GO WAY FEST 2.0

HOW DO YOU STRUCTURE YOUR GO APPS?

KAT ZIEŃ // @KASIAZIEN

image: https://www.flickr.com/photos/hisgett/24764746083

🖐 KAT ZIEŃ

@kasiazien

QUESTIONS, DECISIONS 🤔

- Should I put everything in the main package?
- Should I start with one package and extract other packages over time?
- How do I decide if something should be in its own package?
- Should I just use a framework?
- What's the programming paradigm for Go?
- Microservices or monolith?
- How much should be shared between packages?

BECAUSE IF GO IS GOING TO BE A LANGUAGE THAT COMPANIES INVEST IN FOR THE LONG TERM, THE MAINTENANCE OF GO PROGRAMS, THE EASE OF WHICH THEY CAN CHANGE, WILL BE A KEY FACTOR IN THEIR DECISION.

Dave Cheney, Golang UK 2016 keynote

Image: https://www.flickr.com/photos/photommo/4082927817 2

GOOD STRUCTURE GOALS

GOOD STRUCTURE GOALS

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

image: https://www.flickr.com/photos/sergejf/15849692023

How the software works

☝🏽

Design

☝🏾

Structure

A BEER REVIEWING SERVICE

- Users can add a beer.
- Users can add a review for a beer.
- Users can list all beers.
- Users can list all reviews for a given beer.
- Option to store data either in memory or in a JSON file.
- Ability to add some sample data.

(for simplicity we'll skip deleting, updating and some error handling and tests 😱)

FLAT STRUCTURE

FLAT STRUCTURE

```
/
├── data.go
├── handlers.go
├── main.go
├── model.go
├── storage.go
├── storage_json.go
└── storage_mem.go
```

GROUP BY FUNCTION

GROUP BY FUNCTION (“LAYERED ARCHITECTURE”)

- presentation / user interface
- business logic
- external dependencies / infrastructure

GROUP BY FUNCTION

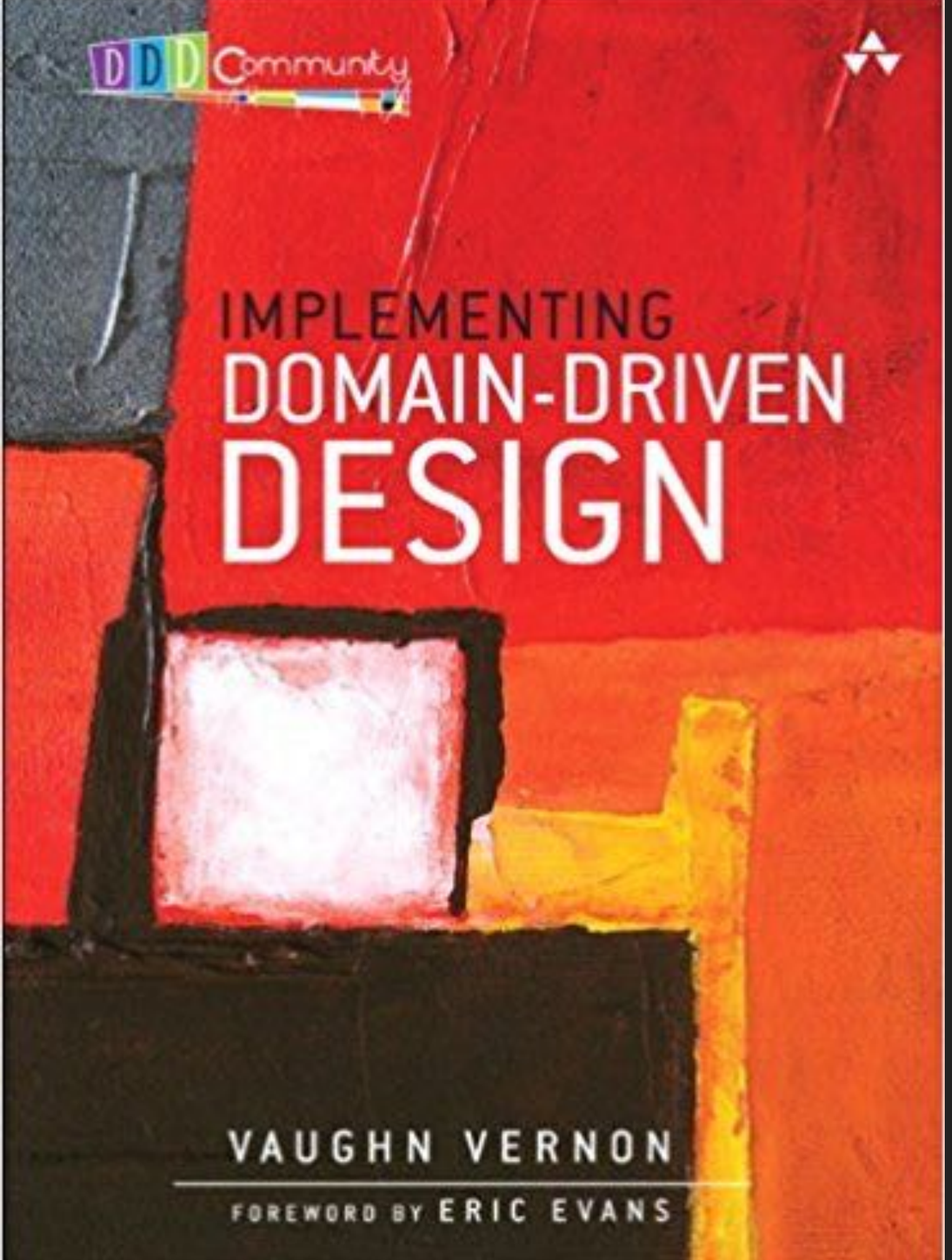
```
.
├── data.go
├── handlers
│   ├── beers.go
│   └── reviews.go
├── main.go
├── models
│   ├── beer.go
│   ├── review.go
│   └── storage.go
└── storage
    ├── json.go
    └── memory.go
```

