

Введение в Event sourcing

Яков Повар

Обо мне

- team lead в Positive Technologies
- в .NET с 2011г.
- разрабатываю системы в области ИБ

https://t.me/jacob_povar

<https://github.com/jacobpovar>



О чем поговорим

введение

event sourcing – теория

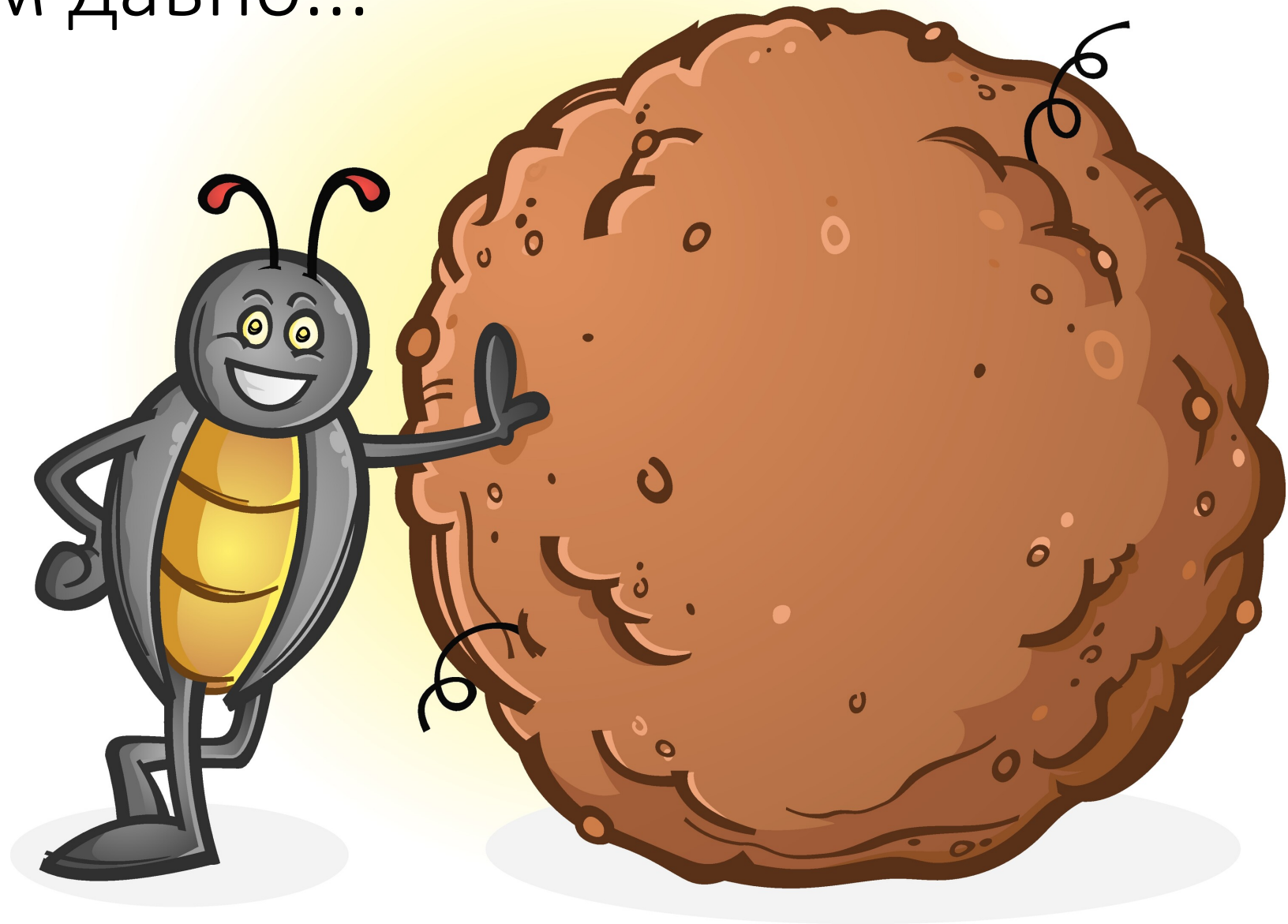
чуть-чуть ближе к коду

версионирование

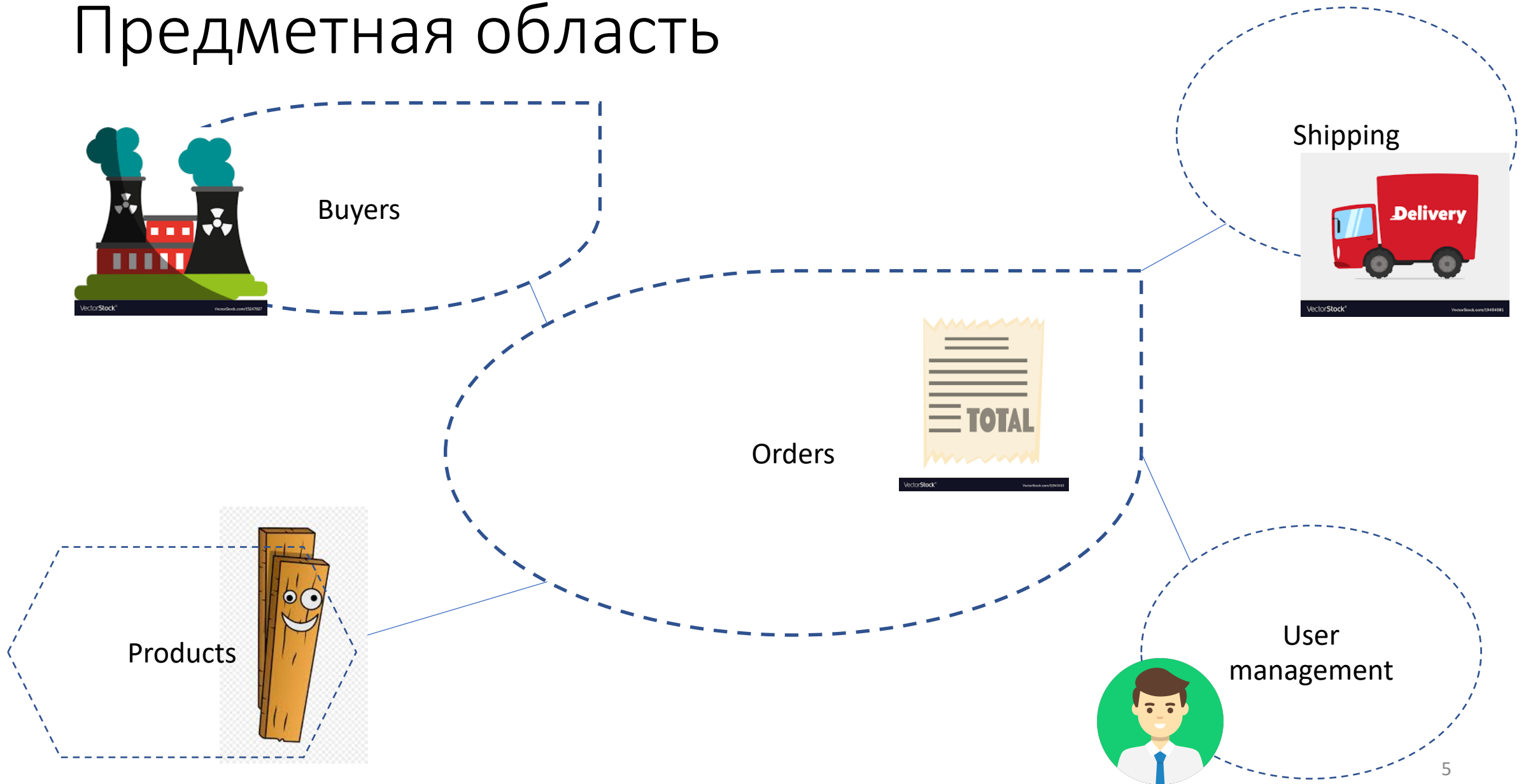
распределенный event sourcing

ИТОГИ

Давным давно...



Предметная область



Что мы хотим от системы?

- нужны сложные выборки по данным

Что мы хотим от системы?

- нужны сложные выборки по данным
- запросов на чтение больше чем на запись

95 %
READ

5 %
WRITE

Что мы хотим от системы?

- нужны сложные выборки по данным
- запросов на чтение больше чем на запись
- история всех изменений, возможность откатить изменения

Что мы хотим от системы?

- нужны сложные выборки по данным
- запросов на чтение больше чем на запись
- история всех изменений, возможность откатить изменения
- возможность развивать приложение

Что мы хотим от системы?

- нужны сложные выборки по данным
- запросов на чтение больше чем на запись
- история всех изменений, возможность откатить изменения
- возможность развивать приложение
- **распределенность**
 - географическая (например, пользовательские данные)
 - мобильные приложения

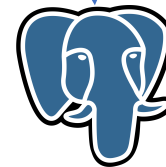
Традиционная архитектура

Application

```
var order = new Order(...);  
orderRepository.Save(order);
```

Domain model

```
class Order  
{  
  ...  
}
```



```
class OrdersContext : DbContext  
{  
  public DbSet<Order> Orders { ... }  
}
```

Чем плохо

- object-relational impedance mismatch

Чем плохо

- object-relational impedance mismatch
- тяжелые, медленные выборки

Чем плохо

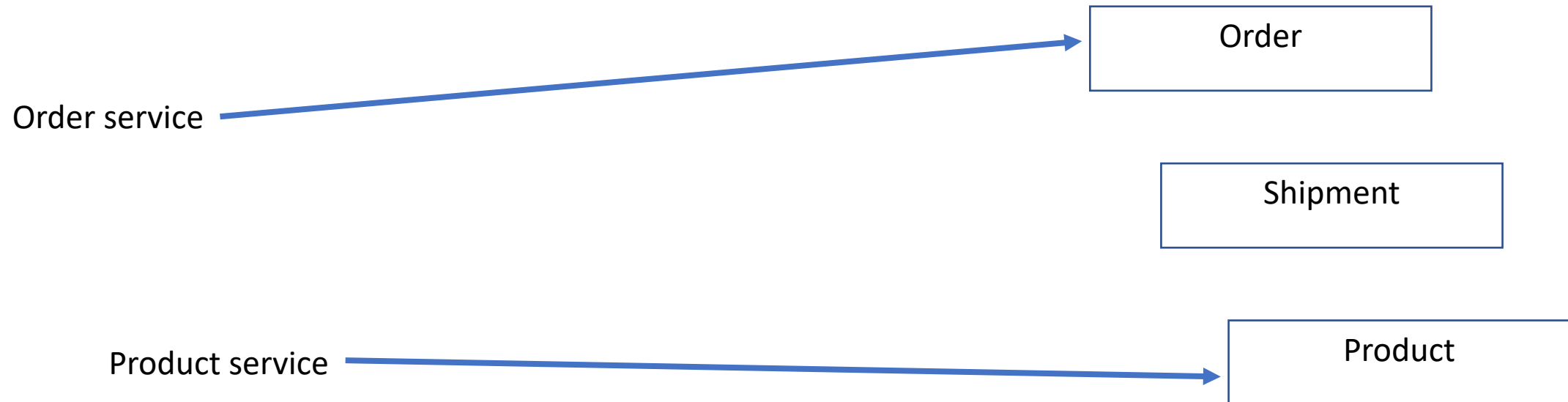
- **теряем знания** о работе нашей системы
- - какие данные изменились?
- - почему они изменились?

Id	Статус
42	Ошибка



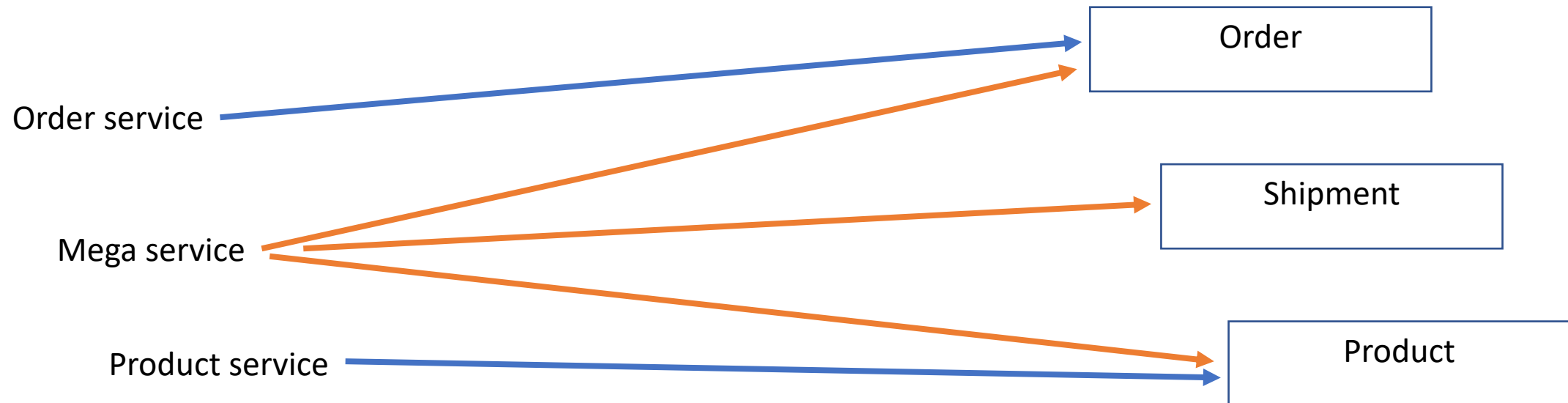
Чем плохо

- OrderService, ProductService – сервисы, привязанные к сущностям



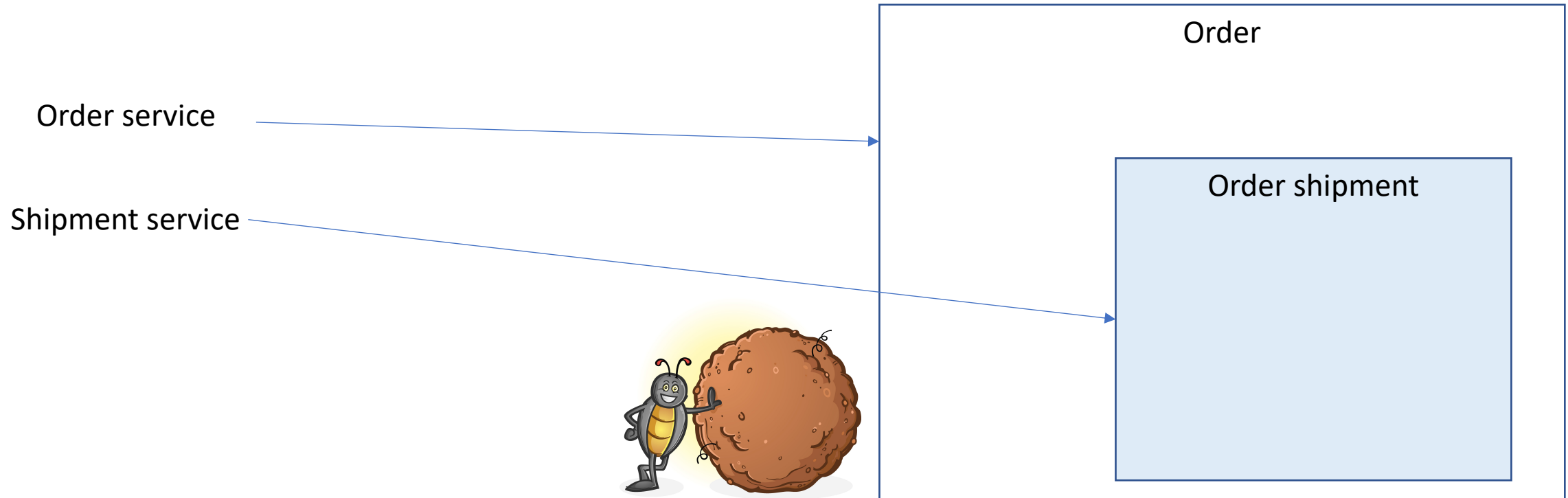
Чем плохо

- OrderService, ProductService – сервисы, привязанные к сущностям



Чем плохо

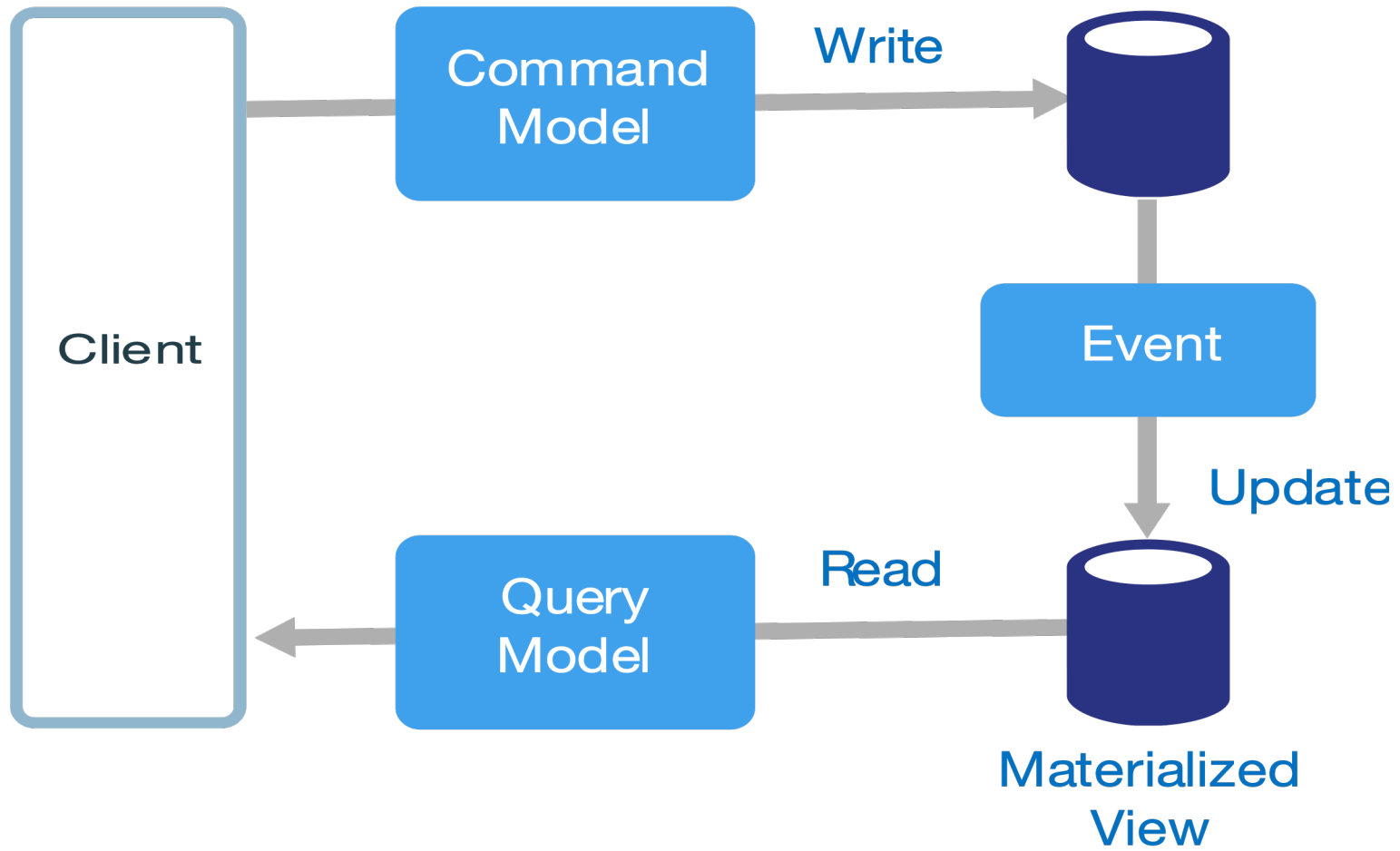
- легко нарушить границы агрегатов



Цели

- **нужны сложные выборки по данным**
- **запросов на чтение больше чем на запись**
- история всех изменений, возможность откатить изменения
- возможность развивать приложение
- распределенность

CQRS



Цели

- **нужны сложные выборки по данным**
- ✓ запросов на чтение больше чем на запись
- **история всех изменений, возможность откатить изменения**

