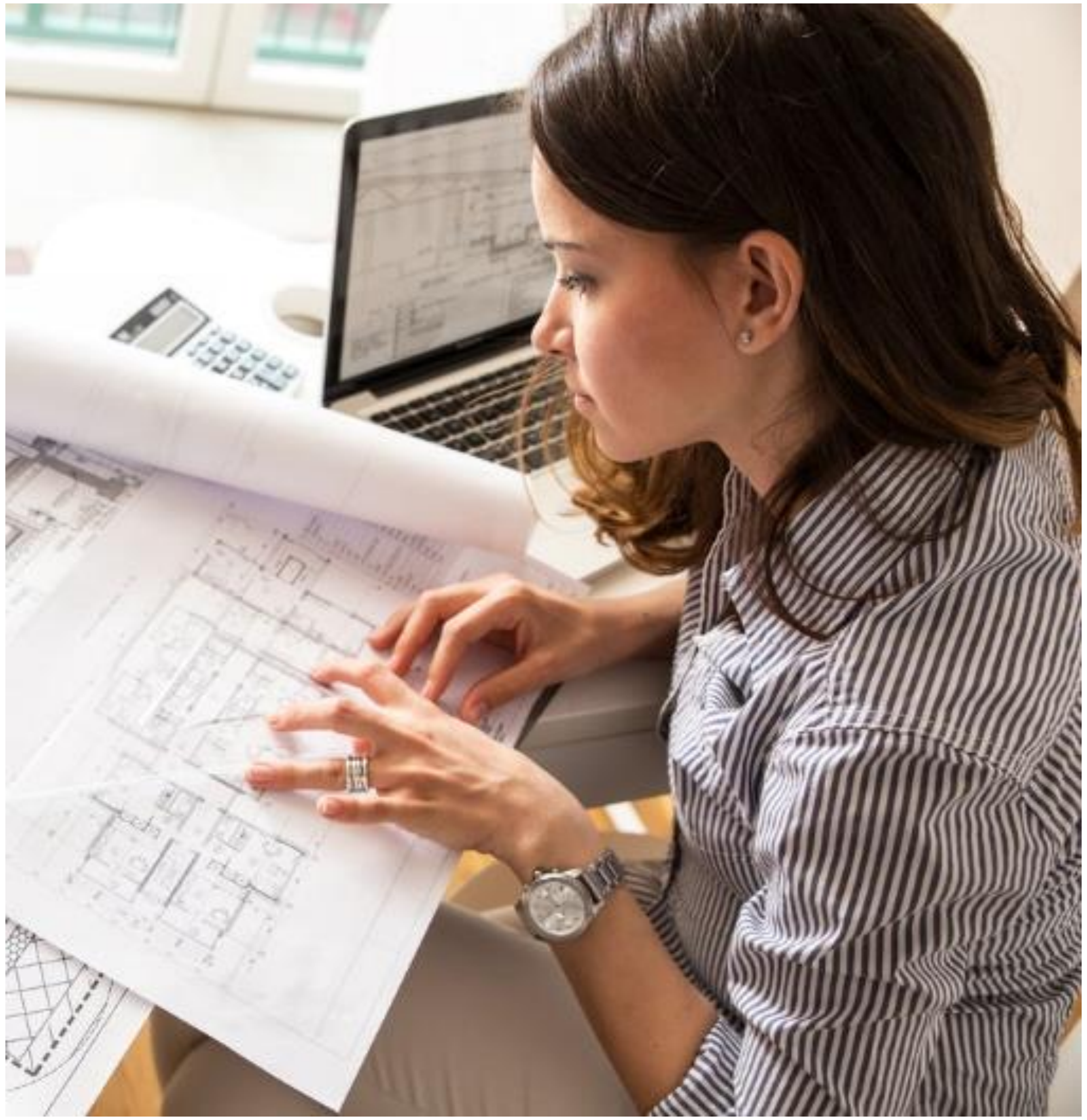
Clean Architecture

Patterns, Practices, and Principles

@matthewrenze

#DevSum17



About Me

Independent consultant

Education

- B.S. in Computer Science (ISU)
- B.A. in Philosophy (ISU)

Community

- Public Speaker
- Pluralsight Author
- Microsoft MVP
- ASPInsider
- Open-Source Software

Overview

1. Clean Architecture
2. Domain-Centric Architecture
3. Application Layer
4. Commands and Queries
5. Functional Organization
6. Microservices

Focus

Enterprise Architecture

Line-of-Business Applications

Modern equivalent of 3-Layer

Focus

Generally applicable

6 Key Points

Q & A

What is Software Architecture?

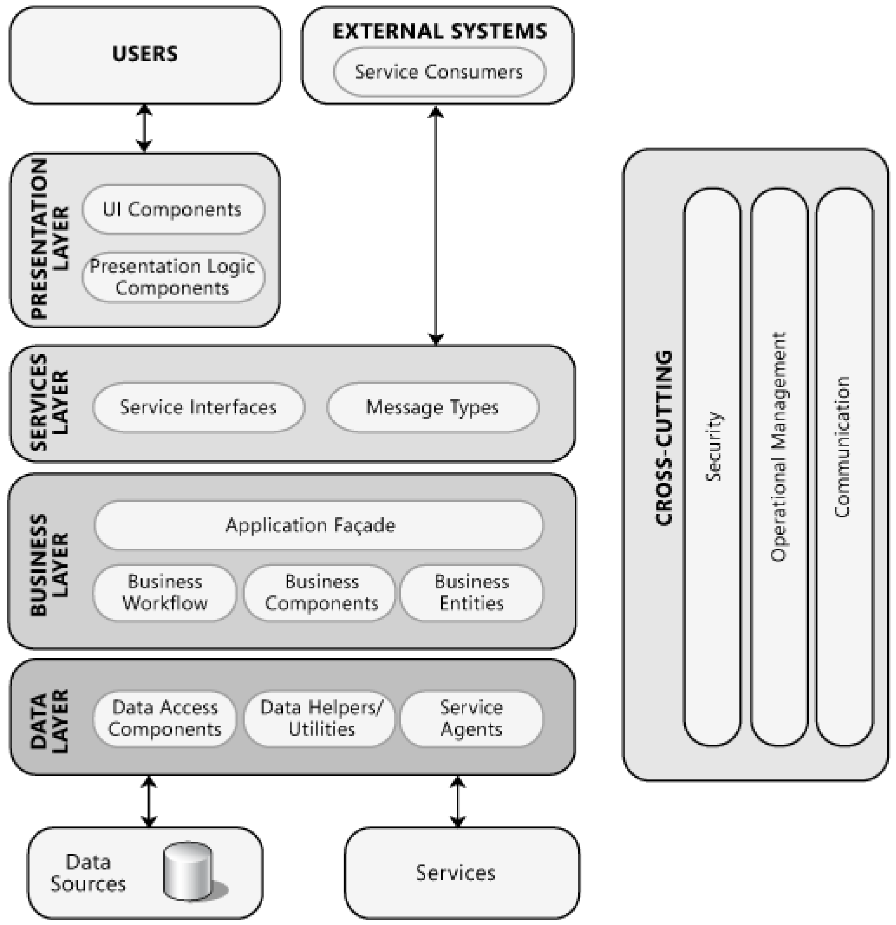
High-level

Structure

Layers

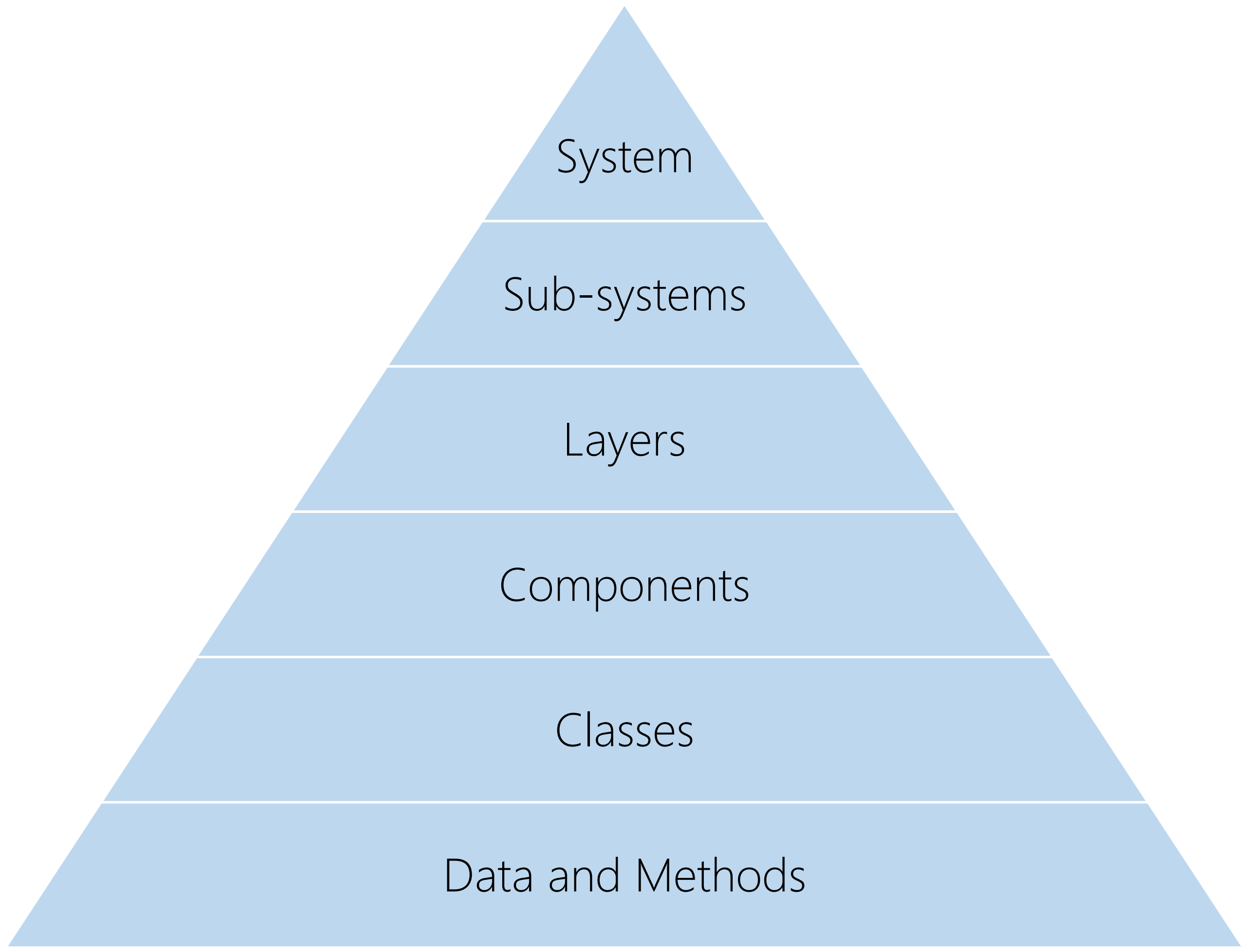
Components

Relationships

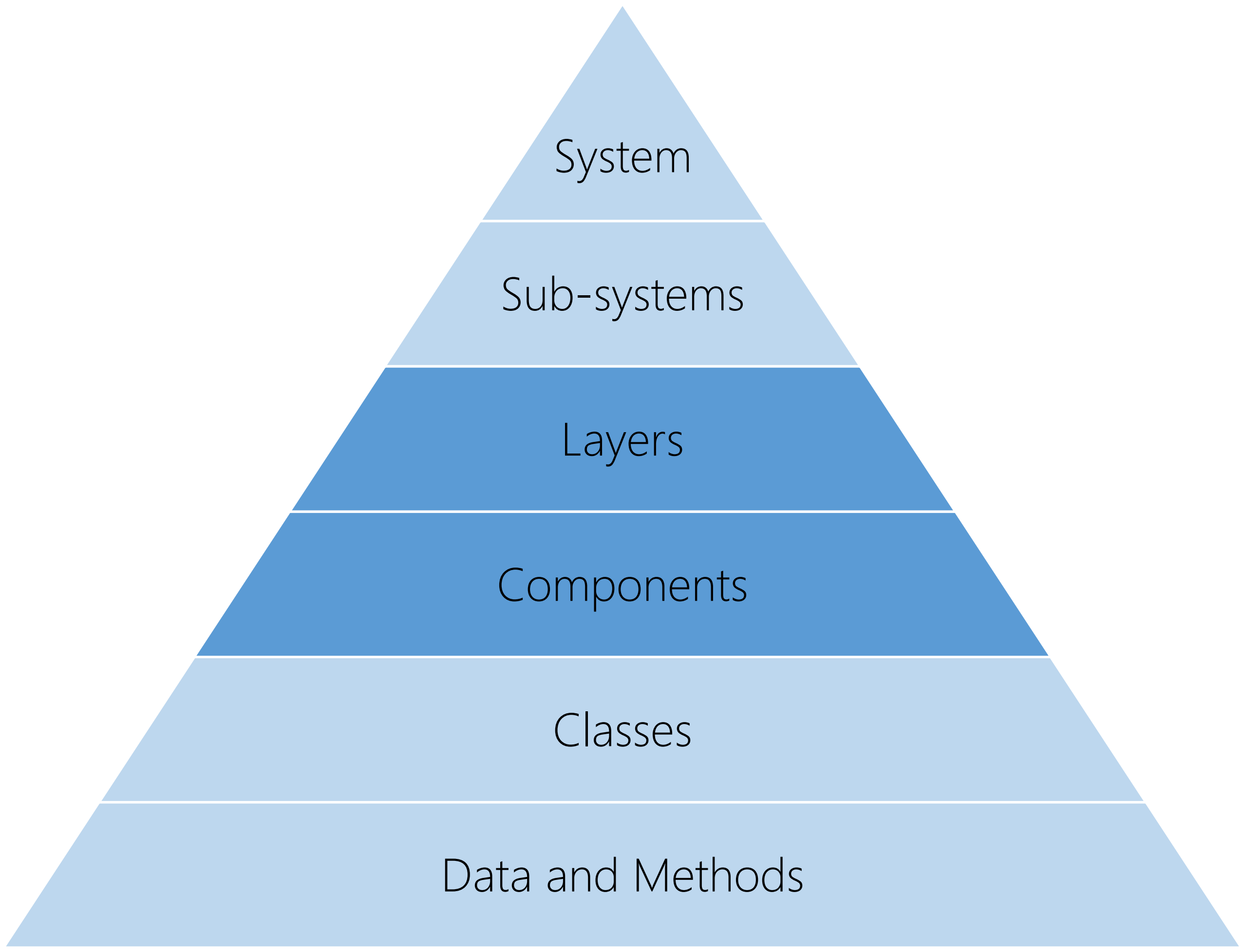


Source: http://msdn.microsoft.com/en-us/library/ff650706.aspx

Levels of Architectural Abstraction



Levels of Architectural Abstraction



Messy vs Clean Architecture

Messy vs Clean Architecture

Messy vs Clean Architecture

What Is Bad Architecture?

Complex

Inconsistent

Incoherent

Rigid

Brittle

Untestable

Unmaintainable

What Is Clean Architecture?

Simple

Understandable

Flexible

Emergent

Testable

Maintainable

What Is Clean Architecture?

Architecture that is designed for the inhabitants of the architecture…
not for the architect… or the machine

What Is Clean Architecture?

Architecture that is designed for the **inhabitants** of the architecture…
not for the architect… or the machine

What Is Clean Architecture?

Architecture that is designed for the inhabitants of the architecture…
not for the **architect**… or the machine

What Is Clean Architecture?

Architecture that is designed for the inhabitants of the architecture…
not for the architect… or the **machine**

What Is Clean Architecture?

Architecture that is designed for the inhabitants of the architecture...
not for the architect... or the machine

Why Is Clean Architecture Important?

