The Event

The secret to building large scale distributed cloud apps.

Yves Goeleven

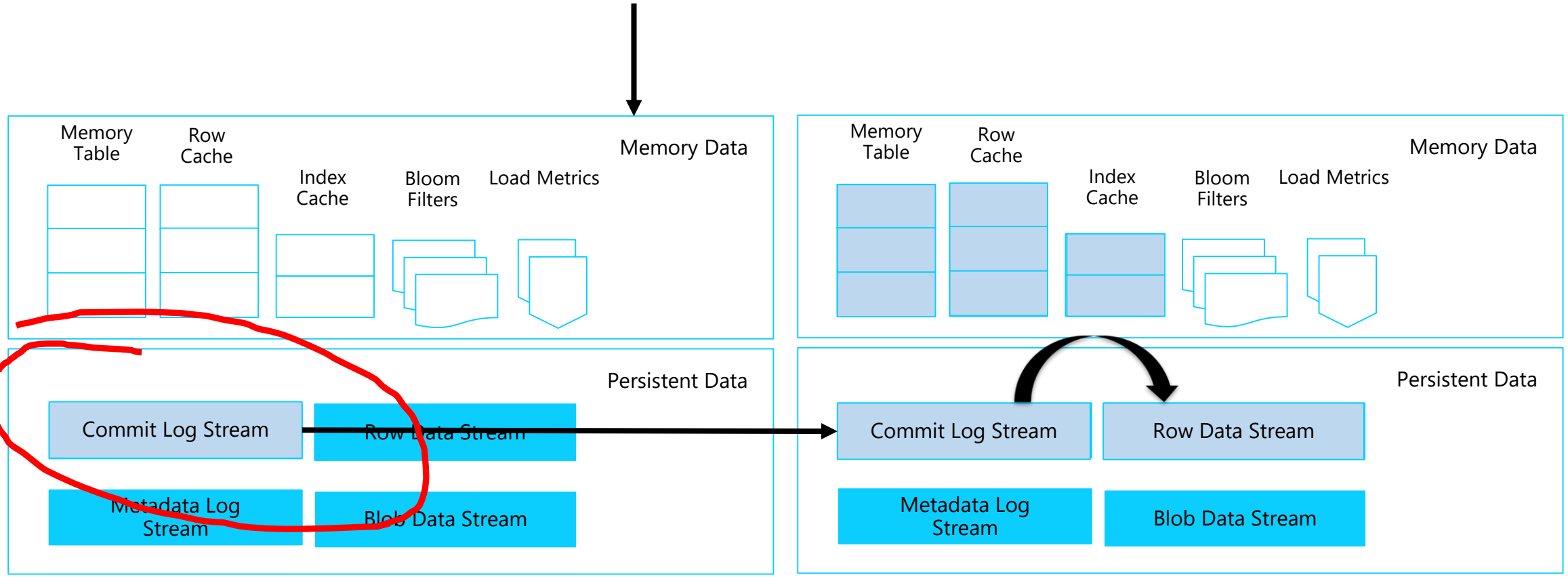
Solution Architect - Particular Software

- Shipping software since 2001
- Azure MVP since 2010
- Co-founder & board member AZUG
- NServiceBus & MessageHandler

Who attended my azure storage talk?

The secret

A log of what happened



How can I apply this concept to my own app?

Commit Log Entry => Event
Commit Log => Event Store

Agenda

Event Driven Architecture

- Definition
  - Event
  - Event Driven Architecture
  - Where it can fit
  - Benefits

- Patterns
  - Event Generators
  - Event Store
  - Event Sourcing
  - Event Channel
  - Event Processing
  - Event Projections
  - Event Transformations

What is an event?

Definition of an event

What is an event

A structured record

- Something happened
- Rooted in business domain

Well known structure

- Event Data Payload
- Context Metadata
  - When
  - What
  - Where
  - Who...

Immutable

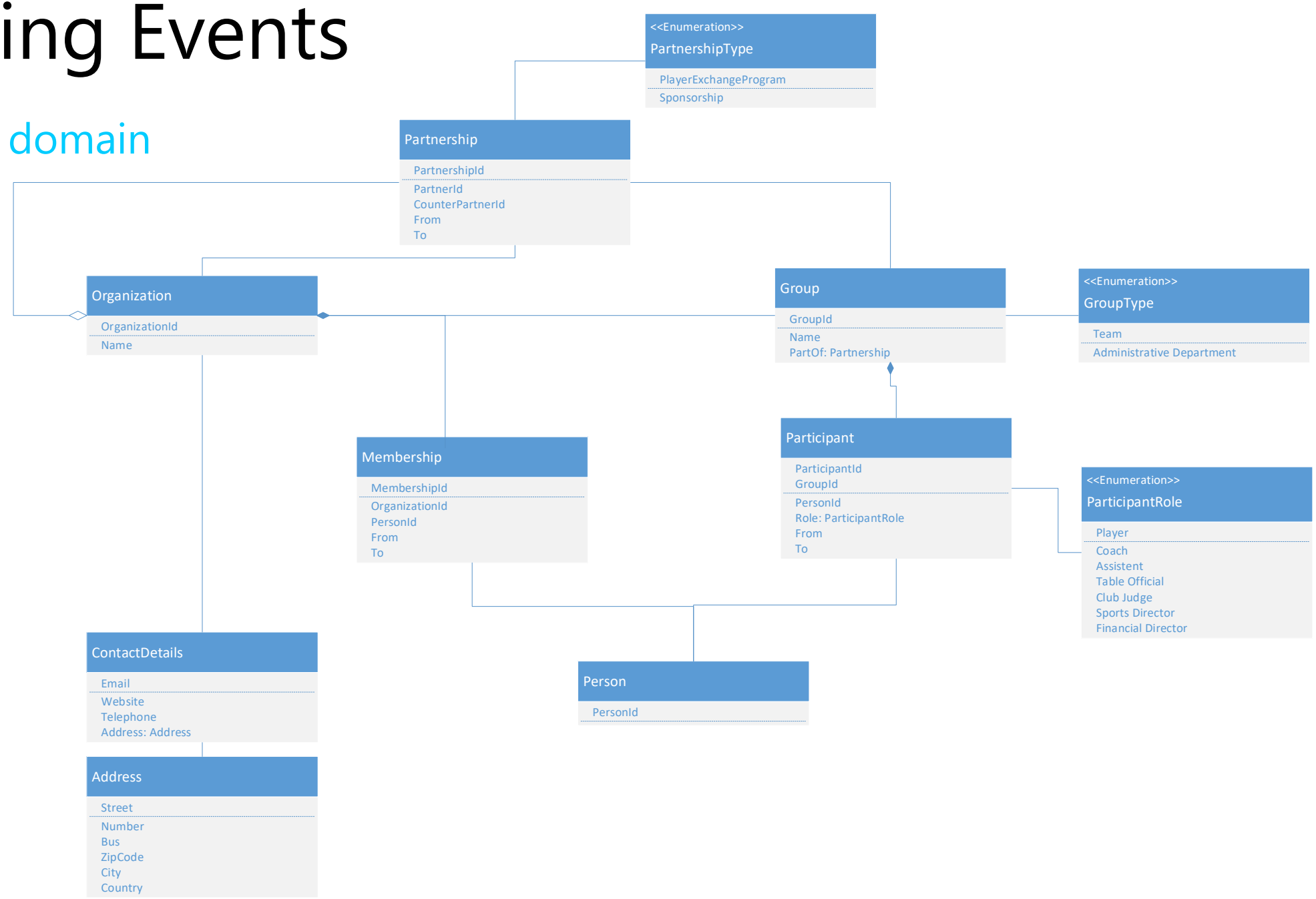
- Cannot change
- Safe to
  - broadcast
  - Store
  - Replicate

