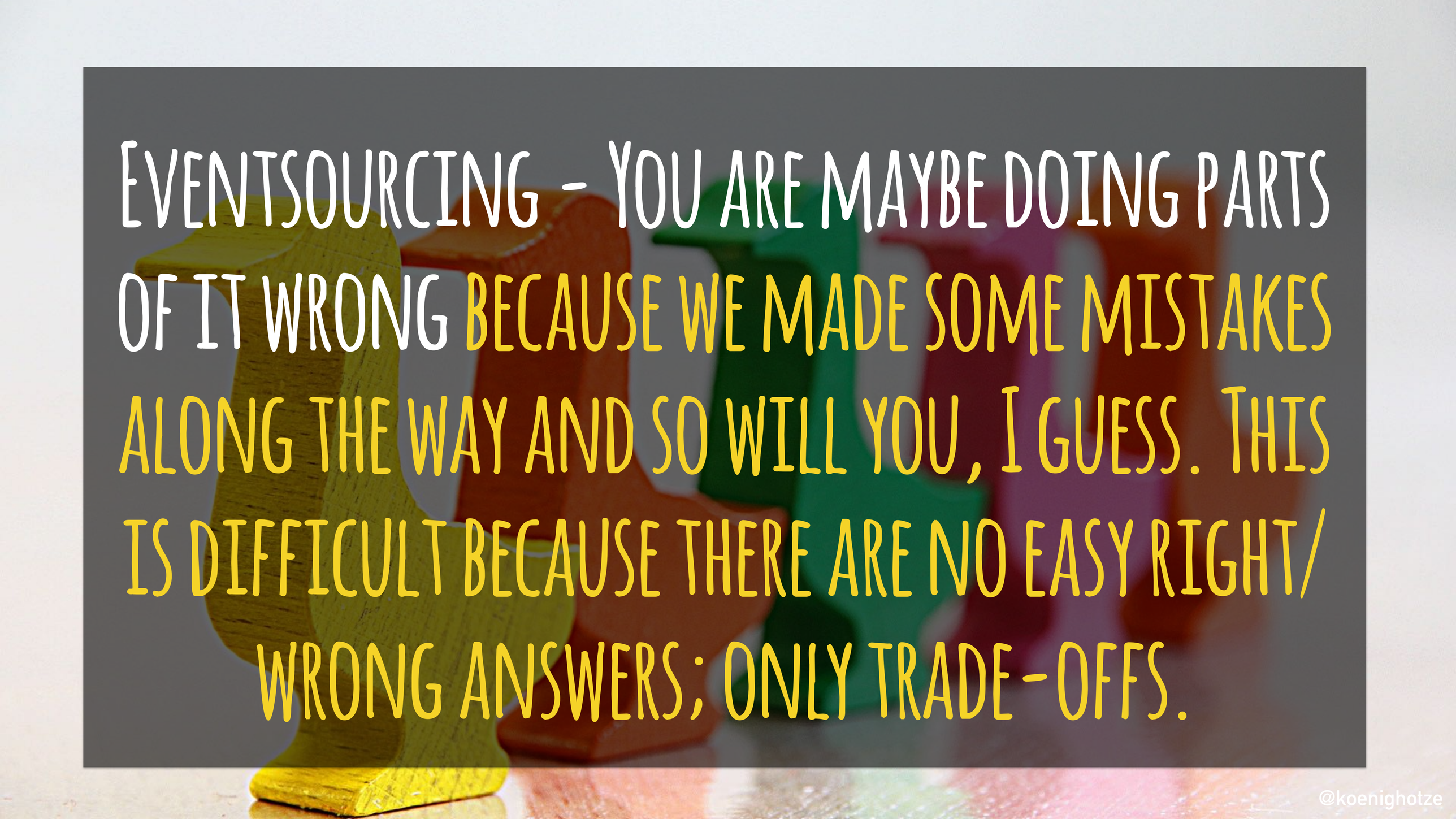
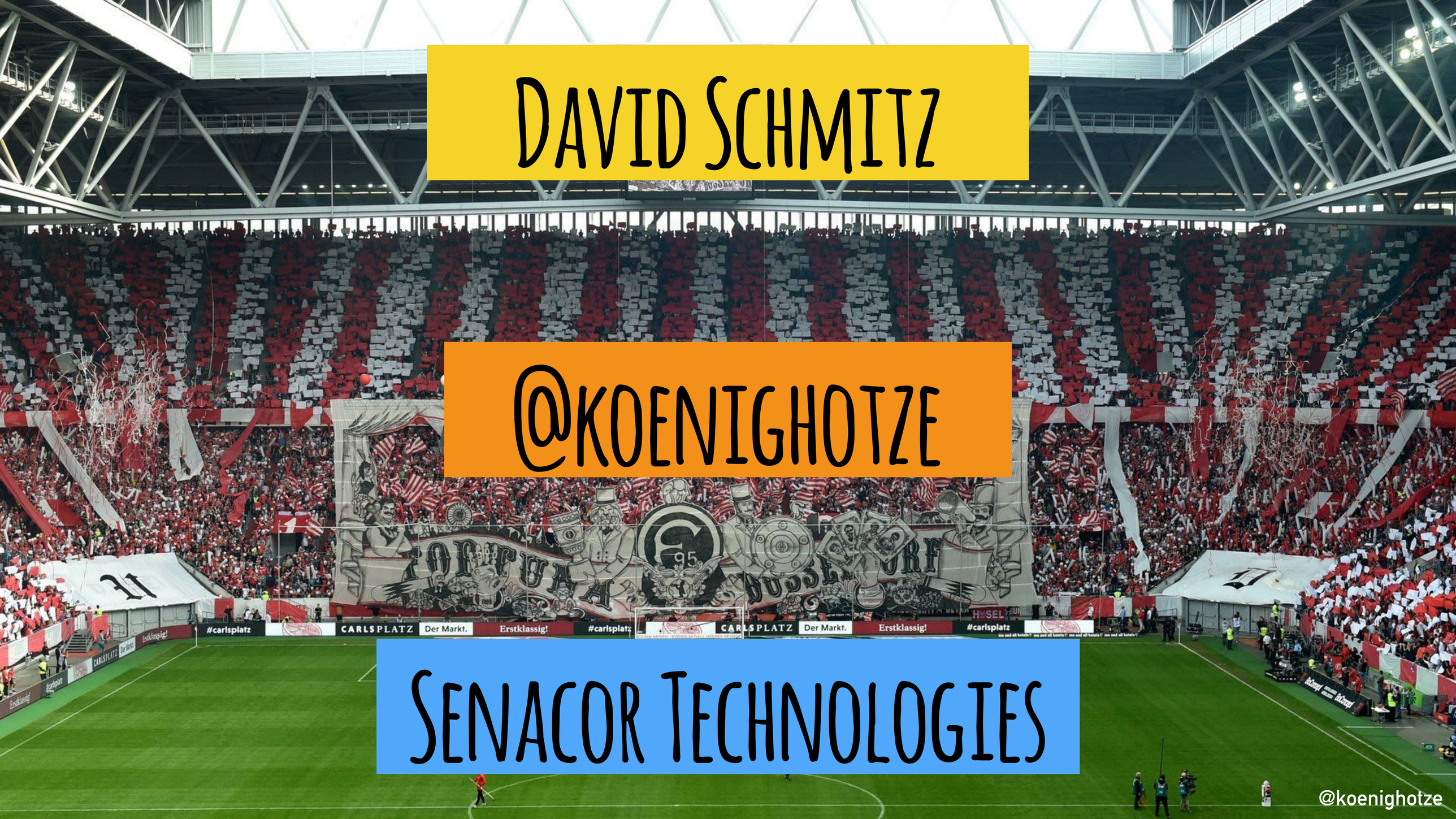

Eventsourcing
You are doing it wrong
@koenighotze


EVENTSOURCING
YOU ARE DOING IT WRONG BECAUSE I KNOW BEST
@koenighotze


Eventsourcing - You are maybe doing parts of it wrong because we made some mistakes along the way and so will you, I guess. This is difficult because there are no easy right/wrong answers; only trade-offs.
@koenighotze


EVENTSOURCING
YOU ARE PROBABLY DOING PARTS OF IT WRONG
@koenighotze


David Schmitz
@koenighotze
Senacor Technologies
#carlsplatz
CARLSPLATZ
Der Markt.
Erstklassig!
@koenighotze

ARE YOU BUILDING MICROSERVICES?
ARE YOU DOING DOMAIN DRIVEN DESIGN?
ARE YOU APPLYING EVENTSOURCING?
ARE YOU USING KAFKA AS AN EVENTSTORE?

Typical misconceptions

Patterns "we" found useful

Traps to avoid

Not a Kafka-rant

What works for us, might not work for you

...and the other way around

👑David Schmitz💾
@koenighotze

F4CK! The moment you notice, it is a 40 min slot and not a 60 min slot. Well, I still have 2 days...or...I'll just speak faster!

¯_(ツ)_/¯

11:11 AM · Mar 18, 2019


@koenighotze

# Eventsourcing bootcamp

@koenighotze

# Domain driven design
# Event driven architecture
# Distributed systems

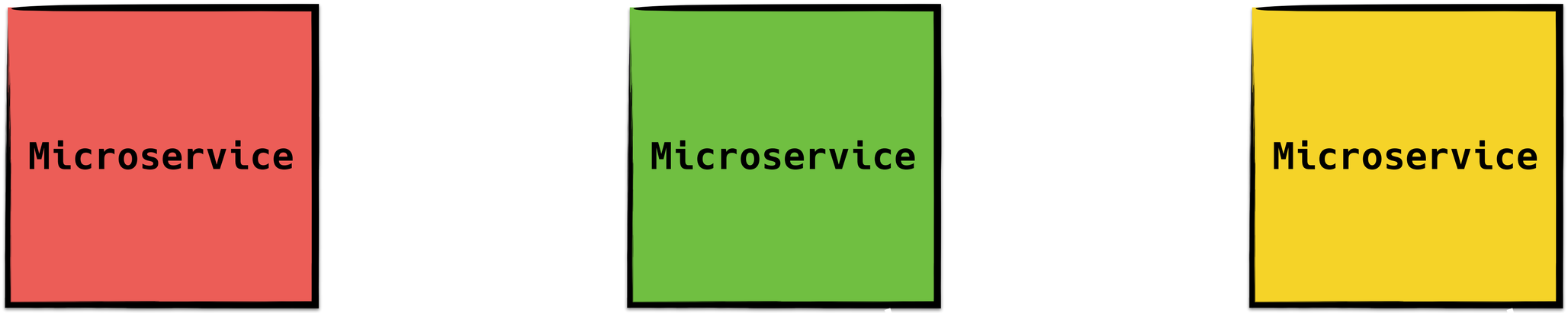
# Event driven business


Sip If You Dare!
Chilling White Mocha with a Haunting of Raspberry.
Coffee ordered

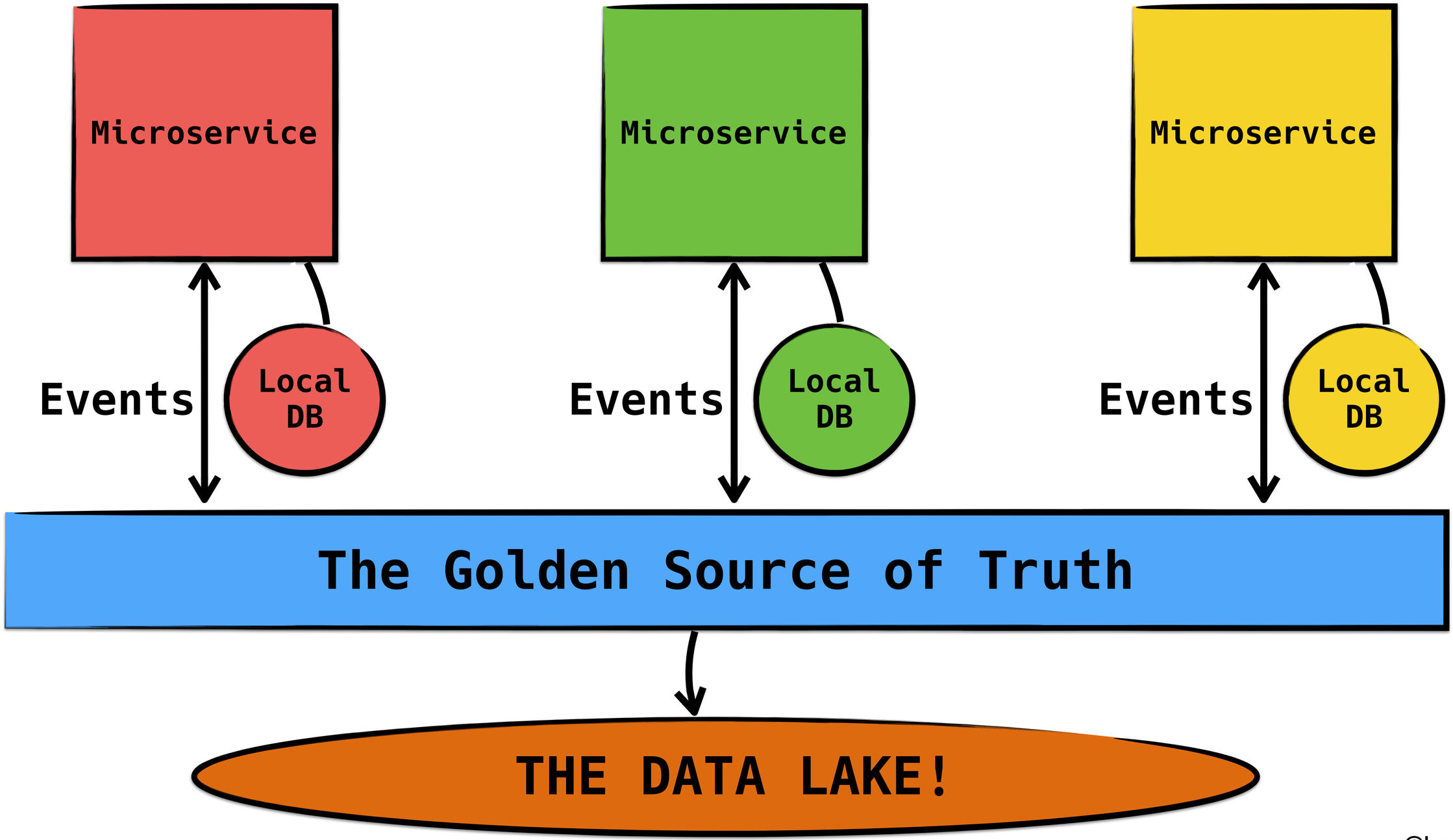

Order payed
Sip If You Dare!
Chilling White Mocha with a Haunting of Raspberry
AMAZING INGREDIENTS


Coffee prepared
Chilling White Mocha with a Haunting of Raspberry.
CLASSIC WHOLE GRAIN OATMEAL

# Eventsourcing from an architects perspective

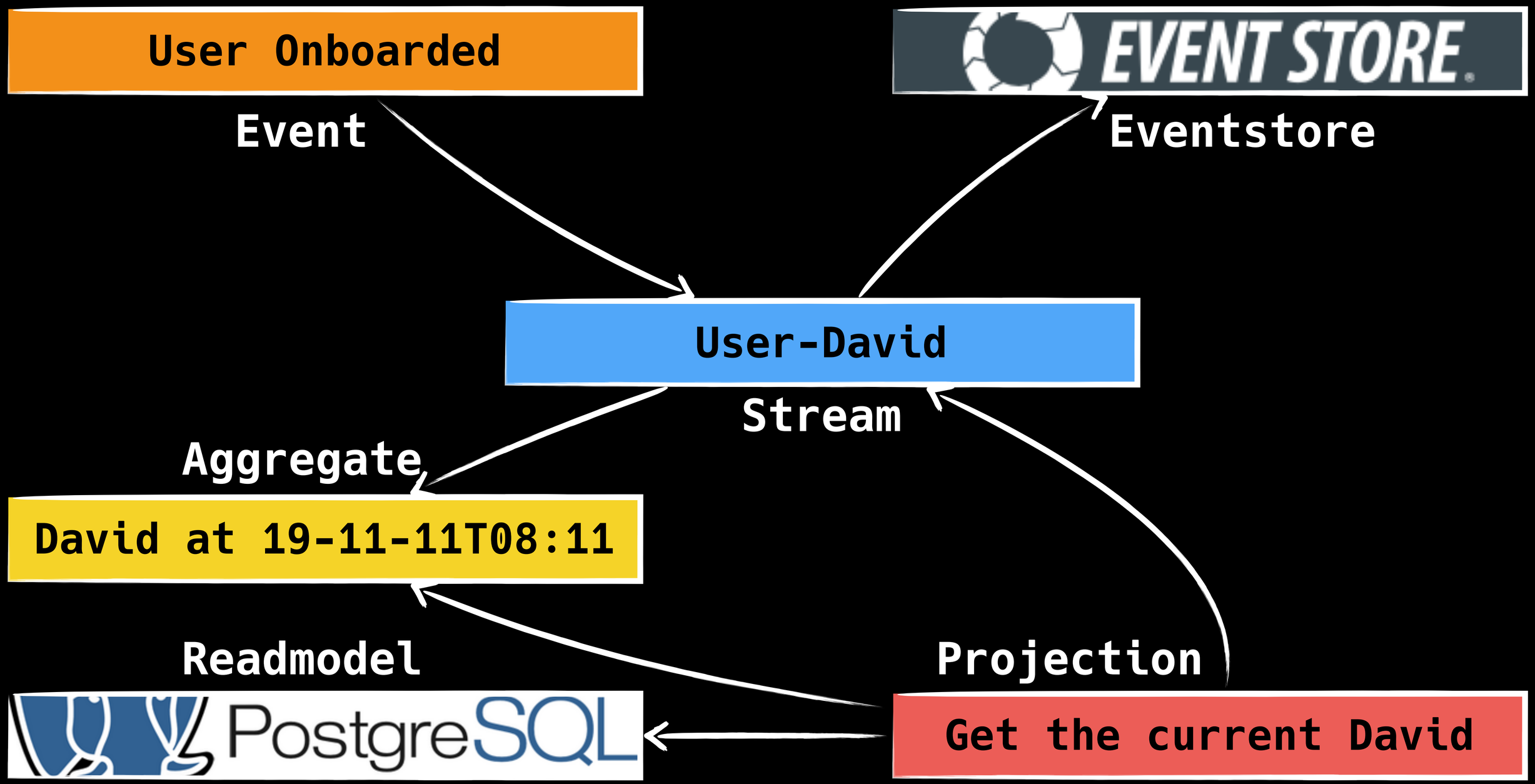


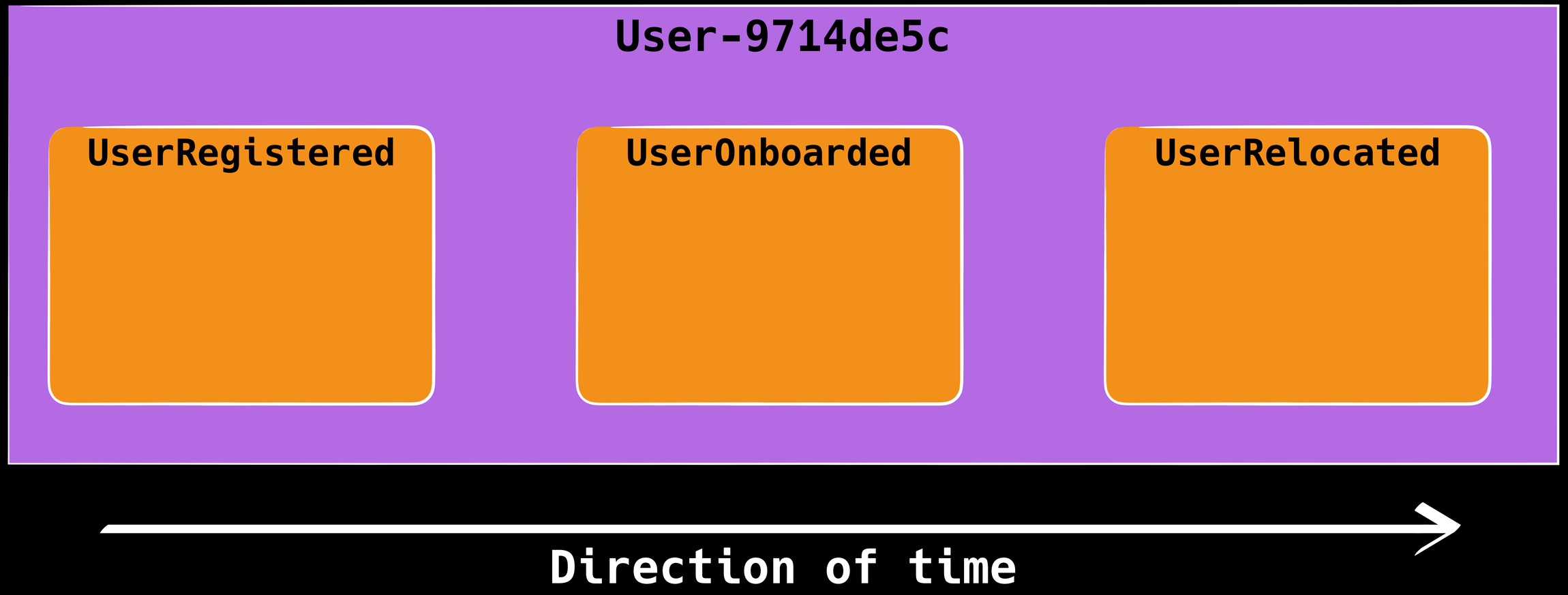

# How to represent data and how to represent dynamics?

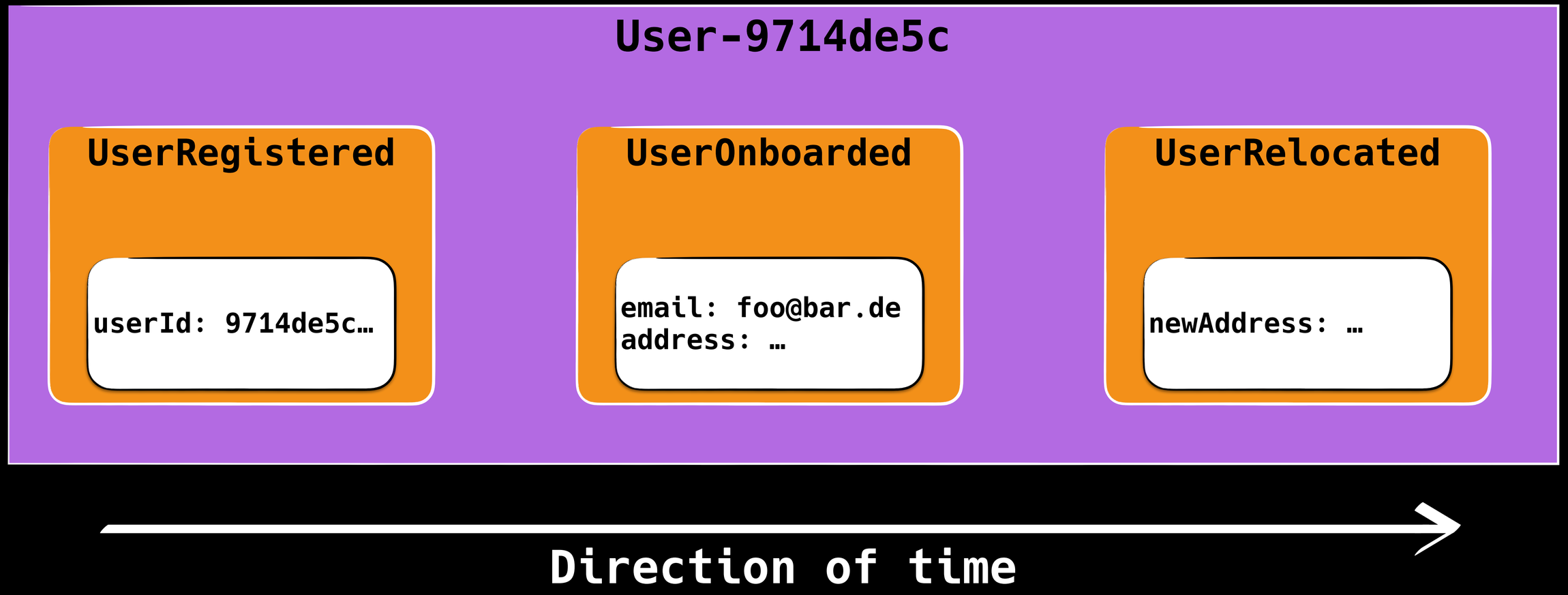

Microservice
Microservice
Microservice
Events
Local DB
Events
Local DB
Events
Local DB
The Golden Source of Truth
THE DATA LAKE!

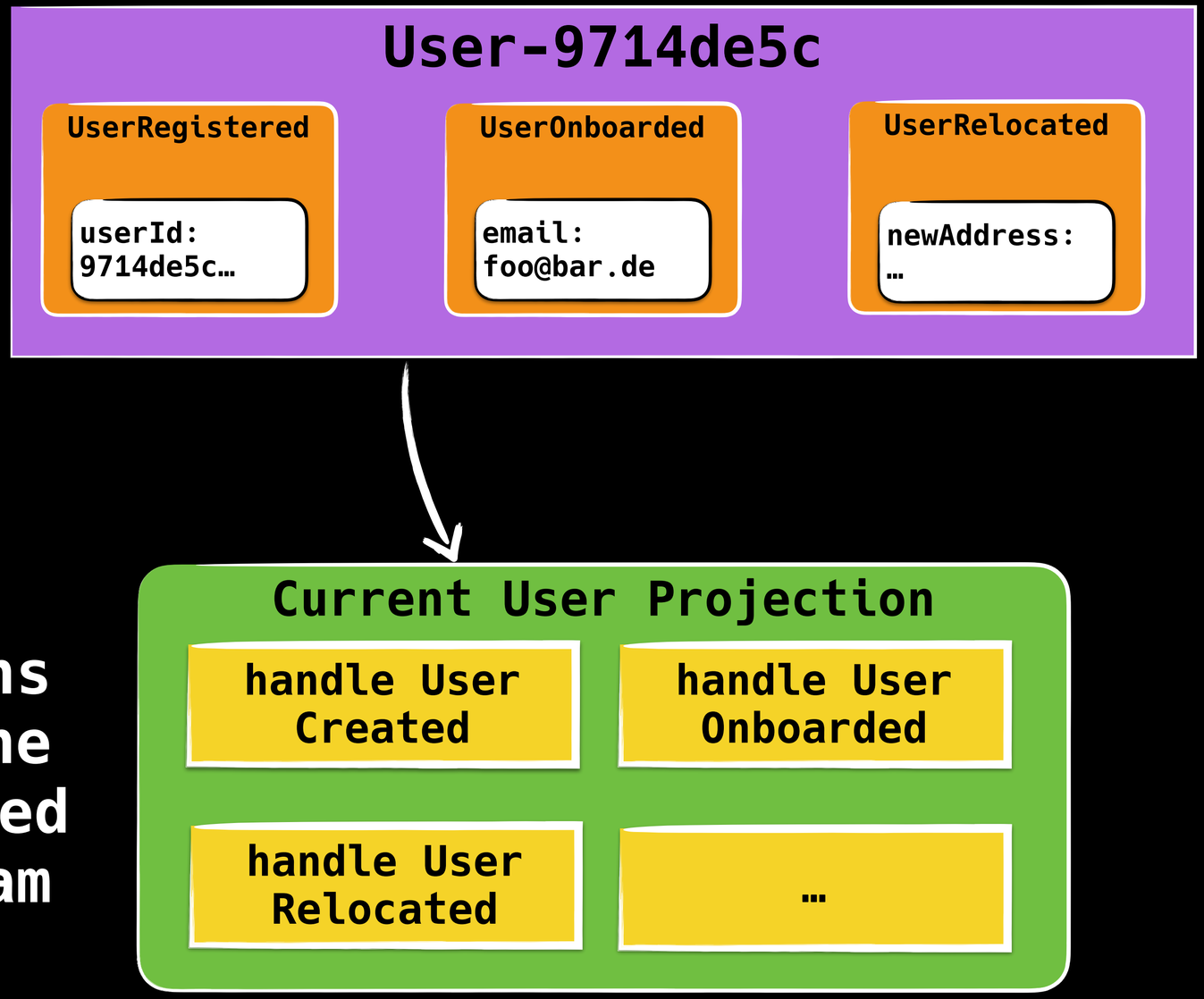

Microservice
Microservice
Microservice
Even
Never mind the details.
That is eventsourcing magic.
Just do the Right Thing!
THE DATA LAKE!

