Вебинар 3
Основы создания CRUD-приложений, продвинутый уровень

1) CQS & CQRS
2) MediatR
3) Event Sourcing

+ more DDD

(из этой же оперы - flux, redux, vuex)

Event-Driven,

CQRS,

etc.

Продолжаем погружаться в DDD

(но пока все еще не переходим
на Clean Architecture... хотя, почти)))

CQS – Command / Query Separation

https://abdullin.com/command-query-separation/

```
Result SaveUser(string name, int age)
{
    if (name == "") {
        return Error("Name is empty")
    }
    if (age <= 0 || age >= 100) {
        return Error("Age is out of range")
    }

    users.Add(NewUser(name, age))
    return null;
}
```

CQS – Command / Query Separation

```
void CreateUser(string name, int age)
{
    users.Add(NewUser(name, age));
}
```

```
Result ValidateUser(string name, int age)
{
    if name == "" {
        return NewError("Name is empty")
    }
    if age <= 0 || age >= 100 {
        return NewError("Age is out of range")
    }
    return null;
}
```

CQS – Command / Query Separation

```
Booking CreateBooking(BookingInfo info)
{
    var driver = _driverService.GetAvailableDriver();
    if (driver == null) return null;

    var booking = new Booking(driver, info);
    var addedBooking = _bookingRepository.Add(booking);

    return addedBooking;
}
```

CQS – Command / Query Separation

1

```
void CreateBooking(BookingInfo info, Driver driver)
{
    var booking = new Booking(driver, info);
    _bookingRepository.Add(booking);
}
```

