



From **Software Architecture** to **N-tiers Architectures**

Borrowed to Sébastien Mosser

Software Architecture **Definition**

“ The **structure** of the system, which
comprise **software elements**,
externally **visible properties** of
those elements, and the
relationships among them.

Architecture versus Design?

Architecture is a subset of design

"External" design

Software Architecture **Objectives**

- It has the **functionality** required by the customer
- It is safely buildable on the **required schedule**
- It **performs adequately**
- It is **reliable**
- It is **usable** and **safe to use**



[Plonk et replonk]

De la presquitude des choses.

thanks to Clèm for the reference

Software Architecture **Concerns**

Producibility

Functionality

Changeability

Security

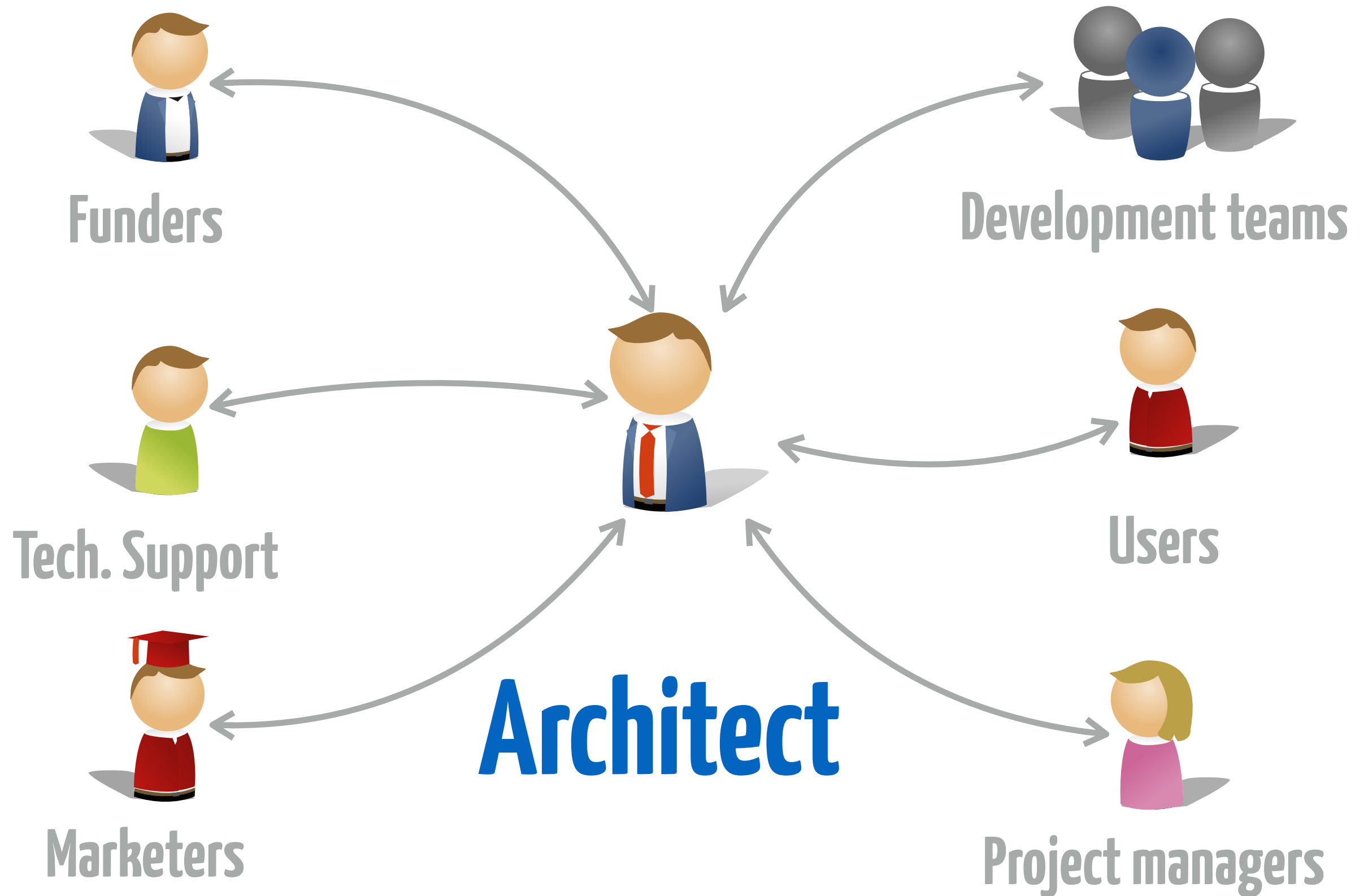
Modularity

Performance

Ecosystem

Buildability

The Architect Ecosystem



Expect the **unexpected**!



Parameters that were
never going to **change**
now need to be **modified**.

1

Architectural rule of thumb

2

Layered Architectures

3

Support from the



Architectural **rule of thumb**

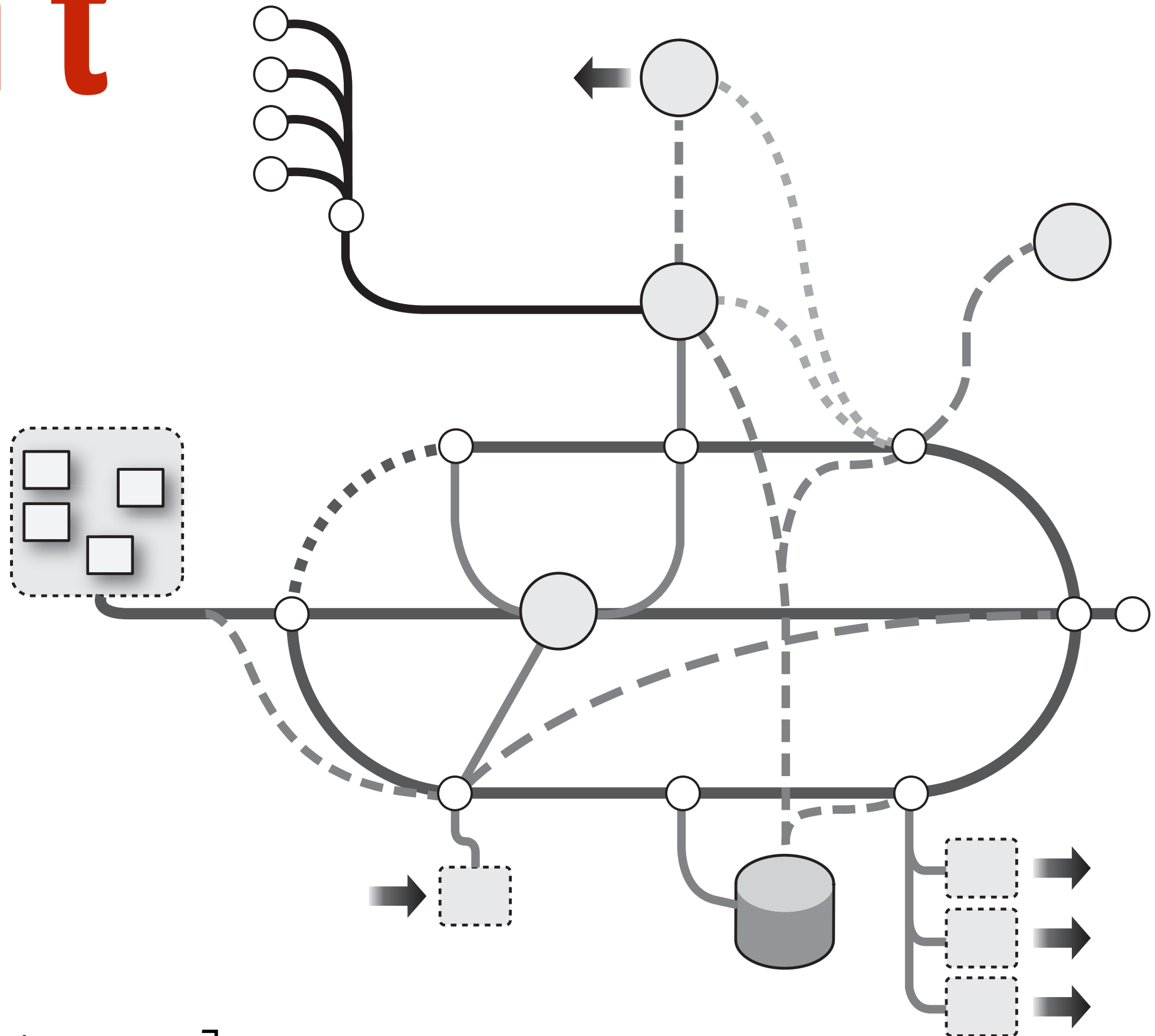
[Beautiful Architecture]

“

Software **architecture** is
not set in stone.

Change if you need it.

Don't



[Beautiful Architecture]

“

A **fuzzy architecture**

leads to **individual** code,

duplication of code and effort.

