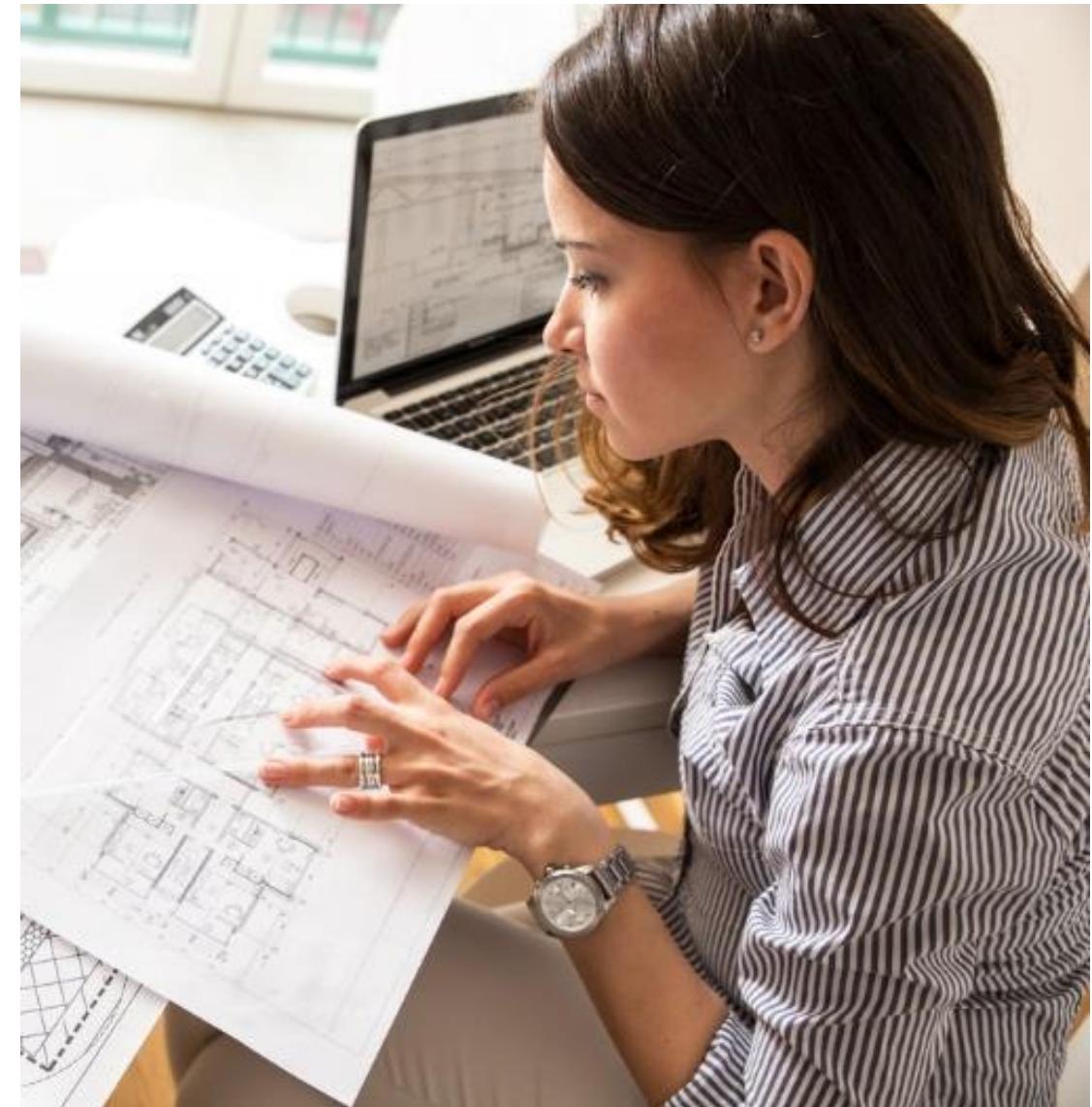


Clean Architecture

Patterns, Practices, and Principles

@MatthewRenze

#CodeMash













Overview

1. Clean Architecture

Overview

1. Clean Architecture
2. Domain-Centric Architecture

Overview

1. Clean Architecture
2. Domain-Centric Architecture
3. Application Layer

Overview

1. Clean Architecture
2. Domain-Centric Architecture
3. Application Layer
4. Commands and Queries

Overview

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

Overview

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

Focus

Enterprise Architecture

Focus

Enterprise Architecture
Modern equivalent of 3-Layer

Focus

Enterprise Architecture

Modern equivalent of 3-Layer

Generally applicable

Focus

Enterprise Architecture

Modern equivalent of 3-Layer

Generally applicable

6 Key Points

Focus

Enterprise Architecture

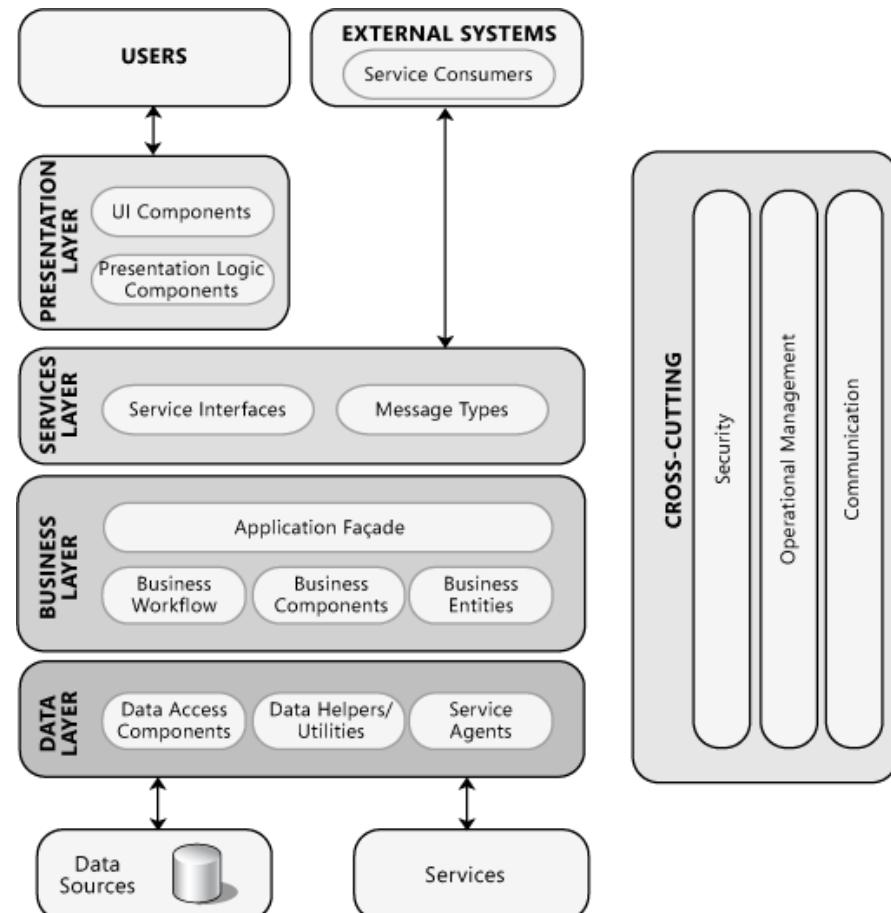
Modern equivalent of 3-Layer

Generally applicable

6 Key Points

Q & A

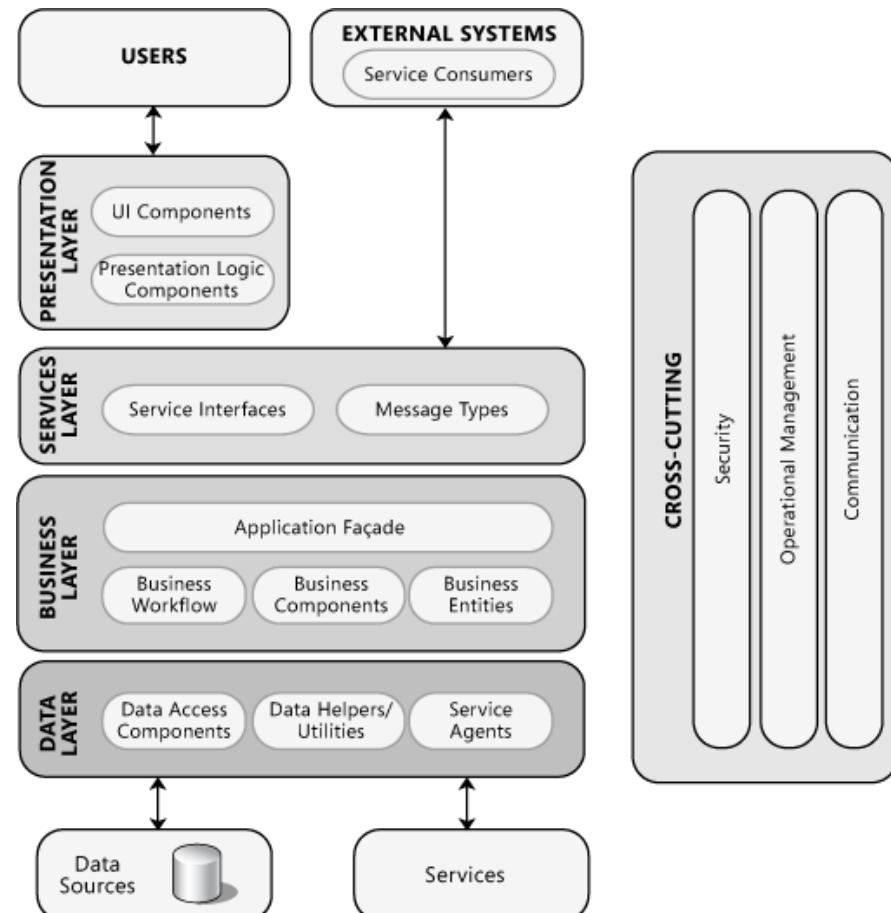
What is Software Architecture?



Source: [http://msdn.microsoft.com/
en-us/library/ff650706.aspx](http://msdn.microsoft.com/en-us/library/ff650706.aspx)

What is Software Architecture?

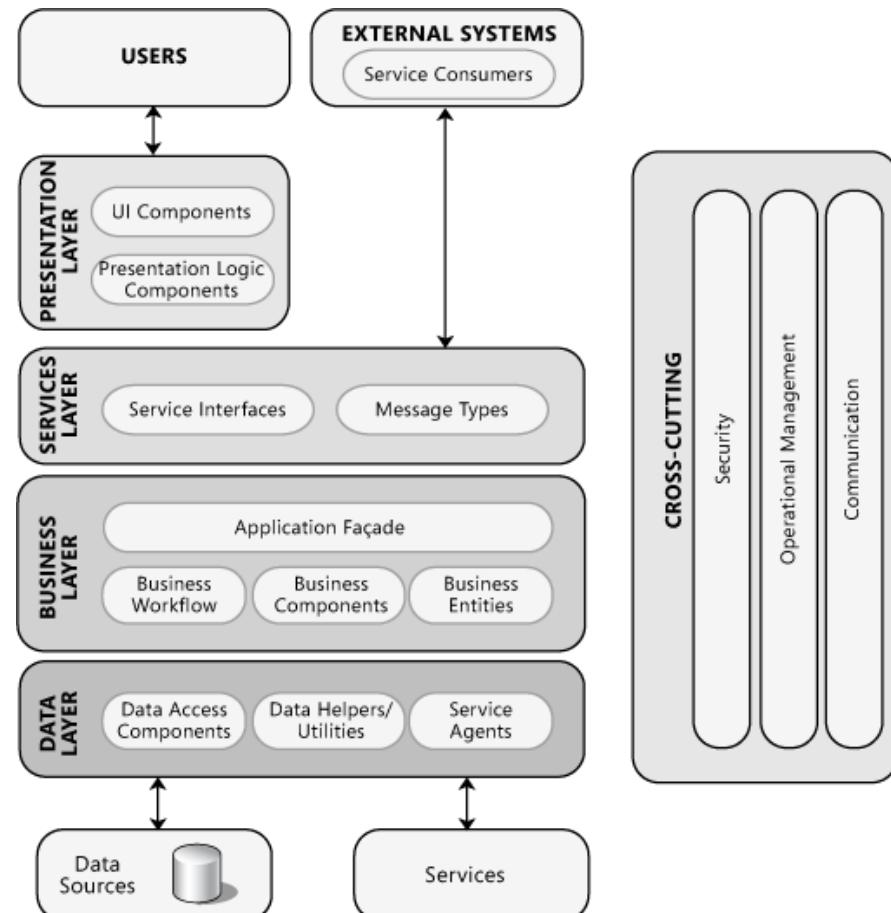
High-level



Source: [http://msdn.microsoft.com/
en-us/library/ff650706.aspx](http://msdn.microsoft.com/en-us/library/ff650706.aspx)

What is Software Architecture?

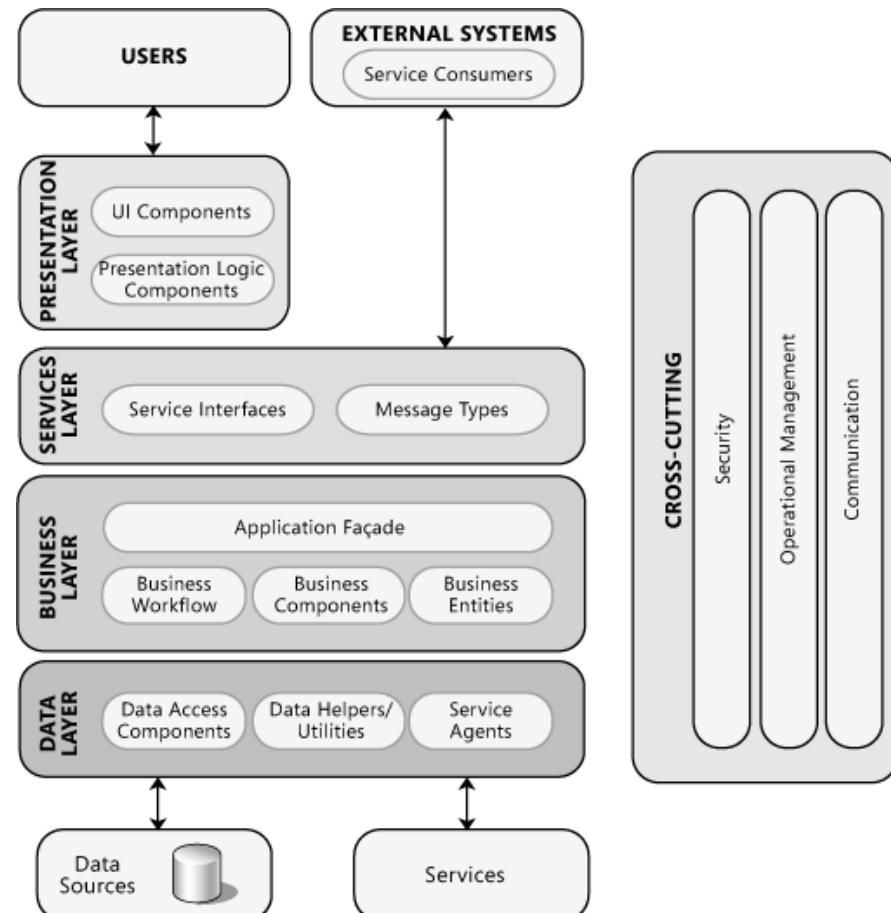
High-level
Structure



Source: [http://msdn.microsoft.com/
en-us/library/ff650706.aspx](http://msdn.microsoft.com/en-us/library/ff650706.aspx)

What is Software Architecture?

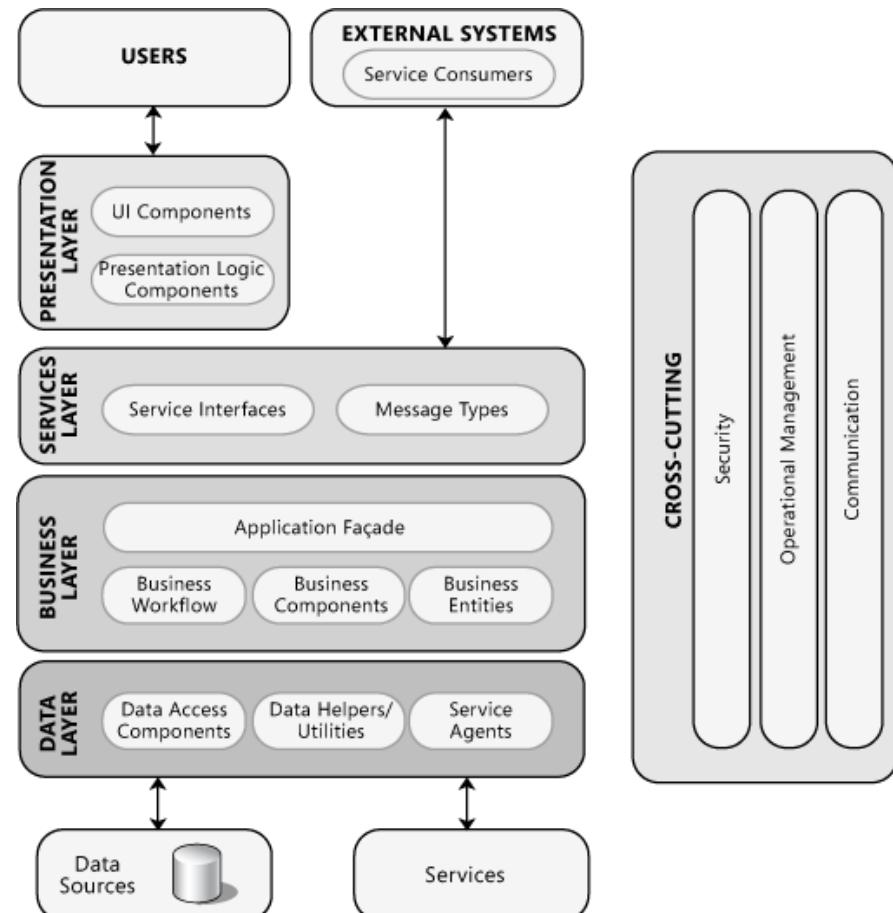
High-level
Structure
Layers



Source: [http://msdn.microsoft.com/
en-us/library/ff650706.aspx](http://msdn.microsoft.com/en-us/library/ff650706.aspx)

What is Software Architecture?

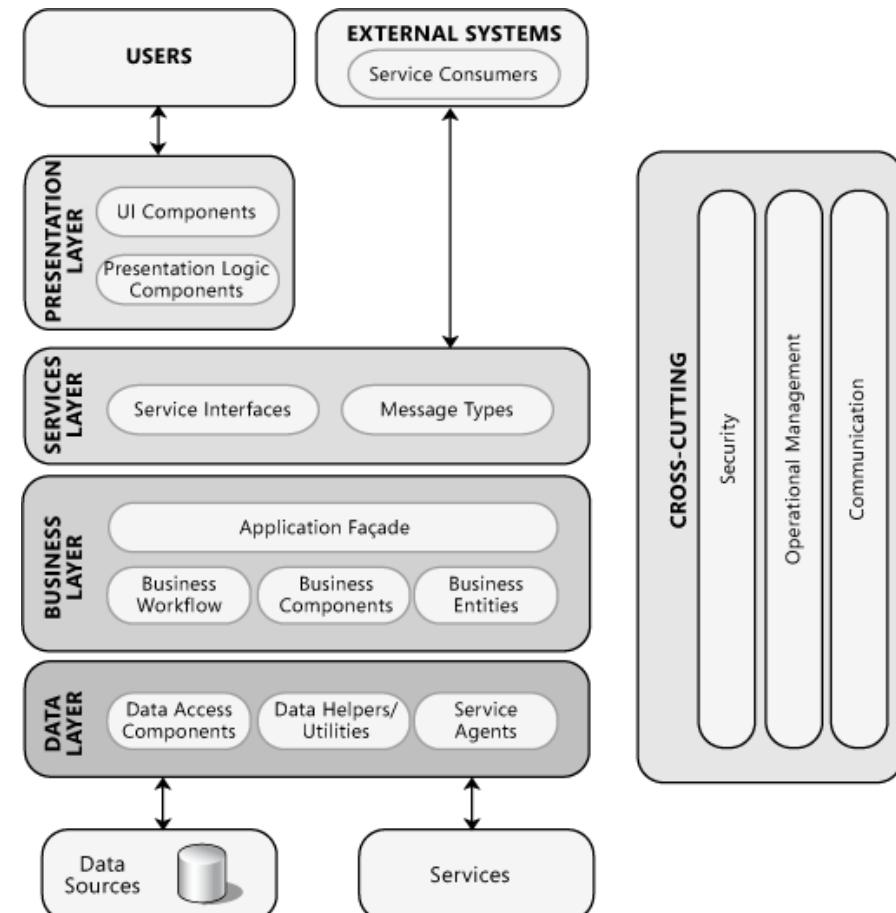
High-level
Structure
Layers
Components



Source: [http://msdn.microsoft.com/
en-us/library/ff650706.aspx](http://msdn.microsoft.com/en-us/library/ff650706.aspx)

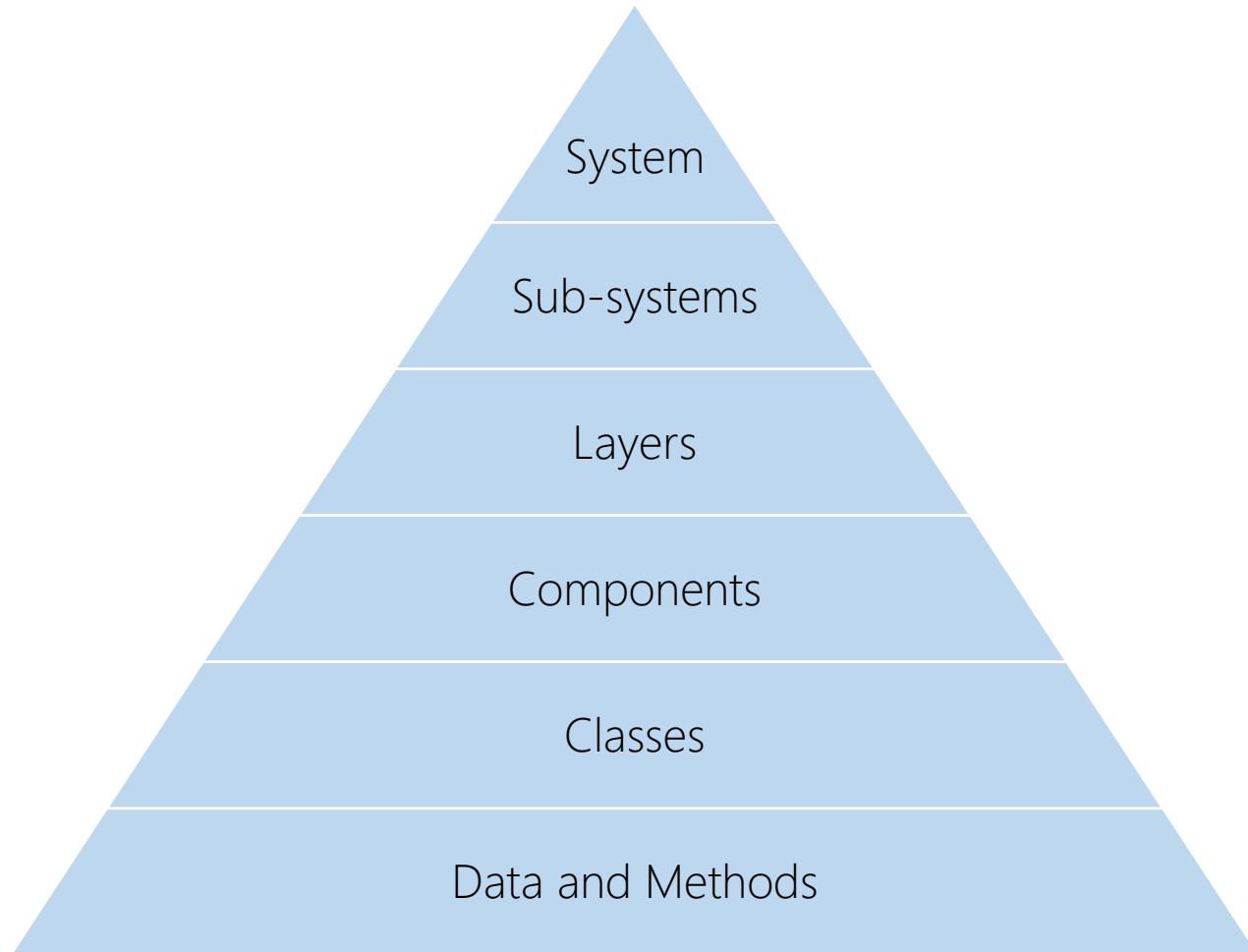
What is Software Architecture?

High-level
Structure
Layers
Components
Relationships

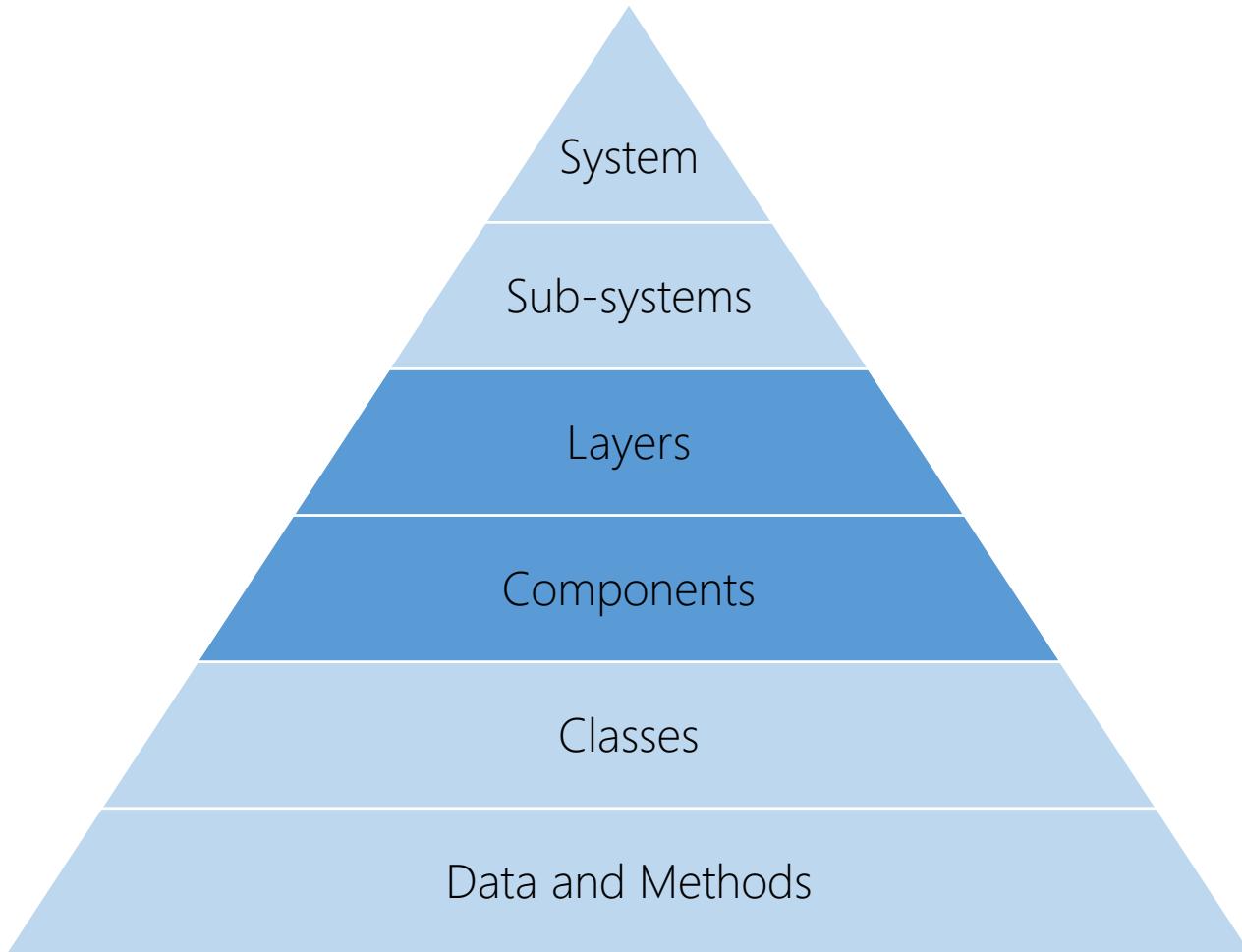


Source: [http://msdn.microsoft.com/
en-us/library/ff650706.aspx](http://msdn.microsoft.com/en-us/library/ff650706.aspx)

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



Why Is Clean Architecture Important?

Cost/benefit

Minimize cost to maintain



Why Is Clean Architecture Important?

Cost/benefit

Minimize cost to maintain

Maximize business value



Why Is Clean Architecture Important?

Cost/benefit

Minimize cost to maintain

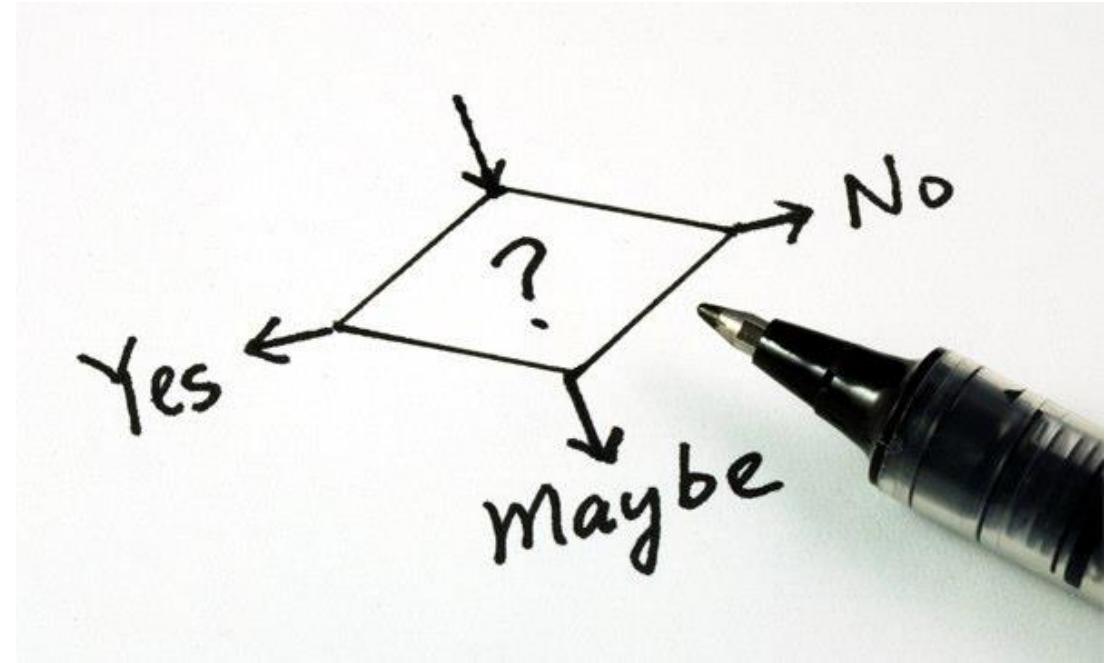
Maximize business value

Maximize total ROI



Decisions, Decisions, Decisions...

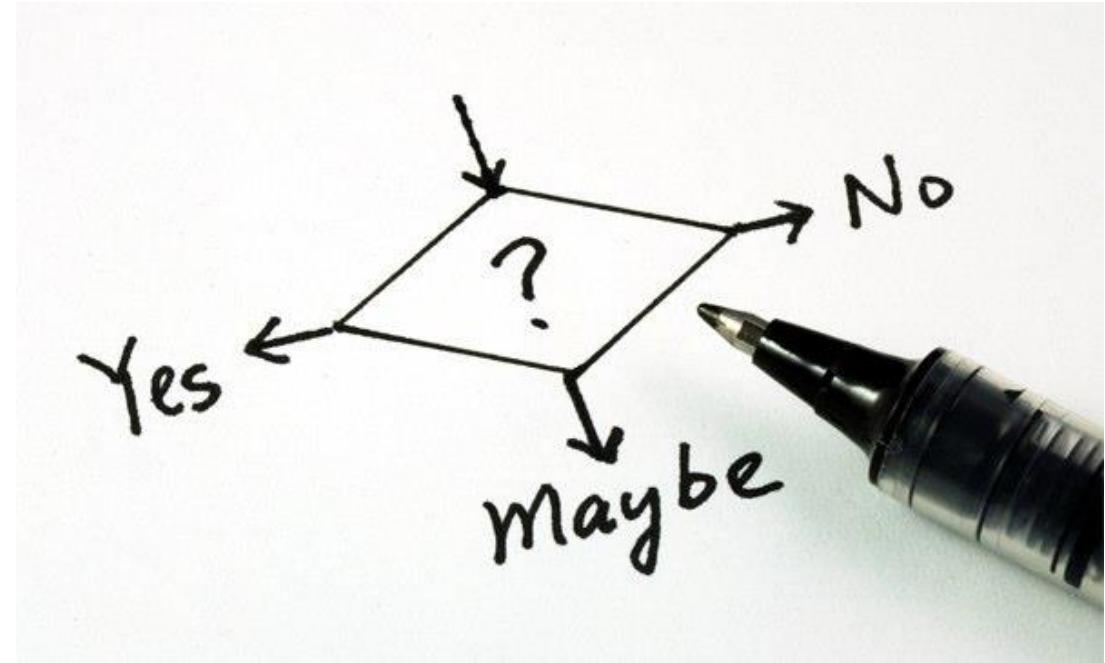
Context is king



Decisions, Decisions, Decisions...

