Designing Event Based Microservices

About me

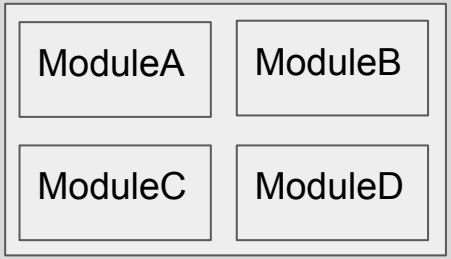
Grigoriadis Grigoris

Co-Founder and Software Architect @ itsaur

Microservices enthusiast!!!

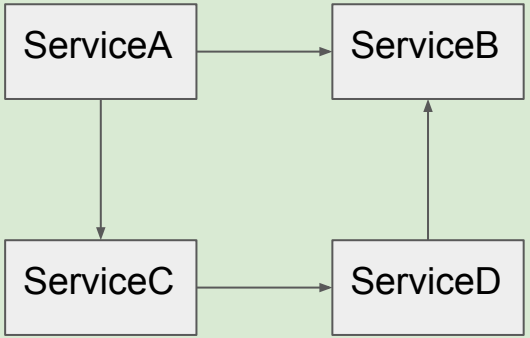
Microservices

Monolithic



- Application is separated into modules
- Modules communicate via method calls

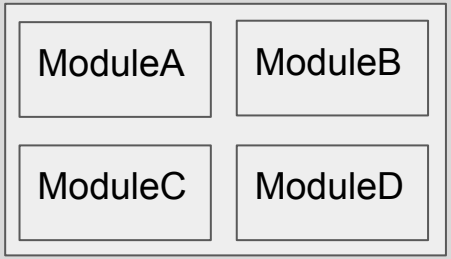
Microservices



- Application is separated into services
- Services are small & focused on doing one thing
- Communication between services is via Network calls

Microservices

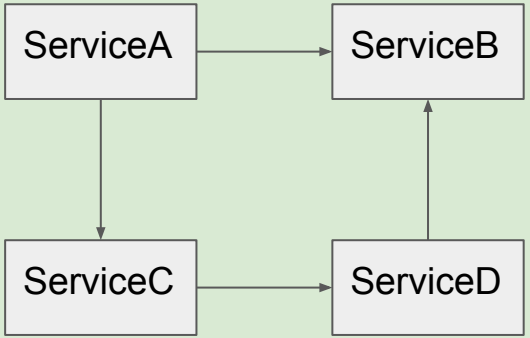
Monolithic



Pros

- Easier to start with
- Easier to deploy
- Easier to test

Microservices



Pros

- Scalable
- Resilient
- Maintainable

Microservices - Patterns & Practices

Evolution of an e-commerce Application

Authentication

- User registration
- Authentication
- Permissions

Product Catalog

- Create/Edit Product details (name, description etc)

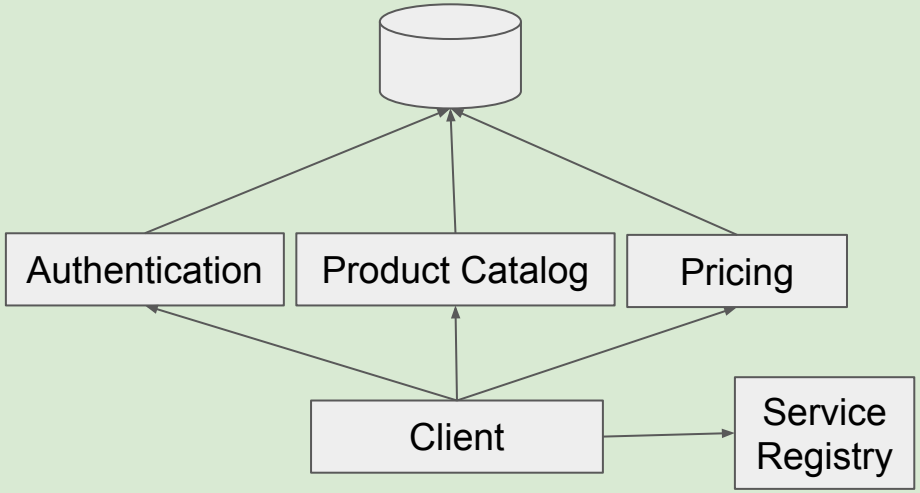
Pricing

- Manage pricing of products

Monolithic

- Client-Server Architecture
- Restful API
- Single Page Application

Shared Database



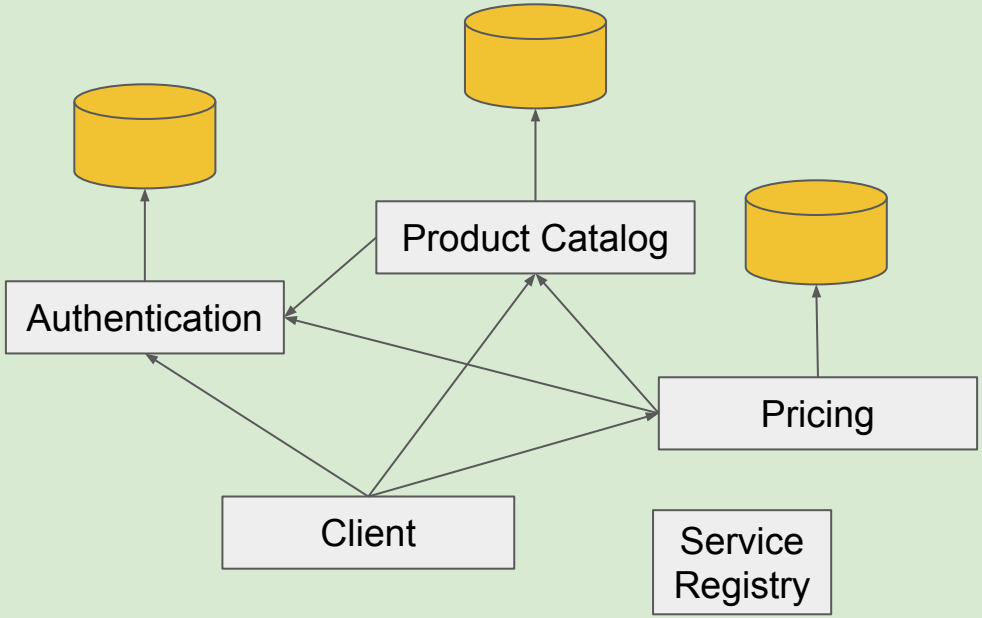
Pros

- Simple to start with

Cons

- Becomes more complex as system grows
- Difficult to define and enforce clear boundaries between systems
- Difficult to change/refactor as number of stakeholders increases
- Difficult to enforce caching
- Difficult to scale
- Need different indexes depending on the microservice

Database per Service



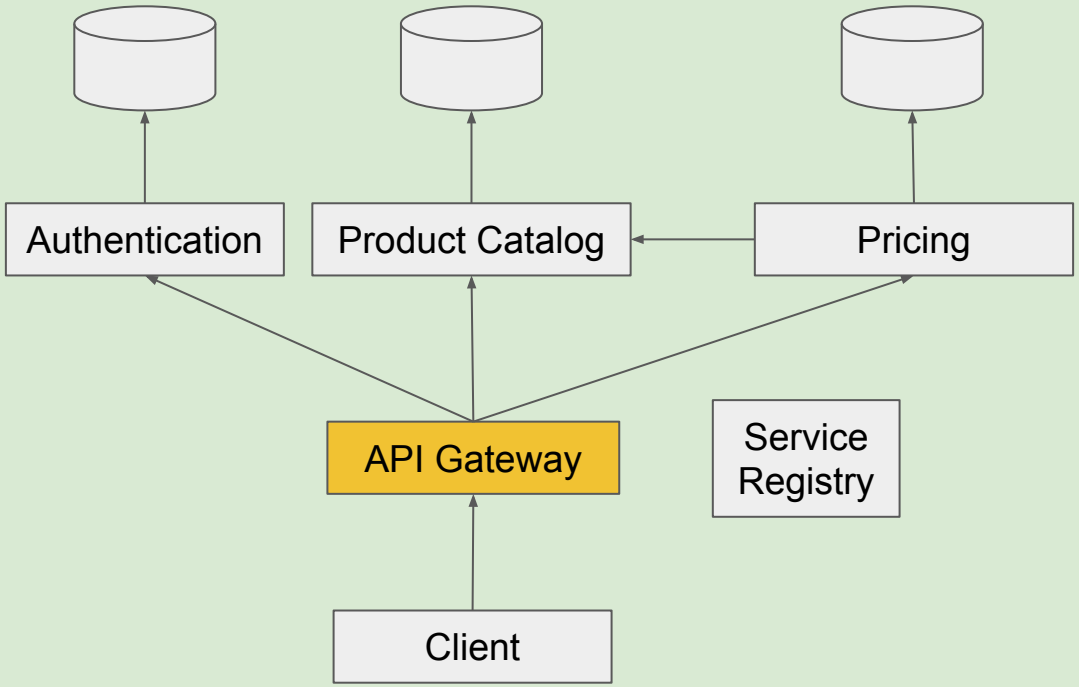
Pros

- Isolated database makes changes to schema easier without affecting consumers
- Easier to scale
- Easier to cache data since everyone communicates via api calls
- Polyglot persistence (one microservice might need kv store, another might user document store)