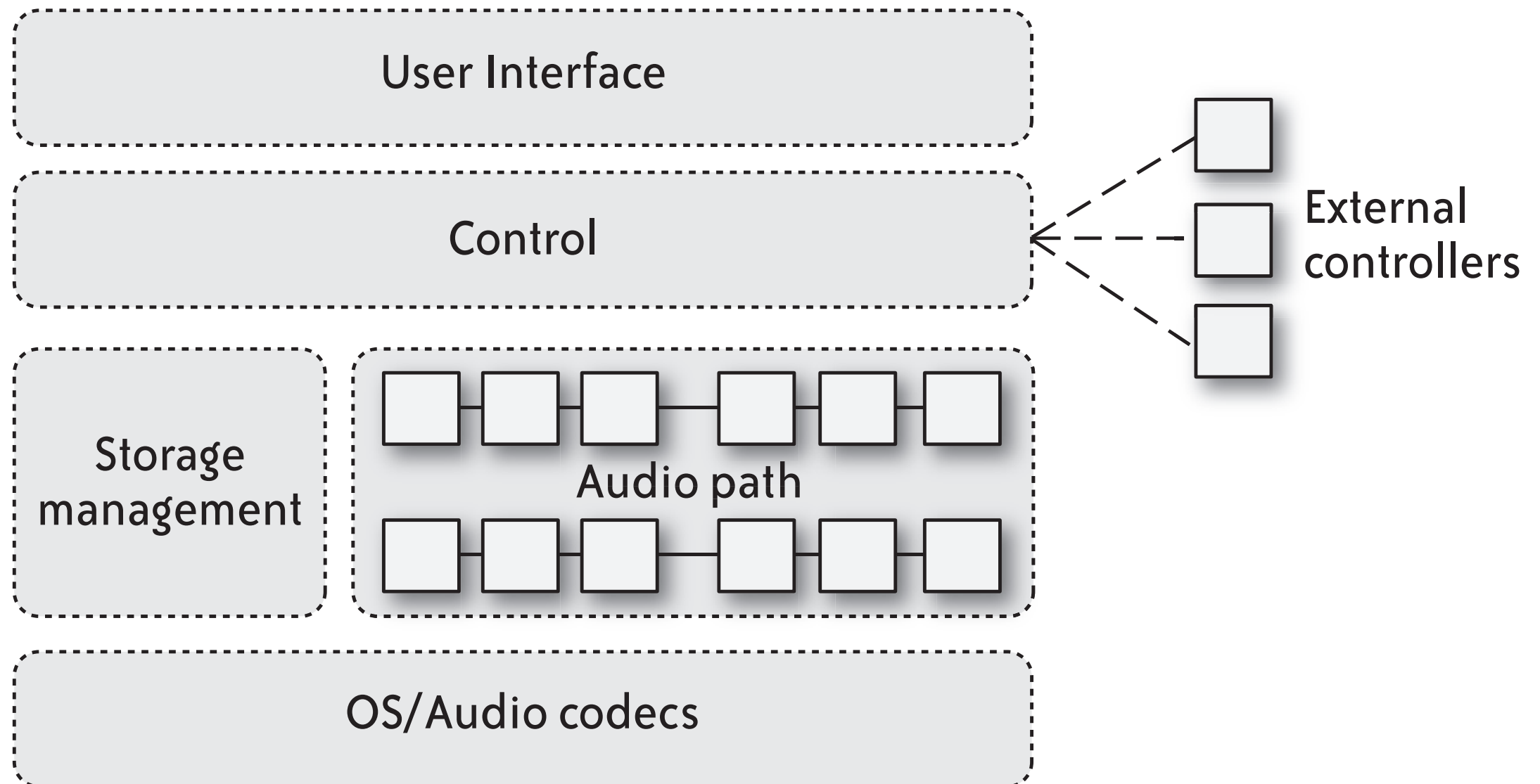
“

Bad architectural **design**

leads to further

bad architectural **design.**

Do



[Beautiful Architecture]

“

A clear architectural

design leads to a

consistent system.

[Beautiful Architecture]

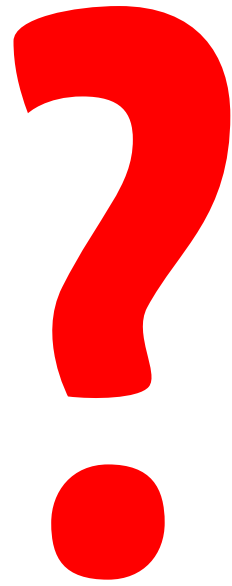
Cohesion versus Coupling

Cohesion:

"how related functionality is gathered together".

Coupling:

"Measurement of interdependency between modules".

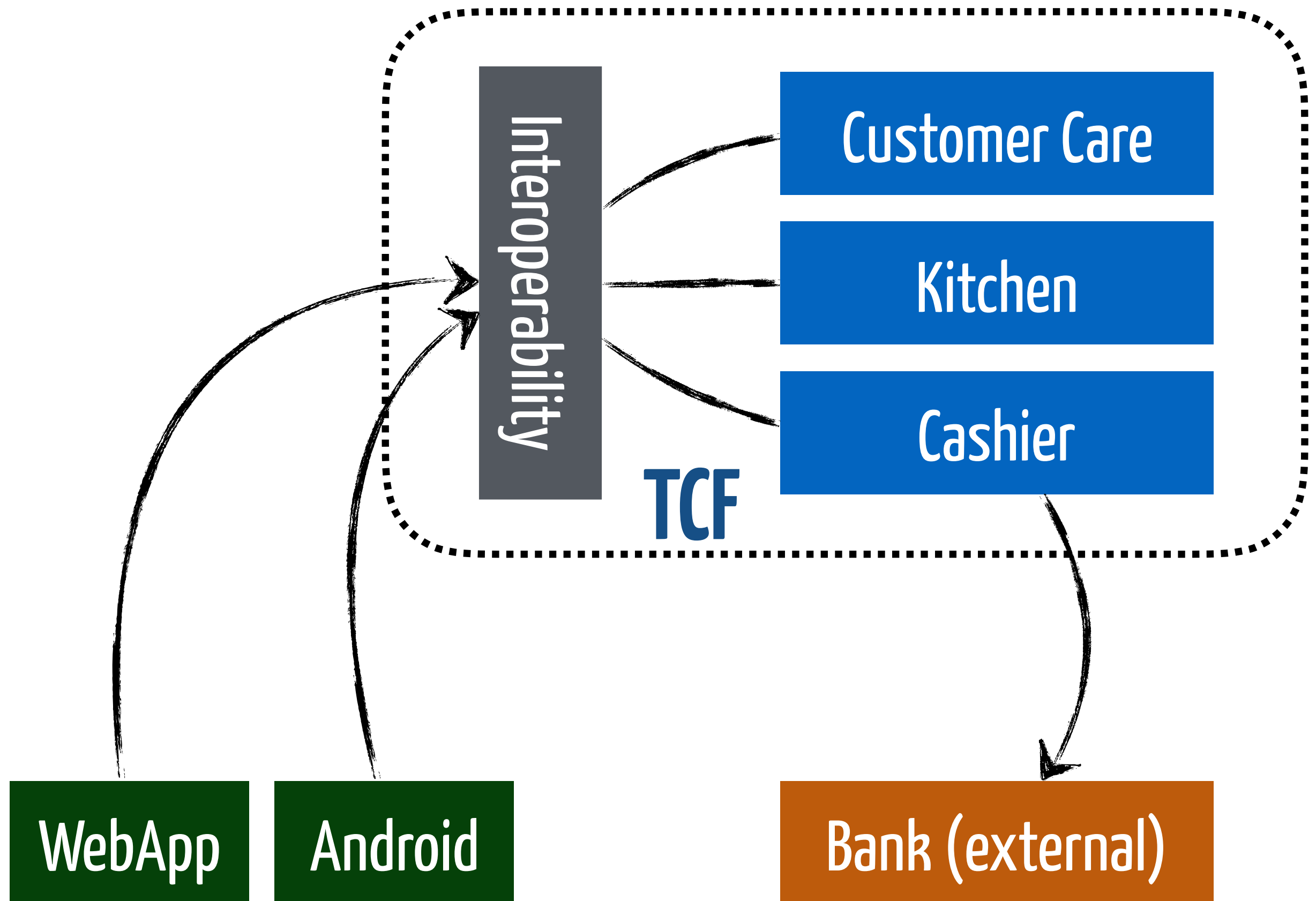


Low cohesion,
Strong coupling

Strong cohesion,
Low coupling

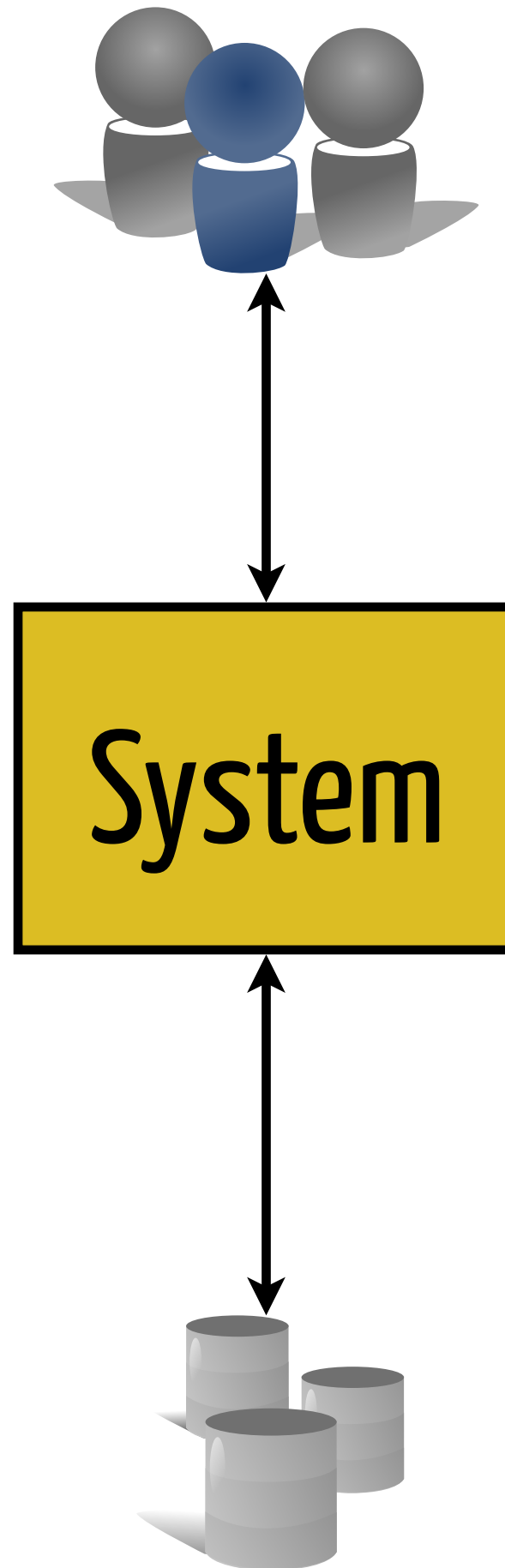


What are the “modules” in TCF?
How are they related to each others?



