Domain-driven Design: A Complete Example

Eberhard Wolff
Head of Architecture
https://swaglab.rocks/
https://ewolff.com/

SWAGLab

Why This Talk?

- Domain-driven Design provides lots of tools.
- Which are really useful?
- How can you combine them?

Event Storming

Why Event Storming?

How does domain knowledge become software?

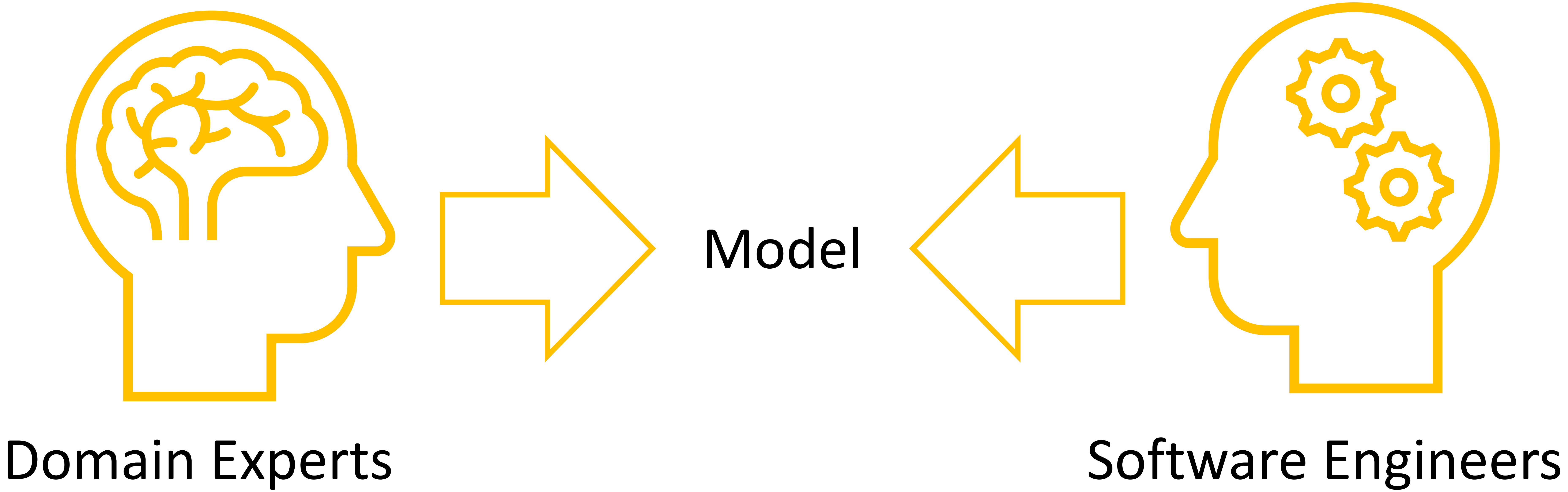
Domain Experts

Understand the domain

Software Engineers

Structure knowledge to become software

Why Event Storming?



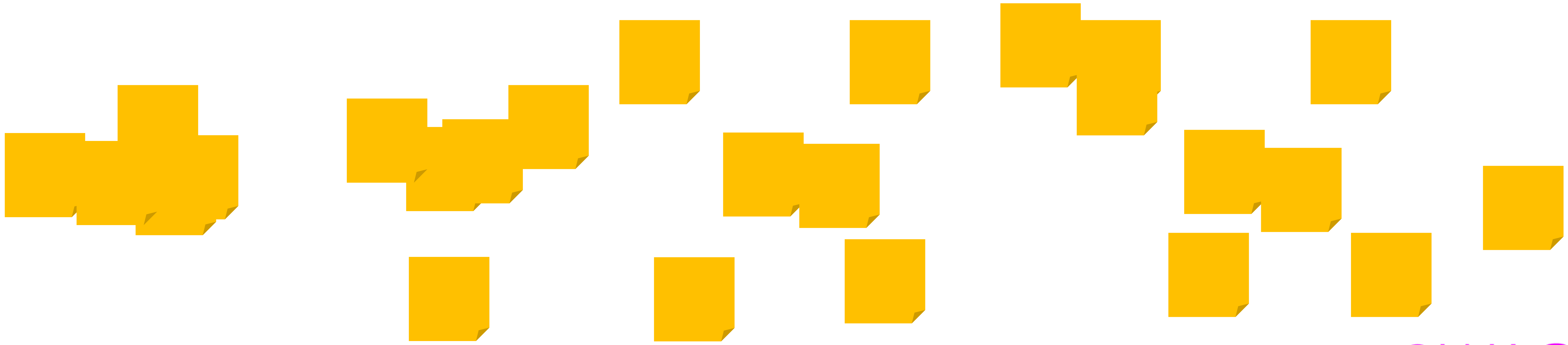
What is Event Storming?

- Event in the past
- At least noun + verb
- Verb in past tense
- Write event on orange sticky

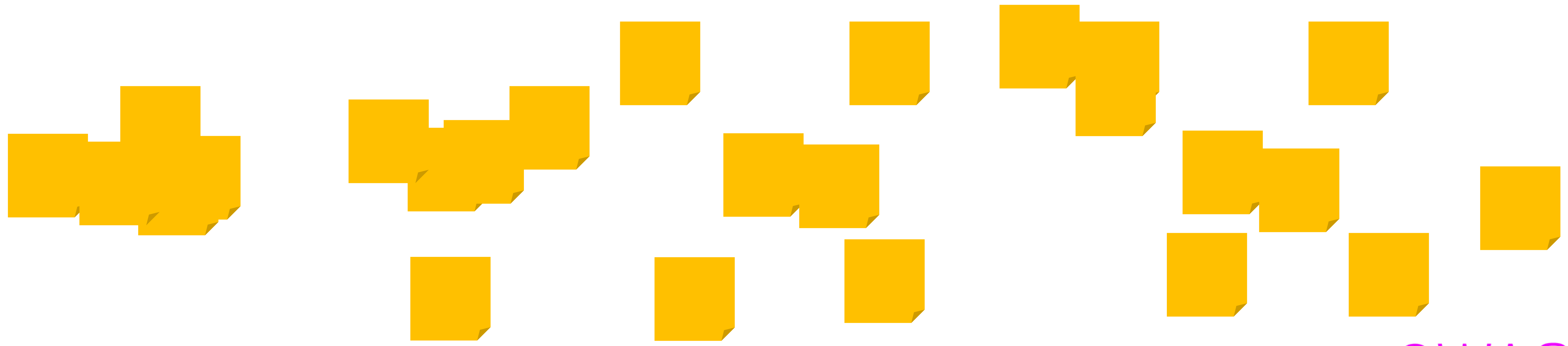
Order accepted

Phase: Chaotic Exploration

- Create as many events as possible

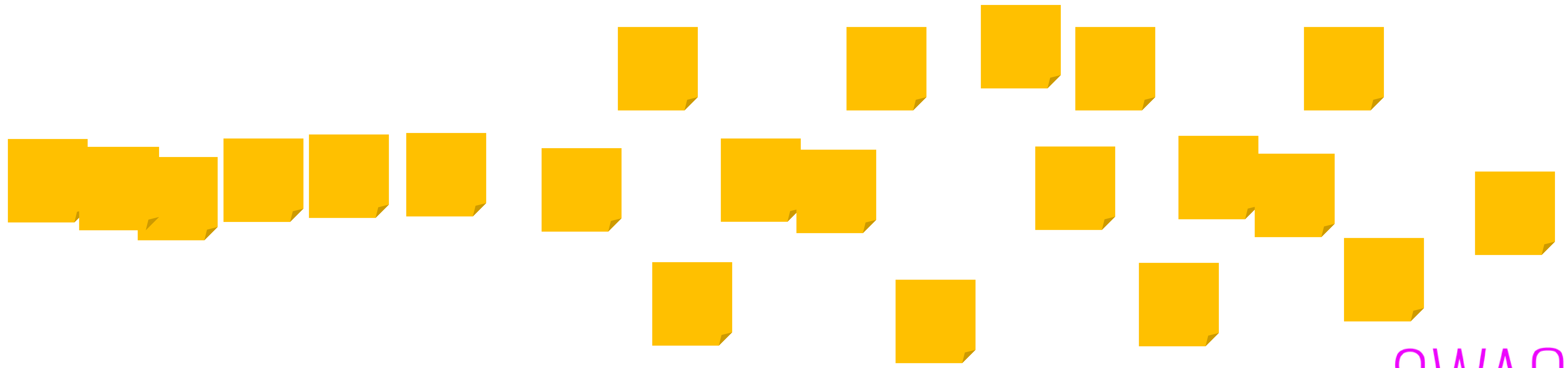


Phase: Enforce the timeline



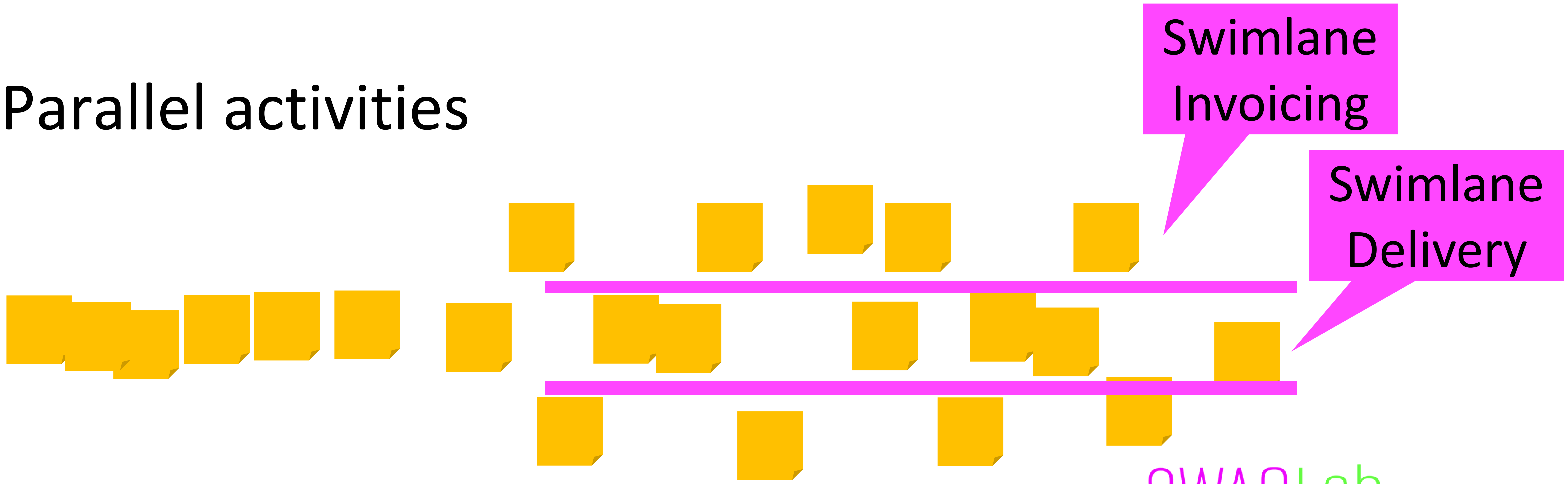
SWAGLab

Phase: Enforce the timeline



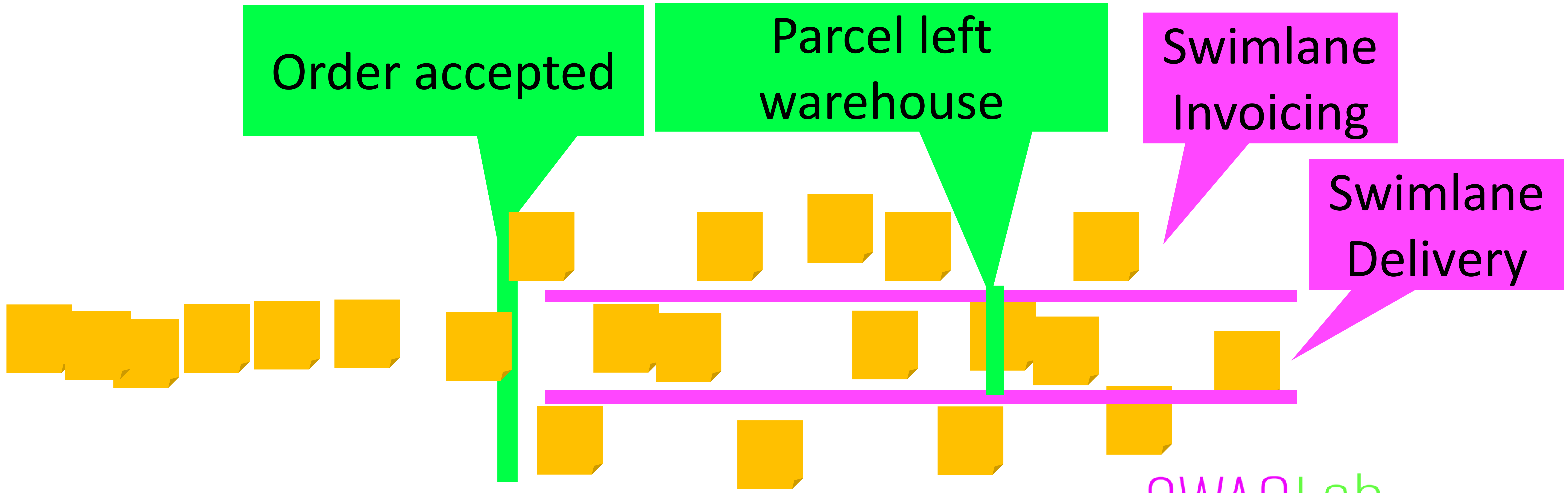
Phase: Identify Swim Lanes

- Parallel activities



Phase: Identify Pivotal Events

- Afterwards the world is different



Event Storming: Benefits

- Low-tech: easy to understand for domain experts
- People can work in parallel.
- Social structures become obvious.

Event Storming: Result

- Common understanding of the domain
- A model of the domain

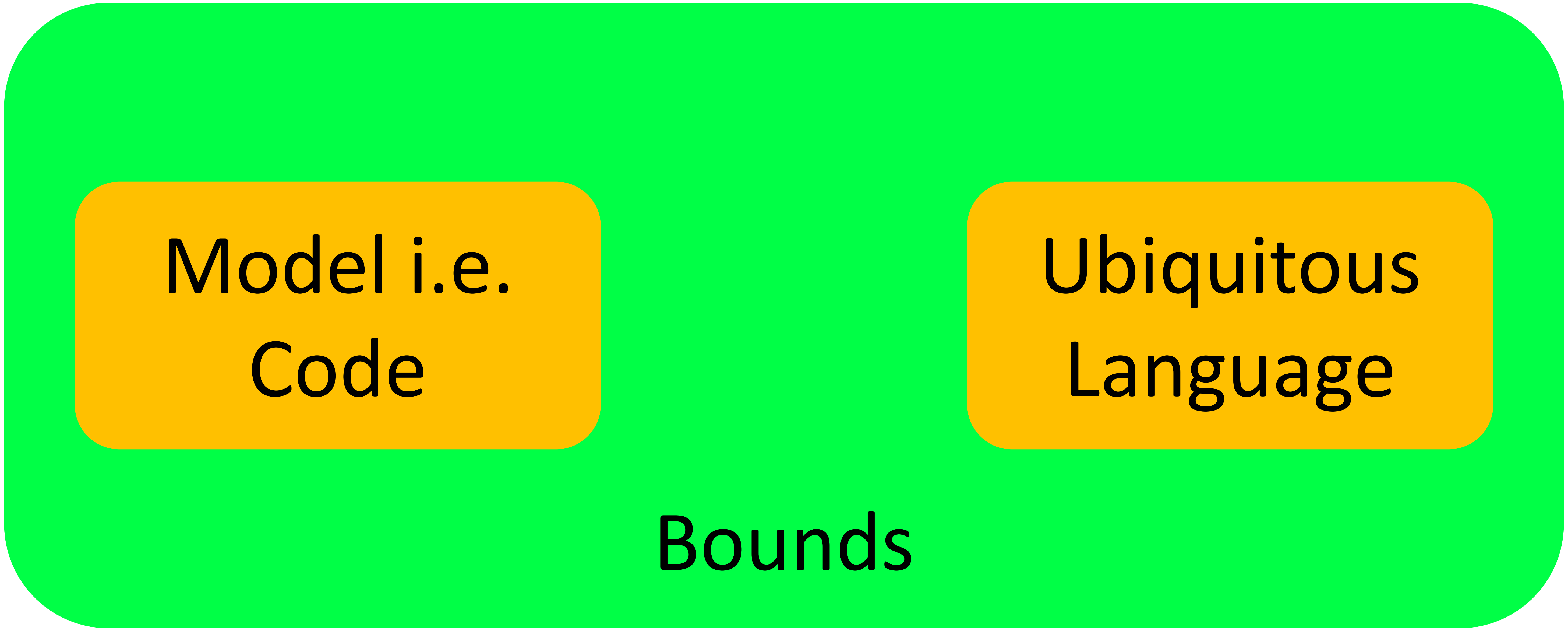
…that must be tweaked before it can be implemented

