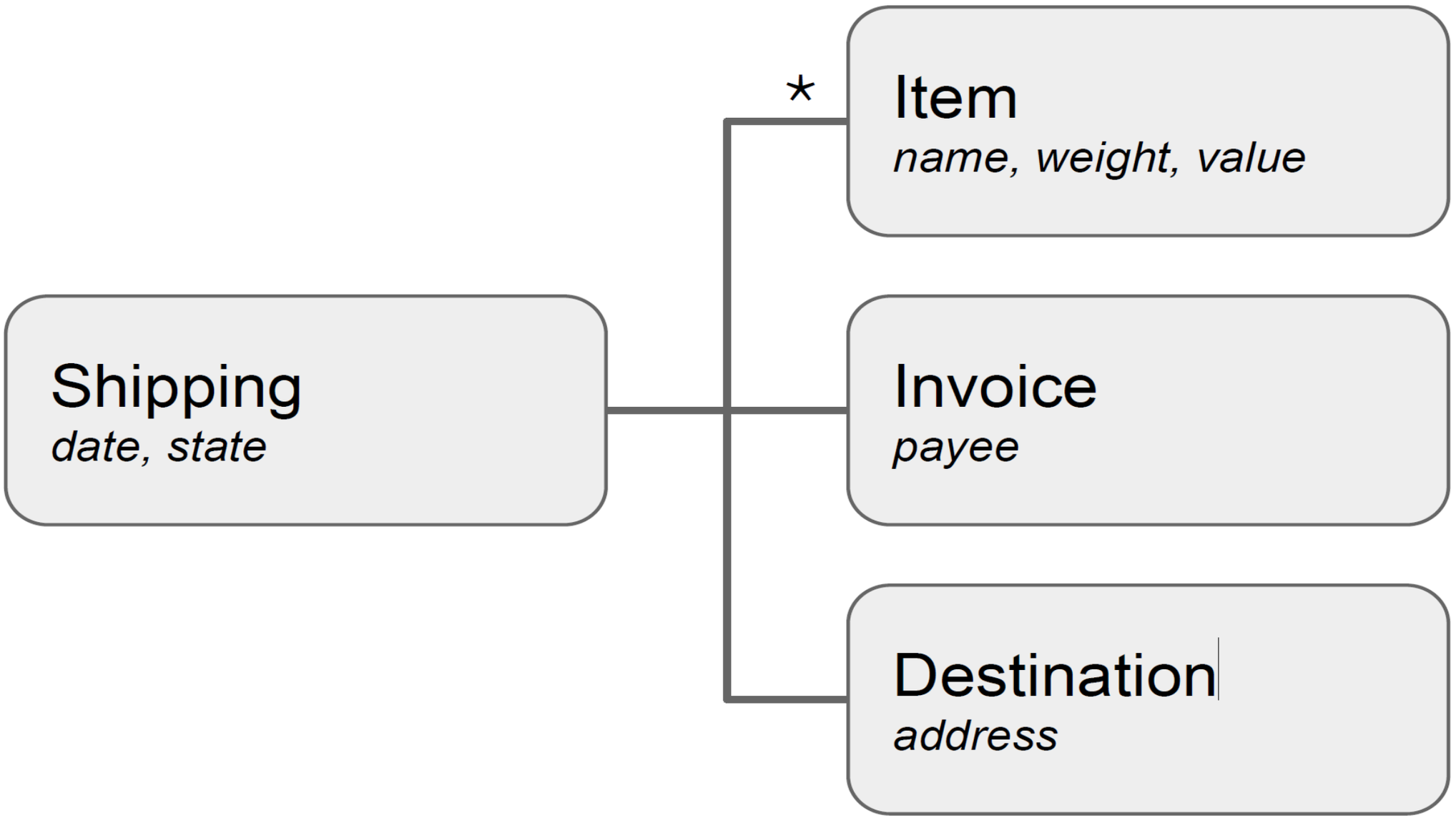
Event Sourcing

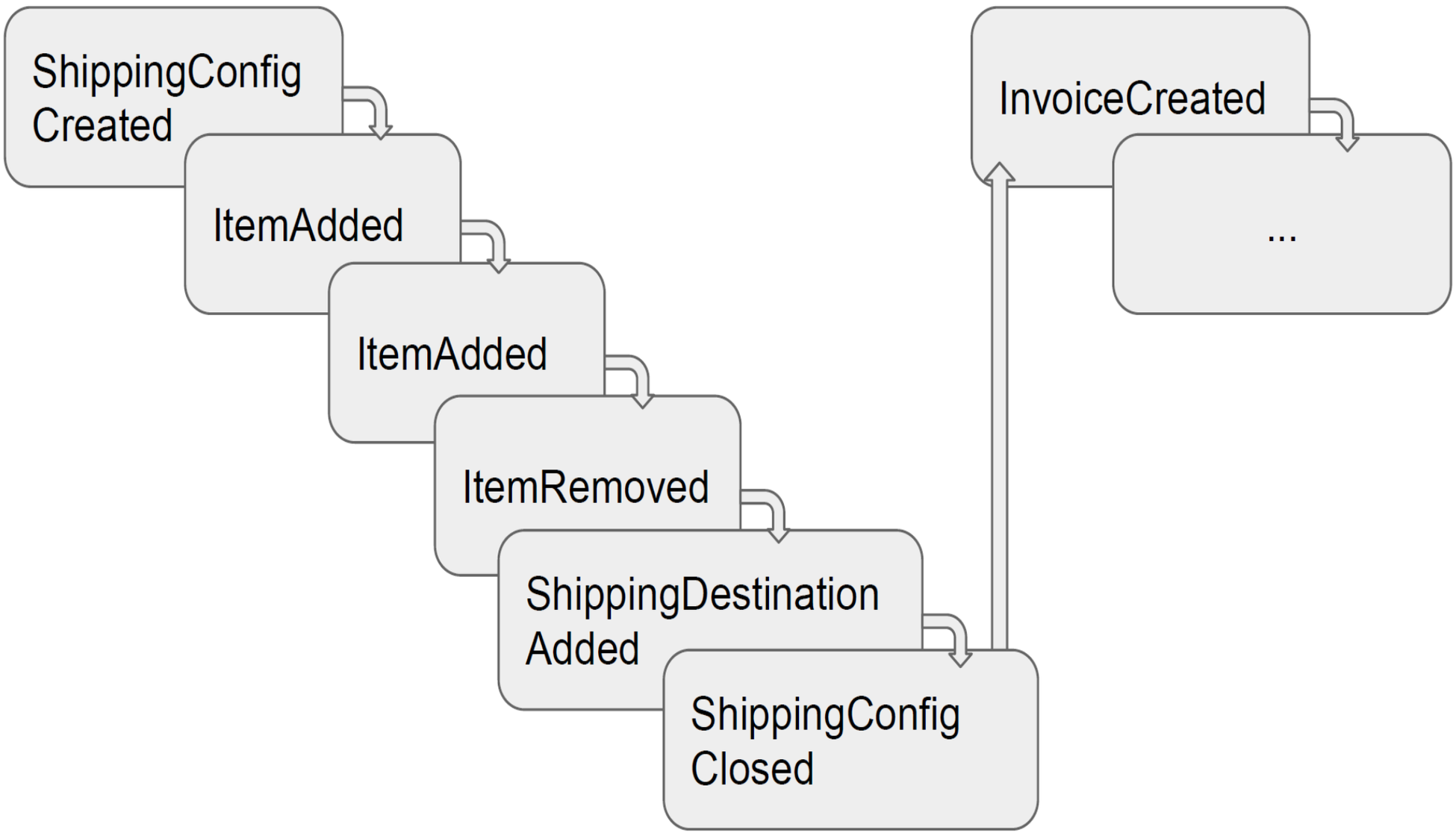
Robert Rudko

Prezentacja na Technologie Programistyczne

Domena / klasyczny model



Model zdarzeniowy systemu



Ale po co?

```
ShoppingCart {
    Book1, Book2, Book3
}
```

=

```
ShoppingCart {
    Book1, Book2, Book3
}
```

```
BankAccount {
    Owner: Janusz
    Balance: 1000.00
}
```

=

```
BankAccount {
    Owner: Janusz
    Balance: 1000.00
}
```

Niekoniecznie...

