

Communication Styles for Distributed Architectures and Microservices

ThoughtWorks®

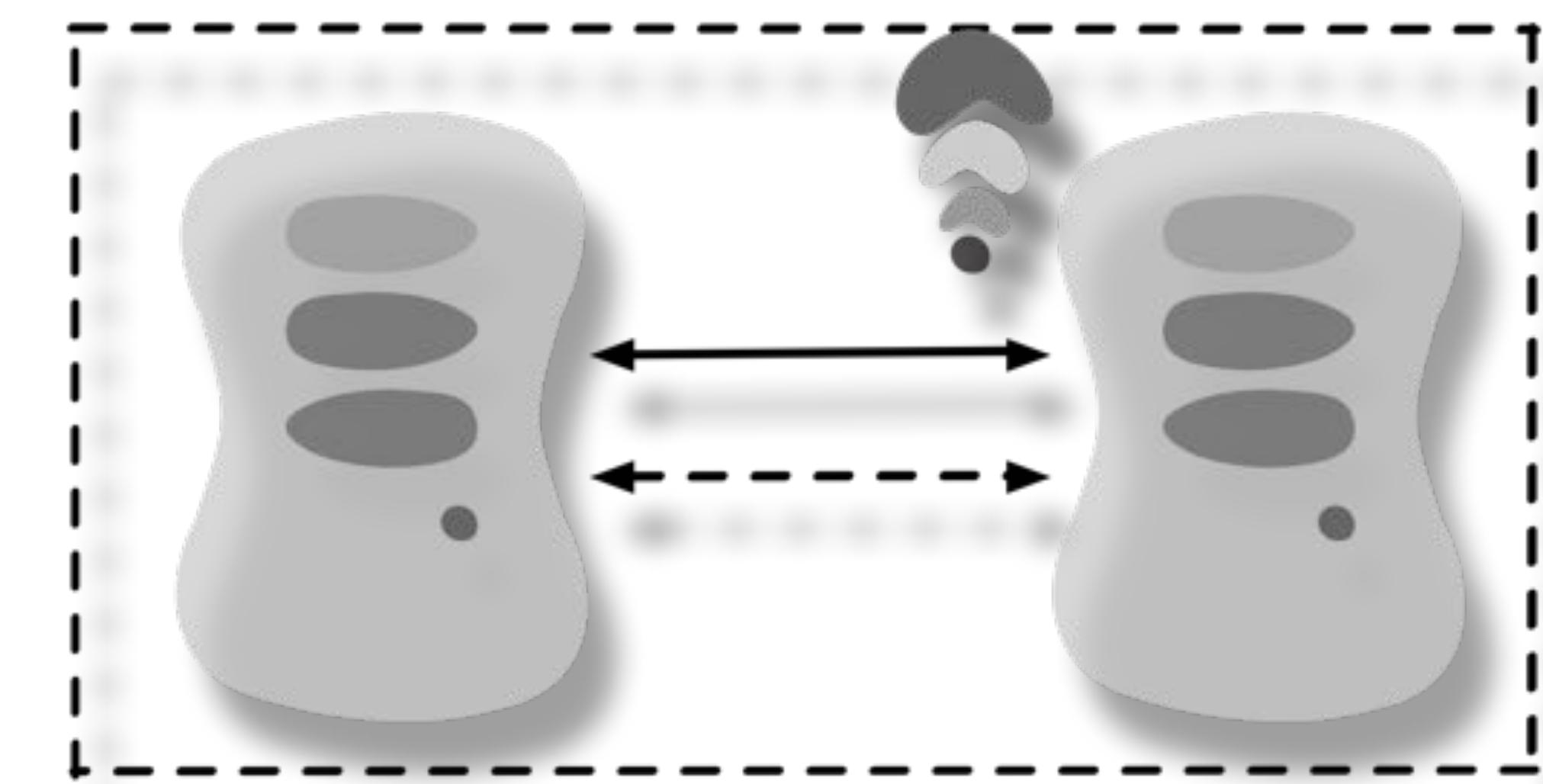
NEAL FORD

Director / Software Architect / Meme Wrangler



@neal4d

<http://nealford.com>



(this | other) course ?

?

?

?

?

?

!

.

!

.

?

?

?

?

?

?

?

?

?

?

?

?

!

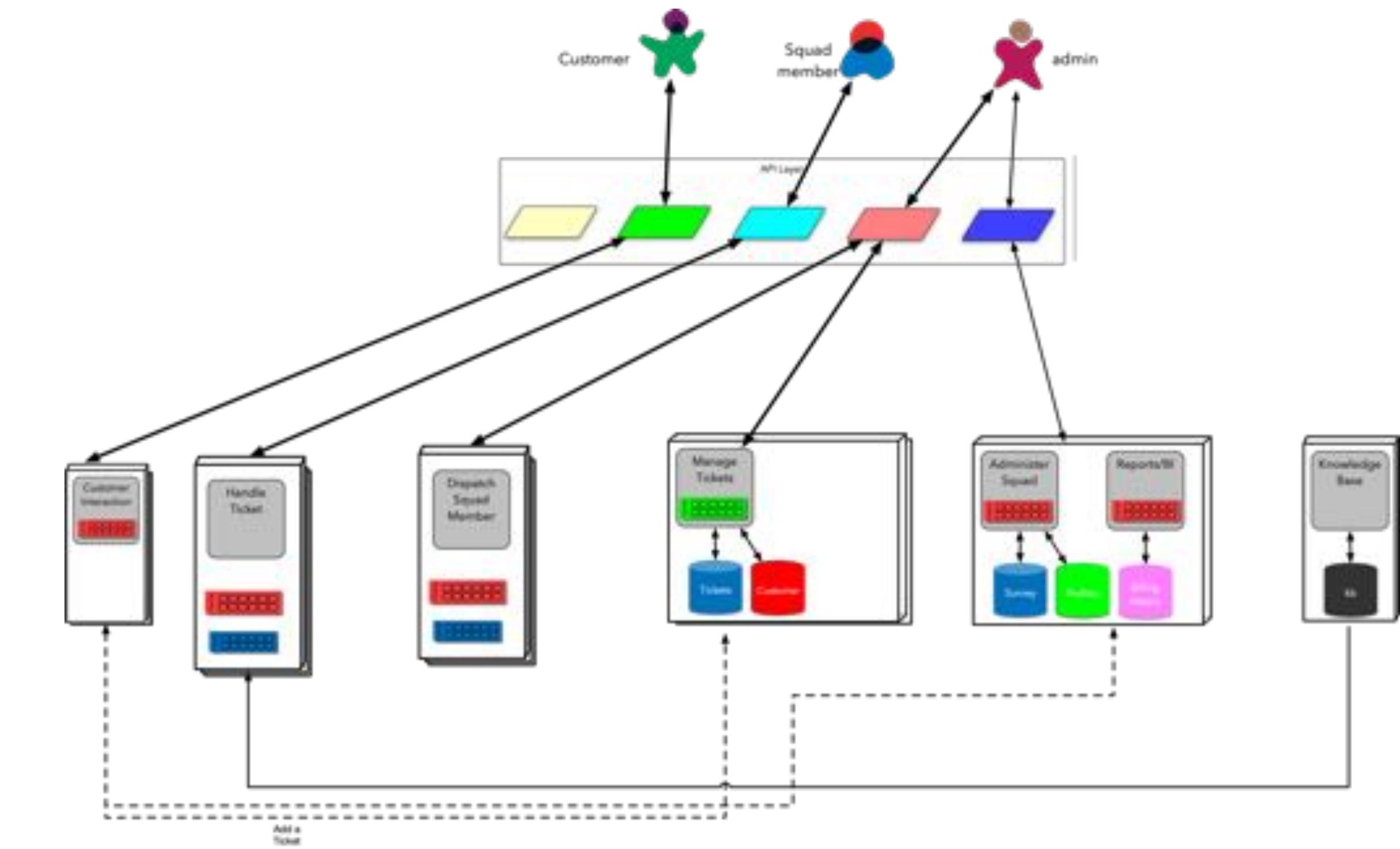
.

this ***OR*** that

tradeoffs

a single thing

abstract => implementation



agenda

Communication Styles for Distributed Architectures and Microservices

introduction

monolith | distributed ?

synchronous | asynchronous ?

(semantic | syntactic) coupling ?

EDA | microservices ?

contracts | versions ?

