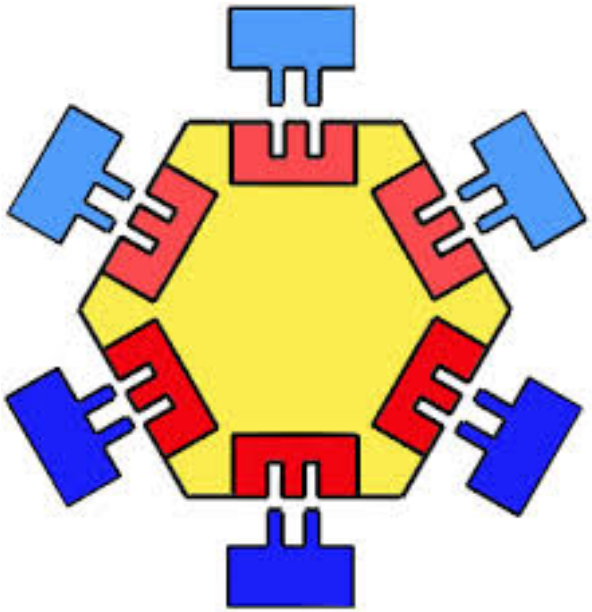
The Domain Driven Design and the Hexagonal Architecture

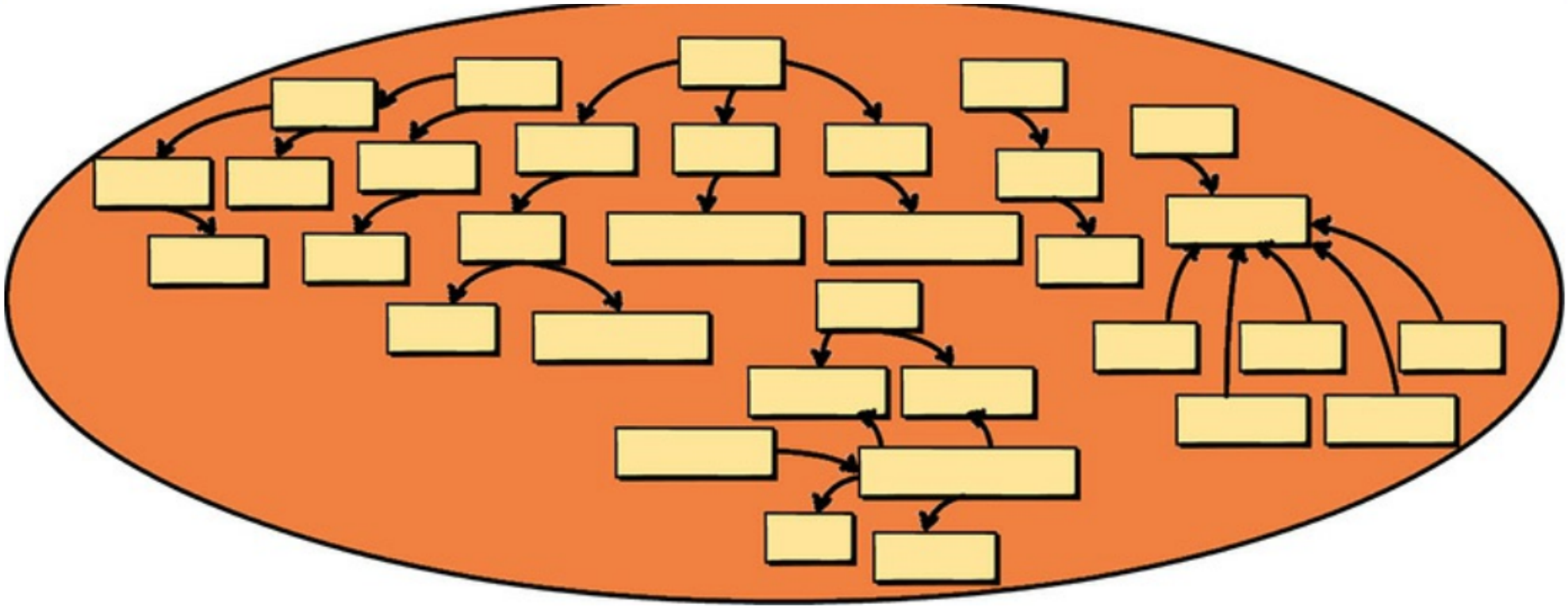
Auxenta Tech Talk - June 2017



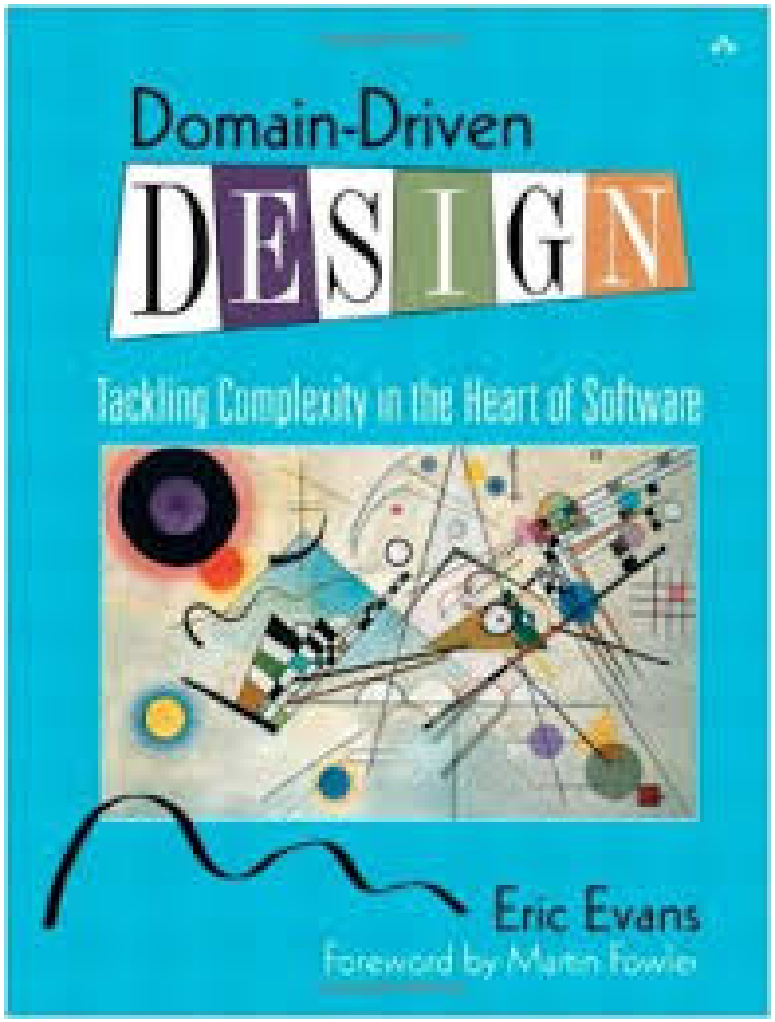
Auxenta

Crishantha Nanayakkara
(@crishantha)

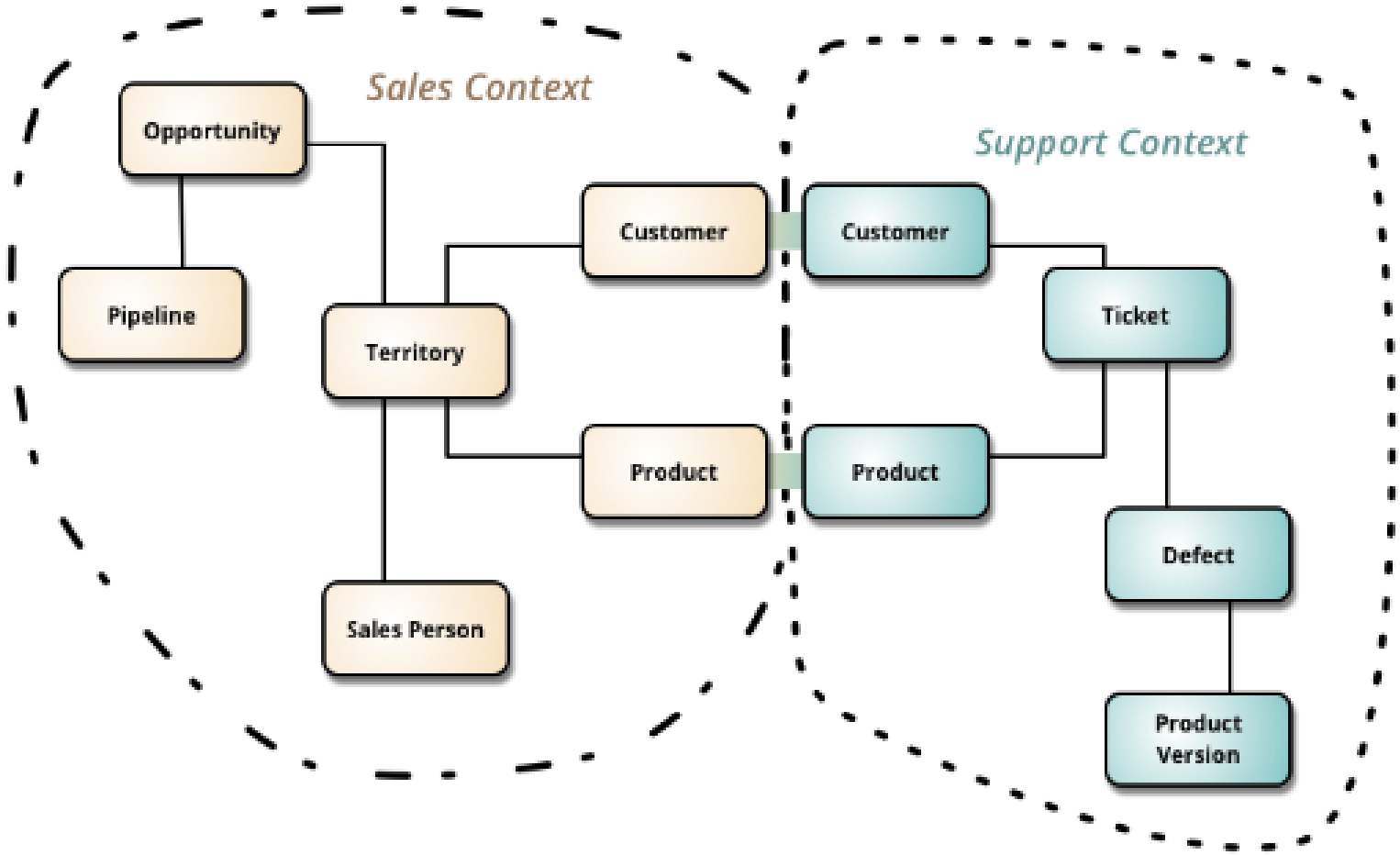
The Strategic Design



Domain Driven Design (DDD)