Хочу видеть заказы у которых сегодня
добавили новые товары

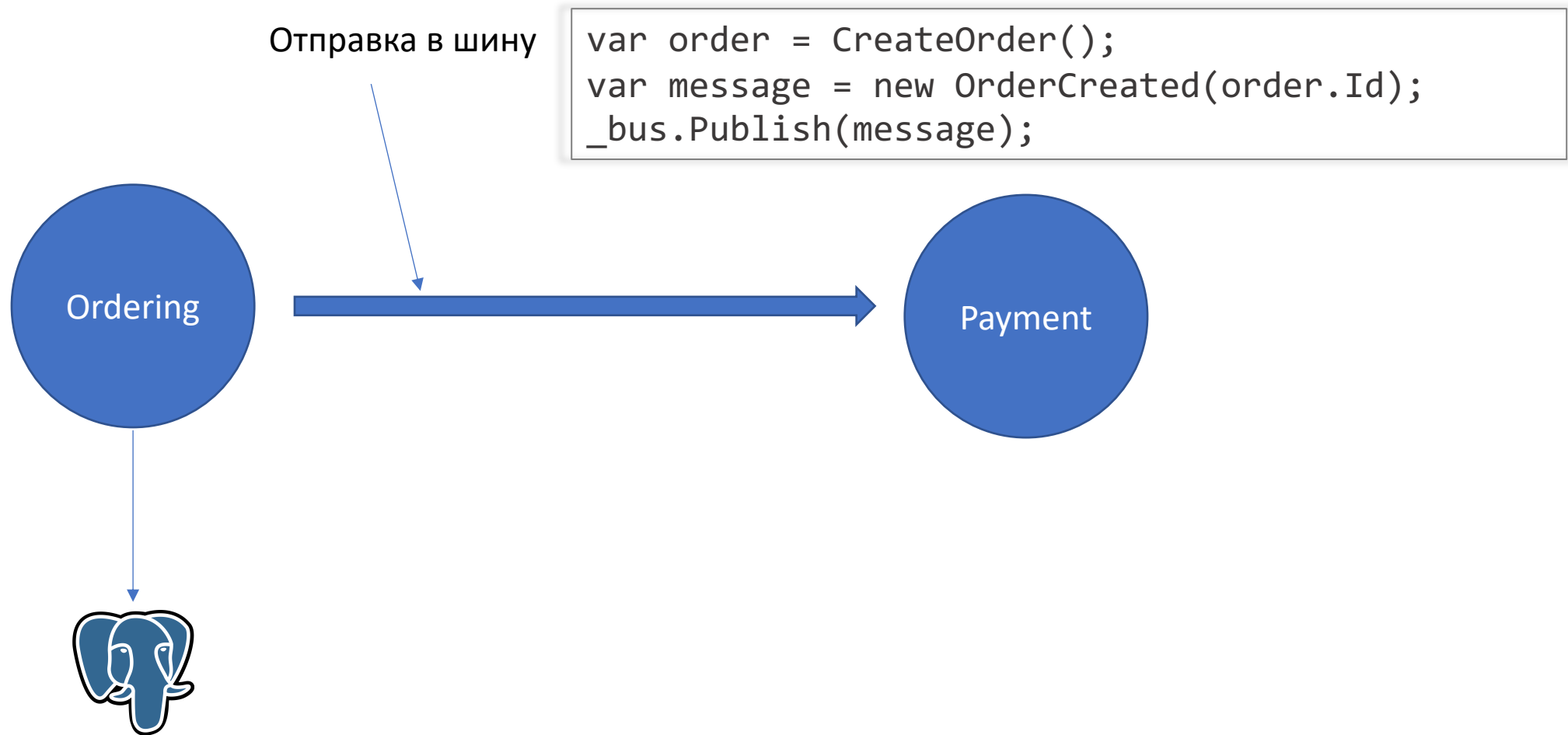


Event sourcing

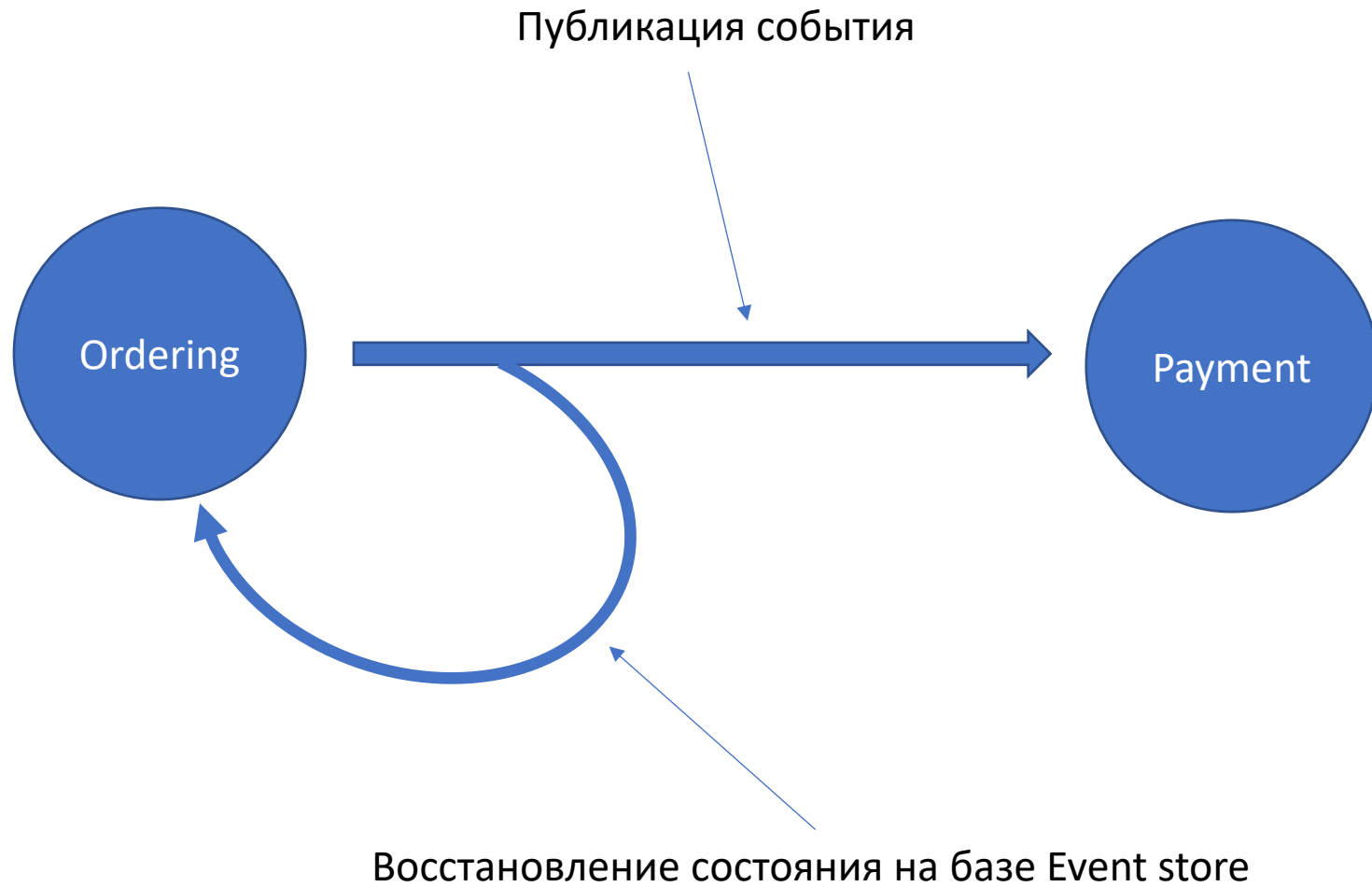
- события являются **единственным первоначальным источником знаний о том, что произошло в системе.**
- Другие компоненты используют события для того чтобы построить на **их основе** свое внутреннее состояние
- события неизменяемы и хранятся в Event store



Event – driven architecture

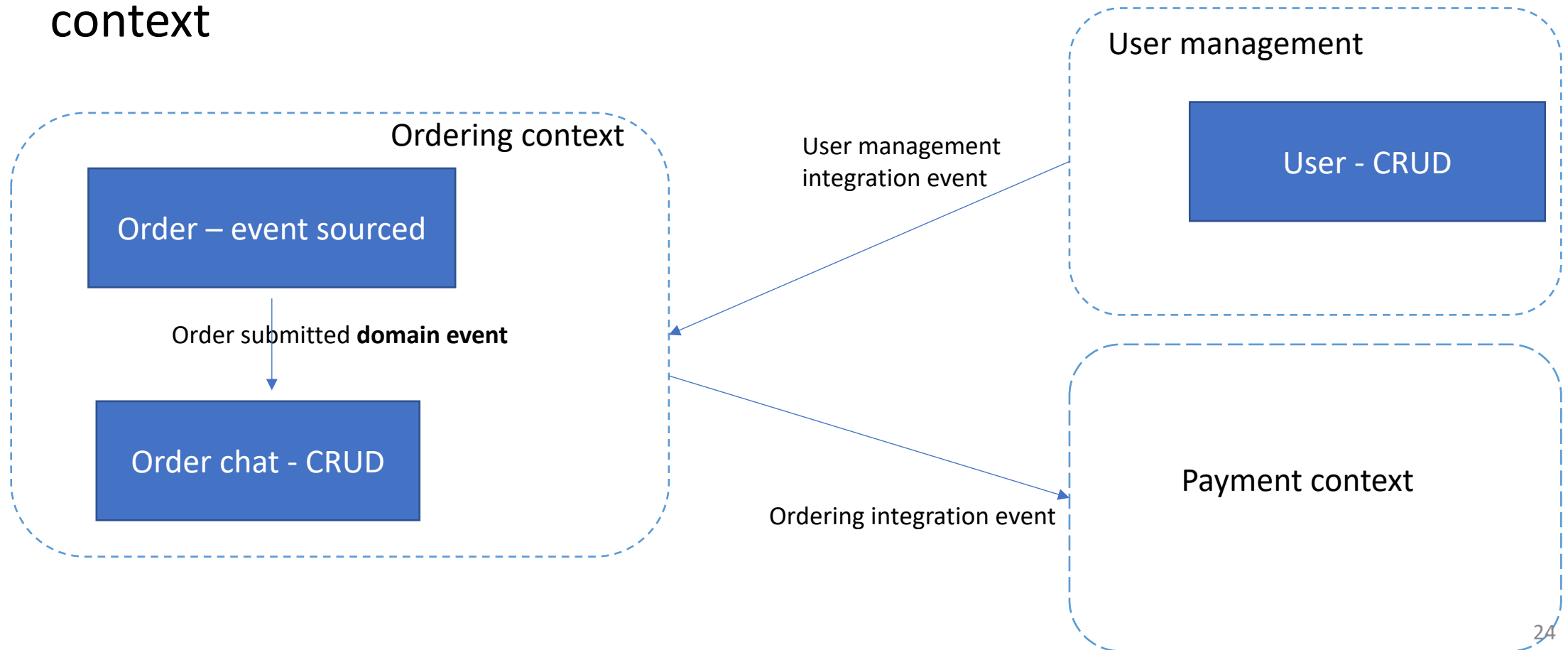


Event sourcing



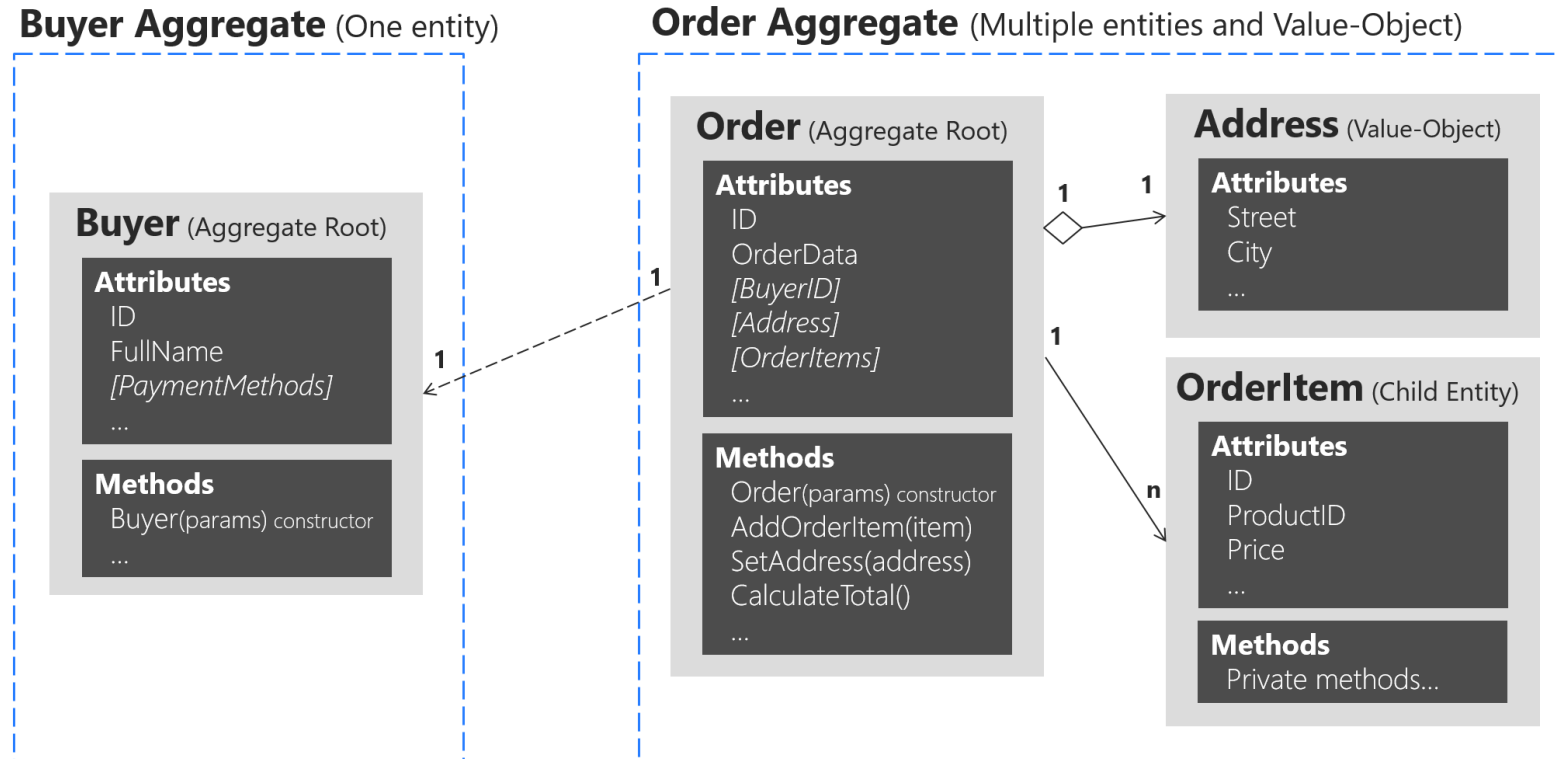
Где применять Event sourcing?

- Определяем область применения CQRS/ES, например Bounded context



Как применять Event sourcing?

Aggregate pattern



Какие события у нас есть?

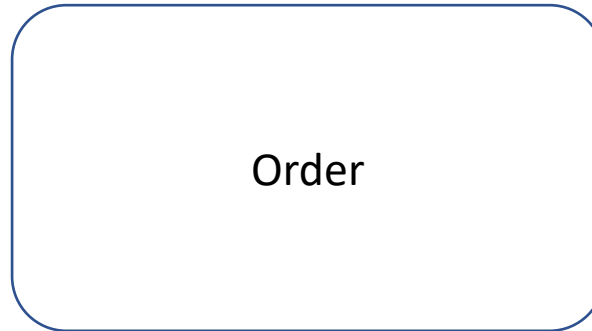
- OrderPlaced – создан новый заказ
- OrderAssignedToManager – заказ назначен на оператора
- OrderItemAdded – к заказ добавлен товар
- OrderSumbitted – заказ отправлен на оплату
- BuyerCancelledOrder – покупатель отклонил заказ
- ...

Любое изменение состояния - событие

Place order



BuyerId: 42



Любое изменение состояния - событие

Place order



BuyerId: 42

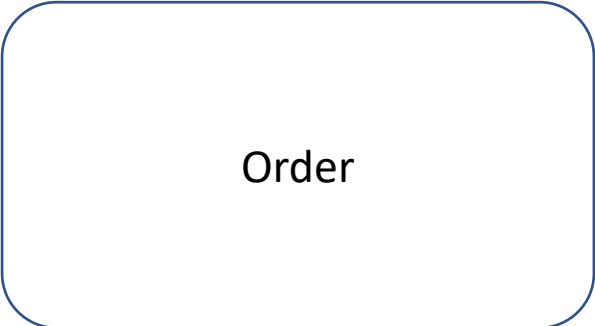


Любое изменение состояния - событие

Place order

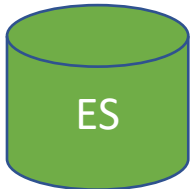
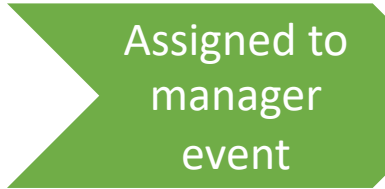
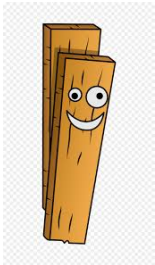


BuyerId: 42



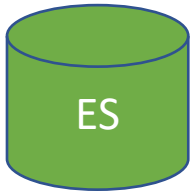
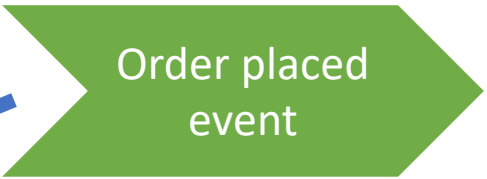
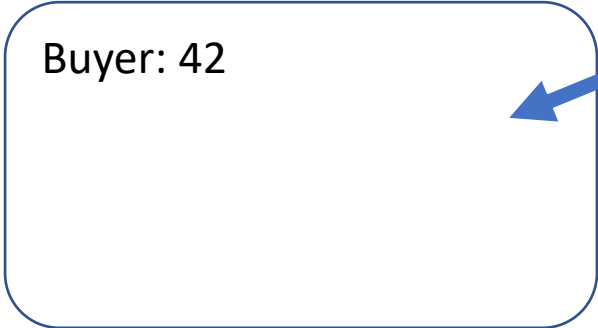
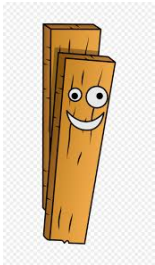
Любое изменение состояния - событие

Add order item



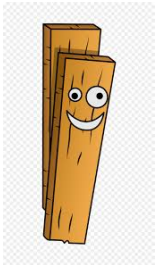
Любое изменение состояния - событие

Add order item

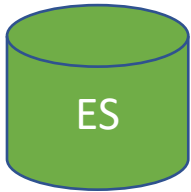
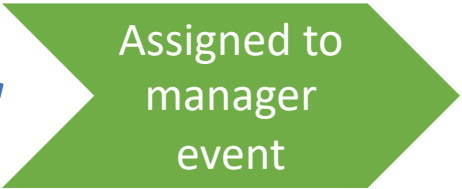
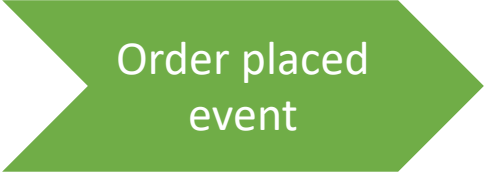


Любое изменение состояния - событие

Add order item



Buyer: 42




Любое изменение состояния - событие


Add order item



Buyer: 42



Доски



Order placed event

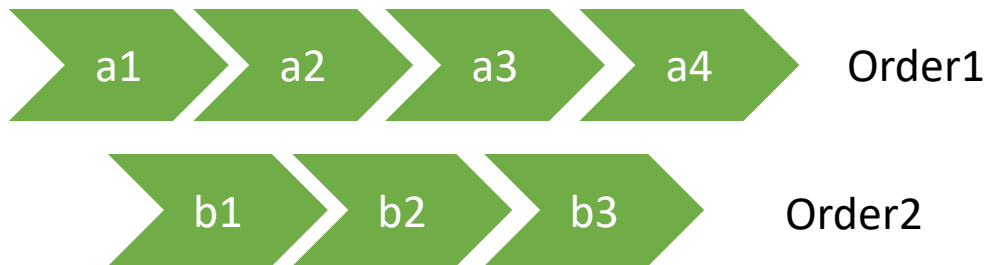
Assigned to manager event

Item added event

ES

Как применять Event sourcing?