Bounded Context

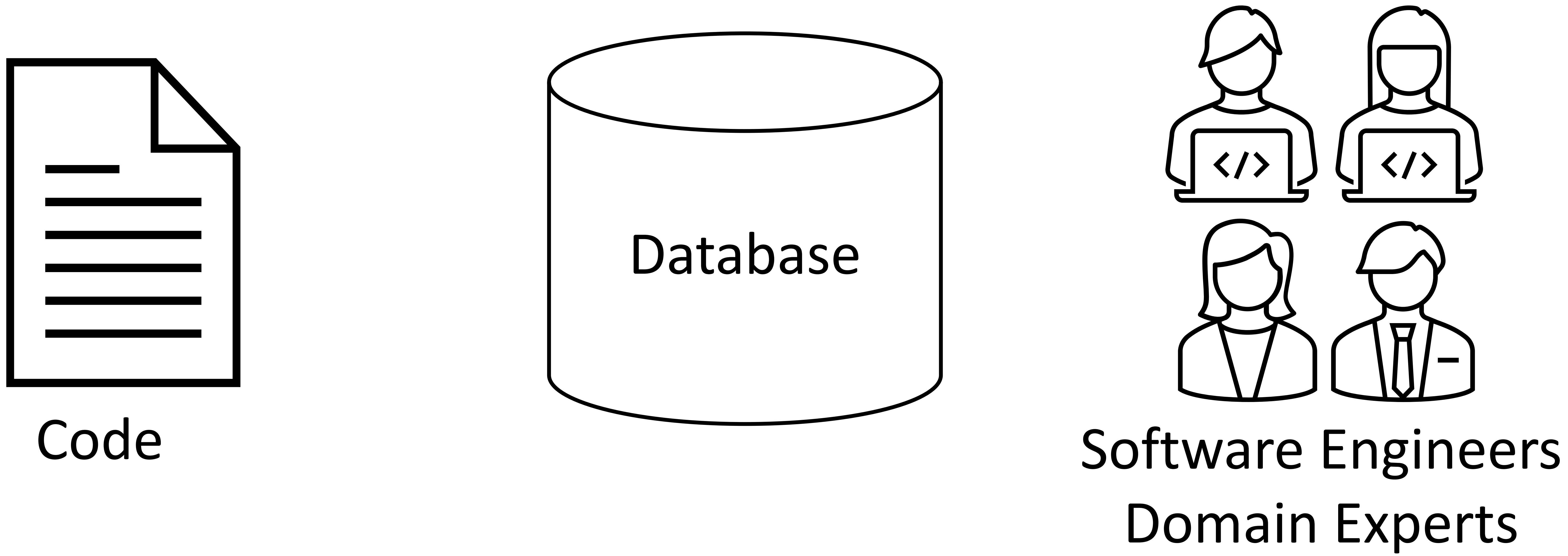
Why Bounded Context?

- Coarse-grained split of the domain
- A scope that might be assigned to a team

Bounded Context



Ubiquitous Language



Bounded Context Example

Invoicing
Process

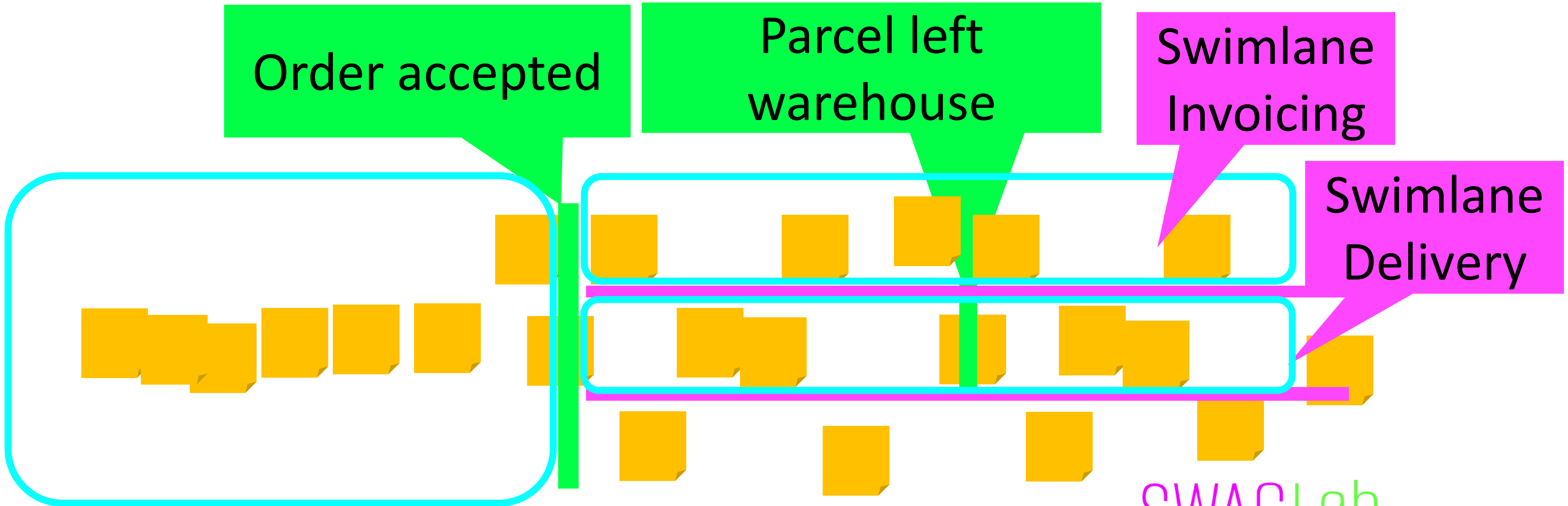
Customer who pays the bill

Delivery

Customer who the products are sent to

Identify Candidates for Bounded Contexts

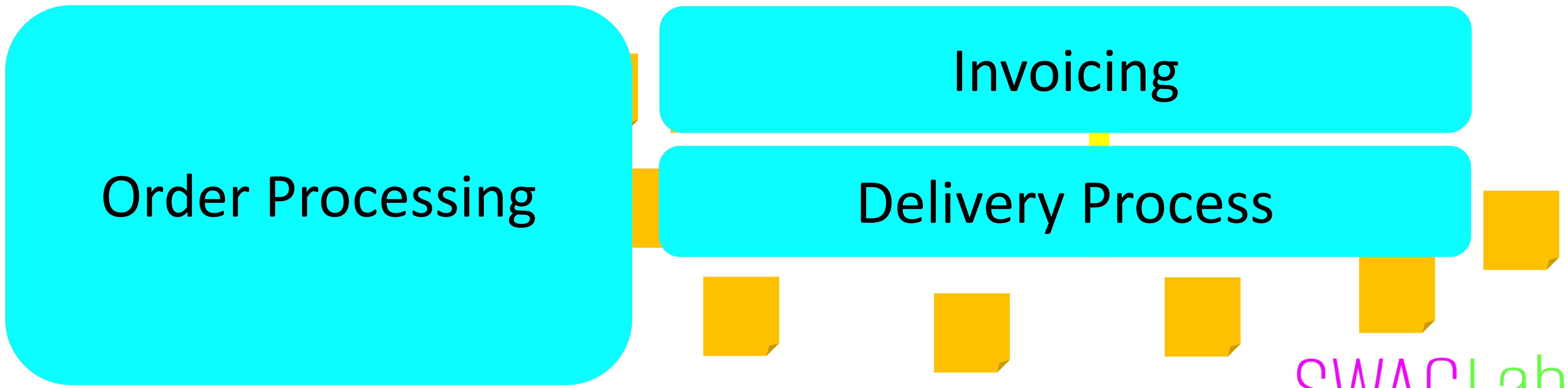
- Areas between swim lanes and pivotal events are good candidates for Bounded Contexts



SWAGLab

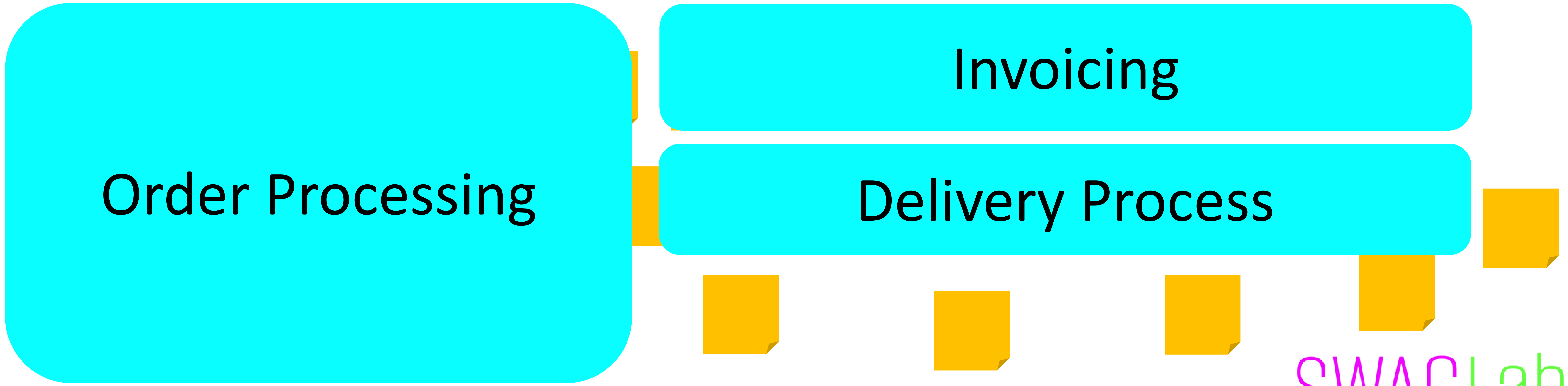
Identify Candidates for Bounded Contexts

- Areas between swim lanes and pivotal events are good candidates for Bounded Contexts



Identify Candidates for Bounded Contexts

- Reimbursements handled by invoicing?
- Return handled by delivery process?



SWAGLab

Bounded Context: Benefits

- Structures domain logic
- Request probably local to one bounded context
- Changes probably local to one bounded context
- i.e. a great architecture!

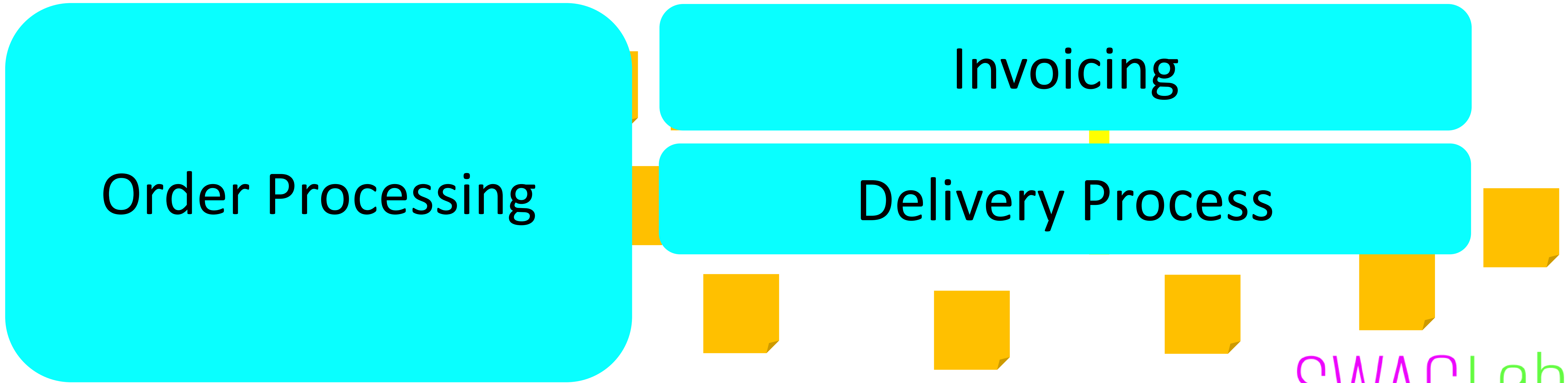
Core Domain

Core Domain

- Core domain = motivation for the project
- Reduce the complexity of the model
- Focus on maintainability of this part of the system

What is the Core Domain?

- Differentiation: quick and reliable delivery
- Core domain = delivery process



SWAGLab

Core Domain: Result

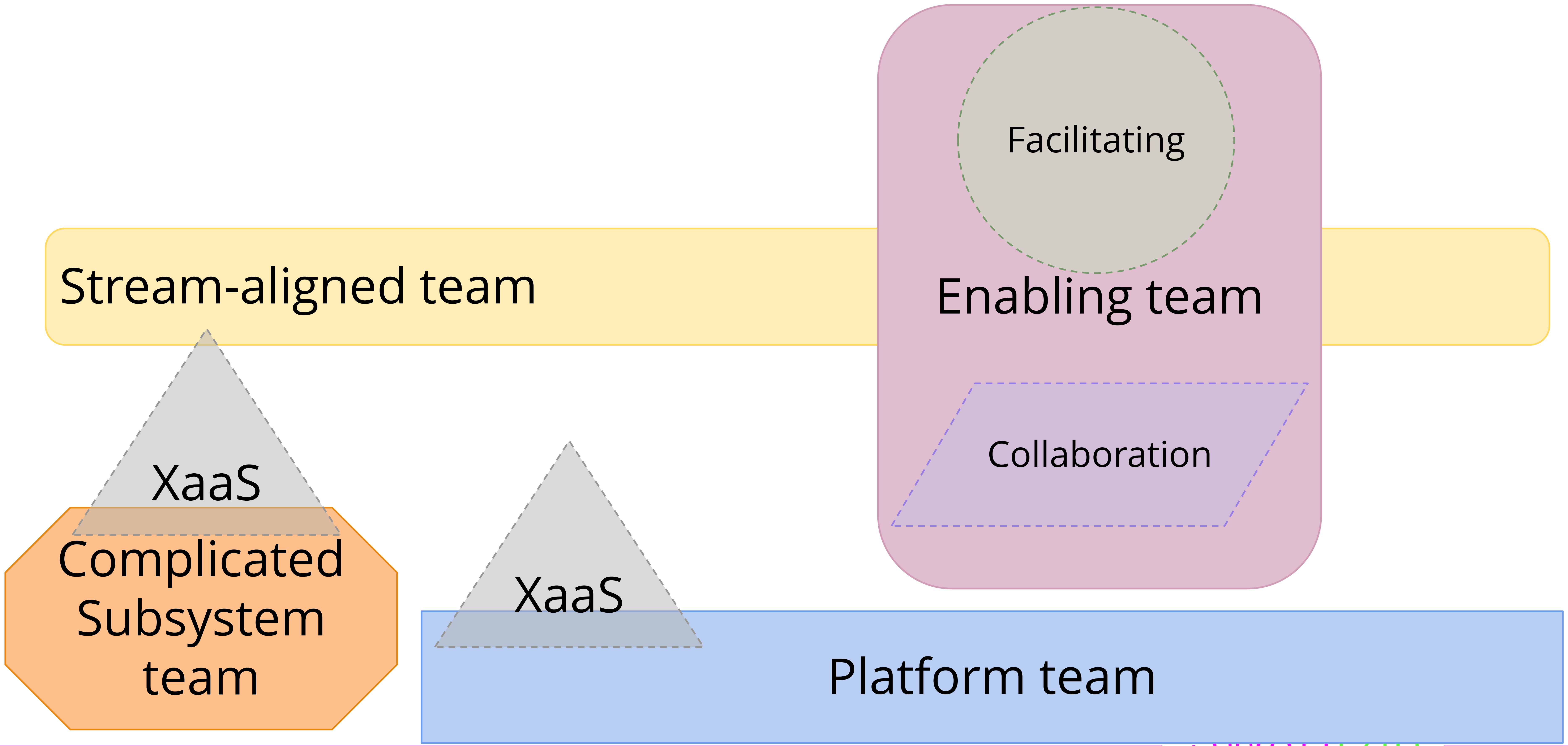
- Clear focus
- Better understanding of the domain

