



CONFOO.CA

HOW DO YOU STRUCTURE YOUR APPS?

KAT ZIEN // @KASIAZIEN

QUESTIONS, DECISIONS 🤔

- ▶ Should I put everything in one namespace?
- ▶ Should I start with one and extract other namespaces over time?
- ▶ How do I decide if something should be in its own namespace?
- ▶ Should I just use a framework?
- ▶ Microservices or monolith?
- ▶ How much should be shared?
- ▶ What's the “best way”?



GOOD STRUCTURE GOALS

GOOD STRUCTURE GOALS

- ▶ Consistent.
- ▶ Easy to understand, navigate and reason about ("makes sense").
- ▶ Easy to change, loosely-coupled.
- ▶ Easy to test.
- ▶ Little to no technical debt.
- ▶ "As simple as possible, but no simpler."
- ▶ Design reflects exactly how the software works.
- ▶ Structure reflects the design exactly.



How the software works



Design



Structure



ARCHITECTURE



Maintainability
Understandability

- ▶ Testability
- ▶ Little technical debt

A (SIMPLE) BEER REVIEWING SERVICE

- ▶ Users can add a review for a beer.
- ▶ Users can list all beer reviews.
- ▶ Option to store data either in memory or in a JSON file.

FLAT STRUCTURE

FLAT STRUCTURE

```
/  
├── handlers.php  
├── models.php  
├── storage.php  
├── storage_mem.php  
├── storage_json.php  
└── index.php
```

GROUP BY FUNCTION

GROUP BY FUNCTION (“LAYERED ARCHITECTURE”)

- ▶ presentation / user interface
- ▶ business logic
- ▶ storage / external dependencies



GROUP BY FUNCTION

```
•
  ├── handlers
  |   ├── list_reviews.php
  |   └── add_review.php
  ├── index.php
  ├── models
  |   ├── beer.php
  |   ├── review.php
  |   └── storage.php
  └── storage
      ├── json.php
      └── memory.php
```

GROUP BY MODULE

GROUP BY MODULE

```
.
```

- ├── beers
 - | ├── beer.php
 - | └── handler.php
- ├── index.php
- ├── reviews
 - | ├── handler.php
 - | └── beer_review.php
- └── storage
 - | ├── json.php
 - | ├── memory.php
 - | └── storage.php

GROUP BY CONTEXT



IMPLEMENTING DOMAIN-DRIVEN DESIGN



VAUGHN VERNON

FOREWORD BY ERIC EVANS

DOMAIN DRIVEN DESIGN! (DDD)

DOMAIN DRIVEN DESIGN (DDD)

- ▶ Establish your domain and business logic.
- ▶ Define your bounded context(s), the models within each context and the ubiquitous language.
- ▶ Categorising the building blocks of your system:

Entity

Value Object

Domain Event

Aggregate

Service

Repository

Factory

DEMO PROJECT

Context: beer tasting

Language: beer, review, storage, ...

Entities: Beer, Review, ...

Value Objects: Brewery, Author, ...

Aggregates: BeerReview

Service: Beer Review adder, Beer Review lister

Events: Beer added, Review added, Duplicate review, ...