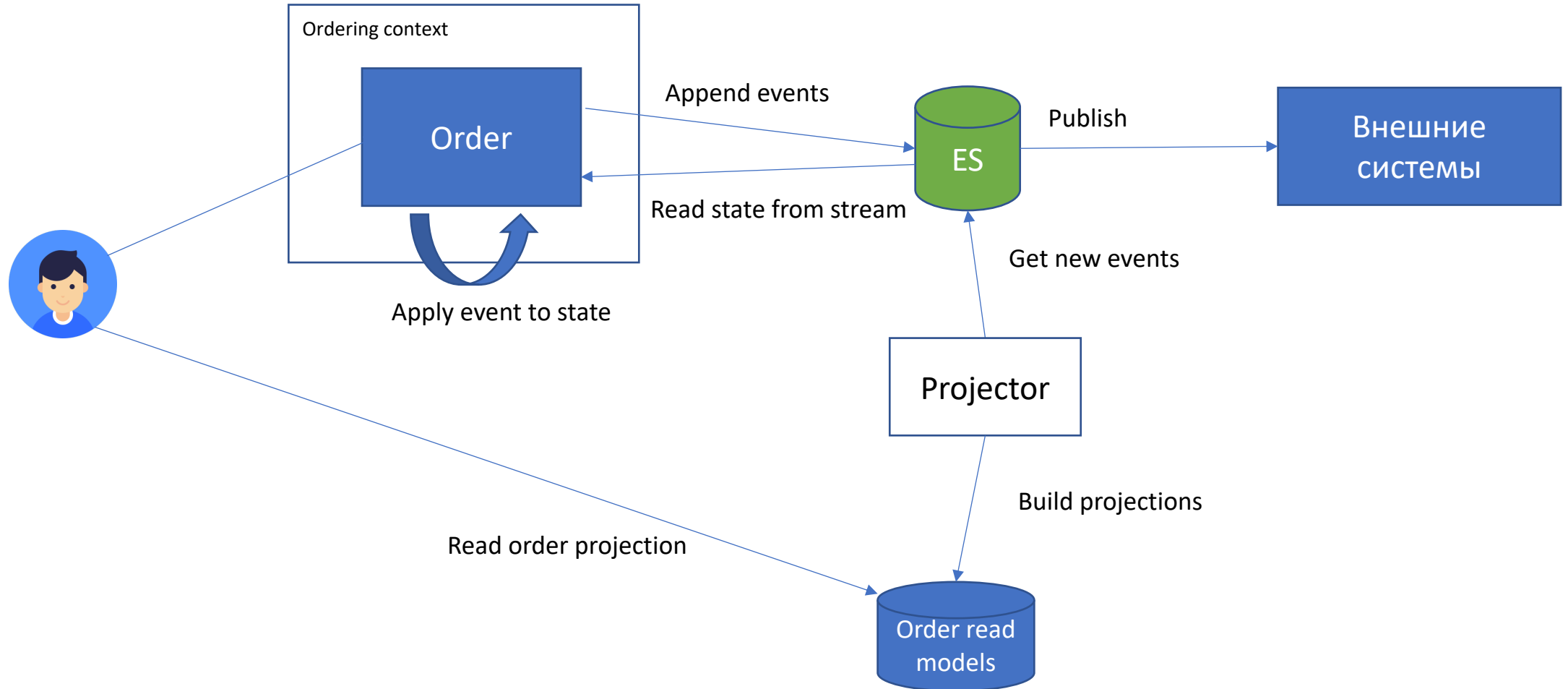
- Любое изменение состояния - события
- События привязываются к агрегату
(streamId, version)



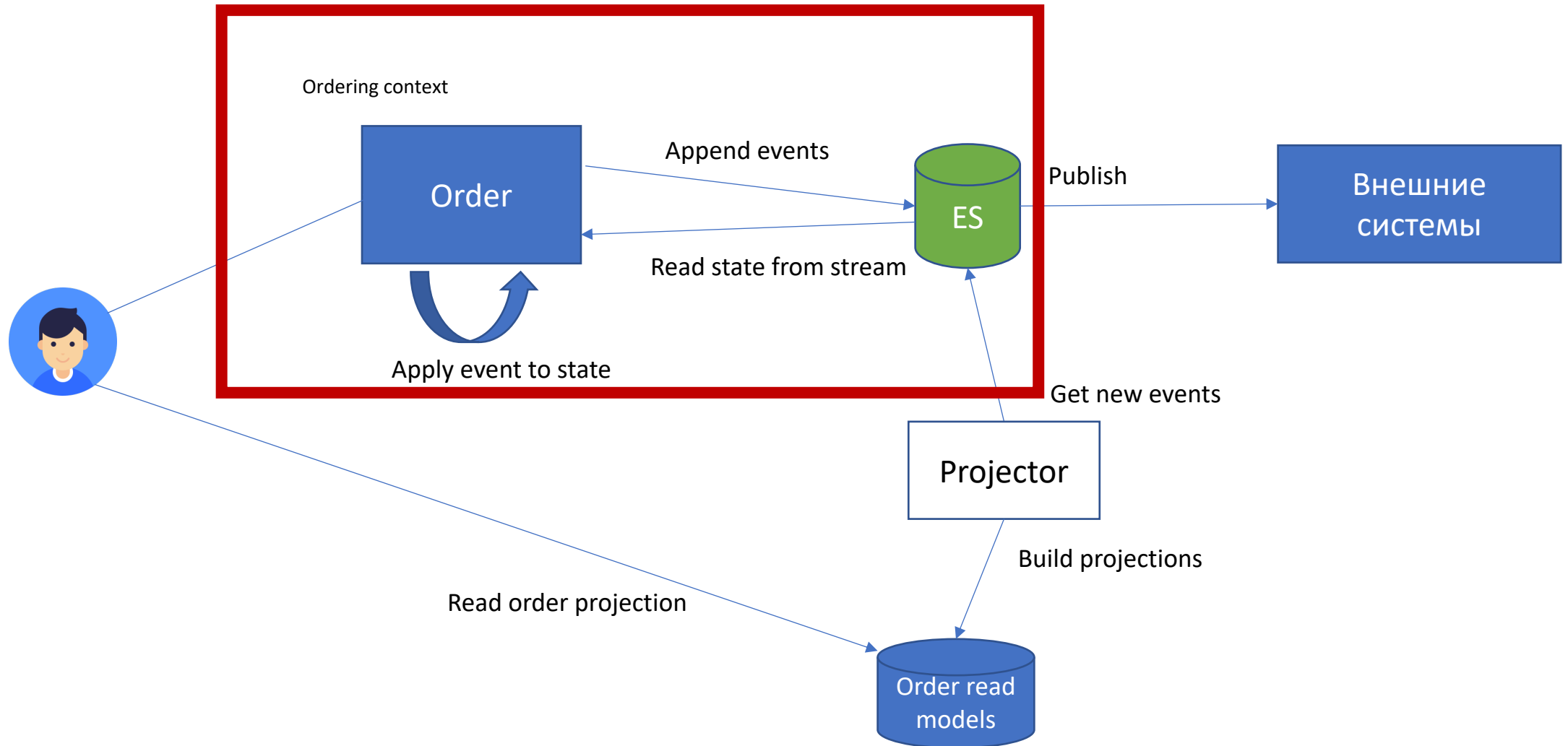
Как применять Event sourcing?

- Любое изменение состояния - события
- События привязываются к агрегату
(streamId, version)
- События никогда не валидируются

Event sourced архитектура



Write часть



Реализация – события заказа

```
class OrderPlaced
{
    public Guid Buyer { get; set; }
}
```

```
class OrderItemAdded
{
    public string ItemId { get; set; }
}
```

Создание агрегата

```
class Order
{
    public Order(Guid id, Guid buyer)
    {
        Validate(...);

        var event = new OrderPlaced
        {
            AggregateId = id,
            Buyer = buyer
        };

        PendingEvents.Add(event);
        Apply(event);
    }
}
```

Создание агрегата

```
public void Apply(object event)
{
    switch (event)
    {
        case OrderPlaced created:
            When(created);
            break;
        ...
    }
}

public void When(OrderPlaced event)
{
    Id = event.AggregateId;
    Buyer = event.Buyer;
}
```


Модификация

```
public void AddLineItem(string lineItemName)
{
    var event = new OrderItemAdded
    {
        ItemId = lineItemName
    };

    PendingEvents.Add(event);
    Apply(event);
}
```

```
public void When(OrderItemAdded event) => LineItems.Add(event.ItemId);
```

Реализация - Event store

```
Task AppendEvents(string aggregateId, IEnumerable<object> events);
```

```
IReadOnlyCollection<EventEnvelope> ReadStream(string aggregateId);
```

```
class EventEnvelope
```

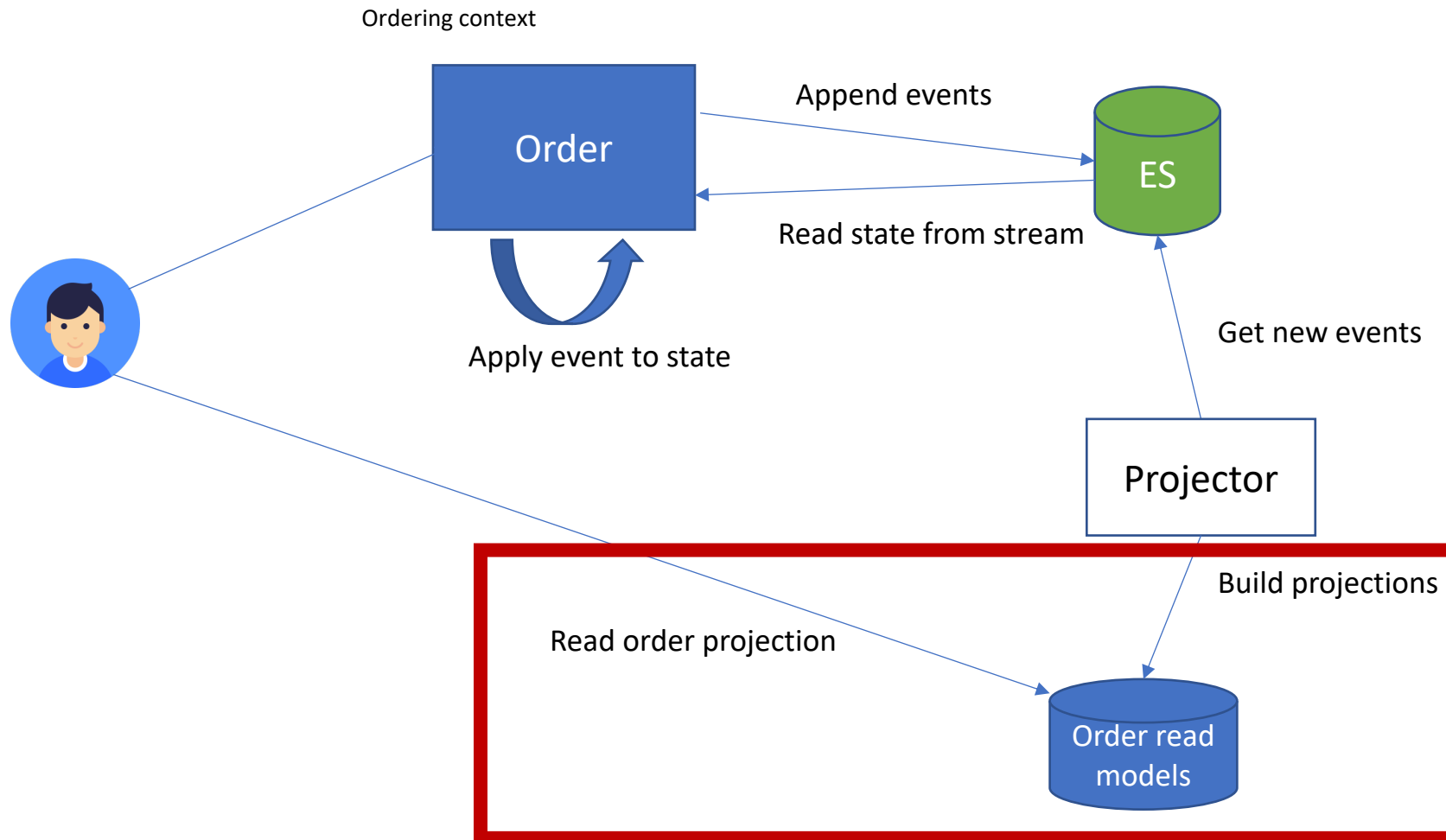
```
{
```

```
    object Data { get; set; } // само событие
```

```
    EventMetadata Metadata { get; set; } // метаданные события
```

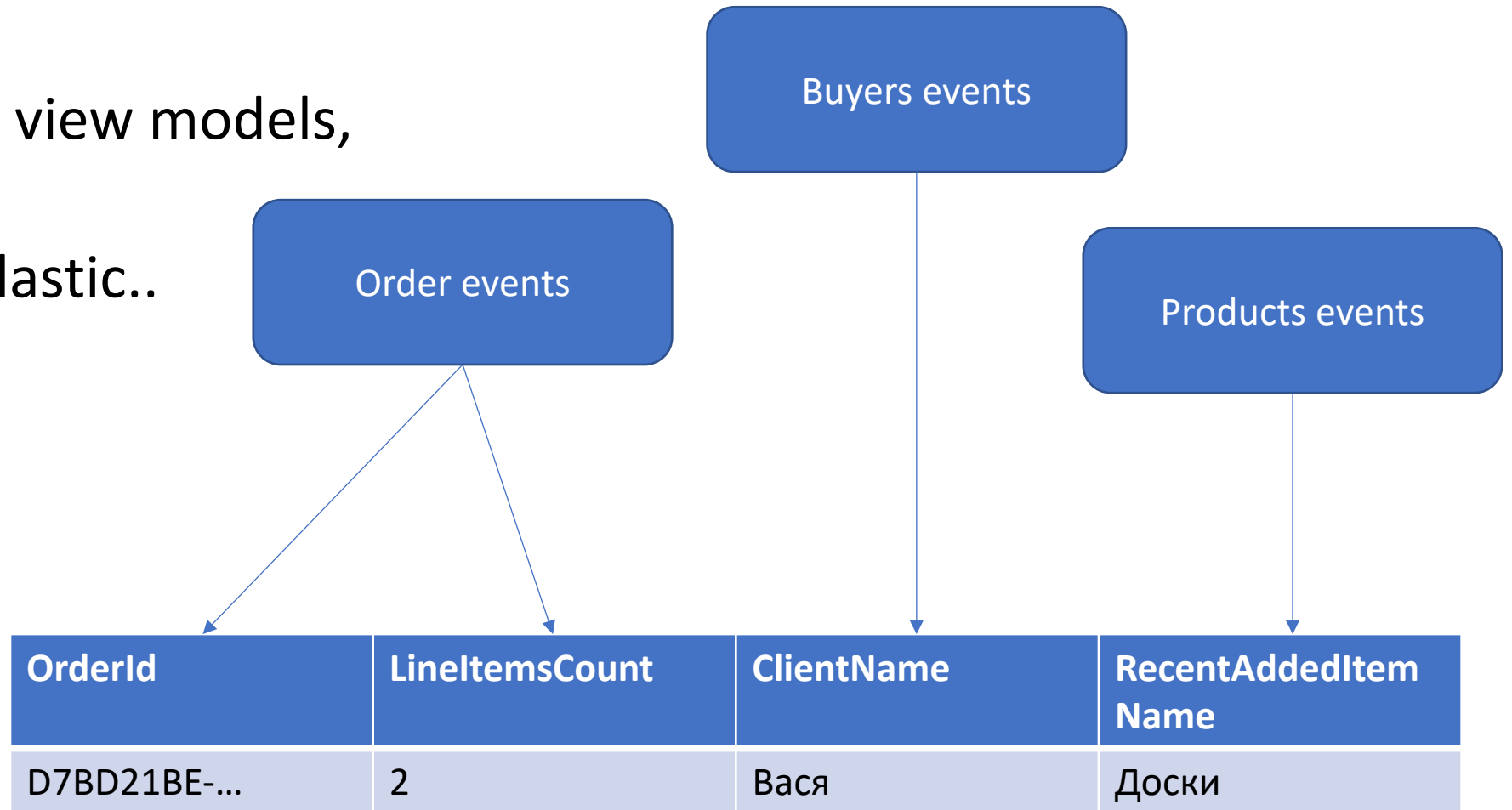
```
}
```

Read часть

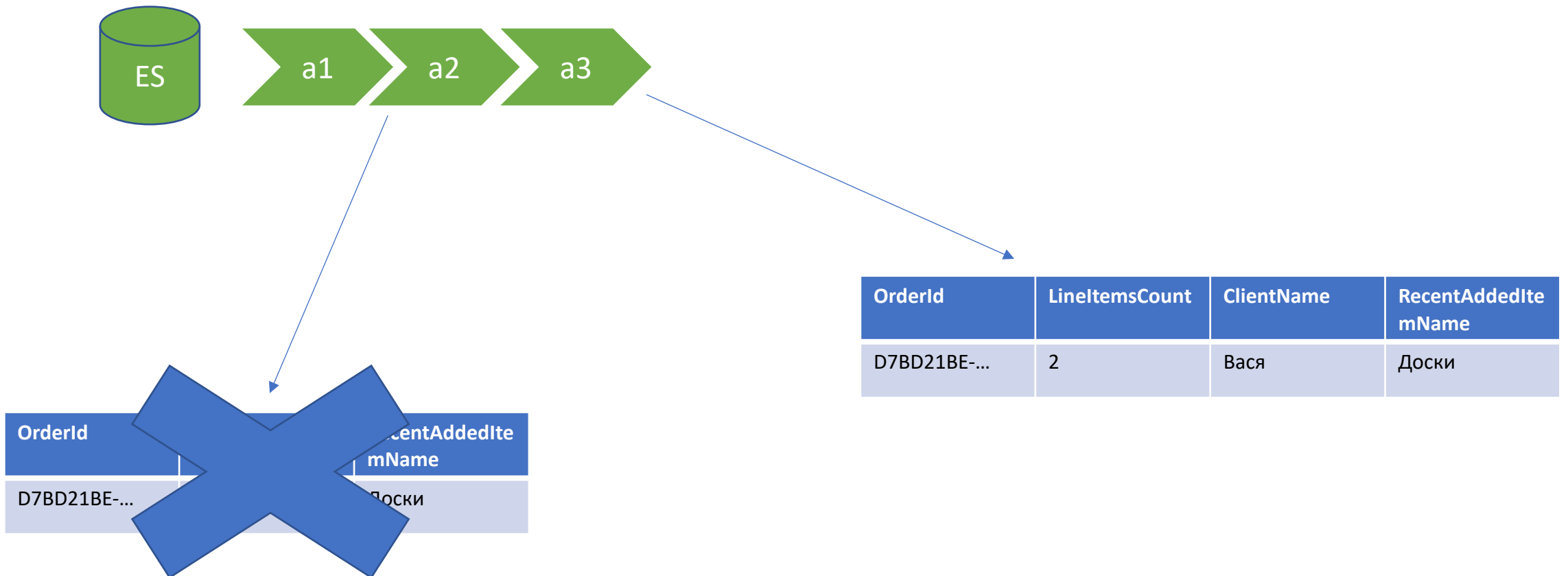


Read часть

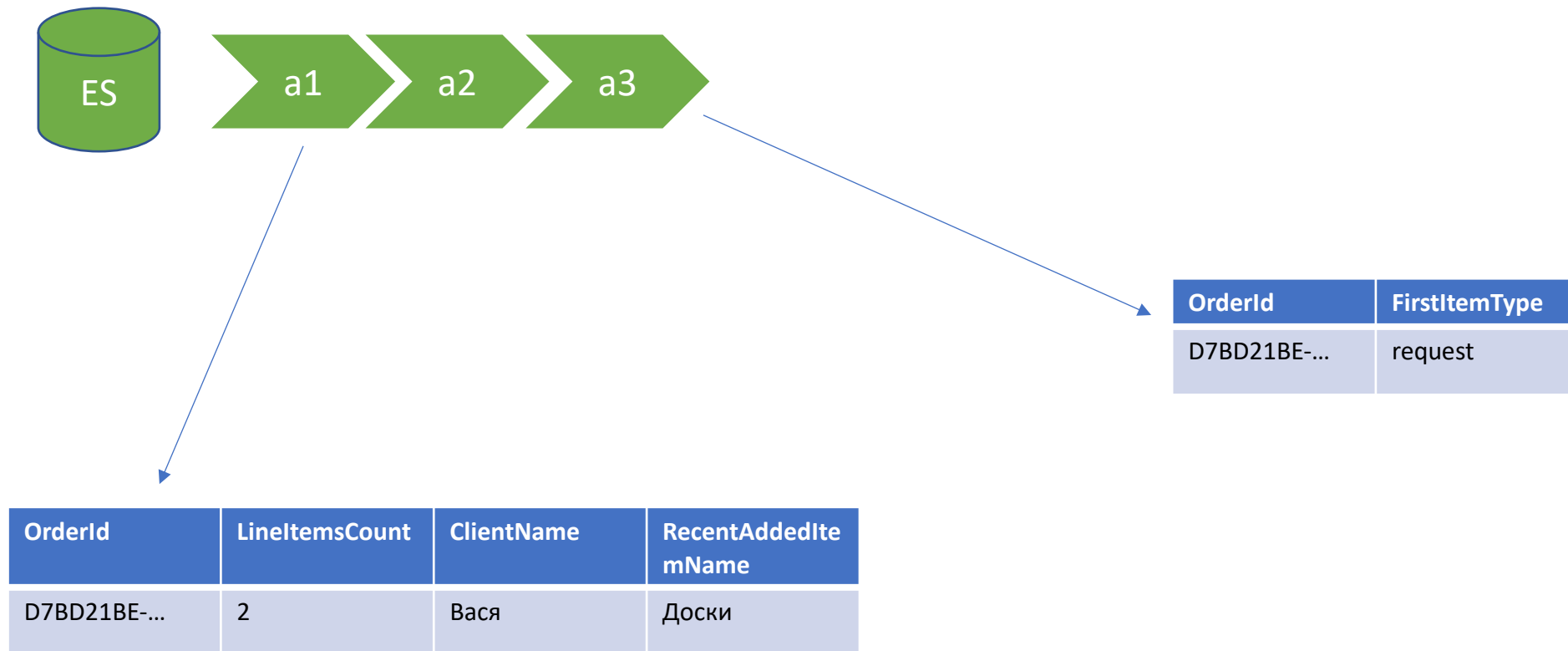
- Read models, view models, projections
- СУБД, json, Elastic..



