# Event Sourced Story

CQRS

ES

DDD

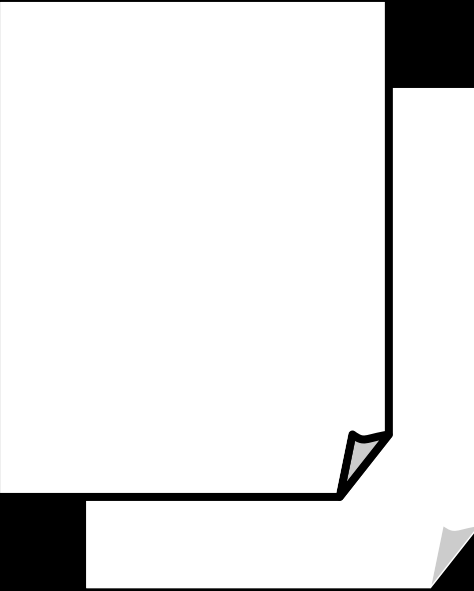
EDA

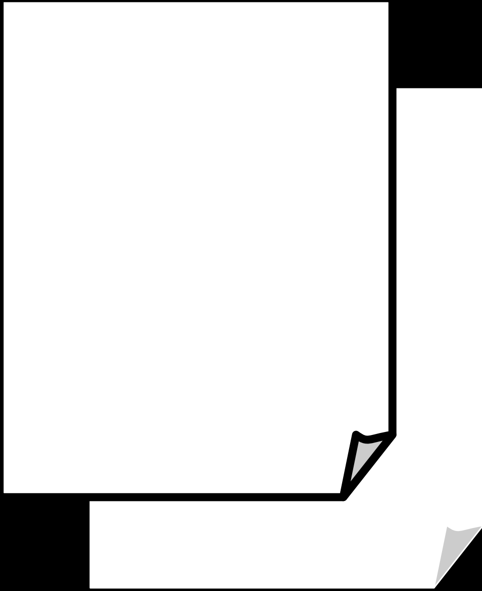
CQRS

ES

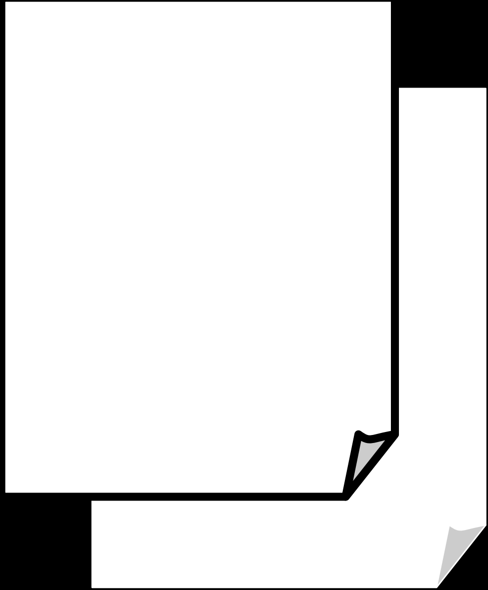
DDD

EDA



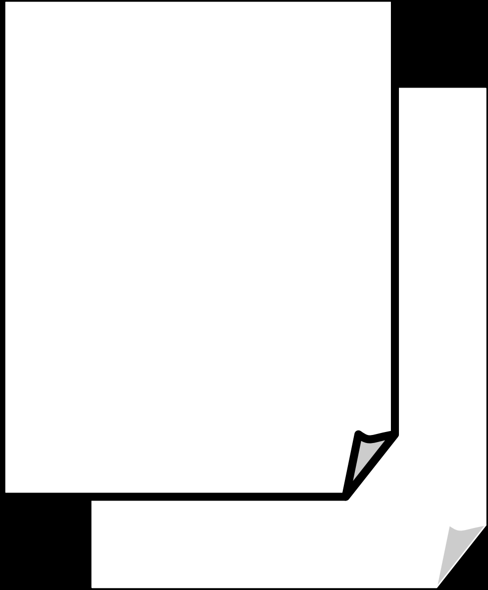


| id | | |
|---|---|---|
| | | |

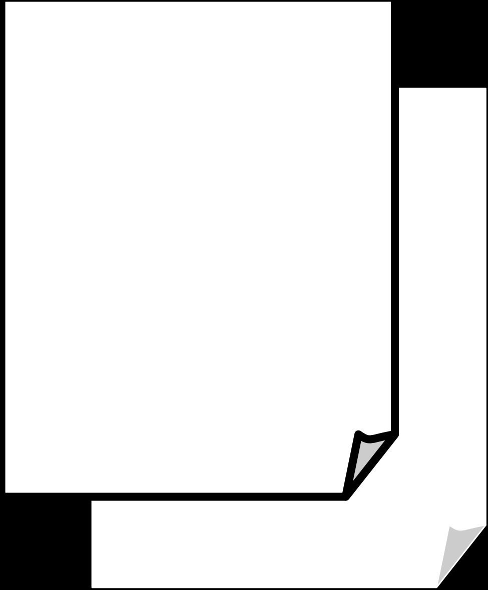


| id | | |
|---|---|---|
| | | |

| event_id | | |
|---|---|---|
