How to

break down a domain to bounded contexts?

About me

- Oliver Tigges, @otigges, oliver.tigges@innoq.com
- innoQ, http://innoQ.com
- Software develop & IT consultant since 2001
- many domains and businesses:
  banking, payment, insurance, e-commerce, industrial, media, railway, environmental agencies, medical, de-mail, IoT, internet start ups, etc.

About this talk

- Goal: Find adequate and sustainable Bounded Contexts in your domain
- What are the most important influence factors?
- What are suitable approaches and methods?
- Context: Distributed applications, Microservices architectures

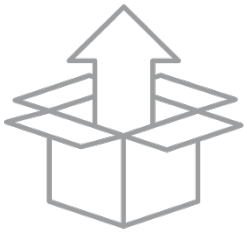
- Before we talk about „how“…
- Let‘s talk about:
  - Why?
  - Who?
  - When?

› Goal: Independence of systems and teams

1

Design &
Implementation

2



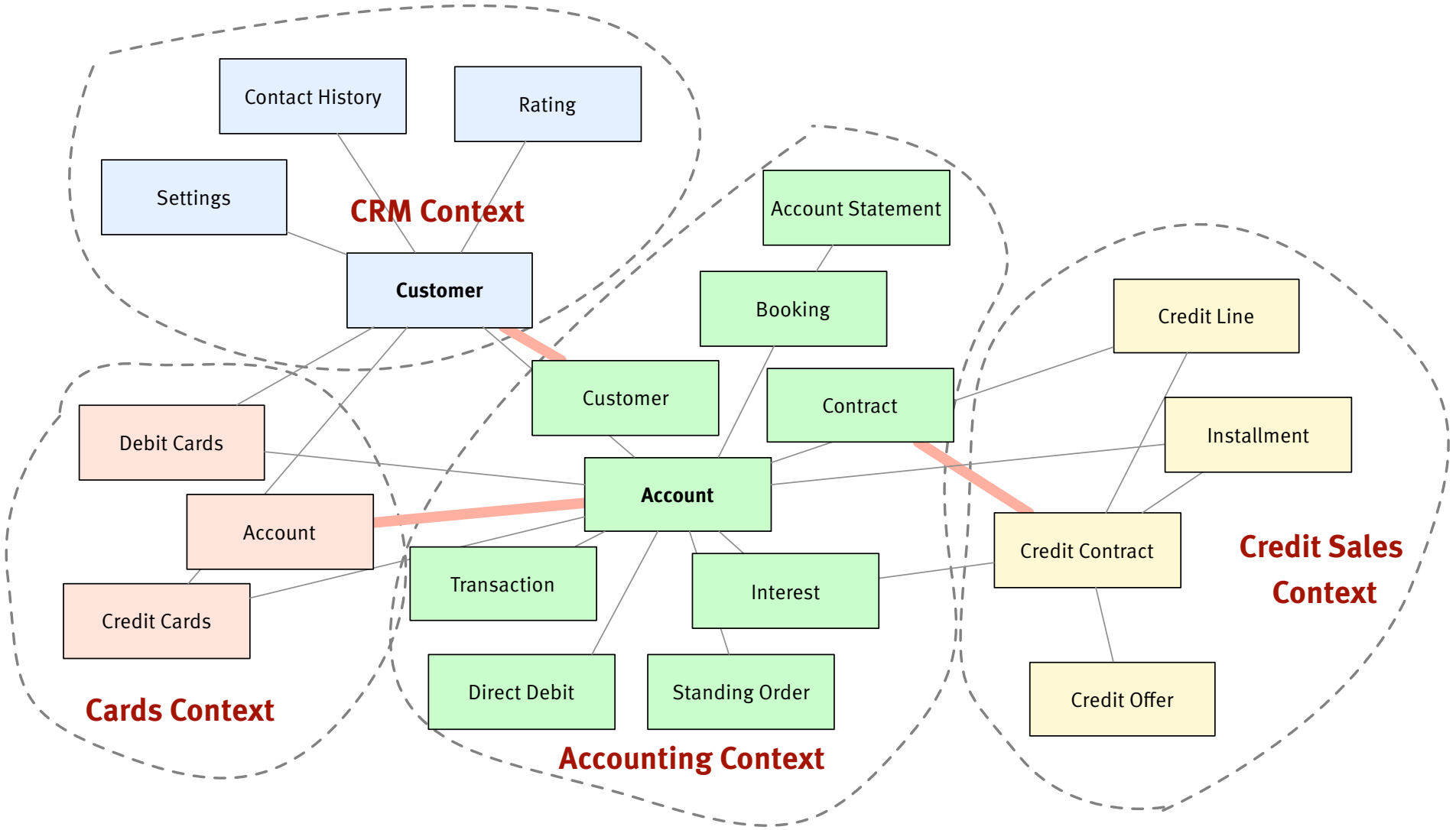
Releasing &
Deployment

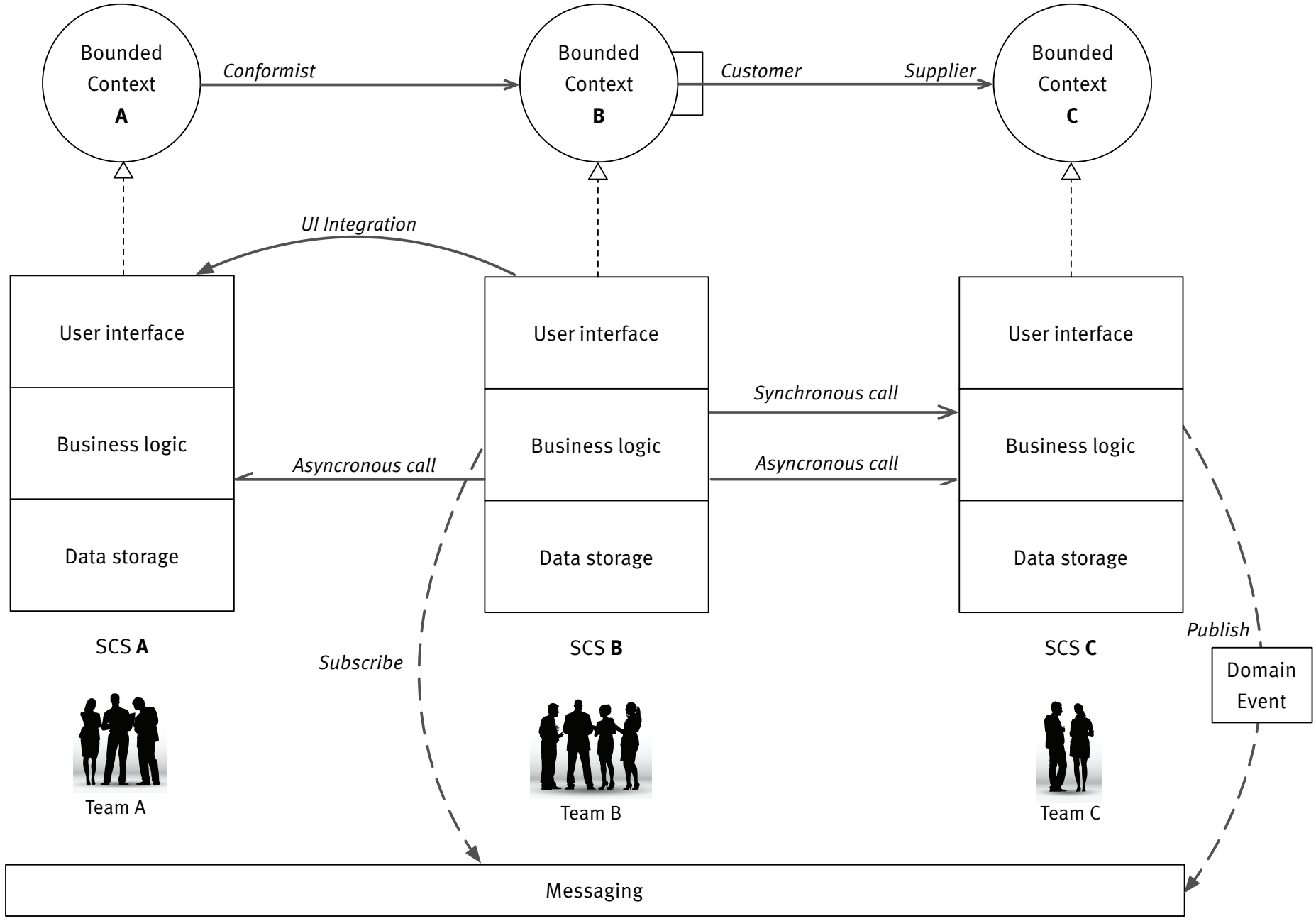
3

Runtime &
Operations

How to achieve this?