Cons

- Multiple network calls in order to perform an action
- If one service fails it might affect the whole system (e.g. authentication)

API Gateway



Pros

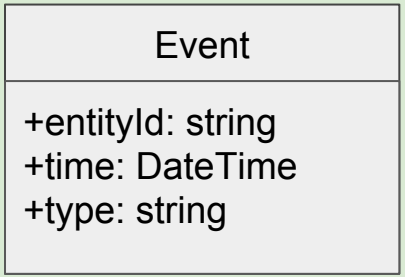
- Limited communication between services
- Client communicates only with API Gateway

Cons

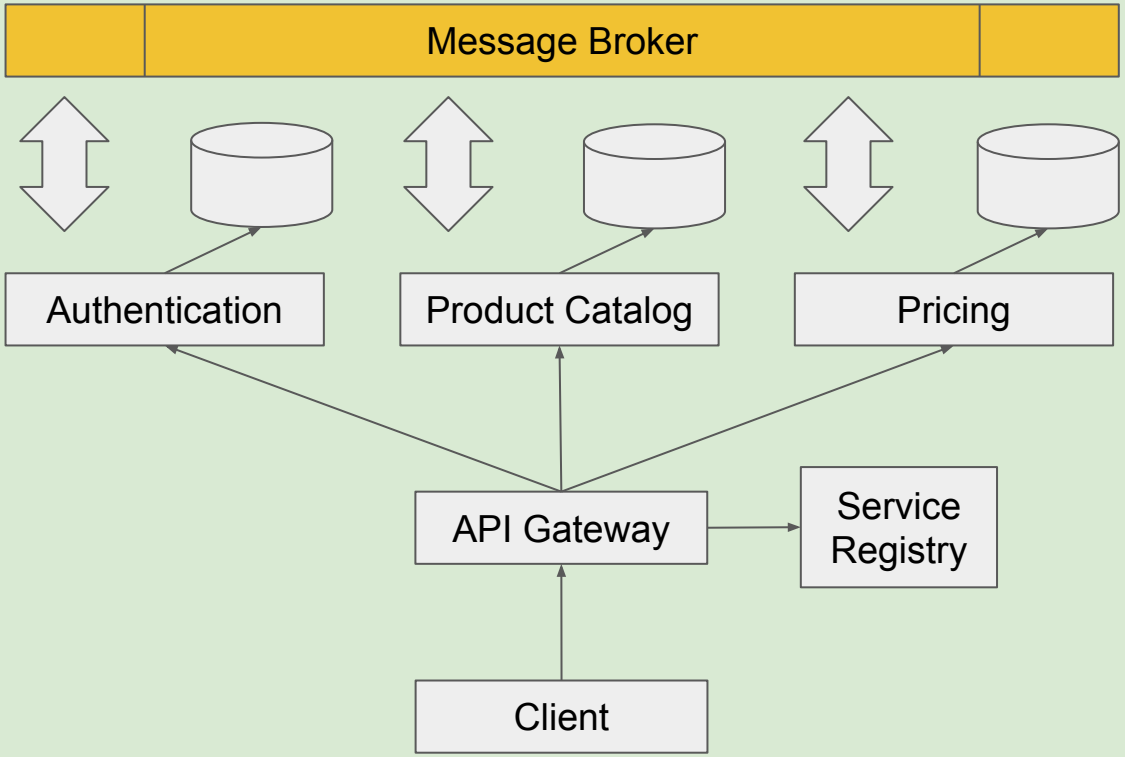
- API Gateway can become overly complex
- Business Logic might leak to gateway
- Multiple network calls by the gateway in order to retrieve data

Event Based

- An event is a fact, something that has already happened and cannot change
- Every state change must generate an event
- Events should NOT be overly generic (e.g. OrderUpdated), prefer more specific events like OrderStatusChanged or even better OrderDispatched etc.
- Events should contain only data relevant to the event.



Event Based



Pros

- No communication between services, each service duplicates the necessary data to its own DB

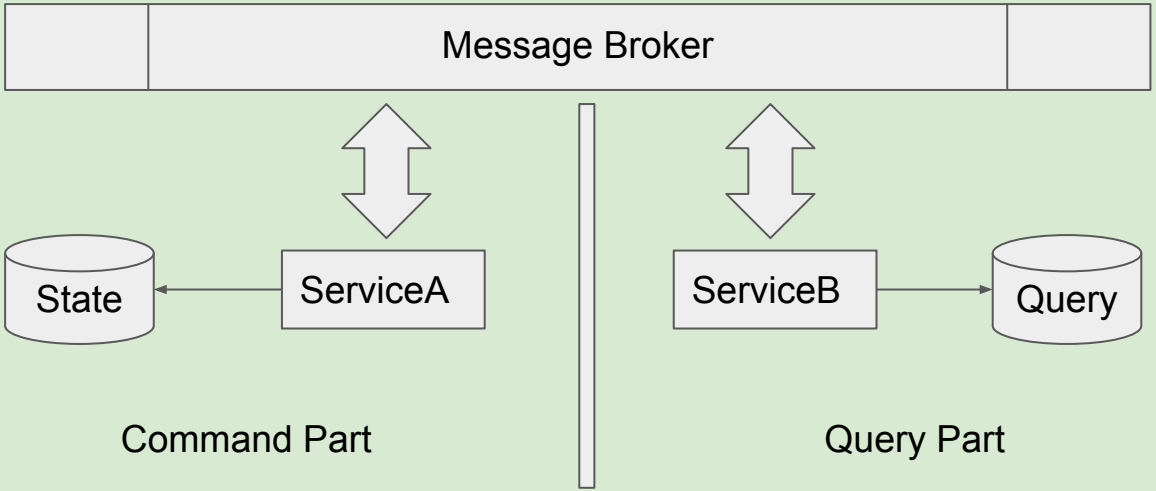
Cons

- Multiple network calls by the gateway in order to retrieve data

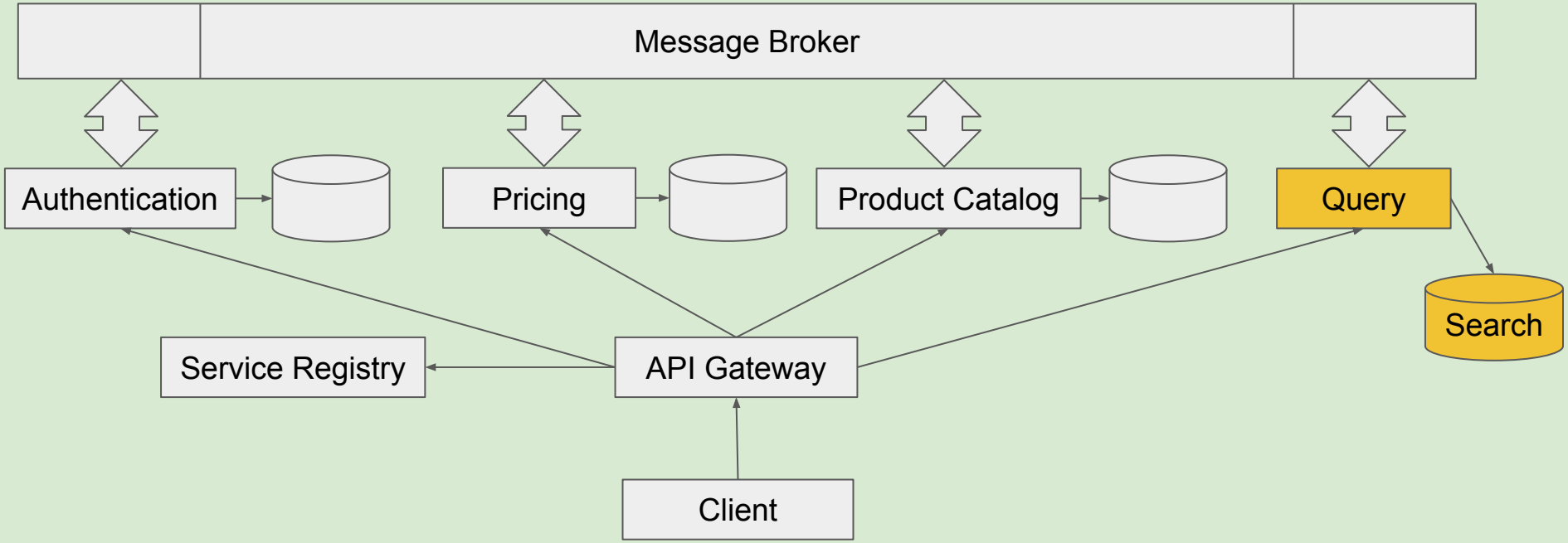
CQRS

Command Query Responsibility Segregation

- Command microservices deal with business logic and update the state of our application
- Query microservices deal with keeping the data in a schema appropriate for the client
- Command microservices publish an event for each state change
- Query microservices update their database by consuming events



CQRS


Message Broker
Authentication
Pricing
Product Catalog
Query
Search
Service Registry
API Gateway
Client

CQRS

Pros

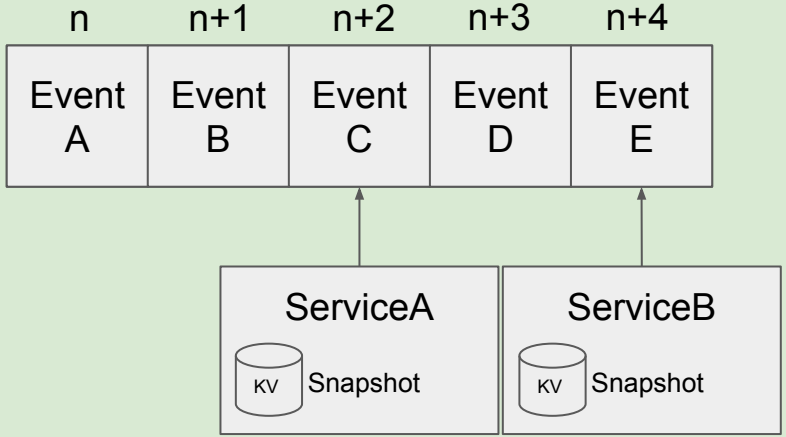
- Decoupled read from write schema can lead to huge performance improvements. (Different DB technologies to write and read).
- API Gateway can be very simple since query service has the data in a client-friendly format.

