Architectural **Viewpoints**

AM Dery
Fortement inspirée des cours de
S Mosser

pictures: sxc.hu

A **complex system** is much more **effectively** described by **a set of interrelated views** […] than by **a single overloaded model**

[SSA]

A **view** is a representation of one or more **structural aspect** of an **architecture**

[SSA]

A **viewpoint** is a **collection** of **patterns**, **templates**, and **conventions** for **constructing one type of view**

SSA]

Viewpoints and views: Pros

Separation of concerns

Improved developer focus

Communication with stakeholders

Management of complexity

Viewpoints and views: Pitfalls

Inconsistency

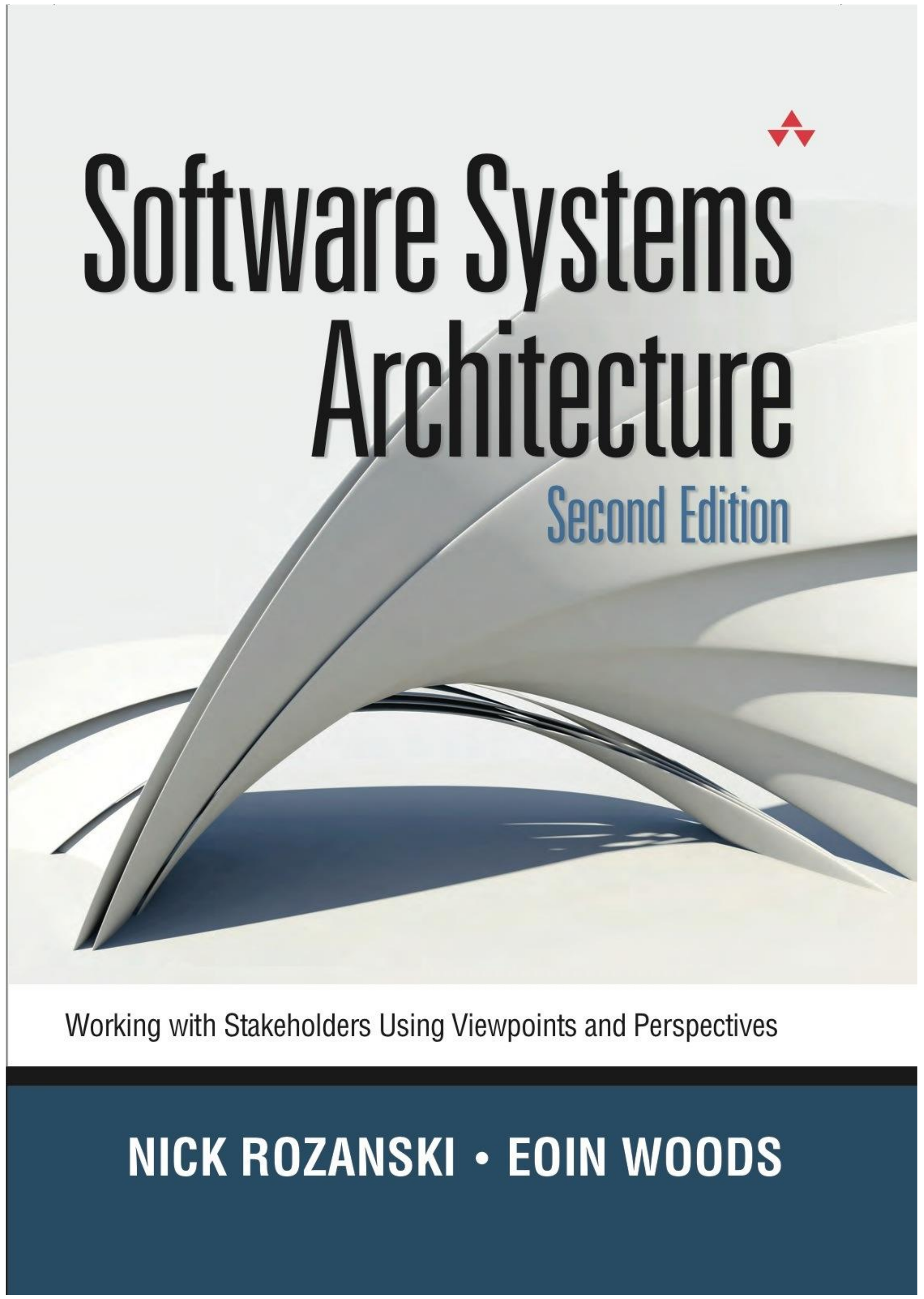
Selection of the wrong set of views

Fragmentation

SSA]

Bibliography

- Chapters
  - **#3**: Viewpoints and Views
  - #**17**: Functional Viewpoint
  - **#20**: Development Viewpoint
  - **#21**: Deployment viewpoint


Software Systems Architecture
Second Edition
Working with Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI • EOIN WOODS

[SSA, 2011]

1

Functional Viewpoint

Definition

Describes the system's runtime **functional elements** and their **responsibilities**, **interfaces** and **primary interactions**

Démarche

1. Analyse fonctionnelle : **contours du système à développer**

Démarche

2. Conception du système global

Identification des services métiers requis et fournis

Choix des systèmes externes avec lesquels interagir

Démarche

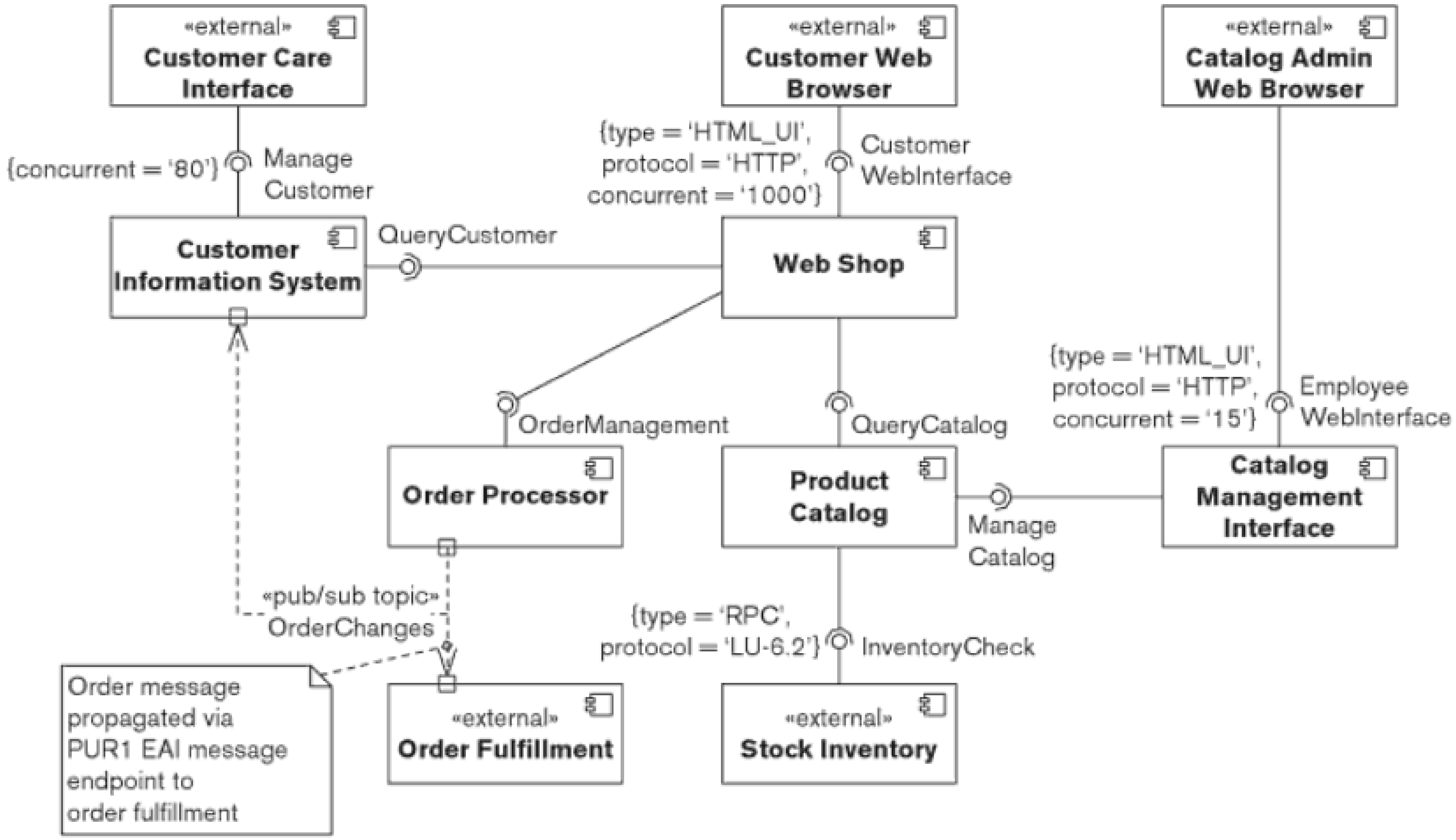
3. Conception du cœur système

Identification des composants métiers

Identification des bindings entre composants

Identification des objets métiers

UML Component Diagram as a support


«external» Customer Care Interface
{concurrent = '80'}
Manage Customer
«external» Customer Web Browser
{type = 'HTML_UI', protocol = 'HTTP', concurrent = '1000'}
Customer WebInterface
«external» Catalog Admin Web Browser
Customer Information System
QueryCustomer
Web Shop
{type = 'HTML_UI', protocol = 'HTTP', concurrent = '15'}
Employee WebInterface
OrderManagement
QueryCatalog
Order Processor
Product Catalog
Catalog Management Interface
Manage Catalog
«pub/sub topic» OrderChanges
{type = 'RPC', protocol = 'LU-6.2'}
InventoryCheck
Order message propagated via PUR1 EAI message endpoint to order fulfillment
«external» Order Fulfillment
«external» Stock Inventory

SSA]

Elicitation process

Requirements $\longrightarrow$ Components

1. Identify the elements
2. Assign responsibilities
3. Design the interfaces
4. Design the connectors
5. Check functional traceability
6. Walk through common scenarios
7. Analyse the interactions
8. Analyse for flexibility

SSA]

De bonnes interfaces ?