GROUP BY MODULE

GROUP BY MODULE

```
.
├── beers
│   ├── beer.go
│   └── handler.go
├── main.go
├── reviews
│   ├── handler.go
│   └── review.go
└── storage
    ├── data.go
    ├── json.go
    ├── memory.go
    └── storage.go
```

GROUP BY CONTEXT



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 adder / adding, Review adder, Beer lister / listing, Review lister

Events: Beer added, Review added, Beer already exists, Beer not found, ...

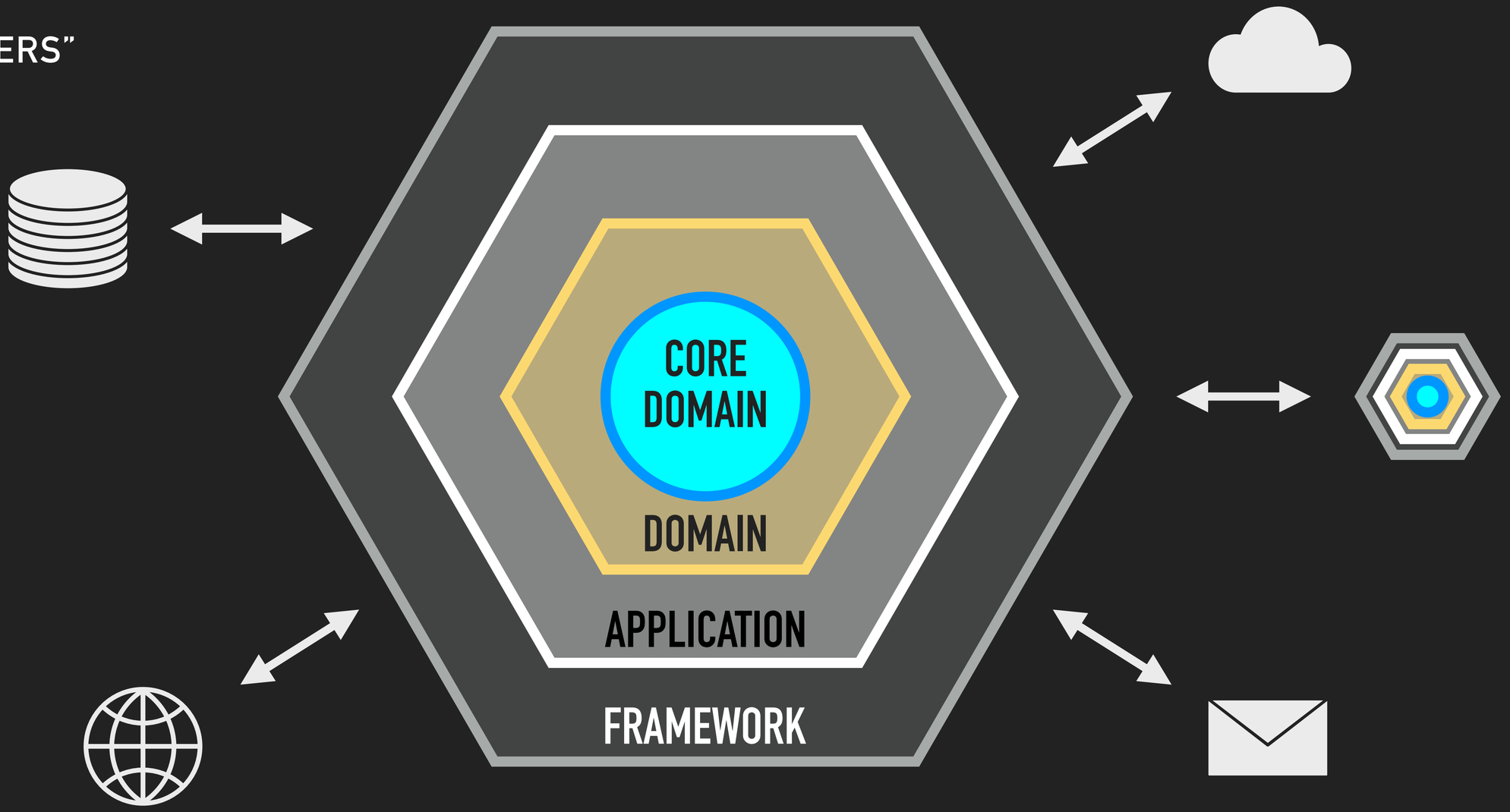
Repository: Beer repository, Review repository

GROUP BY CONTEXT

```
.
├── adding
│   ├── endpoint.go
│   └── service.go
├── beers
│   ├── beer.go
│   └── sample_beers.go
├── listing
│   ├── endpoint.go
│   └── service.go
├── main.go
├── reviewing
│   ├── endpoint.go
│   └── service.go
├── reviews
│   ├── review.go
│   └── sample_reviews.go
└── storage
    ├── json.go
    ├── memory.go
    └── type.go
```