| | | |
| | | |
| | | |



| event_id | | |
|---|---|---|
| | | |
| | | |
| | | |



| event_id | | |
| --- | --- | --- |
| | | |
| | | |
| | | |

# Aggregate

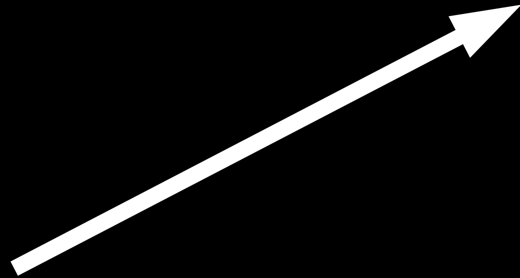
# Aggregate

- Event handlers

# Aggregate

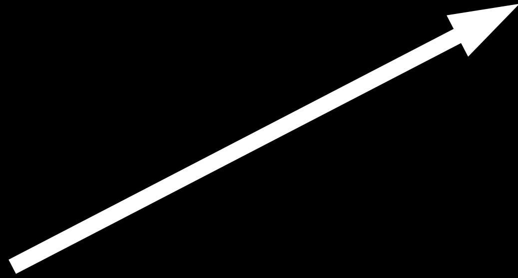
- Event handlers
- Commands

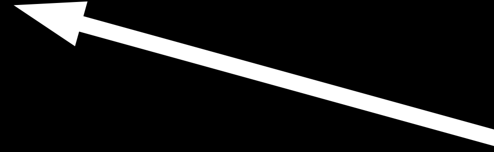
Aggregate