Review often to assess understandability

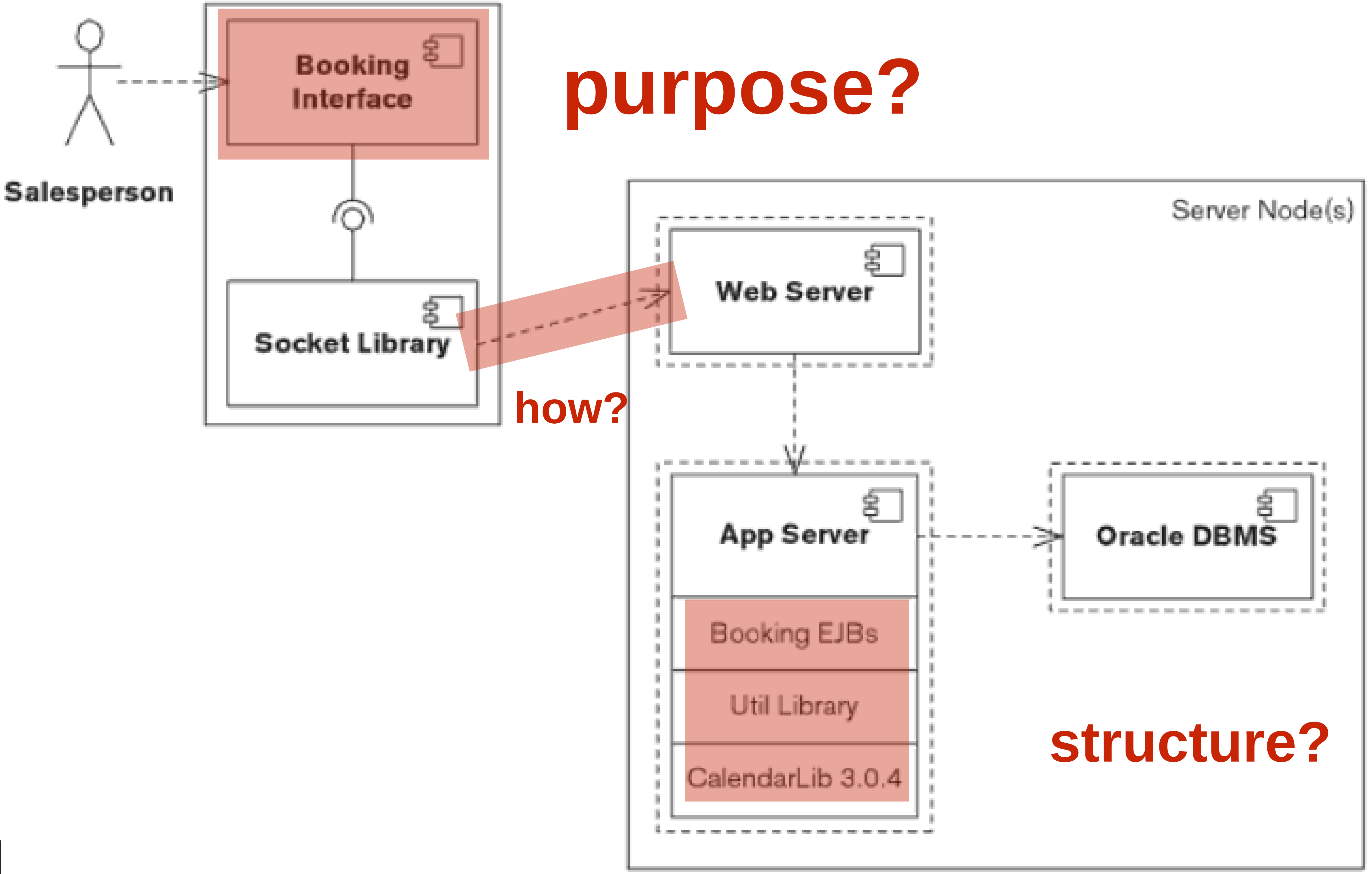
Define interfaces and connectors **ASAP**

Bind **operations**, **semantics** and **examples**

Interface design ⇒ definition **completion**

SSA]

Eviter la redondance Client Serveur



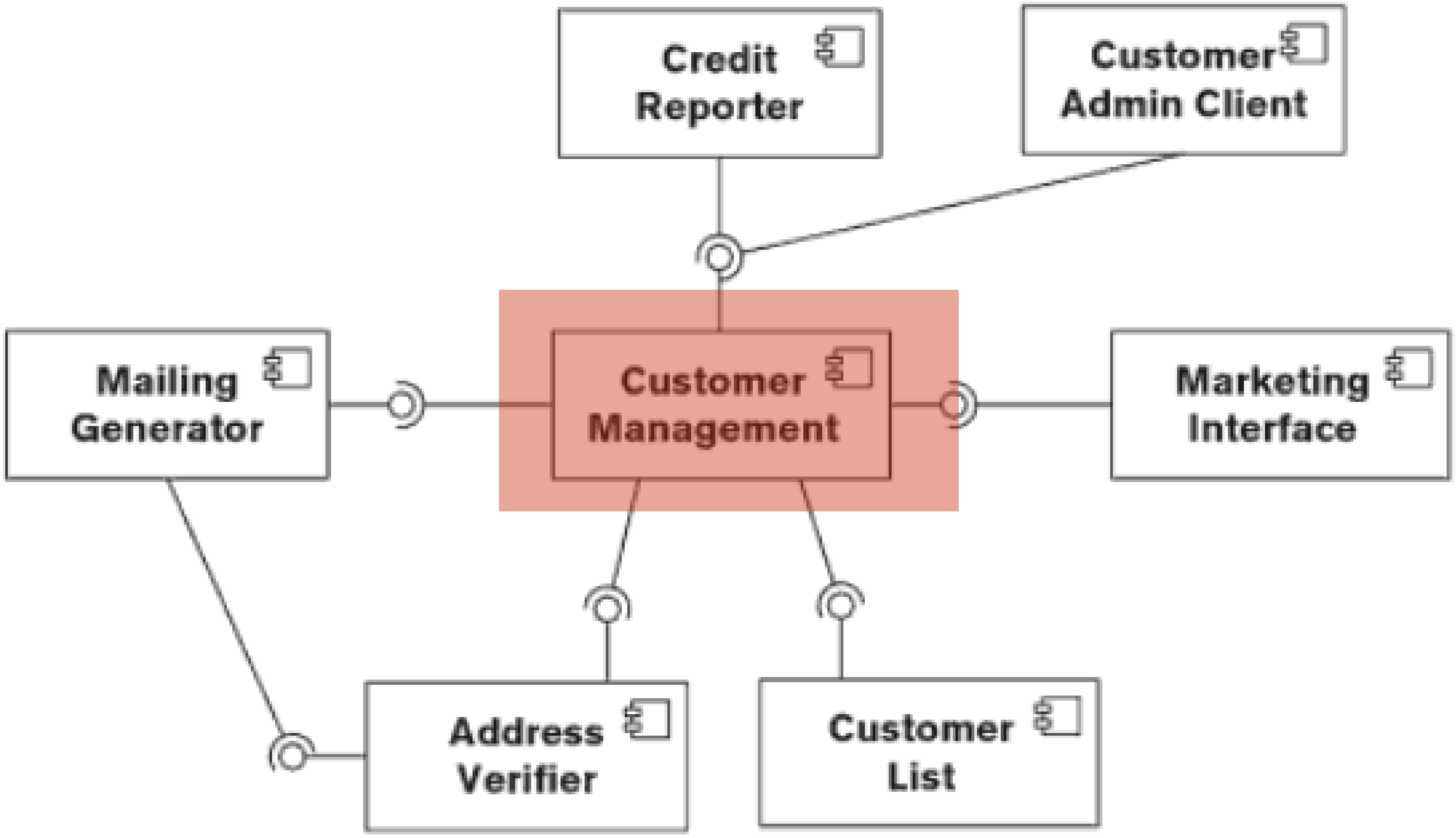
Bon niveau de granularité

At most **10 elements** / view

Divide into a "**system of systems**"

