Reactive Programming with Commands, Actors and Events

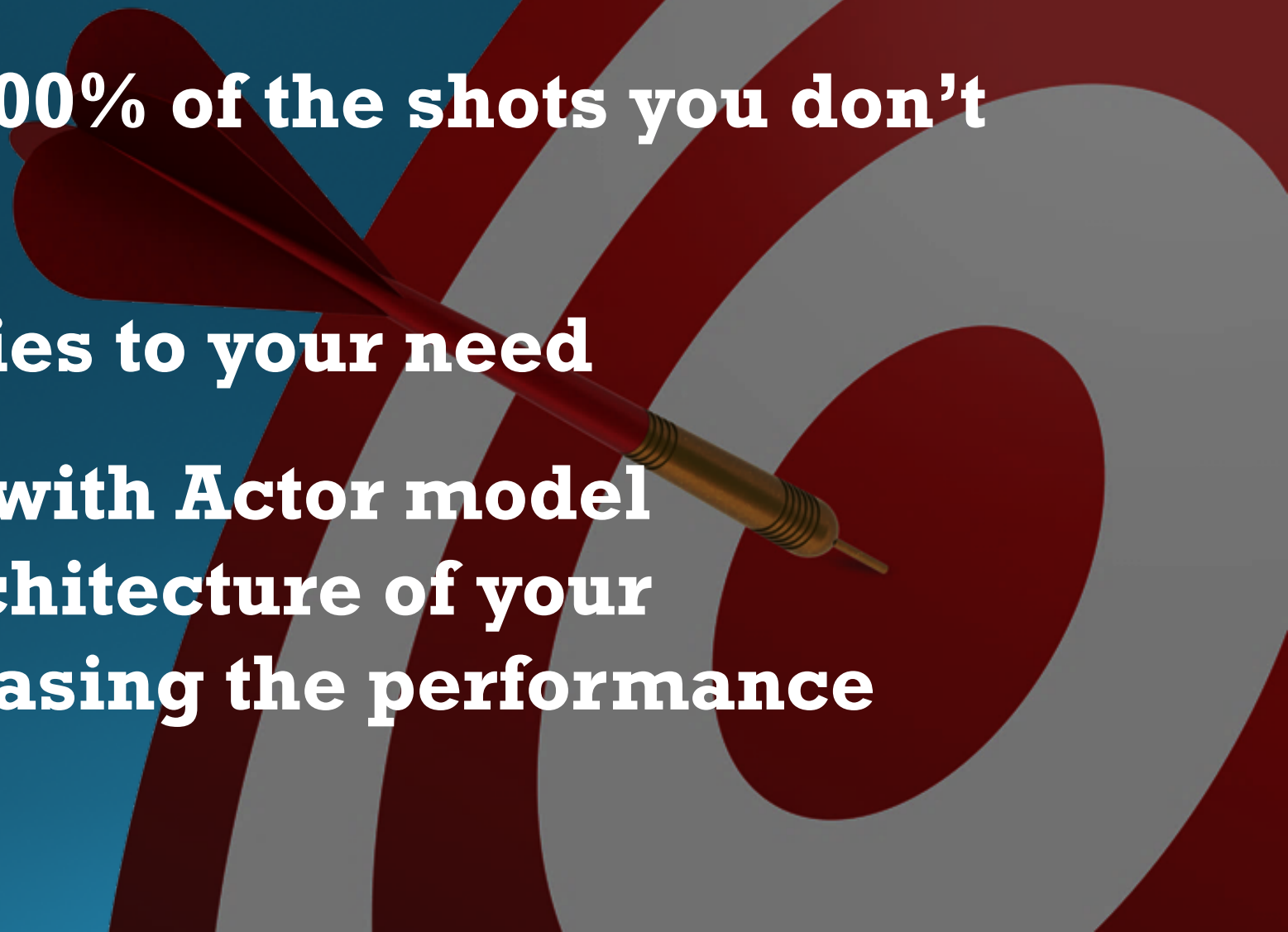
"We cannot solve our problems with the same thinking we used when we created them"

- Albert Einstein

Agenda

- The rise of programming models and big data
- CQRS and Event Sourcing
- Where and how functional programming fits in the big picture
- Apply CQRS to Reactive systems with Actor model for best of both
- Demo with code review

Objectives



- You'll will miss 100% of the shots you don't take
- Bend technonlogies to your need
- CQRS combined with Actor model simplifies the architecture of your application increasing the performance
- Go Reactive !

Introduction - Riccardo Terrell

- Originally from Italy, currently - Living/working in Washington DC
- +/- 19 years in professional programming
  - C++/VB → Java → .Net C# → Scala → Haskell → C# & F# → ??
- Author of the book “Functional Concurrency in .NET” - Manning
- *Polyglot programmer - believes in the art of finding the right tool for the job*

@trikace *rickyterrell.com* *tericcardo@gmail.com*

Microsoft® Most Valuable Professional

What tool should we use ?

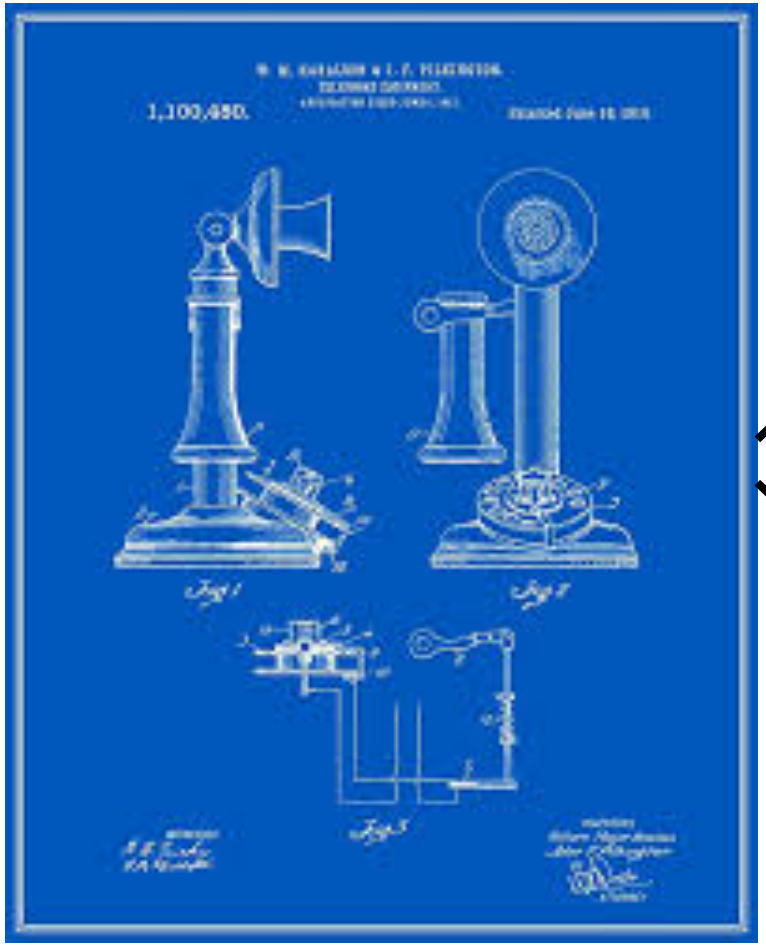

4
1
5
6
7
8
9
0

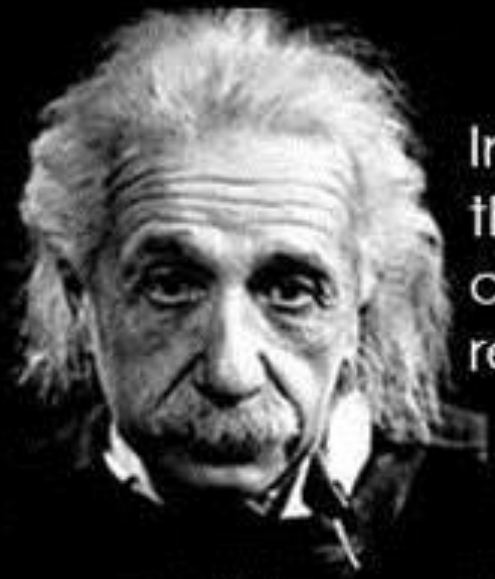

9:41 AM
100%
Messages
Calendar
Photos
Camera
Weather
Clock
Maps
Videos
Wallet
Notes
Reminders
Stocks
iTunes Store
App Store
iBooks
News
Health
Settings
Phone
Safari
Mail
Music
4
5
6

Requirements

- Scalable Application
- Responsive
- Modular/Distributed
- Loosely coupled
- Fault tolerant
- Buzz Buzz…

Re-using the same blueprint has minimal impact to the result

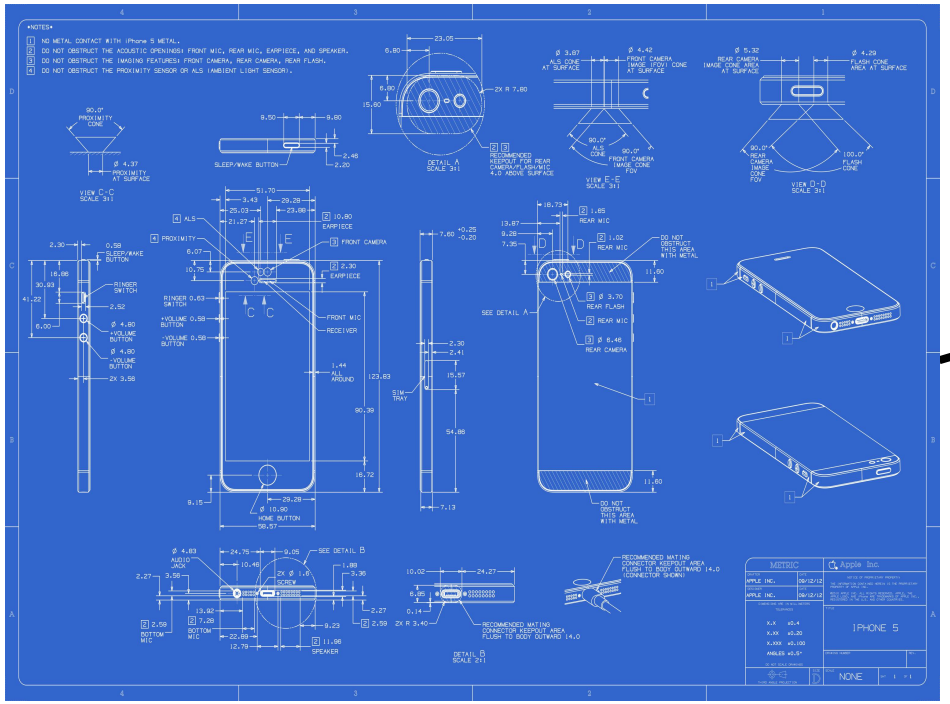




Insanity is doing the same thing over and over again and expecting different results.

- Albert Einstein

Changing the blueprint to obtain different & better results



?

