Agile meets Architecture

How Testability Supports Your Agility

– and why this matters for your architecture!

Thomas Much
@thmuch

6 September 2022, Berlin

TL'DR

- Agility in software development:
  Continuously deliver value to our customers.
  Easily & quickly **adapt to change**.
- Requires **safety**, for example by fast, continuous test automation.
- Difficult if code and **architecture** aren't designed for **testability**.
- Separation of integration code and domain code
  is one of the **fundamental ideas** for testability.
- Fosters collaboration & thinking
  about testing & testability in **different roles**.

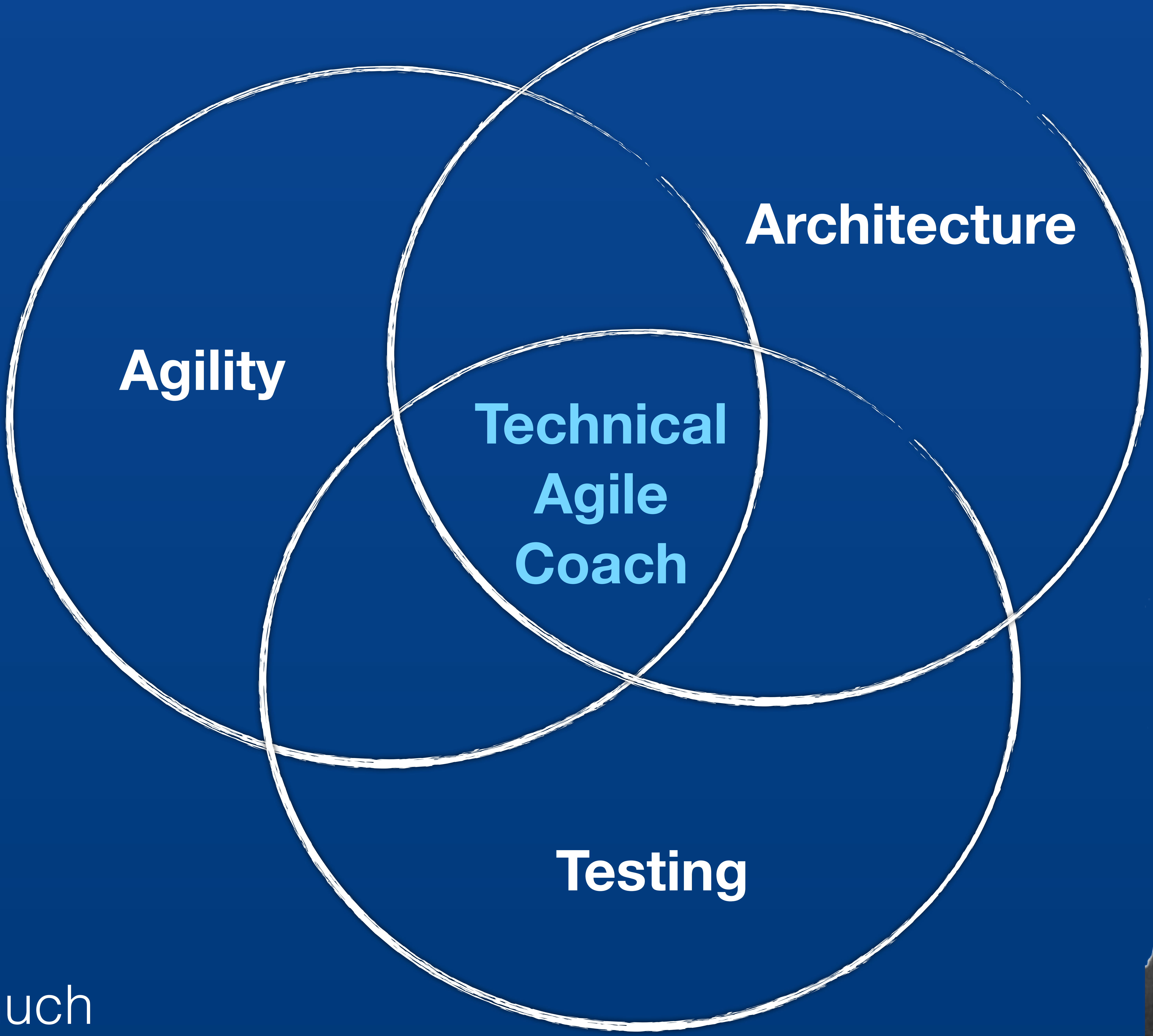
Please Note

Ideas presented today are **not new**.

Necessary to talk about **basics** from time to time.

These basics are still **essential**.

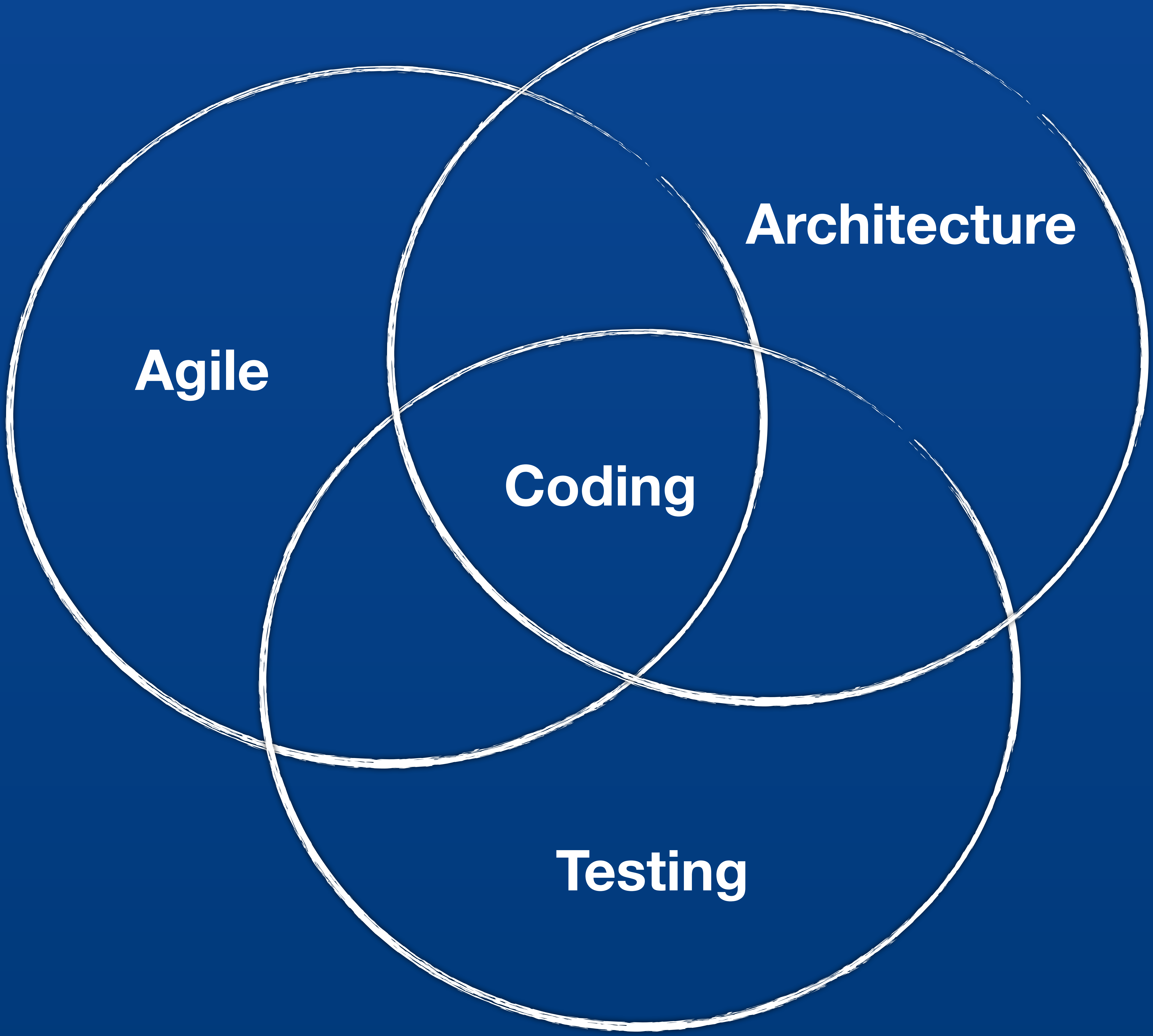
Maybe more essential than 20 years ago,
because more & more companies want to be agile
and deliver software (value) continuously.



Die Techniker

www.tk.de/IT

@thmuch



Short survey:

Who mostly has to do with …

Coding & Testing

Agile

Architecture

Typical situations:

Our team isn’t working "Agile" enough.

Customers not satisfied, results come in too slowly.

"We don't see any progress"

Typical Symptoms

No tests
(just maybe some random clicking here and there)

Mostly manual tests
(slow!)

Slow automated tests
(usually e2e / ui)

Fragile, unstable, flakey tests
(dependent on environment / external systems)