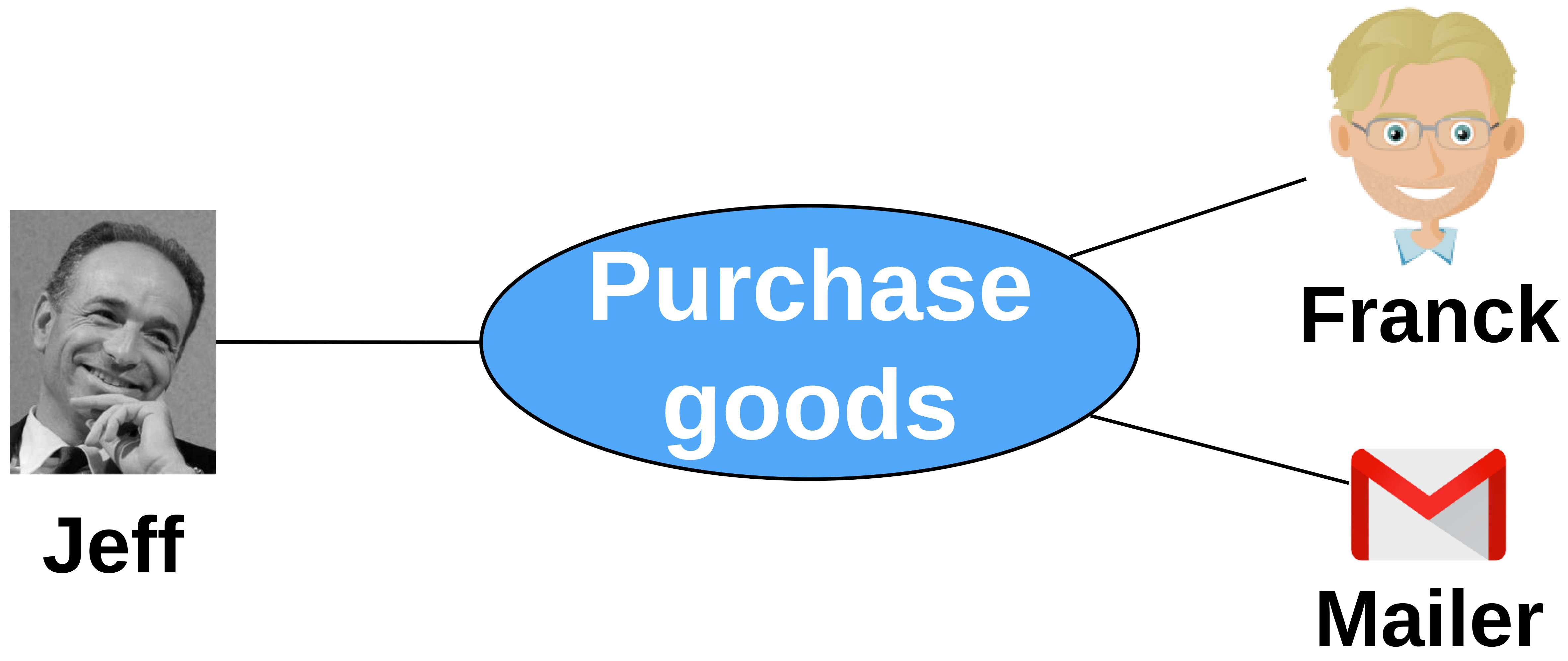
SSA]

Eviter les goulots d'étranglement : composant Dieu



SSA] Contraposition: too many dependencies

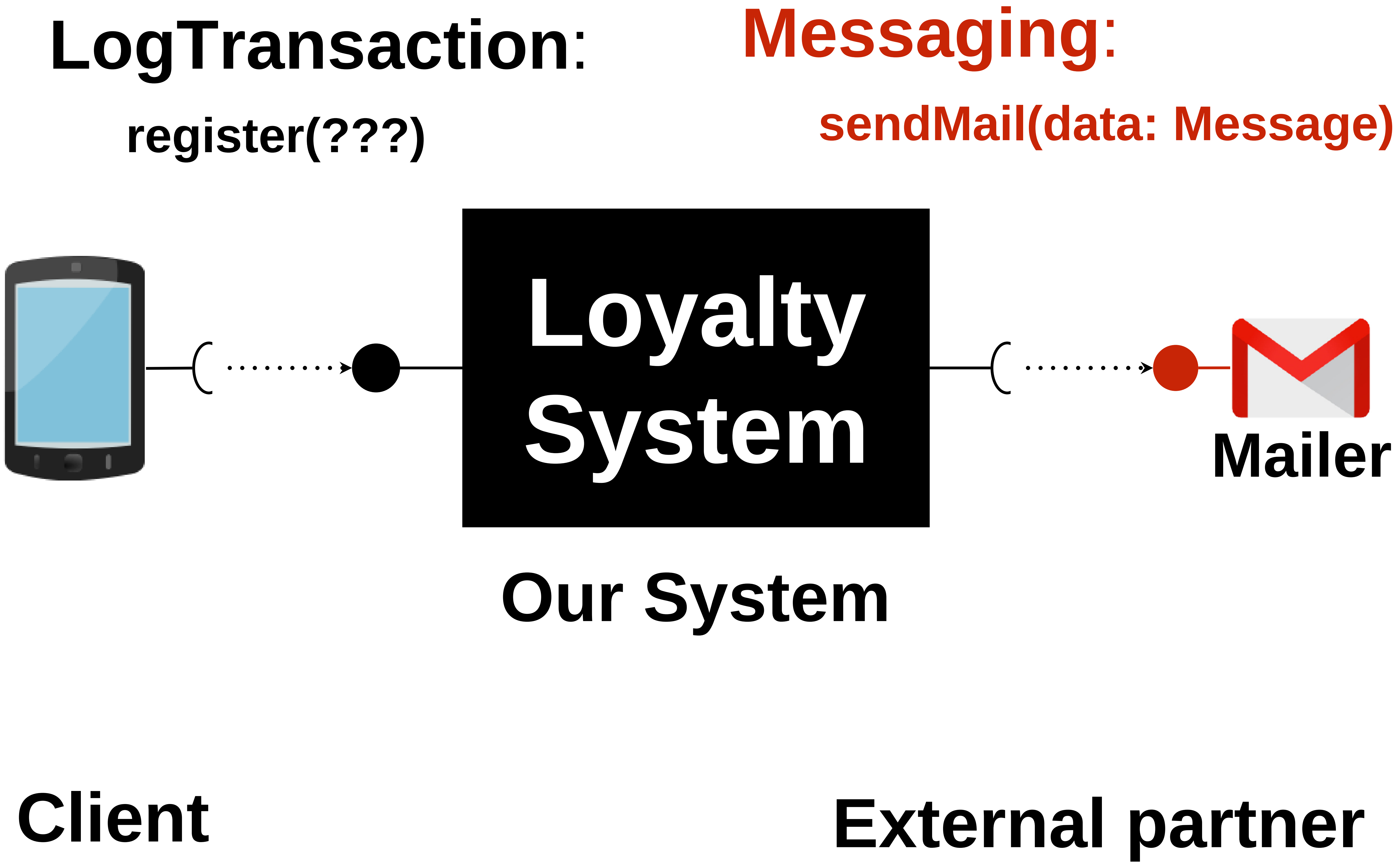
Exemple

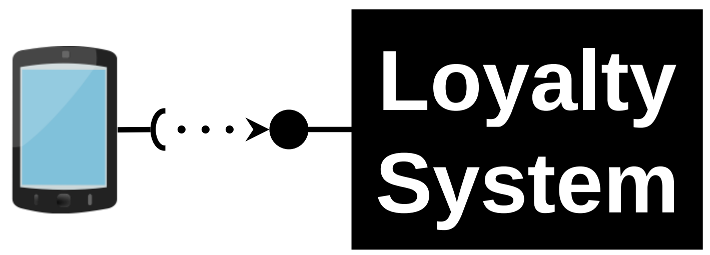


As Jeff (Customer),
I want to log a purchase on my card
So that I know my loyalty credit increase

Scenario: Purchase Goods (MVP)

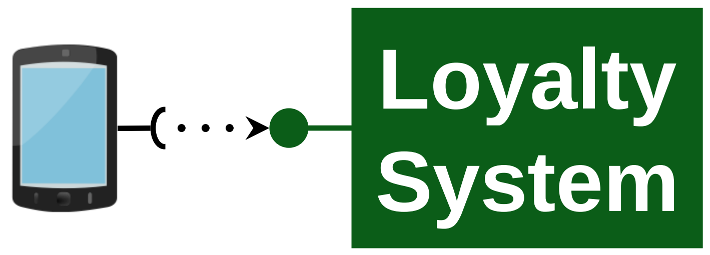
1. Jeff (a Customer) presents a loyalty card and the goods to be purchased;
2. Franck (a Dealer) scans the card, and logs the purchase information;
3. These data are sent to the Loyalty System;
4. The purchase amount is transformed into Loyalty credit points;
5. This amount is added to the balance of the customer (based on the card ID);
6. An email is sent to the user email with the new balance.