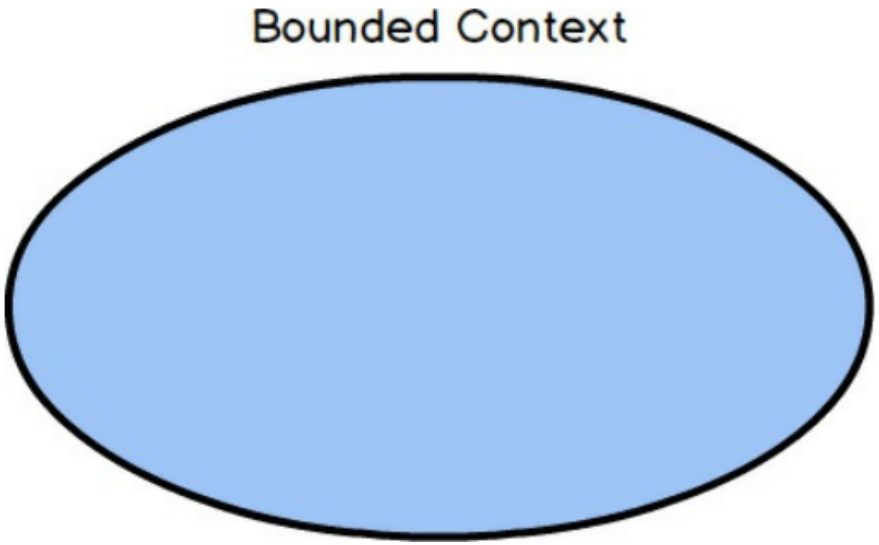


Domain Driven Design (DDD)



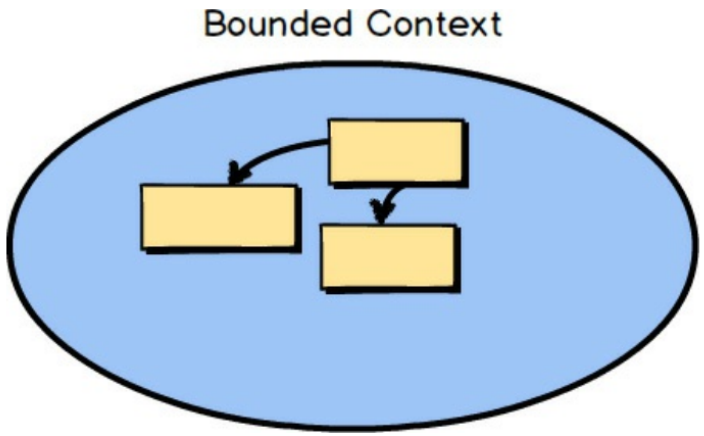
Bounded Context

- It is a “Strategic Design Pattern”.
- It describes a “Semantic Contextual Boundary”.
  - Within the boundary, each component of the software model has a specific meaning and does specific things.



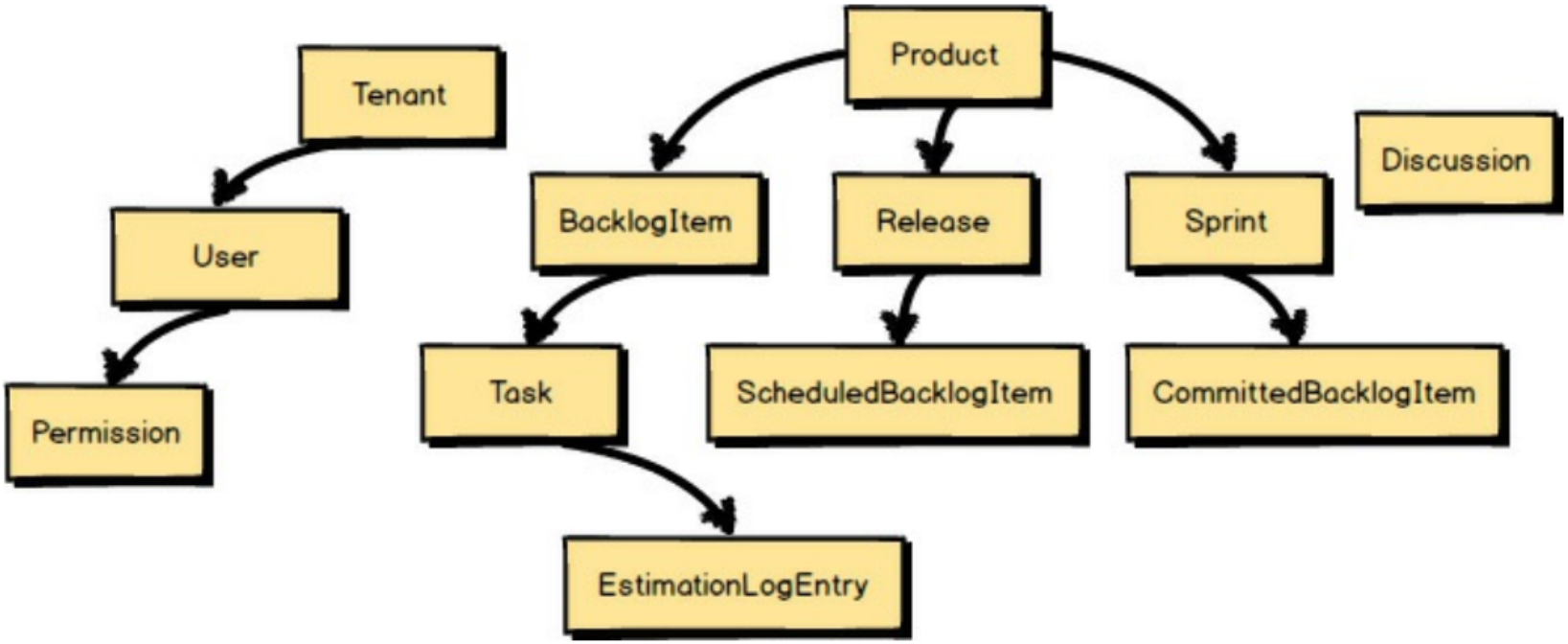
Core Domain(s)

- An Enterprise consists of multiple “Core Domain(s)”
- Core Domain = Bounded Context
- The boundary is completely determined by the Core Domain owner
- Each Core Domain / Bounded Context, consists of
  - A source code repository
  - A database
  - Set of Concepts