Context is king

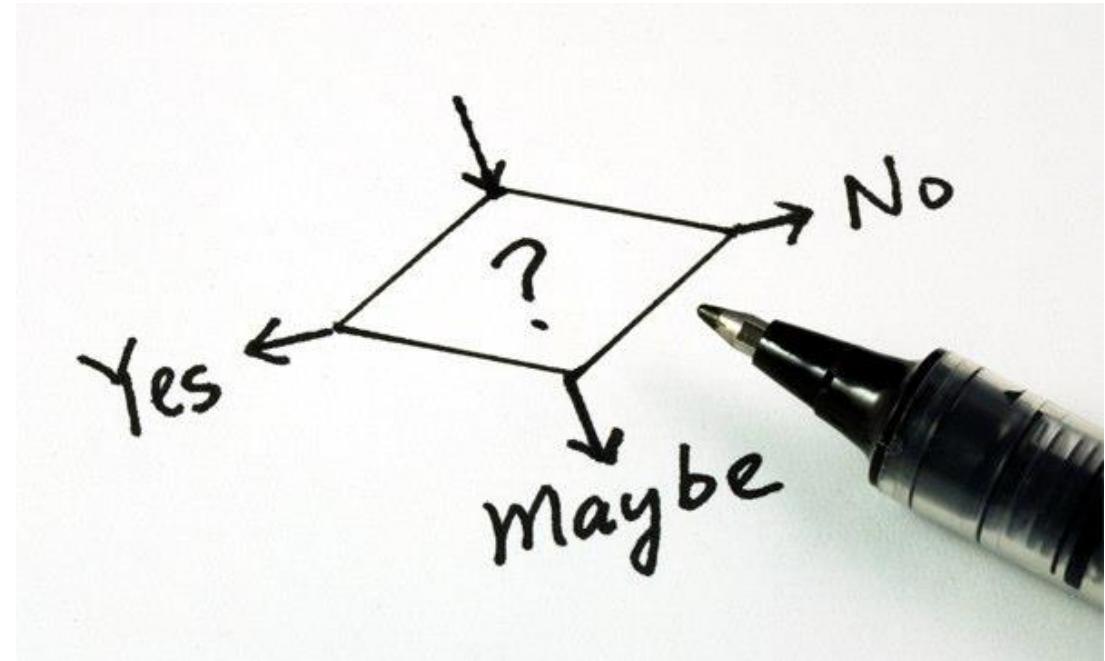
All decisions are a tradeoff



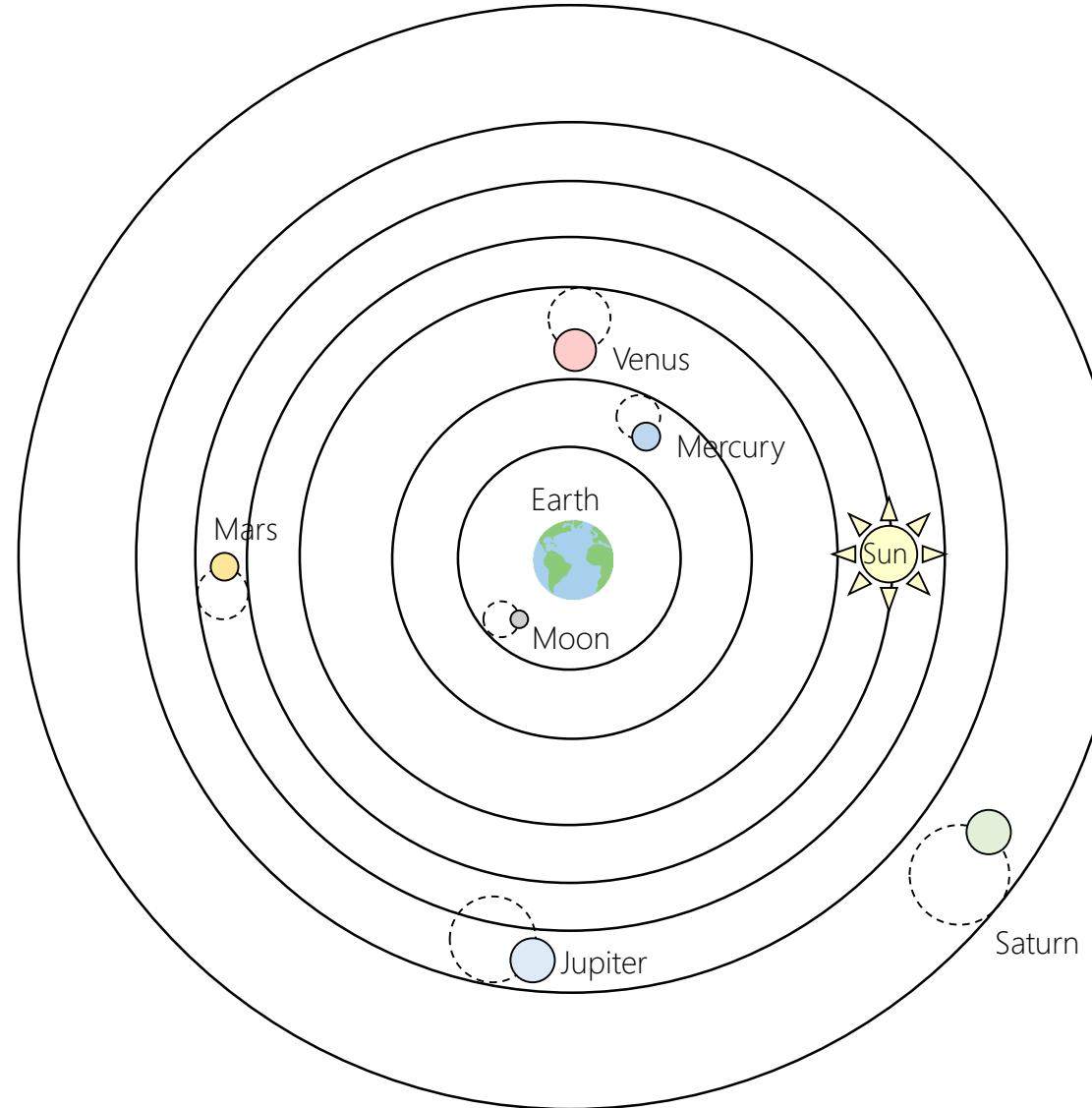
Decisions, Decisions, Decisions...

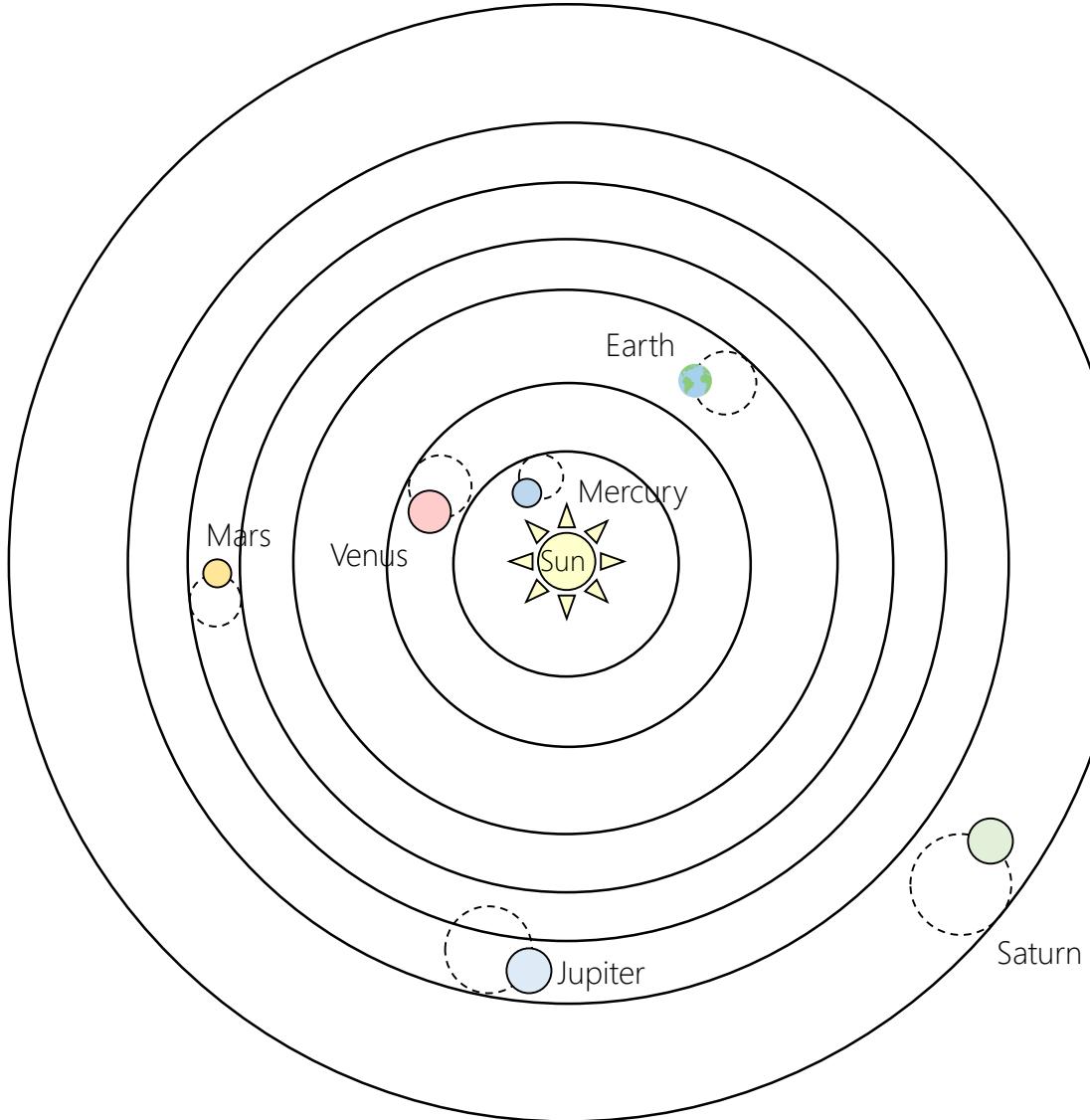
Context is king

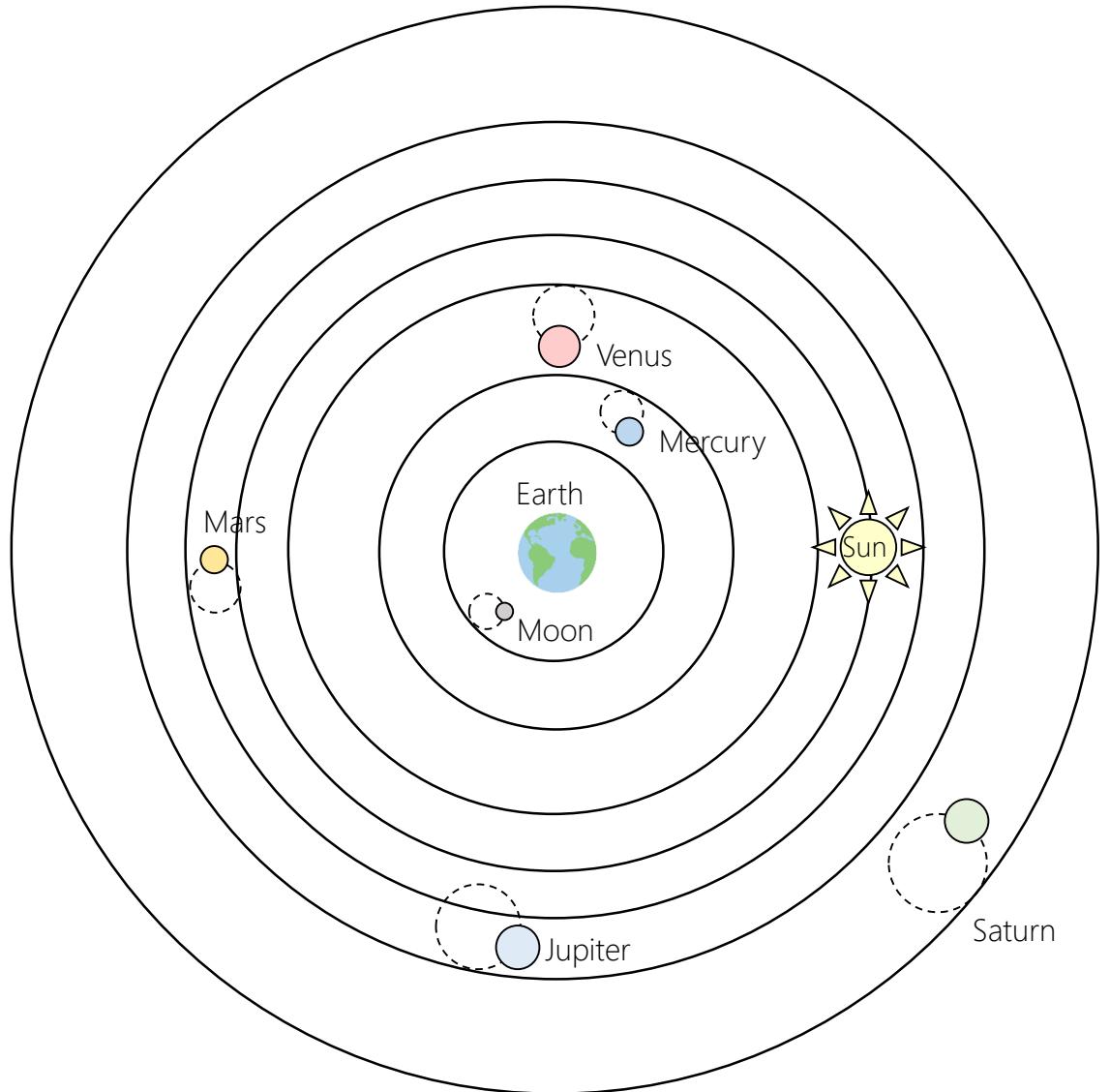
All decisions are a tradeoff
Use your best judgement



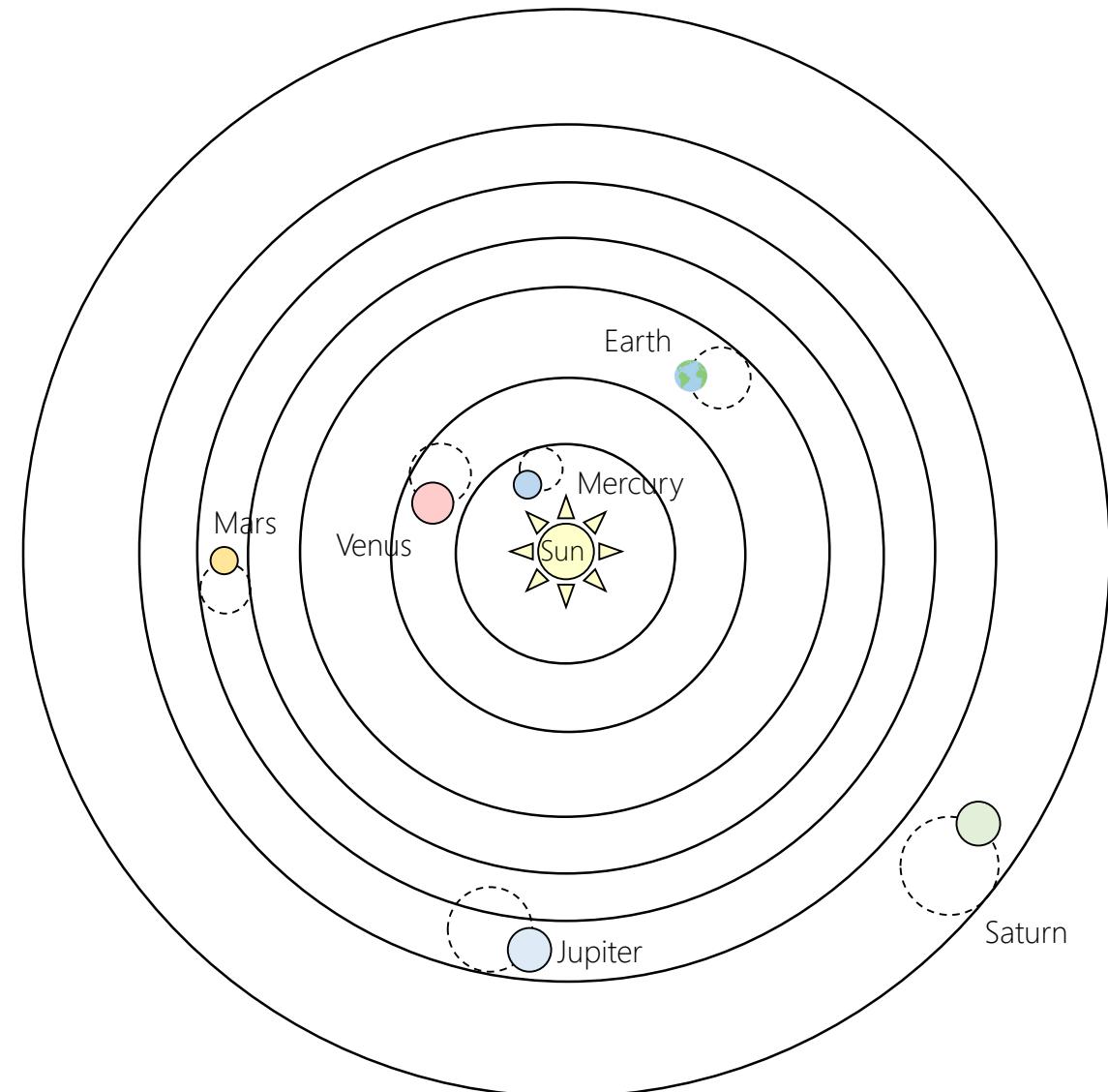
Domain-Centric Architecture





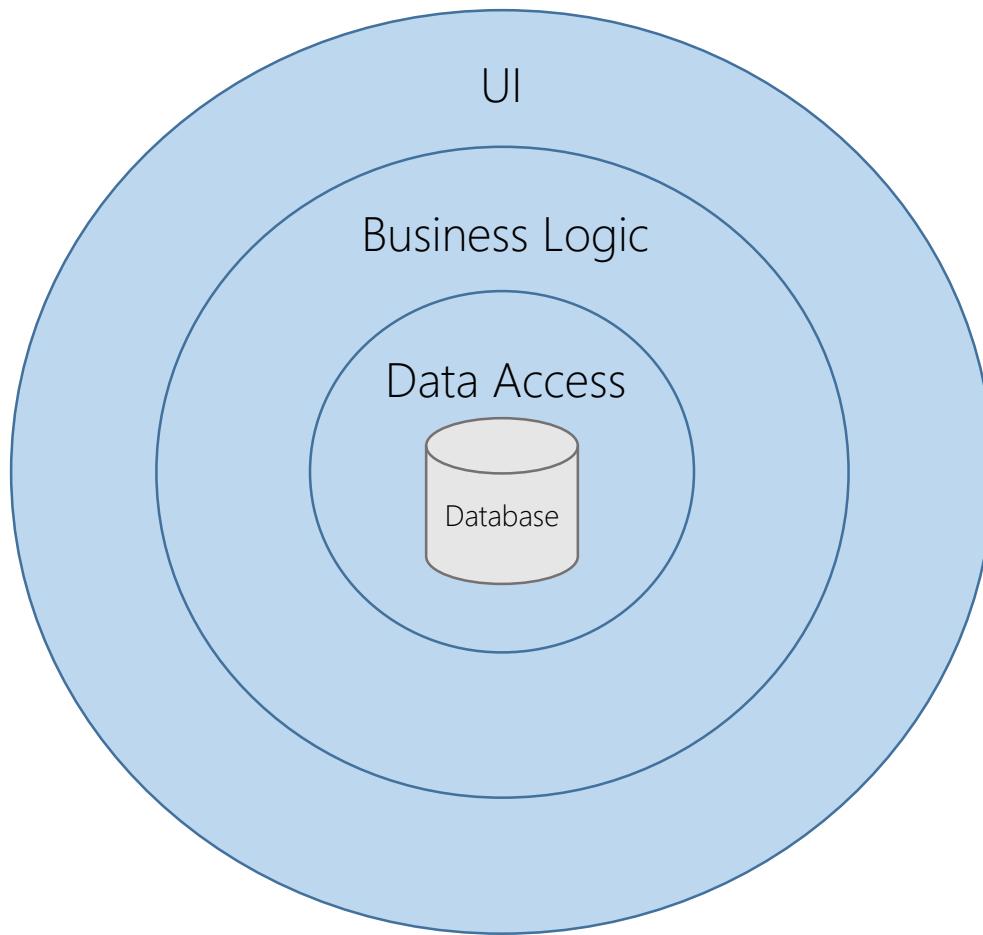


Geocentric Model

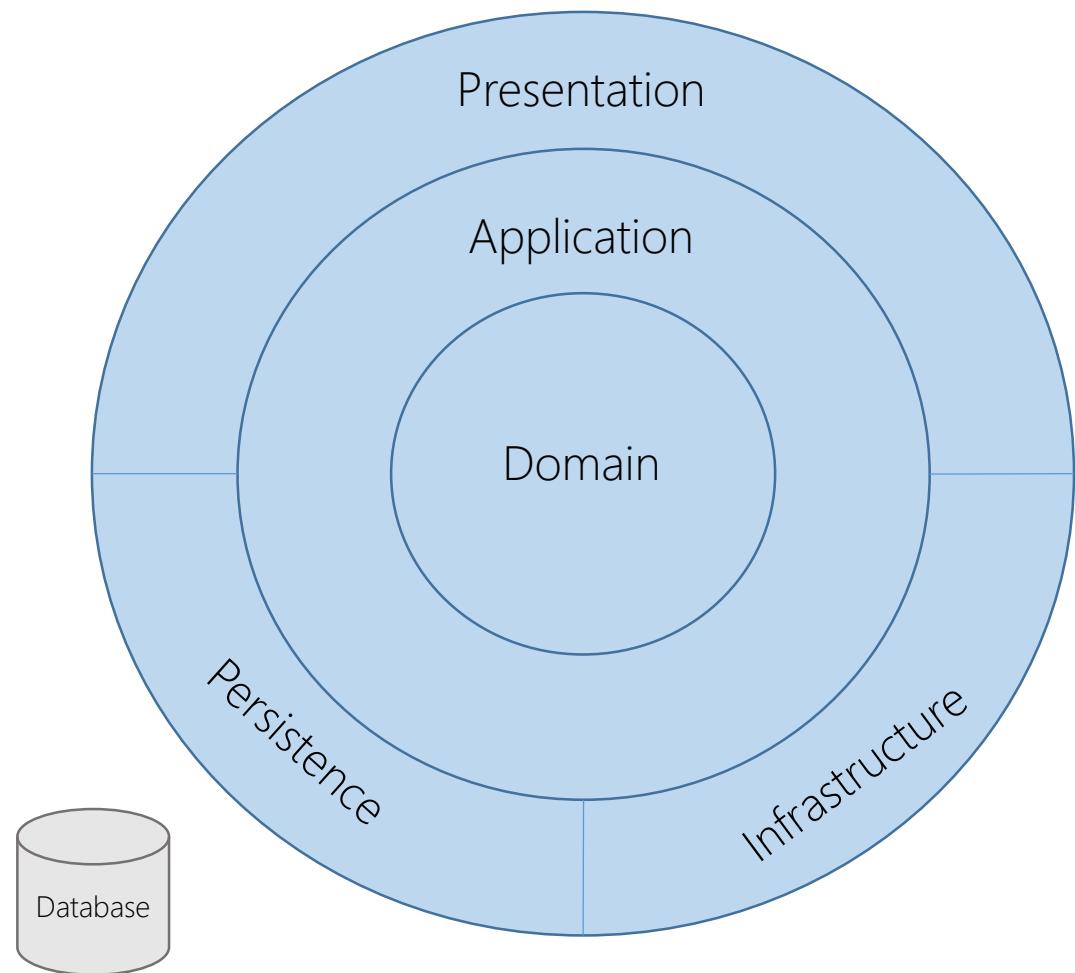
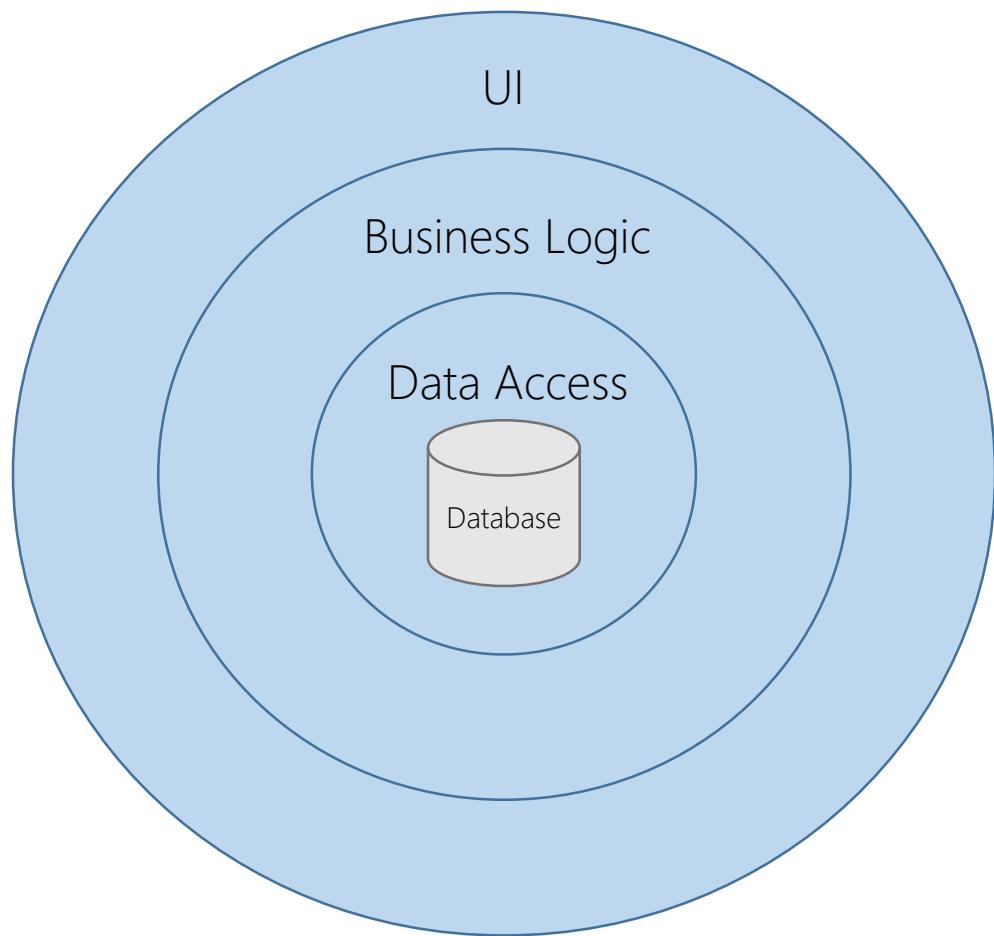


Heliocentric Model

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



Essential vs. Detail

Space is essential



Essential vs. Detail

Space is essential

Usability is essential



Essential vs. Detail

Building material is a detail



Essential vs. Detail

Building material is a detail

Ornamentation is a detail



Essential vs. Detail



Essential vs. Detail

Domain is essential



Essential vs. Detail

Domain is essential

Use cases are essential



Essential vs. Detail

Presentation is a detail



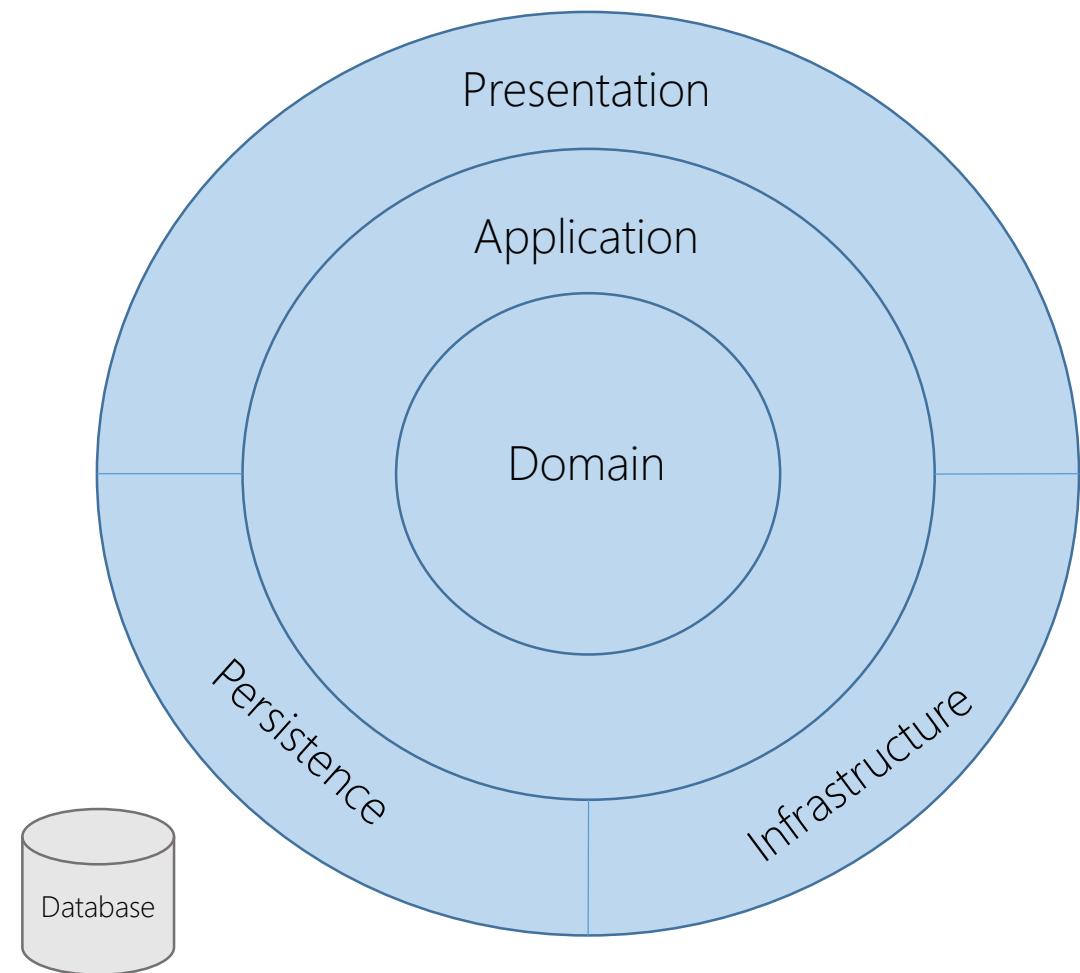
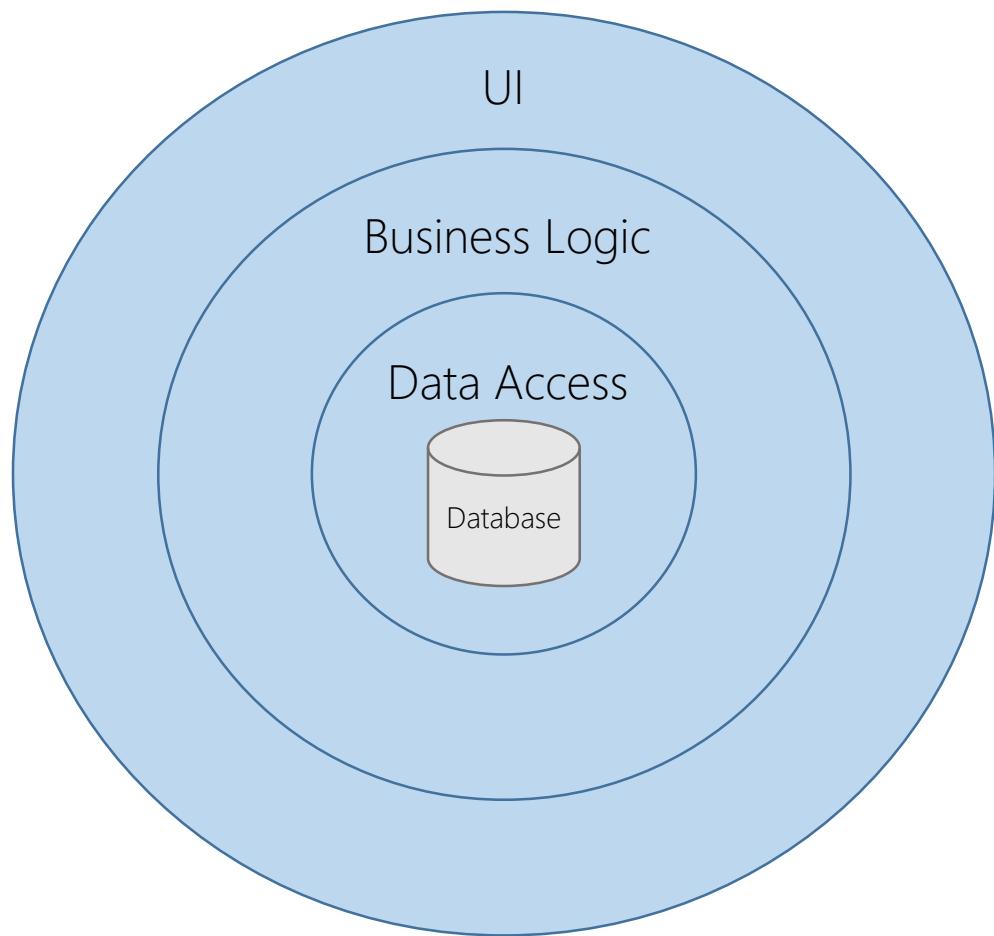
Essential vs. Detail