LogTransaction:
register(???)
Messaging:
sendMail(data: Message)
Loyalty System
Mailer
Our System
Client
External partner



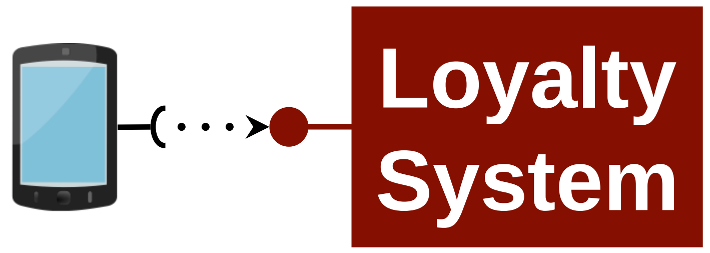
LogTransaction:

register(shop: Shop, card: Image, prod: Product, quantity: Int)



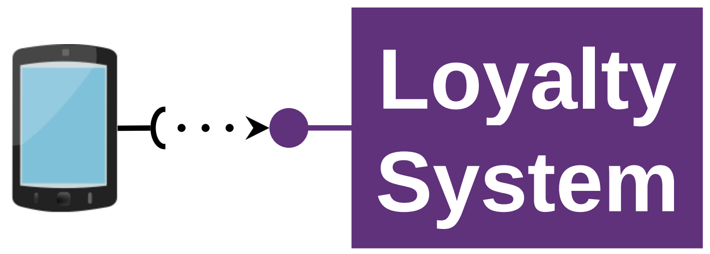
LogTransaction:

register(shop: ID, card:ID, product: Product, value: Float)



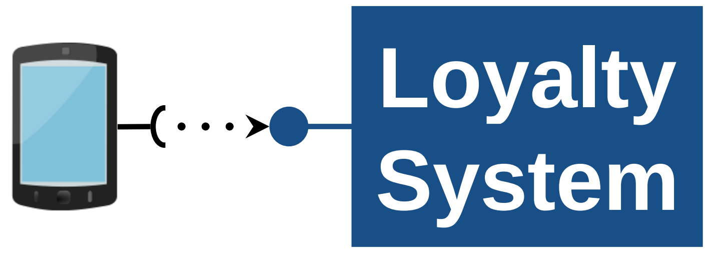
LogTransaction:

register(shop: ID, card:ID, product: ID, value: Float)



LogTransaction:

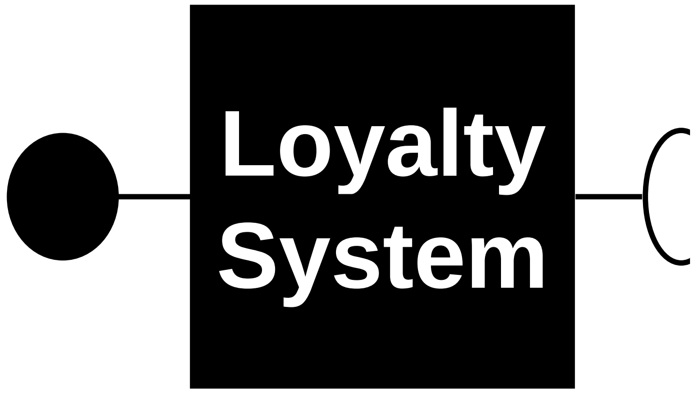
register(shop: ID, card:ID, value: Float)

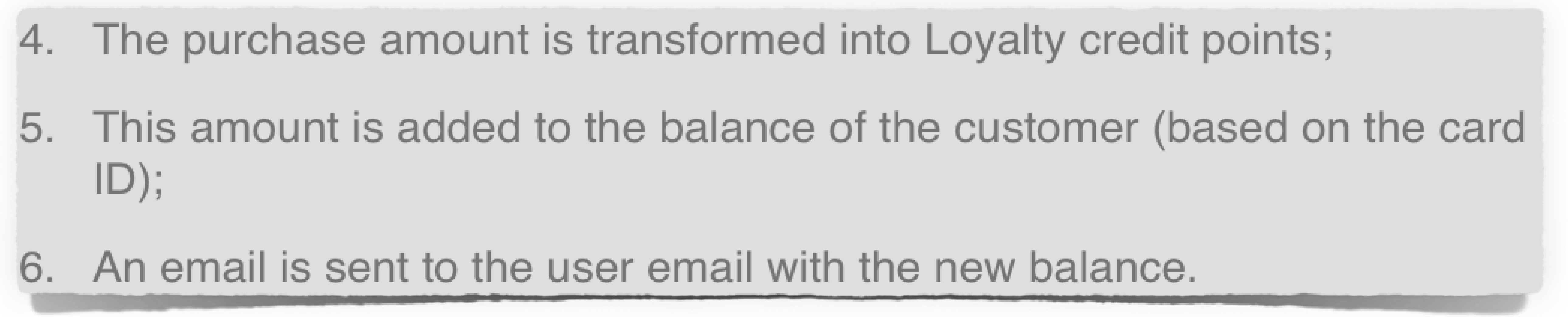


LogTransaction:

register(transaction: Transaction)

Componentizing the system





4. The purchase amount is transformed into Loyalty credit points;
5. This amount is added to the balance of the customer (based on the card ID);
6. An email is sent to the user email with the new balance.

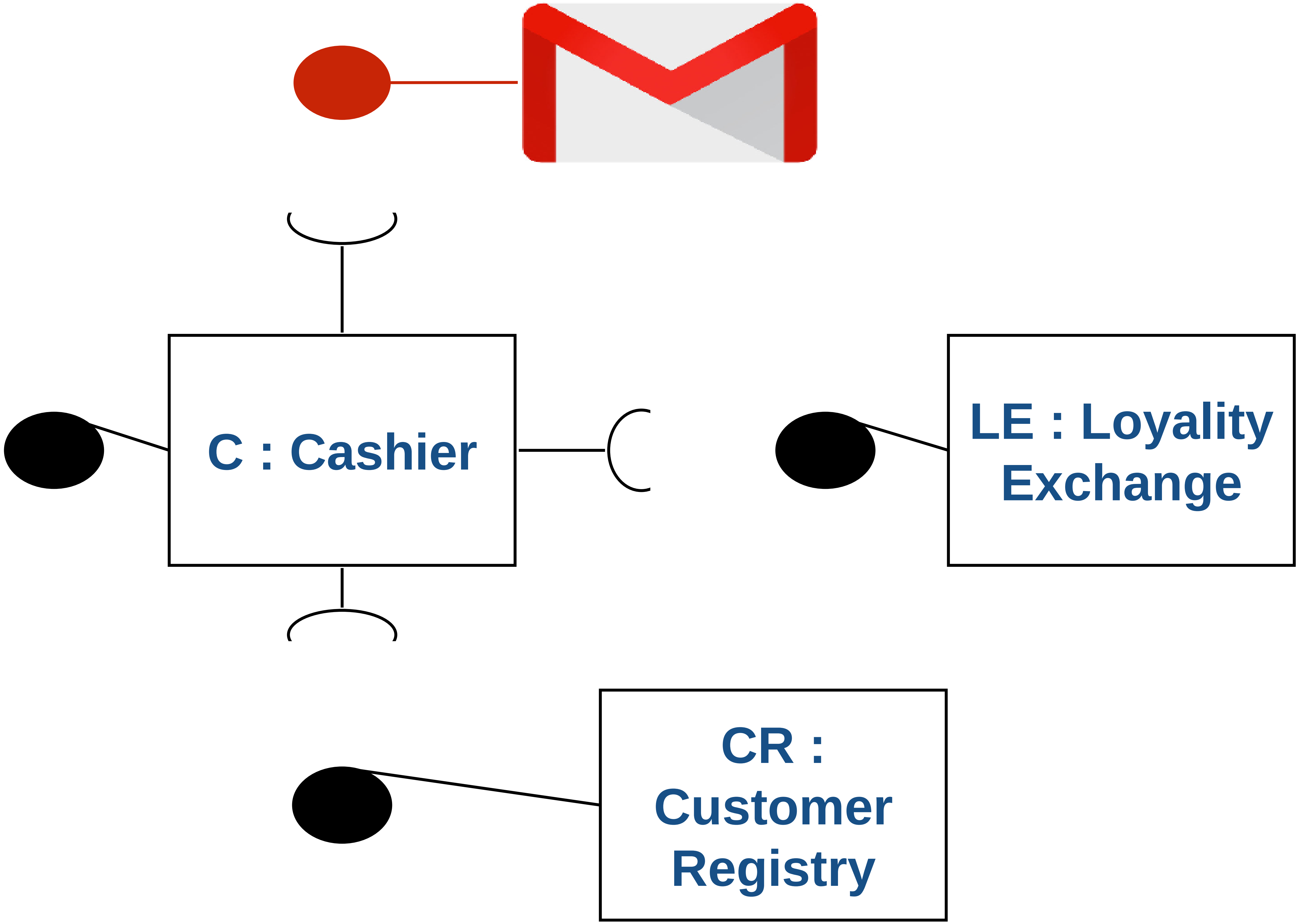
Registry:
update(card: ID, points: int)