My Findings

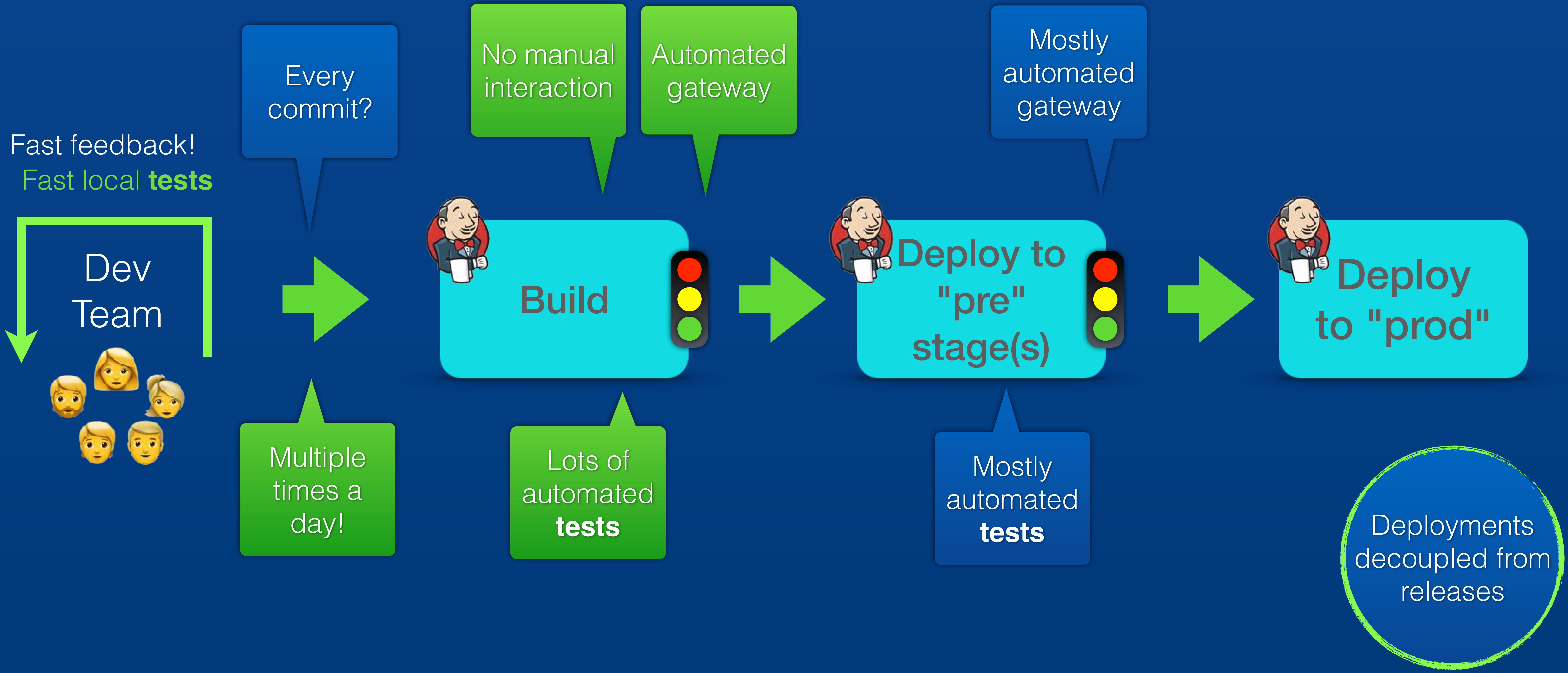
Symptom?
Cause?

Teams with **fast test automation**
are **more agile***
than those without

*) deliver more often, release in smaller batches, have a common understanding of their software, can support each other better – and they can write tests more easily

Not necessarily all "Agile" teams I've met …

What Makes Them More Agile?



How Dare They?

Safety net of a fast, predictable, comprehensive test suite

They **deliver value continuously**
(and if something Bad™ happens, it was only
a small change, easy to analyze, easy to revert)

Happy devs
(and probably happy **customers**)

They dare to make changes,
dare to try what helps the customers

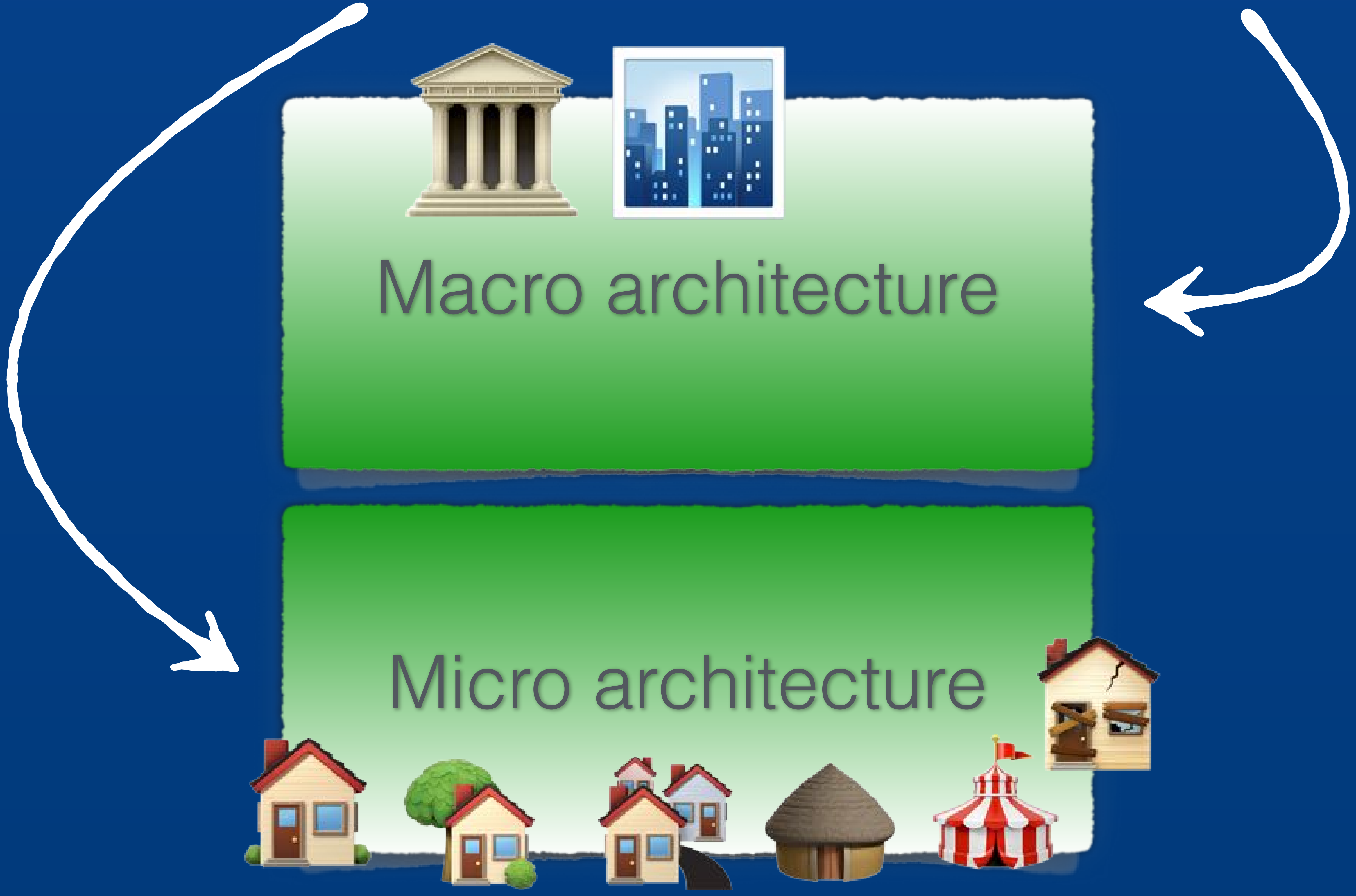
That's Pretty Agile! 😃

Modern Agile, actually.

Modern Software Engineering.

Result of Evolution

(Most) often **started small**, then scaled.
With lots of **freedom** and some **guidelines**.



What is Architecture, Anyway?

"Everything that is hard to change"

All decisions that need **lots of** rework/redesign/refactoring later on.

How to decide?

Testability is a **property of your architecture**

Some general principles to keep in mind, pretty old ones.

Aspects of Structure And Design

High modularity

Strong Cohesion

Loose coupling

Information Hiding

Factors Affecting Test Speed

Separation of systems

Size of systems

Dependen-cies between systems

High modularity

Strong Cohesion

Loose coupling

Information Hiding

Dependen-cies within one system

Focus here today

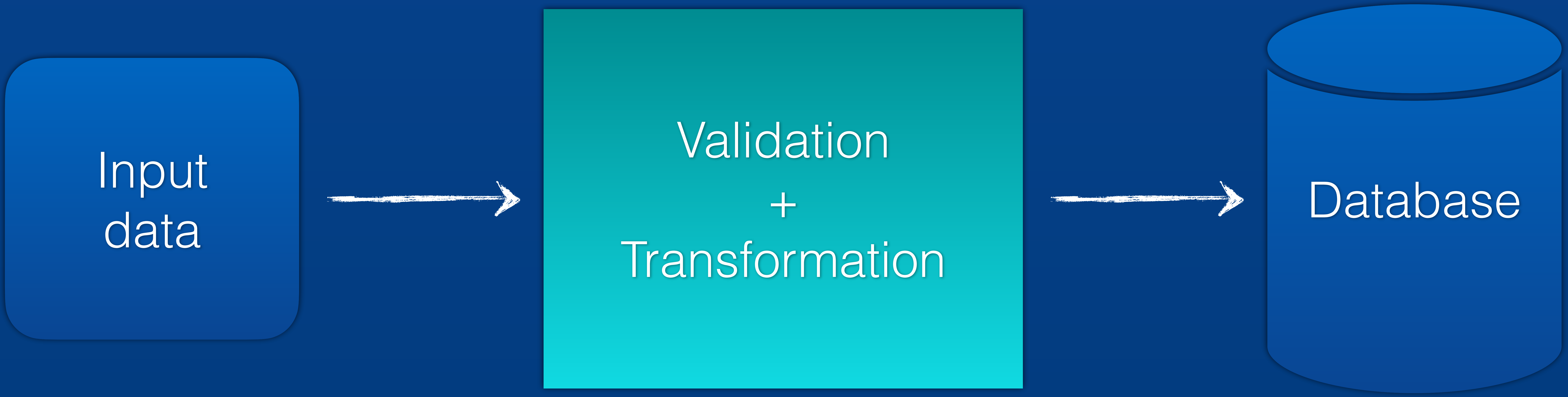
- Essential for a solid foundation of **fast tests**.
- Helpful for testing on all levels.

Separation of **Integration** Code and **Domain** Code

- **Local decision**.
- Often **possible in legacy systems**, too.

