



GOPHERCON DENVER 2018

HOW DO YOU STRUCTURE
YOUR \Rightarrow **GO** APPS?

KAT ZIEŃ // @KASIAZIEN

👋 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.



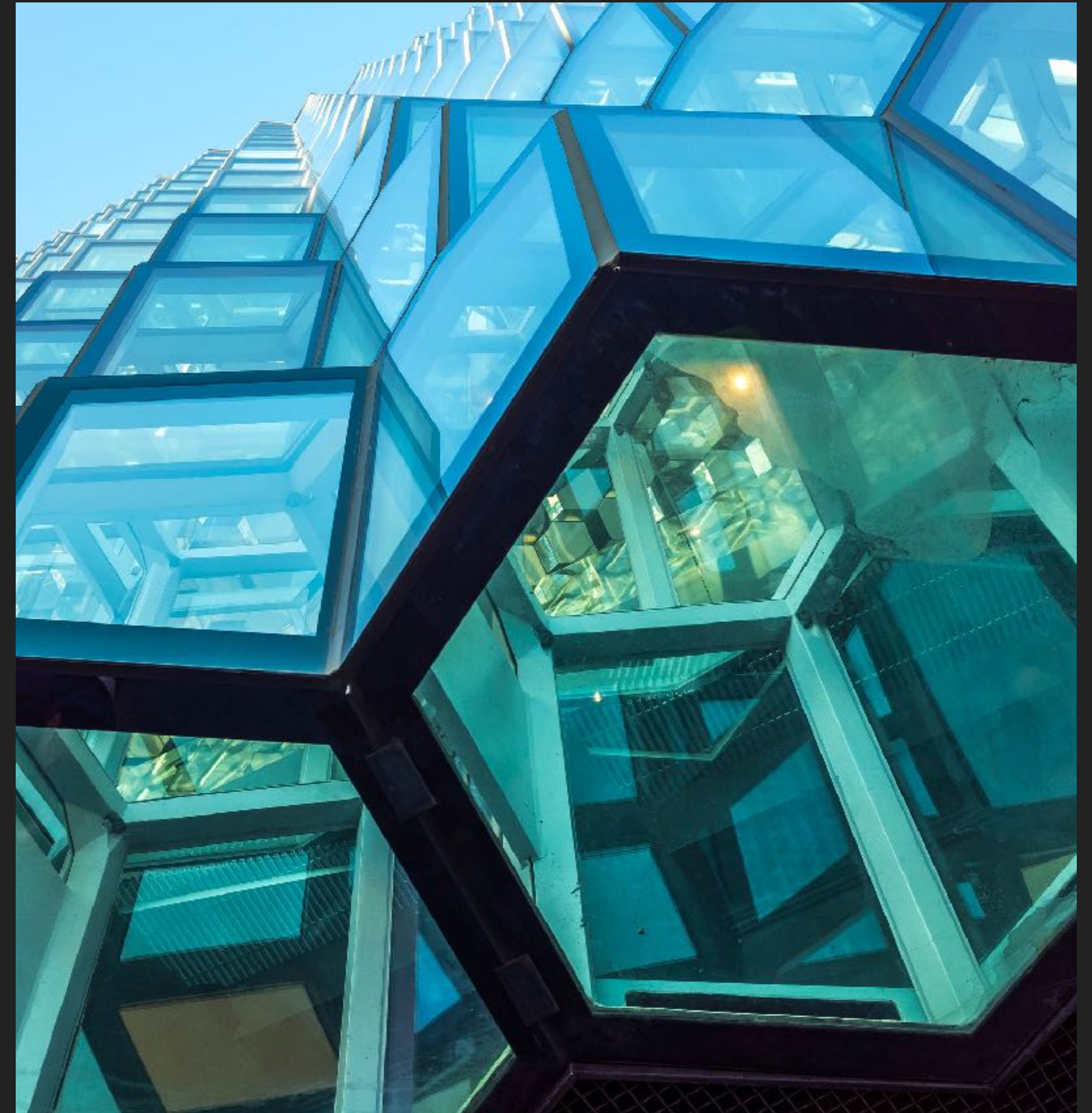
Dave Cheney, Golang UK 2016 keynote



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.



