



Mark Richards

Independent Consultant

Hands-on Software Architect, Published Author

Founder, DeveloperToArchitect.com

@markrichardssa

AP



Neal Ford

ThoughtWorks

Director / Software Architect / Meme Wrangler

<http://www.nealford.com>

@neal4d



Zhamak Dehghani

ThoughtWorks

Principle Consultant

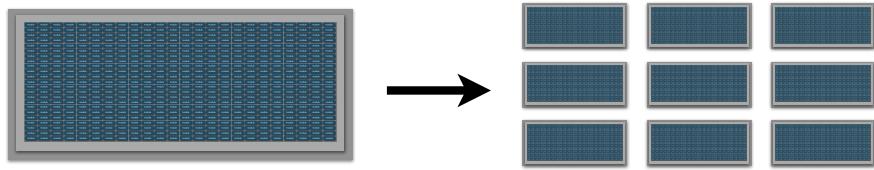
<https://www.thoughtworks.com/profiles/zhamak-dehghani>

@zhamakd

Architecture: The Hard Parts

Virtual Workshop

course agenda



part 1: pulling things apart

architectural modularity

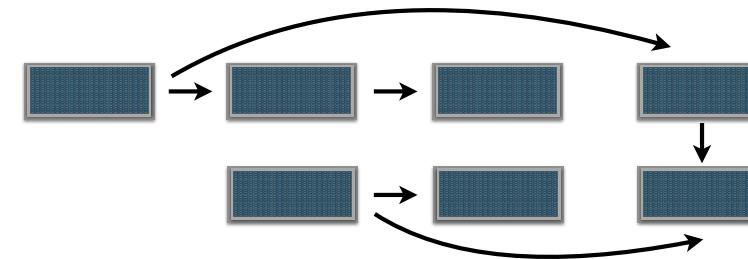
components as building blocks

granularity vs. modularity

domain vs. technical partitioning

breaking apart transactional data

breaking apart analytical data



part 2: putting them back together

synchronous vs. async communication

data ownership and access

orchestration and workflow

data mesh for analytical data

contract management

O'REILLY®



Fundamentals of Software Architecture

An Engineering Approach

Mark Richards & Neal Ford

Fundamentals of Software Architecture

by Mark Richards and Neal Ford

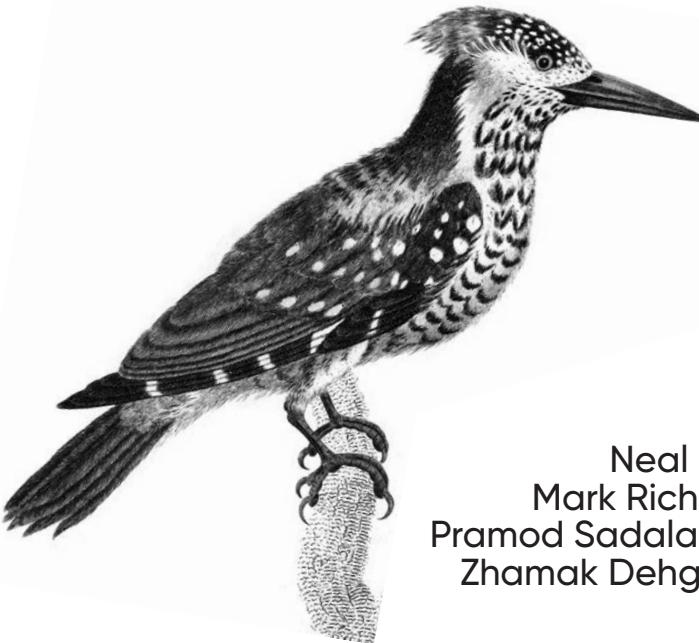
<https://www.amazon.com/gp/product/1492043451>



O'REILLY®

Software Architecture: The Hard Parts

Modern Tradeoff Analyses



Neal Ford,
Mark Richards,
Pramod Sadalage &
Zhamak Dehghani

Software Architecture: The Hard Parts

by Neal Ford, Mark Richards, Zhamak Dehghani,
Pramod Sadalage

Release date: Fall 2021

COMING SOON



Architecture Katas

fundamentalsofsoftwarearchitecture.com

Architectural Katas About Architectural Katas Fundamentals of Software Architecture List of Architecture Katas

Architectural Katas

inspired by Ted Neward's original [Architectural Katas](#)

"How do we get great designers?
Great designers design,
of course."
Fred Brooks

"So how are we supposed to get great architects, if
they only get the chance to architect fewer than
a half-dozen times in their career?"
Ted Neward

About

Architectural Katas are intended as a small-group (3-5 people) exercise, usually as part of a larger group (4-10 groups are ideal), each of whom is doing a different kata. A Moderator keeps track of time, assigns Katas (or allows this website to choose one randomly), and acts as the facilitator for the exercise.

Each group is given a project (in many ways, an RFP—Request For Proposal) that needs development. The project team meets for a while, discovers requirements that aren't in the original proposal by asking questions of the "customer" (the Moderator), discusses technology options that could work, and sketches out a rough vision of what the solution could look like. Then, after they've discussed for a while, the project team must present their solution to the other project teams in the room, and answer challenges (in the form of hard-but-fair questions) from the other project teams. Once that challenge phase is done, the room votes on their results, and the next project team takes the floor.

Rules

Doing an Architectural Kata requires you to obey a few rules in order to get the maximum out of the activity. [Read Rules »](#)

Rules

The rules are broken down by the different Phases of the exercise. However, one rule trumps all the others: **Any other questions that are not already covered by these rules, you may ask the Moderator about.** When in doubt, ask.

The Sysops Squad

Best Electronics is a large electronics giant that has numerous retail stores throughout the country. When customers buy computers, TV's, stereos, and other electronic equipment, they can choose to purchase a support plan. Customer-facing technology experts (the "Sysops Squad") will then come to the customers residence (or work office) to fix problems with the electronic device.



Sysops Squad - A Bad Situation...

Things have not been good with the Sysops Squad lately. The current trouble ticket system is a large monolithic application that was developed many years ago. Customers are complaining that consultants are never showing up due to lost tickets, and often times the wrong consultant shows up to fix something they know nothing about. Customers and call-center staff have been complaining that the system is not always available for web-based or call-based problem ticket entry. Change is difficult and risky in this large monolith - whenever a change is made, it takes too long and something else usually breaks. Due to reliability issues, the monolithic system frequently "freezes up" or crashes - they think it's mostly due a spike in usage and the number of customers using the system. If something isn't done soon, Best Electronics will be forced to abandon this very lucrative business line and fire all of the experts (including you, the architect).

Current process in the monolithic system:

1. Sysops squad experts are added and maintained in the system through an administrator, who enters in their locale, availability, and skills.
2. Customers who have purchased the support plan can enter a problem ticket using the sysops squad website. Customer registration for the support service is part of the system. The system bills the customer on an annual basis when their support period ends by charging their registered credit card.
3. Once a trouble ticket is entered in the system, the system then determines which sysops squad expert would be the best fit for the job based on skills, current location, service area, and availability (free or currently on a job).
4. The sysops squad expert is then notified via a text message that they have a new ticket. Once this happens an email or SMS text message is sent to the customer (based on their profile preference) that the expert is on their way.
5. The sysops squad expert then uses a custom mobile application on their phone to access the ticketing system to retrieve the ticket information and location. The sysops squad expert can also access a knowledge base through the mobile app to find out what things have been done in the past to fix the problem.
6. Once the sysops squad expert fixes the problem, they mark the ticket as "complete". The sysops squad expert can then add information about the problem and fix to the knowledge base.
7. After the system receives notification that the ticket is complete, the system sends an email to the customer with a link to a survey which the customer then fills out.

The Sysops Squad

Penultimate Electronics is a large electronics giant that has numerous retail stores throughout the country. When customers buy computers, TV's, stereos, and other electronic equipment, they can choose to purchase a support plan. Customer-facing technology experts (the “Sysops Squad”) will then come to the customers residence (or work office) to fix problems with the electronic device.



Sysops Squad - A Bad Situation...

Things have not been good with the Sysops Squad lately, and if something isn't done soon, the company will be forced to abandon this very lucrative business line and fire all of the experts (including you, the architect).

1. The current trouble ticket system is a large monolithic application that was developed many years ago.
2. Customers are complaining that consultants are never showing up due to lost tickets.
3. Often times the wrong consultant shows up to fix something they know nothing about.
4. Customers have been complaining that the system is not always available for web-based problem ticket entry.
5. Change is difficult and risky in this large monolith - whenever a change is made, it takes too long and something else usually breaks (the team now calls new feature releases "bug releases").
6. Due to reliability issues, the monolithic system frequently "freezes up" or crashes - they think it's mostly due to an increase in usage and the number of customers using the system.

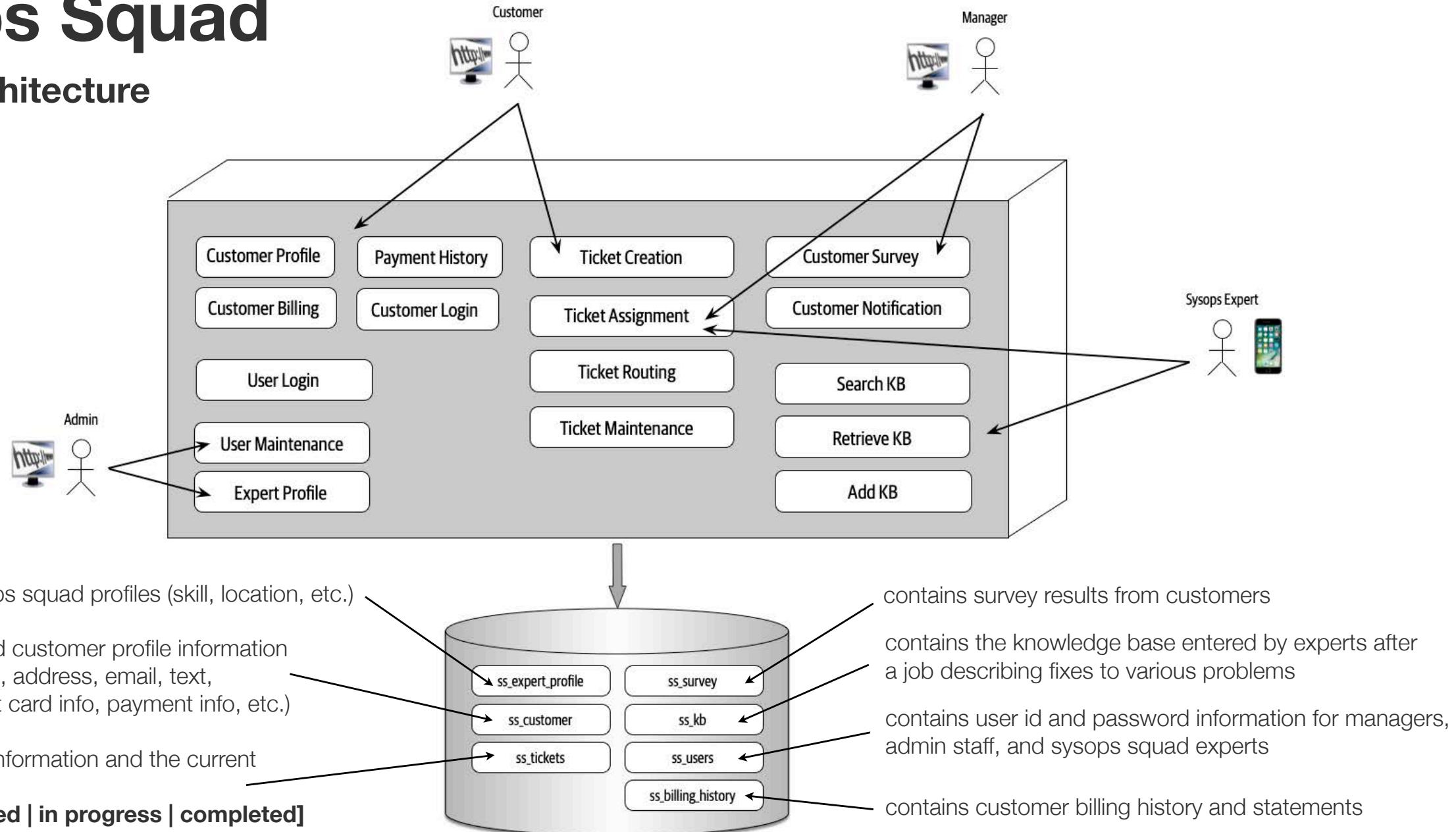
Sysops Squad - A Bad Situation...

Current process in the monolithic system

1. Sysops squad experts are added and maintained in the system through an administrator, who enters in their locale, availability, and skills.
2. Customers who have purchased the support plan can enter a problem ticket using the sysops squad website.
3. Once a problem ticket is entered in the system, the system then determines which sysops squad expert would be the best fit for the job based on skills, current location, service area, and availability.
4. Once an expert is found, the ticket is assigned to the expert and they are notified through a mobile app on their phone and the customer is notified that the expert is on their way.
5. The sysops squad expert uses a custom mobile application on their phone to access the ticket information and location. The sysops squad expert can also access a knowledge base through the mobile app to find out what things have been done in the past to fix the problem.
6. Once the sysops squad expert fixes the problem, they mark the ticket as “complete”. The sysops squad expert can then add information about the problem and fix to the knowledge base.
7. After the system receives notification that the ticket is complete, the system send an email to the customer with a link to a survey which the customer then fills out.

Sysops Squad

Current Architecture





Part I: Pulling Things Apart

Attempting to divide a cohesive module would only result in increased coupling and decreased readability.
—Larry Constantine



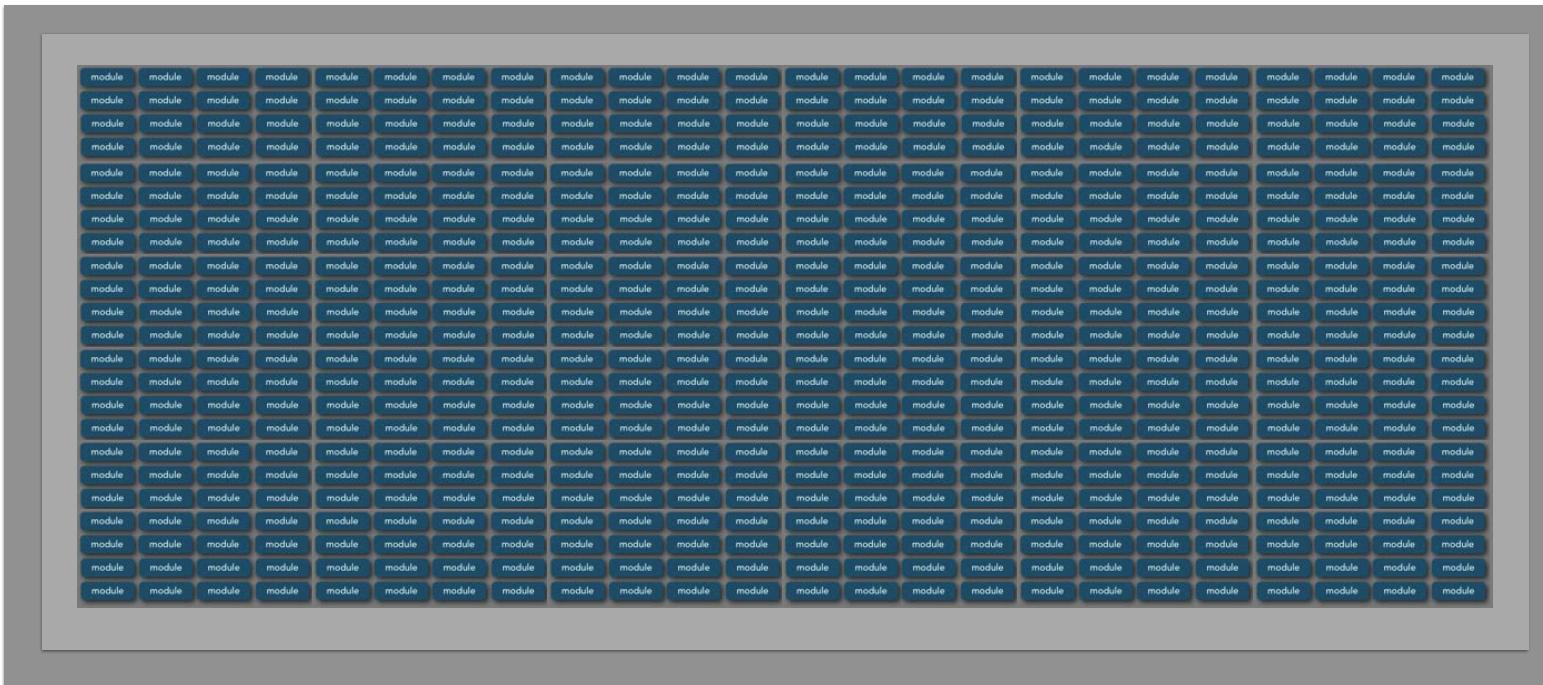
Architectural Modularity

architectural modularity

“when should I consider breaking apart my application into smaller distributed components?”



architectural modularity



architectural modularity













architectural modularity

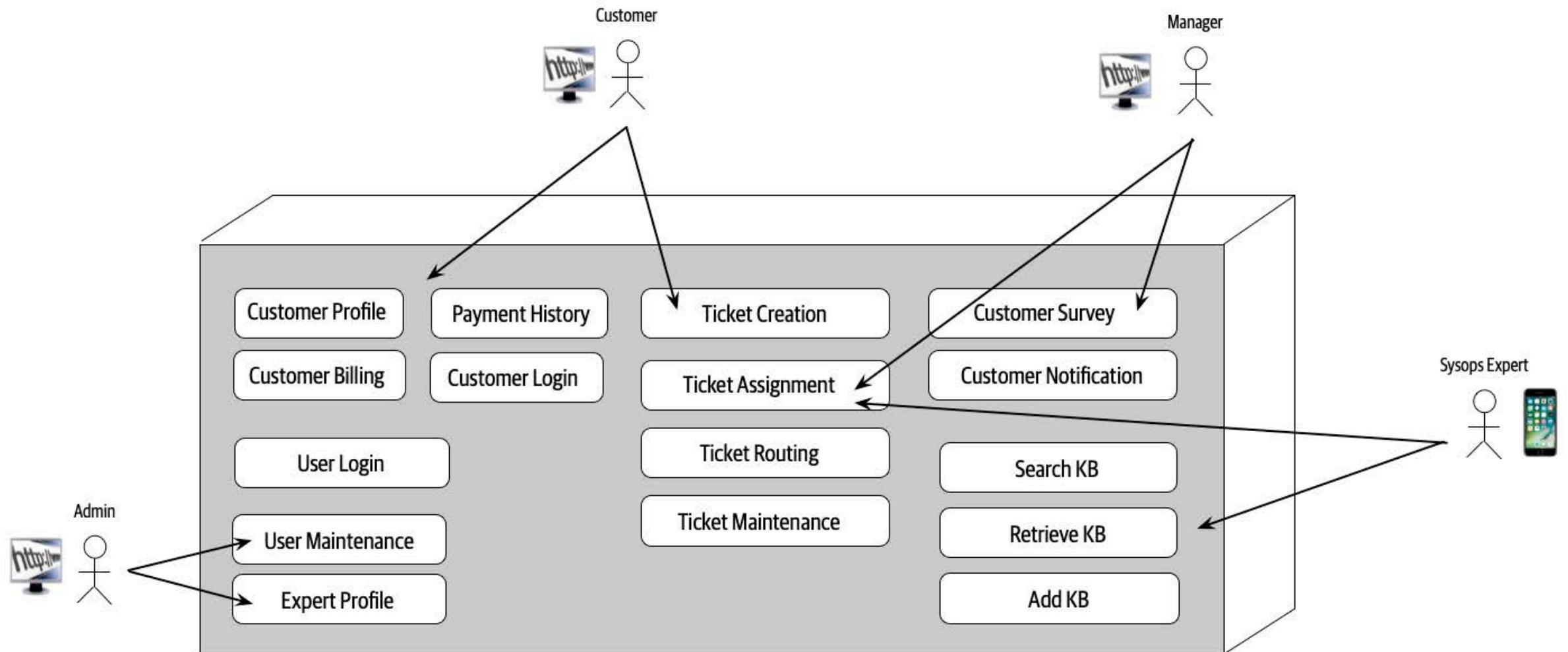




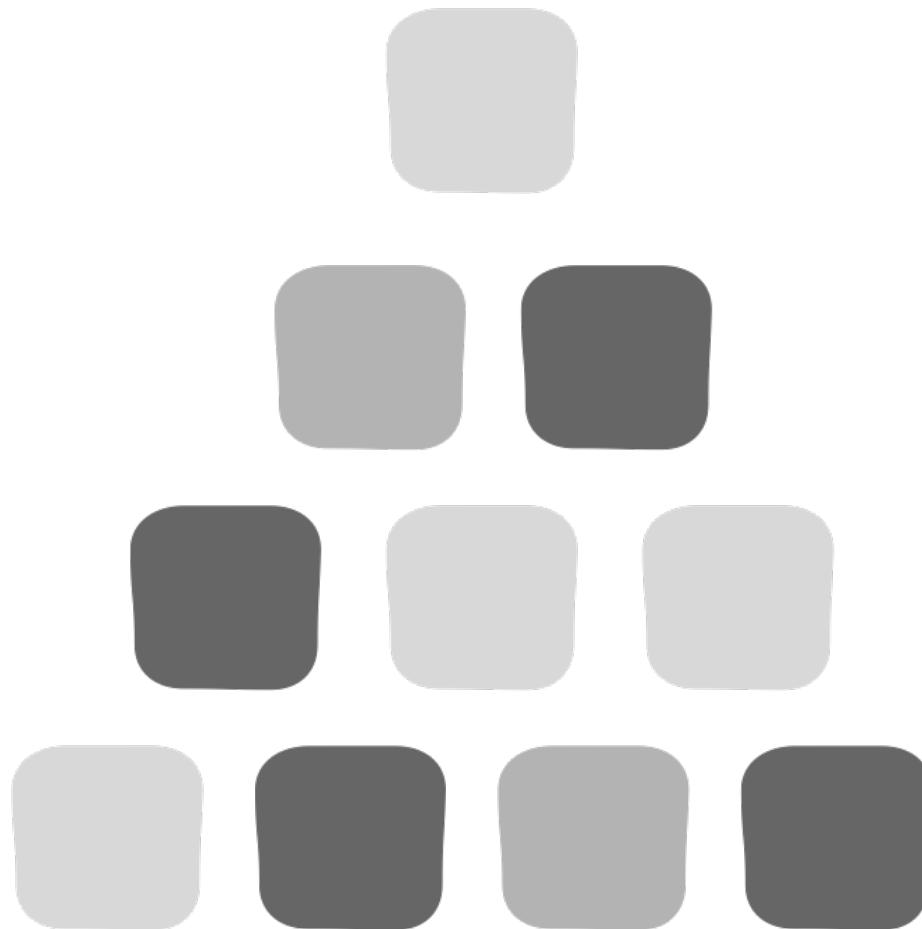
Components as Building Blocks

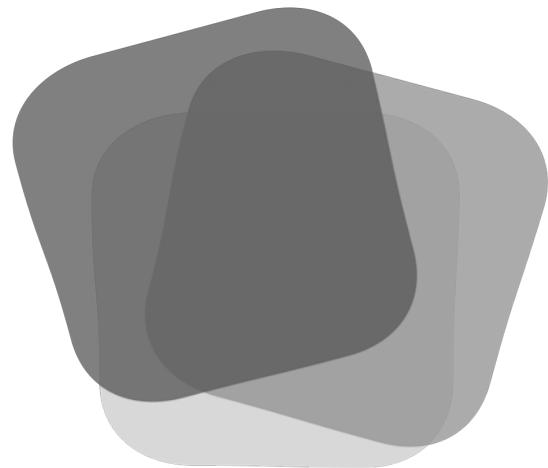
Kata Exercise - Components and Quanta

Based on the components and data below, identify the architecture quanta and components included in each quantum

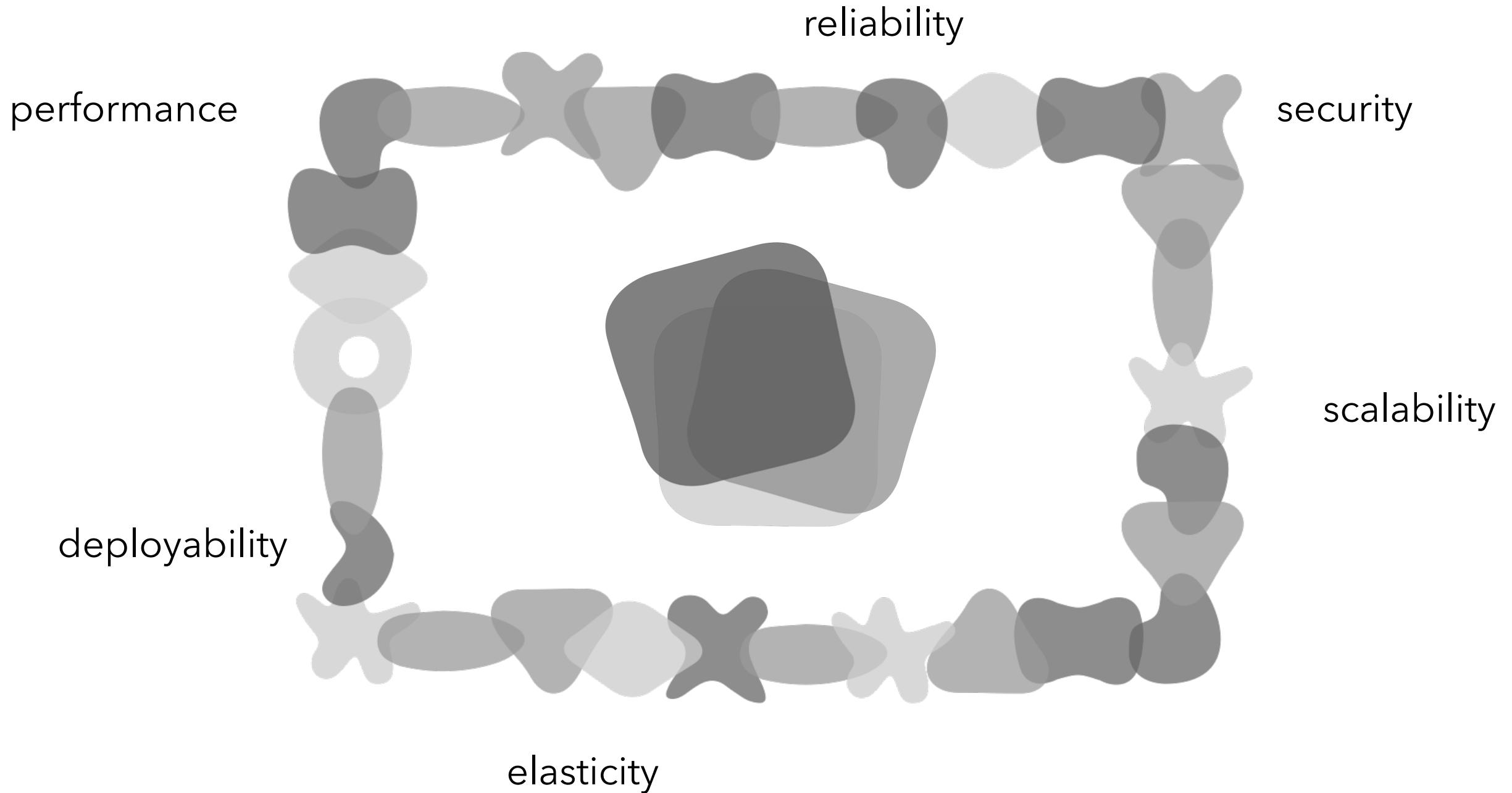


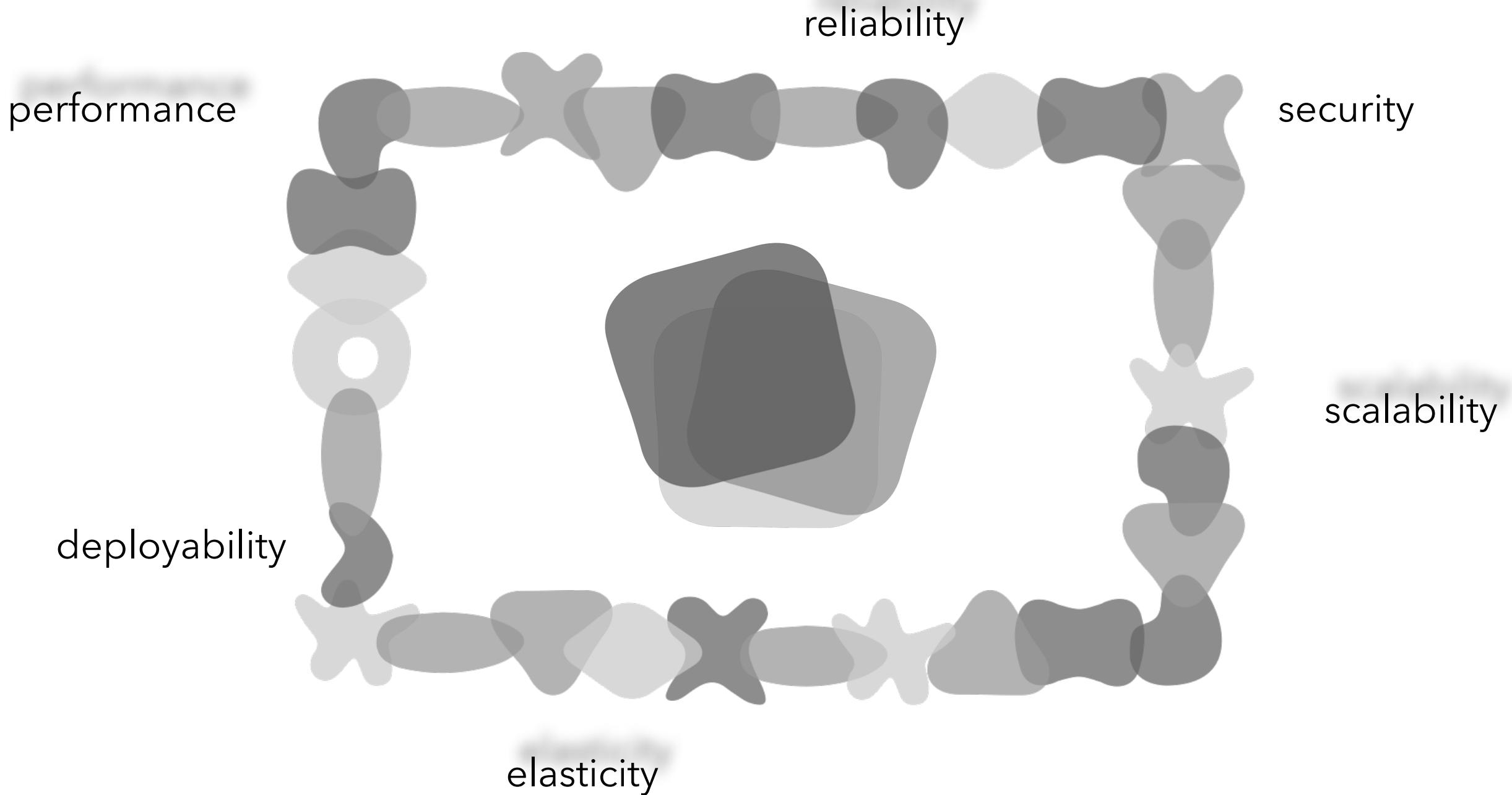
structural design in architecture



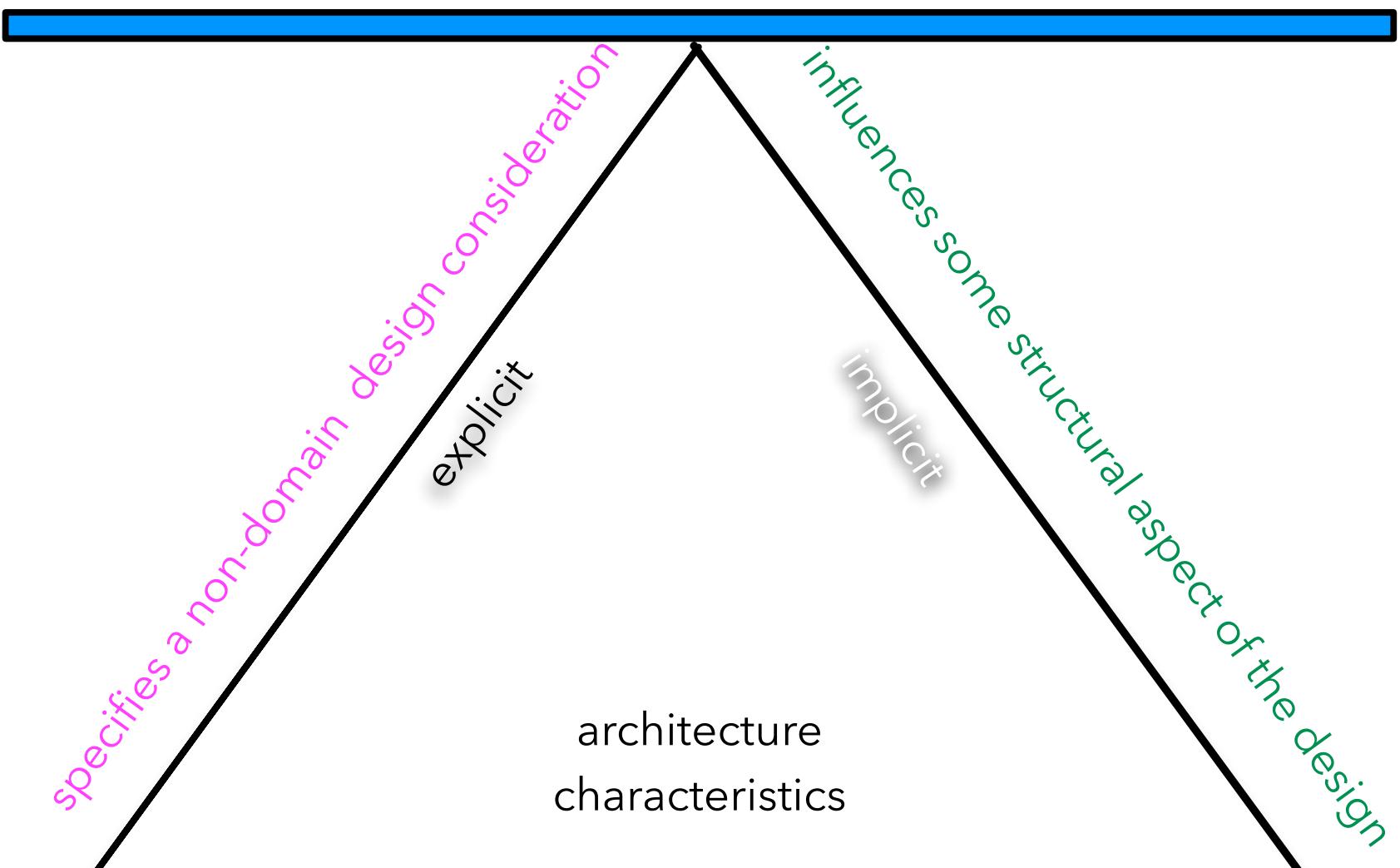


requirements | use cases | story cards | DDD event-storm output | ?

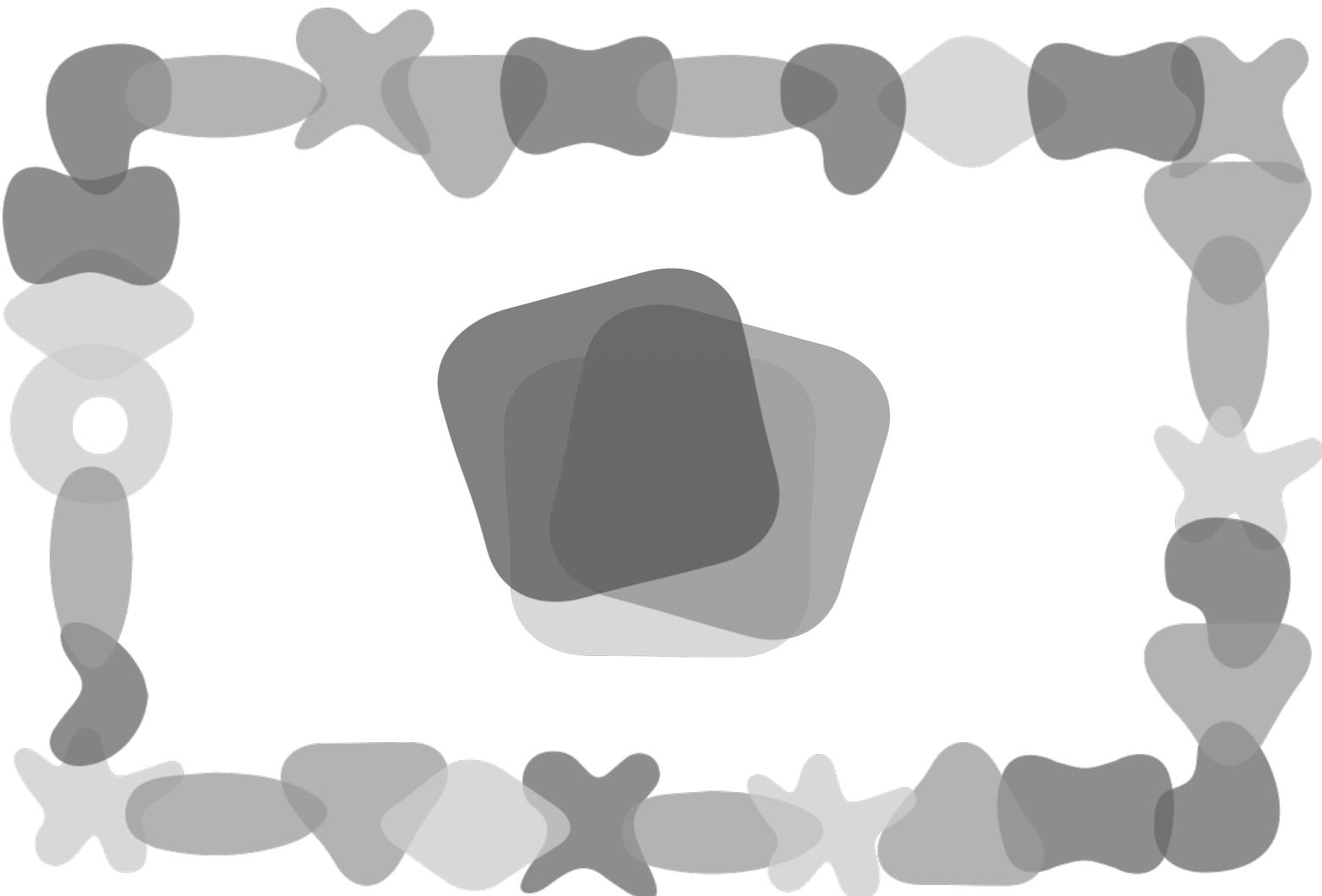


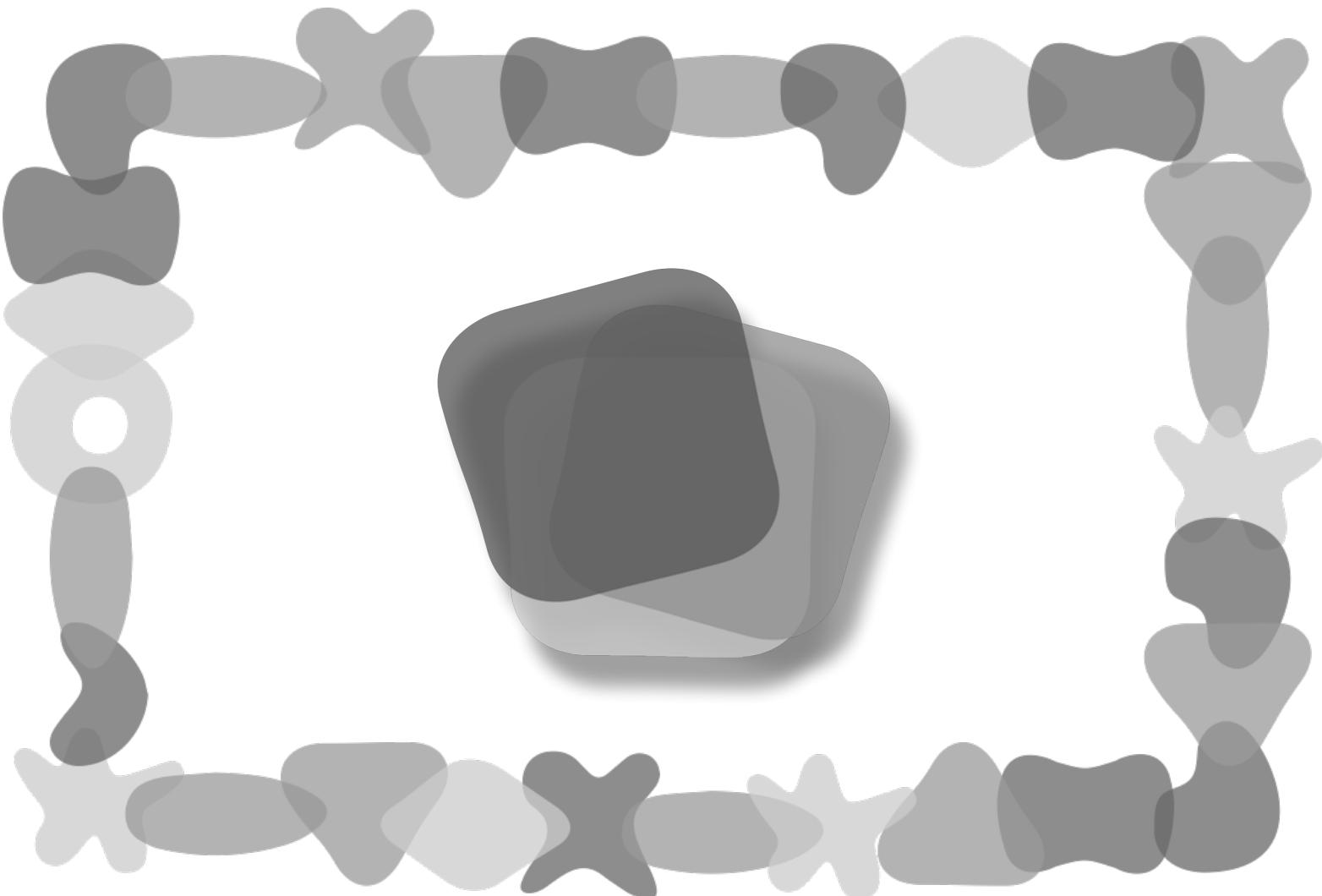


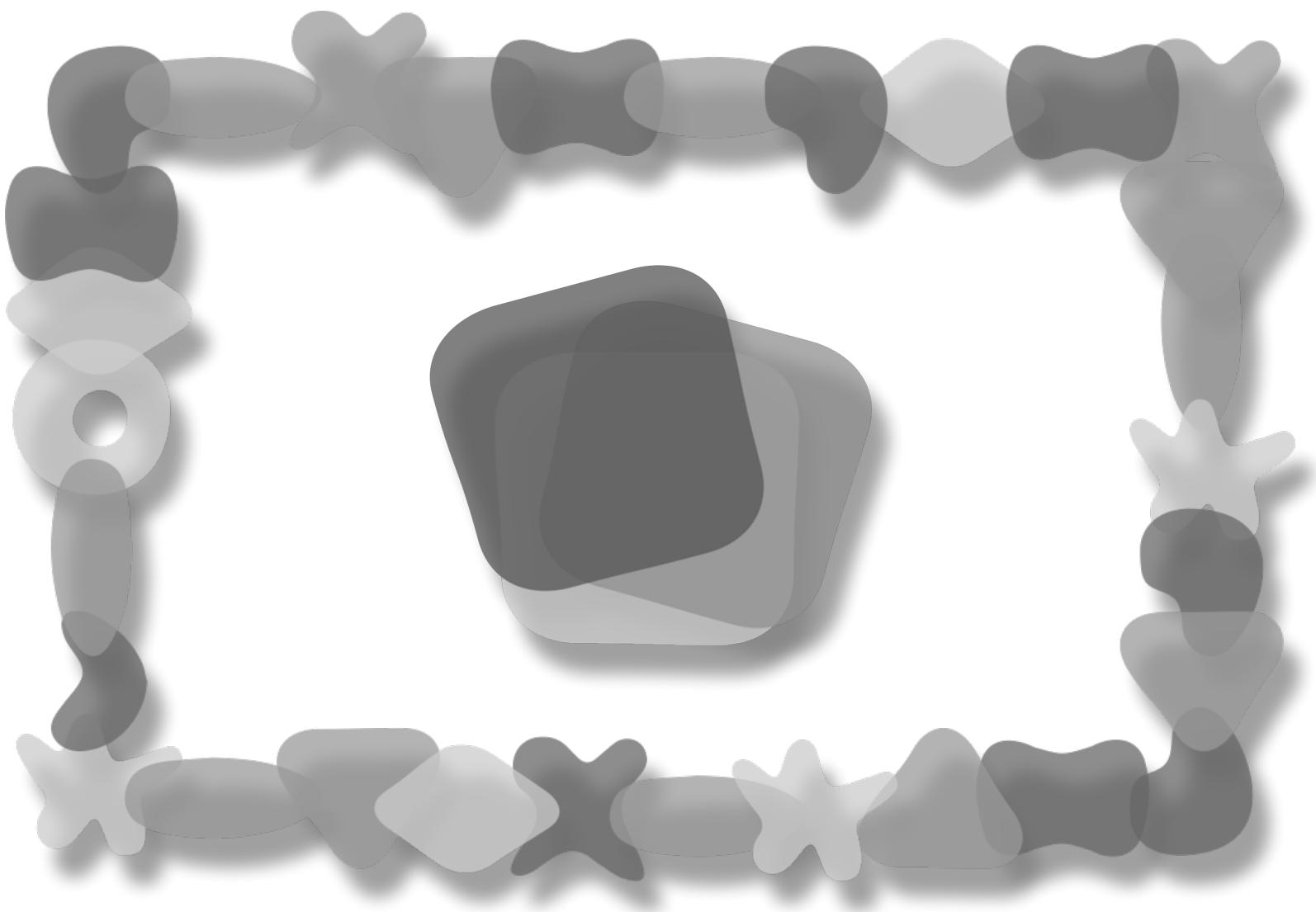
design



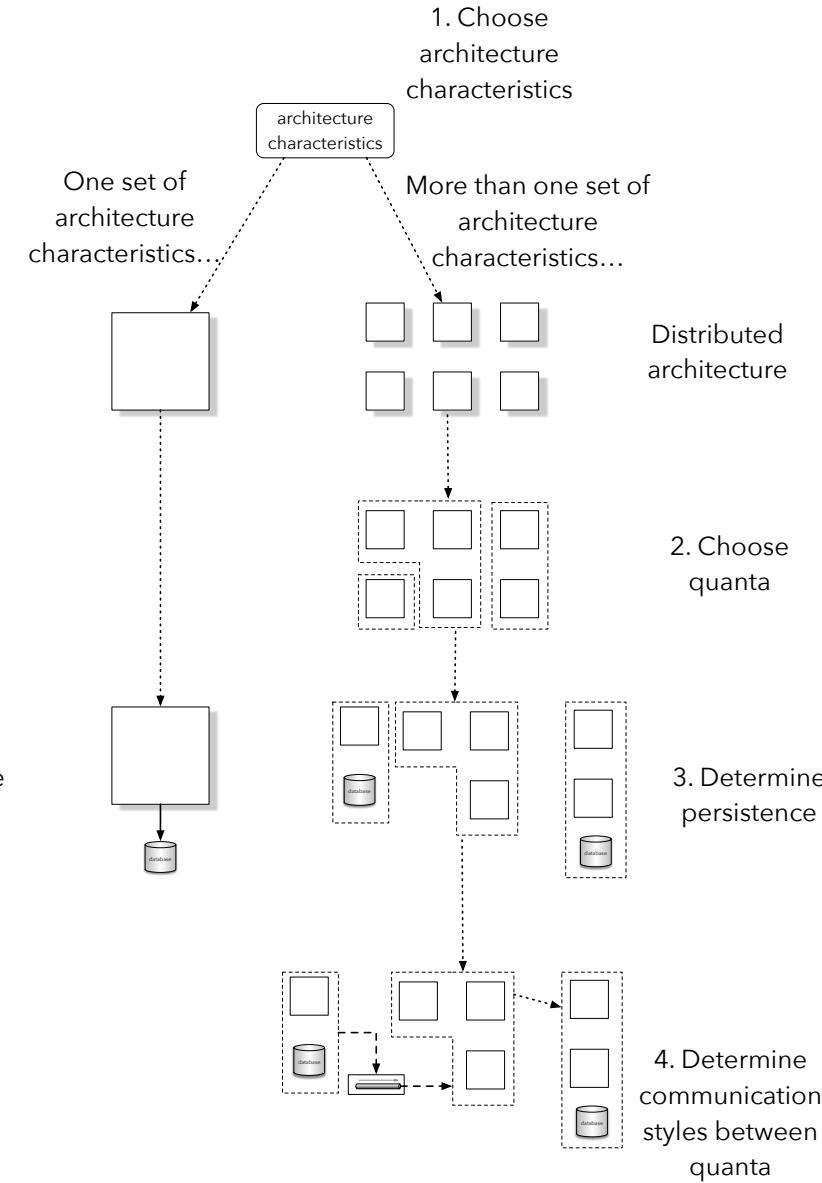
critical or important to application success



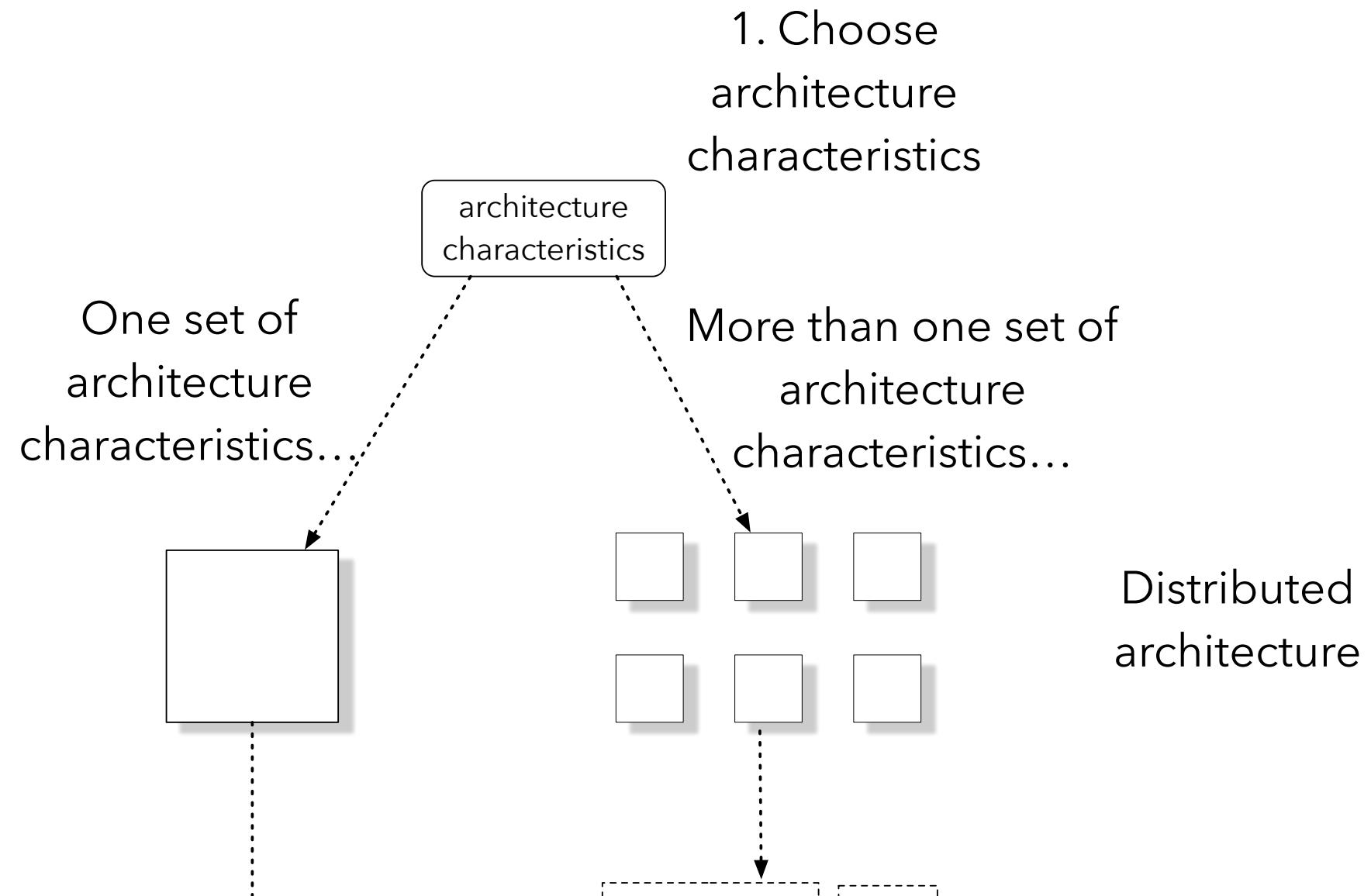




monolith | distributed?



monolith | distributed?



Your Architectural Kata is...

Going Going Gone!

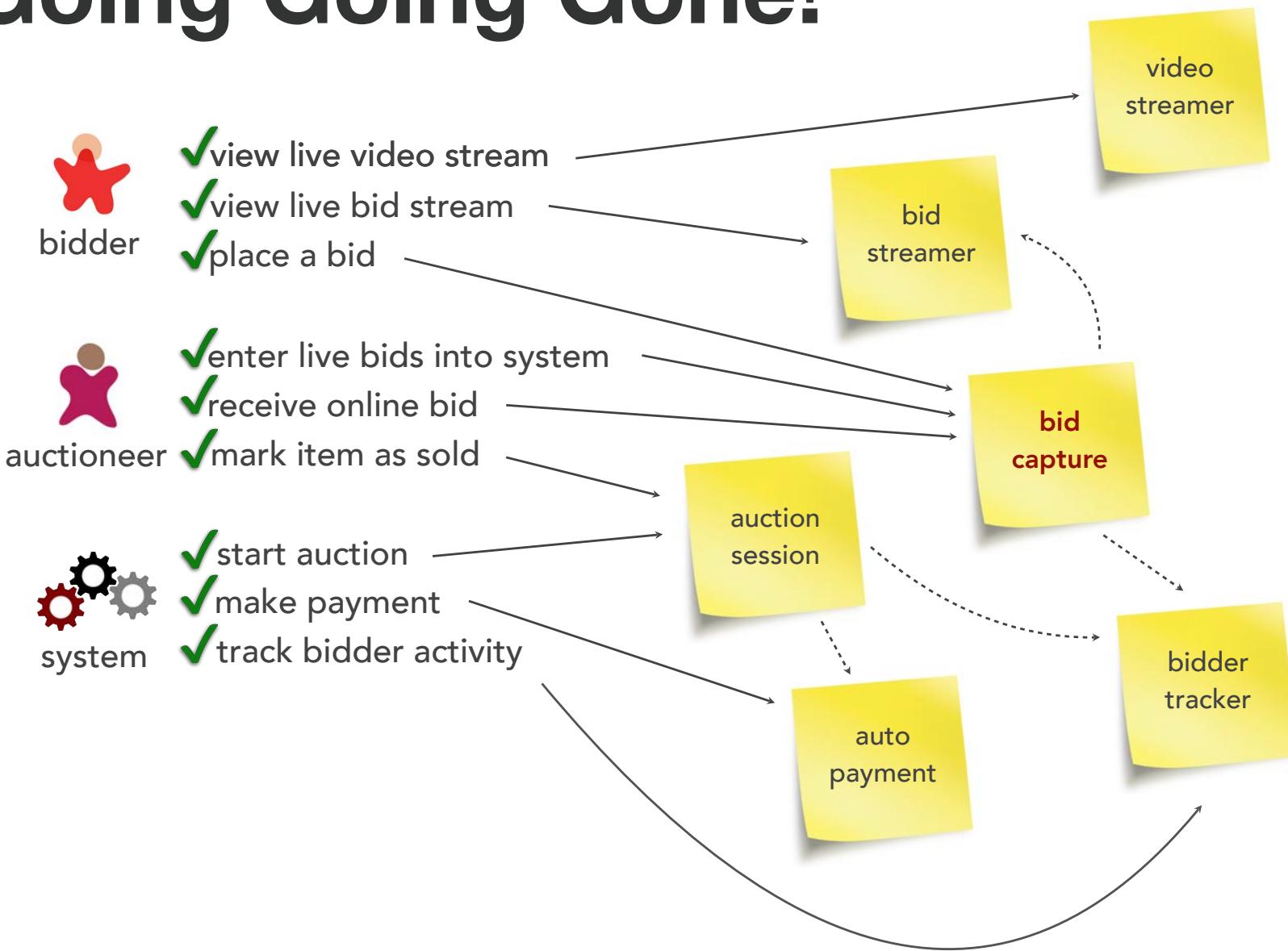
An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users:** scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements:**
 - bidders can see a live video stream of the auction and see all bids as they occur
 - auctions must be as real-time as possible
 - both online and live bids must be received in the order in which they are placed
 - bidders register with credit card; system automatically charges card if bidder wins
 - participants must be tracked via a reputation index
- **Additional Context:**
 - auction company is expanding aggressively by merging with smaller competitors
 - if nationwide auction is a success, replicate the model overseas
 - budget is not constrained--this is a strategic direction
 - company just exited a lawsuit where they settled a suit alleging fraud

availability reliability performance scalability elasticity security

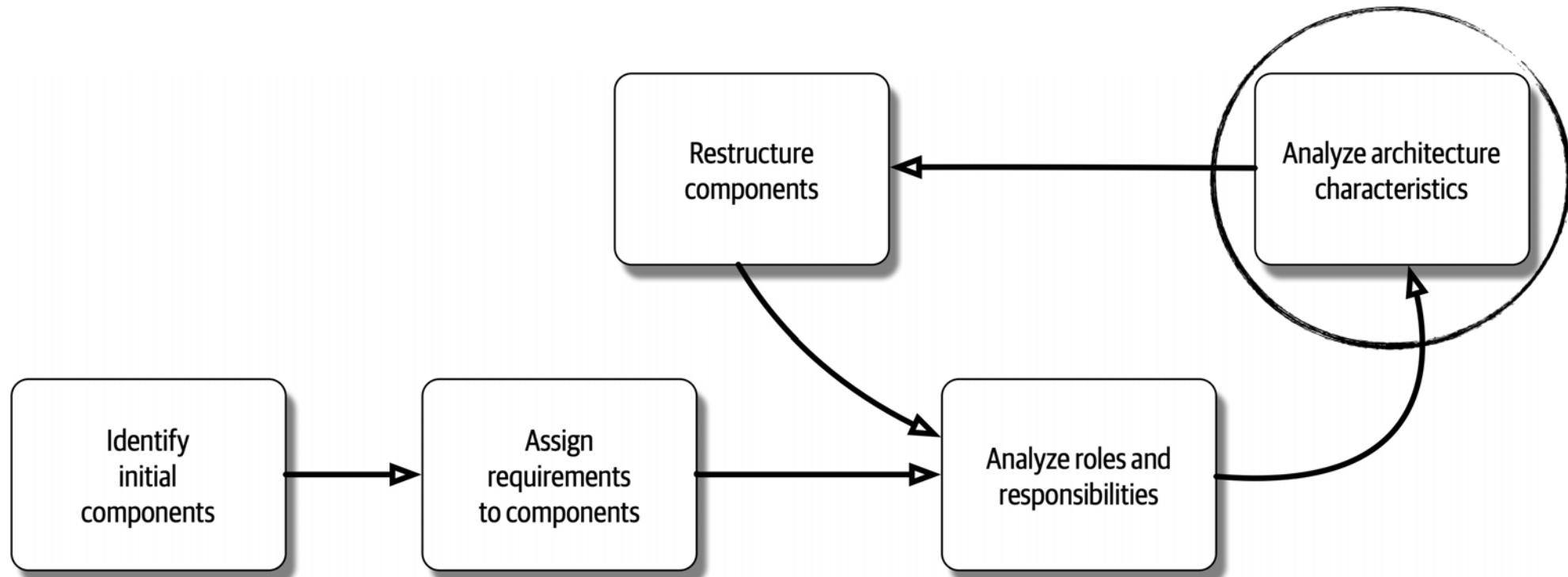
Your Architectural Kata is...

Going Going Gone!



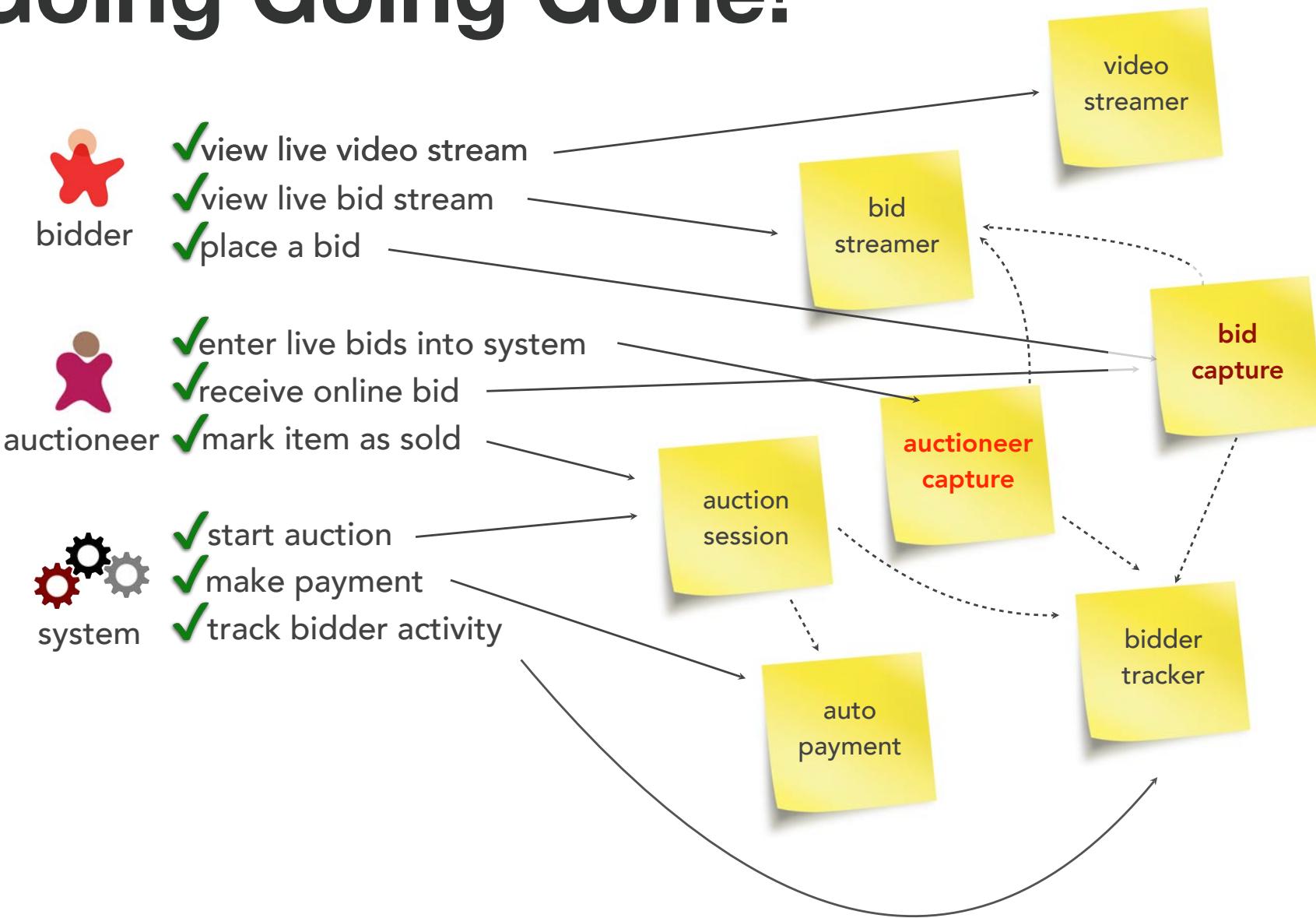
Your Architectural Kata is...

Going Going Gone!

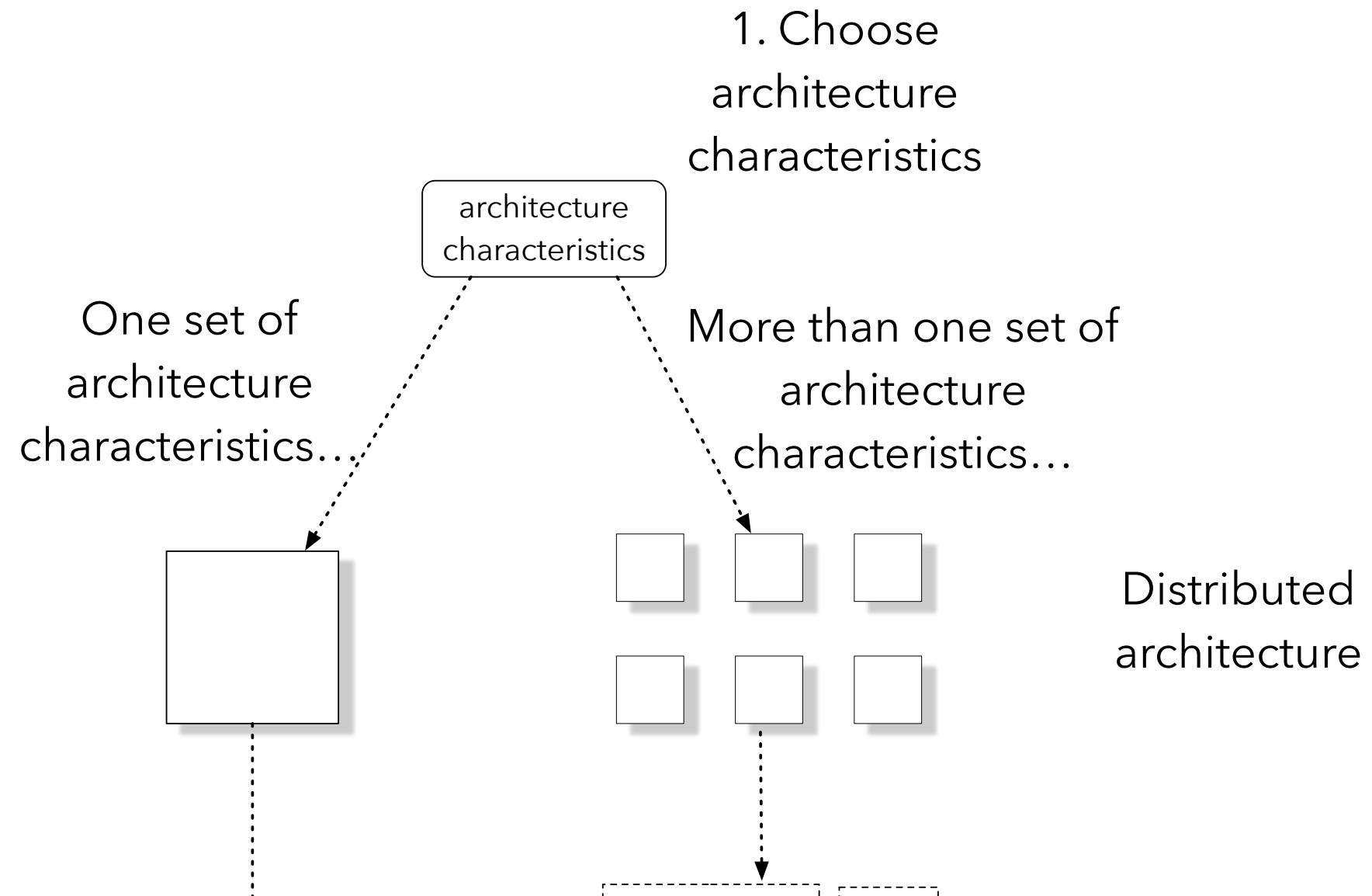


Your Architectural Kata is...

Going Going Gone!



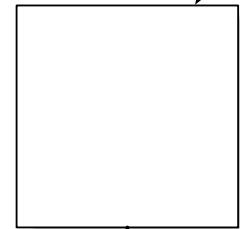
monolith | distributed?



monolith | distributed?

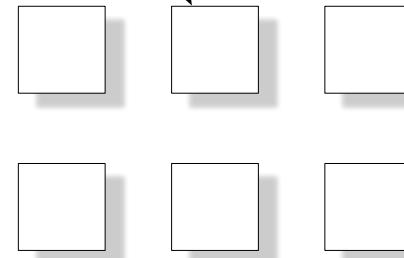
Monolithic
architecture

One set of
architecture
characteristics...



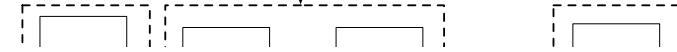
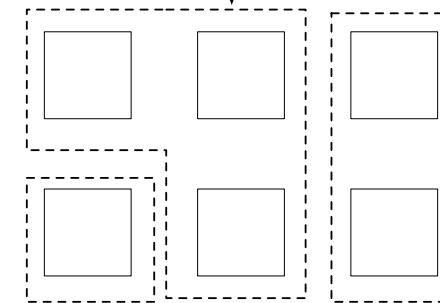
architecture
characteristics

More than one set of
architecture
characteristics...

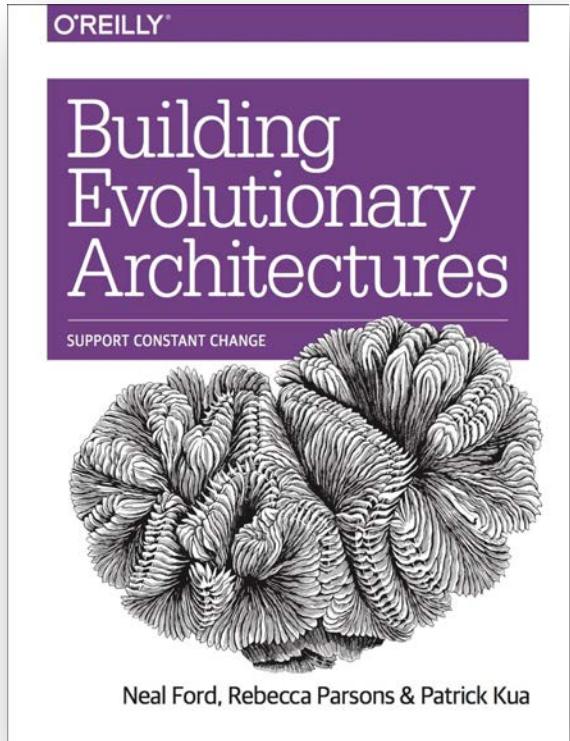


Distributed
architecture

2. Choose
quanta

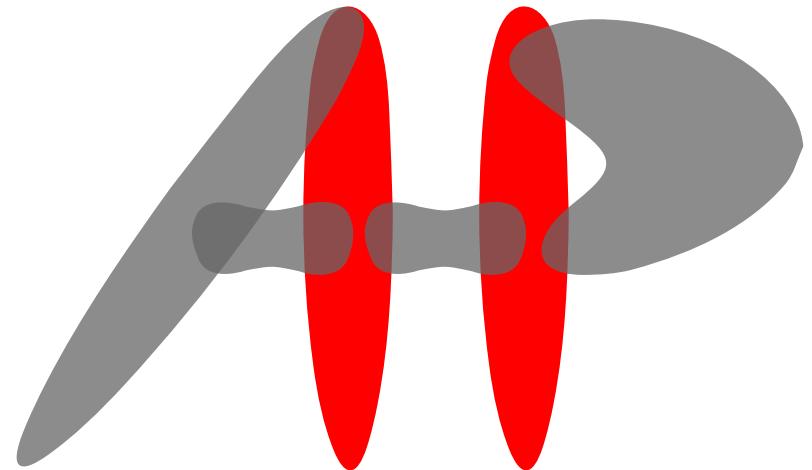


architectural quantum



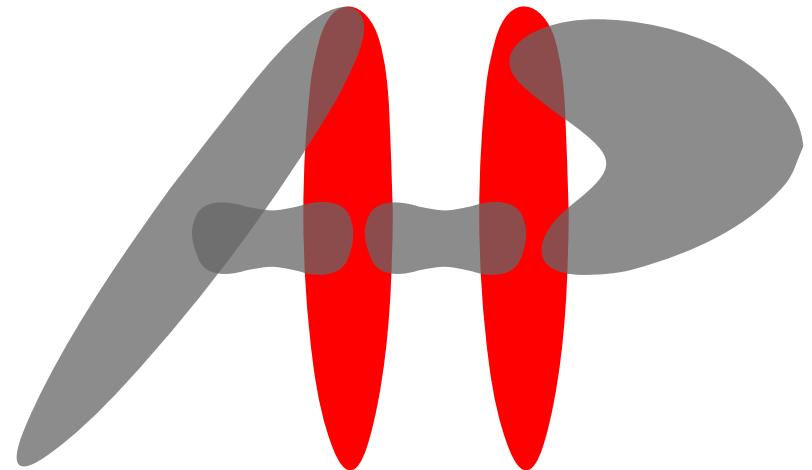
An architectural quantum is an independently deployable component with high functional cohesion .

architectural quantum 2021



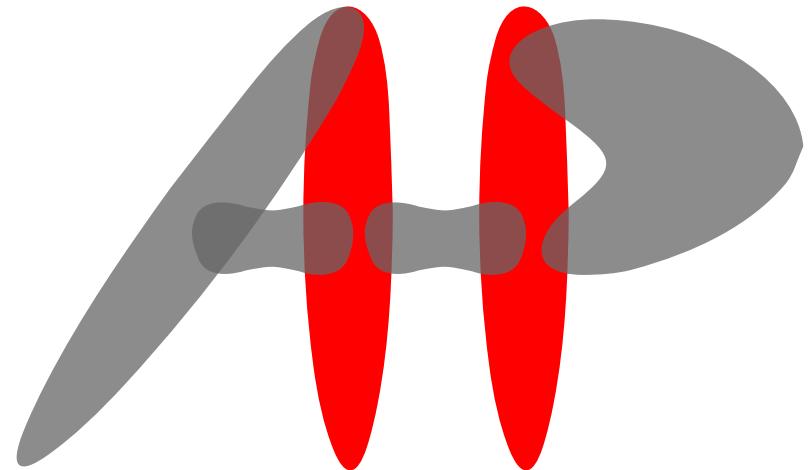
an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling.

architectural quantum 2021



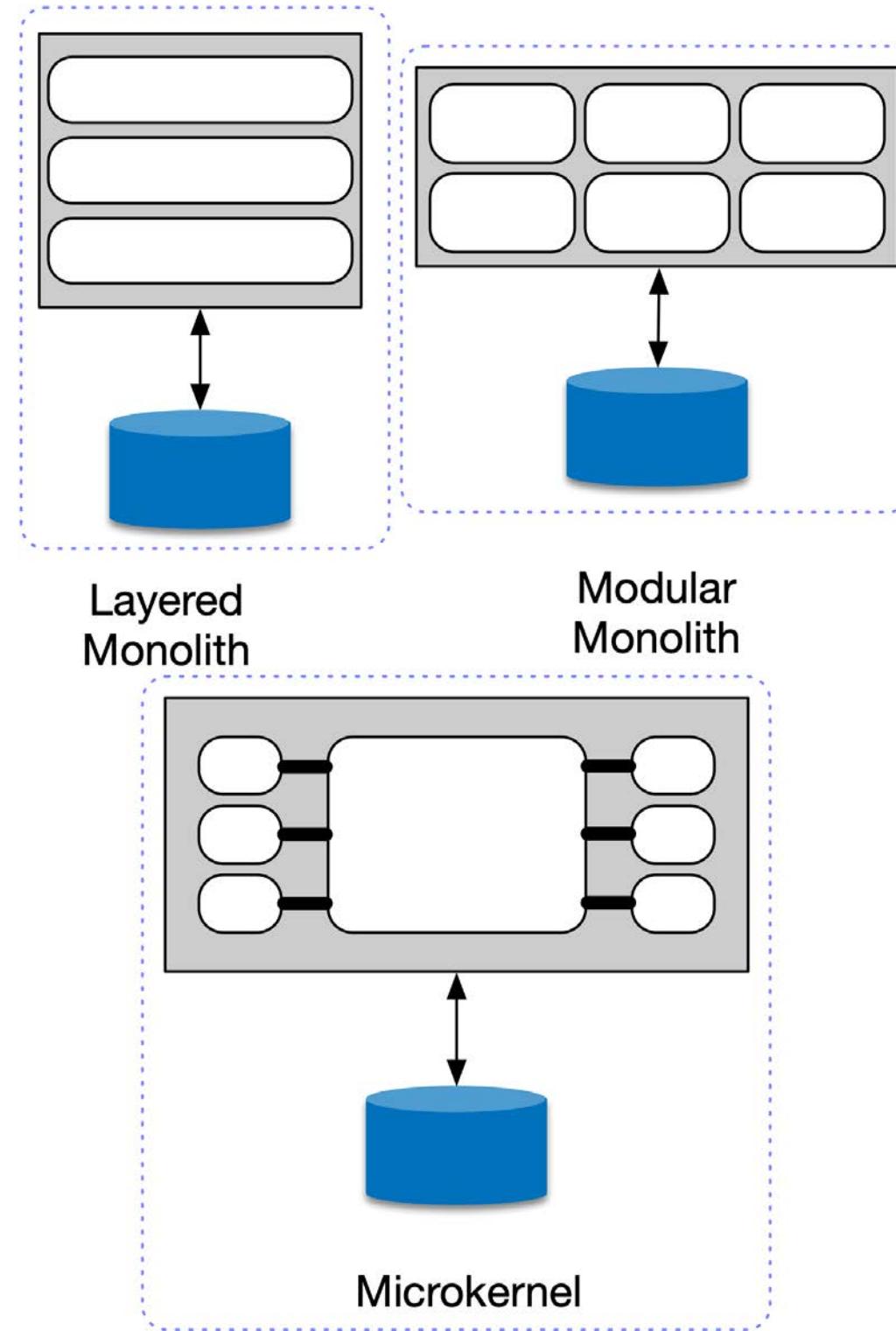
an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling.

architectural quantum 2021

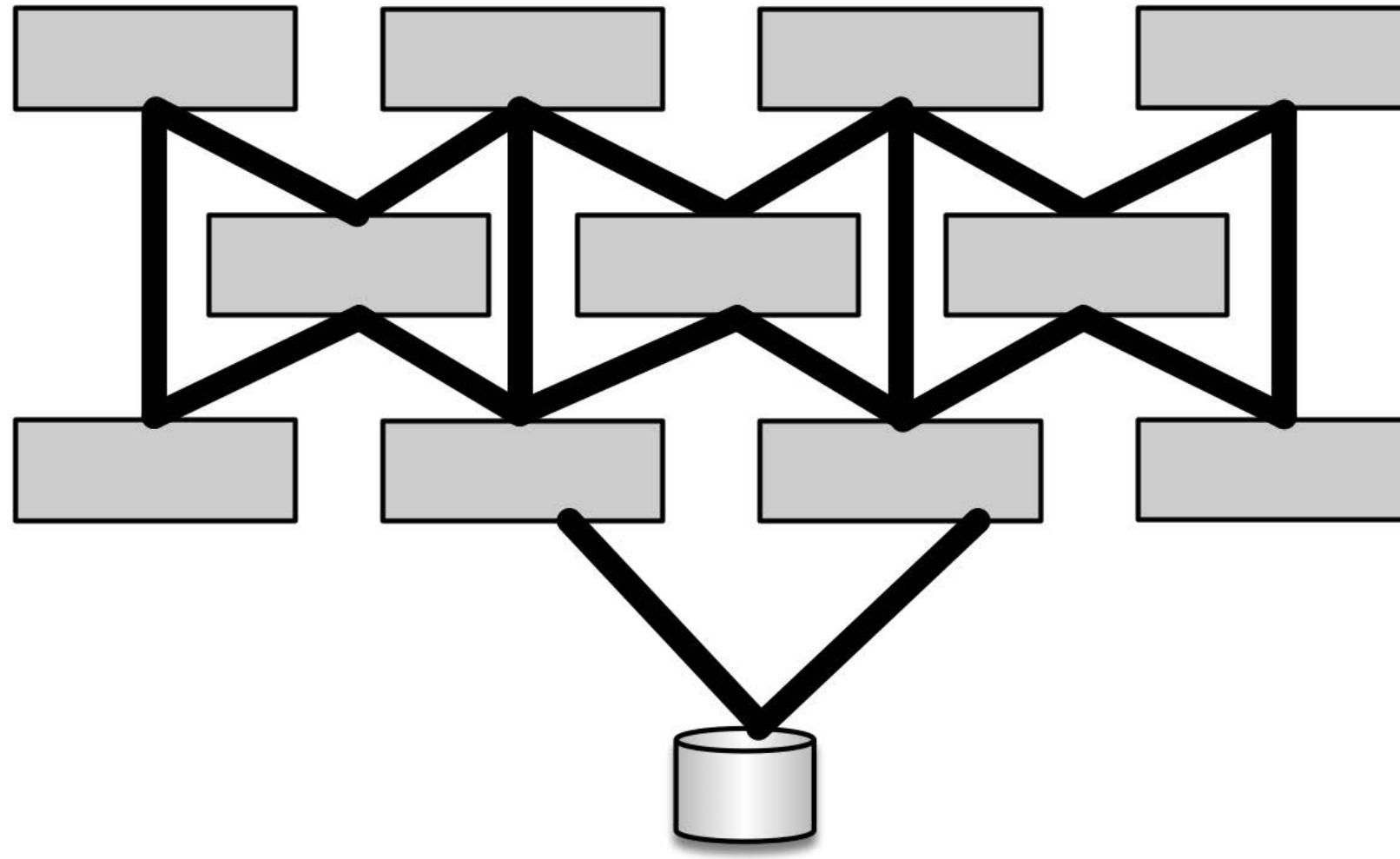


an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling.

Single Static Quantum

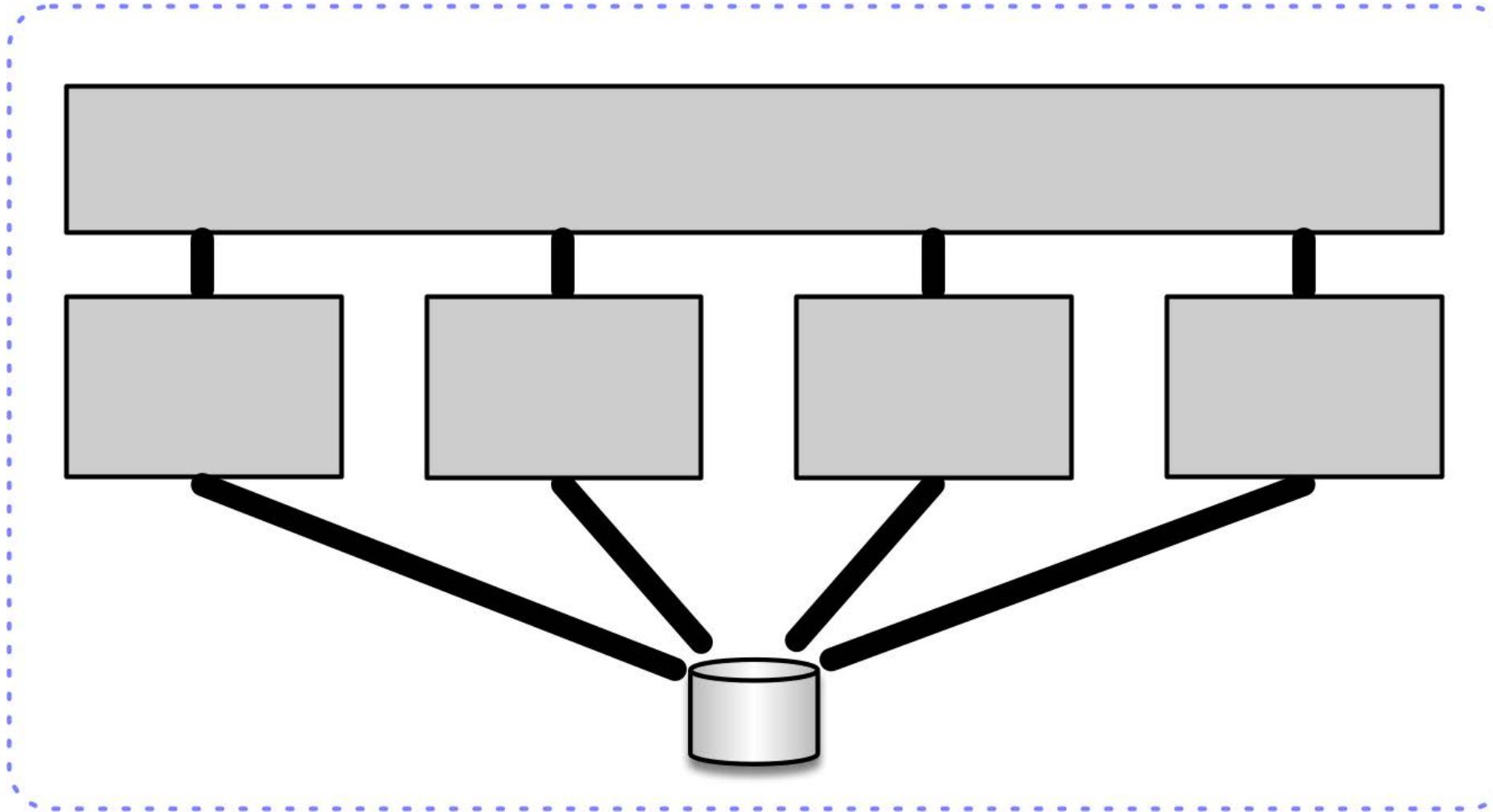


Single Static Quantum



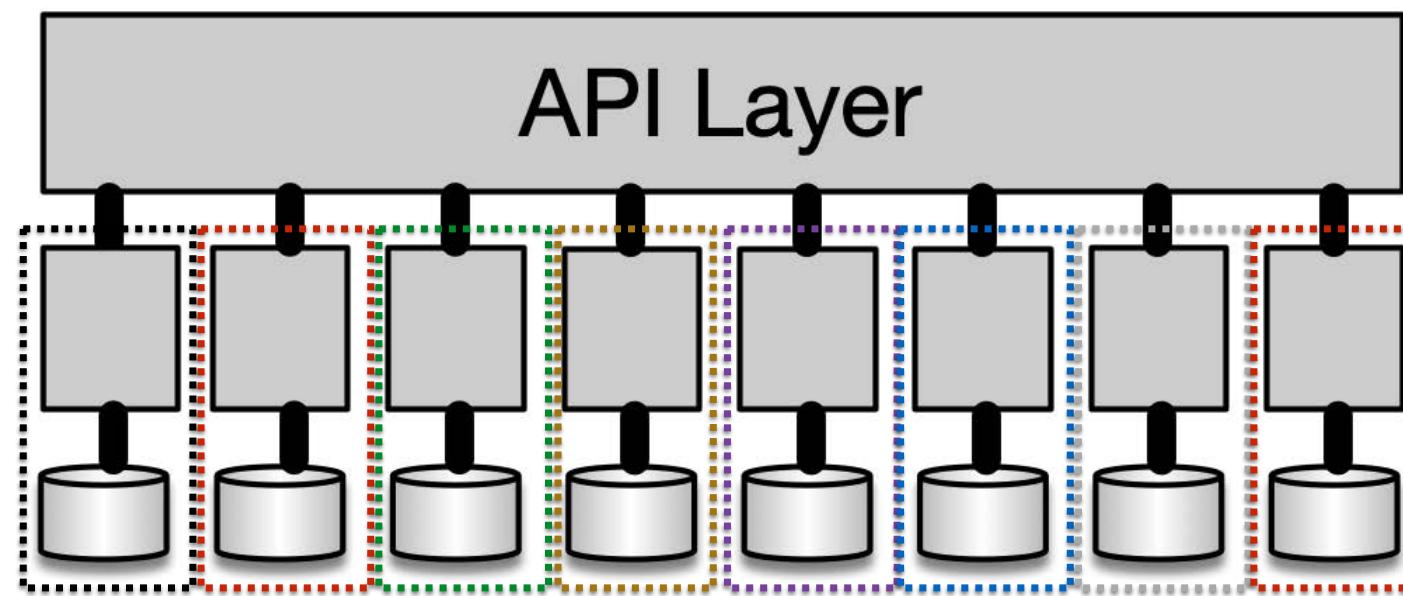
Event-driven
Architecture

Single Static Quantum

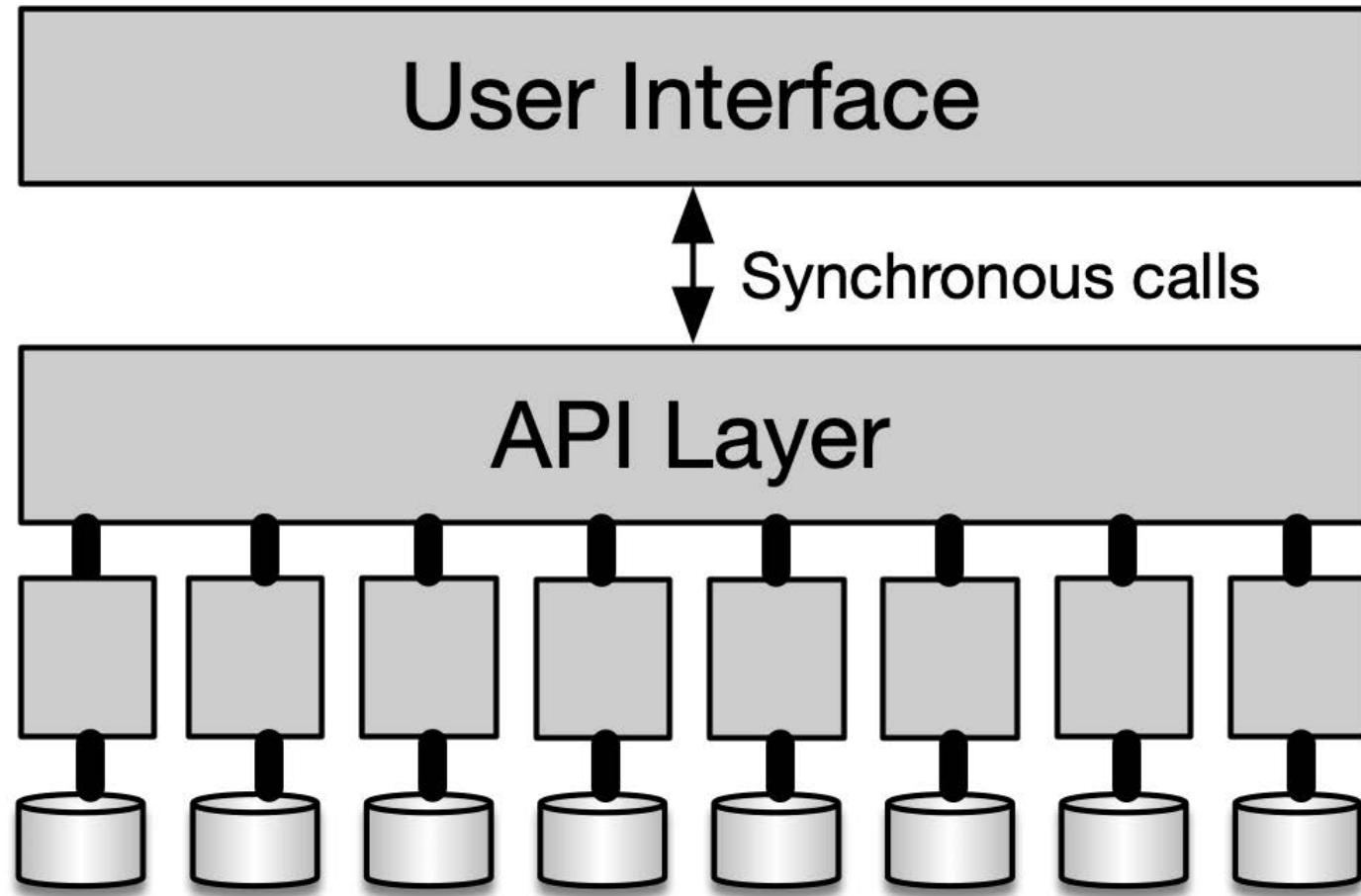


Service-based
Architecture

Microservices Quanta (no UI)

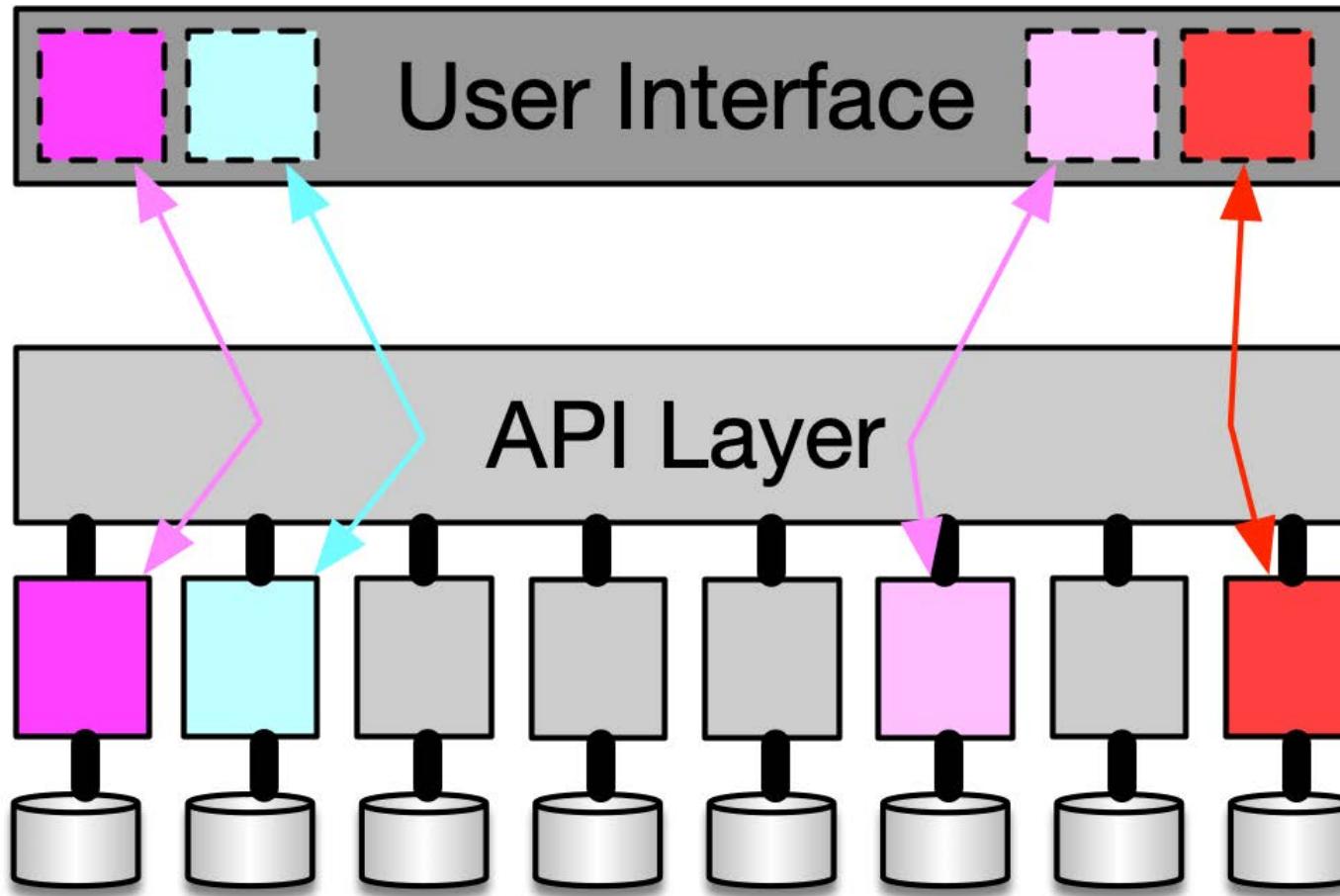


Single Static Quantum



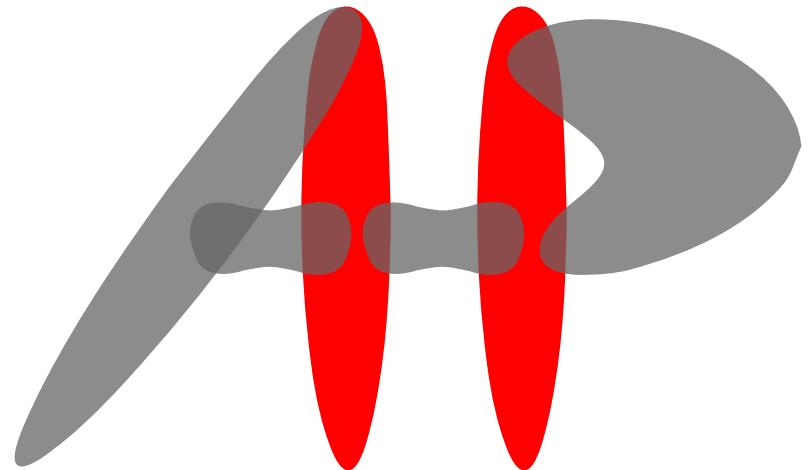
Multi-quanta (> 4)

Micro-front Ends



<https://martinfowler.com/articles/micro-frontends.html>

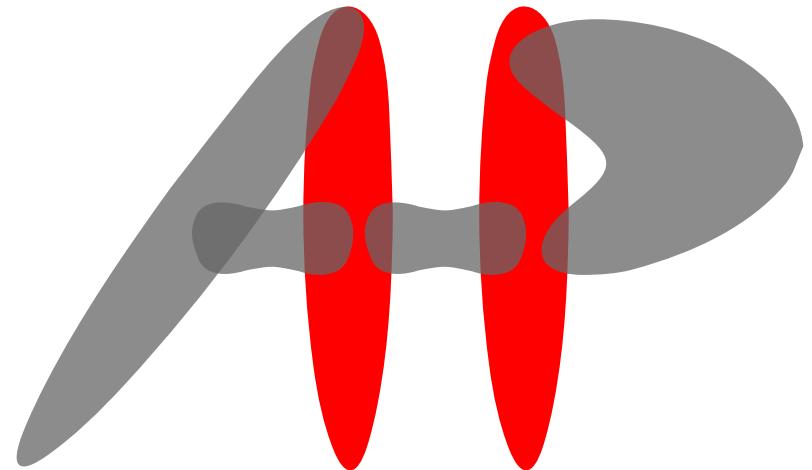
architectural quantum 2021



Represents how static dependencies resolve within the architecture via contracts. These dependencies include operating system, frameworks and/or libraries delivered via transitive dependency management, and any other operational requirement to allow the quantum to operate.

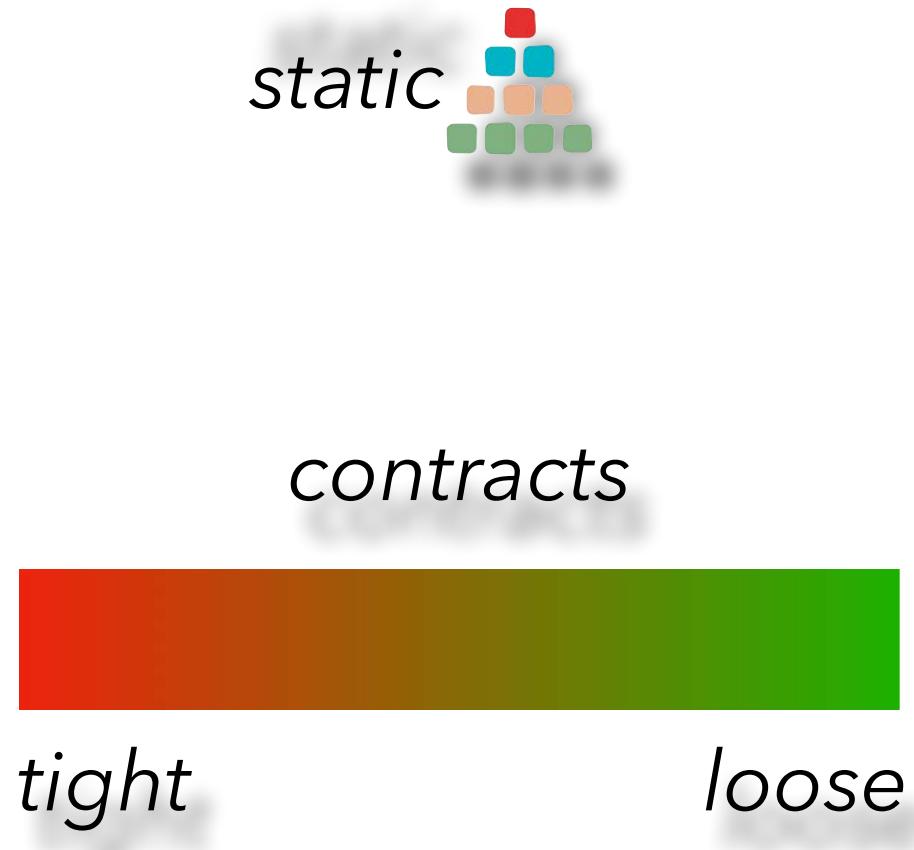
an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling.

architectural quantum 2021



an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling

quantum coupling

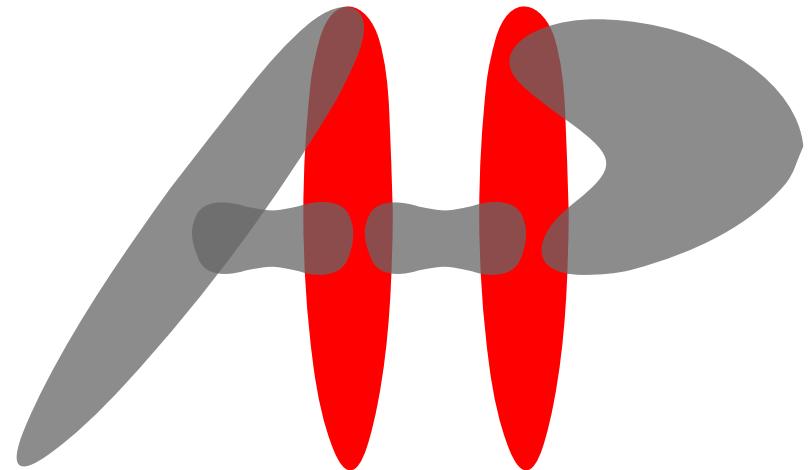


how quanta communicate

impact on operational
(and other) architecture characteristics

useful for hybrid architecture design,
architecture migration, integration, etc.

architectural quantum 2021



*architectural characteristics
live at the quantum level*

an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling.

architectural quantum

Your Architectural Kata is...

Going Green

A large electronics store wants to get into the electronics recycling business and needs a new system to support it. Customers can send in their small personal electronic equipment (or use local kiosks at the mall) and possibly get money for their used equipment if it is in working condition.

Requirements:

- Customers can get a quote for used personal electronic equipment (phones, cameras, etc.) either through the web or a kiosk at a mall.
- Customers will receive a box in the mail, send in their electronic, and if it is in good working order receive a check.
- Once the equipment is received, it is assessed (inspected) to determine if it can be either recycled (destroyed safely) or sold (eBay, etc.).
- The company anticipates adding 5-10 new types of electronic that they will accept each month.
- Each type of electronic has its own set of rules for quoting and assessment.
- This is a highly competitive business and is a new line of business for us

Users: Hundreds, hopefully thousands to millions

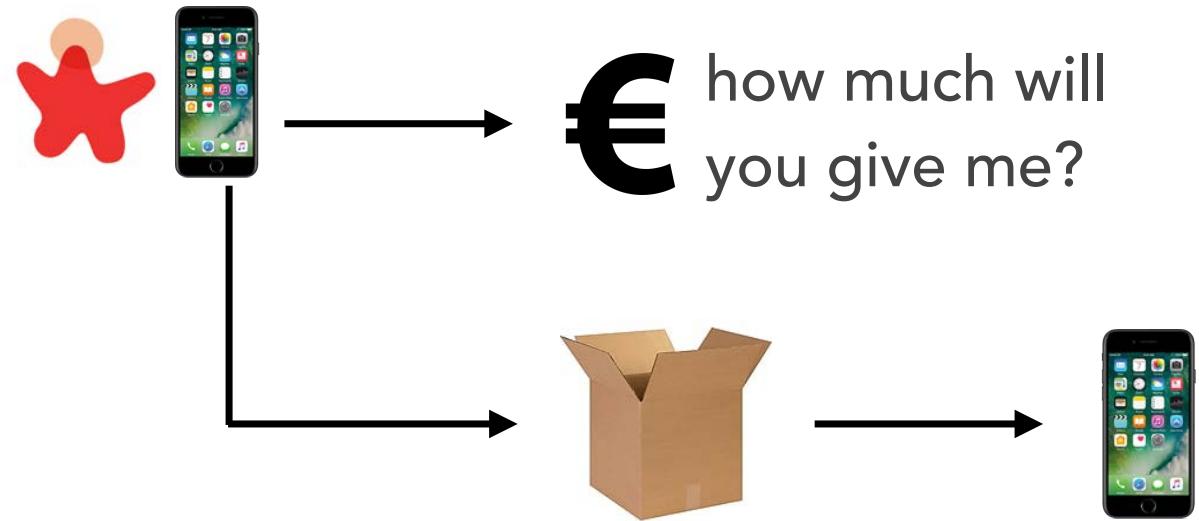
architectural quantum

electronics recycling



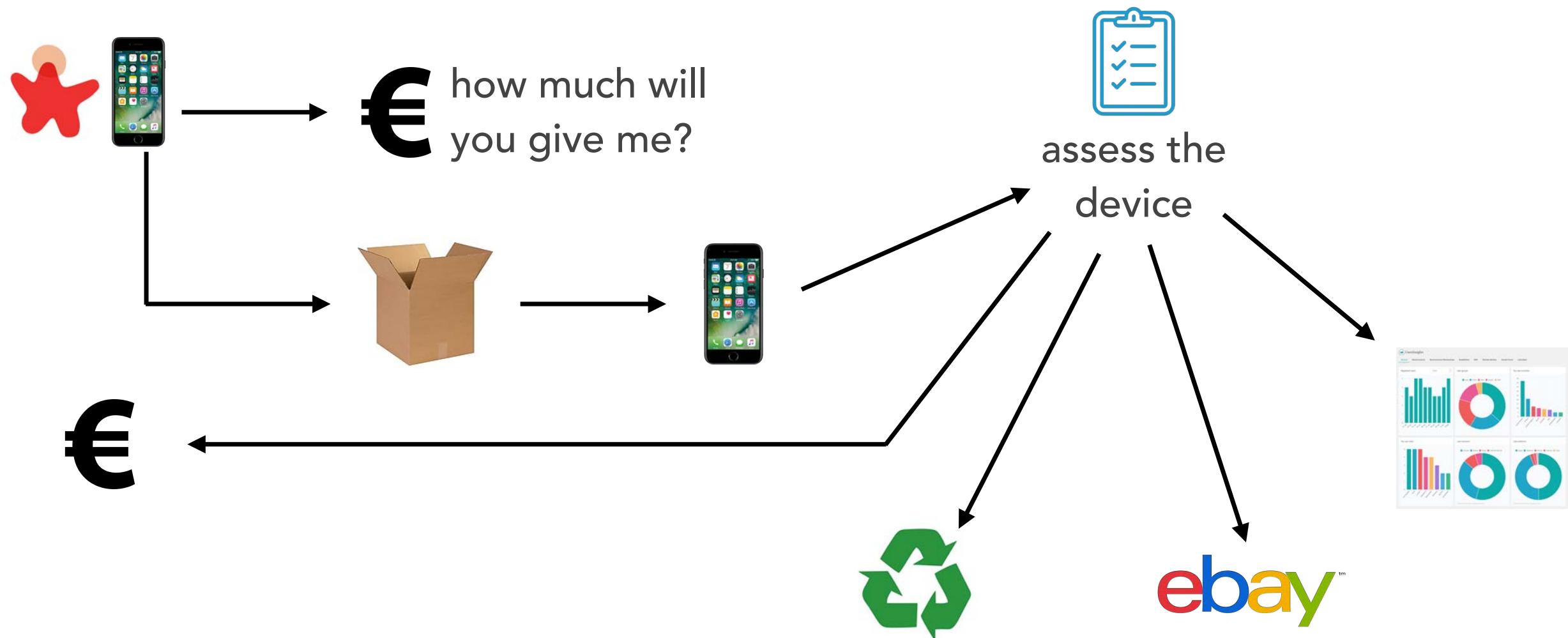
architectural quantum

electronics recycling



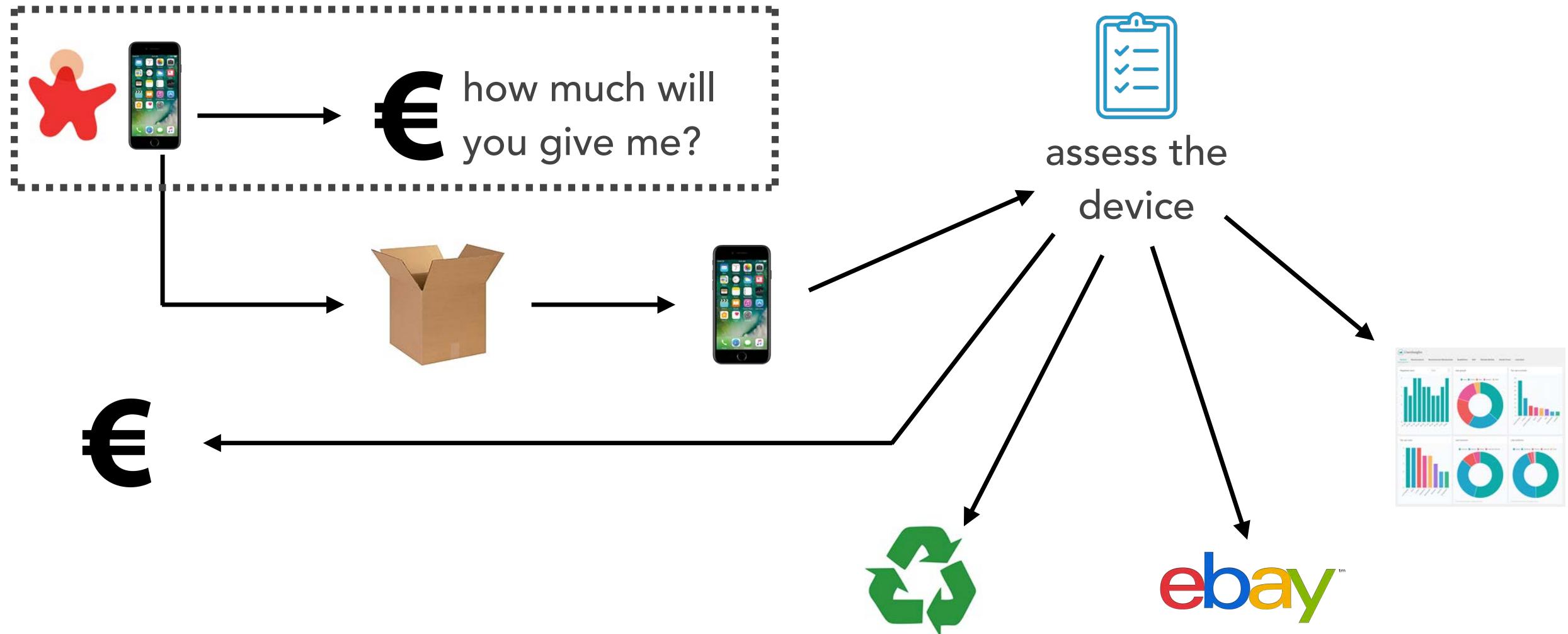
architectural quantum

electronics recycling



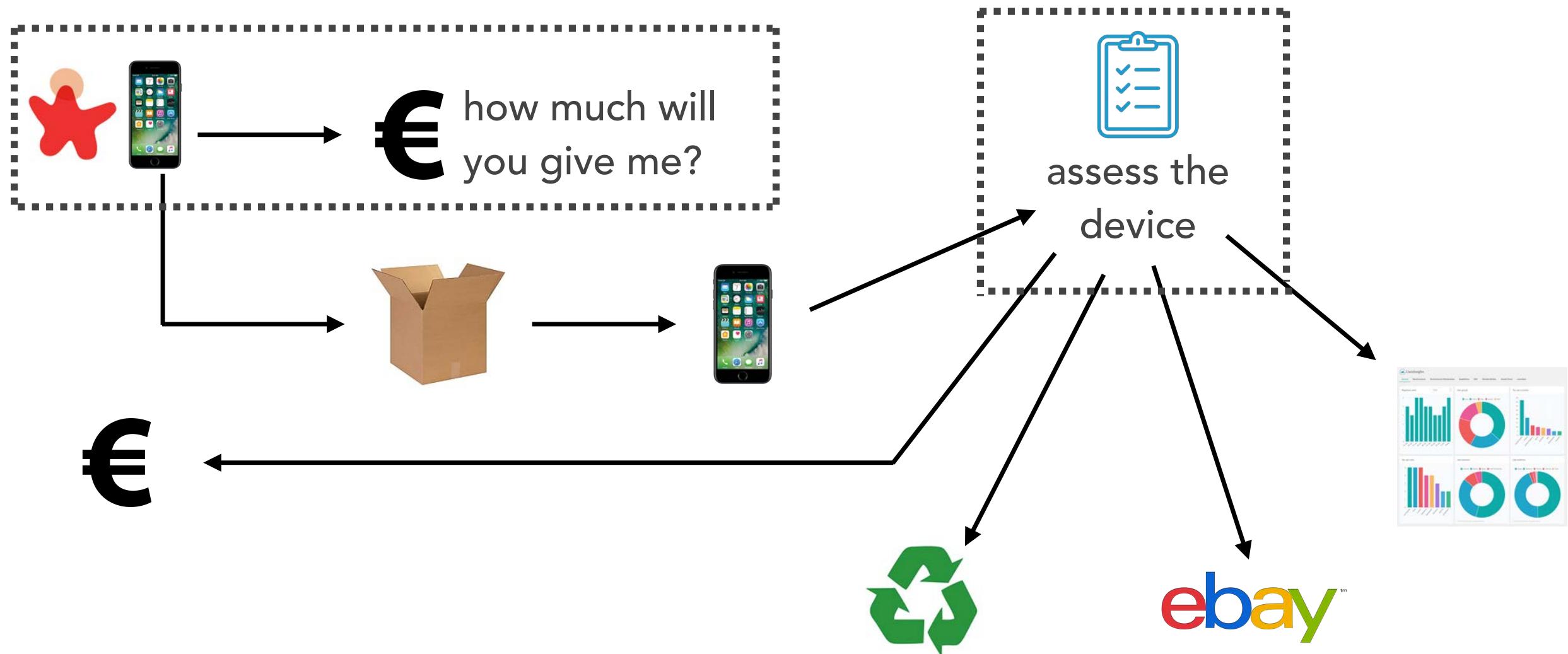
architectural quantum

electronics recycling



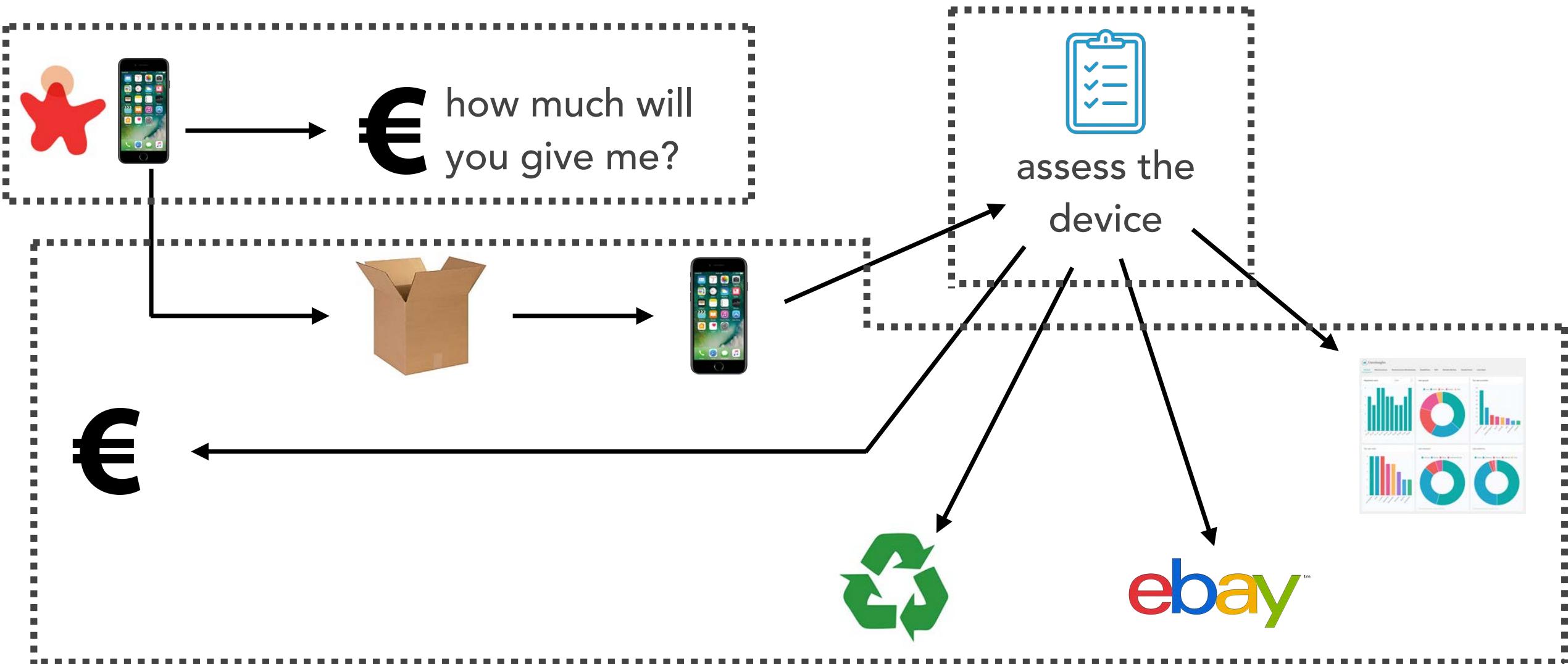
architectural quantum

electronics recycling



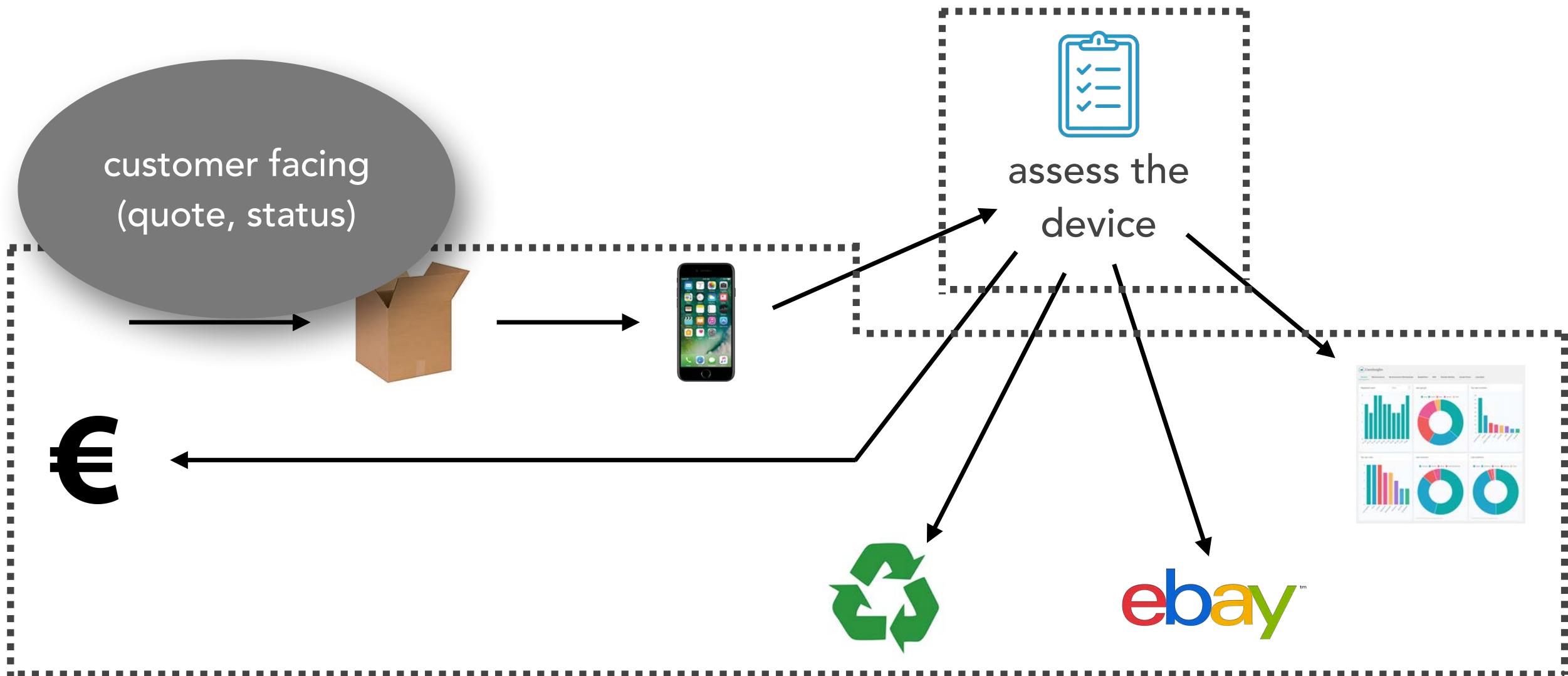
architectural quantum

electronics recycling



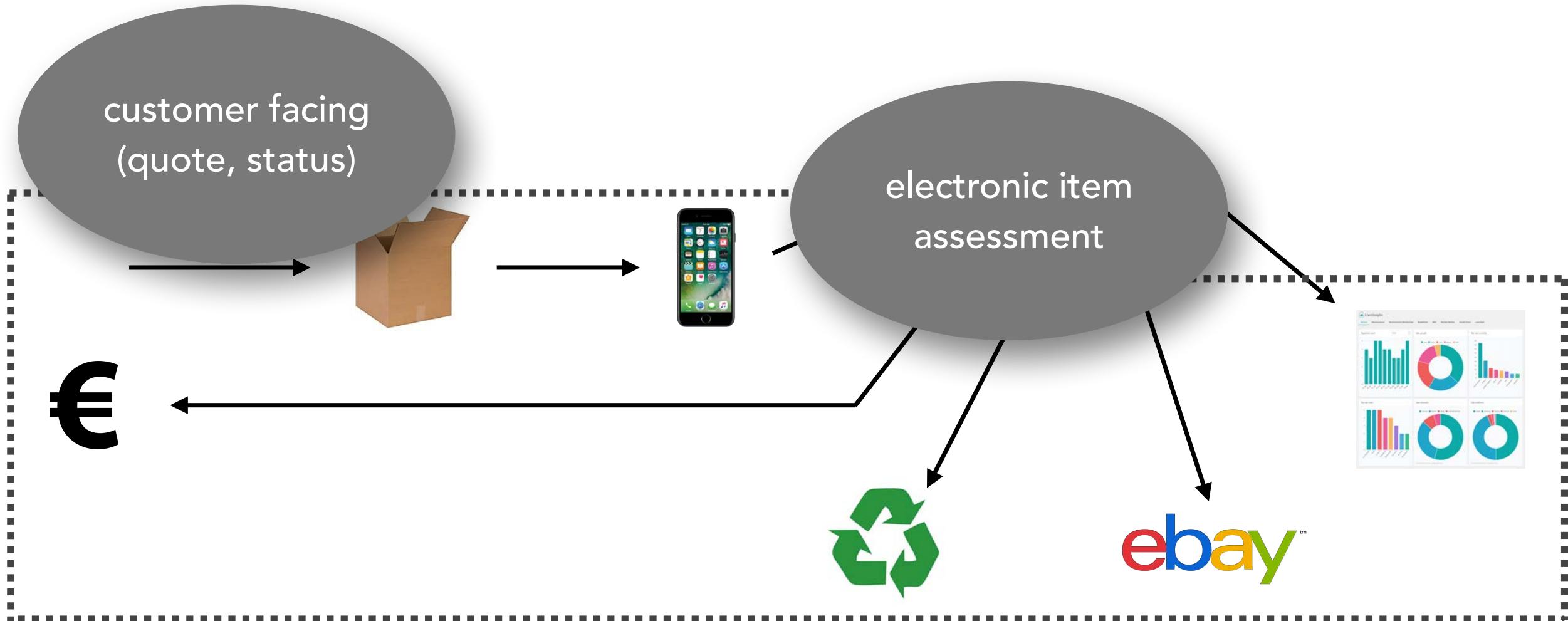
architectural quantum

electronics recycling



architectural quantum

electronics recycling



architectural quantum

electronics recycling

customer facing
(quote, status)

electronic item
assessment

recycling, reporting,
and accounting

architectural quantum

electronics recycling

customer facing
(quote, status)

scalability
availability
agility

electronic item
assessment

recycling, reporting,
and accounting

architectural quantum

electronics recycling

customer facing
(quote, status)

scalability
availability
agility

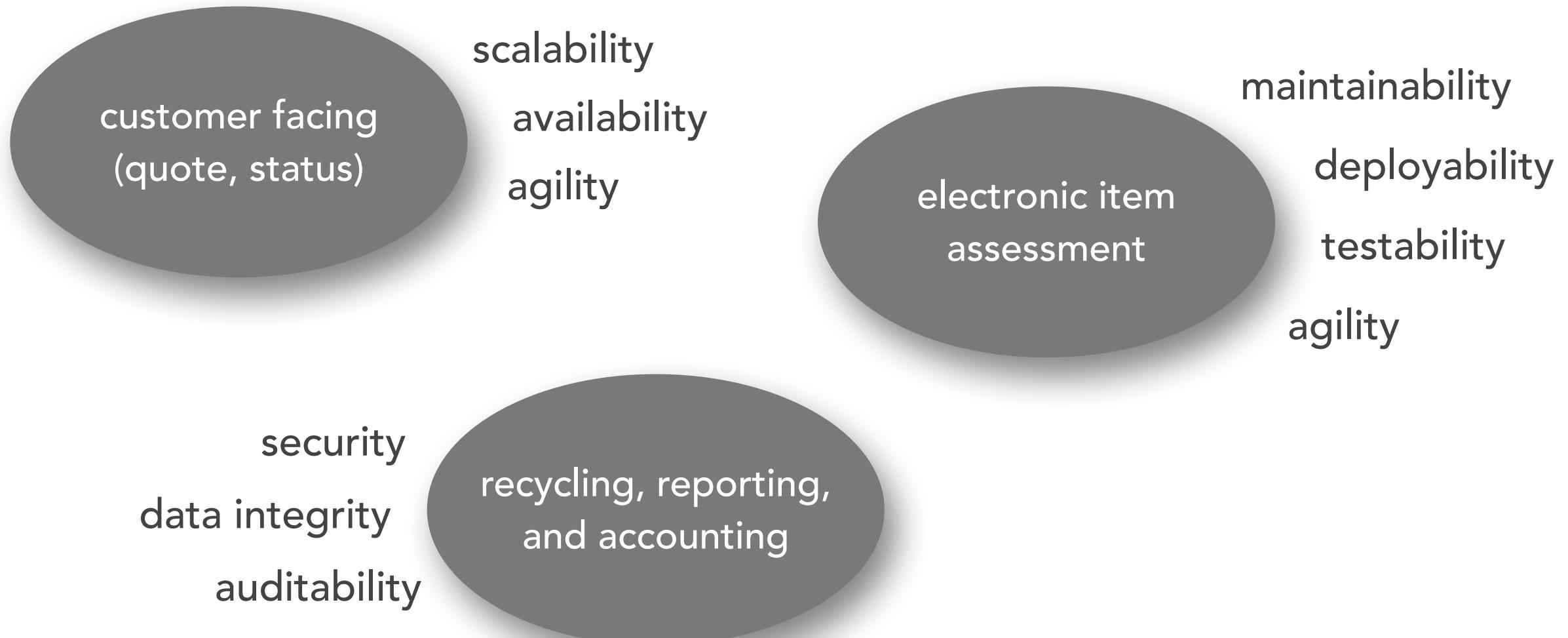
recycling, reporting,
and accounting

electronic item
assessment

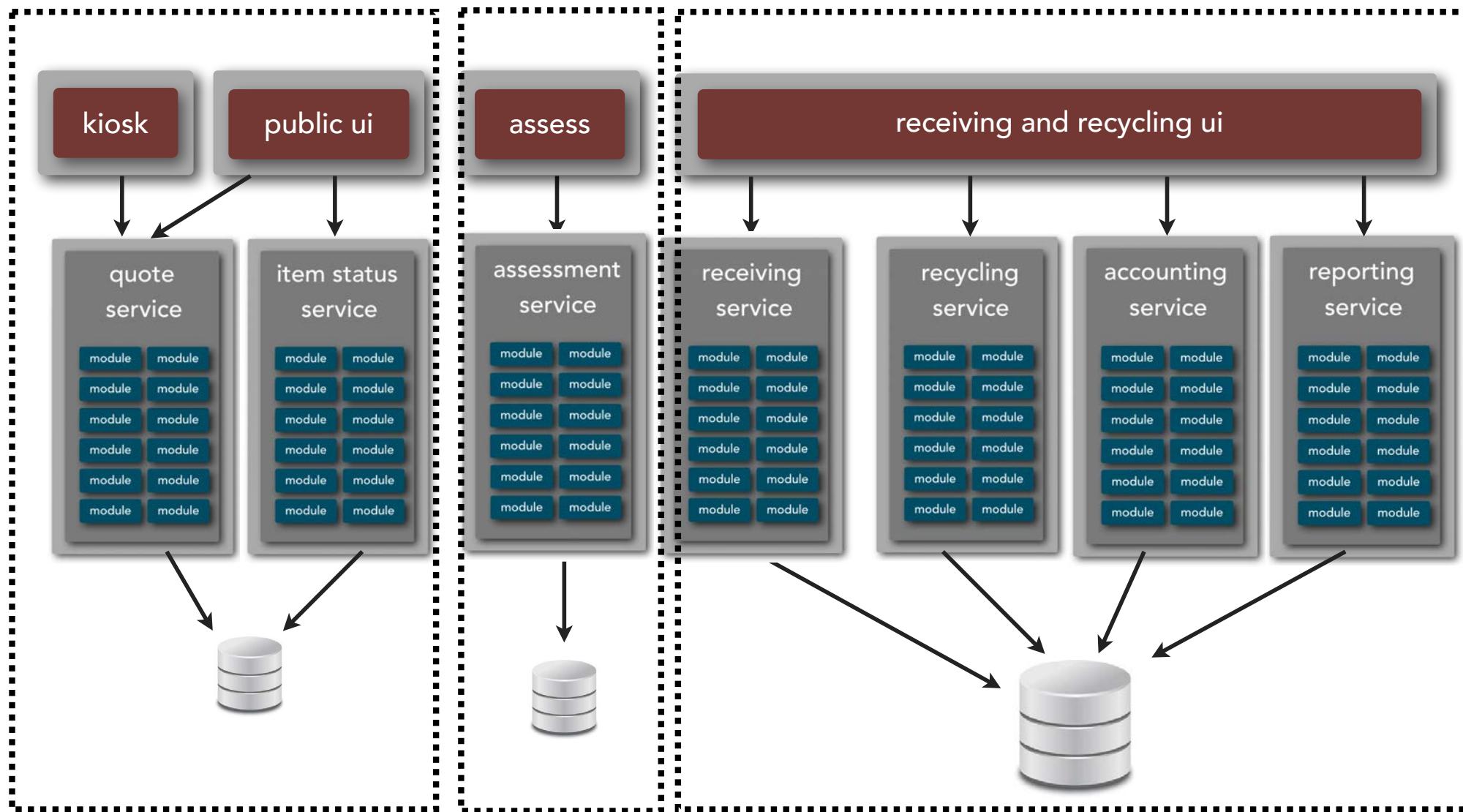
maintainability
deployability
testability
agility

architectural quantum

electronics recycling



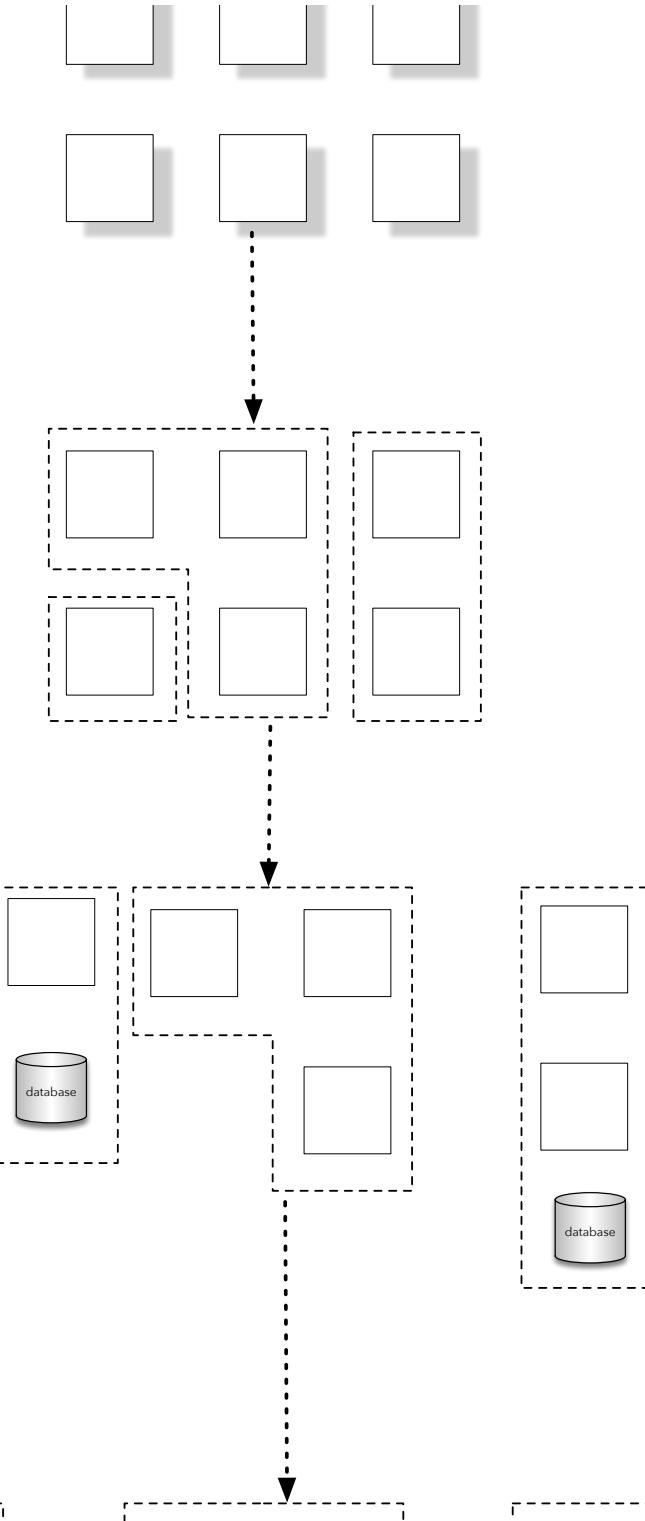
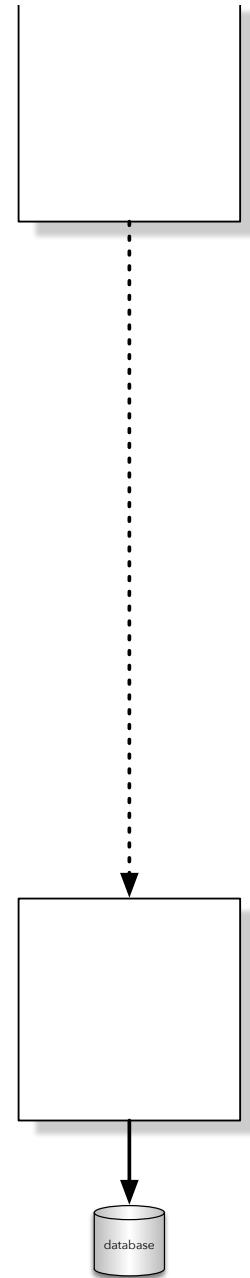
architectural quantum



monolith | distributed ?

2. Determine persistence

Monolithic
architecture



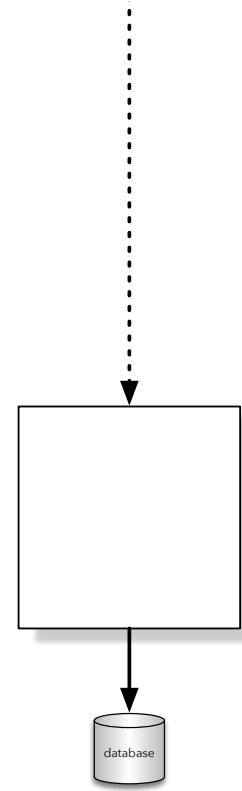
2. Choose quanta

3. Determine persistence

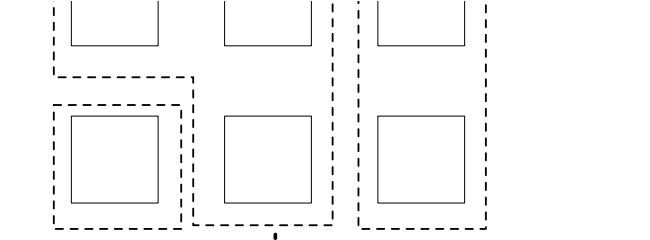
Distributed
architecture

monolith | distributed ?

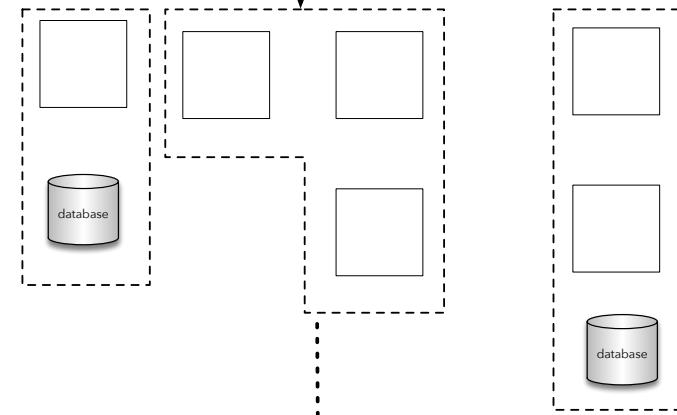
2. Determine persistence



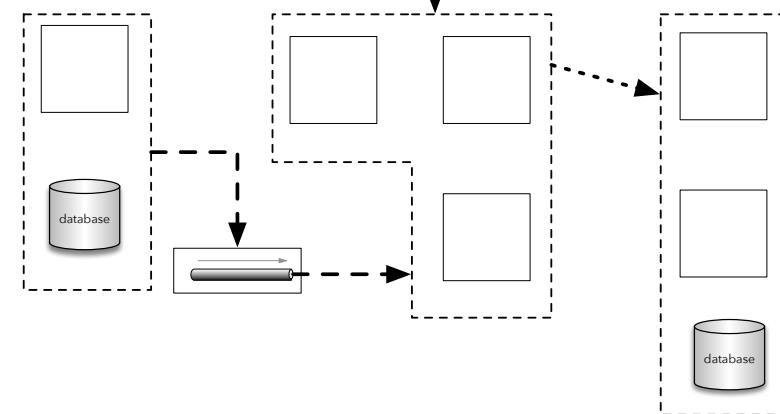
2. Choose quanta



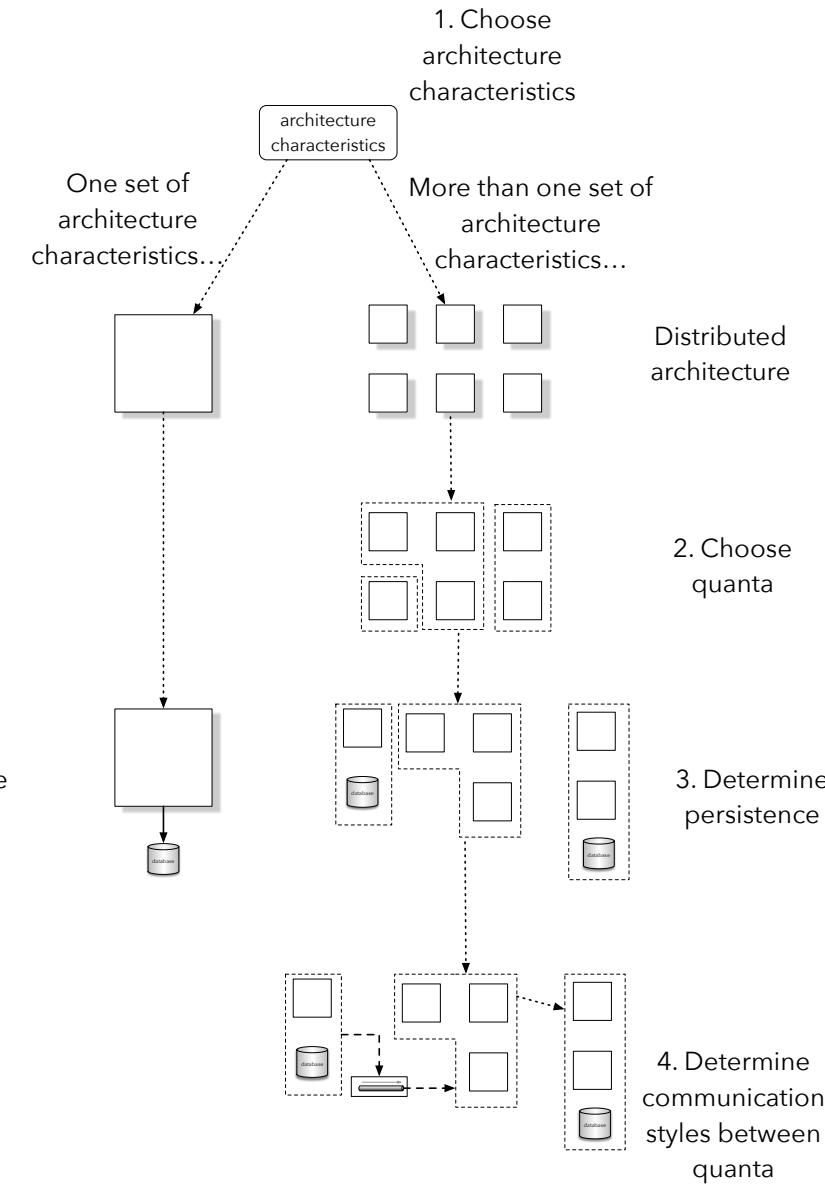
3. Determine persistence



4. Determine communication styles between quanta

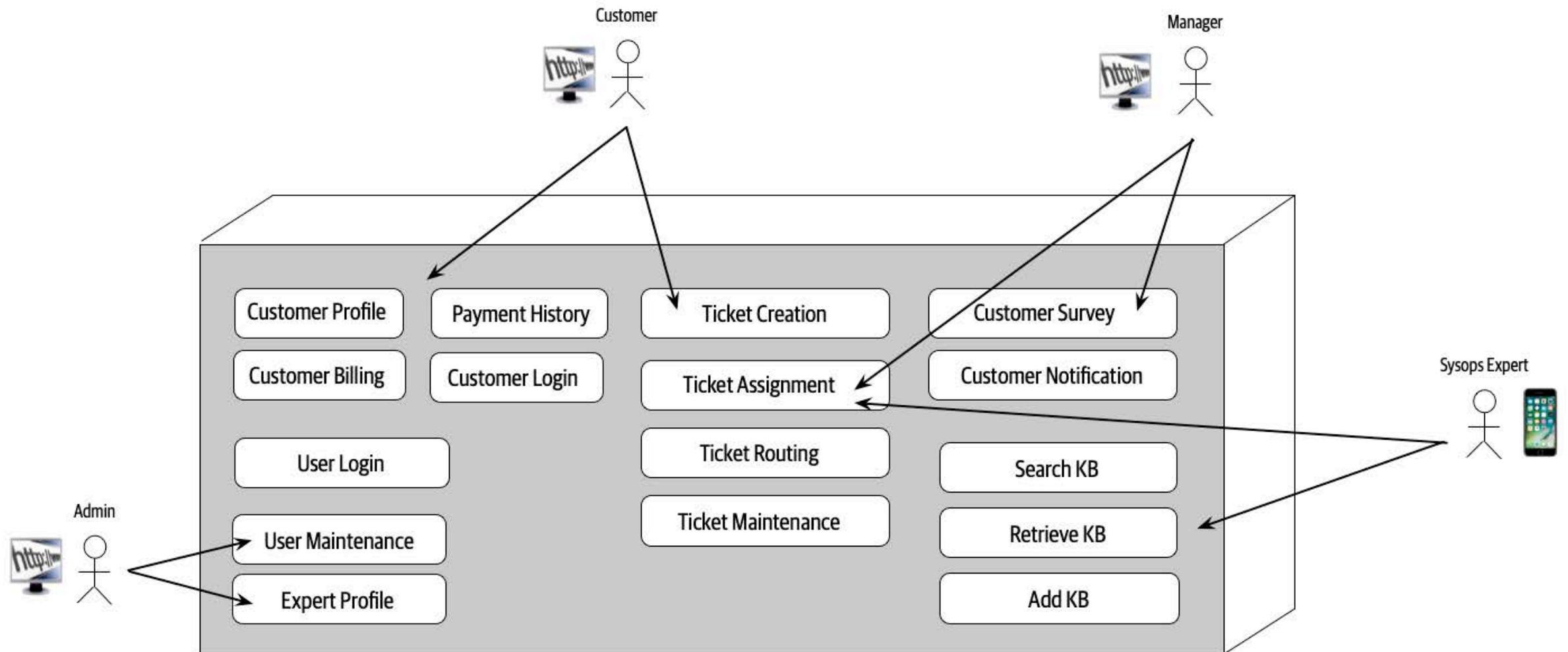


monolith | distributed?



Kata Exercise - Components and Quanta

Based on the components and data below, identify the architecture quanta and components included in each quantum

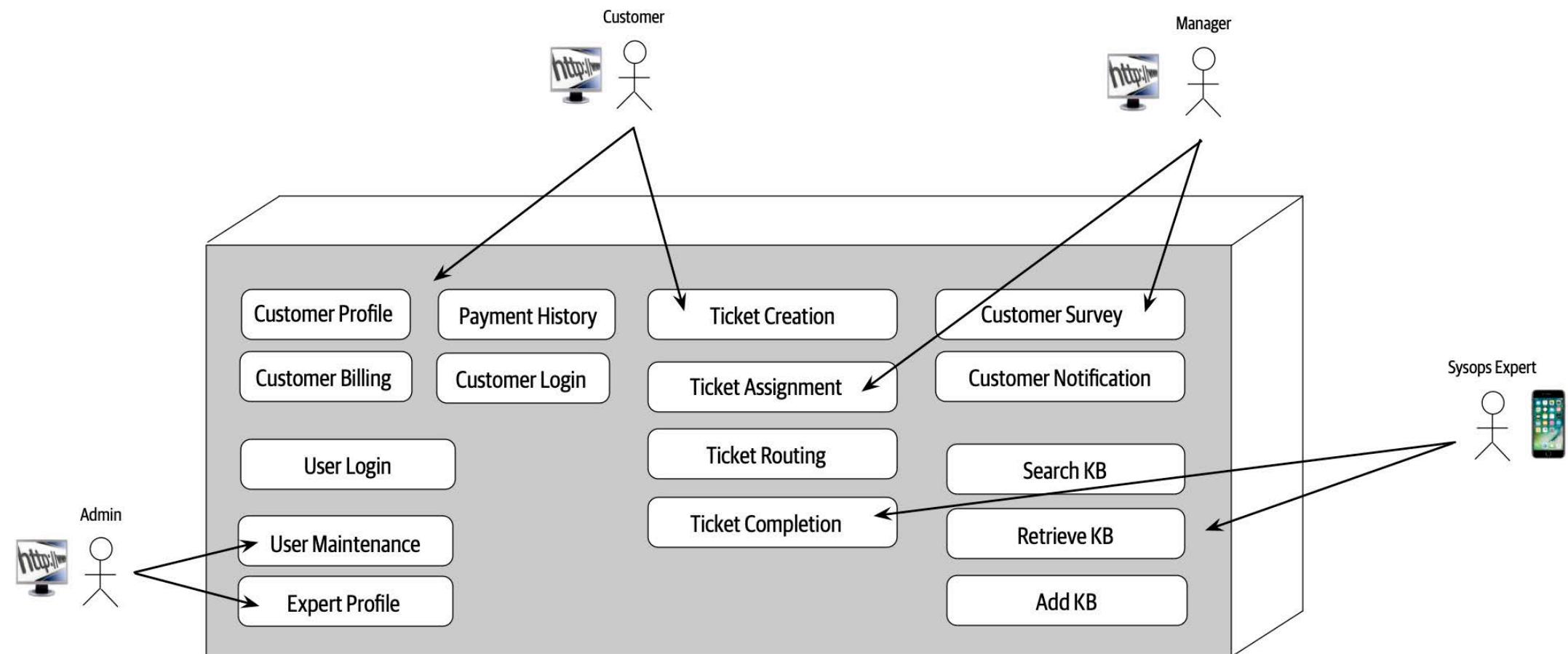




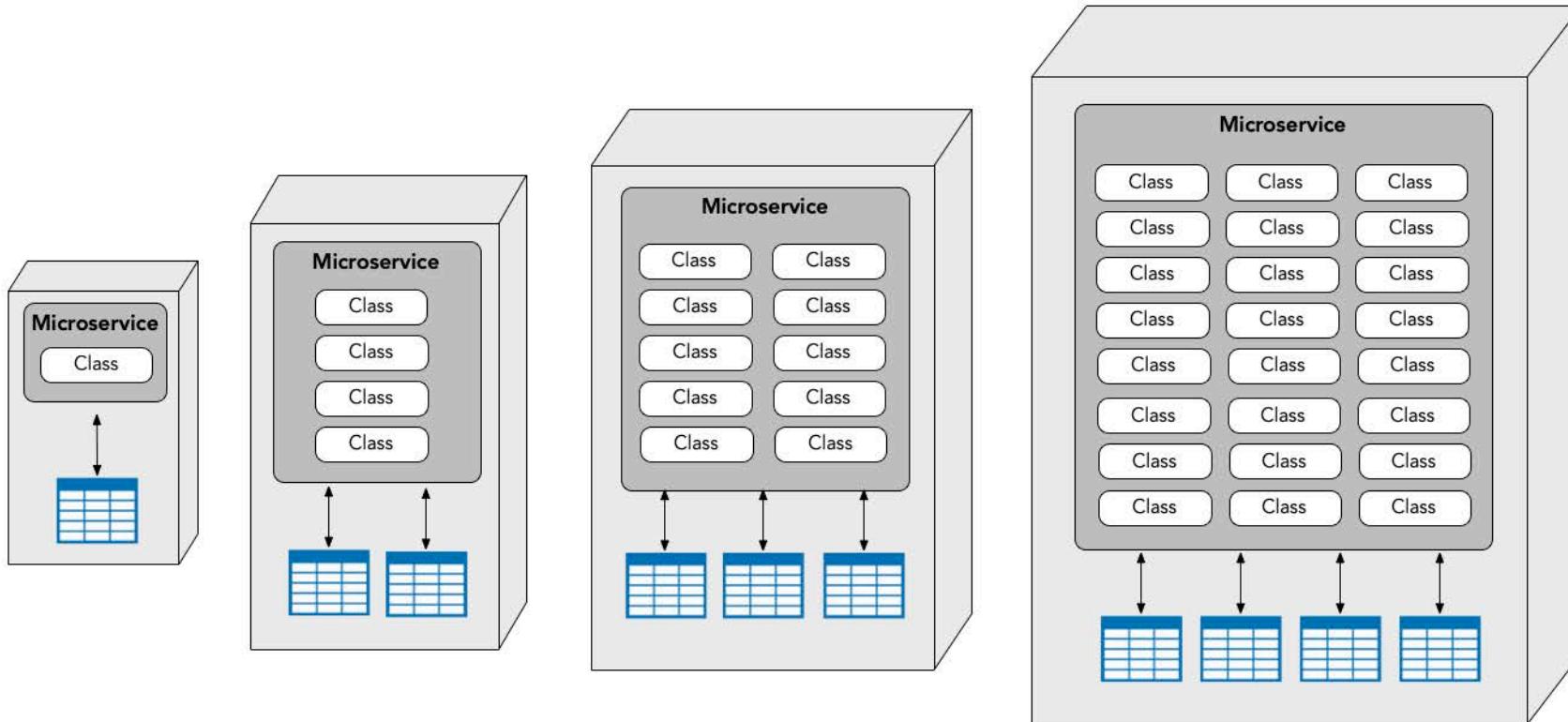
Granularity vs Modularity

Kata Exercise - Service Identification

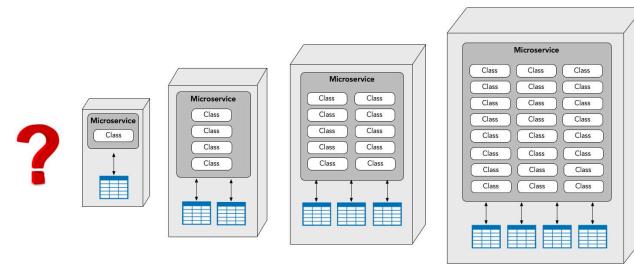
Management has given the go-ahead for moving to a distributed architecture. Based on the components illustrated below and what you know about service granularity, your job now is to identify the initial service candidates.



what is the right level of granularity for a service?



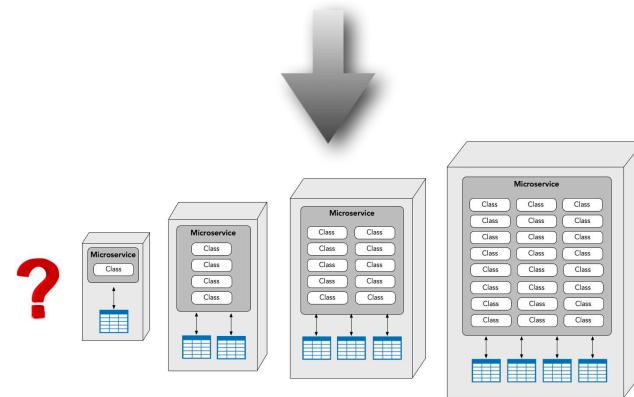
service granularity



service granularity

granularity disintegrators

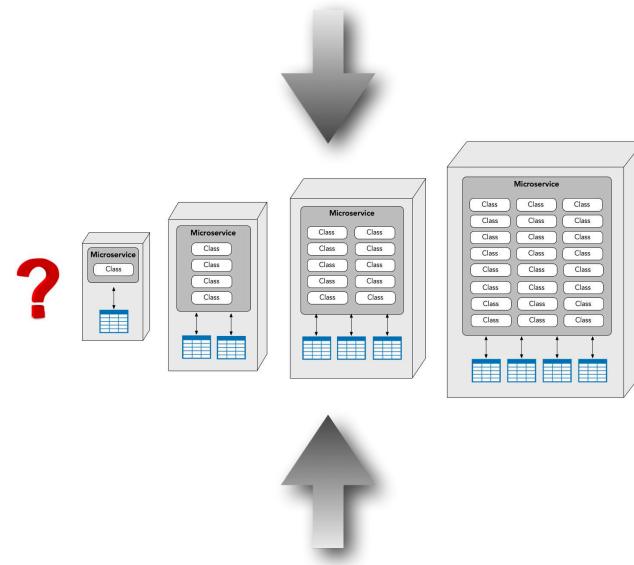
“when should I consider breaking apart a service?”



service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



granularity integrators

“when should I consider putting services back together?”

service granularity

granularity disintegrators

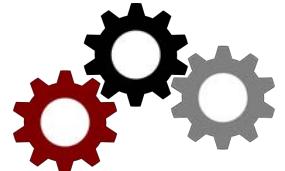
“when should I consider breaking apart a service?”



service granularity

granularity disintegrators

“when should I consider breaking apart a service?”

