

Course: Domain Driven Design & Microservices for Architects

Section: Bounded Context & Relationships

<http://acloudfan.com/>

Pragmatic Paths Inc © 2021

Contact: raj@acloudfan.com

Discount Link to course:

<https://www.udemy.com/course/domain-driven-design-and-microservices/?referralCode=C5DCD3C4CC0F0298EC1A>

Bounded Context Relationships



Bounded Contexts are independent but NOT *isolated* from other BCs around them; models in BCs collaborate to fulfil requirements of a system

Symmetric Relationship

- 2 BC that are dependent on each other



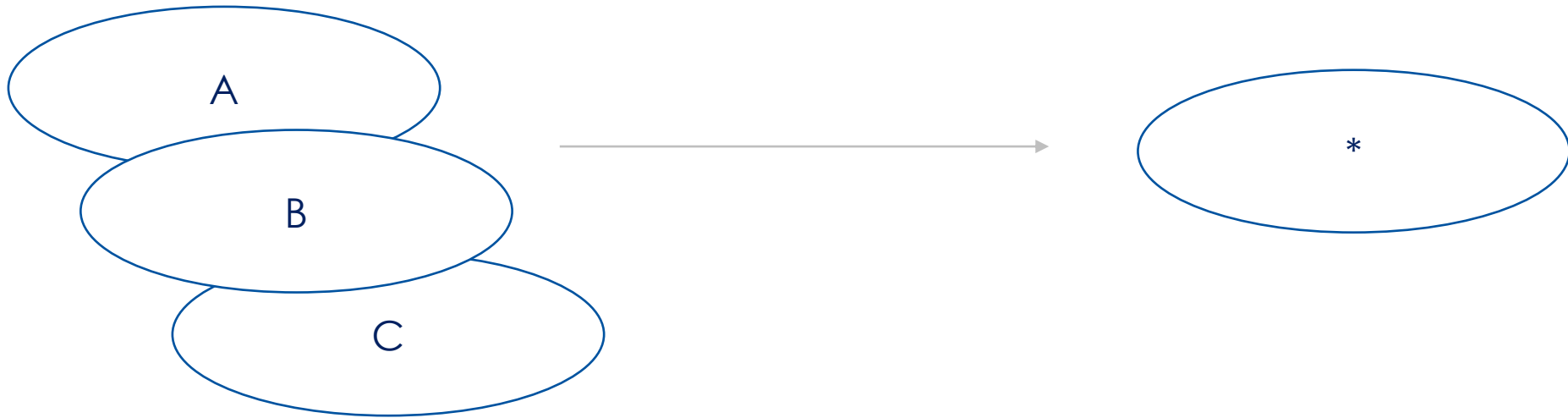
Asymmetric Relationship

- One BC depends on another BC



One to Many Relationship

- Multiple BC depends on One BC



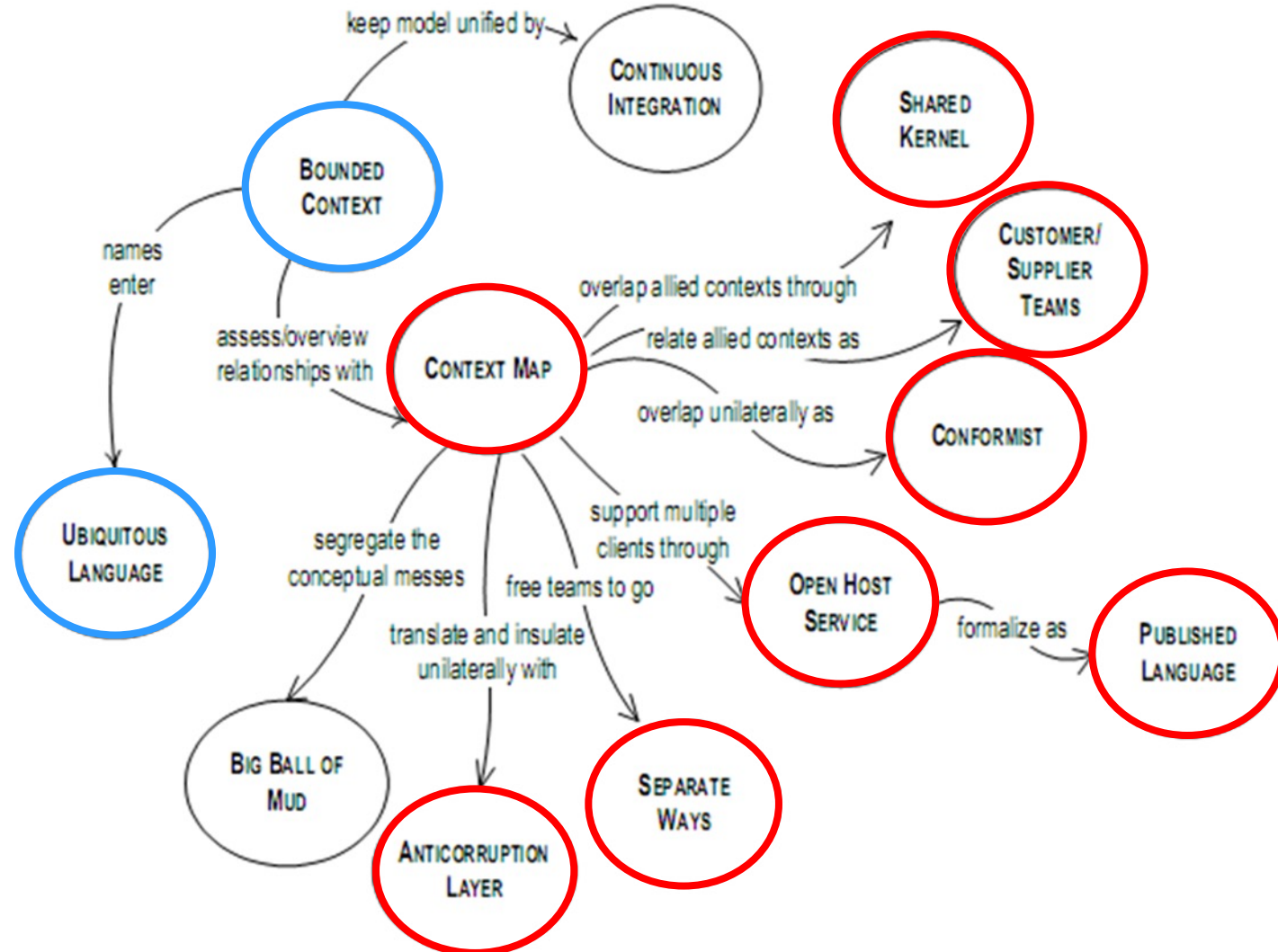
Strategic Relationship patterns

- Defines the dependency relationship between BCs



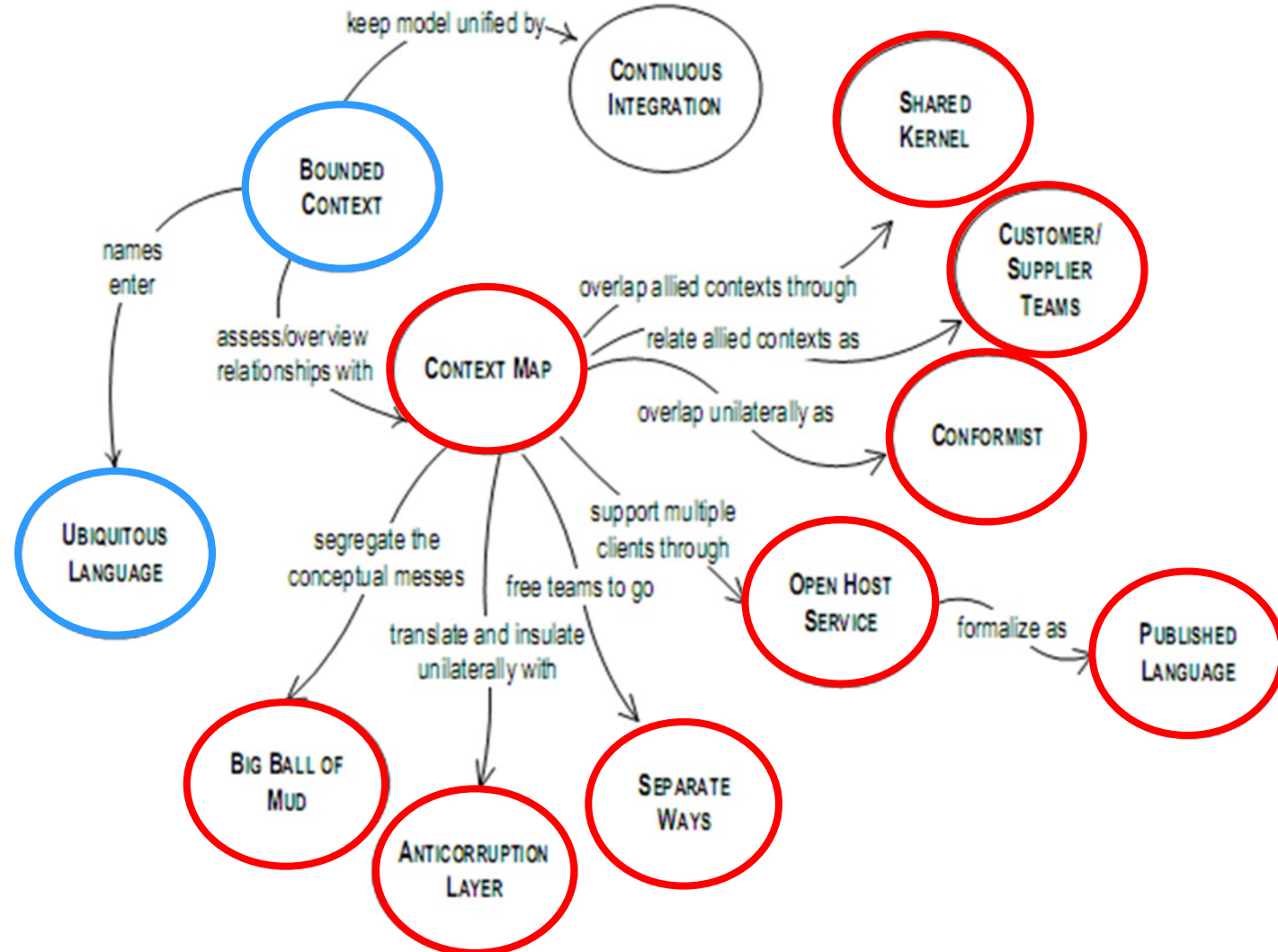
Bounded Context Relationships

Relationship Patterns



Bounded Context Relationships

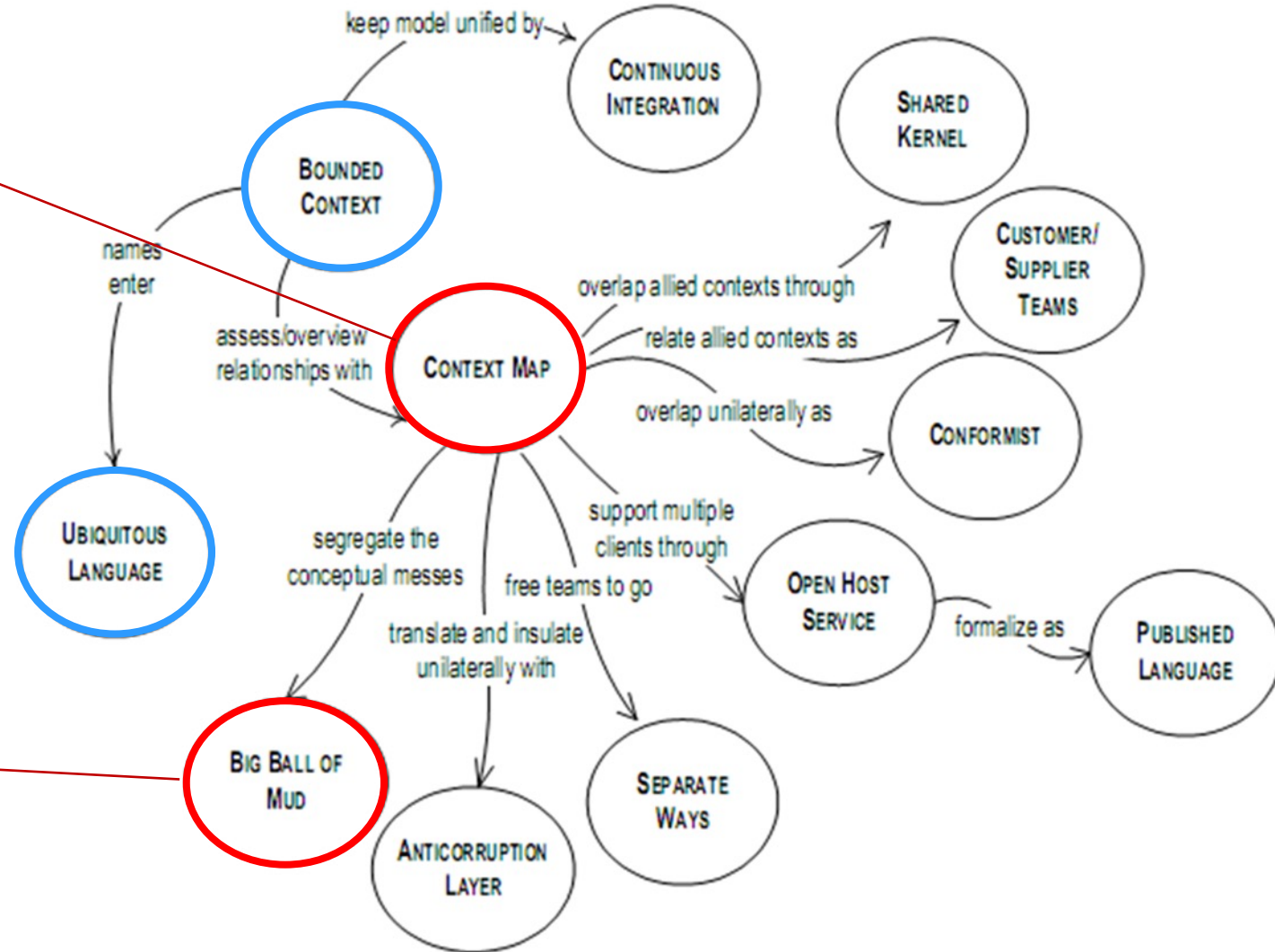
DDD Strategic Patterns




Bounded Context Relationships

Visual Representation

Anti Pattern



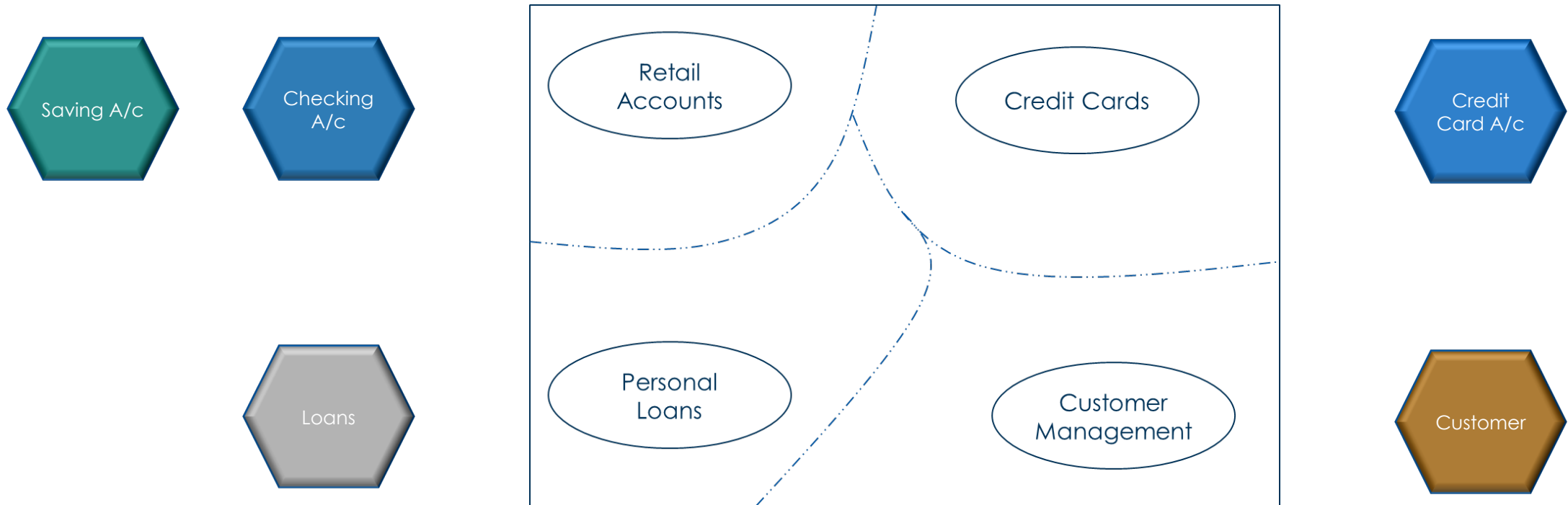
- 
- 1 Big Ball of Mud
 - 2 Context Maps
 - 3 Symmetric Relationship : Separate ways , Partnership, Shared Kernel
 - 4 Asymmetric Relationship: Customer-Supplier, Conformist, Anti Corruption Layer
 - 5 One-to-Many Relationship: Open Host Service, Published Language

BC : Maintaining the Integrity

- BC A will refer to models in BC B
- This will lead to contextual confusion !!!