# JUST USE MAGIC

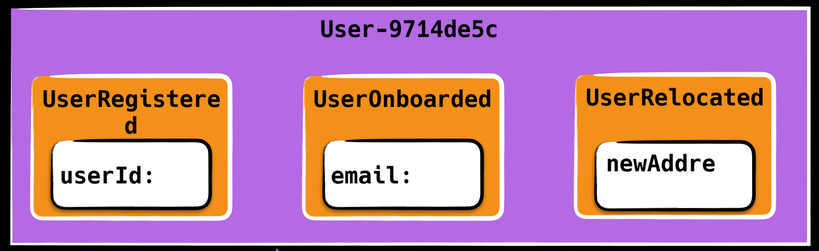

User Onboarded
Event
EVENT STORE.
Eventstore
User-David
Stream
Aggregate
David at 19-11-11T08:11
Readmodel
PostgreSQL
Projection
Get the current David

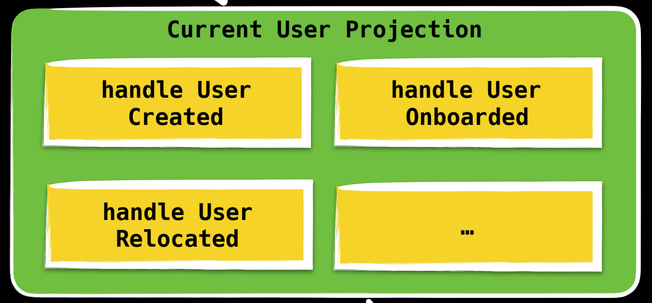

User-9714de5c
UserRegistered
UserOnboarded
UserRelocated
Direction of time


User-9714de5c
UserRegistered
userId: 9714de5c…
UserOnboarded
email: foo@bar.de
address: …
UserRelocated
newAddress: …
Direction of time




Projections hydrate the state stored in a stream



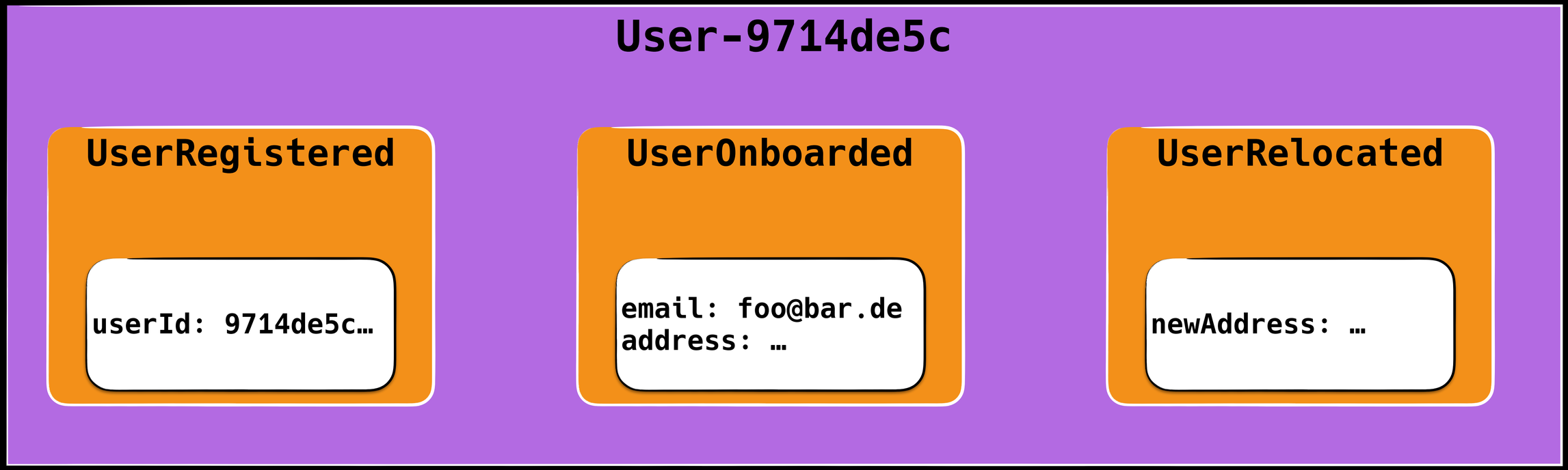




**Represents the user at the time of the last event read**

## User Aggregate at 20:13 11.03.2008

```
userId: 9714de5c…
email: foo@bar.de
address: <new address>
```


Read models
...and why you may not need to them (initially)
@koenighotze

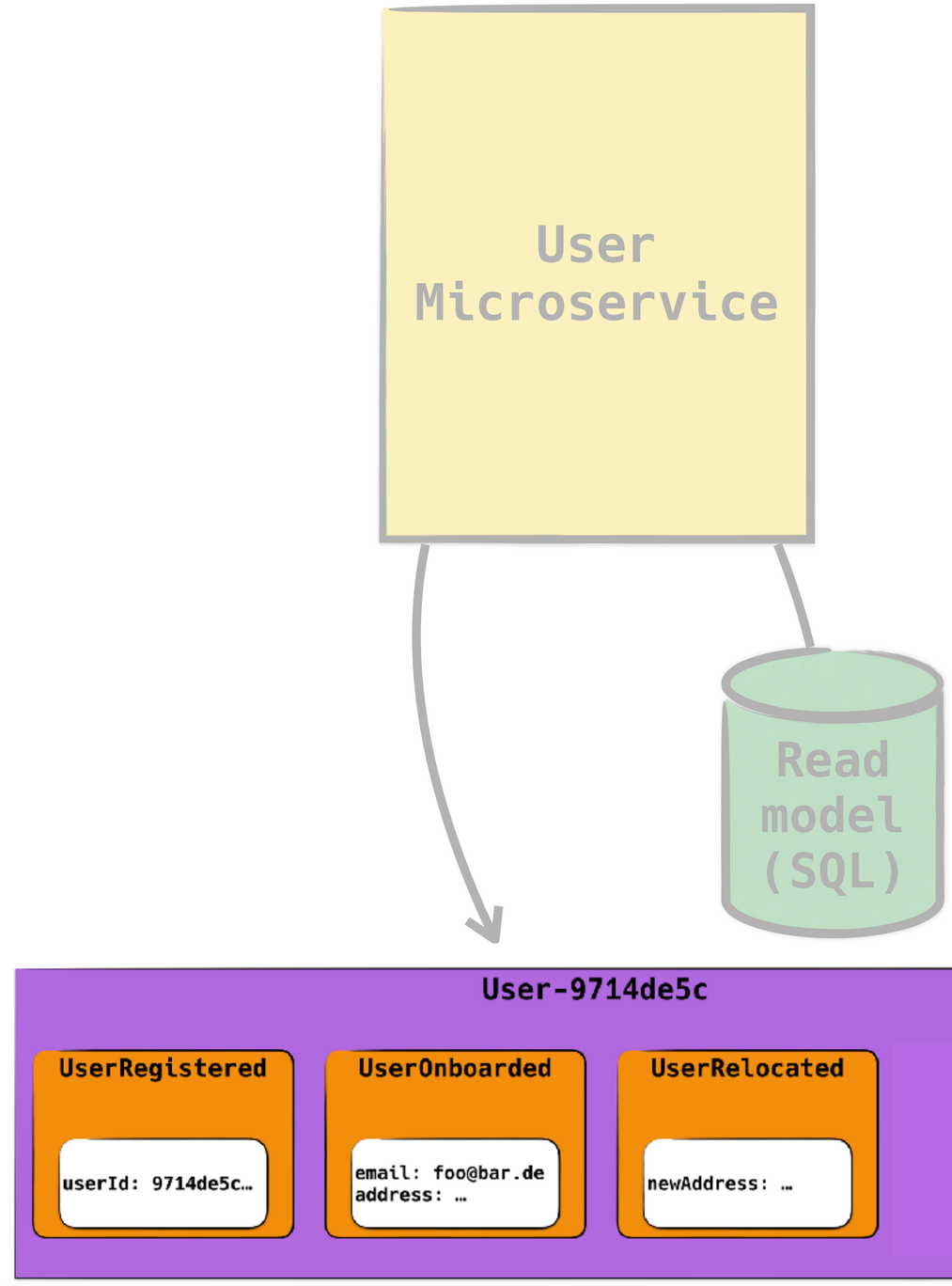

User-9714de5c
UserRegistered
userId: 9714de5c…
UserOnboarded
email: foo@bar.de
address: …
UserRelocated
newAddress: …


User-9714de5c
UserRelocated
newAddress: …

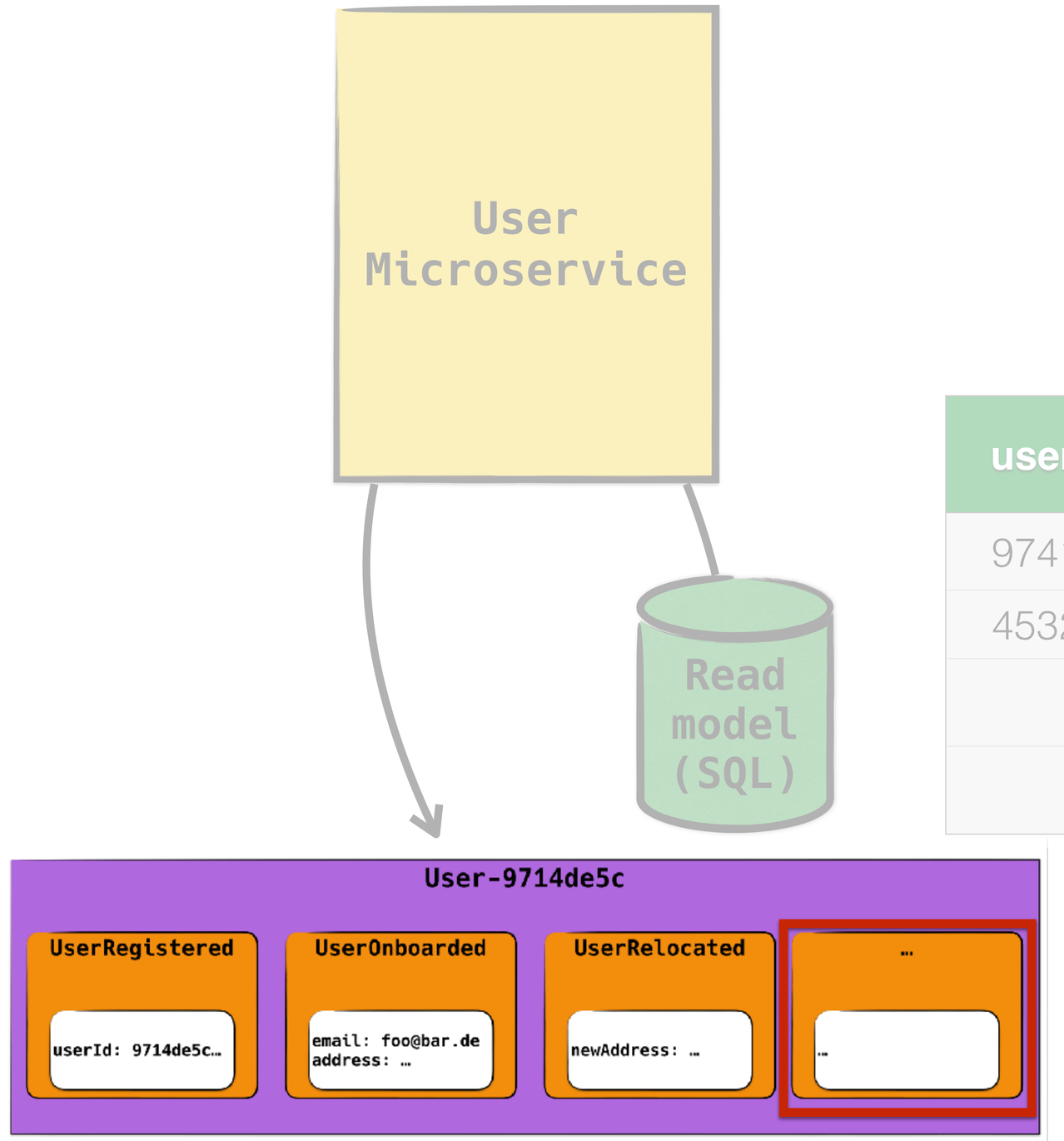

User Aggregate at 20:13 11.03.2008
userId: 9714de5c…
email: foo@bar.de
address: <new address>

# How can we handle read models?

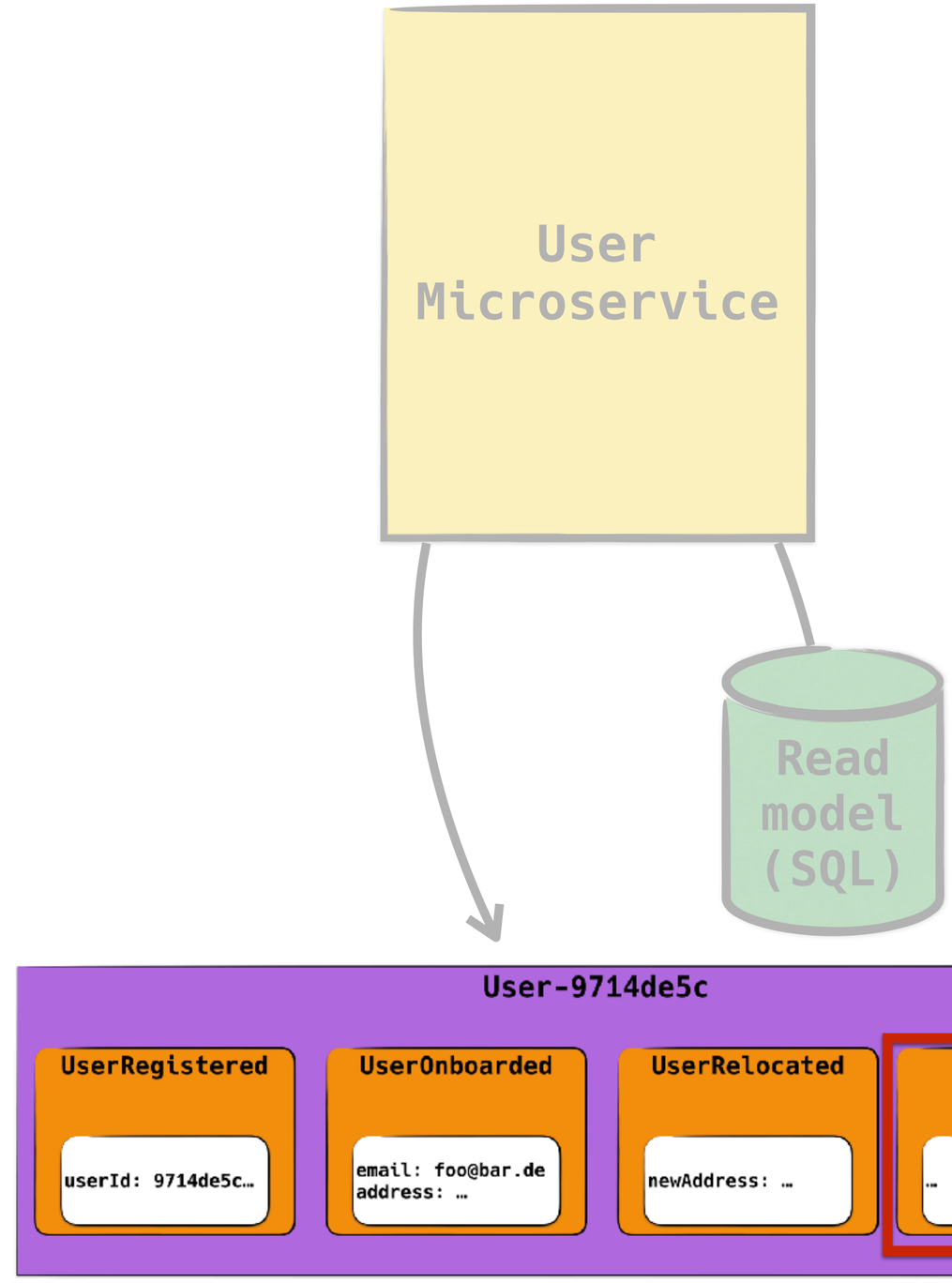
# Just use a local database



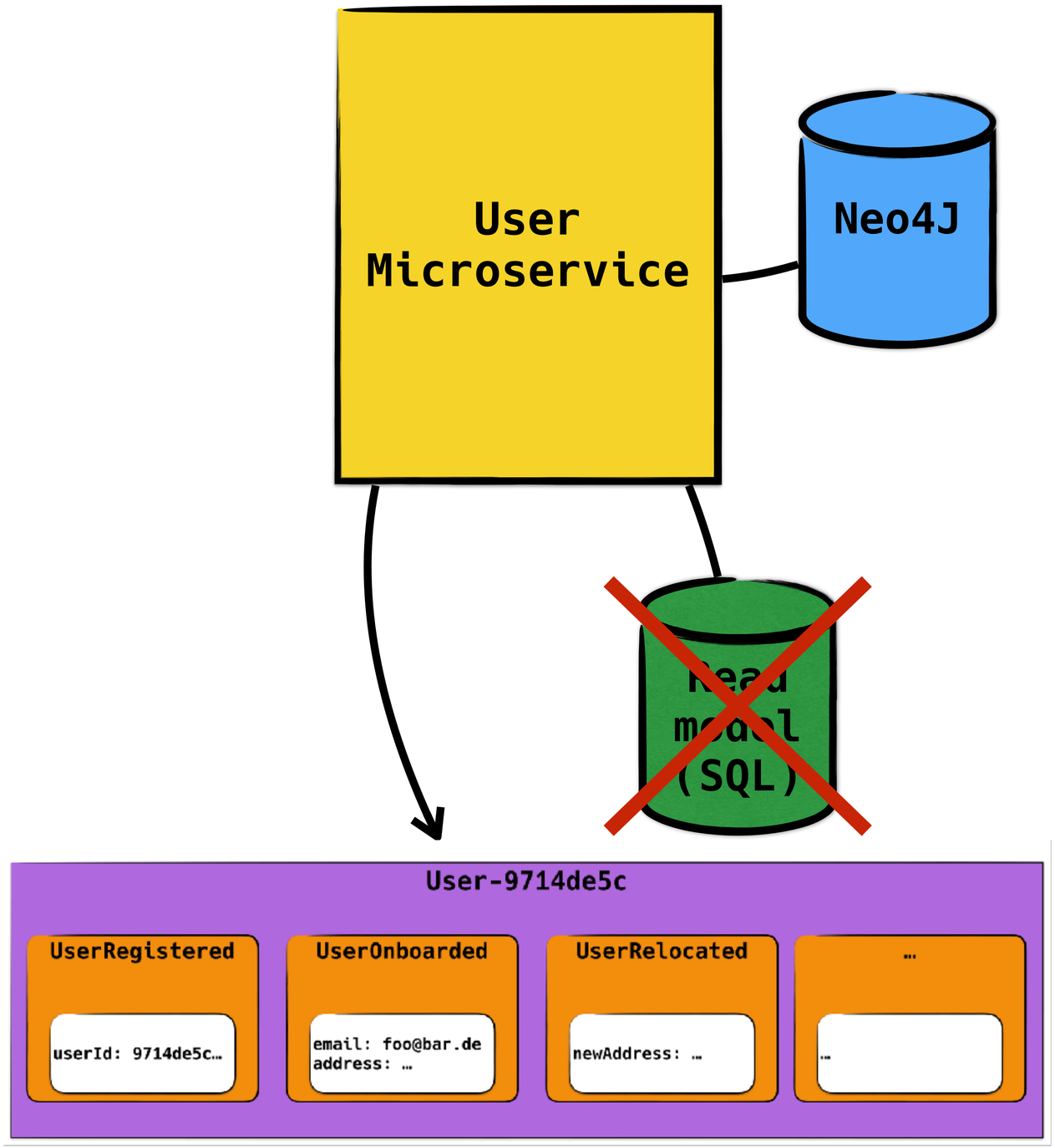

| user id | last event id | user name | email | street |
|---|---|---|---|---|
| 9741… | 3 | David | foo@bar.de | … |
| 4532… | 7 | Martin | null | … |
| | | | | |
| | | | | |




| user id | last event id | user name | email | street |
|---|---|---|---|---|
| 9741… | 3 | David | foo@bar.de | … |
| 4532… | 7 | Martin | null | … |
| | | | | |
| | | | | |




| user id | last event id | user name | email | street |
|---|---|---|---|---|
| 9741… | **4** | David | **qux@ba.ze** | … |
| 4532… | 7 | Martin | null | … |
| | | | | |
| | | | | |

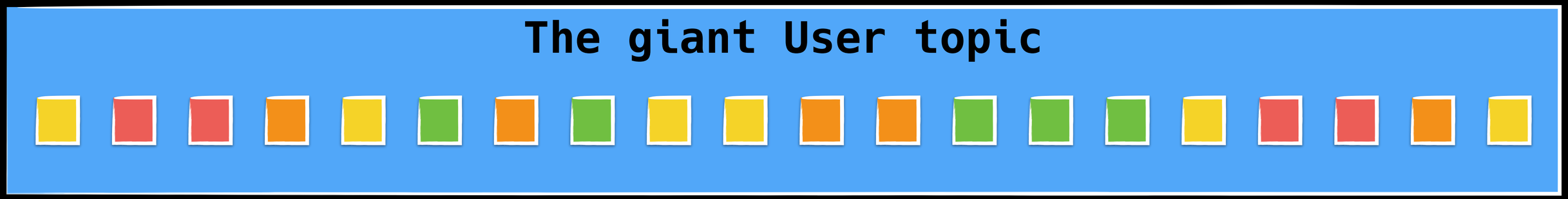

User
Microservice
Neo4J
(SQL)
User-9714de5c
UserRegistered
userId: 9714de5c…
UserOnboarded
email: foo@bar.de
address: …
UserRelocated
newAddress: …
…
…

# Challenges?

Eventual consistency
Replays and rebuilds without downtime
Re-deliveries and Effectively once
Operational complexity

# YOU MAY NOT NEED A READ MODEL

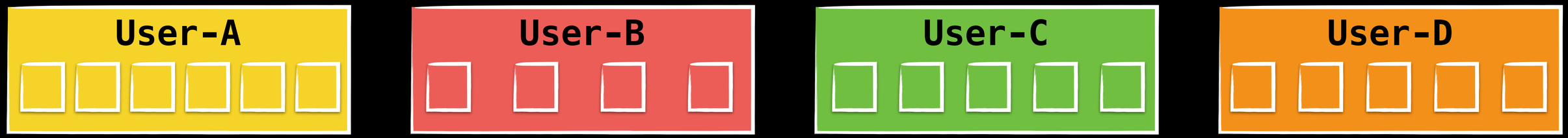
# Typical strategies for storing events



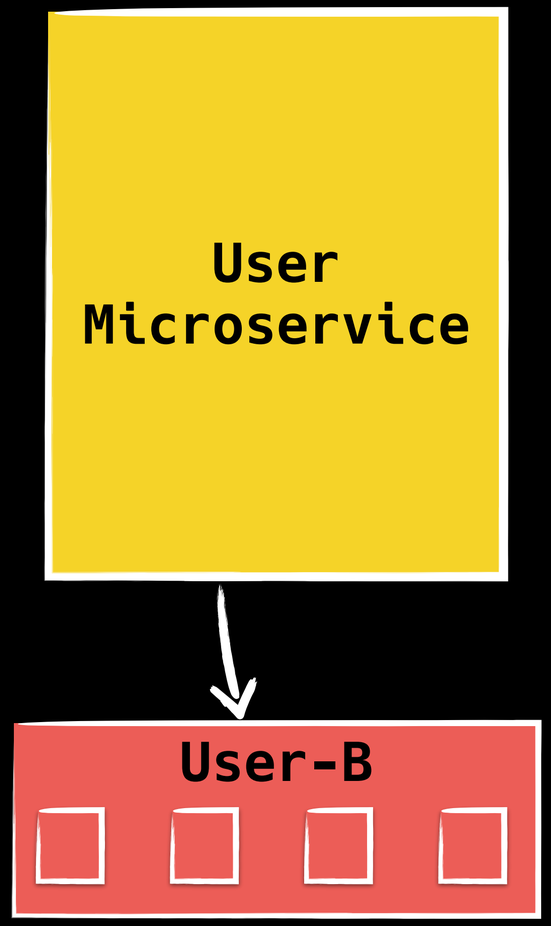

User A User B User C User D

# Better: One stream per aggregate

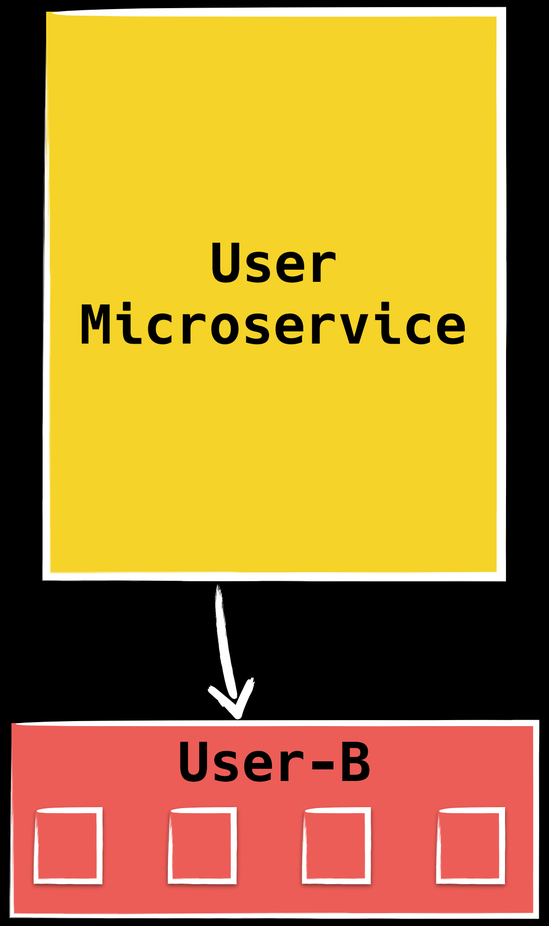
The giant User topic



GET /users/B



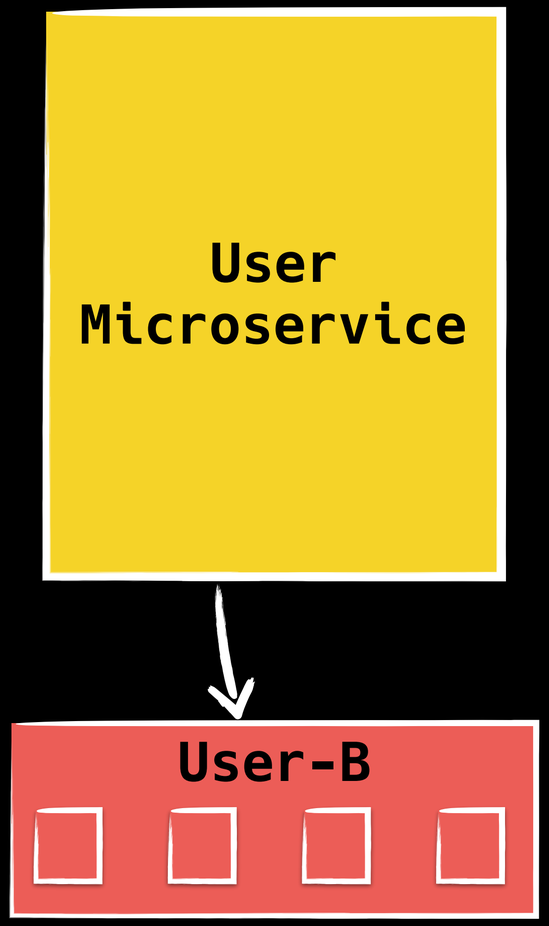
GET /users/B




```
handlers = {
  'UserCreated':   (current, event) => {},
  'UserOnboarded': (current, event) => {},
  'UserDeleted': ...
}
```

```
aggregate = readAggregateFromStream(
  'user',
  'B',
  {},
  fromStartOfStream,
  handlers
)
```

## GET /users/B




```
handlers = {
  'UserCreated':   (current, event) => {},
  'UserOnboarded': (current, event) => {},
  'UserDeleted': ...
}
```

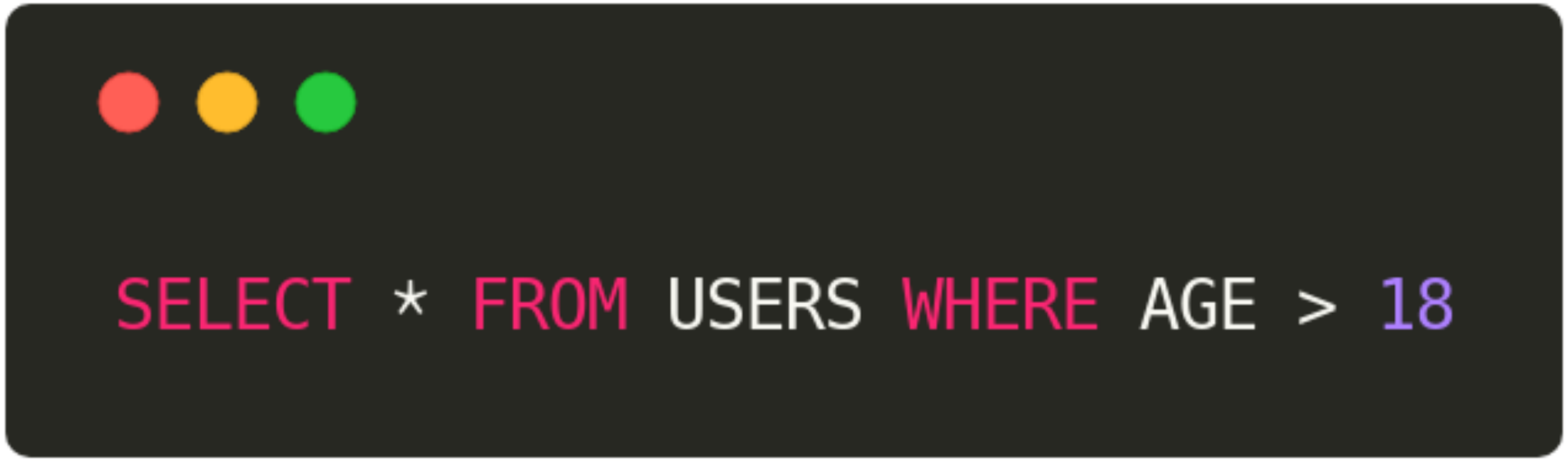
```
aggregate = readAggregateFromStream(
  'user',
  'B',
  {},
  fromStartOfStream,
  handlers
)
```

Consistent read
No operational overhead
Super-simple programming logic

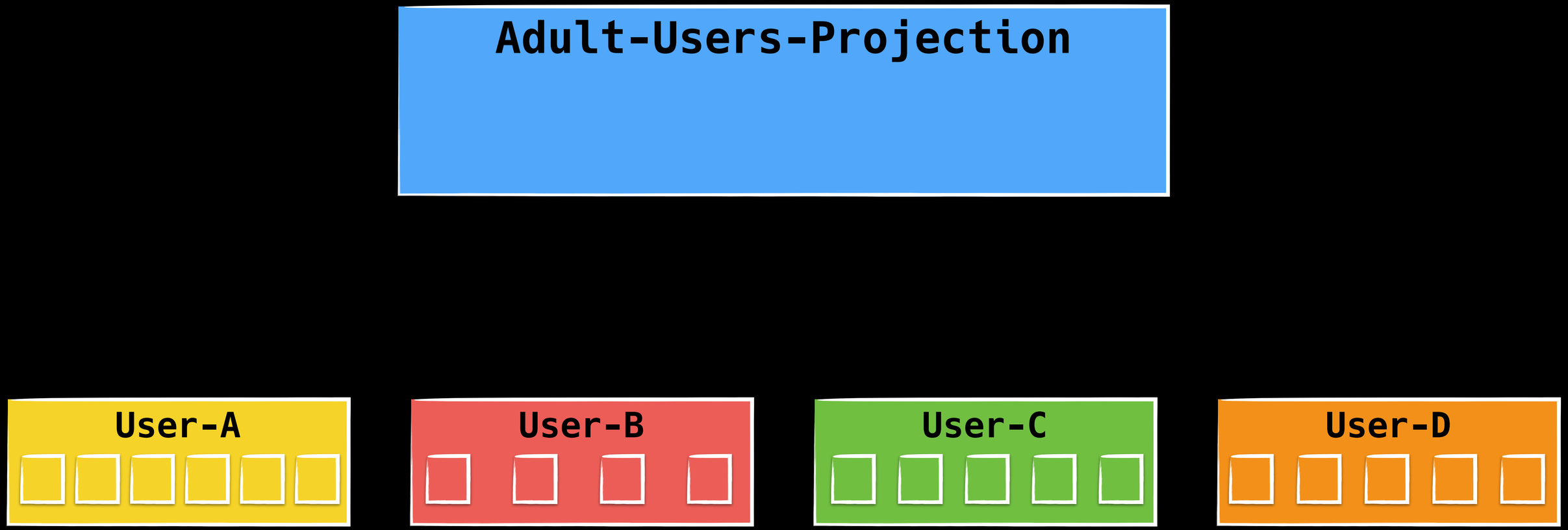
# But, what about speed?