Entities

Aggregate



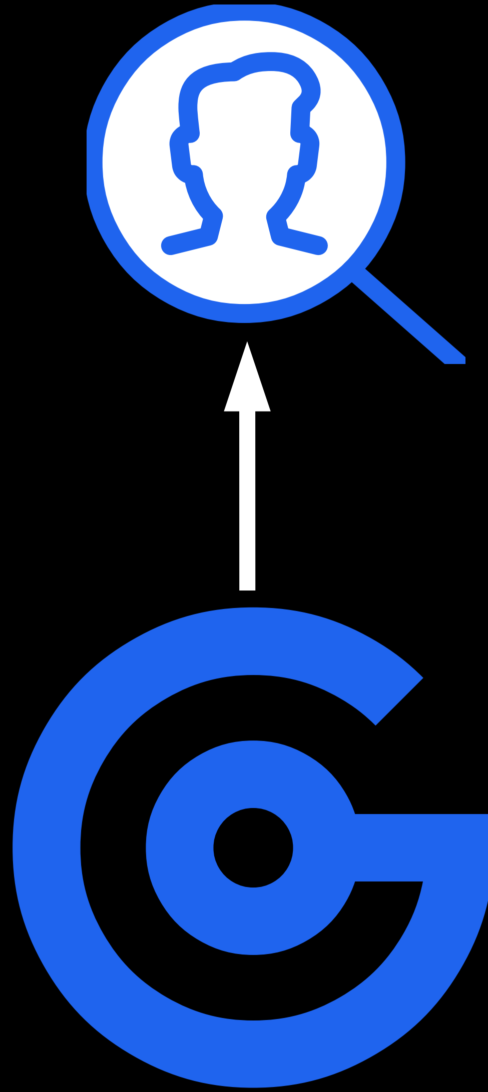


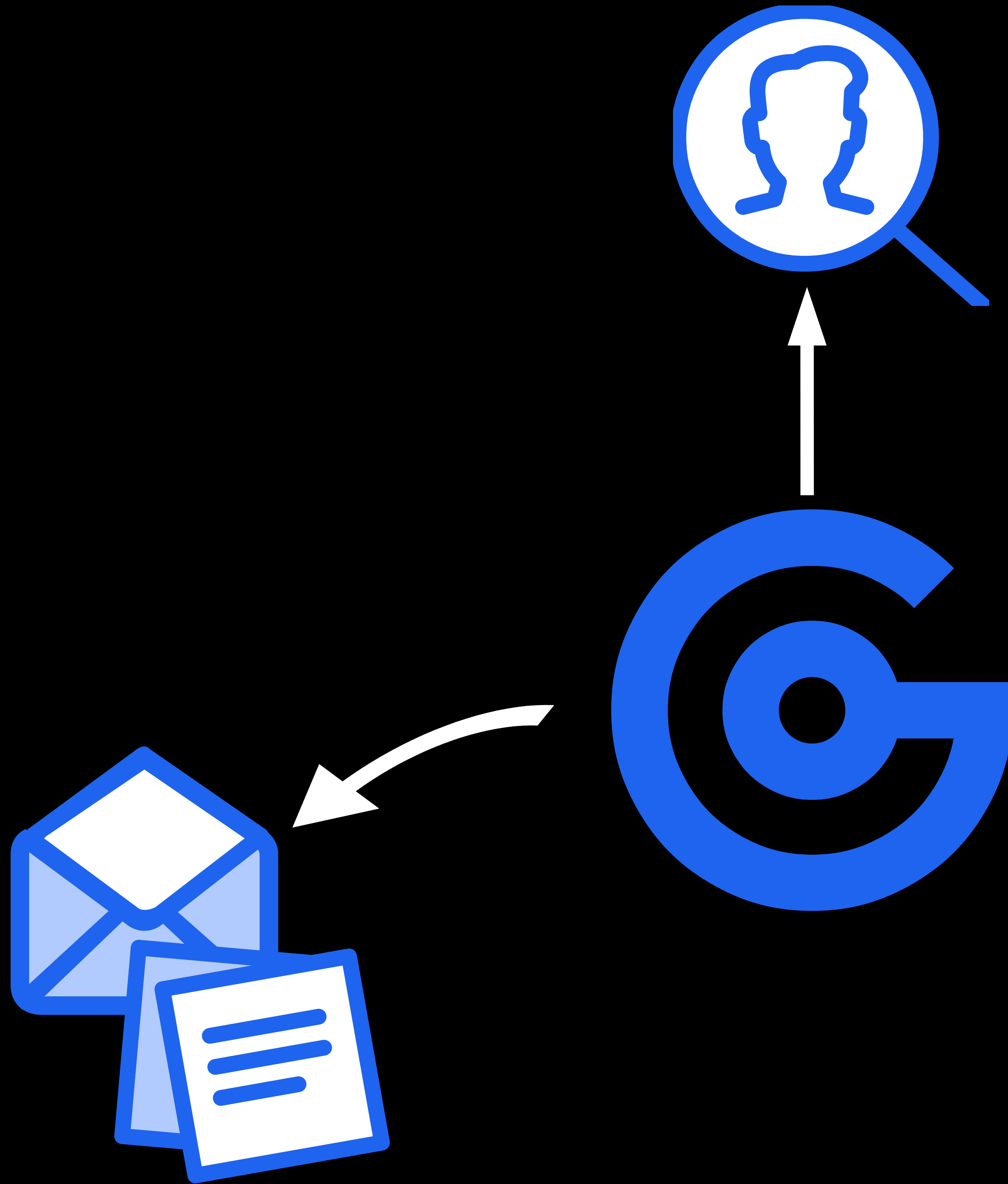
Entities

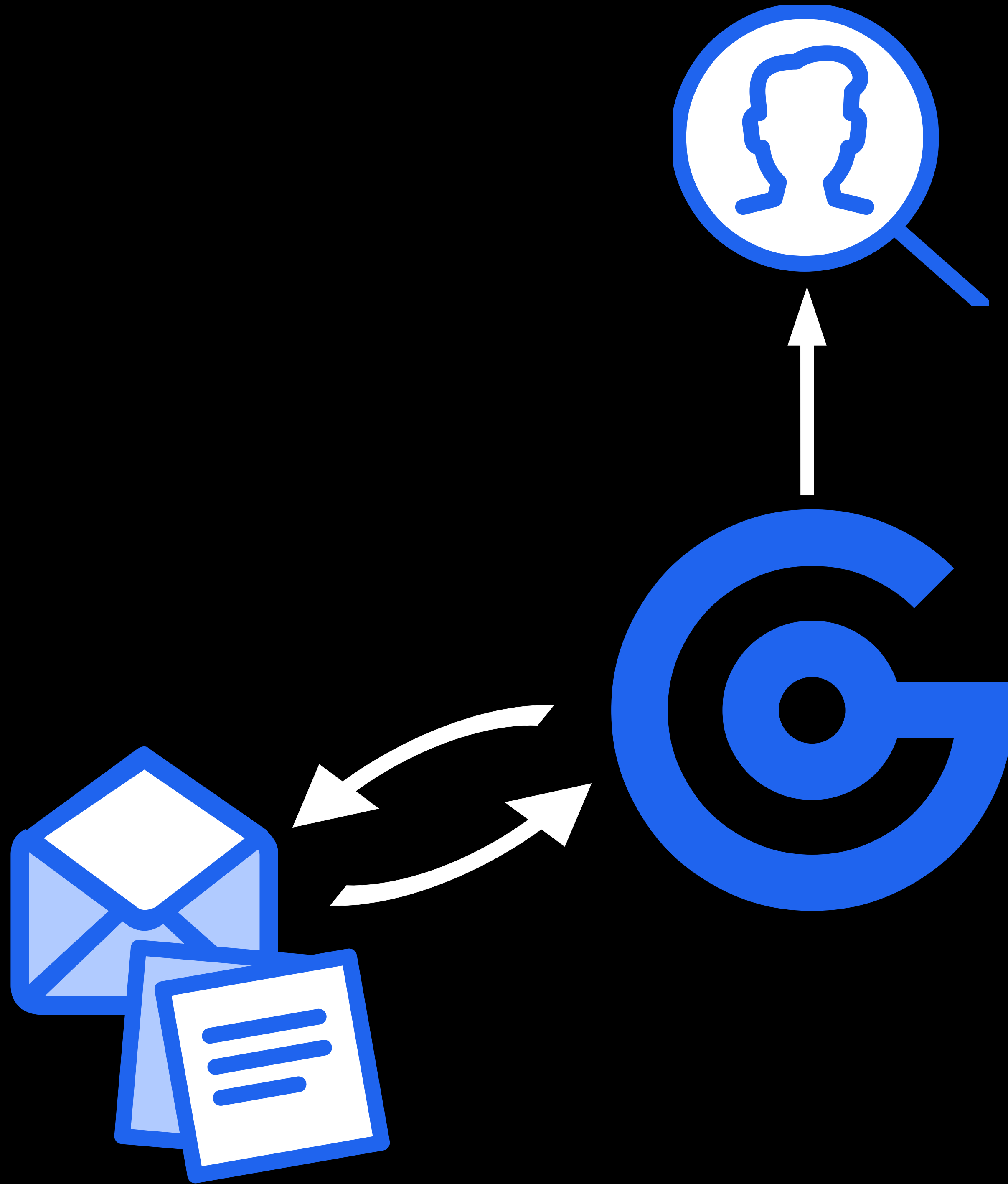
Value objects

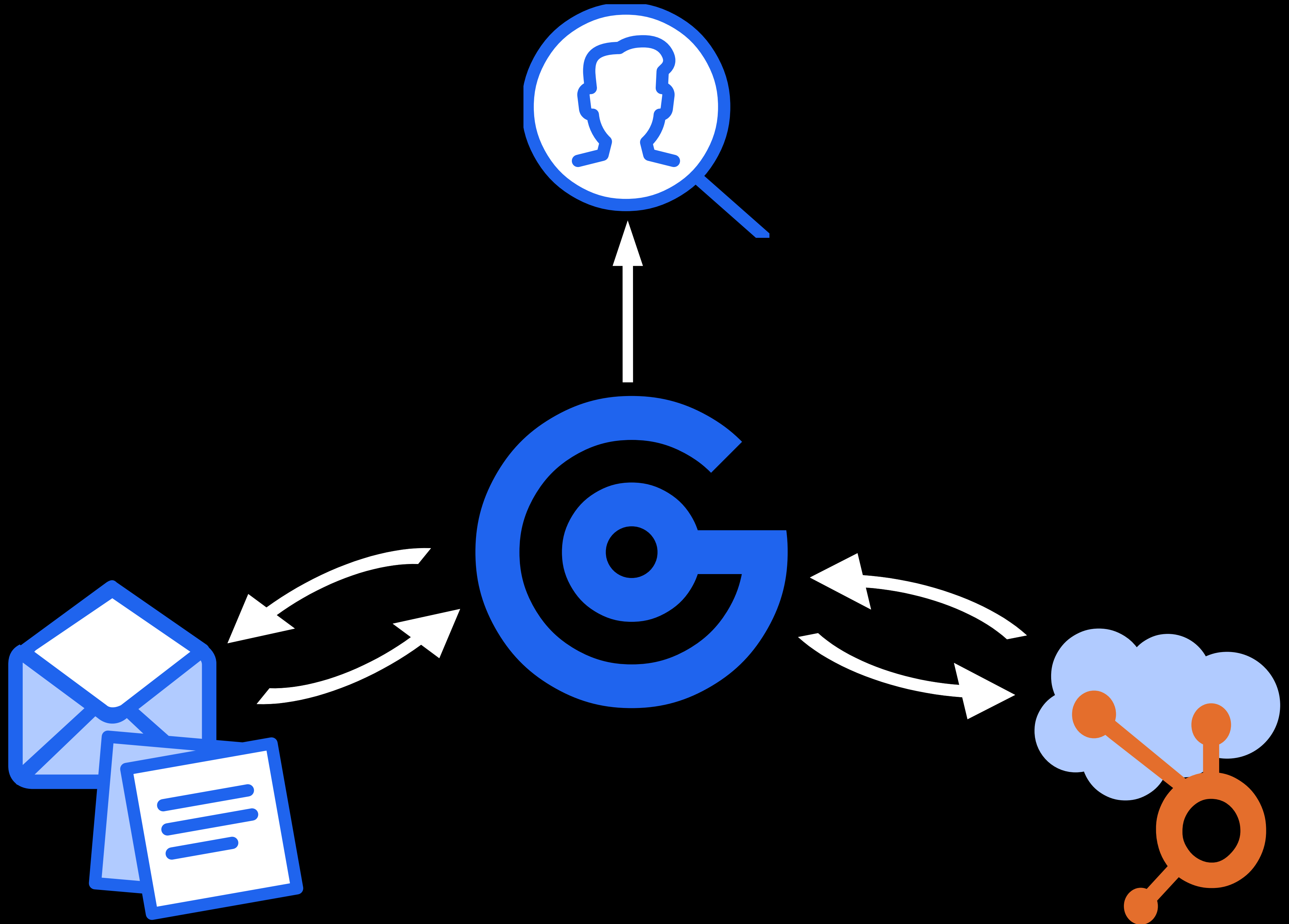
Why, tho?

Growbots









# Relevant

- Web application

# Relevant

- ~~Web application~~

# Relevant

- ~~Web application~~
- Web services

# Relevant

- ~~Web application~~
- ~~Web services~~

# Relevant

- ~~Web application~~
- ~~Web services~~
- Using (mostly) RDBMSs as storage

# Relevant

- ~~Web application~~
- ~~Web services~~
- ~~Using (mostly) RDBMSs as storage~~

# Conway's Law

# Conway's Law

organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.

-M. Conway

Growbots

# Problems

# Problems

- Complicated lifecycle of objects

# Problems

- Complicated lifecycle of objects
- Business logic based on events

# Problems

- Complicated lifecycle of objects
- Business logic based on events
- Issues hard/impossible to trace

# Problems

- Complicated lifecycle of objects
- Business logic based on events
- Issues hard/impossible to trace
- Need to expose history