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



GROUP BY MODULE

GROUP BY MODULE

.



GROUP BY CONTEXT





IMPLEMENTING DOMAIN-DRIVEN DESIGN

VAUGHN VERNON

FOREWORD BY ERIC EVANS

DOMAIN DRIVEN DEVELOPMENT! (DDD)

DOMAIN DRIVEN DEVELOPMENT (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

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

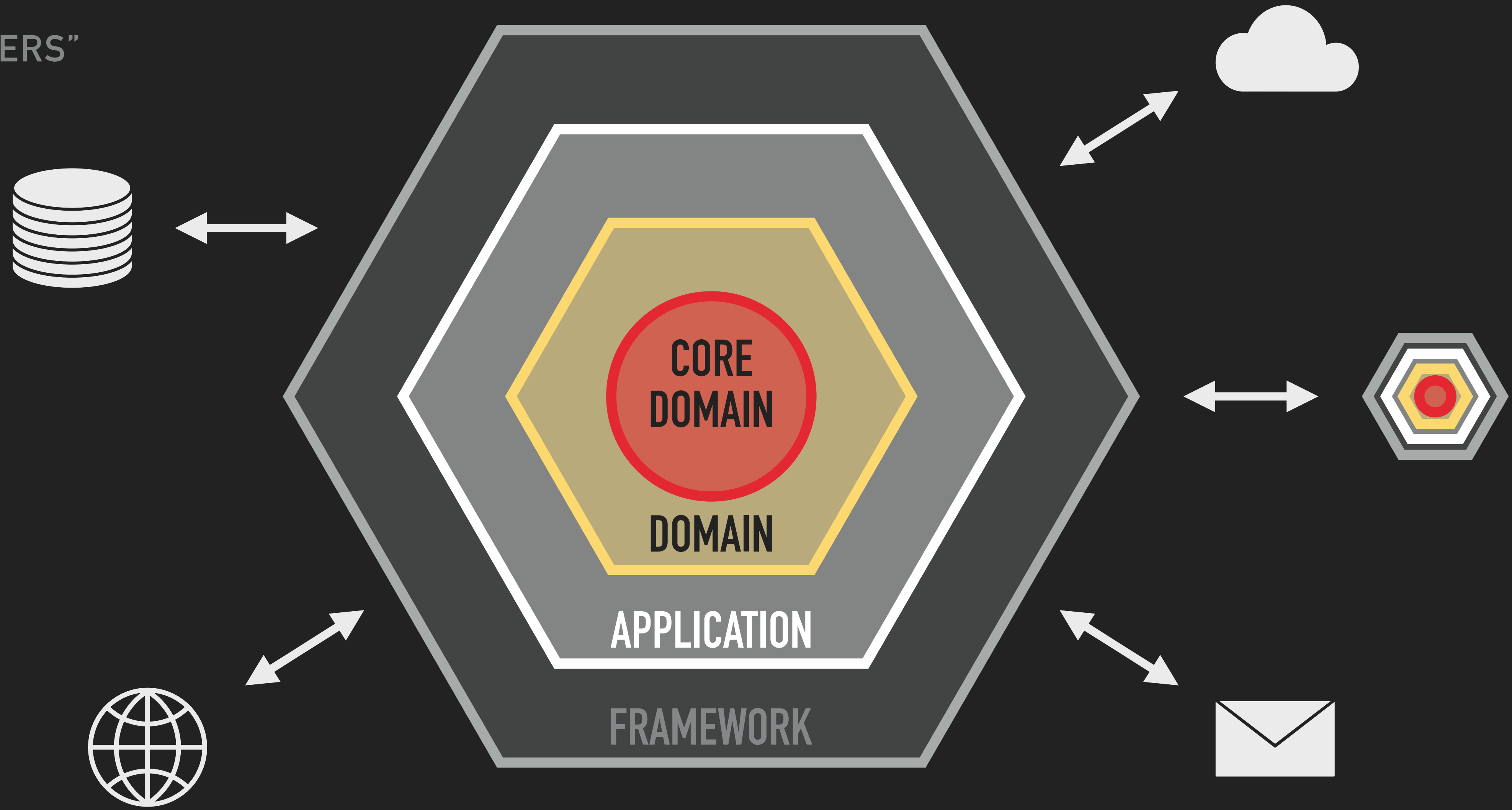
GROUP BY CONTEXT

•

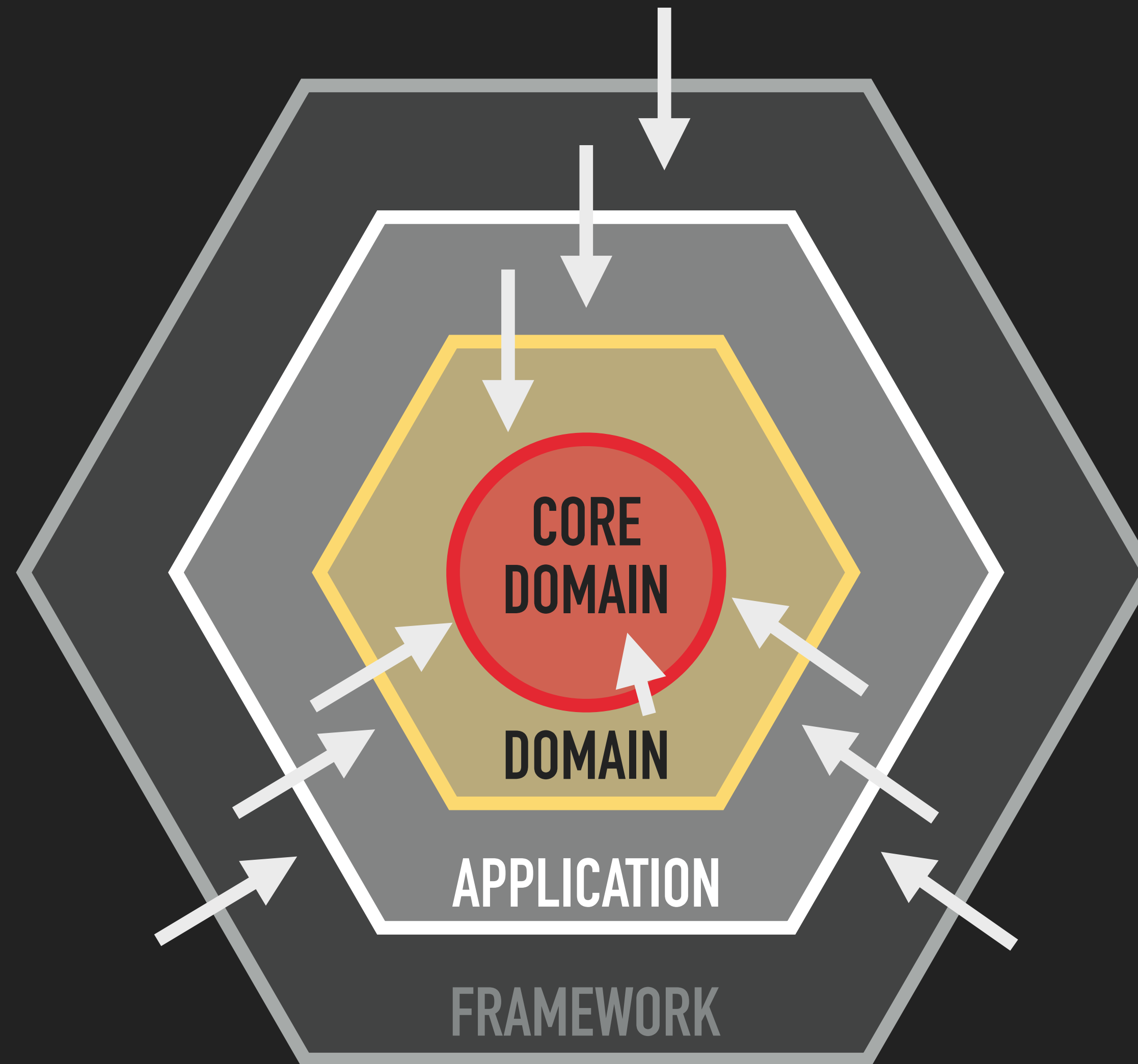


HEXAGONAL ARCHITECTURE

“PORTS AND ADAPTERS”



