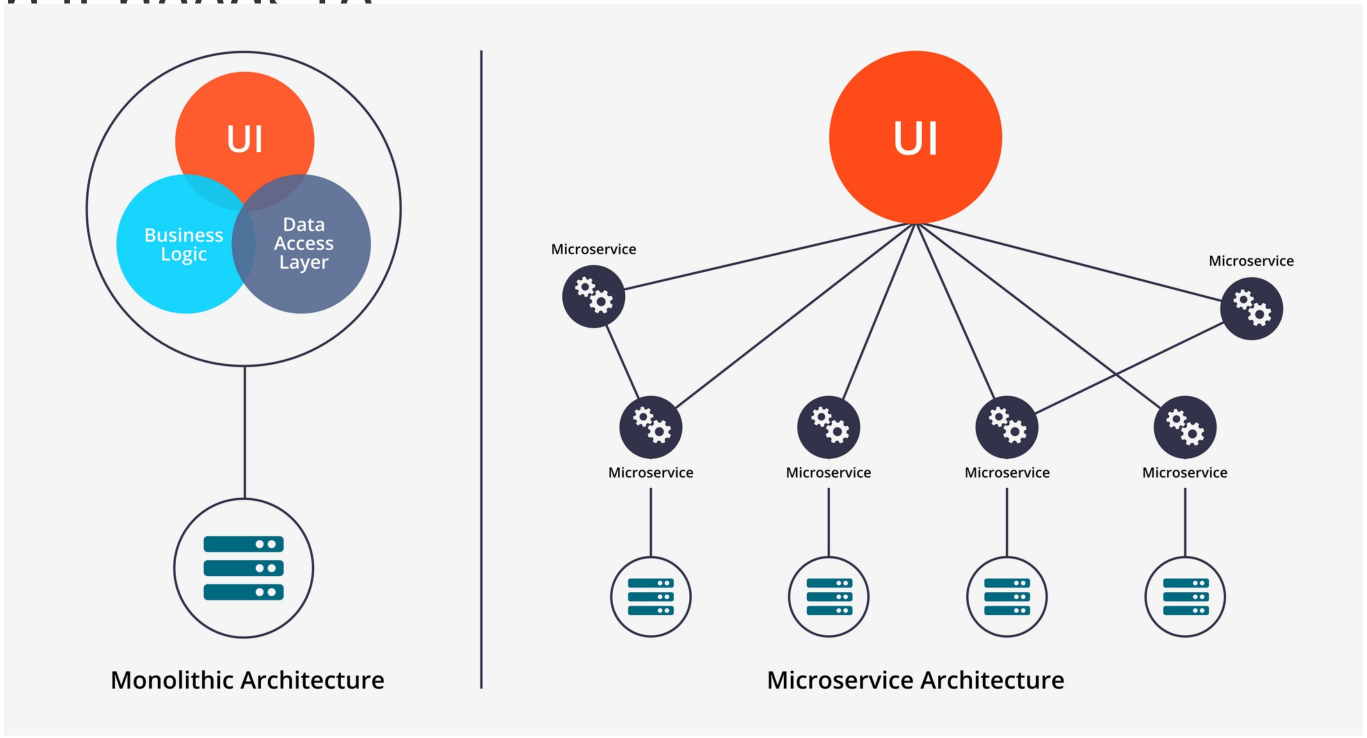
Software Architecture

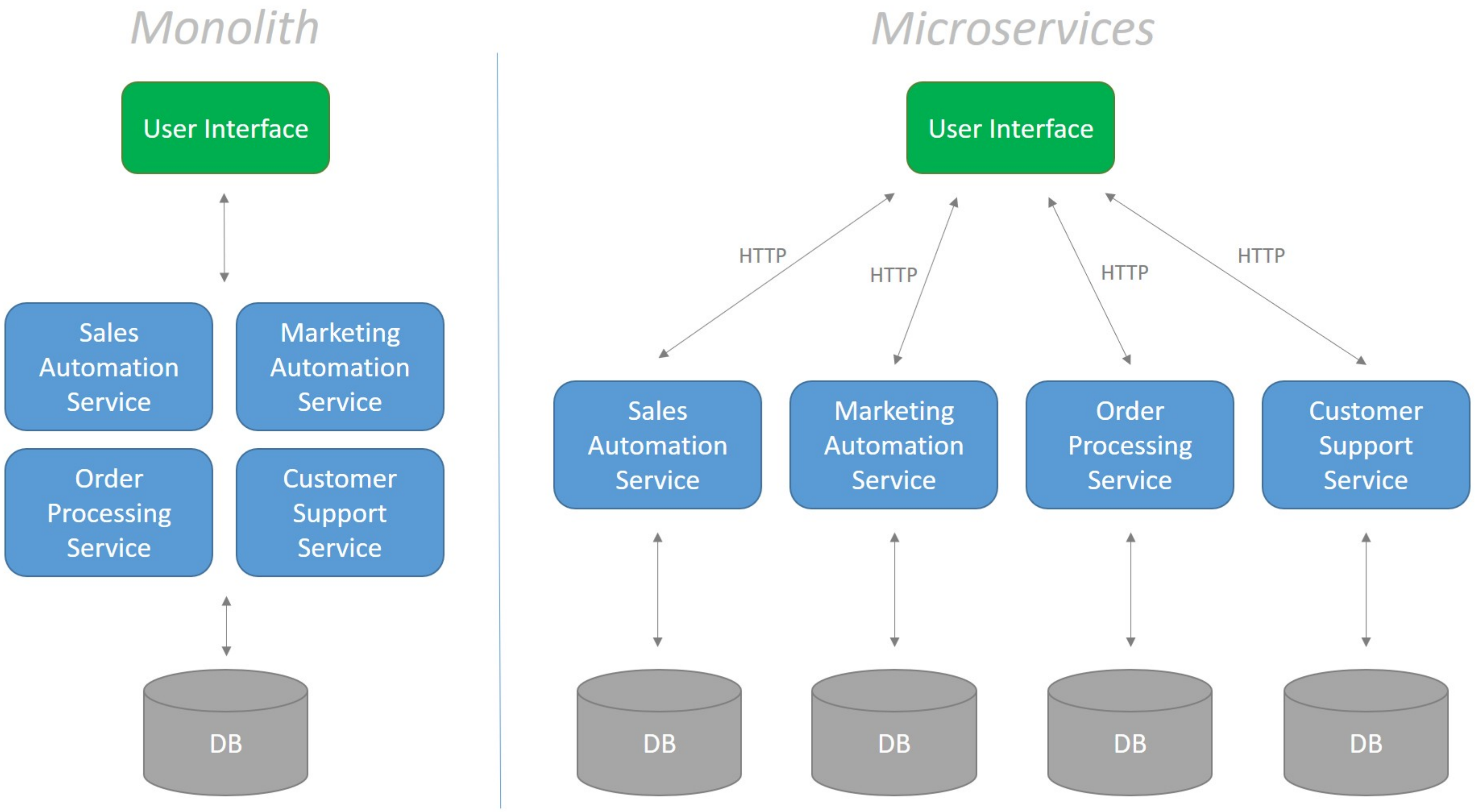
Design

Remember Microservices ?

- Deployed Independently, has its own process(es)
- Small team to create and manage
- Does one thing really well, communicates with others when it needs to

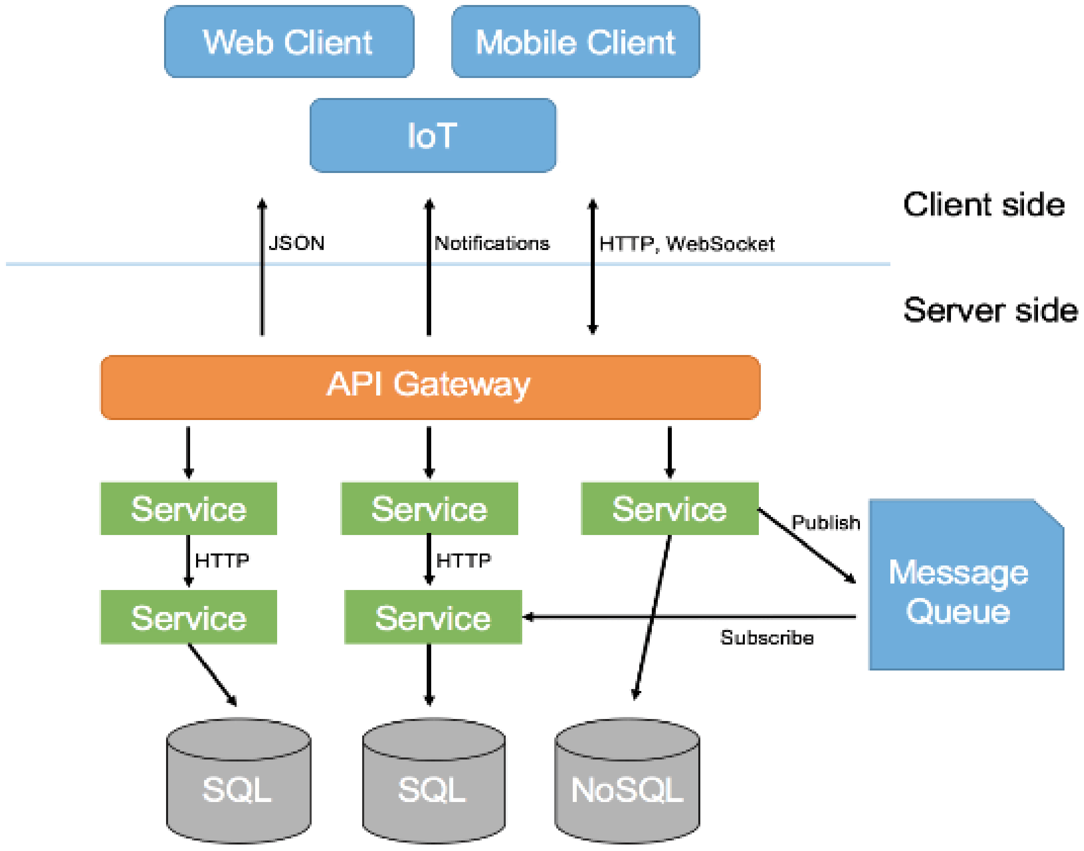


https://www.weave.works/blog/what-are-microservices/



https://www.carlrippon.com/scalable-and-performant-asp-net-core-web-apis-microservices/