Cons

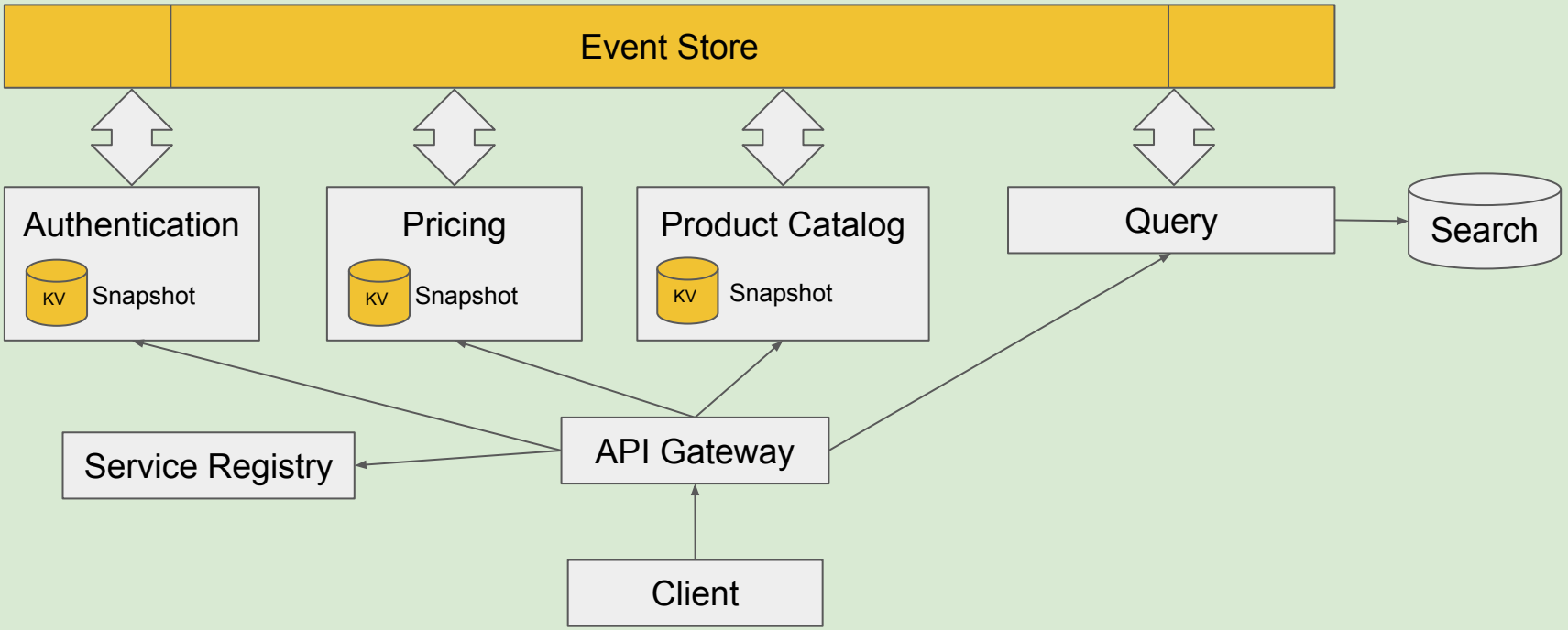
- Must handle cases where the state has been saved to DB but queue is down and cannot sent the message.

Event Sourcing

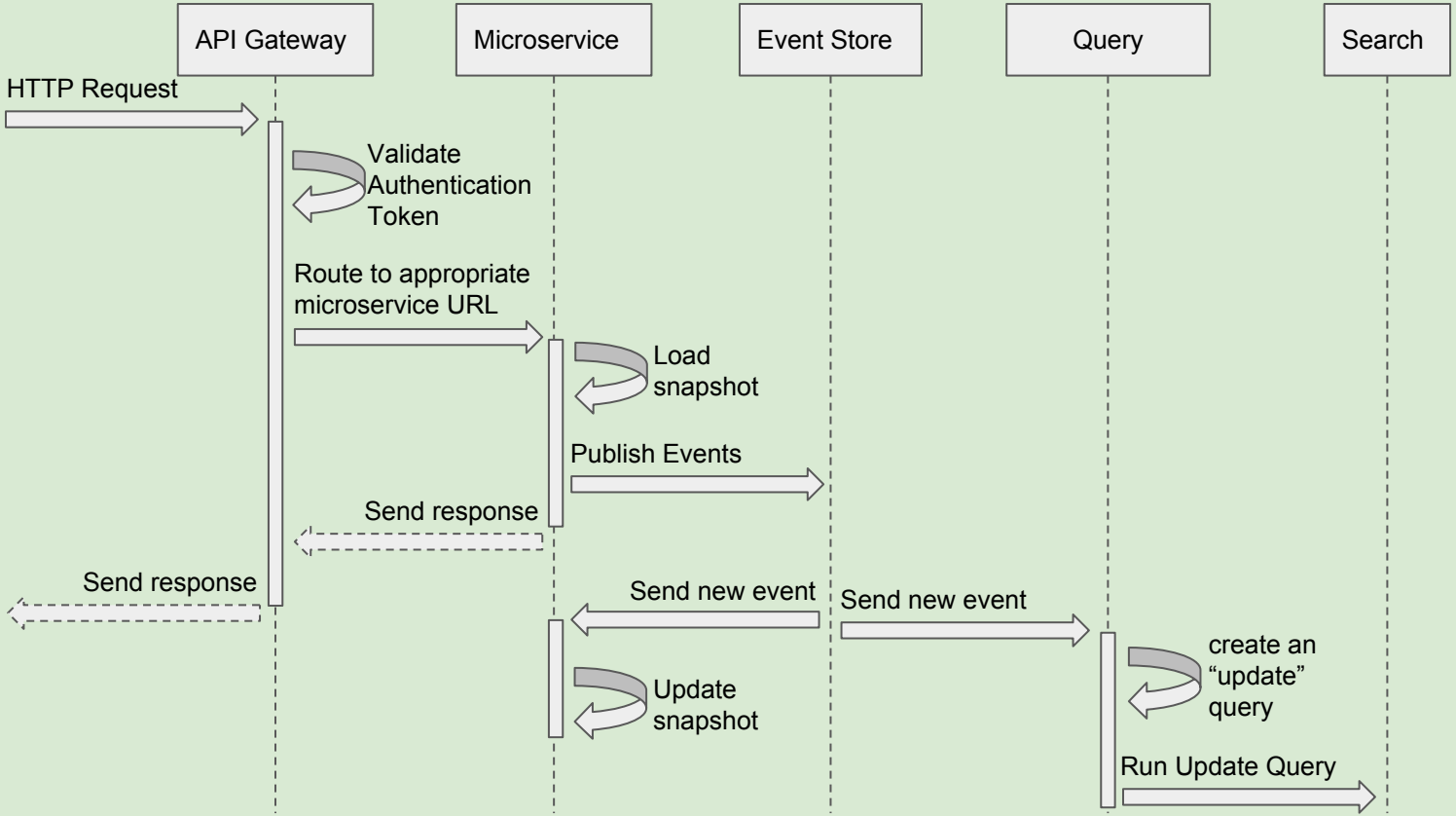
- All state changes in our system are represented as events.
- All events are persisted in an event store.
- Our application creates its state from the events in the event store and keeps a snapshot for faster access.
- Snapshots are best kept as embedded KV or document stores in order to avoid network calls and external system dependencies.
- We can recreate the state of our application at any given time by replaying the events up to that time.