Layered Architectures



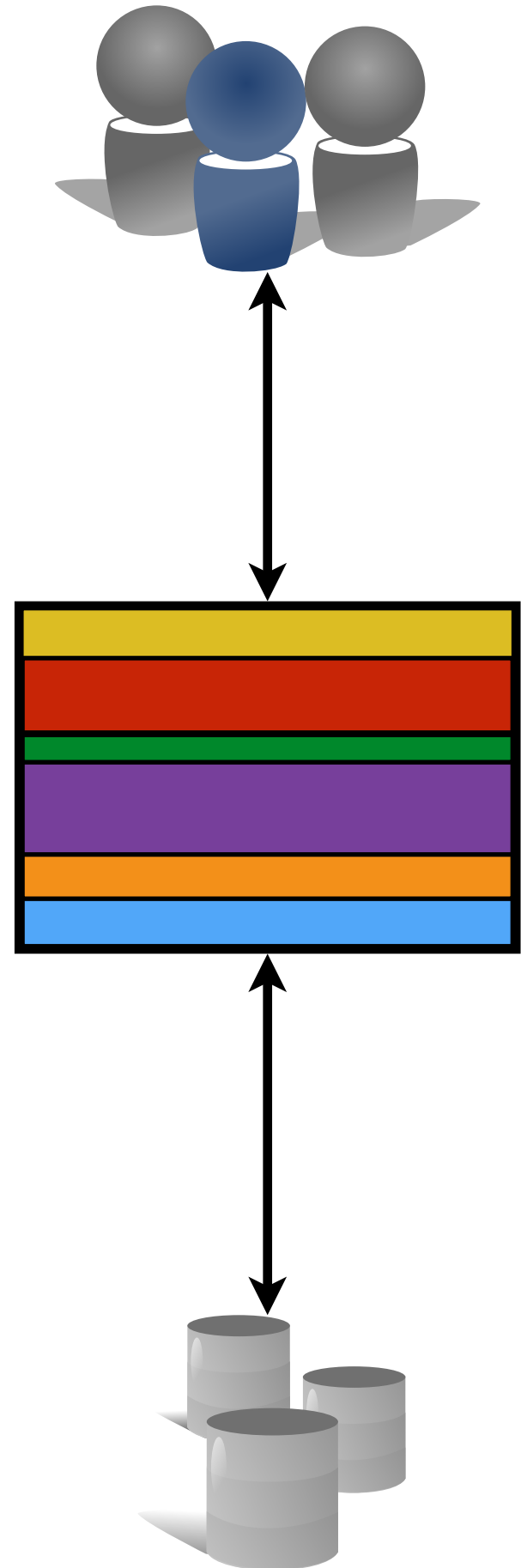


???

Layers

support

modularity



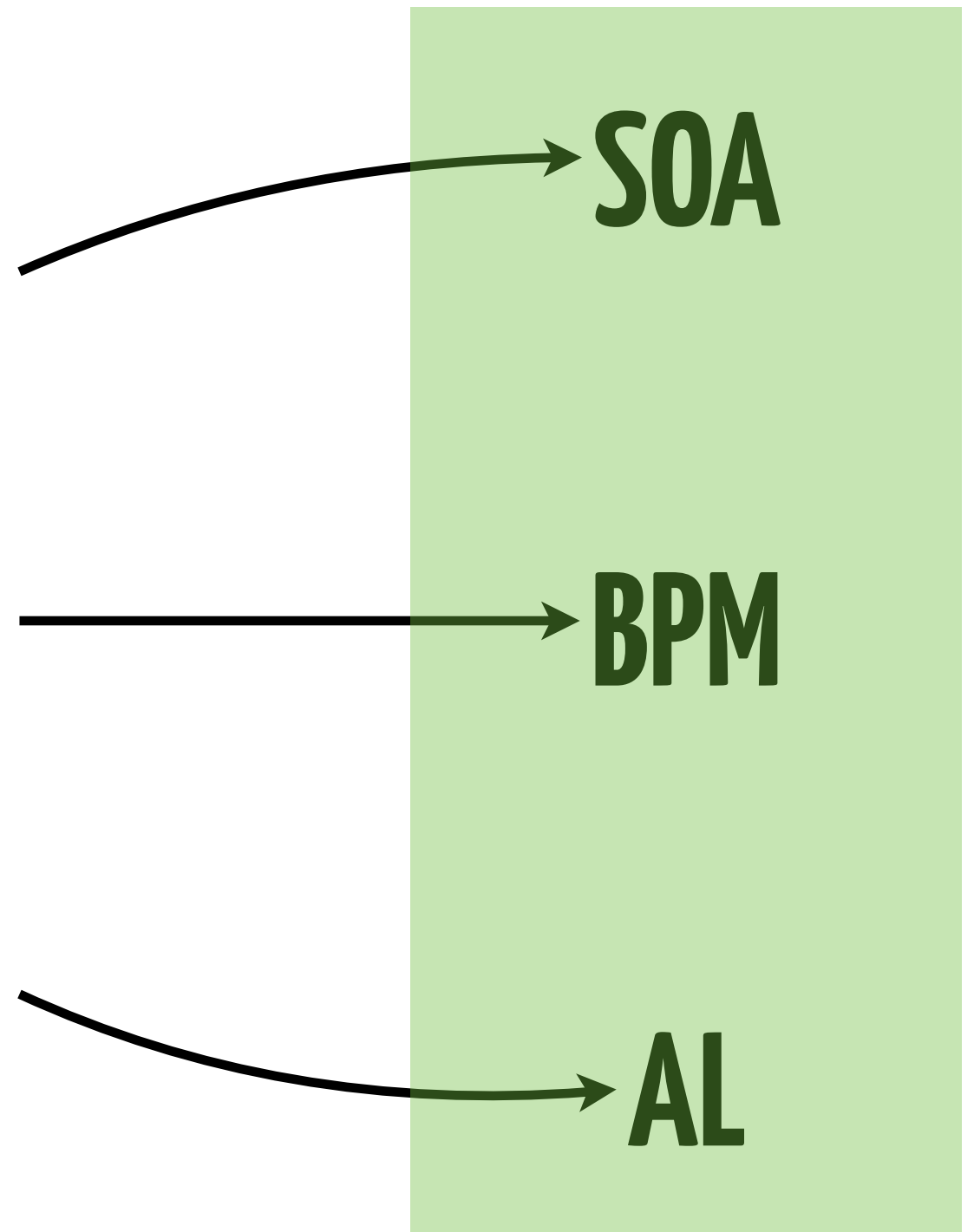
K eep

I t

S tupid,

S imple.

4A



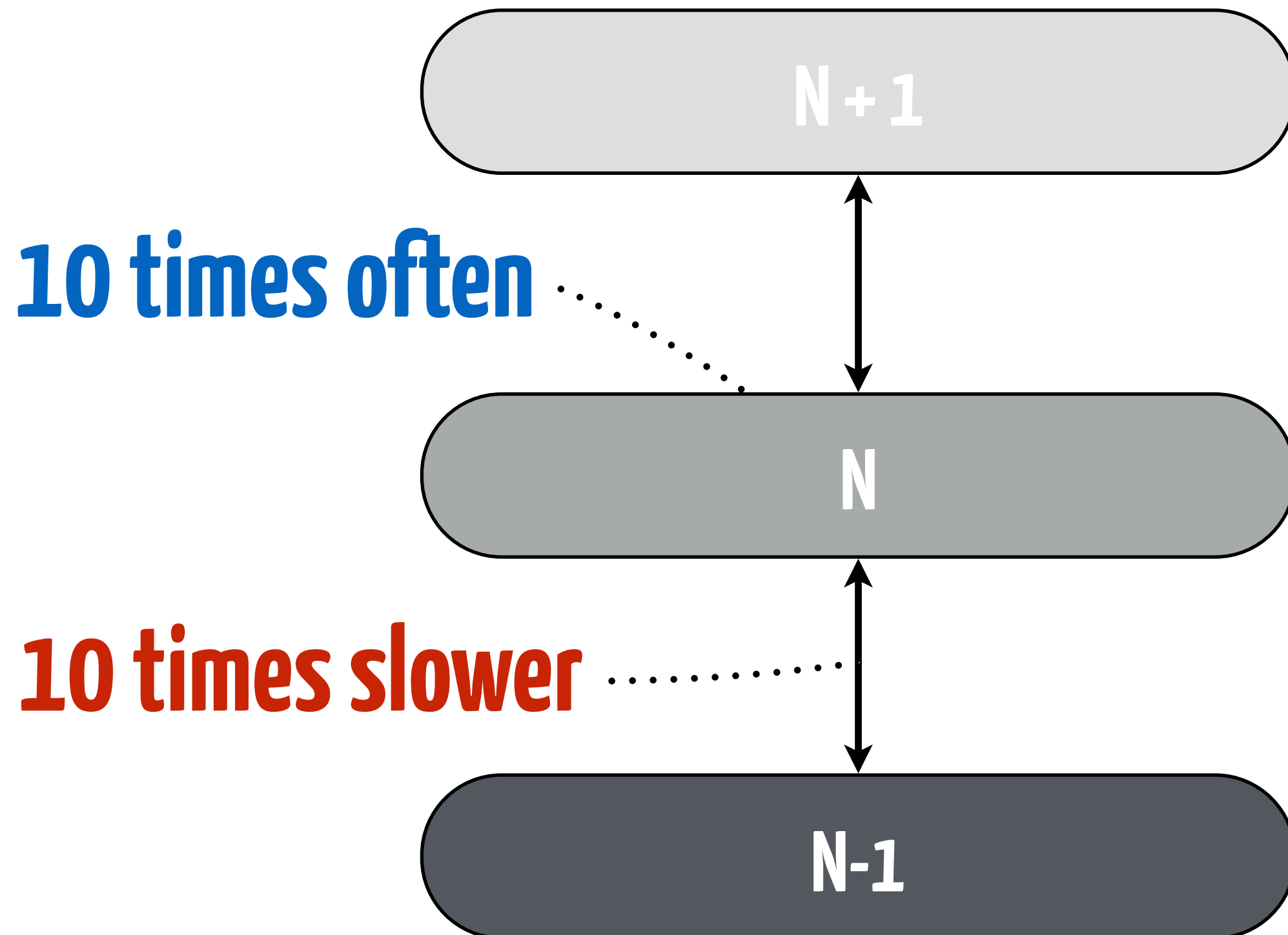
SOA

BPM

AL

5A

The rule of 10



Theory & Practice

Theory:

you know everything but nothing works

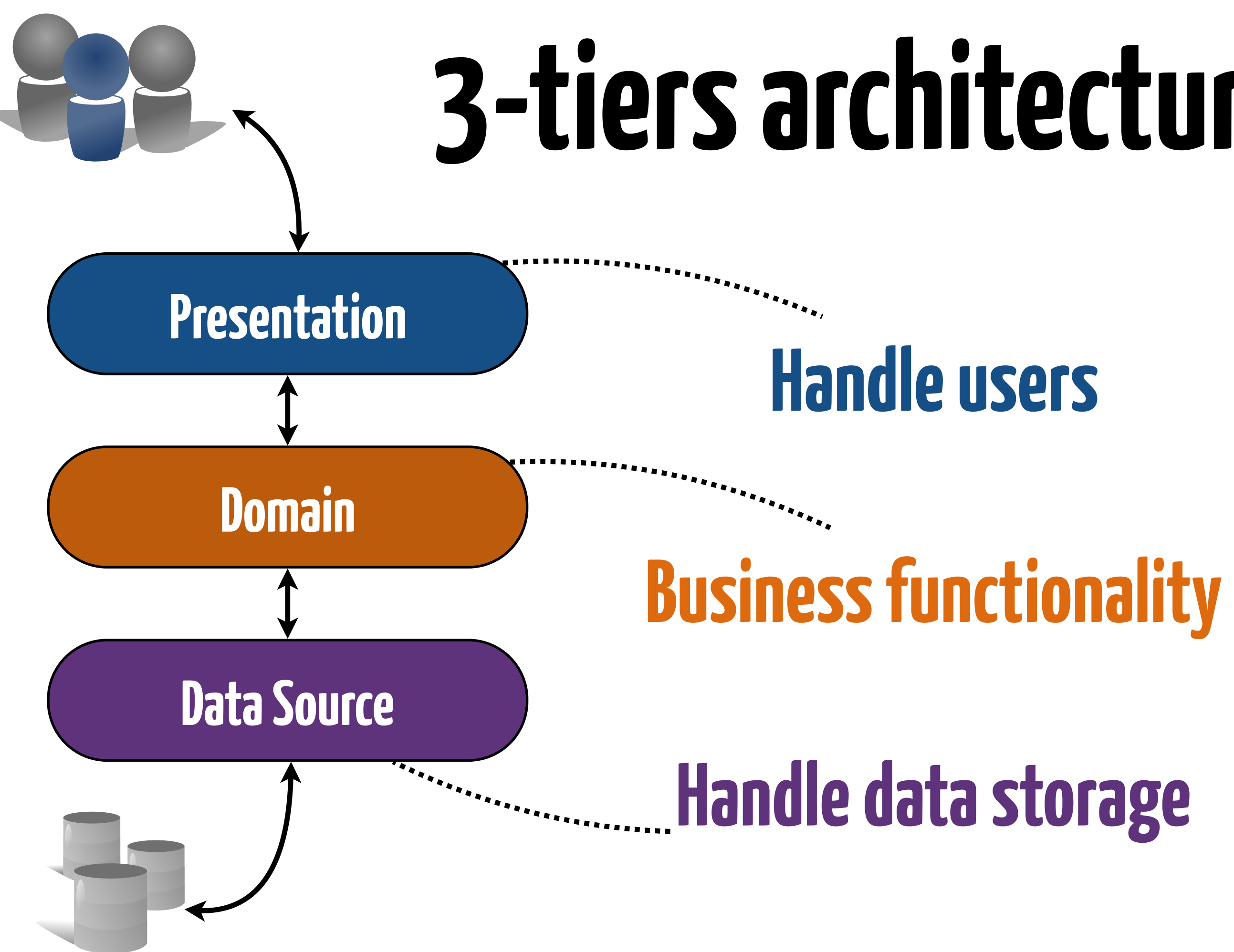
Practice:

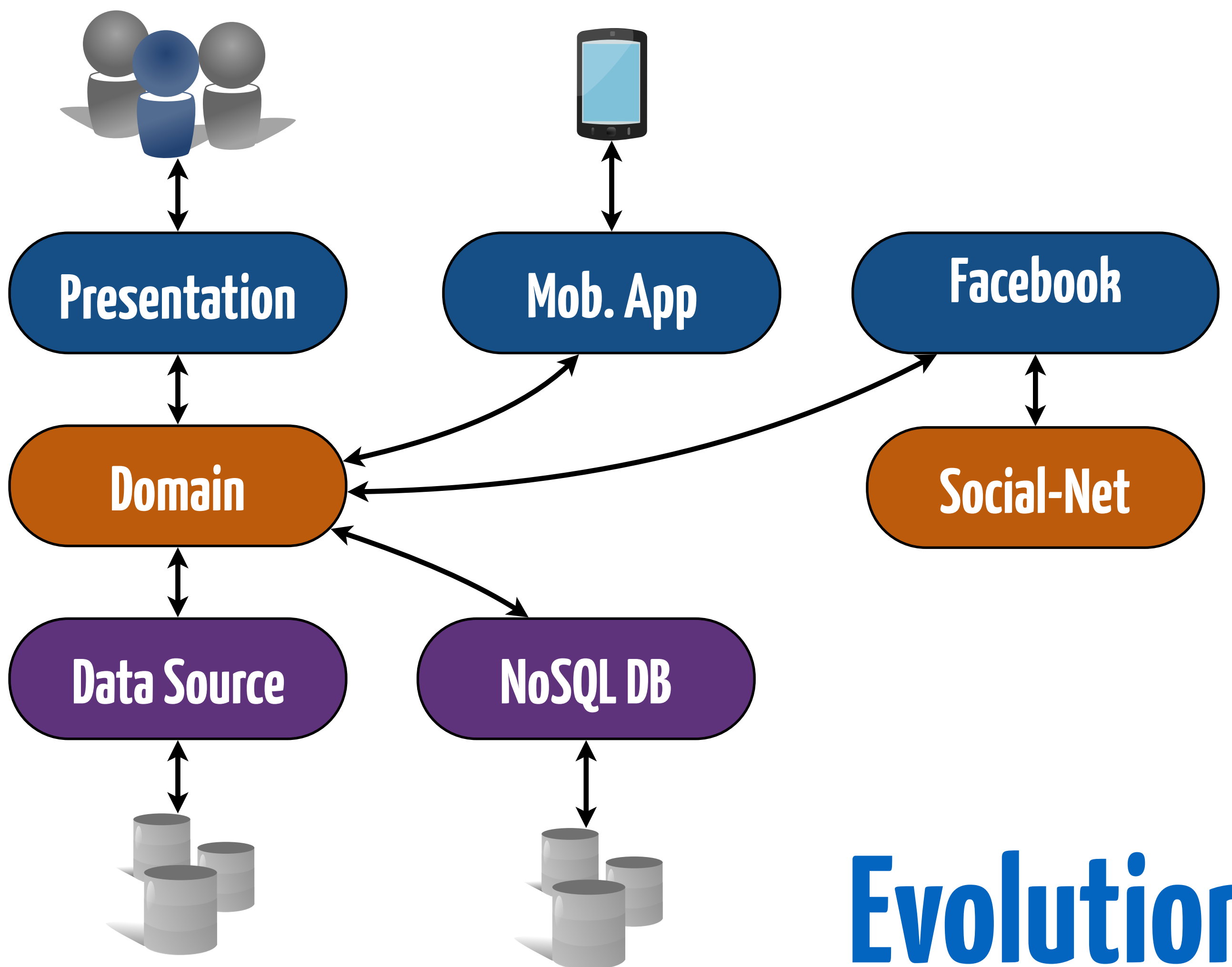
It works, but no one knows why

In theory, **N-tiers** architecture

In practice, $N = 3$
(or 5)

3-tiers architecture





Evolution



Contents of the \neq layers in TCF?
How to chose between layers?