What about the the tooling

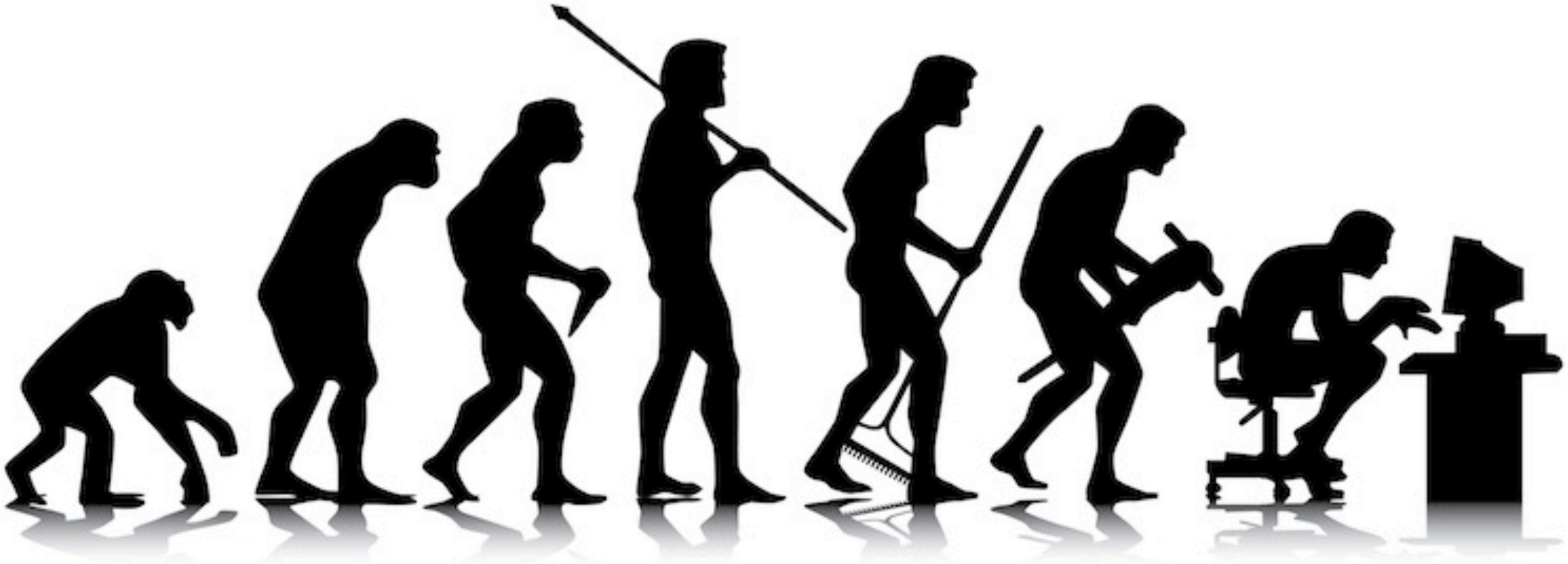
the tools are an implementation details


RETROSPECTIVE

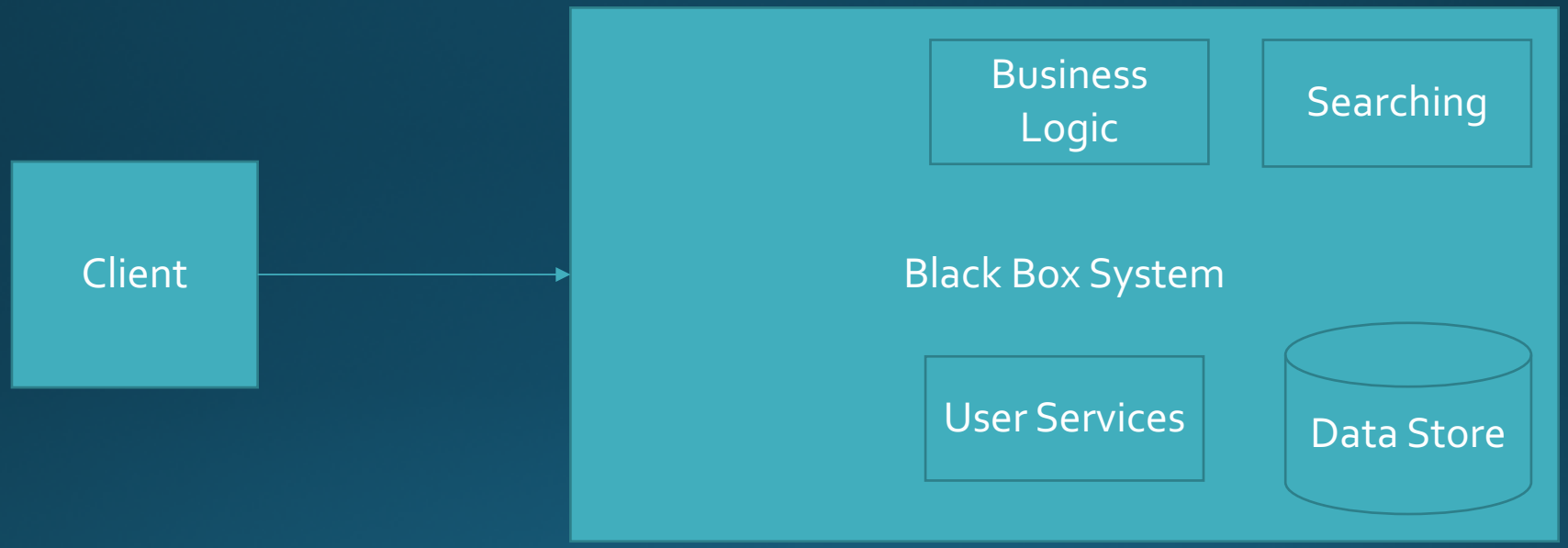

Looking out for different solutions might appear strange…


We like to stay in our comfortable zone…
changes are uncomfortable
we like what we know…

