Whoops! Where did my architecture go?

Approaches to architecture management for Java and Spring applications

Oliver Gierke

Oliver Gierke

SpringSource Engineer

Spring Data

ogierke@vmware.com

olivergierke

www.olivergierke.de

Background

5 years of consulting

Lots of code reviews

Eoin Woods‘ talk on InfoQ

"If you think architecture is expensive, try no architecture.

Macro VS. Micro Architecture

Macro VS. Micro Architecture

Sample Code

http://github.com/olivergierke/whoops-architecture

Roadmap

Divide and conquer

Of layers and slices

A plain Java based approach

Architecture 101

Know your dependencies

Explicit / Visible dependencies

Granularity

Modules

Layers

Vertical slices

Subsystems

Granularity

Java ARchive

Package

Class

Divide and conquer

Component

Component

Single unit to understand

Component

Single unit to change

Component

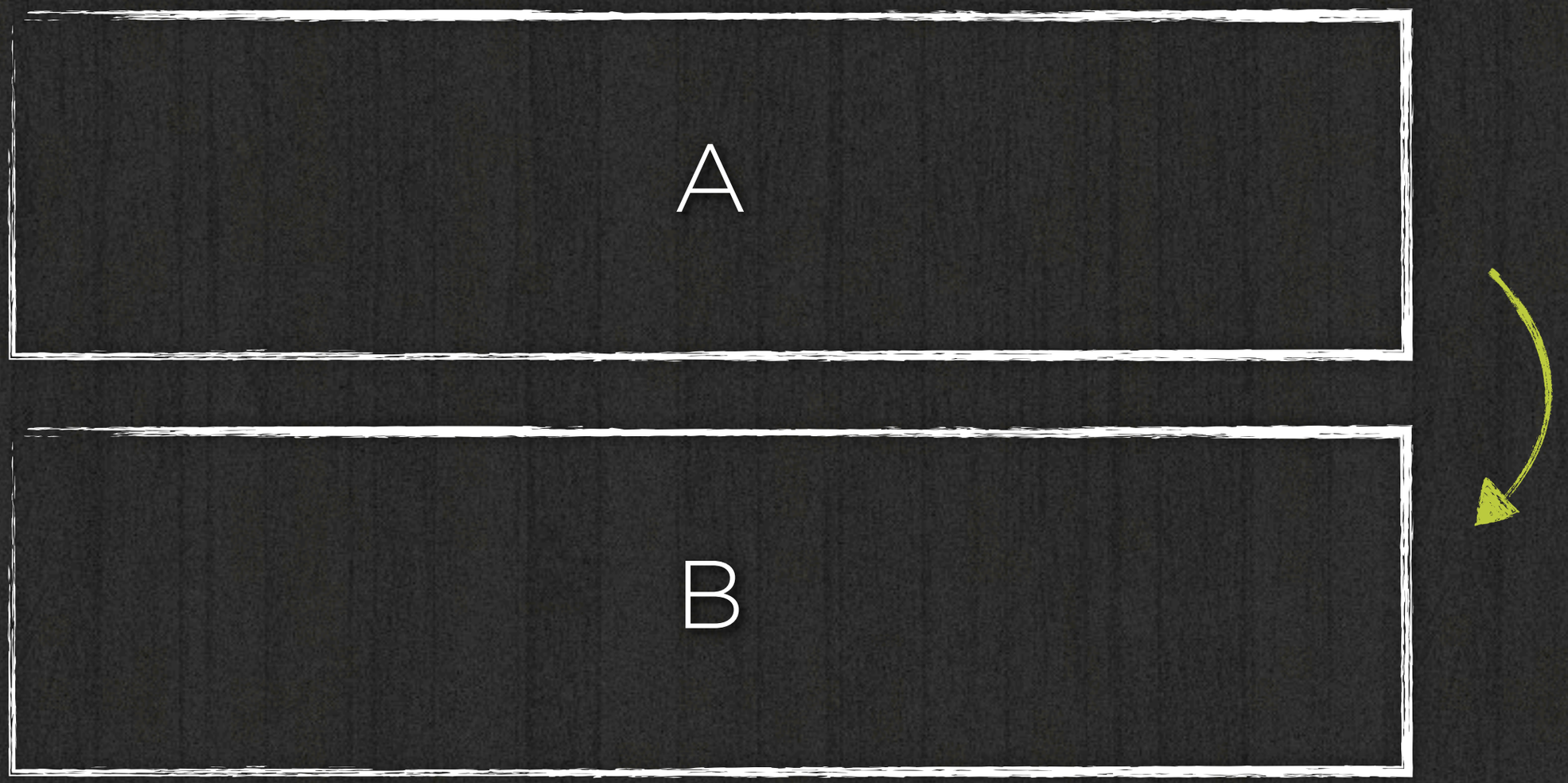
Scope of risk of change

A