Event Sourcing



Event Sourcing



Event Sourcing

Pros

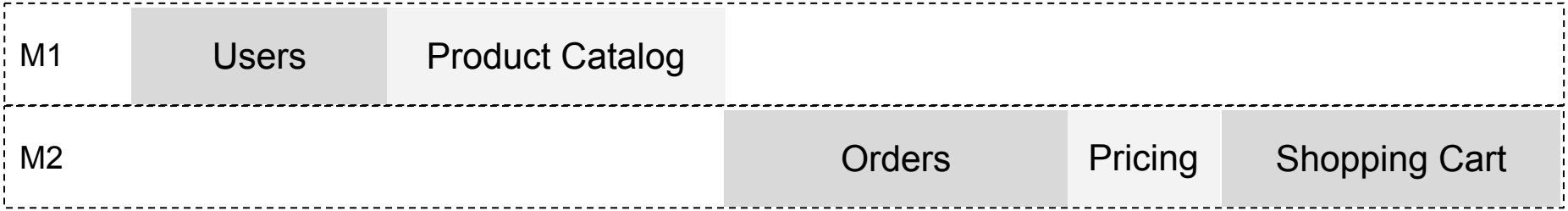
- History
- Improved performance, since we only make 1 network call per user request only to publish the events to the store.
- New services can read the events of our system to populate their state based on older events.

Cons

- In some systems with many events it might be expensive to keep the whole state*
- Can be difficult to implement

* Each service can emit a “snapshot” event in order to delete events up to a point where it doesn’t make sense to keep them anymore.

e-commerce solution



The Teams

Microteam

- Microservices
- Polyglot Persistence
- Eventual Consistency

Monoteam

- Monolithic
- Transactional
- Data-Driven Development

- Client-Server Architecture
- Restful API
- Single Page Application

M1

Features

Users

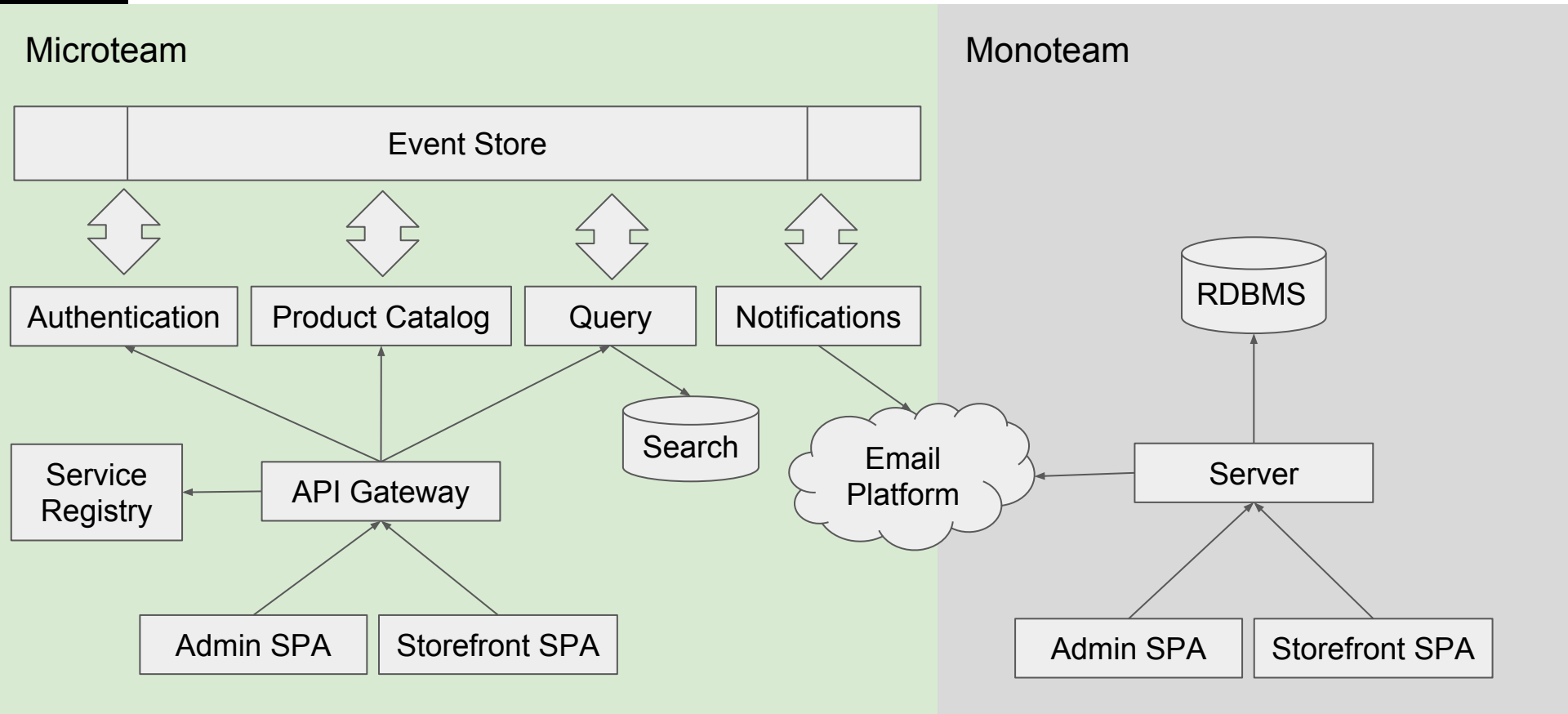
- Registration (Customer)
- Create (Admin)
- Forgot/Change Password
- Email verification
- Authentication
- Roles (Admin/ Customer)

Product Catalog

- Create (Title/ Description / Attributes)
- Media Links (images/videos)
- Search
- Filters using attributes (CPU Intel i7)
- Product Categories/Subcategories

M1

Design Overview



M1

Monoteam Design

DB Schema

USER
ID
EMAIL
PASSWORD
VERIFICATION_TOKEN
FORGOT_TOKEN
VERIFIED_AT
CREATED_AT
ROLE_ID

ROLE
ID

PRODUCT
ID
NAME
DESCRIPTION_HTML
CATEGORY_ID
STATUS
USER_ID

PRODUCT_CATEGORIES
ID
PRODUCT_ID
CATEGORY_ID

PRODUCT_MEDIA
ID
PRODUCT_ID
TITLE
LINK
TYPE
THUMBNAIL_URL

PRODUCT_ATTRIBUTE
ID
PRODUCT_ID
KEY
TYPE
VALUE_INT
VALUE_BOOL
VALUE_STRING
VALUE_DATE

PRODUCT_CATEGORY
ID
NAME
PARENT_CATEGORY

M1

Microteam Design

Authentication

UserRegisteredEvent
email
password
role
verificationToken

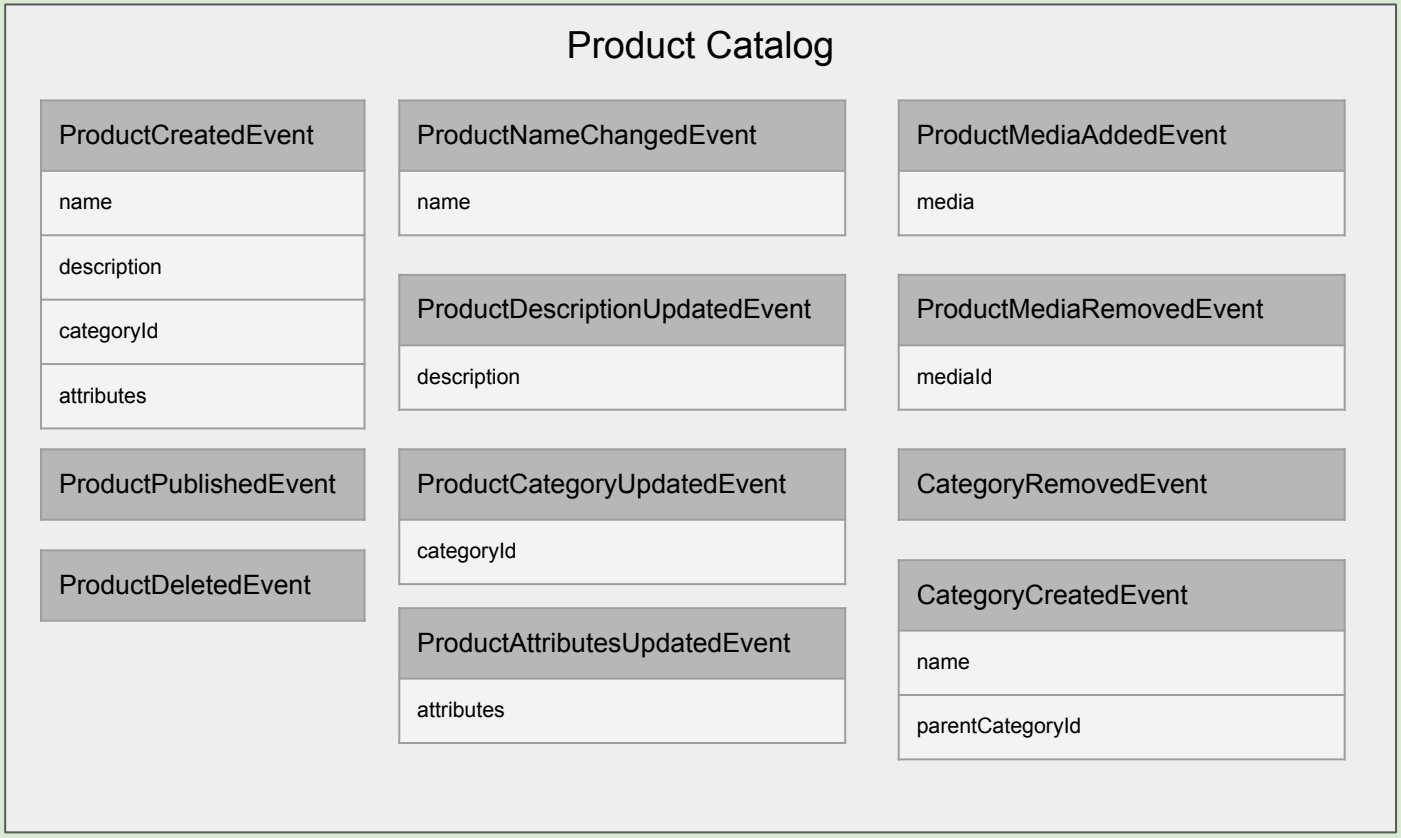
UserPasswordChangedEvent
newPassword
verificationToken

UserVerifiedEvent
verificationToken

UserPasswordForgottenEvent
email
verificationToken

M1

Microteam Design



M1

Microteam Design

Query Microservice - Search Indexes

SearchDB indexes

- Products
- Categories
- Users

Query microservice keeps state in order to make at most 1 call per event/index.