Presentation is a detail

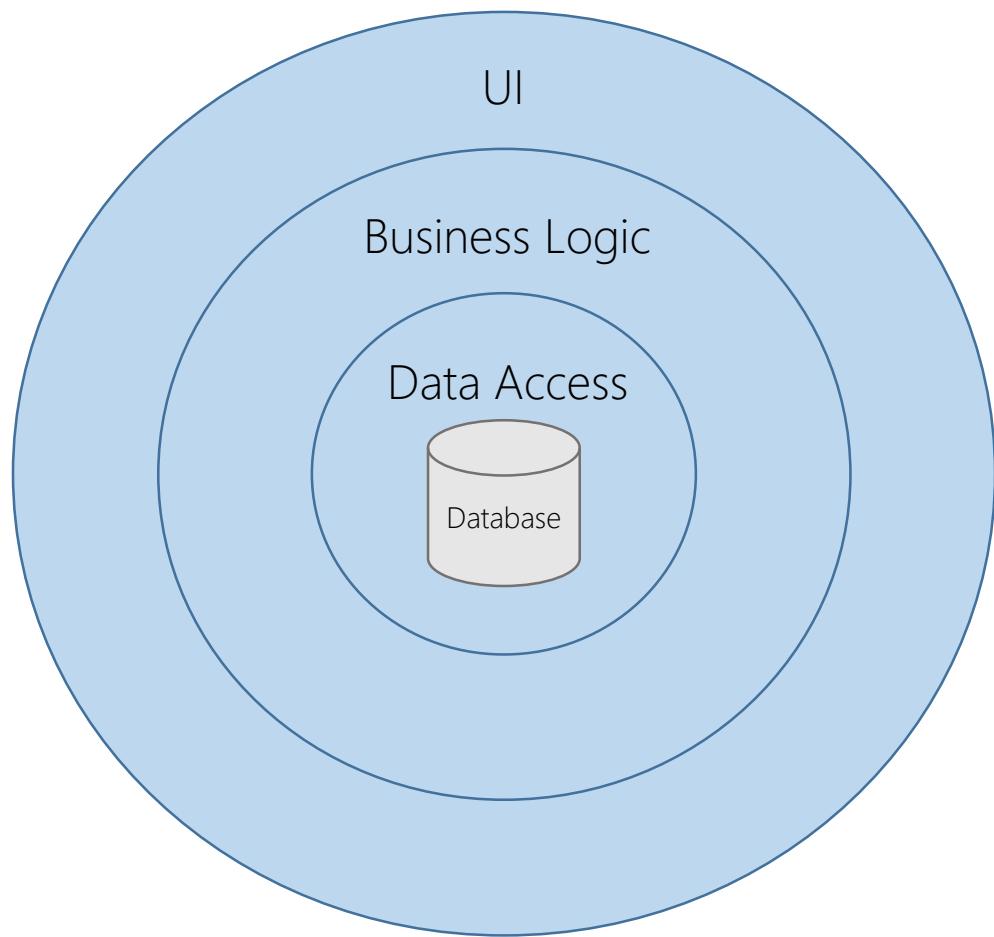
Persistence is a detail



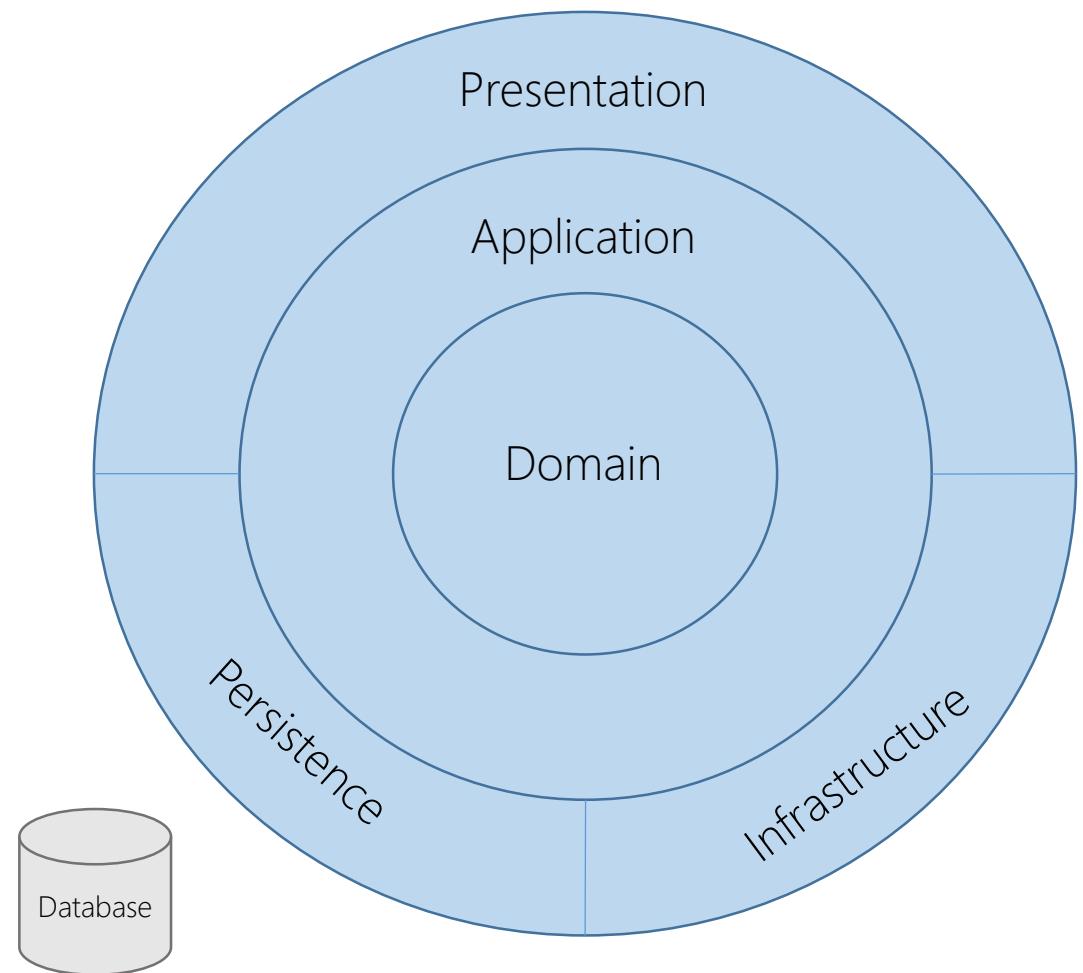
Database- vs. Domain-centric Architecture



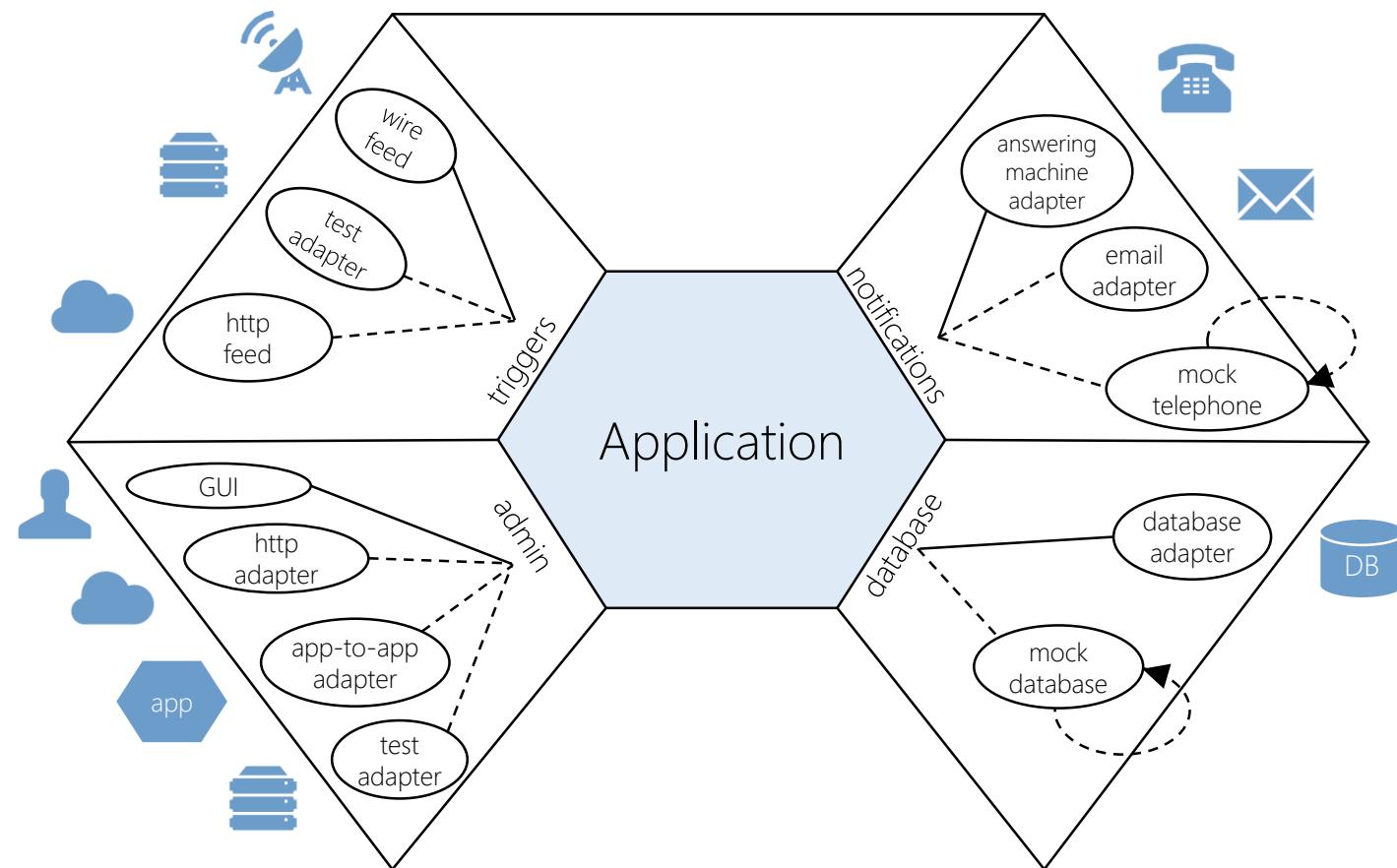
Database- vs. Domain-centric Architecture



Database- vs. Domain-centric Architecture

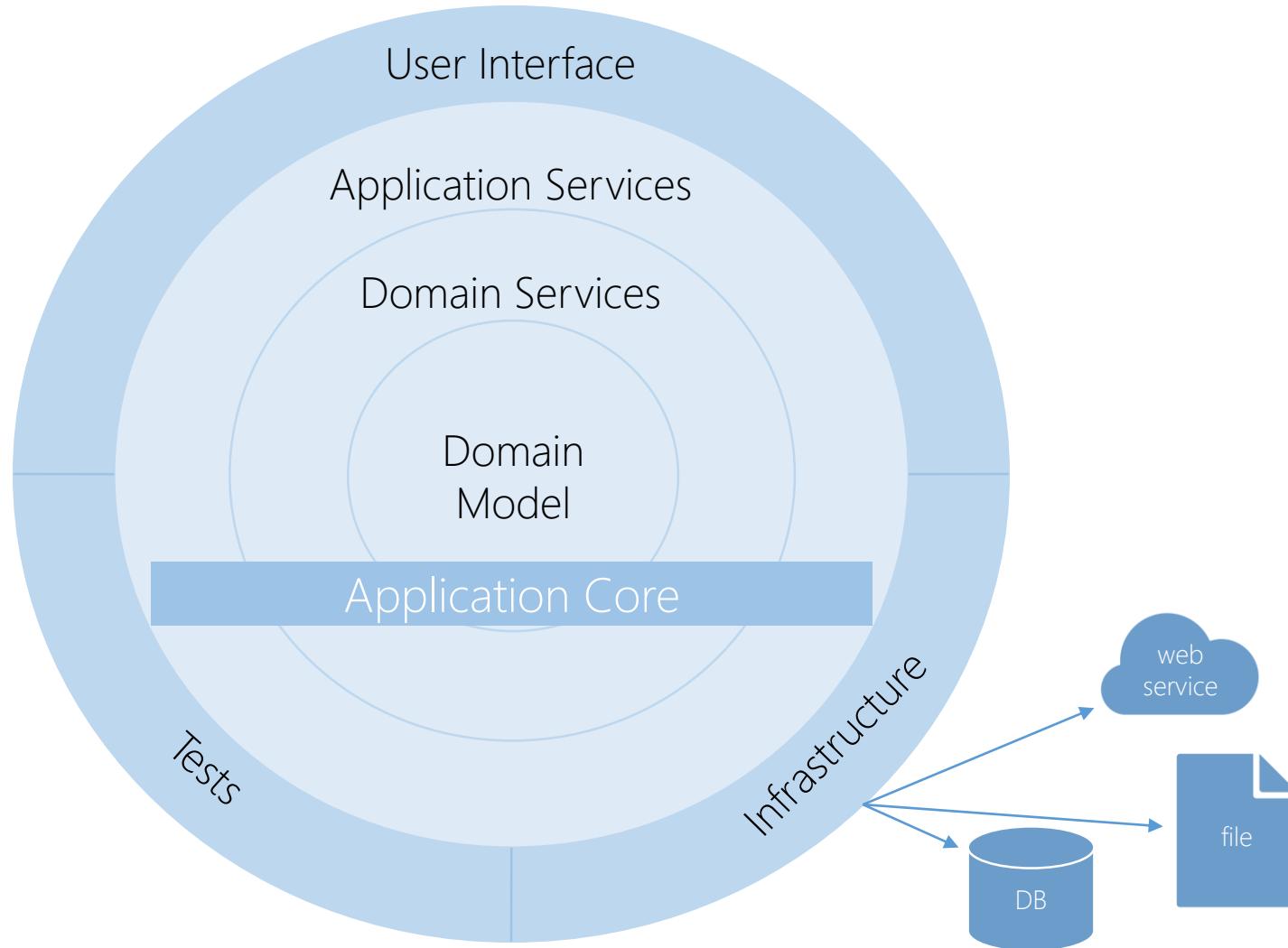


Hexagonal Architecture

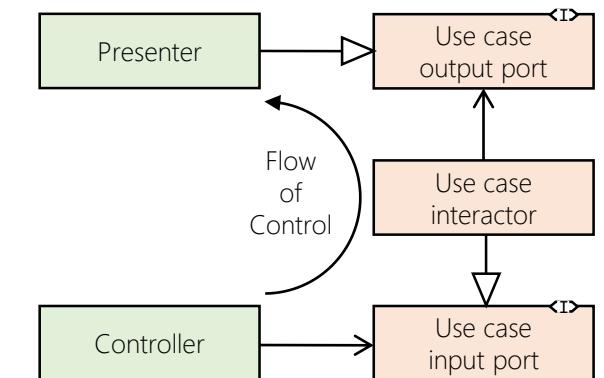
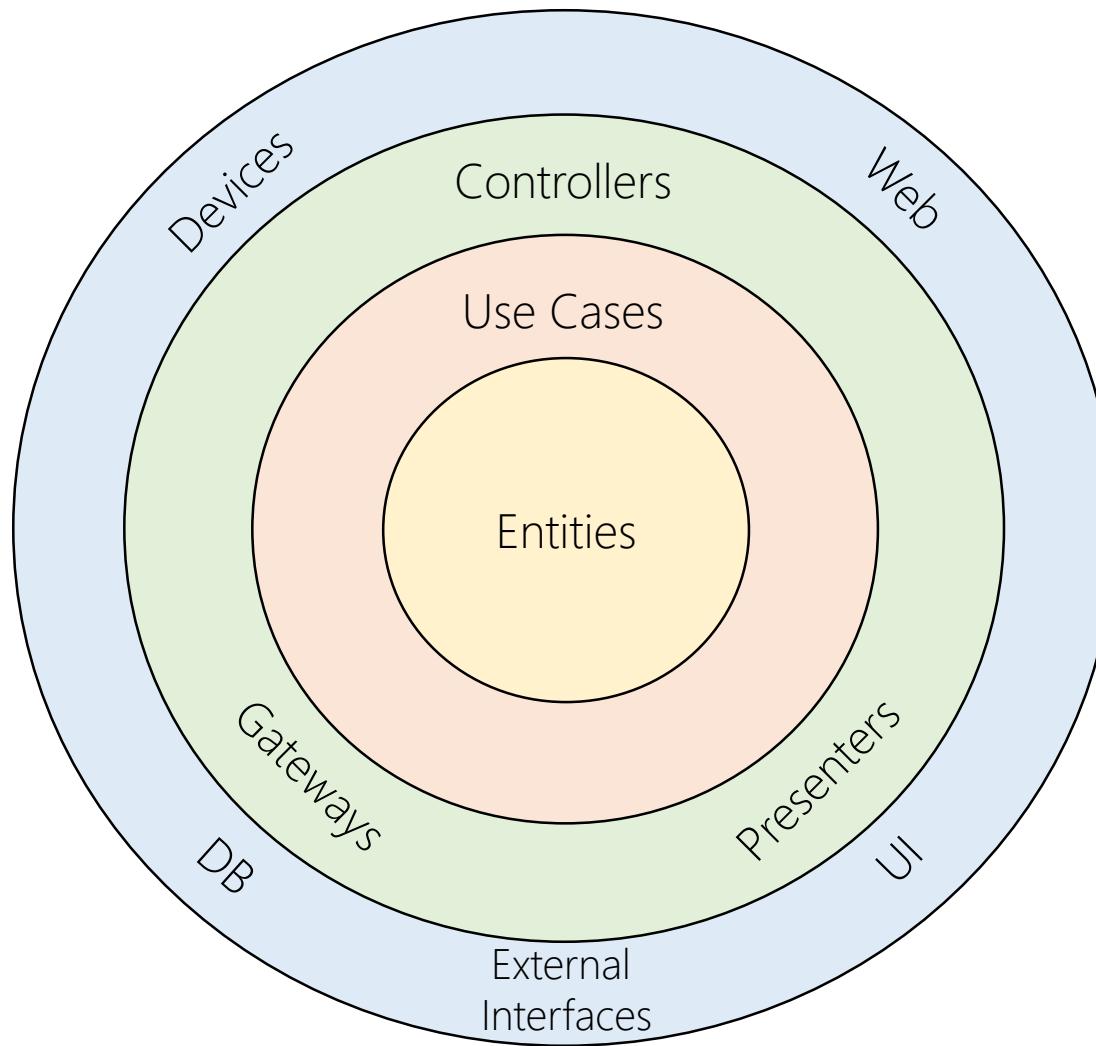


Original source: <http://alistair.cockburn.us/Hexagonal+architecture>

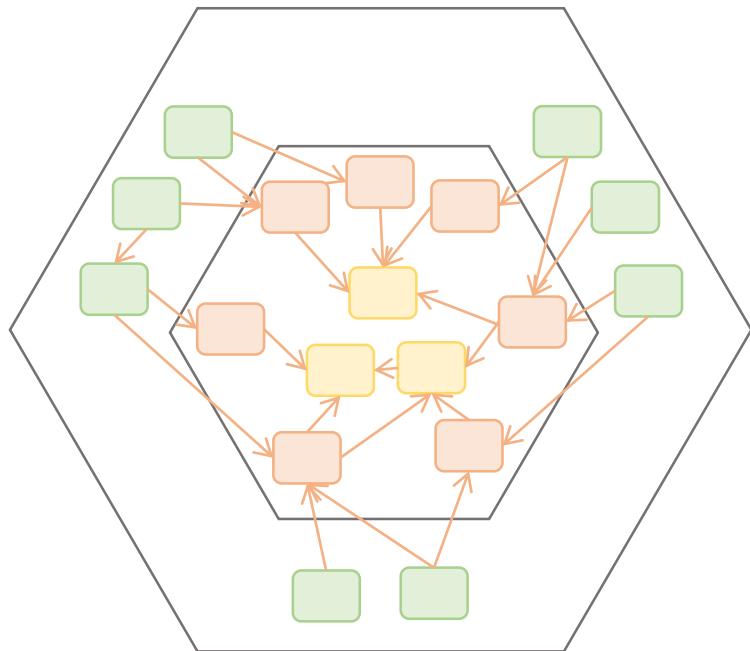
Onion Architecture



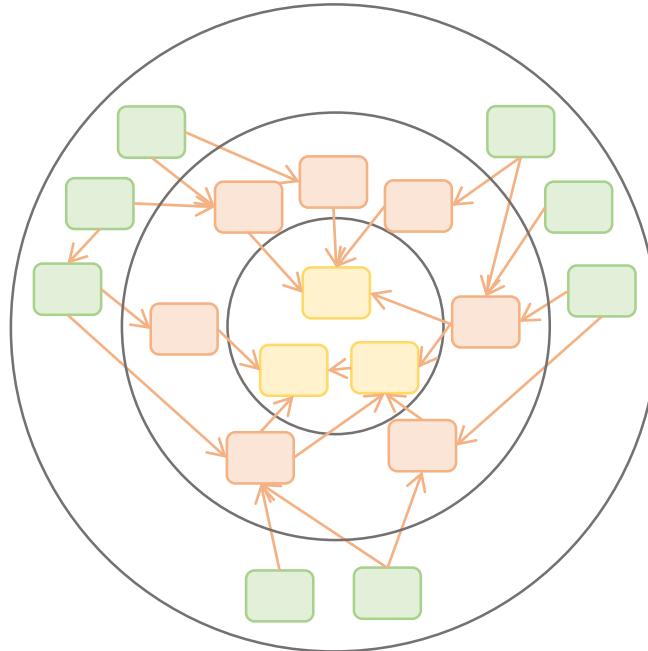
Clean Architecture



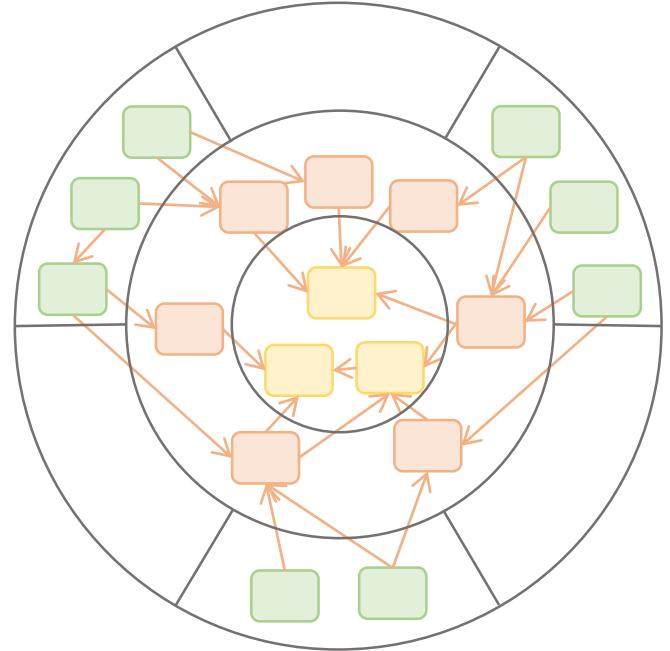
It's All the Same Thing



Hexagonal



Onion



Clean

Why Use Domain-Centric Architecture?

Pros

Focus on essential

Why Use Domain-Centric Architecture?

Pros

Focus on essential

Less coupling to details

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

Why Use Domain-Centric Architecture?

Pros

Focus on essential

Less coupling to details

Necessary for DDD

Cons

Change is difficult

Requires extra thought

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

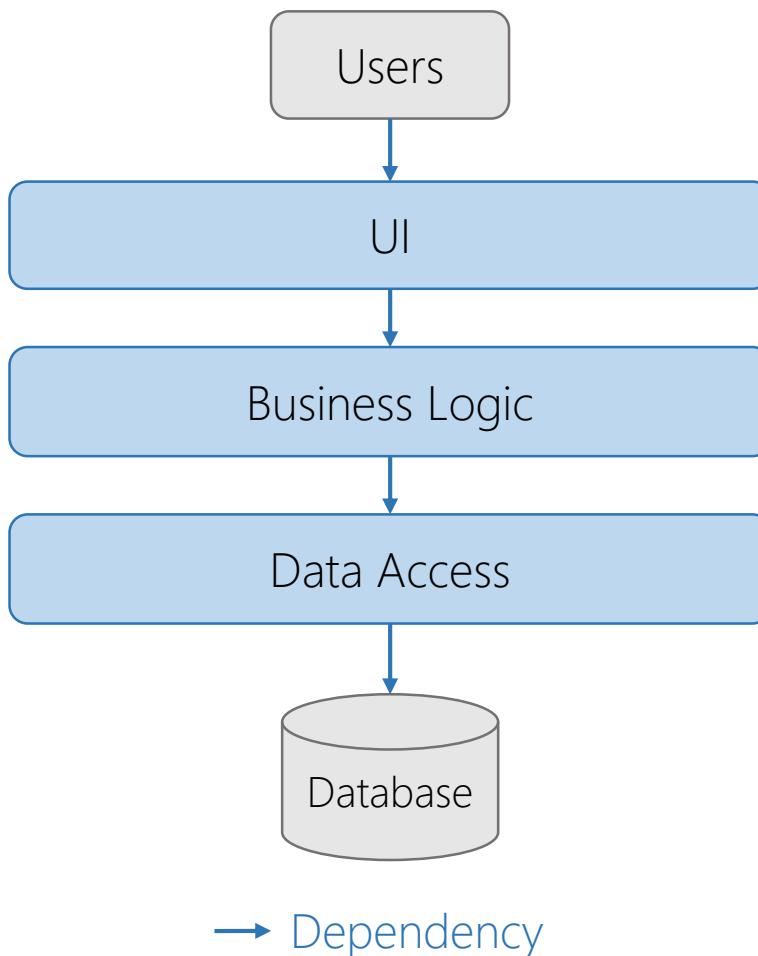
Application Layer

What Are Layers?