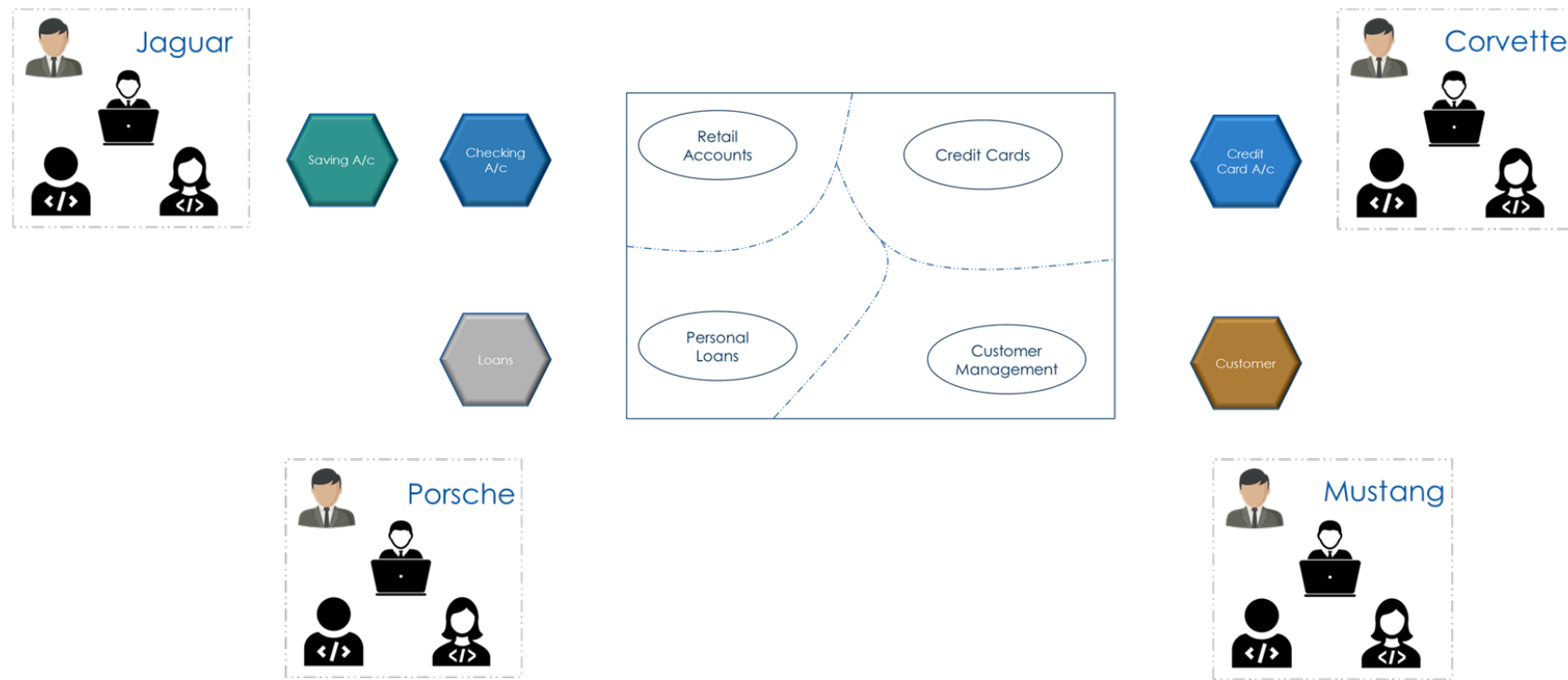


Microservices development



Maintain the Integrity of models across Microservices

To ensure teams can work on Microservices Independently



Relationship between BCs

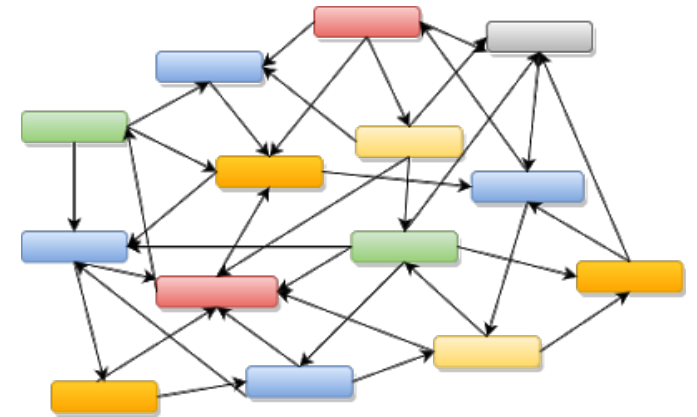
Managing the Bounded Context Dependencies



Unmanaged BC Relationship

Leads to Big Ball of Mud

- Haphazardly structured model
- Leads to spaghetti-code
- Mostly created by unregulated growth | fixes over time



Big Ball of Mud refers to an Anti-Pattern

BC dependencies

Relationships need to be managed otherwise:



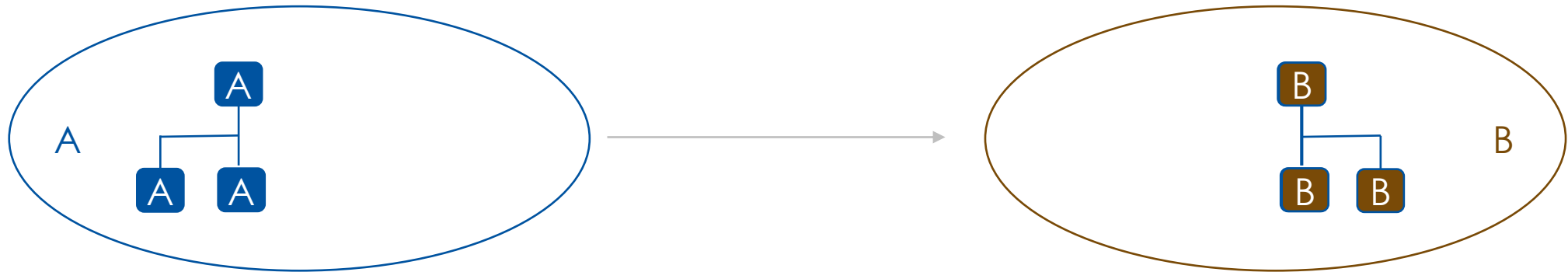
Loss of model integrity



Loss of Team's ability to operate independently

Bounded Context - Dependencies

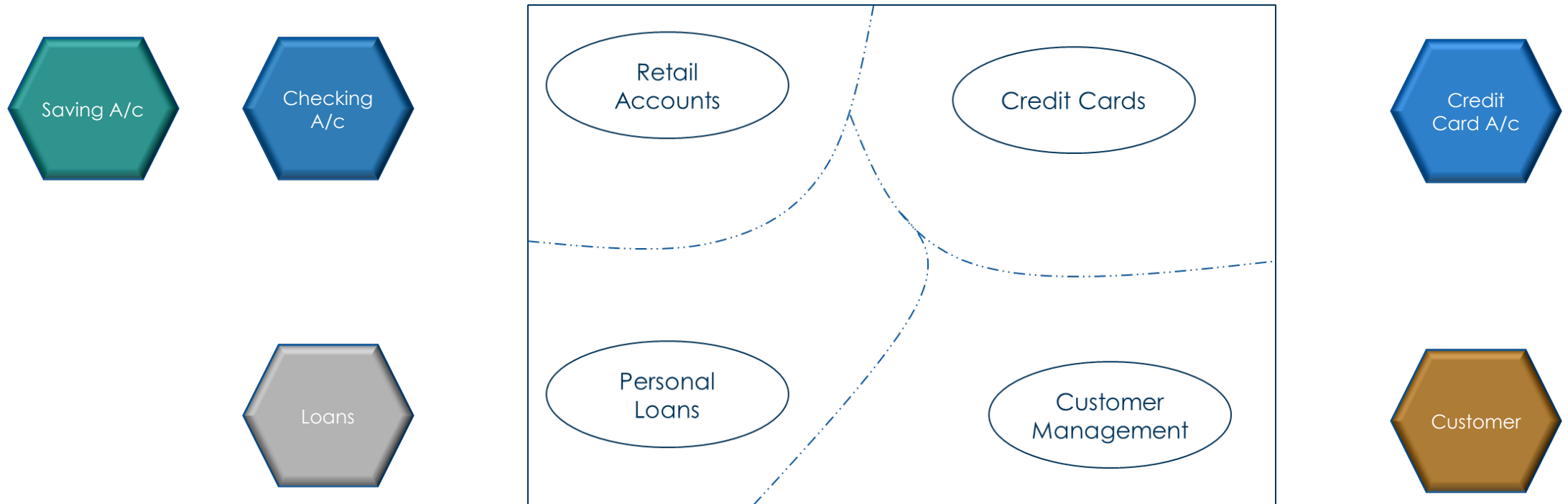
Negatively impacts the model's integrity



Loss of model integrity

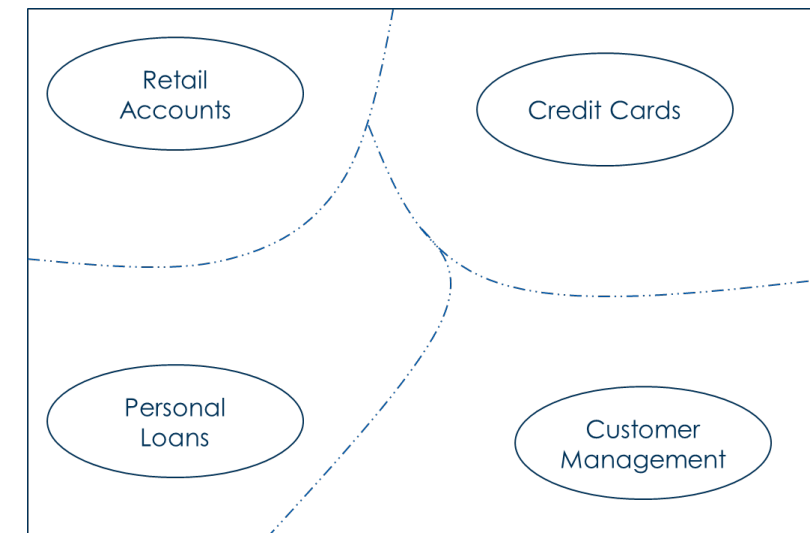
Bounded Contexts => Microservice(s)

Each BC is translated into one or more microservices



Bounded Context - Dependencies

Translates to *Microservices dependencies*



Bounded Context - Dependencies

Translates to *Microservices dependencies*



Loss of agility



Loss of Team's ability to operate independently

Manage the BC Relationships

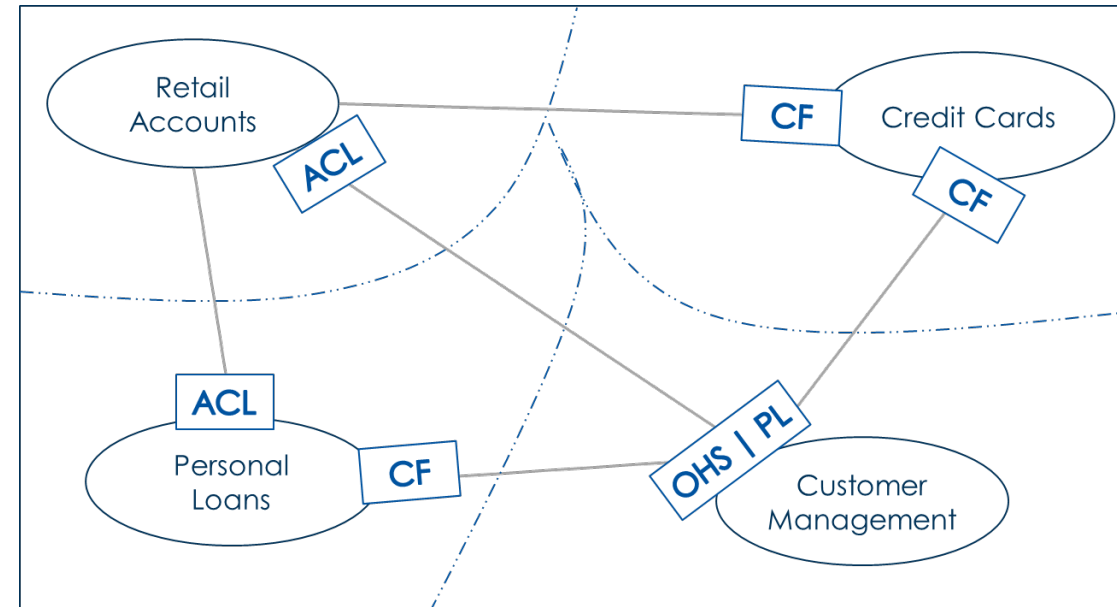
Microservices teams use DDD patterns

- Remember "Big Ball of Mud" is an anti-pattern 😊
- Document the relationships using Context Maps

Context Maps



A visual representation of the system's Bounded Contexts and relationships (a.k.a. integrations) between them



Benefits of Context Mapping

- Easy to understand the big picture
- Helps in understanding the inter dependencies
- Gauge the Level of Collaboration needed between teams
- Helps with refinement of the BCs | models

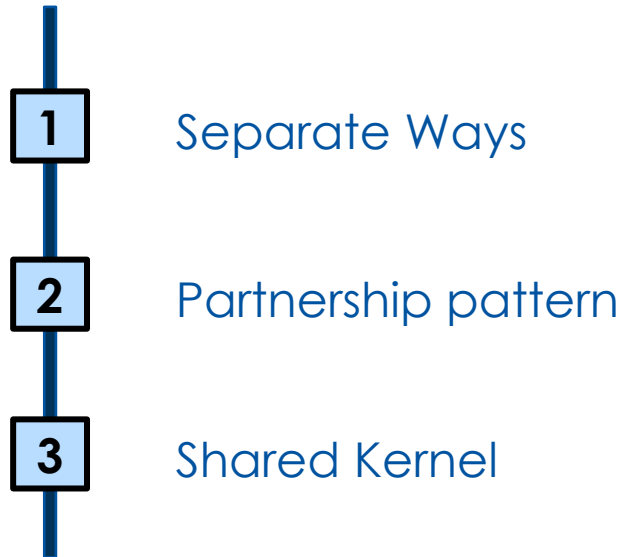


Quick Review

- Avoid creating *Big Ball of Mud*
- Use well defined relationship patterns in your models
- Document the relationships using Context Maps

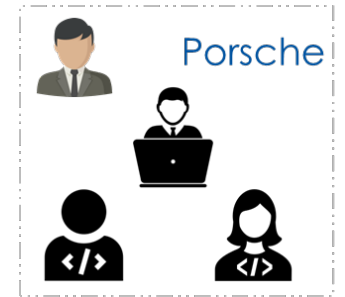
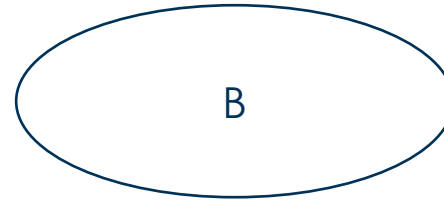
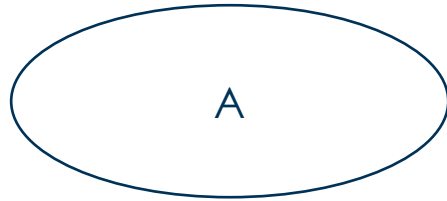
Symmetric Relationship

Independent | Interdependent Bounded Contexts



No Relationship between BCs

The BCs are truly *Independent* of each other



Teams work autonomously

Opportunity to Re-use may exist but "*loss of autonomy*" is a concern !!!

Separate Ways Pattern

Separate Ways Pattern

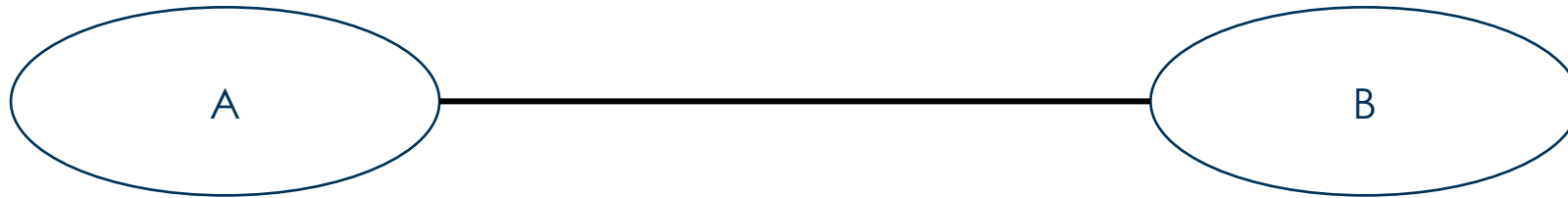
Independent applications | services for the BCs



Teams work at their own pace to meet the business goals !!!