Self descriptive

- Described in terms of the domain model
- All relevant state is encapsulated
- Challenge: relevant inside vs outside

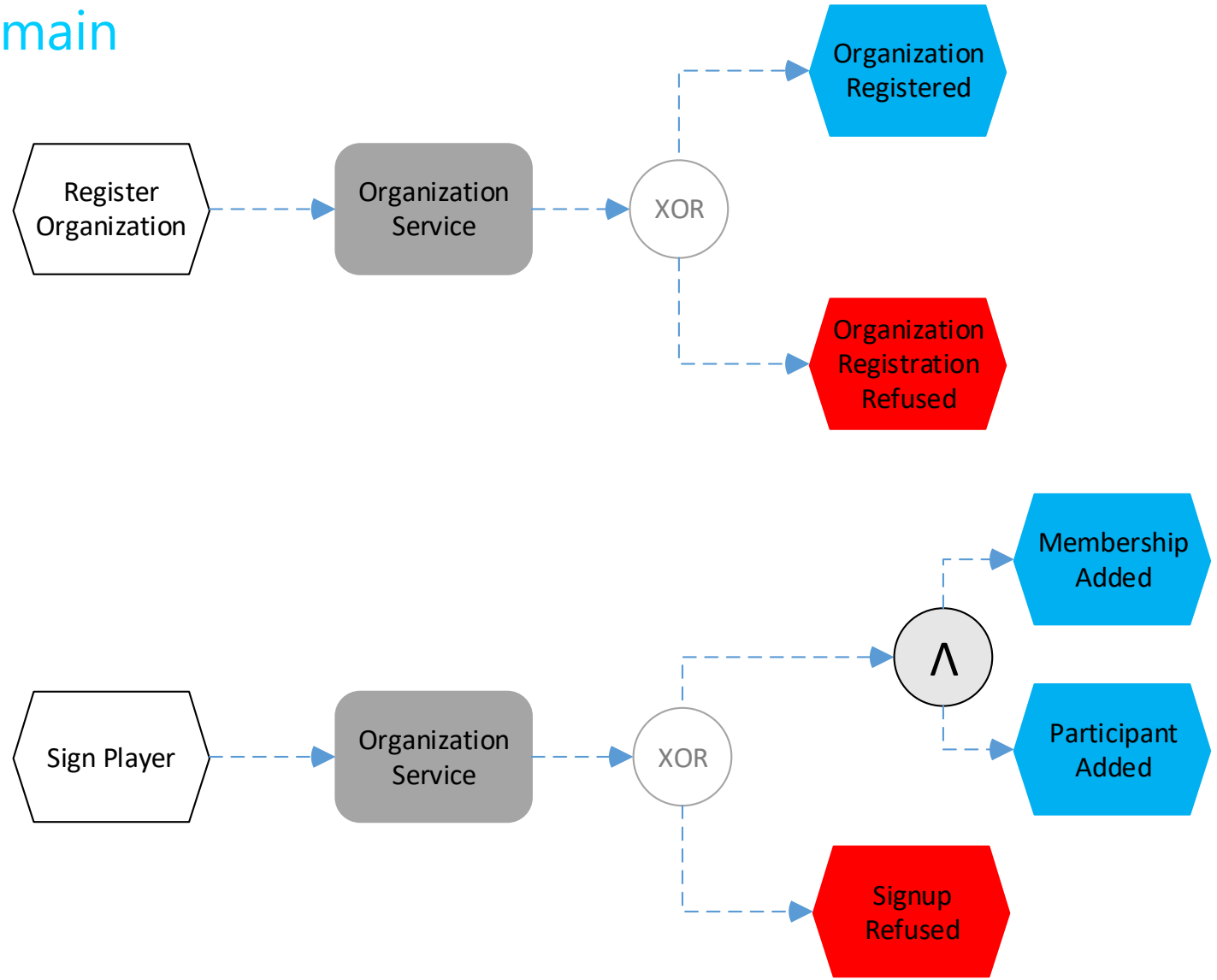
Defining Events

Rooted in domain



Defining Events

Derived from domain



Defining Event Structure

Well known structure

```
{
  context: {
     id: '36b6fc55-7809-4a38-b5d5-ae3e7faf615f',
     what: 'OrganizationRegistered',
     when: '9/1/2017 12:11:04 PM',
     who: 'Yves',
     why: {
        id: 'fcc050fc-fdef-42b4-a131-0f33dcfbcd91',
        what: 'RegisterOrganization',
        when: '9/1/2017 12:11:02 PM',
        who: 'Yves'
     }
   },
   organization: {
     id: 'a7623ed2-038d-49f7-84aa-18cb883c2930',
     name: 'Basket Lummen',
     contactDetails: {
        email: 'info@basketlummen.be',
        address: {
           street: 'Kerkstraat',
           number: 9,
           zipcode: 3560,
           city: 'Lummen’
}}}}
```

Event Driven Architecture Definition

What is an event driven architecture?