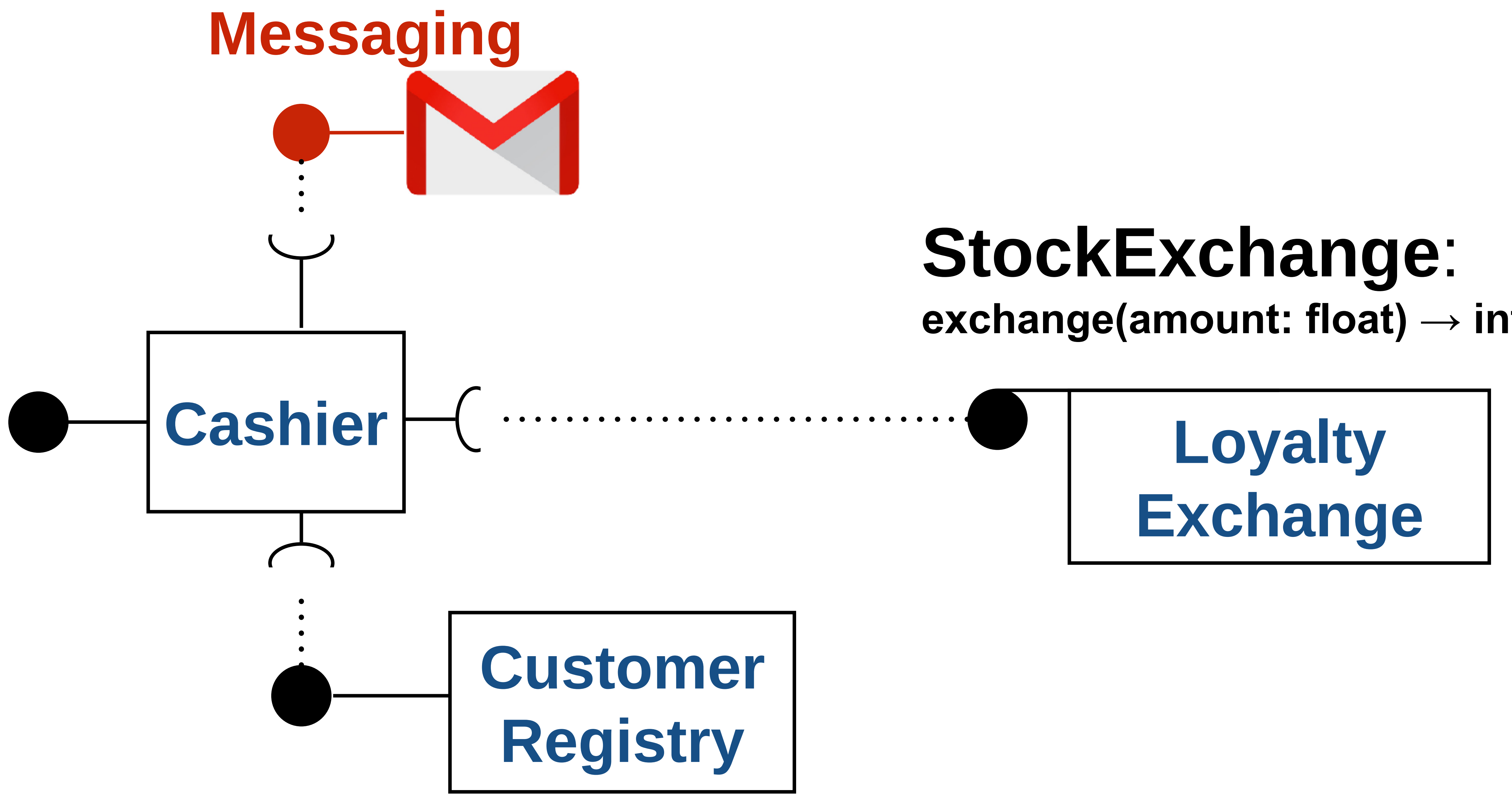
StockExchange:
exchange(amount: float) → int

Finder:
get(card: ID) → Customer

StockExchange:
exchange(amount: float, card: ID)

#1

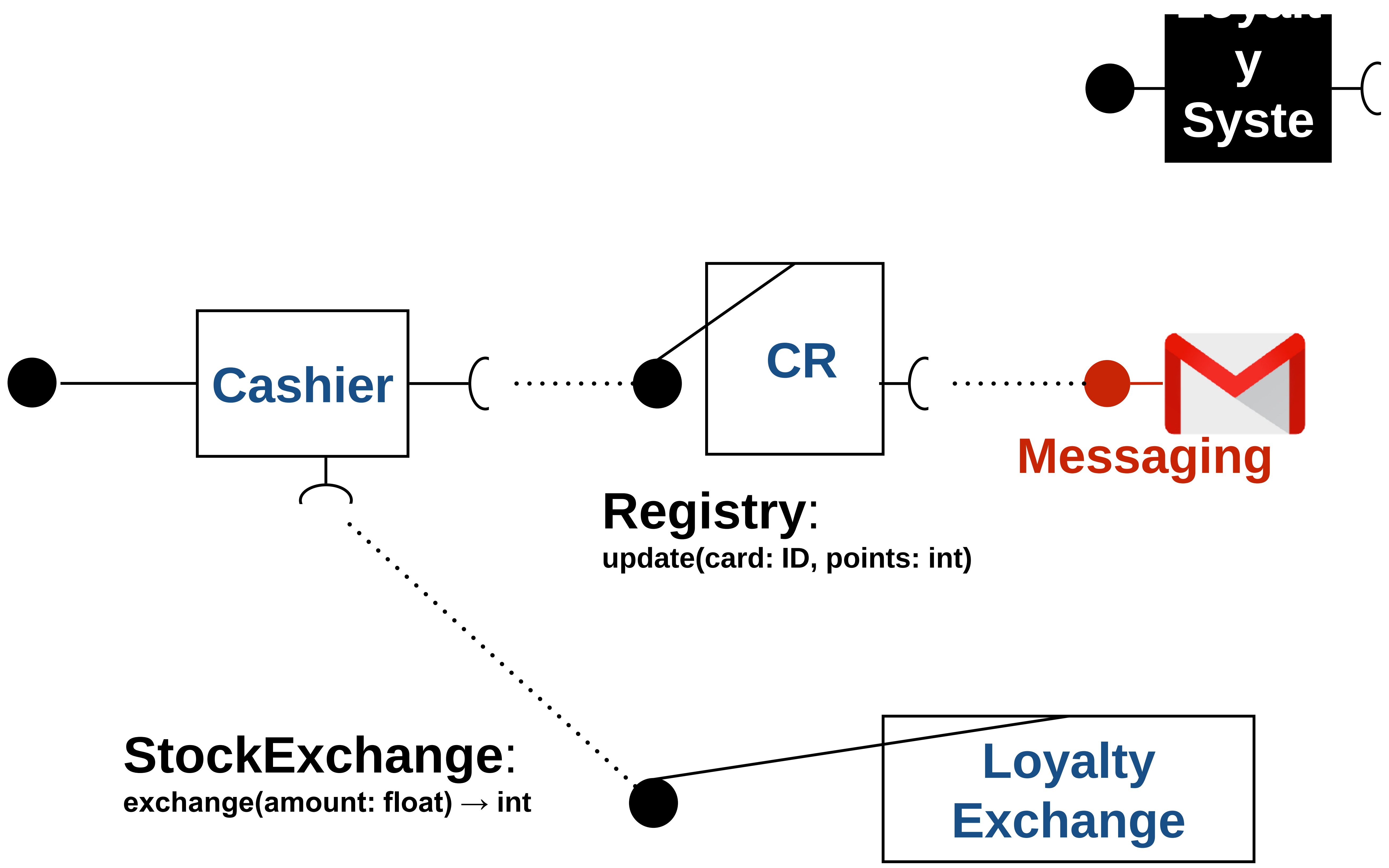

C : Cashier
LE : Loyality Exchange
CR : Customer Registry



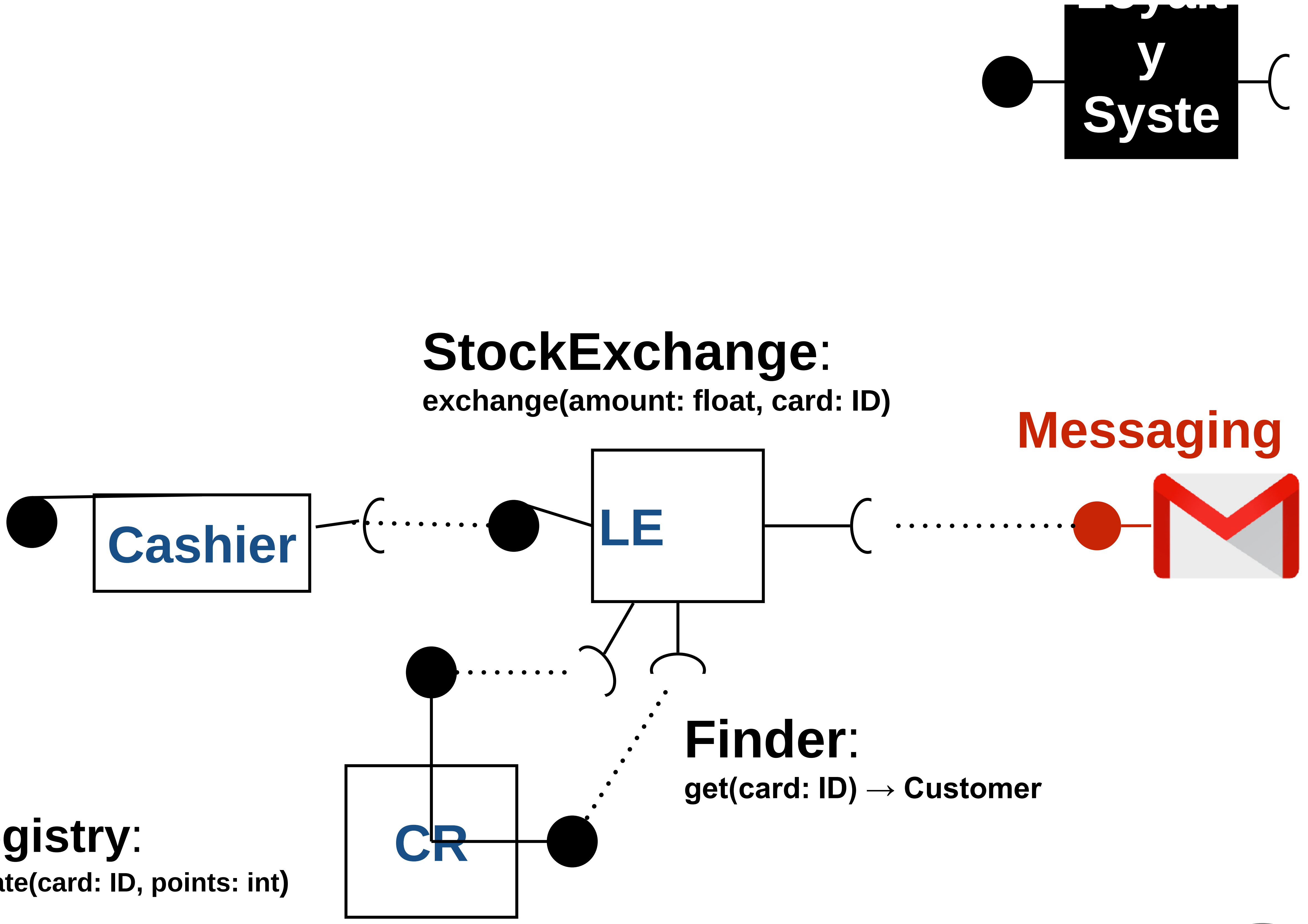
StockExchange:
exchange(amount: float) → int