- Bounded contexts
- (Self contained) Systems matching these bounded contexts


Contact History
Rating
Settings
CRM Context
Customer
Account Statement
Booking
Credit Line
Customer
Contract
Installment
Debit Cards
Account
Account
Credit Contract
Credit Sales Context
Transaction
Interest
Credit Cards
Direct Debit
Standing Order
Credit Offer
Cards Context
Accounting Context


Bounded Context A
Conformist
Bounded Context B
Customer
Supplier
Bounded Context C
UI Integration
User interface
Business logic
Data storage
User interface
Business logic
Data storage
Synchronous call
Asyncronous call
Asyncronous call
User interface
Business logic
Data storage
Publish
Domain Event
SCS A
SCS B
SCS C
Subscribe
Team A
Team B
Team C
Messaging

Who identifies contexts?

Alberto Brandolini says:

- Domain experts
- Dev team
- UX experts
- Facilitator

Actually in most projects:

- Software developers & architects

Approach

Context boundary == System boundary

- If Bounded Context defines the technical system boundaries, it not only partitions domain model but also defines units for:
  - development (teams)
  - deployment
  - availability
  - scalability
  - security zones

What to consider?

1. Domain model: Domain objects and their relations
2. Use Cases, processes and workflows
3. Quality goals, non-functional requirements
4. Organizational aspects

Domain model

- Identify domain objects: events, aggregates, etc.
- Analyze and describe relations between domain objects
- Be aware of an object's varying charachteristics in different use cases
- Maybe try *Event Storming*, Alberto Brandolini

Process ownership

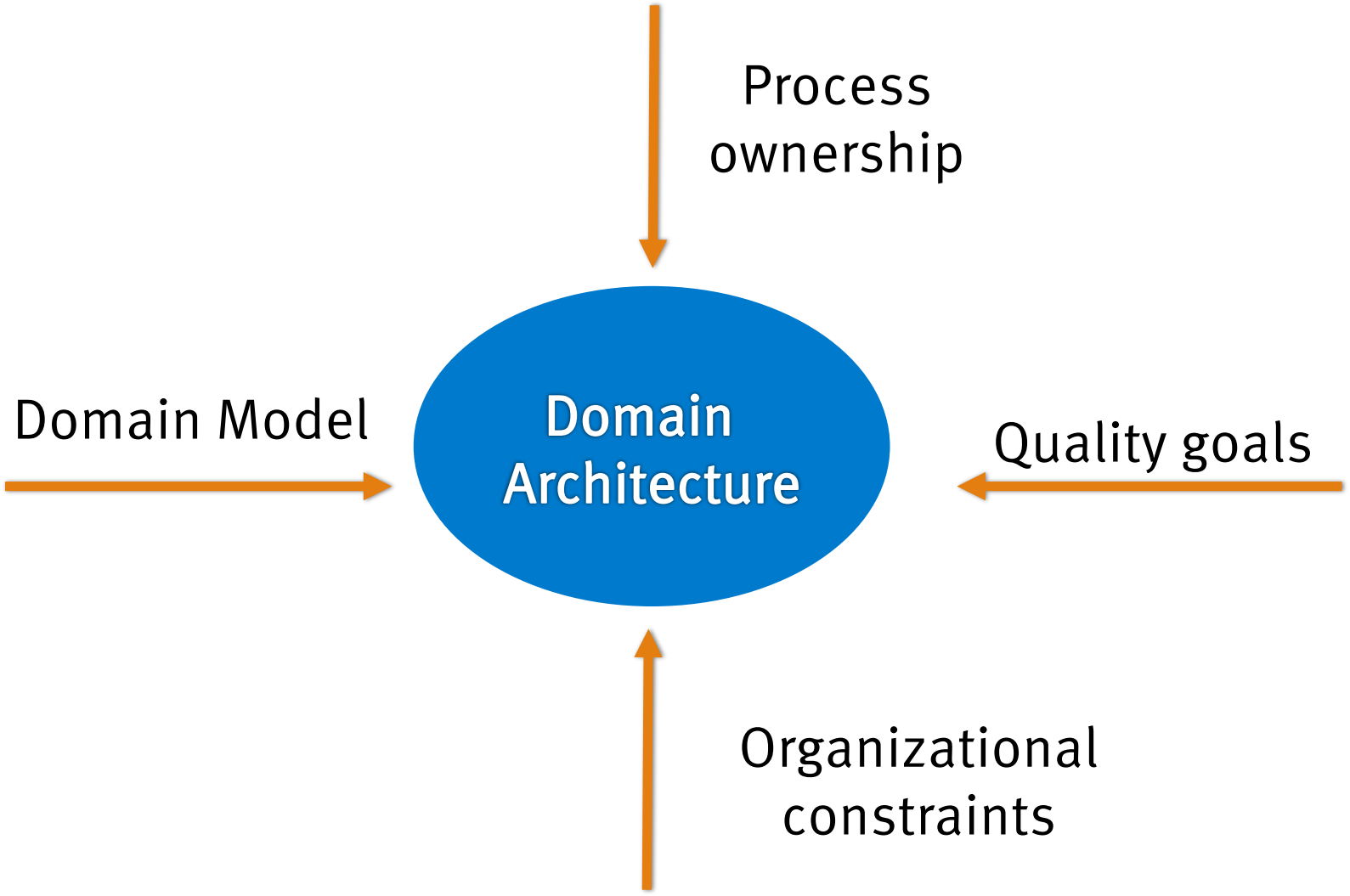
- Identify processes that need to be owned and controlled by one person in charge and one team
- Concentrate responsibility for business goals / KPIs in one hand
  - Examples: User registration, eCommerce checkout, conversion rates

Quality goals

- Derived from business goals
- Examples:
  - Time 2 Market (release/deployment cycles)
  - Security
  - Availability
  - Load and performance (read/write)
  - Scalability
  - User experience
  - ...