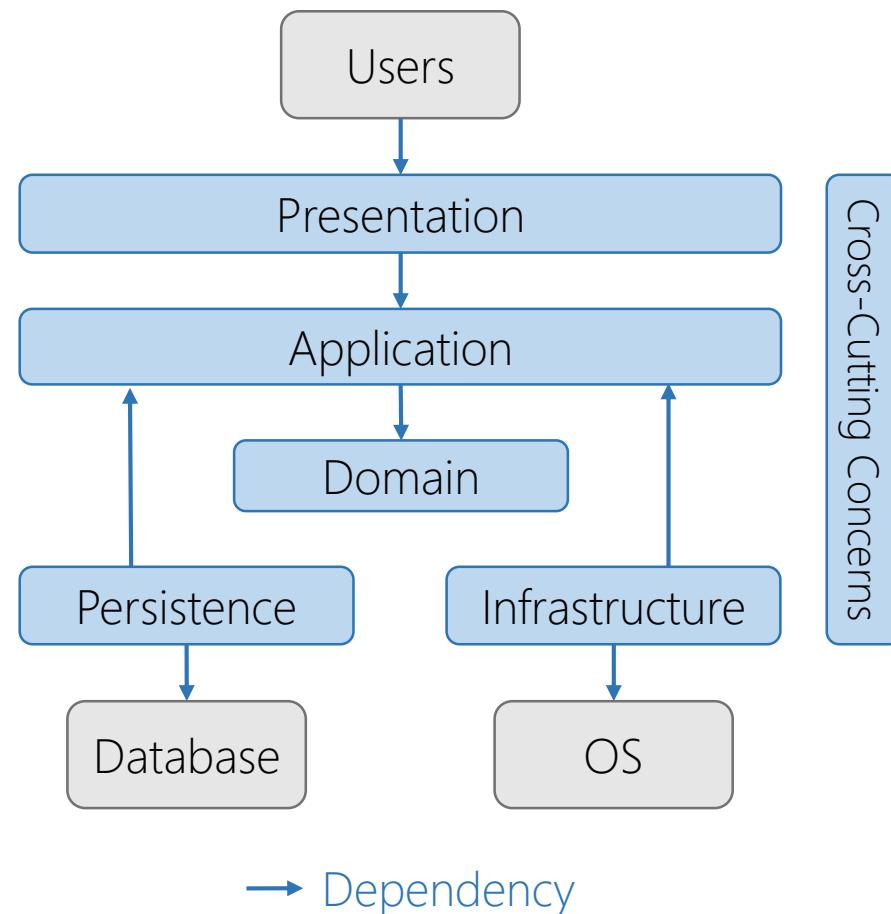
- Levels of abstraction
- Single-Responsibility Principle
- Developer roles / skills
- Multiple implementations
- Varying rates of change



Classic 3-Layer Architecture

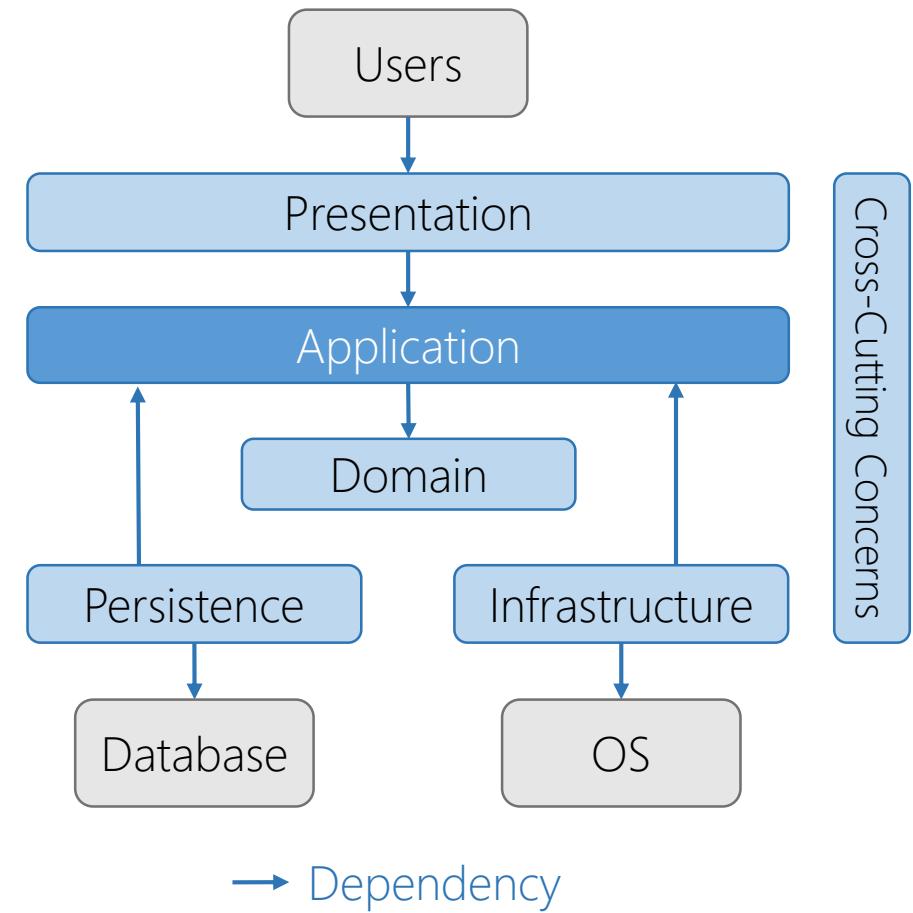


Modern 4-Layer Architecture



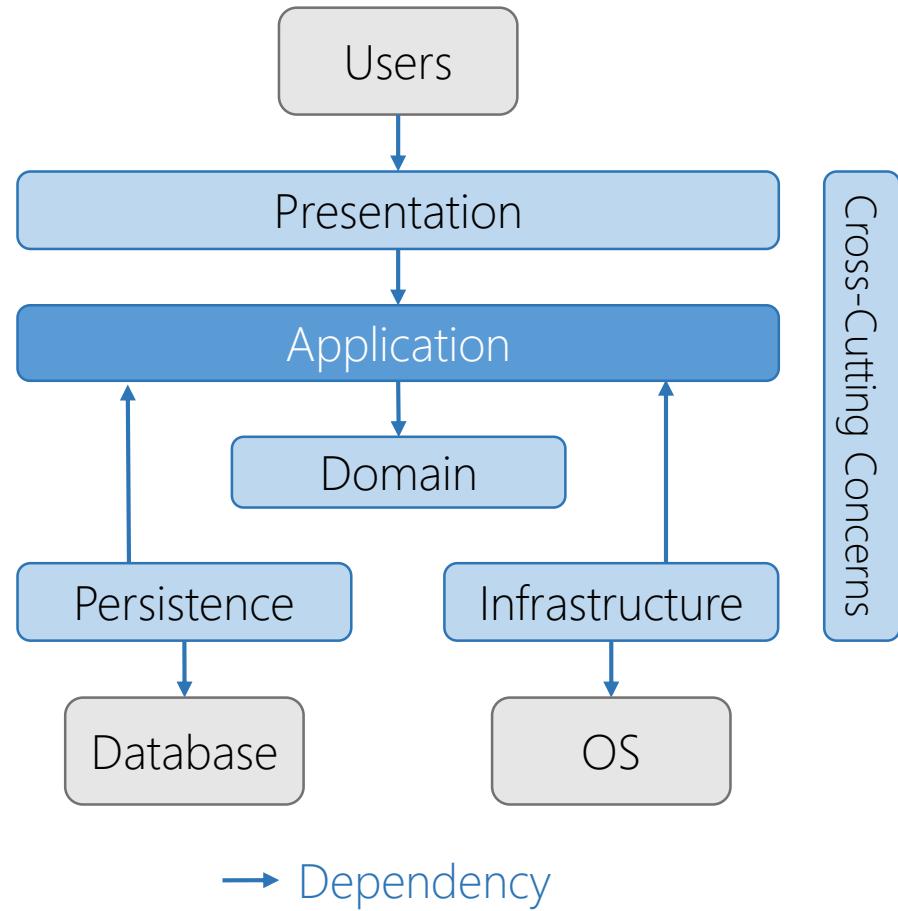
Application Layer

Implements use cases



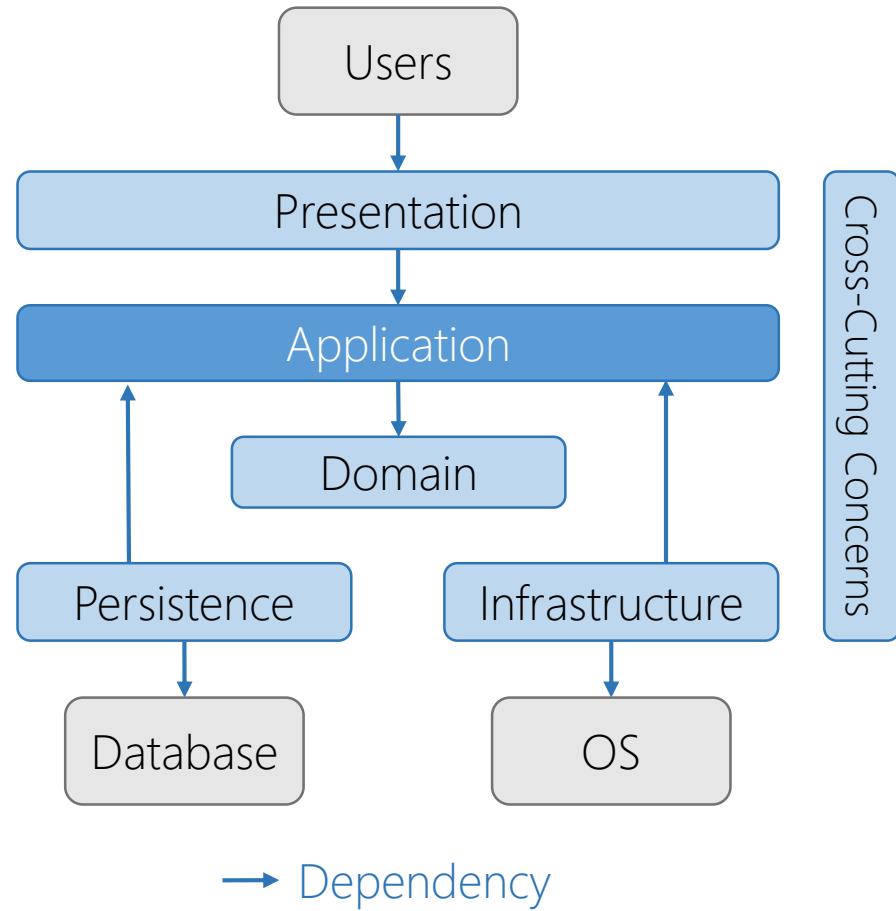
Application Layer

Implements use cases
High-level application logic



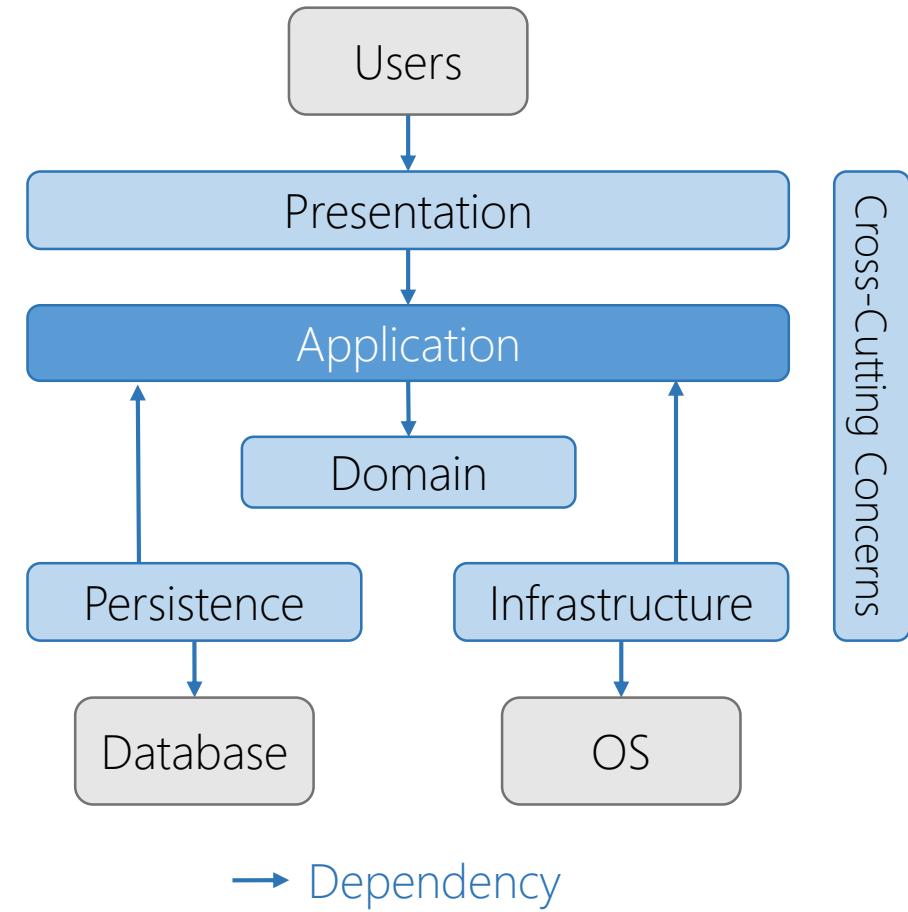
Application Layer

Implements use cases
High-level application logic
Knows about domain



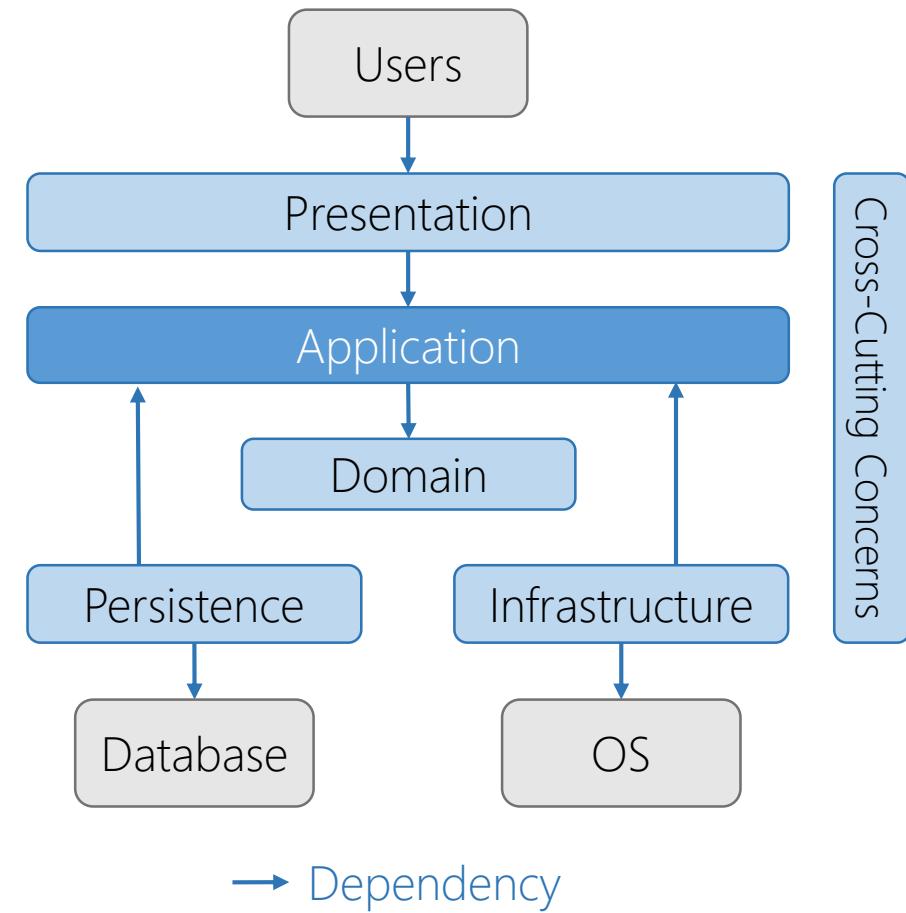
Application Layer

- Implements use cases
- High-level application logic
- Knows about domain
- No knowledge of other layers

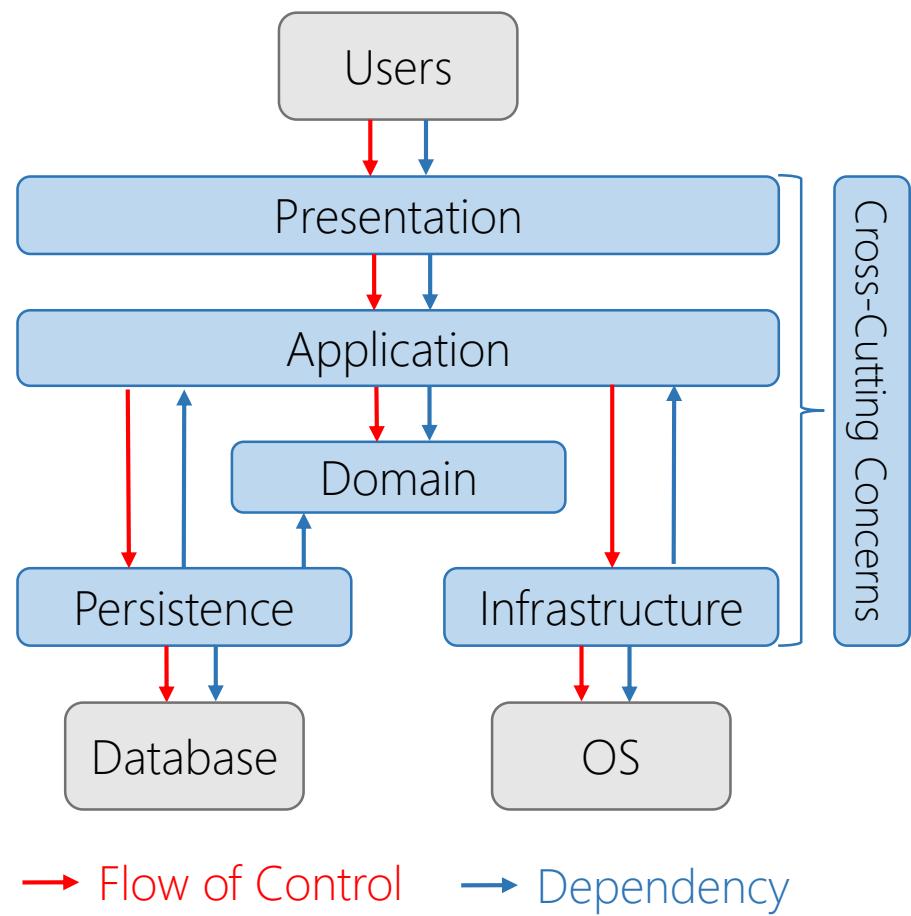


Application Layer

- Implements use cases
- High-level application logic
- Knows about domain
- No knowledge of other layers
- Contains interfaces for details

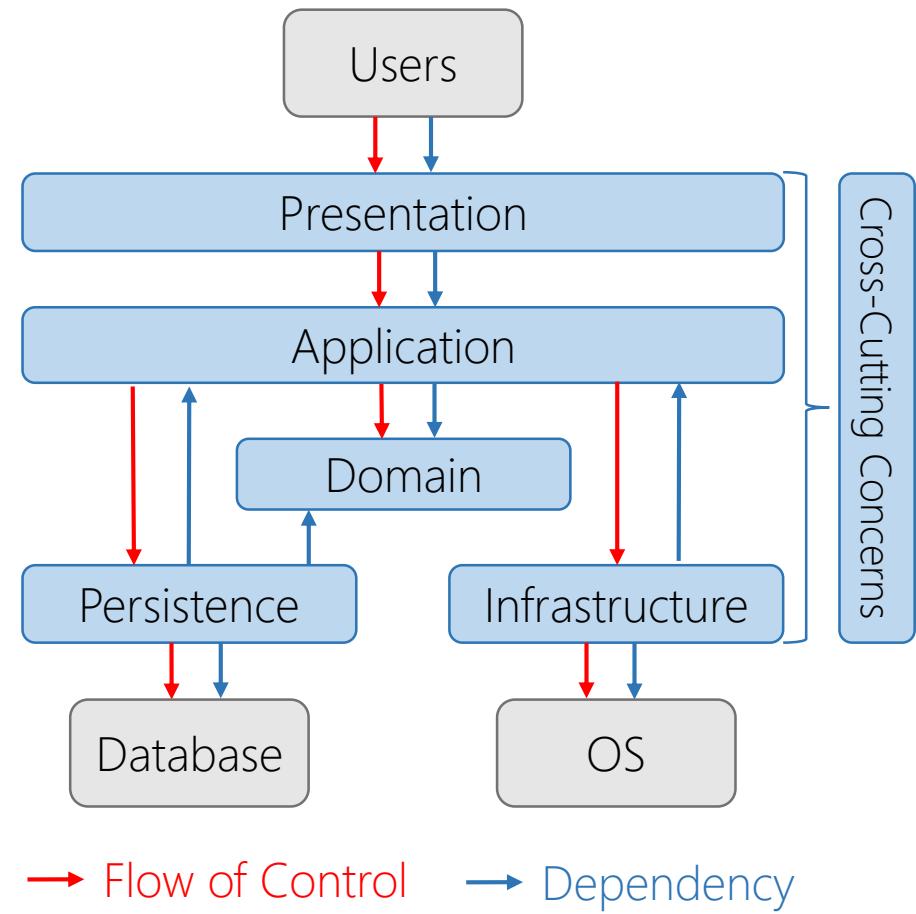


Layer Dependencies



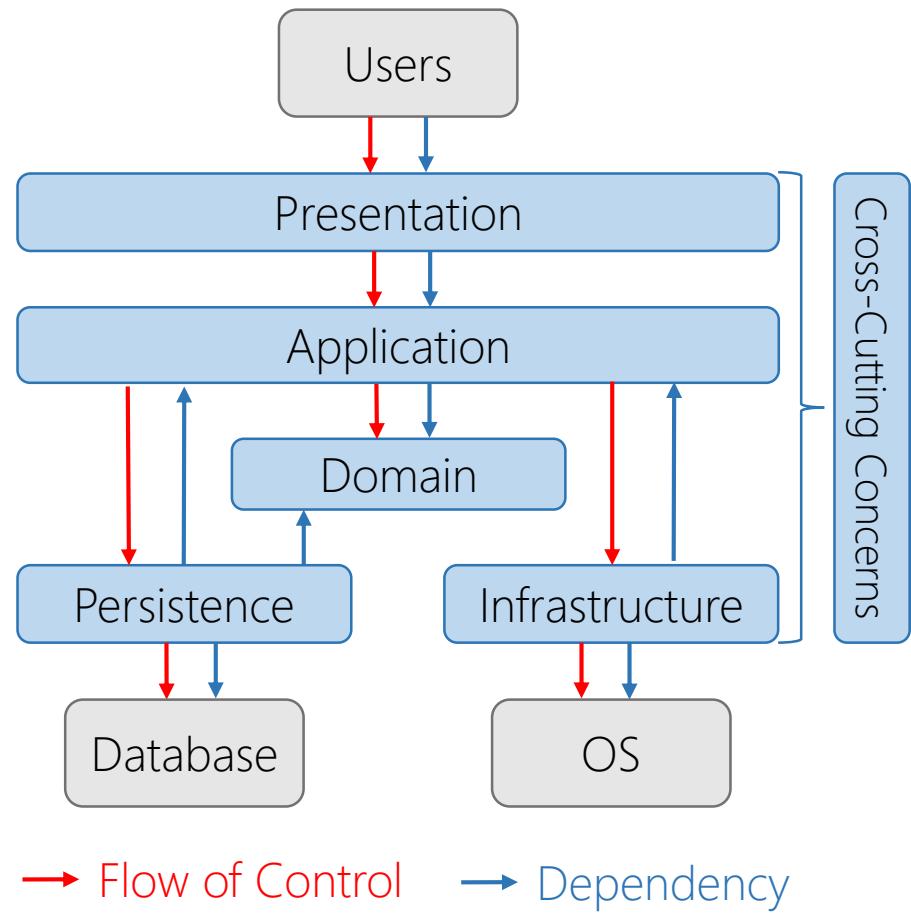
Layer Dependencies

Dependency inversion



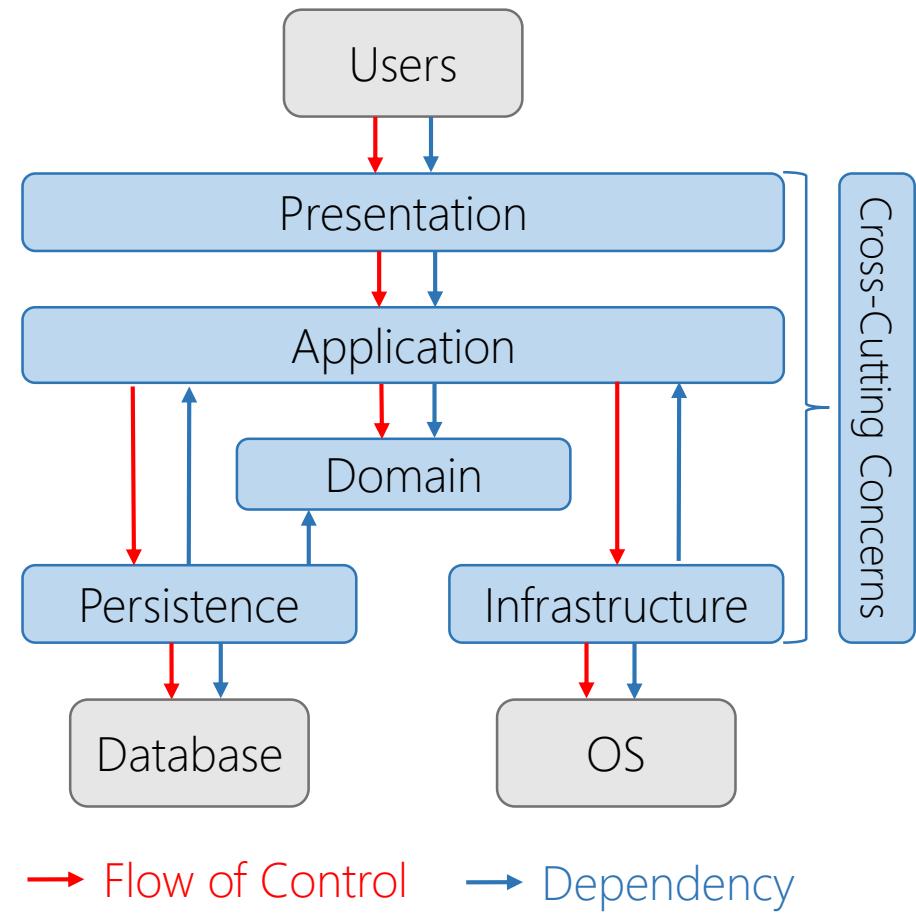
Layer Dependencies

Dependency inversion
Inversion of control



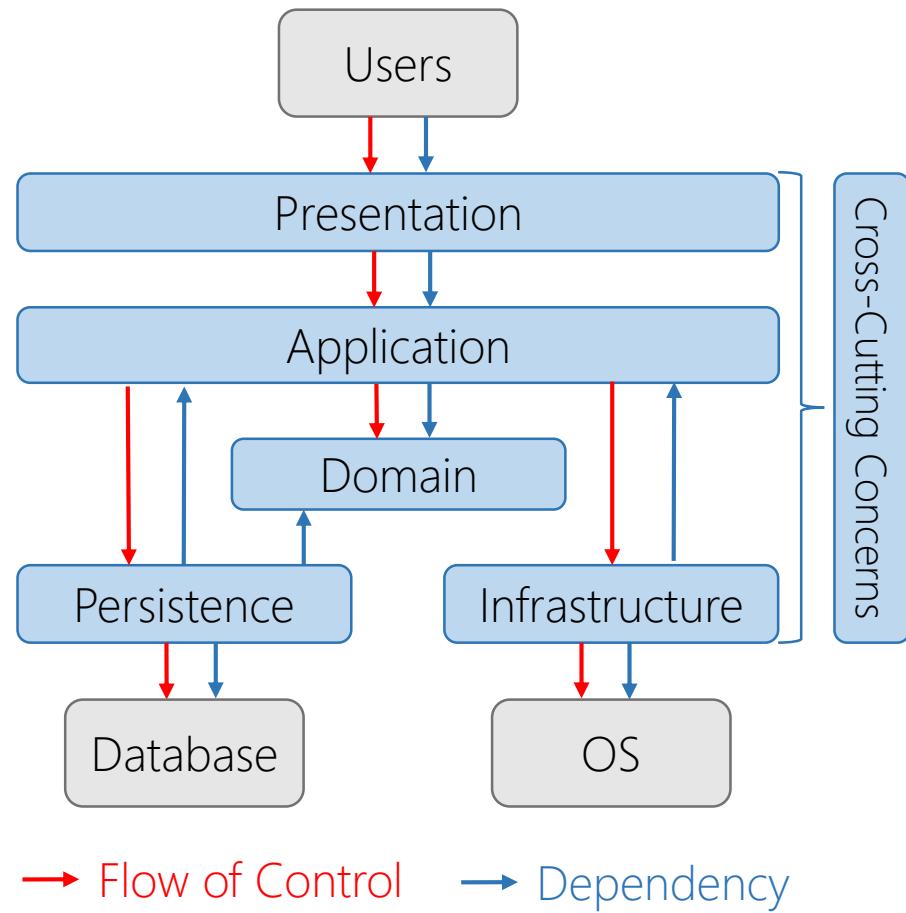
Layer Dependencies

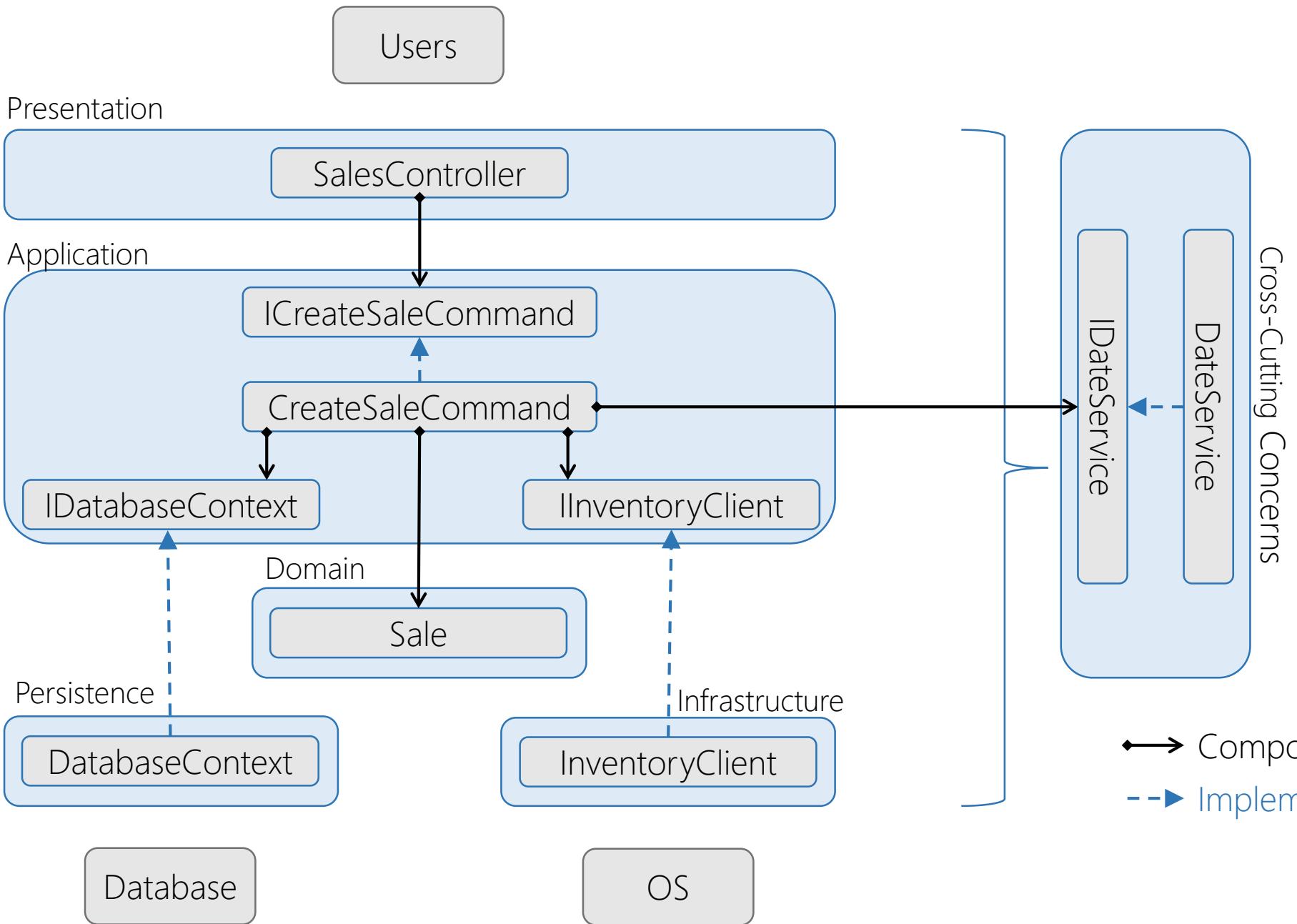
Dependency inversion
Inversion of control
Independent deployability



Layer Dependencies

Dependency inversion
Inversion of control
Independent deployability
Flexibility and maintainability





Why Use an Application Layer?

Pros

Focus is on use cases

Why Use an Application Layer?

Pros

Focus is on use cases

Easy to understand

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

Why Use an Application Layer?

Pros

Focus is on use cases

Easy to understand

Follows DIP

Cons

Additional cost

Requires extra thought

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

Does something

Should modify state

Should not return a value

Query

Answers a question

Should not modify state

Always returns a value

Command-Query Separation

Command

Does something

Should modify state

Should not return a value
(ideally)

Query

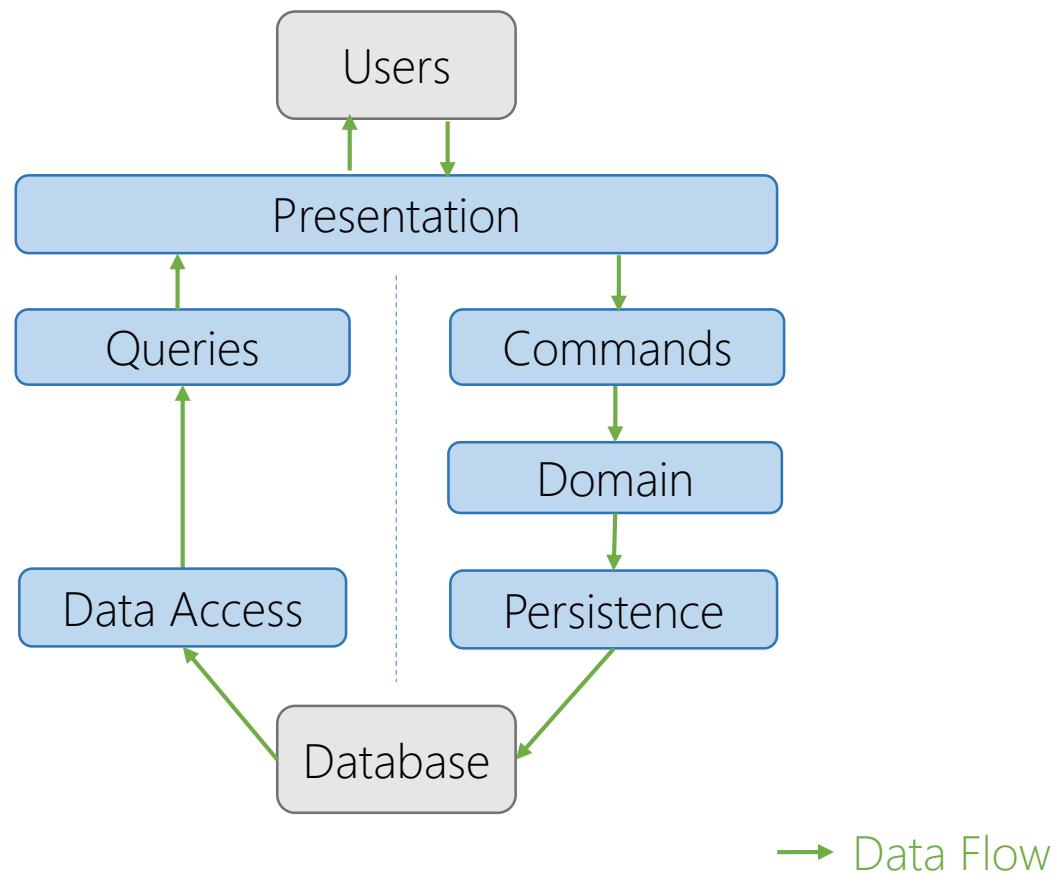
Answers a question

Should not modify state

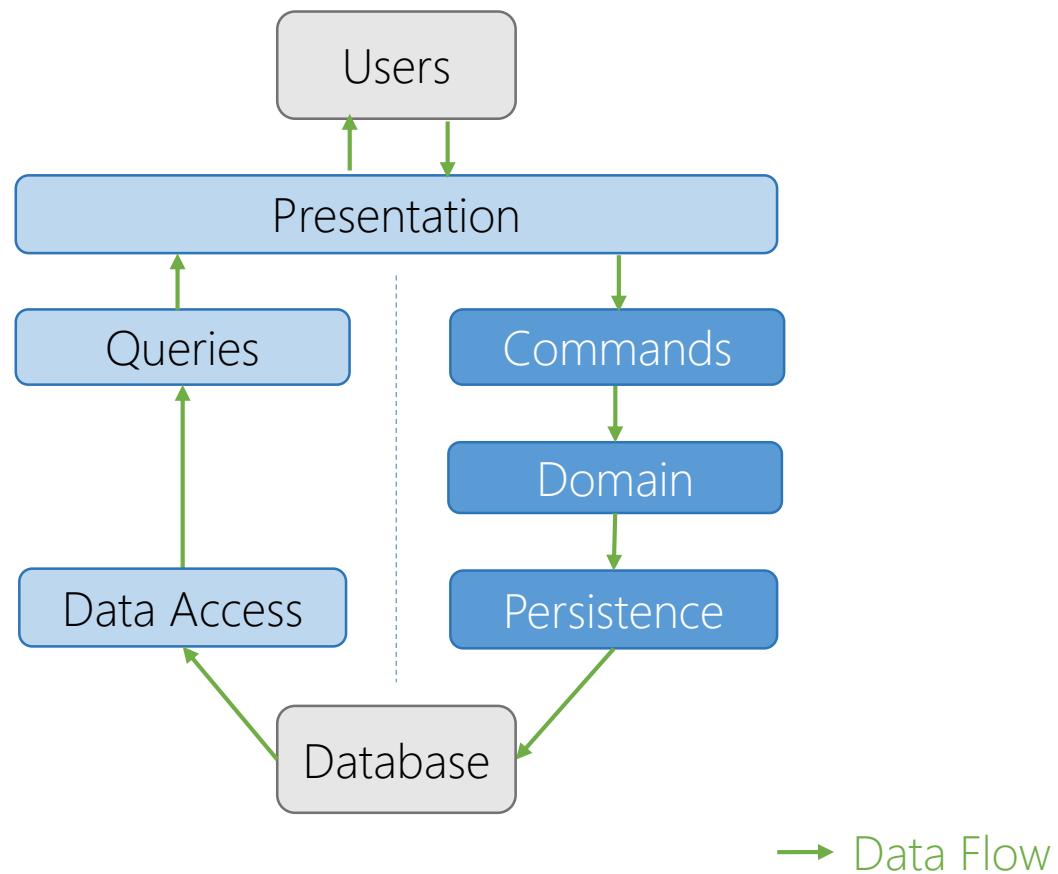
Always returns a value

Avoid mixing the two!