Industry sets the direction and the pace of evolution

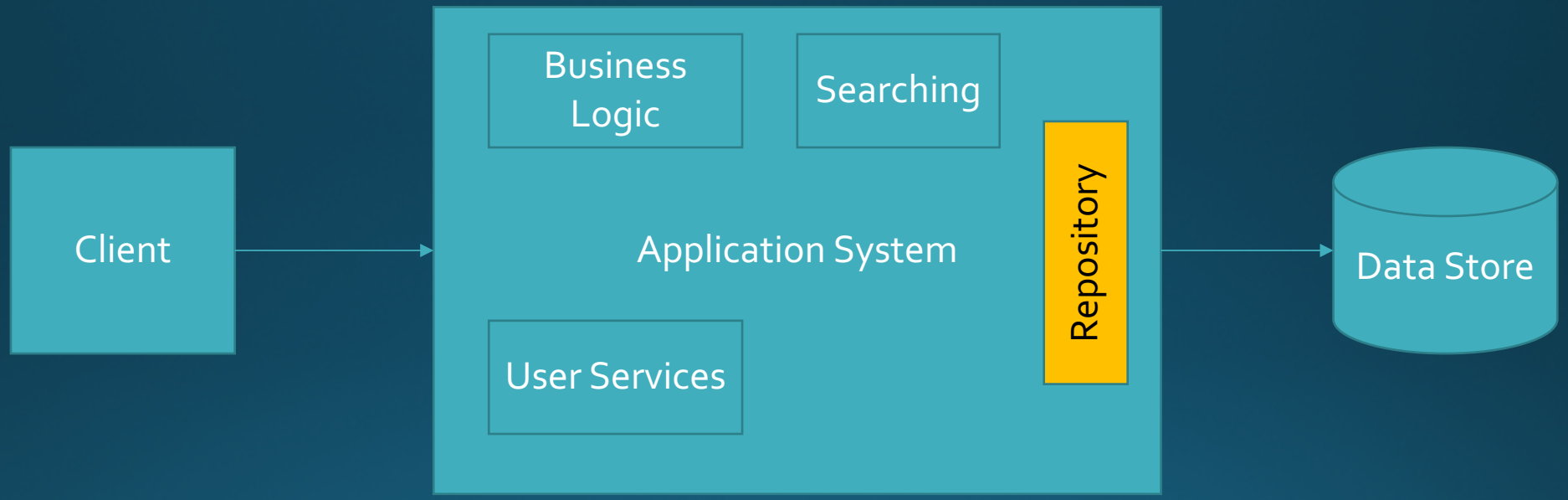


2 tier architecture

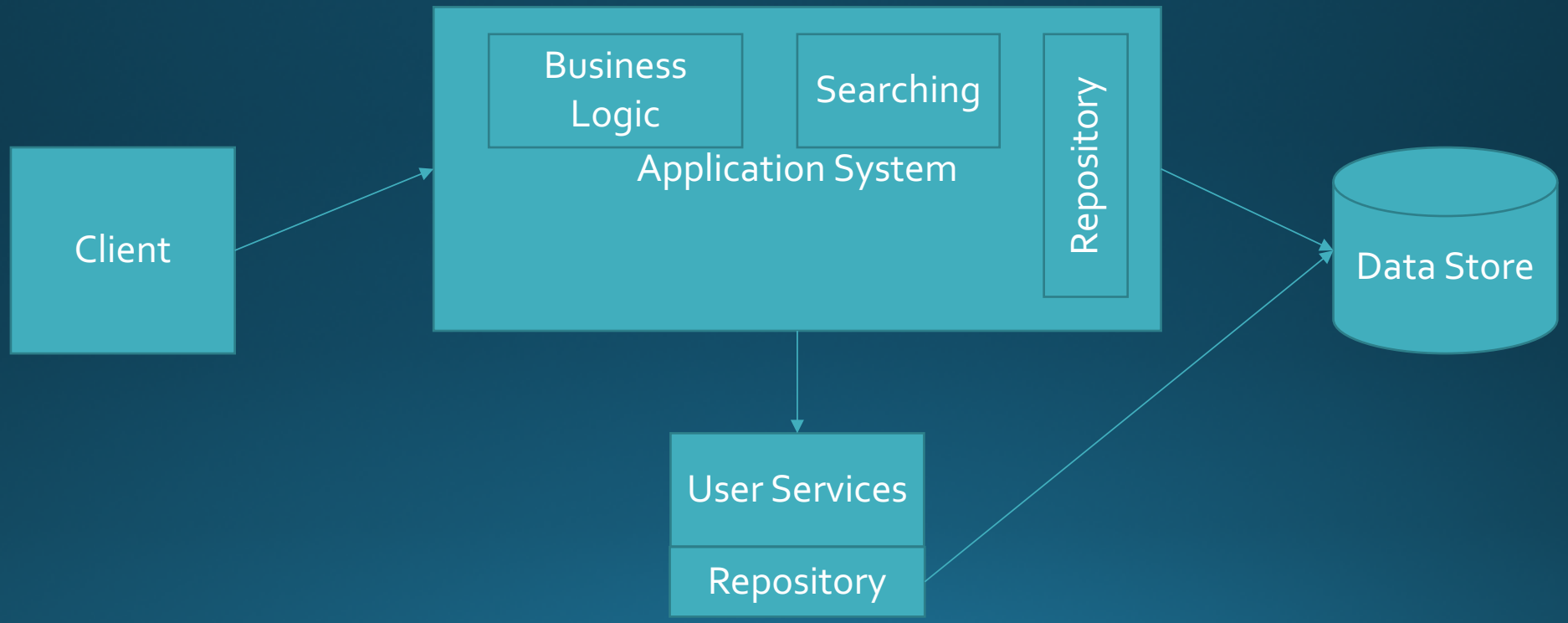


3 tier architecture

Clipboard Inheritance or Copy and Paste Development

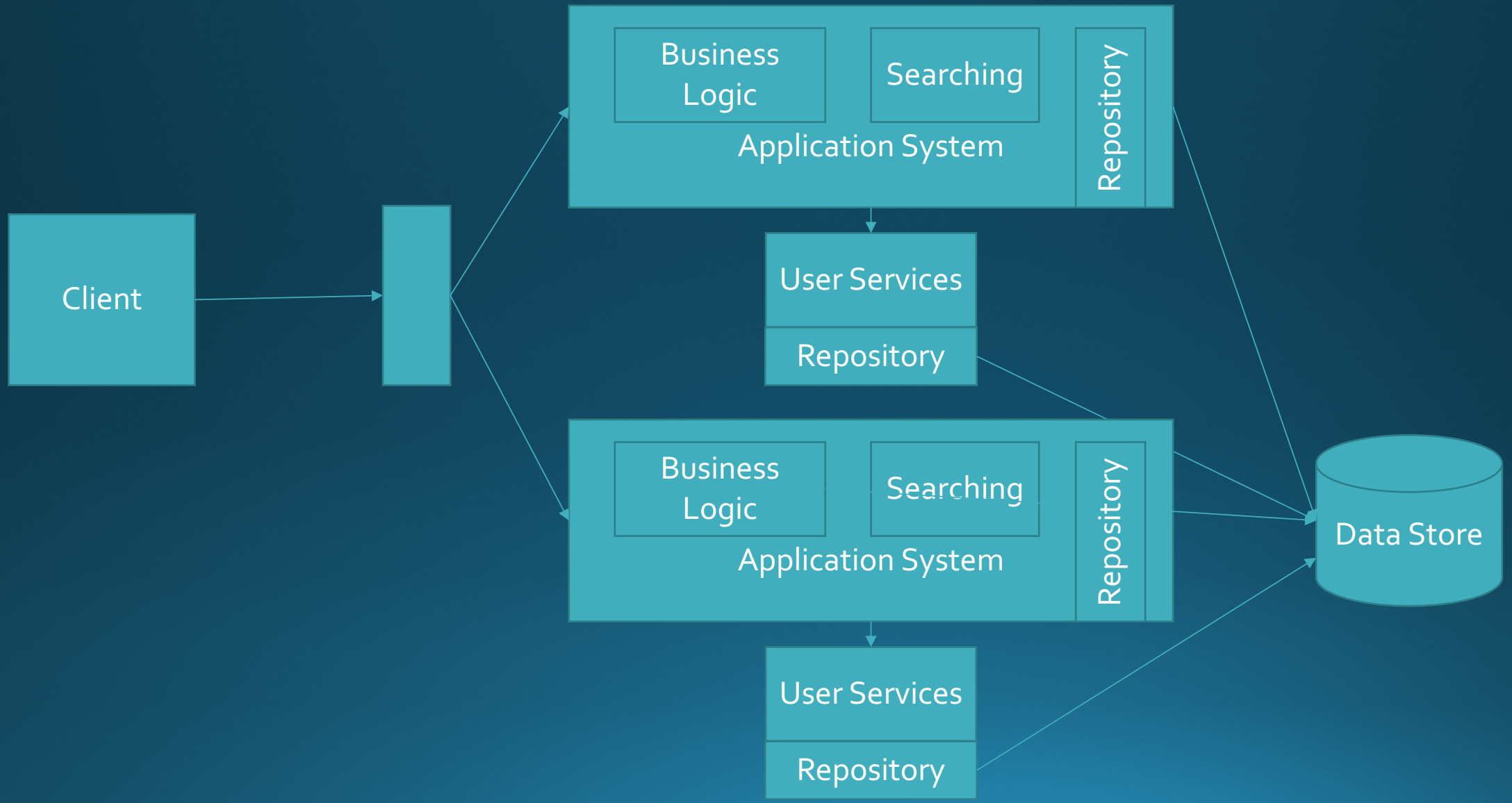


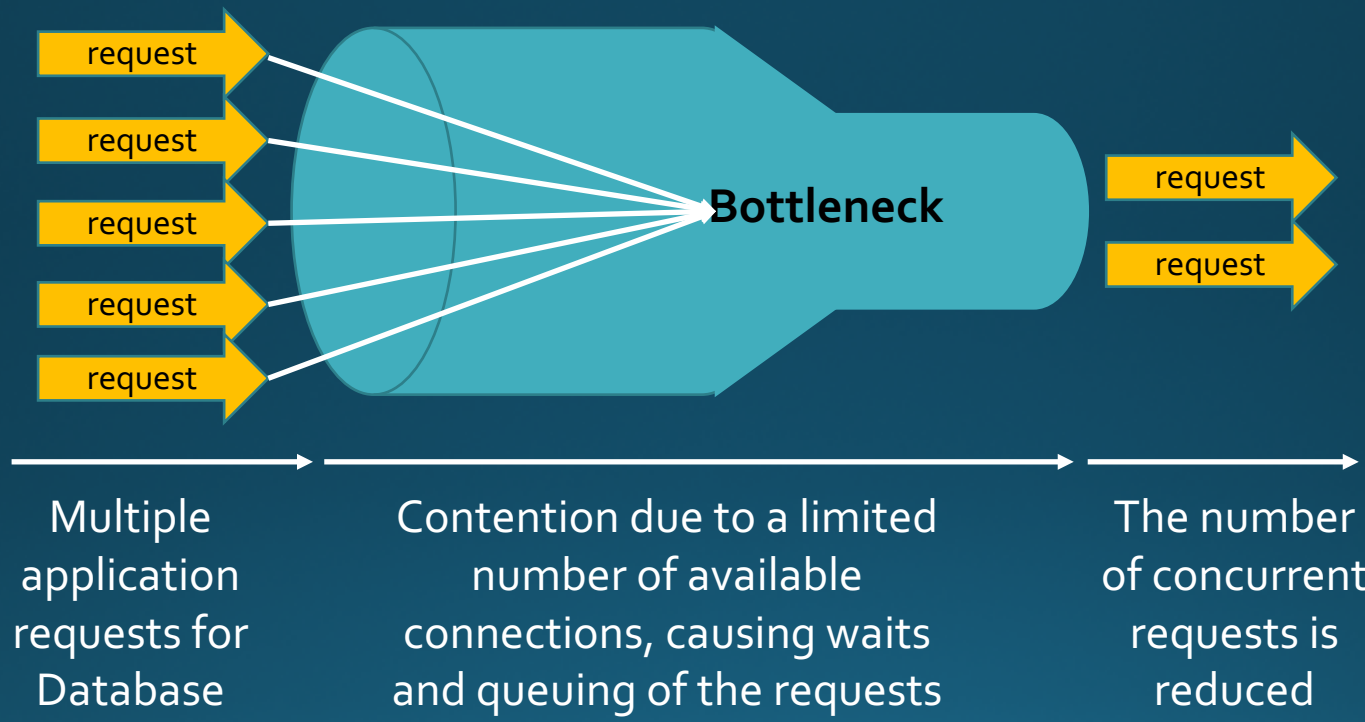
Enterprise with service layer



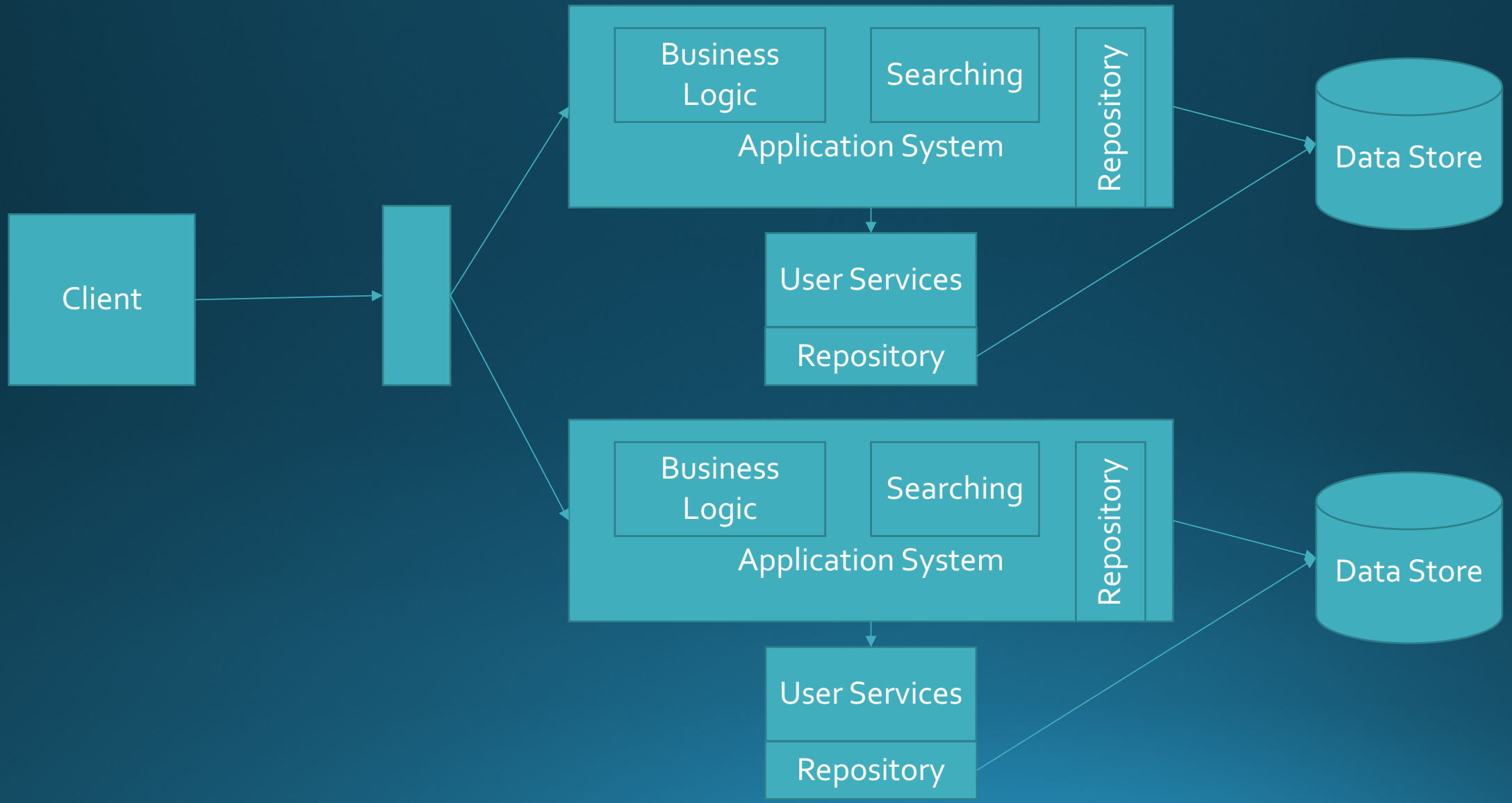
Distributed System

(sort of)

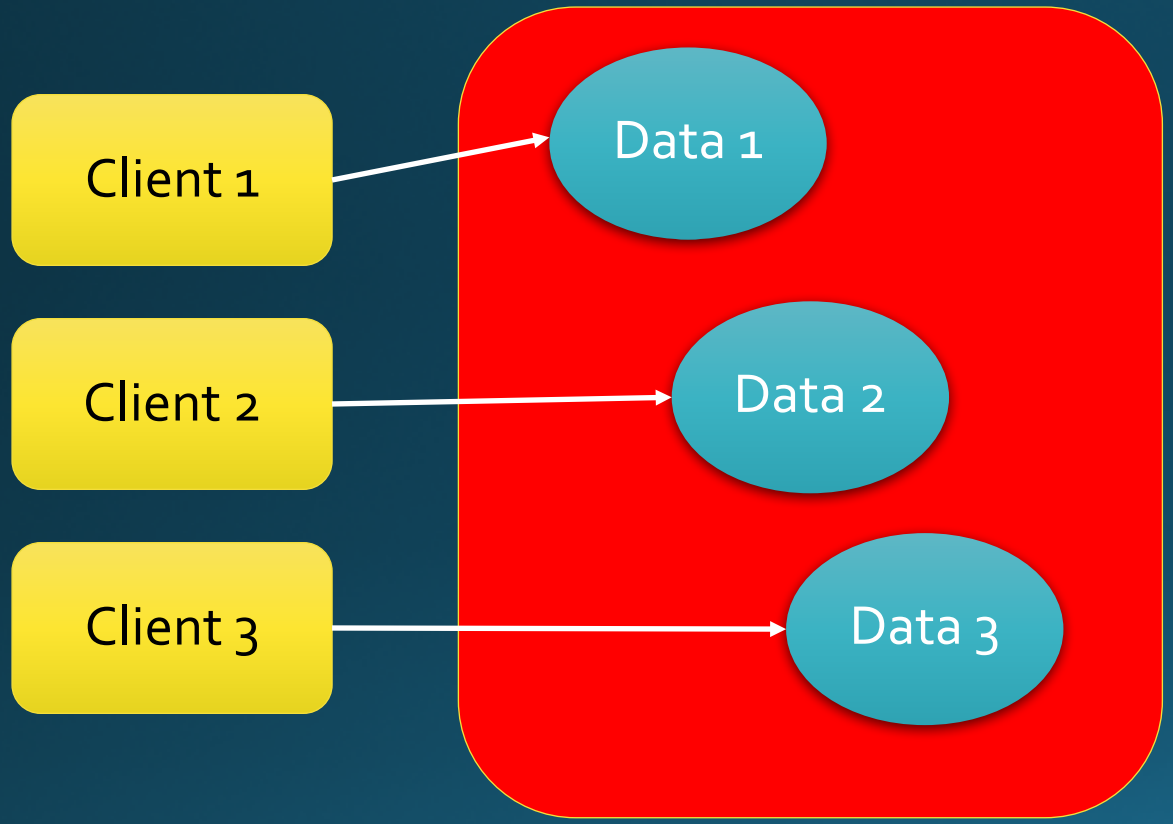



request
request
request
request
request
Bottleneck
request
request
Multiple application requests for Database
Contention due to a limited number of available connections, causing waits and queuing of the requests
The number of concurrent requests is reduced