API Gateway



https://www.techjini.com/blog/microservices/

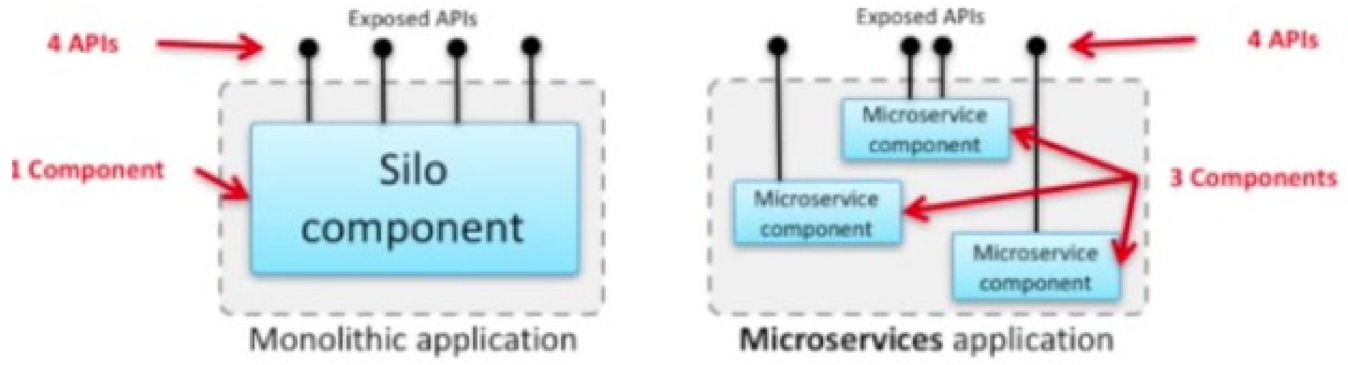
Do you notice any communication patterns ?

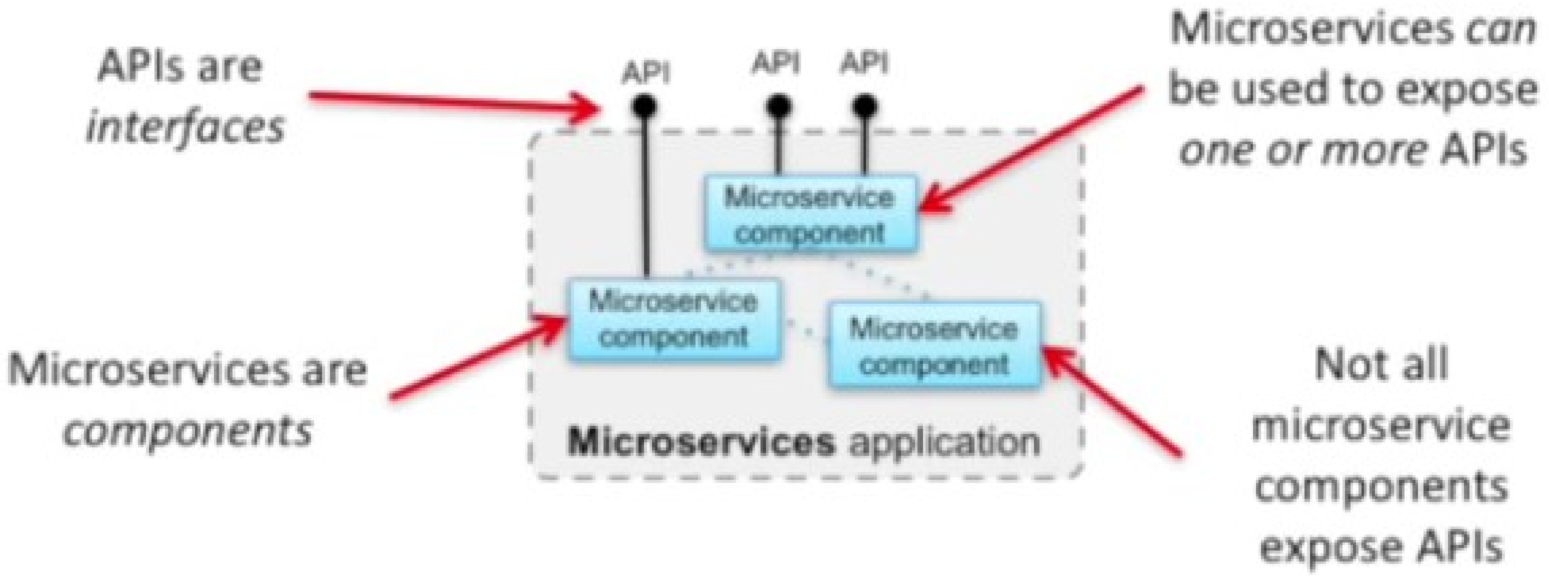
Two Architectural decisions → Design

- How do we determine the scope of each microservice ?!
- How should they communicate ?!
  - Direct API calls ?
  - Events on a shared queue ?
  - Something else …?

Q1) How do we design APIs ?

API vs Microservice




APIs are *interfaces*
API
API
API
Microservices *can* be used to expose *one or more* APIs
Microservice component
Microservice component
Microservice component
Microservices are *components*
Microservices application
Not all microservice components expose APIs

https://www.youtube.com/watch?v=WFtmCaqIjkA

Designing API

What are the main architectural styles for APIs?

API Styles

- Rest
  - HATEOAS
- RPC
  - gRPC
  - SAOP
- GraphQL
  - Falcor

REST

- Representational state transfer
- Very common on web over HTTP methods
  - GET,POST,PUT,..
- Generally stateless calls