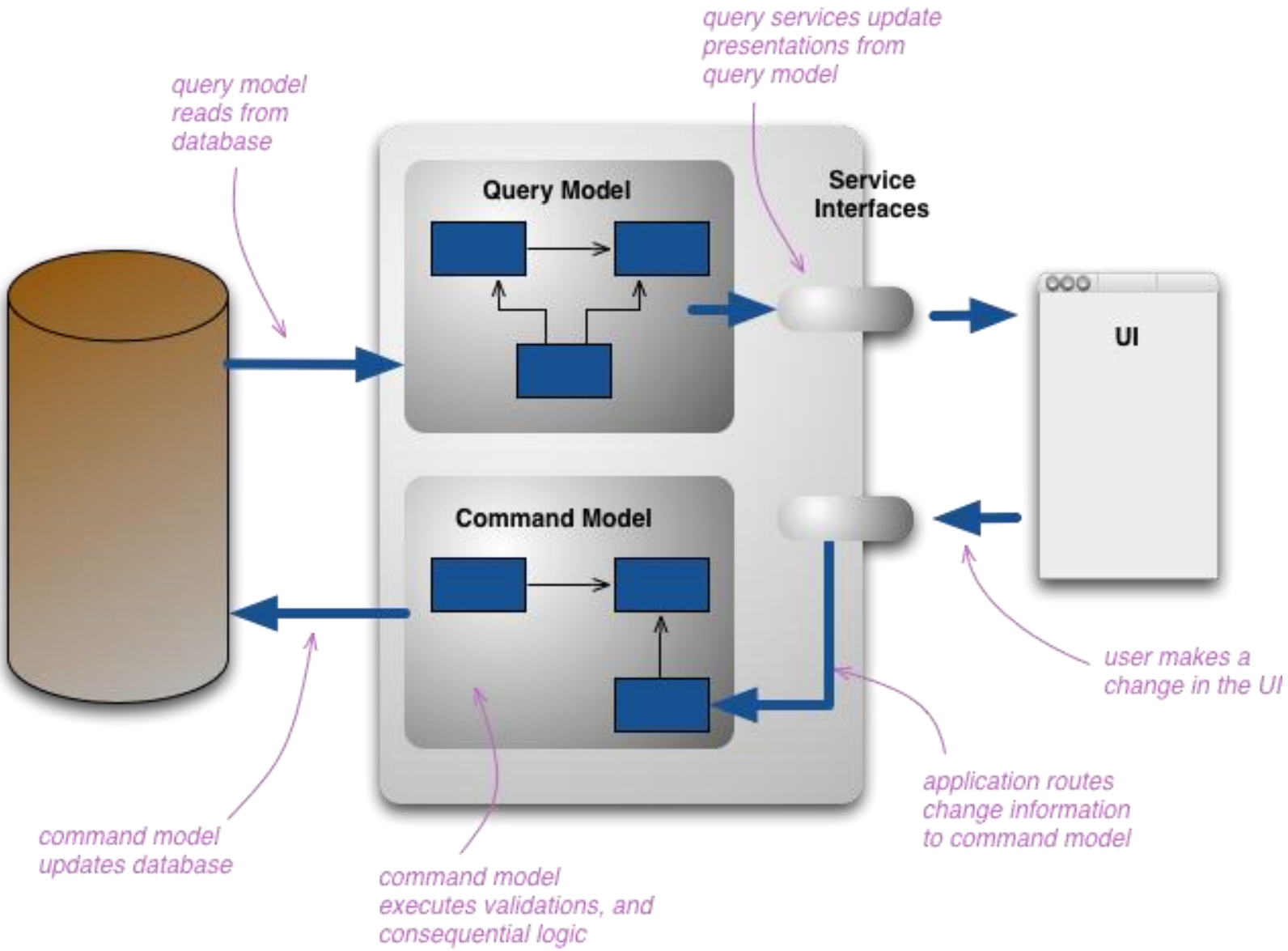
2

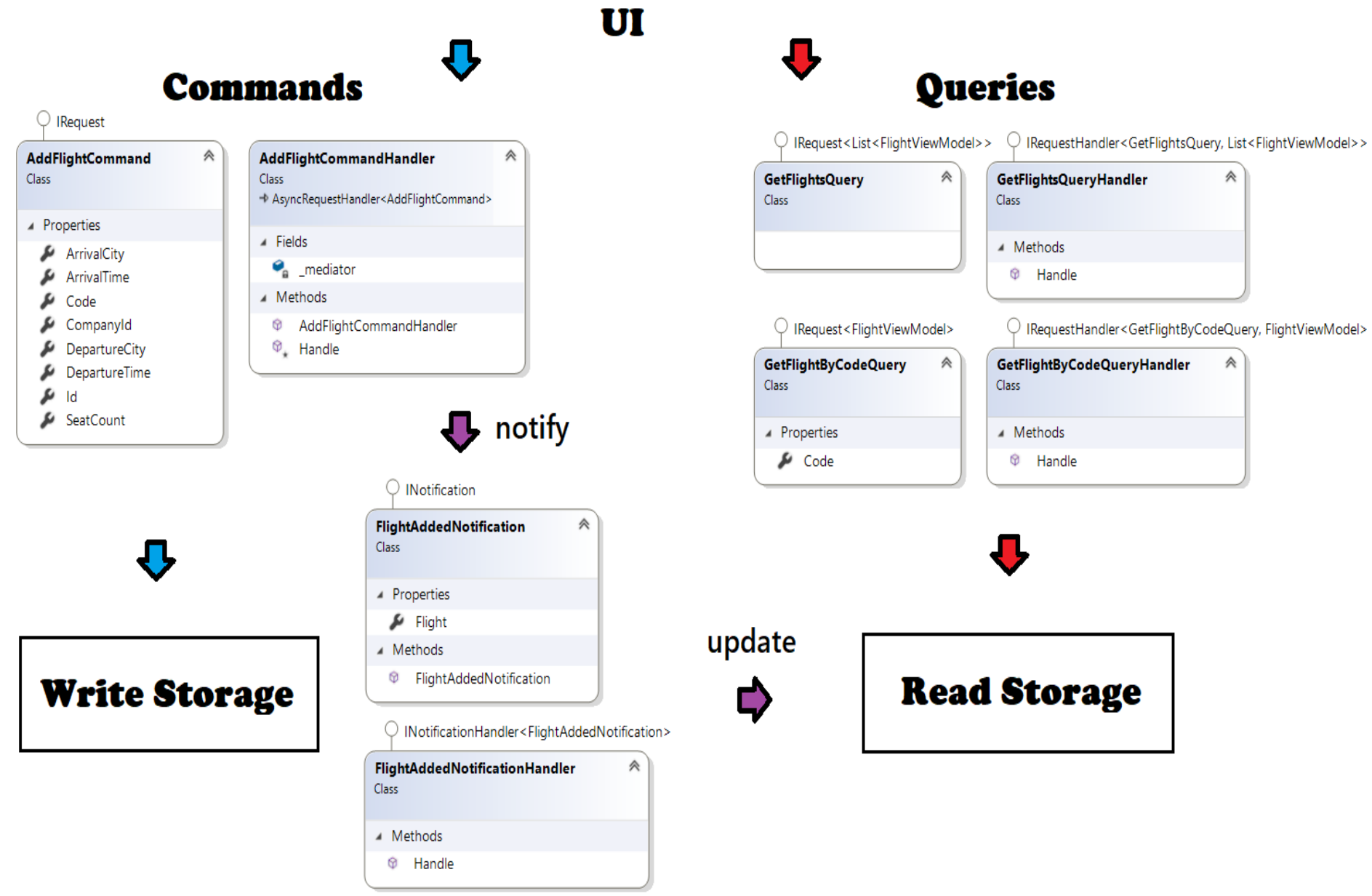
```
Driver FindAvailableDriver()
{
    return _driverService.GetAvailableDriver();
}
```