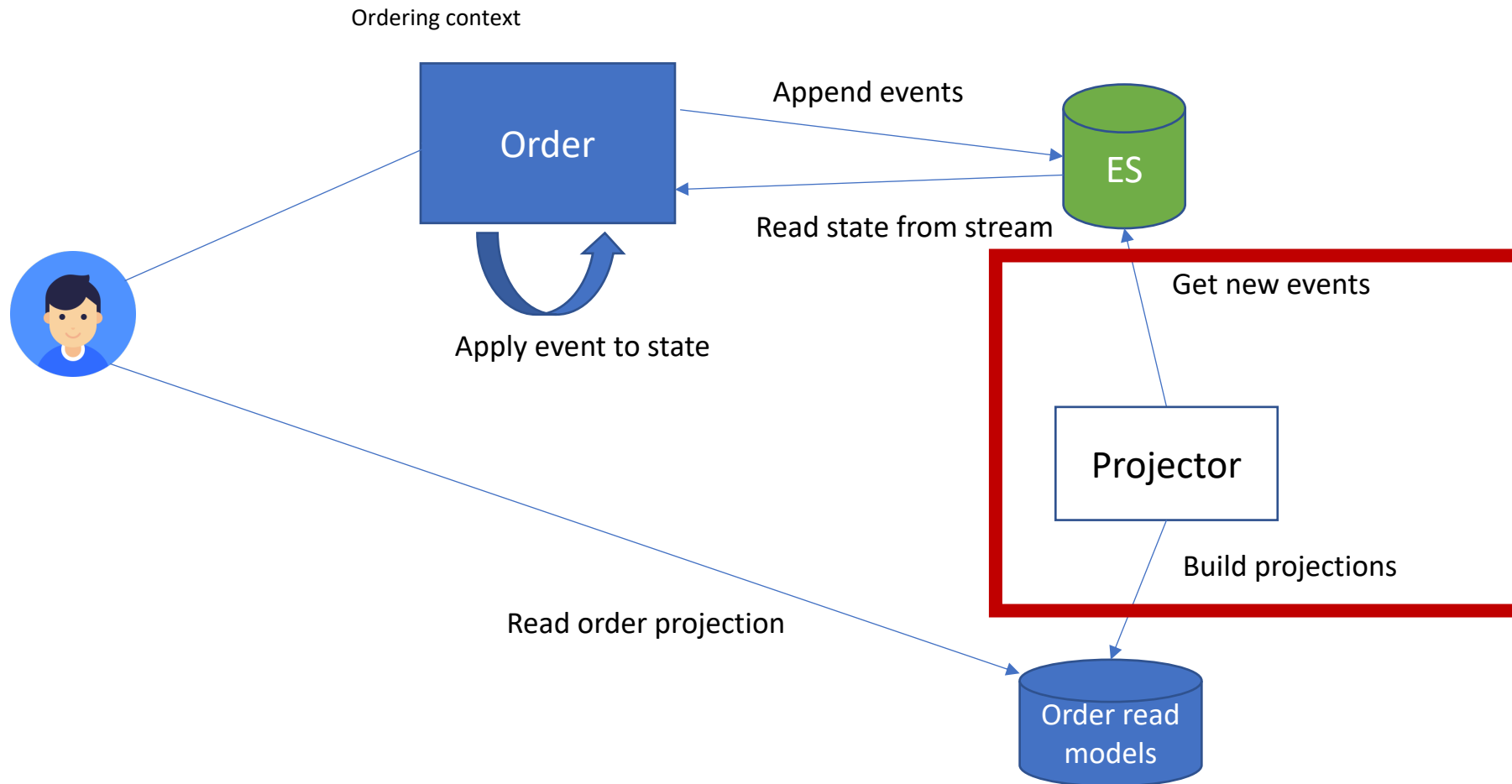
Процесс миграции данных



Можем добавить новое представление данных



Доставка событий



Доставка событий

- Sync vs async

Доставка событий

- Sync vs async
- Push vs pull

Доставка событий

- Sync vs async
- Push vs pull

```
while (true)
{
    var eventsPage = await GetNextPage(lastEncountered);

    foreach (var event in eventsPage.Events)
    {
        await BuildReadModel(event);
    }

    ...
    lastEncountered = eventsPage.LastSequenceNumber;
}
```

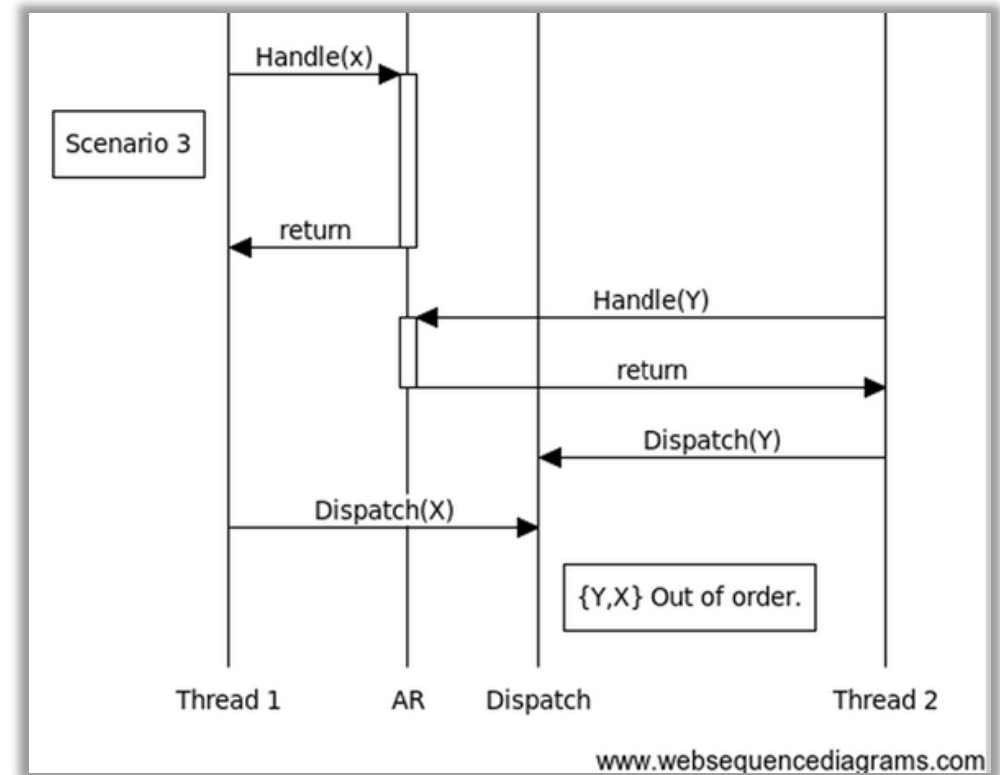
Message bus для доставки событий

```
_eventStore.Save(events);  
foreach (var e in events)  
{  
    _messageBus.Publish(e);  
}
```

Message bus для доставки событий

```
_eventStore.Save(events);  
foreach (var e in events)  
{  
    _messageBus.Publish(e);  
}
```

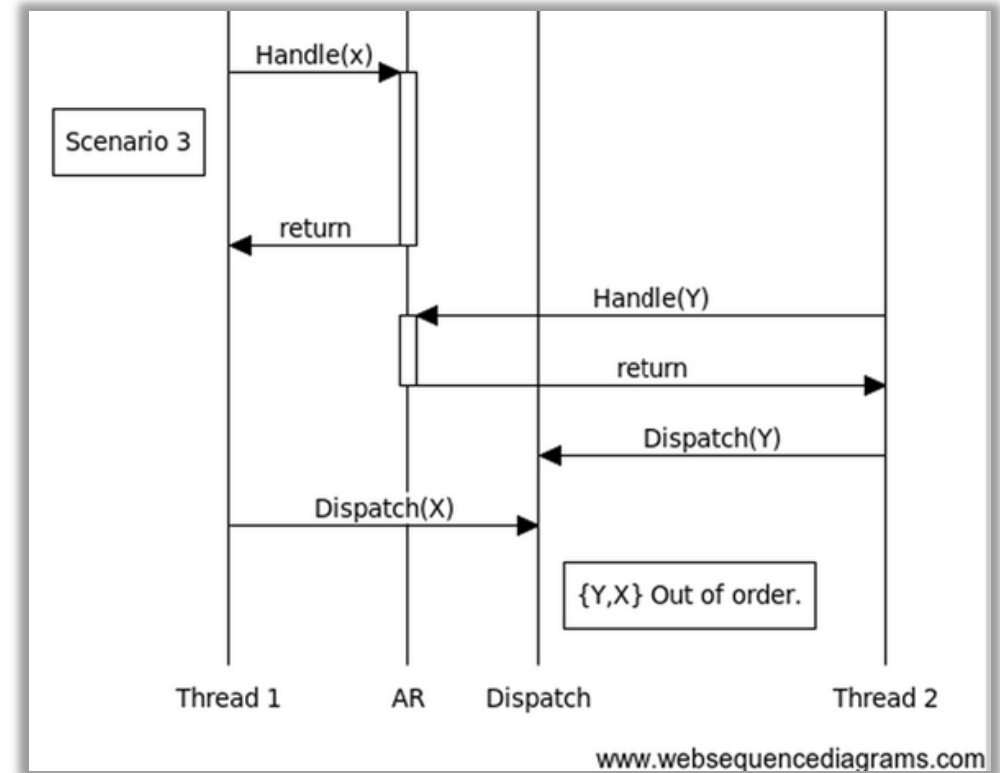
- Новые подписчики
- Out of order publish



Message bus для доставки событий

```
_eventStore.Save(events);  
foreach (var e in events)  
{  
    _messageBus.Publish(e);  
}
```

- Новые подписчики
- Out of order publish
- Redelivery, последовательность доставки
- Батчи событий?



Плюсы Event sourcing

- Доменная модель, свободная от деталей хранения

Плюсы Event sourcing

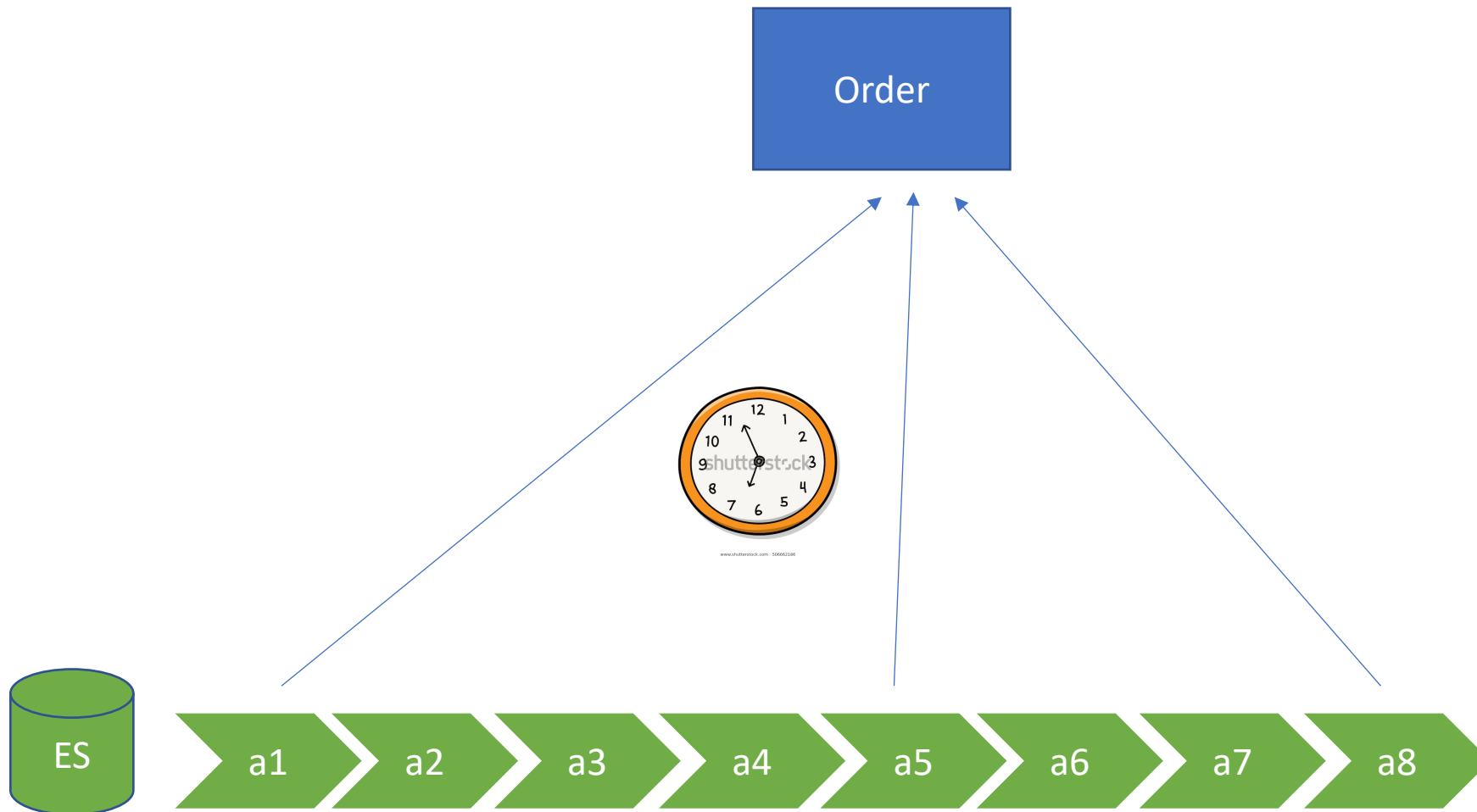
- Доменная модель, свободная от деталей хранения
- Не теряем знания

Плюсы Event sourcing

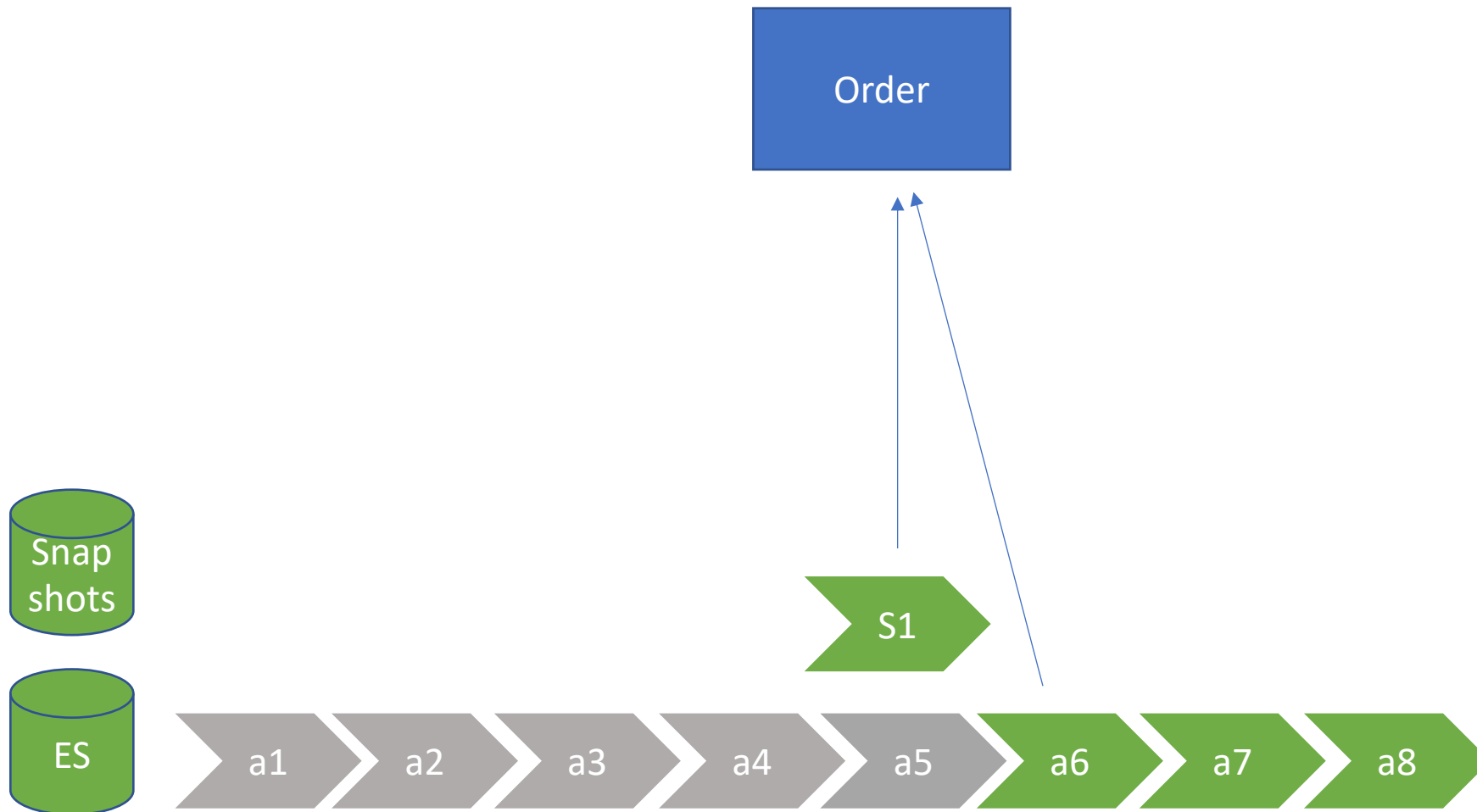
- Доменная модель, свободная от деталей хранения
- Не теряем знания
- Аудит изменений как бонус
 - 2019-08-13 13:45 | Менеджер Яков | применил скидку 10% исходя из статуса клиента
 - 2019-08-13 13:59 | Менеджер Яков | подтвердил заказ
 - 2019-08-14 09-34 | Система | Заказ оплачен клиентом, сумма 42\$
 - 2019-08-14 10-45 | Система | Заказ передан в службу доставки, TrackingNumber: 000123123

Но есть минусы..

Event sourcing – медленно ?