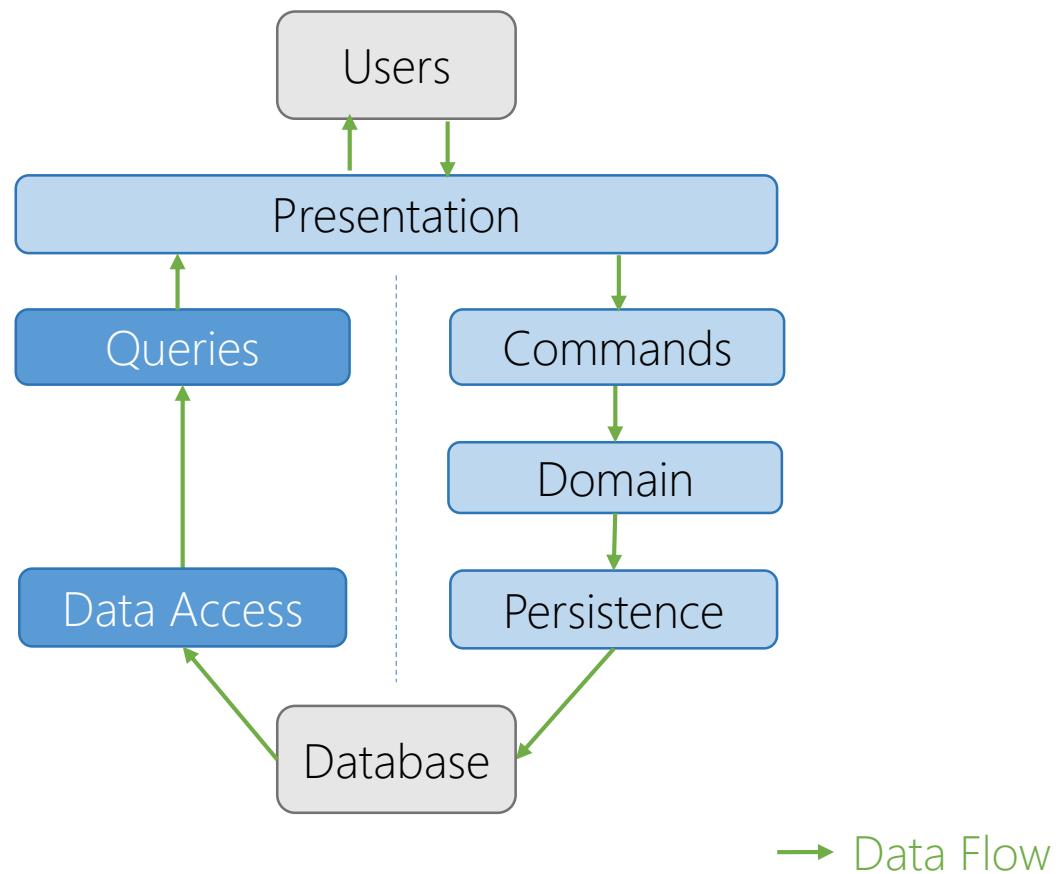
CQRS Architectures



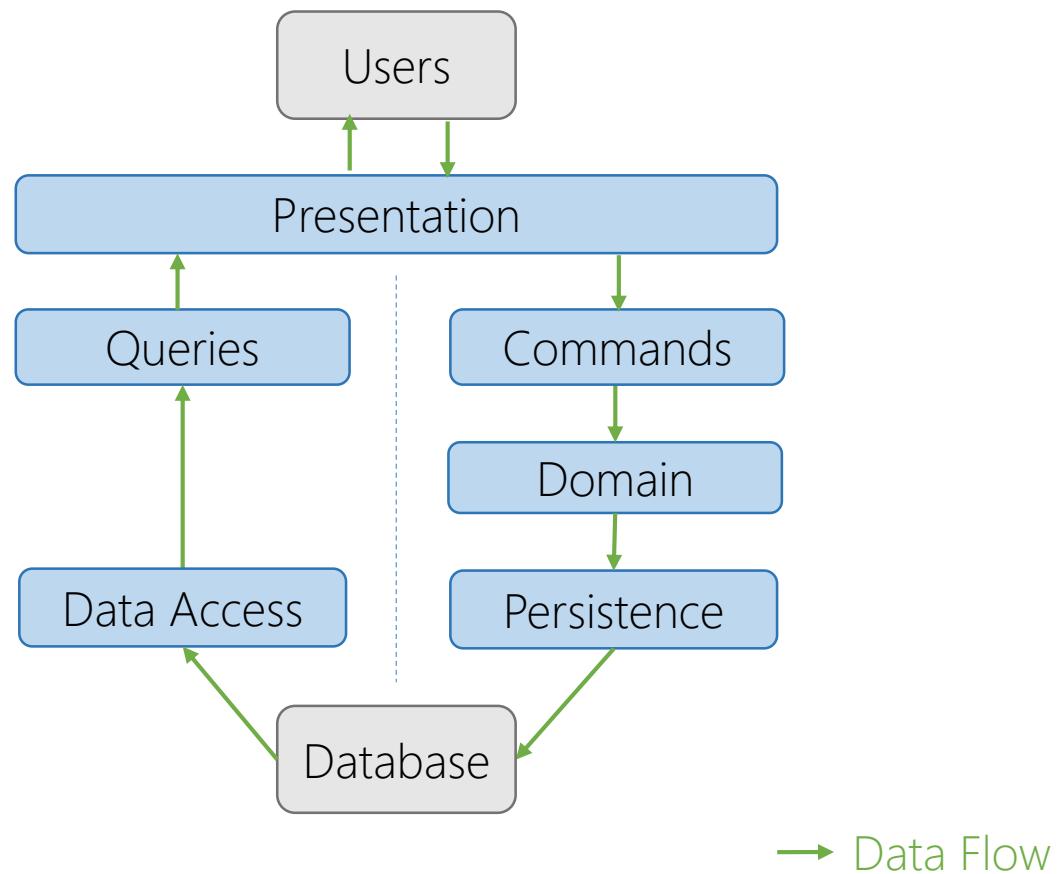
CQRS Architectures



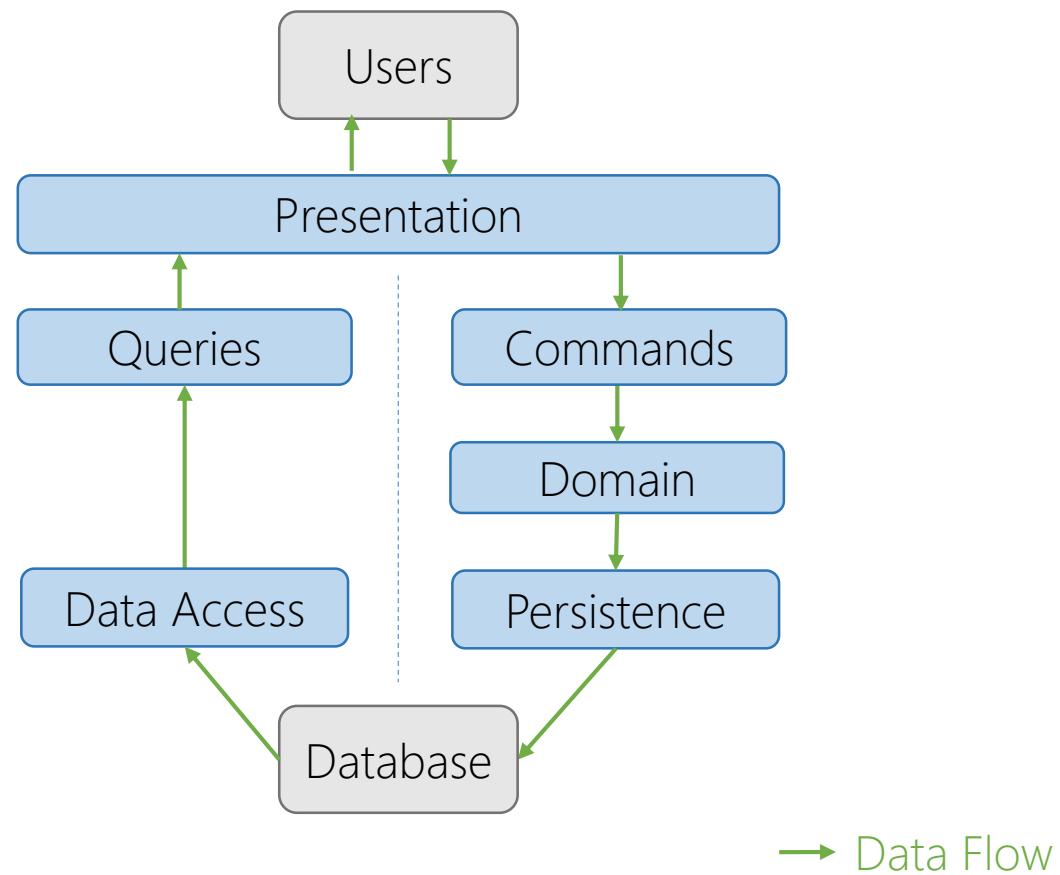
CQRS Architectures



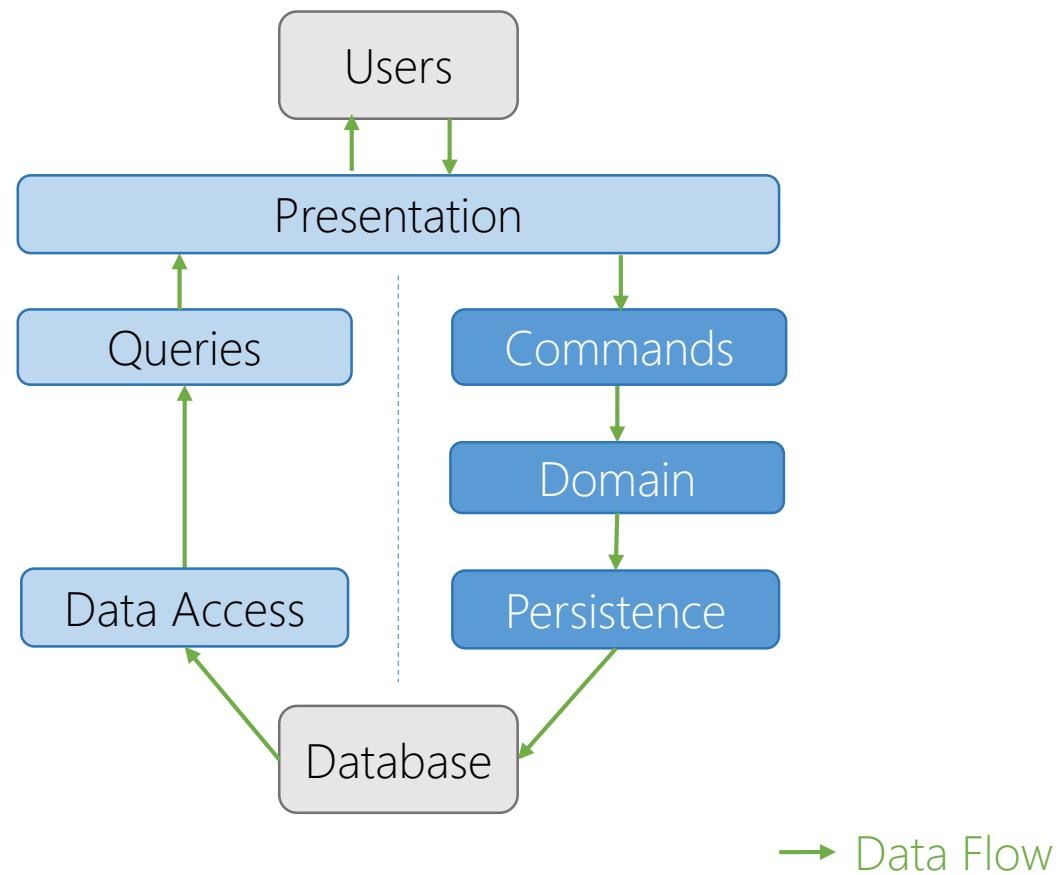
CQRS Architectures



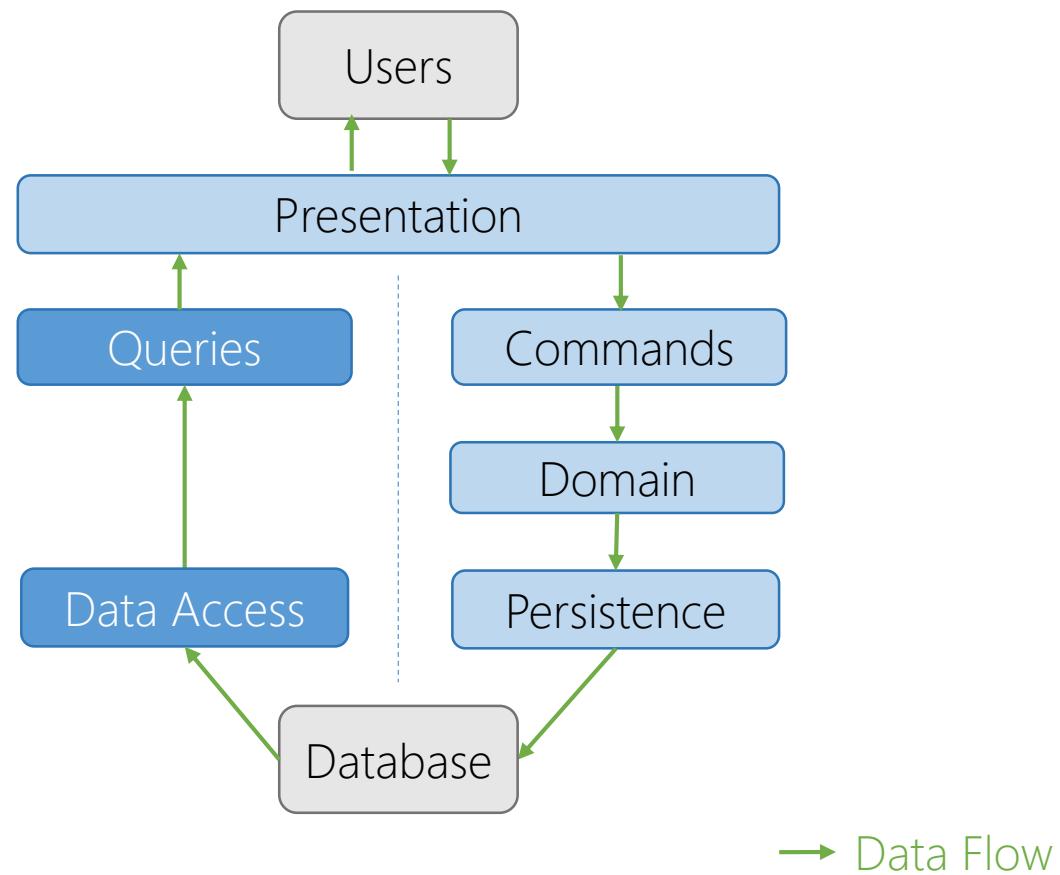
CQRS Type 1 – Single Database



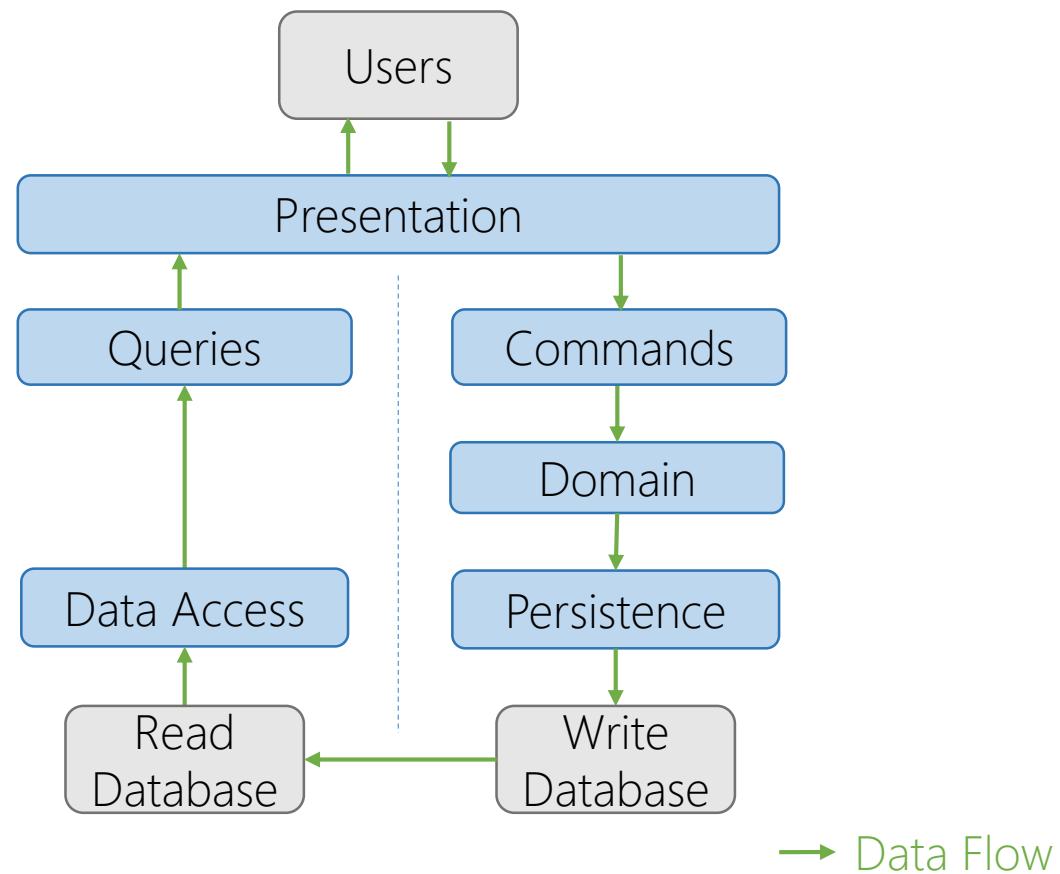
CQRS Type 1 – Single Database



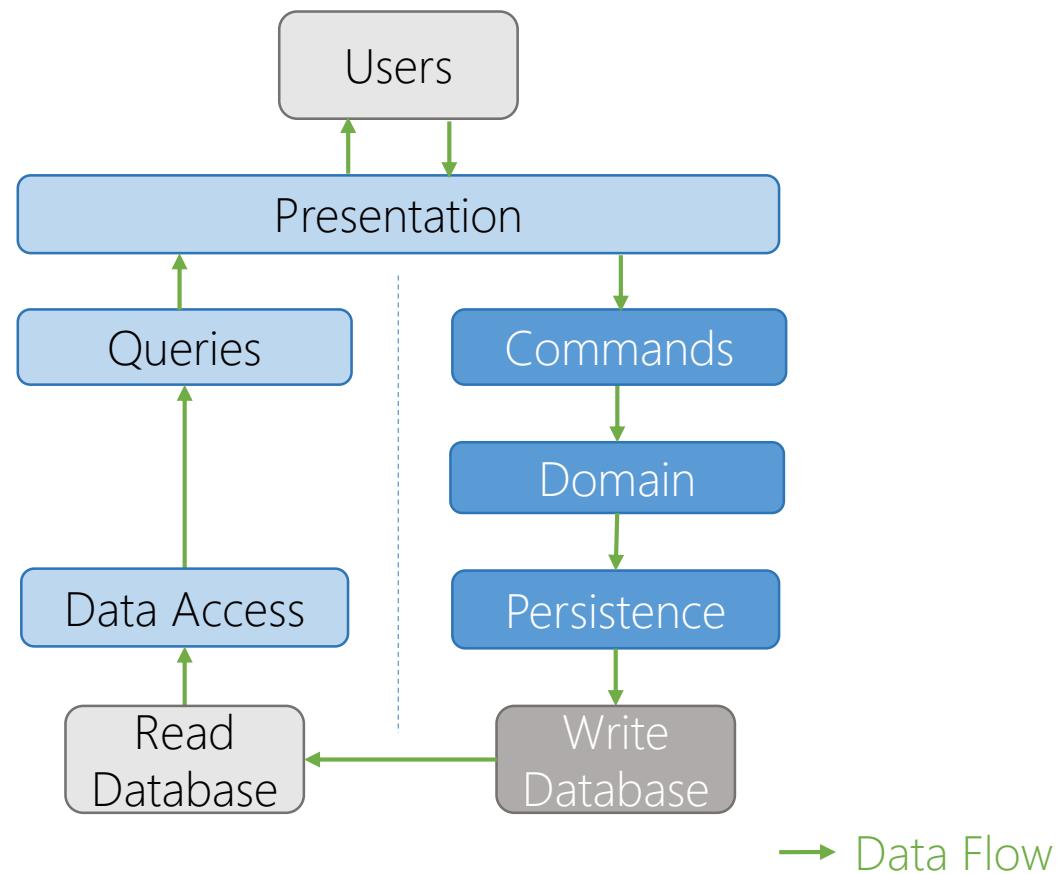
CQRS Type 1 – Single Database



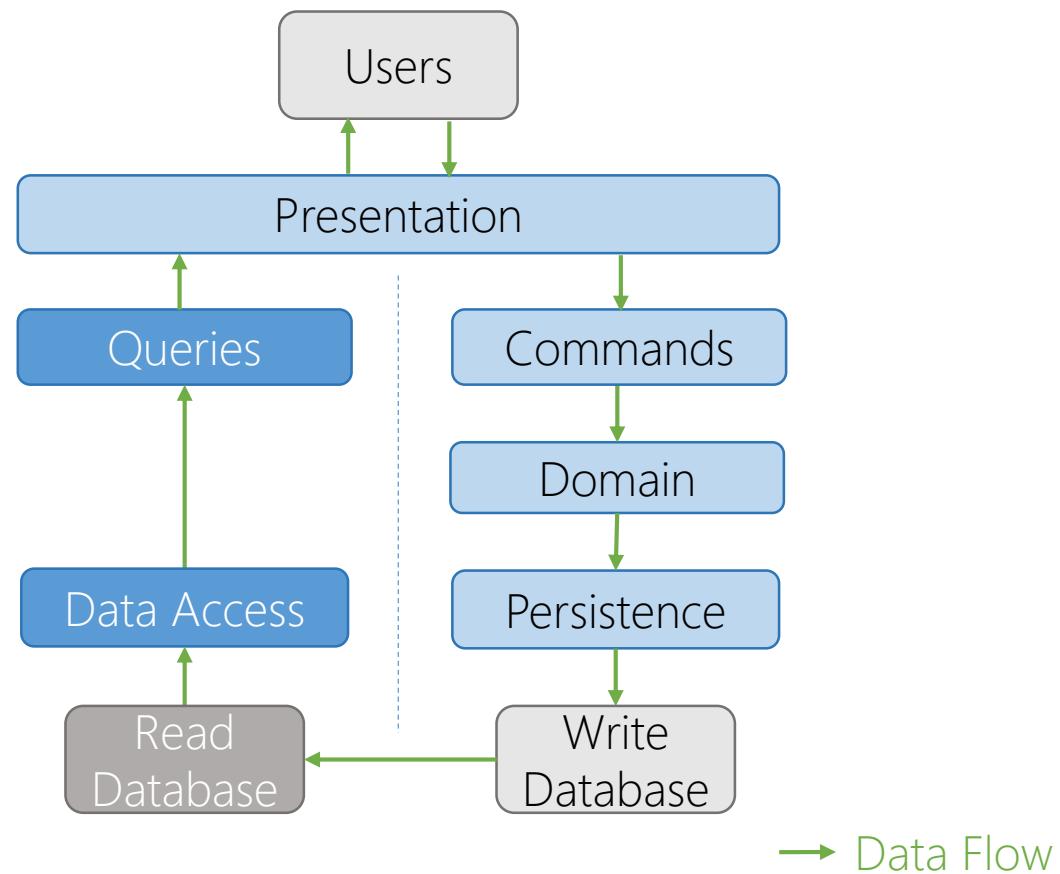
CQRS Type 2 – Read/Write Databases



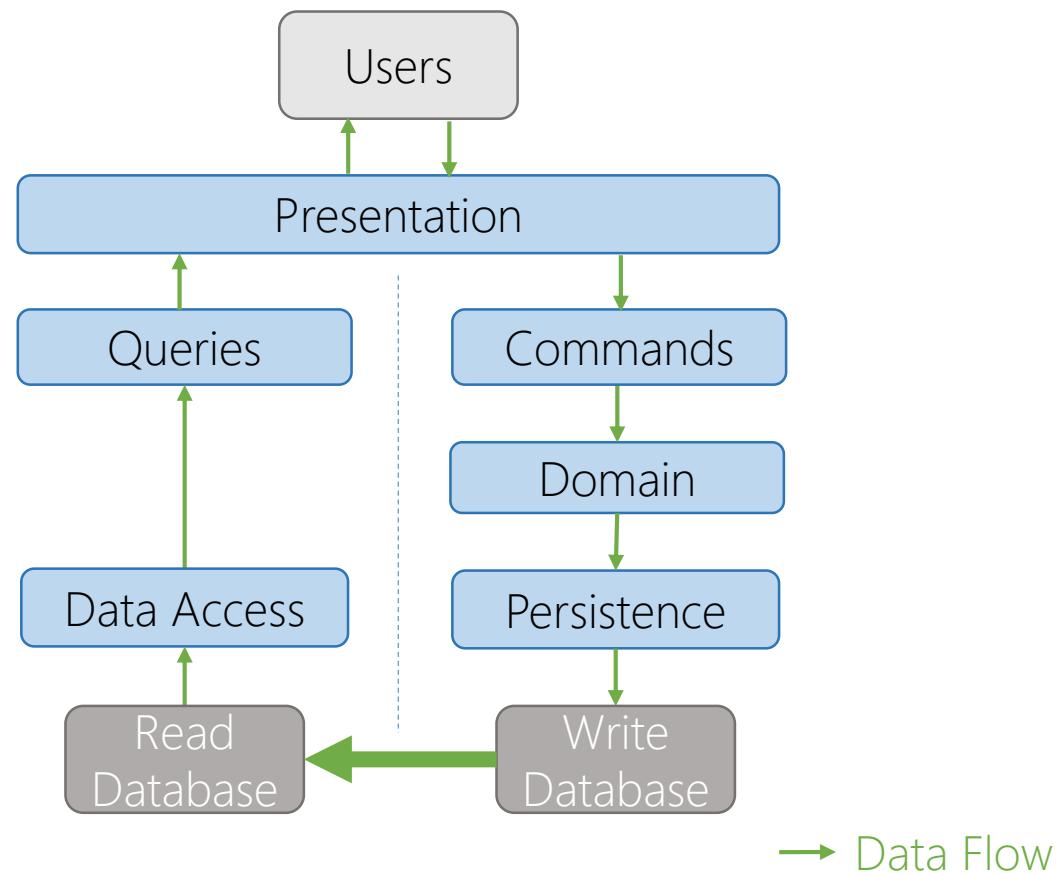
CQRS Type 2 – Read/Write Databases



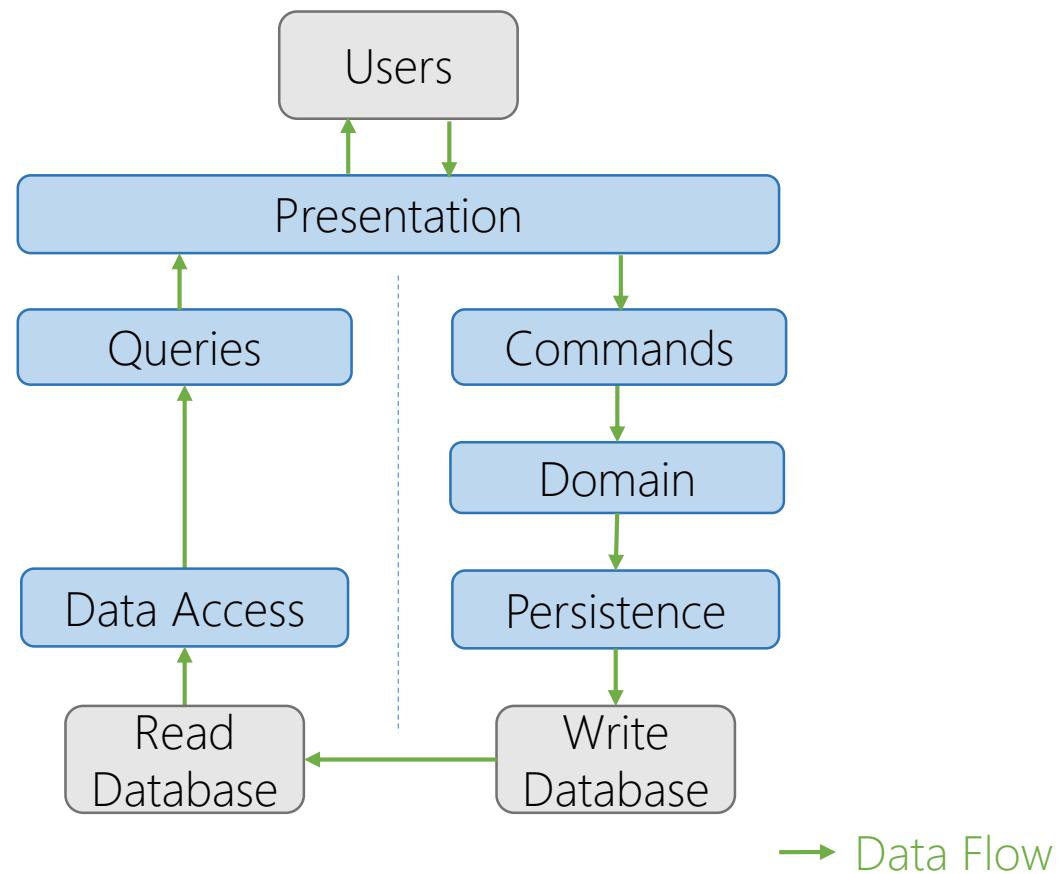
CQRS Type 2 – Read/Write Databases



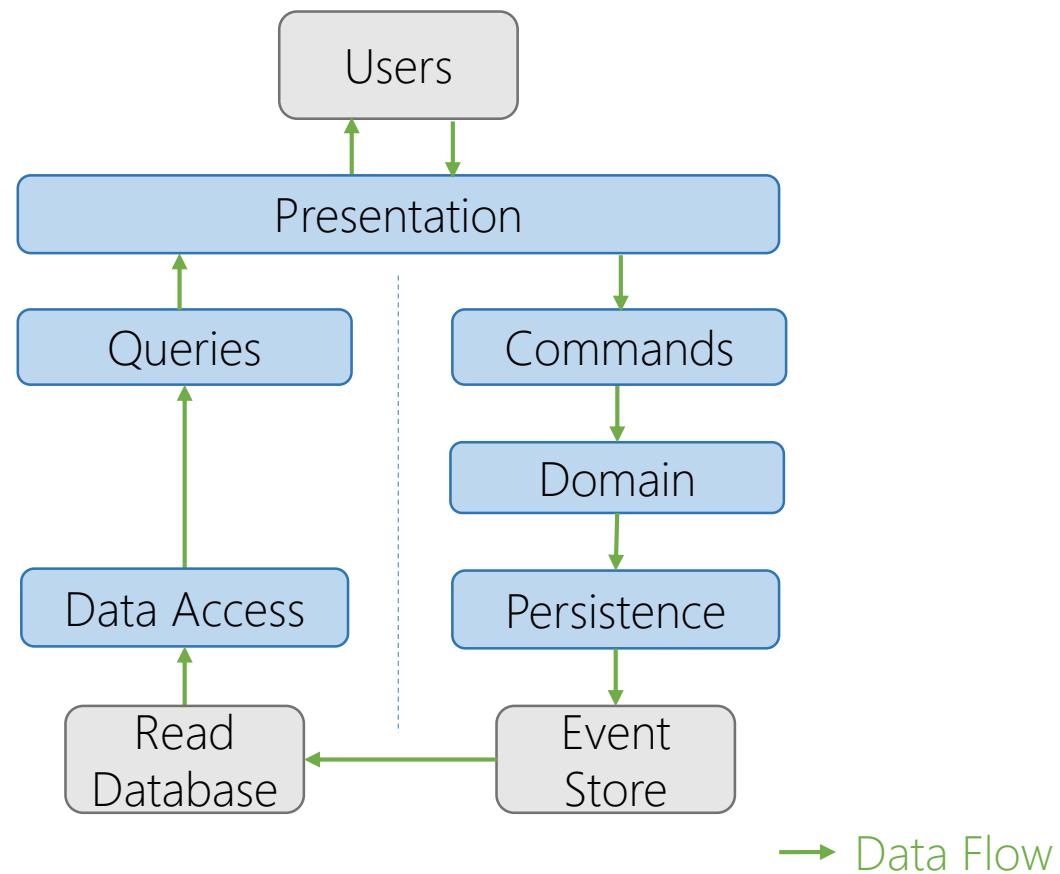
CQRS Type 2 – Read/Write Databases



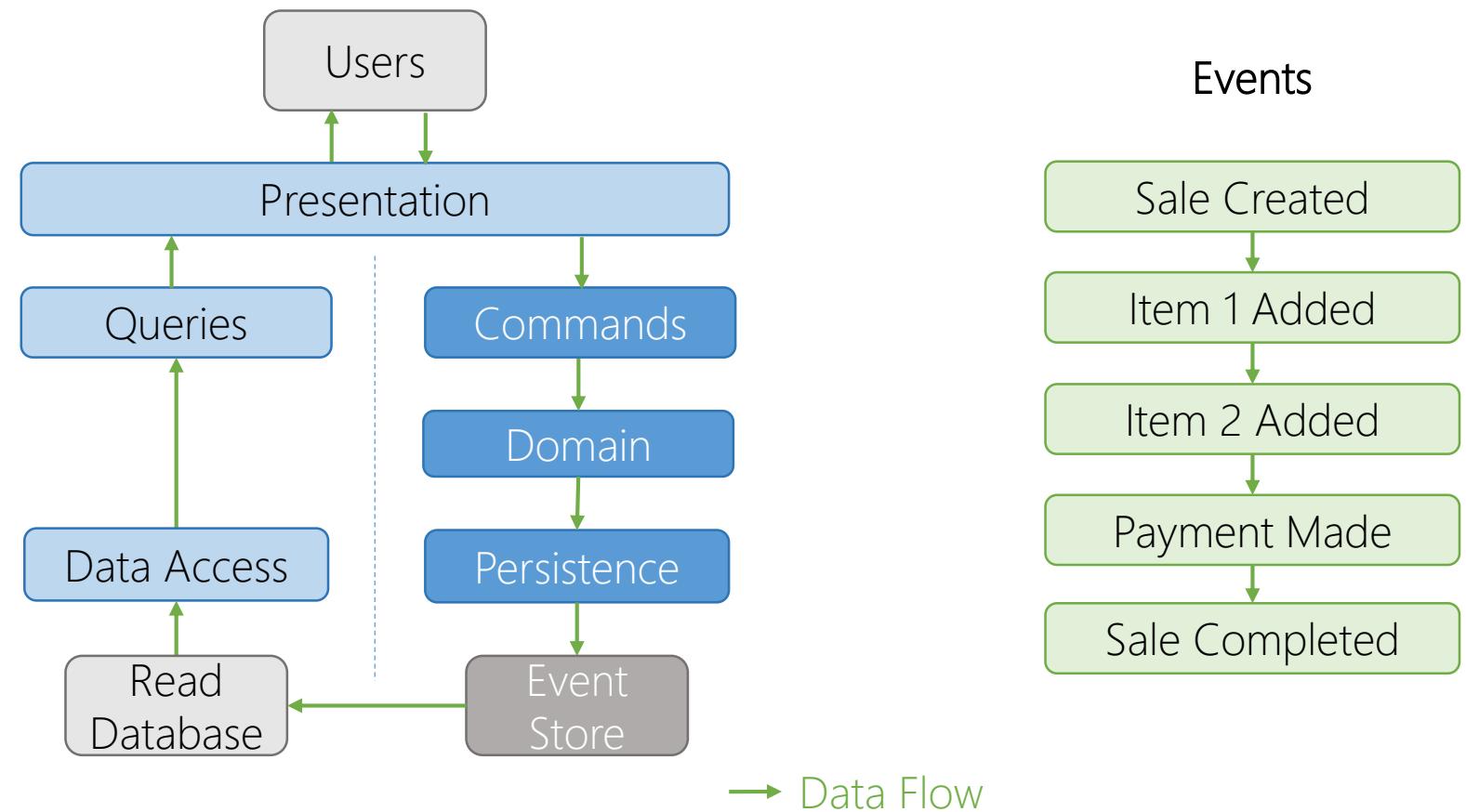
CQRS Type 2 – Read/Write Databases



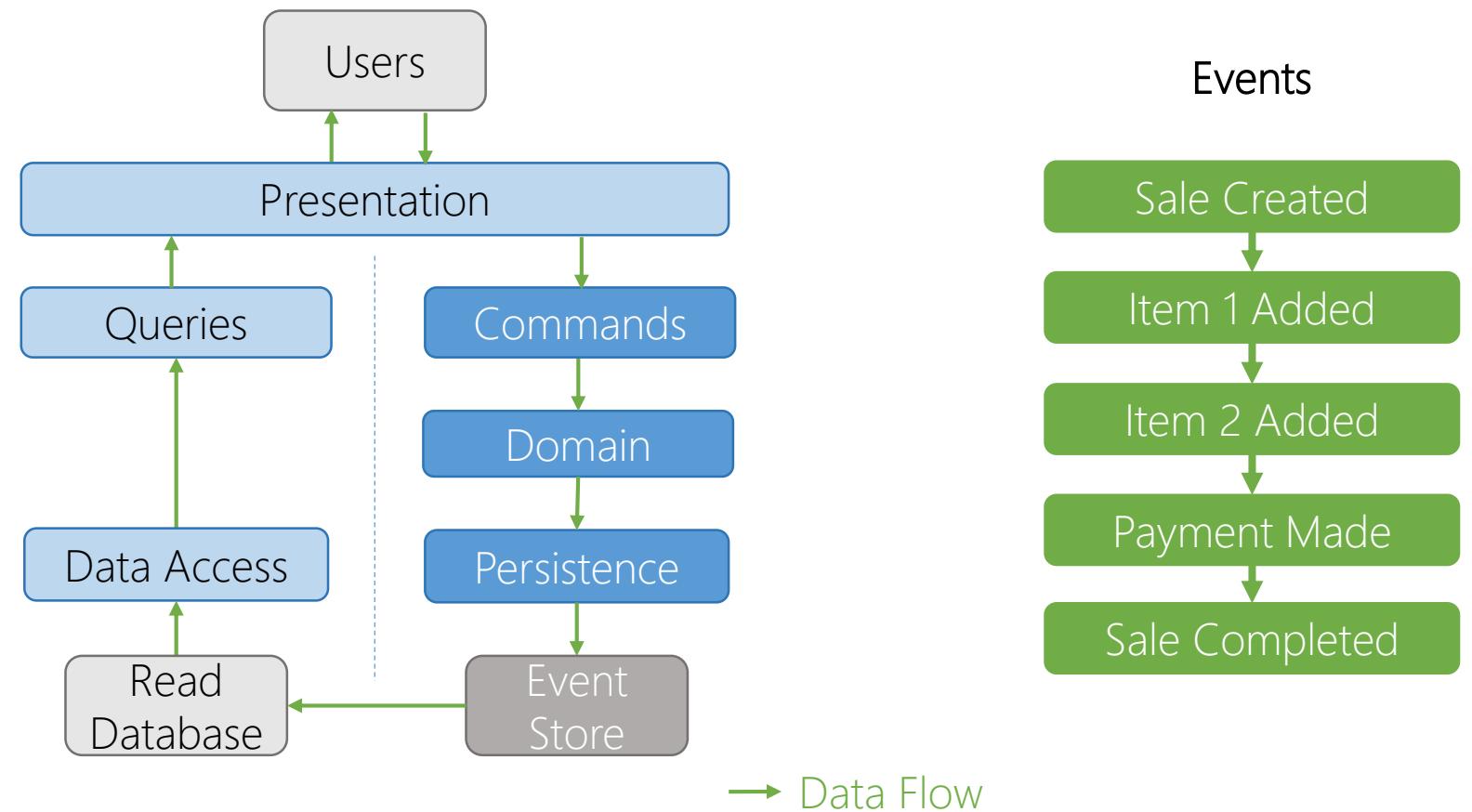
CQRS Type 3 – Event Sourcing



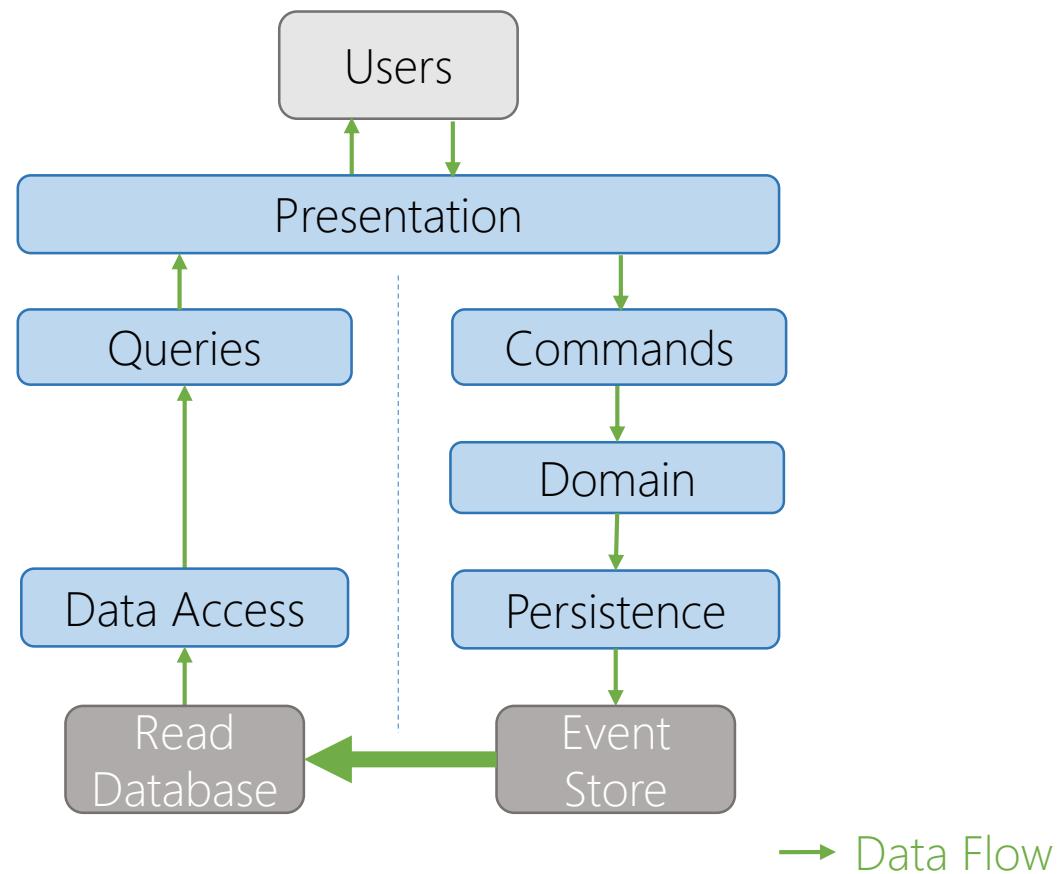
CQRS Type 3 – Event Sourcing



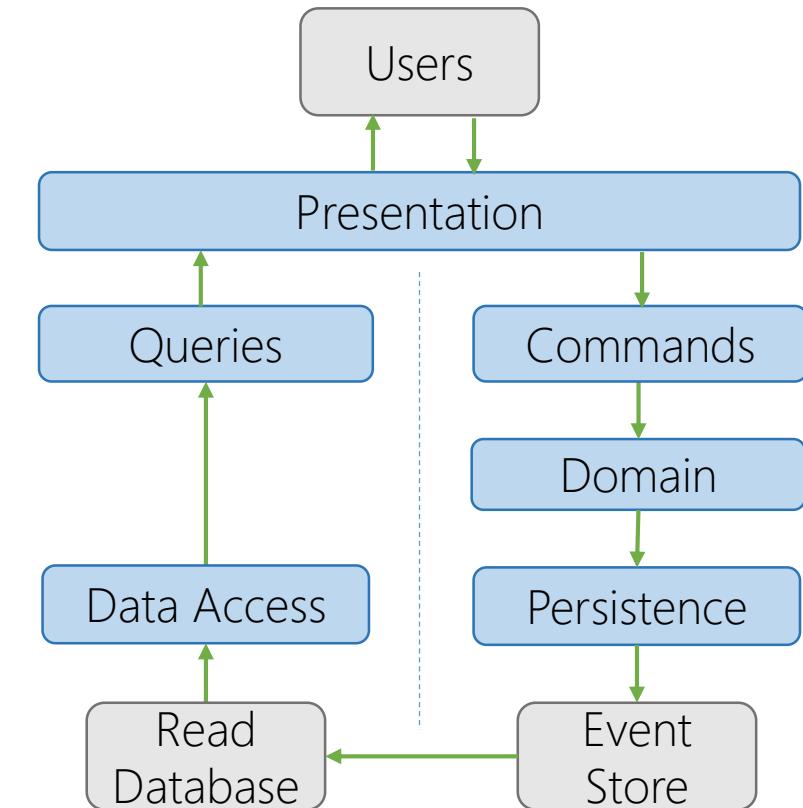
CQRS Type 3 – Event Sourcing



CQRS Type 3 – Event Sourcing

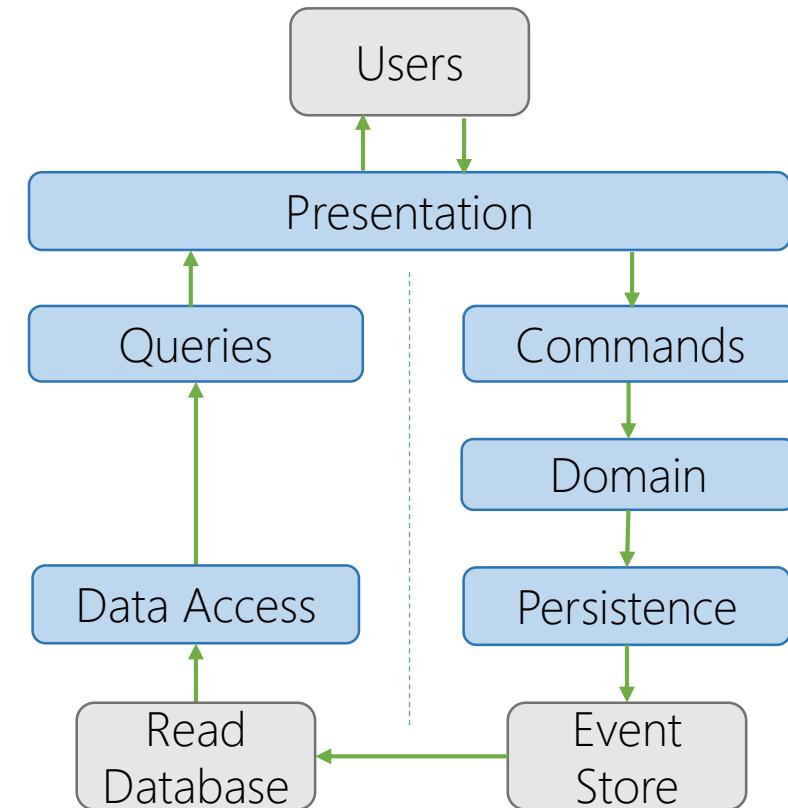


CQRS Type 3 – Event Sourcing



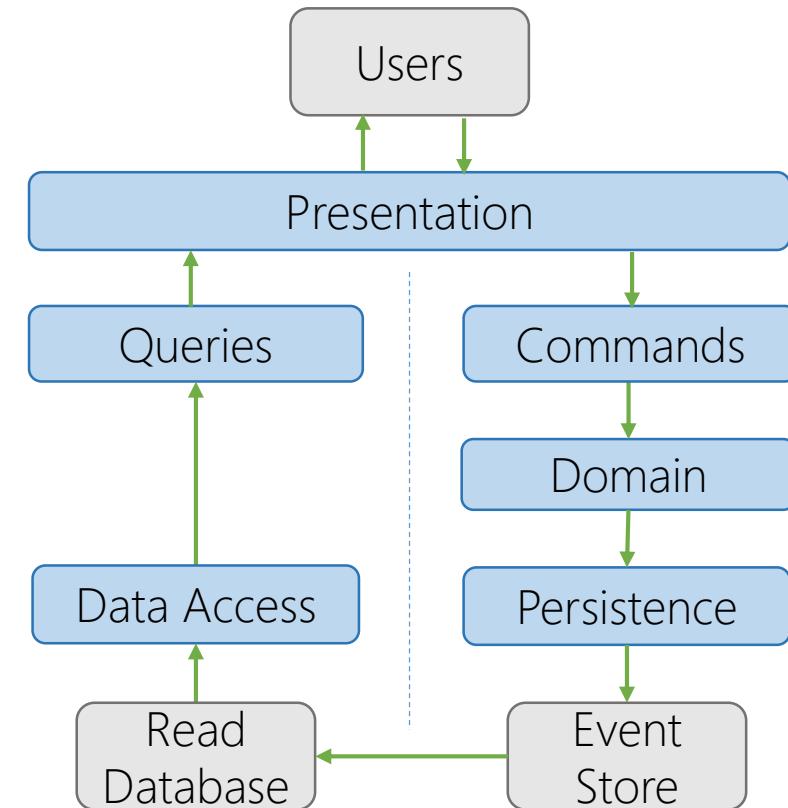
CQRS Type 3 – Event Sourcing