- *"Event-driven architecture (EDA), is a software architecture pattern promoting the production, detection, consumption of, and reaction to events."* – Wikipedia

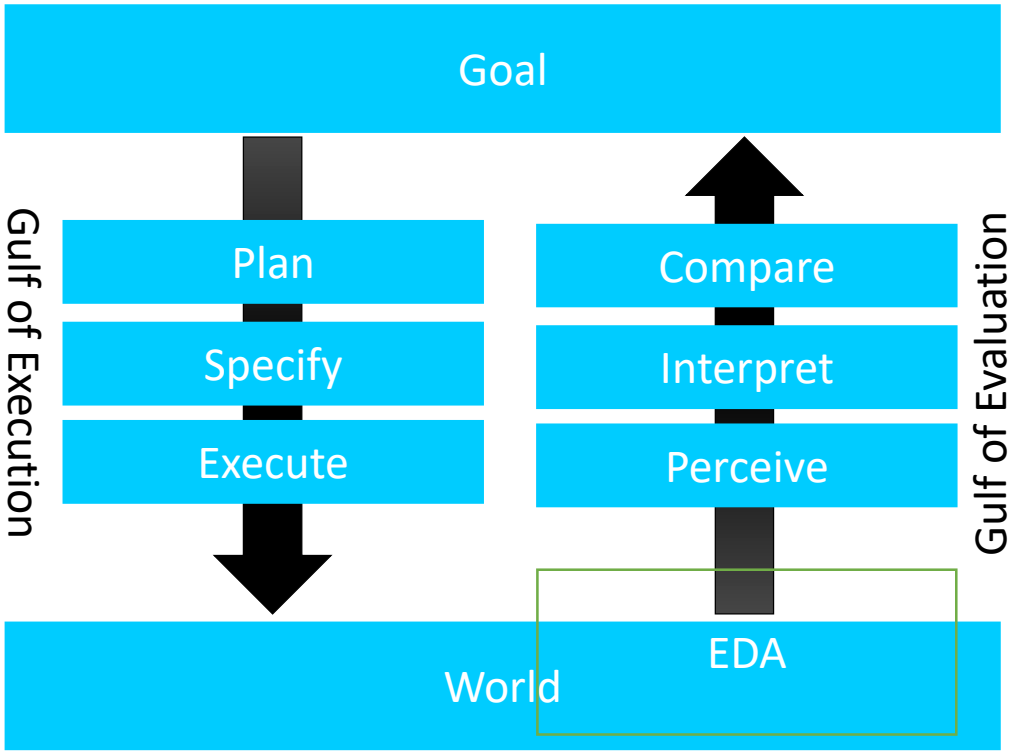
- **It's a mindset!** Not a predefined architecture or set of technologies.

- EDA patterns can be applied at various levels
  - Implementation level: Event driven design patterns used inside a service
  - System level: As top level architecture

Fitting EDA to UX

The 7 stages of action *

- Events originate in the ‘world’
  - In response to user commands
  - From entities in the world