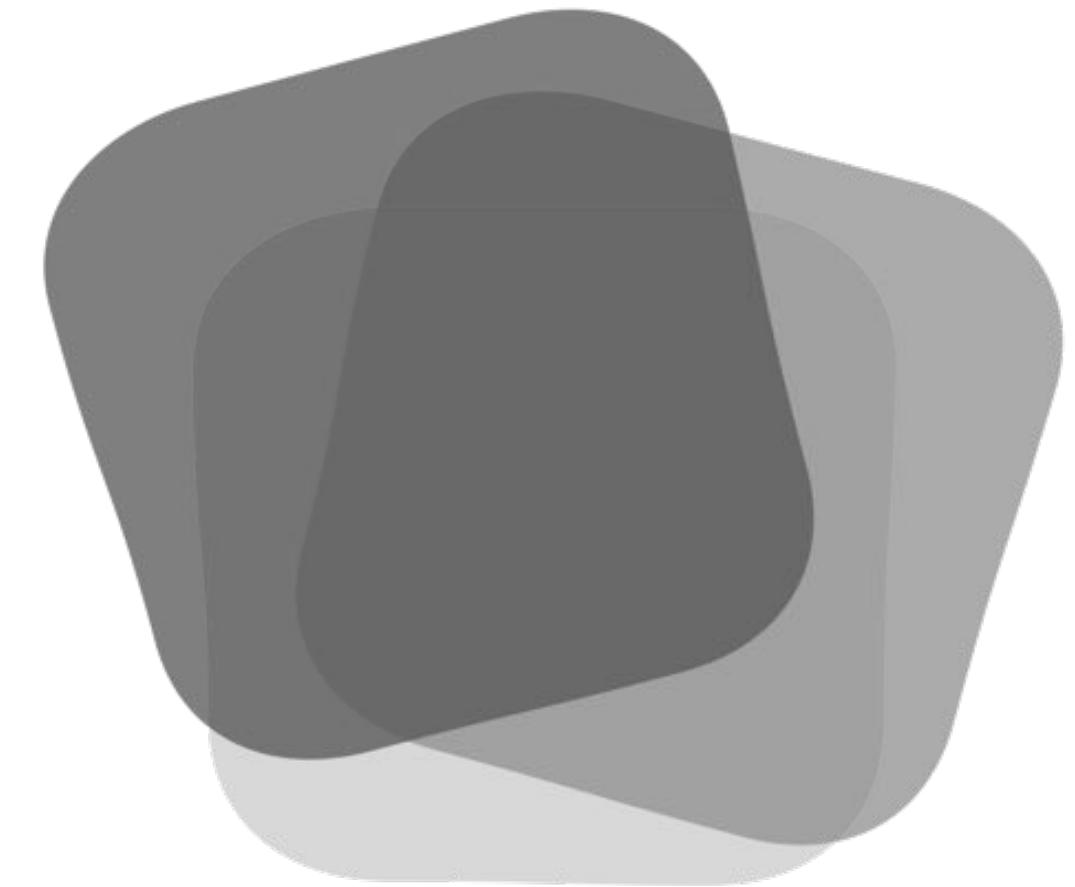
choreography | orchestration |
synchronization | write-ahead
log | sagas ?

(operational | analytical) data ?

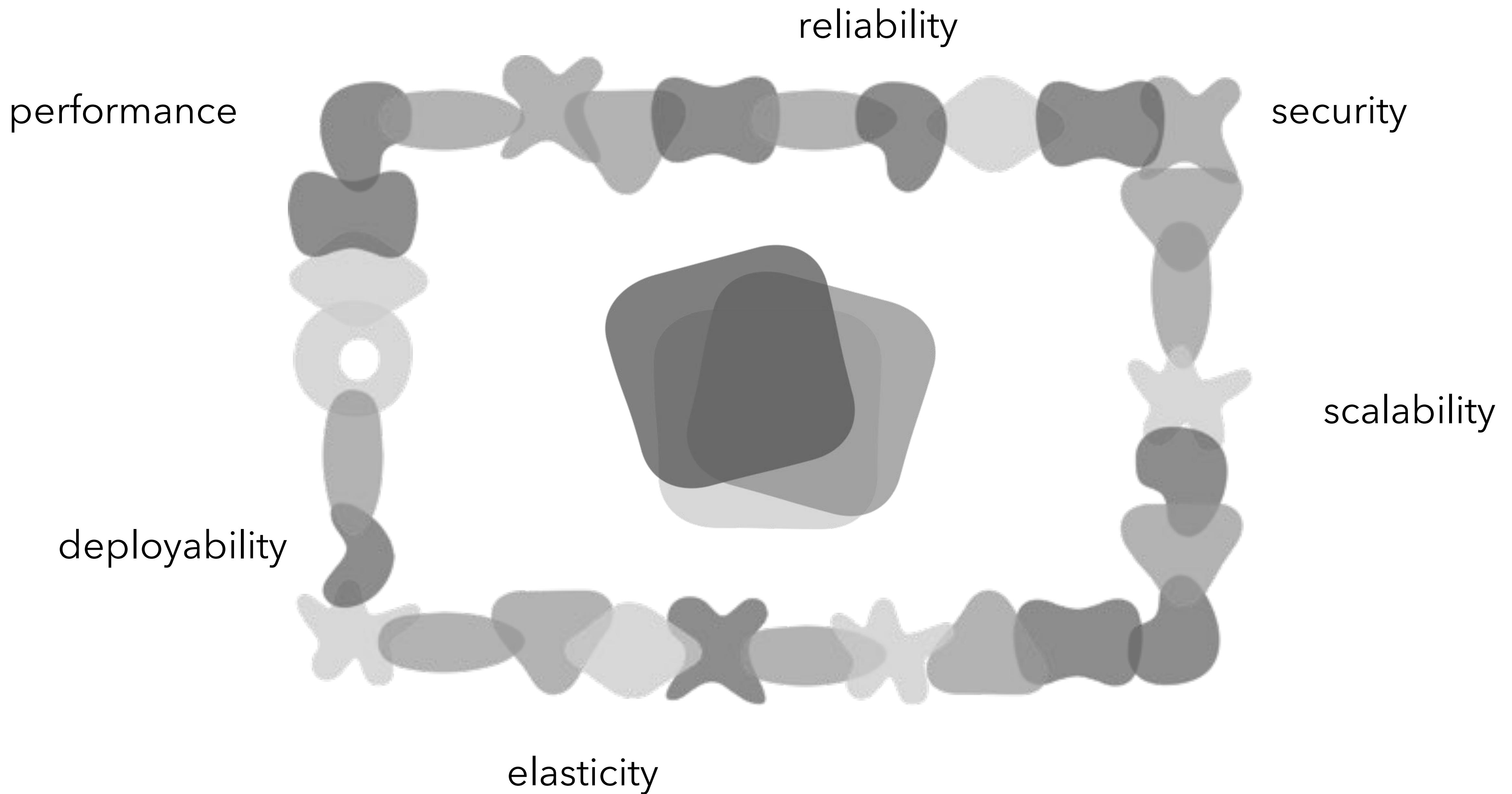
monolith | distributed ?