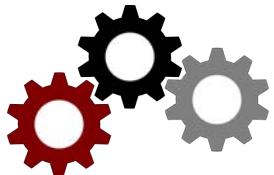


service
functionality

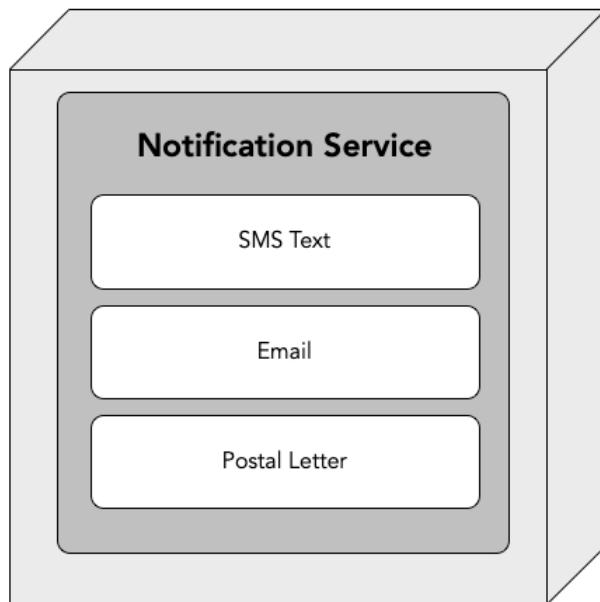
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



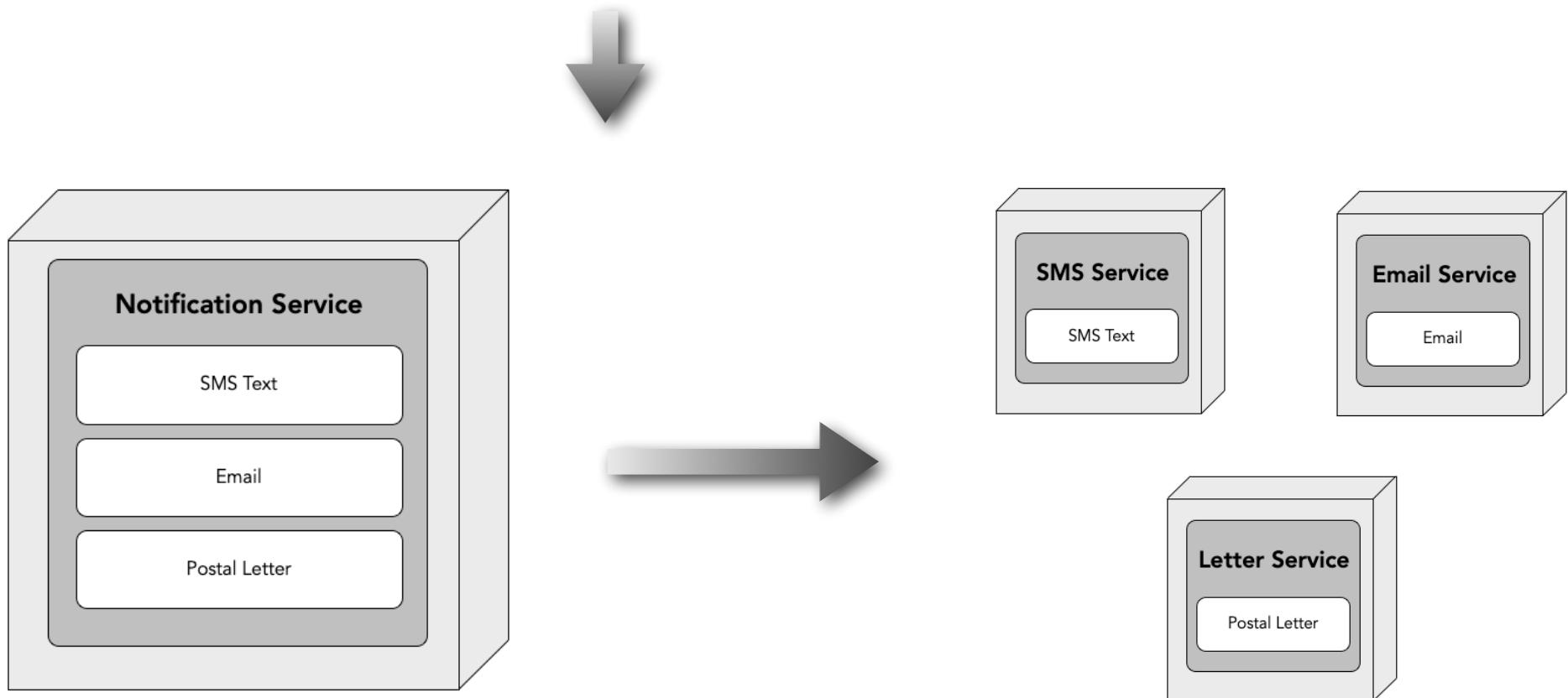
service
functionality



service granularity

granularity disintegrators

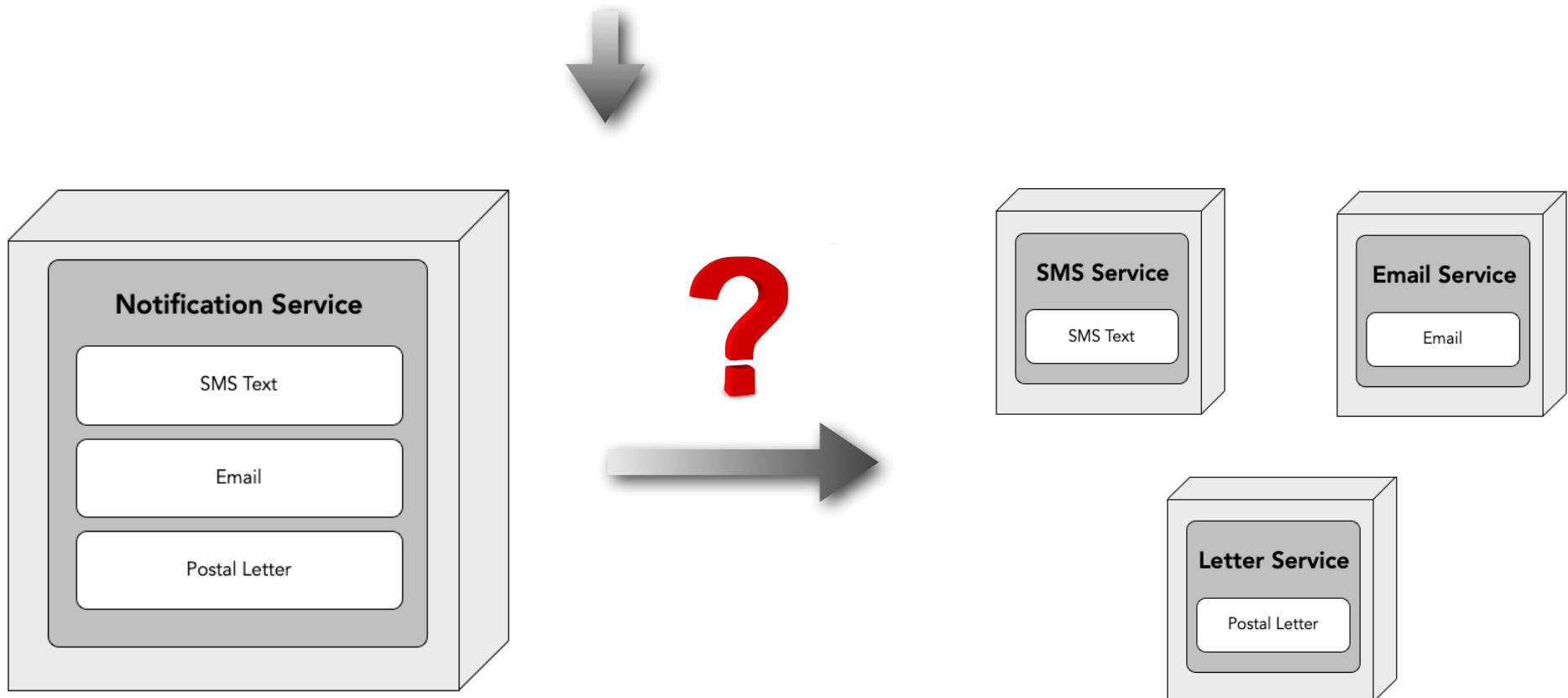
“when should I consider breaking apart a service?”



service granularity

granularity disintegrators

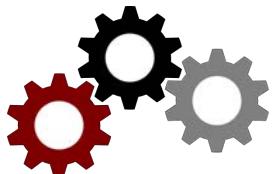
“when should I consider breaking apart a service?”



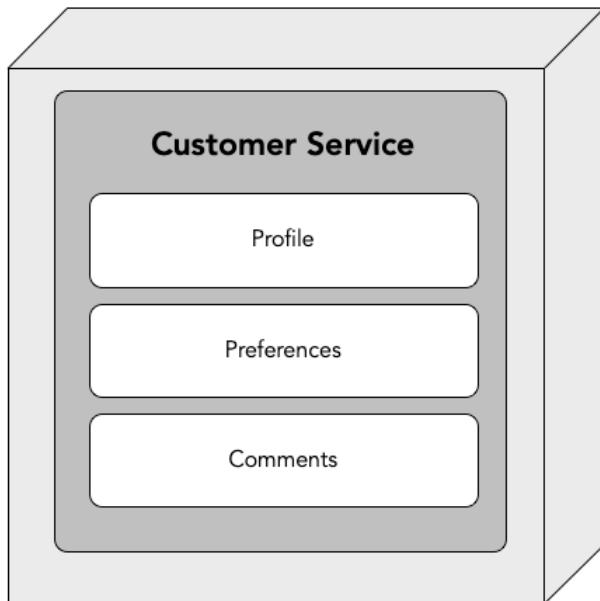
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



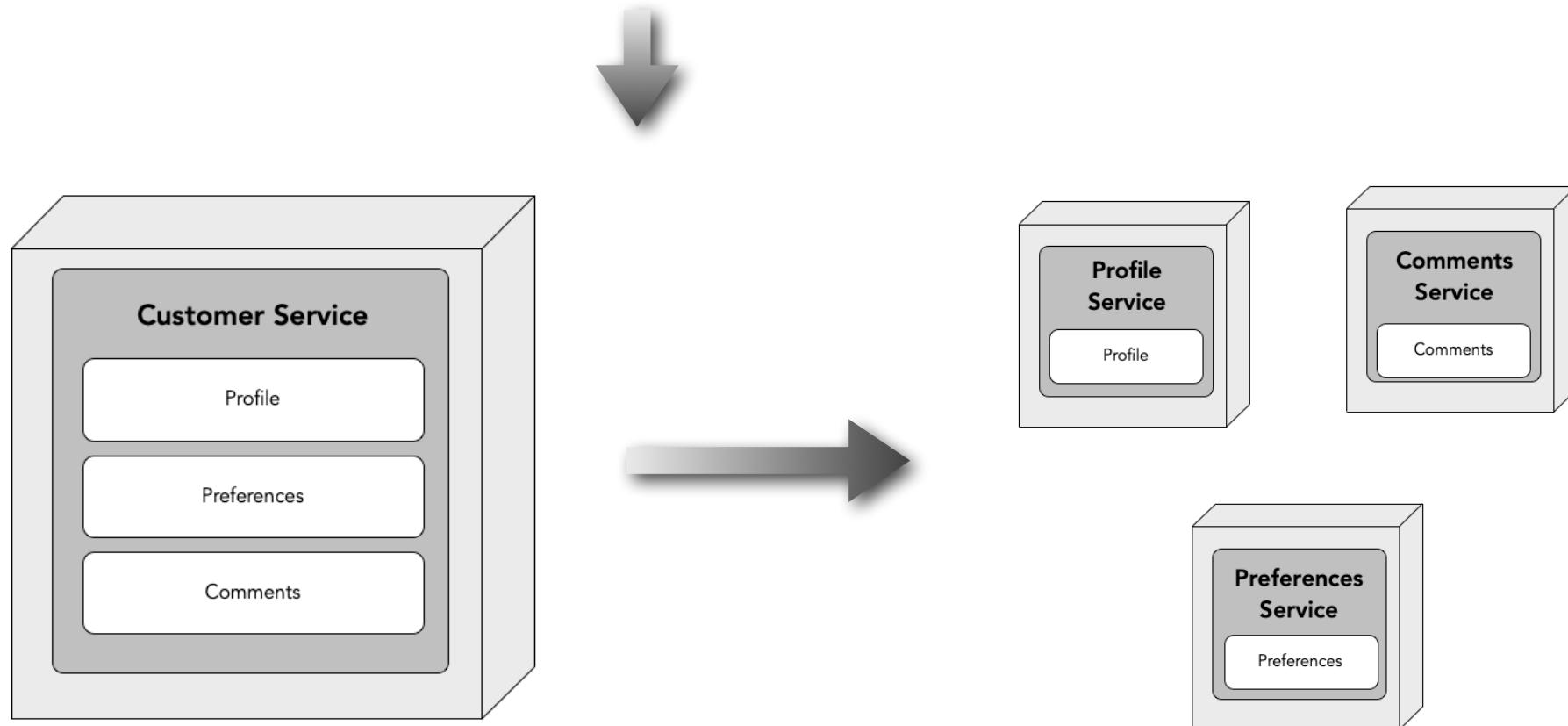
service
functionality



service granularity

granularity disintegrators

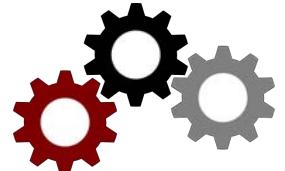
“when should I consider breaking apart a service?”



service granularity

granularity disintegrators

“when should I consider breaking apart a service?”

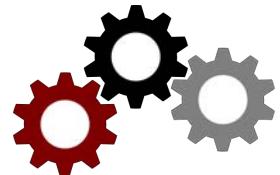


service
functionality

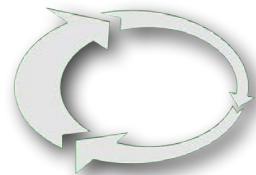
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



service
functionality

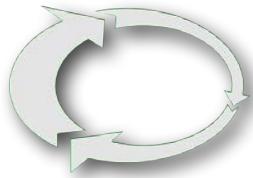


code
volatility

service granularity

granularity disintegrators

“when should I consider breaking apart a service?”

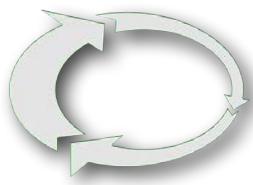


code
volatility

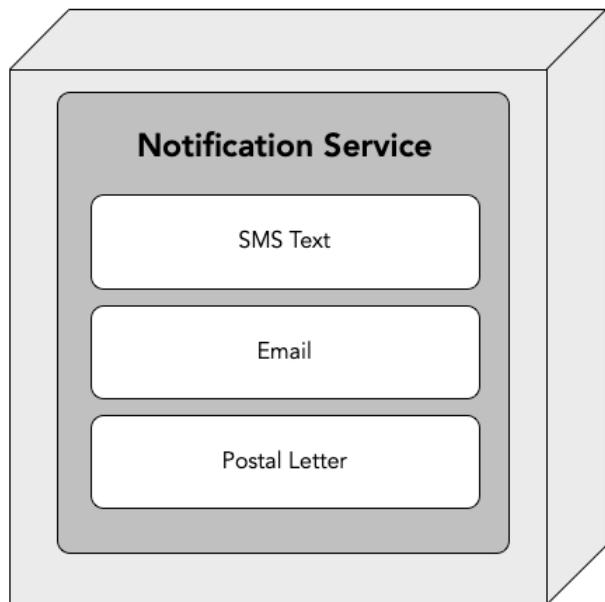
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



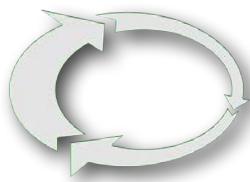
code
volatility



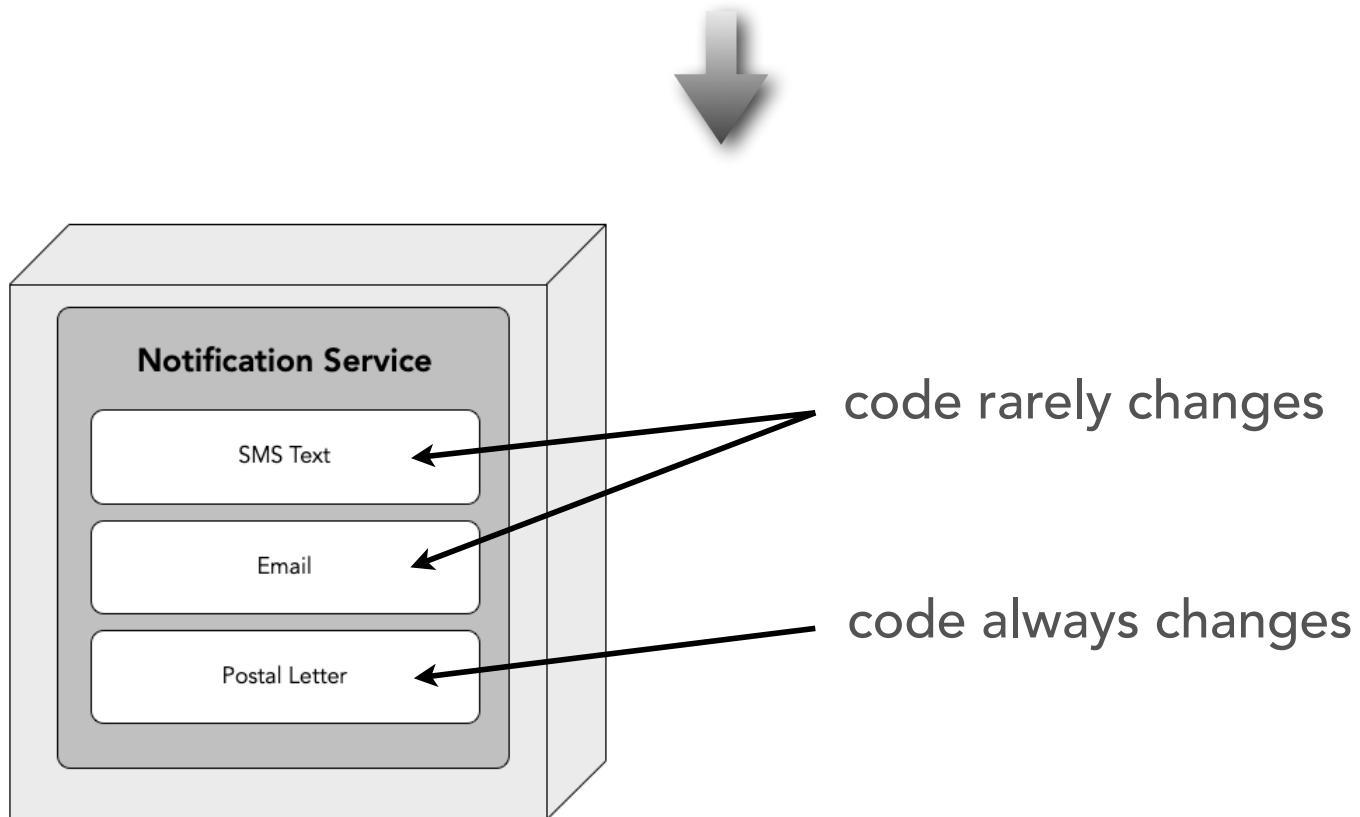
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



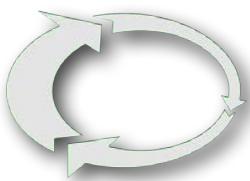
code
volatility



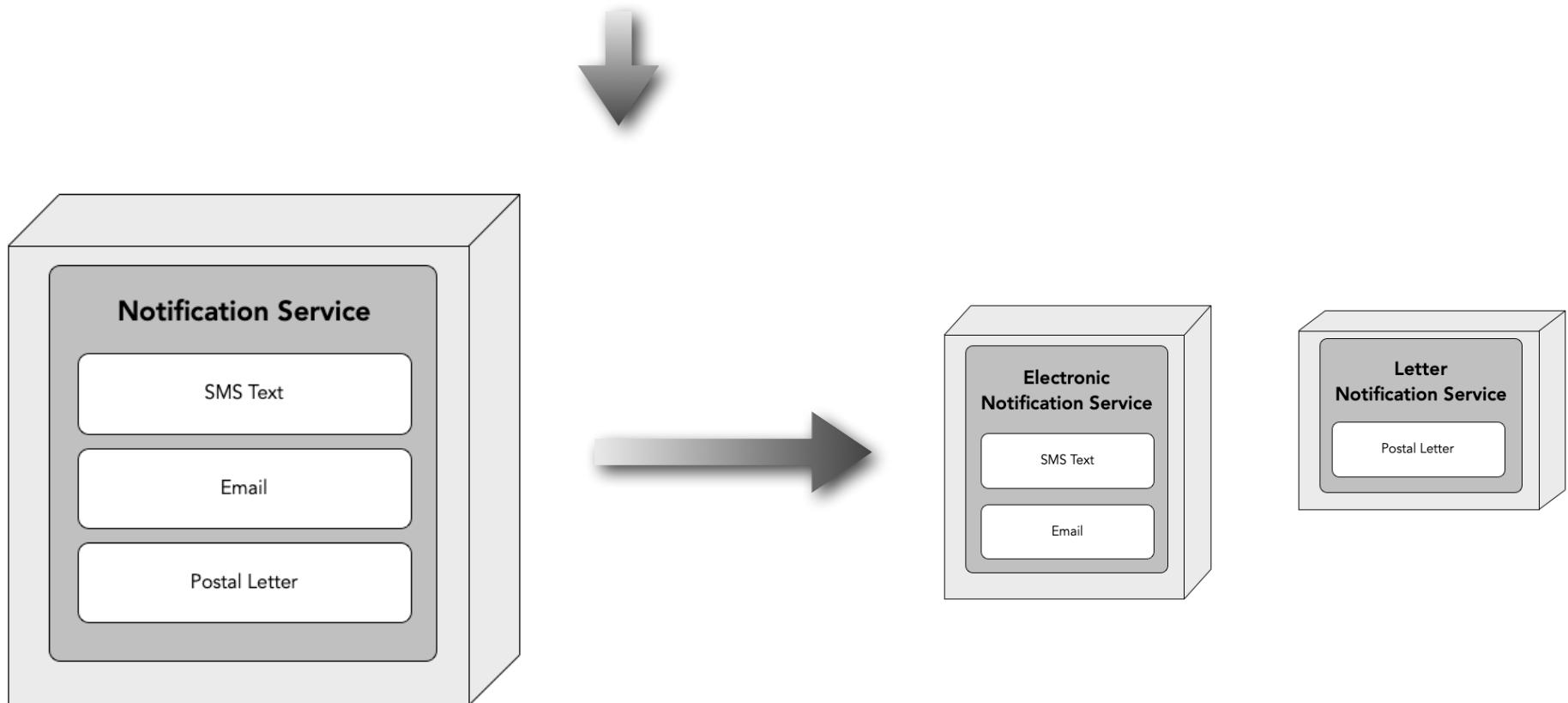
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



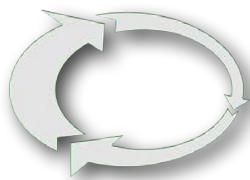
code
volatility



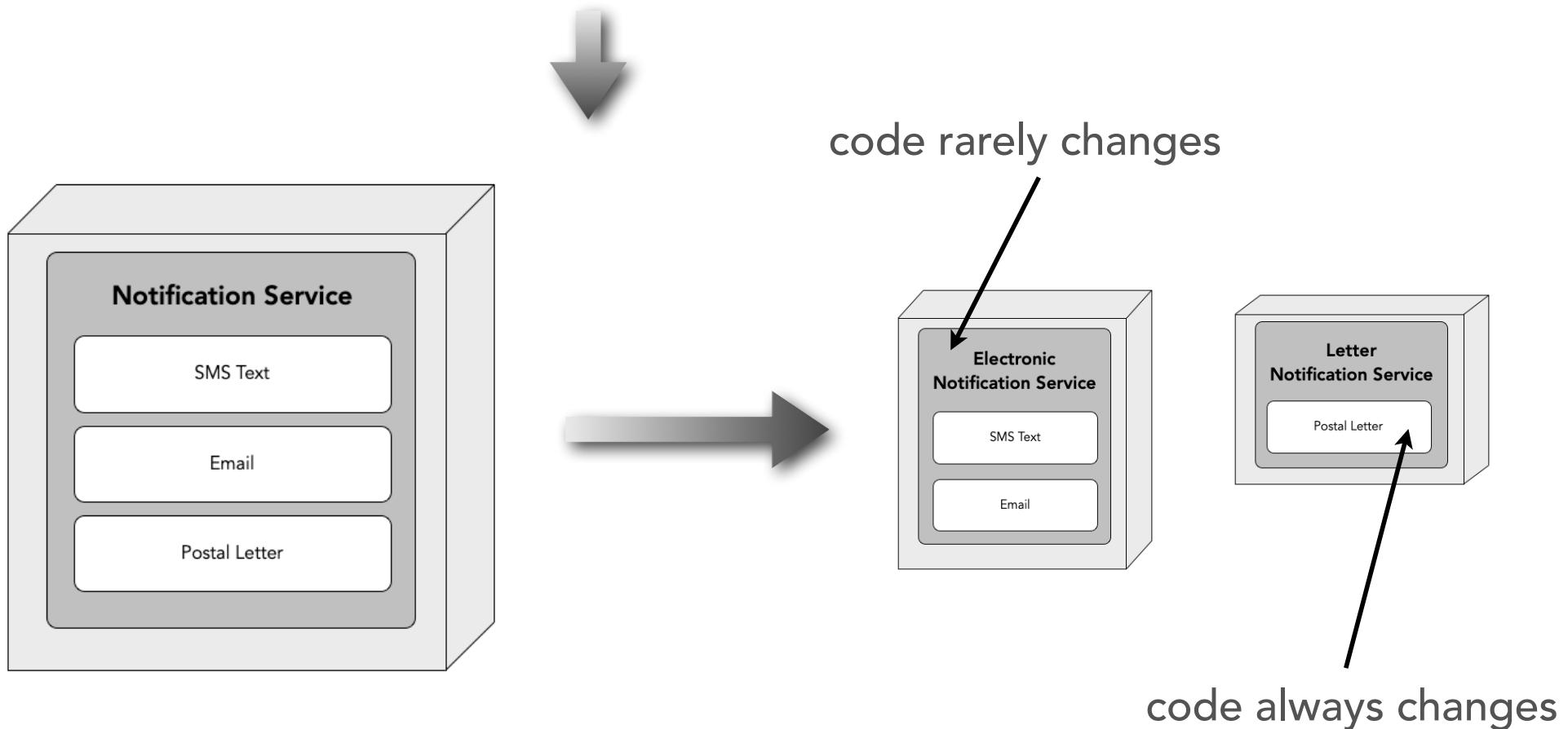
service granularity

granularity disintegrators

"when should I consider breaking apart a service?"



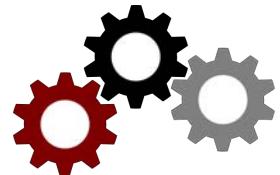
code
volatility



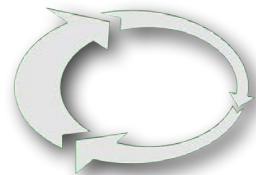
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



service
functionality

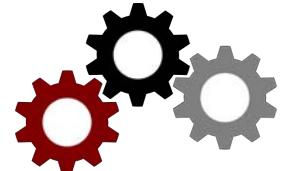


code
volatility

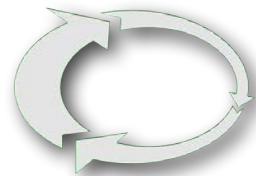
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



service
functionality



code
volatility



scalability and
throughput

service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



scalability and
throughput

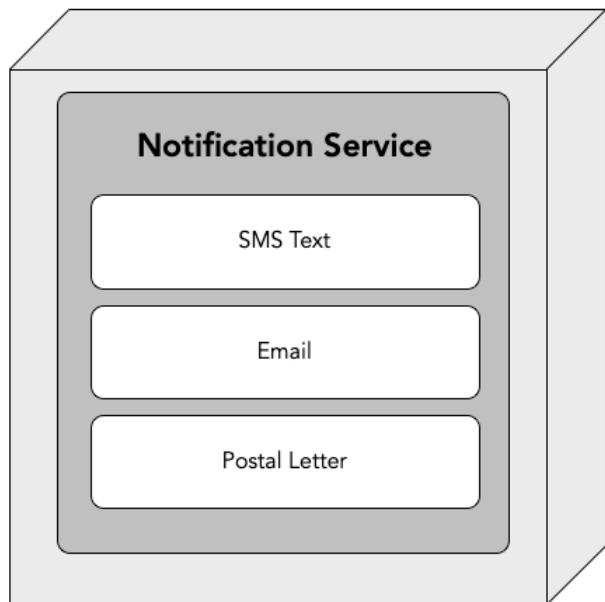
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



scalability and
throughput



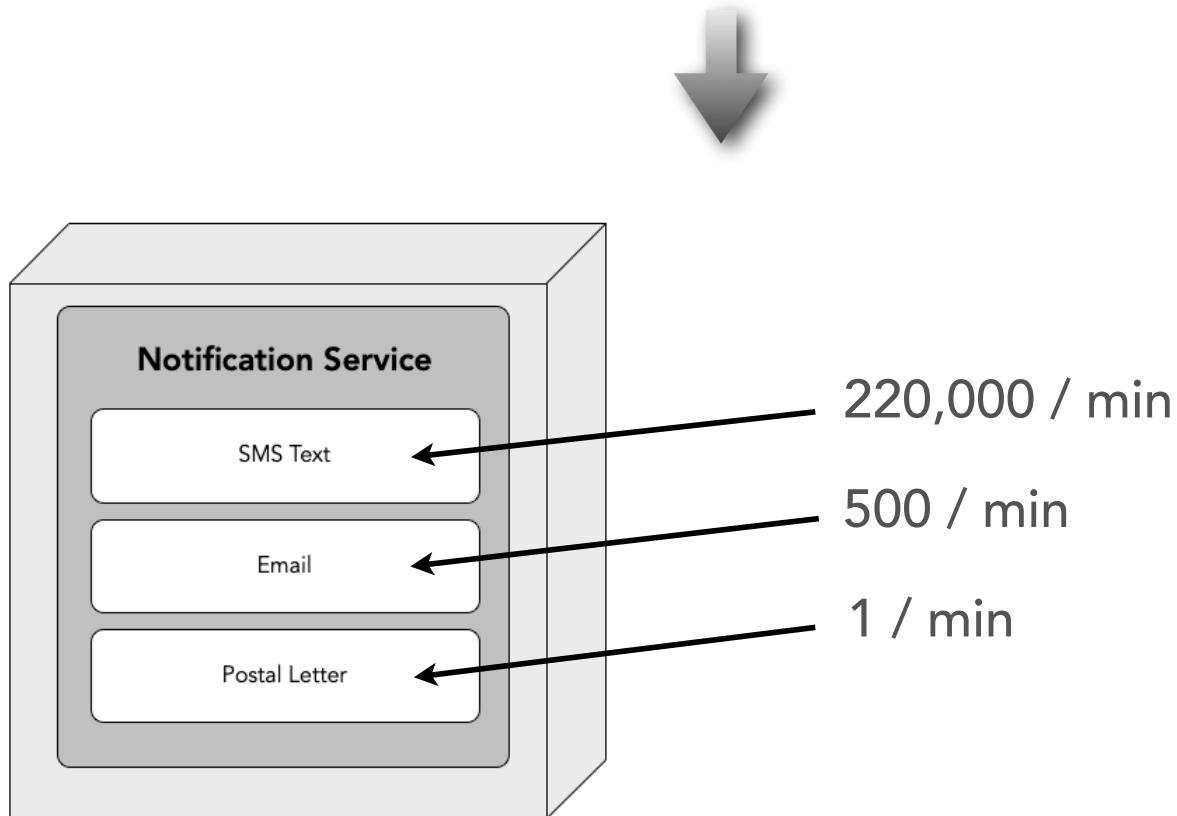
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



scalability and
throughput



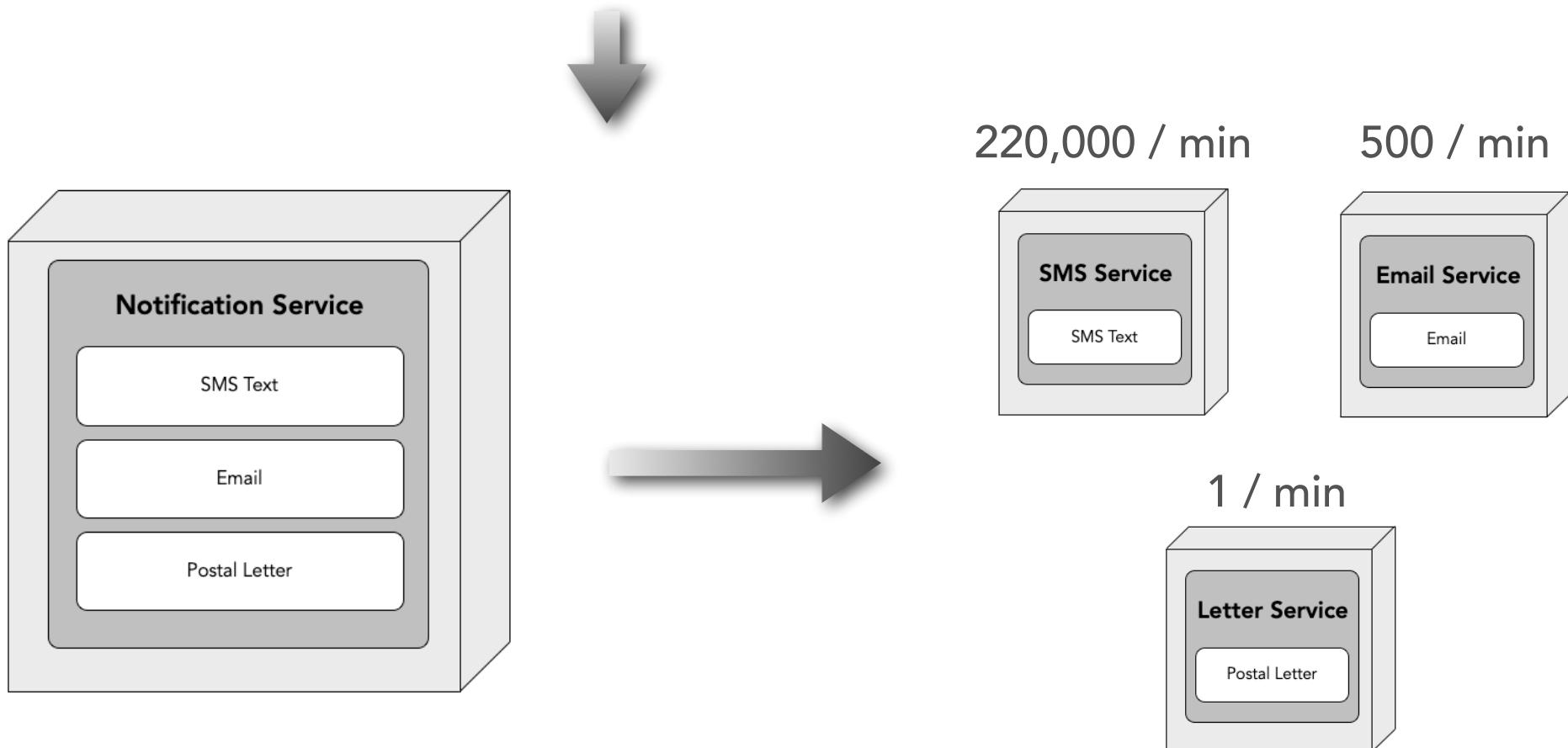
service granularity

granularity disintegrators

"when should I consider breaking apart a service?"



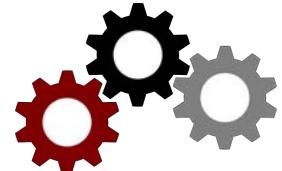
scalability and
throughput



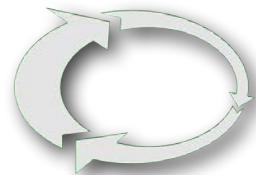
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



service
functionality



code
volatility

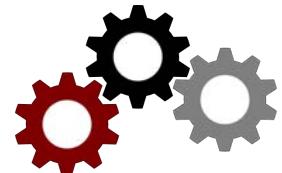


scalability and
throughput

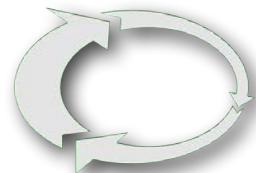
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



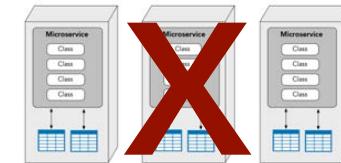
service
functionality



code
volatility



scalability and
throughput

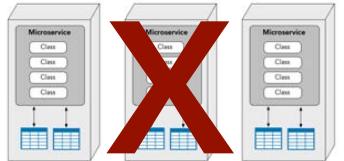


fault
tolerance

service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



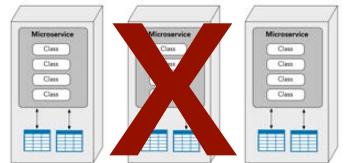
fault

tolerance

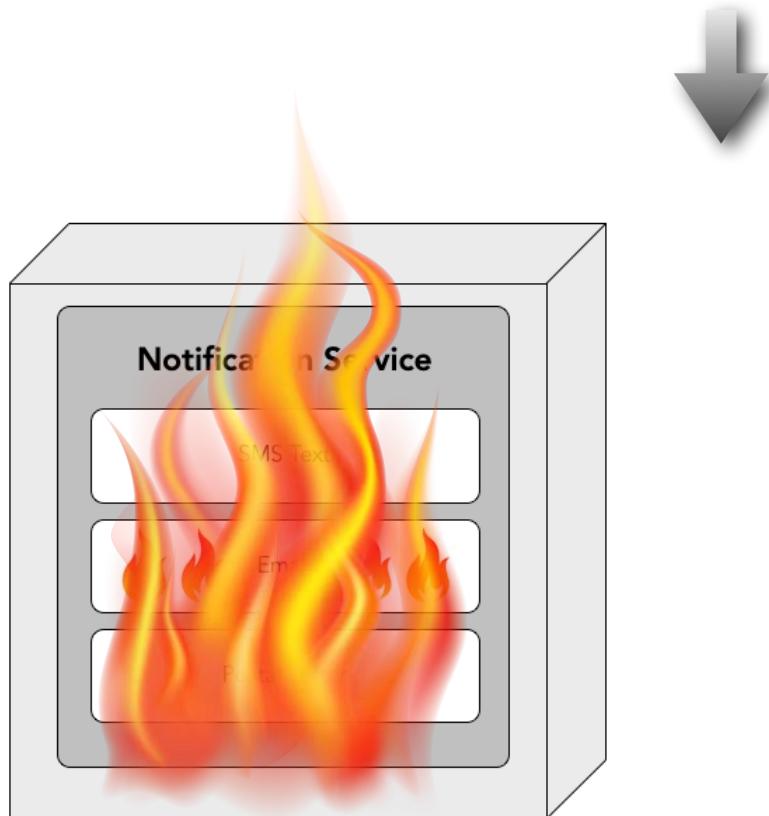
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



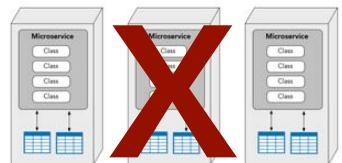
fault
tolerance



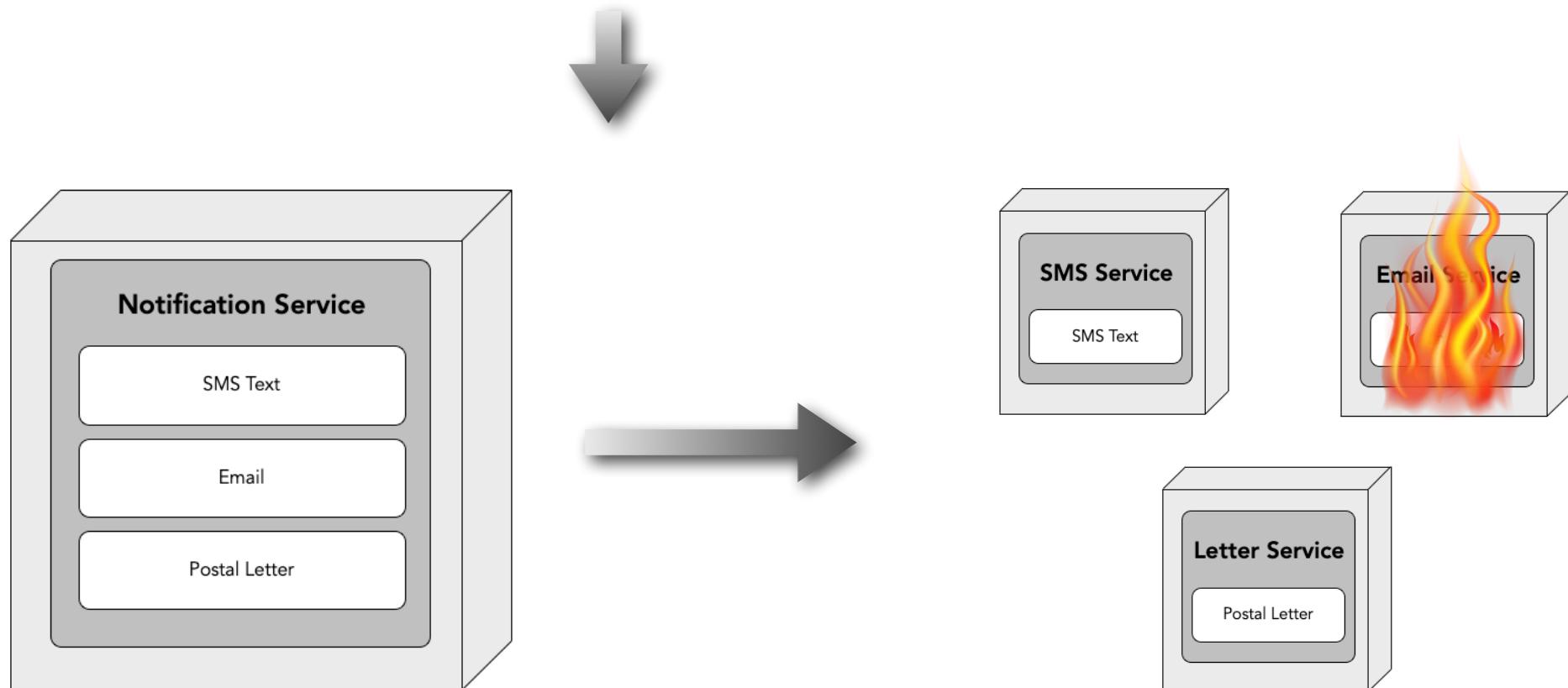
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



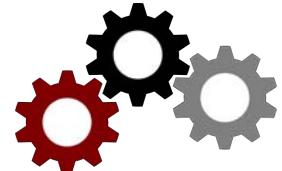
fault
tolerance



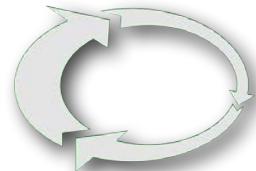
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



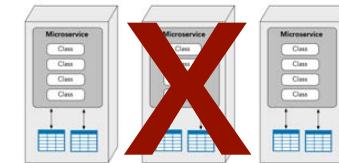
service
functionality



code
volatility



scalability and
throughput

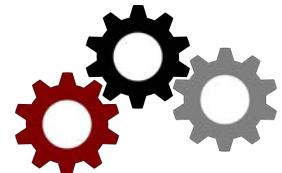


fault
tolerance

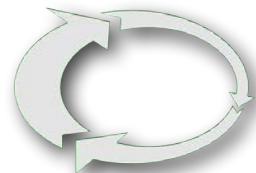
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



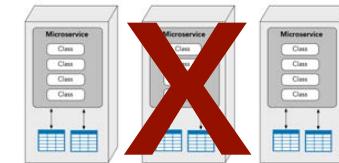
service
functionality



code
volatility



scalability and
throughput



fault
tolerance



data
security

service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



data
security

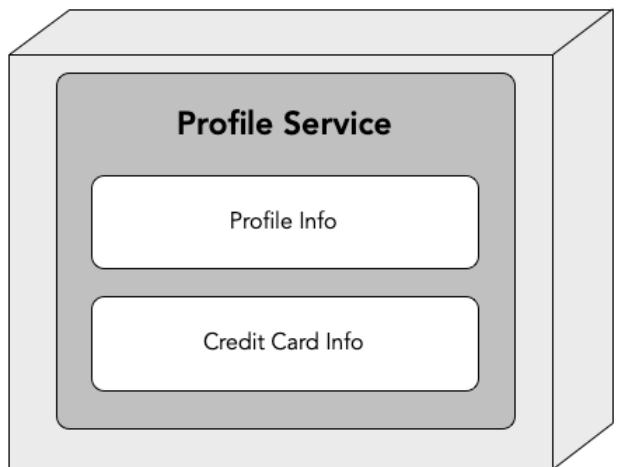
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



data
security



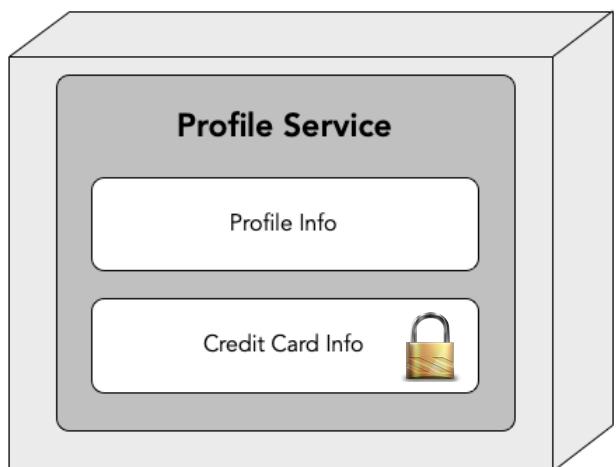
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



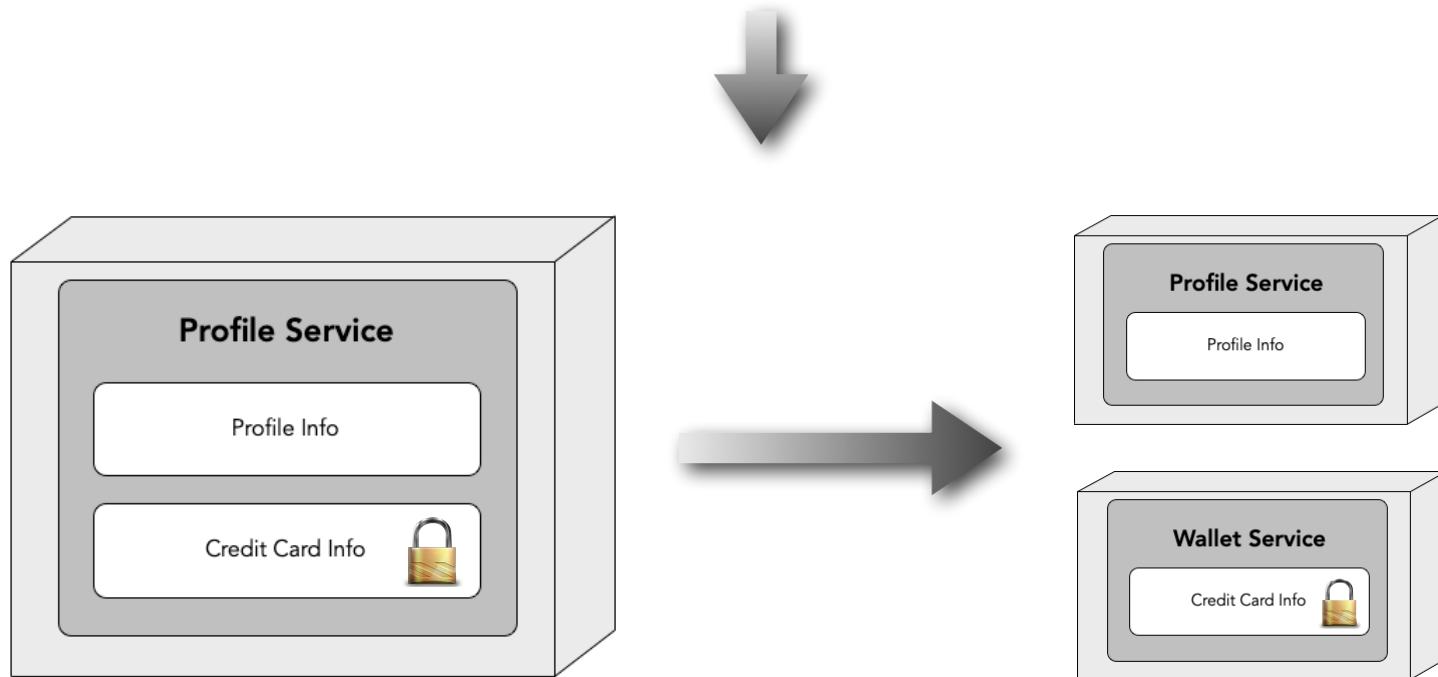
data
security



service granularity

granularity disintegrators

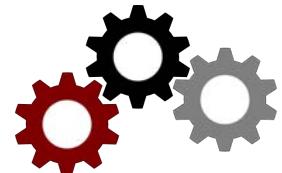
“when should I consider breaking apart a service?”



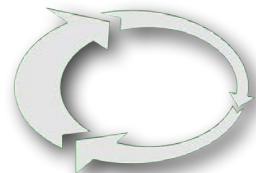
service granularity

granularity disintegrators

“when should I consider breaking apart a service?”



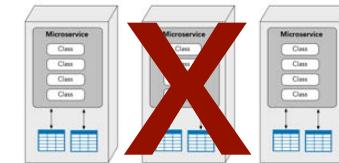
service
functionality



code
volatility



scalability and
throughput



fault
tolerance

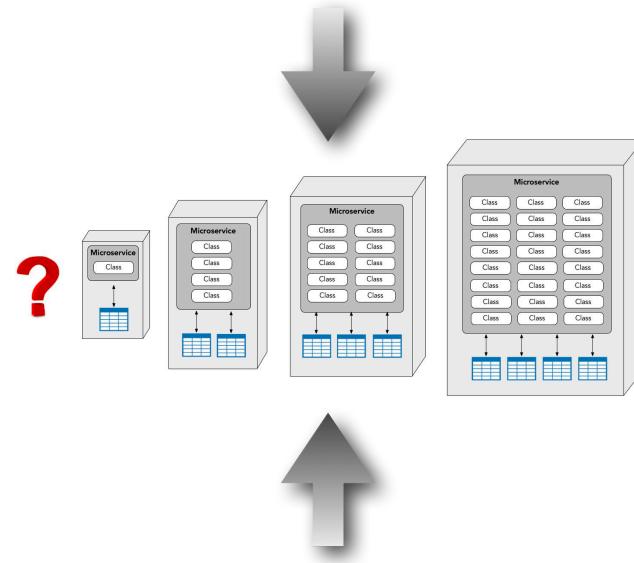


data
security

service granularity

granularity disintegrators

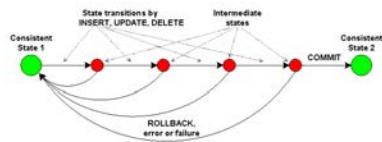
“when should I consider breaking apart a service?”



granularity integrators

“when should I consider putting services back together?”

service granularity



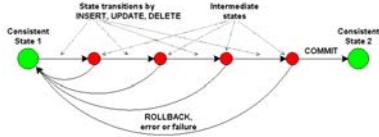
database
transactions



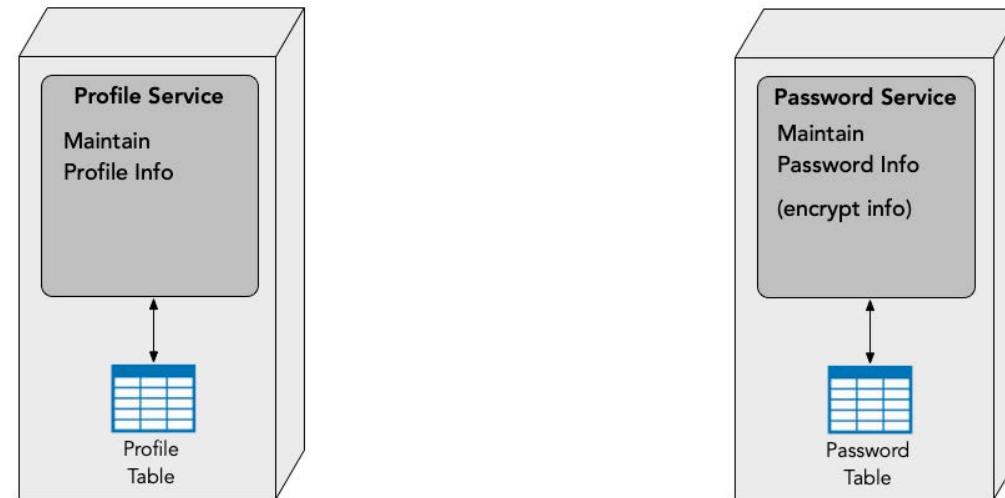
granularity integrators

"when should I consider putting services back together?"

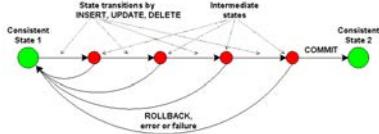
service granularity



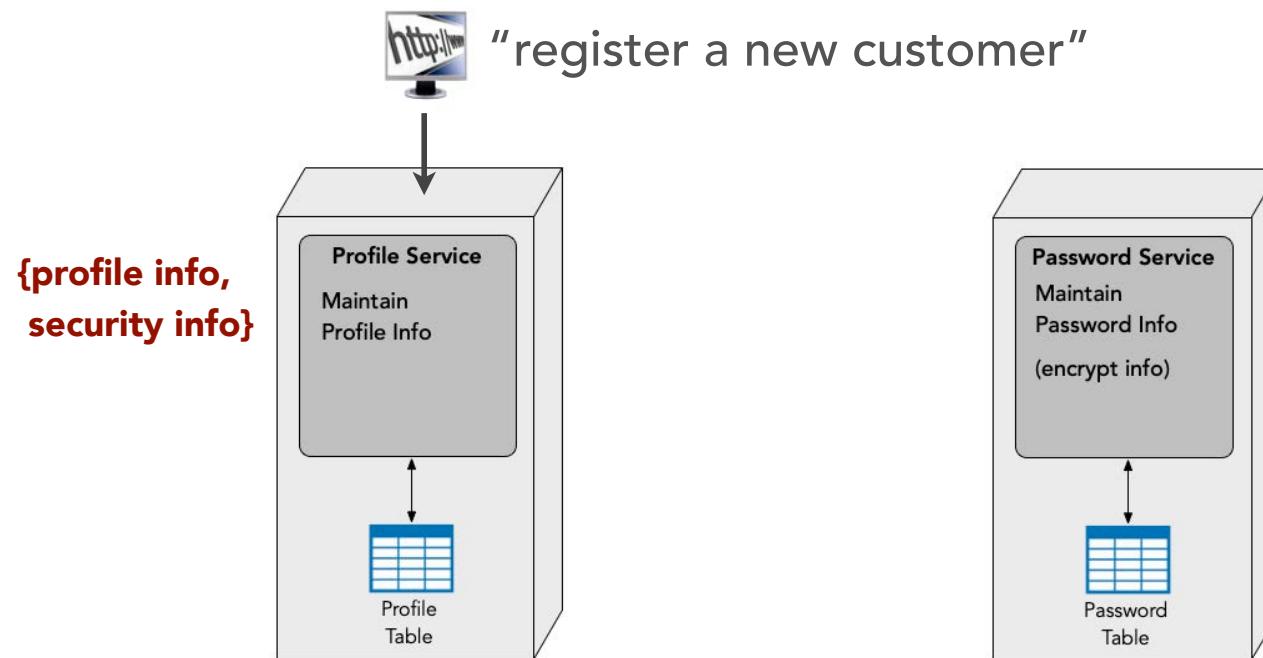
database
transactions



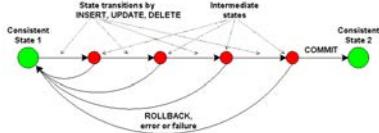
service granularity



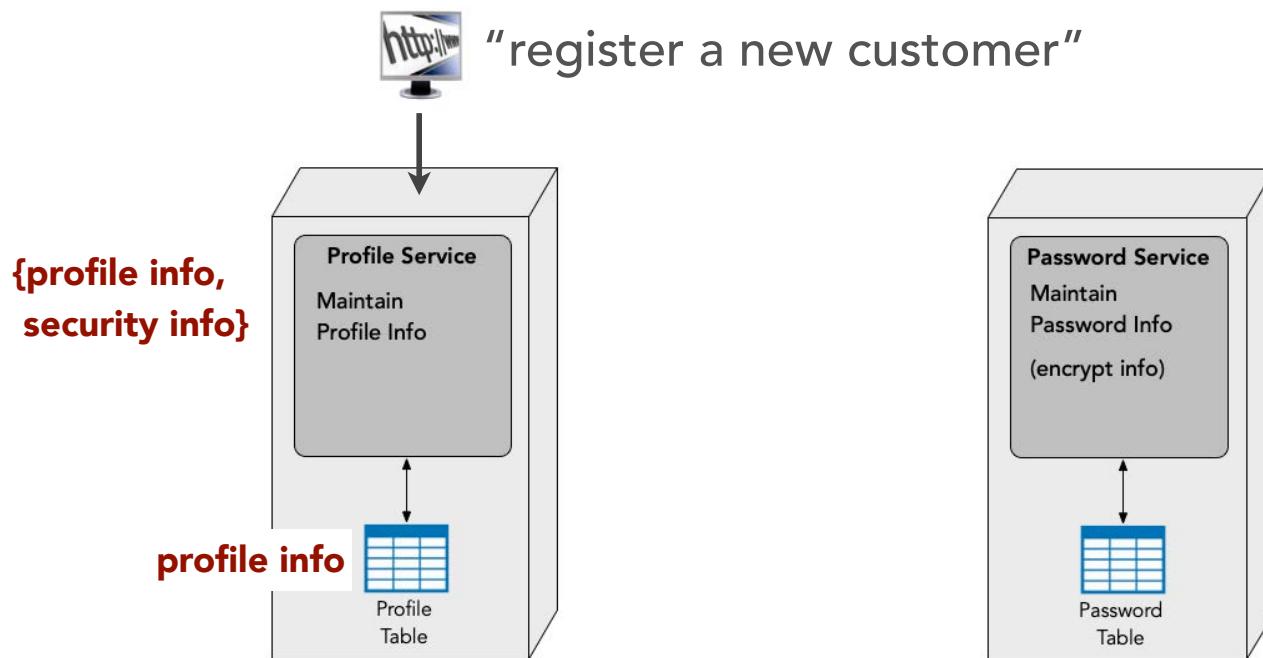
database
transactions



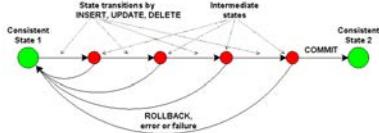
service granularity



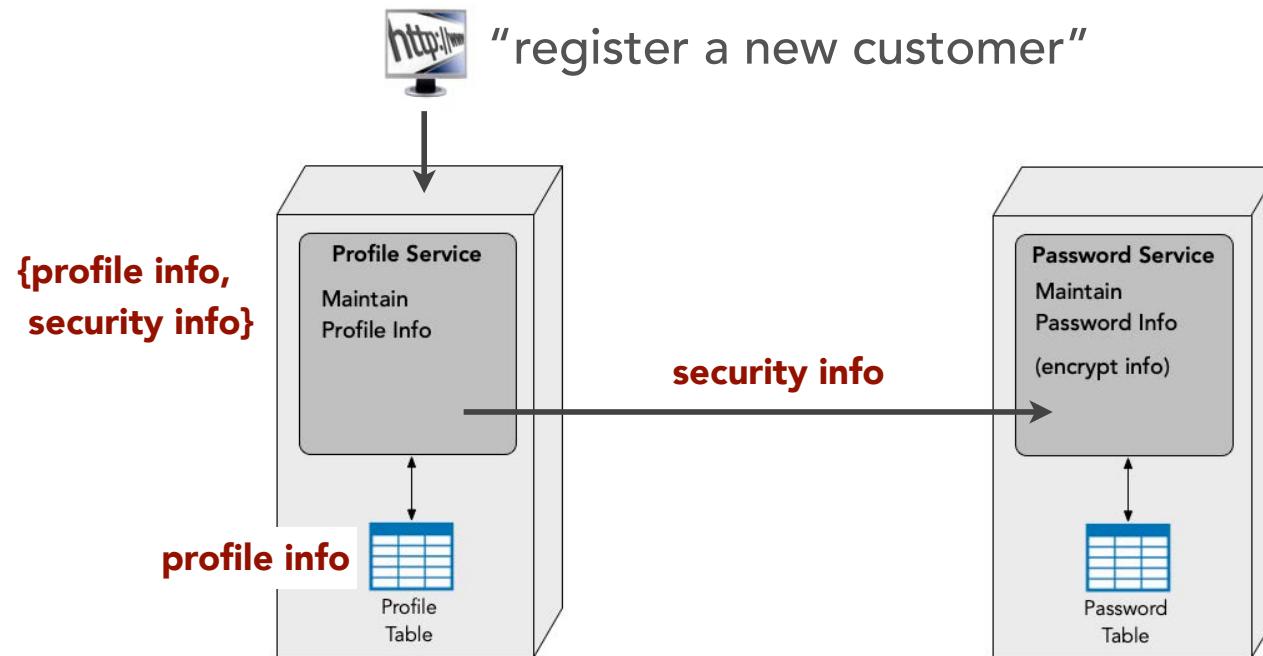
database
transactions



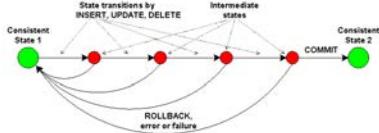
service granularity



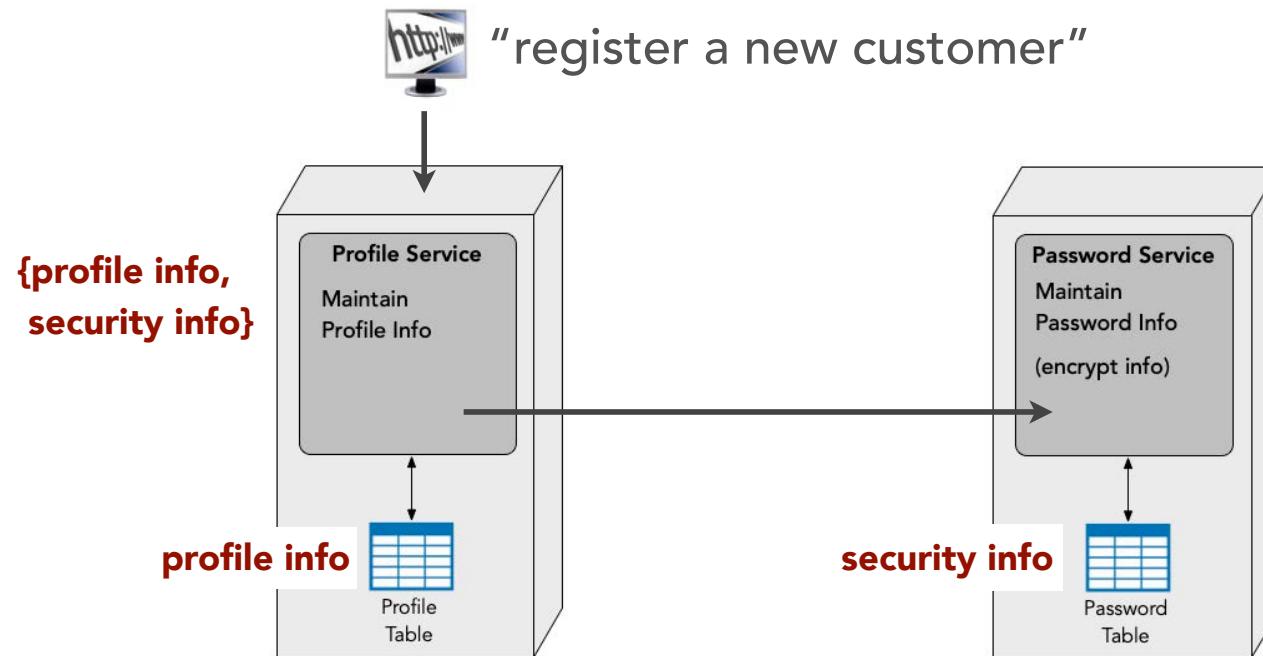
database
transactions



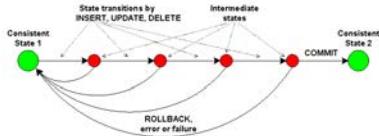
service granularity



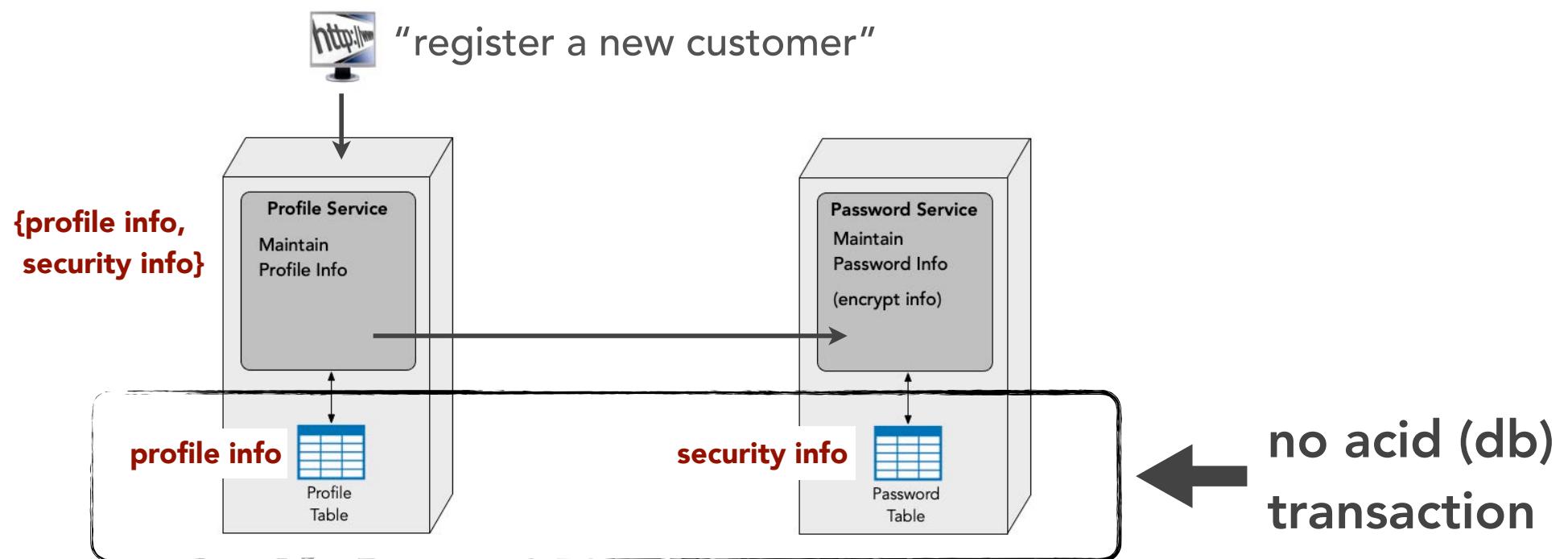
database
transactions



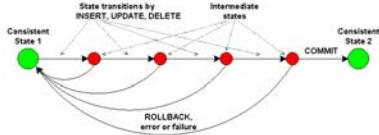
service granularity



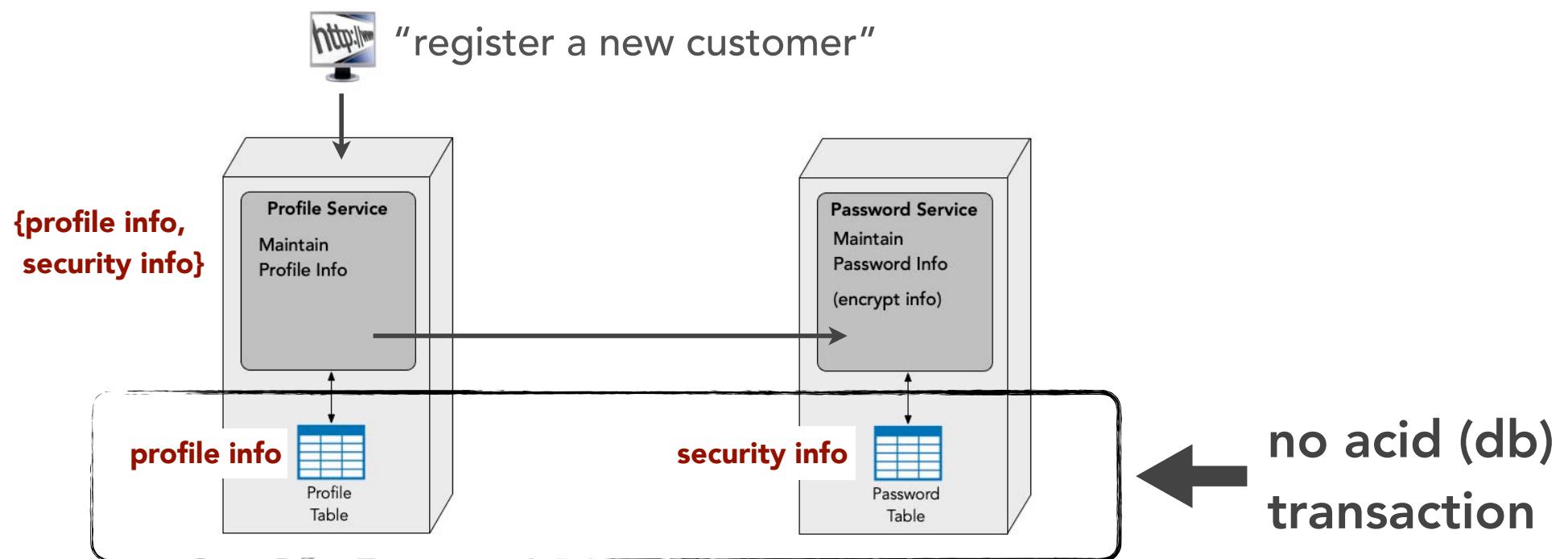
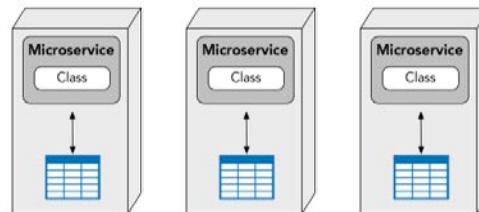
database
transactions



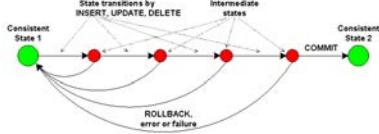
service granularity



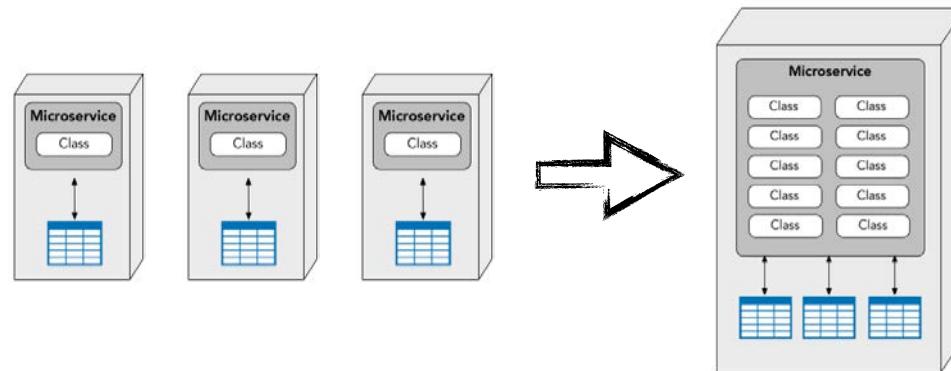
database
transactions



service granularity

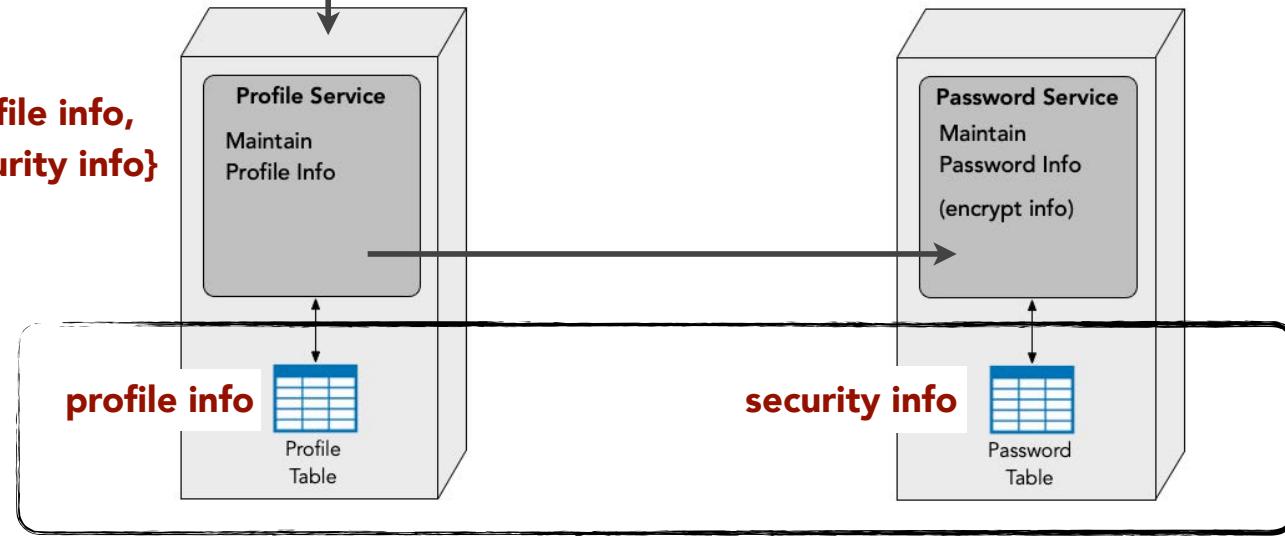


database
transactions



"register a new customer"

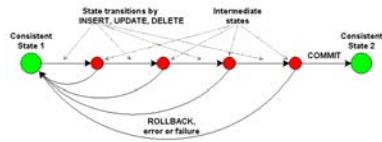
{profile info,
security info}



security info

no acid (db)
transaction

service granularity



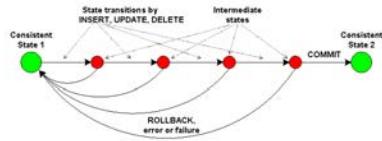
database
transactions



granularity integrators

"when should I consider putting services back together?"

service granularity



database
transactions



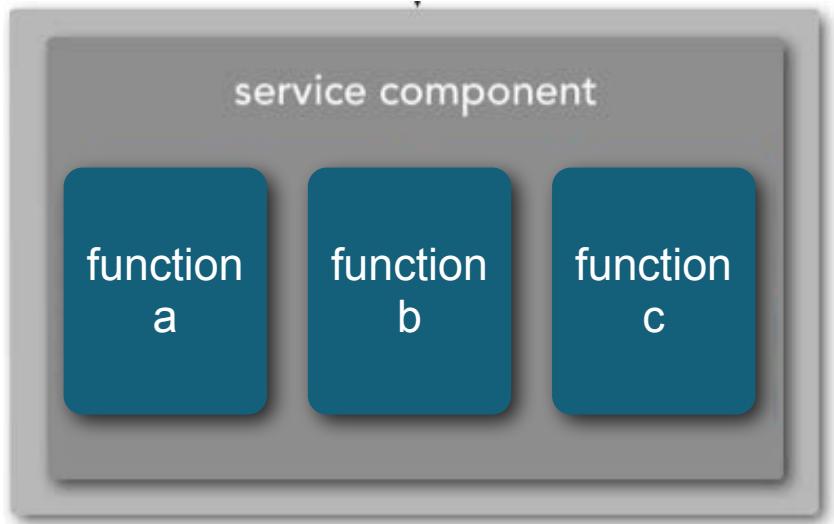
data
dependencies



granularity integrators

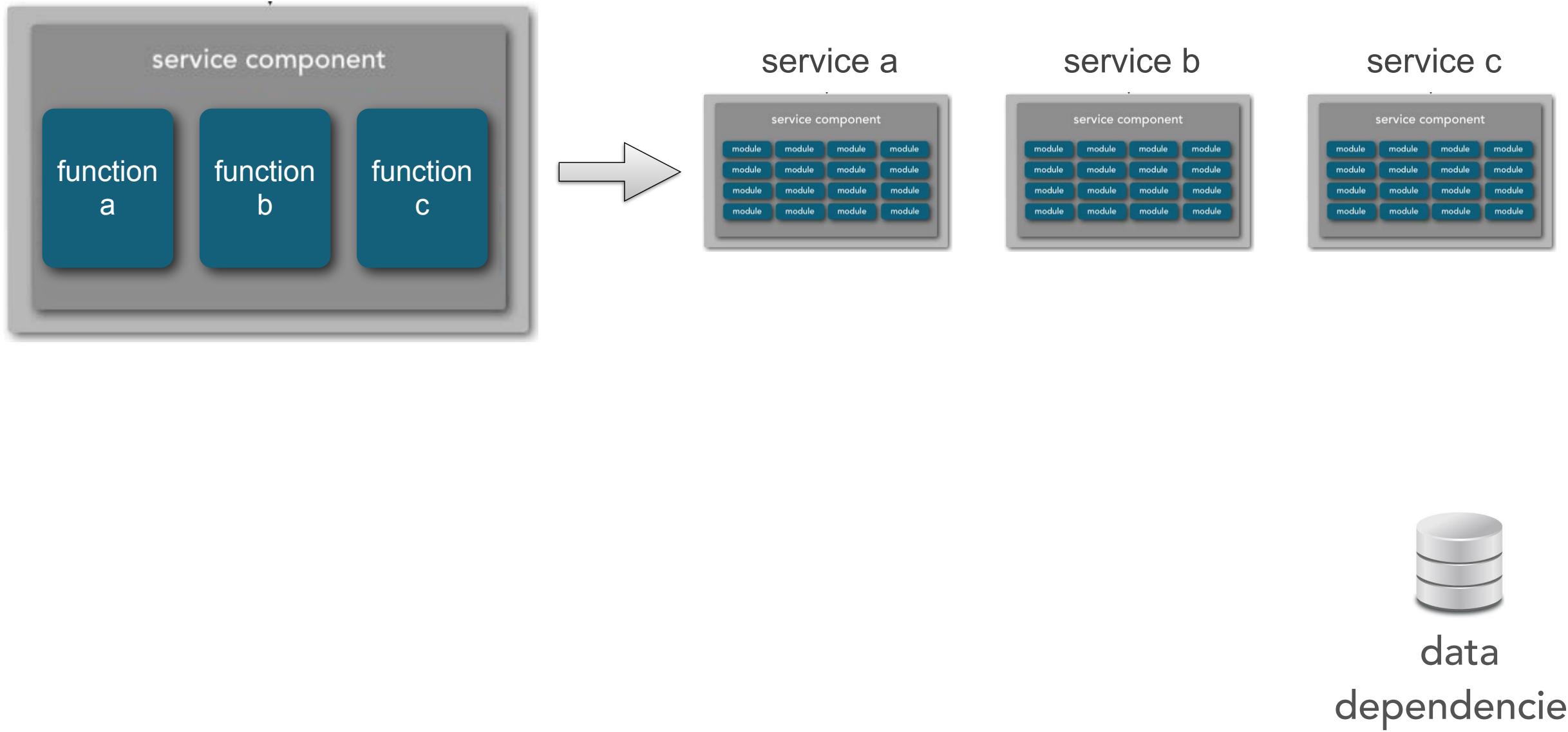
"when should I consider putting services back together?"

service granularity

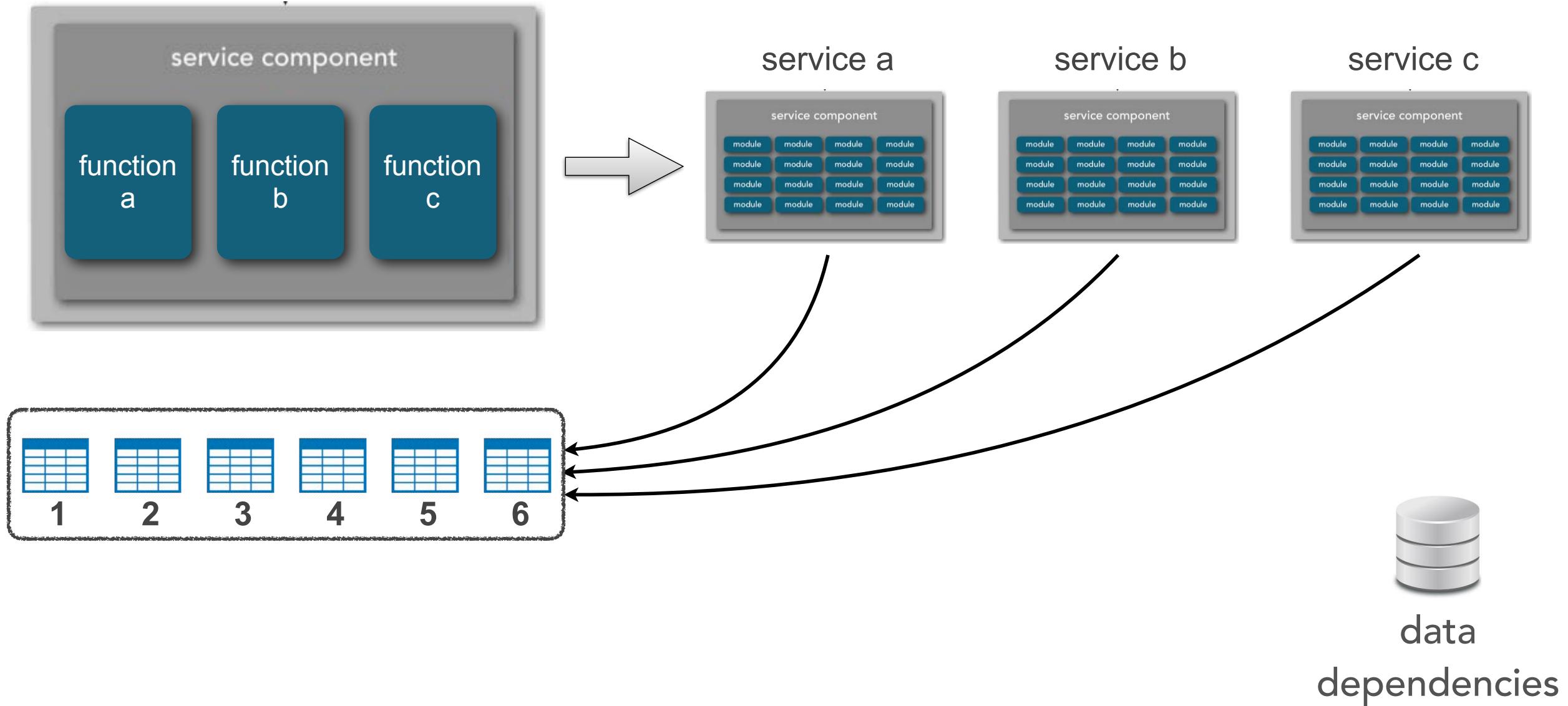


data
dependencies

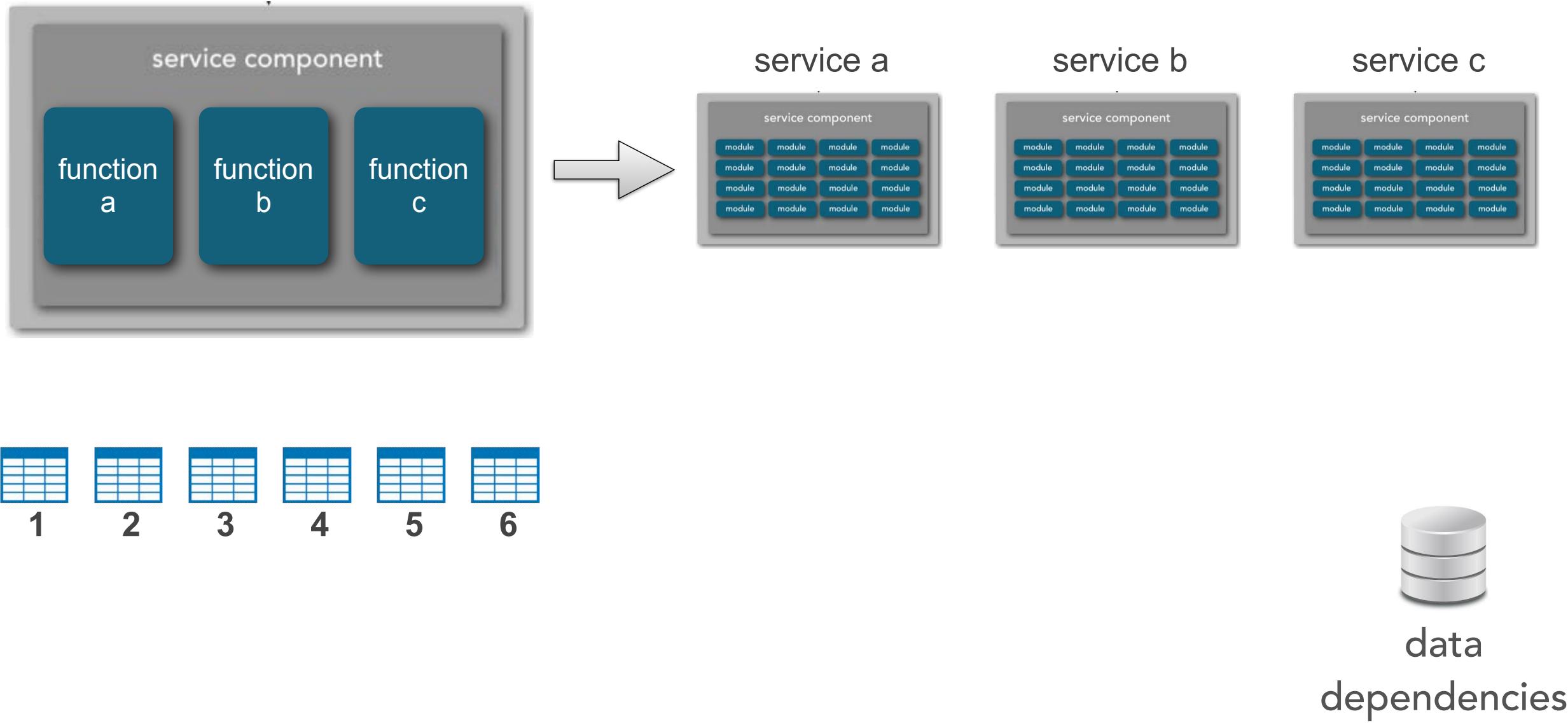
service granularity



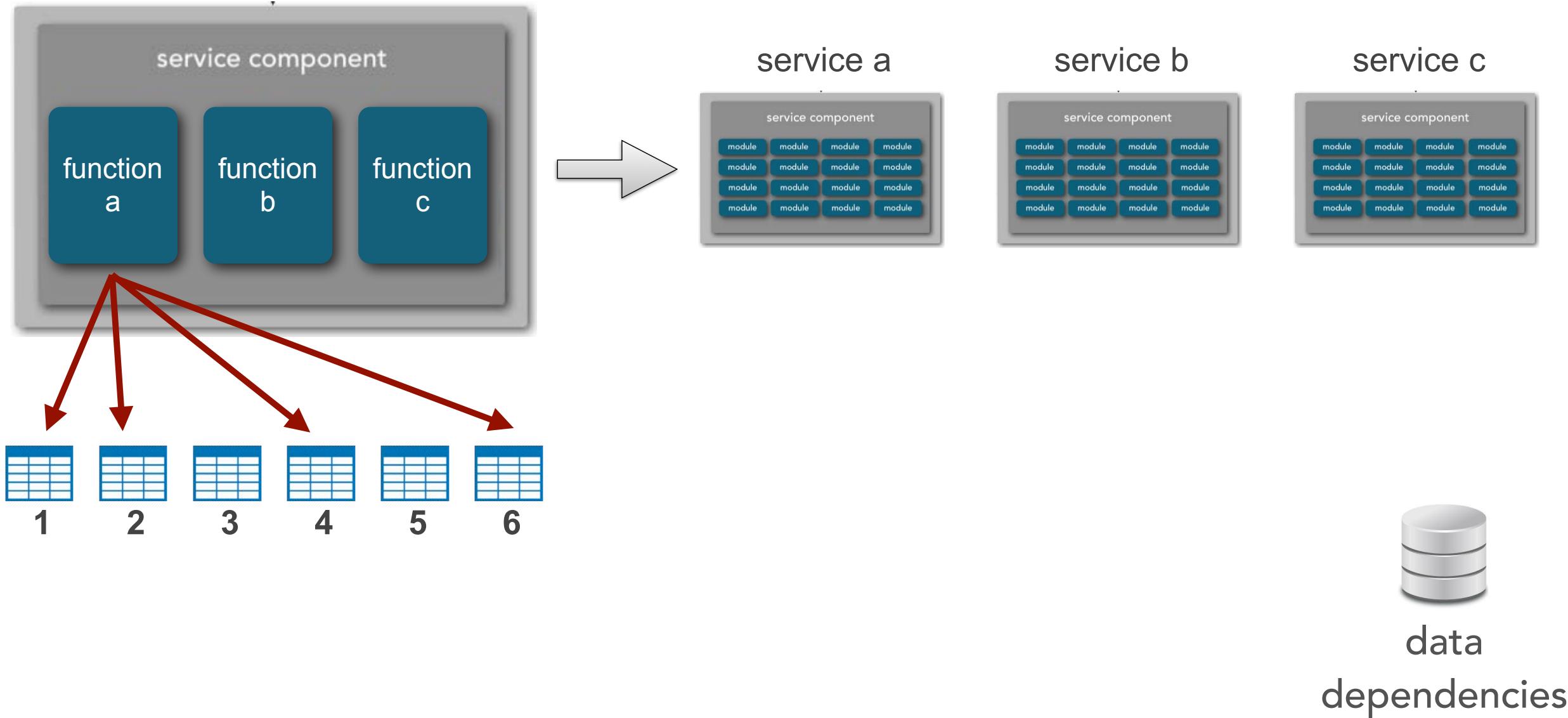
service granularity



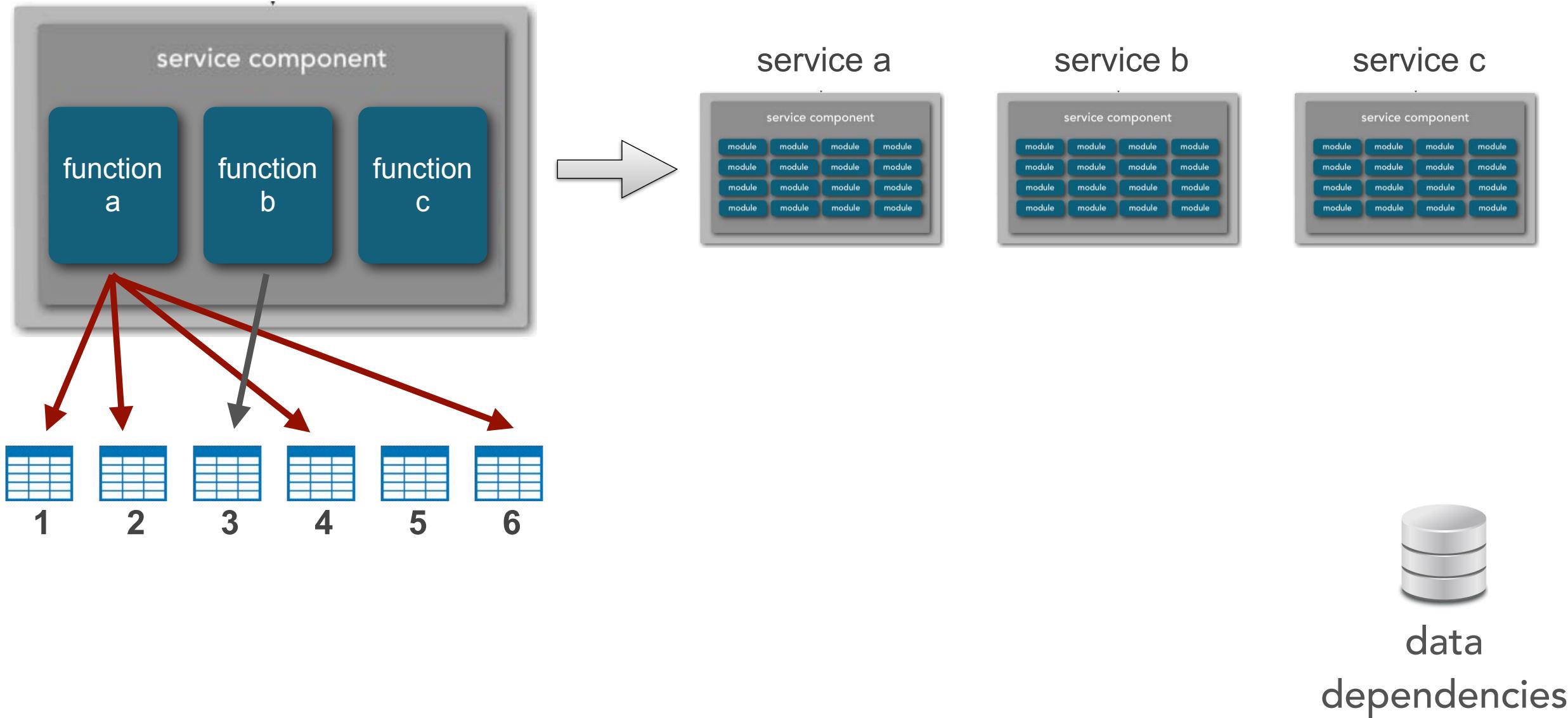
service granularity



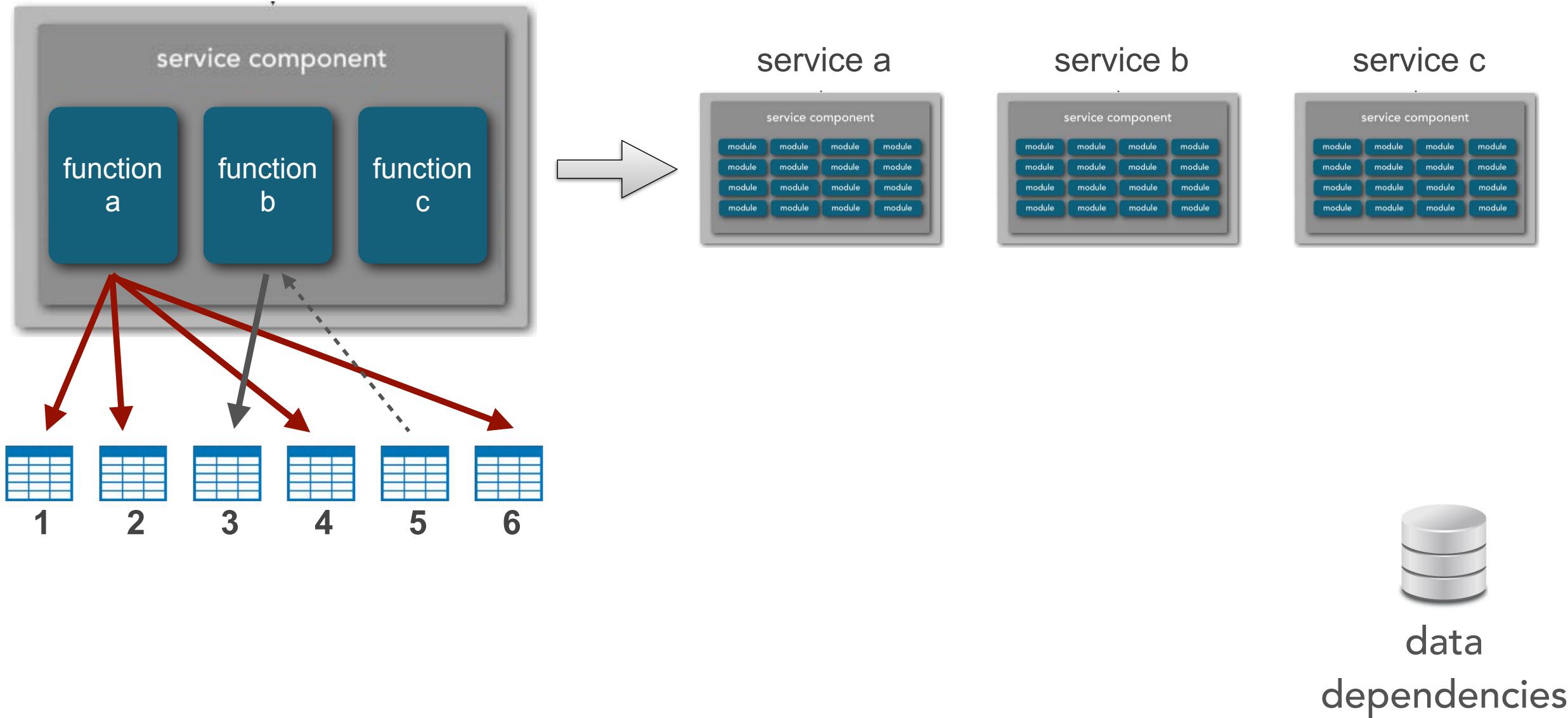
service granularity



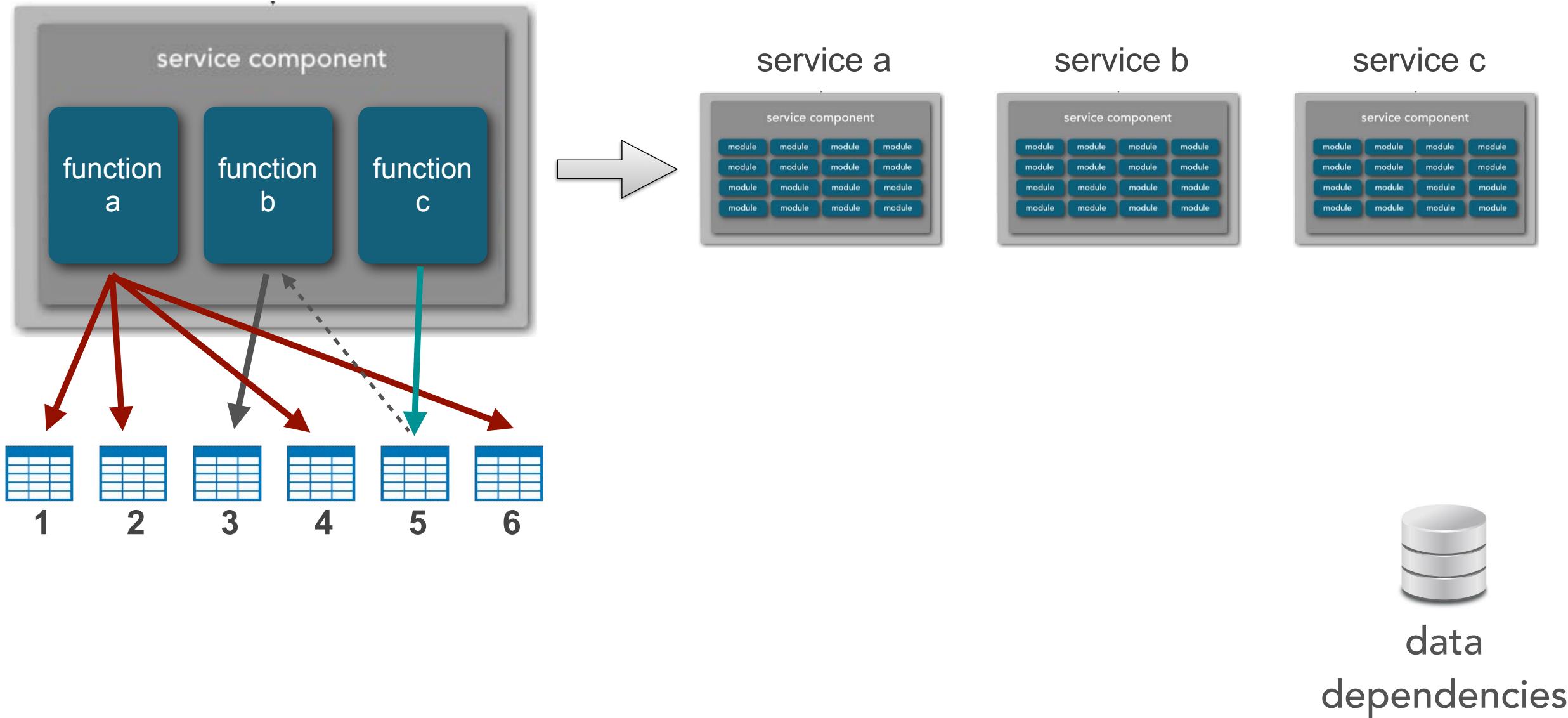
service granularity



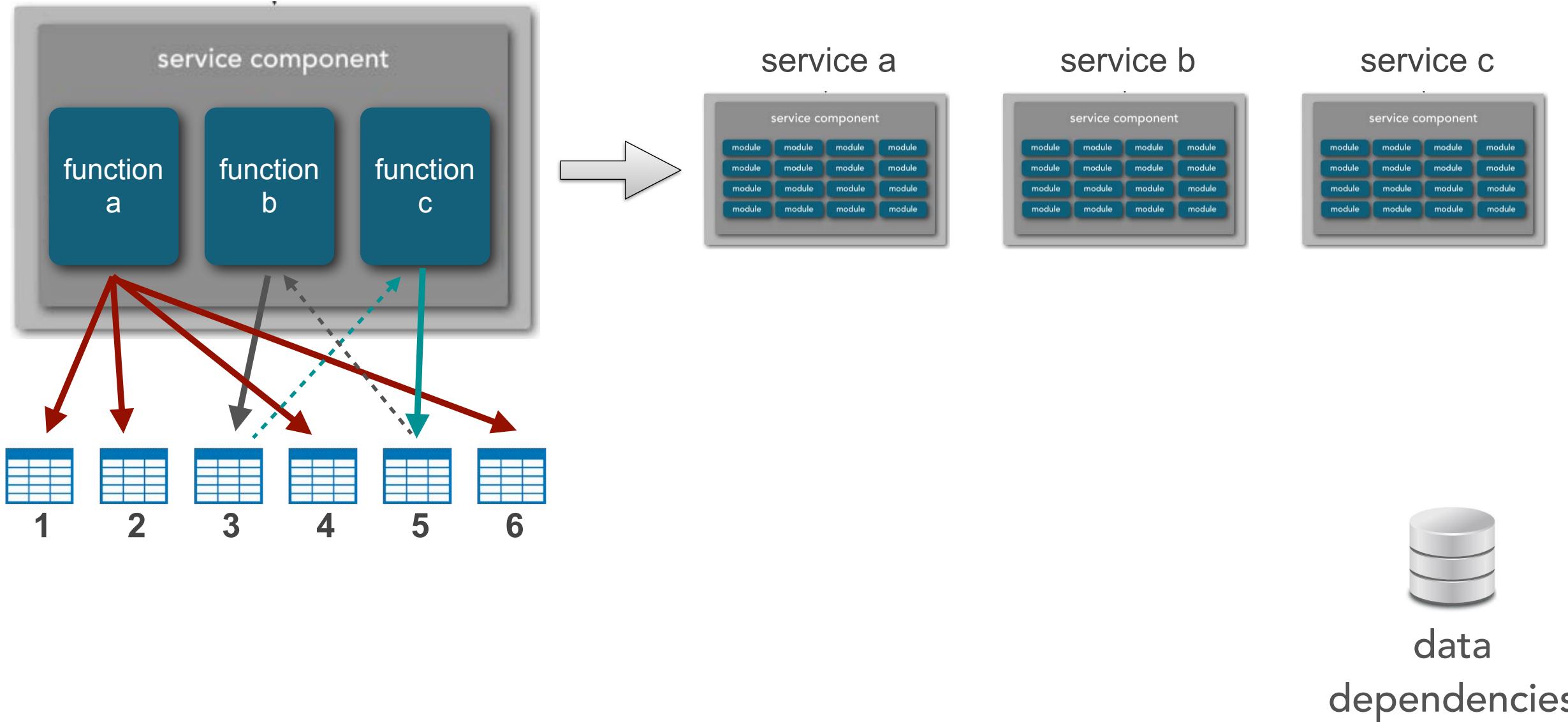
service granularity



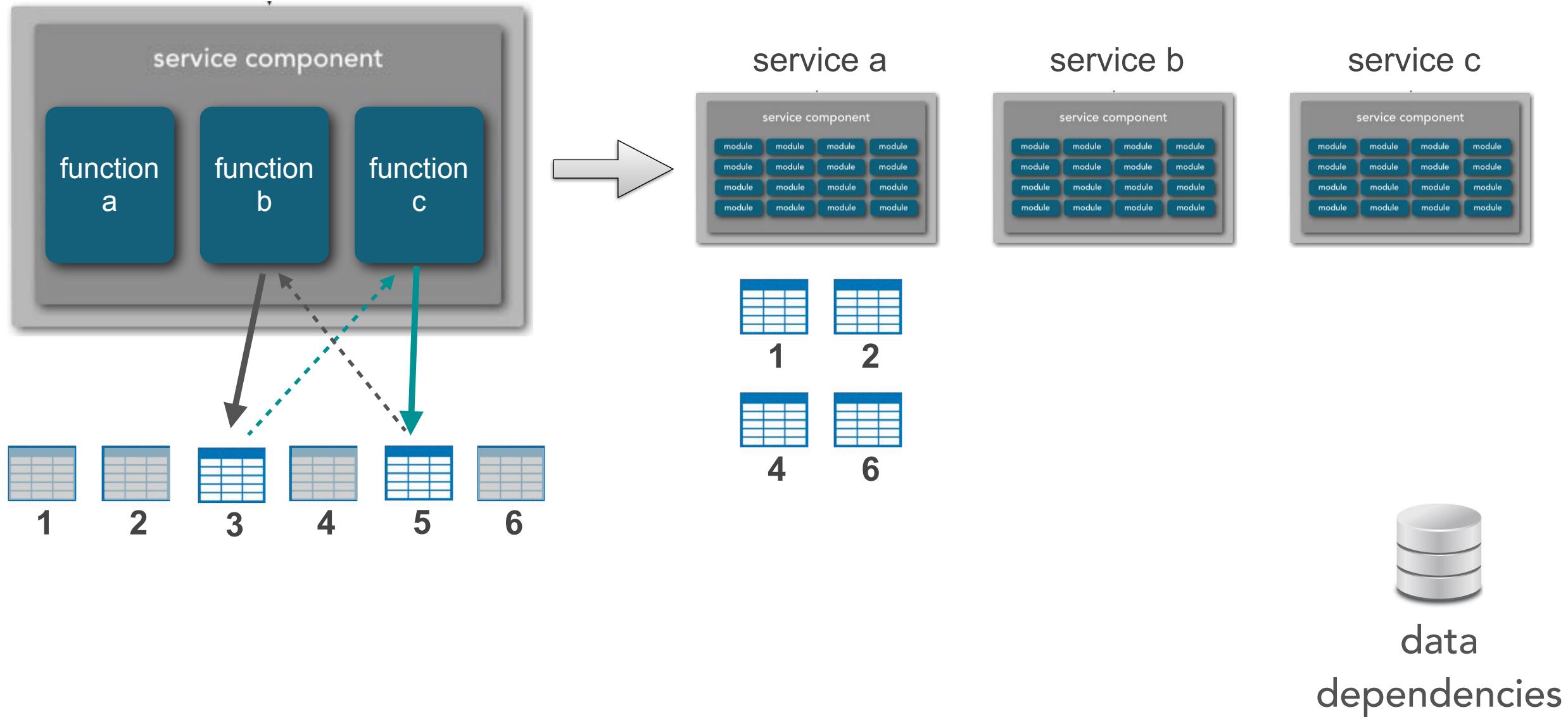
service granularity



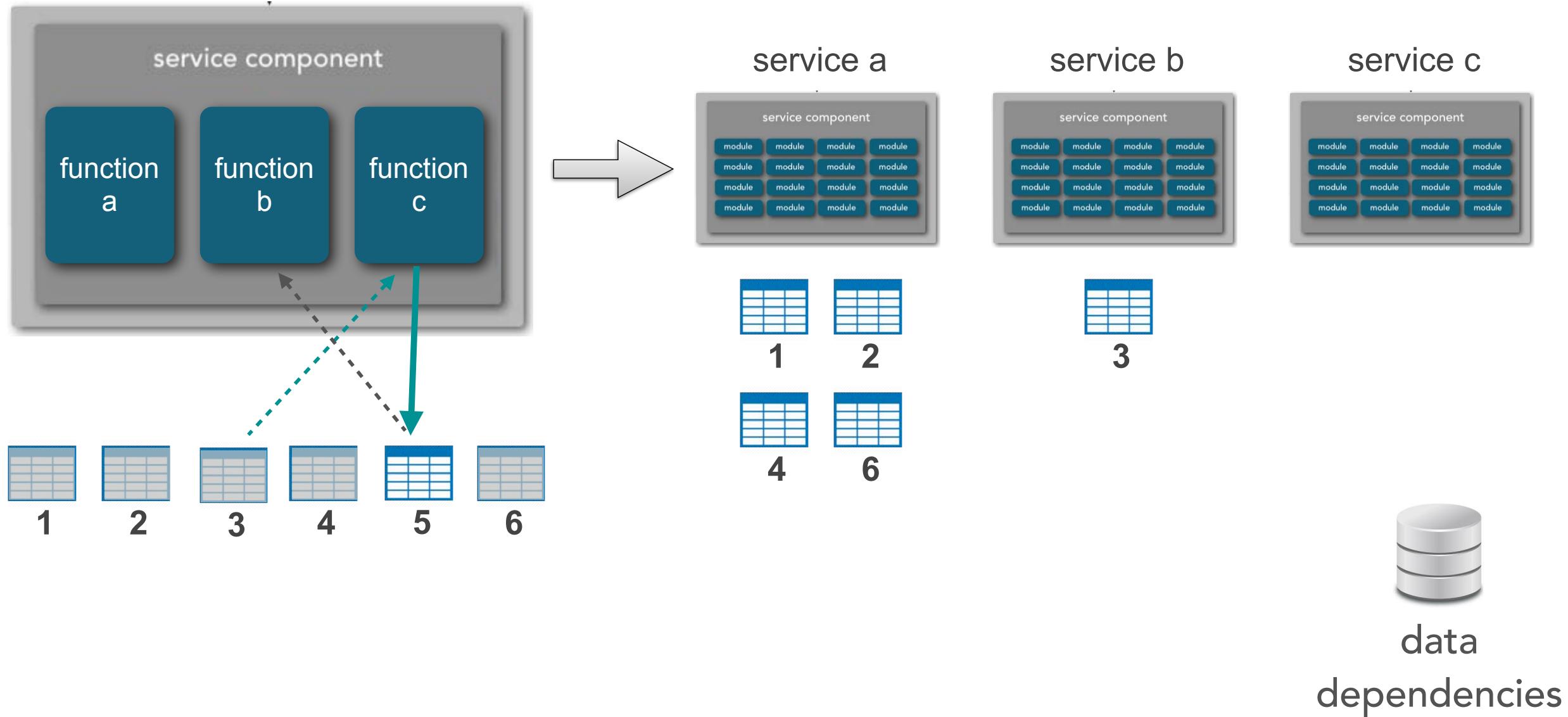
service granularity



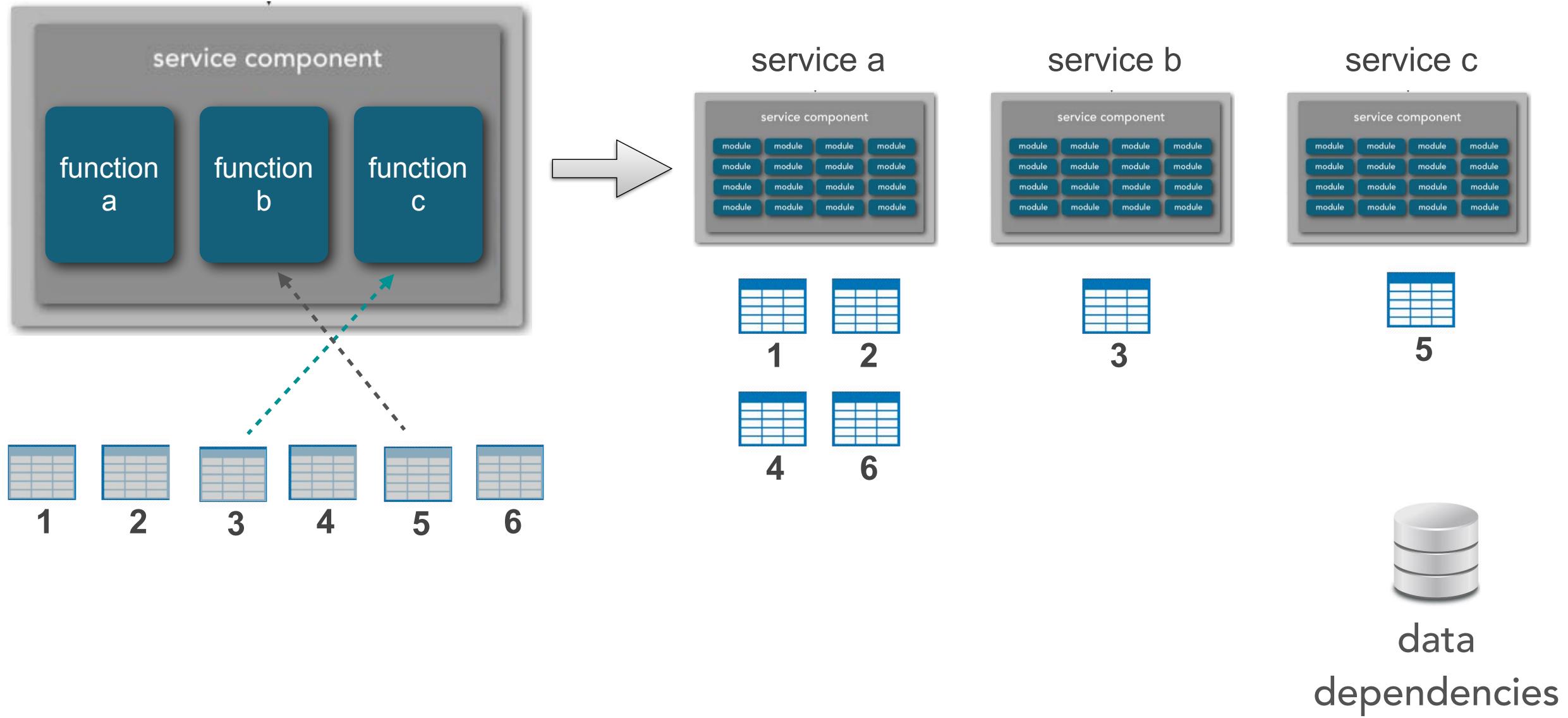
service granularity



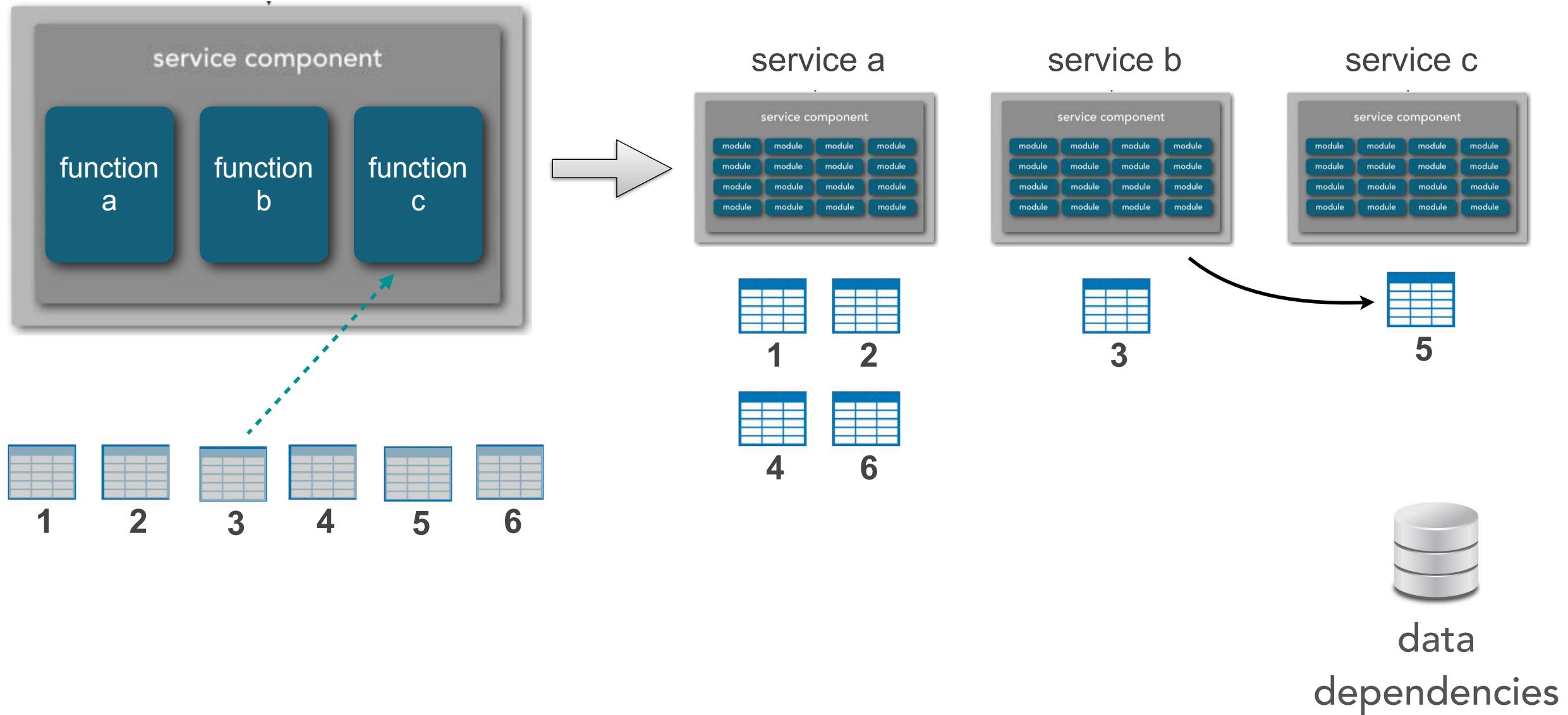
service granularity



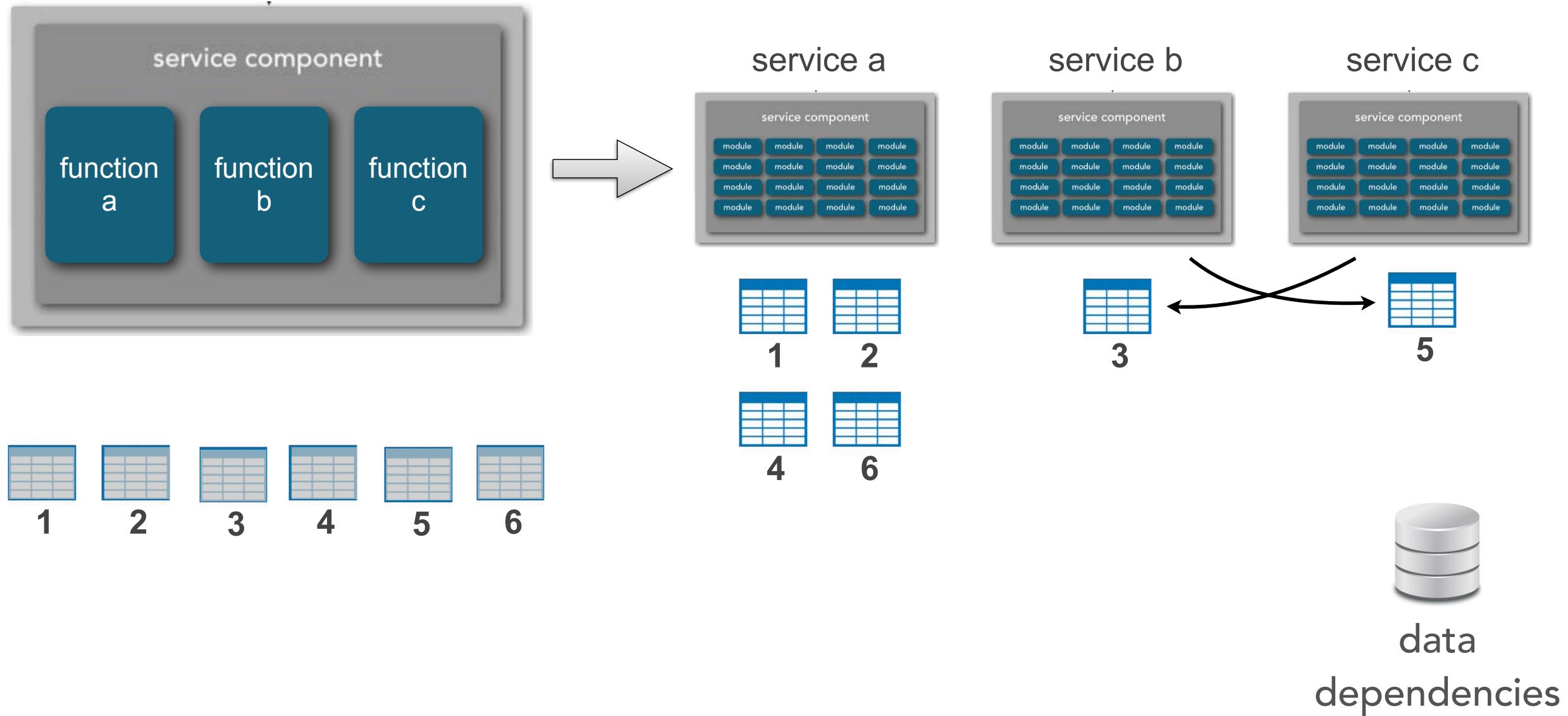
service granularity



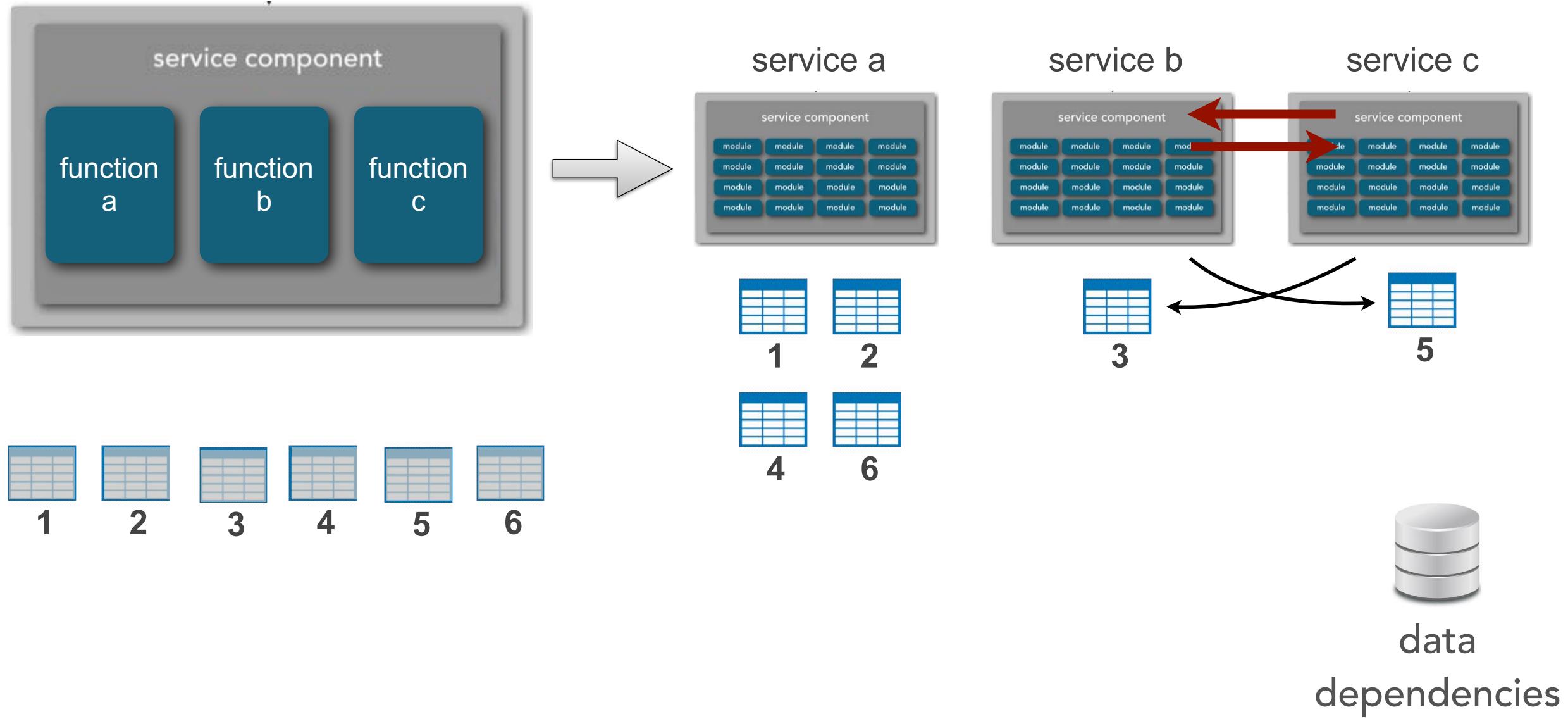
service granularity



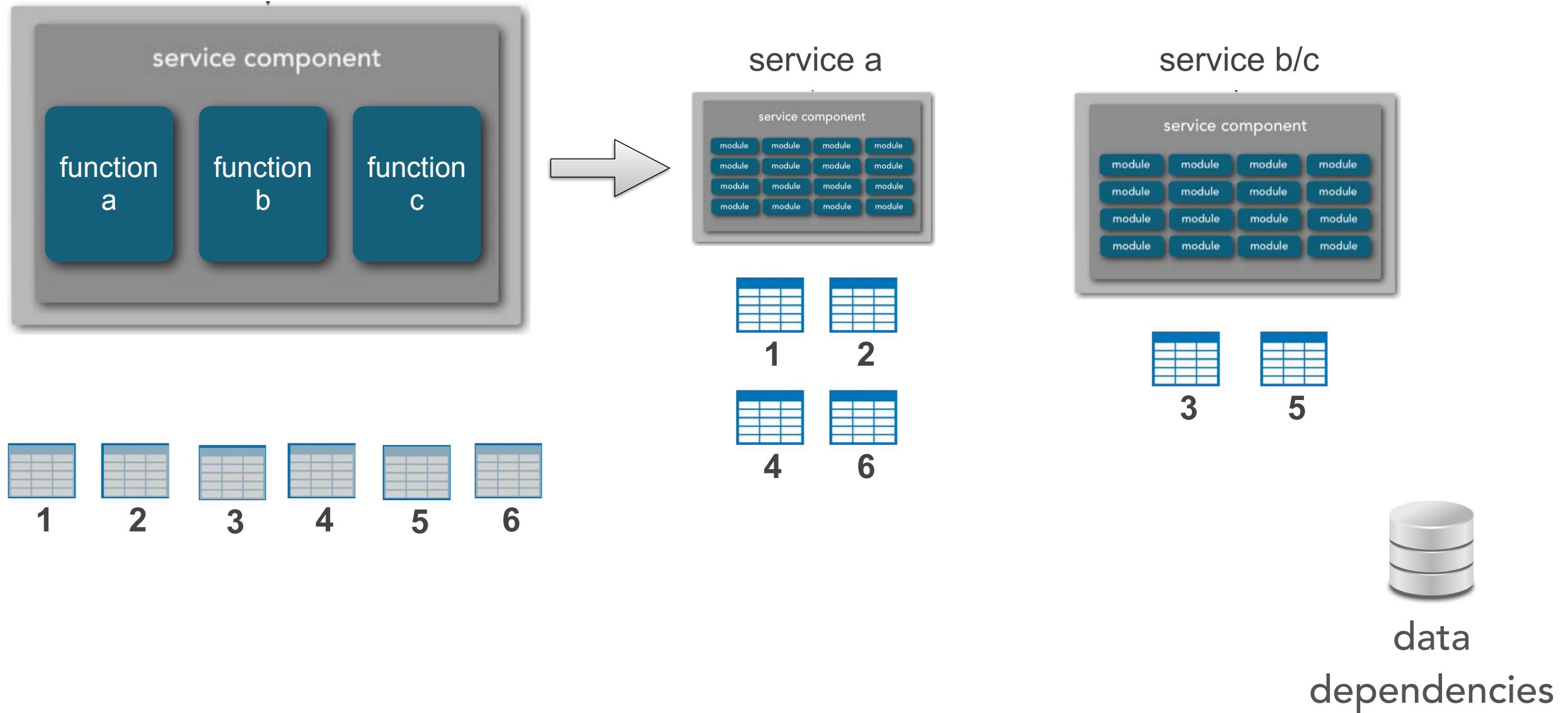
service granularity



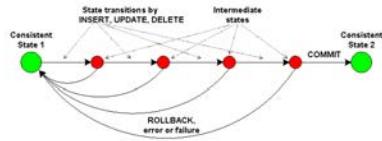
service granularity



service granularity



service granularity



database
transactions



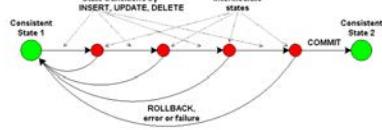
data
dependencies



granularity integrators

"when should I consider putting services back together?"

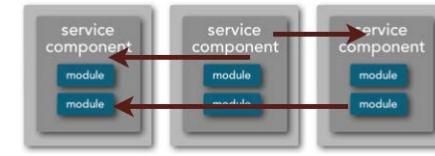
service granularity



database
transactions



data
dependencies



workflow and
choreography



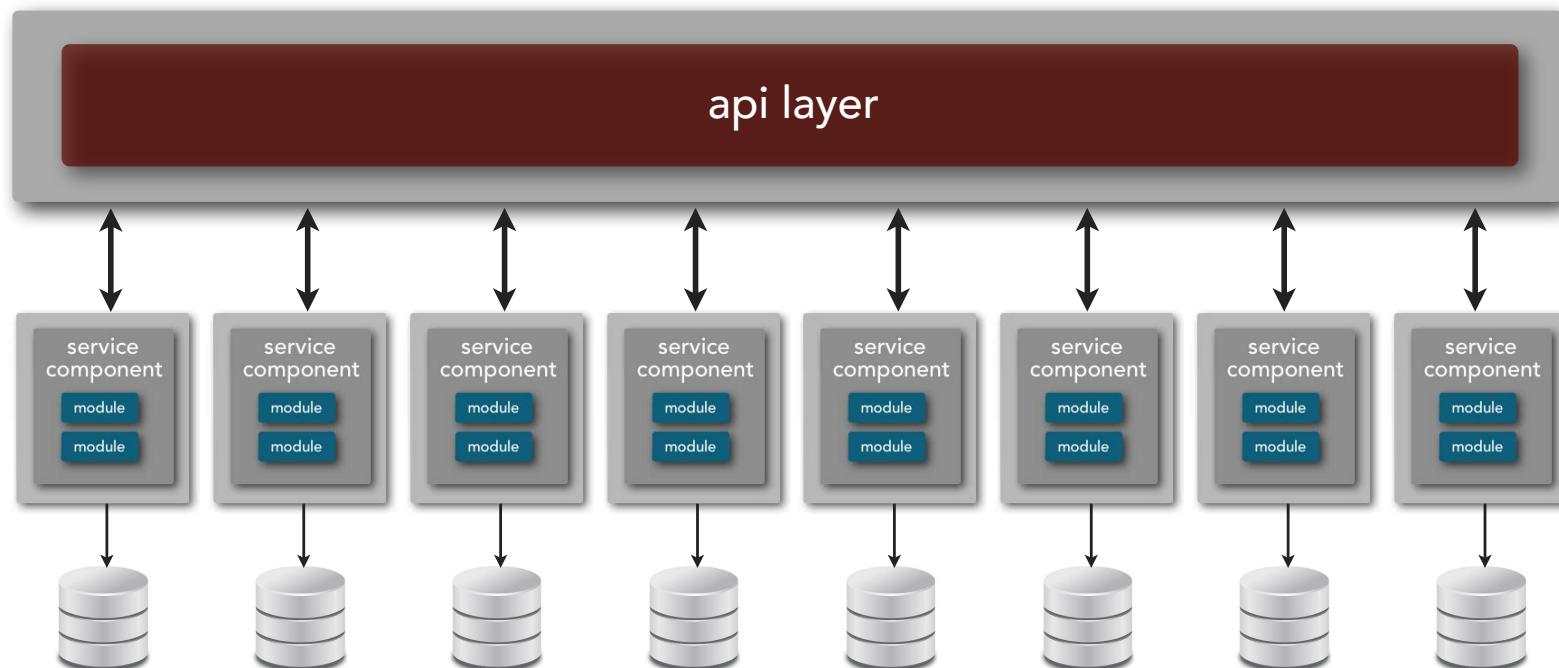
granularity integrators

"when should I consider putting services back together?"

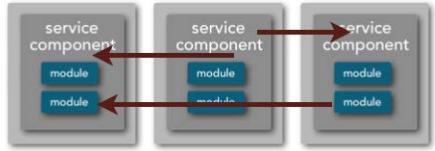
service granularity



workflow and
choreography

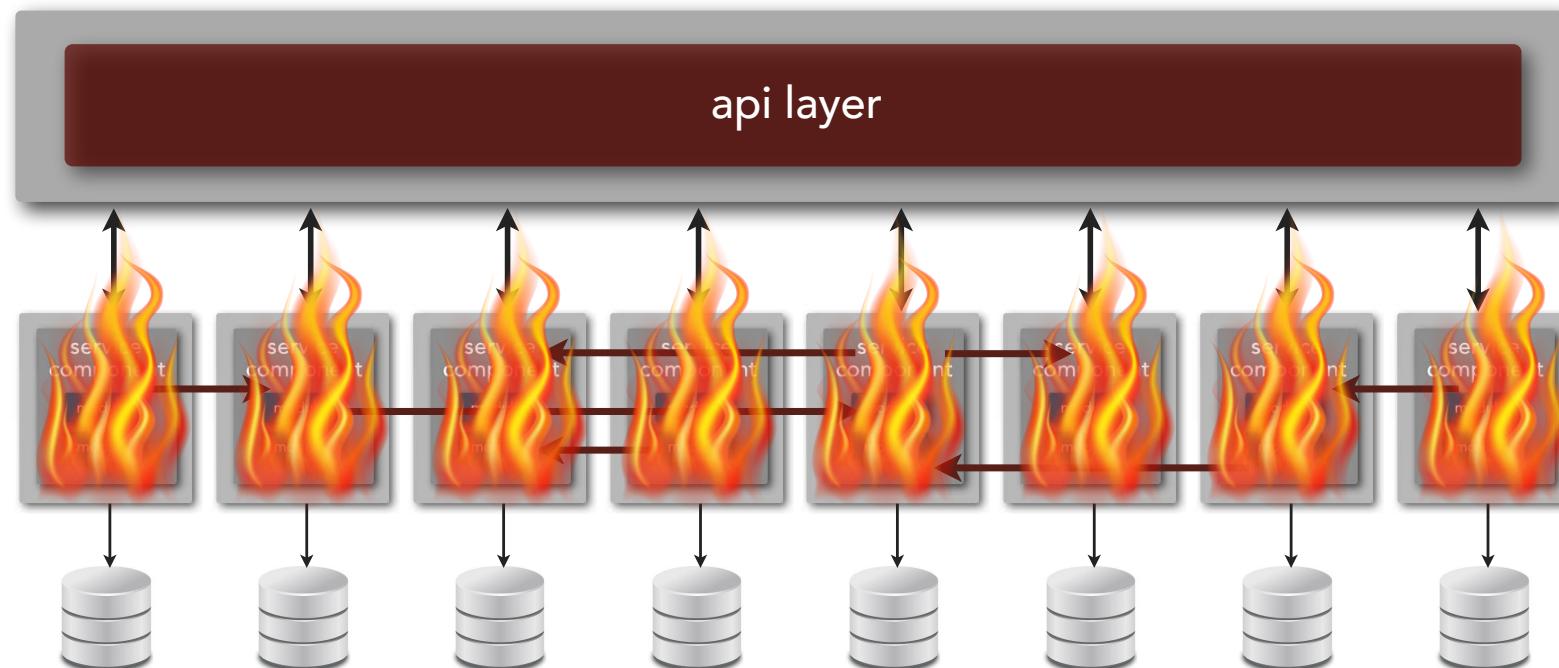


service granularity

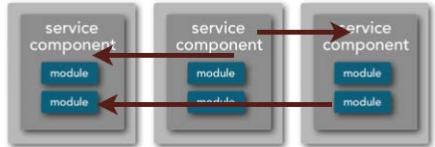


workflow and
choreography

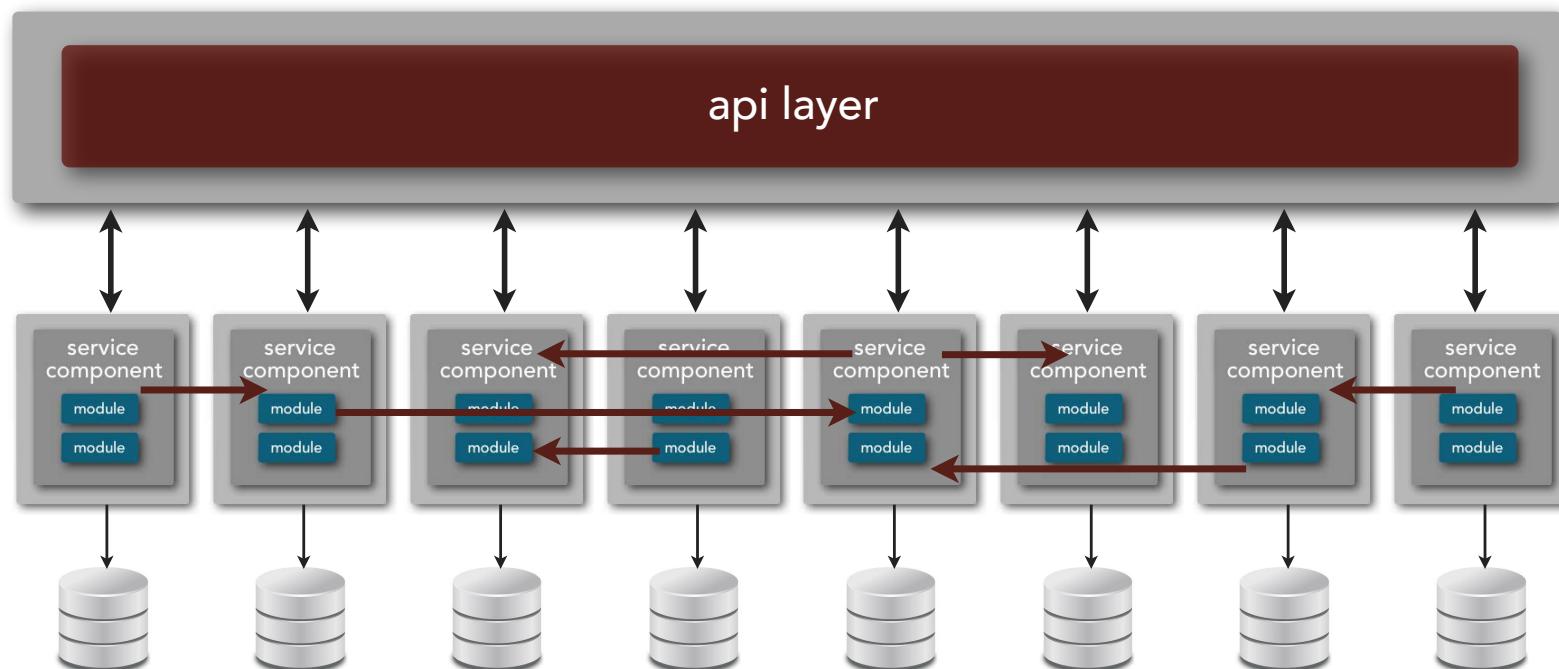
fault tolerance (availability)



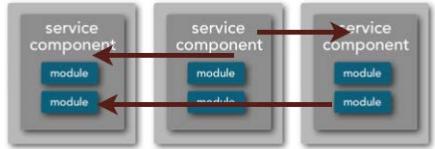
service granularity



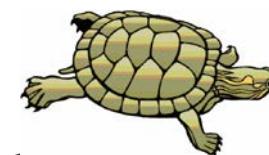
workflow and
choreography



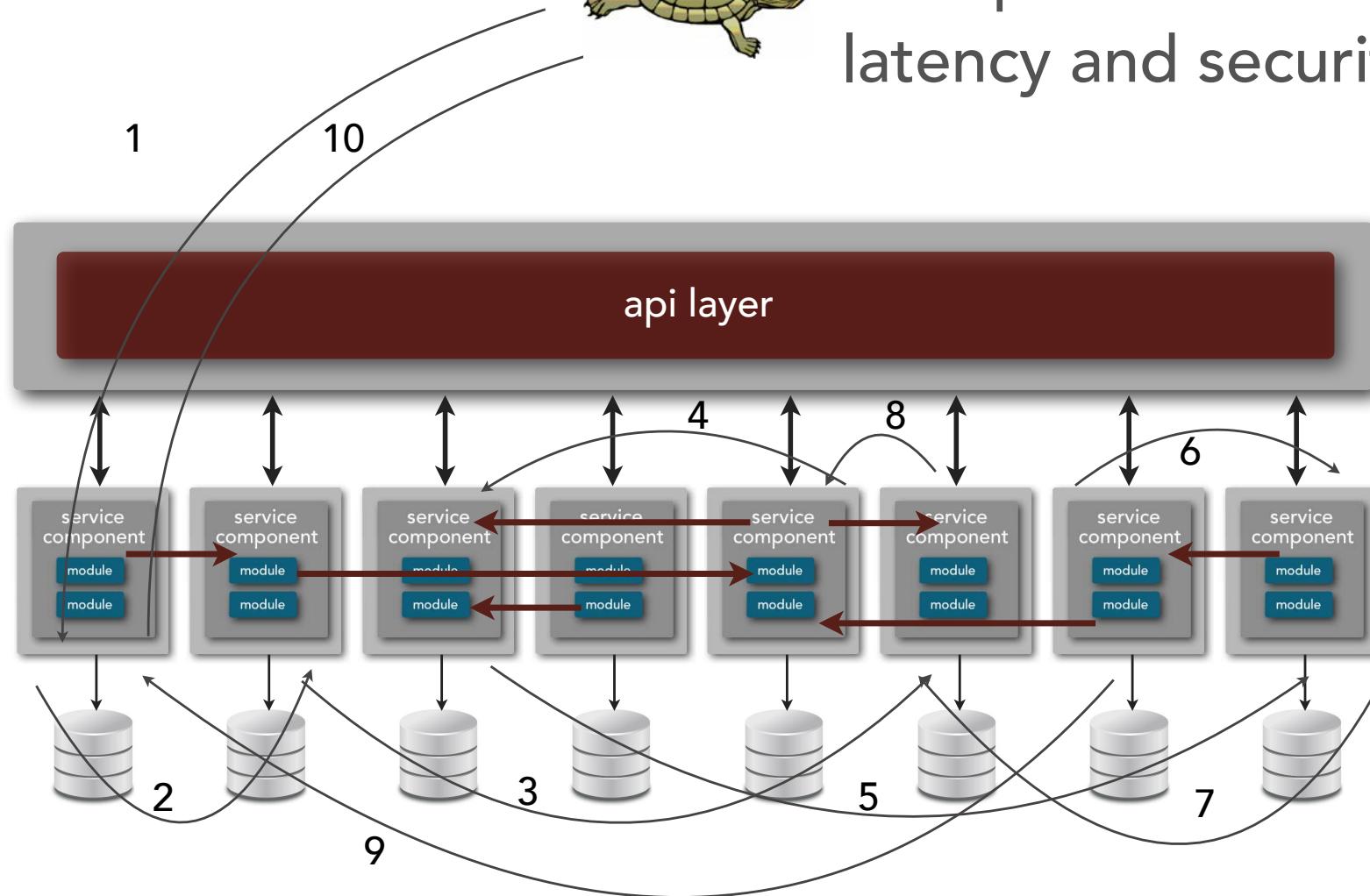
service granularity



workflow and
choreography



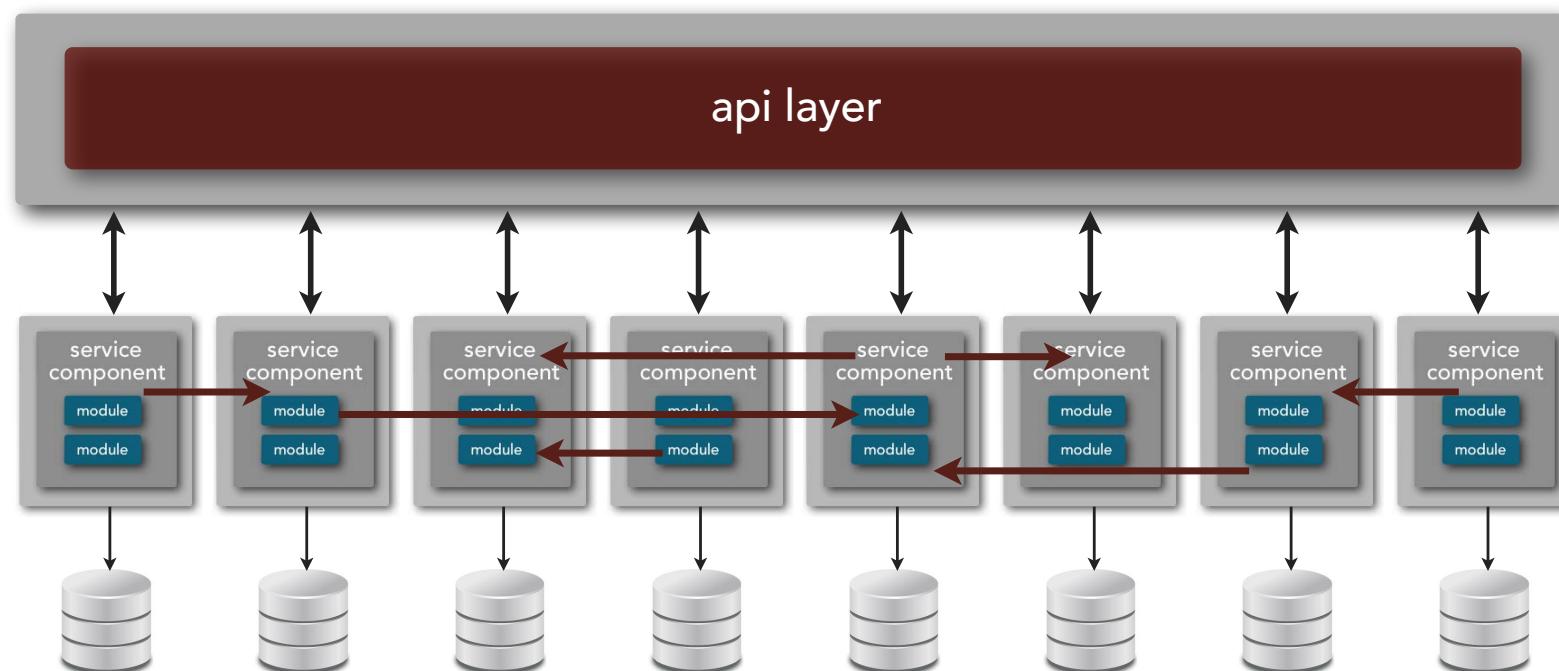
slow performance due to
latency and security



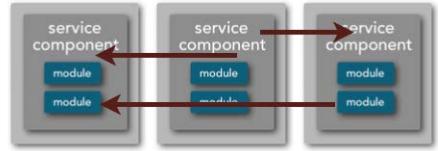
service granularity



workflow and
choreography



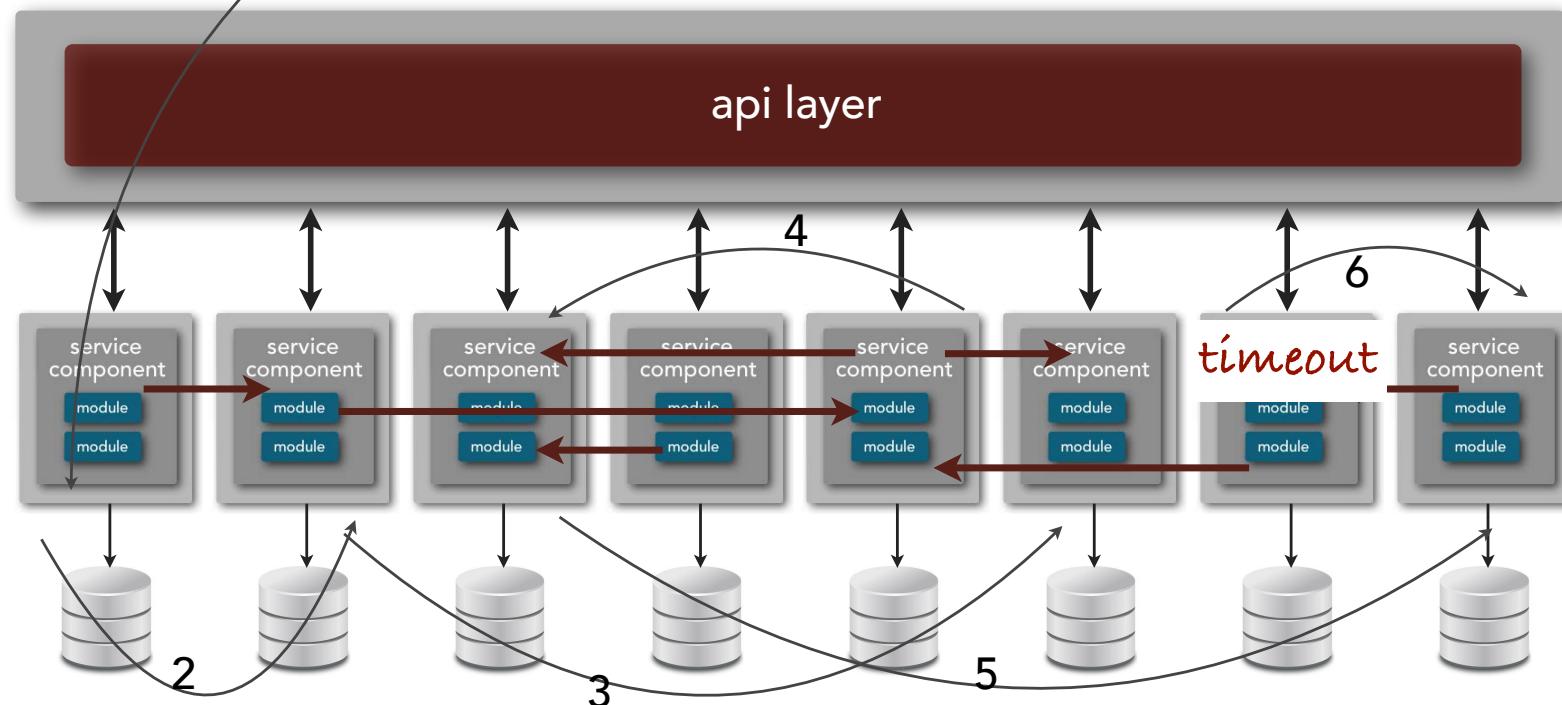
service granularity



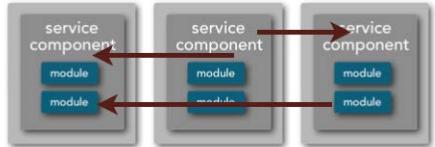
workflow and
choreography



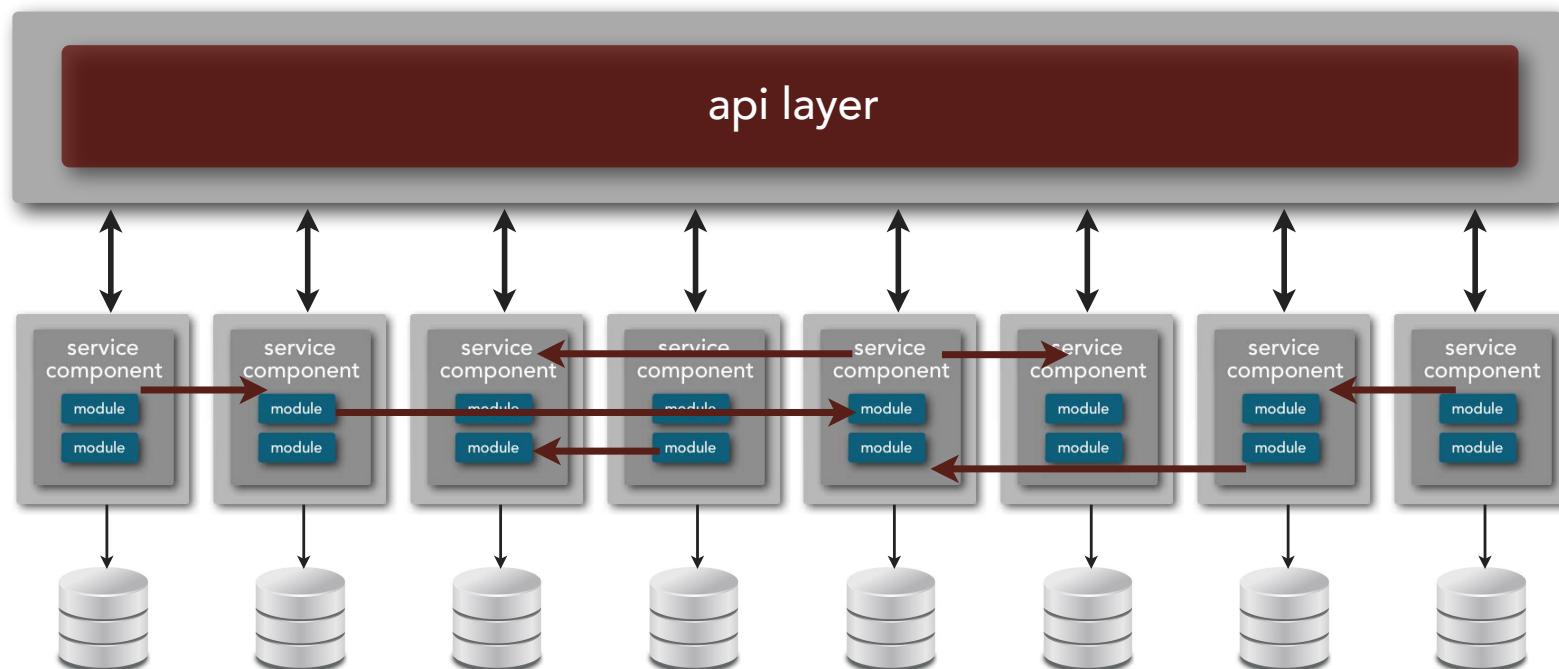
reliability and data consistency



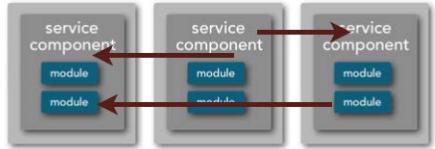
service granularity



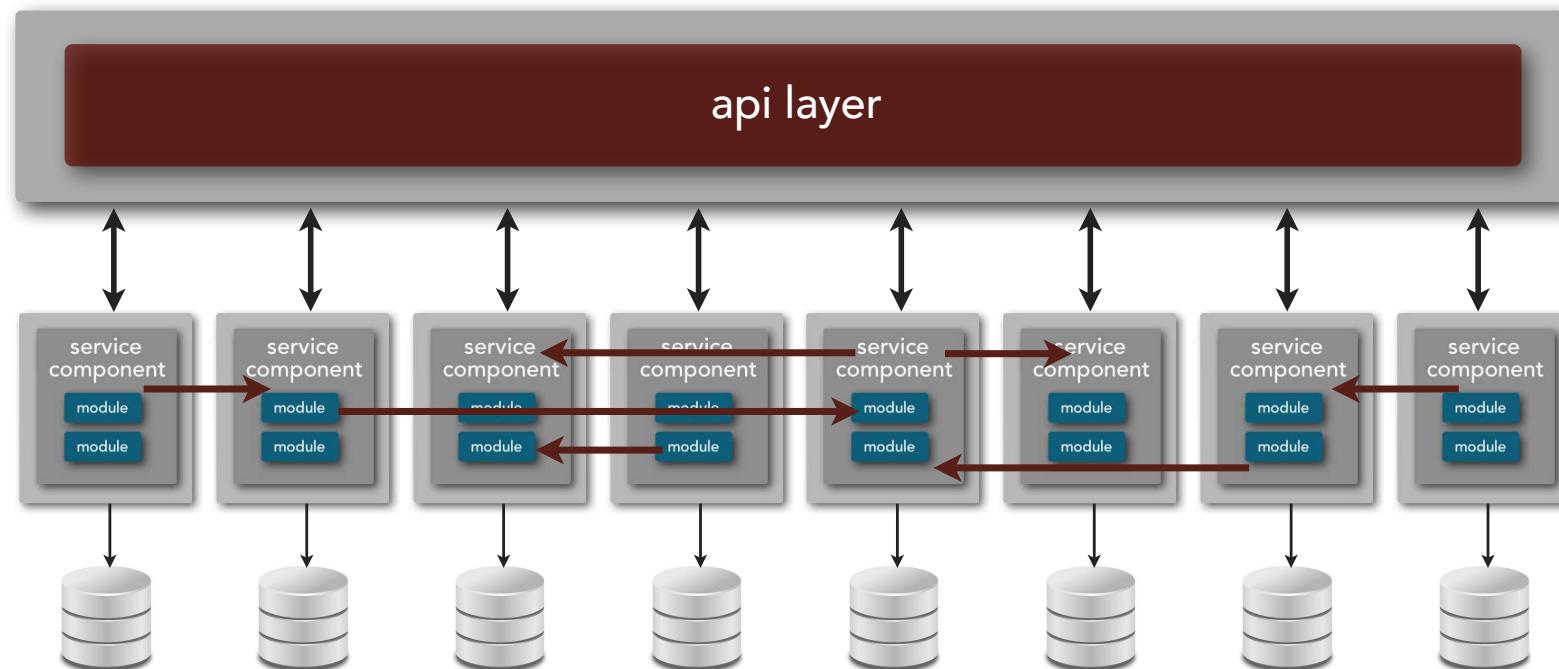
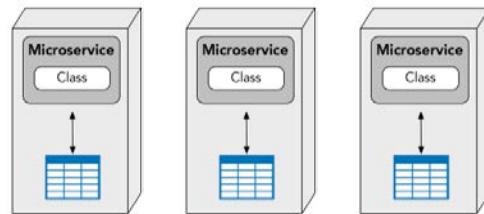
workflow and
choreography



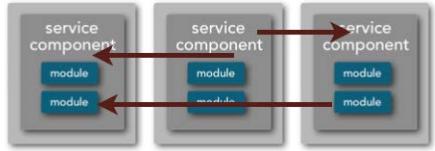
service granularity



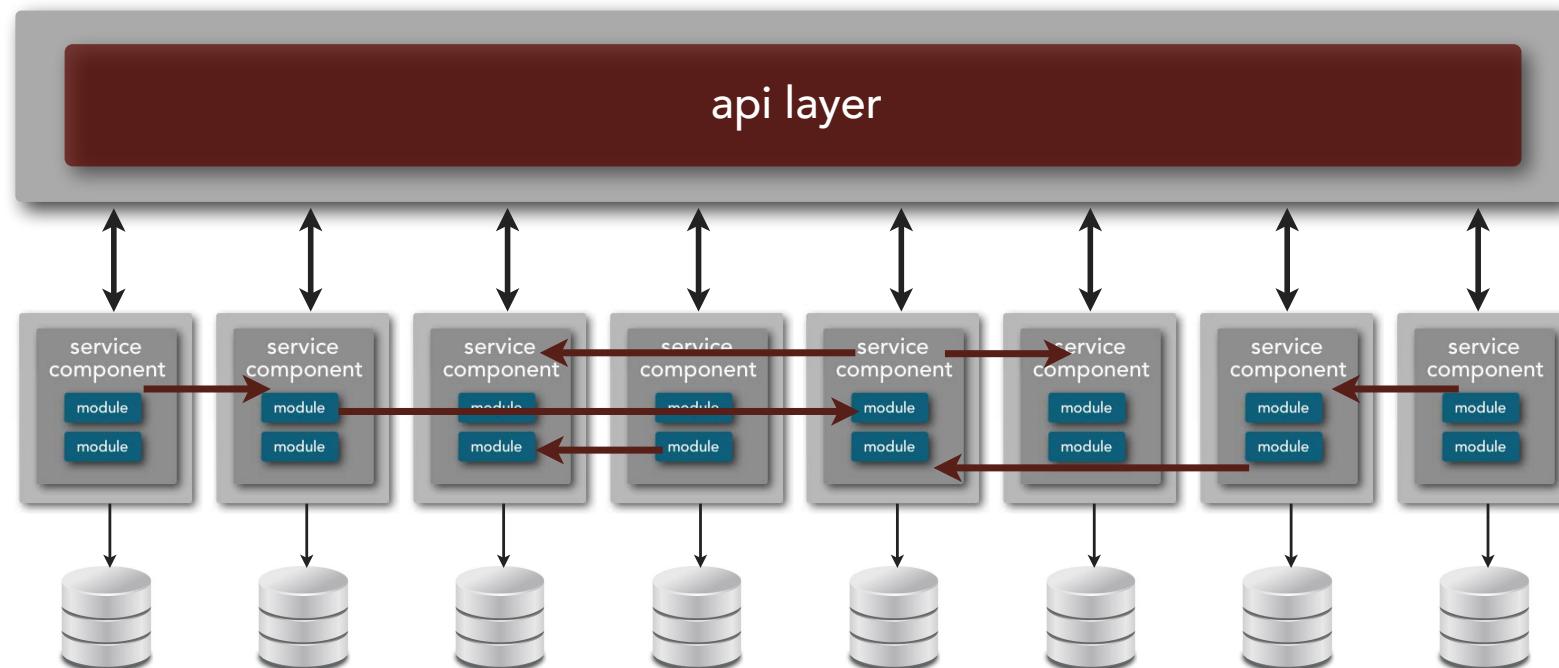
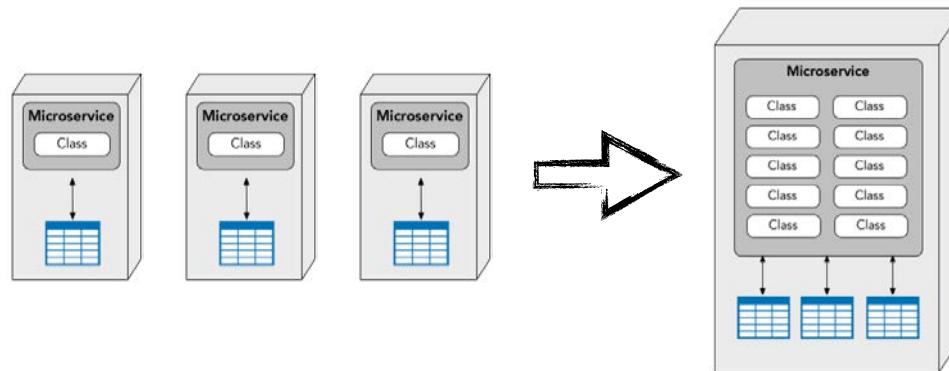
workflow and
choreography



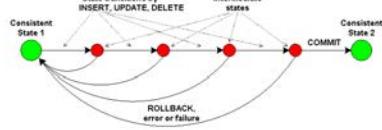
service granularity



workflow and
choreography



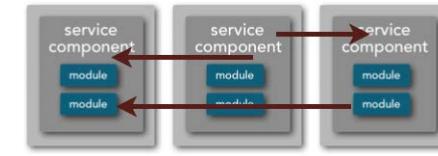
service granularity



database
transactions



data
dependencies



workflow and
choreography



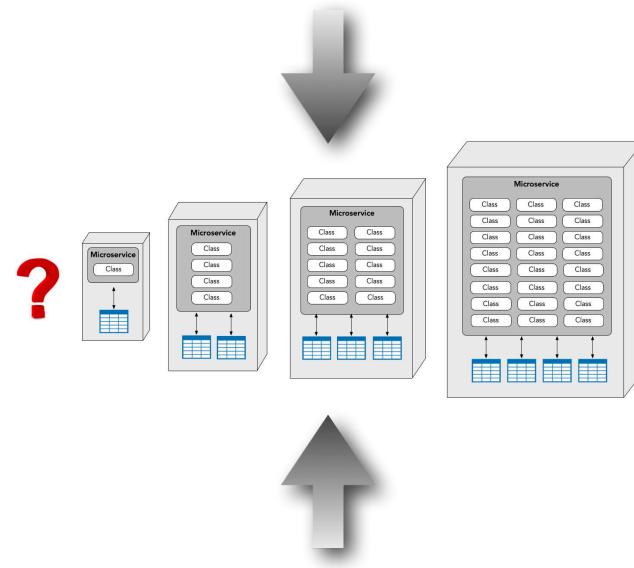
granularity integrators

"when should I consider putting services back together?"

service granularity

granularity disintegrators

“when should I consider breaking apart a service?”