WebApp

Android

Interoperability

Customer Care

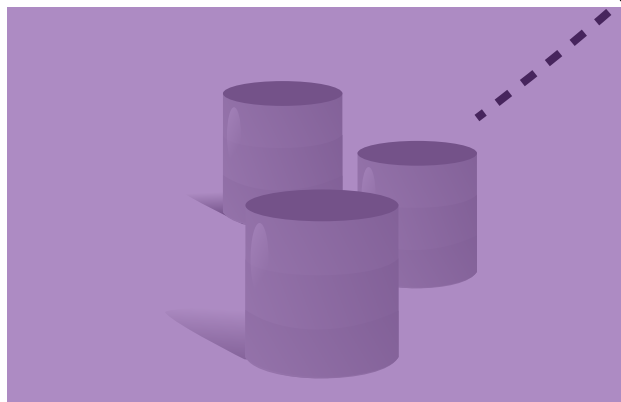
Kitchen

Cashier

Presentation

Domain

Bank (external)

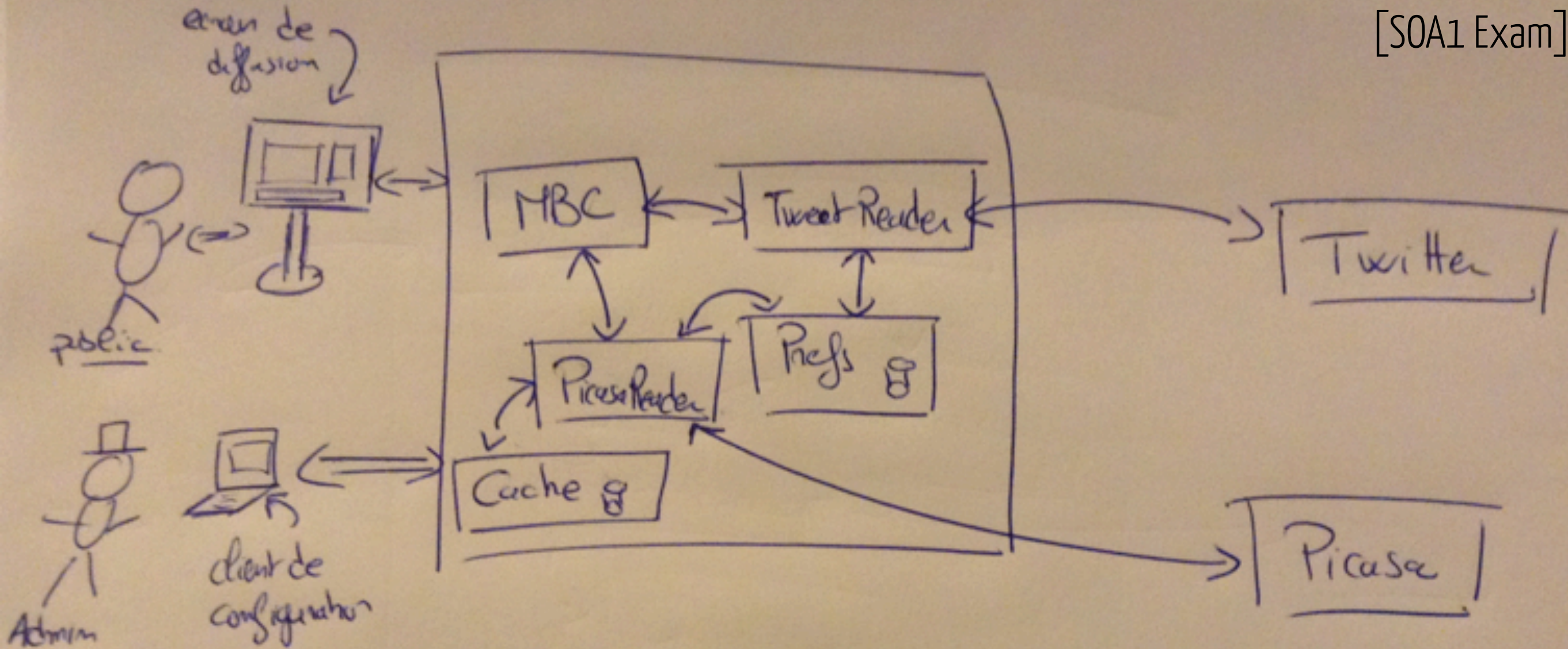


Data Source

Support from the

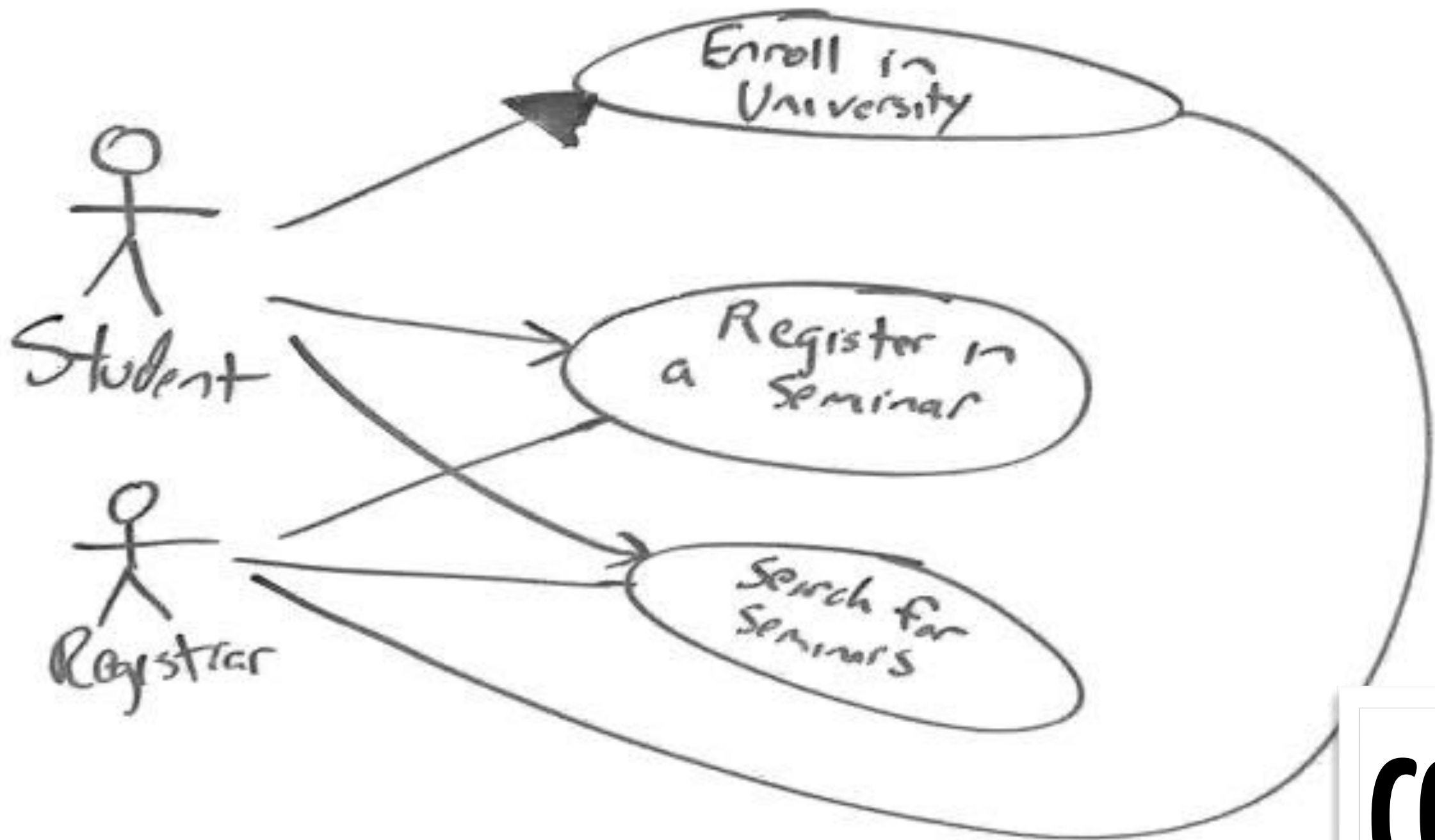


The **UML** is just a
standard syntax
for modeling



One can design a Software Architecture
without the UML

Use cases diagrams



COO



TCF MVP

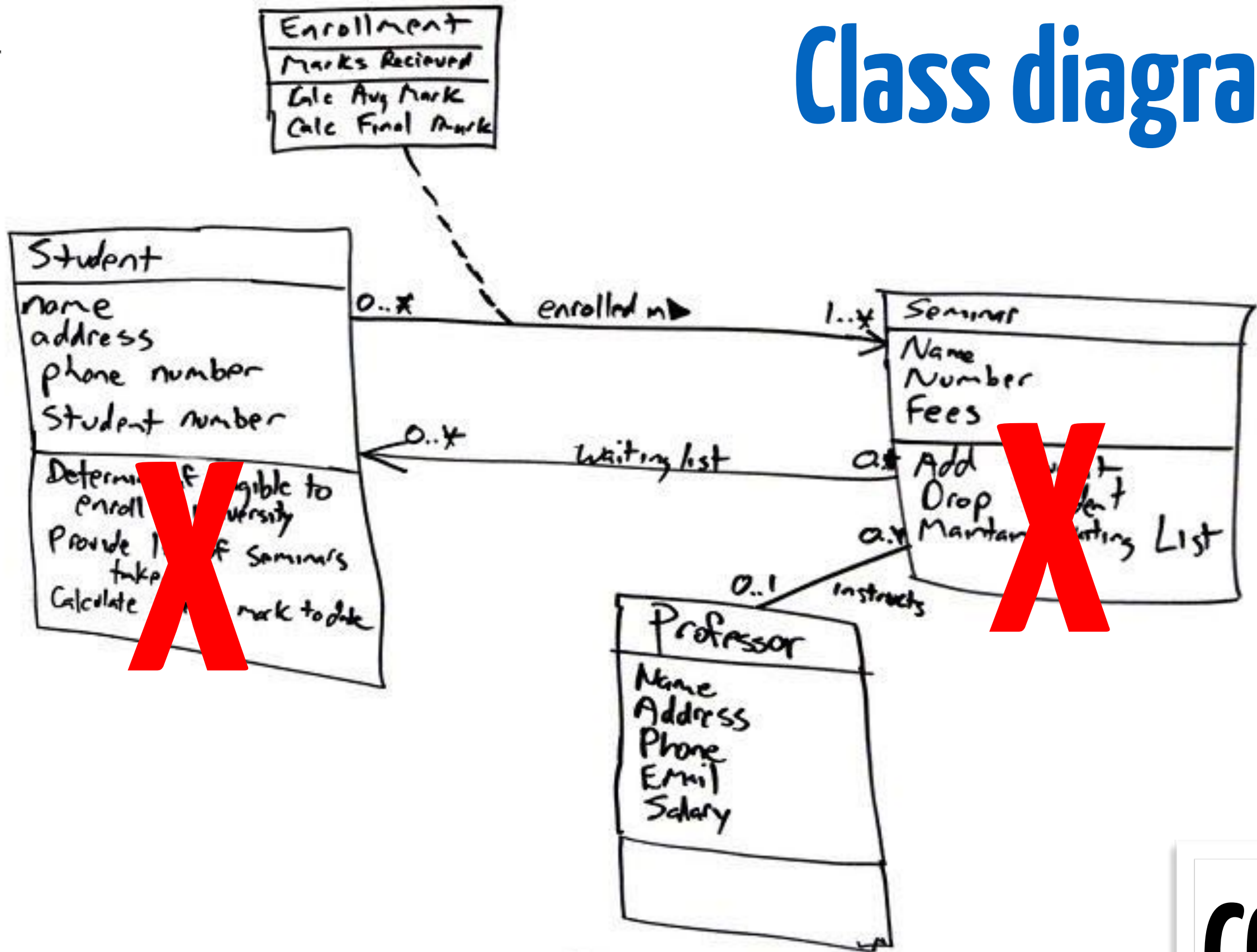
Minimal & Viable Use Case ?