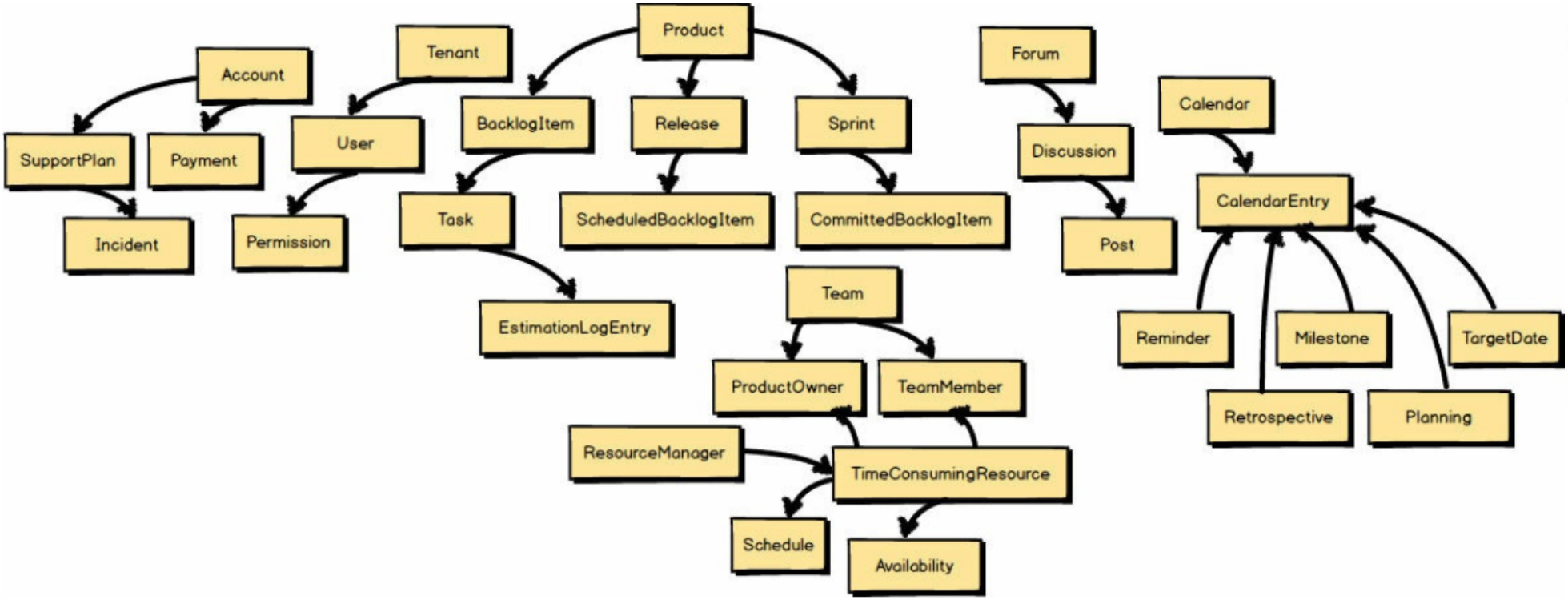
Bounded Context

The original set of Concepts



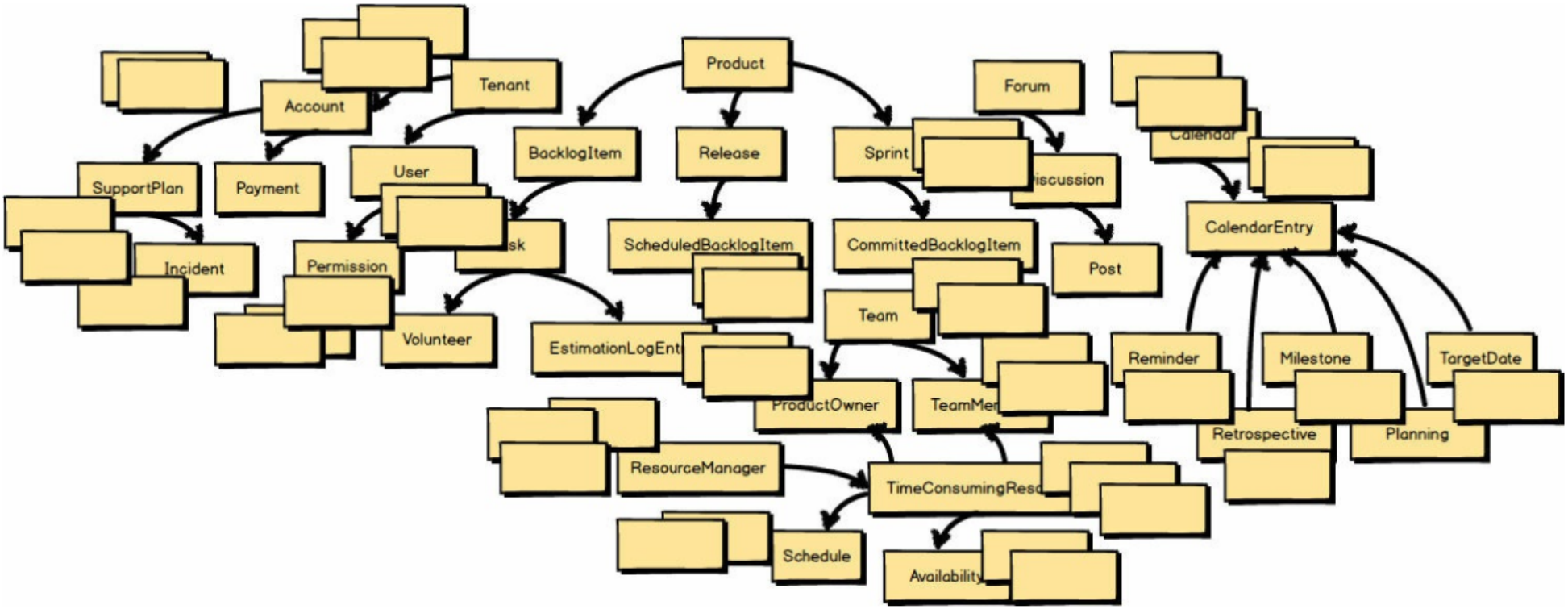
Bounded Context

The Concepts are kept on getting added



Bounded Context

The “Big Ball of Mud”



Bounded Context

- While selecting the Core Domain/ Bounded Context it is important to give the FOCUS on the business complexity than the technical complexity
- Hence, it is required the developers to dig into the business model along with Domain Experts!

Developers

Domain Experts

Ubiquitous Language

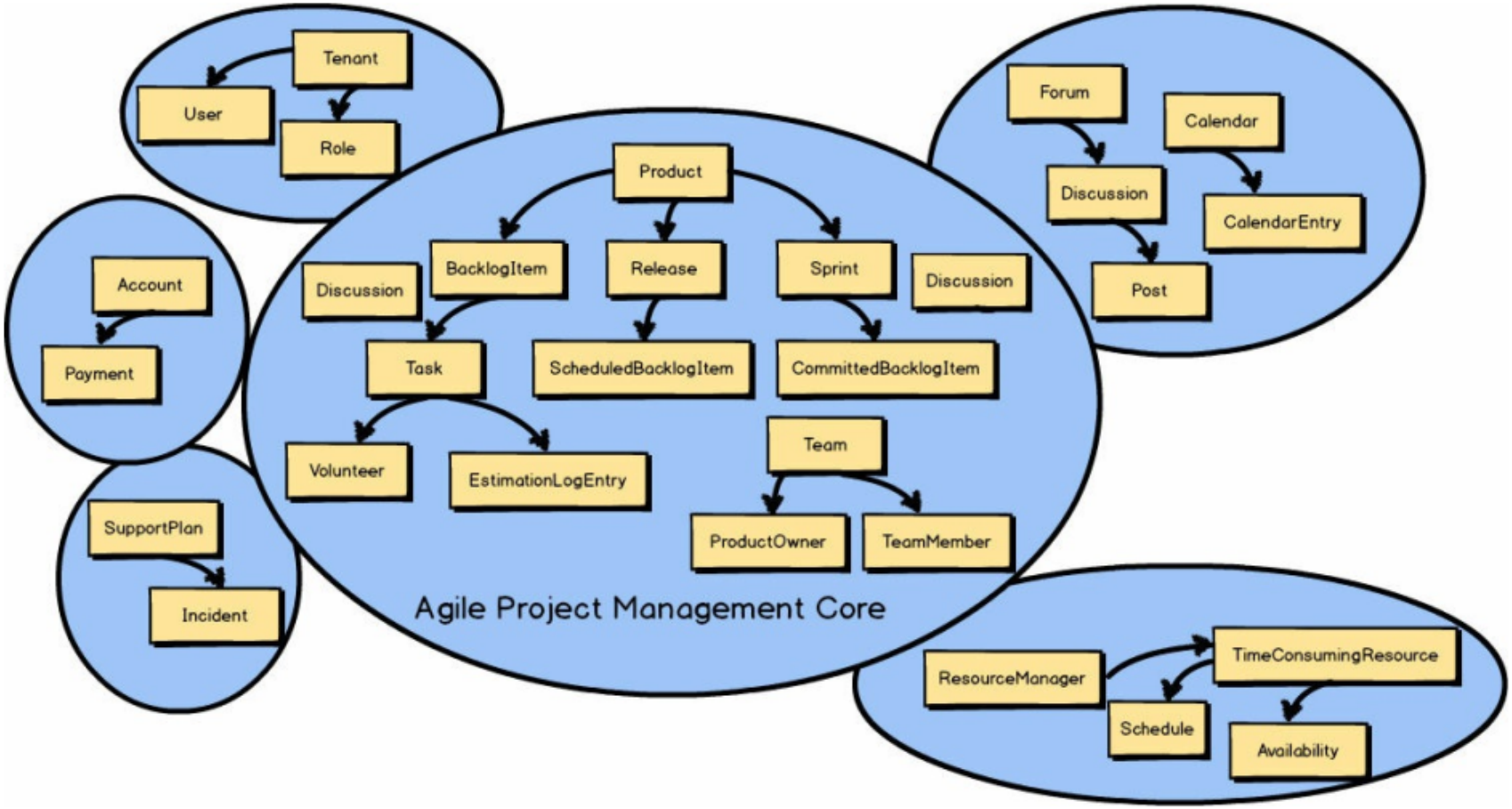
- In DDD, finding the correct Ubiquitous Language is important to find the “In Context” and the “Out Context”
- It is not about finding only nouns in the context
- It is about finding the scenarios in the context
- Scenarios are not the typical use cases that you find in the OOD

Sub Domains

- In a DDD project, there can be multiple “Bounded Contexts”.
- In a DDD project, a “Bounded Context” should align 1:1 with a single “Sub Domain”
- A “Sub Domain” is a sub-part of your overall business domain.
- There are three primary types of Sub Domains in an Enterprise.
  - Core Domain
  - Supporting Domain
  - Generic Domain