HEXAGONAL ARCHITECTURE

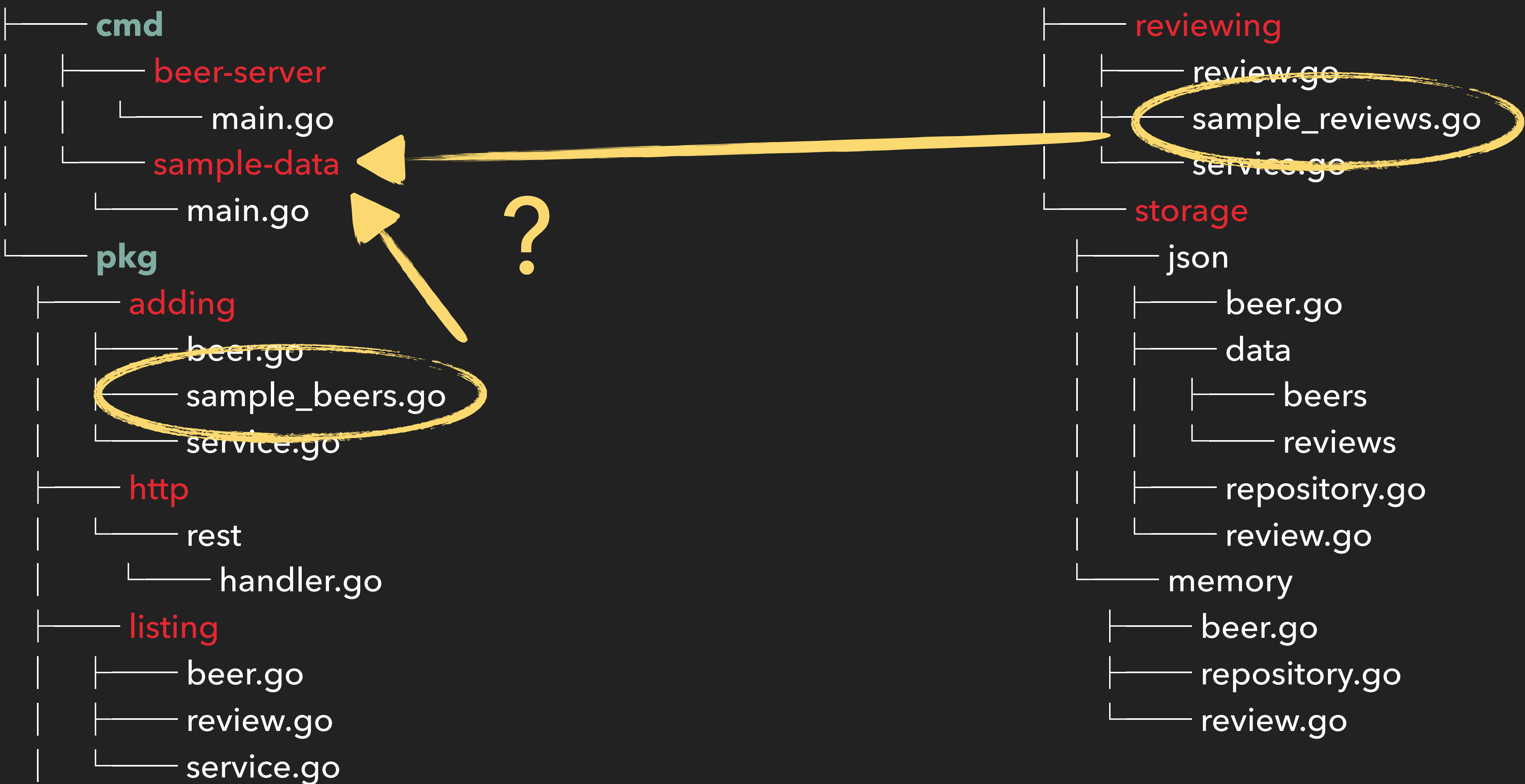


Dependencies
only point inwards.