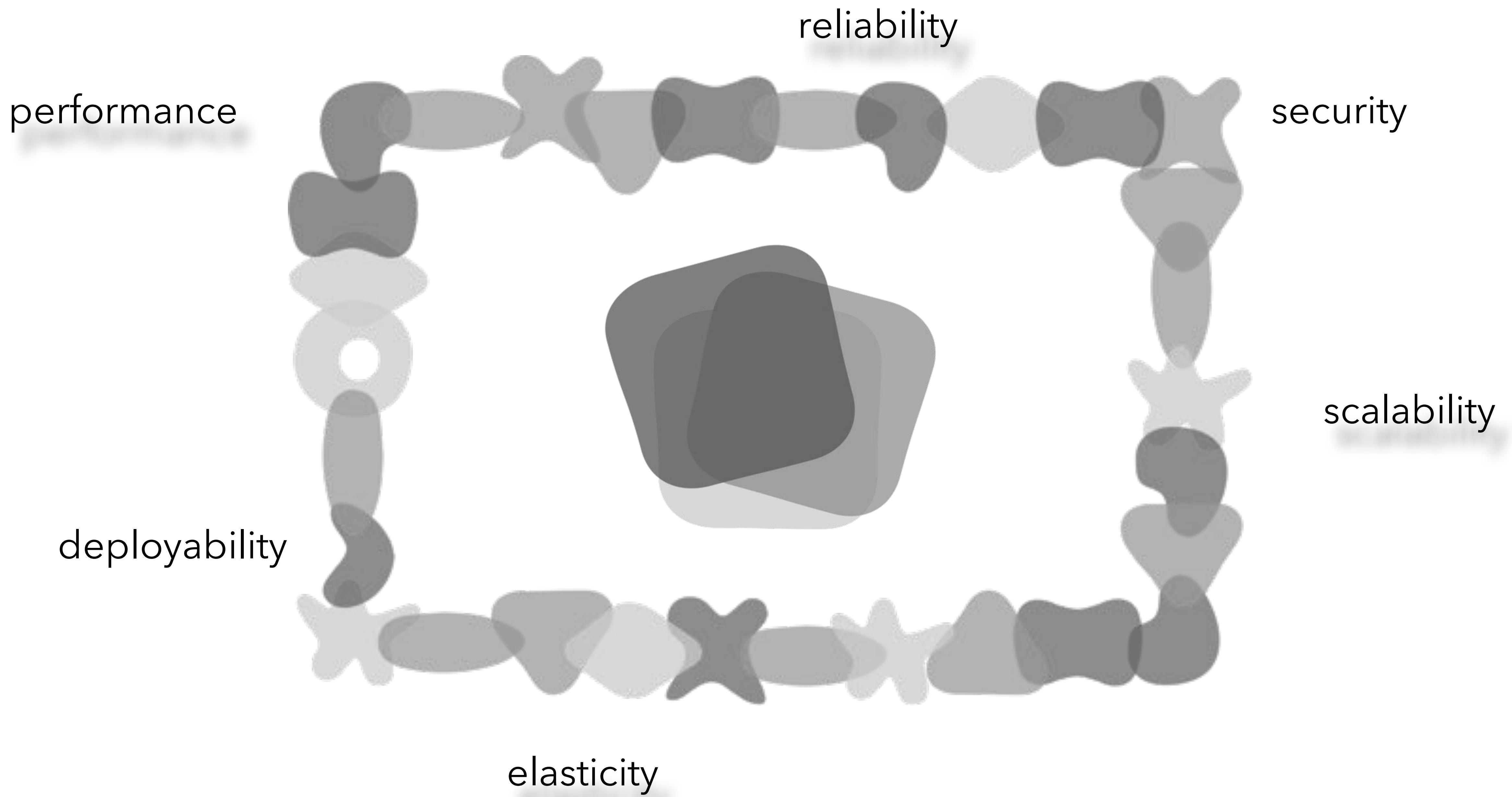
structural design in architecture

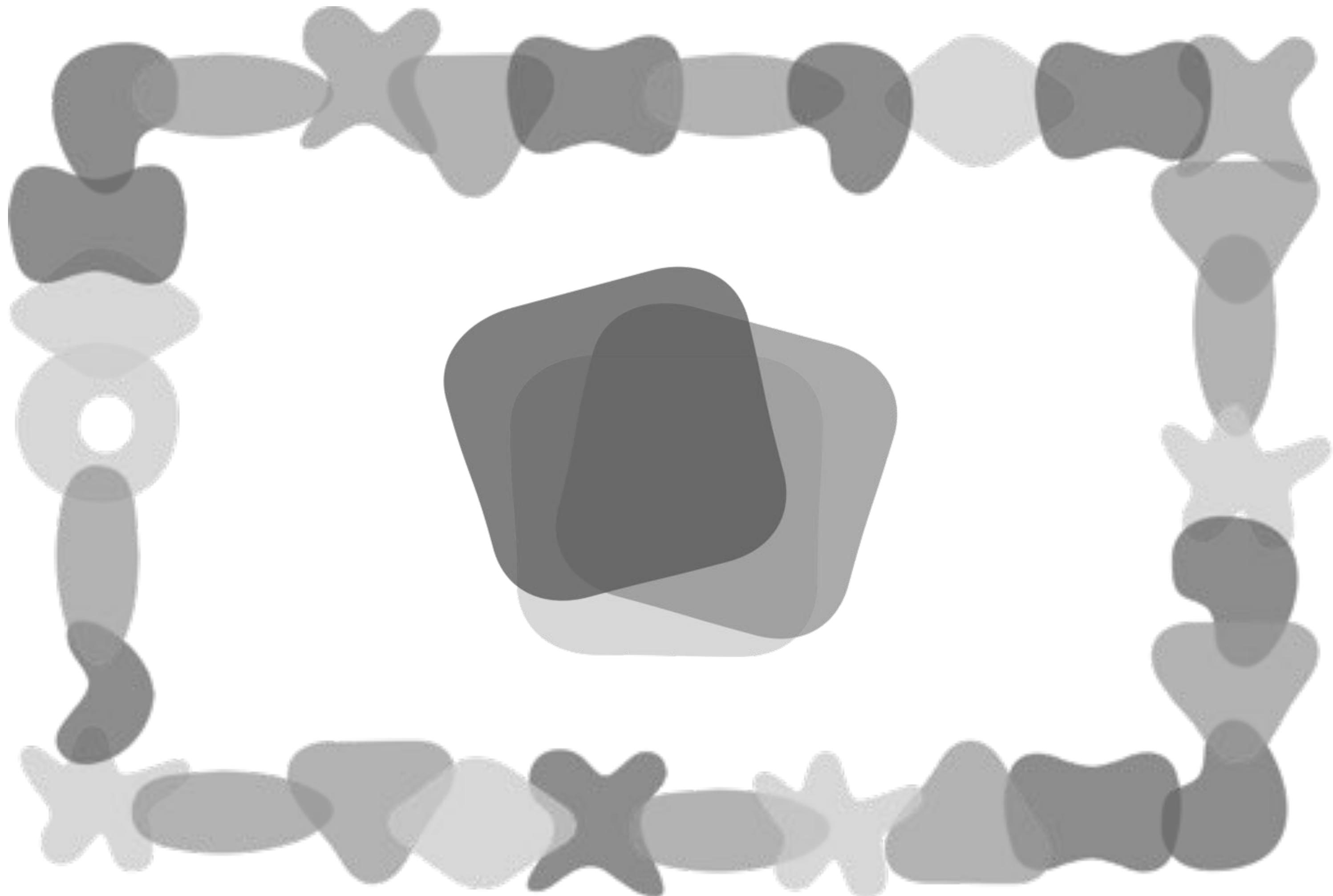


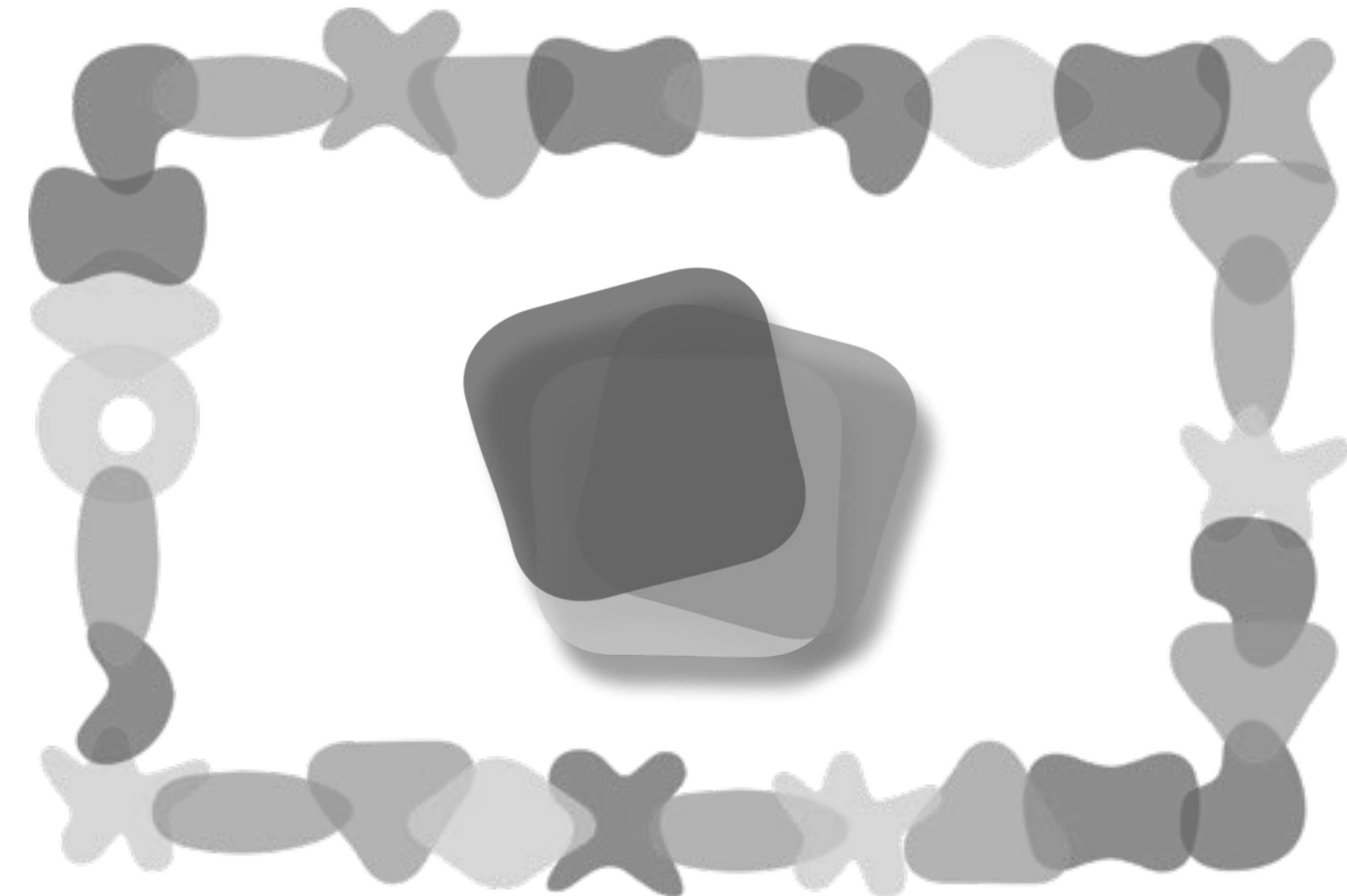


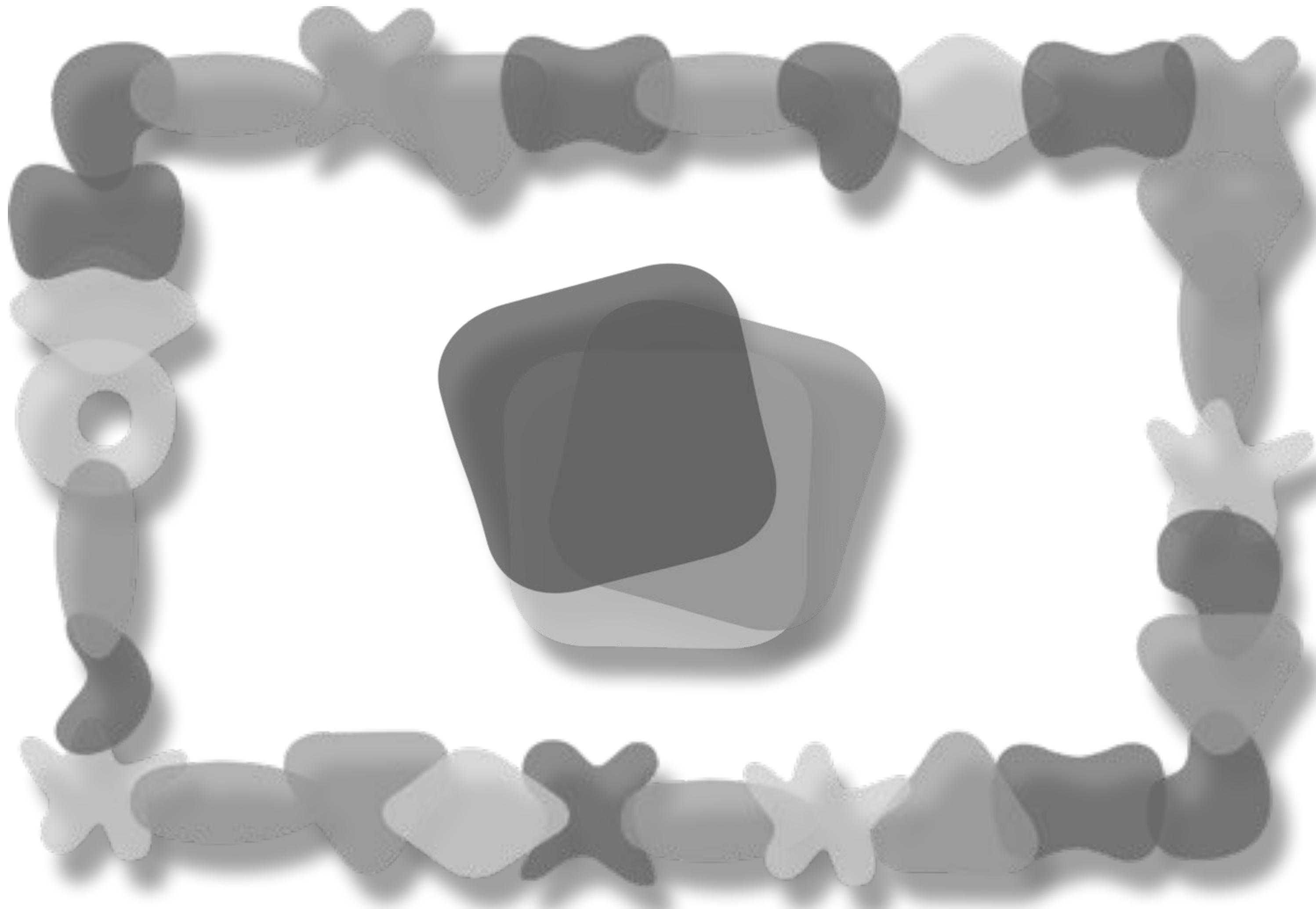