Products

```
{
  "id": "uuid",
  "name": "product 1",
  "description": "html",
  "links": [
    {
      "type": "video | image",
      "url": "url",
      "thumbnailUrl": "url"
    }
  ],
  “categories”: [
    {
      “id”: “uuid”,
      “name”: “string”
    }
  ],
  “status”: “draft | published | deleted”,
  "attributes": {
    “key”: “object”
  },
  “attributeMetadata”: [
    {
      “key”: “string”,
      “Type”: “Number | Text”
    }
  ]
}
```

Categories

```
{
  "id": "uuid",
  "name": "string",
  "subcategories": [
    {
      "id": "uuid",
      “name”: “string”,
      “subcategories”: [
      ]
    }
  ]
}
```

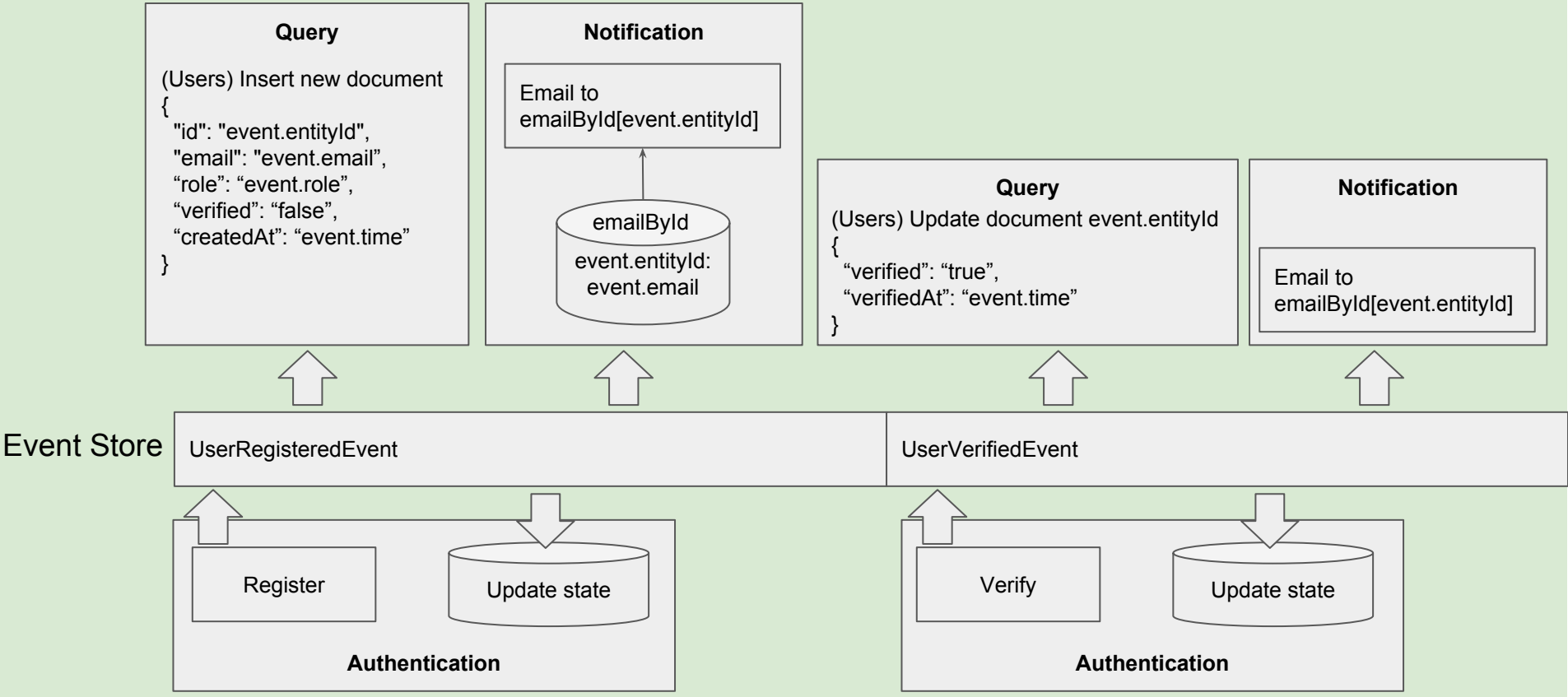
Users

```
{
  "id": "uuid",
  "email": "email”,
  “role”: “string”,
  “verified”: “boolean”,
  “createdAt”: “date”
  “verifiedAt”: “date”
}
```