Symmetric Relationship | Bidirectional Dependency

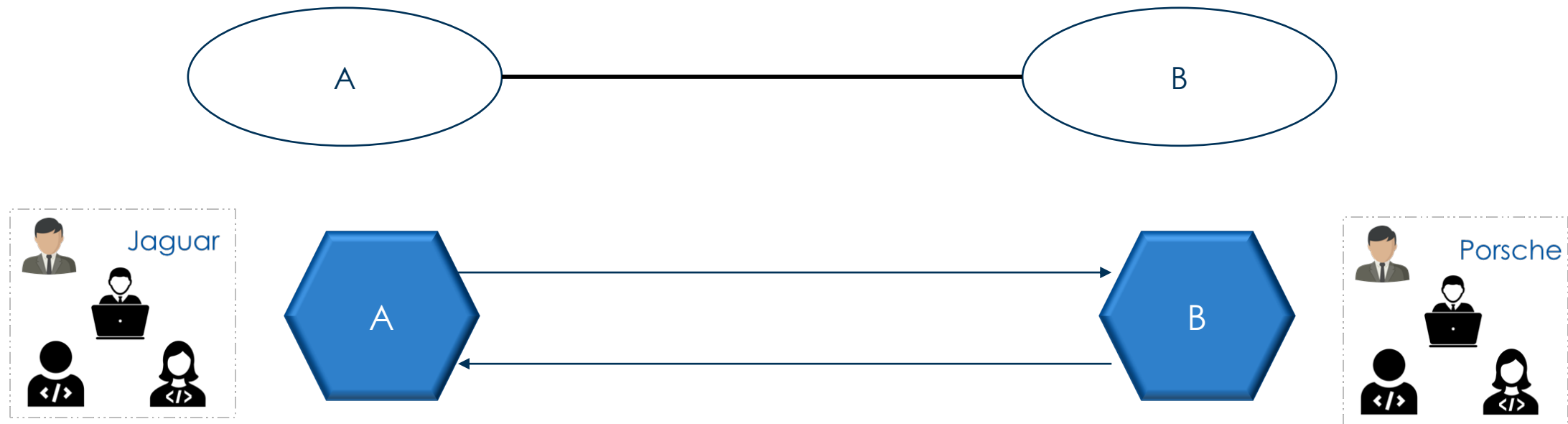
The BCs are dependent on each other



High levels of COUPLING between the BCs

Partnership Pattern

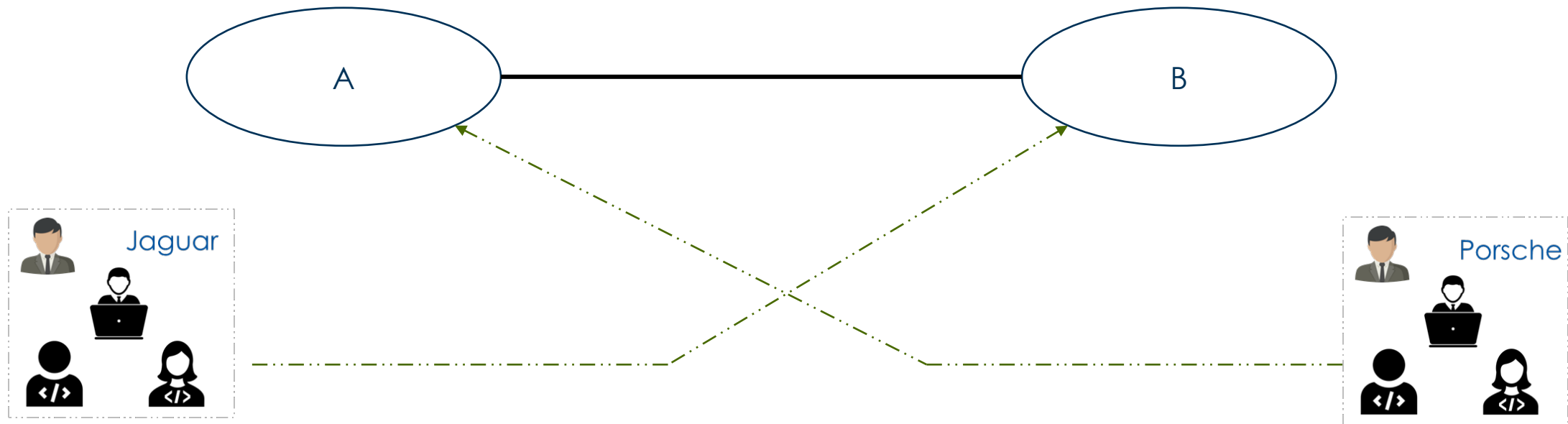
Translates into Microservices that have mutual dependencies



Teams are no more independent ☹️

Partnership Pattern

Translates into Microservices that have mutual dependencies

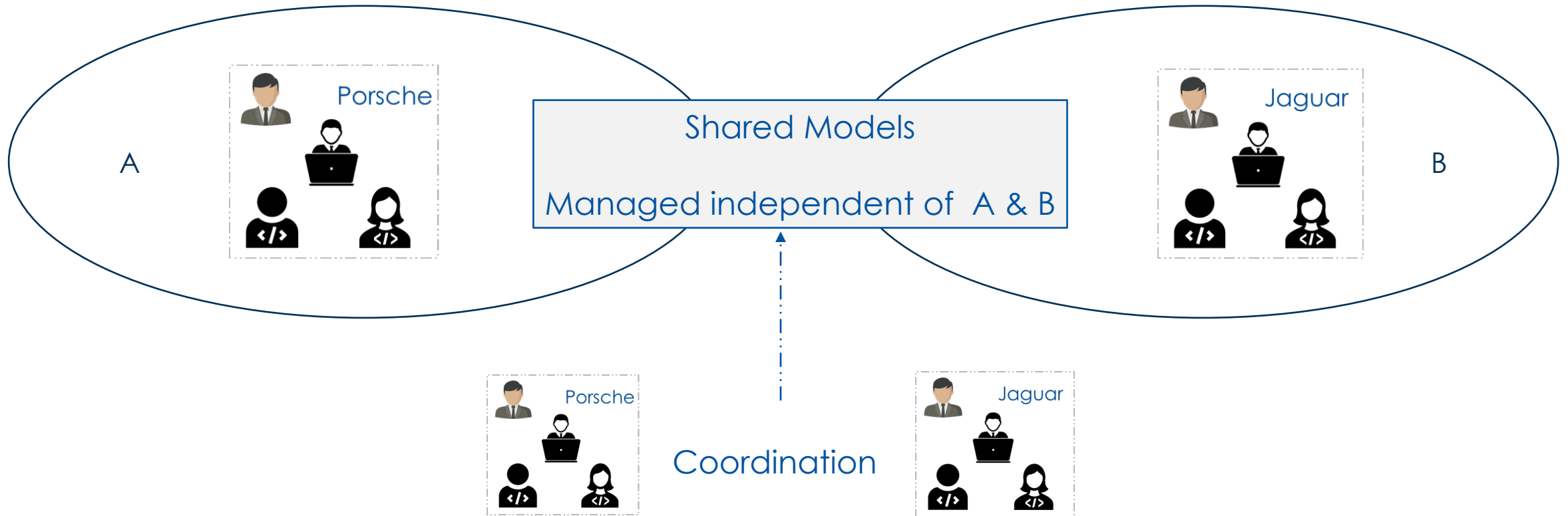


Learn Business Models & Ubiquitous Language!!

Teams are no more independent ☹

Solving to Partnership

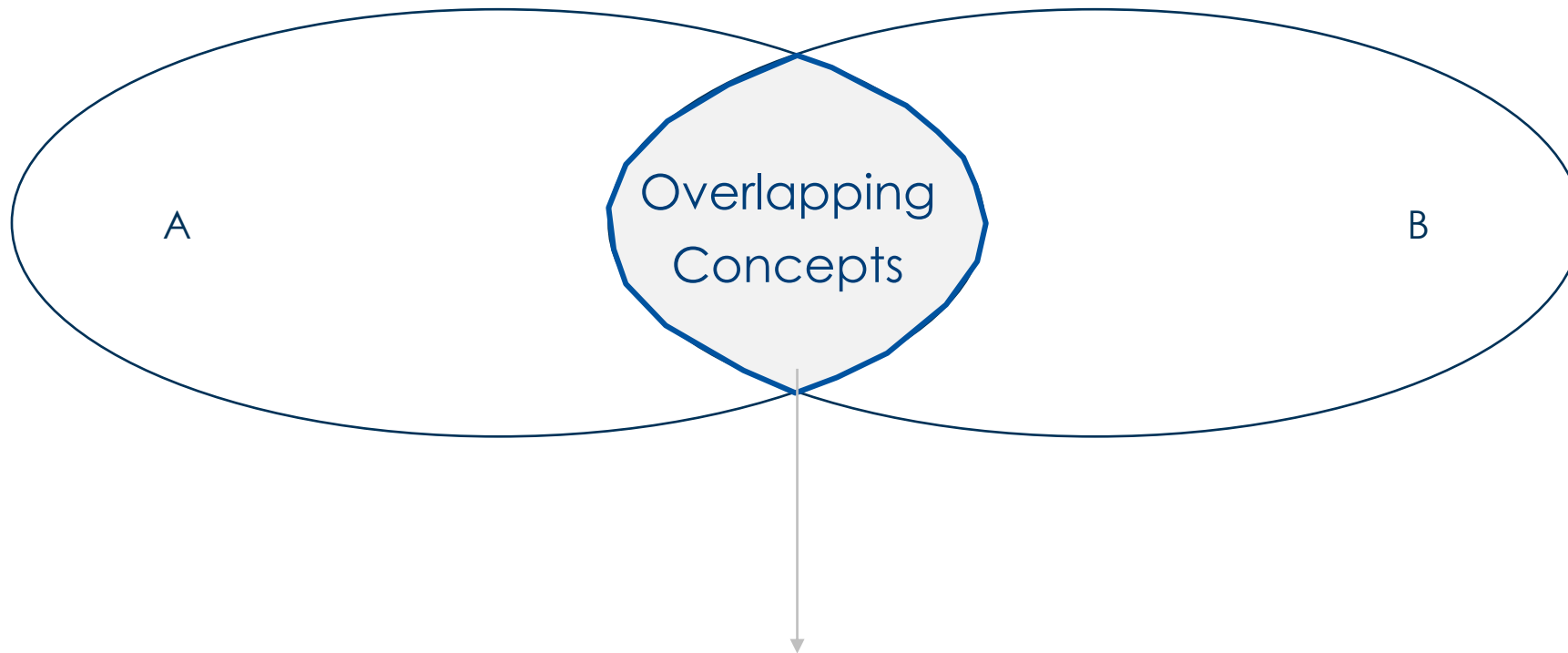
Demarcate the boundaries for shared models



Shared Kernel Pattern

Pattern: Shared Kernel

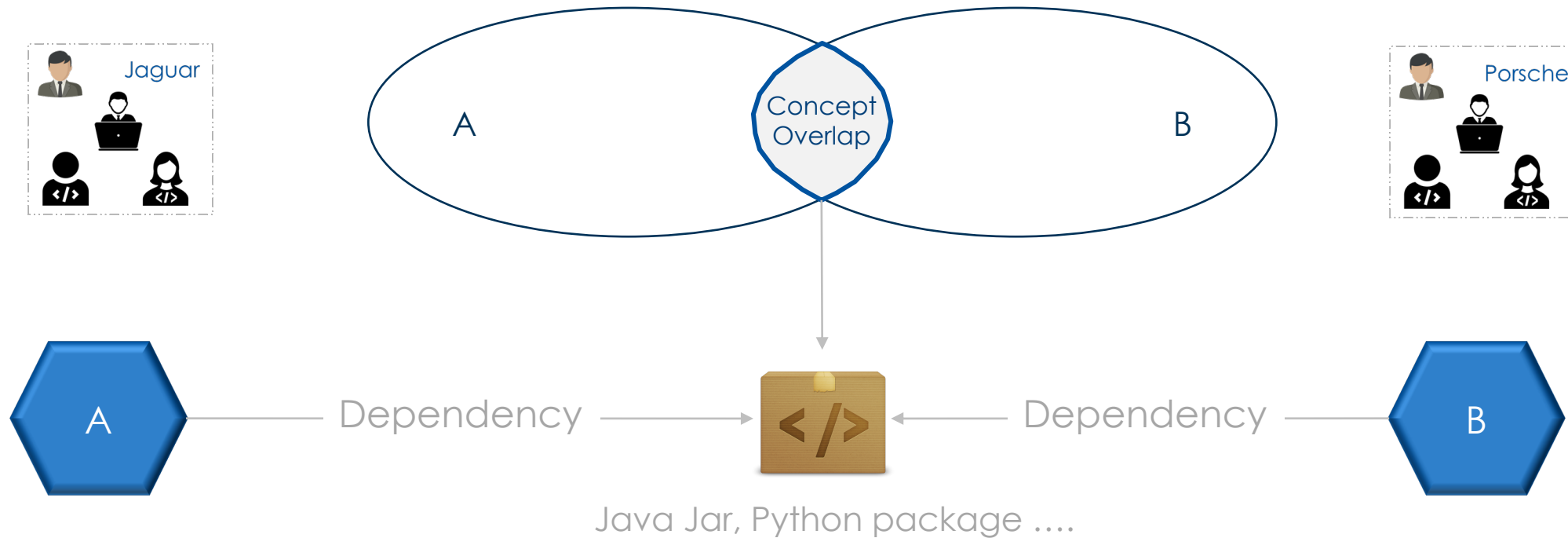
Create a common model for the overlapping concepts



Shared domain model & business language for 'A' and 'B'

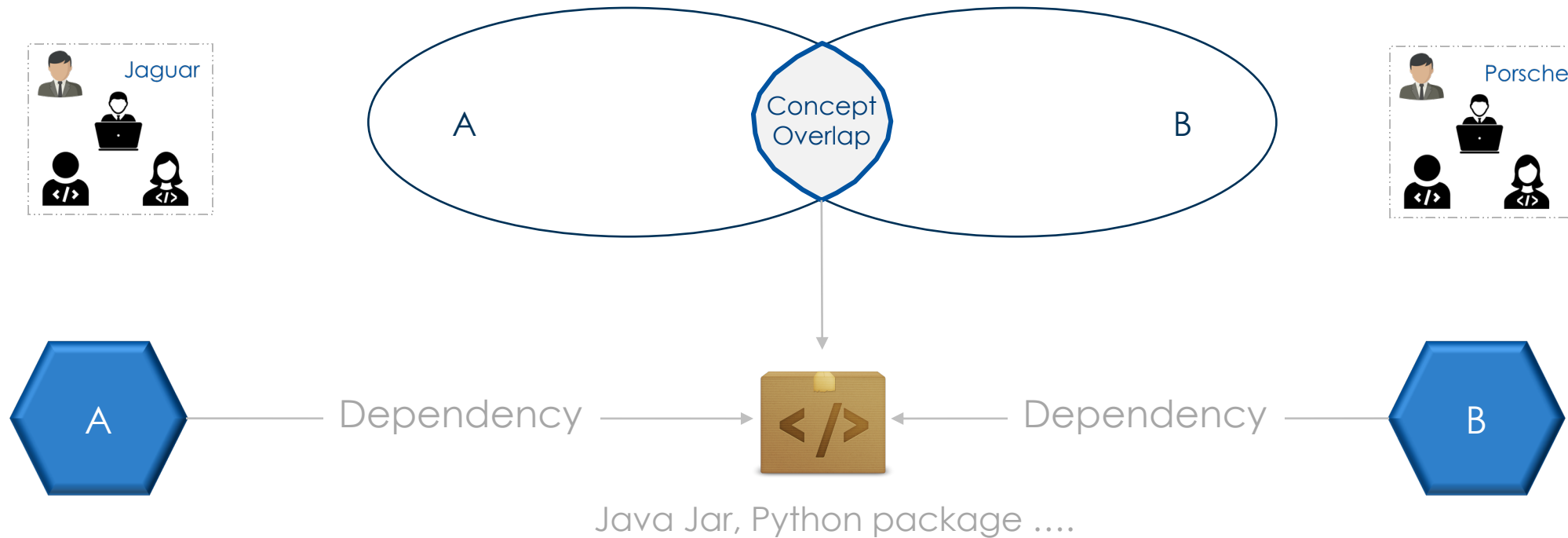
Shared Kernel Realization

May be implemented as a shared library



Shared Kernel Realization

May be implemented as a shared library



Shared Kernel Management

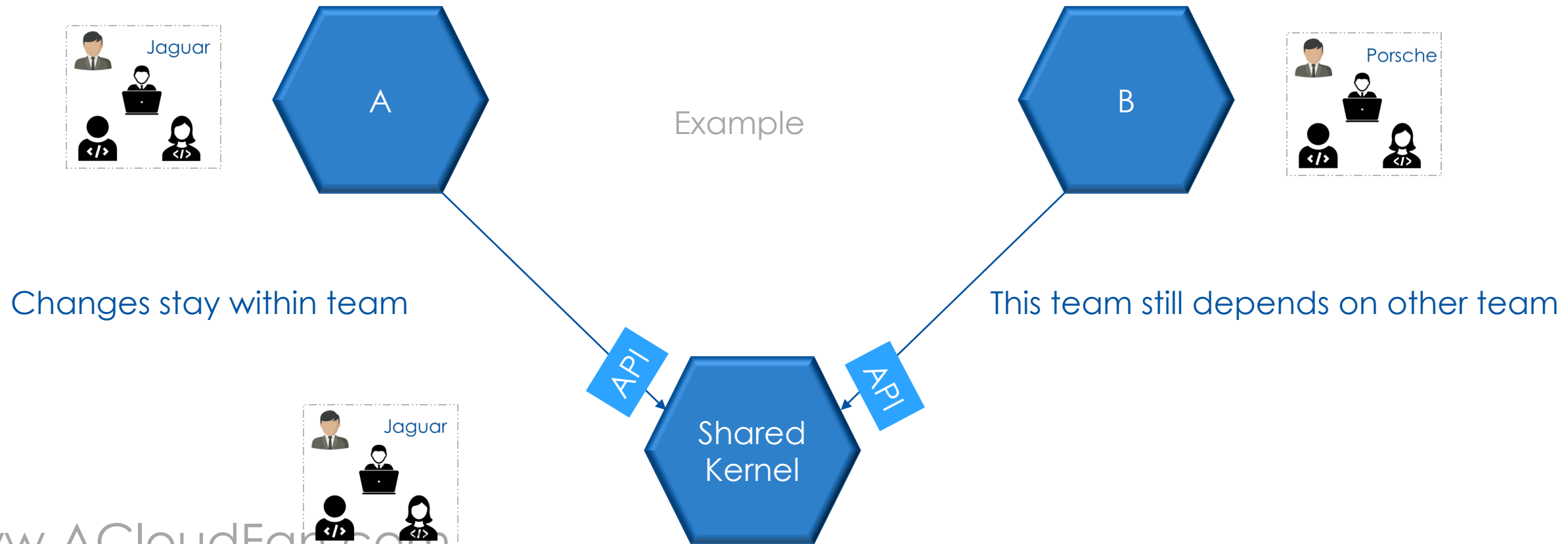


OK for a small library - small set of shared concepts

Difficult to maintain the integrity of the boundaries of the Bounded Contexts

Shared Kernel grown too big?