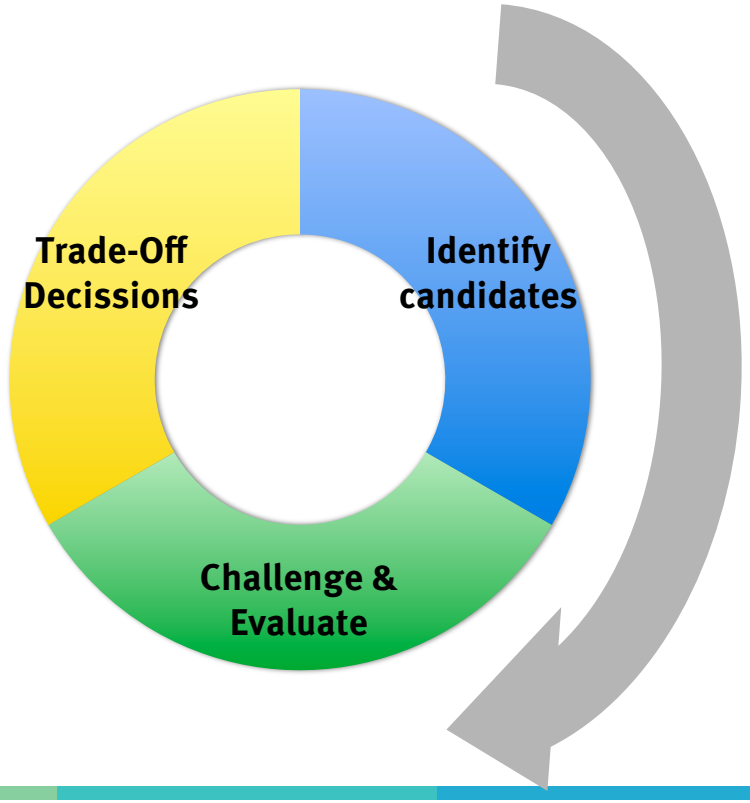
Organizational constraints

- Do you have authorization and power to adapt the organization to your system design?
- What are the constraints you can‘t change?
  - Corporate structures
  - Teams, people and skillsets
  - …


Process ownership
Domain Model
Domain Architecture
Quality goals
Organizational constraints

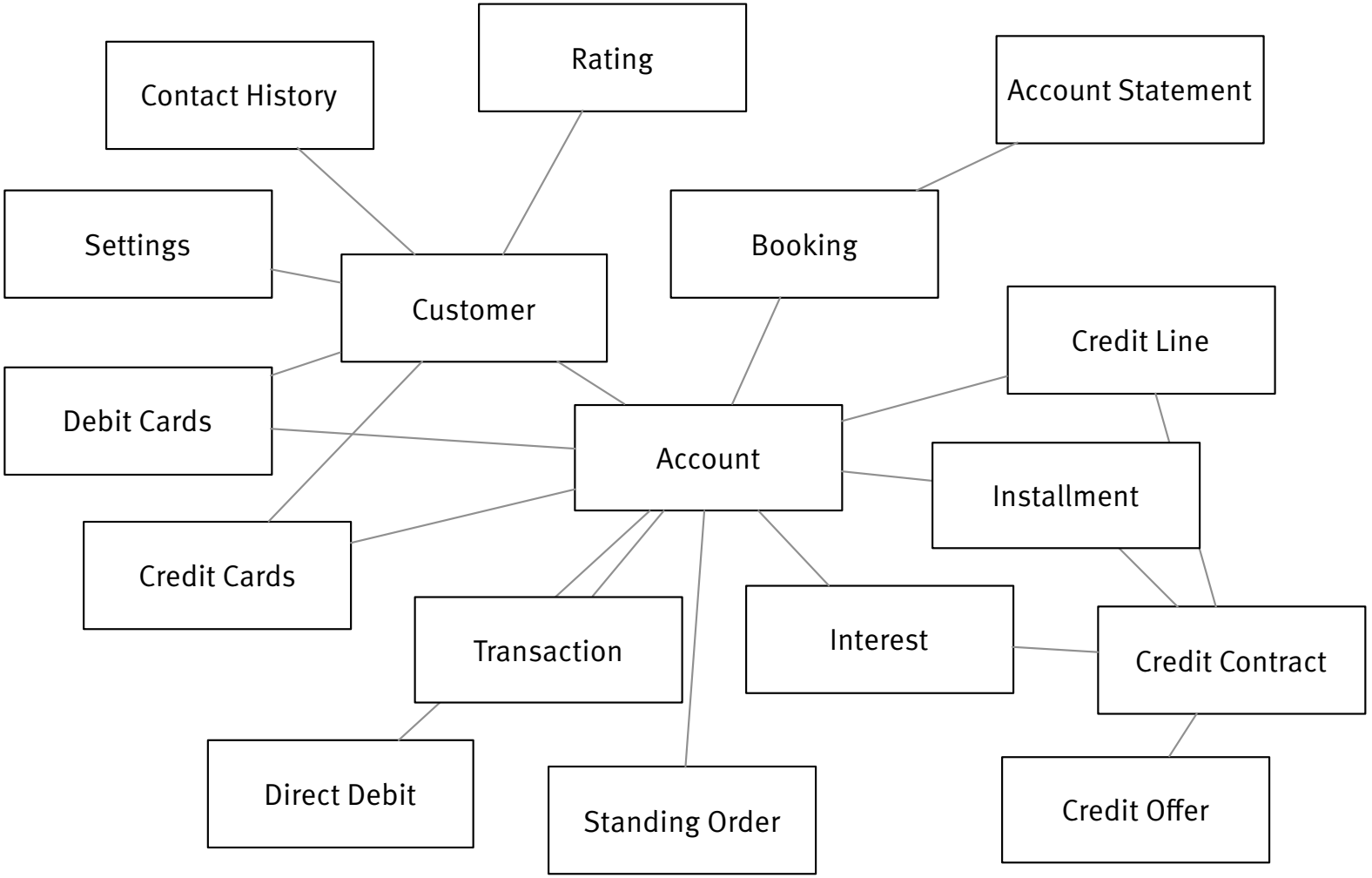
Approach

- Like every process in software architecture and development:
  - Iterative
    - Identify system candidates
    - Evaluate
    - Trade-offs
    - Repeat