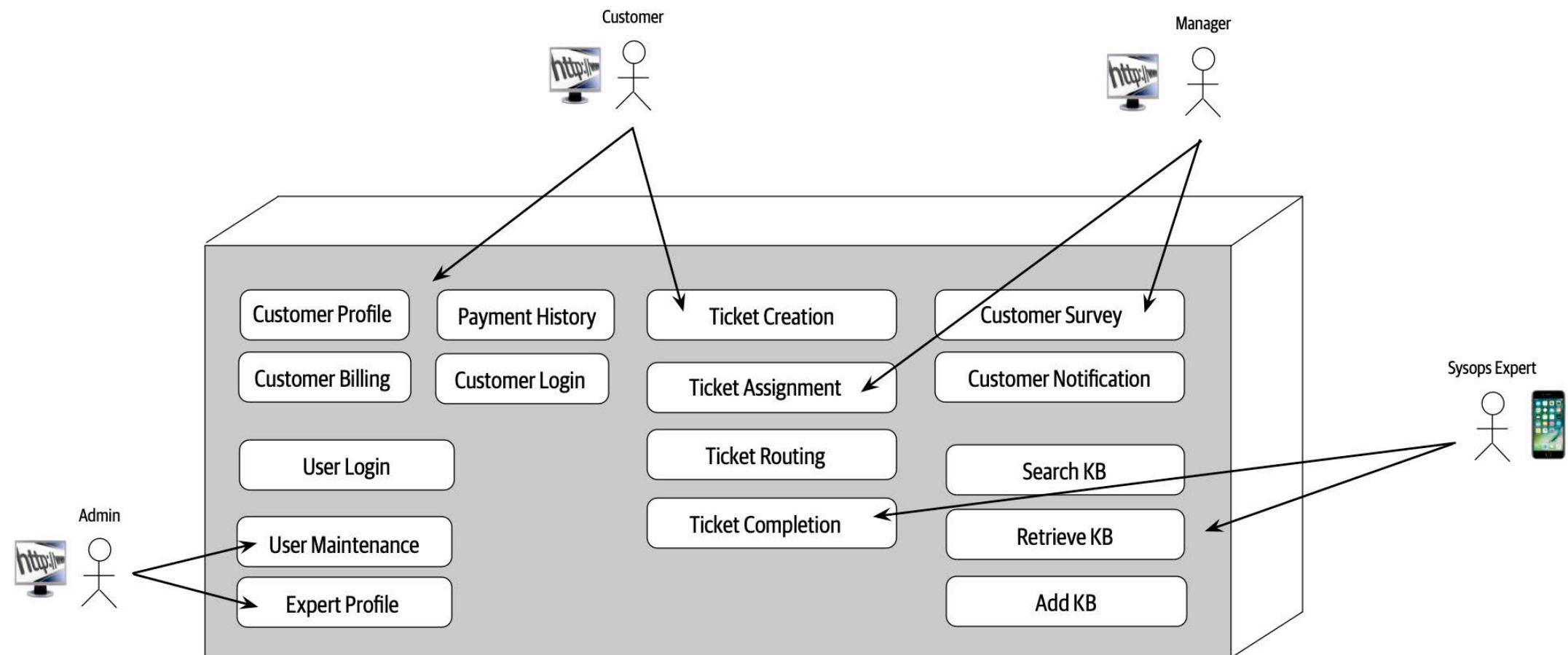


granularity integrators

“when should I consider putting services back together?”

Kata Exercise - Service Identification

Management has given the go-ahead for moving to a distributed architecture. Based on the components illustrated below and what you know about service granularity, your job now is to identify the initial service candidates.





Domain vs. Technical Partitioning

Your Architectural Kata is...

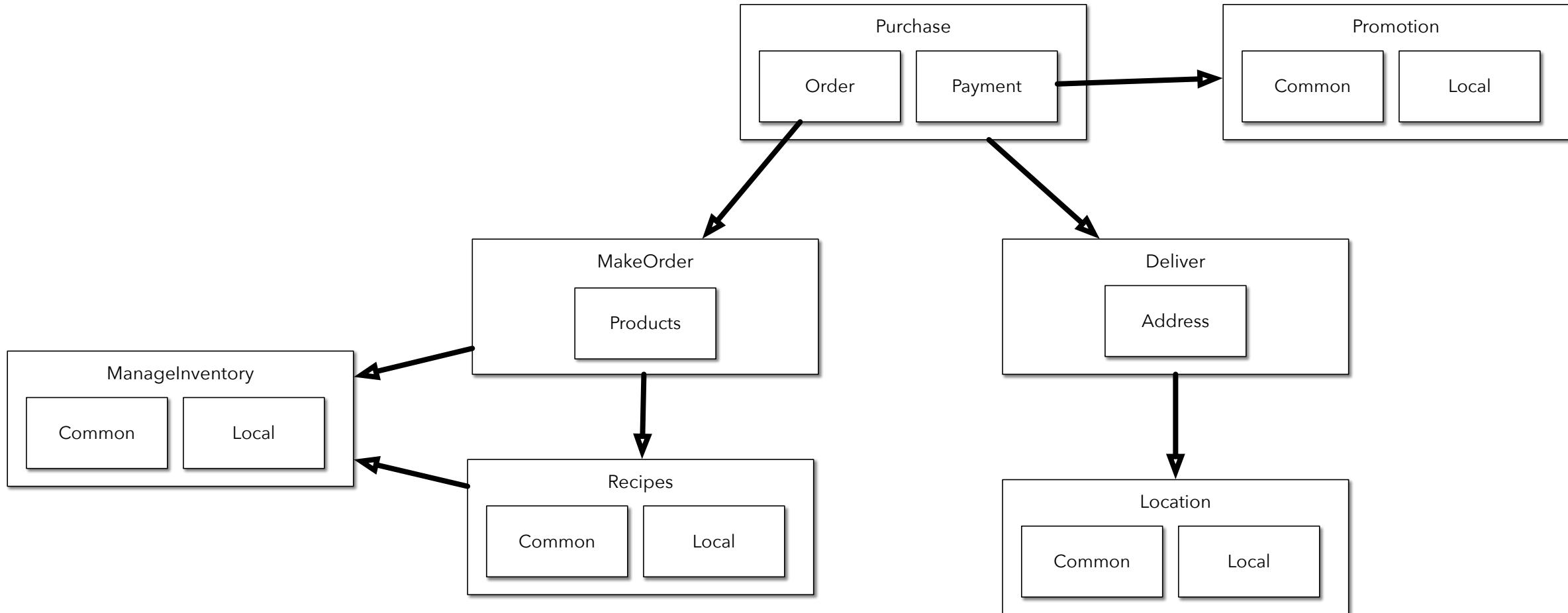
Silicon Sandwiches

A national sandwich shop wants to enable Internet ordering (in addition to their current call-in service).

- **Users:** thousands, perhaps one day millions
- **Requirements:**
 - users will place their order, then be given a time to pick up their sandwich and directions to the shop (which must integrate with several external mapping services that include traffic information)
 - if the shop offers a delivery service, dispatch the driver with the sandwich to the user
 - mobile-device accessibility
 - offer national daily promotion/specials
 - offer local daily promotion/specials
 - accept payment online or in person/on delivery
- **Additional Context:**
 - Sandwich shops are franchised, each with a different owner.
 - Parent company has near-future plans to expand overseas.
 - Corporate goal is to hire inexpensive labor to maximize profit.

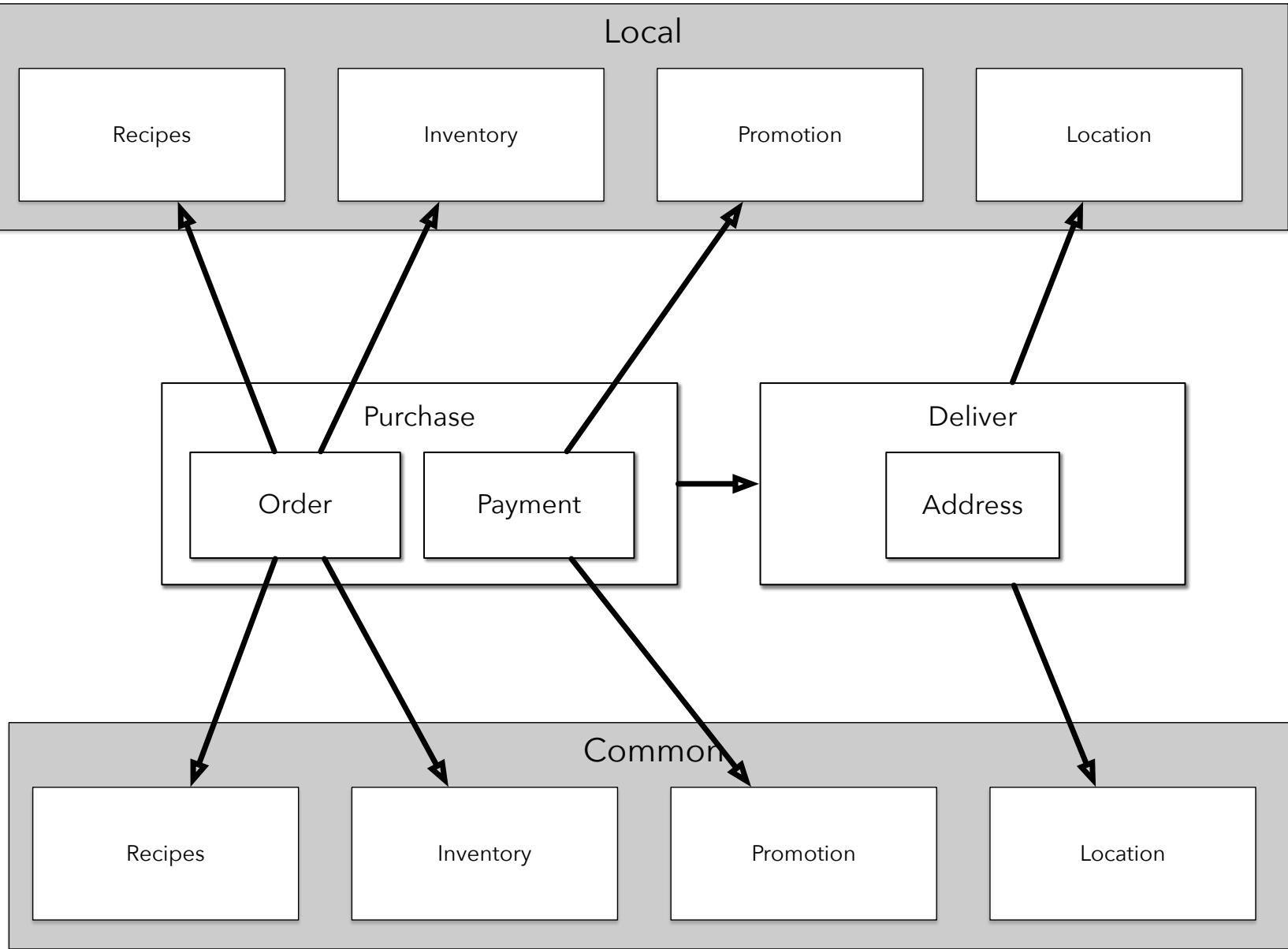
Your Architectural Kata is...

Silicon Sandwiches



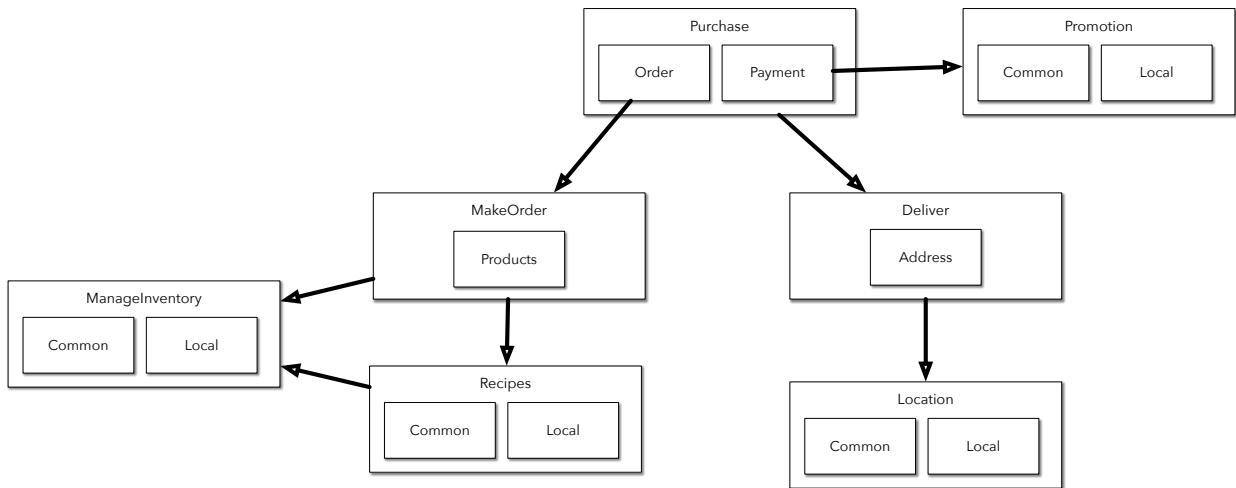
Your Architectural Kata is...

Silicon Sandwiches

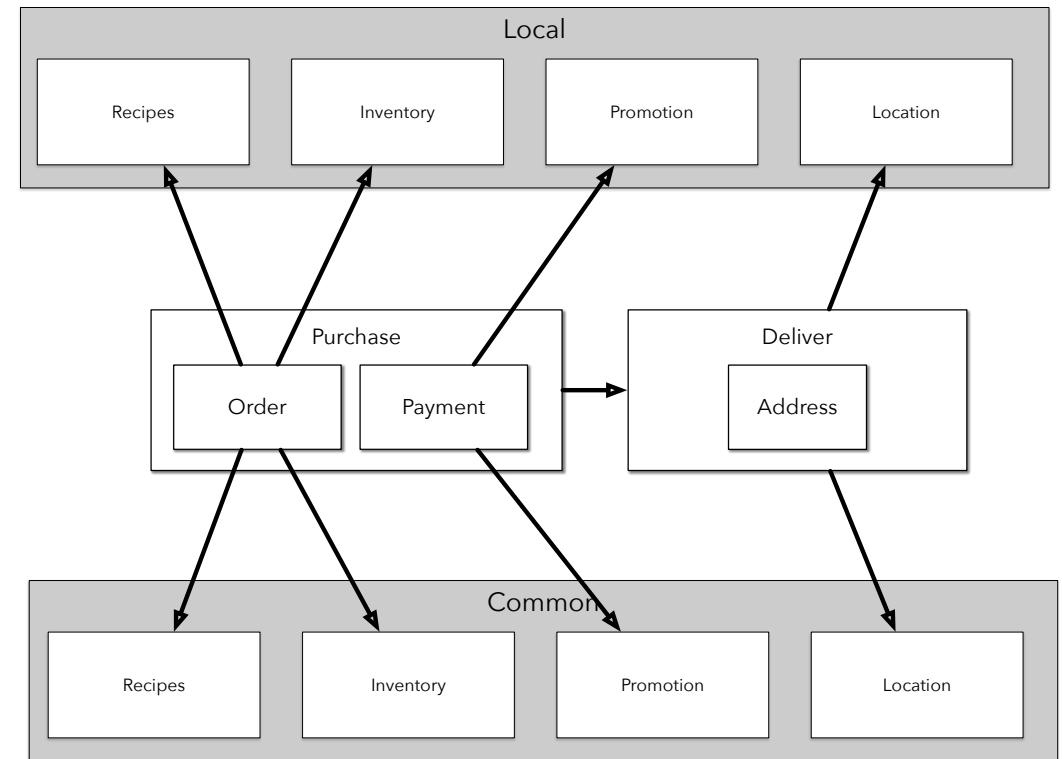


Your Architectural Kata is...

Silicon Sandwiches

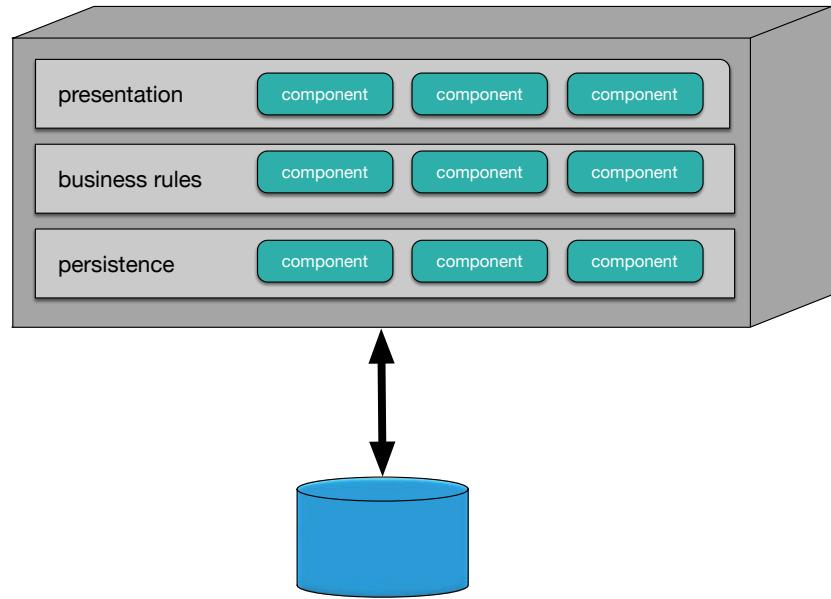


domain partitioned



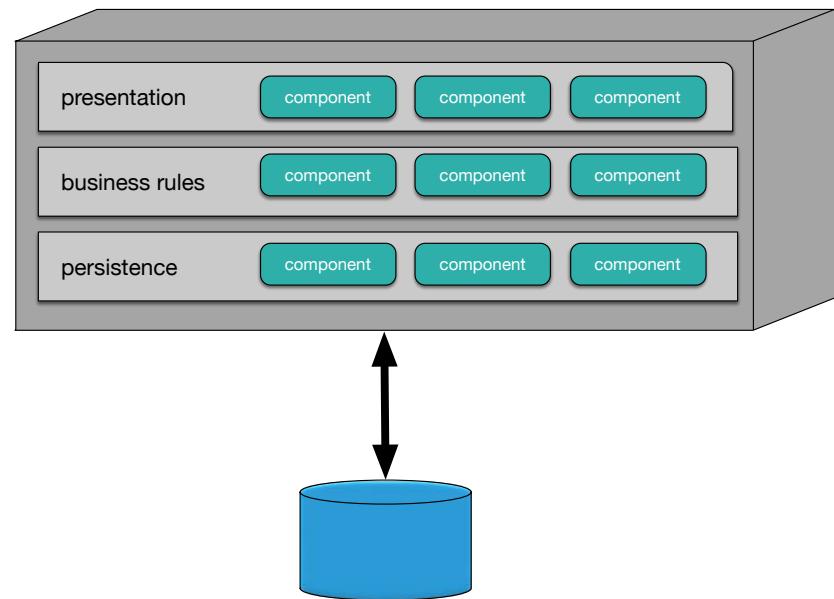
technically partitioned

top level partitioning



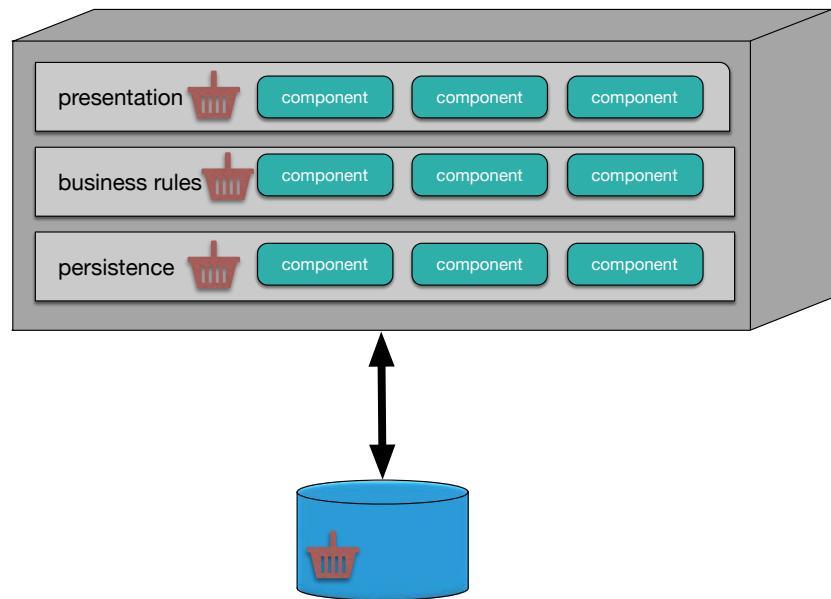
top level partitioning

technical partitioning



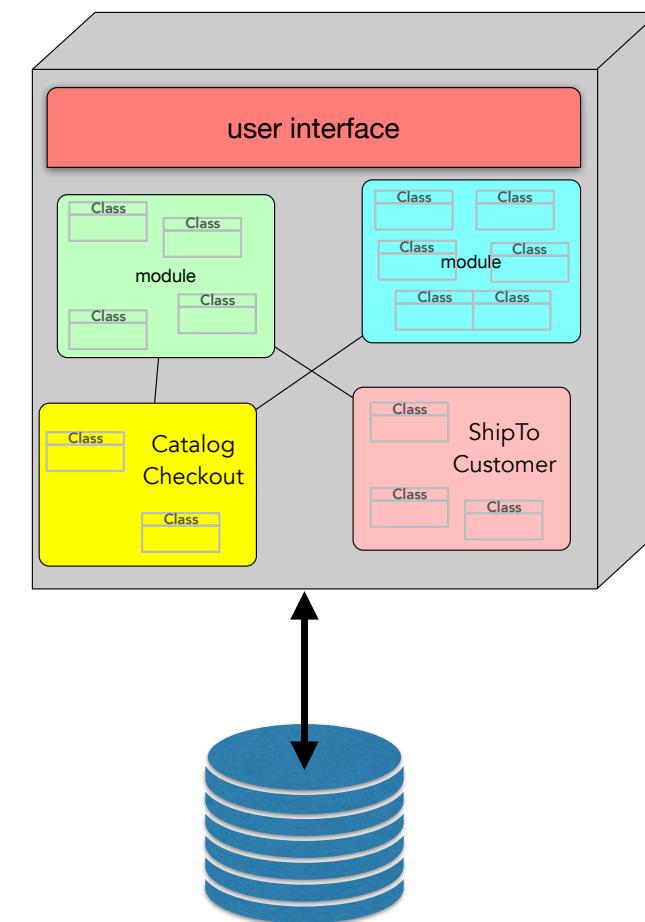
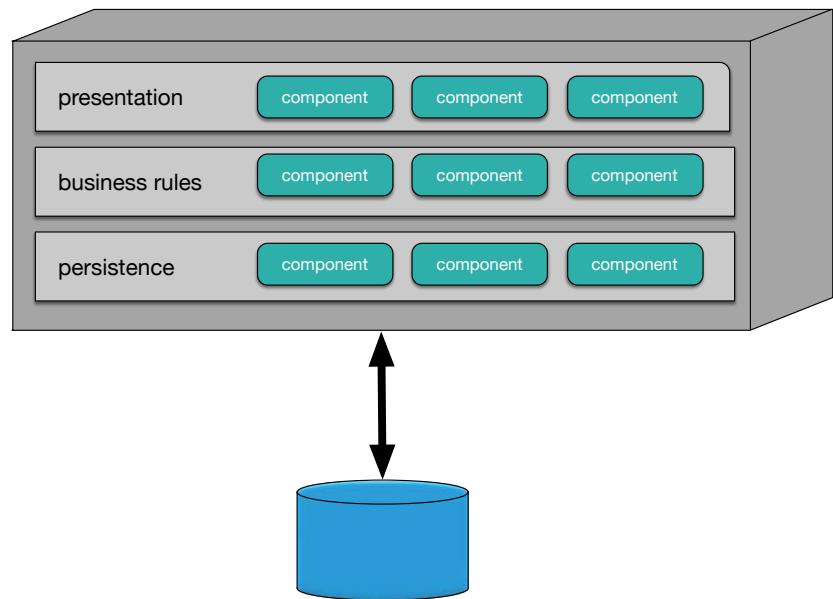
top level partitioning

technical partitioning



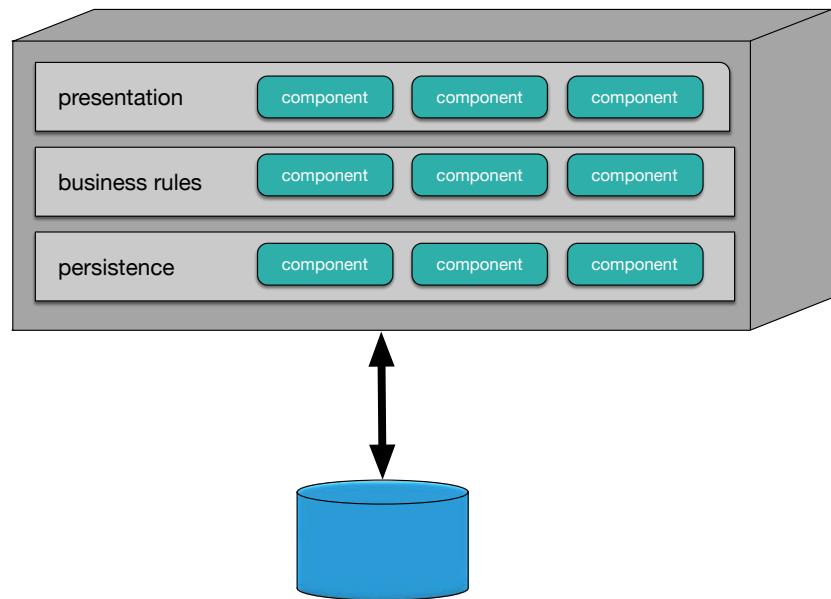
top level partitioning

technical partitioning

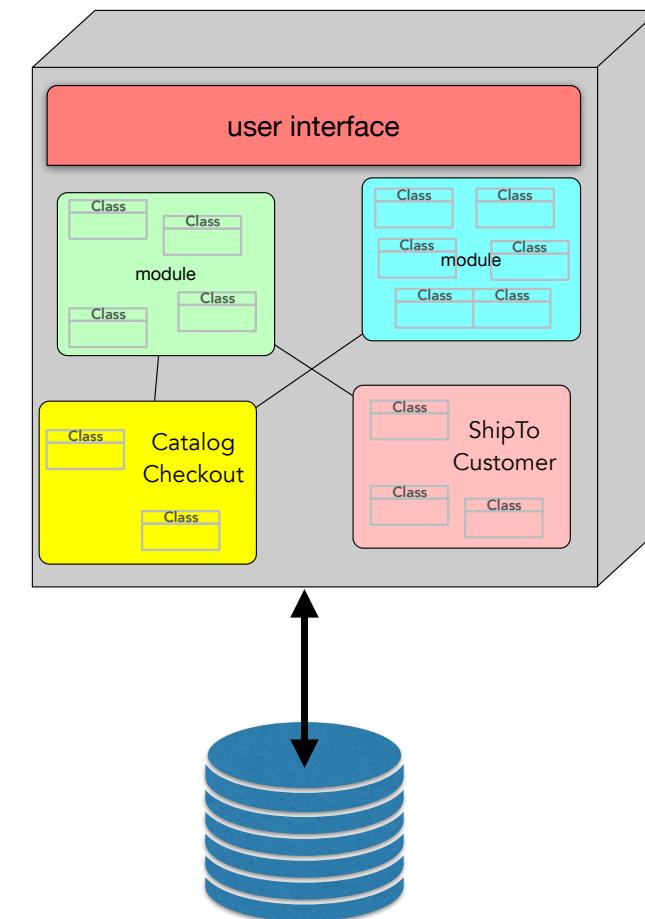


top level partitioning

technical partitioning

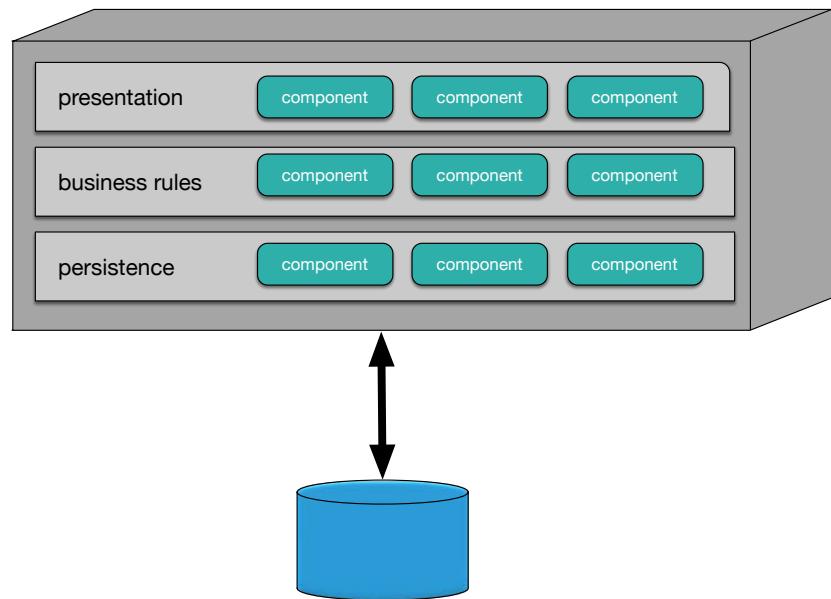


domain partitioning

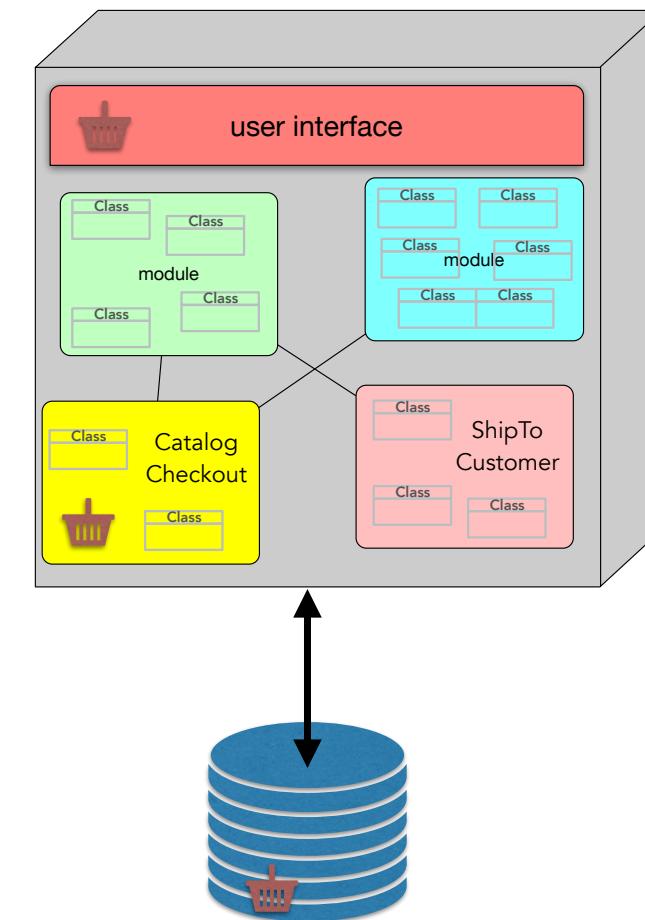


top level partitioning

technical partitioning

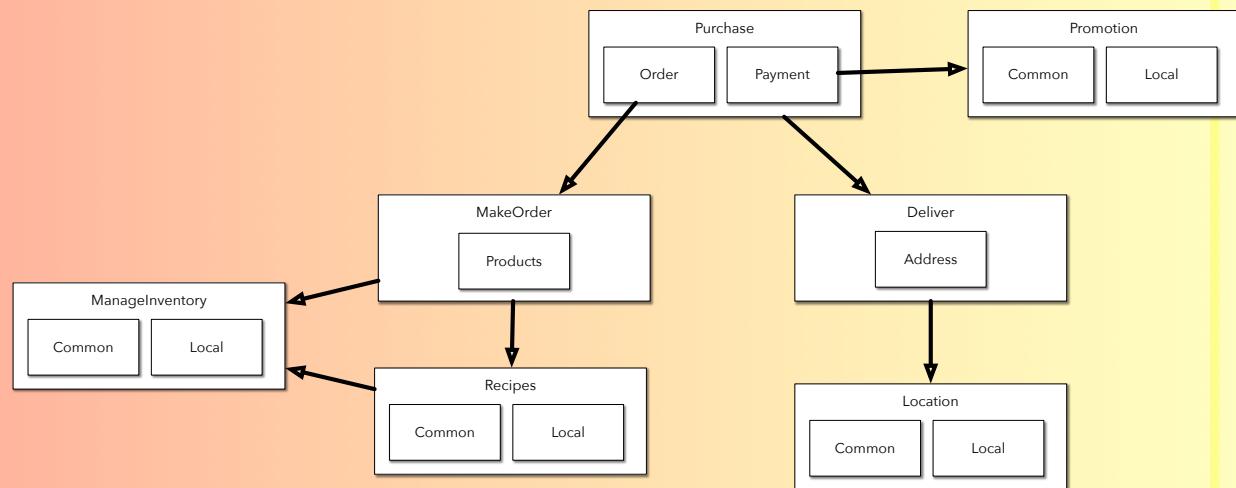


domain partitioning

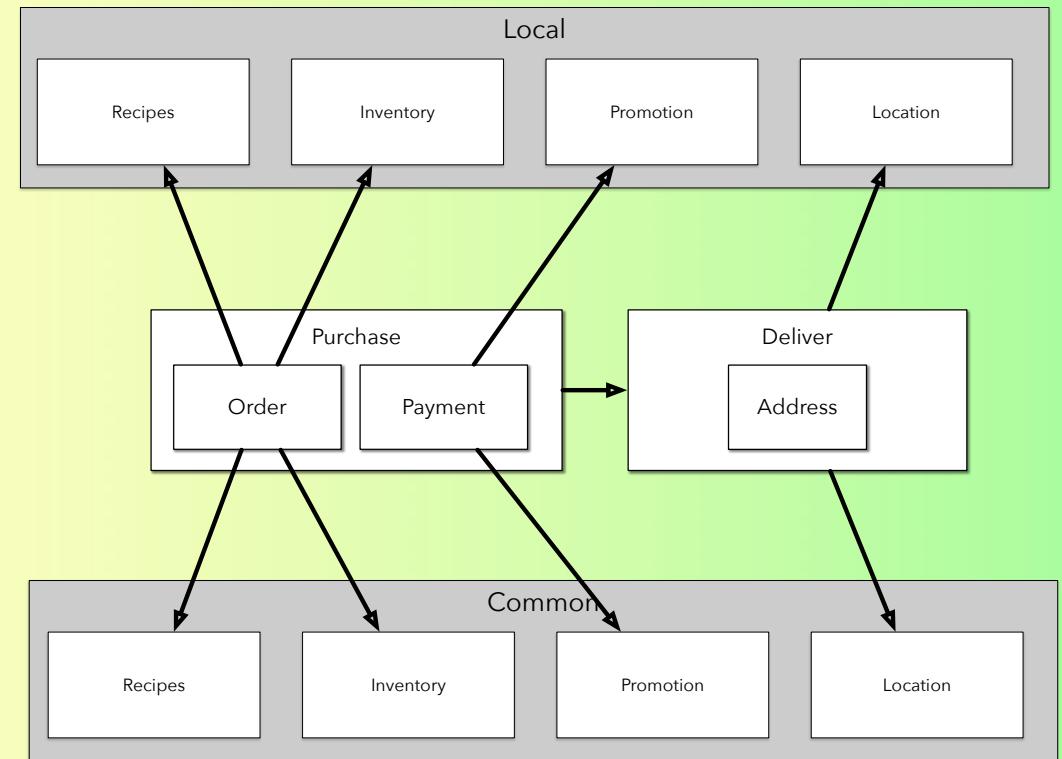


tradeoffs

domain VS technical partitioning



domain partitioned



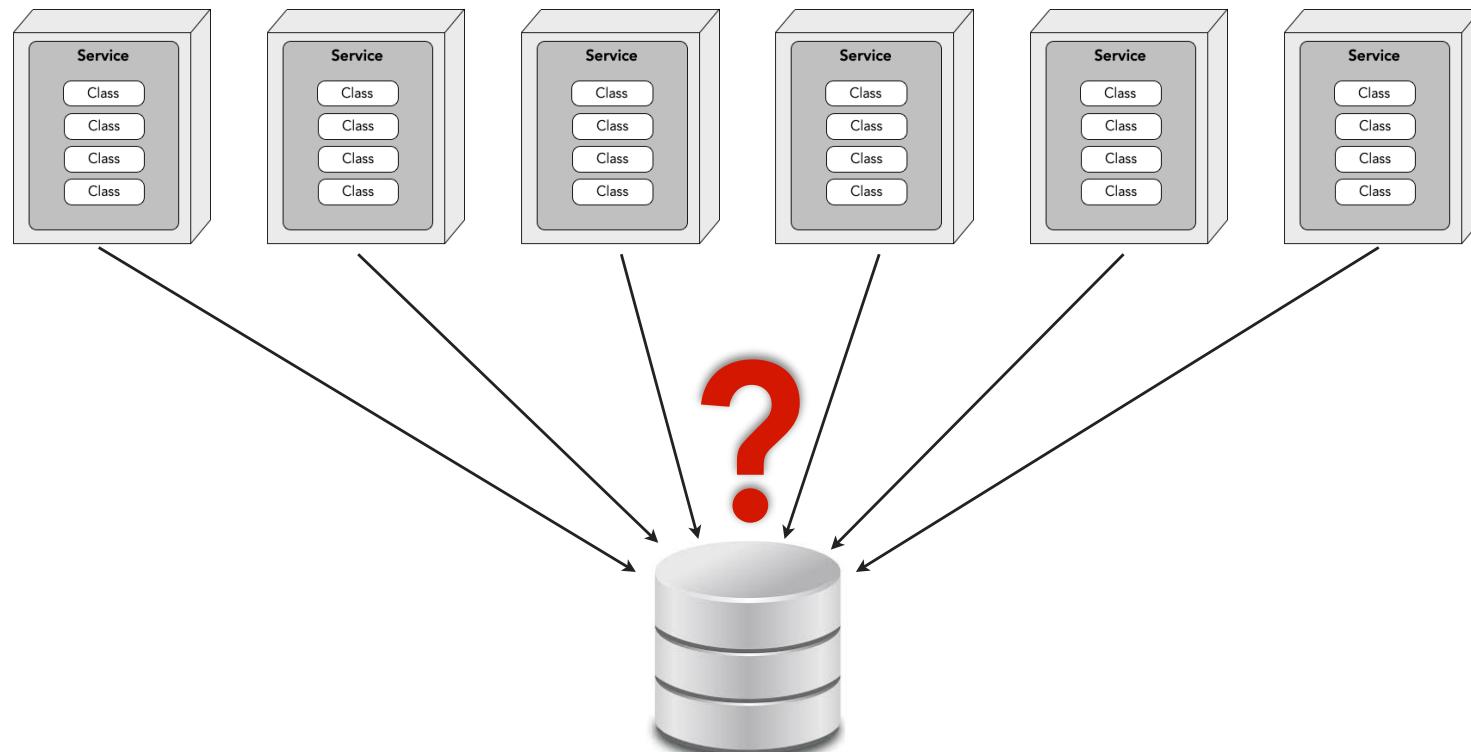
technically partitioned



Breaking Apart Transactional Data

breaking apart data

“when should I consider breaking apart my data?”



breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers

breaking apart data

“when should I consider breaking apart my data?”

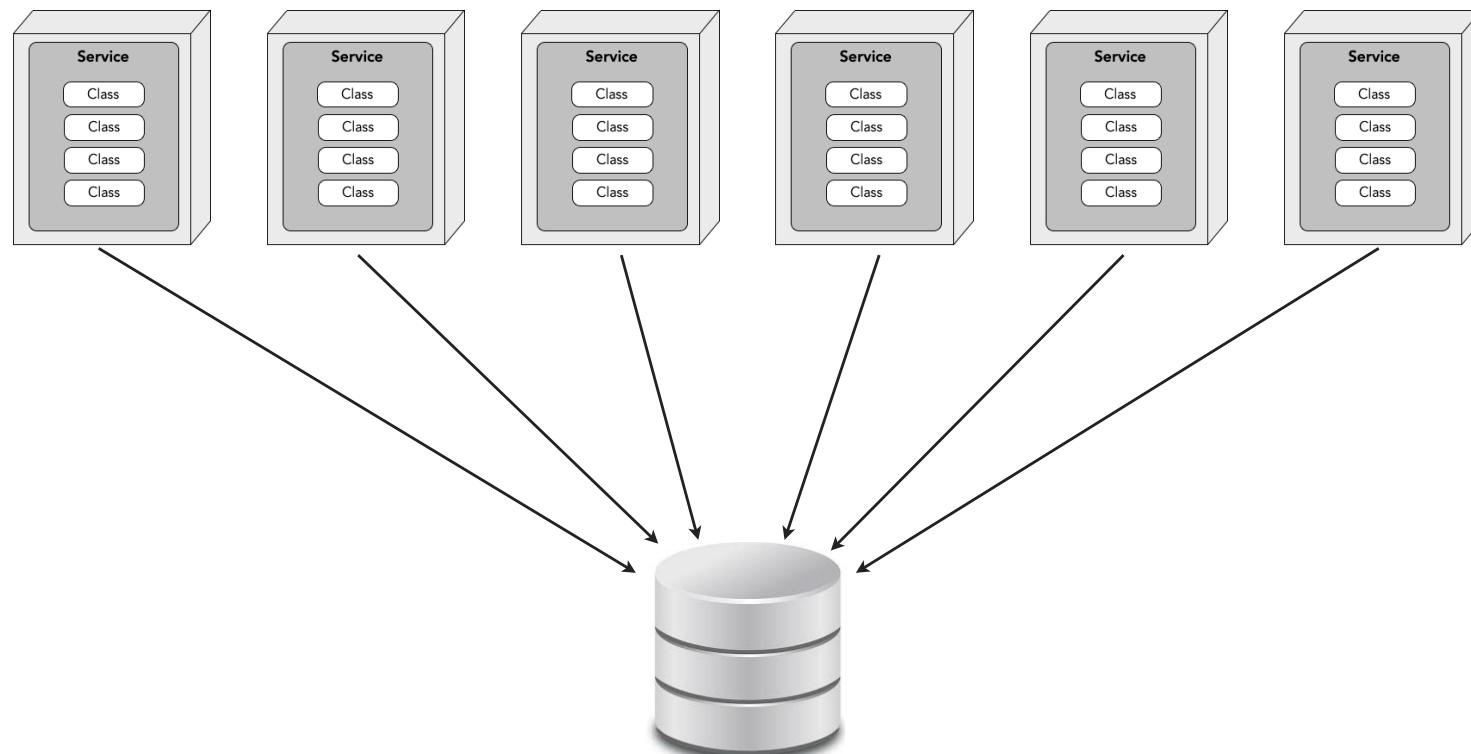
database granularity drivers



change
control

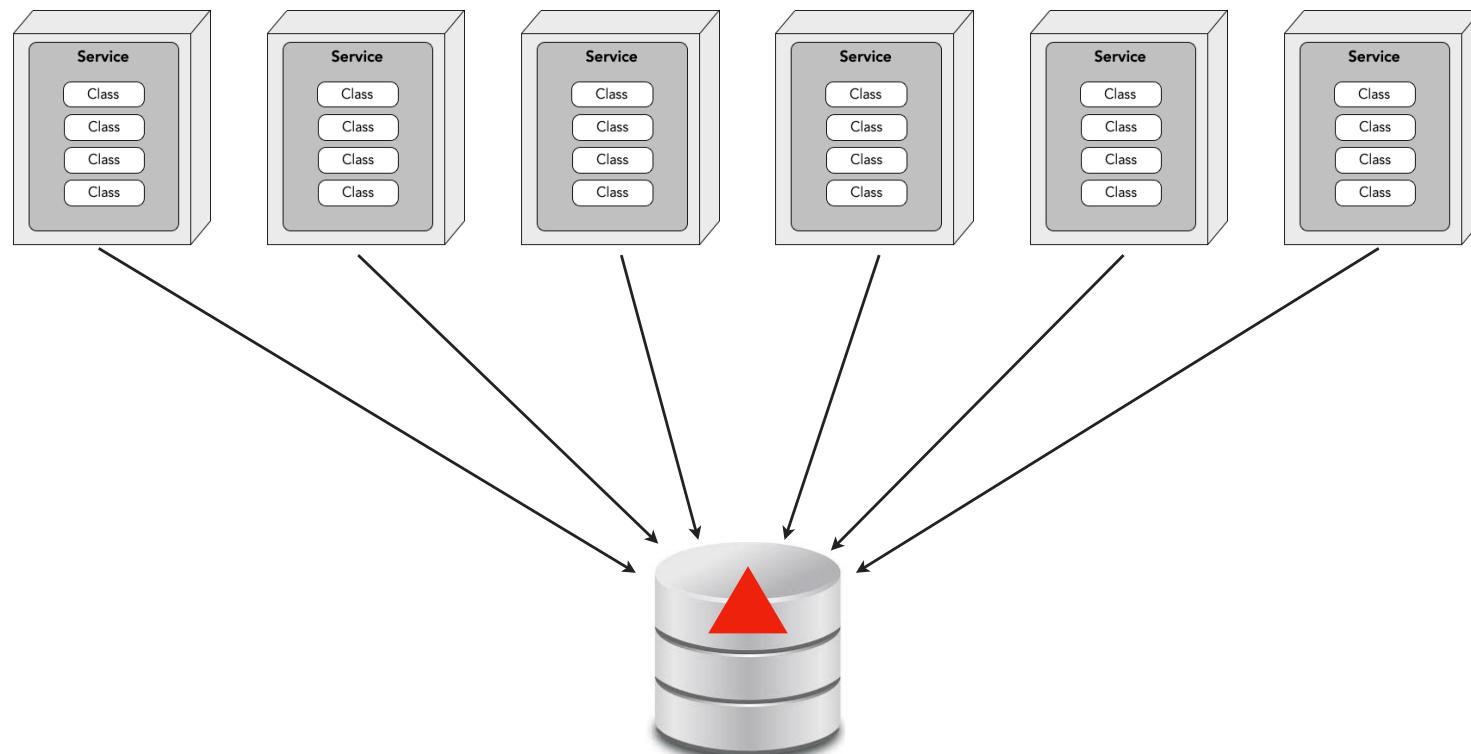
breaking apart data

change control



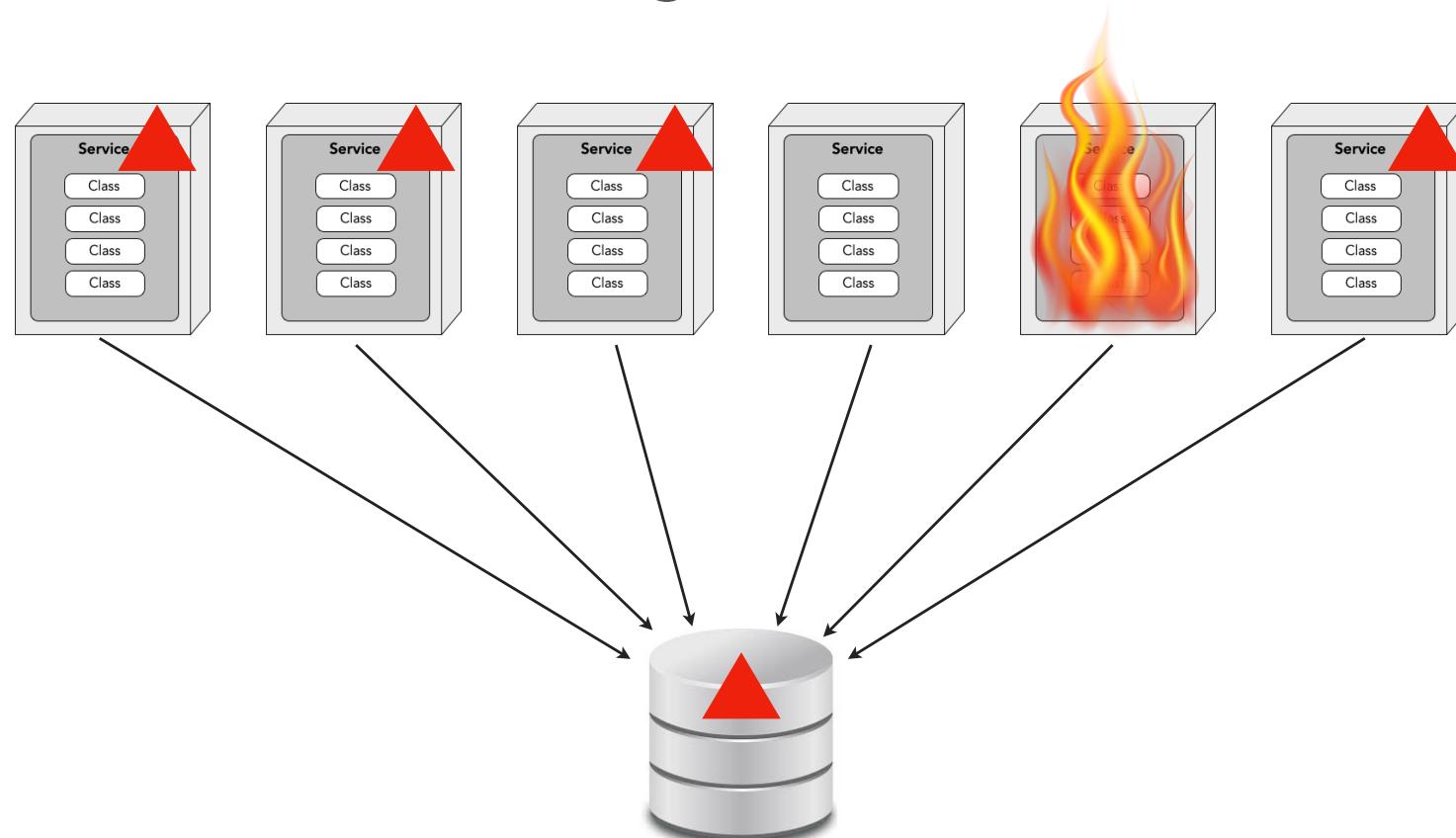
breaking apart data

change control



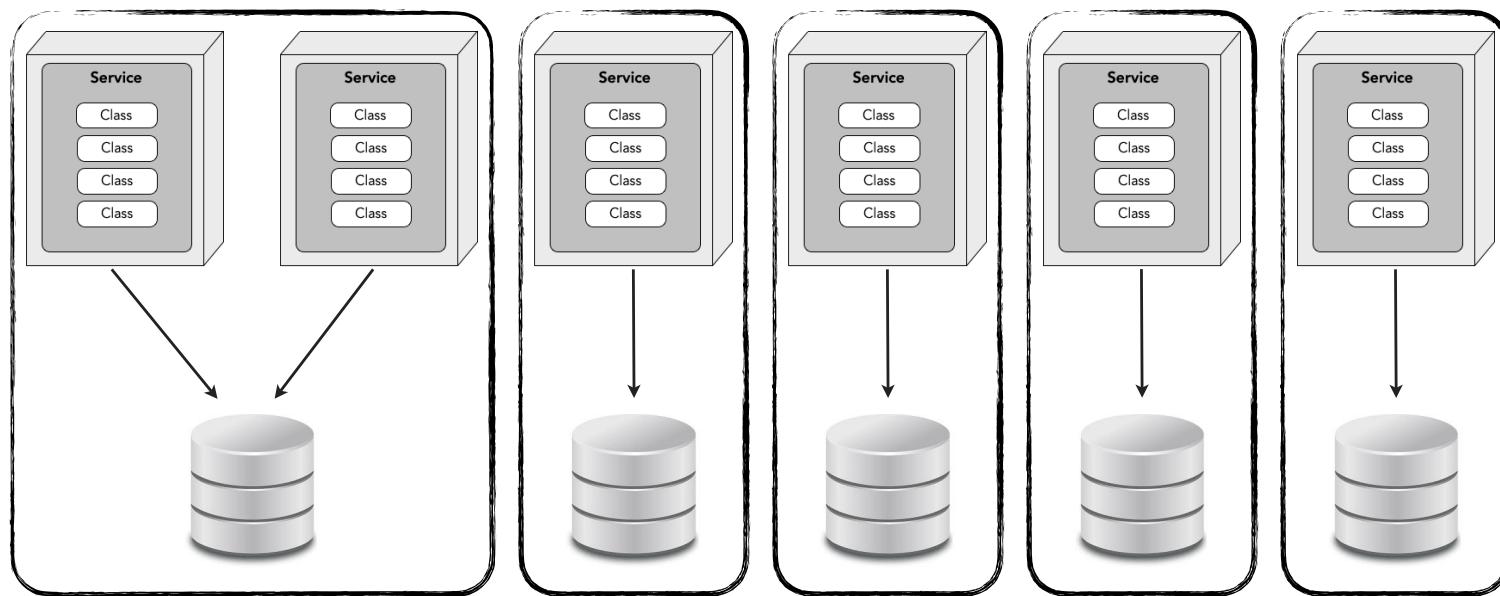
breaking apart data

change control



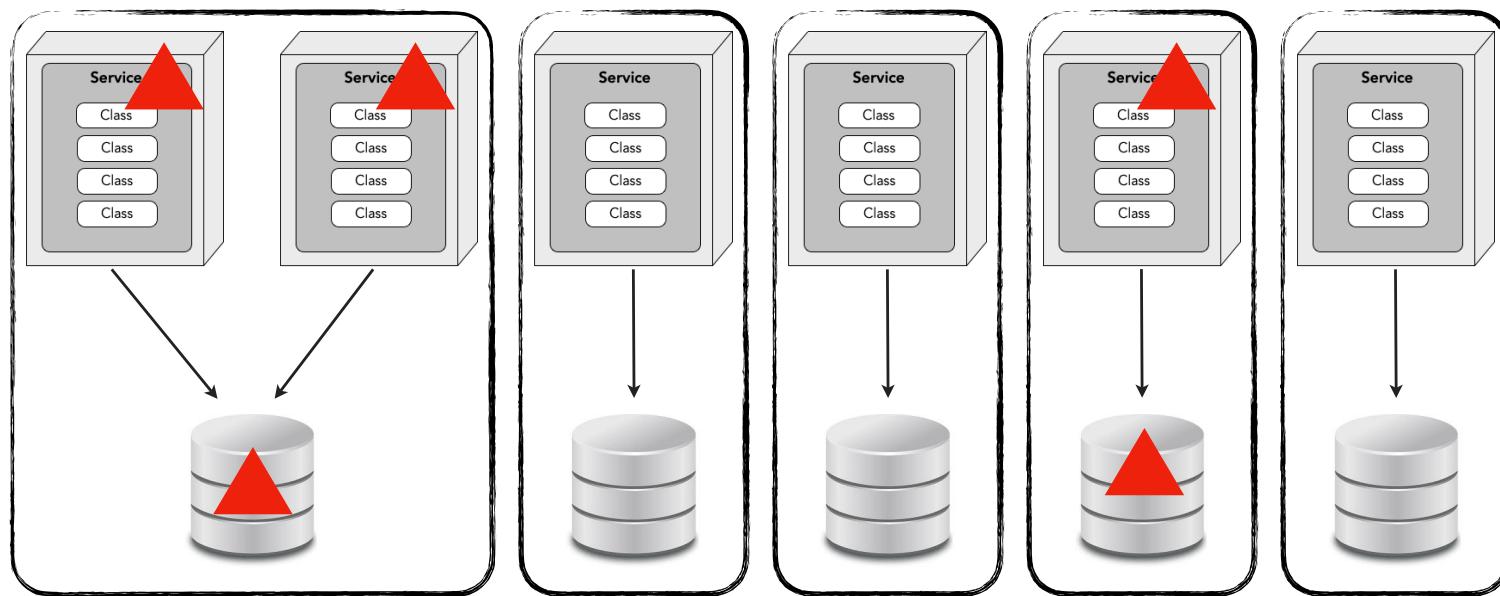
breaking apart data

change control



breaking apart data

change control



breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers



change
control

breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers



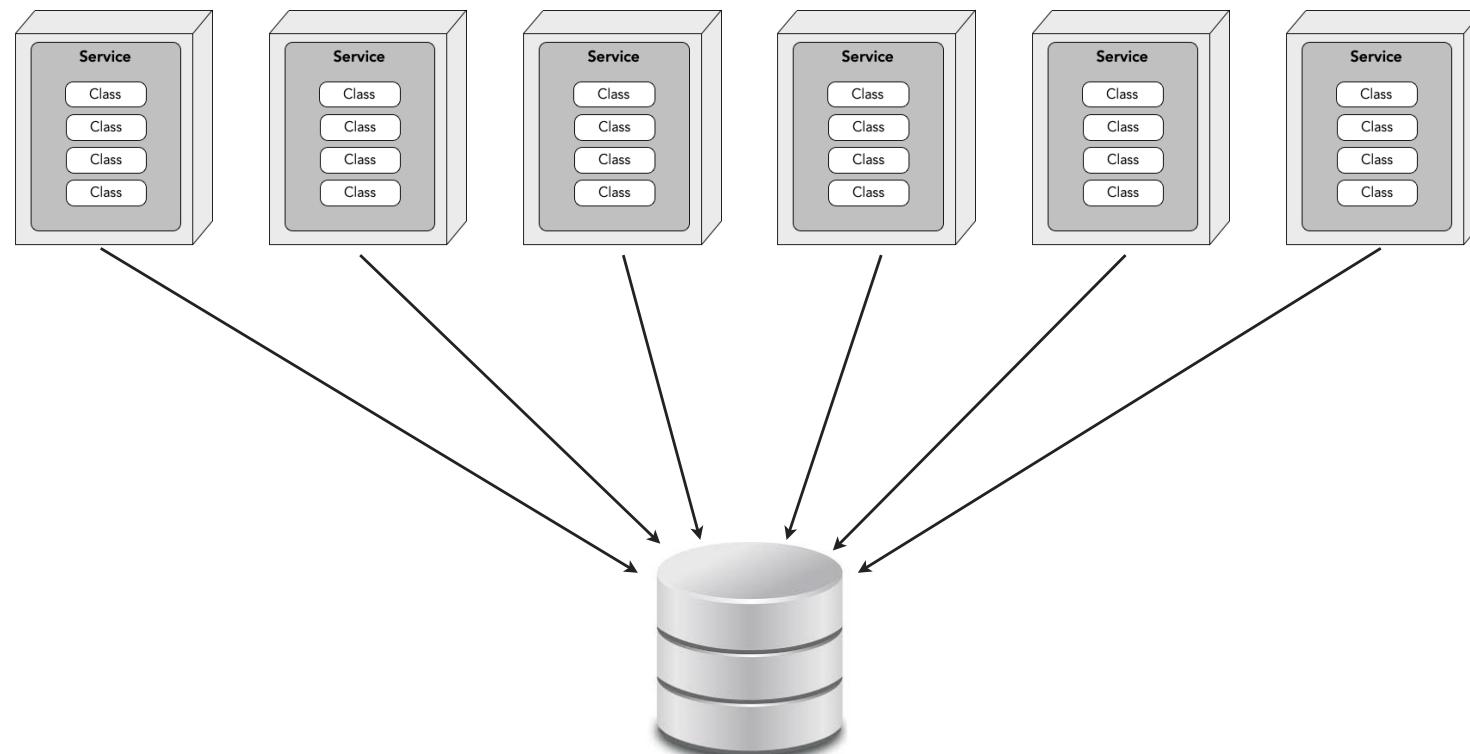
change
control



database
connections

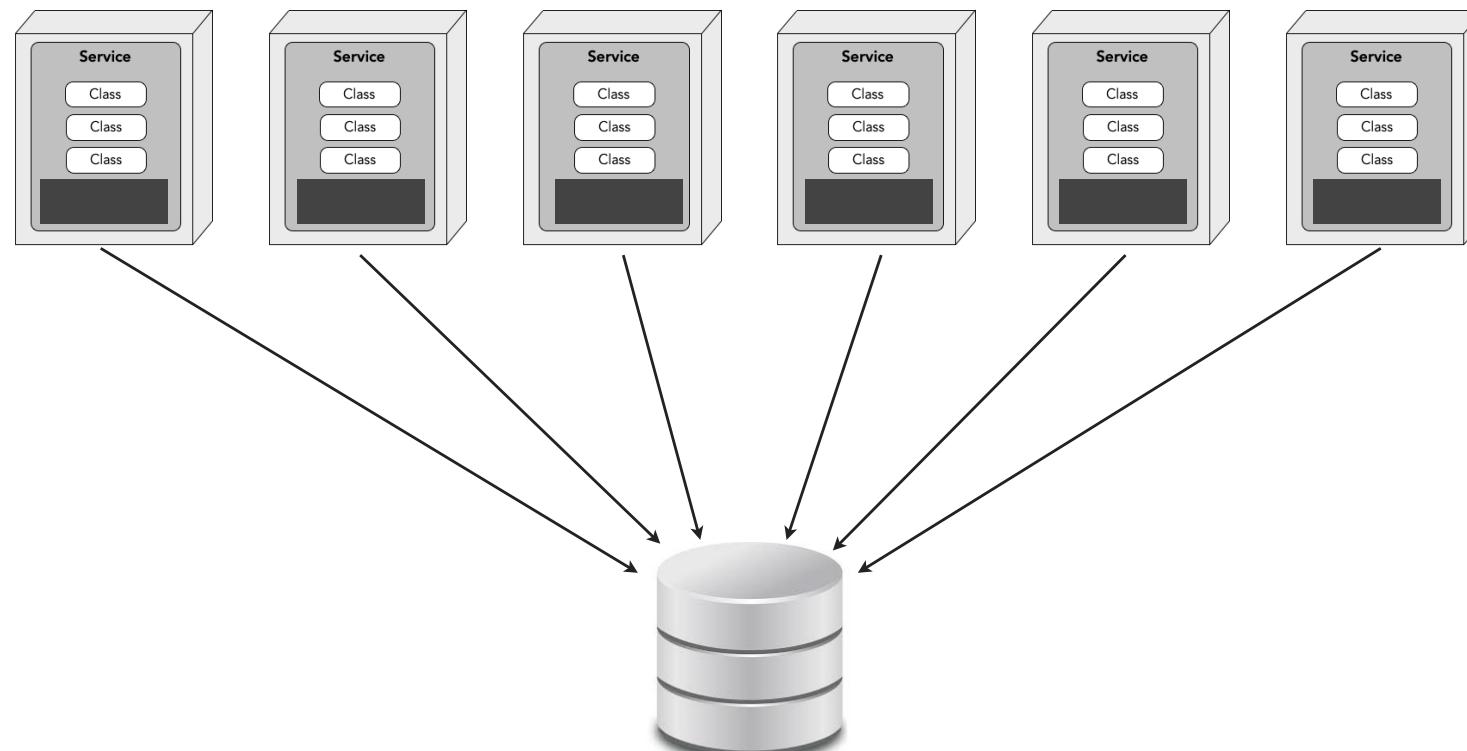
breaking apart data

database connections



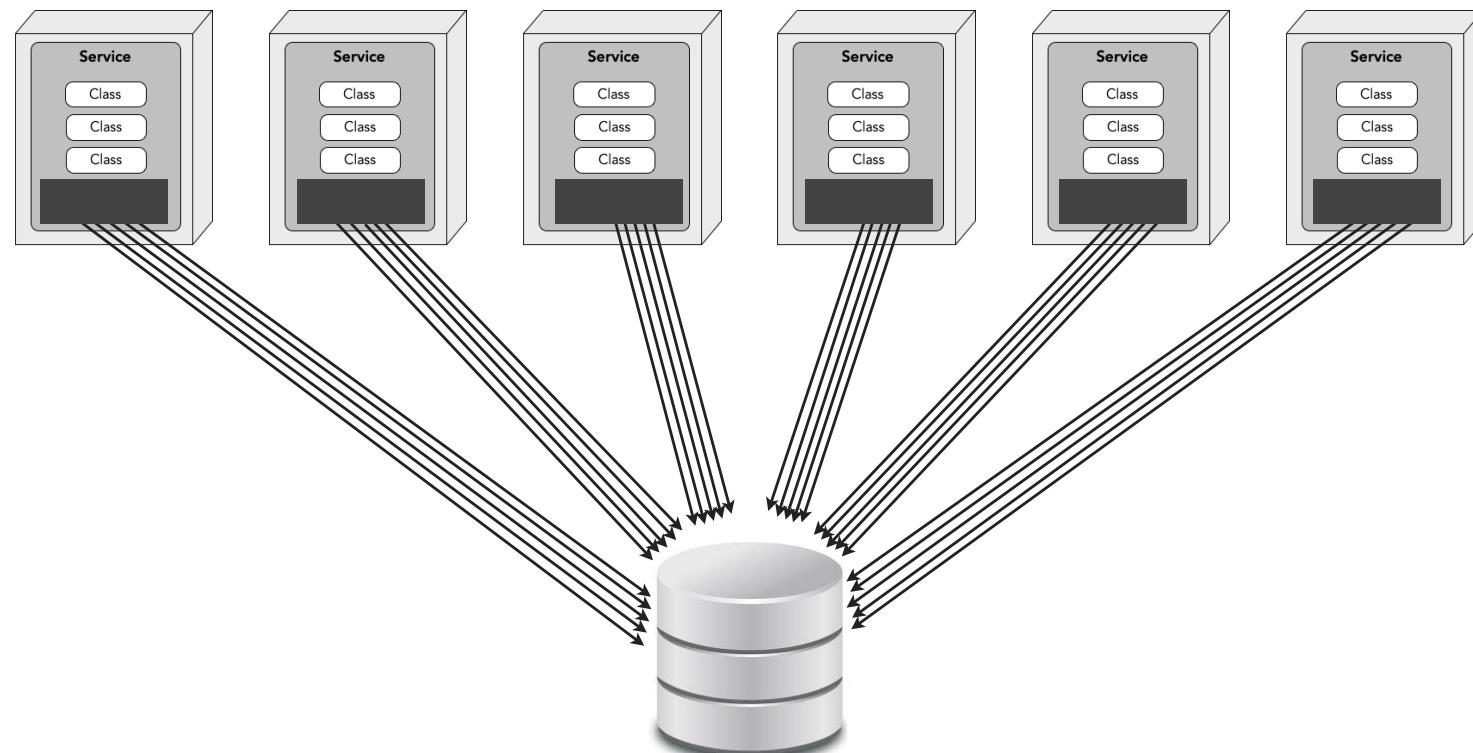
breaking apart data

database connections



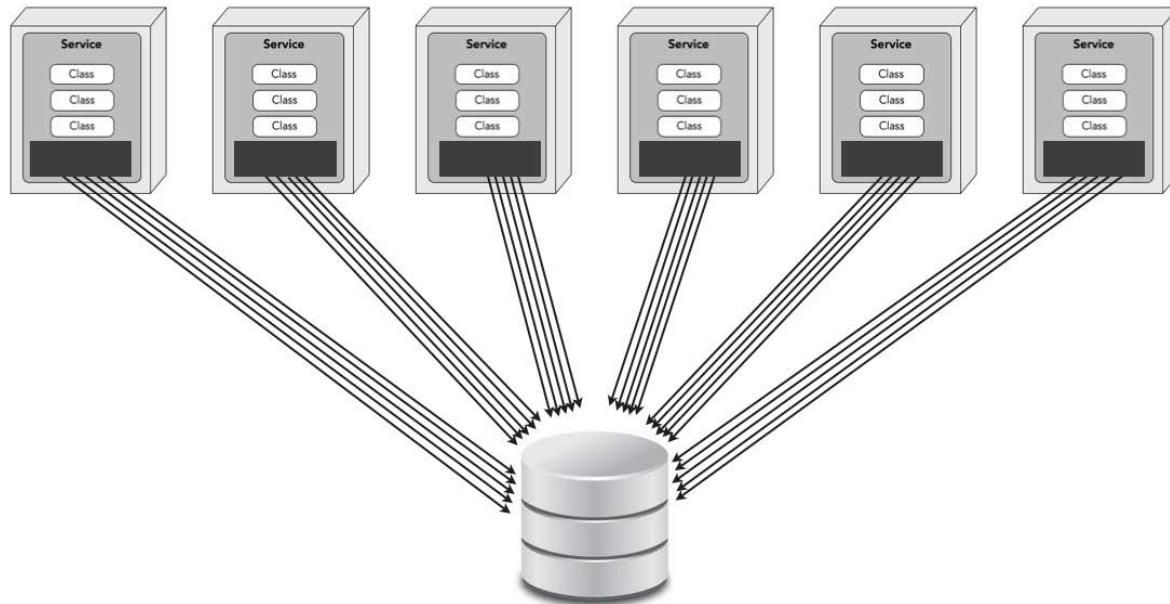
breaking apart data

database connections



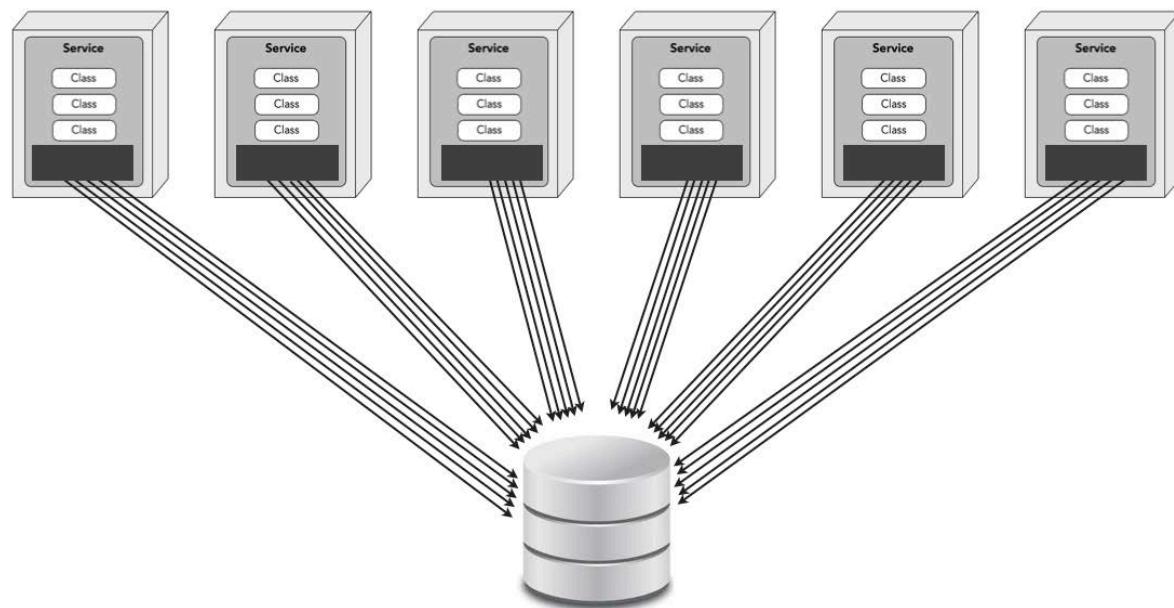
breaking apart data

database connections



breaking apart data

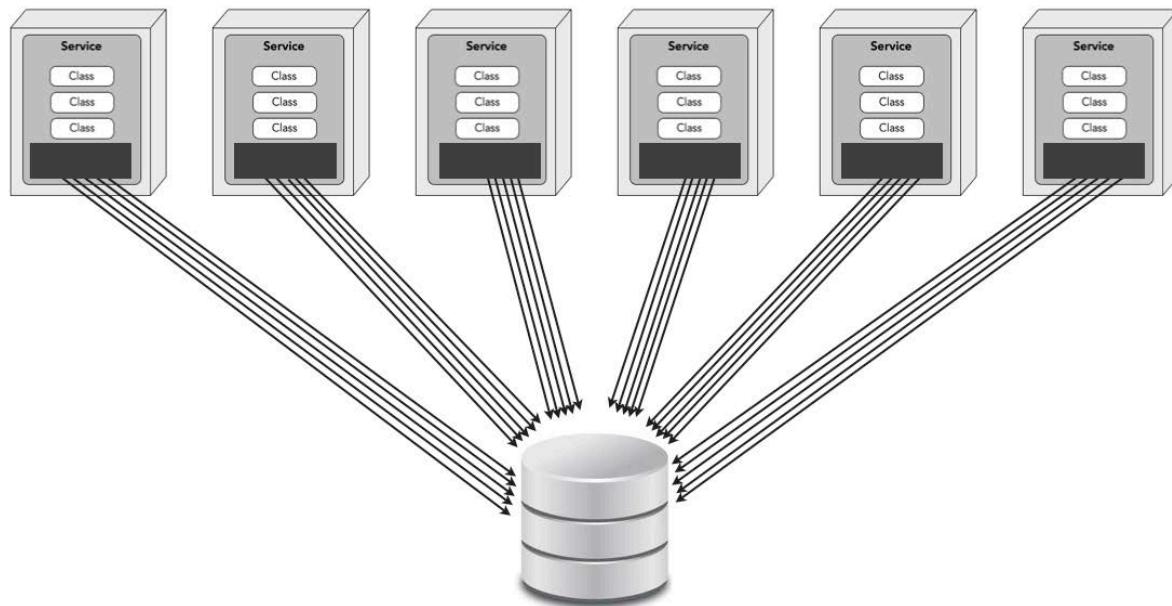
database connections



monolithic application: 200 connections

breaking apart data

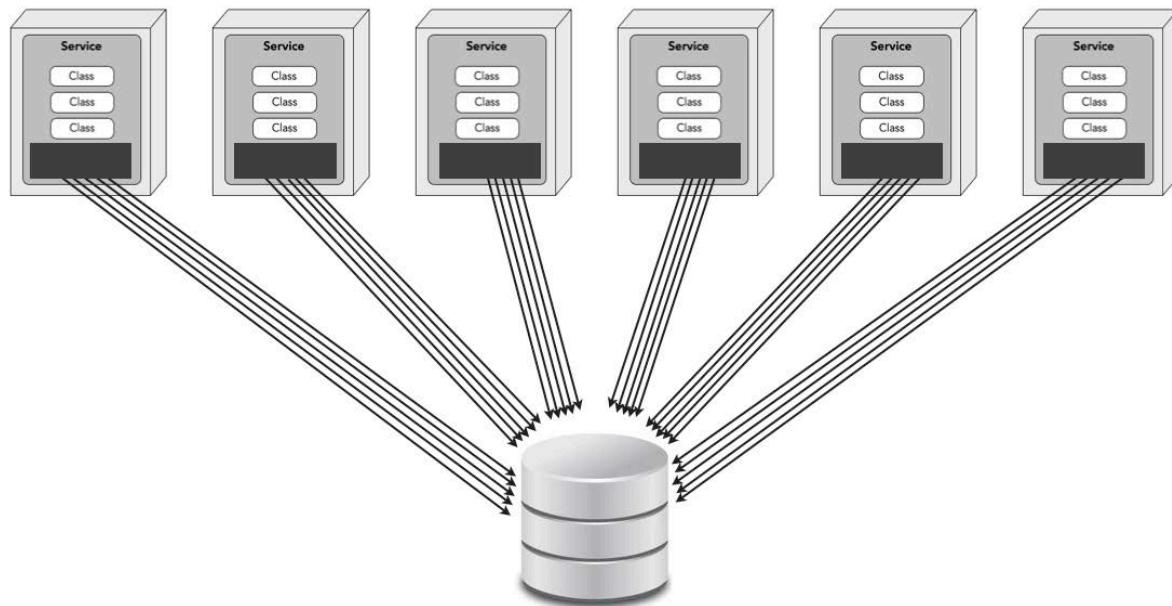
database connections



monolithic application: 200 connections
distributed services: 50

breaking apart data

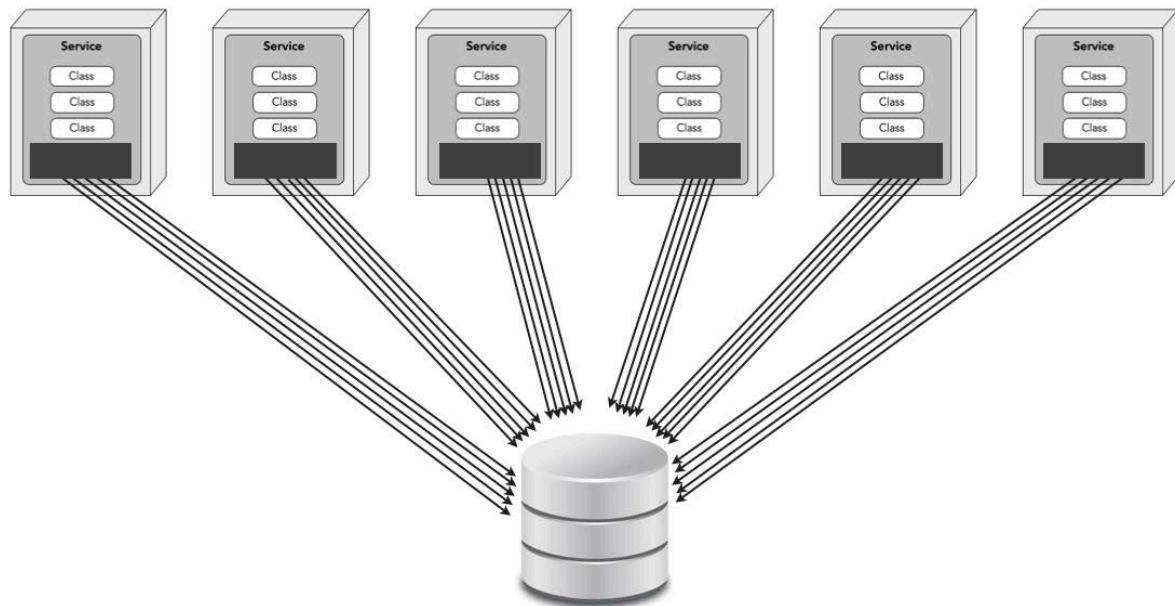
database connections



monolithic application: 200 connections
distributed services: 50
connections per service: 10

breaking apart data

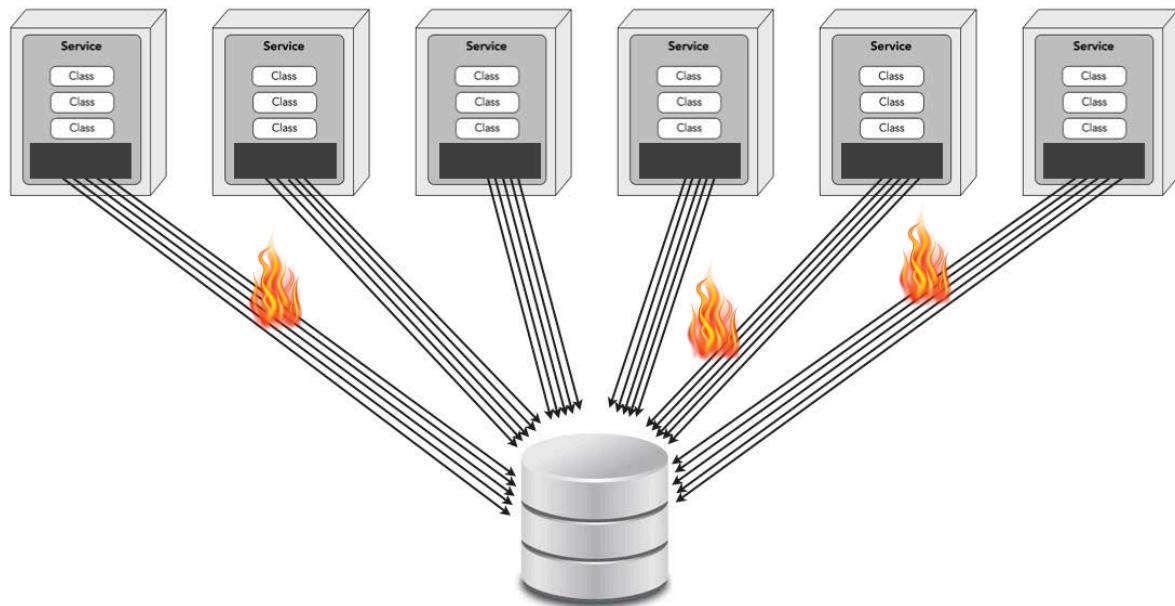
database connections



monolithic application: 200 connections
distributed services: 50
connections per service: 10
service instances (min): 2

breaking apart data

database connections



monolithic application: 200 connections

distributed services: 50

connections per service: 10

service instances (min): 2

distributed connections: 1000 connections

breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers



change
control



database
connections

breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers



change
control



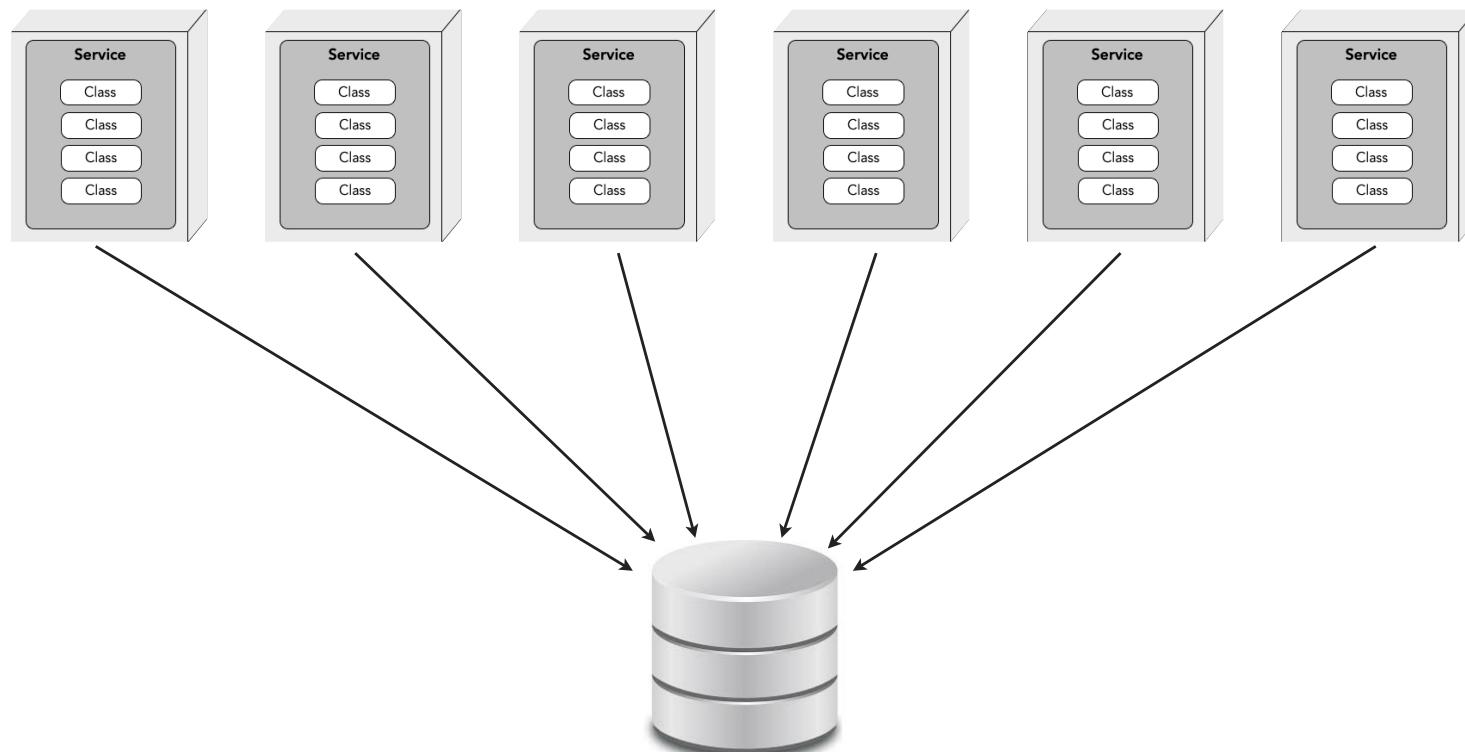
database
connections



database
scalability

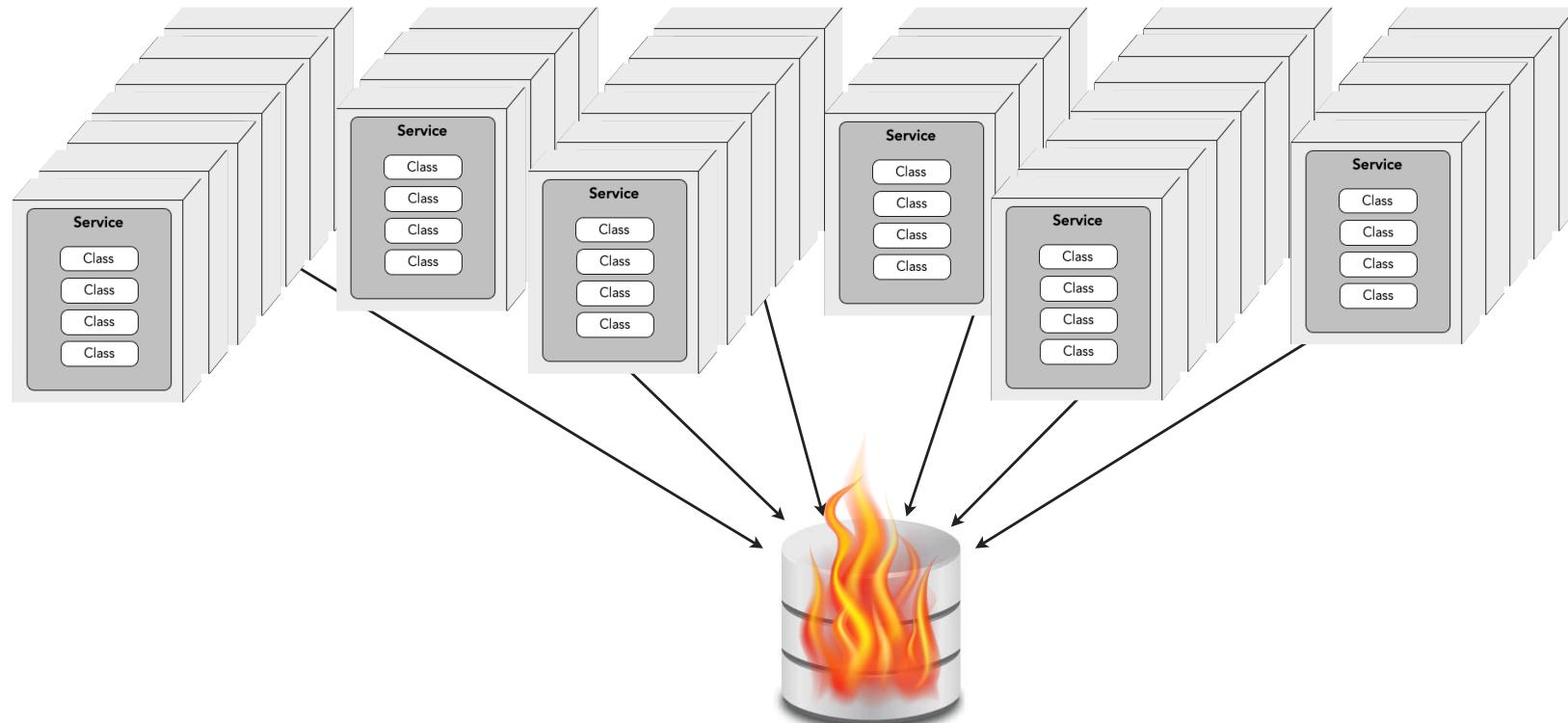
breaking apart data

database scalability



breaking apart data

database scalability



breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers



change
control



database
connections



database
scalability

breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers



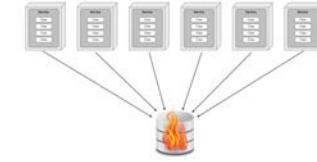
change
control



database
connections



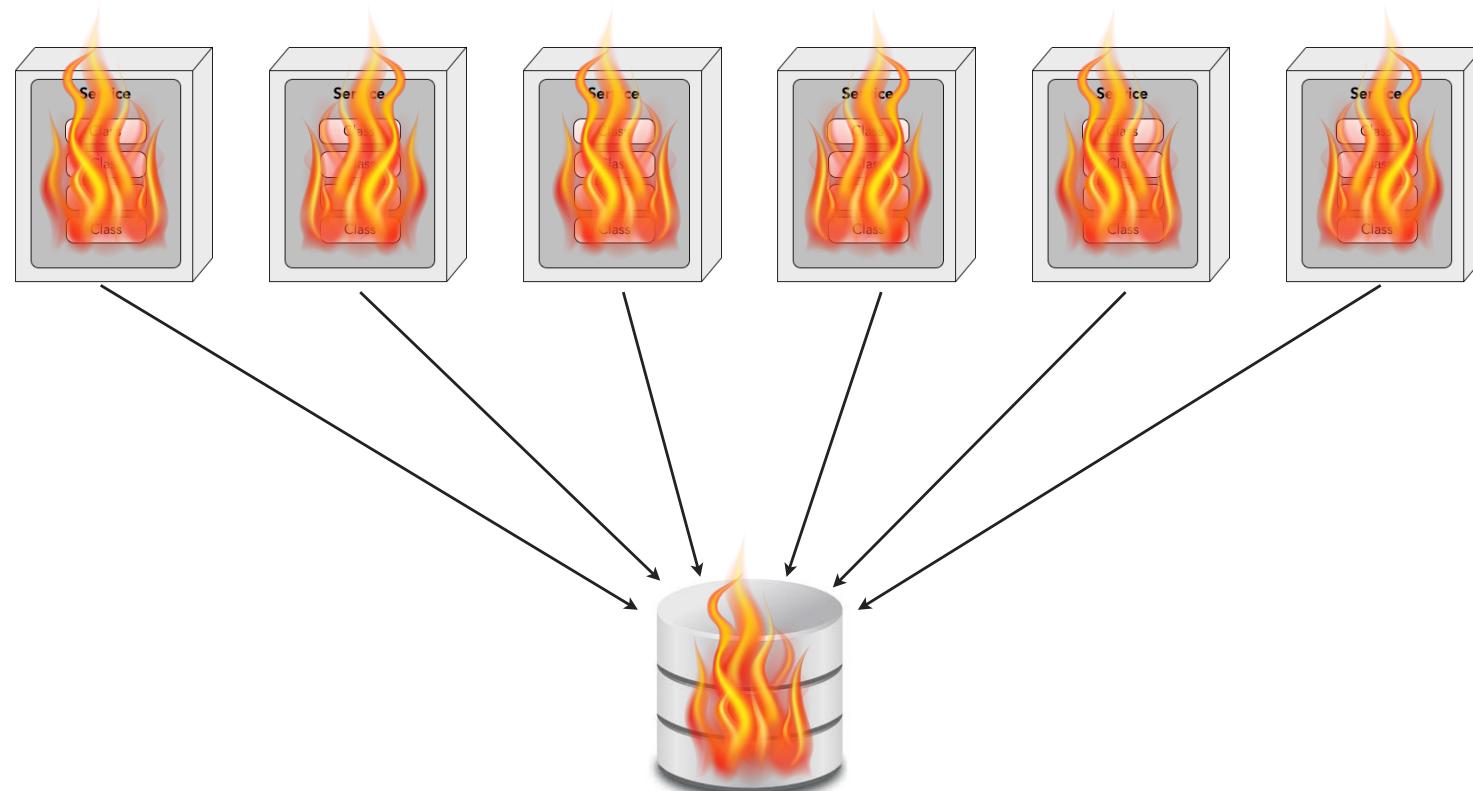
database
scalability



fault
tolerance

breaking apart data

fault tolerance



breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers



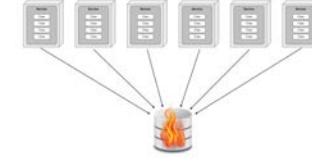
change
control



database
connections



database
scalability



fault
tolerance

breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers



change
control



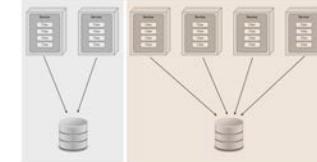
database
connections



database
scalability



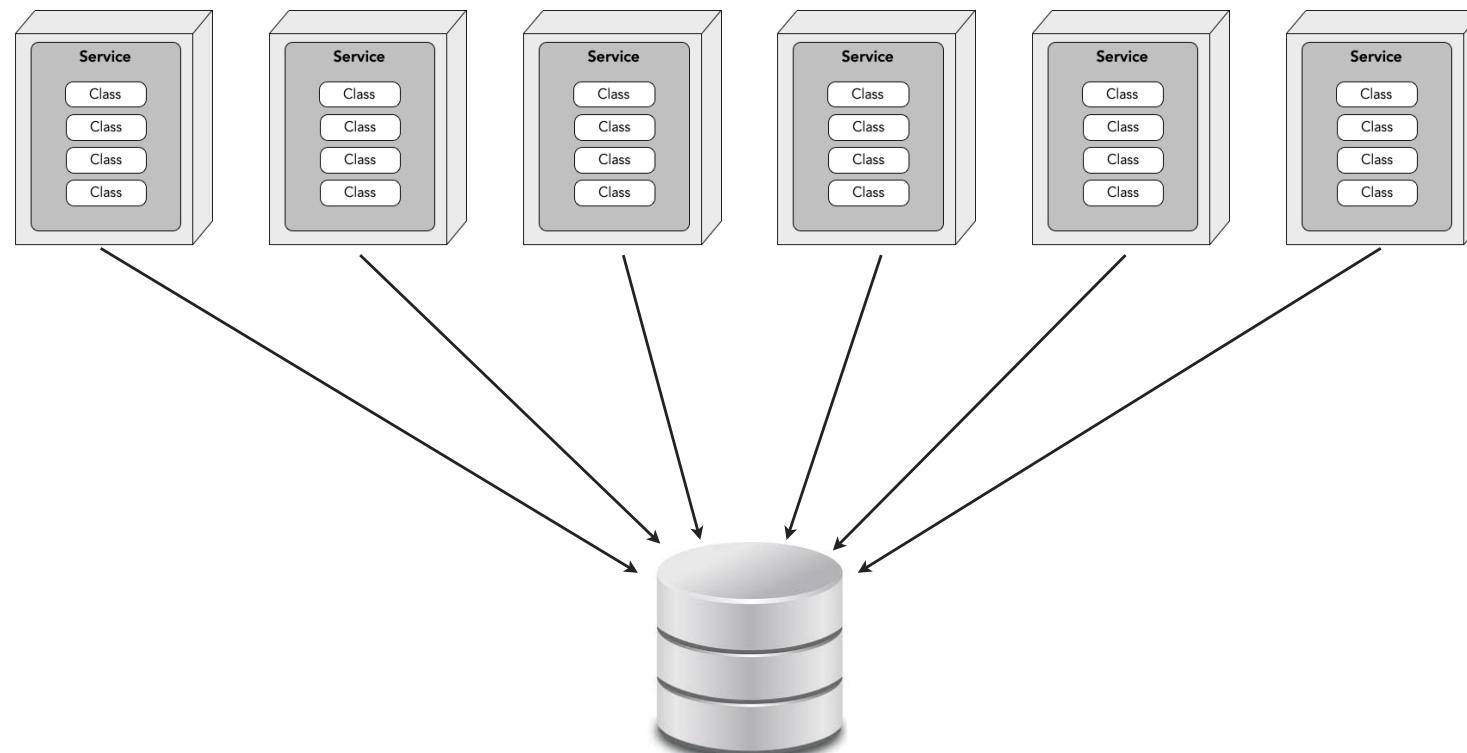
fault
tolerance



architectural
quantum

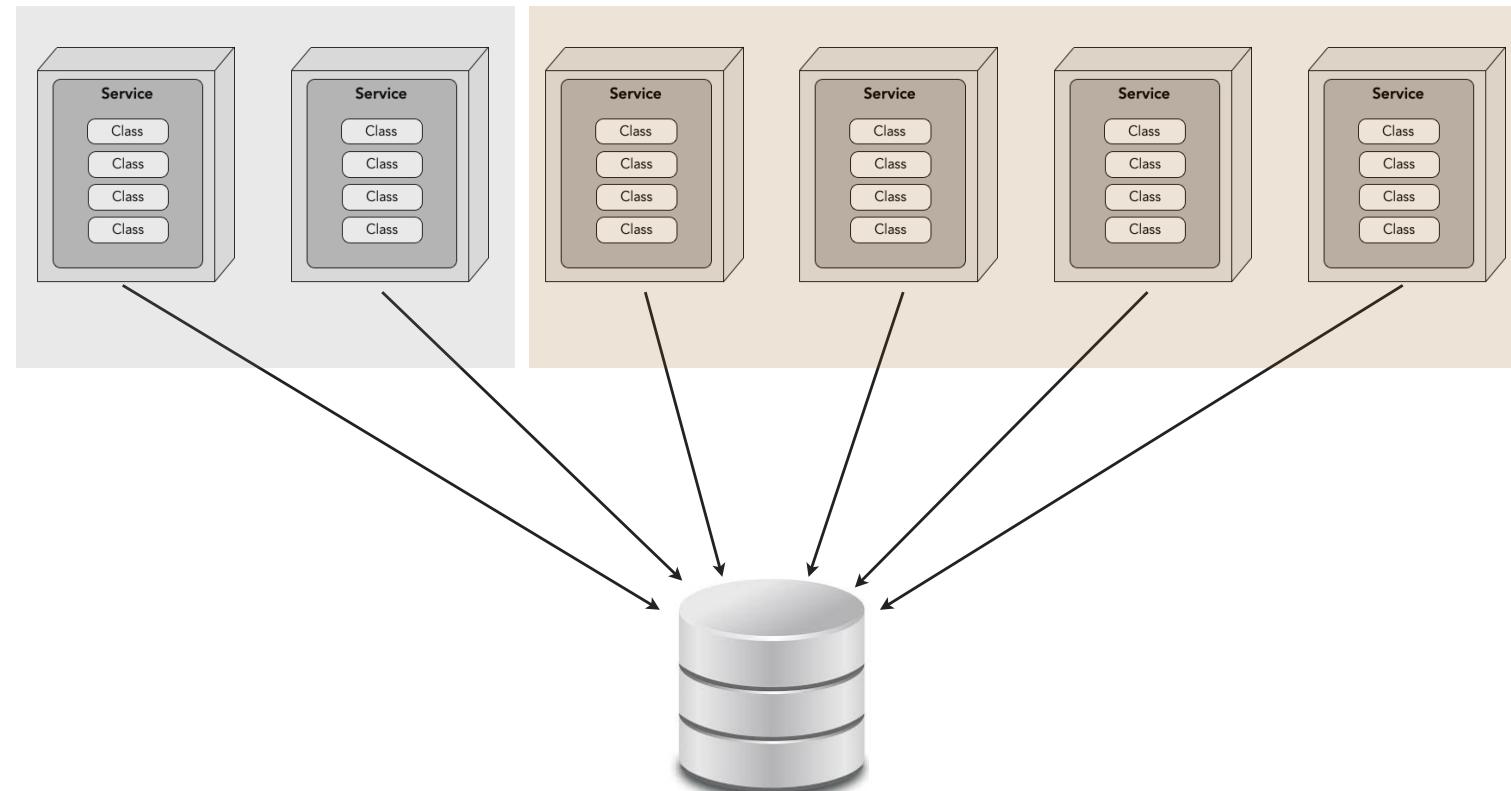
breaking apart data

architectural quantum



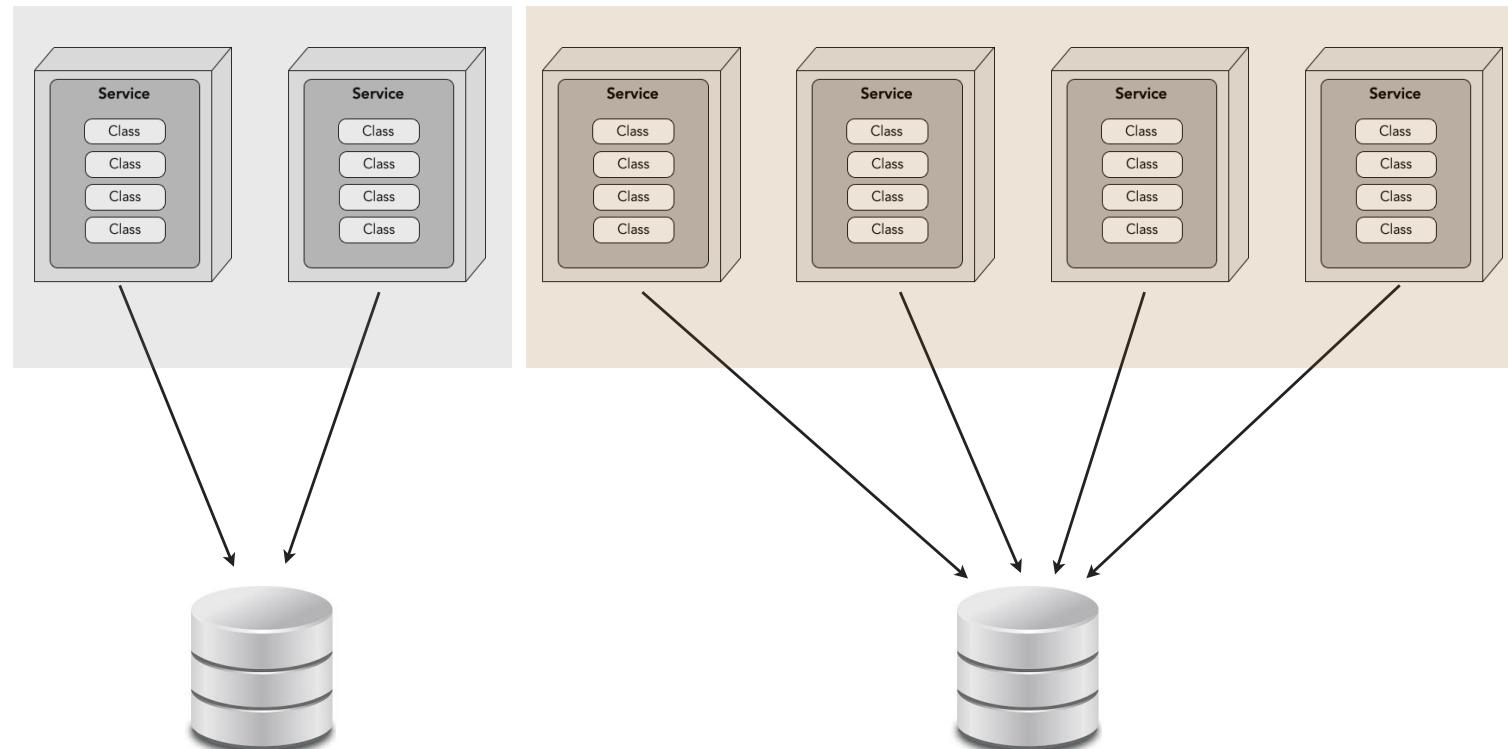
breaking apart data

architectural quantum



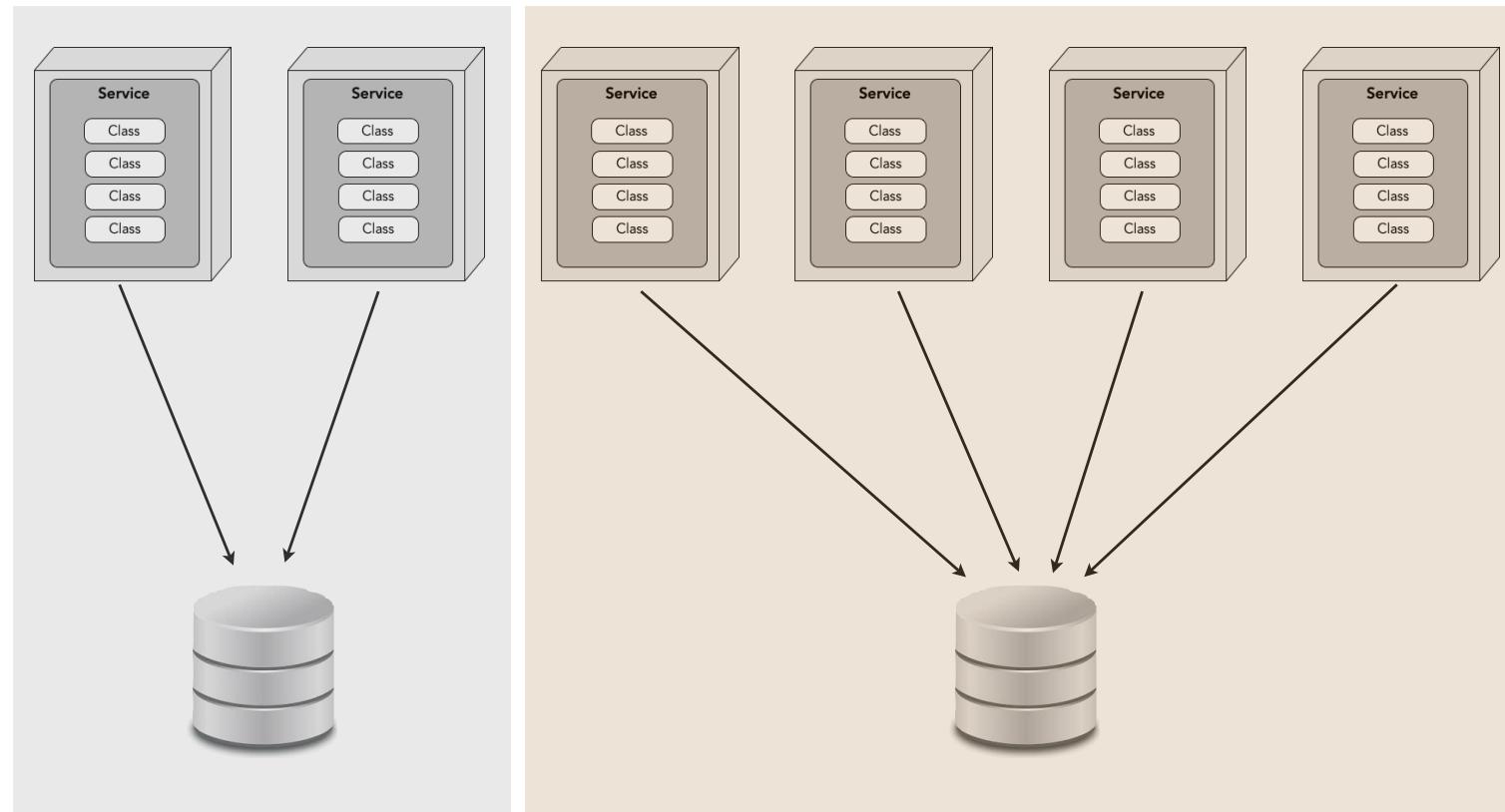
breaking apart data

architectural quantum



breaking apart data

architectural quantum



breaking apart data

“when should I consider breaking apart my data?”

database granularity drivers



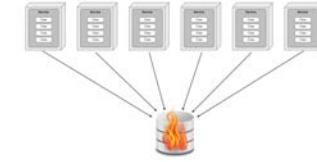
change
control



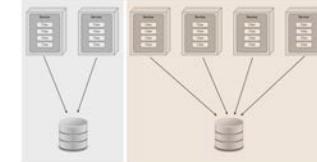
database
connections



database
scalability

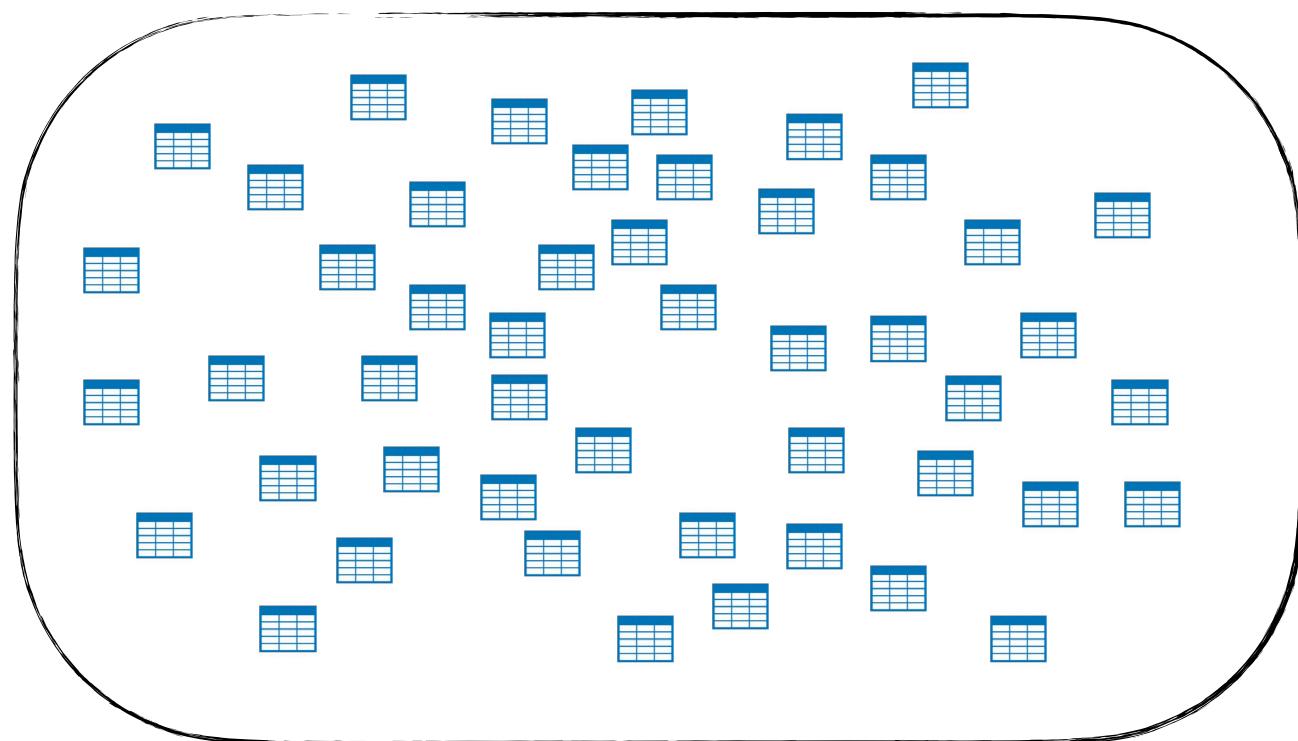


fault
tolerance

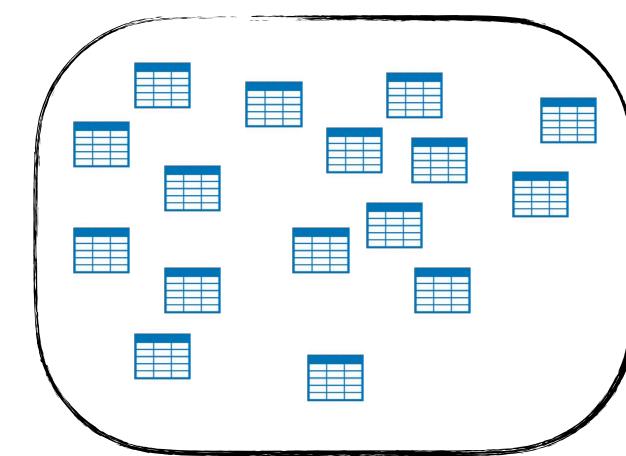
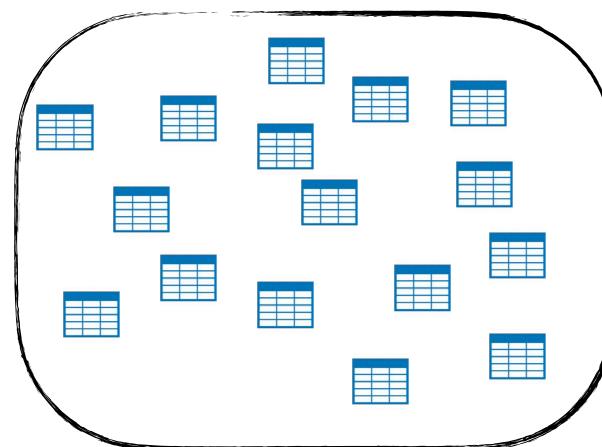
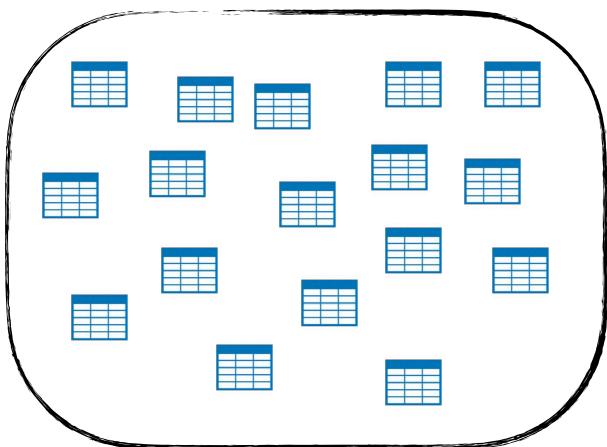