Consider extracting the kernel as separate BC





Quick Review

Separate Ways

Teams can work on BC independently

Partnership

Teams **MUST** coordinate to make changes

Shared Kernel

Coordination needed **ONLY** for making changes to the shared components

Asymmetric Relationships

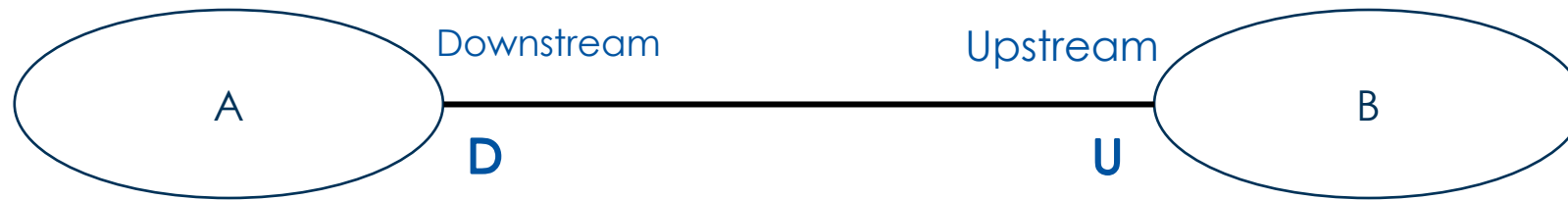
One Bounded Context depends on the other



- 1 BC Roles : Upstream & Downstream
- 2 Customer-Supplier Pattern
- 3 Conformist Pattern
- 4 Anti Corruption Layer Pattern

Asymmetric Relationship | Unidirectional Dependency

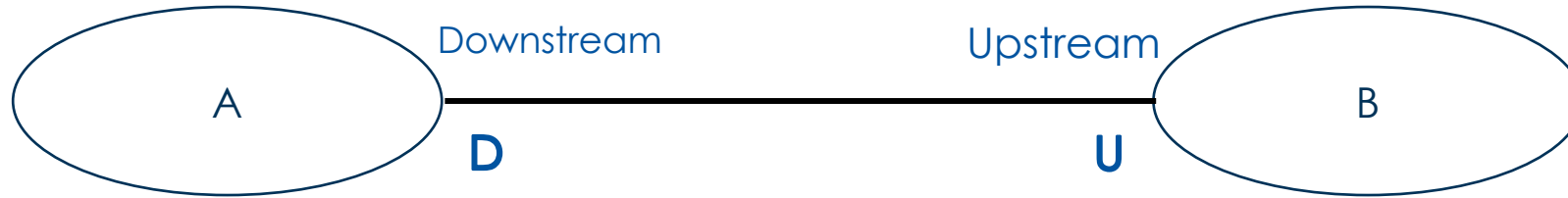
One BC depends on another BC



Bounded Context A has knowledge of models in Bounded Context B

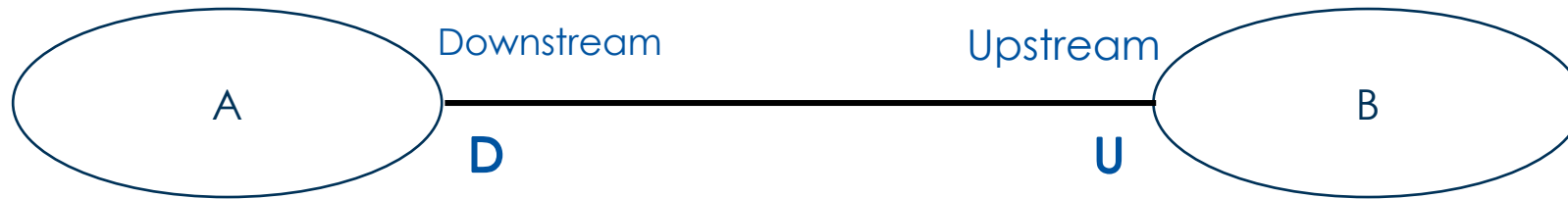
NOTE: Does NOT indicate the flow of data or information

Functionality



Upstream BC exposes functionality & models that are consumed by the Downstream BC

2 Possibilities



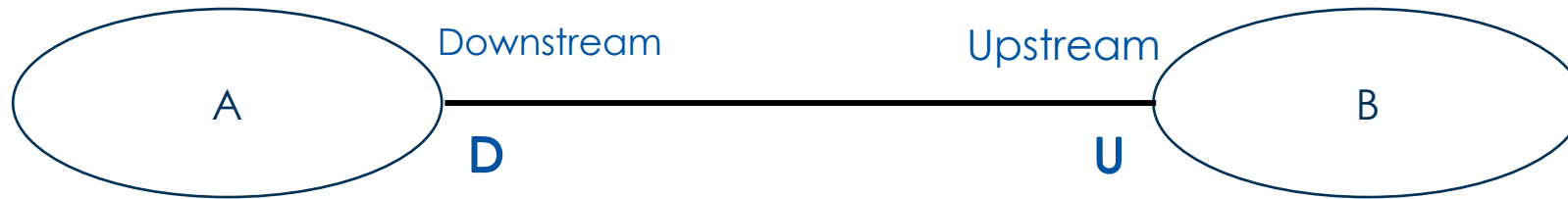
1

Upstream BC exposes models based on needs of
Downstream BC

2

Upstream BC exposes models with NO consideration to needs of
Downstream BC

BC Roles: Upstream & Downstream



1

Upstream BC fulfils some specific needs of Downstream BC

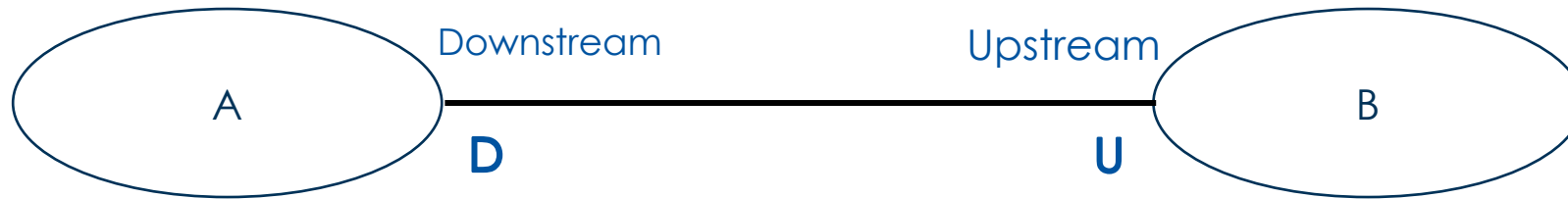
Customer - Supplier Pattern

Customer-Supplier Pattern Realization (example)



Supplier adjusts to fulfil needs of the customer

Downstream accepts Upstream models



2

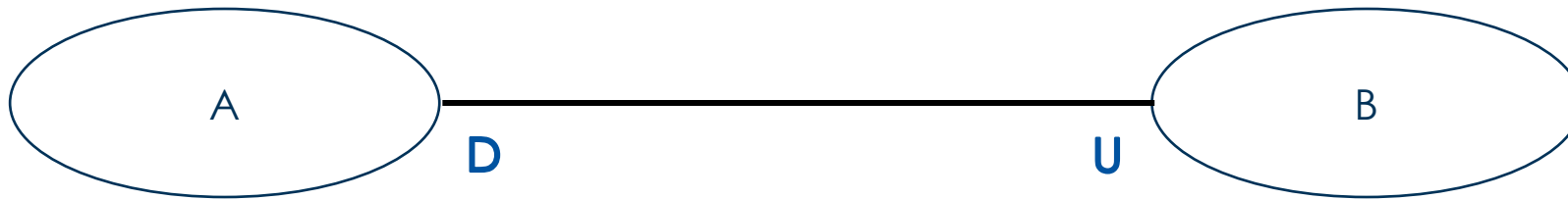
Upstream BC exposes models with no regard to ANY Downstream BC

Downstream BC accepts models exposed by Upstream BC

Conformist Pattern

Conformist Pattern

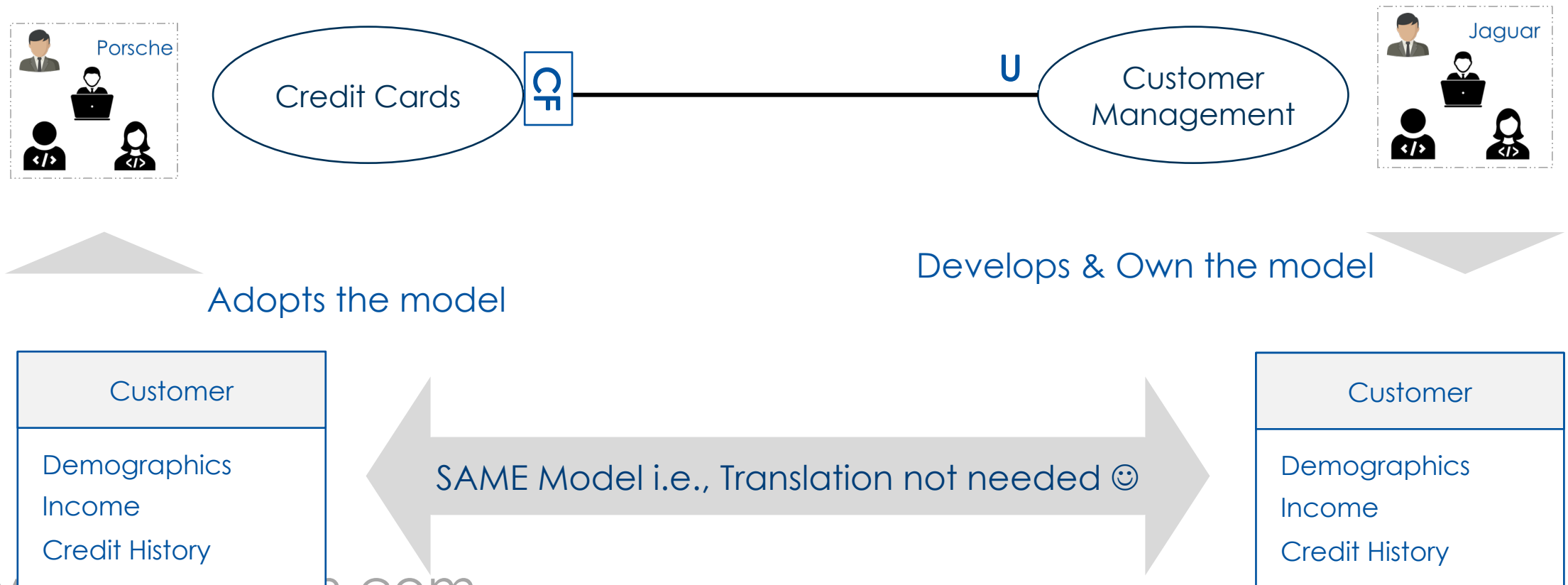
Downstream BC conforms to the Upstream BC Models



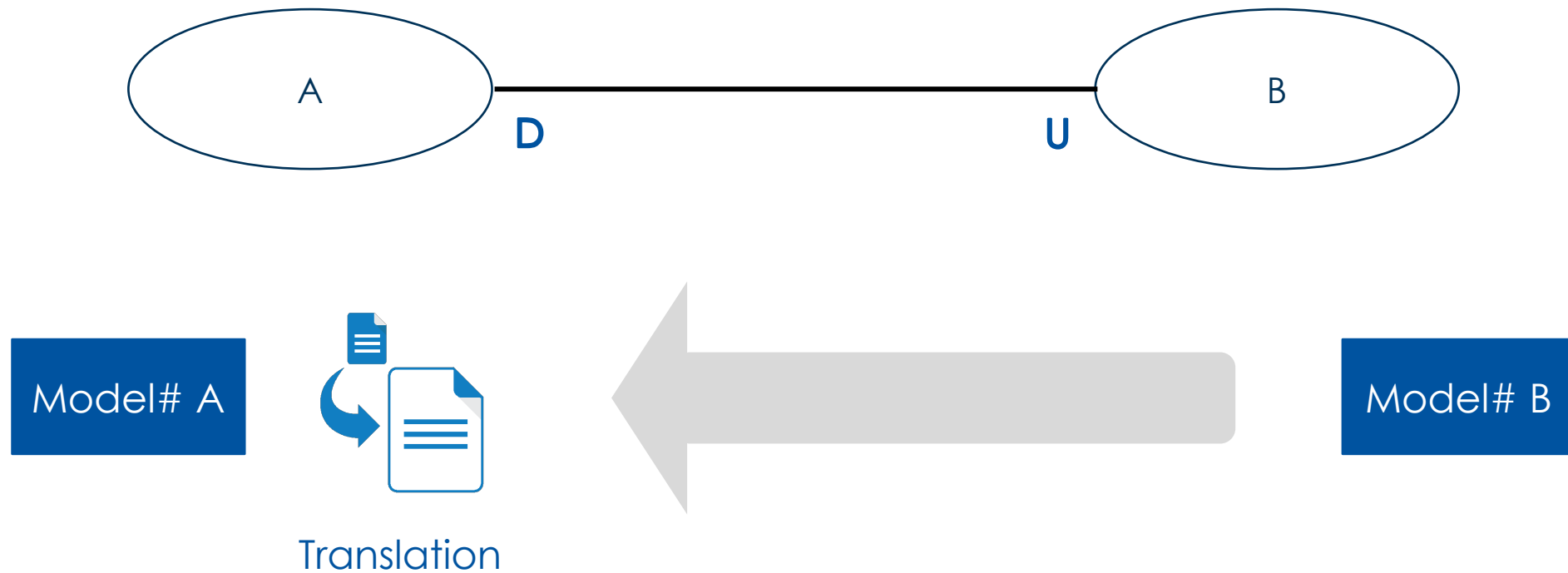
Depicted by using Ω on Downstream

Conformist Pattern (Example)

Same model in use in both the BCs



What if Downstream is NOT Conformist?