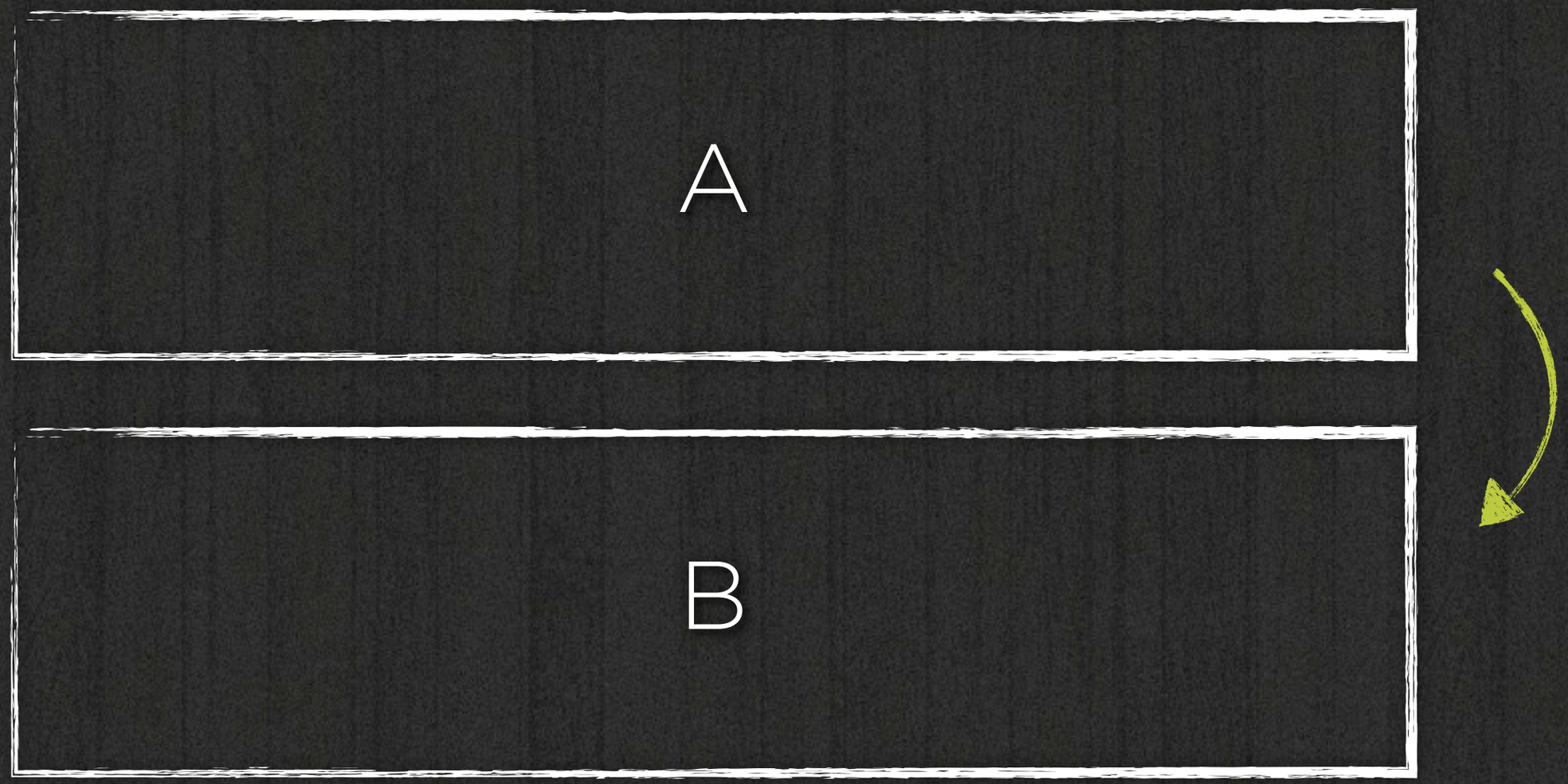
B

A

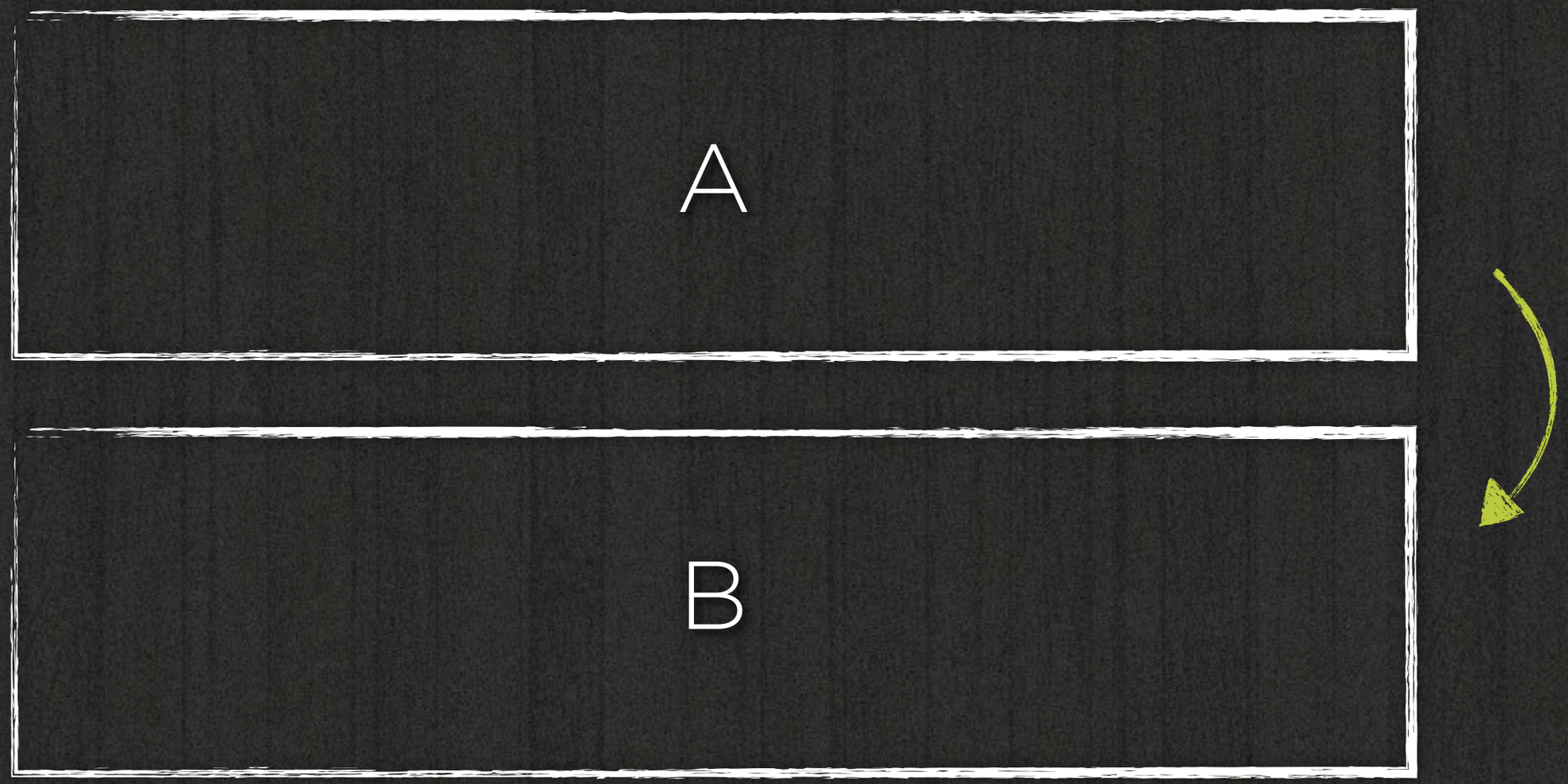
B



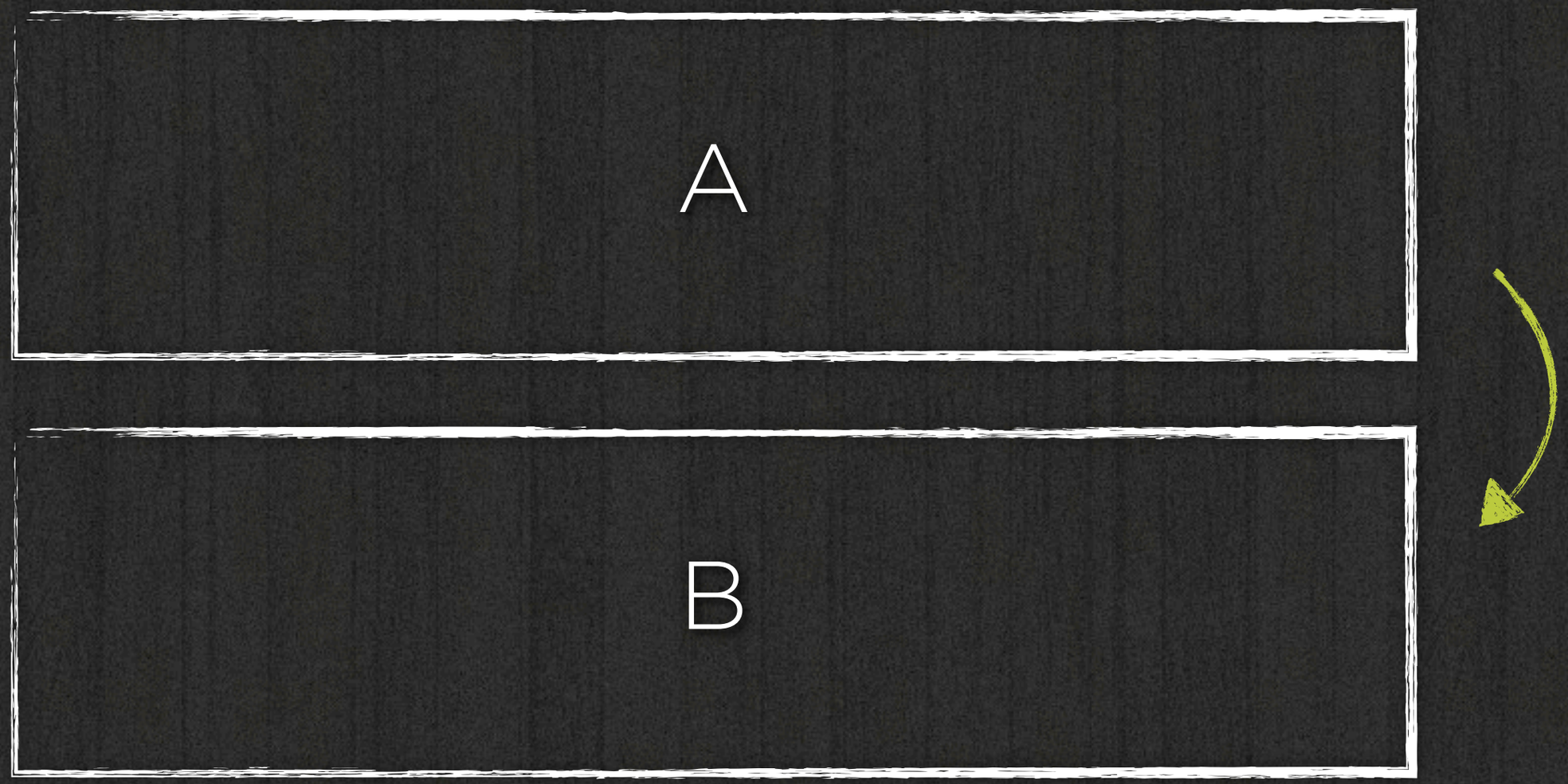
Cost of separation



Definition and maintenance of dependencies

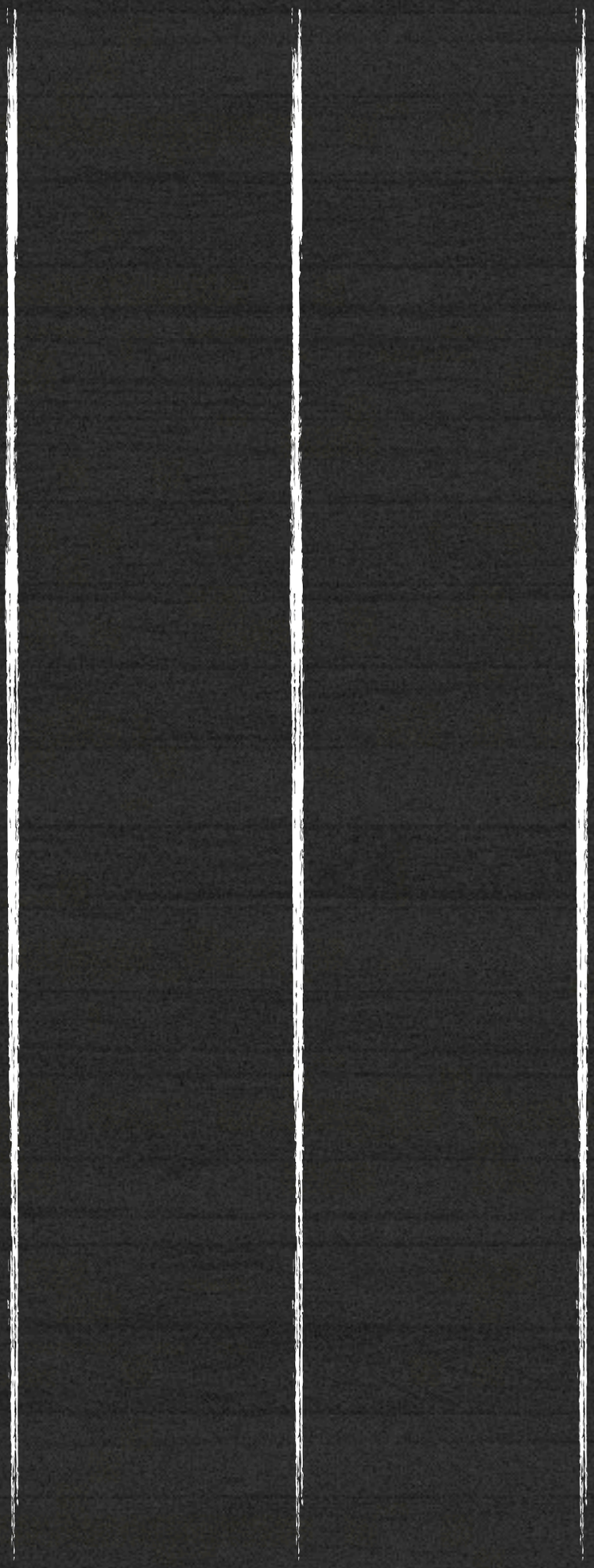


Smaller unit to understand



Reduced risk of change

Of layers and slices...