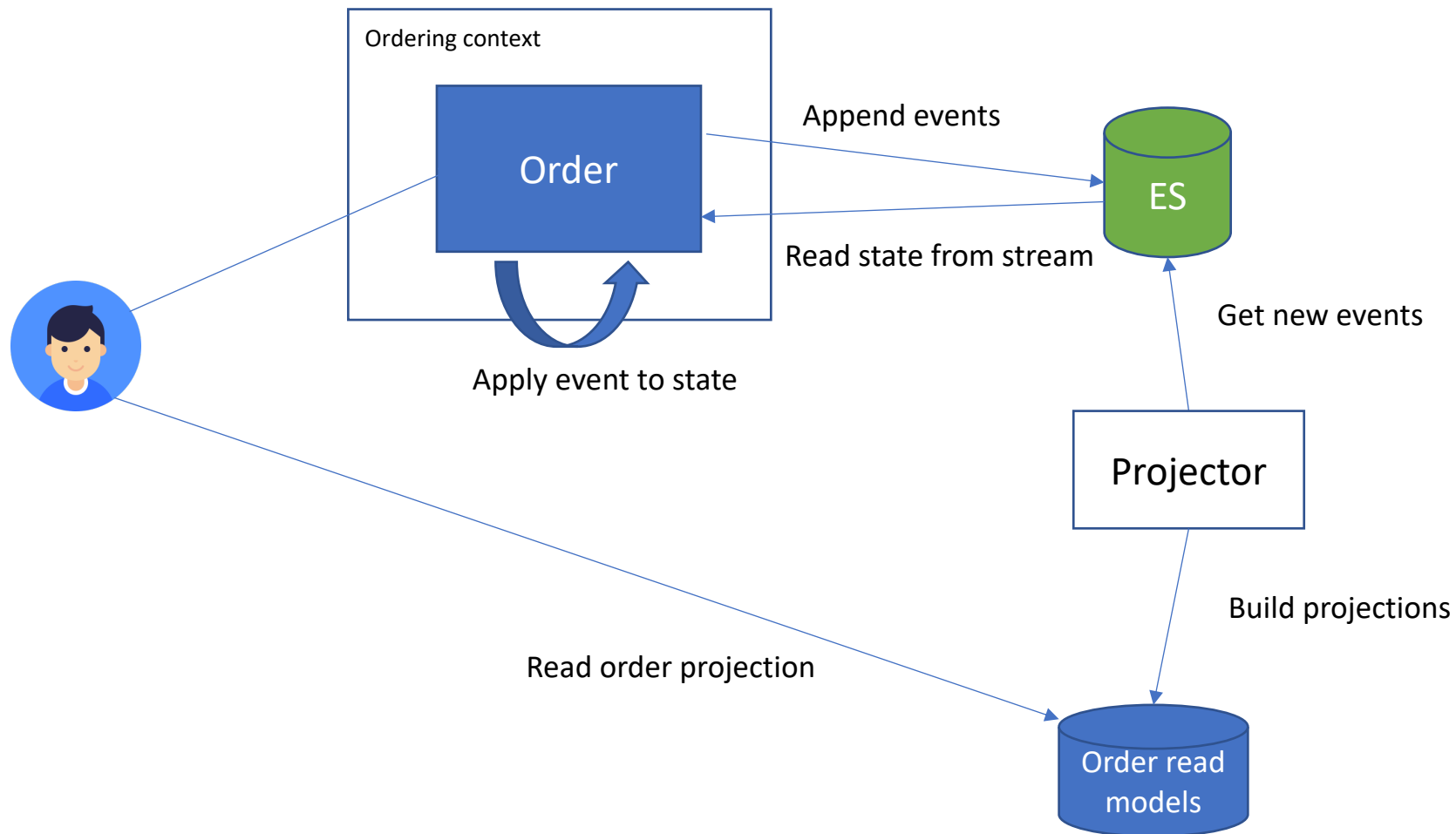


СНЭПШОТЫ

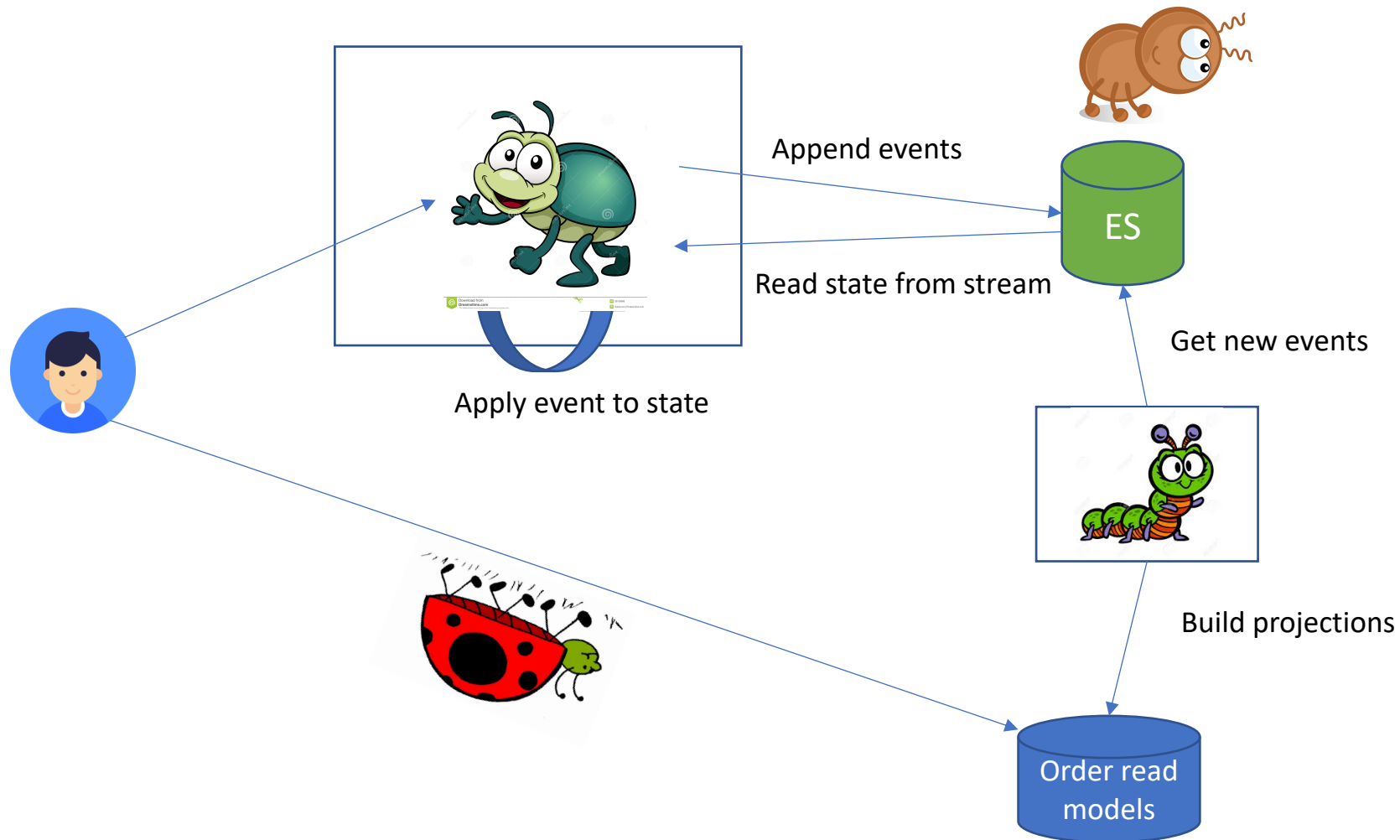


Не стоит делать ВСЮ систему
через Event sourcing

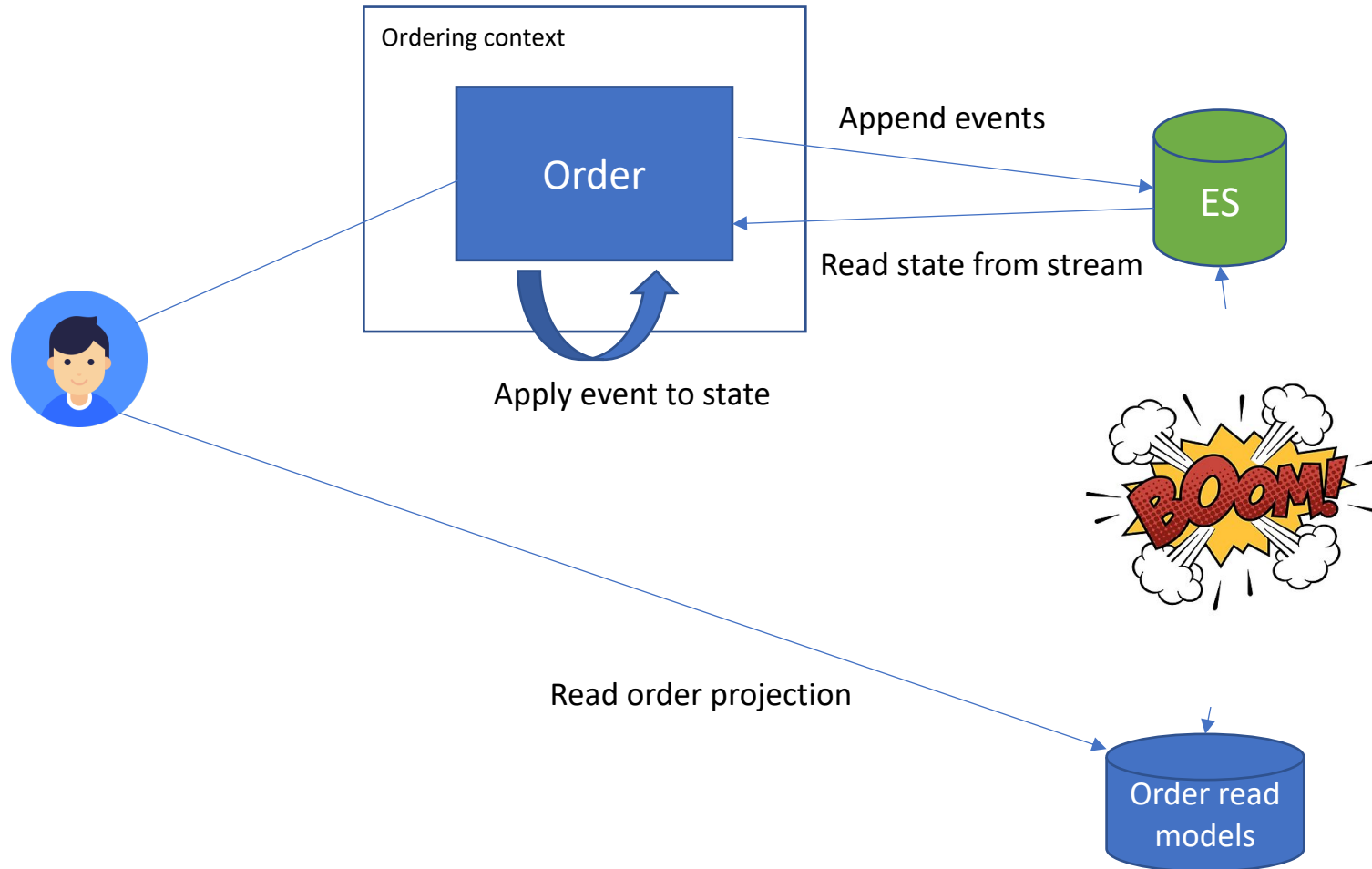
Выше порог входа



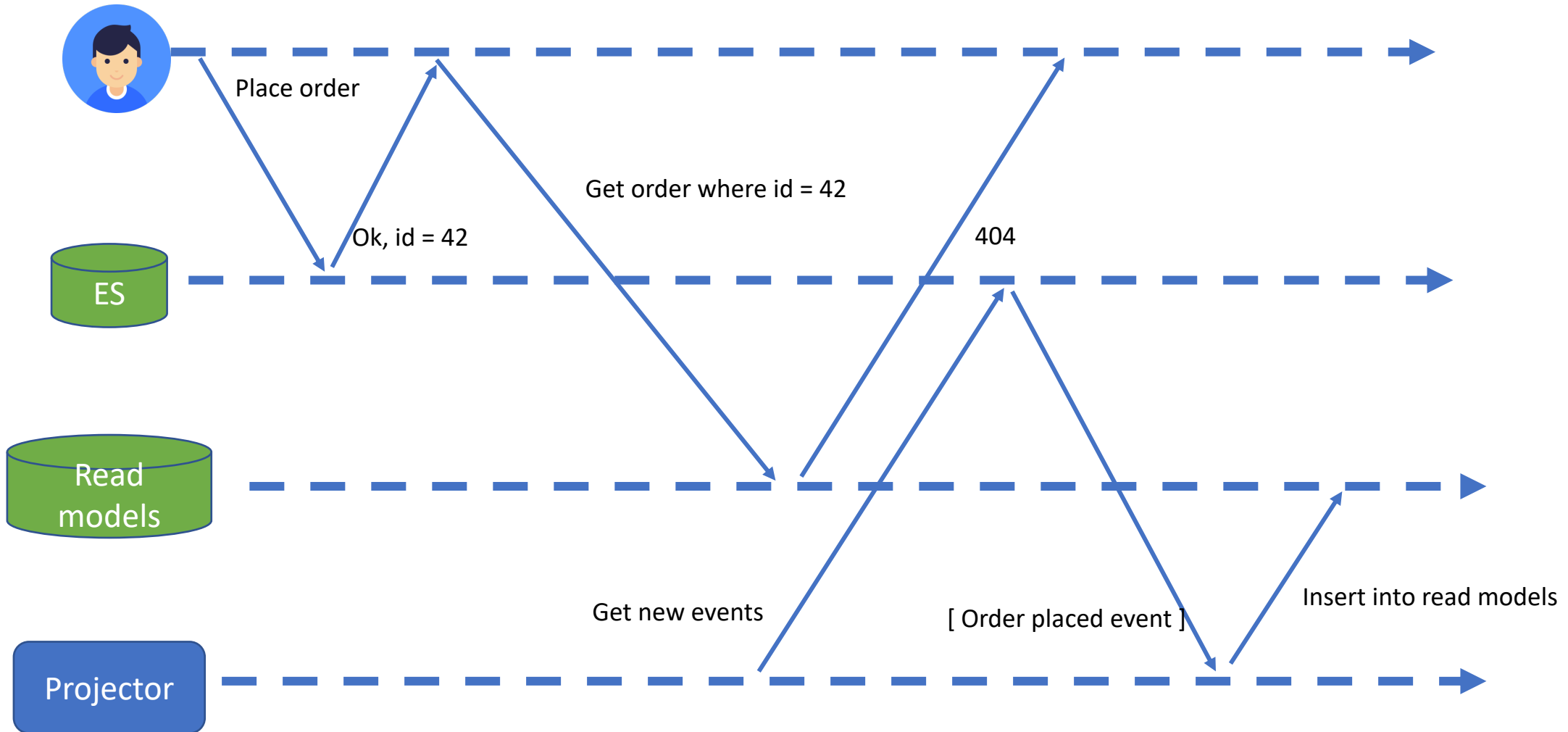
Выше порог входа



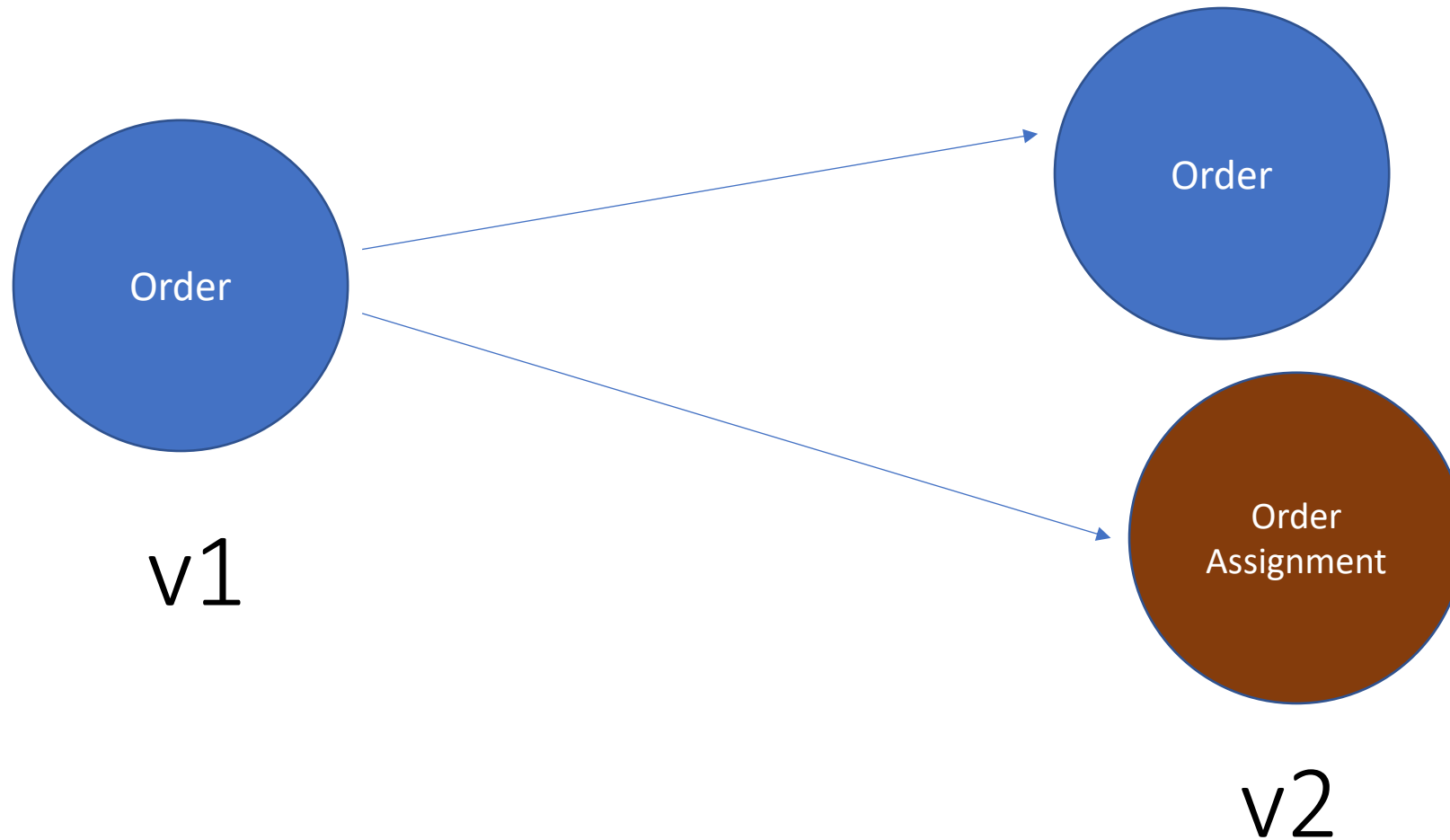
Больше причин для отказа



Eventual consistency



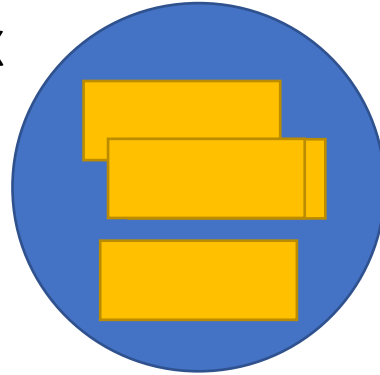
Требует более ответственно проектирования



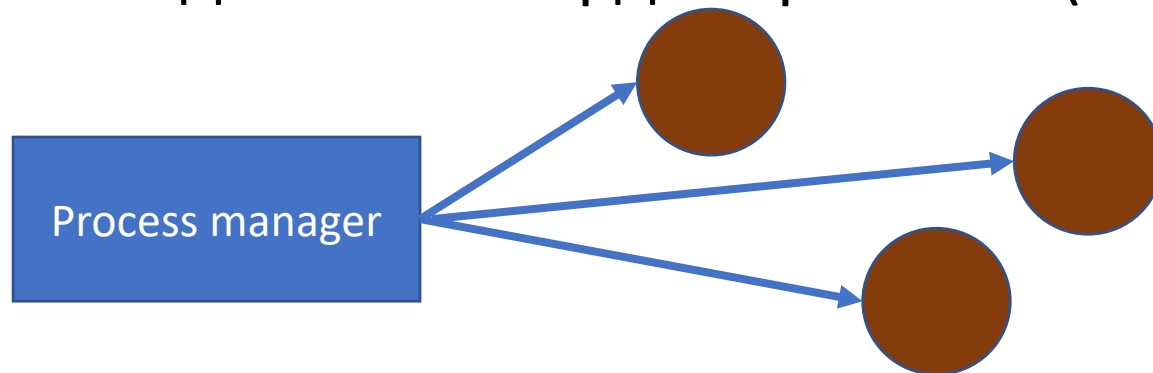
Требует более ответственного проектирования доменной модели

- Неоптимальные границы агрегатов

- Загрузка лишних данных



- Необходимость координирования (например, саги)



Требует более ответственно проектирования доменной модели

- Неоптимальные границы агрегатов
- Эти границы не так просто изменить

Цели

- ...
- **ВОЗМОЖНОСТЬ развивать приложение**
- распределенность

Хочу один заказ на нескольких
КЛИЕНТОВ



Несколько так несколько..

```
public class OrderPlaced
{
    public Guid BuyerOrganization { get; set; }
}
```

// превращается в ...

```
public class OrderPlaced
{
    public Guid[] Buyers { get; set; }
}
```

Версионирование

- Ошибки

Версионирование

- Ошибки
- Изменений требований

Версионирование

- Ошибки
- Изменений требований
- Refactoring toward deeper insight

Версионирование

- Ошибки
- Изменений требований
- Refactoring toward deeper insight
- Важно понимать природу изменения

Версионирование

- Ошибки
- Изменений требований
- Refactoring toward deeper insight
- Важно понимать природу изменения
- О стратегии версионирования лучше подумать заранее

Что есть сейчас

```
public class OrderPlaced
{
    public Guid BuyerOrganization { get; set; }
}

public Order(Guid id, Guid buyerOrganization)
{
    var event = new OrderPlaced
    {
        AggregateId = id,
        BuyerOrganization = buyerOrganization
    };

    RaiseEvent(event);
}
```

Поддержка разных версий

```
public class OrderPlaced
{
    public Guid BuyerOrganization { get; set; }
}
```

```
public class OrderPlaced_V2
{
    public Guid[] Buyers { get; set; }
}
```