- ShoppingCart {
  +Book1
  +Book2
  +Book3
  }

ShoppingCart {
+Book1
+Book5
+Book3
+Book2
+Book6
-Book6
-Book5
}

Ale po co?

- Naturalniejsza reprezentacja biznesowych procesów (zdarzenia vs stan)
- Nawiązanie do DDD / Event Stormingu
  Brak translacji między programistami a biznesem
- W ogólności drogie/ciężkie, tylko do konkretnych celów!

I co to daje?

- Analityka – czemu ktoś usunął książki?
- Stany pośrednie – nie tylko finalny
- Historia, audyt – z czego wynika taki stan konta?
- Immutable log → Dyski write once read many
- Rebuilding stanu aplikacji
- Temporal query – z dowolnego momentu!
- Wszystkie zmiany w obiektach domenowych poprzez eventy
- Machine learning

Intuicja

- Jakie popularne systemy opierają się na podobnych zasadach?

  → Różne VCSy (Version Control System)

- Co działa podobnie?
  → Księgowość (zawsze długopis, nigdy ołówek)

Modelowanie

- **Command →**

  Request for change to domain, not a statement of fact, may be refused eg. ConfirmOrder

- **Event →**

  Represents something that took place in the domain. Can be considered a statement of fact and used to take decisions in other parts of the system eg. OrderConfirmed (Differs from Command mostly by intent)

- **Invariants →**

  An invariant describes something that must be true within your design, at all times (Constraints/ Czy żądanie komendy może się wydarzyć)

Agregat

A DDD aggregate is a cluster of domain objects that can be treated as a single unit.