requirements | use cases | story cards | DDD event-storm output | ?



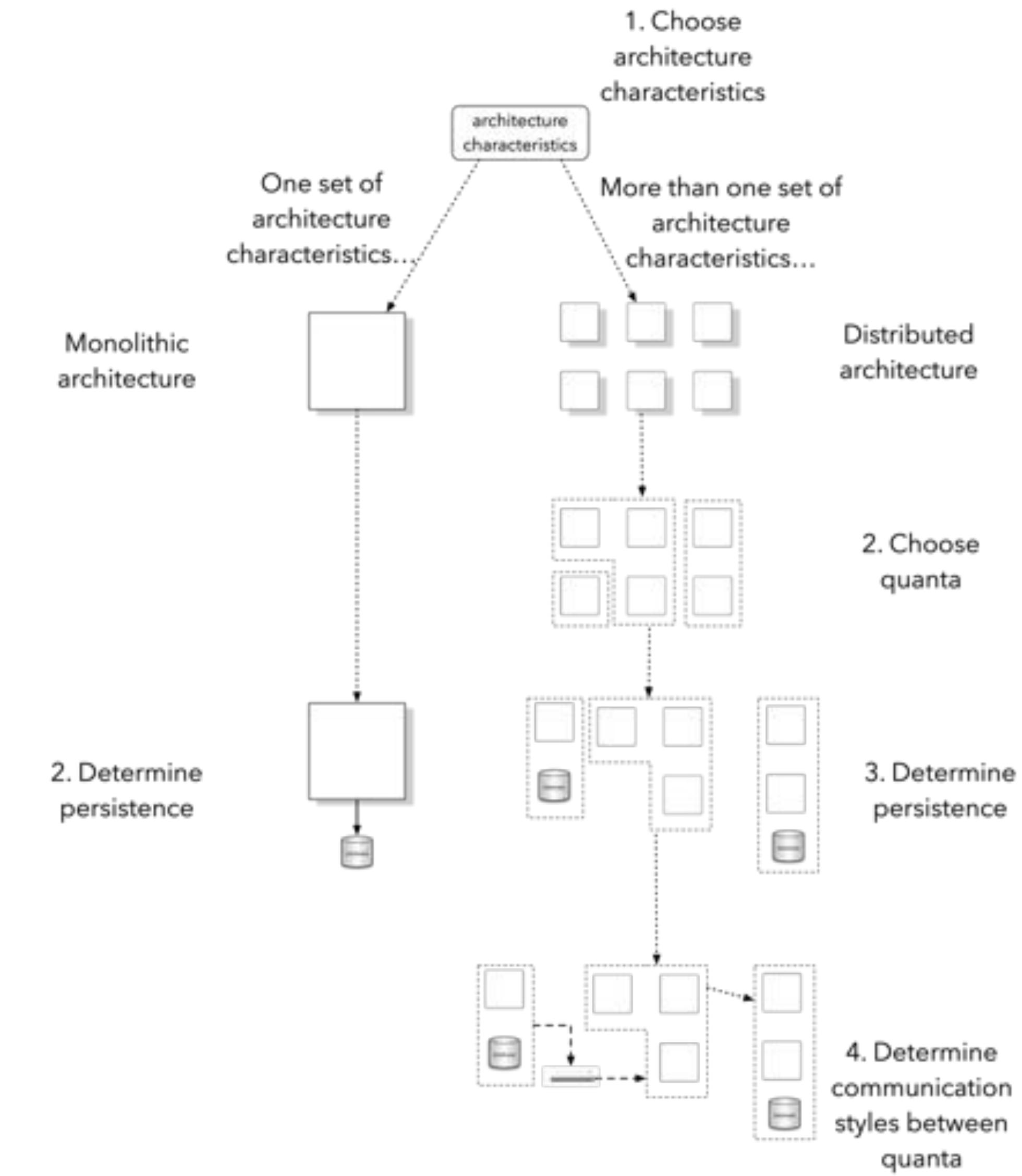




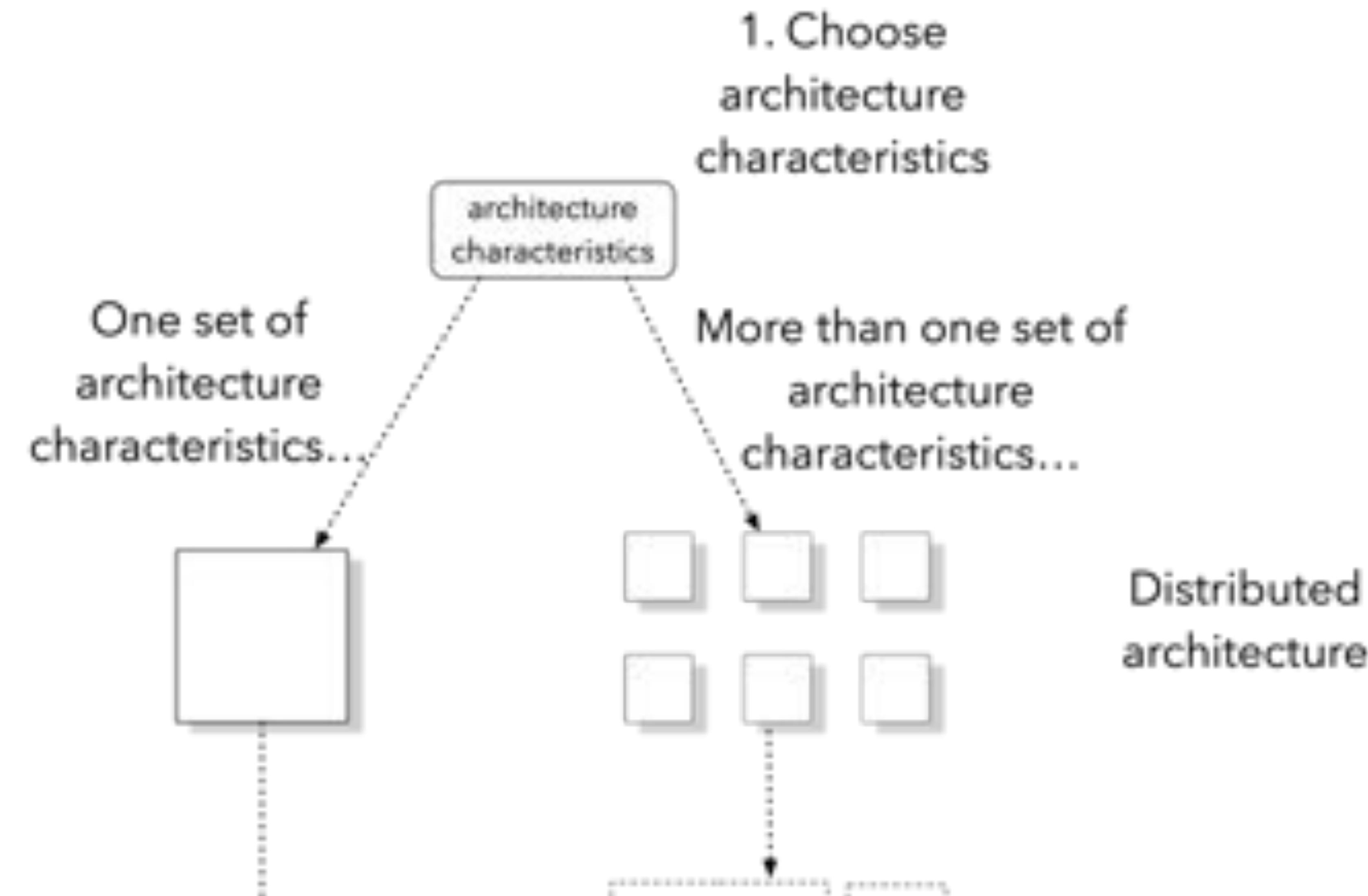




monolith | distributed ?



monolith | distributed?