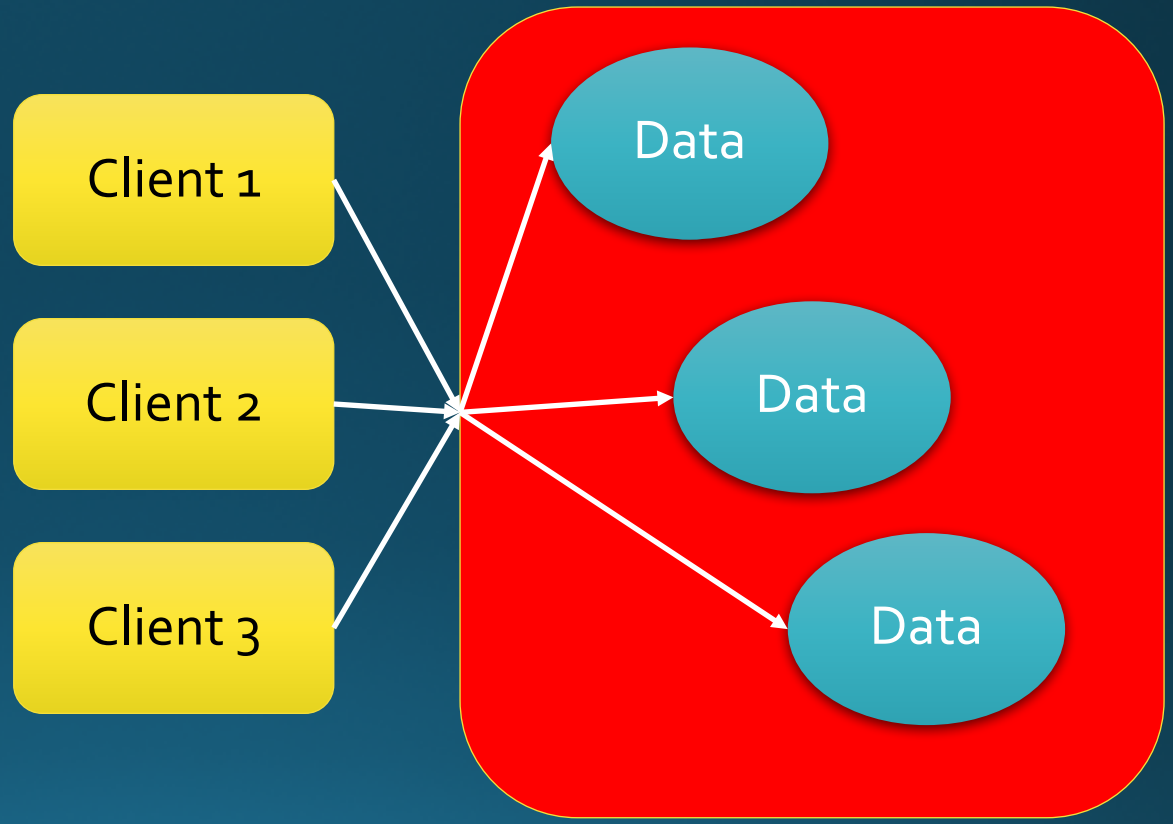
Distributed System with Database replicas



Stateful Server

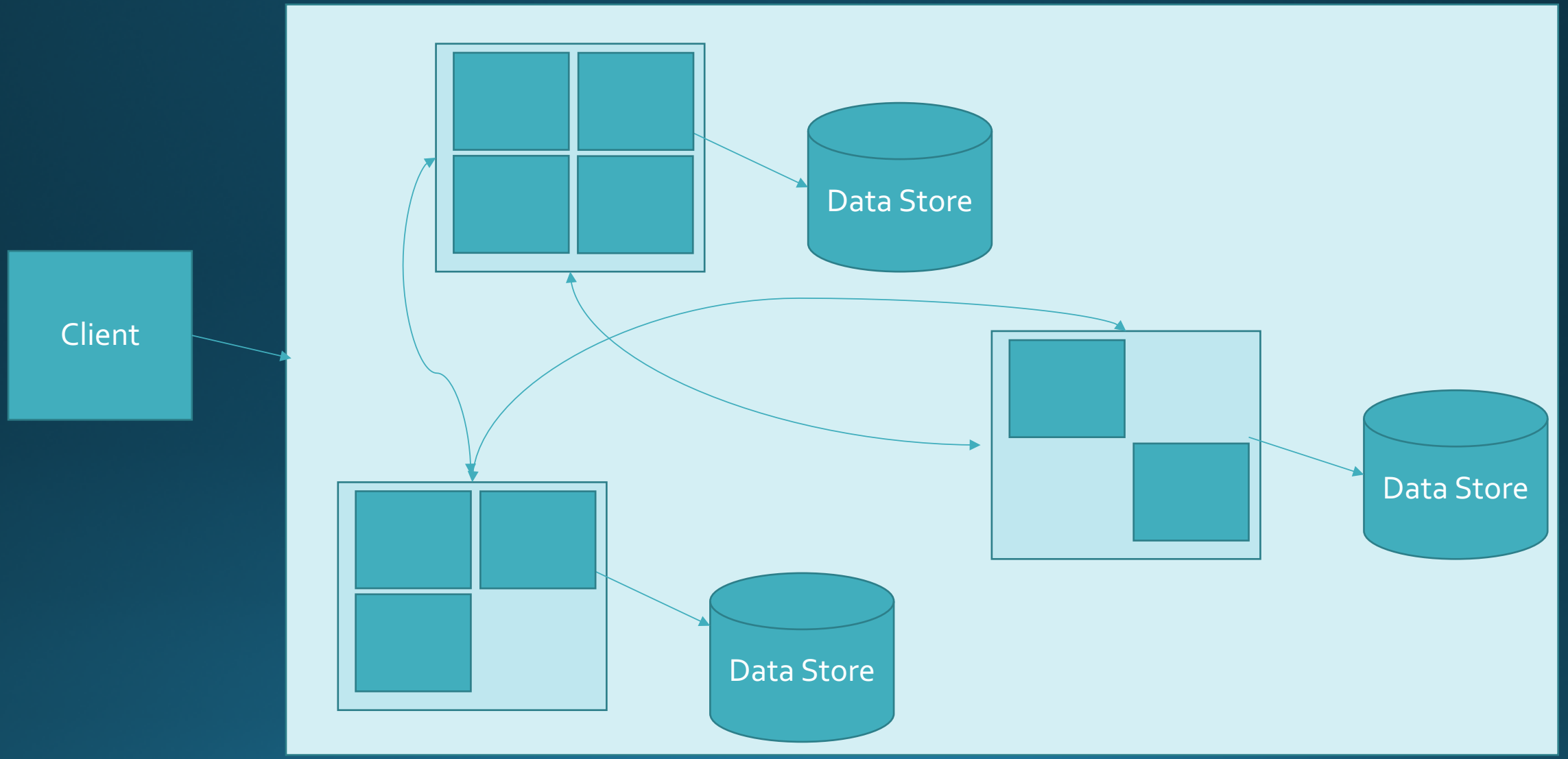


Stateless Server




Flash forward

Microservices



Why the evolution?

Because of the rise of the data age

- 1997-1999: During the late nineties internet had 280 million users
- 2000-2005: The Internet had 1 billion users
- Twitter wasn't alive yet (2006)
- Netflix had yet to introduce video streaming (2007)
- 2016 there are approximately 3,250,000,000 (over 3 billion) Internet users
  - Facebook has approximately 1.5 billion users
  - Twitter has approximately 300 million users

Why the evolution?

Because of the rise of the data age

- 1997-1999: During the late nineties internet had 280 million users
- 2000-2005: The Internet had 1 billion users
- Twitter wasn't alive yet (2006)
- Netflix had yet to introduce video streaming (2007)
- 2016 there are approximately 3,250,000,000 (over 3 billion) Internet users
  - Facebook has approximately 1.5 billion users
  - Twitter has approximately 300 million users

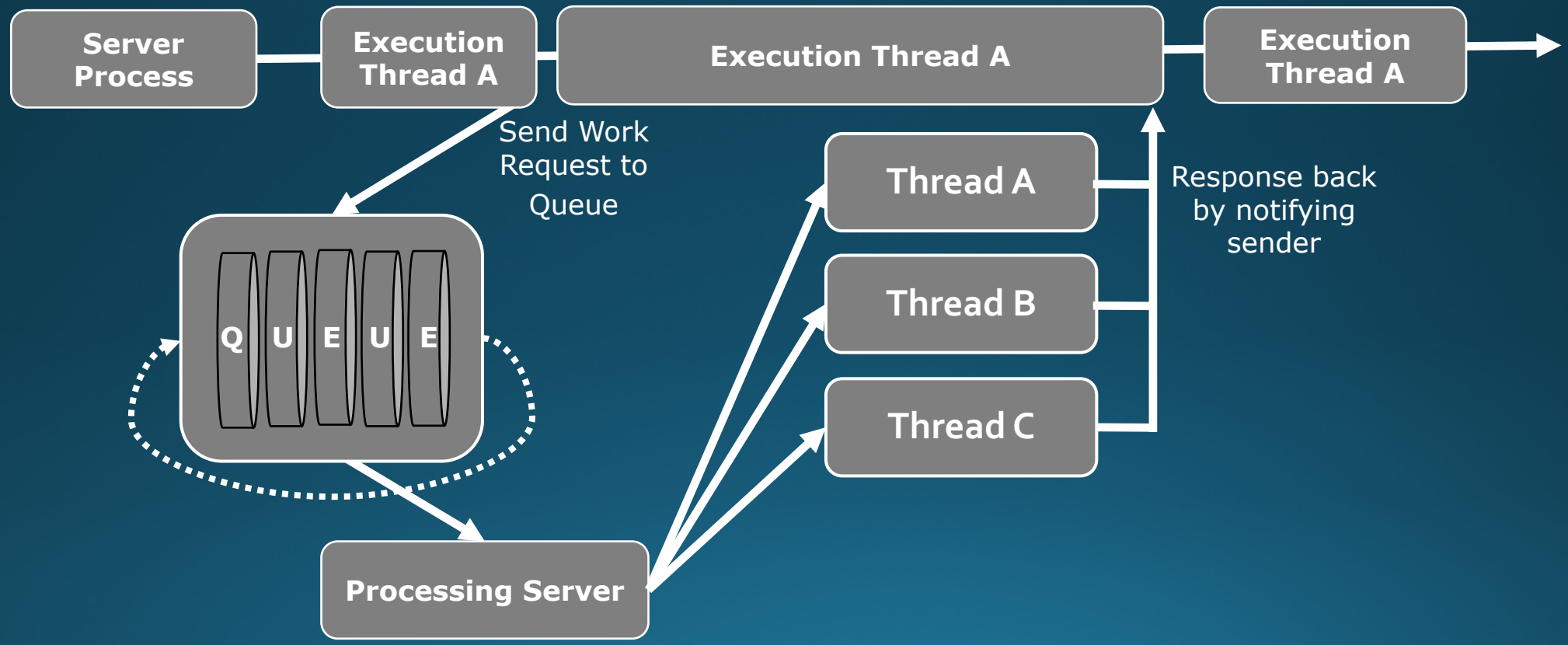

Scalability
is the ability to cope and perform under an increasing workload

ACD is the secret sauce…
for a successful performant application

- **Asynchronicity :** an operation that completes later in the future
- **Caching** : avoids any work that is not necessary to repeat
- **Distribution** : partitions the requests across multiple systems to scale out processing