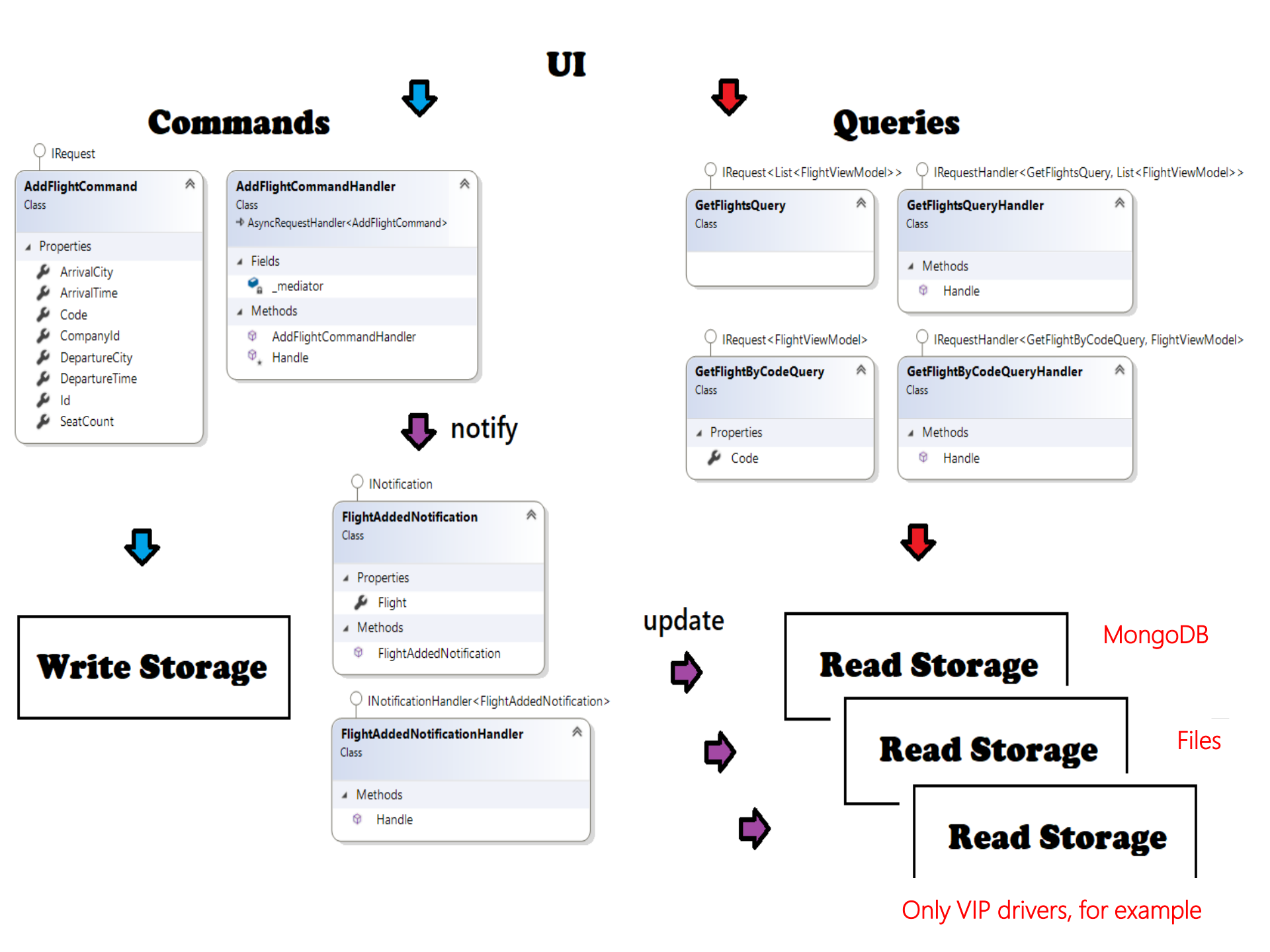
3

```
Booking GetBooking(BookingInfo info)
{
    return _bookingRepository.Get(info).LastOrDefault();
}
```