customer



Class diagrams



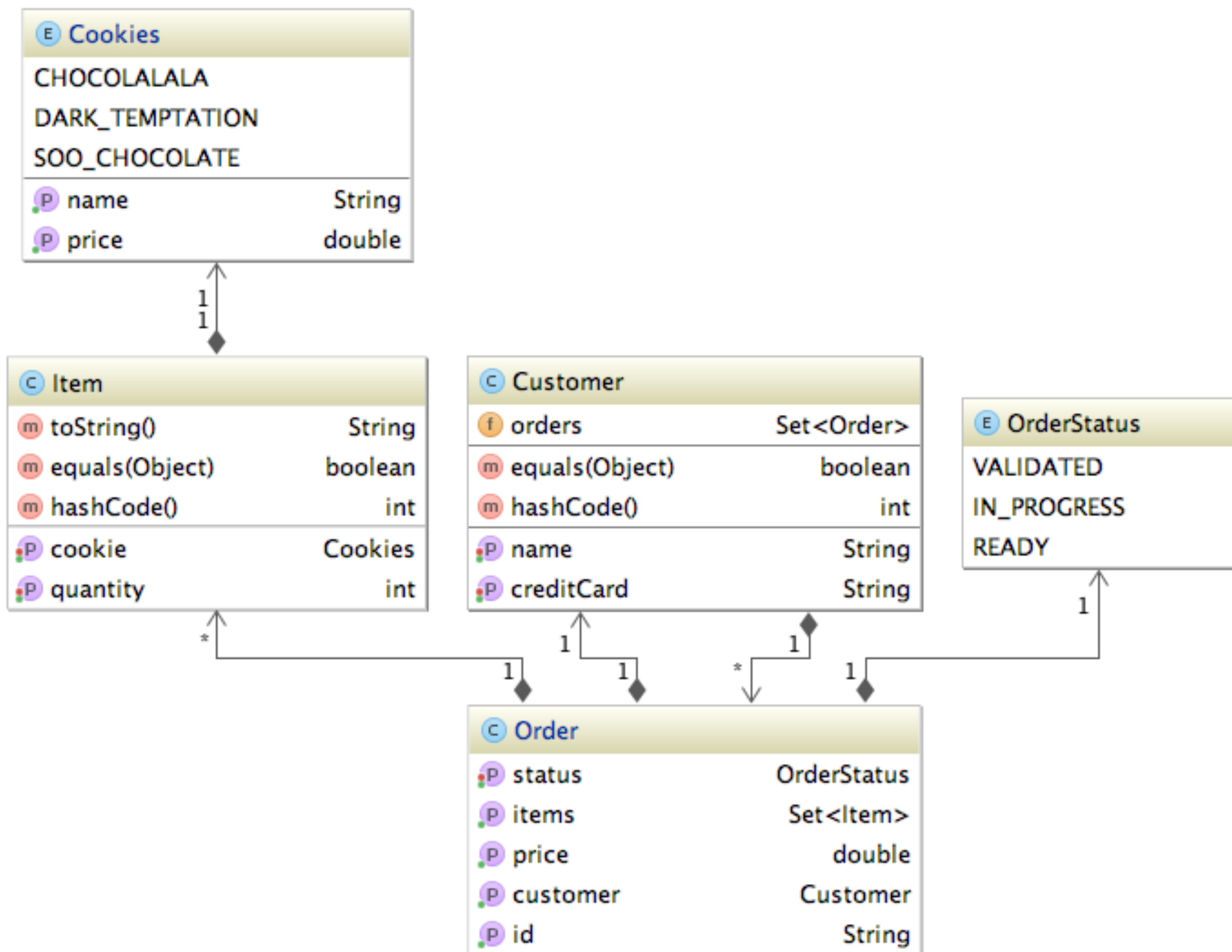
Business objects => **no methods**

COO

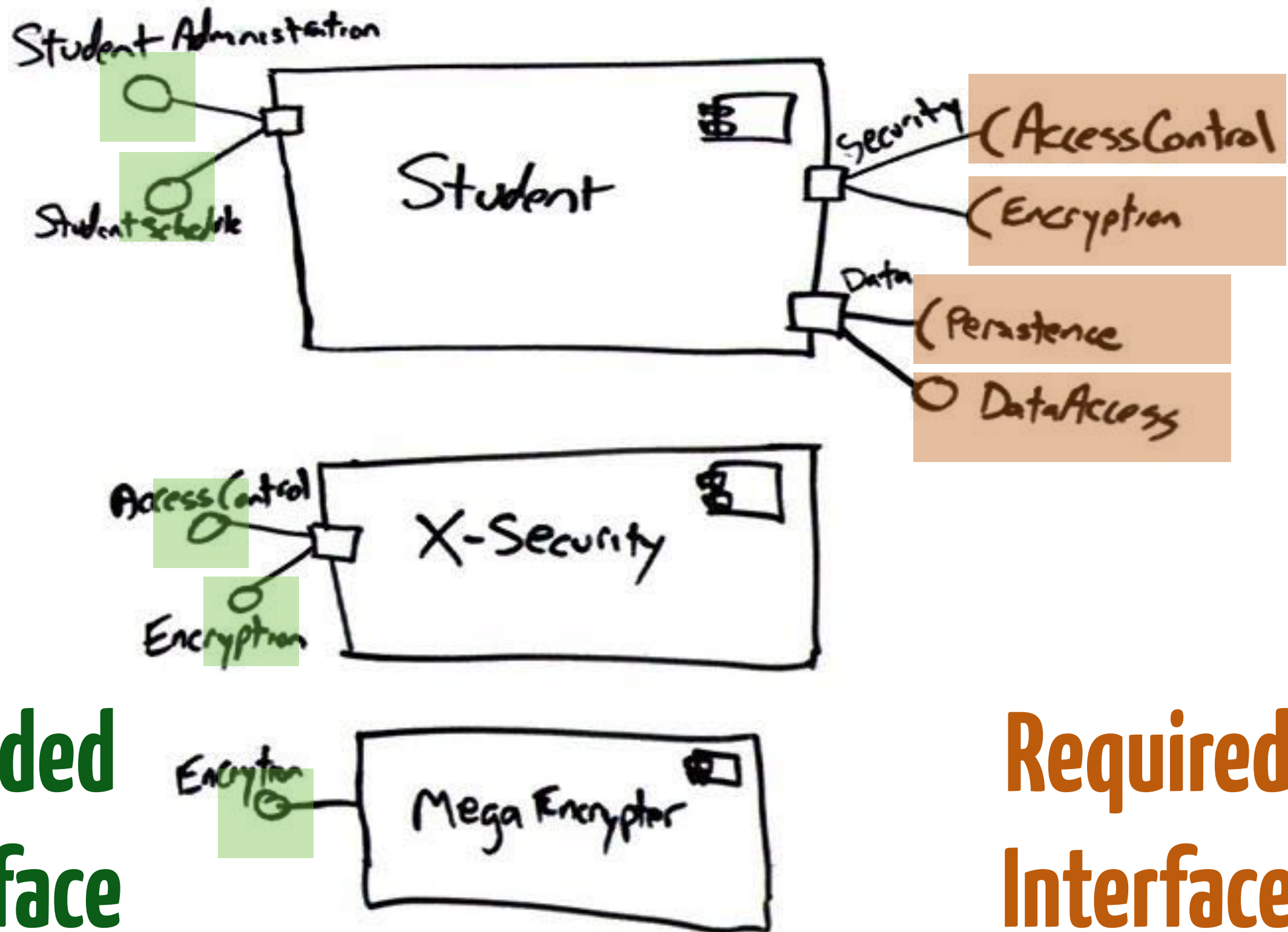


TCF MVP

Business Objects (\neq “objects” as in COO)