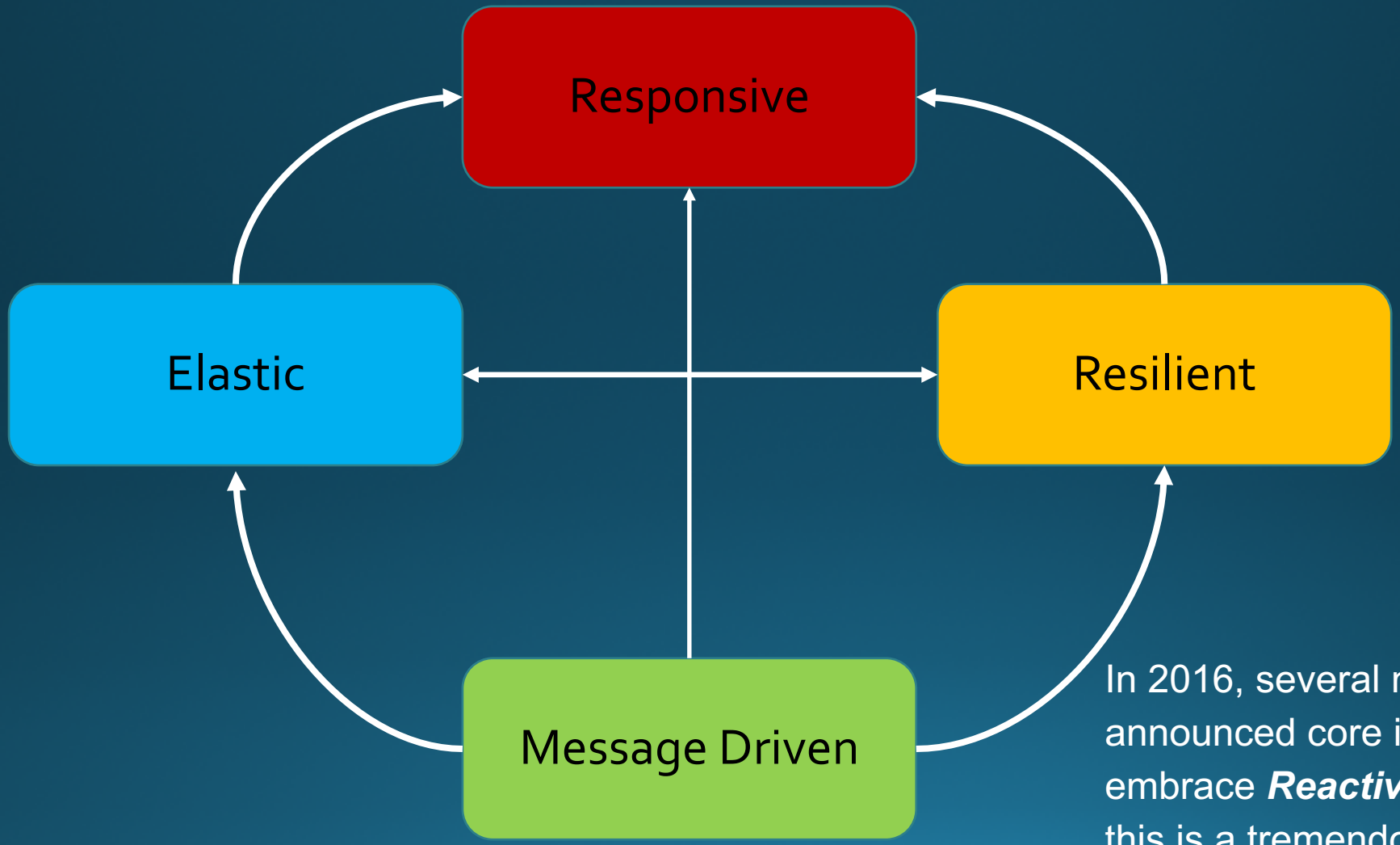
Scaling is the ability to cope and perform under an increasing workload.

A different asynchronous pattern
Queuing work for later execution




Let’s go back

Reactive Manifesto



In 2016, several major vendors have announced core initiatives to embrace ***Reactive Programming*** this is a tremendous validation of the problems faced by companies today.

http://www.reactivemanifesto.org

Foundation of CQRS

is a pattern that encourages separate domain model interfaces for dealing with commands and queries

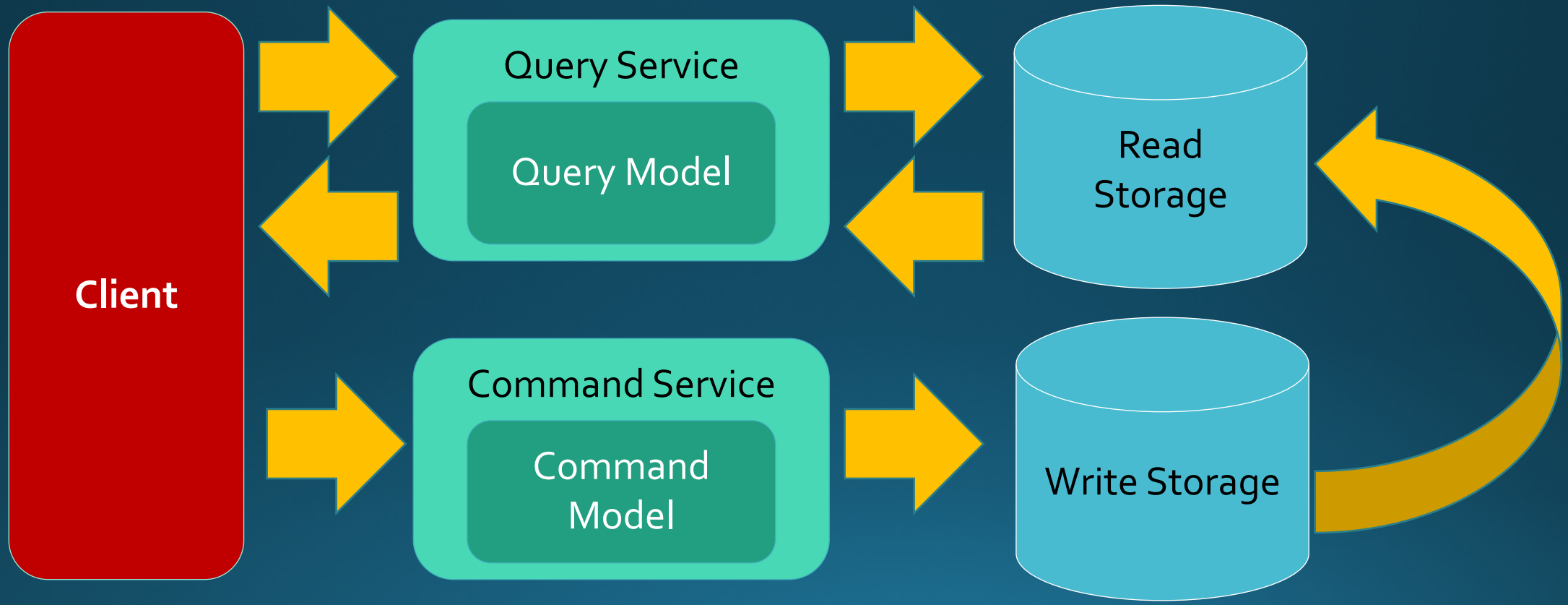
- Command
- Query
- Responsibility
- Segregation

What is CQRS?

*"CQRS is simply the creation of **two objects** where there was previously only one. The separation occurs based upon whether the methods are a command or a query (the same definition that is used by Meyer in Command and Query Separation: a command is any method **(object)** that mutates state and a query is any method **(object)** that returns a value)"*

- Greg Young

CQRS pattern



Command

- Commands are directives to perform some action to the domain (Behaviors)
- Commands can be rejected by the domain (validation)
- Processing a Command will result in o:n Events raised
- Commands are imperative and are components to describe the domain

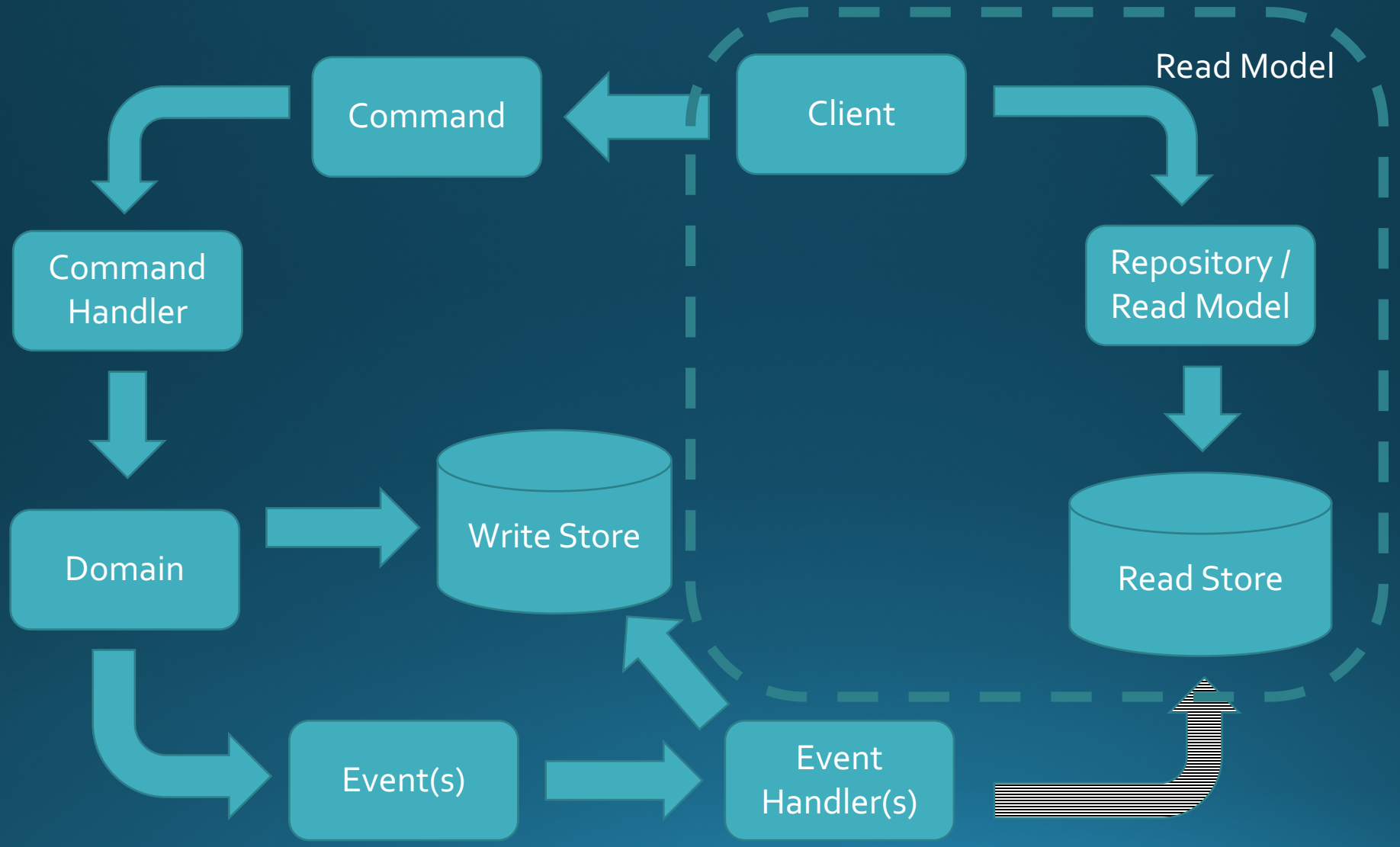
```
public sealed class SubmitOrder : ICommand
{
    public readonly int Quantity;
    public readonly string Ticker;
    public readonly decimal Price;

    public SubmitOrder(string ticker, int quantity, decimal price)
    {
        Ticker = ticker;
        Quantity = quantity;
        Price = price;
    }
}
```

Event

- Says that something happened
- Events are **notification** that report on something that has **already** happened
- Events cannot be **rejected**
- Events are in past-tense and are components of the Ubiquitous Language to describe the domain (OrderSubmited Event)

```
public sealed class OrderSubmited : IEvent
{
    public readonly Guid OrderID;

    public Ordersubmited(Guid orderId)
    {
        OrderId = orderId;
    }
}
```