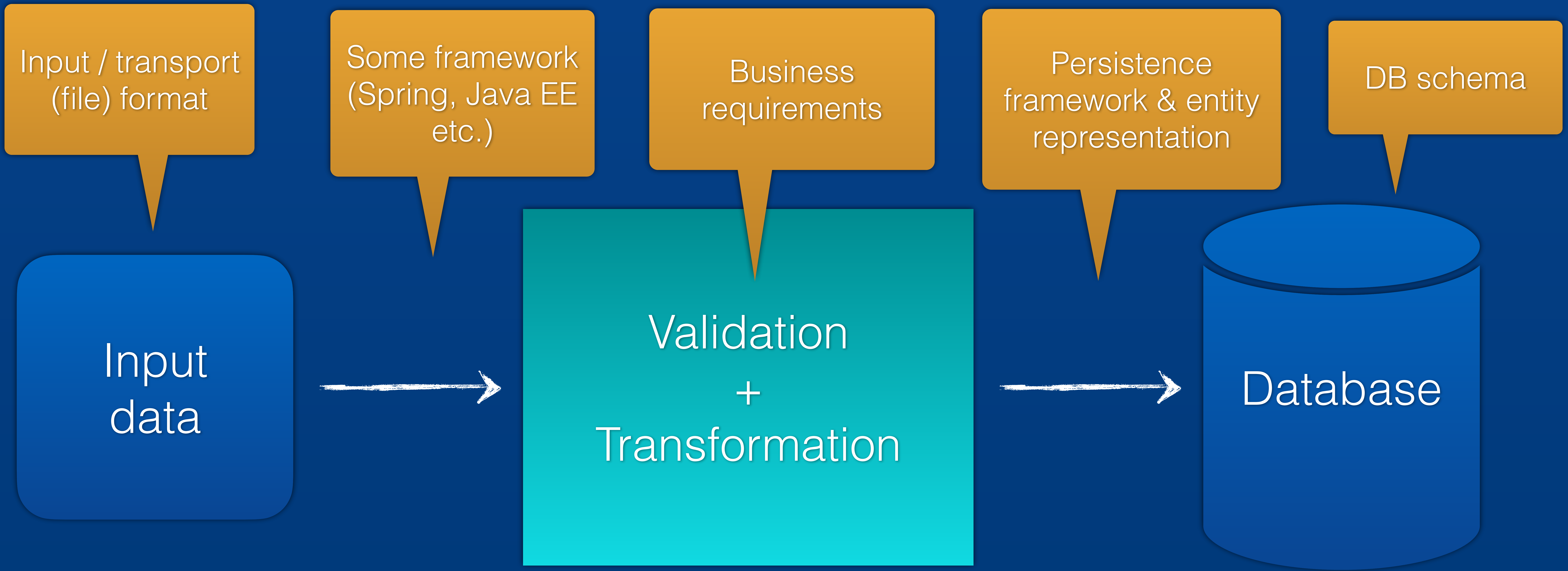
I Need an Example

A Common Real-World Example



Lots of Dependencies

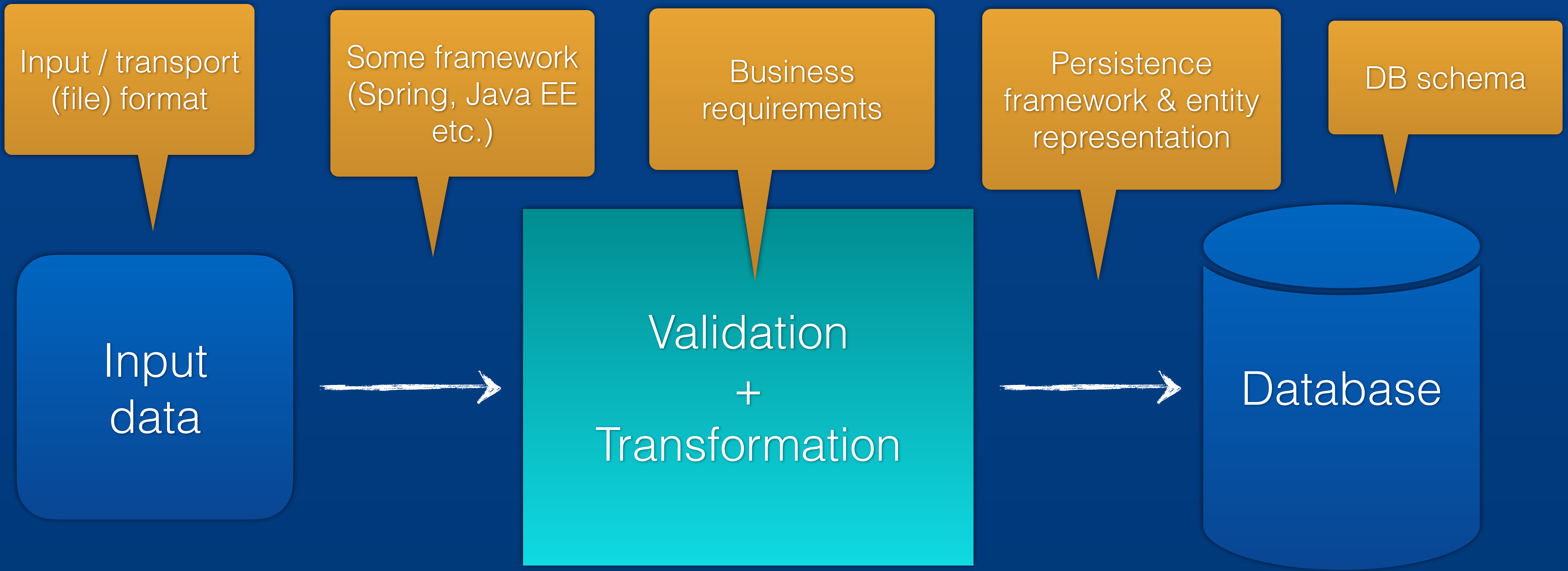
Dependencies are often **beyond our control**


Input / transport (file) format
Some framework (Spring, Java EE etc.)
Business requirements
Persistence framework & entity representation
DB schema
Input data
Validation + Transformation
Database

Dependencies … Reasons for Changes

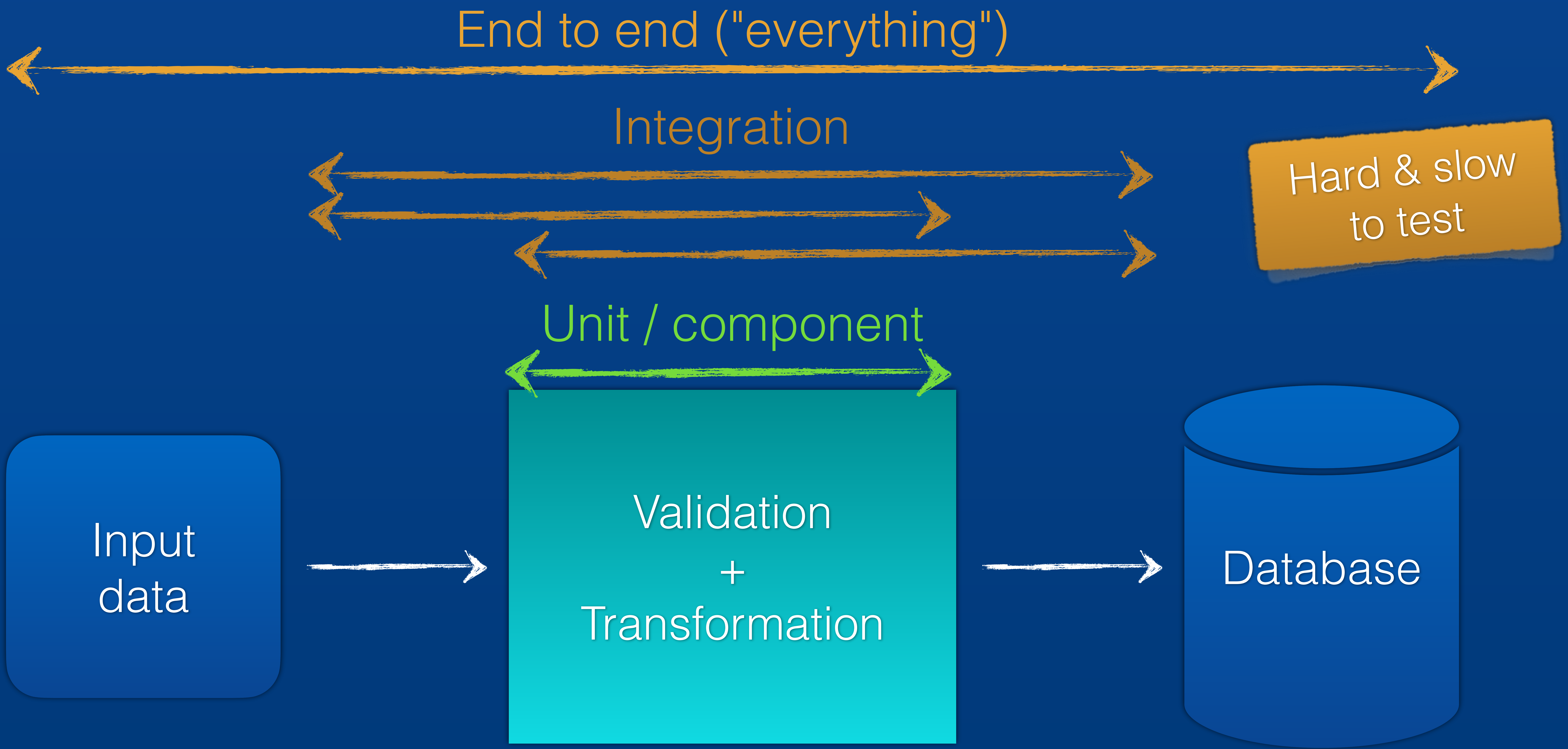
… changes that break the code

… changes that break the tests

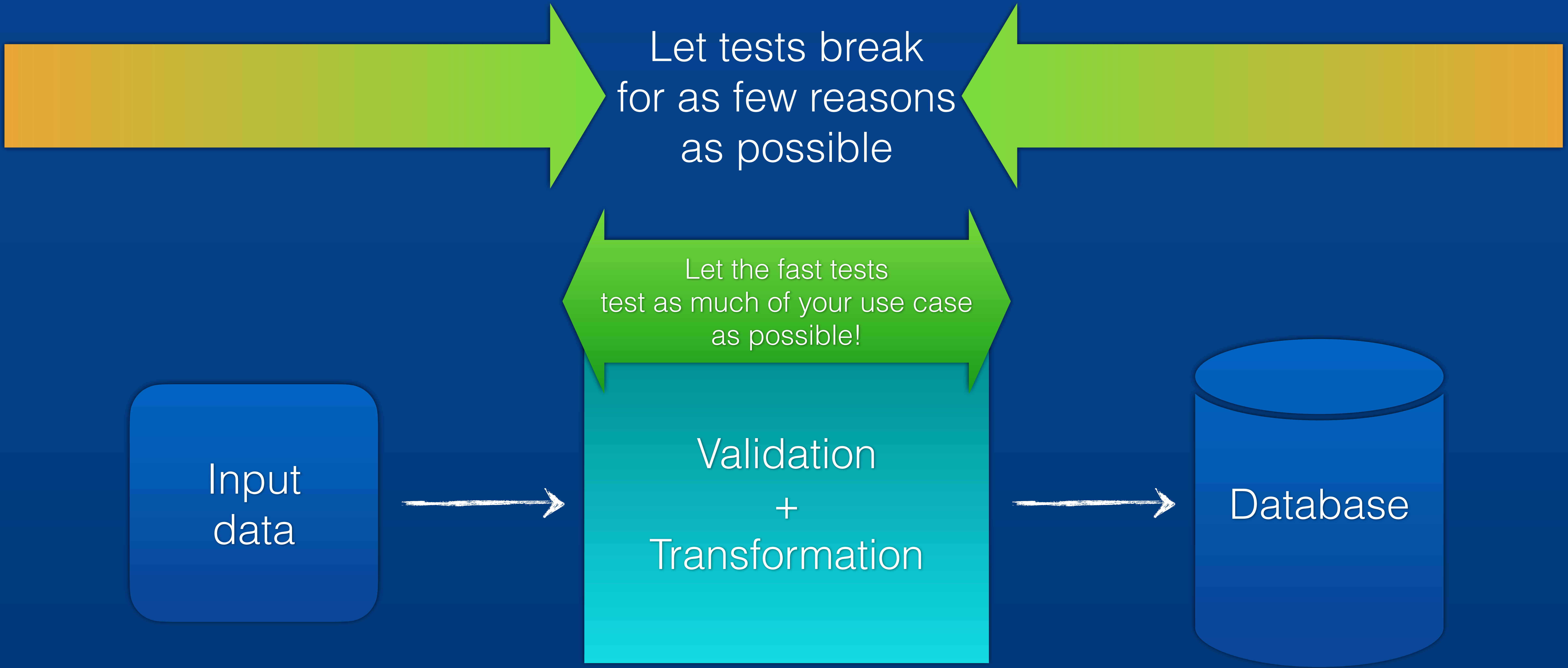


Input / transport (file) format
Some framework (Spring, Java EE etc.)
Business requirements
Persistence framework & entity representation
DB schema
Input data
Validation + Transformation
Database