Components diagrams



**Provided
Interface**

**Required
Interface**

I & D of the **SOLID** principles

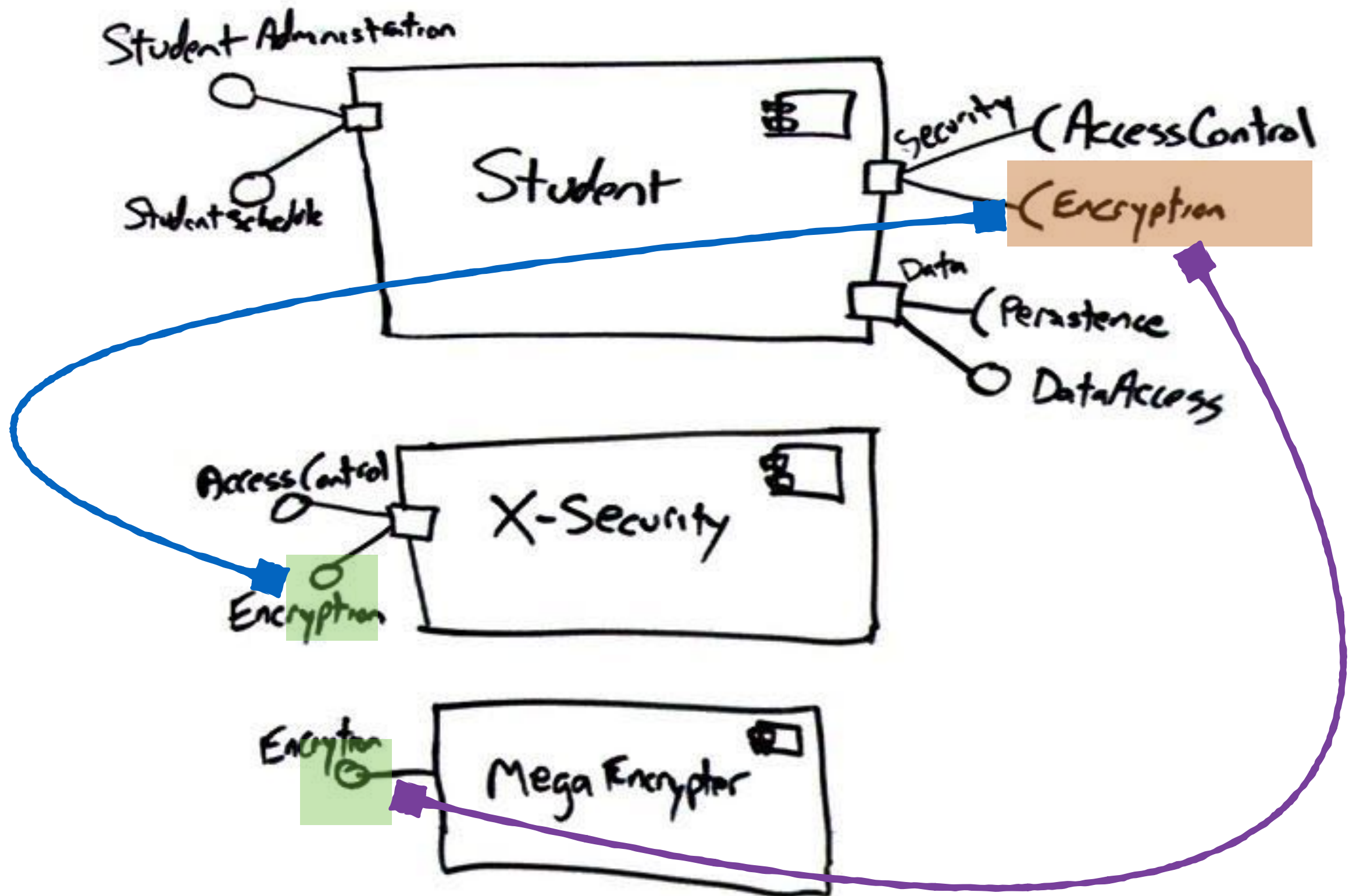
3A

Initial	Stands for	Concept
S	SRP ^[4]	Single responsibility principle a class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)
O	OCP ^[5]	Open/closed principle "software entities ... should be open for extension, but closed for modification."
L	LSP ^[6]	Liskov substitution principle "objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program." See also design by contract .
I	ISP ^[7]	Interface segregation principle "many client-specific interfaces are better than one general-purpose interface." ^[8]
D	DIP ^[9]	Dependency inversion principle one should "depend upon abstractions, [not] concretions." ^[8]

4A

[https://en.wikipedia.org/wiki/SOLID_\(object-oriented_design\)](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design))

Binding Components



Example of Implementation

```
class Student implements  
    StudentAdministration, StudentSchedule {  
  
    Encryption e = new MegaEncrypter(...)  
  
}
```

```
class Student implements  
    StudentAdministration, StudentSchedule {  
  
    AssemblyContext ctx = ...  
    Encryption e = ctx.inject(Encryption.class)  
  
}
```

Teaser: Annotation-based injection

Provided Interface

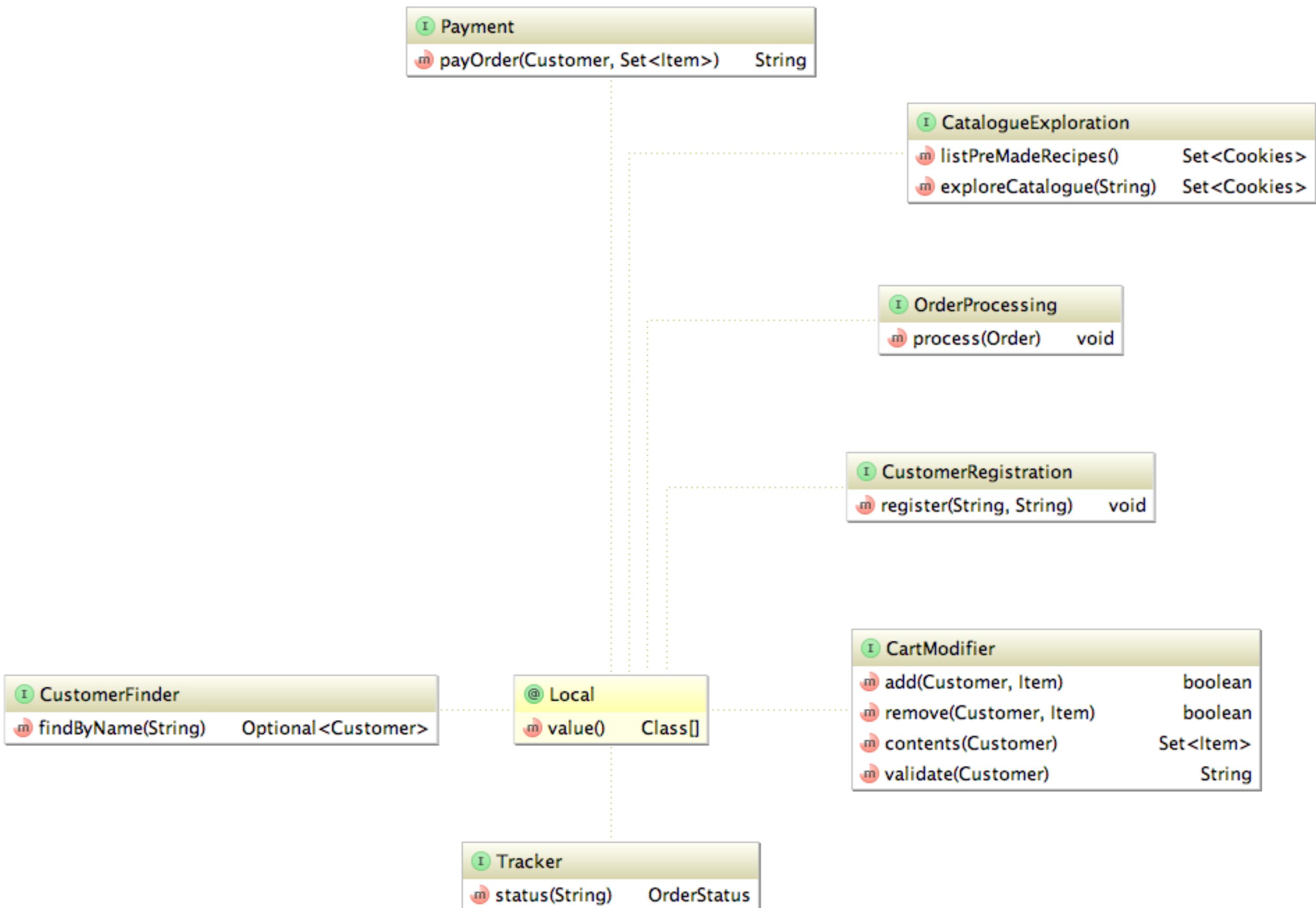
```
class Student implements  
    StudentAdministration, StudentSchedule {  
  
    @Inject  
    private Encryption e;  
  
}
```

