Event Sourcing

Dino Esposito
AUTHOR

@despos www.software2cents.wordpress.com

In the real world you observe events.

In software, you tend to write models.

Immutabl e

Don’t miss a thing

Can be replayed

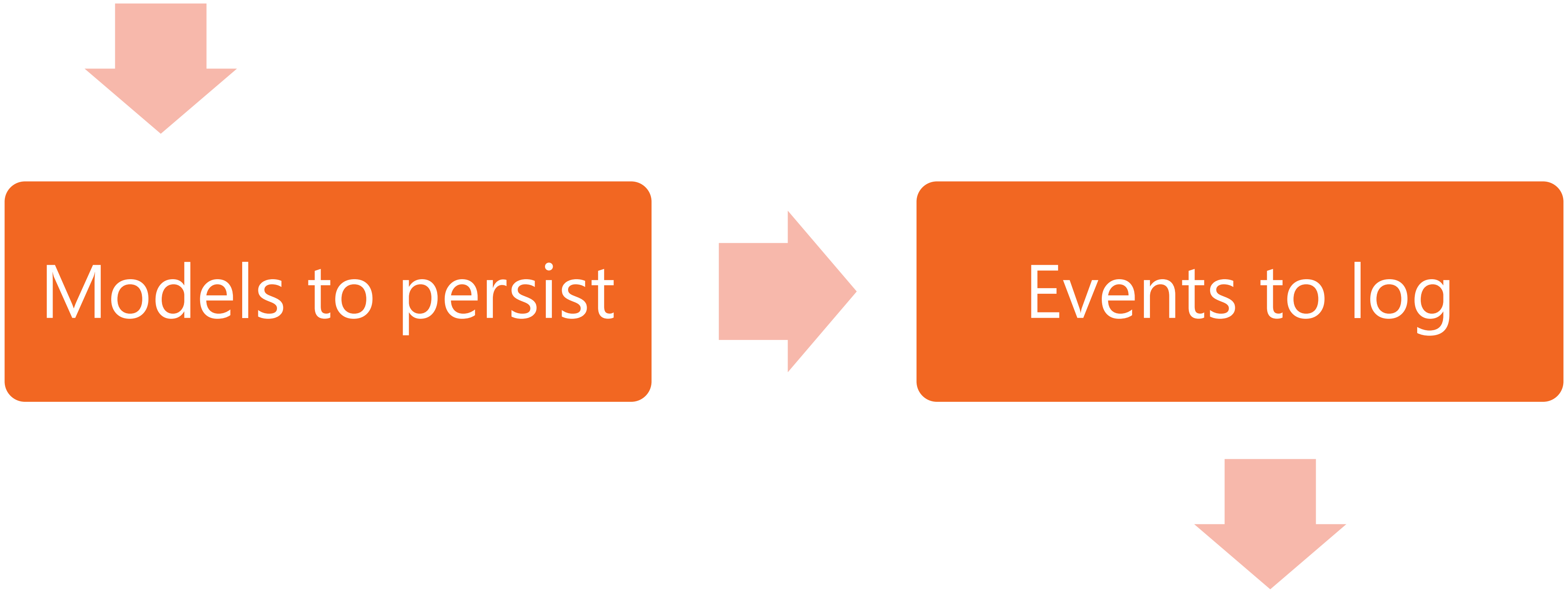
Key Points

Events

Events as **Data Source**

Build **Projections** of Event Data

CQRS



Deep impact on system architecture

Events are **also** for the common application.

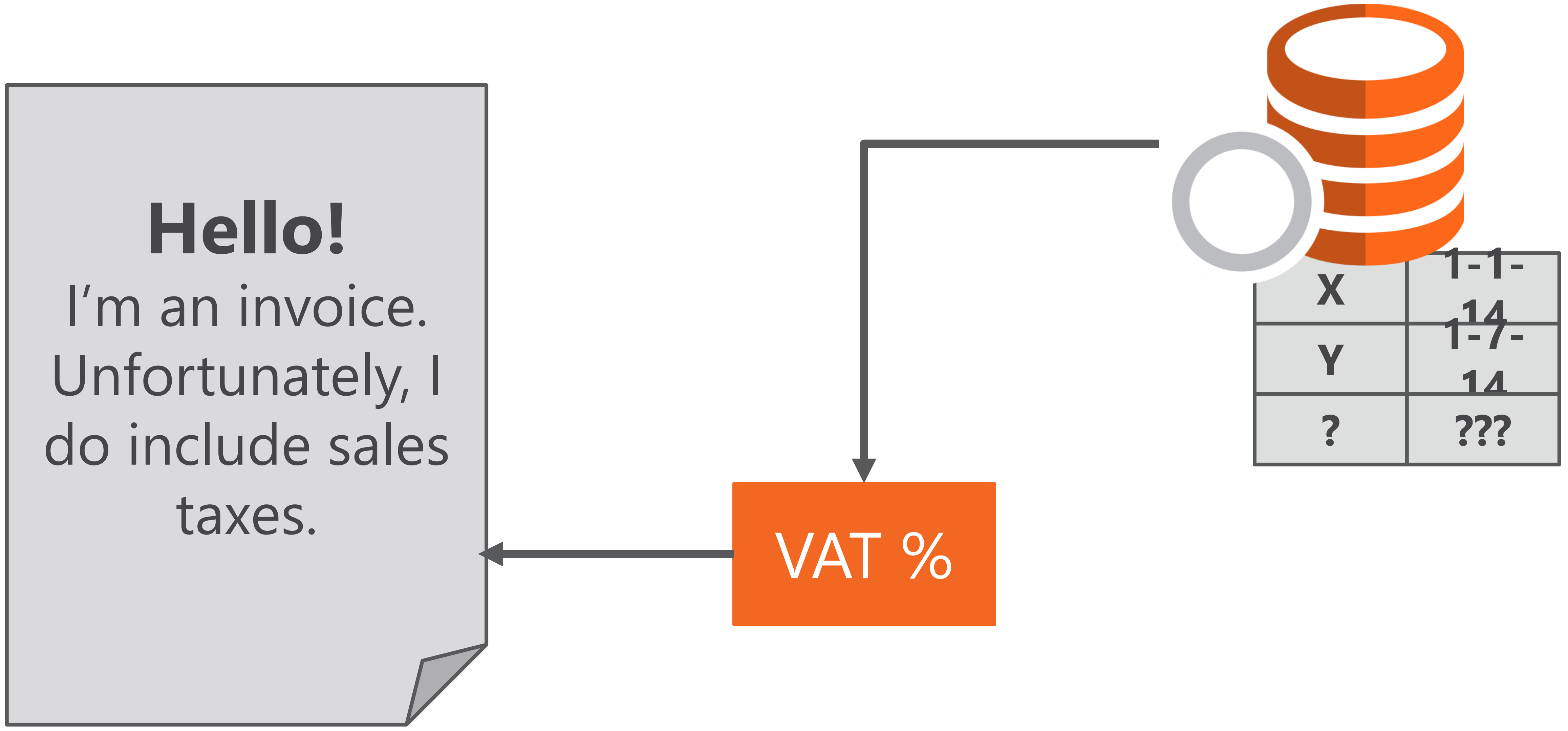
Newton's tree, Botanic Gardens, Cambridge