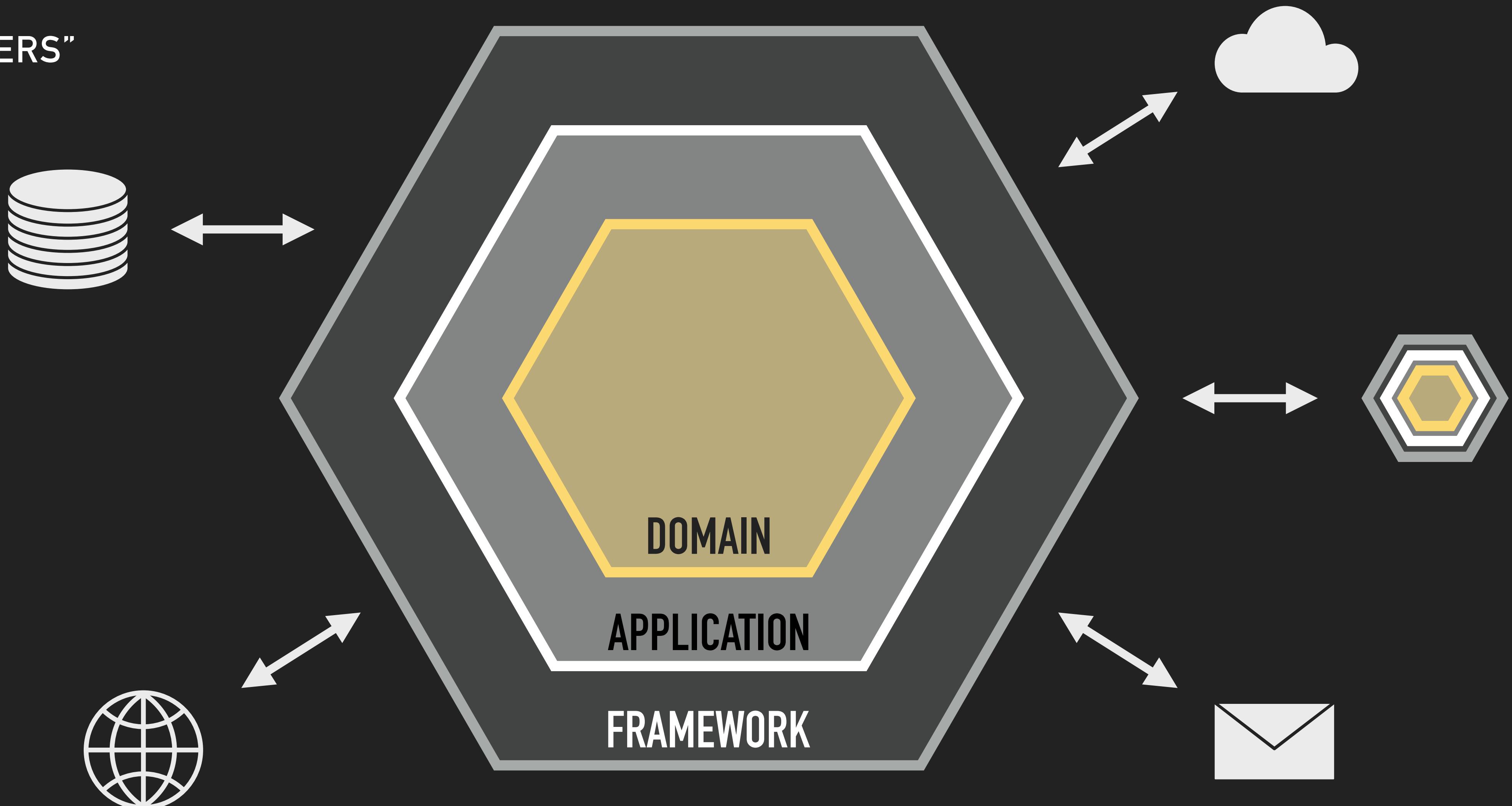
Repository: Beer repository, Review repository

GROUP BY CONTEXT

- - |- **reviewing**
 - |- **handler.php**
 - |- **beer_review.php**
 - |- **listing**
 - |- **handler.php**
 - |- **reviews.php**
 - |- **index.php**
 - |- **storage**
 - |- **json.php**
 - |- **memory.php**
 - |- **storage.php**