CQRS

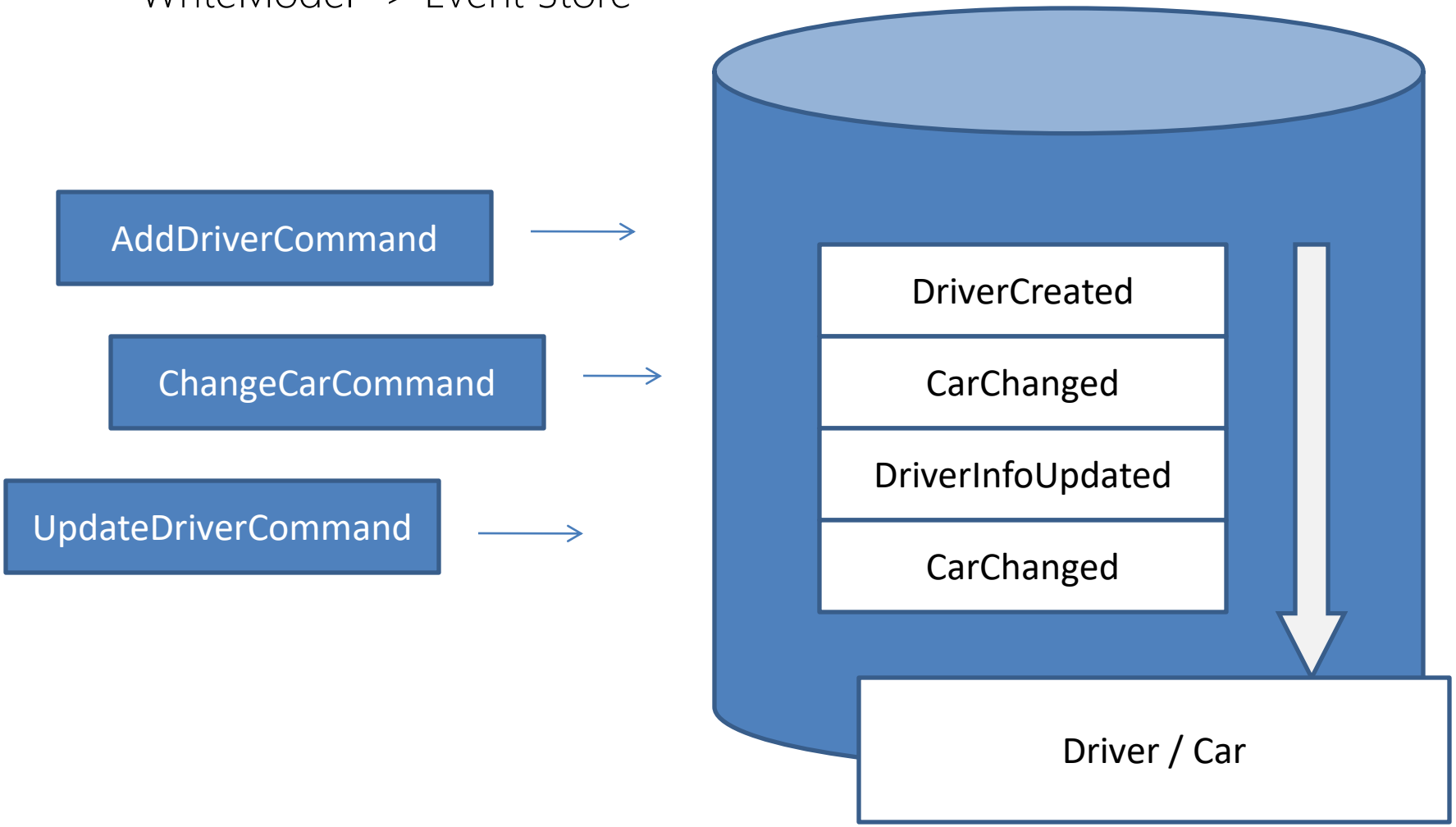

query services update
presentations from
query model
query model
reads from
database
Query Model
Service
Interfaces
UI
Command Model
user makes a
change in the UI
application routes
change information
to command model
command model
updates database
command model
executes validations, and
consequential logic