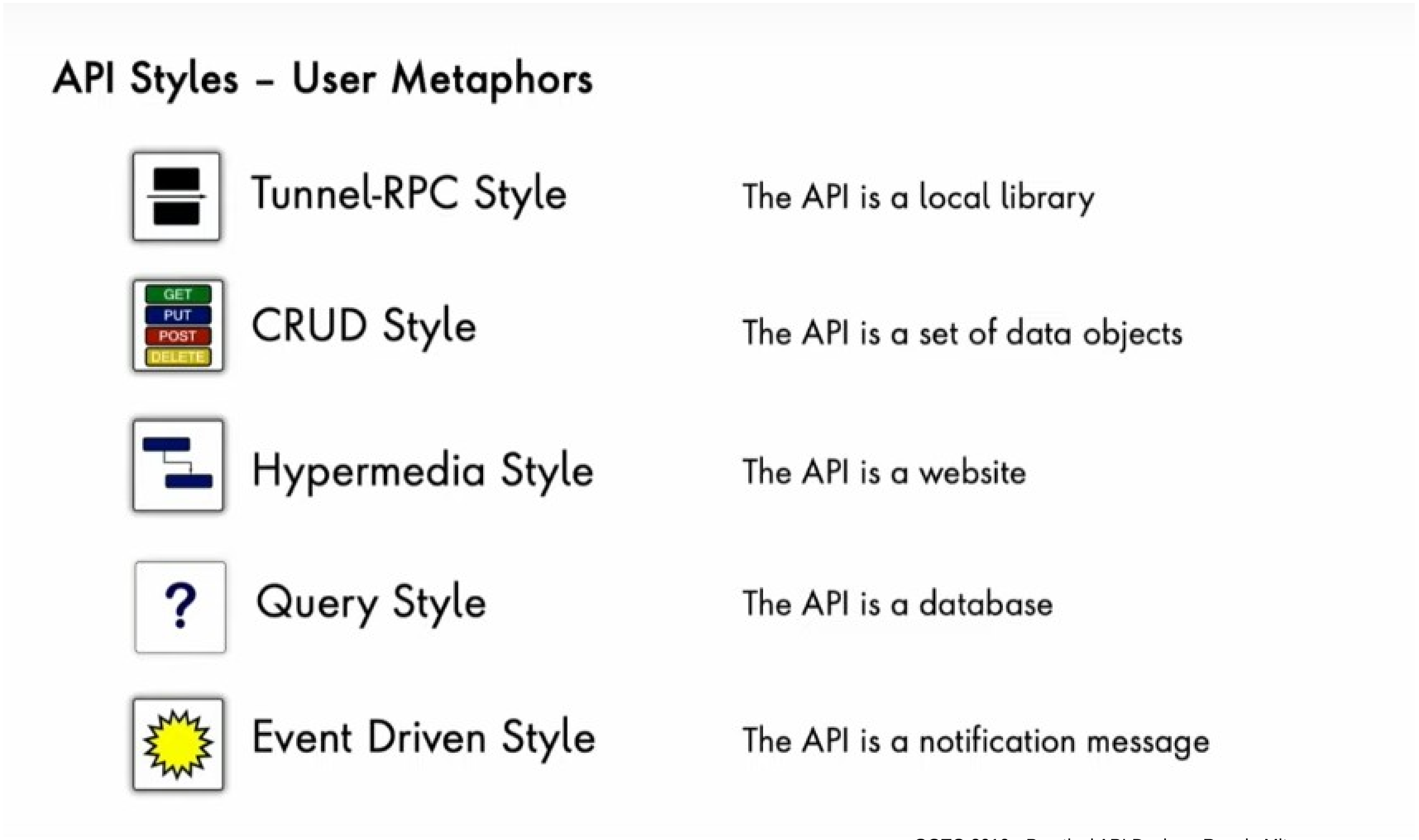
gRPC

- Remote Procedure calls, are efficient and strongly typed
- gRPC is optimized:
  - Transport over http/2
  - Binary protocol & serializing (e.g ProtoBuf)
- Messages support schema evolution
- Bi-directional communication/streaming
- Checkout:
  - https://www.grpc.io/

GraphQL

- Query parts of JSON like object
- Clients get exactly what they ask for and nothing more!
- Checkout:
  - https://graphql.org/

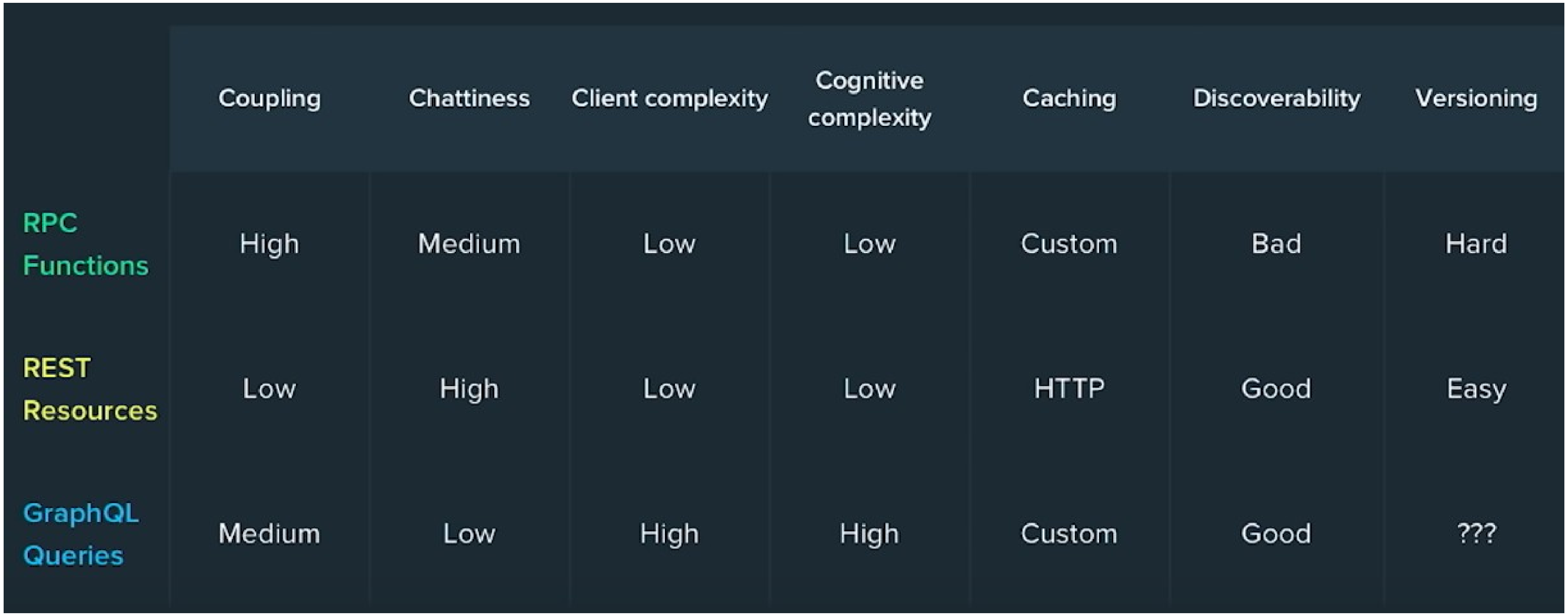
API Styles



API Styles – User Metaphors

GET PUT POST DELETE	Tunnel-RPC Style	The API is a local library
	CRUD Style	The API is a set of data objects
	Hypermedia Style	The API is a website
?	Query Style	The API is a database
	Event Driven Style	The API is a notification message

Trade offs



	Coupling	Chattiness	Client complexity	Cognitive complexity	Caching	Discoverability	Versioning
RPC Functions	High	Medium	Low	Low	Custom	Bad	Hard
REST Resources	Low	High	Low	Low	HTTP	Good	Easy
GraphQL Queries	Medium	Low	High	High	Custom	Good	???

Nate Barbettini – API Throwdown: RPC vs REST vs GraphQL, Iterate 2018

API styles

- Let’s watch :
  - Nate Barbettini – API Throwdown: RPC vs REST vs GraphQL, Iterate 2018
    - https://youtu.be/IvsANO0qZEg?t=75

Designing API

- Read:
  - https://api-university.com/blog/styles-for-apis-soap-rest-and-rpc/

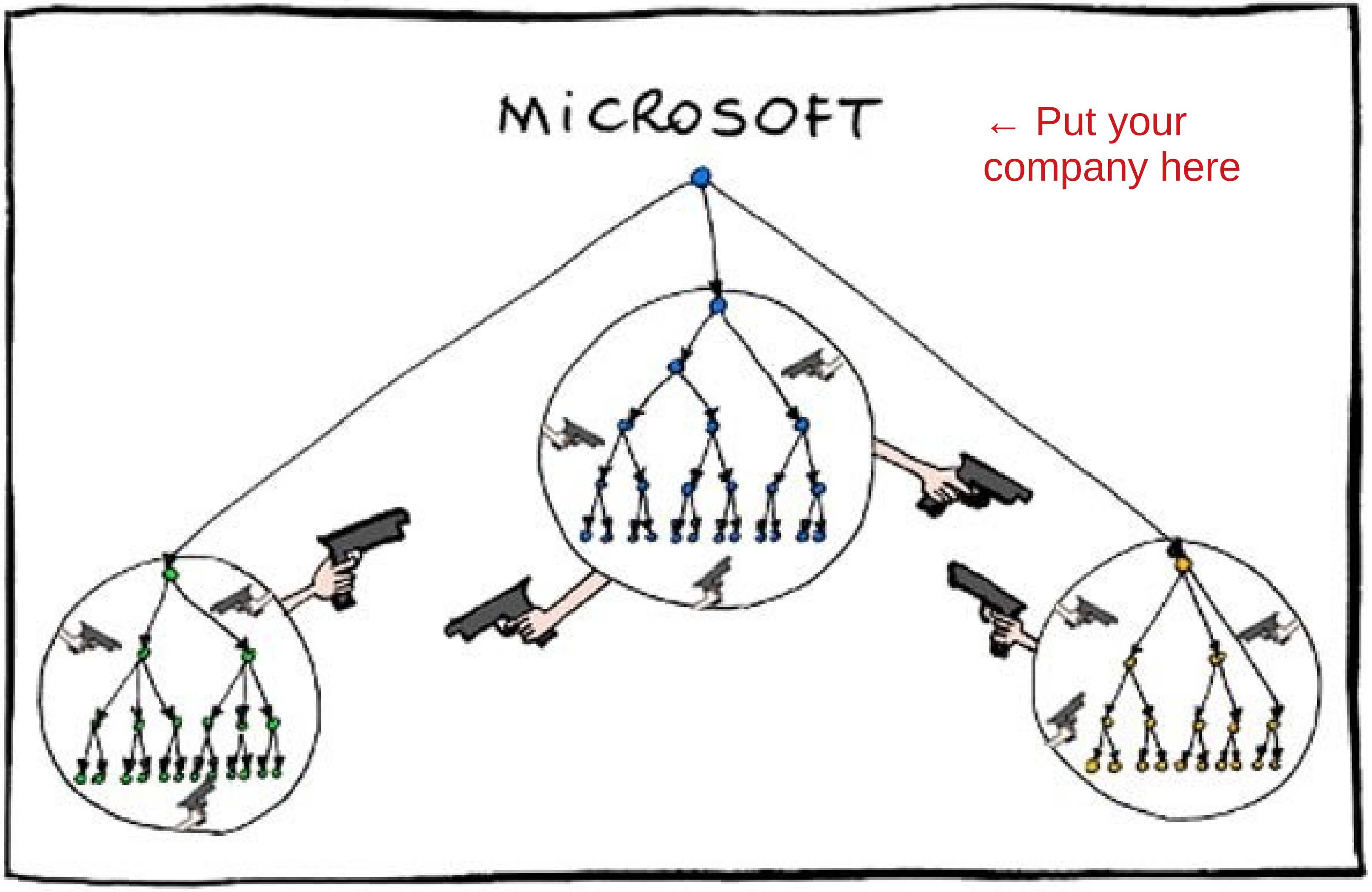
Designing API

- Got time?, Watch:
  - Designing Quality APIs (Cloud Next '18)
    - https://www.youtube.com/watch?v=P0a7PwRNLVU
  - GOTO 2019 • Practical API Design • Ronnie Mitra
    - https://www.youtube.com/watch?v=272ZZ53HS_4

Q2) How do we divide our monolith into microservices ?