Your Architectural Kata is...

Going Going Gone!

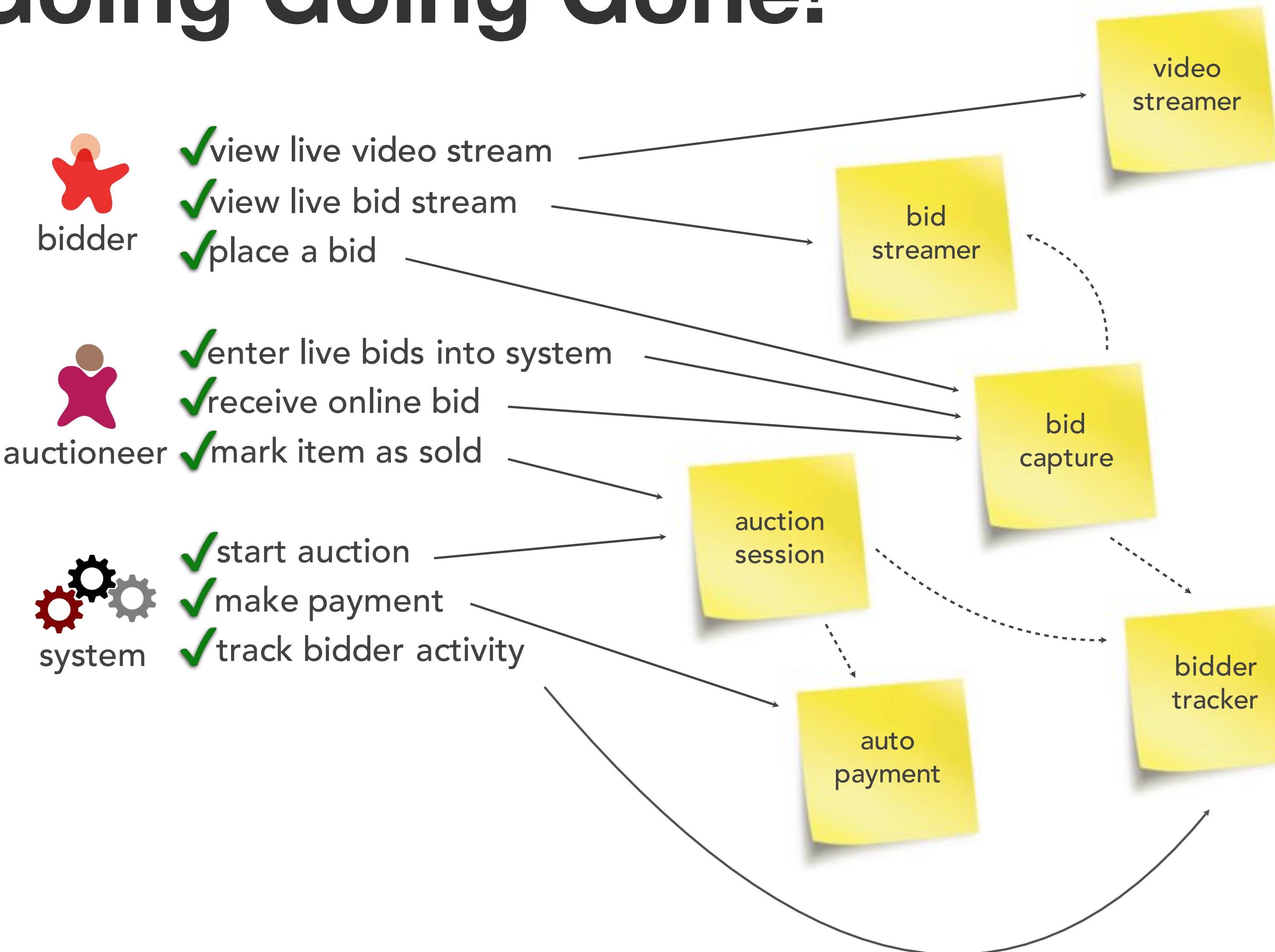
An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **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

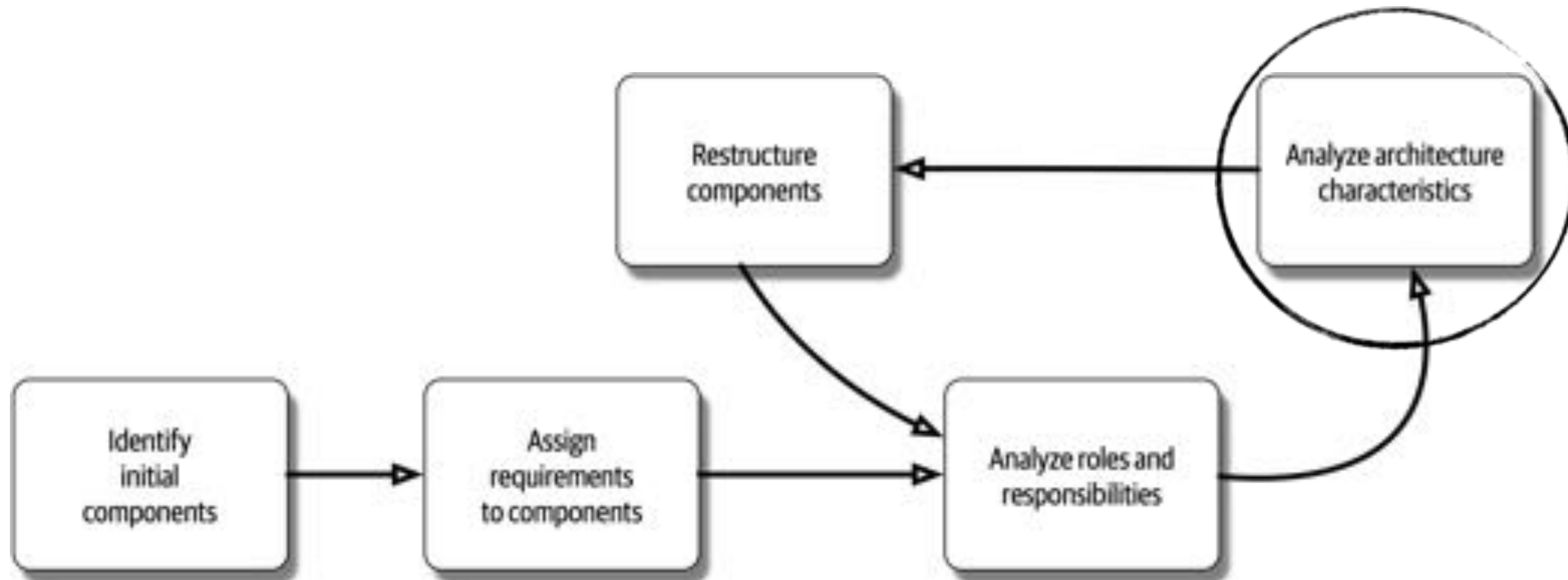
Your Architectural Kata is...

Going Going Gone!