HEXAGONAL ARCHITECTURE

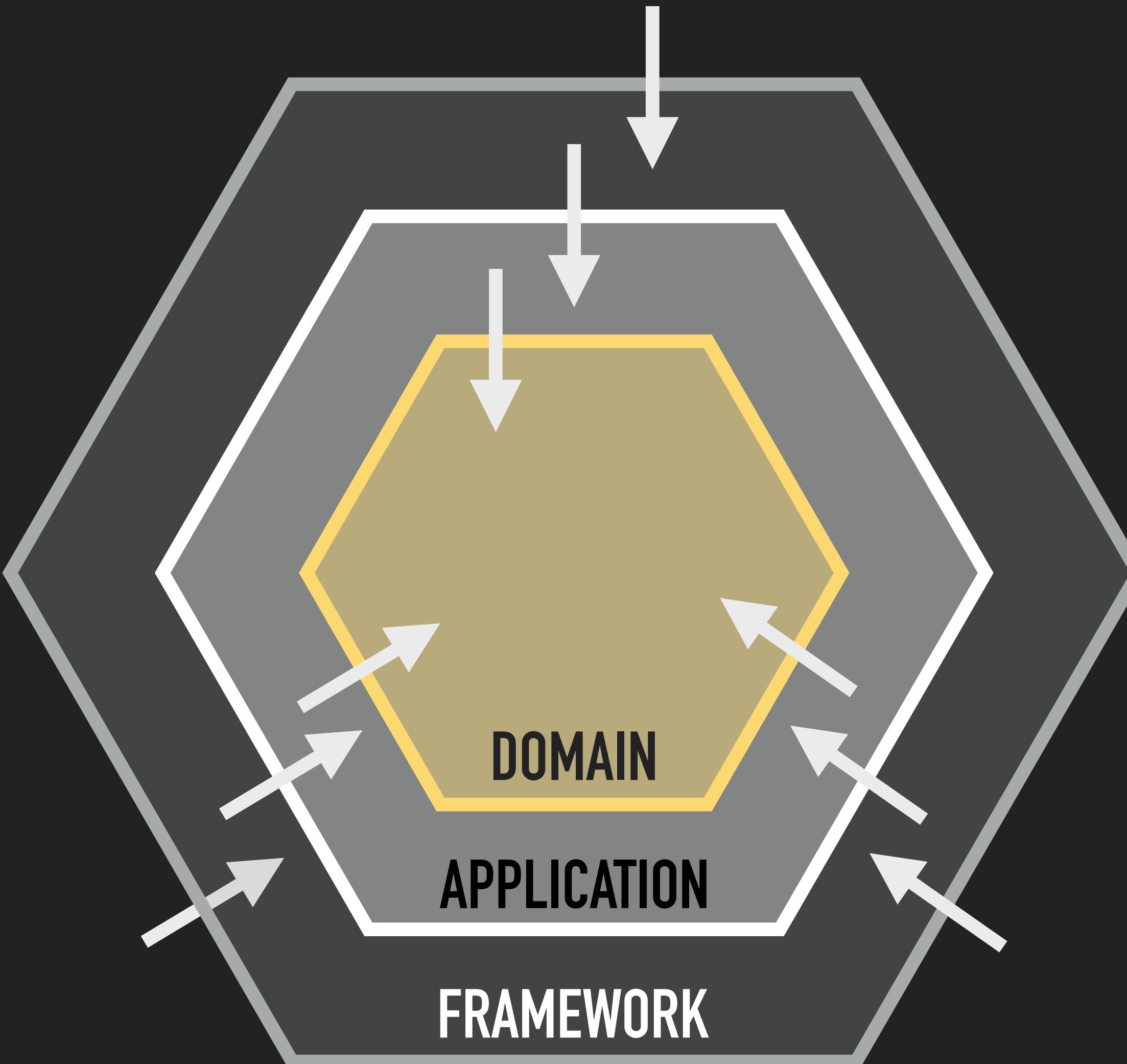
“PORTS AND ADAPTERS”