Complete audit trail



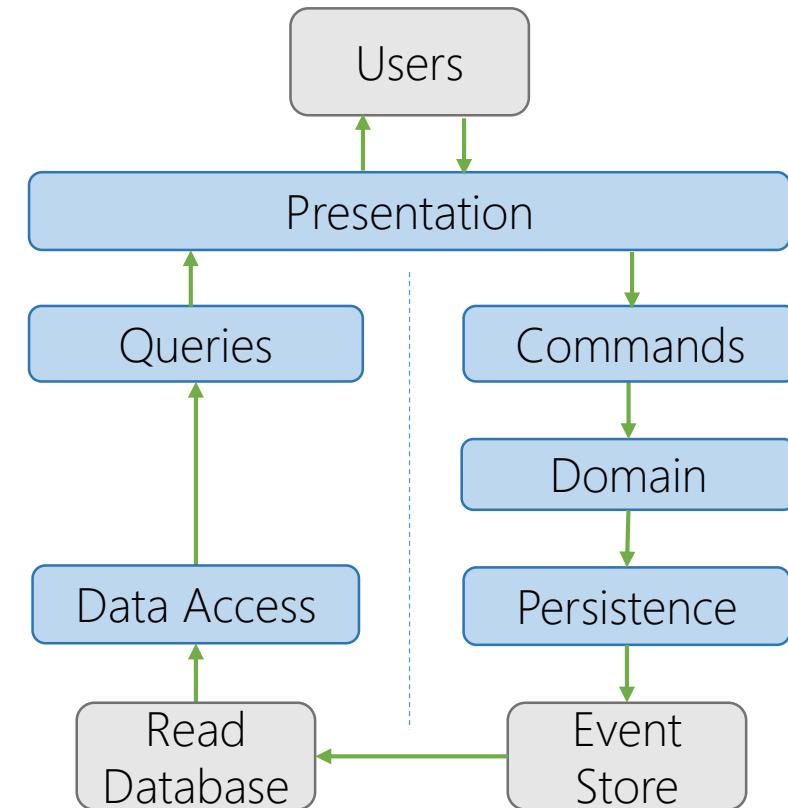
CQRS Type 3 – Event Sourcing

Complete audit trail
Point-in-time reconstruction



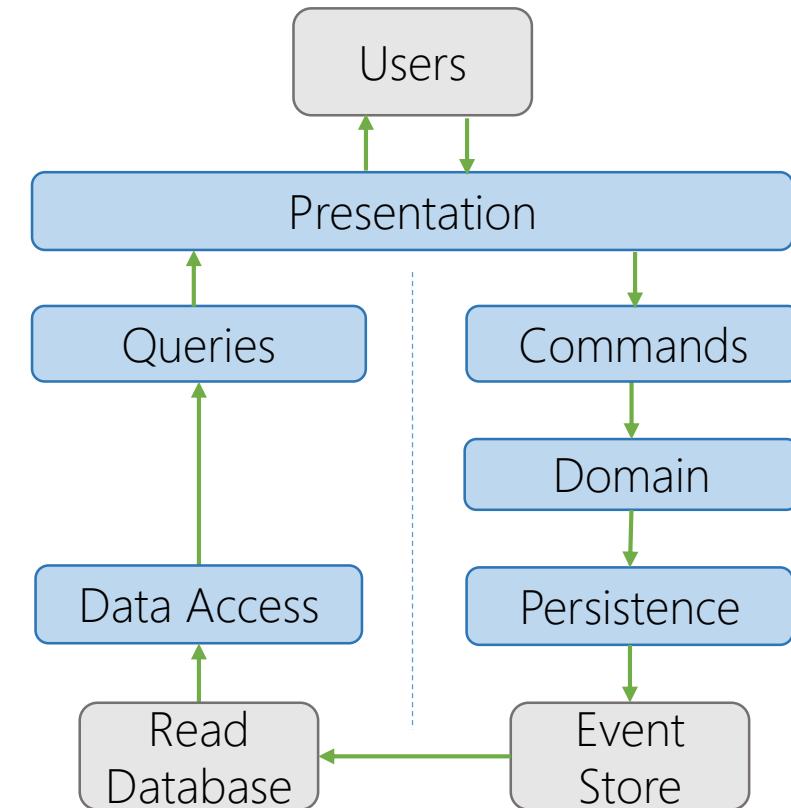
CQRS Type 3 – Event Sourcing

Complete audit trail
Point-in-time reconstruction
Replay events



CQRS Type 3 – Event Sourcing

Complete audit trail
Point-in-time reconstruction
Replay events
Rebuild production database



Why Use CQRS?

Pros

More efficient design

Why Use CQRS?

Pros

More efficient design

Simpler within each stack

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

Why Use CQRS?

Pros

- More efficient design
- Simpler within each stack
- Optimized performance

Cons

- Inconsistent across stacks
- Type 2 is more complex

Why Use CQRS?

Pros

- More efficient design
- Simpler within each stack
- Optimized performance

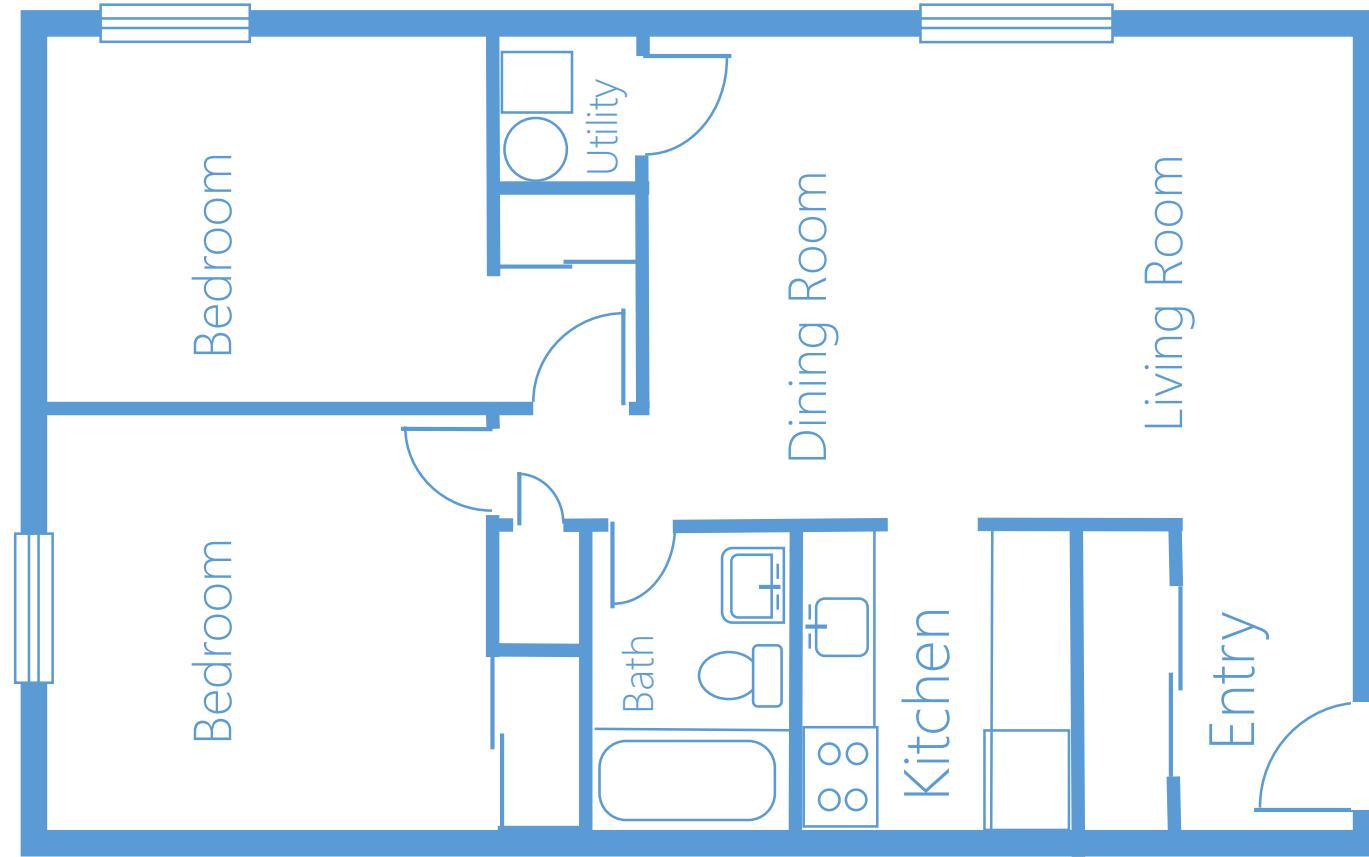
Cons

- 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





Be

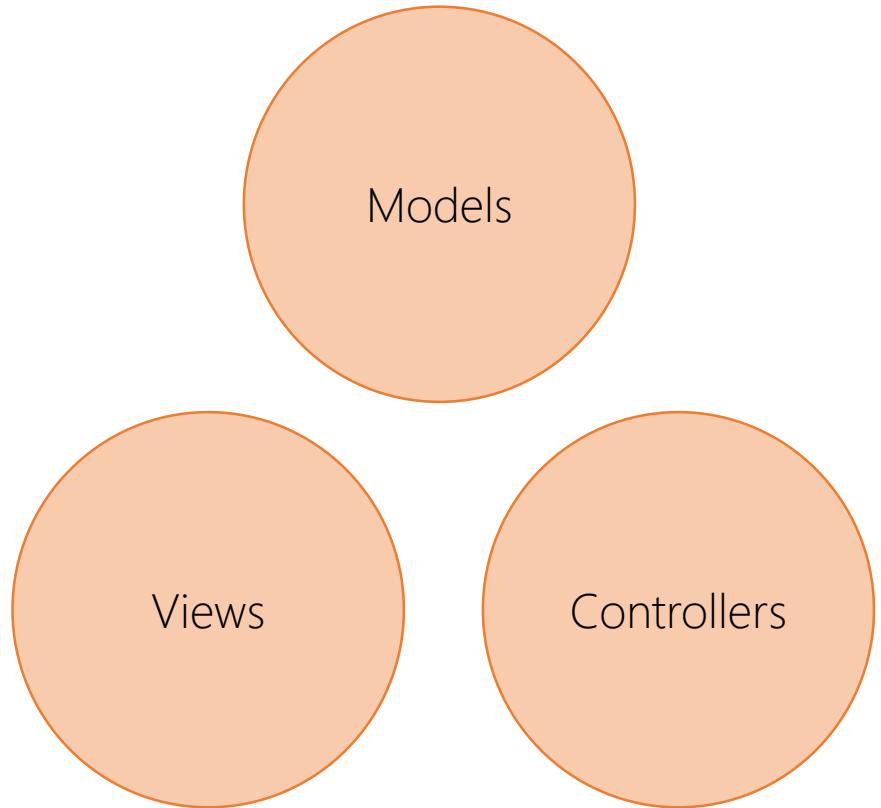


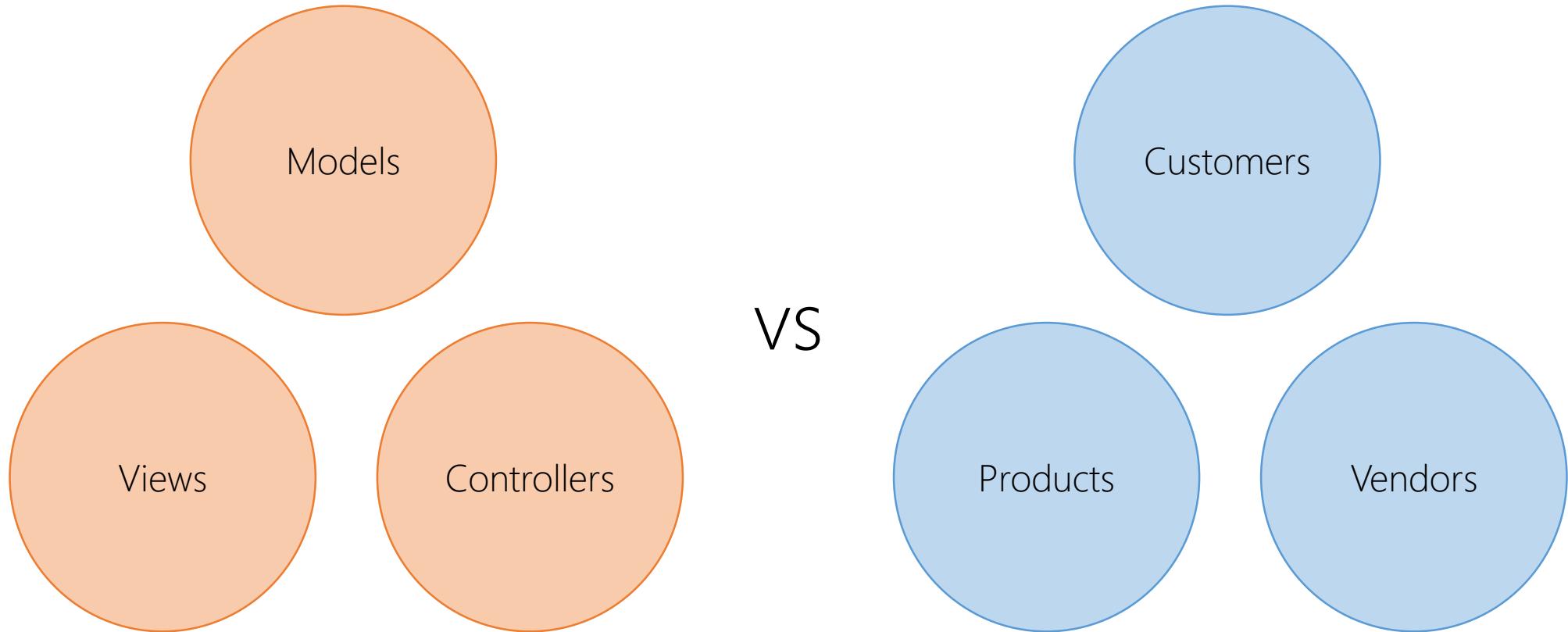
Din



Utility

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





 Content Controllers Models Scripts Views

-  Content
-  Controllers
-  Models
-  Scripts
-  Views

vs

-  Customers
-  Employees
-  Products
-  Sales
-  Vendors

So what?