CQRS pattern visualized



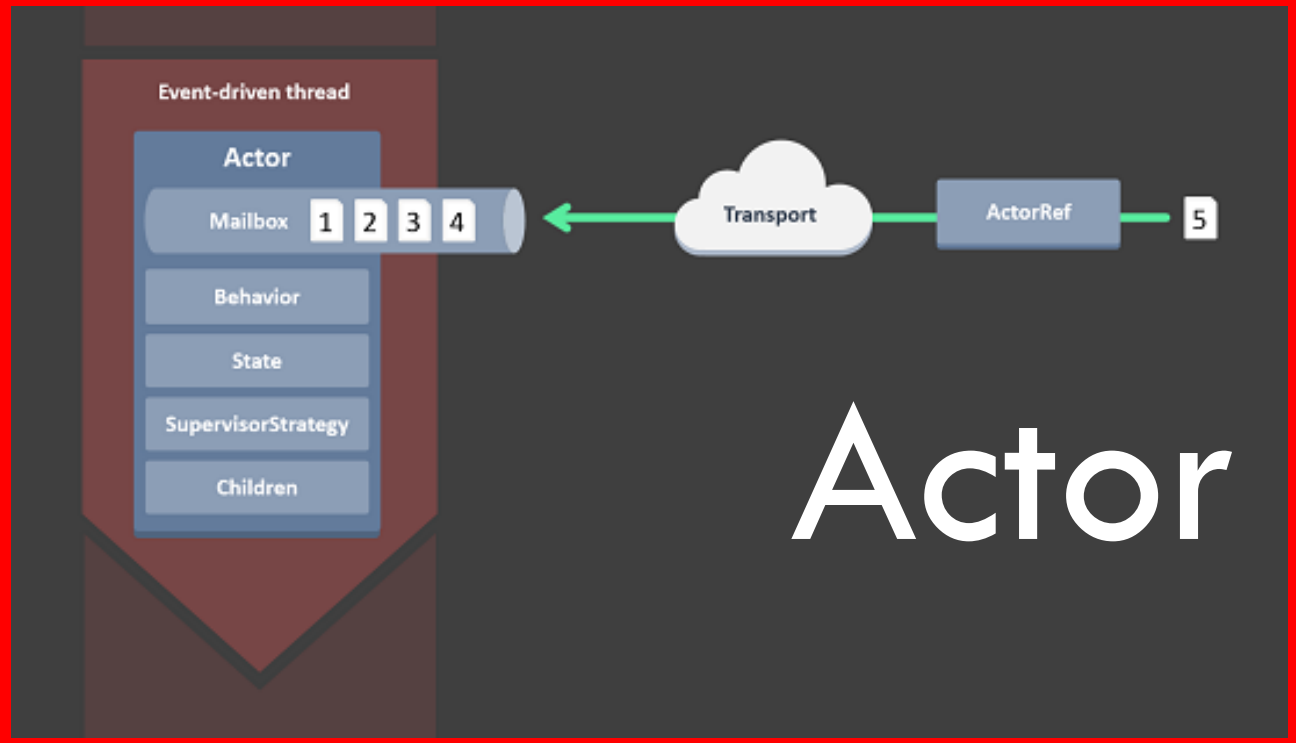
Eventual Consistency

By applying CQRS, the concepts of Writes and Reads have been **separated**. If we keep the paths segregated, how do we keep them **consistent**?

- Business **determines** how long between sync
- Pushed **asynchronously** from the write side
- Read side has listeners
- Queue can be used
- Use the event store as your queue

- “Almost” infinite scalability
- Clear understanding of the code
- Separation of concerns
- Message Based
- Segregation of Responsibilities
- Smaller, simpler classes
- Decouples handlers
- Increased isolation of areas
- Benefit optimize for read
- Increase in performance

- “Almost” infinite scalability
- Clear understanding of the code
- Separation of concerns
- Message Based
- Segregation of Responsibilities
- Smaller, simpler classes
- Decouples handlers
- Increased isolation of areas
- Benefit optimize for read
- Increase in performance


Event-driven thread
Actor
Mailbox
1
2
3
4
Behavior
State
SupervisorStrategy
Children
Transport
ActorRef
5

Actor Model

Actor Model Message-Driven

What is an Actor?

- **Share Nothing**
- **Messages are passed by value**
- Light weight processes/threads communicating through messaging
- Communication only by messages
- Lightweight object
- Processing
- Storage – State
- Running on it’s own thread.
- Messages are buffered in a “mailbox”

Actor

What an Actor Offers

High throughput

Fault tolerance

No manual thread management

Scale Out & Up

High Availability

Asynchronous

Combining Actor Model & CQRS

- CQRS core lacks in resiliency (self-healing) … Actor helps
- Actor Model provides a message-based pattern
- Actor Model is driving to reactive-based development
- Message-based communication with concurrent processing
- Messages in an Actor System include both Commands and Events

CQRS combined with Actor...?

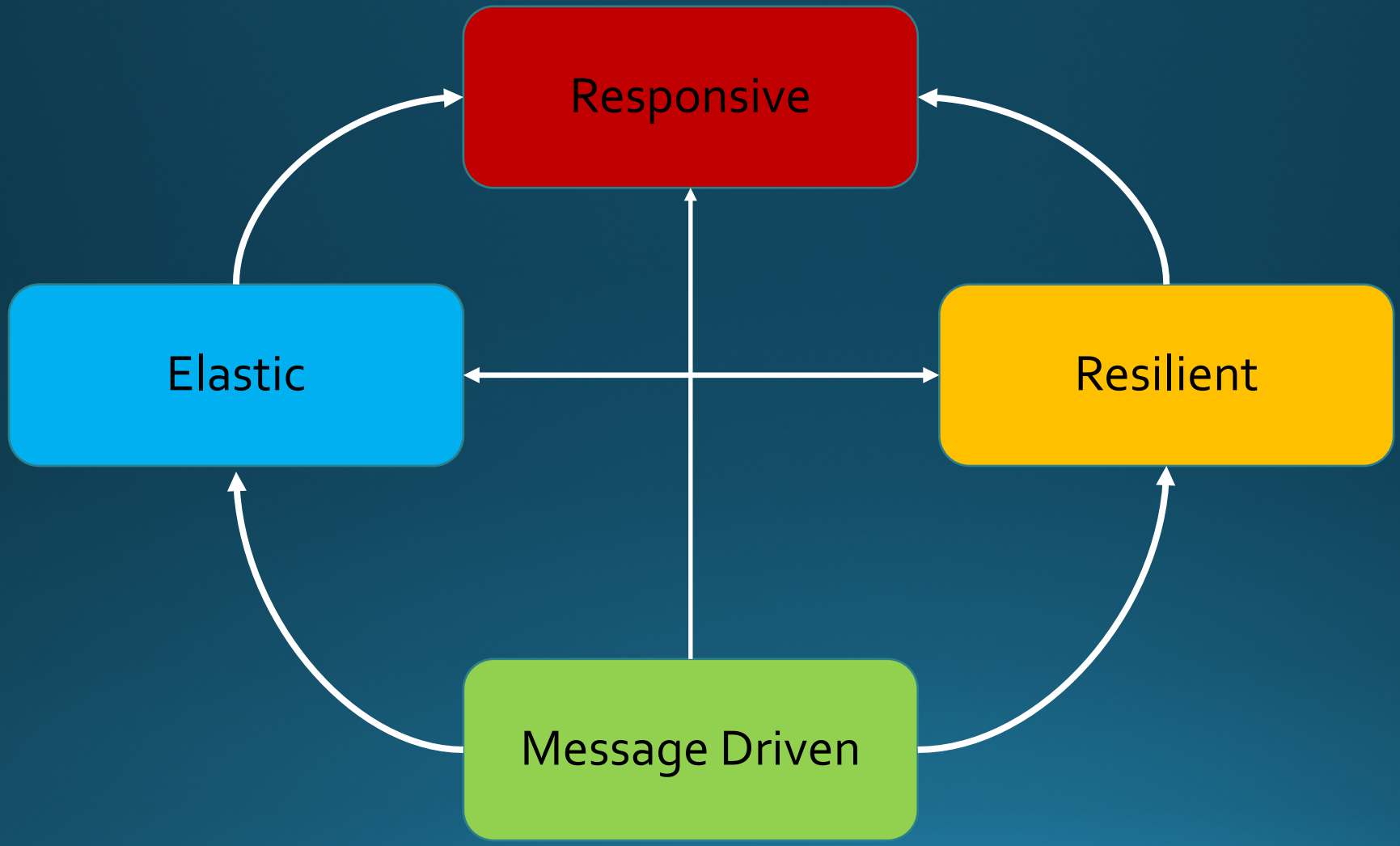
Responsive Read Model
Queries are (can be) against an optimized read model

Group Handlers into Clusterable Groups
Elastic

Message Driven
Command dispatcher

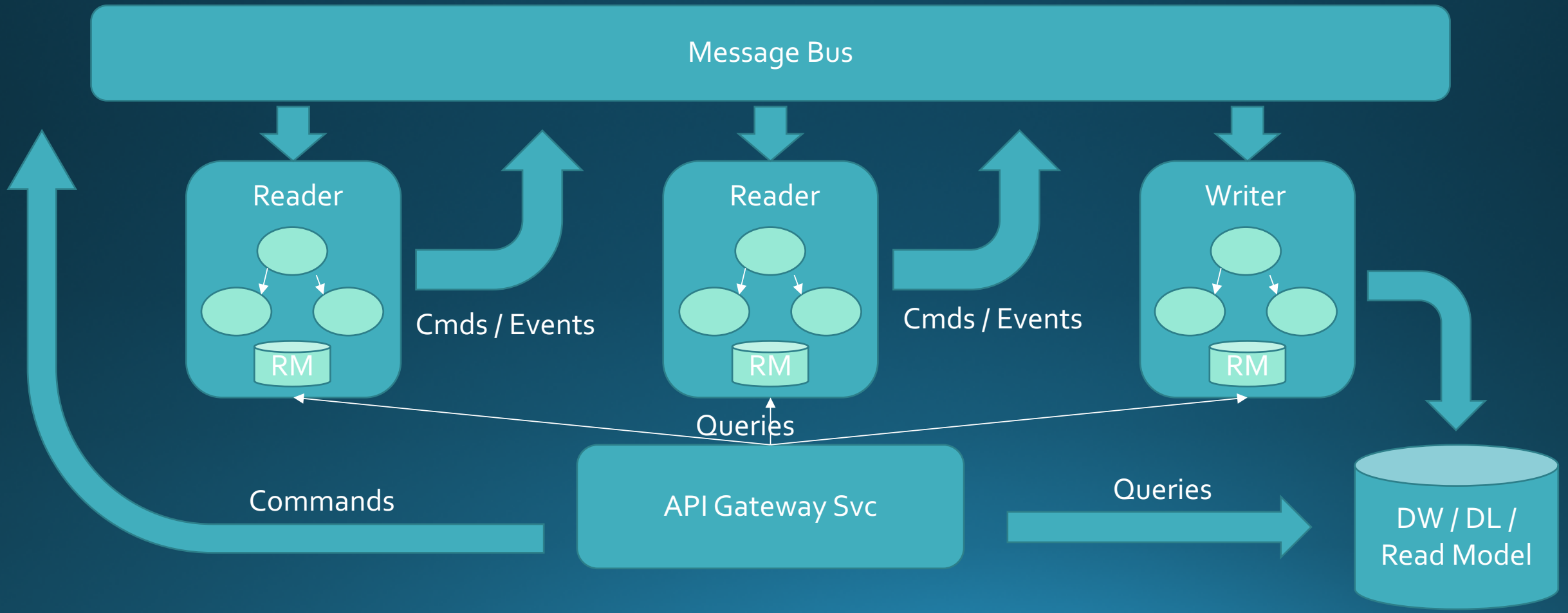
Async Friendly
Responsive

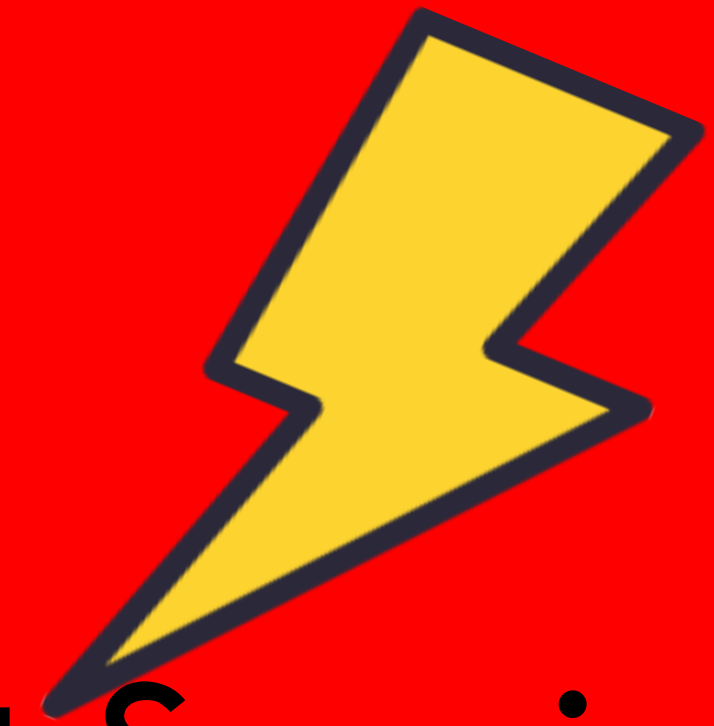
Reactive Manifesto



http://www.reactivemanifesto.org

Actor Model + CQRS





Event Sourcing

Event Sourcing

*"The **majority** of business applications today rely on storing **current state** in order to process transactions. As a result in order to track history or implement audit capabilities **additional** coding or frameworks are required."*