HEXAGONAL ARCHITECTURE

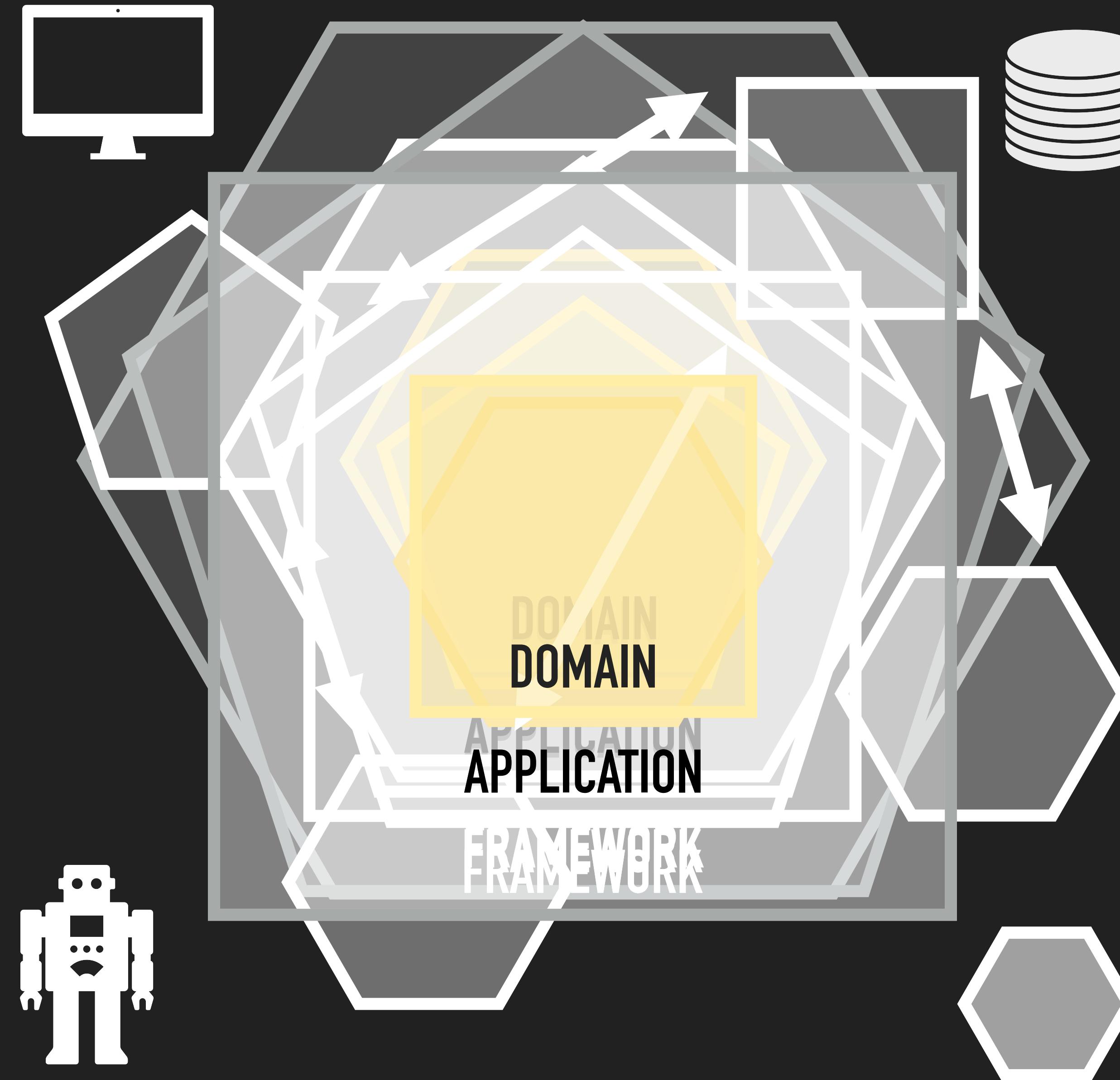
“PORTS AND ADAPTERS”

Dependencies
only point inwards.