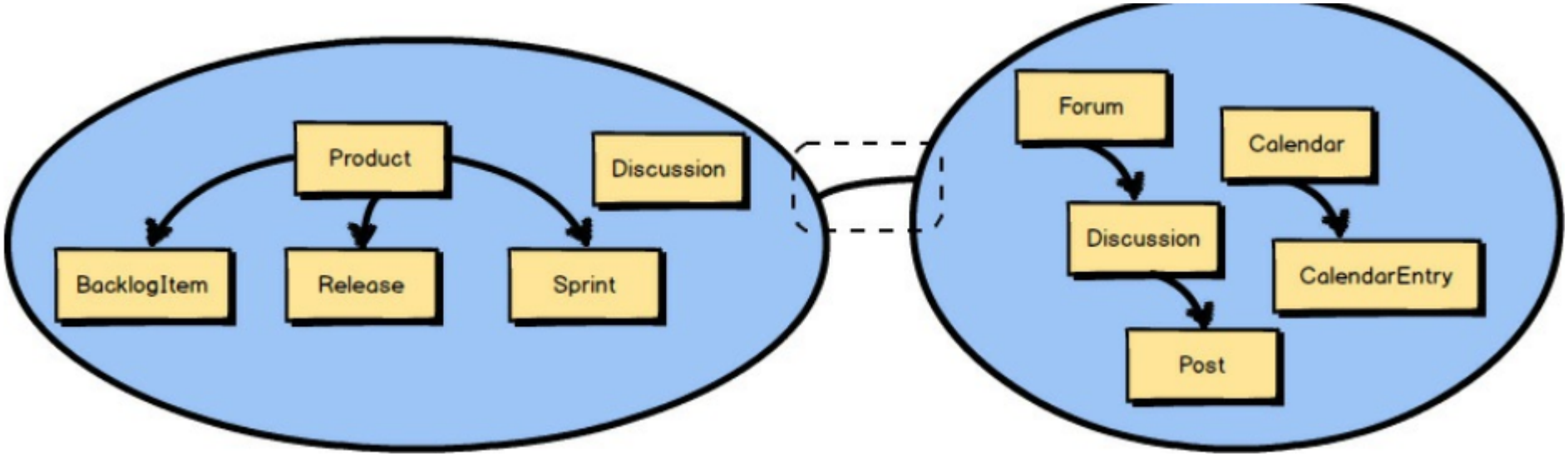
Sub Domains

- There are six sub domains in the following including the Core Domain, which is the “Agile Project Management Core”



Context Mapping

- When one Bounded Context is integrate with some other Bounded Context, this integration is known as the “Context Mapping”

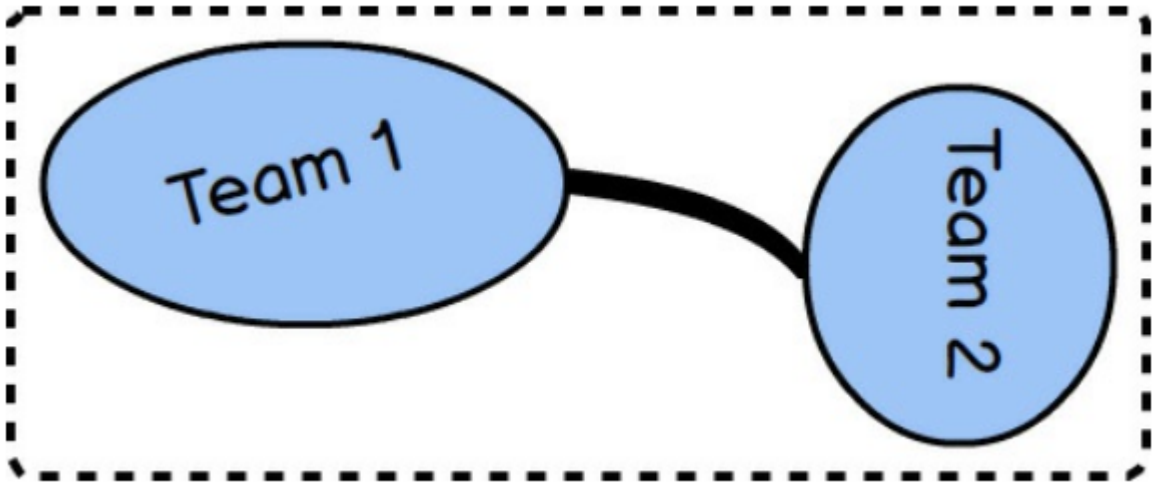


Context Mapping Types

- Partnership
- Shared Kernel
- Customer-Supplier
- Conformist
- Anti-corruption Layer
- Open Host Service
- Published Language
- Separate Ways

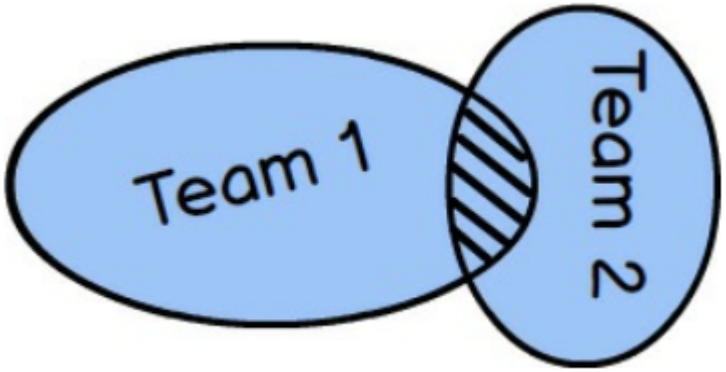
Context Mapping Types

- Partnership
  - Two teams create a partnership with a dependent set of goals
  - Two teams will succeed or fail together
  - The partnership will succeed as long as the commitment lies between both parties



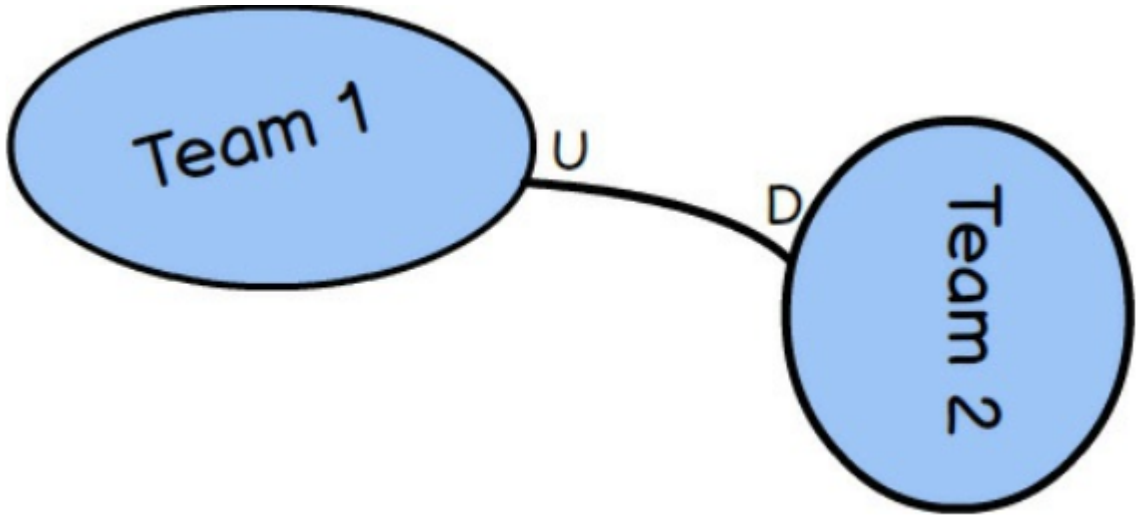
Context Mapping Types

- Shared Kernel
  - Two teams share a small but a common model
  - Often difficult to manage



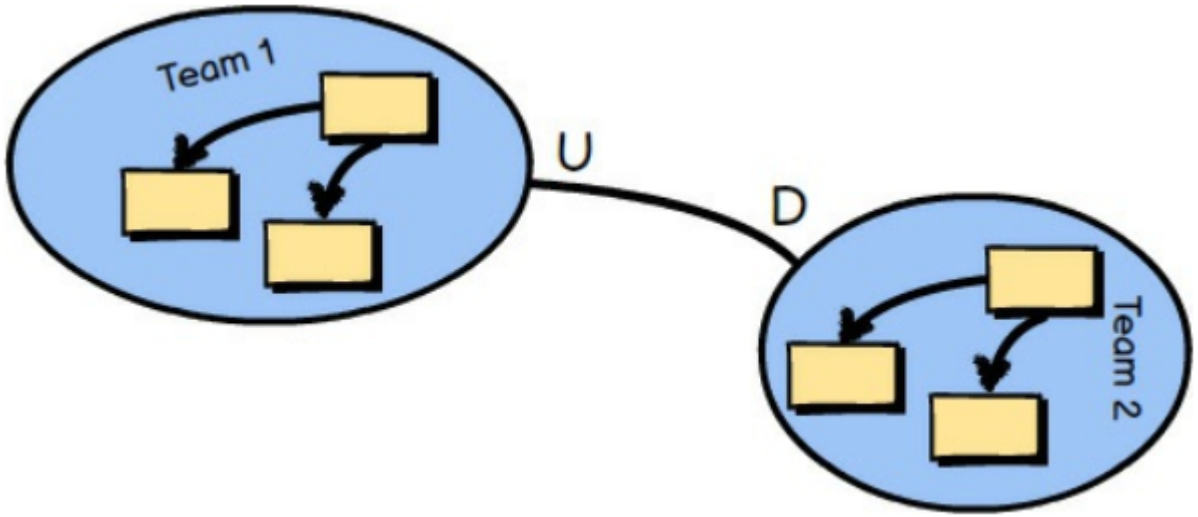
Context Mapping Types

- Customer-Supplier
  - Supplier (Upstream); Customer (Downstream)
  - Supplier decides what the customer gets