Presentation

Service

Data Access

Presentation

Service

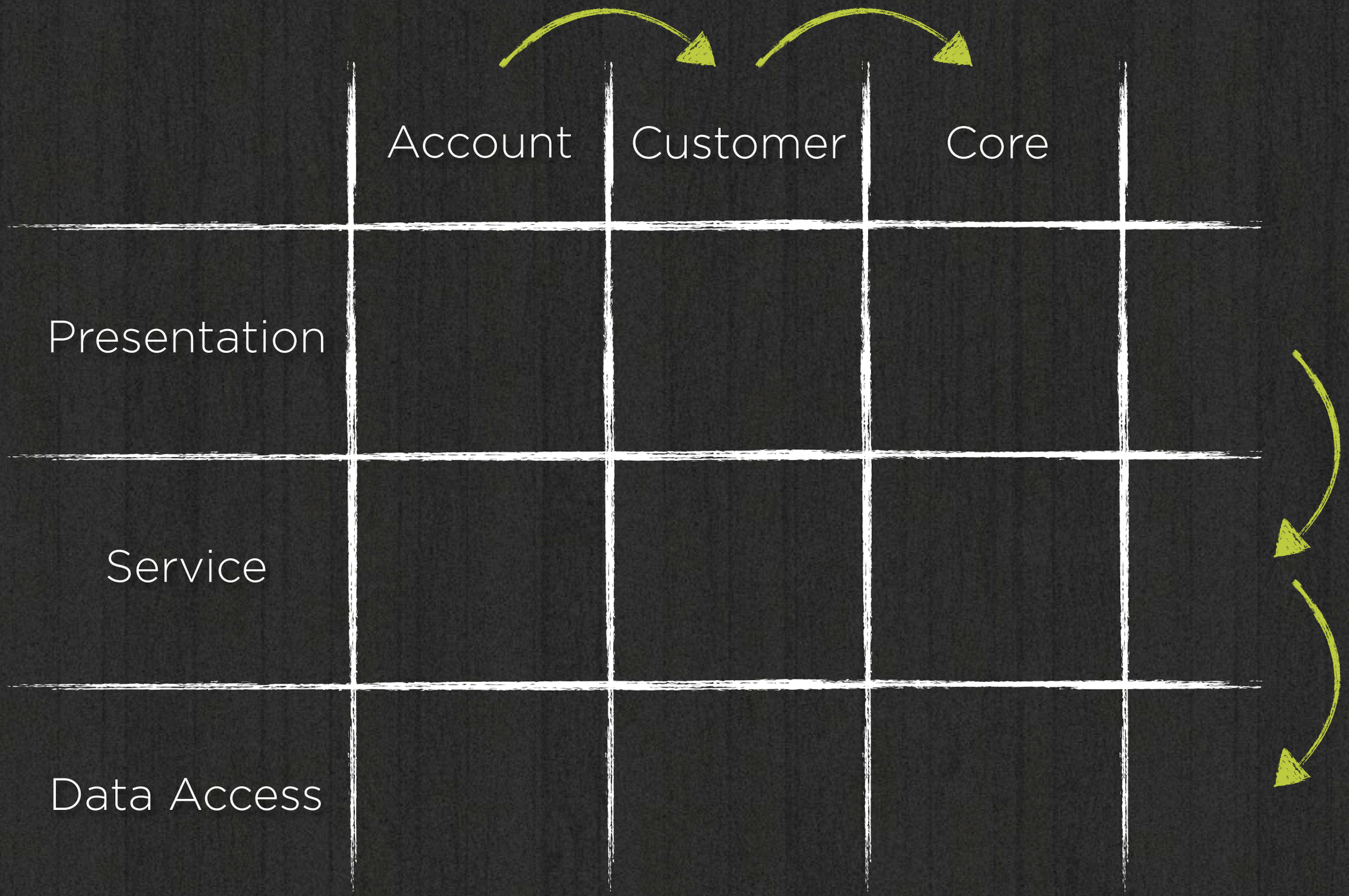
Data Access

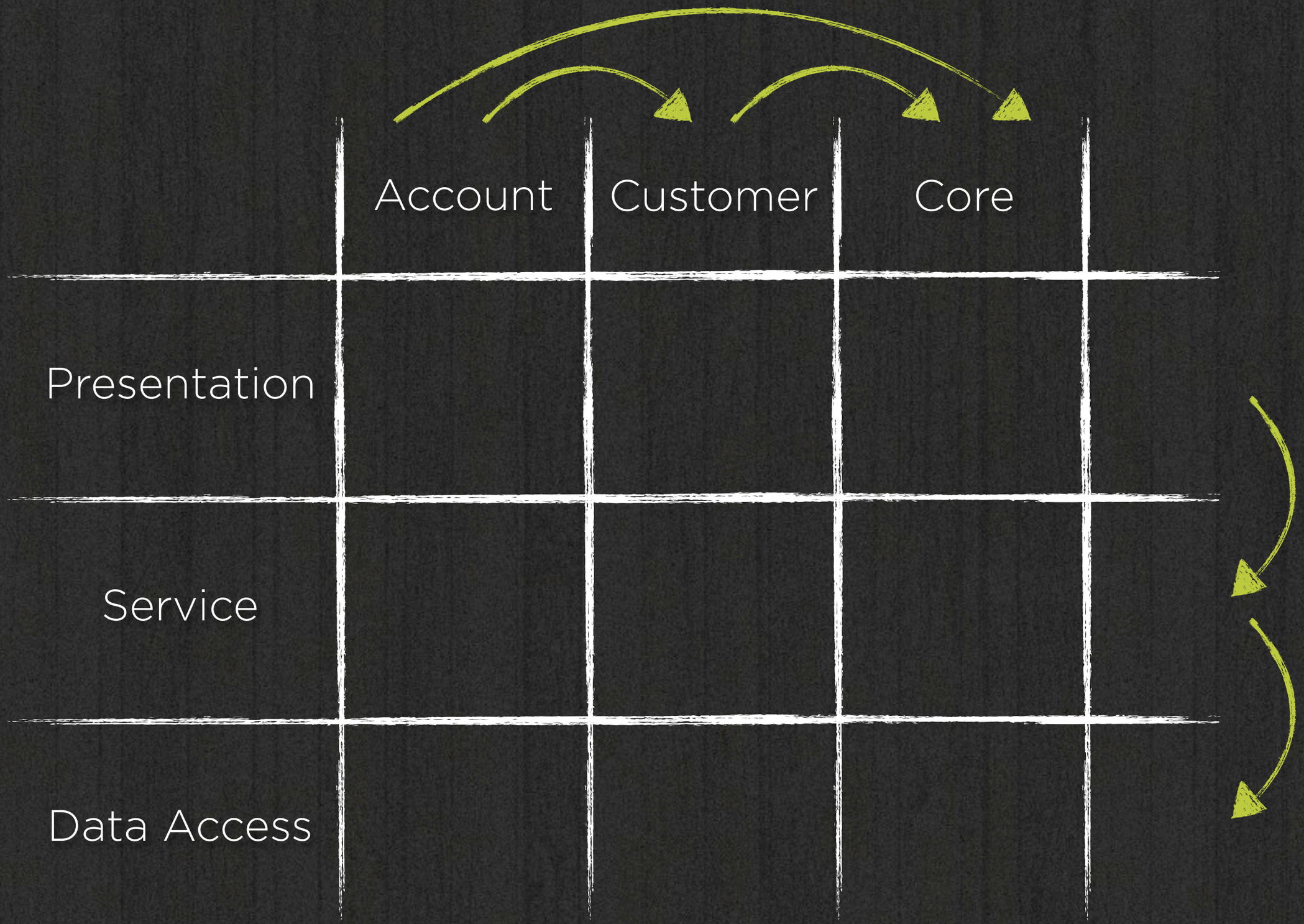
Presentation

Service

Data Access

	Account	Customer	Core
Presentation			
Service			
Data Access			


Account
Customer
Core
Presentation
Service
Data Access


Account
Customer
Core
Presentation
Service