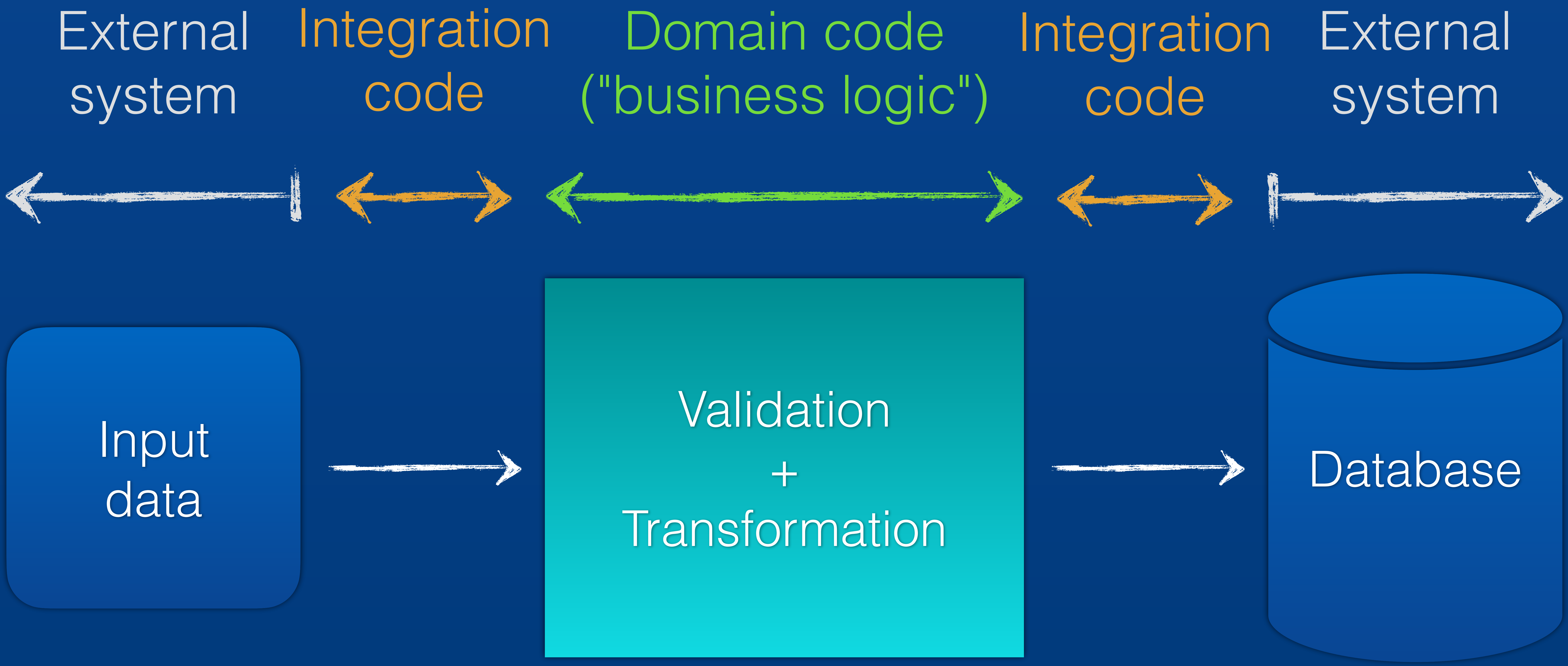
What & Where to Test?



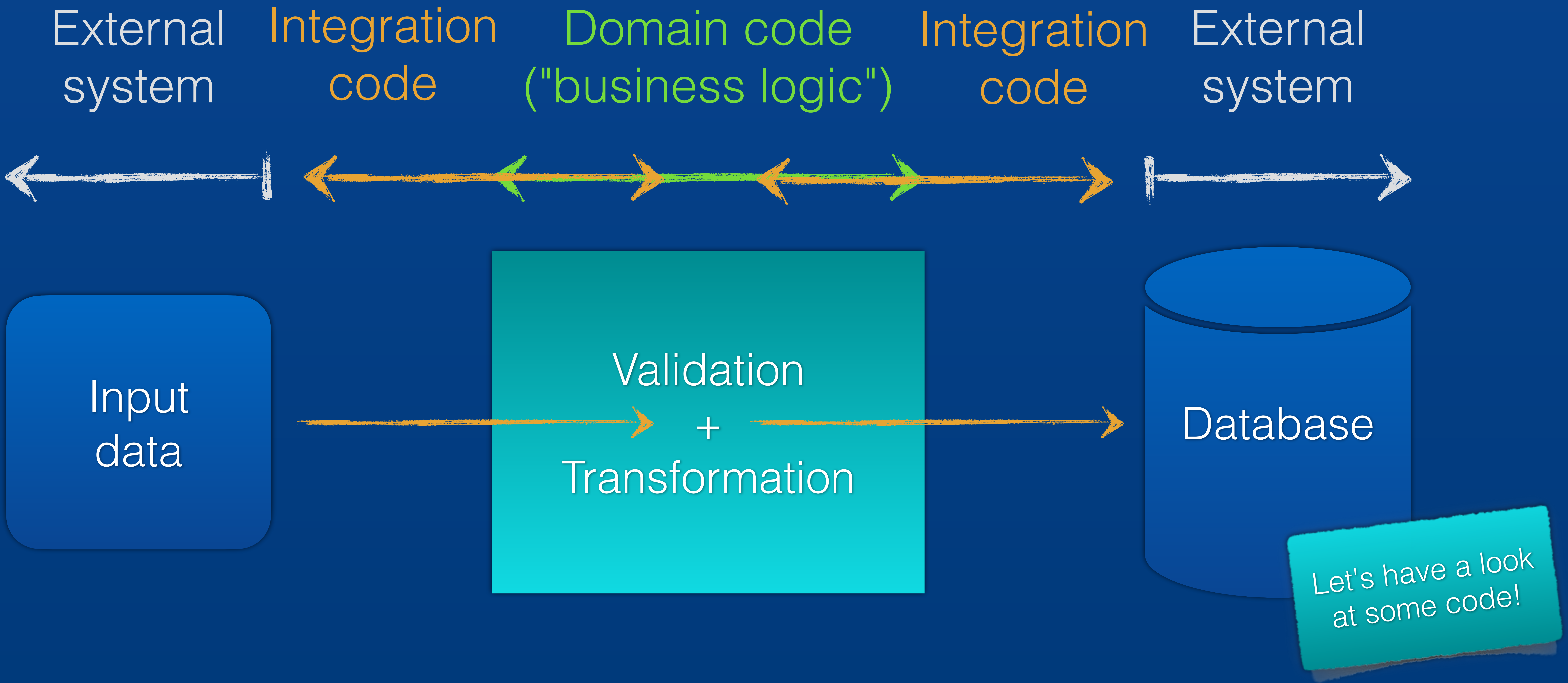
Few Dependencies, Maximum Scope



Ideal World – Clear Responsibilities



Dependencies Creep Into Our Domain


External system
Integration code
Domain code ("business logic")
Integration code
External system
Input data
Validation + Transformation
Database
Let's have a look at some code!

Unwanted

Dependencies in Domain Code

```
function validateAndTransform(jsonEntity) {

    validateSomeBusinessRules(jsonEntity);
    transformSomeValues(jsonEntity);

    mapToDatabaseEntity(jsonEntity) -> dbEntity;
    saveToDatabase(dbEntity);
}
```

Oh no!