Conway’s Law

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



← Put your company here

http://www.businessinsider.com/the-ballmer-re-org-chart-2013-7

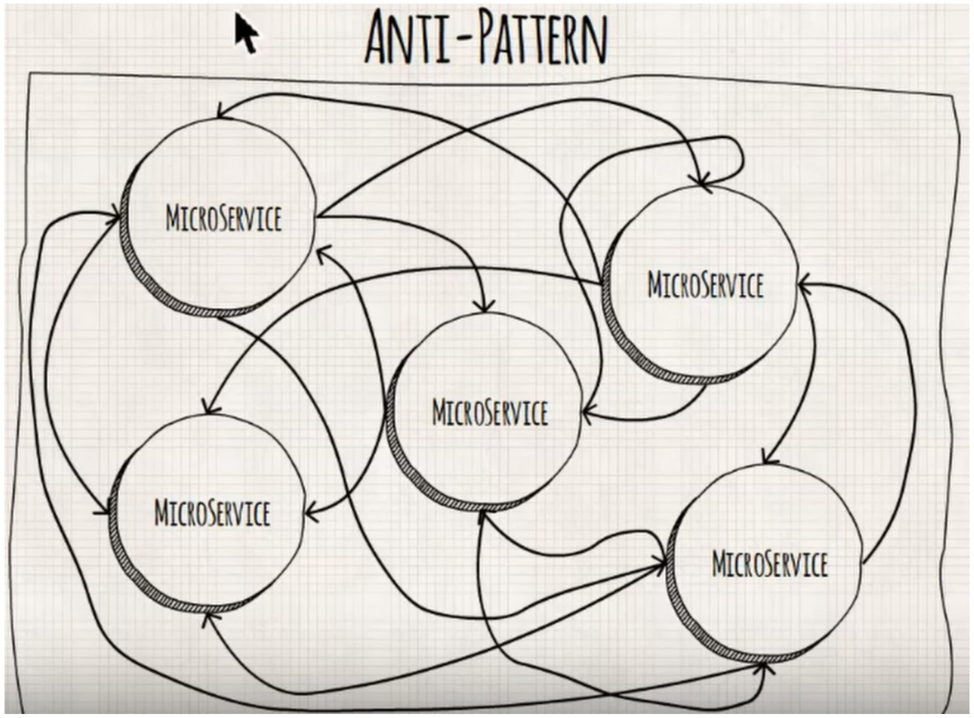
Domain Driven design (DDD)

- Concepts we'll cover:
  - Bounded contexts
  - Aggregates
  - Event sourcing (event driven state)
    - Command query responsibility segregation (CQRS)

Chatty services→ Big ball of Mud

- Don’t allow services to talk directly to each other whenever they want → maintenance/traceability nightmare

What can we do ?!