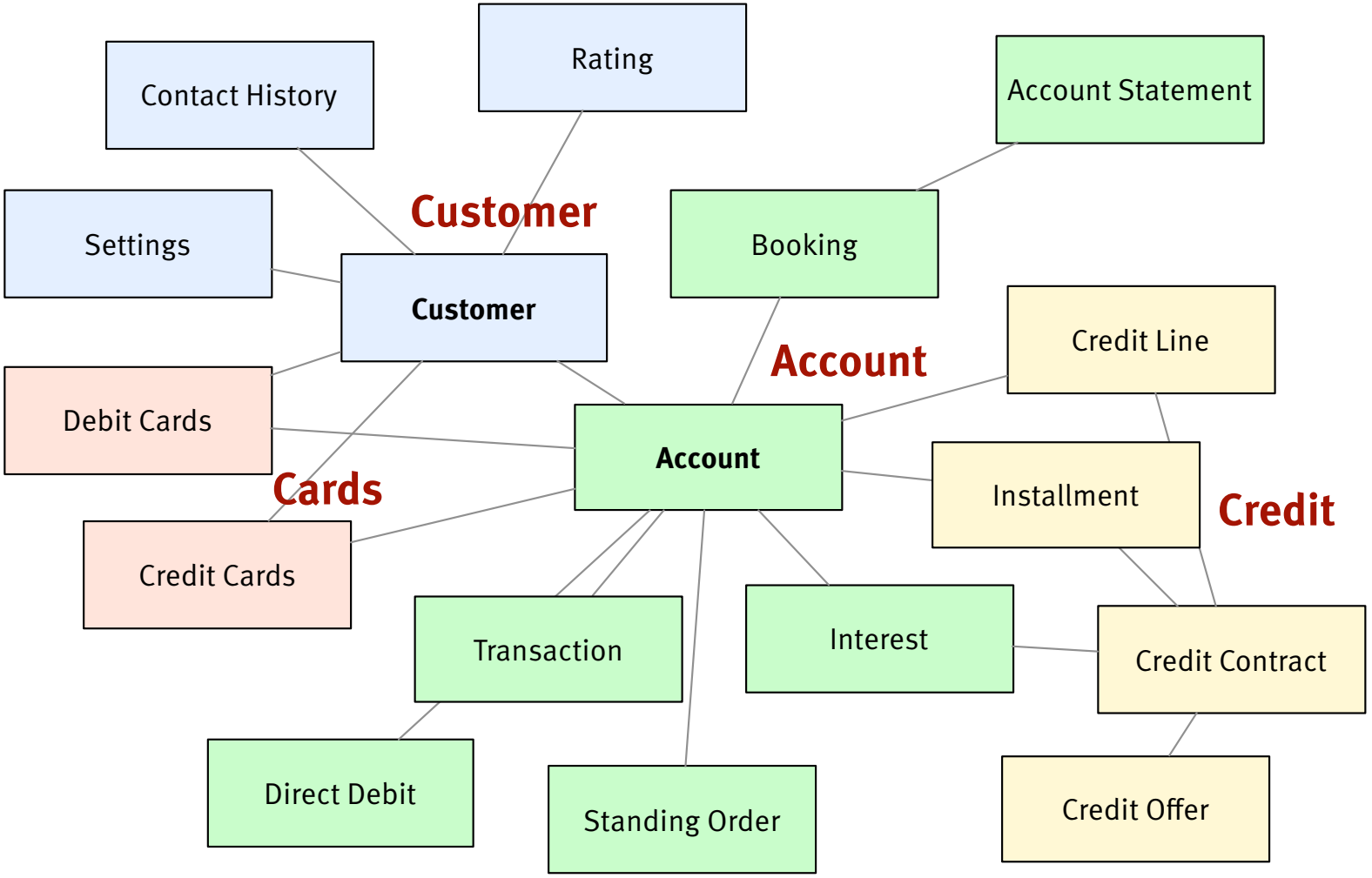


Example
Retail Banking

Find initial system candidates

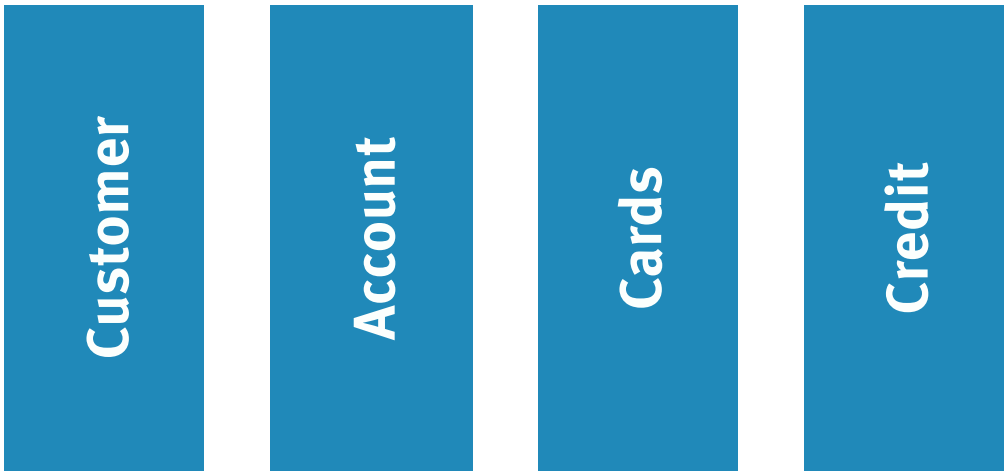


Find initial system candidates



Initial candidates

- Looking at the domain model you could identify these candidates for Bounded Contexts / systems:



Challenge system candidates

- Typical change scenarios in our example system
  - Implement additional TAN method
  - New credit product
  - New conditions for loans
  - Changes in legal or supervisory regulations
  - Reversal of design decisions
- Observe potential issues
  - Number of systems that need to be changed and released for a change
  - Coordination efforts over several teams

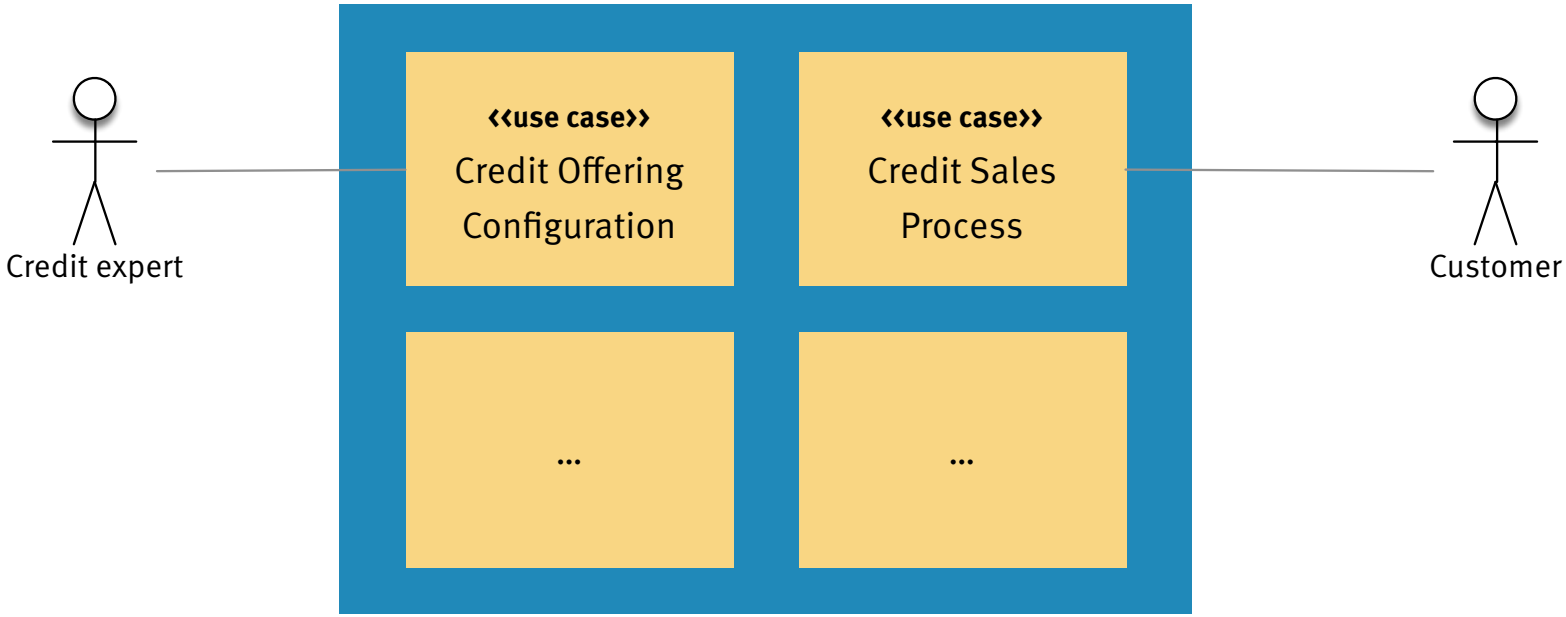
Scenario	Customer	Account	Cards	Credit
Change of credit conditions	S	S	-	L
New verification method	S	L	L	-
Change of external rating agency	L	-	-	M
New credit product	-	-	-	L
...				