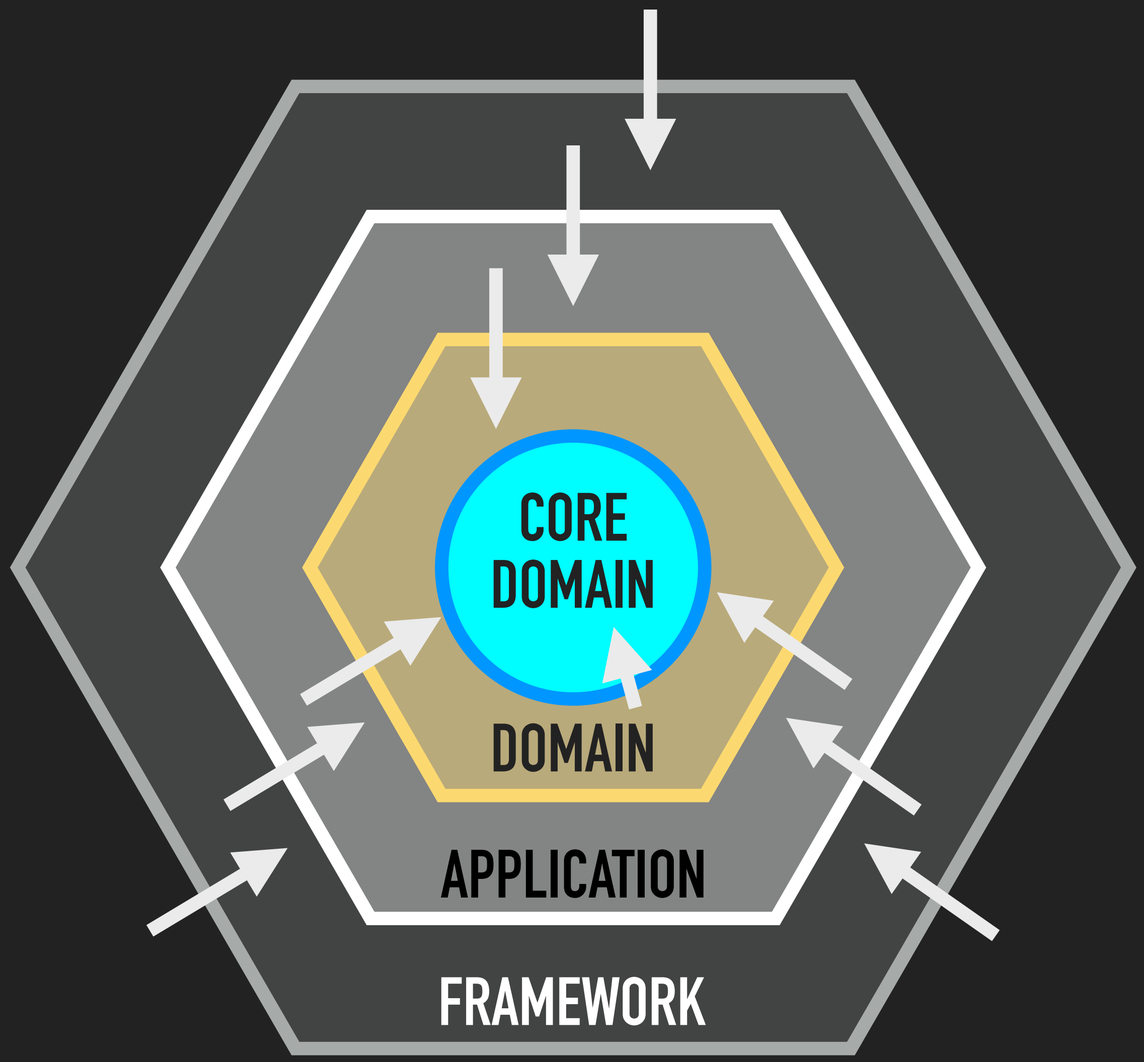
HEXAGONAL ARCHITECTURE

"PORTS AND ADAPTERS"