Data Access

Layers

Well understood

Known to developers

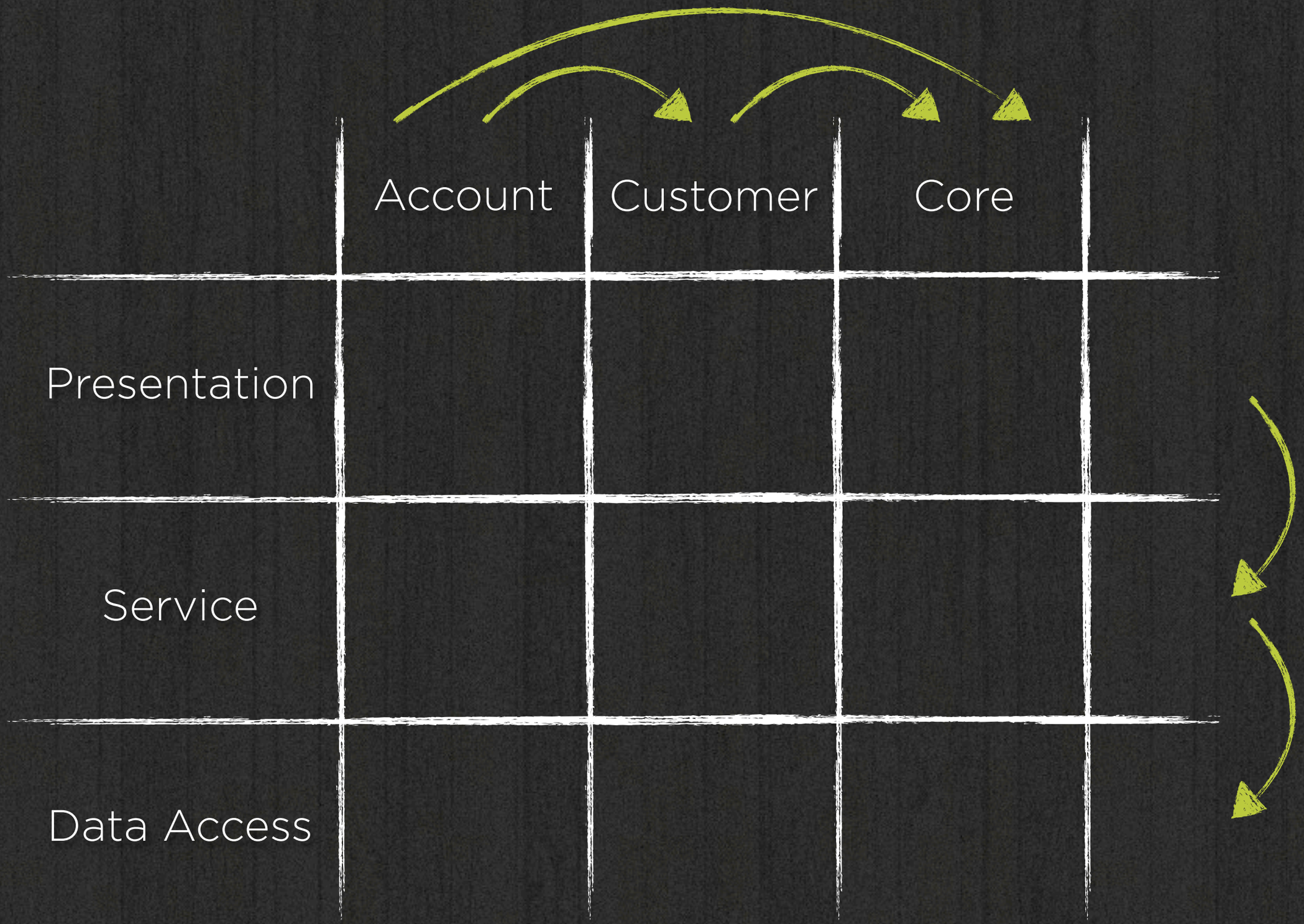
Less important to business

Slices

Hardly understood

New to developers

Key for business req


Account
Customer
Core
Presentation
Service
Data Access

"How to implement an architecture inside a codebase?

Architecture vs. Codebase