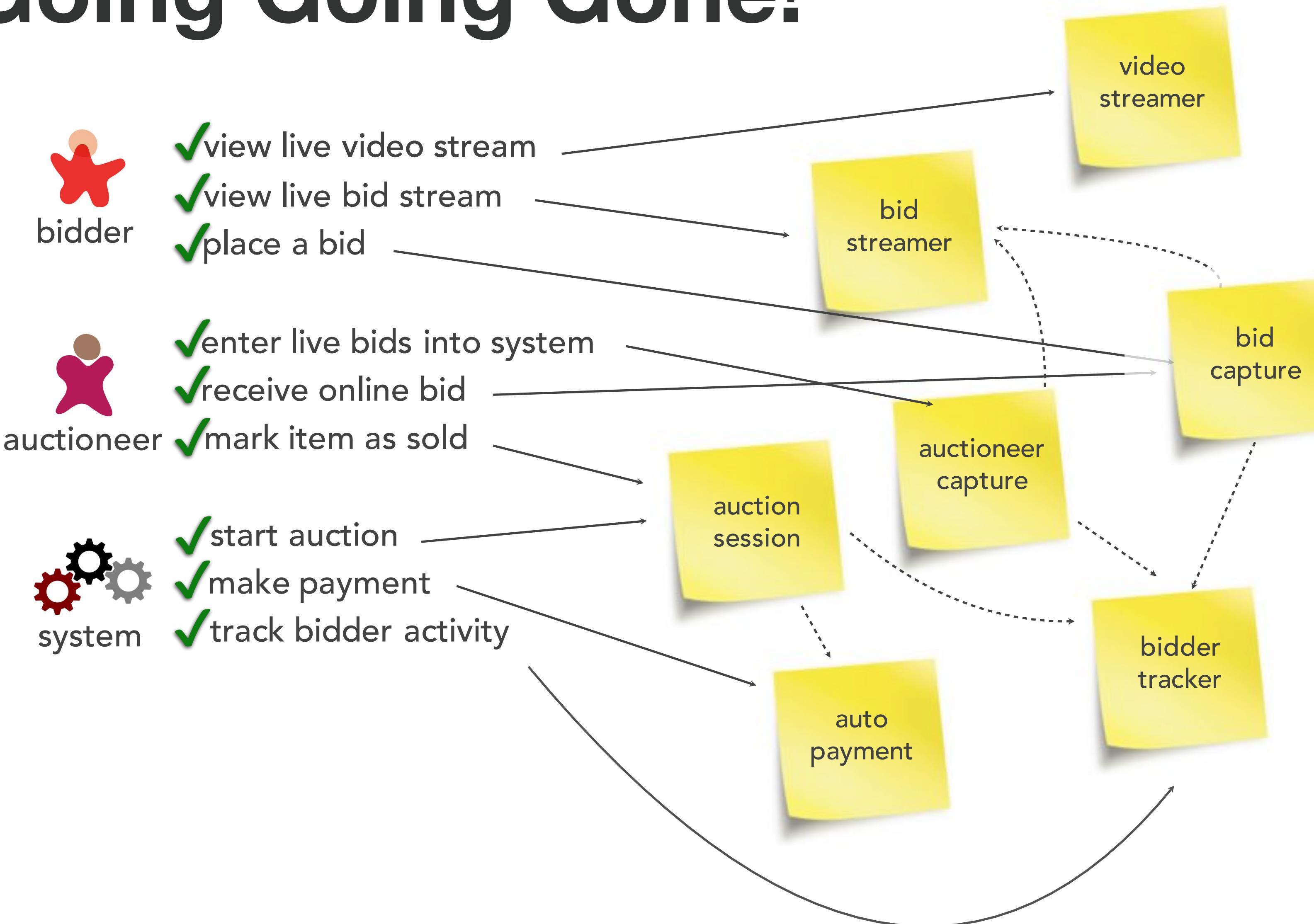
Your Architectural Kata is...

Going Going Gone!

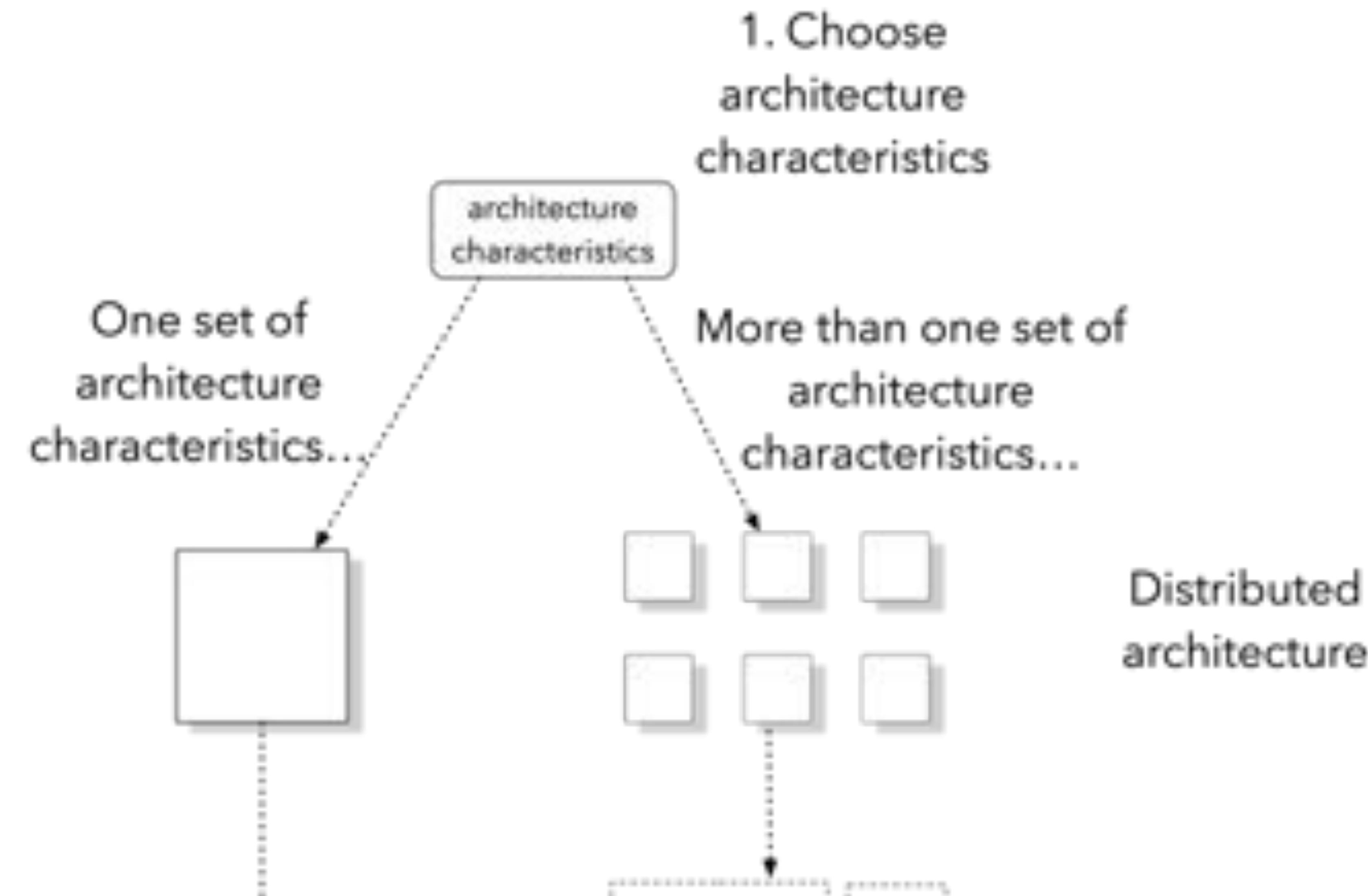


Your Architectural Kata is...

Going Going Gone!



monolith | distributed?



monolith | distributed?

Monolithic
architecture

One set of
architecture
characteristics...

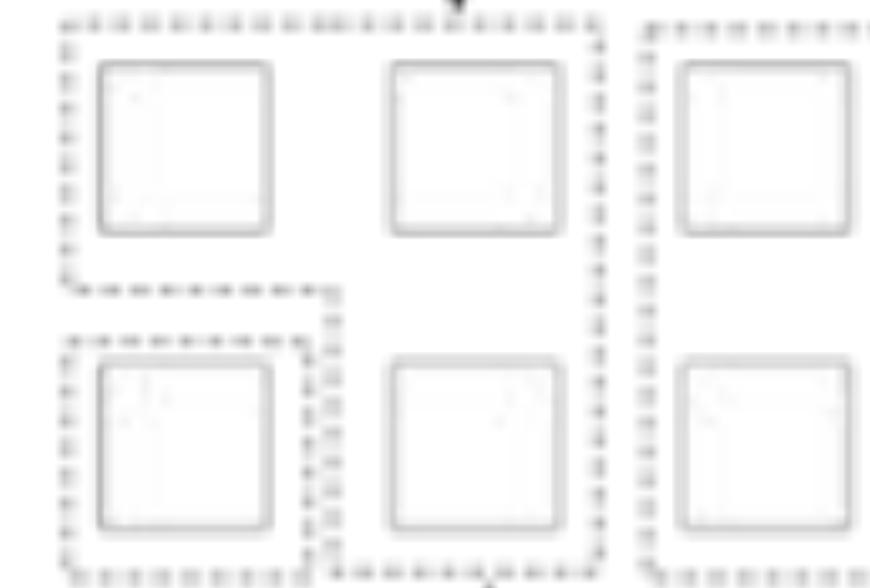


architecture
characteristics

More than one set of
architecture
characteristics...