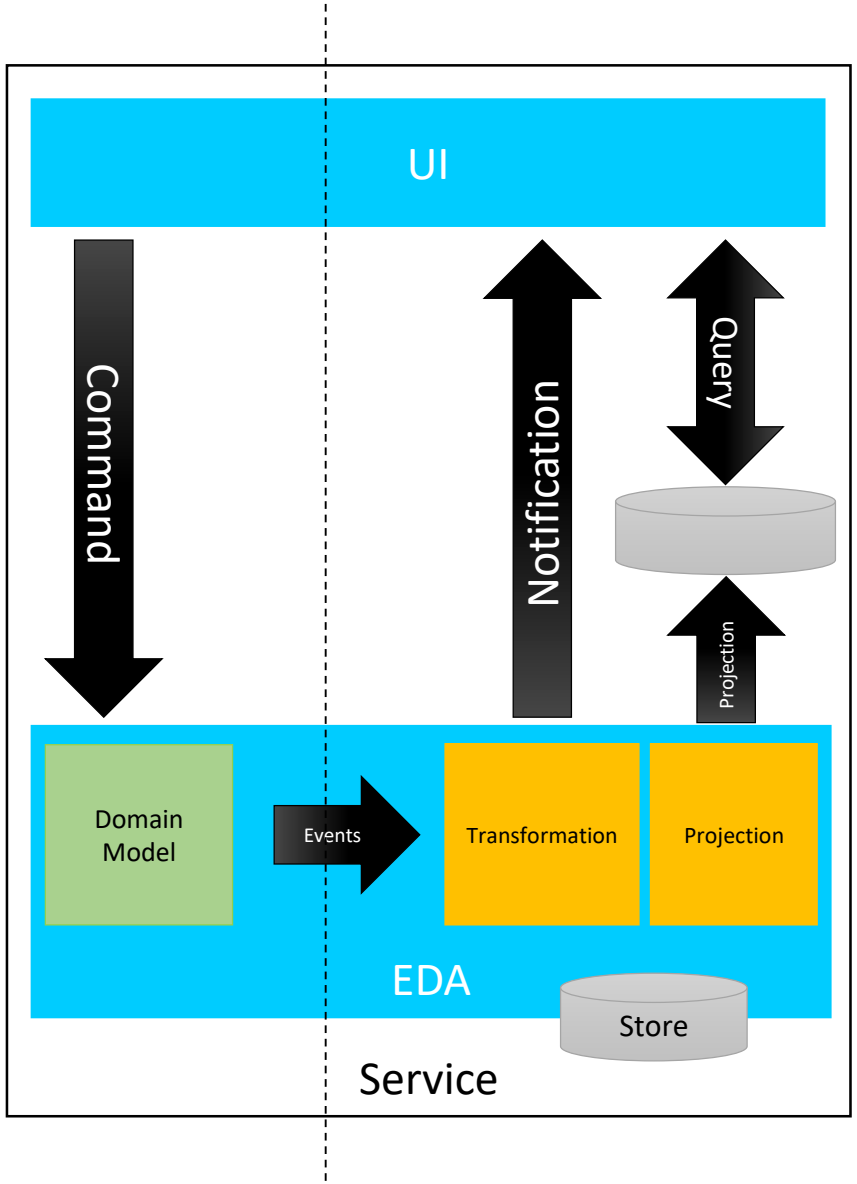


* Donald Norman: The Design Of Everyday Things

Filling the UX Gap

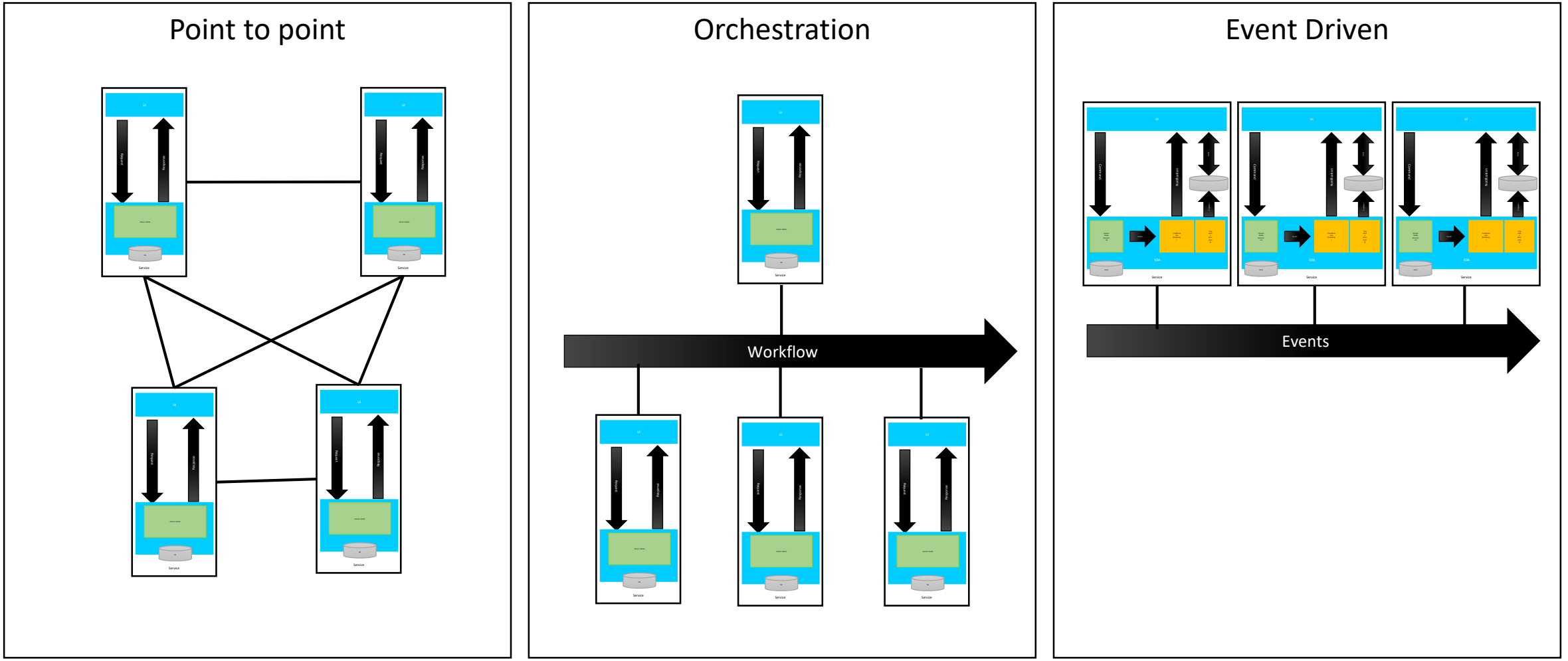
CQRS

- Command Query Responsibility Seggregation
- Different stack for
  - Execution (Command)
  - Evaluation (Query)
- EDA Patterns can bridge both stacks, inside a service