Not Strategic Design but Team Topologies

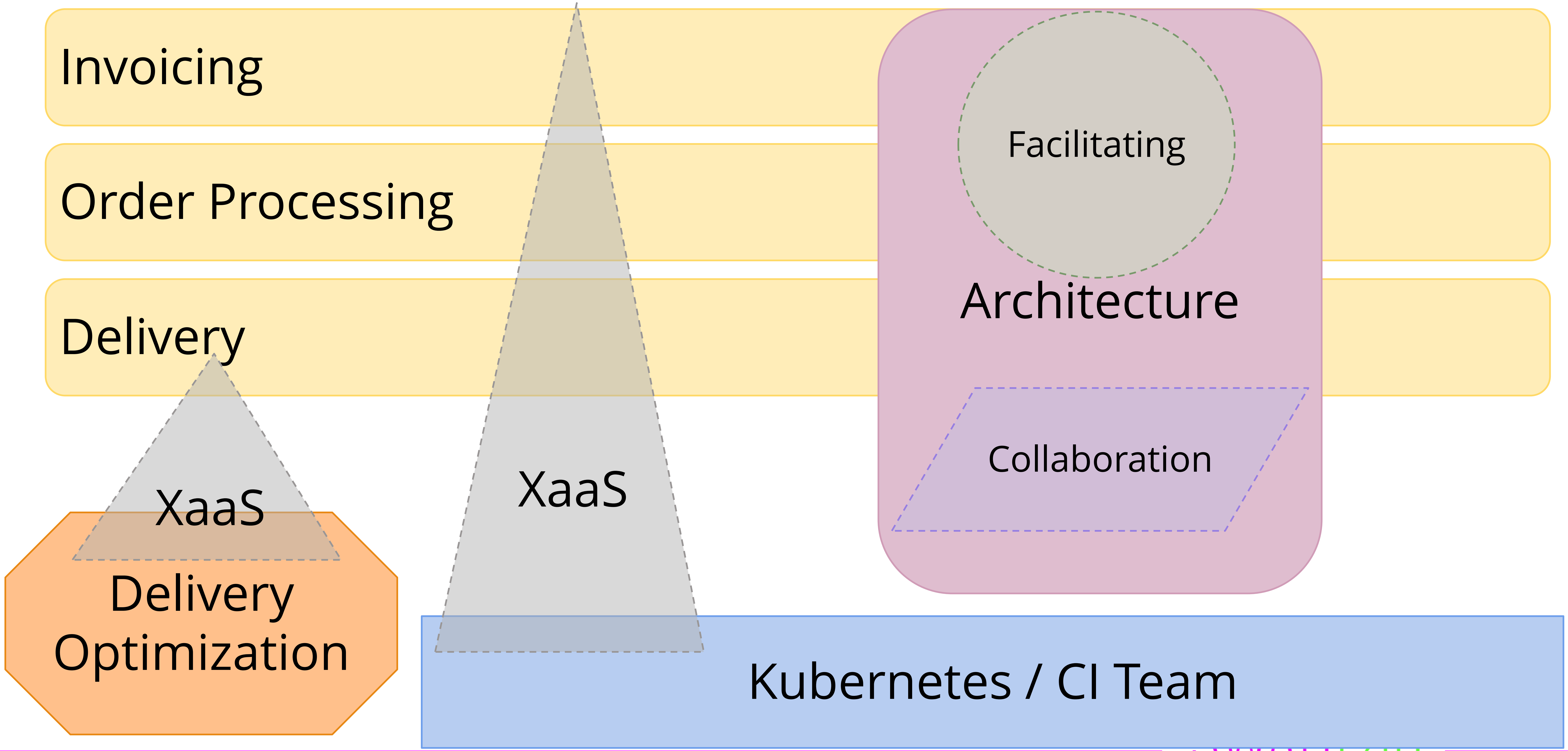
Why Team Topologies?

- Somehow teams need to collaborate
- Not too complex
- Intuitive (?)
- Covers more “fracture planes” then just Bounded Contexts e.g. location
- Covers more team types than development teams

Team Topologies

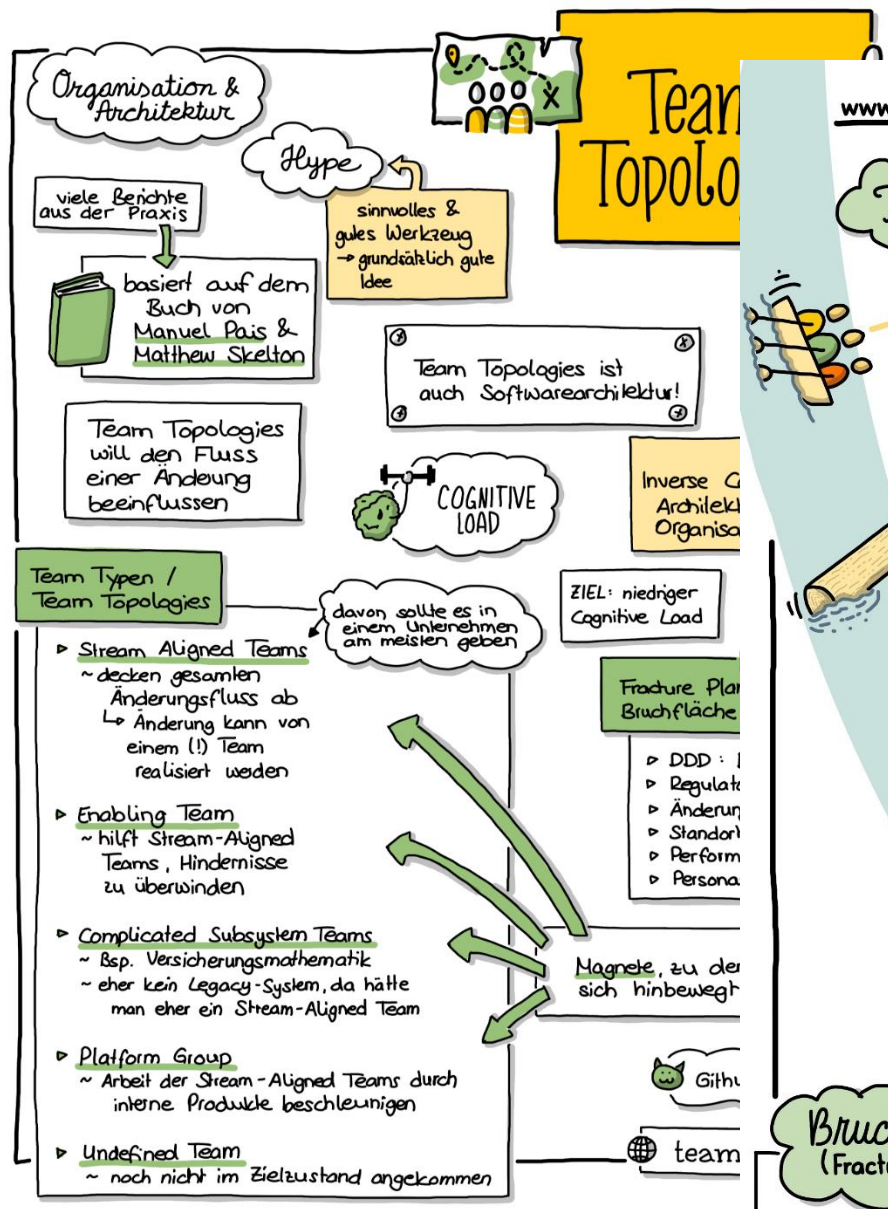


Team Topologies



Team Topologies: Result

- Team setup defined
- Collaborations defined


Organisation & Architektur

Hype

viele Berichte aus der Praxis

sinnvolles & gutes Werkzeug → grundsätzlich gute Idee

basiert auf dem Buch von Manuel Pais & Matthew Skelton

Team Topologies ist auch Softwarearchitektur!

Team Topologies will den Fluss einer Änderung beeinflussen

COGNITIVE LOAD

Inverse C Architekt Organisa

Team Typen / Team Topologies

davon sollte es in einem Unternehmen am meisten geben

ZIEL: niedriger Cognitive Load

- Stream Aligned Teams
  ~ decken gesamten Änderungsfluss ab
  ↳ Änderung kann von einem (!) Team realisiert werden
- Enabling Team
  ~ hilft Stream-Aligned Teams, Hindernisse zu überwinden
- Complicated Subsystem Teams
  ~ Bsp. Versicherungsmathematik
  ~ eher kein Legacy-System, da hätte man eher ein Stream-Aligned Team
- Platform Group
  ~ Arbeit der Stream-Aligned Teams durch interne Produkte beschleunigen
- Undefined Team
  ~ noch nicht im Zielzustand angekommen

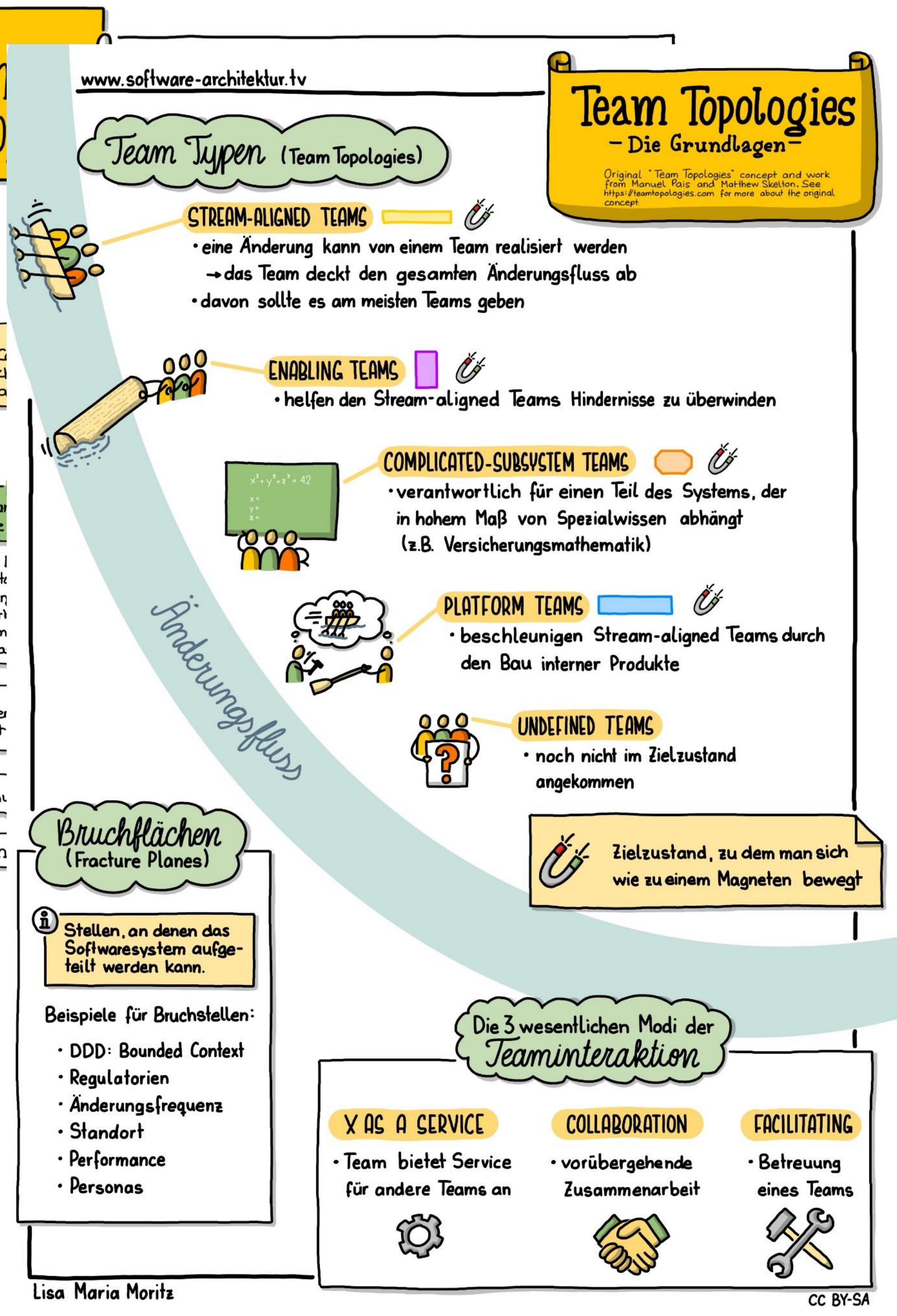
Fracture Plan Bruchfläche

- DDD :
- Regulat
- Änderun
- Standort
- Perform
- Persona

Magnete, zu den sich hinbewegt

Githu

team


www.software-architektur.tv

Team Topologies
– Die Grundlagen –

Original "Team Topologies" concept and work from Manuel Pais and Matthew Skelton. See https://teamtopologies.com for more about the original concept

Team Typen (Team Topologies)

STREAM-ALIGNED TEAMS
- eine Änderung kann von einem Team realisiert werden
  → das Team deckt den gesamten Änderungsfluss ab
- davon sollte es am meisten Teams geben

ENABLING TEAMS
- helfen den Stream-aligned Teams Hindernisse zu überwinden

COMPLICATED-SUBSYSTEM TEAMS
- verantwortlich für einen Teil des Systems, der in hohem Maß von Spezialwissen abhängt (z.B. Versicherungsmathematik)

PLATFORM TEAMS
- beschleunigen Stream-aligned Teams durch den Bau interner Produkte

UNDEFINED TEAMS
- noch nicht im Zielzustand angekommen

Zielzustand, zu dem man sich wie zu einem Magneten bewegt

Änderungsfluss

Bruchflächen (Fracture Planes)

Stellen, an denen das Softwaresystem aufgeteilt werden kann.

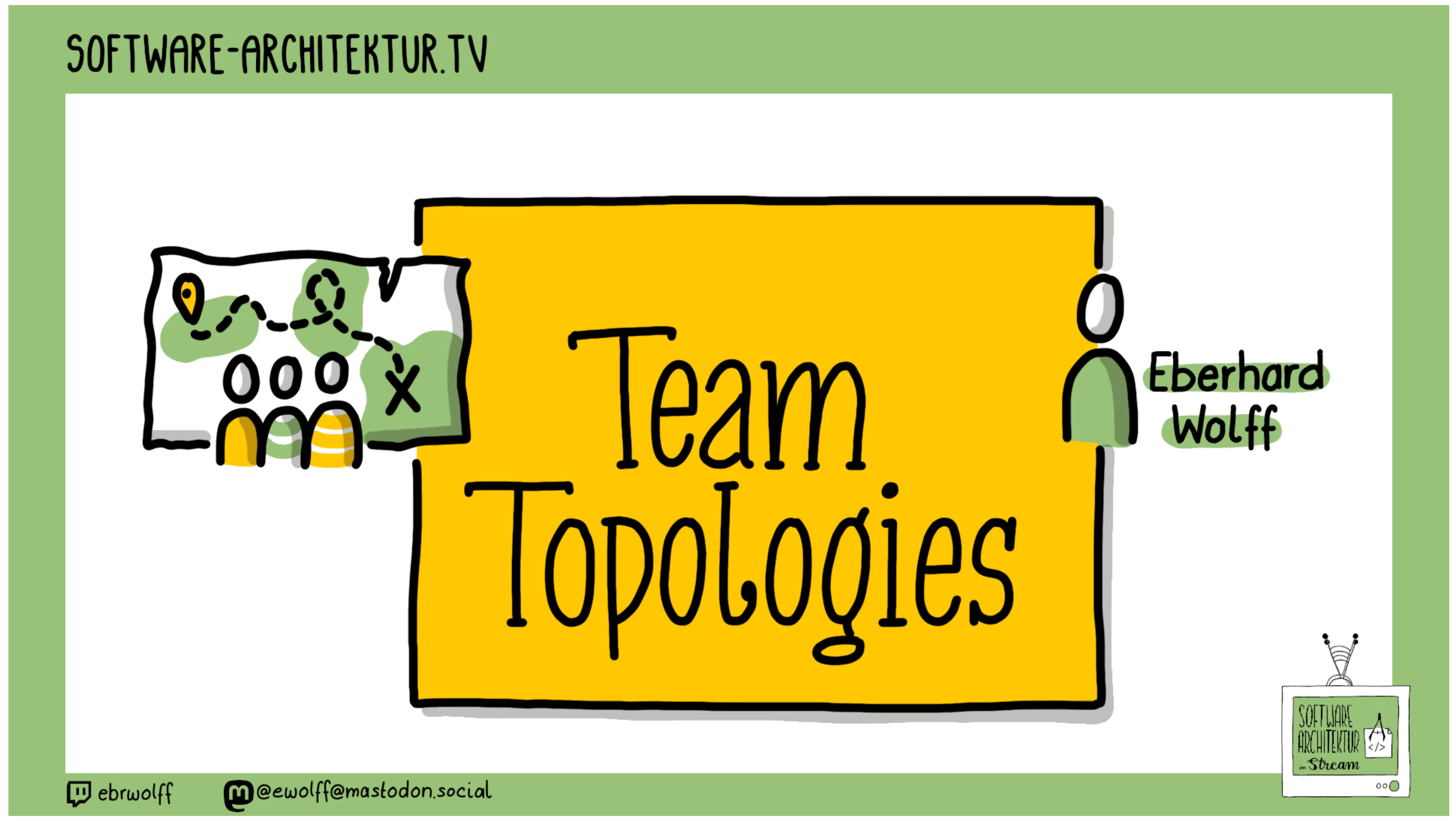
Beispiele für Bruchstellen:
- DDD: Bounded Context
- Regulatorien
- Änderungsfrequenz
- Standort
- Performance
- Personas

Die 3 wesentlichen Modi der Teaminteraktion

X AS A SERVICE	COLLABORATION	FACILITATING
Team bietet Service für andere Teams an	vorübergehende Zusammenarbeit	Betreuung eines Teams

Lisa Maria Moritz

CC BY-SA


SOFTWARE-ARCHITEKTUR.TV

Team Topologies

Eberhard Wolff

ebrwolff

@ewolff@mastodon.social

https://software-architektur.tv/2024/04/18/folge213.html

Finally: Coding! 🎉

Finally: Coding! 🎉
(sort of 😬)

Tactical Design

Why Tactical Design?