M1

Microteam Design

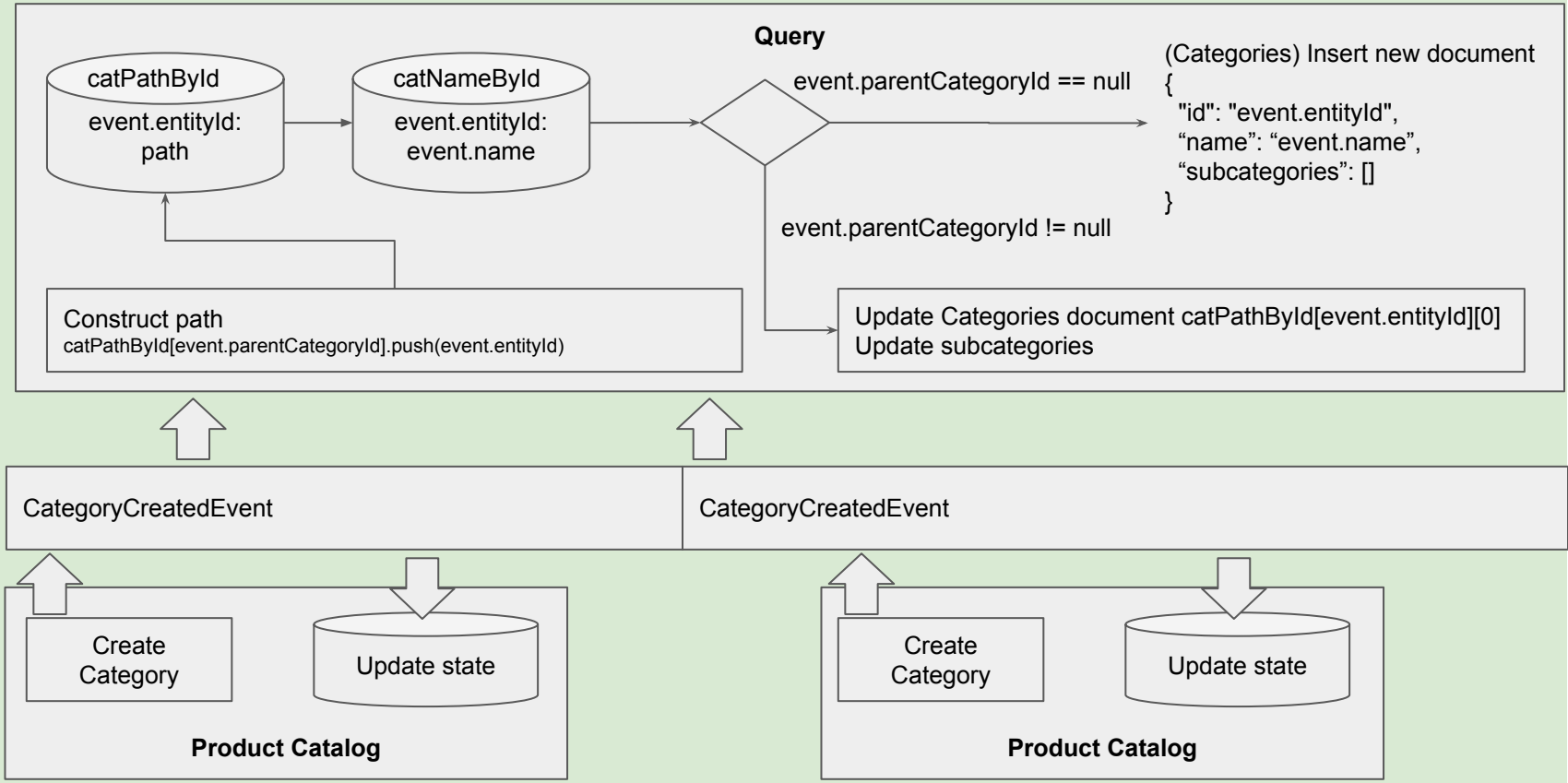
User Choreography



M1

Microteam Design

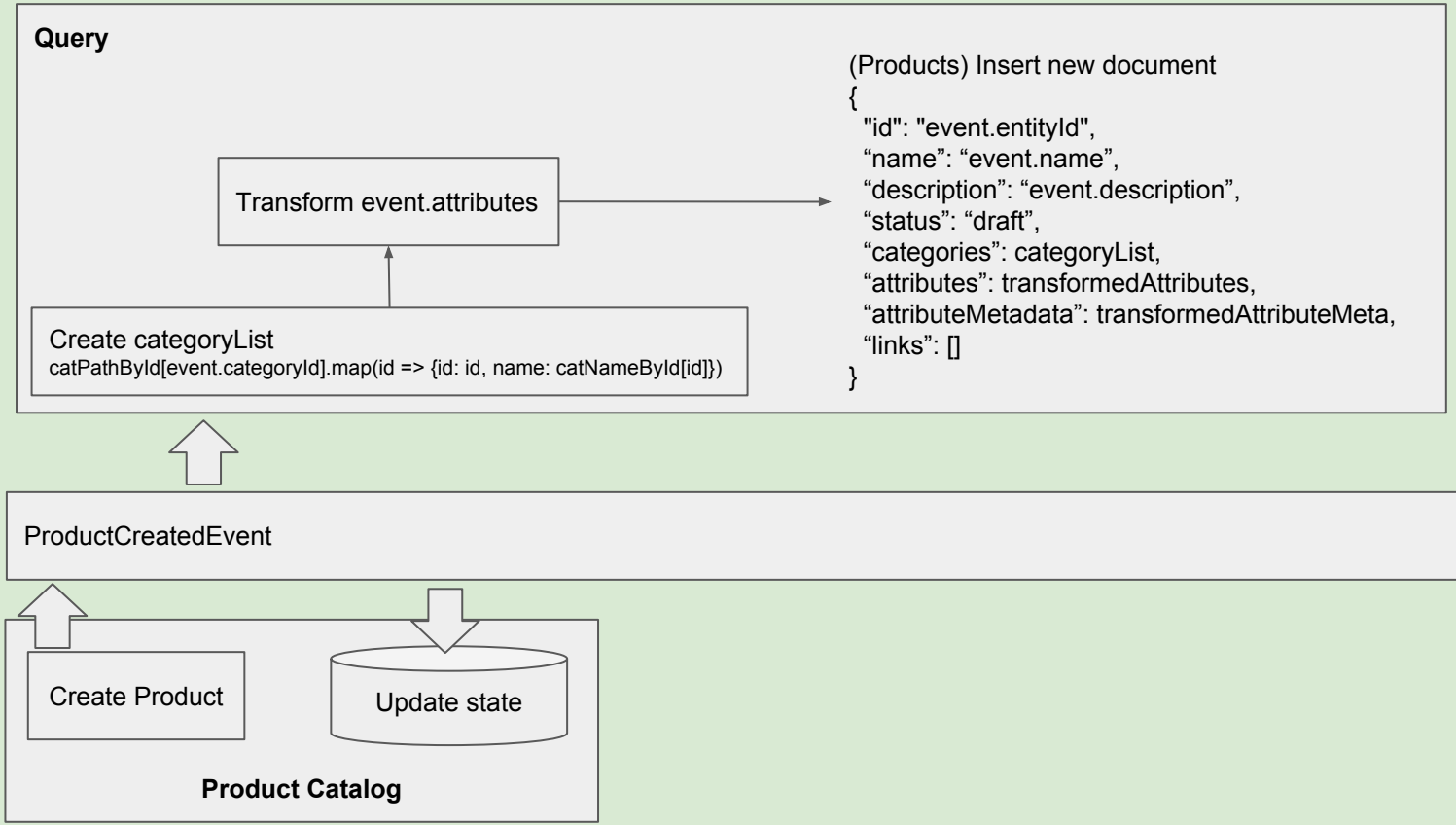
Category Choreography



M1

Microteam Design

Product Choreography



Conclusion

Pros
- Better performance
- Events describe better what the system does.

Cons
- Difficult to implement
- Duplicated Data
- Deployment Costs

Pros
- Faster to implement

Cons
- Slow performance
- Failure Handling
- Scaling

M2

Features

Pricing
- Set/view product price

Orders
- Place order from cart
- View Order status
- View pending orders
- Update order status

Shopping Cart
- Add Product to cart
- Remove Product from cart

Users
- Contact Info (Phone number, addresses)
- New sales role

M2

Monoteam Design

DB Schema

PRODUCT
PRICE

USER_ADDRESS
ID
USER_ID
ADDRESS_LINE
CITY
COUNTRY
STATE
ZIP_CODE
PHONE
ACTIVE

CART
ID
USER_ID

CART_ITEM
ID
CART_ID
PRODUCT_ID
QUANTITY

ORDER
ID
USER_ID
ADDRESS_ID
CREATED_AT
LAST_STATUS_ID

ORDER_STATUS
ID
ORDER_ID
STATUS
CHANGED_AT
COMMENT

ORDER_ITEM
ID
ORDER_ID
PRODUCT_ID
QUANTITY
PRICE

M2

Microteam Design

Pricing

PriceChangedEvent

- price

Cart

CartCreatedEvent

- userId

CartItemAddedEvent

- productId
- quantity

CartCheckoutEvent

- items

CartItemQuantityUpdatedEvent

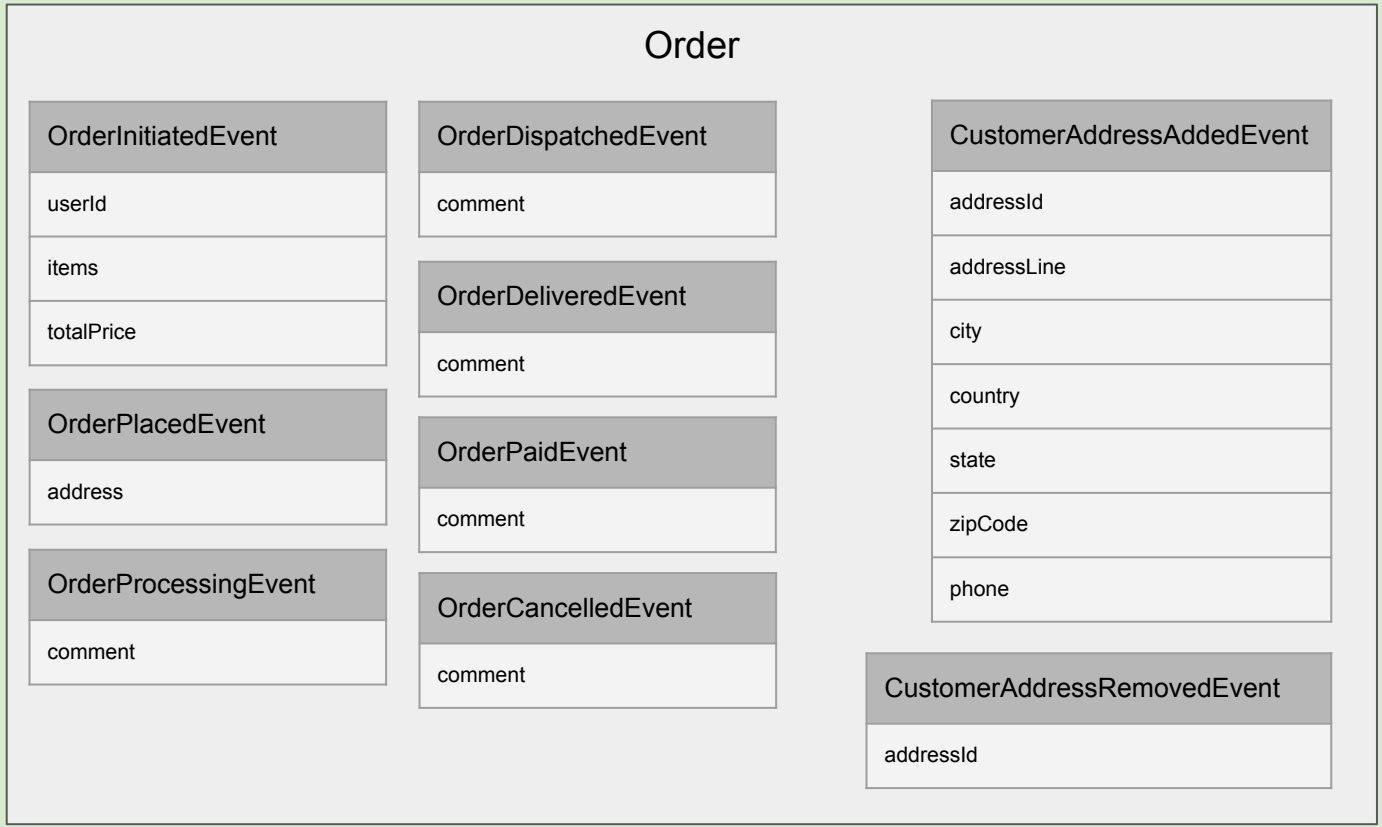
- productId
- quantity

CartItemRemovedEvent

- productId

M2

Microteam Design



M2

Microteam Design

Query Microservice - Search Indexes

Orders

```
{
  "id": "uuid",
  “user”: “uuid”,
  “status”: {
    “status”: “string”,
    “updateAt”: “date”,
    “comment”: “string”
  },
  "statusHistory": [
    {
      “status”: “string”,
      “updateAt”: “date”,
      “comment”: “string”
    }
  ],
  “Items”: [
    {
      “productId”: “uuid”,
      “productName”: “string”,
      “productImageUrl”: “string”,
      “quantity”: “number”,
      “productPrice”: “number”,
      “itemPrice”: “number”
    }
  ],
  “address”: {
      “addressLine”: “string”,
      “city”: “string”,
      “country”: “string”
      “state”: “string”,
      “zipCode”: “string”,
      “phone”: “string”
  },
  “totalPrice”: “number”
}
```