You have a CRUD system. → Need to know the state of the system at a given time

You may or may not be able to extract just that information with canonical and common techniques.

It’s not that you don’t need events.

You just don’t need events yet.



Hello!
I'm an invoice. Unfortunately, I do include sales taxes.

X	1-1-14
Y	1-7-14
?	???

Event Sourcing

It’s about ensuring that all changes made to the application state during the entire lifetime of the application are stored as a sequence of events.

This is not how the vast majority of applications work today.

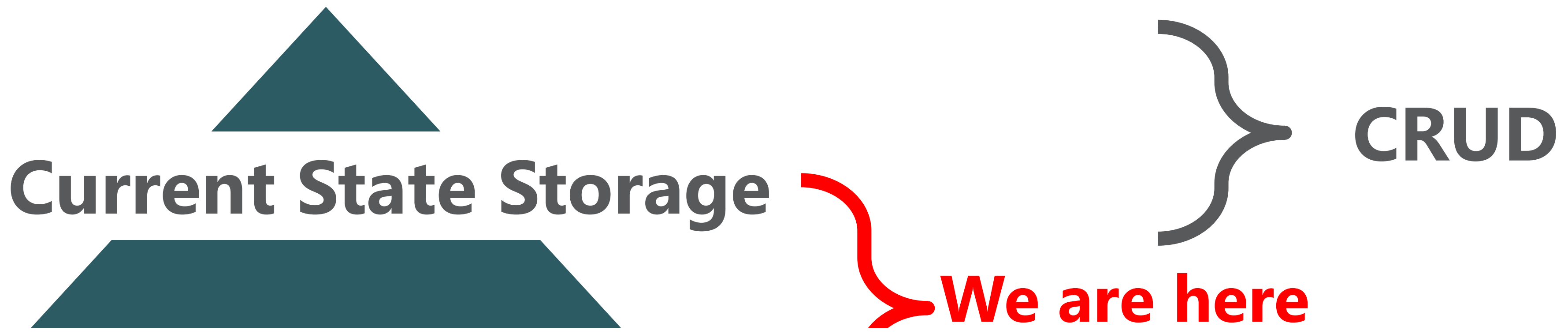
Most applications work by storing the current state and using stored states to process business transactions.

Structural representation

- List of ordered goods
- Payment information
- Shipping information

Event representation

- Add item #1
- Add item #2
- Add payment info
- Update item #2
- Remove item #1
- Add shipping info


CRUD
Current State Storage
We are here

Key Facts of **Event Sourcing**

An event is something that has happened in the past

Events are expression of the ubiquitous language

Events are not imperative and are named using past tense verbs

Have a persistent store for events

Append-only, no delete

Replay the (related) events to get to the last known state of an entity

- Replay from the beginning or a known point (**snapshot**)

An Event Is Something That Happened in the Past

- **Once stored, events are immutable**
  - Can be duplicated and replicated (for scalability reasons)
- **Any behavior associated with the event has been performed**
  - Replaying the event doesn’t require to repeat the behavior
- **You don’t miss a thing**
  - Track everything that happened at the time it happened
  - Regardless of the effects it produced

Data saved at a lower abstraction level

CQRS is not for just a few apps either.

CQRS and Events

CQRS is the dawn of a new software design experience. **Events** are what you find when the dawn has actually turned into a brand new day.

