Are users using the feature? Are they using it the way we intended?*

essentialist.dev

Where we'll learn more