HEXAGONAL ARCHITECTURE

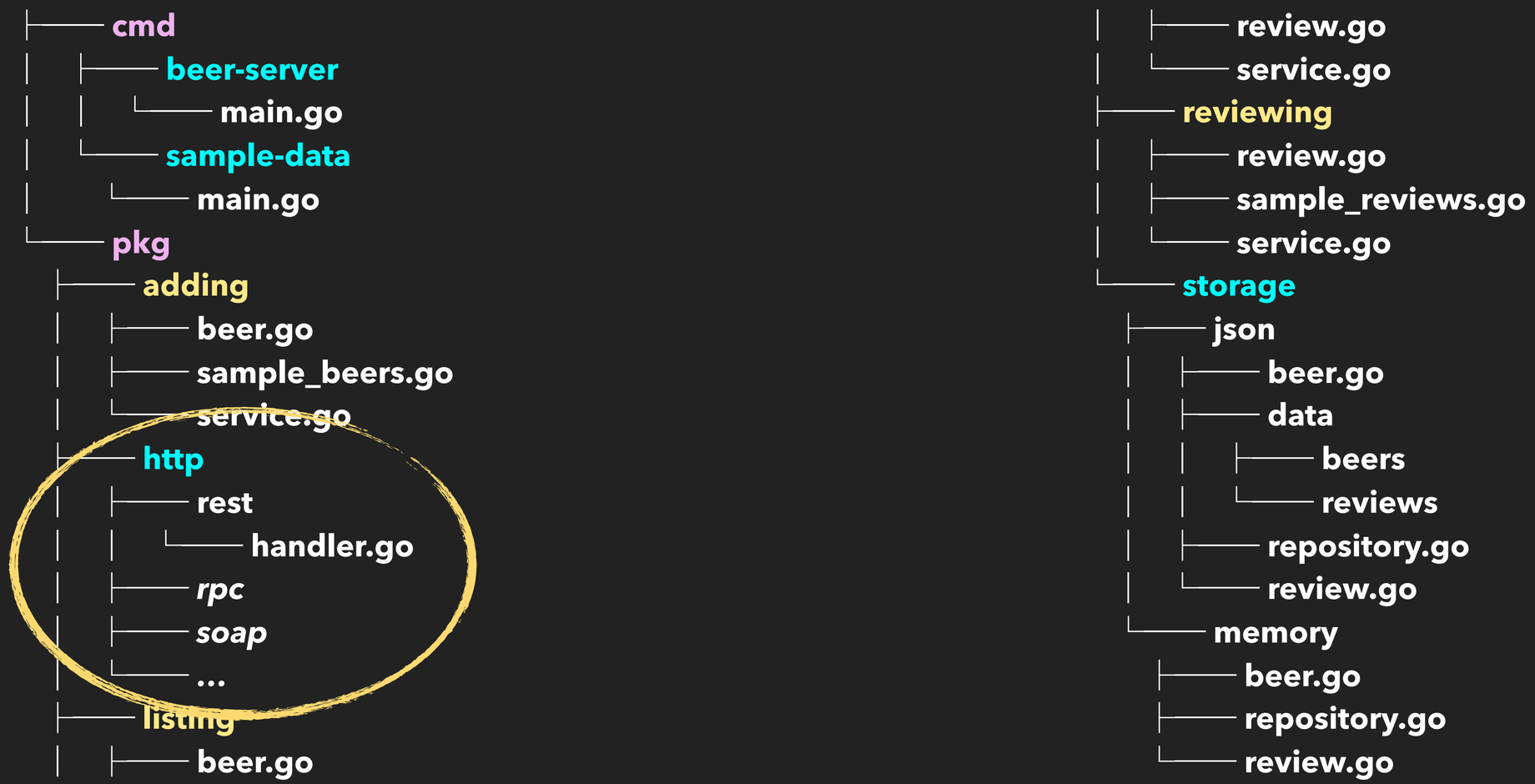
Dependencies
only point inwards.



