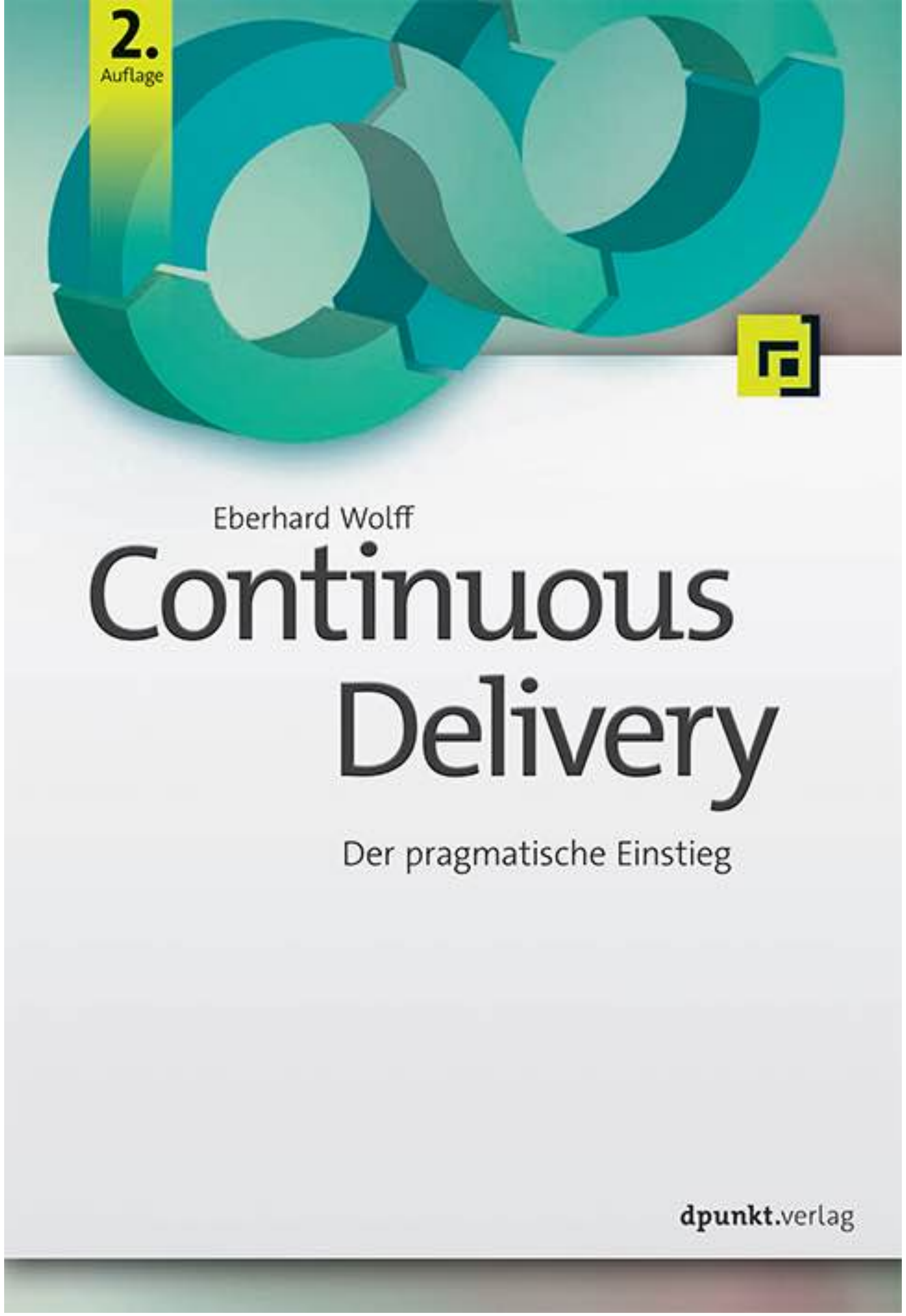
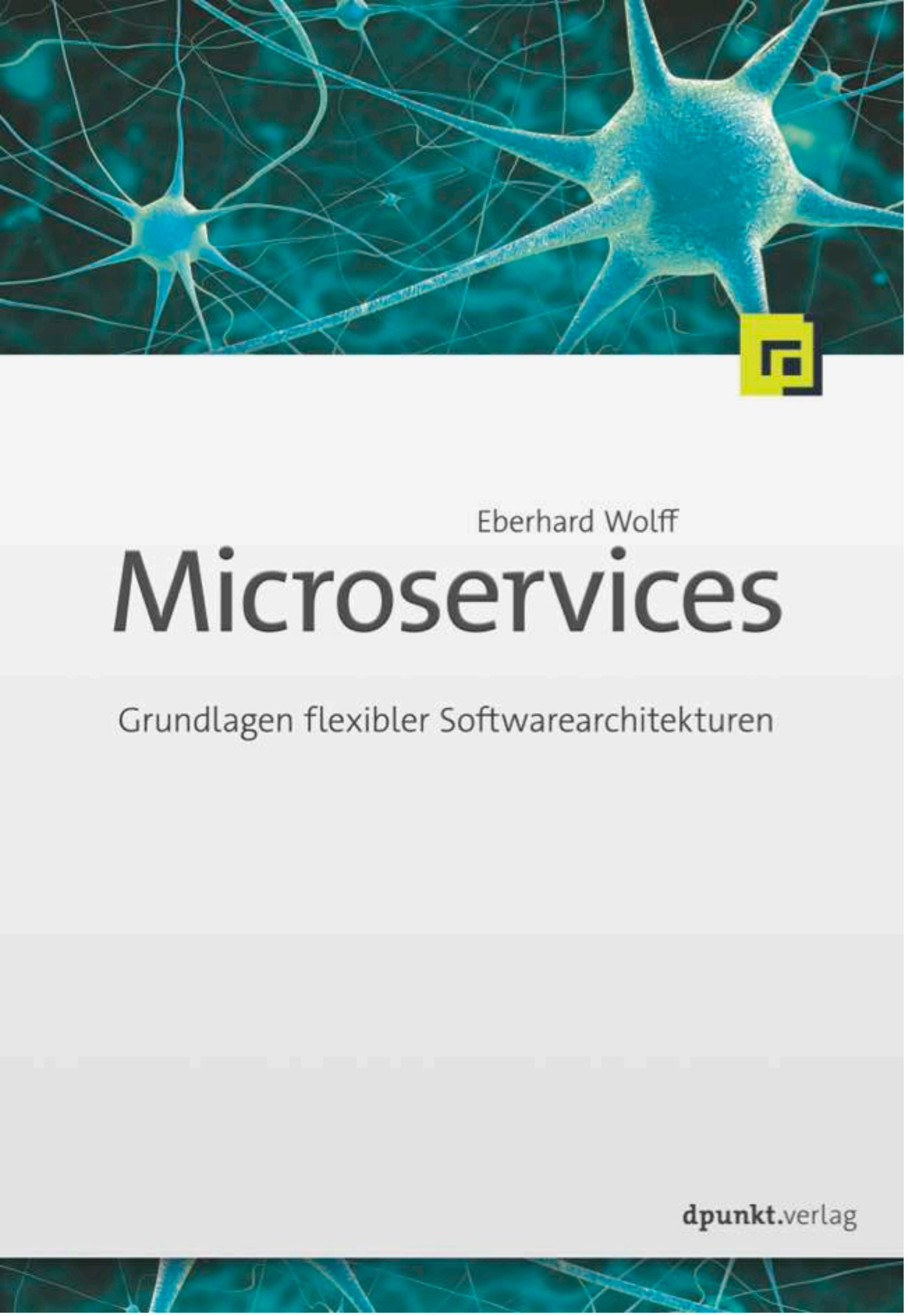
Data Architecture – Not Just for Microservices

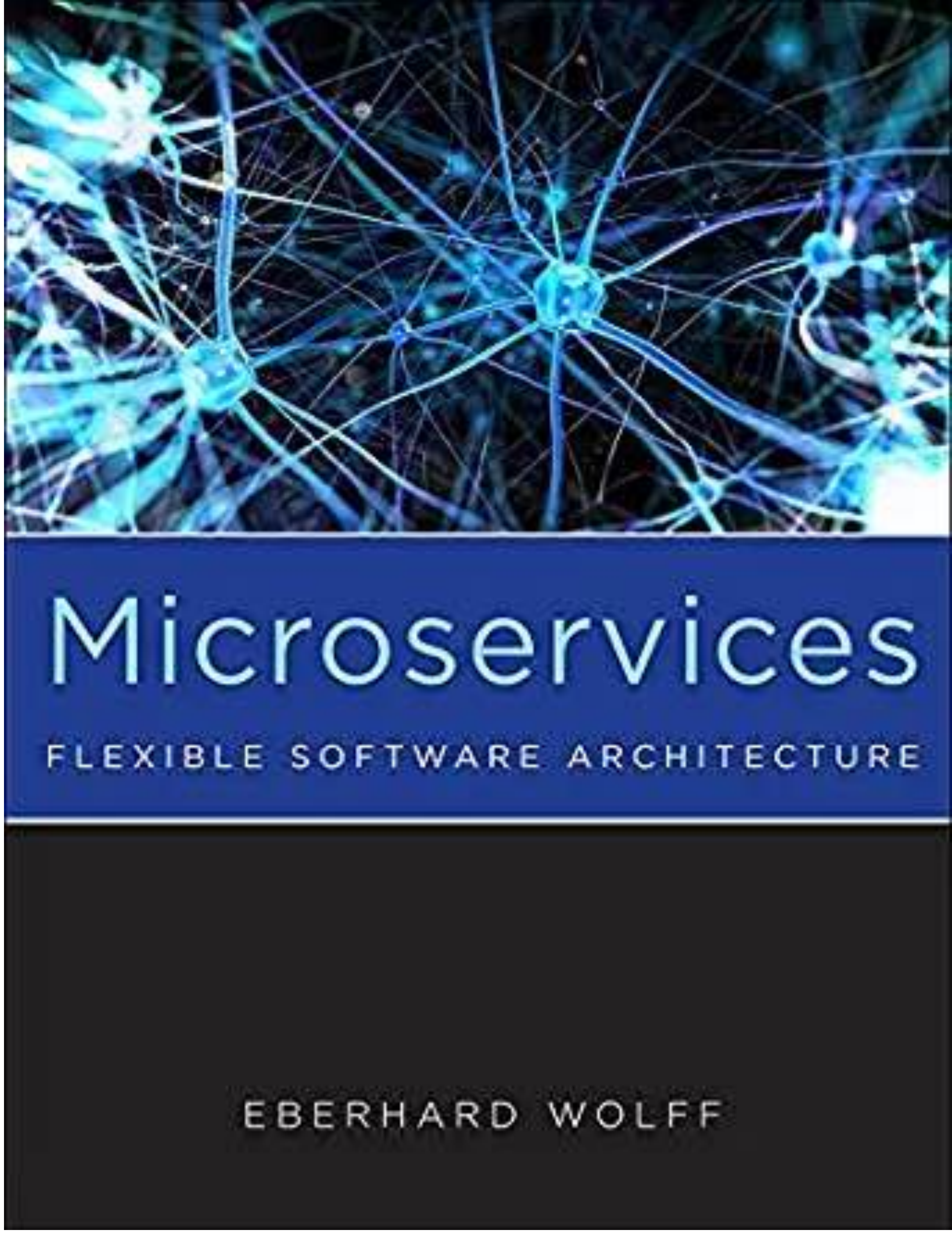
Eberhard Wolff
@ewolff
Fellow

innoQ

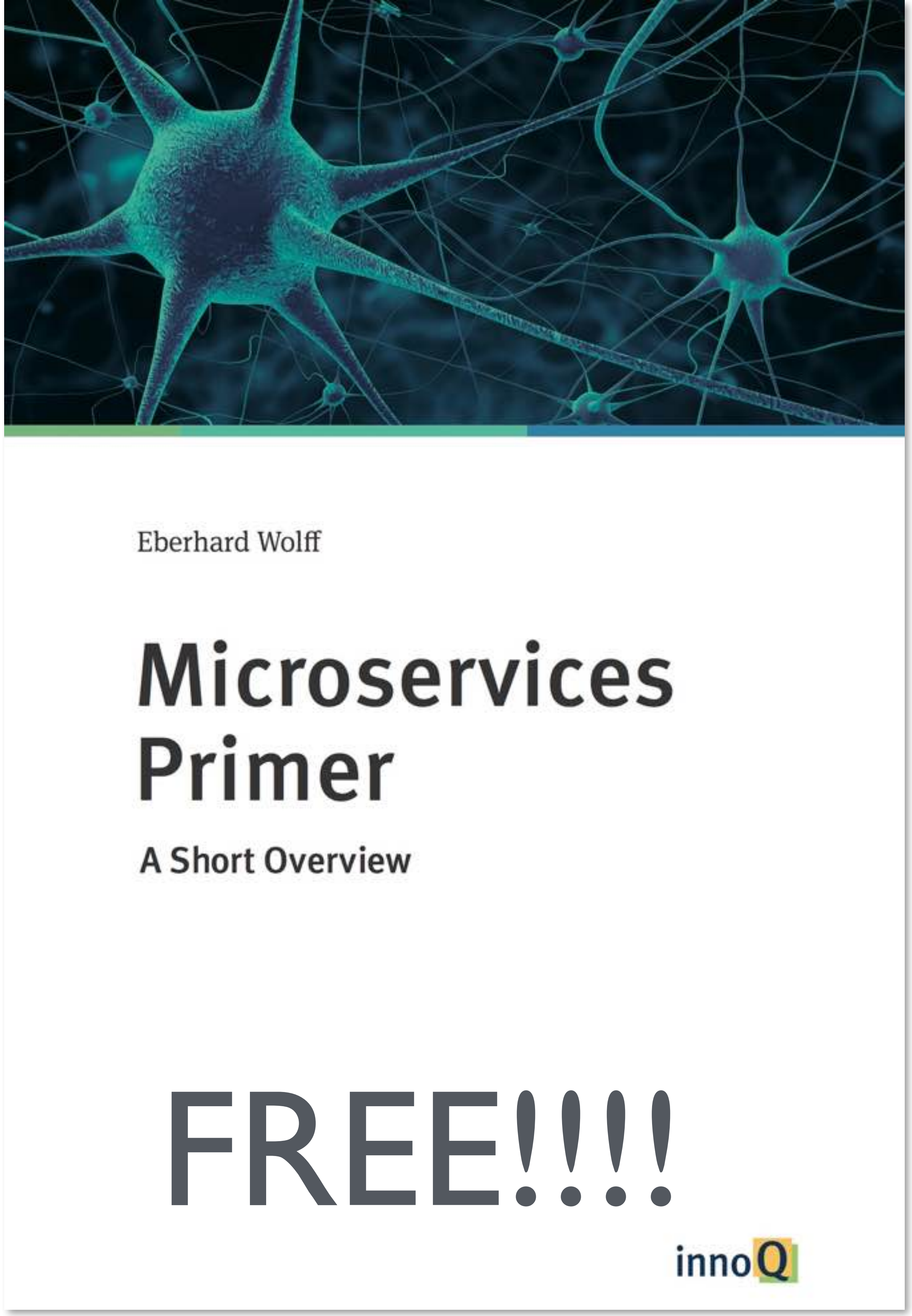

2.
Auflage
Eberhard Wolff
Continuous Delivery
Der pragmatische Einstieg
dpunkt.verlag

http://continuous-delivery-buch.de/





http://microservices-buch.de/ http://microservices-book.com/


Eberhard Wolff

Microservices Primer

A Short Overview

innoQ

FREE!!!!