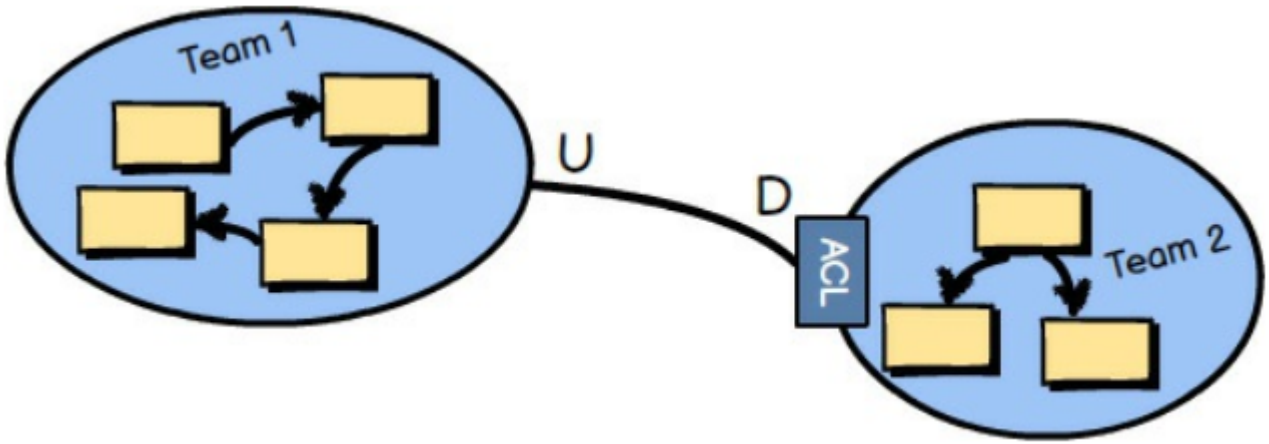
Context Mapping Types

- Conformist
  - A Customer-Supplier type
  - Downstream model should conform to the Upstream model always



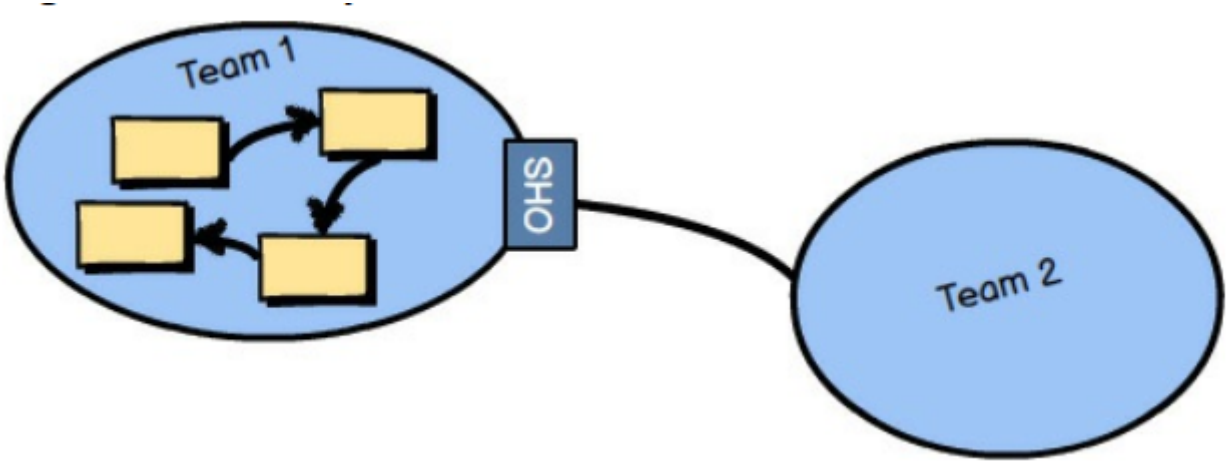
Context Mapping Types

- Anti-Corruption Layer
  - The layer isolates the Upstream model from the Downstream model and translates between the two



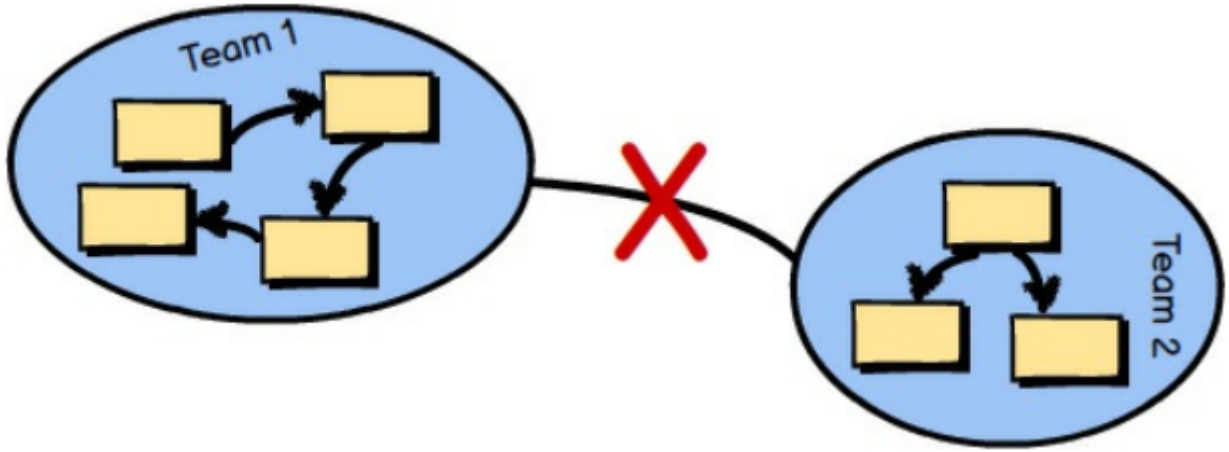
Context Mapping Types

- Open Host Service / Published Language
  - The host defines the interface as an API and it can be consumed by any customer
  - This is a much better method compared to others
  - The “Published Language” is when you are using a standard interface, such as WSDL or JSON

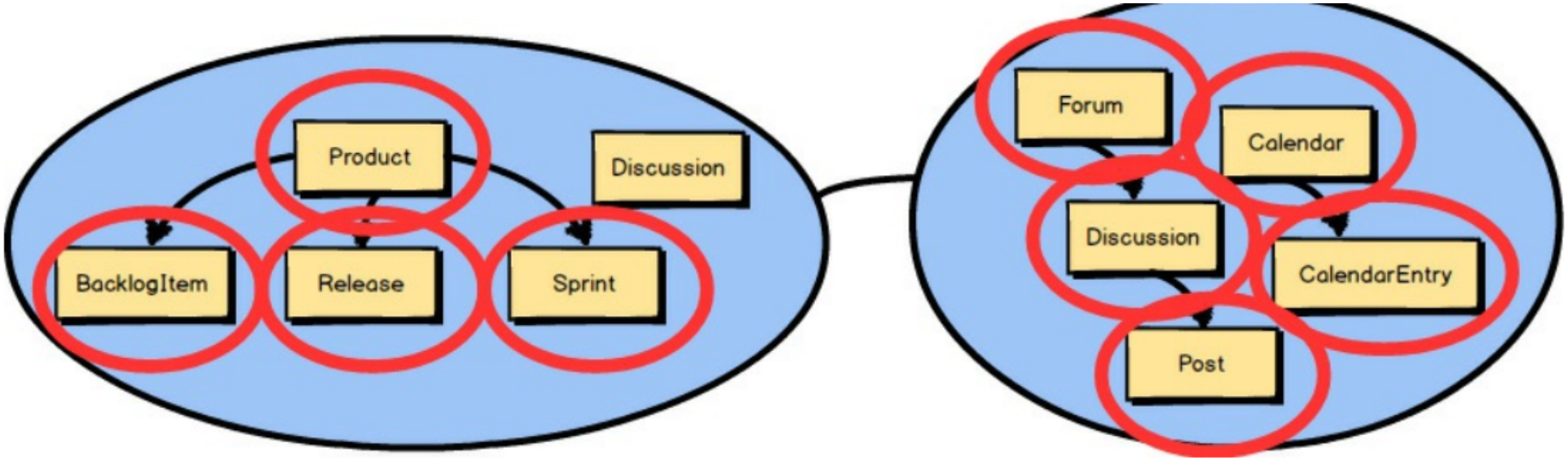


Context Mapping Types

- Separate Ways
  - There is no connection between two bounded contexts



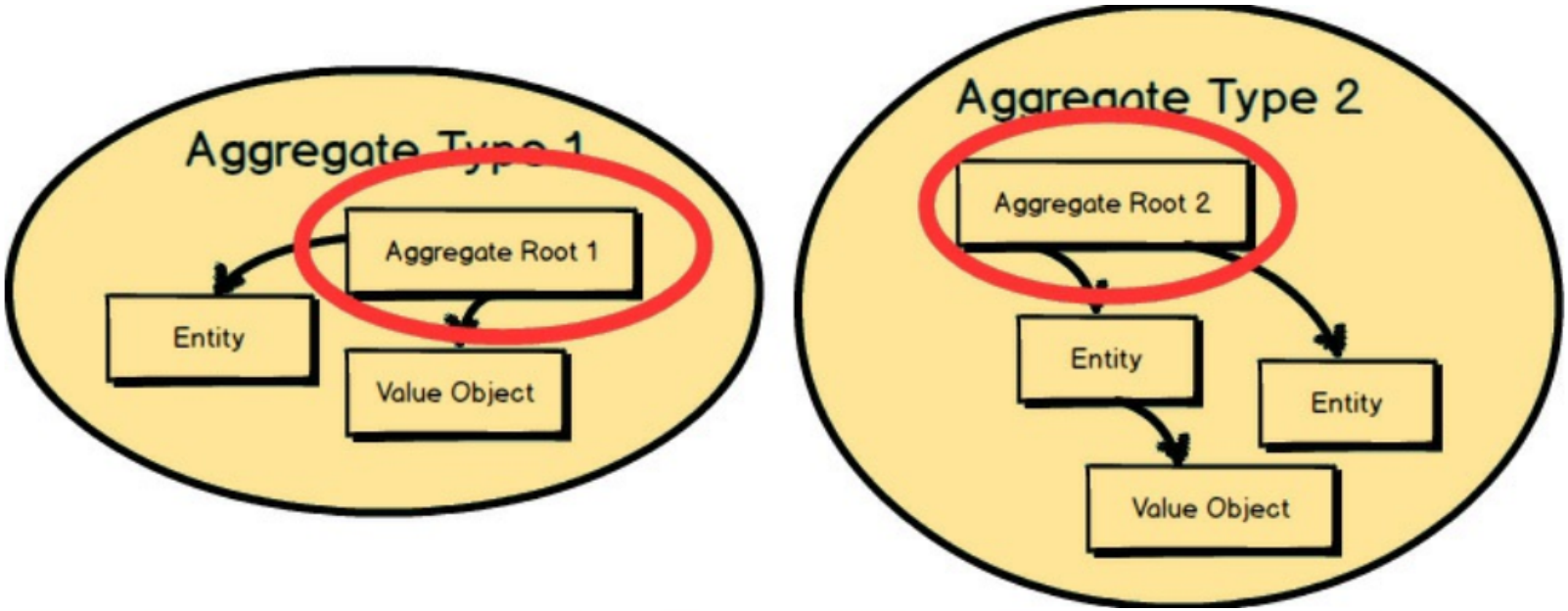
Aggregate



- There are two Bounded Contexts here
- Each Bounded Context has got multiple Aggregates
  - e.g. The “Product”, “Backlog Item”, “Release”, “Sprint” are aggregates on the left Bounded Context