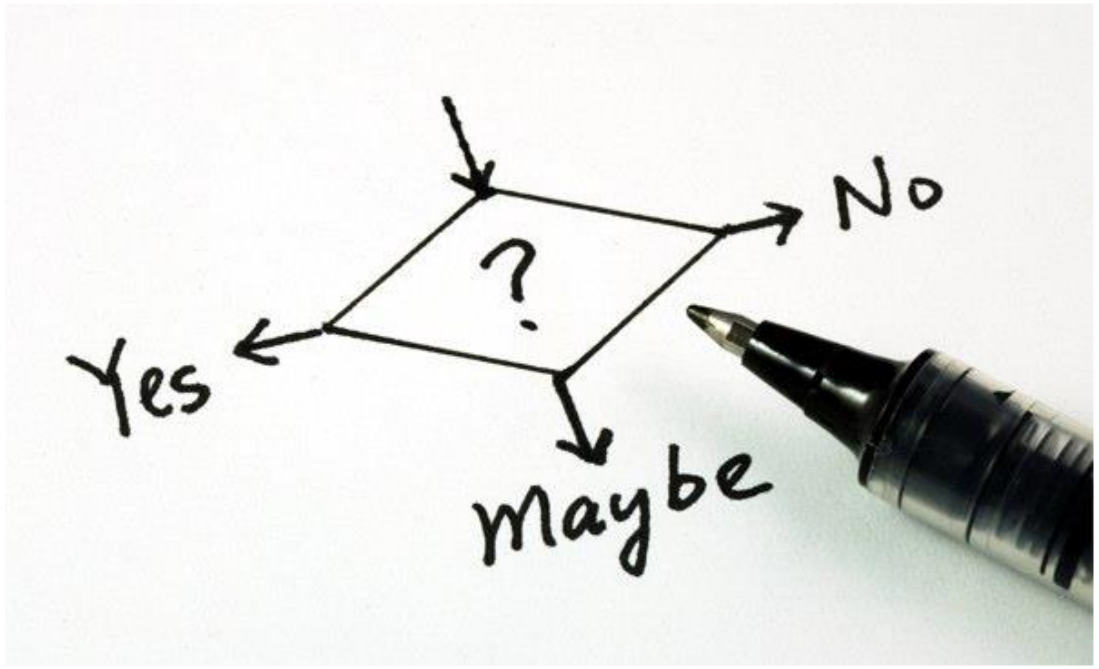
Cost/benefit

Minimize cost to maintain

Maximize business value

Decisions, Decisions, Decisions…

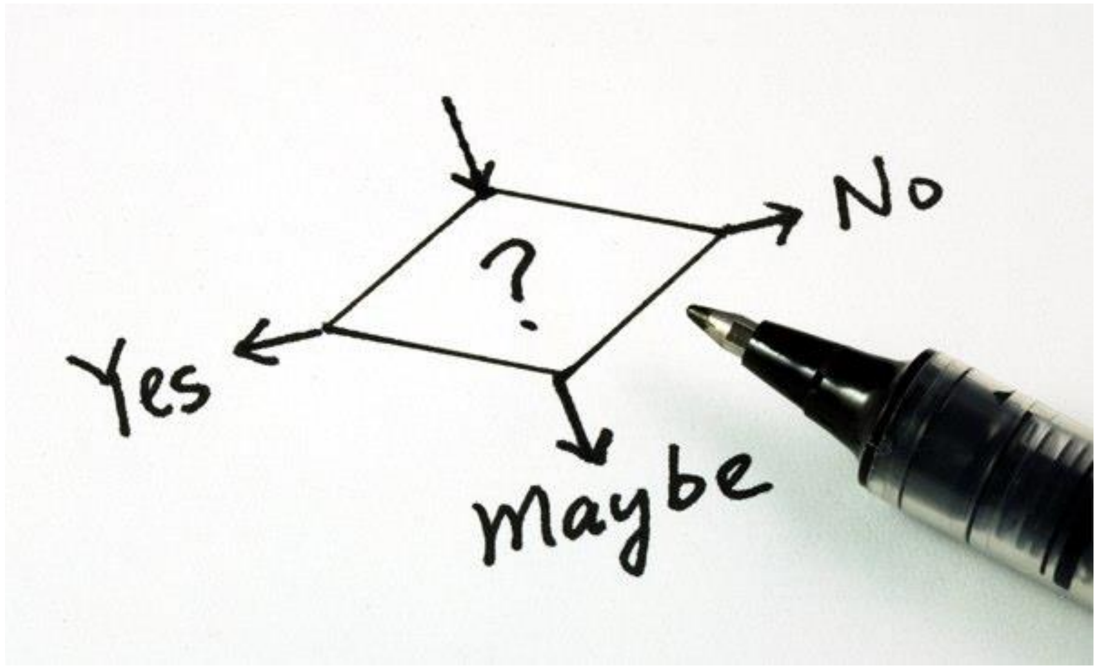
Context is king



Decisions, Decisions, Decisions…

Context is king

All decisions are a tradeoff

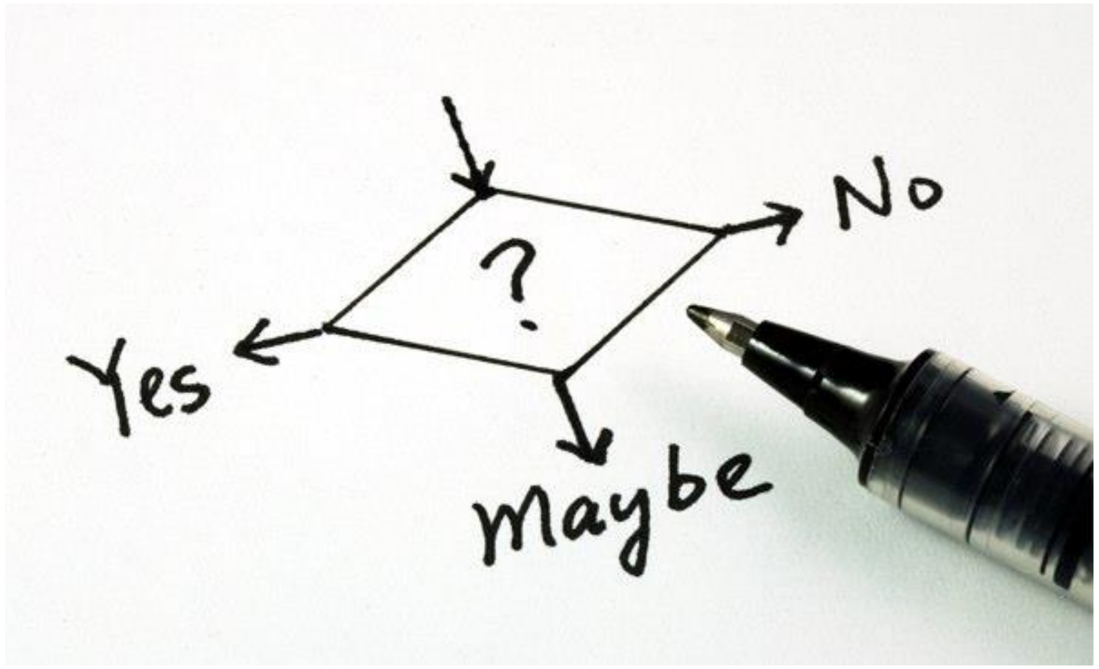


Decisions, Decisions, Decisions…

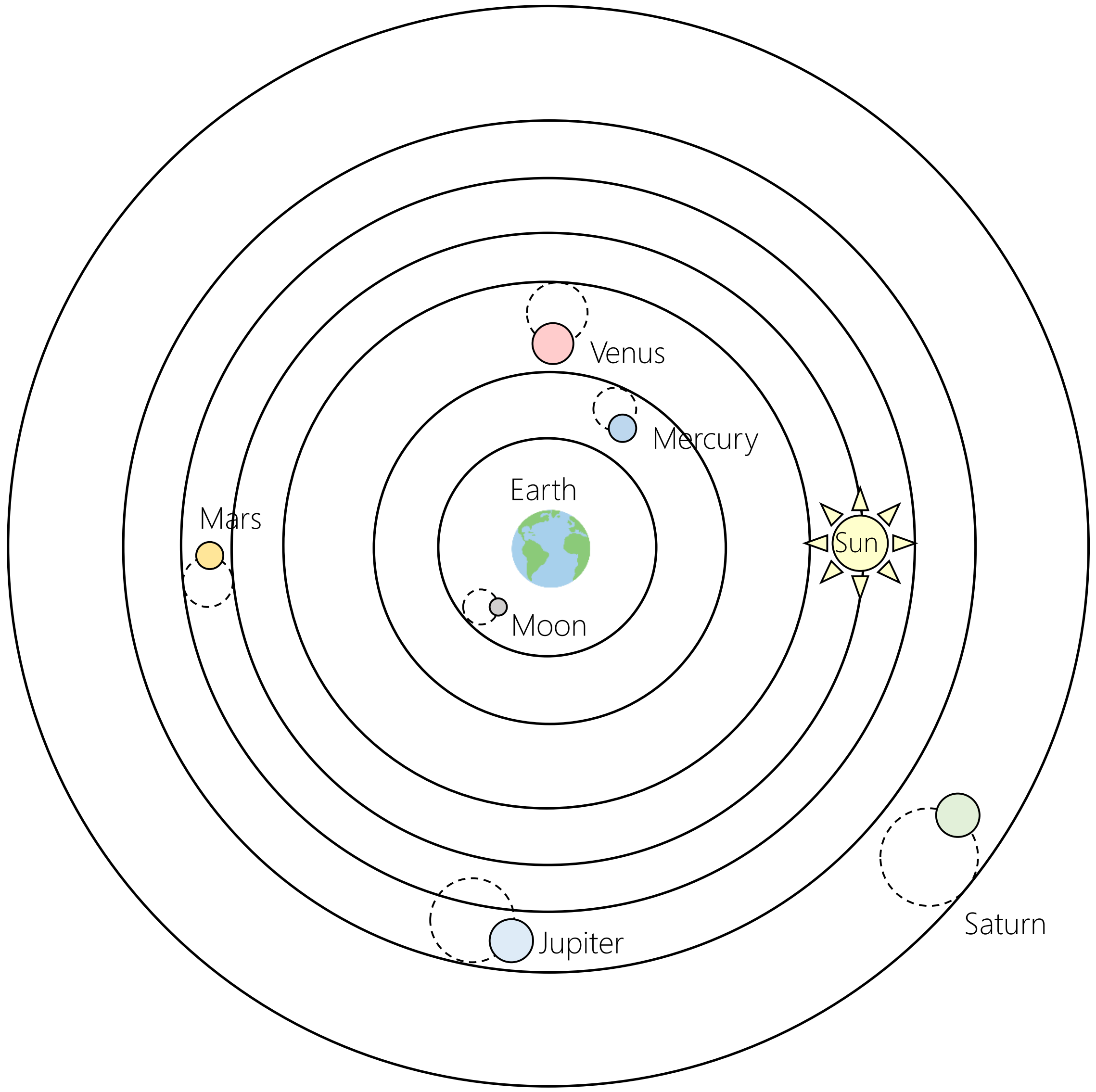
Context is king

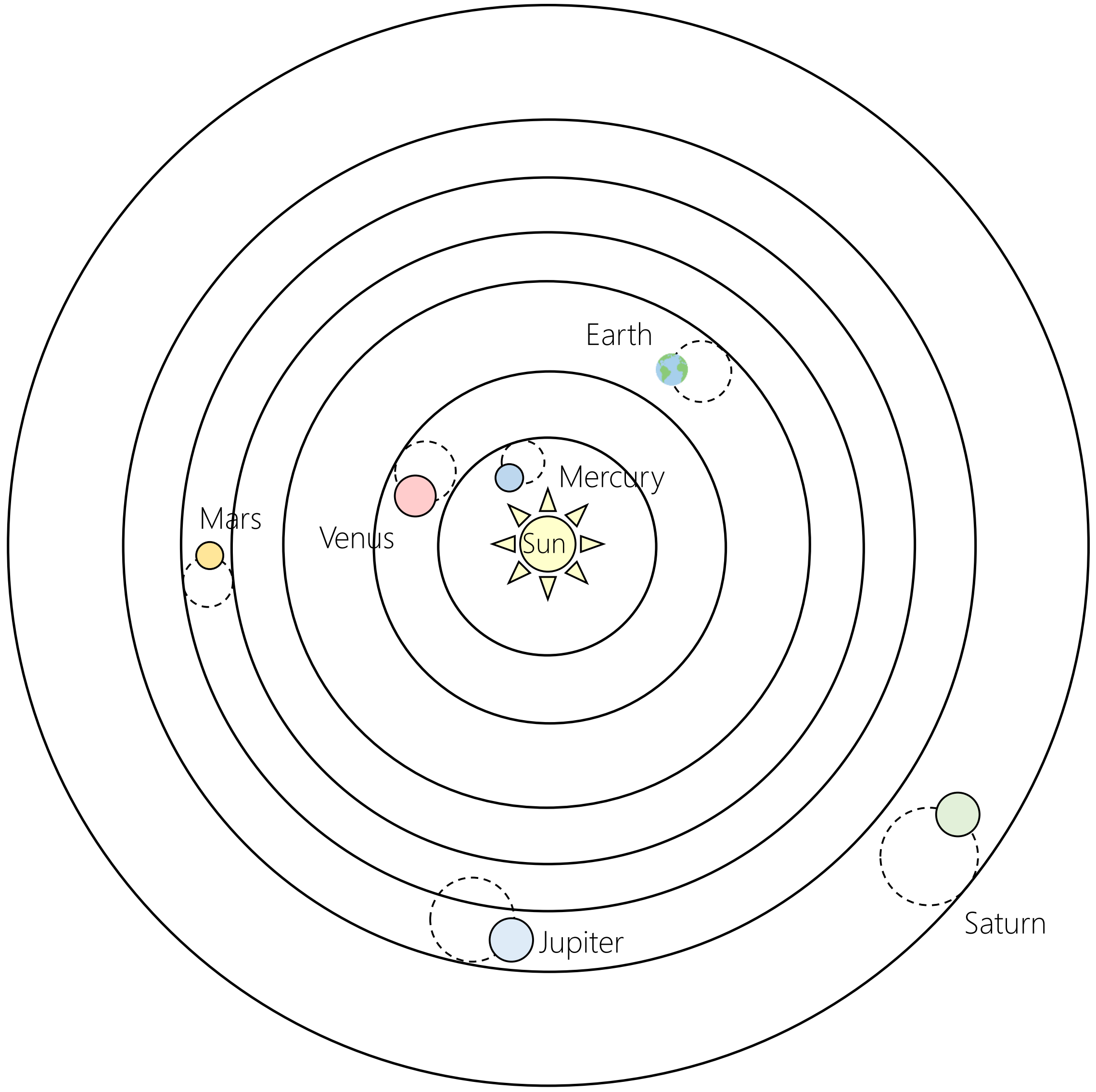
All decisions are a tradeoff

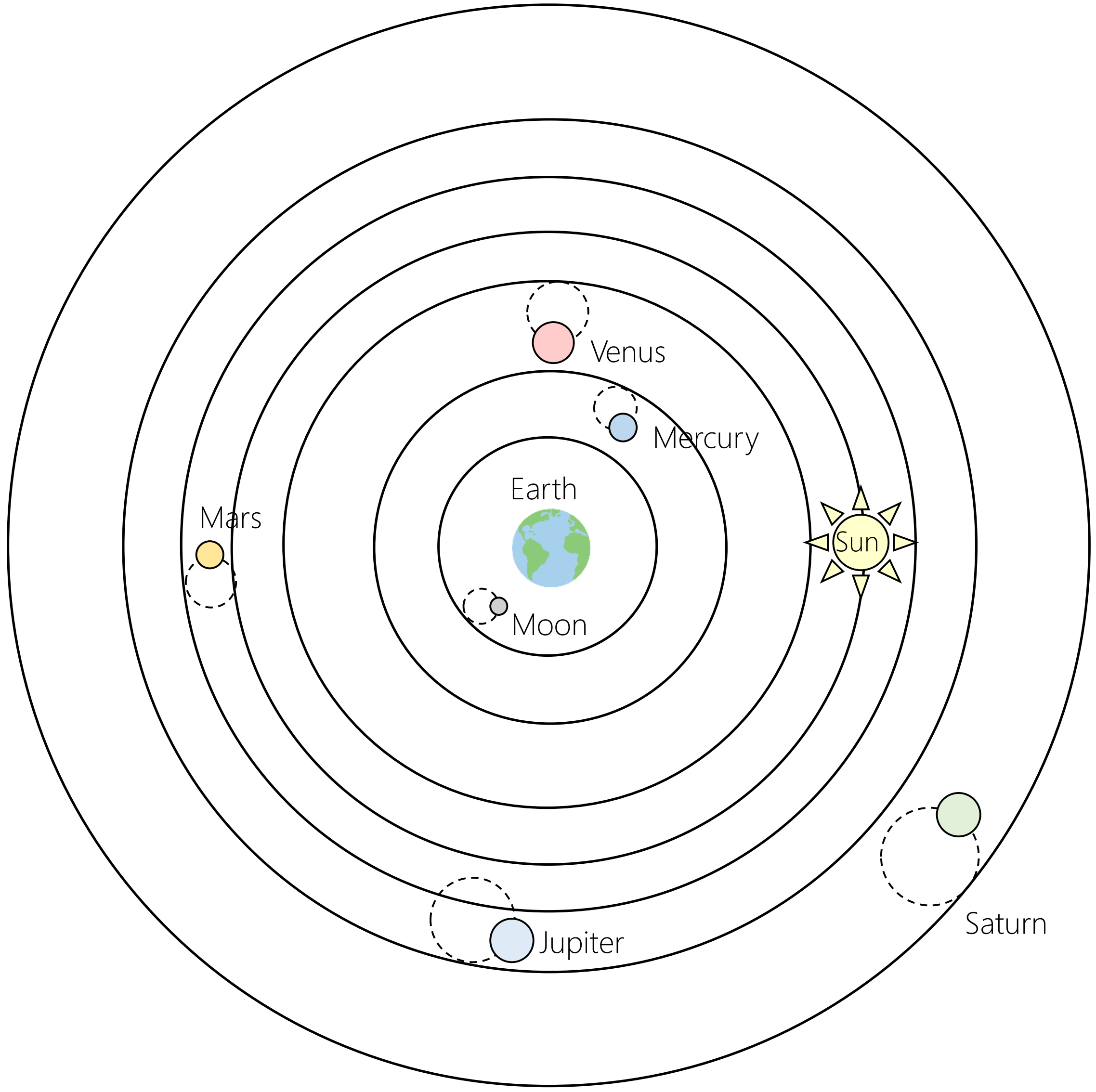
Use your best judgement



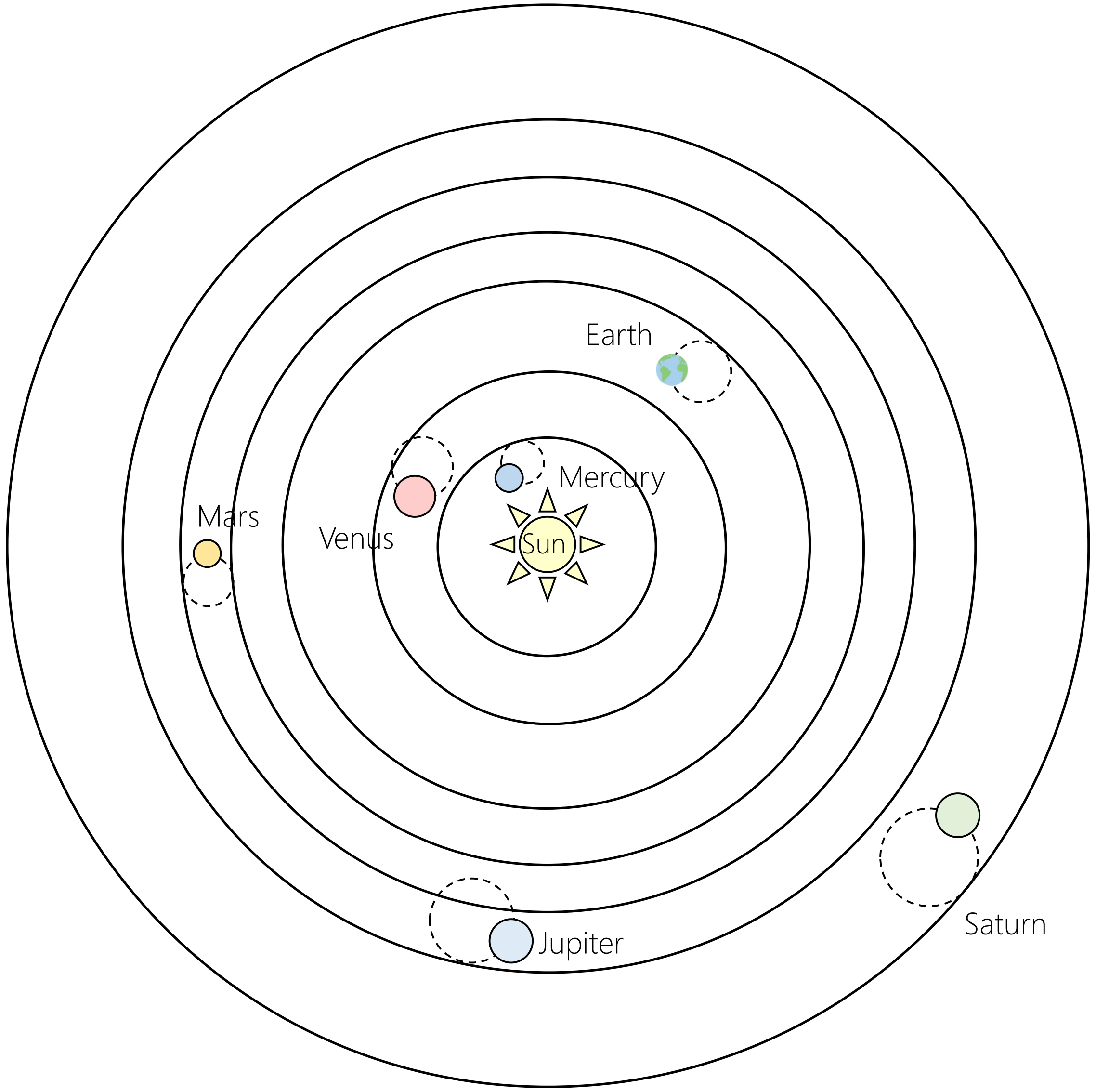
Domain-Centric Architecture


Venus
Mercury
Earth
Mars
Sun
Moon
Jupiter
Saturn


Earth
Mercury
Mars
Venus
Sun
Jupiter
Saturn

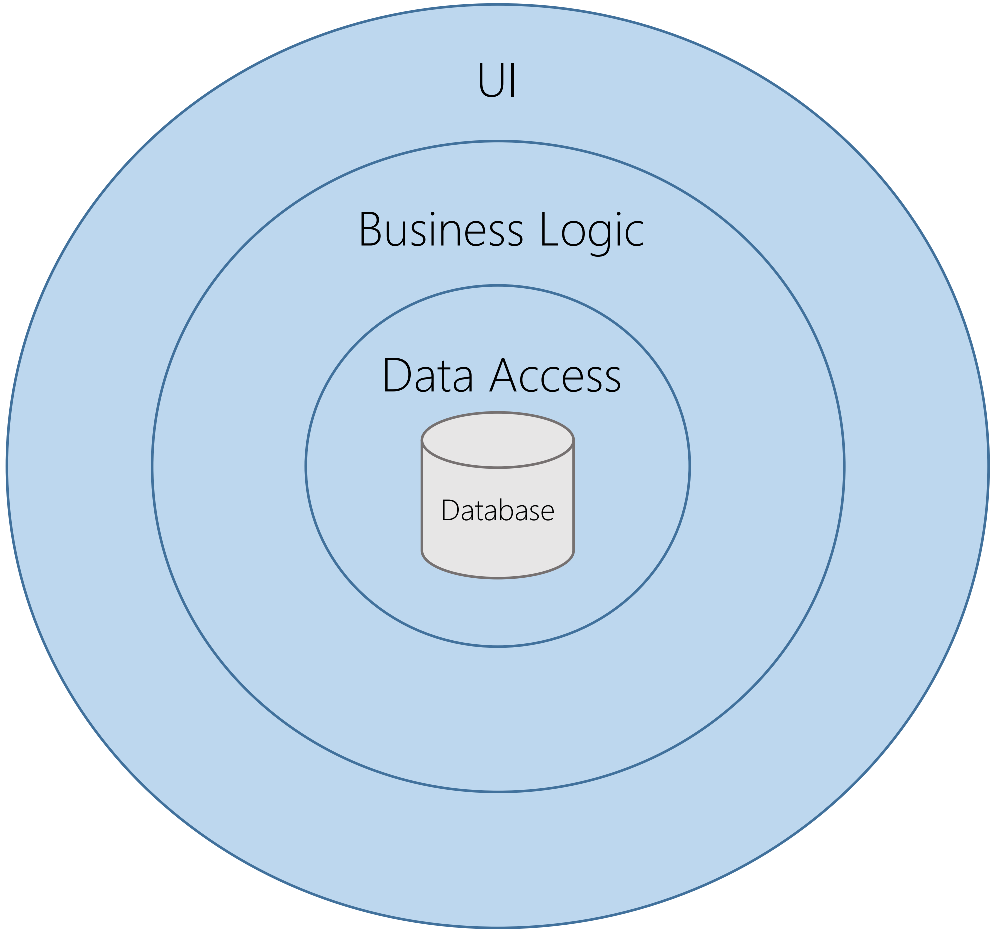


Geocentric Model

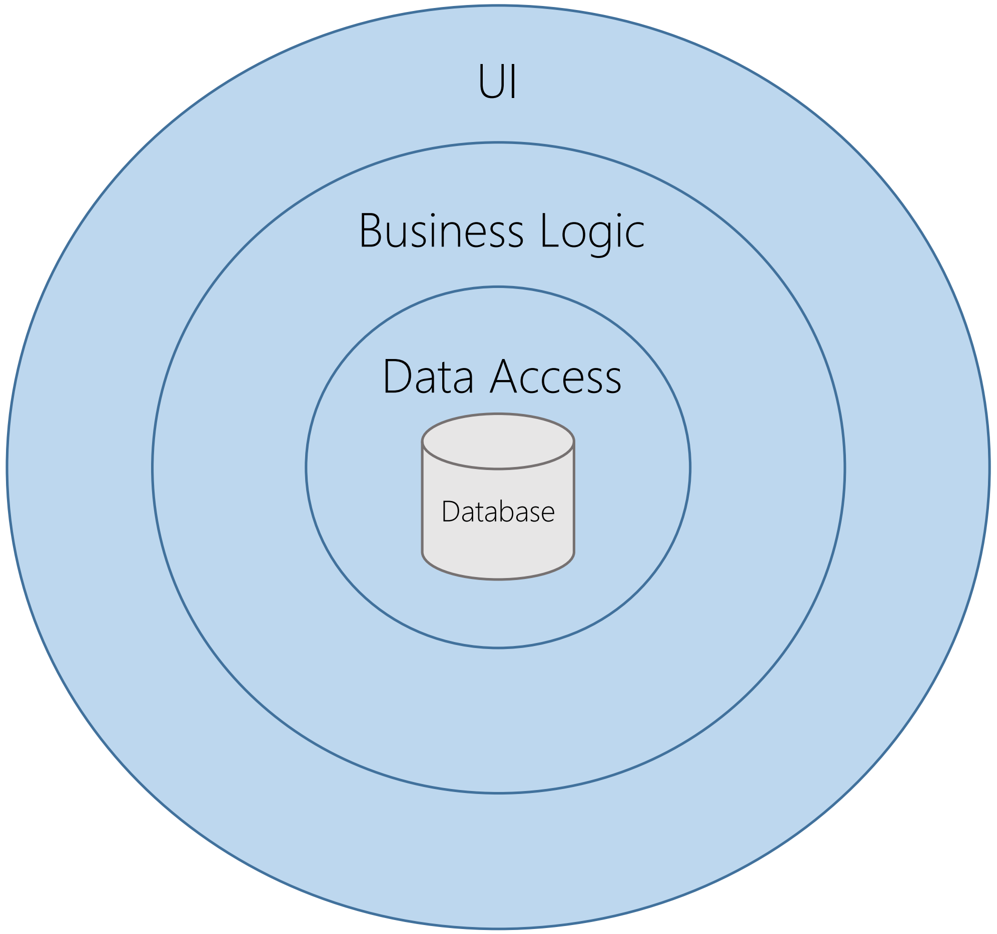


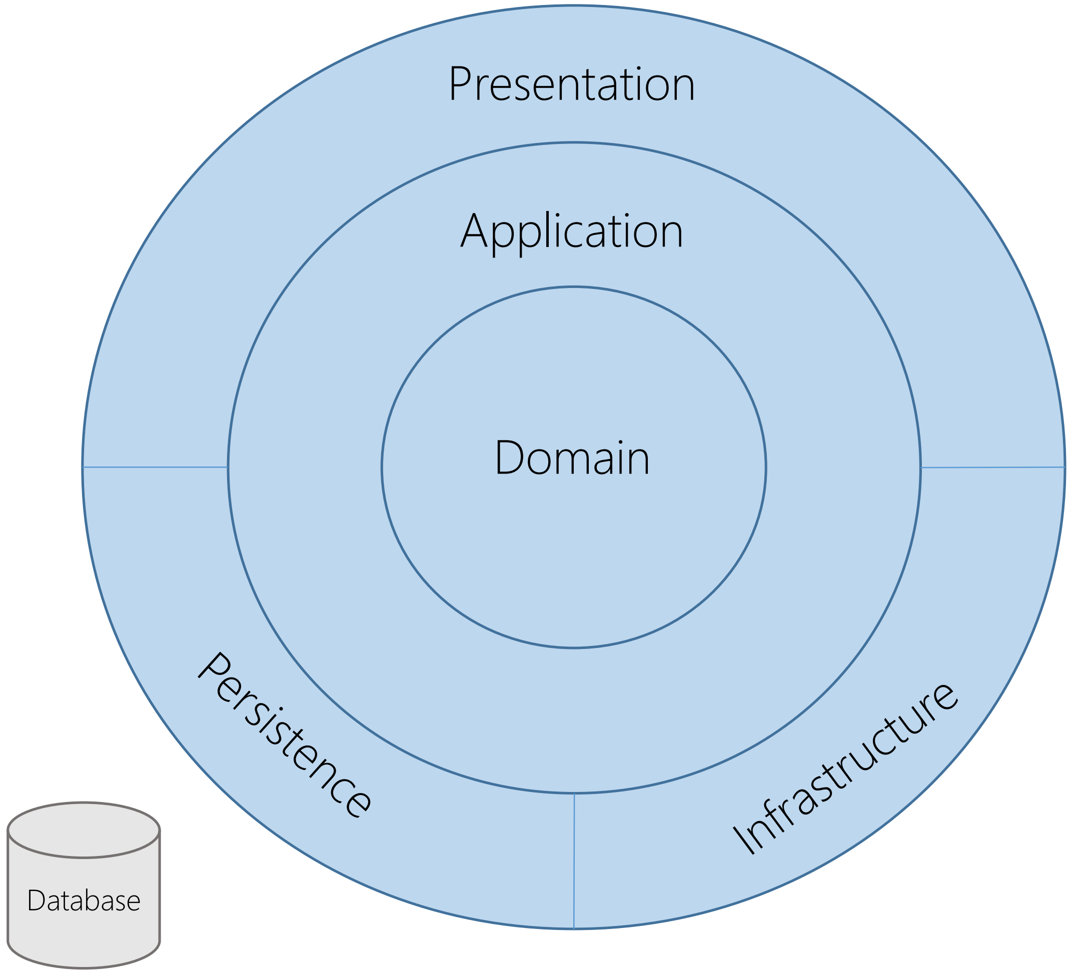
Heliocentric Model

Classic 3-layer Database-centric Architecture



Database- vs. Domain-centric Architecture





"The first concern of the architect is to make sure that the house is usable, it is not to ensure that the house is made of brick."