http://microservices-book.com/primer.html

Volume 9 entwicklerspezial

MICROSERVICES

DIE ANDERE ART DER MODULARISIERUNG

ARCHITEKTUR
Die Vor- und Nachteile von Microservices

TECHNOLOGIEN
Serverlose Microservices mit Lambda

KULTUR
Wie Microservices Unternehmen verändern

DE € 9,80
AT € 10,80
CH sFr 19,50
L € 11,00

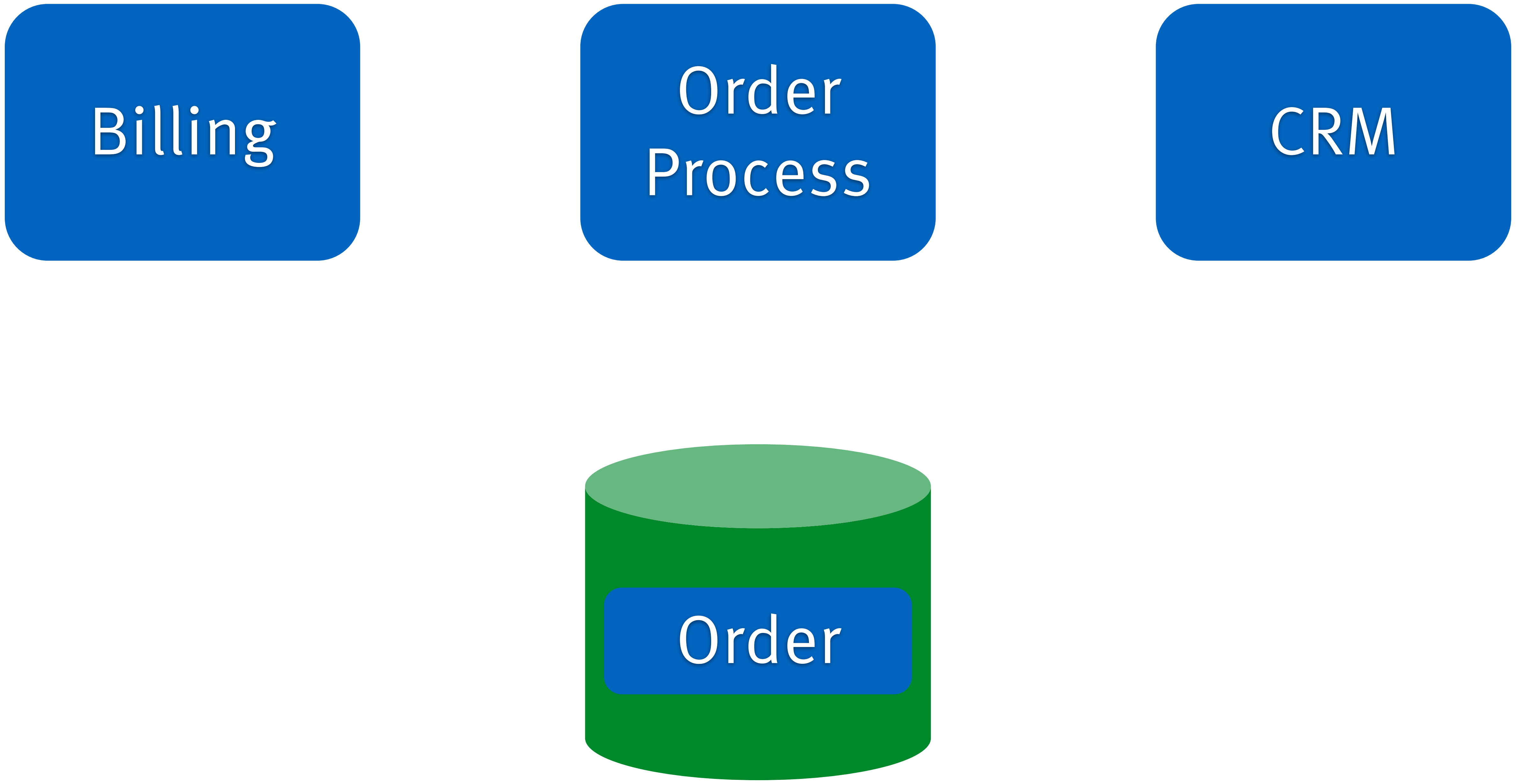
www.entwickler-magazin.de

+++ Architektur +++ Kultur +++ Entwicklung +++ Technologien +++ Domain-driven Design +++ Netflix +++

Classic Data Architecture

- Centralized databases
- …or services that provide data
- Ensures consistency across systems
- …for data model
- …and updates to data
- Reuse

Classic Data Architecture



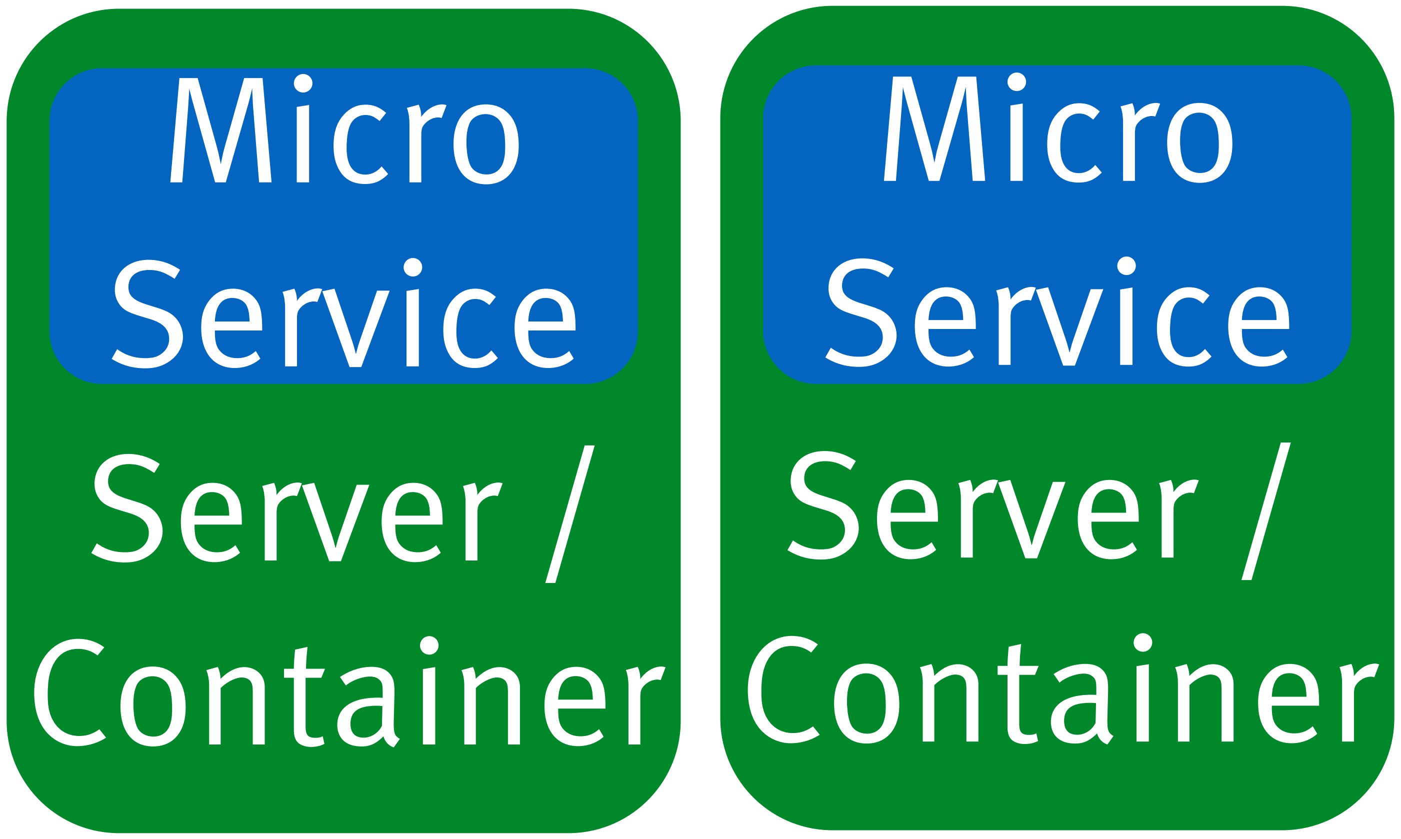
Who is using a centralized database?

Who likes the centralized database?

Microservices: Definition

- No consistent definition
- Microservices are modules
- Independent deployment units
- E.g. processes, Docker container
- Microservice owned by one team

Microservices: Definition



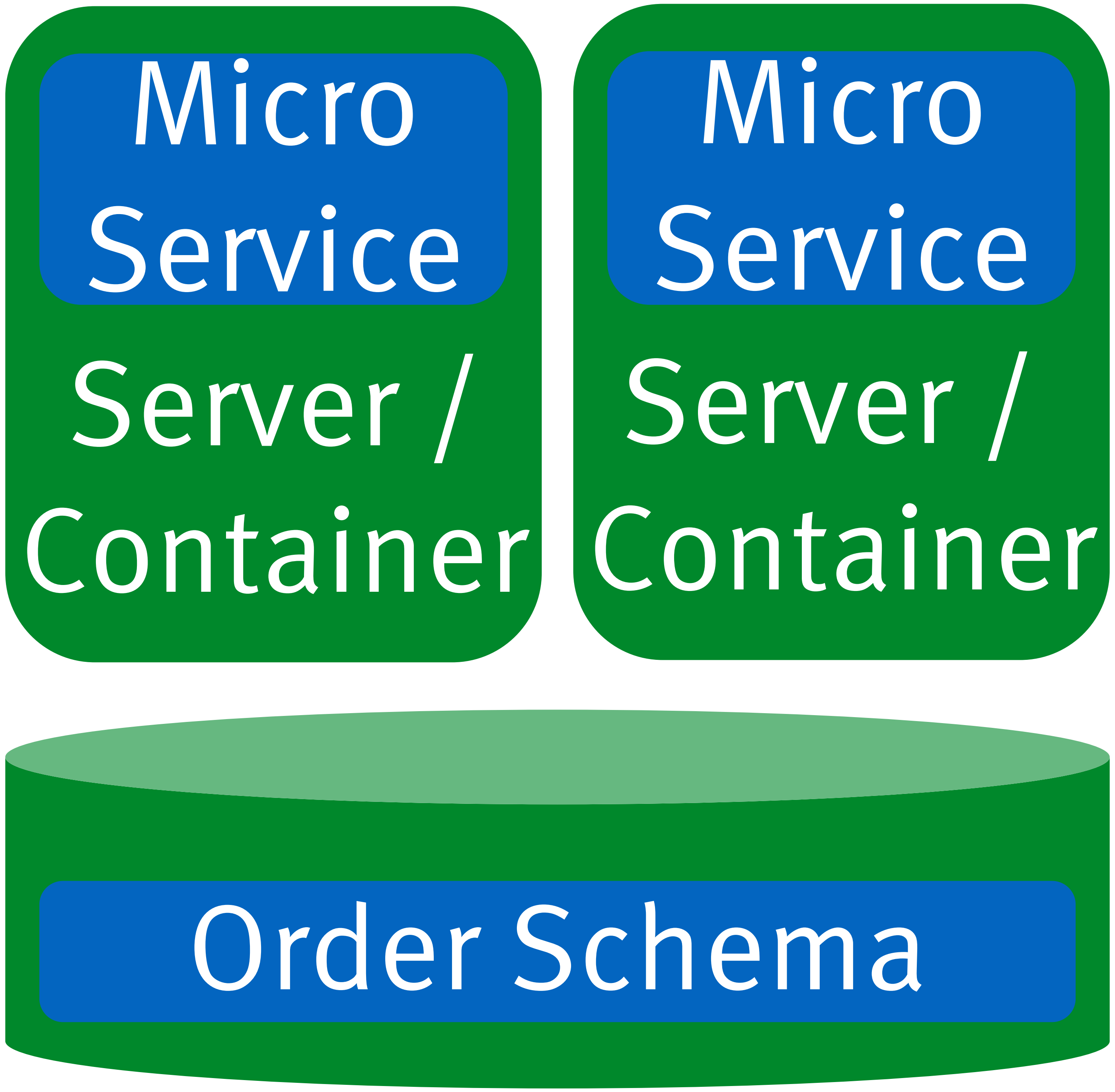
Why Microservices?