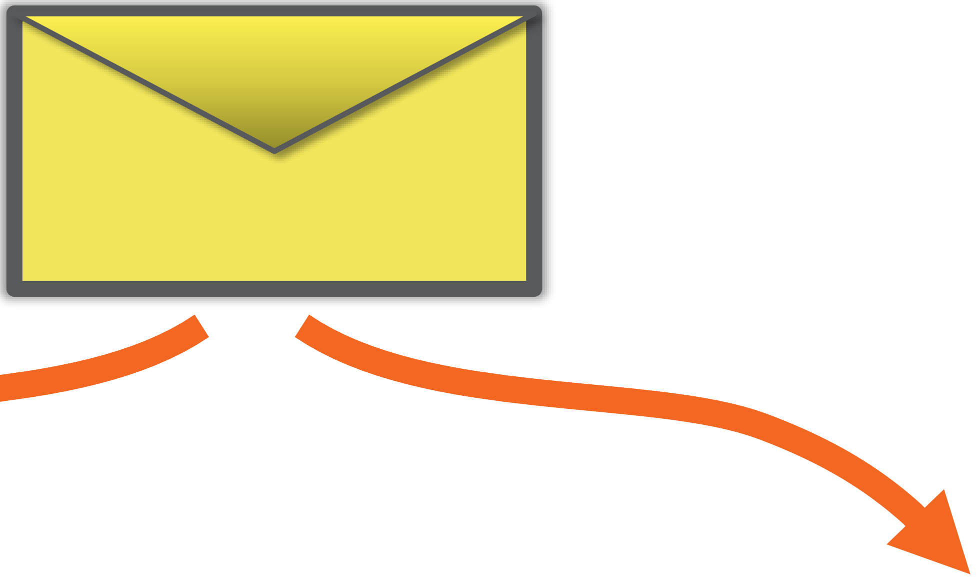
Not really
revolutionary

Events are a revolutionary new approach.

Not really
new

Events are not for just a few apps either.

Power to Events



1. **Store current state**
2. Use events to log relevant facts

1. **Store events**
2. Build relevant snapshots of facts

Event stream

- ➕ Q=1, Product=123
  Tennis racket
- ➕ Q=1, Product=456
  Ball pack
- ➕ CC 123456789
- ↻ Q=2, Product=456
- ✕ Remove Product=123
- ➕ 1 Unknown Place

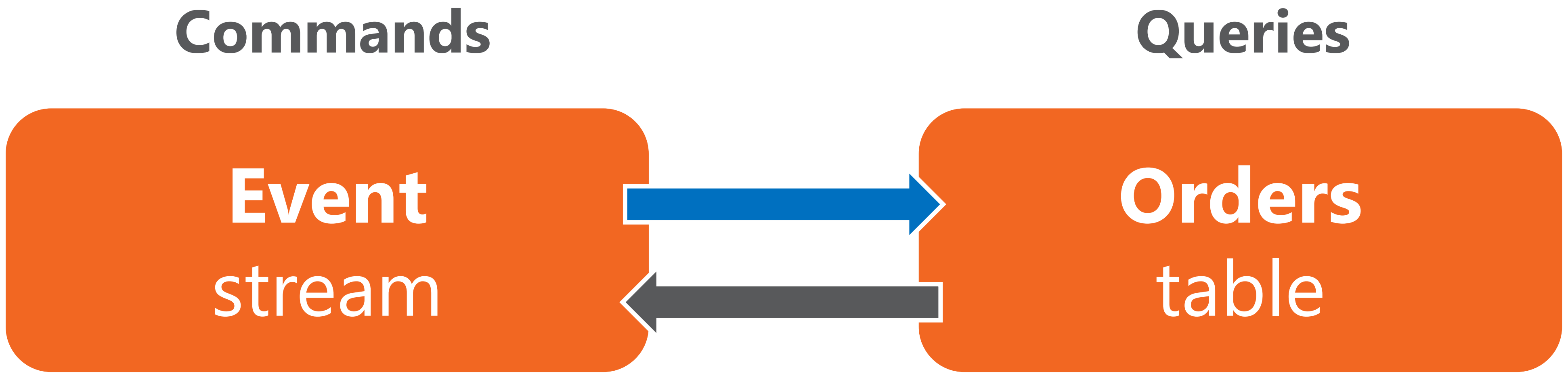
Ship to:
1 Unknown Place

2	Ball pack ID=123	5.50 EUR
		11 EUR

Credit card:
123456789

Is it important to track **what** was added and then removed ?

Is it important to track **when** an item was removed from cart?