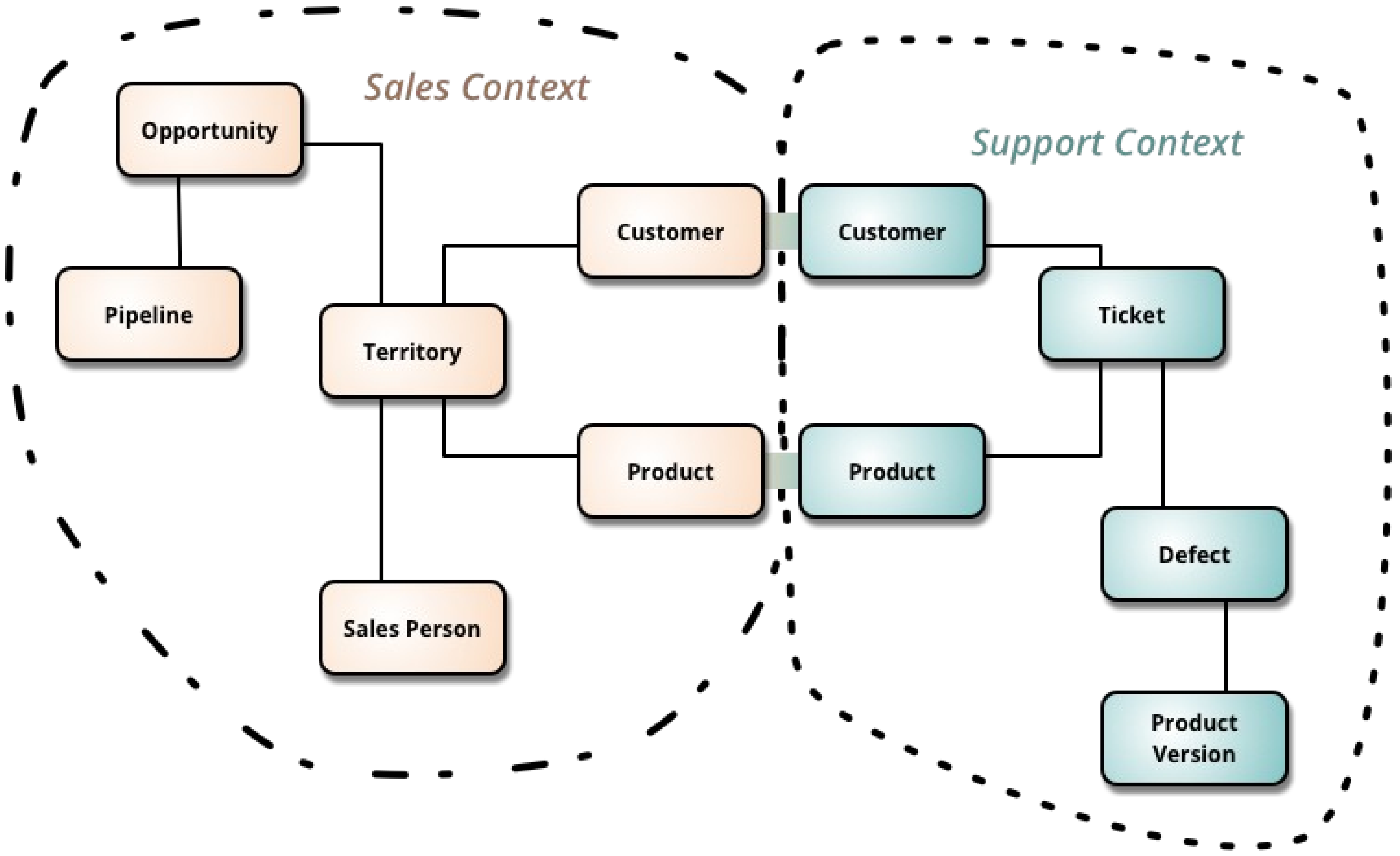


https://www.youtube.com/watch?v=zppvtwqcjv4

DDD

- Start here:
  - http://domainlanguage.com/ddd/

Bounded Context



https://martinfowler.com/bliki/BoundedContext.html

Bounded Context

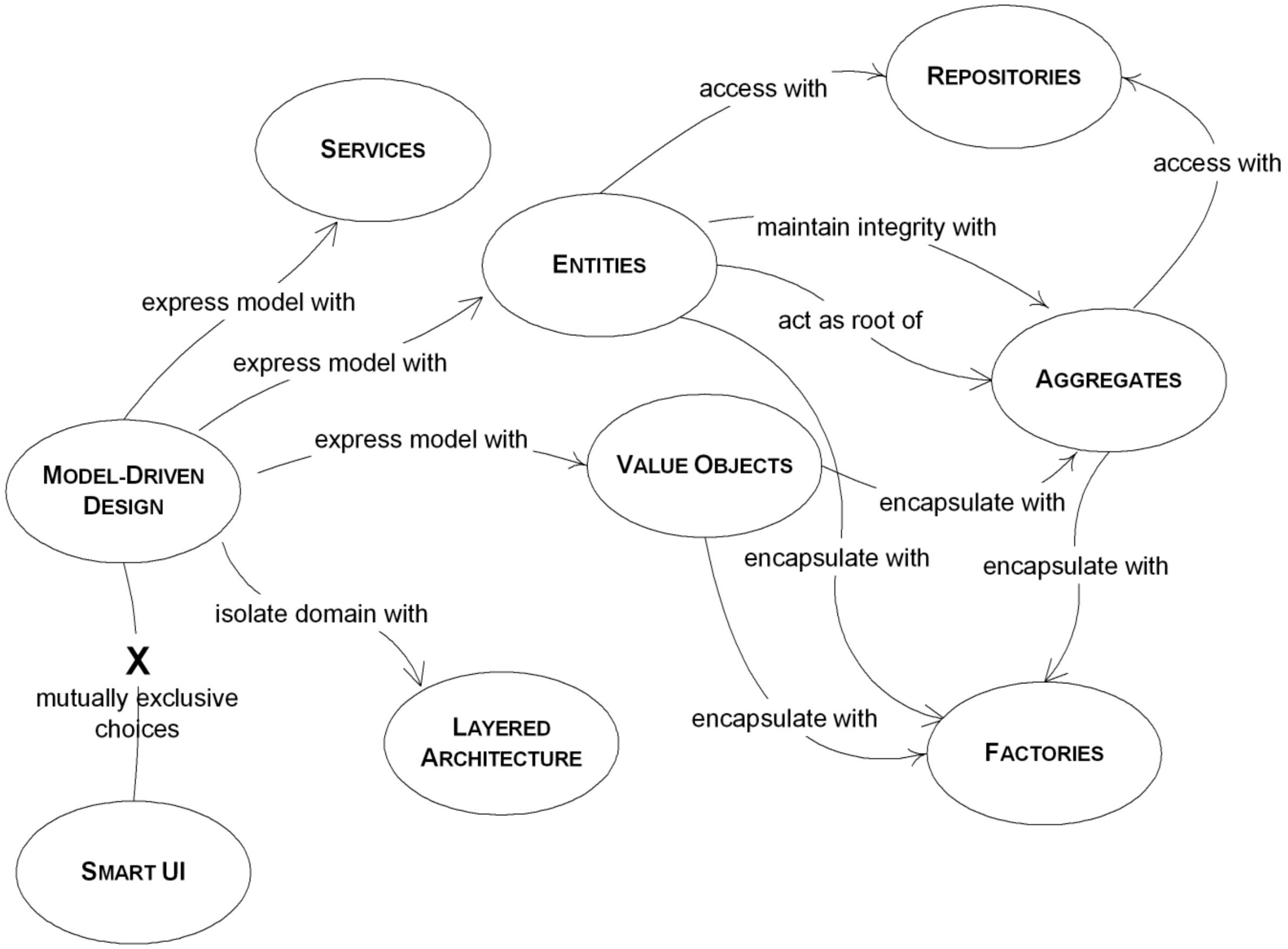
- Bounded Context
  - A subsystem or the work of a particular team
- e.g. a Customer defined differently in different bounded contexts
  - Sales
  - Marketing
  - Orders

Bounded context

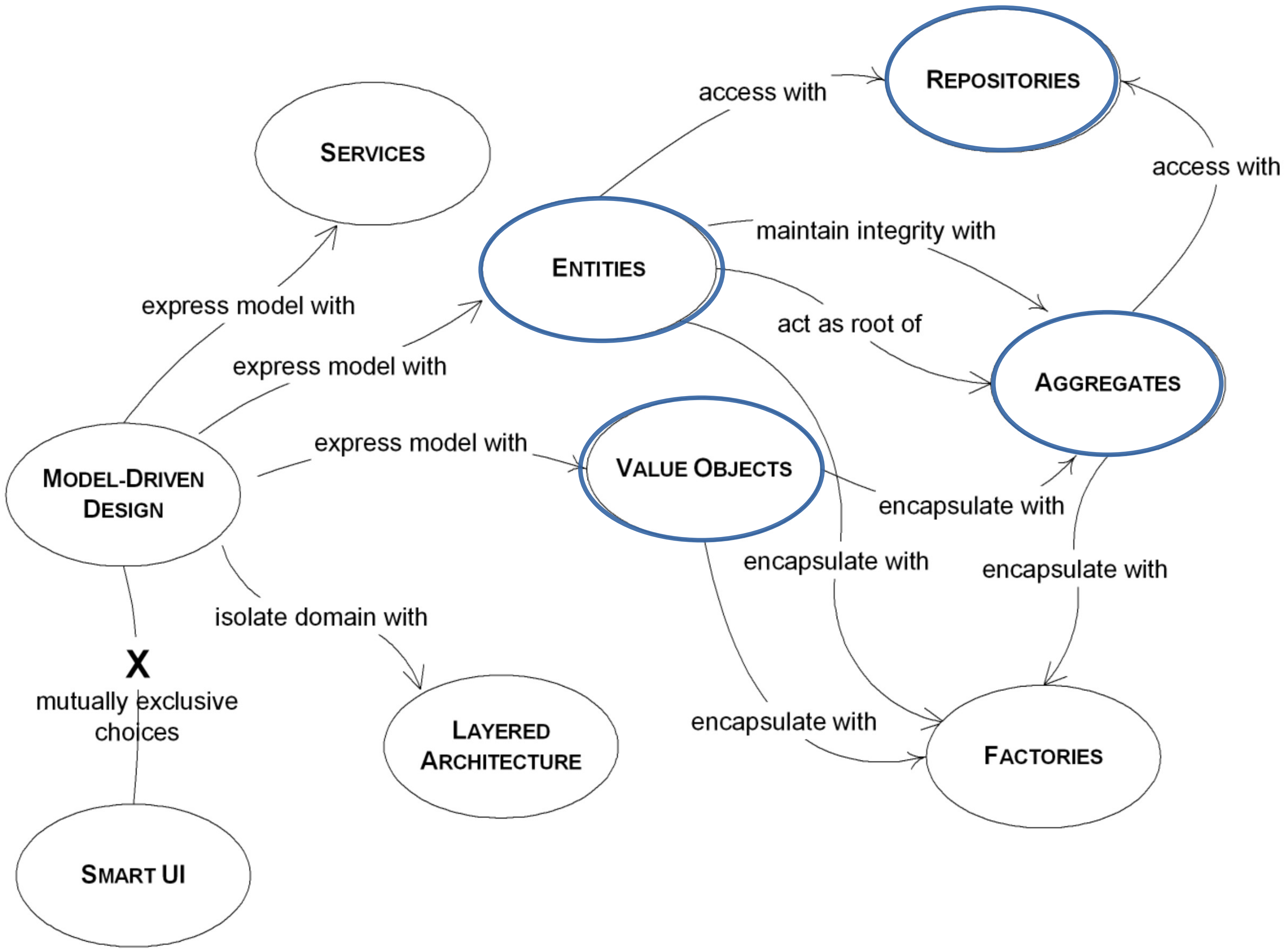
- Read:
  - https://martinfowler.com/bliki/BoundedContext.html

Event Driven

- Microservices - Event Driven
  - https://www.youtube.com/watch?v=stdHZ6B8q0Q



http://static.olivergierke.de/lectures/ddd-and-spring/images/ddd-building-blocks.png



http://static.olivergierke.de/lectures/ddd-and-spring/images/ddd-building-blocks.png

Value Object

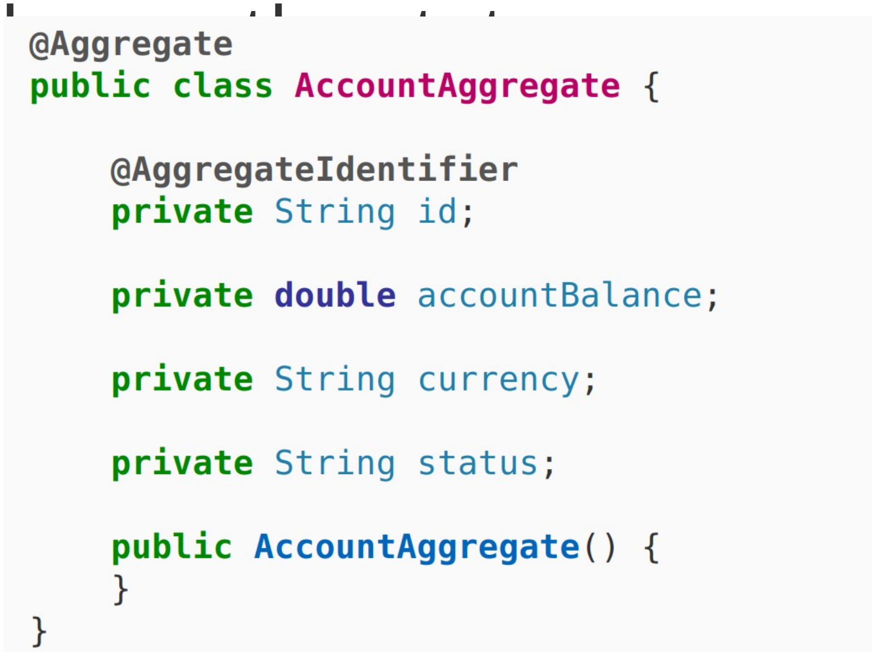
- Immutable, has no Identity (ID)
- Two value objects are equal if they have same set of values for their properties
- Examples:
  - Shipping Info (street address , name , zipcode,..)
  - Payment method (CardType, number , expiration date, ccv,..)