Required Interface

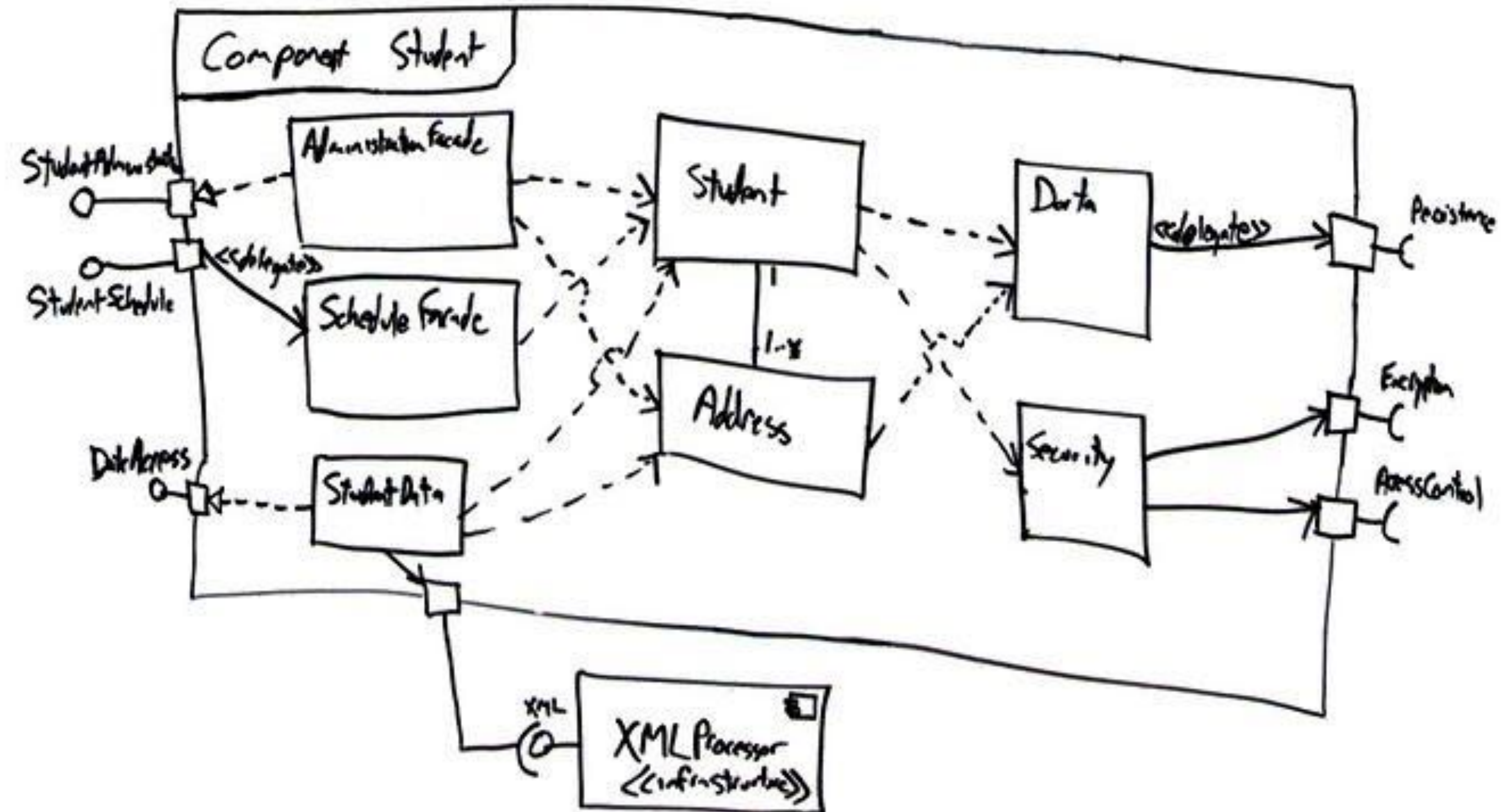


TCF MVP

Functional Interfaces for TCF ? Components ?



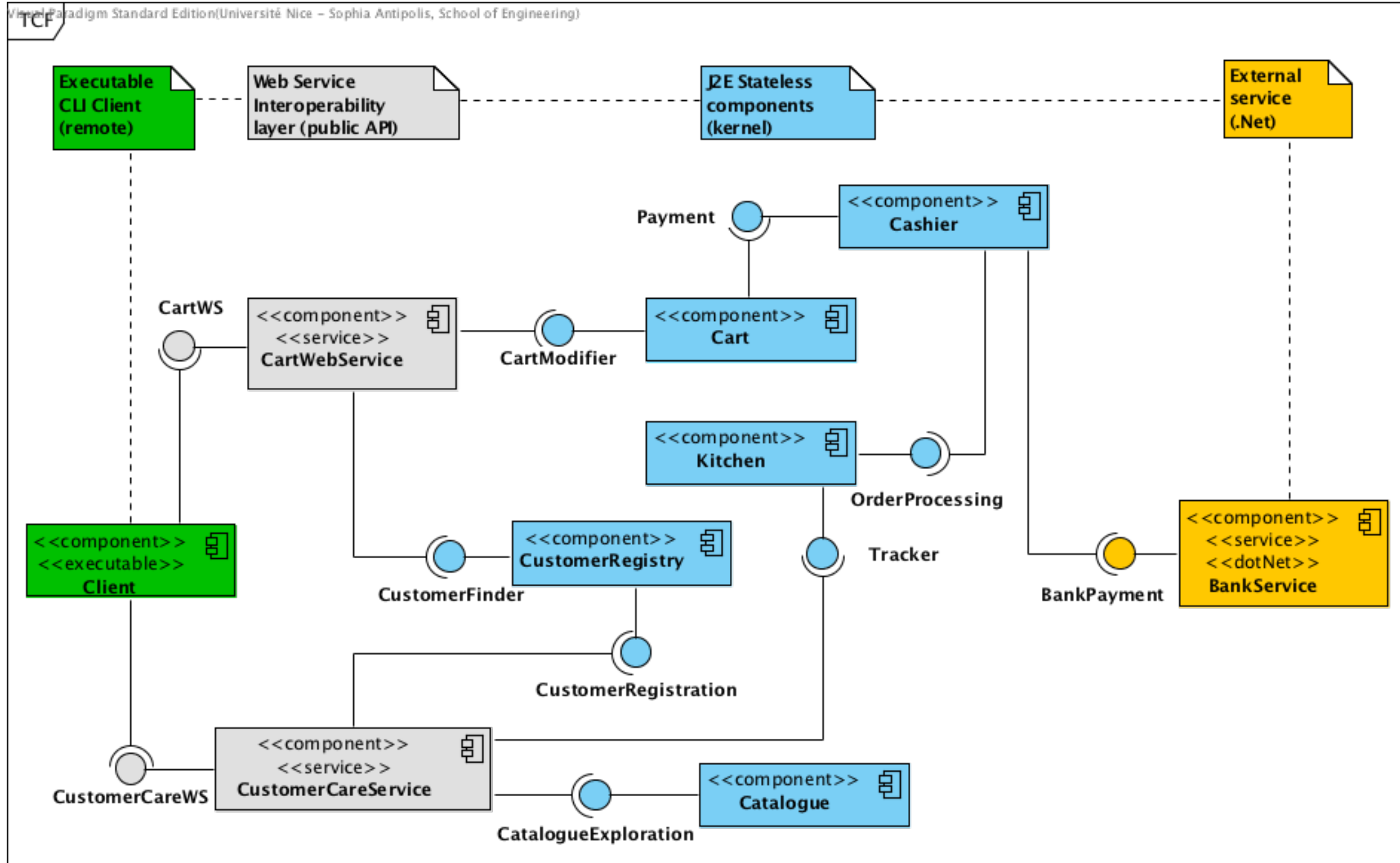
Components Assembly



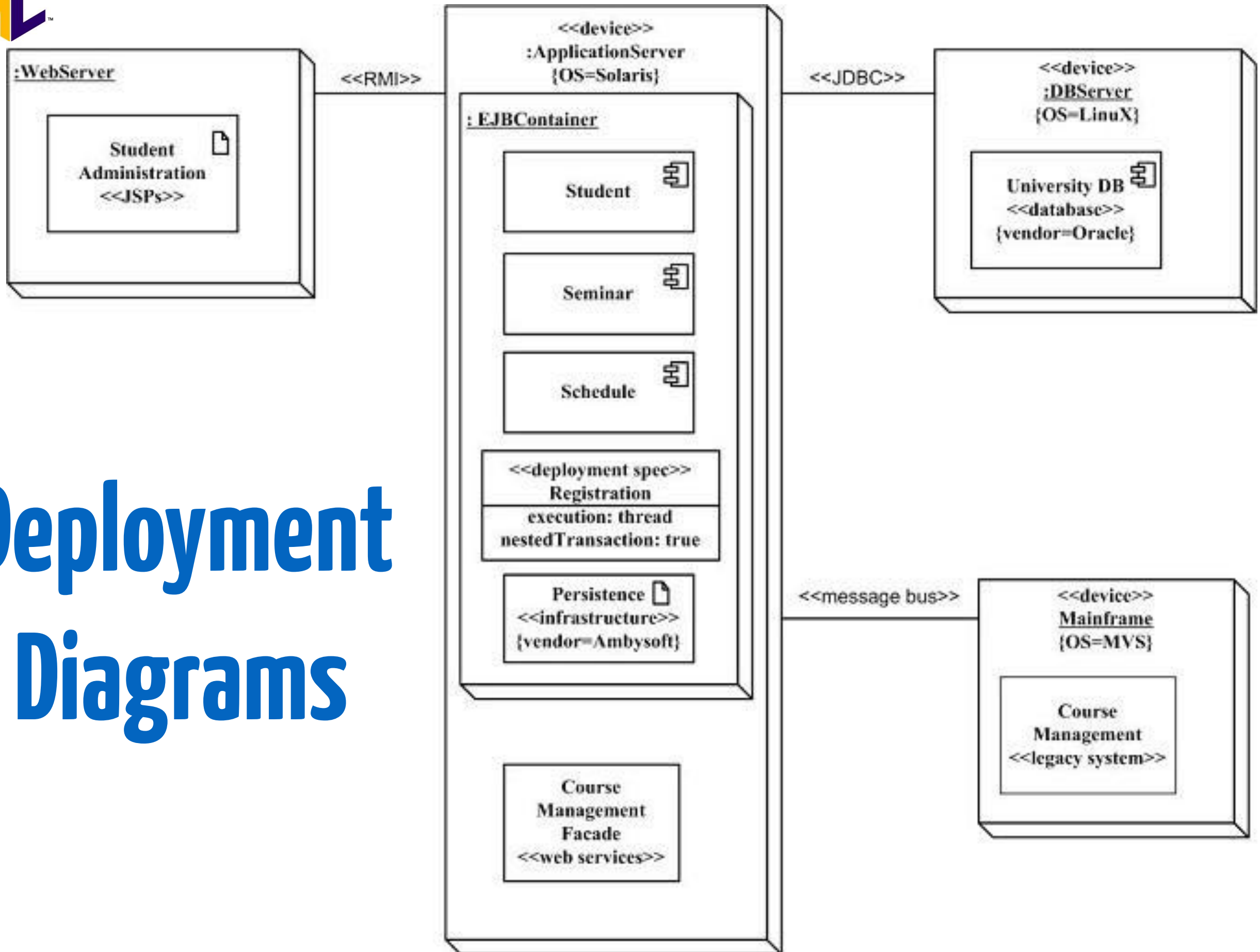


TCF MVP

Component Assembly ?



Deployment Diagrams





TCF MVP

How to deploy TCF ?