- Develop a feature
- …bring it into production
- …with no coordination

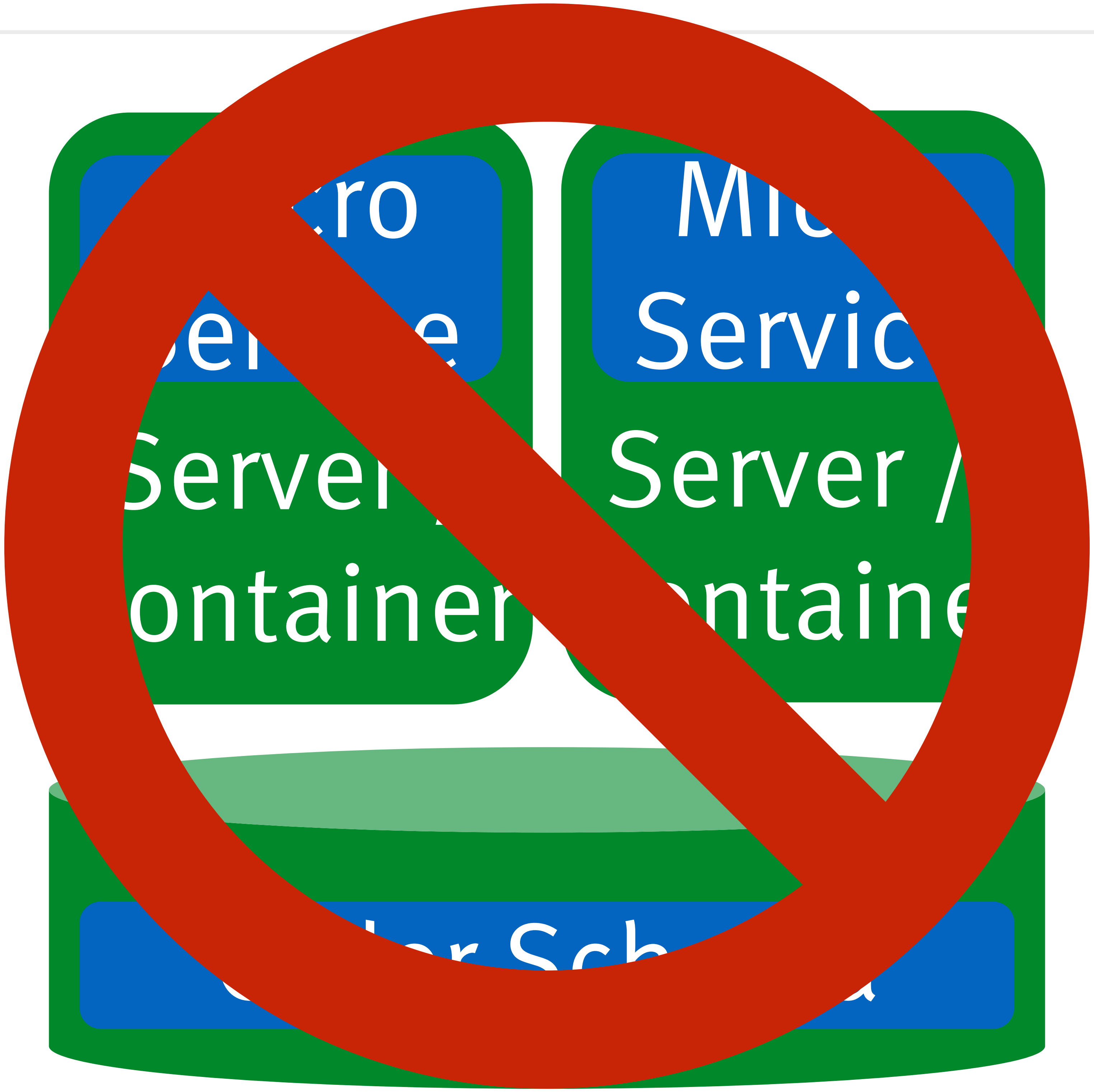
- Independent scaling
- Free choice of technology
- Robustness
- Security

Microservices aim for decoupling

Microservices & Data



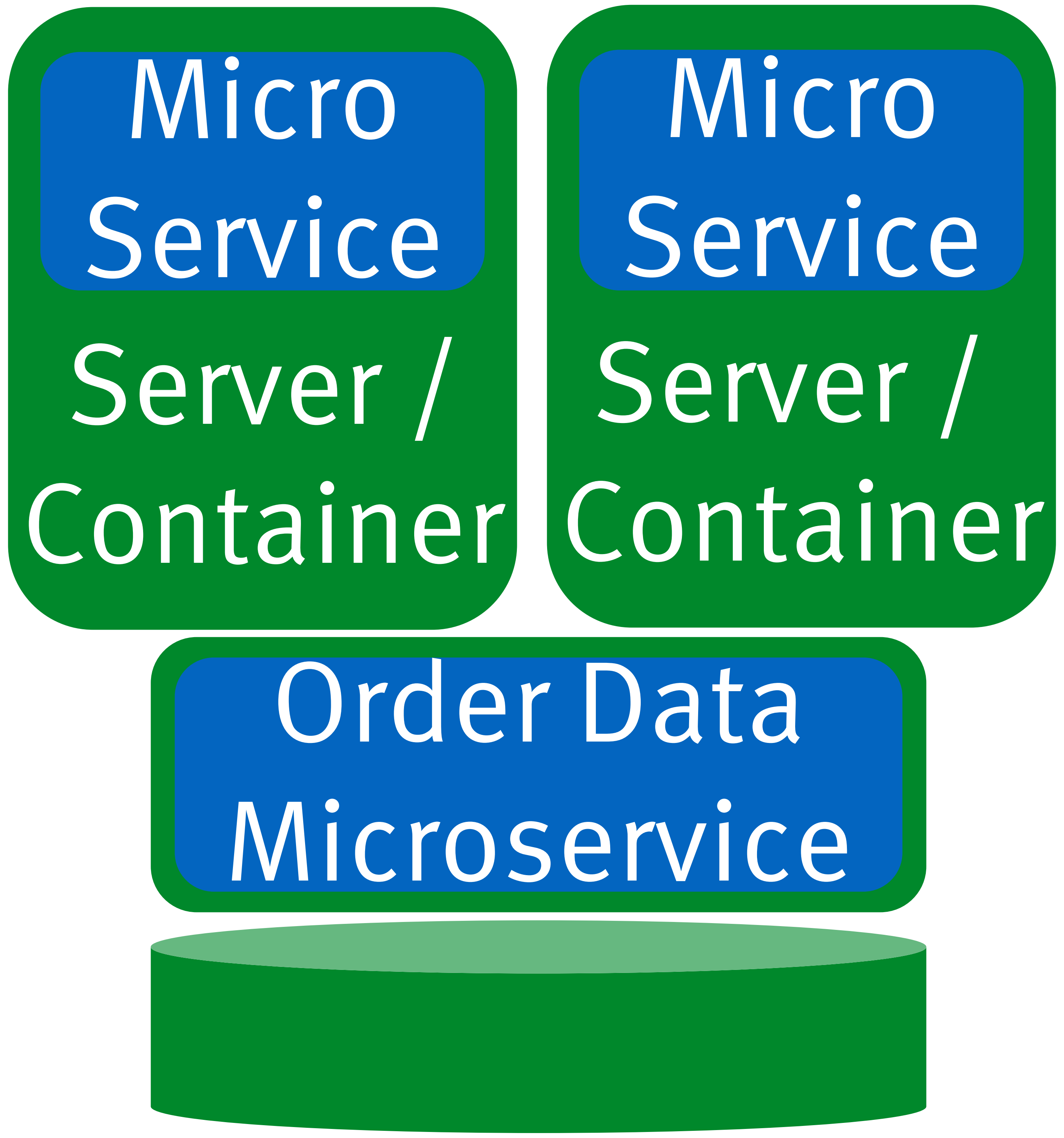
Microservices & Data



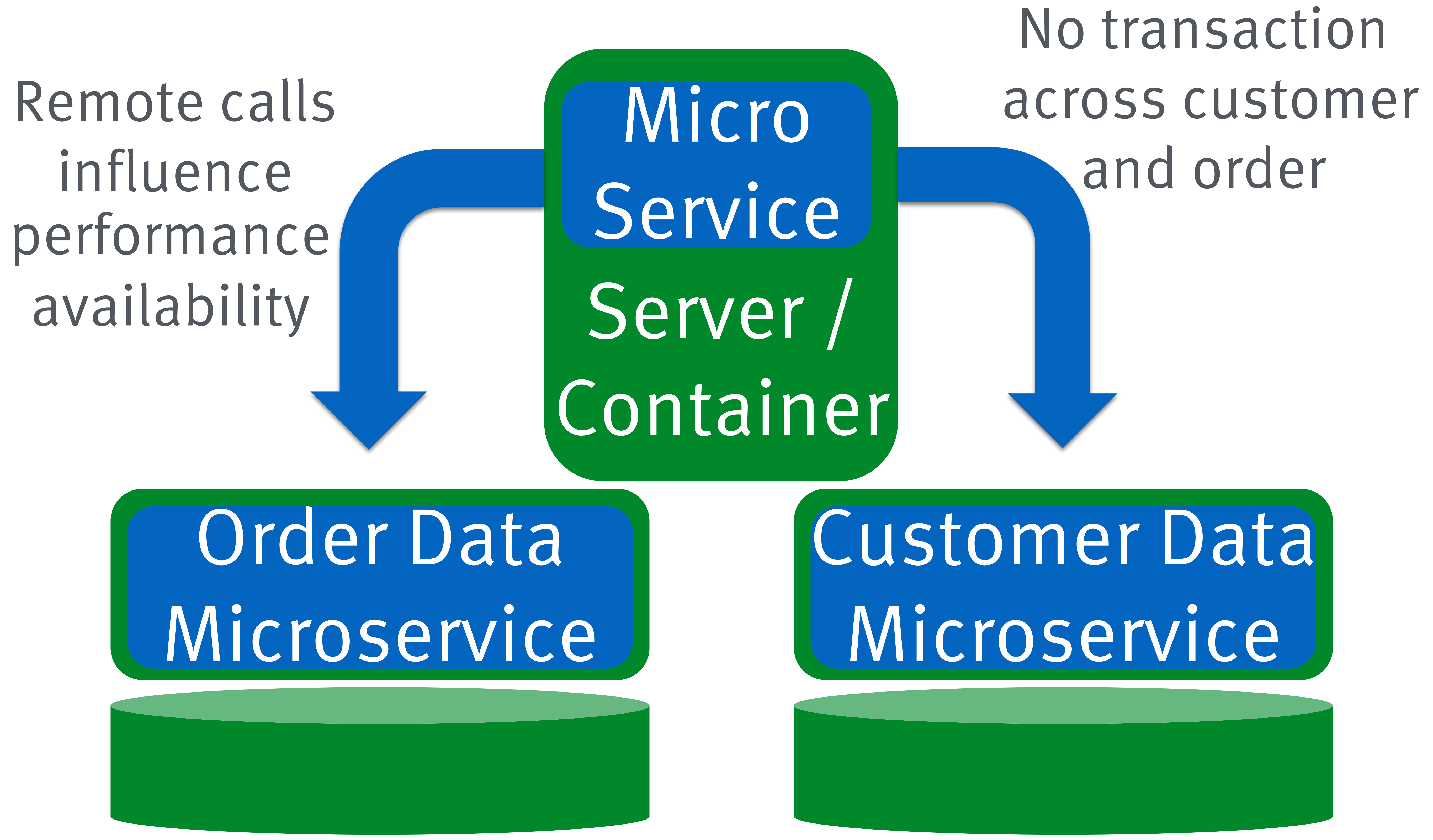
Microservices & Data

- Decoupling for data, too
- Separate data storage

Data Microservices



Data Microservices


Remote calls influence performance availability
No transaction across customer and order
Micro Service
Server / Container
Order Data Microservice
Customer Data Microservice

Data Microservice

- Change two microservices if new feature requires change to data schema
- Transactions?
- But: data in one place
- No consistency issues

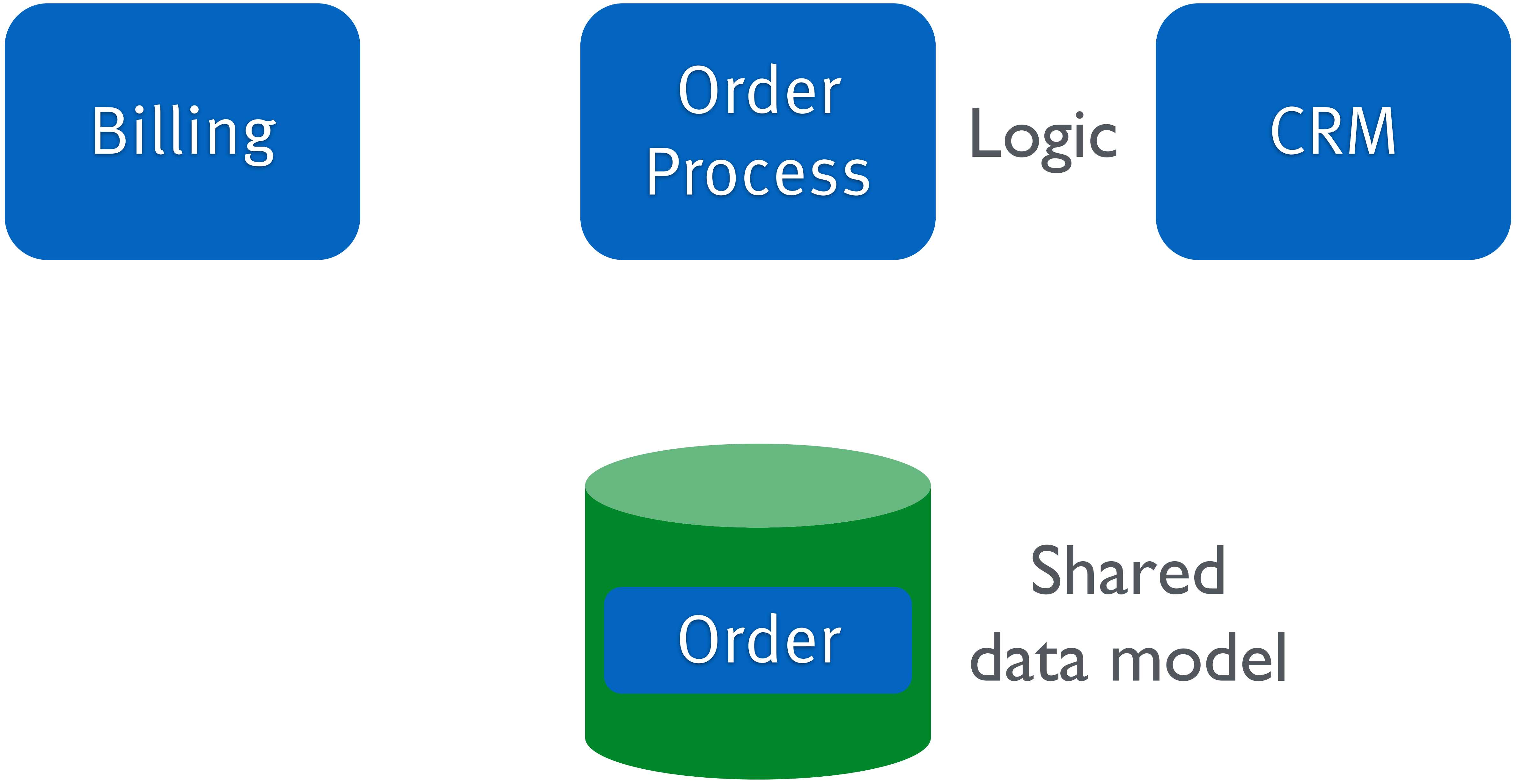
Data microservice limits decoupling.