architectural
quantum

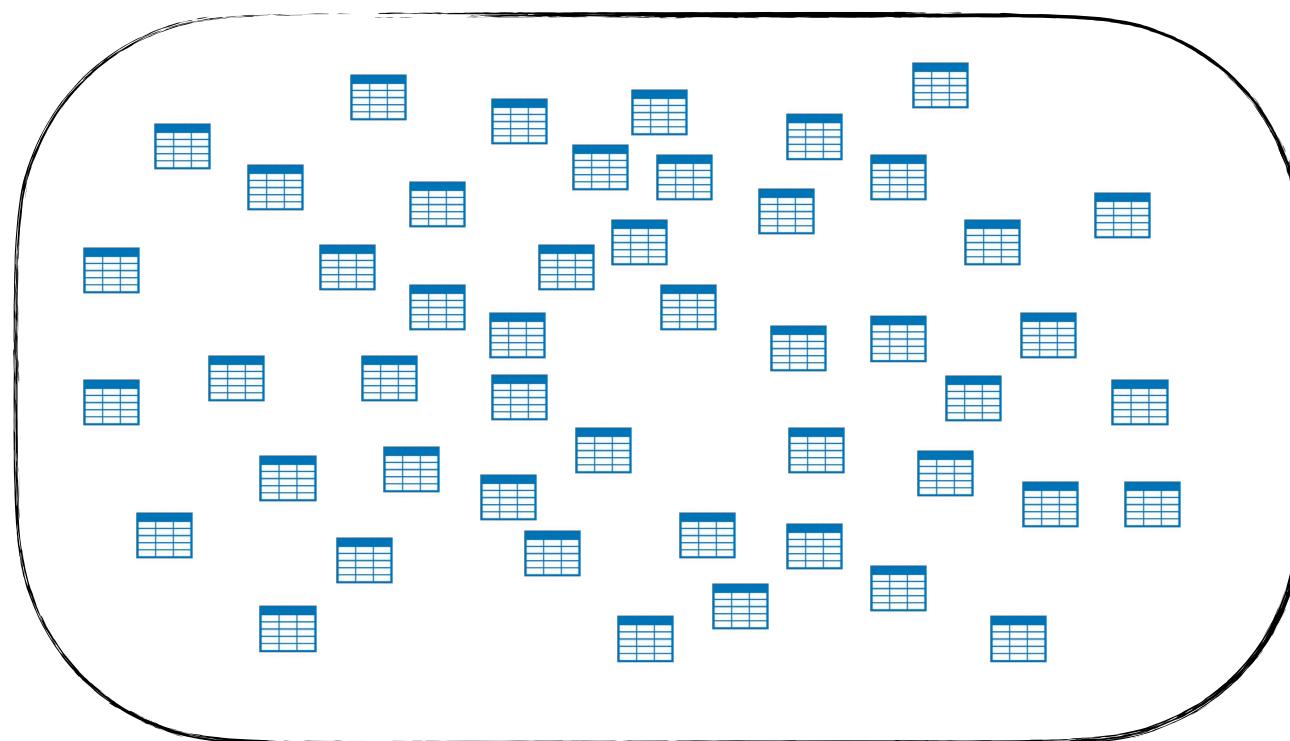
decomposing data



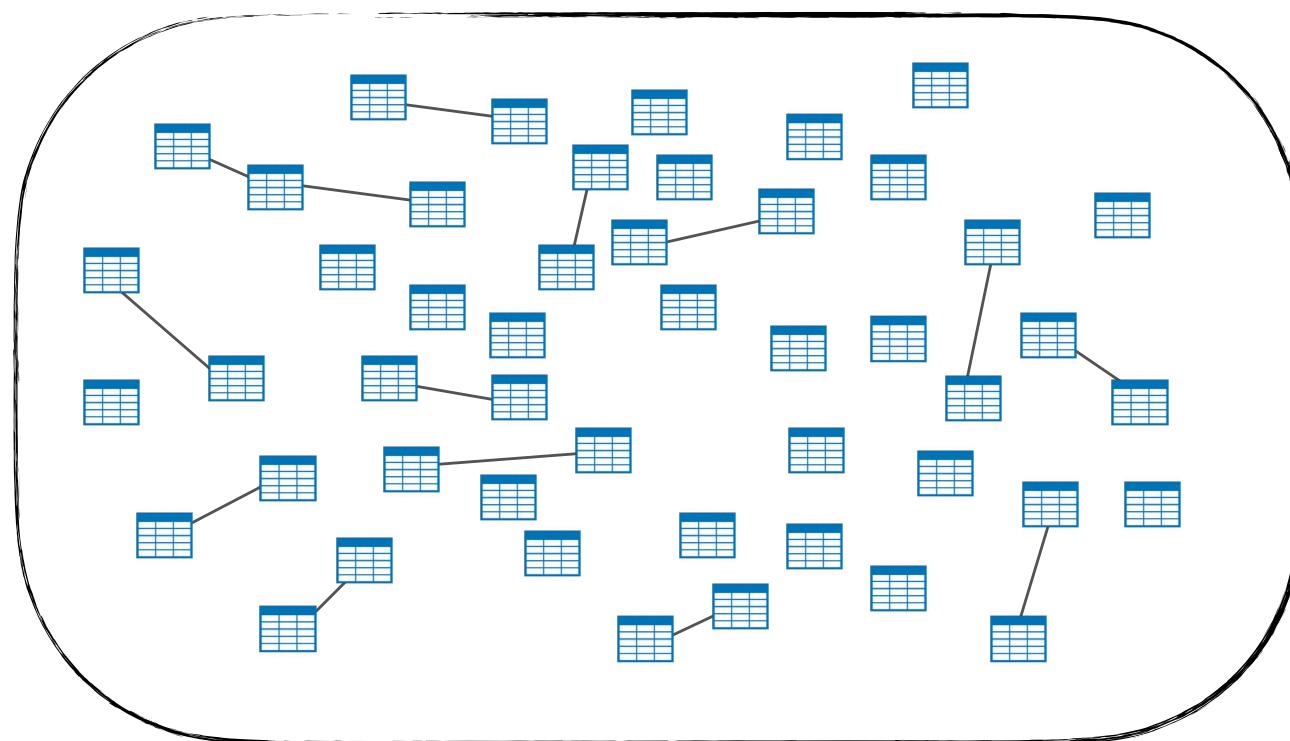
decomposing data



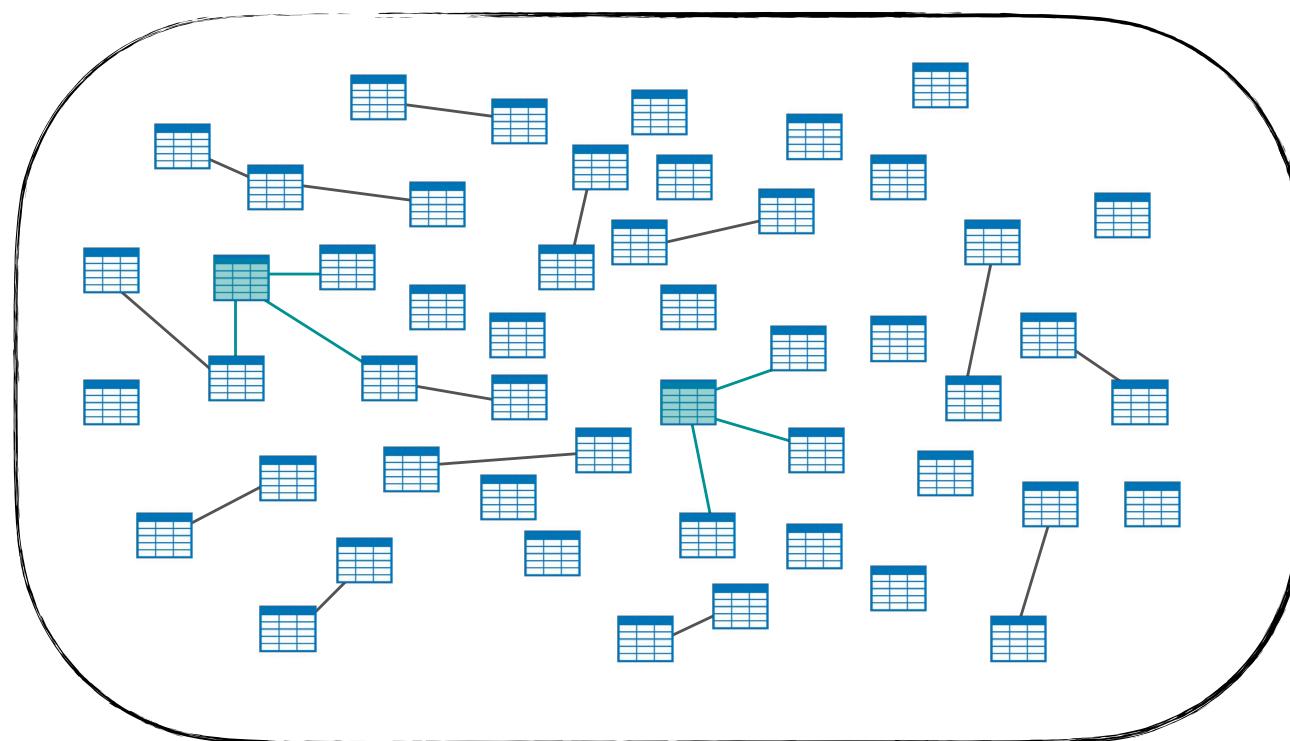
decomposing data



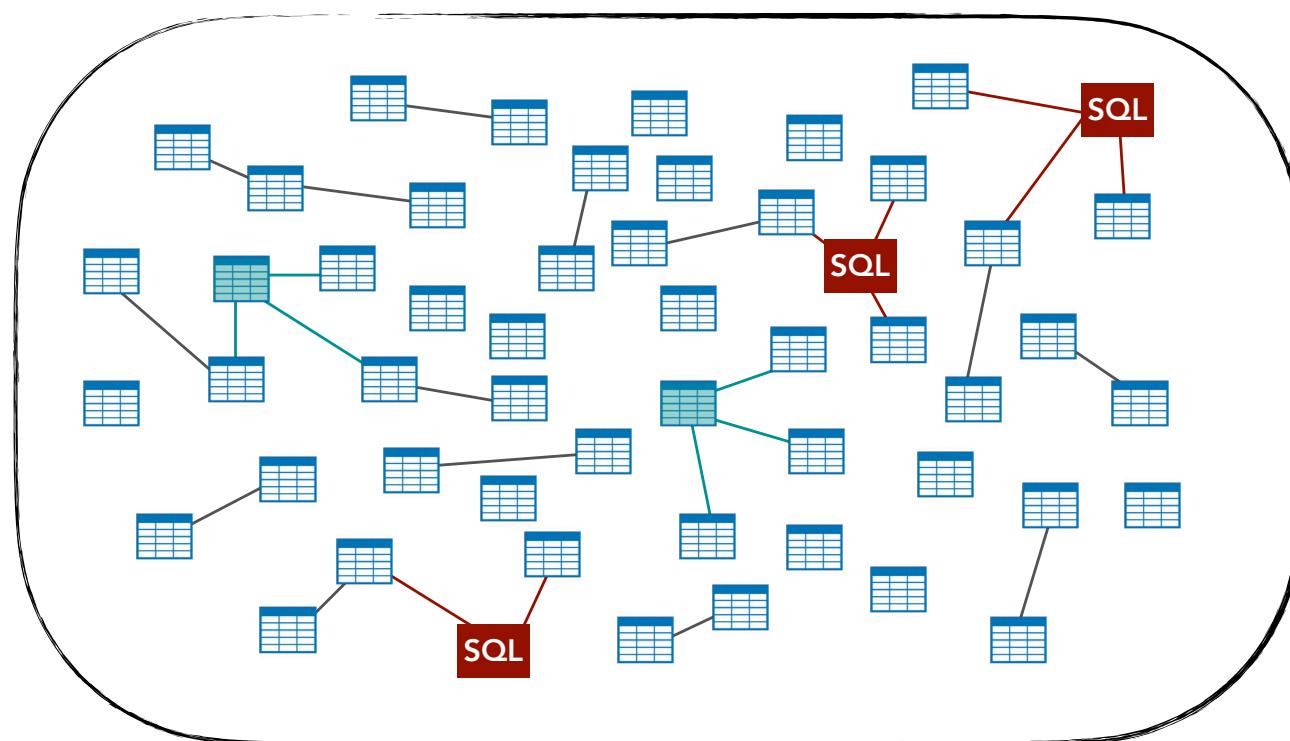
decomposing data



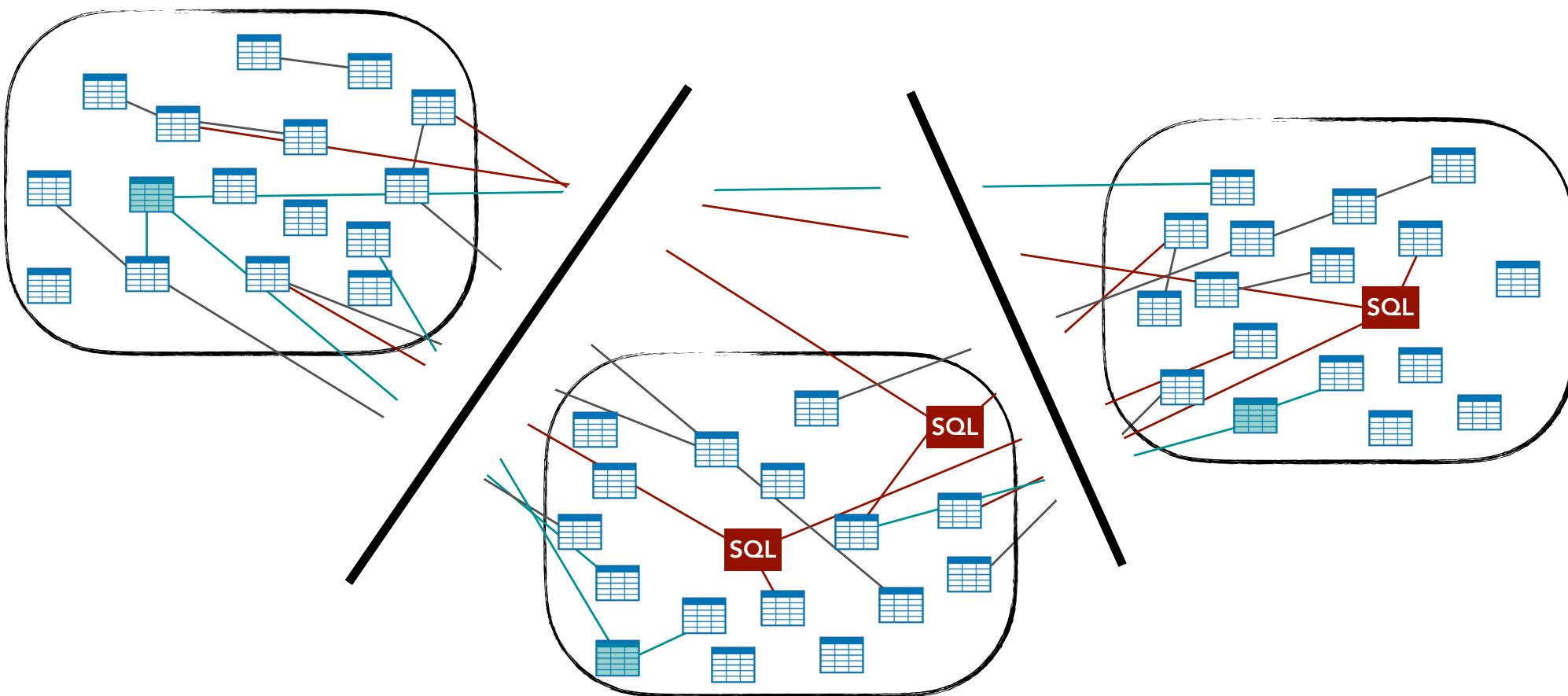
decomposing data



decomposing data

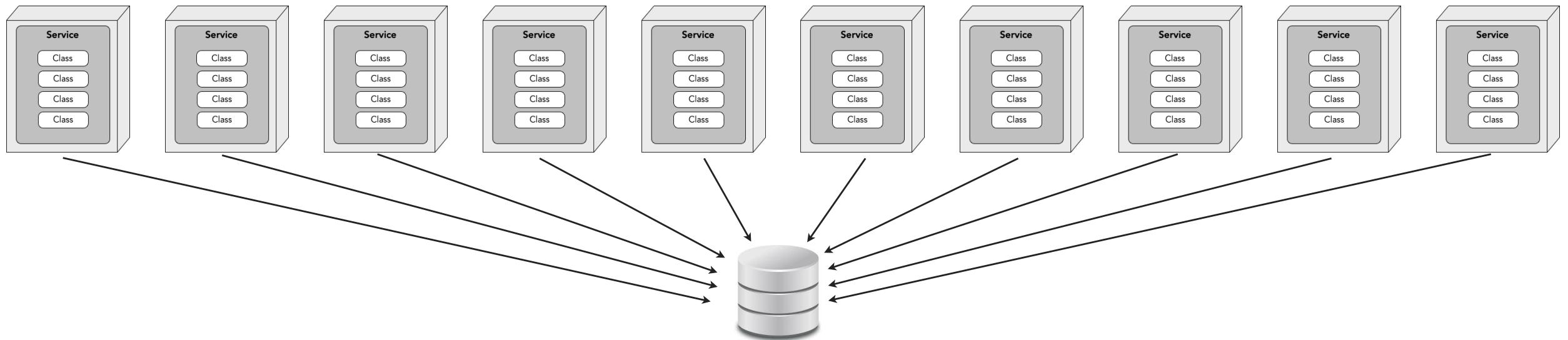


decomposing data

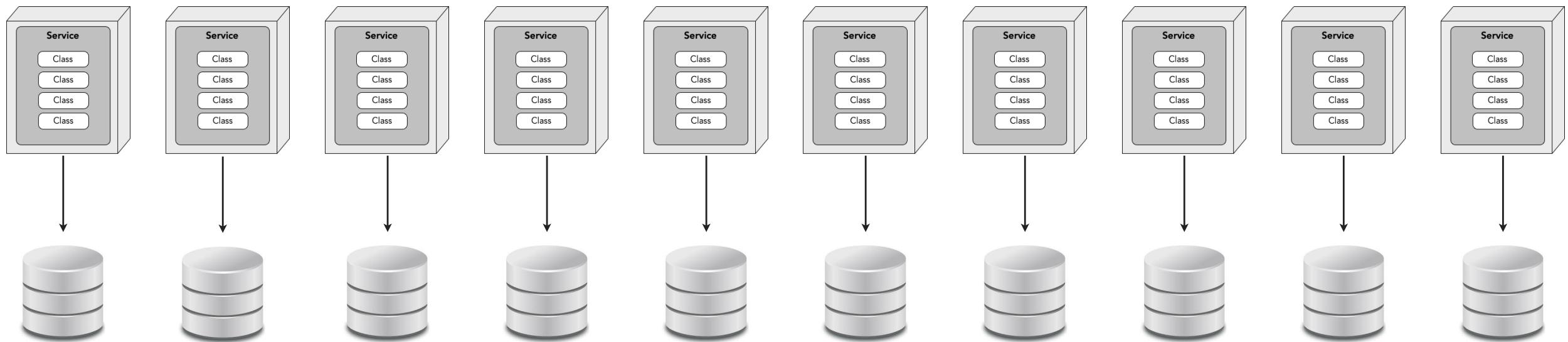


Data Domains

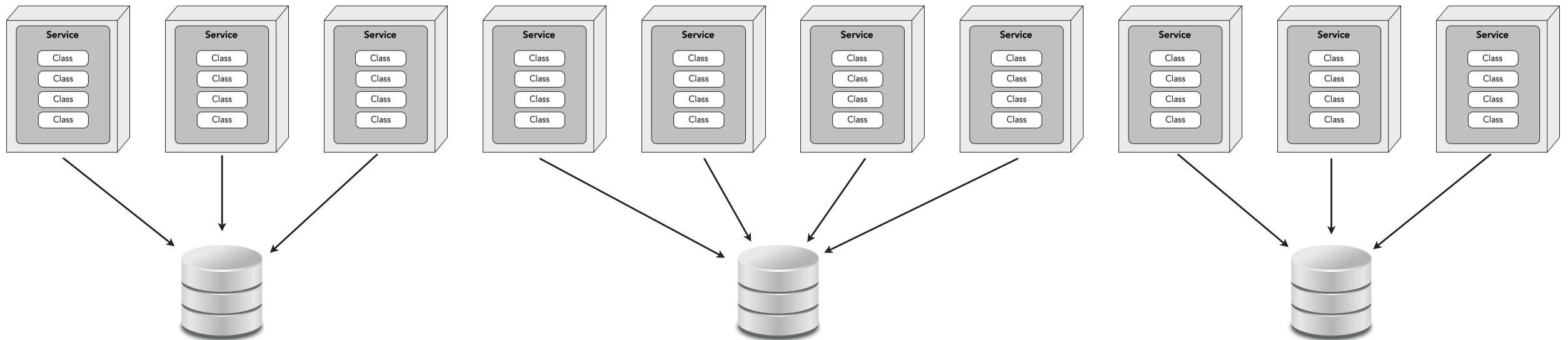
single monolithic database



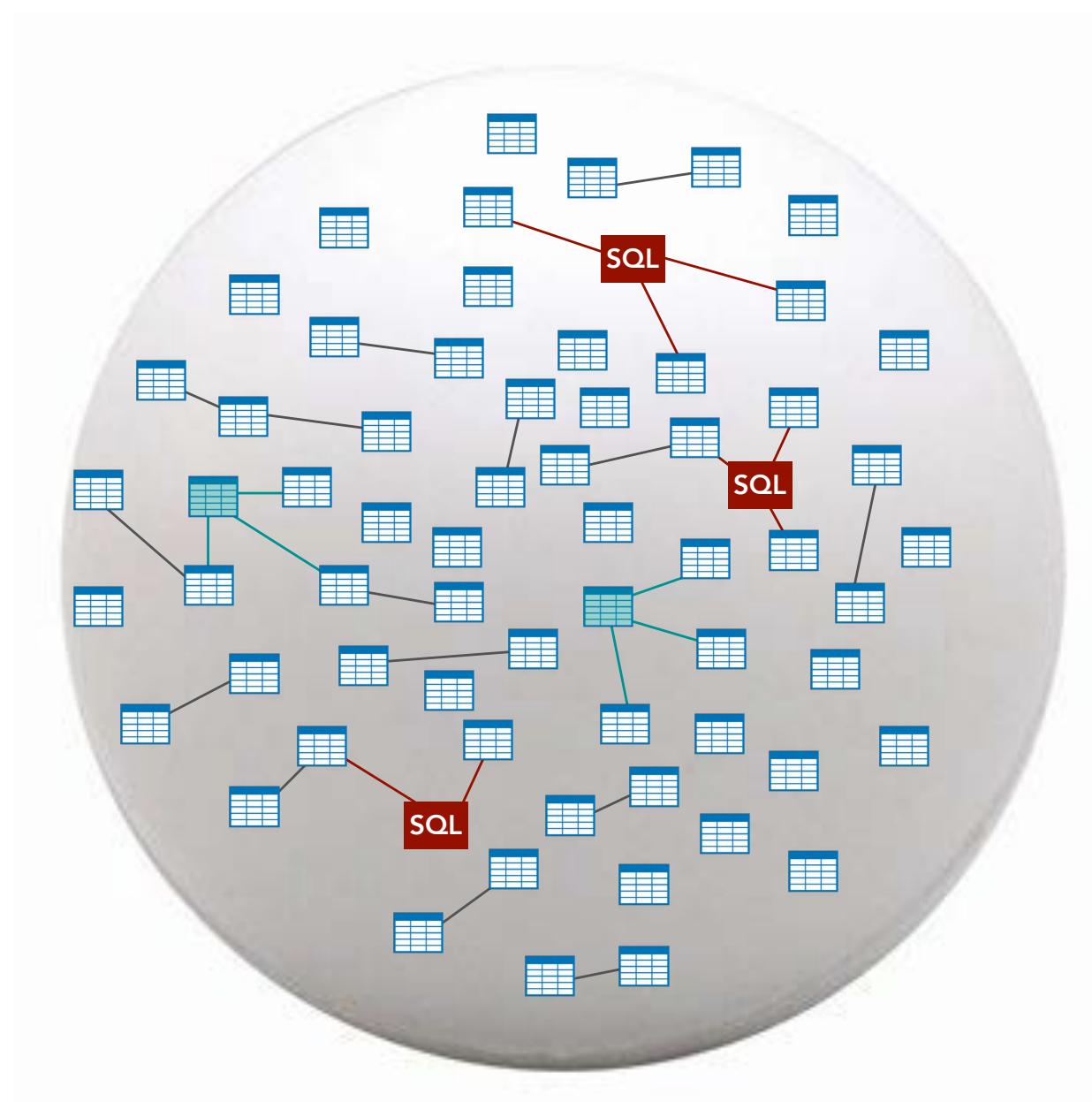
separate databases for each service



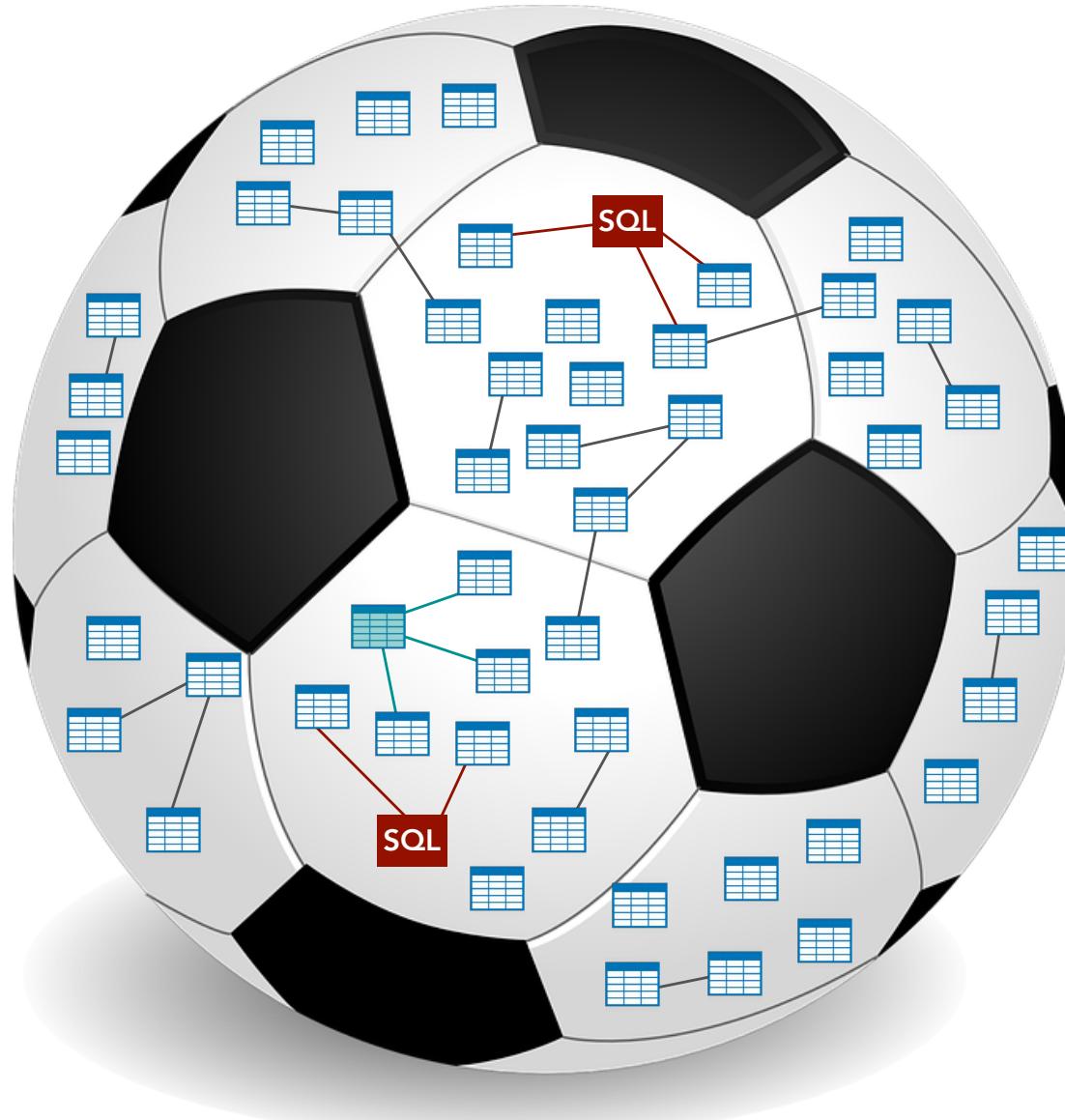
data domains



data domains



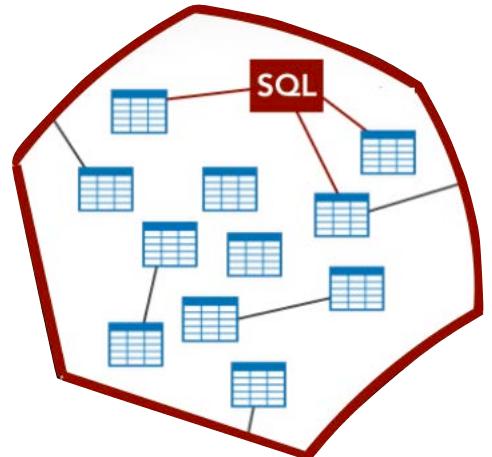
data domains



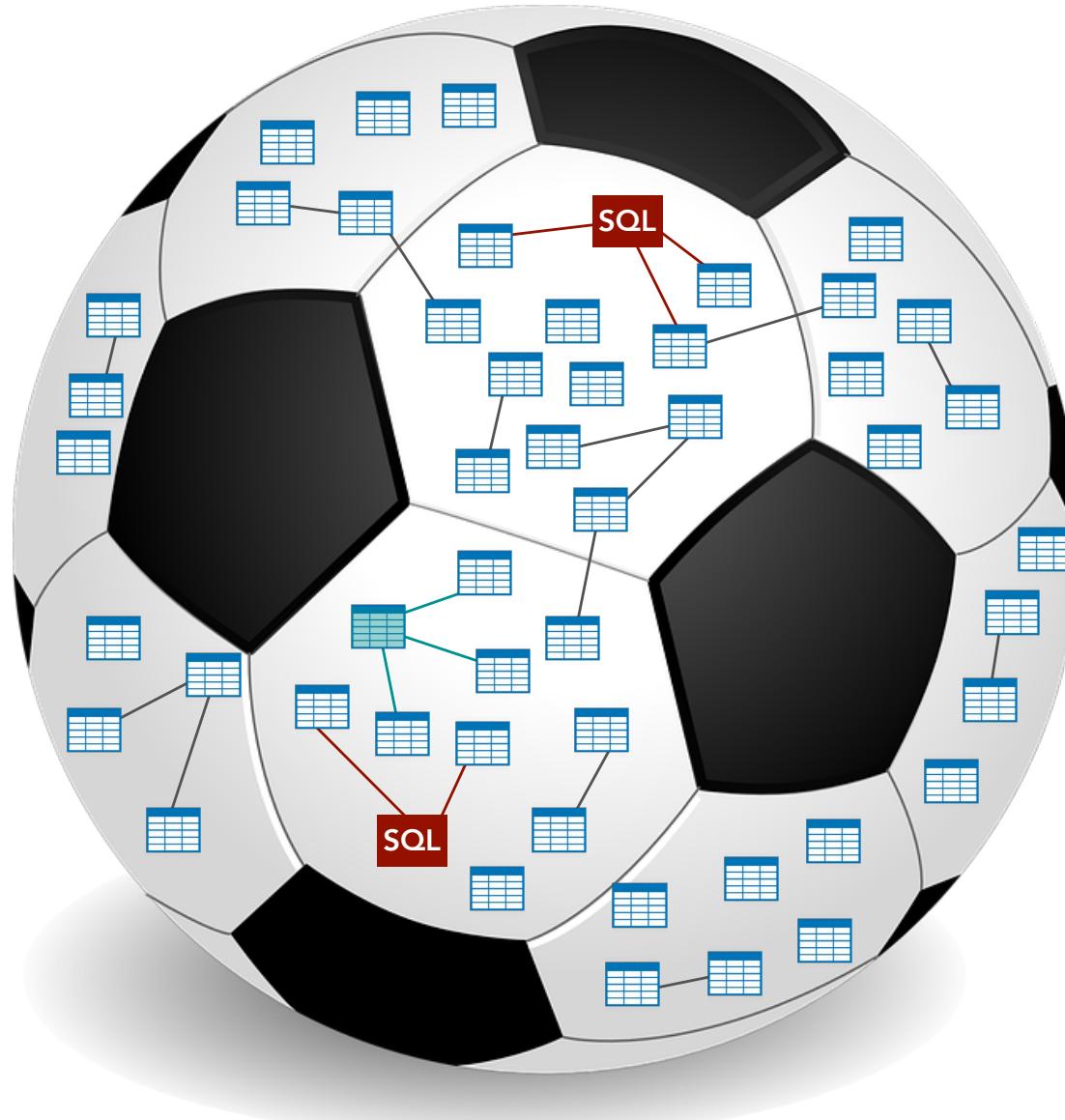
data domains



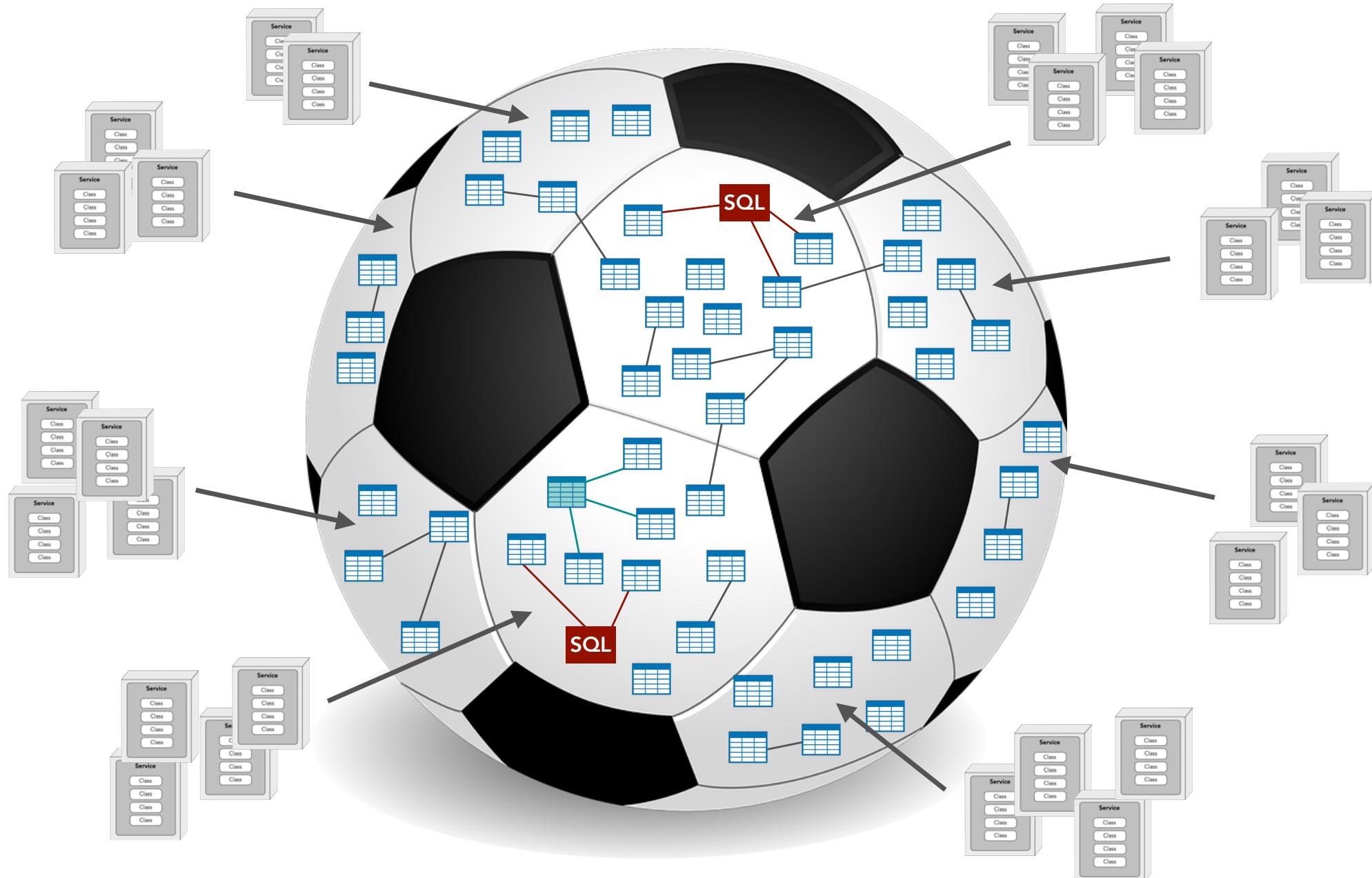
data domains



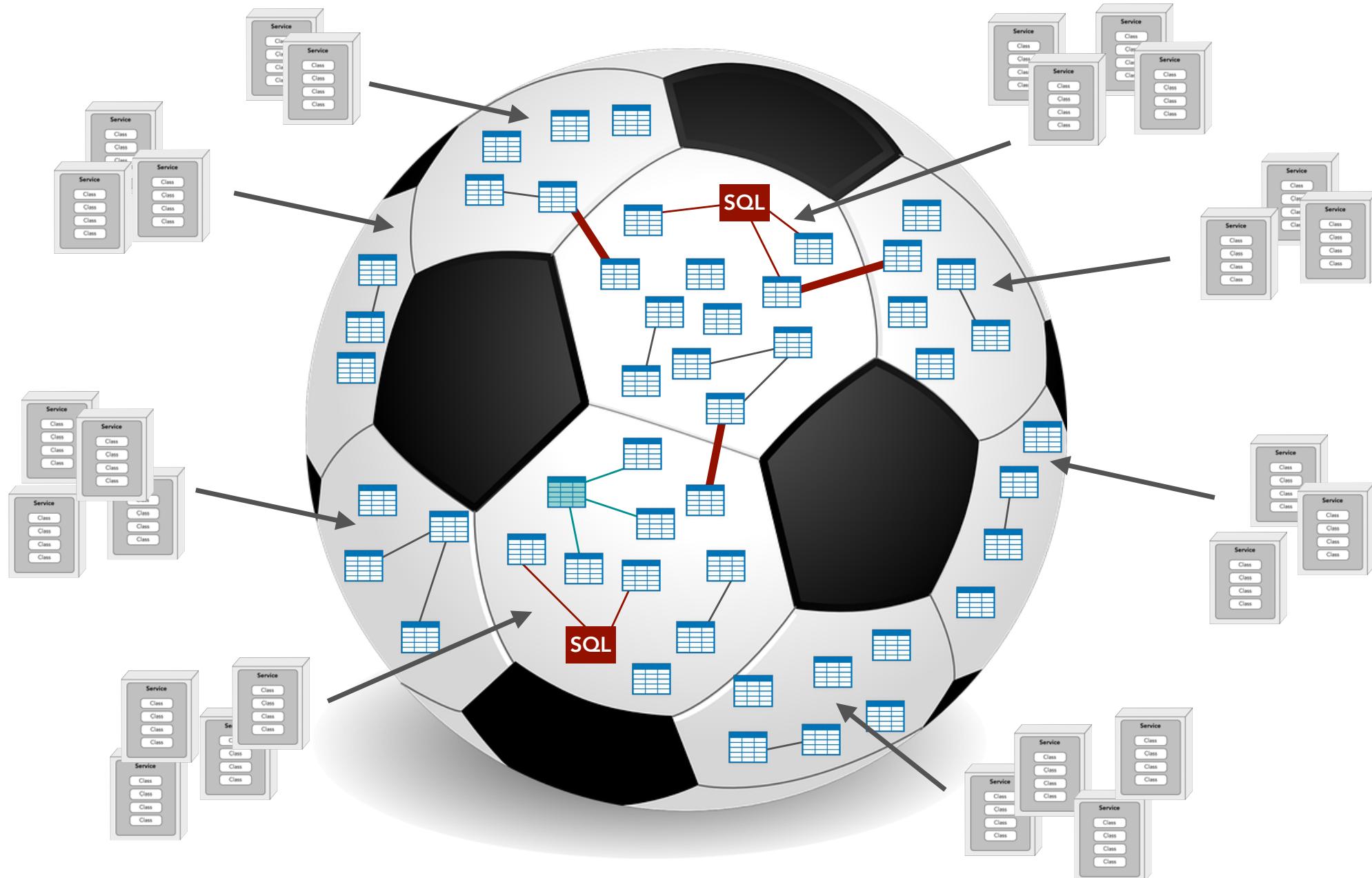
data domains



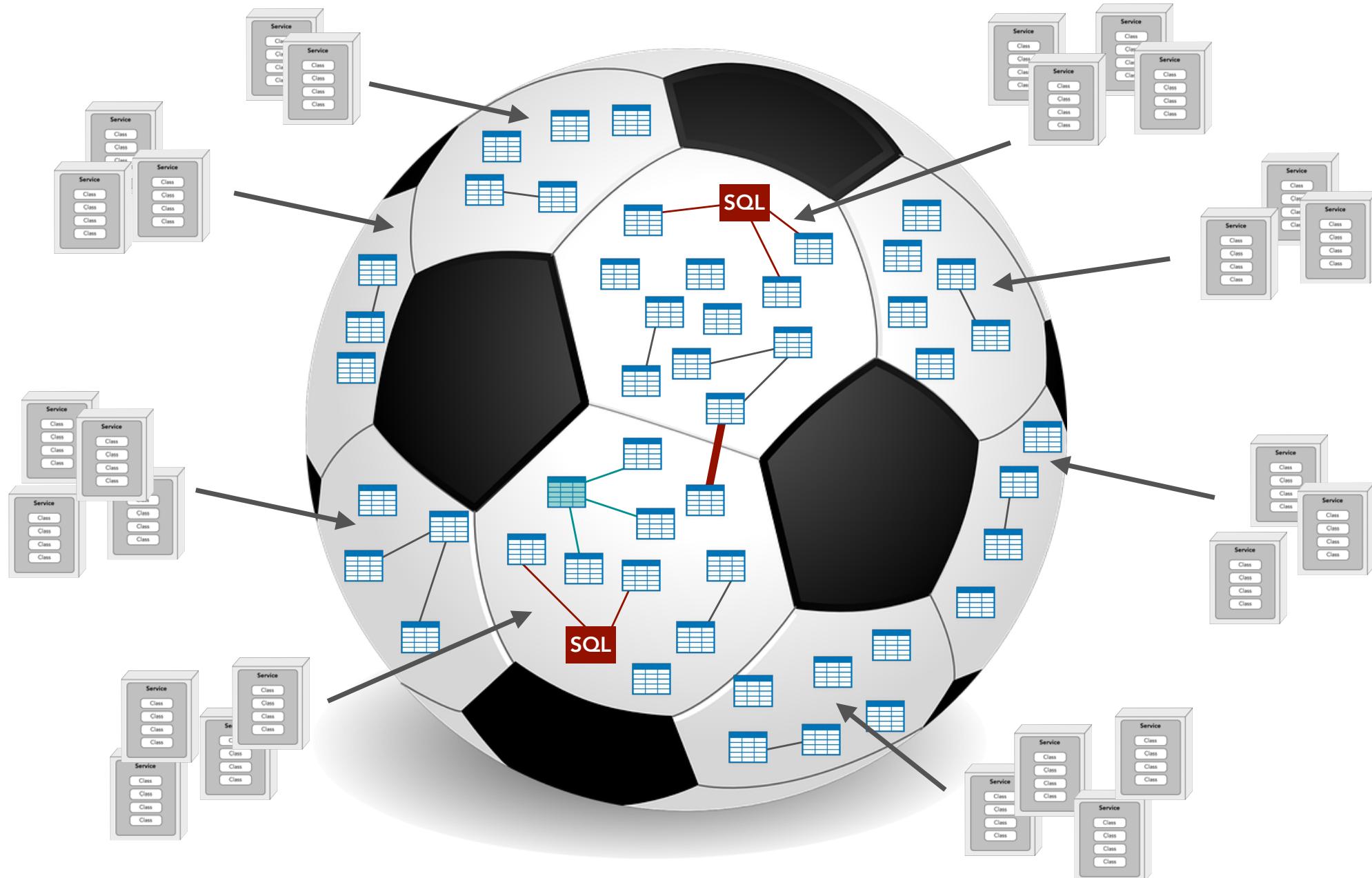
data domains



data domains



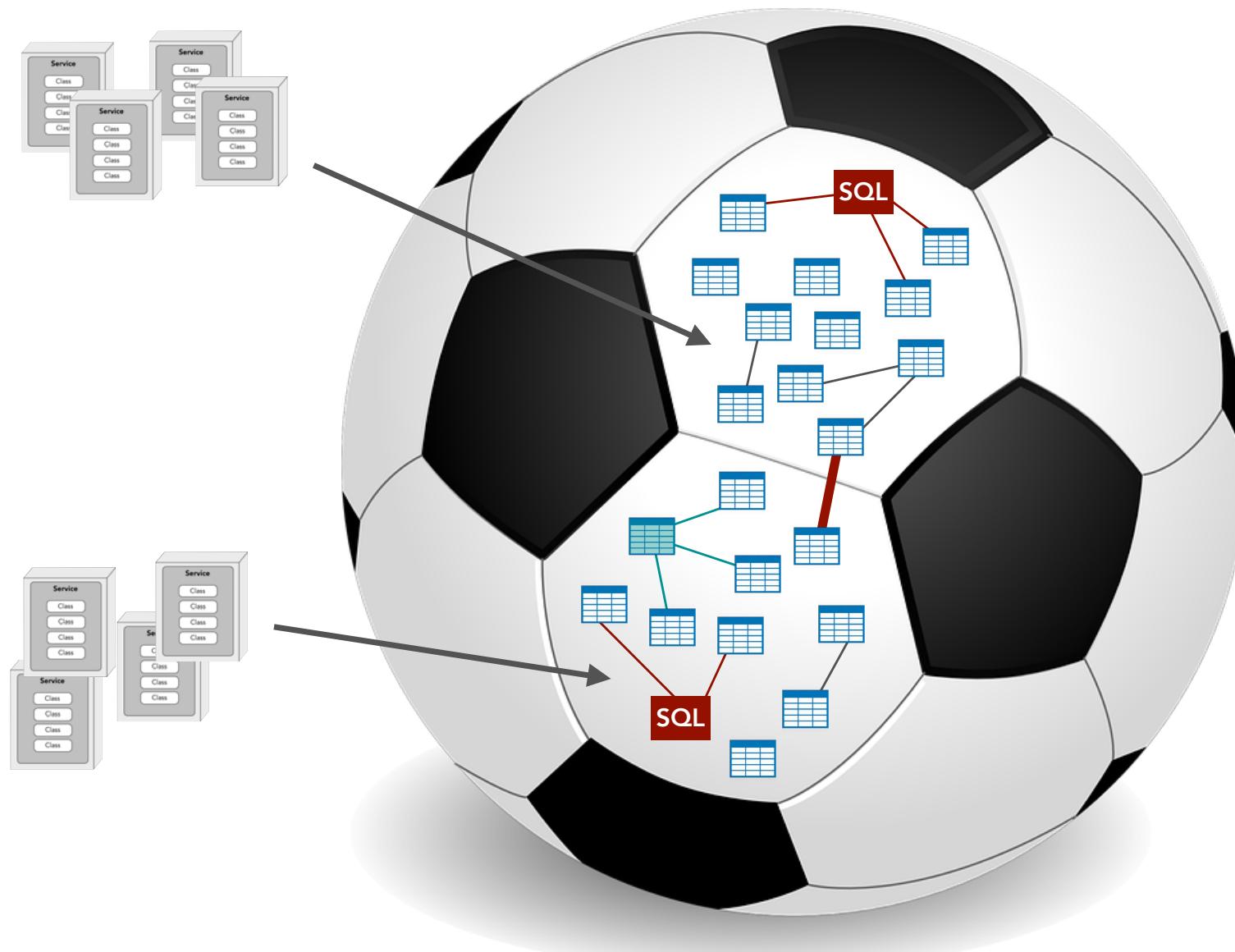
data domains



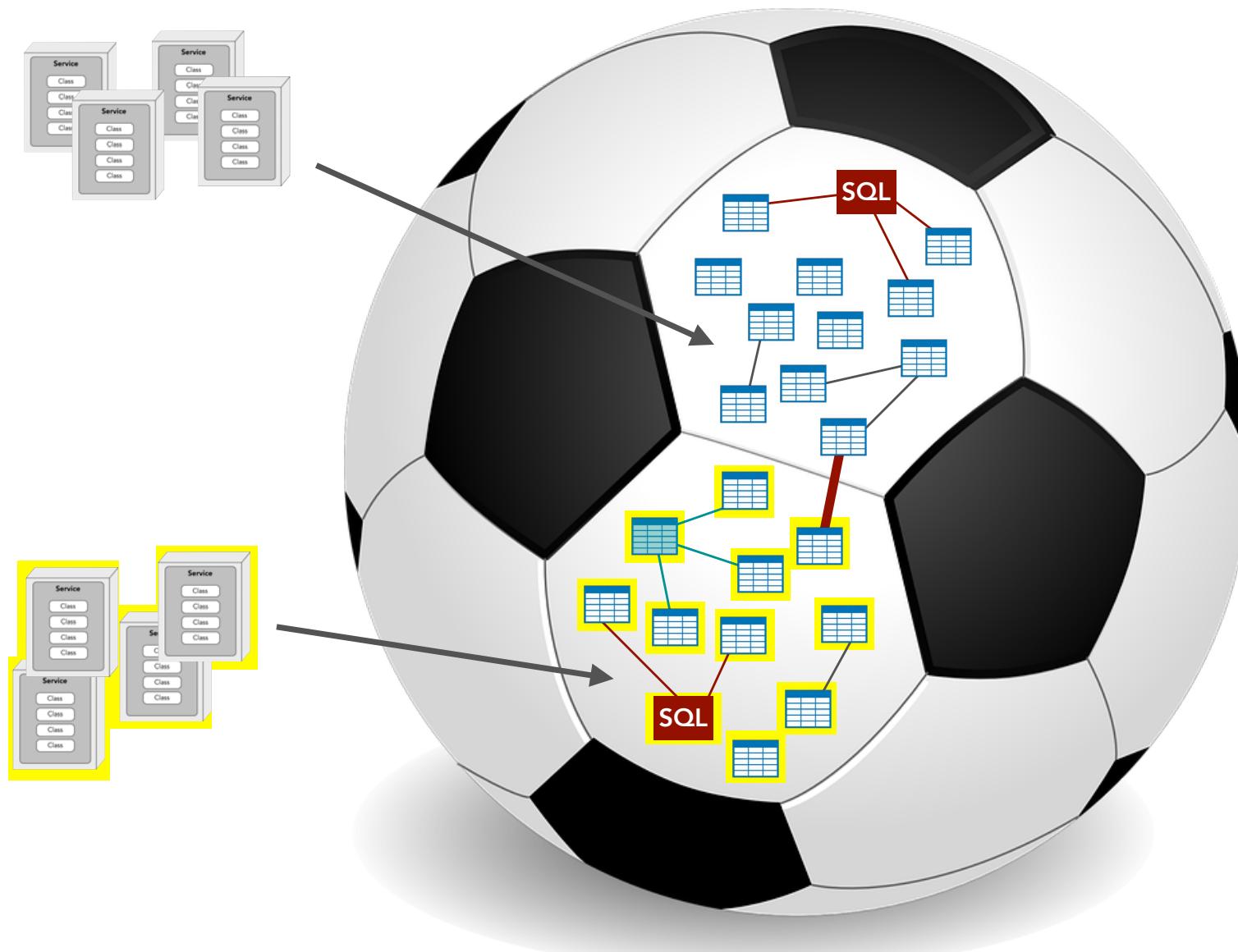
data domains



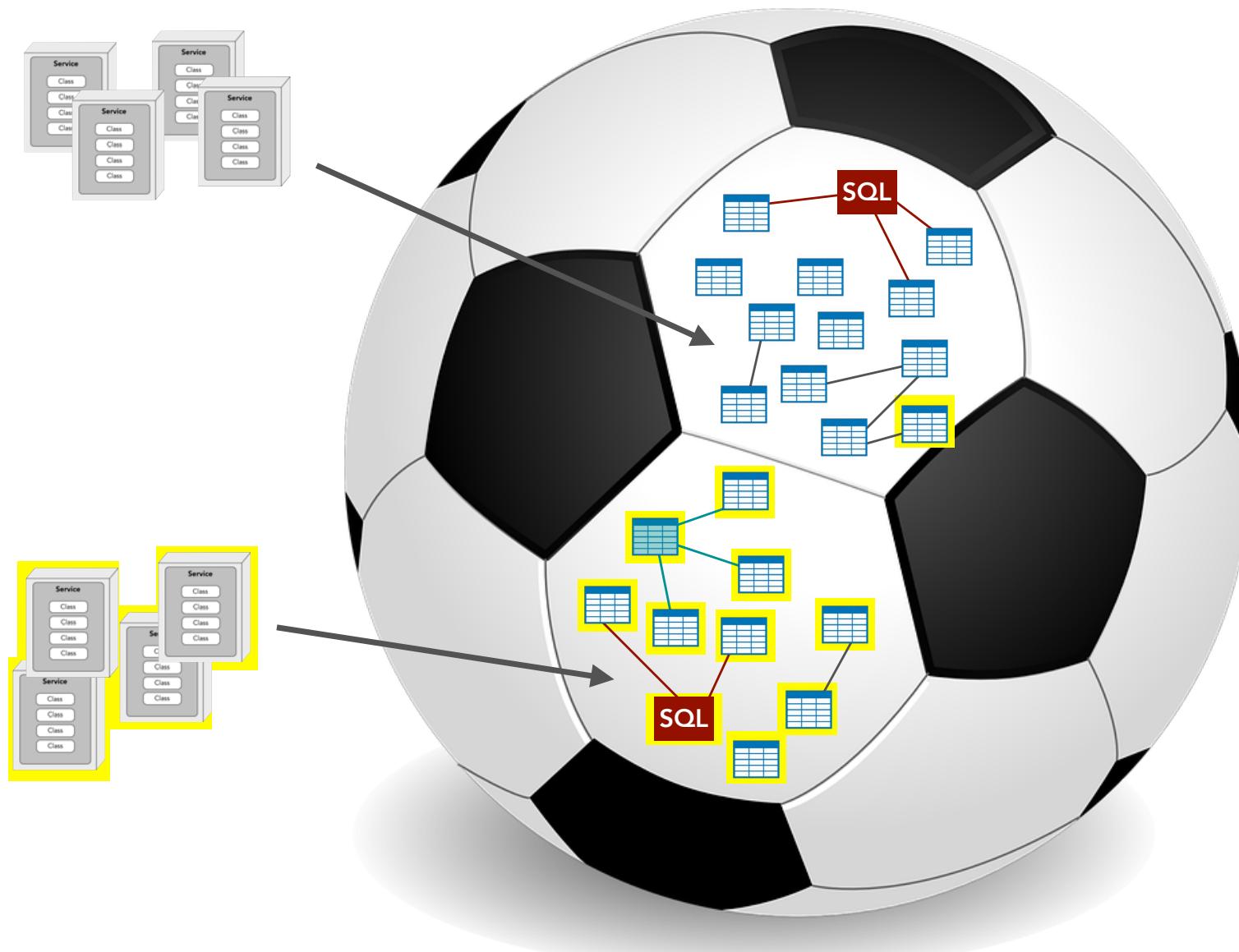
data domains



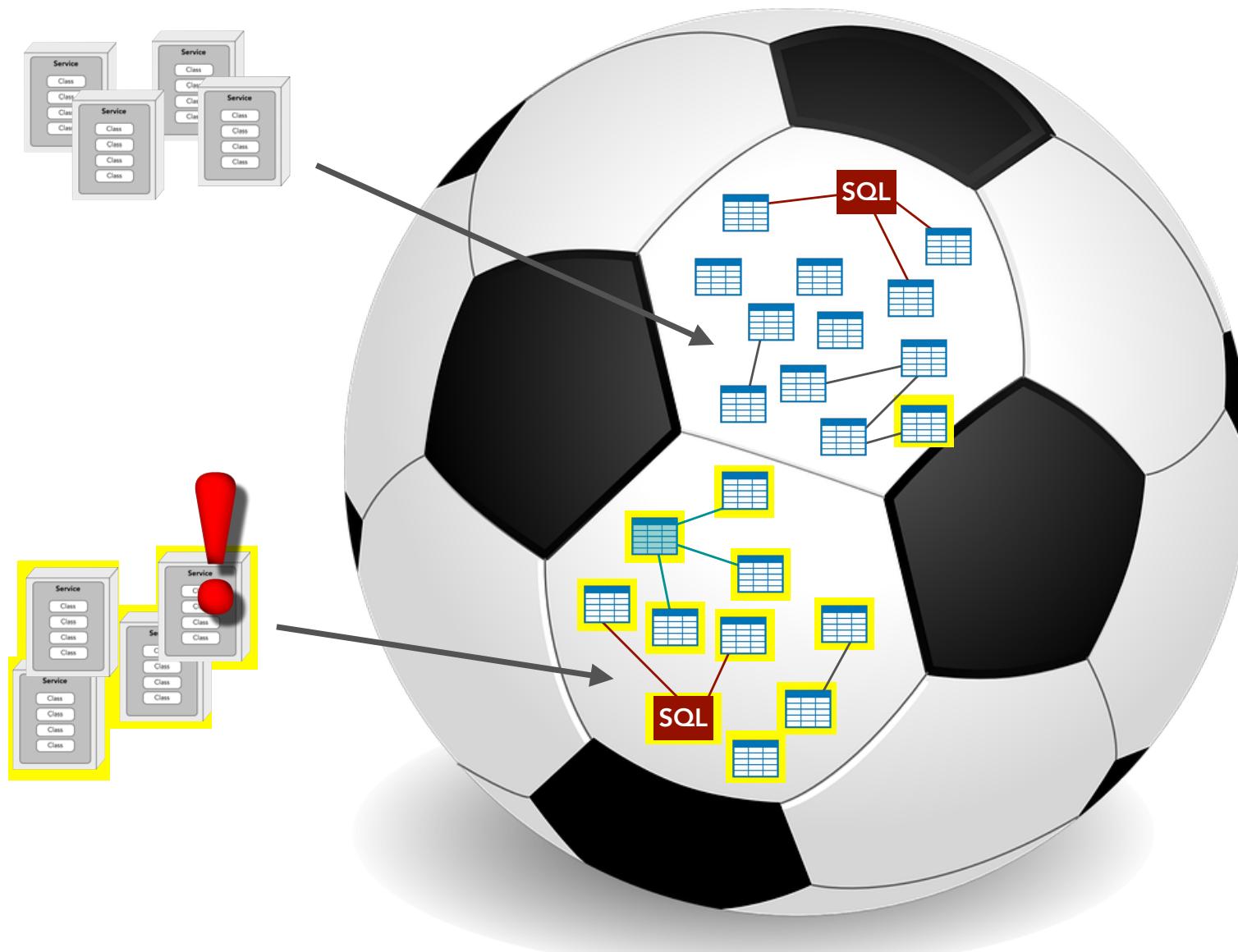
data domains



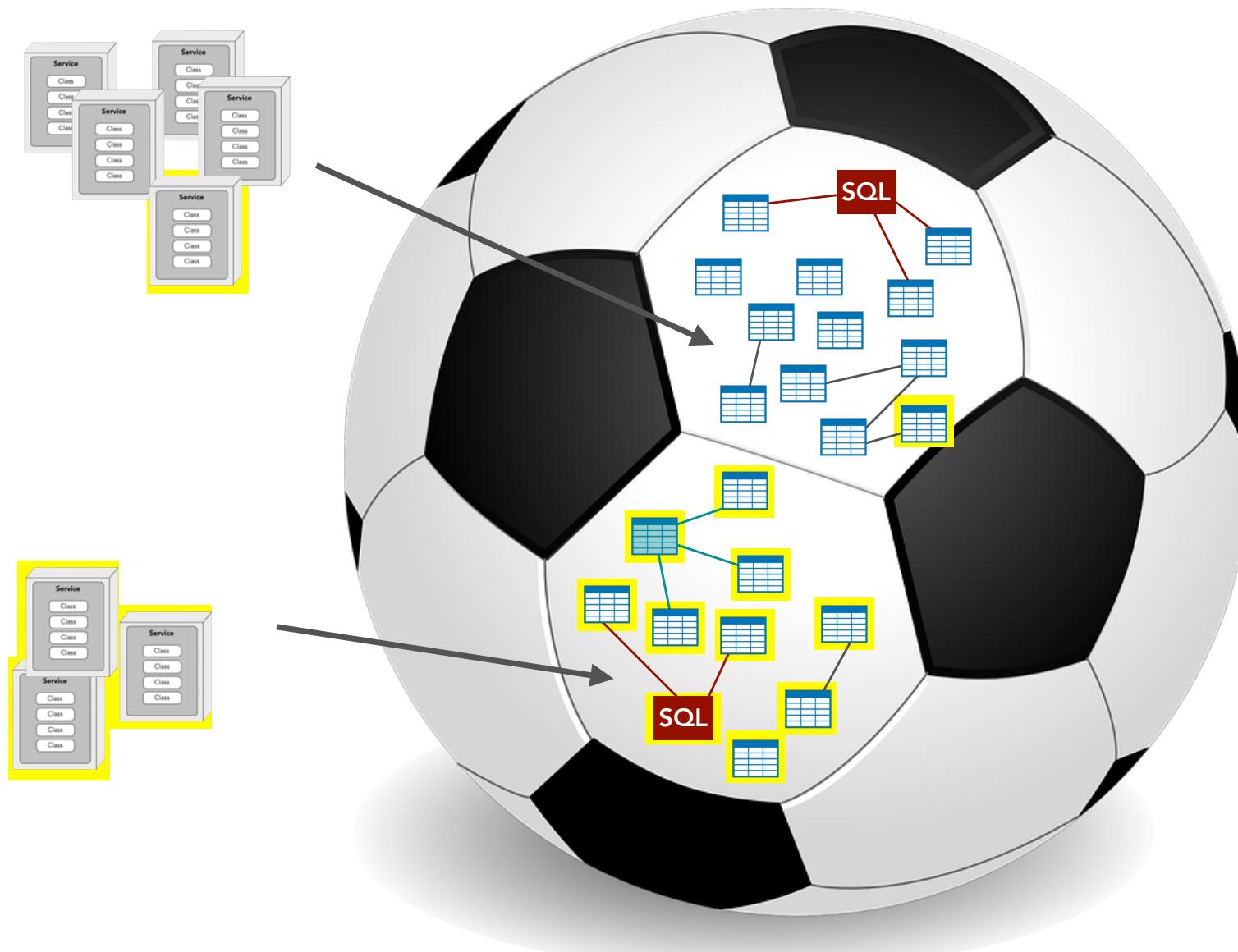
data domains



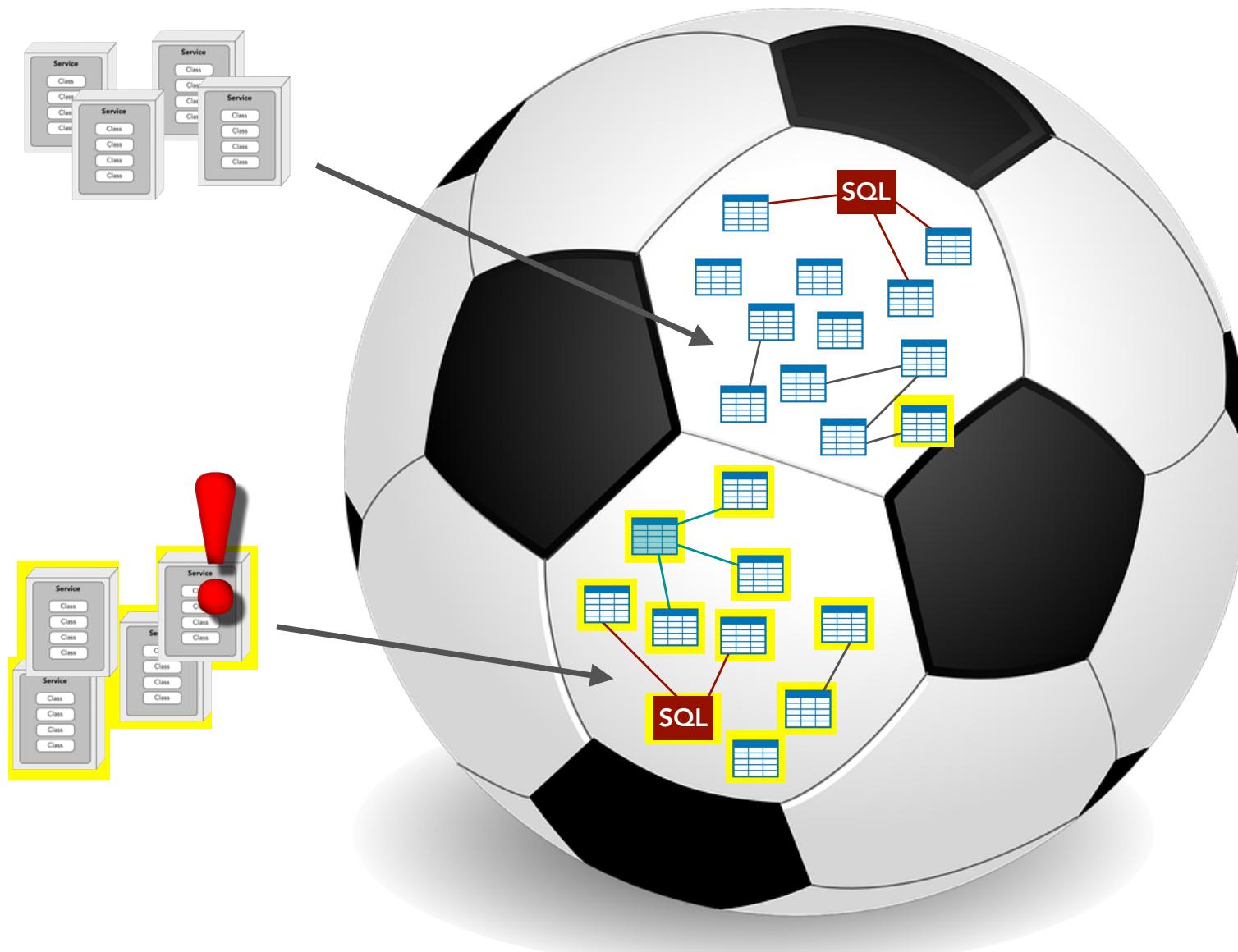
data domains



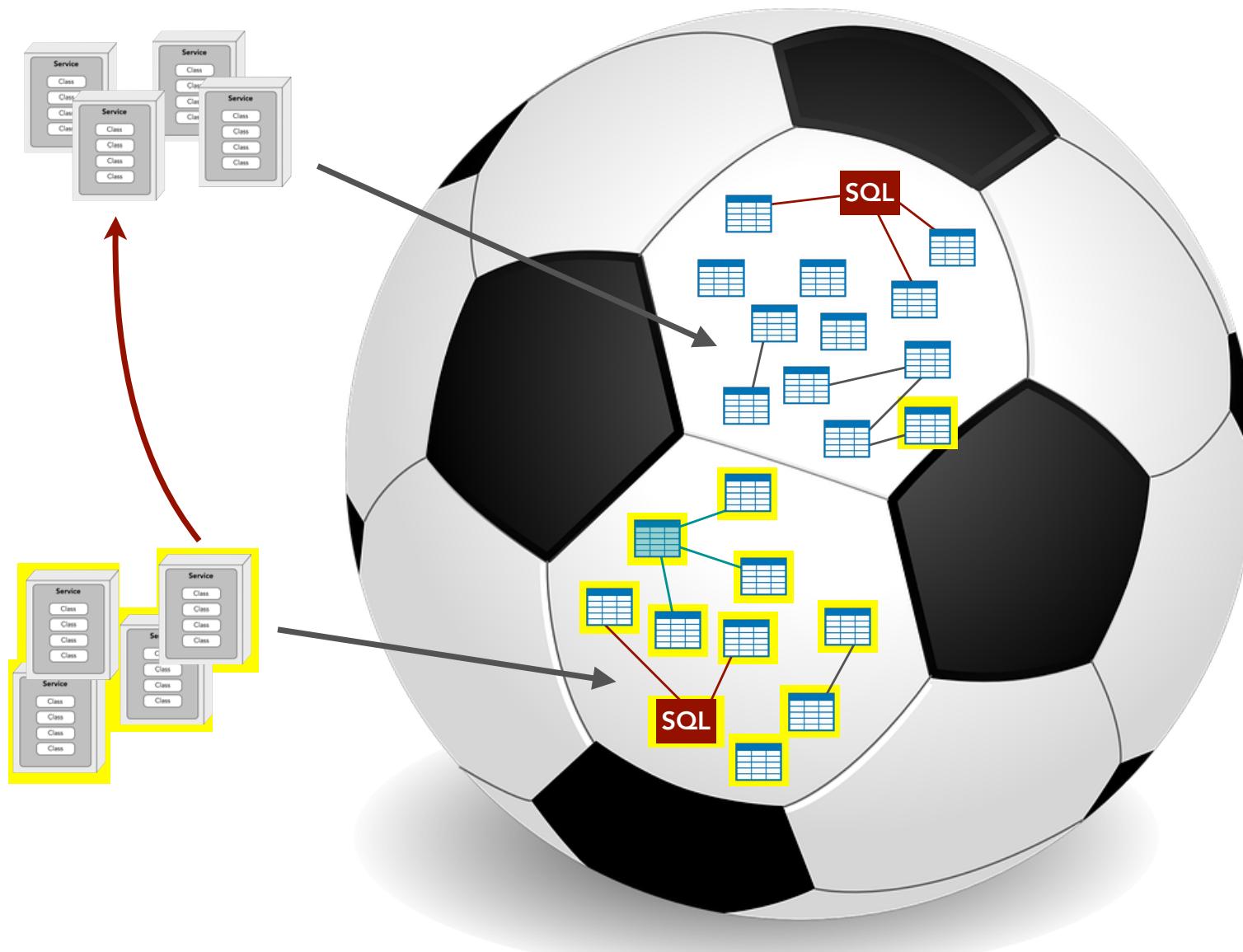
data domains



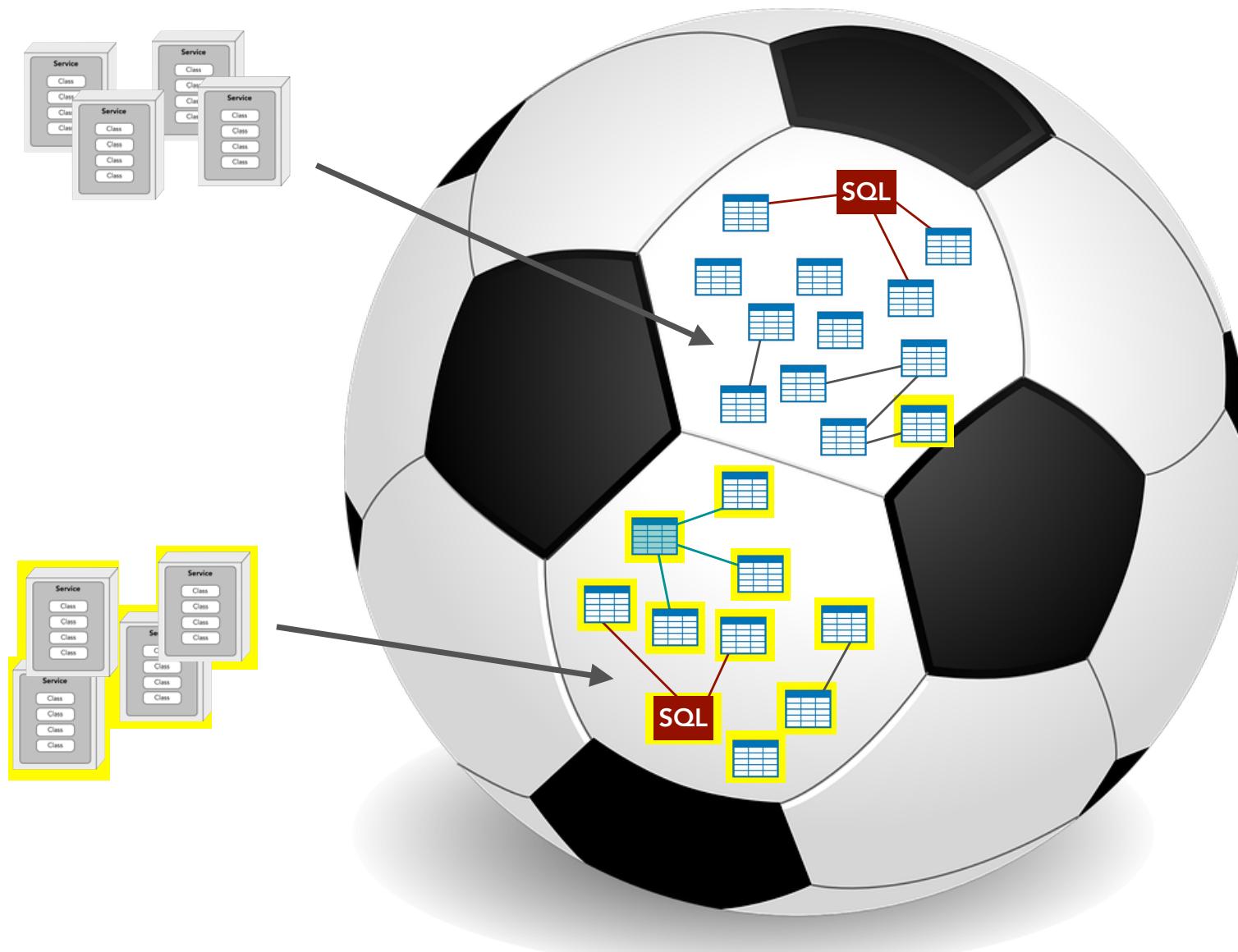
data domains



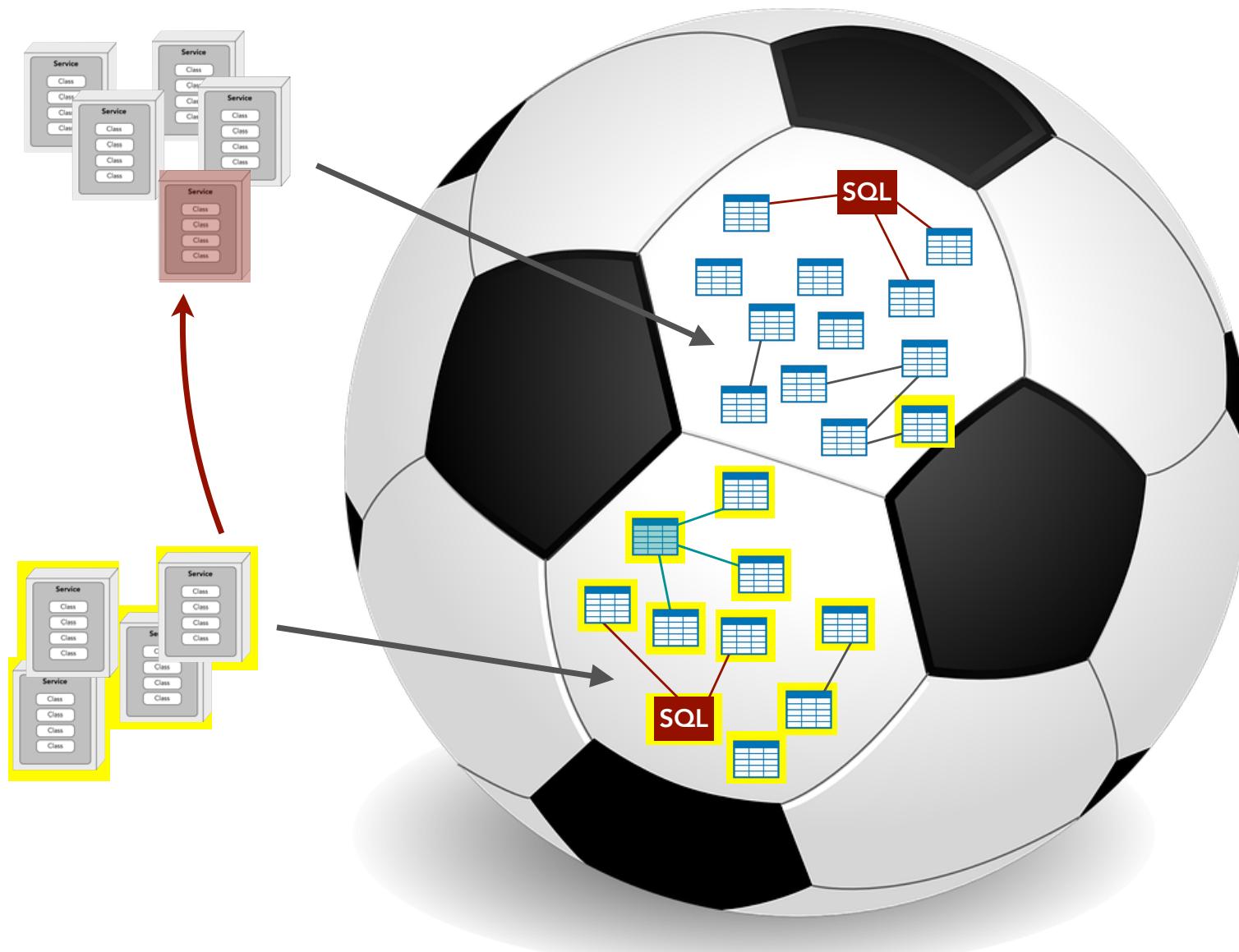
data domains



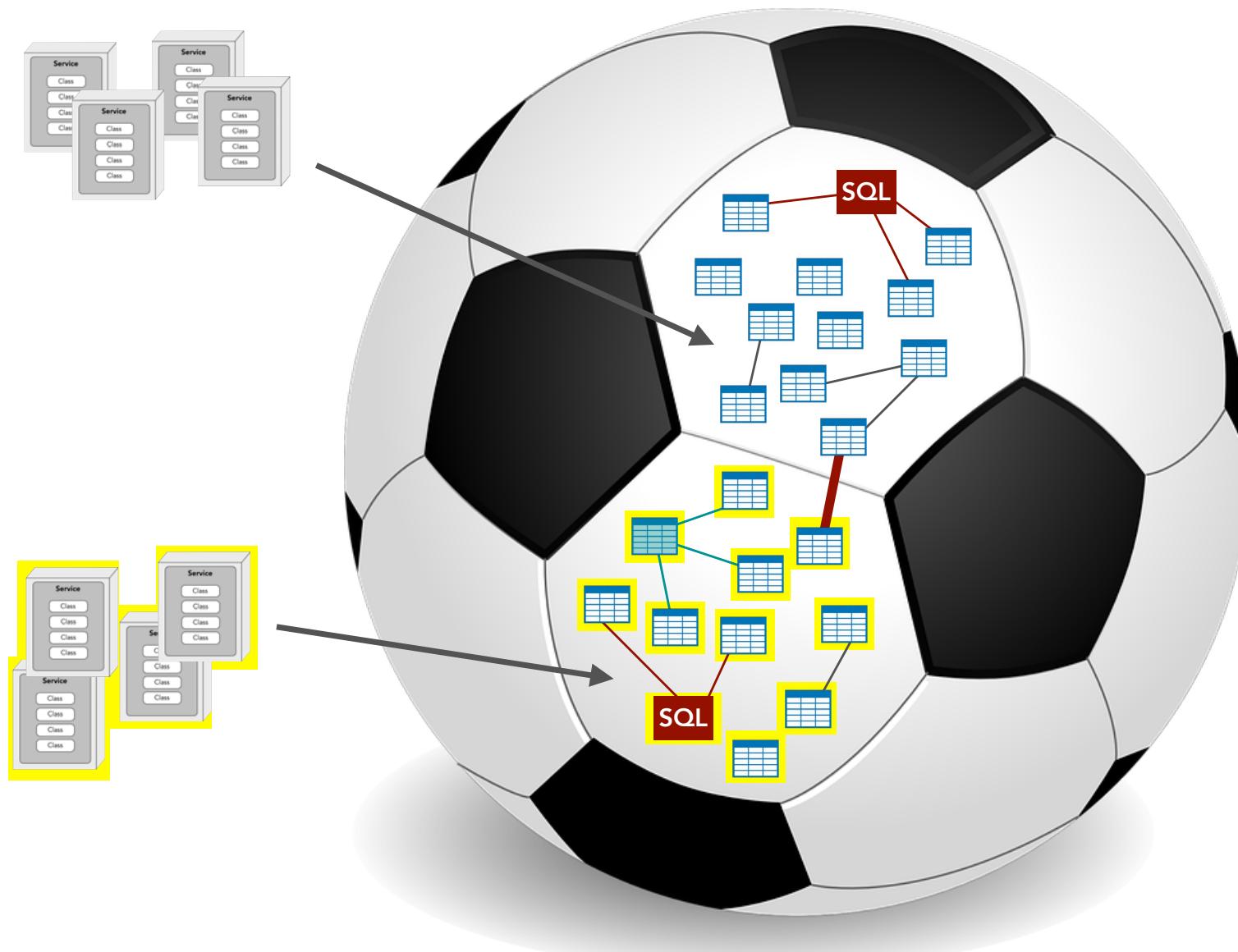
data domains



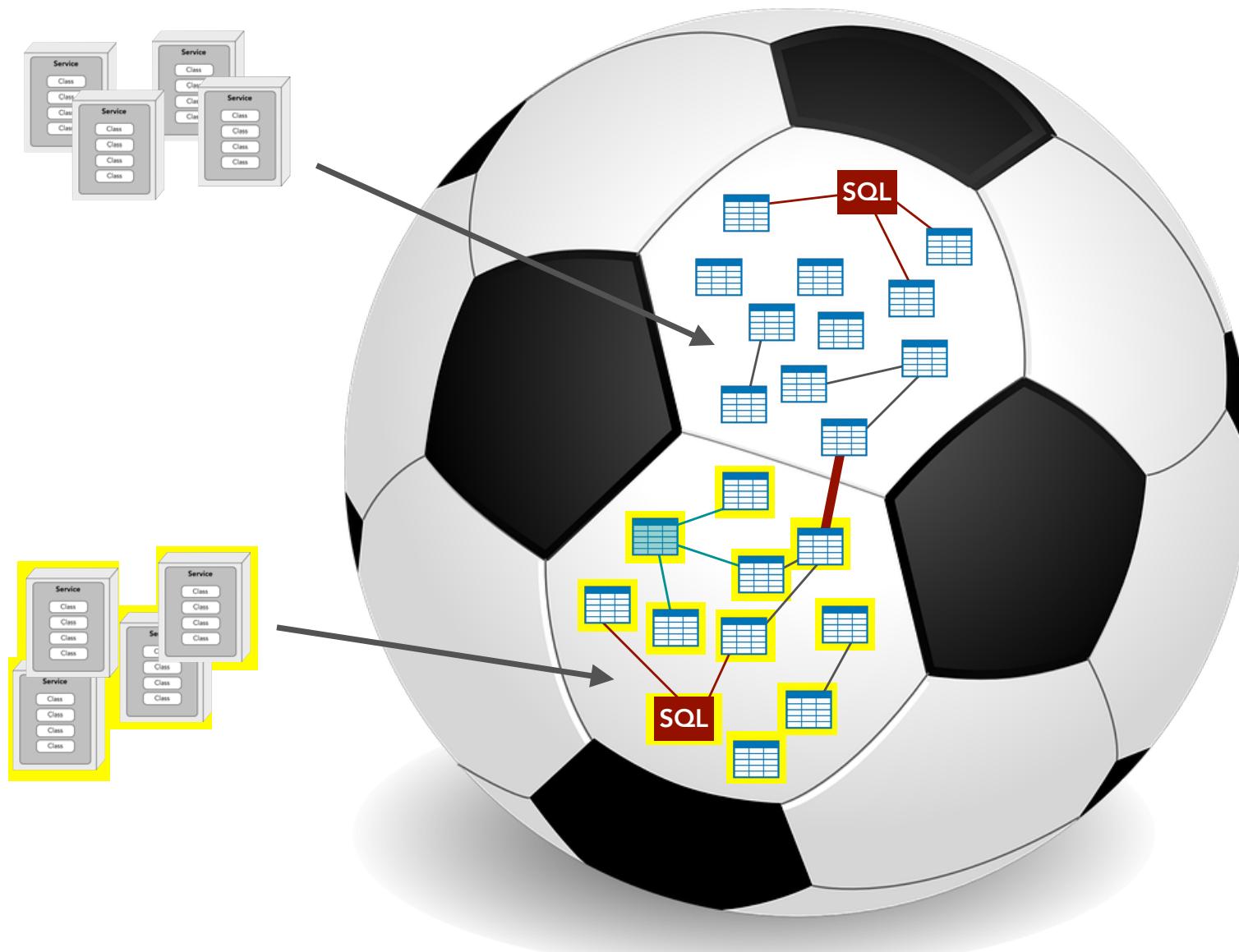
data domains



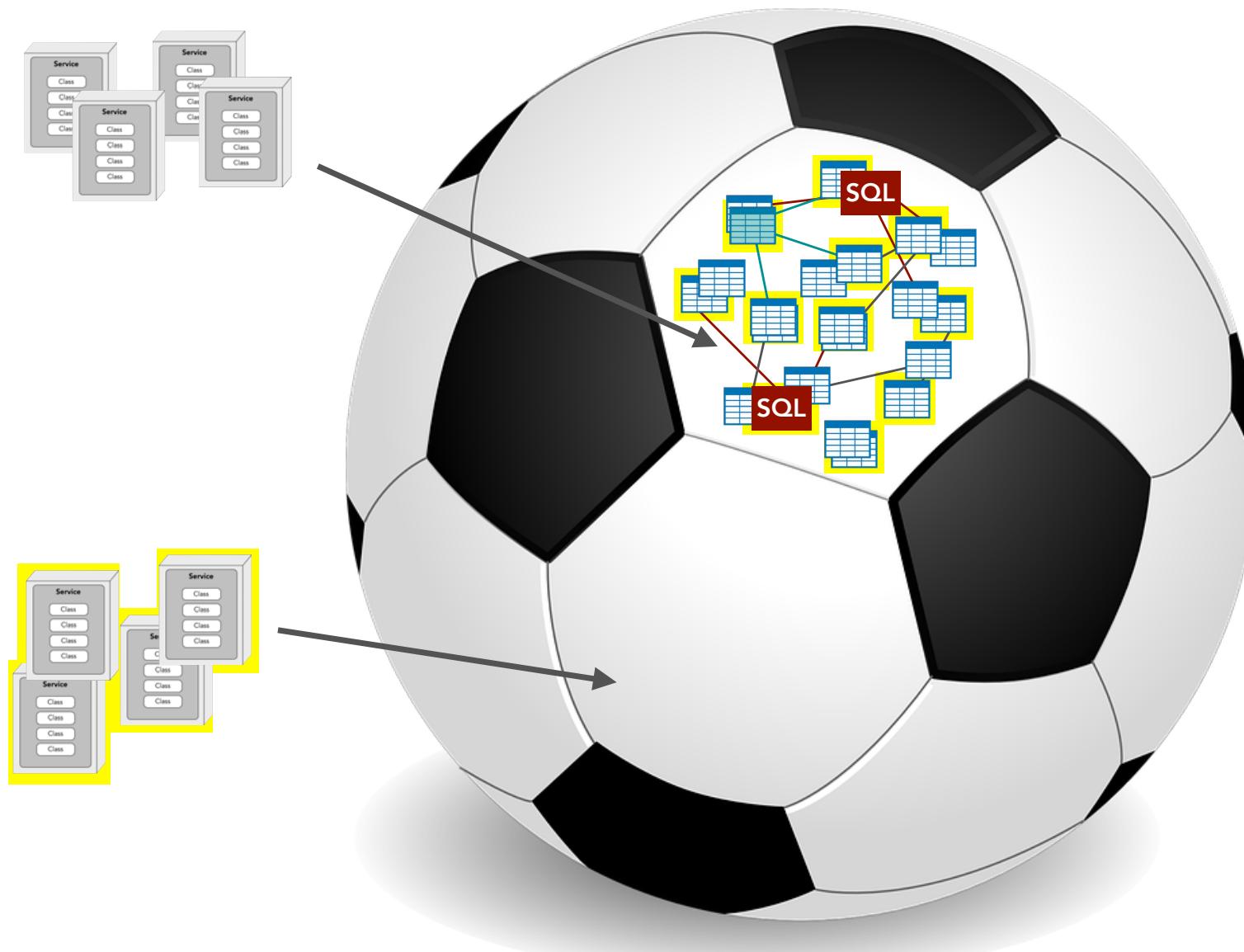
data domains



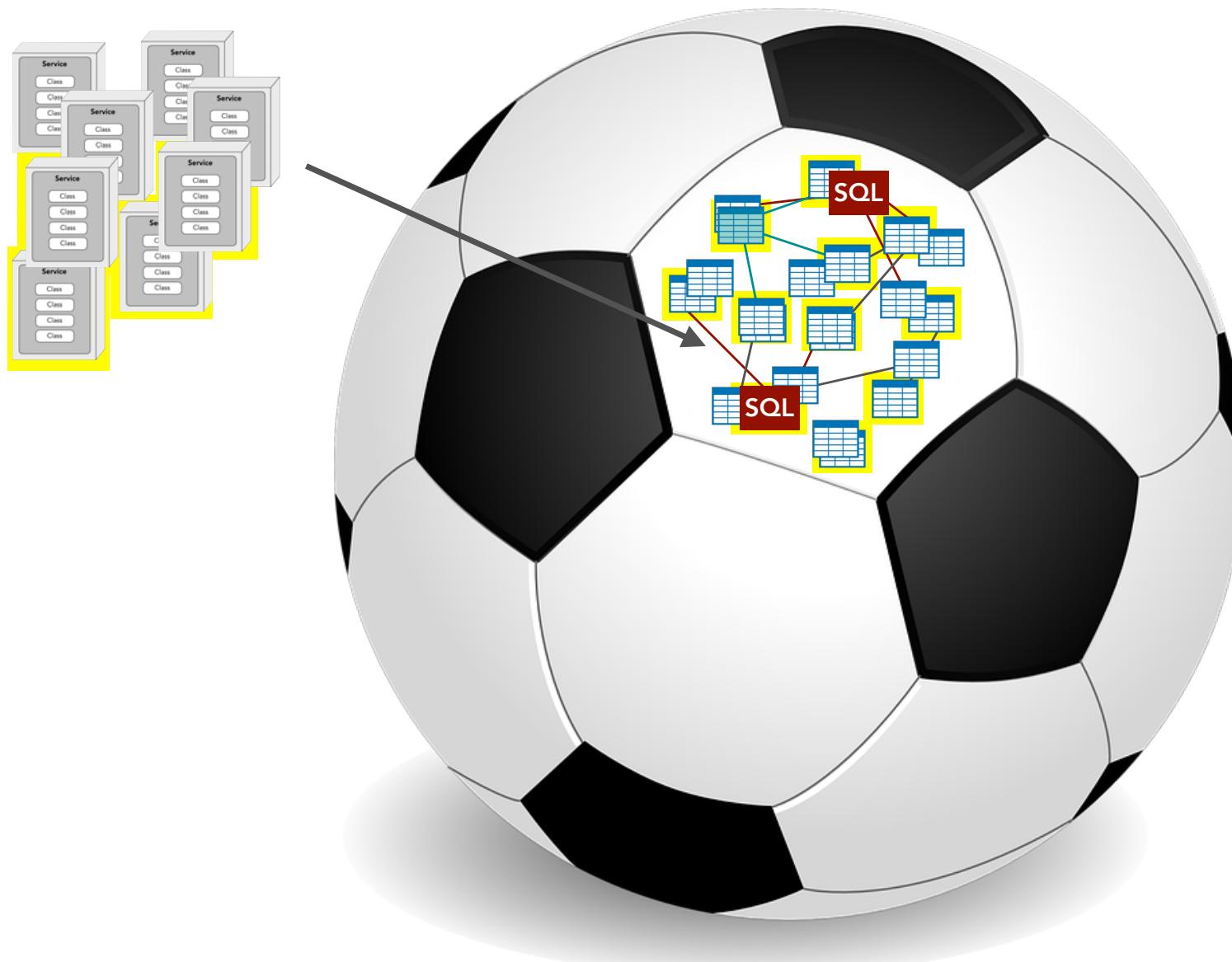
data domains



data domains



data domains

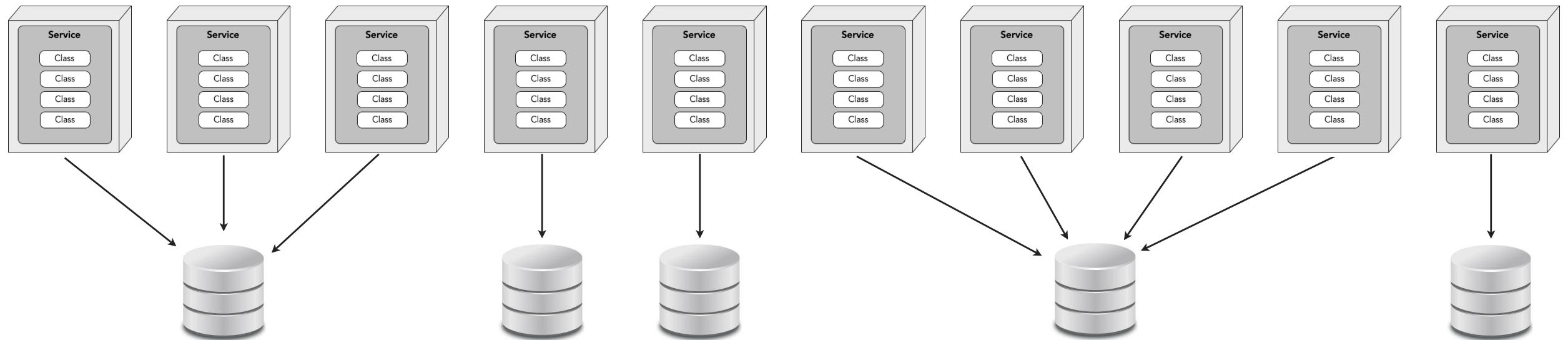


data domains

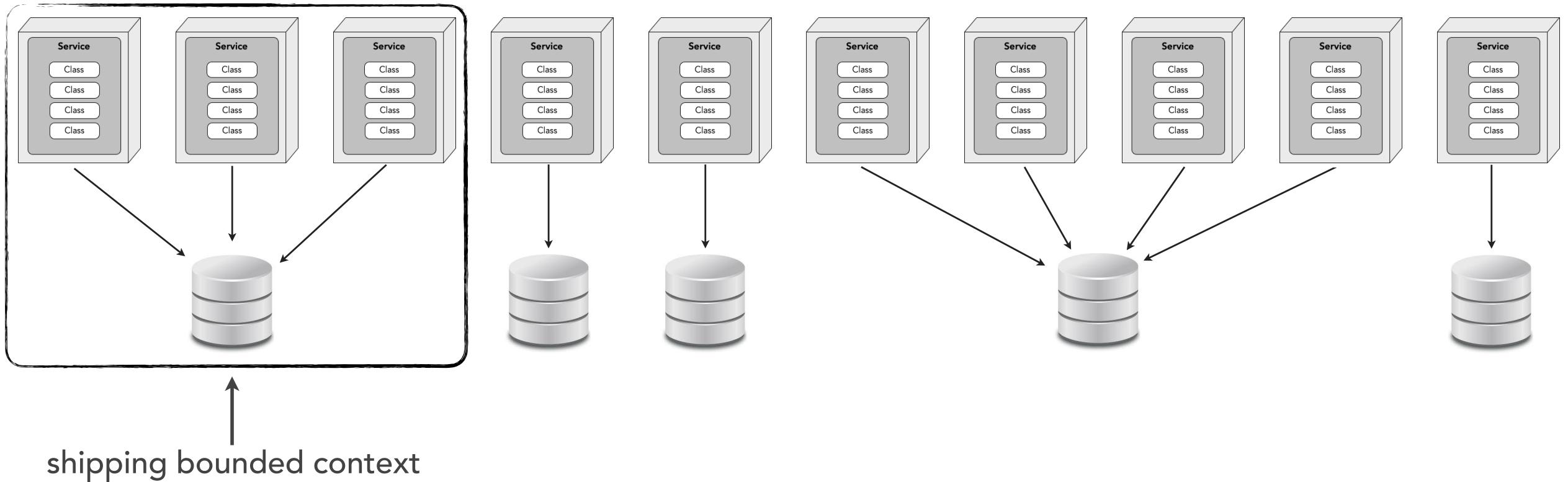


Managing Broad Bounded Contexts

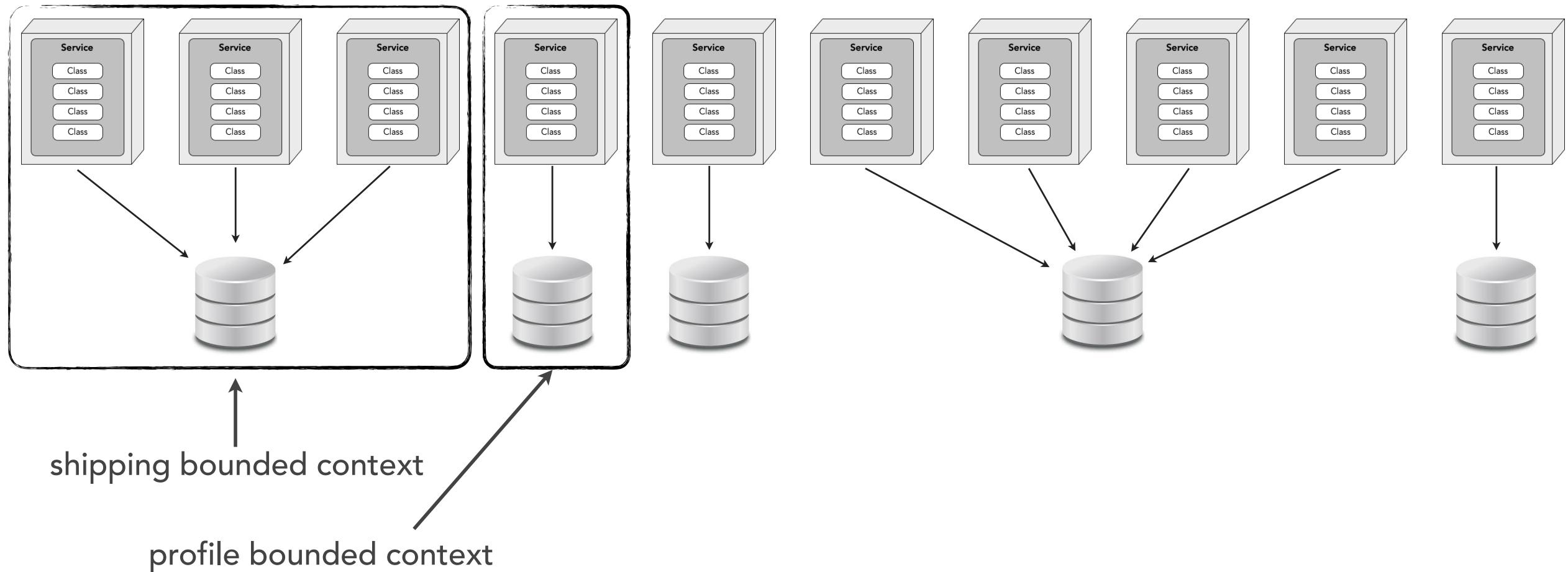
bounded context



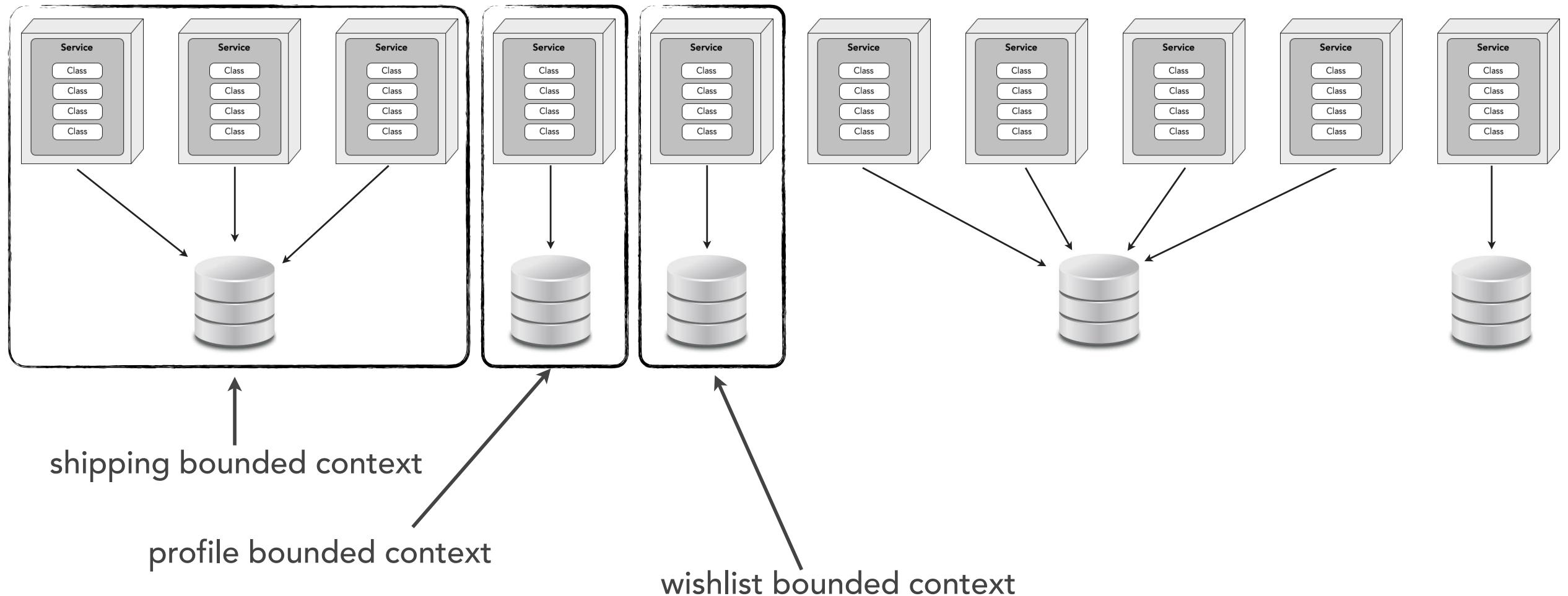
bounded context



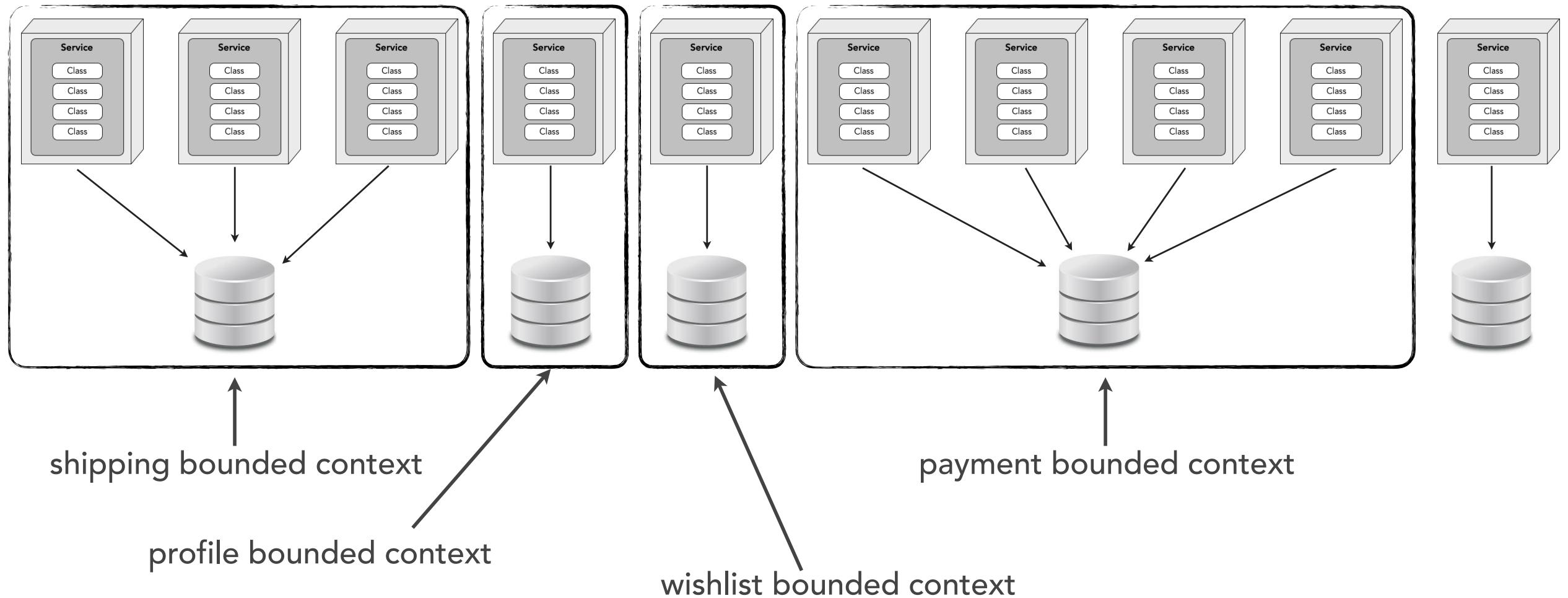
bounded context



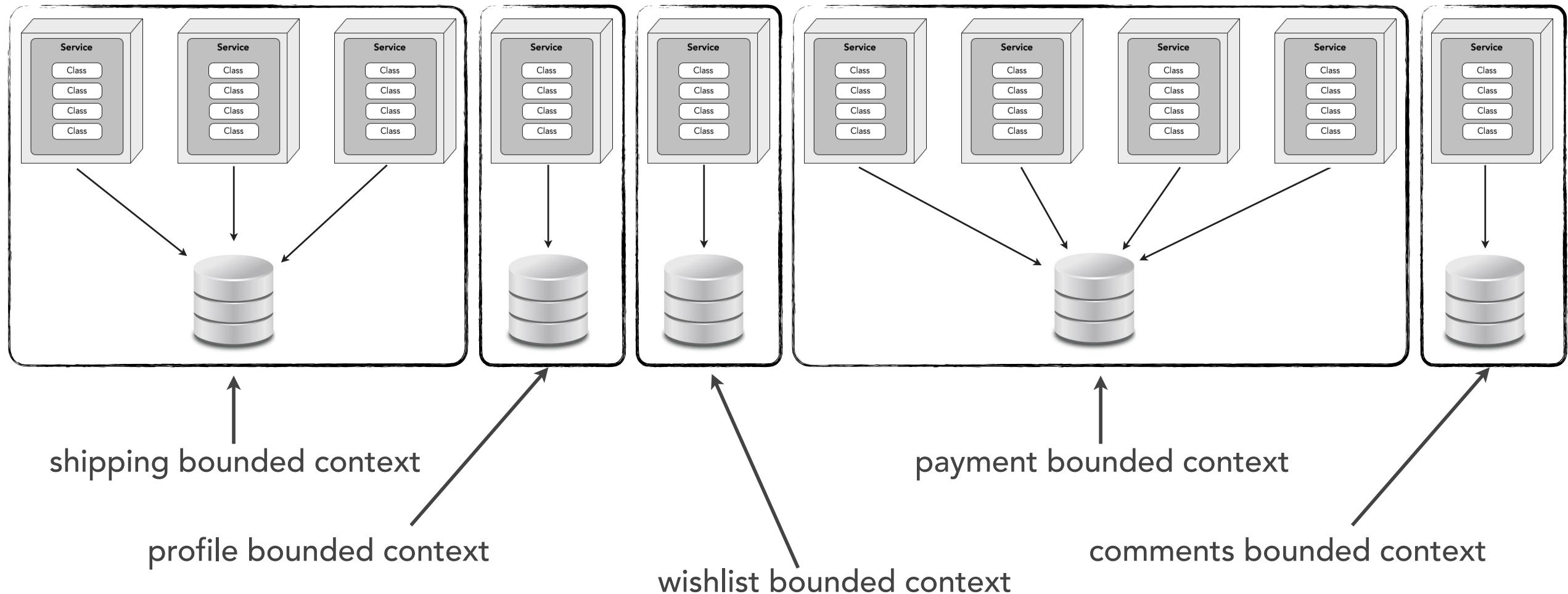
bounded context



bounded context



bounded context



bounded context

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface BoundedContext {
    public Context value();

    public enum Context {
        SHIPPING,
        PAYMENT,
        WISHLIST,
        COMMENTS,
        PROFILE
    }
}
```

bounded context

```
@ServiceEntrypoint  
@FunctionalService  
@BoundedContext(Context.PAYMENT)  
public class CreditCardServiceAPI {  
    ...  
}
```

bounded context

```
@ServiceEntrypoint  
@FunctionalService  
@BoundedContext(Context.PAYMENT)  
public class CreditCardServiceAPI {
```

...

```
}
```

```
@ServiceEntrypoint  
@FunctionalService  
@BoundedContext(Context.PAYMENT)  
public class GiftCardServiceAPI {
```

...

```
}
```

bounded context

```
@ServiceEntrypoint  
@FunctionalService  
→ @BoundedContext(Context.PAYMENT)  
public class CreditCardServiceAPI {  
    ...  
}  
  