Entity

- Represents a concept
- Defined by its ID , even when we change other attributes later
  - e.g Customer , order
- Life cycle and Integrity are handled by Aggregate

Aggregate

- Represents your main Domain Object
- An Aggregate is an entity or group of entities that is always kept in a consistent state
- Validate commands, and apply/generate domain events to c



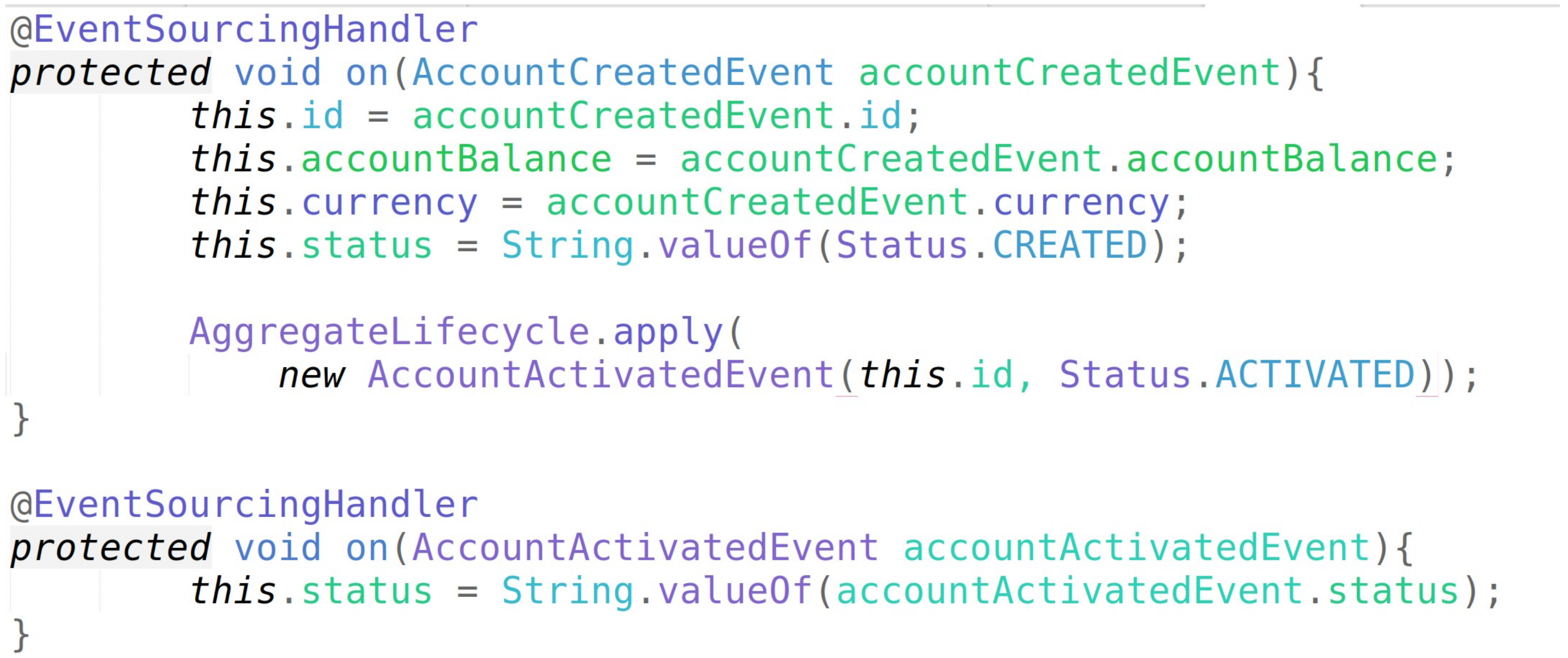
```java
@Aggregate
public class AccountAggregate {

    @AggregateIdentifier
    private String id;

    private double accountBalance;

    private String currency;

    private String status;

    public AccountAggregate() {
    }
}
```

Domain Event

- Result of Commands or other events
- Generated and applied on Aggregates to change their state
  - Account created , transaction processed , last name changed , account deactivated ,..

Notice the past tense, as aggregate already did the state validation

Domain Event



```java
@EventSourcingHandler
protected void on(AccountCreatedEvent accountCreatedEvent){
        this.id = accountCreatedEvent.id;
        this.accountBalance = accountCreatedEvent.accountBalance;
        this.currency = accountCreatedEvent.currency;
        this.status = String.valueOf(Status.CREATED);

        AggregateLifecycle.apply(
            new AccountActivatedEvent(this.id, Status.ACTIVATED));
}

@EventSourcingHandler
protected void on(AccountActivatedEvent accountActivatedEvent){
        this.status = String.valueOf(accountActivatedEvent.status);
}
```

Repository

- Separate the state representation (Aggregate) from back end storage/ persistence
  - An aggregate can be represented as a set of events/messages on an event store (e.g. queue)
    - Repository can snapshot state→ don’t replay from first event every time

Commands

- Means to change state of the system
  - They're immutable themselves
  - can be sent as a message over a queue
  - Has payload for details
- e.g. CreateAccount:
  - ID , Person Info , account type ,....

Commands

- The Aggregate handles the command
  - Apply validation , check current state
  - Generate further events
    - e.g. Account created event , transaction rejected ,..

Query

- Request of some state of the aggregates
  - Read Only messages requests
- The query might be served by different back ends, or materialized views!
  - Search order by desc text → Elastic search
  - Fetch latest orders → time stamped index like Cassandra!

How they communicate then?

Event storming

- Define all the events between different bounded contexts

Event flows

- Define upstream--downstream contexts
  - A change in A generates event(s) , subscribed to by B

Event sourcing

- Don’t store state → Store all the stream of events that lead to that state
- Append only → No Updates on the event store
- Some Domain problems are append only by nature:
  - Think of your medical history, criminal record, contract/constitution amendment accounting ,..

Event sourcing

- With Distributed queues →elastic, responsive
- Decouple services
- Time travel!
  - Create future services as if they existed from day 1 (by replaying events)

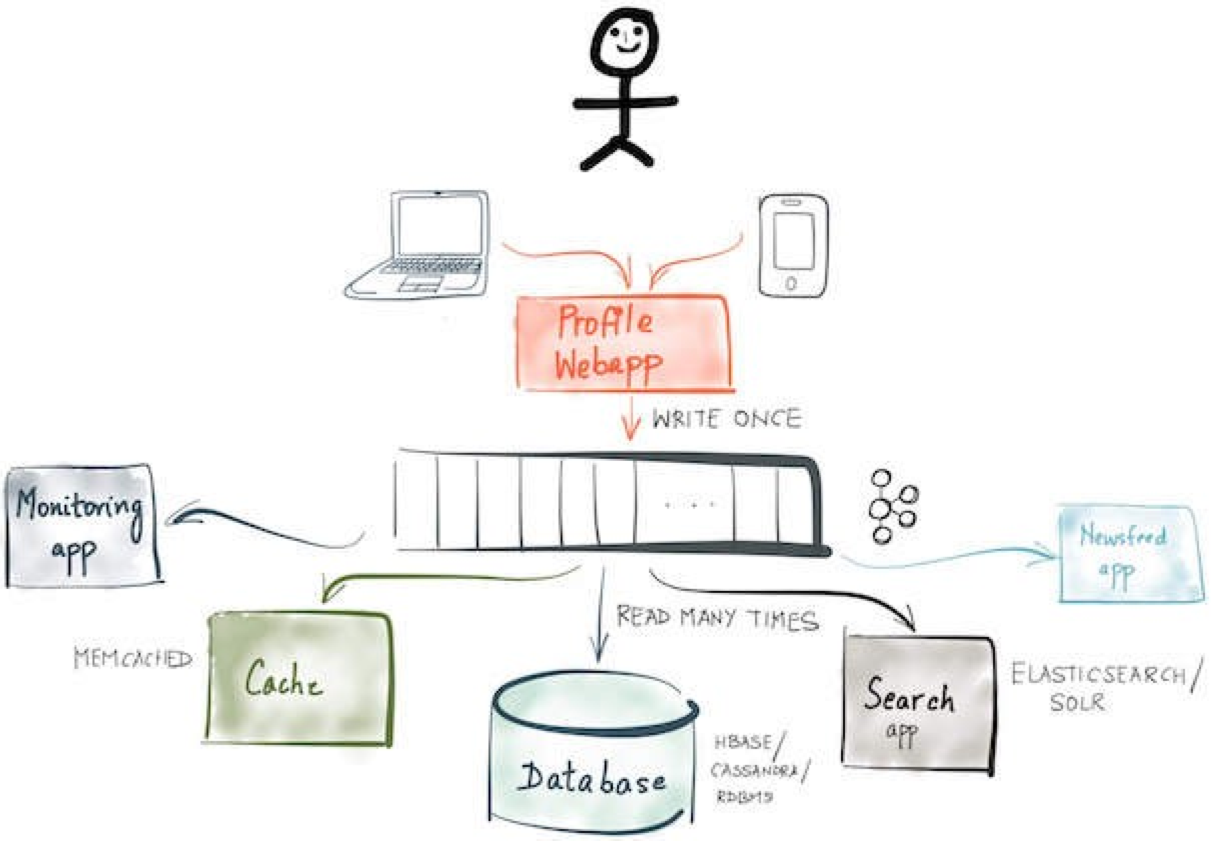
Event sourcing

- Cons:
  - Systems become eventually consistent
  - Increased Complexity , learning curve
  - More storage requirement (Cheap)
  - Poor initial event design is hard to change later
  - Possibility of duplicate or out of order event delivery, subscribers needs to handle it.