- Object-oriented concepts
- Make a lot of sense to build good object-oriented systems!

Tactical Design

Entity

Identity: a person

Value: 2€, 2m

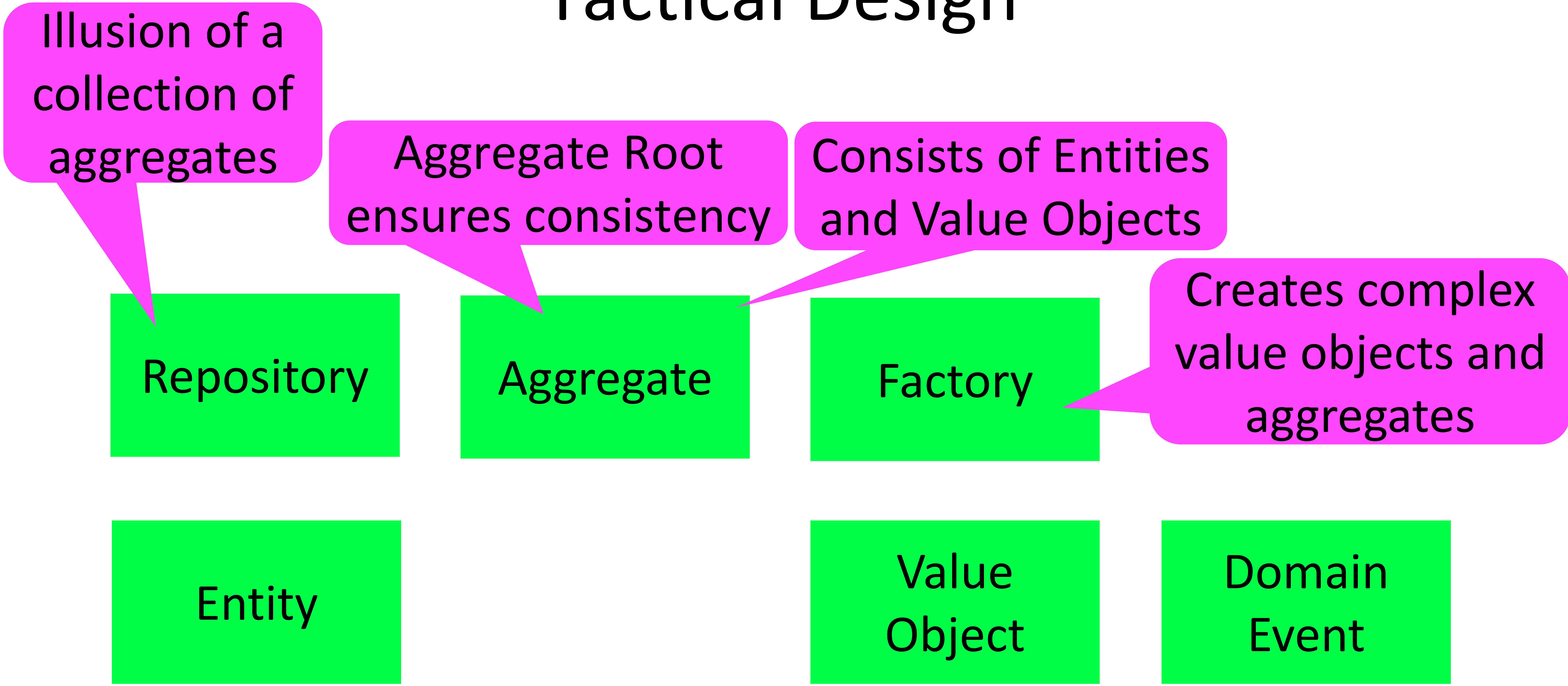
Value Object

Matters to domain experts

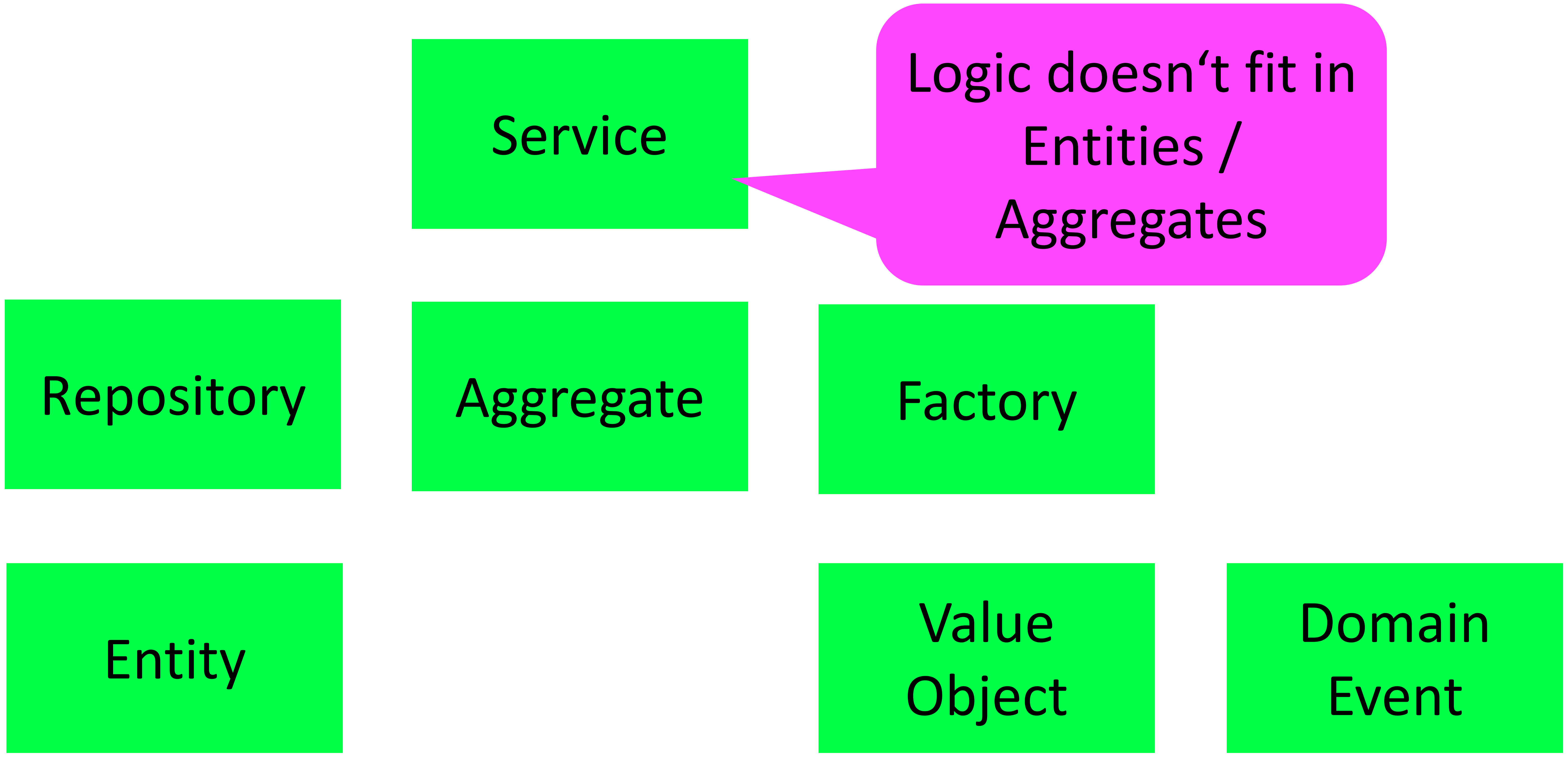
Domain Event

Something has happened.

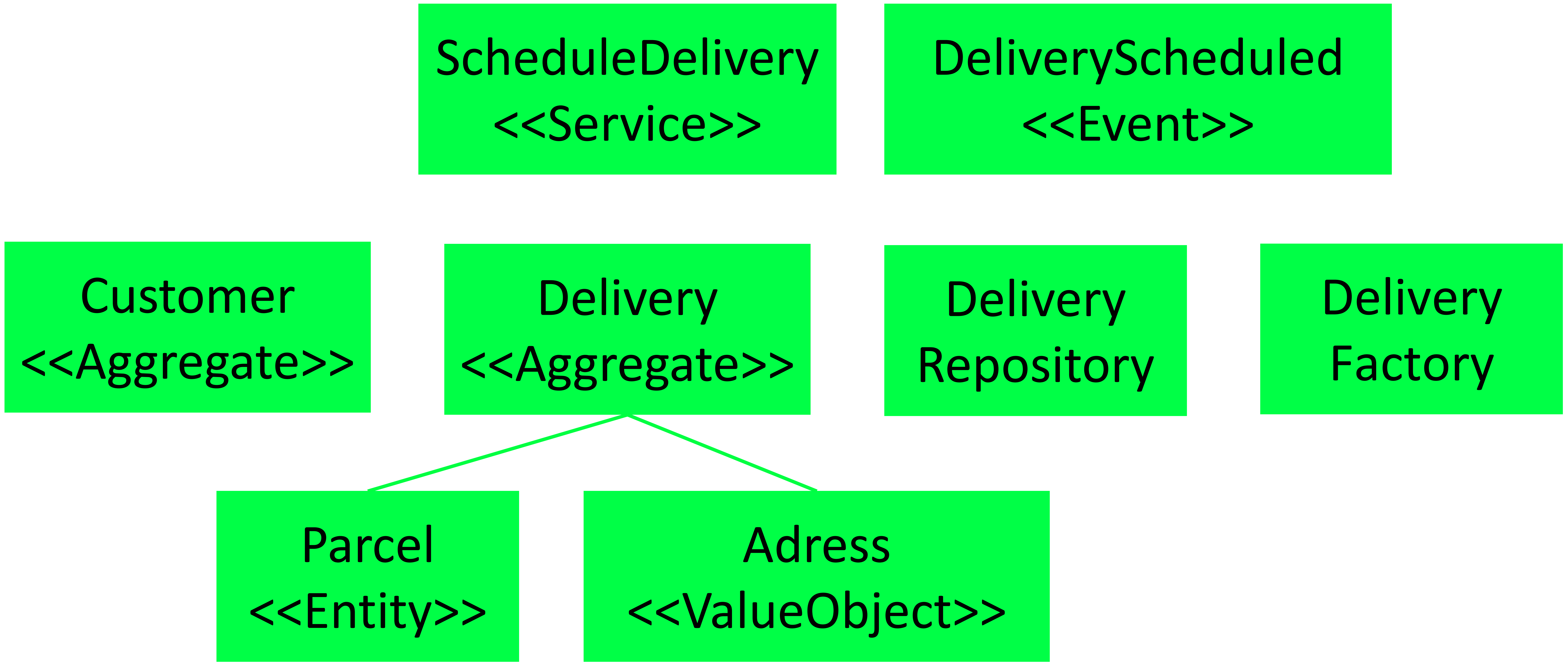
Tactical Design

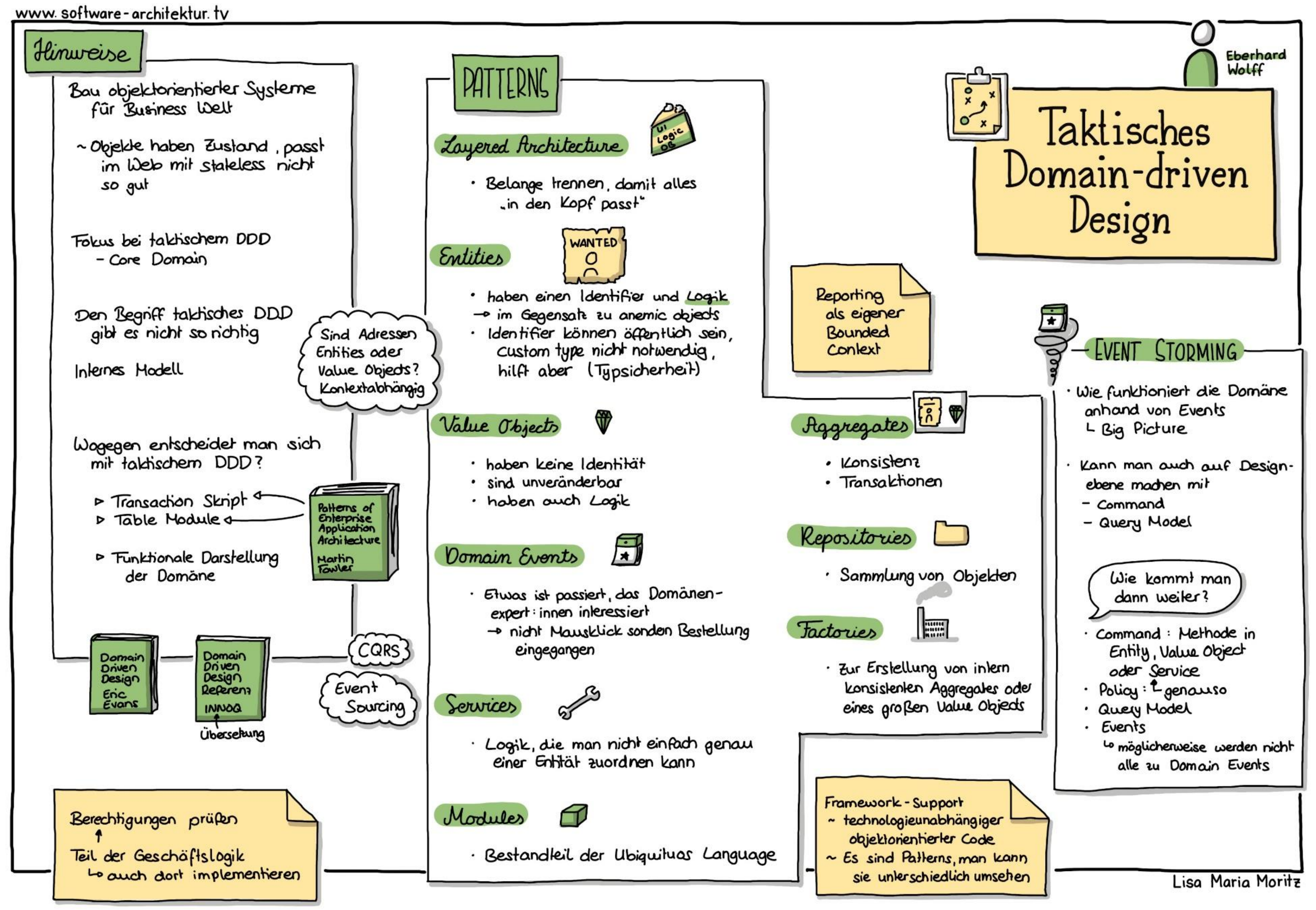

Illusion of a collection of aggregates
Aggregate Root ensures consistency
Consists of Entities and Value Objects
Creates complex value objects and aggregates
Repository
Aggregate
Factory
Entity
Value Object
Domain Event

Tactical Design



Example: Delivery Bounded Context





www. software-architektur. tv

Taktisches Domain-driven Design

Eberhard Wolff

Hinweise

Bau objektorientierter Systeme für Business Welt

~ Objekte haben Zustand, passt im Web mit stateless nicht so gut

Fokus bei taktischem DDD
- Core Domain

Den Begriff taktisches DDD gibt es nicht so richtig

Internes Modell

Sind Adressen Entities oder Value Objects? Kontextabhängig

Wogegen entscheidet man sich mit taktischem DDD?