@ServiceEntrypoint  
@FunctionalService  
→ @BoundedContext(Context.PAYMENT)  
public class GiftCardServiceAPI {  
    ...  
}
```



Part II:

Putting Things Back Together

Leverage modularity, but beware of granularity.
—Mark Richards



Data Mesh

Analytical Data Management Architecture

Sysop Squad

The electrics giant is continuously looking to optimize its business. They are intending to use the data that have available to them about their experts, incoming calls, and quality of services over time to make improvements. One of the early improvements they are looking to make is to have an optimized supply of experts at hand to respond to their customers needs. They are looking to create reports on the skillsets in high demand, tailor and run experts trainings for areas of improvements, and hire unique specializations that are needed.

Another area of dynamic optimization is based on location and routing, looking to dynamically allocate experts to locations that see a sudden rise in calls due to power outages, or concentration of businesses, etc.

Requirements:

- The enhancements to collect data, generate reports, as well as train and use machine learning models are built as extensions / modifications to the existing systems and architecture in place.
- While the electric giants is hypothesizing on two initial use cases - optimized resource management and routing - they are hoping to evolve and improve their business extending the use of their data to optimize other aspects of their business.
- Privacy and protection of their customers personal information is a must.

Kata Exercise - Analytical Data Architecture

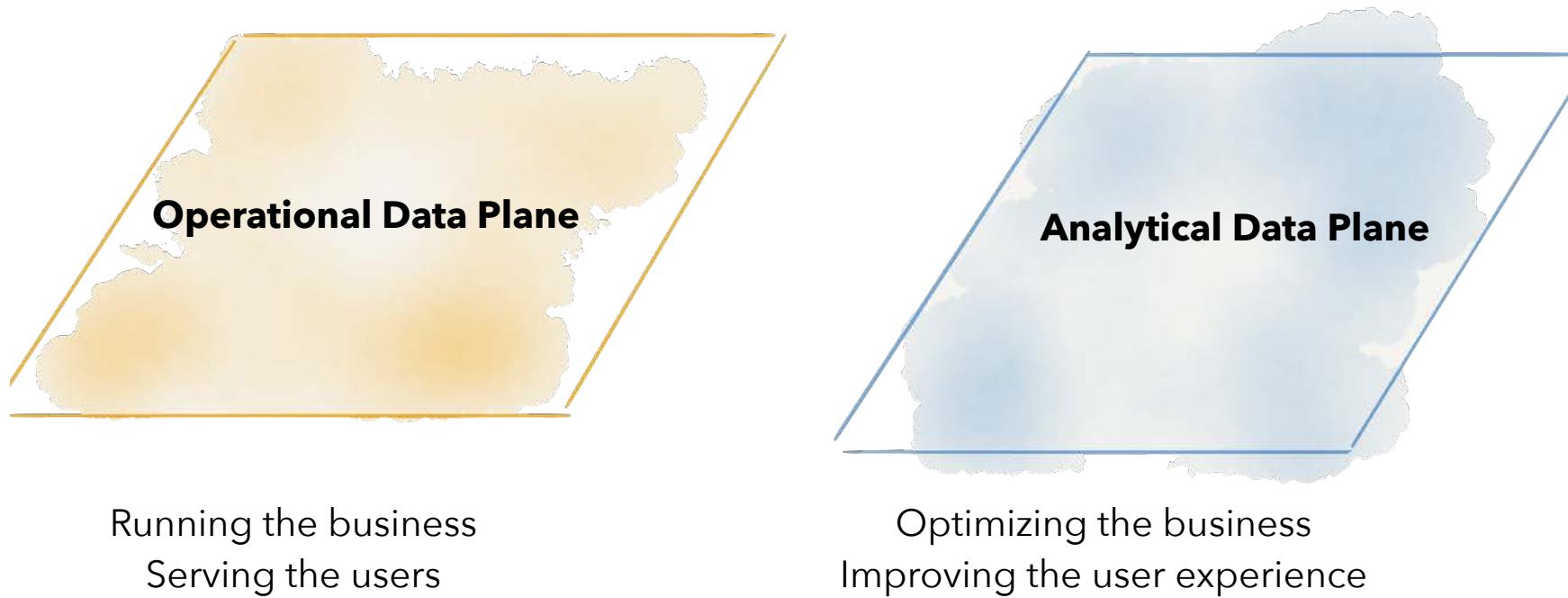
Design an analytical data architecture that can meet the requirements and data aspirations that *Penultimate Electronics* has.

In your architecture:

- Demonstrate how you decompose the components of your architecture; How you decompose analytical data.
- Demonstrate how you integrate the analytical plane with data plane
- Demonstrate the end to end integration of components, in order to address the use cases of (a) experts skillsets demand trends monthly reports, and (b) daily experts supply planning datasets.

Operational and Analytical Data

“The great divide”



Operational and Analytical Data

“The great divide”



Operational Data Plane

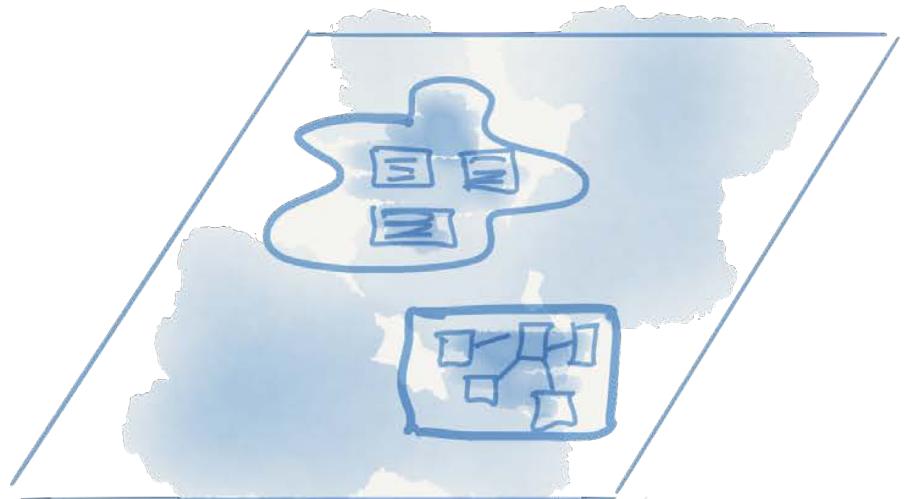
Running the business

Serving the users

- Optimized for application/services
- Captures current state of applications
- Support transactional CRUD operations
- Accessed through APIs - Data on the inside
- Each service or application choose their type:
Polyglot - graph database, no-sql document store, relational database, etc.

Operational and Analytical Data

“The great divide”



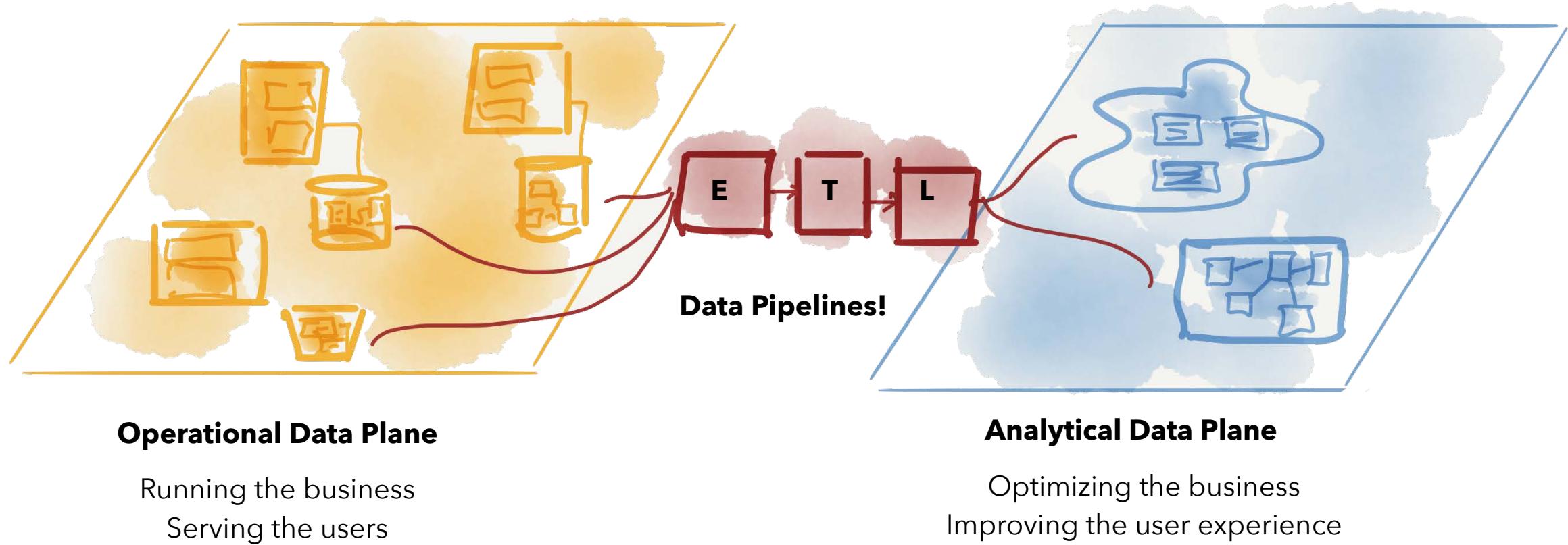
Analytical Data Plane

Optimizing the business
Improving the user experience

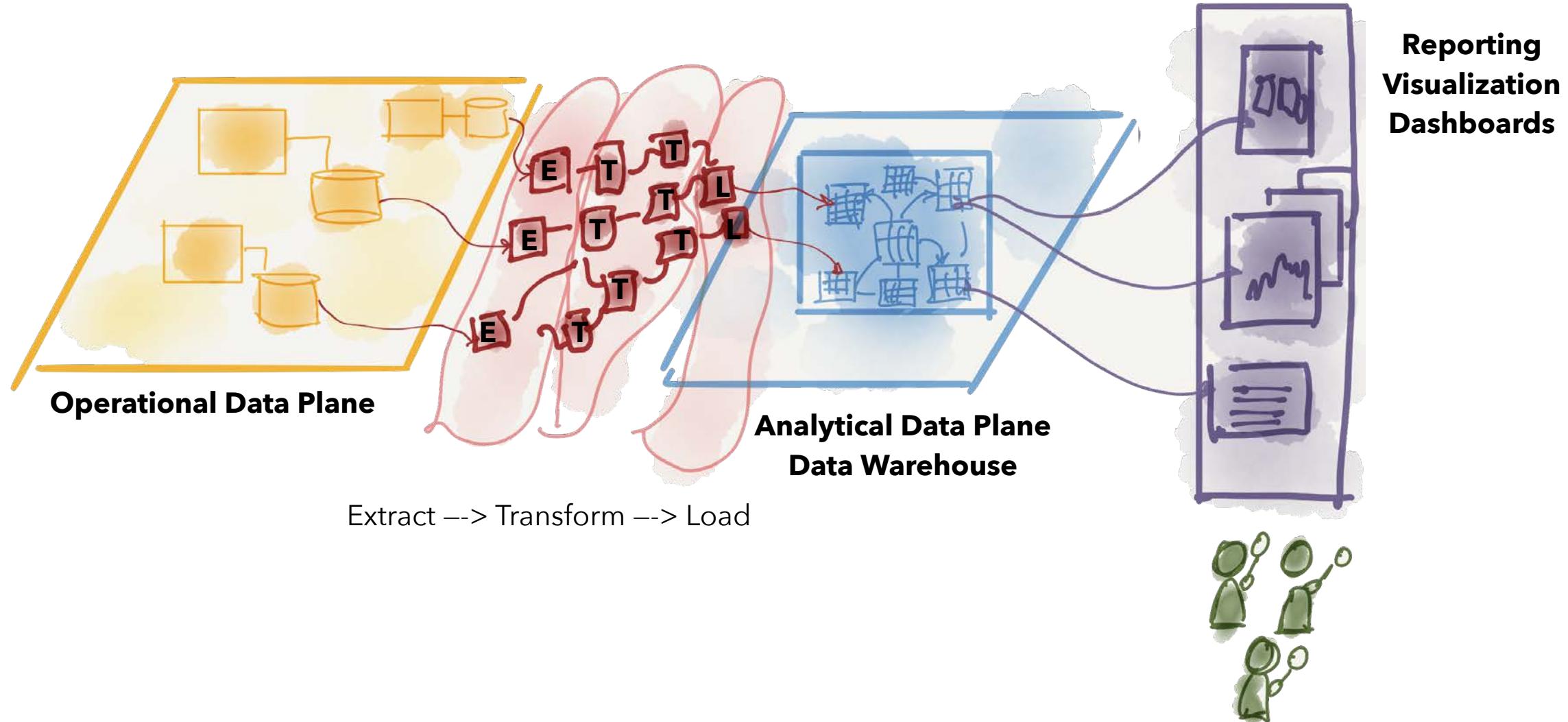
- Optimized for analytical logic - i.e. create reports
- Optimized for intelligent modeling - i.e. machine learning training
- Temporal
- Data on the outside - events, files, tables
- Polyglot e.g. object store, big table, streams

Operational and Analytical Data

“The great divide”



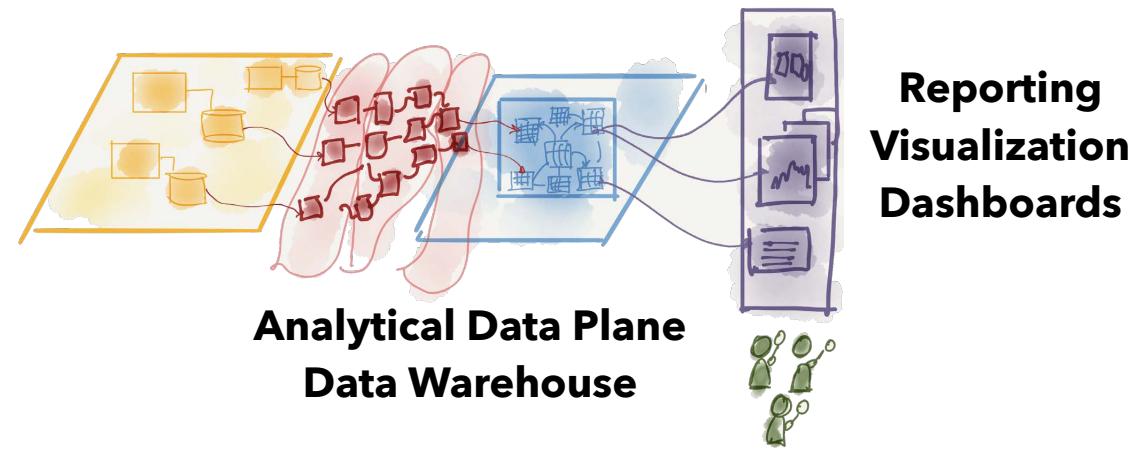
Analytical Data Warehouse Architecture



(Analytical) Data Warehouse Architecture

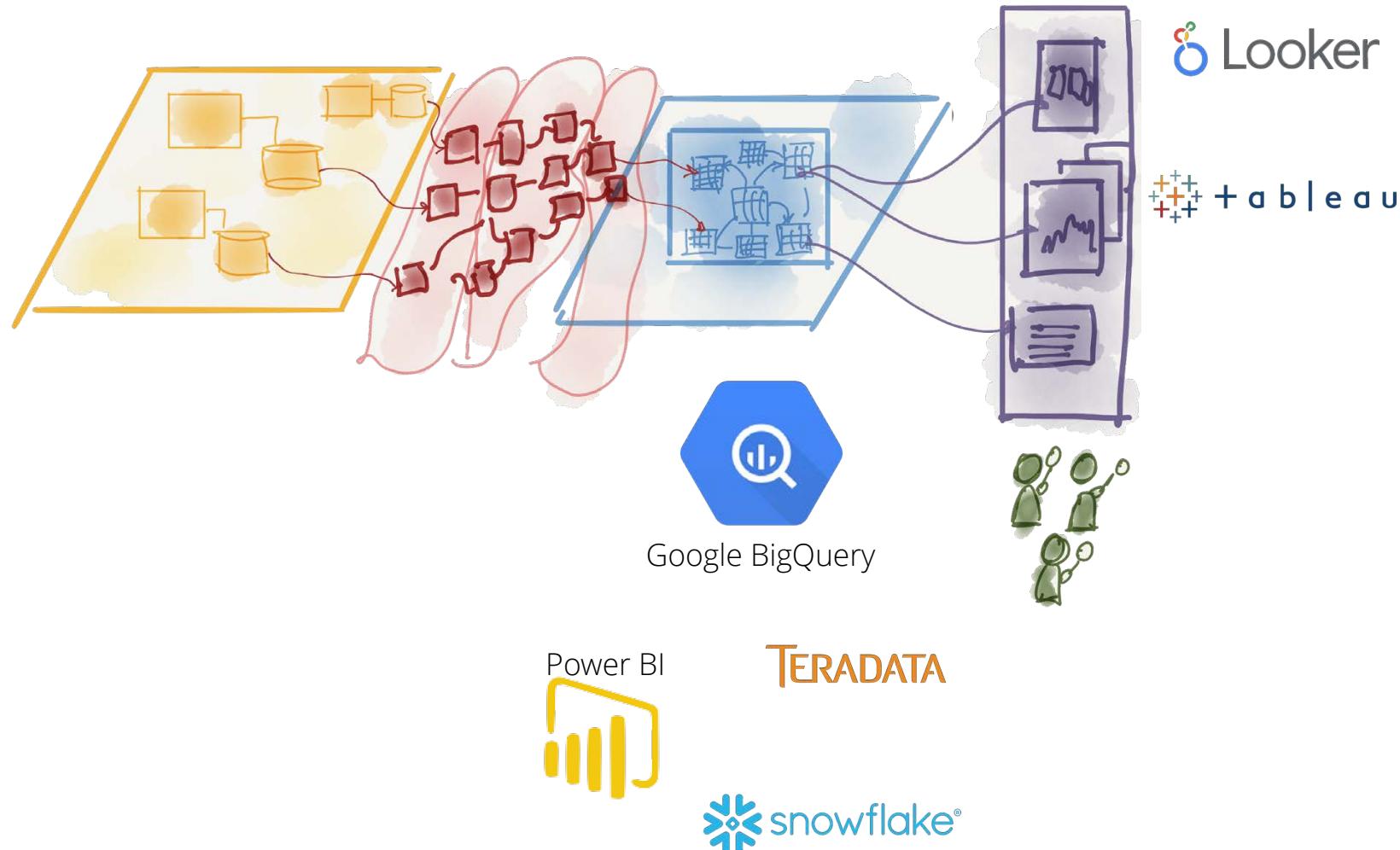
“Characteristics”

- Data **extracted** from many sources - *pathological coupling*
- Data is **transformed** to single multi-dimensional schema
- Data is **loaded** into warehouse tables
- Warehouse is a single stack technology - *monolithic*
- Warehouse is used by analysis and visualization layer
- Data analysts use the analysis layer
- Data analysts create business intelligence reports and dashboards
- Data in warehouse tables is accessed with SQL-ish interface

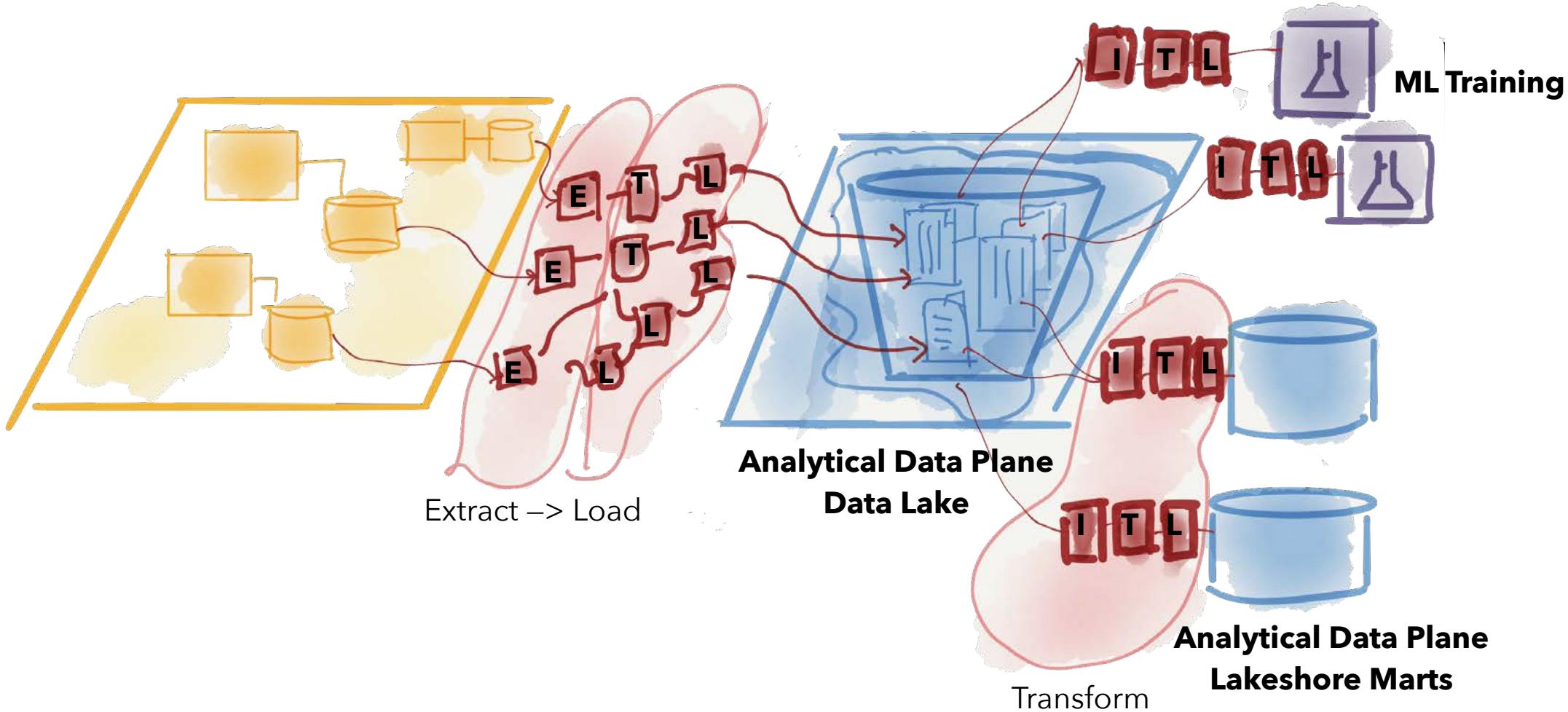


(Analytical) Data Warehouse Architecture

“Technologies”



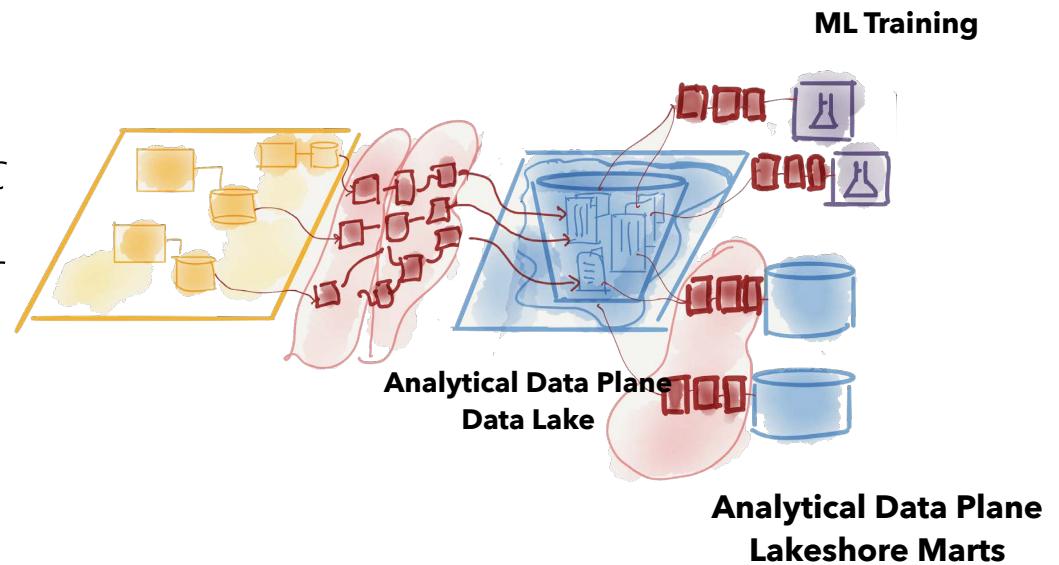
(Analytical) Data Lake Architecture



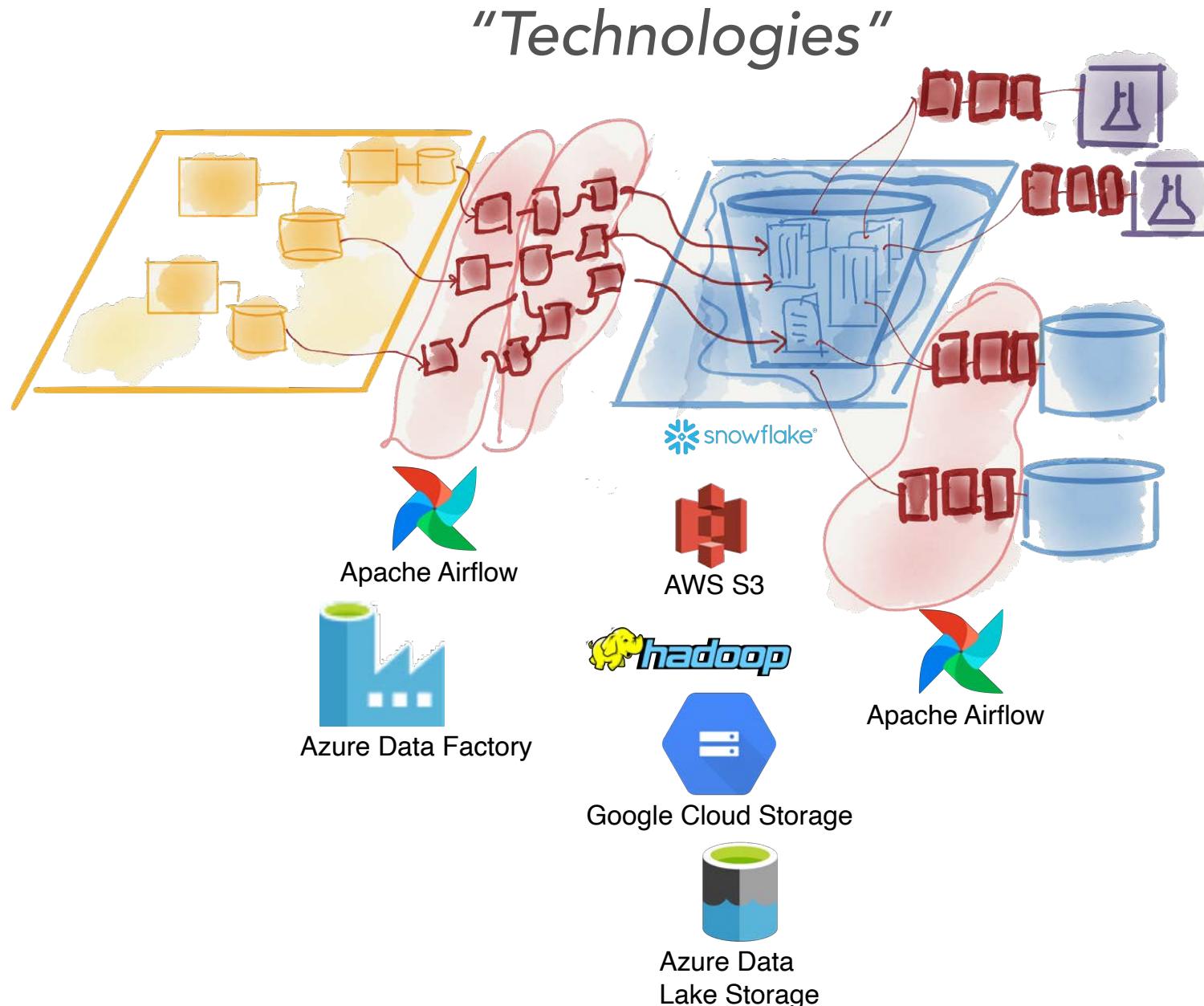
(Analytical) Data Lake Architecture

“Characteristics”

- Data is **extracted** from many sources
- Data is **loaded** into the lake in raw format
- Raw format includes semi-structured (e.g. JSON) files - *monolithic*
- Data scientists consume data from lake and **transform** for their usage
- Analysts and other data consumers **transform** data into their lakeshore databases for fit-for-purpose usages e.g. reports
- ML model training, data-driven apps both consume data from lake

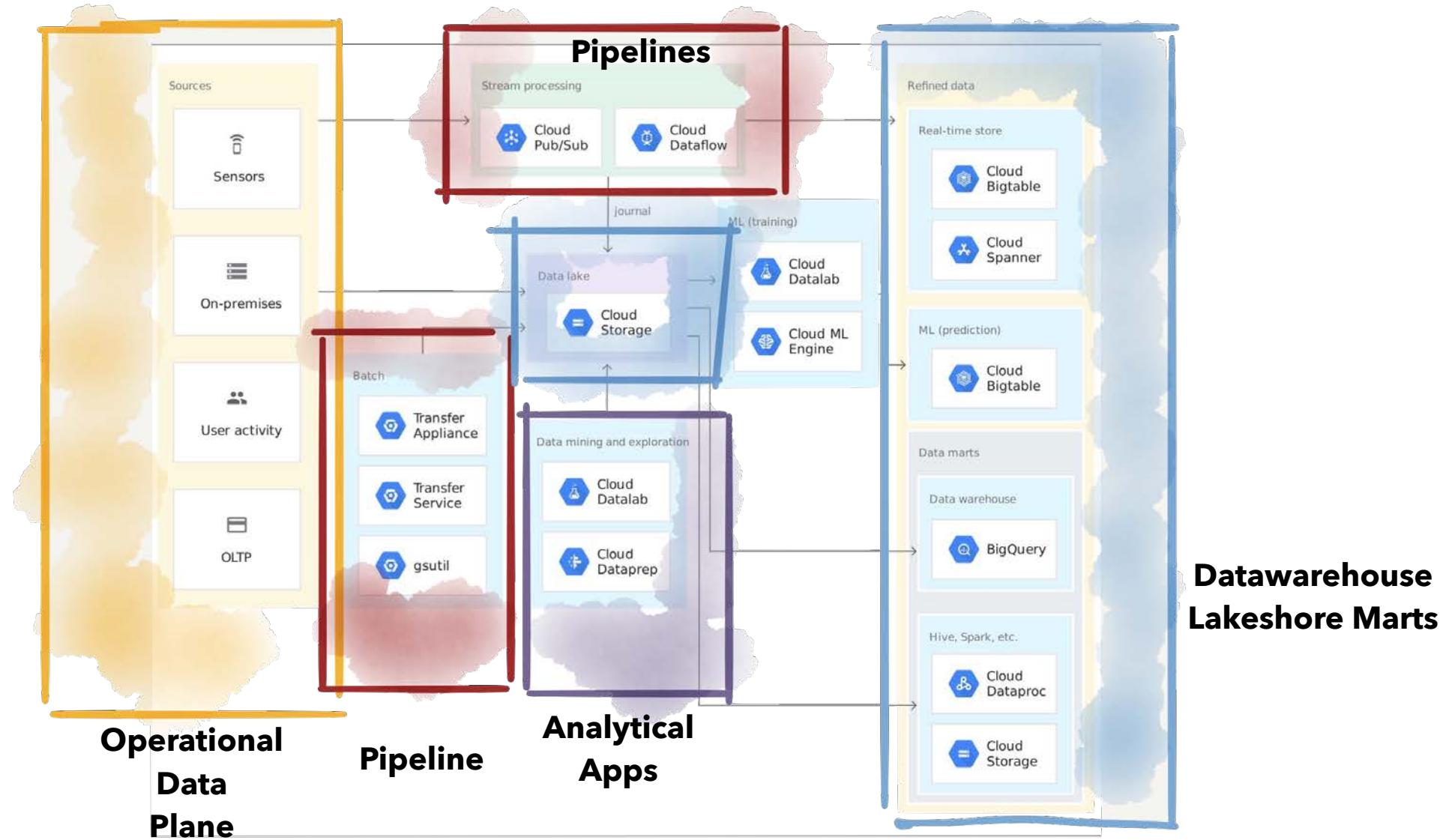


(Analytical) Data Lake Architecture



(Analytical) Data Lake Architecture On Cloud

“Solution Architecture”



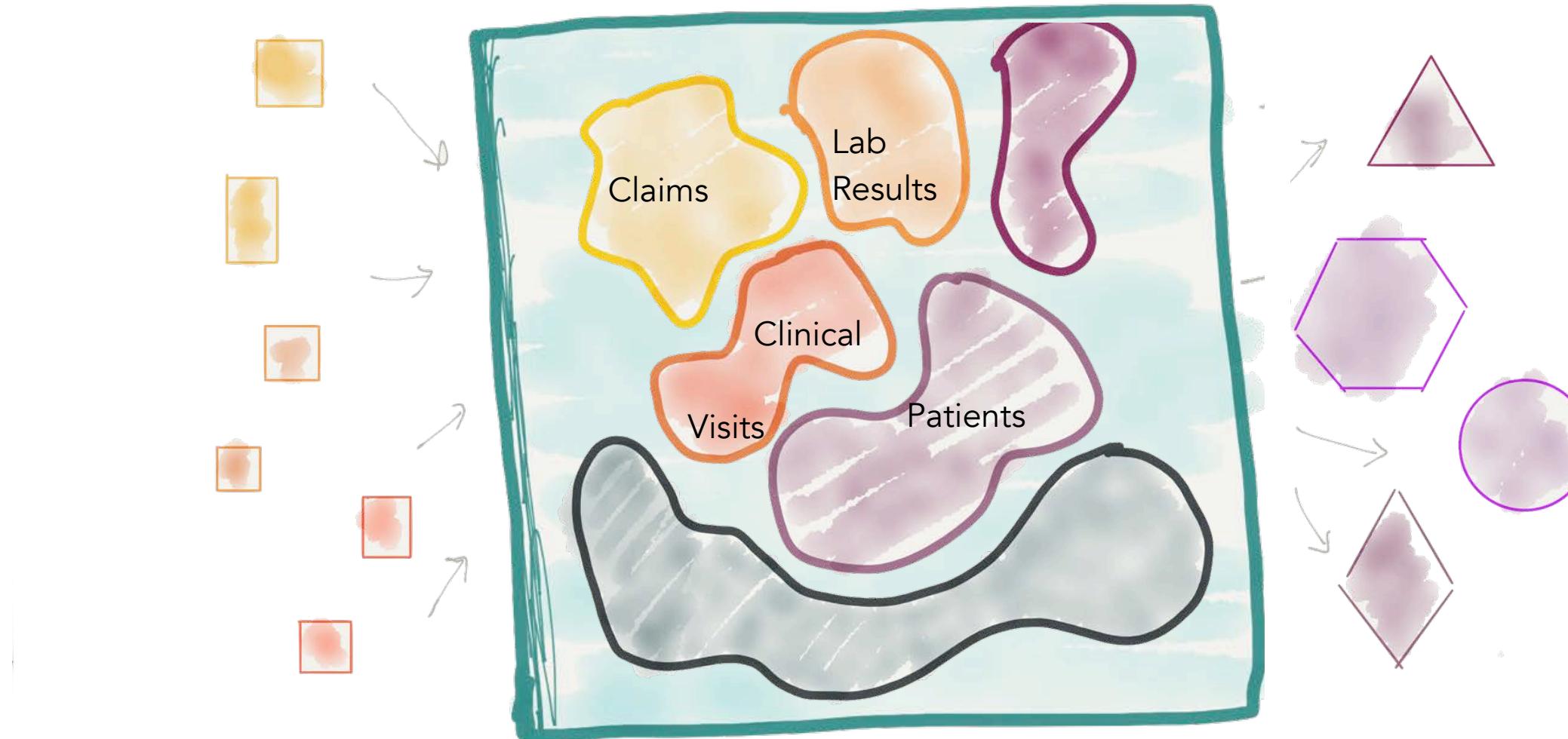
Data Lake/Warehouse Architectural Characteristics

"Centralized"



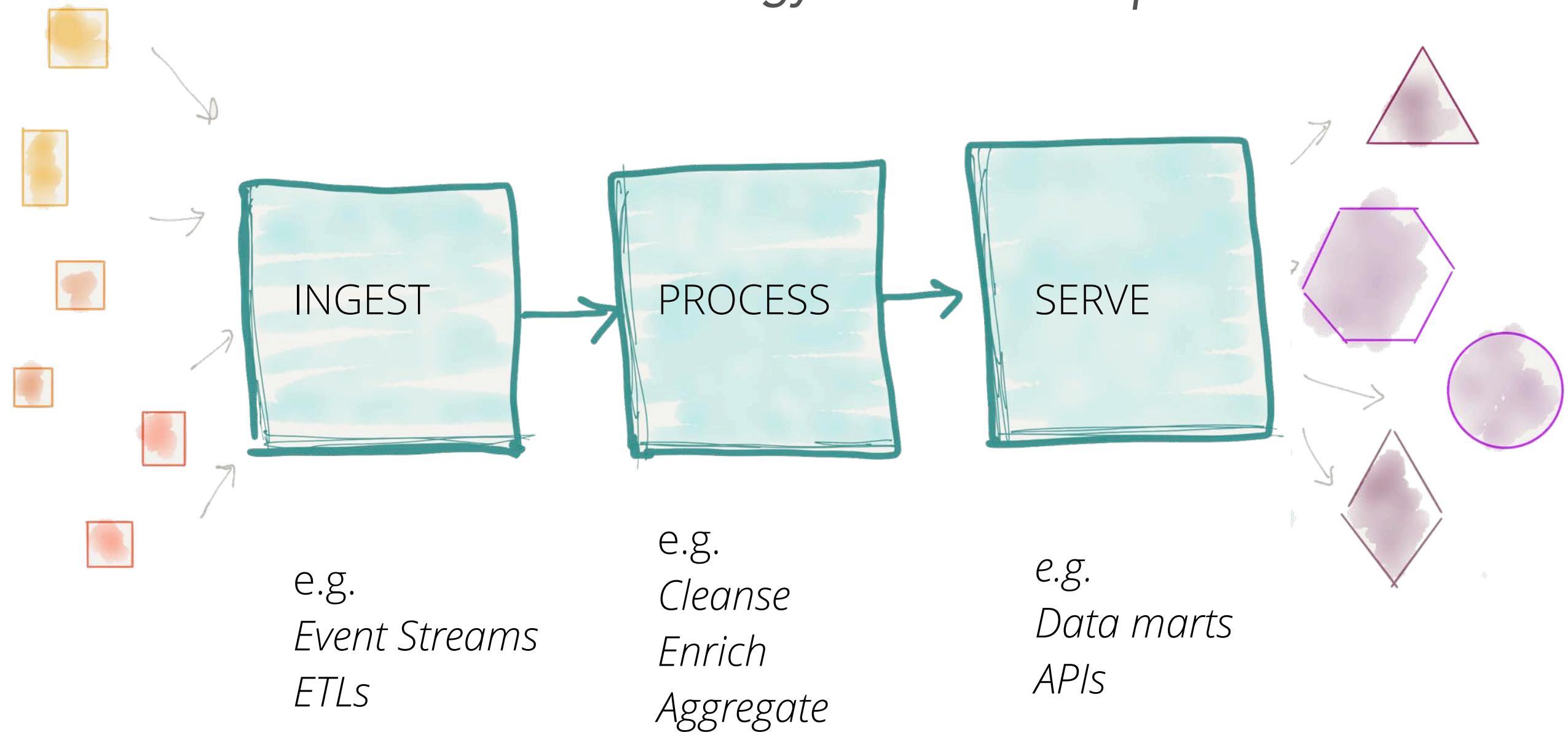
Data Lake/Warehouse Architectural Characteristics

"Monolithic"



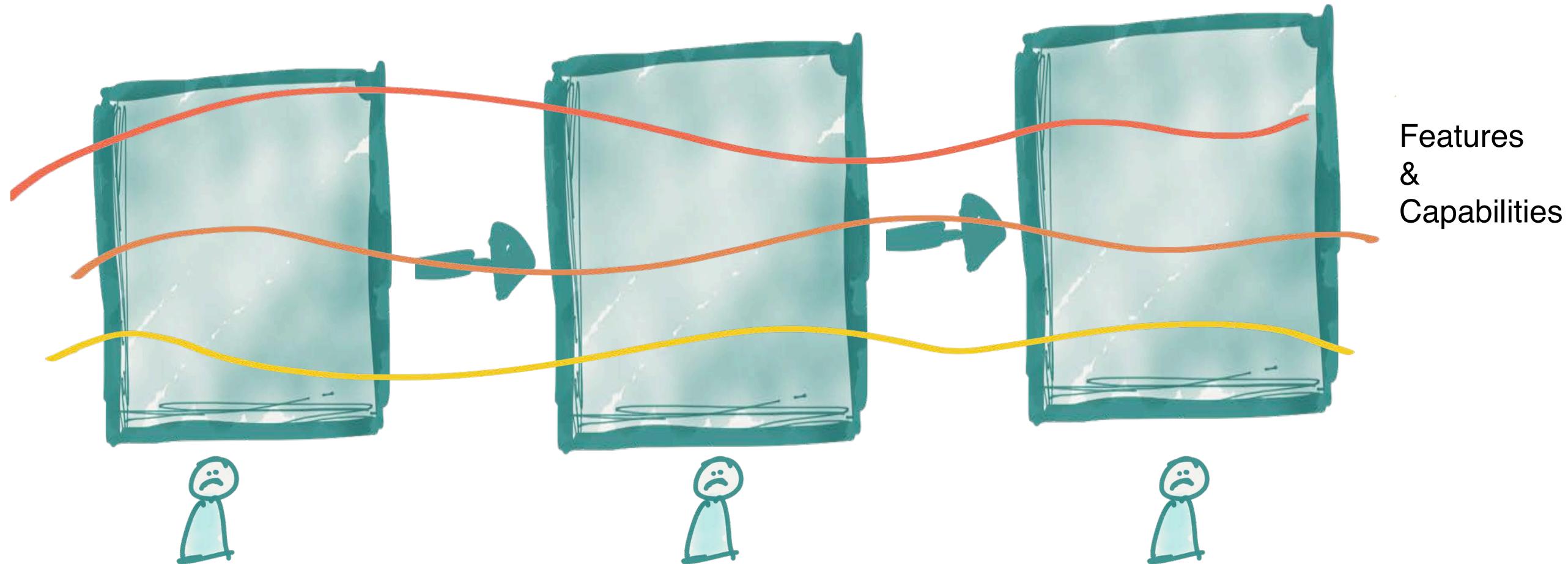
Data Lake/Warehouse Architectural Characteristics

“Functional and technology centric decomposition”



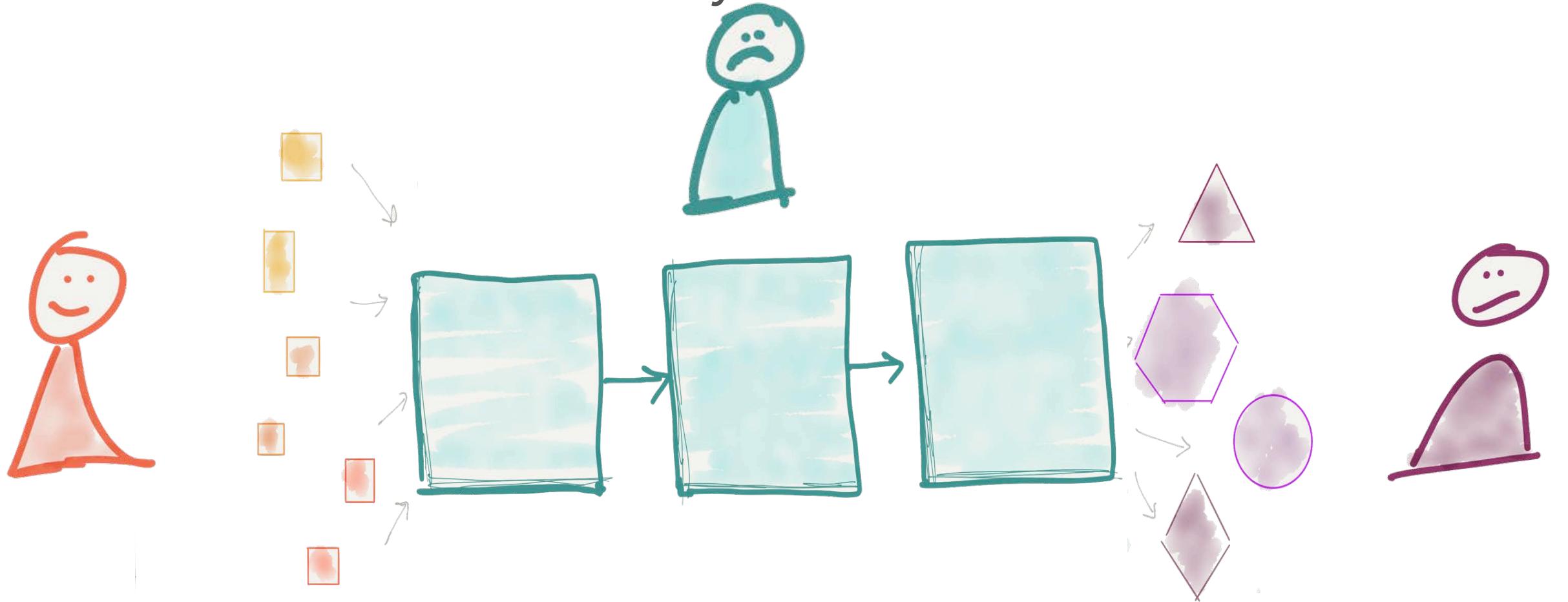
Data Lake/Warehouse Architectural Characteristics

"Tightly coupled - change orthogonal to the axis of change"



Data Lake/Warehouse Organizational Characteristics

“Functionally divided teams”

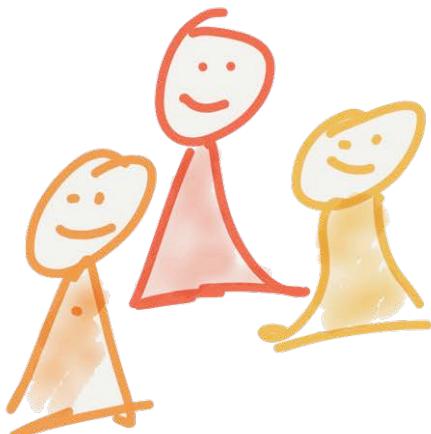


Domains' Operational systems
(plane) Teams

Data Scientists, BI teams, ...

Data Lake/Warehouse Organizational Characteristics

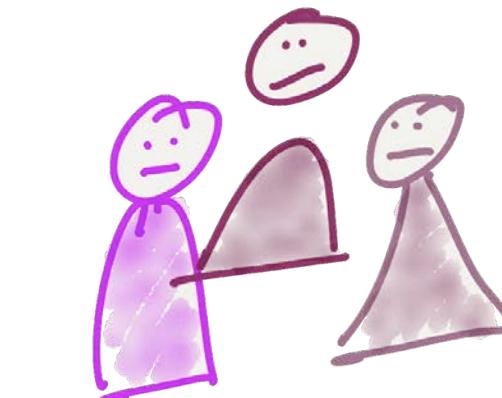
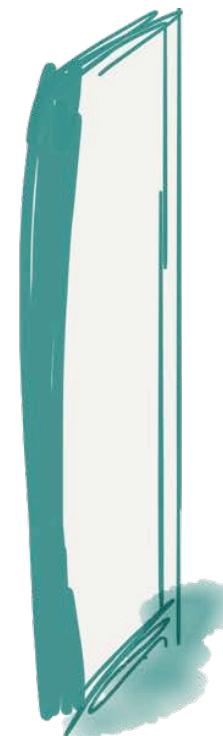
“Hyper-specialized silo”



Cross-functional
Domain oriented source teams



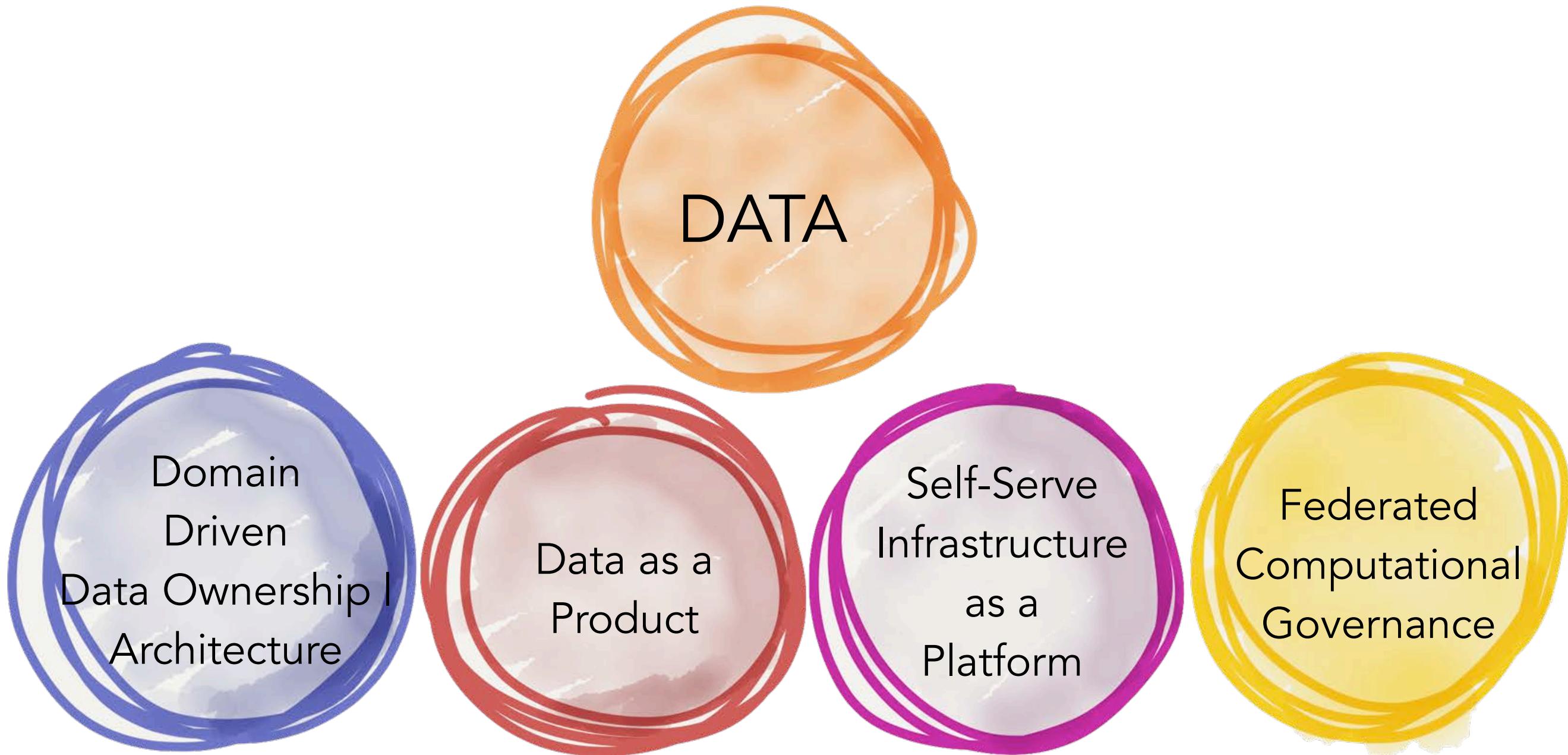
Hyper-specialized
Data & ML Platform Engineers



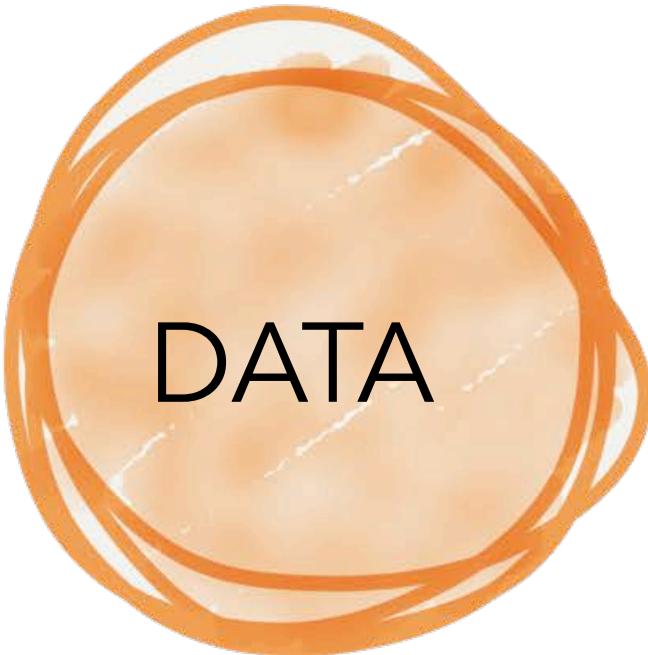
Cross-functional
Domain oriented consumer teams

What is the alternative?

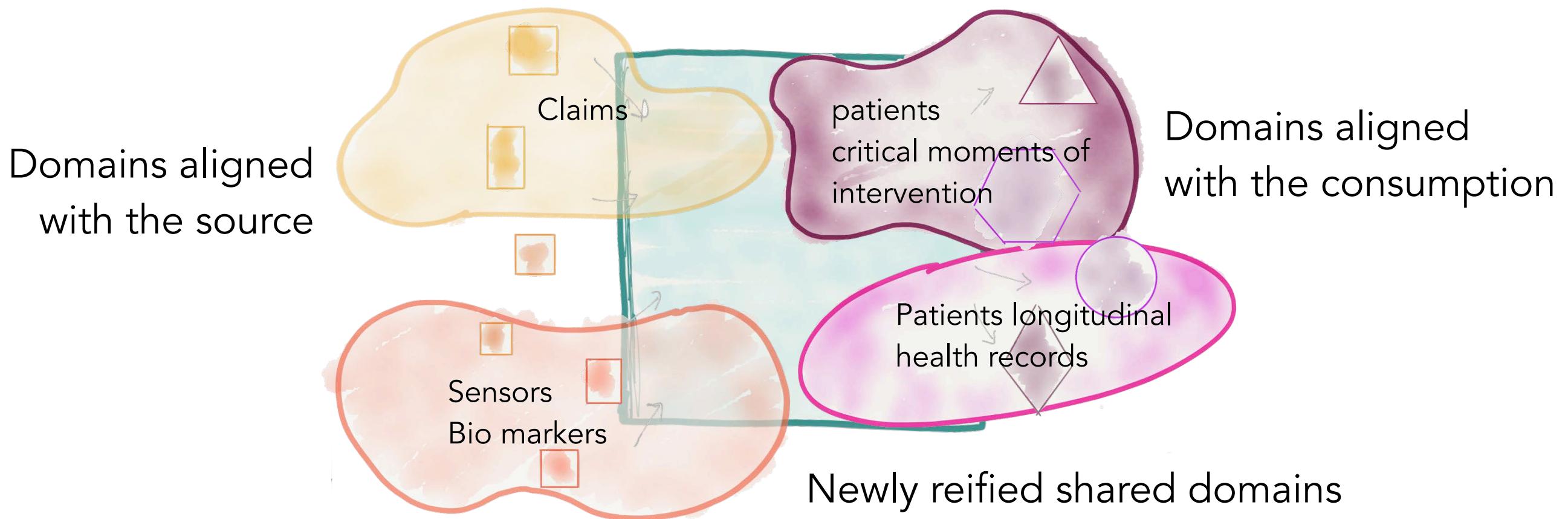
DATA MESH PRINCIPLES



DATA MESH PRINCIPLES



DOMAIN ORIENTED DATA DECOMPOSITION & OWNERSHIP



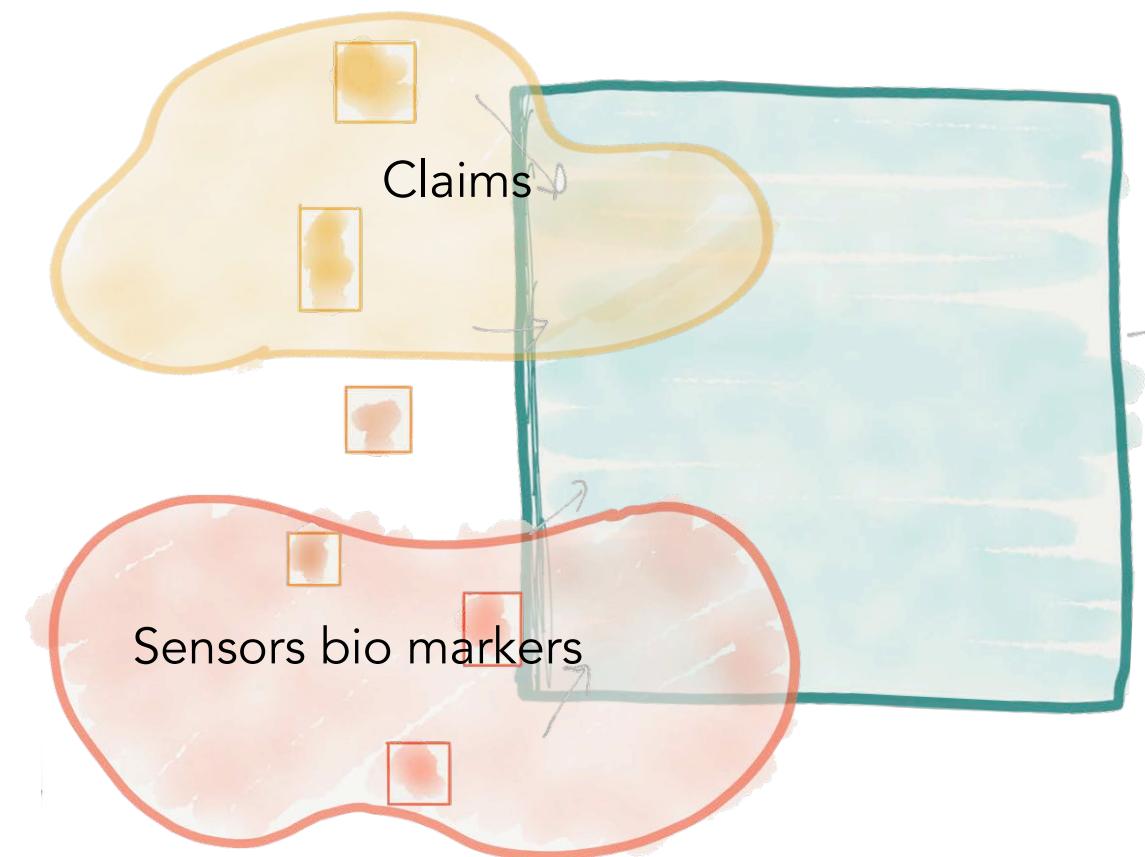
SOURCE ORIENTED (NATIVE) DOMAIN DATA

Facts & reality of business

Immutable timed events /
Historical snapshots

Change less frequently

Permanently captured



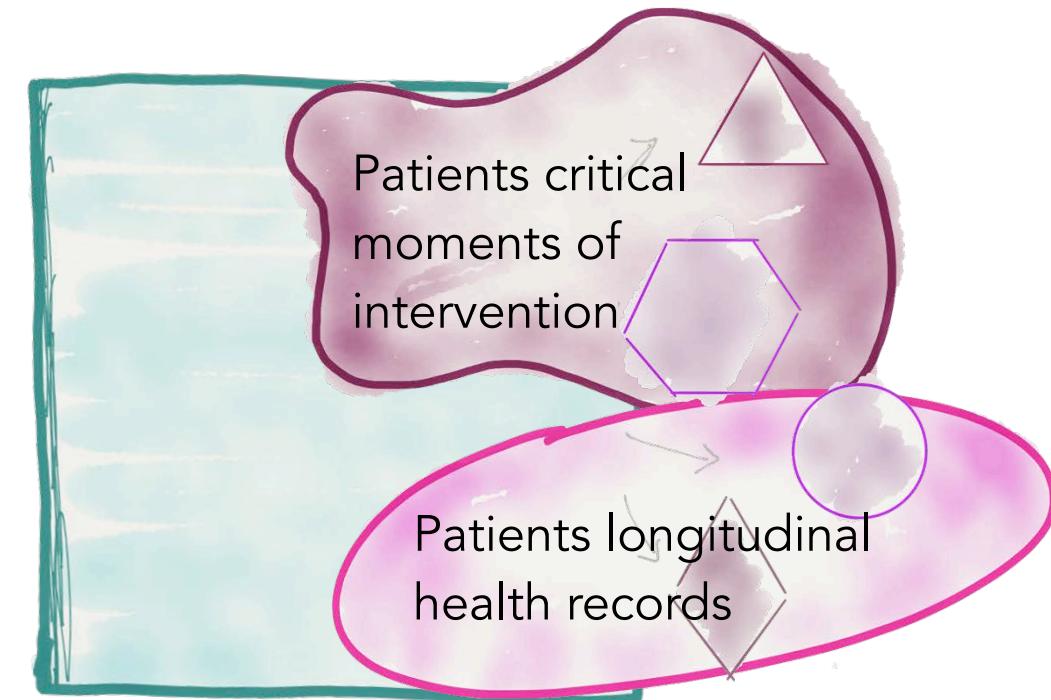
CONSUMER ORIENTED DOMAIN DATA

Fit for consumer purpose

Aggregation / Projections / Transformed

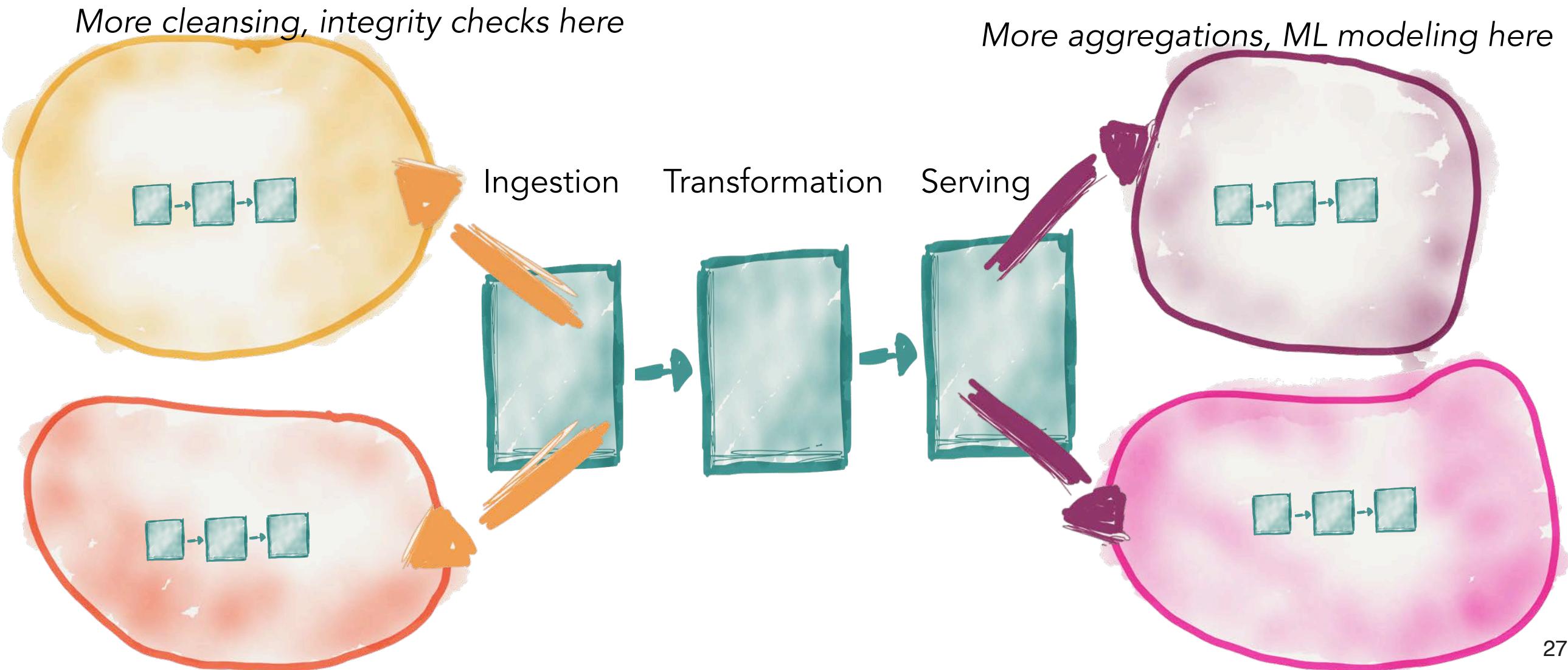
Change more often

Can be recreated



DISTRIBUTED PIPELINES IN DOMAINS

“Unification of Computation and Data”



Decomposition Heuristics

Aligned with source domains to retain business facts

Aligned with common consumptions to share modeled data

Differing protection requirements

Domain data complexity

Unique access patterns e.g. Graph data



Domains are the first class concern

Top Level partitions

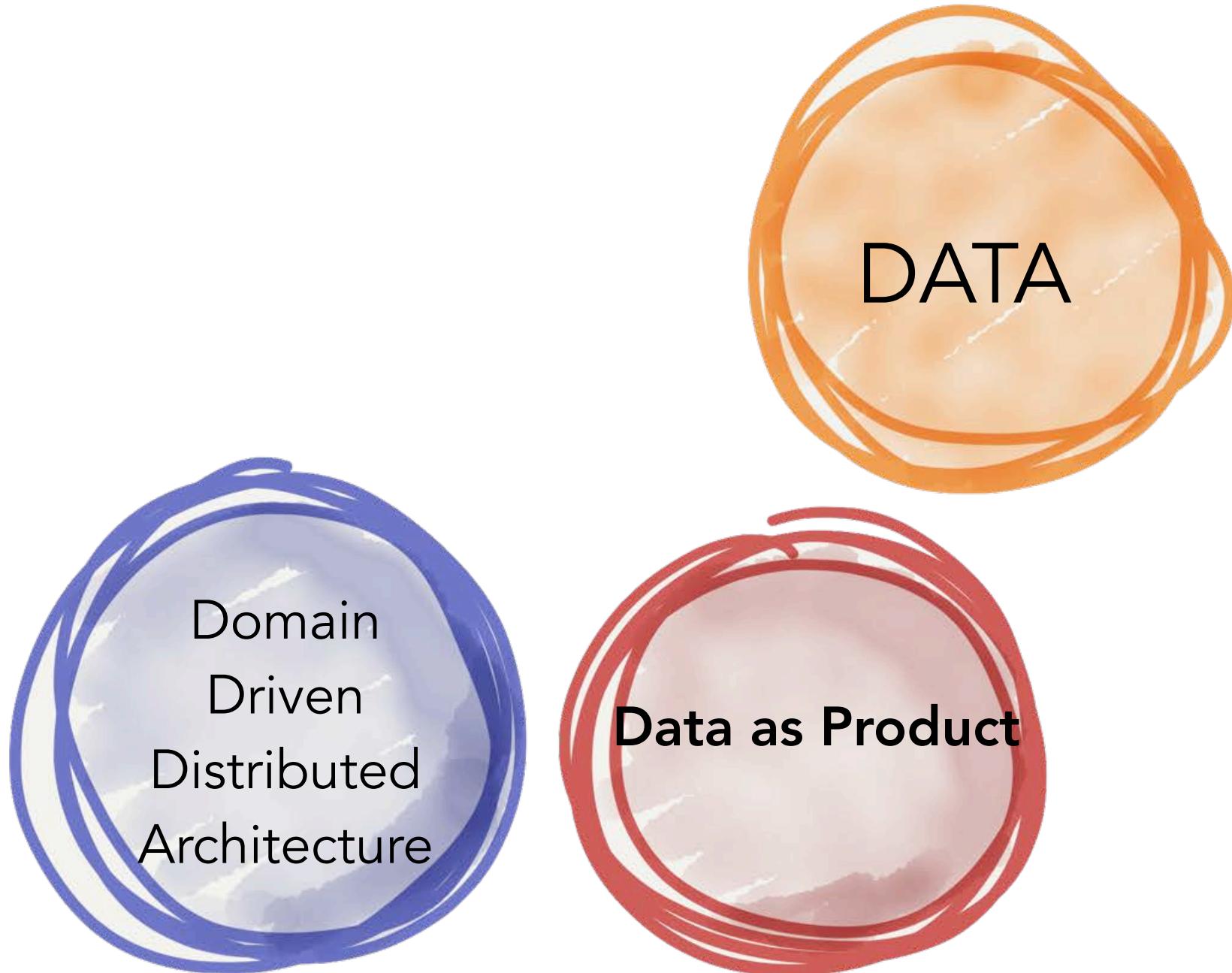
Data pipelines are second class concern

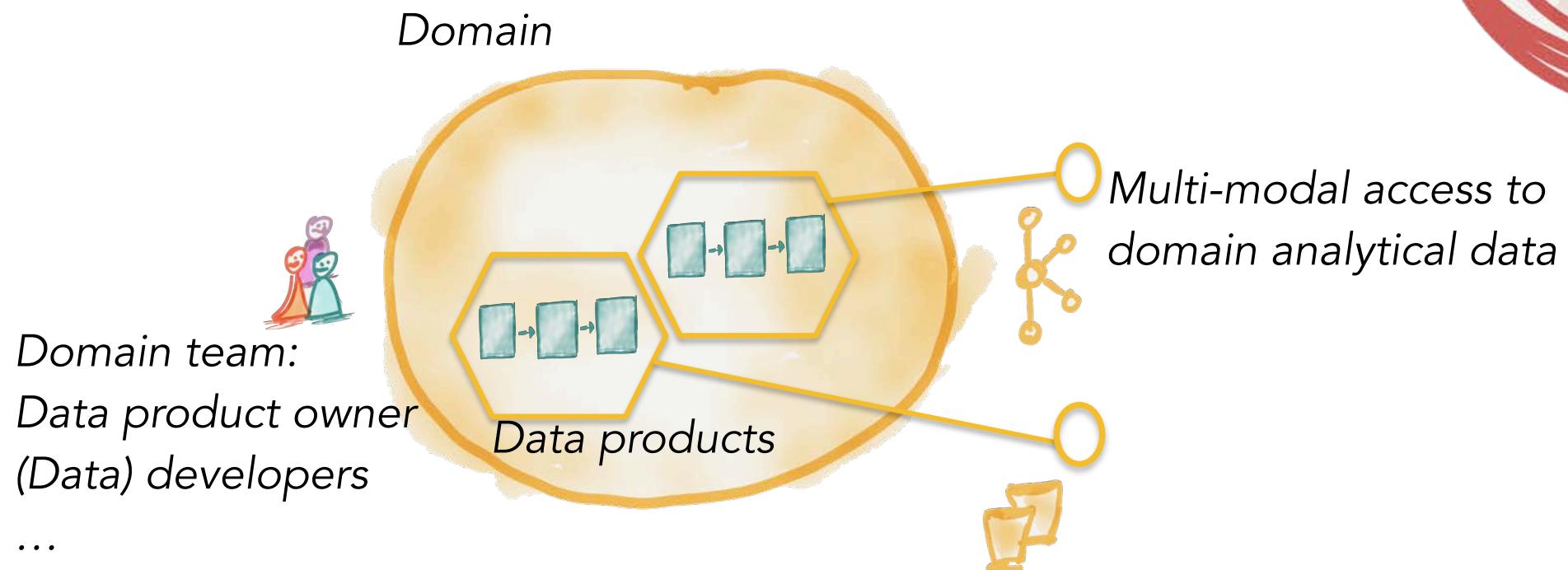
Implementation details

Architectural Quantum shifts from a
a pipeline to a domain (datasets+computation)

Domain datasets are immutable (time series)

DATA MESH PRINCIPLES





Data as Product



SHARED | DISCOVERABLE



SELF-DESCRIBING



ADDRESSABLE



INTER OPERABLE
(GOVERNED
BY GLOBAL STANDARDS)



TRUSTWORTHY
(DEFINED & MONITORED SLOs)



SECURE
(GOVERNED
BY GLOBAL ACCESS CO
NTROL)



Domain Datasets as a product

Discoverable
Inter-operable
Explicit Quality Objectives
Secure
Shared

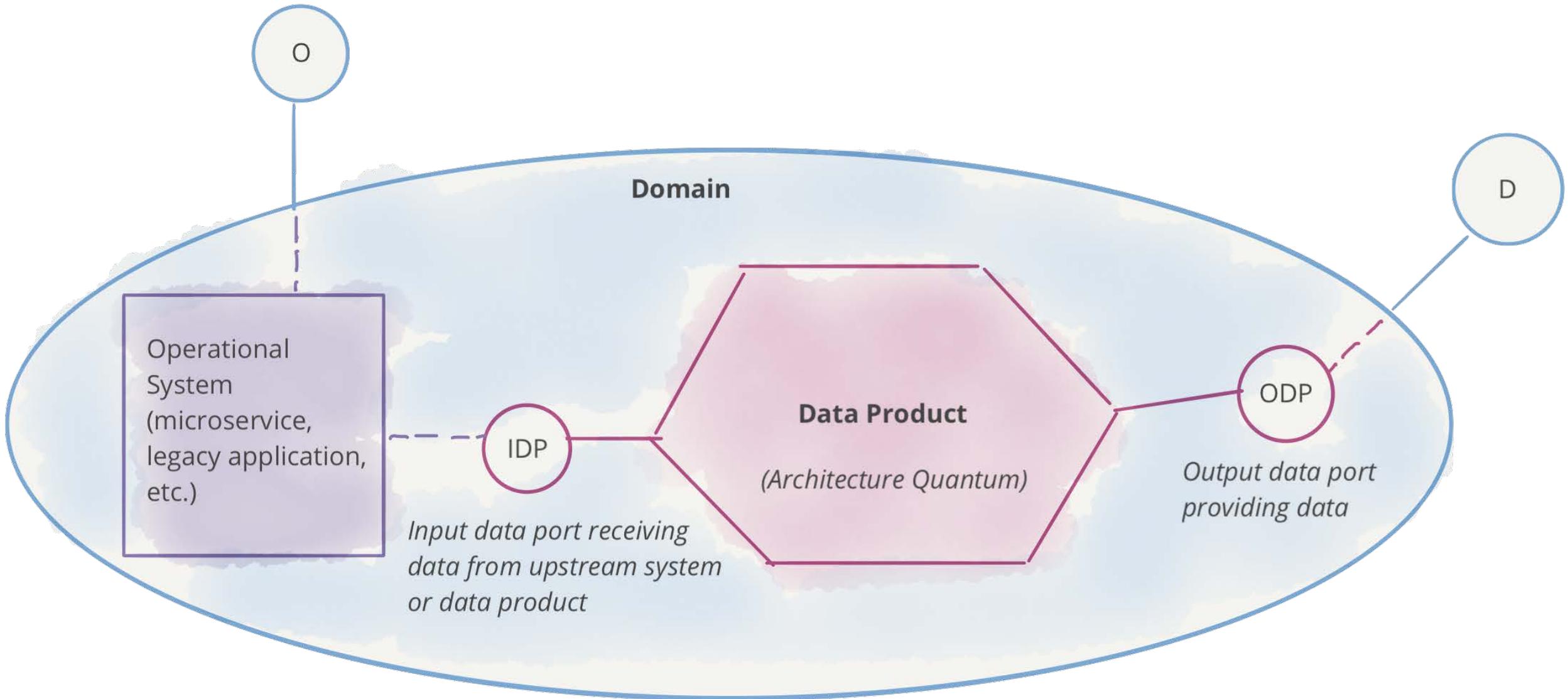
Data consumers as customers

Data Product Owner role

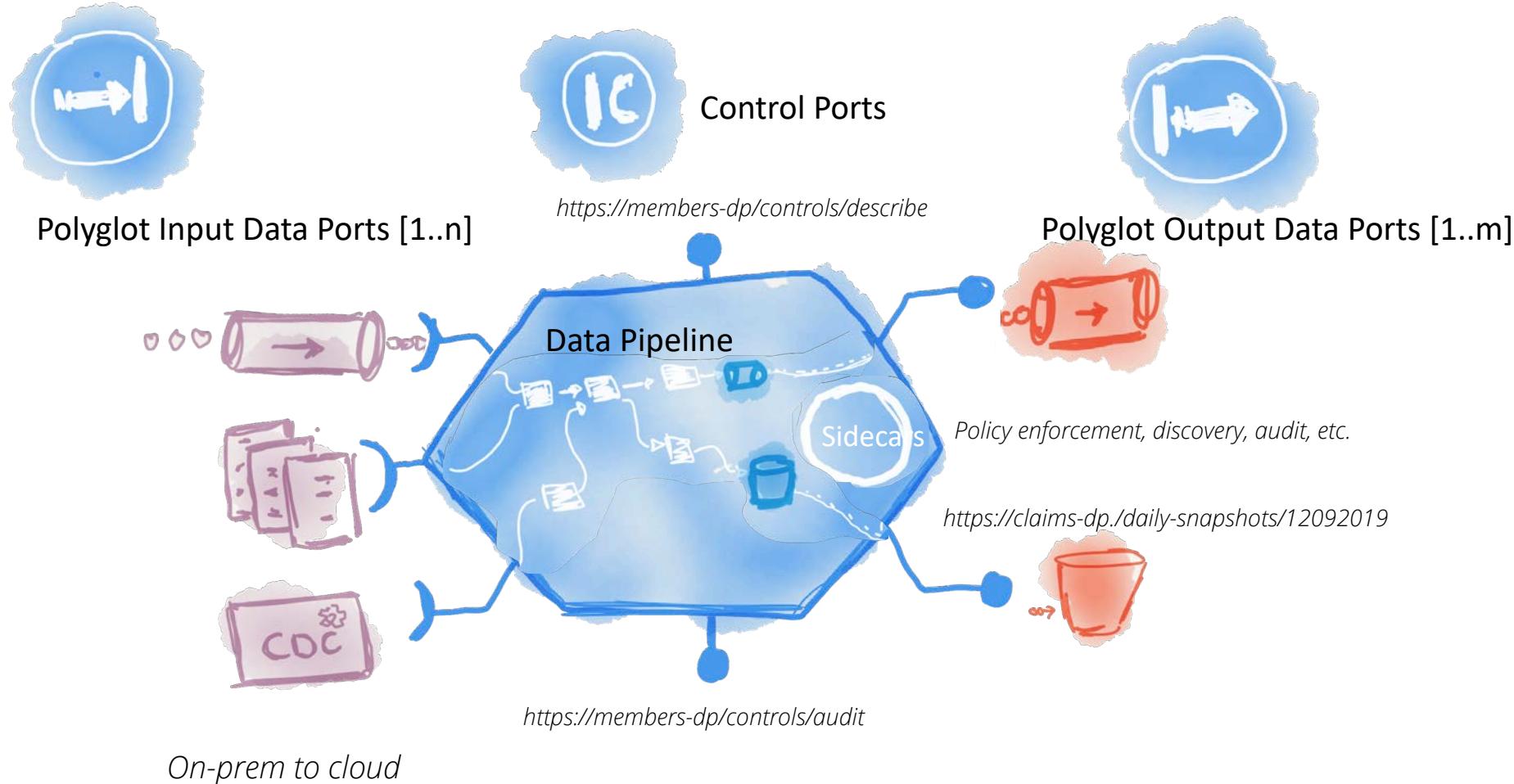
Cross-functional team ownership

Success criteria: Decreased lead time to discover and consume a data product

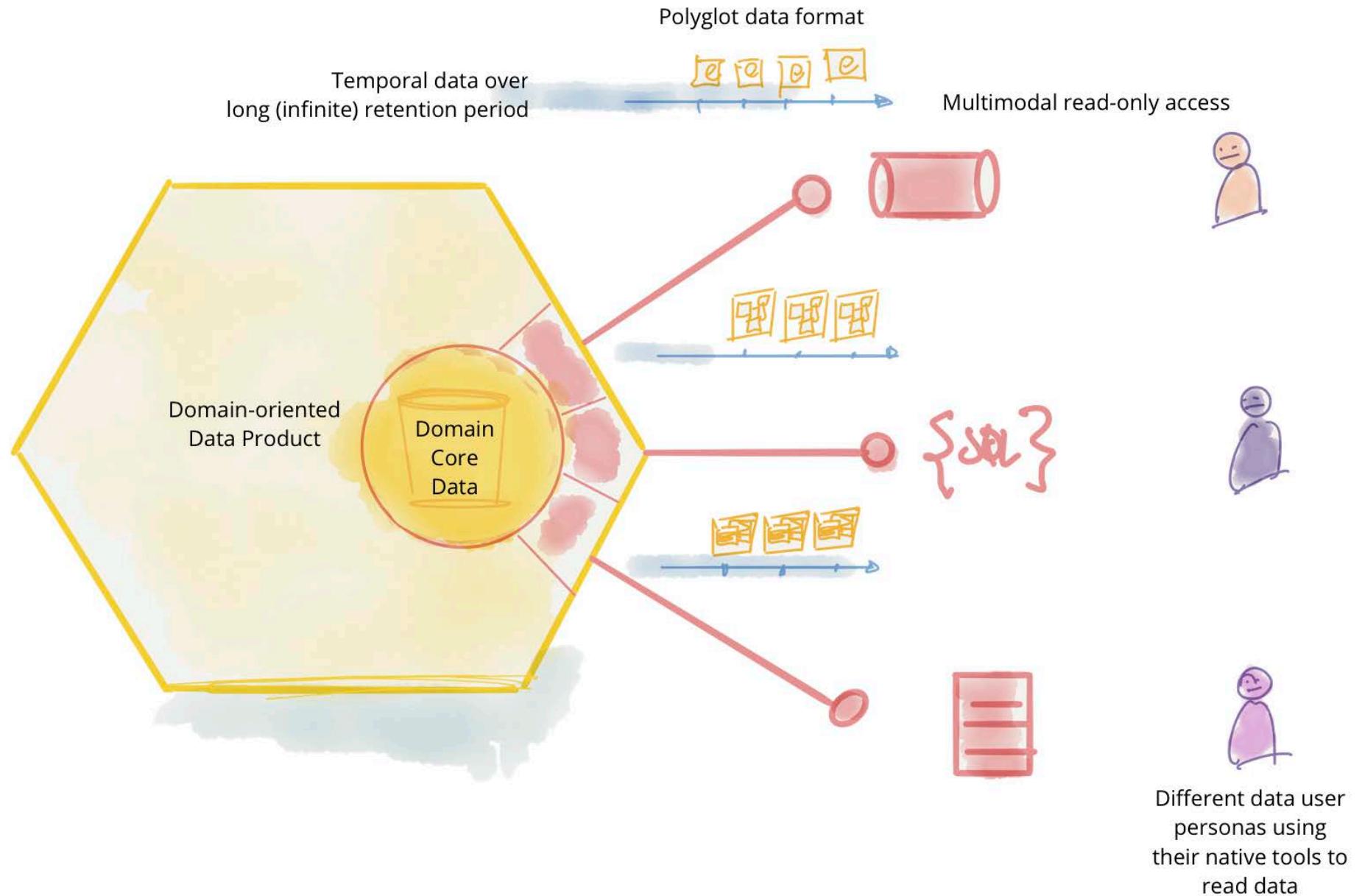
DATA PRODUCT ARCHITECTURE



DATA PRODUCT ARCHITECTURE



DATA PRODUCT DATA ACCESS AFFORDANCE



Sysop Squad

The electrics giant is continuously looking to optimize its business. They are intending to use the data that have available to them about their experts, incoming calls, and quality of services over time to make improvements. One of the early improvements they are looking to make is to have an optimized supply of experts at hand to respond to their customers needs. They are looking to create reports on the skillsets in high demand, tailor and run experts trainings for areas of improvements, and hire unique specializations that are needed.

Another area of dynamic optimization is based on location and routing, looking to dynamically allocate experts to locations that see a sudden rise in calls due to power outages, or concentration of businesses, etc.

Requirements:

- The enhancements to collect data, generate reports, as well as train and use machine learning models are built as extensions / modifications to the existing systems and architecture in place.
- While the electric giants is hypothesizing on two initial use cases - optimized resource management and routing - they are hoping to evolve and improve their business extending the use of their data to optimize other aspects of their business.
- Privacy and protection of their customers personal information is a must.

Kata Exercise - Analytical Data Architecture

Design an analytical data architecture that can meet the requirements and data aspirations that *Penultimate Electronics* has.

In your architecture:

- Demonstrate how you decompose the components of your architecture; How you decompose analytical data.
- Demonstrate how you integrate the analytical plane with data plane
- Demonstrate the end to end integration of components, in order to address the use cases of (a) experts skillsets demand trends monthly reports, and (b) daily experts supply planning datasets.

Kata Exercise - Analytical Data Architecture

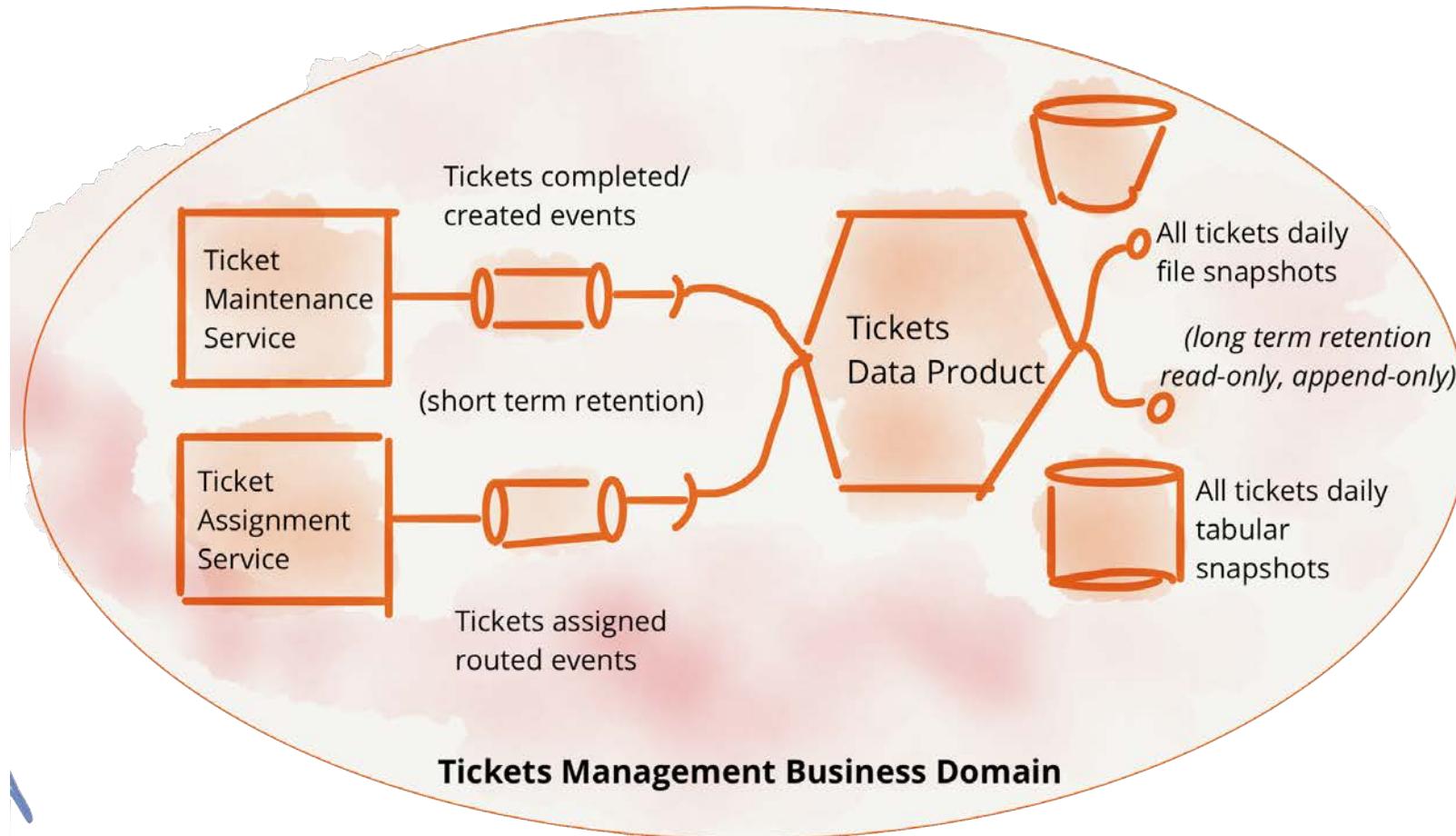
Given the aspirations of the company, improving their business and customer experience with ML and analytics, they need to have **insights** into:

- what their customers need and how that changes over time
- how the customers feel about their services over time,
- Understand their experts population is changing over time

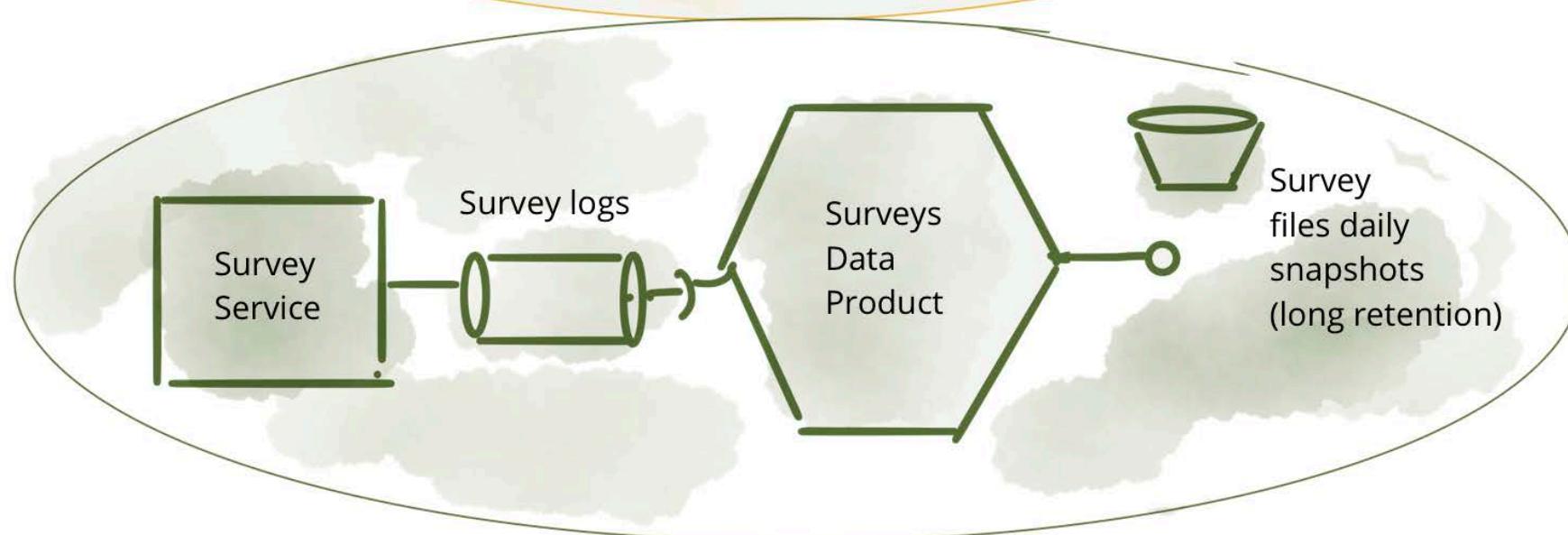
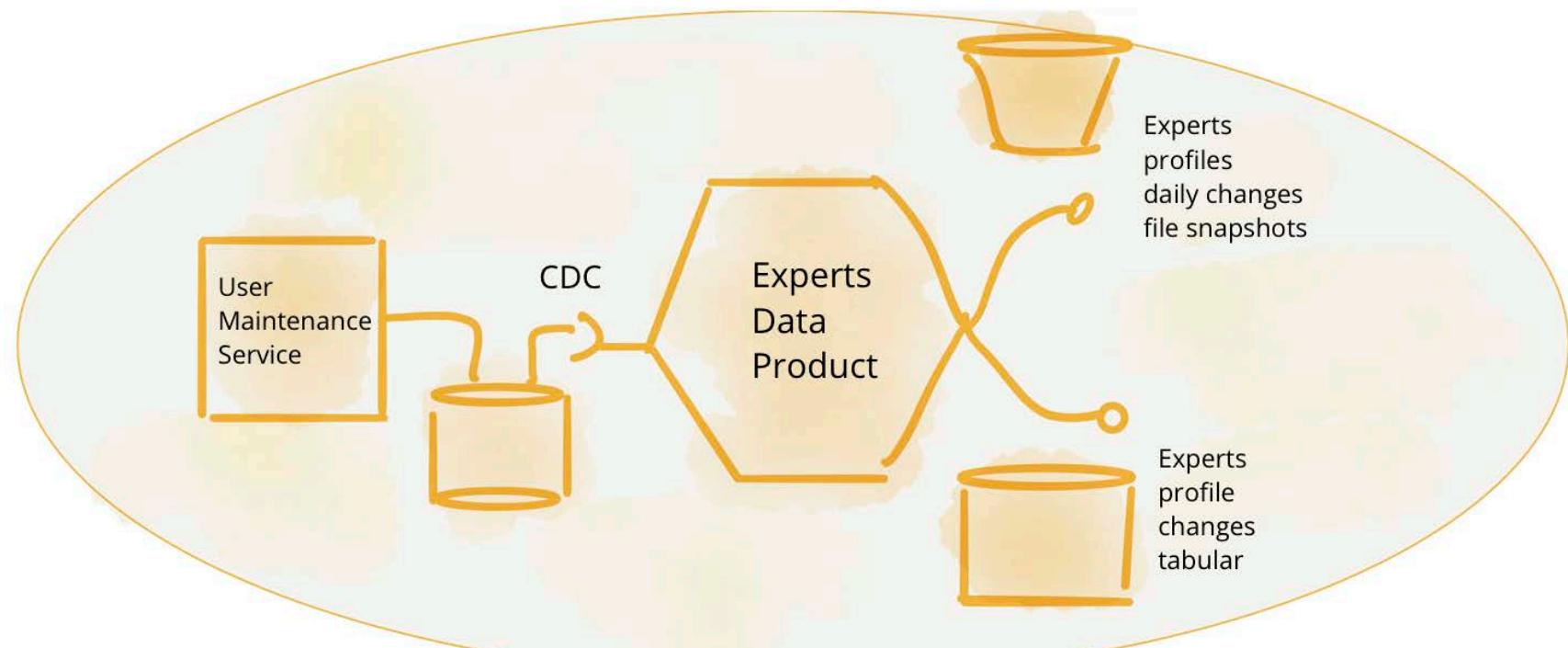
For that, they need to do a few things:

- capture domains analytical data (over time) that contribute to forming the insights mentioned above.
- form new domain's data that fuels their analytical capabilities - ML models, or reports.
- build new business domain to operate a “smart experts supply management” including just-in-time training, smart routing, smart experts supply planning, experts population adjustment, etc.
- Hire data analysts and data scientists and provide them data in the format that is native to them and toolsets they use.

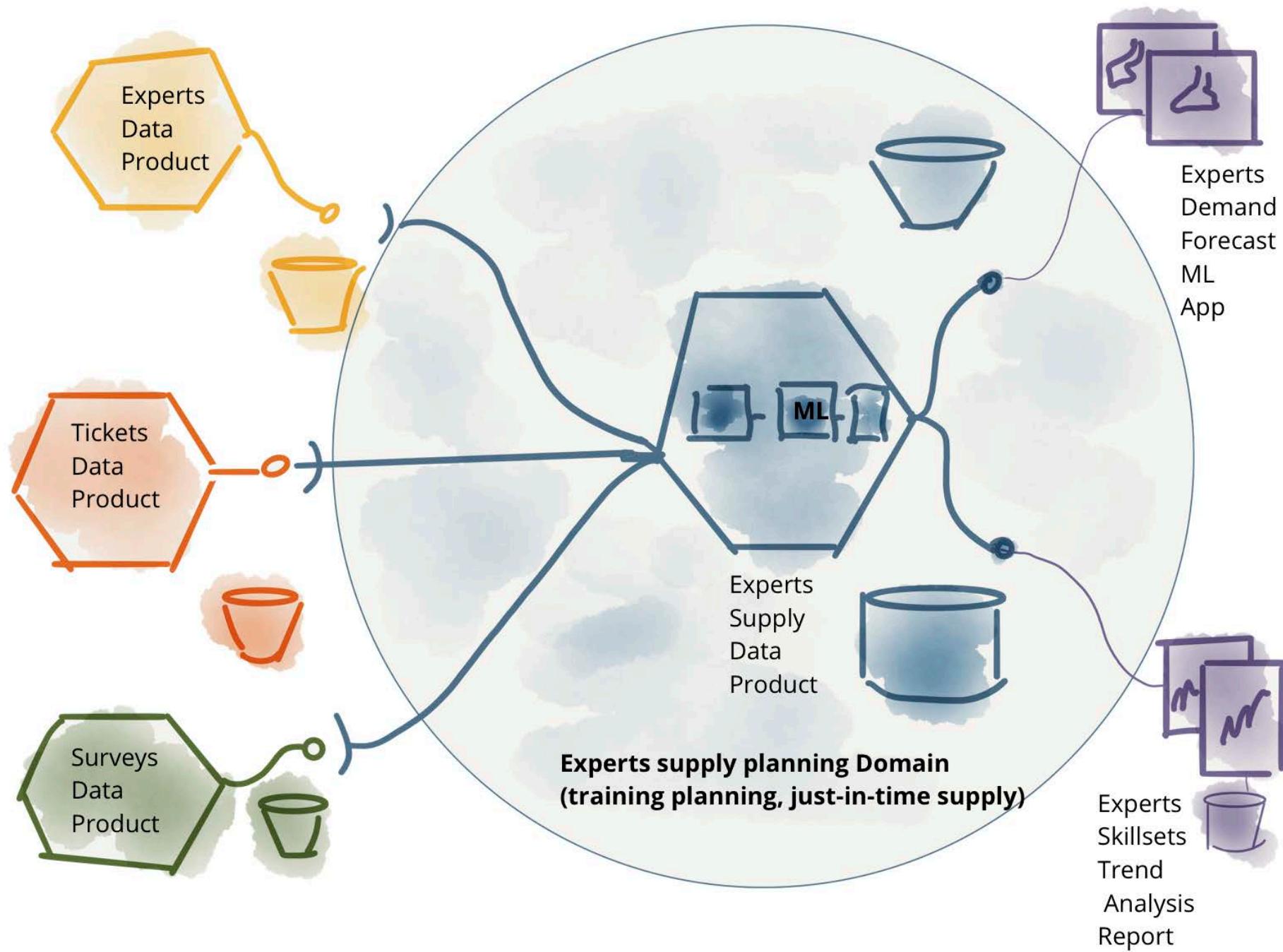
Kata Exercise - Analytical Data Architecture



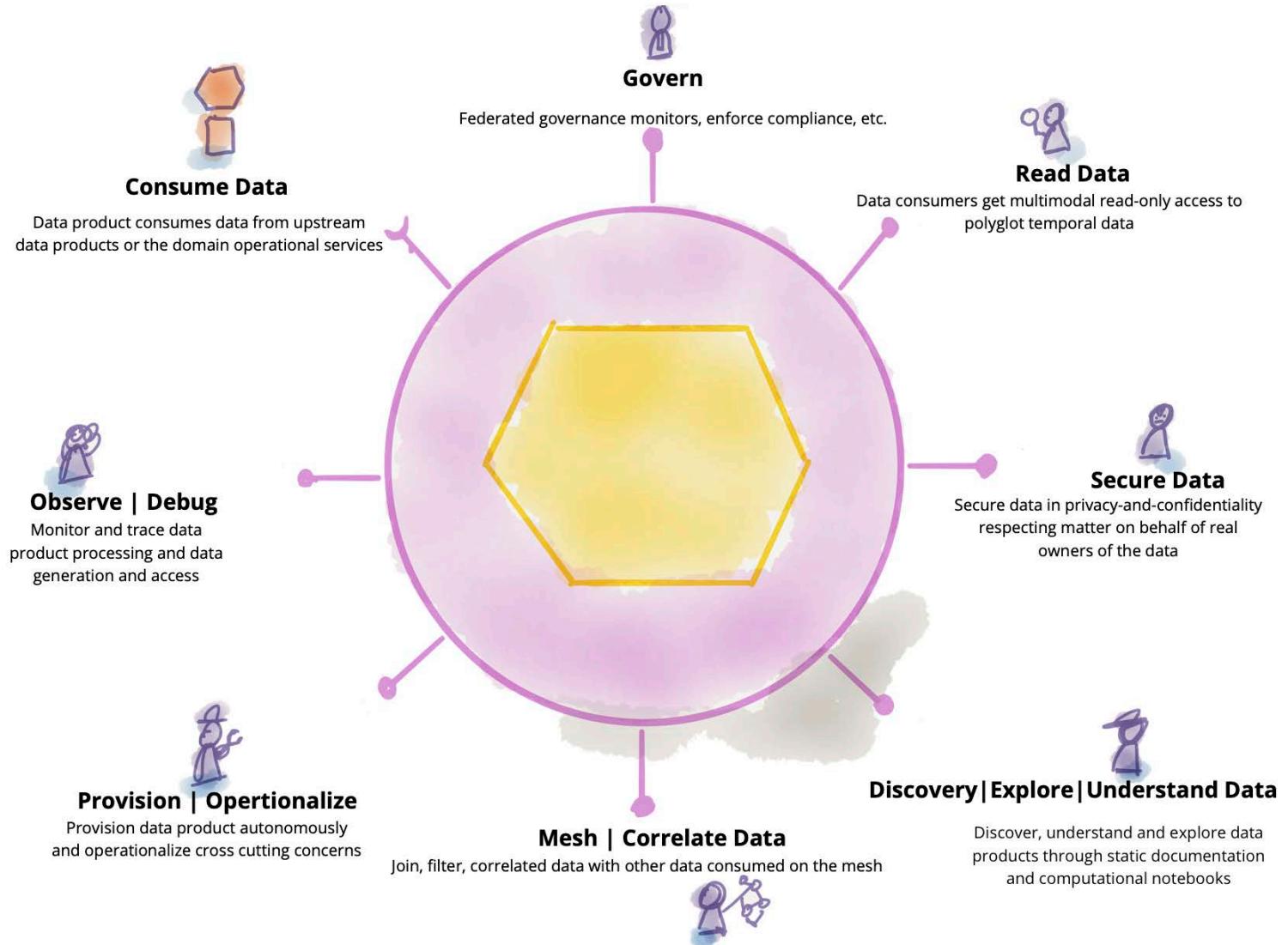
Kata Exercise - Analytical Data Architecture



Kata Exercise - Analytical Data Architecture



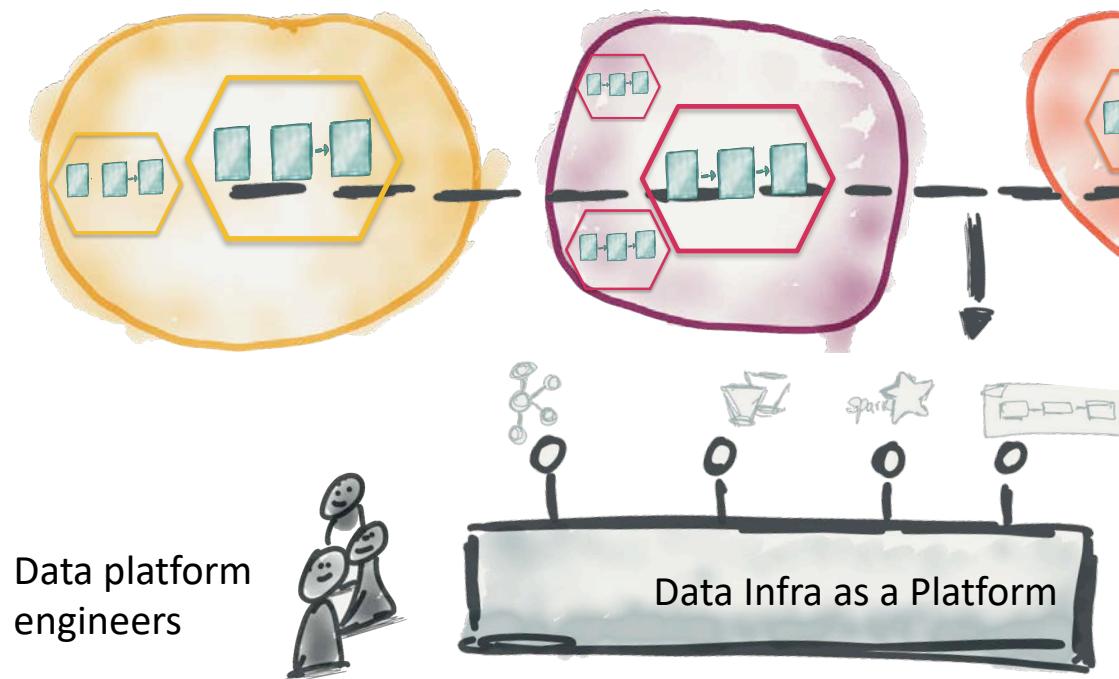
DATA PRODUCT AFFORDANCES



DATA MESH PRINCIPLES

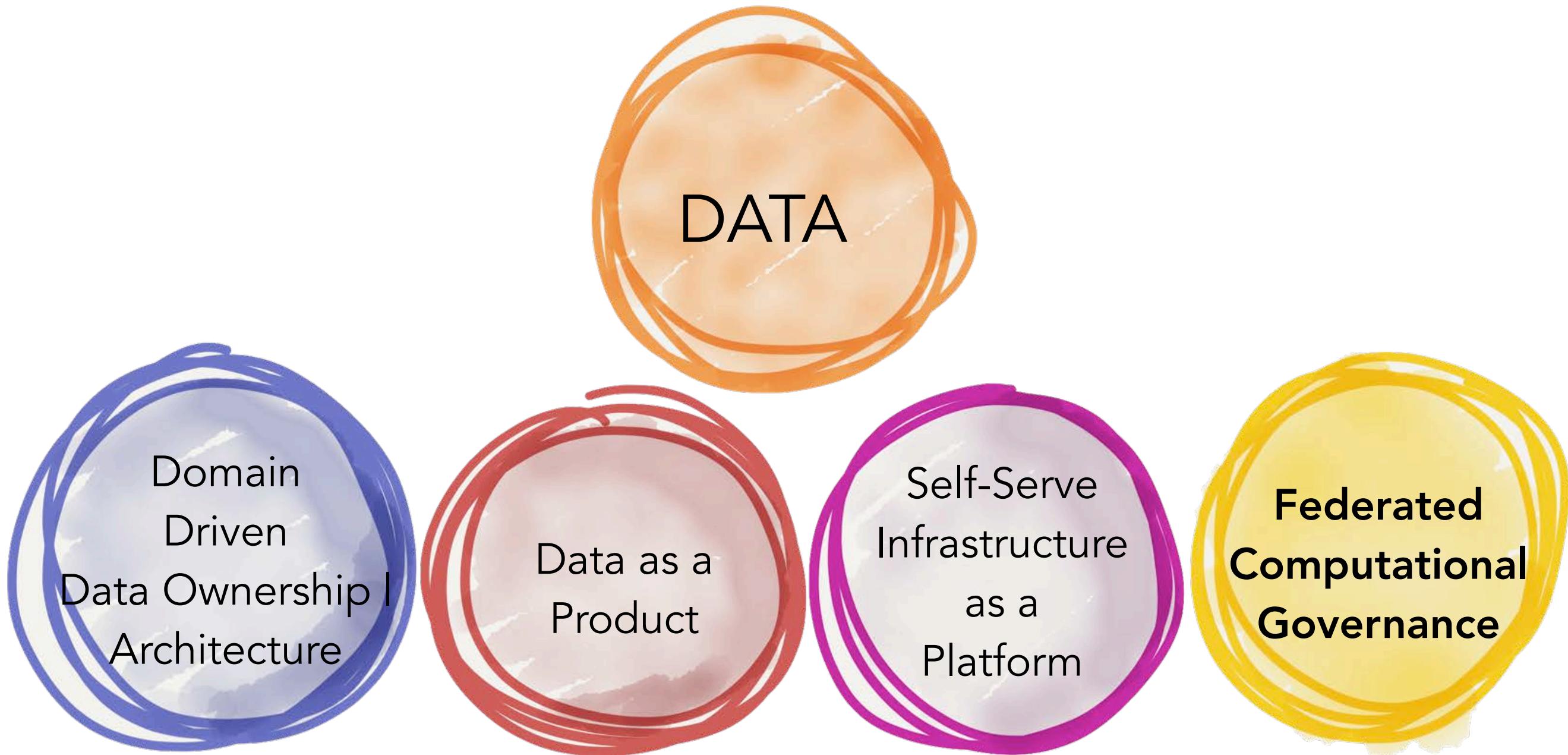


Self-Serve Infrastructure as a Platform

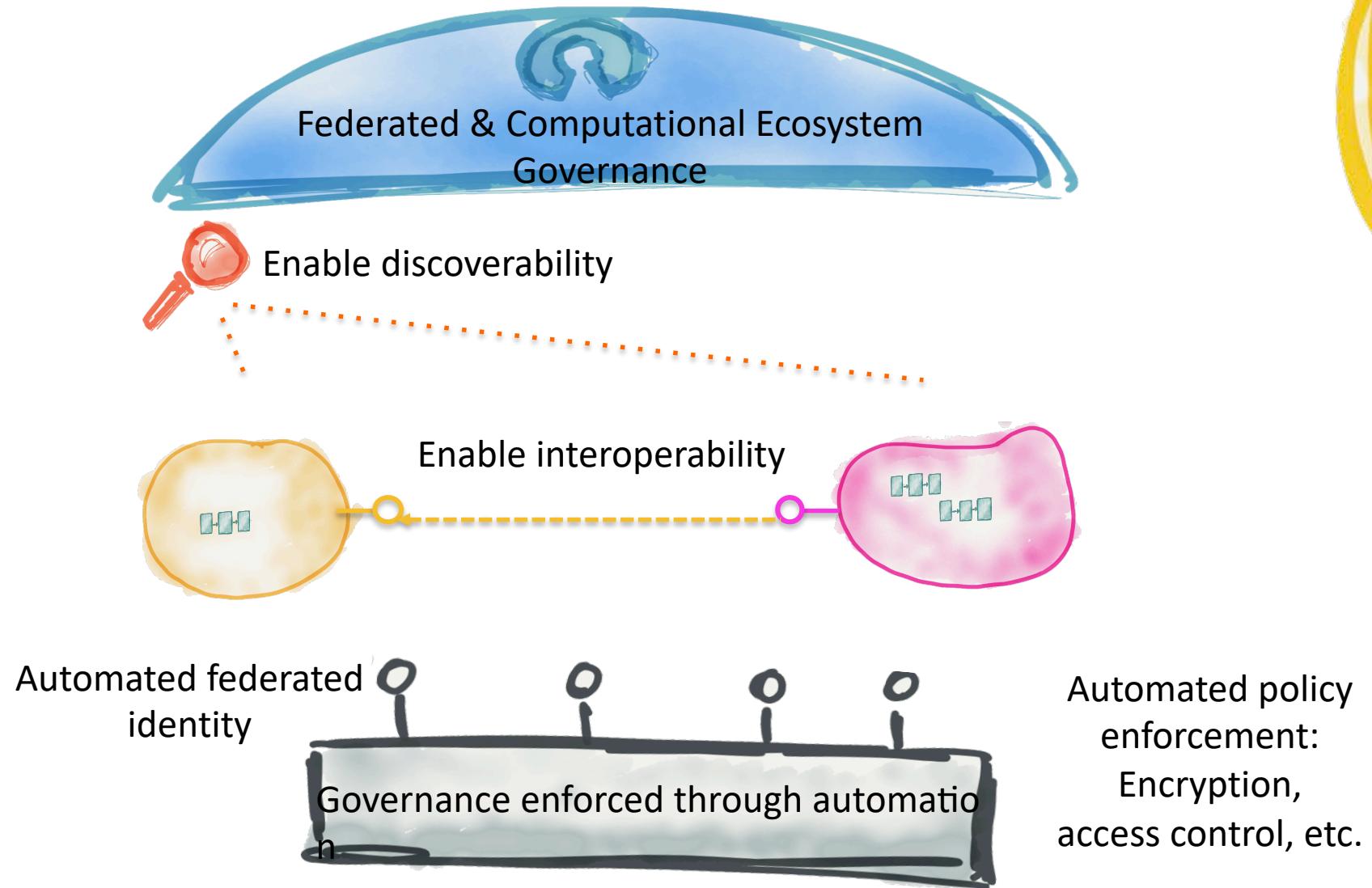


Domain agnostic shared/centralized self-serve tooling / infrastructure to create secure data products quickly, discover them, execute their pipelines, etc.

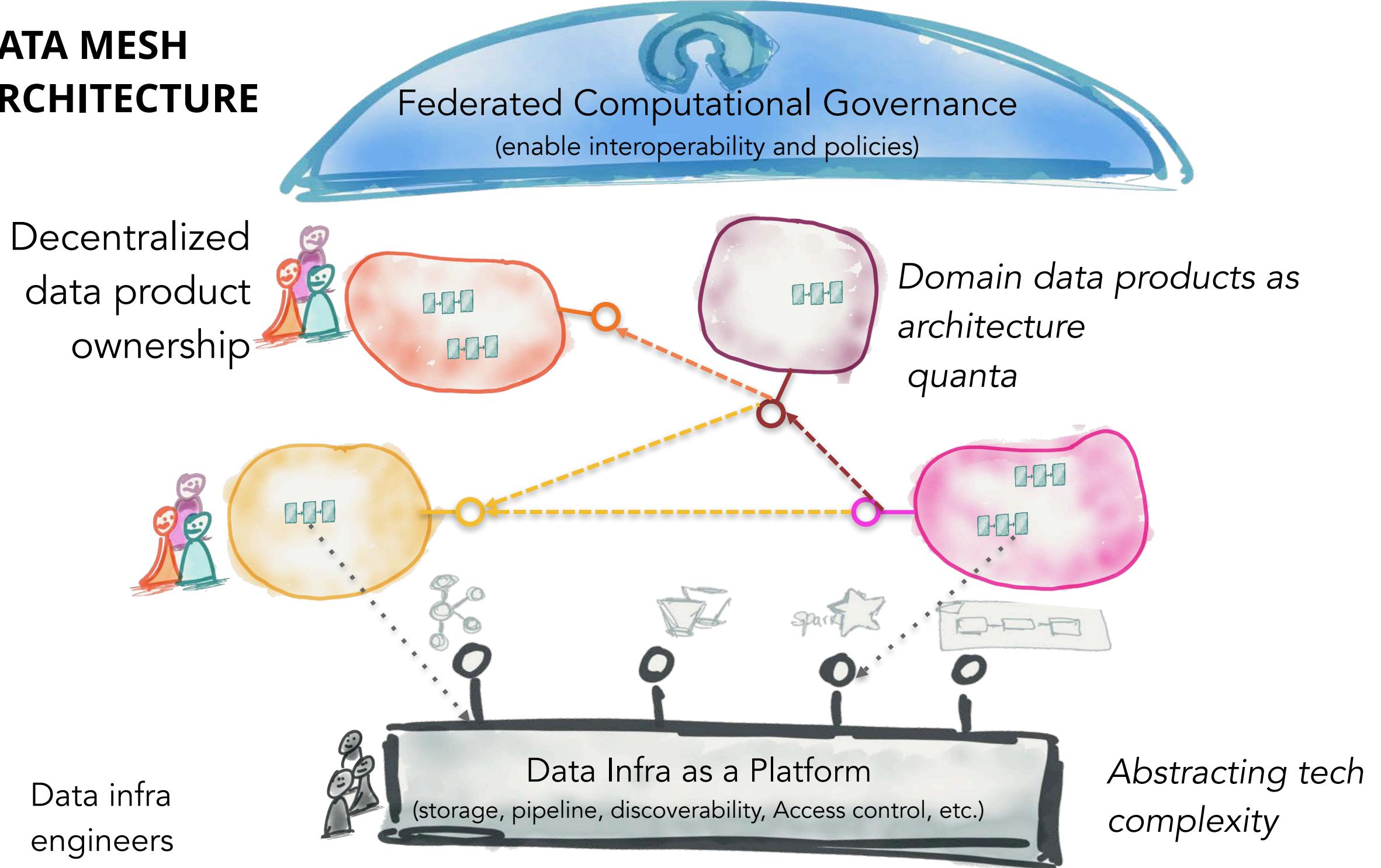
DATA MESH PRINCIPLES



Federated Computational Governance



DATA MESH ARCHITECTURE



How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh

Many enterprises are investing in their next generation data lake, with the hope of democratizing data at scale to provide business insights and ultimately make automated intelligent decisions. Data platforms based on the data lake architecture have common failure modes that lead to unfulfilled promises at scale. To address these failure modes we need to shift from the centralized paradigm of a lake, or its predecessor data warehouse. We need to shift to a paradigm that draws from modern distributed architecture: considering domains as the first class concern, applying platform thinking to create self-serve data infrastructure, and treating data as a product.

20 May 2019



CONTENTS

- [The current enterprise data platform architecture](#)
 - [Architectural failure modes](#)
 - [Centralized and monolithic](#)
 - [Coupled pipeline decomposition](#)
 - [Siloed and hyper-specialized ownership](#)
- [The next enterprise data platform architecture](#)
 - [Data and distributed domain driven architecture convergence](#)
 - [Domain oriented data decomposition and ownership](#)
 - [Source oriented domain data](#)
 - [Consumer oriented and shared domain data](#)
 - [Distributed pipelines as domain internal implementation](#)
 - [Data and product thinking convergence](#)
 - [Domain data as a product](#)
 - [Discoverable](#)
 - [Addressable](#)
 - [Trustworthy and truthful](#)
 - [Self-describing semantics and syntax](#)
 - [Inter-operable and governed by global standards](#)
 - [Secure and governed by a global access control](#)
 - [Domain data cross-functional teams](#)

 [ENTERPRISE ARCHITECTURE](#)

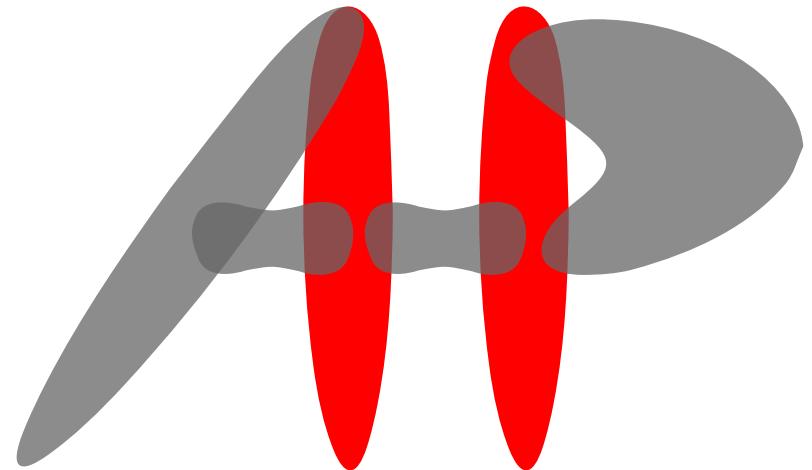
 [BIG DATA](#)

<https://martinfowler.com/articles/data-monolith-to-mesh.html>



Synchronous vs Async Communication

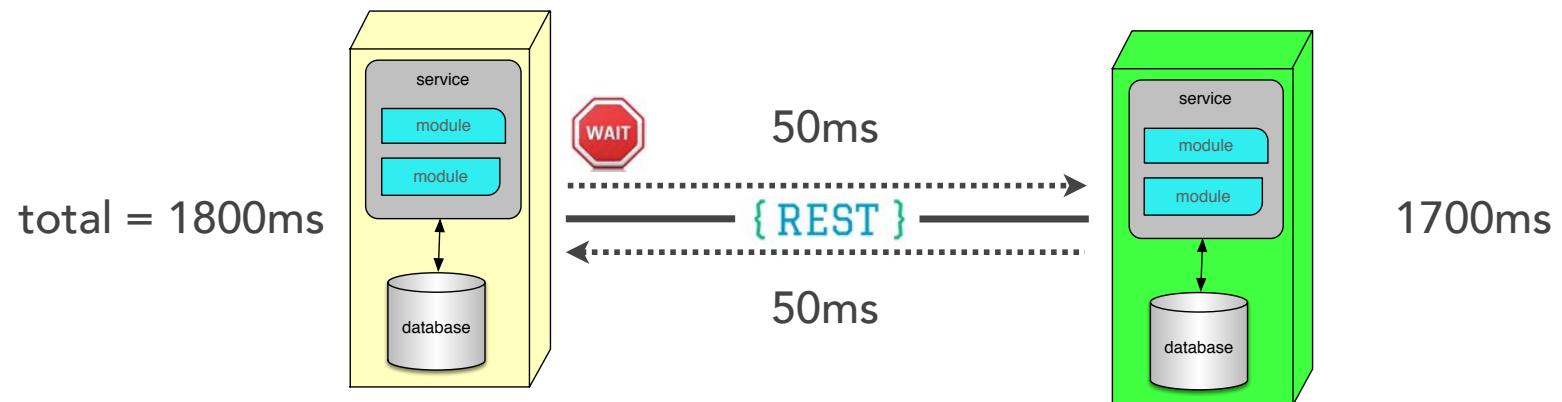
architectural quantum 2021



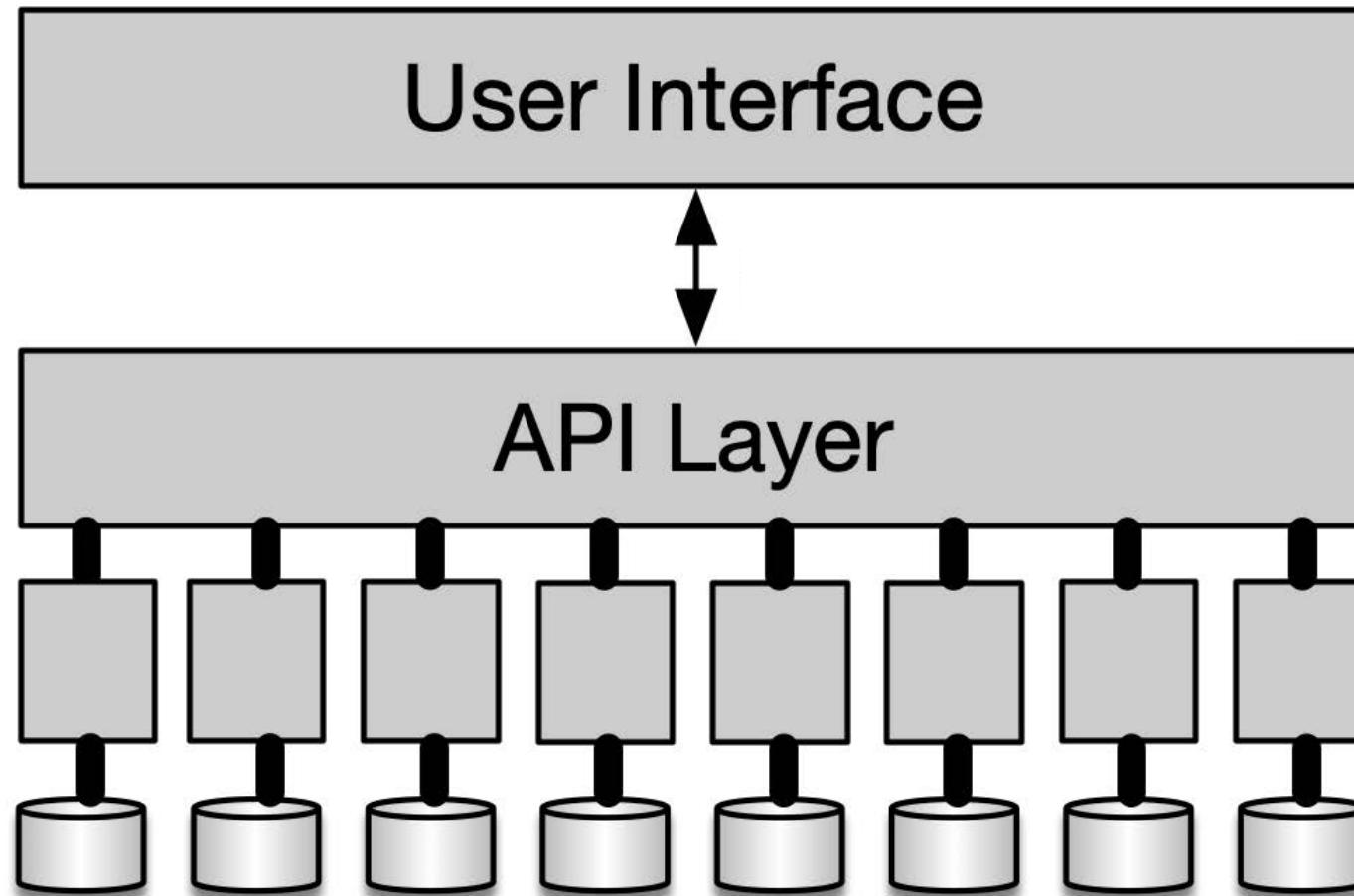
an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling

Dynamic Coupling

synchronous

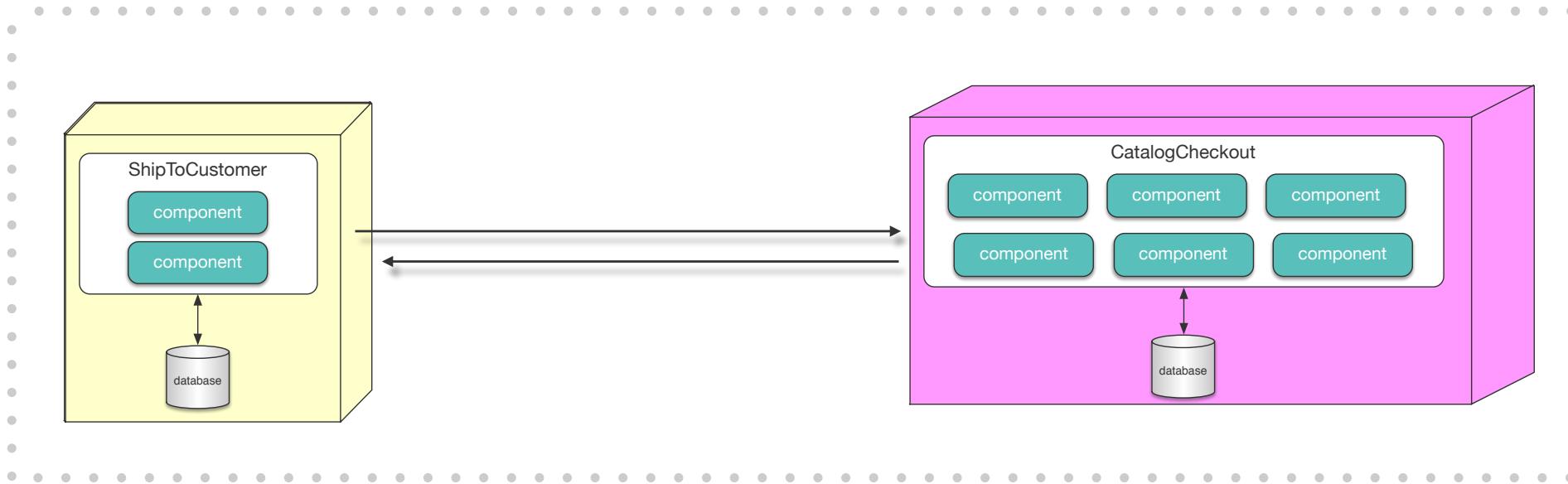


Single Dynamic Quantum



Dynamic Coupling

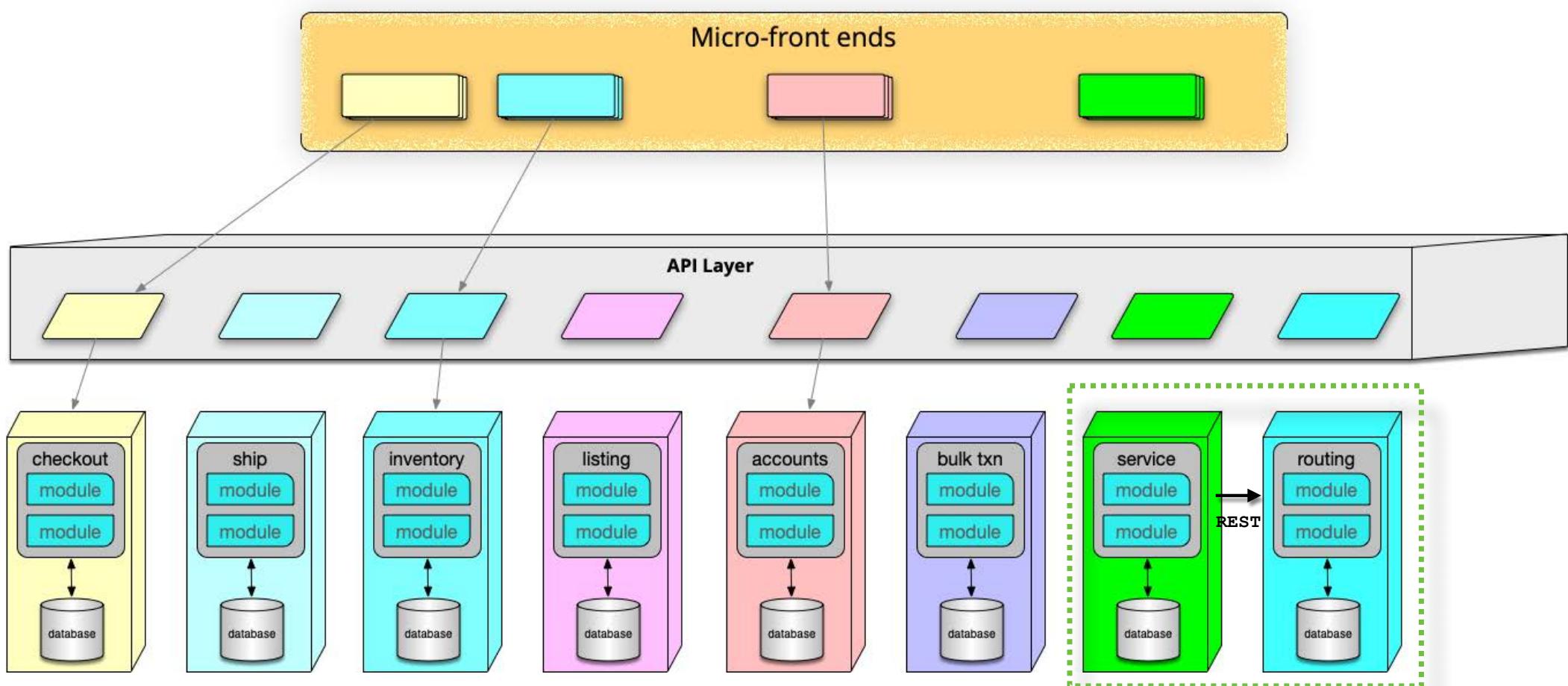
synchronous



synchronous calls create a
dynamic quantum entanglement
around operational architecture characteristics

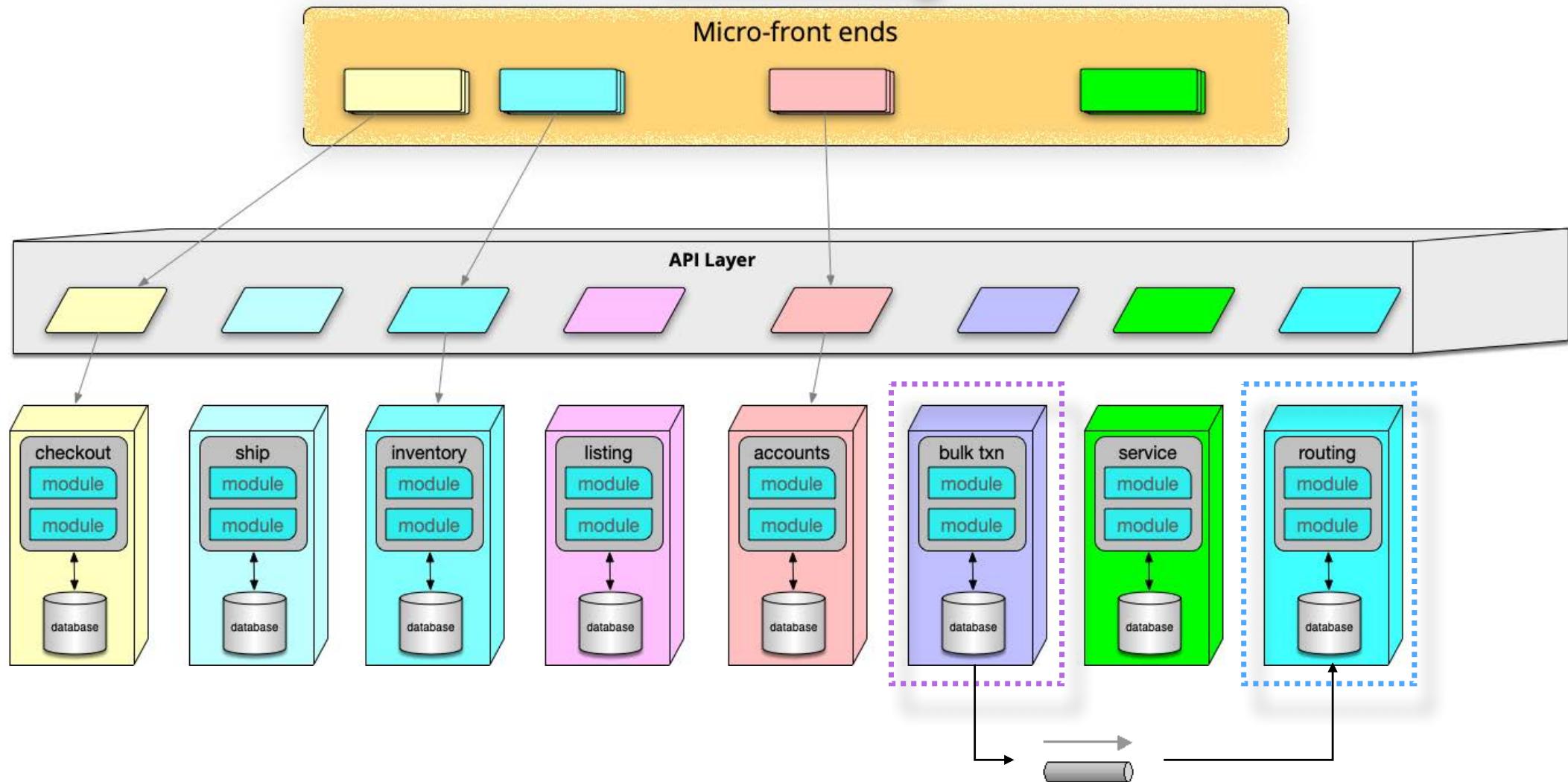
Dynamic Coupling

synchronous



Dynamic Coupling

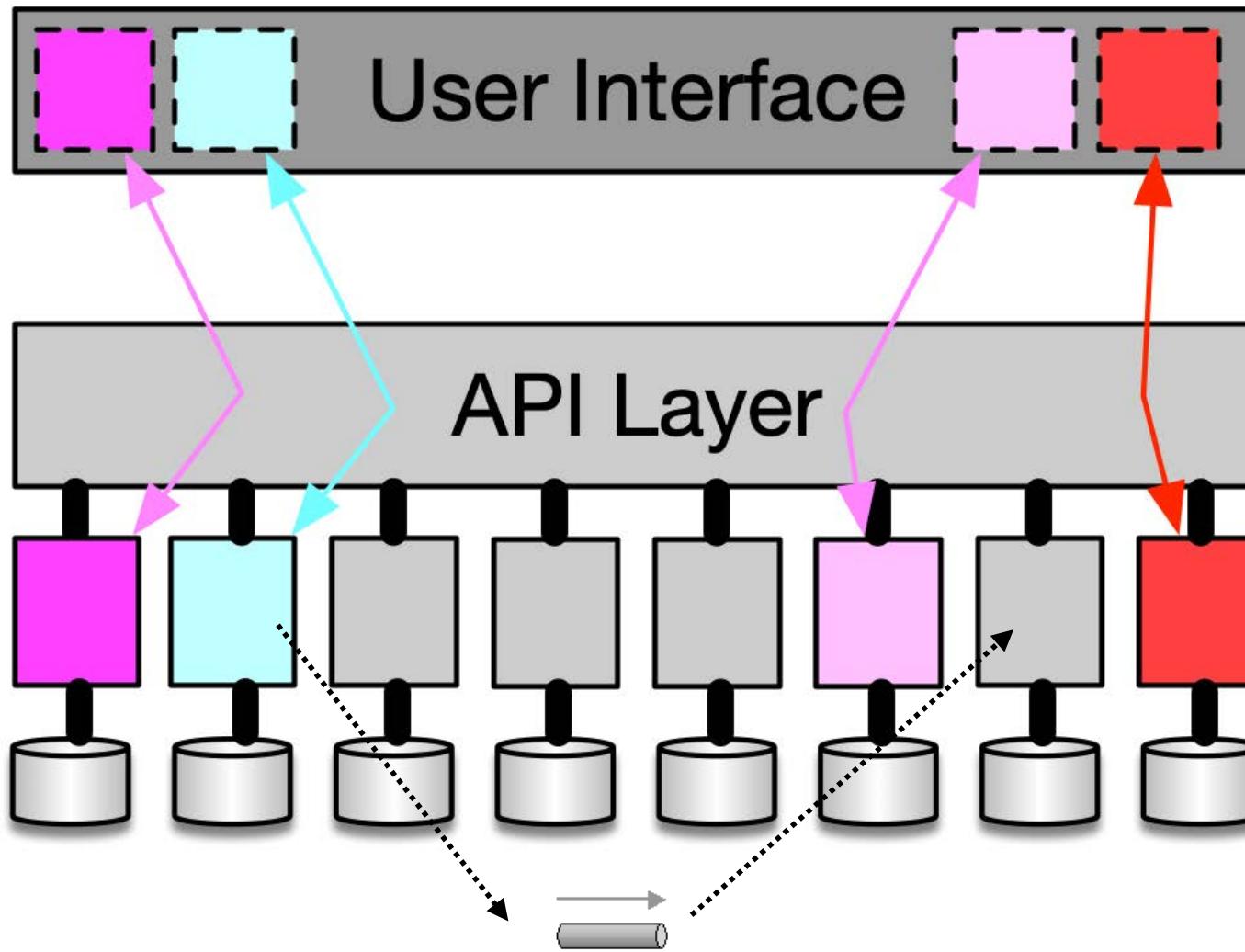
asynchronous



asynchronous

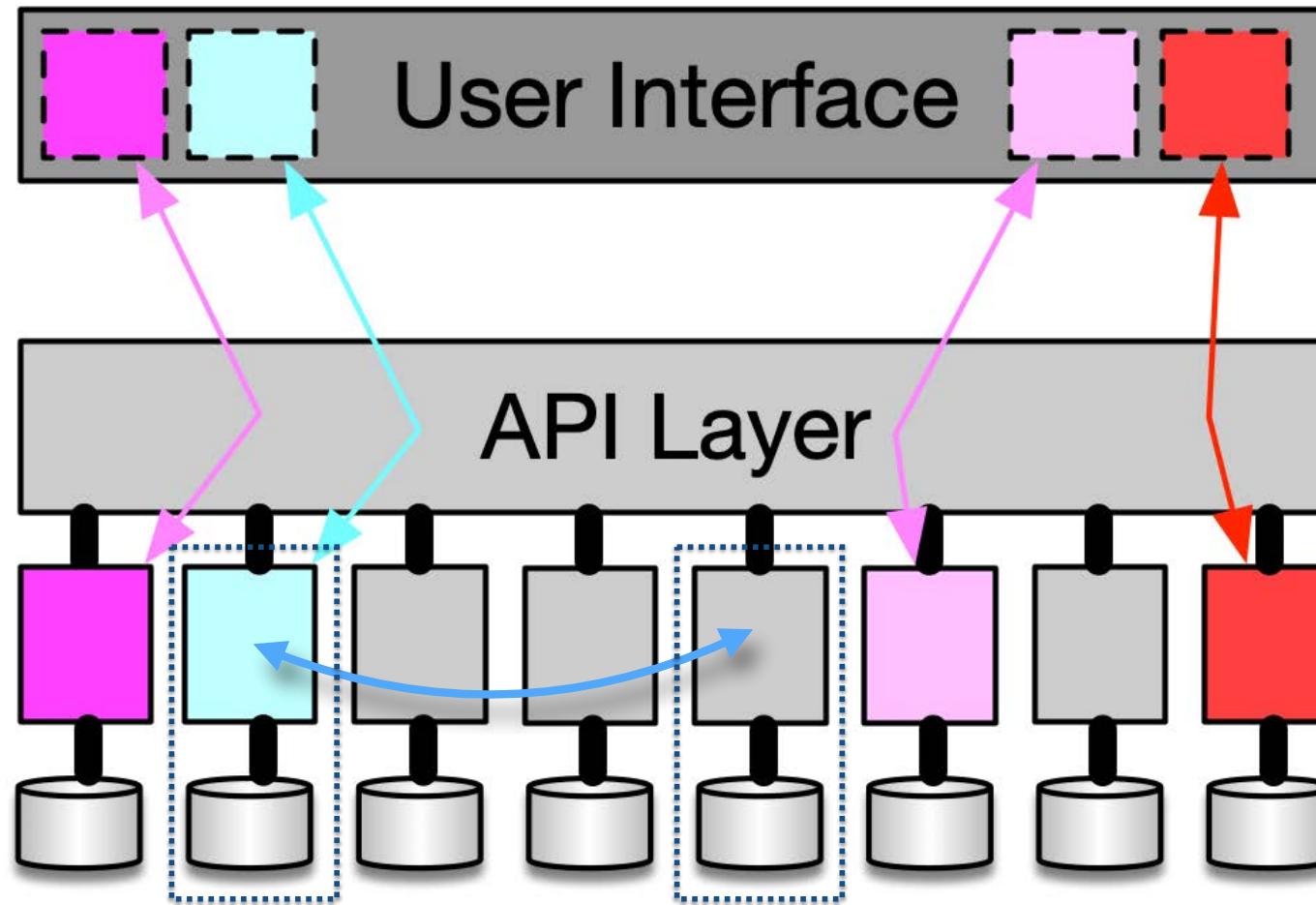
Async == independence

Micro-front Ends



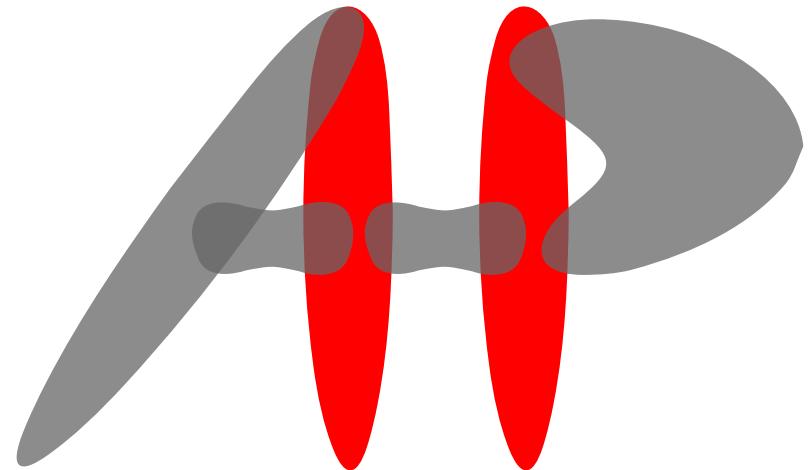
<https://martinfowler.com/articles/micro-frontends.html>

Micro-front Ends



One quantum for the invocation duration

architectural quantum 2021



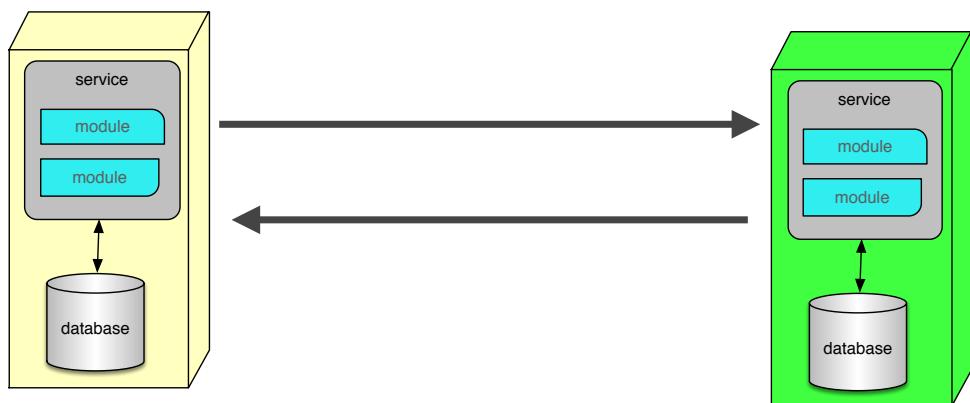
Represents how quanta communicate at runtime, either synchronously or asynchronously.

an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling

sync | async ?

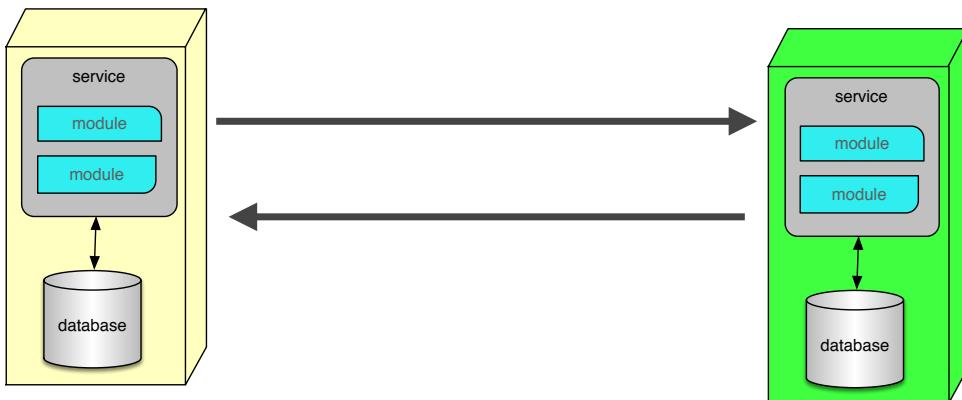
sync | async ?

Caller waits (blocks) for response

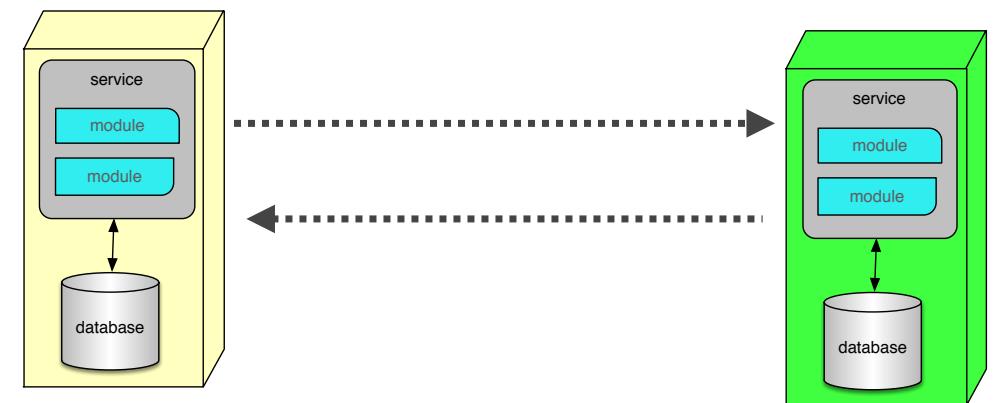


sync | async ?

Caller waits (blocks) for response

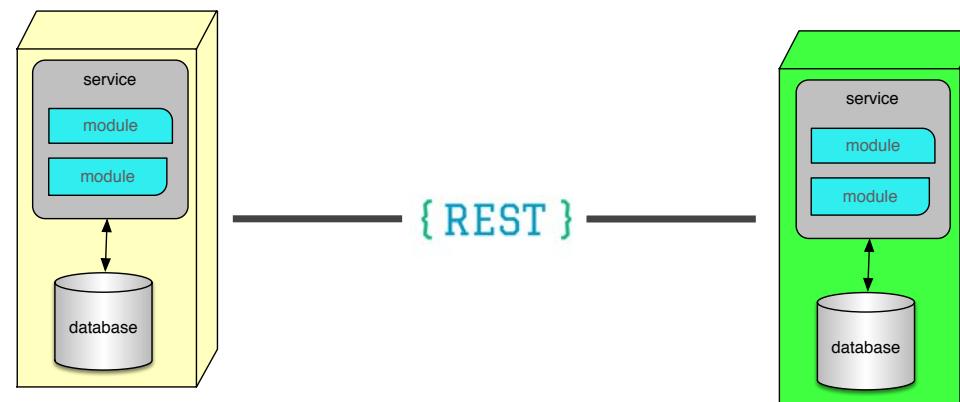


Caller receives response asynchronously



sync | async ?

synchronous



sync | async ?

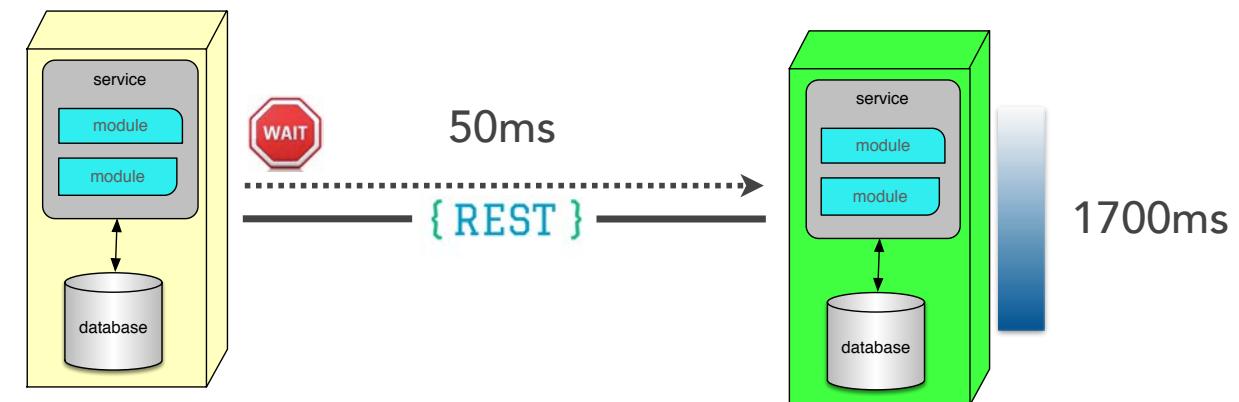
communication

synchronous



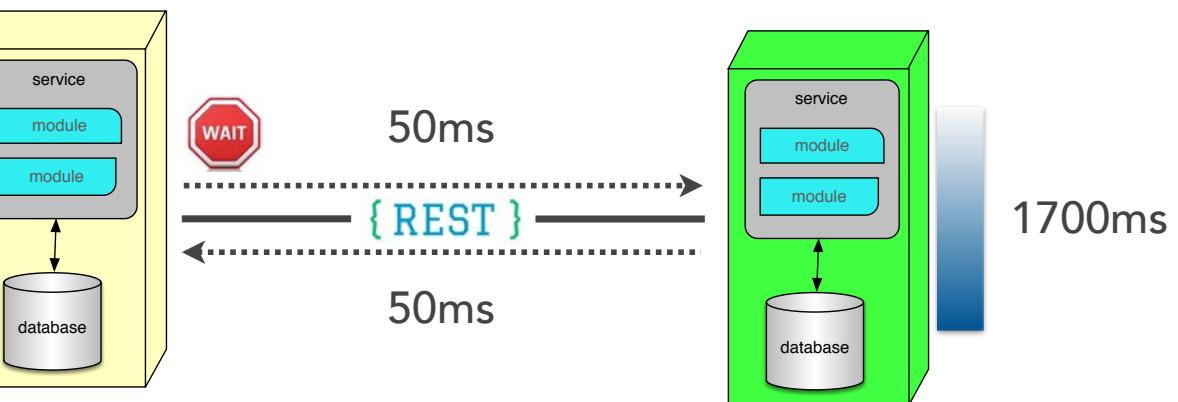