Unwanted

Dependencies in Domain Code

Nobody would code like this, right? 🤣😬

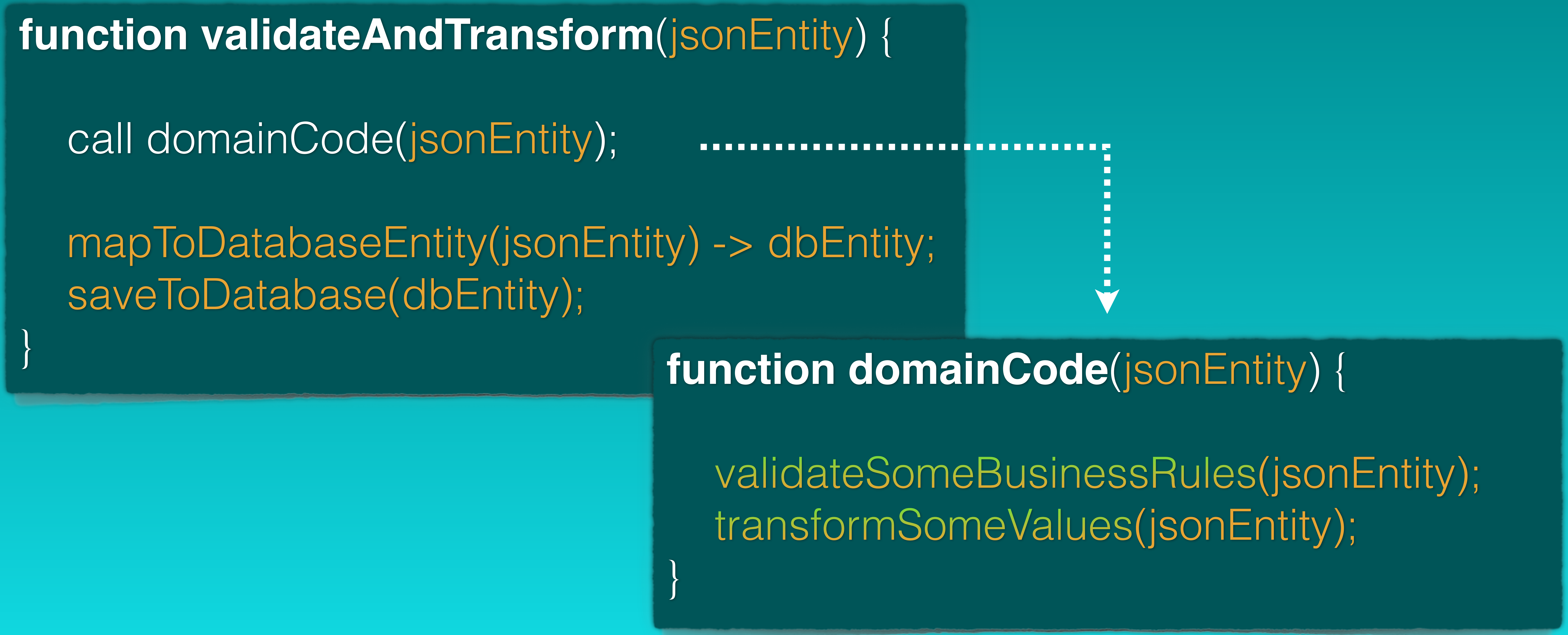
```
function validateAndTransform(jsonEntity) {

    validateSomeBusinessRules(jsonEntity);
    transformSomeValues(jsonEntity);

    mapToDatabaseEntity(jsonEntity) -> dbEntity;
    saveToDatabase(dbEntity);
}
```

Become aware of this strong coupling

Maybe time for some refactoring?

Unwanted

Dependencies in Domain Code



```
function validateAndTransform(jsonEntity) {

    call domainCode(jsonEntity);

    mapToDatabaseEntity(jsonEntity) -> dbEntity;
    saveToDatabase(dbEntity);
}
```

```
function domainCode(jsonEntity) {

    validateSomeBusinessRules(jsonEntity);
    transformSomeValues(jsonEntity);
}
```

Dependencies in Domain Code

```
function validateAndTransform(jsonEntity) {
    mapToDomainEntity(jsonEntity) -> domainEntity;
    call domainCode(domainEntity);

    mapToDatabaseEntity(domainEntity) -> dbEntity;
    saveToDatabase(dbEntity);
}
```

Lots of fast (unit) tests here
(including edge cases)

```
function domainCode(domainEntity) {

    validateSomeBusinessRules(domainEntity);
    transformSomeValues(domainEntity);
}
```

Only some integration tests here
("happy path" & error case?)

Patterns & Styles

Code design patterns

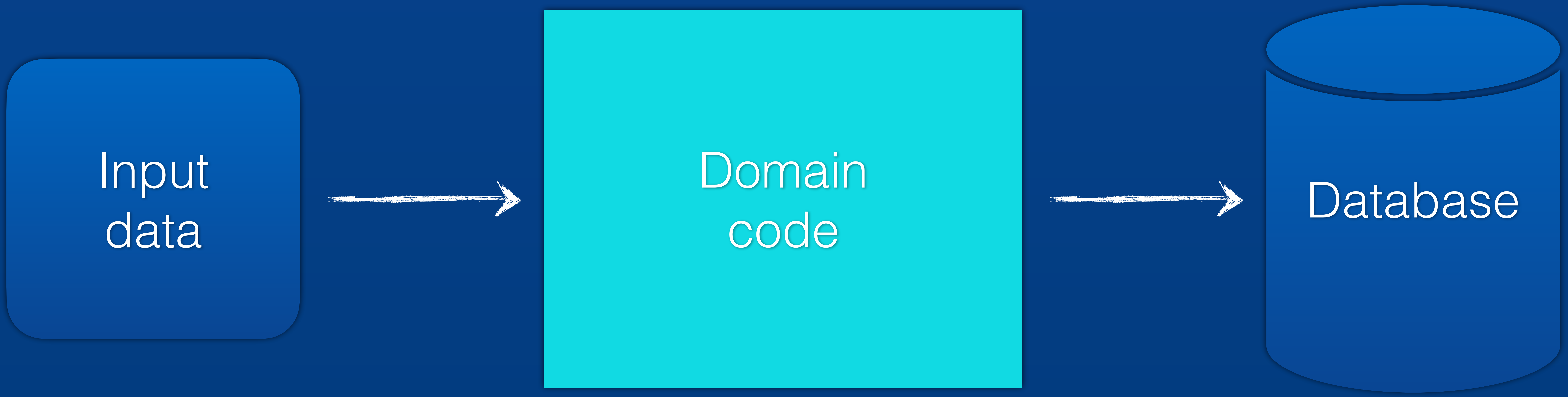
"Integration Operation Segregation Principle" (IOSP)

"Single Layer of Abstraction" (SLA)

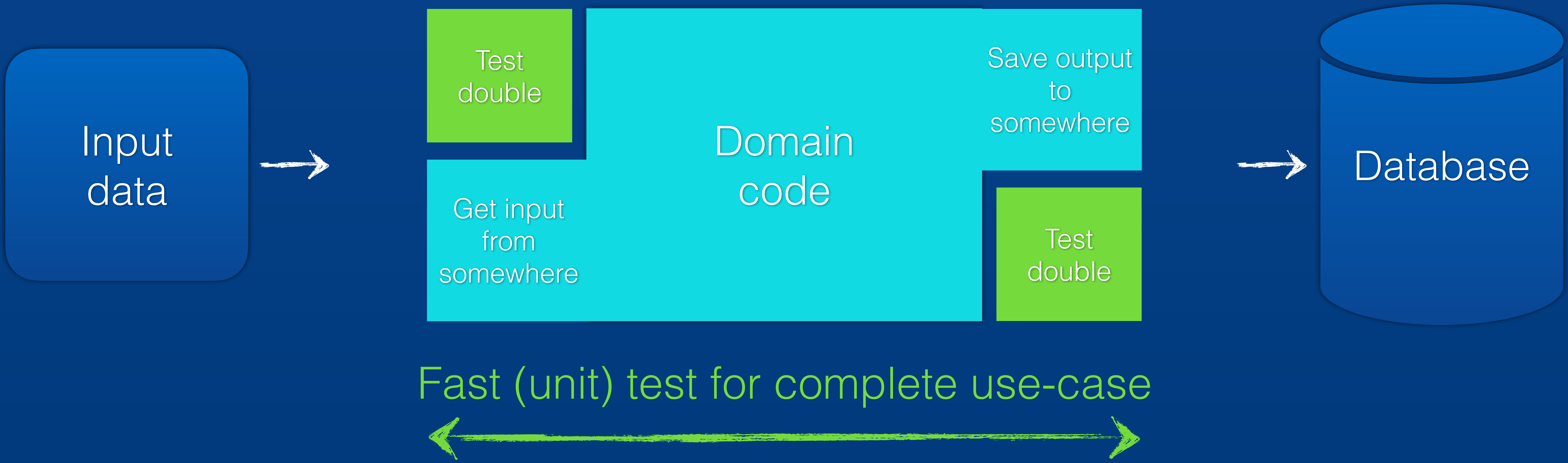
etc.

There are similar **architectural patterns & styles** as well!