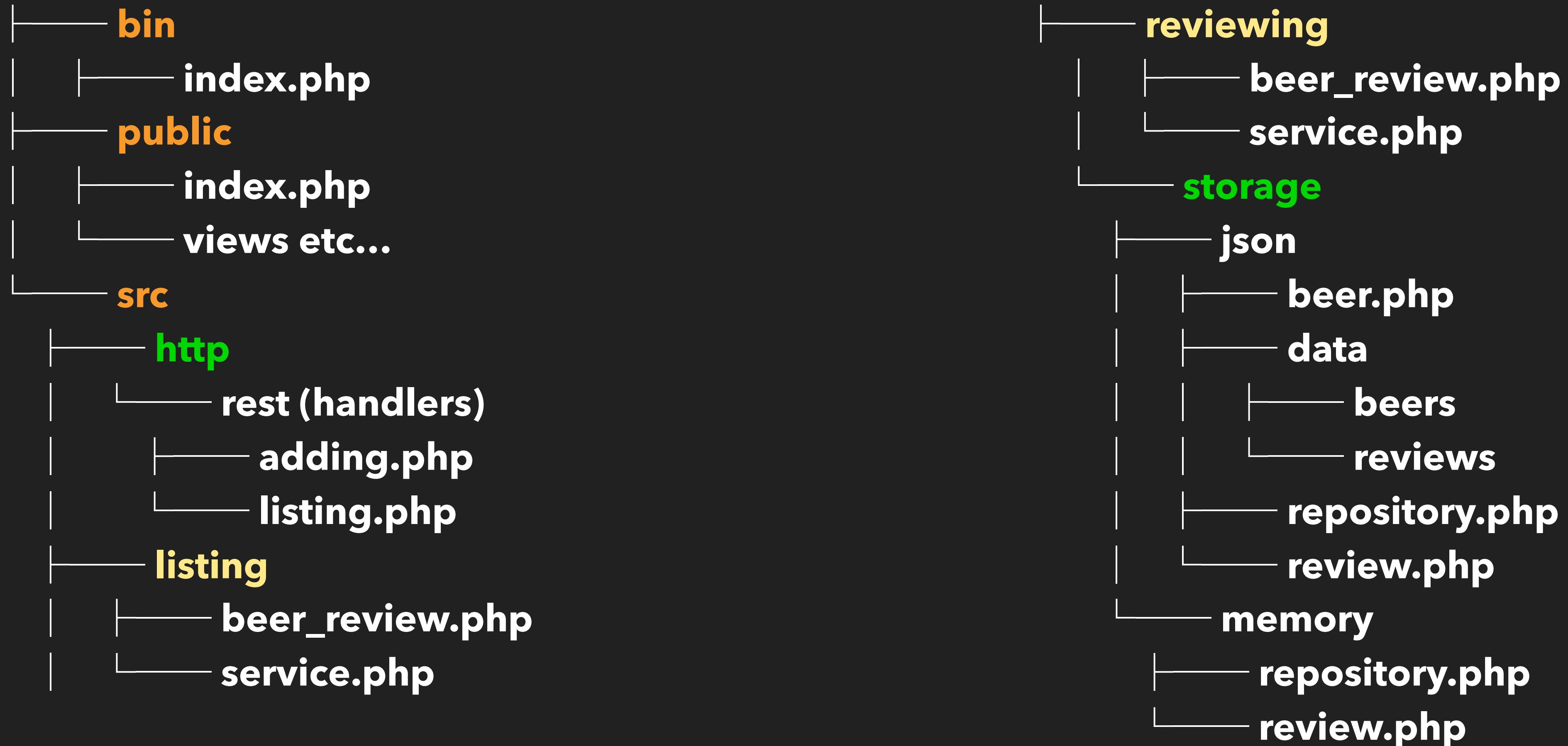


HEXAGONAL ARCHITECTURE

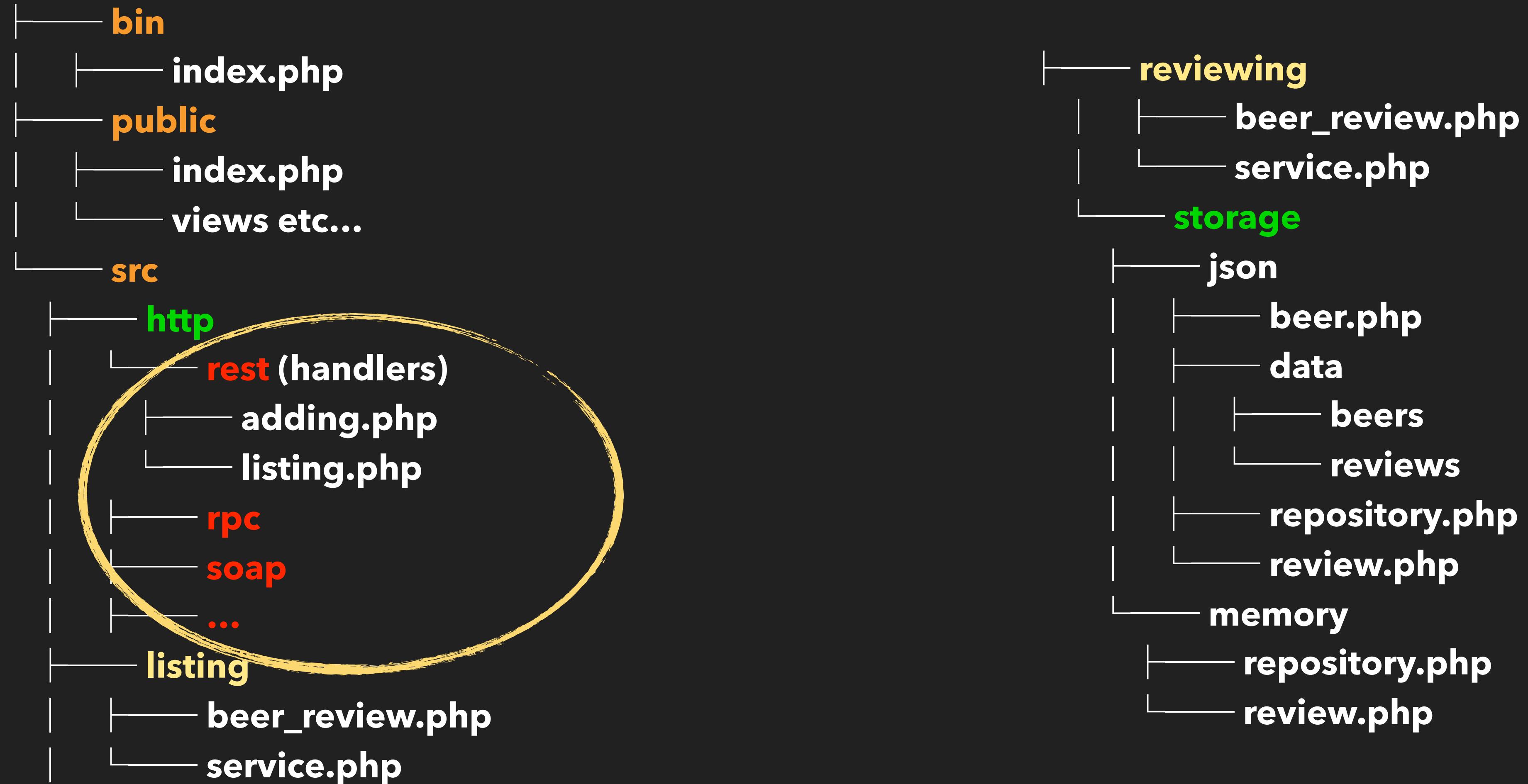
“PORTS AND ADAPTERS”



HEXAGONAL ARCHITECTURE



HEXAGONAL ARCHITECTURE





FRAMEWORKS?