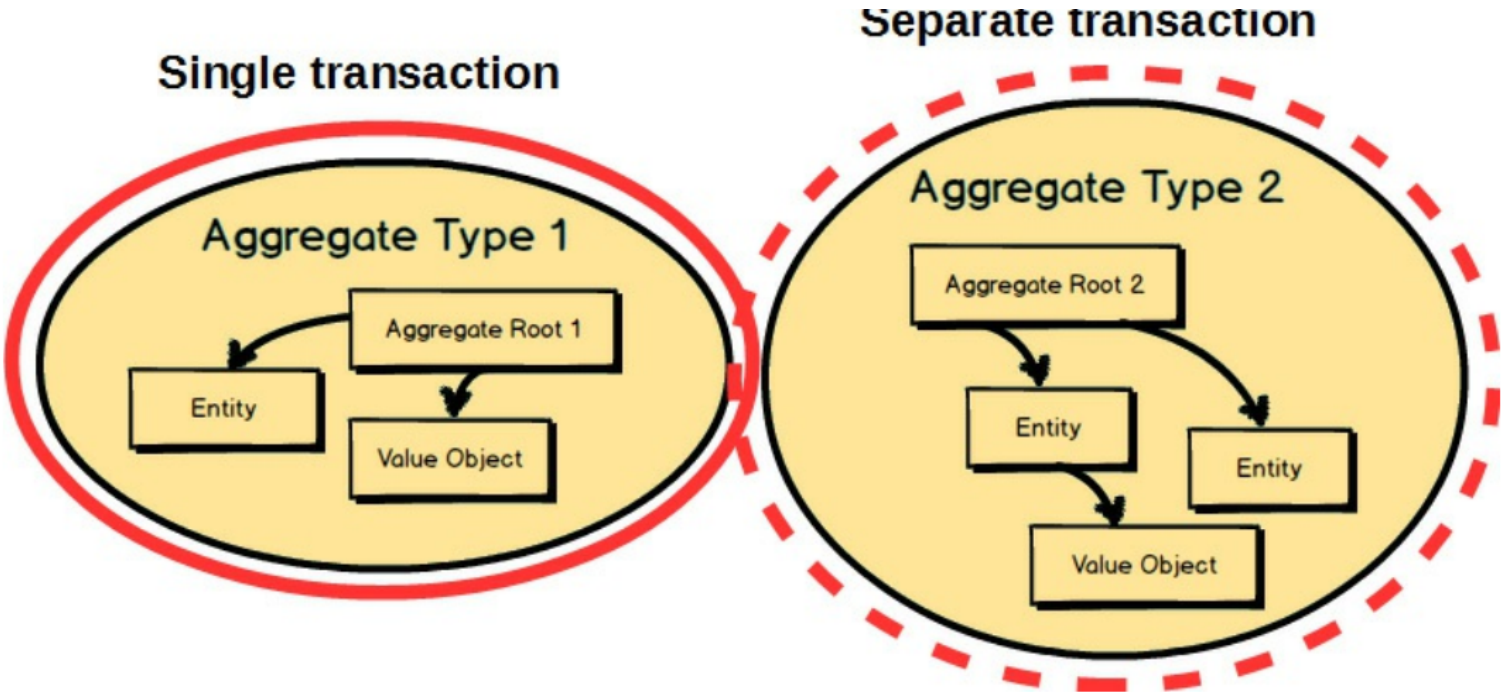
Aggregate

- Aggregate is a pattern in DDD, which is a cluster of domain objects that can be treated as a single unit
- One of its component objects be the “aggregate root”
- The aggregate root can ensure the integrity of the aggregate as a whole



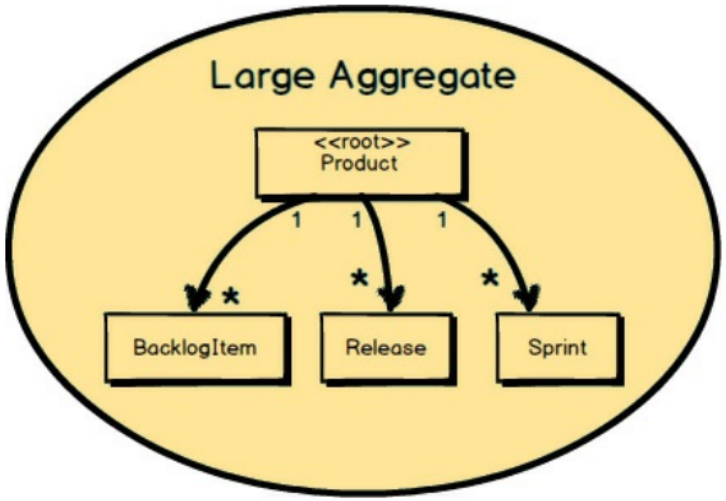
Aggregate

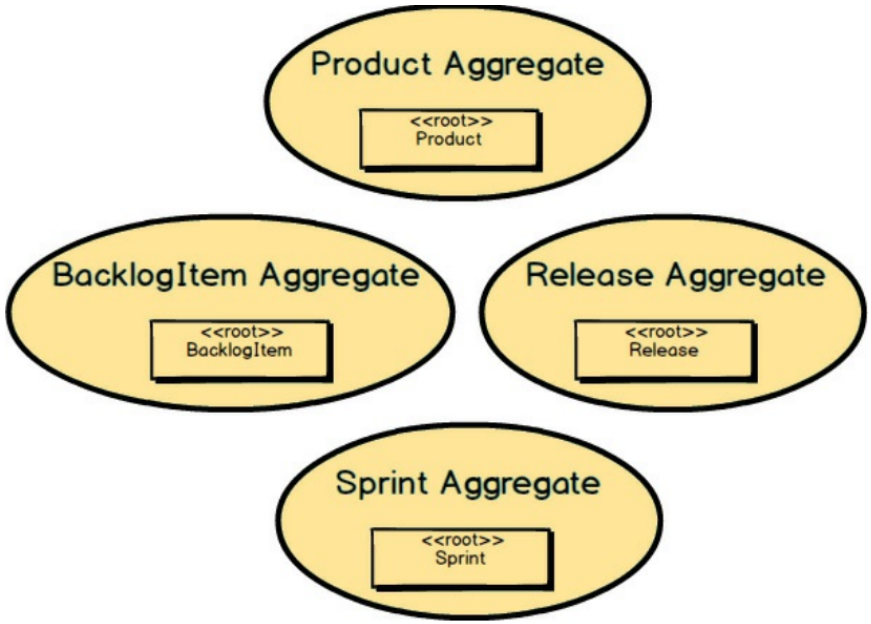
- Each aggregate forms a transactional consistency boundary
- Only one aggregate should be committed in a single transaction (The general rule in the aggregate design)
- Any other aggregate should be committed in a separate transaction



Aggregate

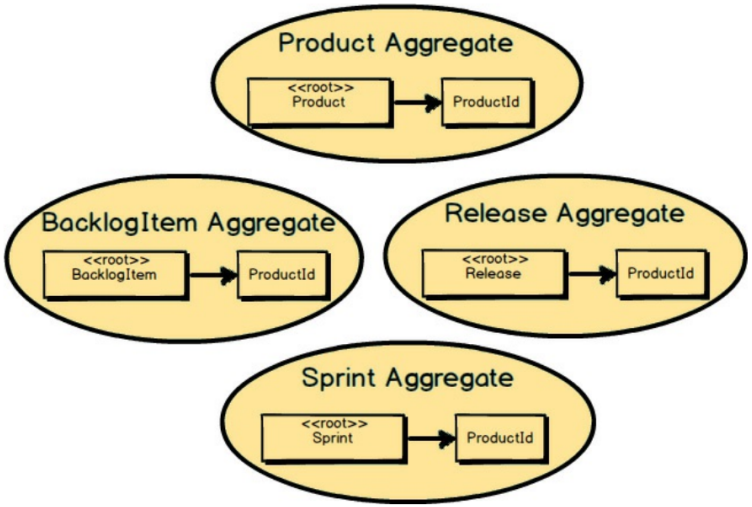
- Design Small Aggregates and eliminate large aggregates
- Smaller aggregates are easier to work on and scalable
- Follows the SRP (Single Responsibility Principle)