Encapsulation

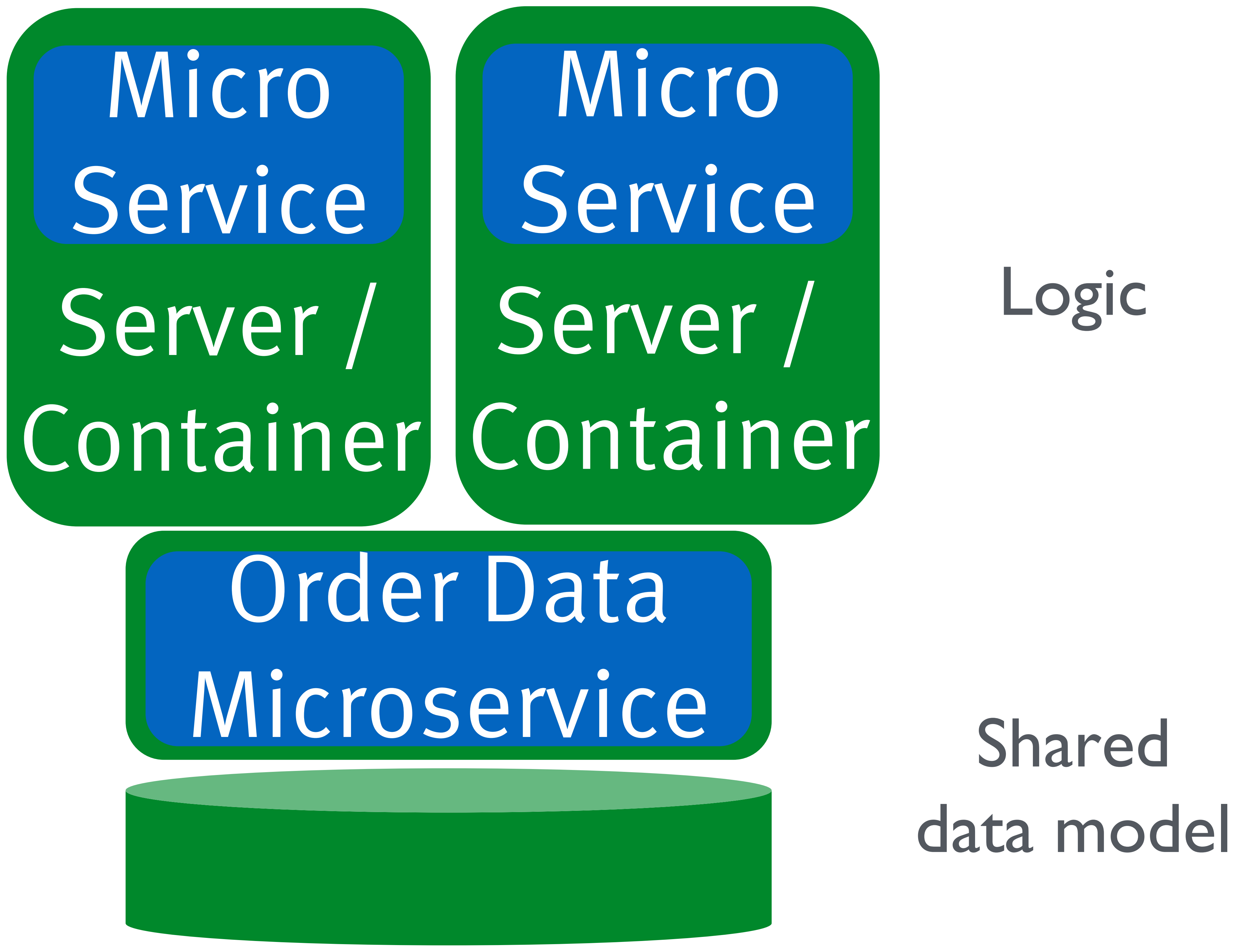
- Information hiding
- Hide the internal data structure

- Provide access only through a well defined interface
- Data and databases should not be exported

Violates Encapsulation



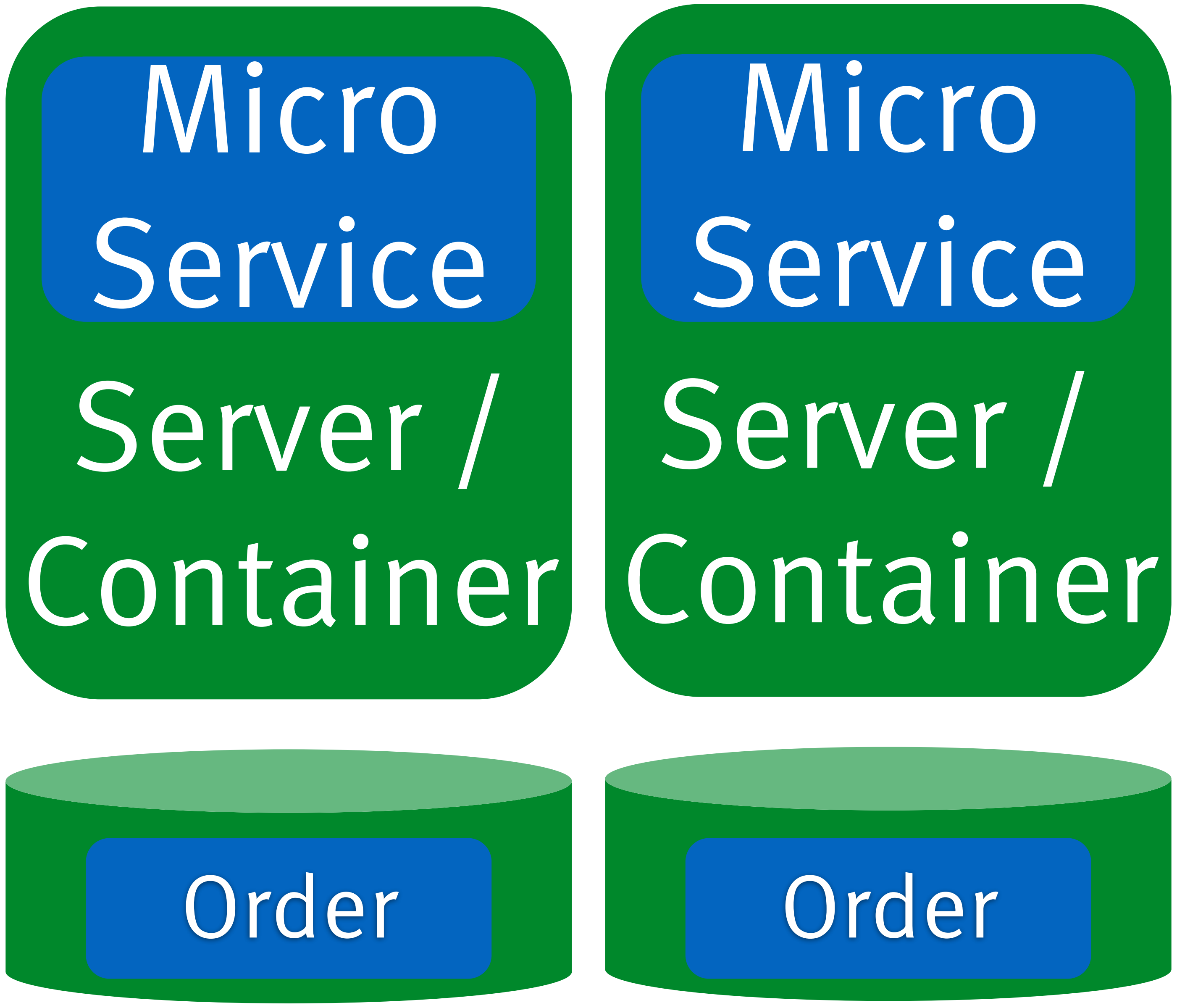
Violates Encapsulation



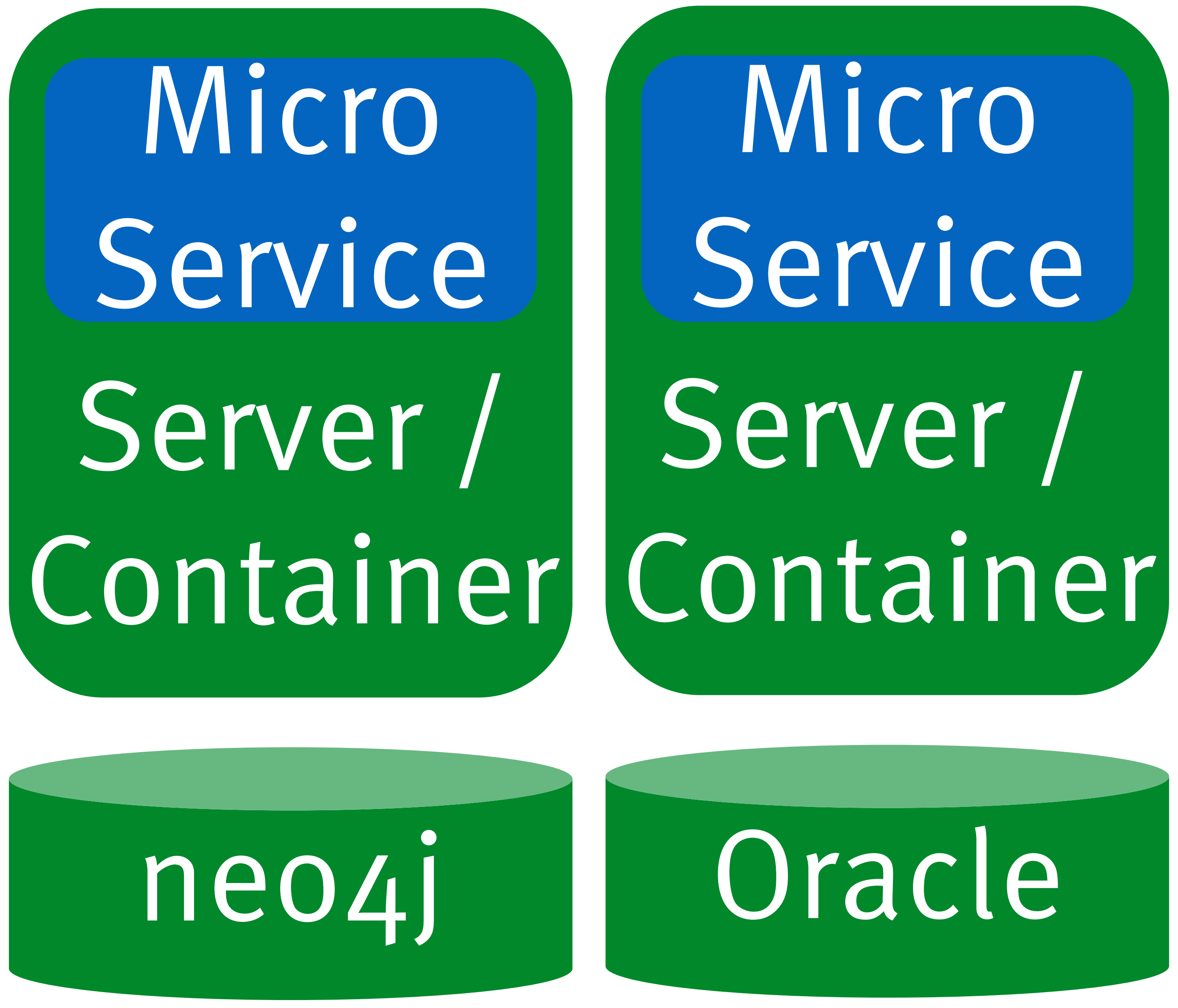
Why You Should Avoid a Canonical Data Model (Stefan Tilkov)