– Uncle Bob

Essential vs. Detail

Space is essential

Usability is essential

Essential vs. Detail

Building material is a detail

Ornamentation is a detail

Essential vs. Detail

Domain is essential

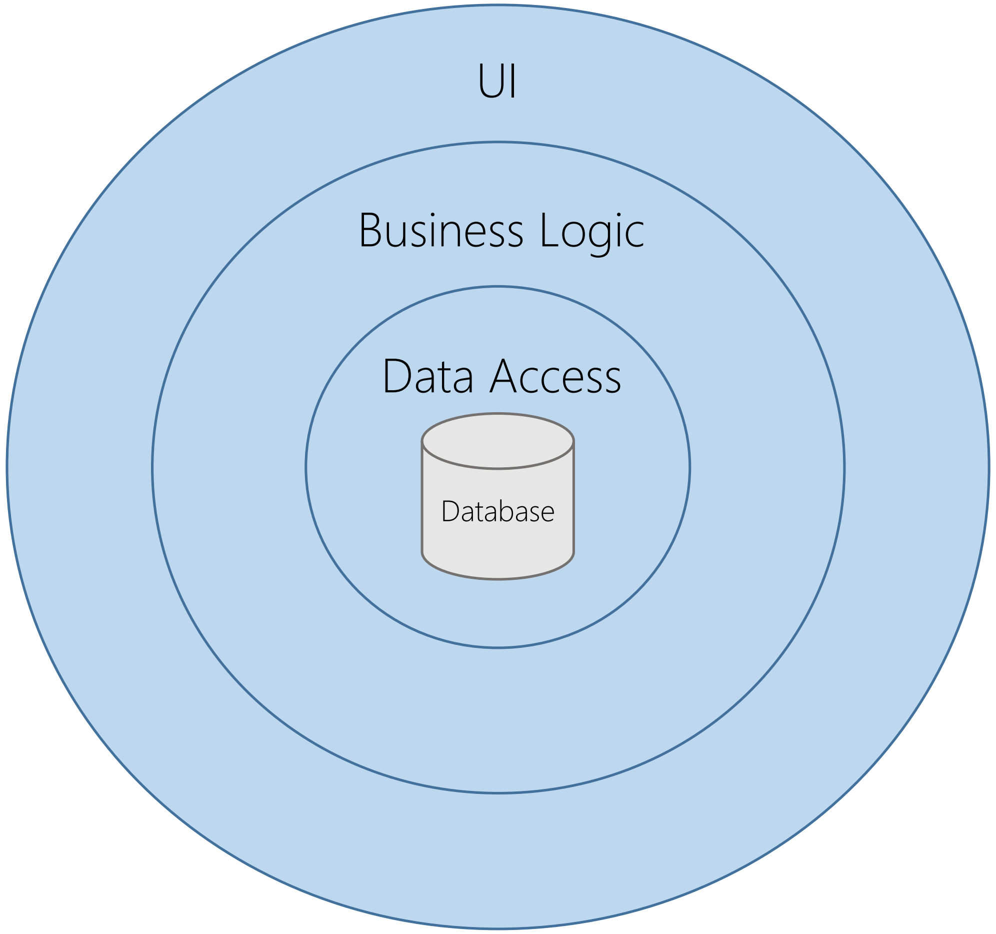
Use cases are essential

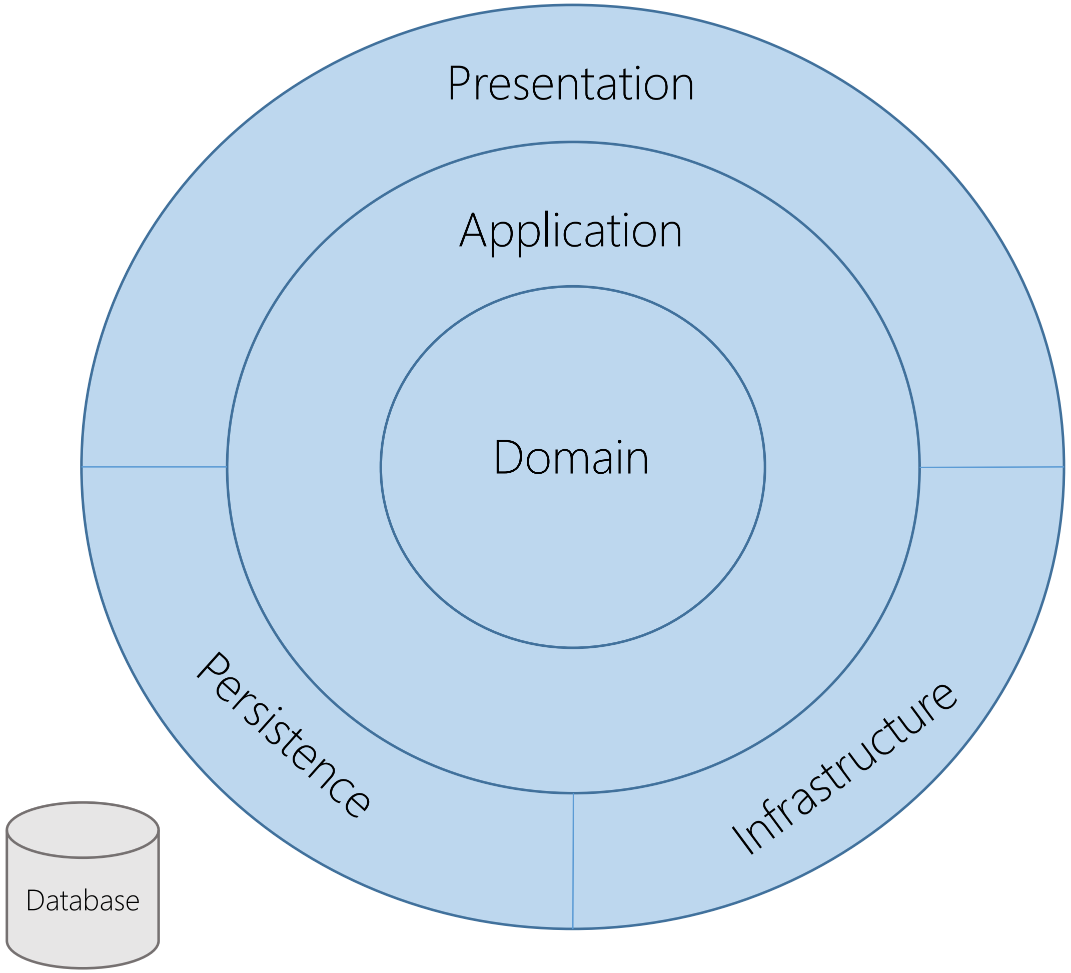
Essential vs. Detail

Presentation is a detail

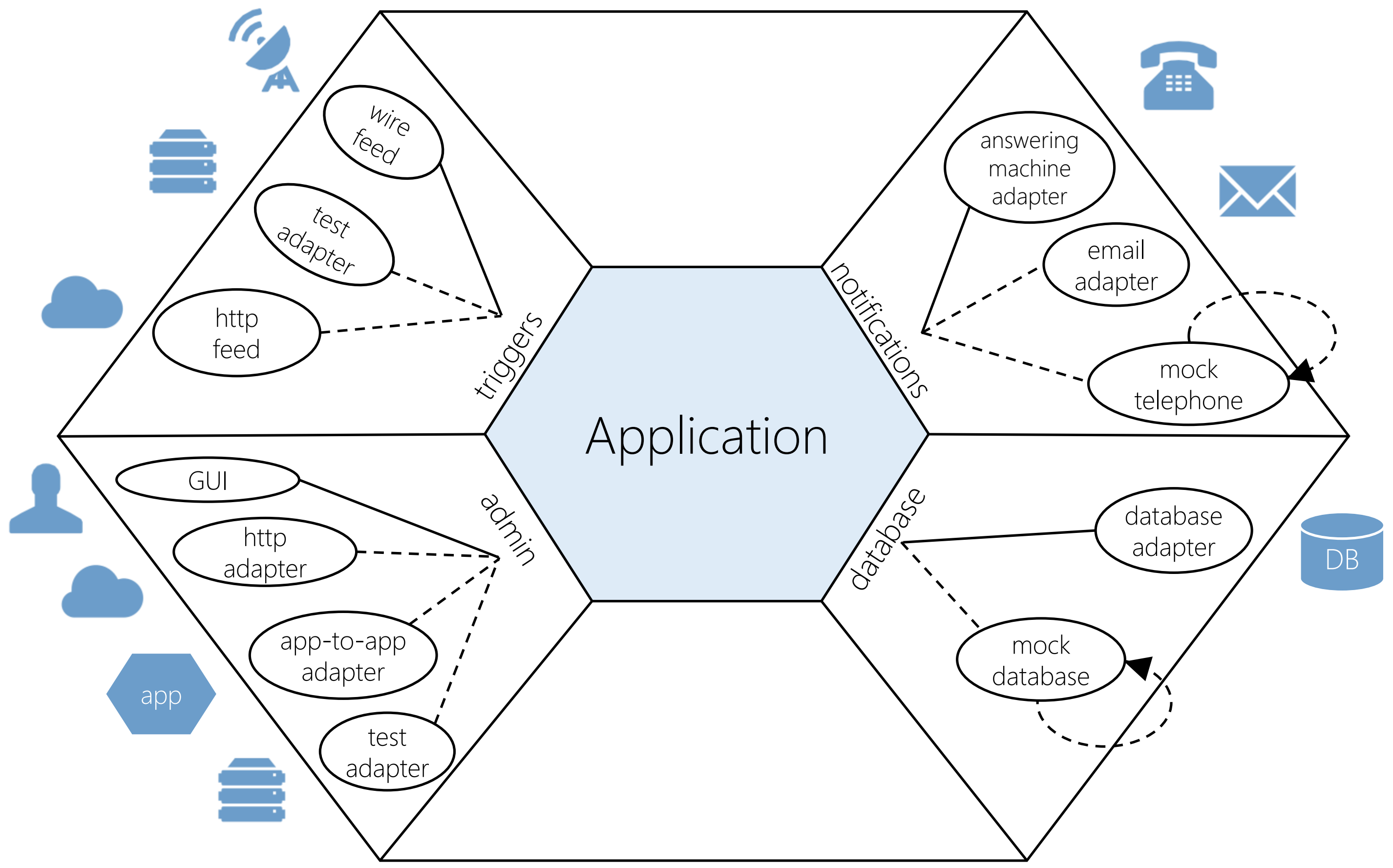
Persistence is a detail

Database- vs. Domain-centric Architecture



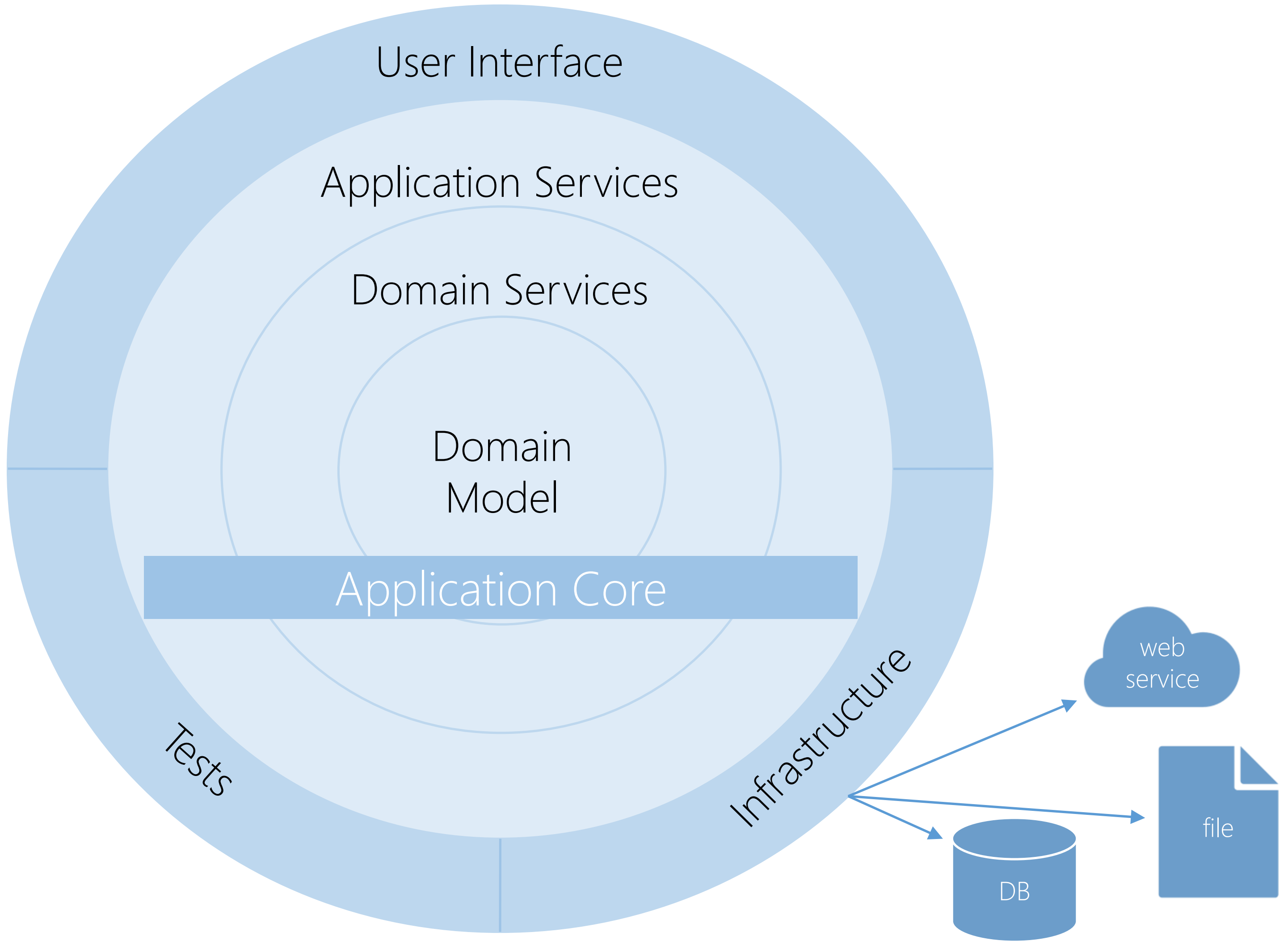


Hexagonal Architecture



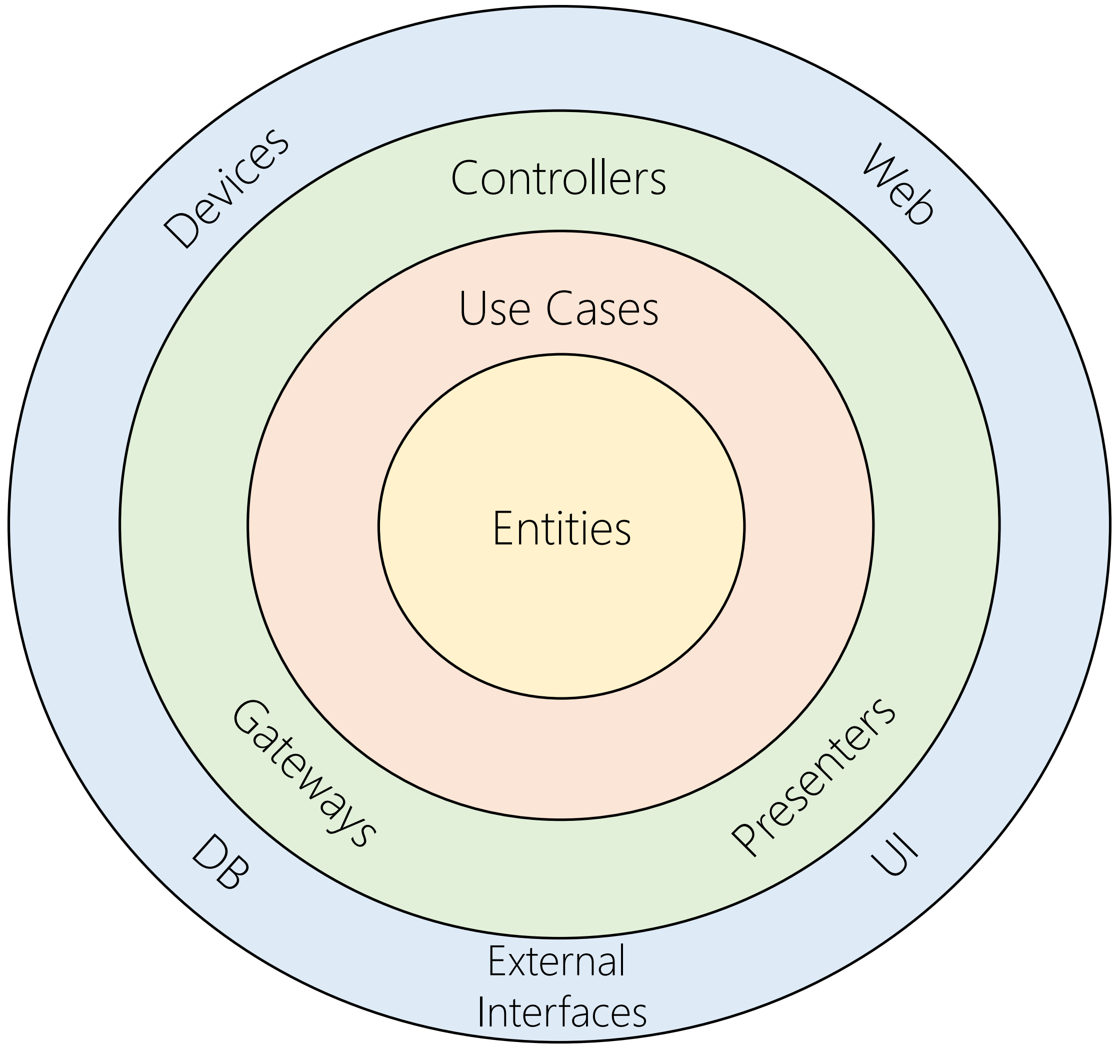
Original source: http://alistair.cockburn.us/Hexagonal+architecture

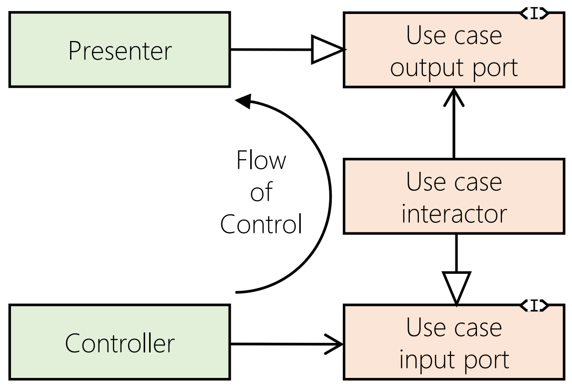
Onion Architecture



Original source: http://jeffreypalermo.com/blog/the-onion-architecture-part-2/

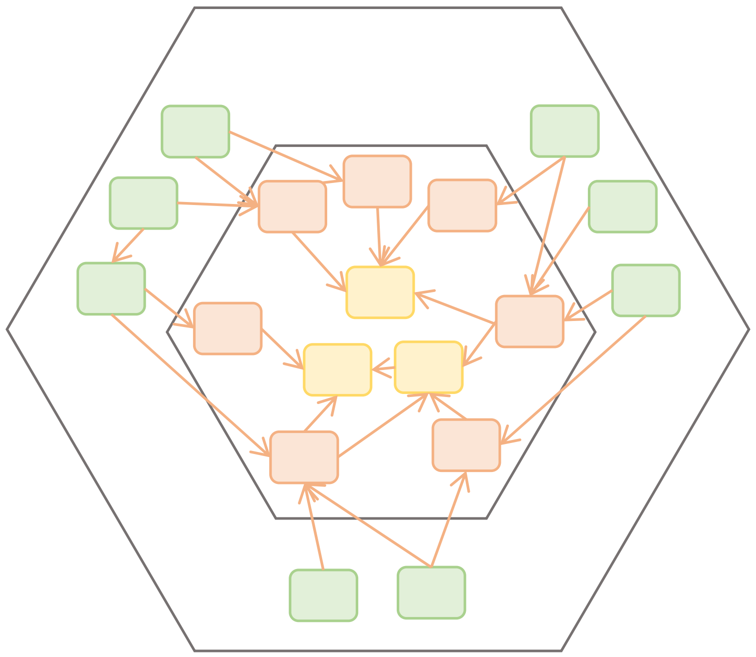
Clean Architecture



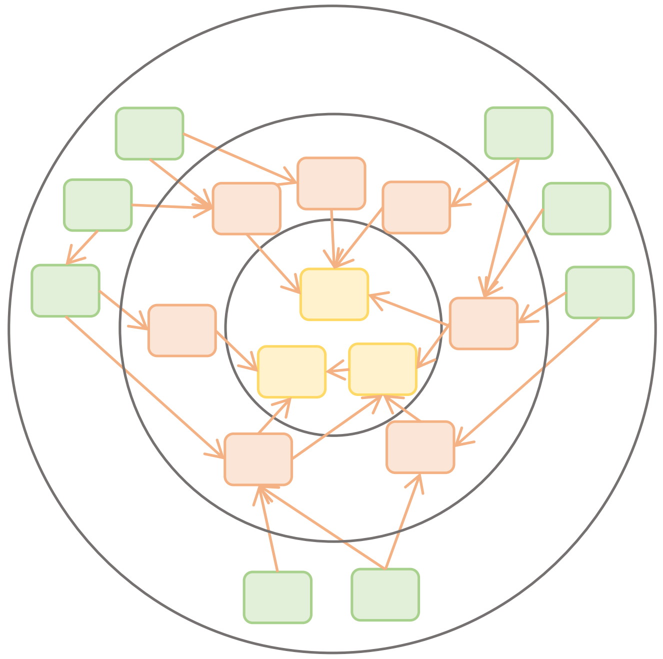


Original source: http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

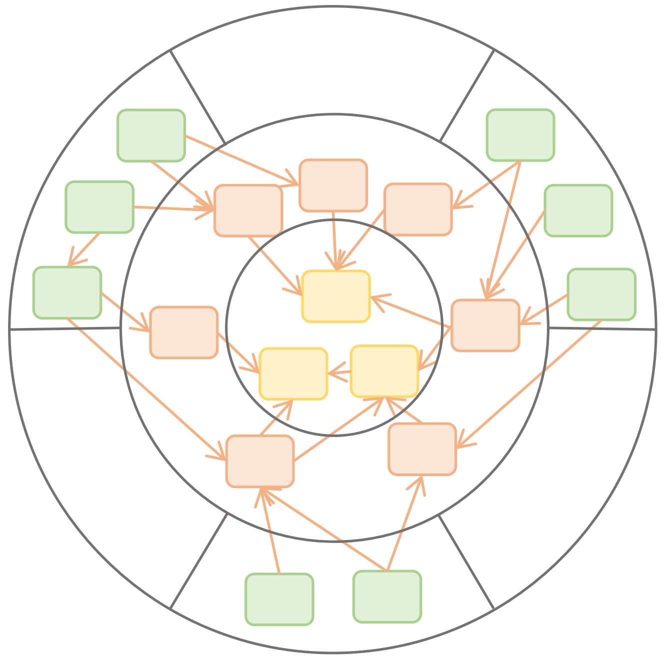
It’s All the Same Thing



Hexagonal



Onion



Clean

Original Source: http://blog.ploeh.dk/2013/12/03/layers-onions-ports-adapters-its-all-the-same/

Why Use Domain-Centric Architecture?

Pros

Focus on essential

Less coupling to details

Necessary for DDD

Why Use Domain-Centric Architecture?

Pros

Focus on essential

Less coupling to details

Necessary for DDD

Cons

Change is difficult

Requires extra thought

Initial higher cost

Application Layer

What Are Layers?

Levels of abstraction

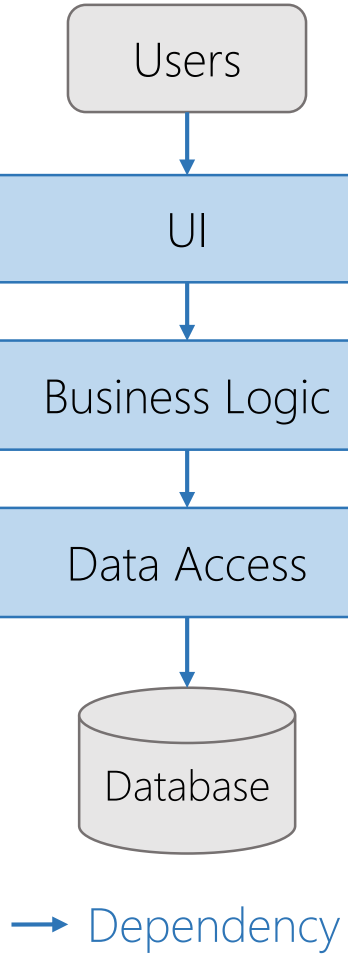
Single-Responsibility Principle

Developer roles / skills

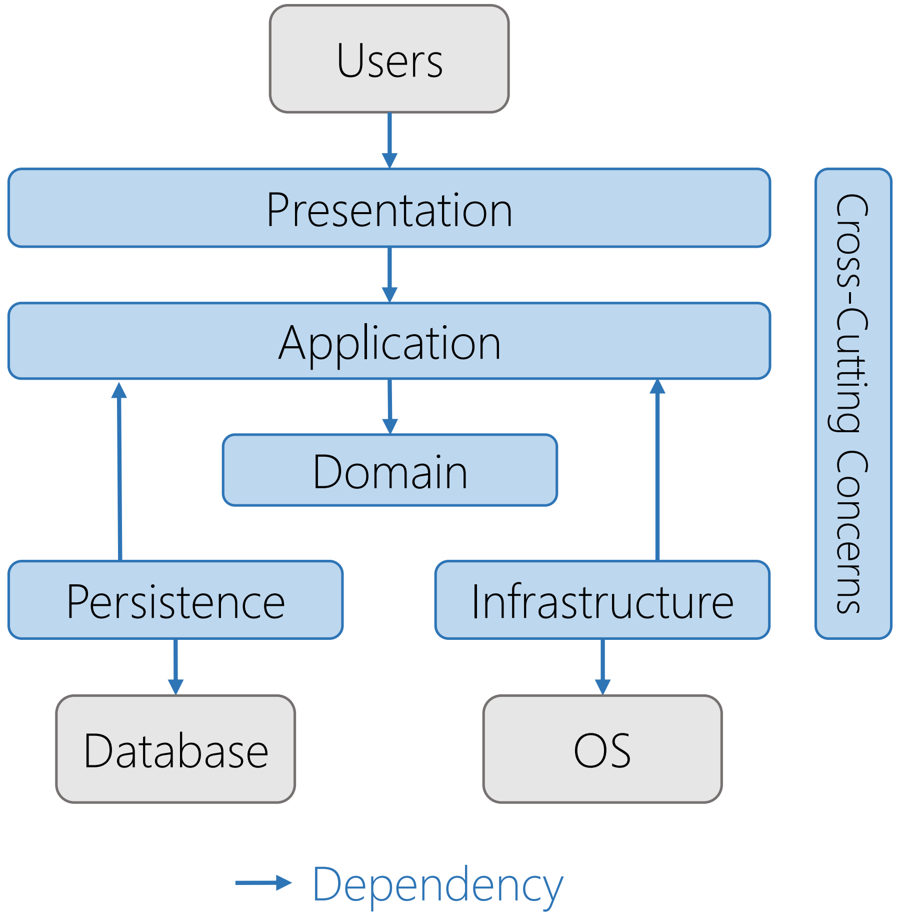
Multiple implementations

Varying rates of change

Classic 3-Layer Architecture



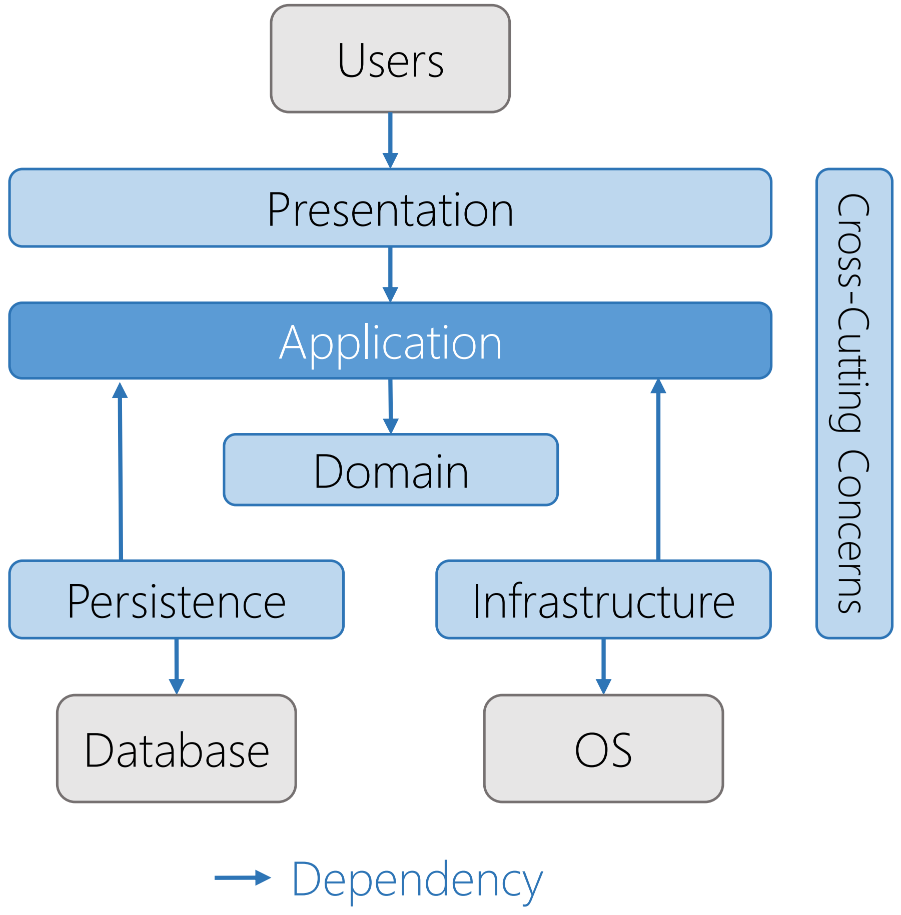
Modern 4-Layer Architecture



Application Layer

Implements use cases

High-level application logic

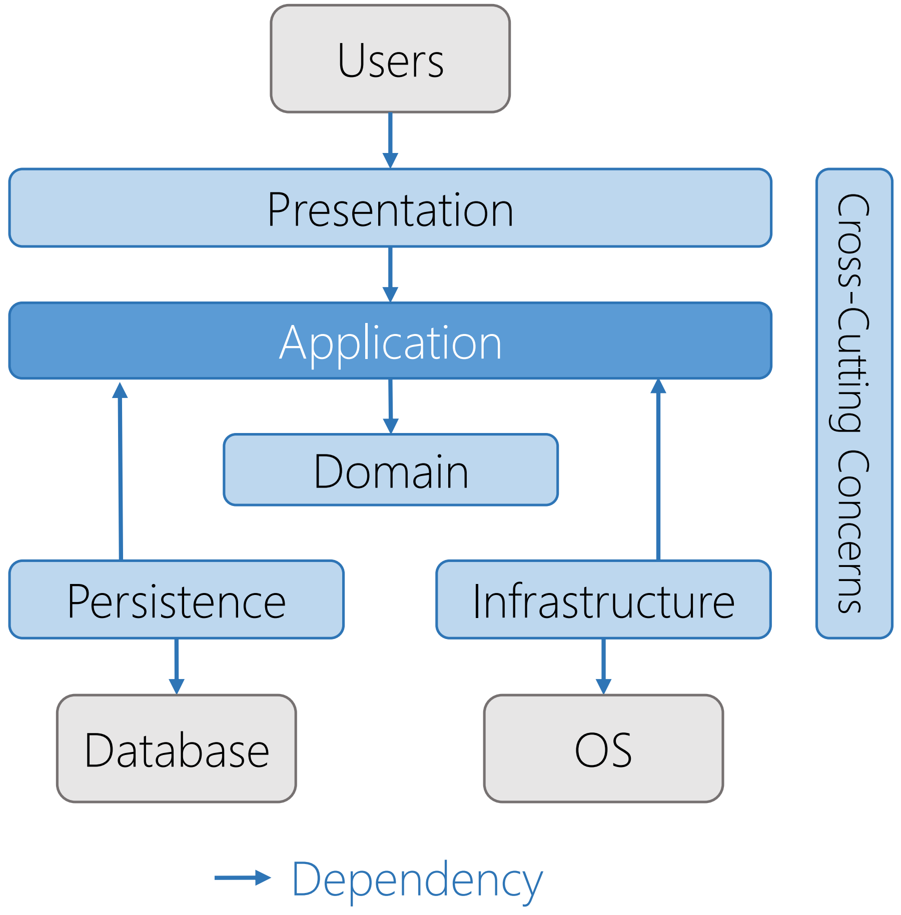


Application Layer

Knows about domain

No knowledge of other layers

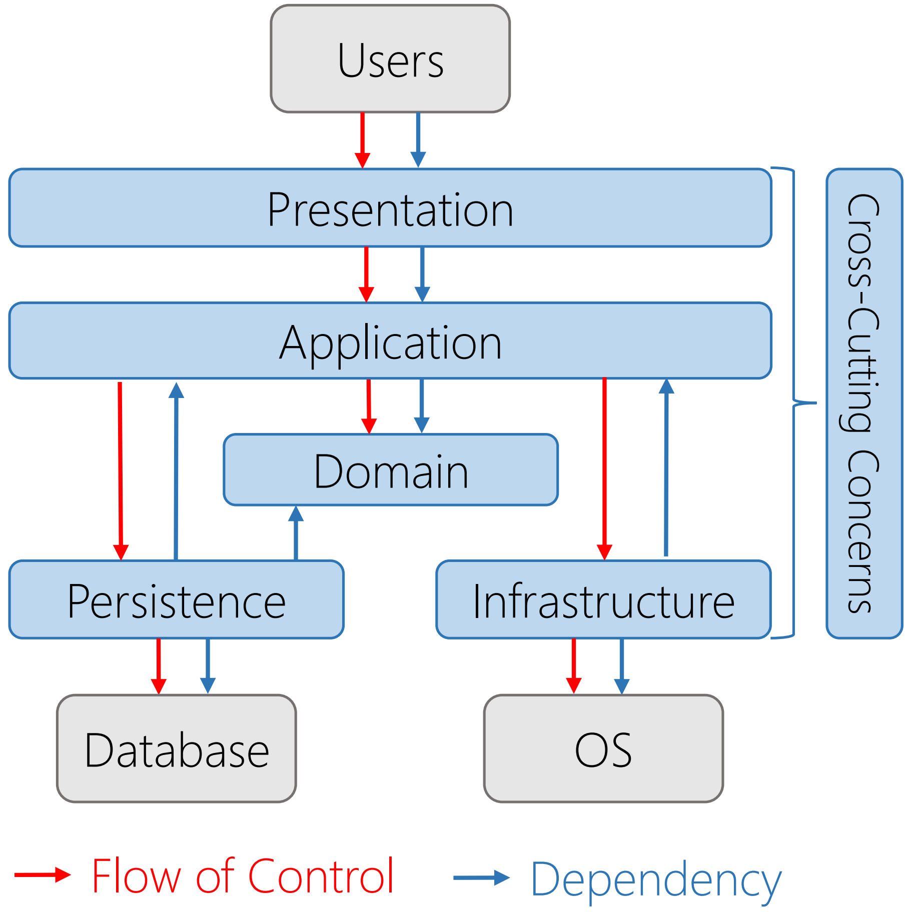
Contains interfaces for details



Layer Dependencies

Dependency inversion

Inversion of control



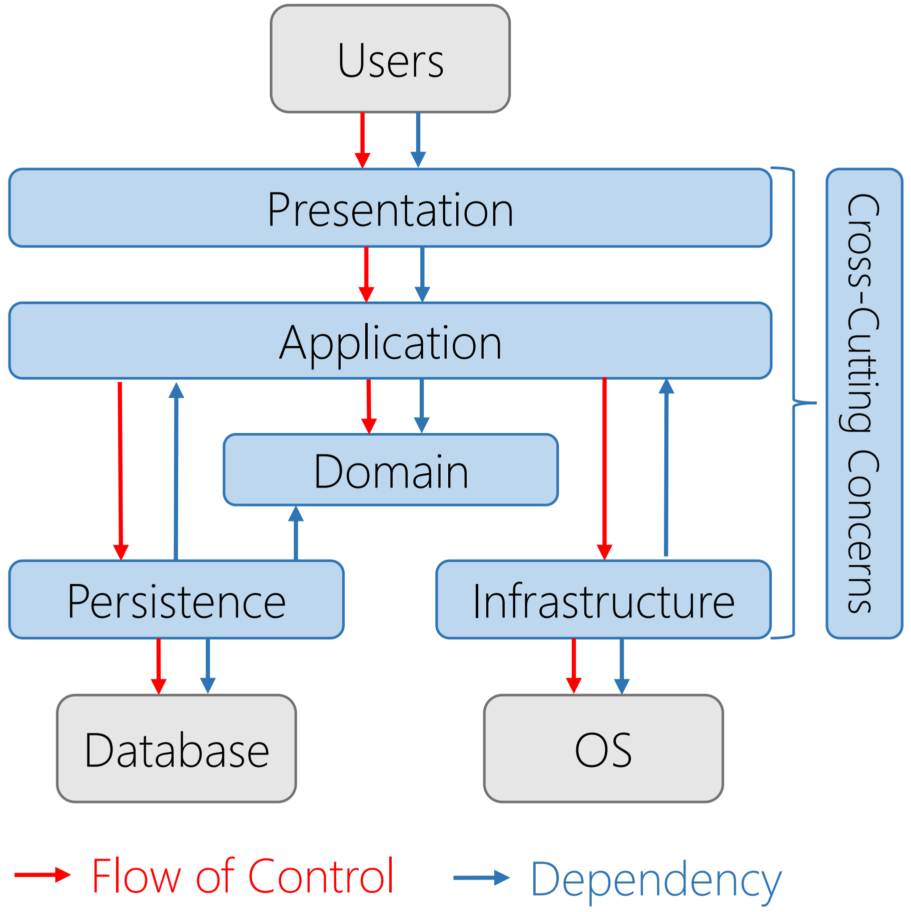
Layer Dependencies

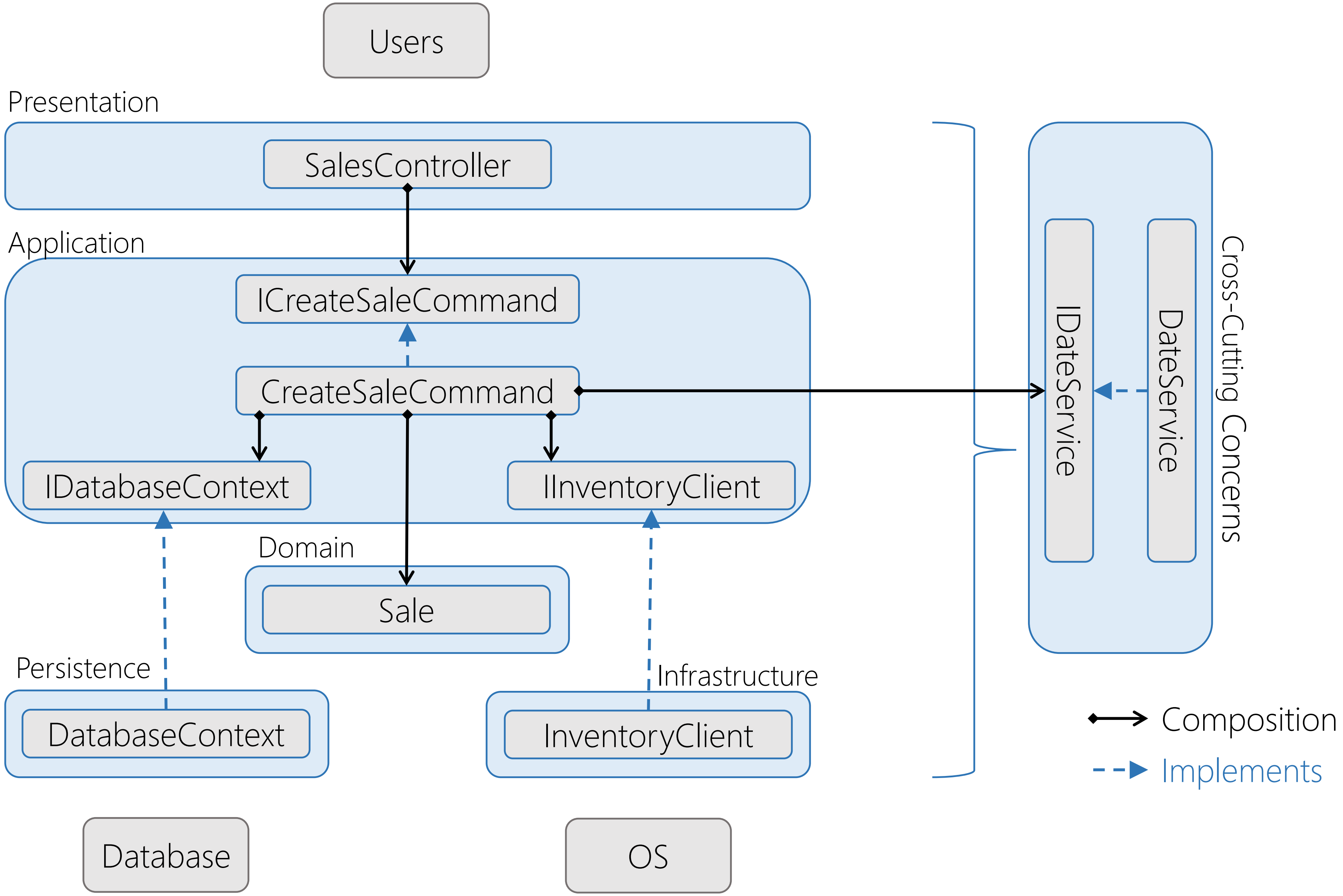
Dependency inversion

Inversion of control

Independent deployability

Flexibility and maintainability




Users
Presentation
SalesController
Application
ICreateSaleCommand
CreateSaleCommand
IDatabaseContext
IInventoryClient
Domain
Sale
Persistence
DatabaseContext
Infrastructure
InventoryClient
Database
OS
IDateService
DateService
Cross-Cutting Concerns
Composition
Implements

Why Use an Application Layer?