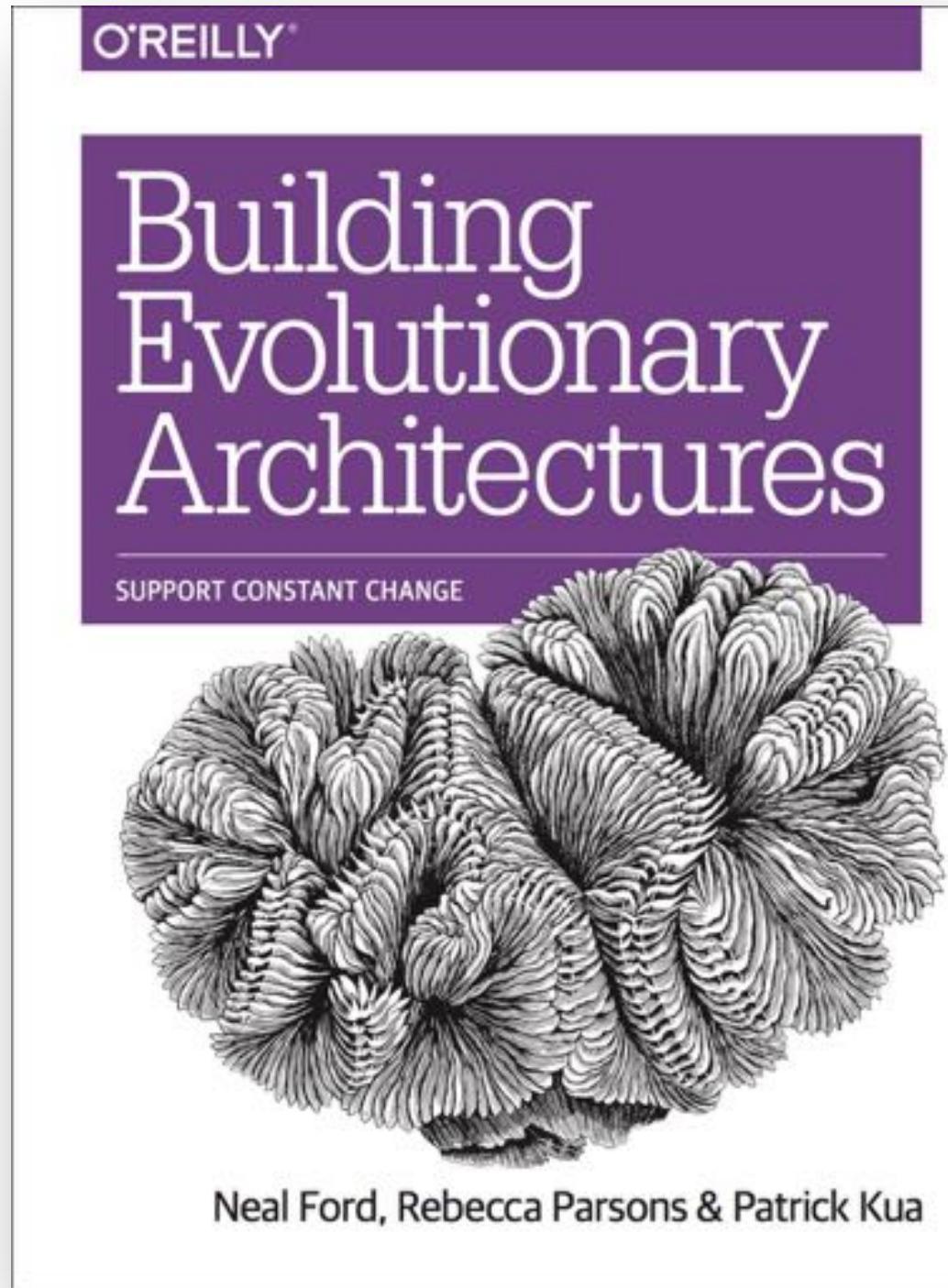


Distributed
architecture



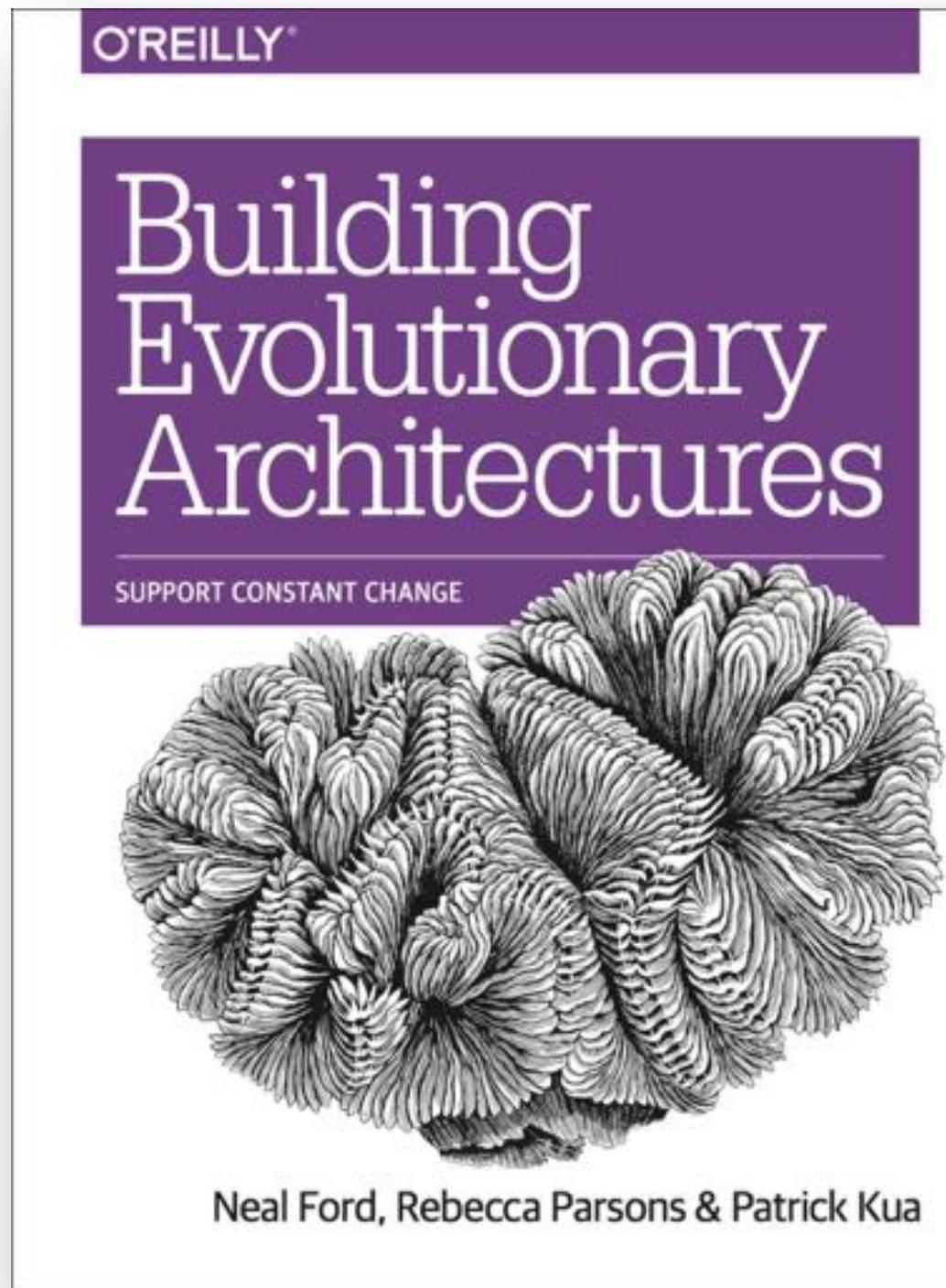
2. Choose
quanta

architectural quantum



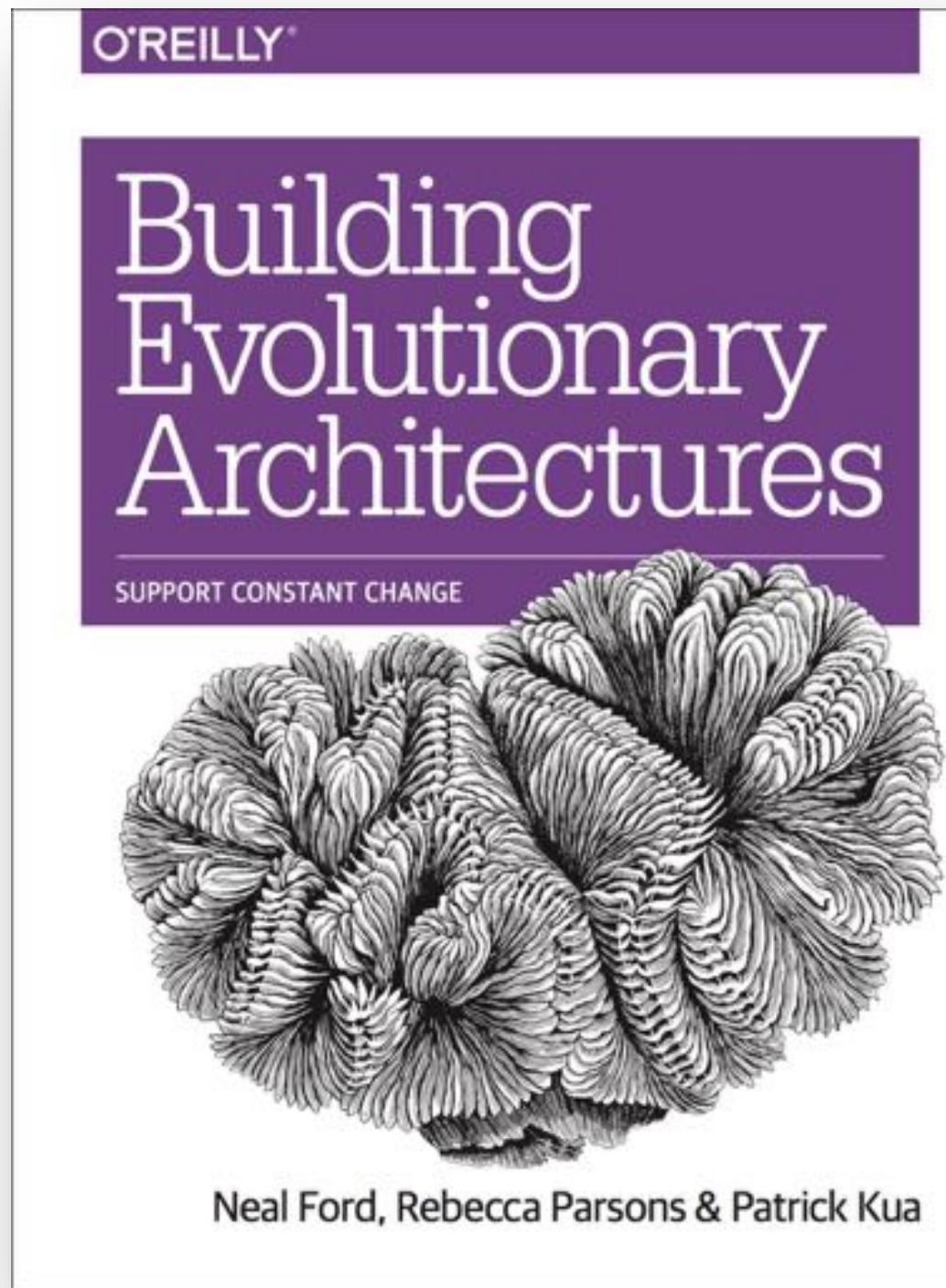
An *architectural quantum* is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