- ▷ Transaction Skript
- ▷ Table Module
- ▷ Funktionale Darstellung der Domäne

Patterns of Enterprise Application Architecture Martin Fowler

Domain Driven Design Eric Evans

Domain Driven Design Referenz INNOQ
↑ Übersetzung

CQRS

Event Sourcing

Berechtigungen prüfen
↑
Teil der Geschäftslogik
↳ auch dort implementieren

PATTERNS

Layered Architecture

UI Logic DB

- Belange trennen, damit alles „in den Kopf passt"

Entities

WANTED

- haben einen Identifier und Logik
  → im Gegensatz zu anemic objects
- Identifier können öffentlich sein, custom type nicht notwendig, hilft aber (Typsicherheit)

Value Objects

- haben keine Identität
- sind unveränderbar
- haben auch Logik

Domain Events

- Etwas ist passiert, das Domänen-expert:innen interessiert
  → nicht Mausklick sondern Bestellung eingegangen

Services

- Logik, die man nicht einfach genau einer Entität zuordnen kann

Modules

- Bestandteil der Ubiquituos Language

Reporting als eigener Bounded Context

Aggregates

- Konsistenz
- Transaktionen

Repositories

- Sammlung von Objekten

Factories

- Zur Erstellung von intern konsistenten Aggregates oder eines großen Value Objects

Framework-Support
~ technologieunabhängiger objektorientierter Code
~ Es sind Patterns, man kann sie unterschiedlich umsetzen

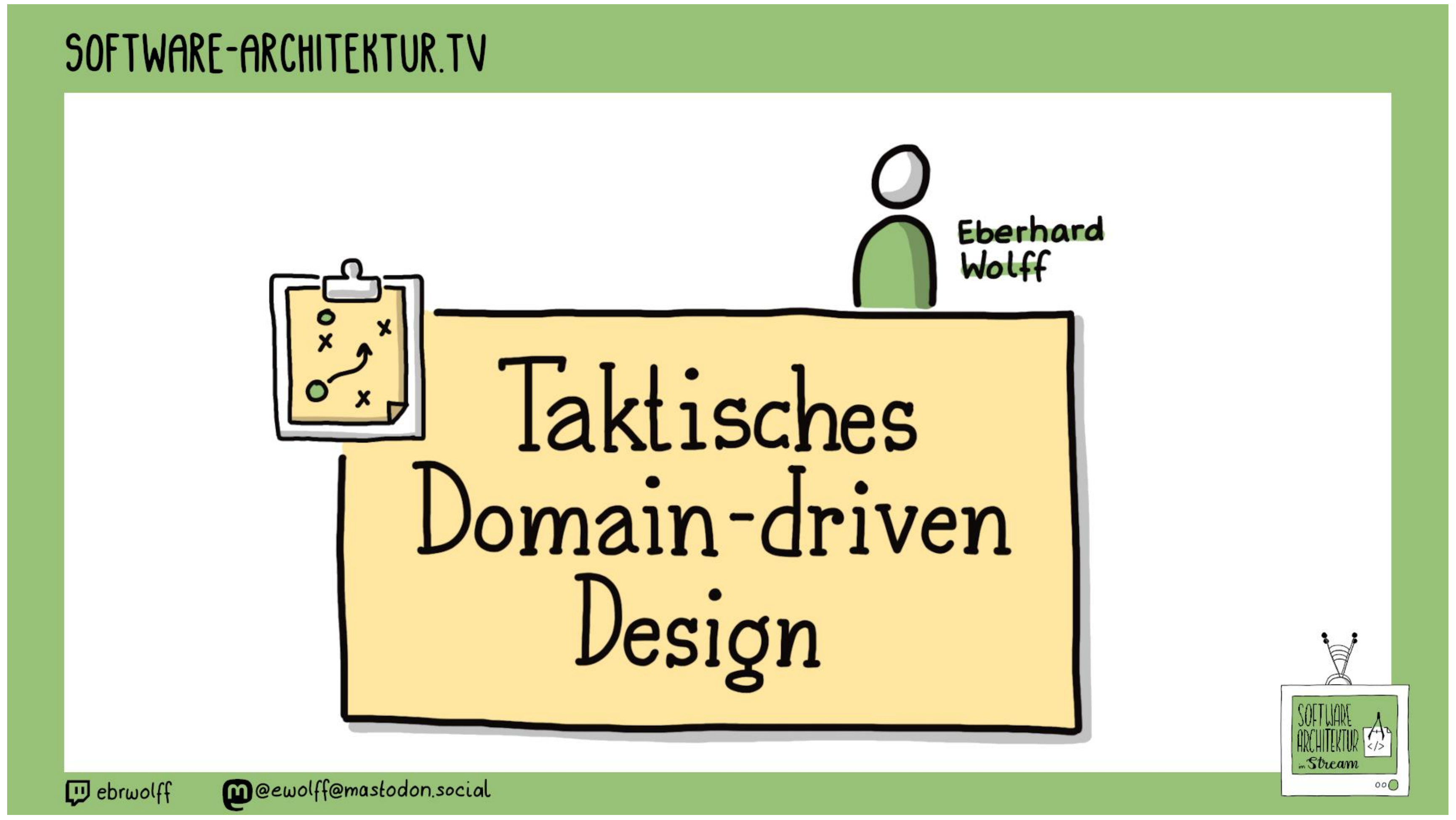
EVENT STORMING

- Wie funktioniert die Domäne anhand von Events
  └ Big Picture
- Kann man auch auf Design-ebene machen mit
  - Command
  - Query Model

Wie kommt man dann weiter?

- Command : Methode in Entity, Value Object oder Service
- Policy : ↑ genauso
- Query Model
- Events
  ↳ möglicherweise werden nicht alle zu Domain Events

Lisa Maria Moritz



https://software-architektur.tv/2024/05/03/folge214.html

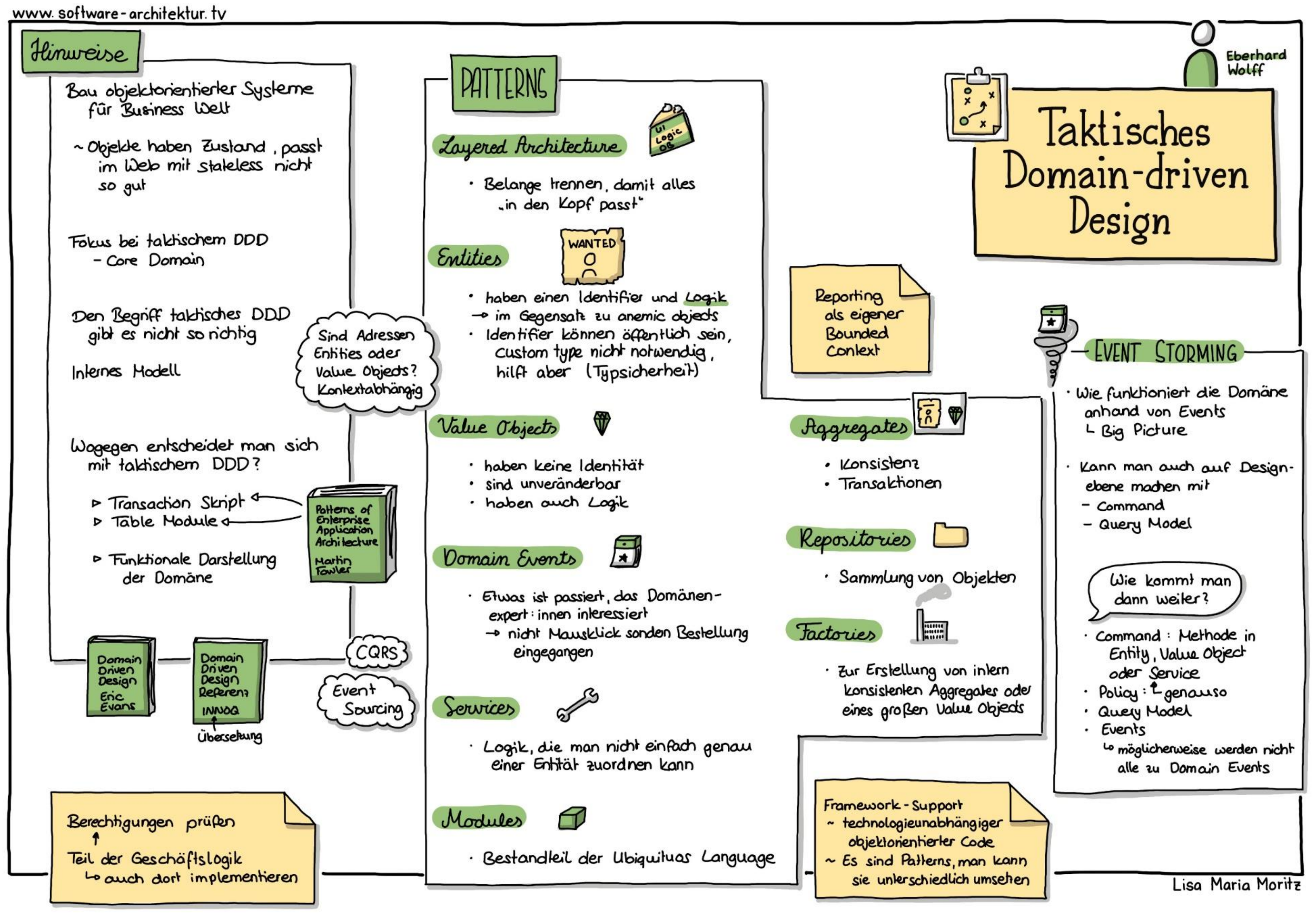
SWAGLab

Alternative: Functional Programming

- Side-effect free functional core
- i.e. no entities, aggregates etc
- Side-effects separated

Alternatives for Less Complex Systems

- Transaction script: handles a single request from the presentation incl. database code.
- Table model: Single instance handles business logic for all rows in a database table or view.



www. software-architektur. tv

Hinweise

Bau objektorientierter Systeme für Business Welt

~ Objekte haben Zustand, passt im Web mit stateless nicht so gut

Fokus bei taktischem DDD
- Core Domain

Den Begriff taktisches DDD gibt es nicht so richtig

Internes Modell

Sind Adressen Entities oder Value Objects? Kontextabhängig

Wogegen entscheidet man sich mit taktischem DDD?

- ▷ Transaction Skript
- ▷ Table Module
- ▷ Funktionale Darstellung der Domäne

Patterns of Enterprise Application Architecture Martin Fowler

Domain Driven Design Eric Evans

Domain Driven Design Referenz INNOQ

Übersetzung

CQRS

Event Sourcing

Berechtigungen prüfen
↑
Teil der Geschäftslogik
↳ auch dort implementieren

PATTERNS

Layered Architecture

- Belange trennen, damit alles „in den Kopf passt"

Entities

- haben einen Identifier und Logik
- → im Gegensatz zu anemic objects
- Identifier können öffentlich sein, custom type nicht notwendig, hilft aber (Typsicherheit)

Value Objects

- haben keine Identität
- sind unveränderbar
- haben auch Logik

Domain Events

- Etwas ist passiert, das Domänenexpert:innen interessiert
  → nicht Mausklick sonden Bestellung eingegangen

Services

- Logik, die man nicht einfach genau einer Entität zuordnen kann

Modules

- Bestandteil der Ubiquituos Language

Reporting als eigener Bounded Context

Aggregates

- Konsistenz
- Transaktionen

Repositories

- Sammlung von Objekten

Factories

- Zur Erstellung von intern konsistenten Aggregates oder eines großen Value Objects

Framework-Support
~ technologieunabhängiger objektorientierter Code
~ Es sind Patterns, man kann sie unterschiedlich umsetzen

Taktisches Domain-driven Design

Eberhard Wolff

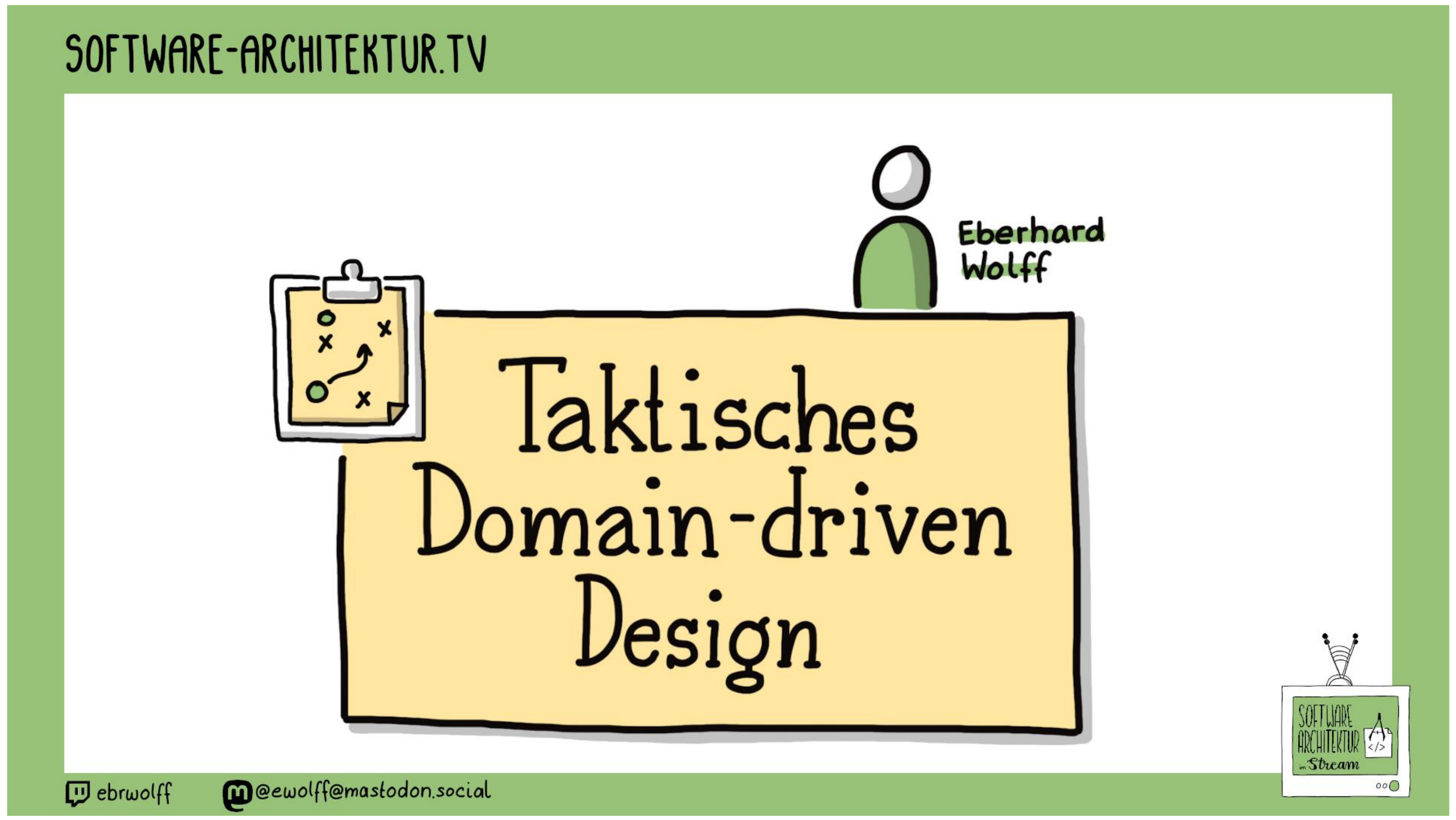
EVENT STORMING

- Wie funktioniert die Domäne anhand von Events
  └ Big Picture
- Kann man auch auf Designebene machen mit
  - Command
  - Query Model

Wie kommt man dann weiter?