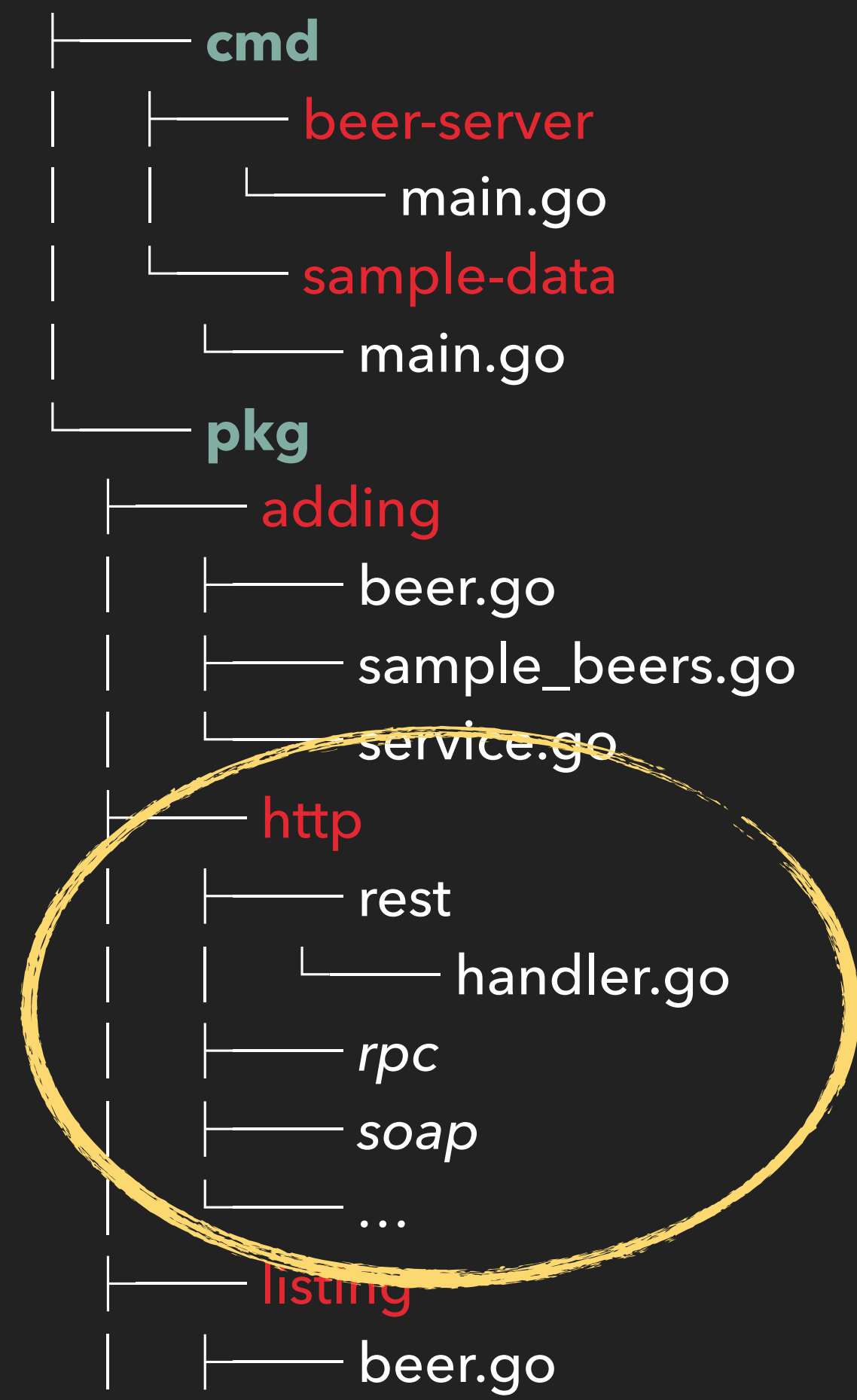
HEXAGONAL ARCHITECTURE



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).