sync | async ?

synchronous



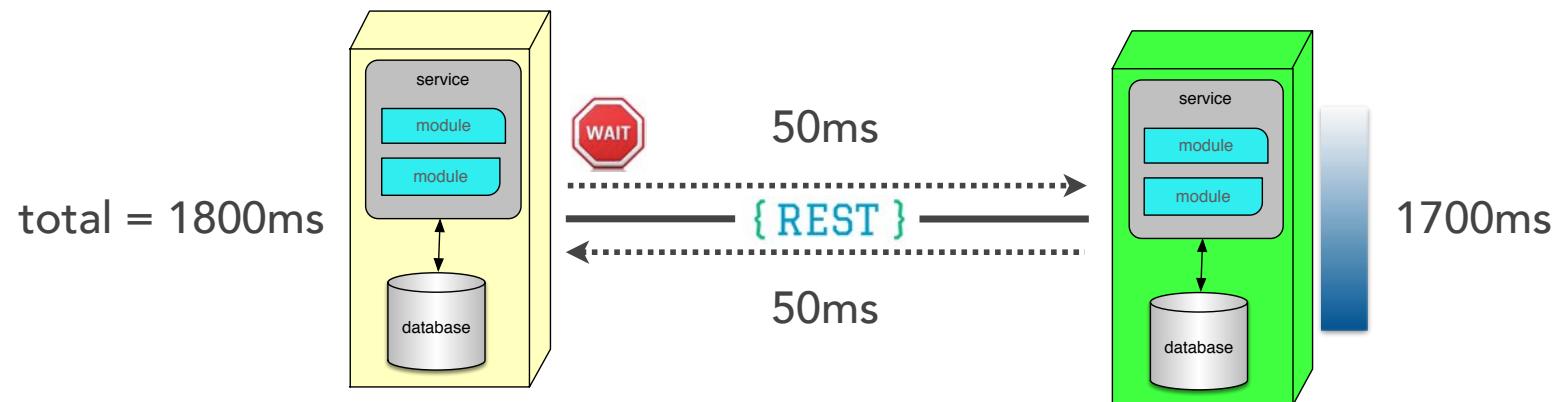
sync | async ?

synchronous

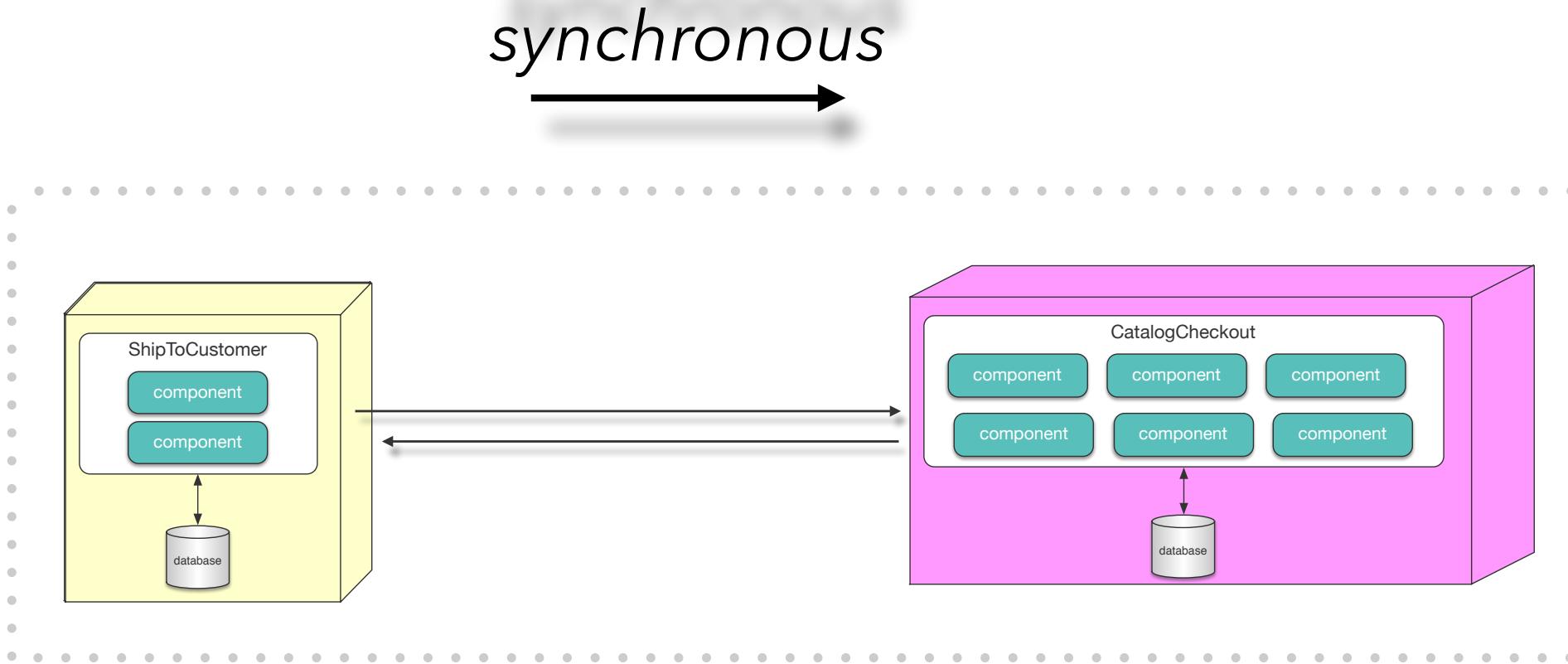


sync | async ?

synchronous

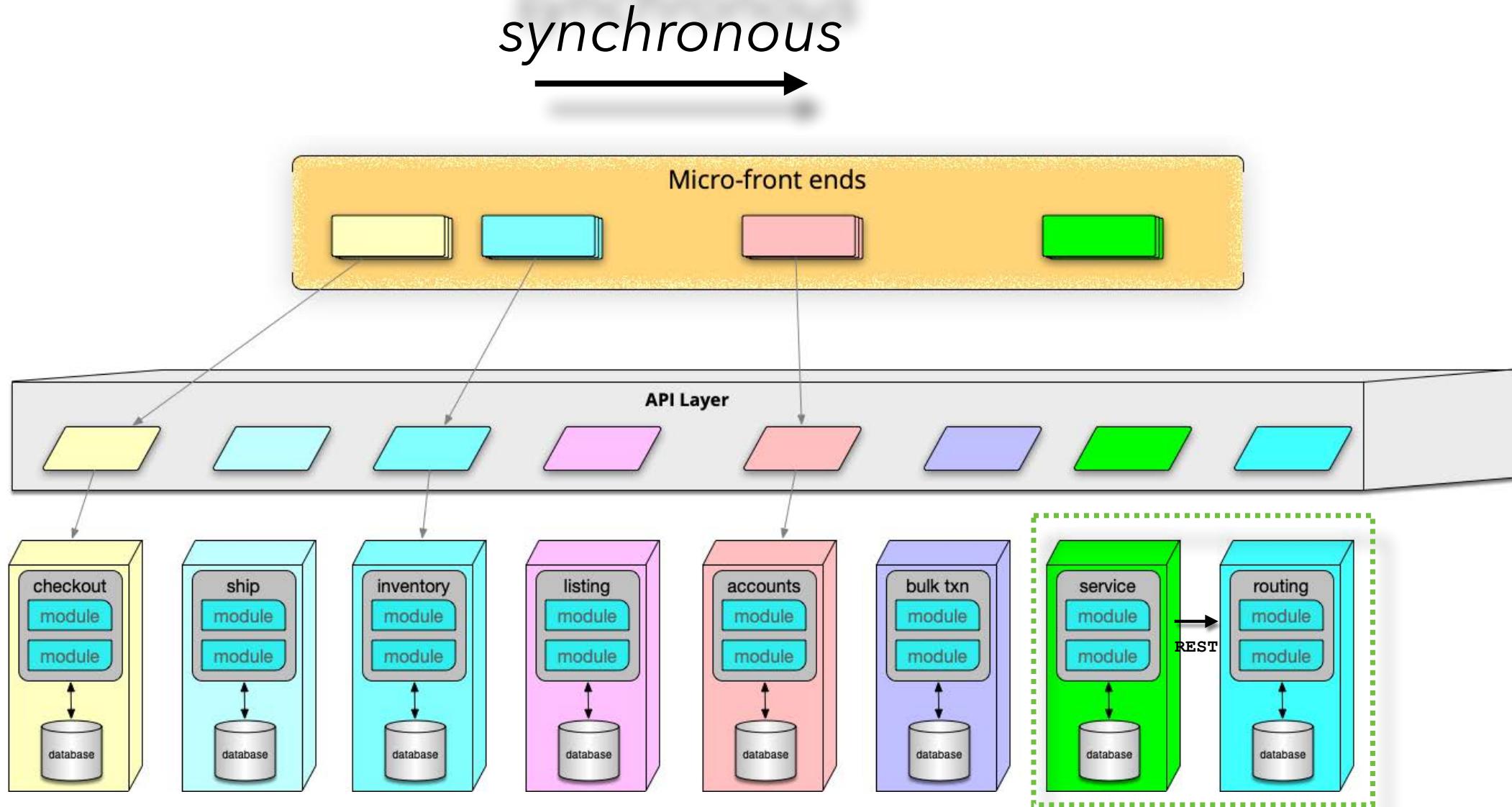


sync | async ?



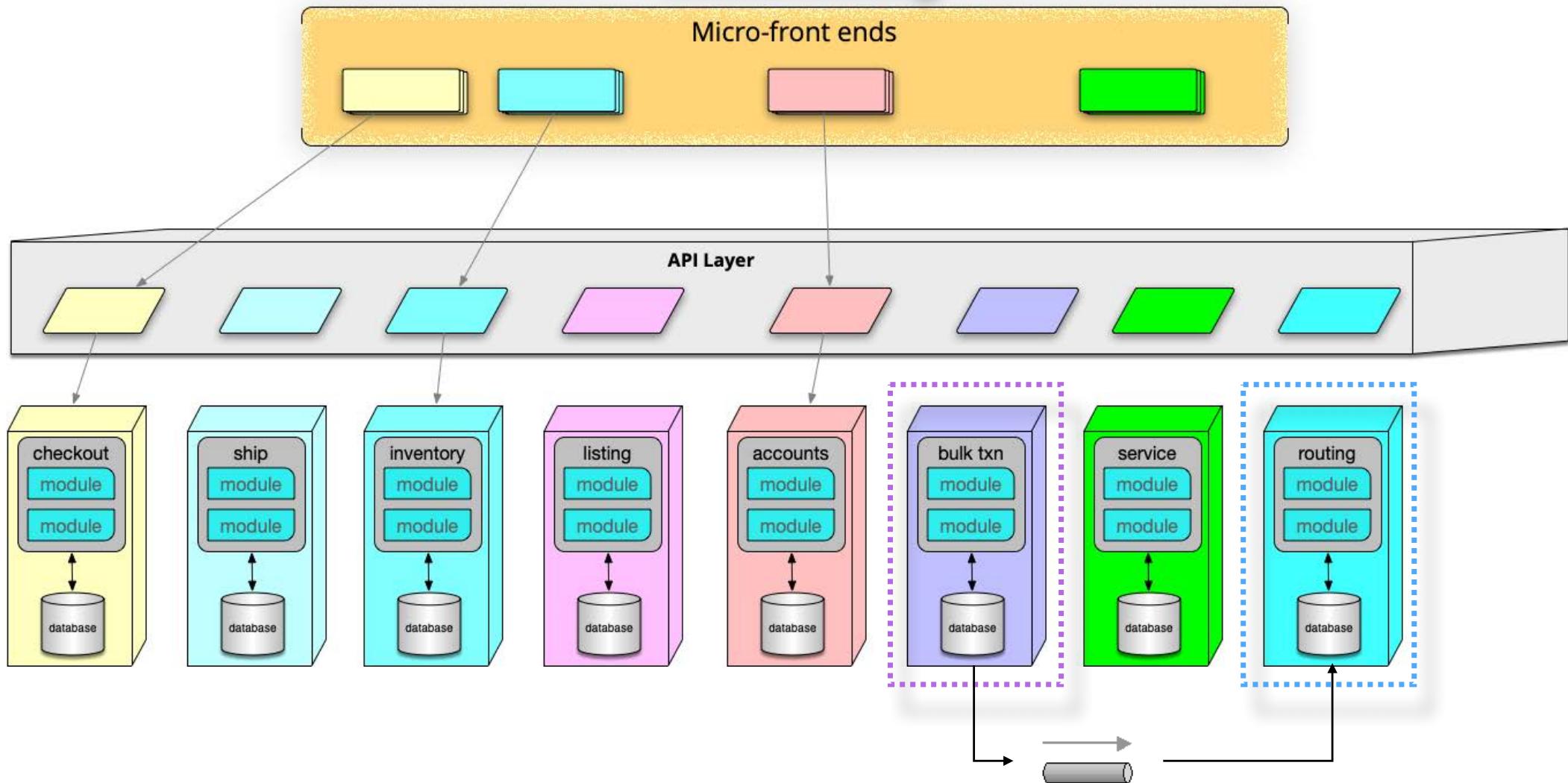
synchronous calls create a
dynamic quantum entanglement
around ***operational*** architecture characteristics

sync | async ?



sync | async ?

asynchronous



asynchronous

tradeoffs

sync

- performance impact on highly interactive systems
 - + easy to model transactional behavior
- creates dynamic quantum entanglements
 - + mimics non-distributed method calls
- creates limitations in distributed architectures
 - + easier to implement

tradeoffs

async

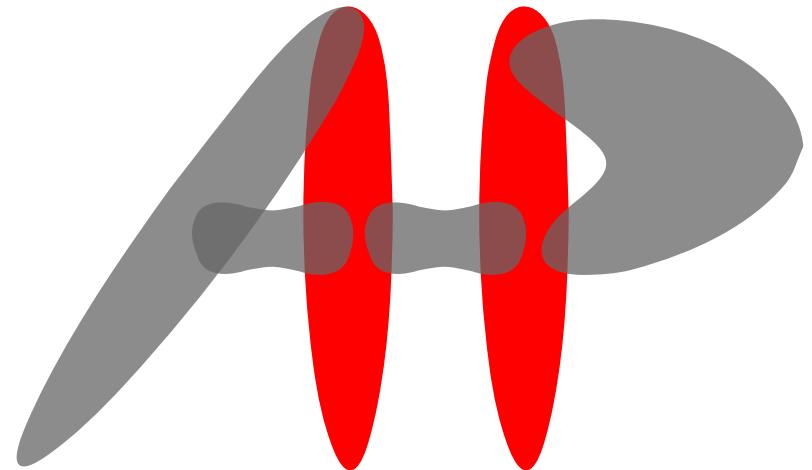
- complex to build, debug
- presents difficulties for transactional behaviors (prefer BASE)
- error handling
- + allows highly decoupled systems
- + common performance tuning technique
- + high performance and scale

synchronous | asynchronous ?



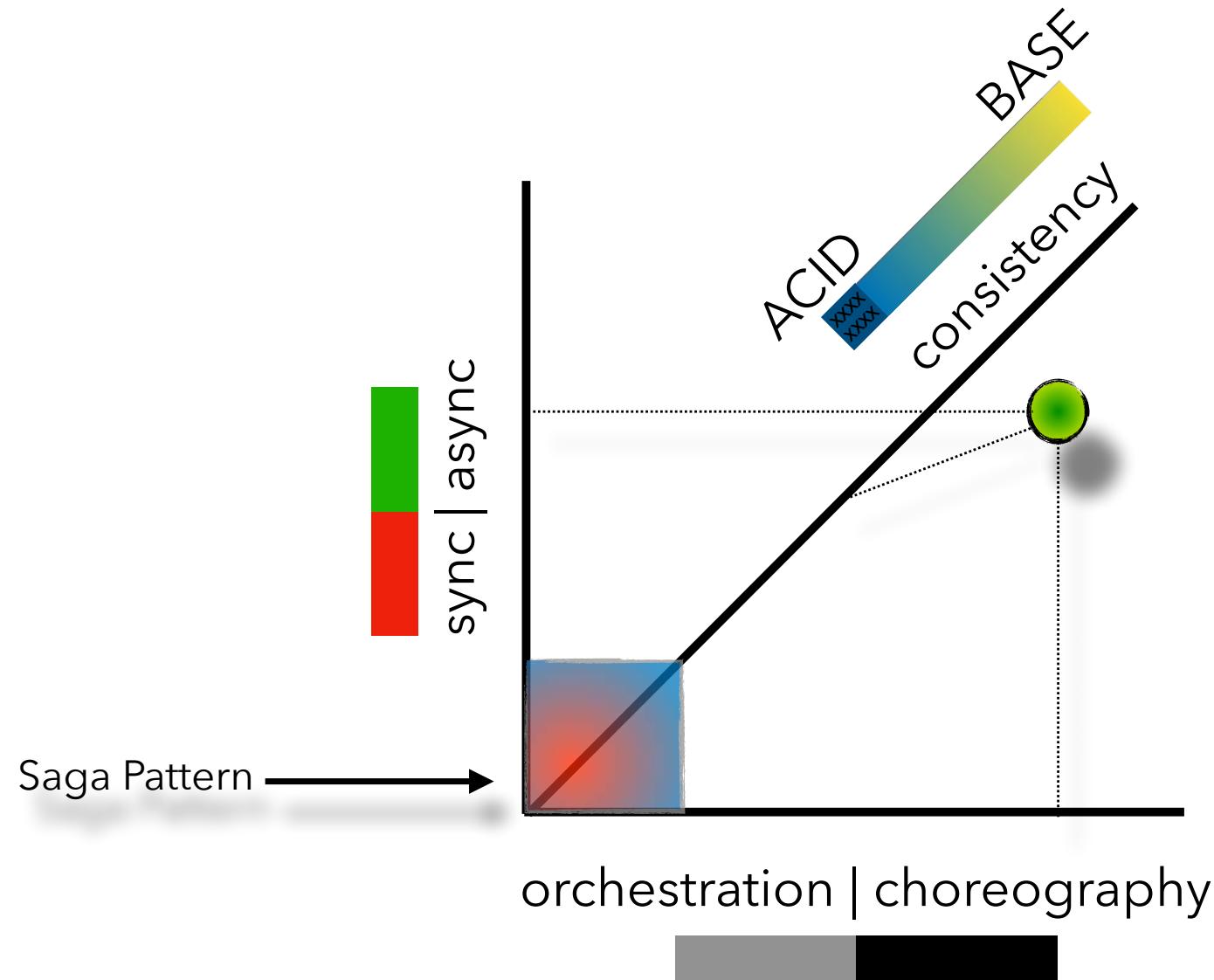
- architecture concern: synchronous versus asynchronous
- exception: implementation detail is important/unique
- design: how to implement the architecture concern
- use layers in drawing tools to differentiate

architectural quantum 2021



an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling.

dimensions of dynamic quantum coupling

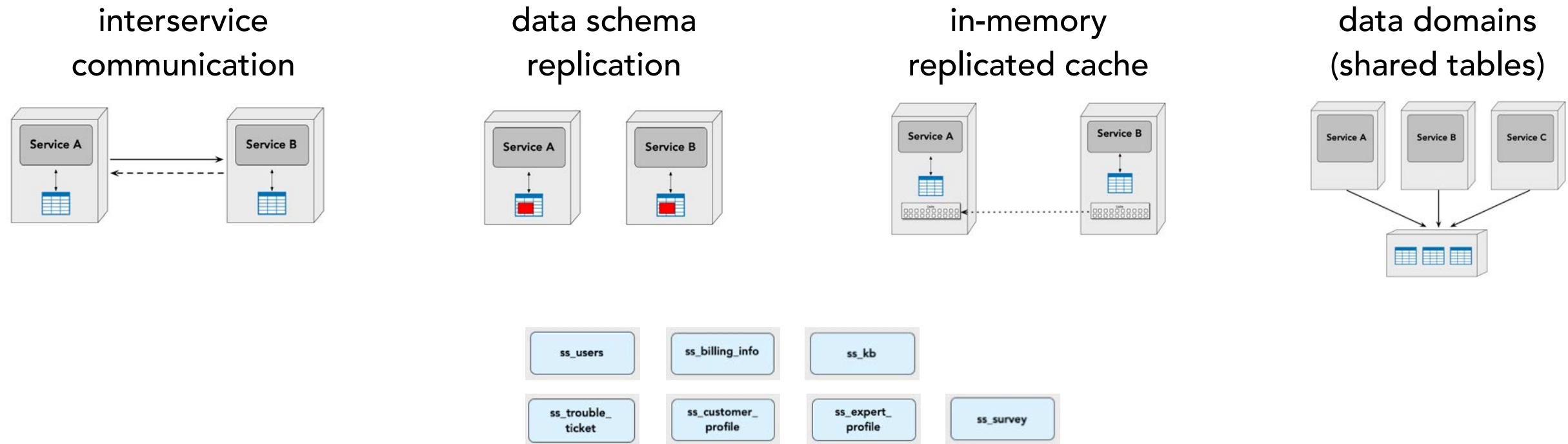




Data Ownership and Access

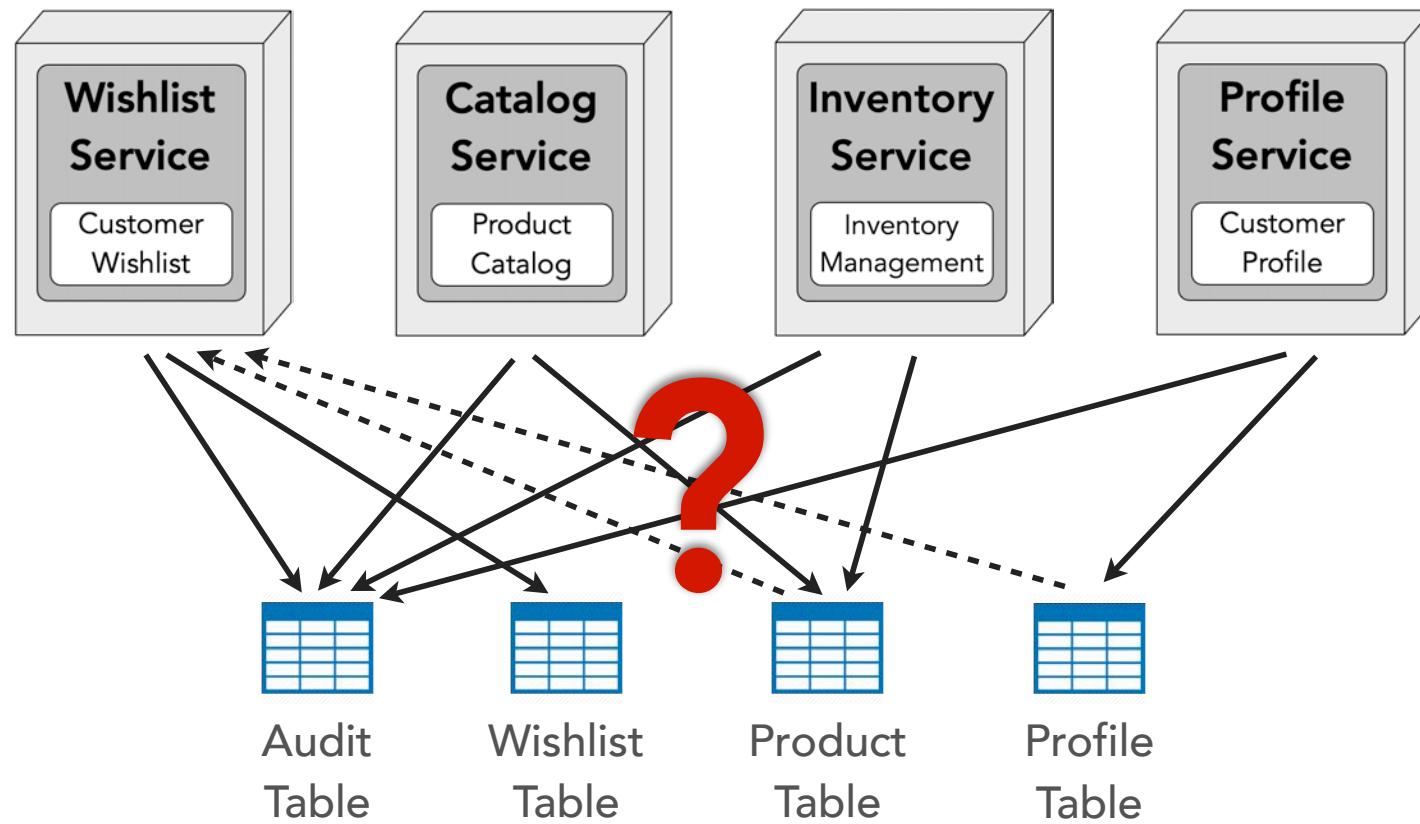
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices for data access are indicated below. Don't forget you can use any combination of these depending on the data.

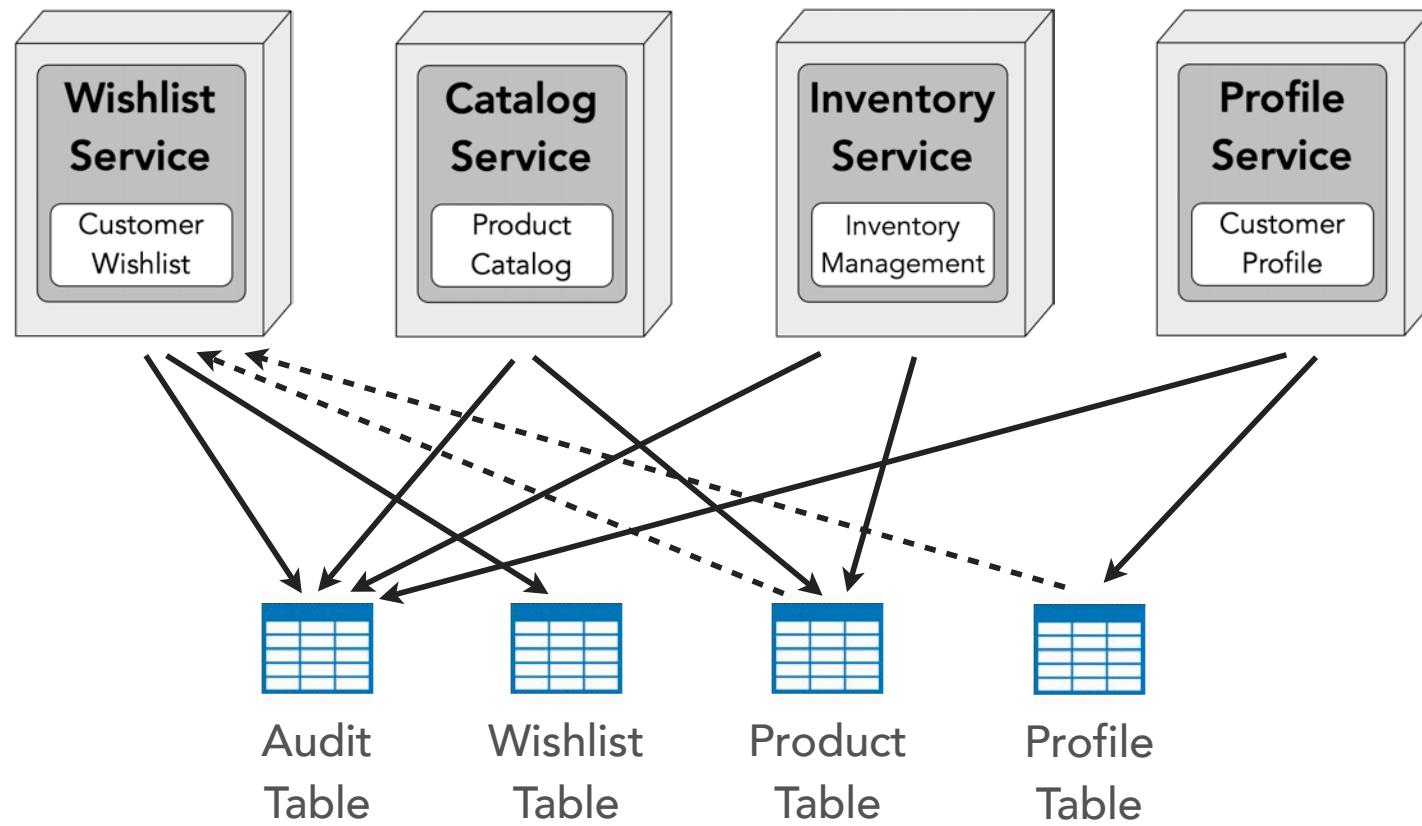


data ownership

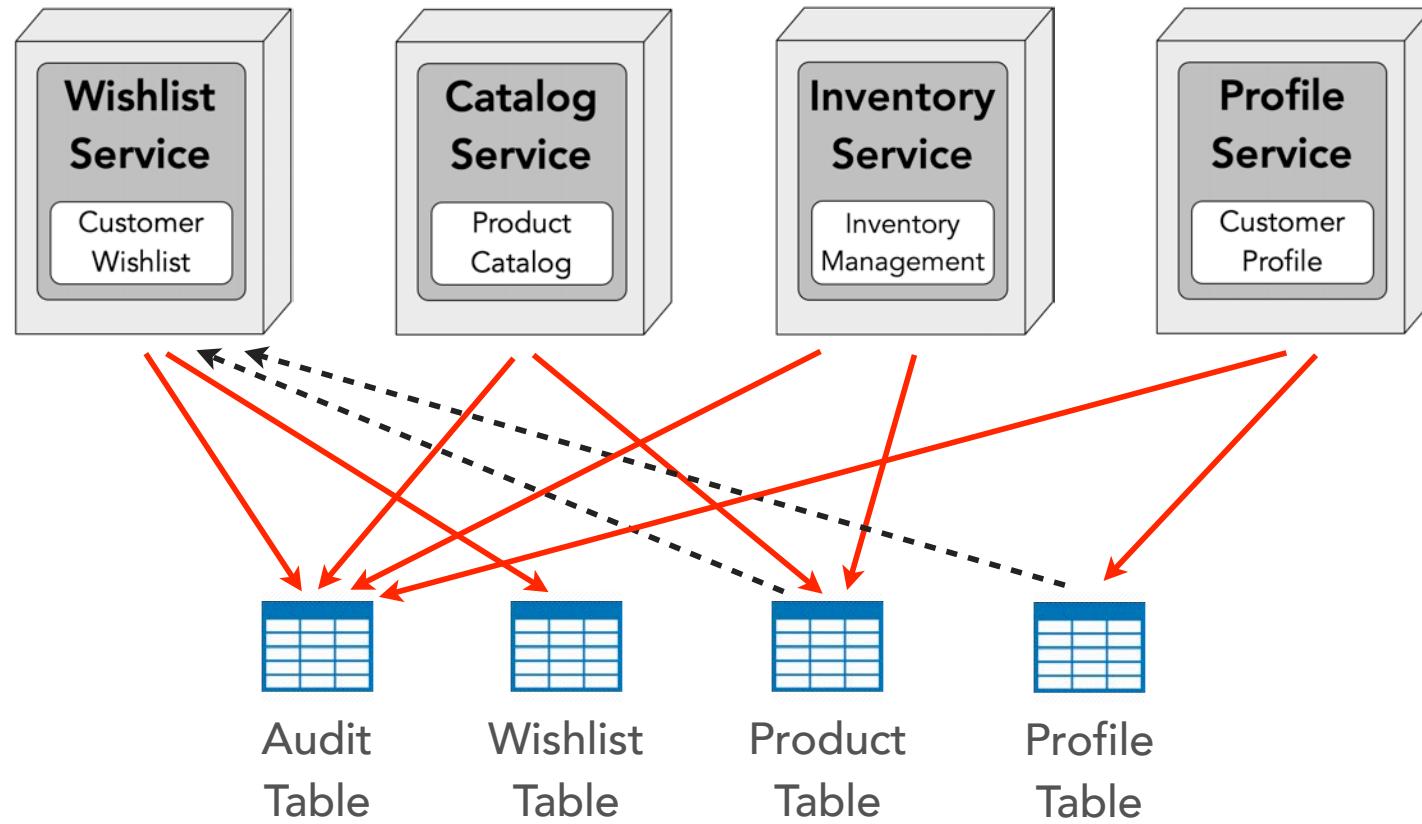
“How do I assign table ownership to my services?”



data ownership

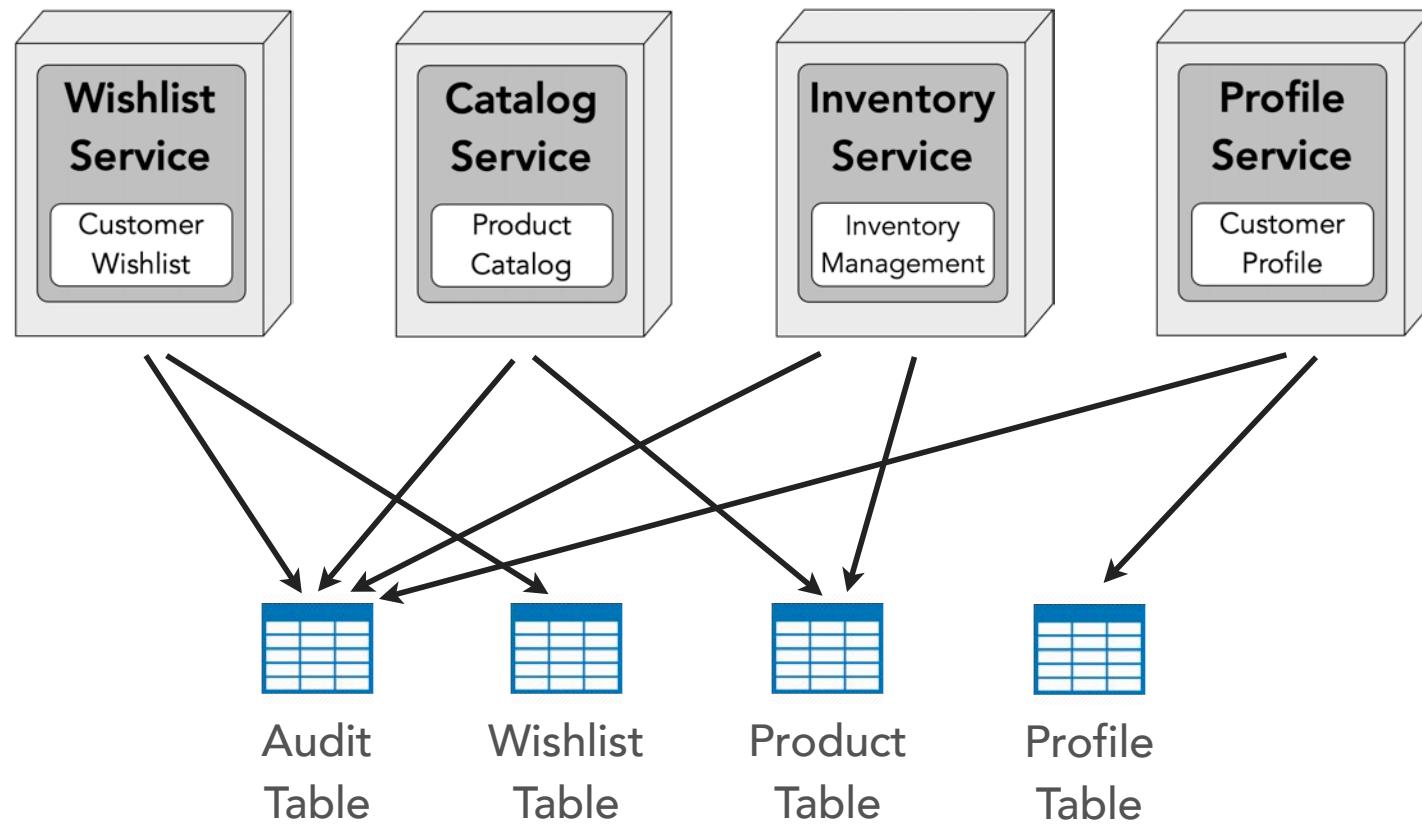


data ownership

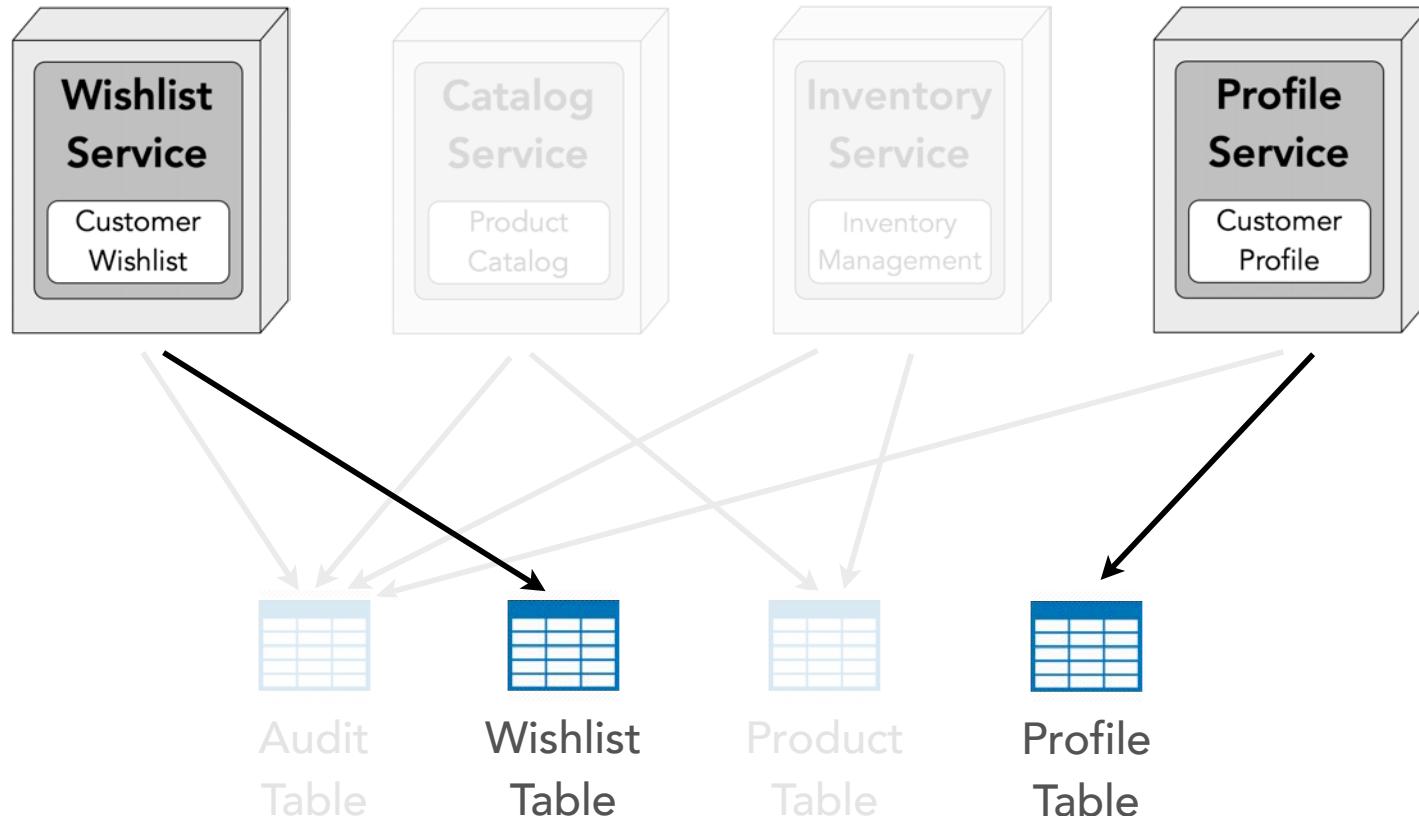


the owner is the one who writes to the table

data ownership

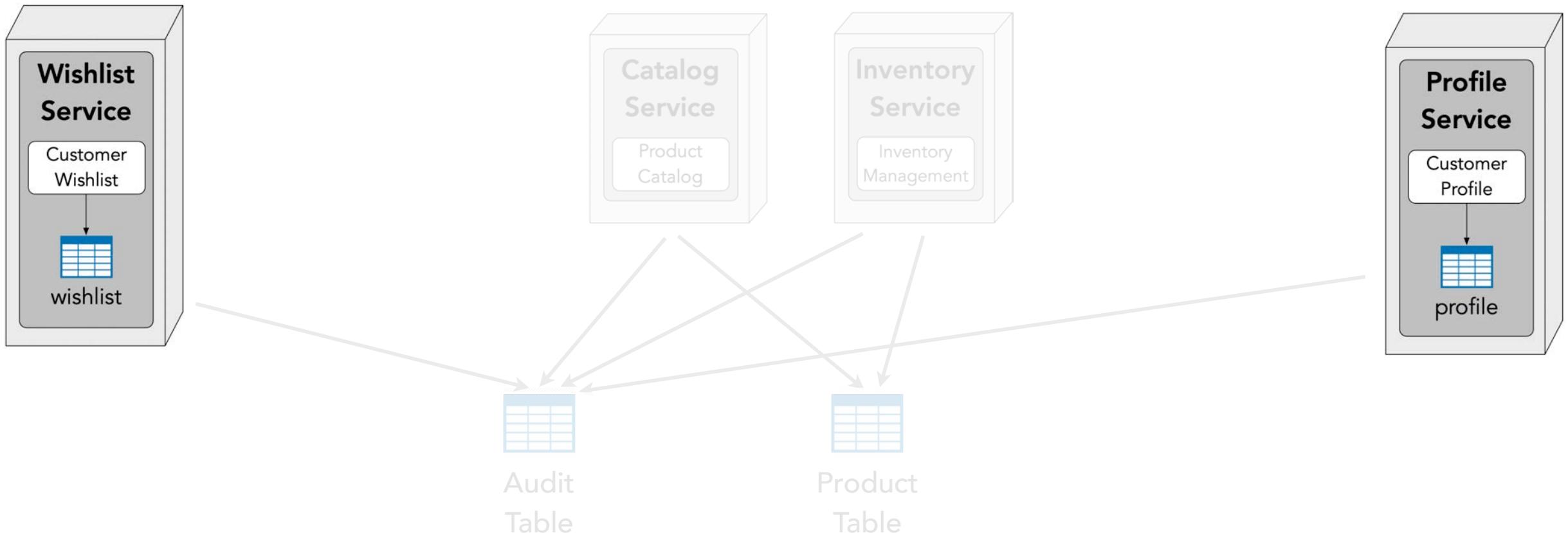


data ownership



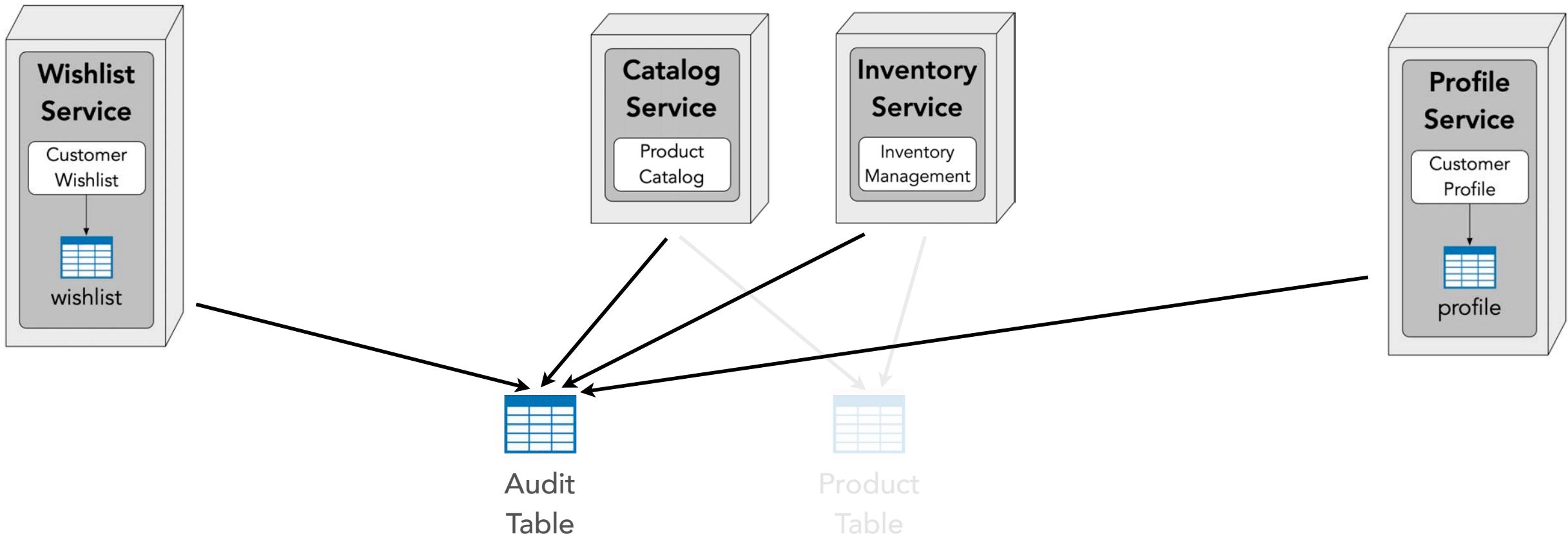
scenario: single ownership

data ownership



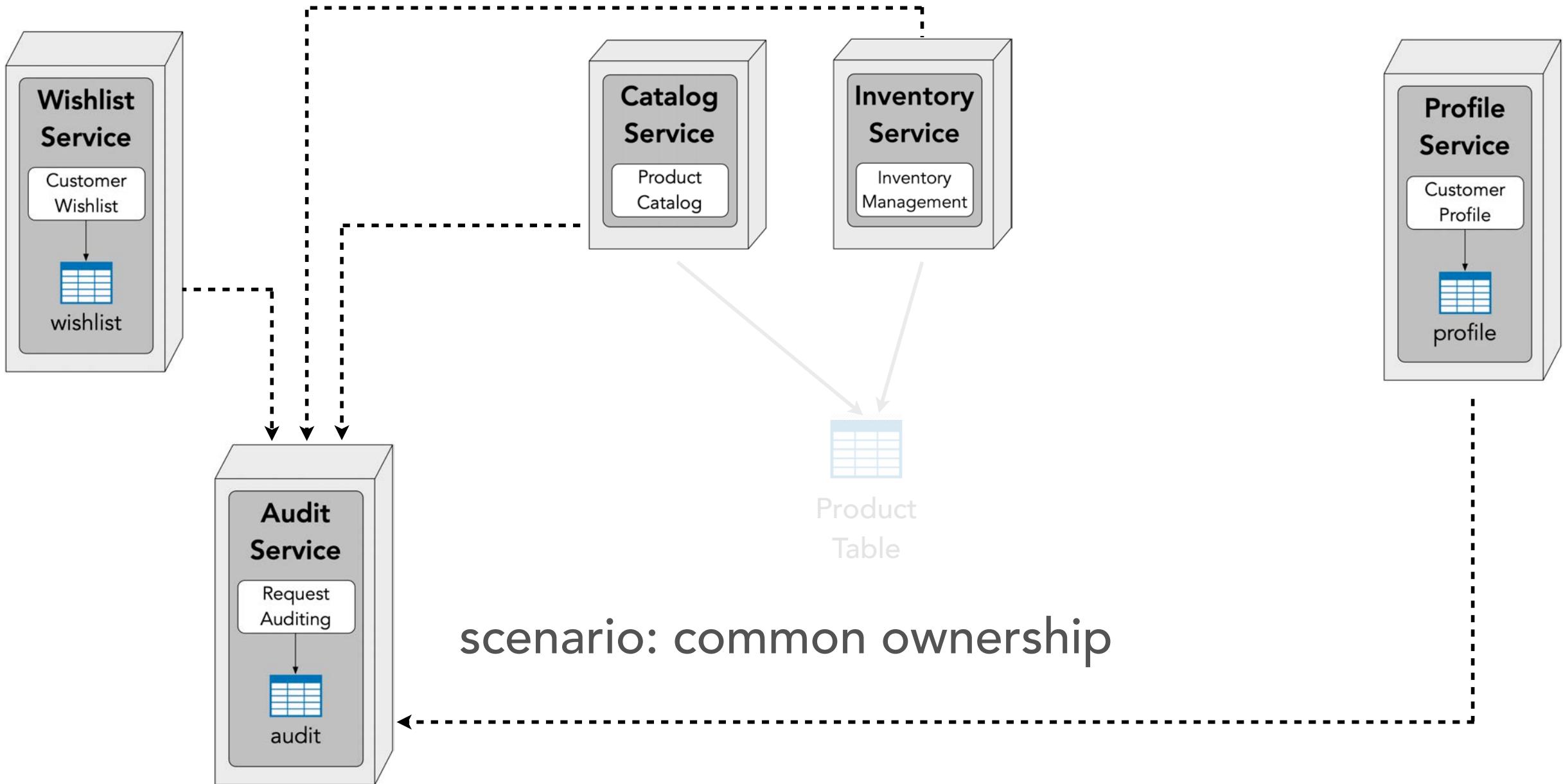
scenario: single ownership

data ownership

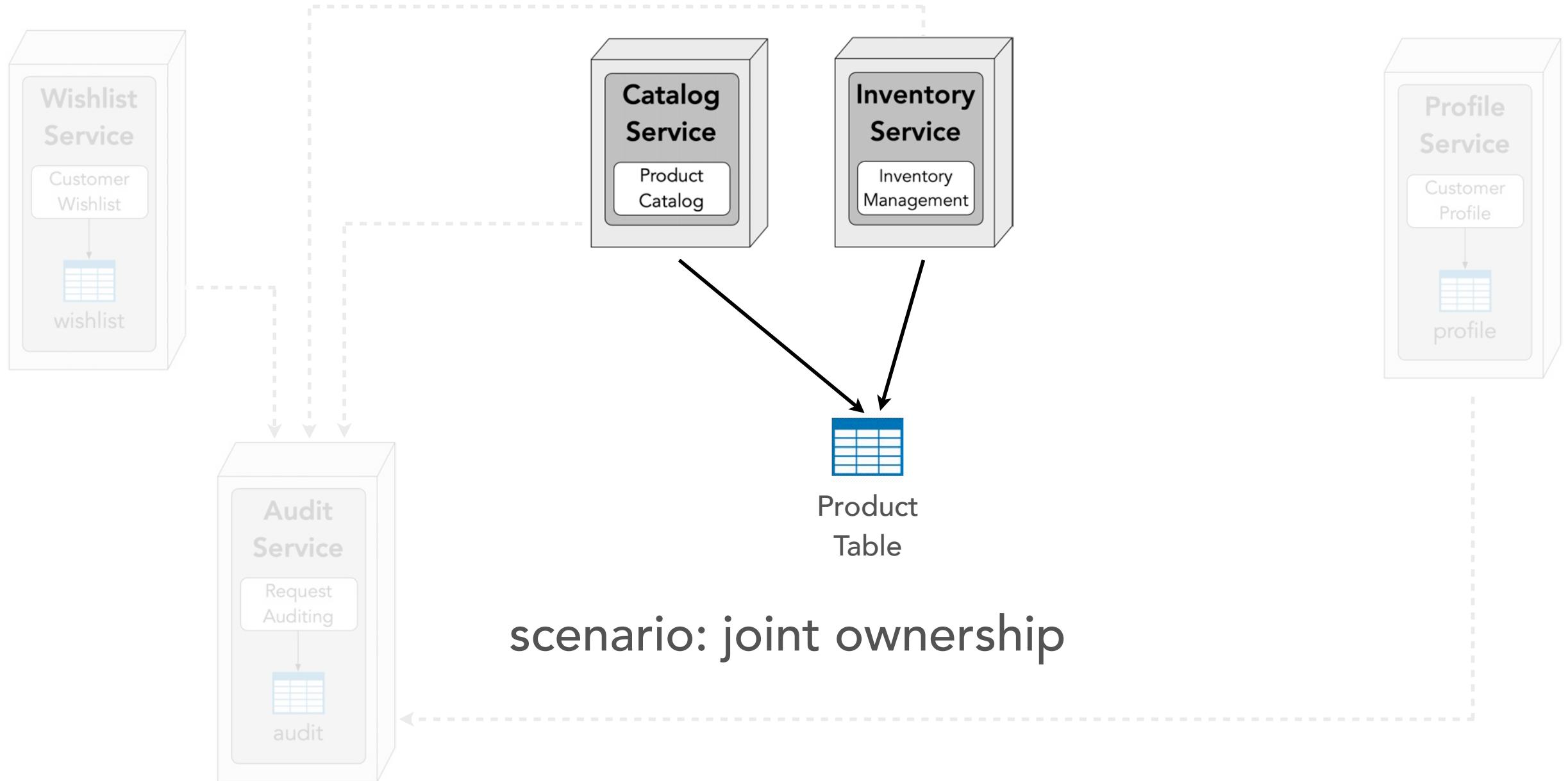


scenario: common ownership

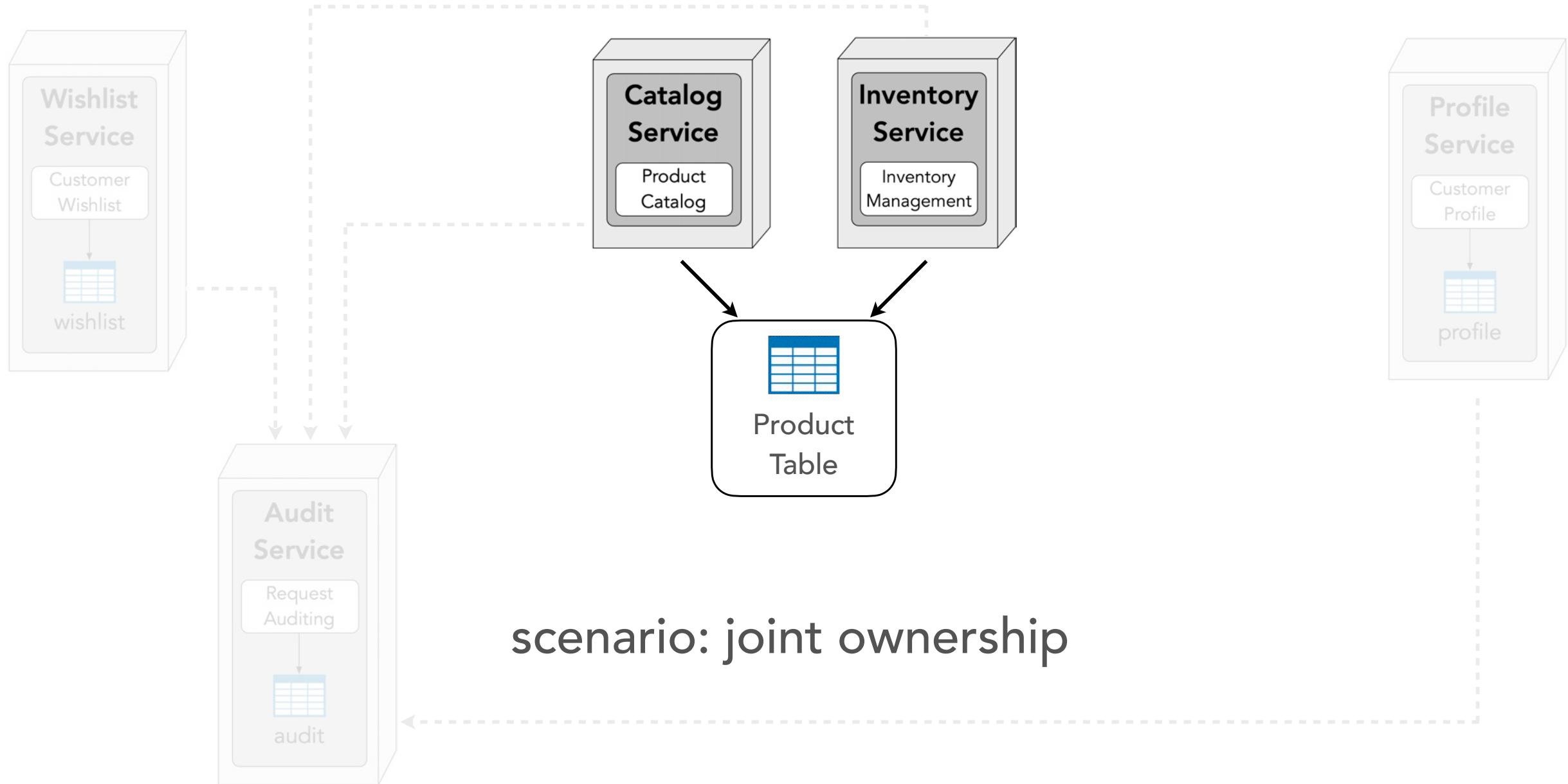
data ownership



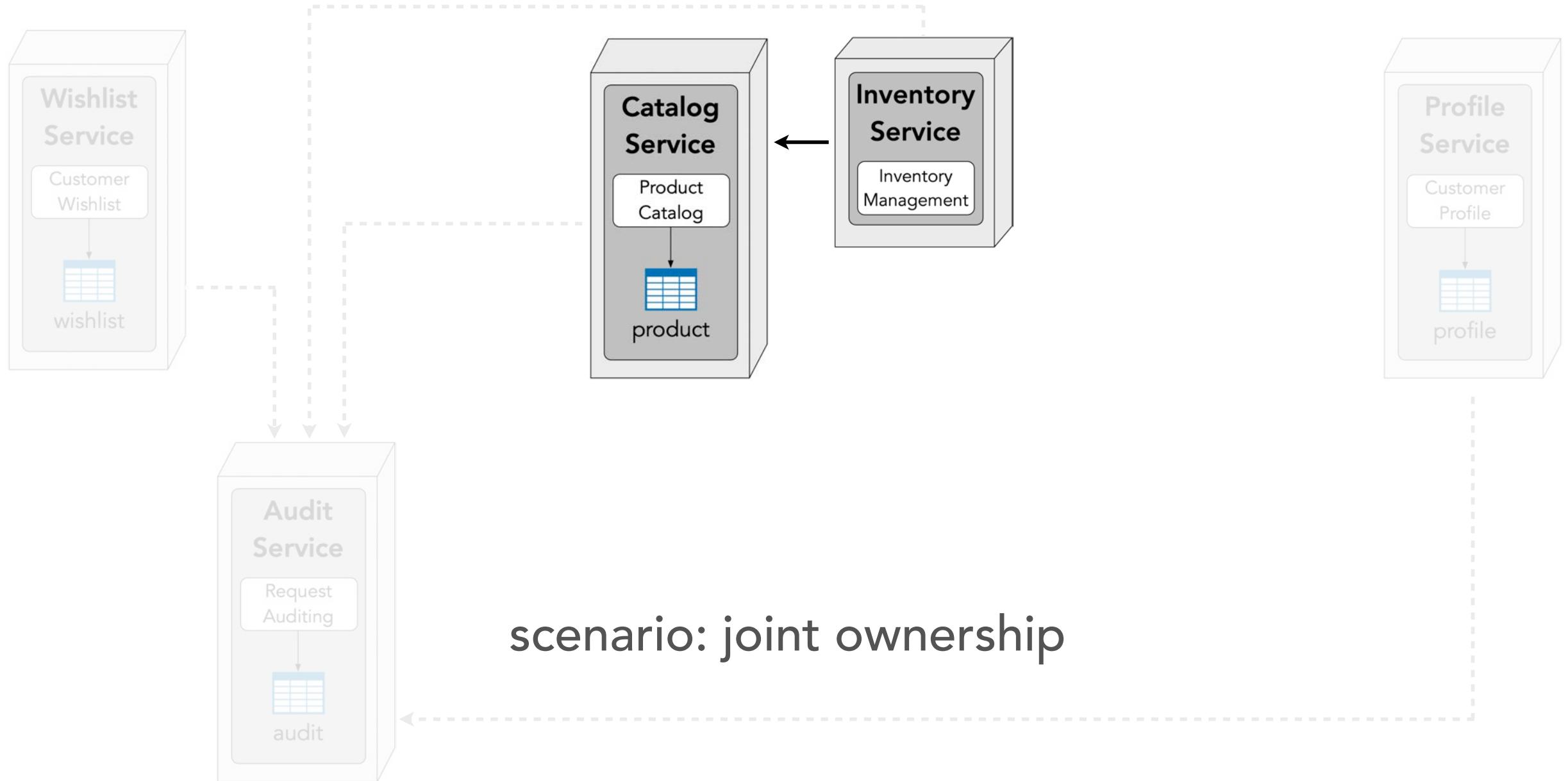
data ownership



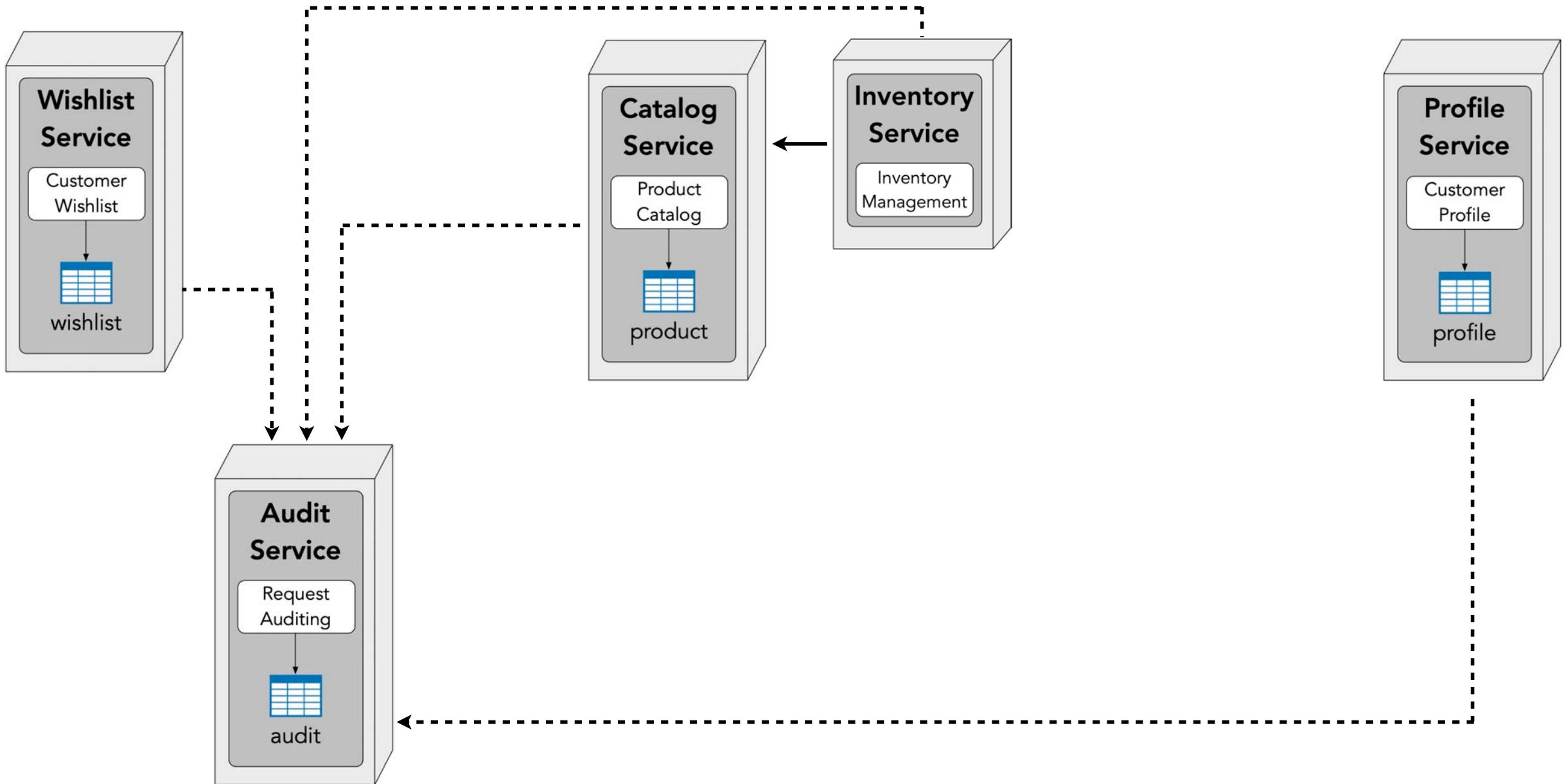
data ownership



data ownership

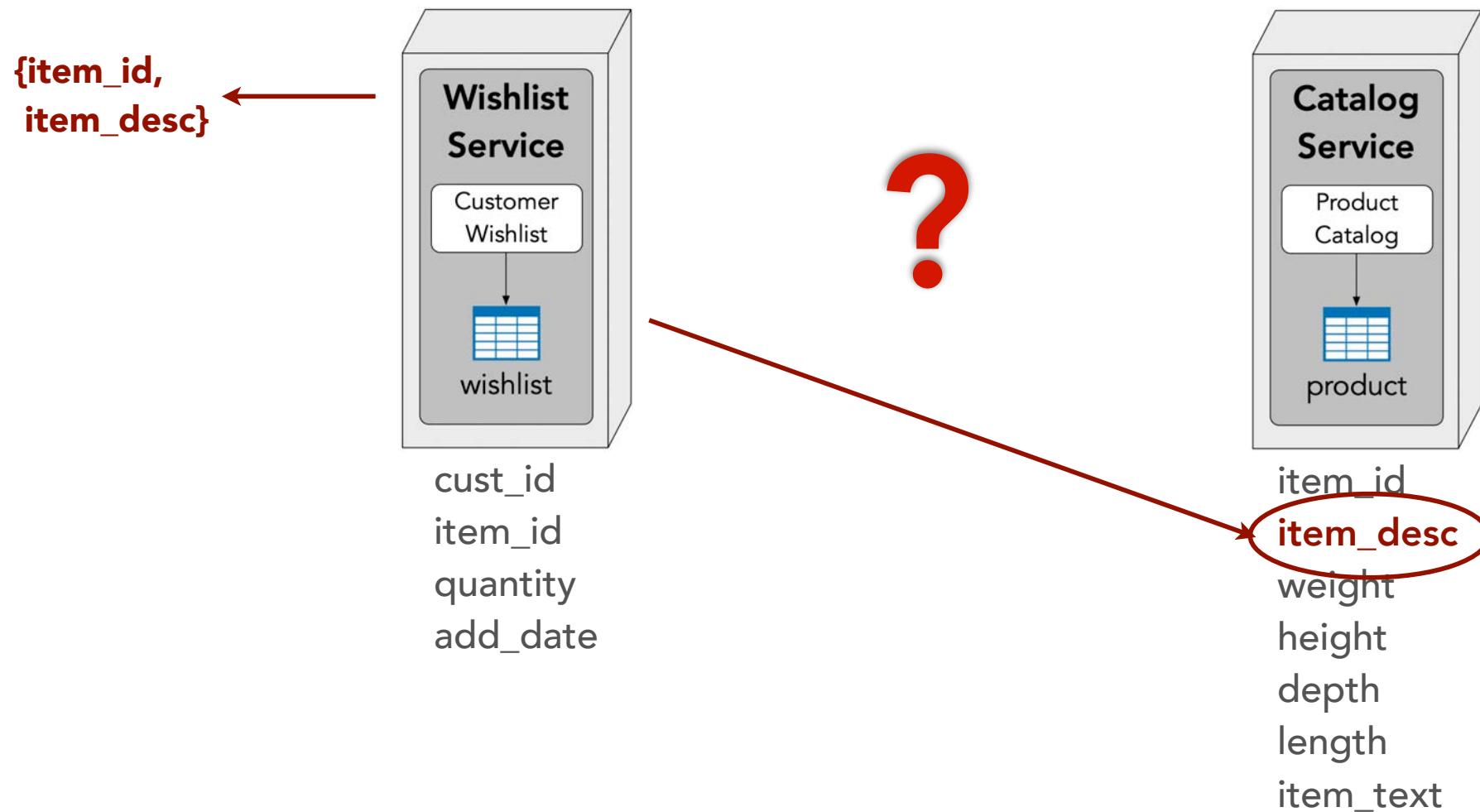


data ownership



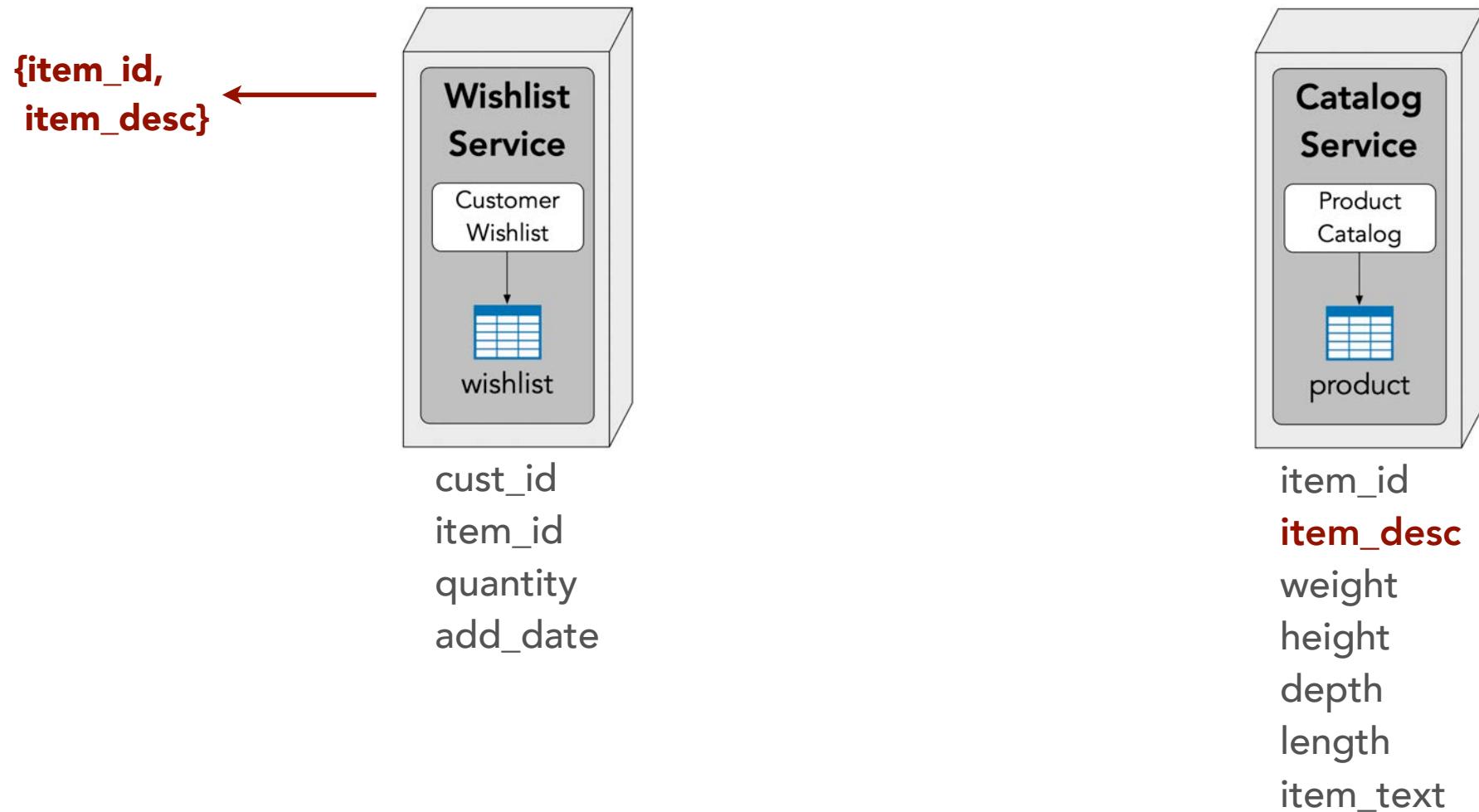
data access

“once I break apart my data, how do I access data I don’t own?”



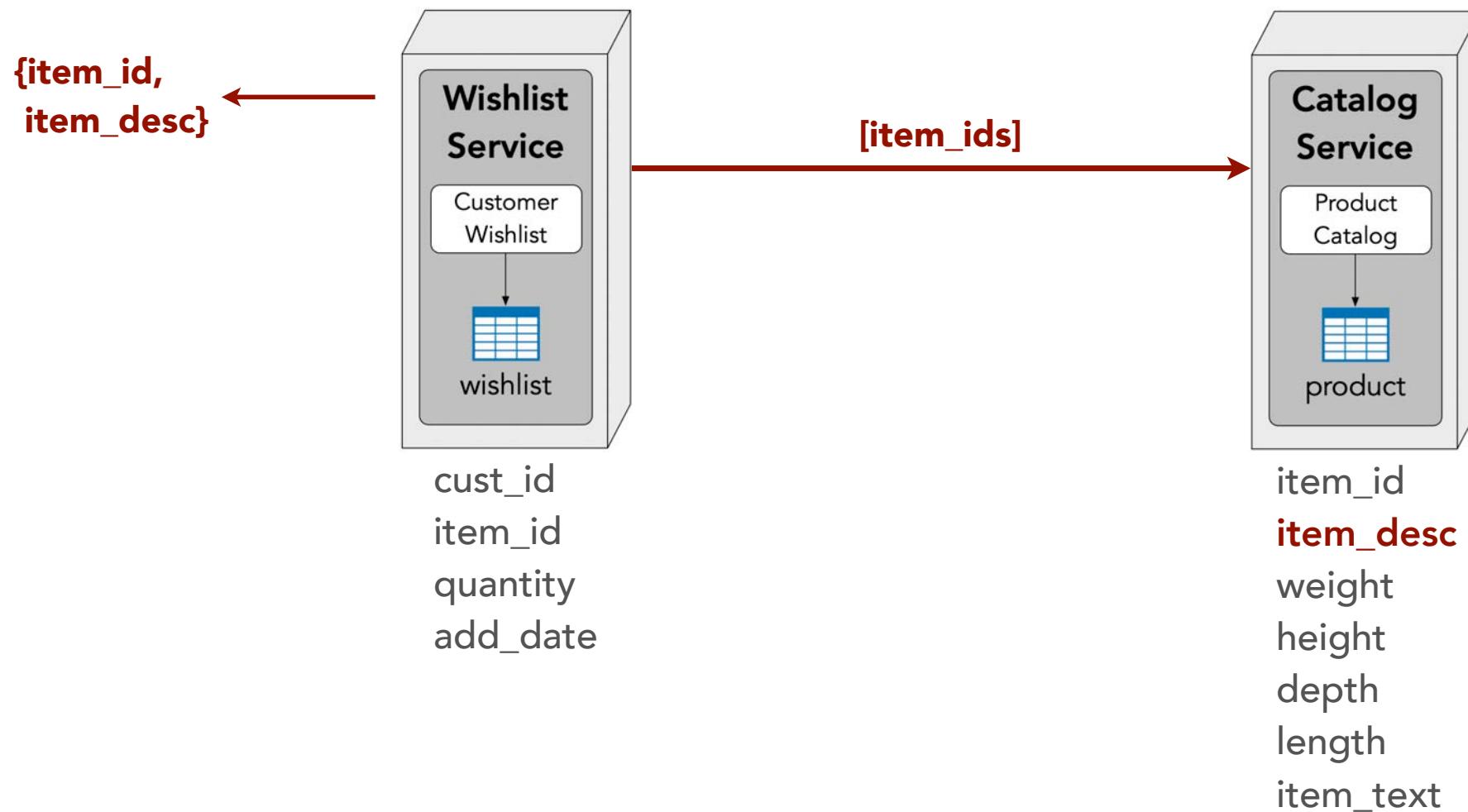
data access

option 1: interservice communication



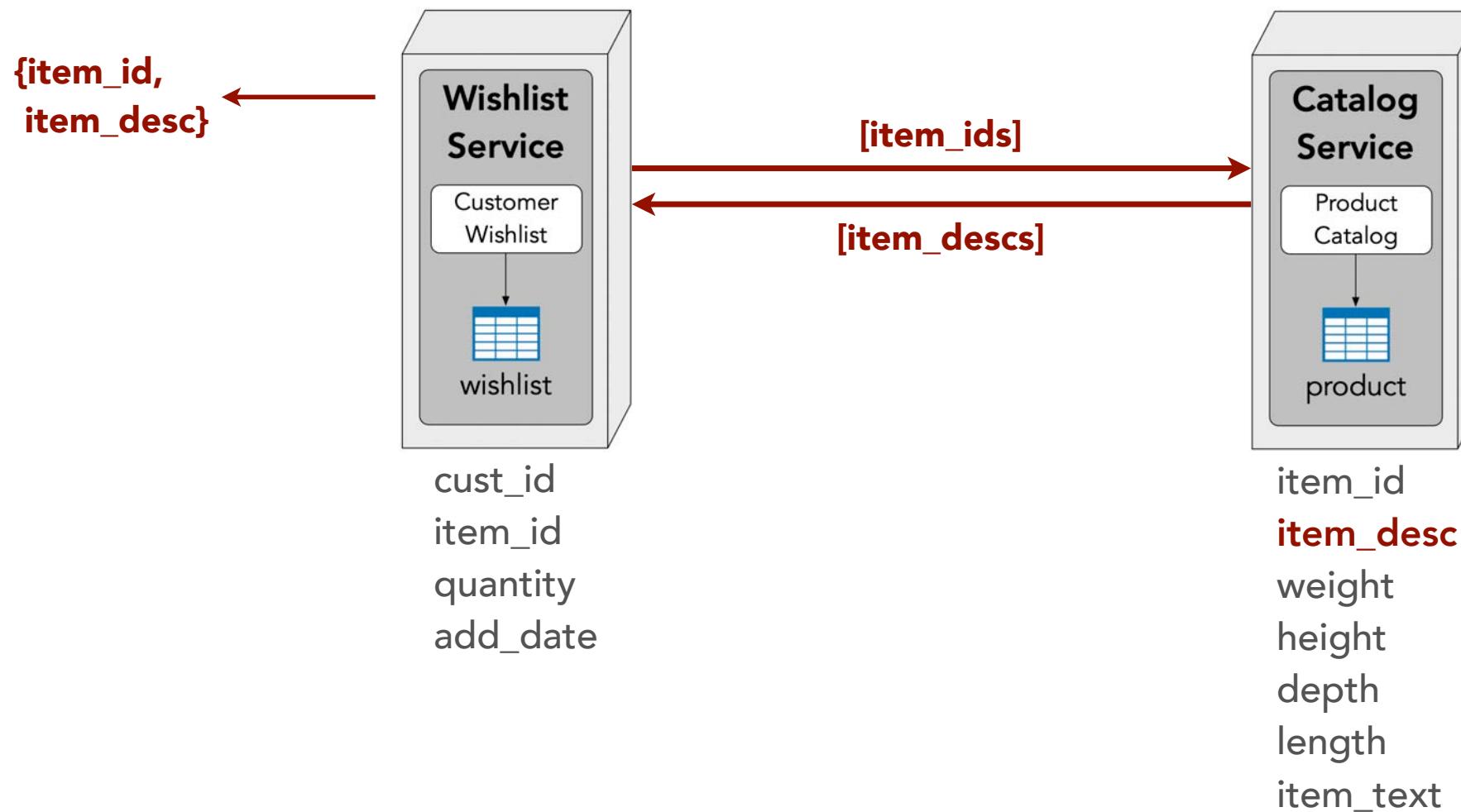
data access

option 1: interservice communication



data access

option 1: interservice communication



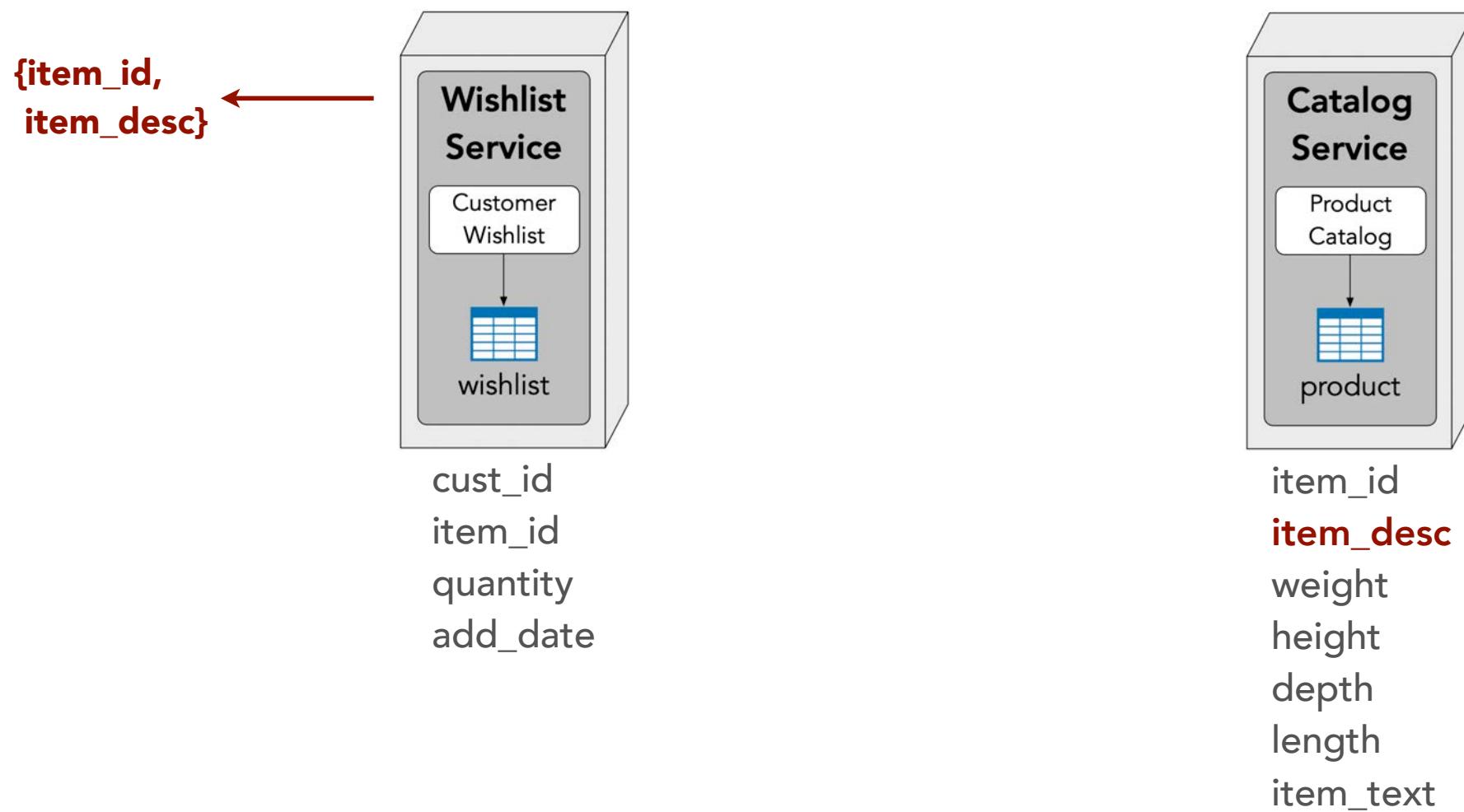
tradeoffs

option 1 - interservice communication

- network and security latency
- scalability and throughput
- fault tolerance (dependencies)

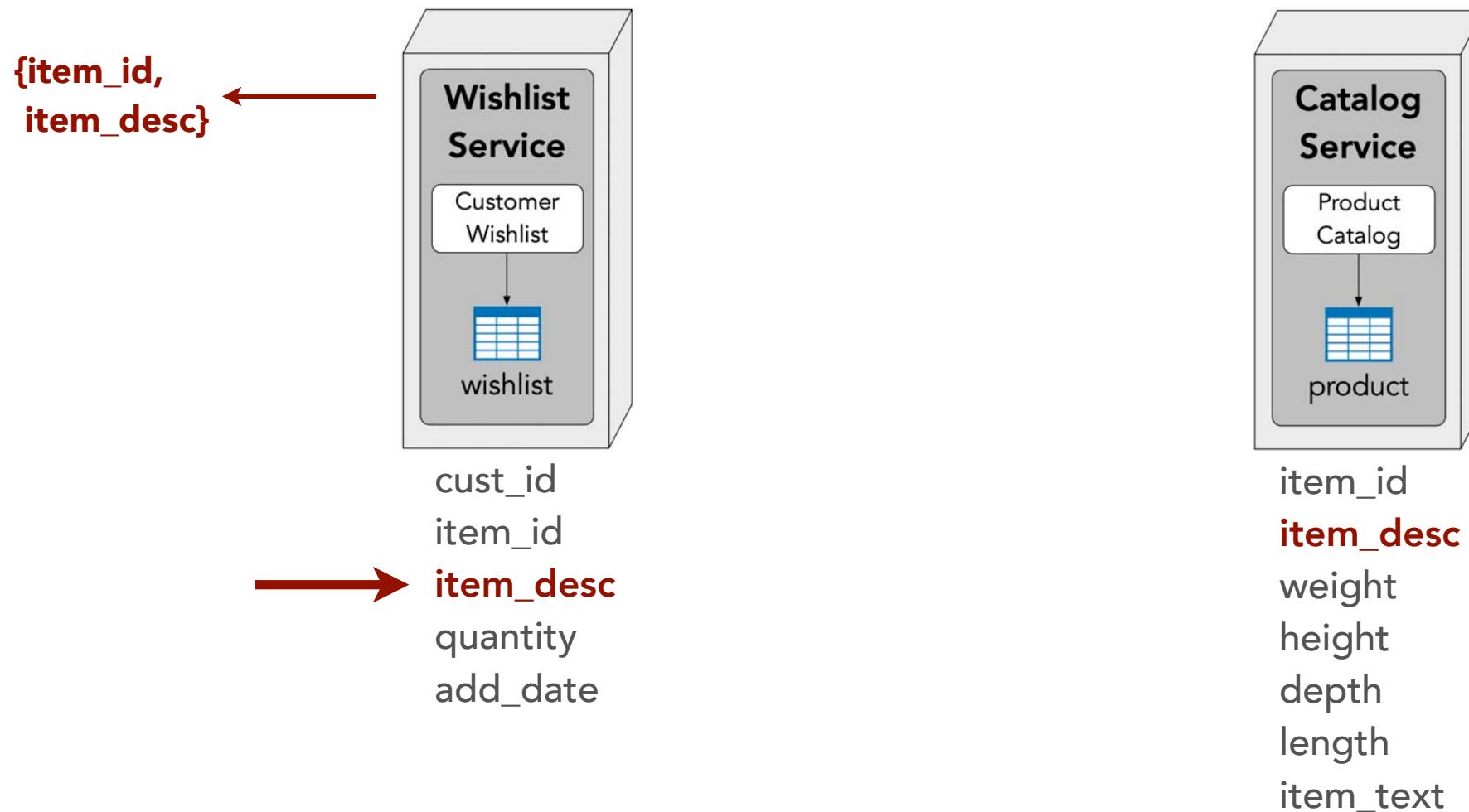
data access

option 2: data replication



data access

option 2: data replication



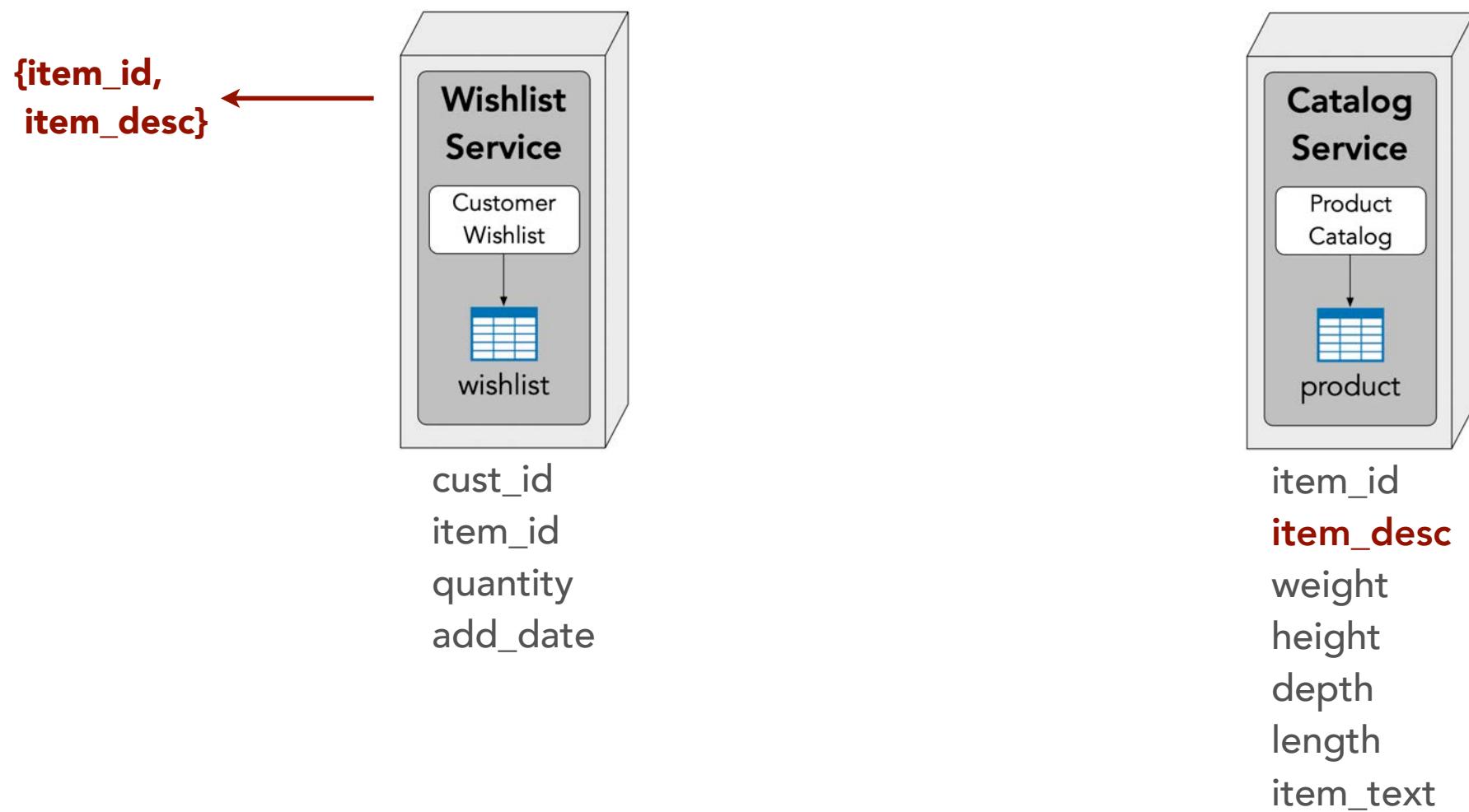
tradeoffs

option 2 - data replication

- + network and security latency (1)
 - data consistency issues
- + scalability and throughput (1)
 - data ownership issues
- + fault tolerance (1)

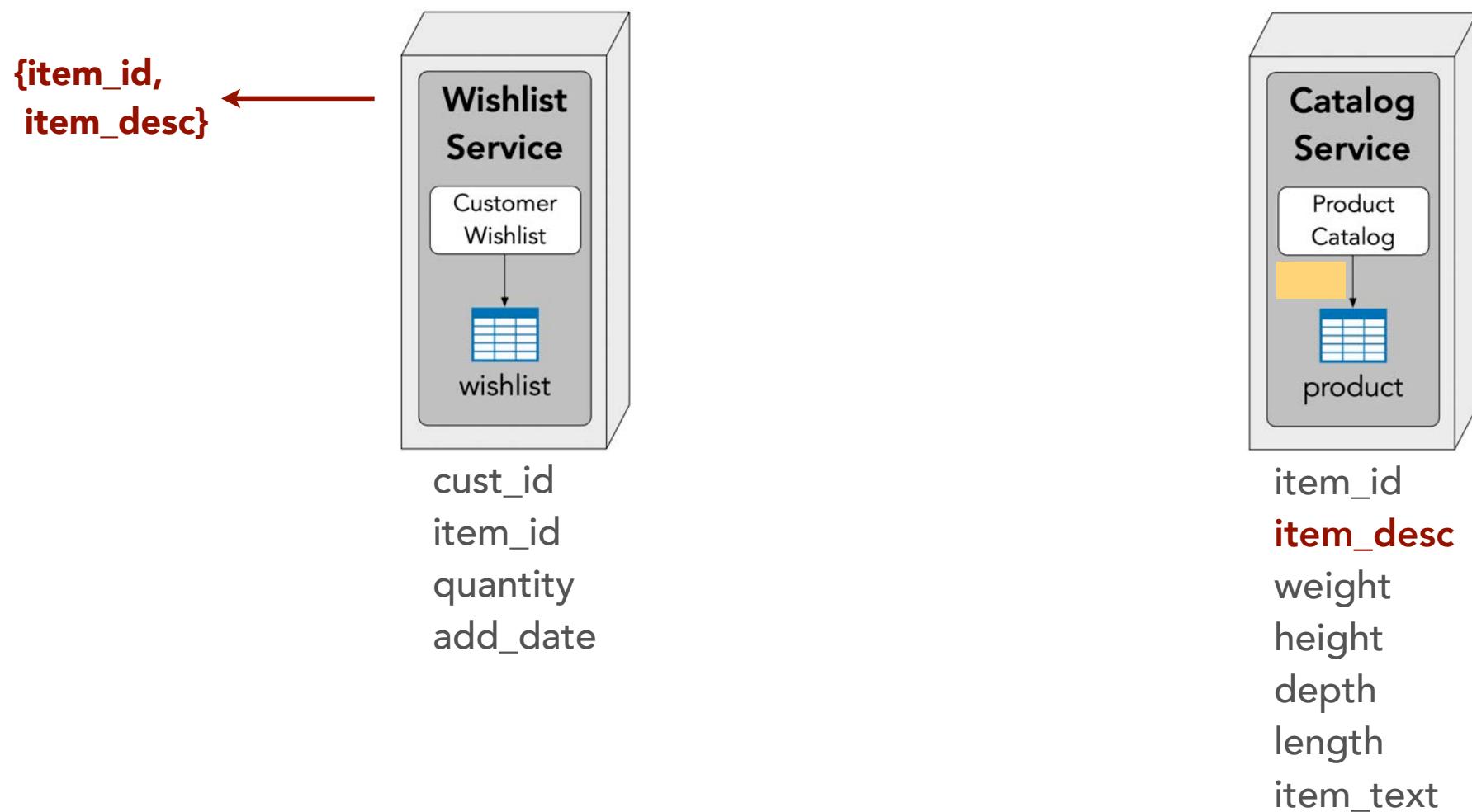
data access

option 3: in-memory replicated cache



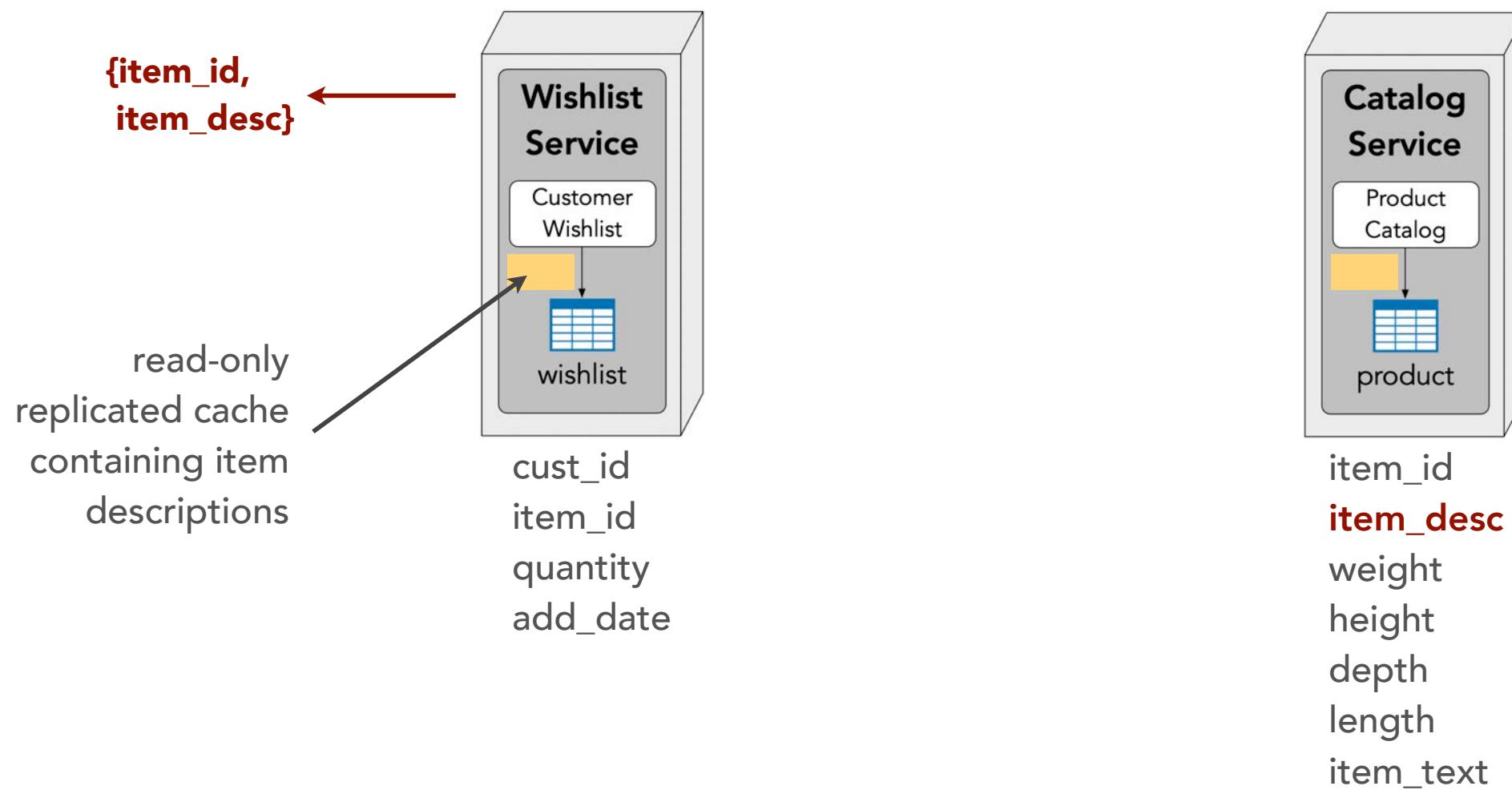
data access

option 3: in-memory replicated cache



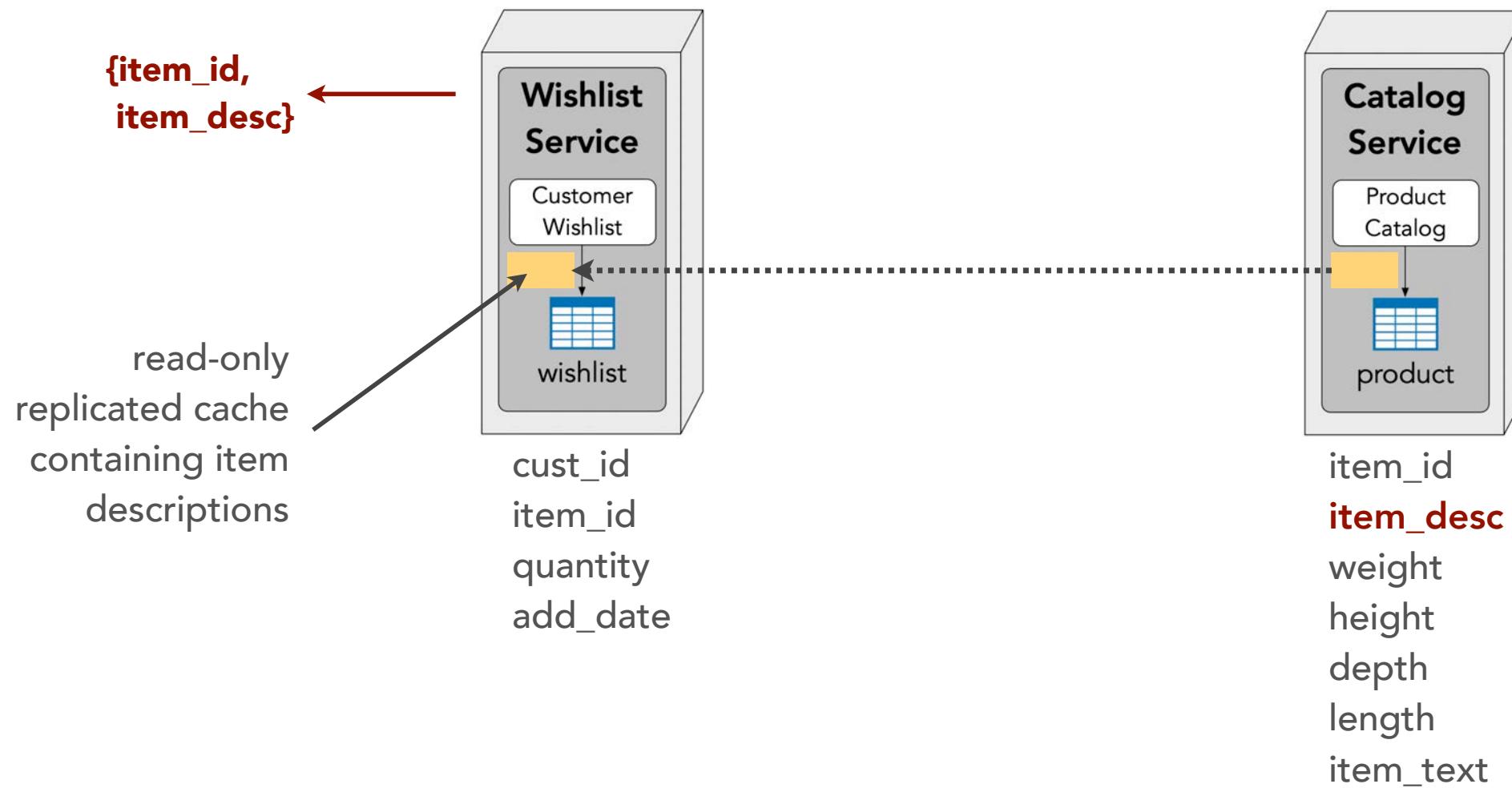
data access

option 3: in-memory replicated cache



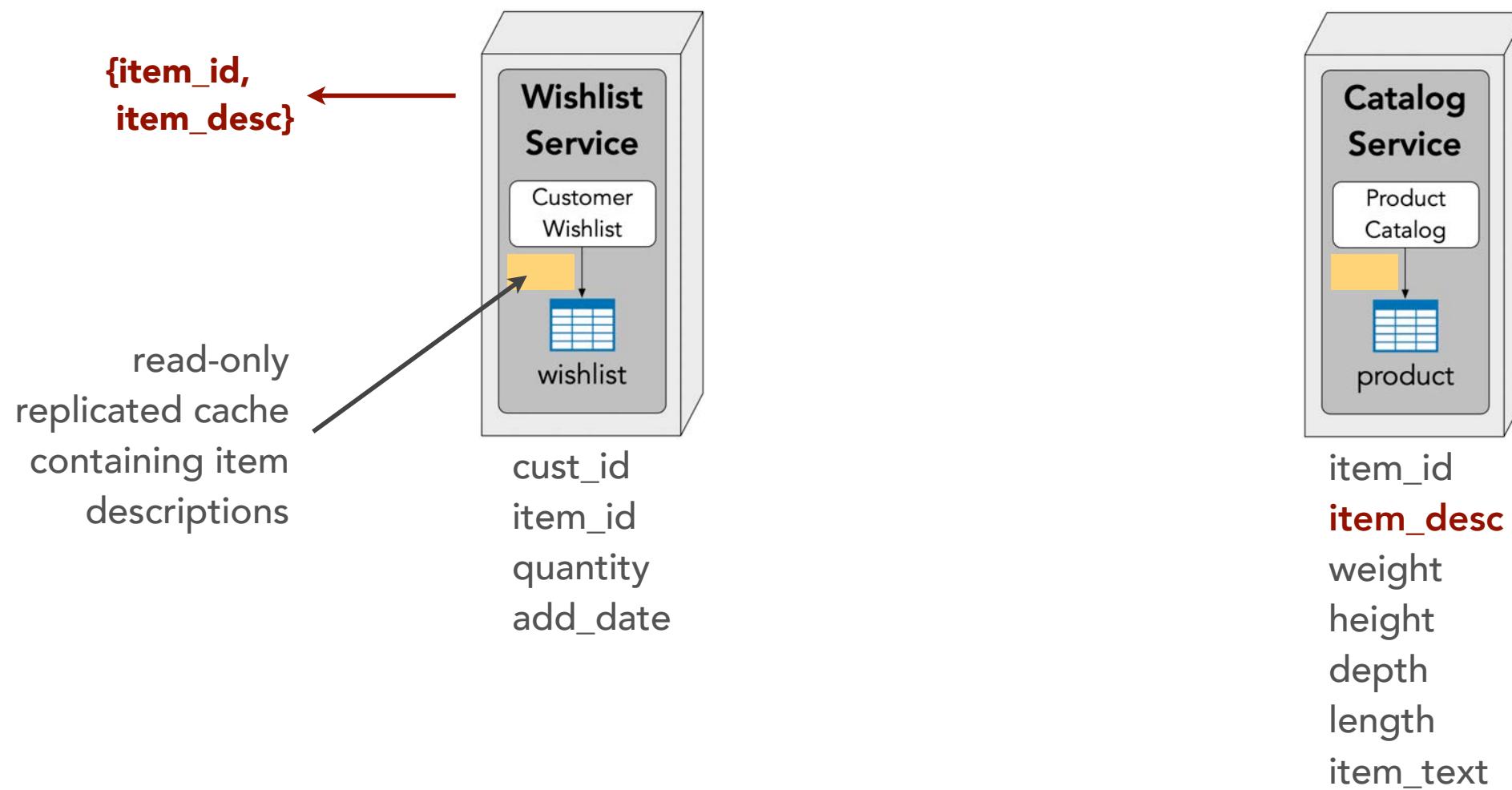
data access

option 3: in-memory replicated cache



data access

option 3: in-memory replicated cache



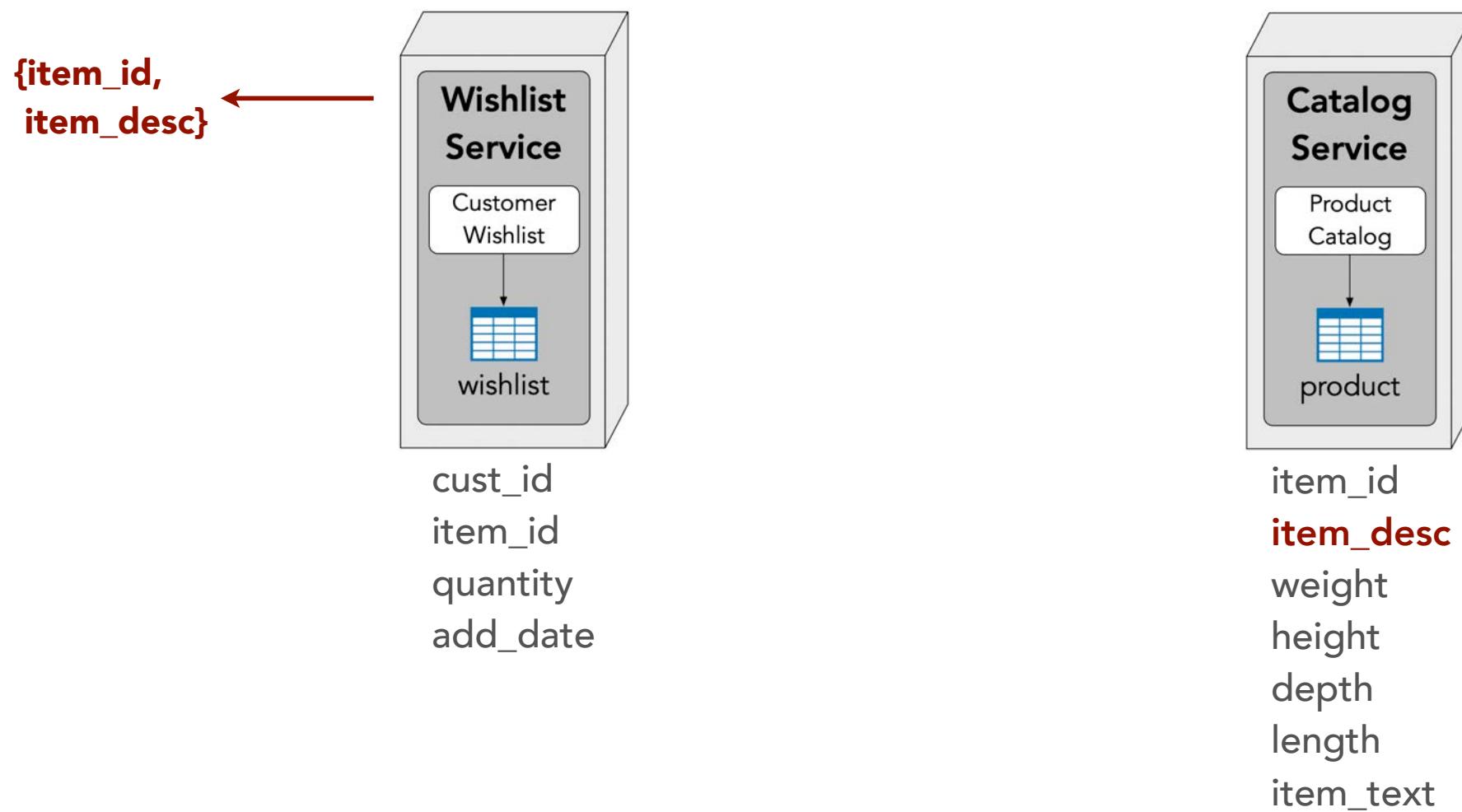
tradeoffs

option 3 - in-memory replicated cache

- + network and security latency (1)
- + scalability and throughput (1) – eventually consistent
- + fault tolerance (1) – data volume issues
- + data consistency (2) – update rate issues
- + data ownership (2)

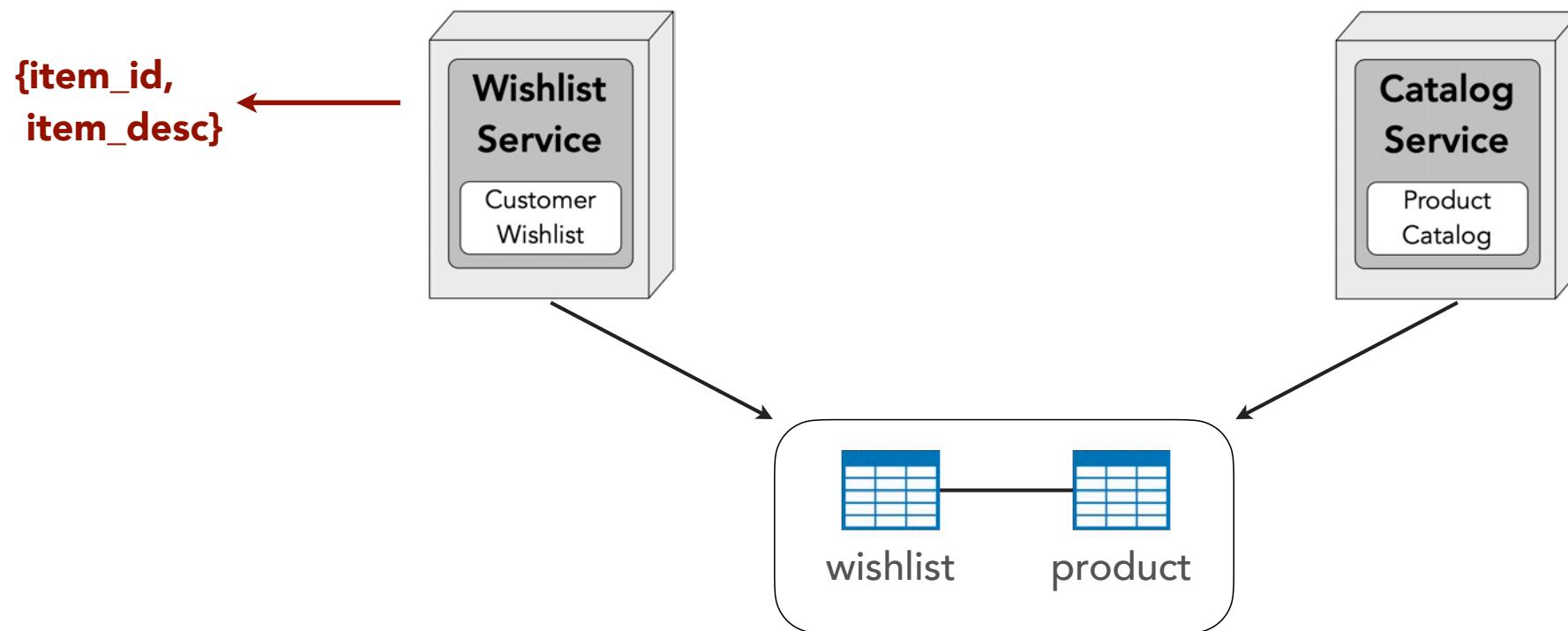
data access

option 4: data domain



data access

option 4: data domain



tradeoffs

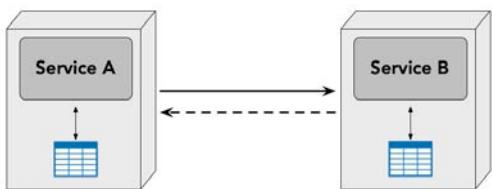
option 4 - data domain

- + network and security latency (1)
- + scalability and throughput (1)
- + fault tolerance (1) – change control
- + data consistency (2) – data ownership (read/write responsibility)
- + data ownership (2) – broader bounded context
- + eventual consistency (3)
- + data volume (3)
- + update rate (3)

data access

which one is better?

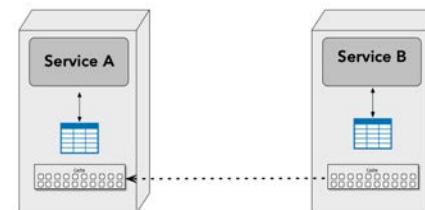
option 1:
interservice
communication



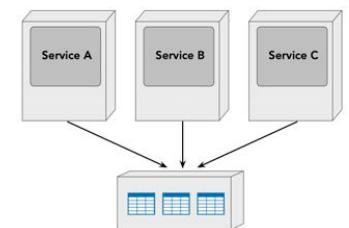
option 2:
data schema
replication



option 3:
in-memory
replicated cache



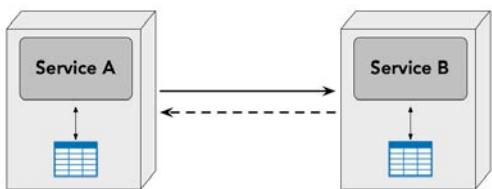
option 4:
data domains
(shared tables)



data access

which one is better?

option 1:
interservice
communication

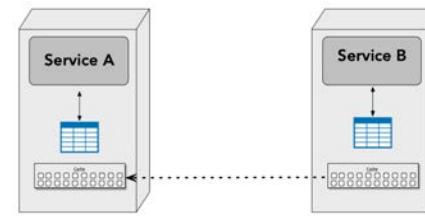


- ✓ large data volume
- ✓ low responsiveness

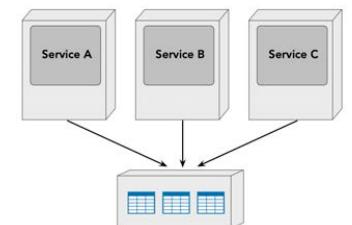
option 2:
data schema
replication



option 3:
in-memory
replicated cache



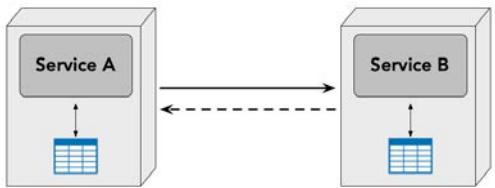
option 4:
data domains
(shared tables)



data access

which one is better?

option 1:
interservice
communication



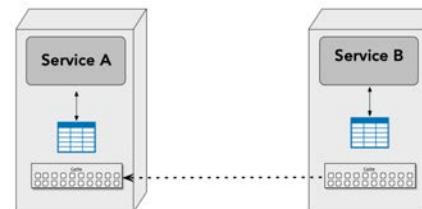
- ✓ large data volume
- ✓ low responsiveness

option 2:
data schema
replication

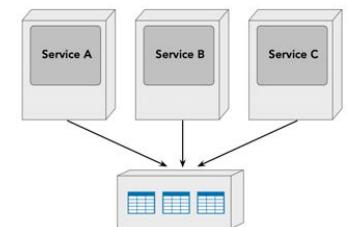


- ✓ reporting
- ✓ data aggregation

option 3:
in-memory
replicated cache



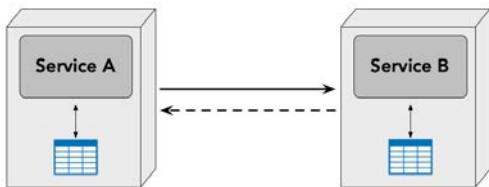
option 4:
data domains
(shared tables)



data access

which one is better?

option 1:
interservice
communication



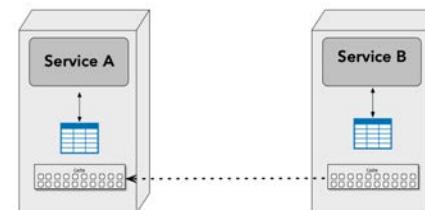
- ✓ large data volume
- ✓ low responsiveness

option 2:
data schema
replication



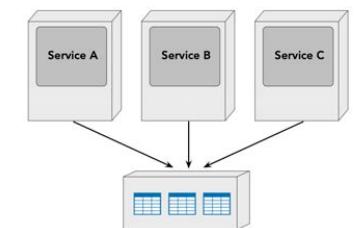
- ✓ reporting
- ✓ data aggregation

option 3:
in-memory
replicated cache



- ✓ low data volume
- ✓ high responsiveness

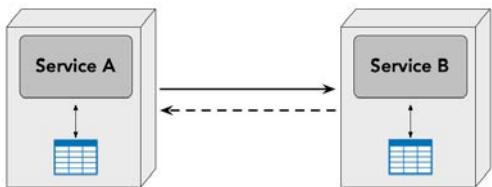
option 4:
data domains
(shared tables)



data access

which one is better?

option 1:
interservice
communication



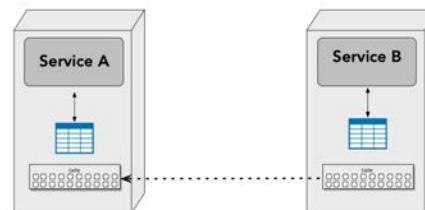
- ✓ large data volume
- ✓ low responsiveness

option 2:
data schema
replication



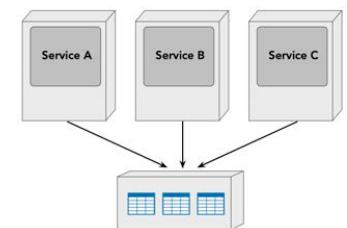
- ✓ reporting
- ✓ data aggregation

option 3:
in-memory
replicated cache



- ✓ low data volume
- ✓ high responsiveness

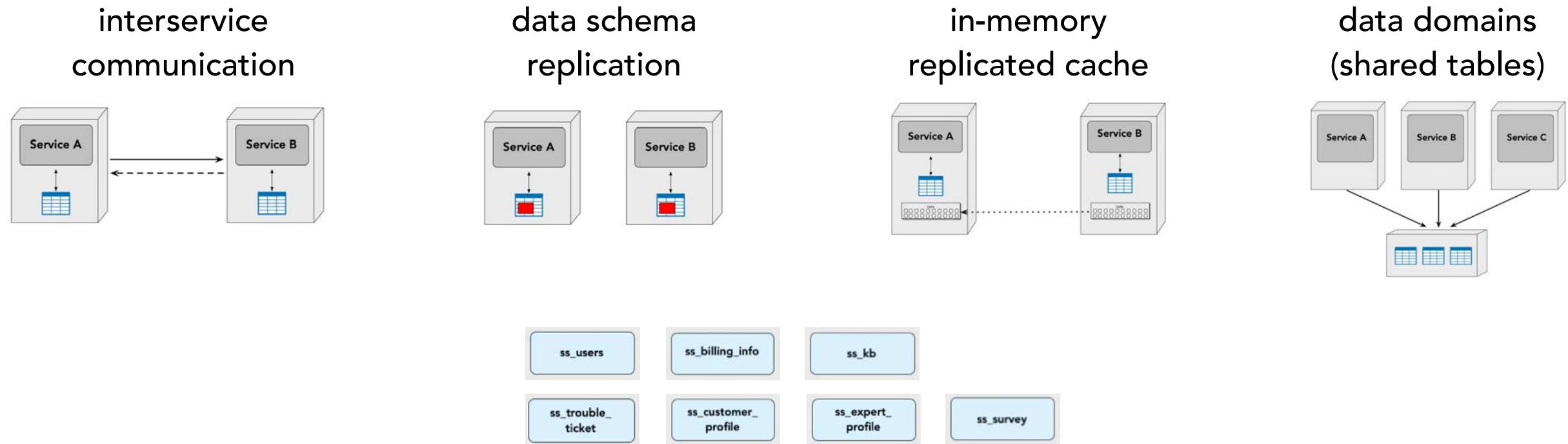
option 4:
data domains
(shared tables)



- ✓ multiple owners (write)
- ✓ high responsiveness

Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices for data access are indicated below. Don't forget you can use any combination of these depending on the data.

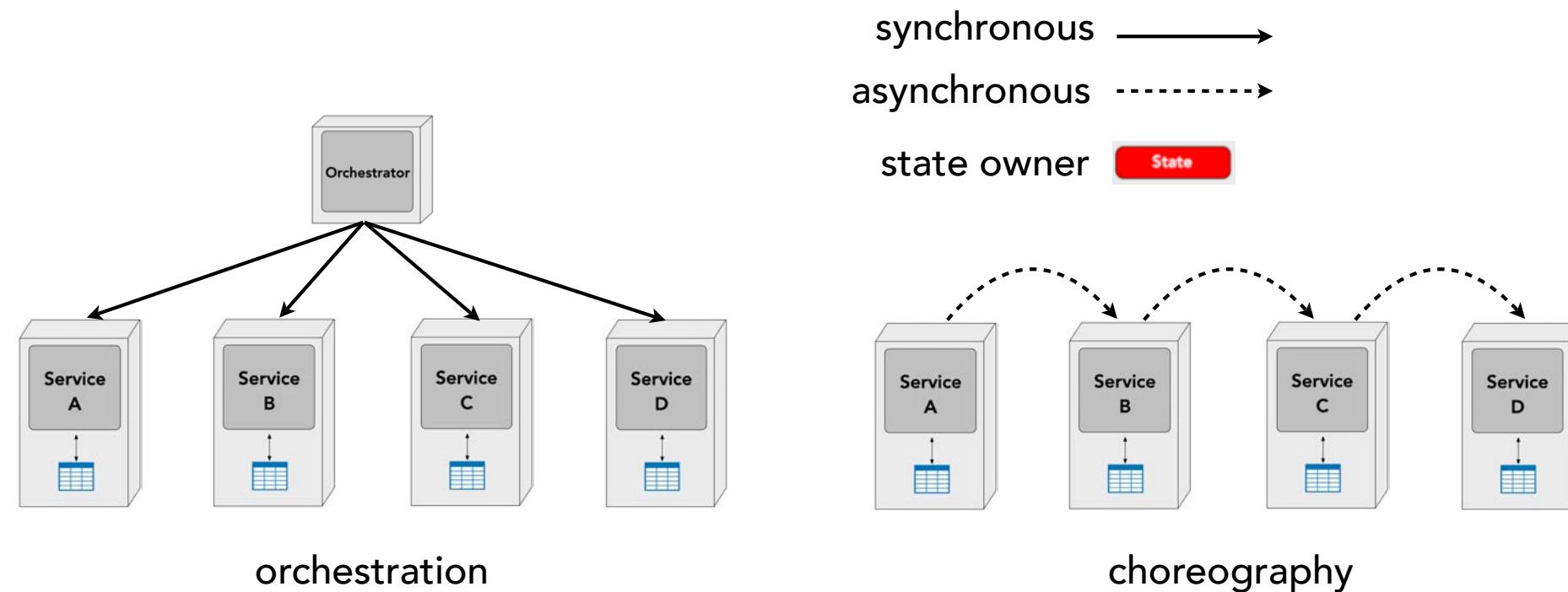




Orchestration & Workflow

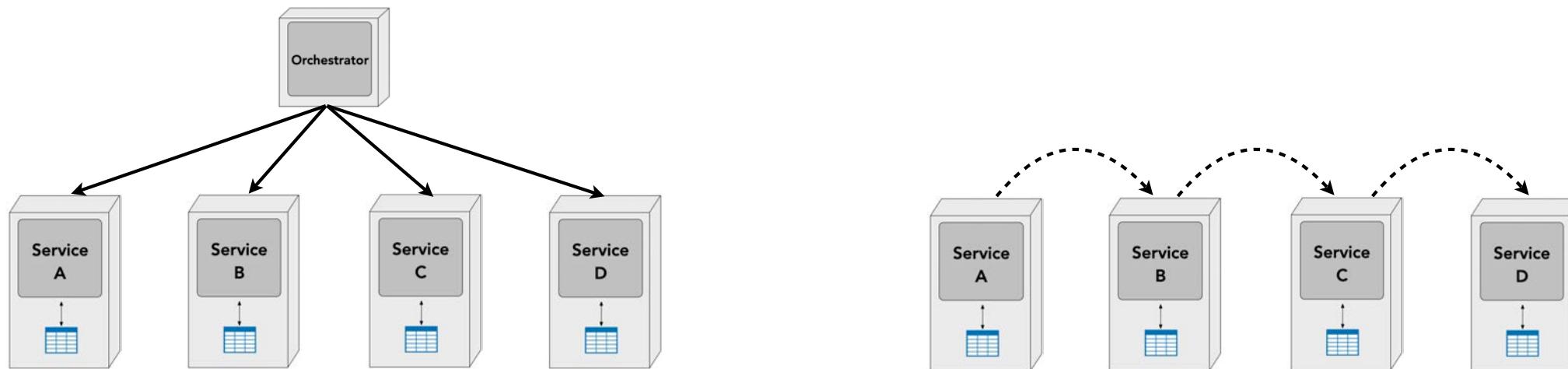
Kata Exercise - Workflow and Communications

Now that you broke things apart, you now need to stitch them back together. Based on the workflows in the system, how should the services communicate with each other? Asynchronous or synchronous? Orchestration or choreography?

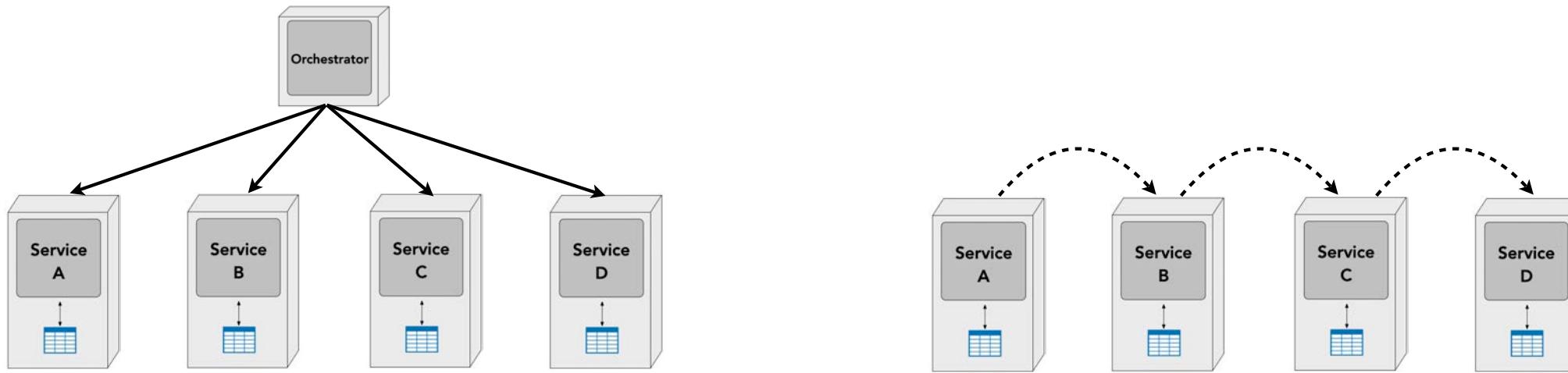


orchestration and workflow

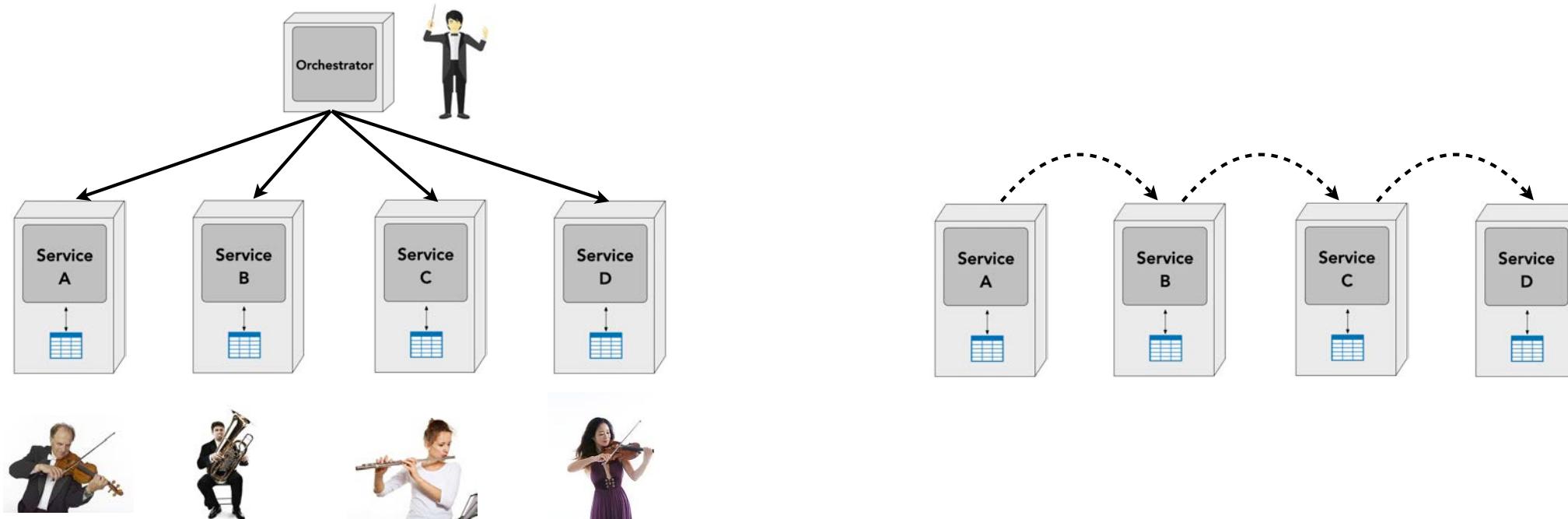
“when should I consider using orchestration vs. choreography?”



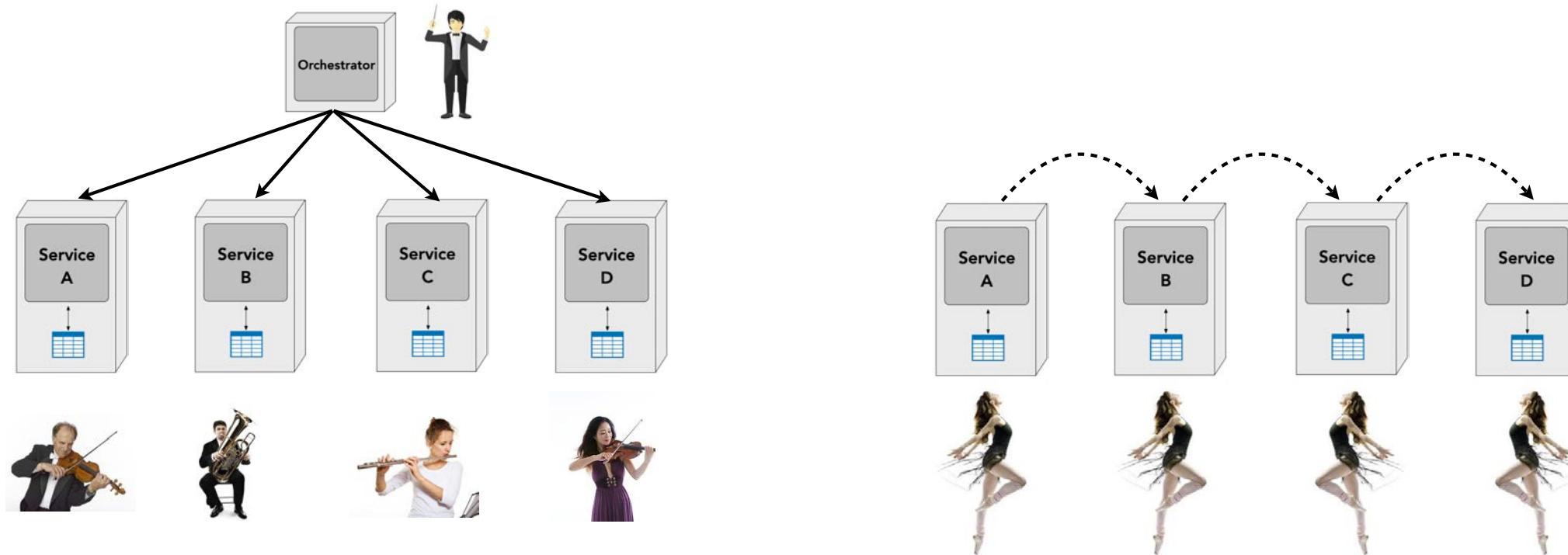
orchestration and choreography



orchestration and choreography

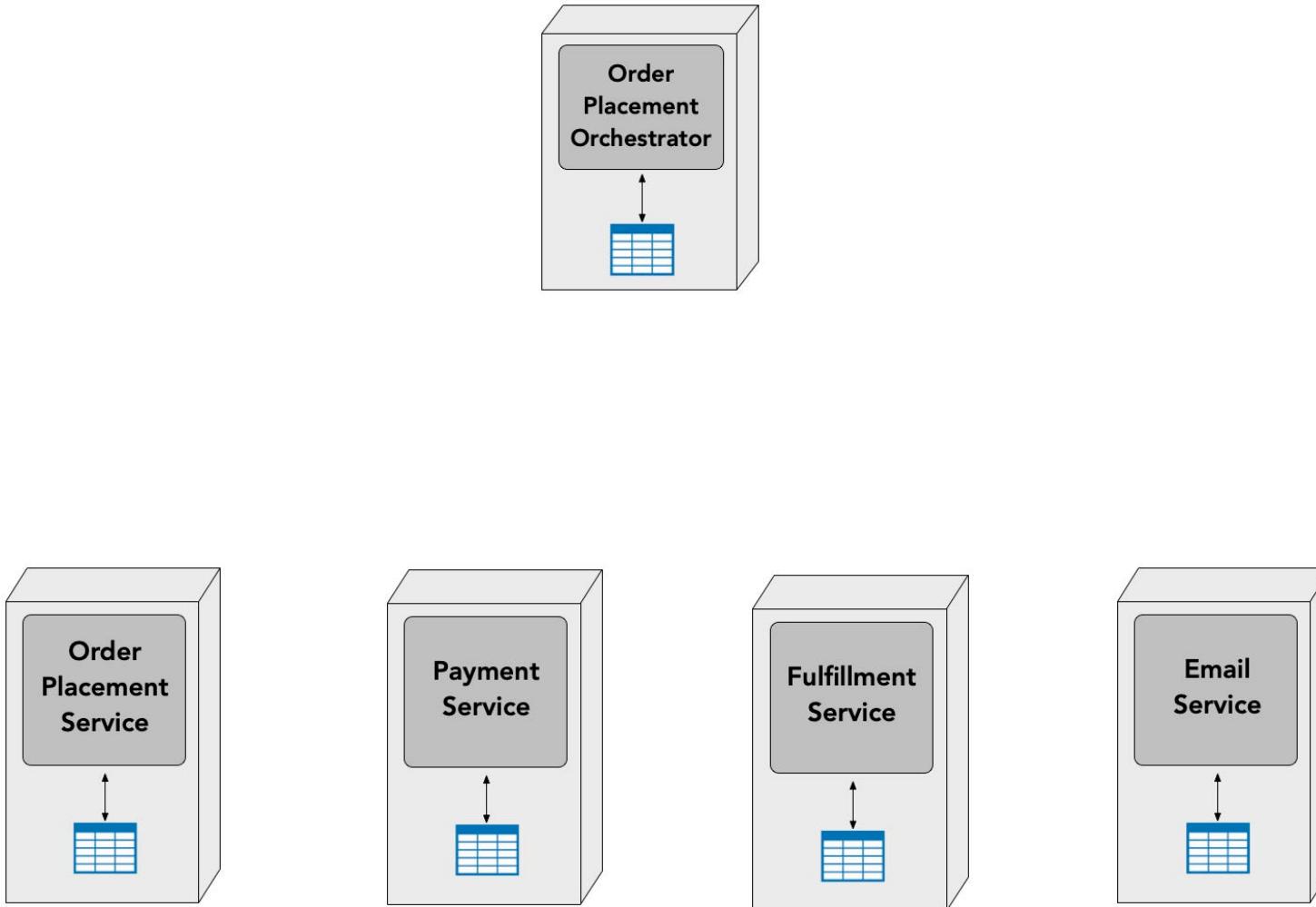


orchestration and choreography

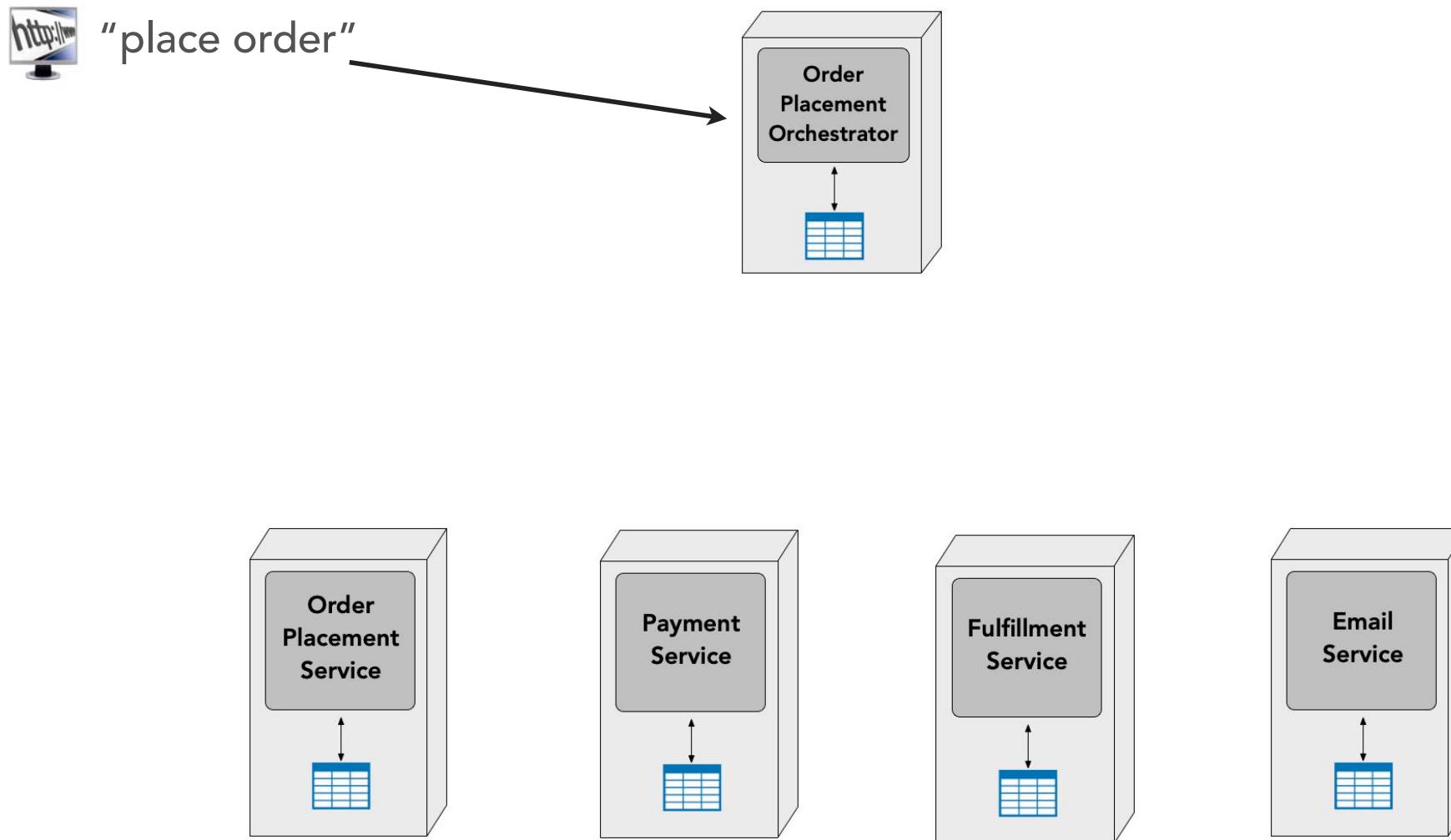


managing workflows

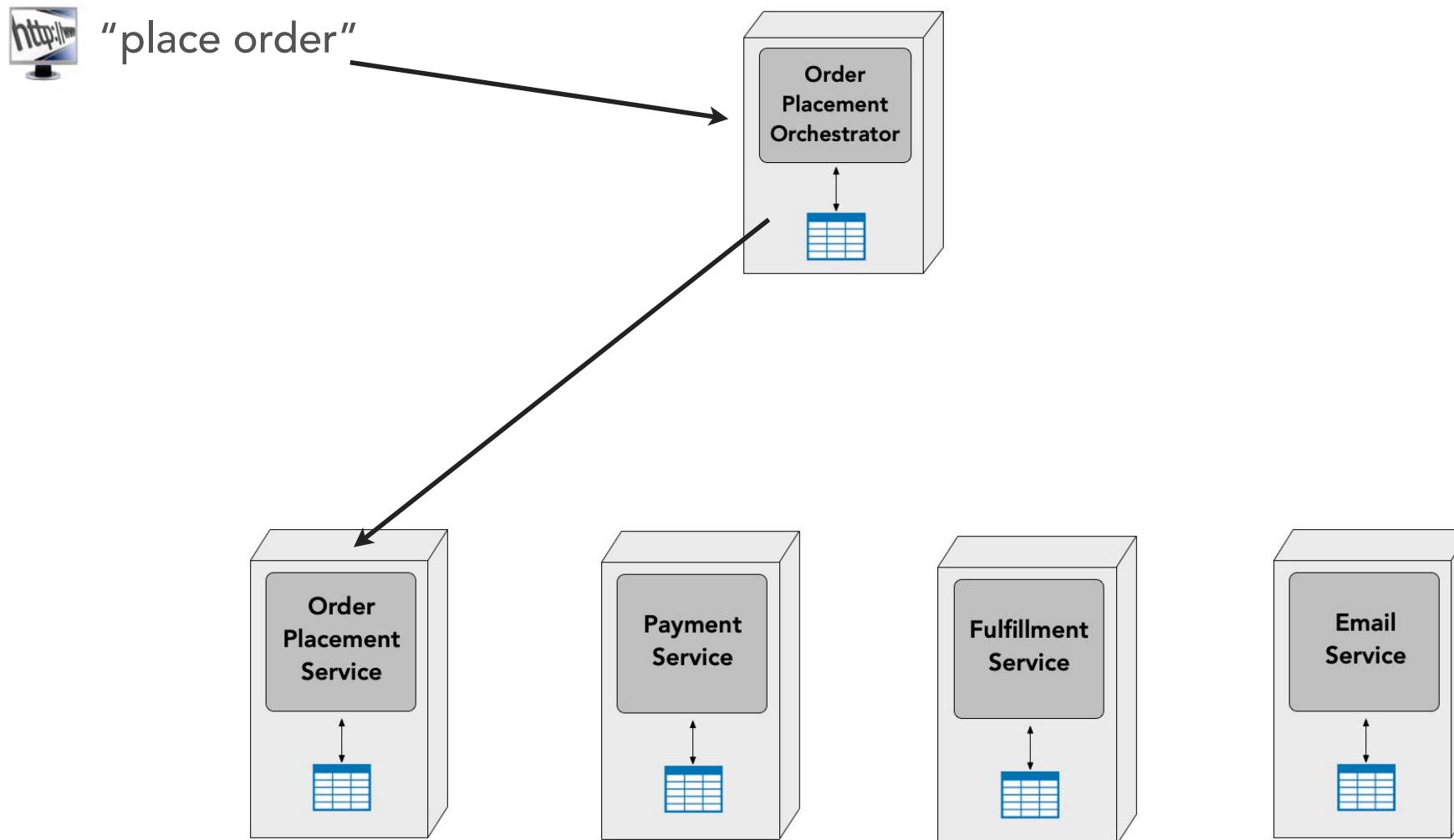
managing workflows



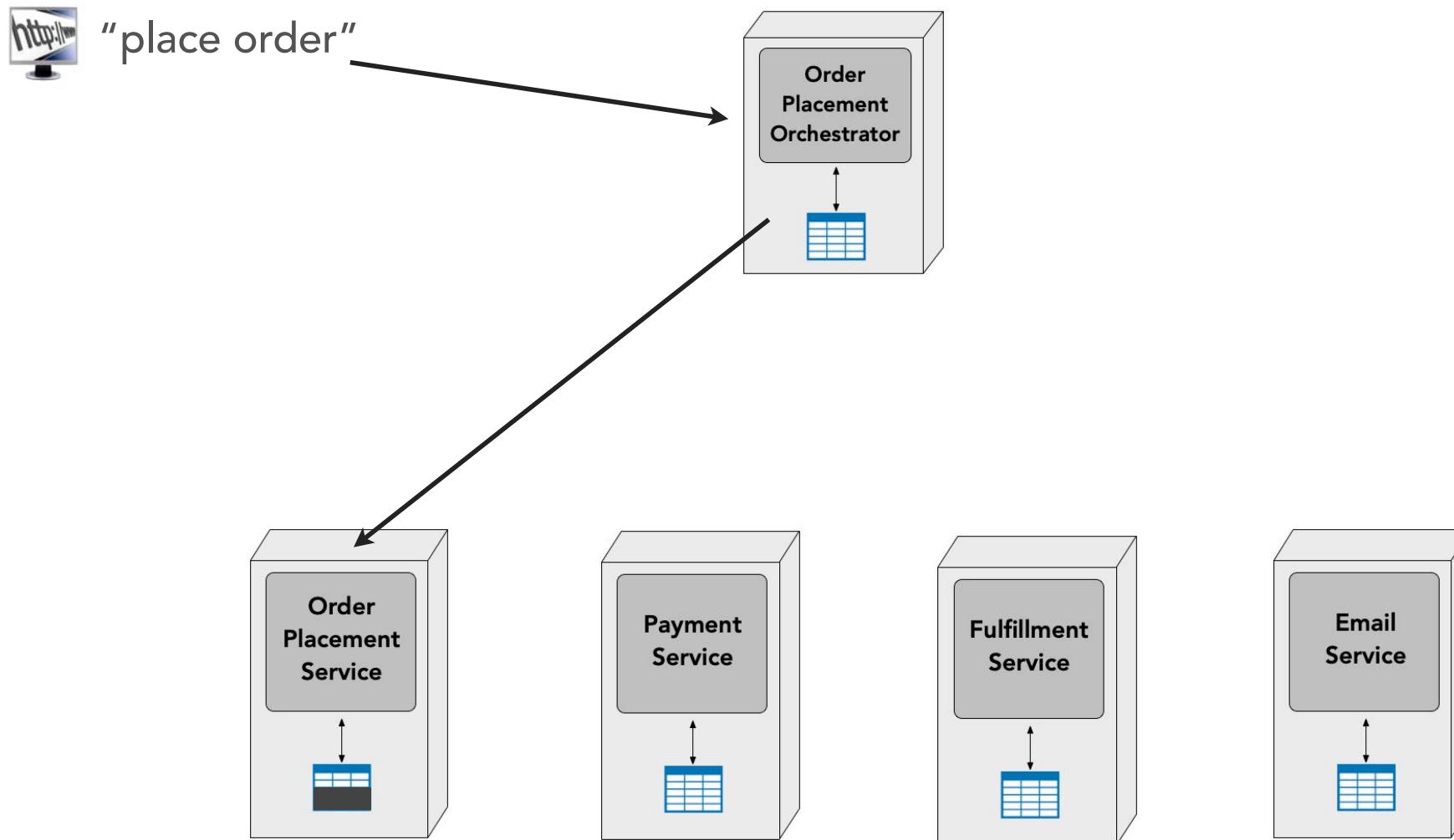
managing workflows



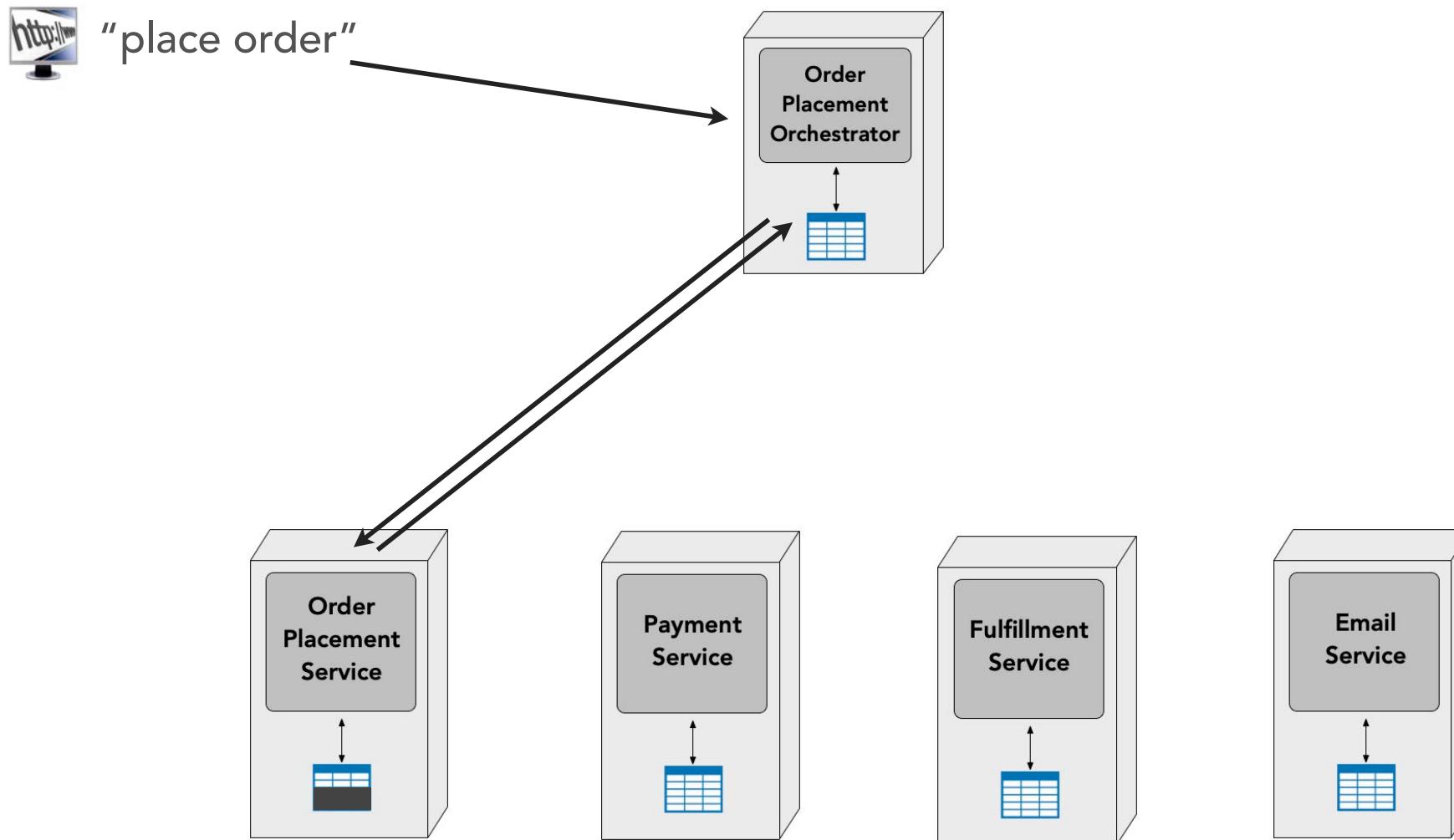
managing workflows



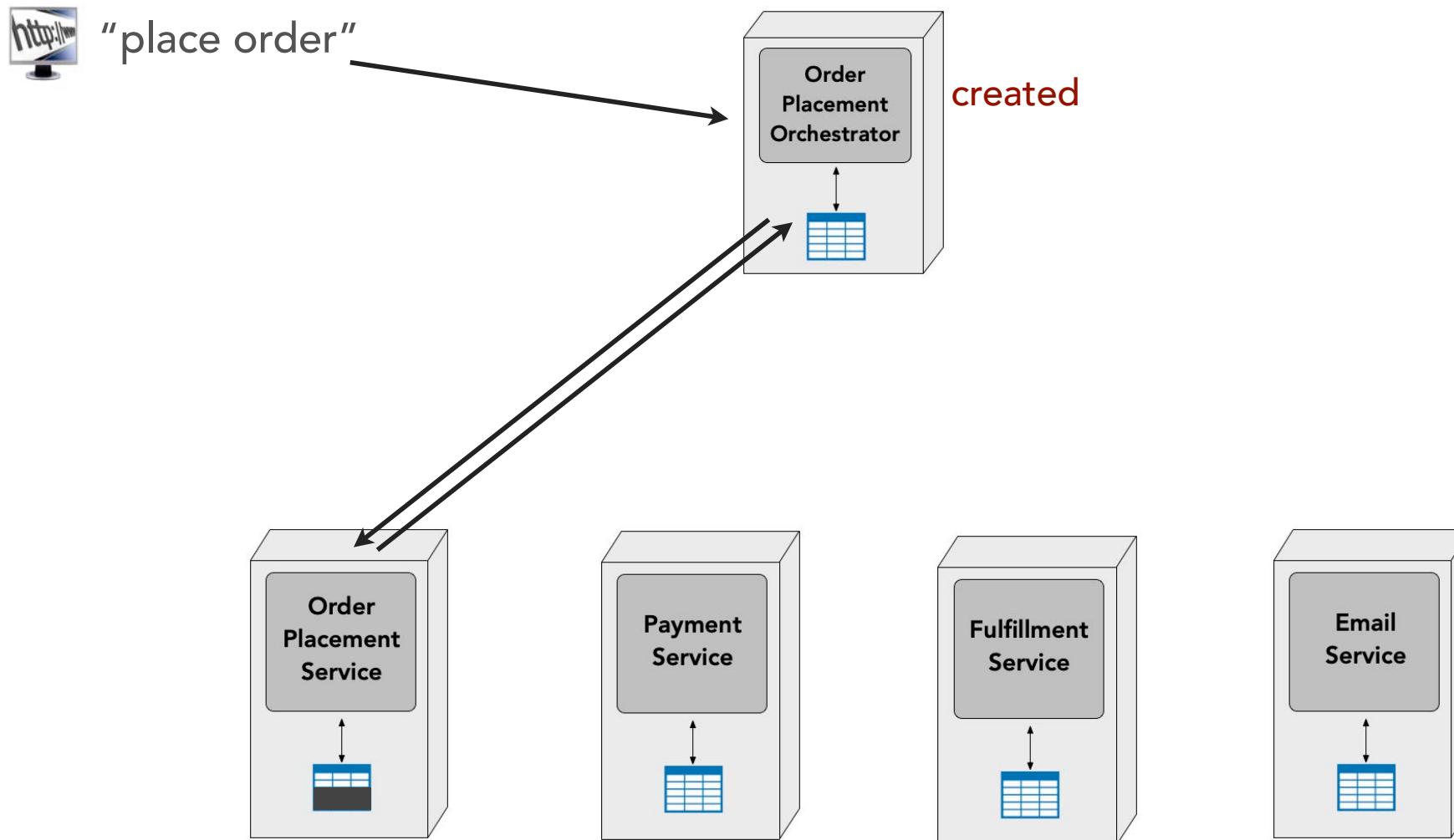
managing workflows



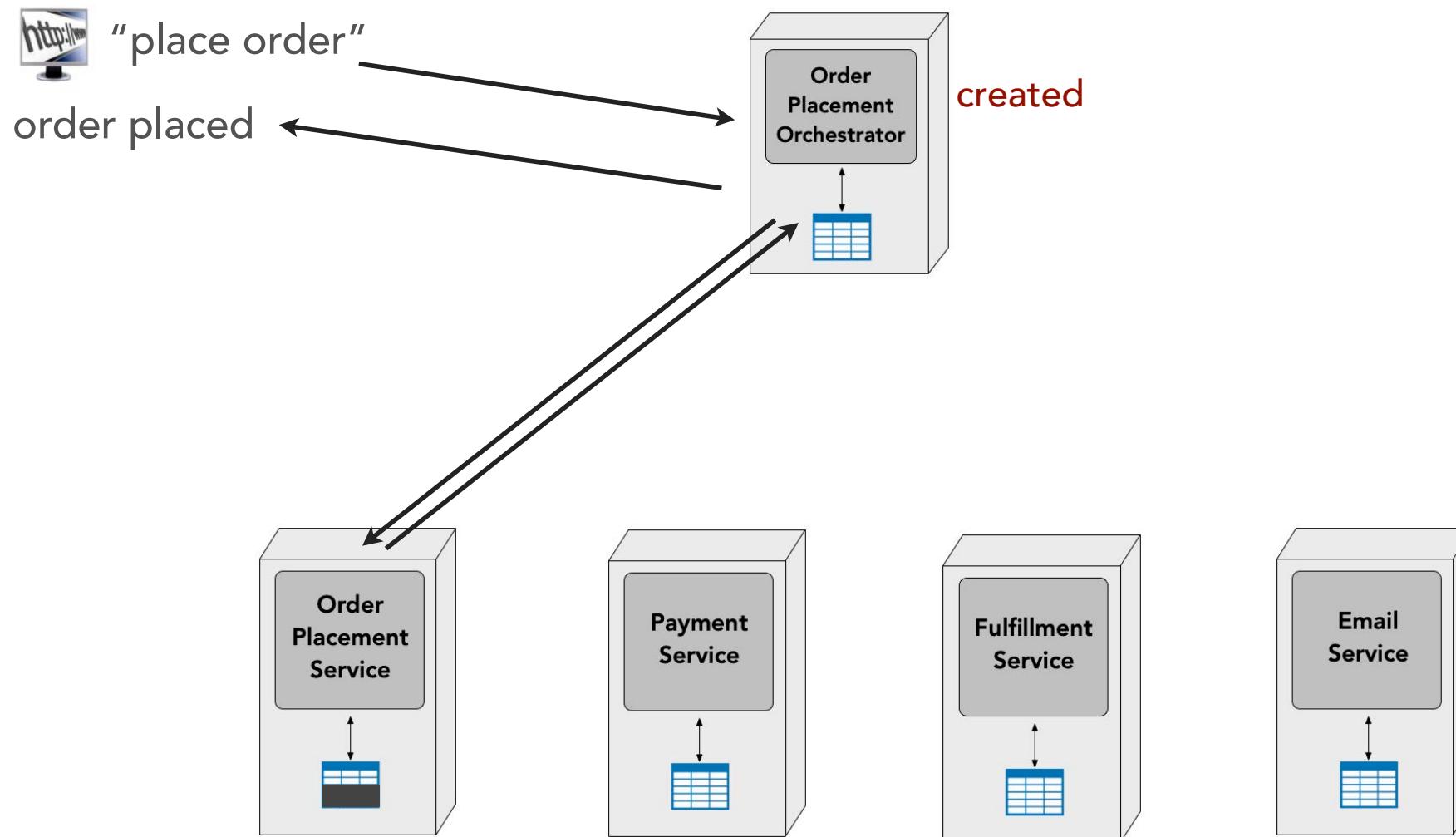
managing workflows



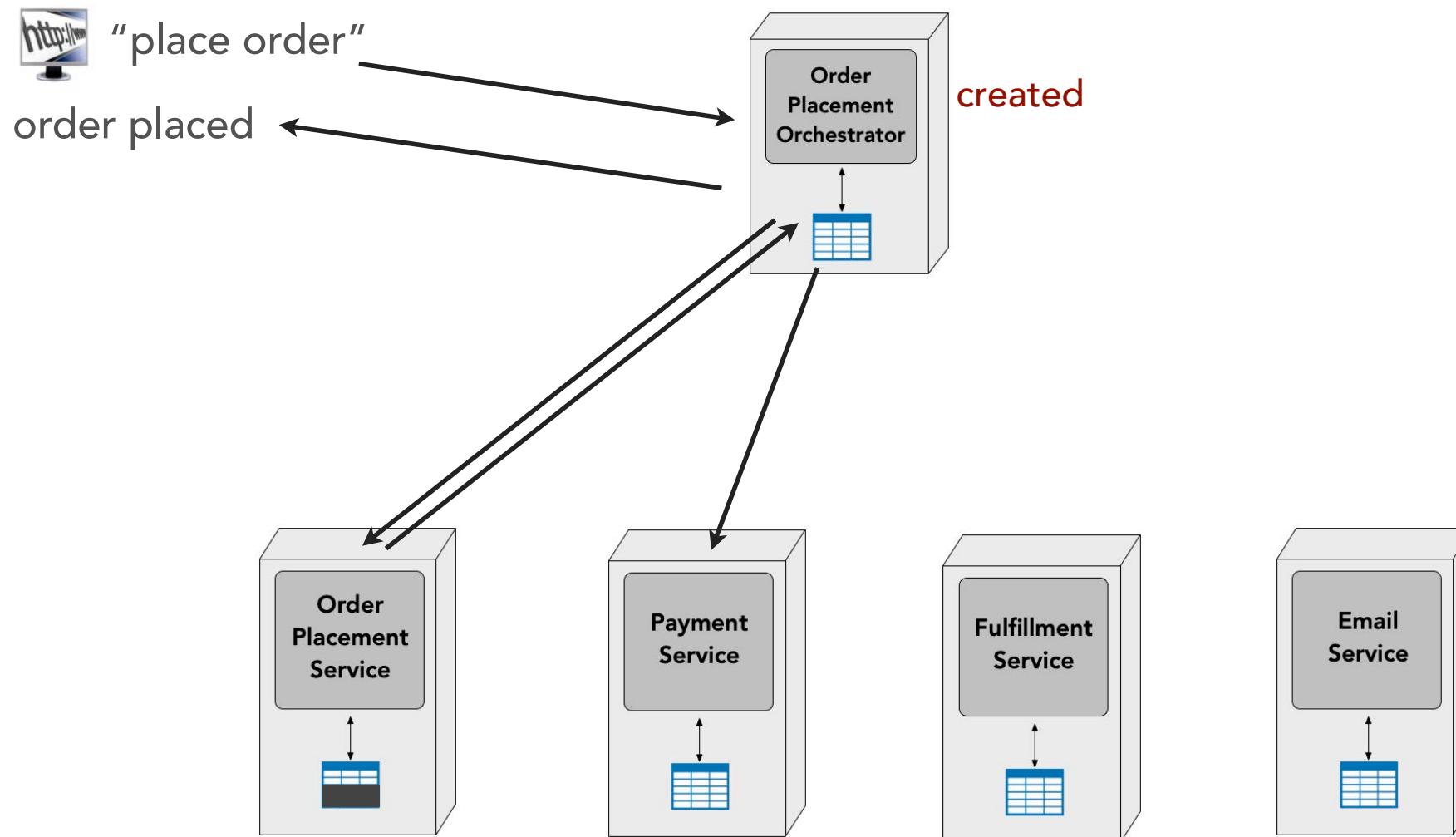
managing workflows



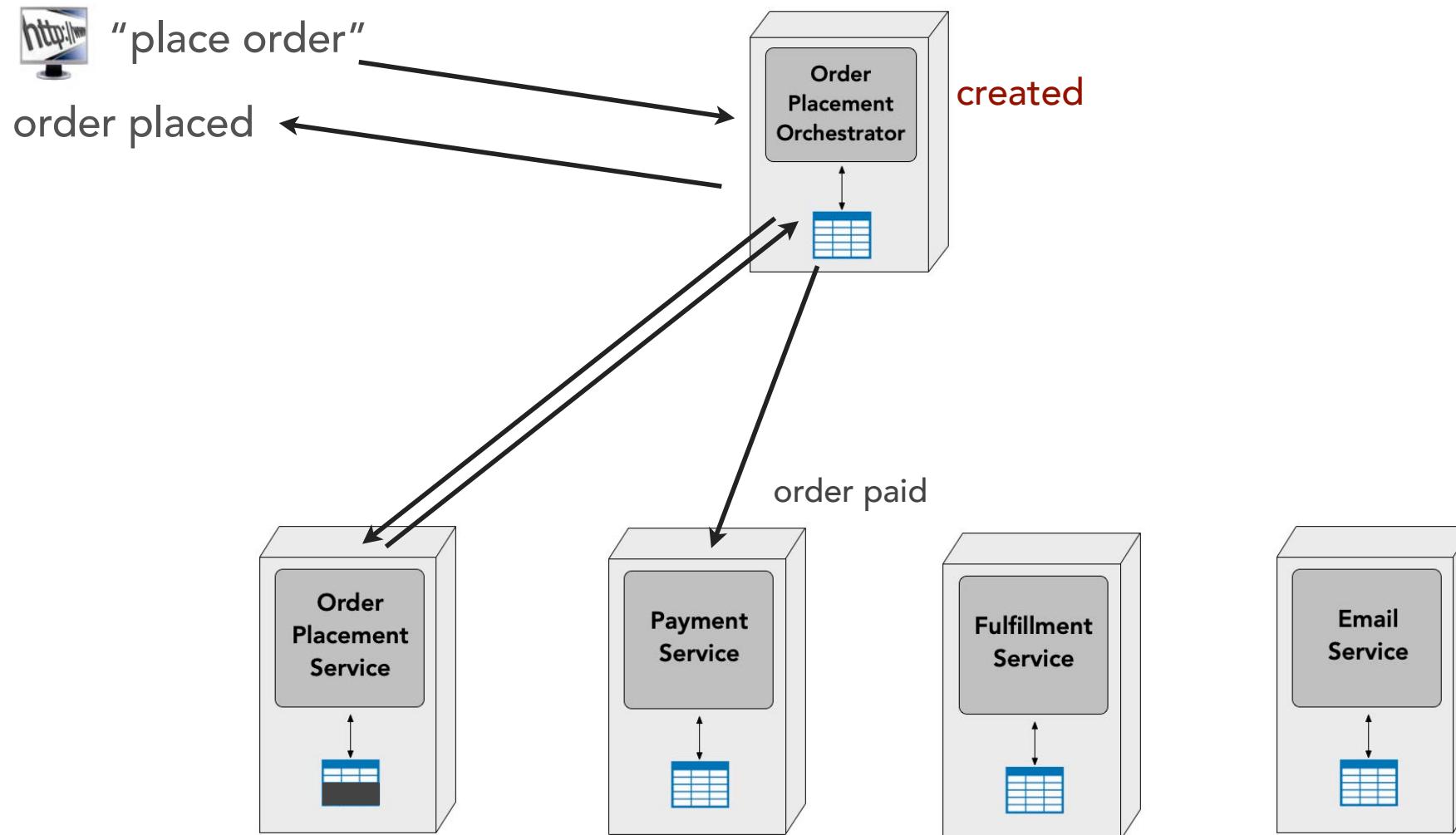
managing workflows



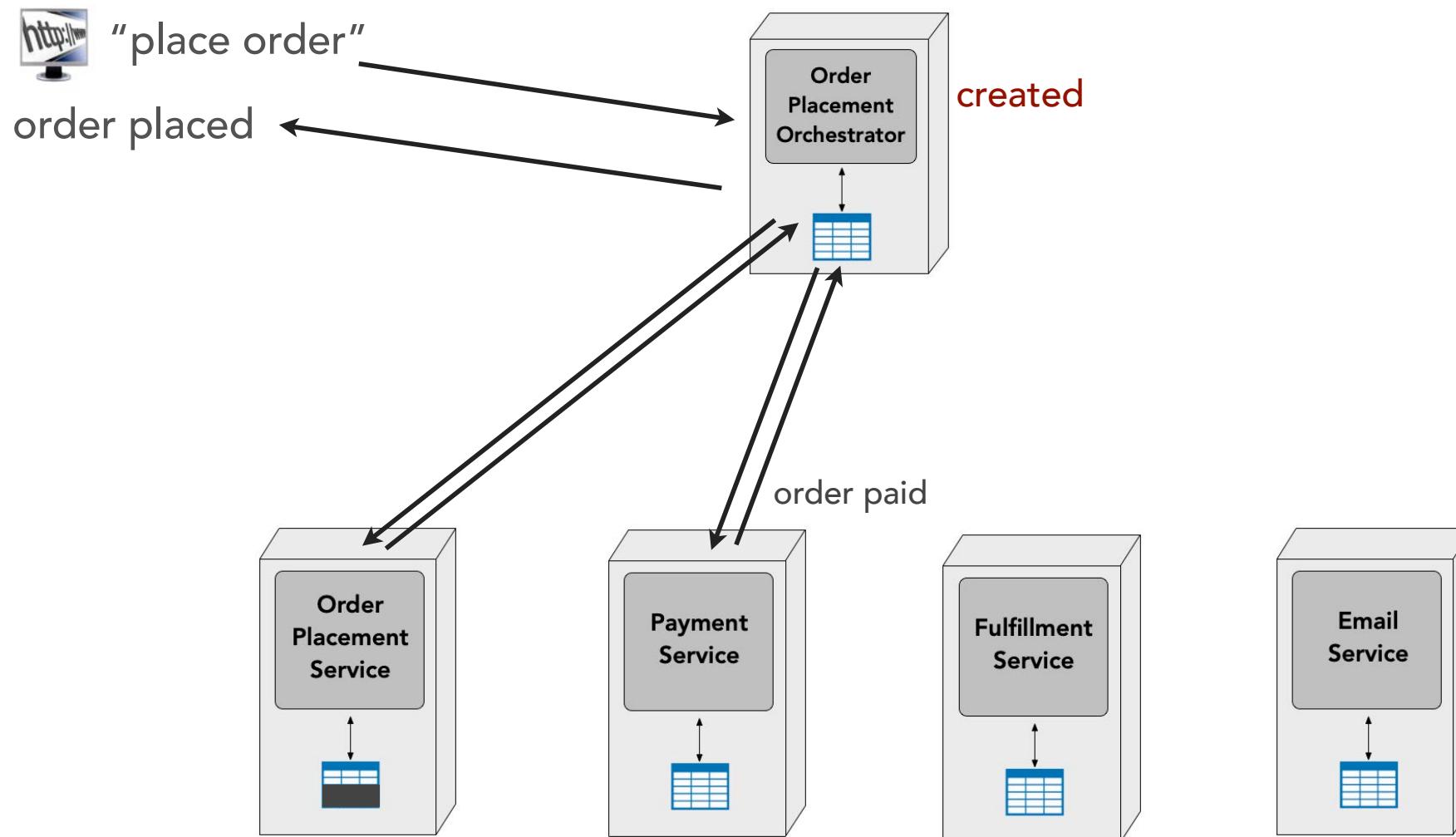
managing workflows



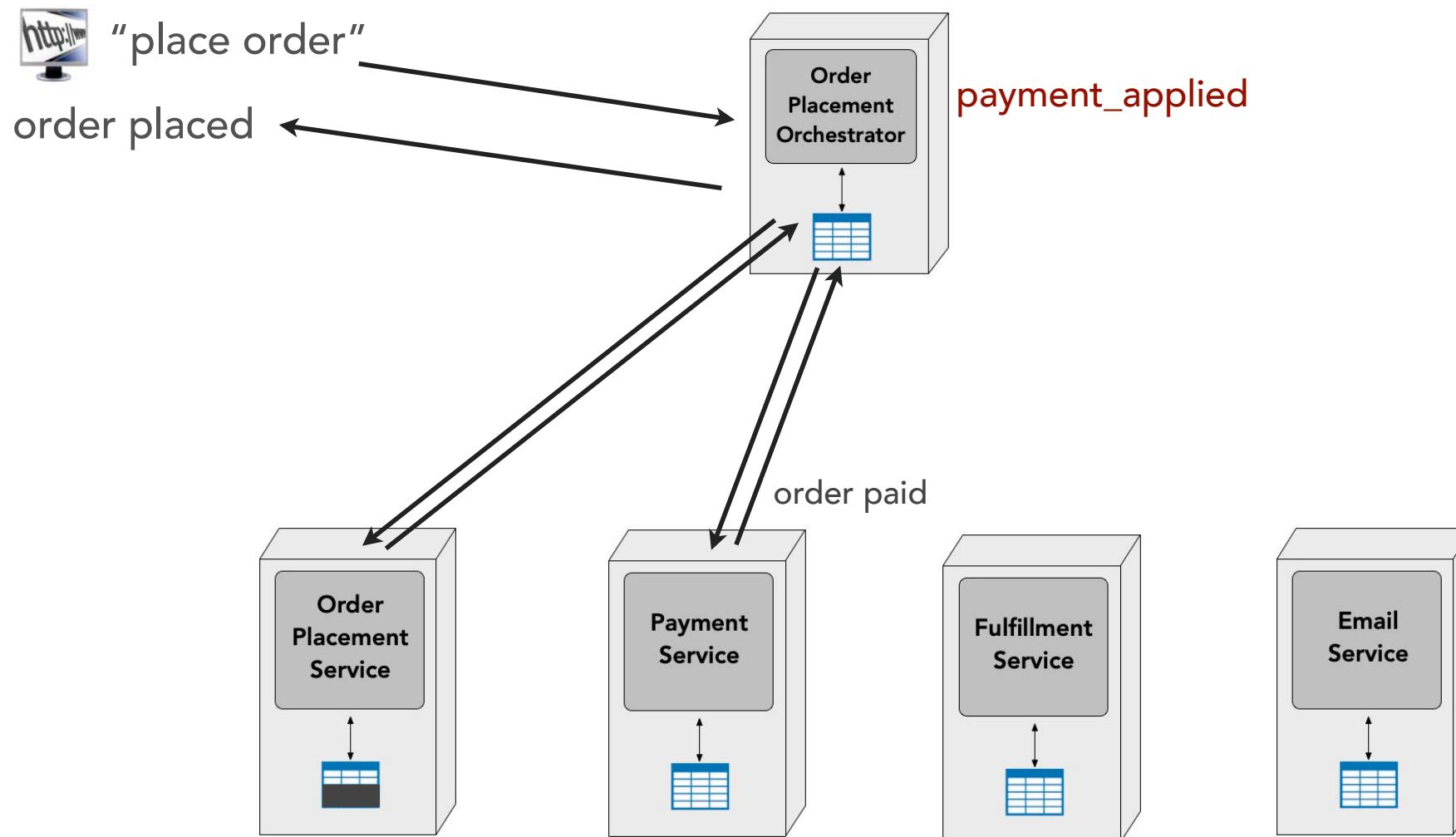
managing workflows



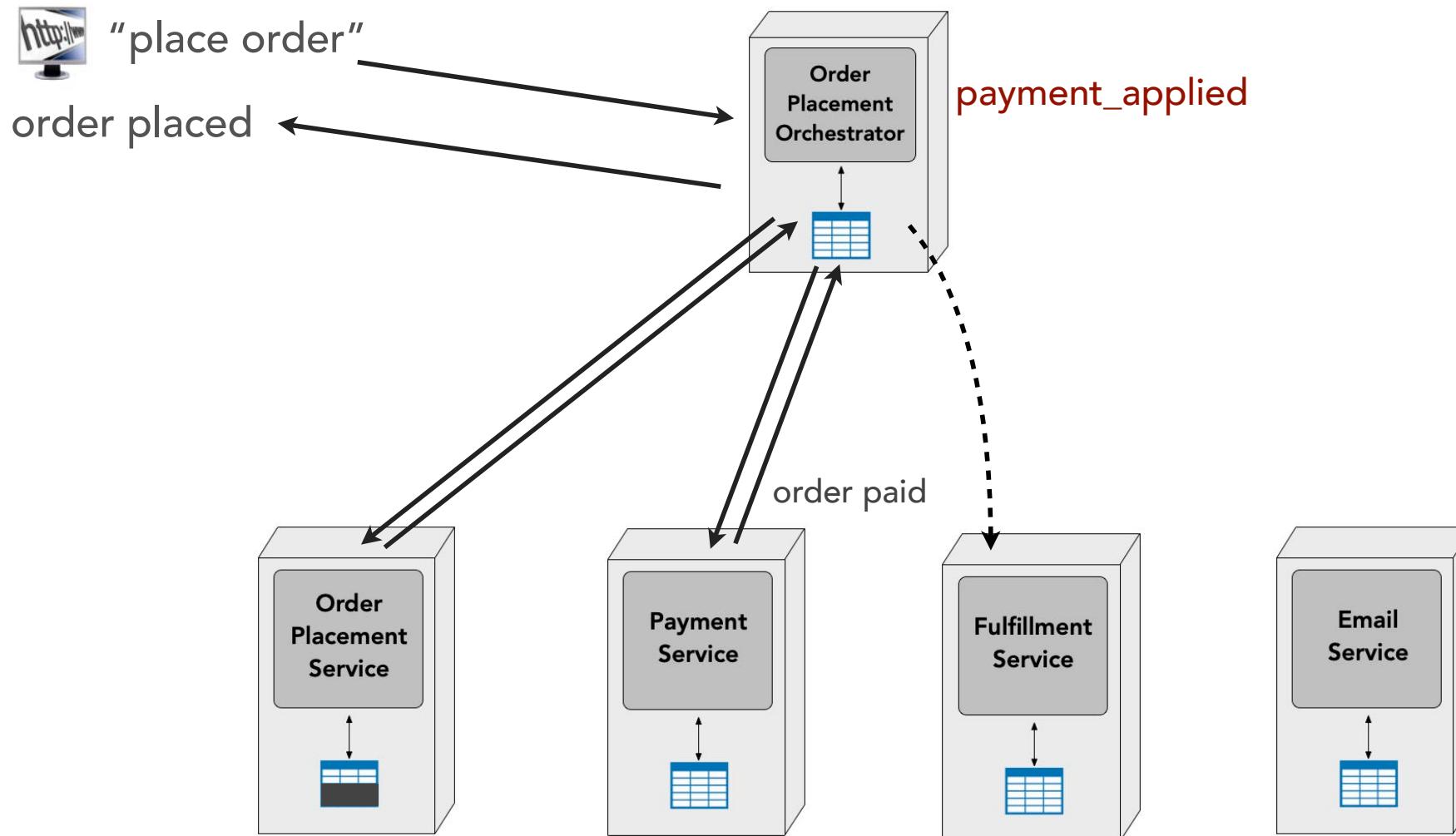
managing workflows



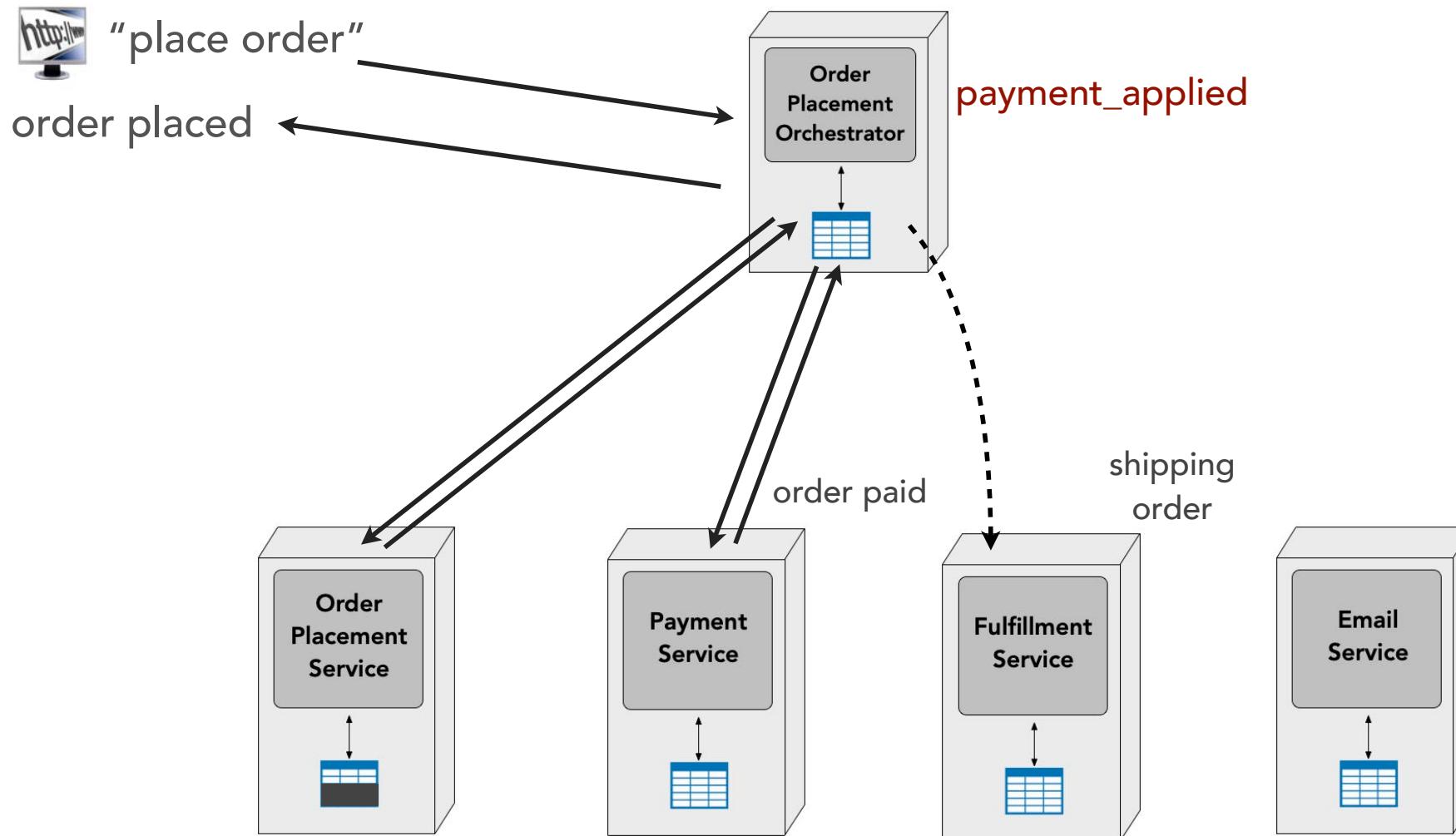
managing workflows



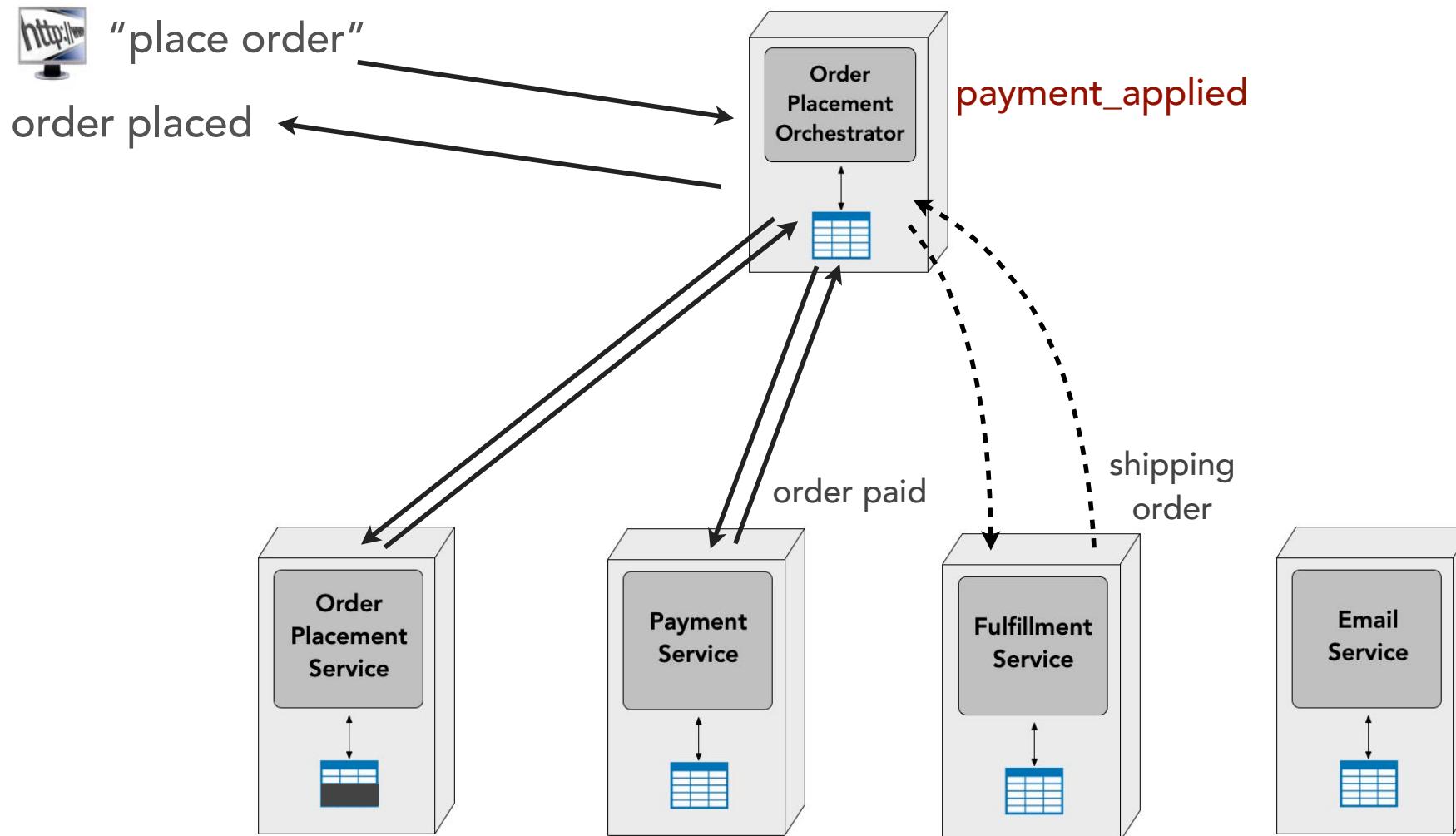
managing workflows



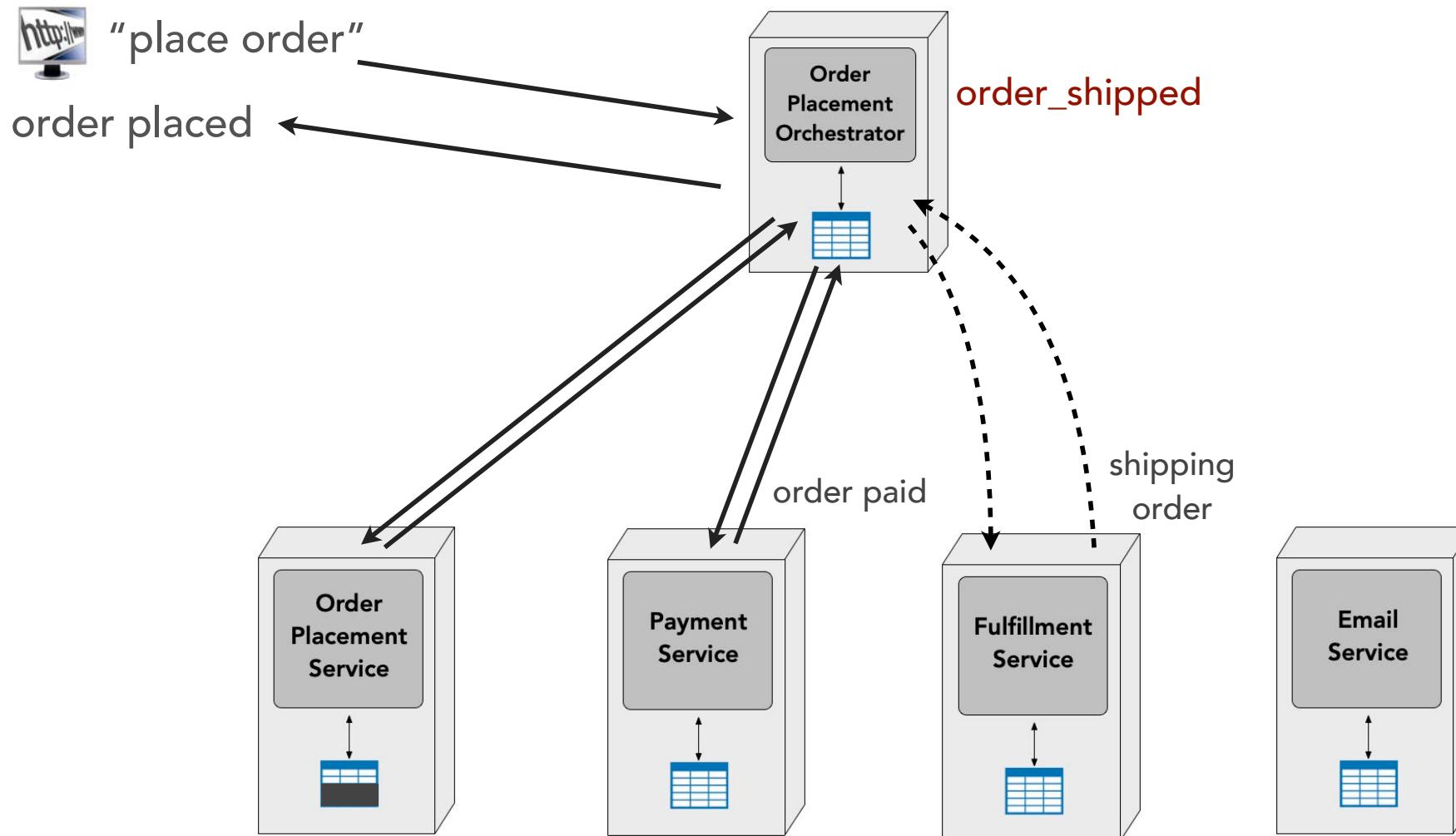
managing workflows



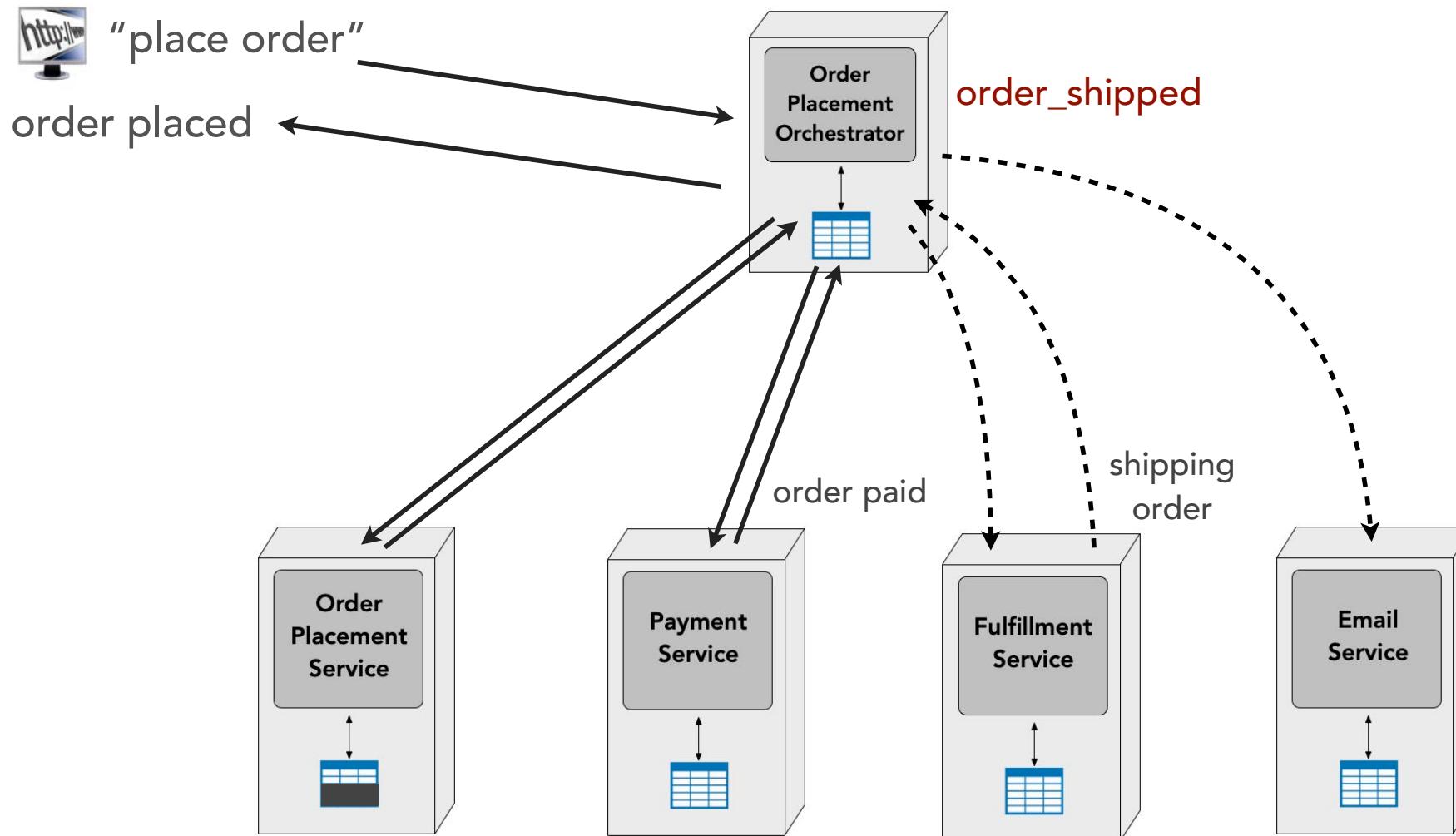
managing workflows



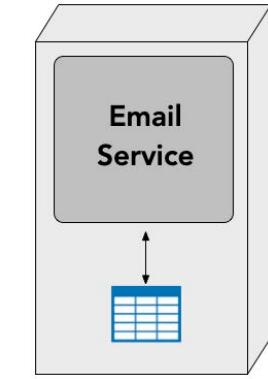
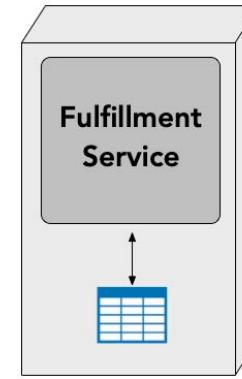
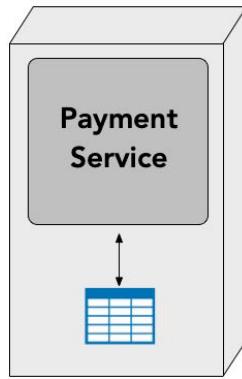
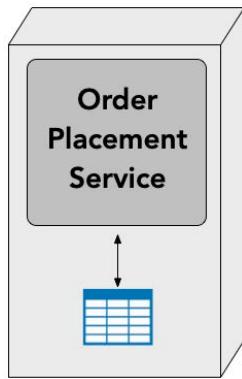
managing workflows



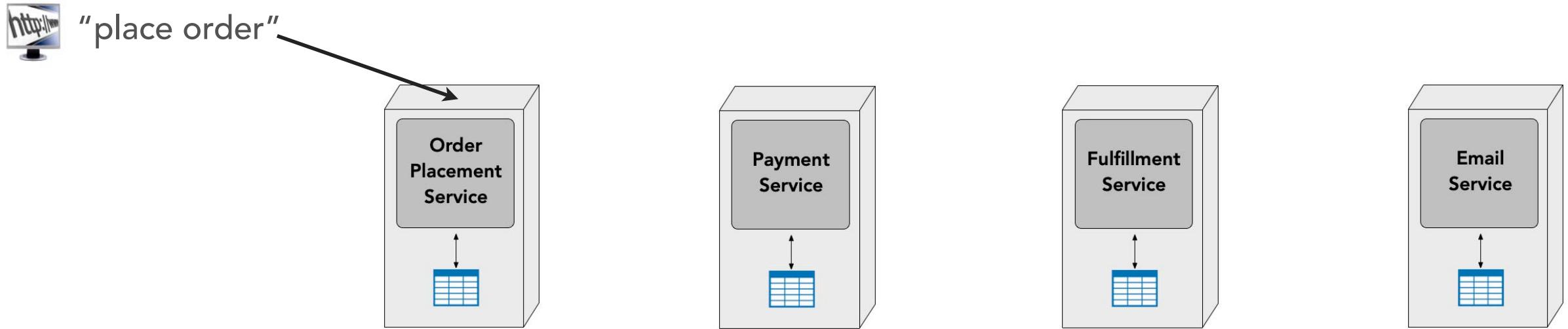
managing workflows



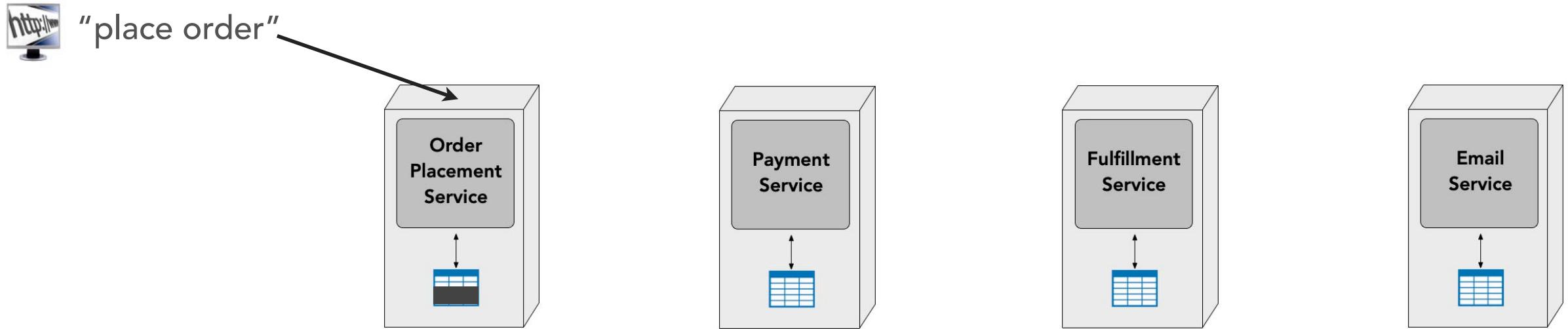
managing workflows



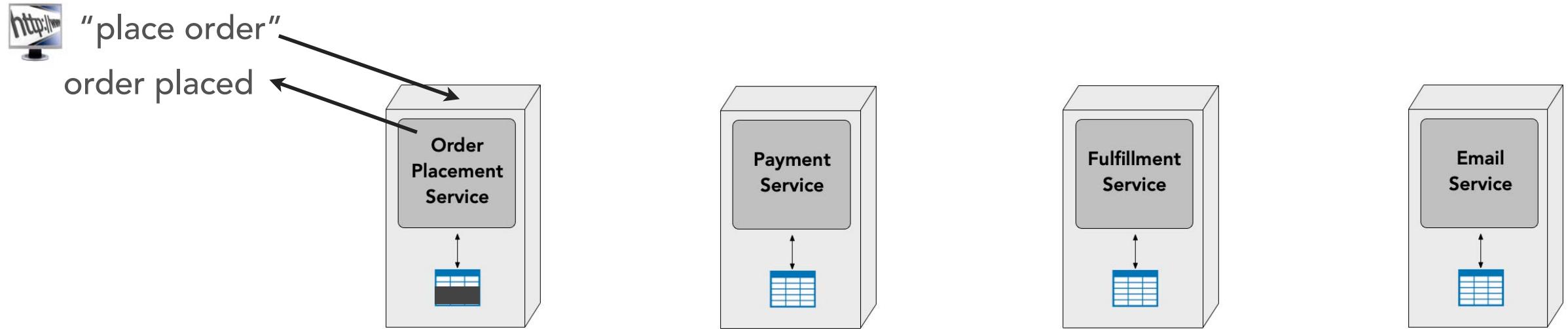
managing workflows



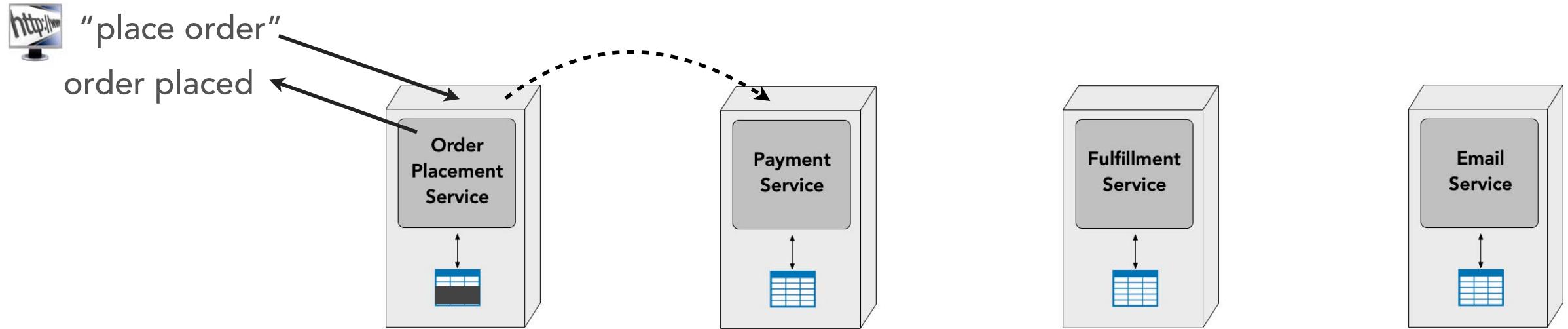
managing workflows



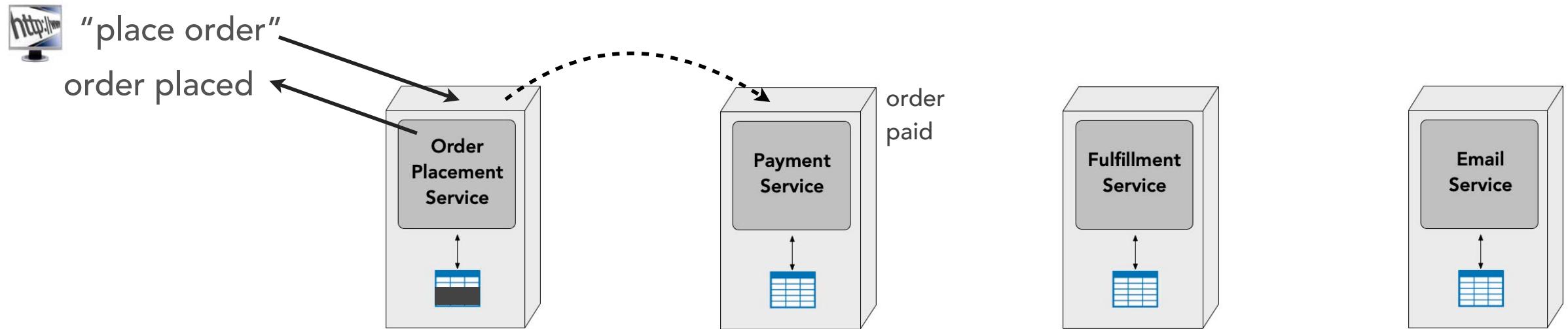
managing workflows



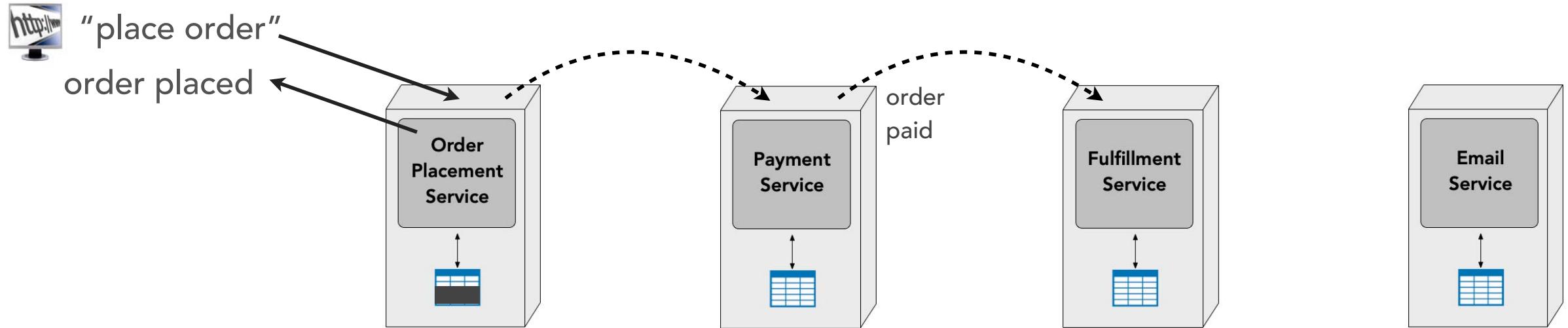
managing workflows



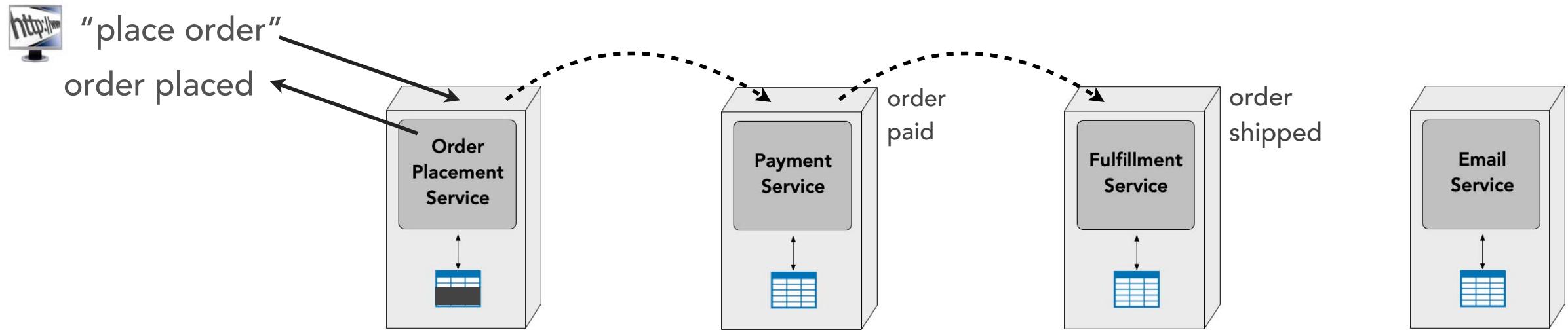
managing workflows



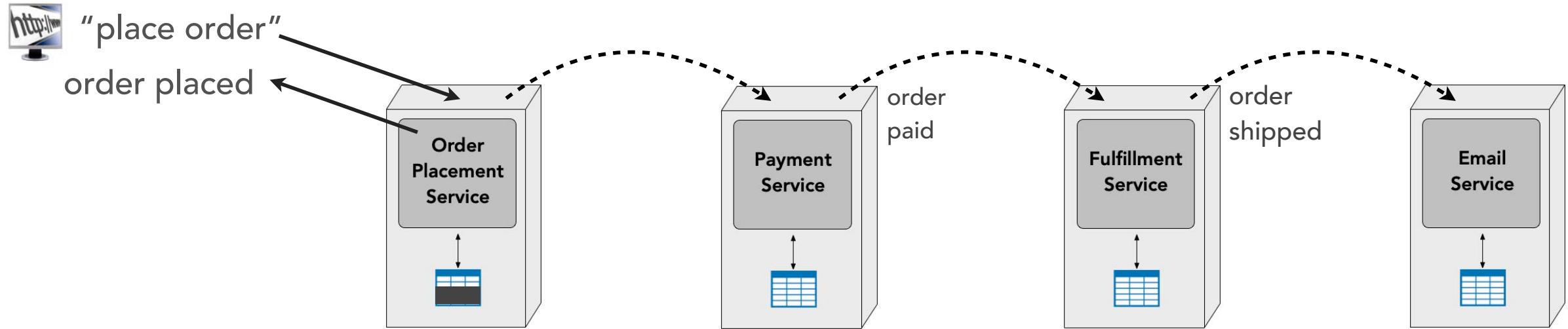
managing workflows



managing workflows

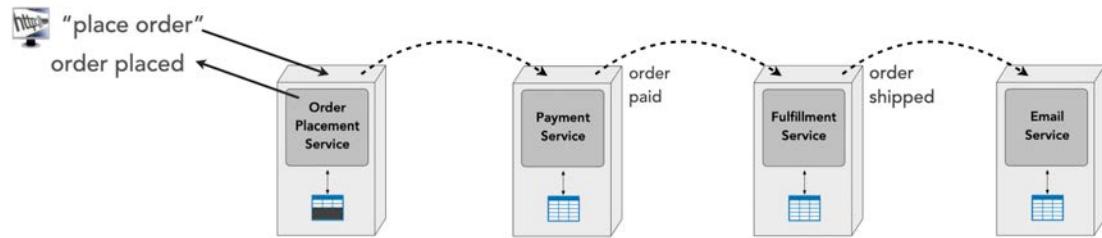


managing workflows

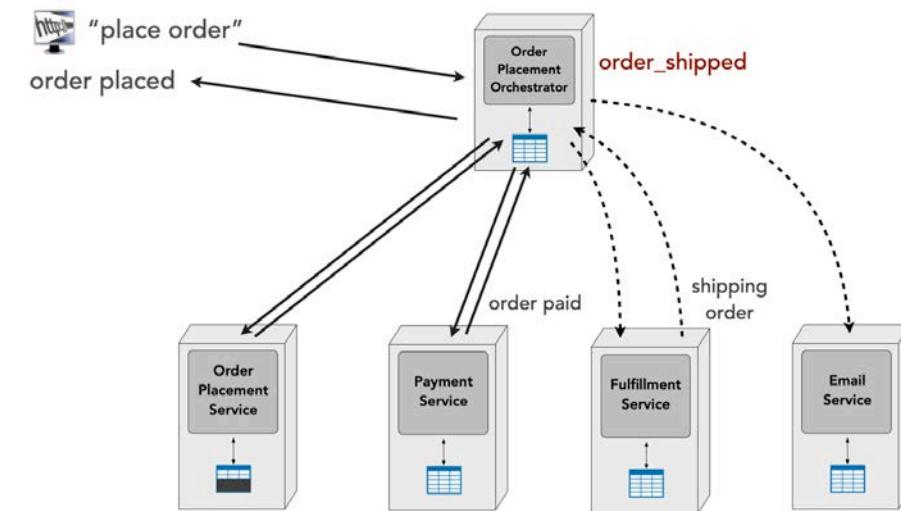


managing workflows

choreography

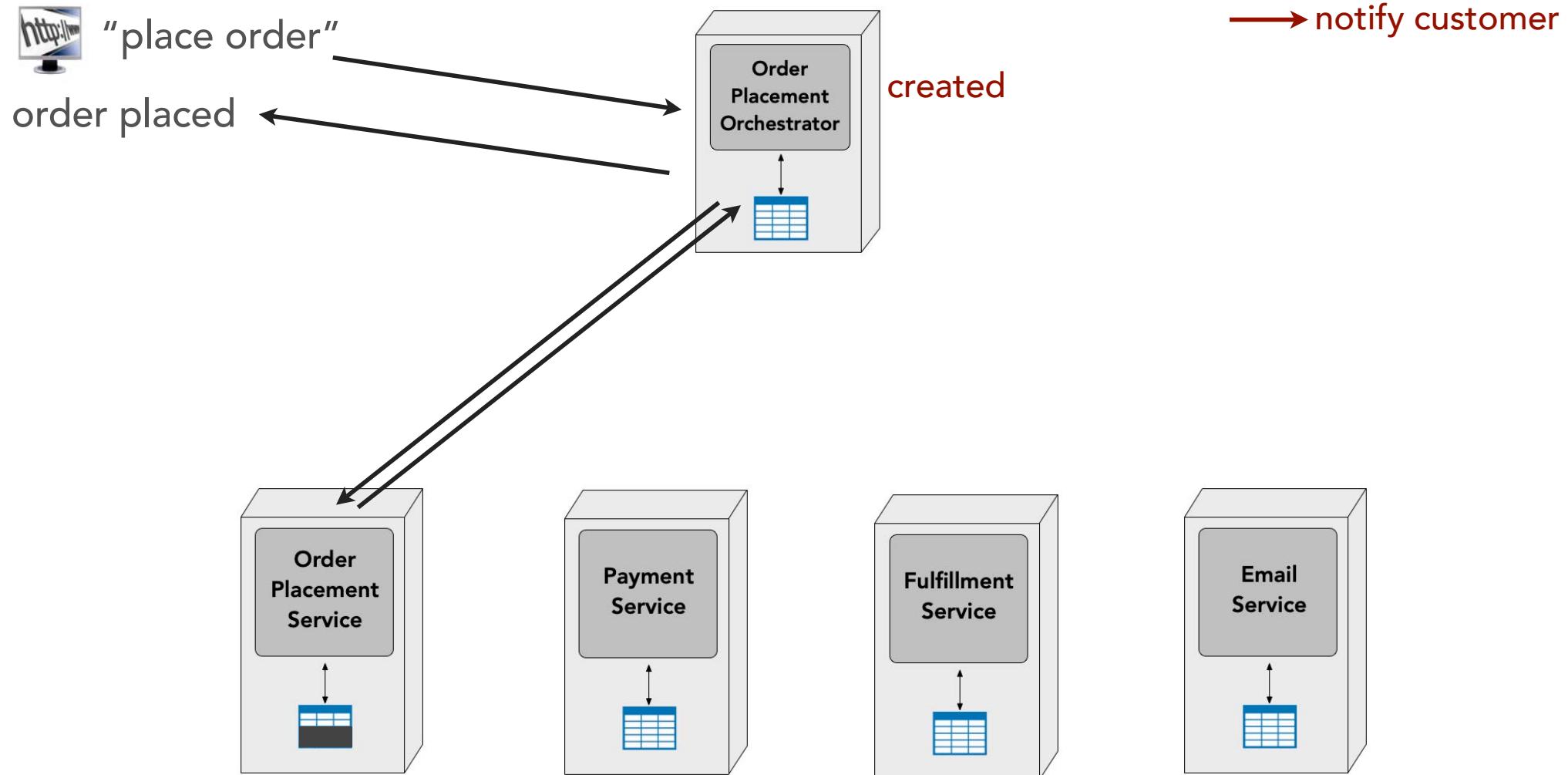


orchestration



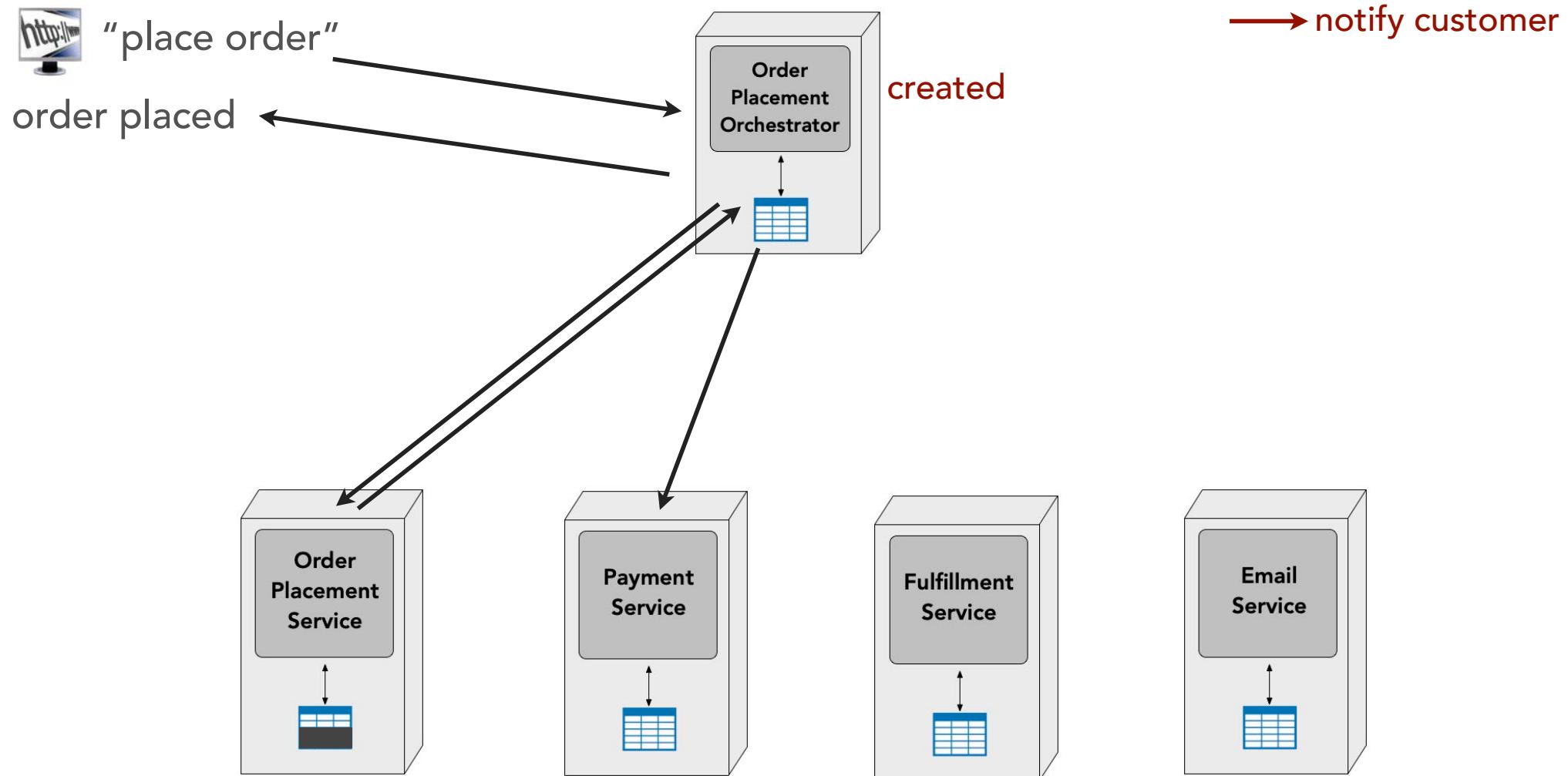
managing workflows

error handling



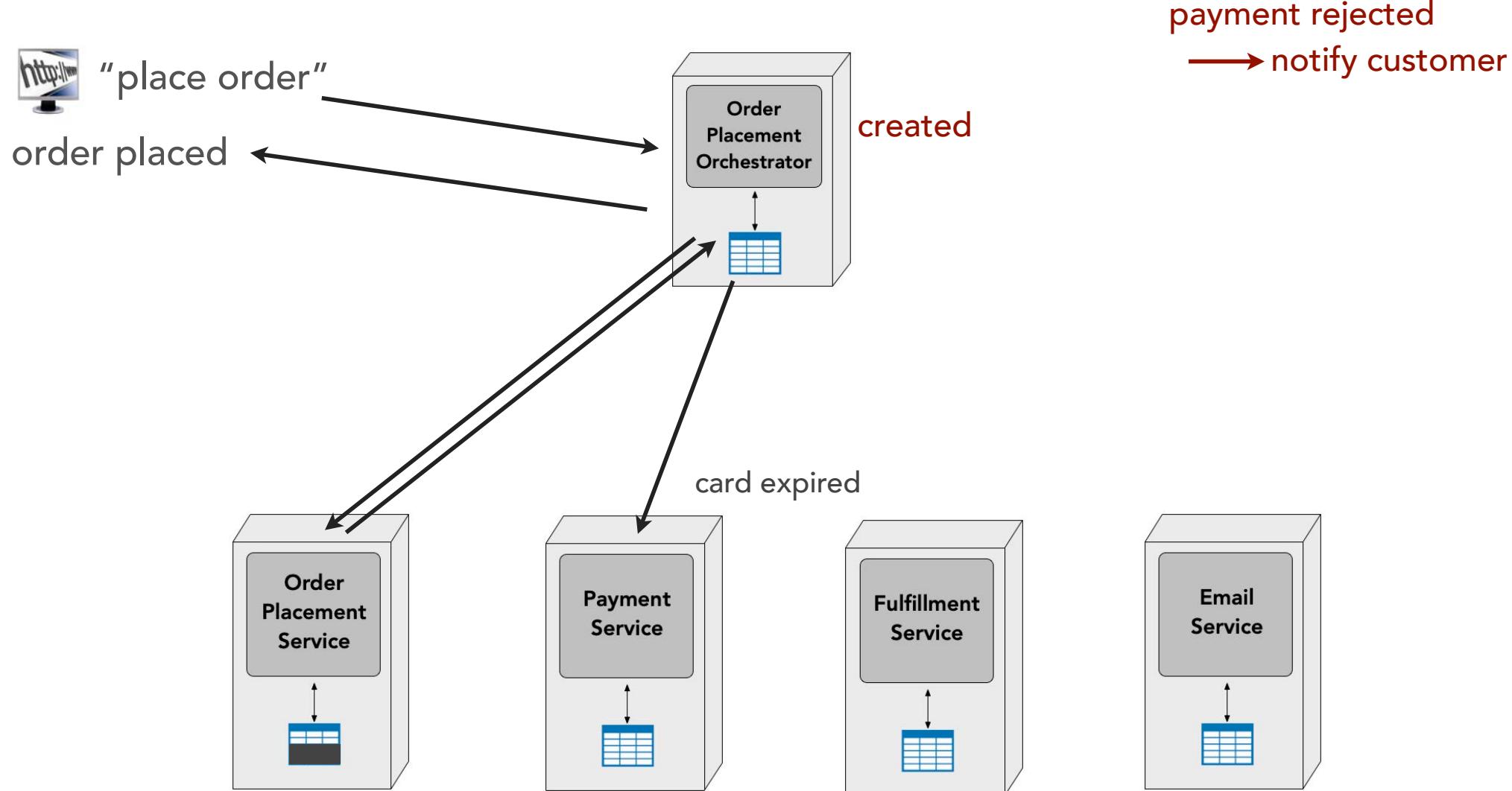
managing workflows

error handling



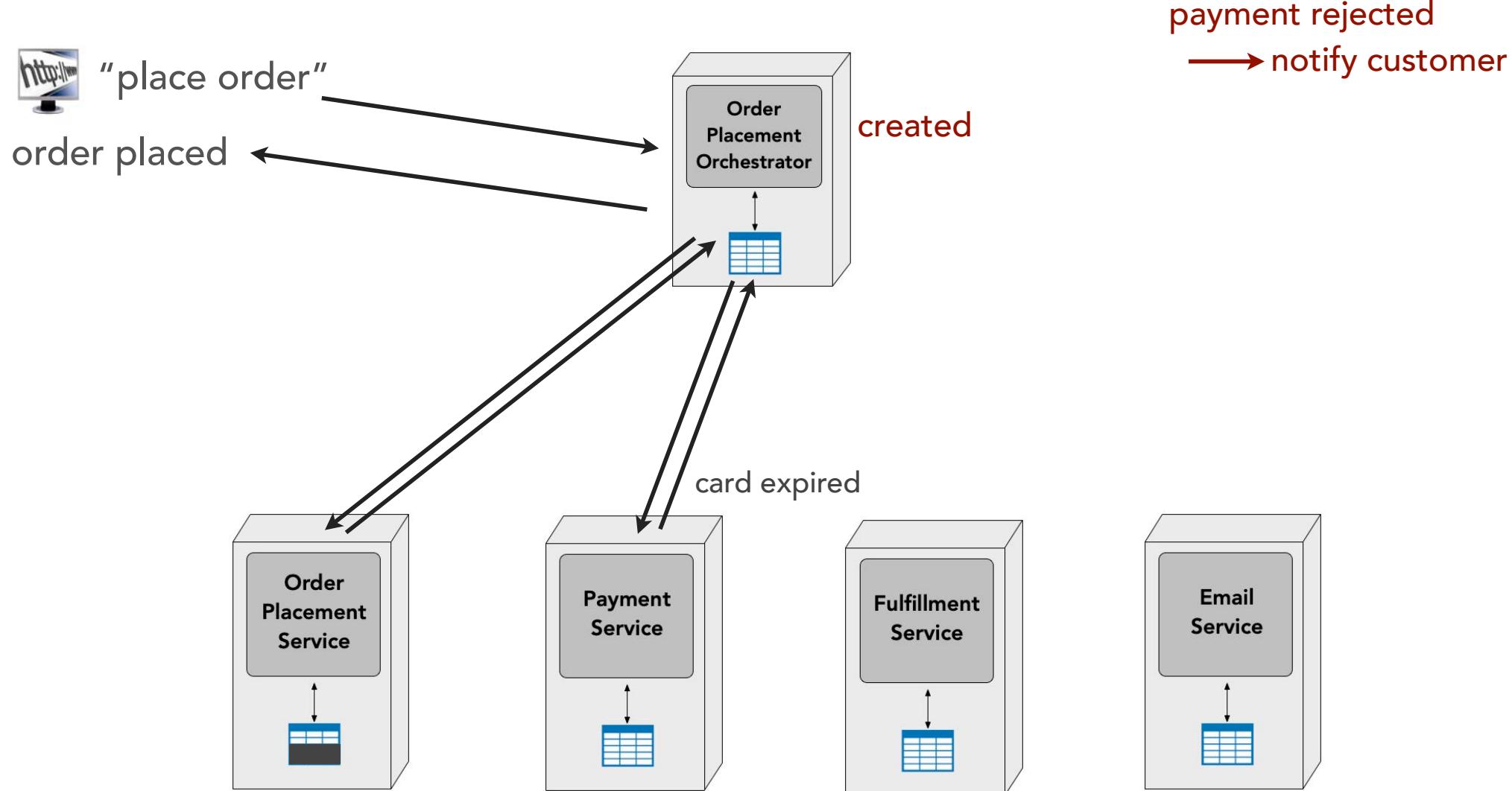
managing workflows

error handling



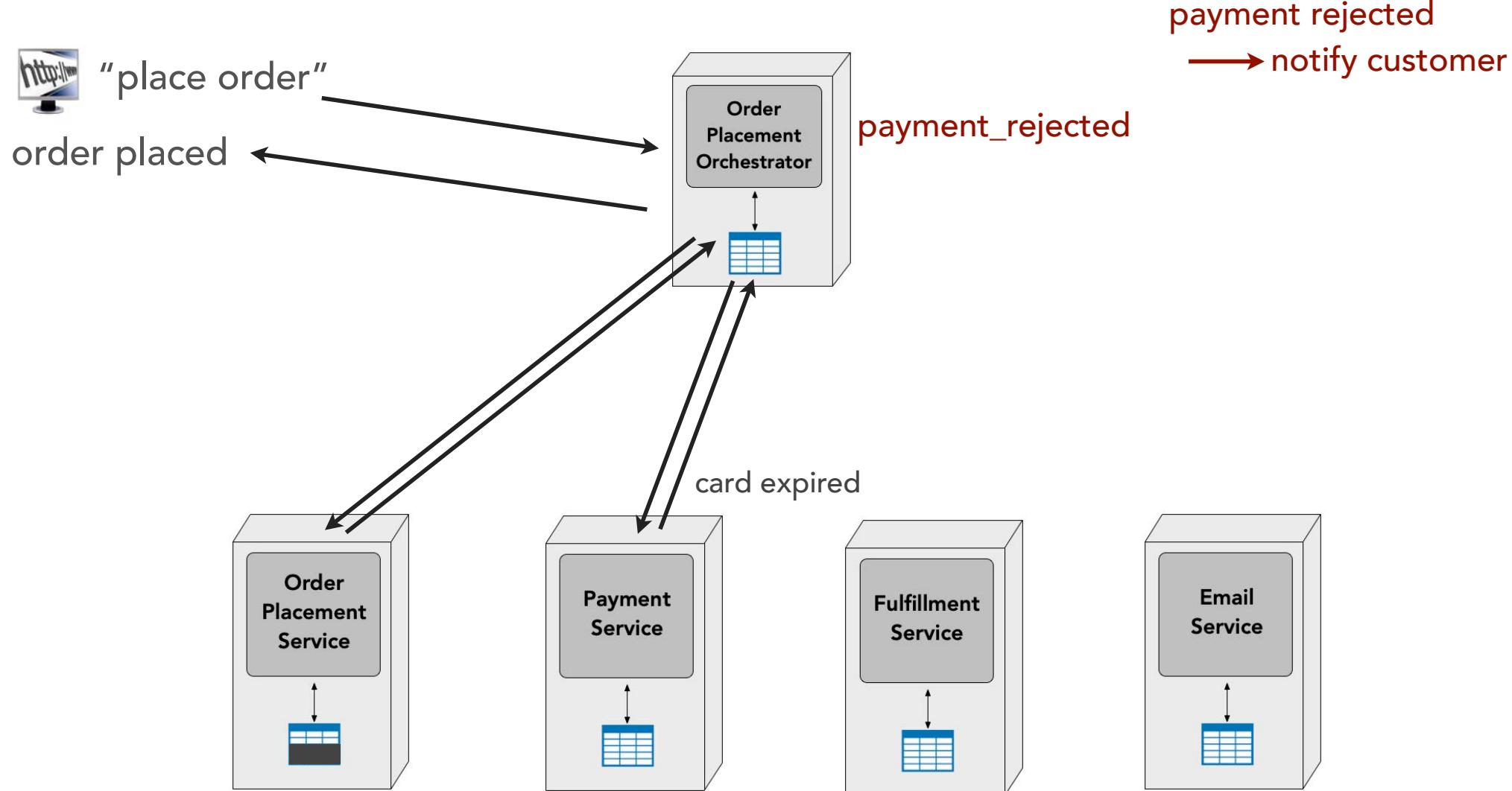
managing workflows

error handling



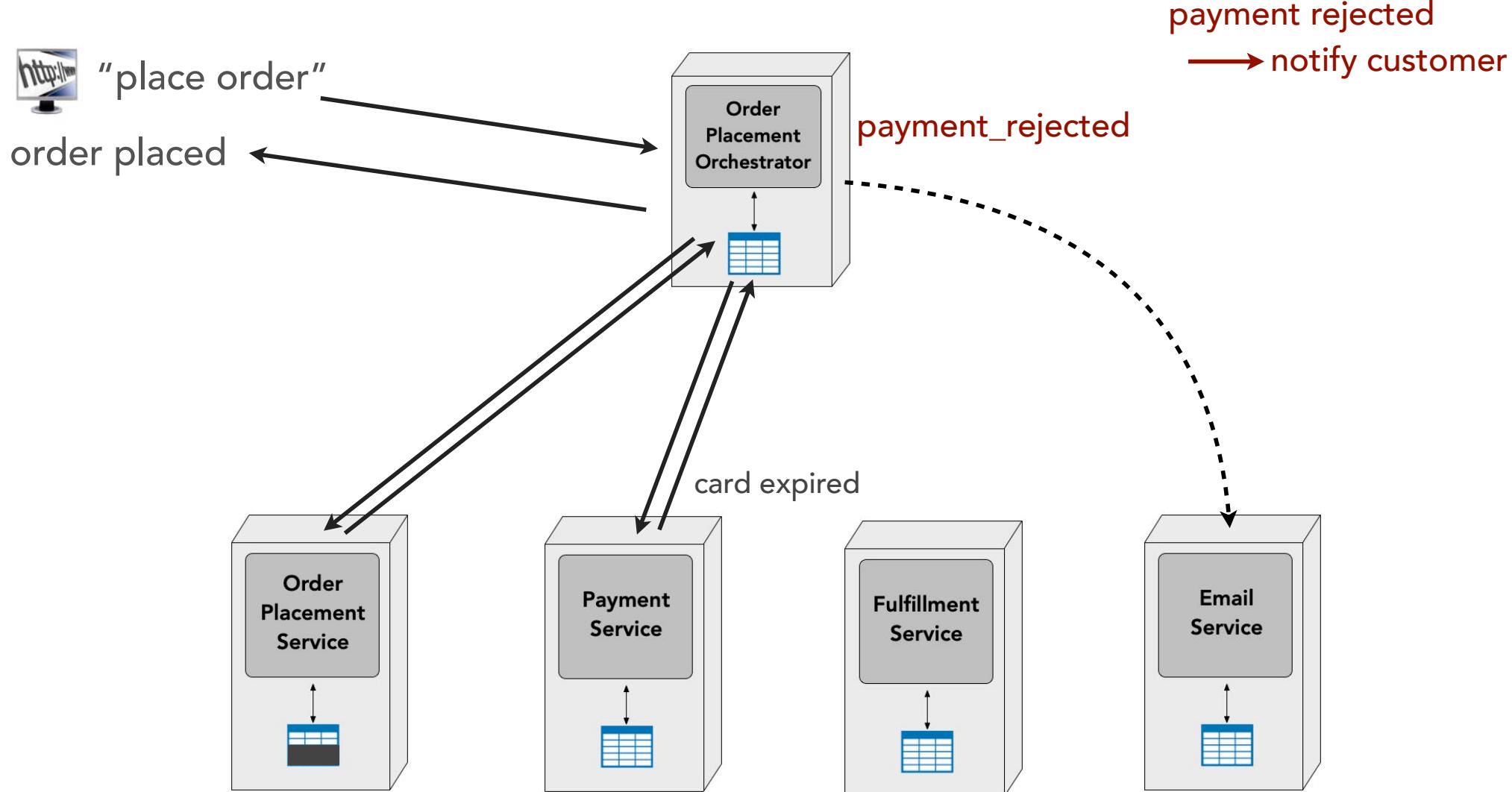
managing workflows

error handling



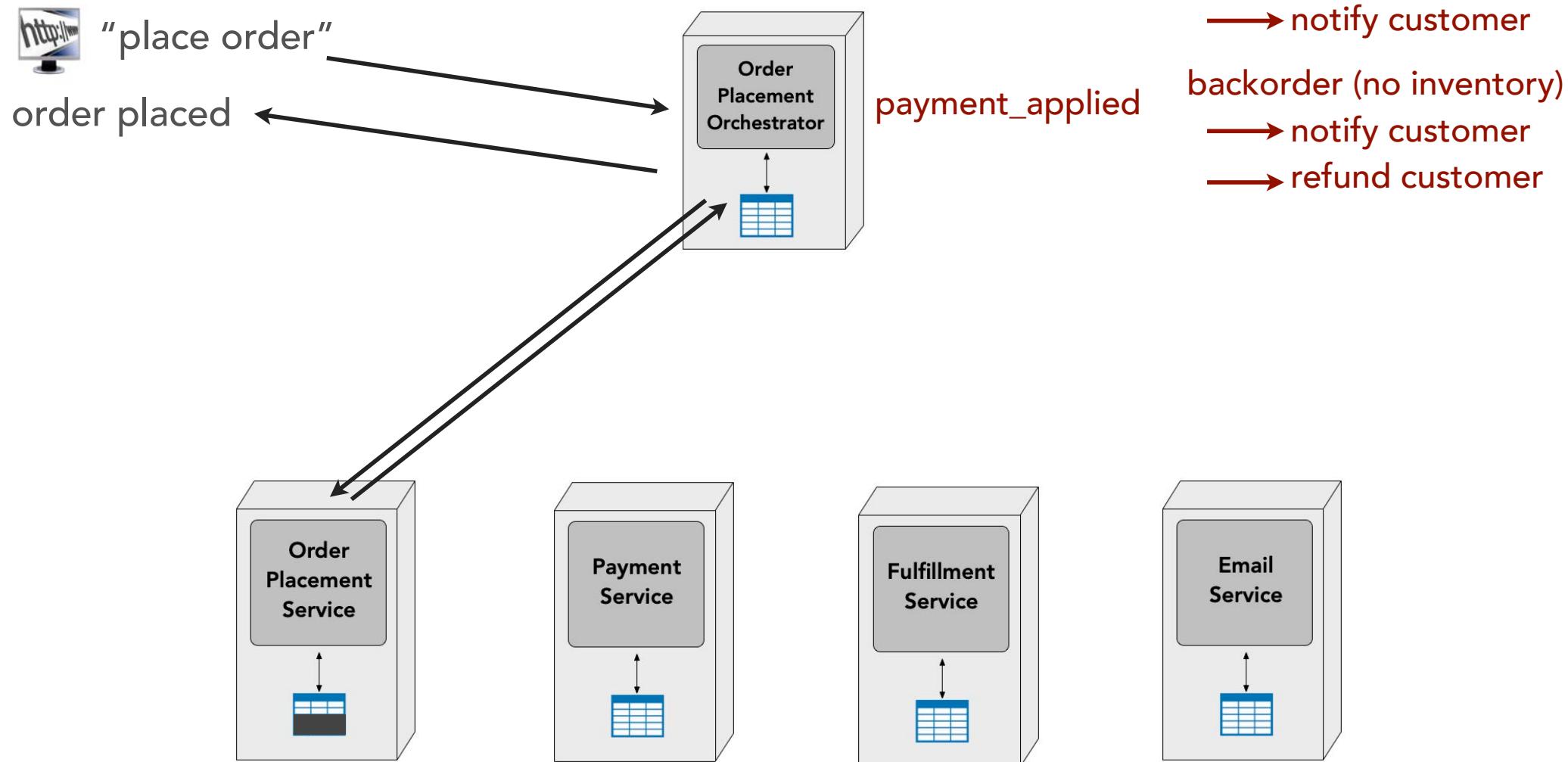
managing workflows

error handling



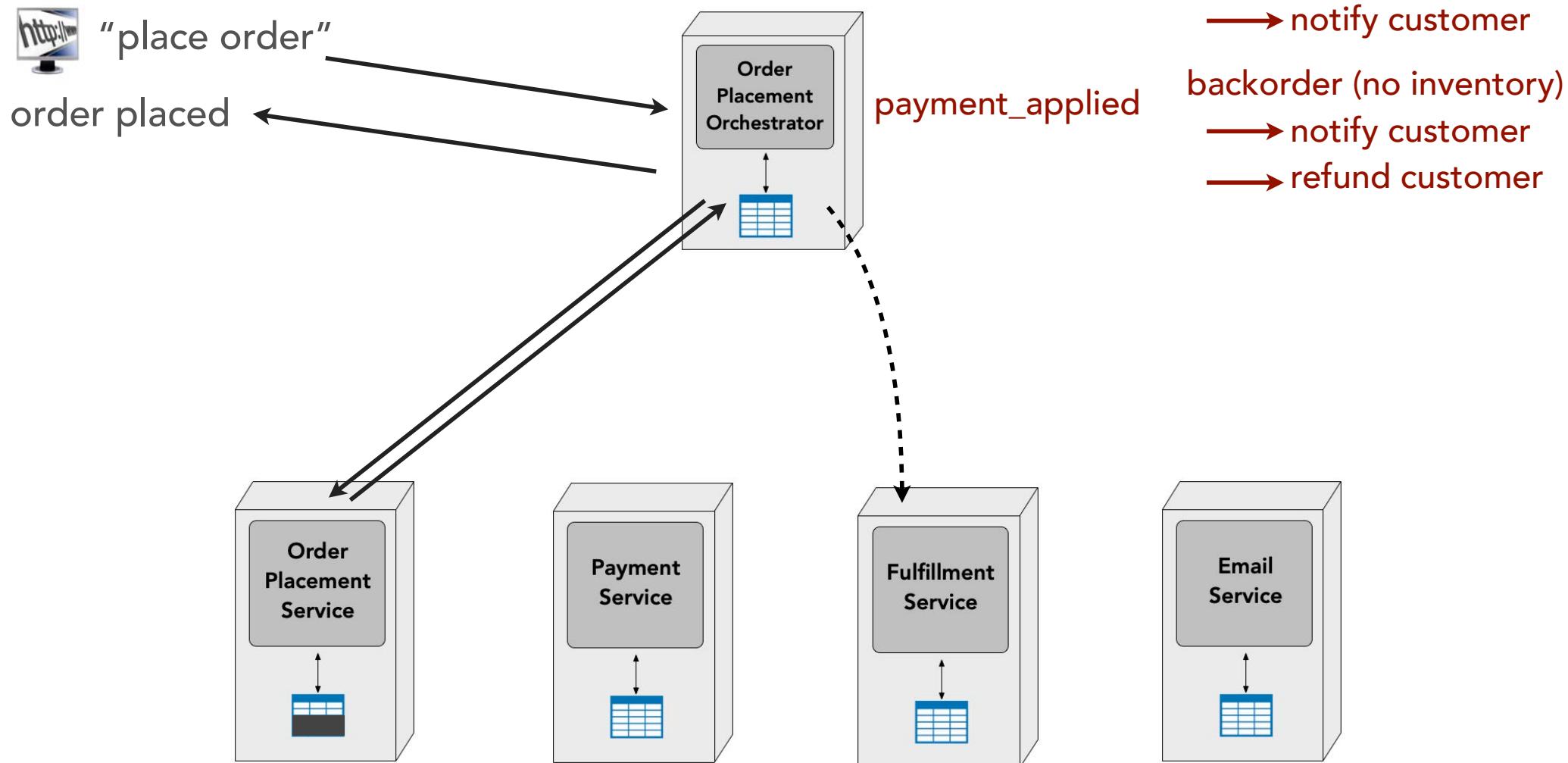
managing workflows

error handling



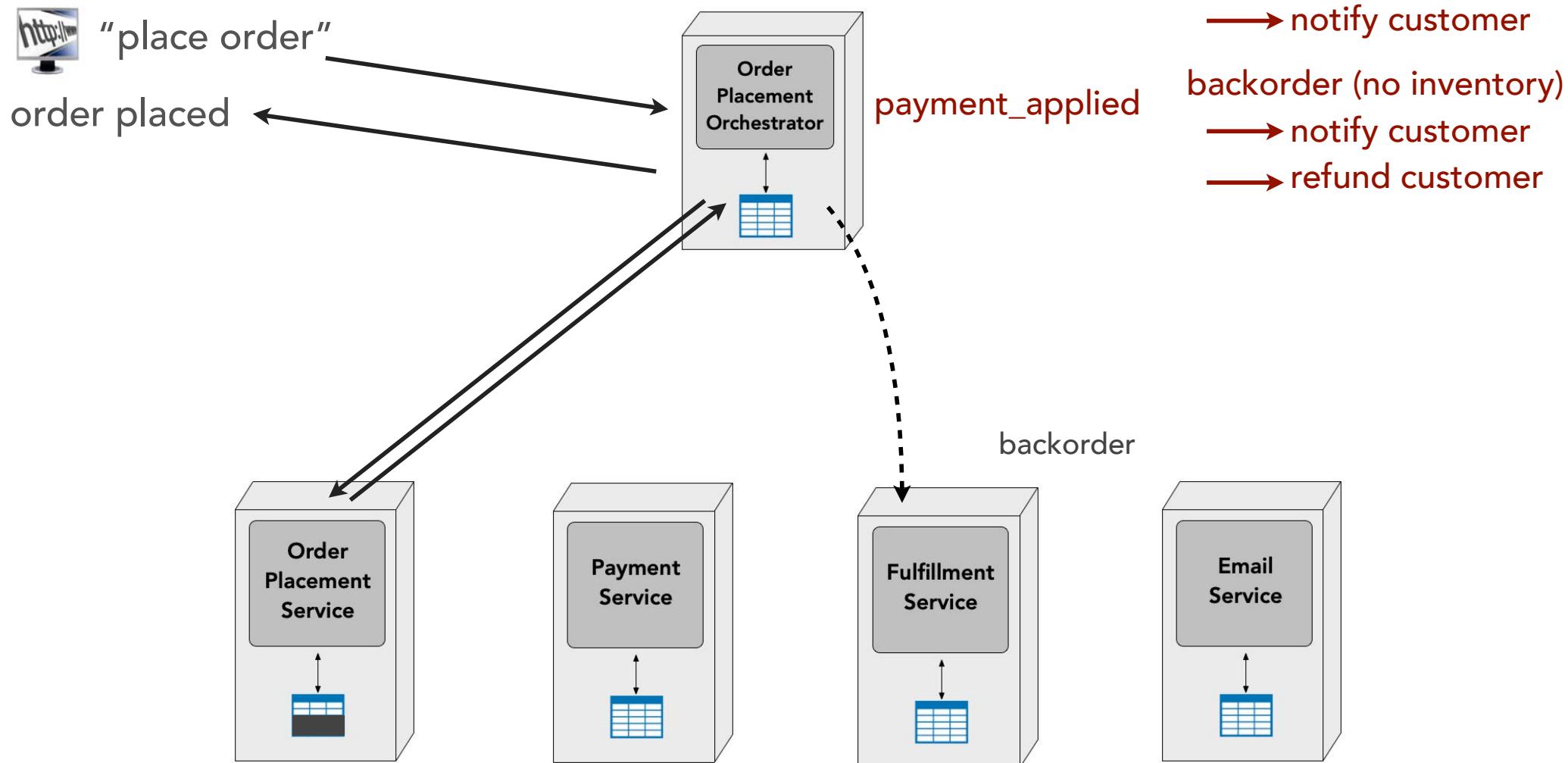
managing workflows

error handling



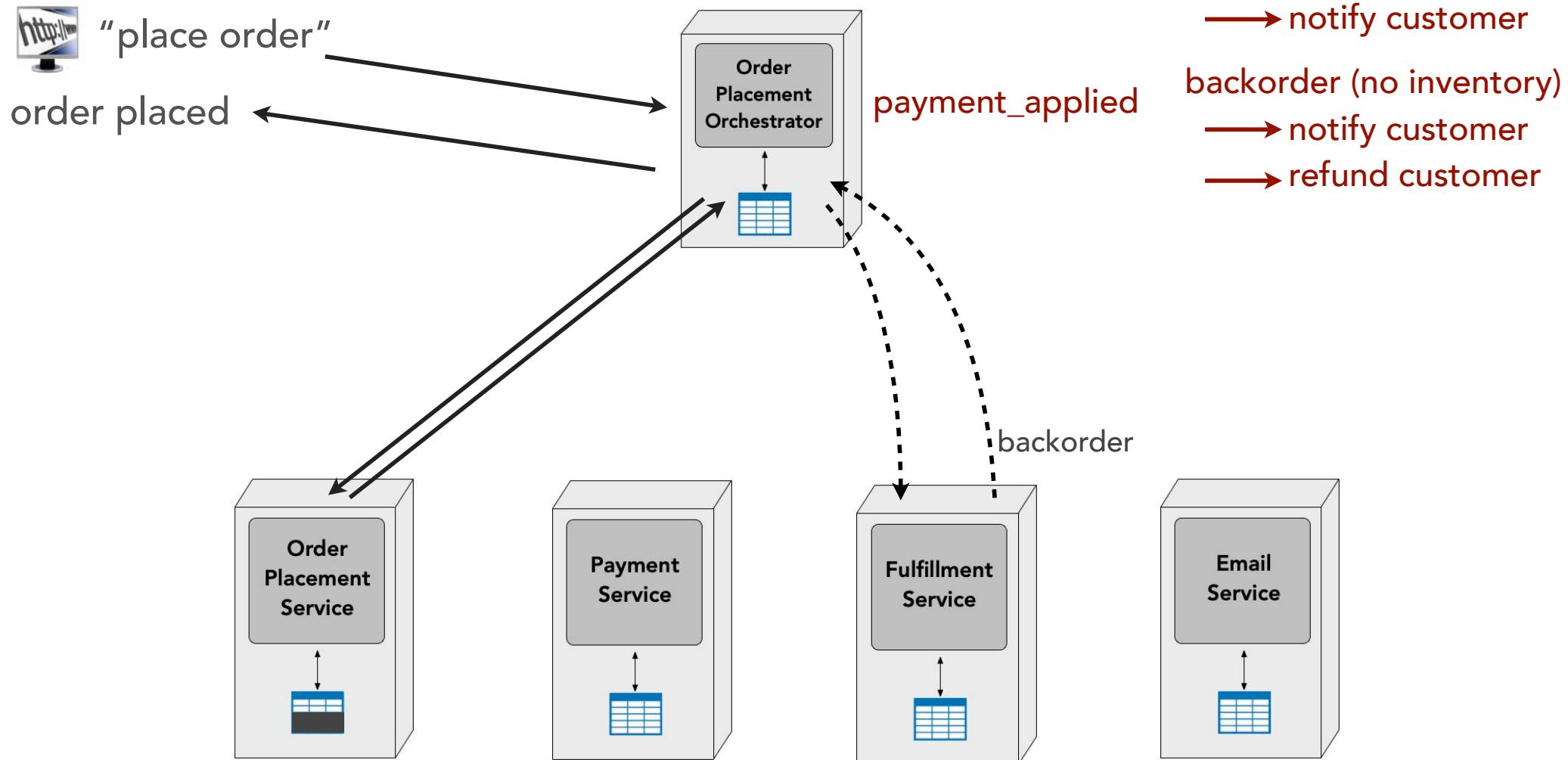
managing workflows

error handling



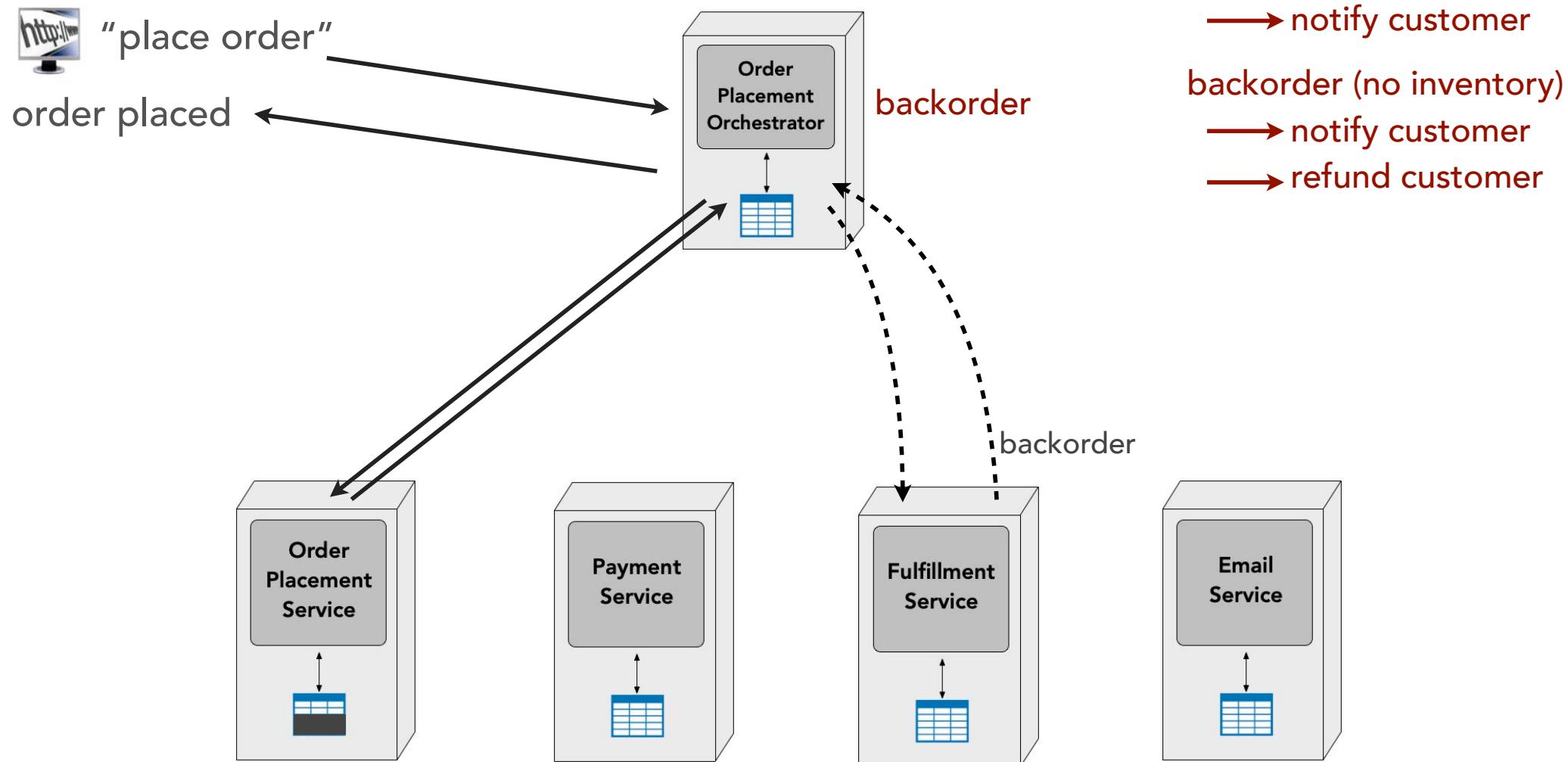
managing workflows

error handling



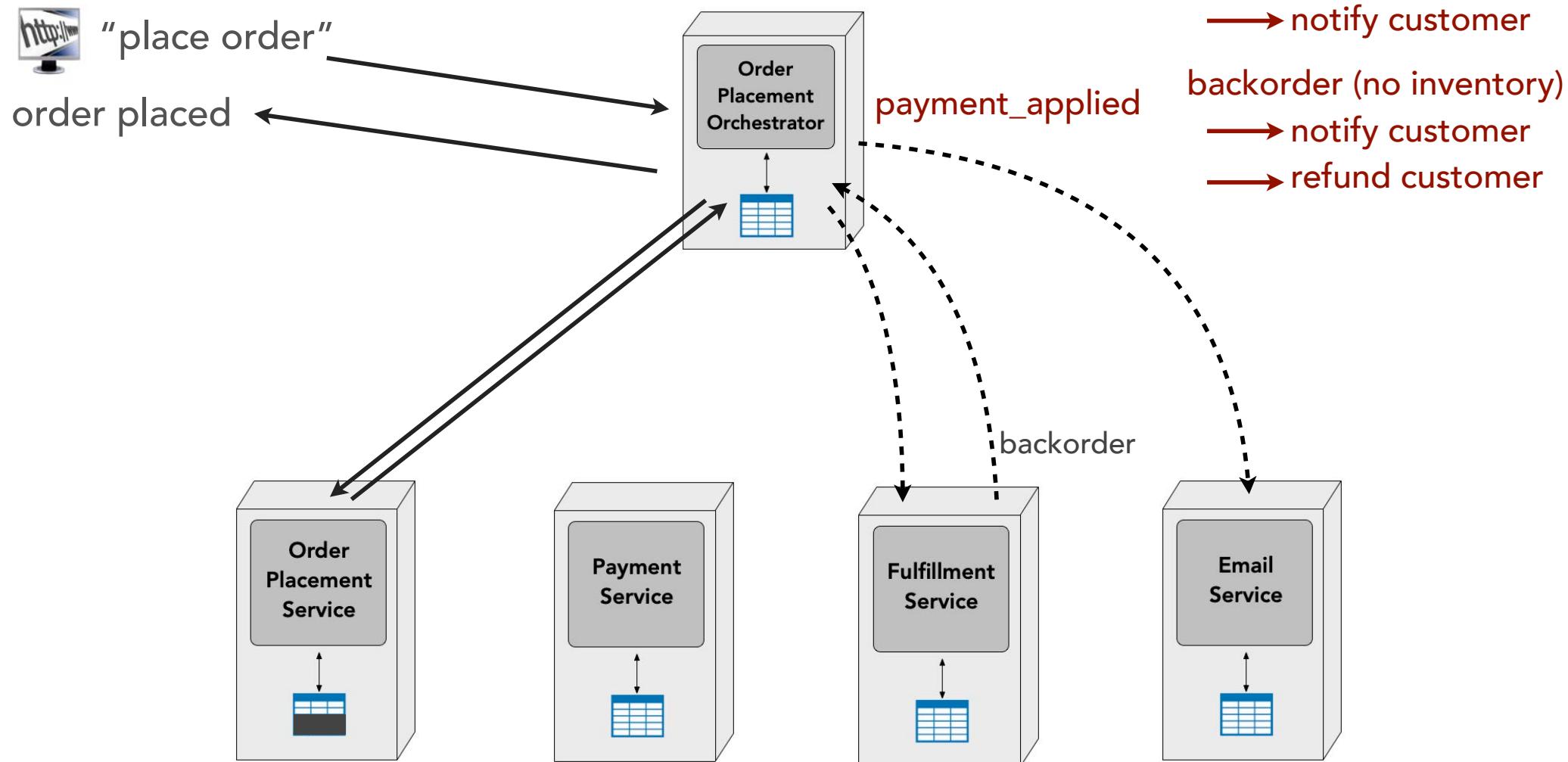
managing workflows

error handling



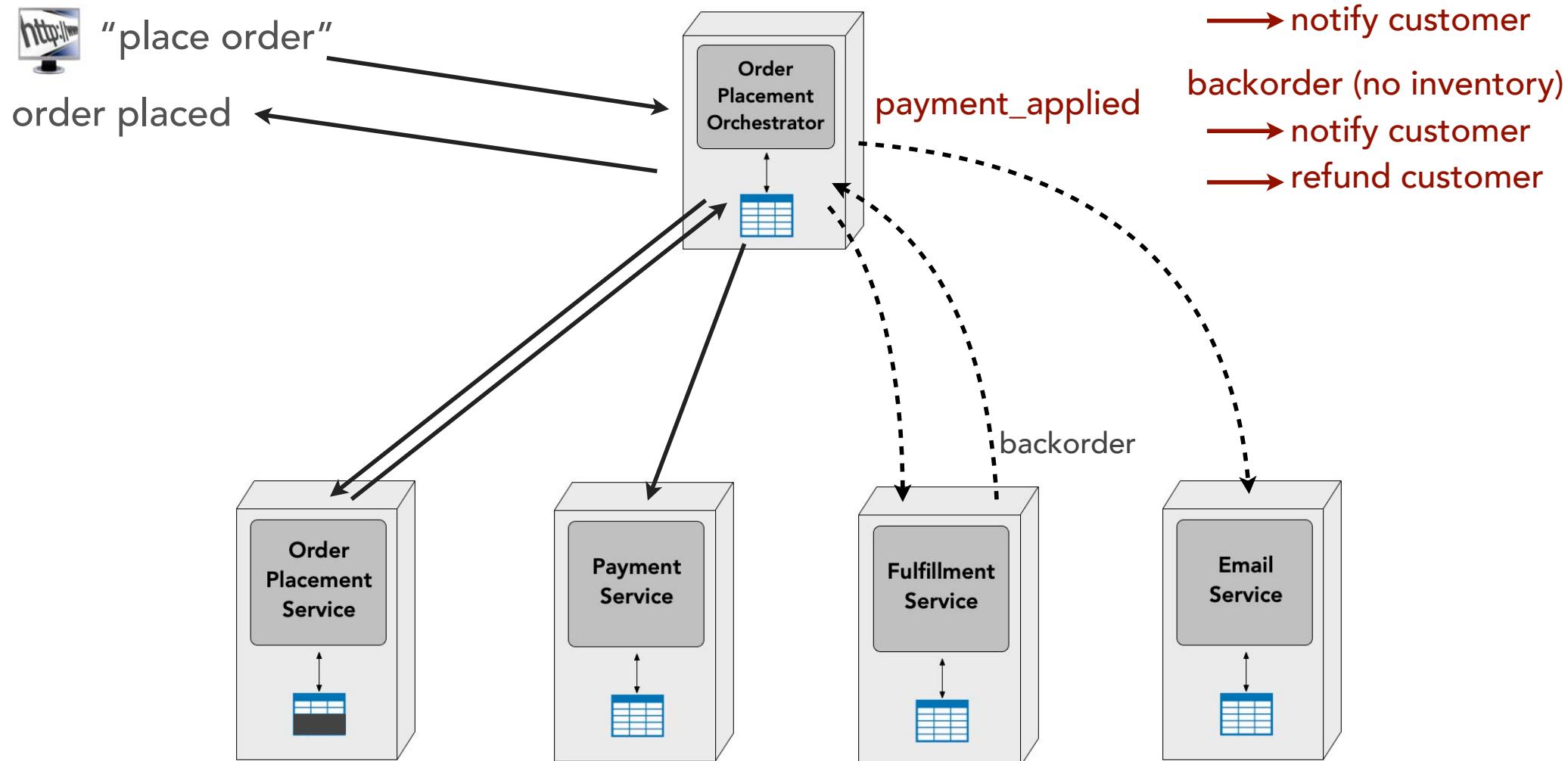
managing workflows

error handling



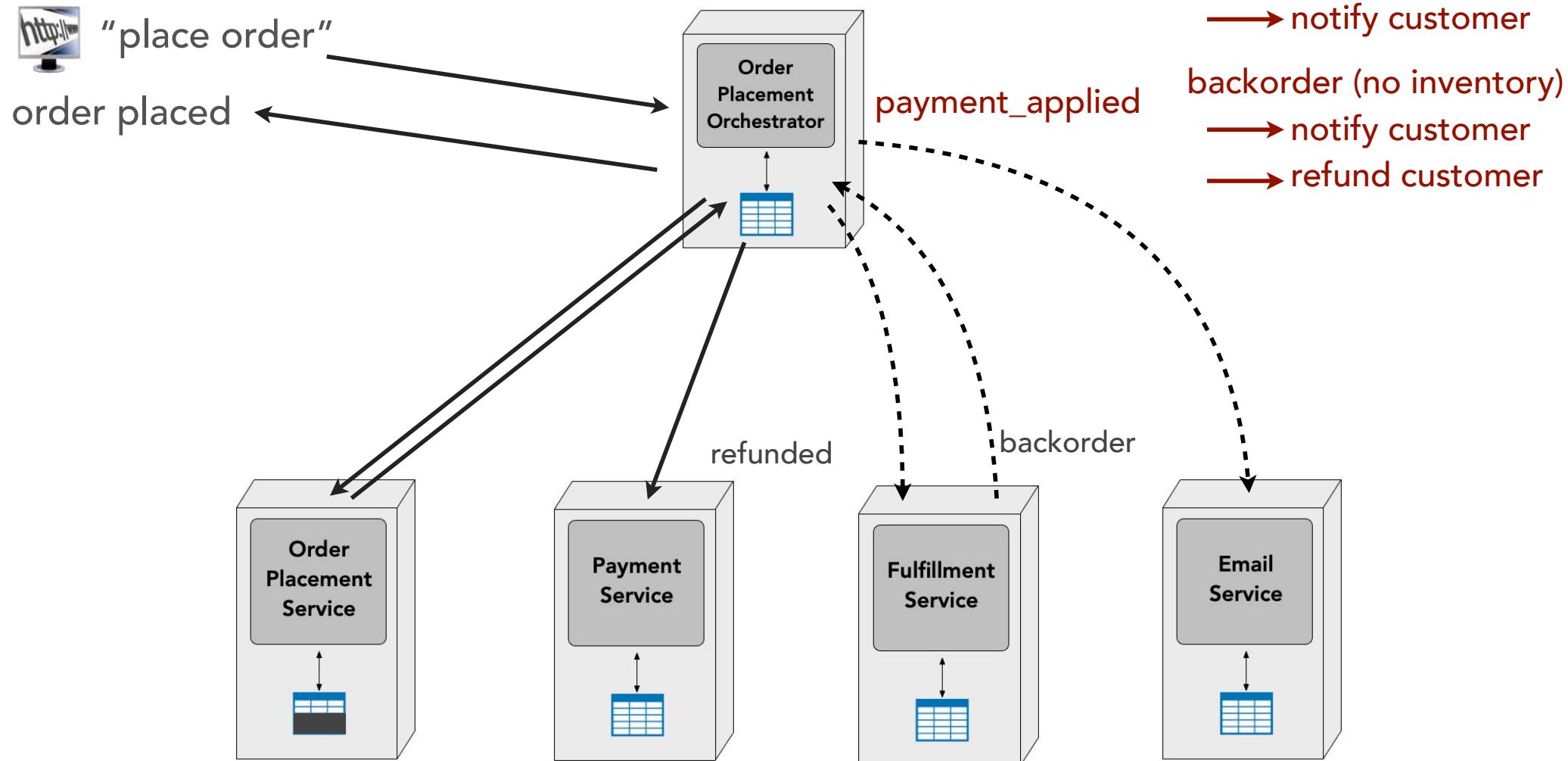
managing workflows

error handling



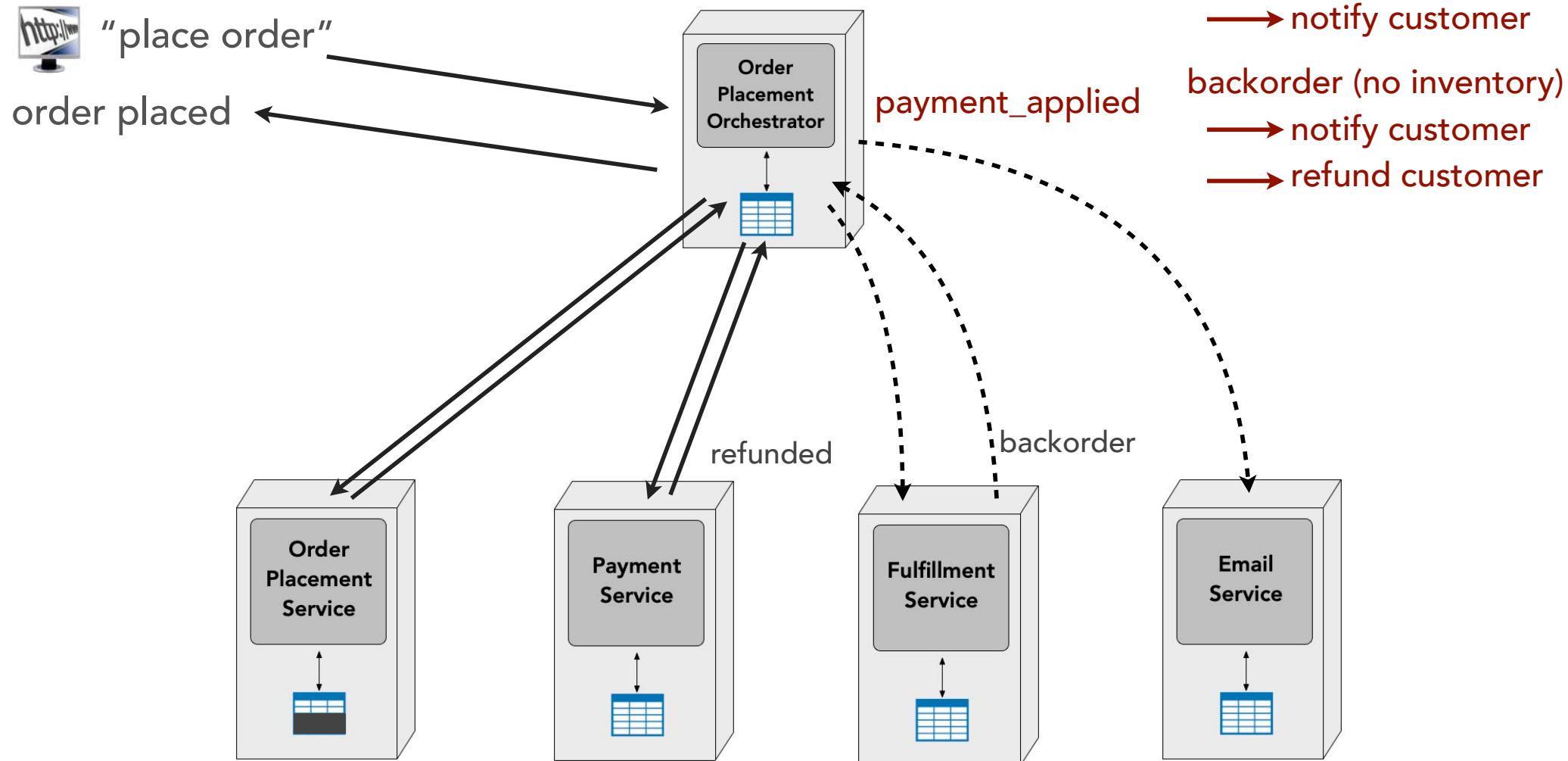
managing workflows

error handling



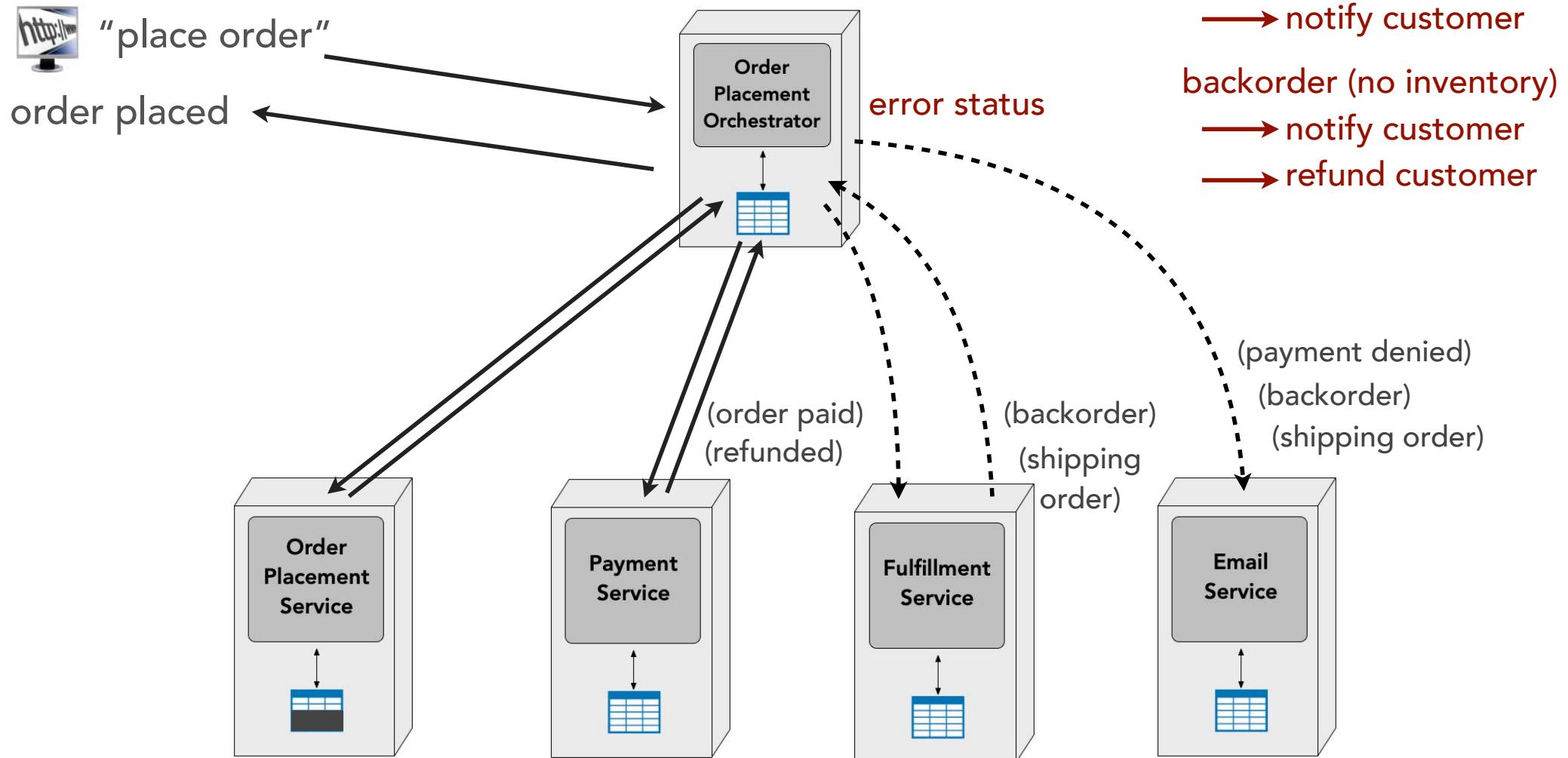
managing workflows

error handling



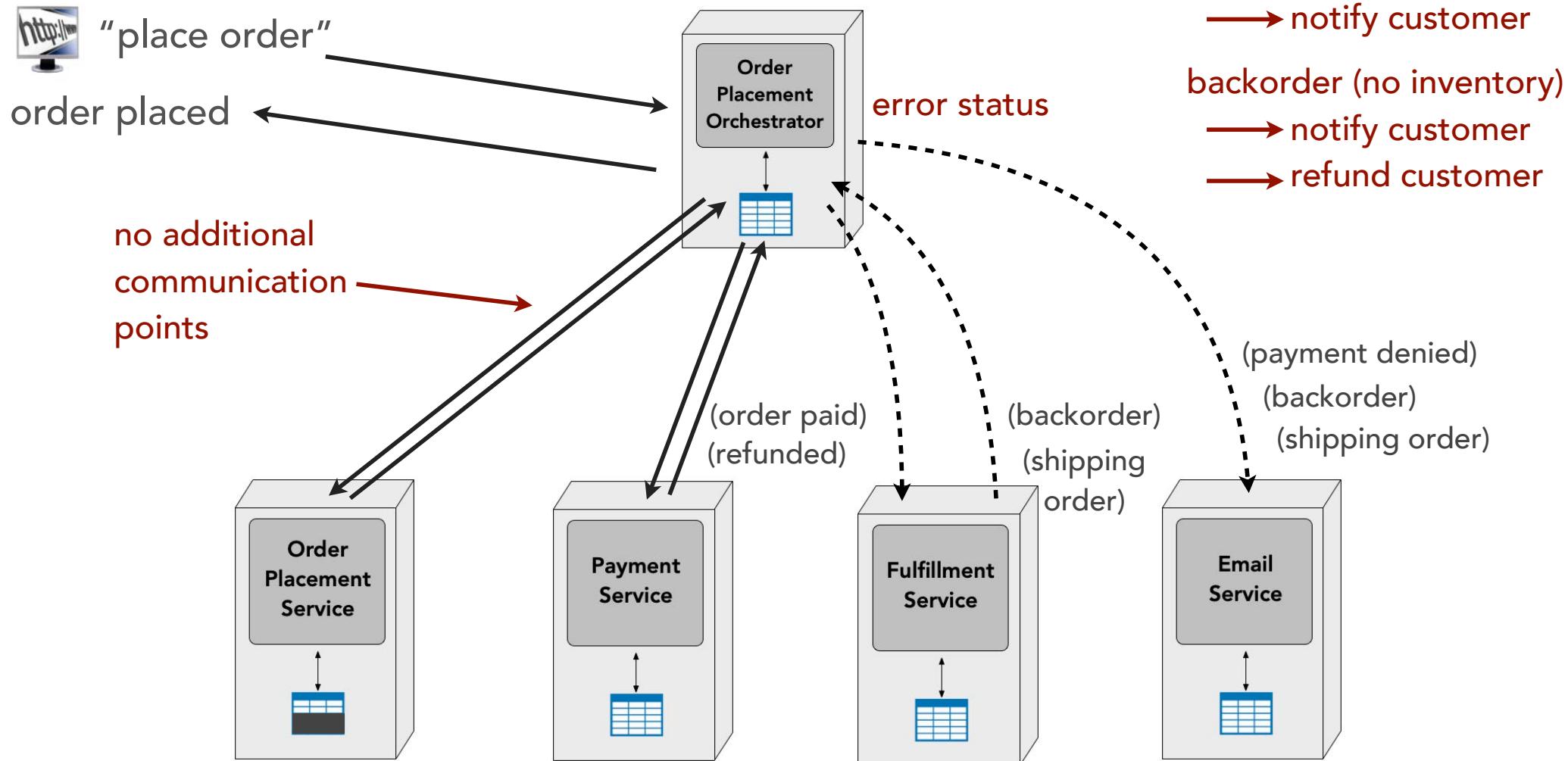
managing workflows

error handling



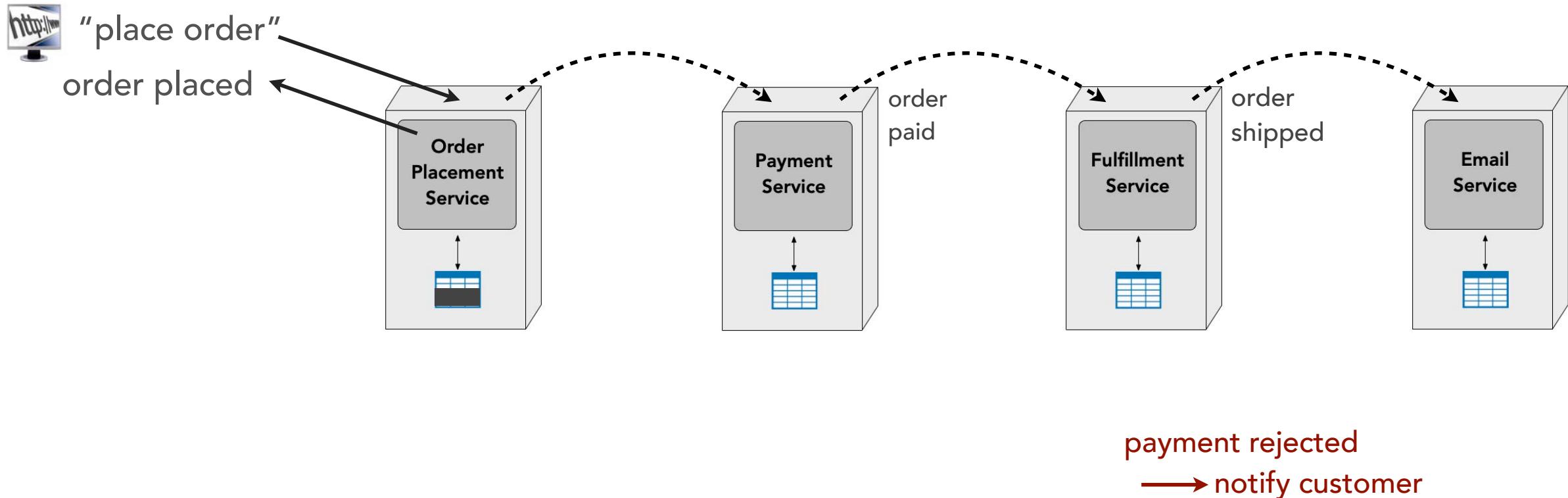
managing workflows

error handling



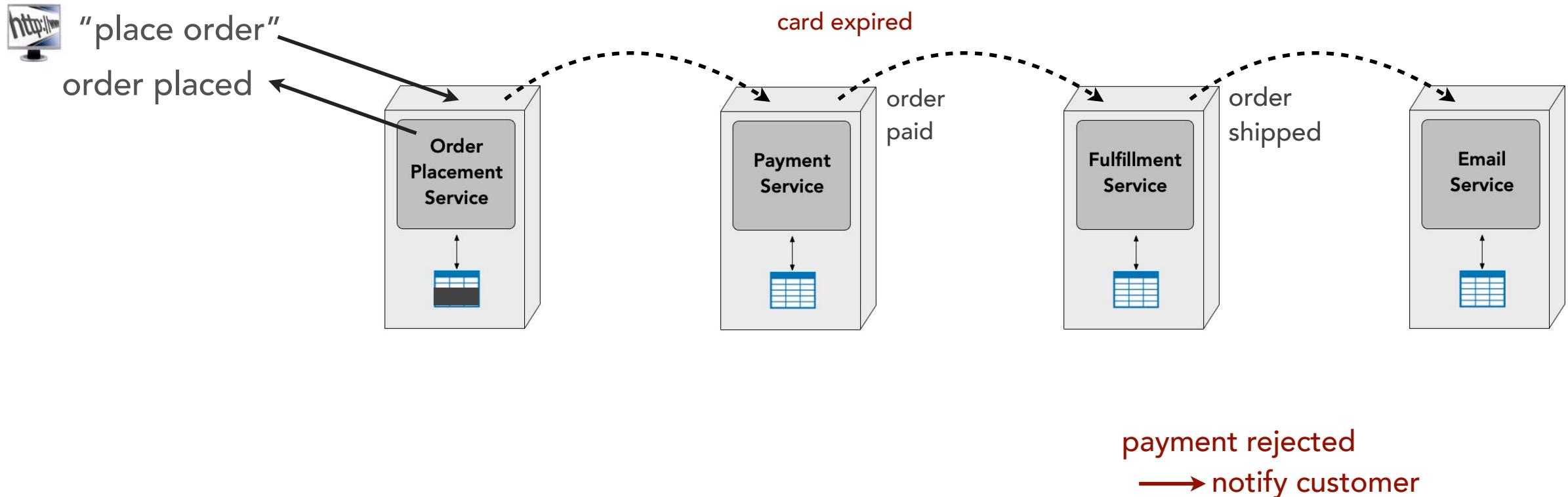
managing workflows

error handling



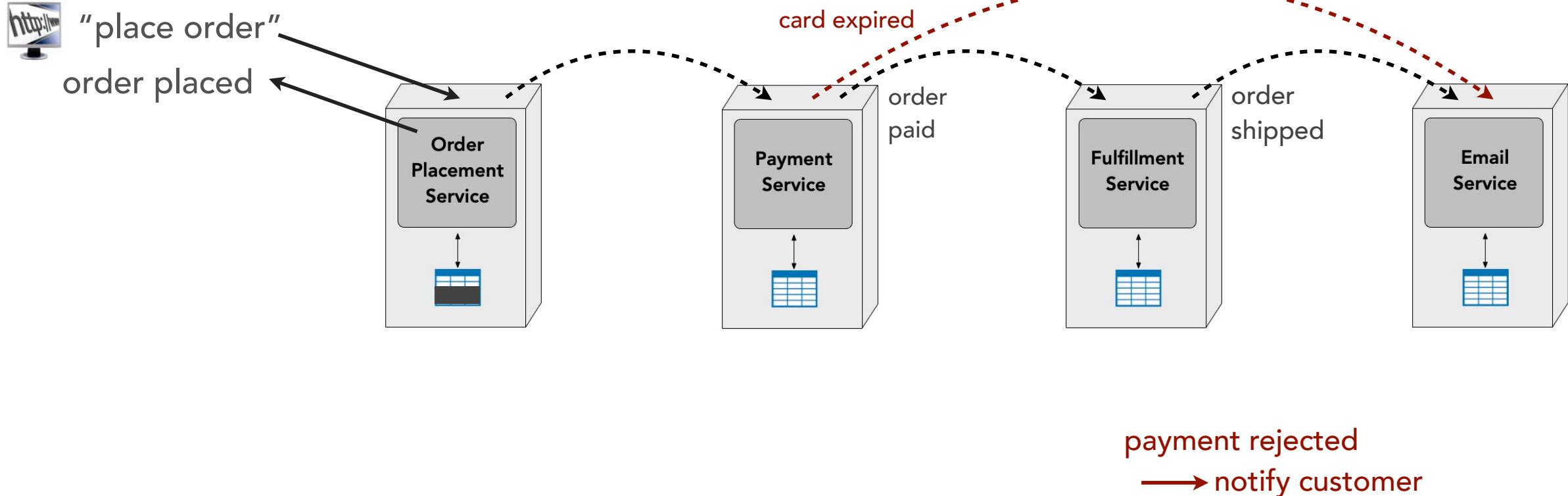
managing workflows

error handling



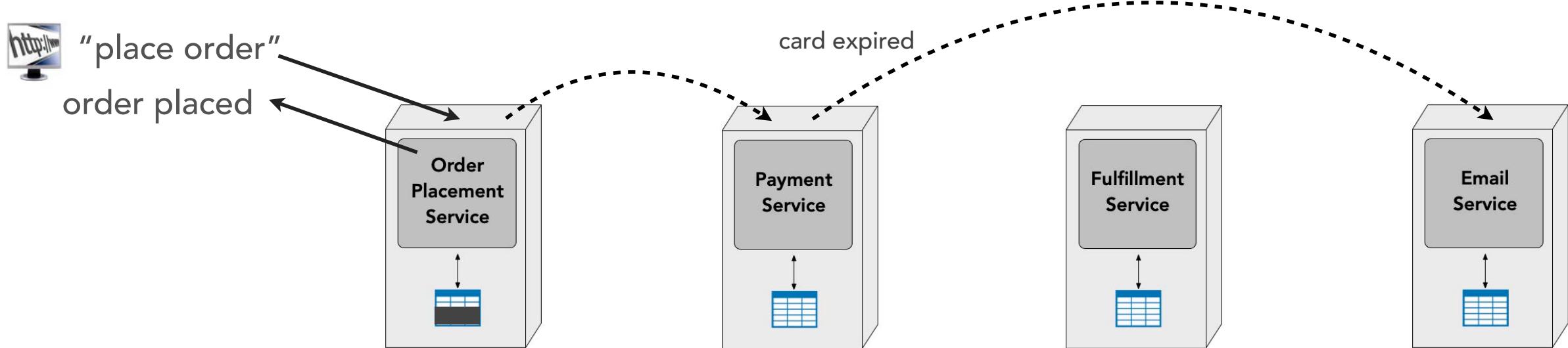
managing workflows

error handling



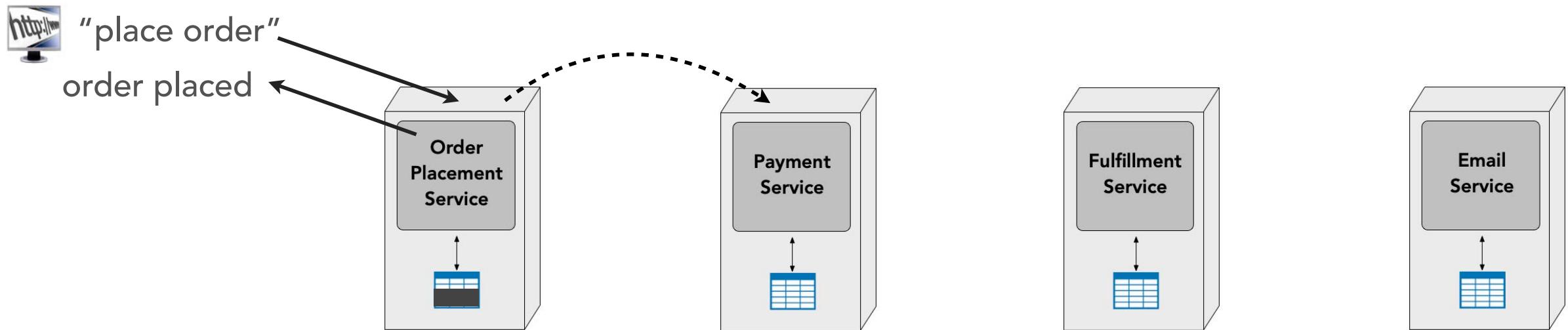
managing workflows

error handling



managing workflows

error handling



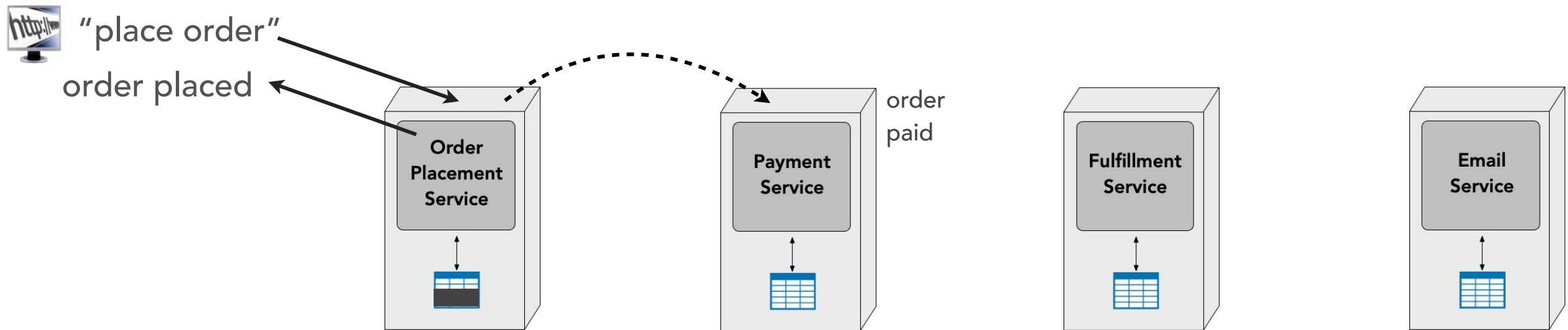
backorder (no inventory)

→ notify customer

→ refund customer

managing workflows

error handling



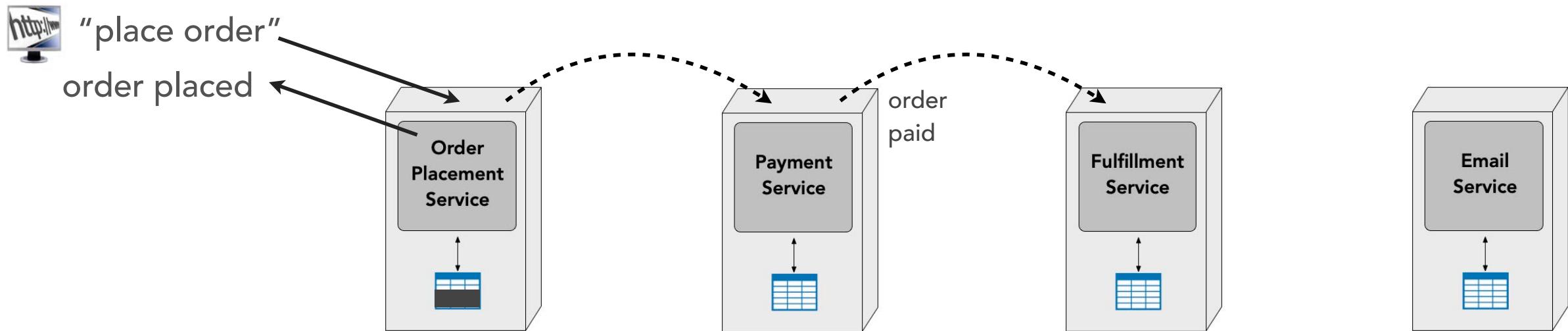
backorder (no inventory)

→ notify customer

→ refund customer

managing workflows

error handling



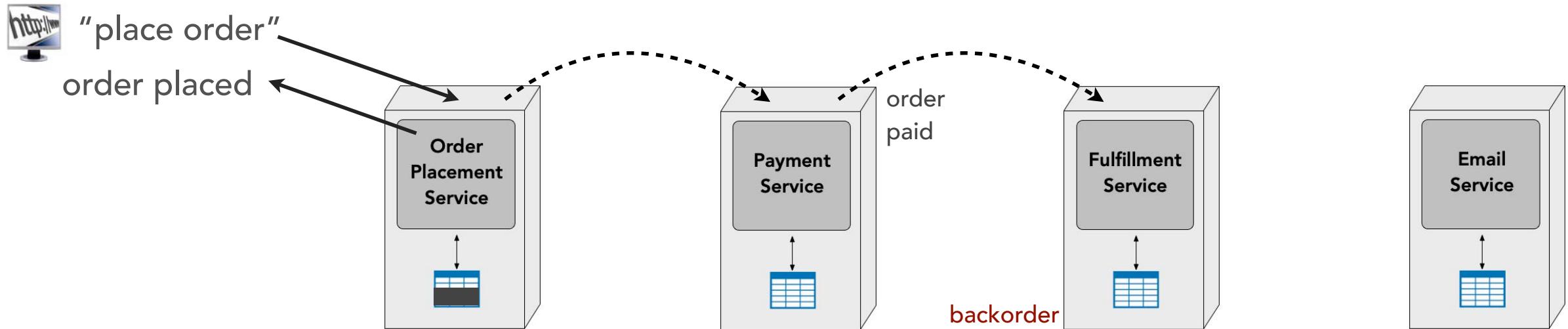
backorder (no inventory)

→ notify customer

→ refund customer

managing workflows

error handling



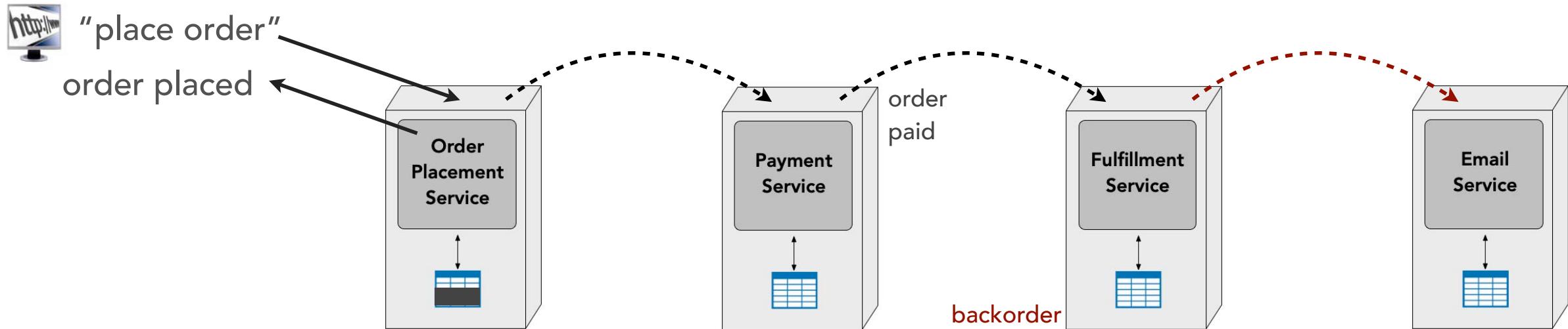
backorder (no inventory)

→ notify customer

→ refund customer

managing workflows

error handling



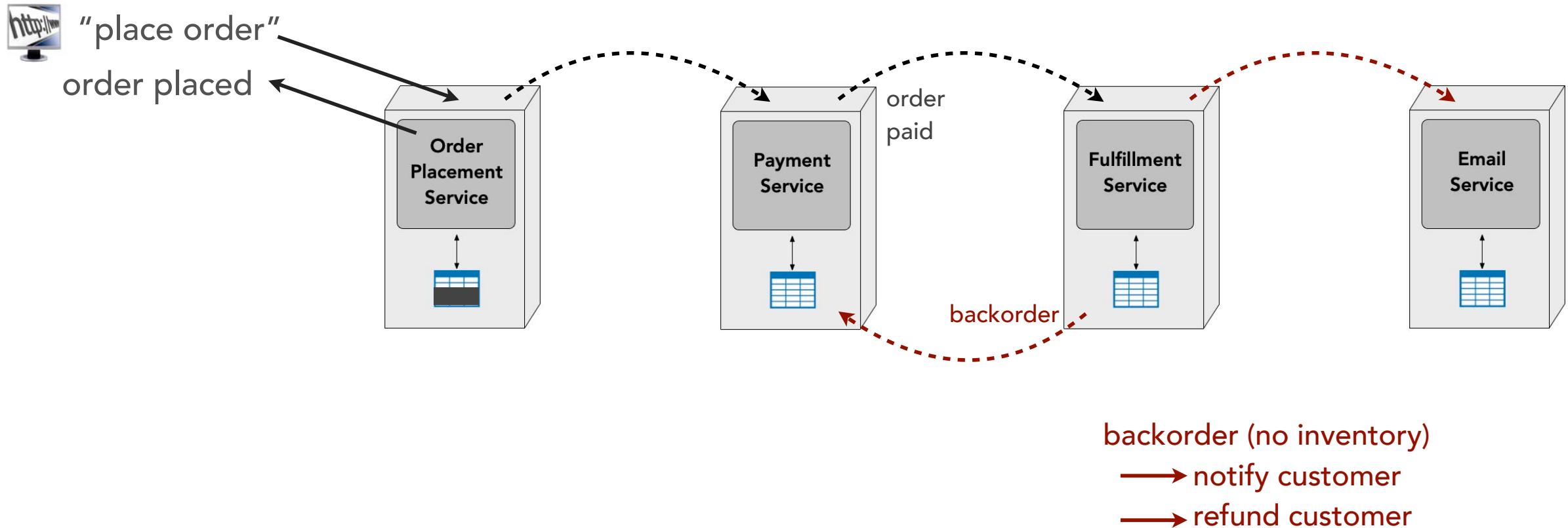
backorder (no inventory)

→ notify customer

→ refund customer

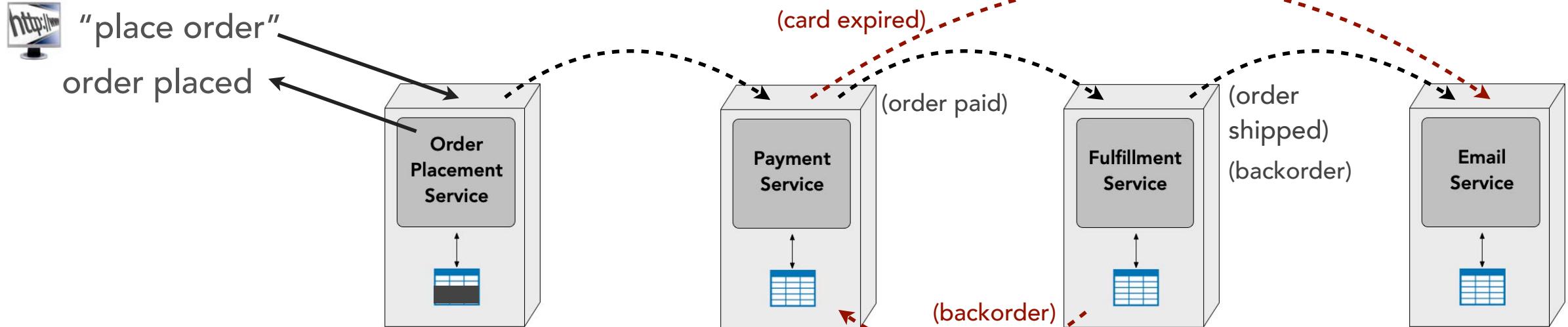
managing workflows

error handling



managing workflows

error handling

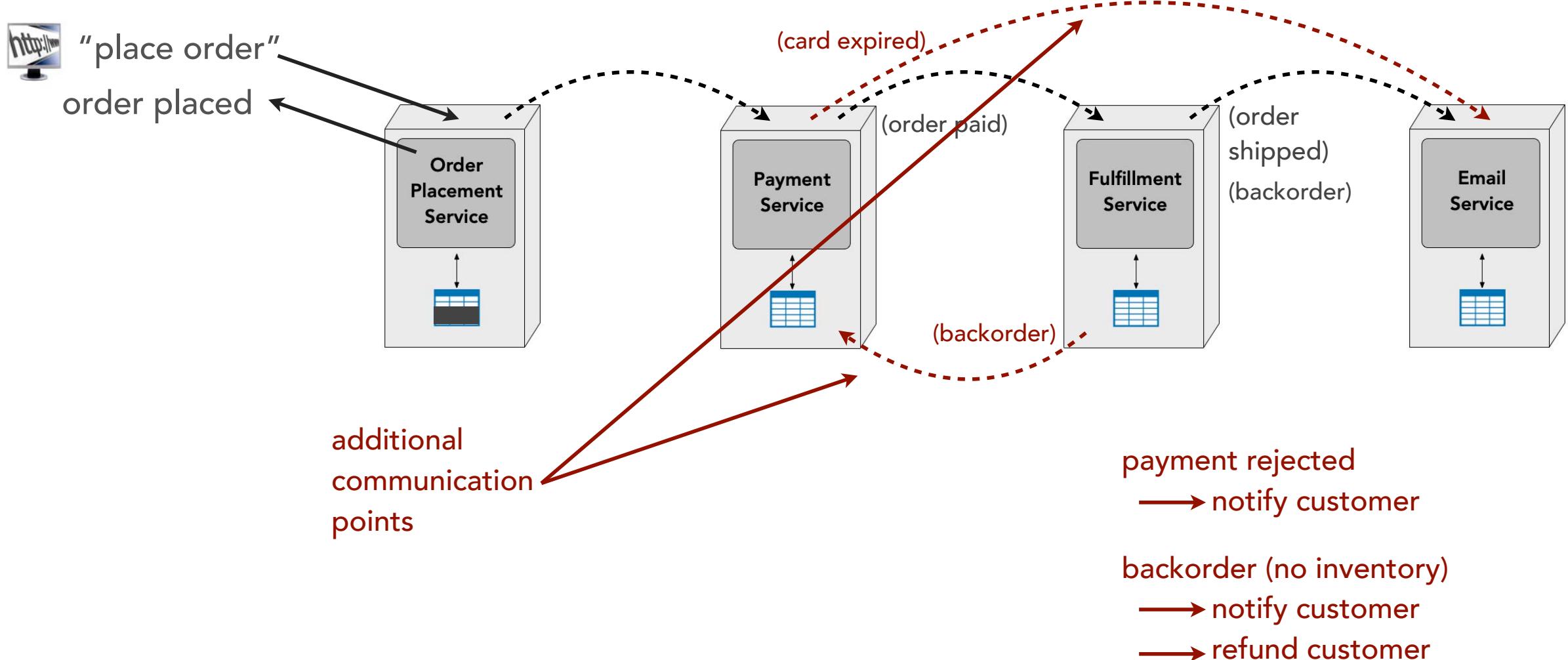


payment rejected
→ notify customer

backorder (no inventory)
→ notify customer
→ refund customer

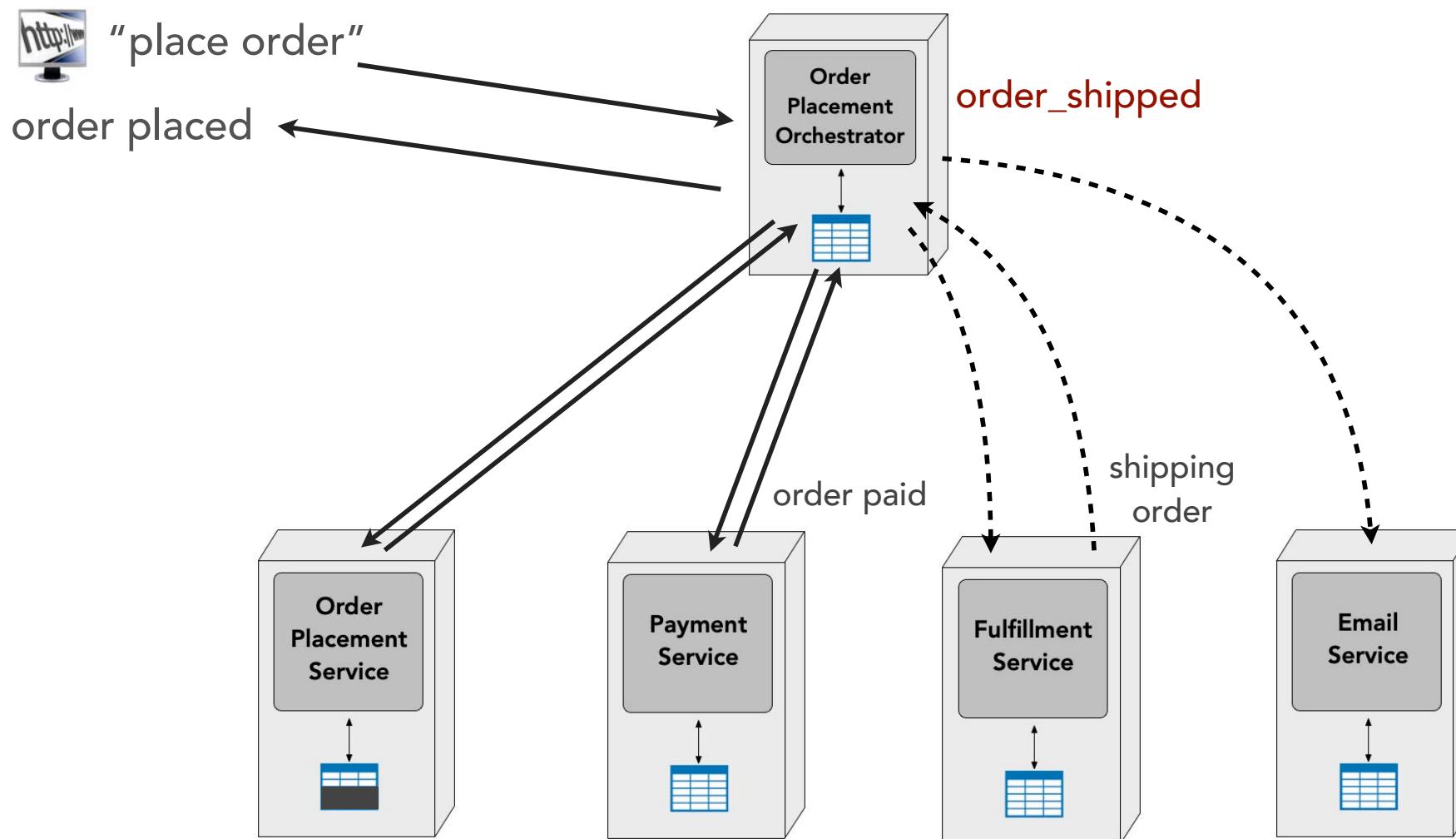
managing workflows

error handling

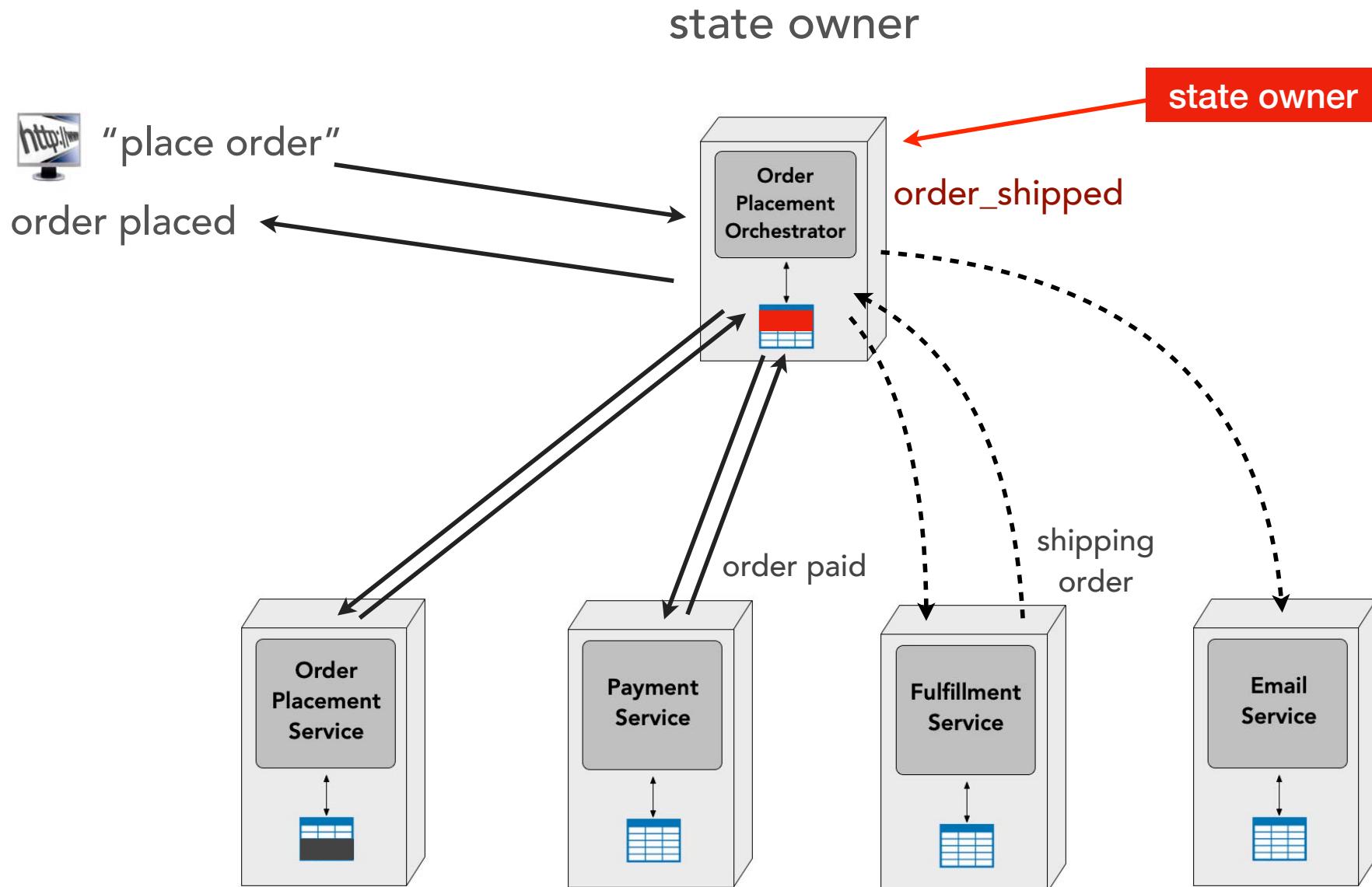


managing workflows

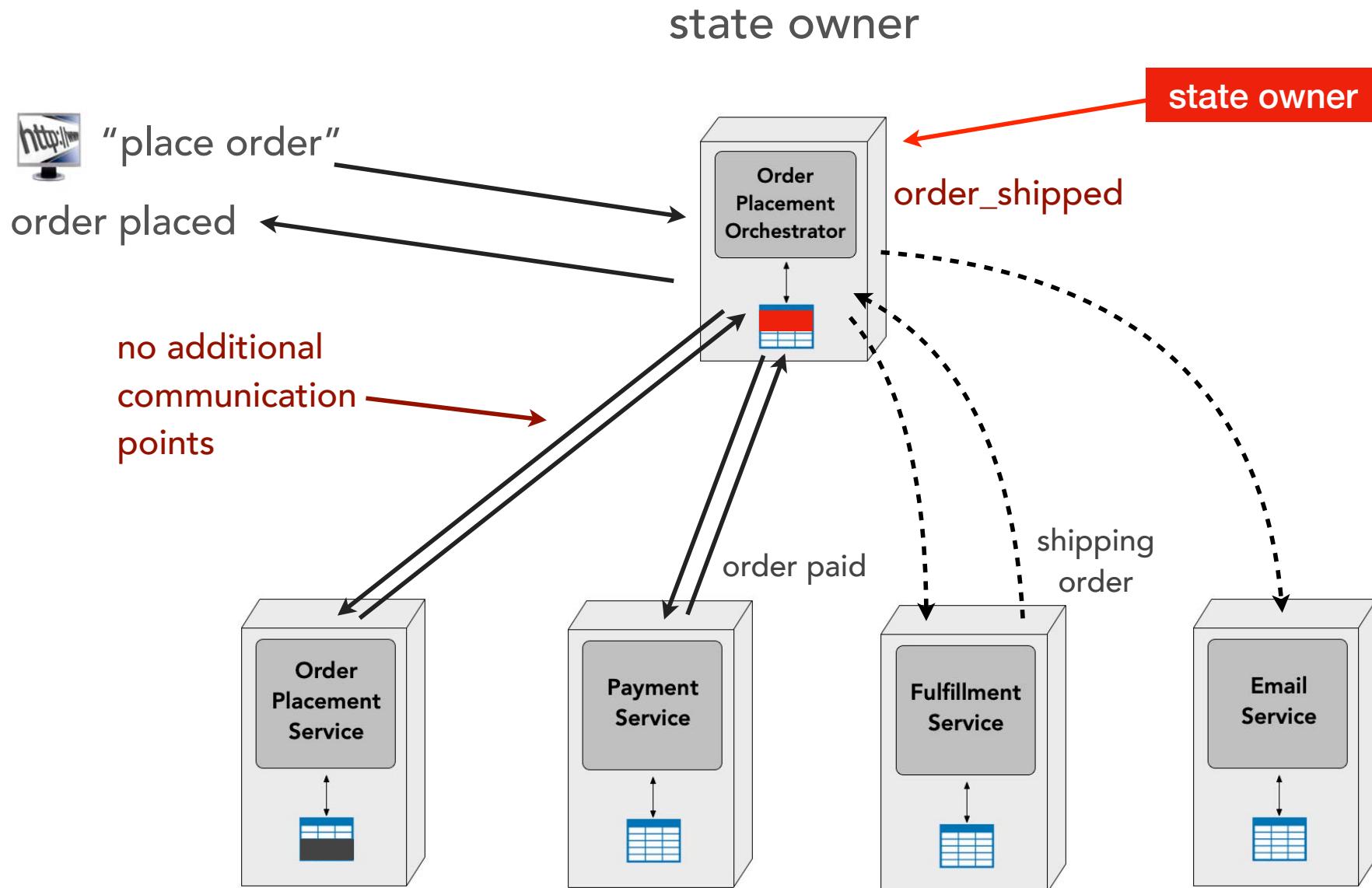
state owner



managing workflows

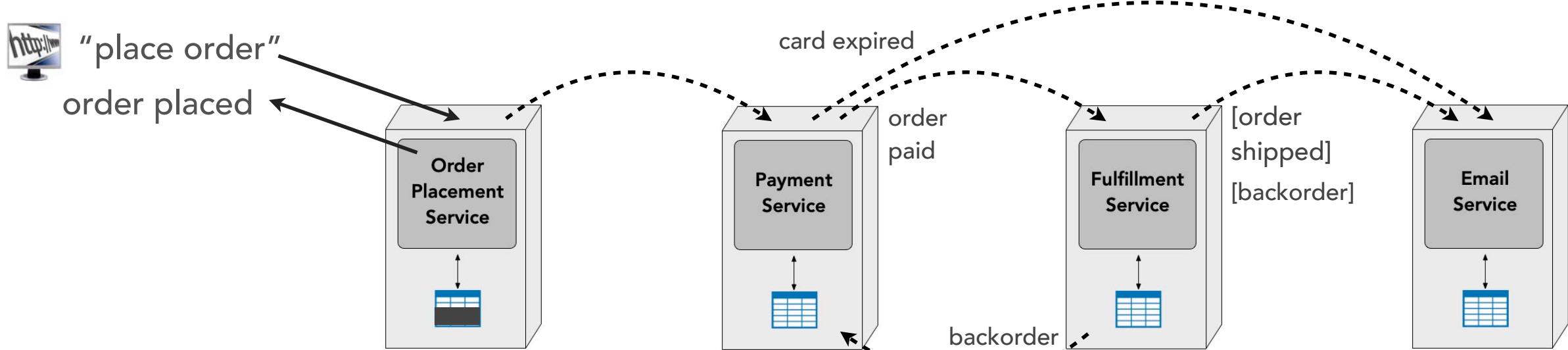


managing workflows



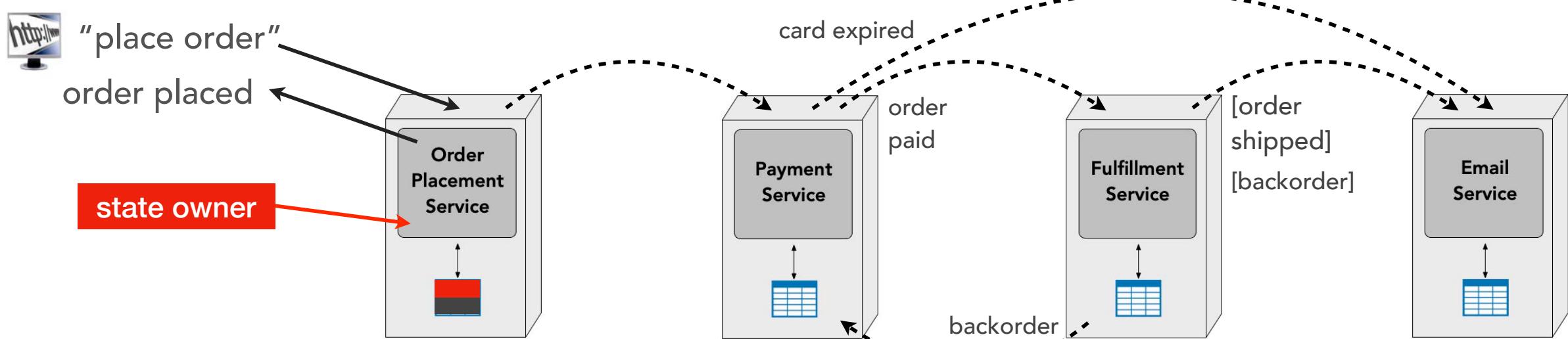
managing workflows

state owner



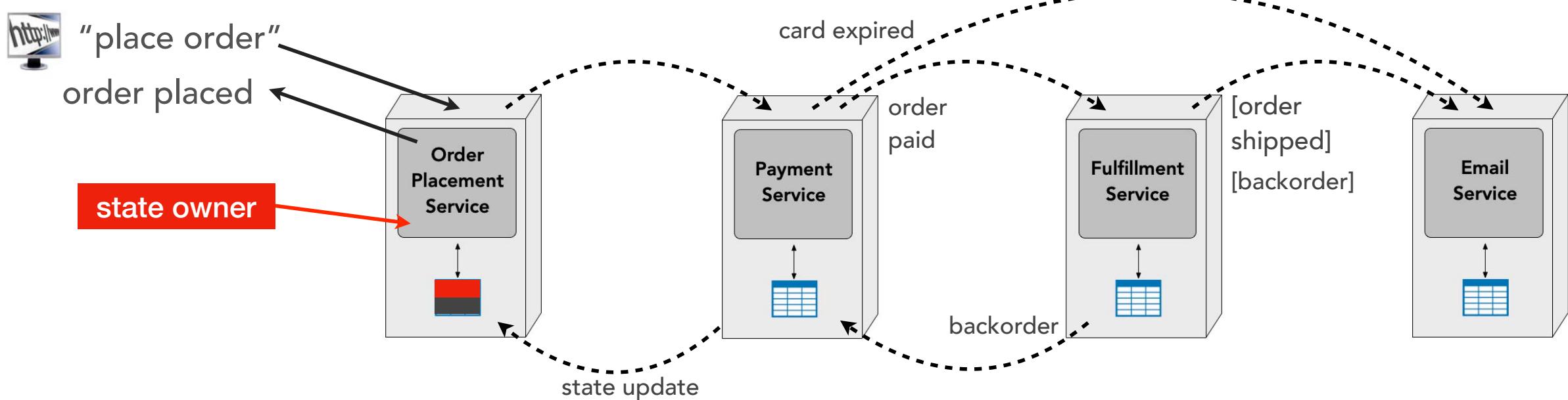
managing workflows

state owner



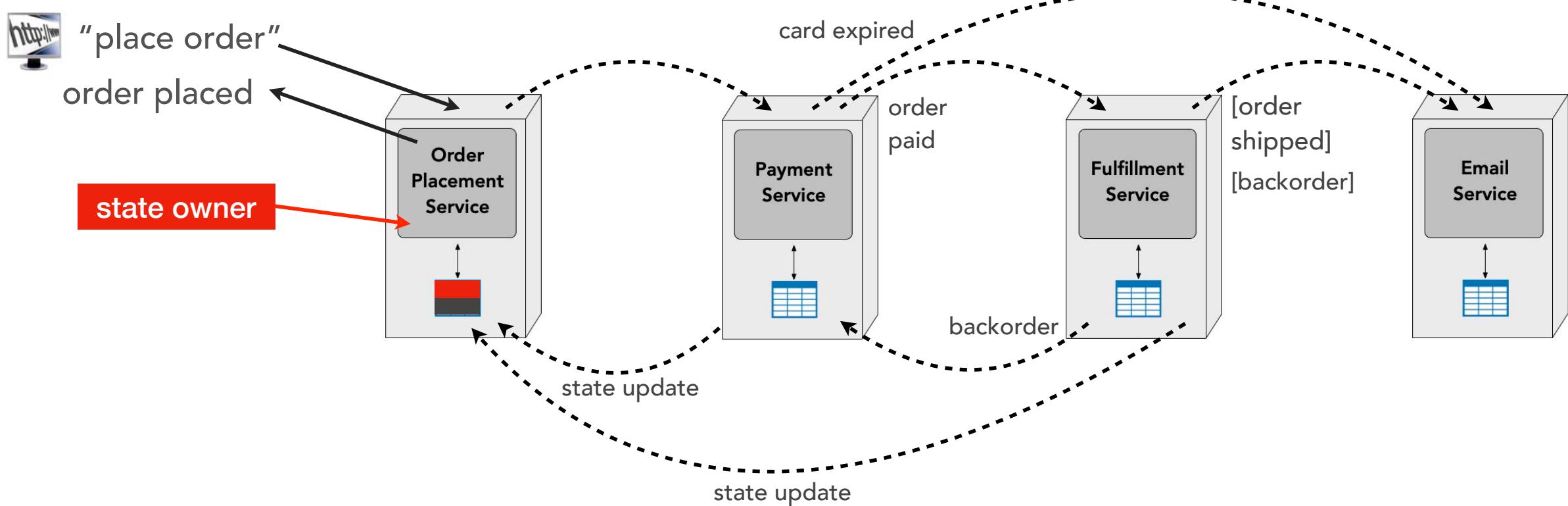
managing workflows

state owner



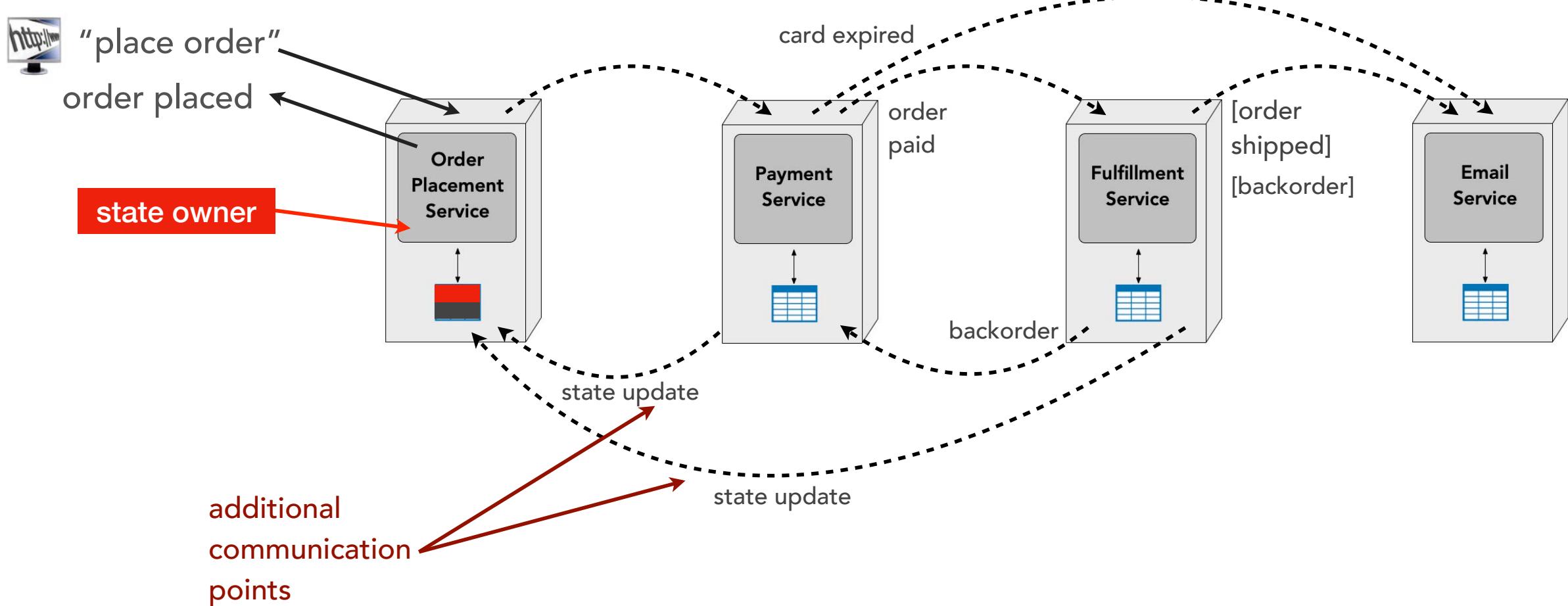
managing workflows

state owner



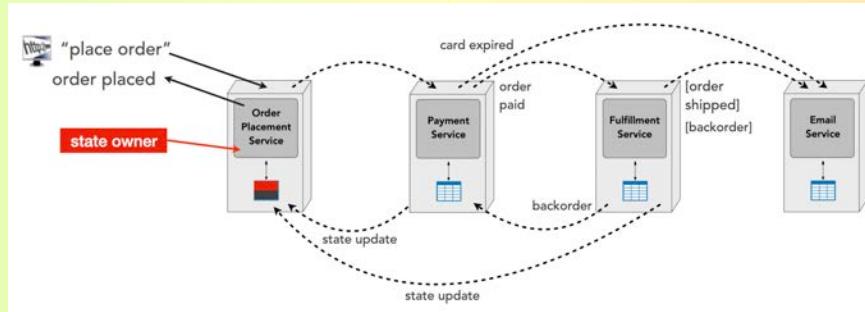
managing workflows

state owner



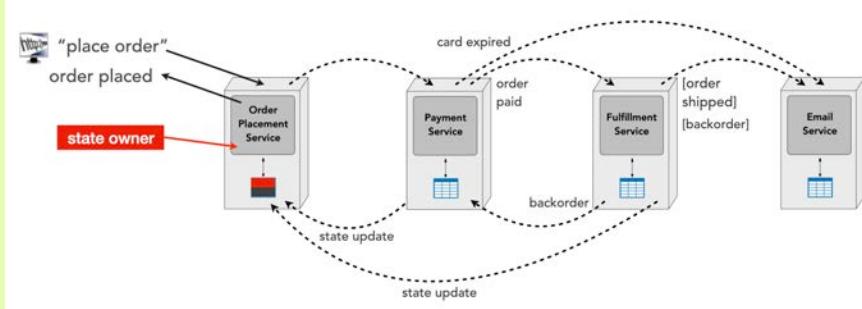
tradeoffs

workflow-based choreography



tradeoffs

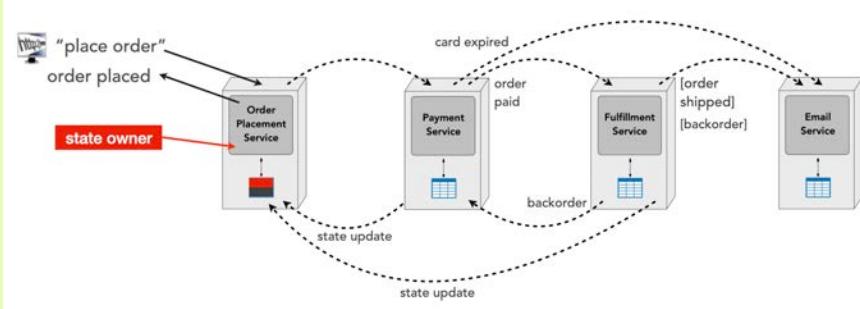
workflow-based choreography



- + responsiveness
- + scalability
- + fault tolerance
- + service decoupling

tradeoffs

workflow-based choreography



+ responsiveness

+ scalability

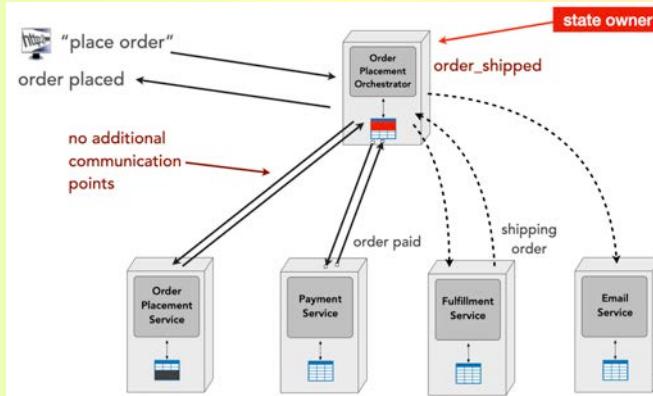
+ fault tolerance

+ service decoupling

- distributed workflow
- state management
- error handling
- recoverability

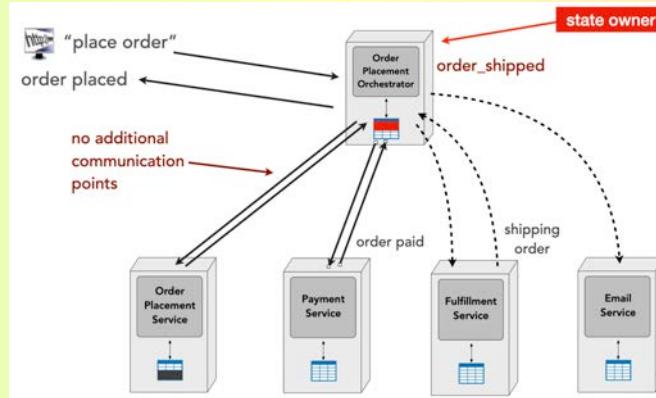
tradeoffs

workflow-based orchestration



tradeoffs

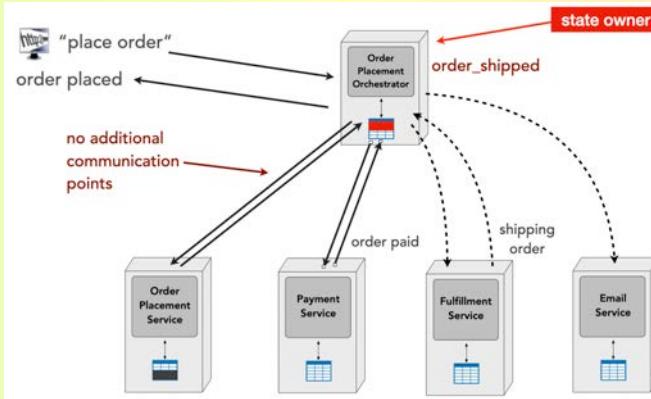
workflow-based orchestration



- + centralized workflow
- + error handling
- + recoverability
- + state management

tradeoffs

workflow-based orchestration

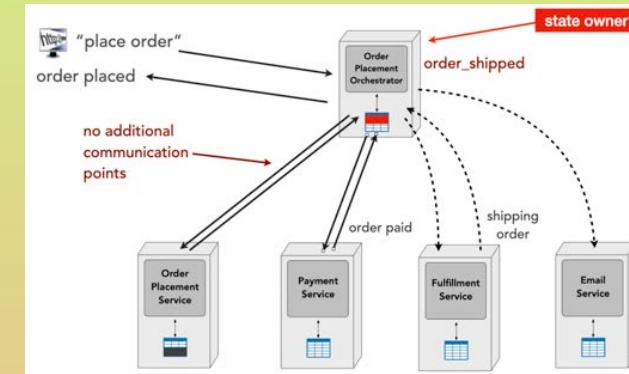
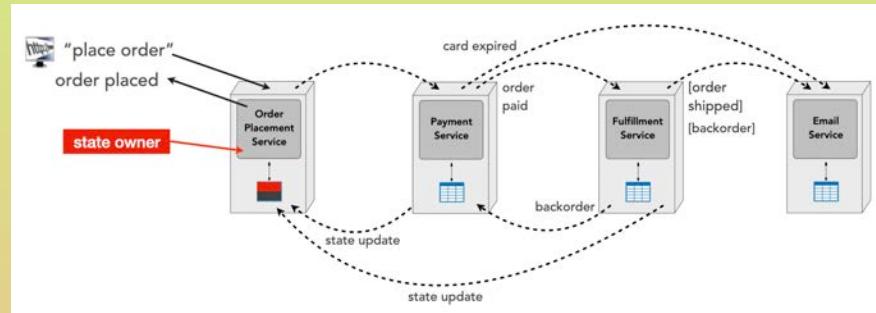


- + centralized workflow
- + error handling
- + recoverability
- + state management

- responsiveness
- fault tolerance
- scalability
- service coupling

tradeoffs

workflow-based choreography vs. orchestration

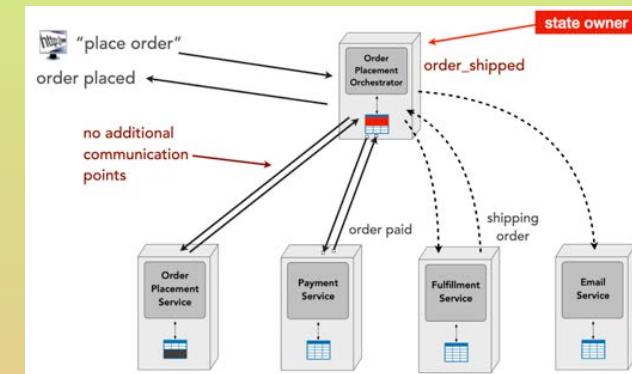
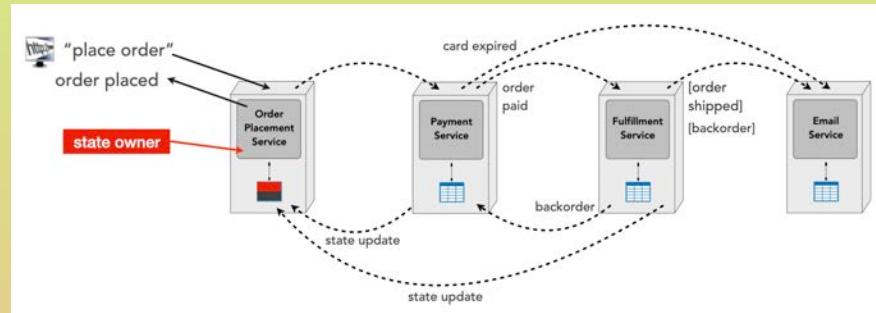


✓ responsiveness and scalability

✓ workflow control and error handling

tradeoffs

workflow-based choreography vs. orchestration



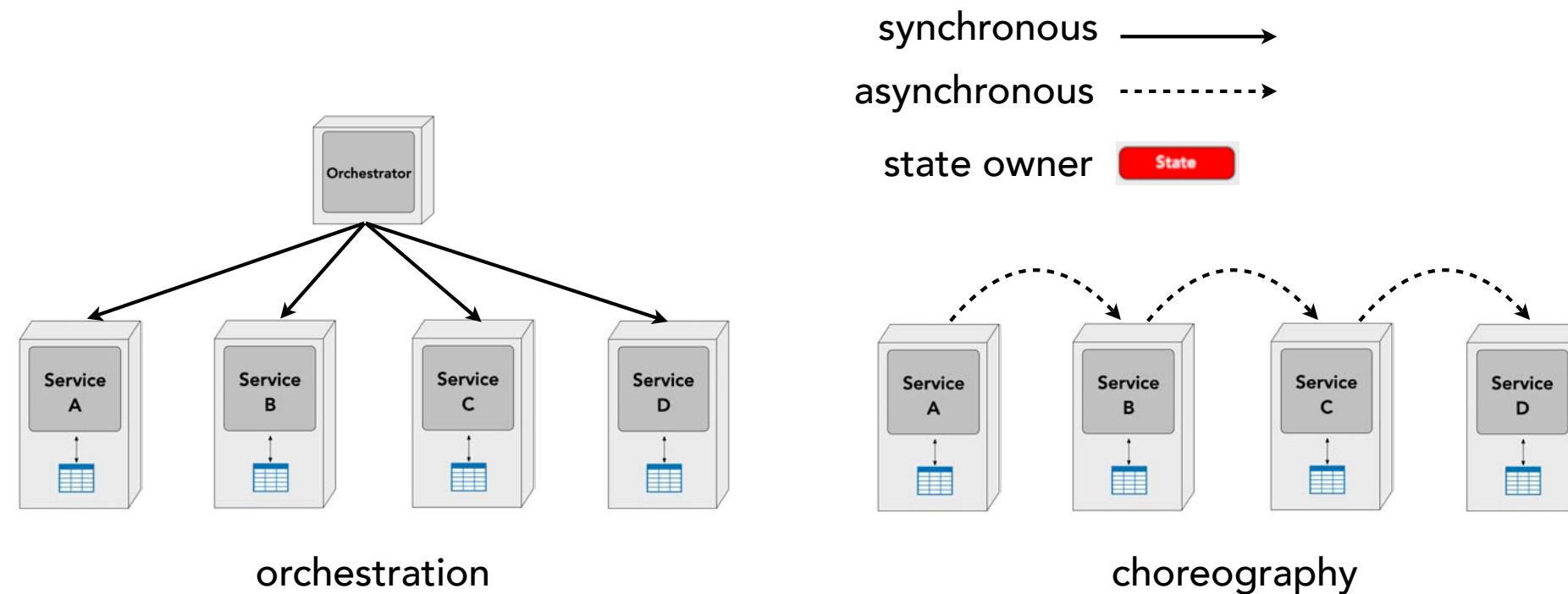
✓ responsiveness and scalability

✓ workflow control and error handling

which is more important?

Kata Exercise - Workflow and Communications

Now that you broke things apart, you now need to stitch them back together. Based on the workflows in the system, how should the services communicate with each other? Asynchronous or synchronous? Orchestration or choreography?





Contracts

Strict vs. Loose Contracts

contract considerations

strict vs. loose contracts

strict

loose



XML Schema

GraphQL

value-driven contracts

JSON Schema

simple JSON

Object

KVP arrays (maps)

RPC (including gRPC)

contract considerations

strict vs. loose contracts

strict

loose



{