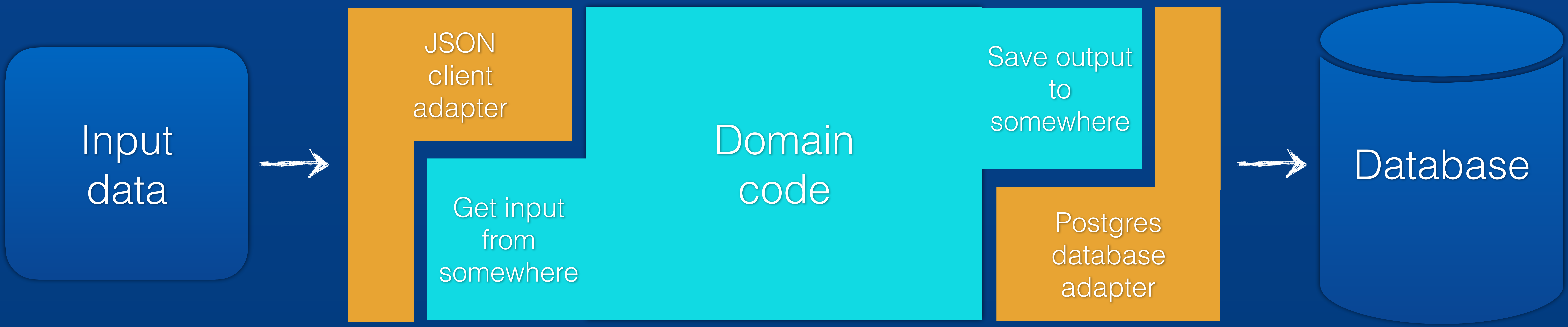
Architectural Patterns



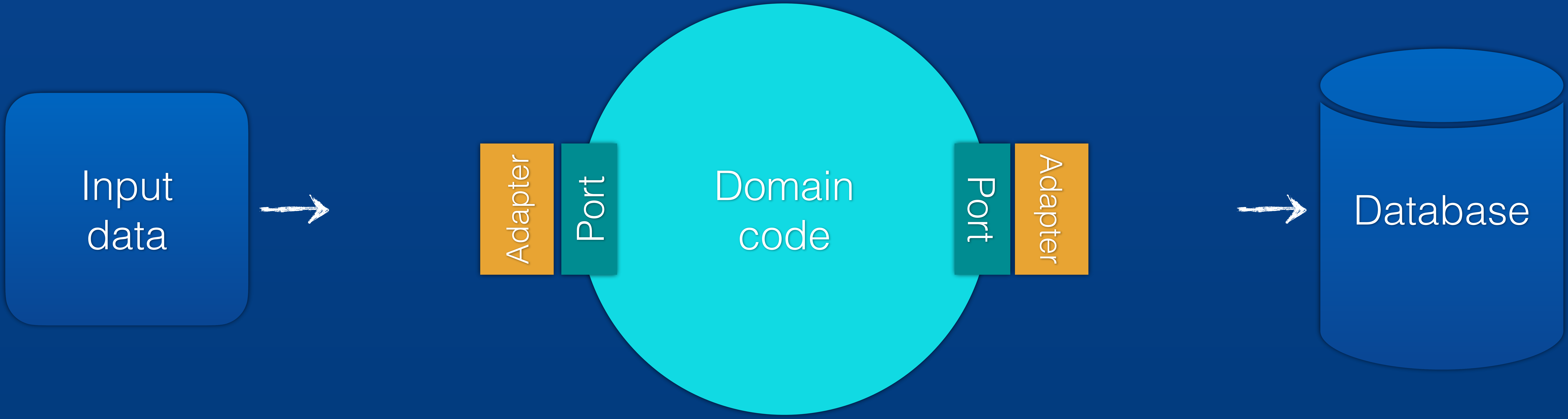
Architectural Patterns



Architectural Patterns

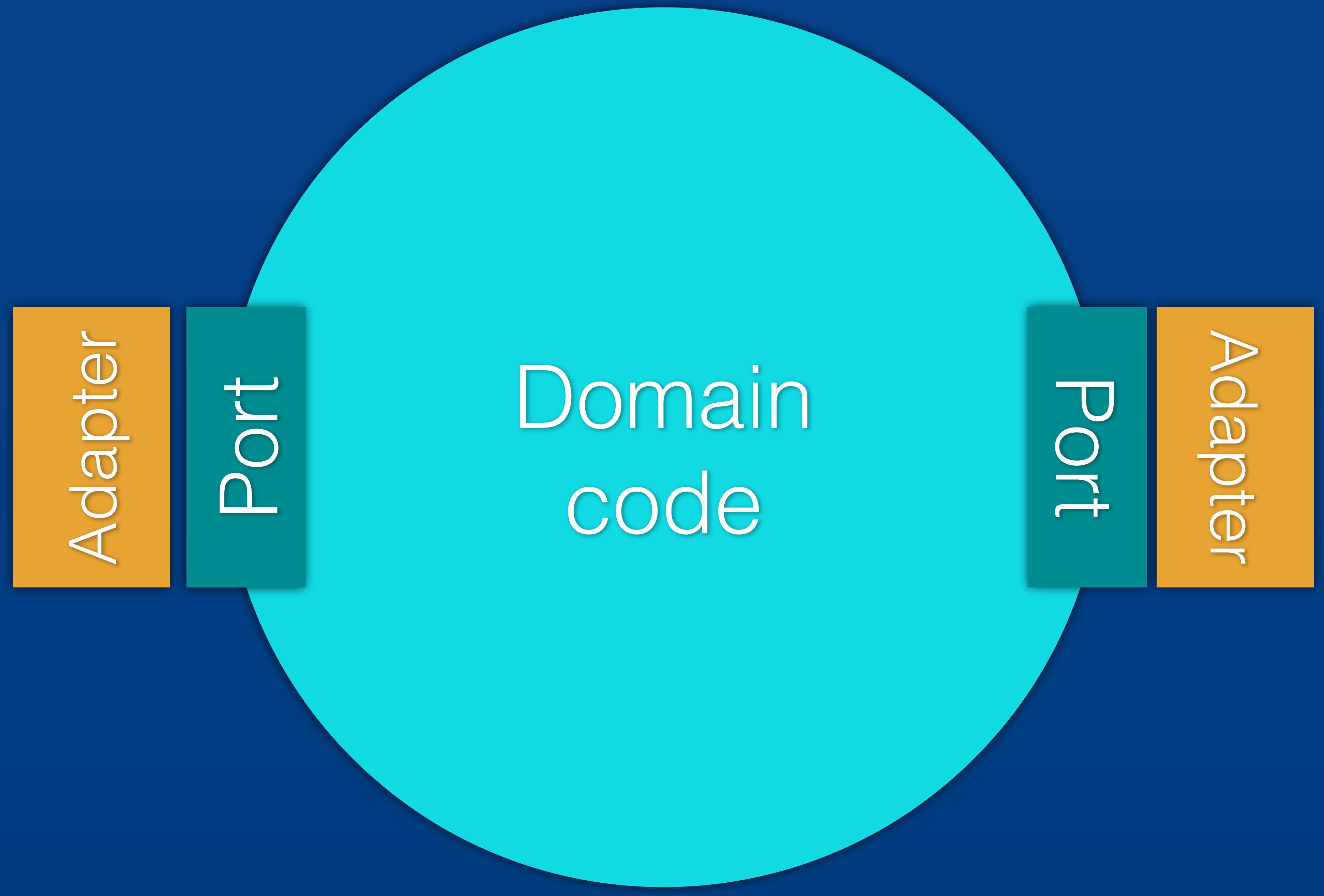


Architectural Styles

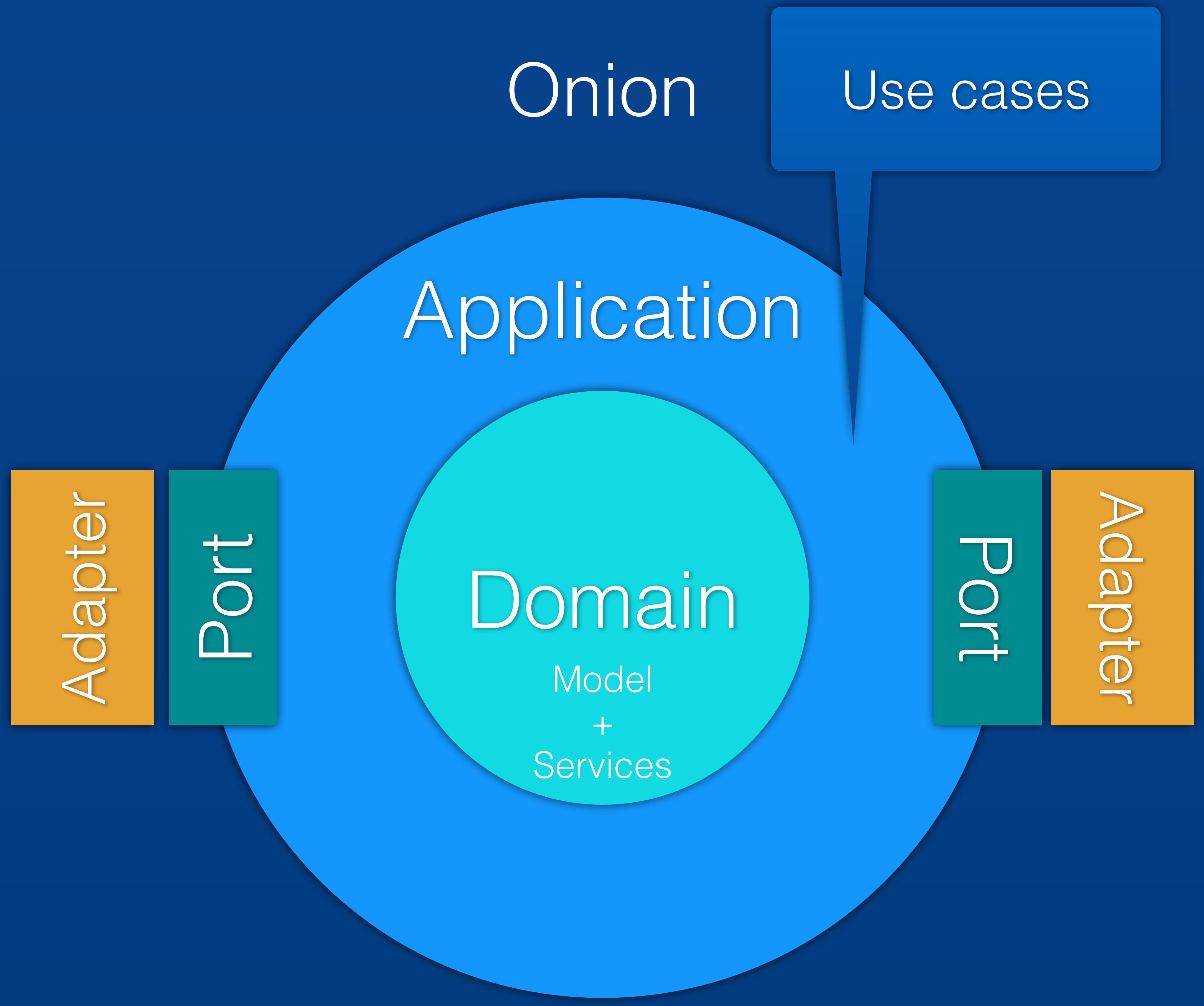


Architectural Styles

Ports & Adapters ("Hexagonal")