Pros

Focus is on use cases

Easy to understand

Follows DIP

Why Use an Application Layer?

Pros

Focus is on use cases

Easy to understand

Follows DIP

Cons

Additional cost

Requires extra thought

IoC is counter-intuitive

Commands and Queries

Command-Query Separation

Command

Does something

Should modify state

Should not return a value

Command-Query Separation

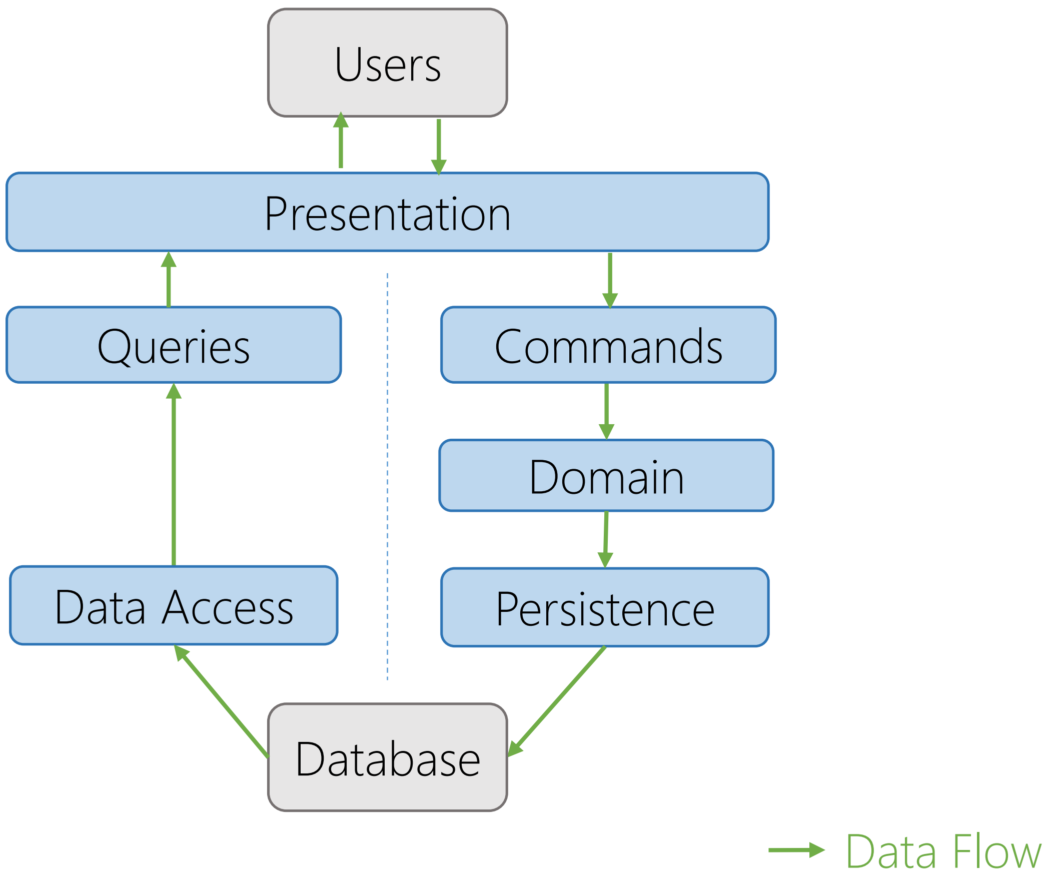
Command	Query
Does something	Answers a question
Should modify state	Should not modify state
Should not return a value	Always returns a value

Command-Query Separation

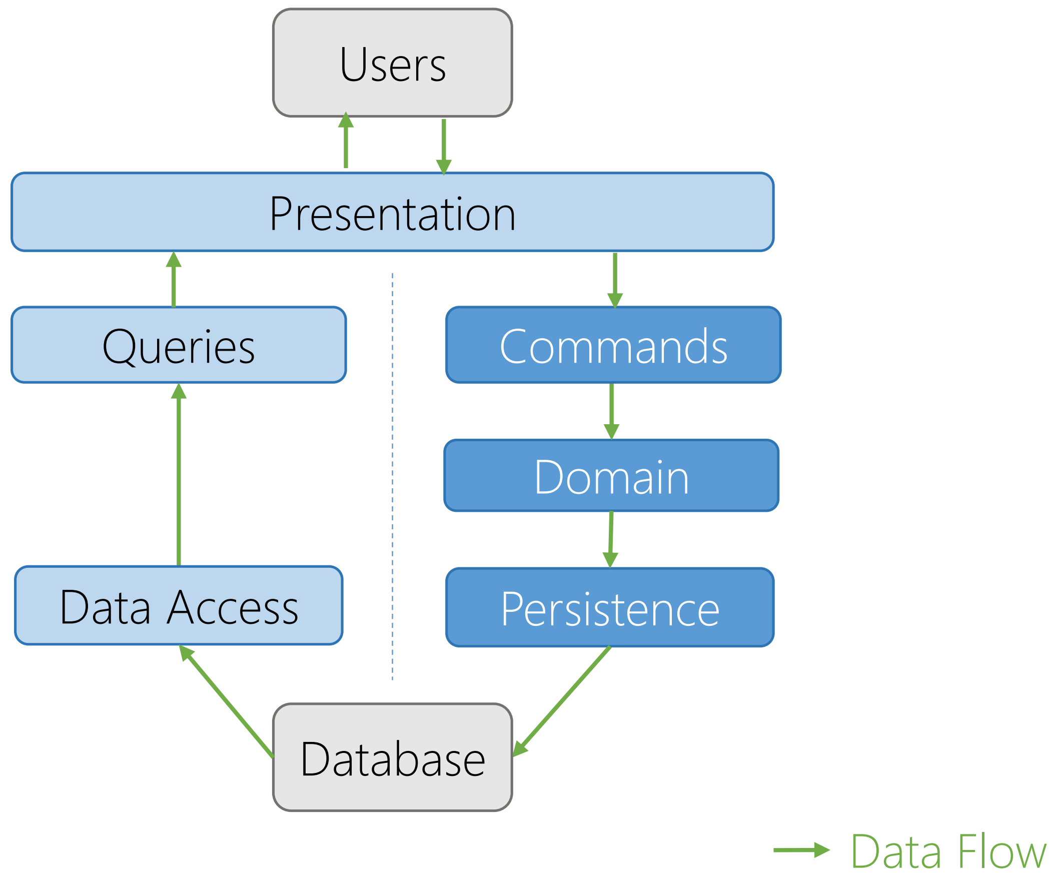
Command	Query
Does something	Answers a question
Should modify state	Should not modify state
Should not return a value *(ideally)*	Always returns a value

Avoid mixing the two!