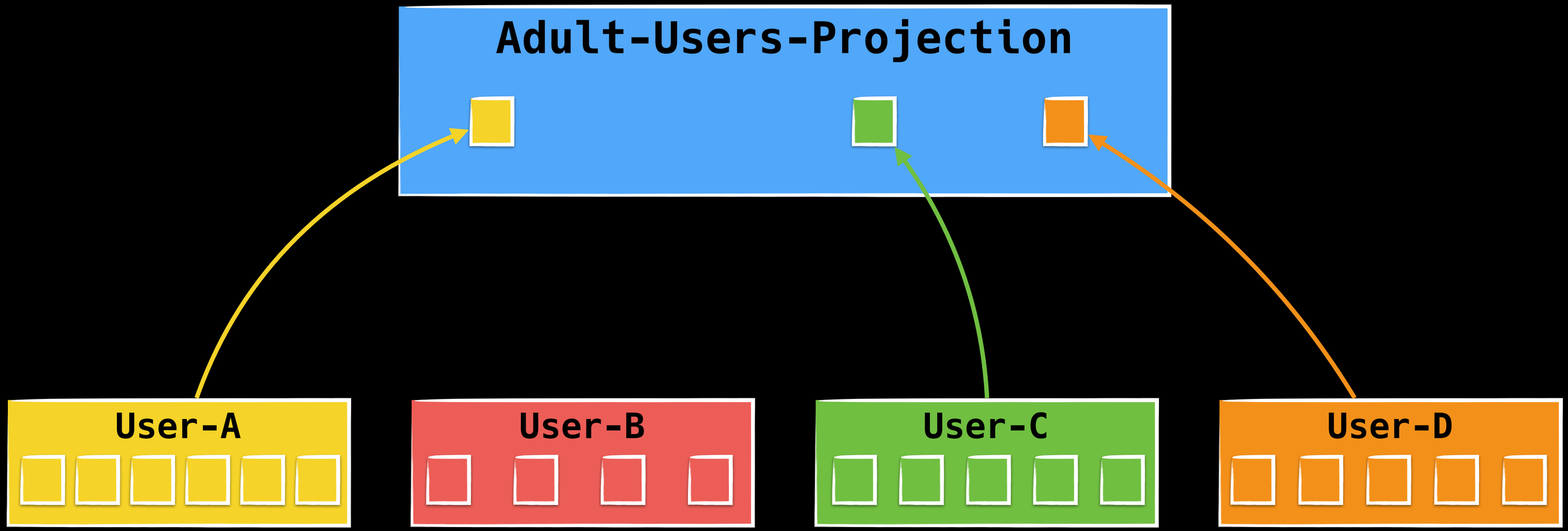
readEntity 100: 66.047ms

# But, what about queries?


SELECT * FROM USERS WHERE AGE > 18

# Build specific query projections


Adult-Users-Projection
User-A
User-B
User-C
User-D


Adult-Users-Projection
User-A
User-B
User-C
User-D

Event handlers are not CPU / IO intensive

Prefer small aggregates

Measure and introduce persistent read models only if needed

# Transactions, concurrency and your eventstore

# How can we guarantee correctness when writing?

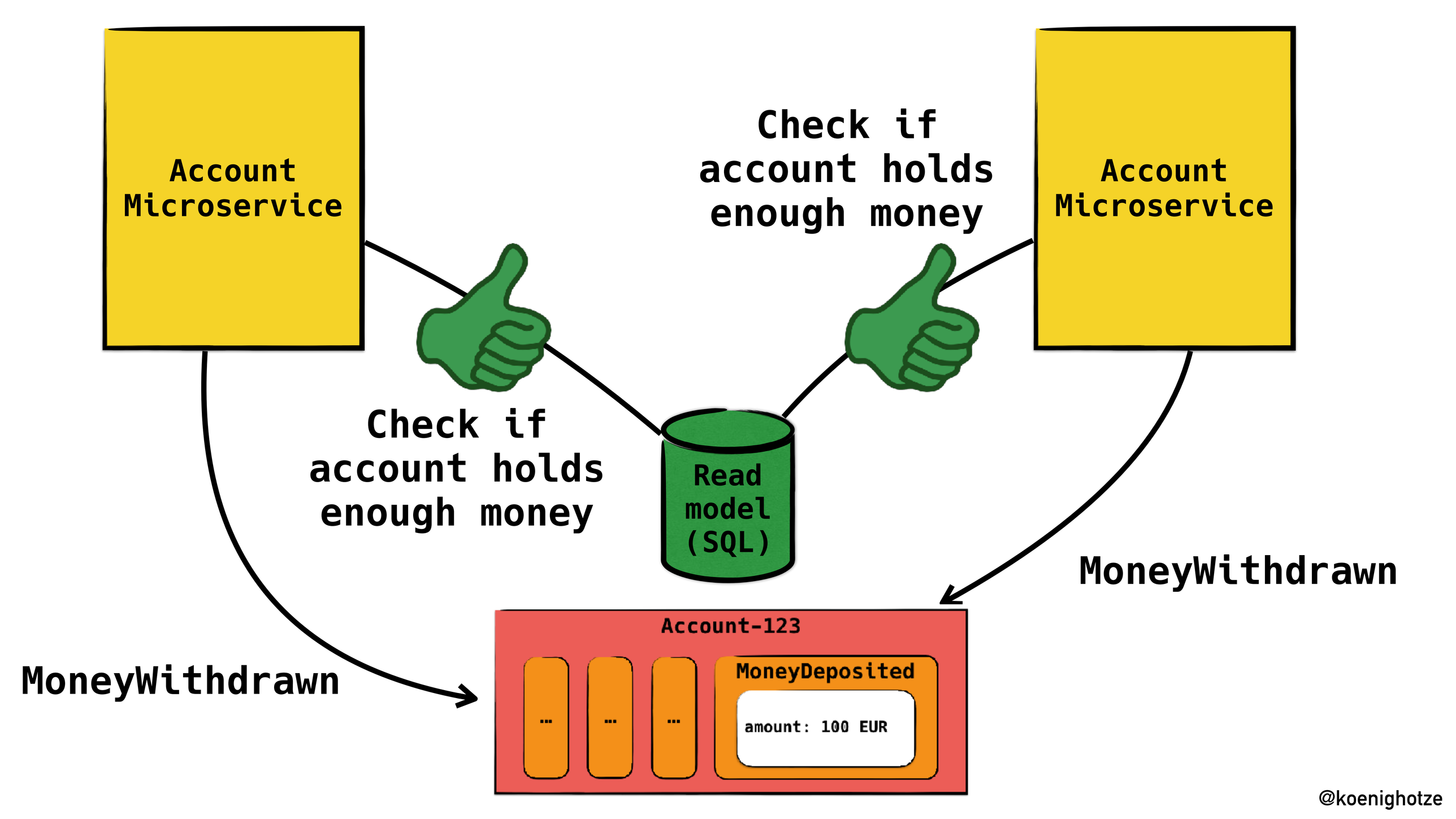
# "Only withdraw money, if the bank account holds enough money!"*

**Actually, a real bank would not want such a business rule. They earn money if you overdraw your account. An overdraft fee is one of the most expensive fees banks charge. Just saying…*

I CAN USE A READMODEL FOR FAST VALIDATION, CAN I?

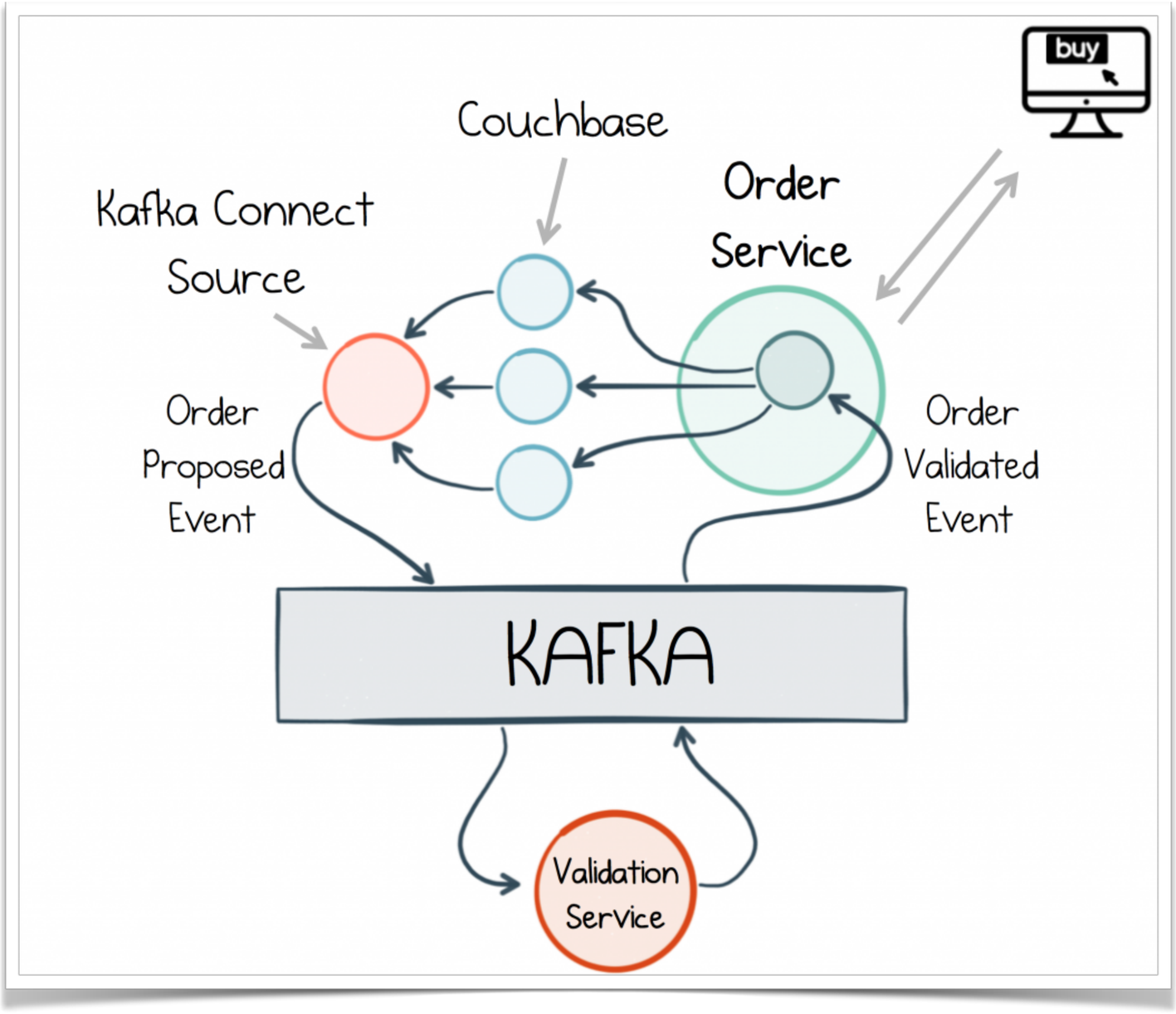
# Maybe not

# ARE YOUR SYSTEMS SINGLE-THREADED?



# Validation against a read model is prone to inconsistencies

JUST MANAGE TRANSACTIONS WITH A DATABASE AND USE SINGLE-WRITER

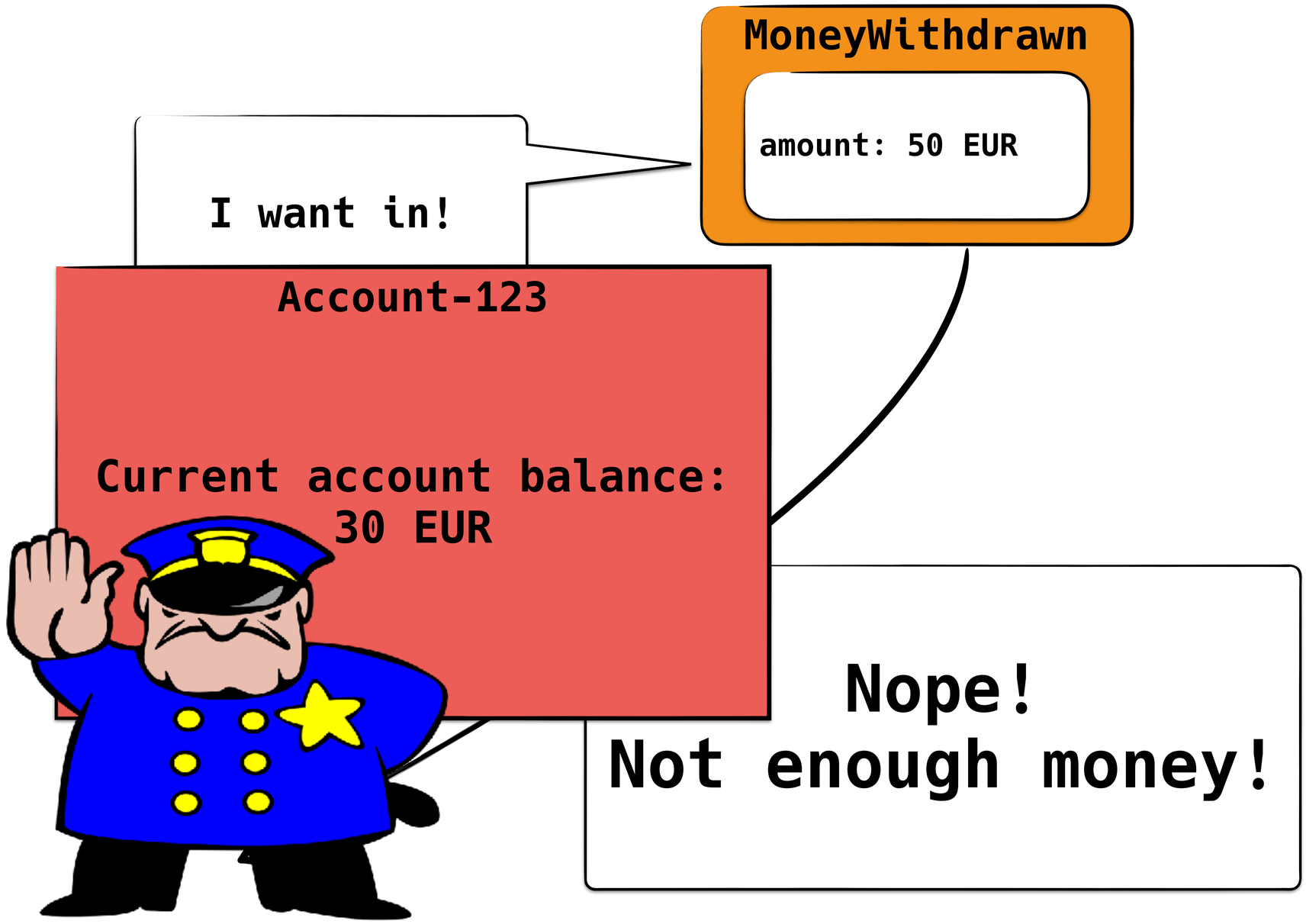



https://www.confluent.io/blog/messaging-single-source-truth/

# Quick tip for finding friends in ops:

ASK THEM TO JUST INSTALL AND MAINTAIN PRODUCTION GRADE KAFKA AND COUCHBASE INSTALLATIONS ON AWS

# The aggregate is responsible for enforcing business invariants


MoneyWithdrawn
amount: 50 EUR
I want in!
Account-123
Current account balance:
30 EUR
Nope!
Not enough money!

# The aggregate is the "Transaction Boundary"

# Optimistic concurrency control

From Wikipedia, the free encyclopedia

**Optimistic concurrency control** (**OCC**) is a concurrency control method applied to transactional systems such as relational database management systems and software transactional memory. OCC assumes that multiple transactions can frequently complete without interfering with each other. While running, transactions use data resources without acquiring locks on those resources. Before committing, each transaction verifies that no other transaction has modified the data it has read. If the check reveals conflicting modifications, the committing transaction rolls back and can be restarted.[1] Optimistic concurrency control was first proposed by H.T. Kung and John T. Robinson.[2]

https://en.wikipedia.org/wiki/Optimistic_concurrency_control

# Optimistic concurrency control

From Wikipedia, the free encyclopedia

**OCC assumes that multiple transactions can frequently complete without interfering with each other.**

transaction has modified the data it has read. If the check reveals conflicting modifications, the committing transaction rolls back and can be restarted.[1] Optimistic concurrency control was first proposed by H.T. Kung and John T. Robinson.[2]

https://en.wikipedia.org/wiki/Optimistic_concurrency_control

# Optimistic concurrency control

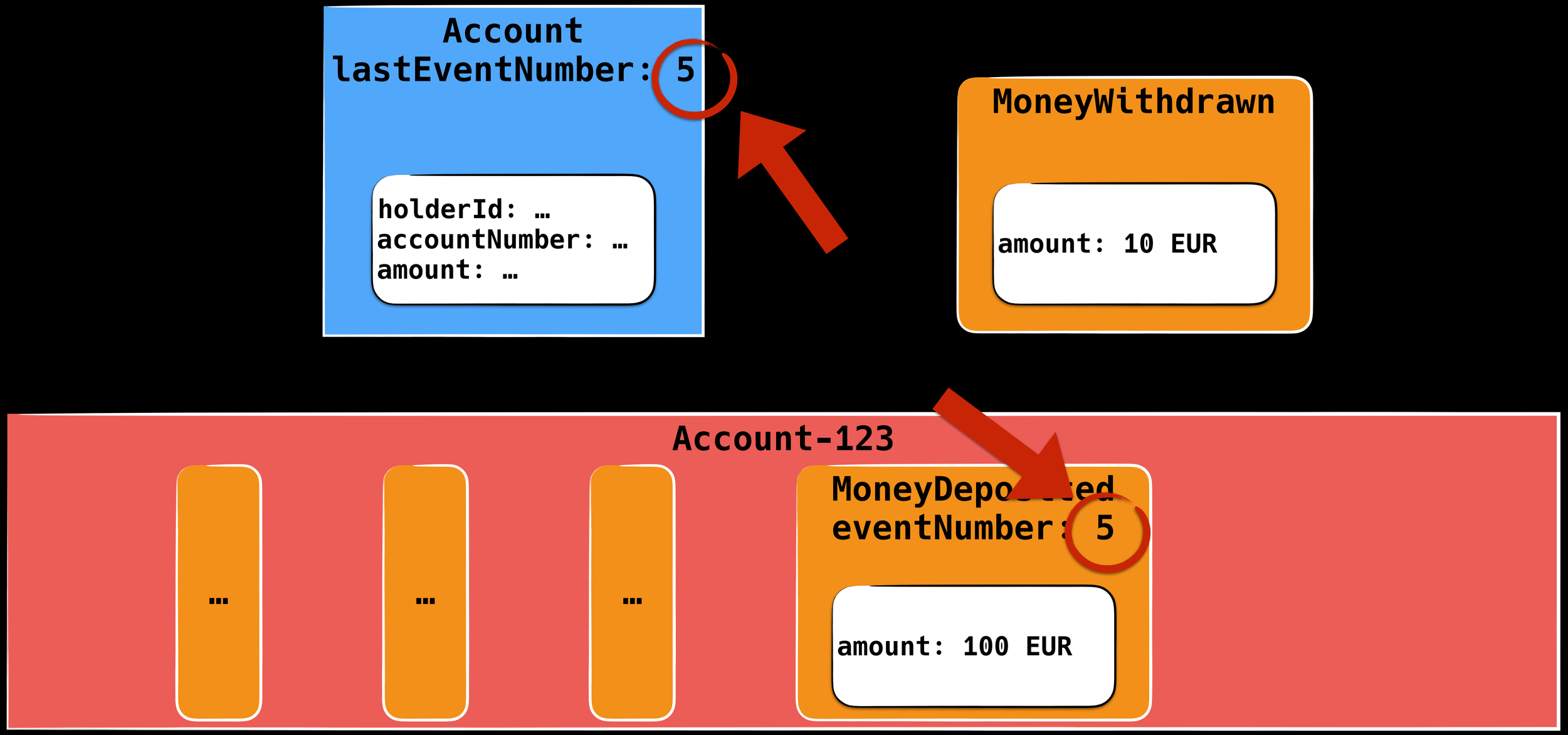
From Wikipedia, the free encyclopedia

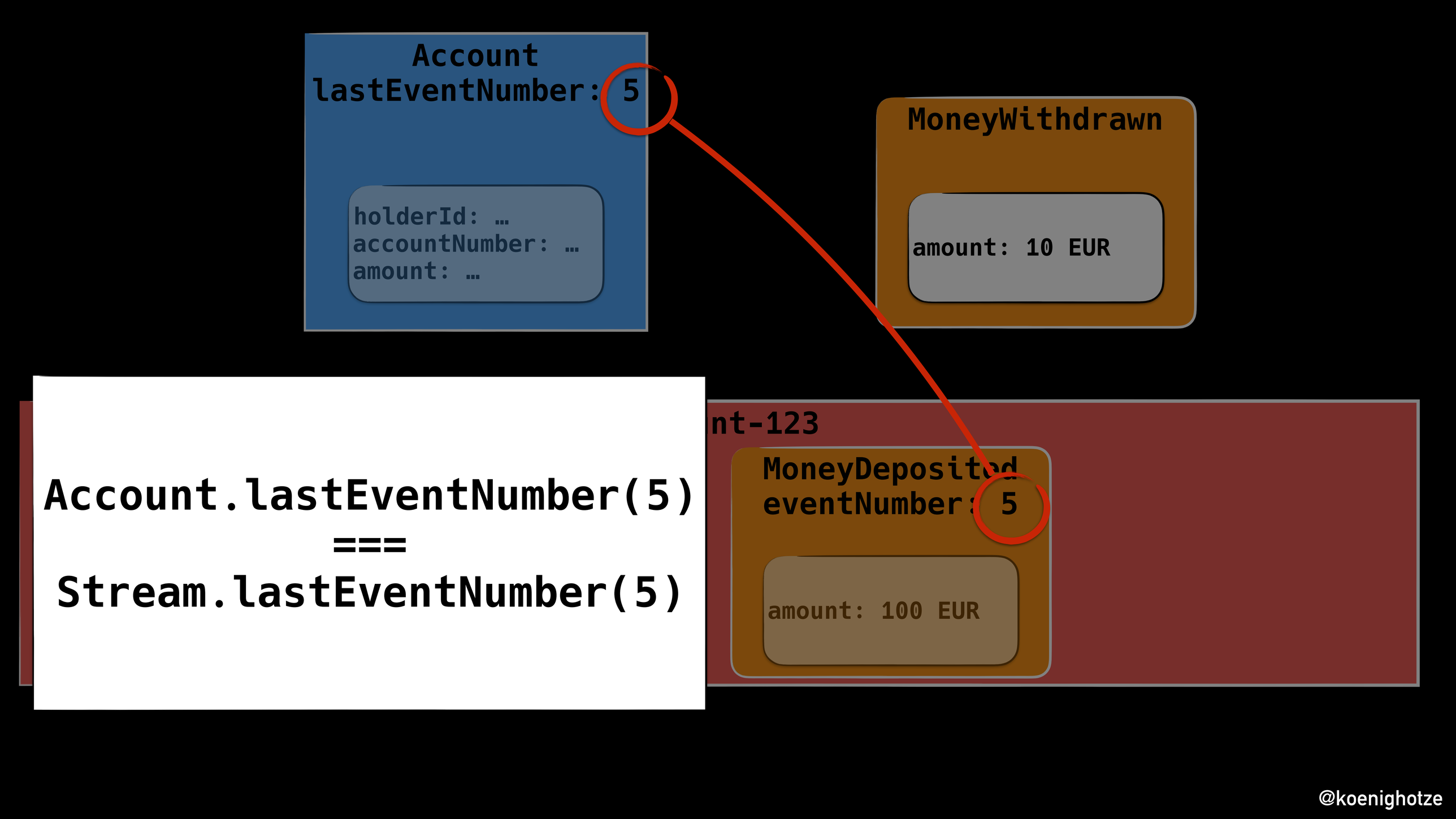
**…each transaction verifies that no other transaction has modified the data it has read…**

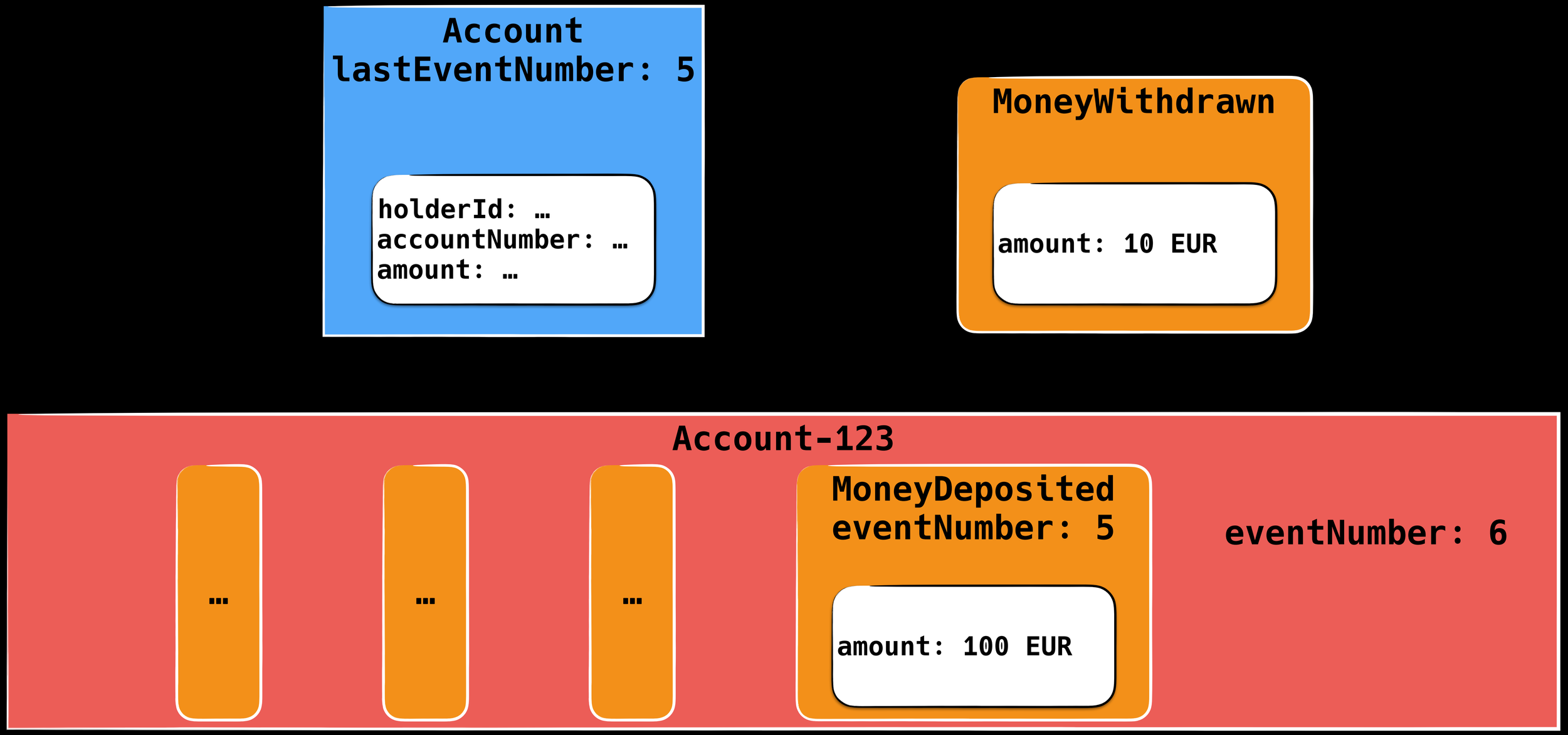
transaction has modified the data it has read. If the check reveals conflicting modifications, the committing transaction rolls back and can be restarted.[1] Optimistic concurrency control was first proposed by H.T. Kung and John T. Robinson.[2]

https://en.wikipedia.org/wiki/Optimistic_concurrency_control

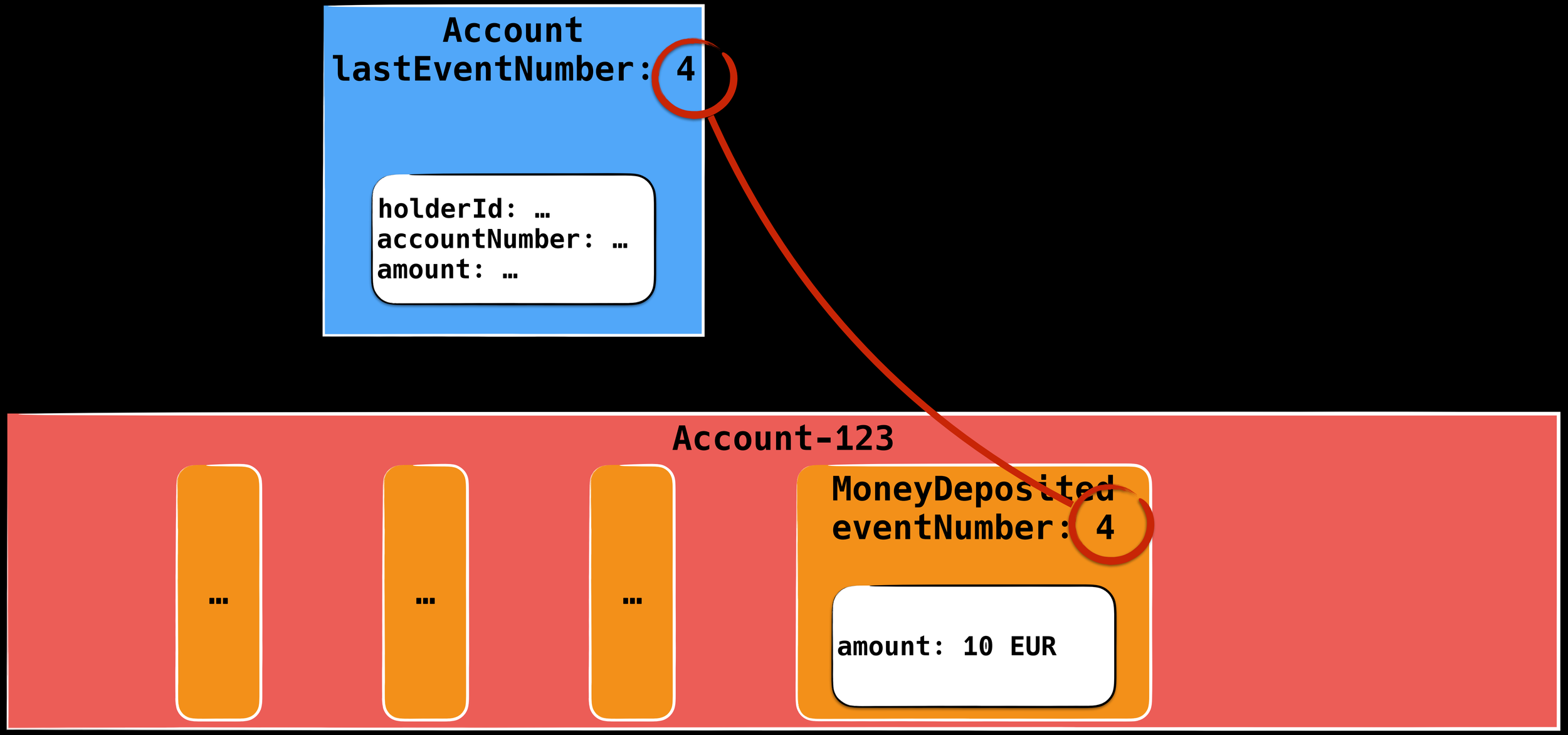
# The happy path

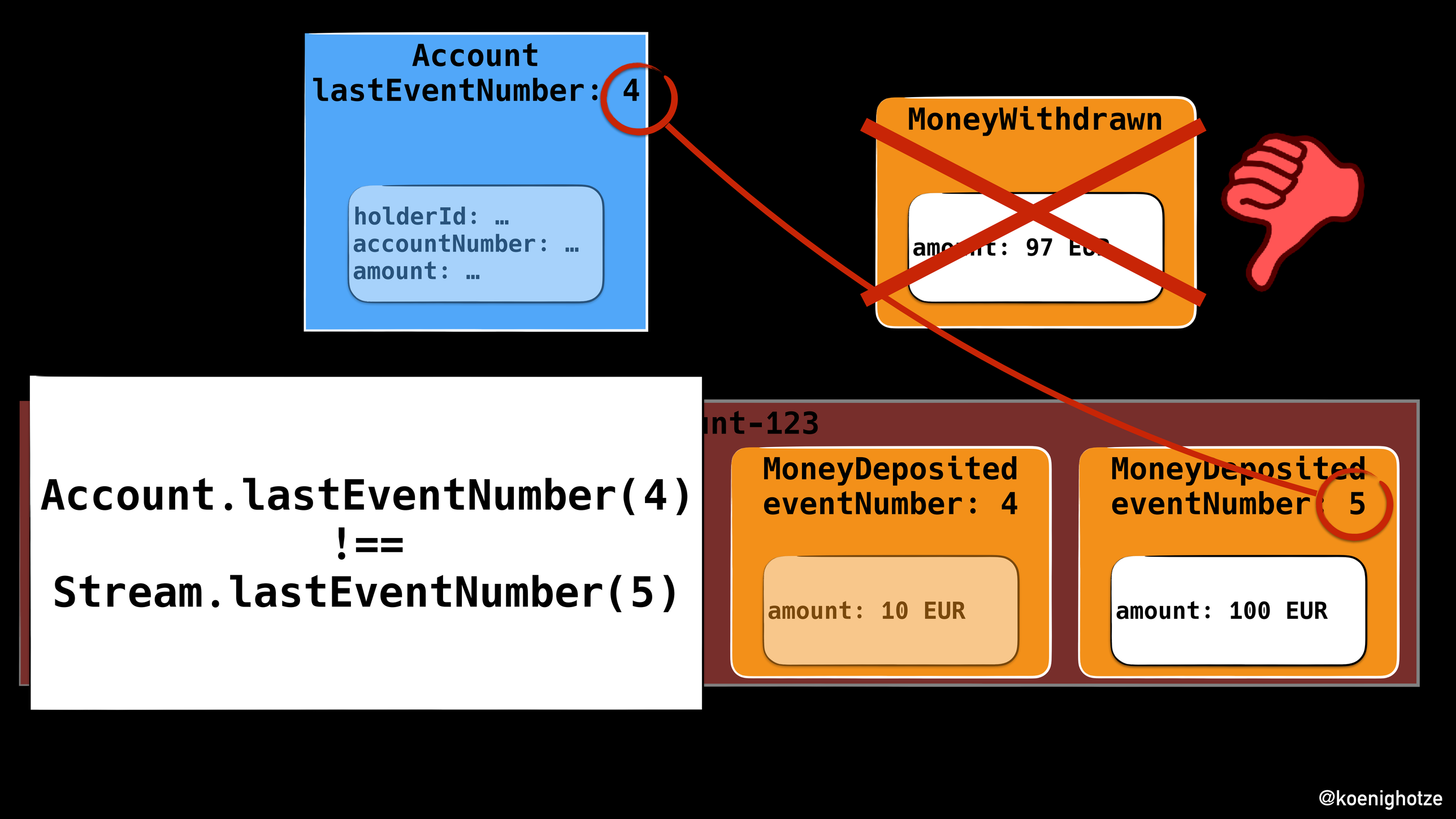

Account
lastEventNumber: 5
holderId: …
accountNumber: …
amount: …
MoneyWithdrawn
amount: 10 EUR
Account-123
…
…
…
MoneyDeposited
eventNumber: 5
amount: 100 EUR