Foundation of **Persistence**

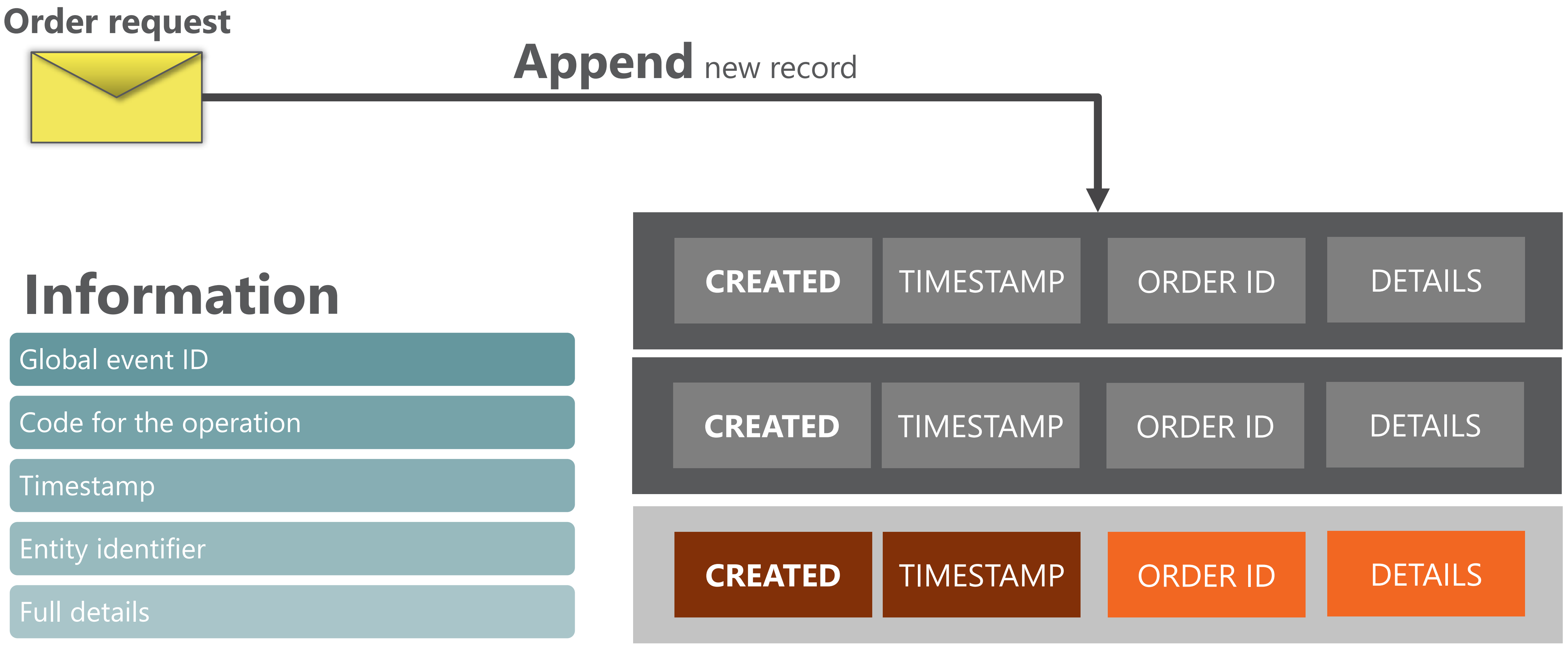
CREATE

UPDATE

DELETE

QUERIES

CREATE Operations

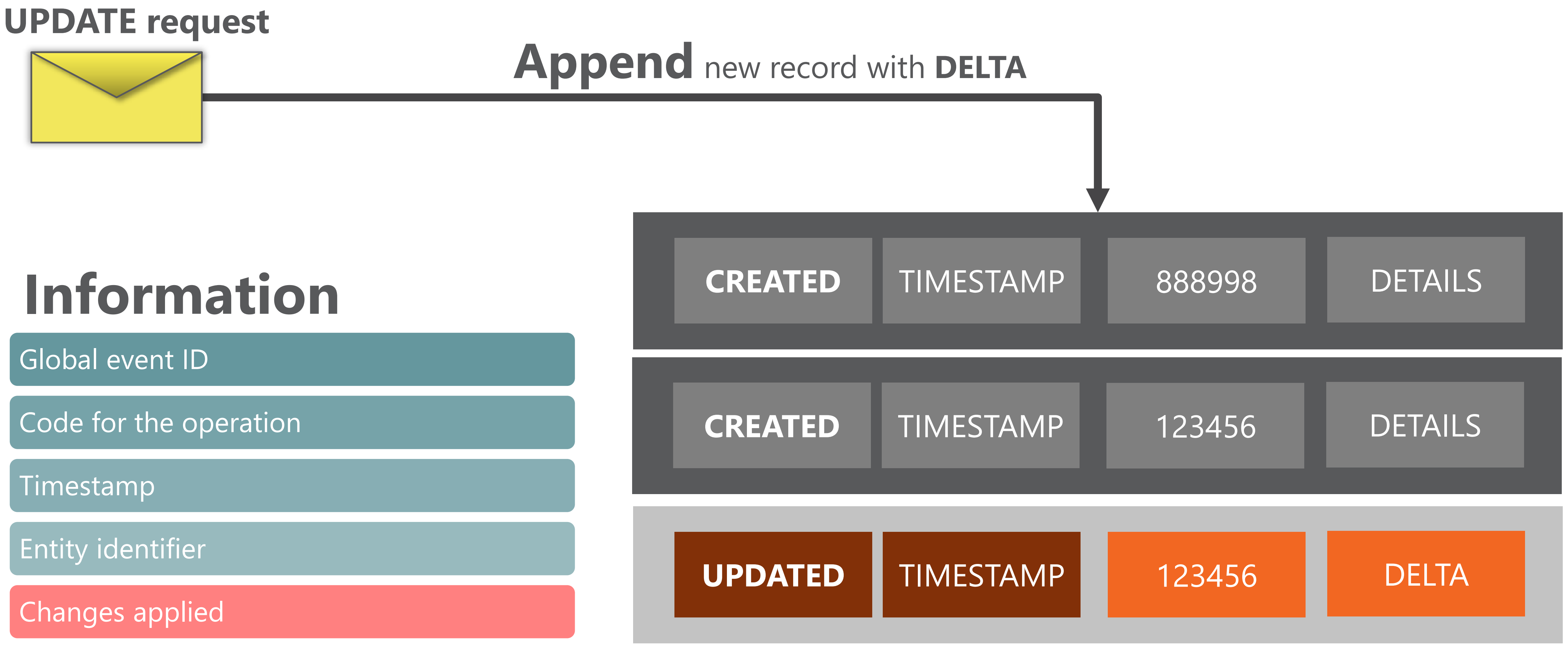


Information

- Global event ID
- Code for the operation
- Timestamp
- Entity identifier
- Full details