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

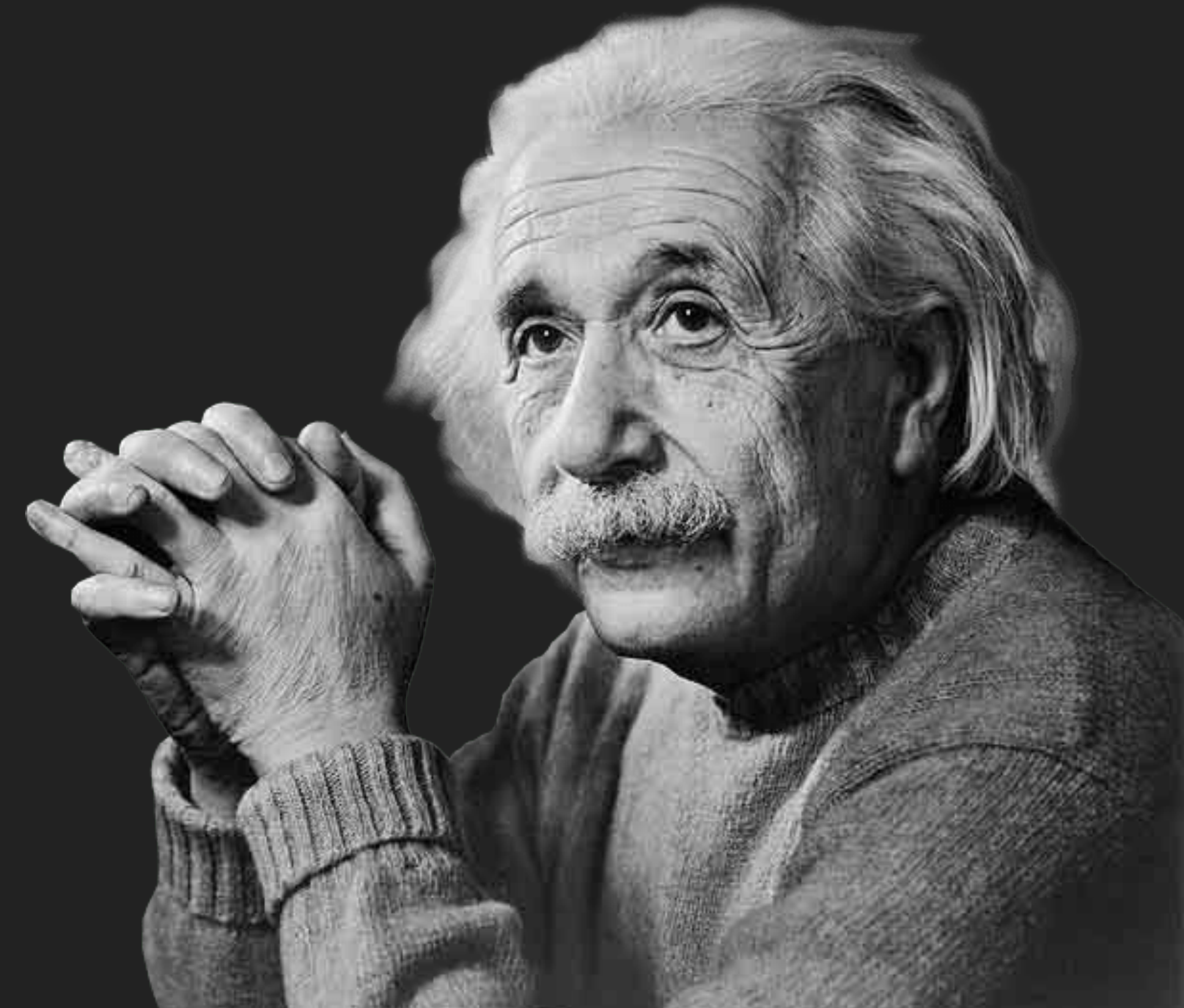


PUTTING IT ALL TOGETHER IN GO

- 👉 Two top-level directories:
cmd (for your binaries) and pkg (for your packages).
- ✌️ Group by context, not generic functionality.
- 🤘 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

THANK YOU!