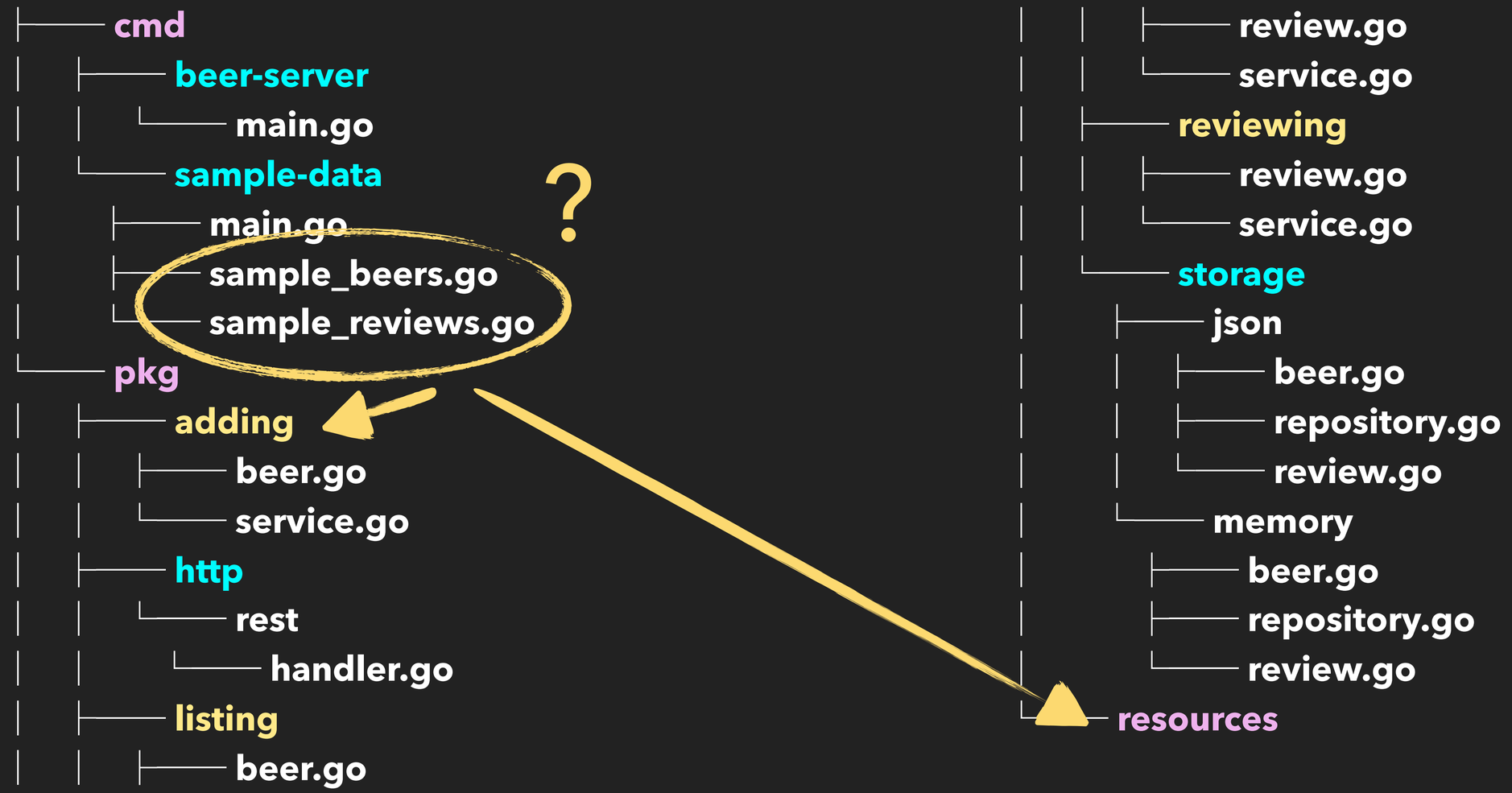
HEXAGONAL ARCHITECTURE

```
├── cmd
│   ├── beer-server
│   │   └── main.go
│   └── sample-data
│       └── main.go
└── pkg
    ├── adding
    │   ├── beer.go
    │   ├── sample_beers.go
    │   └── service.go
    ├── http
    │   └── rest
    │       └── handler.go
    ├── listing
    │   ├── beer.go
    │   ├── review.go
    │   └── service.go
    ├── reviewing
    │   ├── review.go
    │   ├── sample_reviews.go
    │   └── service.go
    └── storage
        ├── json
        │   ├── beer.go
        │   ├── data
        │   │   ├── beers
        │   │   └── reviews
        │   ├── repository.go
        │   └── review.go
        └── memory
            ├── beer.go
            ├── repository.go
            └── review.go
```

HEXAGONAL ARCHITECTURE



HEXAGONAL ARCHITECTURE



FRAMEWORKS?

Revel, Buffalo, Beego, Martini, …

intelligentbee.com/2017/08/14/golang-guide-list-top-golang-frameworks-ides-tools/

Shortcut for rolling on your own?

https://github.com/thockin/go-build-template

TESTING

- Keep the _test.go files next to the main files.
- Use a shared mock subpackage.

NAMING

- Choose package 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.
- Follow the usual go conventions (https://talks.golang.org/2014/names.slide).
- Avoid stutter (e.g. strings.Reader not strings.StringReader).
- Accessibility - choose screenreader-friendly names! (see Julia's talk)

MAIN

Mat Ryer @ GopherCon
@matryer

Following

No matter what kind of thing I'm building, my #golang main function ends up looking like this

```
func main() {
    if err := run(); err != nil {
        fmt.Fprintf(os.Stderr, "%v", err)
        os.Exit(1)
    }
}
```

4:13 PM - 13 Aug 2018

34 Retweets 215 Likes

PUTTING IT ALL TOGETHER IN GO

👌 Flat and simple is ok.