UI
Commands
Queries
IRequest
AddFlightCommand
Class
Properties
ArrivalCity
ArrivalTime
Code
CompanyId
DepartureCity
DepartureTime
Id
SeatCount
AddFlightCommandHandler
Class
AsyncRequestHandler<AddFlightCommand>
Fields
_mediator
Methods
AddFlightCommandHandler
Handle
IRequest<List<FlightViewModel>>
GetFlightsQuery
Class
IRequestHandler<GetFlightsQuery, List<FlightViewModel>>
GetFlightsQueryHandler
Class
Methods
Handle
IRequest<FlightViewModel>
GetFlightByCodeQuery
Class
Properties
Code
IRequestHandler<GetFlightByCodeQuery, FlightViewModel>
GetFlightByCodeQueryHandler
Class
Methods
Handle
notify
INotification
FlightAddedNotification
Class
Properties
Flight
Methods
FlightAddedNotification
INotificationHandler<FlightAddedNotification>
FlightAddedNotificationHandler
Class
Methods
Handle
update
Write Storage
Read Storage


UI
Commands
Queries
IRequest
AddFlightCommand
Class
Properties
ArrivalCity
ArrivalTime
Code
CompanyId
DepartureCity
DepartureTime
Id
SeatCount
AddFlightCommandHandler
Class
AsyncRequestHandler<AddFlightCommand>
Fields
_mediator
Methods
AddFlightCommandHandler
Handle
IRequest<List<FlightViewModel>>
IRequestHandler<GetFlightsQuery, List<FlightViewModel>>
GetFlightsQuery
Class
GetFlightsQueryHandler
Class
Methods
Handle
IRequest<FlightViewModel>
IRequestHandler<GetFlightByCodeQuery, FlightViewModel>
GetFlightByCodeQuery
Class
Properties
Code
GetFlightByCodeQueryHandler
Class
Methods
Handle
notify
INotification
FlightAddedNotification
Class
Properties
Flight
Methods
FlightAddedNotification
INotificationHandler<FlightAddedNotification>
FlightAddedNotificationHandler
Class
Methods
Handle
update
Write Storage
Read Storage
Read Storage
Read Storage
MongoDB
Files
Only VIP drivers, for example

Event Sourcing:

WriteModel -> Event Store



CQRS != Event Sourcing

AirportCqrs –
чистый базовый CQRS

Webinar3App.AspNetMvc –
промежуточный тип проекта между чистым CQRS и CQRS + EventSourcing

https://martinfowler.com/bliki/CQRS.html

Despite these benefits, **you should be very cautious about using CQRS**. Many information systems fit well with the notion of an information base that is updated in the same way that it's read, adding CQRS to such a system can add significant complexity. I've certainly seen cases where it's made a significant drag on productivity, adding an unwarranted amount of risk to the project, even in the hands of a capable team. So while CQRS is a pattern that's good to have in the toolbox, beware that it is difficult to use well and you can easily chop off important bits if you mishandle it.