Account
lastEventNumber: 5
holderId: …
accountNumber: …
amount: …
MoneyWithdrawn
amount: 10 EUR
nt-123
MoneyDeposited
eventNumber: 5
amount: 100 EUR
Account.lastEventNumber(5)
===
Stream.lastEventNumber(5)

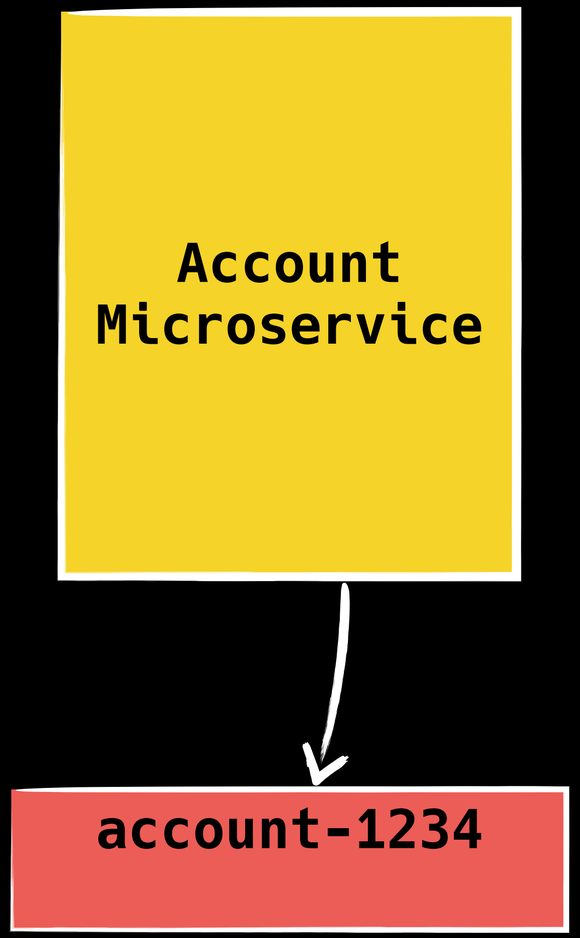

Account
lastEventNumber: 5
holderId: …
accountNumber: …
amount: …
MoneyWithdrawn
amount: 10 EUR
Account-123
…
…
…
MoneyDeposited
eventNumber: 5
amount: 100 EUR
eventNumber: 6

# The not-so-happy path

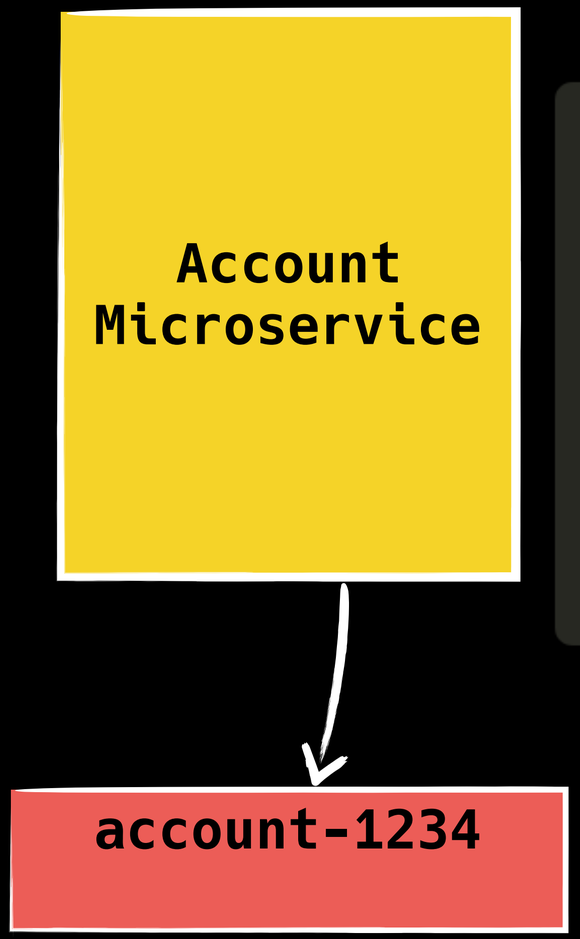

Account
lastEventNumber: 4
holderId: …
accountNumber: …
amount: …
Account-123
…
…
…
MoneyDeposited
eventNumber: 4
amount: 10 EUR


Account
lastEventNumber: 4
holderId: …
accountNumber: …
amount: …
MoneyWithdrawn
amount: 97 EUR
nt-123
MoneyDeposited
eventNumber: 4
amount: 10 EUR
MoneyDeposited
eventNumber: 5
amount: 100 EUR
Account.lastEventNumber(4)
!==
Stream.lastEventNumber(5)

# PUT /account/1234



# PUT /account/1234

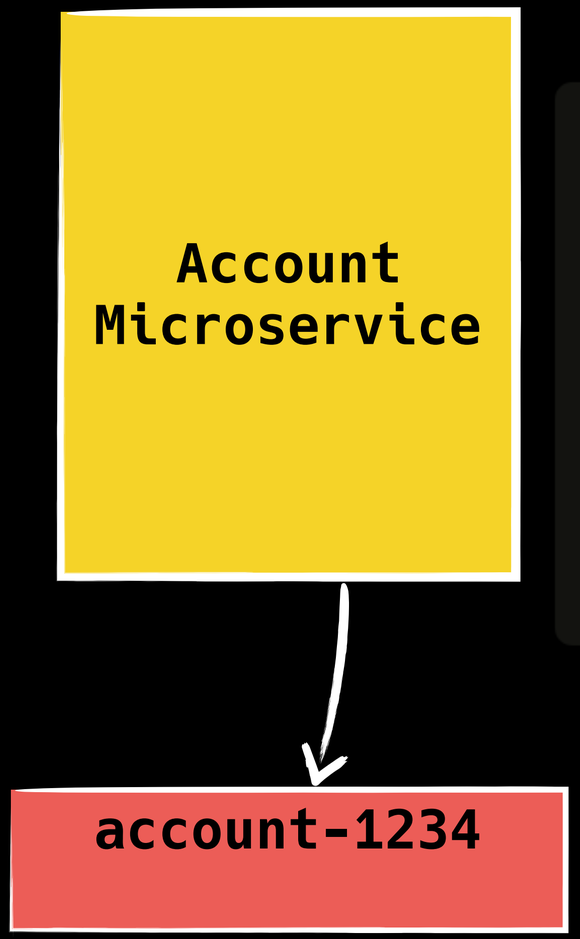



```
{ aggregate, lastVersionNumber } = readAggregateFromStream(...)

events = executeBusinessLogic(...)

emitEvents('account', '1234', events, lastVersionNumber)
```

# PUT /account/1234

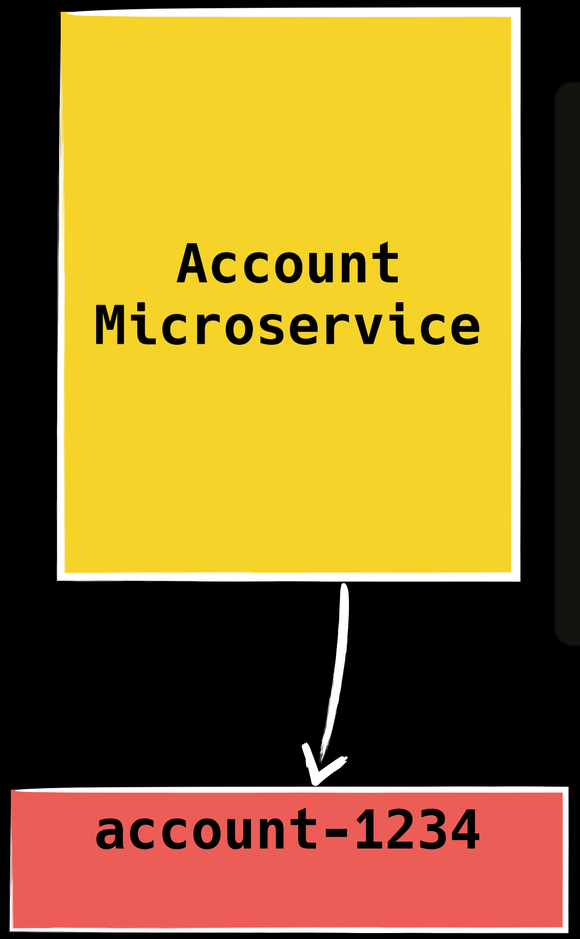



```
{ aggregate, lastVersionNumber } = readAggregateFromStream(...)

events = executeBusinessLogic(...)

emitEvents('account', '1234', events, lastVersionNumber)
```

# PUT /account/1234

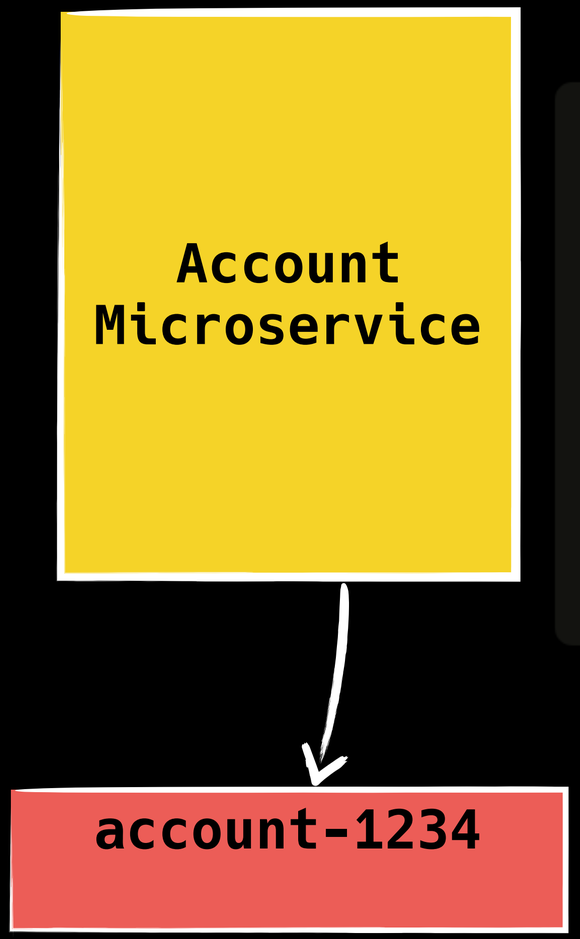



```
{ aggregate, lastVersionNumber } = readAggregateFromStream(...)

events = executeBusinessLogic(...)

emitEvents('account', '1234', events, lastVersionNumber)
```

# PUT /account/1234




```
{ aggregate, lastVersionNumber } = readAggregateFromStream(...)

events = executeBusinessLogic(...)

emitEvents('account', '1234', events, lastVersionNumber)
```

# And Kafka?

Kafka / KAFKA-2260

# Allow specifying expected offset on produce

**Details**

| | | | |
|---|---|---|---|
| Type: | Improvement | Status: | OPEN |
| Priority: | Minor | Resolution: | Unresolved |
| Affects Version/s: | None | Fix Version/s: | None |
| Component/s: | producer | | |
| Labels: | None | | |

**Description**

I'd like to propose a change that adds a simple CAS-like mechanism to the Kafka producer. This update has a small footprint, but enables a bunch of interesting uses in stream processing or as a commit log for process state.

---

Andy Bryant added a comment - 27/Jul/18 04:14

👍 Would prove very handy in event source based designs

---

Russell Ferriday added a comment - 9 hours ago

This would enable full-on eventsourcing on Kafka, without having to restrict to single-thread designs.
One example of a great (>250 github star) FOSS project being held back by this:

https://github.com/johnbywater/eventsourcing/issues/108

Can we see this soon?

Kafka / KAFKA-2260

# Allow specifying expected offset on produce

**Details**

| | | | |
|---|---|---|---|
| Type: | Improvement | Status: | OPEN |
| Priority: | Minor | Resolution: | Unresolved |
| Affects Version/s: | None | Fix Version/s: | None |


Andy Bryant added a comment - 27/Jul/18 04:14

Would prove very handy in event source based designs

Russell Ferriday added a comment - 9 hours ago

This would enable full-on eventsourcing on Kafka, without having to restrict to single-thread designs.
One example of a great (>250 github star) FOSS project being held back by this:

https://github.com/johnbywater/eventsourcing/issues/108

Can we see this soon?

Kafka / KAFKA-2260

# Allow specifying expected offset on produce

**Details**

| | | | |
|---|---|---|---|
| Type: | Improvement | Status: | OPEN |
| Priority: | Minor | Resolution: | Unresolved |
| Affects Version/s: | None | Fix Version/s: | None |
| Component/s: | producer | | |

This would enable full-on eventsourcing on Kafka, without having to restrict to single-thread designs.
One example of a great (>250 github star) FOSS project being held back by this:

Andy Bryant added a comment - 27/Jul/18 04:14

Would prove very handy in event source based designs

Russell Ferriday added a comment - 9 hours ago

This would enable full-on eventsourcing on Kafka, without having to restrict to single-thread designs.
One example of a great (>250 github star) FOSS project being held back by this:

https://github.com/johnbywater/eventsourcing/issues/108

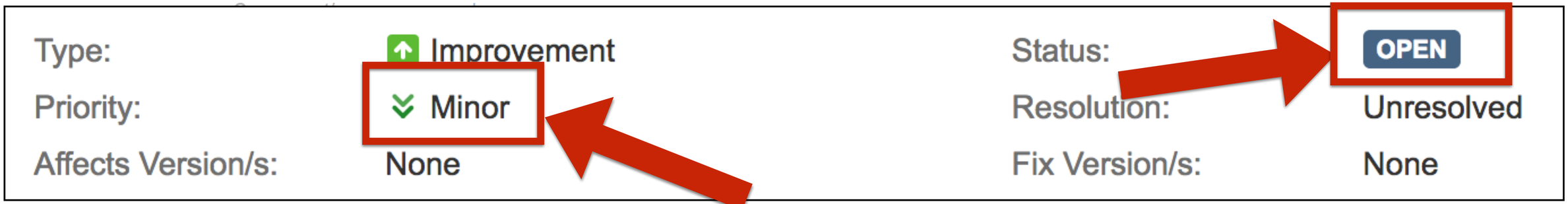
Can we see this soon?

Kafka / KAFKA-2260

# Allow specifying expected offset on produce

**Details**

| | | | |
|---|---|---|---|
| Type: | Improvement | Status: | OPEN |
| Priority: | Minor | Resolution: | Unresolved |
| Affects Version/s: | None | Fix Version/s: | None |




Andy Bryant added a comment - 27/Jul/18 04:14

Would prove very handy in event source based designs

Russell Ferriday added a comment - 9 hours ago

This would enable full-on eventsourcing on Kafka, without having to restrict to single-thread designs.
One example of a great (>250 github star) FOSS project being held back by this:

https://github.com/johnbywater/eventsourcing/issues/108

Can we see this soon?

Scalability without locks

Consistency

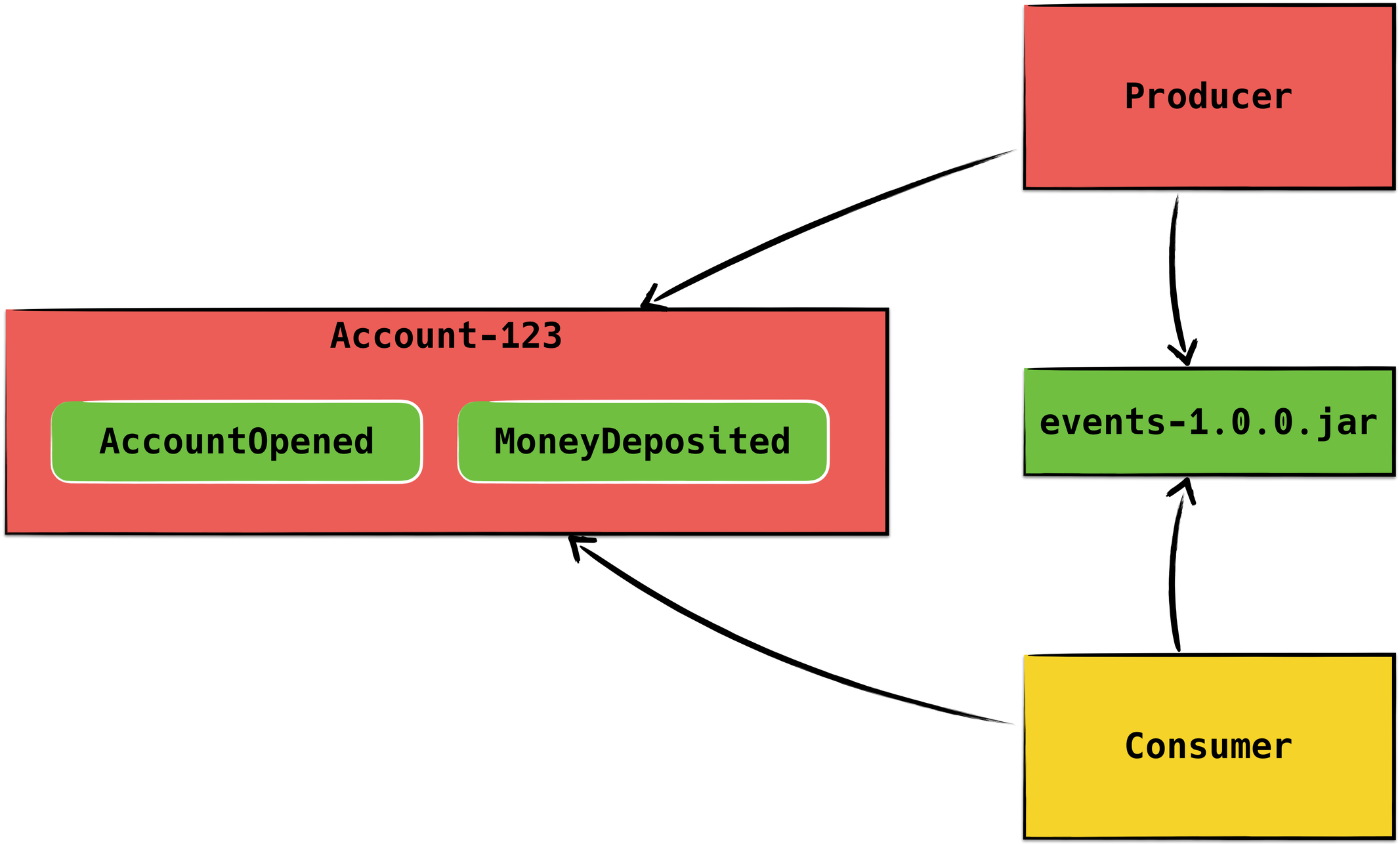
Design choice

Super-simple programming logic

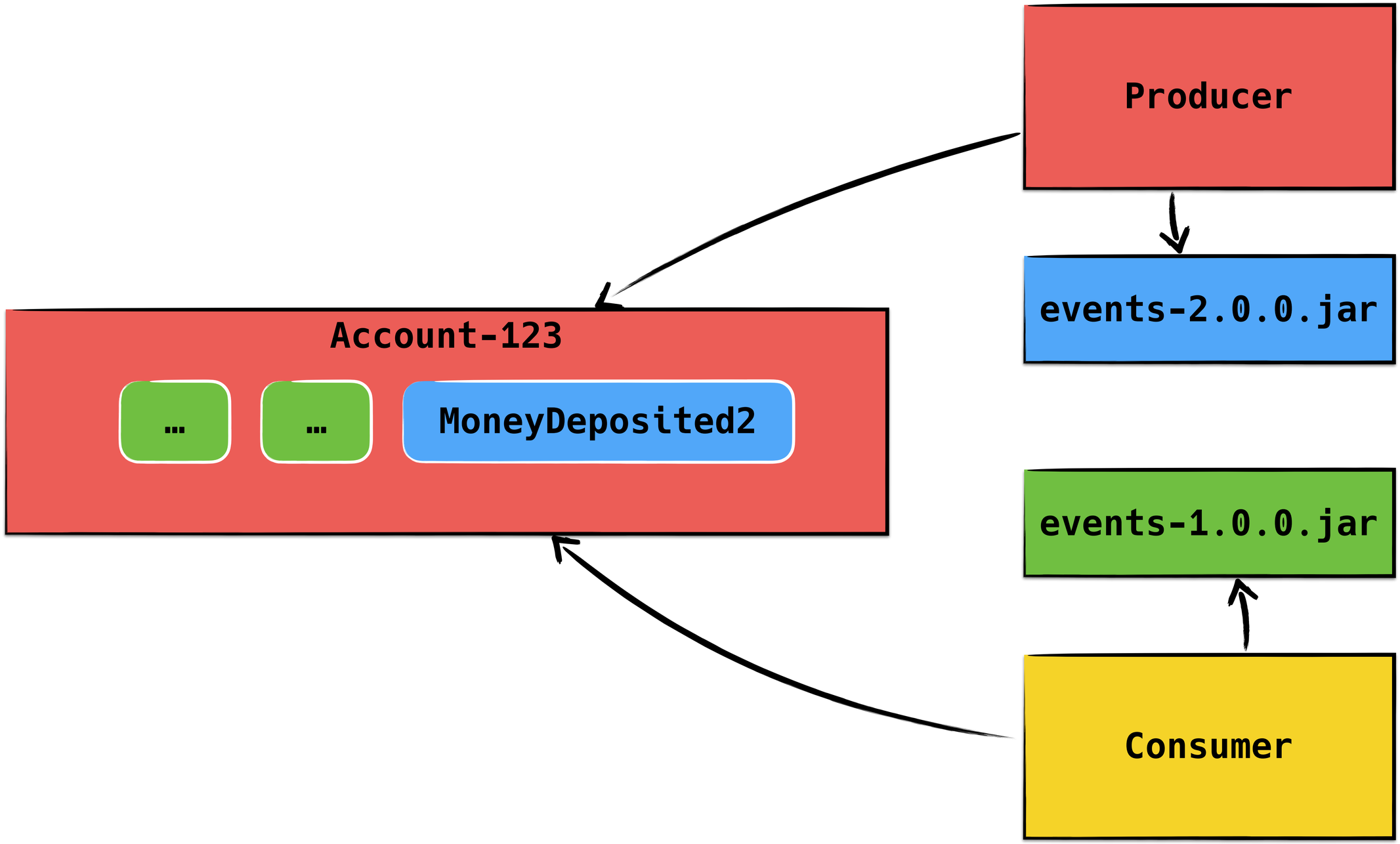

VERSIONS, UP-FRONT-DESIGN AND
BREAKING THINGS DOWN THE ROAD
@koenighotze

# How can we deal with versions without going crazy?

# JUST USE SEMANTIC VERSIONING AND TYPED EVENTS

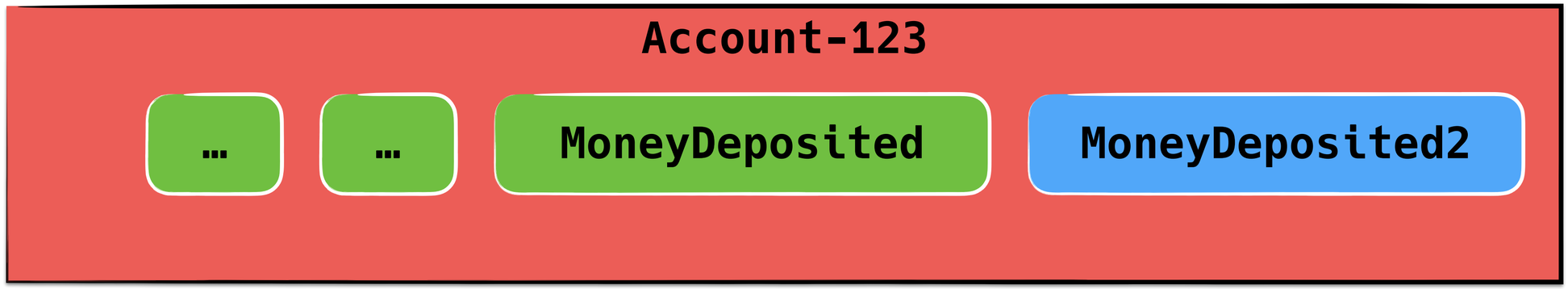

Producer
Account-123
AccountOpened
MoneyDeposited
events-1.0.0.jar
Consumer

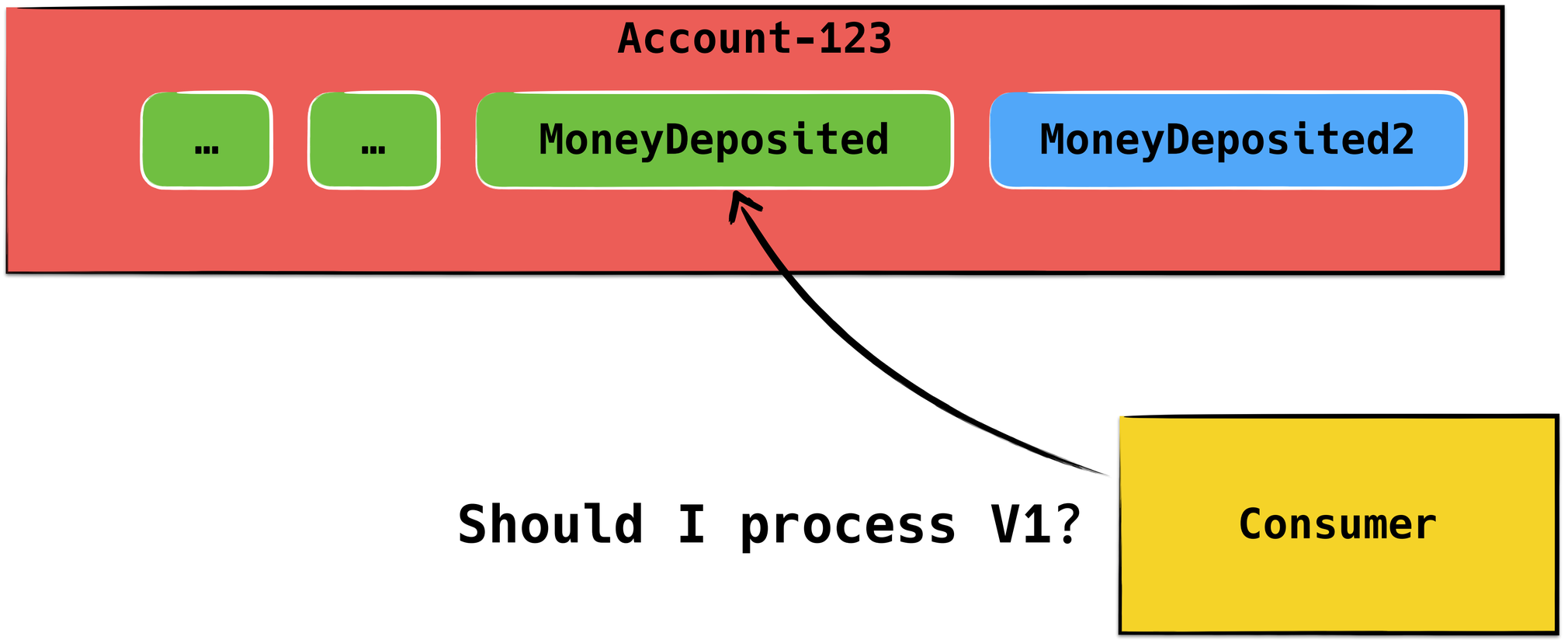
# ...THEN CHANGE SOME EVENT TYPES

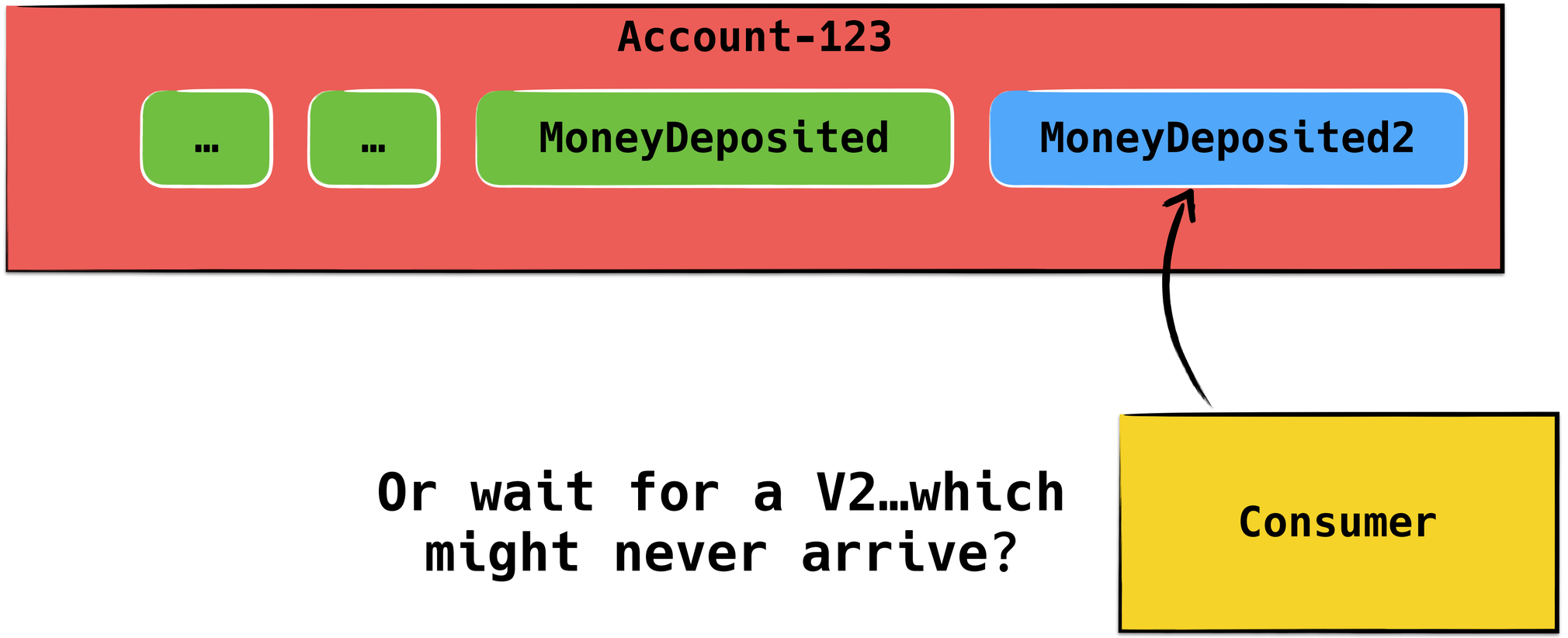

Producer
events-2.0.0.jar
Account-123
...
...
MoneyDeposited2
events-1.0.0.jar
Consumer


public class InvalidClassException extends ObjectStreamException {
...
}

# Gosh...just apply "Double Write"


Account-123
…
…
MoneyDeposited
MoneyDeposited2


Account-123
…
…
MoneyDeposited
MoneyDeposited2
Should I process V1?
Consumer




Or wait for a V2…which might never arrive?