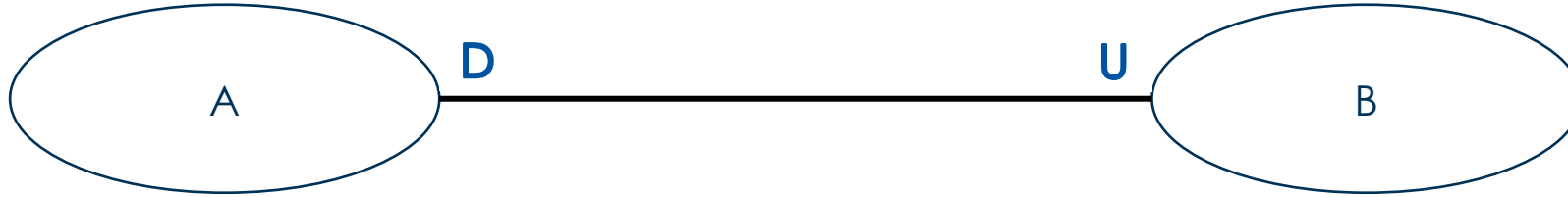


Downstream BC isolates the translation logic in a separate layer

Anti Corruption Layer - Pattern

Anti Corruption Layer - Pattern

Protect the BC from corruption

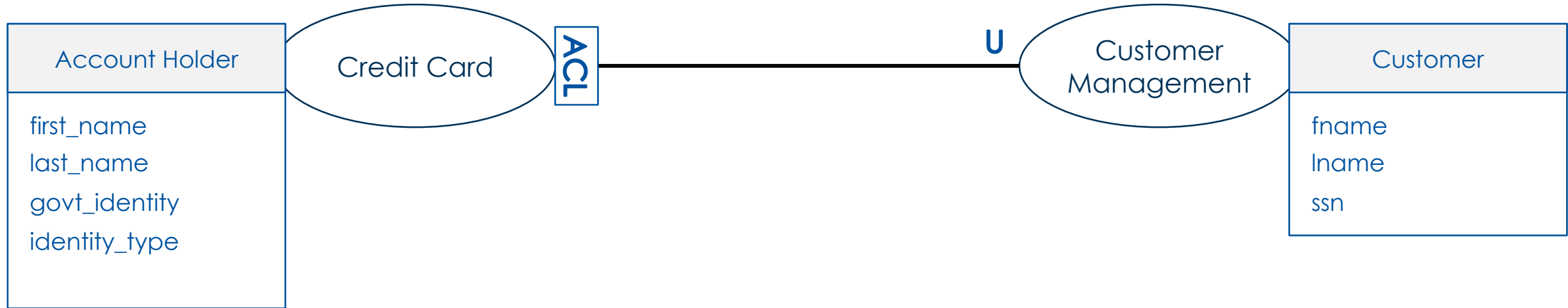


Depicted by using ACL on Downstream

Downstream BC uses ACL to translate the Upstream Models

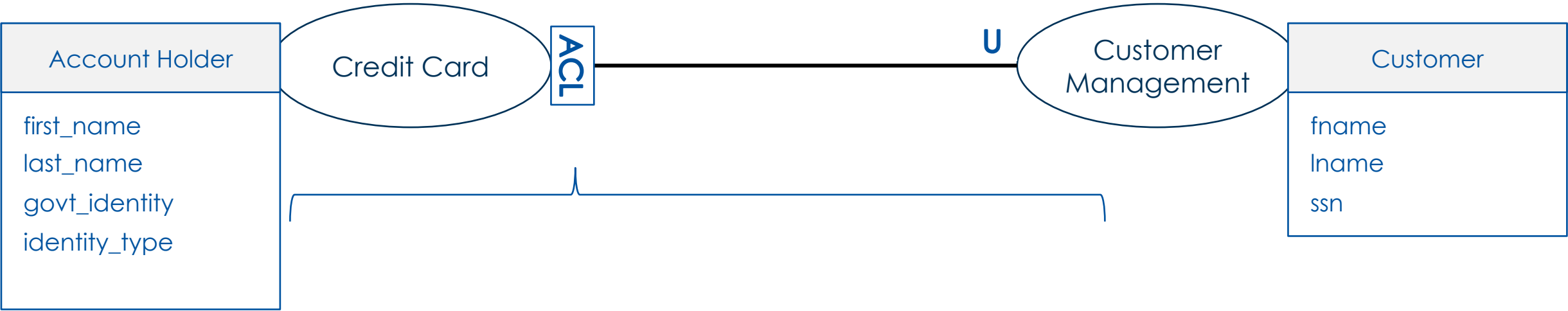
ACL Pattern (Example)

ACL translates from upstream model to downstream model



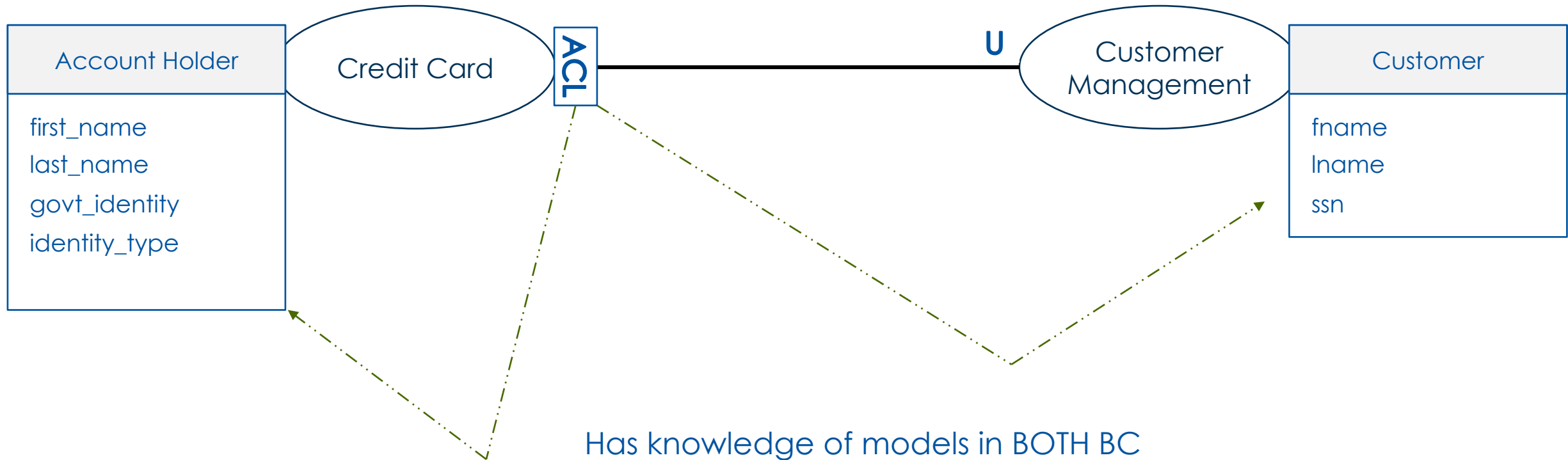
ACL Pattern (Example)

ACL translates from upstream model to downstream model



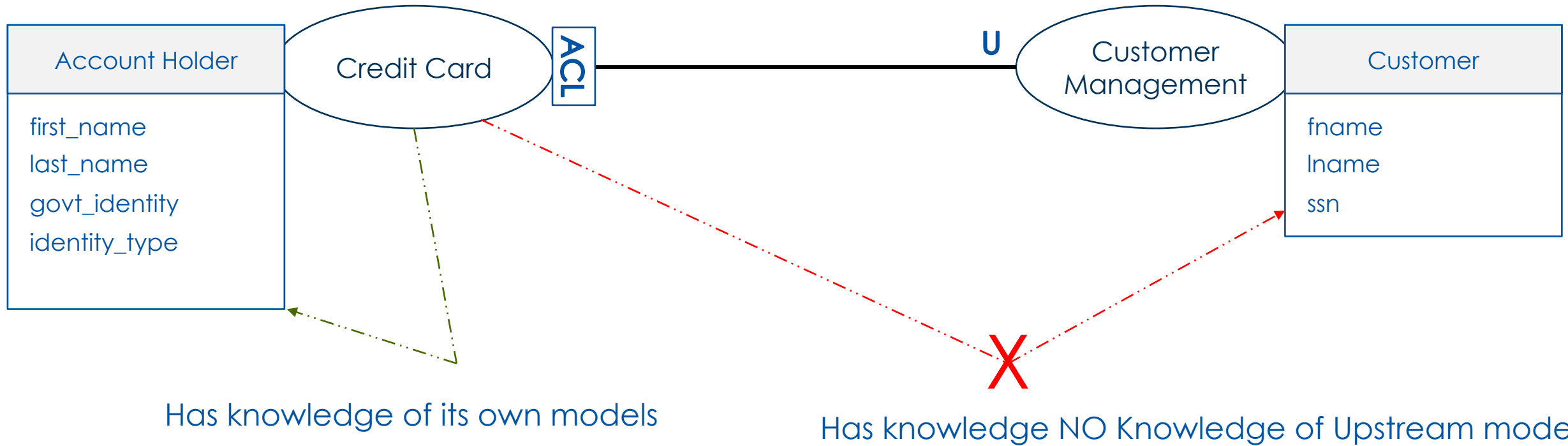
ACL Dependencies

ACL has knowledge of BOTH BC models



Downstream BC Dependencies

Downstream BC has NO knowledge of Upstream BC Model



Downstream protected from Upstream changes by way of **ACL**



Quick Review

Asymmetric Relation

Downstream BC depends on the Upstream BC

Customer-Supplier

Upstream BC adjusts to the needs of Downstream BC

Conformist

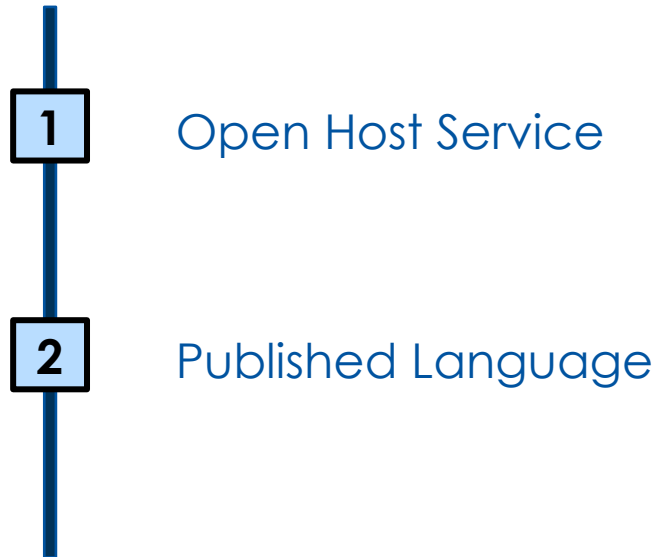
Downstream conforms to the Upstream Models
Upstream has no knowledge of Downstream

ACL

Protects the Downstream from Upstream
i.e., model translation isolated to the ACL layer

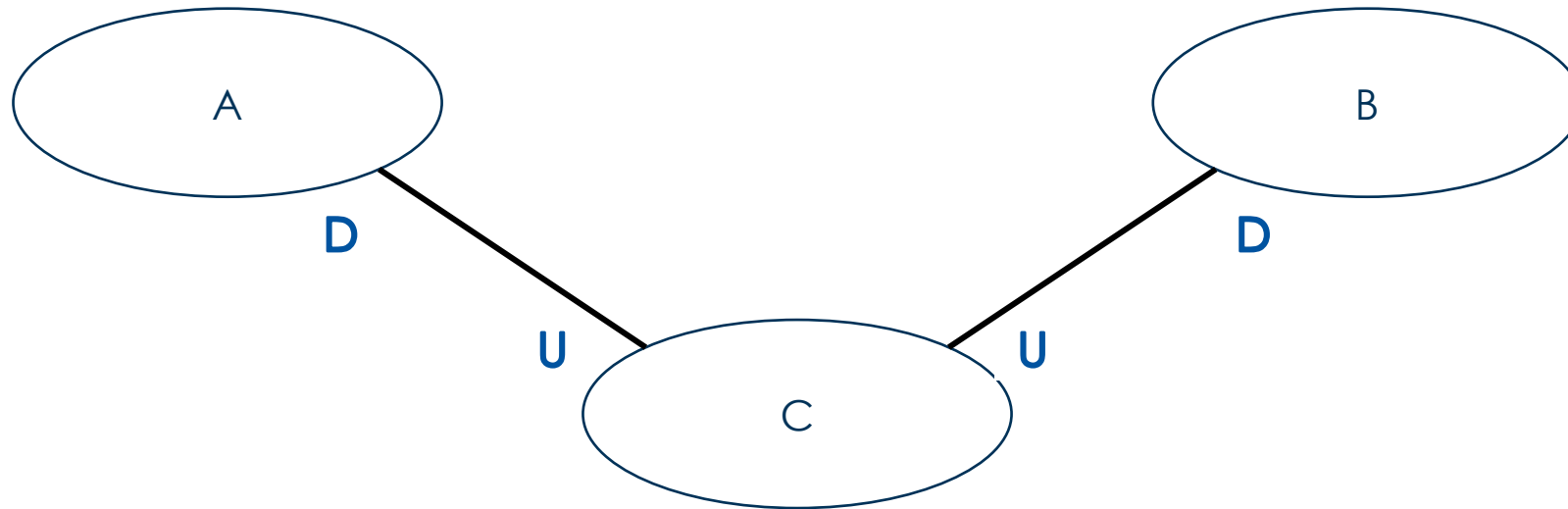
One to Many Relationship

Multiple Bounded Contexts depend on one Bounded Context



One to Many Relationship

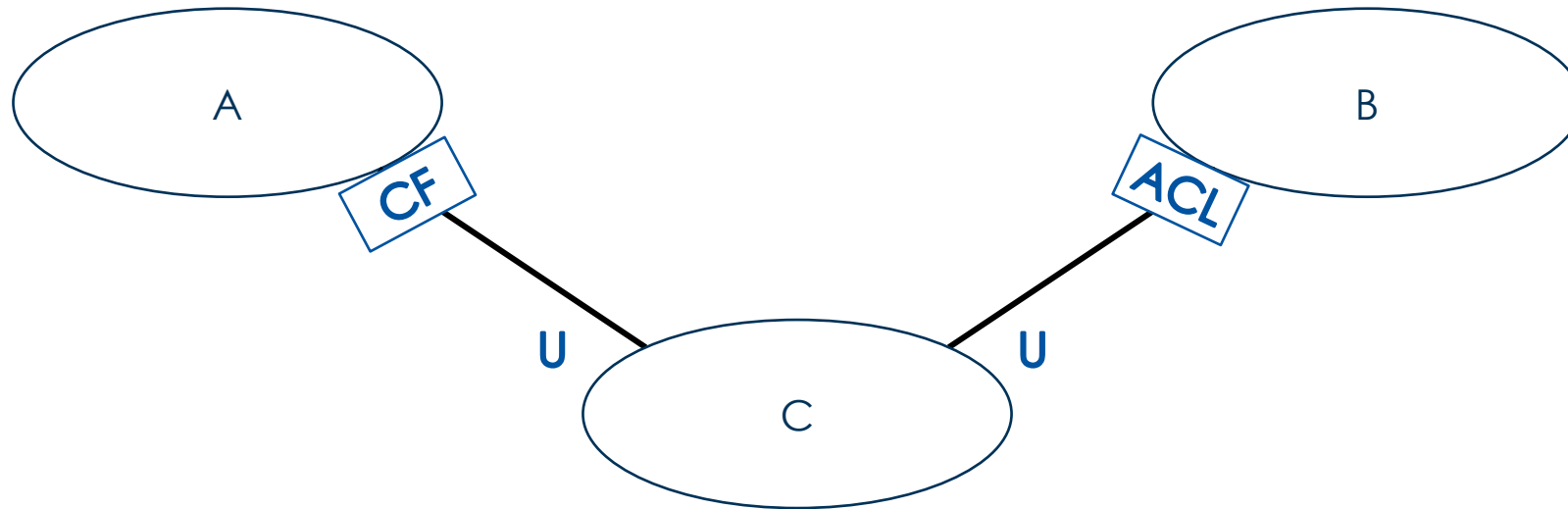
Upstream provider offers common services to other BC's



The Upstream provides common integration model for all Downstream(s)

One to Many Relation

Downstream may *Conform* or use *ACL*

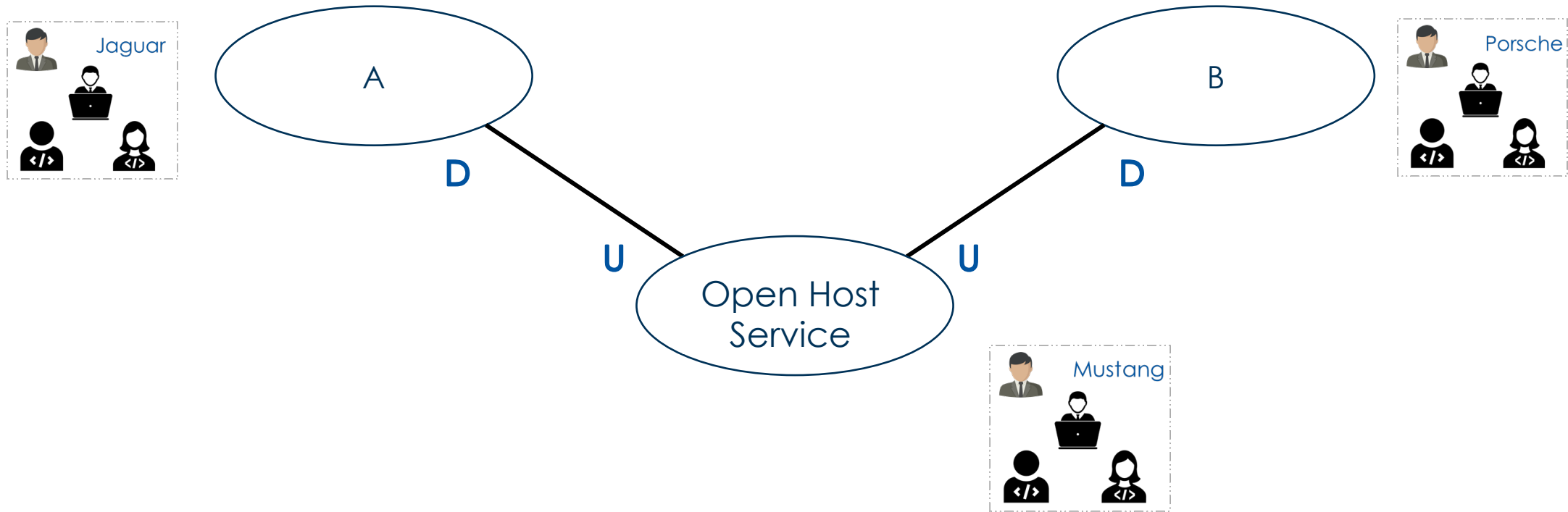


Example: A is a Conformist

B is using ACL for protection

Services Realization

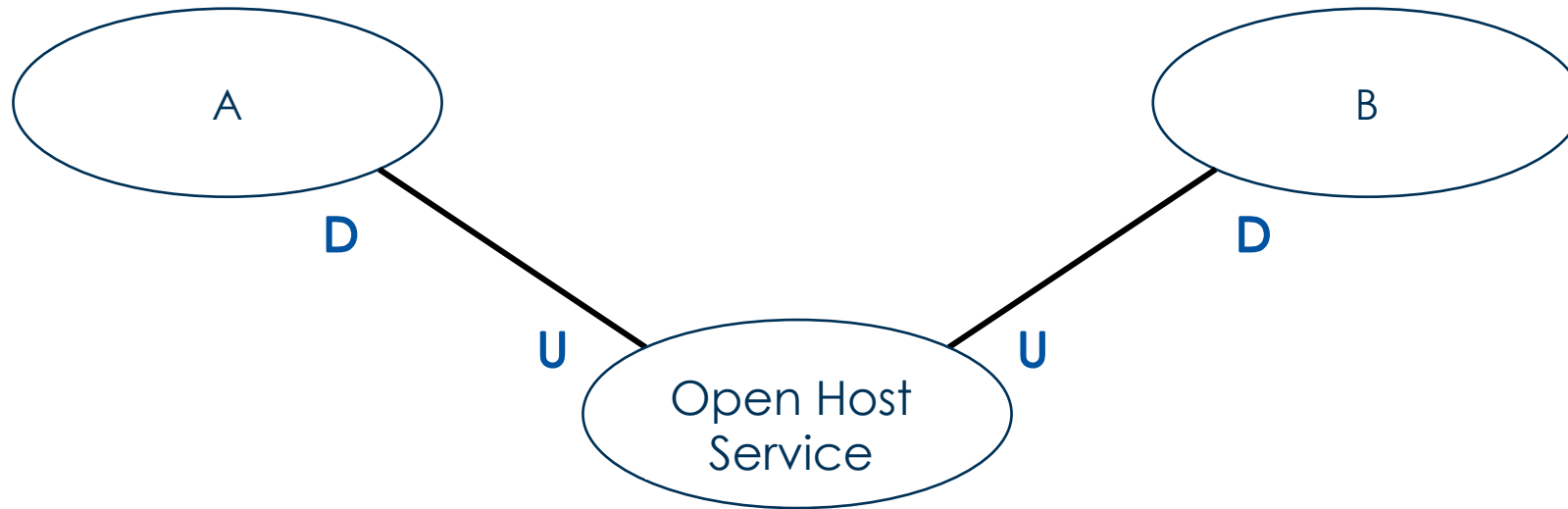
Teams assigned to the BC work independently