- Greg Young

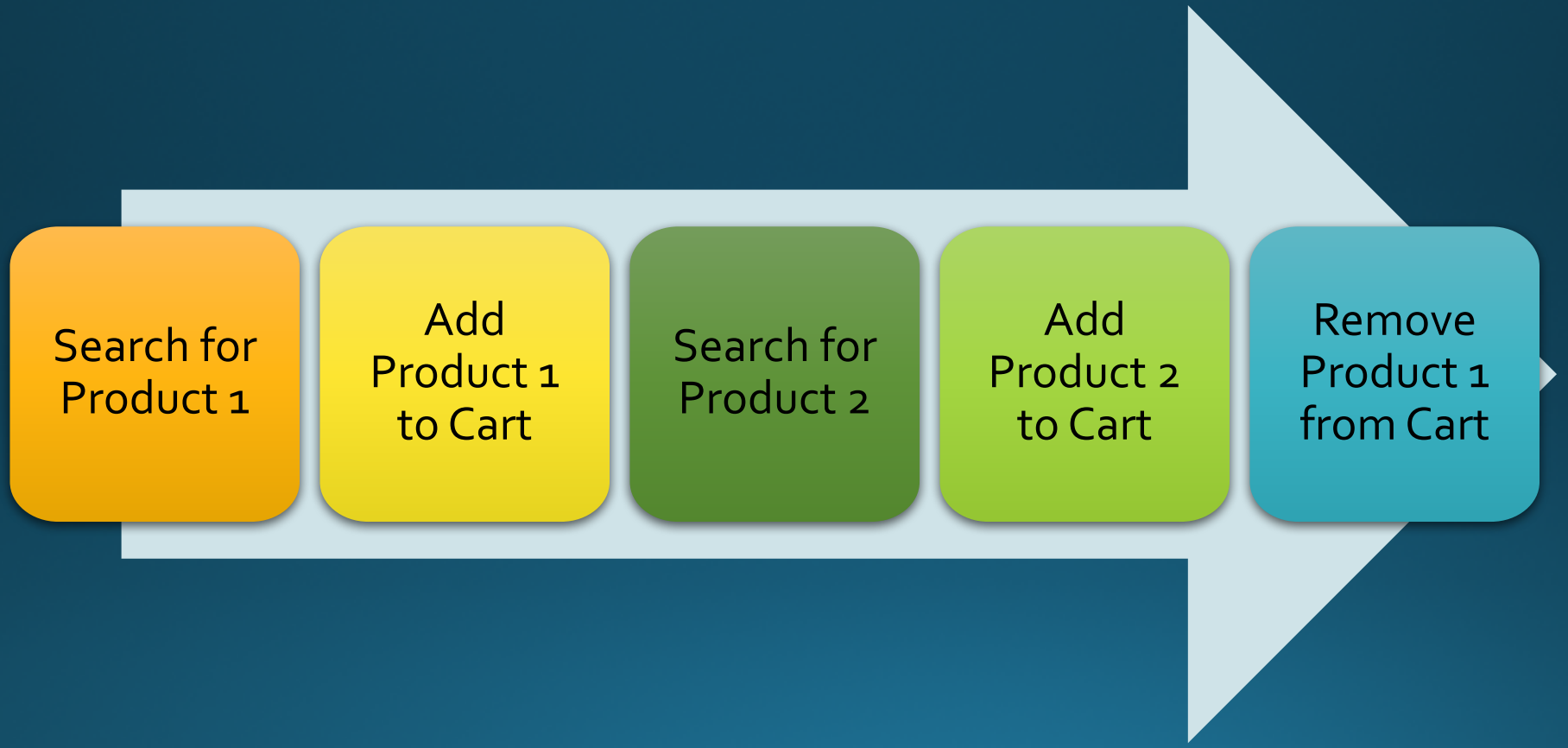
"Event sourcing provides a means by which we can capture the real intent of our users"

Append only events

CRUD Shopping Cart

1. Cart created
2. Item 1 @ $30 added
3. Item 2 @ $15 added
4. Item 3 @ $12 added
5. Item 4 @ $5 added
6. Shipping information added
7. Total @ $62 generated
8. Order 123 inserted

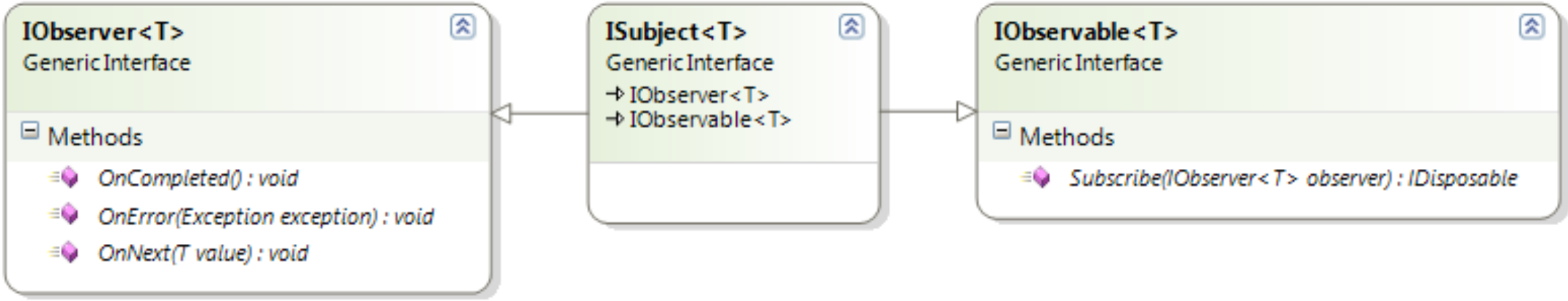
Event Sourcing



Reactive Extensions

What are Reactive Extensions

Rx is a library for composing asynchronous and event-based programs using observable sequences and LINQ-style query operators



IObserver & IObservable

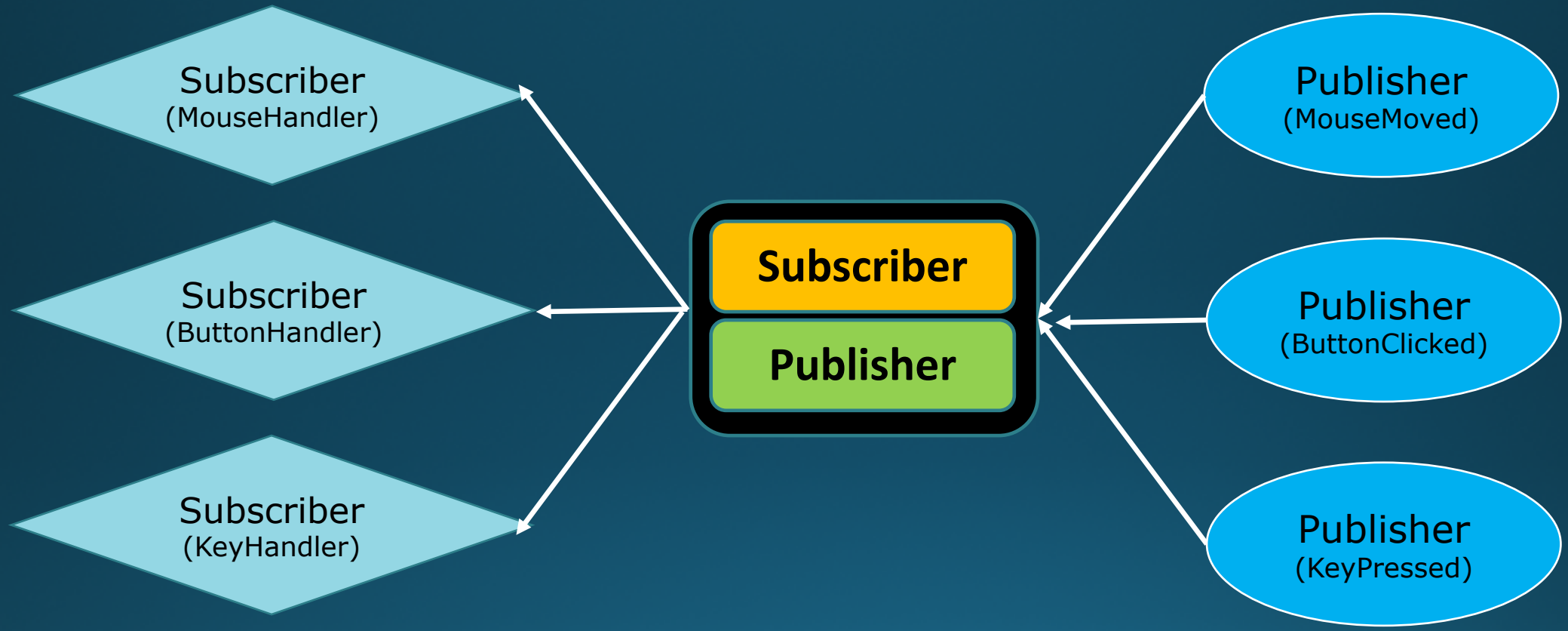
```
type IObserver<'a> = interface
    abstract OnCompleted : unit -> unit
    abstract OnError : exn -> unit
    abstract OnNext : 'a -> unit
end

type IObservable<'a> = interface
    abstract Subsrcribe : IObserver<'a> -> IDisposable
end
```

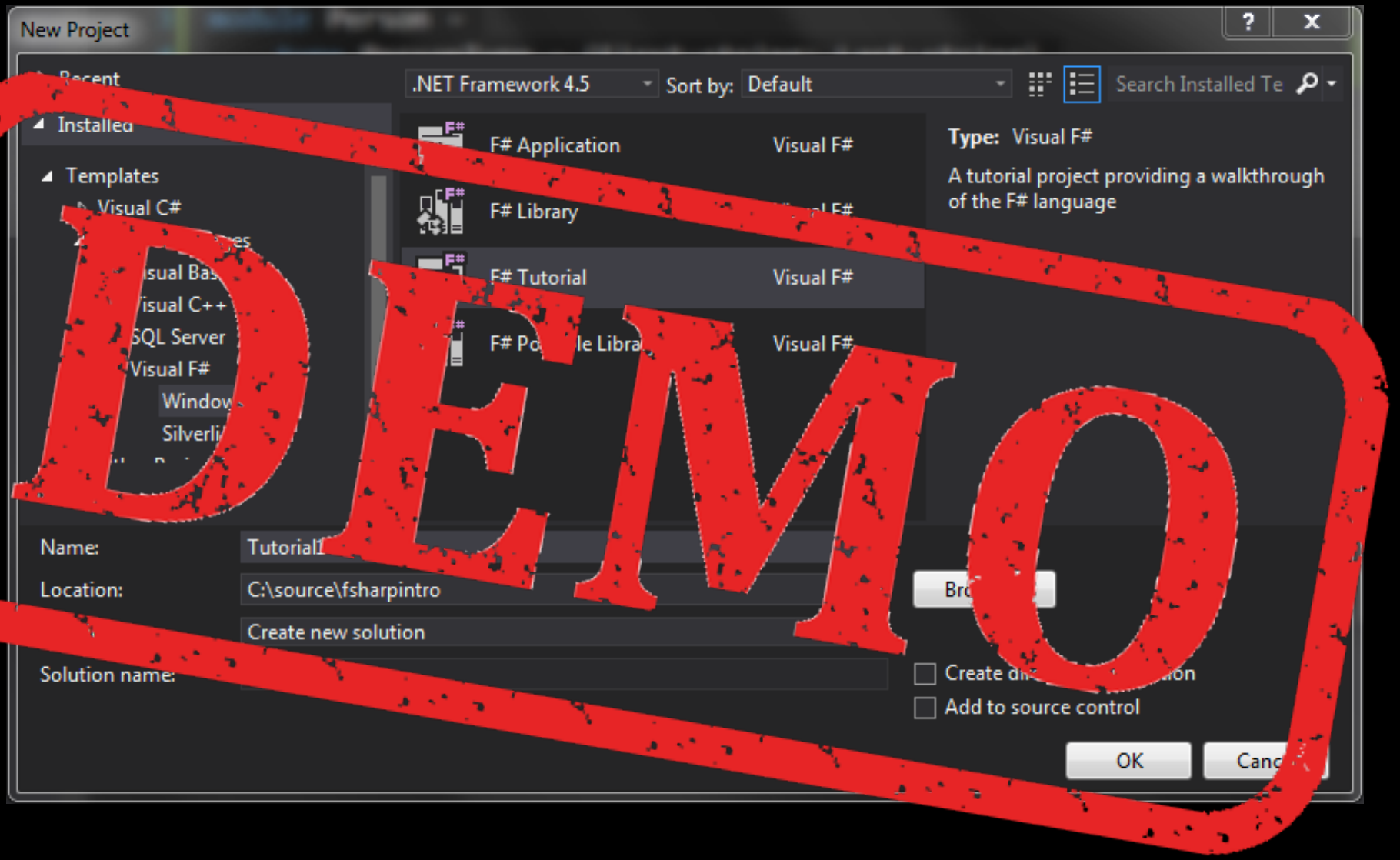
IObservable the dual of IEnumerable

```
type IObserver<'a> = interface
  abstract OnNext : 'a with set
  abstract OnCompleted : unit -> unit
  abstract OnError : Exception -> unit
end


type IObservable<'a> = interface
  abstract Subscribe : IObserver<'a>
                    -> IDisposable
end
```