Event sourcing

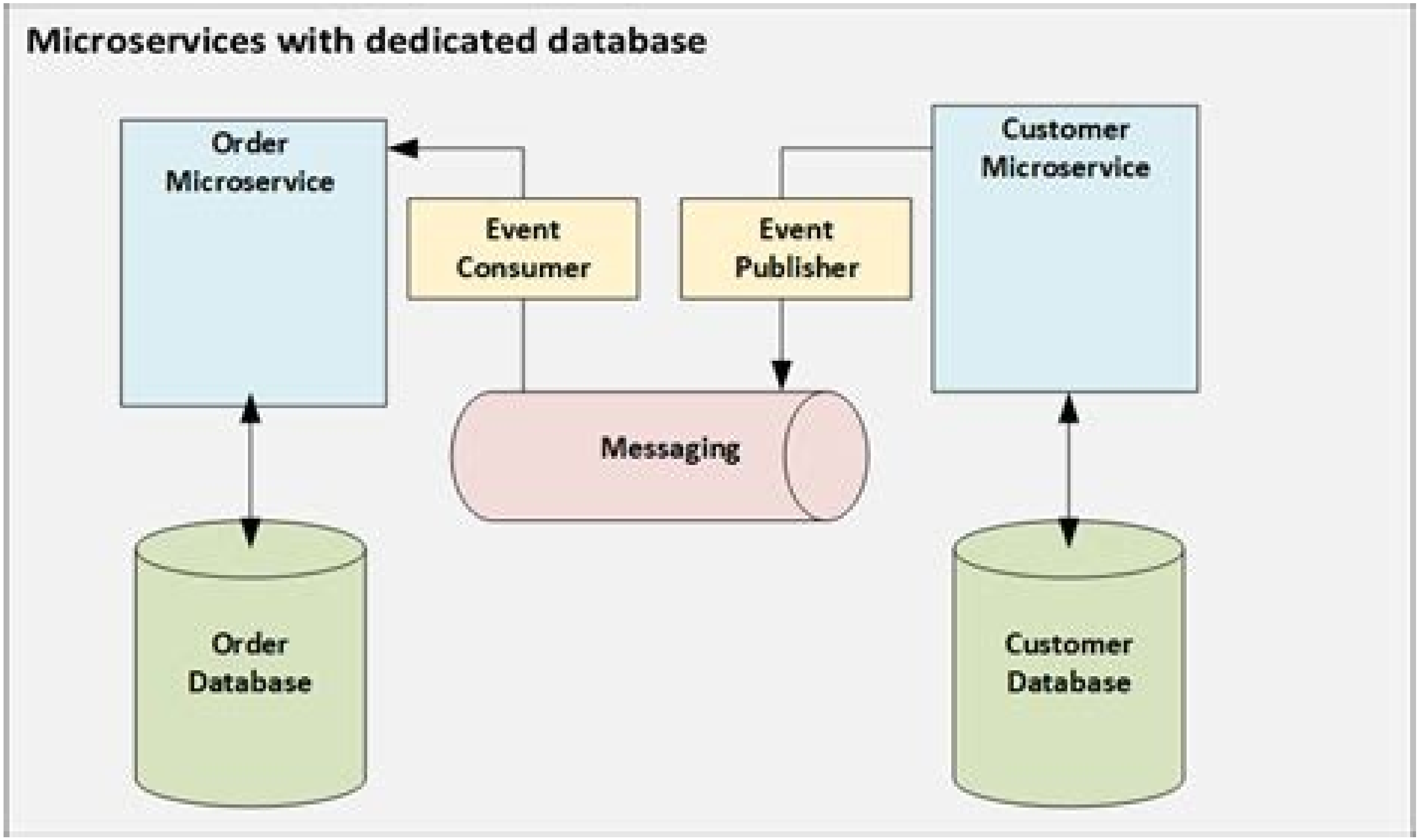
- Let’s read:
  - https://martinfowler.com/eaaDev/EventSourcing.html

Event sourcing

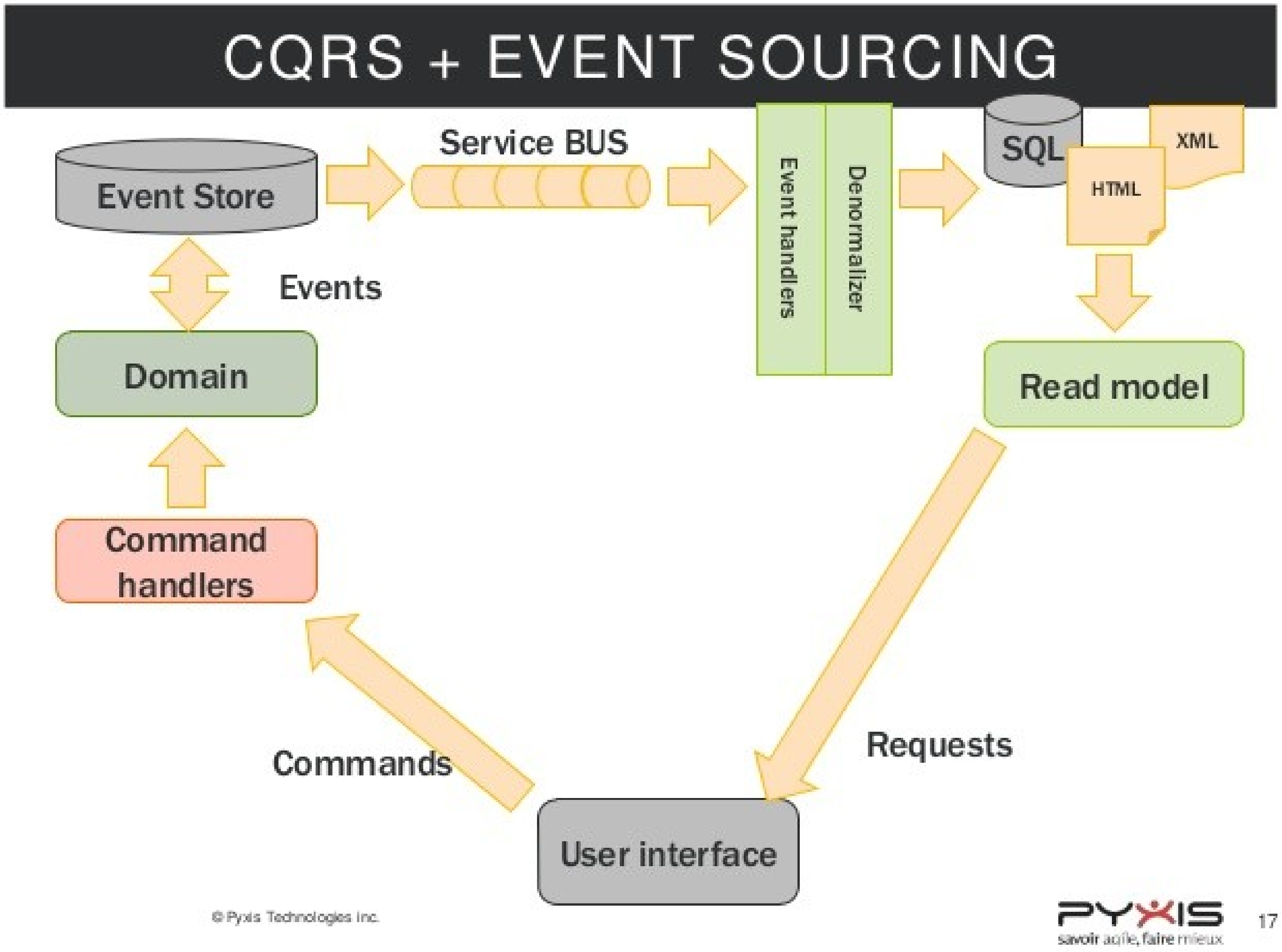


https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/

Event sourcing



Where’s the state
How to make different queries ?



CQRS + EVENT SOURCING

© Pyxis Technologies Inc.

https://www.slideshare.net/ericdecarufel/cqrs-event-sourcing-pyxis-v2-en

Why CQRS+ES ?

- Two microservices can be optimized separately, yet communicate over event store (e.g. Queue)
- Segregation allows query services to build materialized views/Projections optimized for each query
  - Elastic search query
  - Timestamped Cassandra
  - Cached or In memory ,…

Still confused ?