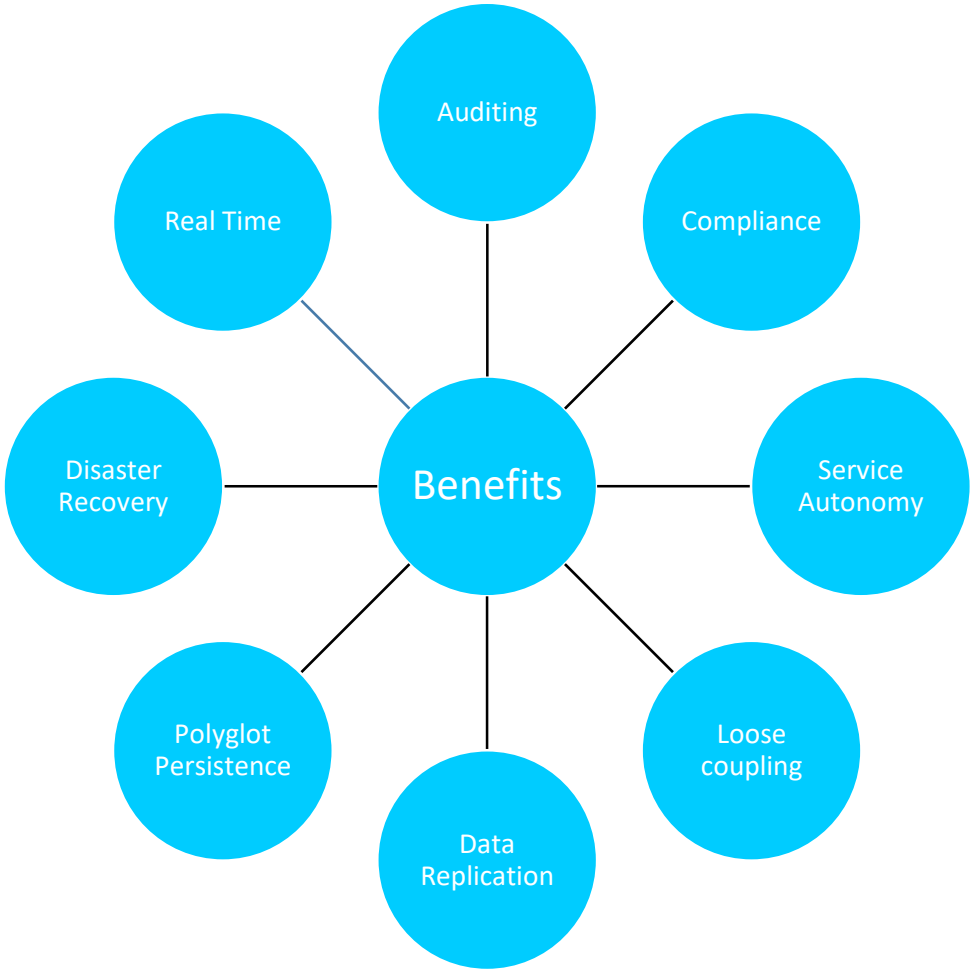
EDA as Top Level Architecture

Different flavors of SOA



Benefits of applying EDA

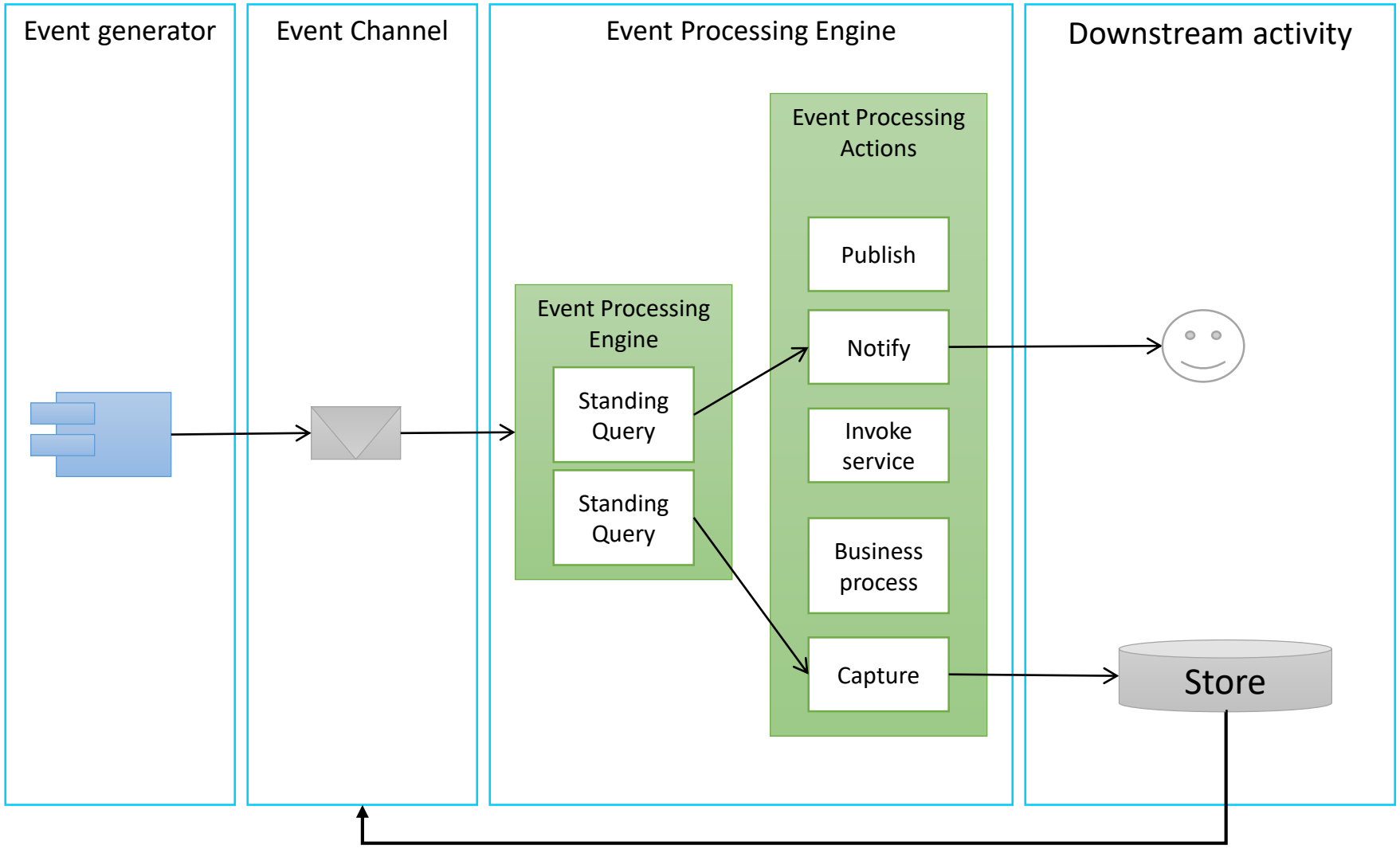
For large scale distributed cloud apps



Design Patterns

Overview

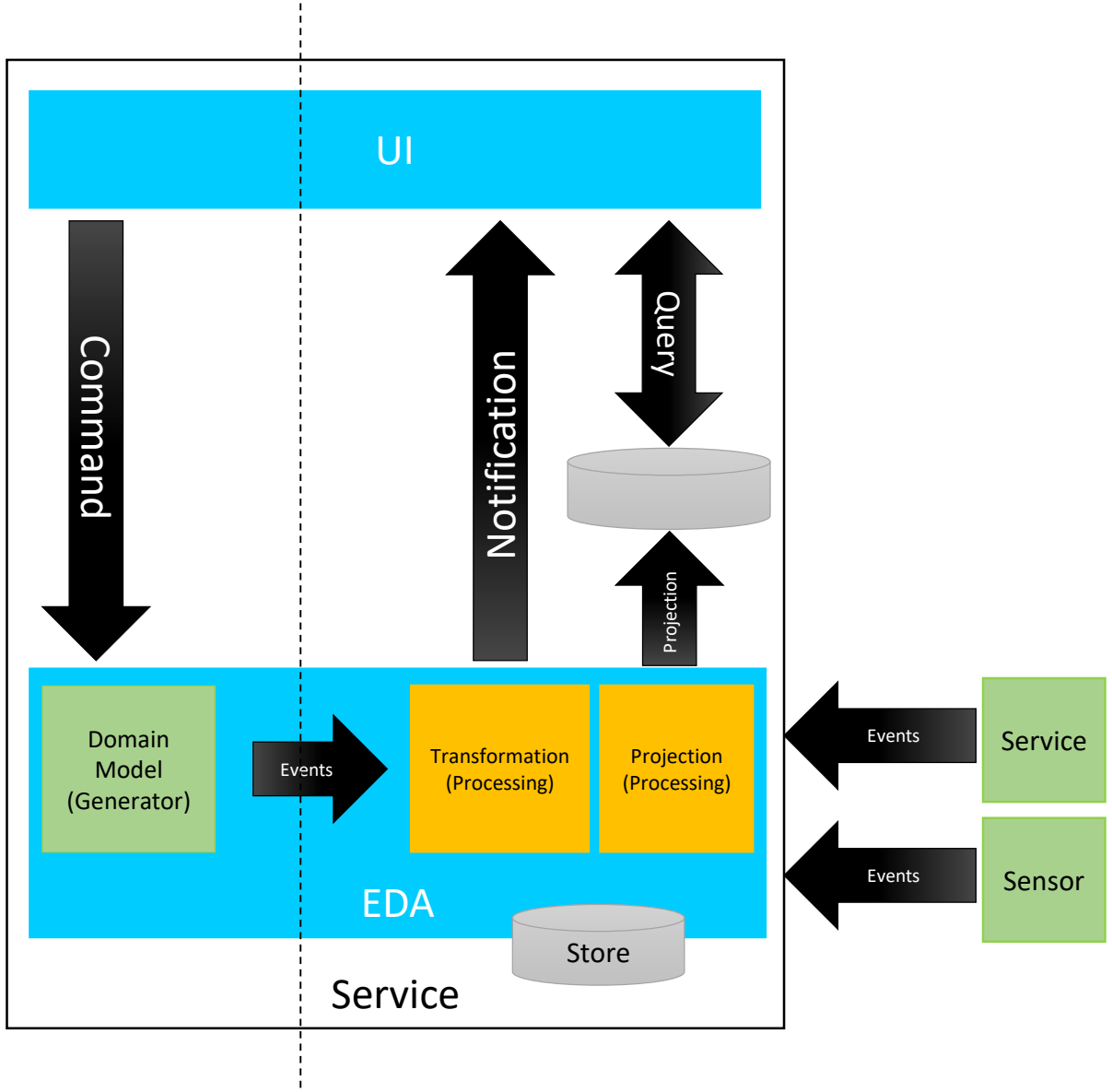
Event flow layers



Mapping to CQRS

Patterns applied

- Event Generators
  - Domain Model
  - Other services (& IoT)
- Event Processing
  - Projection logic
  - Notification logic
- Downstream activity
  - Event Store
  - Notification
  - DB update
- Event Channels