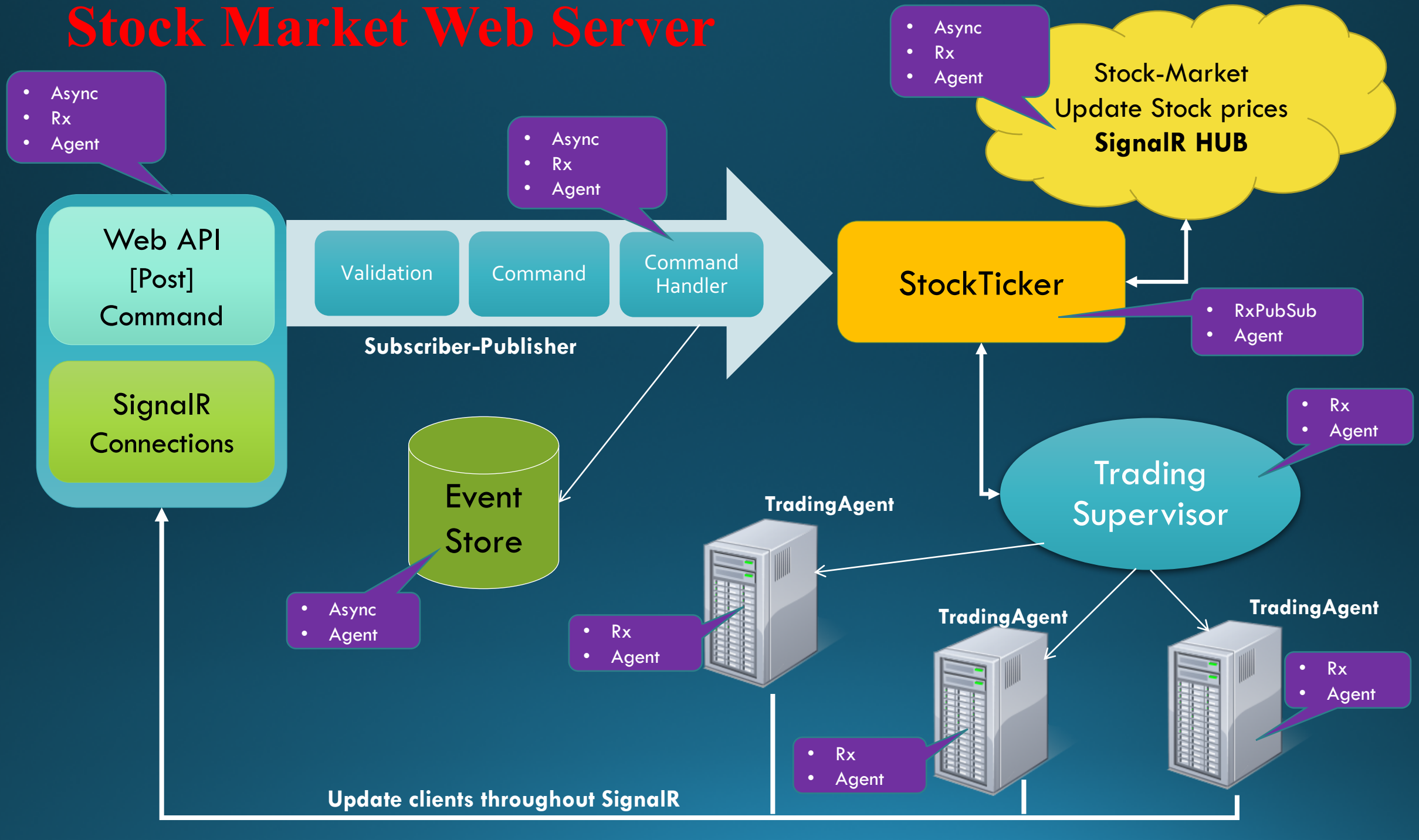
```
type IEnumerator<'a> = interface
   interface IDisposable
   interface IEnumerator

   abstract Current : 'a with get
   abstract MoveNext : unit -> bool
end

type IEnumerable<'a> = interface
   interface IEnumerable

   abstract GetEnumerator : IEnumerator<'a>
end
```

Reactive Extensions as a Bus

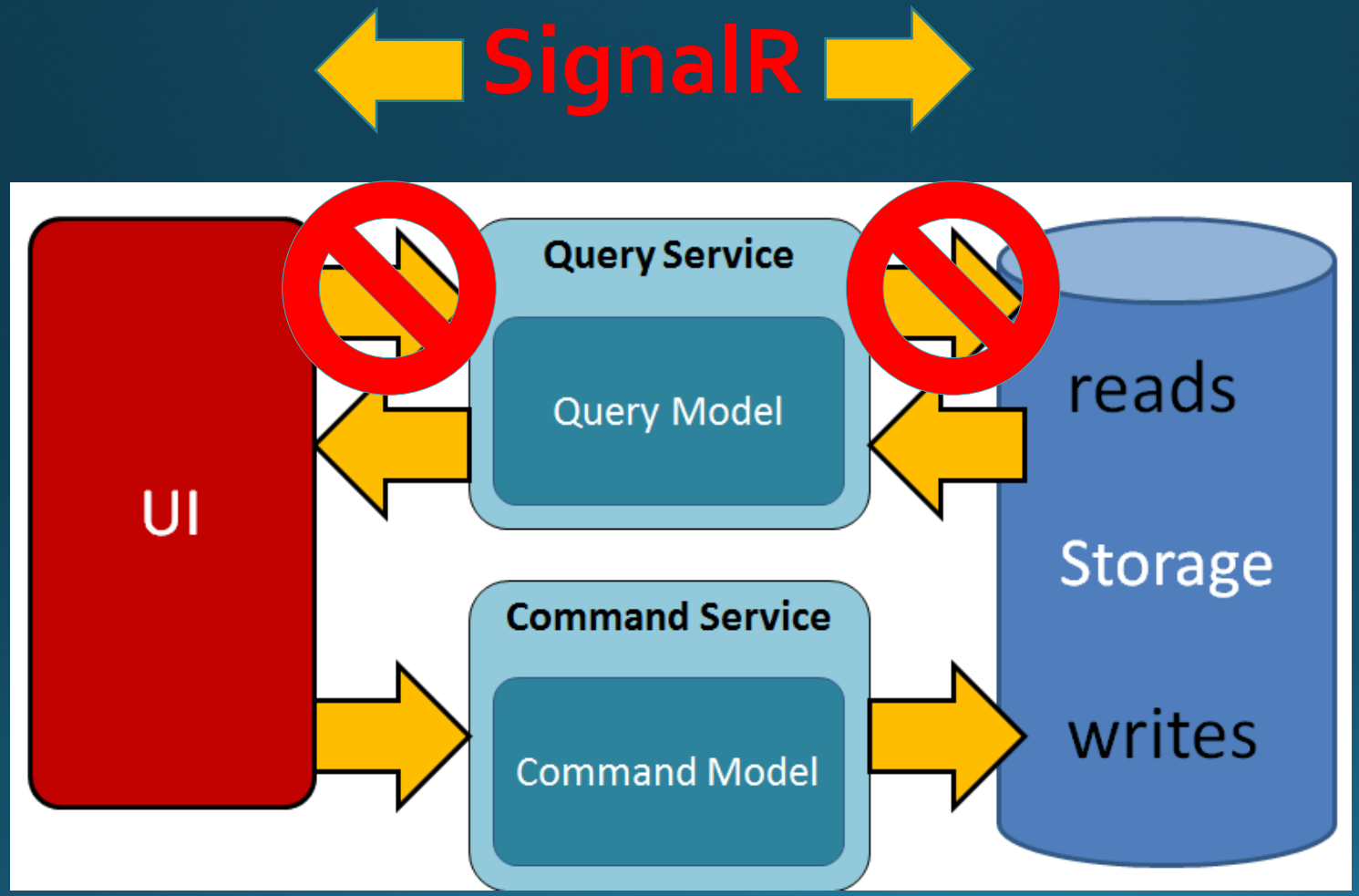



New Project
.NET Framework 4.5
Sort by:
Default
Search Installed Te
Installed
Templates
Visual C#
Visual C++
SQL Server
Visual F#
Silverli
F# Application
Visual F#
F# Library
F# Tutorial
Visual F#
Visual F#
Type: Visual F#
A tutorial project providing a walkthrough of the F# language
Name:
Tutorial1
Location:
C:\source\fsharpintro
Create new solution
Solution name:
Add to source control
OK
DEMO

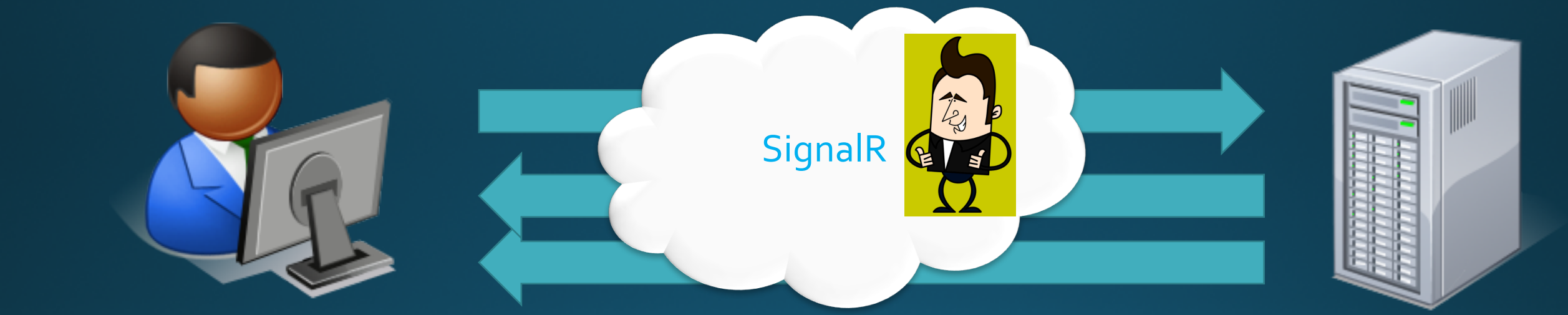
Stock Market Web Server



CQRS… How to keep the UI Updated?



What is SignalR?



Client

javaScript
.Net
WinRT

WebSocket
Long Polling
Server sent events
Forever Frame

Asp.Net
Self-Host
Owin

Server

The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities.

-- Edsger Dijkstra


That's all Folks!