MoneyDeposited_v1
MoneyDeposited_v2

...

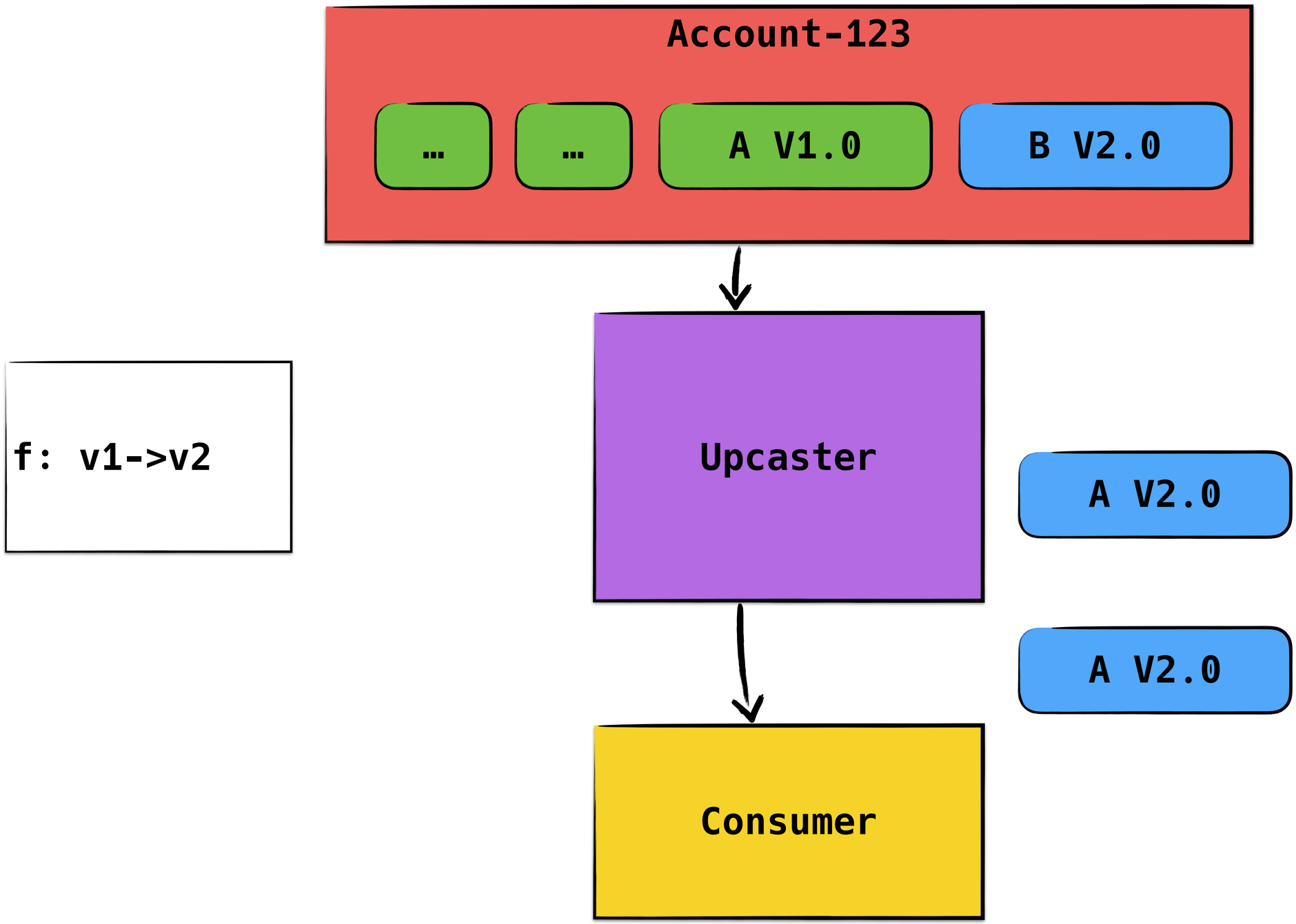
MoneyDeposited_v100

MoneyDeposited_v1Handler
MoneyDeposited_v2Handler

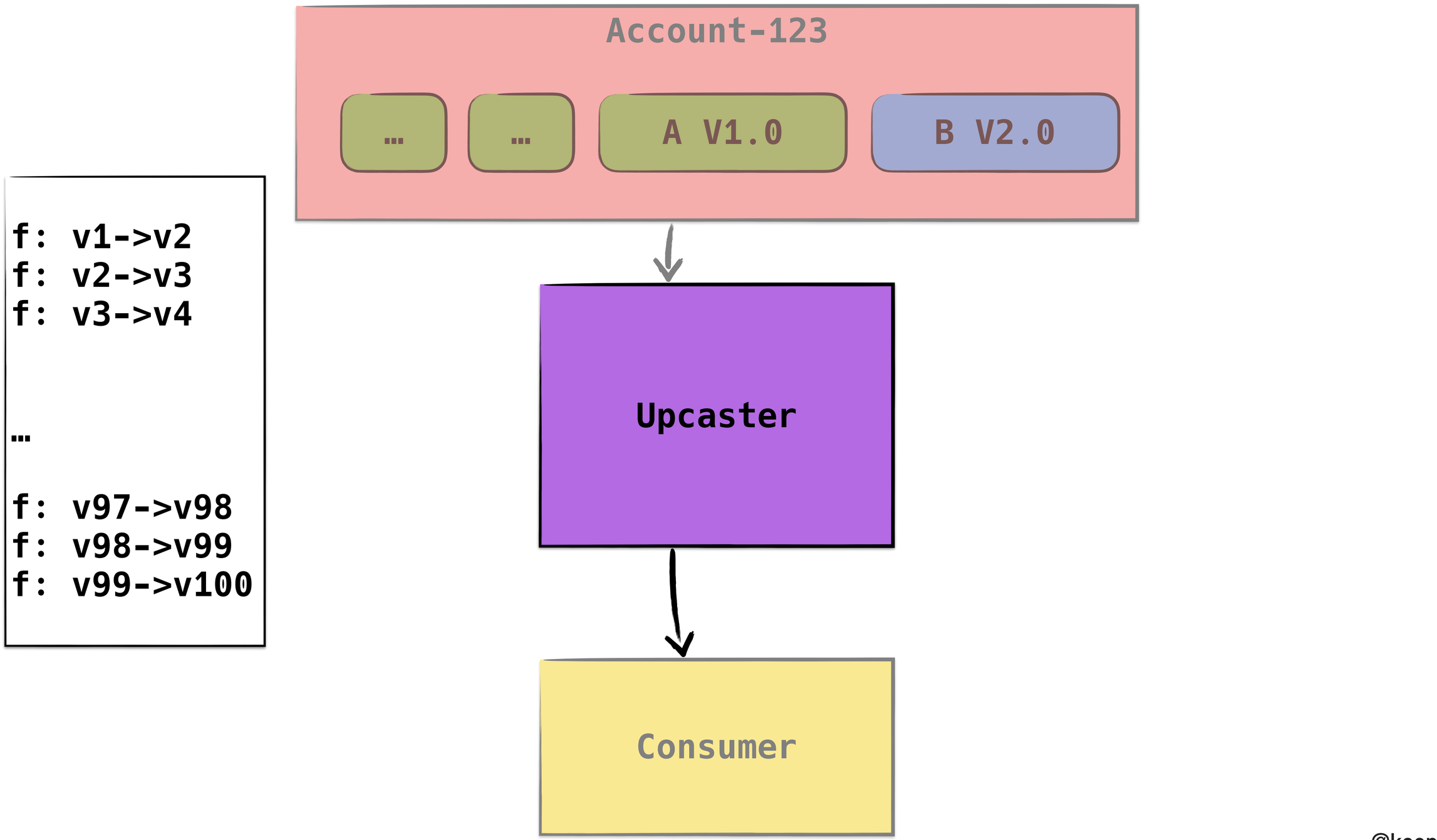
...

MoneyDeposited_v100Handler

# Upcaster


Account-123
...
...
A V1.0
B V2.0
f: v1->v2
Upcaster
A V2.0
A V2.0
Consumer

# 5 MONTHS LATER...

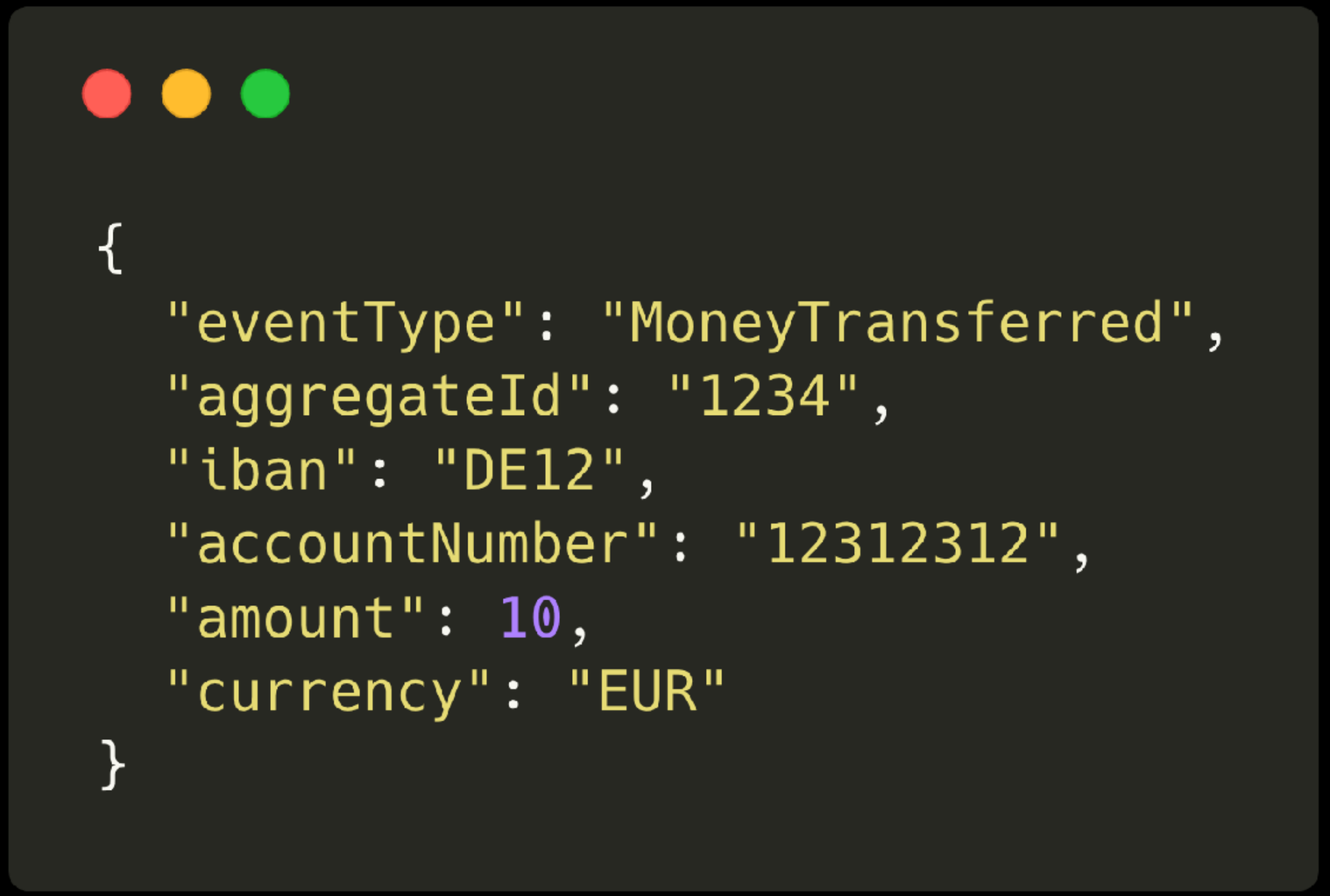

Account-123
…
…
A V1.0
B V2.0
f: v1->v2
f: v2->v3
f: v3->v4
…
f: v97->v98
f: v98->v99
f: v99->v100
Upcaster
Consumer

# Good luck maintaining that monster

# Incompatible event changes indicate a new event

Prefer simple, text-based, human readable events

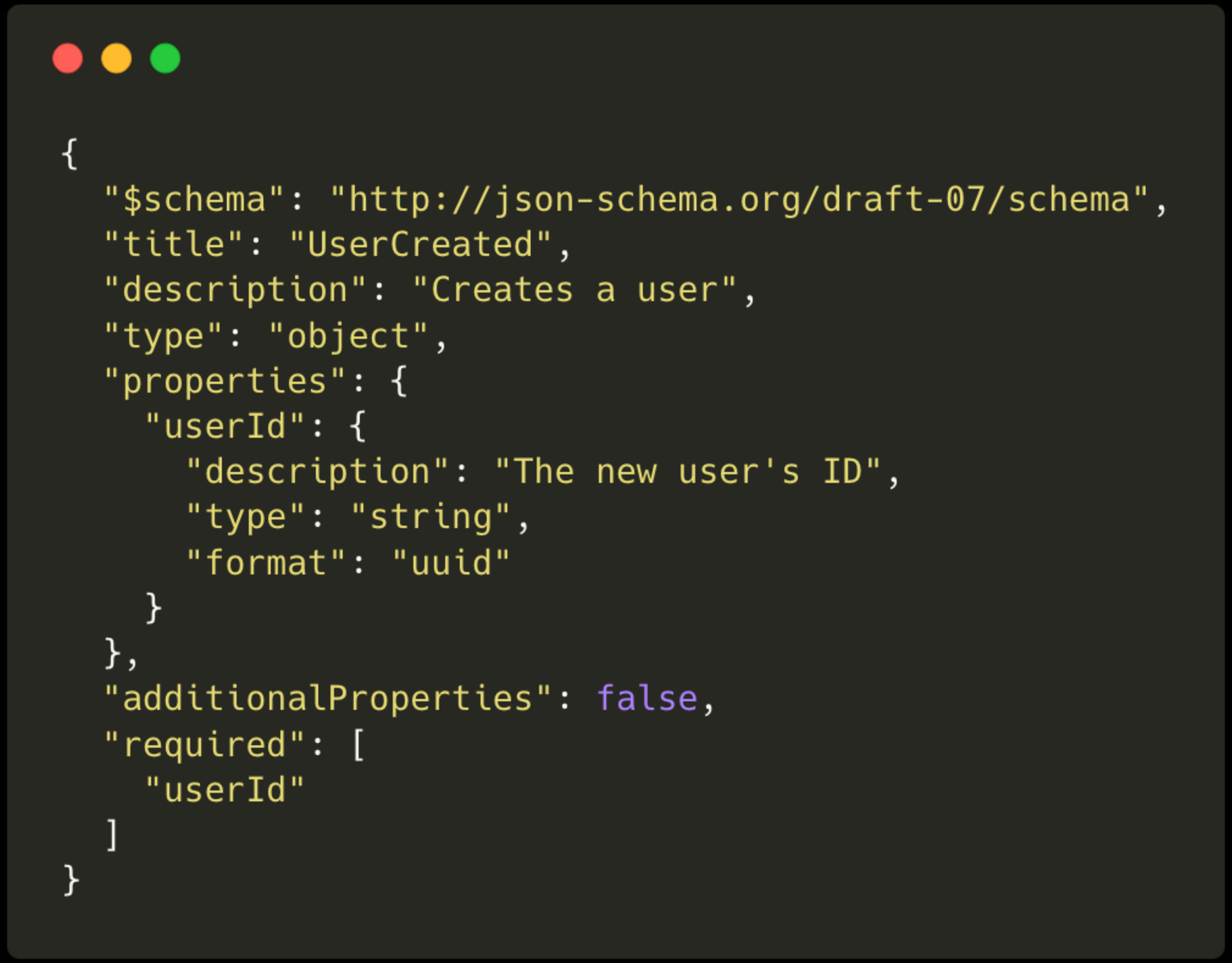
# Fancy speak for JSON


{
"eventType": "MoneyTransferred",
"aggregateId": "1234",
"iban": "DE12",
"accountNumber": "12312312",
"amount": 10,
"currency": "EUR"
}

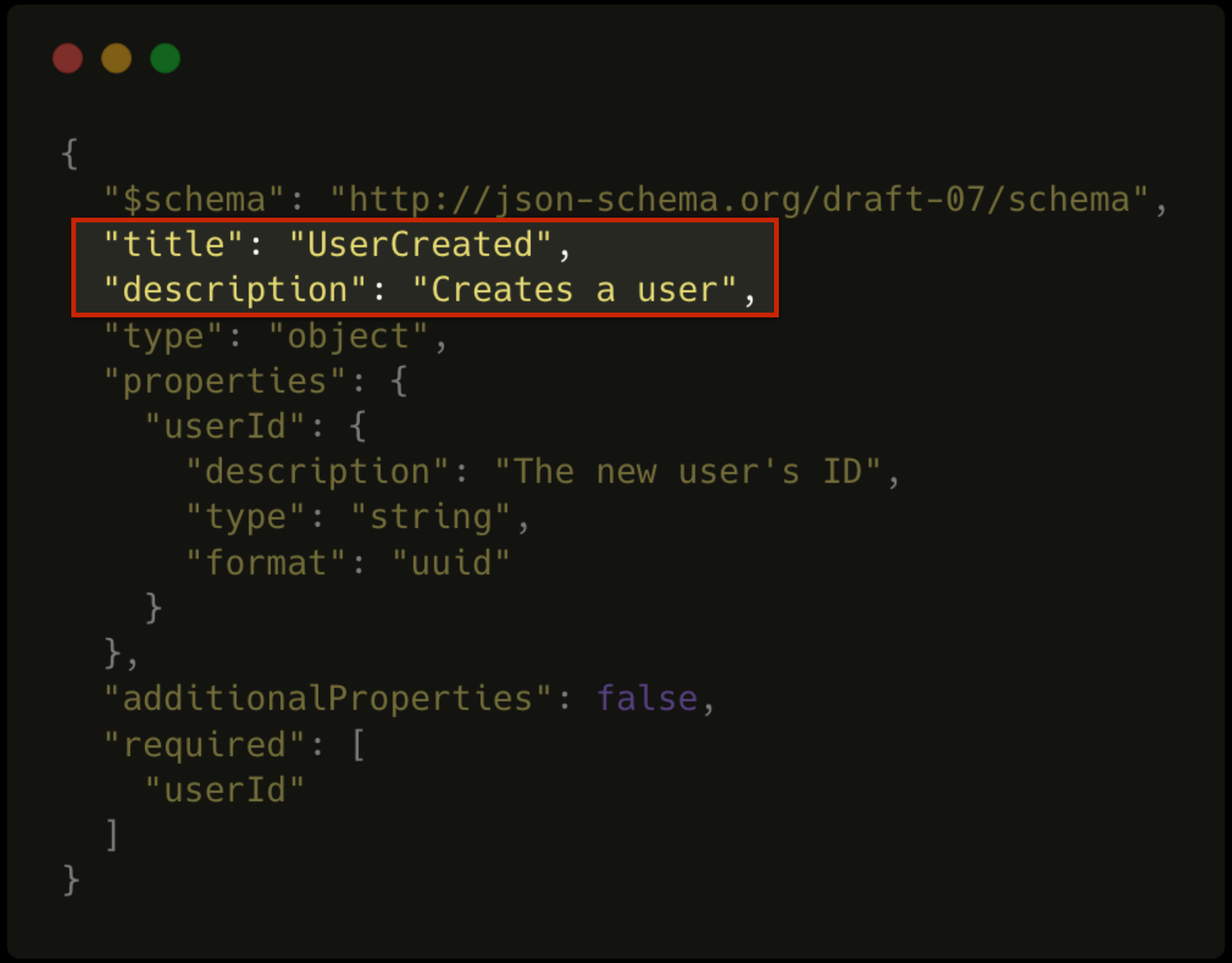
# And correctness?

# String-ly typed events work really well

# Weak schema to the rescue



```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "title": "UserCreated",
  "description": "Creates a user",
  "type": "object",
  "properties": {
    "userId": {
      "description": "The new user's ID",
      "type": "string",
      "format": "uuid"
    }
  },
  "additionalProperties": false,
  "required": [
    "userId"
  ]
}
```



```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "title": "UserCreated",
  "description": "Creates a user",
  "type": "object",
  "properties": {
    "userId": {
      "description": "The new user's ID",
      "type": "string",
      "format": "uuid"
    }
  },
  "additionalProperties": false,
  "required": [
    "userId"
  ]
}
```

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "title": "UserCreated",
  "description": "Creates a user",
  "type": "object",
  "properties": {
    "userId": {
      "description": "The new user's ID",
      "type": "string",
      "format": "uuid"
    }
  },
  "additionalProperties": false,
  "required": [
    "userId"
  ]
}
```