"How to implement an architecture inside a codebase?

"How to ~~implement~~ an architecture inside a codebase?

"How to maintain an architecture inside a codebase?

Code analysis

JDepend

Sonar

Demo

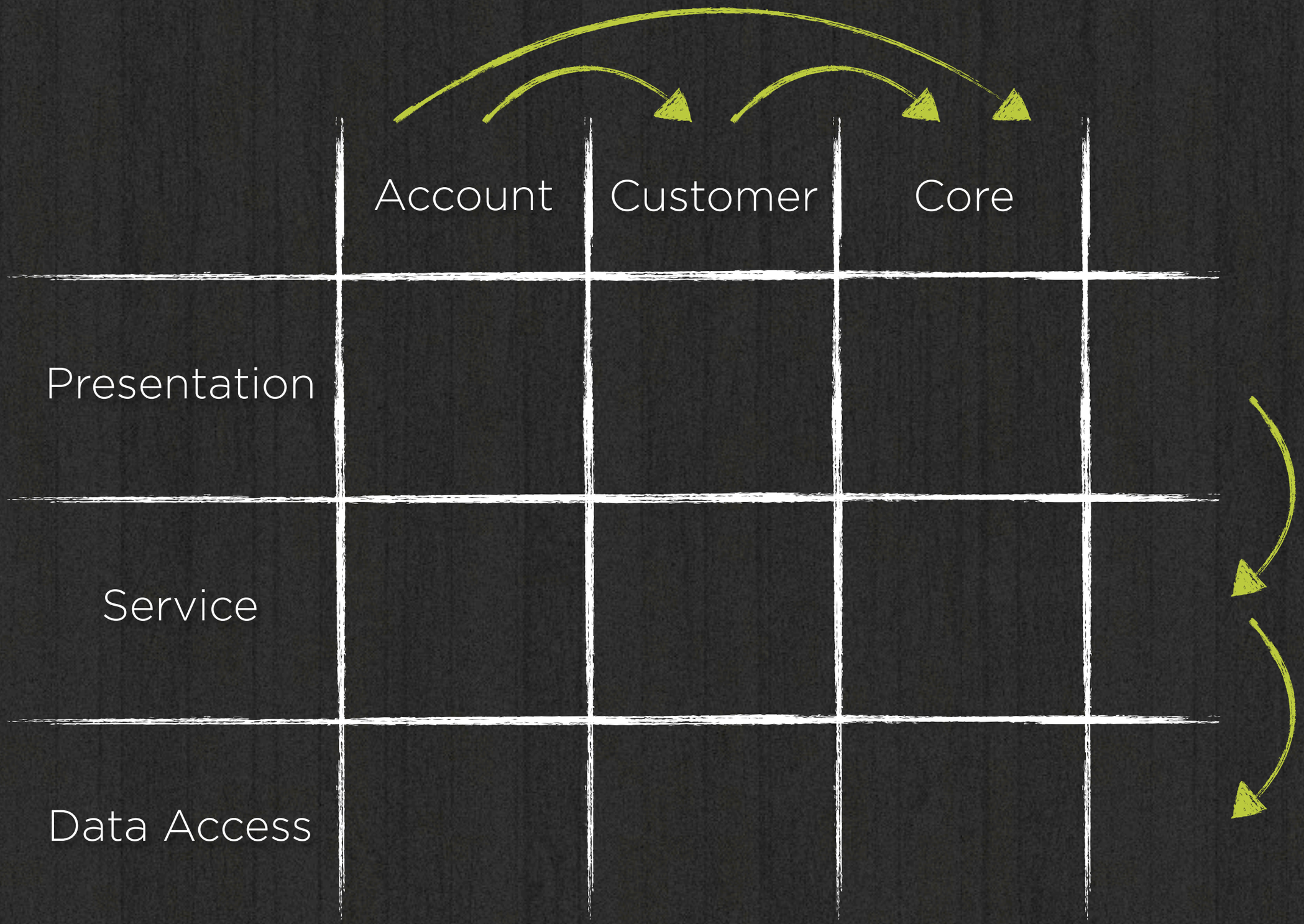
Sonargraph

Formerly known as SonarJ

Demo

A plain Java based approach

"How far can we get with **plain Java means** only?