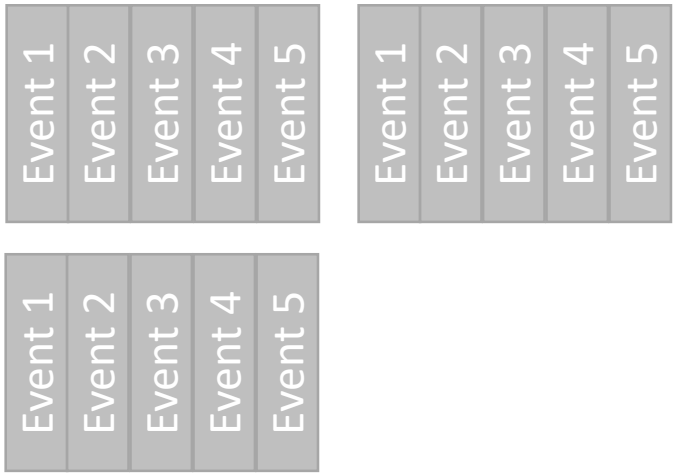


Store

Event Store

Store events for future use

- Partition by streams (e.g. type)
- Maintain order (append only)
- Store metadata & body
- Popular media
  - Files (e.g. GetEventStore, Eventhubs Archive, Kafka)
  - Tables (e.g. Azure Table Storage)
- Auditing & compliance



If you don't store your events you will have a hopefully consistent system,
not an eventually consistent one.

Event Store

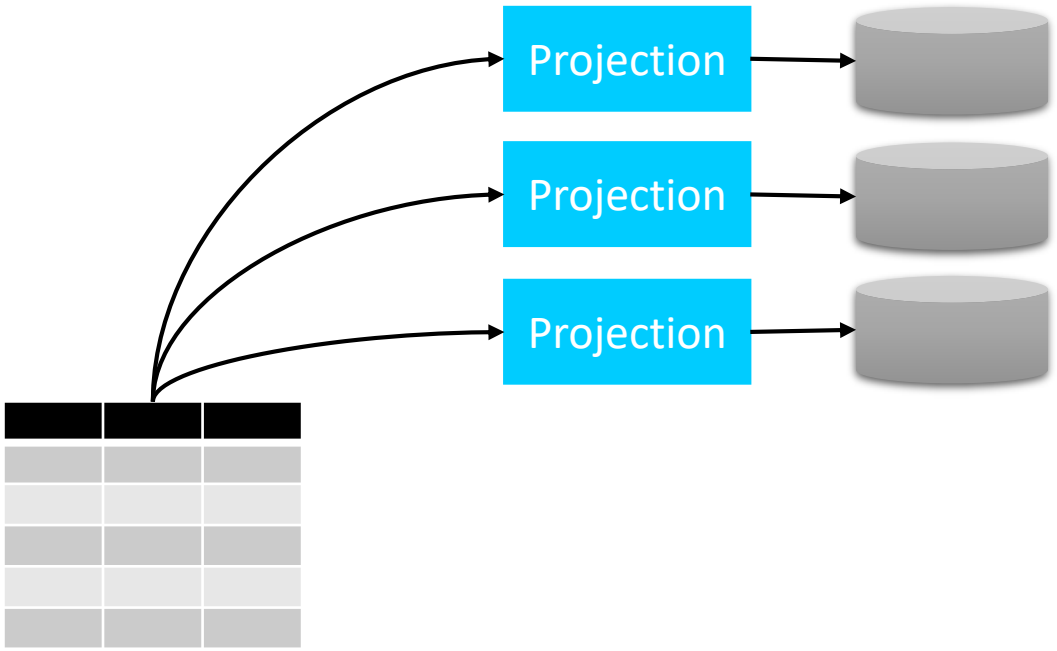
Example

PartitionKey	RowKey	Timestamp	Body	EventType	SourceId	Version
Organization	1_000001	2017-08-31T19:25:04	{ context: { ….	OrganizationRegistered	1	1
Organization	1_000002	2017-08-31T19:26:43	{ context: { ….	OrganizationRenamed	1	2
Organization	1_000003	2017-09-01T09:45:16	{ context: { ….	MemberAdded	1	3
Person	1_000001	2017-08-31T19:25:04	{ context: { ….	PersonRegistered	1	1
Person	1_000002	2017-08-31T19:25:5	{ context: { ….	ContactDetailsAdded	1	2

Replay stored events

To solve the following problems

- Disaster recovery
- Data replication
- Polyglot persistence
- Data versioning



Projection

Turns events into database records

```
public class MyProjection : IProjection<SearchEntry, OrganizationRegistered>
  {
     public void Project(SearchEntry record, OrganizationRegistered msg)
     {
        record.Id = msg.Organization.Id;
        record.Title = msg.Organization.Name;
        record.Description = msg.Organization.ContactDetails?.Address?.ToString();
        record.Type = "Organization";
     }
  }
```

Replay

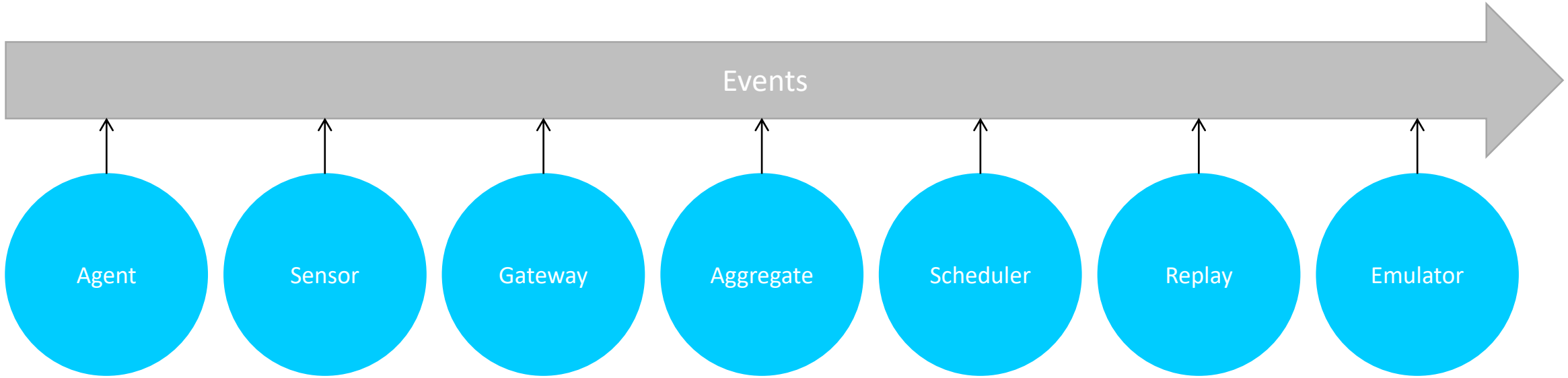
Stored Events

```
var configuration = new EventsourcingConfiguration();
configuration.UseEventSource(new AzureTableStorageEventSource(connectionString, table));
configuration.EnableProjections(typeof(MyProjection));
var projection = configuration.CreateProjector();

var records = await projection.Restore<SearchEntry>("Organization");

await search.Upsert(index, records);
```