Transparent storage: relational, NoSQL, graph

UPDATE Operations

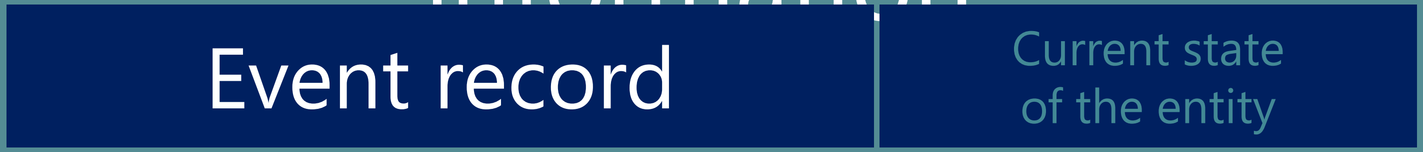


Information

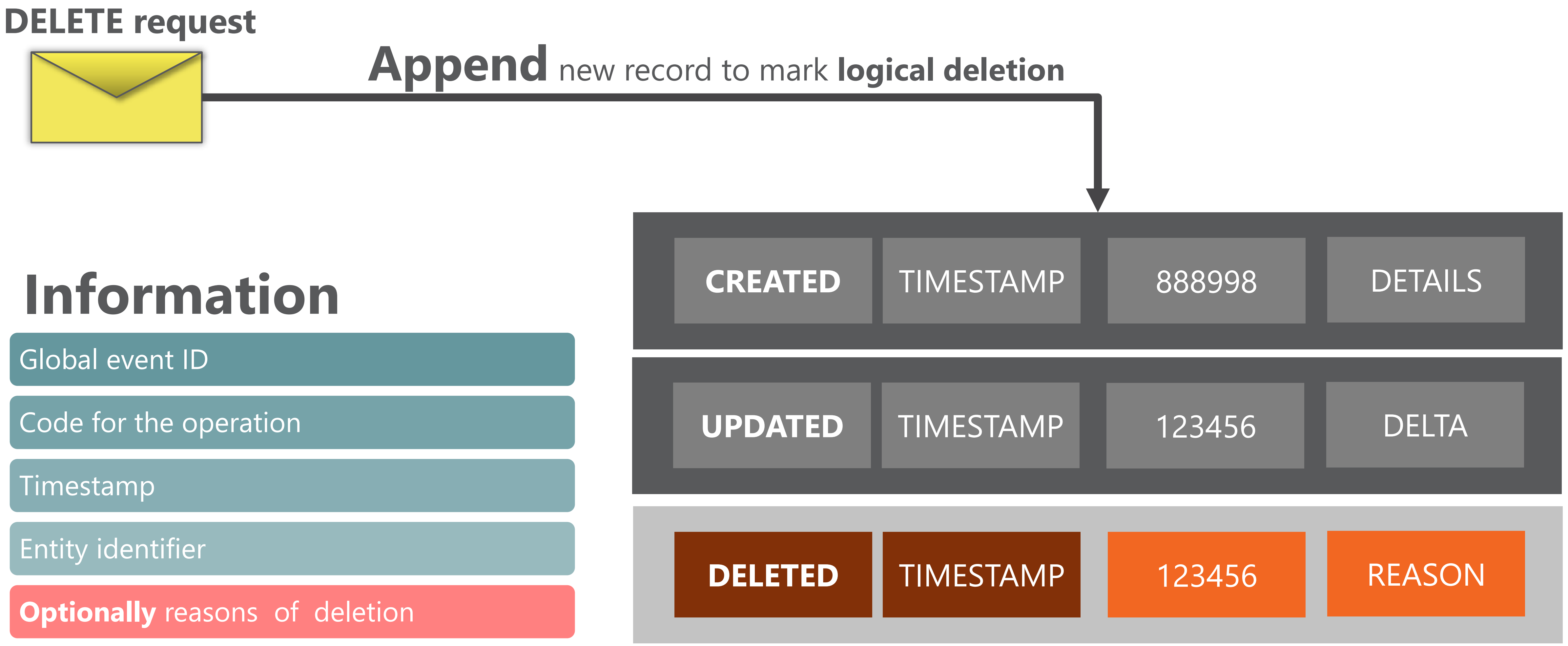
- Global event ID
- Code for the operation
- Timestamp
- Entity identifier
- Changes applied

Transparent storage: relational, NoSQL, graph

In some cases, you might want to consider storing **the full state of the entity** along with the specific event information



DELETE Operations



Information

- Global event ID
- Code for the operation
- Timestamp
- Entity identifier
- **Optionally** reasons of deletion

Transparent storage: relational, NoSQL, graph

Physical record deletion of events in case of UNDO functionality?