- Command: Methode in Entity, Value Object oder Service
- Policy: ↑ genauso
- Query Model
- Events
  ↳ möglicherweise werden nicht alle zu Domain Events

Lisa Maria Moritz

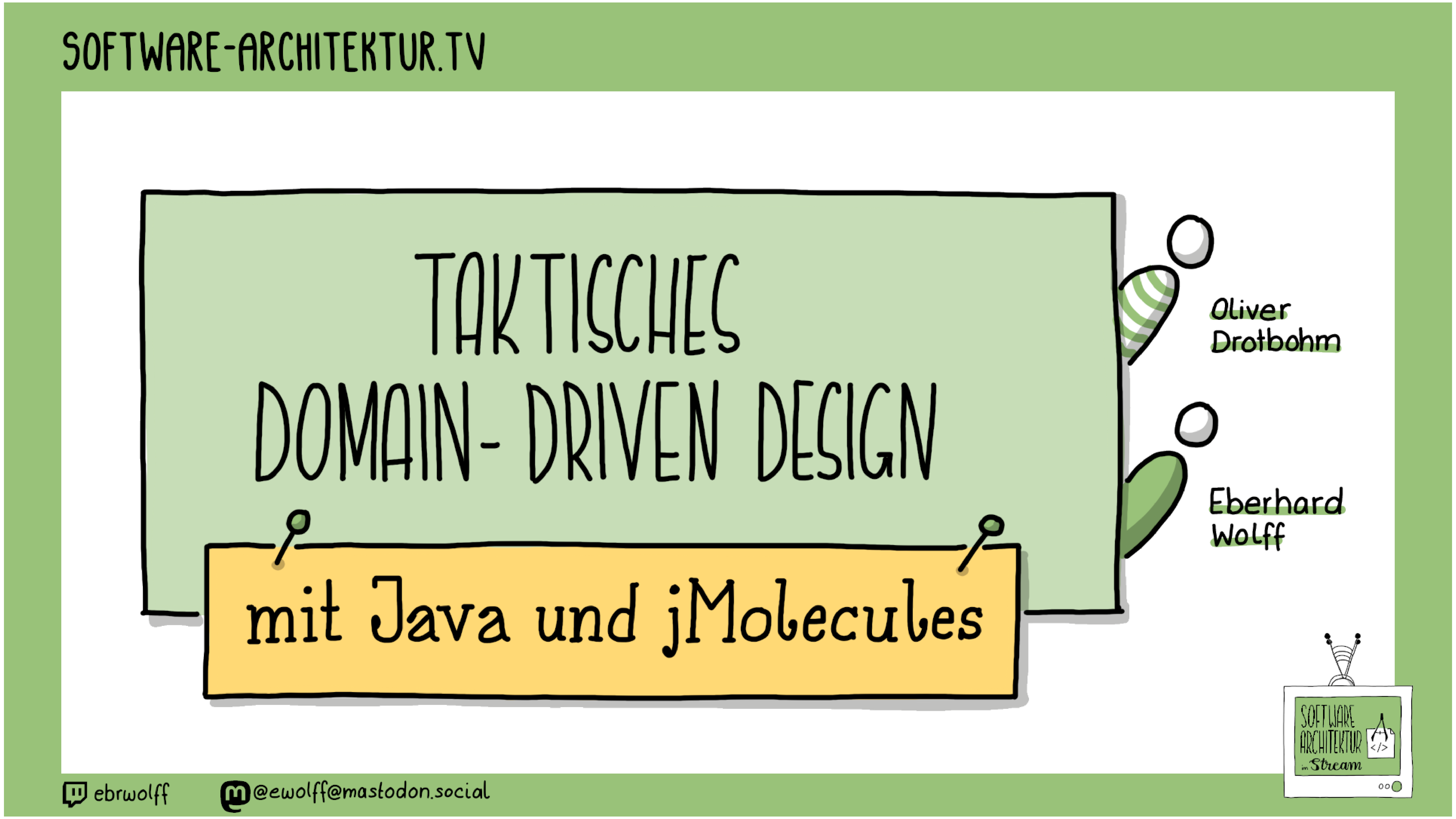


https://software-architektur.tv/2024/05/03/folge214.html

SWAGLab

Framework?

- POJO (Plain Old Java Objects) might be enough
- https://xmolecules.org/ supports DDD concepts
- https://odrotbohm.de/2020/03/Implementing-DDD-Building-Blocks-in-Java/ describes the idea


SOFTWARE-ARCHITEKTUR.TV

TAKTISCHES DOMAIN-DRIVEN DESIGN

mit Java und jMolecules

Oliver Drotbohm

Eberhard Wolff

ebrwolff

@ewolff@mastodon.social

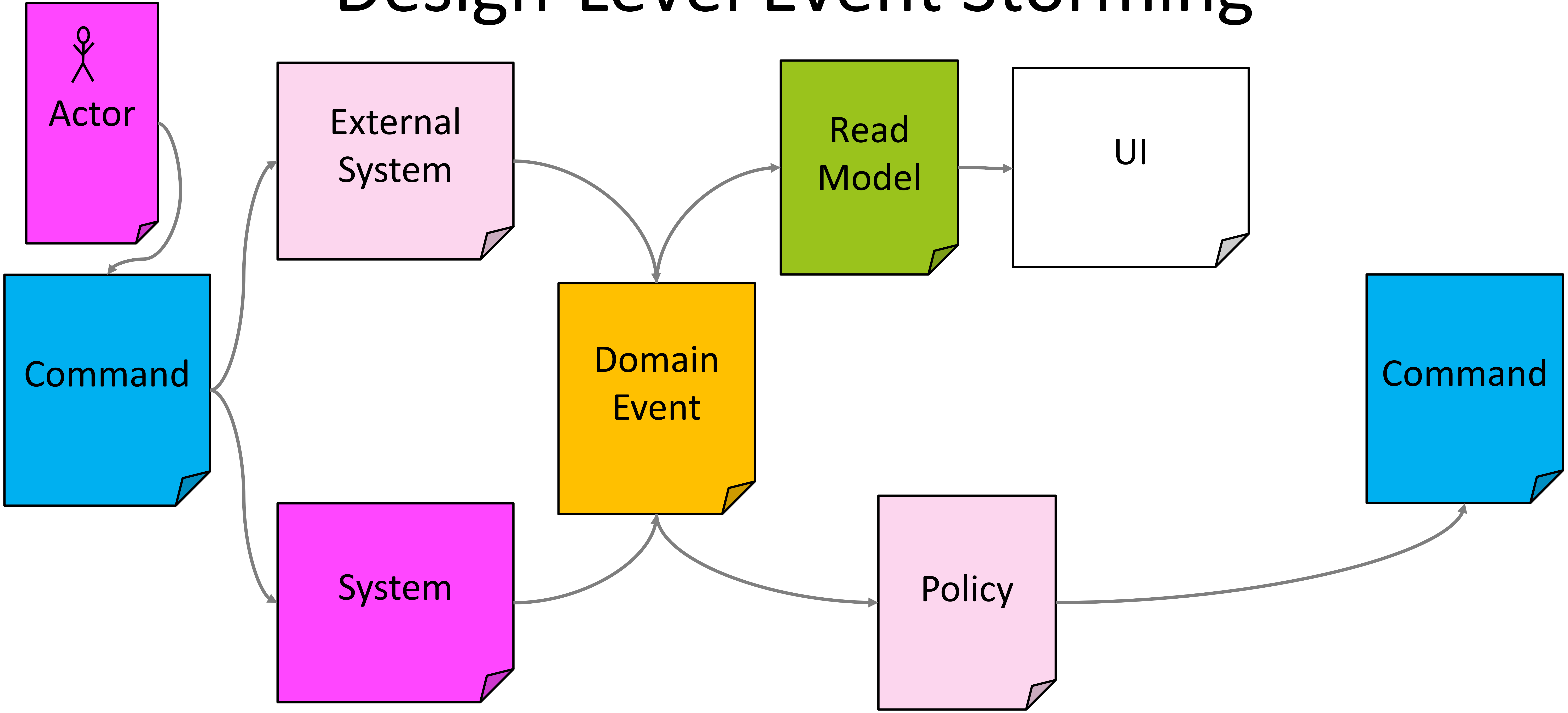
SOFTWARE ARCHITEKTUR im Stream

https://software-architektur.tv/2024/05/31/episode219.html

Framework?

- Might use architecture management tools to enforce dependencies
- https://software-architektur.tv/tags.html#Architecture%20Management

Design-Level Event Storming

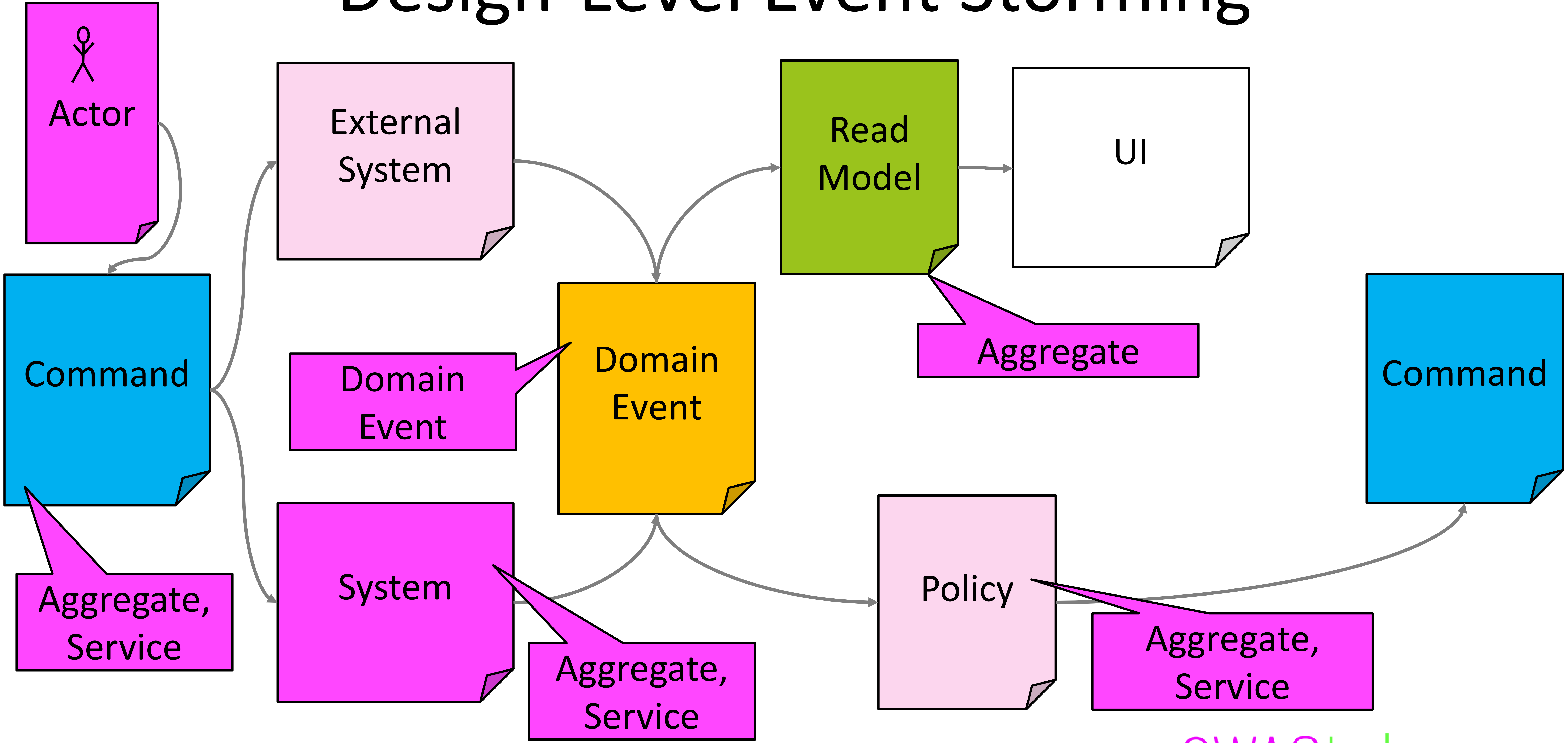


SWAGLab

Design-Level Event Storming

- Helps to understand the domain on the necessary level of detail
- But no easy mapping to tactical domain-driven design

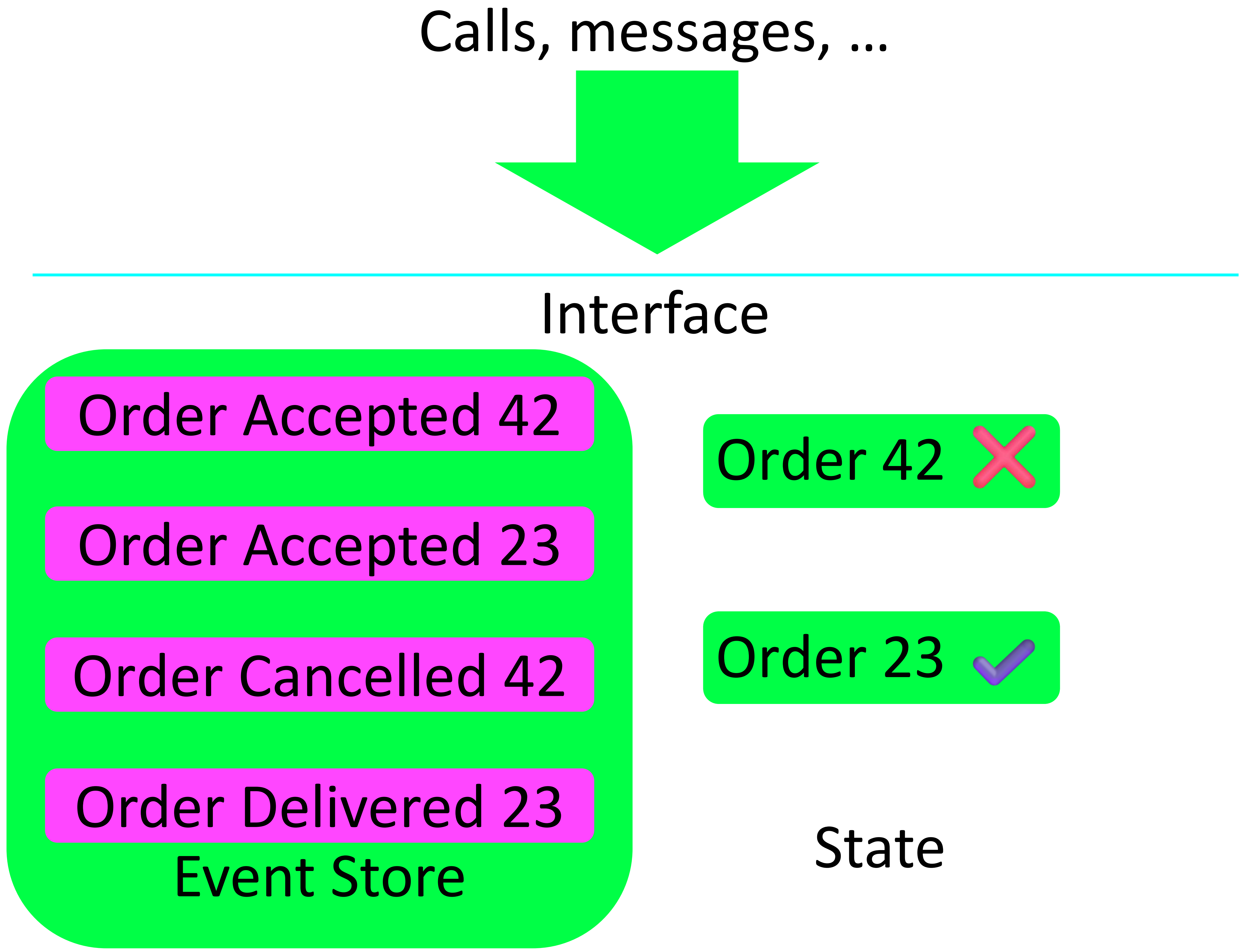
Design-Level Event Storming



Event Sourcing

- Store events that lead to a specific state
- Might also store state (optional)