```
"$schema": "http://json-schema.org/draft-04/schema#",
"properties": {
    "acct": {"type": "number"},
    "cusip": {"type": "string"},
    "shares": {"type": "number", "minimum": 100}
},
"required": ["acct", "cusip", "shares"]
}
```

contract considerations

strict vs. loose contracts

strict

loose



{

```
"$schema": "http://json-schema.org/draft-04/schema#",
"properties": {
    "acct": {"type": "number"},
    "cusip": {"type": "string"},
    "shares": {"type": "number", "minimum": 100}
},
"required": ["acct", "cusip", "shares"]
```

}

contract considerations

strict vs. loose contracts

strict

loose



customer wishlist

```
type Profile {  
    name: String  
}
```

customer profile

```
type Profile {  
    name: String  
    addr1: String  
    addr2: String  
    country: String  
    ...  
}
```

contract considerations

strict vs. loose contracts

strict

loose



customer wishlist

```
type Profile {  
    name: String  
}
```

customer profile

```
type Profile {  
    name: String  
    addr1: String  
    addr2: String  
    country: String  
    ...  
}
```

contract considerations

strict vs. loose contracts

strict

loose

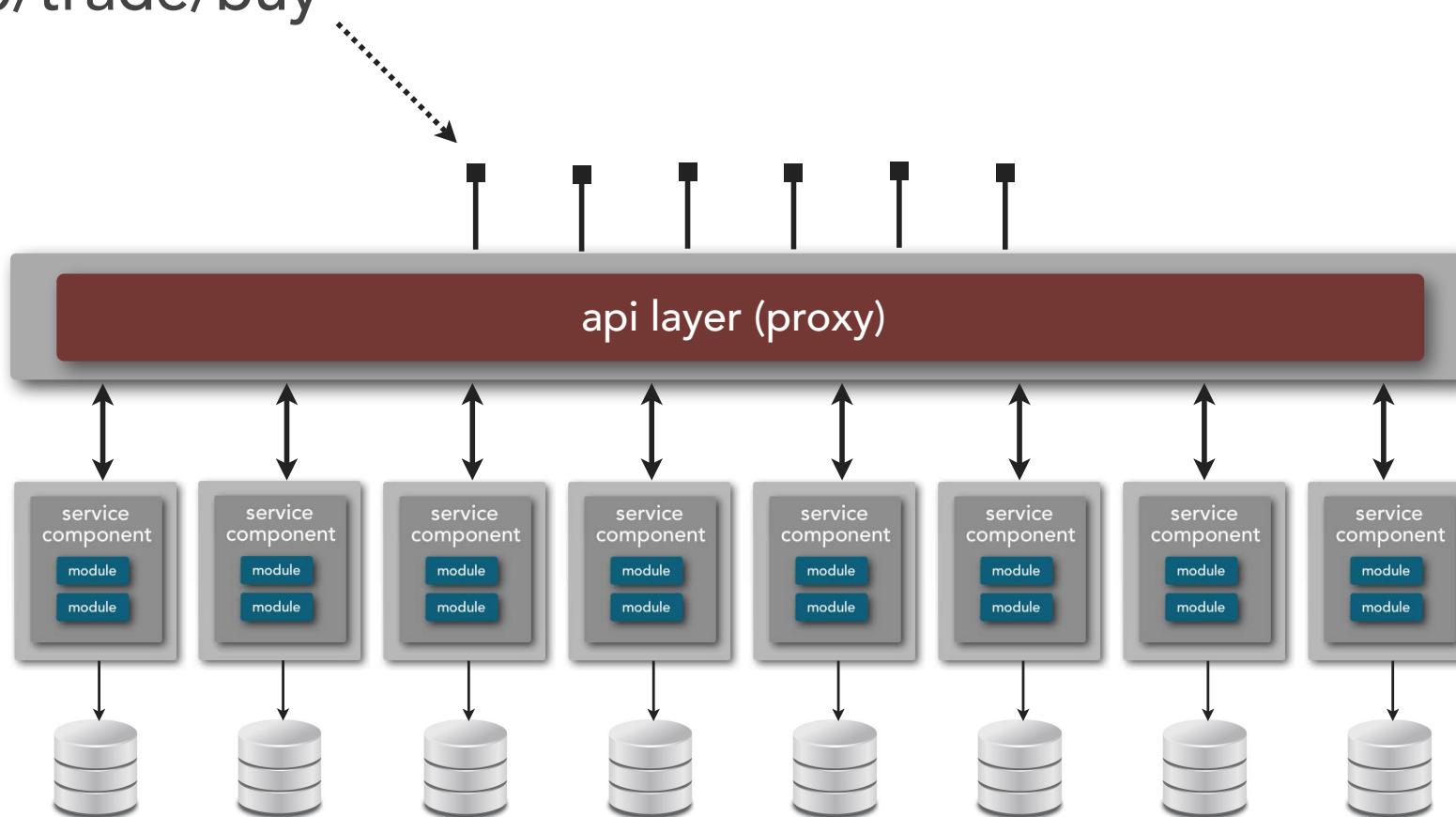


```
{  
  "name": "Mark",  
  "status": "active",  
  "joined": "2003"  
}
```

Contract Versioning

contract versioning

POST app/1.0/trade/buy



contract versioning

Apple Common Stock

- ✓ Symbol: **AAPL**
- ✓ CUSIP (Committee on Uniform Security Identification Procedures): **037833100**
- ✓ SEDOL (Stock Exchange Daily Official List): **2046251**

AAPL = 037833100 = 2046251

contract versioning

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "properties": {  
    "acct": {"type": "number"},  
    "cusip": {"type": "string"},  
    "shares": {"type": "number", "minimum": 100}  
  },  
  "required": ["acct", "cusip", "shares"]  
}
```

contract versioning

consumer 1



```
POST /app/1.0/trade/buy
Accept: application/json
{ "acct": "12345",
  "cusip": "037833100",
  "shares": "1000" }
```

consumer 2



```
POST /app/1.0/trade/buy
Accept: application/json
{ "acct": "76512",
  "cusip": "037833100",
  "shares": "200" }
```

contract versioning

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "properties": {  
    "acct": {"type": "number"},  
    → "cusip": {"type": "string"},  
    "shares": {"type": "number", "minimum": 100}  
  },  
  "required": ["acct", "cusip", "shares"]  
}
```

contract versioning

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "properties": {  
    "acct": {"type": "number"},  
    "sedol": {"type": "string"},  
    "shares": {"type": "number", "minimum": 100}  
  },  
  "required": ["acct", "sedol", "shares"]  
}
```

contract versioning

consumer 1



```
POST /app/1.0/trade/buy  
Accept: application/json  
{ "acct": "12345",  
  "sedol": "2046251",  
  "shares": "1000" }
```

consumer 2

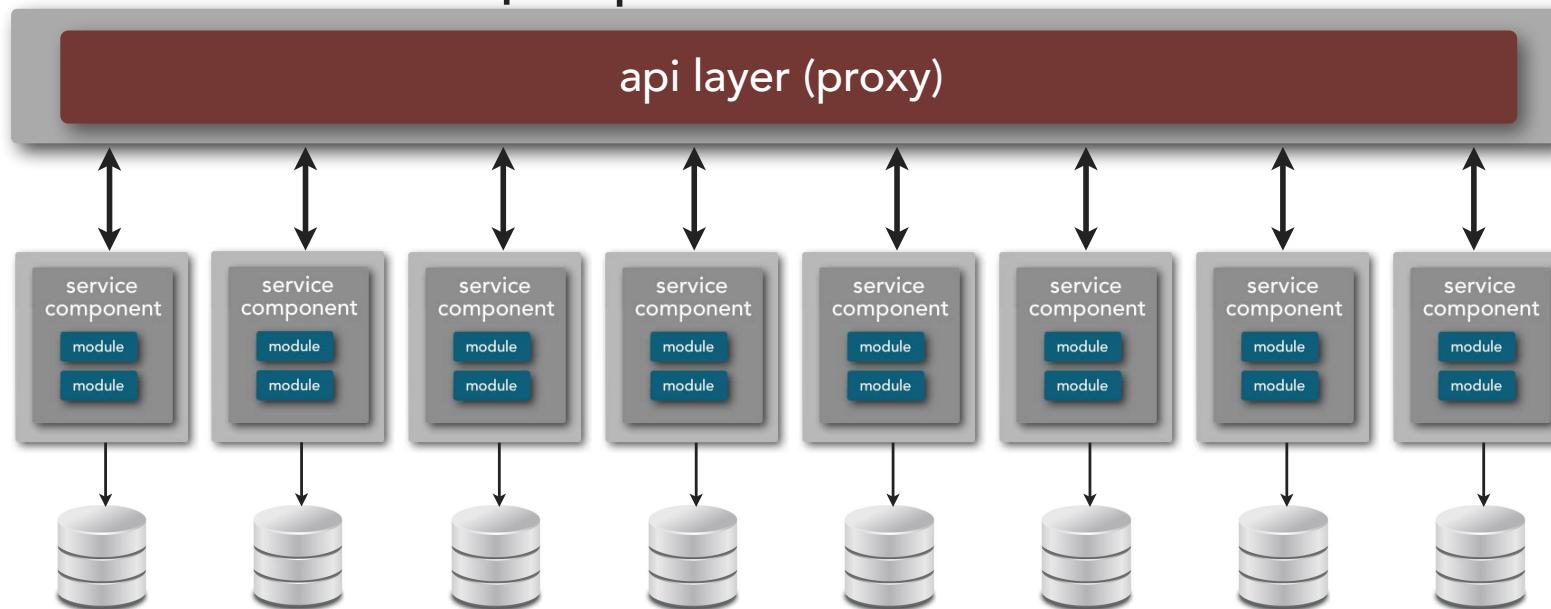


```
POST /app/1.0/trade/buy  
Accept: application/json  
{ "acct": "76512",  
  "cusip": "037833100",  
  "shares": "200" }
```

contract versioning

POST app/1.0/trade/buy
(CUSIP)

POST app/1.1/trade/buy
(SEDOL)



contract versioning

apply versioning at the header level

Accept: application/vnd.service.trade.v2+json



vendor mime type

version 1

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "properties": {  
    "acct": {"type": "number"},  
    "cusip": {"type": "string"},  
    "shares": {"type": "number", "minimum": 100}  
  },  
  "required": ["acct", "cusip", "shares"]  
}
```

version 2

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "properties": {  
    "acct": {"type": "number"},  
    "sedol": {"type": "string"},  
    "shares": {"type": "number", "minimum": 100}  
  },  
  "required": ["acct", "sedol", "shares"]  
}
```

contract versioning

consumer 1



```
POST /app/1.0/trade/buy
Accept: application/vnd.service.trade.v2+json
{ "acct": "12345",
  "sedol": "2046251",
  "shares": "1000" }
```

consumer 2



```
POST /app/1.0/trade/buy
Accept: application/vnd.service.trade.v1+json
{ "acct": "76512",
  "cusip": "037833100",
  "shares": "200" }
```

contract versioning

consumer 1



```
POST /app/1.0/trade/buy
Accept: application/vnd.service.trade+json; version=2
{ "acct": "12345",
  "sedol": "2046251",
  "shares": "1000" }
```

consumer 2



```
POST /app/1.0/trade/buy
Accept: application/vnd.service.trade+json; version=1
{ "acct": "76512",
  "cusip": "037833100",
  "shares": "200" }
```

contract versioning

```
POST /app/1.0/trade/buy
Accept: application/vnd.service.trade.v2+json
{ "acct": "12345",
  "sedol": "2046251",
  "shares": "1000" }
```

```
def version
  request.headers["Accept"][/^application/vnd.service.trade.v(d)/, 1].to_i
end
```

contract versioning

```
POST /app/1.0/trade/buy
Accept: application/vnd.service.trade.v2+json
{ "acct": "12345",
  "sedol": "2046251",
  "shares": "1000" }
```

```
if version == 1 then
  def.version
else if request.headers["Accept"][/^application/vnd.service.trade.v(d)/, 1].to_i
  end..
else if version == 3 then
  ....
else if version == 4 then
  ....
...
...
```

contract version

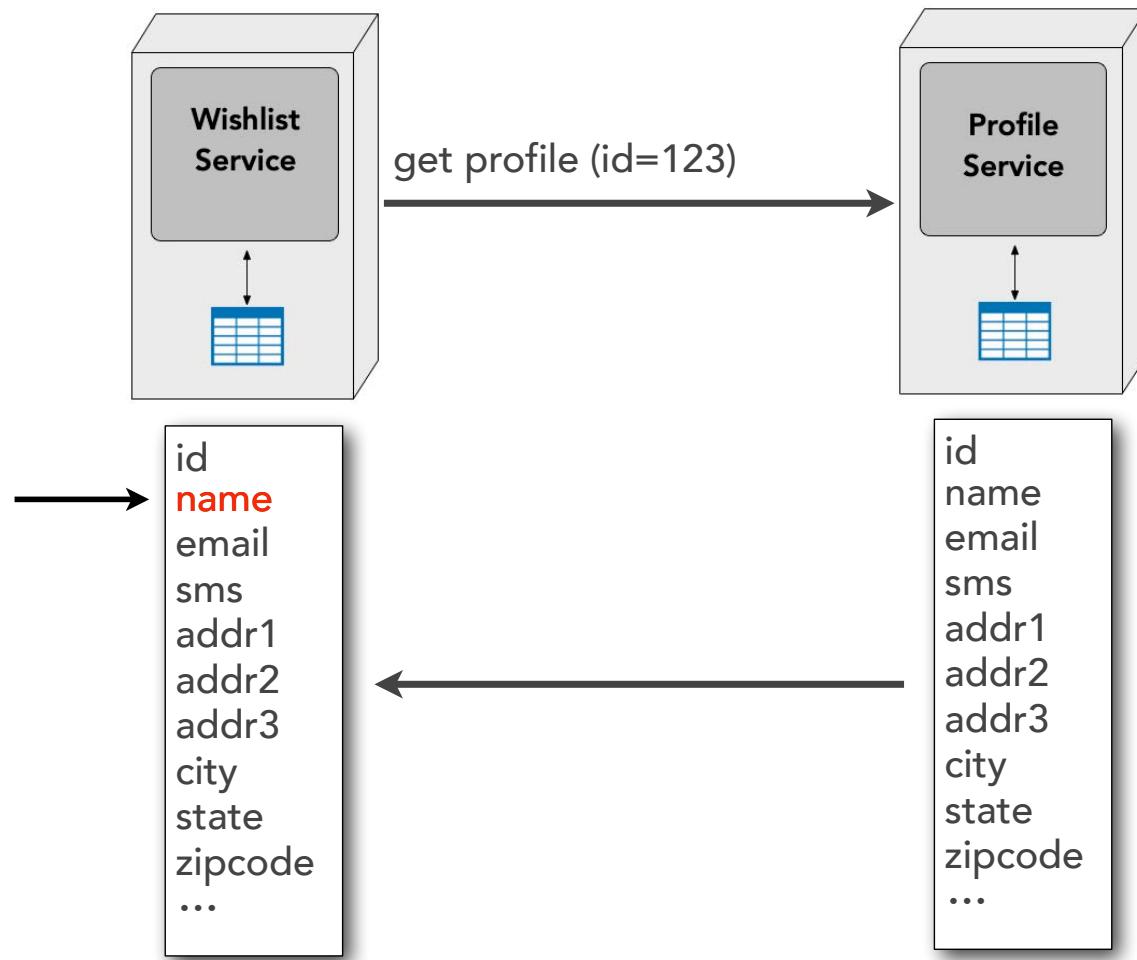
deprecation strategy



Stamp Coupling and Bandwidth Issues

stamp coupling and bandwidth

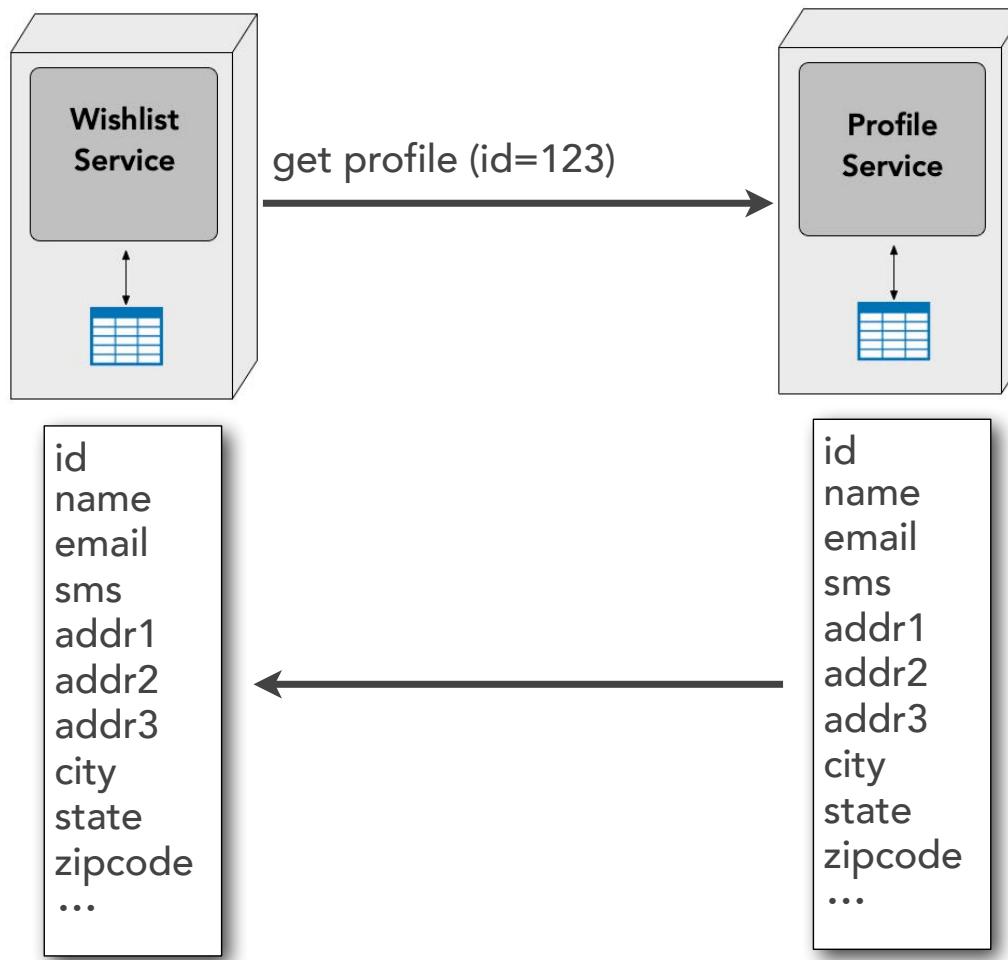
stamp coupling



{
 "\$schema": "http://json-schema...",
 "properties": {
 "id": {"type": "string"},
 "name": {"type": "string"},
 "email": {"type": "string"},
 "sms": {"type": "number"},
 "addr1": {"type": "string"},
 "addr2": {"type": "string"},
 "addr3": {"type": "string"},
 "city": {"type": "string"},
 "state": {"type": "string"},
 "zipcode": {"type": "number"},
 ...
 },
}

stamp coupling and bandwidth

stamp coupling

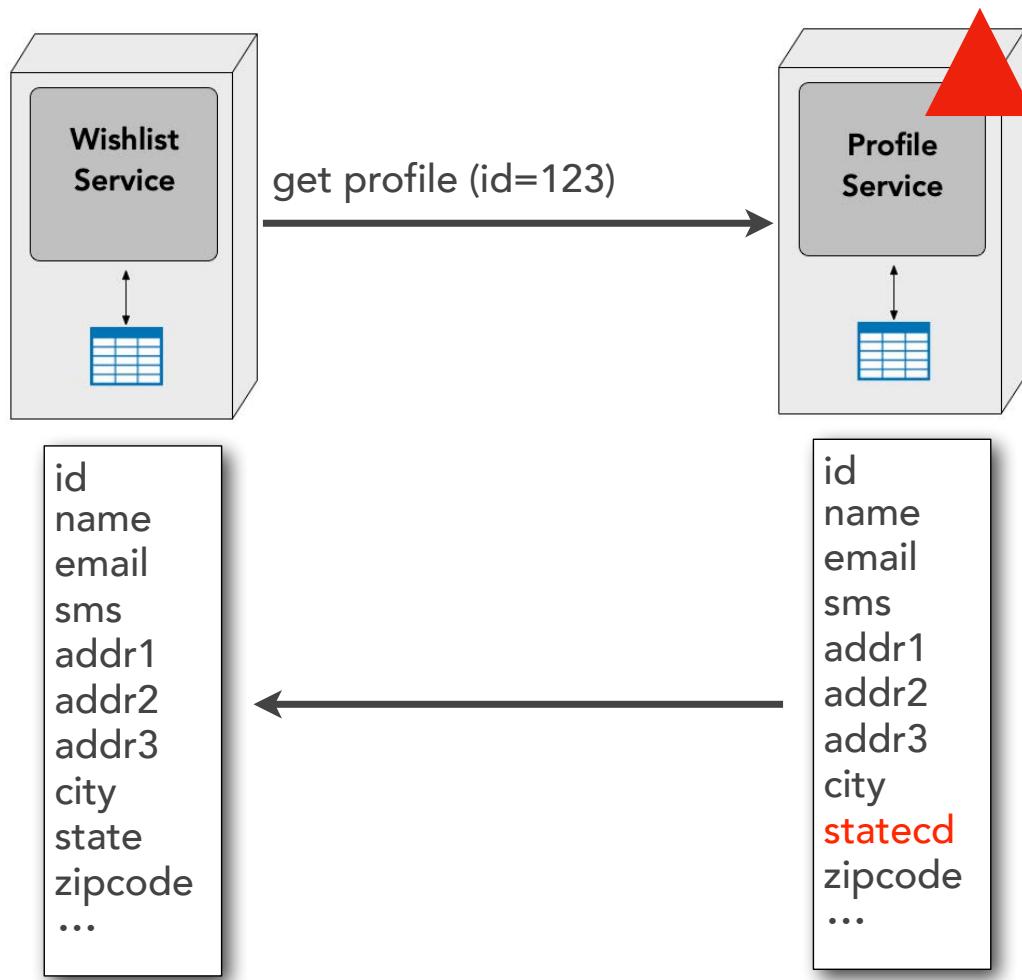


customer profile

```
{  
  "$schema": "http://json-schema...",  
  "properties": {  
    "id": {"type": "string"},  
    "name": {"type": "string"},  
    "email": {"type": "string"},  
    "sms": {"type": "number"},  
    "addr1": {"type": "string"},  
    "addr2": {"type": "string"},  
    "addr3": {"type": "string"},  
    "city": {"type": "string"},  
    "state": {"type": "string"},  
    "zipcode": {"type": "number"},  
    ...  
  },  
}
```

stamp coupling and bandwidth

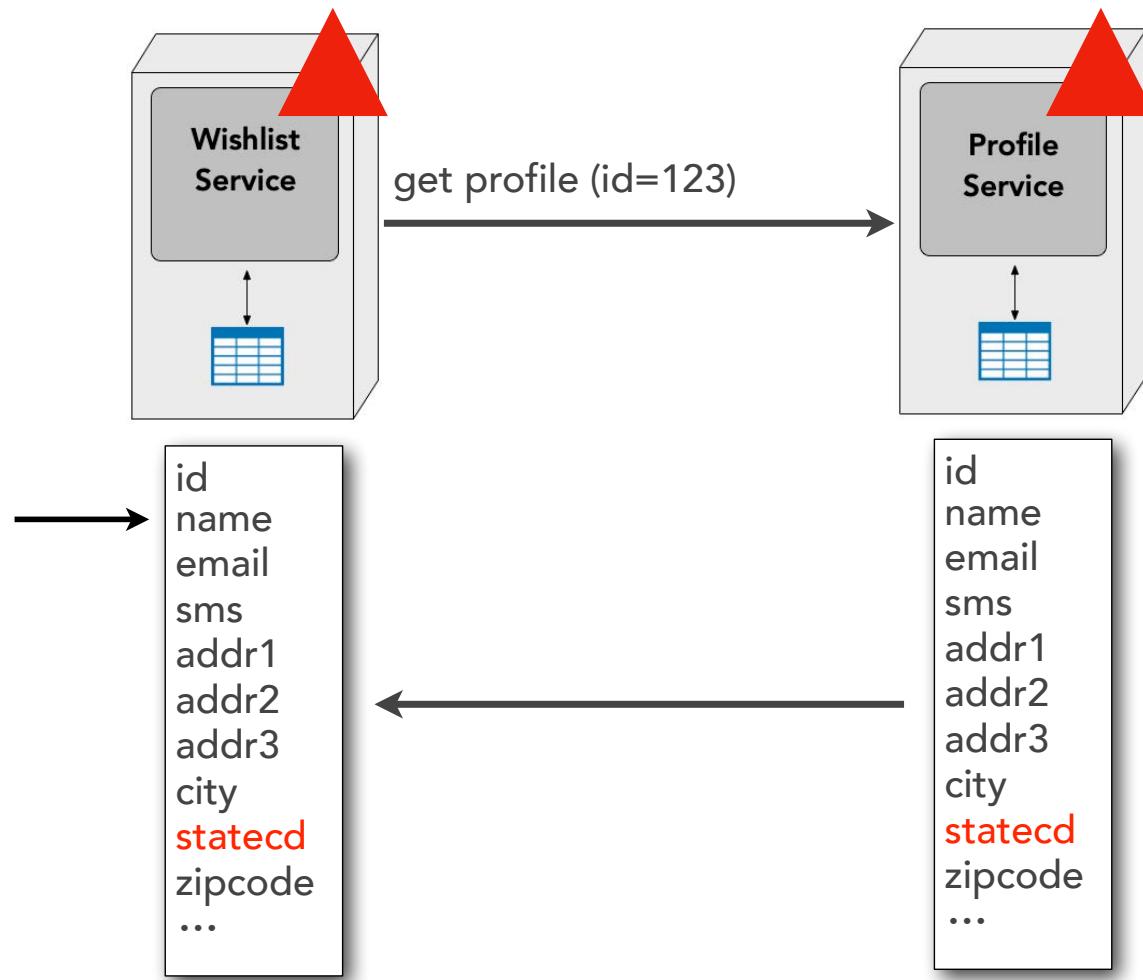
stamp coupling



customer profile

stamp coupling and bandwidth

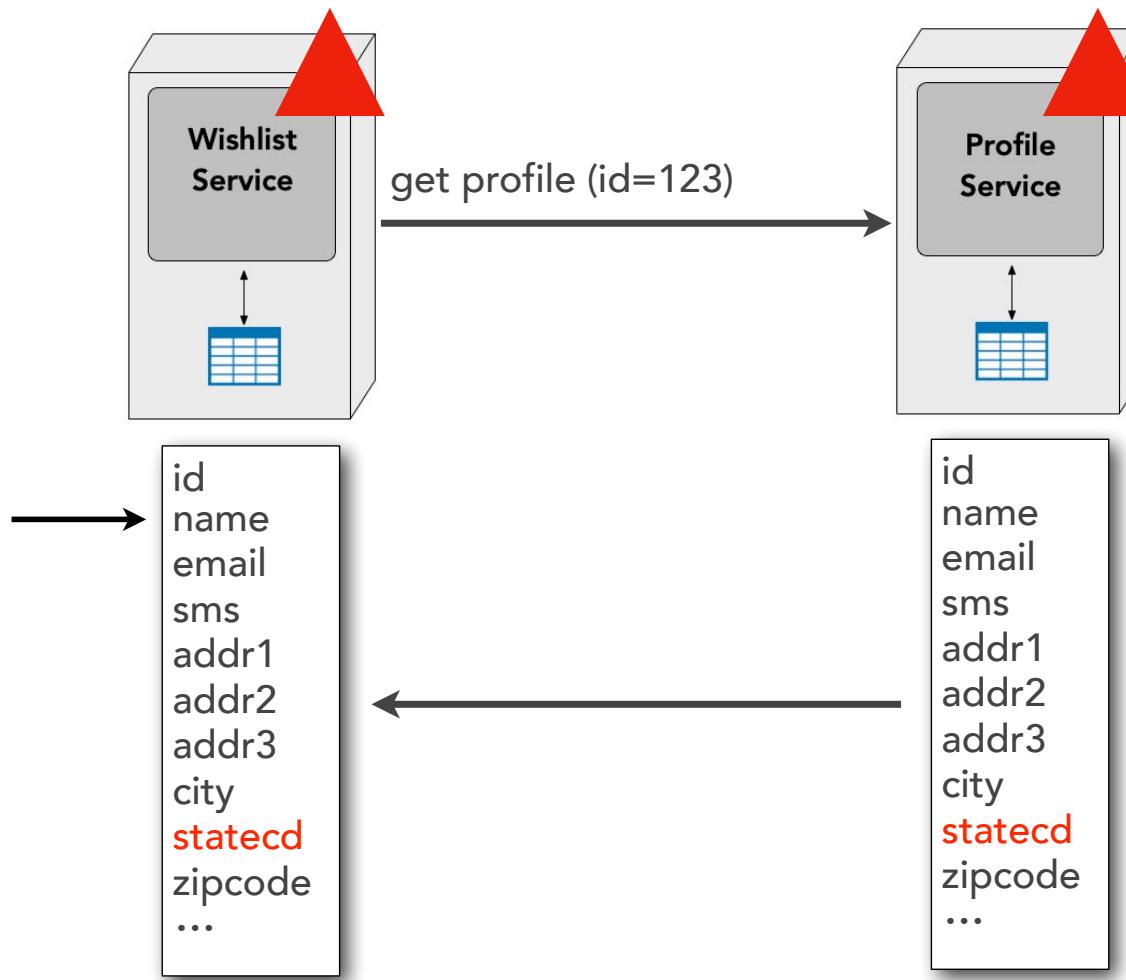
stamp coupling



customer profile

stamp coupling and bandwidth

stamp coupling



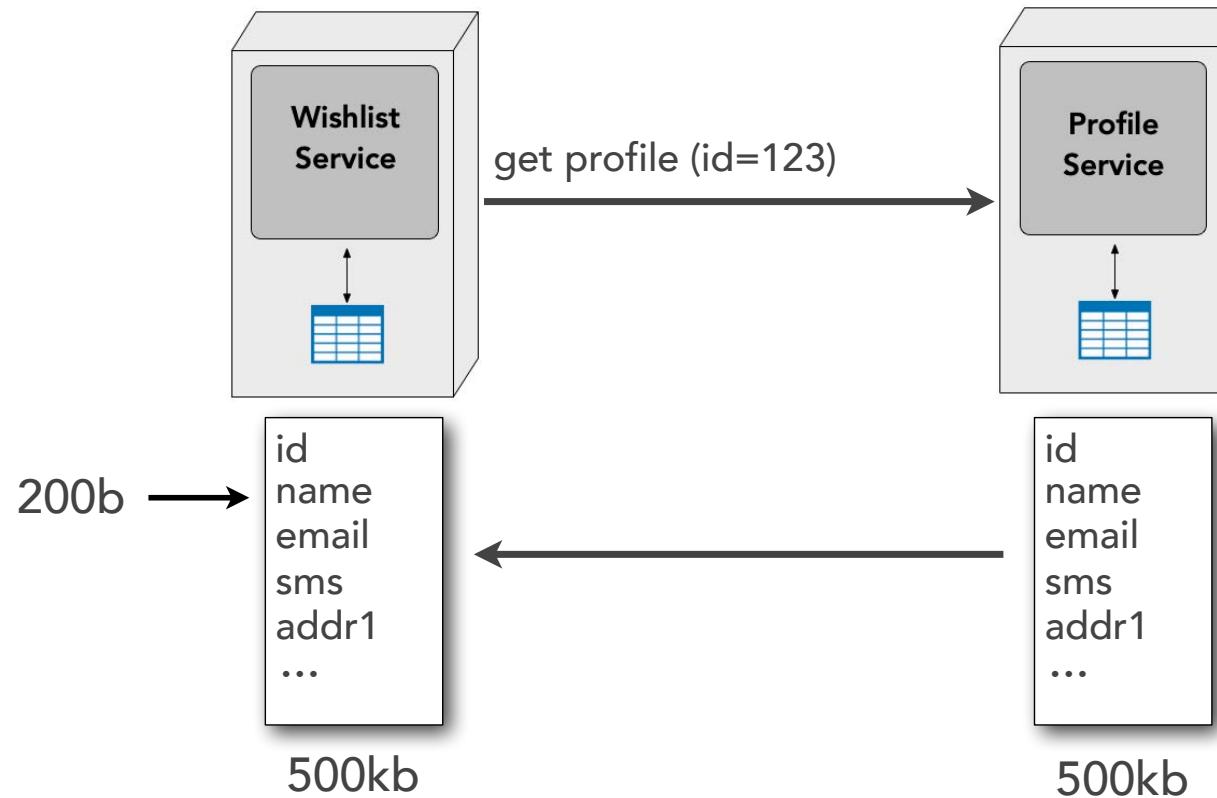
customer profile

```
{  
  "$schema": "http://json-schema...",  
  "properties": {  
    "id": {"type": "string"},  
    "name": {"type": "string"},  
    "email": {"type": "string"},  
    "sms": {"type": "number"},  
    "addr1": {"type": "string"},  
    "addr2": {"type": "string"},  
    "addr3": {"type": "string"},  
    "city": {"type": "string"},  
    "statecd": {"type": "string"},  
    "zipcode": {"type": "number"},  
    ...  
  },  
}
```

- ✓ use value-driven contracts (loose contracts)
- ✓ manage changes through fitness functions

stamp coupling and bandwidth

bandwidth issues



2000/sec

500kb



1,000,000kb/sec

2000/sec

200b



400kb/sec

- ✓ return partial data (requires logic in profile service)
- ✓ use GraphQL (requires GraphQL server)
- ✓ create a new endpoint with a different contract

tradeoffs

tight

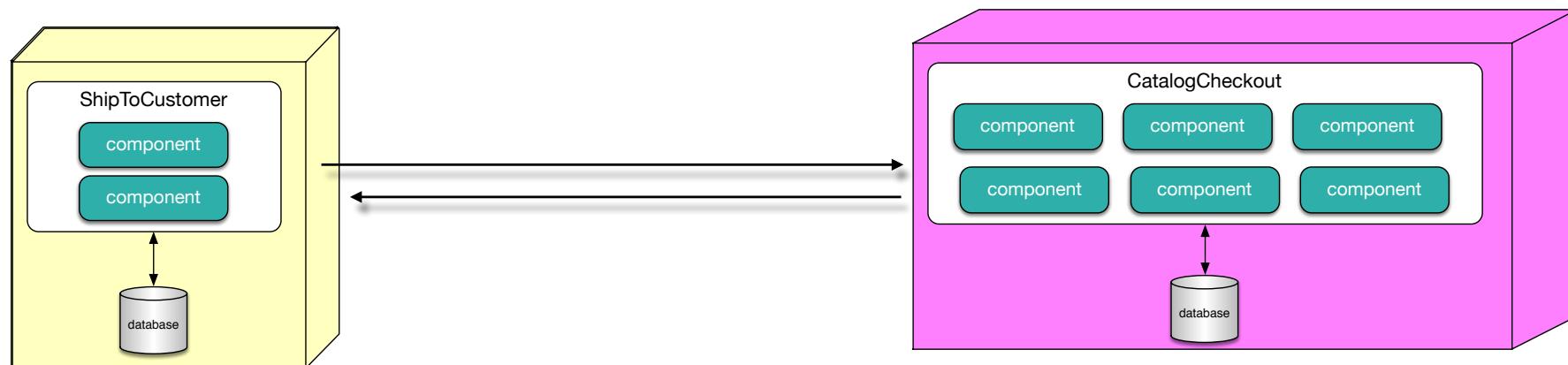
loose

- * guaranteed contract fidelity
- * distinct versions (+ or -?)
- * build-time contracts

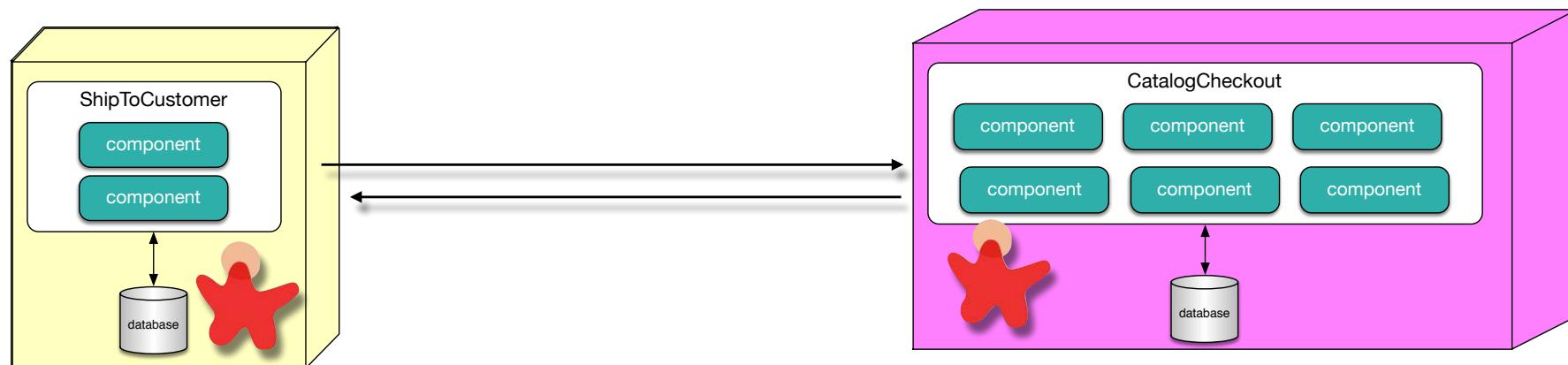
- * decoupled...
- * ...therefore better for decoupled architectures
- * contract management



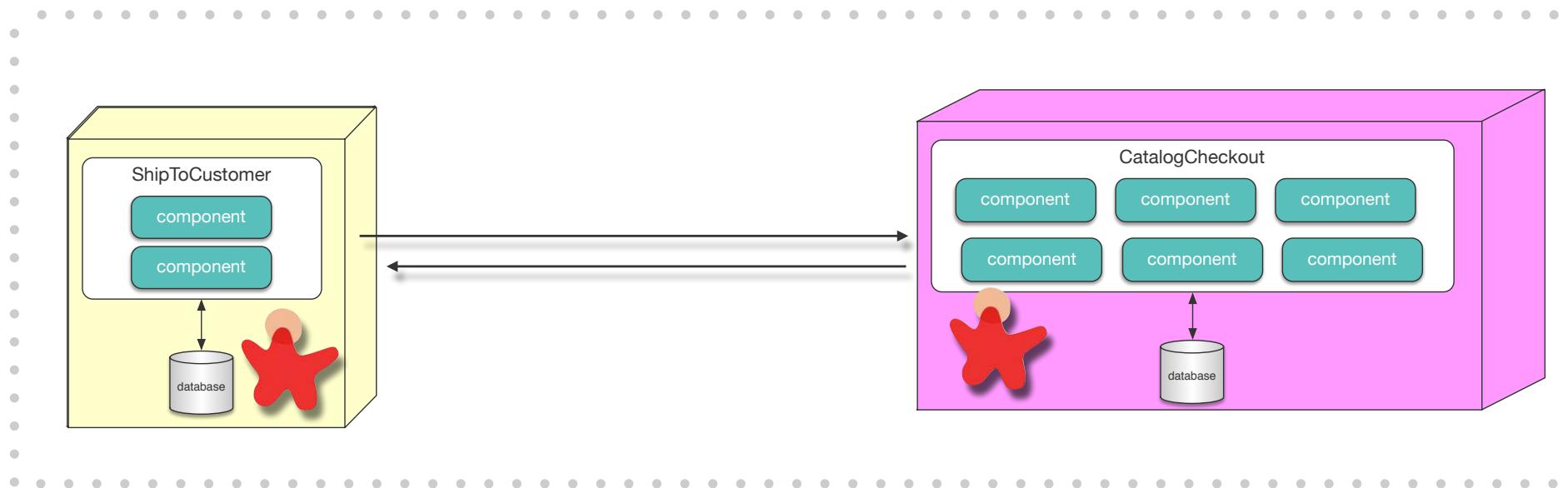
contracts in microservices



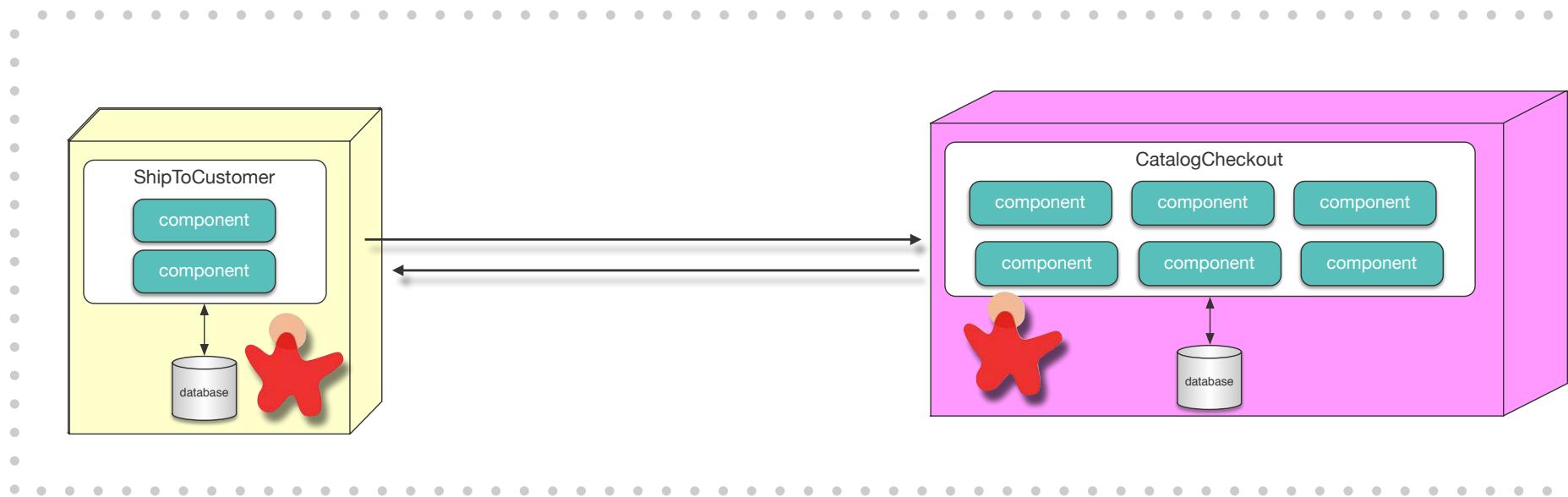
contracts in microservices



contracts in microservices



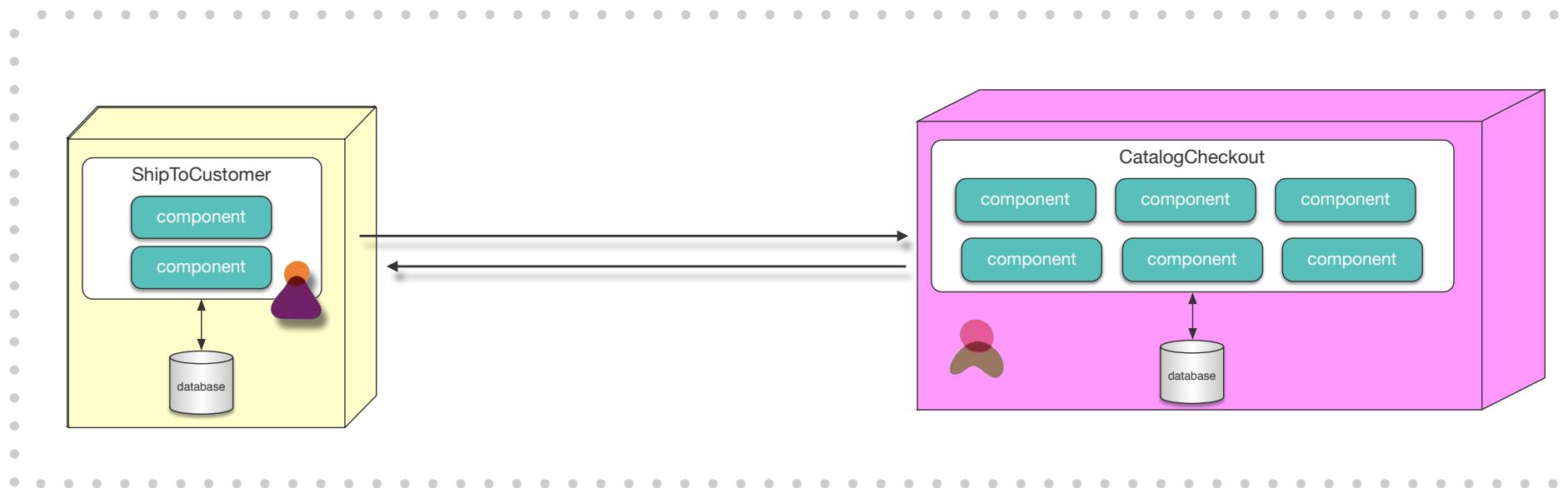
contracts in microservices



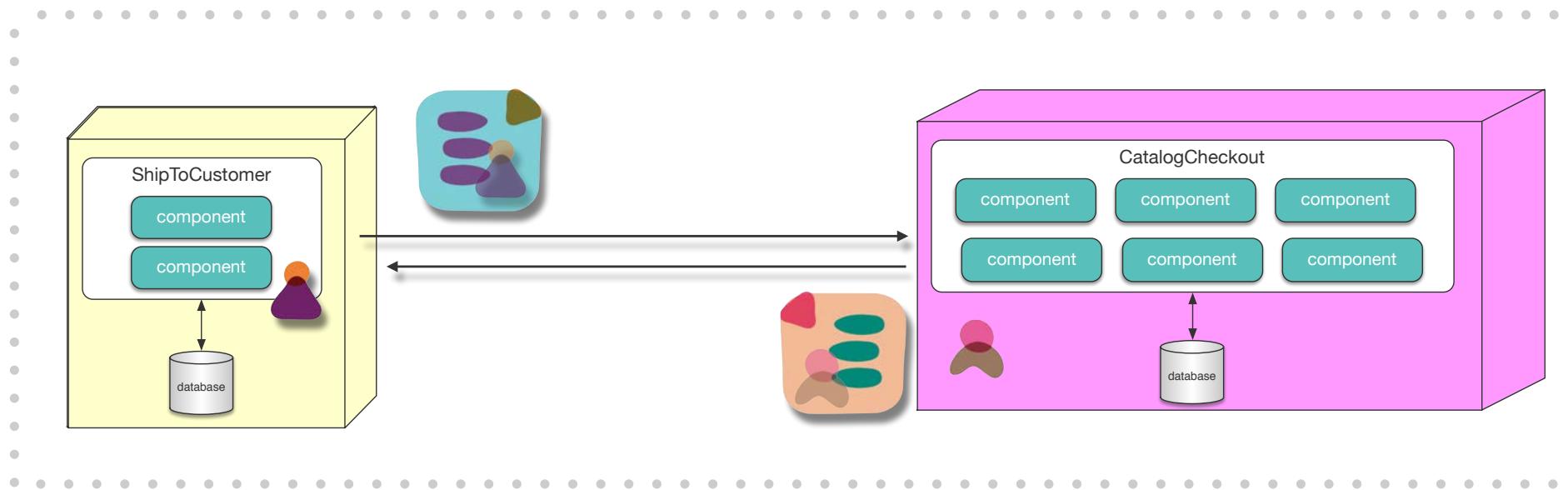
tight

loose

contracts in microservices

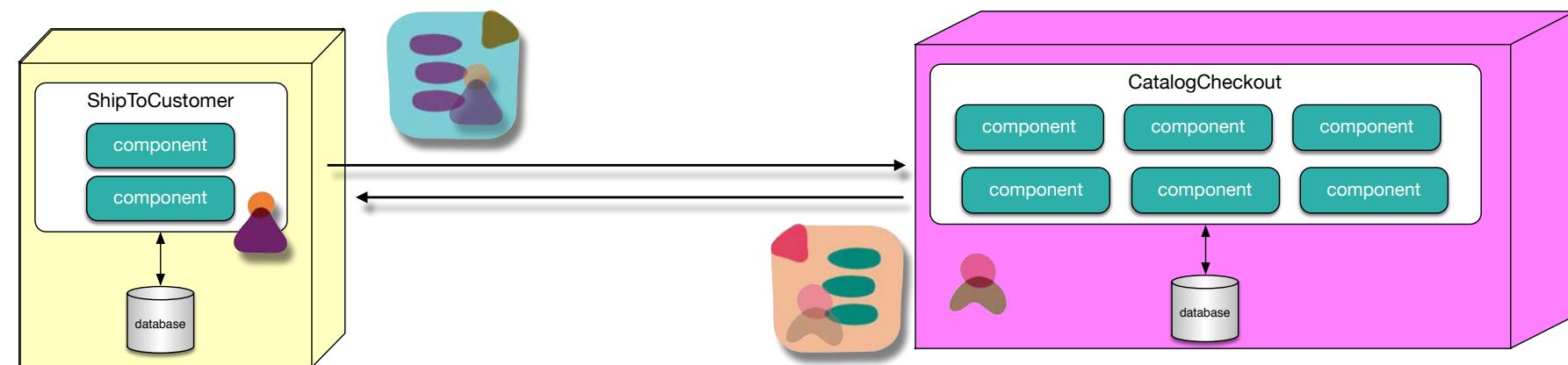


contracts in microservices



contracts in microservices

Transfer values, not types.

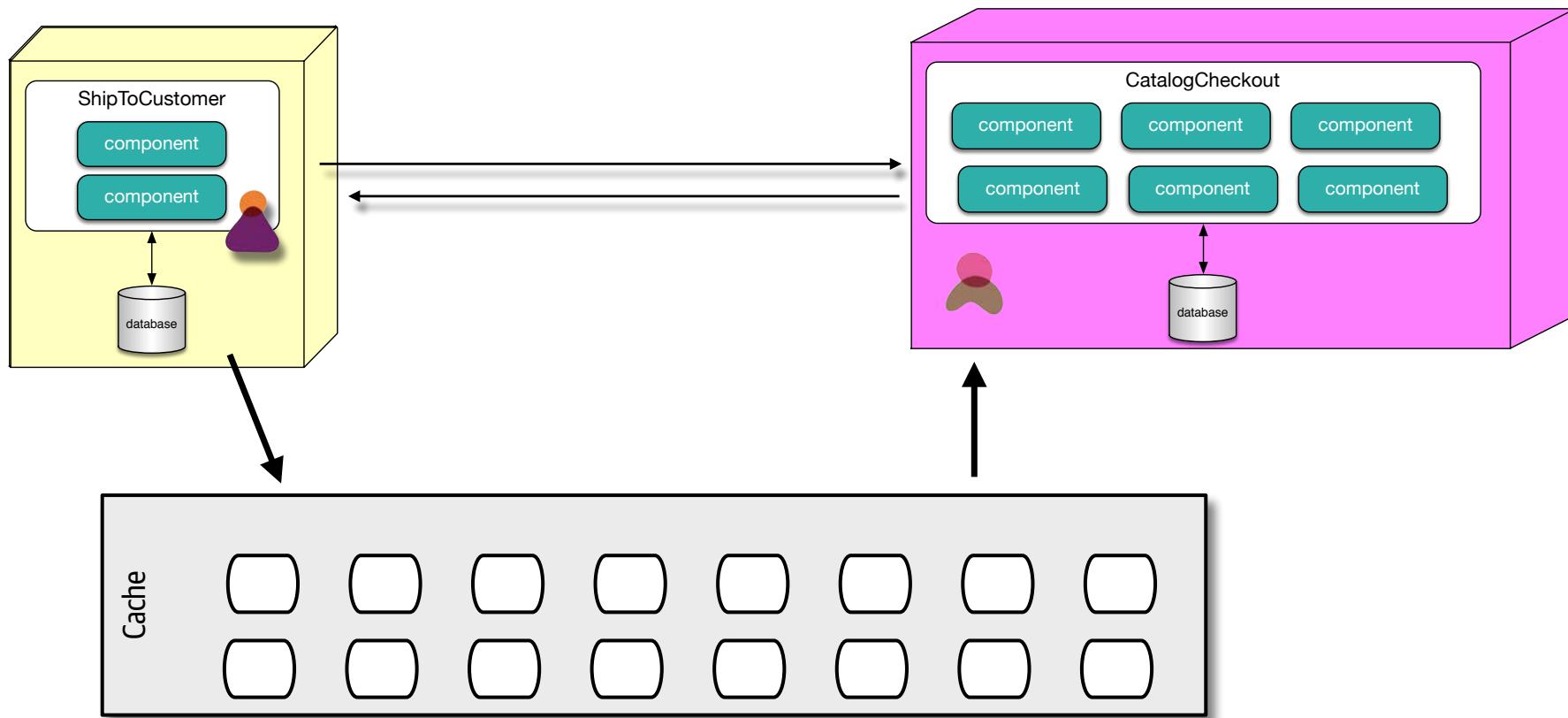


tight

loose

contracts in microservices

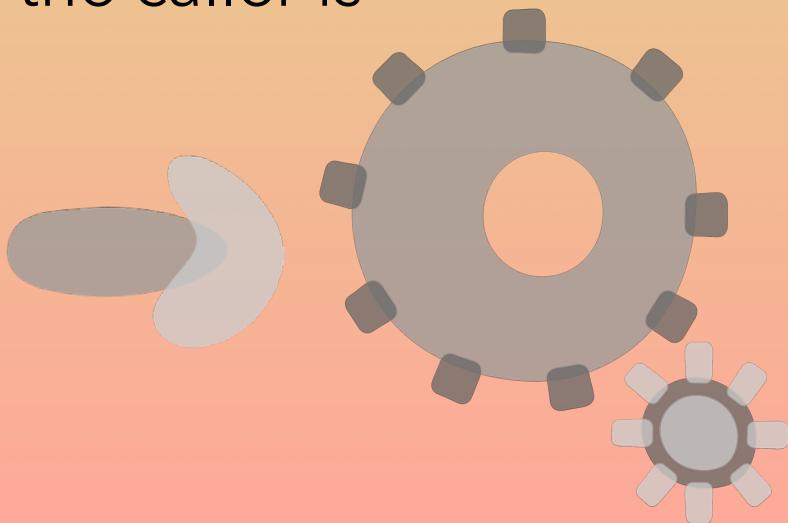
Transfer values, not types.