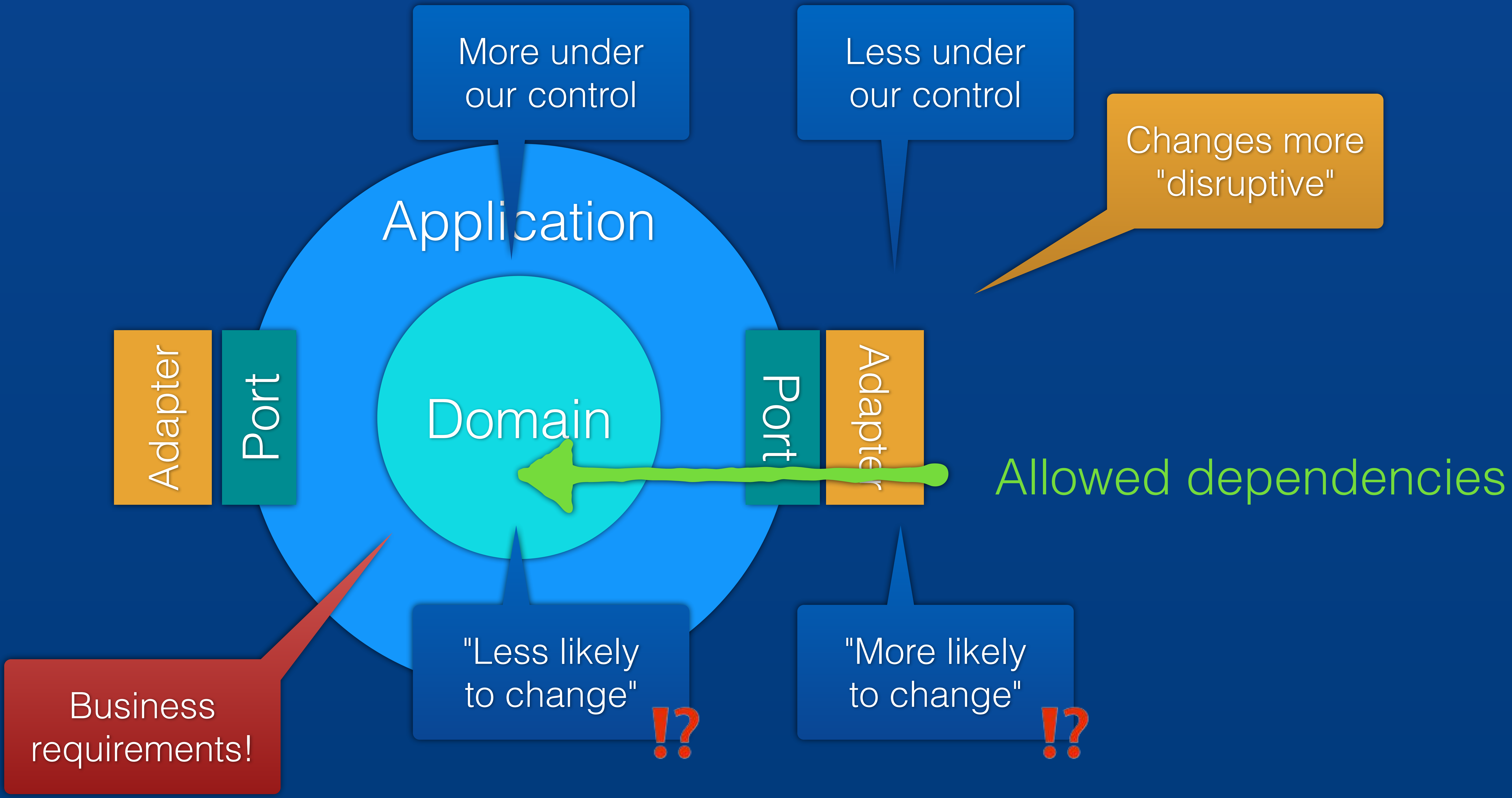
Onion



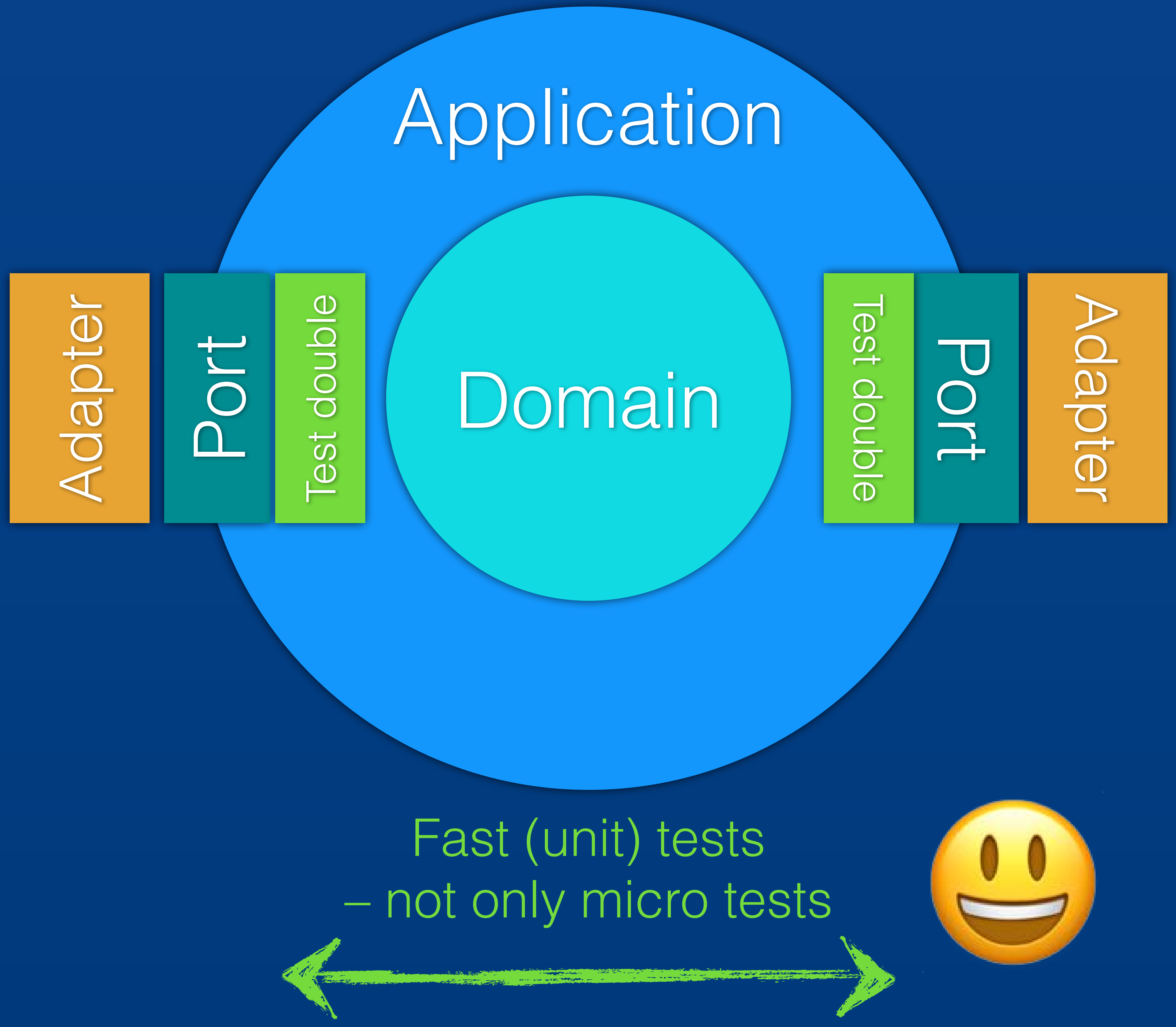
Outside & Inside for Longevity

No more layers (above vs. below)

More inside vs. outside



Fast Tests for Complete Use Cases



Patterns & Styles for Testability

Focus not so much on structuring code

Focus more on **ways of thinking** – on "why"

Learn how to **build software with testability in mind**

Easier to grasp

🧔👱‍♀️

Agile Developers
(XP Practitioners)

"Why not just do TDD all the way?"

Just enough tests.

Code design technique / strategy ✅

Makes your code testable (and probably your architecture, too) ✅

TDD is Not the Goal

TDD can be hard to grasp. May seem like ideology.

"Schools" can be confusing.

"Inside-out", "outside-in", "London", "Belfast" and "Berlin" …

Needs experience.

Especially for creating your architecture with TDD.

If TDD is the only cure, you'll often encounter reluctance

TDD is a Really Useful Tool

Motivate "why" of structuring patterns & styles for testability.

Then **use TDD as a means for "how".**

Bonus

Learn/show/experiment
how to use **TDD not only for micro tests** ("unit tests")

but for **fast tests of complete use cases**.

What BDD aims at. As TDD was intended?

Bonus

How to Keep Testability?

Test your architecture!

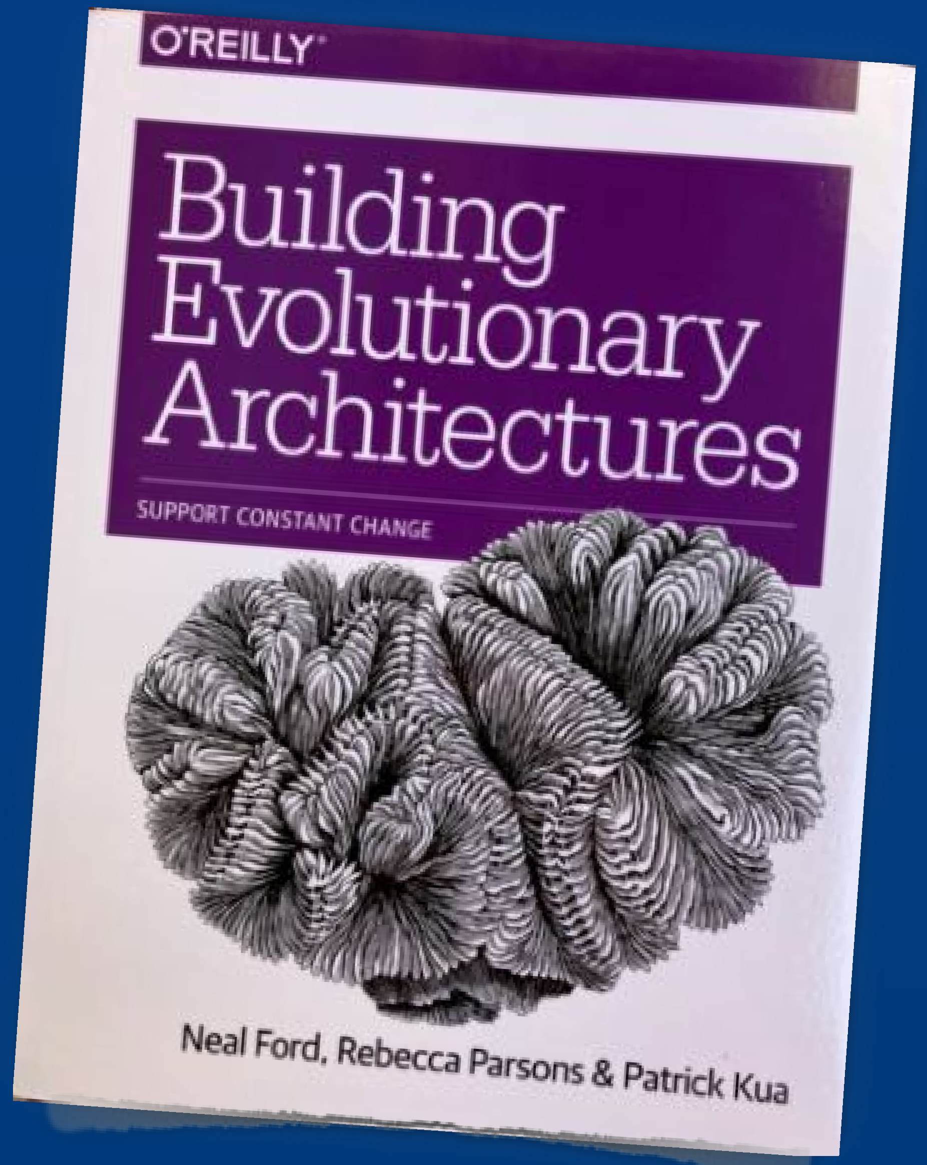
Build **Fitness Functions** for the core ideas behind your architecture

"dependencies from outside to inside only"
"no framework dependencies in domain code"
etc.

ArchUnit

Use suitable tools, for example **ArchUnit**

First step towards an **evolutionary architecture**?!