Generators

Generators

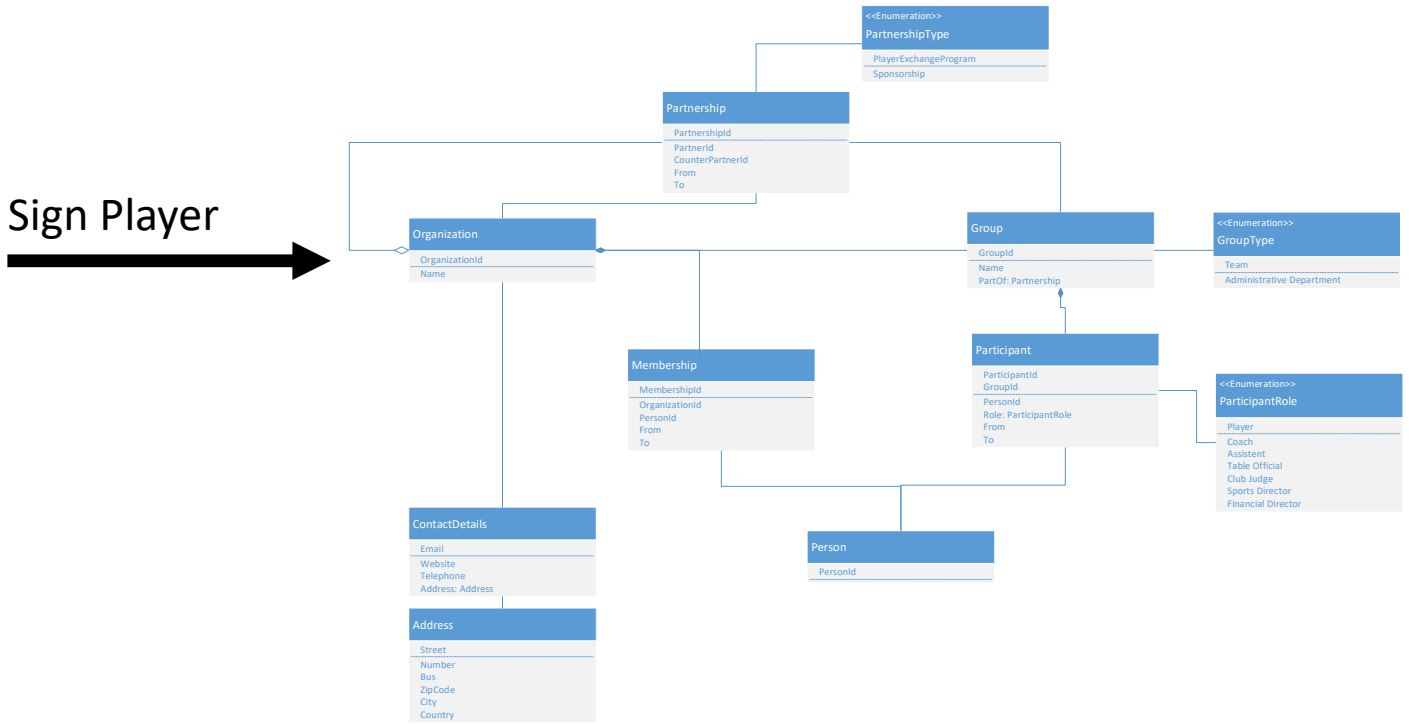
Any object that generates events



Aggregates

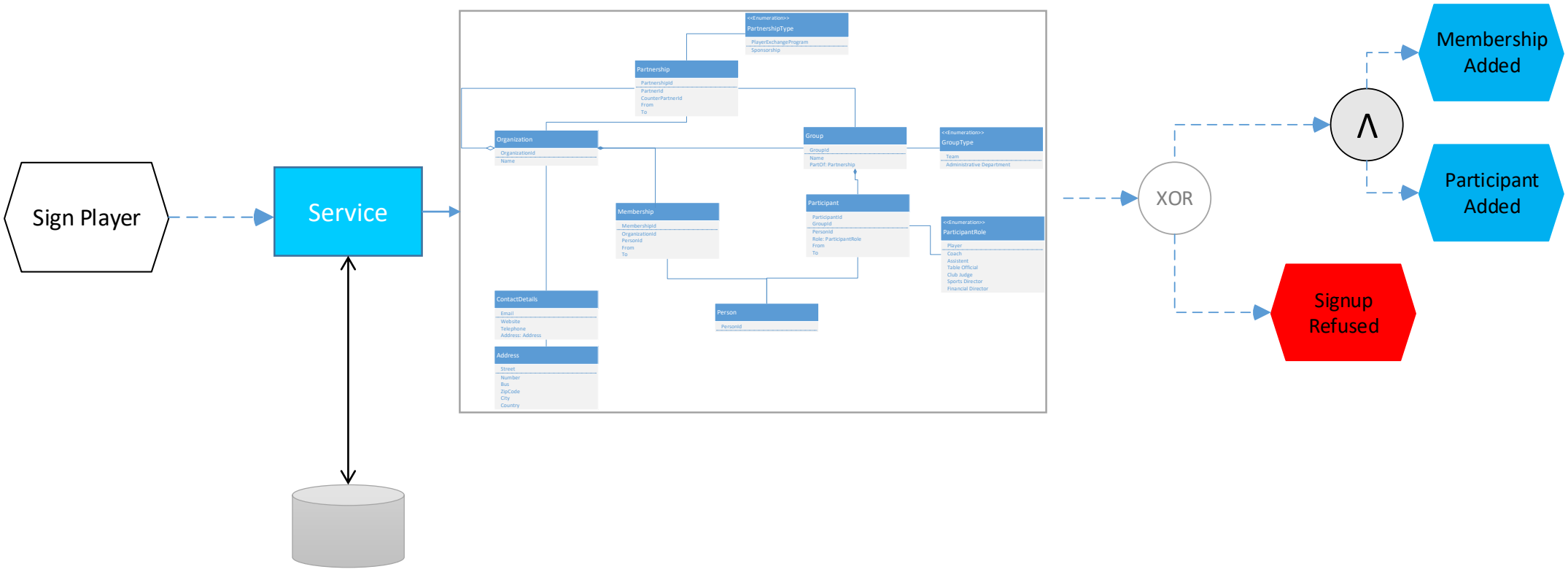
Transactional Boundary

- *"A DDD aggregate is a cluster of domain objects that can be treated as a single unit. Aggregates are the basic element of transfer of data storage - you request to load or save whole aggregates. Transactions should not cross aggregate boundaries."* – Martin Fowler