Поддержка разных версий

```
public Order(Guid id, Guid[] buyers)
{
    var event = new OrderPlaced_V2
    {
        AggregateId = id,
        Buyers = buyers
    };

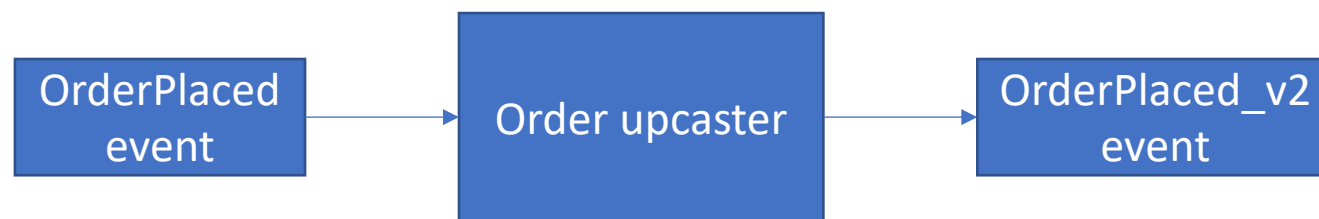
    RaiseEvent(event);
}
```

Поддержка разных версий

```
public void When(OrderPlaced event)
{
    Id = event.AggregateId;
    Buyers.Add(event.BuyerOrganization);
}
```

```
public void When(OrderPlaced_V2 event)
{
    Id = event.AggregateId;
    Buyers.AddRange(event.Buyers);
}
```

Апкаст



Апкаст

```
class OrderEventUpcaster
{
    object Upcast(OrderPlaced src)
    {
        return new OrderPlaced_V2
        {
            AggregateId = src.AggregateId,
            Buyers = new [] { src.BuyerOrganization }
        };
    }
}

public void When(OrderPlaced_V2 event)
{
    Id = event.AggregateId;
    Buyers.AddRange(event.Buyers);
}
```


Если апкаста не хватает

- Для более сложных сценариев есть более сложные подходы

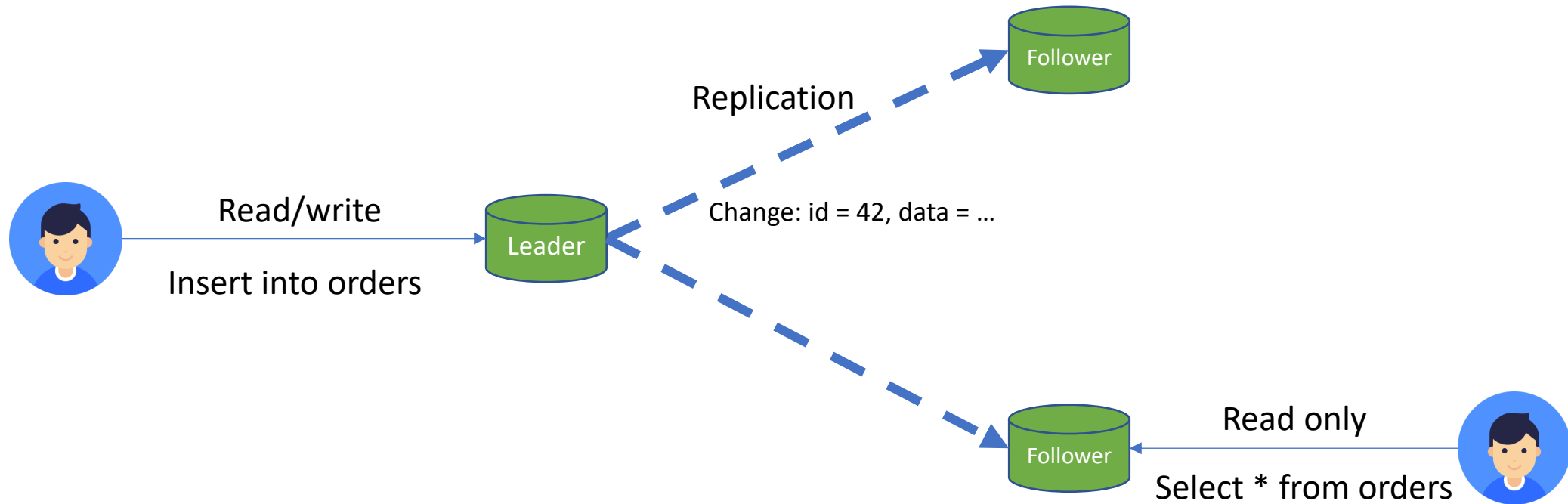
Если апкаста не хватает

- Для более сложных сценариев есть более сложные подходы
- Существующие события лучше не изменять

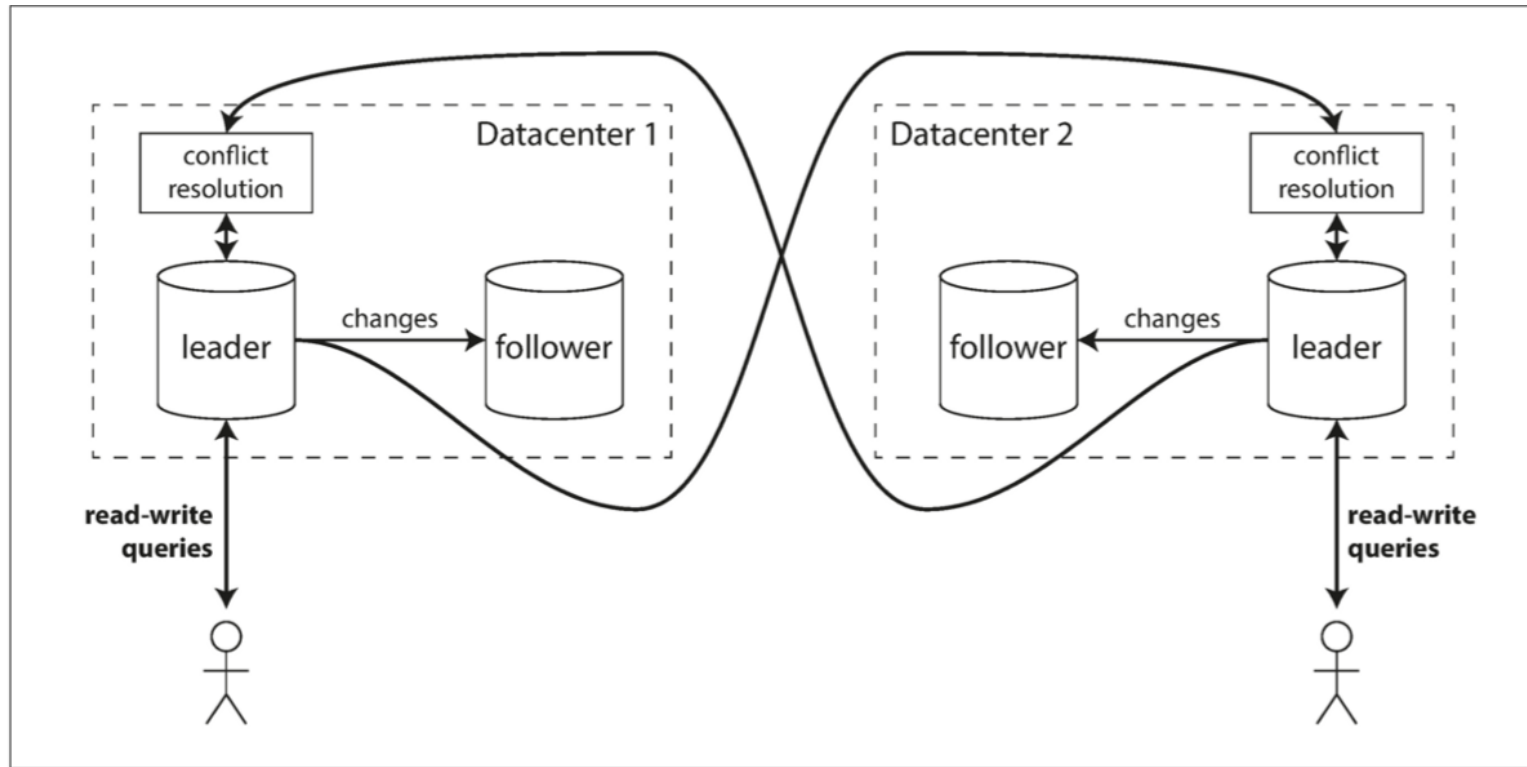
Цели

- ✓ нужны сложные выборки по данным
- ✓ запросов на чтение больше чем на запись
- ✓ история всех изменений, возможность откатить изменения
- ✓ возможность развивать приложение
- **распределенность**

Replication – single leader



Multileader replication



Подходы к репликации

- Statement based

Подходы к репликации

- Statement based
- WAL shipping

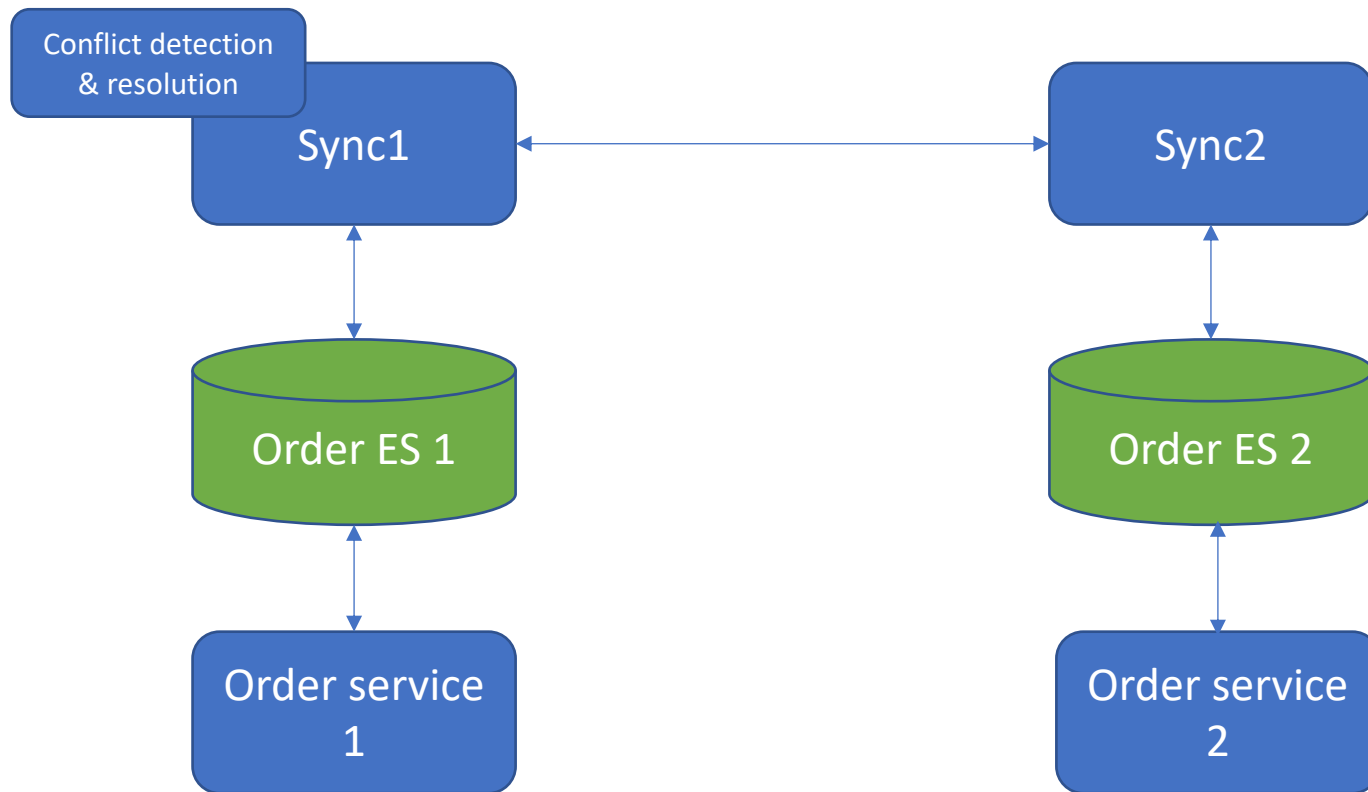
Подходы к репликации

- Statement based
- WAL shipping
- Logical replication

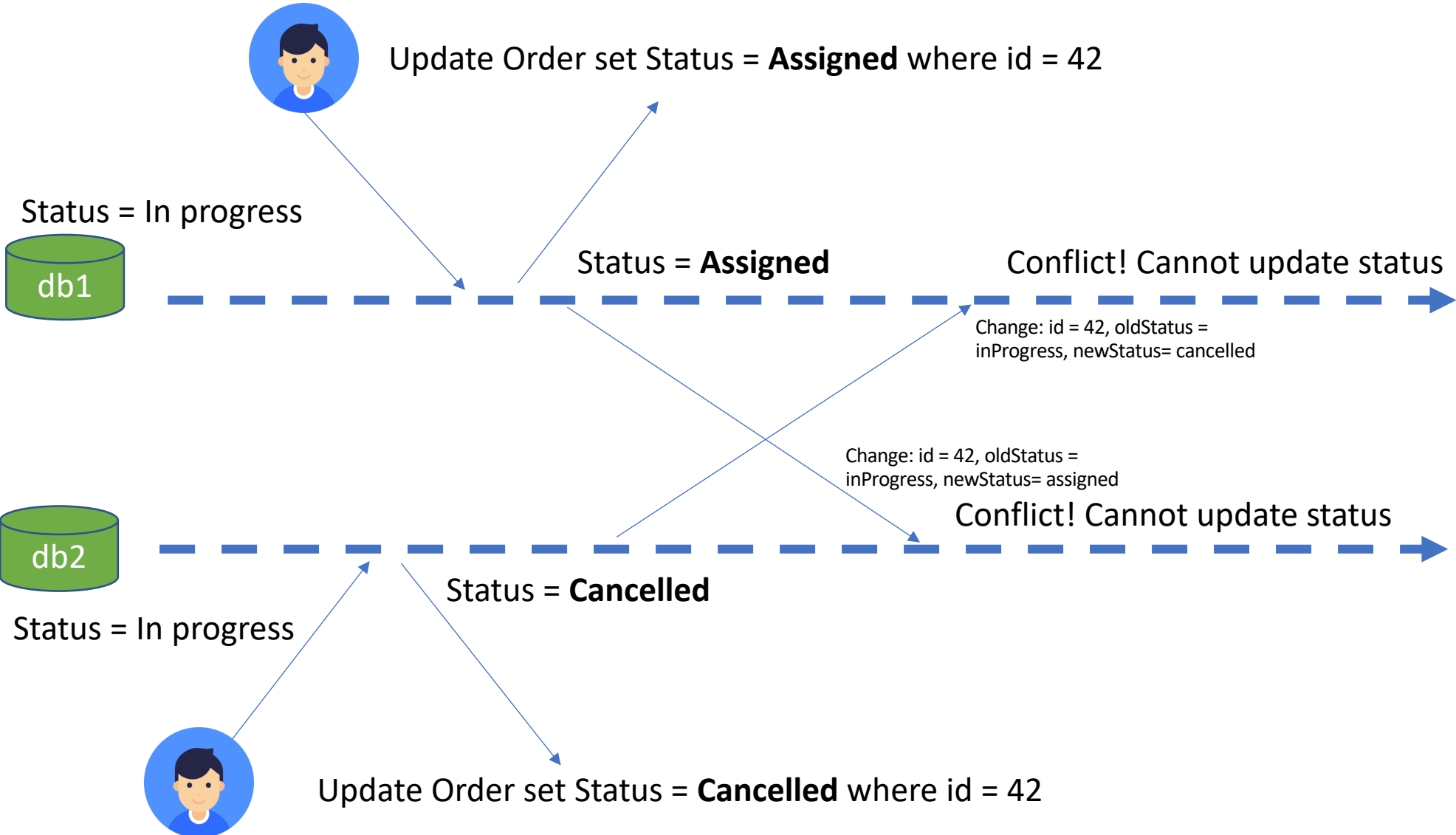
Подходы к репликации

- Statement based
- WAL shipping
- Logical replication
- Возможна репликации на уровне приложений.
В нашем случае – репликация event store

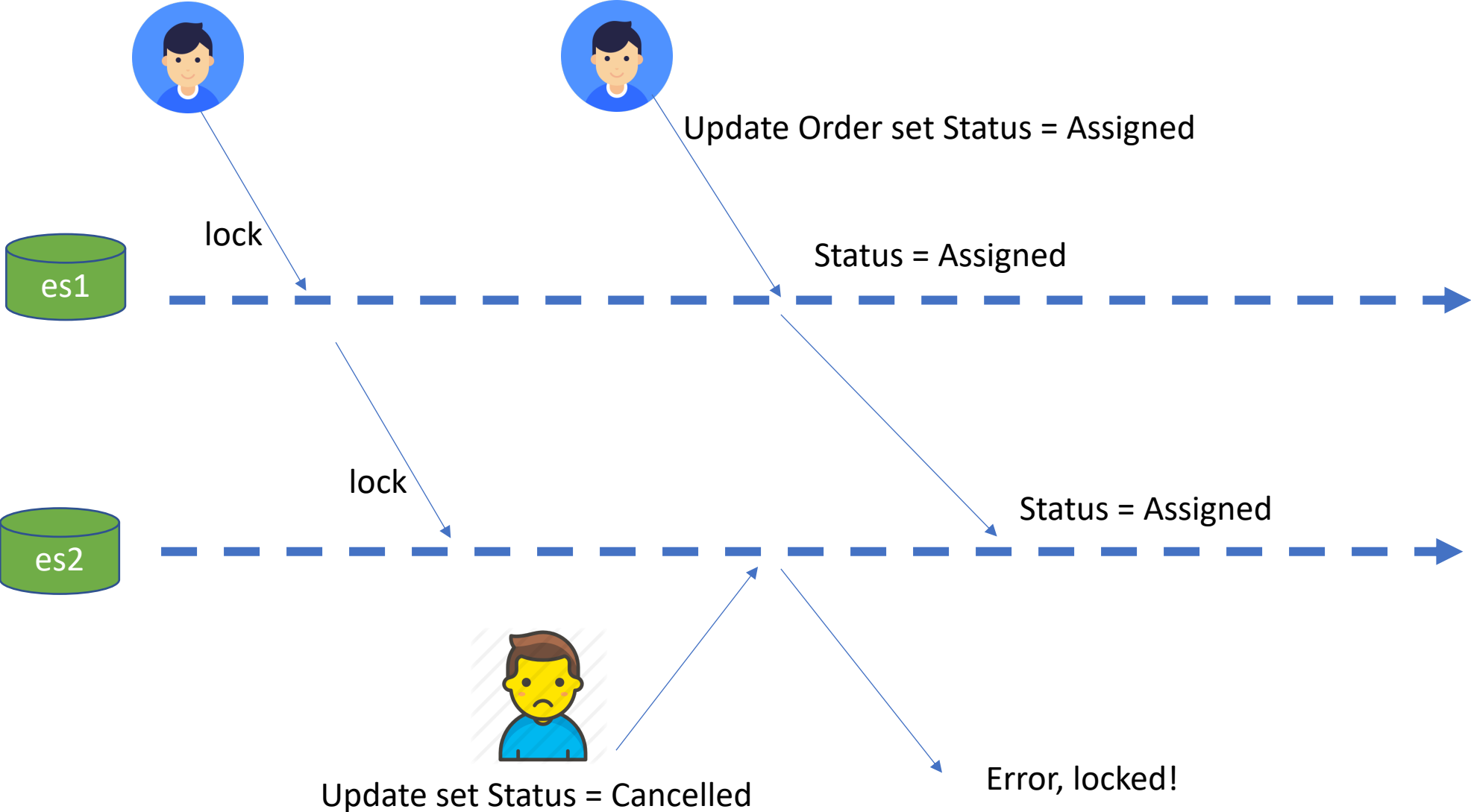
Event sourced replication



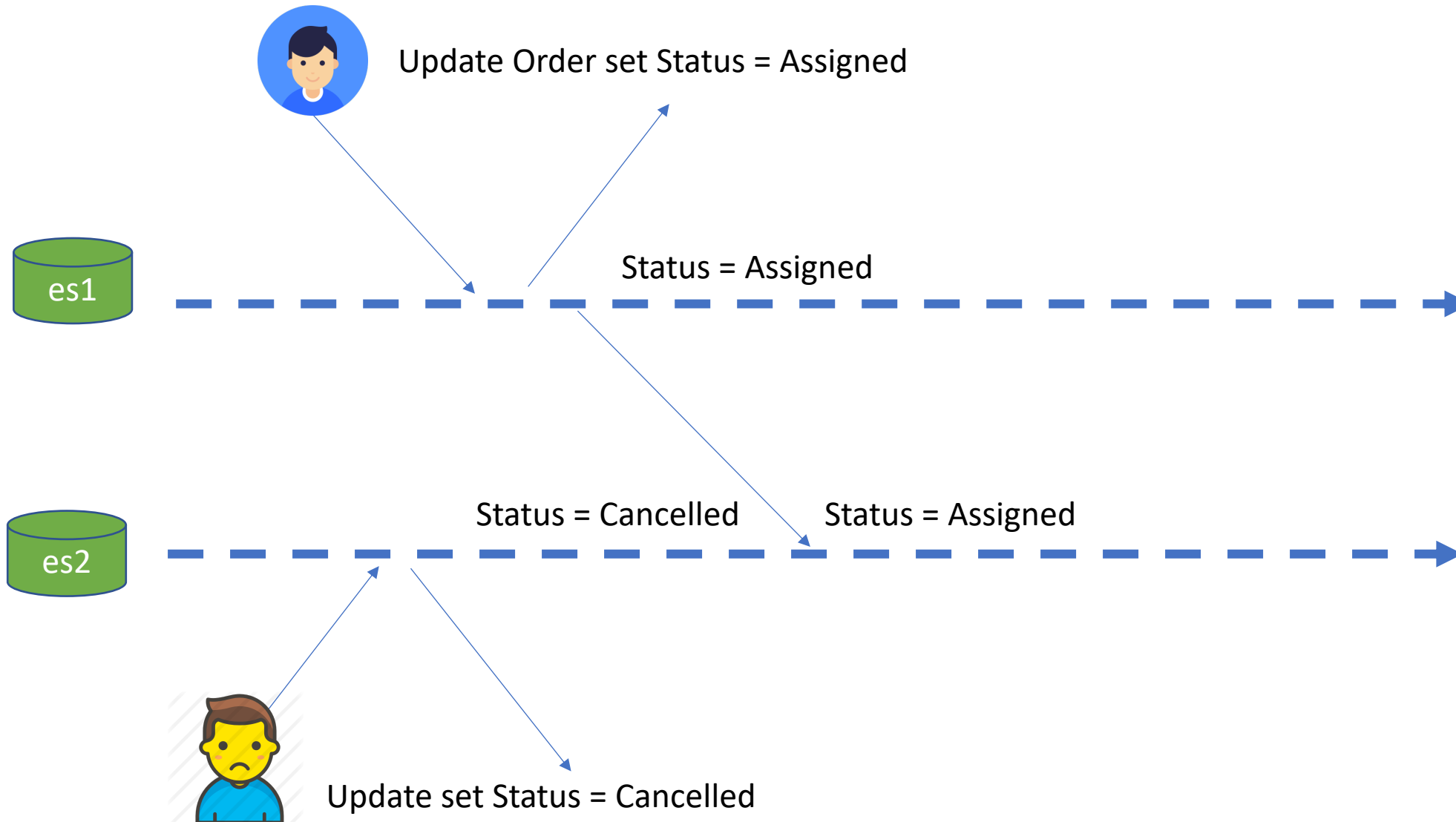
Replication conflict



Locks



LWW (last write wins)