Quality requirements

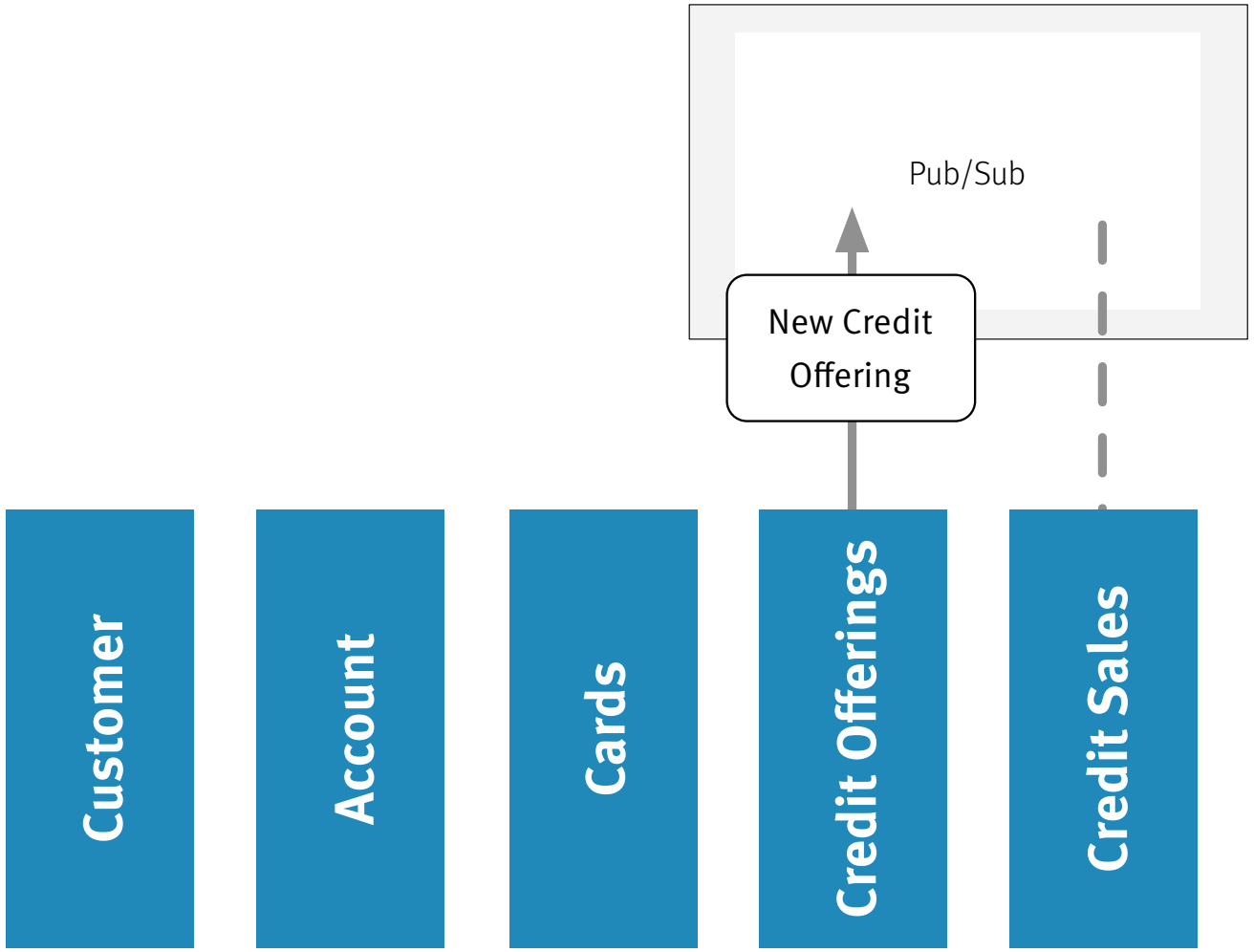
- Identify main building blocks and use cases of each system candidate
- List quality requirements of each building block
  - Security (PCI scope?) and data privacy (personal data?)
  - Time to market, expected release frequency
  - Availability, max downtime, max recovery time
  - User groups and UX requirements
  - Performance, response times, throughput, reads/writes
  - And other relevant requirements
- Quality requirements of system are the sum of their building block‘s requirements

~ 10 Employees/clerks	**Users**	› 100,000 Customers
functional, experts	**UI/UX**	customer experience
low	**Availability**	high
complex, versioned	**Data model**	simple, flat
few reads/writes	**Data access**	many reads
monthly	**Releasing**	daily