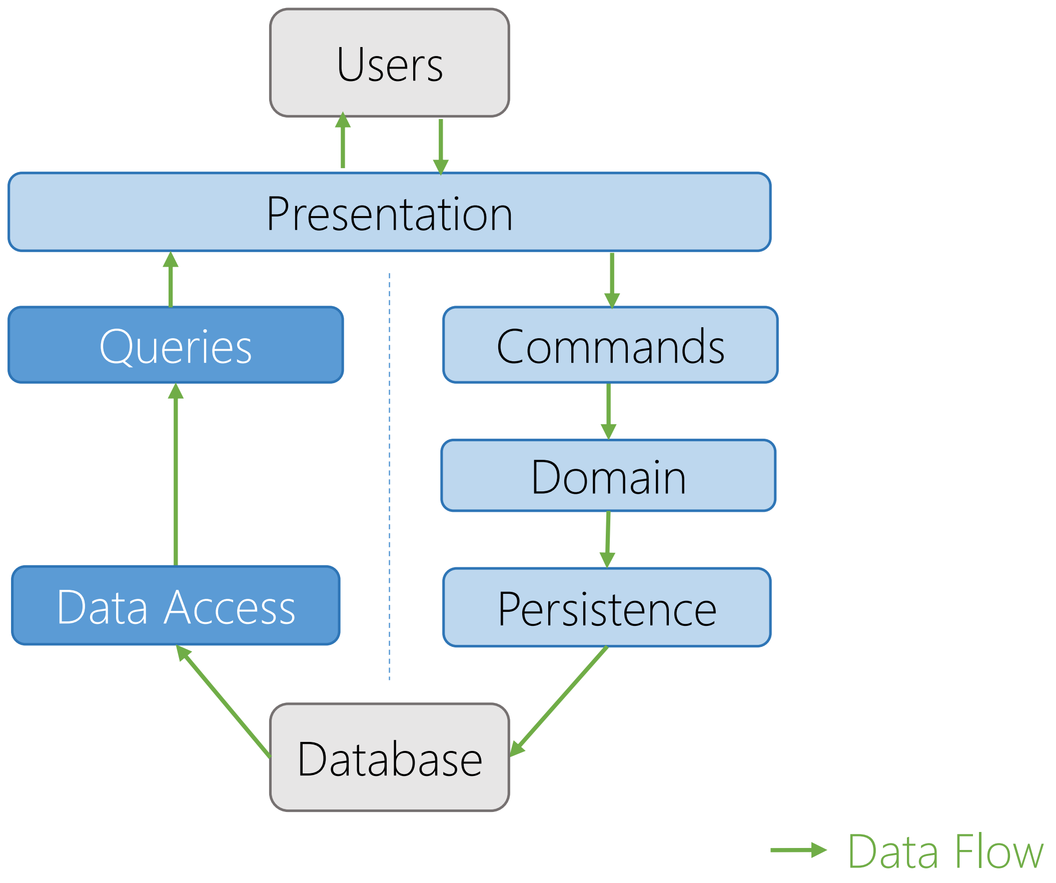
CQRS Architectures



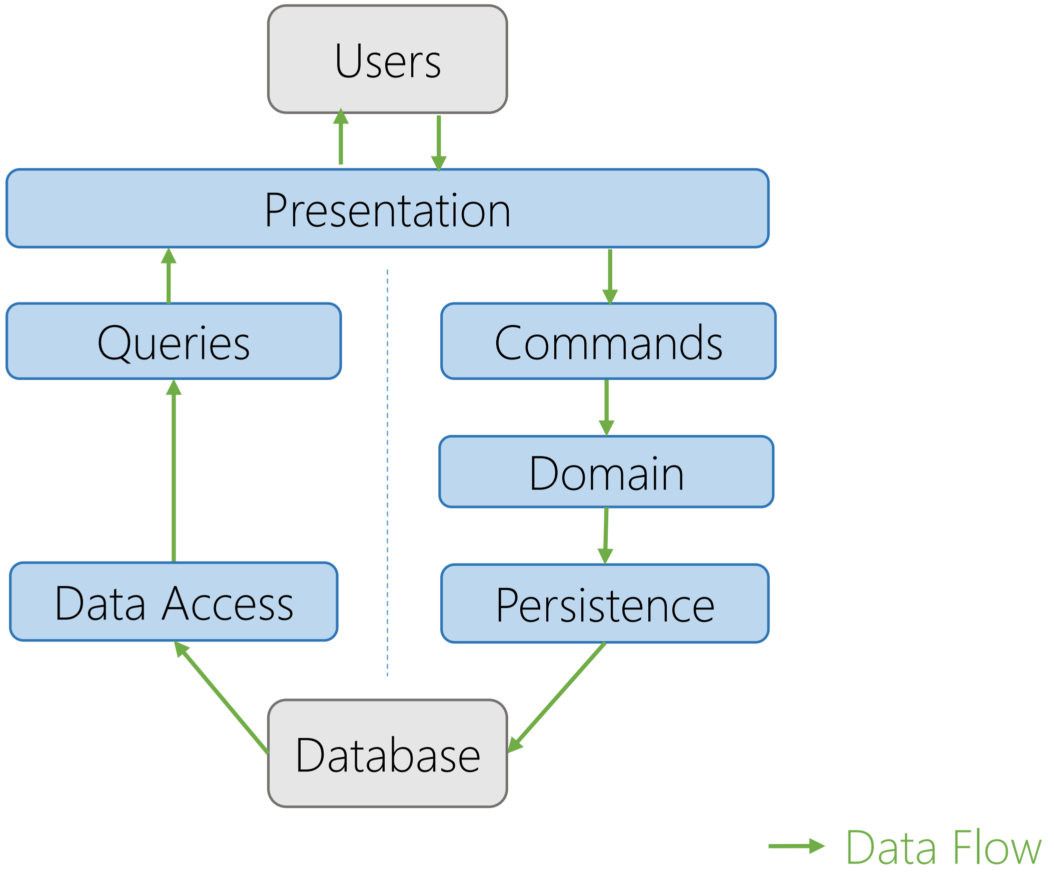
CQRS Architectures



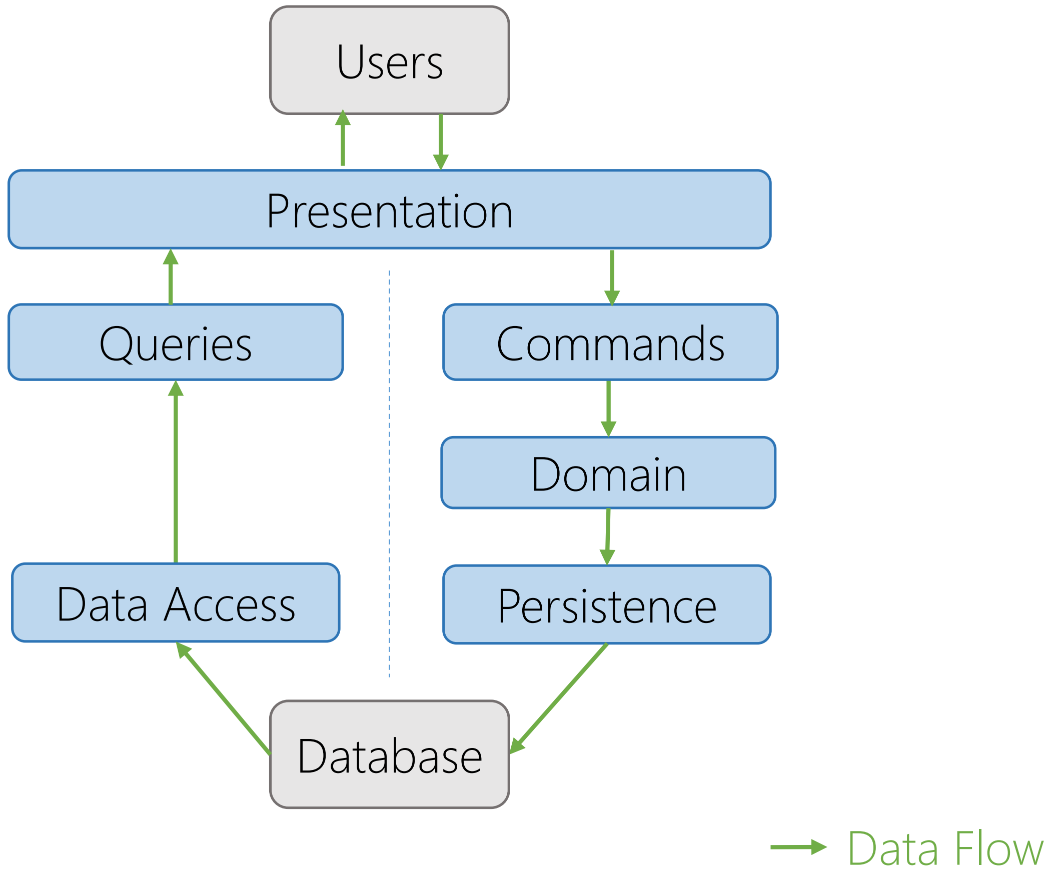
CQRS Architectures



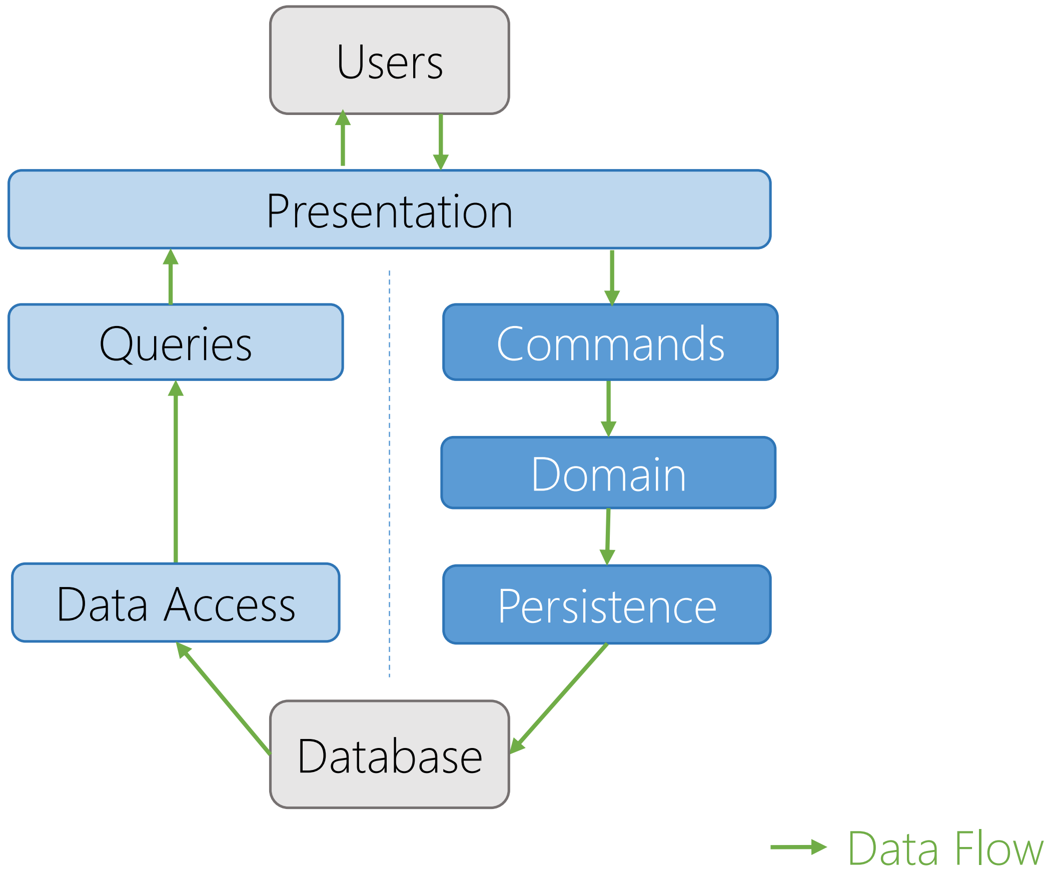
CQRS Architectures



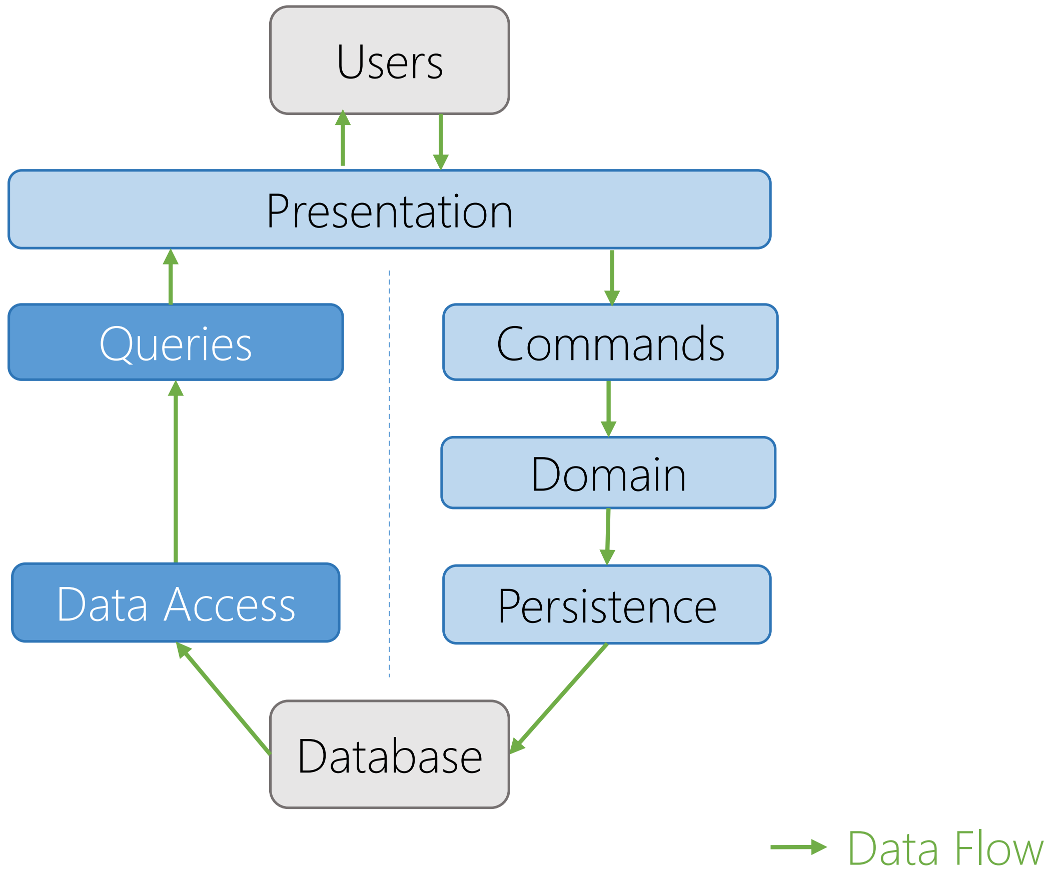
CQRS Type 1 – Single Database



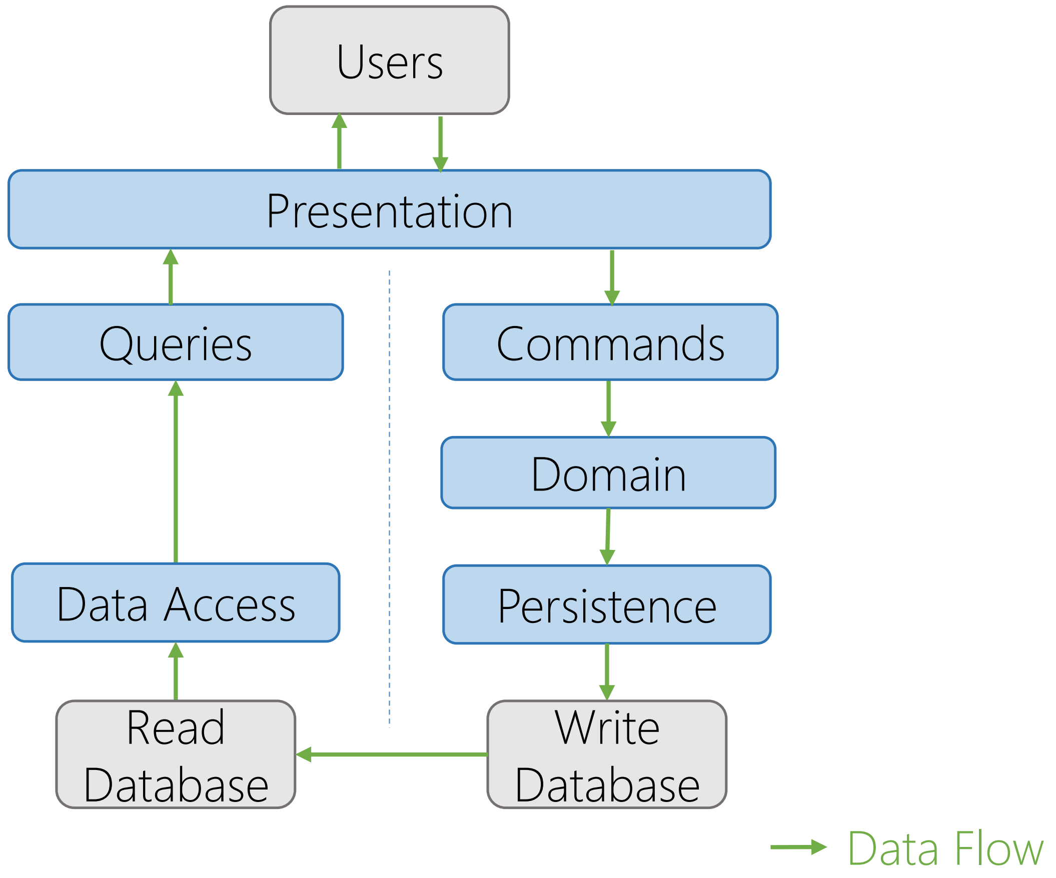
CQRS Type 1 – Single Database



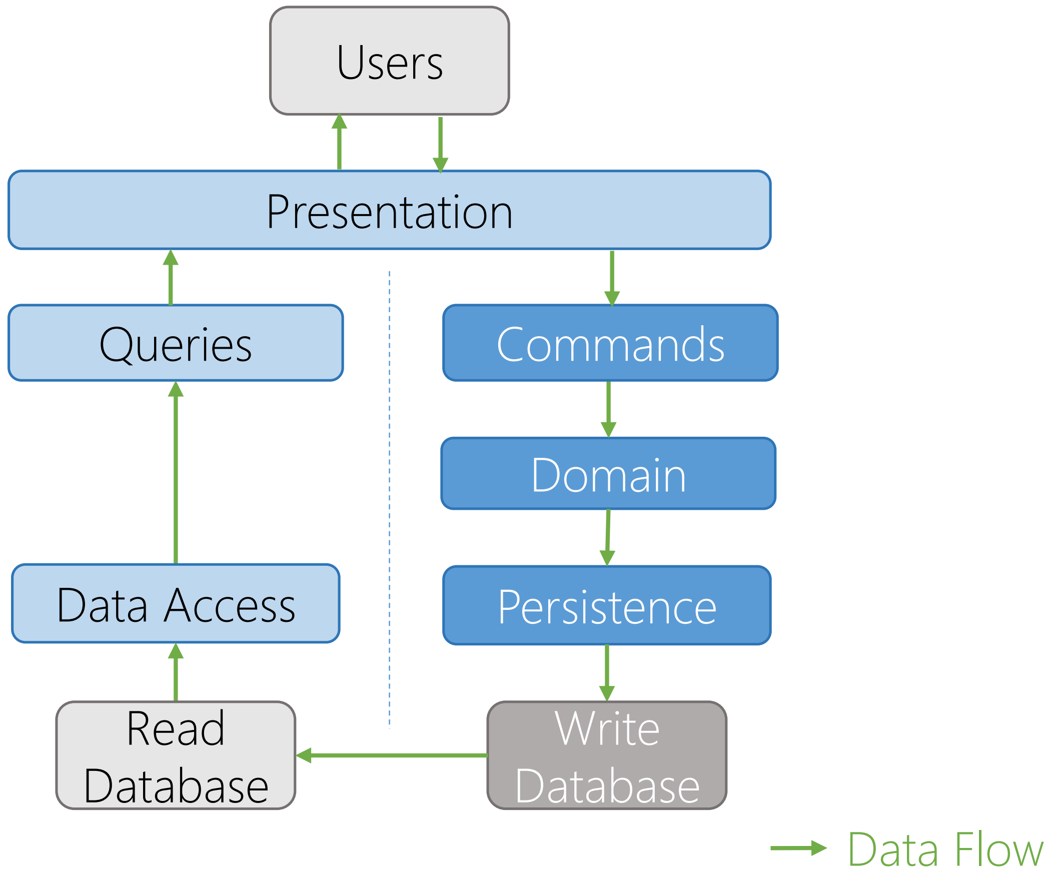
CQRS Type 1 – Single Database



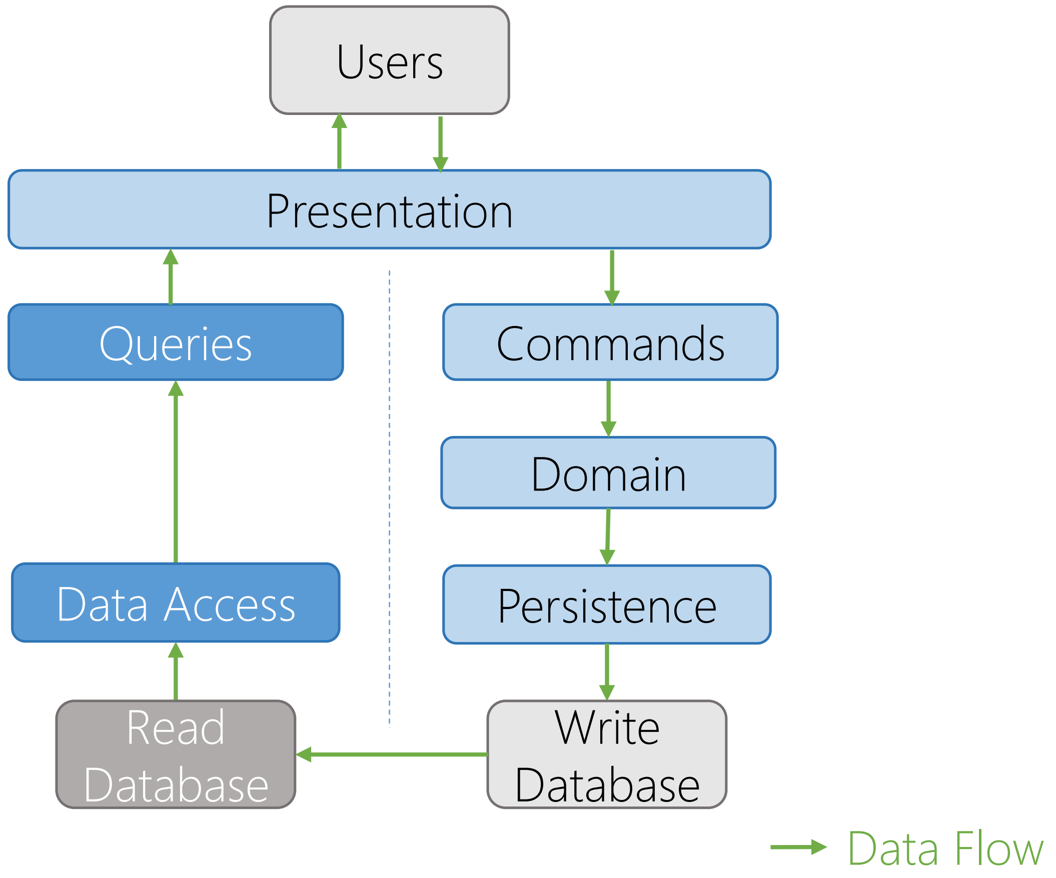
CQRS Type 2 – Read/Write Databases



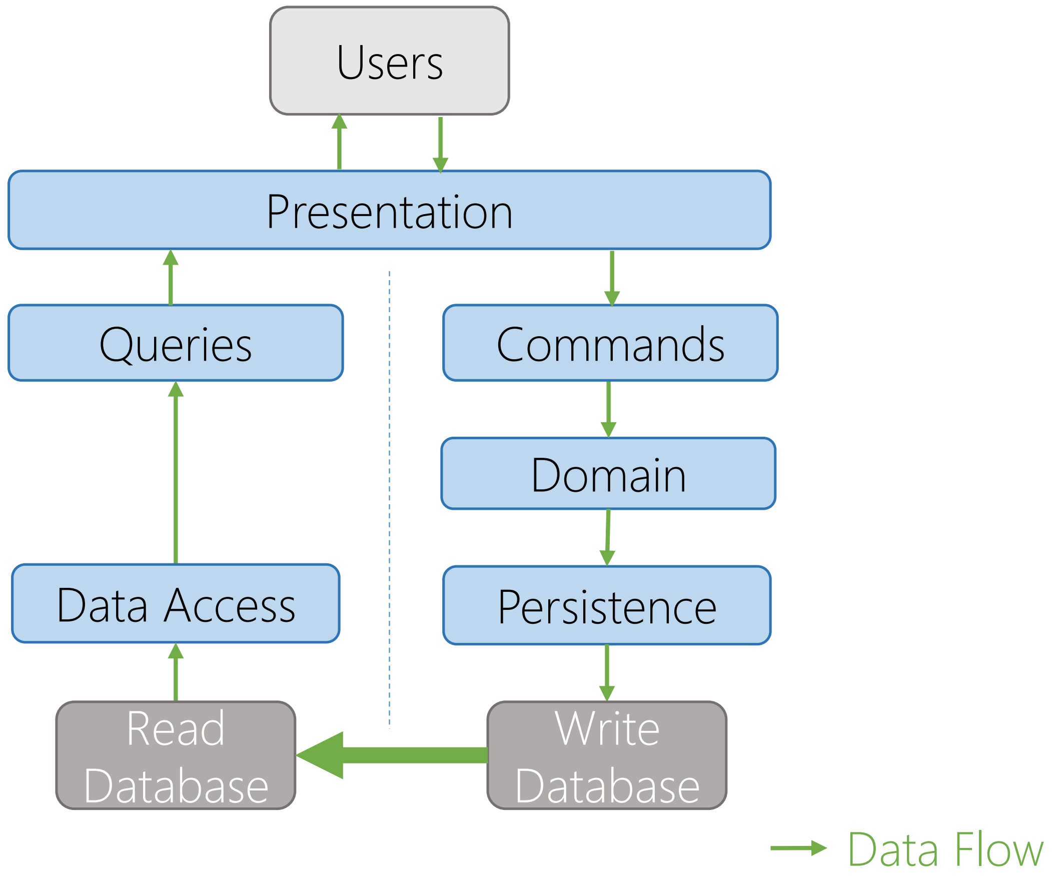
CQRS Type 2 – Read/Write Databases



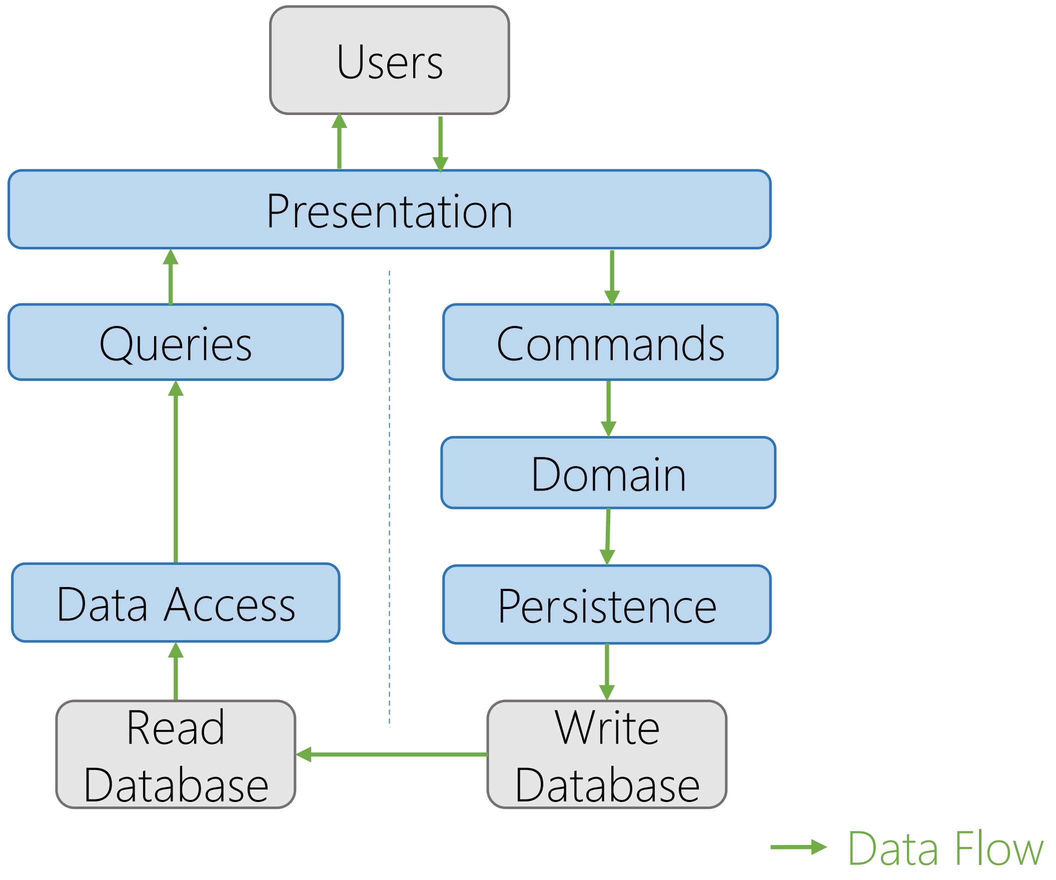
CQRS Type 2 – Read/Write Databases



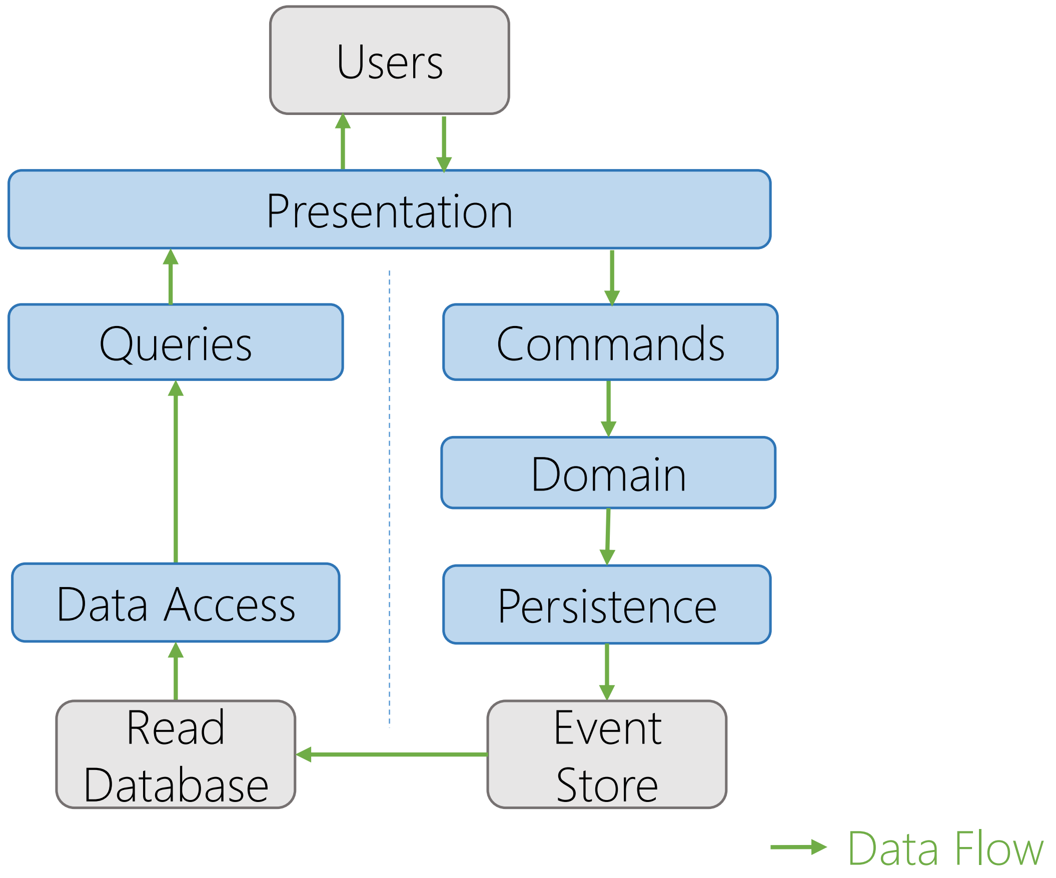
CQRS Type 2 – Read/Write Databases



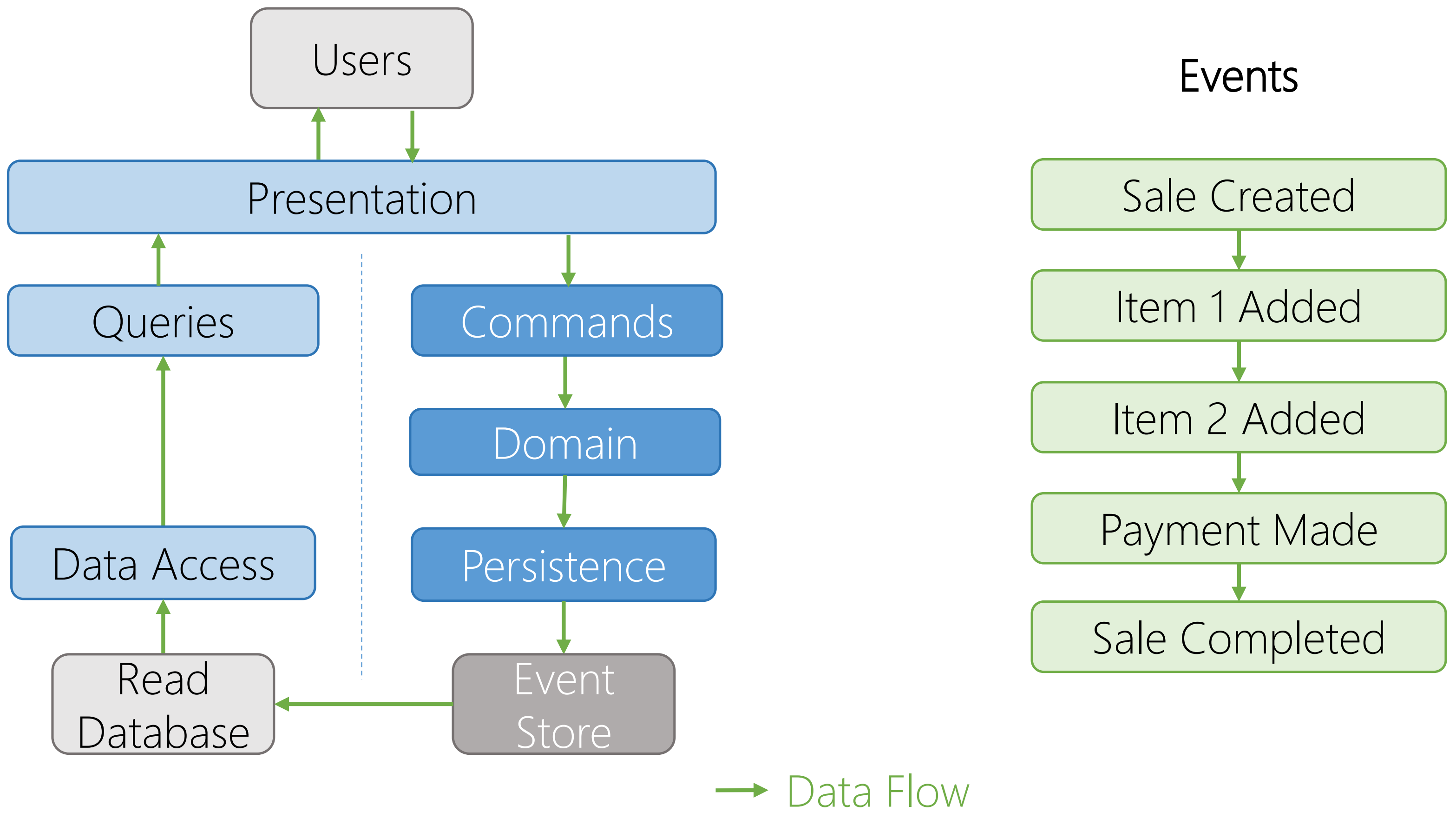
CQRS Type 2 – Read/Write Databases



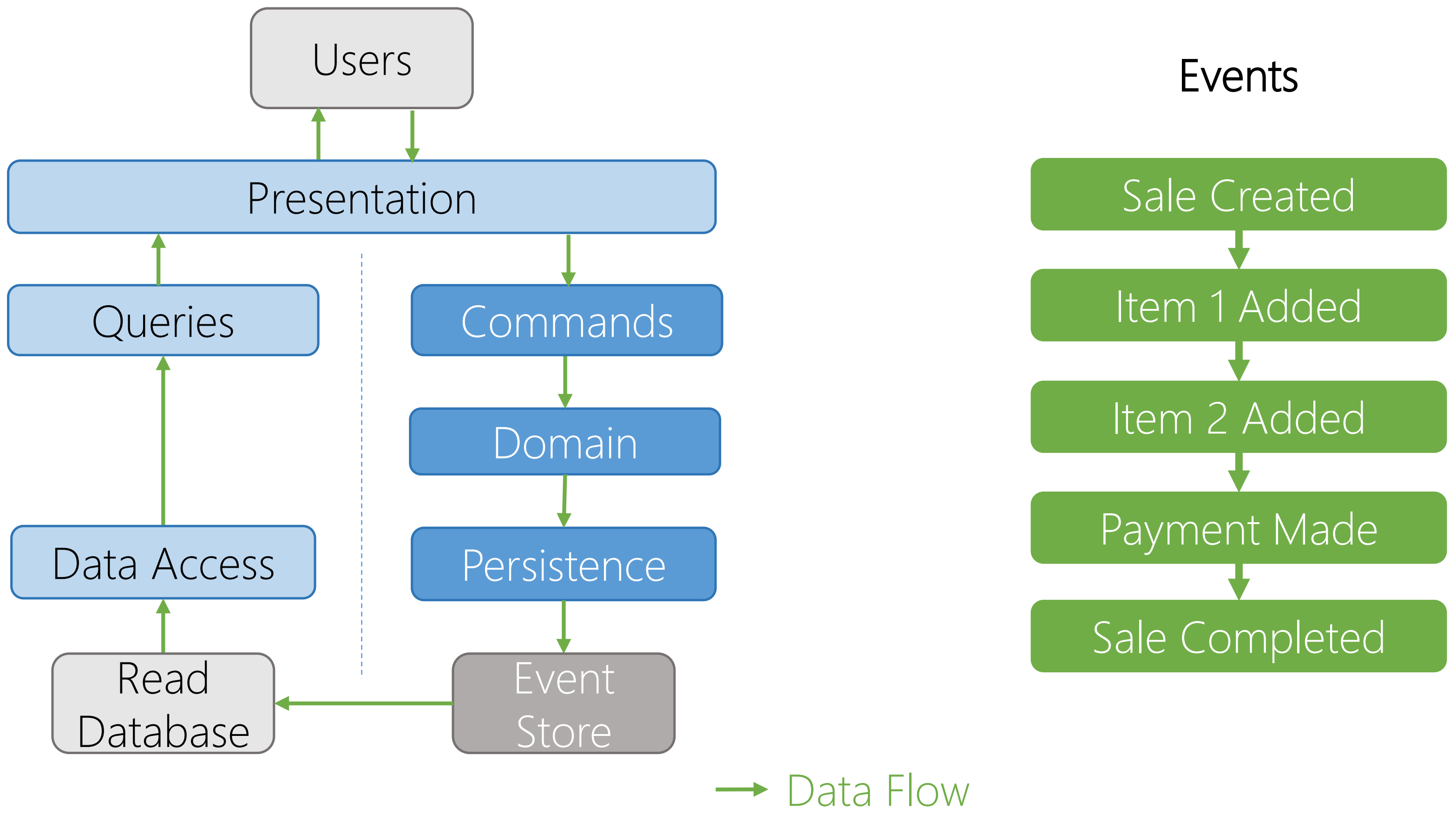
CQRS Type 3 – Event Sourcing



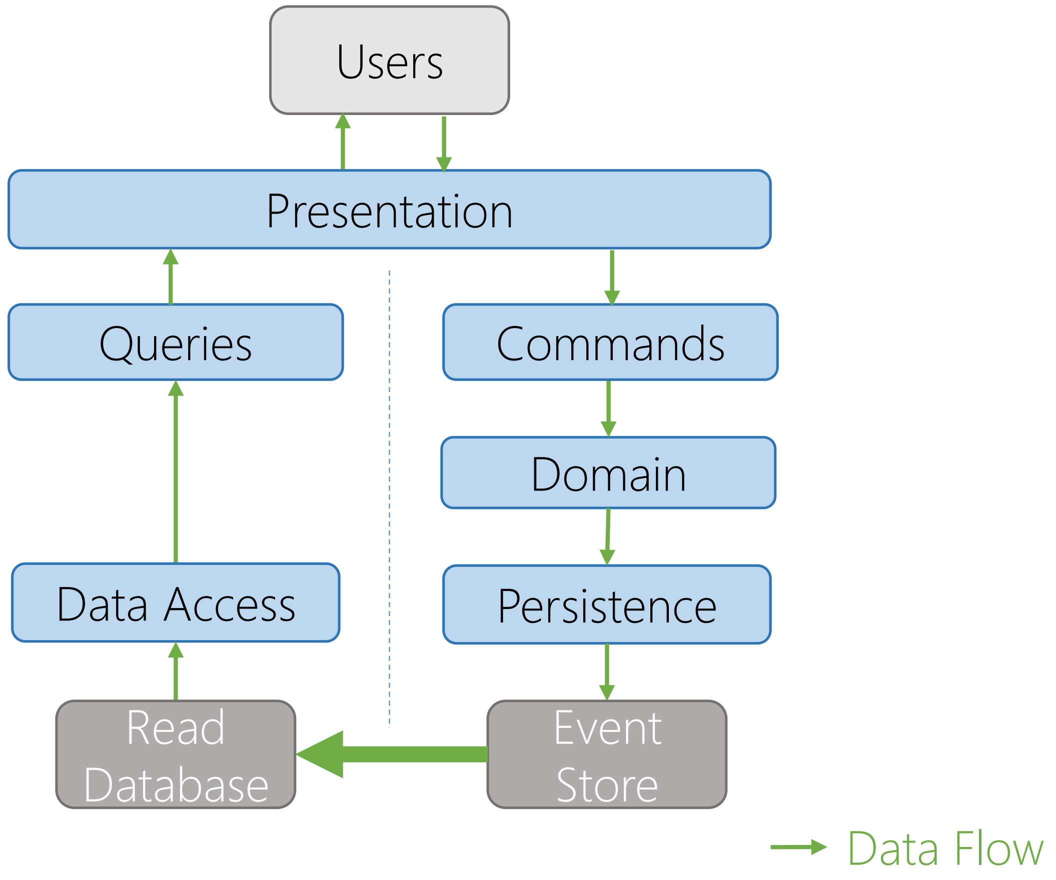
CQRS Type 3 – Event Sourcing



CQRS Type 3 – Event Sourcing



CQRS Type 3 – Event Sourcing



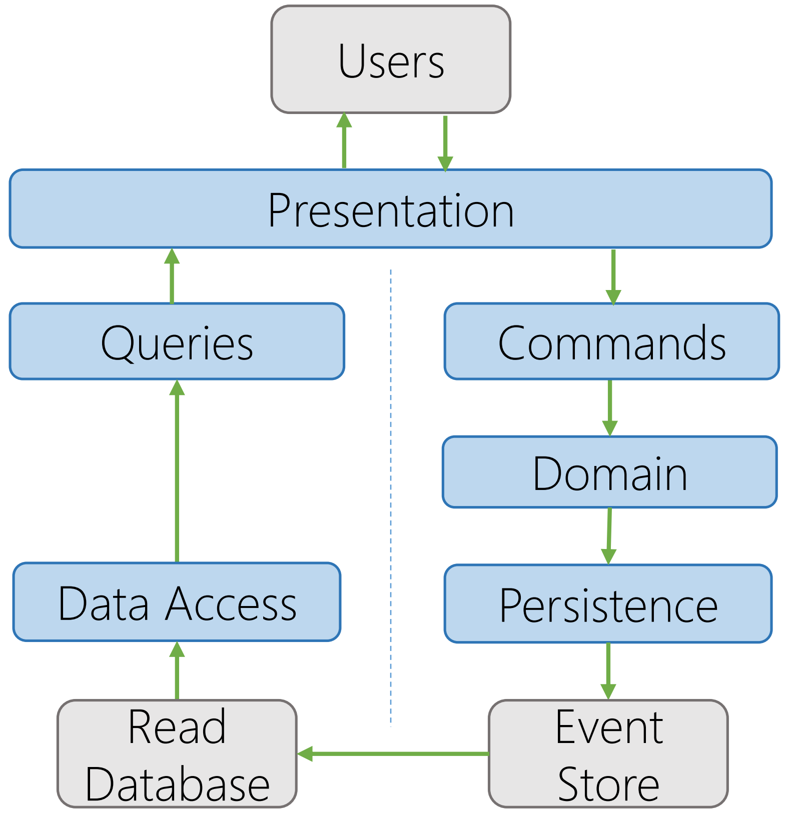
CQRS Type 3 – Event Sourcing

Complete audit trail

Point-in-time reconstruction

Replay events

Rebuild production database



Why Use CQRS?

Pros

More efficient design

Simpler within each stack

Optimized performance

Why Use CQRS?