- System of record: State or events


Calls, messages, …
Interface
Order Accepted 42
Order Accepted 23
Order Cancelled 42
Order Delivered 23
Event Store
Order 42 ✗
Order 23 ✓
State

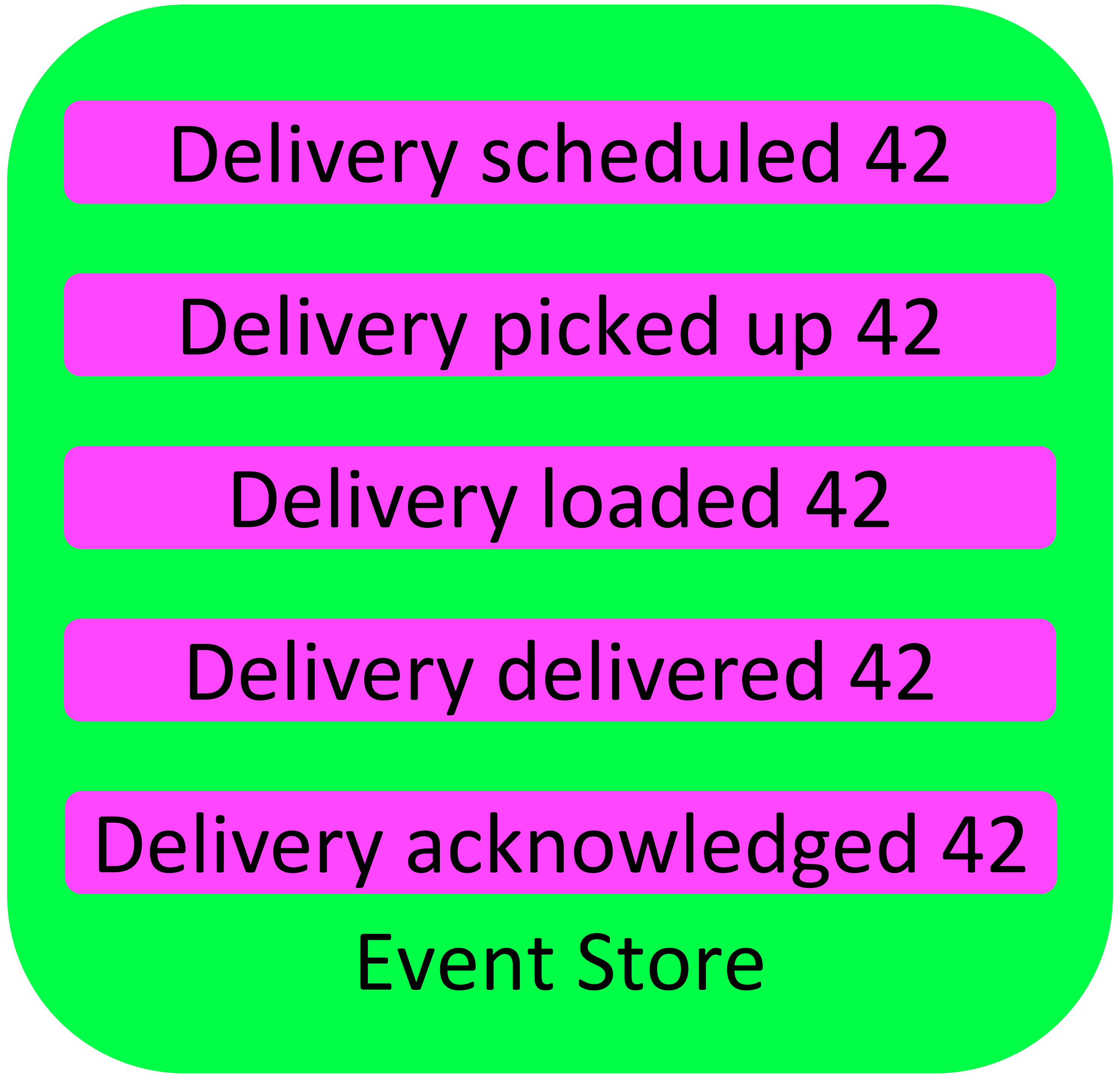
Event Sourcing

Events Calculate State on Demand	State+ Events	State

Event Sourcing Example

Can you model delivery without an event store?

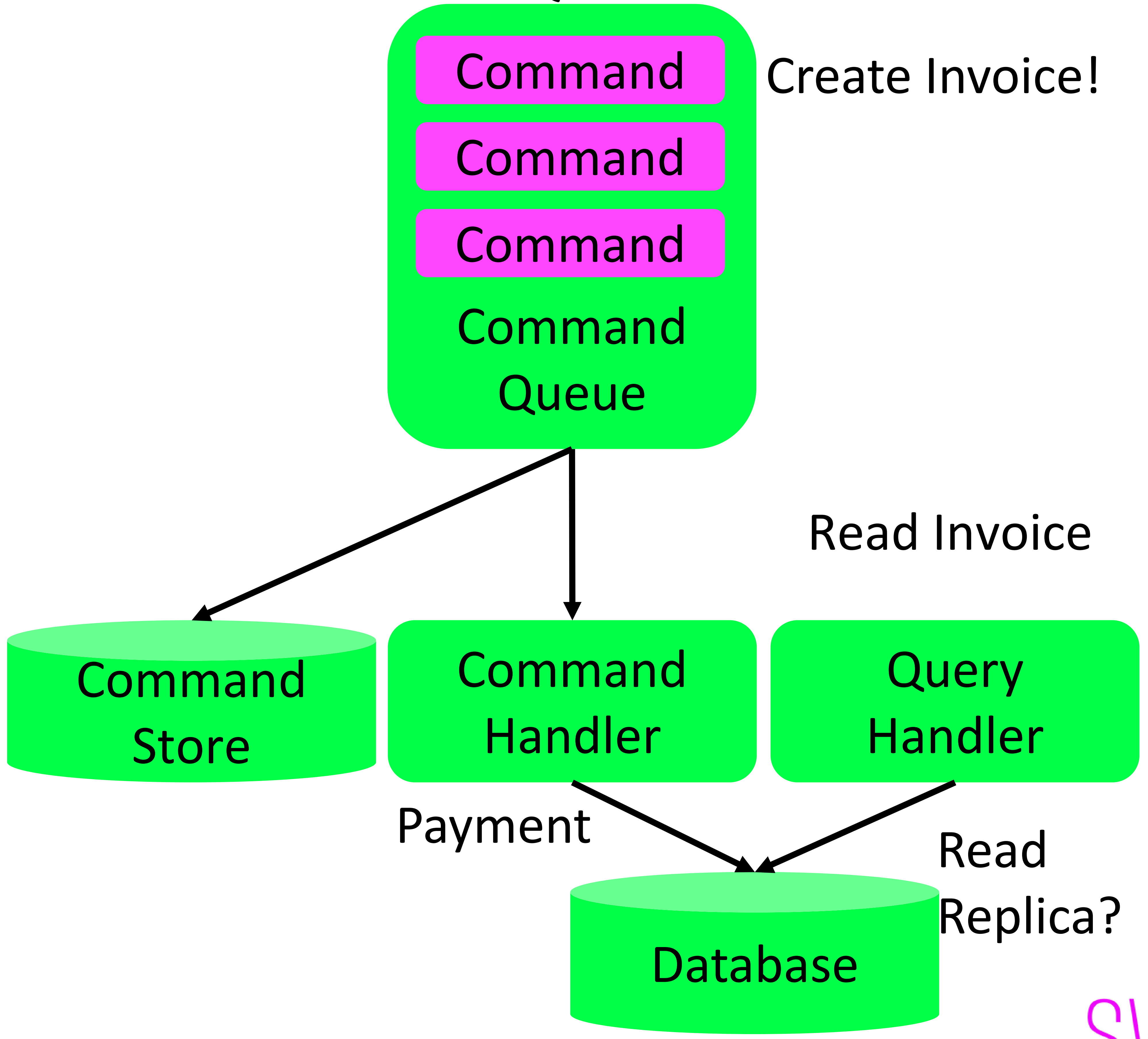
Why calculate the state of a delivery based on the events and not store the state?



CQRS

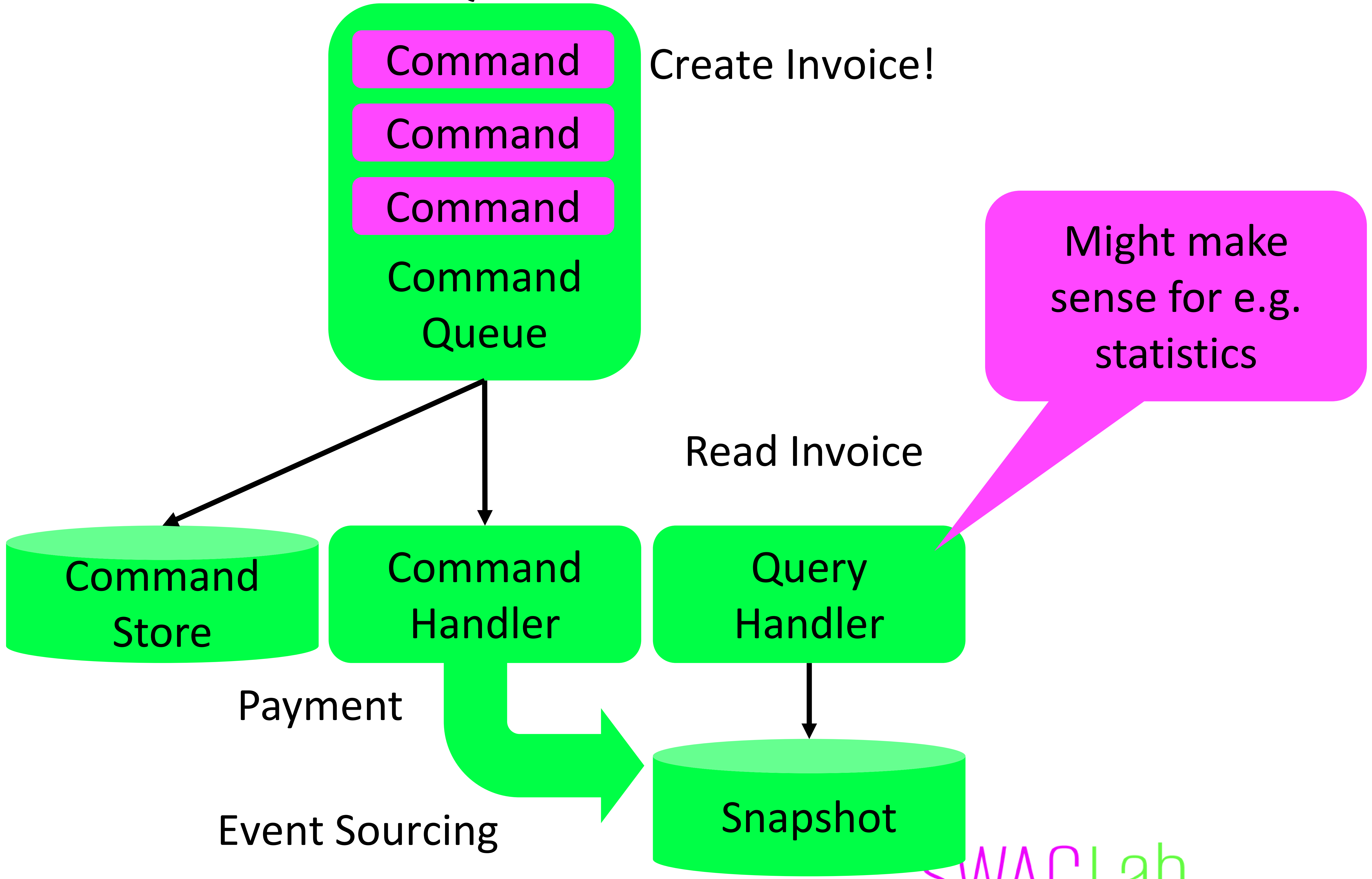
- Command Query Responsibility Separation
- E.g. separate read and write

CQRS



SWAGLab

CQRS

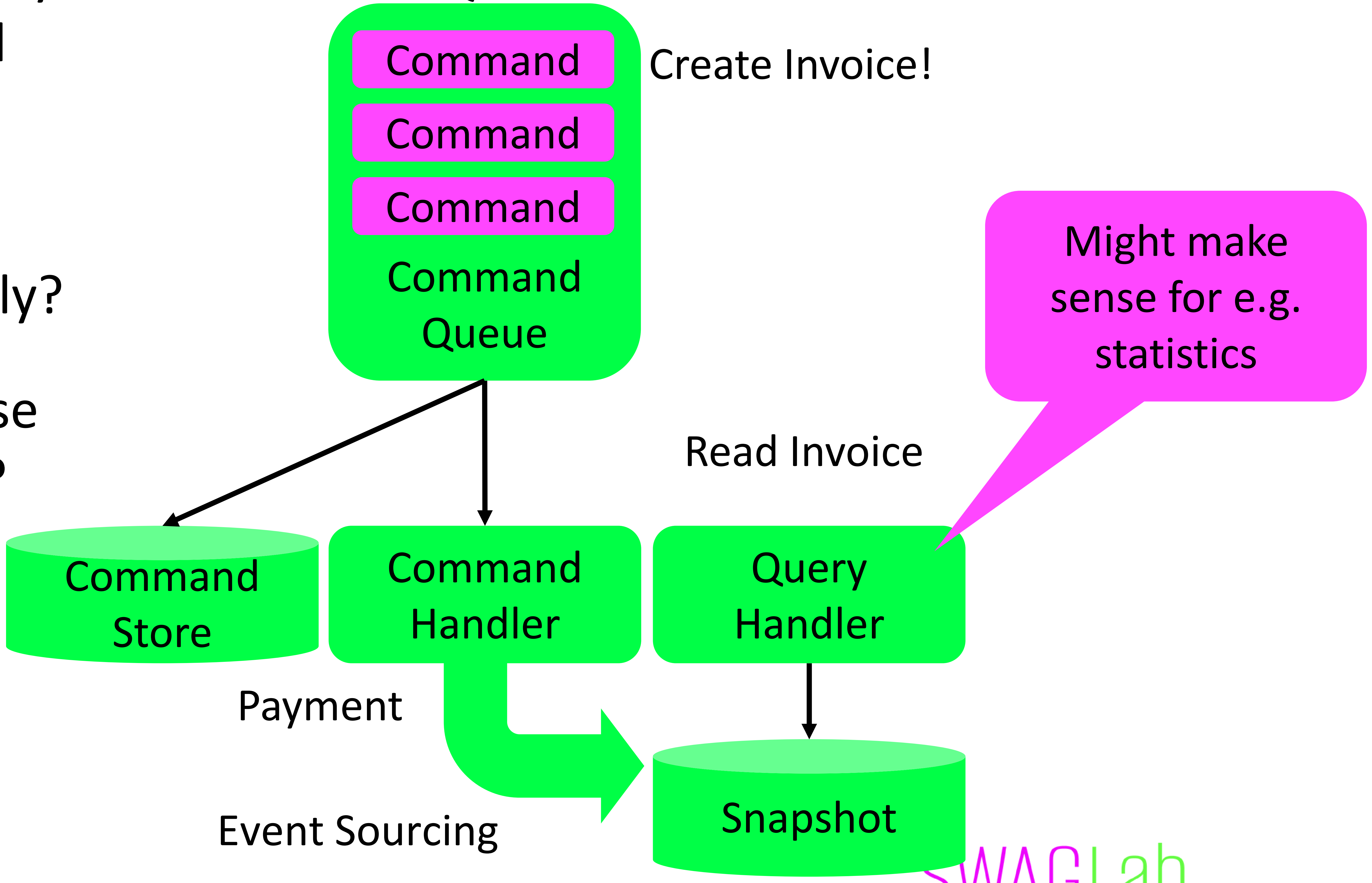

Command
Command
Command
Command Queue
Create Invoice!
Might make sense for e.g. statistics
Read Invoice
Command Store
Command Handler
Query Handler
Payment
Snapshot
Event Sourcing

CQRS

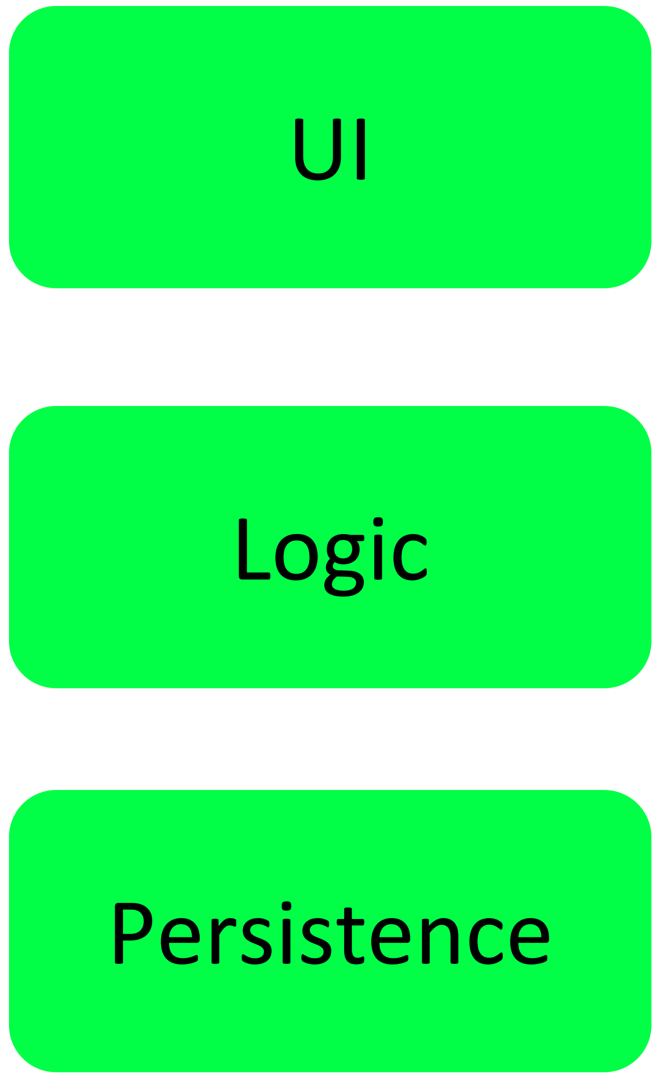
Statistics is probably a different bounded context.