System candidate „Credit“

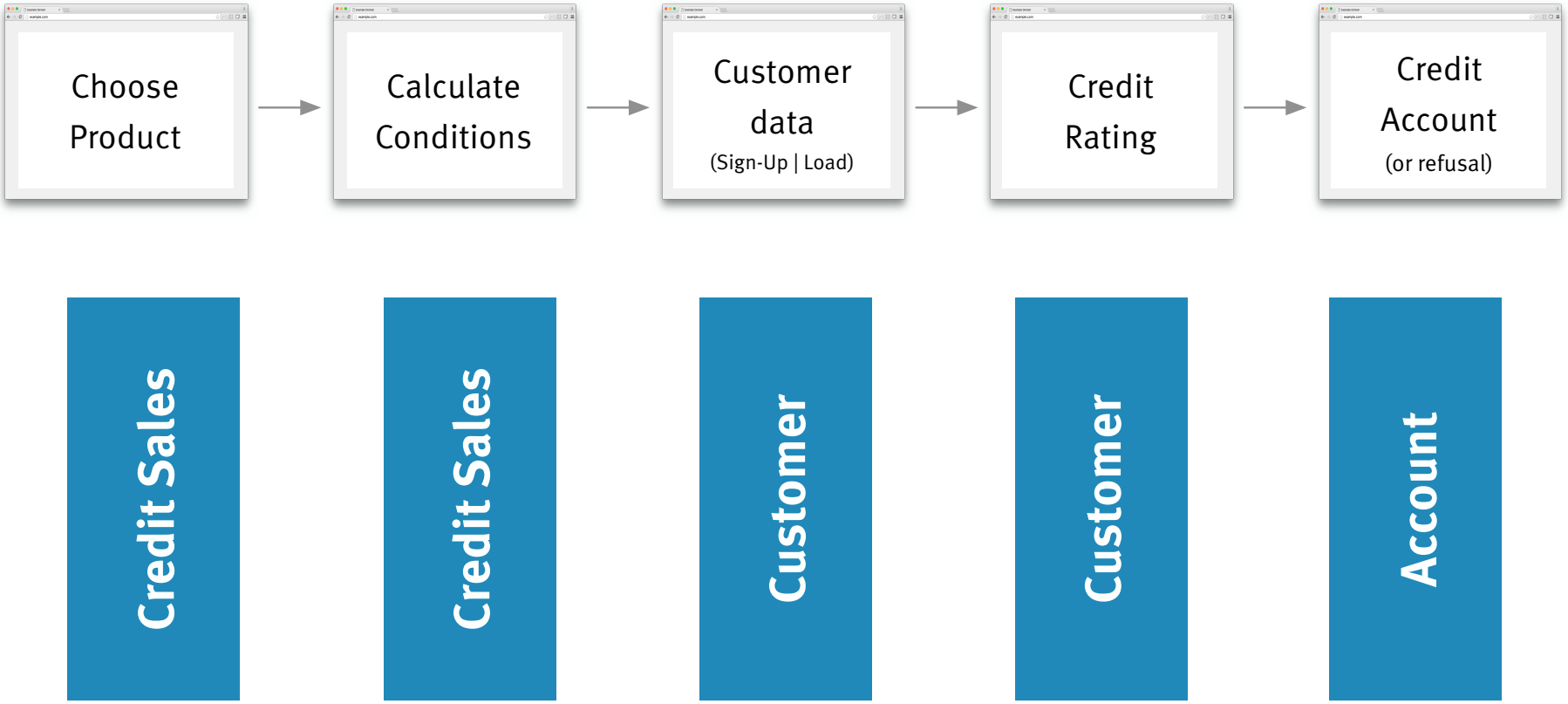


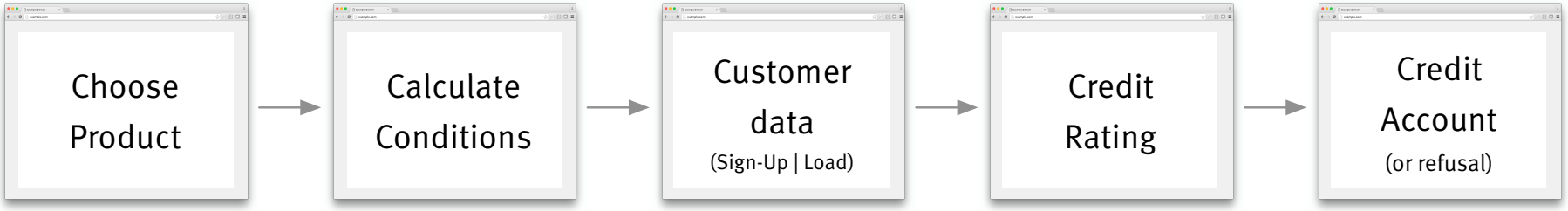
Iteration 1


Pub/Sub
New Credit Offering
Customer
Account
Cards
Credit Offerings
Credit Sales

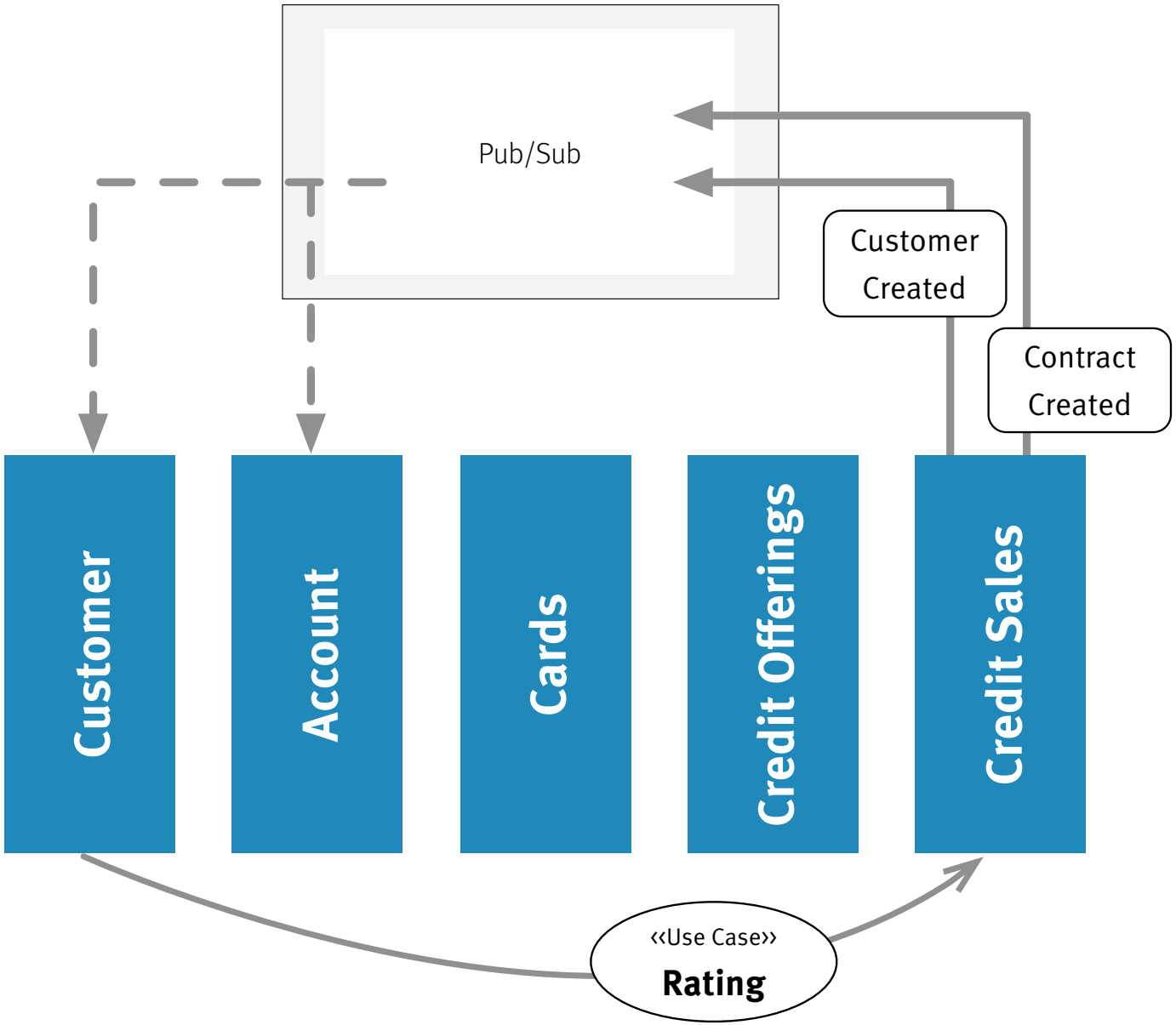
Process ownership

- Some processes will span several of identified system candidates, e.g.:
  - Sales processes for credits/loans: Customer, Account, Credit Sales
- Probably one owner should be responsible for:
  - End to end functionality
  - Consistent, smooth user experience


Choose Product
Calculate Conditions
Customer data
(Sign-Up | Load)
Credit Rating
Credit Account
(or refusal)
Credit Sales
Credit Sales
Customer
Customer
Account