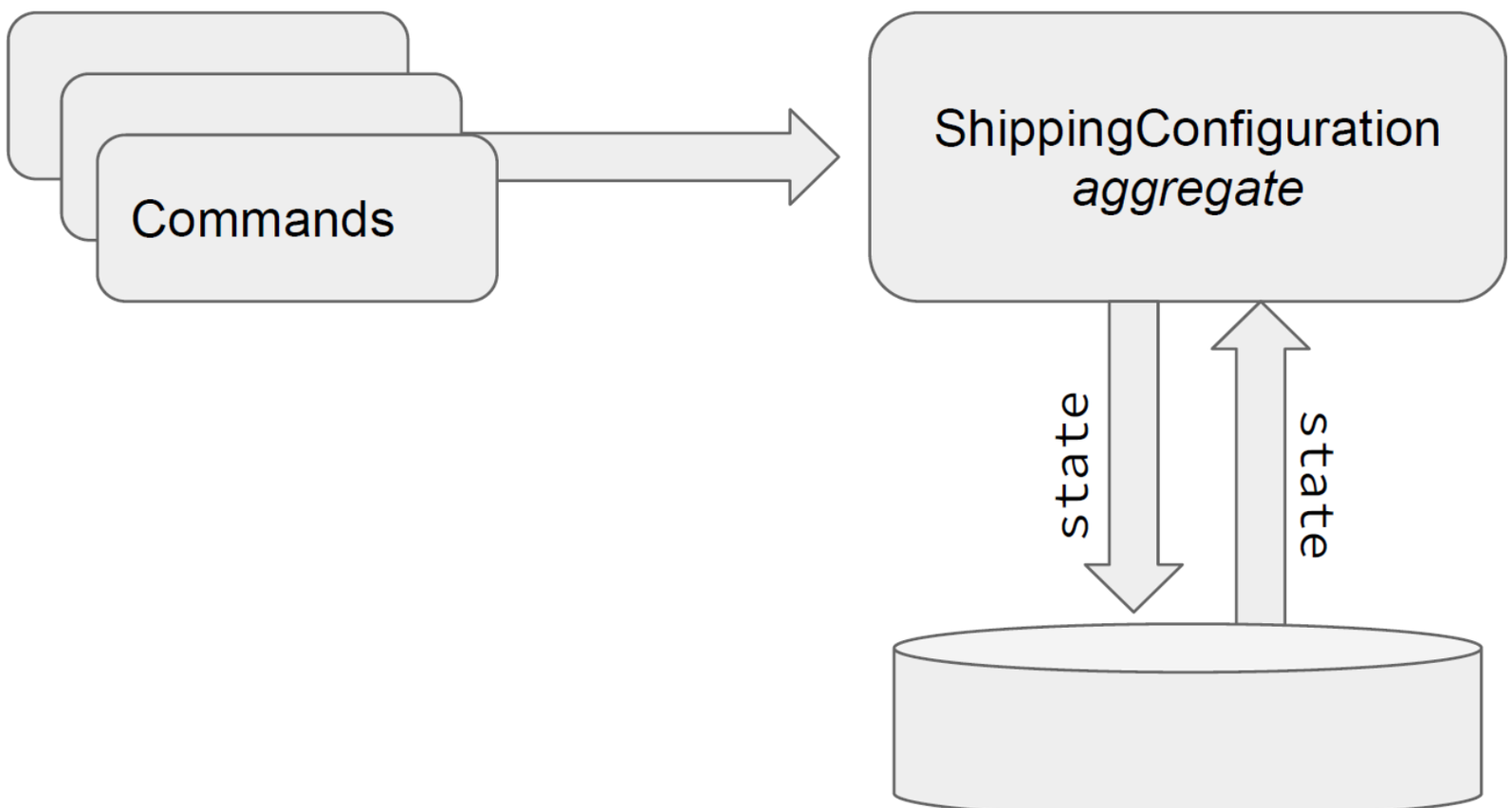
Within an Aggregate there is an Aggregate Root. The Aggregate Root is the parent Entity to all other Entities and Value Objects within the Aggregate.

A Repository operates upon an Aggregate Root