Pros

More efficient design

Simpler within each stack

Optimized performance

Cons

Inconsistent across stacks

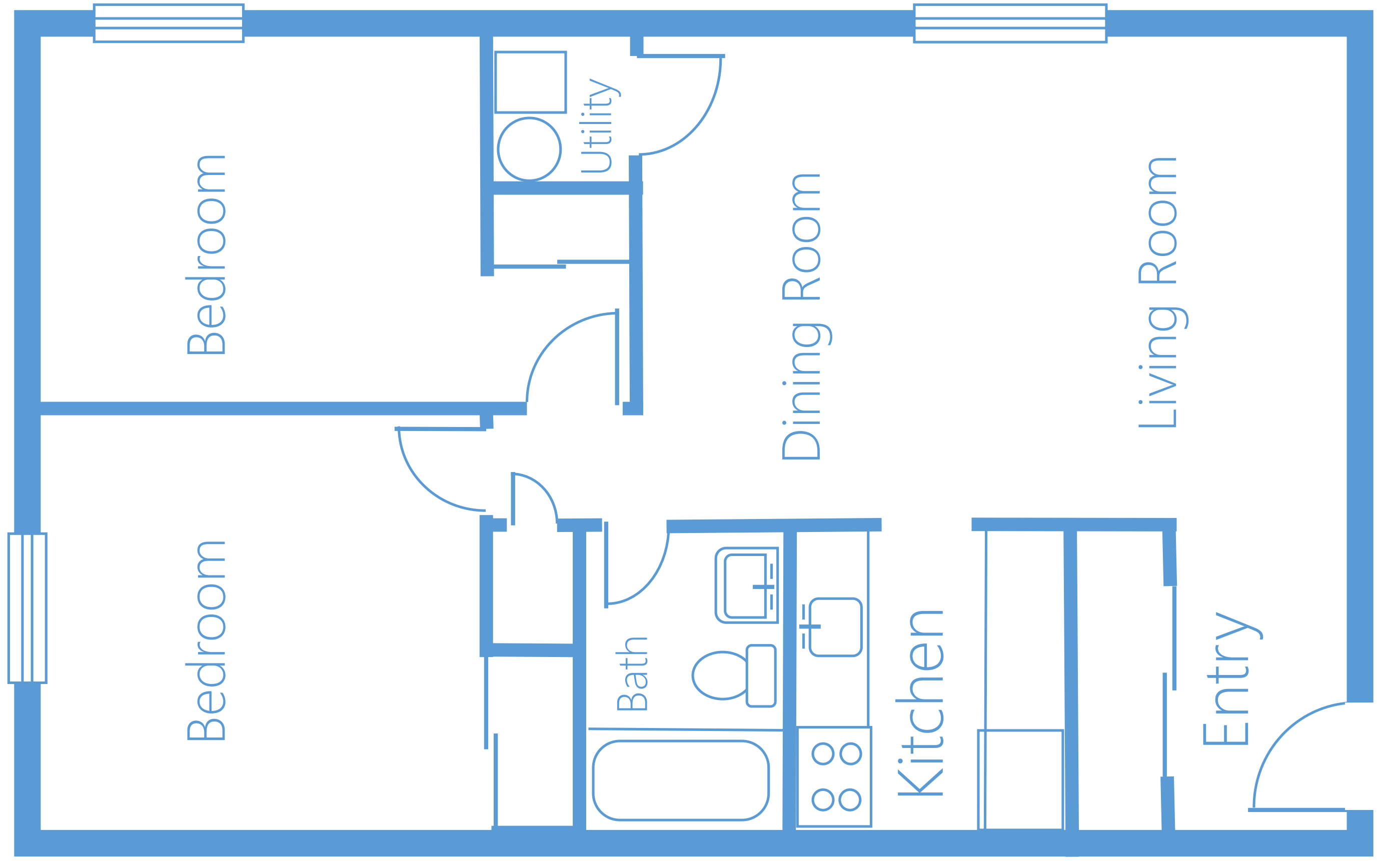
Type 2 is more complex

Type 3 might be overkill

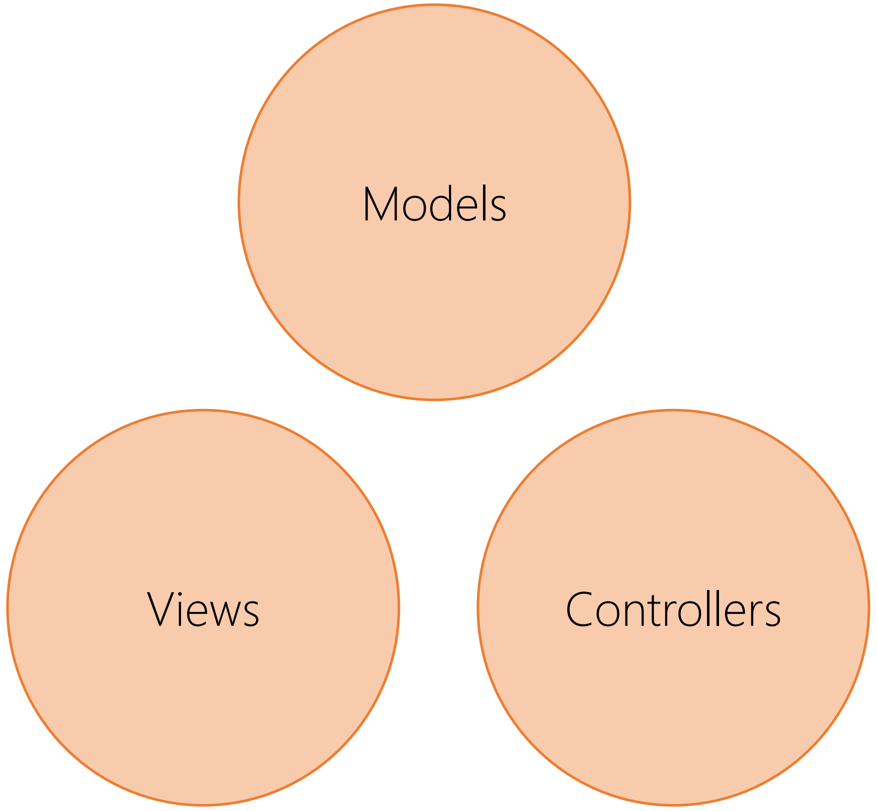
Functional Organization

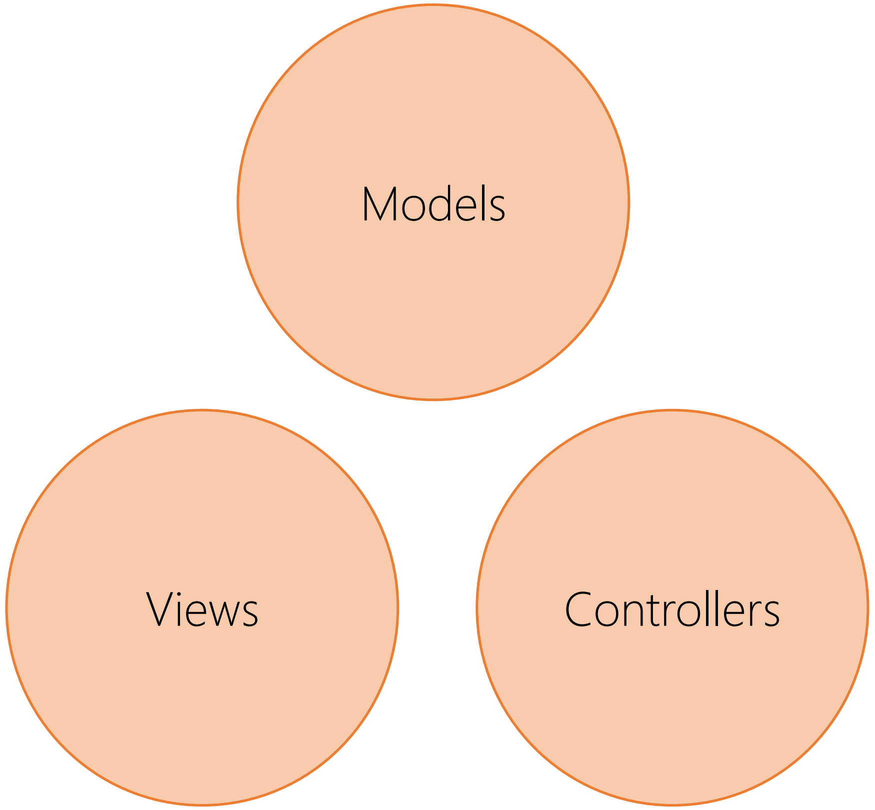
“The architecture should scream
the intent of the system!”

– Uncle Bob

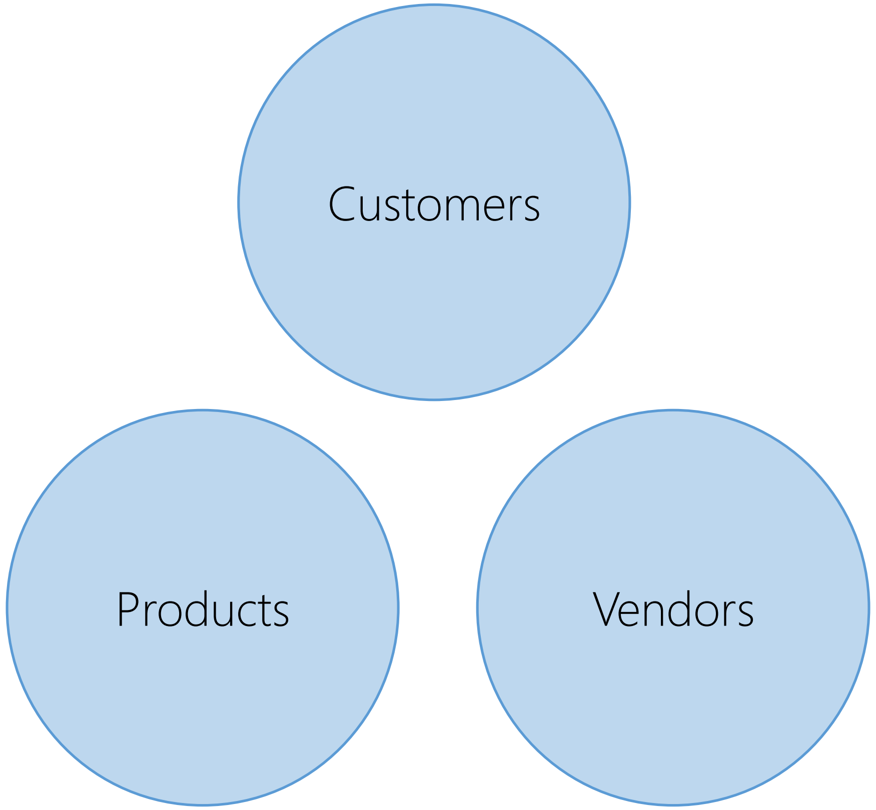

Bedroom
Utility
Dining Room
Living Room
Bedroom
Bath
Kitchen
Entry

Material	Quantity	Cost
Appliances	5	$5,000
Cabinets	10	$2,500
Doors	15	$750
Fixtures	12	$2,400
Floors	9	$4,000
Walls	20	$10,000
Windows	8	$2,500


Models
Views
Controllers



VS



Content

Controllers

Models

Scripts

Views

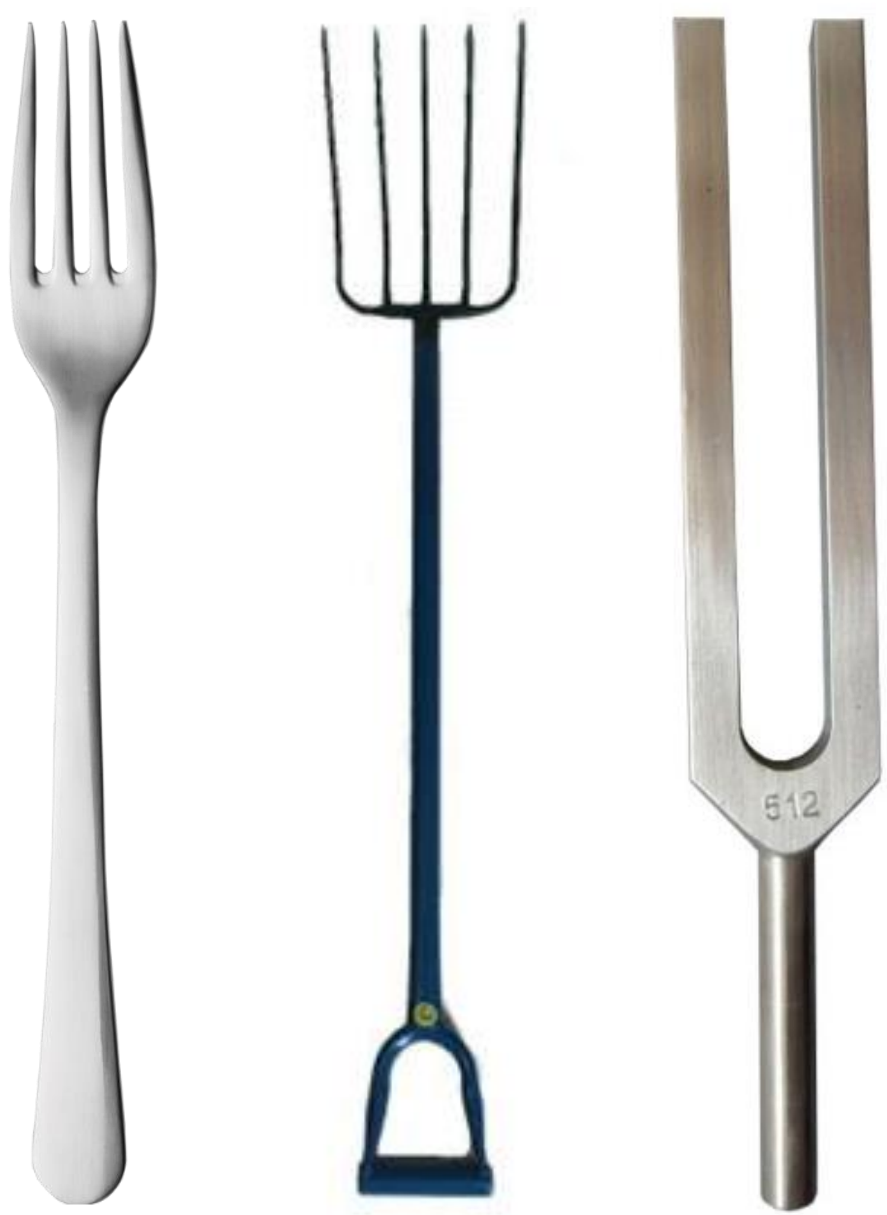
- Content
- Controllers
- Models
- Scripts
- Views

vs

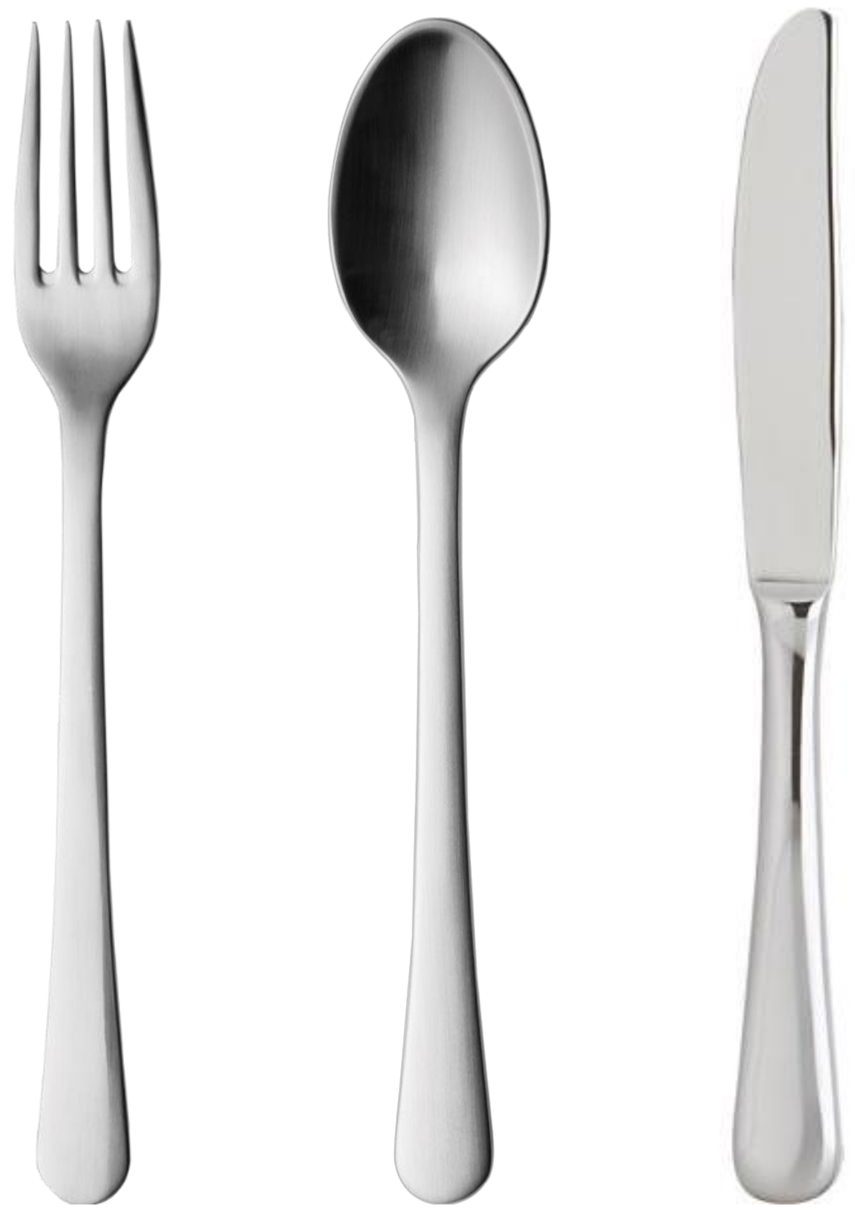
- Customers
- Employees
- Products
- Sales
- Vendors

So what?

512



VS



Why Use Functional Organization

Pros

Spatial locality

Easy to navigate

Avoid vendor lock-in

Why Use Functional Organization

Pros

Spatial locality

Easy to navigate

Avoid vendor lock-in

Cons

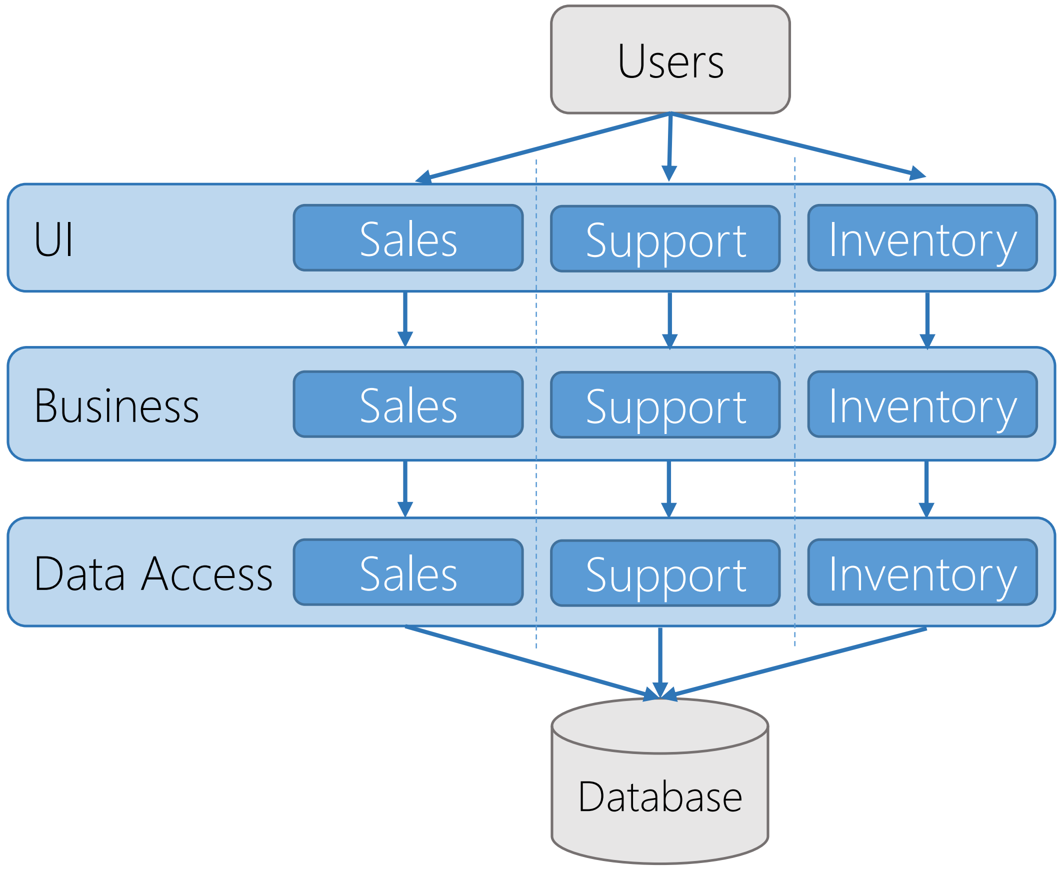
Lose framework conventions

Lose automatic scaffolding

Categorical is easier at first

Microservices

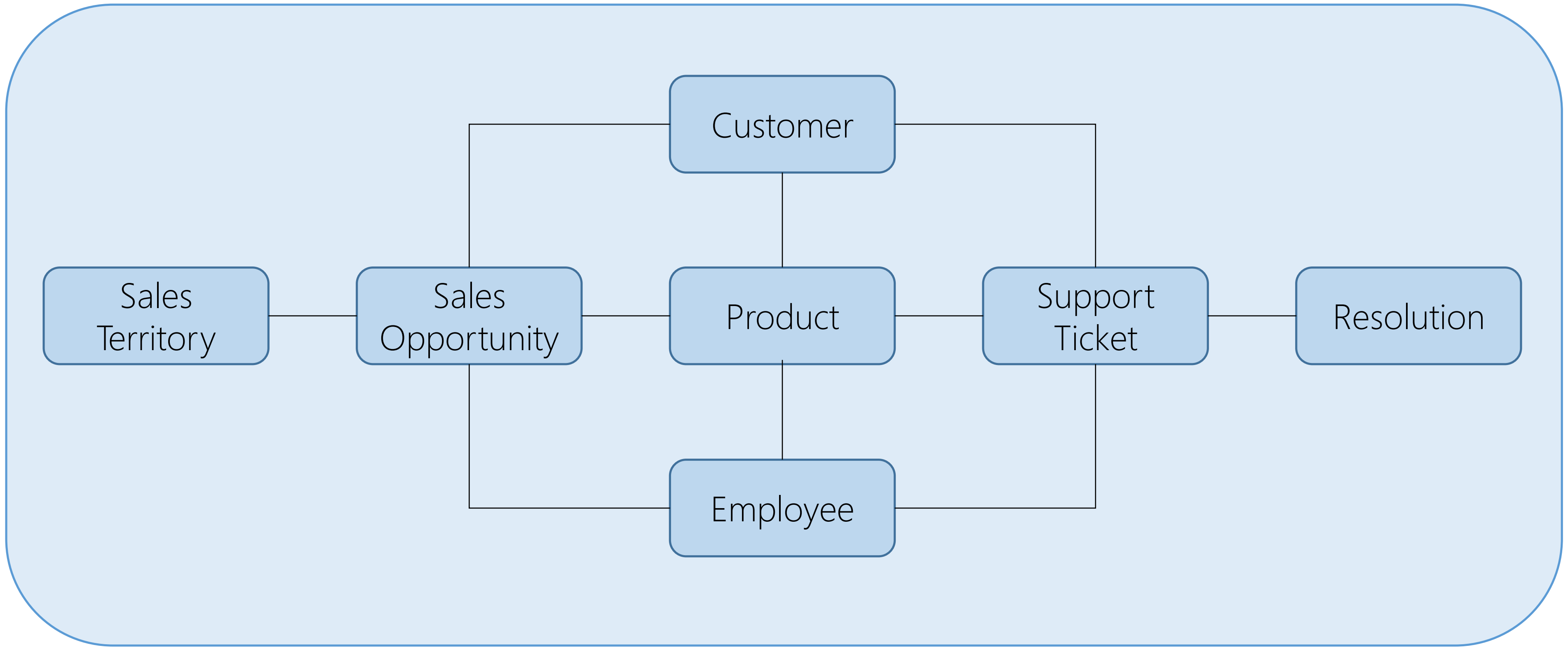
Components


Users
UI
Sales
Support
Inventory
Business
Sales
Support
Inventory
Data Access
Sales
Support
Inventory
Database

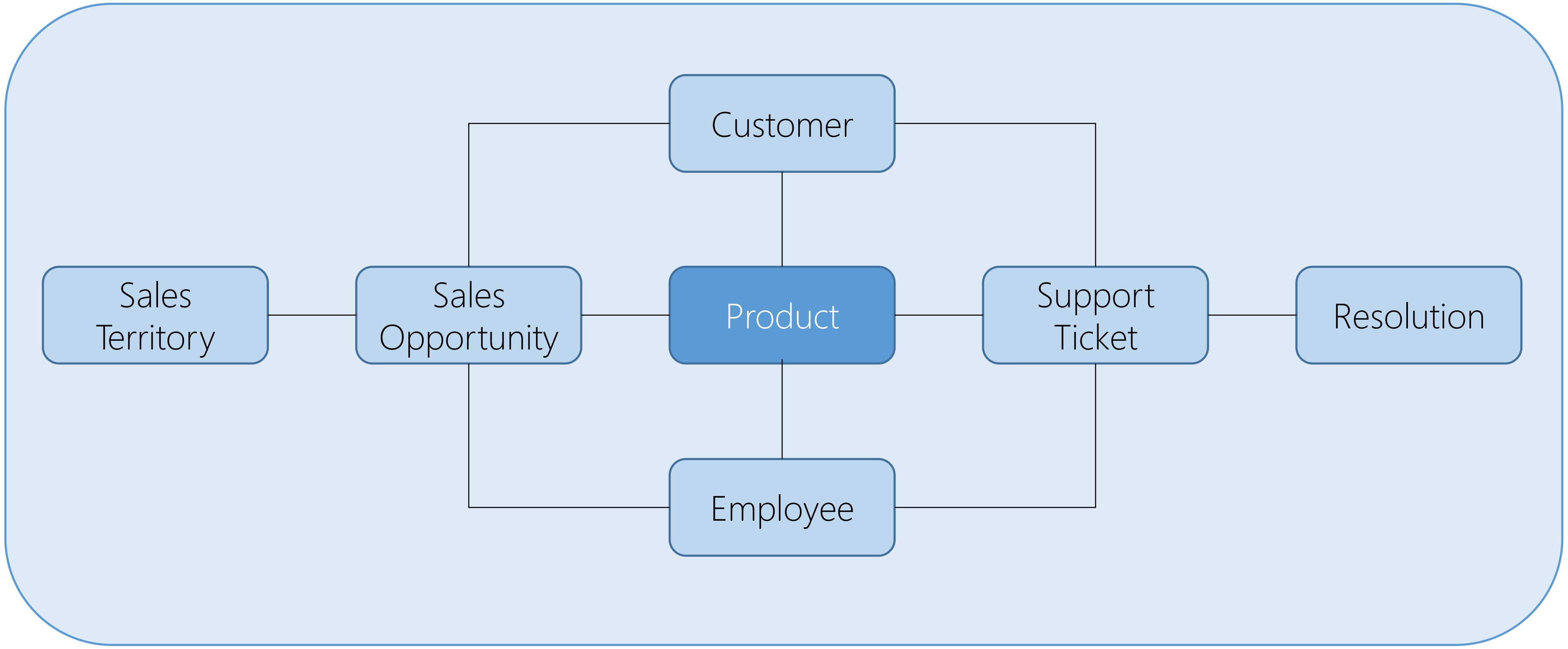
Problem Domain

Sales	Support
Sales Opportunity	Support Ticket
Contact	Customer
Sales Person	Support Person
Product	Product
Sales Territory	Resolution