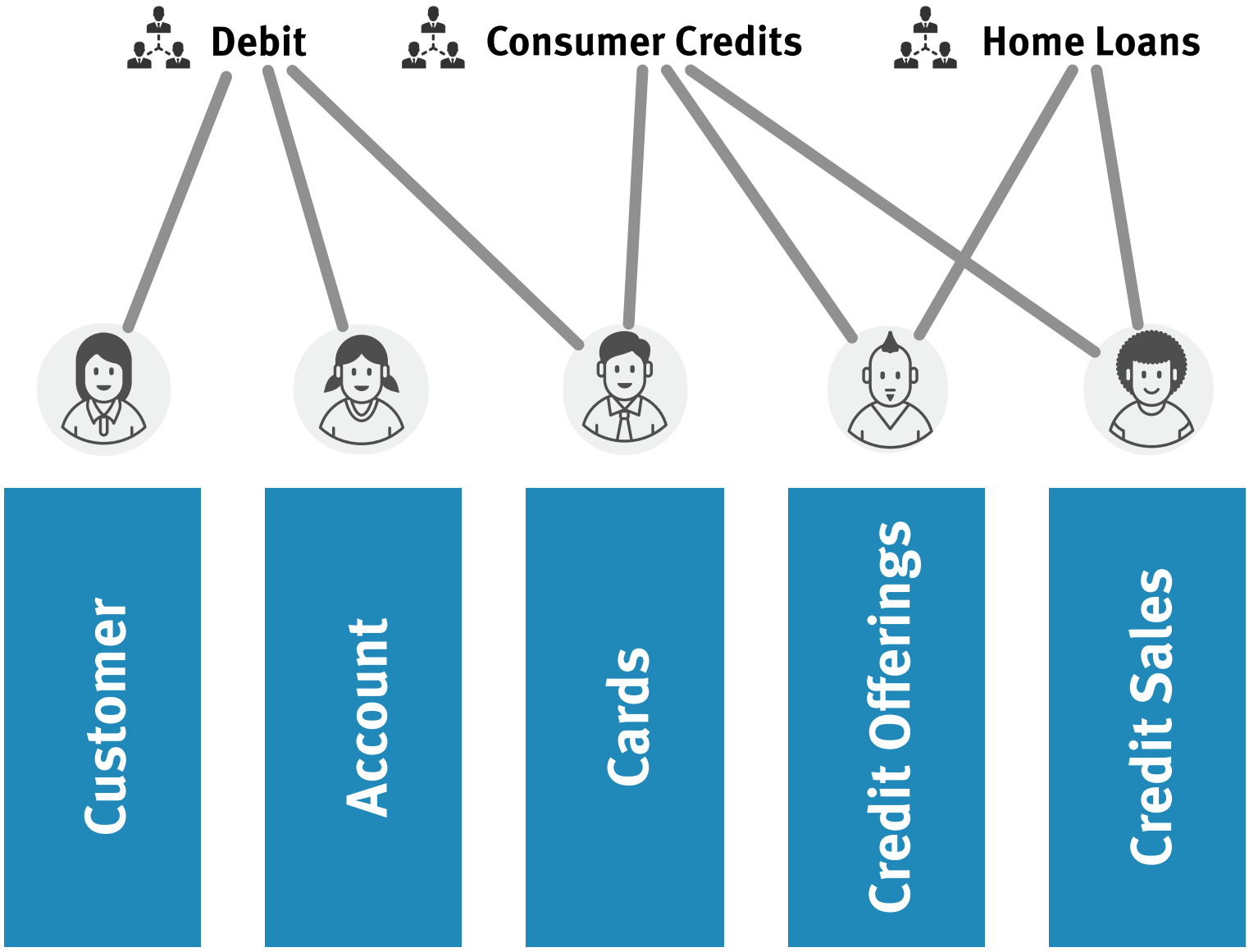


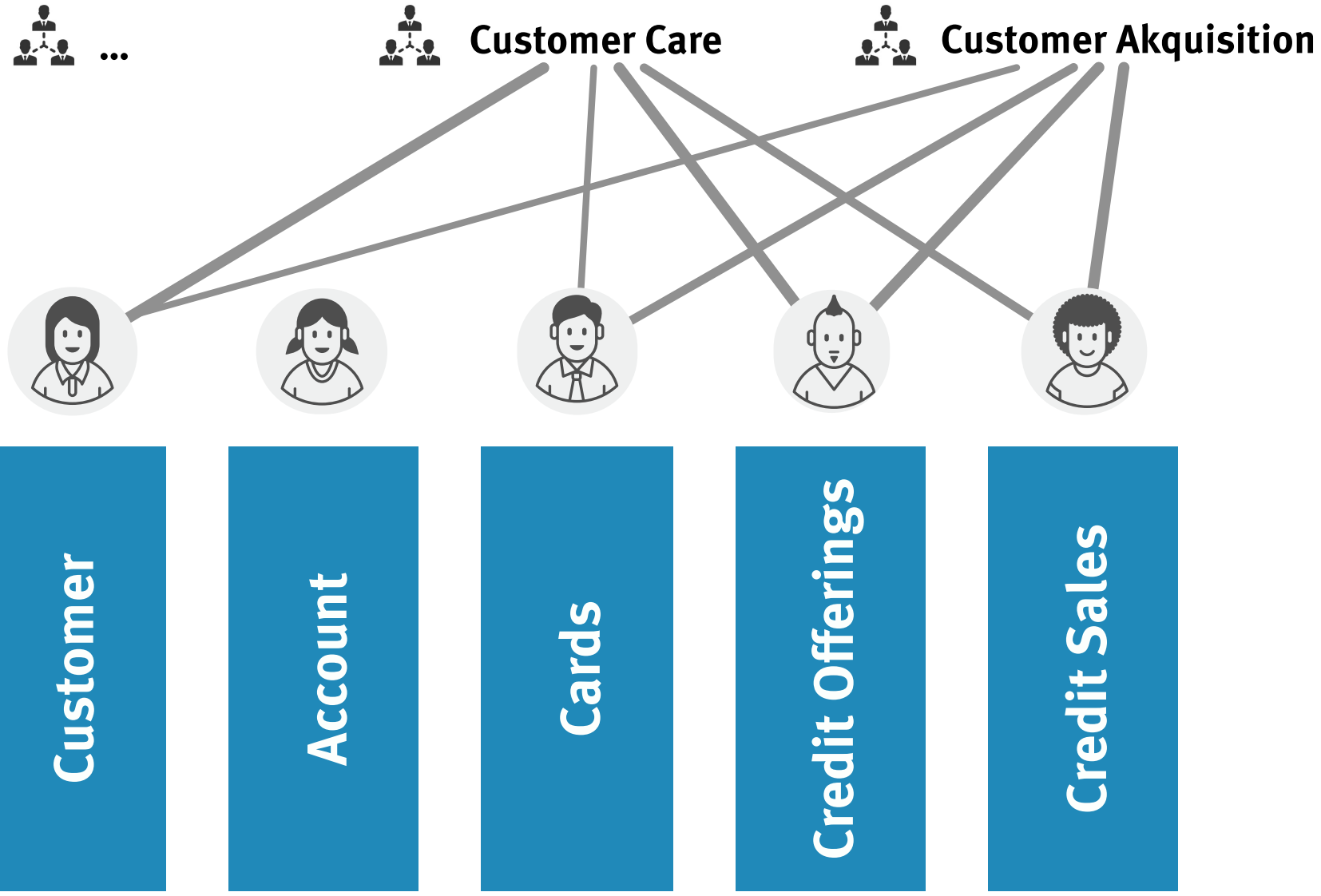
Credit Sales

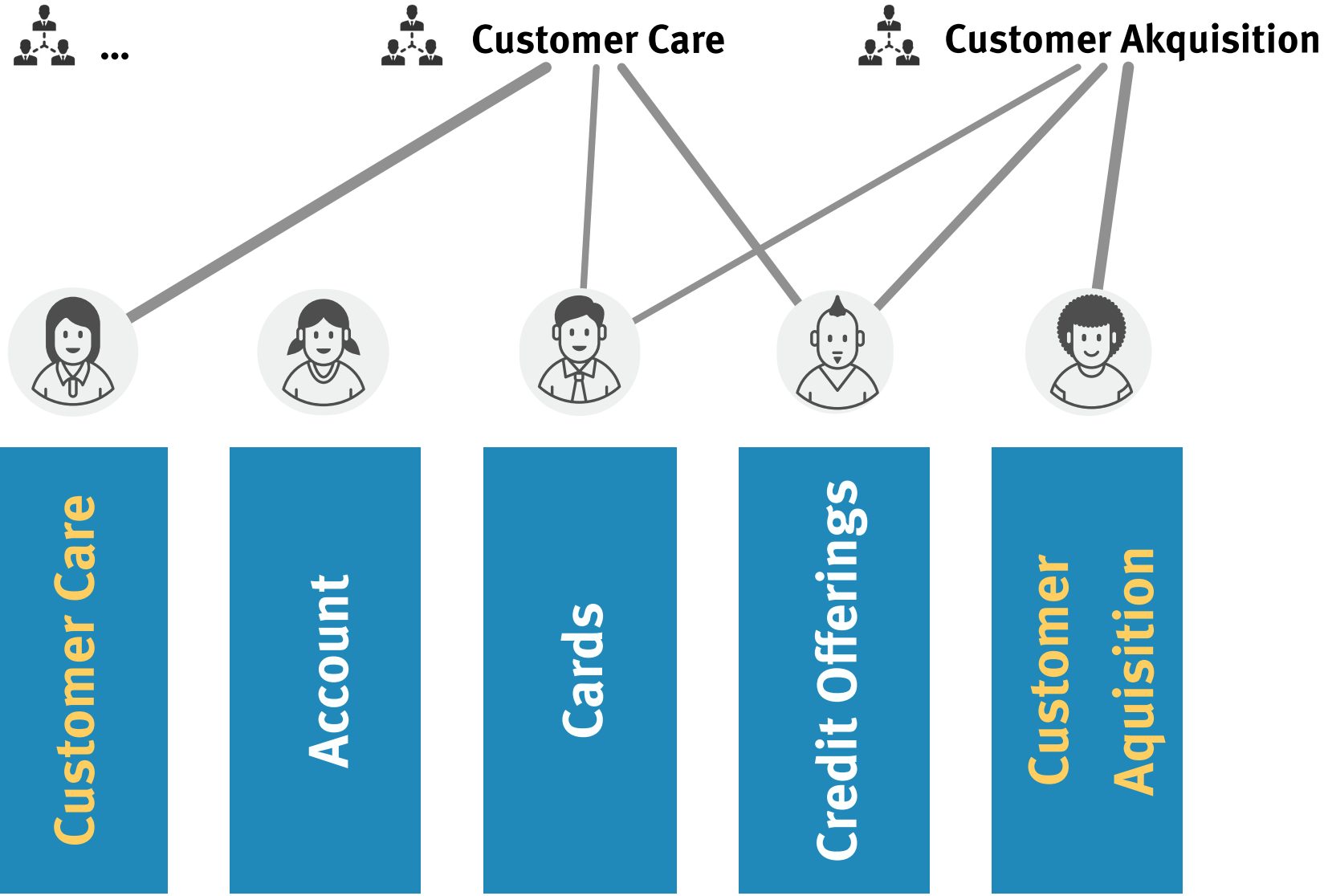

Pub/Sub
Customer Created
Contract Created
Customer
Account
Cards
Credit Offerings
Credit Sales
«Use Case»
Rating

Organizational aspects

- System design could/should reflect structures of the organization:
  - By products: debit, credit, investment, real estate
  - By sales channels: direct, stationary, brokers, agencies
  - By customer segments: existing customers, new customers, high-networth, etc.
  - By any informal structures developed by people or history

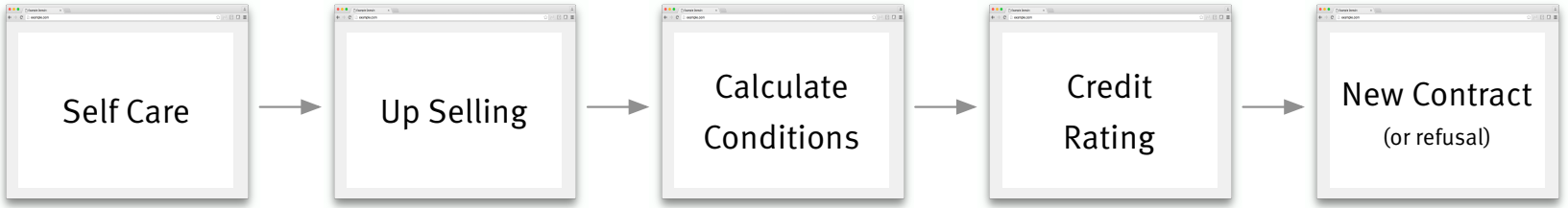

Debit
Consumer Credits
Home Loans
Customer
Account
Cards
Credit Offerings
Credit Sales


...
Customer Care
Customer Akquisition
Customer
Account
Cards
Credit Offerings
Credit Sales

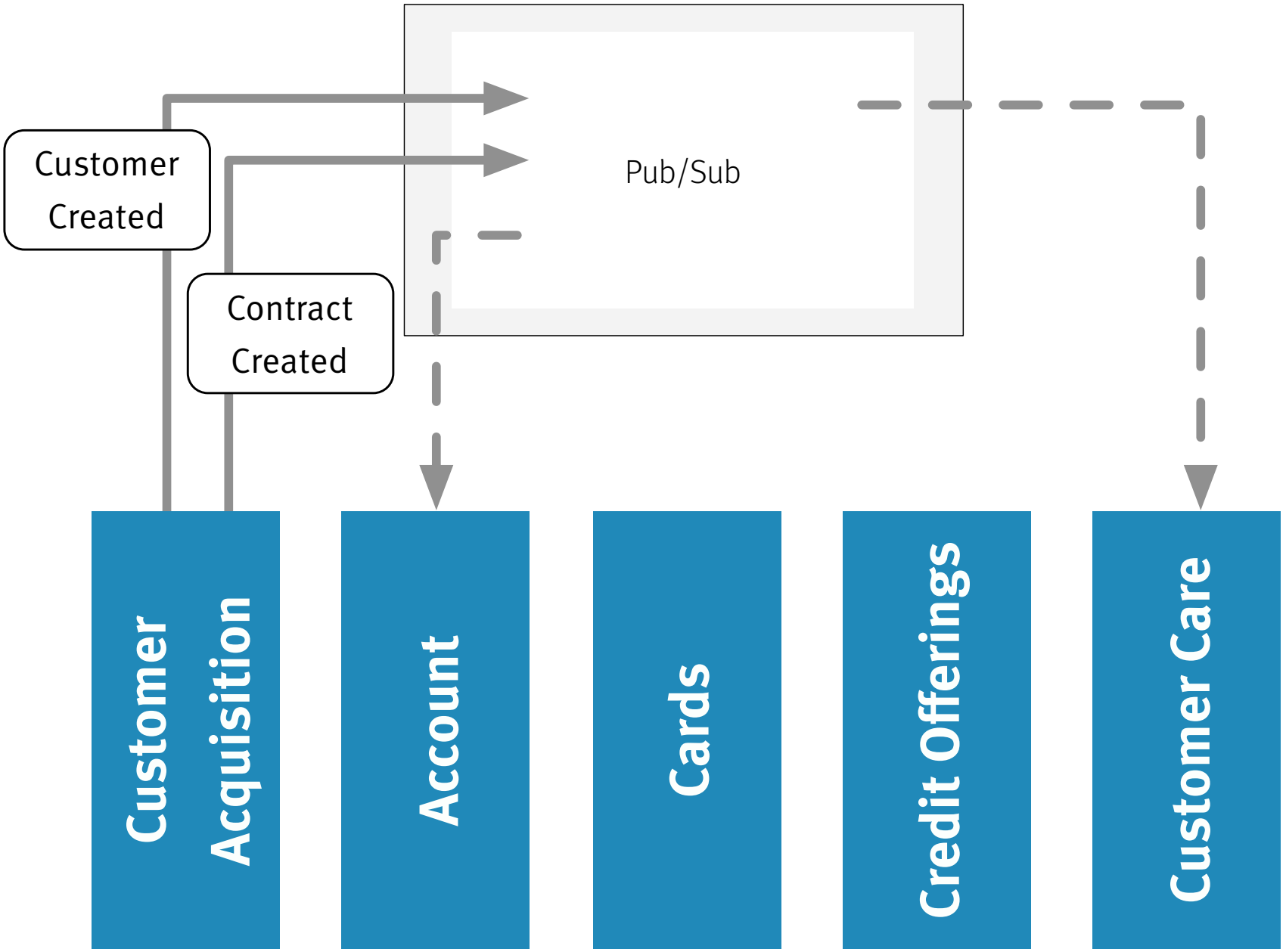

...
Customer Care
Customer Akquisition
Customer Care
Account
Cards
Credit Offerings
Customer Aquisition

Redundancy vs. Autonomy

Customer Acquisition



Customer Care


Customer Created
Contract Created
Pub/Sub
Customer Acquisition
Account
Cards
Credit Offerings
Customer Care

Customer Acquisition

Account

Cards

Credit Offerings

Customer Care

PO 1

PO 2

Team A

Team B

Team C

Wrap-up

Practical tips

- Record system design decisions:
  - Options considered
  - Options discarded
  - Reason for discarding
  - Advantages for current design
- Document assumptions, quality requirements and organizational constraints

Conclusion

- Finding sustainable, autonomous SCS can be a long-running process
- Right people: Domain experts, product owners and architects/engineers should work out the system design cooperatively
- There are a lot of aspects to consider and trade-offs to be made
- The iterative process of challenging and adapting the system design is never finished.

Thank you