DO NOT DELETE in the middle of the stream

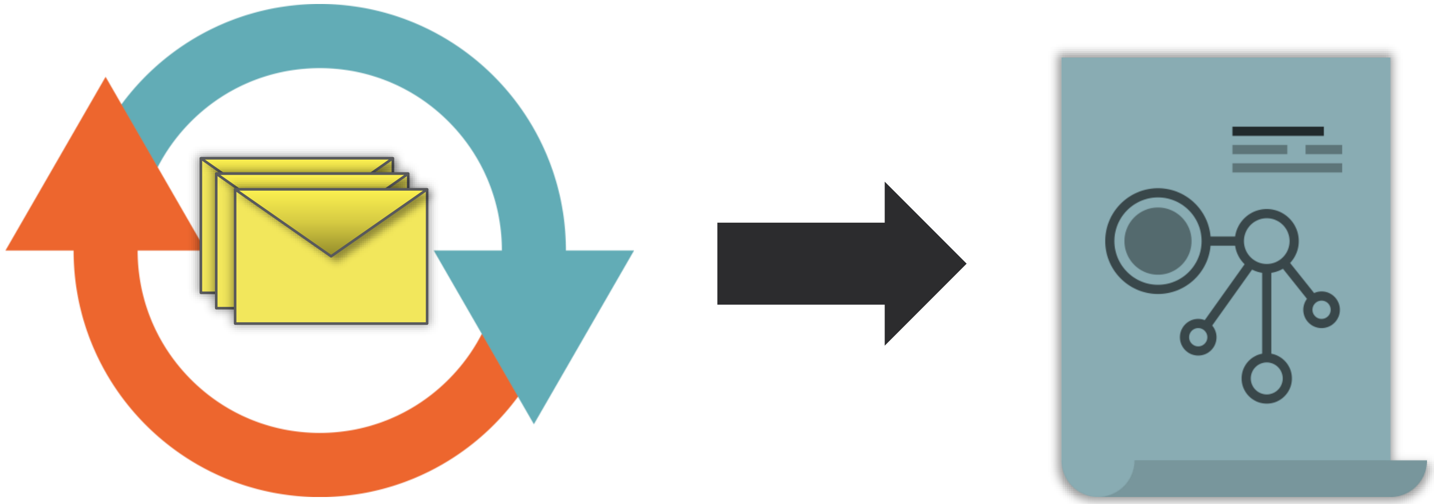
Queries?

QUERY Operations

Grab all events by entity ID

64	**CREATED**	TIMESTAMP	123456	DETAILS
92	**UPDATED**	TIMESTAMP	123456	DELTA

Replay through all events



REPLAY of Events

Get the full or partial stream of events

RavenDB query example

```
public IEnumerable<GenericEventWrapper> GetEventStream(String id)
{
    return DocumentSession
                 .Query<GenericEventWrapper>()
                 .Where(t => t.AggregateId == id)
                 .OrderBy(t => t.Timestamp)
                 .ToList();
}
```

or an analogous relational query

REPLAY of Events

Generic event wrapper class

```
public class GenericEventWrapper
{
    public string EventId { get; set; }
    public string EventOperationCode { get; set; }
    public DateTime TimeStamp { get; set; }
    public string AggregateId { get; set; }
    public DomainEvent Data { get; set; }
}
```

REPLAY of Events

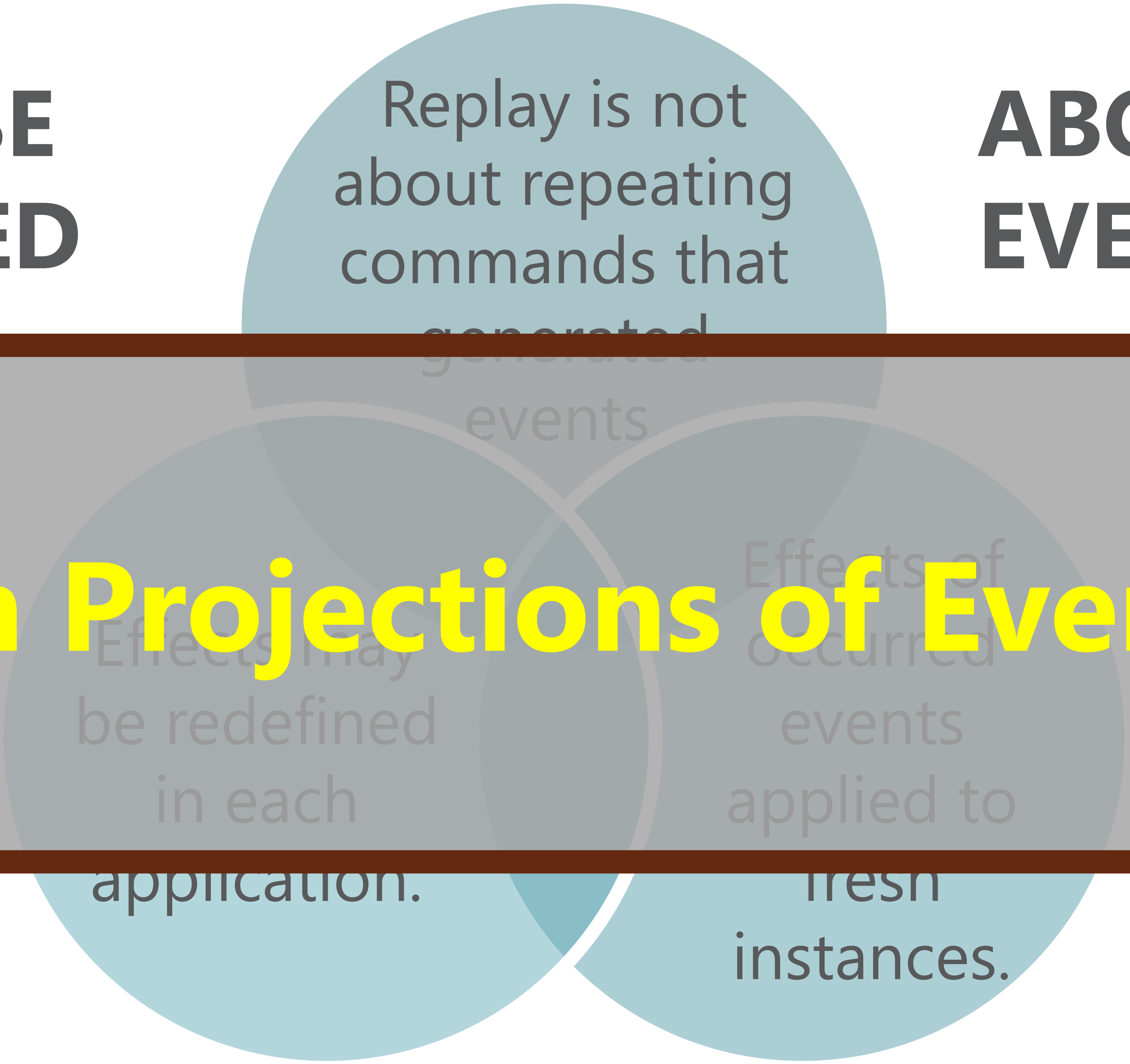
Rebuilding the state

```
public static Aggregate PlayEvents(String id, IEnumerable<DomainEvent> events)
    {
        var aggregate = new Aggregate(id);
        foreach (var e in events)
        {
            if (e is AggregateCreatedEvent)
                aggregate.Create(e.Data);

            if (e is AggregateUpdatedvent)
                aggregate.Update(e.Data);

            if (e is AggregateDeletedEvent)
                aggregate.Delete(e.Data);
        }
        return aggregate;
    }
}
```