Open Host Service - Pattern

Open Host Service - Pattern

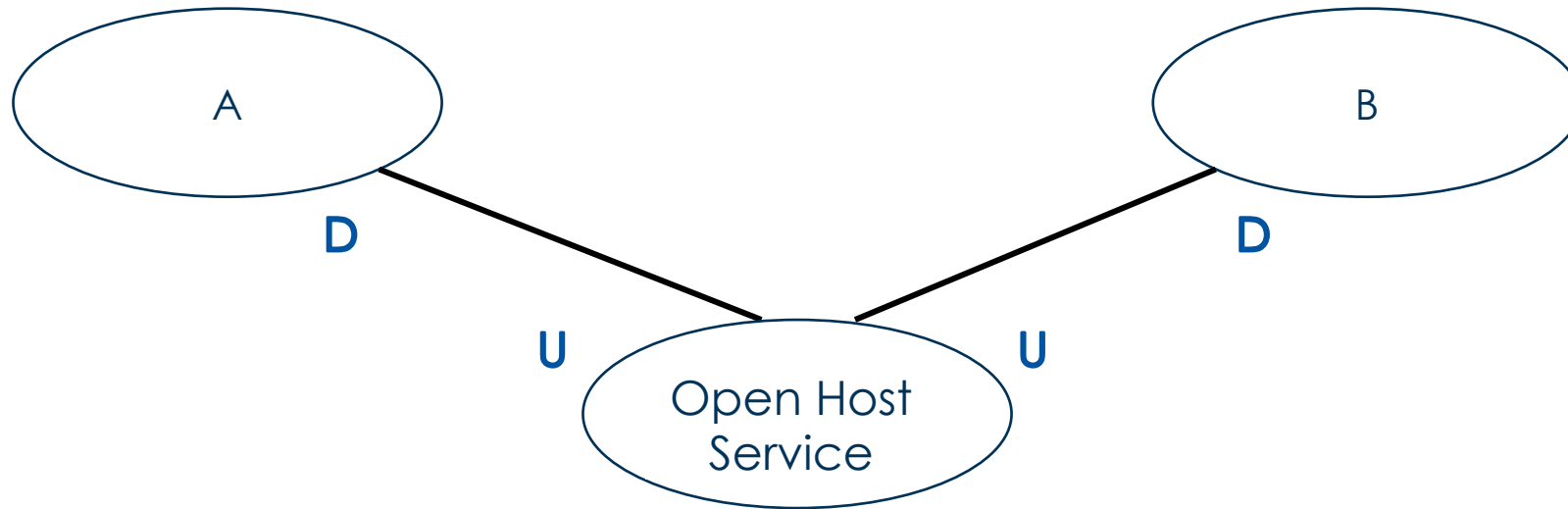
Upstream provides common services to Downstream(s)



Depicted by placing **OHS** in front of Upstream

Open Host Service - Pattern

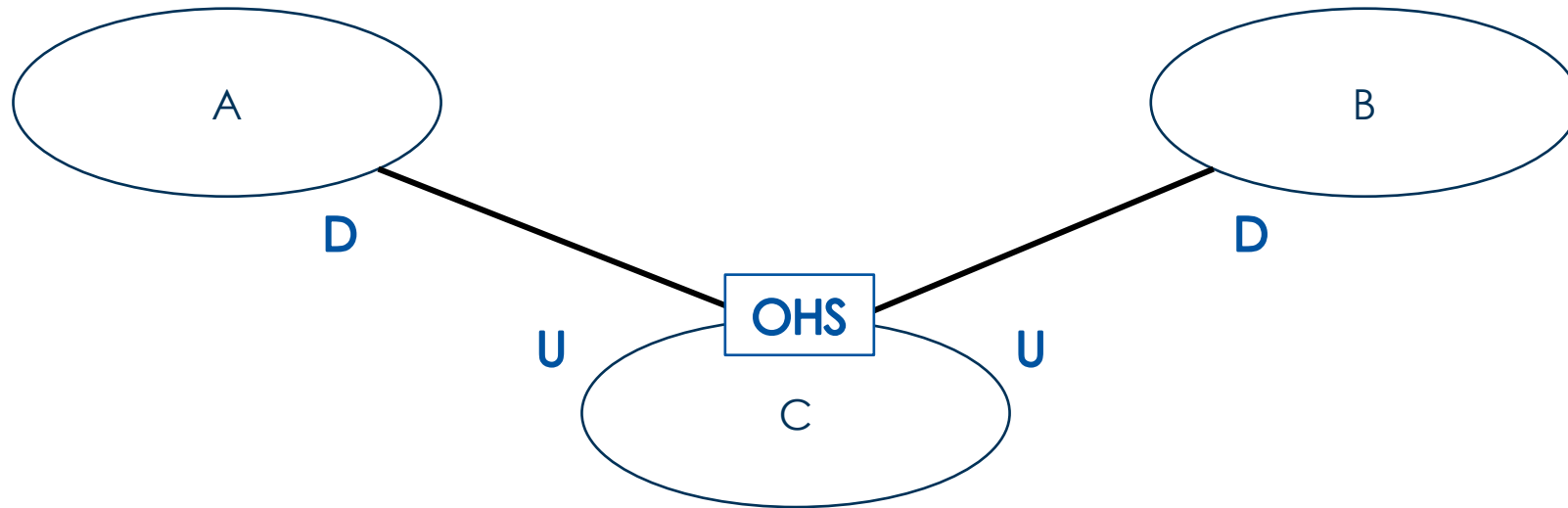
Upstream provides common services to Downstream(s)



Depicted by placing **OHS** in front of Upstream

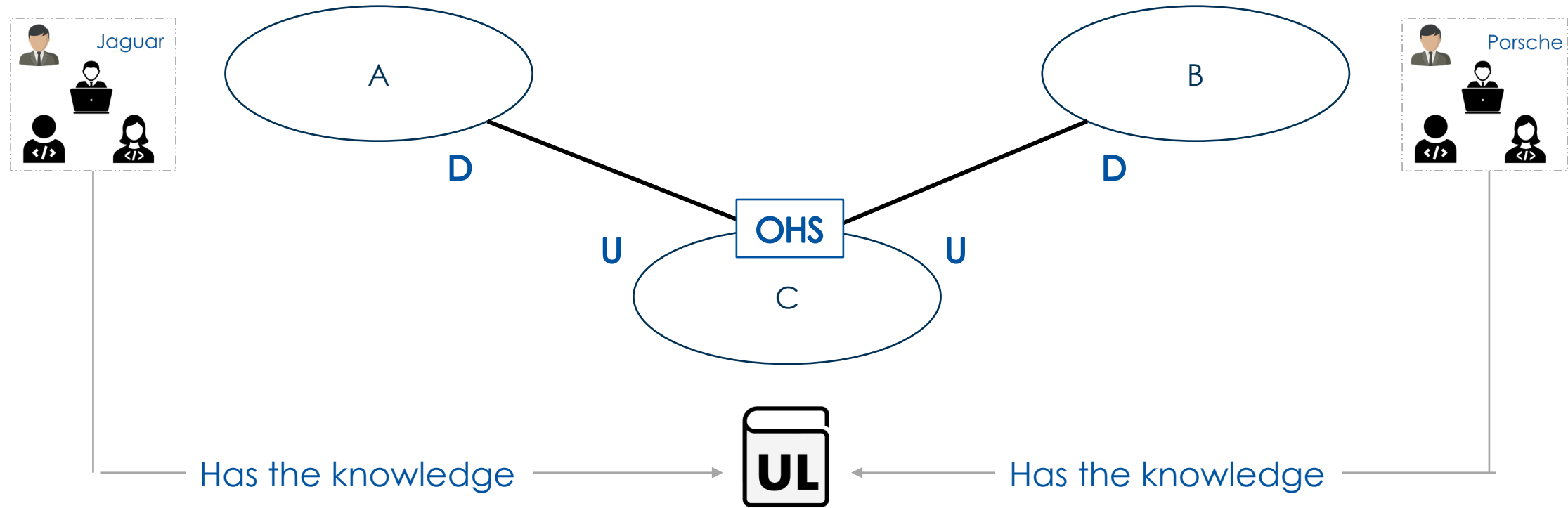
Common Language

OHS publishes a common Language for the Integrations



Common Language

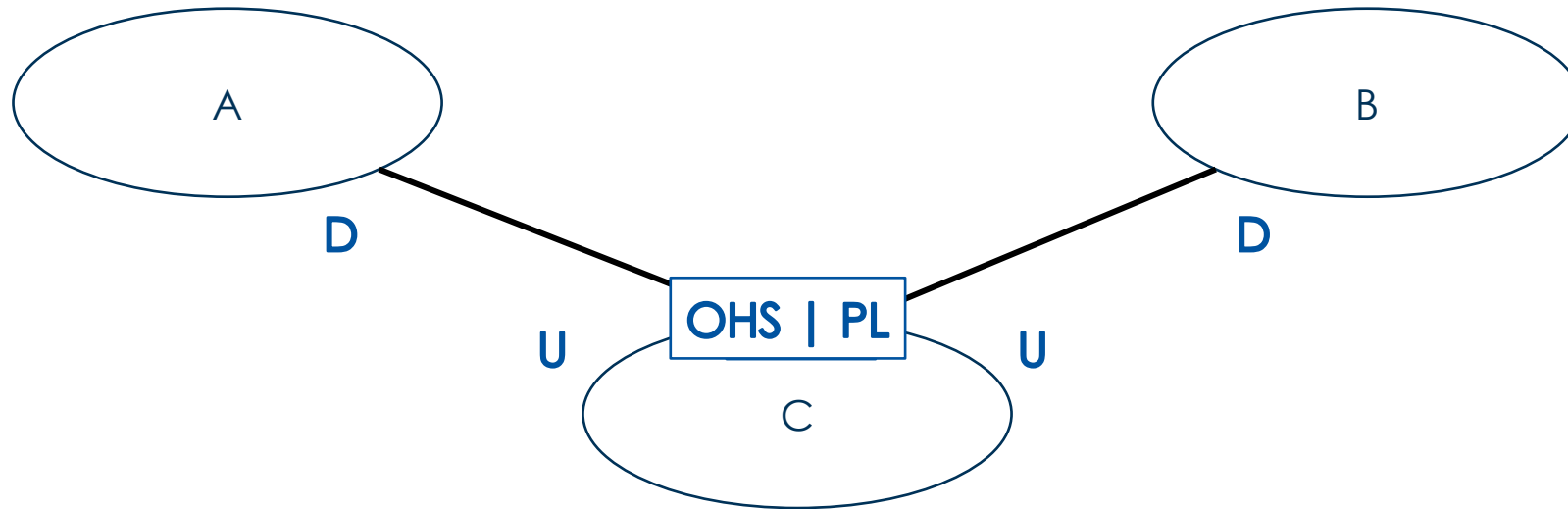
This common language is well accepted by Downstream



Published Language - Pattern

Published Language - Pattern

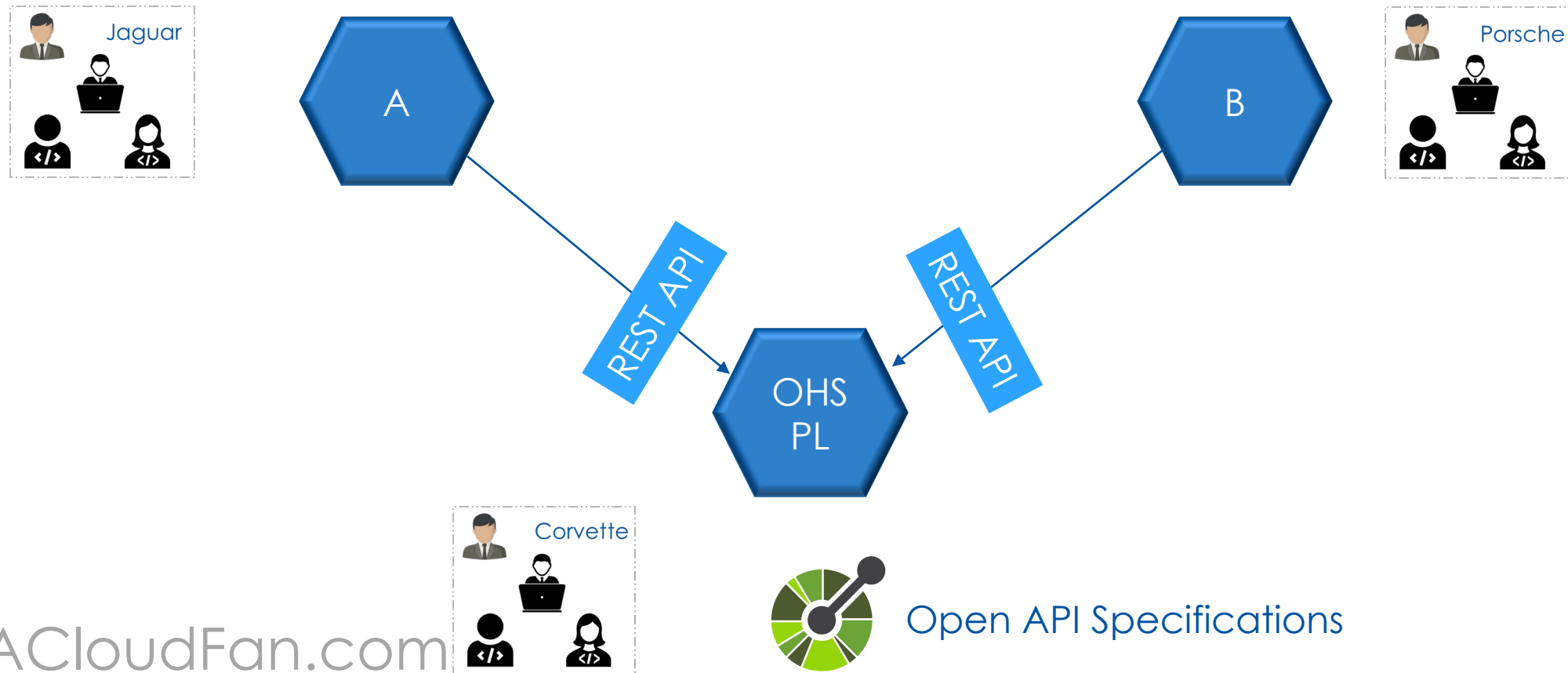
Common Language used by the OHS



Depicted by placing **PL** in front of Upstream

OHS | PL Realization Example

Functionality exposed by OHS BC is well accepted





Quick Review

Open Host Service (OHS)

Upstream BC exposes common services

Published Language (PL)