```
assertIsValid(eventData, ajv.compile(schema))
event = newEvent(
    {
      aggregateId,
      aggregateType,
      eventData
    }
  )
```

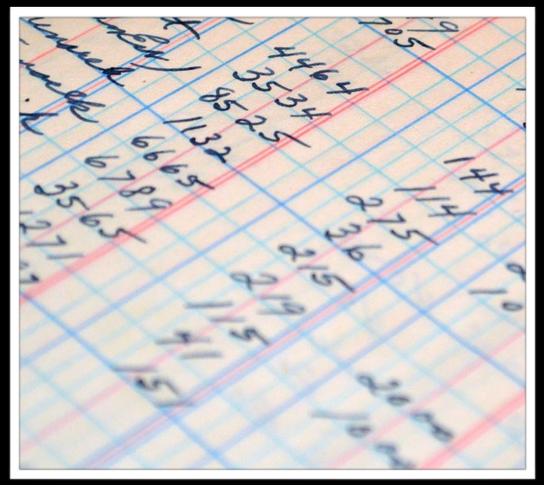
```
assertIsValid(eventData, ajv.compile(schema))
event = newEvent(
    {
      aggregateId,
      aggregateType,
      eventData
    }
  )
```

# Schema as a description not as a contract

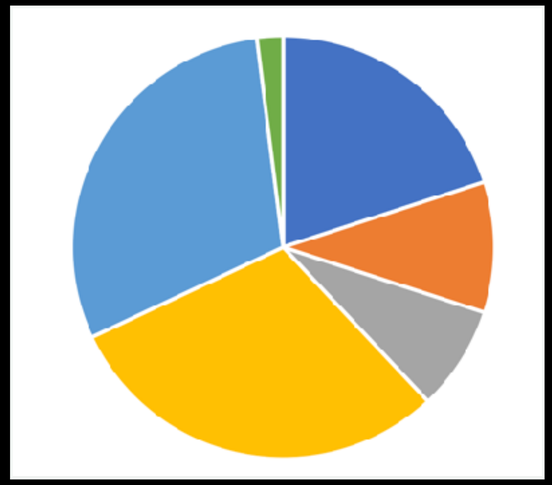
# Schema-on-Read

# Reusing event data?

Transactionledger Microservice



Budget Planer Microservice



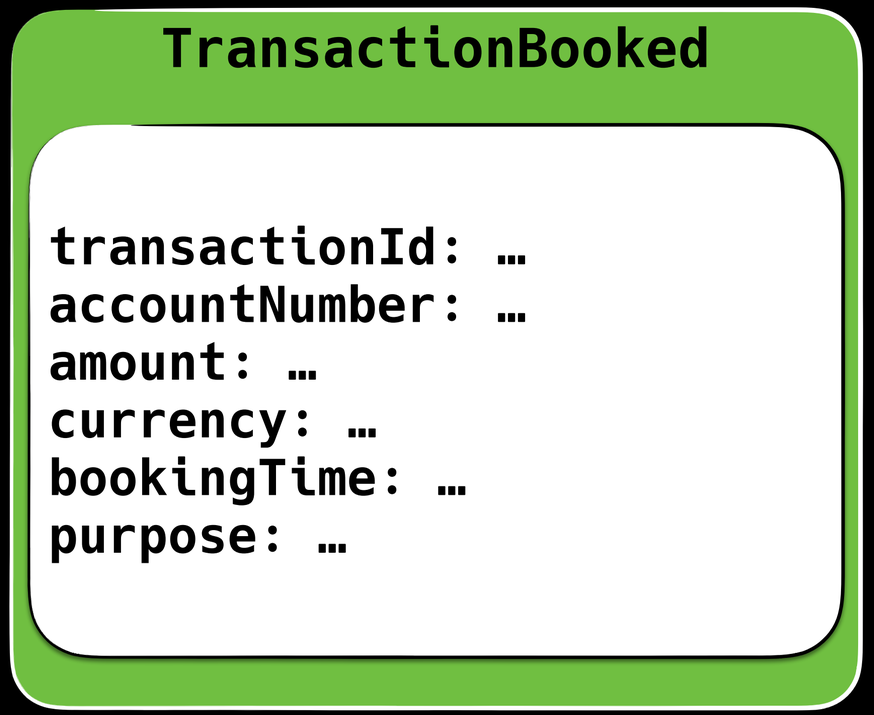
Transactionledger Microservice

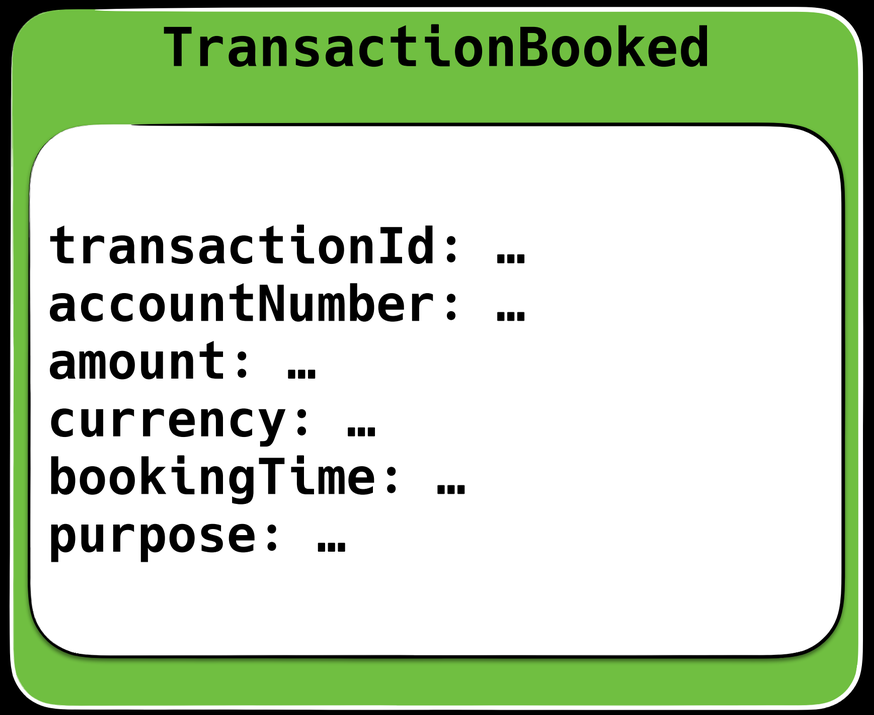
TransactionBooked

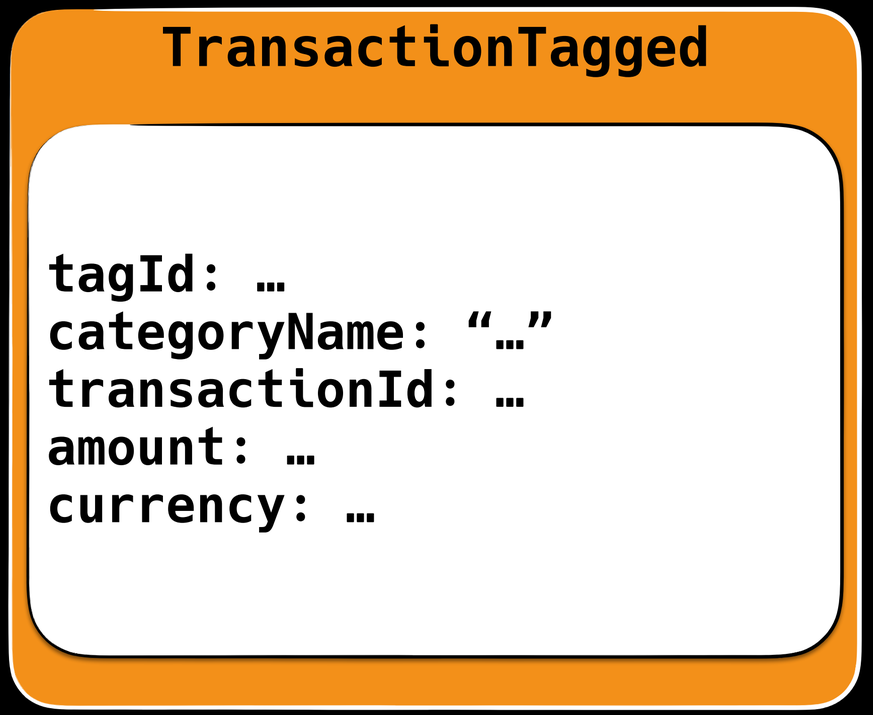
Budget Planer Microservice

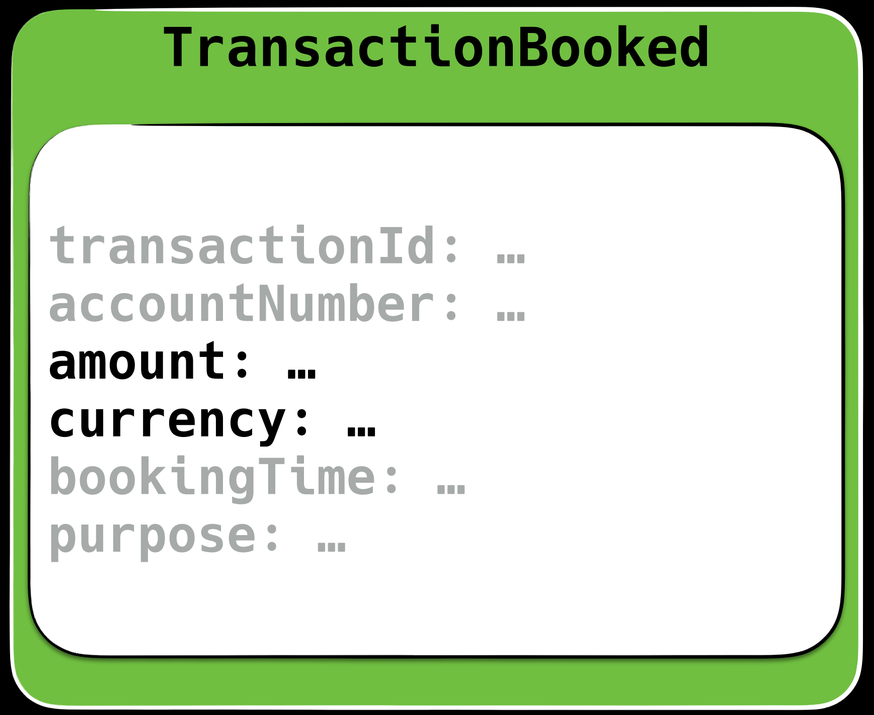
TransactionTagged

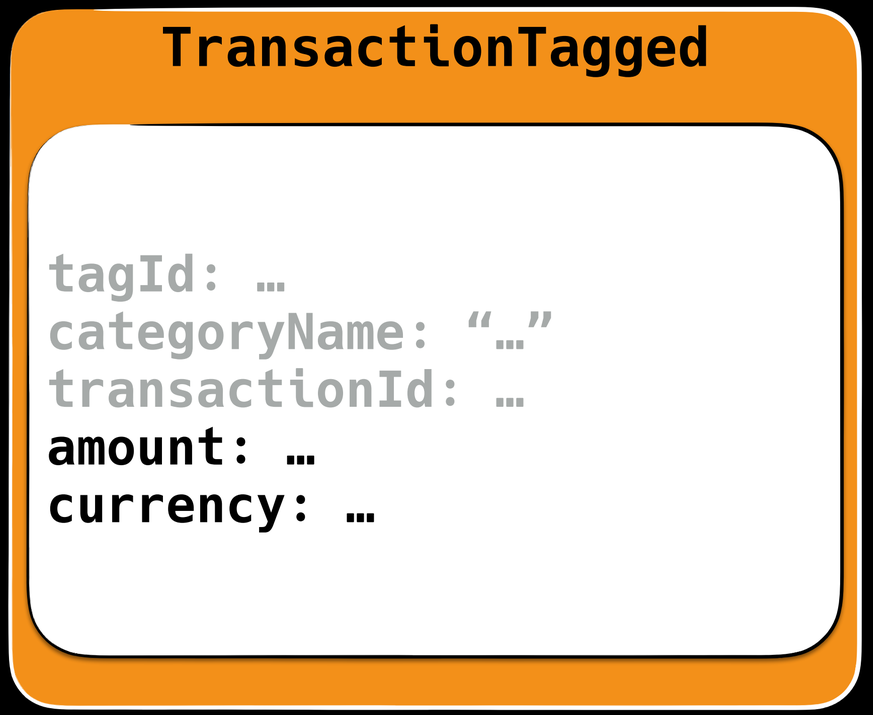
JUST COPY DATA INTO DIFFERENT EVENTS, JUST SO CONVENIENT


TransactionBooked
transactionId: …
accountNumber: …
amount: …
currency: …
bookingTime: …
purpose: …

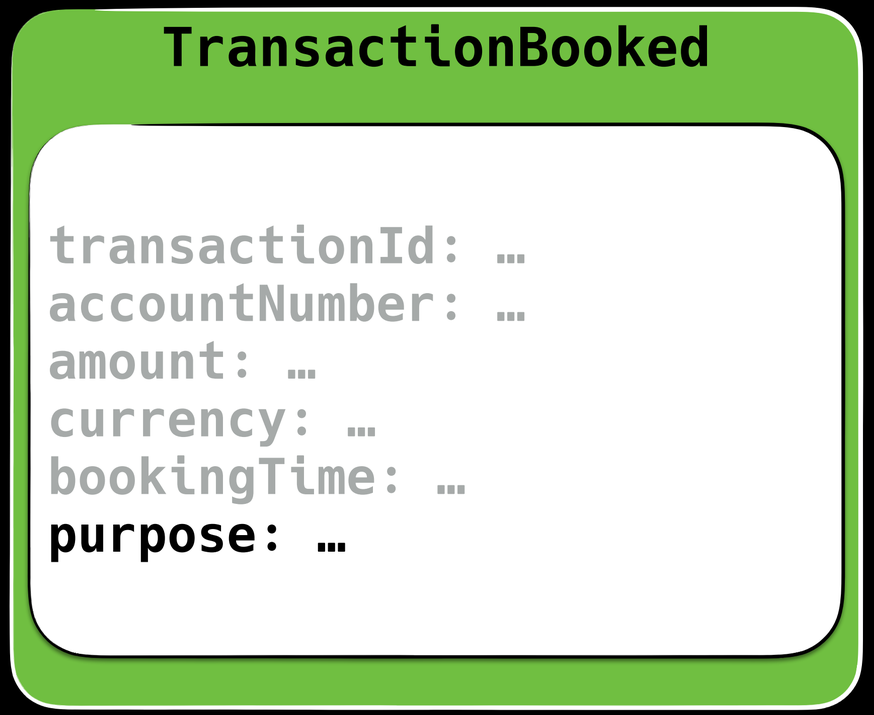

TransactionBooked
transactionId: …
accountNumber: …
amount: …
currency: …
bookingTime: …
purpose: …

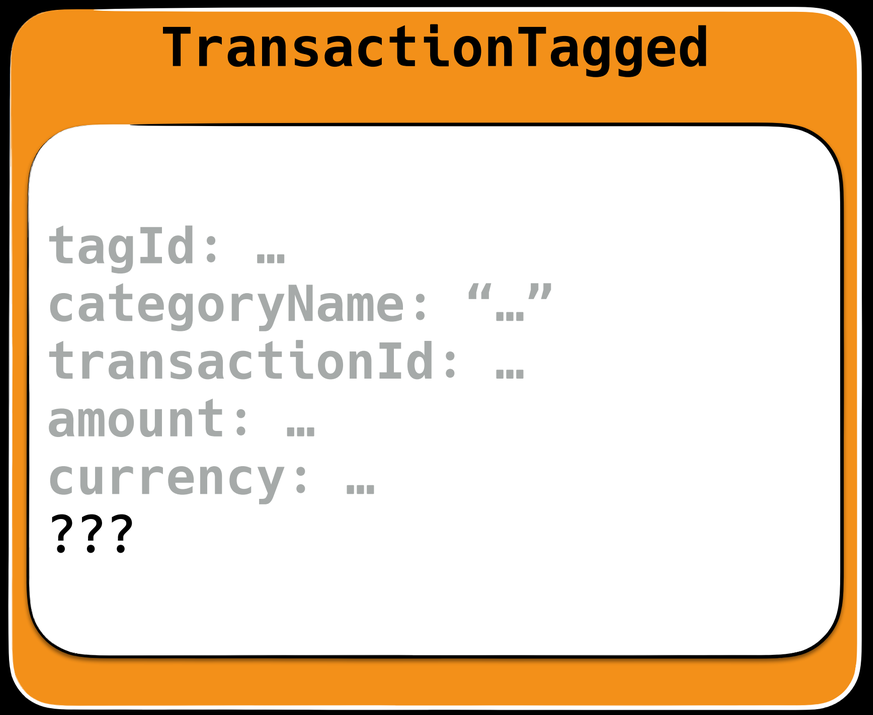

TransactionTagged
tagId: …
categoryName: “…”
transactionId: …
amount: …
currency: …


TransactionBooked
transactionId: …
accountNumber: …
amount: …
currency: …
bookingTime: …
purpose: …


TransactionTagged
tagId: …
categoryName: “…”
transactionId: …
amount: …
currency: …

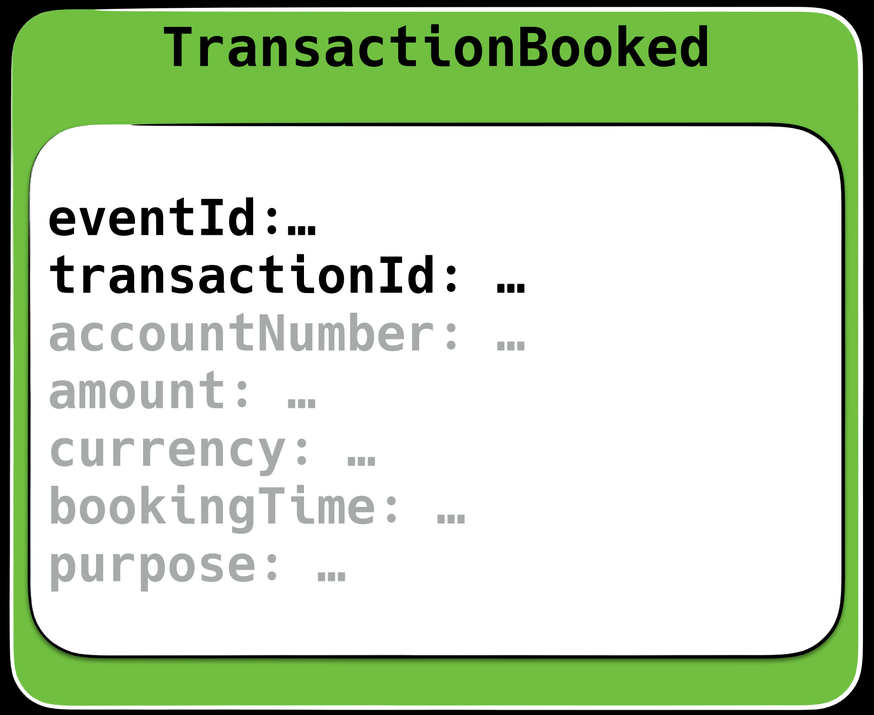

But I need to display the transaction purpose, too
Budget Planer Microservice

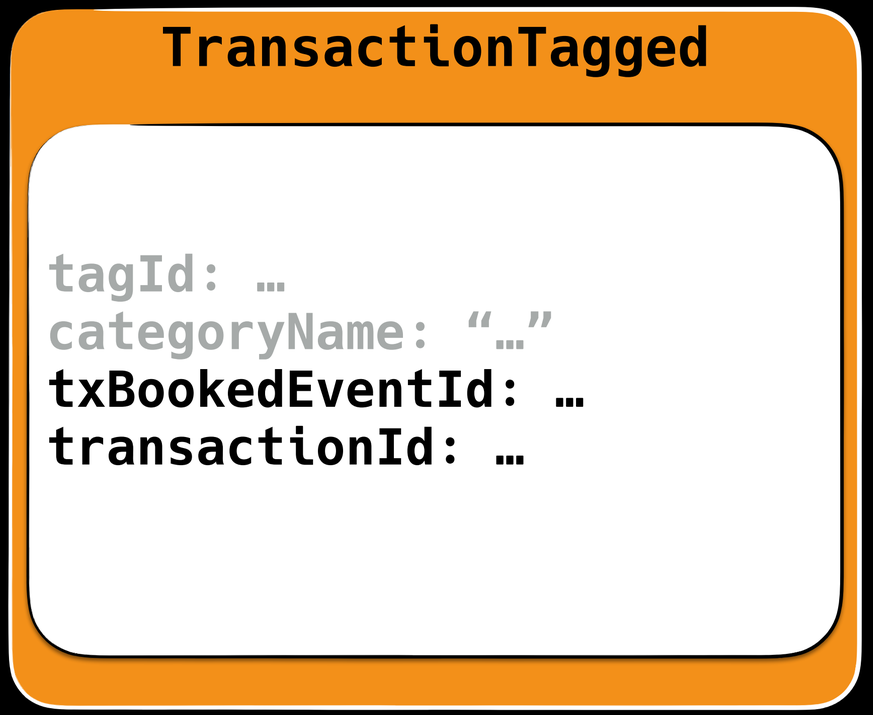

TransactionBooked
transactionId: …
accountNumber: …
amount: …
currency: …
bookingTime: …
purpose: …

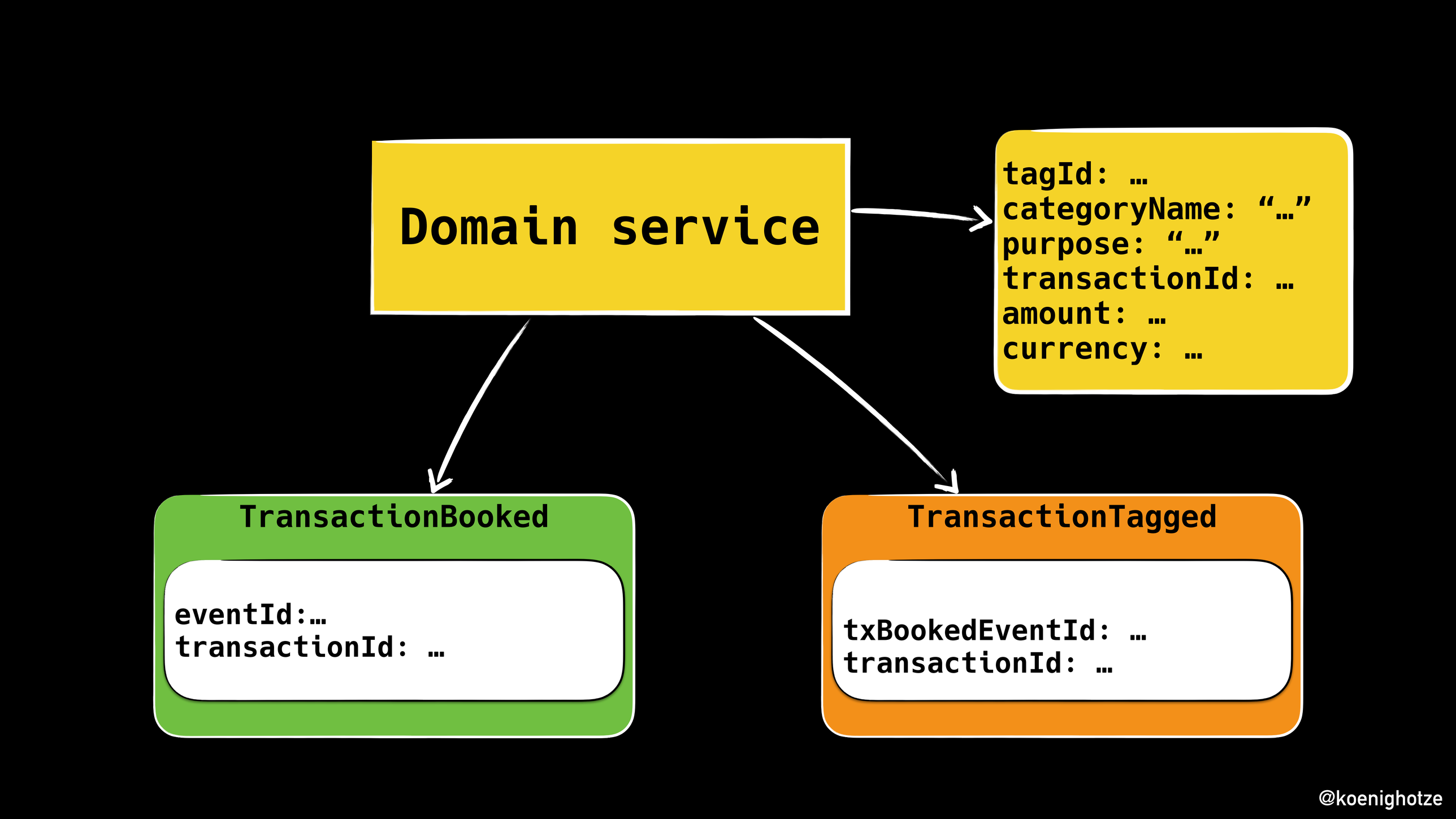

TransactionTagged
tagId: …
categoryName: “…”
transactionId: …
amount: …
currency: …
???

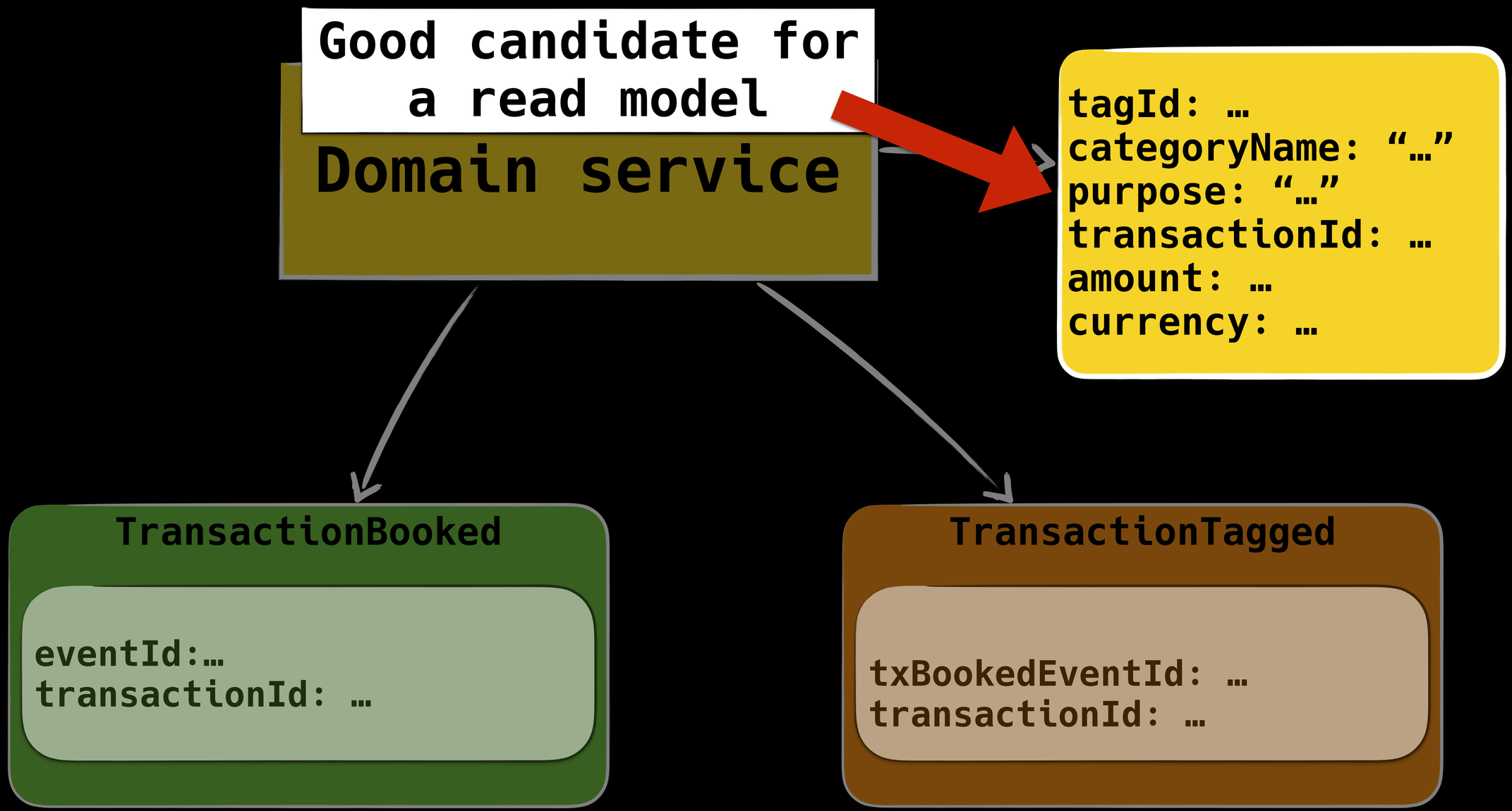
# The lossy event

Only reference aggregates via their root id or event ids


TransactionBooked
eventId:…
transactionId: …
accountNumber: …
amount: …
currency: …
bookingTime: …
purpose: …


TransactionTagged
tagId: …
categoryName: “…”
txBookedEventId: …
transactionId: …


Domain service
tagId: …
categoryName: “…”
purpose: “…”
transactionId: …
amount: …
currency: …
TransactionBooked
eventId:…
transactionId: …
TransactionTagged
txBookedEventId: …
transactionId: …


Good candidate for a read model
Domain service
tagId: …
categoryName: “…”
purpose: “…”
transactionId: …
amount: …
currency: …
TransactionBooked
eventId:…
transactionId: …
TransactionTagged
txBookedEventId: …
transactionId: …

Don't copy parts of an event

Prefer building use case specific projections

# HOW DO I HANDLE LARGE STREAMS?

# How can you handle event data over a long period of time?

# YOU DON'T

Just create a snapshot of the stream