- Watch: Introduction to CQRS - Event Sourcing, Distributed Systems & CQRS
  - https://www.youtube.com/watch?v=qJA6MaQ90YY

DDD, ES , CQRS

- Check out Axon framework for java
  - https://axoniq.io/resources/architectural-concepts
- Let’s read
  - https://www.slideshare.net/opencredo/a-visual-introduction-to-event-sourcing-and-cqrs-by-lorenzo-nicora

Let’s see it in action

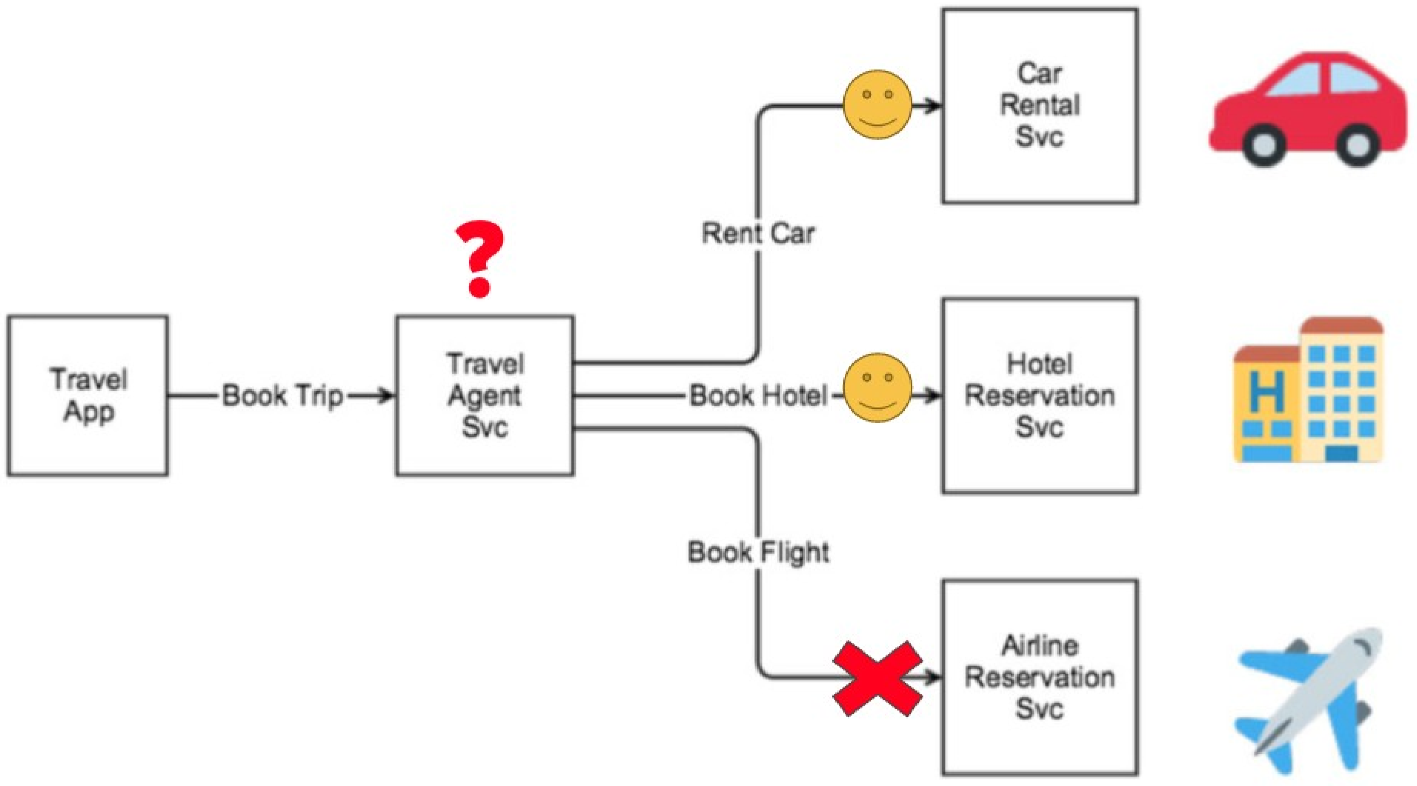
- Let’s watch (45 min):
  - Building Event-Driven Microservices with Event Sourcing and CQRS | Lidan Hifi
    - https://www.youtube.com/watch?v=XWTrcBqXi6s

Aggregates/ES in Spring

- Watch this by yourself (1hr):
  - Developing microservices with aggregates | Chris Richardson
    - https://www.youtube.com/watch?v=7kX3fs0pWwc

How to perform logic/transaction ?
“In a distributed manner”

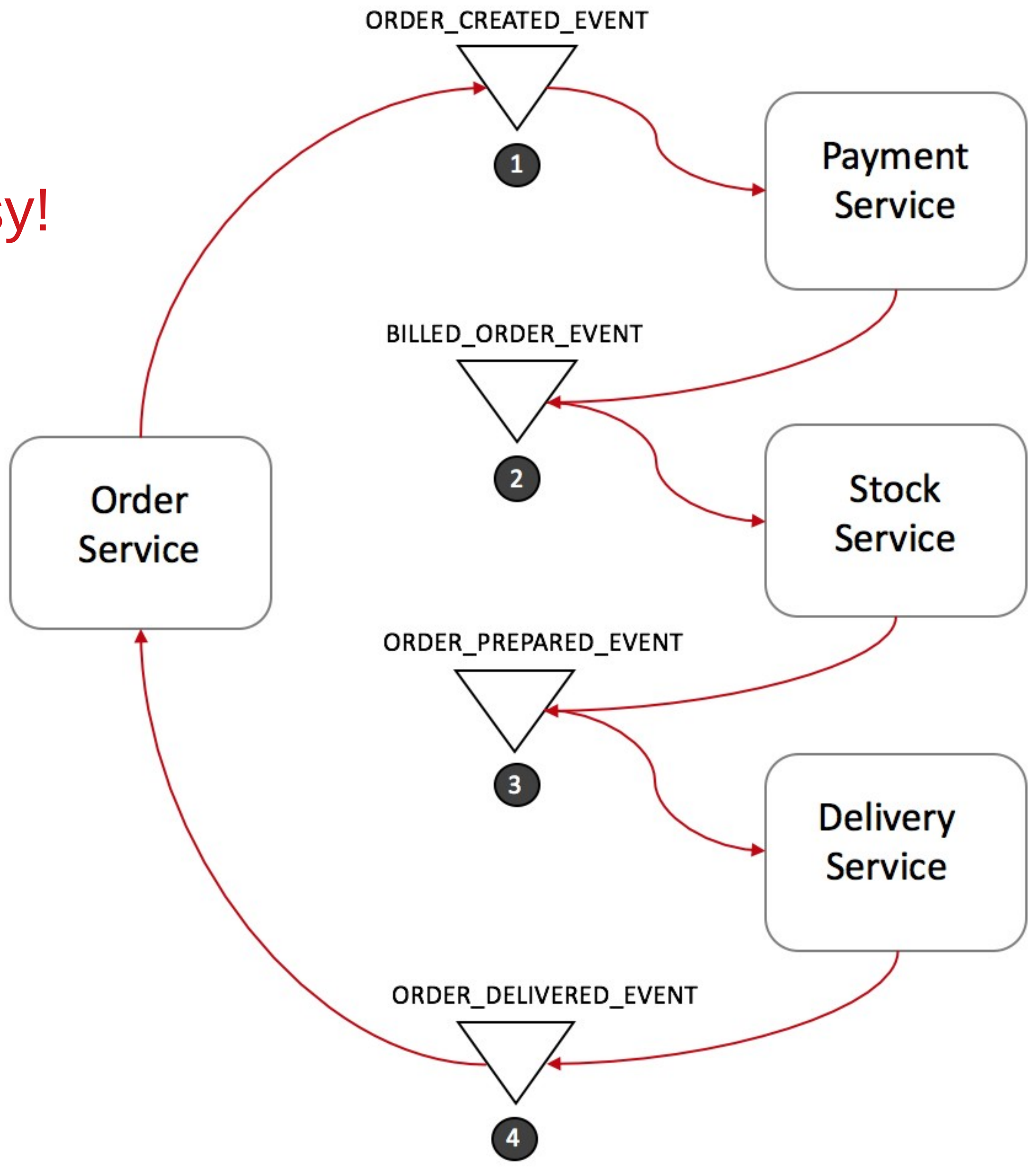
Distributed Sagas



Saga

- Captures one business process of flow
- Communicates with several other services
- Handles failures , rollback
  - e.g. cancel orders if payment service fail
- Not so easy :(

Not Easy!



https://blog.couchbase.com/wp-content/uploads/2018/01/Screen-Shot-2018-01-09-at-6.13.39-PM.png

Distributed Saga

- Let’s read:
  - https://dzone.com/articles/distributed-sagas-for-microservices
- Also read:
  - https://dzone.com/articles/saga-pattern-how-to-implement-business-transaction
  - https://dzone.com/articles/saga-pattern-how-to-implement-business-transaction-1