☝️ Two top-level directories:
cmd (for your binaries) and pkg (for your packages).

✌🏾 Group by context, not generic functionality. Try DDD / hex.

🤟🏼 Dependencies: own packages.

🖐🏻 Mocks: shared subpackage.

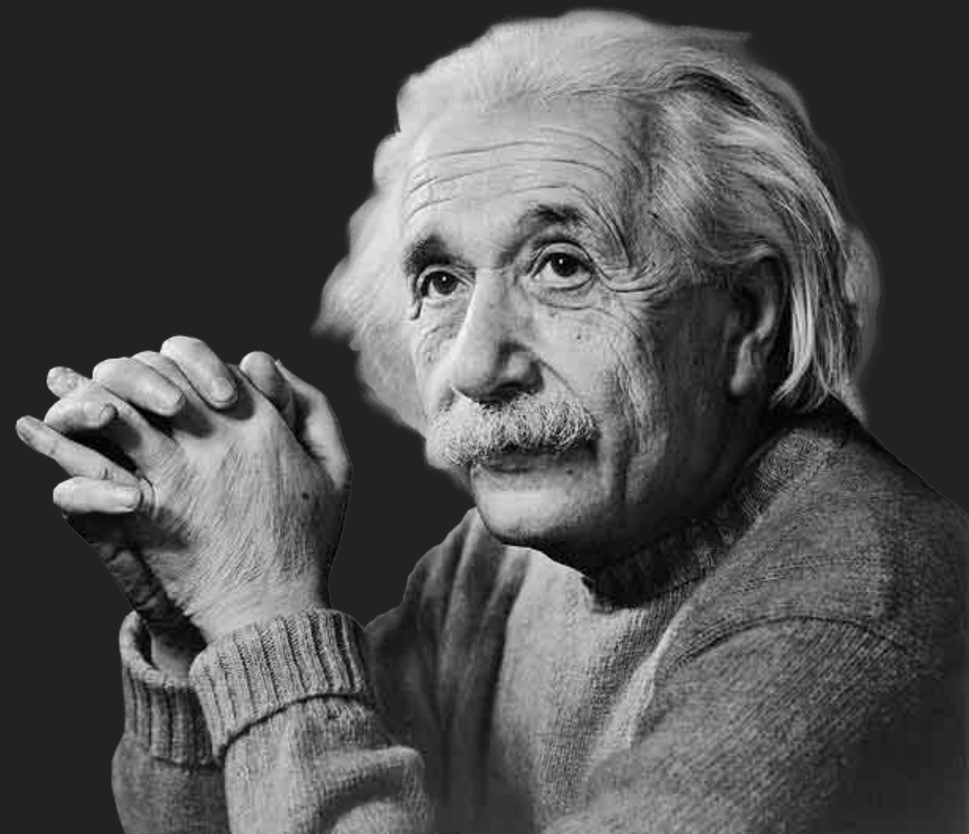
🖐 All other project files (fixtures, resources, docs, Docker, ...): root dir of your project.

🙌🏿 Main package initialises and ties everything together.

👎🏼 Avoid global scope and init().

CONCLUSION

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



(HUGE) CREDIT TO:

Peter Bourgon

peter.bourgon.org/blog/2017/06/09/theory-of-modern-go.html

Ben Johnson

medium.com/@benbjohnson/standard-package-layout-7cdbc8391fc1

Marcus Olsson

github.com/marcusolsson/goddd

QUESTIONS? LINKS!

- @kasiazien
- demo code: https://github.com/katzien/go-structure-examples
- references:

  Go and a Package Focused Design, Gopher Academy Blog

  Standard Package Layout by Ben Johnson

  Repository structure by Peter Bourgon

  Building an enterprise service in Go by Marcus Olsson

  Go best practices by Brian Ketelsen

  Hexagonal architecture by Chris Fidao

image: https://goway.io

THANK YOU!

KAT ZIEŃ // @KASIAZIEN