# Hard part

# Ubiquitous language

# Domain, events, aggregates

```
class MyEvent:
    def __init__(self, ...):

    @classmethod
    def unserialize(cls, ...) ->
“MyEvent”:

    def serialize(self) -> dict:
```

```
class MyEvent:
    def __init__(self, ...):

    @classmethod
    def unserialize(cls, ...) ->
“MyEvent”:

    def serialize(self) -> dict:
```

```
class MyEvent:
    def __init__(self, ...):

    @classmethod
    def unserialize(cls, ...) ->
“MyEvent”:

    def serialize(self) -> dict:
```

```
class MyEvent:
    def __init__(self, ...):

    @classmethod
    def unserialize(cls, ...) ->
“MyEvent”:

    def serialize(self) -> dict:
```

```
class MyEvent:
    def __init__(self, ...):

    @classmethod
    def unserialize(cls, ...) ->
"MyEvent":

    def serialize(self) -> dict:
```

```
class MyAggregate:
    @classmethod
    def create(cls, ...) ->
“MyAggregate”:

    def do_stuff(self):

    def apply_event(self, event):
```

```
class MyAggregate:
    @classmethod
    def create(cls, ...) ->
“MyAggregate”:

    def do_stuff(self):

    def apply_event(self, event):
```

```
class MyAggregate:
    @classmethod
    def create(cls, ...) ->
"MyAggregate":

    def do_stuff(self):

    def apply_event(self, event):
```

```
class MyAggregate:
    @classmethod
    def create(cls, ...) ->
“MyAggregate”:

    def do_stuff(self):

    def apply_event(self, event):
```

```
class MyEvent:
    def __init__(self, ...):

    @classmethod
    def unserialize(cls, ...) ->
"MyEvent":

    def serialize(self) -> dict:

class MyAggregate:
    @classmethod
    def create(cls, ...) -> "MyAggregate":

    def do_stuff(self):

    def apply_event(self, event):
```

```
class Aggregate:
    def pop_pending_events(self)
-> List[Event]:
```

```python
class MyAggregate2:
    def do_stuff(self):
        ...
        db.session.add(events)
```

# Event store

- ID (integer, auto increment)

# Event store

- ID (integer, auto increment)
- aggregate ID

# Event store

- ID (integer, auto increment)
- aggregate ID
- version (integer, from app)

# Event store

- ID (integer, auto increment)
- aggregate ID
- version (integer, from app)
  - UNIQUE(aggregate ID, version)

# Event store

- ID (integer, auto increment)
- aggregate ID
- version (integer, from app)
  - UNIQUE(aggregate ID, version)
- tenant ID

# Event store

- ID (integer, auto increment)
- aggregate ID
- version (integer, from app)
  - UNIQUE(aggregate ID, version)
- tenant ID
- payload (JSON/unicode)

# Event store, optional

- event type (unicode)

# Event store, optional

- event type (unicode)
- correlation ID

# Event store, optional

- event type (unicode)
- correlation ID
- timestamp

# Event store, numbers

- 250M events, 360 GB
  - indices: 46 GB
- 95th percentile for fetching an aggregate: 50-100 ms
- 4-8 new events/s

# Ordering

# Concurrency

- pessimistic (explicit locks)

# Concurrency

- pessimistic (explicit locks)
- optimistic (unique index over aggregate, version)

# Immutability

# Projections

# Projections

## Events

| id | event |
| --- | --- |
| | |
| | |
| | |
| | |

## Read model

| | | | |
| --- | --- | --- | --- |
| | | | |
| | | | |
| | | | |
| | | | |

# Eventual consistency

# Testing

```
def test_my_event_serialize():
    ...
    assert event.serialize() == …

def test_my_event_unserialize():
    ...
    assert event.unserialize() ==
…
```

# Testing

```
def test_my_agg_do_stuff():
    # Given aggregate’s state:
    agg = ...

    agg.do_stuff()

    # Check events:
    assert agg._pending_events ==
...
```

# Testing

```
def test_my_agg_stuff_done():
    # Given aggregate’s state:
    agg = ...
    # and some event:
    event = ...

    agg.apply_event(event)

    # Check new state:
    assert agg....
```

# Events evolution

# Events evolution

- payload as JSON/string

# Events evolution

- payload as JSON/string
- new event type, e.g. stuff_done_v2
  - unless additive-only

# Events evolution

- payload as JSON/string
- new event type, e.g. stuff_done_v2
  - unless additive-only
- tech debt factory

# Data inspection

# Introducing ES
# in an existing app

# Stuff we don't even

- Snapshots
- Compensating events
- ES library/framework

# Related

- Domain-Driven Design
- Repository pattern
- Layered architecture
- Command-Query Responsibility Segregation
- Event-Driven Architecture

# Cons

complexity

# Pros

- History as first-class citizen

# Pros

- History as first-class citizen
- Simpler implementation of complicated business logic

# Pros

- History as first-class citizen
- Simpler implementation of complicated business logic
- Temporal thinking

# Pros

- History as first-class citizen
- Simpler implementation of complicated business logic
- Temporal thinking
- Interactions awareness

# Pros

- History as first-class citizen
- Simpler implementation of complicated business logic
- Temporal thinking
- Interactions awareness
- Domain understanding

# Thank you! :)

**kolodziejj.info**
**@unit03**
**tinyurl.com/es-gb-talk**

Growbots