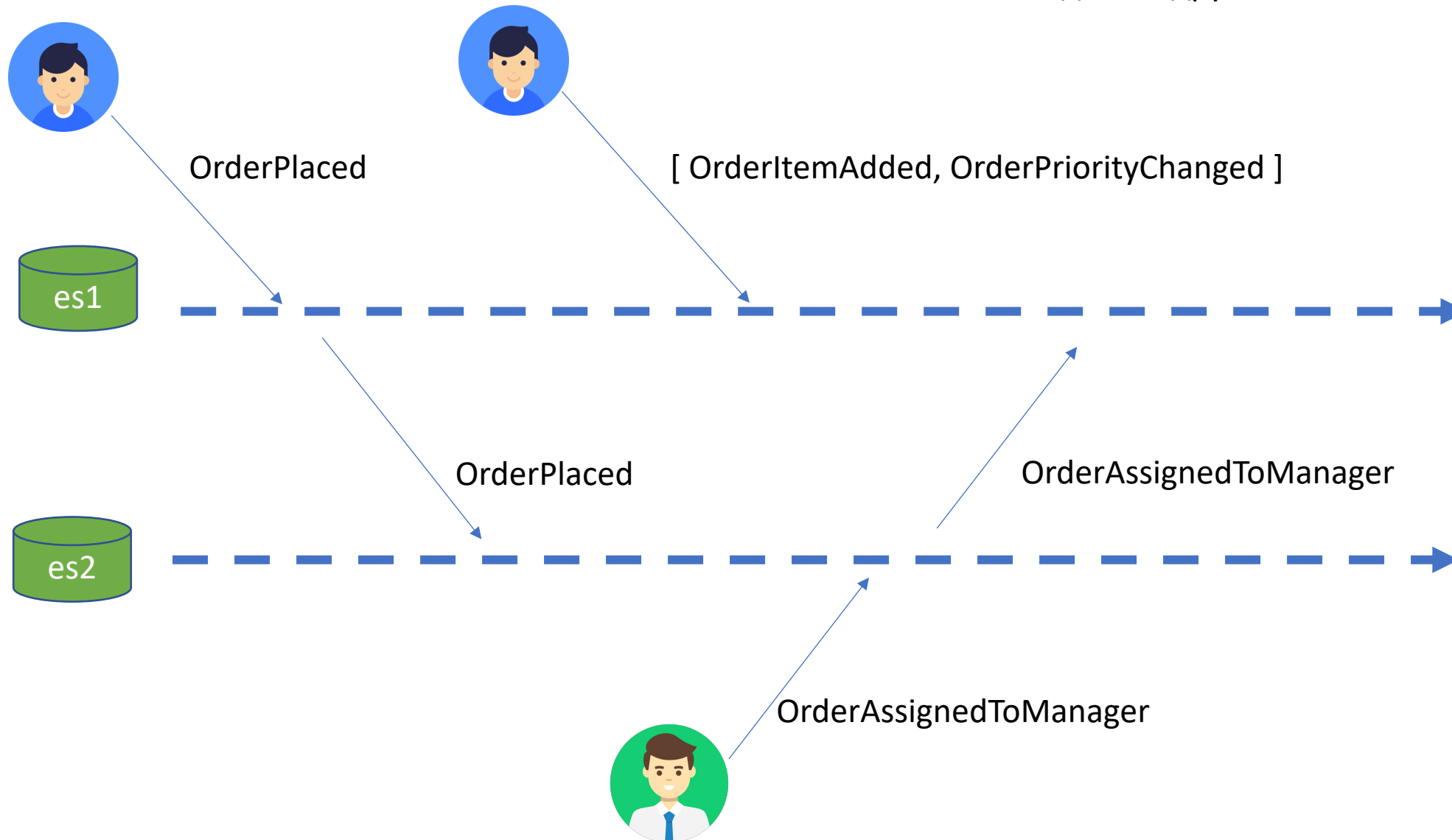


CRDTs

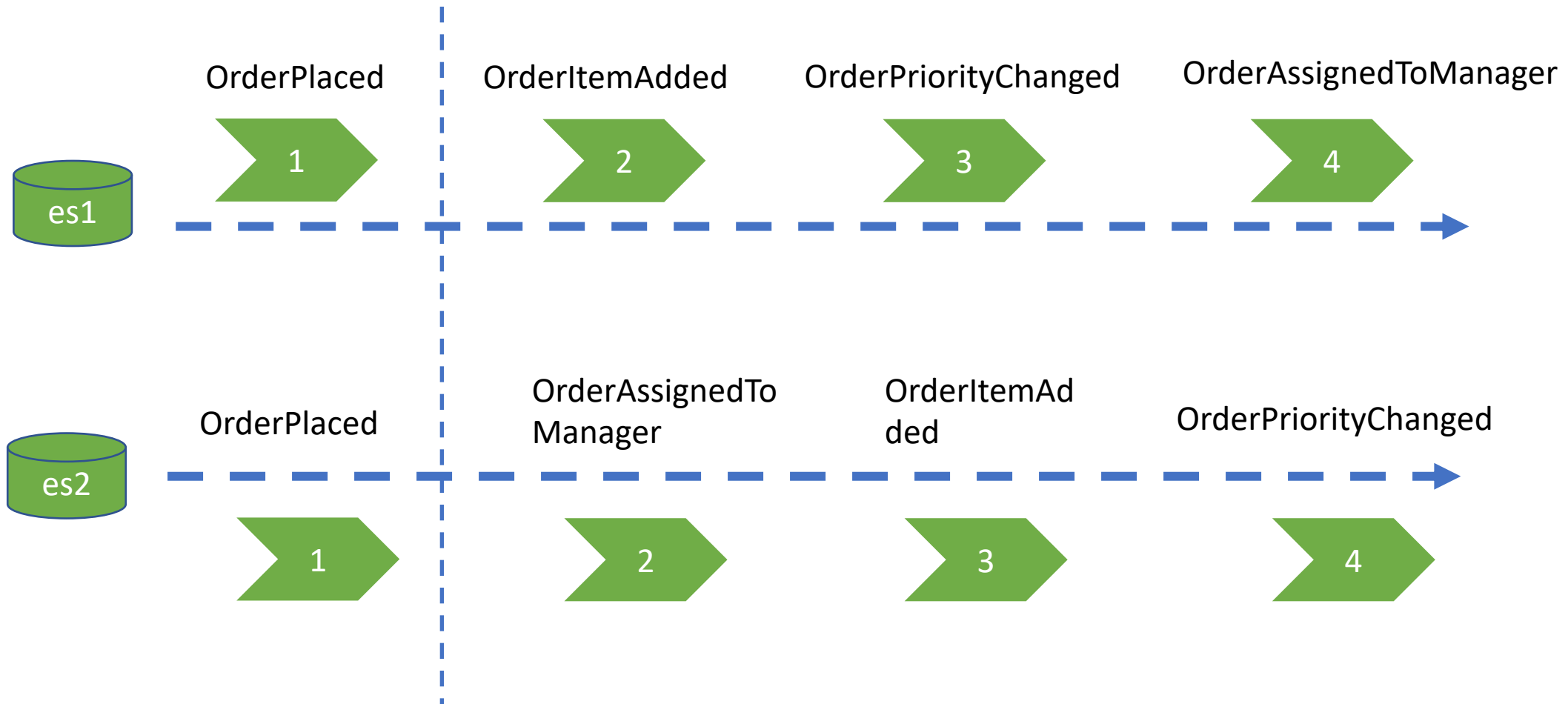
- State-based / operation-based
- Counter
- LWW-Register
- MV-Register
- G-Set
- 2P-Set
- JSON CRDT (<https://arxiv.org/pdf/1608.03960.pdf>)

Event sourced replication

Как понять что одно событие – следствие другого?



Sequence внутри каждого из ES - разные



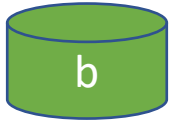
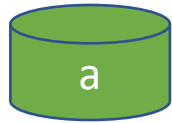
Vector clock (version vector)

- Colin Fidge and [Friedemann Mattern](#), 1988
- $v(e) = [(id1, clockValue1), (id2, clockValue2)]$
- Пример: $v = [(\text{“replica1”}, 1), (\text{“replica2”}, 3)]$

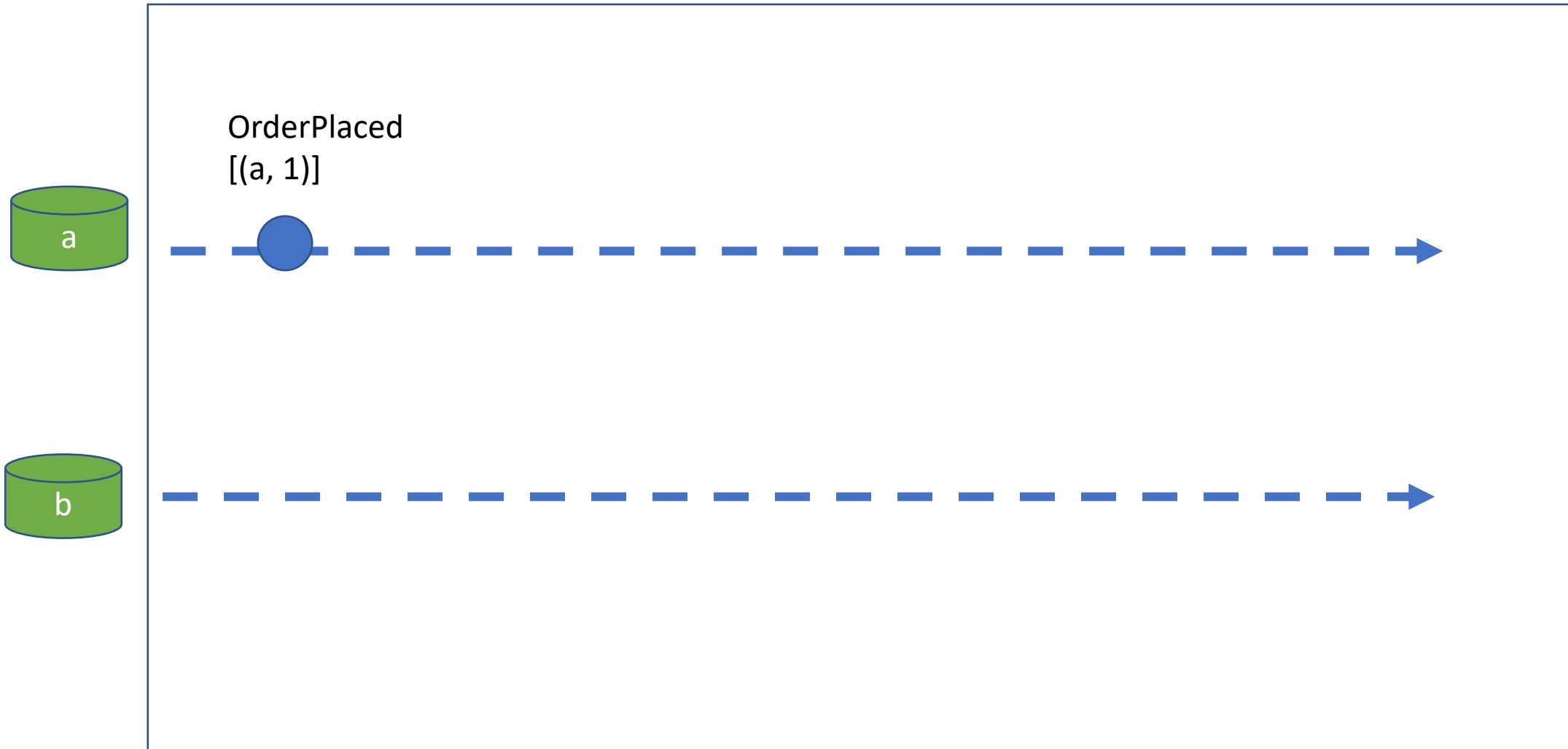
Vector clock (version vector)

- Colin Fidge and [Friedemann Mattern](#), 1988
- $v(e) = [(id1, clockValue1), (id2, clockValue2)]$
- Пример: $v = [(\text{“replica1”}, 1), (\text{“replica2”}, 3)]$
- $v1 < v2$, если для любого id $v1[id] \leq v2[id]$ и хотя бы один строго меньше
- если $v(e1) < v(e2)$, то $e1 \rightarrow e2$ (следует из)
- отношение частичного порядка (partial ordering)

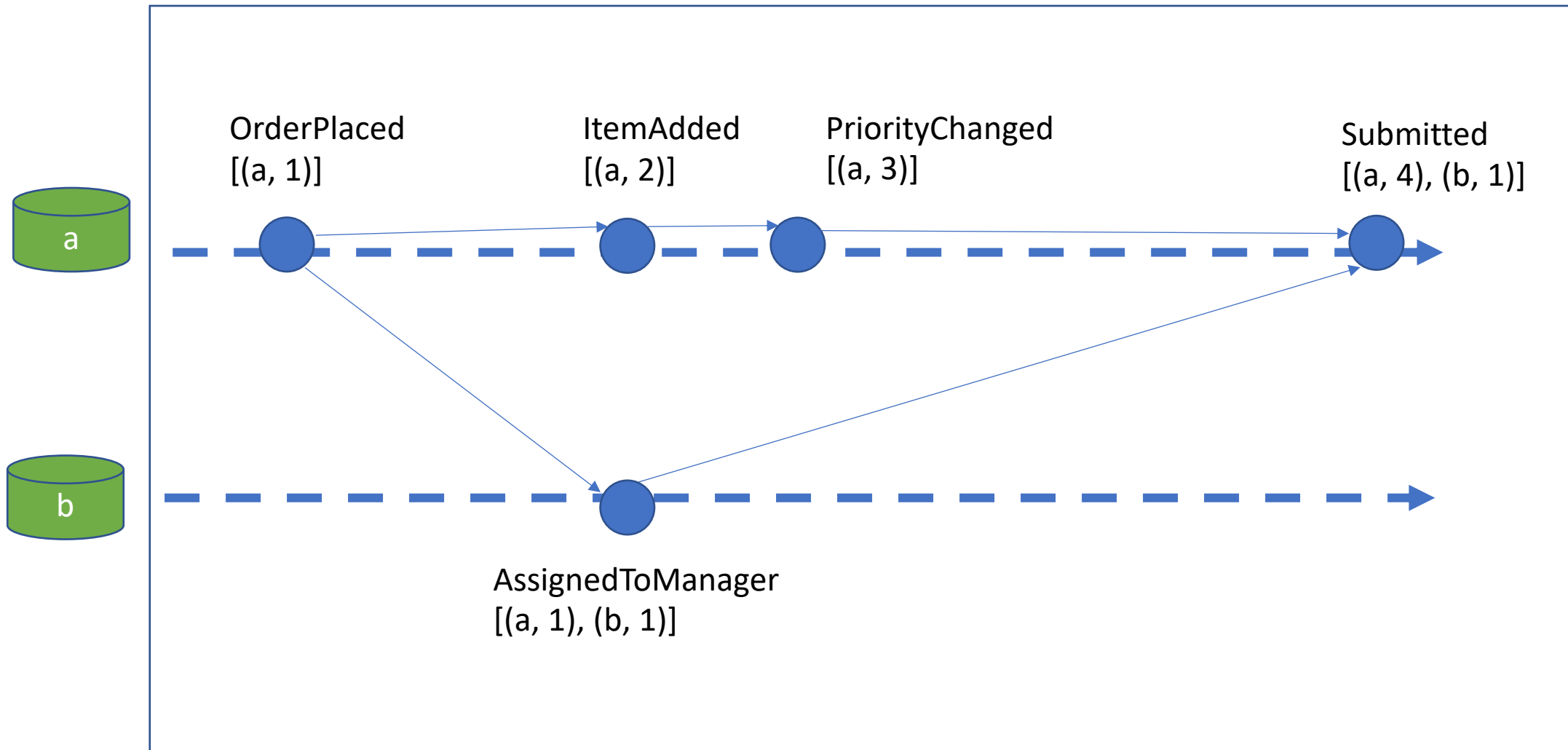
Vector clock (version vector)



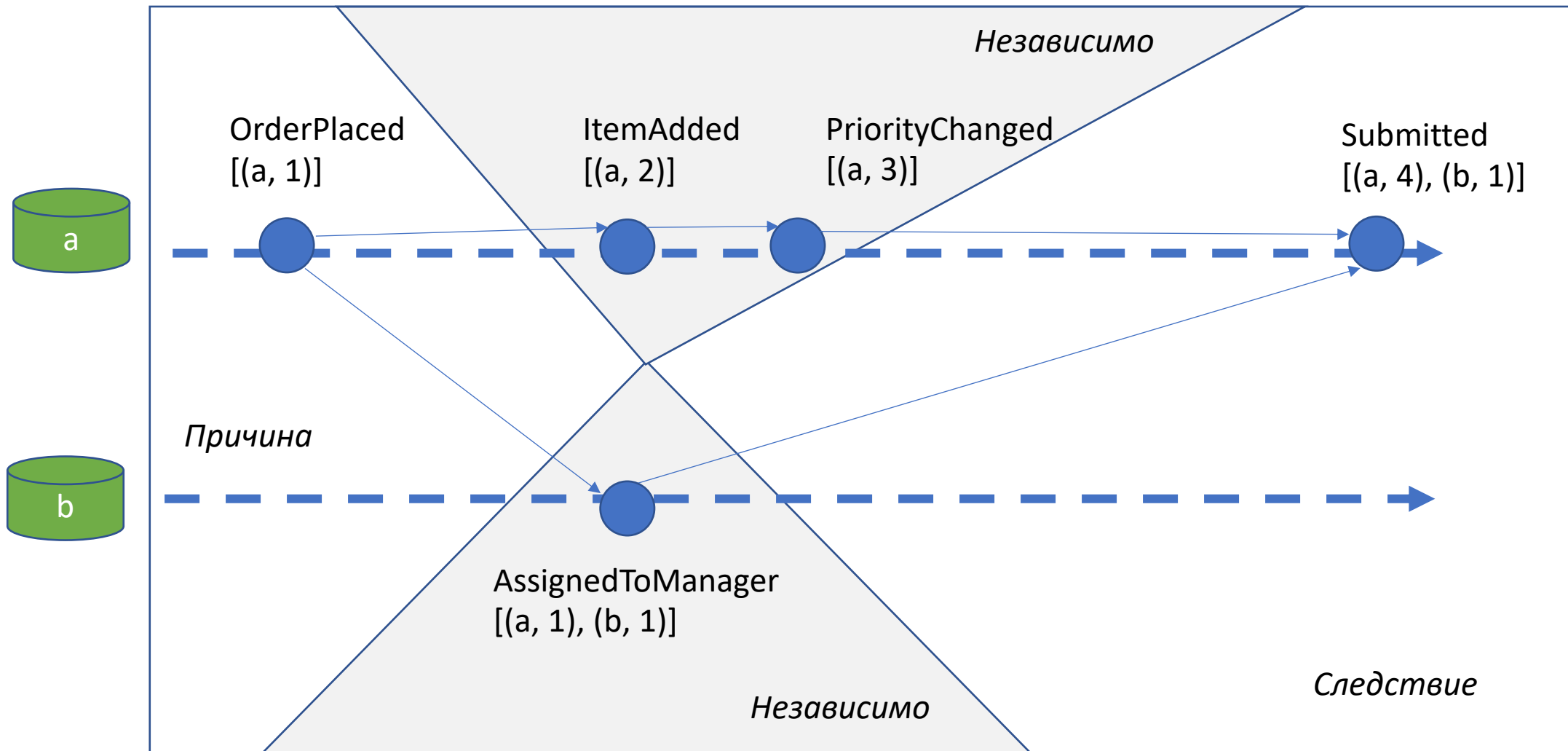
Vector clock (version vector)