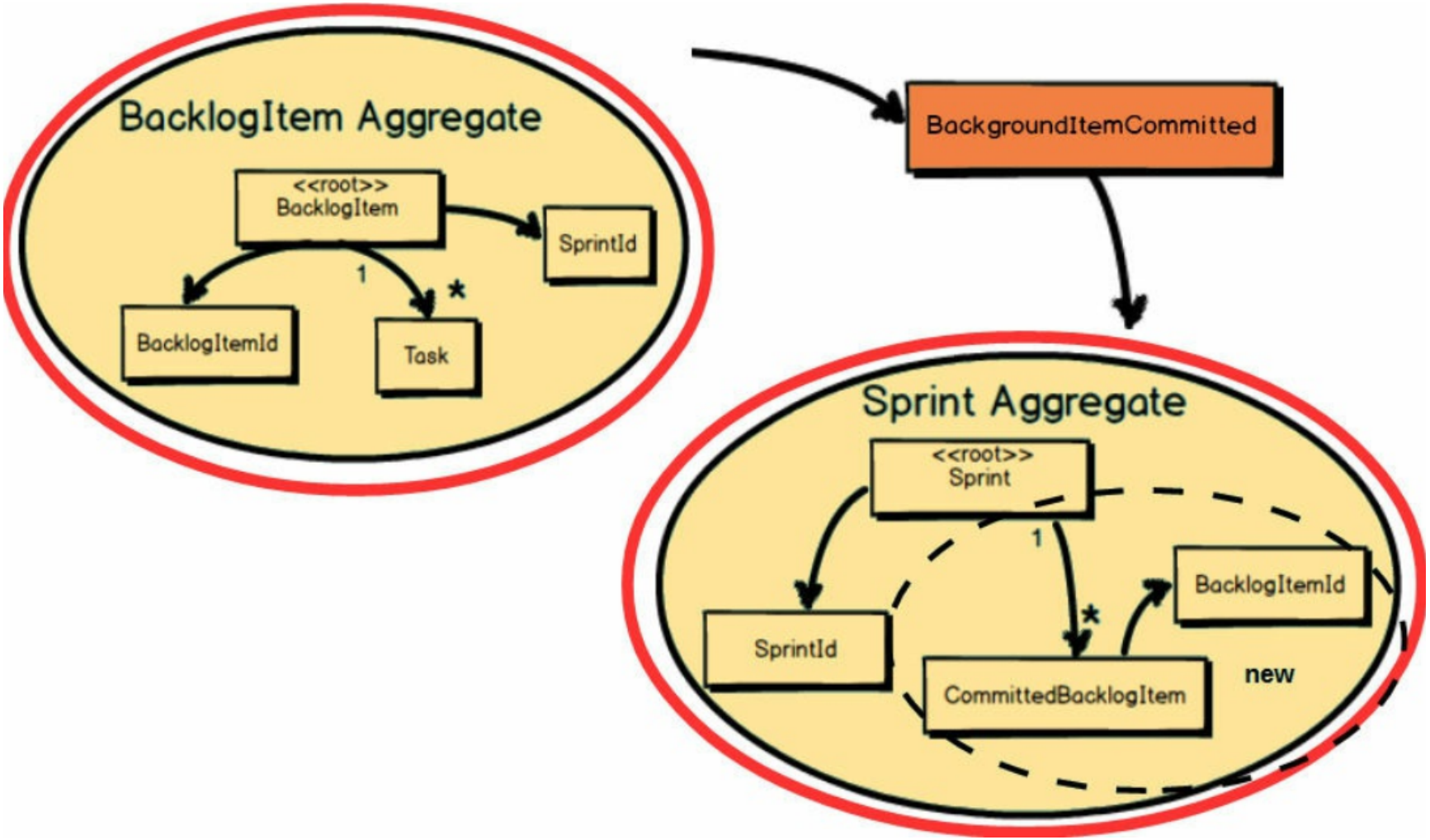
Aggregate

- Reference other aggregates by Identity only. Helps to keep Aggregates small and prevents reaching out to modify multiple aggregates in the same transaction
- Lower memory requirement and quicker loading from the persistent store
- Facilitates multiple persistence stores such as Relational Databases, Document Databases, Key Value Stores



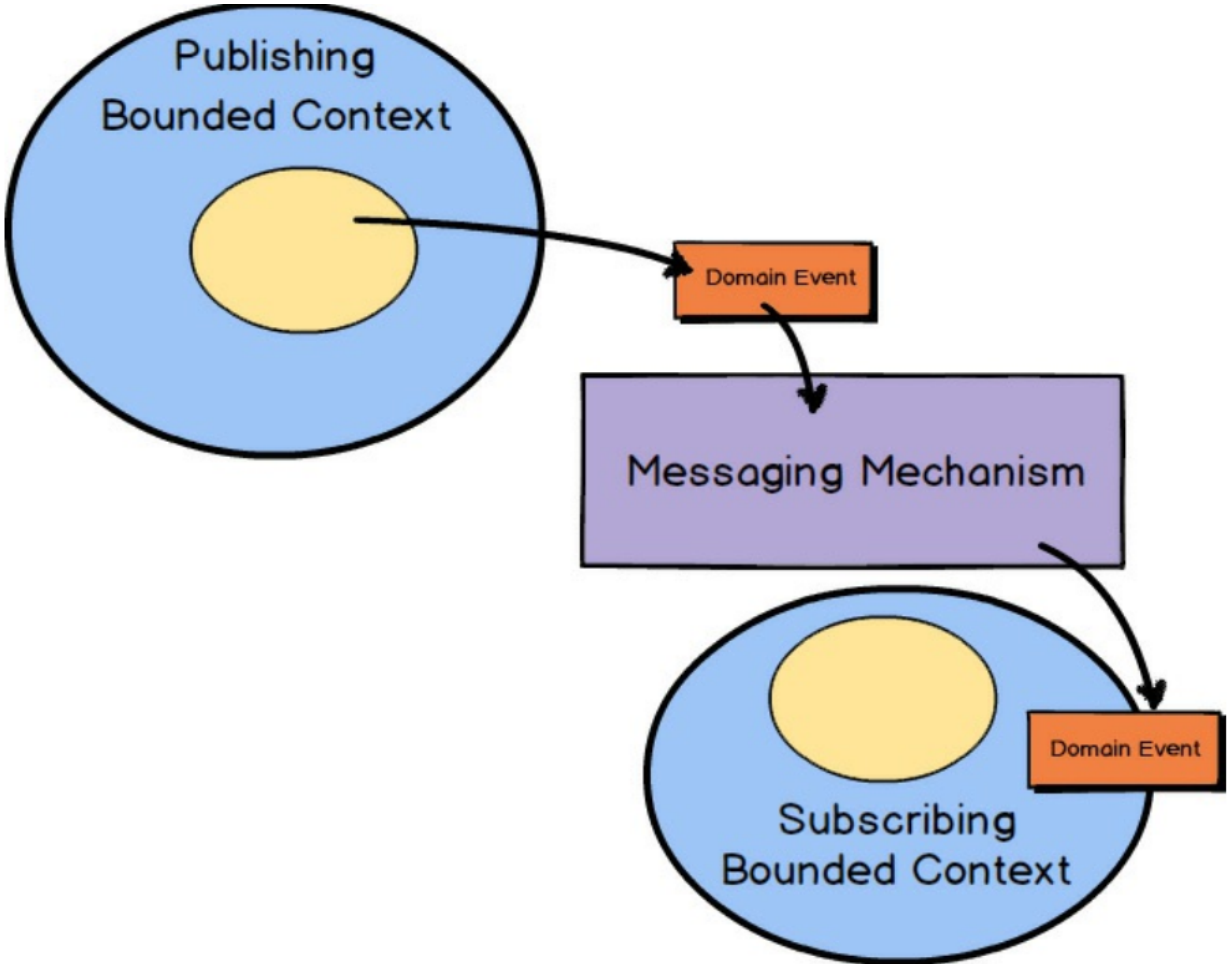
Aggregate

- Updating the other Aggregates using “Domain Events”



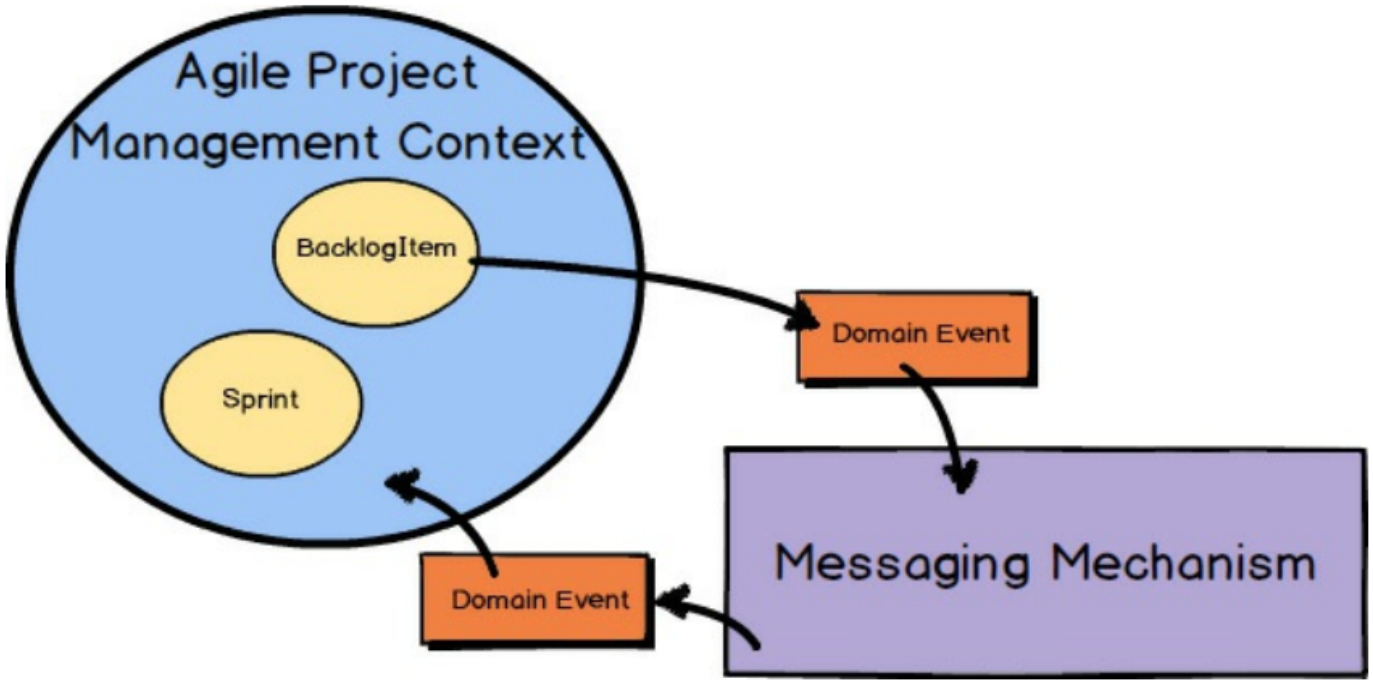
Aggregates and Domain Events

- Updating the other Aggregates using “Domain Events” and a “Messaging Mechanism”