Carts

```
{
  "id": "uuid",
  “user”: “uuid”,
  “Items”: [
    {
      “productId”: “uuid”,
      “productName”: “string”,
      “productImageUrl”: “string”,
      “quantity”: “number”,
      “productPrice”: “number”,
      “itemPrice”: “number”
    }
  ],
  “totalPrice”: “number”
}
```

Users

```
{
  “addresses”: [
    {
      “addressLine”: “string”,
      “city”: “string”,
      “country”: “string”
      “state”: “string”,
      “zipCode”: “string”,
      “phone”: “string”
    }
  ]
}
```

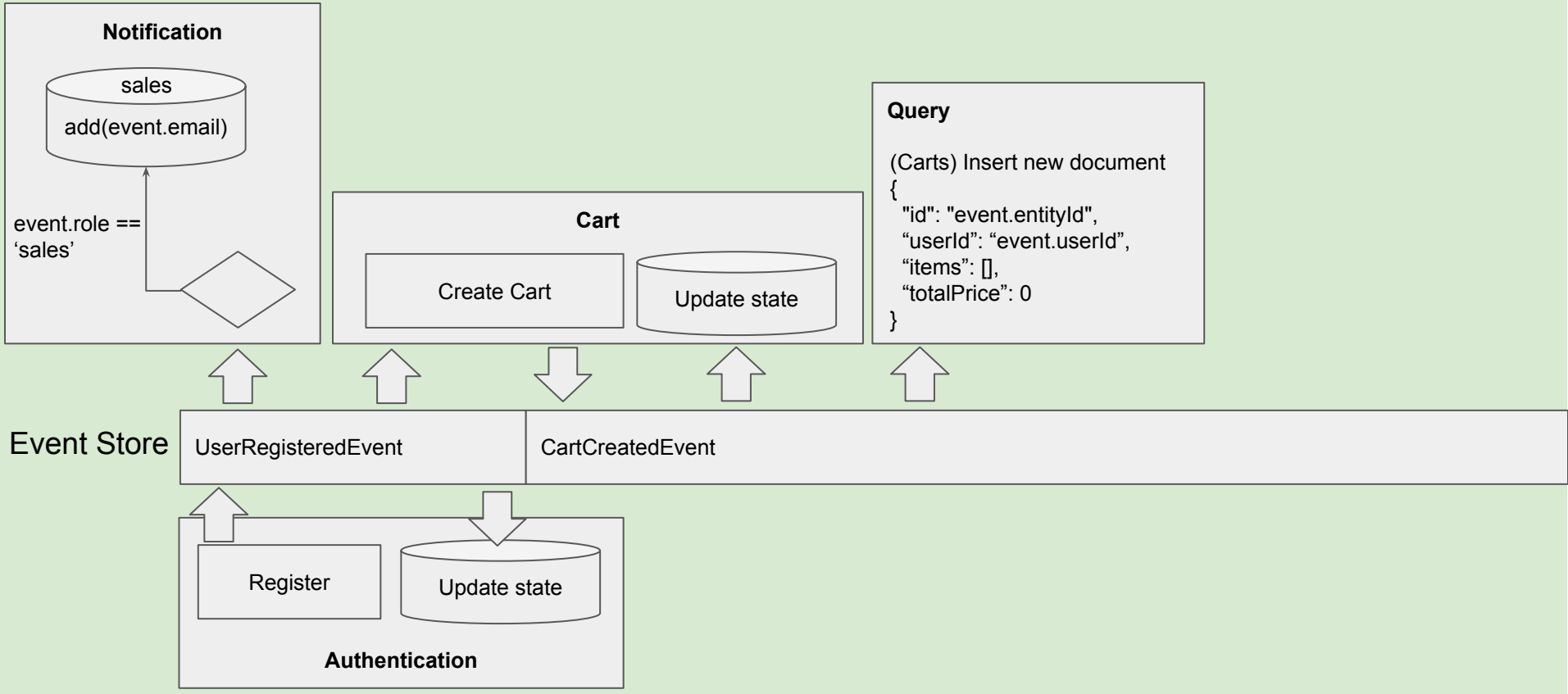
Products

```
{
  “price”: “number”
}
```