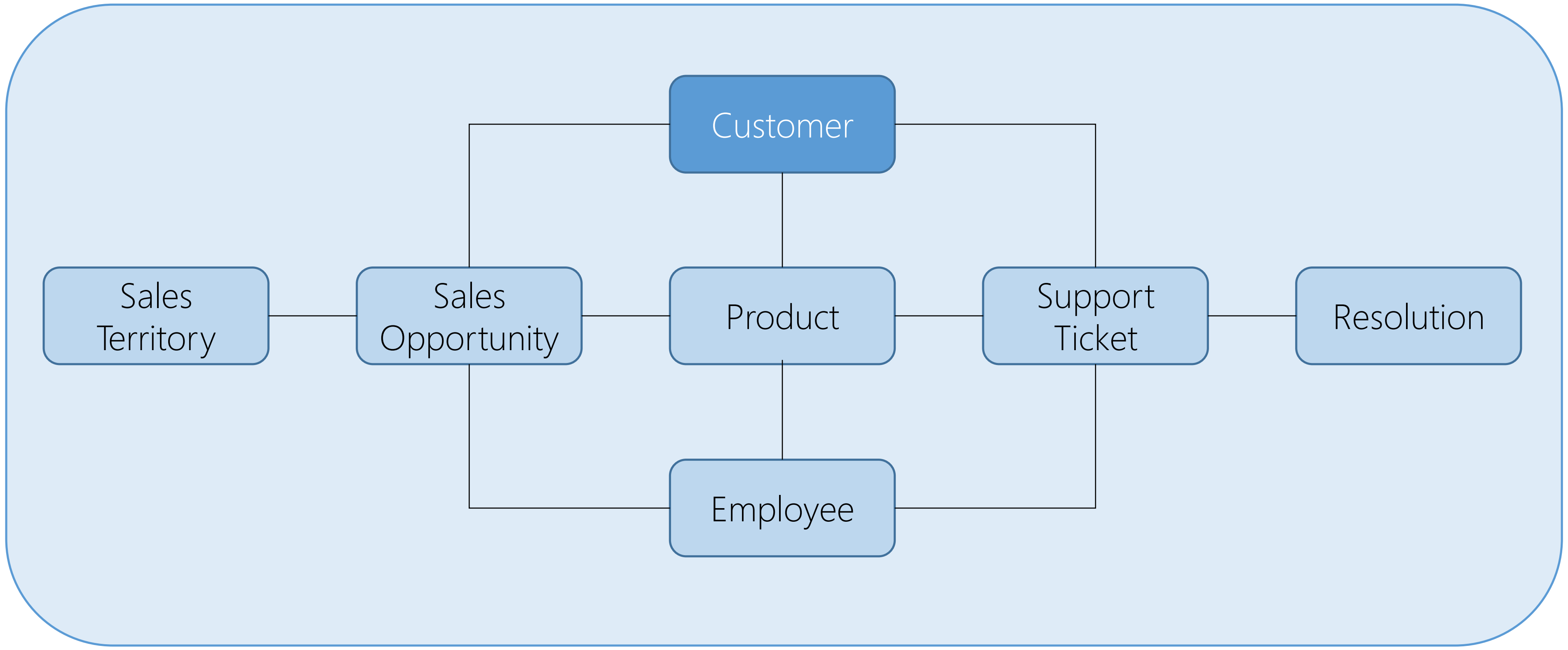
Single Domain Model



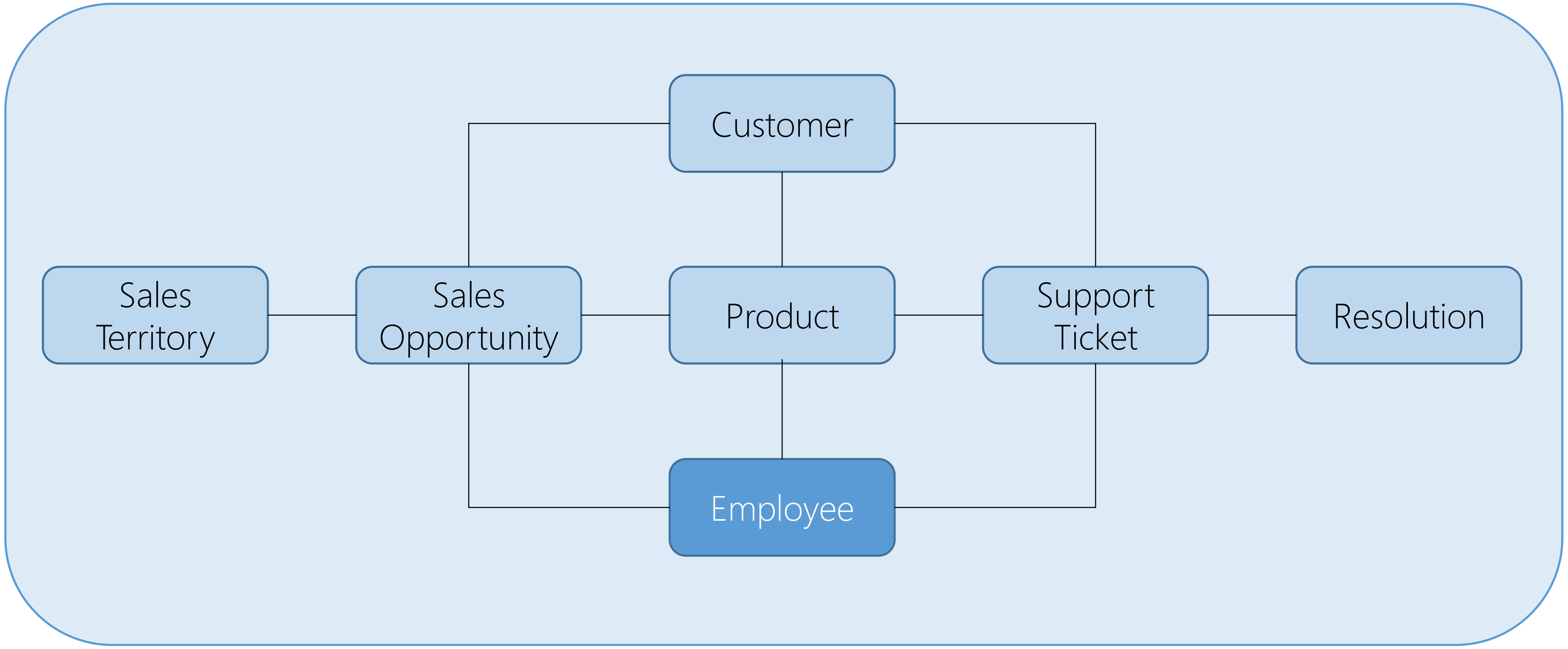
Single Domain Model



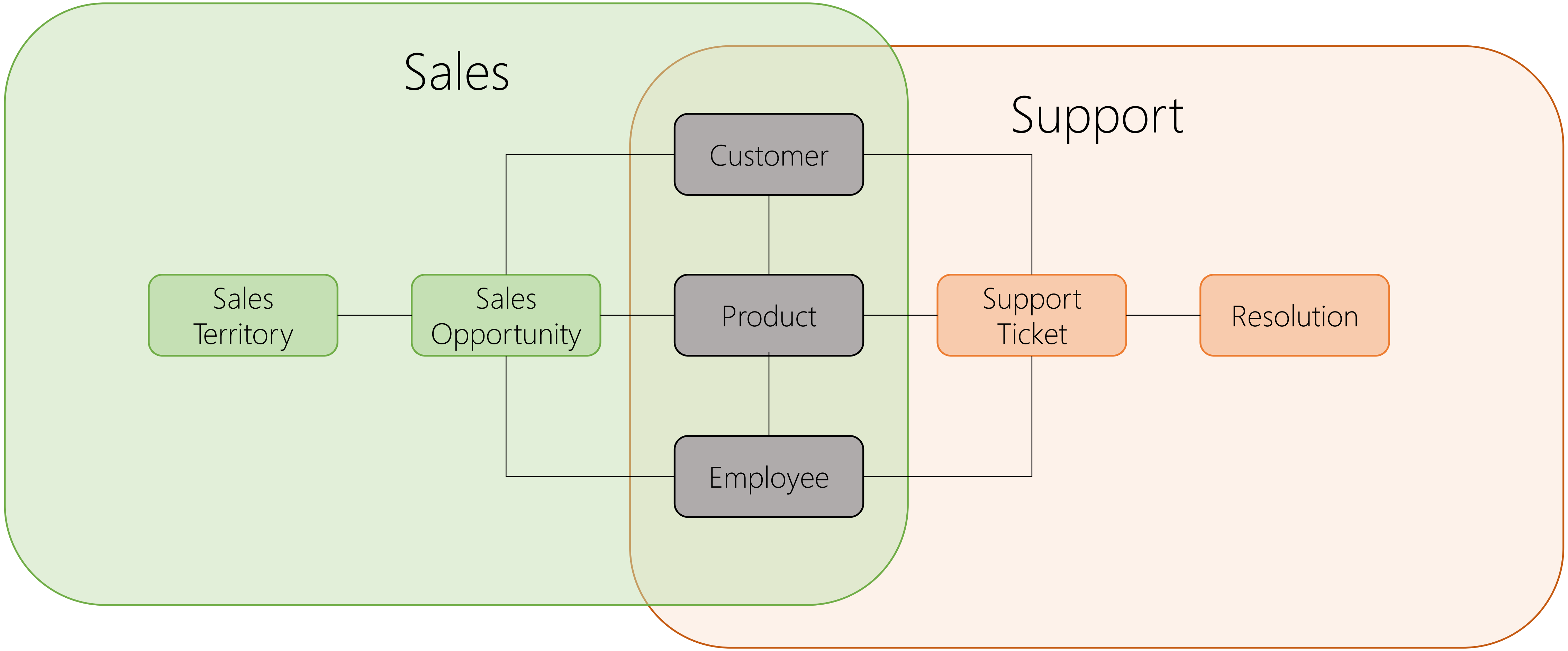
Single Domain Model



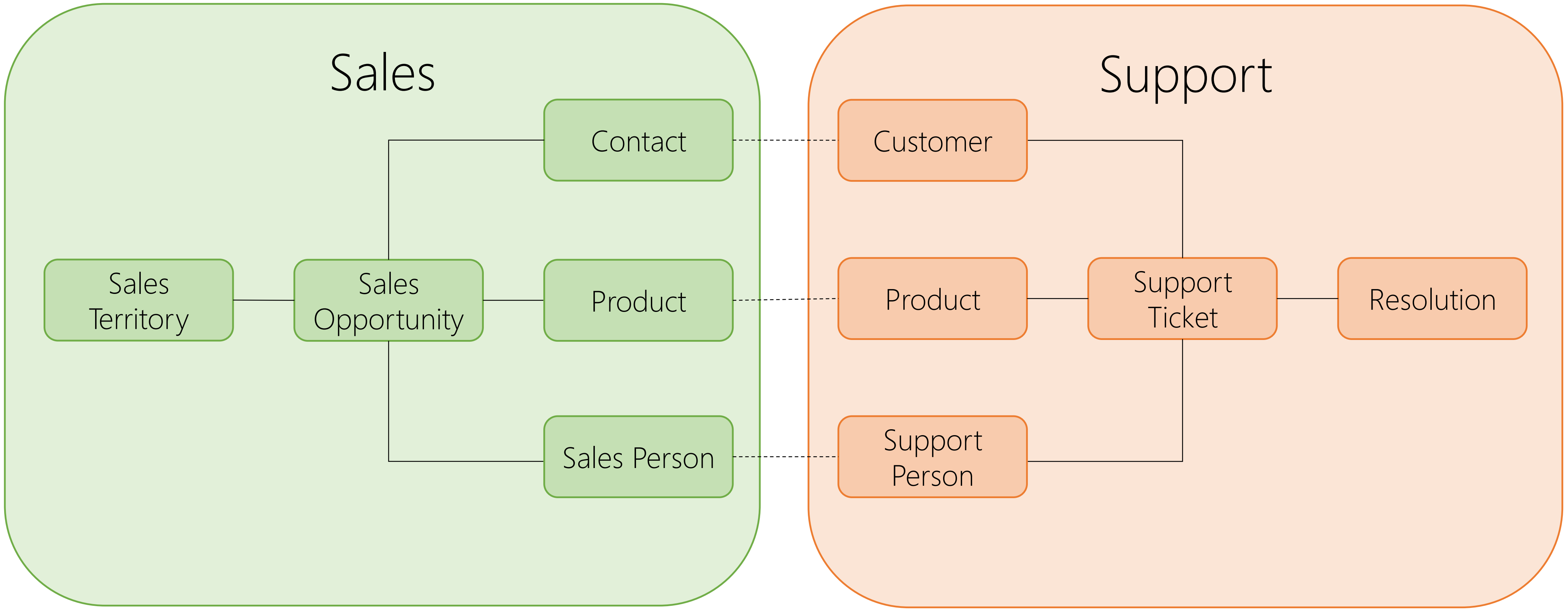
Single Domain Model



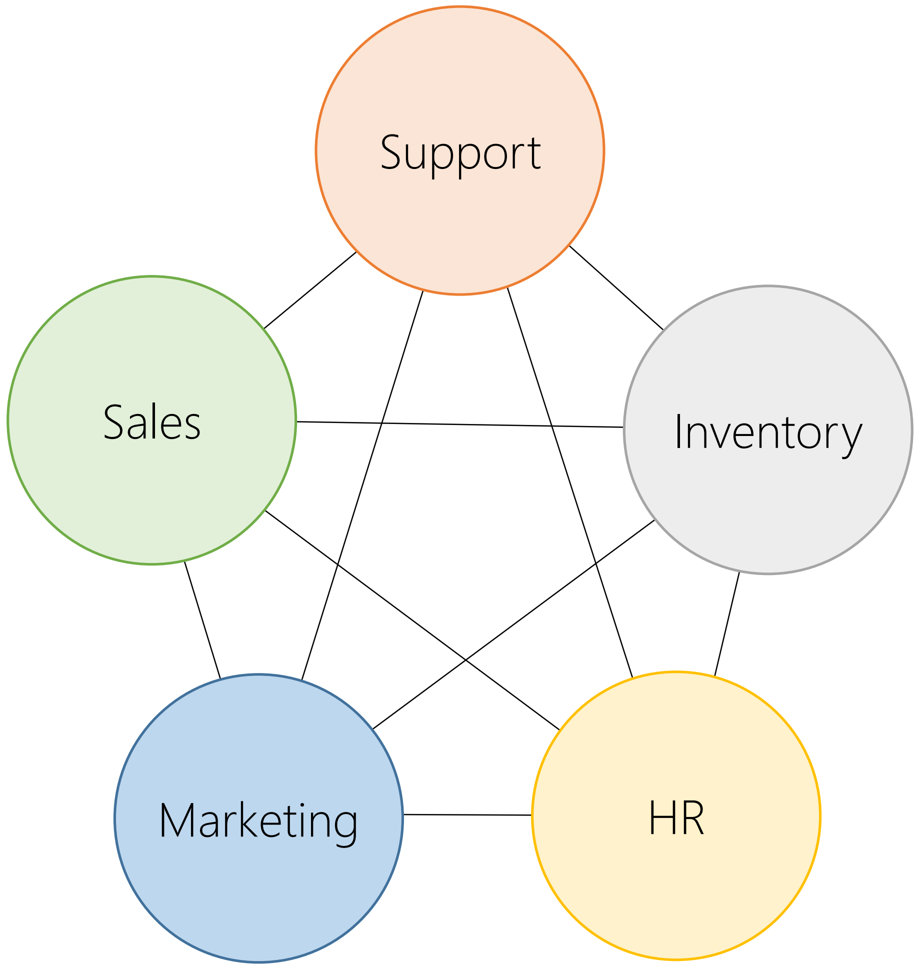
Overlapping Contexts



Bounded Contexts

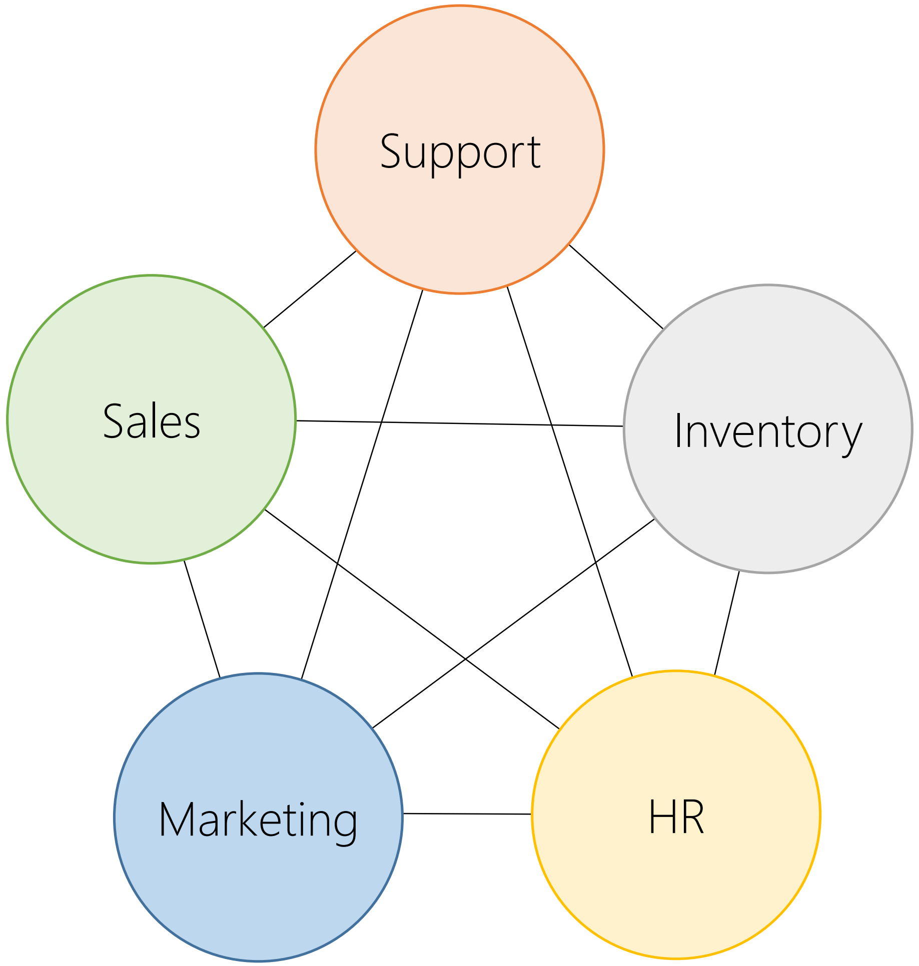


Microservice Architectures



Microservice Architectures

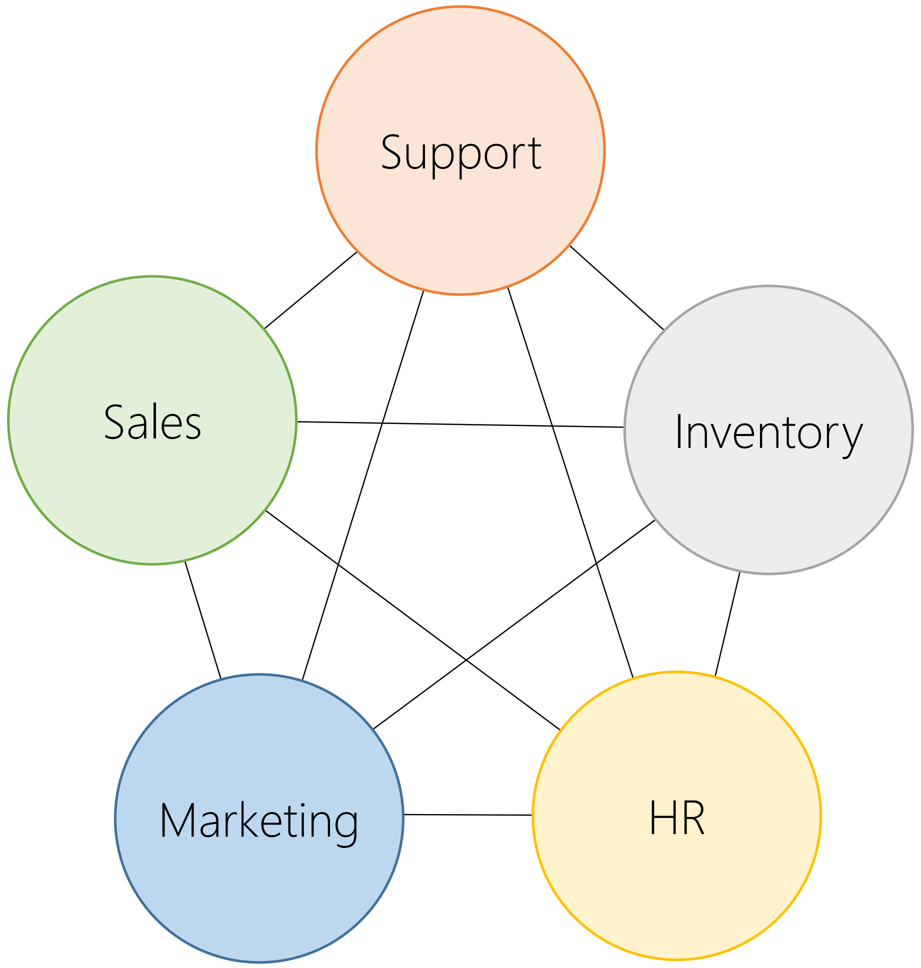
Subdivide system



Microservice Architectures

Subdivide system

Light-weight APIs

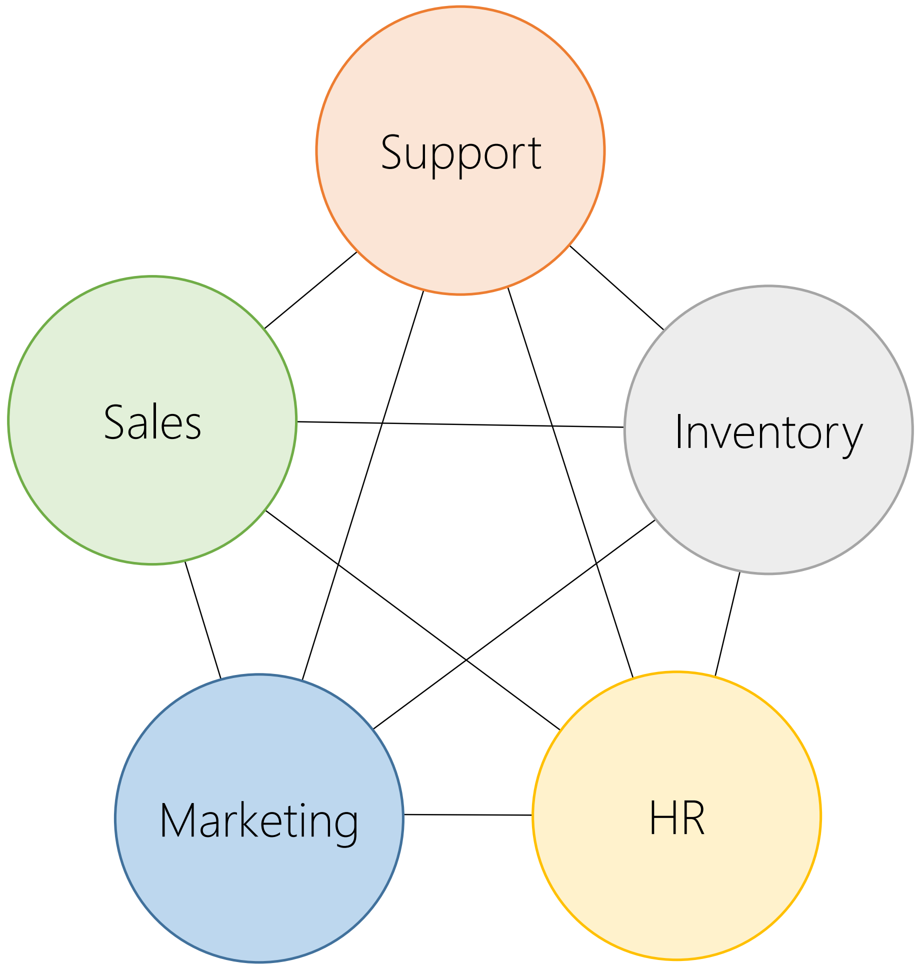


Microservice Architectures

Subdivide system

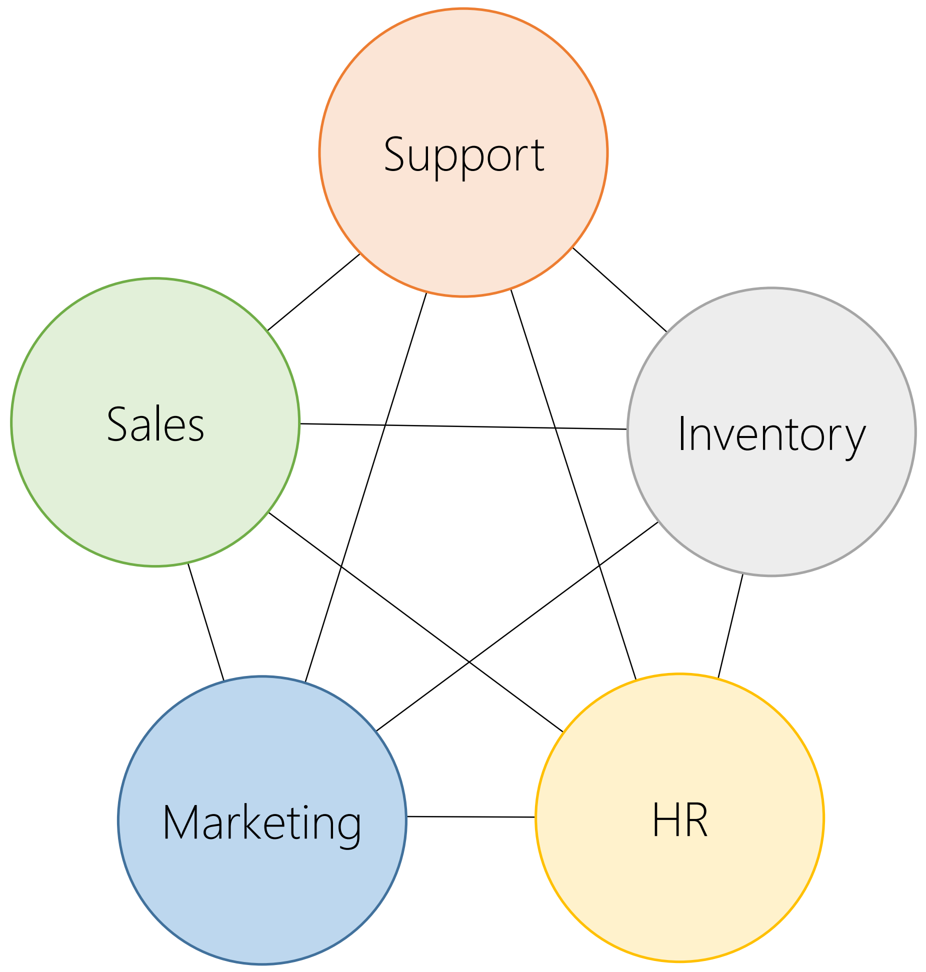
Light-weight APIs

Small teams



Microservice Architectures

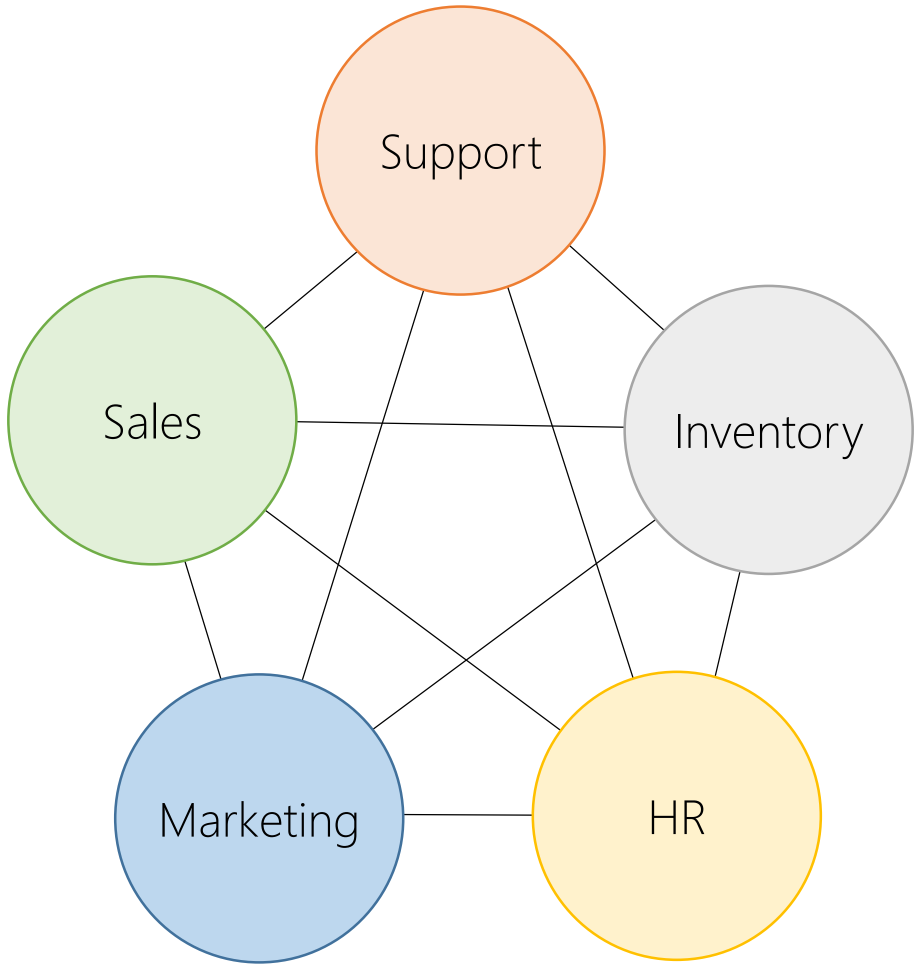
Independent



Microservice Architectures

Independent

Similar to SOA

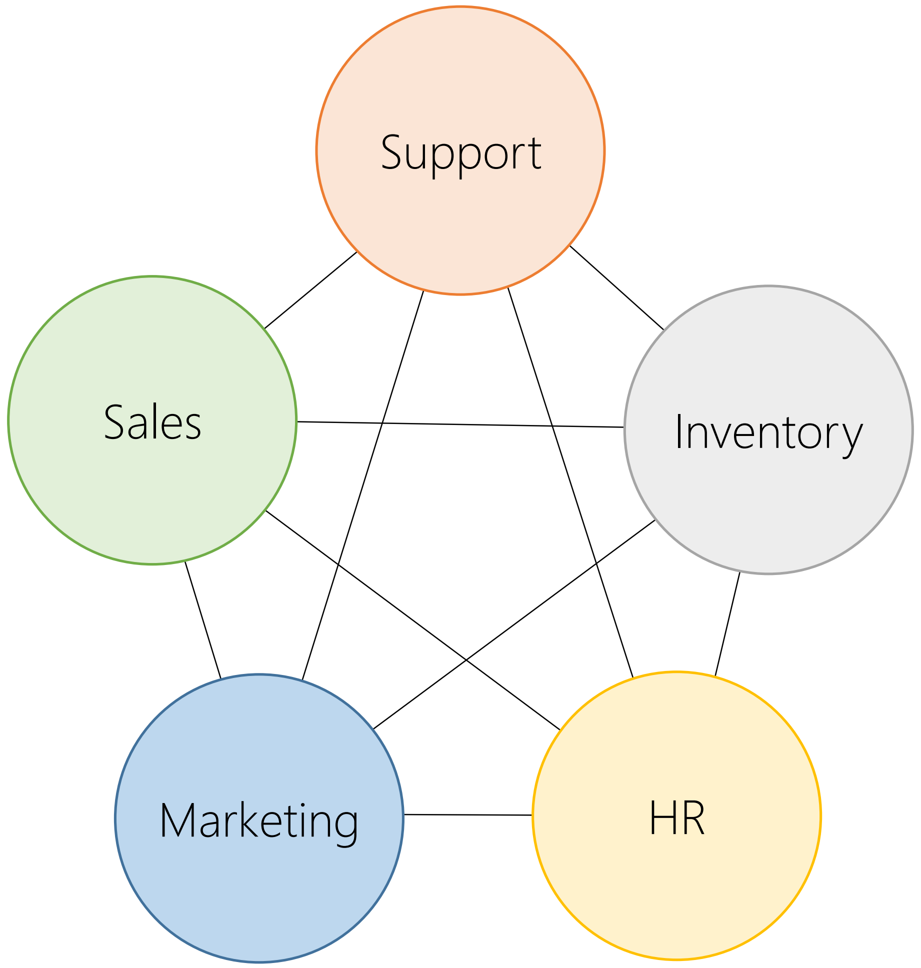


Microservice Architectures

Independent

Similar to SOA

Size matters



Why Use Microservices?