https://www.innoq.com/de/blog/thoughts-on-a-canonical-data-model/

Separate Databases



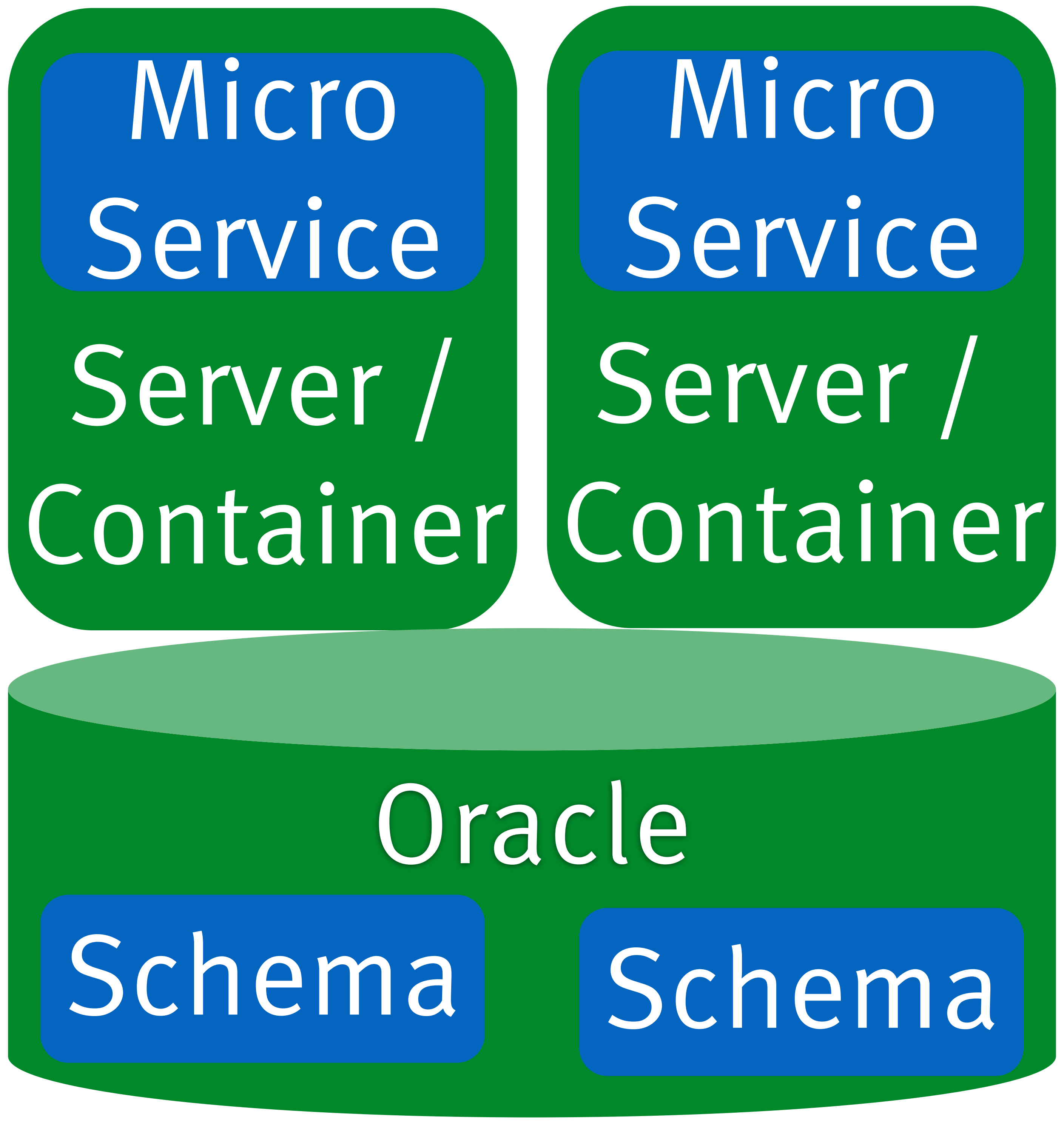
Different Databases



Different Databases

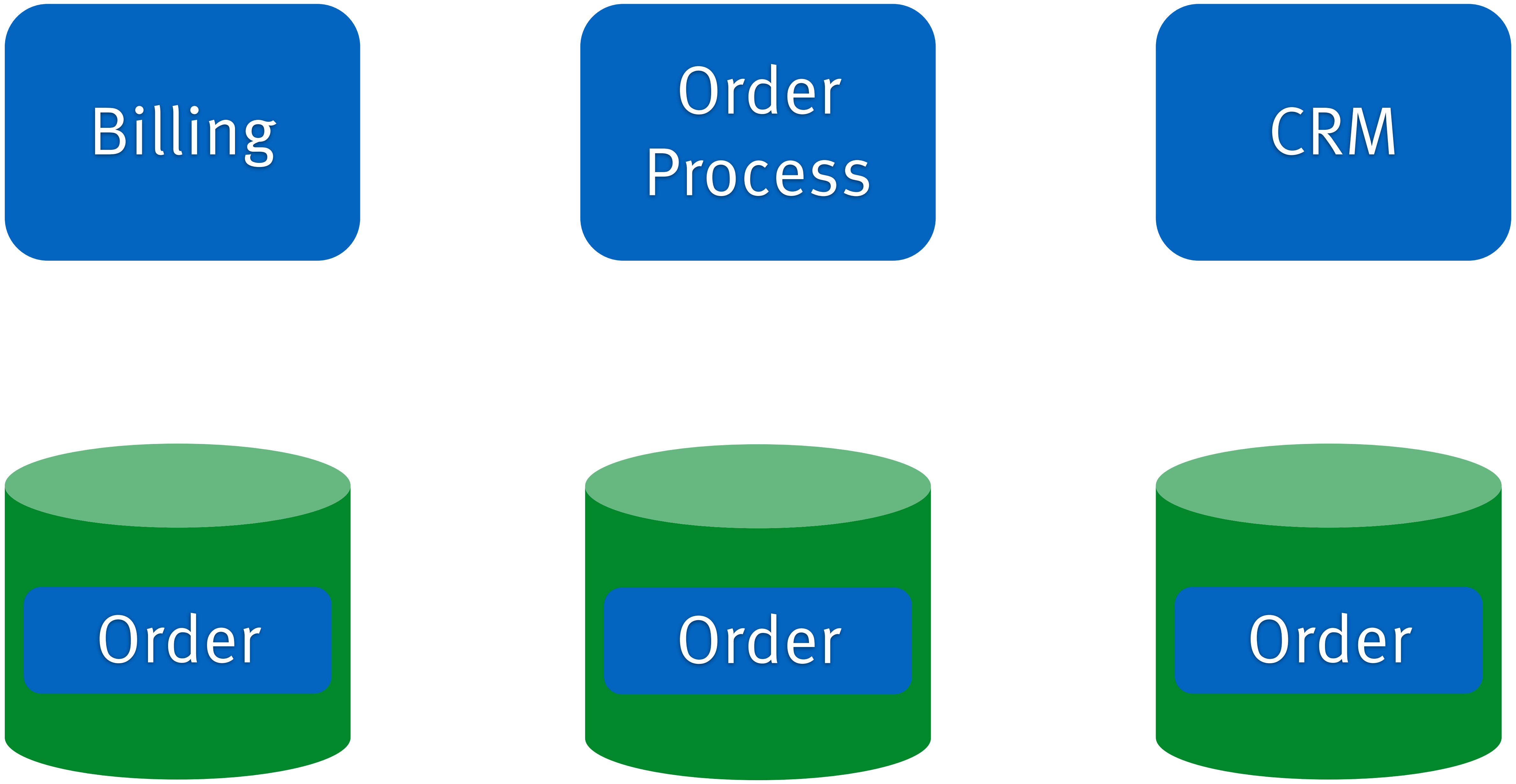
- “Polyglot persistence”
- Use the best tool for the job
- Technology freedom
  – advantage of microservices
- …but extra effort
- Backup, disaster recovery etc.
- Not as easy as e.g. different frameworks

Separate Schema



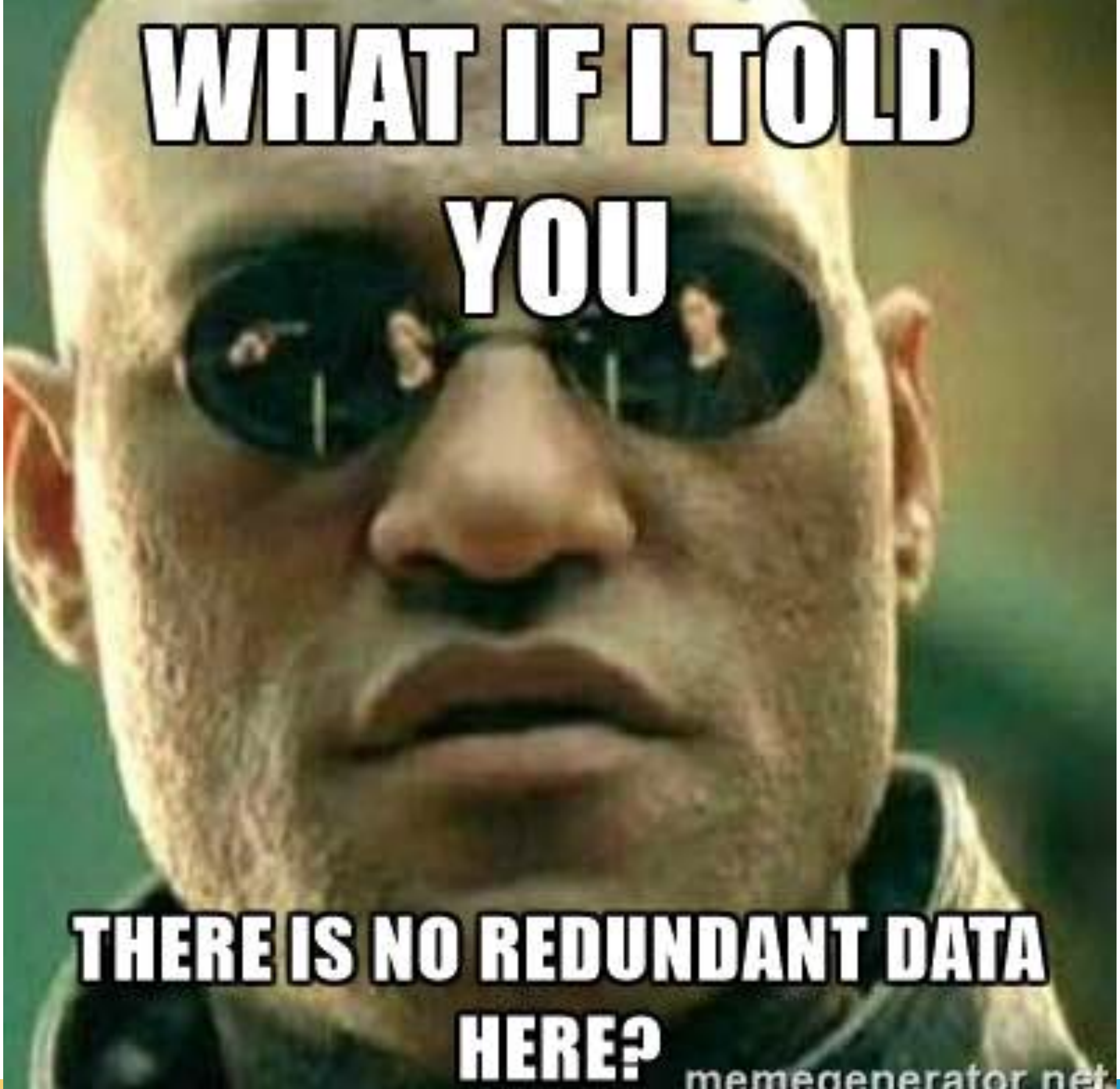
Separate Schemas

- Less effort
- Decoupled data models
- ...but limited independent scaling and robustness


Billing
Order Process
CRM
Order
Order
Order

Redundancy!!!

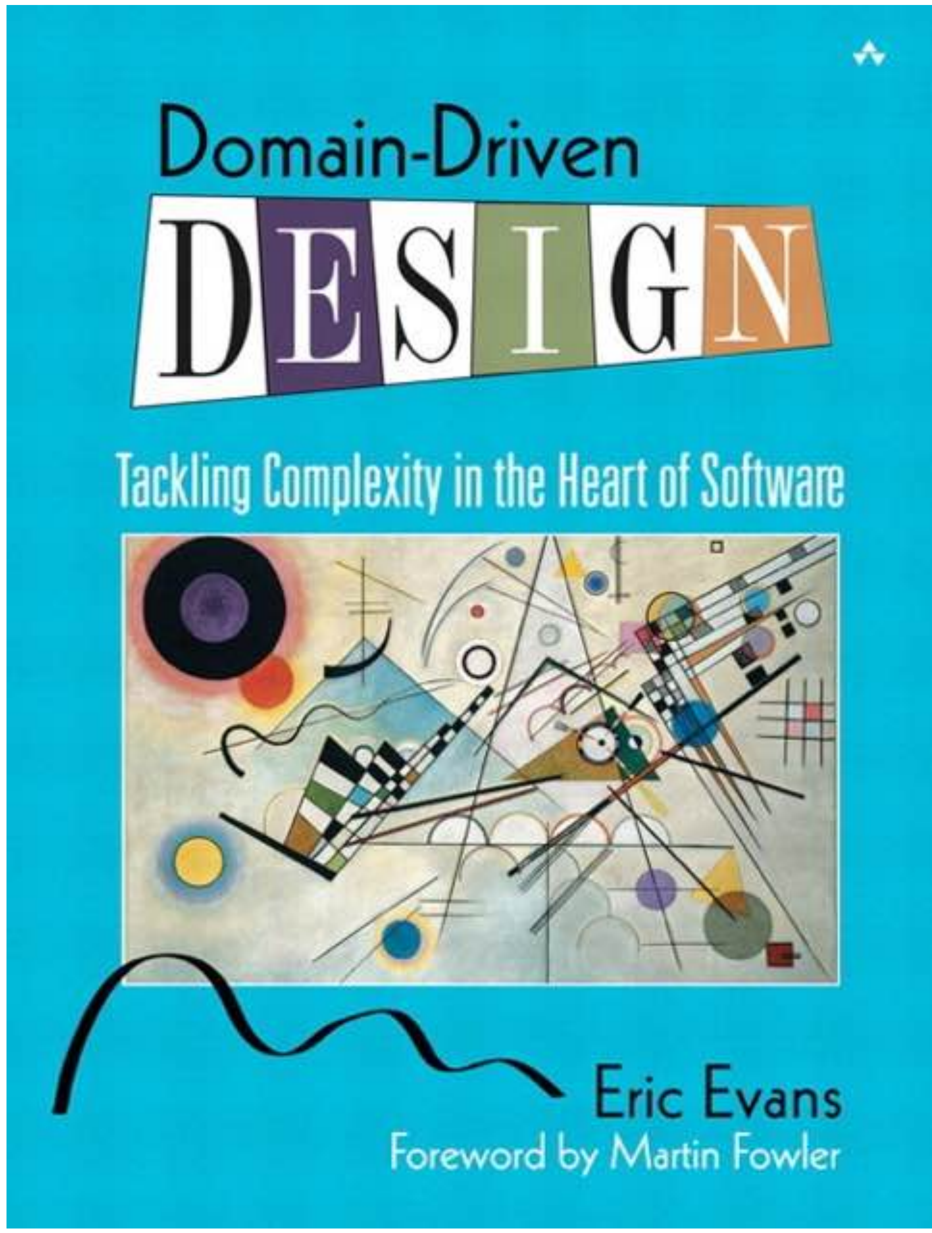
THE END IS NEAR
memegenerator.net


WHAT IF I TOLD YOU
THERE IS NO REDUNDANT DATA HERE?
memegenerator.net

Domain-driven Design

Domain-driven Design

- 2004
- Still very relevant
- By Eric Evans
- Focus on part IV
- Free reference: http://domainlanguage.com/ddd/reference/


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Order

Order #

Shipping address

Tracking #

Items

Item Categories

Priority shipping

Customs #

Account #

Credit card #

…

My Domain Model
is a mess!

Bounded Context

- Domain model is only valid for one context

- There is no universal data model!
- See all failed SOA attempts

Tracking

Order

Shipping address
Tracking #
Priority shipping

Payment

Order

Account #
Credit card #

Order

Order #
Shipping address
Tracking #
Items
Item Categories
Priority shipping
Customs #
Account #
Credit card #
…