O'REILLY
Building Evolutionary Architectures
SUPPORT CONSTANT CHANGE
Neal Ford, Rebecca Parsons & Patrick Kua

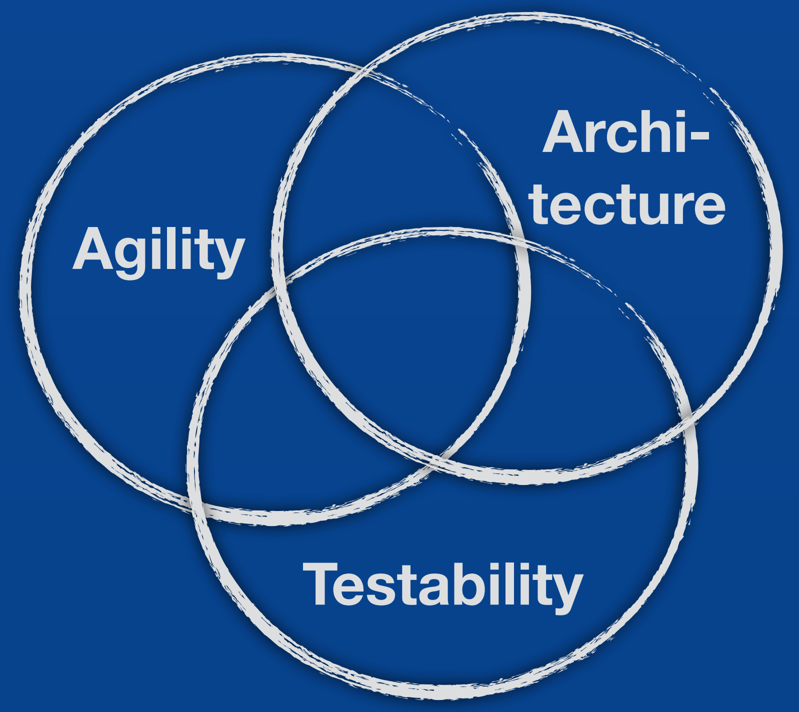
Wrap-Up

Testability at the Speed of Light

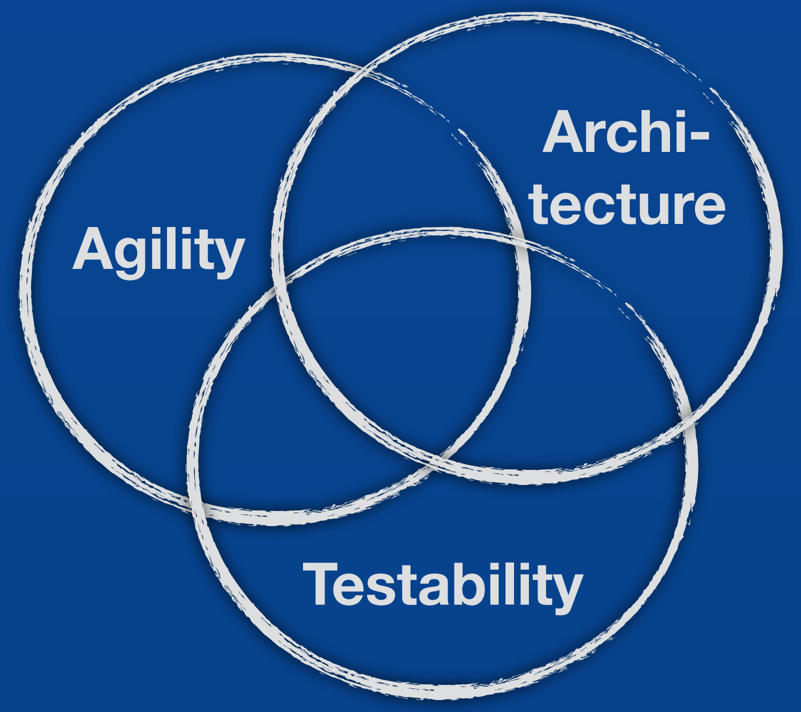
Testability is a **property of your architecture.**

Enables a **safety net that promotes agility.**

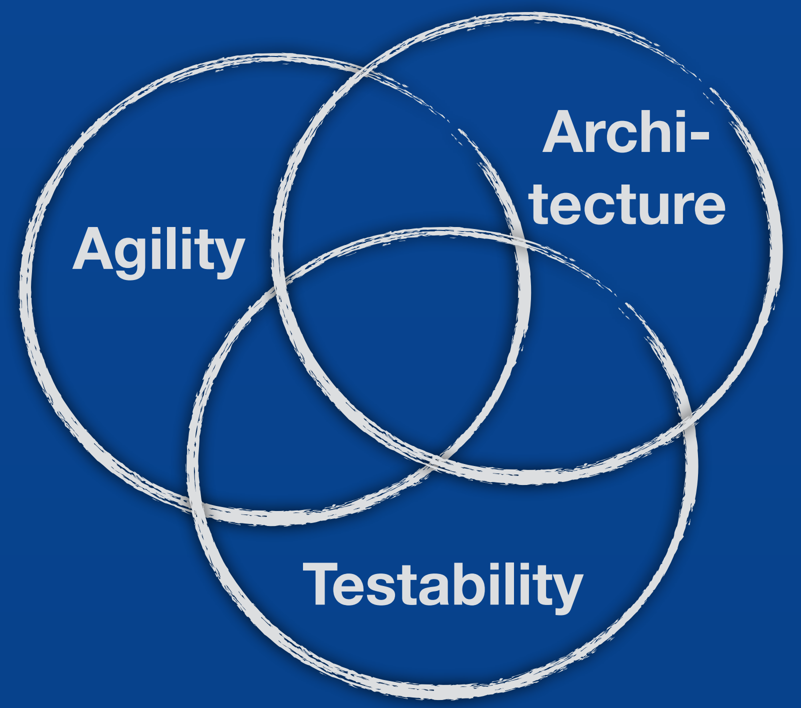
Learn how to build software with testability in mind!



Explore freedom of (design) choices



Start where you are



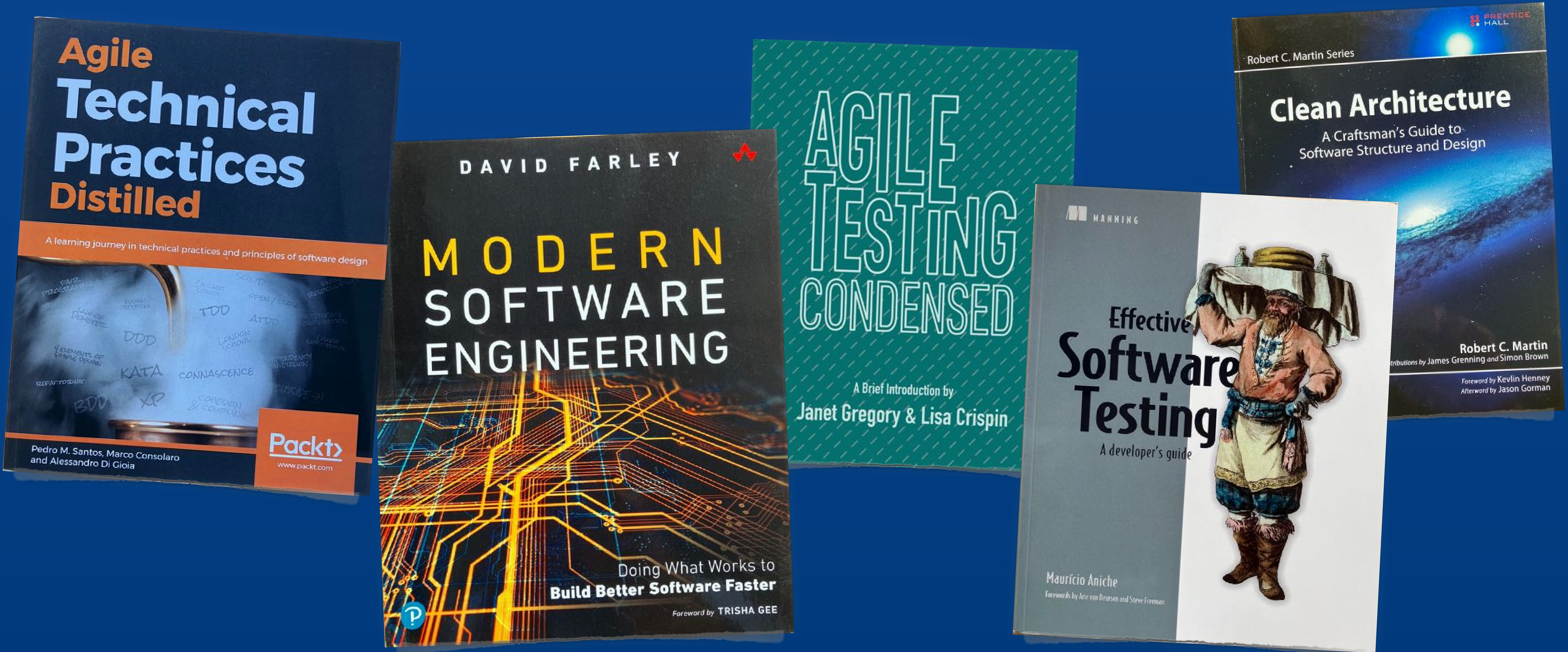
Collaborate.

Work together.

Code together.

Learn from each other.

Further Reading


Agile Technical Practices Distilled
A learning journey in technical practices and principles of software design
Pedro M. Santos, Marco Consolaro and Alessandro Di Gioia
Packt
www.packt.com
DAVID FARLEY
MODERN SOFTWARE ENGINEERING
Doing What Works to Build Better Software Faster
Foreword by TRISHA GEE
AGILE TESTING CONDENSED
A Brief Introduction by
Janet Gregory & Lisa Crispin
Robert C. Martin Series
PRENTICE HALL
Clean Architecture
A Craftsman's Guide to Software Structure and Design
Robert C. Martin
Contributions by James Grenning and Simon Brown
Foreword by Kevlin Henney
Afterword by Jason Gorman
MANNING
Effective Software Testing
A developer's guide
Maurício Aniche
Forewords by Arie van Deursen and Steve Freeman

Agile meets Architecture

Agility

Fast Tests

Testability

Cohesion

Let's talk!

(Q & A)

Decoupling

Architecture

Dependencies

Code Design

Continuous Delivery

@thmuch

Fast Feedback Loops

Agile meets Architecture

Thank You 😊

Die Techniker

www.tk.de/IT

@thmuch

"Do not depend on volatile things"
(Robert C. Martin)

"Make the change easy (this can be hard!), then make the easy change"
(Kent Beck)

"Many More Much Smaller Steps"
(GeePaw Hill)