Pros

Less cost for large domains

Smaller teams

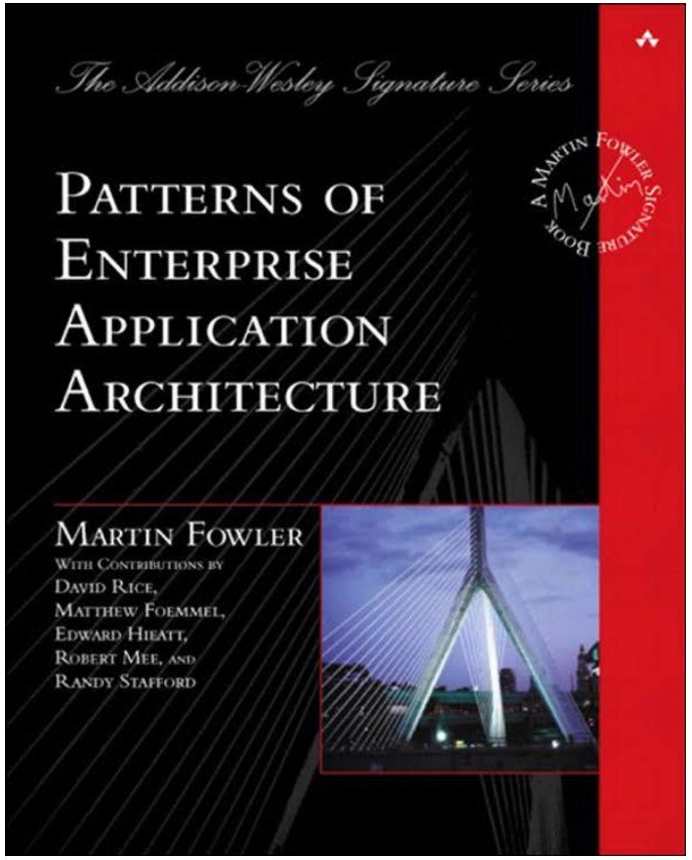
Independence

Why Use Microservices?

Pros	Cons
Less cost for large domains	Only for large domains
Smaller teams	Higher up-front cost
Independence	Distributed system costs

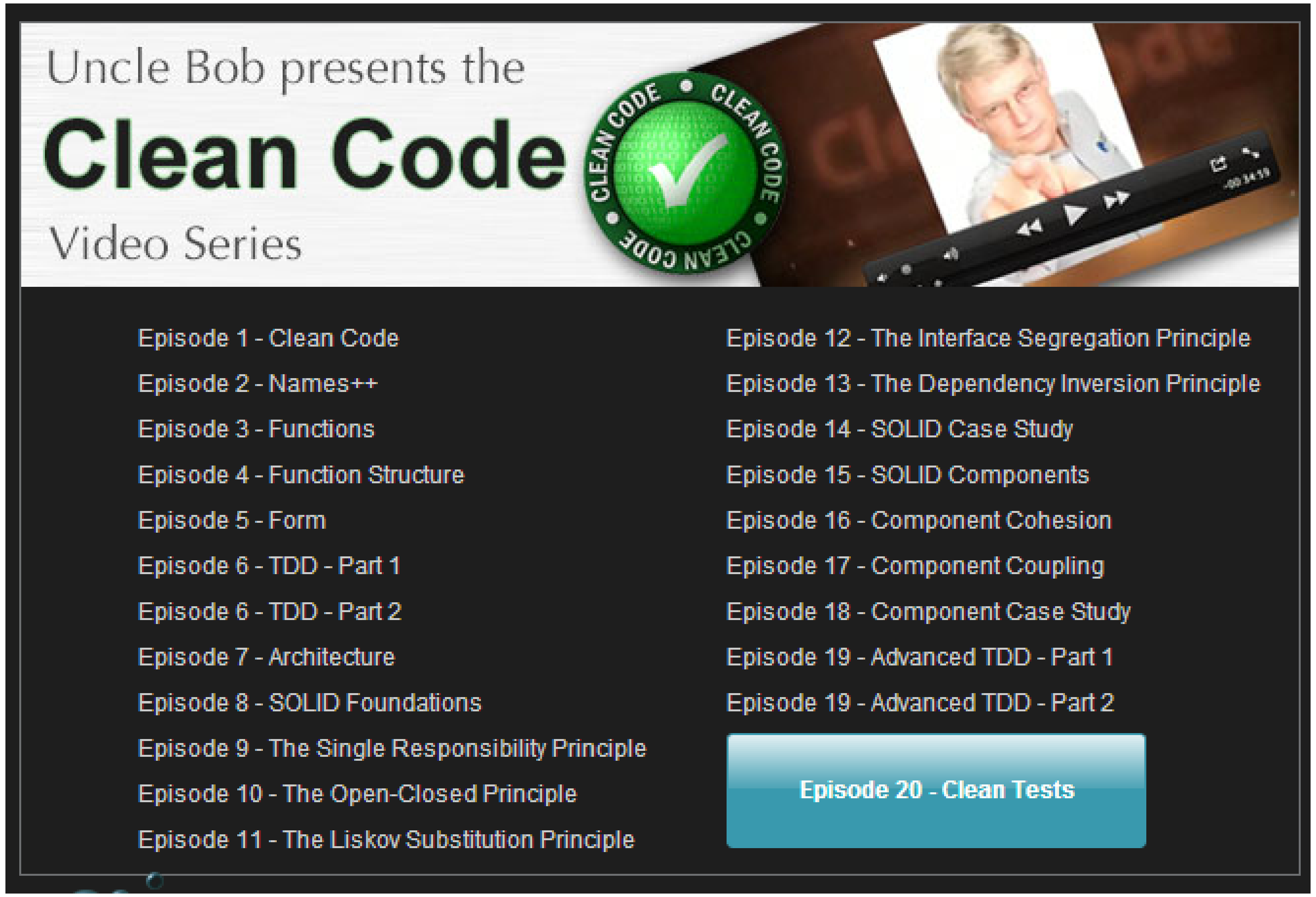
Code Demo

Where to Go Next?


The Addison-Wesley Signature Series
Patterns of Enterprise Application Architecture
A Martin Fowler Signature Book
Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford

Martin Fowler

Where to Go Next?



Uncle Bob presents the

Clean Code

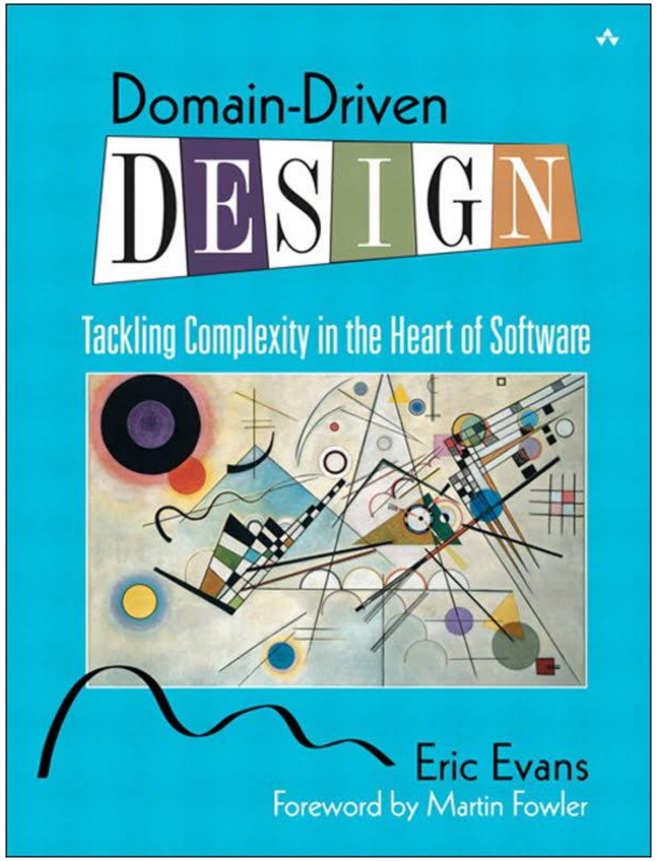
Video Series

Episode 1 - Clean Code
Episode 2 - Names++
Episode 3 - Functions
Episode 4 - Function Structure
Episode 5 - Form
Episode 6 - TDD - Part 1
Episode 6 - TDD - Part 2
Episode 7 - Architecture
Episode 8 - SOLID Foundations
Episode 9 - The Single Responsibility Principle
Episode 10 - The Open-Closed Principle
Episode 11 - The Liskov Substitution Principle
Episode 12 - The Interface Segregation Principle
Episode 13 - The Dependency Inversion Principle
Episode 14 - SOLID Case Study
Episode 15 - SOLID Components
Episode 16 - Component Cohesion
Episode 17 - Component Coupling
Episode 18 - Component Case Study
Episode 19 - Advanced TDD - Part 1
Episode 19 - Advanced TDD - Part 2
Episode 20 - Clean Tests

http://cleancoders.com/

Robert C. Martin

Where to Go Next?


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Eric Evans

Where to Go Next?

Greg Young

Udi Dahan

Where to Go Next?

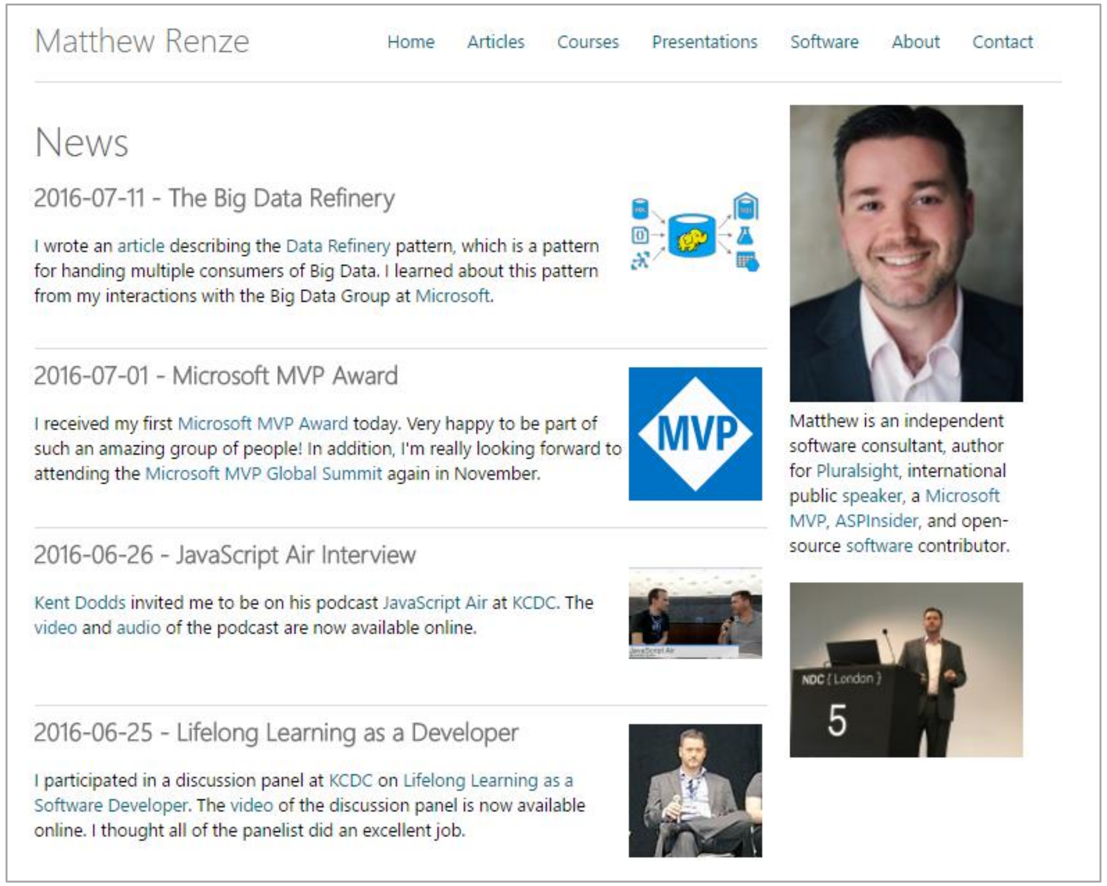
Articles

Courses

Presentations

Source Code

Videos



Matthew Renze

Home Articles Courses Presentations Software About Contact

News

2016-07-11 - The Big Data Refinery

I wrote an article describing the Data Refinery pattern, which is a pattern for handing multiple consumers of Big Data. I learned about this pattern from my interactions with the Big Data Group at Microsoft.

2016-07-01 - Microsoft MVP Award

I received my first Microsoft MVP Award today. Very happy to be part of such an amazing group of people! In addition, I'm really looking forward to attending the Microsoft MVP Global Summit again in November.

2016-06-26 - JavaScript Air Interview

Kent Dodds invited me to be on his podcast JavaScript Air at KCDC. The video and audio of the podcast are now available online.

2016-06-25 - Lifelong Learning as a Developer

I participated in a discussion panel at KCDC on Lifelong Learning as a Software Developer. The video of the discussion panel is now available online. I thought all of the panelist did an excellent job.

Matthew is an independent software consultant, author for Pluralsight, international public speaker, a Microsoft MVP, ASPInsider, and open-source software contributor.

www.matthewrenze.com

Clean Architecture:
Patterns, Practices, and Principles

INTRODUCTION

Matthew Renze
SOFTWARE CONSULTANT

@matthewrenze www.matthewrenze.com

www.pluralsight.com/authors/matthew-renze

Conclusion

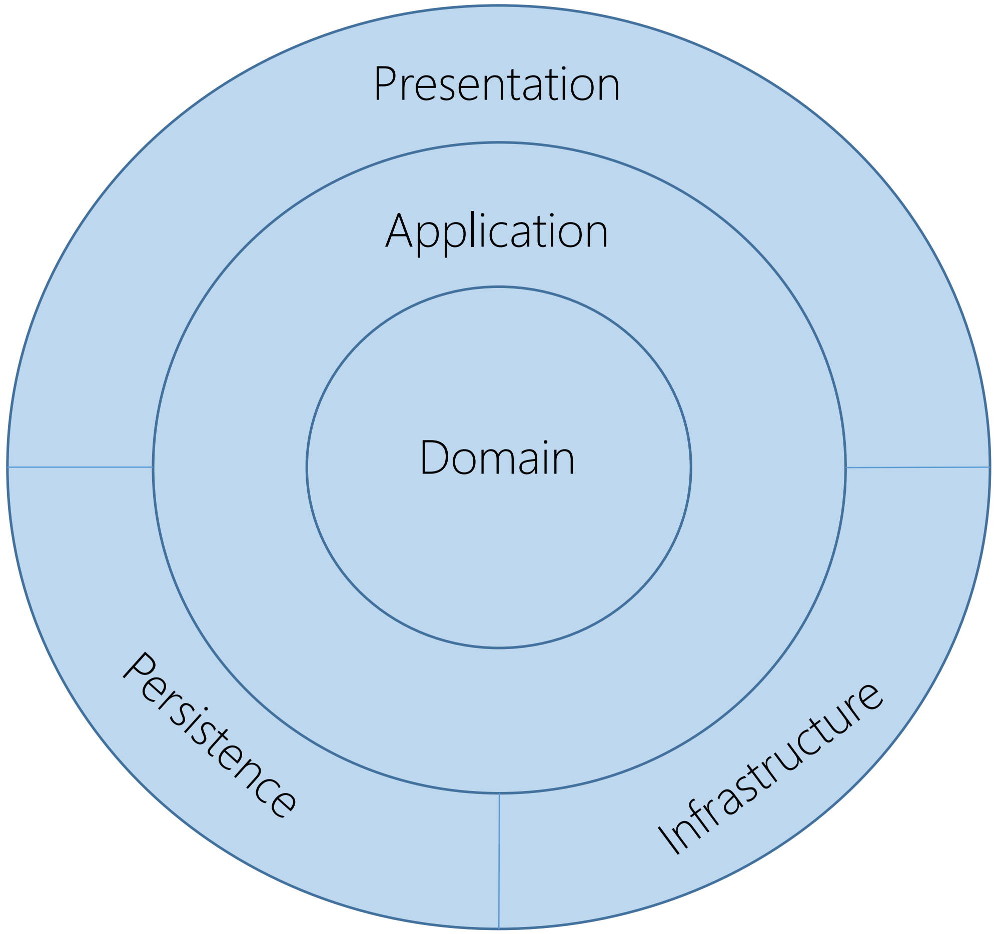
Summary

Focus on the inhabitants

Summary

Focus on the inhabitants

Domain-centric Architecture

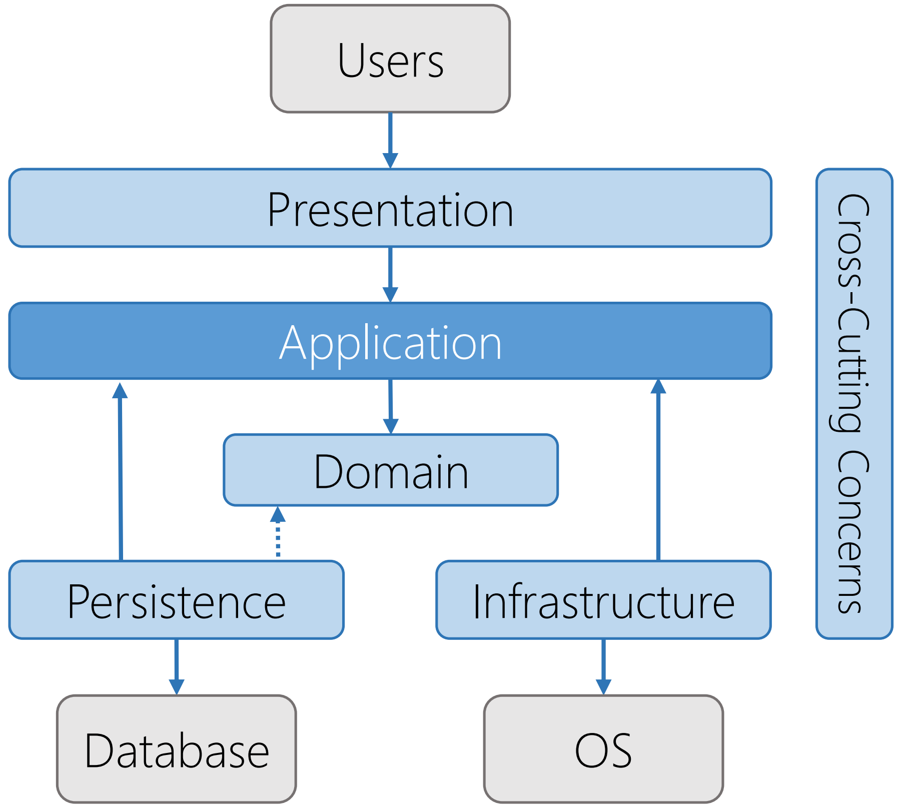


Summary

Focus on the inhabitants

Domain-centric Architecture

Application Layer



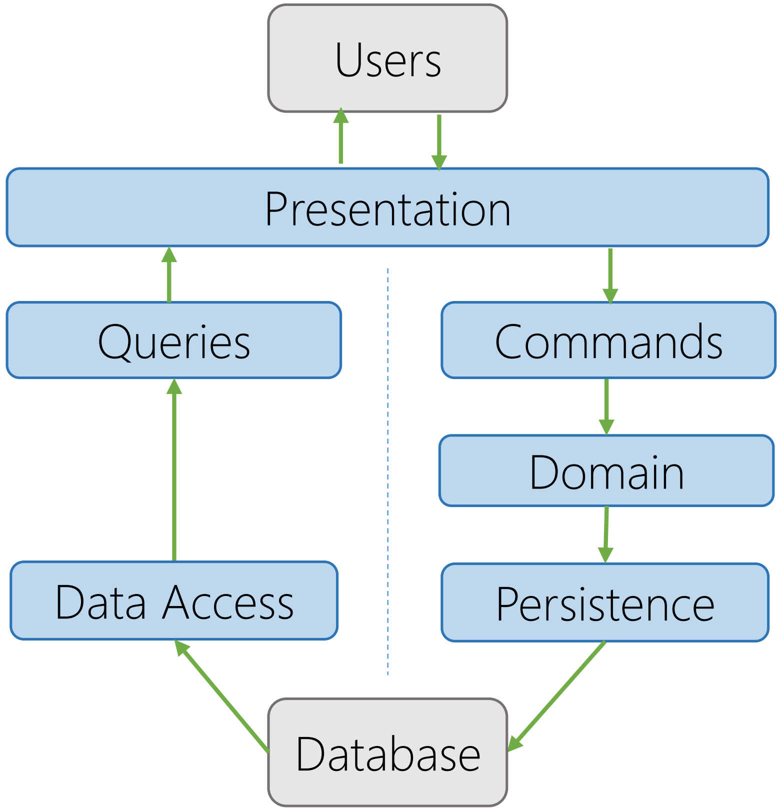
Summary

Focus on the inhabitants

Domain-centric Architecture

Application Layer

Commands and Queries



Summary

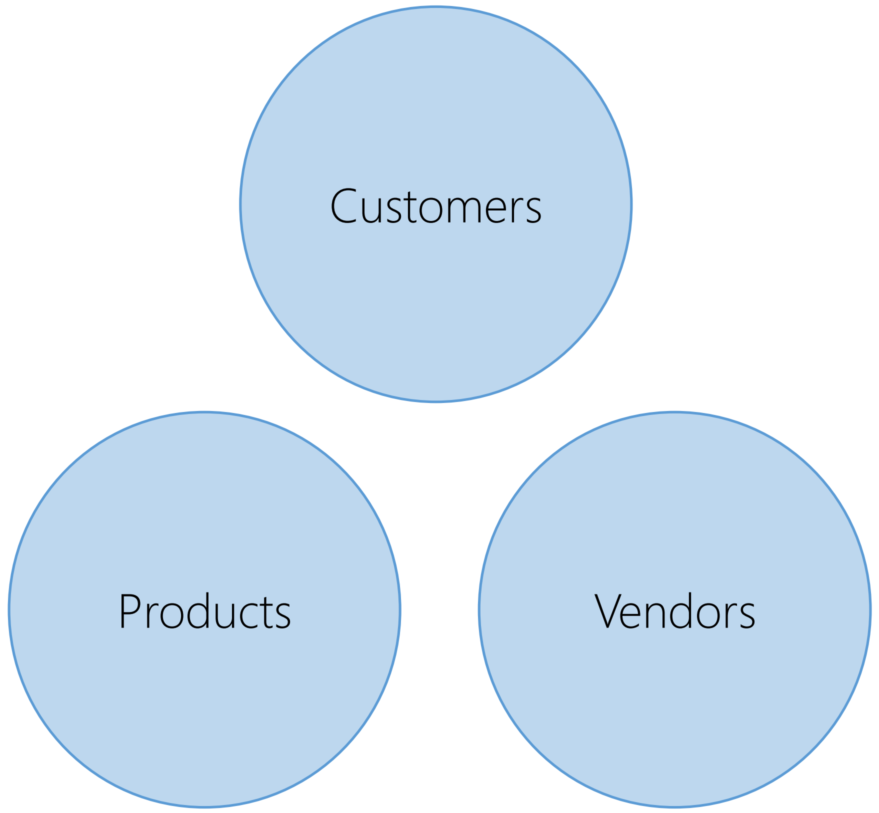
Focus on the inhabitants

Domain-centric Architecture

Application Layer

Commands and Queries

Functional Cohesion



Summary

Focus on the inhabitants

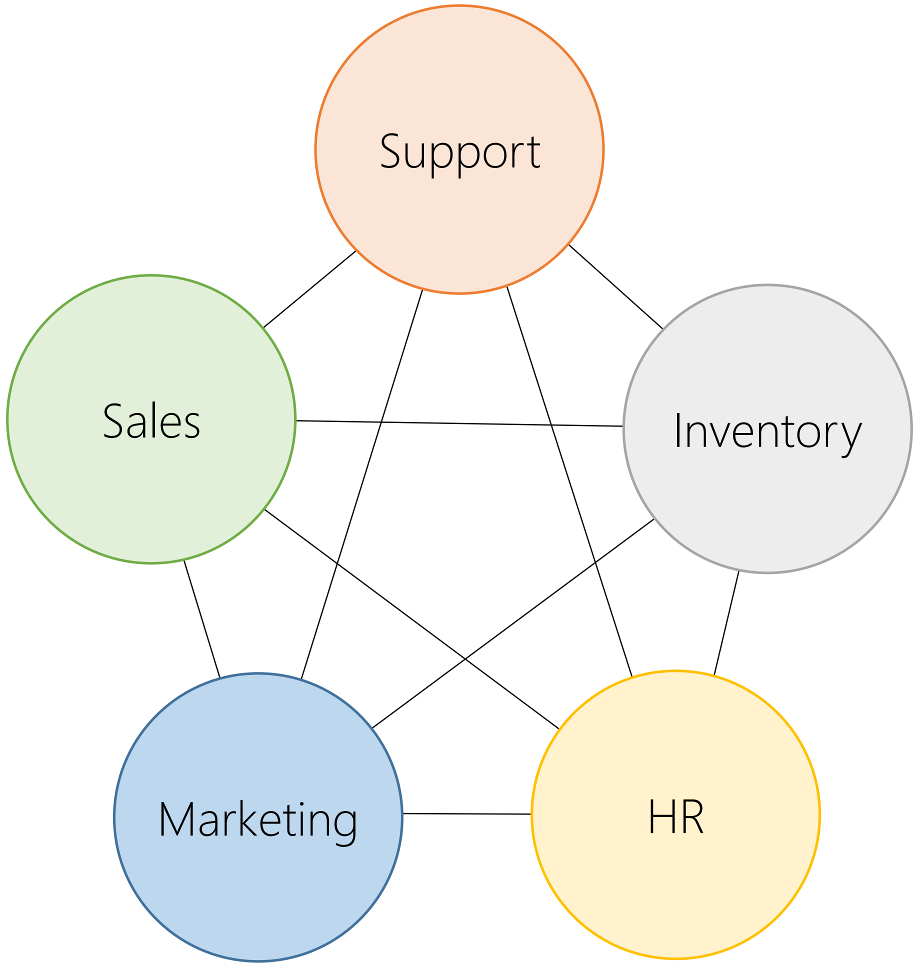
Domain-centric Architecture

Application Layer

Commands and Queries

Functional Cohesion

Bounded Contexts



Feedback

Very important to me!

One thing you liked?

One thing I could improve?

Contact Info

Matthew Renze

Data Science Consultant

Renze Consulting

Twitter: @matthewrenze

Email: matthew@matthewrenze.com

Website: www.matthewrenze.com

Thank You! :)