VS



Why Use Functional Organization

Pros

Spatial locality

Why Use Functional Organization

Pros

Spatial locality

Easy to navigate

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

Why Use Functional Organization

Pros

Spatial locality

Easy to navigate

Avoid vendor lock-in

Cons

Lose framework conventions

Lose automatic scaffolding

Why Use Functional Organization

Pros

Spatial locality

Easy to navigate

Avoid vendor lock-in

Cons

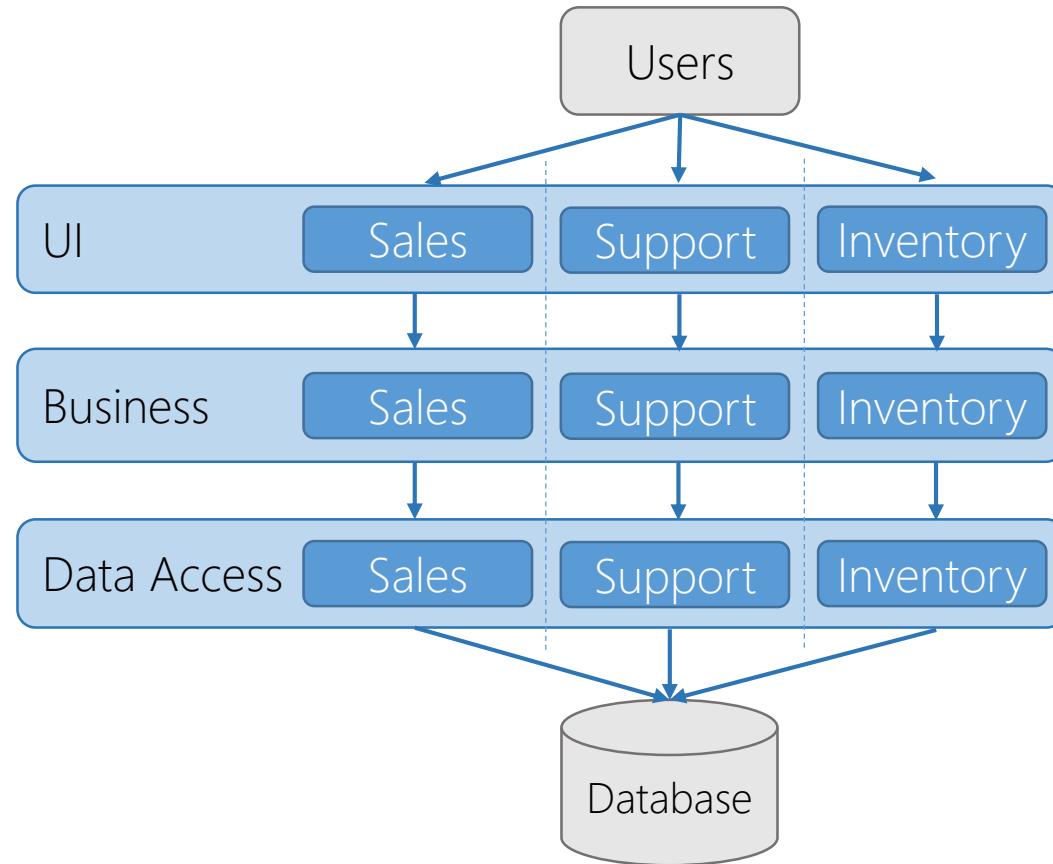
Lose framework conventions

Lose automatic scaffolding

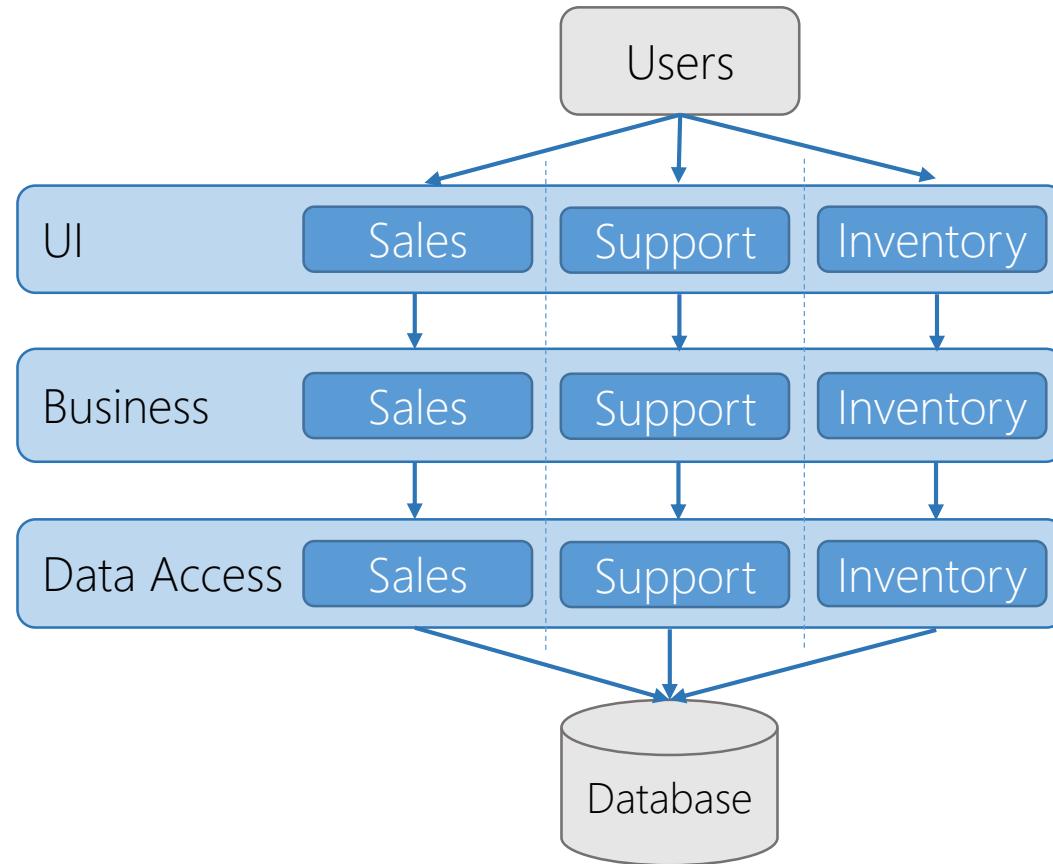
Categorical is easier at first

Microservices

Components



Components



Problem Domain

Sales

Sales Opportunity

Contact

Sales Person

Product

Sales Territory

Support

Support Ticket

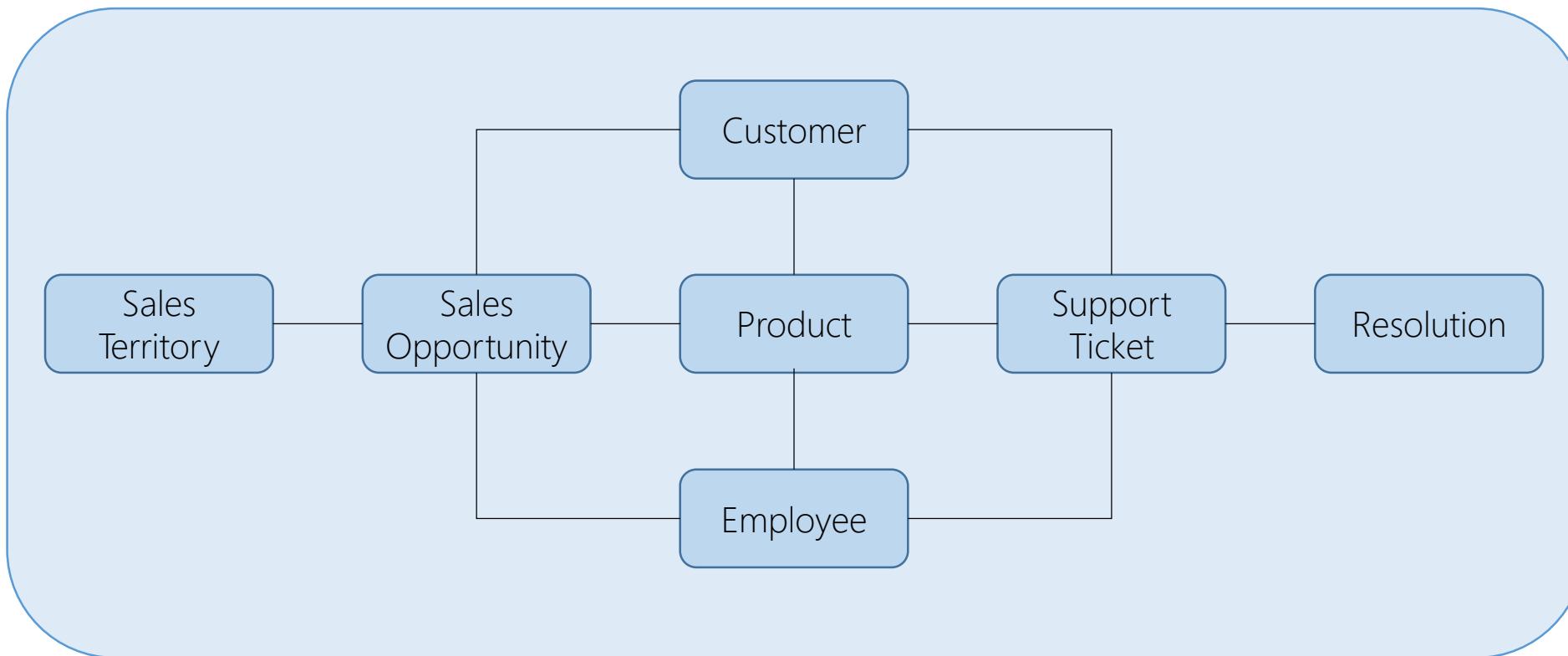
Customer

Support Person

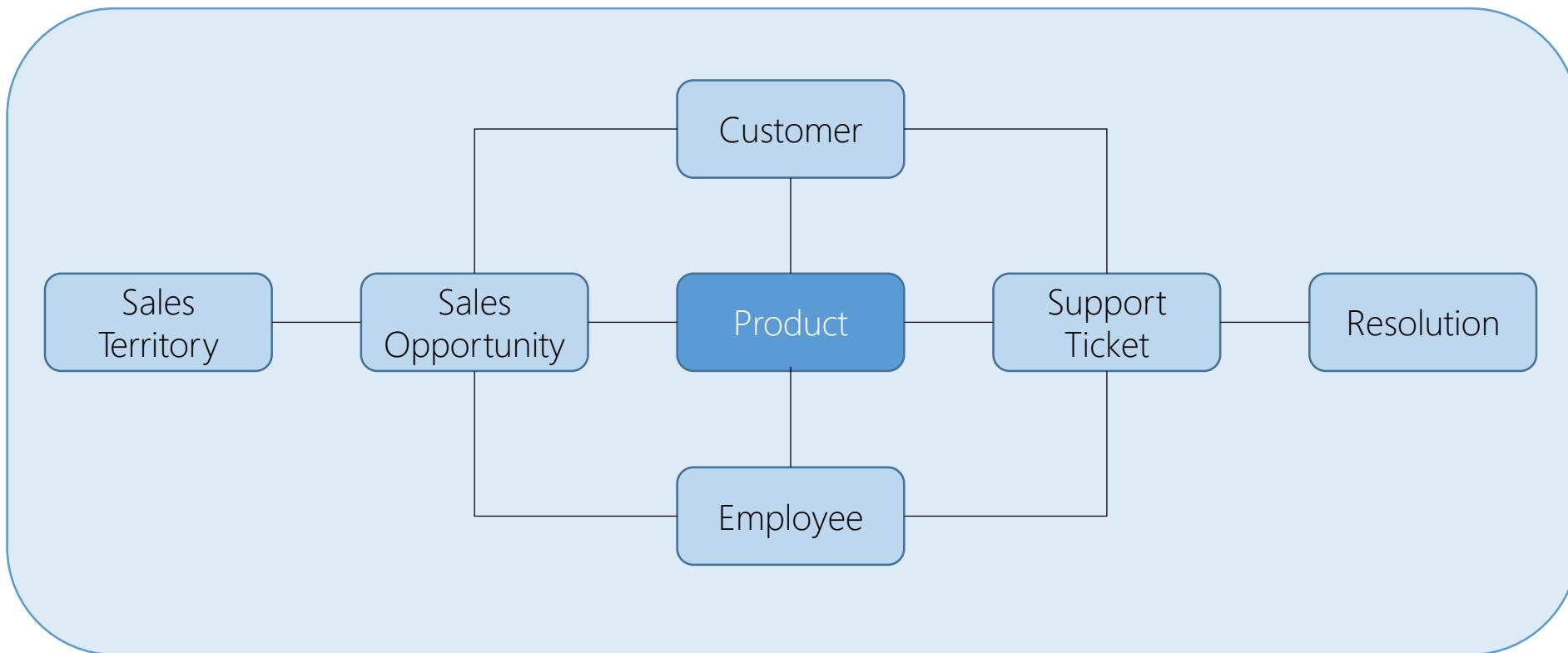
Product

Resolution

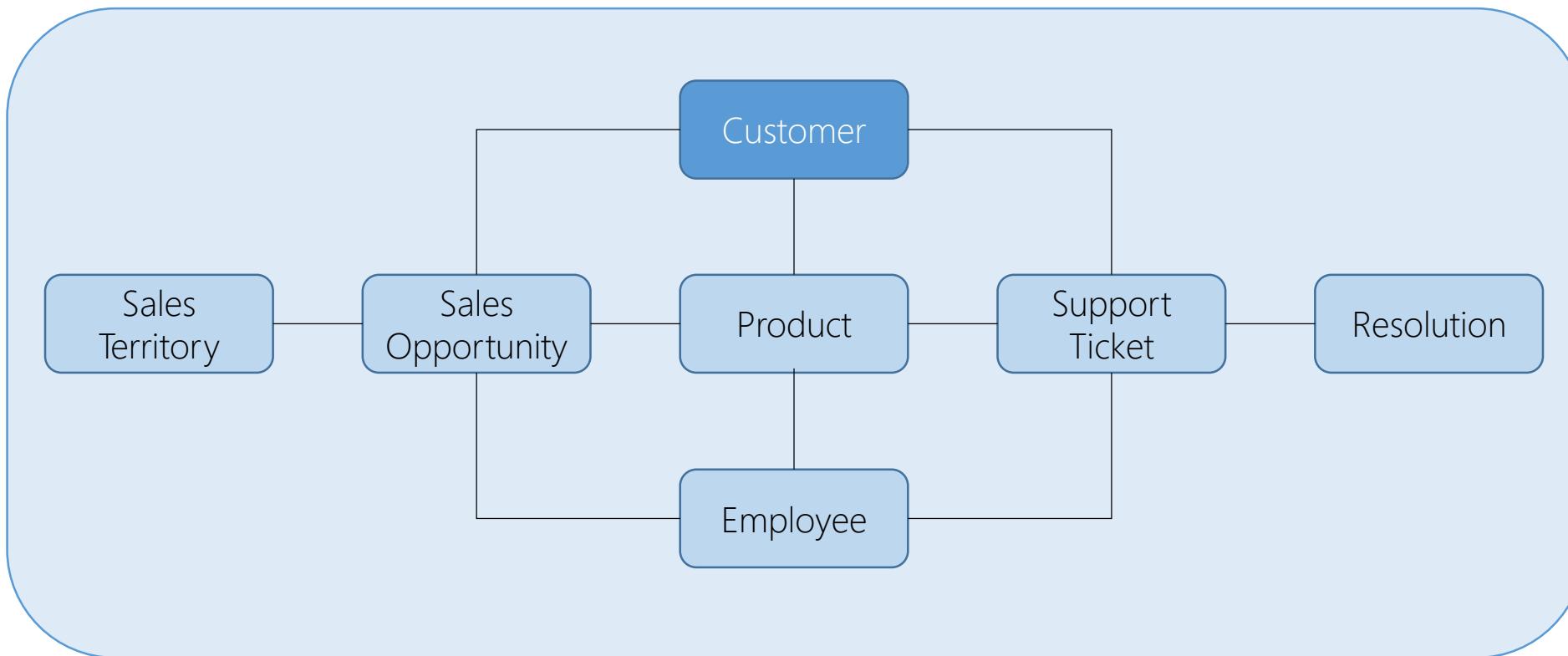
Single Domain Model



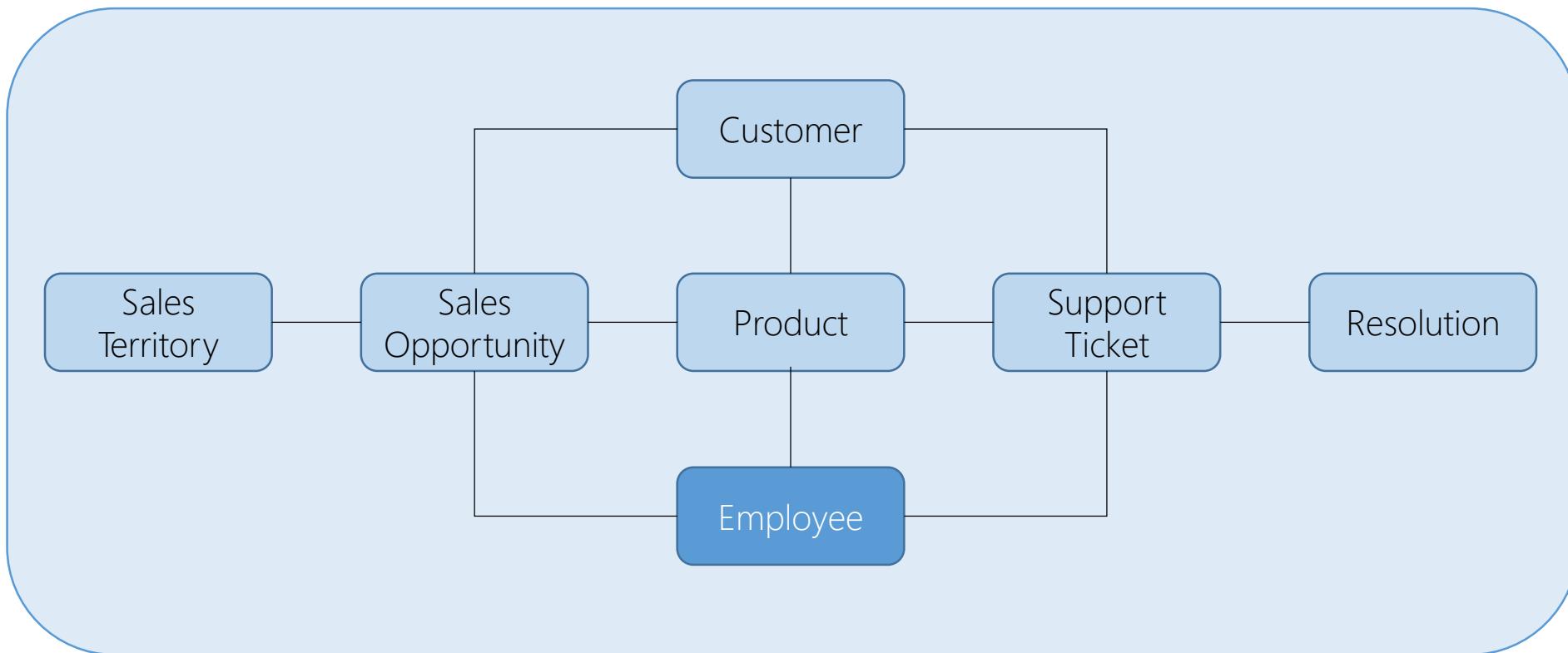
Single Domain Model



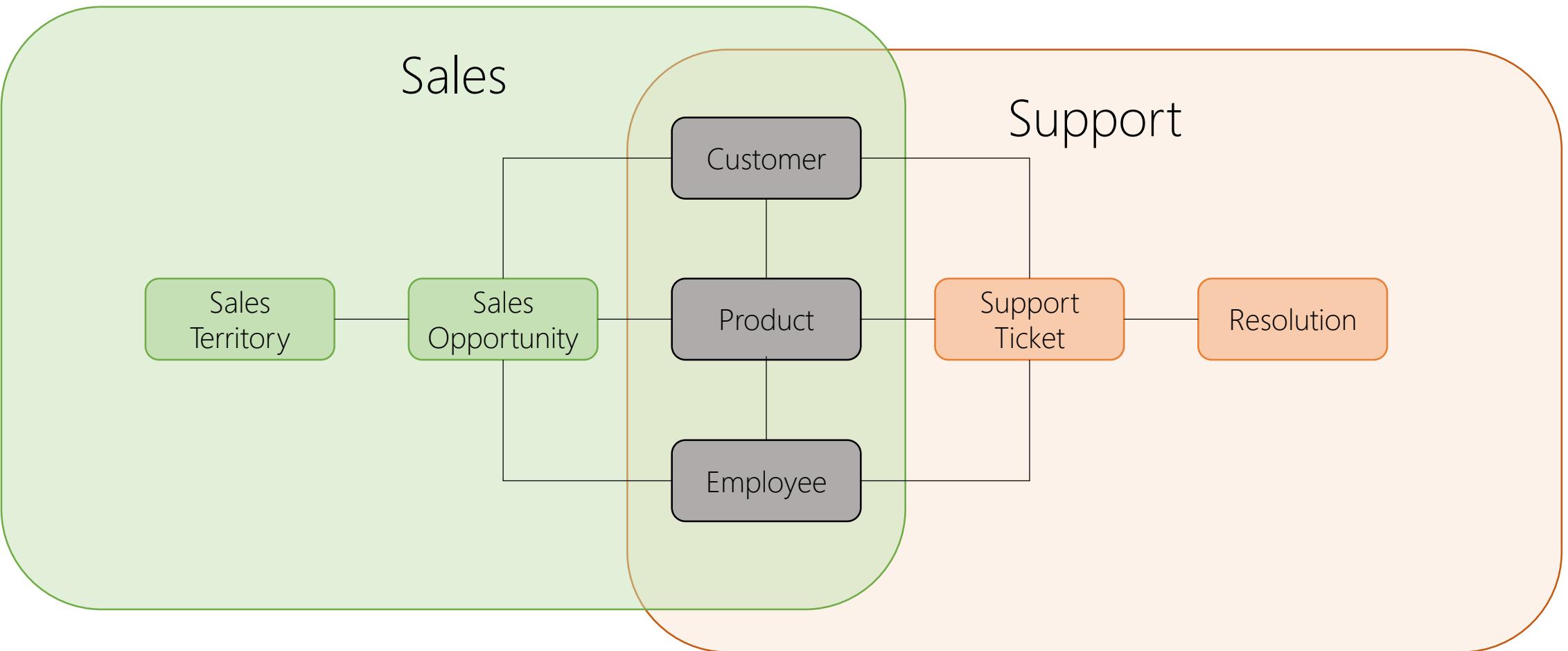
Single Domain Model



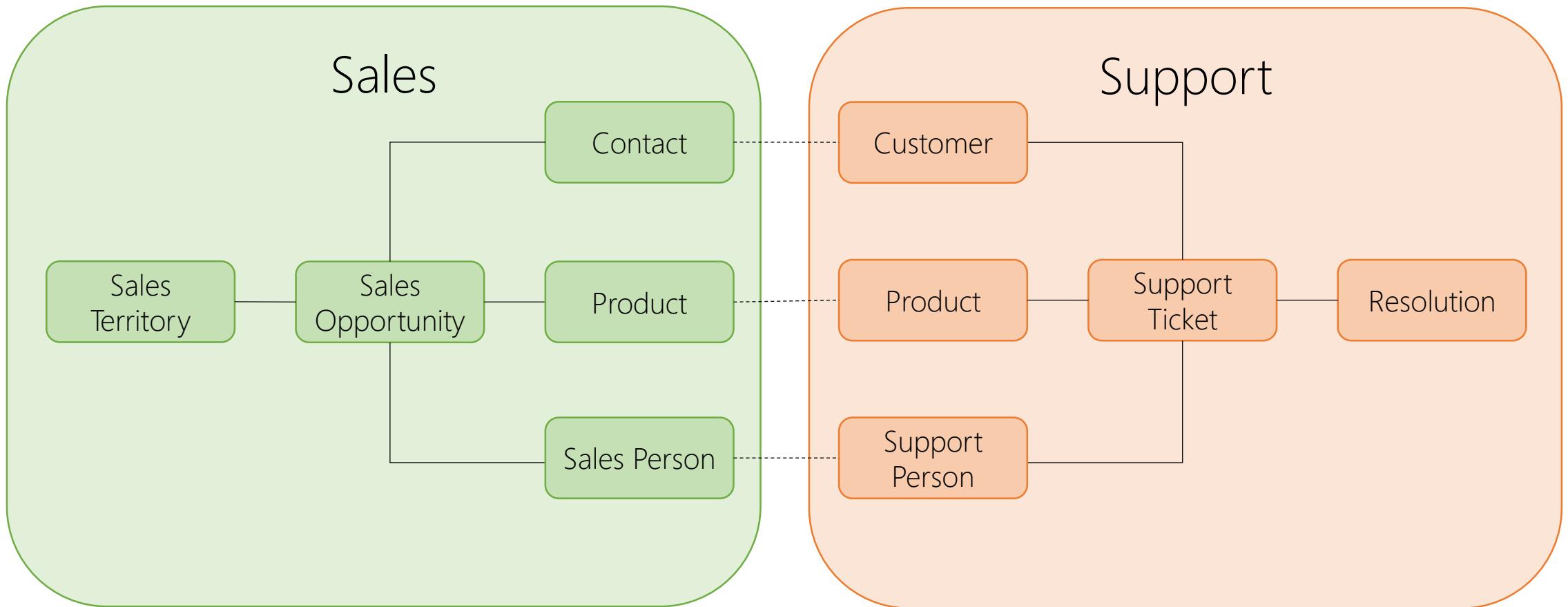
Single Domain Model



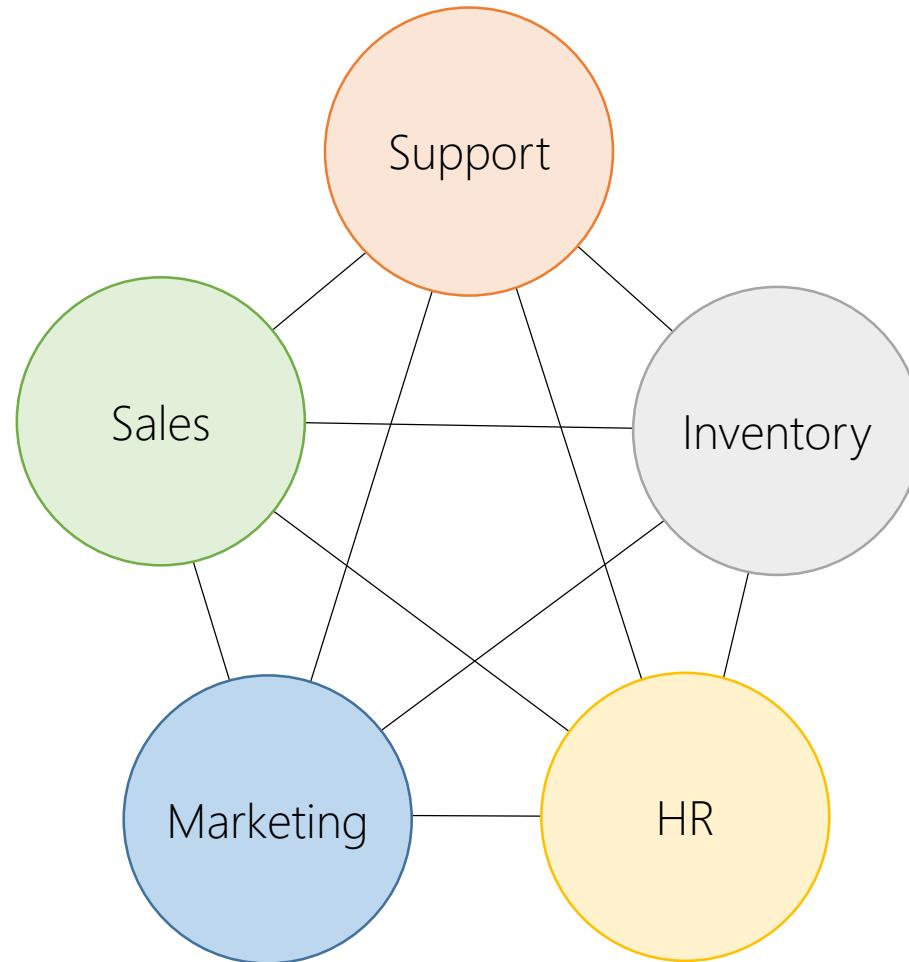
Overlapping Contexts



Bounded Contexts

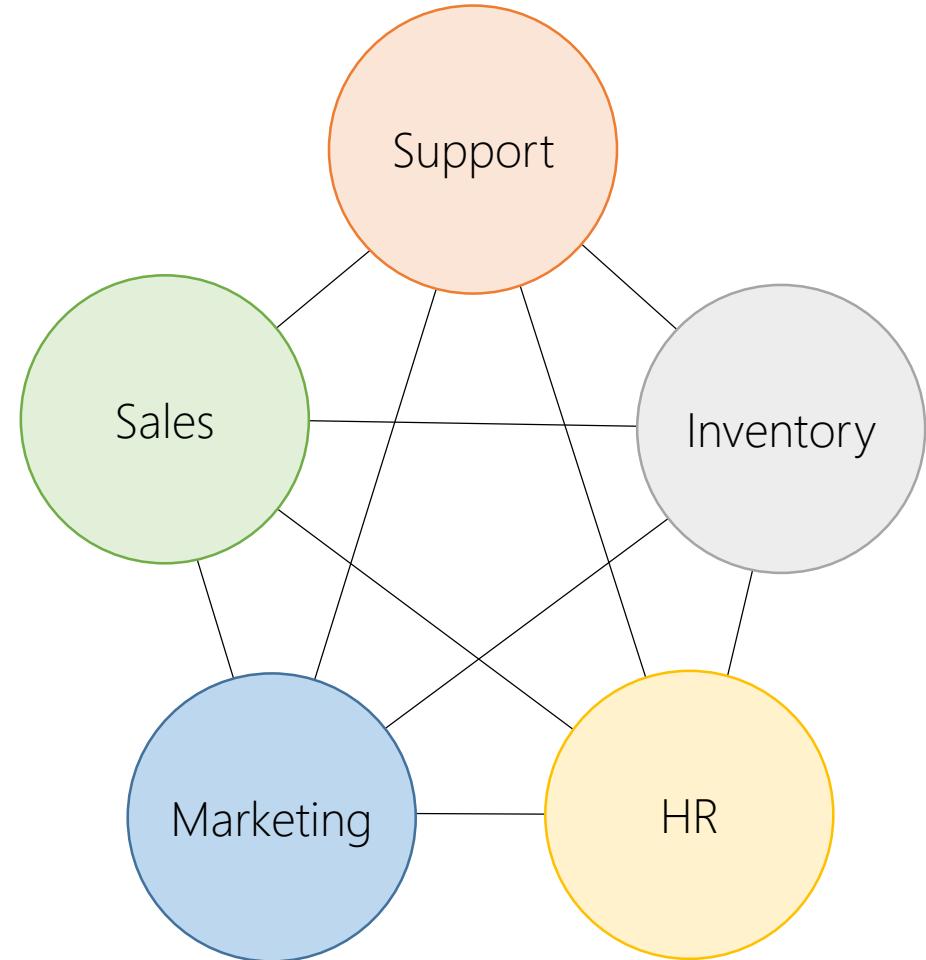


Microservice Architectures



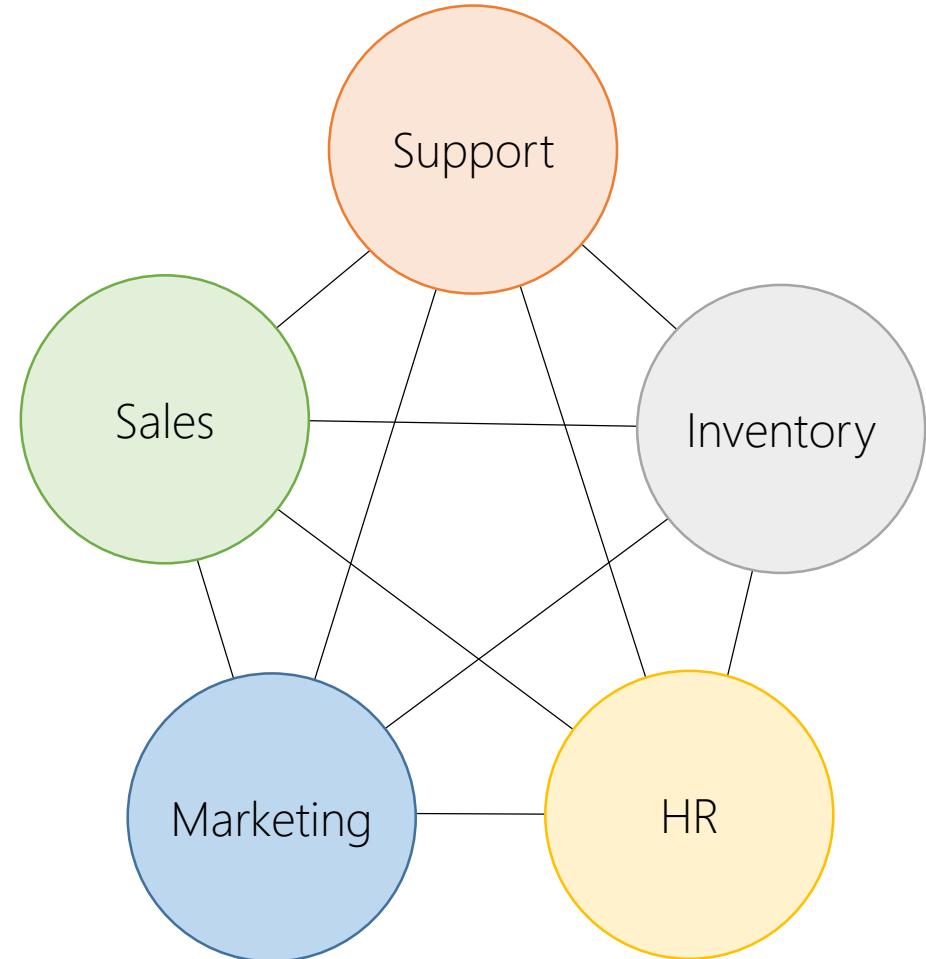
Microservice Architectures

Subdivide system



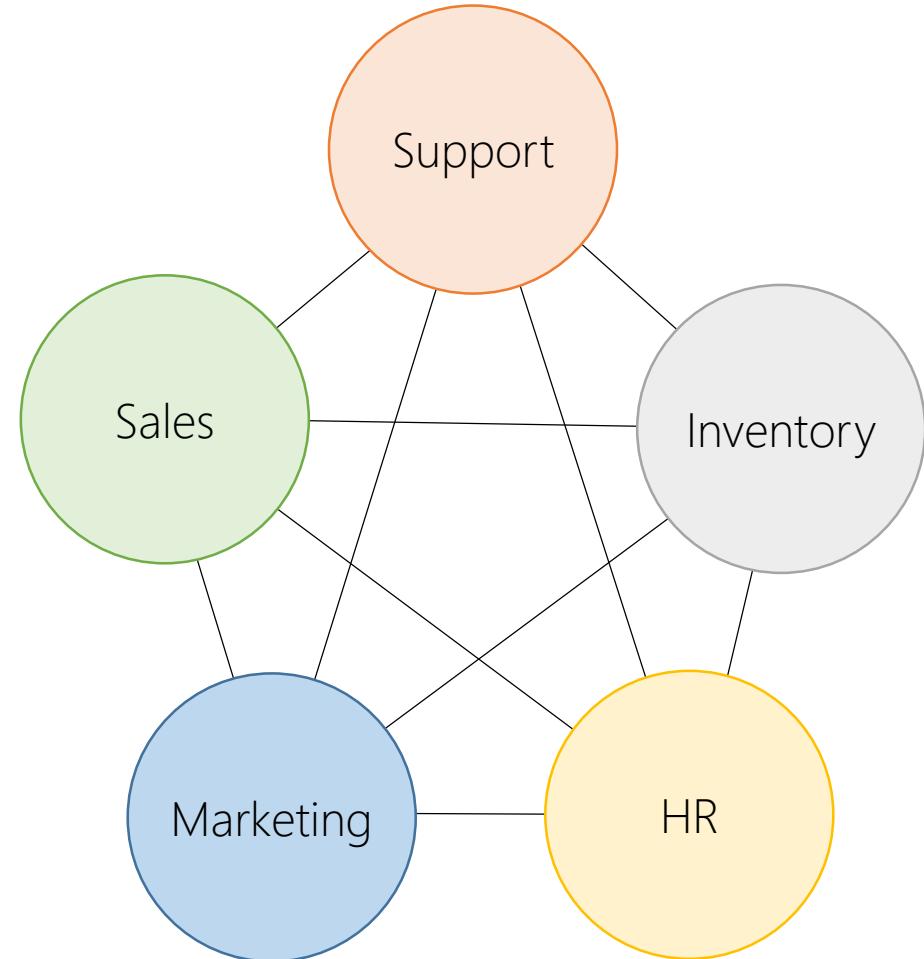
Microservice Architectures

Subdivide system
Light-weight APIs



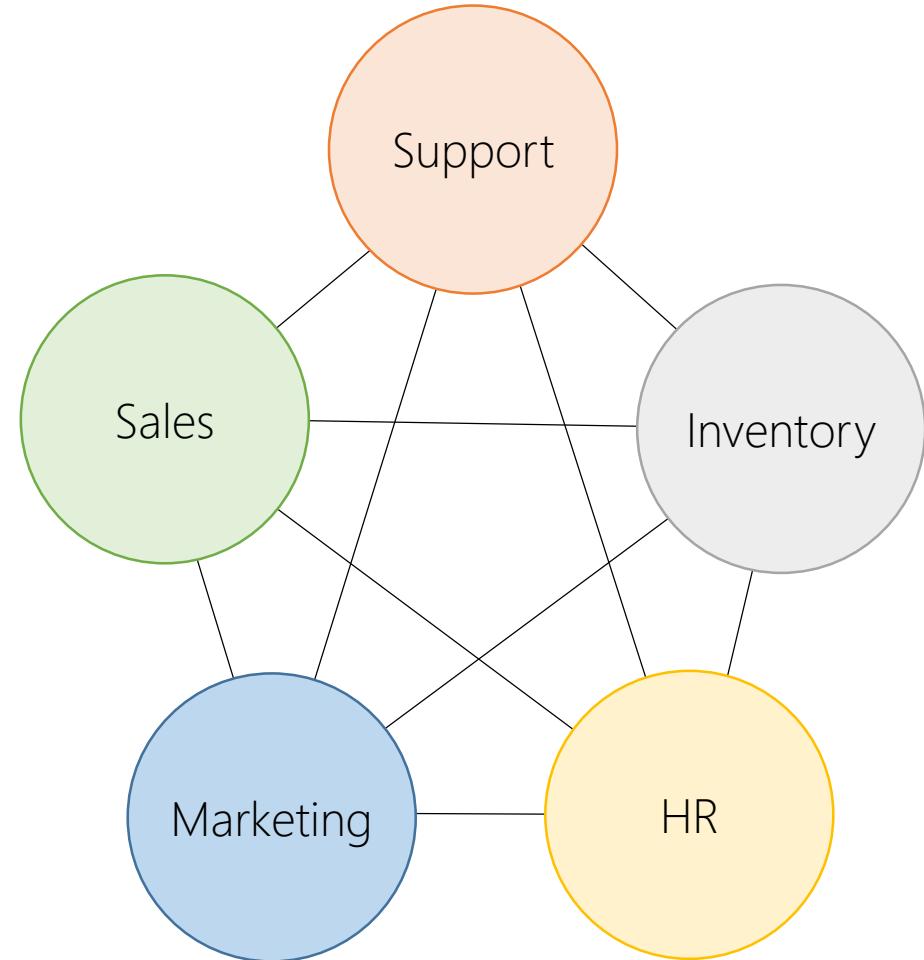
Microservice Architectures

Subdivide system
Light-weight APIs
Small teams



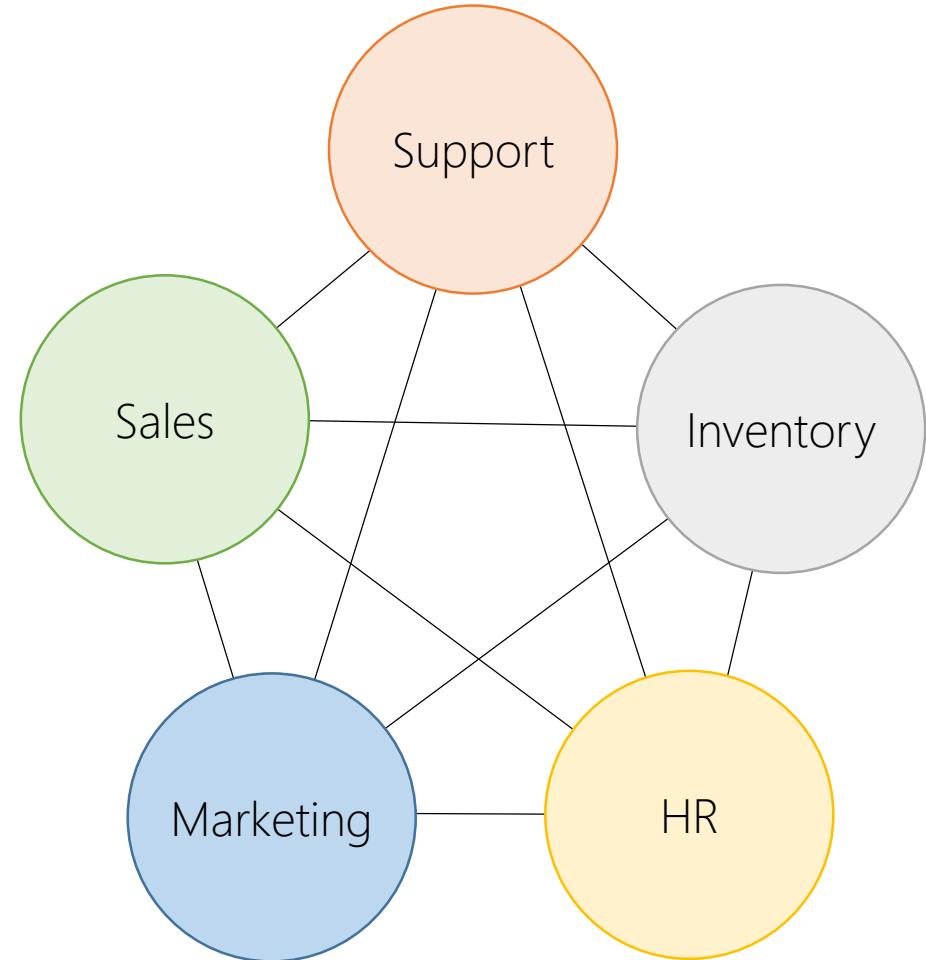
Microservice Architectures

Independent



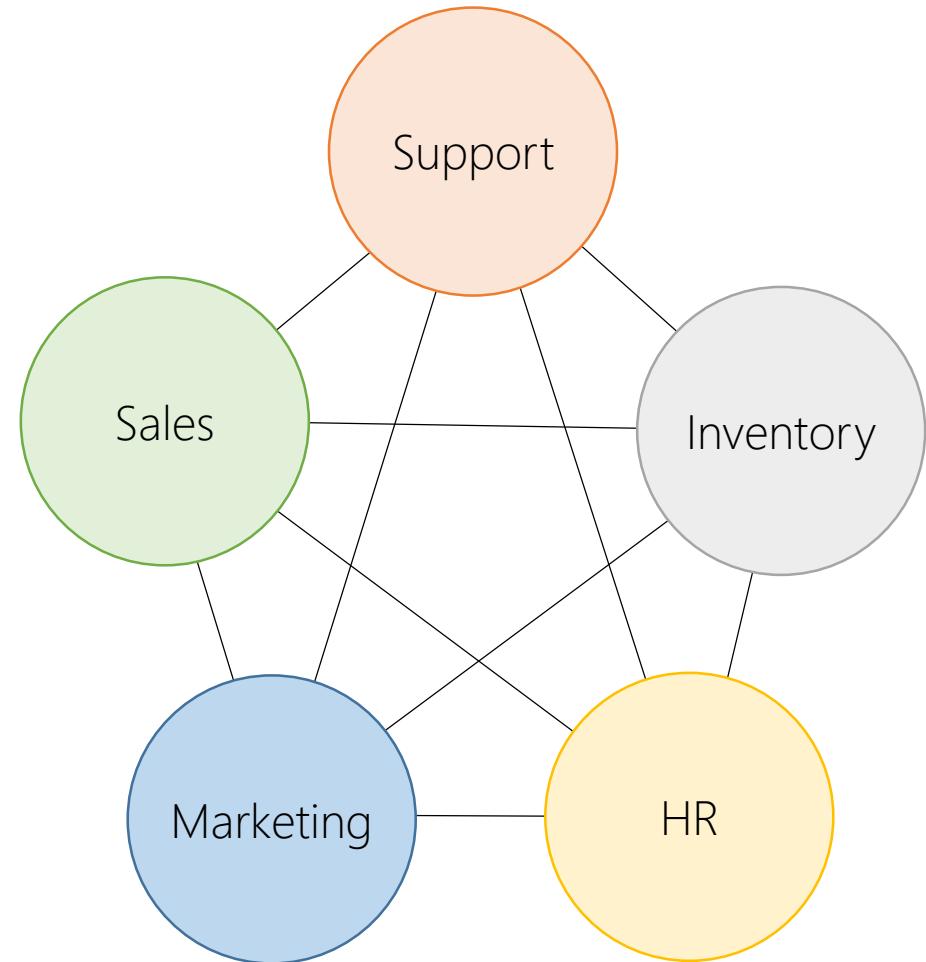
Microservice Architectures

Independent
Similar to SOA



Microservice Architectures

Independent
Similar to SOA
Size matters



Why Use Microservices?

Pros

Less cost for large domains

Why Use Microservices?

Pros

Less cost for large domains

Smaller teams

Why Use Microservices?

Pros