TESTING

- ▶ Unit test `#allthedomain!`
- ▶ Easy mocking of external dependencies

NAMING

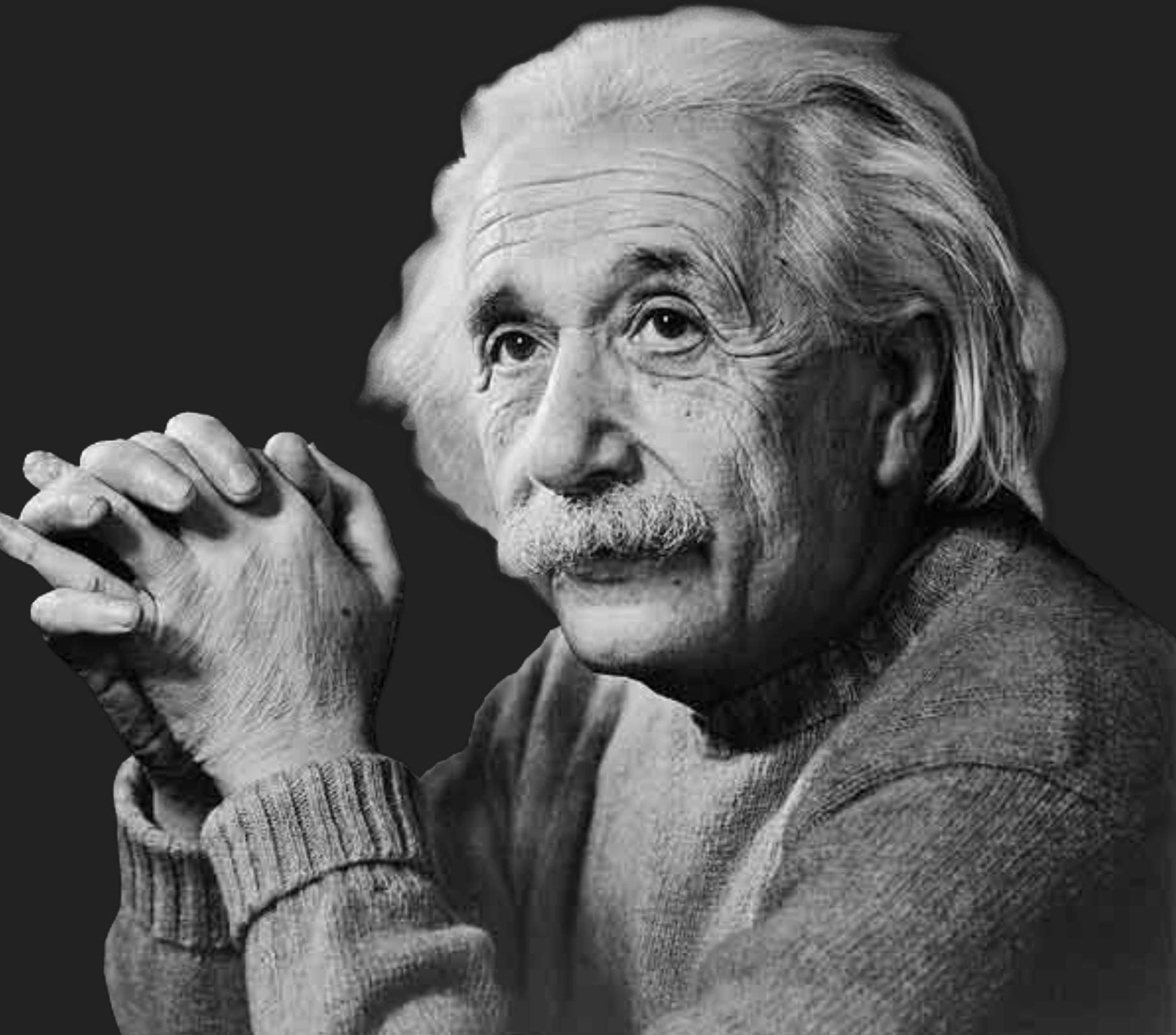
- ▶ Choose names that suggest well what can be expected inside.
 - ▶ communicate what they provide, as opposed to what they contain
- ▶ Avoid generic names like util, common etc.
- ▶ Avoid stutter (e.g. Controllers/FooBarController -> Controllers/FooBar).
- ▶ Accessibility - choose screenreader-friendly names! (see [Julia's talk](#))

PUTTING IT ALL TOGETHER

- 👌 Flat and simple is ok.
- ☝ Two top-level directories:
bin|public (for your binaries/doc root) and src (for all teh codez)
- ✌ Group by context, not generic functionality. Try DDD + hex.
- 👉 All other project files (composer, phpunit, docker): root dir of your project.
- ✋ index.php initialises and ties everything together.
- 👍 Keep the business logic centralised.

CONCLUSION

- ▶ No single right answer (sorry...) 🙄
- ▶ "Be like water"
- ▶ "As simple as possible, but no simpler"
- ▶ Maintain consistency
- ▶ Experiment!
- ▶ Share your ideas 🙌



QUESTIONS? LINKS!

- ▶ @kasiazien
- ▶ demo code: coming soon to a GitHub near you!
- ▶ references:

The Hexagonal (Ports & Adapters) Architecture by Alistair Cockburn

Hexagonal architecture by Chris Fidao

THANK YOU!

ConFoo.CA
DEVELOPER CONFERENCE

KAT ZIĘŃ // @KASIAZIEN