# Year's end procedure

**Year end - also known as an accounting reference date - is the completion of an accounting period. At this time, businesses need to carry out specific procedures to close their books.**

https://debitoor.com/dictionary/year-end

Year end - also known as an accounting reference date - is the completion of an accounting period. At this time, **businesses need to carry out specific procedures to close their books**.

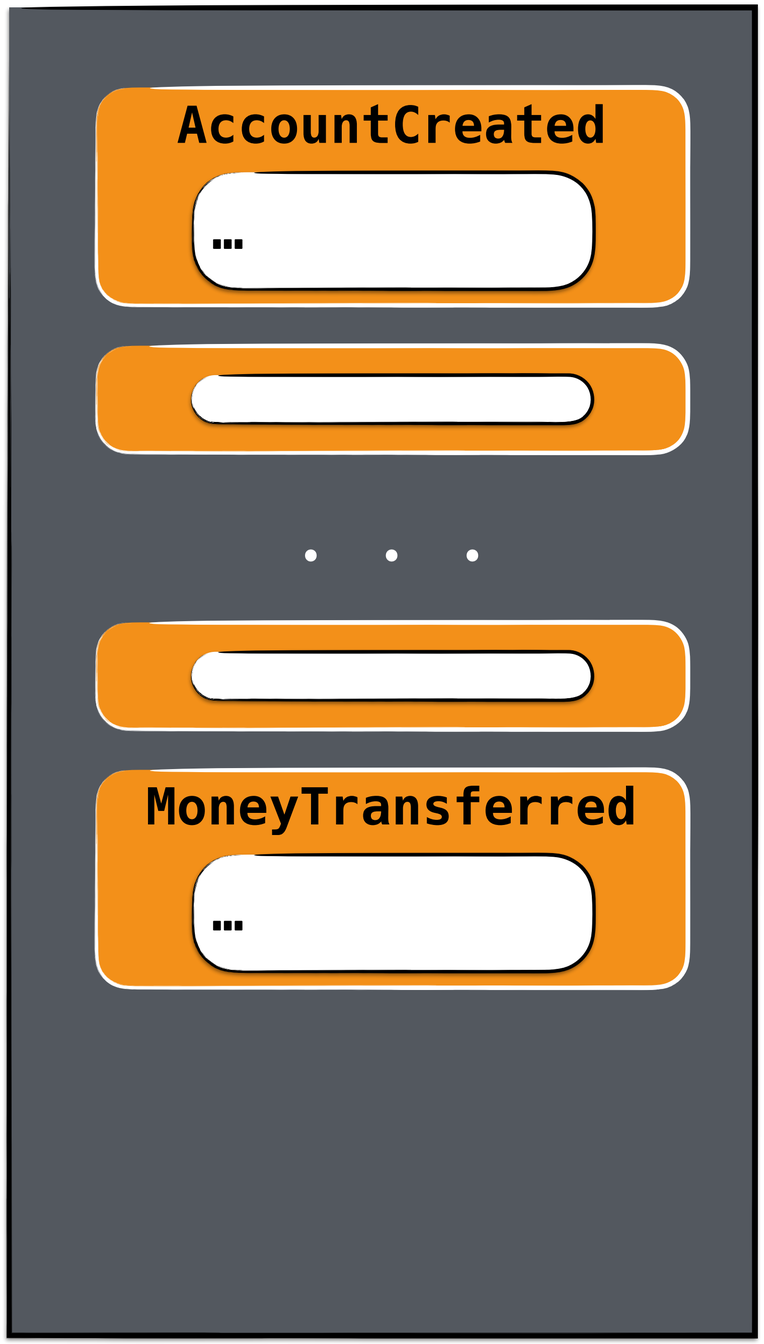
https://debitoor.com/dictionary/year-end

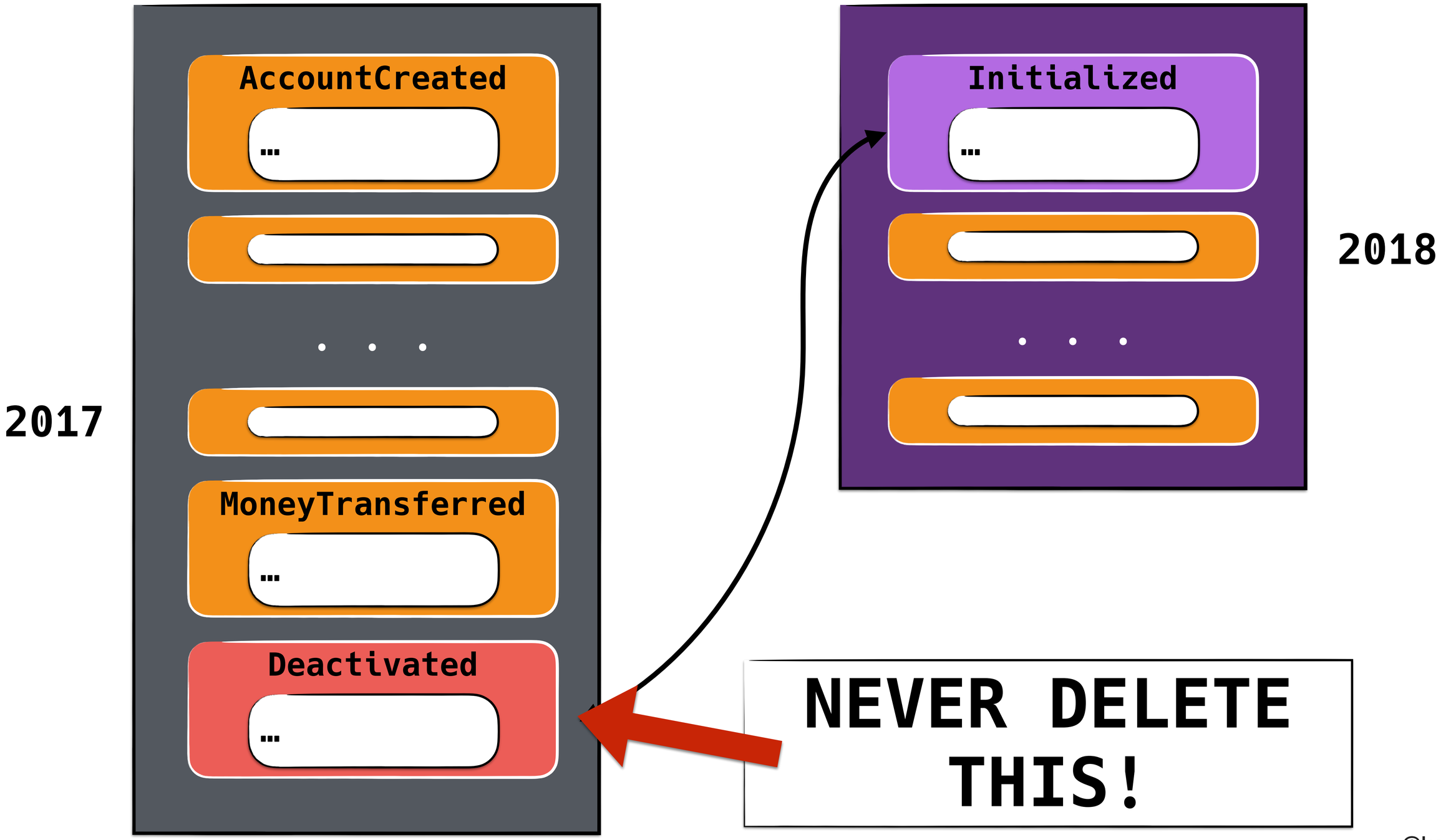
# Copy-Transform

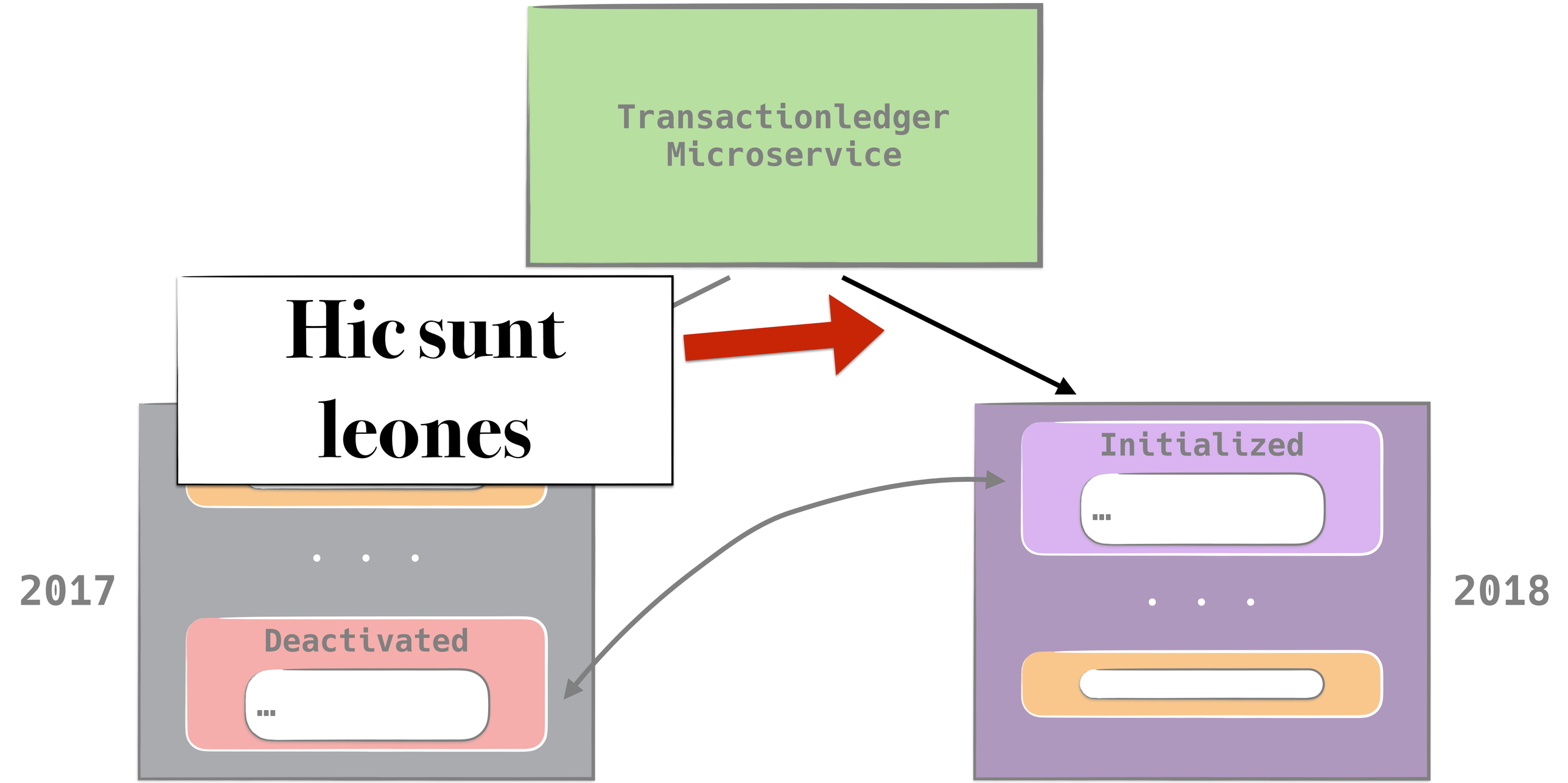
A.K.A. EVENTSOURCING

REFACTORING POWERTOOL

2017

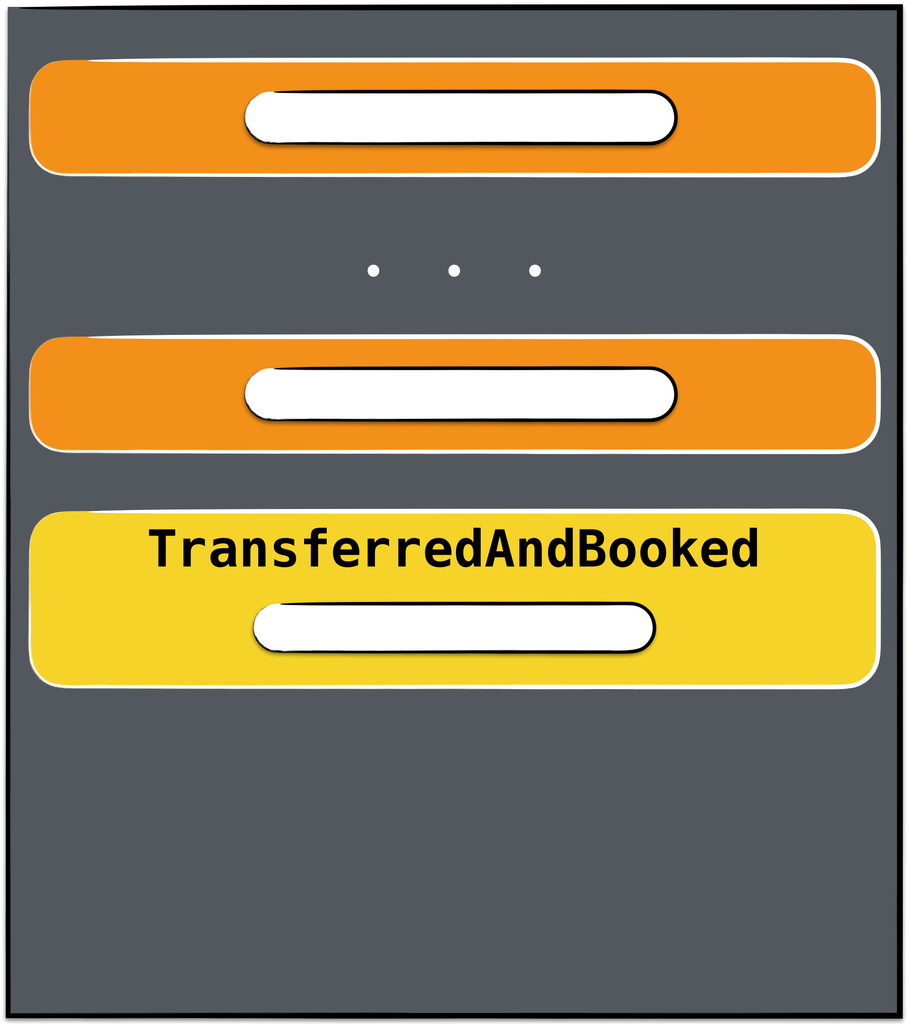


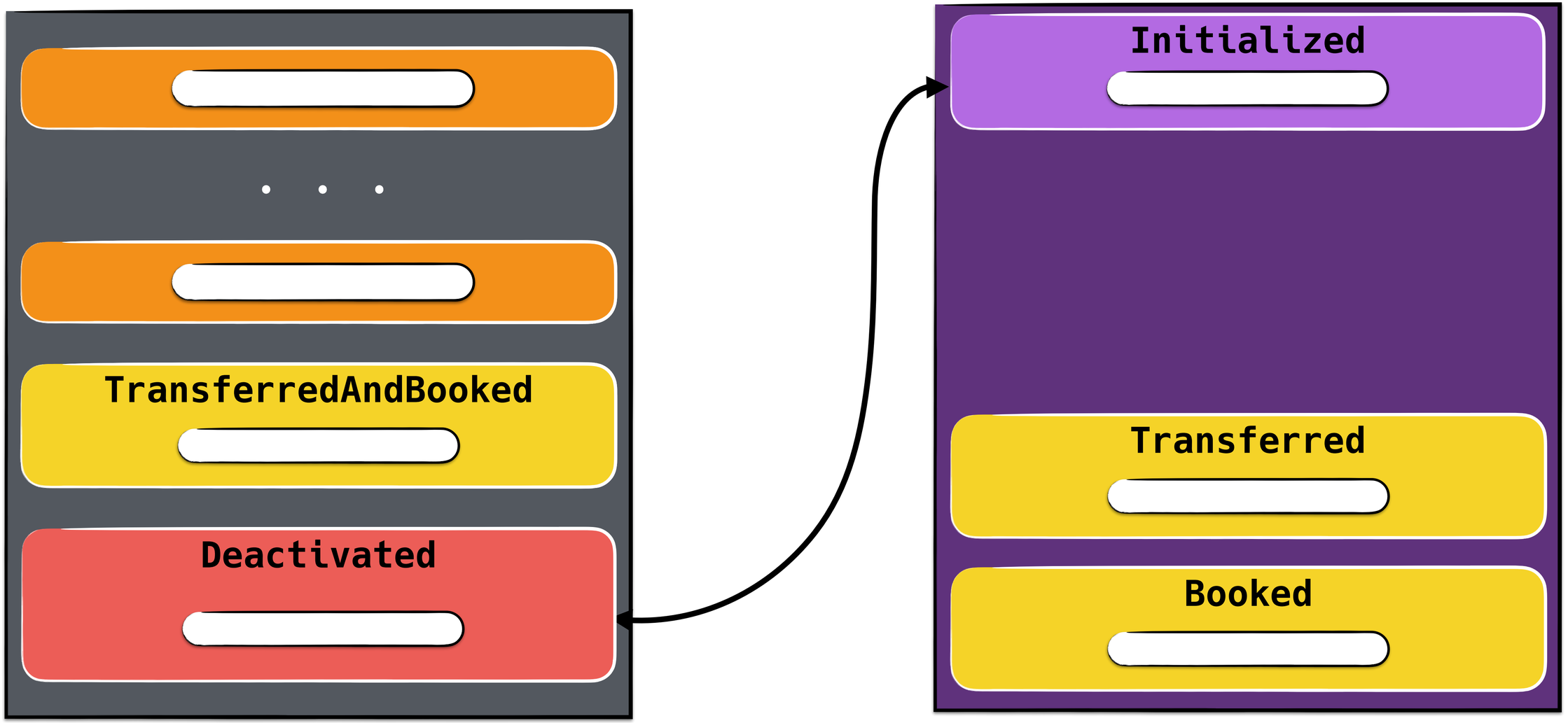

AccountCreated
...
MoneyTransferred
...
Deactivated
...
2017
Initialized
...
2018
NEVER DELETE THIS!


Transactionledger
Microservice
Hic sunt
leones
2017
Deactivated
...
2018
Initialized
...

Same idea if you need to remodel your domain!

FOOANDBAREVENT


TransferredAndBooked


TransferredAndBooked
Deactivated
Initialized
Transferred
Booked

# The devil is in the detail

# Dealing with errors

@koenighotze

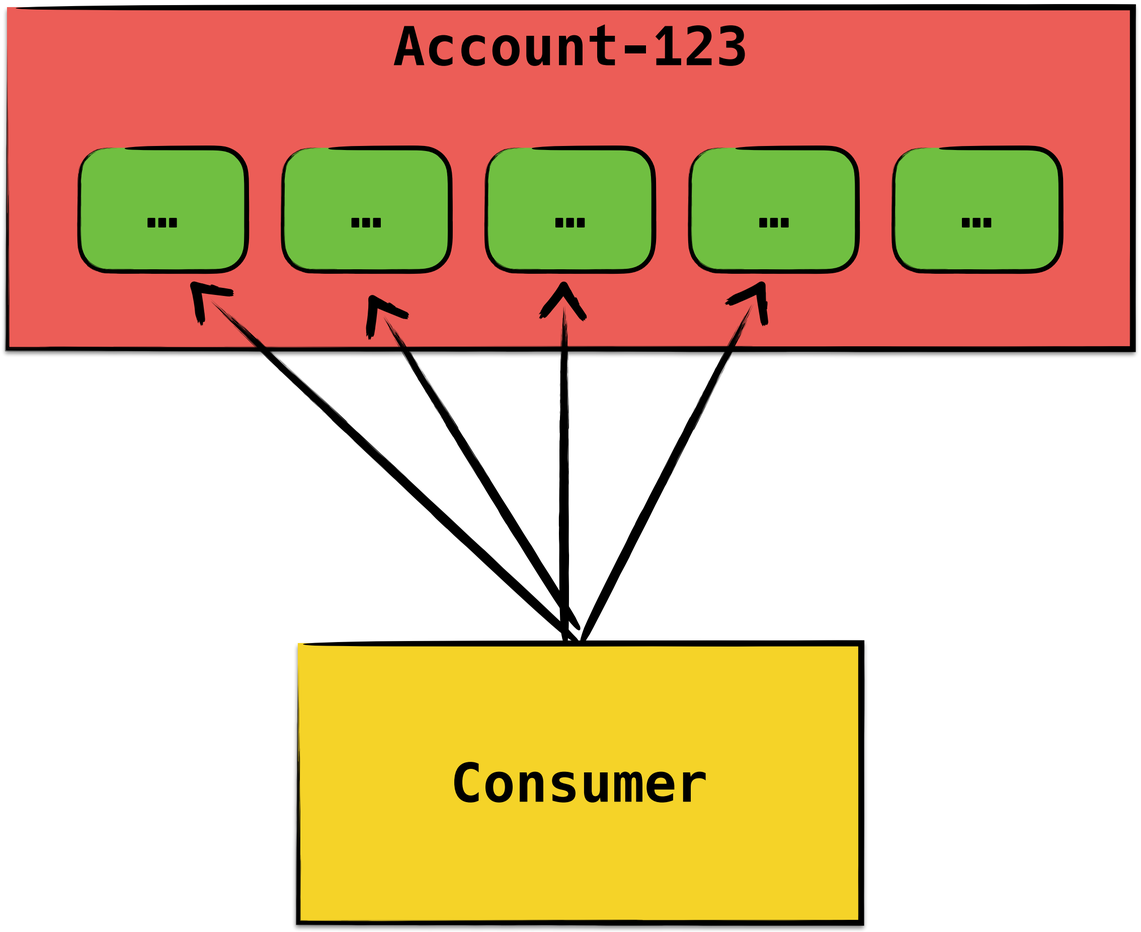
# MoneyTransferred
## eventId: 231233

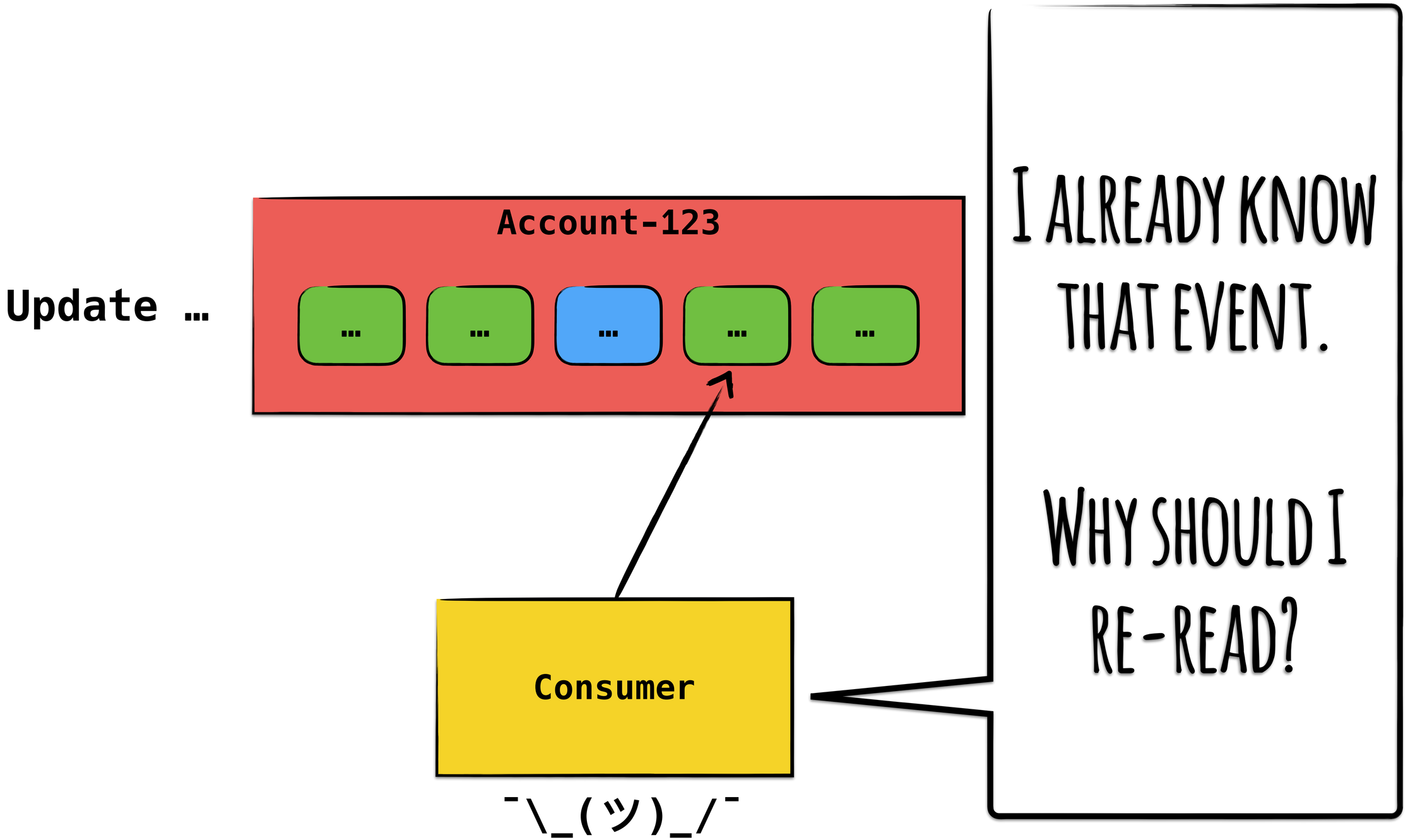
```
amount: 97 Euro
withdrawnAt: 2018-08-30T08:58:26.624
```

Just update the event in the eventstore!


UPDATE transactions
SET currency='EUR'
WHERE eventId='231233'

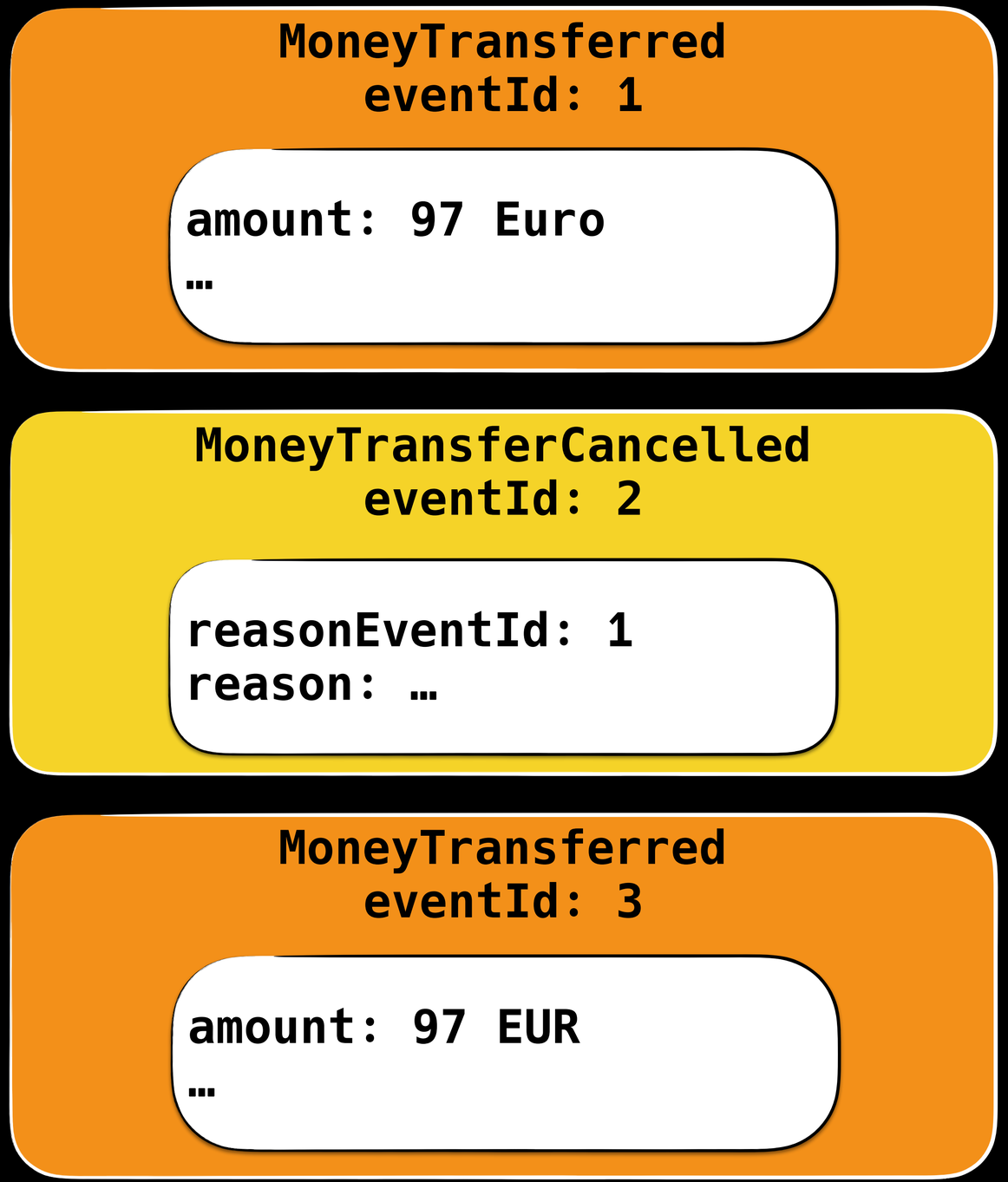
# Challenges?


Account-123
...
...
...
...
...
Consumer


Update …
Account-123
…
…
…
…
…
Consumer
¯_(ツ)_/¯
I ALREADY KNOW THAT EVENT.
WHY SHOULD I RE-READ?

# Ok. Then just use compensation events

# The cancelled or corrected event

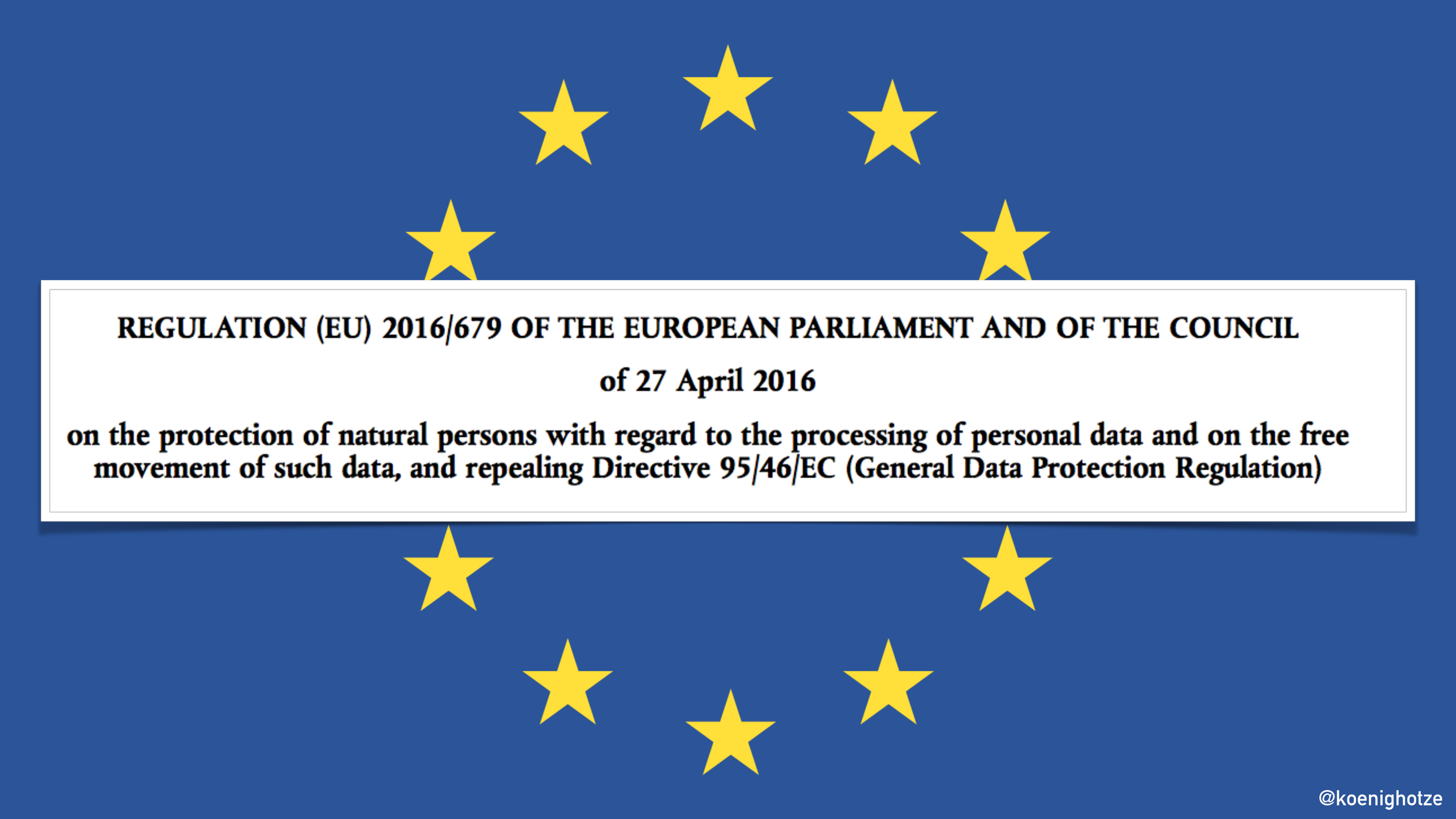

MoneyTransferred
eventId: 1
amount: 97 Euro
…
MoneyTransferCancelled
eventId: 2
reasonEventId: 1
reason: …
MoneyTransferred
eventId: 3
amount: 97 EUR
…

The full compensation makes the reason for compensation explicit


# GDPR, COMPLIANCE AND EVENTSOURCING

@koenighotze


@koenighotze



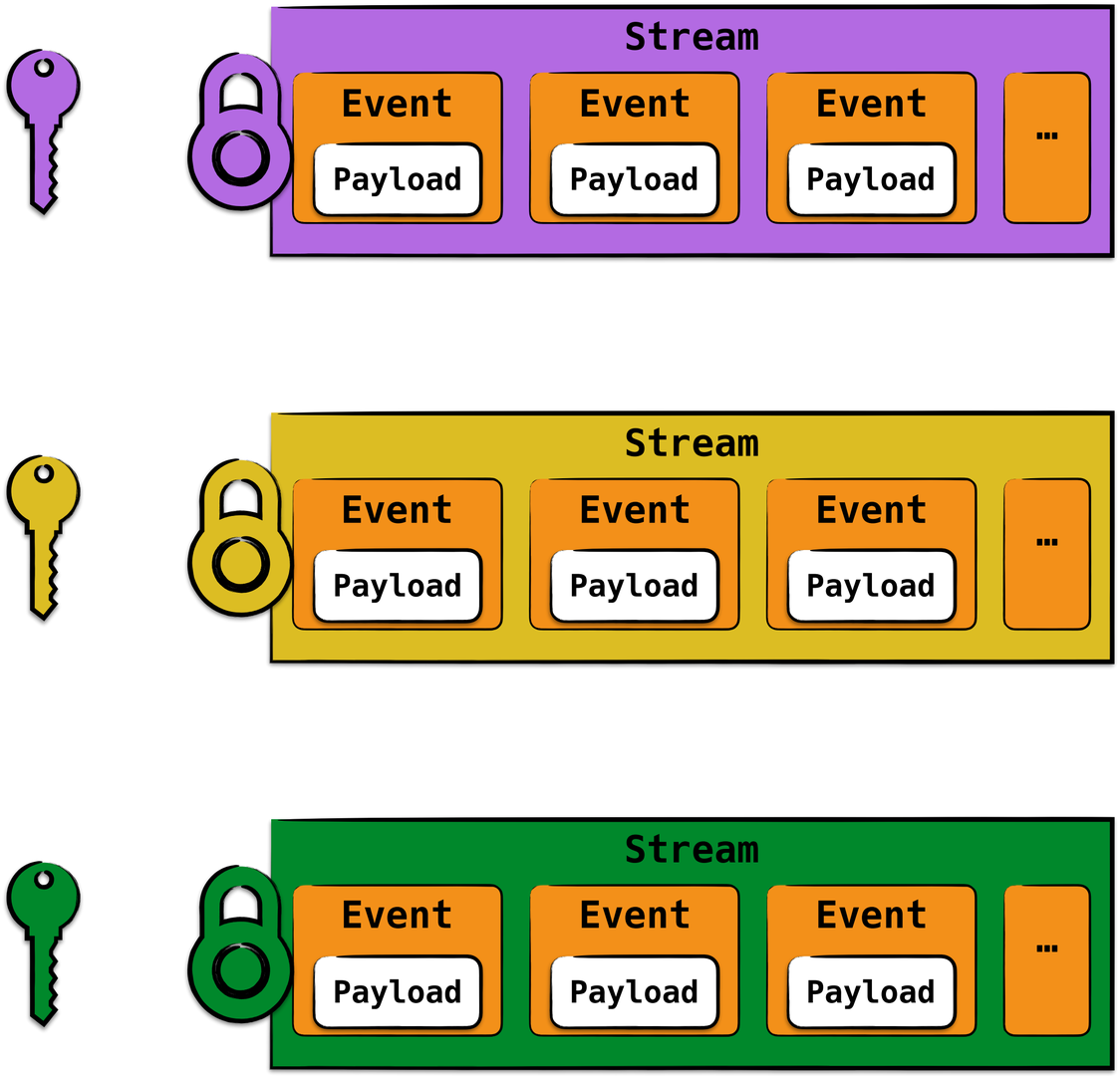

@koenighotze

*Article 17*

**Right to erasure ('right to be forgotten')**

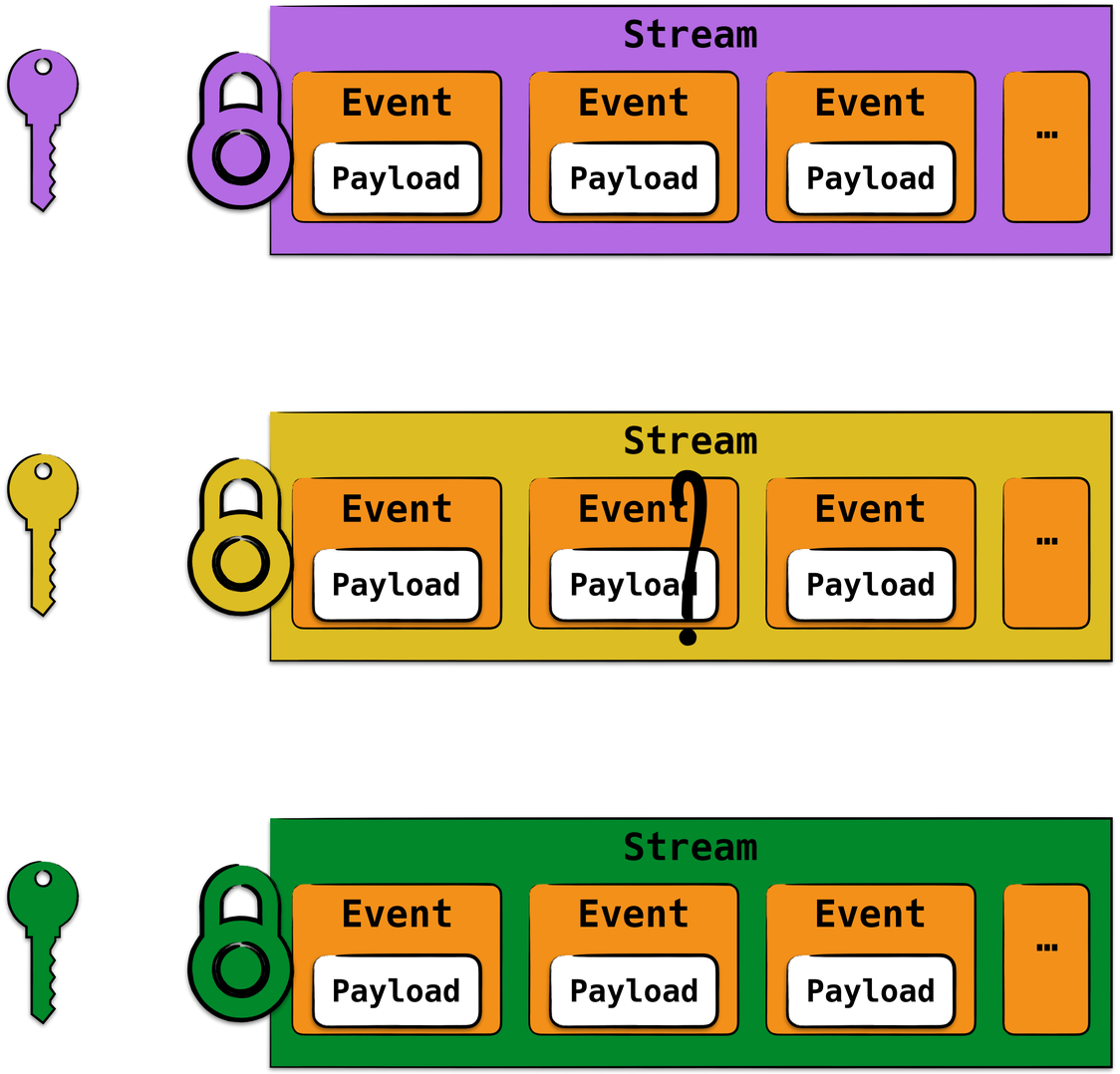
1. The data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay where one of the following grounds applies:

JUST ENCRYPT AND THROW THE KEY AWAY


Stream
Event
Payload
Event
Payload
Event
Payload
…
Stream
Event
Payload
Event
Payload
Event
Payload
…
Stream
Event
Payload
Event
Payload
Event
Payload
…

"PLEASE DELETE ALL MY DATA"

Deletion is effectively deleting the stream-specific key


Stream
Event
Payload
Event
Payload
Event
Payload
…
Stream
Event
Payload
Event
Payload
?
Event