Recommen-dations

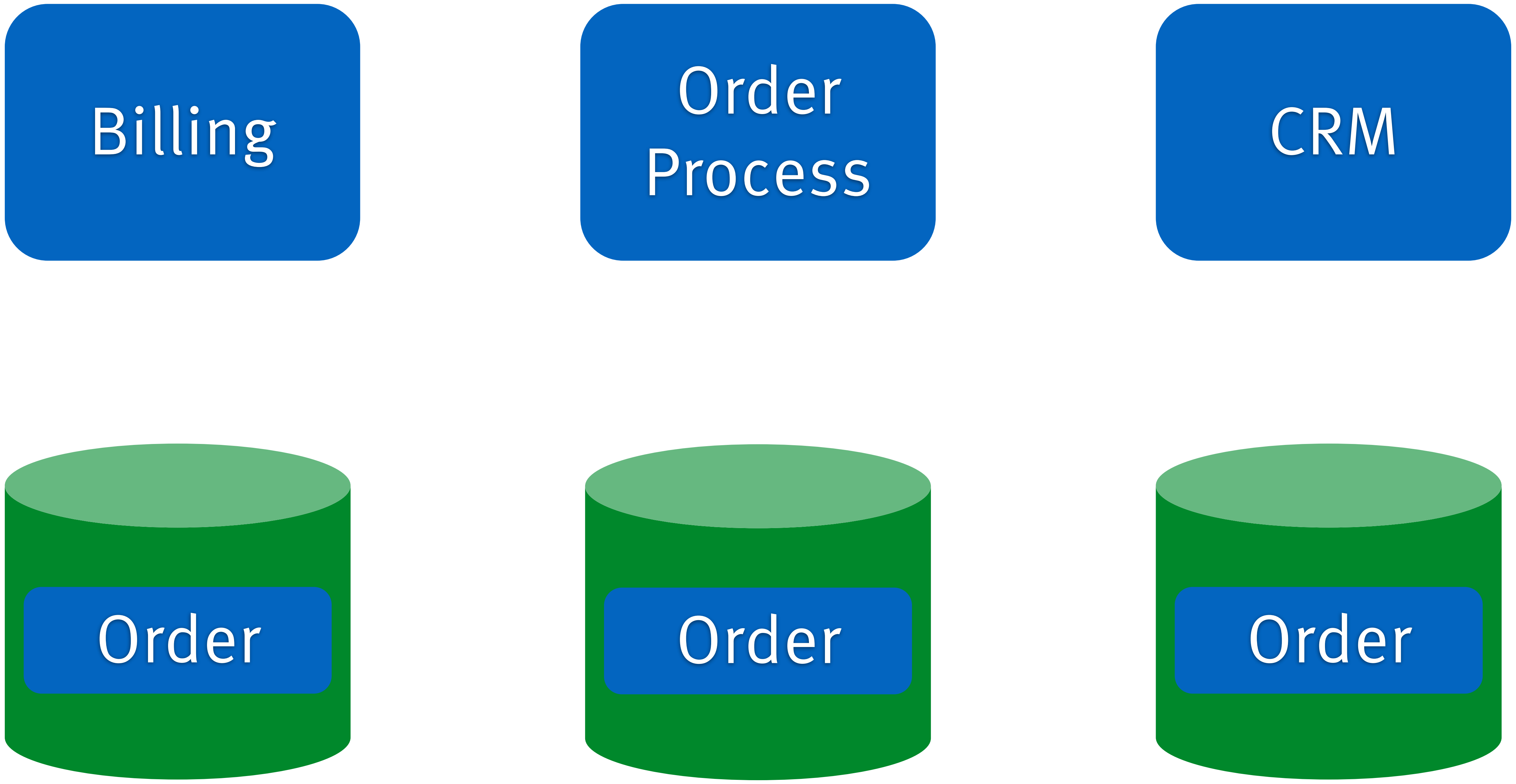
Order

Item Categories

Customs

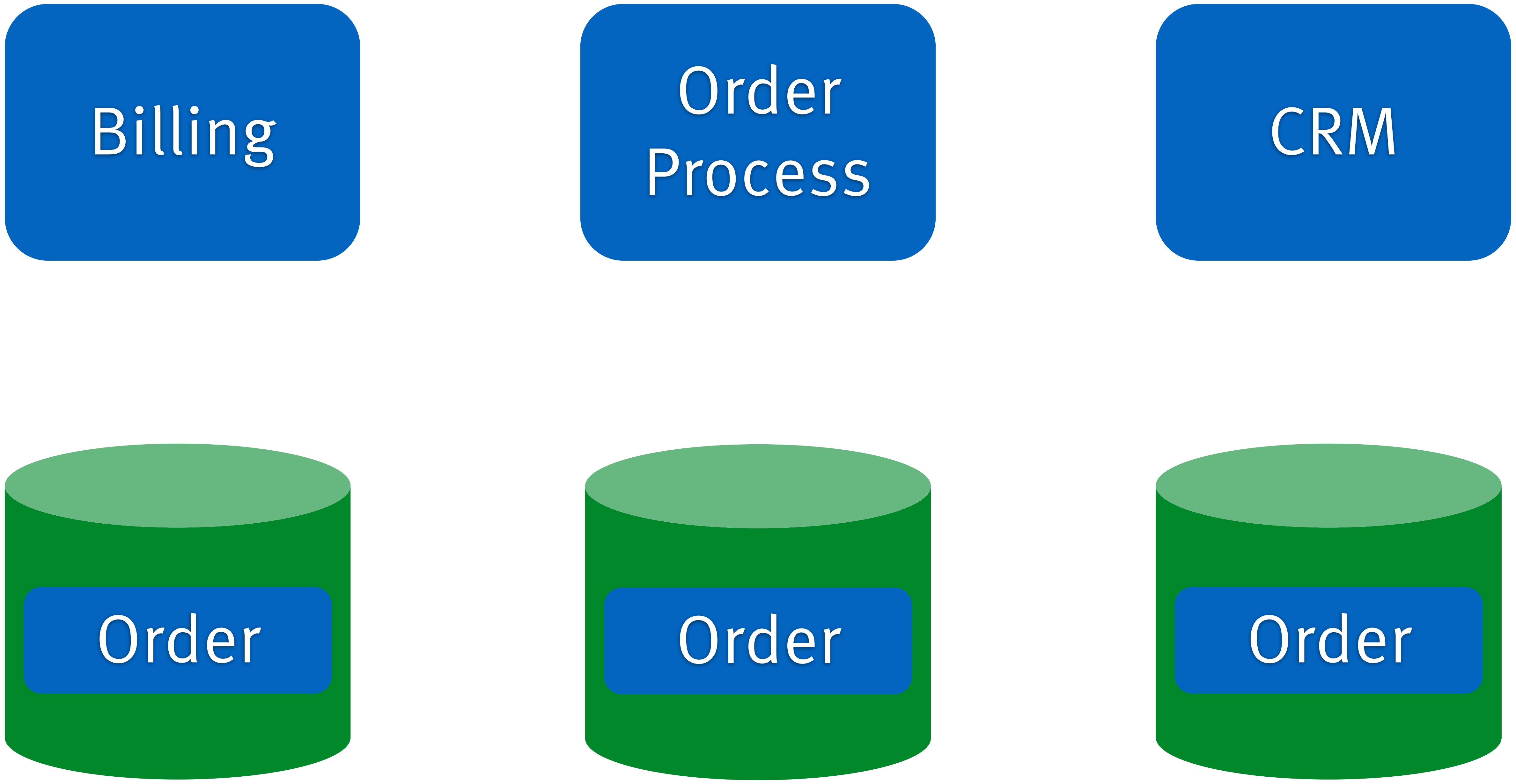
Order

Customs #


Billing
Order Process
CRM
Order
Order
Order

Bounded Context

- Microservice =
  Bounded Contexts
- Changes for new features are local
- …even if data models need to be changed


Billing
Order Process
CRM
Order
Order
Order

Redundancy?

Redundancy?
Not really

Bounded Context

What about basic data of an order?

Strategic Design

- How do BOUNDED CONTEXTS relate to each other?
- Context can have relationships
- DDD defines several relationship patterns

Shared Kernel

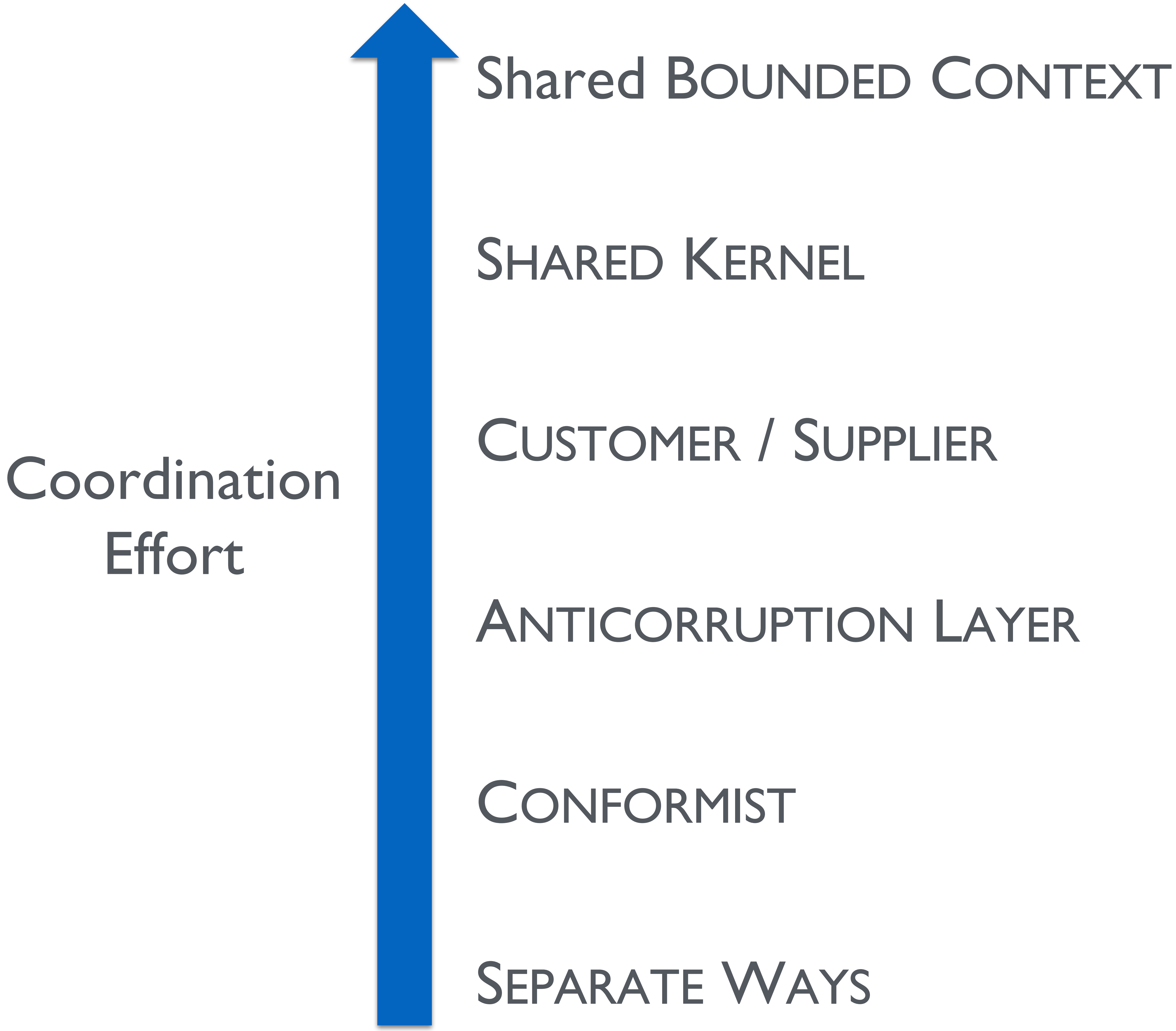
- Subset of a model
- …that two teams share
- Eric Evans: Including code and database
- Microservices: Just sharing a model

Anti-corruption Layer

- Don’t let e.g. a legacy model influence a new model
- Isolate model by additional layer
- No need to modify the old system

Context Relationships

- Team = Deployment Unit = BOUNDED CONTEXT
- Context Relationships define how BOUNDED CONTEXT are used…
- …and how much teams need to collaborate

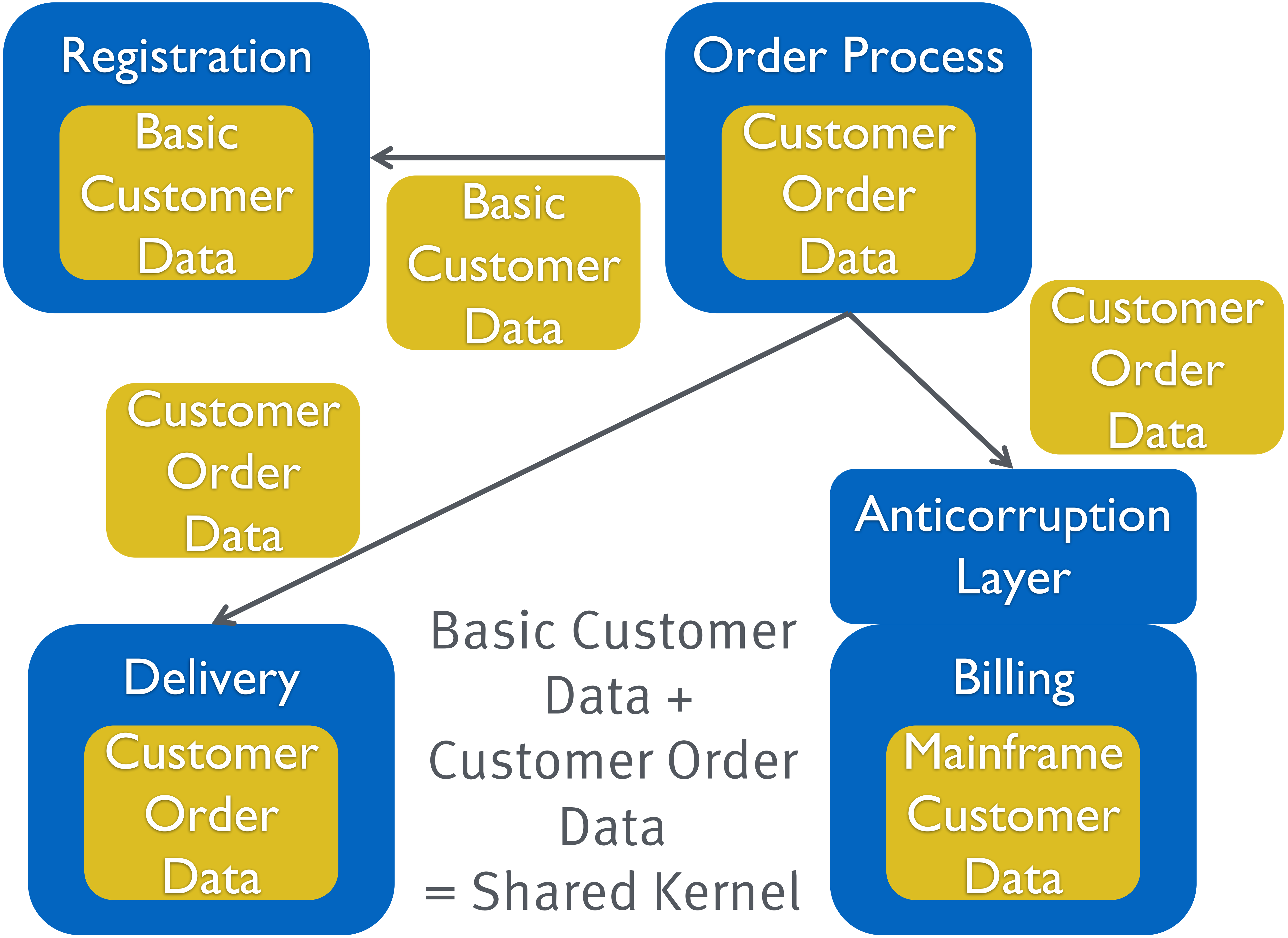

Shared Bounded Context
Shared Kernel
Customer / Supplier
Anticorruption Layer
Conformist
Separate Ways
Coordination Effort