Common Language created managed by the team for OHS

Context Mapping for *Bank*

Context Mapping in Action



1

Setting up the Context Maps

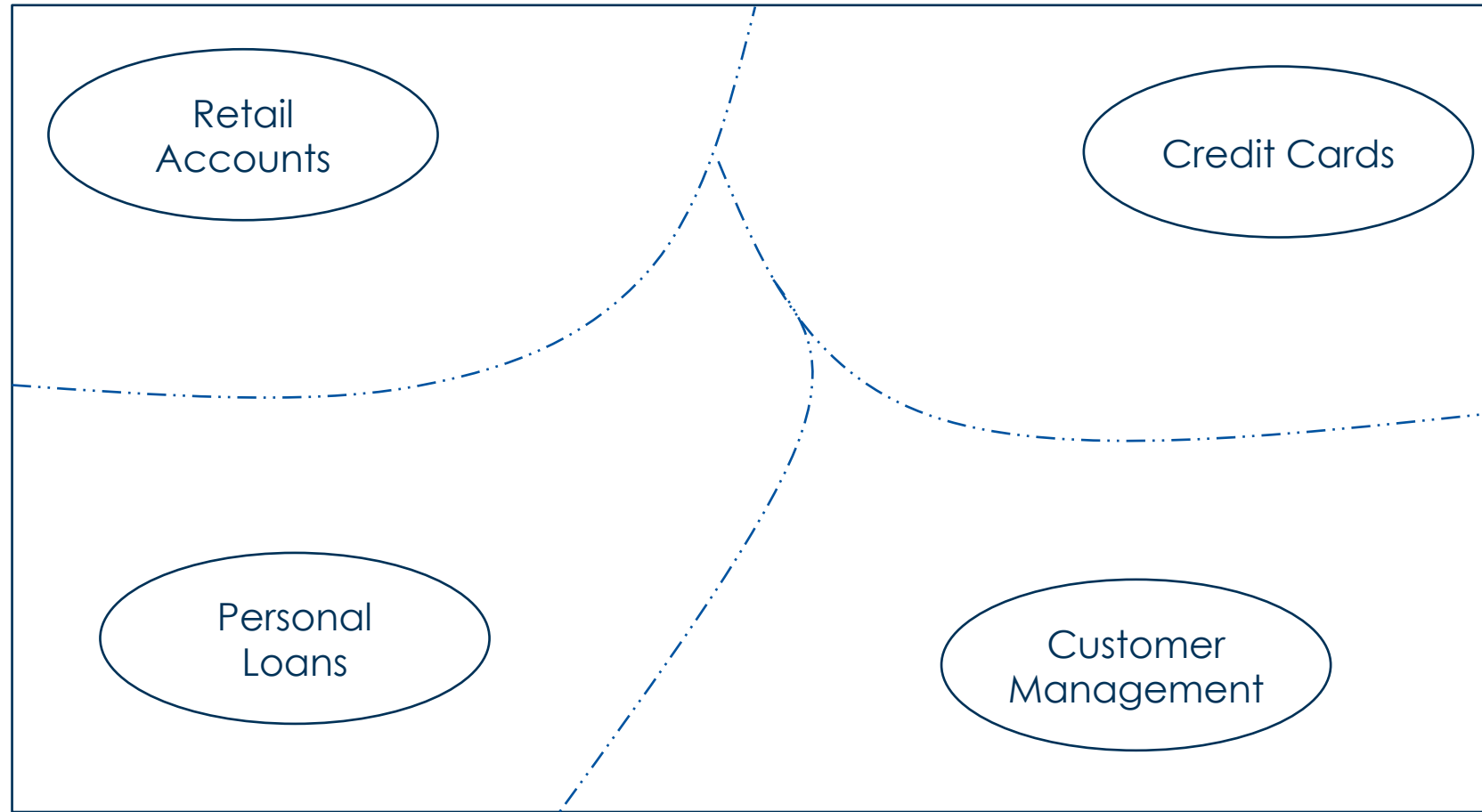
2

Identify the impact of change in one BC on other BC



Use the Bounded Contexts for the Bank

Two Part Exercise



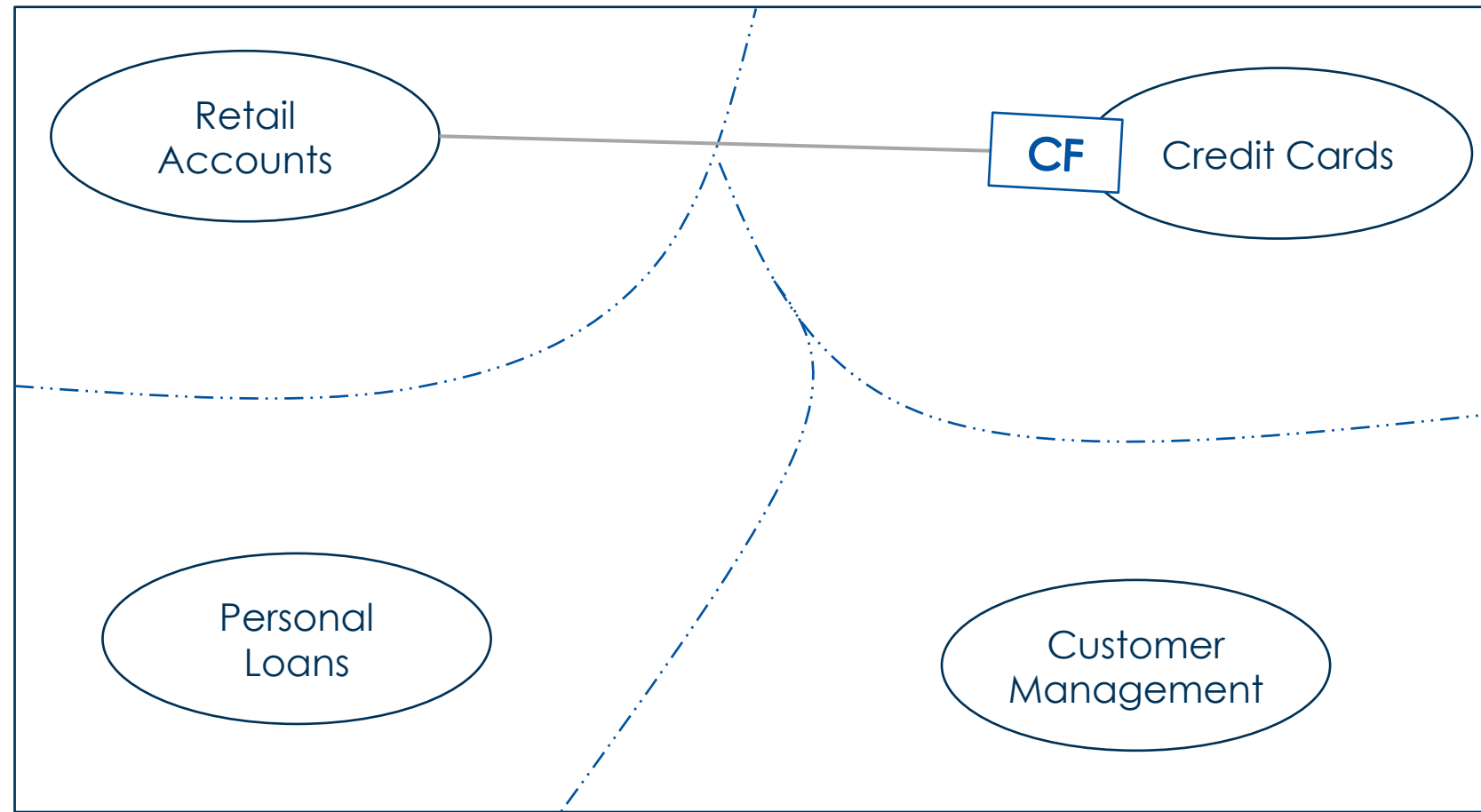


Part - 1

Decide the kind of pattern in use

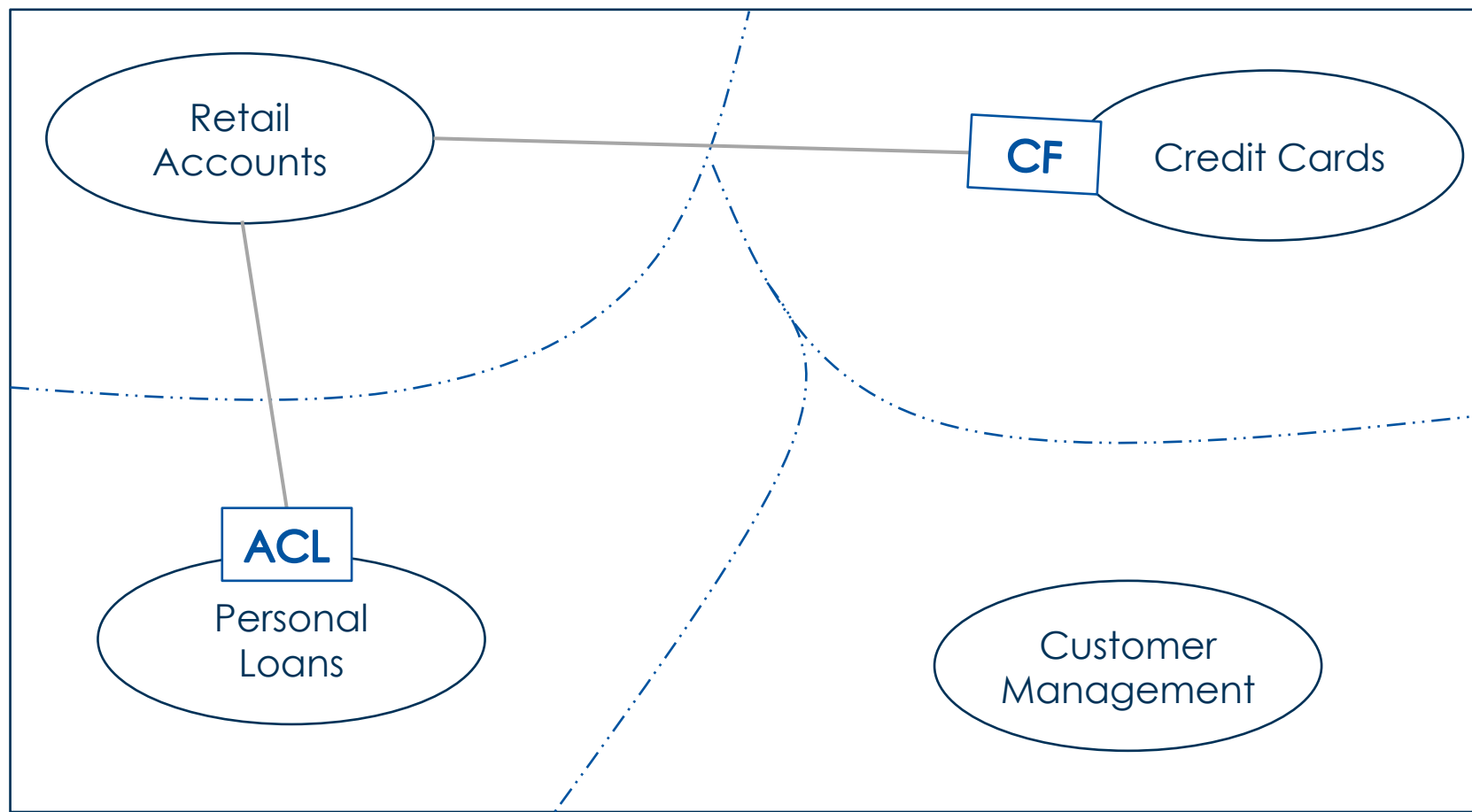


Credit Cards BC depends on Retail Accounts & has decided to accept the models



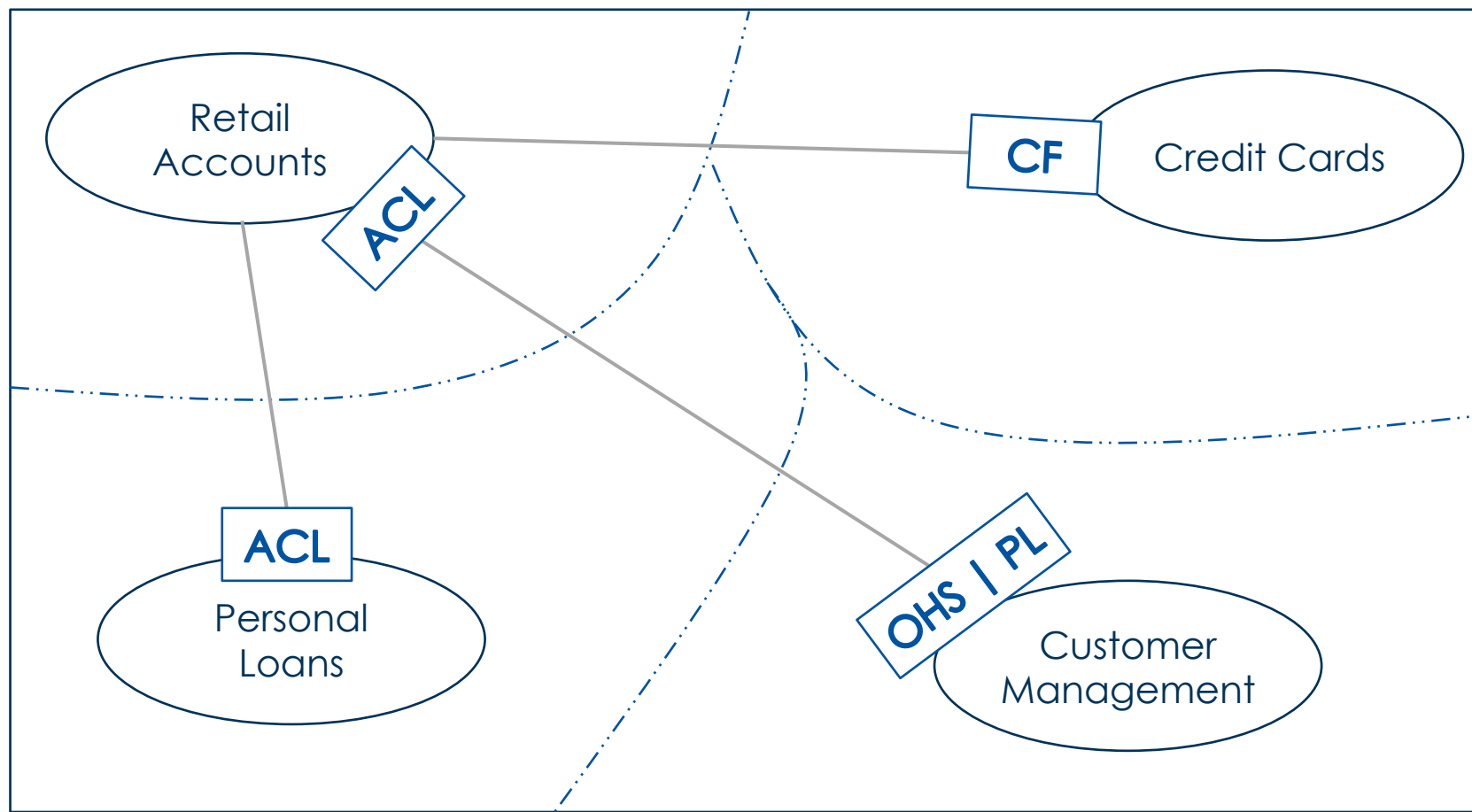


Personal Loans team using the Retail Accounts function but concerned about the use of outside model with their BC



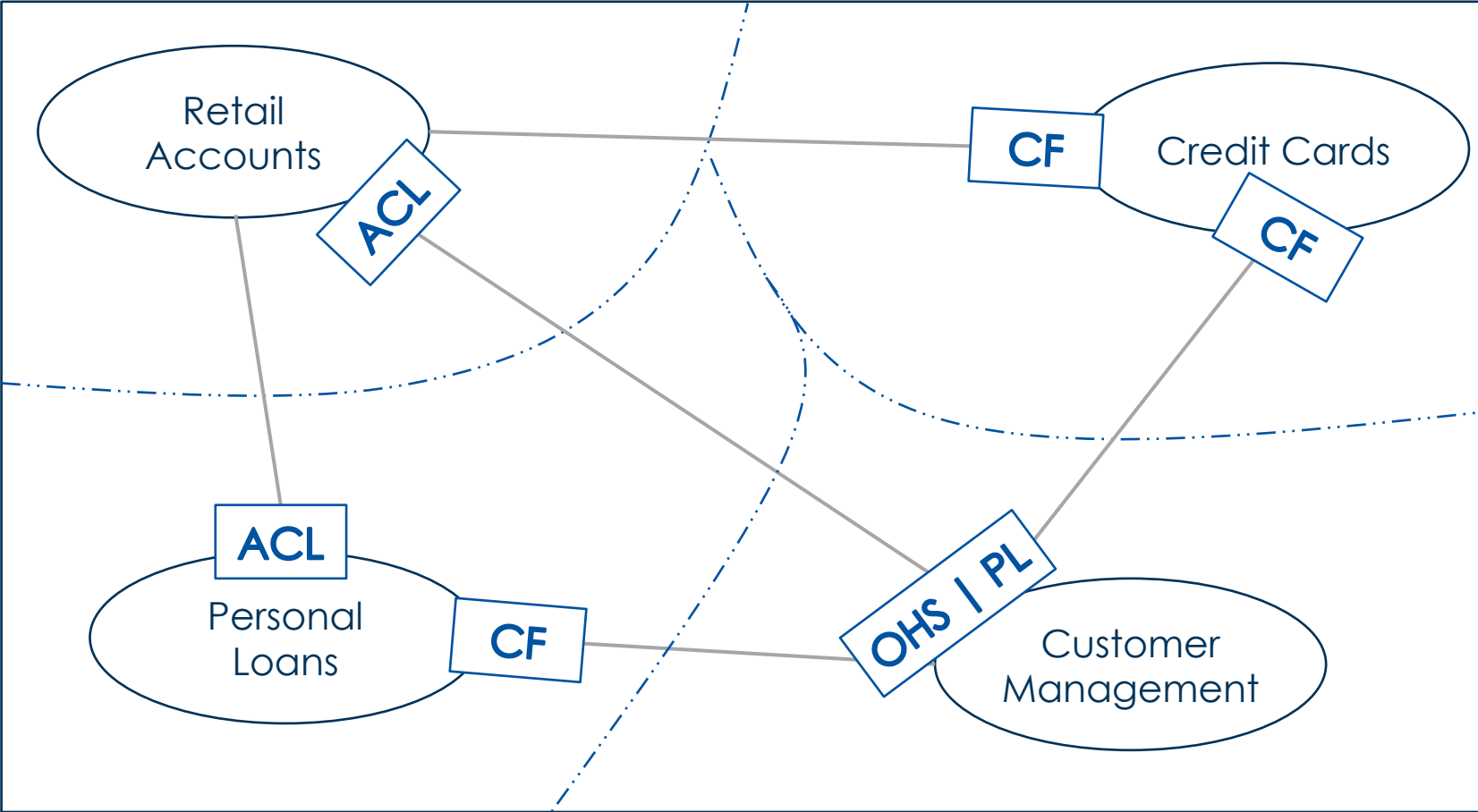


- Customer Management is exposing a common set of services
- Retail Accounts uses the common services with protection