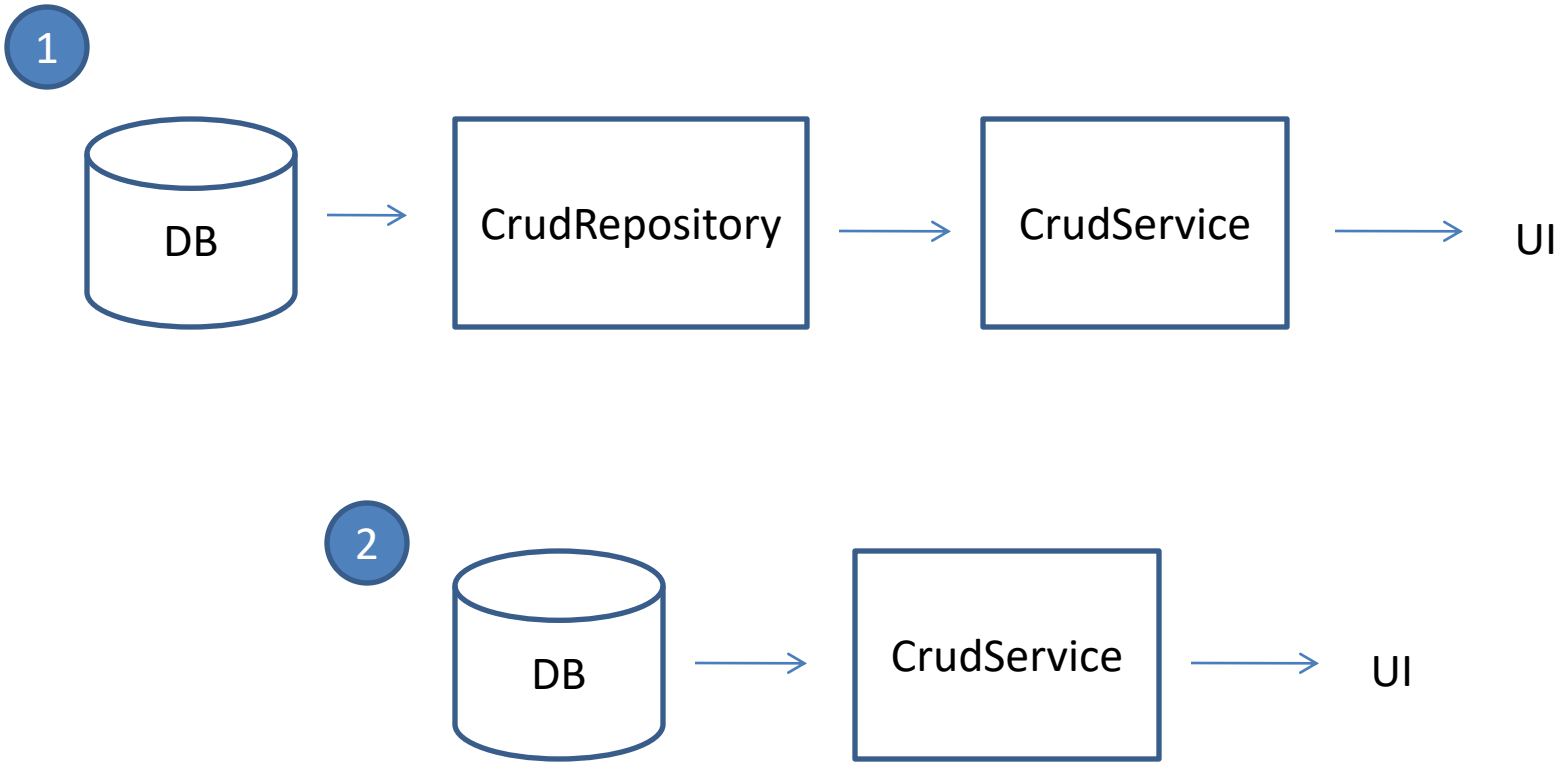
(...это в 2011-ом)

Using an event driven architecture makes a lot of sense, not only for achieving clear command/query separation, but also because it opens new architectural choices and usually fits with an asynchronous programming model (useful if you need to scale your architecture).

+

https://blog.byndyu.ru/2014/07/command-and-query-responsibility.html

CRUDошлепие форева!



Domain Driven Development

Entity

Value Object

Aggregate

+ RDM vs. ADM

Aggregate

Совокупность взаимосвязанных объектов, которые мы воспринимаем как единое целое с точки зрения изменения данных.

У каждого агрегата есть корневой объект и есть граница. Граница определяет, что находится внутри агрегата. Корневой объект — это один конкретный объект-сущность (Entity), содержащийся в агрегате. Корневой объект — единственный член агрегата, на который могут ссылаться внешние объекты, в то время как объекты, заключенные внутри границы, могут ссылаться друг на друга как угодно. Сущности, отличные от корневого объекта, локально индивидуальны, но различаться они должны только в пределах агрегата, поскольку никакие внешние объекты все равно не могут их видеть вне контекста корневой сущности.

1) Mediator pattern (GOF)
2) Responsible for some part of BL

Mediator

Event Aggregator
Event Bus

1) Observer pattern (GOF)
2) No BL, just forwarding events

common: reduce coupling

Mediator

Event Aggregator

Event Bus

Mediator

Event Aggregator

Event Bus