Registry:
update(card: ID, points: int) → EmailAddress

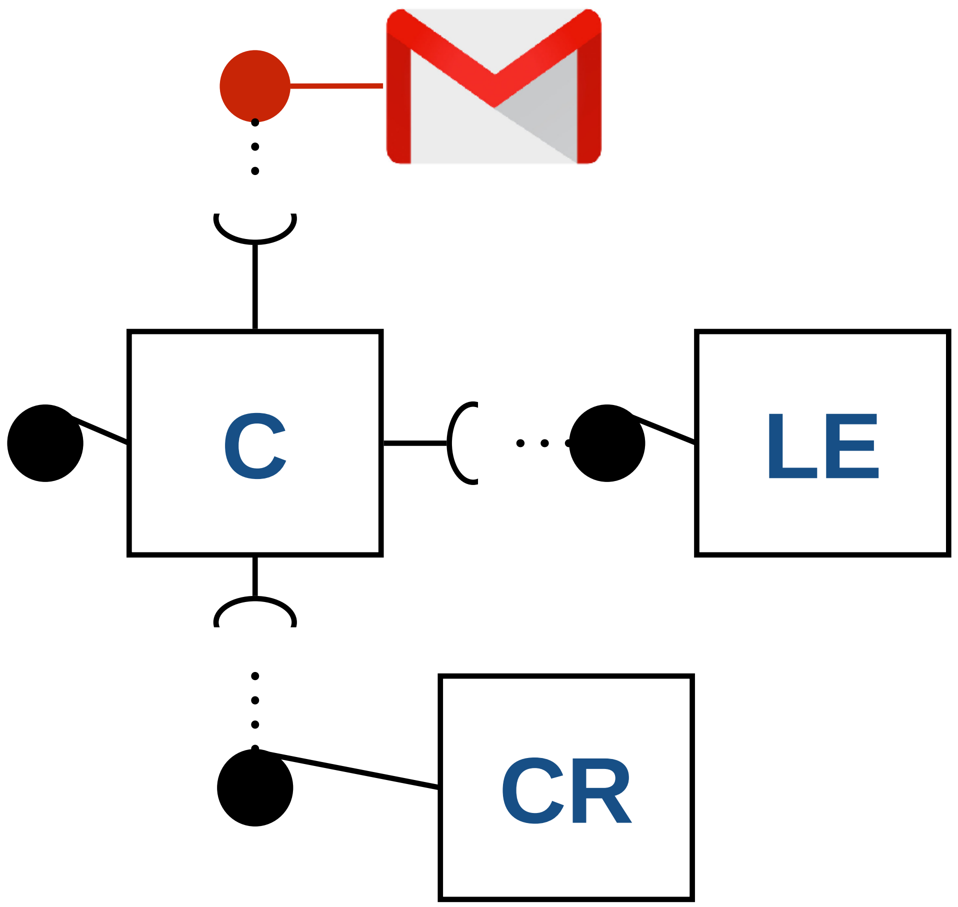
#1

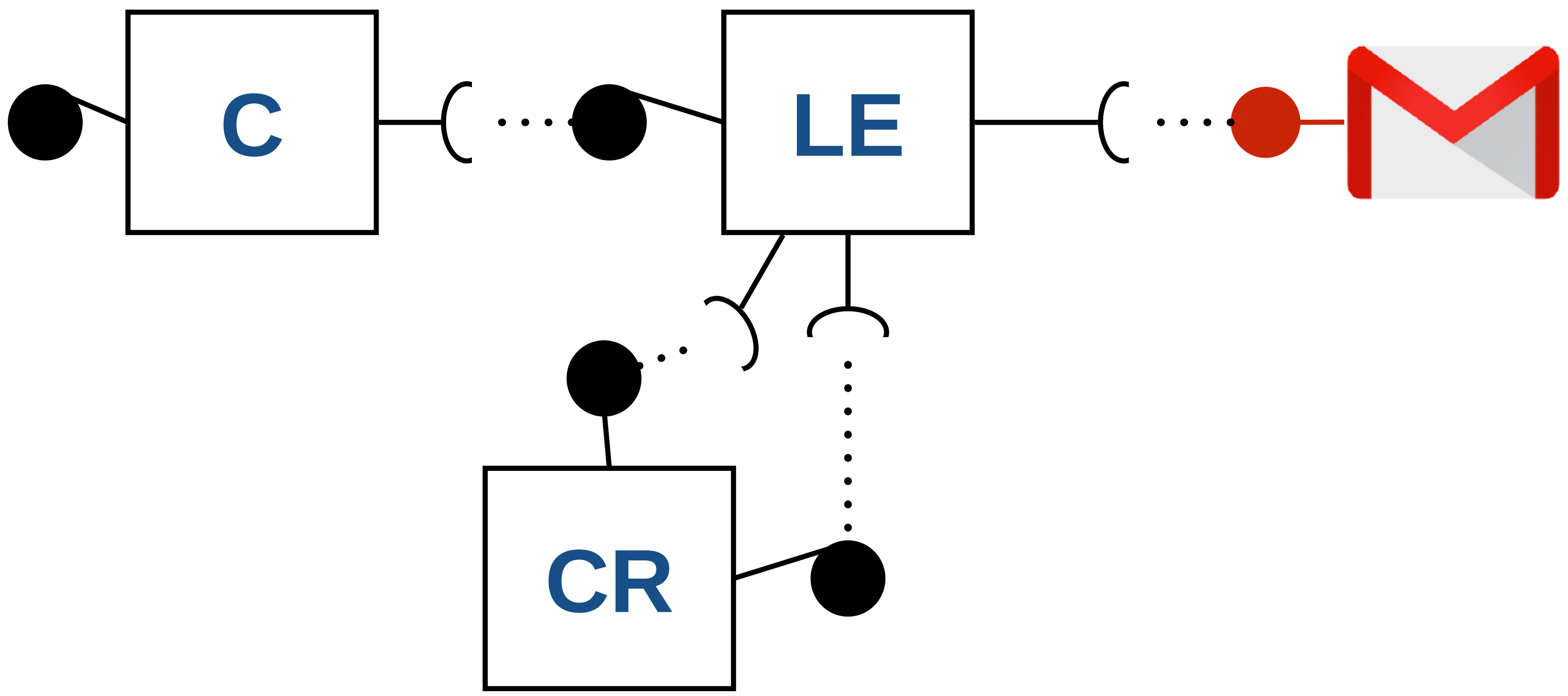

Loyalt
y
Syste
Cashier
CR
Messaging
Registry:
update(card: ID, points: int)
StockExchange:
exchange(amount: float) → int
Loyalty
Exchange

#2

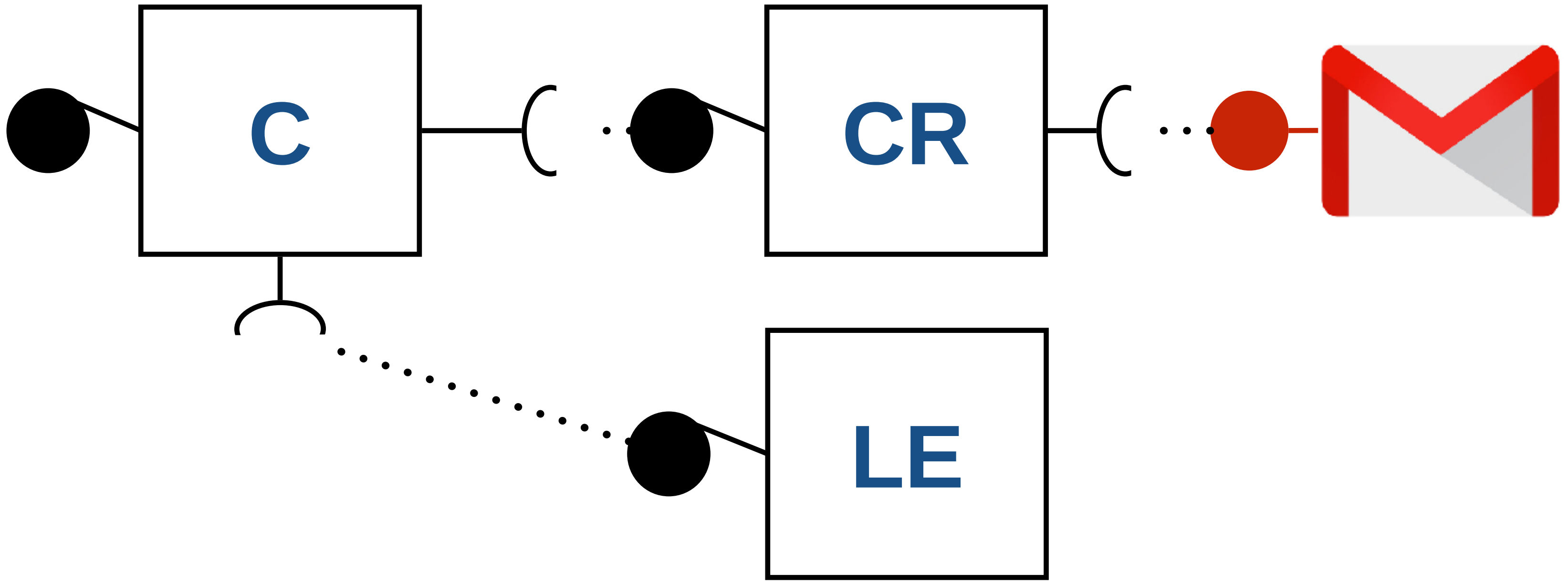

y
Syste
StockExchange:
exchange(amount: float, card: ID)
Messaging
Cashier
LE
Finder:
get(card: ID) → Customer
Registry:
update(card: ID, points: int)
CR