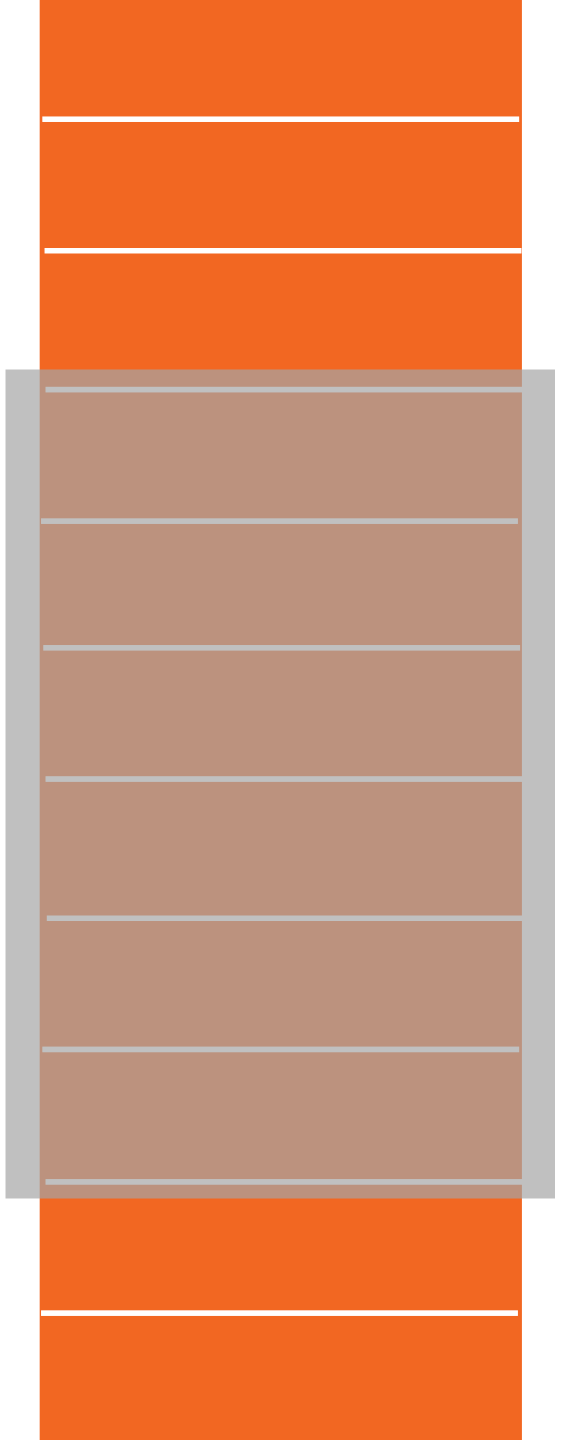
TO BE CONSIDERED ABOUT EVENT REPLAY


Replay is not about repeating commands that generated events.

Effects may be redefined in each application.

Effects of occurred events applied to fresh instances.