Vector clock (version vector)



Vector clock (version vector)

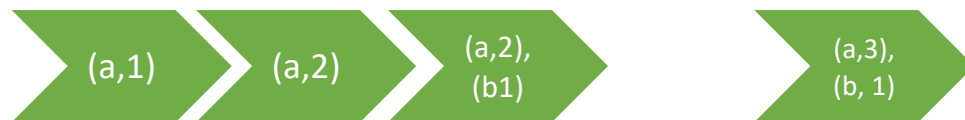


Генерация счетчиков

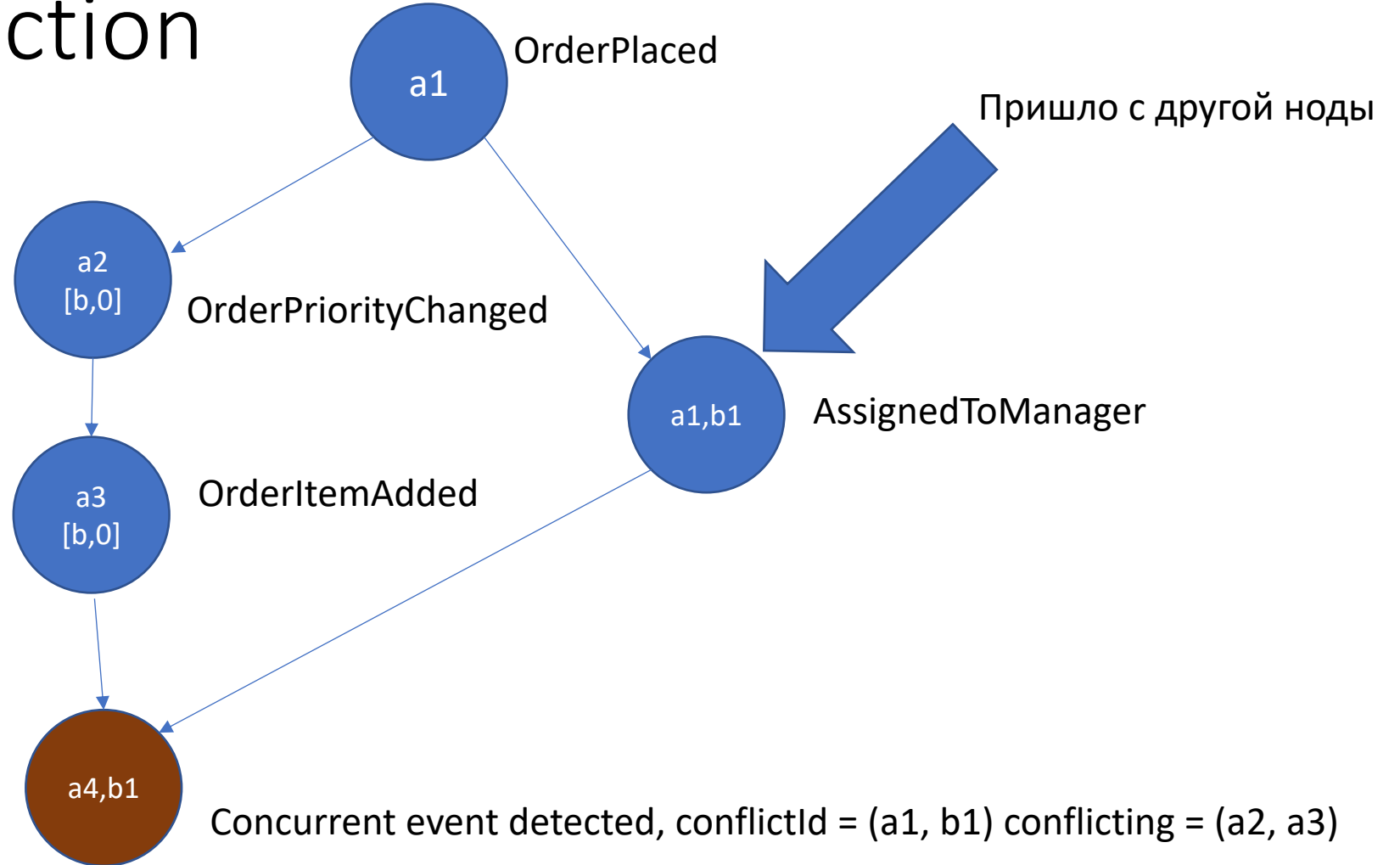
- Не используем Sequence в событии
- Векторные счетчики уникальны на всех нодах в рамках стрима
- В классическом описании у процесса есть свой внутренний регистр, но можно использовать ES

events

```
.Aggregate(new VectorClock(), (current, next) => current.Merge(next))  
.Increment(currentNodeId);
```

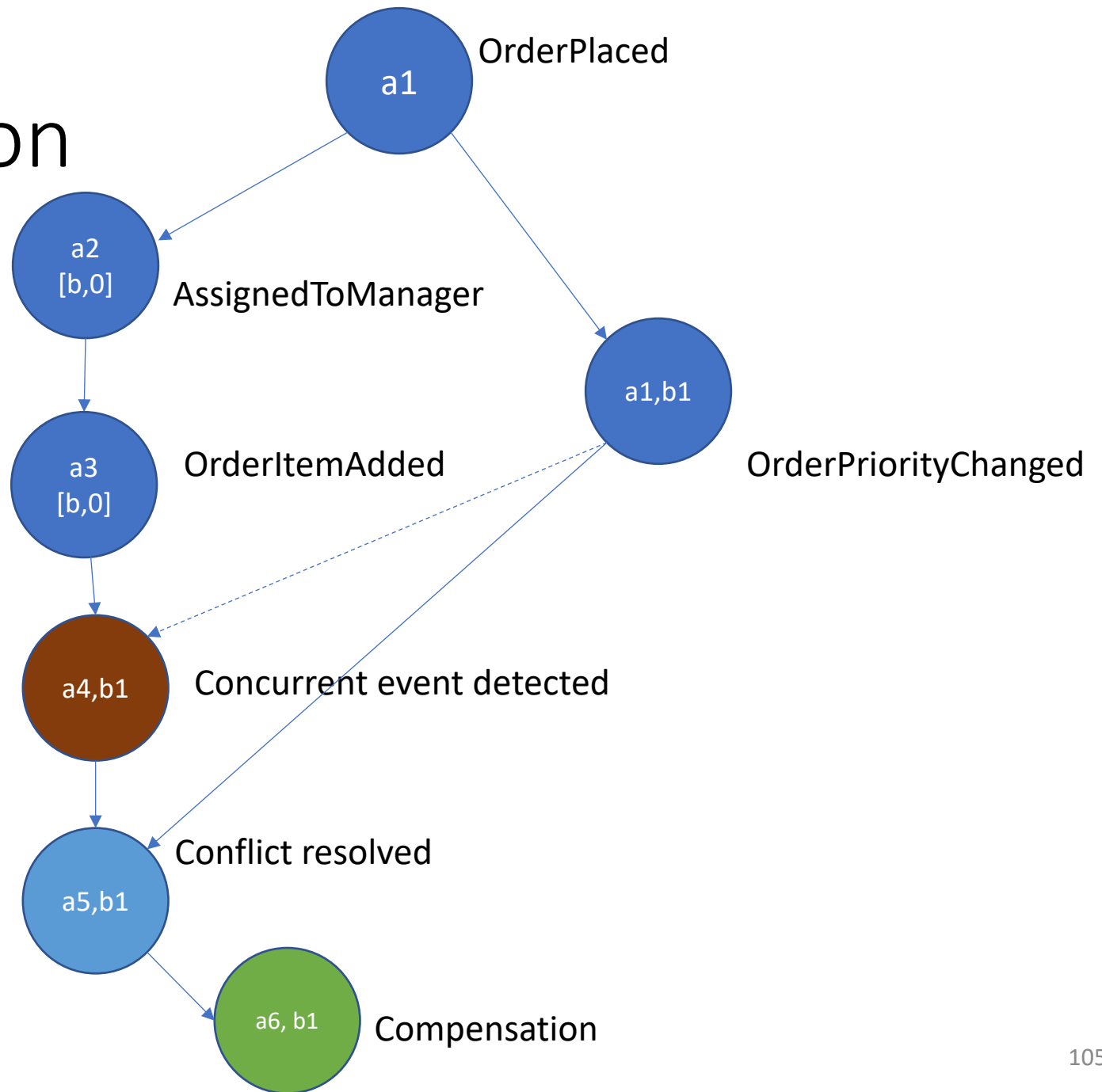


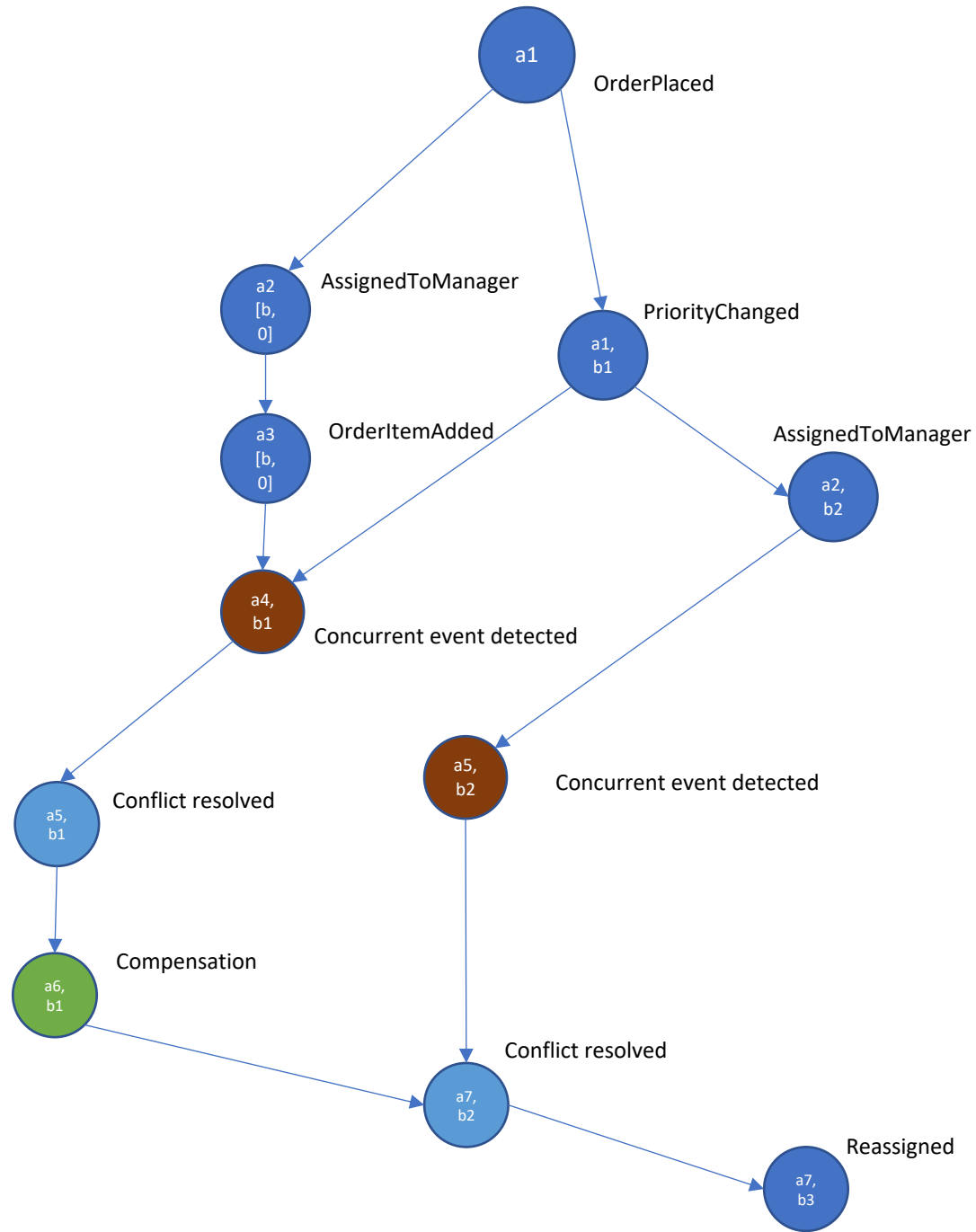
Conflict detection

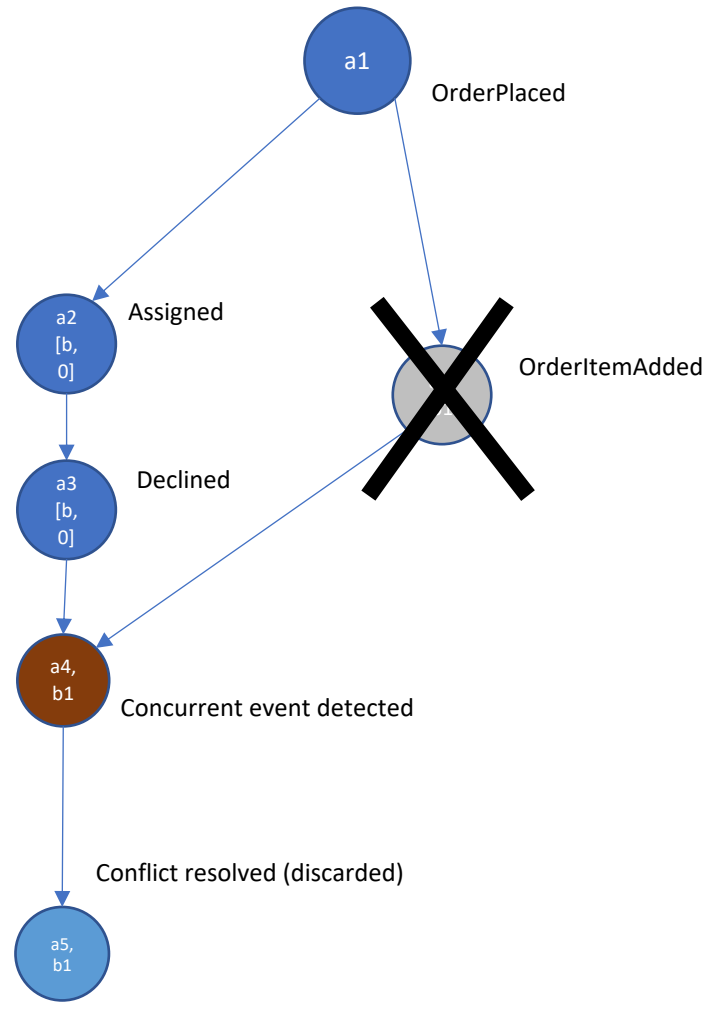


Conflict resolution

1. Sync vs async
2. Concurrent != conflicting
3. Types matter
4. Resolve (a4, b1) = Conflict resolved
5. Для агрегата граф мержится только при Resolve'e







Что делать с локальными изменения на время resolve'a?

- Накатывать на локальные рид модели все события
запрещать редактирование до разрешения конфликтов

Что делать с локальными изменения на время resolve'a?

- Накатывать на локальные рид модели все события
запрещать редактирование до разрешения конфликтов
- Использовать локальную версию
учитывать дописанные локальные события при разрешении

Что делать с локальными изменения на время resolve'a?

- Накатывать на локальные рид модели все события
запрещать редактирование до разрешения конфликтов
- Использовать локальную версию
учитывать дописанные локальные события при разрешении
- Interactive merge

```
Merge Revisions for D:\Projects\Git\Demo\Nancy\src\Nancy\DependencyContextAssemblyCatalog.cs
5 changes, 1 conflict

Local Changes (Read-only) | Result | Changes from Server (revision 9d4b6795...) | 
---|---|---|---
7 | 7 | using System.Reflection; | using System.Reflection;
8 | 8 | using Microsoft.Extensions.DependencyInjection; | using Microsoft.Extensions.DependencyInjection;
9 | 9 | | |
10 | 10 | /// <summary> | /// <summary>
11 | 11 | /// Default implementation | /// Default implementation
12 | 12 | /// retrieving <see cref="IDependencyContext" /> | /// retrieving <see cref="IDependencyContext" />
13 | 13 | /// </summary> | /// </summary>
14 | 14 | public class DependencyContext | public class DependencyContext
15 | 15 | { | {
16 | 16 |     private static readonly IDependencyContext _instance = new DependencyContext(); |     private static readonly IDependencyContext _instance = new DependencyContext();
17 | 17 |     private readonly IDependencyContext _instance; |     private readonly IDependencyContext _instance;
18 | 18 | | |
19 | 19 | | |
20 | 20 | /// <summary> | /// <summary>
21 | 21 | /// Initializes a new instance of the class. | /// Initializes a new instance of the class.
22 | 22 | | |
23 | 23 | public DependencyContext() | public DependencyContext()
24 | 24 |     : this(Assembly.GetExecutingAssembly()) |     : this(Assembly.GetExecutingAssembly())
25 | 25 | { | {
26 | 26 | } | }
27 | 27 | | |
28 | 28 | | |
29 | 29 | | |
30 | 30 | | |
31 | 31 | | |
32 | 32 | | |
33 | 33 | | |
34 | 34 | | |
35 | 35 | | |
```

Минусы такого подхода

- Дополнительные метаданные к каждому событию

Минусы такого подхода

- Дополнительные метаданные к каждому событию
- Построение деревьев каждый раз

Минусы такого подхода

- Дополнительные метаданные к каждому событию
- Построение деревьев каждый раз
- Как делать снимки?

Минусы такого подхода

- Дополнительные метаданные к каждому событию
- Построение деревьев каждый раз
- Как делать снэпшоты?
- Все это в рамках только одного агрегата

Цели

- ✓ нужны сложные выборки по данным
- ✓ запросов на чтение больше чем на запись
- ✓ история всех изменений, возможность откатить изменения
- ✓ возможность развивать приложение
- ✓ географическая распределенность

Когда применять ES?

- предметная область **естественно** описывается в виде событий

Когда применять ES?

- предметная область **естественно** описывается в виде событий
- надо строить и перестраивать представления данных в зависимости от изменяющихся требований

Когда применять ES?

- предметная область **естественно** описывается в виде событий
- надо строить и перестраивать представления данных в зависимости от изменяющихся требований
- нужно хранить всю историю изменений
- могут быть другие причины

Когда НЕ применять ES?

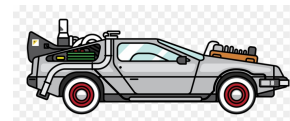
- слишком сложно
 - на этапе проектирования
 - разработки
 - поддержки
- делаем прототип
- все события - СущностьОбновлена
- оно все таки тормозит

Подводя итоги

- Don't drink too much kool aid
- Вносит дополнительную сложность
- Требуется больших усилий в плане проектирования и разработки
- Но..

Подводя итоги

- Don't drink too much kool aid
- Вносит дополнительную сложность
- Требуется больших усилий в плане проектирования и разработки
- Но..
- Мы думаем на уровне домена
- Не теряем информацию
- Можем путешествовать в прошлое



Что посмотреть/почитать

- Event Sourcing Greg Young <https://www.youtube.com/watch?v=8JKjvY4etTY>
- [Pragmatic Event-Driven Microservices Allard Buijze](https://www.youtube.com/watch?v=vSd_0zGxslU&t=2s)
https://www.youtube.com/watch?v=vSd_0zGxslU&t=2s
- <https://ddd-cqrs-es.slack.com>
- Hands-On Domain-Driven Design with .NET Core: Tackling complexity in the heart of software by putting DDD principles into practice
- Versioning in an Event Sourced System (<https://leanpub.com/esversioning>)
- The Dark Side of Event Sourcing: Managing Data Conversion
- Designing Data-Intensive Applications (Kleppmann)
- <https://github.com/RBMHTechnology/eventuate>

Всем спасибо ;)

https://t.me/jacob_povar

<https://github.com/jacobpovar>