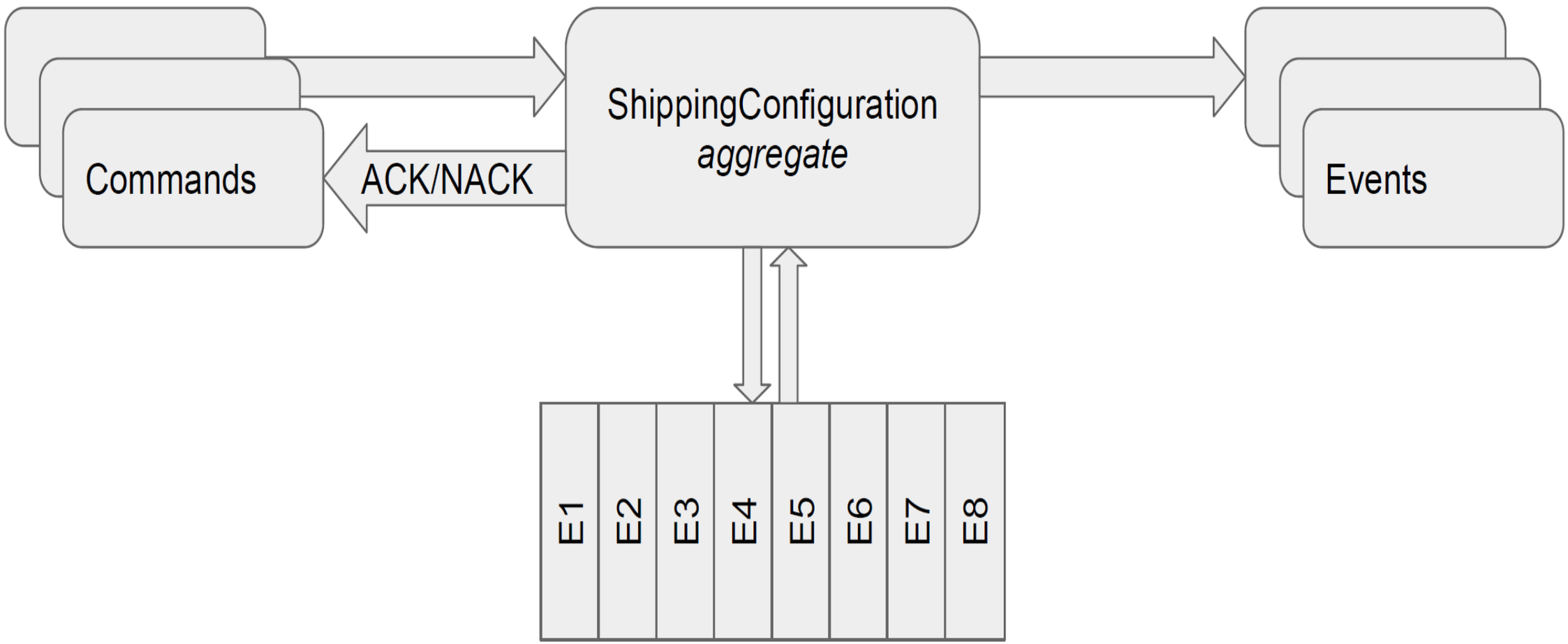
*** What makes the Customer an Aggregate Root is the model's assumption that access to a car or it's components are always through the customer who owns the car.***