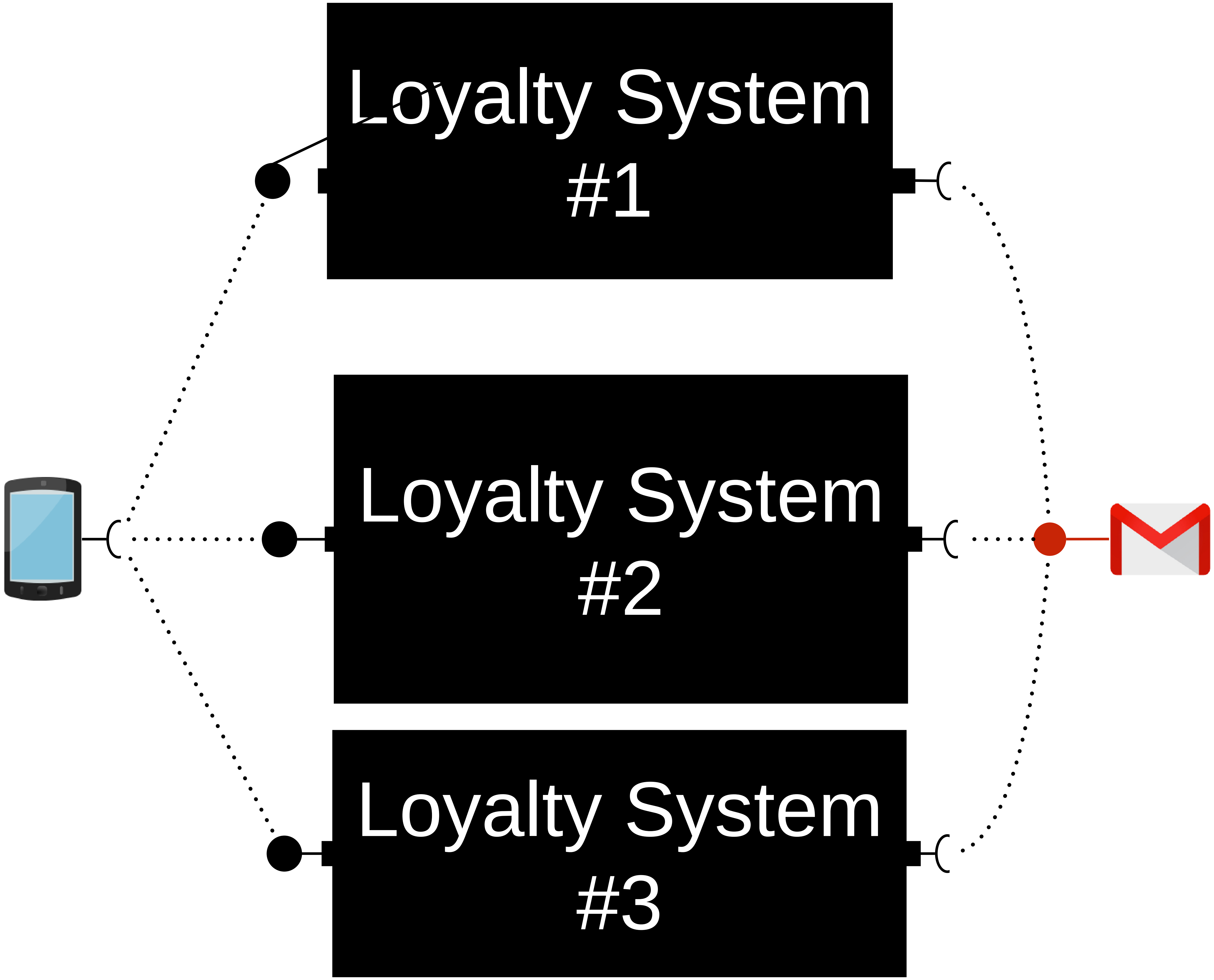
#3


C
LE
CR


C
LE
CR

Deal With It


C
CR
LE


Loyalty System #1
Loyalty System #2
Loyalty System #3

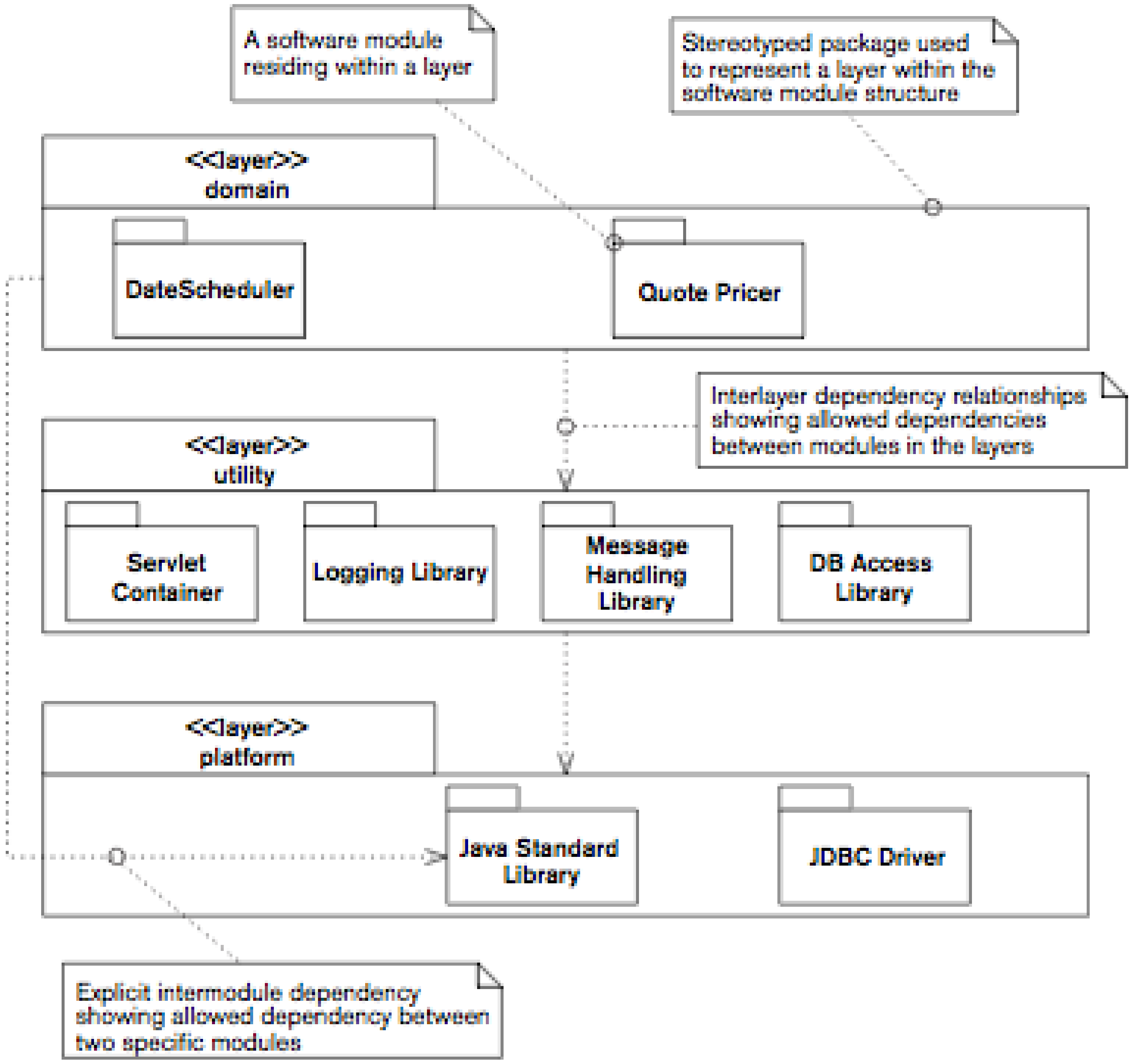
2

Development
Viewpoint

Definition

Describes the **architecture** that supports the **software development process**

SSA]


A software module residing within a layer
Stereotyped package used to represent a layer within the software module structure
<<layer>> domain
DateScheduler
Quote Pricer
Interlayer dependency relationships showing allowed dependencies between modules in the layers
<<layer>> utility
Servlet Container
Logging Library
Message Handling Library
DB Access Library
<<layer>> platform
Java Standard Library
JDBC Driver
Explicit intermodule dependency showing allowed dependency between two specific modules

SSA]