Custom Projections of Event Data



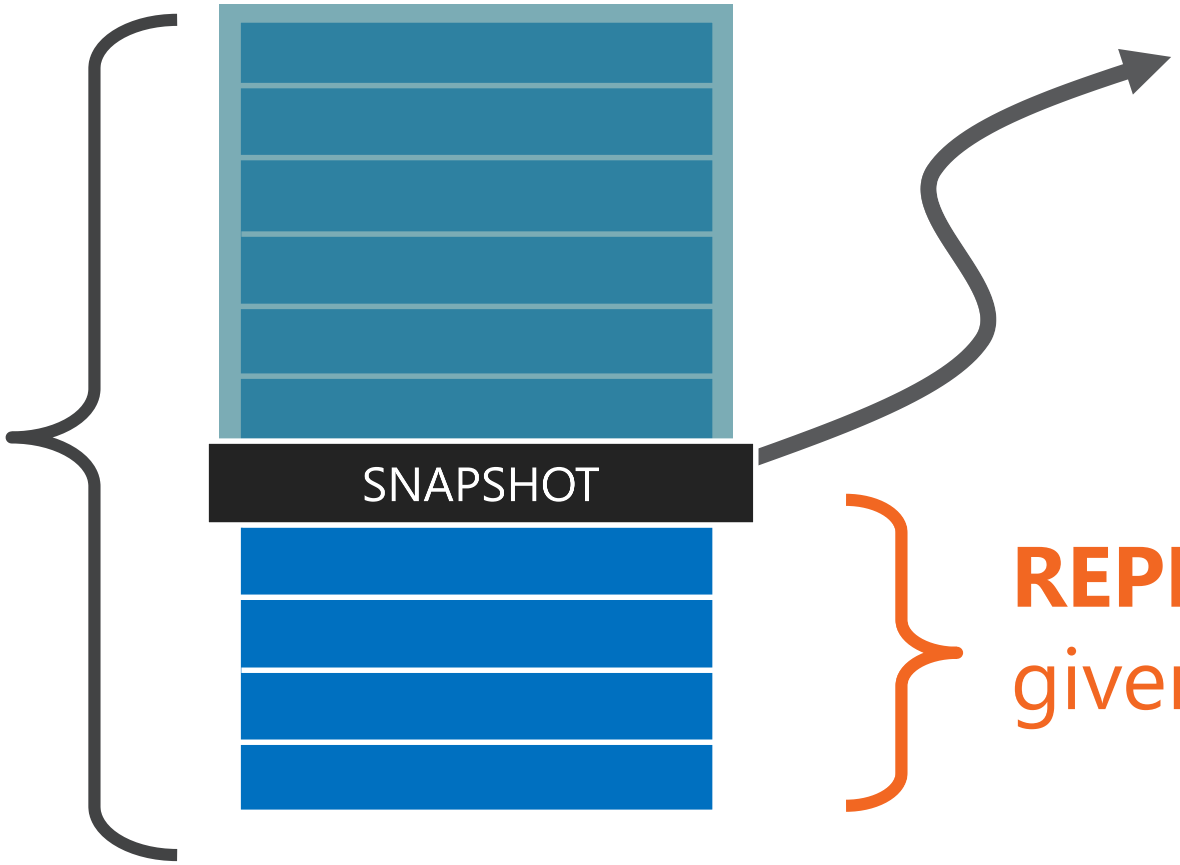
Replay just **these** events

Perform different calculations

Apply different forms of business logic

Effects of **immutability**

Should you have **performance** concerns?



Serialize state of aggregates to a given point in time

REPLAY events from a given snapshot

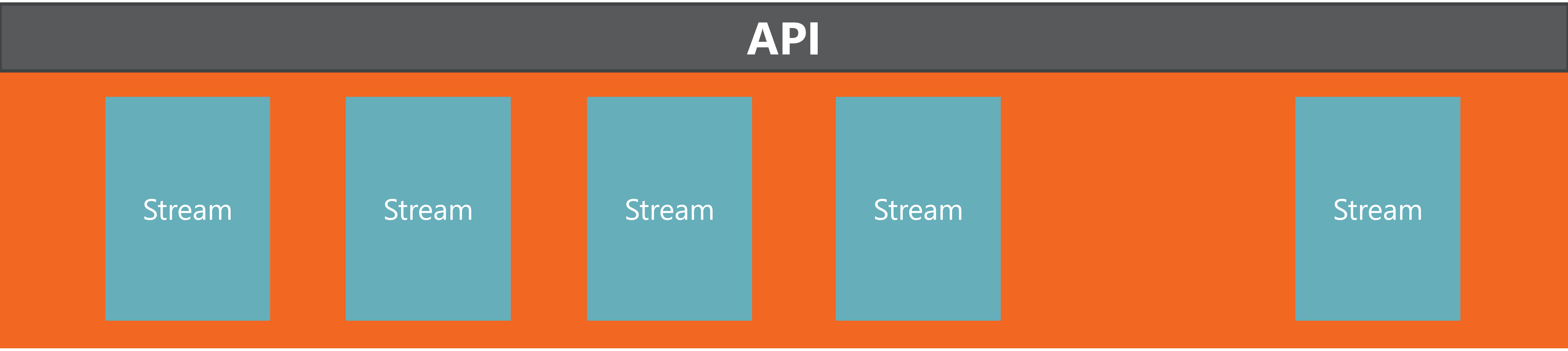
You can definitely arrange an event sourcing solution all by yourself.

But to store events effectively ad hoc tools may be better.

Event-based Data Stores

Event Store

http://geteventstore.com



Event Store Operations

Writing events to a stream

Reading events from a stream

Subscribing to stream to get updates

```
[
  {
     "eventId":   "fbf4a1a1-b4a3-4dfe-a01f-ec52c34e16e4",
     "eventType": "OrderCreatedEvent",
     "data":      { "orderId": "1", ... }
  }
]
```

Event Store Types of Subscriptions

Volatile	Catch-up	Persistent
Call back a function whenever an event is written to a given stream until the subscription is stopped	Call back a function from a given position up to the end and then turns into volatile.	Multiple consumers are guaranteed to receive at least one notification of events written possibly more.