Agregat w ujęciu ES

Aggregates are the basic element of transfer of data storage - you request to load or save whole aggregates. Transactions should not cross aggregate boundaries.



Agregat jako zdarzenia



Ujęcie funkcyjne

State = Hn(H2(H1(initialState, Event1), Event2), Eventn)

State = events.foldLeft(initialState, handler)

Event Storage

- Eventy trzeba zrzucić w końcu do bazy…
- Pamięć jest droga?
- Czy eventów jest średnio dużo?
- Jeśli jest dużo eventów jak to się będzie wczytywać - czas?

Can generally be thought of as an increasing integer for most cases. The version number is unique and sequential only within the context of a given aggregate. (Kolejność eventów)

Events

Aggregate	UUID
Payload (Event data)	JSON
Version	int
Date	Timestamp

Event storage - snapshoty

Events

Aggregate	UUID
Payload (Event data)	JSON
Version	int
Date	Timestamp

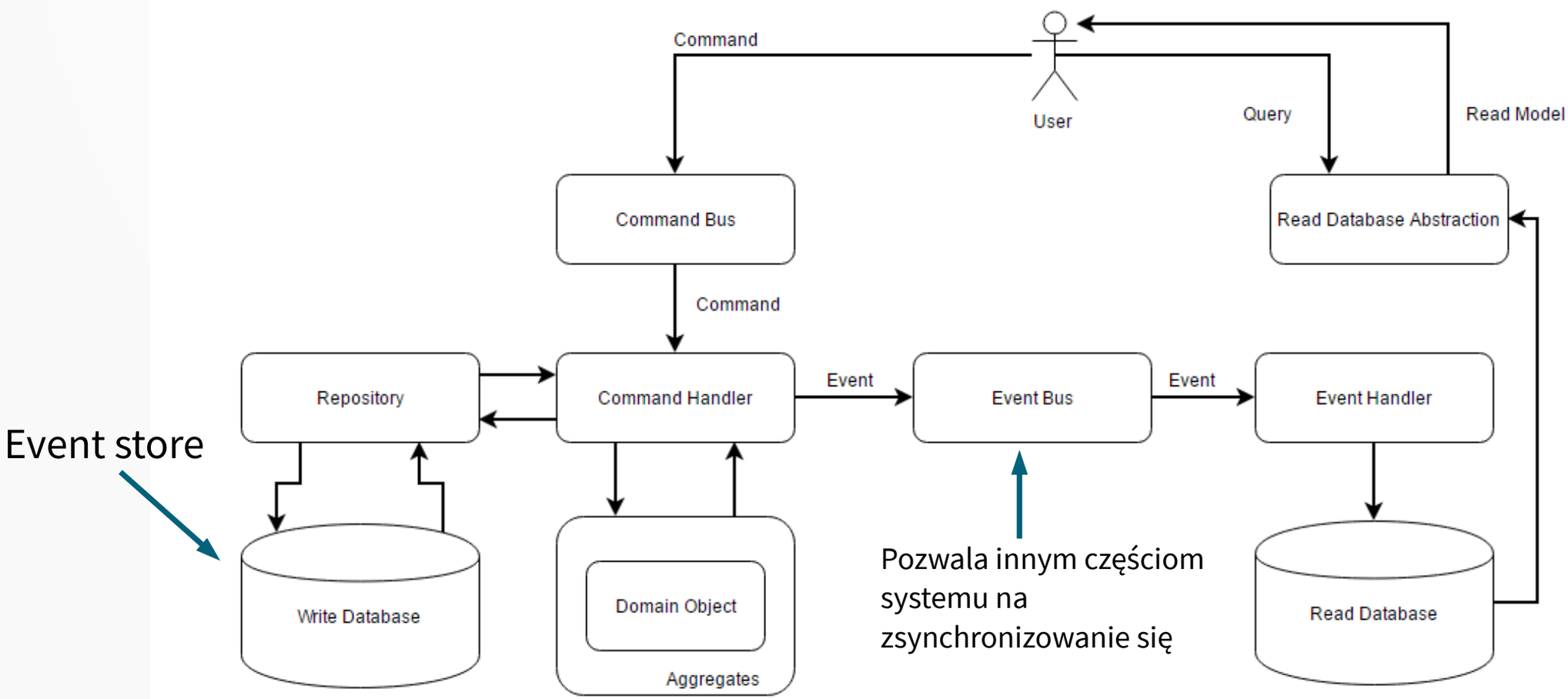
Snapshots

Aggregate	UUID
Payload (Snapshot data)	JSON
Version	int
Date	Timestamp

Query side / CQRS

Command/Domain model – HARD

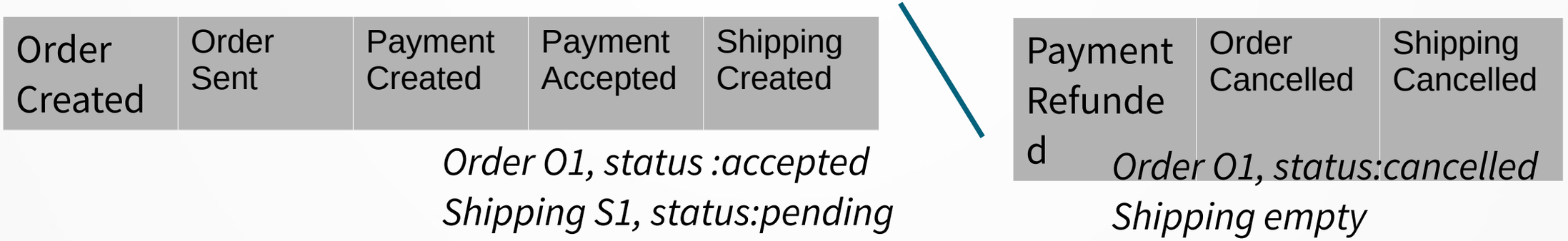
Query model - SIMPLE



Eventual Consistency

- If your „event bus” is asynchronous, you will have to come to grips with the realities of Eventual Consistency.
- What to do about it?
  → Ignore it
  → Notify the user
  „Your blog post has been created and is being processed. It may take several minutes for it to become available.”
  → Wait for the view or read model to become consistent again
  → Discriminate against your read models (prioritization)
  i inne...