Context Map

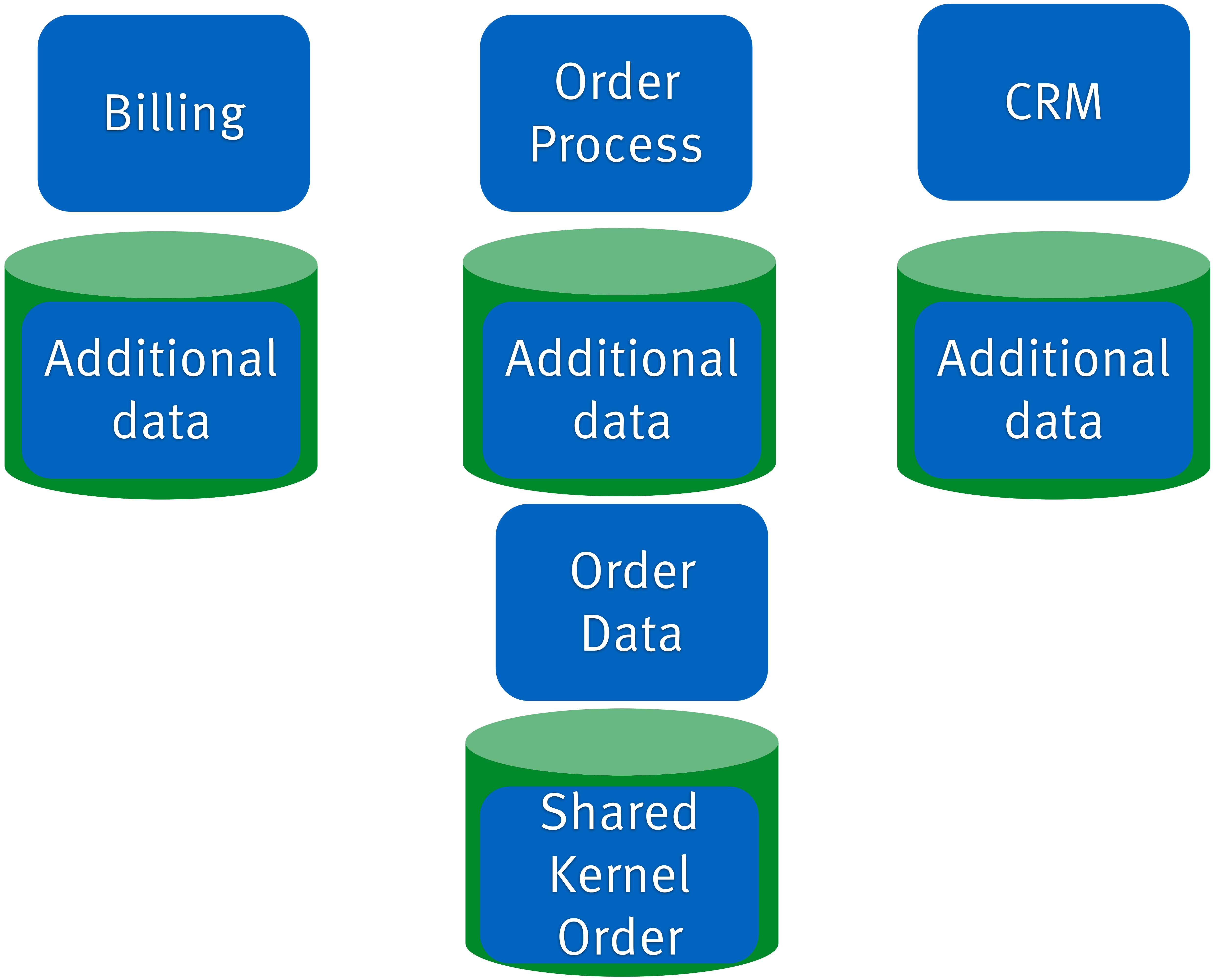
Context Map

- Show the different Bounded Context
- ...and the relation to each other

- Bounded Context might be microservices
- ...or communication links

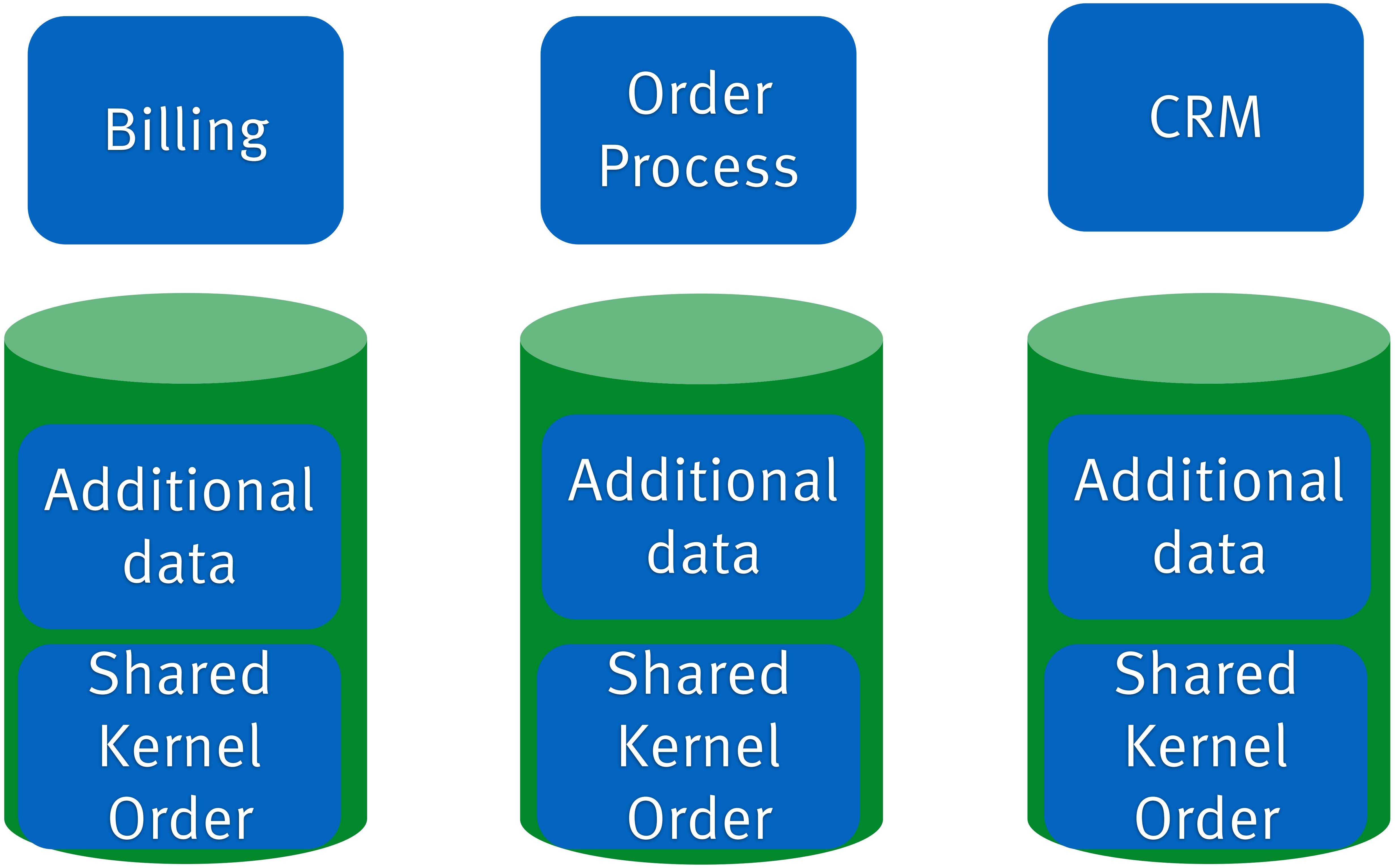


Basic Customer Data + Customer Order Data = Shared Kernel


Billing
Order Process
CRM
Additional data
Additional data
Additional data
Order Data
Shared Kernel Order

Centralized Shared Kernel

- Ensures consistency
- …but needs to be called for a lot of operations
- Resilience / performance / transactions
- Have one master as the source of truth


Billing
Order Process
CRM
Additional data
Shared Kernel Order
Additional data
Shared Kernel Order
Additional data
Shared Kernel Order

Decentralized Shared Kernel

- Might be inconsistent
- ...but all data for all requests is available in the local database

- Better resilience...
- ...and performance

How to Replicate Data?

Database Replication

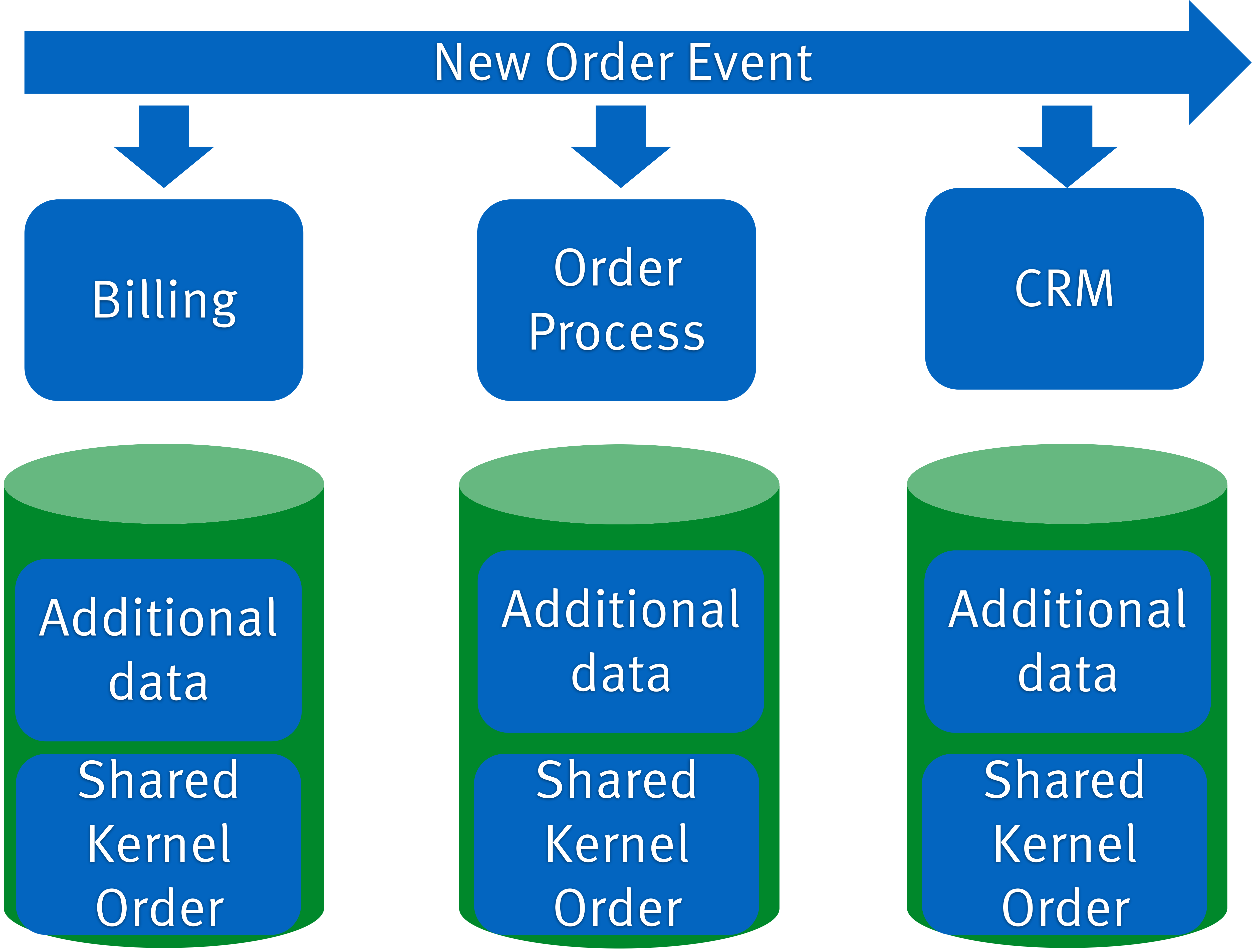
- Built into the database
- Replicate schema across database instances

- But: Microservices have separated schemas
- Every Microservice might have different data
- ...so database replication is not a good fit

Replication with Events

Events

- Later addition to Domain-driven Design
- Events with a business meaning
- Decouple time:
  Asynchronous
- Decouple logic:
  System can handle event as it pleases


New Order Event
Billing
Order Process
CRM
Additional data
Shared Kernel Order
Additional data
Shared Kernel Order
Additional data
Shared Kernel Order

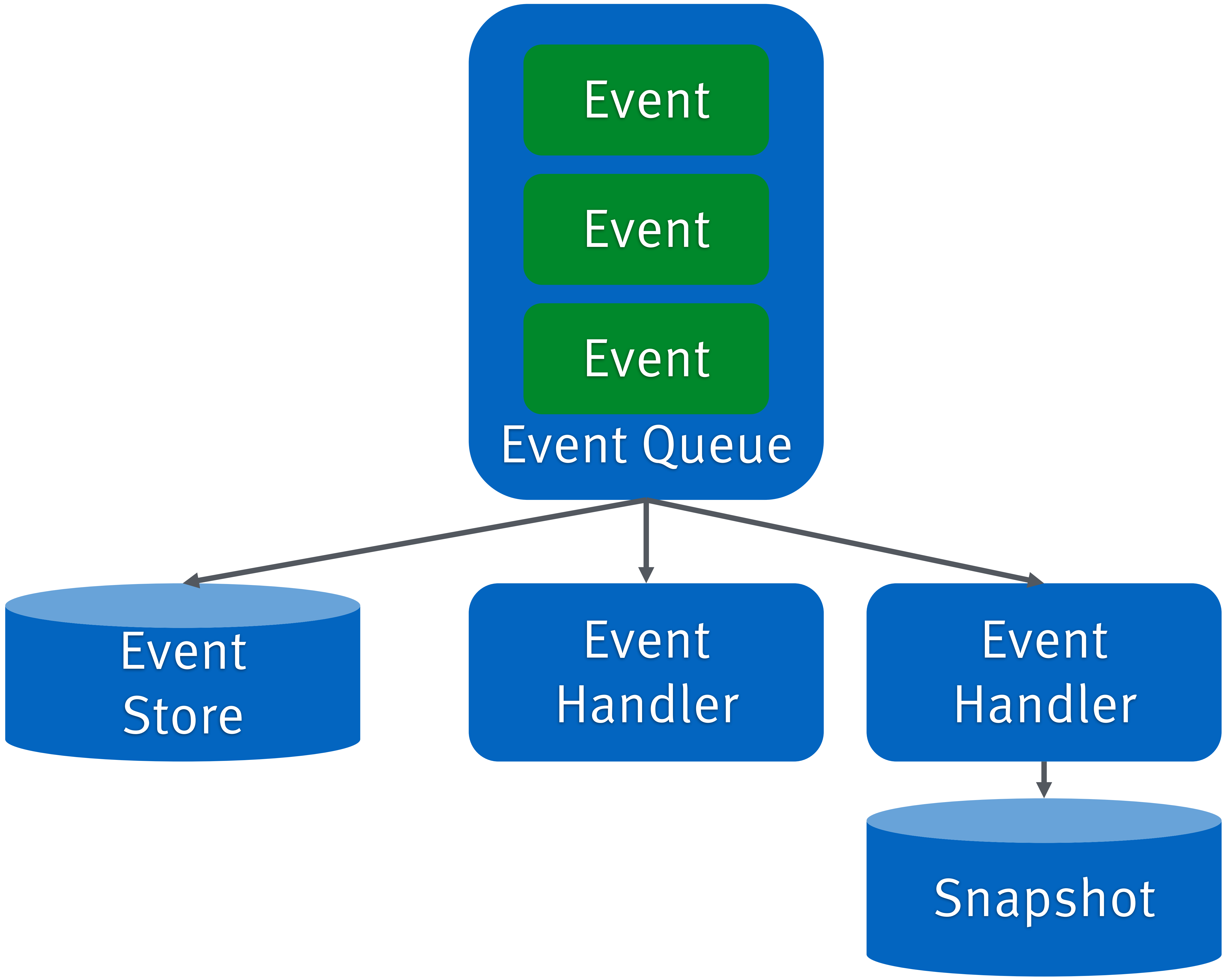
Events & Data Replication

- Events lead to data replication
- i.e. each system stores information it received in an event
- Data stored in separate schema
- Very decoupled
- Hard to repair inconsistencies

More Fun With Events

Event Sourcing

- Internal Structure for Microservice with events
- Current state result of all events
- Calculate state on the fly?