Elicitation process

$$\begin{tikzcd} \text{Requirements} \arrow[d] \\ \text{Modules} \end{tikzcd}$$

1. Identify and classify the modules
2. Identify the dependencies
3. Identify the layering rules

Classical Pitfalls

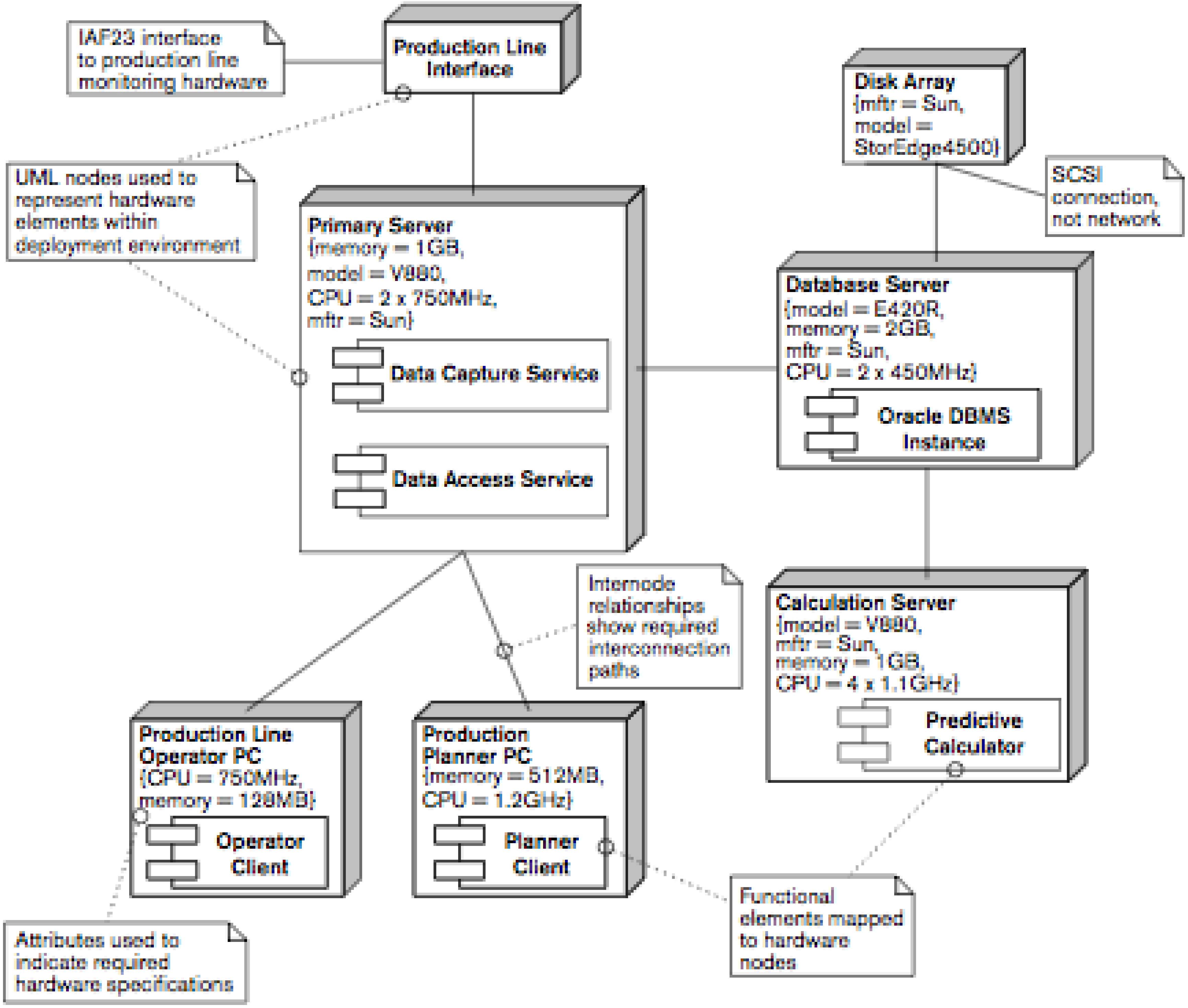
- Too much details
- Overburdened architectural description
- Uneven focus
- Lack of developer focus
- Lack of precision
- Problem with the environment

3

Deployment viewpoint

Definition

Describes the **environment** into which the **system will be deployed** and the **dependencies** that the system has on element of it


IAF23 interface to production line monitoring hardware
Production Line Interface
Disk Array {mftr = Sun, model = StorEdge4500}
SCSI connection, not network
UML nodes used to represent hardware elements within deployment environment
Primary Server {memory = 1GB, model = V880, CPU = 2 x 750MHz, mftr = Sun}
Data Capture Service
Data Access Service
Database Server {model = E420R, memory = 2GB, mftr = Sun, CPU = 2 x 450MHz}
Oracle DBMS Instance
Internode relationships show required interconnection paths
Calculation Server {model = V880, mftr = Sun, memory = 1GB, CPU = 4 x 1.1GHz}
Predictive Calculator
Production Line Operator PC {CPU = 750MHz, memory = 128MB}
Operator Client
Production Planner PC {memory = 512MB, CPU = 1.2GHz}
Planner Client
Functional elements mapped to hardware nodes
Attributes used to indicate required hardware specifications

SSA]

Elicitation process

1. Design the deployment environment
2. Map the element to the hardware
3. Estimate the hardware requirements
4. Conduct a technical evaluation
5. Assess the constraints

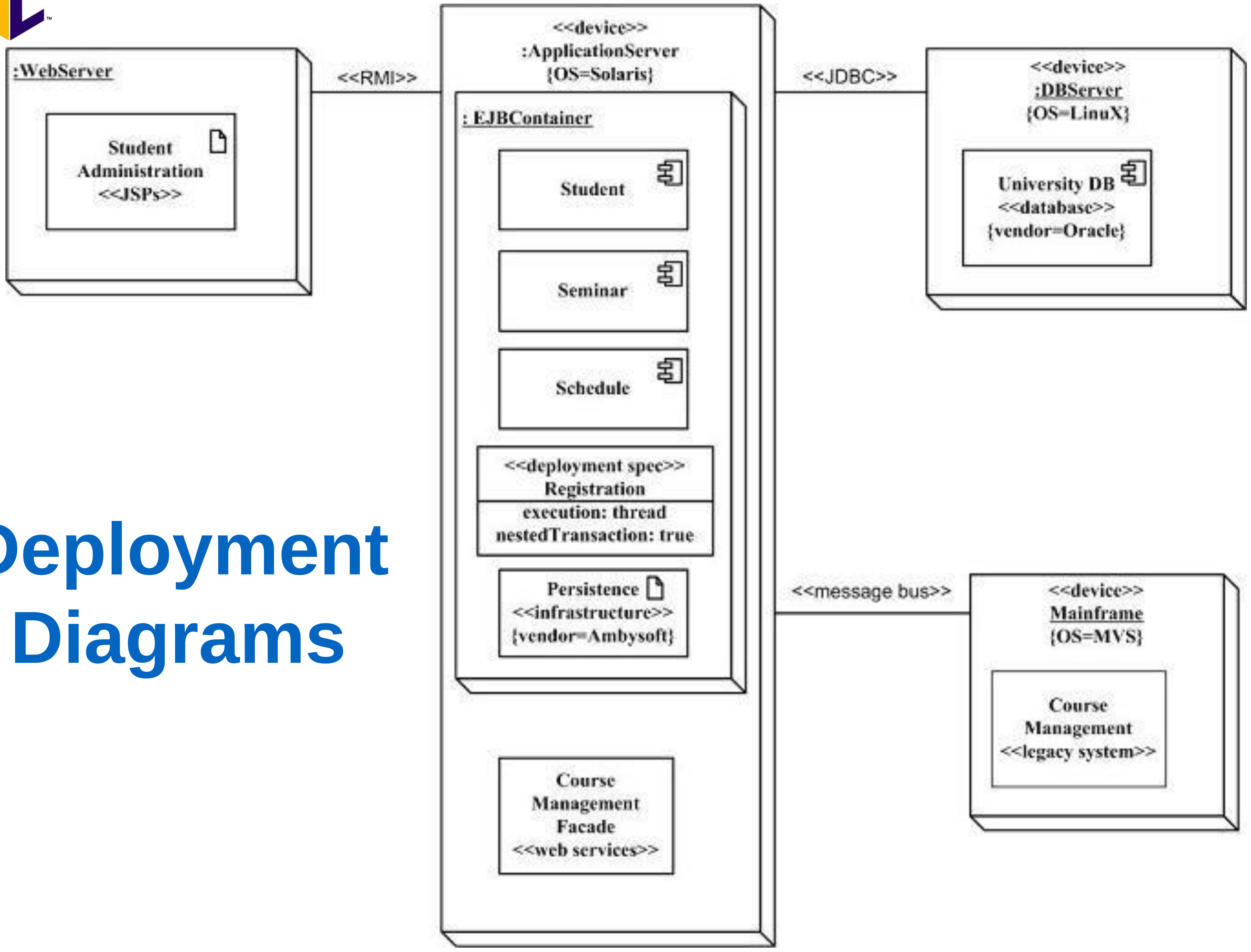
$$\begin{tikzcd}\text{Requirements} \arrow[d] \\ \text{Deployment}\end{tikzcd}$$

SSA]

Classical Pitfalls

- Unclear / Inaccurate dependencies
- Unproven technology
- Unsuitable Service-level agreement
- Lack of technical knowledge
- Late consideration of the environment
- Not specifying a disaster recovery environment

Deployment Diagrams


:WebServer
Student Administration <<JSPs>>
<<RMI>>
<<device>> :ApplicationServer {OS=Solaris}
: EJBContainer
Student
Seminar
Schedule
<<deployment spec>> Registration
execution: thread
nestedTransaction: true
Persistence <<infrastructure>> {vendor=Ambysoft}
Course Management Facade <<web services>>
<<JDBC>>
<<device>> :DBServer {OS=LinuX}
University DB <<database>> {vendor=Oracle}
<<message bus>>
<<device>> Mainframe {OS=MVS}
Course Management <<legacy system>>

ww.agilemodeling.com/artifacts/deploymentDiagram.htm