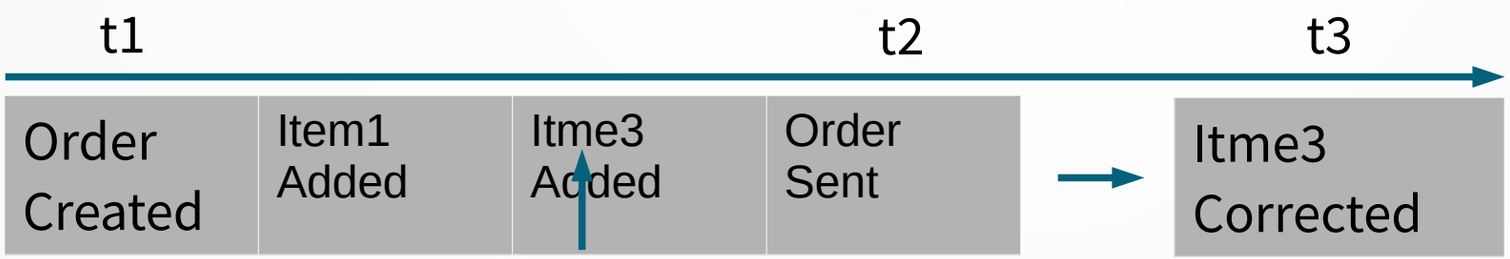
Poprawki - Compensating events



Undo the work performed by a series of steps, which together define an eventually consistent operation, if one or more of the steps fail.

Poprawki - Retroactive events

- Retroactive Event - one that can be used to automatically correct the consequences of a incorrect event that's already been processed.



As at(t2)

Order	Item
O1	Item1
O2	Itme3

As of(t2)

Order	Item
O1	Item1
O2	Item3

class EventProcessor...

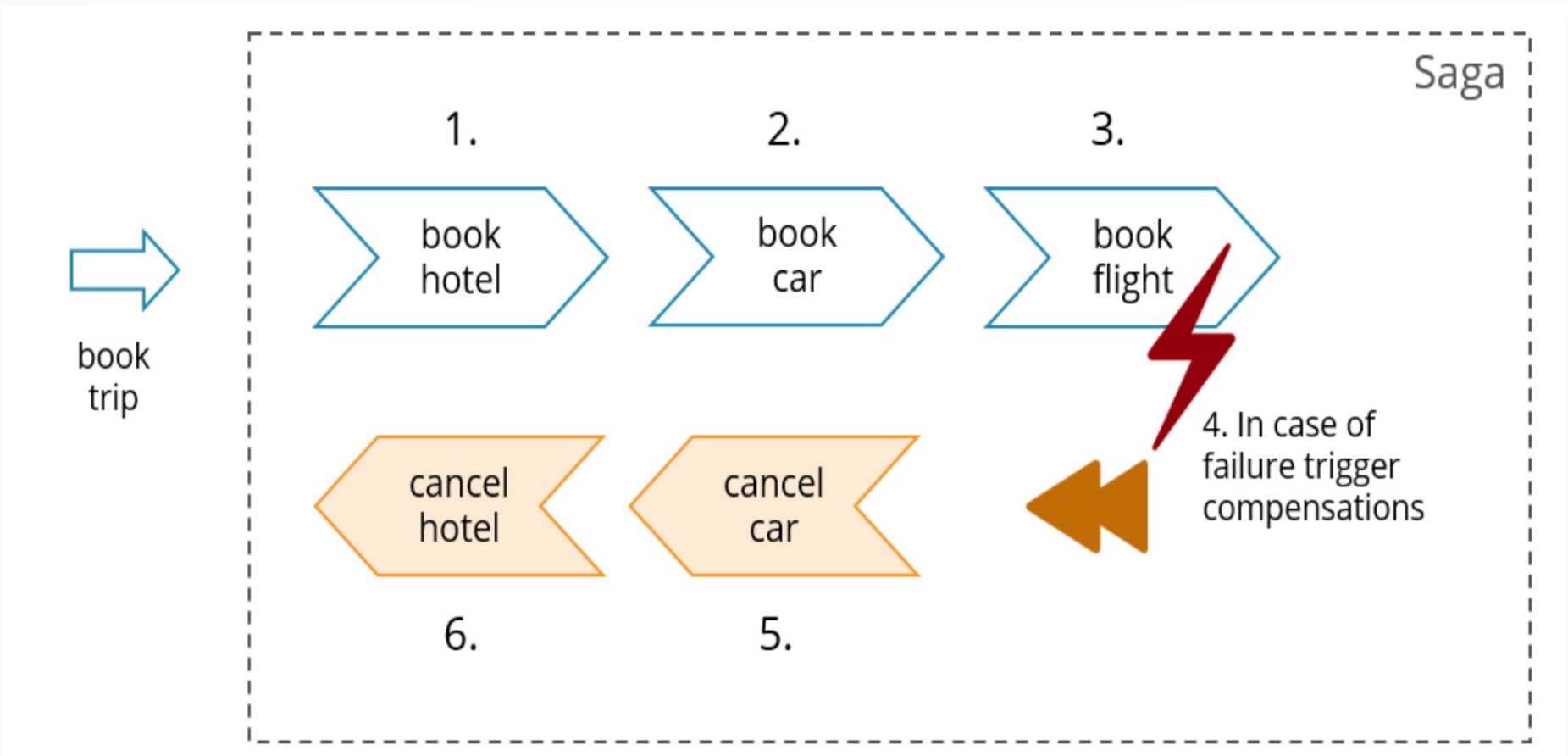
```
private void ProcessOutOfOrder(DomainEvent e) {
  RewindTo(e);
  BasicProcessEvent(e);
  ReplayAfter(e);
}
```