Less cost for large domains

Smaller teams

Independence

Why Use Microservices?

Pros

Less cost for large domains

Smaller teams

Independence

Cons

Only for large domains

Why Use Microservices?

Pros

Less cost for large domains

Smaller teams

Independence

Cons

Only for large domains

Higher up-front cost

Why Use Microservices?

Pros

Less cost for large domains

Smaller teams

Independence

Cons

Only for large domains

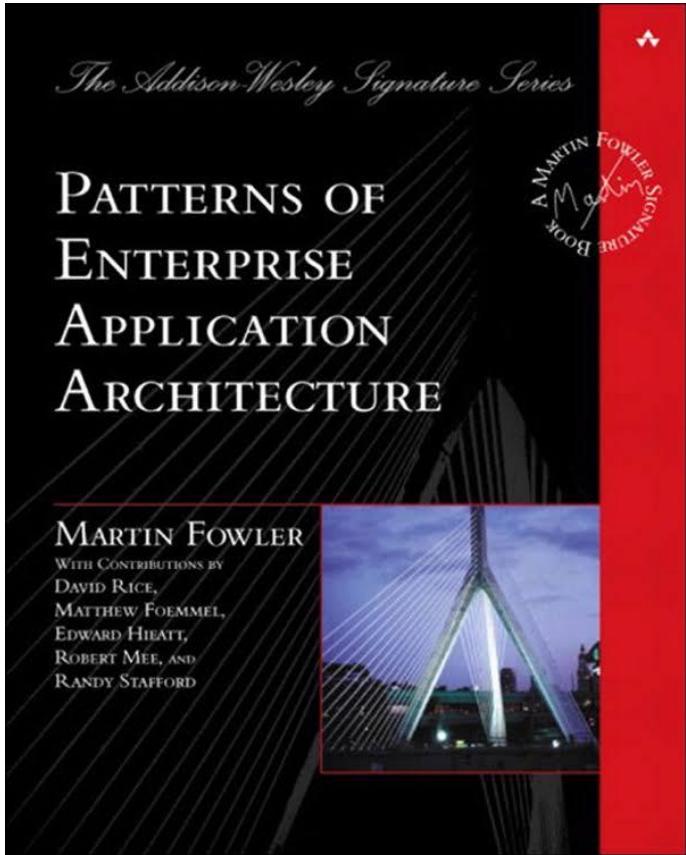
Higher up-front cost

Distributed system costs

Code Demo

Where to Go Next?

Where to Go Next?



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

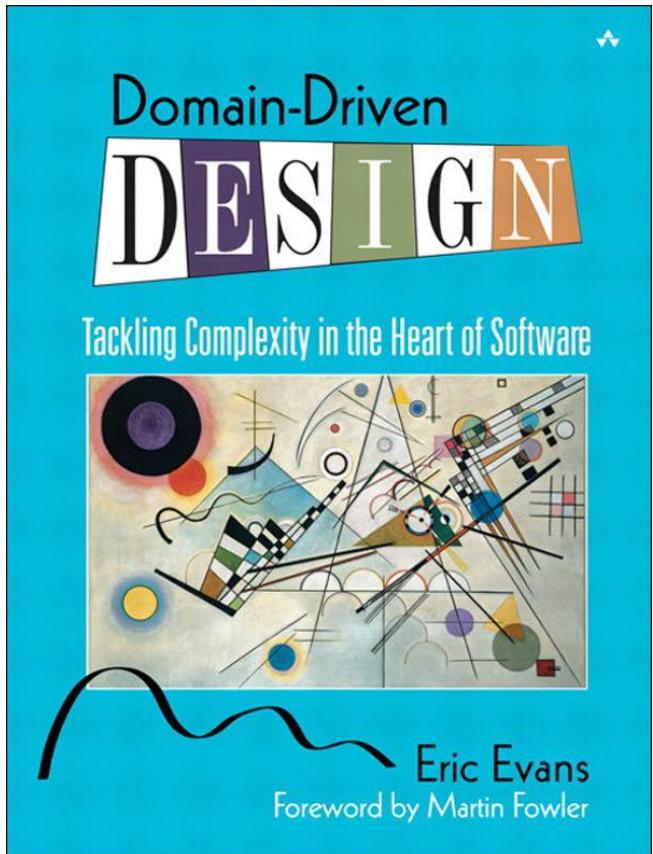
Episode 20 - Clean Tests

<http://cleancoders.com/>



Robert C. Martin

Where to Go Next?



Eric Evans

Where to Go Next?



Greg Young



Udi Dahan

Where to Go Next?

Articles

Courses

Presentations

Source Code

Videos

www.matthewrenze.com

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.



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

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.



Clean Architecture: Patterns, Practices, and Principles

INTRODUCTION



Matthew Renze
SOFTWARE CONSULTANT

@matthewrenze www.matthewrenze.com



www.pluralsight.com/authors/matthew-renze

Conclusion

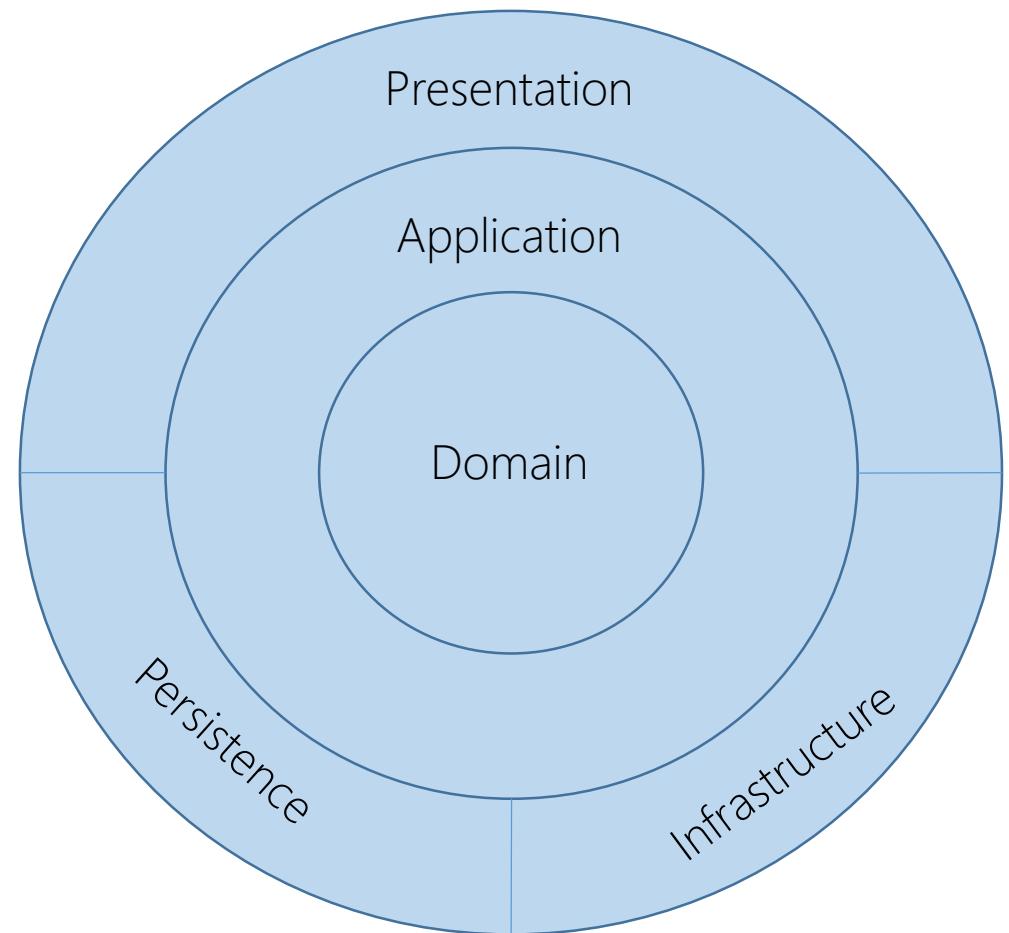
Summary

Focus on the inhabitants



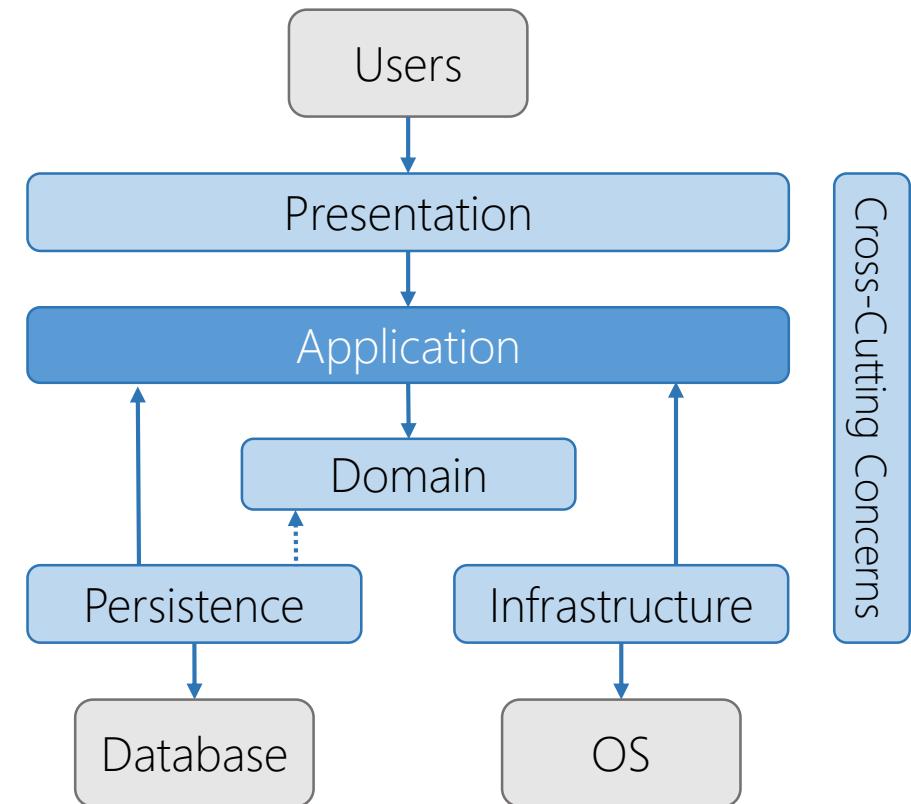
Summary

Focus on the inhabitants
Domain-centric Architecture



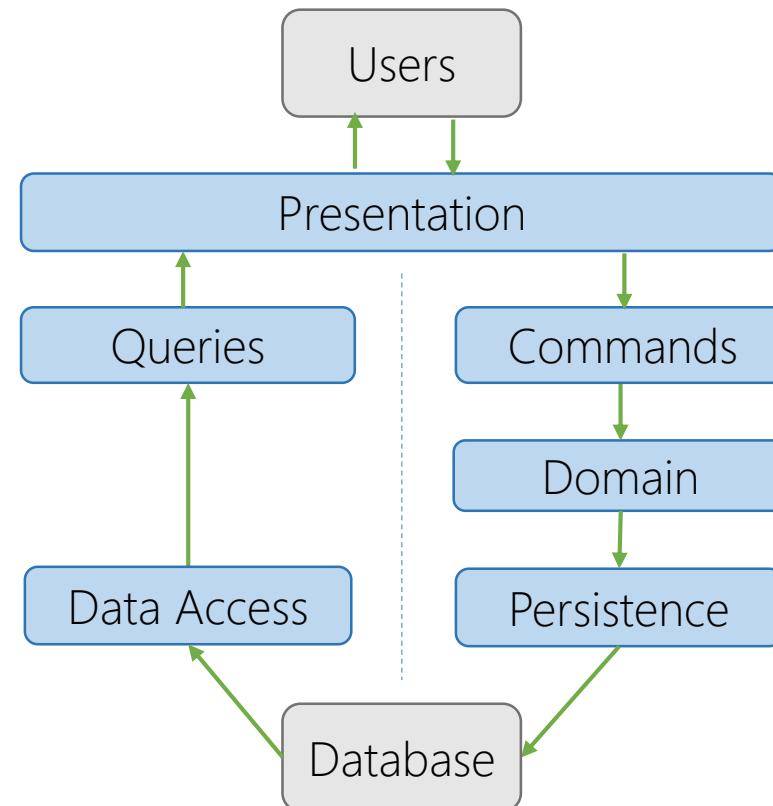
Summary

Focus on the inhabitants
Domain-centric Architecture
Application Layer



Summary

Focus on the inhabitants
Domain-centric Architecture
Application Layer
Commands and Queries



Summary

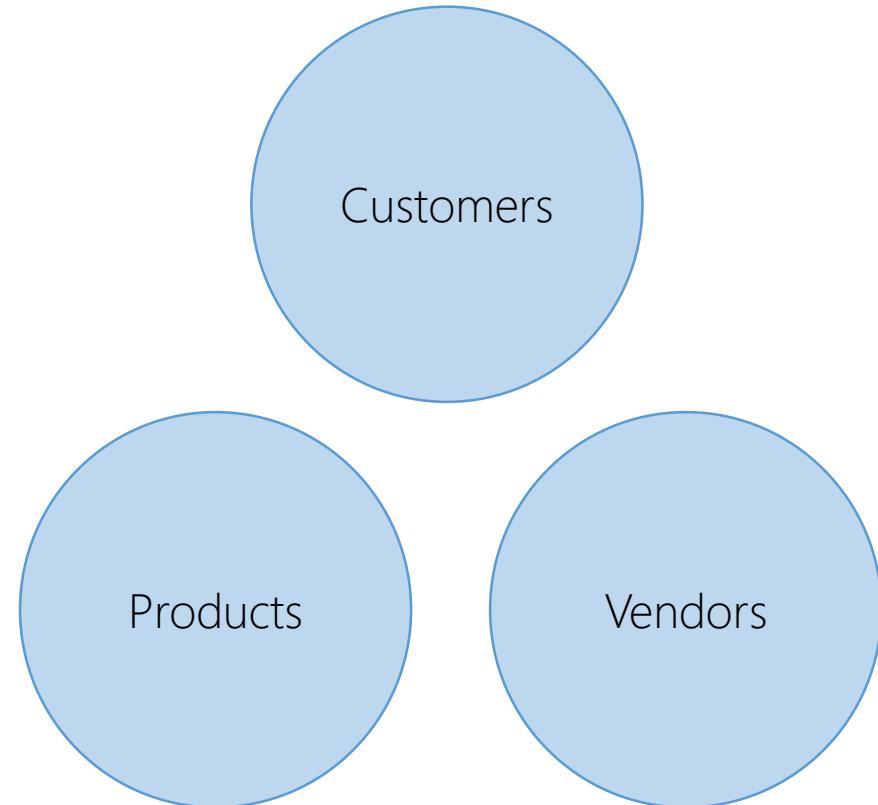
Focus on the inhabitants

Domain-centric Architecture

Application Layer

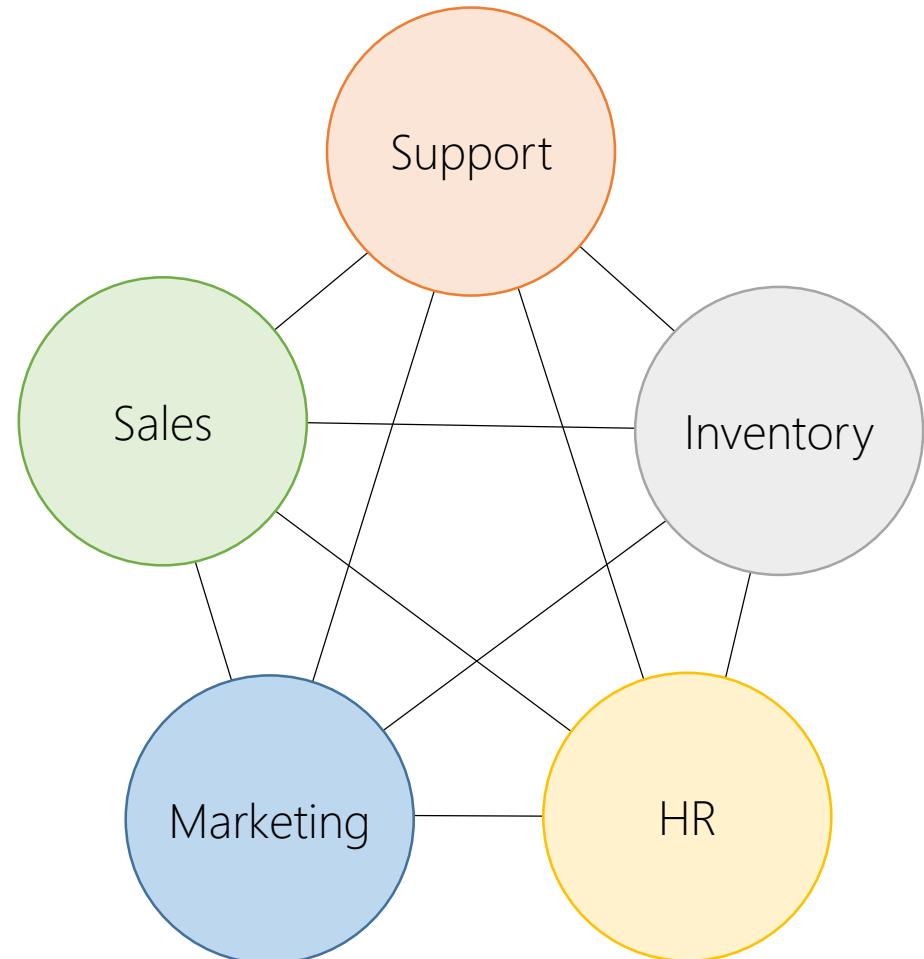
Commands and Queries

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](https://twitter.com/matthewrenze)

Email: info@matthewrenze.com

Website: www.matthewrenze.com



Thank You! :)