Personal Loans & Credit Cards Teams decided to use the common services & accept the OHS models



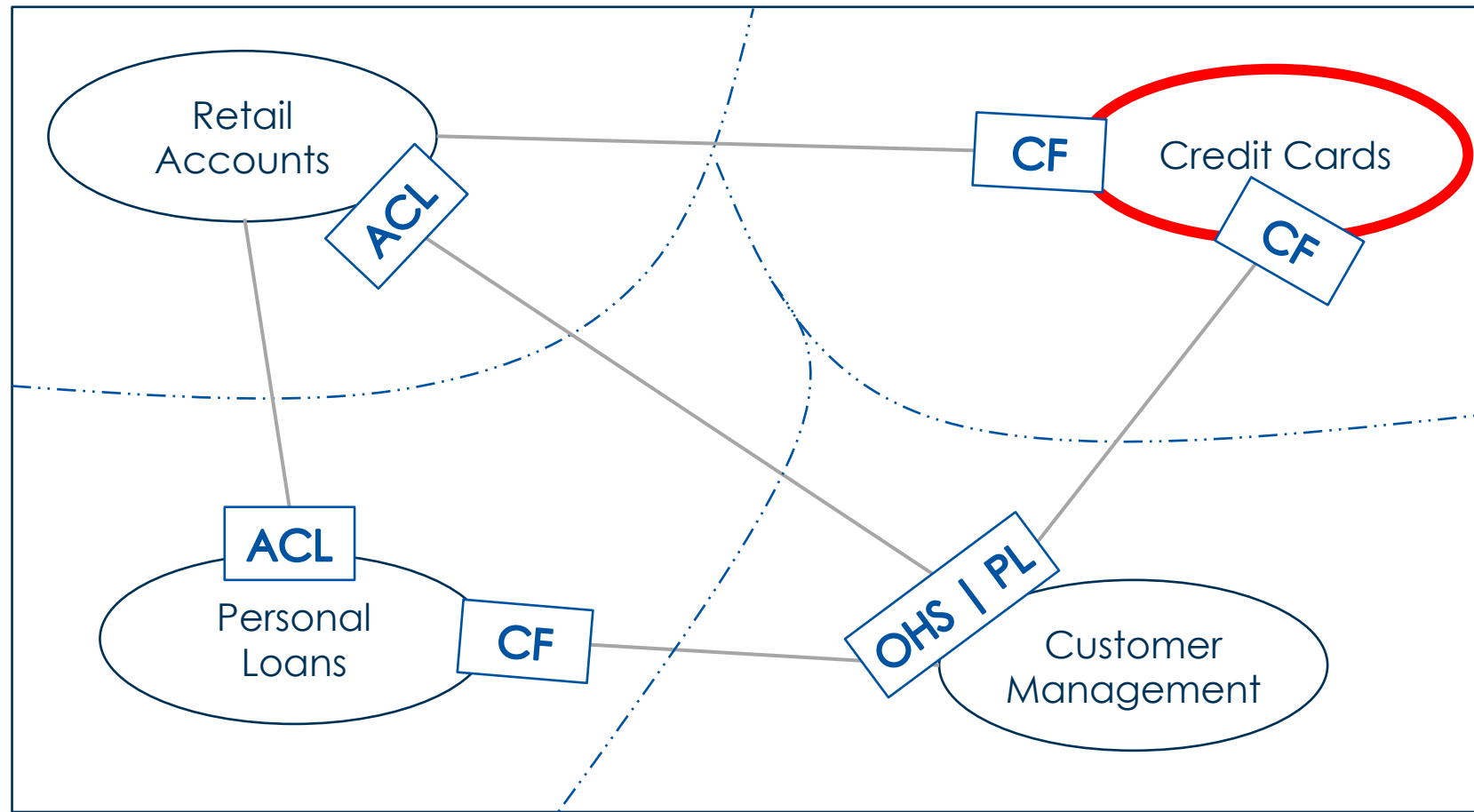


Part - 2

Interpretation of the Context Mapping

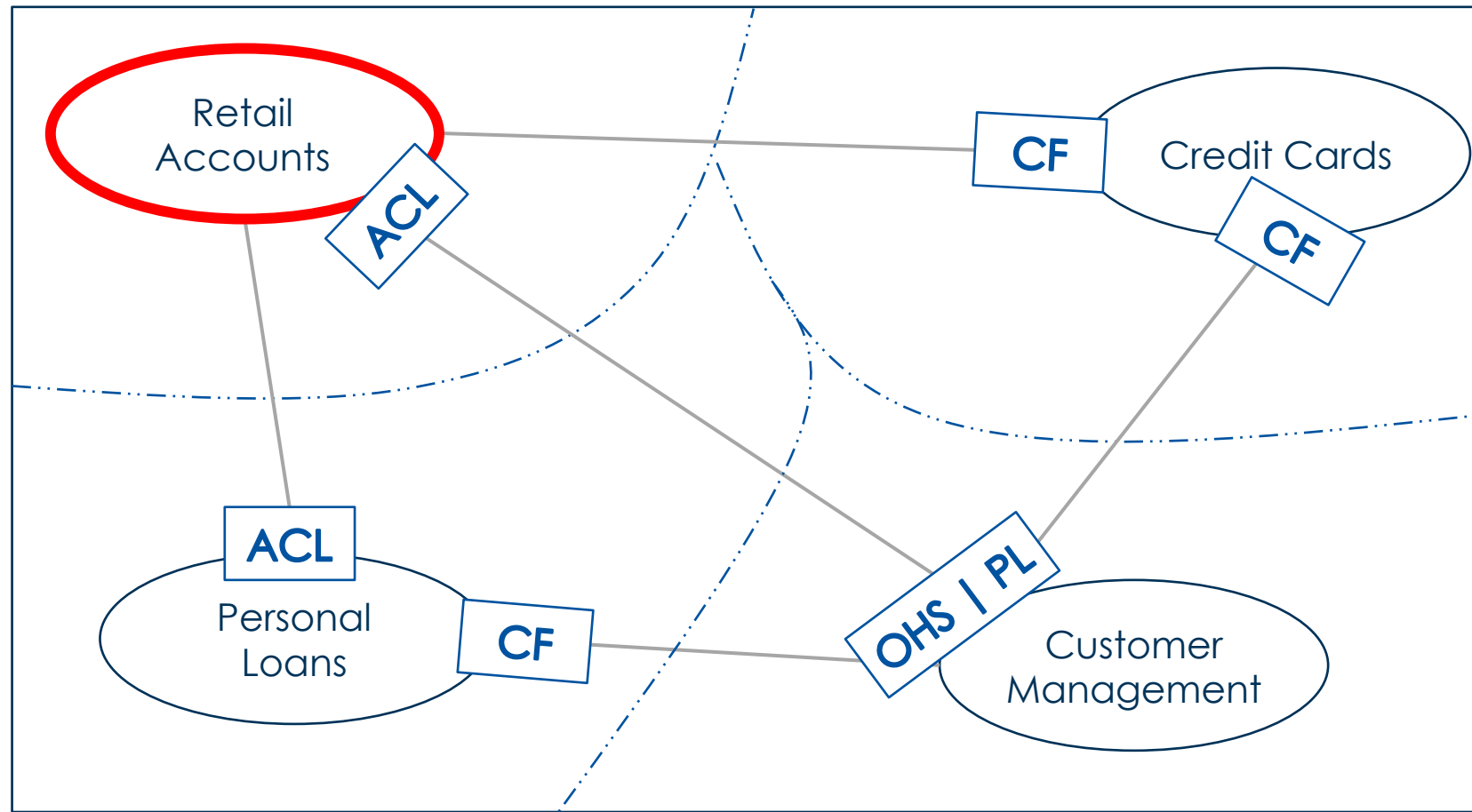


Which BC is expected to be *MOST* influenced by changes in other BCs? Why?



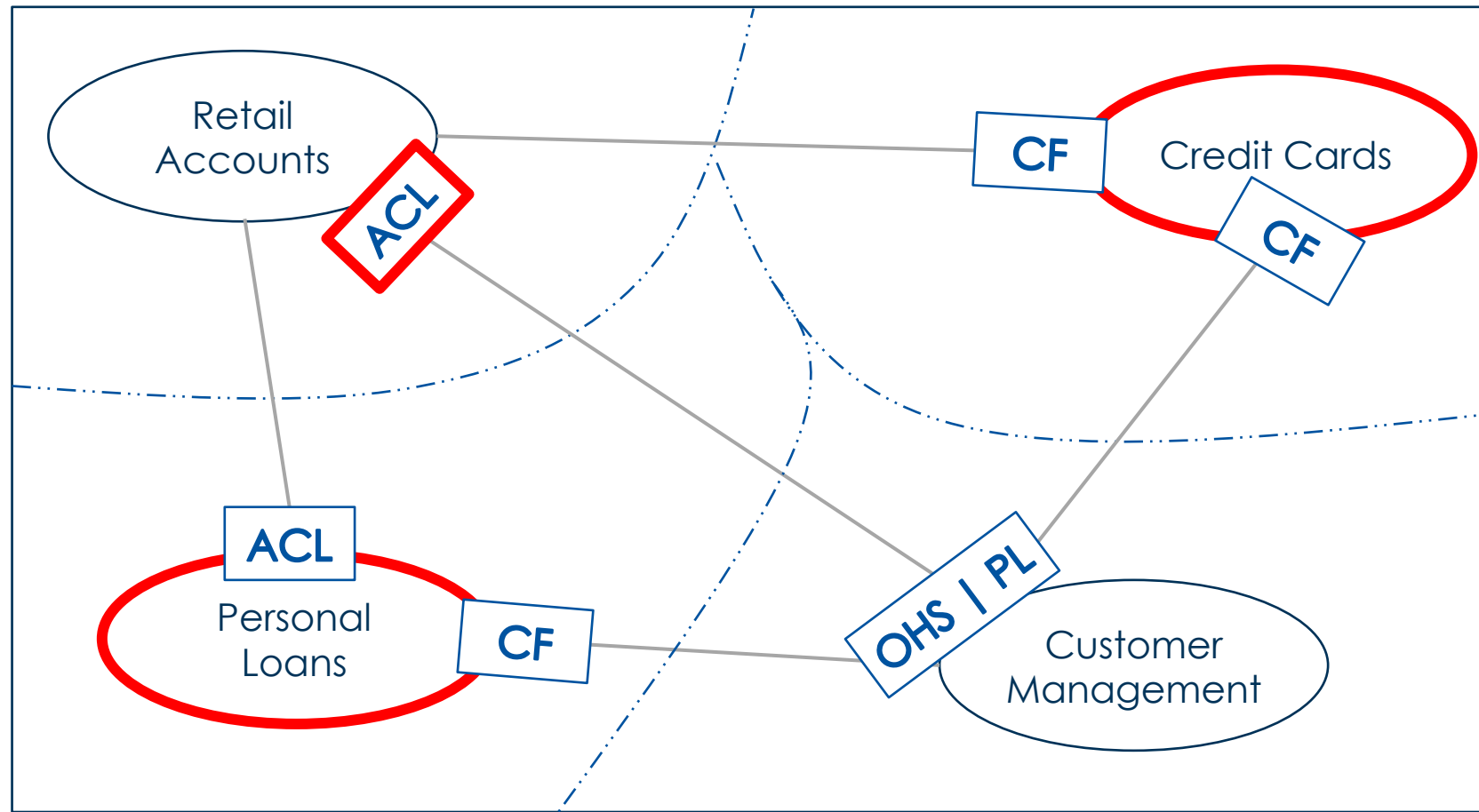


Which BC is NOT expected to be influenced by changes in other BCs? Why?

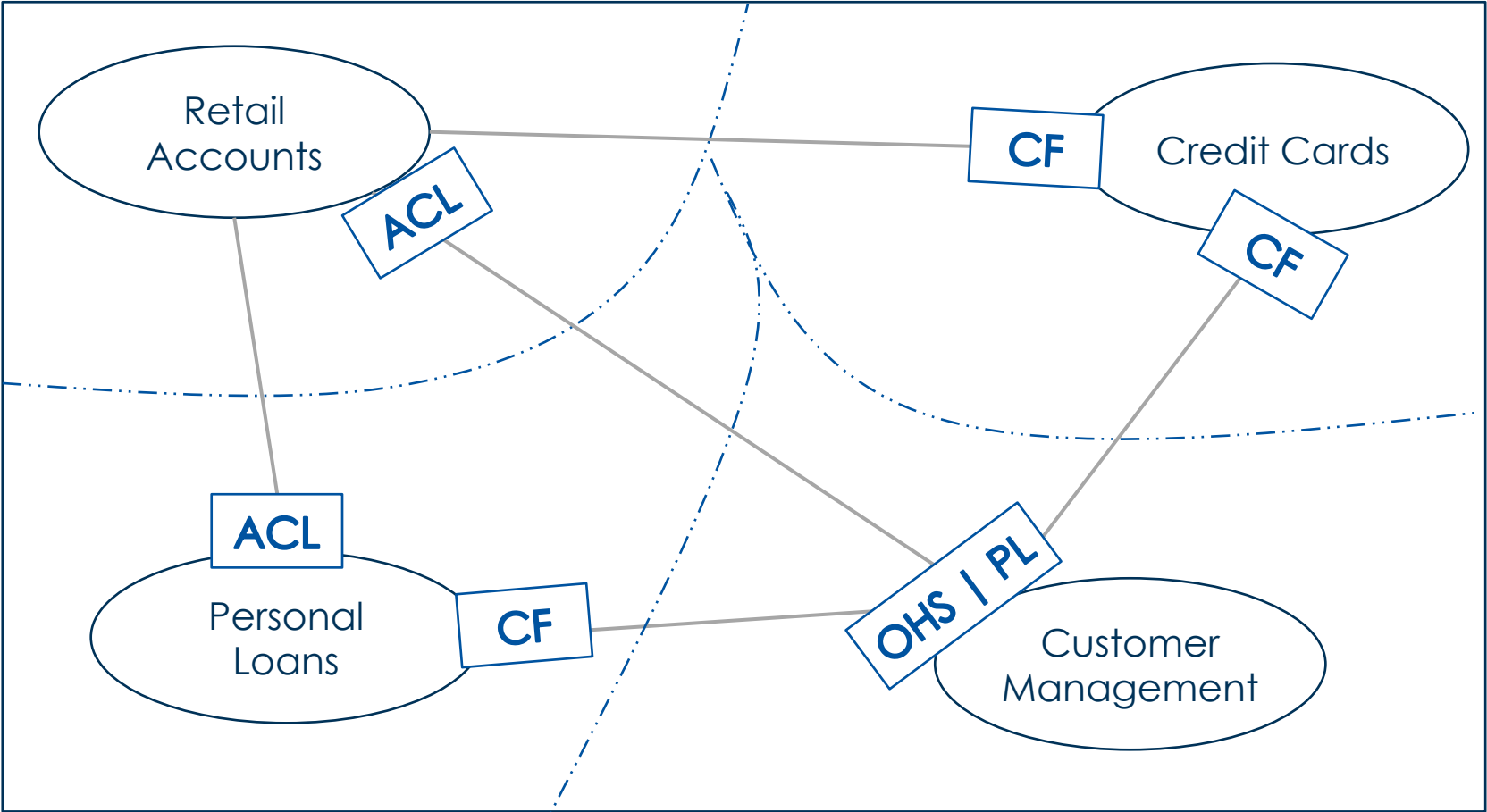




What all will need to be adjusted if there is a change in the integration model exposed by OHS?



Example of Bank's Context Mapping



BC Interactions | Dependencies