Can you model statistics differently?

Why would you use CQRS for the rest?

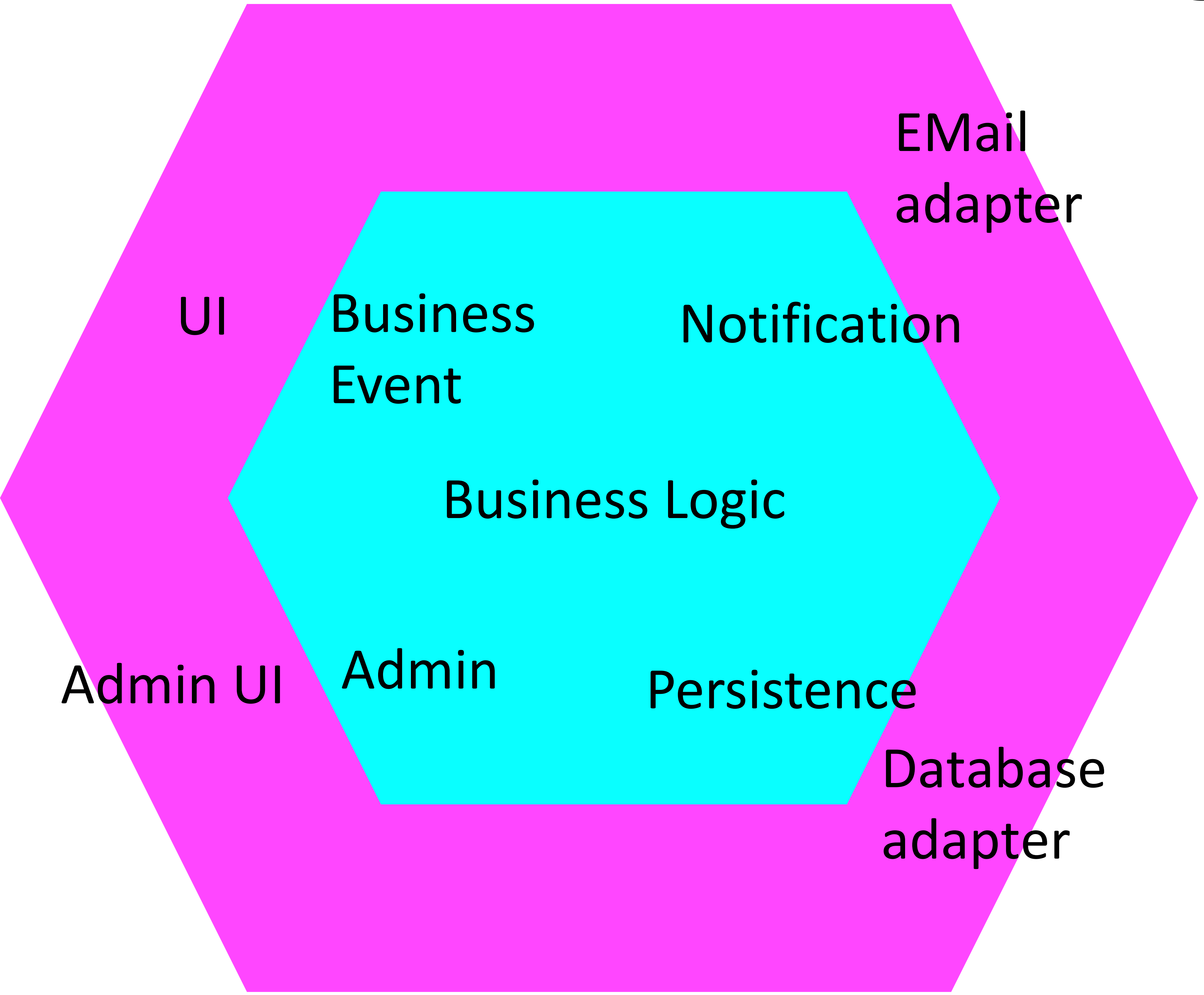


Layers



- Actually pattern in the original DDD book.
- Separate logic
- To isolate logic in one place
- To make the system easier to understand

Hexagonal



- Business logic exports ports
- Adapters implement ports
- Logic isolated and easy to understand
- Better testability

Conclusion

Conclusion

Big Picture Event Storming

Strategic Design?

Team Topologies

Core Domain

Bounded Context

Tactical Design

Design-level Event Storming

Event Sourcing?

CQRS?

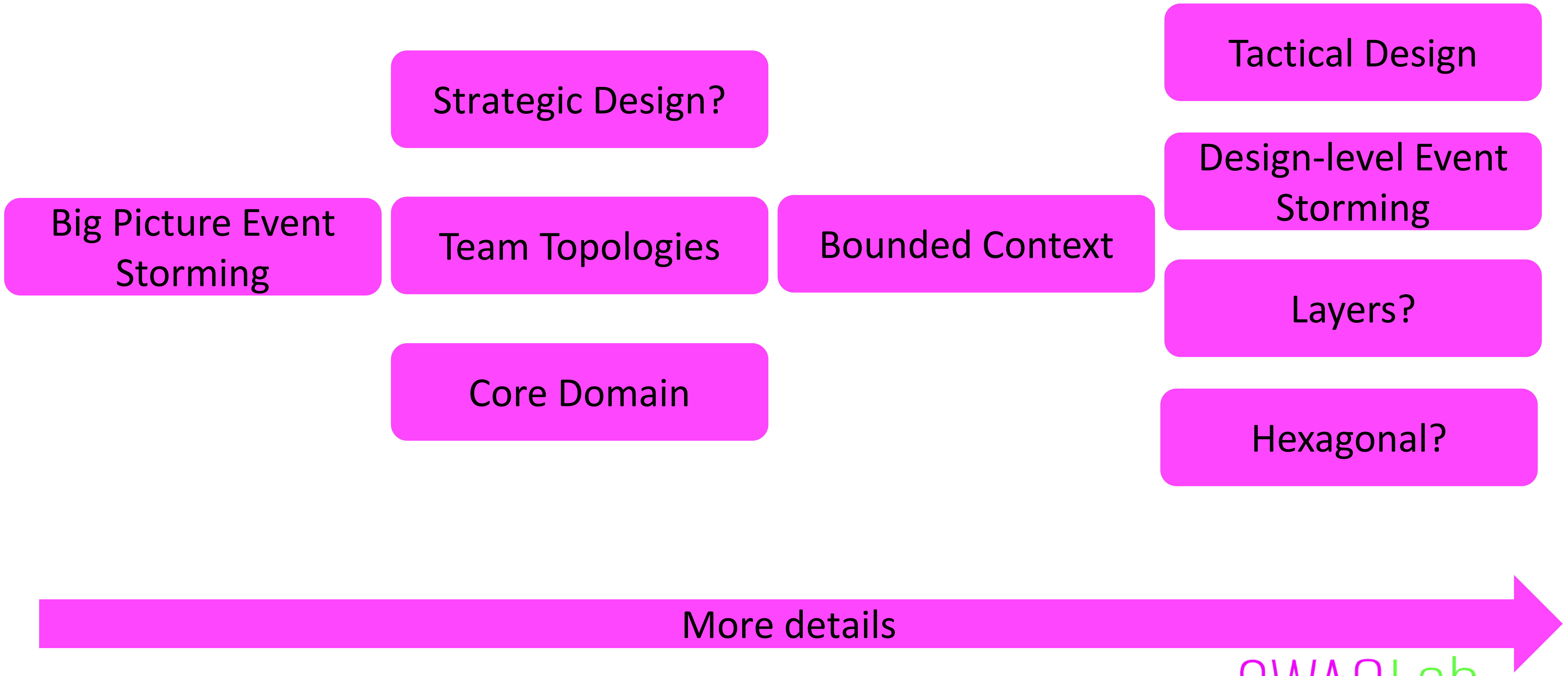
Layers?

Hexagonal?

More details

SWAGLab

Conclusion

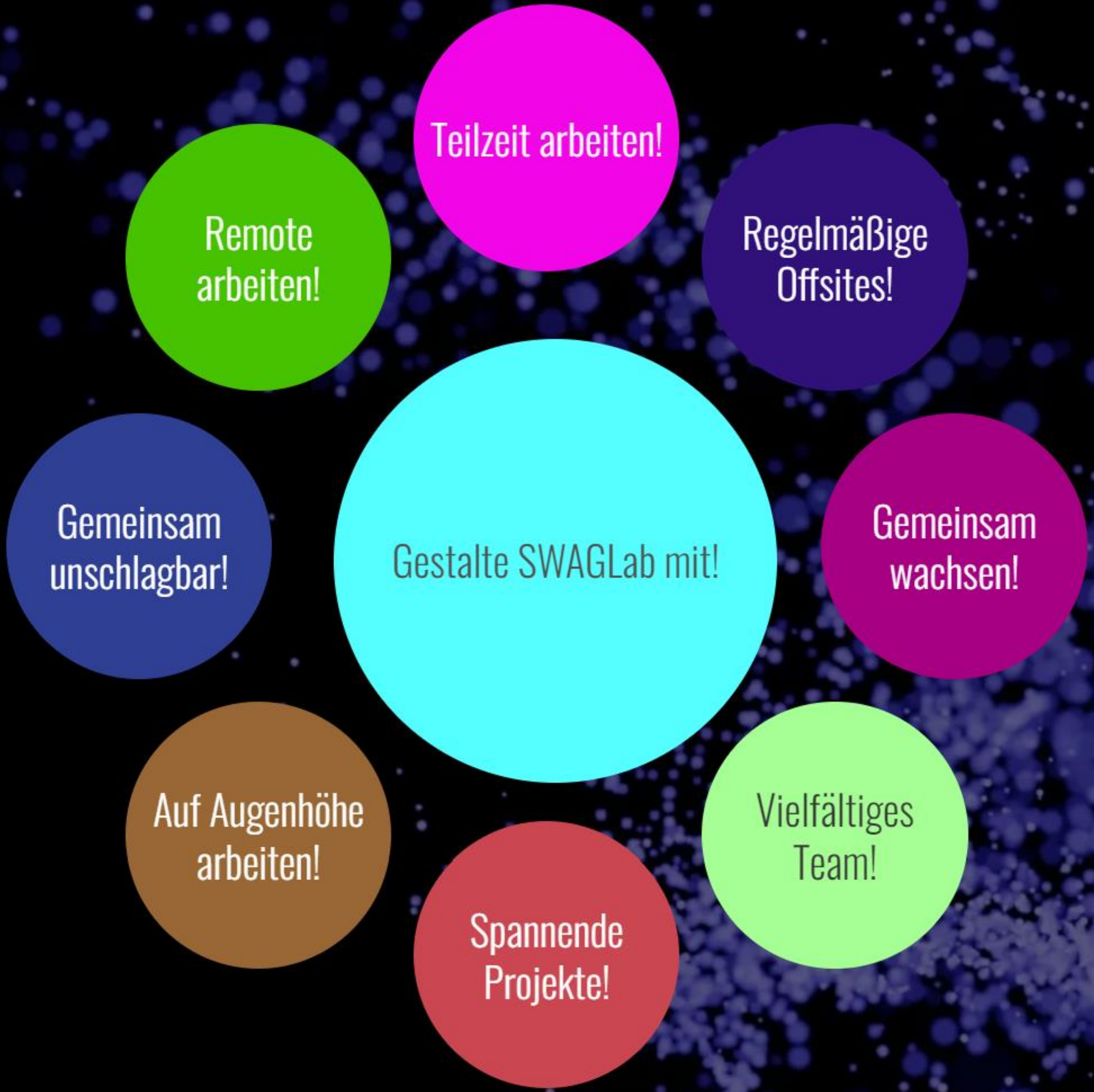

Big Picture Event Storming
Strategic Design?
Team Topologies
Core Domain
Bounded Context
Tactical Design
Design-level Event Storming
Layers?
Hexagonal?
More details

Note

- This might look like a waterfall.
- It is about different levels of abstractions.

- Work in iterations!
- Change the level of abstraction!

Mitgestalter:innen gesucht!



SOCREATORY
The Software Creators' Academy

https://www.socreatory.com/trainers/eberhard-wolff

Spare 20% mit dem Code "JFS2024"

Softwarearchitektur-Kickstart

07.11., 14.11., 21.11., 28.11.2024
je 9:00 - 13:00 Uhr

Online

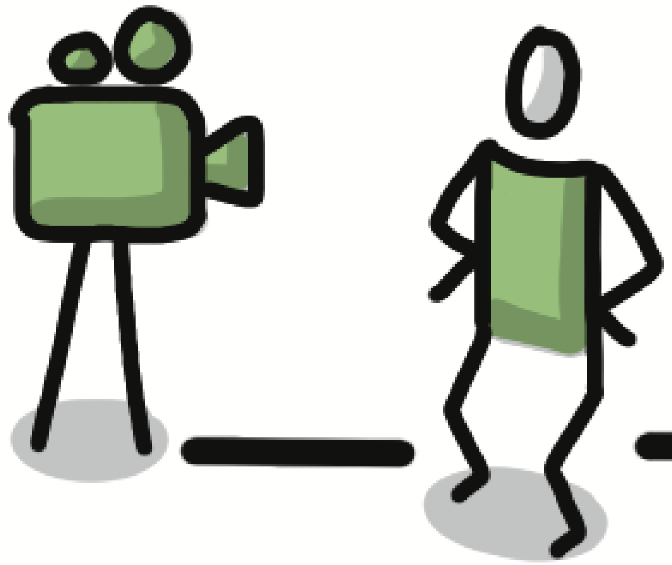
Deine Meinung ist gefragt:

WAS IST DER **wichtigste Skill** IN DER IT?

31.07.24

Java Forum Stuttgart, socreatory-Stand

Sei dabei



www.software-architektur.tv

Send email to jfs2024@ewolff.com

Slides

+ Sample Microservices Book DE / EN

+ Sample Practical Microservices DE/EN

+ Sample of Continuous Delivery Book DE

Powered by Amazon Lambda

& Microservices

EMail address logged for 14 days,
wrong addressed emails handled manually