M2

Microteam Design

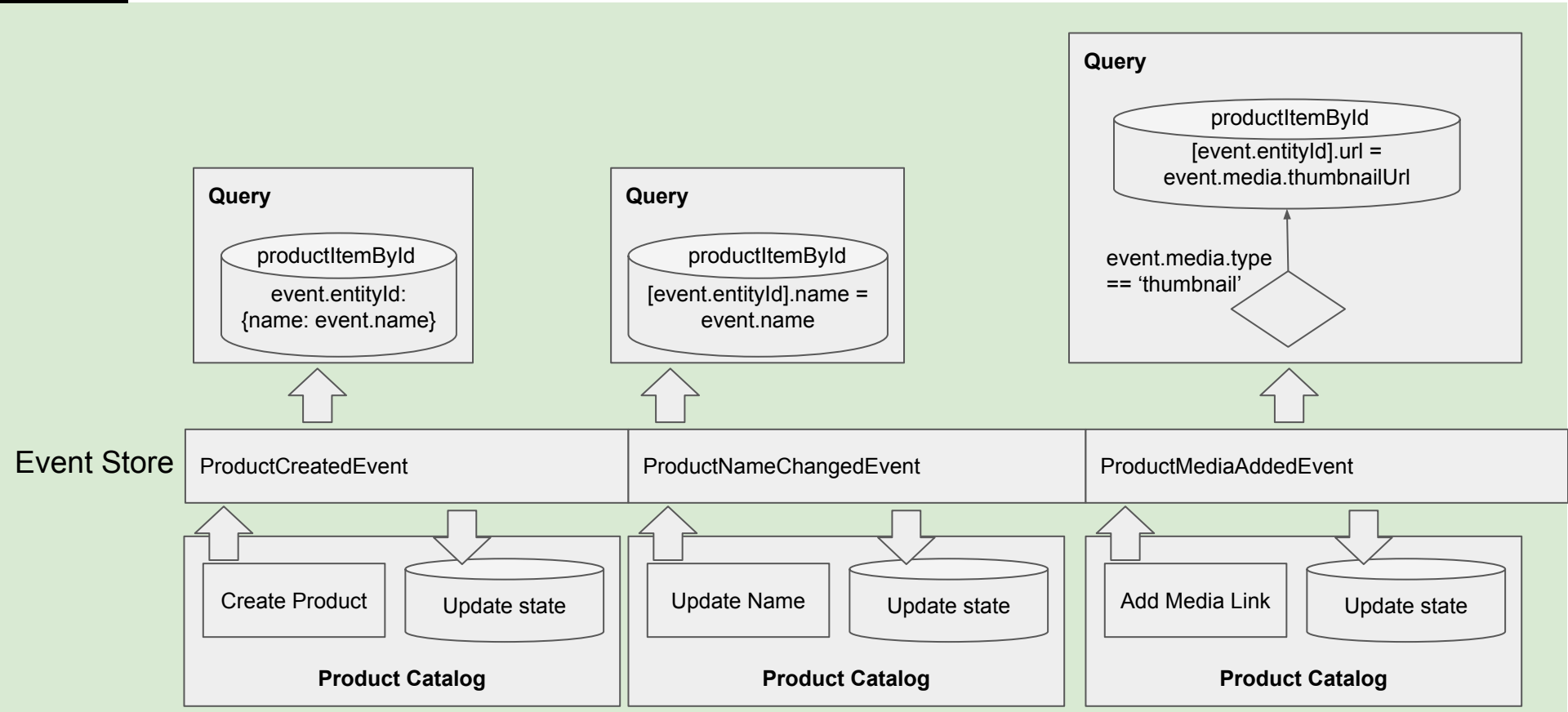
Cart Choreography



M2

Microteam Design

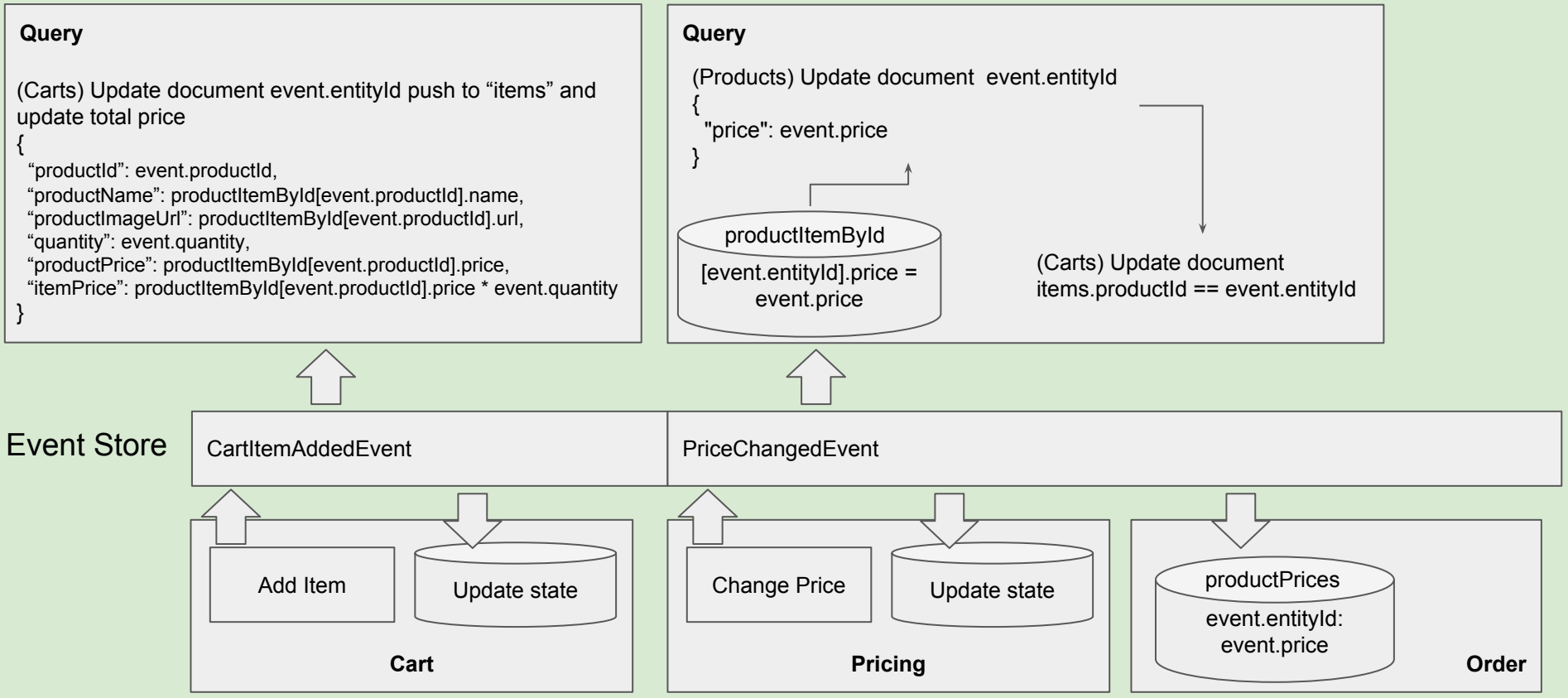
Product Choreography



M2

Microteam Design

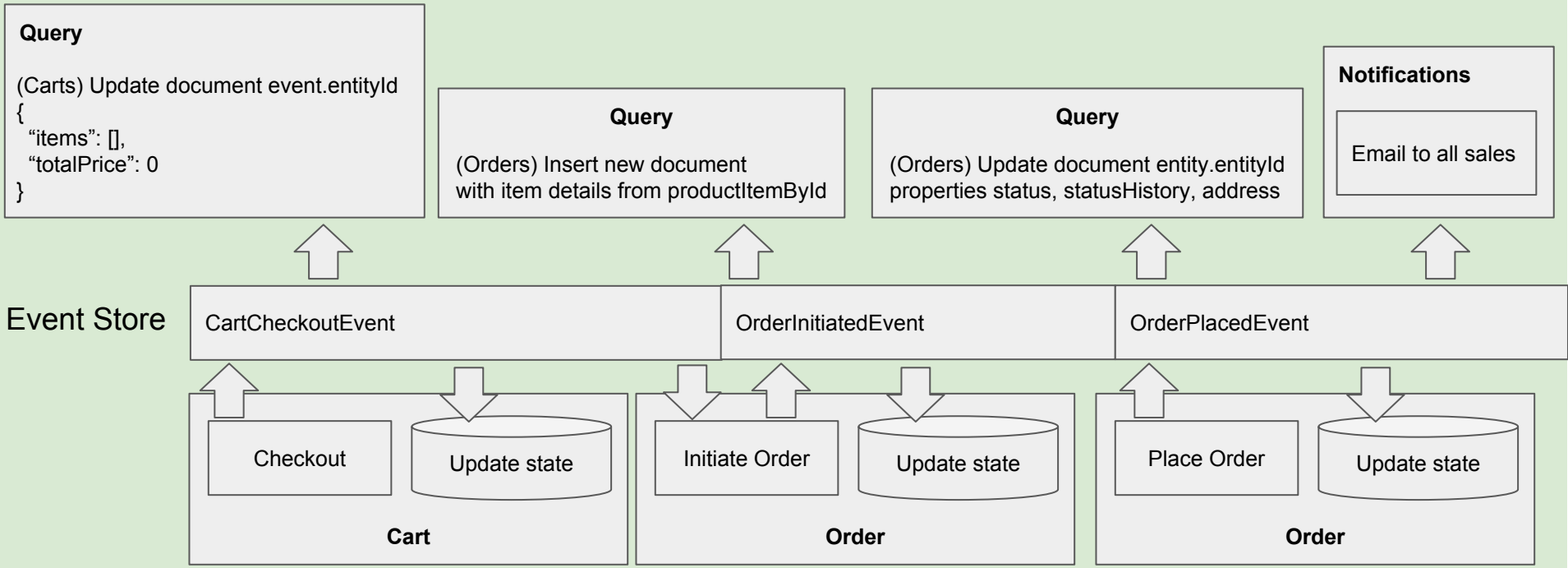
Pricing Choreography



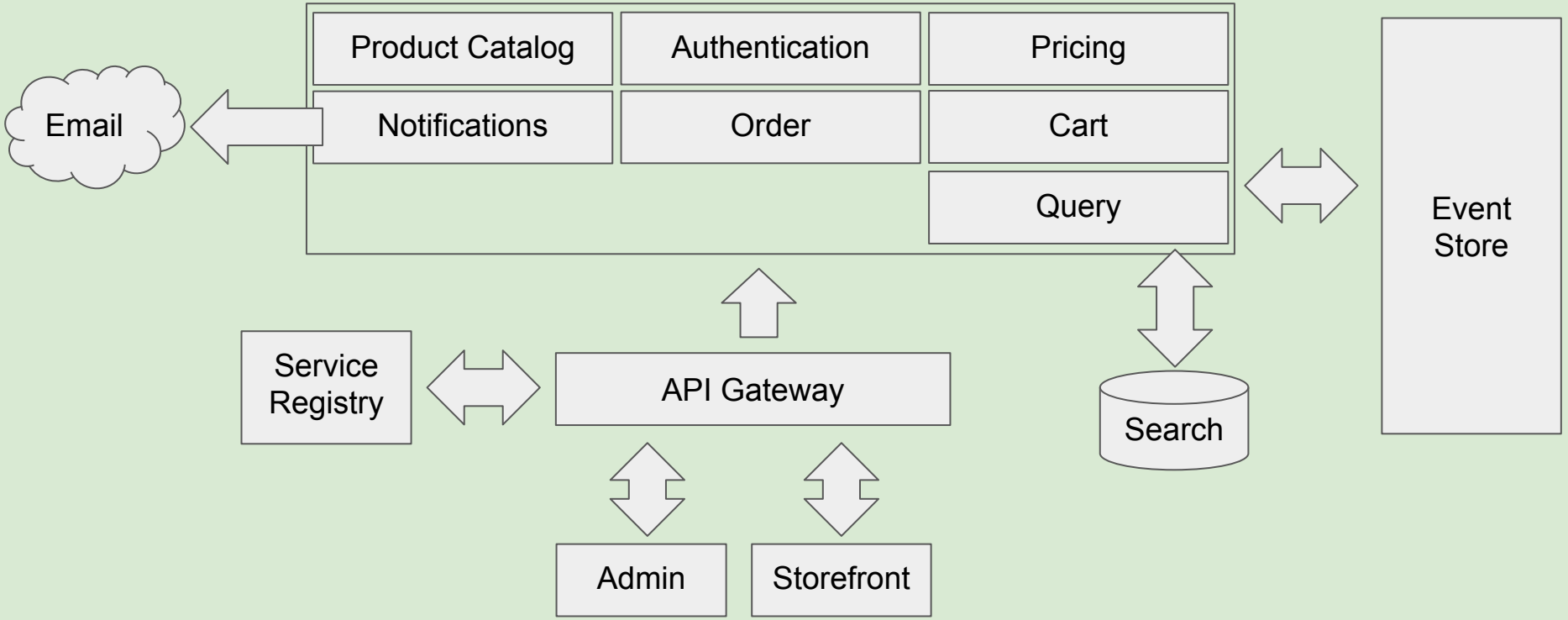
M2

Microteam Design

Order Choreography



Final Design



To microservice or not

- Domain Knowledge
- Deployment Automations
- Monitoring / Centralized Logging
- Development skill set

Q&A