Payload
…
Stream
Event
Payload
Event
Payload
Event
Payload
…

# Challenges?

Key administration

Finding what needs to be deleted

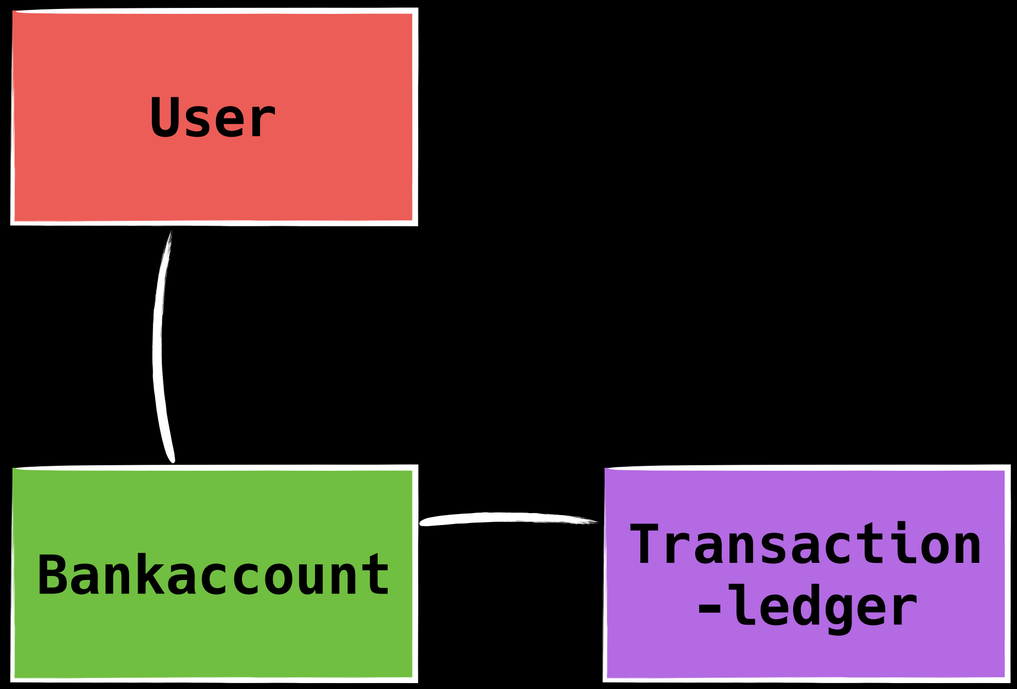
Storage implications

Coding complexity

Dashboards, Monitoring

# Being able to delete is awesome

"PLEASE DELETE USER-456"

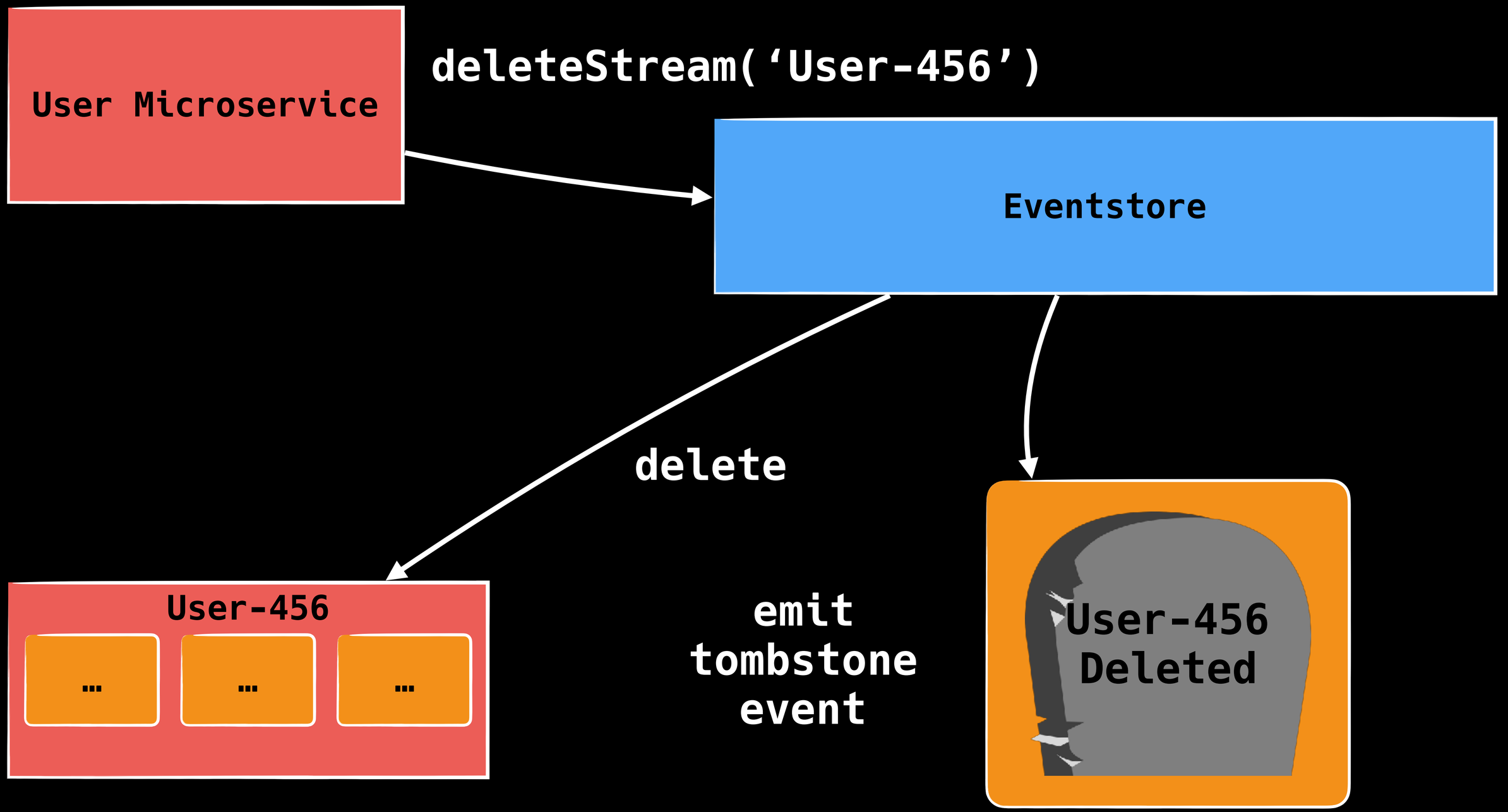

User
Bankaccount
Transaction
-ledger

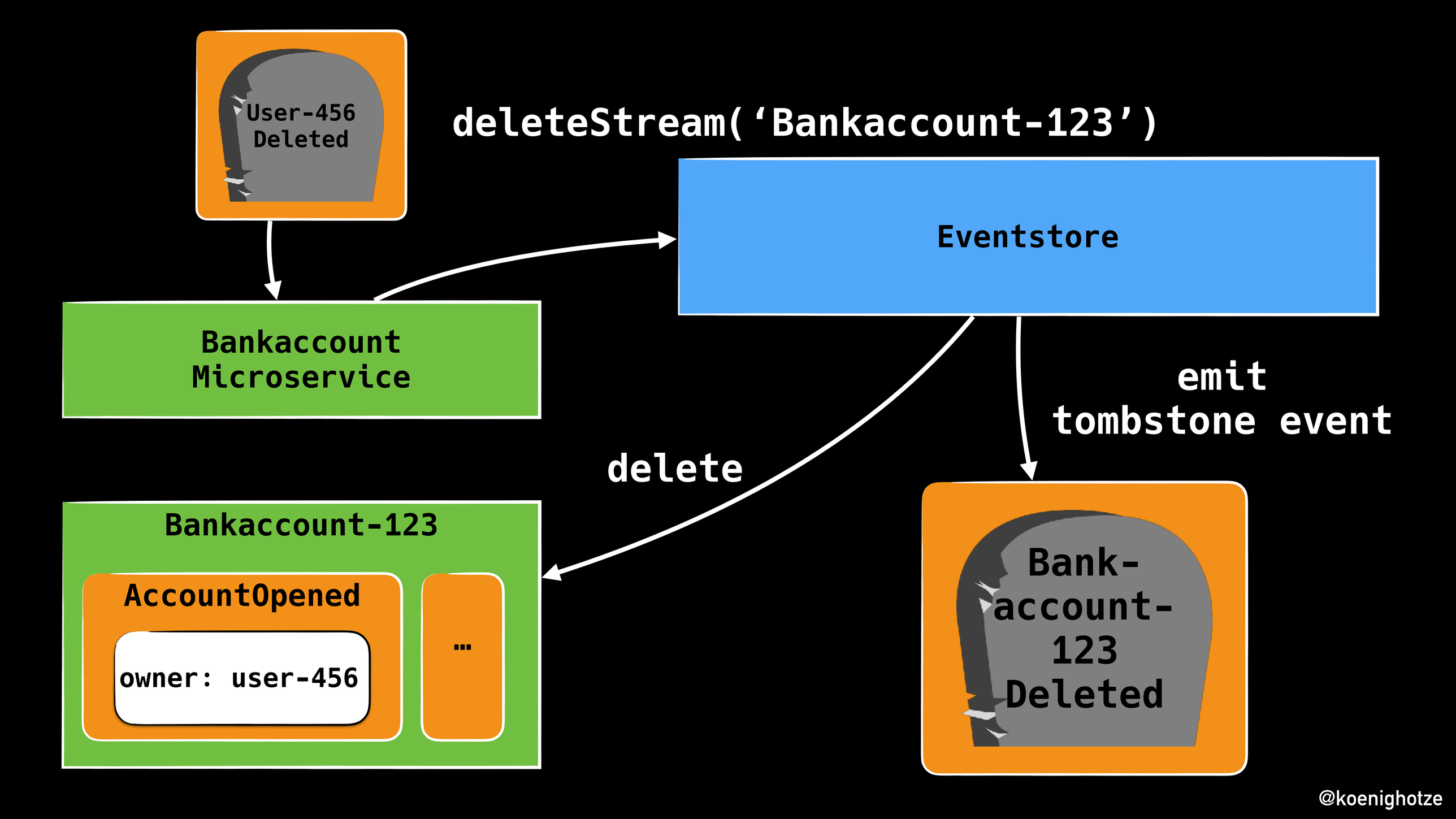
# Cascading deletes with tombstones


User Microservice

Bankaccount Microservice

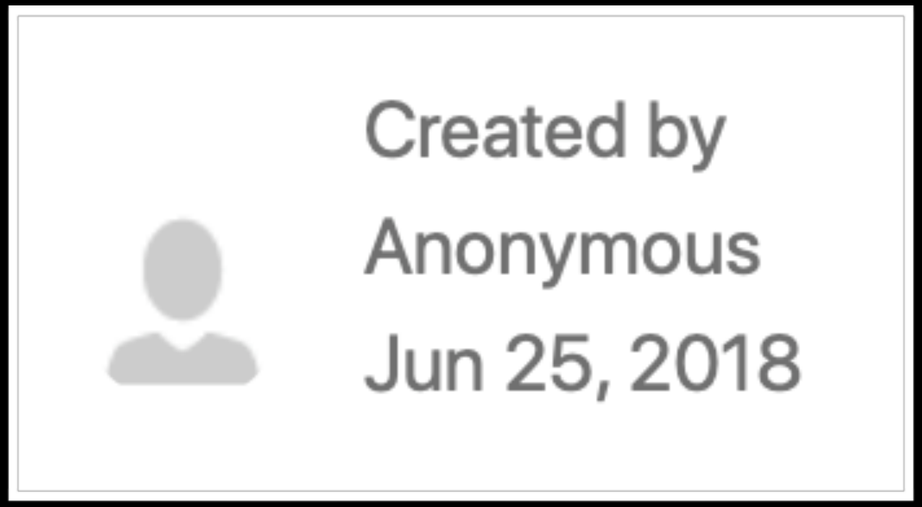
Transactionledger Microservice


User Microservice
deleteStream(‘User-456’)
Eventstore
delete
User-456
…
…
…
emit
tombstone
event
User-456
Deleted

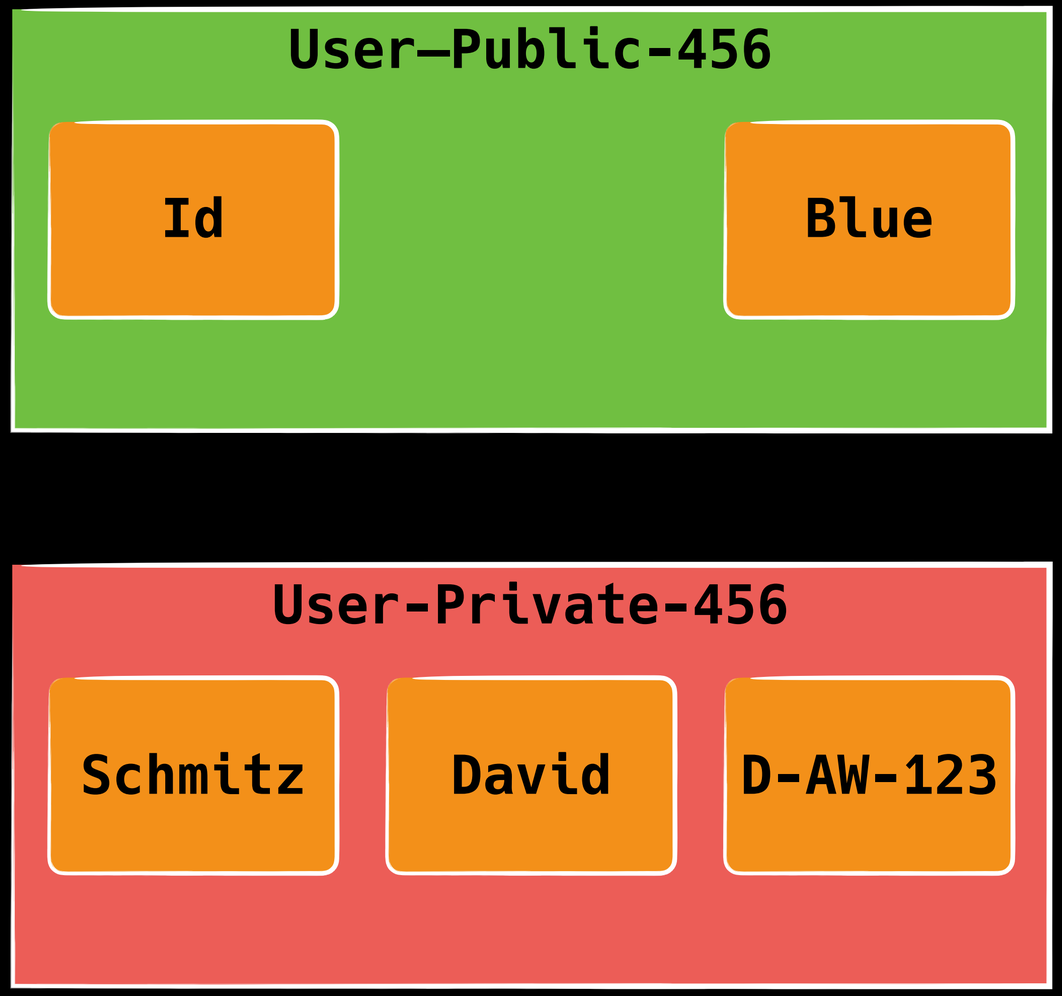


# What about links between aggregates?

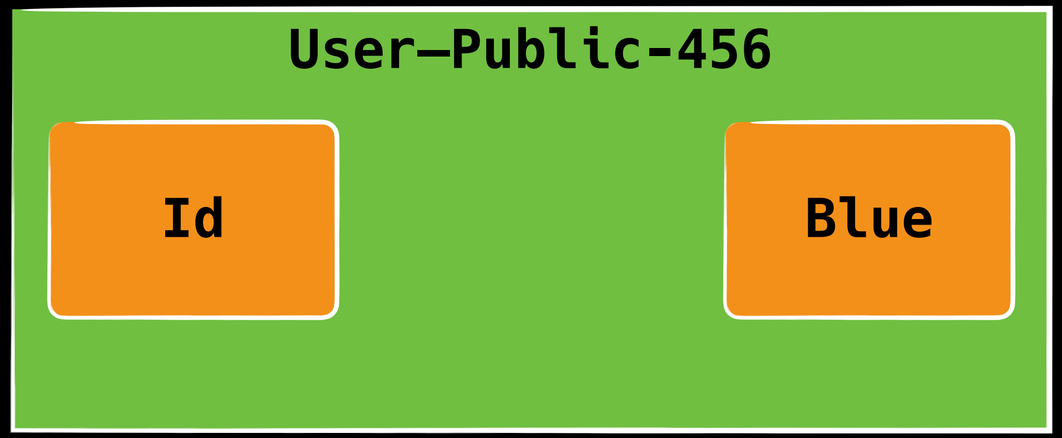
# What if I delete a Confluence account?

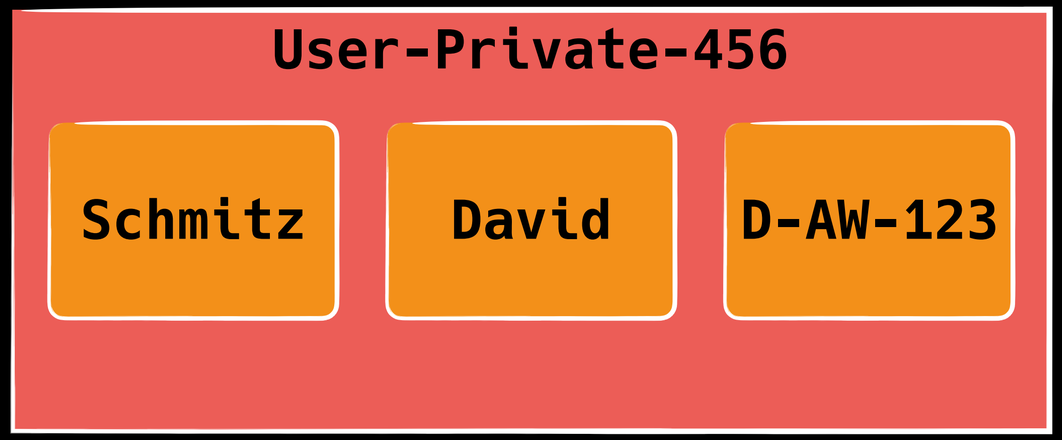

Created by
Anonymous
Jun 25, 2018

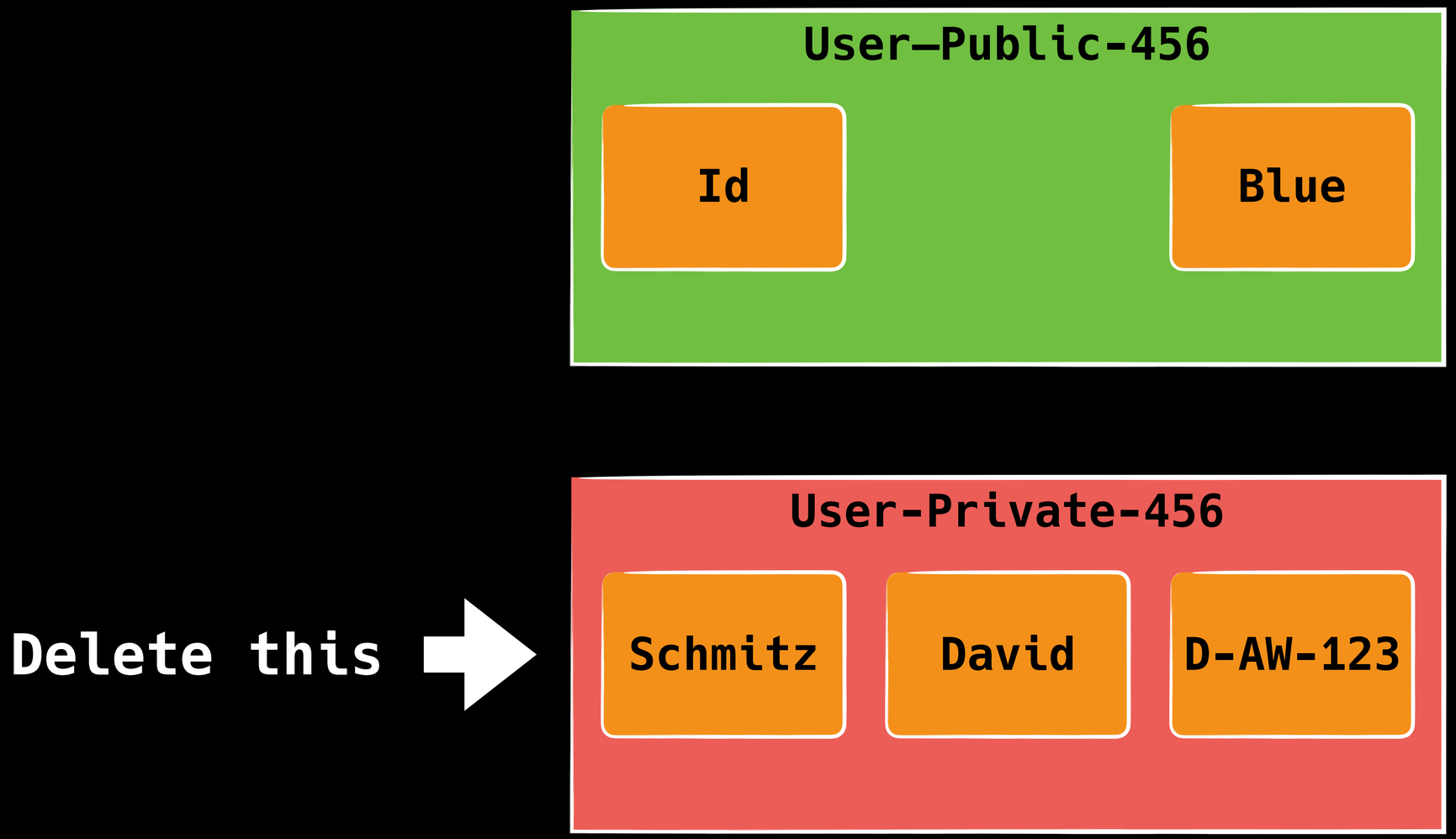
# PUBLIC/PRIVATE DATA


User—Public-456
Id
Blue
User-Private-456
Schmitz
David
D-AW-123

Keep this


User—Public-456
Id
Blue


User-Private-456
Schmitz
David
D-AW-123


User—Public-456
Id
Blue
User-Private-456
Schmitz
David
D-AW-123
Delete this

You MAY be able to keep
REFERENCES to the public data

JUST ANONYMISE THE DATA

# Recital 26
# EU GDPR

(26) The principles of data protection should apply to any information concerning an identified or identifiable natural person.

Personal data which have undergone pseudonymisation, which could be attributed to a natural person by the use of additional information should be considered to be information on an identifiable natural person.

Recital 26
EU GDPR

(26) The principles of data protection should apply to any information concerning an identified or identifiable natural person.

Personal data which have undergone pseudonymisation, which could be attributed to a natural person by the use of additional information should be considered to be information on an identifiable natural person.

# Surprise: No easy answers

# Ask your lawyer or CISO

# That's it?

@koenighotze

Microservices + DDD + ES= 

Needs more up-front design

You can refactor, you can clean up

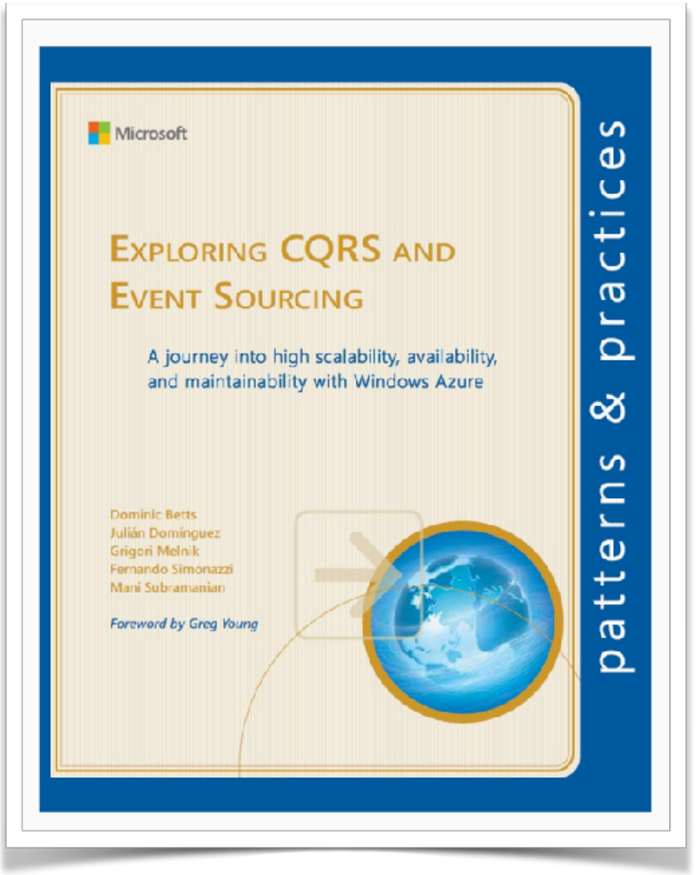
Not enough in-depth books

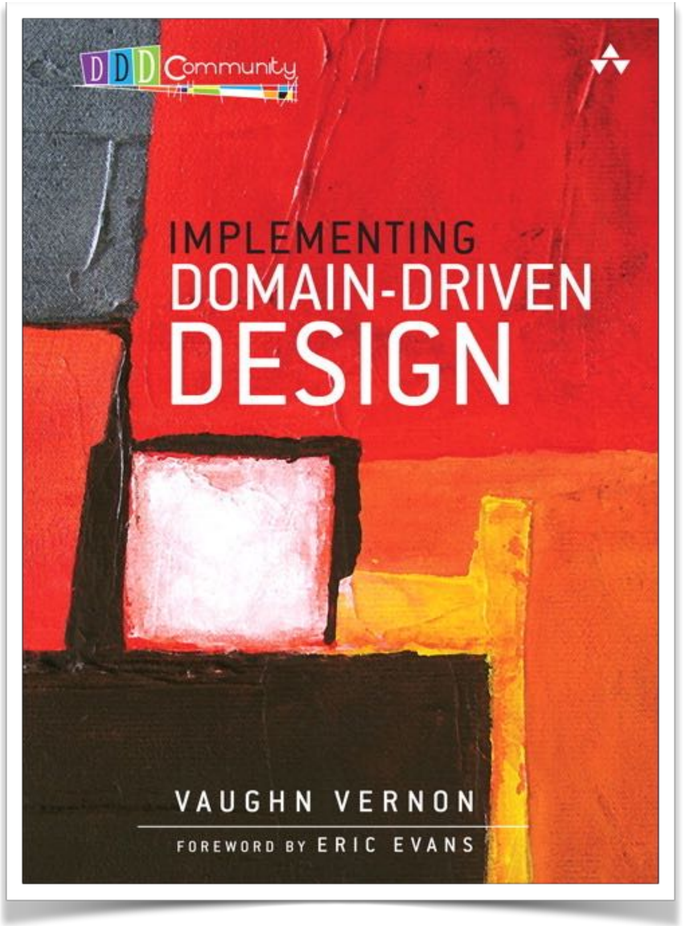
Avoid frameworks esp. if going polyglot

Beware: "just..." or "...made easy"

FORGET THIS TALK...

READ SOME PAPERS



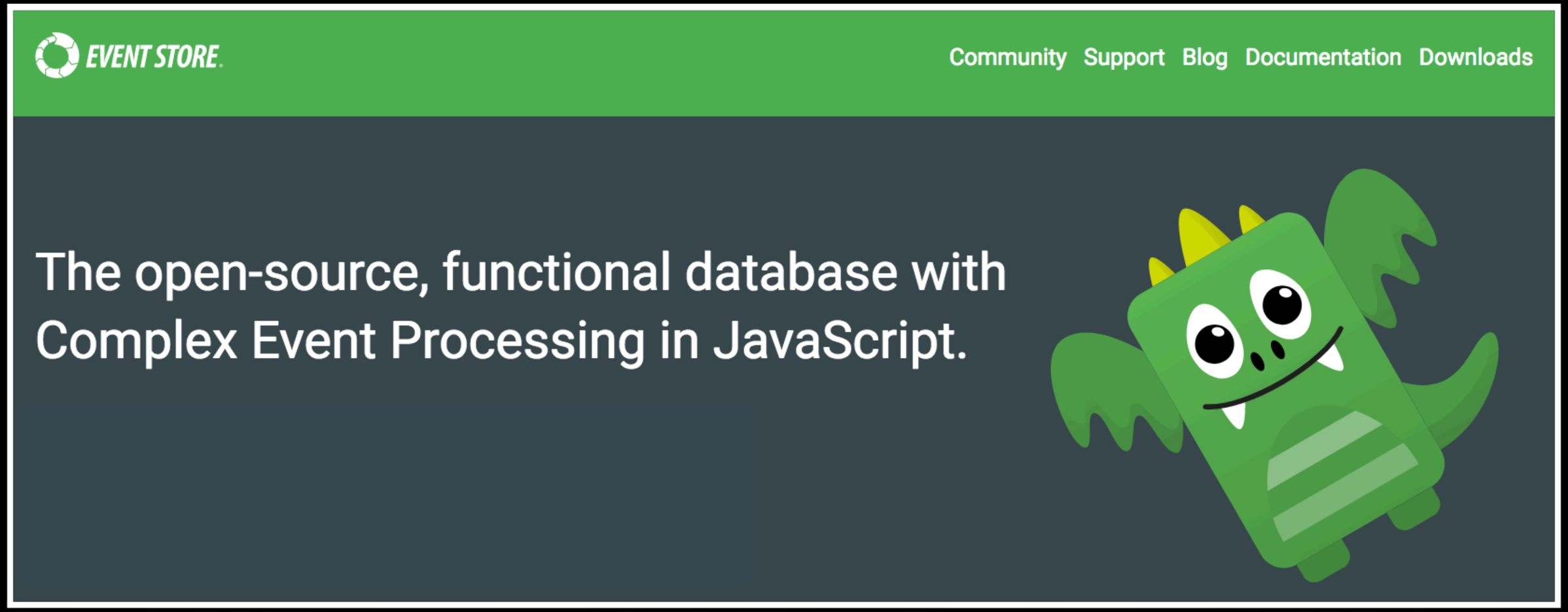

The Dark Side of Event Sourcing: Managing Data Conversion

Michiel Overeem[1], Marten Spoor[1], and Slinger Jansen[2]

**Versioning in an Event Sourced System**

Gregory Young

# Choose the right tool?




# EVENTSTORE.ORG

# Thank you!

## Questions and Feedback?

## @koenighotze