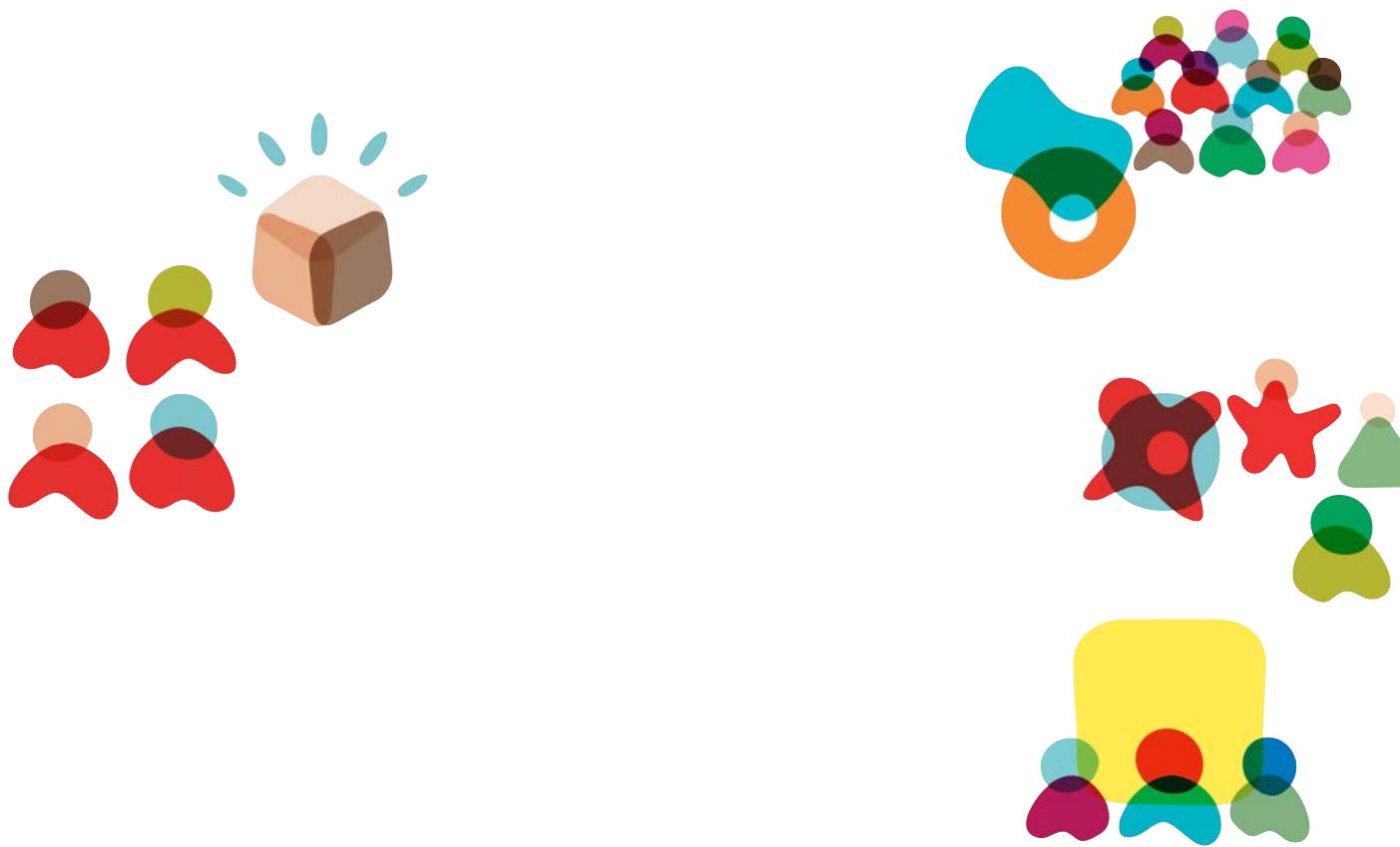
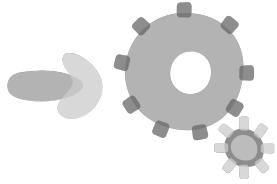
tradeoffs

value-based contracts

- the “contract” part of contract
 - + more malleable integration points
 - + easier to evolve
 - + decouples integration from implementation platform
- must know who the caller is

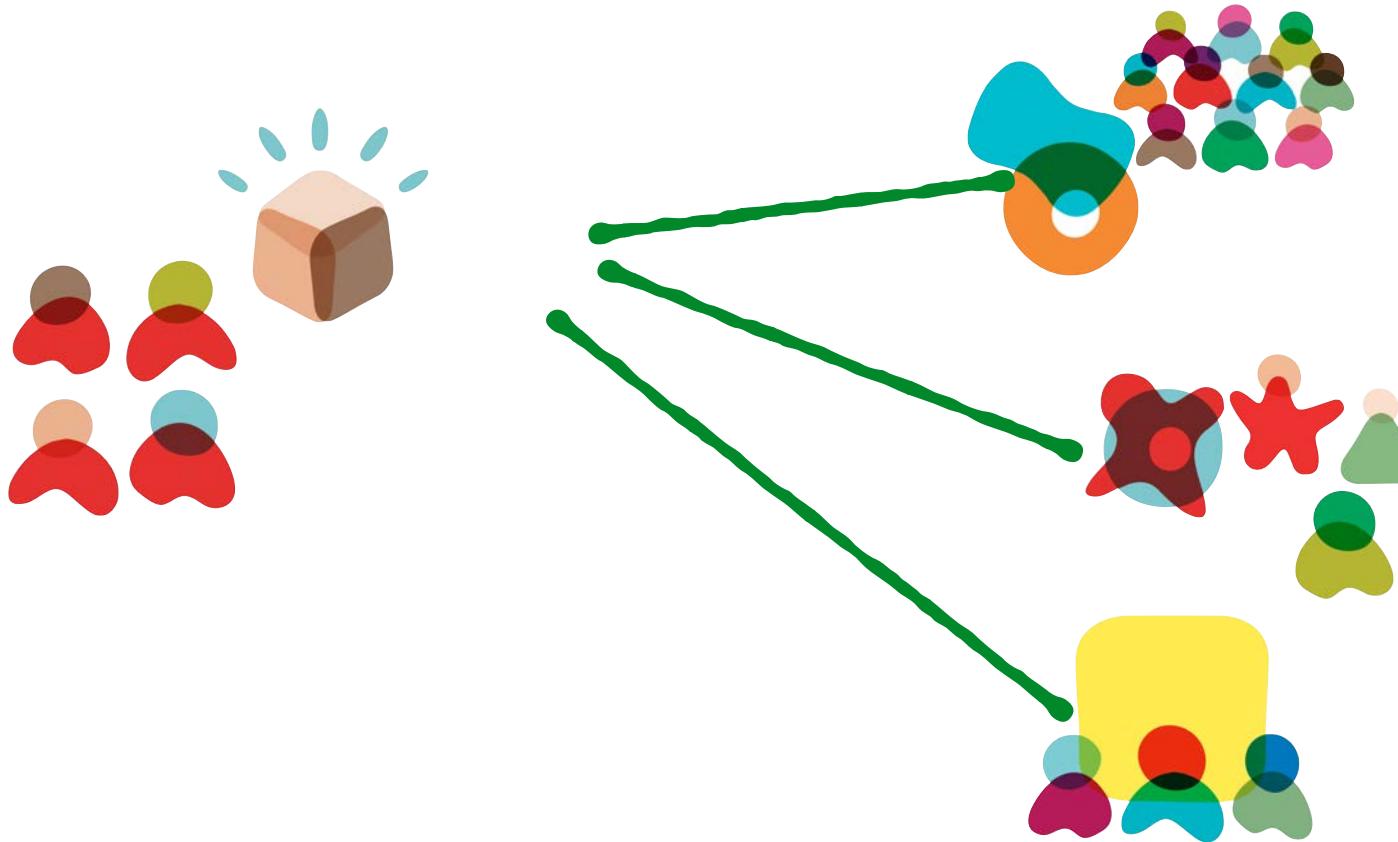
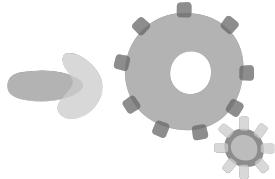


contract fitness function



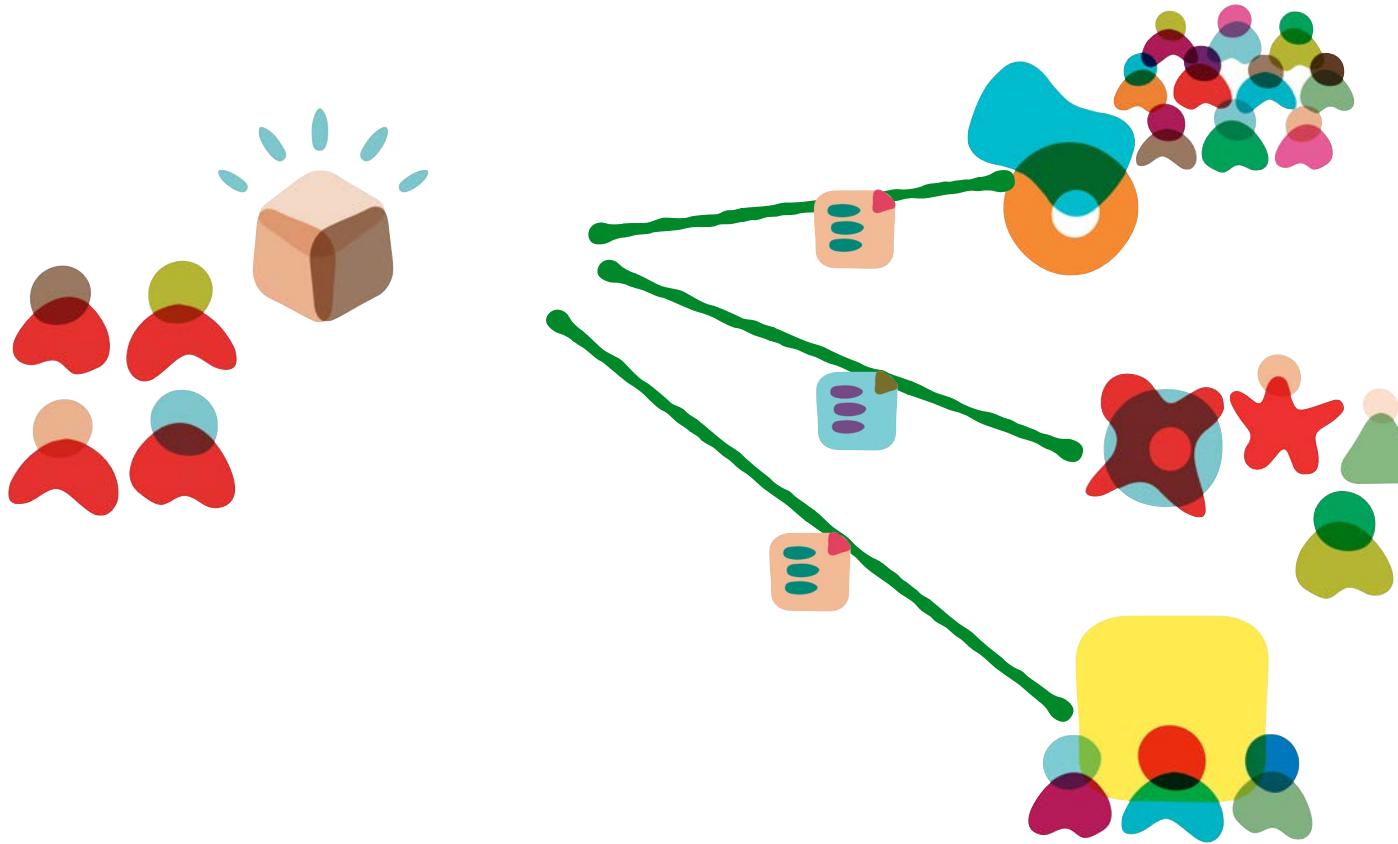
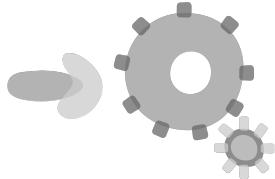
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



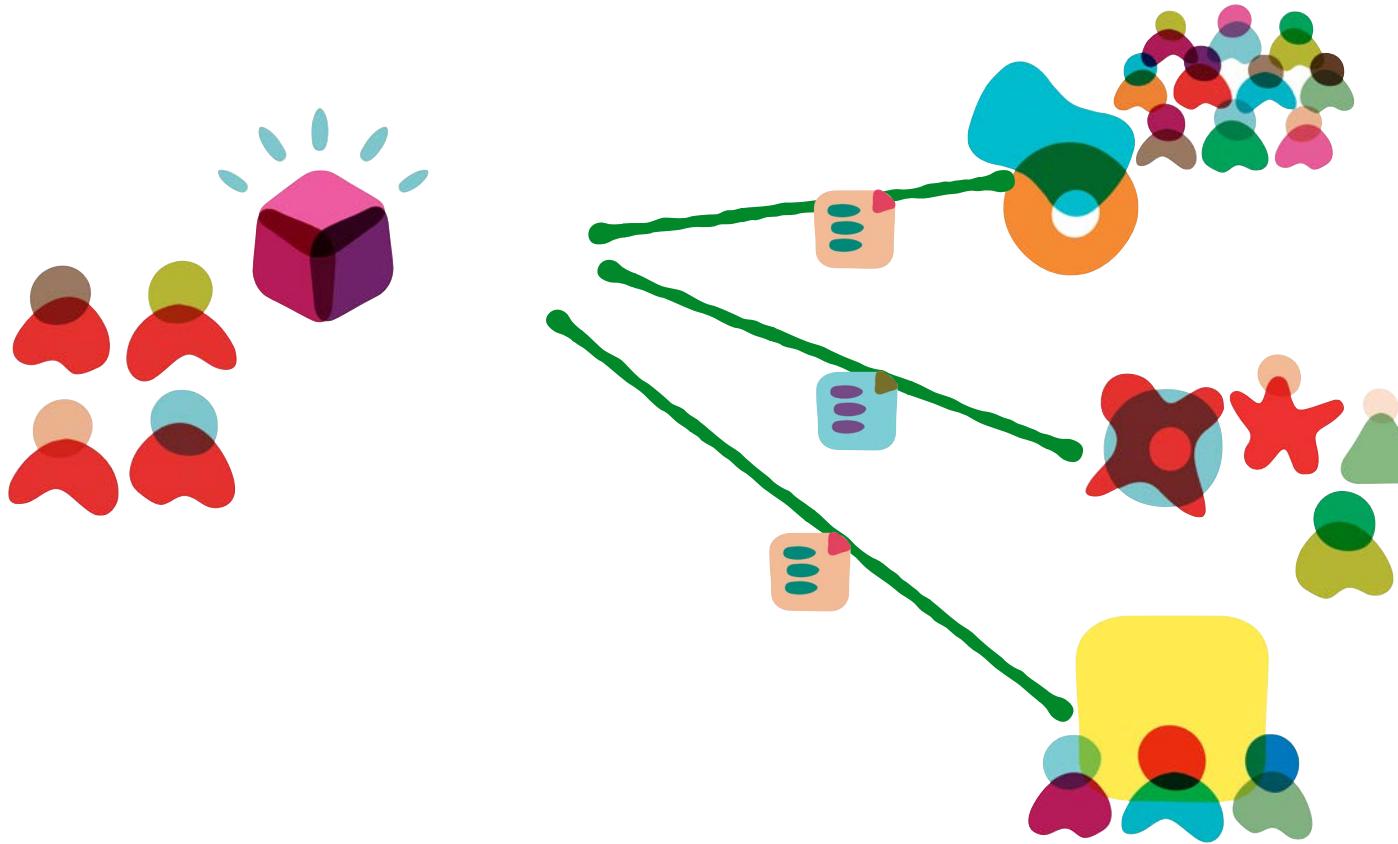
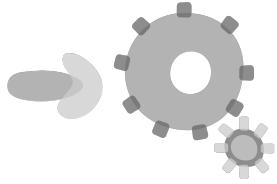
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



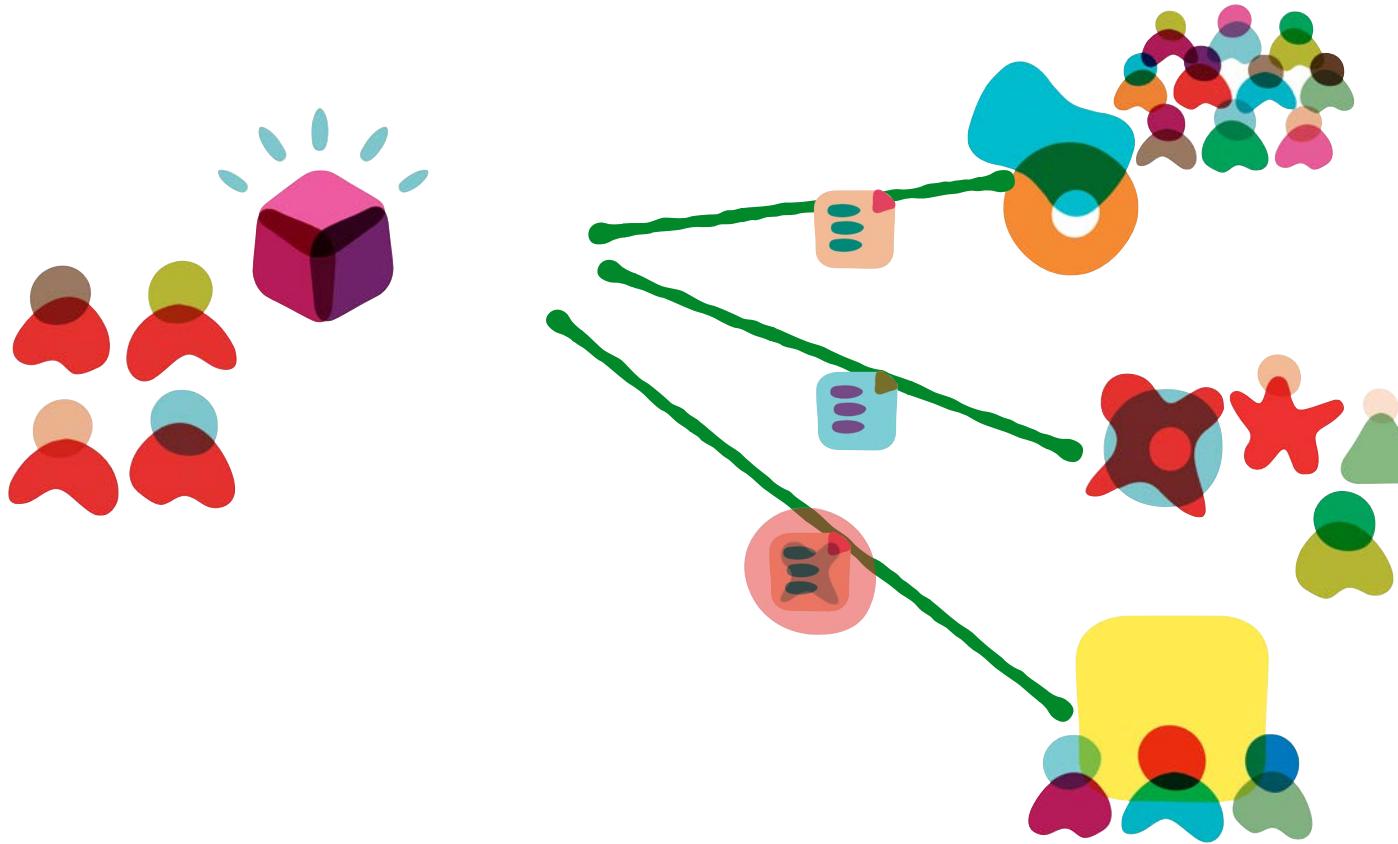
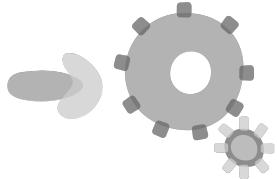
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



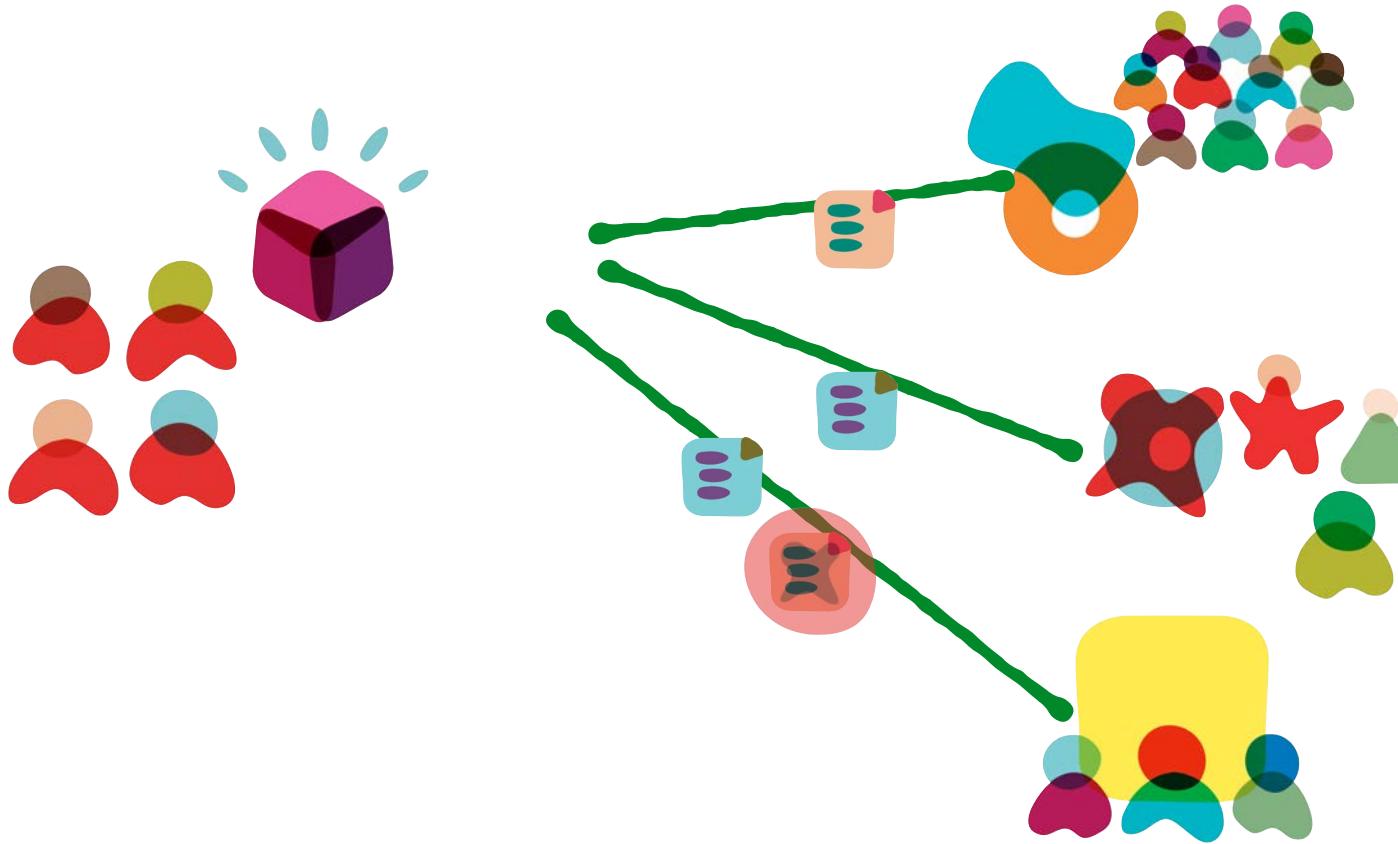
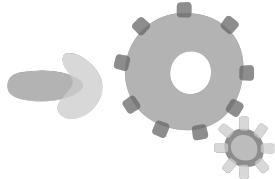
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



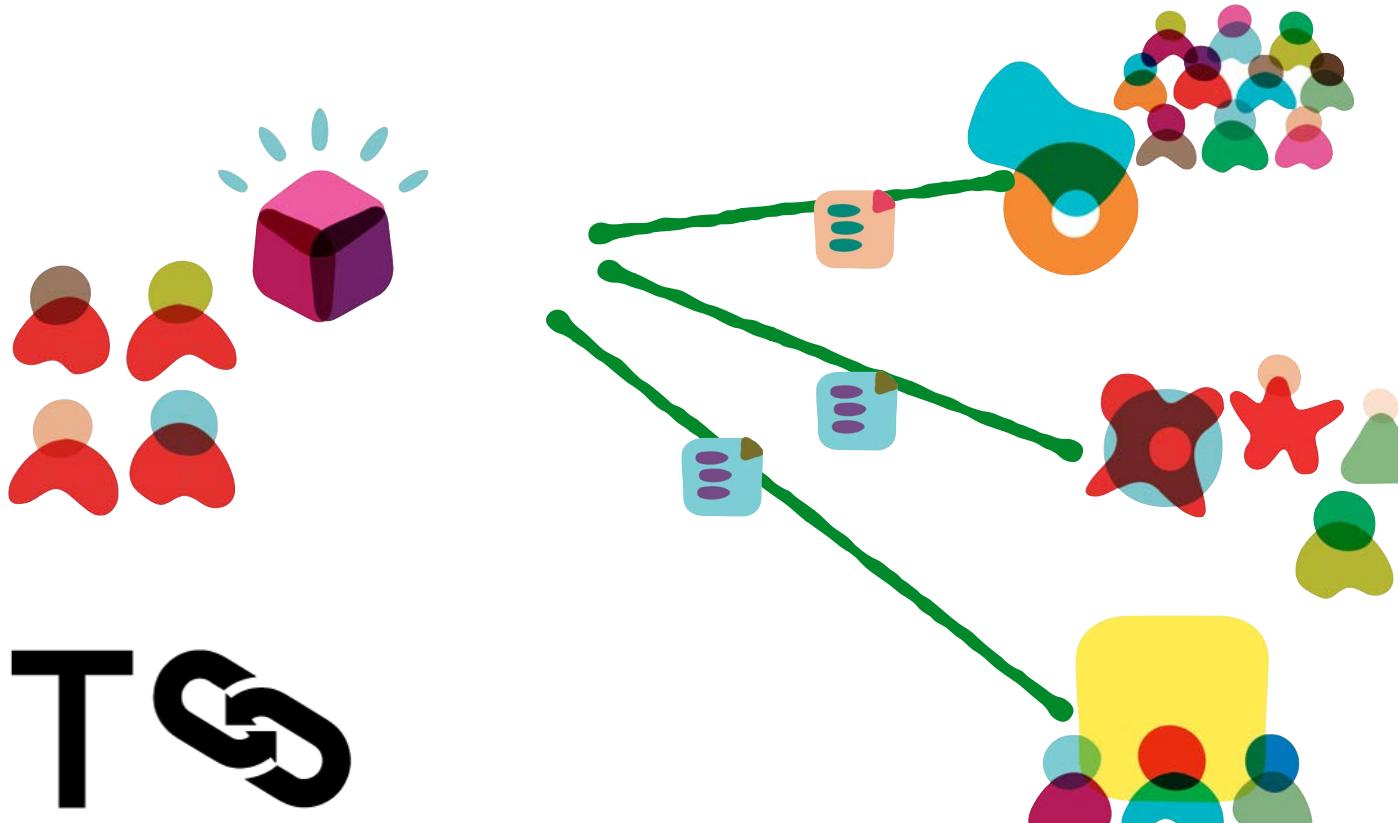
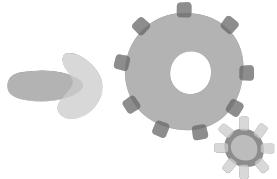
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



martinfowler.com/articles/consumerDrivenContracts.html

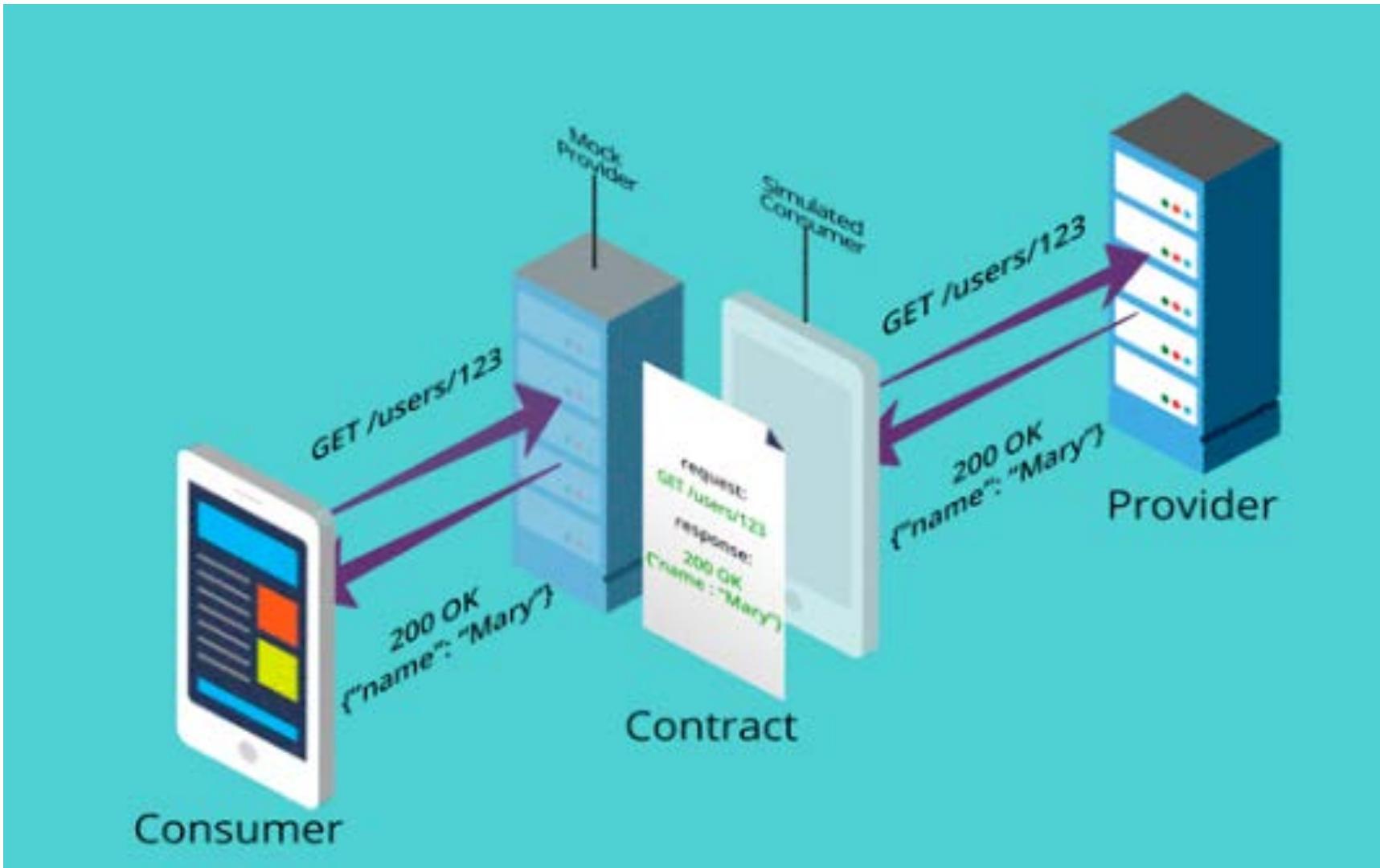
contract fitness function



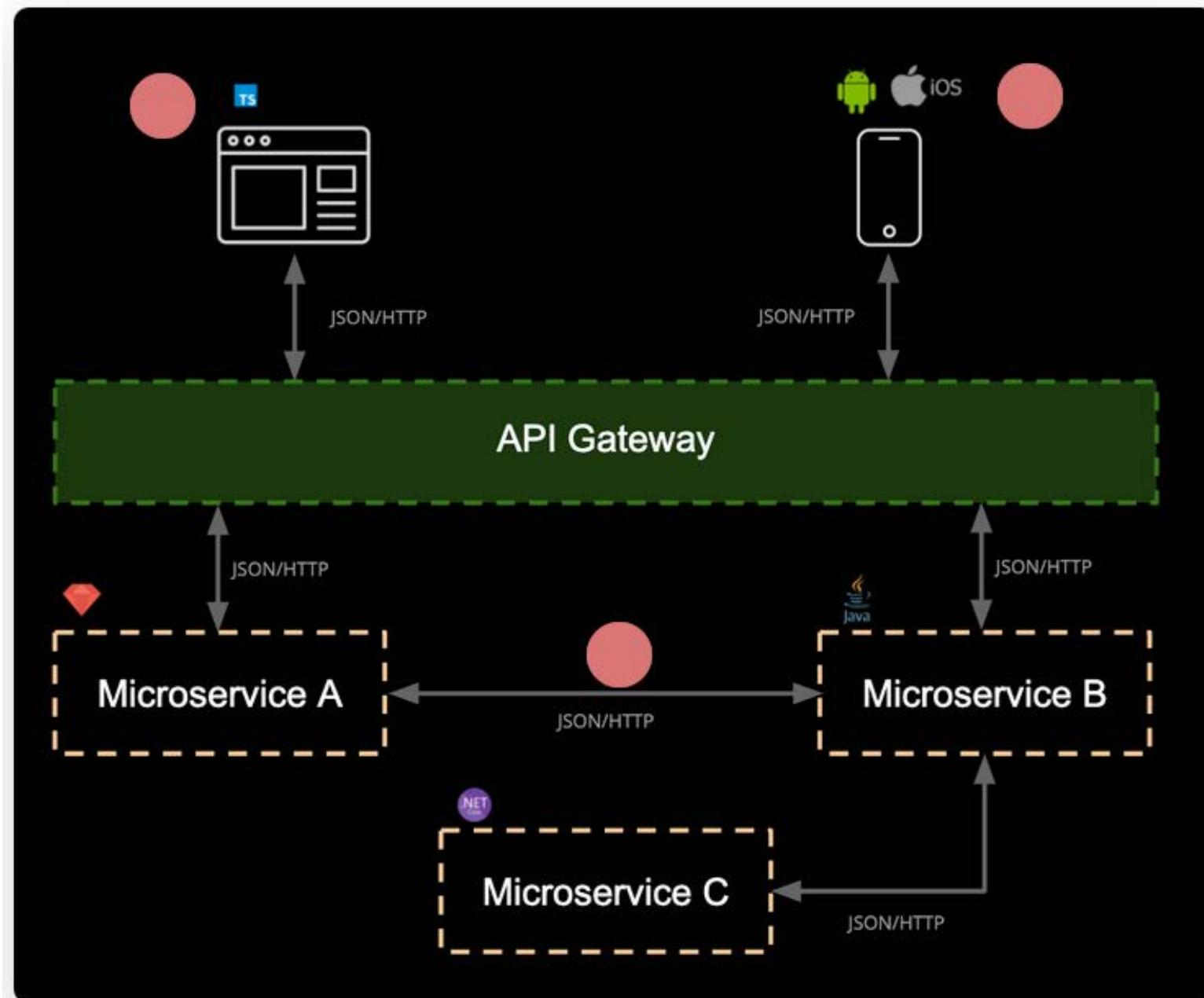
PACTS
<https://docs.pact.io/>

martinfowler.com/articles/consumerDrivenContracts.html

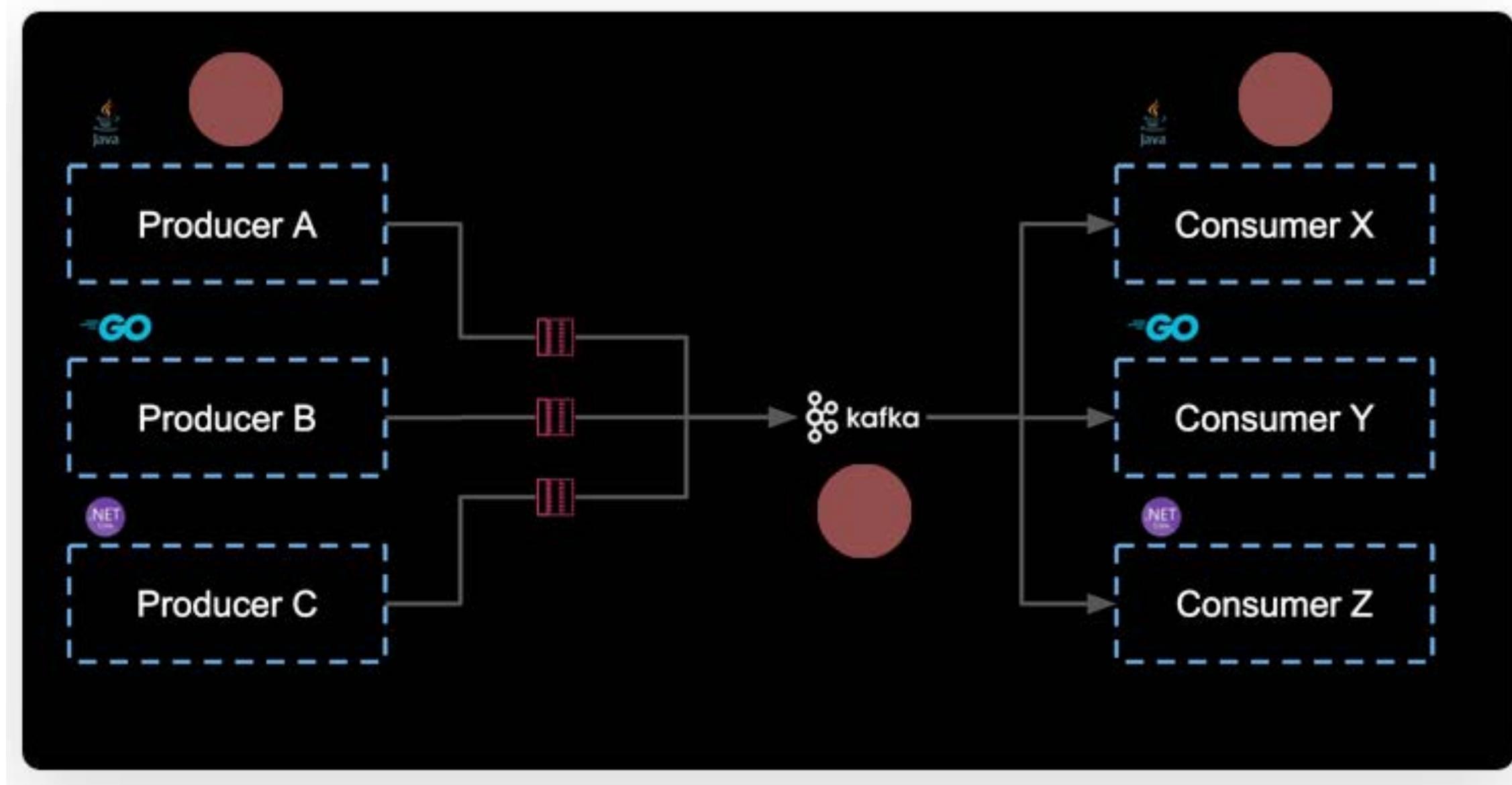
<https://docs.pact.io>



Pact + REST



Pact + EDA



Pact + CI/CD

✓ 🚀 Can I deploy? npm run can-i-deploy

⌚ Ran in 17s ⌚ Waited 5s ⌚ buildkite-i-0954ce6607393c4bc-1

Log Artifacts Agent Environment

+ Expand groups - Collapse groups

>Delete Download Follow

```
1 > Running global environment hook
2 > Setting up elastic stack environment (v2.3.5)
3 > Downloading secrets from buildkite-managedsecretsbucket-1eu@tov5vnt6q
4 > Setting up plugins
5 > Preparing build directory
6 > Running global pre-command hook
7 > Running local pre-command hook
8 > Running plugin github.com/buildkite-plugins/docker-compose-buildkite-plugin#v2.0.0 command hook
9 > Found a pre-built image for wayfinder
10 > Creating docker-compose override file for prebuilt services
11 > Pulling services wayfinder
12 > Running -f docker-compose service: wayfinder
13 $ docker run -d -p buildkite3f89dbc5046b4b72aee0a7480a879d6a -f docker-compose.buildkite-375.override.yml run --name
14 buildkite3f89dbc5046b4b72aee0a7480a879d6a wayfinder npm run can-i-deploy
15
16 ⚡️ network "buildkite3f89dbc5046b4b72aee0a7480a879d6a_default" with the default driver
17
18
19 -test-endpoints@1.0.0 can-i-deploy
20 /scripts/can-i-deploy.sh
21
22
23 -----> Checking if we're safe to deploy!
24 have version: 1.0.0-ab976cccd9ccf3a8
25 running can-i-deploy
26 computer says yes \o/
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 > Container exited normally
138 > Checking linked containers
```

4d888f3cc311306a3a

PROVIDER	P.VERSION	SUCCESS?
.. CentreDiscoveryApi	1.0.0-ab976cccd9ccf3a85ec3be...	true

128 | NUMBER | C.VERSION | successful

129 | discoveryWeb | 0.13.0-ebe9050a | successful

130 | Update results are available. Run npm i -g pact to update.

Using Pact

Defining a contract

```
@Rule  
public PactProviderRuleMk2 mockProvider  
= new PactProviderRuleMk2("test_provider", "localhost", 8080, this);
```

Defining a consumer

```
@Pact(consumer = "test_consumer")  
public RequestResponsePact createPact(PactDslWithProvider builder) {  
    Map<String, String> headers = new HashMap<>();  
    headers.put("Content-Type", "application/json");  
  
    return builder  
        .given("test GET")  
        .uponReceiving("GET REQUEST")  
        .path("/pact")  
        .method("GET")  
        .willRespondWith()  
        .status(200)  
        .headers(headers)  
        .body("{\"condition\": true, \"name\": \"tom\"}")  
        (...)  
}
```

Using Pact

Testing the Consumer

```
@Test
@PactVerification()
public void givenGet_whenSendRequest_shouldReturn200WithProperHeaderAndBody() {

    // when
    ResponseEntity<String> response = new RestTemplate()
        .getForEntity(mockProvider.getUrl() + "/pact", String.class);

    // then
    assertThat(response.getStatusCode().value()).isEqualTo(200);
    assertThat(response.getHeaders().get("Content-Type").contains("application/json")).isTrue();
    assertThat(response.getBody()).contains("condition", "true", "name", "tom");
}
```

Test Output

Testing the Provider

```
@TestTarget
public final Target target = new HttpTarget("http", "localhost", 8082, "/spring-rest");

private static ConfigurableWebApplicationContext application;

@BeforeClass
public static void start() {
    application = (ConfigurableWebApplicationContext)
        SpringApplication.run(MainApplication.class);
}
```

```
Verifying a pact between test_consumer and test_provider
Given test GET
GET REQUEST
    returns a response which
        has status code 200 (OK)
        includes headers
            "Content-Type" with value "application/json" (OK)
        has a matching body (OK)
```

```
Verifying a pact between test_consumer and test_provider
Given test POST
POST REQUEST
    returns a response which
        has status code 201 (OK)
        has a matching body (OK)
```

tradeoffs

strict contracts

- brittle integration points
 - + guaranteed type fidelity
- requires versioning

tradeoffs

value-based contracts

- less certainty in contracts
(requires fitness functions)
 - requires more developer discipline
- + extremely loose coupling
 - + immune from implementation change

tradeoffs

strict contracts

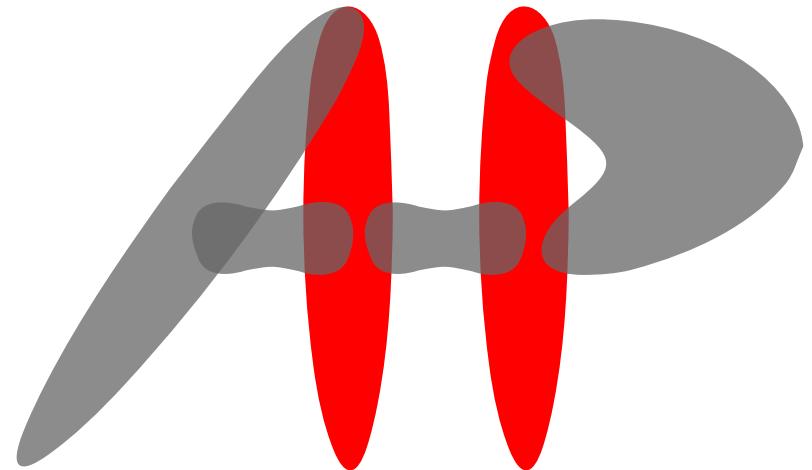
OR

value-based contracts

- * better control of exact parameter passing
- * brittle architecture coupling
- * better documentation via type signatures

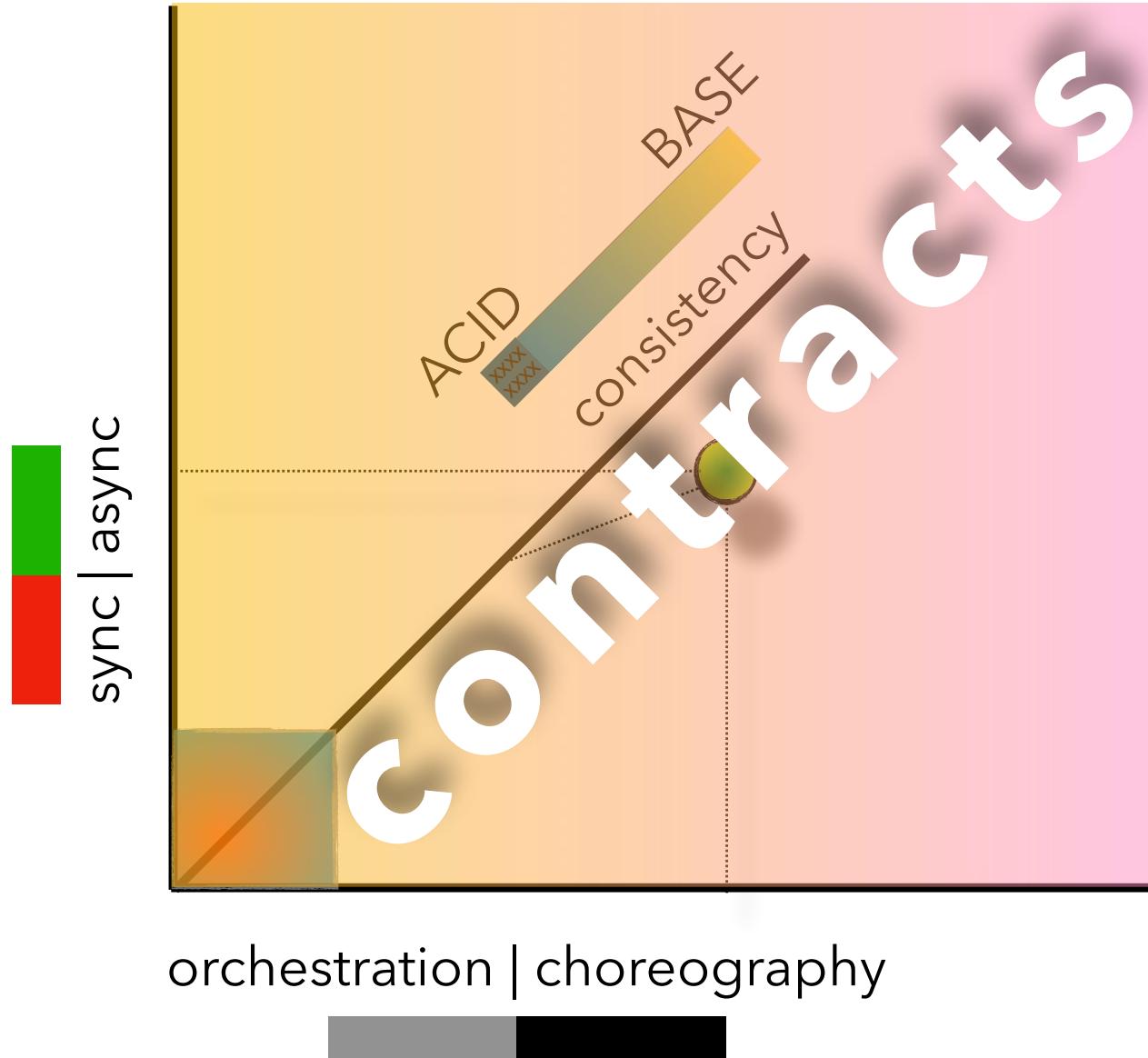
- * requires fitness functions
- * requires documentation
- * extremely loose coupling

architectural quantum 2021



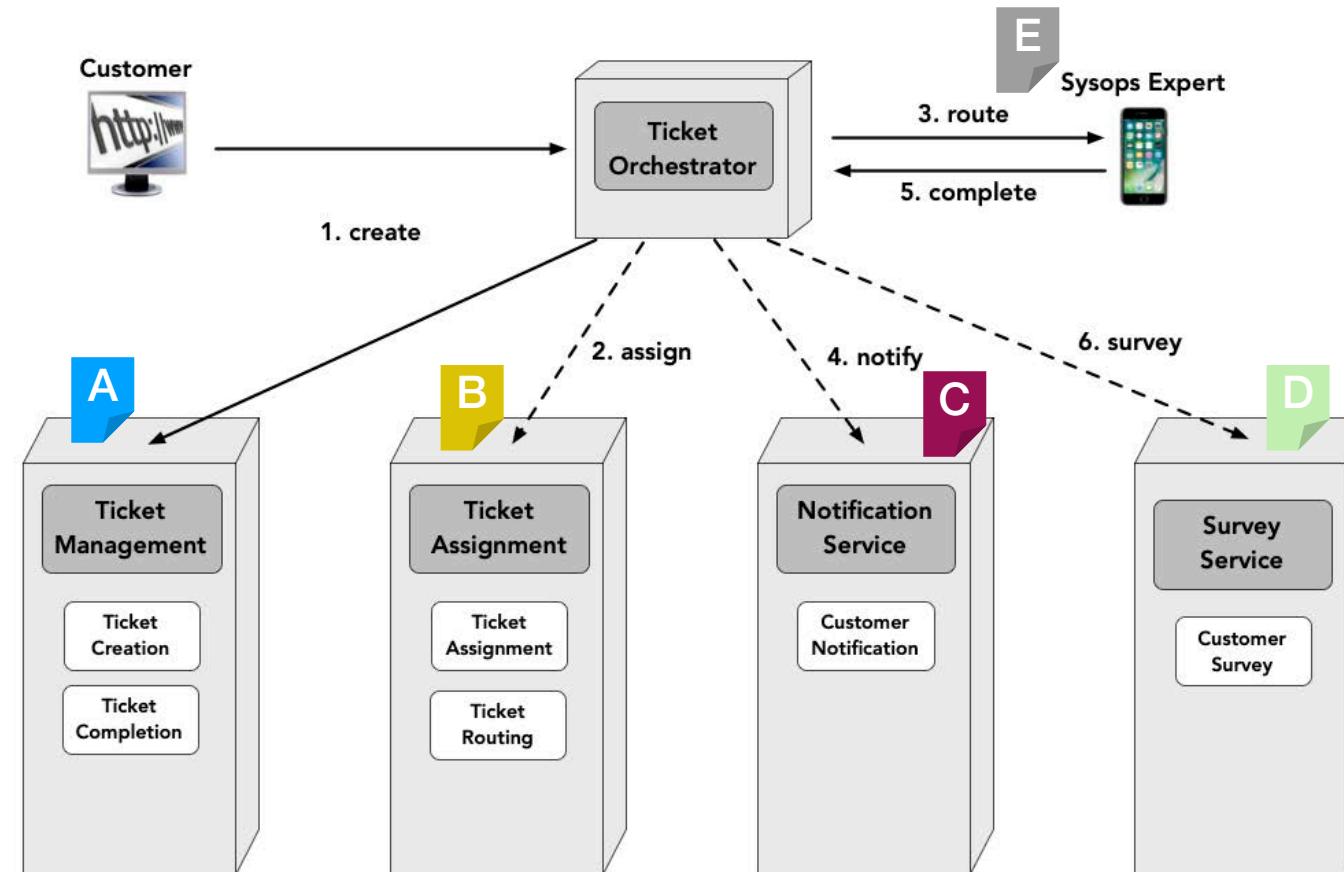
an independently deployable artifact
with high functional cohesion,
high static coupling, &
synchronous dynamic coupling.

dimensions of dynamic quantum coupling



Kata Exercise - Semantic vs. Syntactic Coupling

What type of contract (looser or tighter) should be applied to each of these interactions and why?





Mark Richards

Independent Consultant

Hands-on Software Architect, Published Author

Founder, DeveloperToArchitect.com

@markrichardssa

AP



Neal Ford

ThoughtWorks

Director / Software Architect / Meme Wrangler

<http://www.nealford.com>

@neal4d



Zhamak Dehghani

ThoughtWorks

Principle Consultant

<https://www.thoughtworks.com/profiles/zhamak-dehghani>

@zhamakd

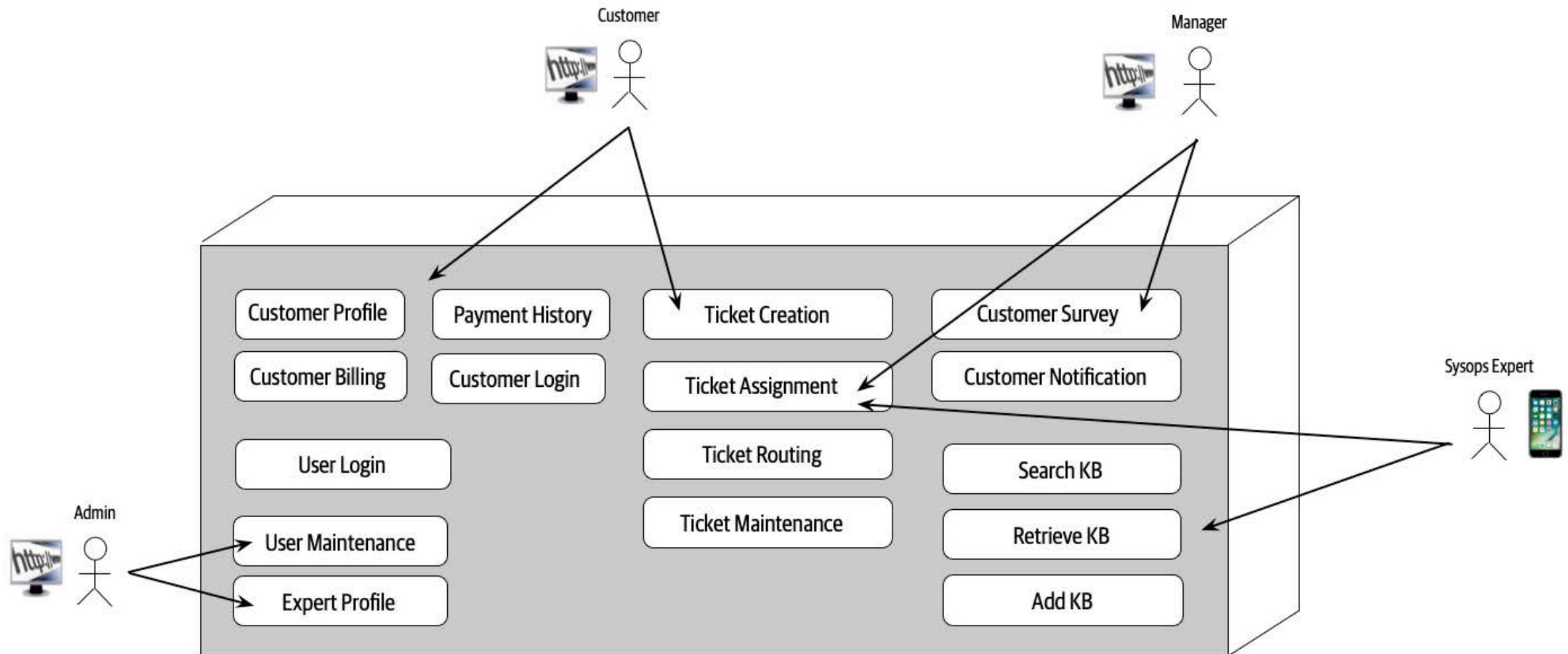
Architecture: The Hard Parts

Virtual Workshop

Our answers (not the only
ones) to Katas

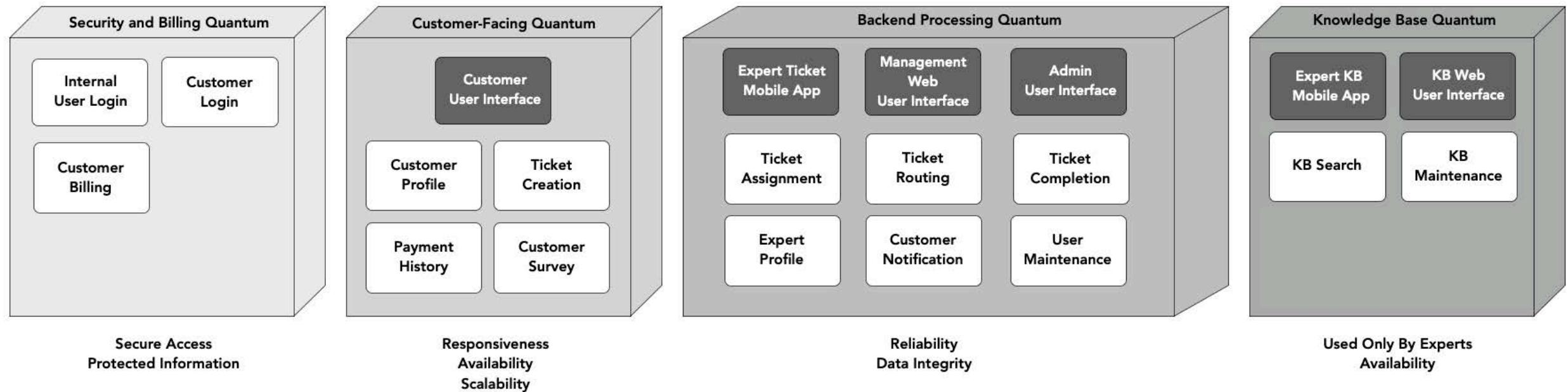
Kata Exercise - Components and Quanta

Based on the components and data below, identify the architecture quanta and components included in each quantum



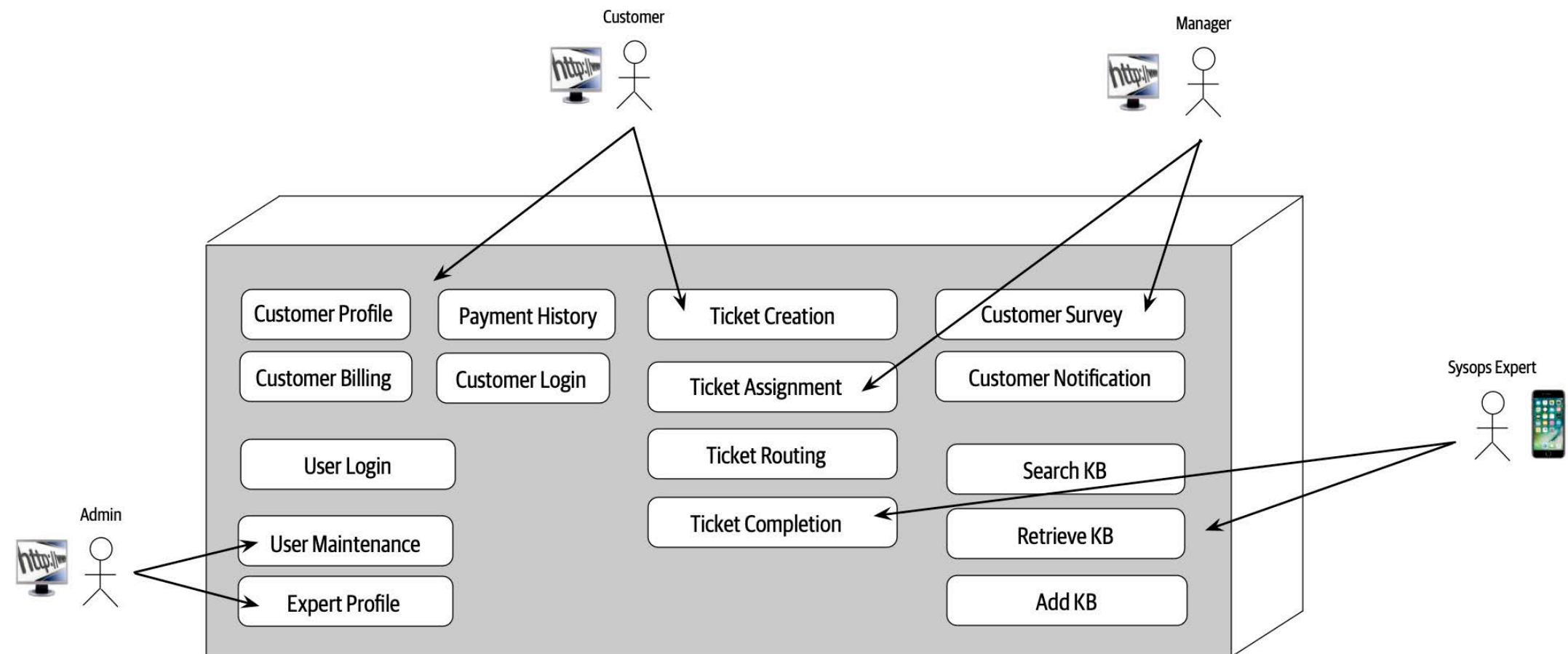
Kata Exercise - Components and Quanta

Based on the components and data below, identify the architecture quanta and components included in each quantum

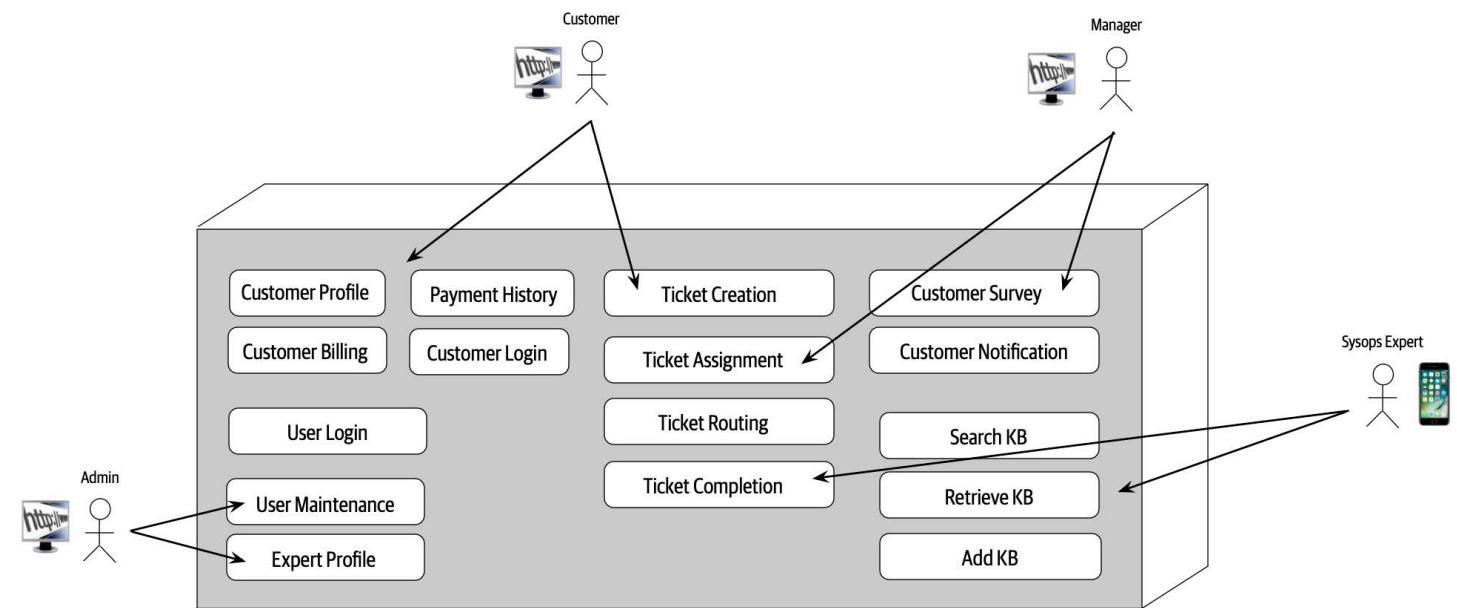


Kata Exercise - Service Identification

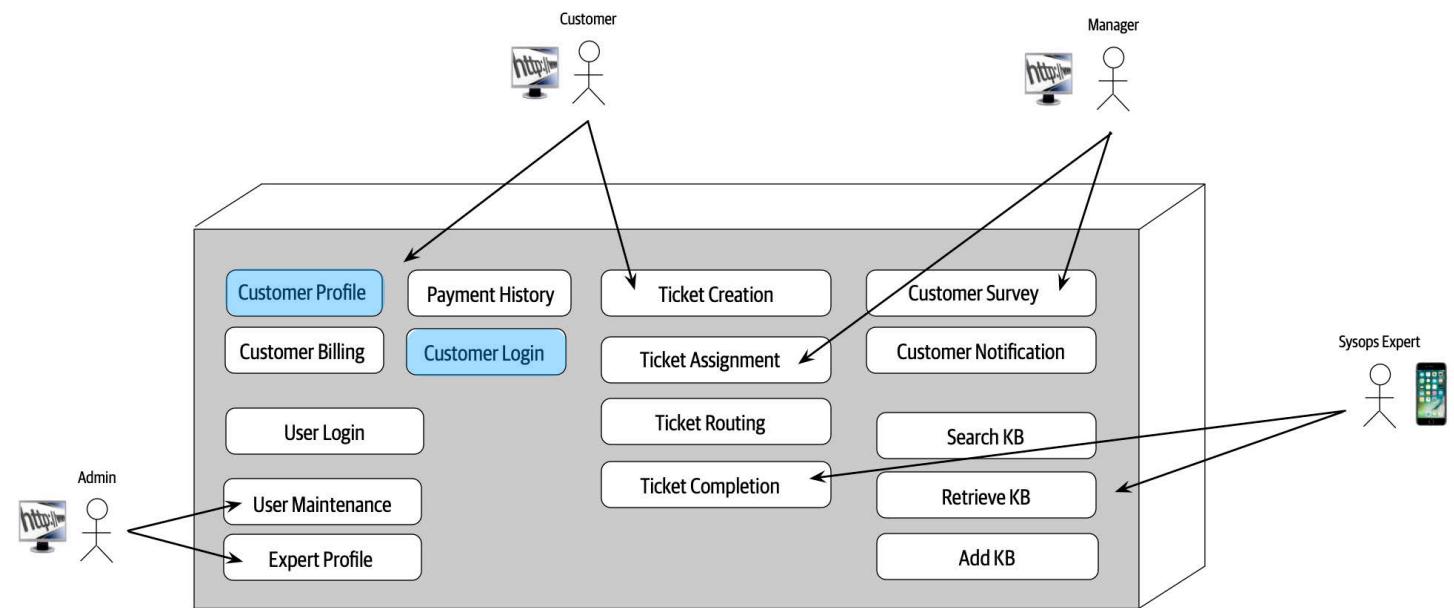
Management has given the go-ahead for moving to a distributed architecture. Based on the components illustrated below and what you know about service granularity, your job now is to identify the initial service candidates.



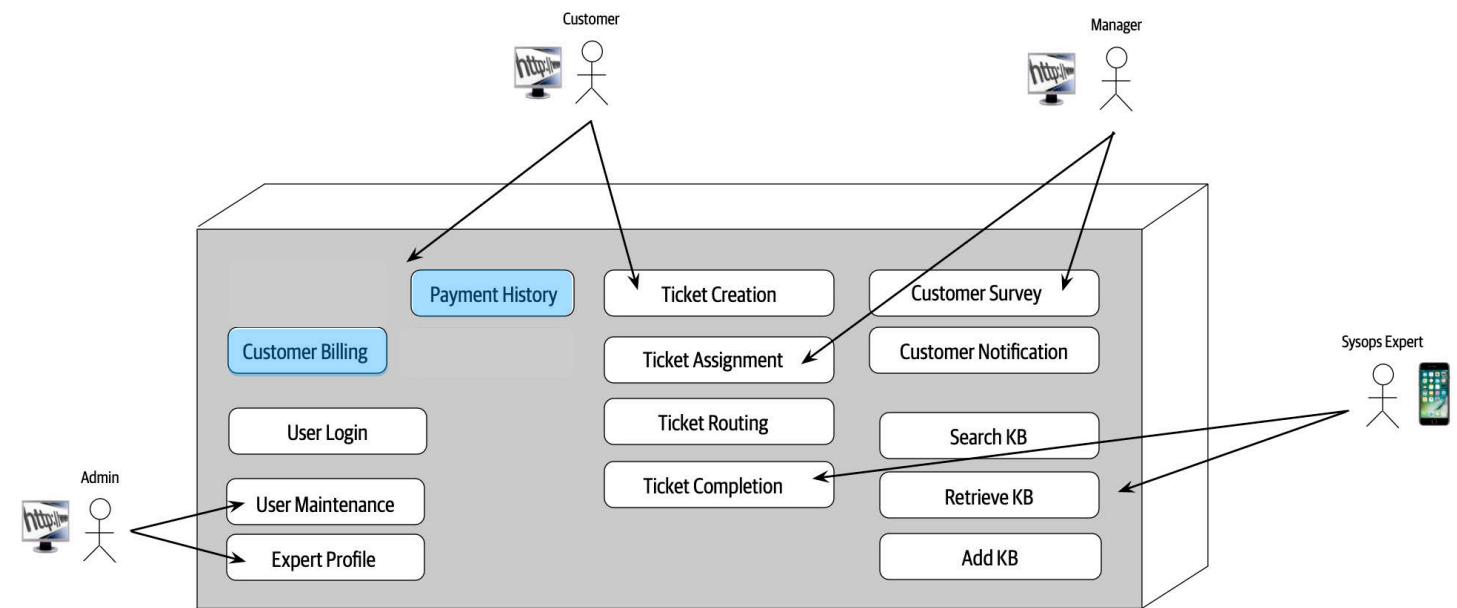
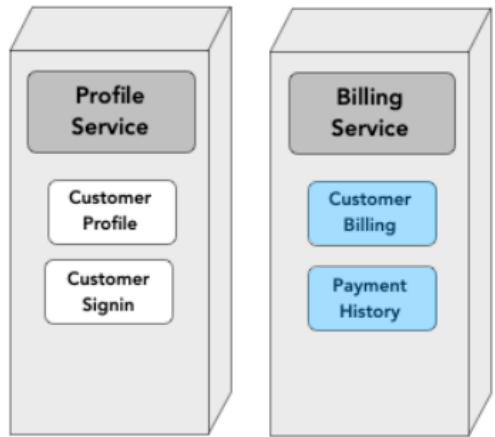
Kata Exercise - Service Identification



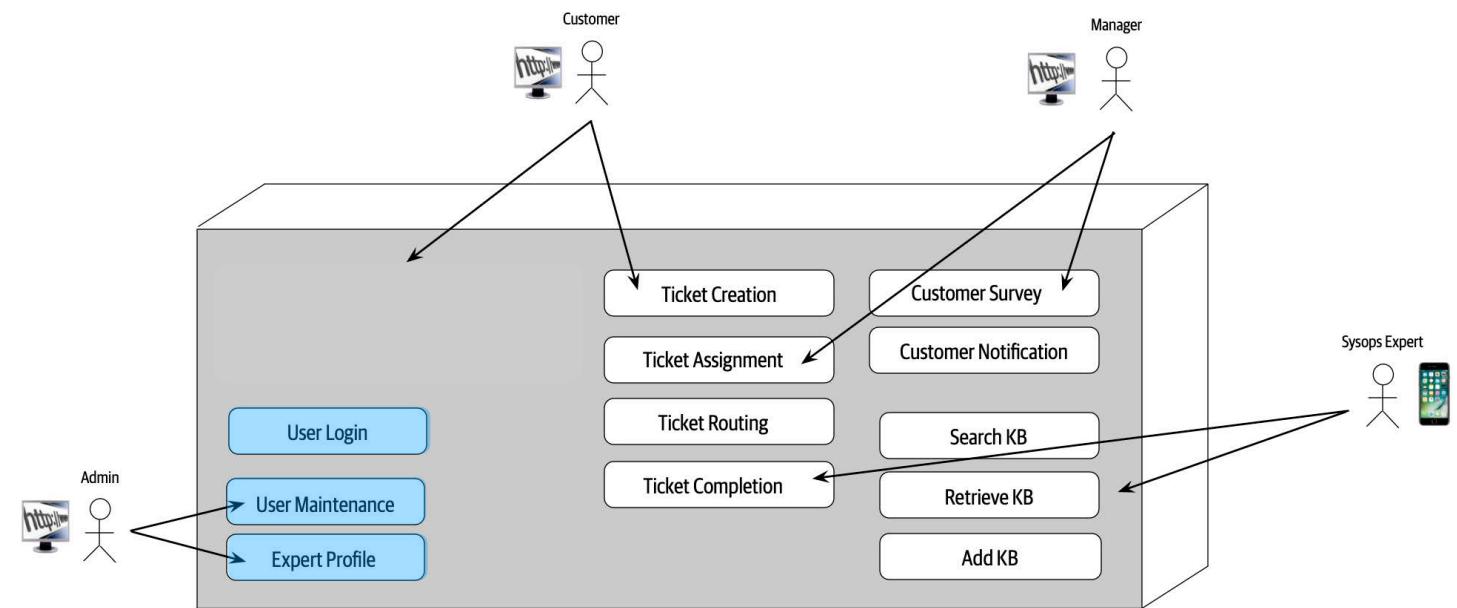
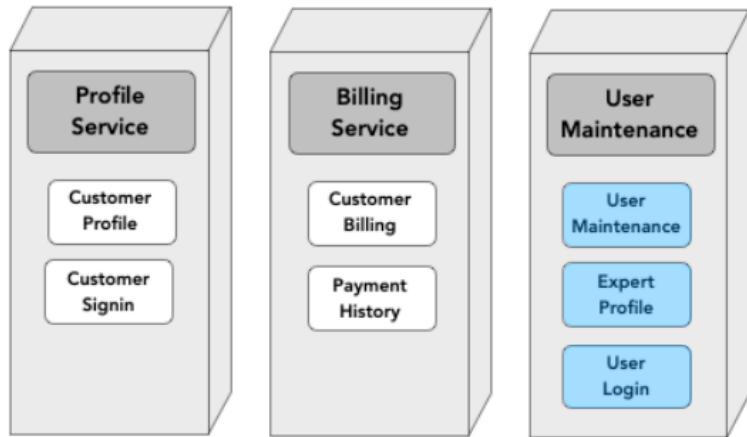
Kata Exercise - Service Identification



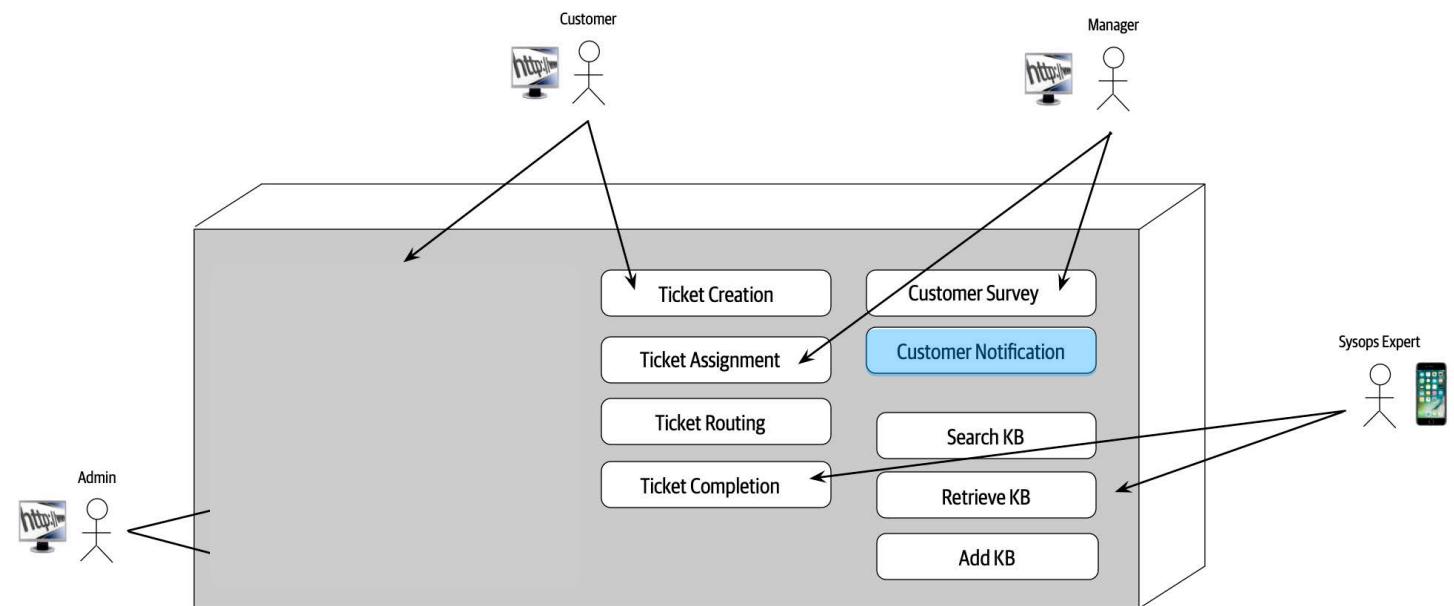
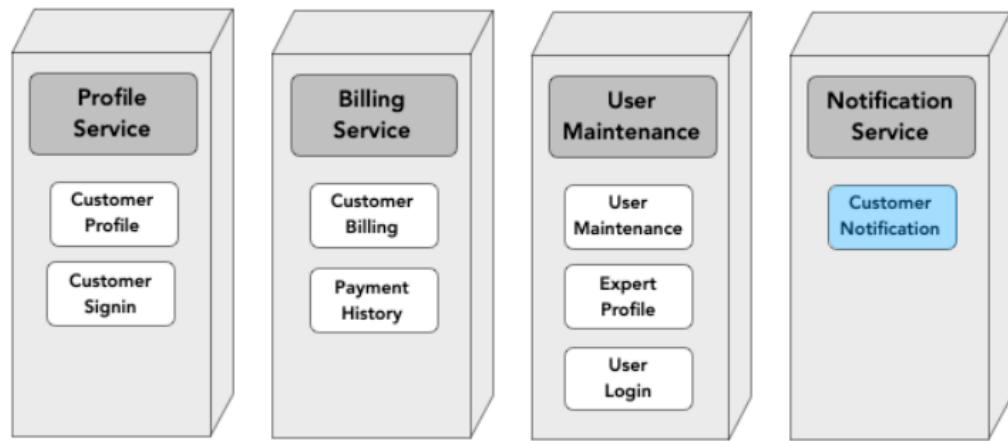
Kata Exercise - Service Identification



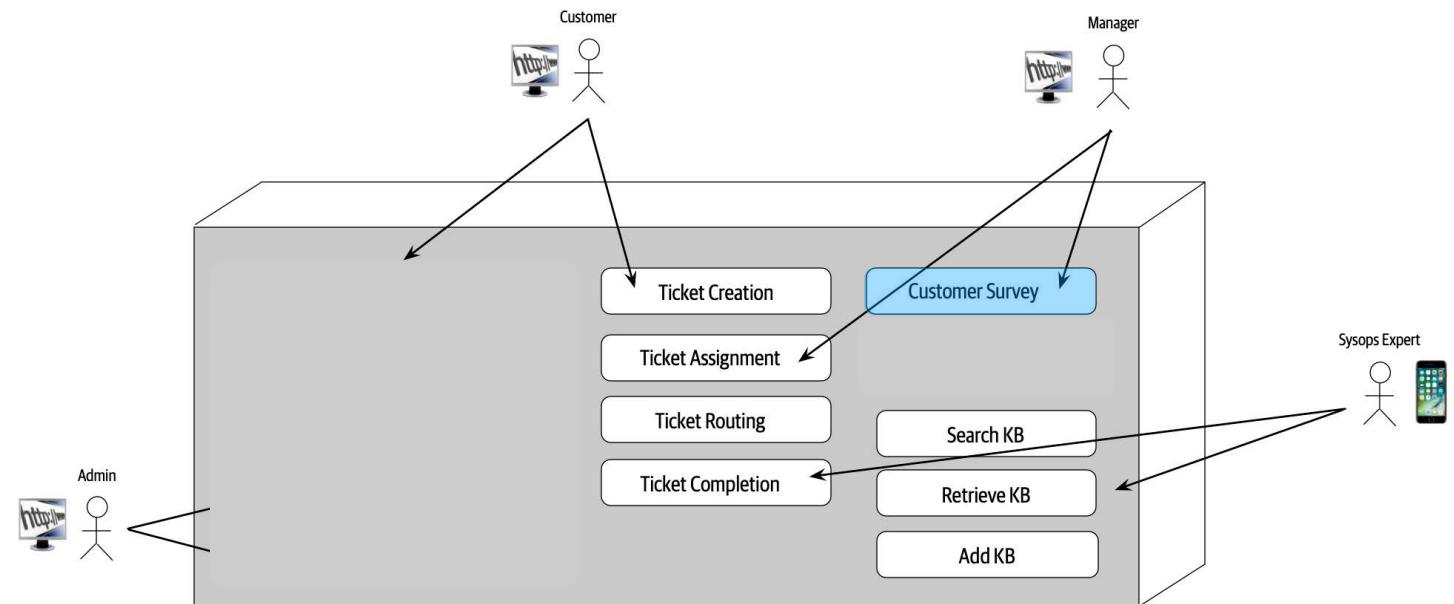
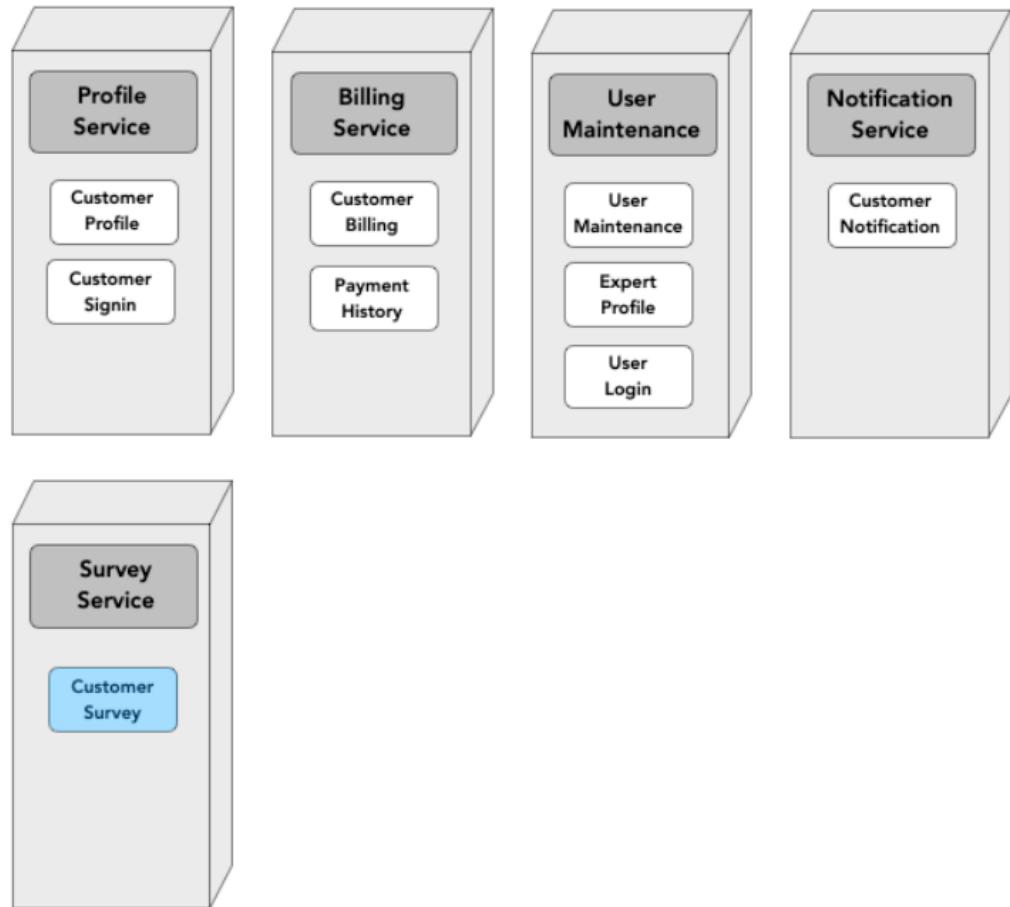
Kata Exercise - Service Identification



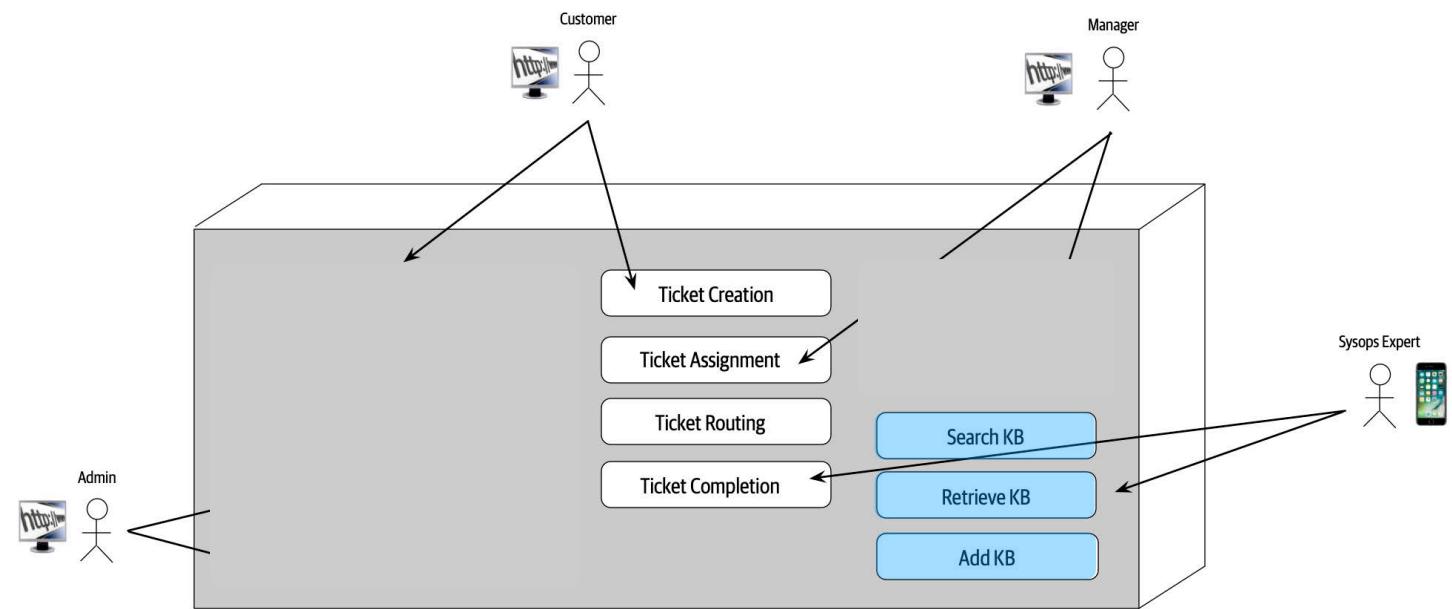
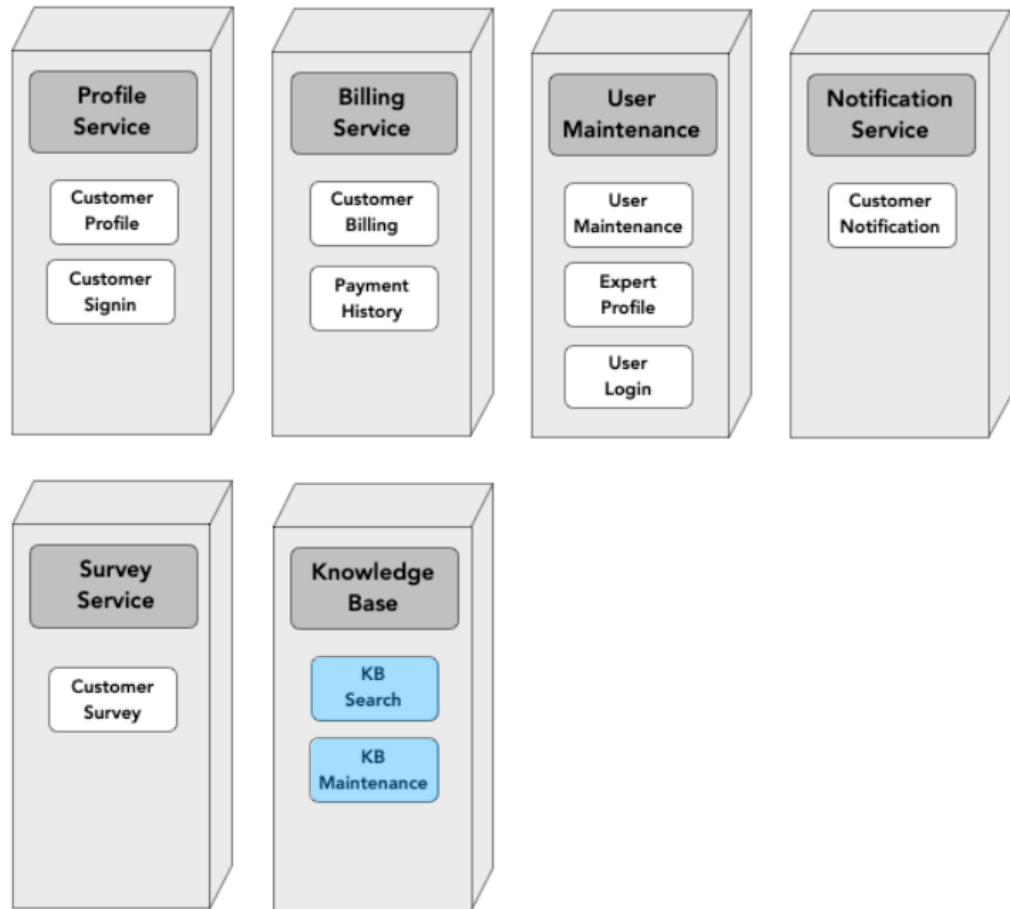
Kata Exercise - Service Identification



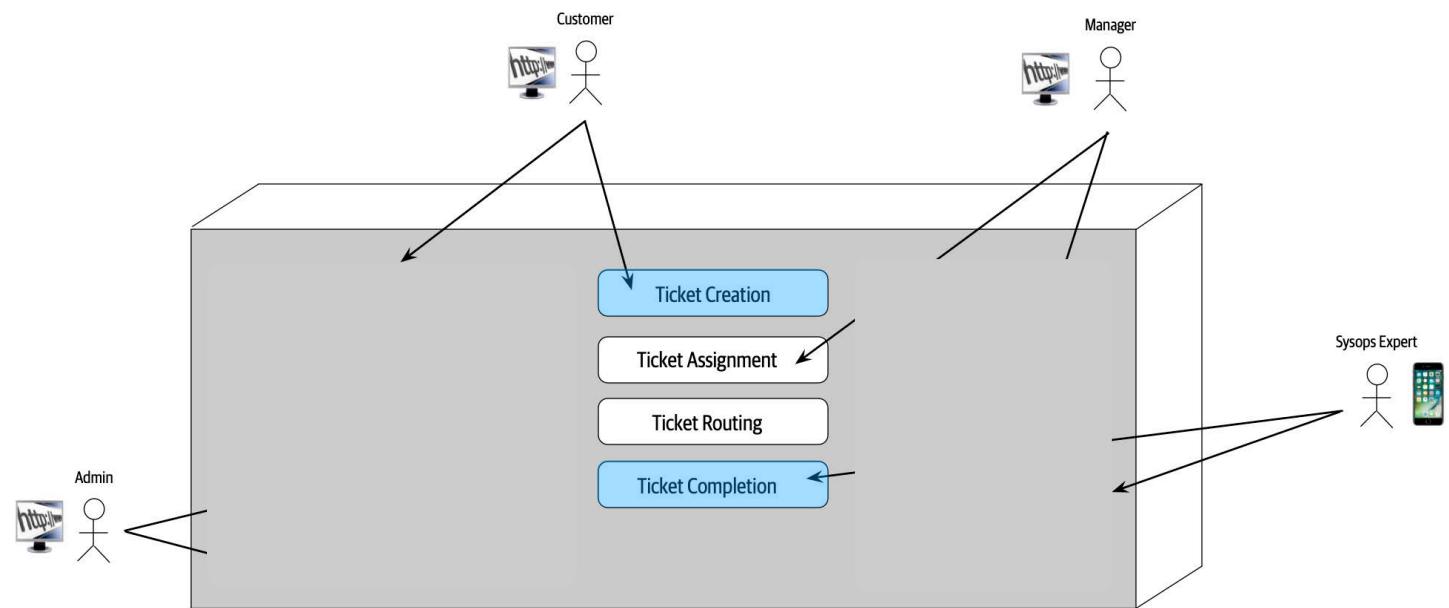
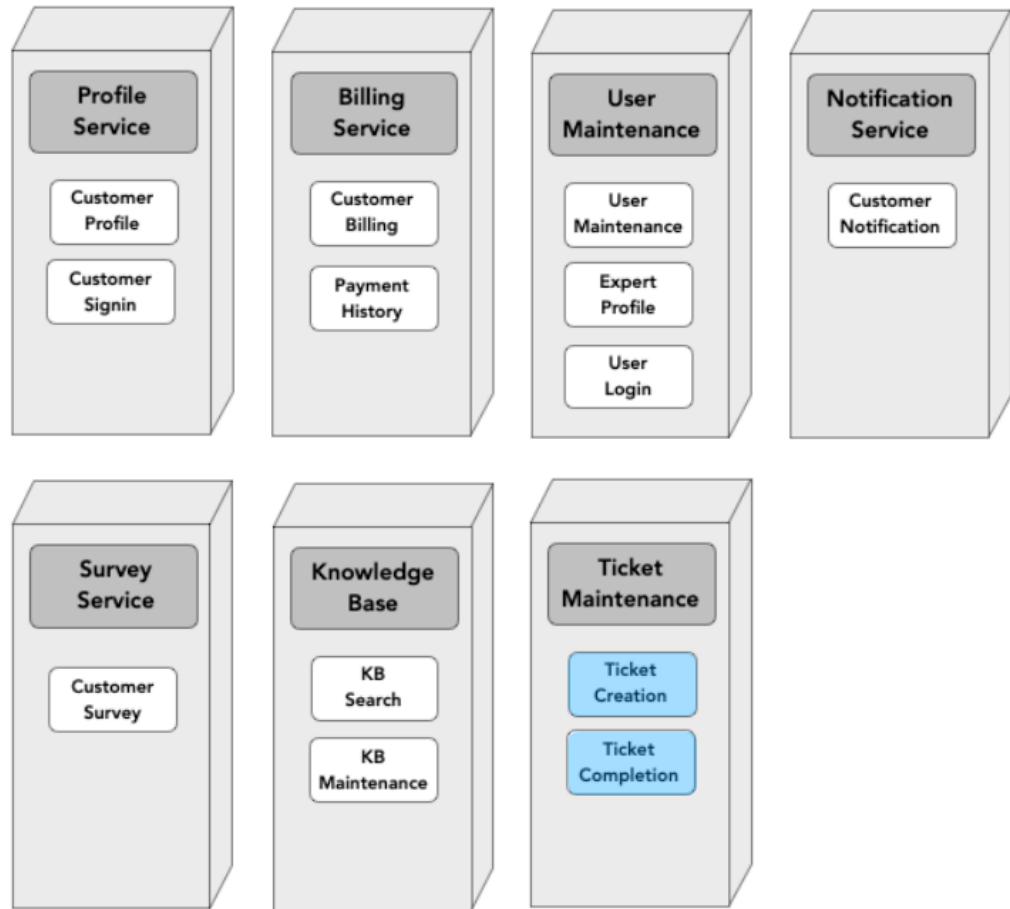
Kata Exercise - Service Identification



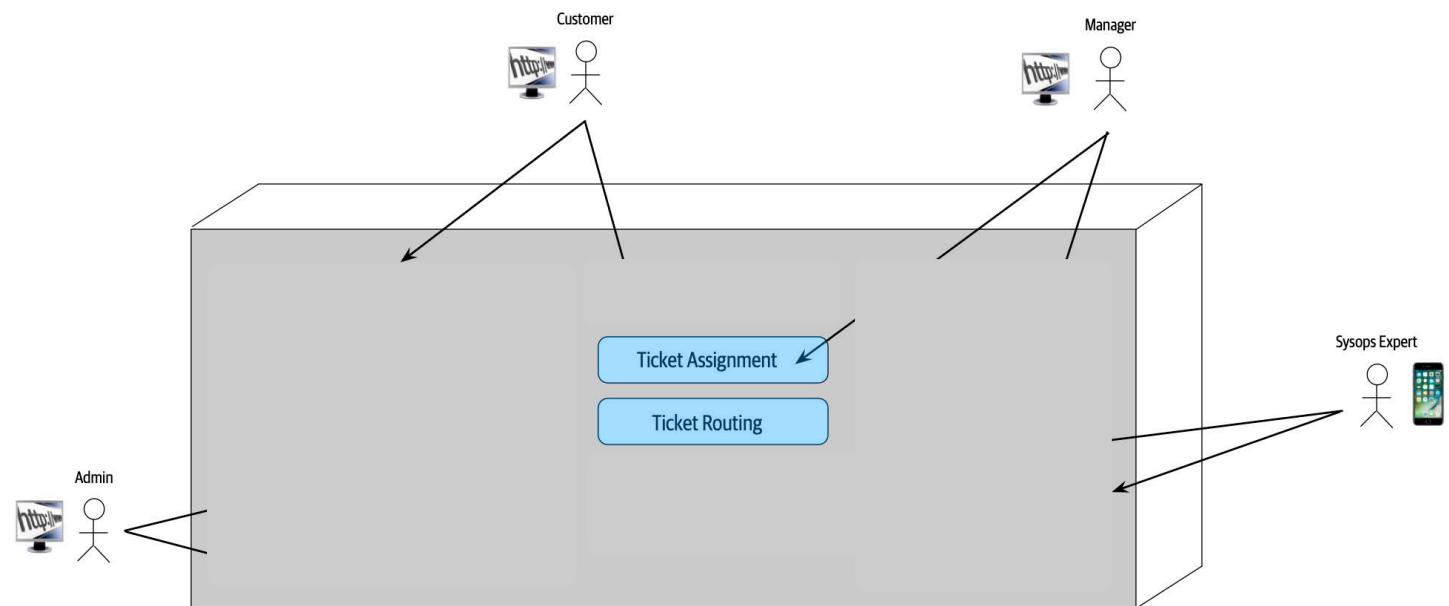
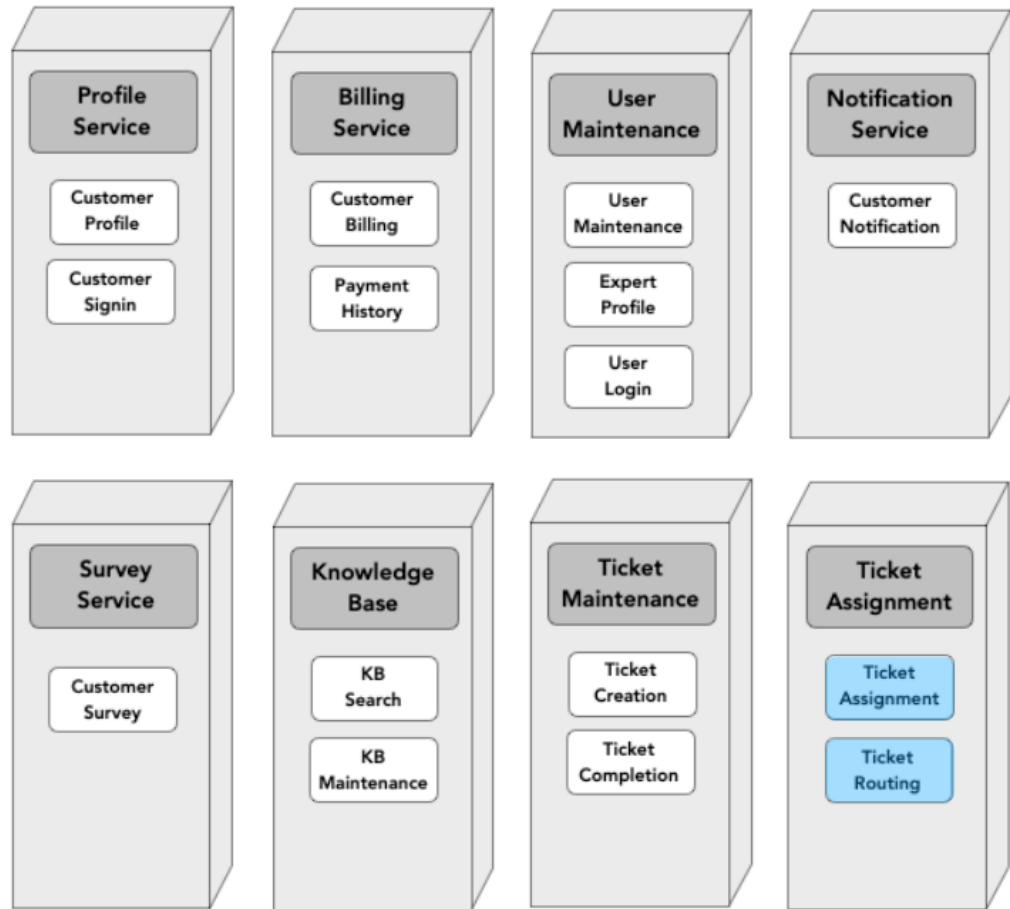
Kata Exercise - Service Identification



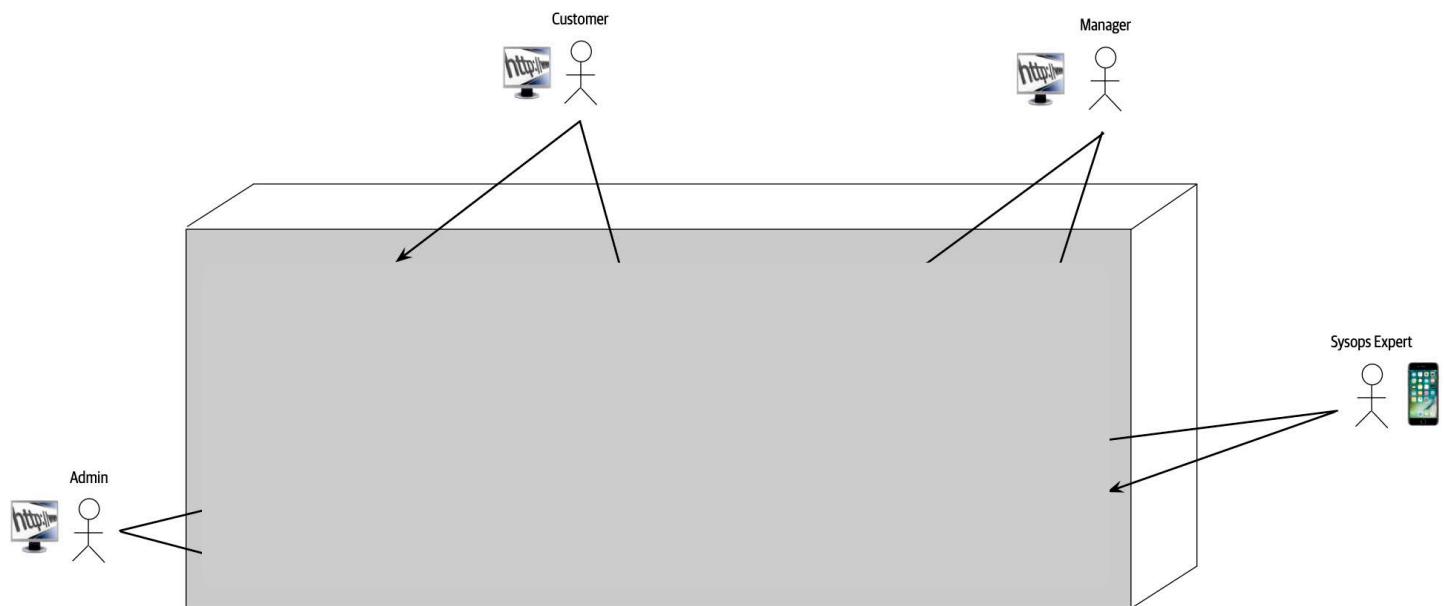
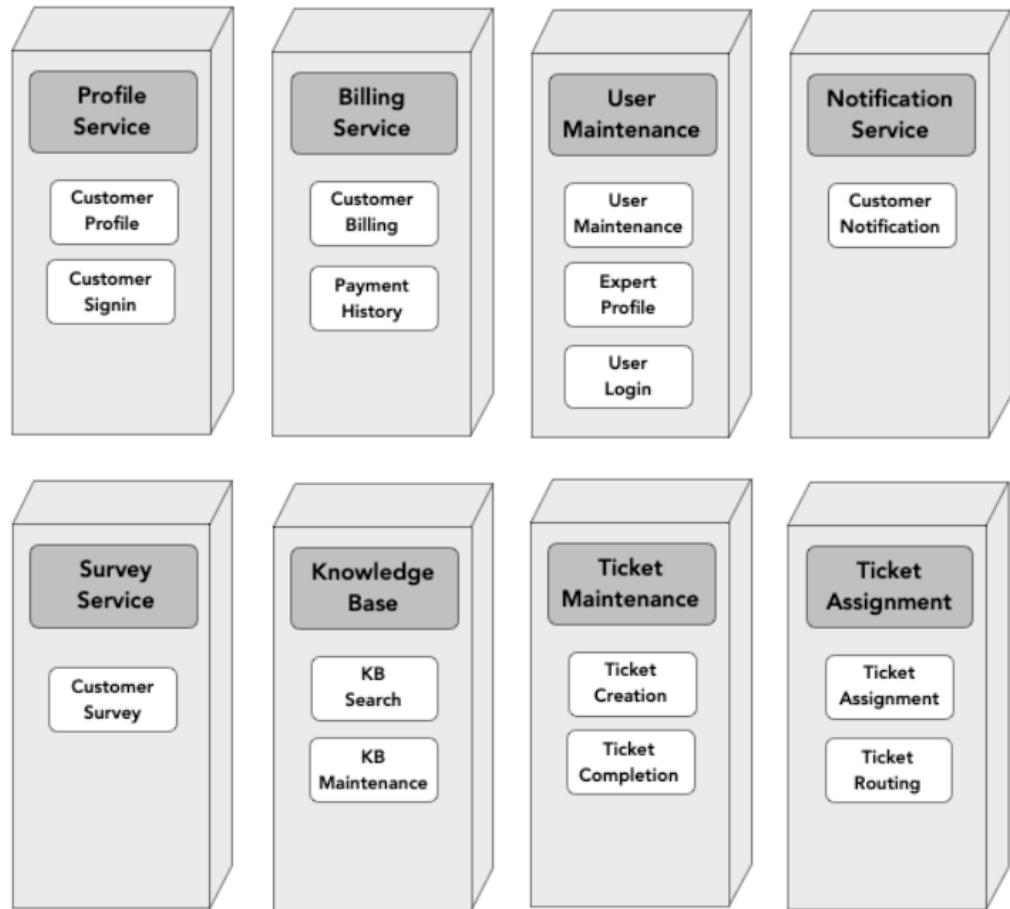
Kata Exercise - Service Identification



Kata Exercise - Service Identification

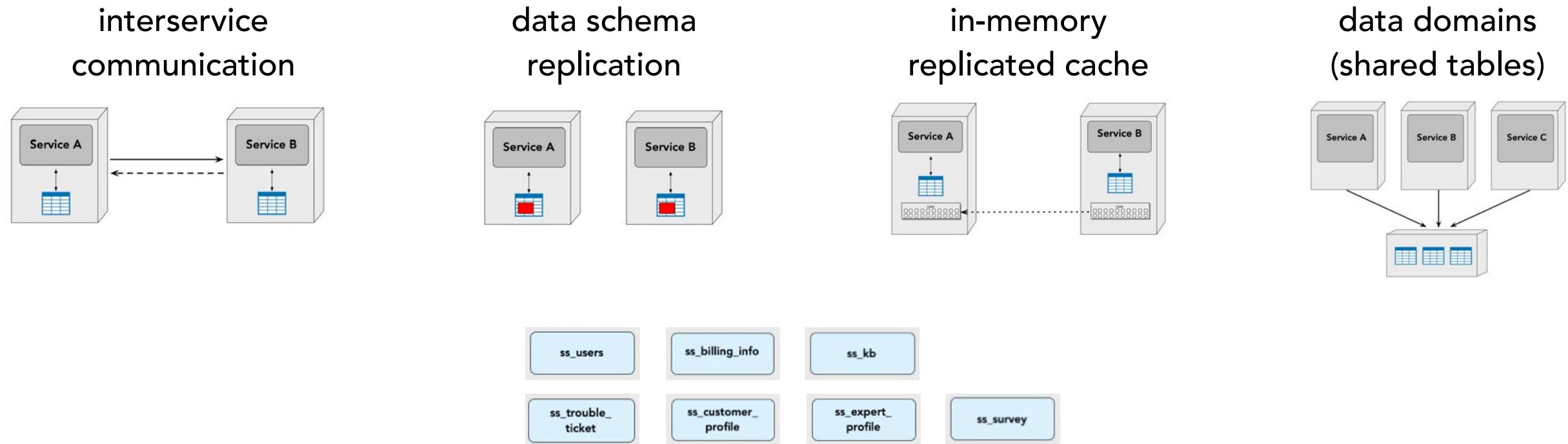


Kata Exercise - Service Identification



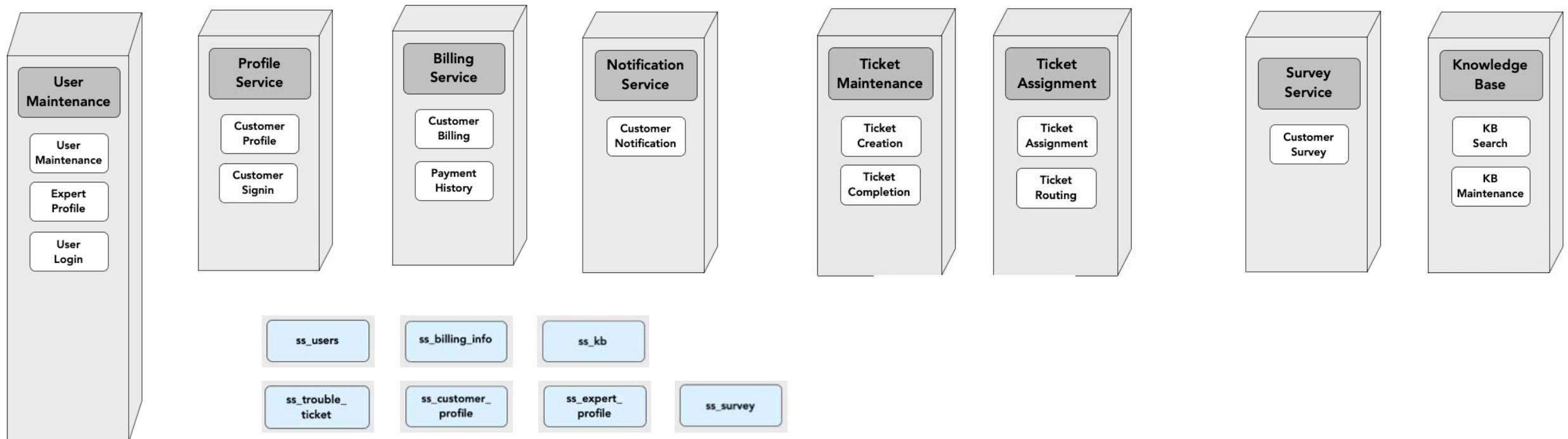
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices for data access are indicated below. Don't forget you can use any combination of these depending on the data.



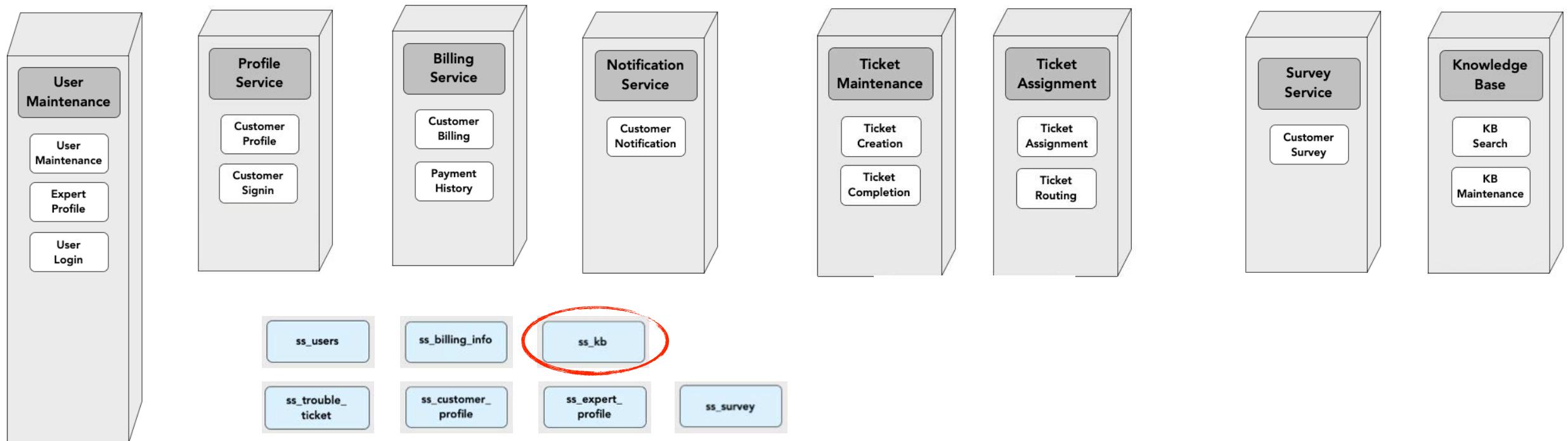
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



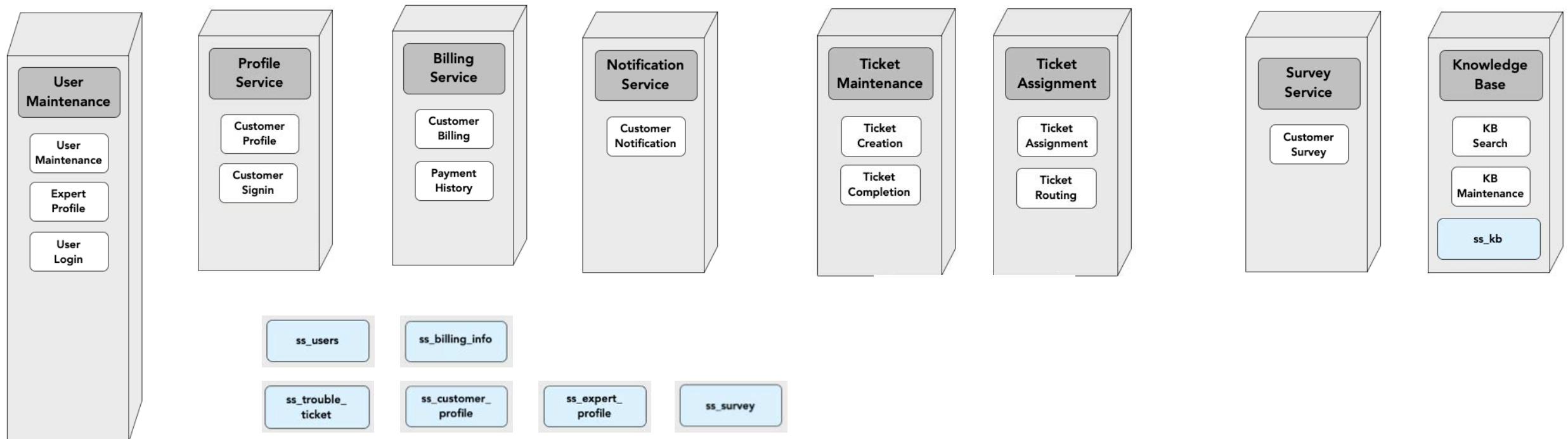
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



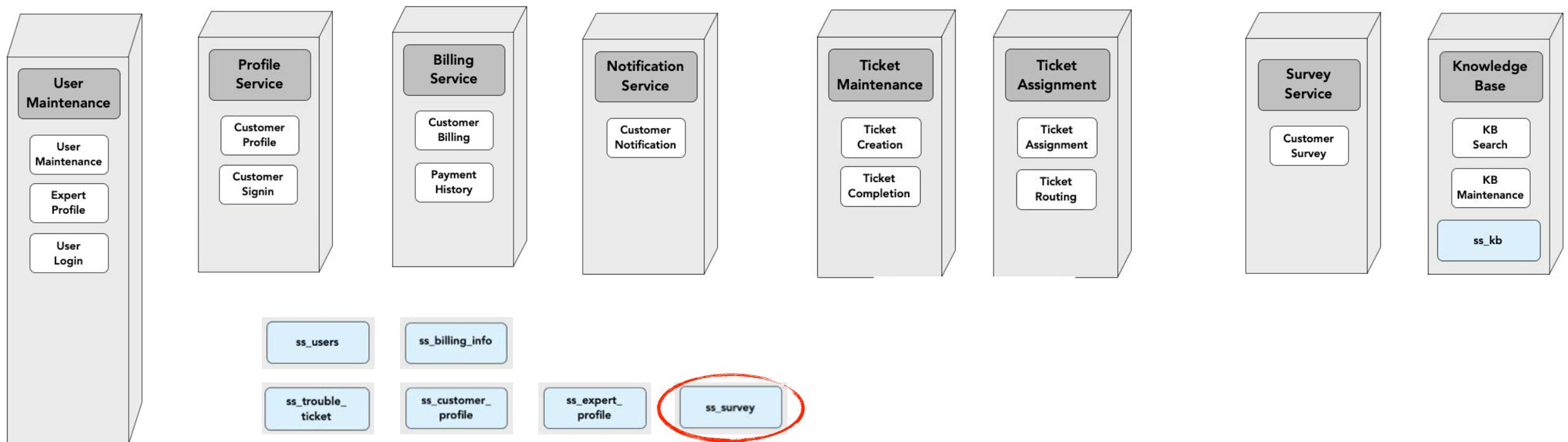
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



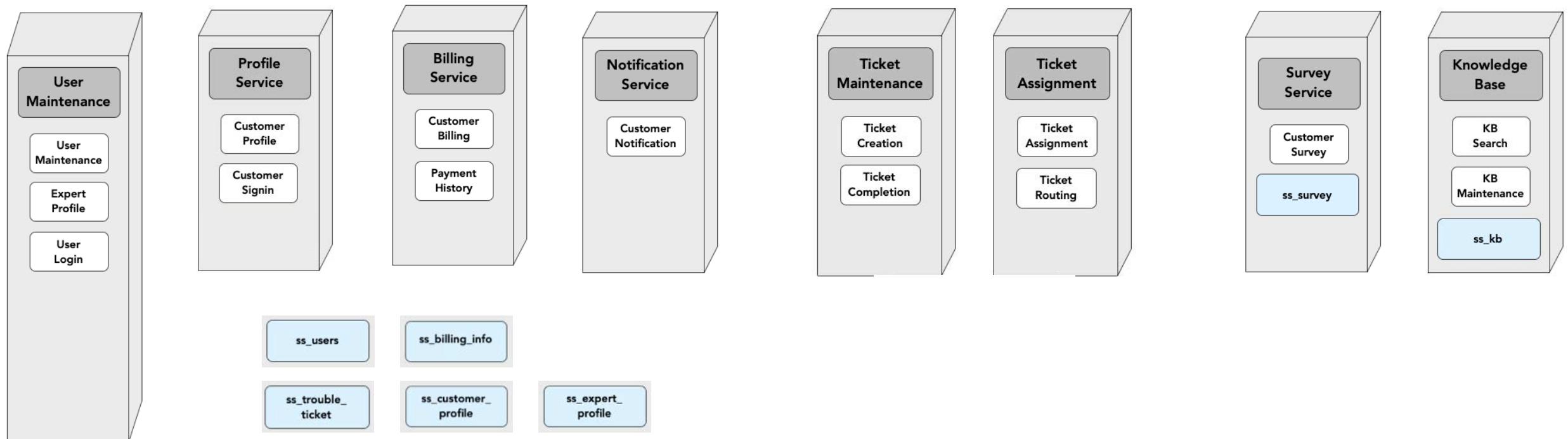
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



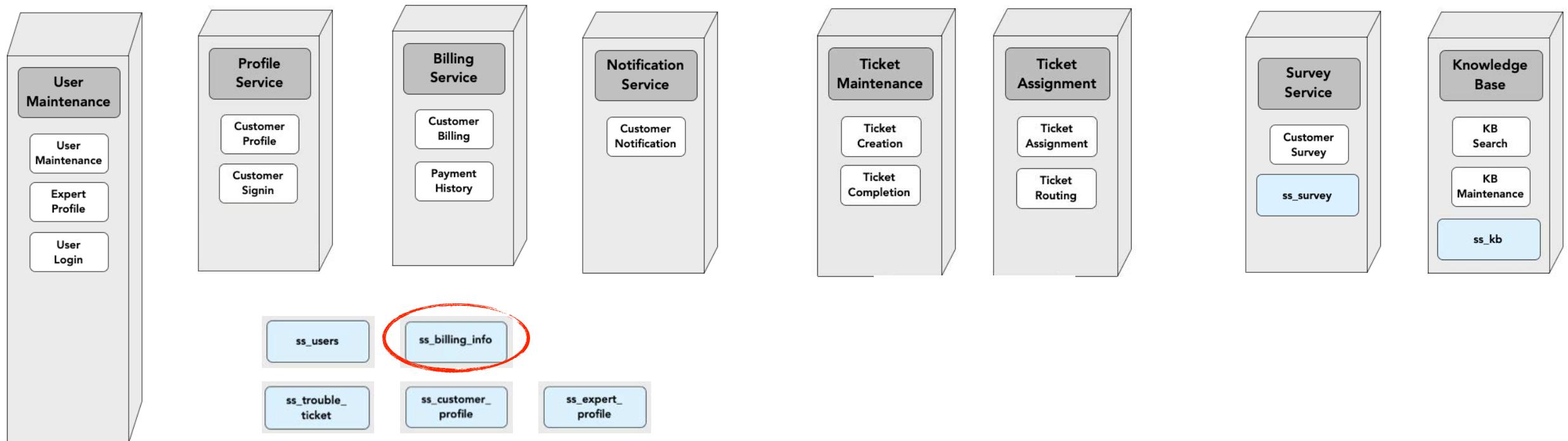
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



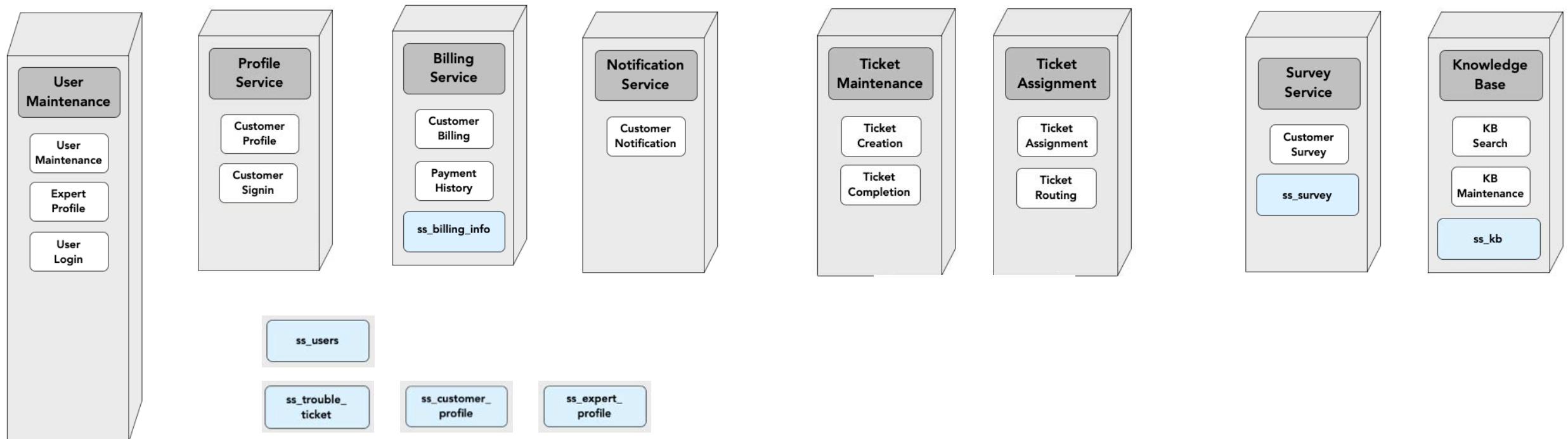
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



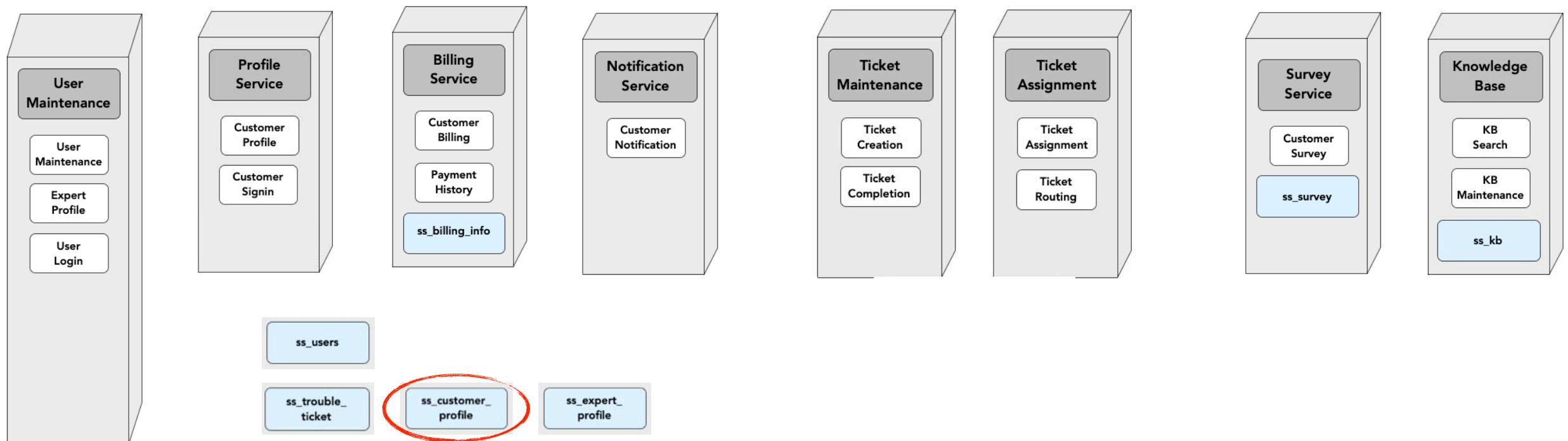
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



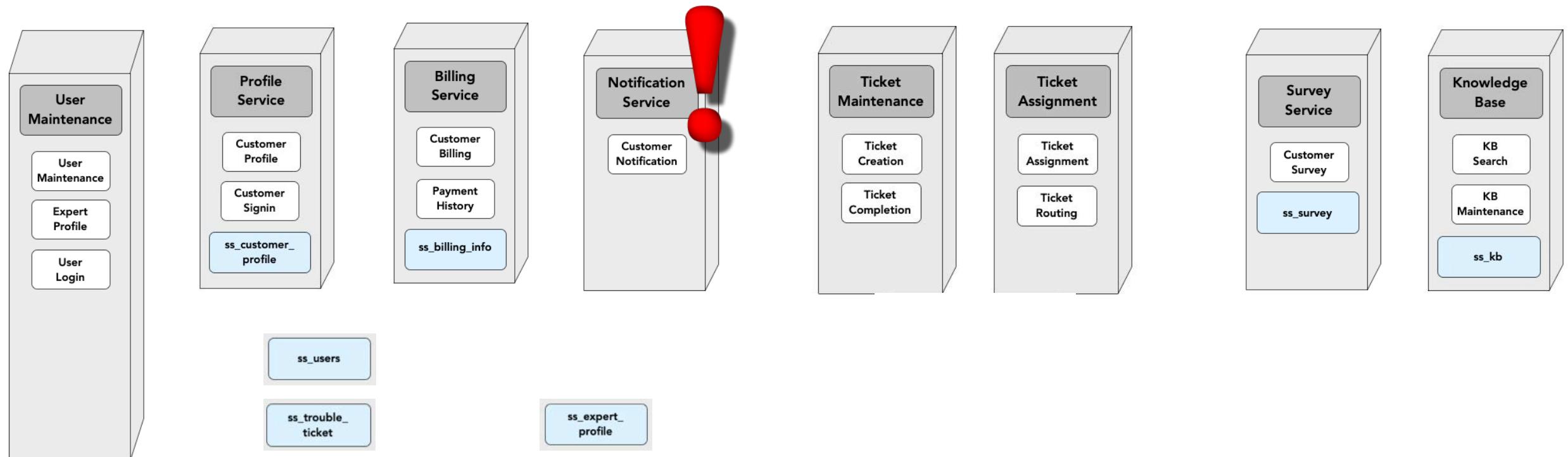
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



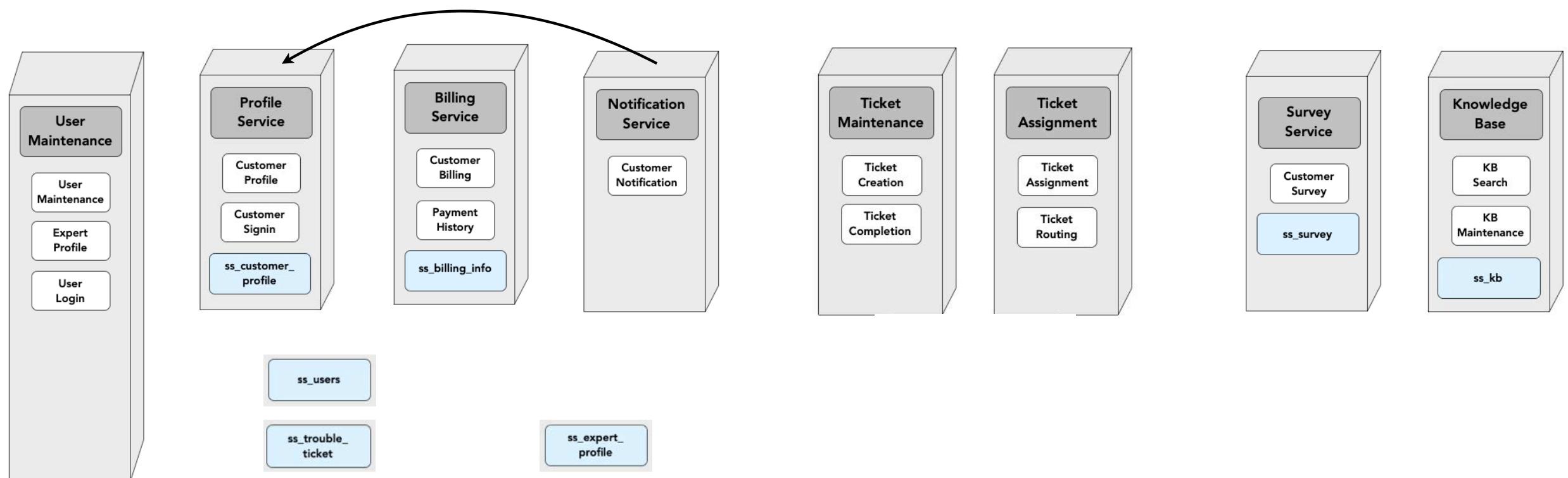
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



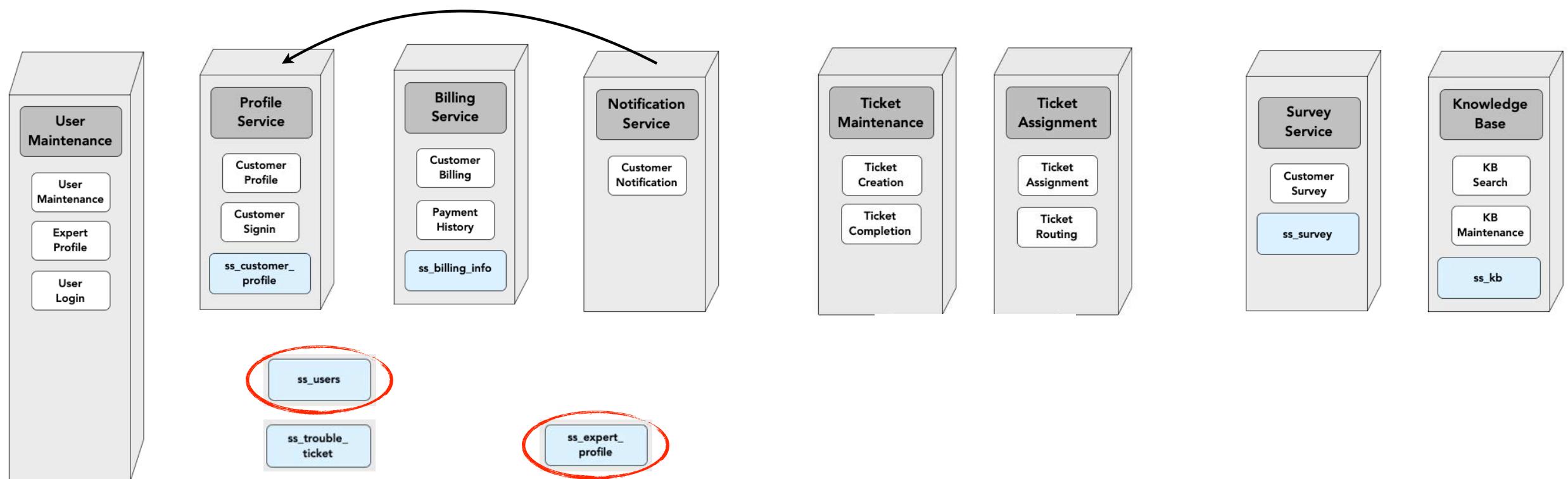
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



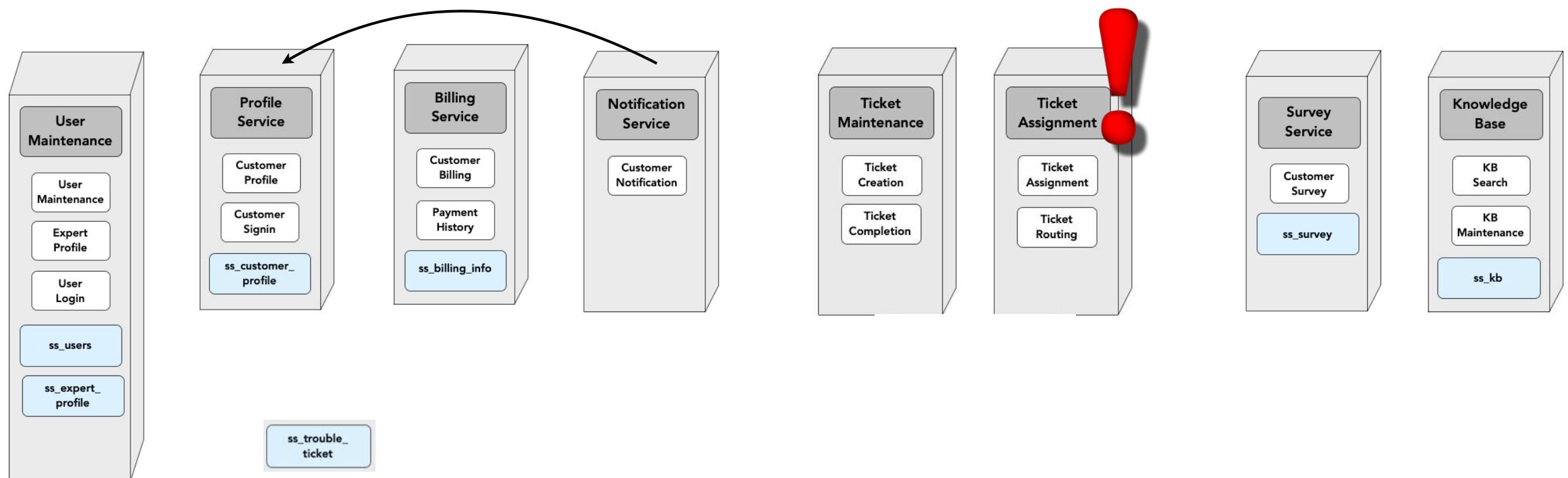
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



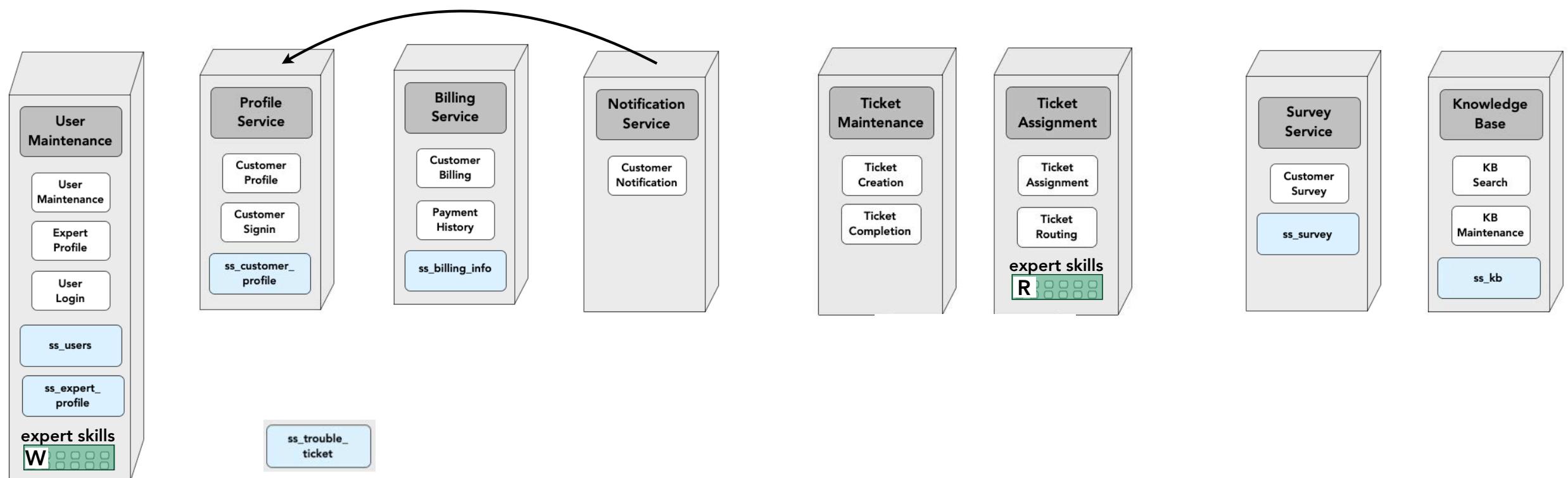
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



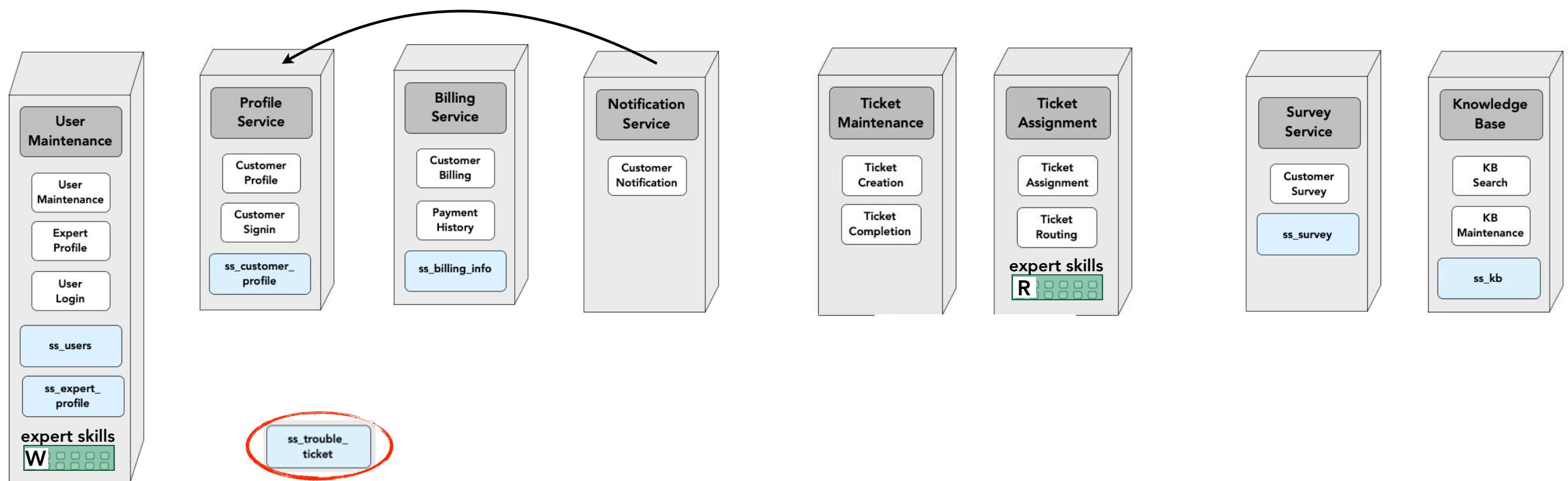
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



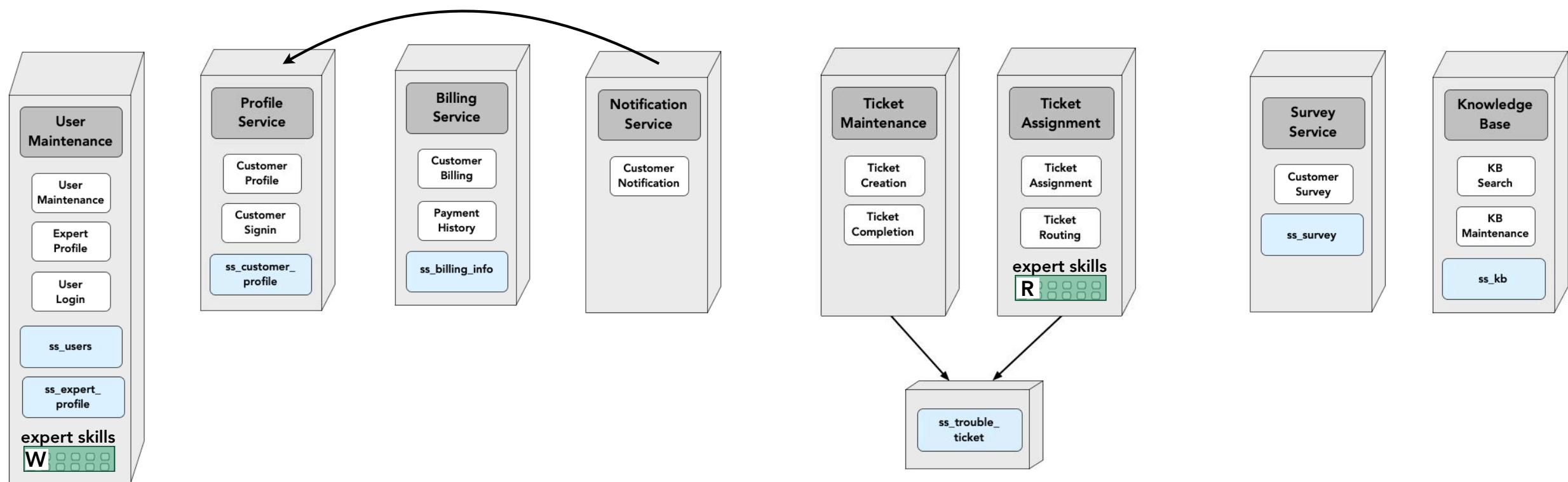
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.



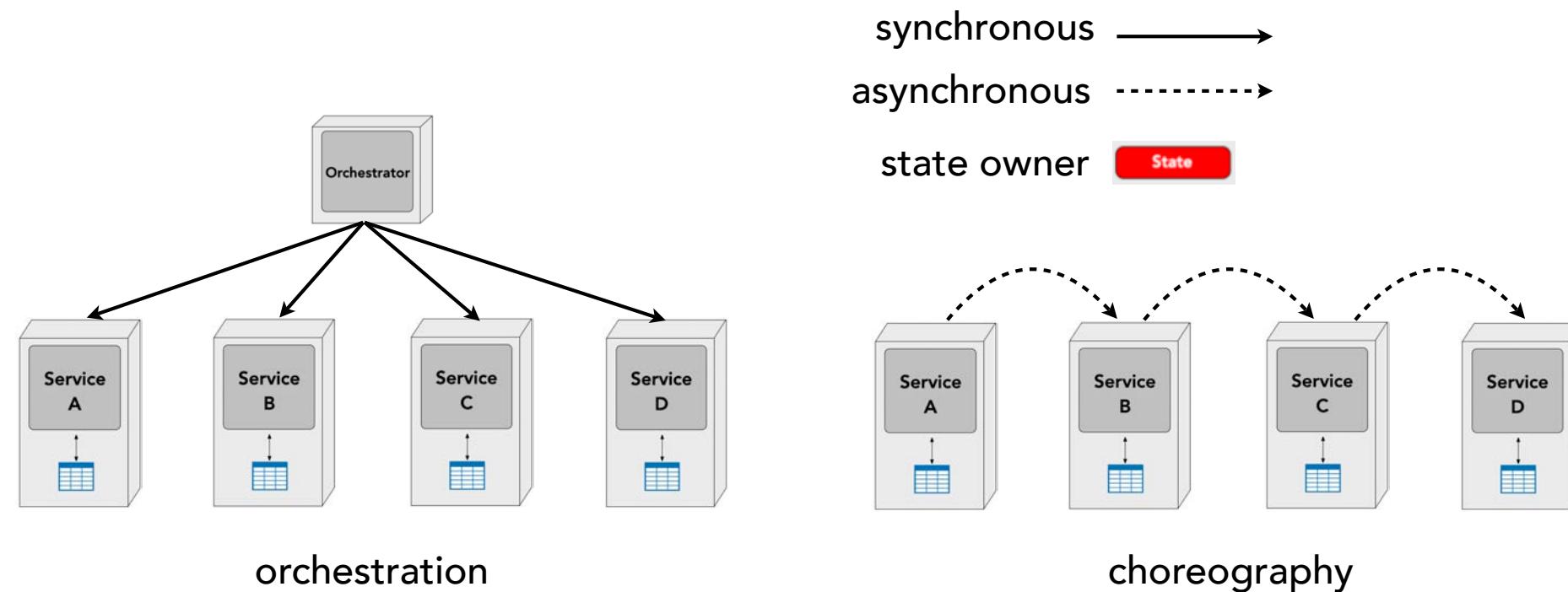
Kata Exercise - Data Ownership and Access

At this point, you now must decide what services should own data (does a write to the data) and how services are going to access data they don't own. Your choices are indicated below. Don't forget you can use any combination of these depending on the data.

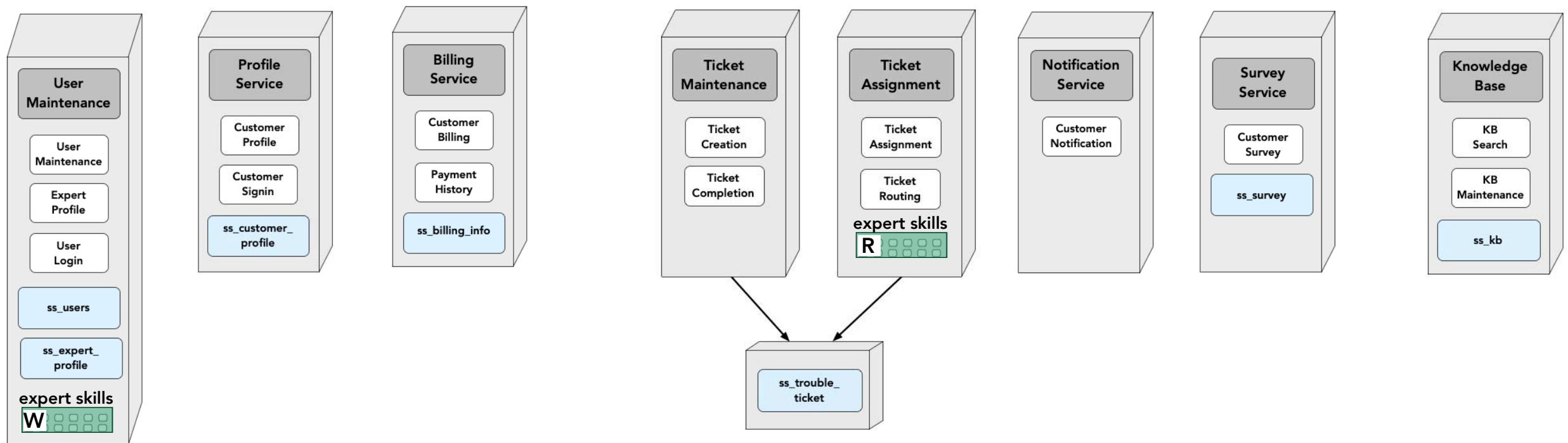


Kata Exercise - Workflow and Communications

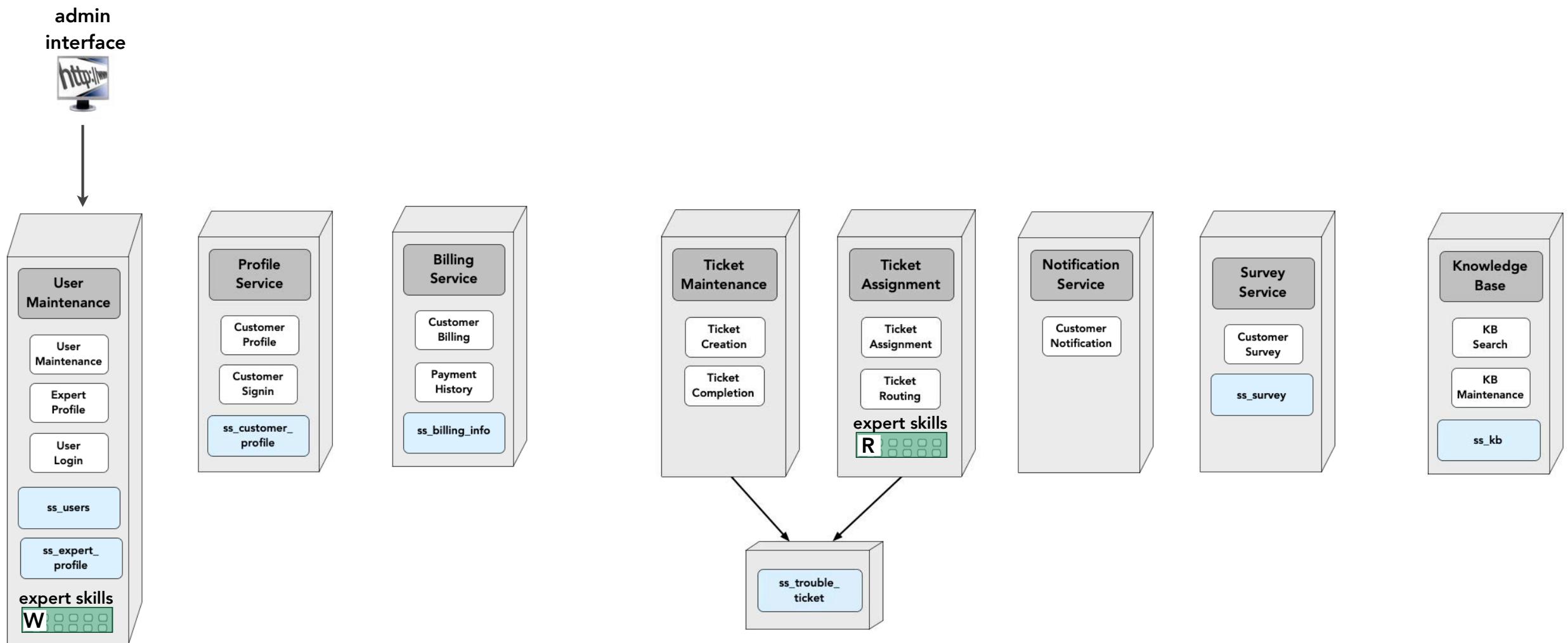
Now that you broke things apart, you now need to stitch them back together. Based on the workflows in the system, how should the services communicate with each other? Asynchronous or synchronous? Orchestration or choreography?



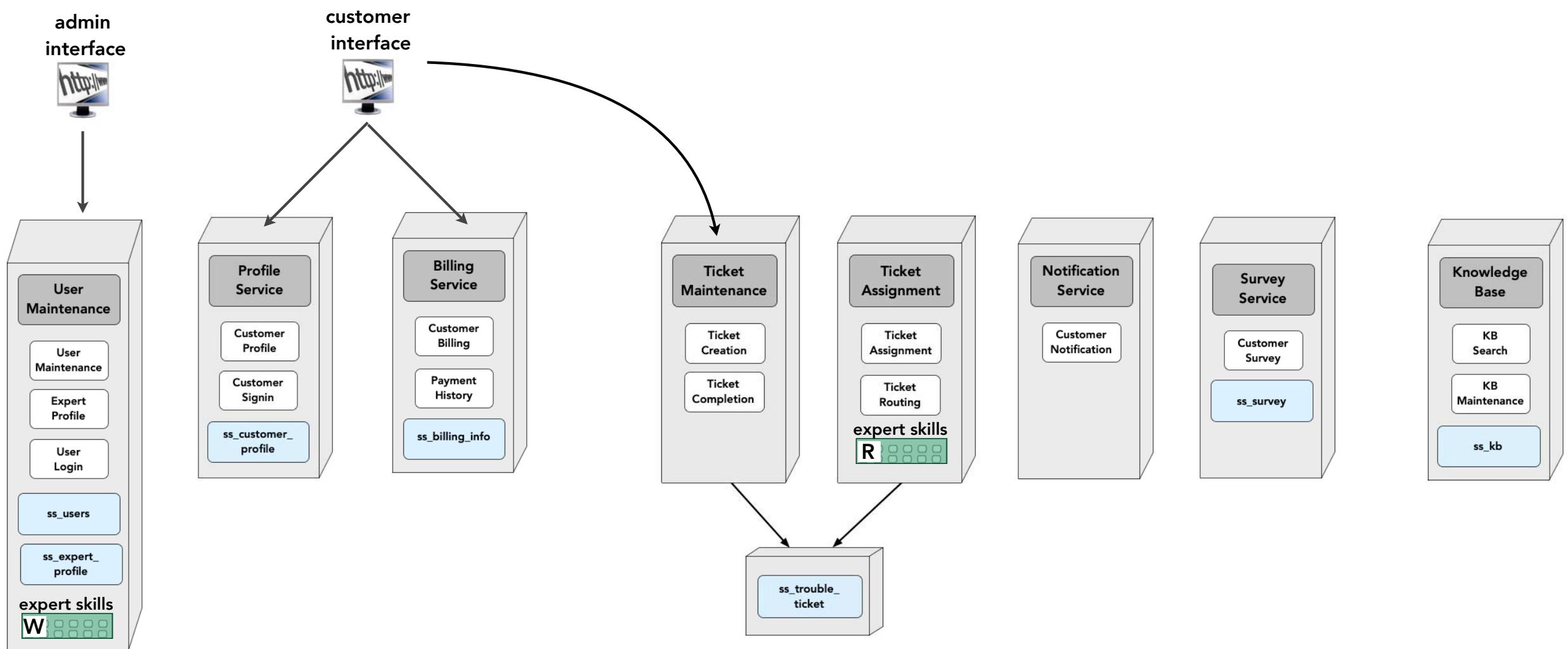
Kata Exercise - Workflow and Communications



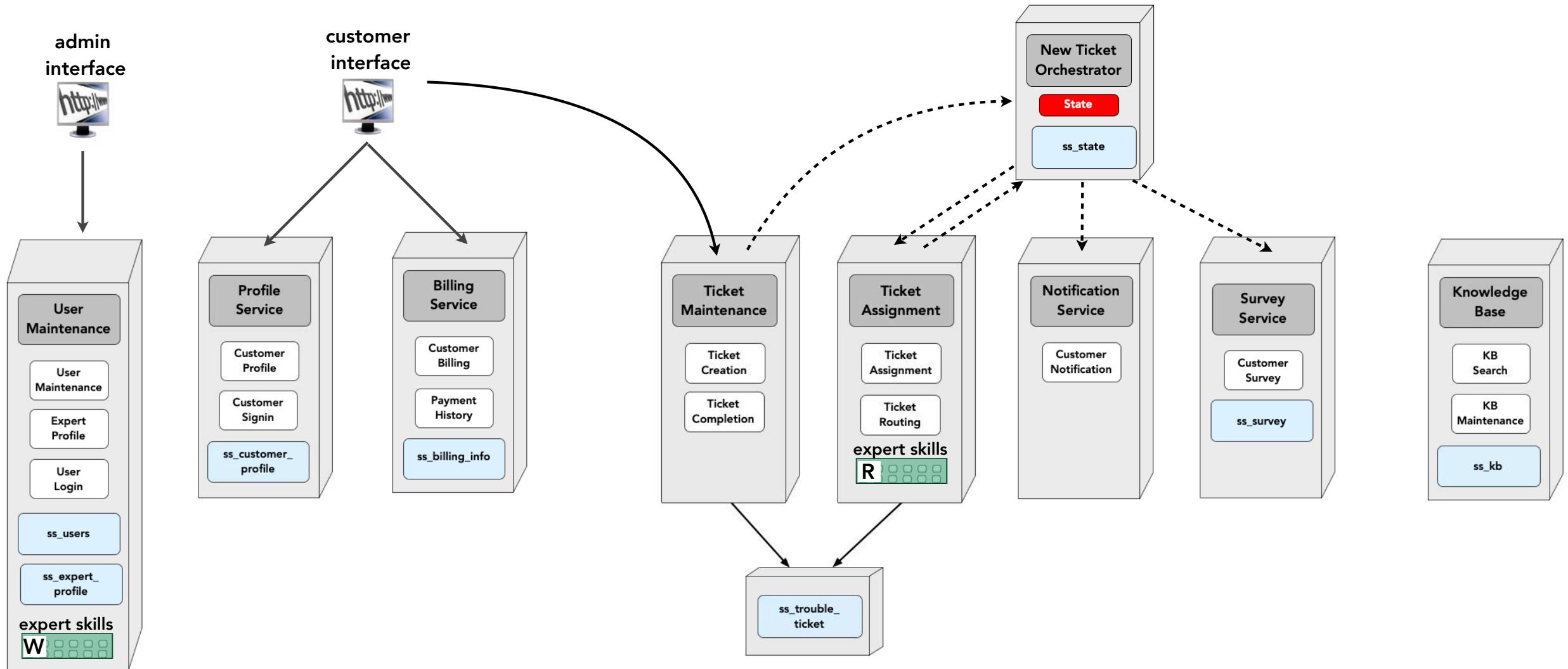
Kata Exercise - Workflow and Communications



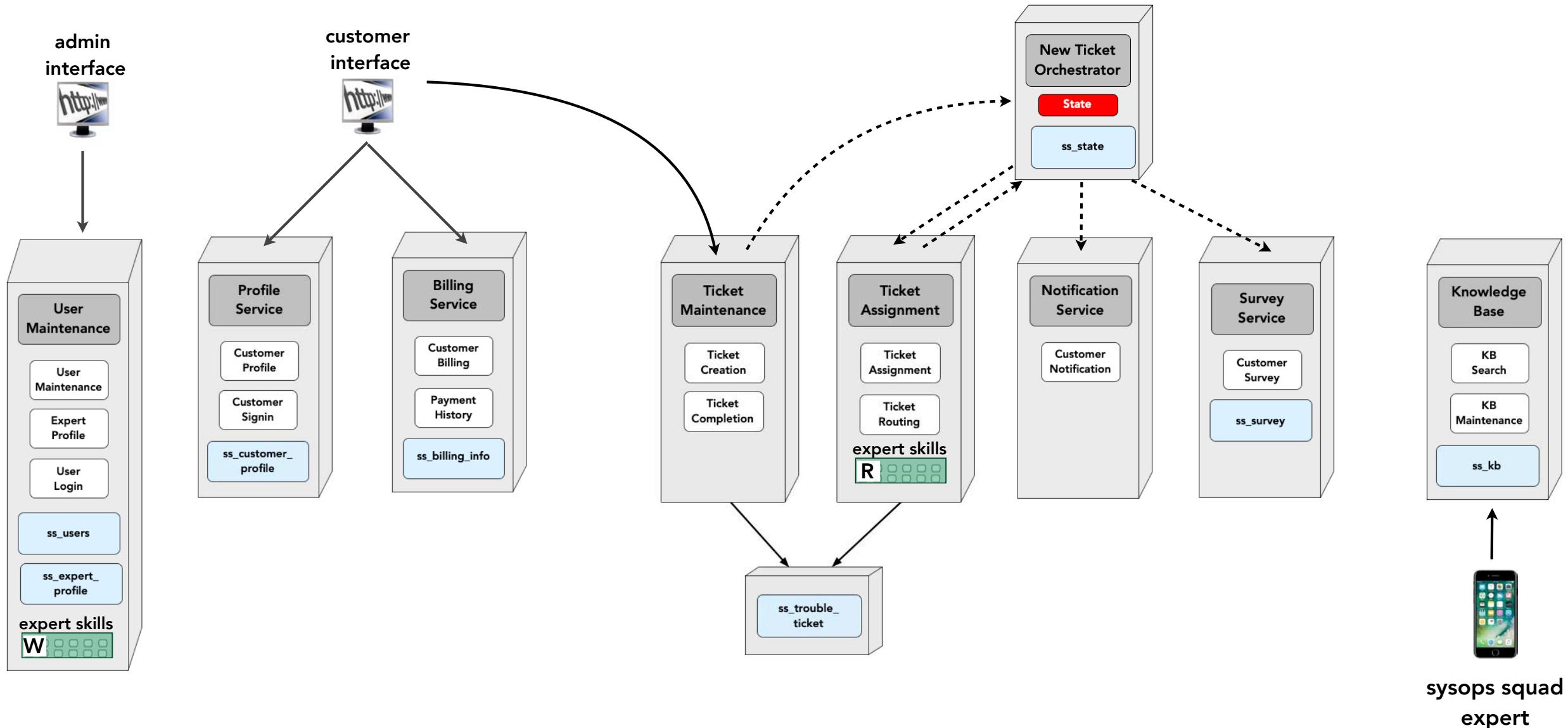
Kata Exercise - Workflow and Communications



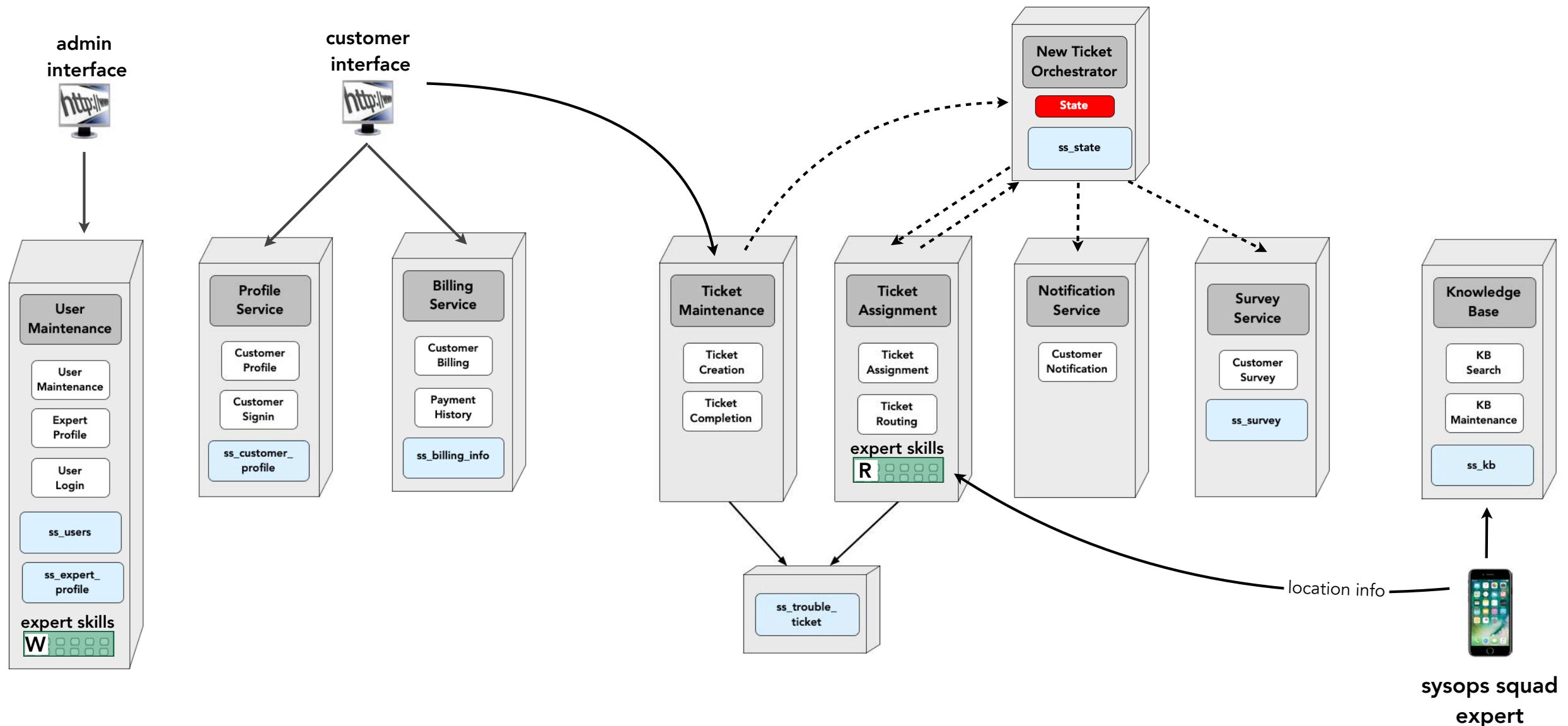
Kata Exercise - Workflow and Communications



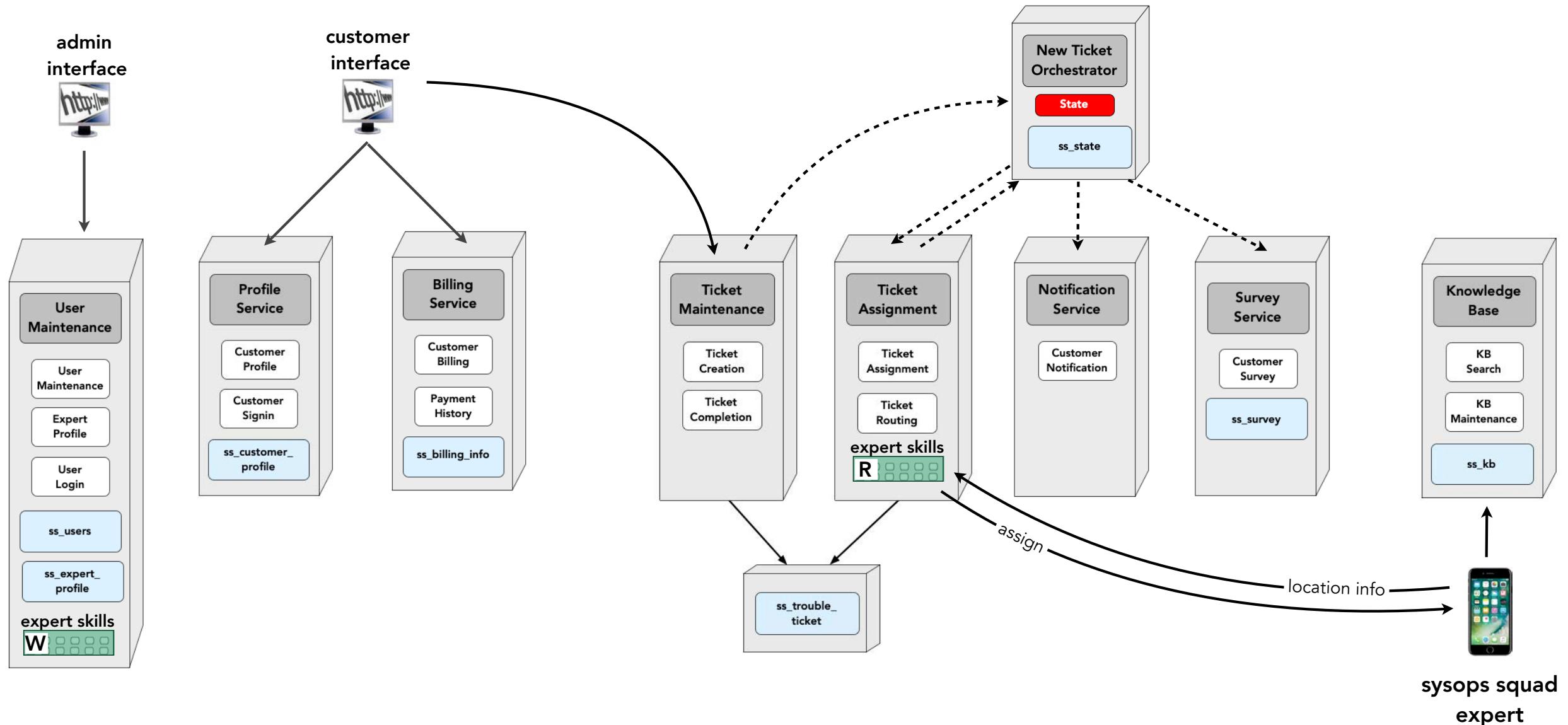
Kata Exercise - Workflow and Communications



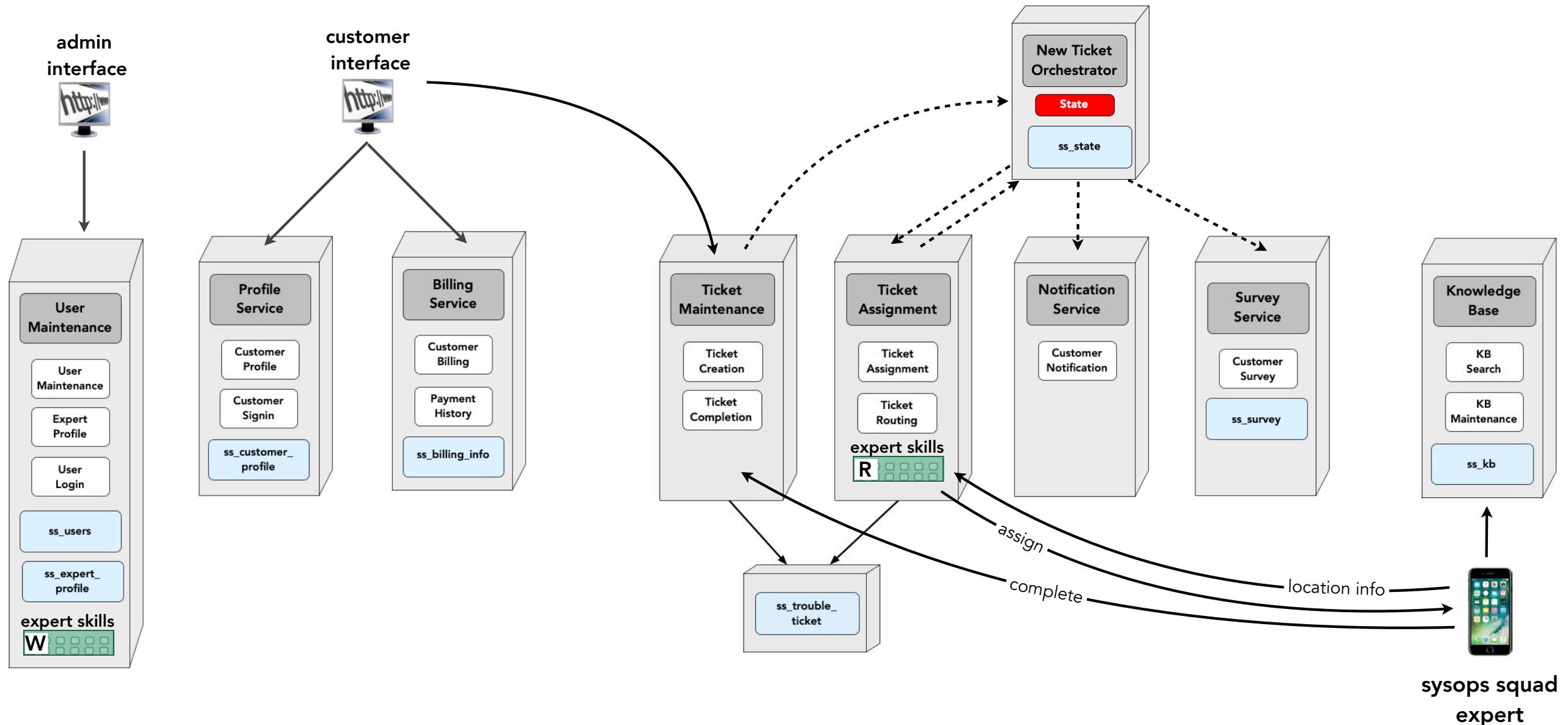
Kata Exercise - Workflow and Communications



Kata Exercise - Workflow and Communications

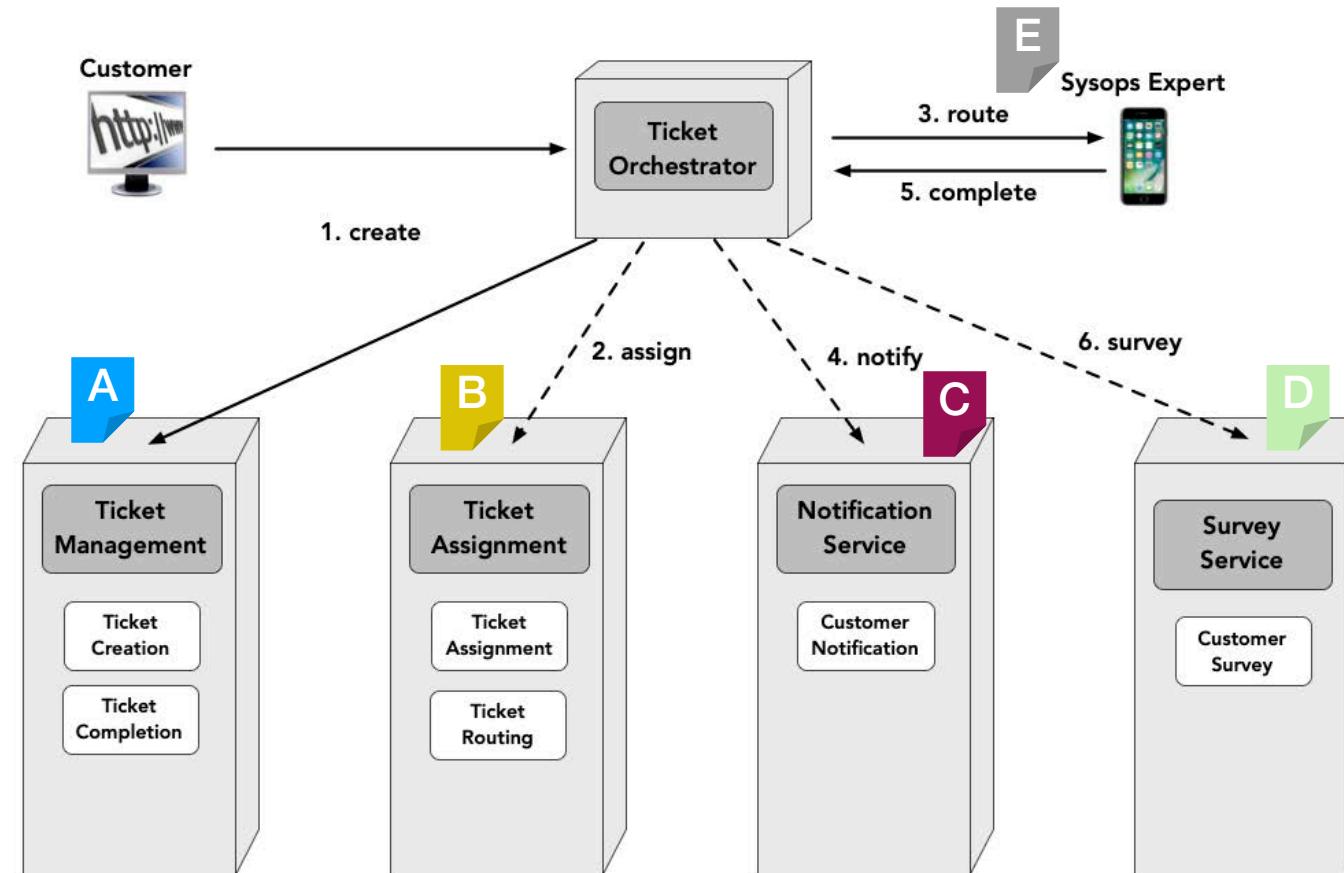


Kata Exercise - Workflow and Communications



Kata Exercise - Semantic vs. Syntactic Coupling

What type of contract (looser or tighter) should be applied to each of these interactions and why?



Kata Exercise - Semantic vs. Syntactic Coupling

What type of contract (looser or tighter) should be applied to each of these interactions and why?