Domain Events

- Though both Aggregates are in the same Bounded Context, it is easy to use the same Messaging Middleware to do the eventing, which probably used for messaging between Bounded Contexts



The Hexagonal Architecture

(Ports and Adapters)

- This is known as the fundamental architecture in DDD.
- It is an “Architecture Style”, also known as the “Ports and Adapters”
- An extension to the traditional “layered architecture”, which has one directional thinking and mostly all the dependencies go in one direction
- It talks about the distinction between the softwares’ inside and outside parts
  - Inside Part: the Domain model
  - Outside Part : UI, Database and Messaging

The Three Components

- There are 3 main components
  - Domain Model
  - Ports
  - Adapters

- Domain Model
  - A conceptual model, a representation of meaningful concepts to the domain that need to be modeled in software

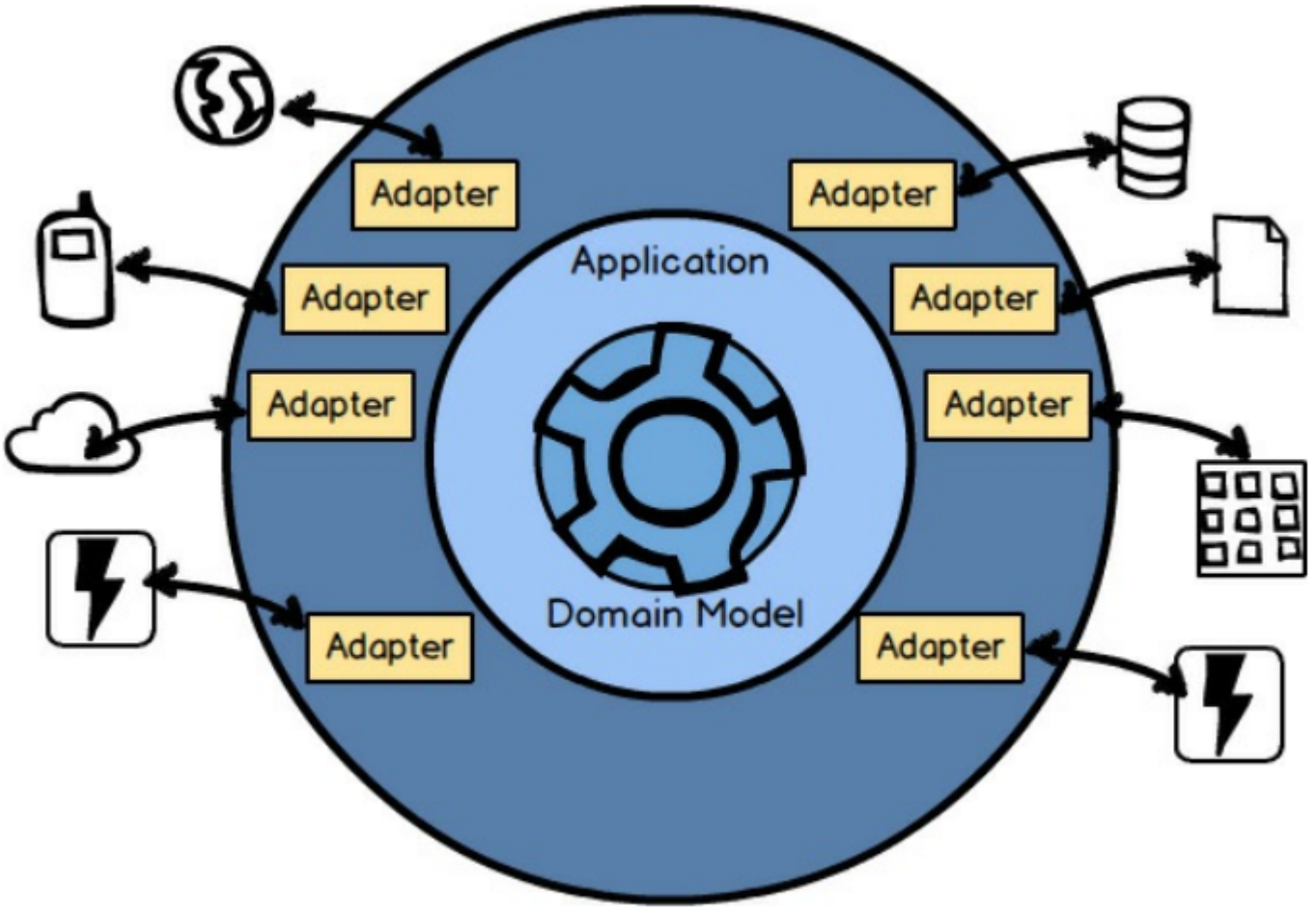
The Three Components

- Ports
  - Primary Ports
    - Main APIs/ Interfaces of the application.
    - They are called by the Primary Adapters
  - Secondary Ports
    - APIs/ Interfaces for the Secondary Adapters
    - They are called by the Core Logic

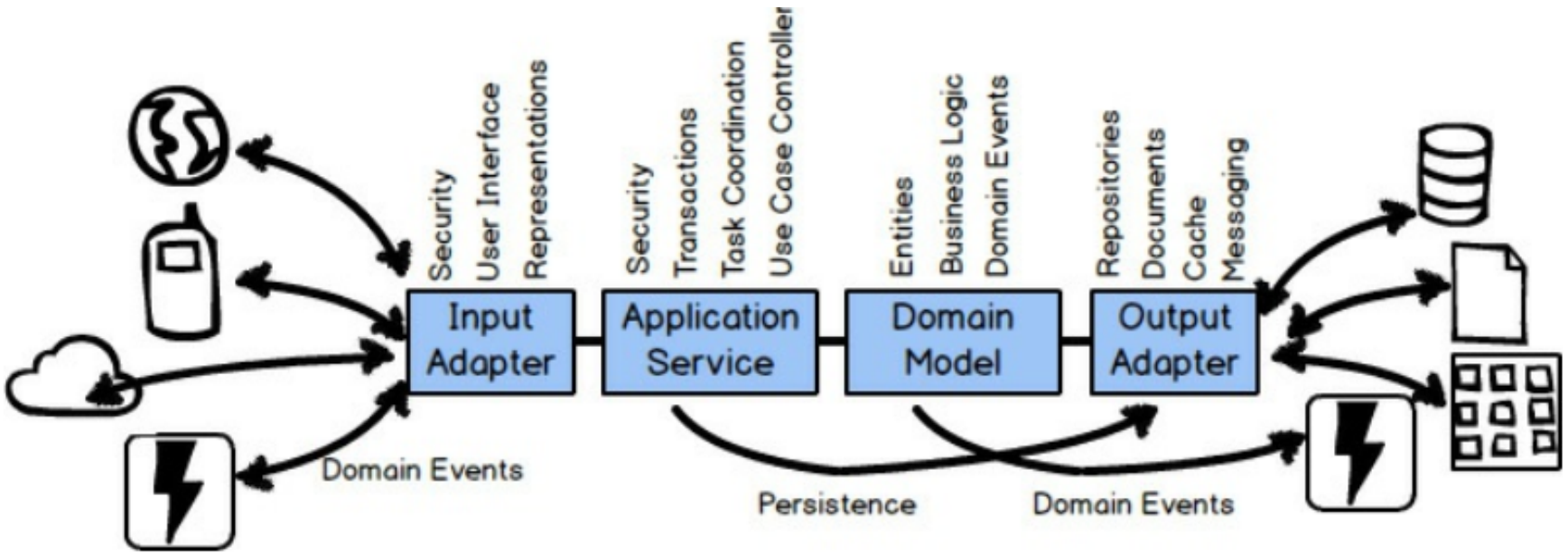
The Three Components

- Adapter is a bridge between the application and the port / service
- Adapters
  - Primary Adapter – A piece of code between the user and the core logic and interact with the Primary port
  - Secondary Adapter – Interacts with the Secondary port

The Architecture



The Architecture



References

- Domain Driven Design Distilled – By Vaughn Vernon
- https://www.thoughtworks.com/insights/blog/domain-driven-design-services-architecture
- https://dzone.com/refcardz/getting-started-domain-driven

Thank You

Crishantha Nanayakkara

Director – Technology, Auxenta
email: crishantha@auxenta.com
Twitter: @crishantha
Blog : www.crishantha.com
LinkedIn: http://linkedin.com/in/crishantha