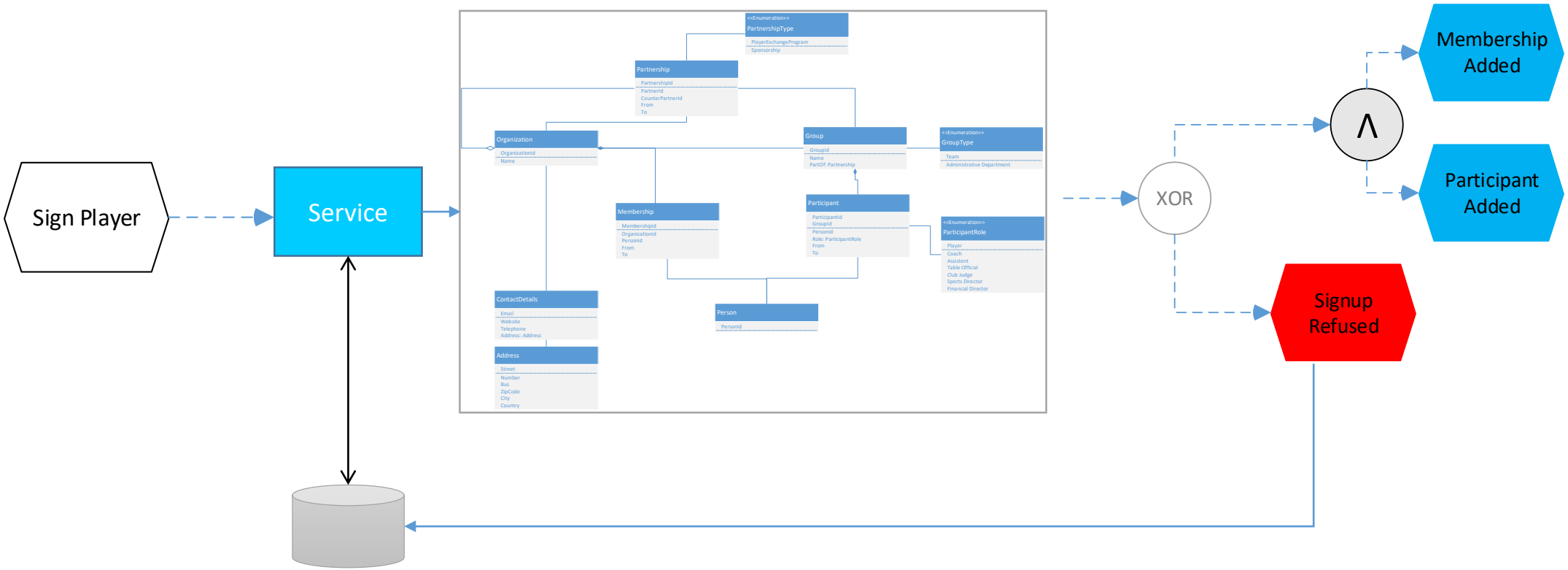
Aggregate as event generator

Events emitted as part of the transaction



Event Sourced aggregate

Aggregate state restored from the emitted events



Event Sourced Aggregate

Code Sample

```
var repository = configuration.CreateAggregateRepository();

var org = await repository.Get<Organization>(orgid);

org.Register(new { Name = "Basket Lummen" });

org.Rename(new { Name = "Basket Lummen 2" });

await repository.Flush();
```

Event Sourced Aggregate

Code Sample

```
public class Organization : EventSourced,
    IApply<OrganizationRegistered>,
    IApply<OrganizationRenamed>
  {
    public string Name { get; private set; }

    public void Register(dynamic msg)
    {
      // todo validate if command can be executed

      Emit(new OrganizationRegistered  {
                  Context = new Context { Id = Id, What = "OrganizationRegistered", Who = "Yves“ },
                  Organization = new Organization { Id = Id, Name = msg.Name }
      });
    }

    public void Rename(dynamic msg) {
      // todo validate if command can be executed

      Emit(new OrganizationRenamed   {
                Context = new Context {  Id = Id, What = "OrganizationRenamed", Who = "Yves“ },
                OldName = Name,
                NewName = msg.Name
      });
    }

    public void Apply(OrganizationRegistered msg)  {
      Name = msg.Organization.Name;
    }

    public void Apply(OrganizationRenamed msg)  {
      Name = msg.NewName;
    }
  }
```

Channel

Channels

Any technology used to transport events between producers and processors

Transactional channels

- Durable
- Every event is consumed and confirmed individually
- Pull
- At least once processing
- Failure to process an event leads to server side retry
- E.g. Queues & Topics

Streaming channels

- Durable
- Events are consumed in batch and confirmed in batch (moving cursor)
- Pull
- At least once processing
- No individual retry, entire batch need to be retried
- E.g. EventHubs & IoTHubs

Volatile channels

- Volatile
- Every event is consumed individually
- Push
- At most once processing
- Failure to process leads to loss of event
- Requires auto scaleout
- Requires client side retry logic to guarantee delivery
- E.g. Eventgrid, SignalR, Webhooks

Pushing Events to Channel

From the eventsourced aggregate

```
var config = new HandlerRuntimeConfiguration();
var dispatcher = config.EventProcessingPipeline()
                            .EnableDispatching()
                                .AddDirectRoute("mytopic", SupportedTransportTypes.Topic, serviceBusConnectionString)
                        .GetDispatcher();

var runtime = await HandlerRuntime.Create(config);
await runtime.Start();

var configuration = new EventsourcingConfiguration();
configuration.UseEventSource(new AzureTableStorageEventSource(storageConnectionstring, "EventSource"));
configuration.UseChannel(msgs => dispatcher.Dispatch(msgs));
```

Processing

Event Processing

3 styles

Simple Event Processing

- Real Time
- Individual event processing
- Action per event

Stream Event Processing

- Real Time
- Assumed ordered in time
- All messages in a given time window are processed at once
- Aggregations, transformations, outlier detection, etc...
- Action per outcome

Complex Event Processing

- Historical analysis
- Multiple event stores
- Unordered
- Event correlation & causality detection
- Fault detection

Real Time Projection

Example of simple event processing

```
var entry = await search.Get<SearchEntry>(settings.Index, m.Context.Id) ?? new SearchEntry();

projection.Apply(entry, Cast(context.Message, GetType(m.Context.What)));

await search.Upsert(settings.Index, new[] { entry });
```

Transformation

Example of stream event processing

```
public class Transformation : IStandingQueryBuilder
{
        public IObservable<ActionableContext> Build(IObservable<IProcessingContext> stream)
        {
                return transformationQuery = from c in stream select Transform(c);
        }

        private ActionableContext Transform(IProcessingContext c)
        {
           dynamic m = c.Message;
           return new ActionableContext(c)
           {
              Message = new {  What = m.Context.What, Id = m.Context.Id,
                 Name = m.Organization != null ? m.Organization.Name : m.NewName
              },
              AutoComplete = true
           };
        }
}
```

Wrapup

Resources

On GitHub & MyGet

- Sample code: https://github.com/MessageHandler/AzureSaturday
- Libraries: https://www.myget.org/F/messagehandler/api/v3/index.json

Q&A