← Out of order Events

Rejected Event (should not have been processed)
Incorrect events (probably valid event types but that carry invalid information)

SAGA pattern

- The Saga pattern describes how to solve distributed (business) transactions without two-phase-commit as this does not scale in distributed systems. The basic idea is to break the overall transaction into multiple steps or activities. Only the steps internally can be performed in atomic transactions but the overall consistency is taken care of by the Saga.



Poprawki - Event upcasting

- Upcasters are classes that take one input event of revision x and output zero or more new events of revision x + 1. Moreover, upcasters are processed in a chain, meaning that the output of one upcaster is sent to the input of the next. This allows you to update events in an incremental manner, writing an Upcaster for each new event revision, making them small, isolated, and easy to understand.
- Event_V2 = f(Event_V1)
- Upcasting satisfies the requirement for an immutable event store. An upcaster is called when reading (old) events from a stream or store, before these events are pushed to event listeners, projections, etc. The original events are not touched, but the upcaster can map our new understanding of the world on those events.
- Upcasting in Axon (← taki framework) is performed in a process before events are sent to the application, meaning that the rest of the application can be completely ignorant about outdated events.

Nasuwające się pytania

- Rodo?

  → W ES dane szyfrowane, klucze poza Event Store

  → W ES odnośniki do danych, faktyczne dane poza ES

  → Eventy odwracające (kontrowersyjne)

  → Grzebanie w historii (mega kontrowersyjne)

Przykłady w kodzie

PYTANIA?

Przydatne linki

https://www.youtube.com/watch?v=JHGkaShoyNs ← Greg Young o Event Sourcingu / CQRS
https://www.youtube.com/watch?v=xDuwrtwYHu8 ← Talk o Saga pattern
https://www.youtube.com/watch?v=dnUFEg68ESM ← Talk Erica Evansa związany z DDD
https://github.com/pilloPl/shop ← przykładowy „projekt”
https://axoniq.io/ ← Axon framework

https://martinfowler.com/bliki/DDD_Aggregate.html
https://martinfowler.com/eaaDev/RetroactiveEvent.html
https://www.erikheemskerk.nl/event-sourcing-eventual-consistency-responding-to-events/
https://docs.microsoft.com/en-us/azure/architecture/patterns/compensating-transaction
https://blog.trifork.com/2012/04/17/refactoring-in-an-event-sourced-world-upcasting-in-axon-2/