Account
Customer
Core
Presentation
Service
Data Access

Packages

….layer.slice

….slice.layer

….slice

….web.core

….service.core

….repository.core

….core.web

….core.service

….core.repository

….core

….customer

….account

"Why the f#$k should I even care?

"Does it make a difference?

Dependency management

"You only need to manage, what you can refer to...

Layers first

Leaks slice internals

Lower layers visible to everyone

Slices first/only

Start with package per slice

Expose interfaces and domain types

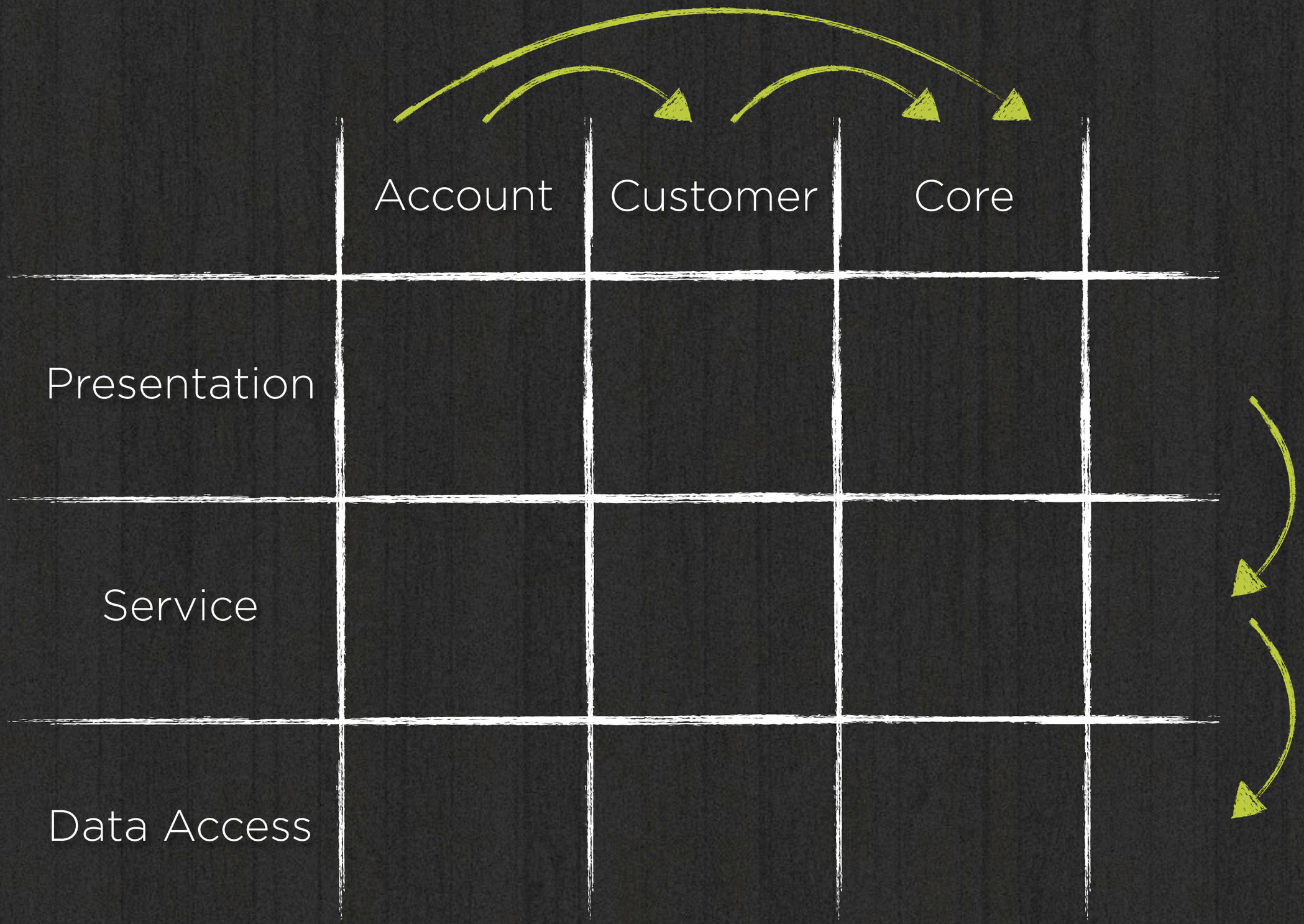
Keep implementations private

Slices first/only

Encapsulates business module

Internals understood anyway

"Start with **less** packages and the **least visibility** possible...


Account
Customer
Core
Presentation
Service
Data Access

Account

Customer

Core

Presentation

Service

Data Access

Demo

Take-aways

Know your dependencies

On every granularity

Start as strict as possible

Get lenient where necessary

Resources

Spring Data JPA @ GitHub

Sonargraph

Blogpost

Thanks & credits

Eoin Woods - Talk @ InfoQ

Uwe Friedrichsen - Slides @ Slideshare