Event
Event
Event
Event Queue
Event Store
Event Handler
Event Handler
Snapshot

Event Sourcing

- Event store and snapshot help to repair inconsistencies
- Event-based architecture in microservices

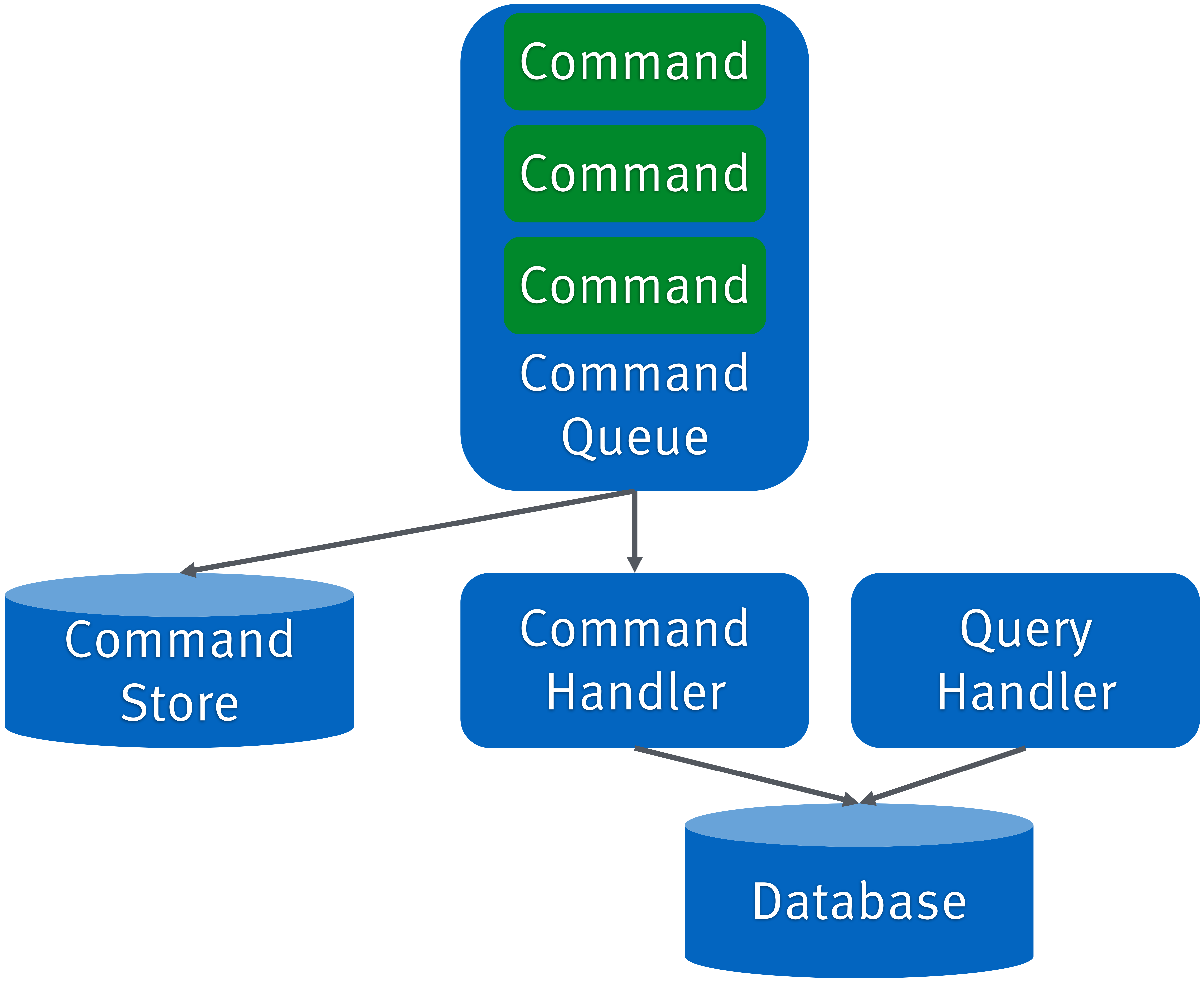
CQRS

- Command – Query Responsibility Segregation
- Commands change data
- Query provide data
- Implement in separate modules
- …or even microservices
- …with potentially different BOUNDED CONTEXTS

Commands vs Events

- Command: Change that data!

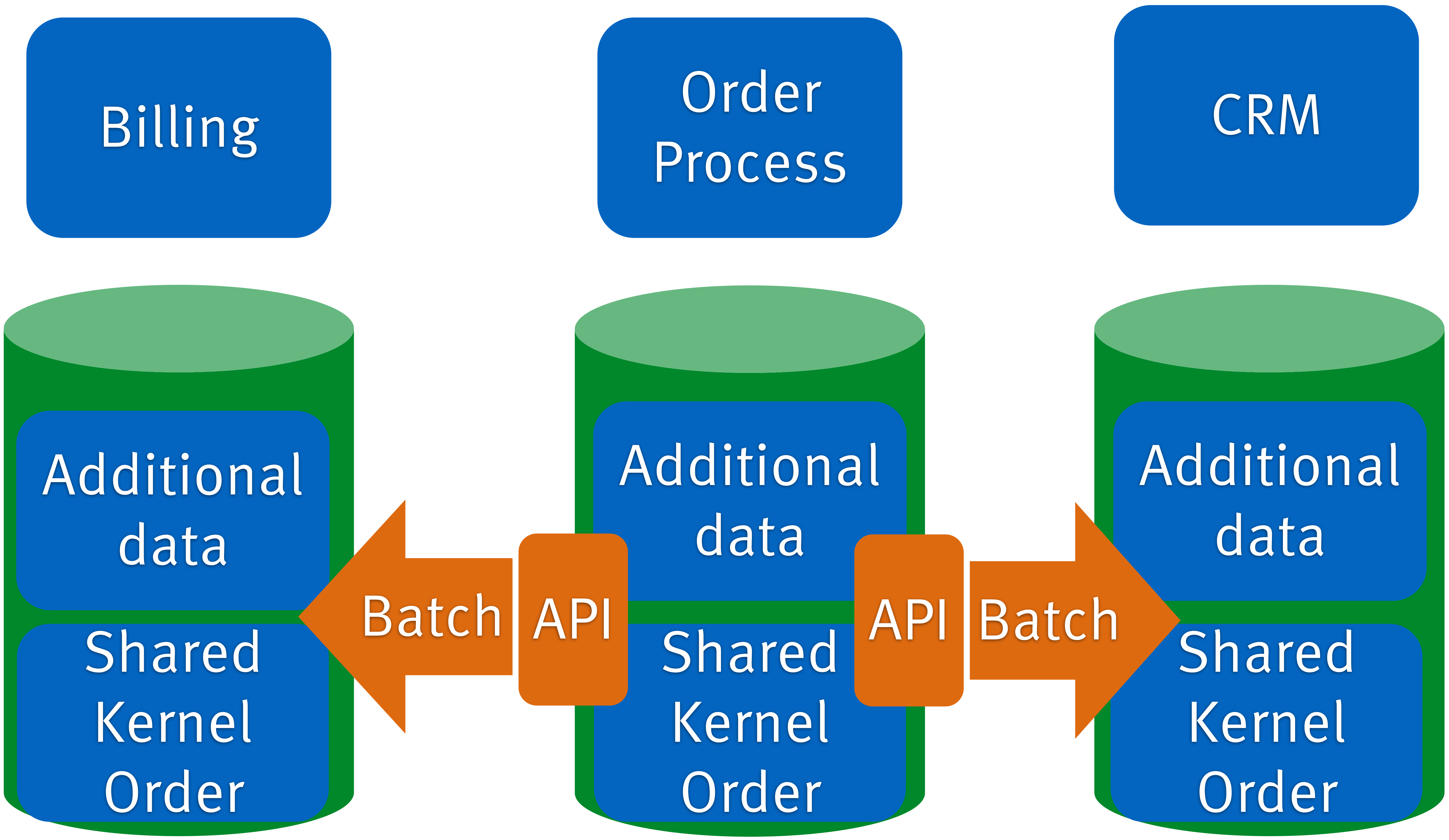
- Event: Something has happened
- Component decides if data should be changed


Command
Command
Command
Command Queue
Command Store
Command Handler
Query Handler
Database

Batch Replication

Batch

- Get all data
- Provide API
- ...to decouple schema
- Copy interesting data into local database


Billing
Order Process
CRM
Additional data
Shared Kernel Order
Batch
API
Additional data
Shared Kernel Order
API
Batch
Additional data
Shared Kernel Order

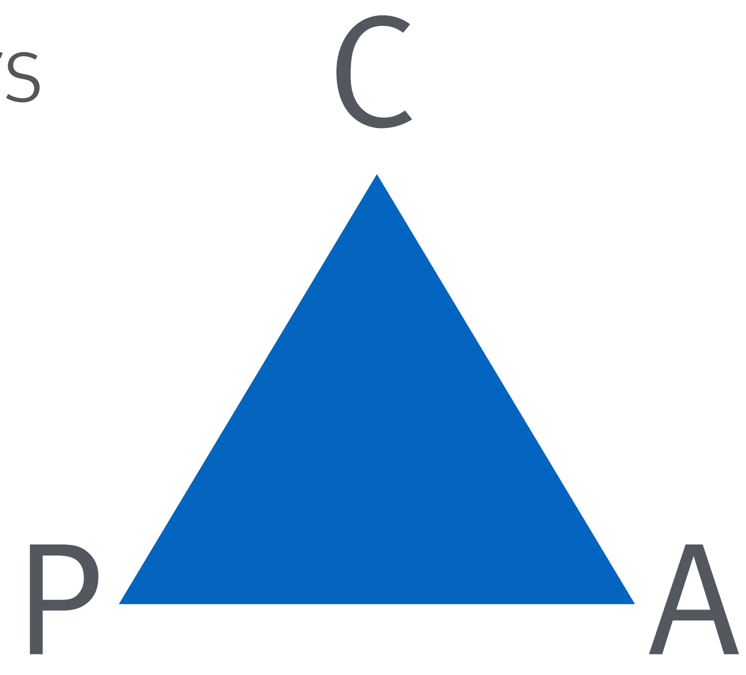
Batch & Data Replication

- Easy to repair inconsistencies
- Batch run at specific points
- i.e. updates take time
- Data not consistent across microservices

CAP: Challenge for Replication

CAP Theorem

- Consistency
  - All nodes see the same data
- Availability
  - Node failures do not prevent survivors from operating
- Partition Tolerance
  - System continues to operate despite arbitrary message loss



CAP Theorem: P

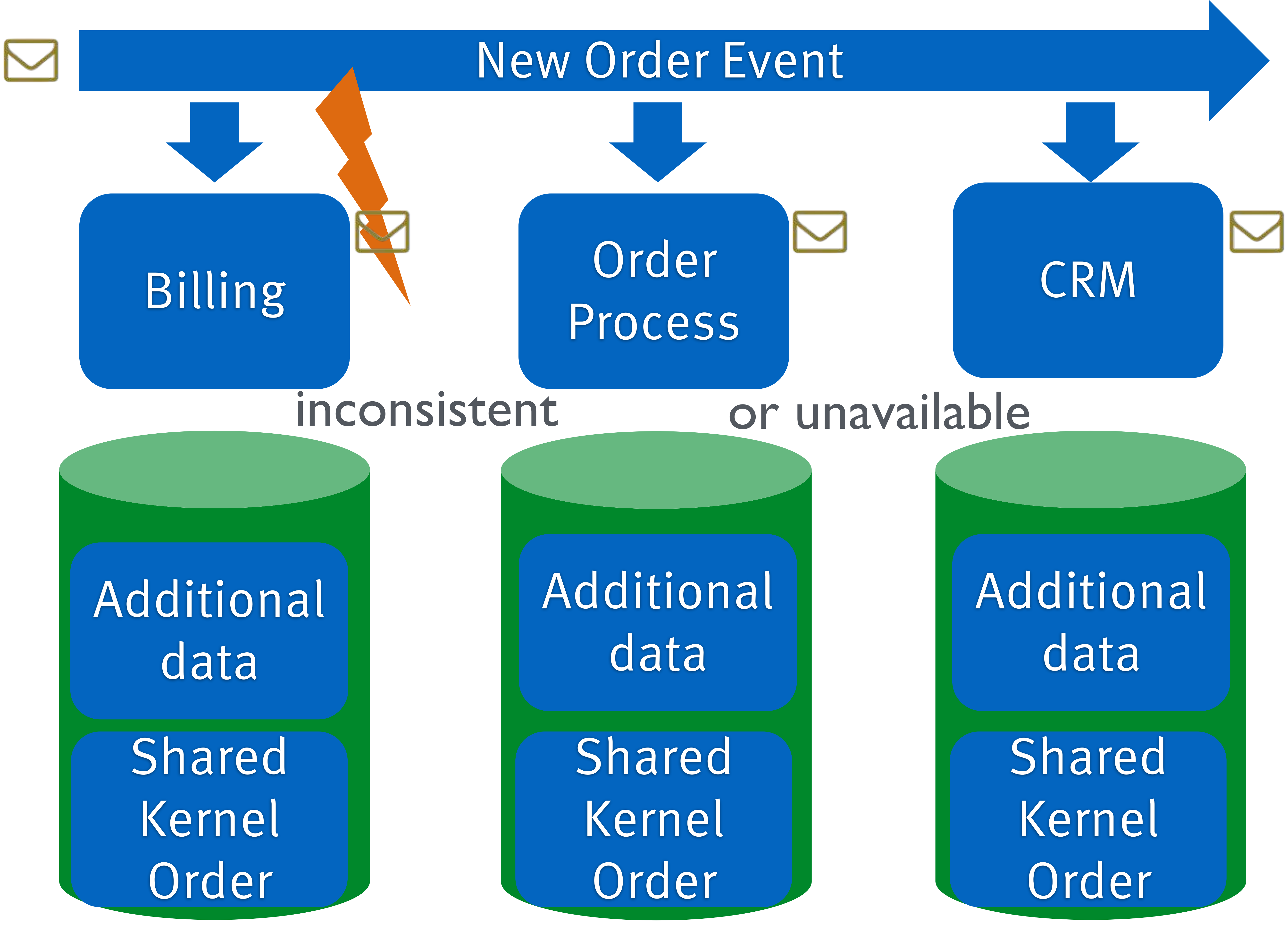
- Network partitions do occur
- Even with highly available network hardware
- Also: very slow response = partition

- Need to deal with P

CAP Theorem: C or A?

- Node cannot access other nodes
- Might have missed updates

- A, not C:
  Answer with a potentially wrong answer
- C, not A:
  Don't answer – the answer might be wrong


New Order Event
Billing
Order Process
CRM
inconsistent
or unavailable
Additional data
Shared Kernel Order
Additional data
Shared Kernel Order
Additional data
Shared Kernel Order

Conclusion

Classic: Centralized Database

Microservices: private database decoupling

Data Microservices: Consistent but resilience / performance / transactions / decoupling?

Schema per Microservice: Simple infrastructure

Database per Microservice: Polyglot Persistence

Redundancy?

Redundant Data or Bounded Context?

Context Map and Context Relations

Replication

e.g. Shared Kernel

Database Replication

Batch

CQRS

Event Sourcing

Events

CAP

Decentralize data!

EMail slideswjax2016@ewolff.com to get:

Slides

+ Microservices Primer

+ Sample Microservices Book

+ Sample of Continuous Delivery Book

Powered by Amazon Lambda & Microservices