architectural quantum



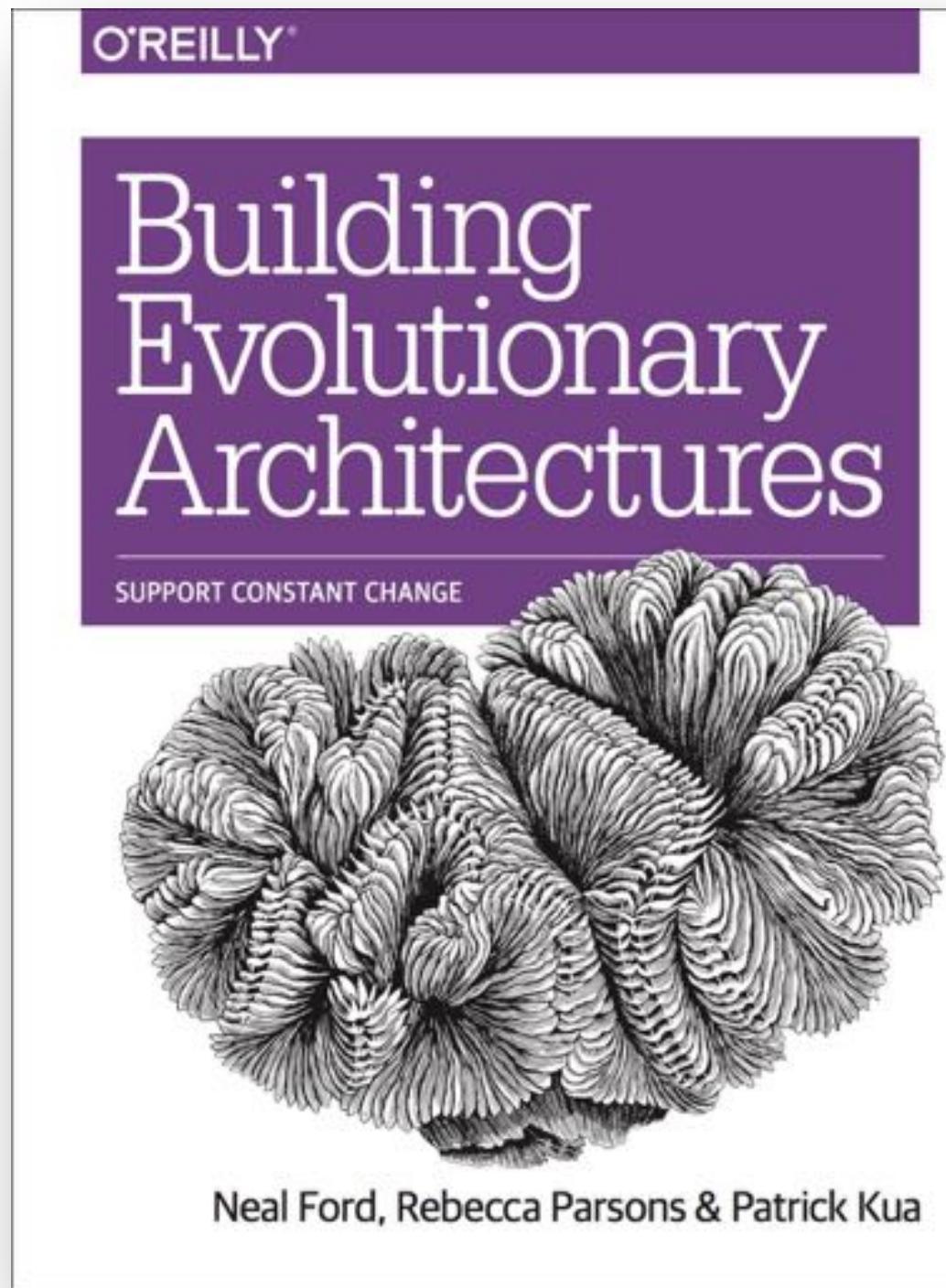
An architectural quantum is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

architectural quantum



An architectural quantum is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

architectural quantum

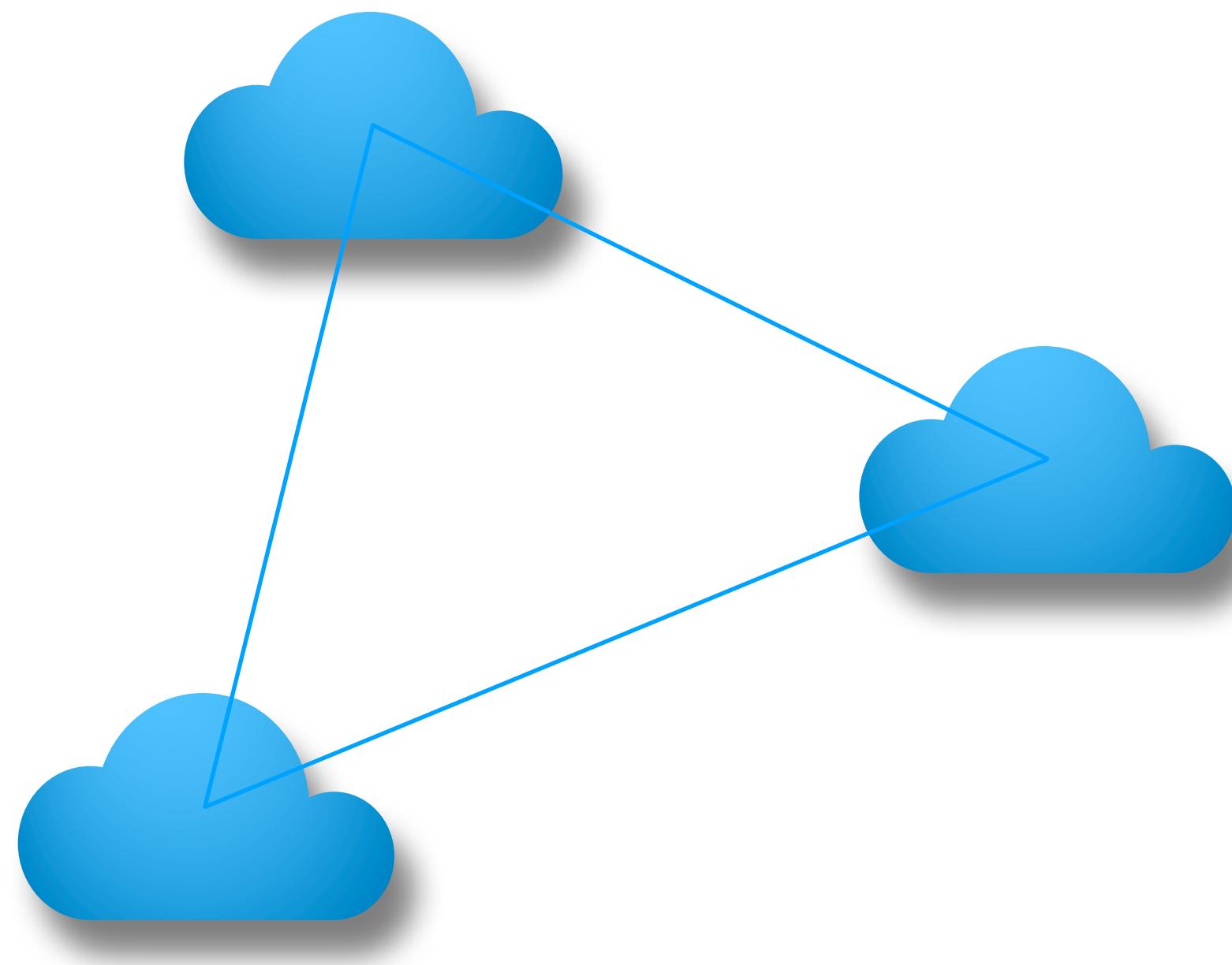


An architectural quantum is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

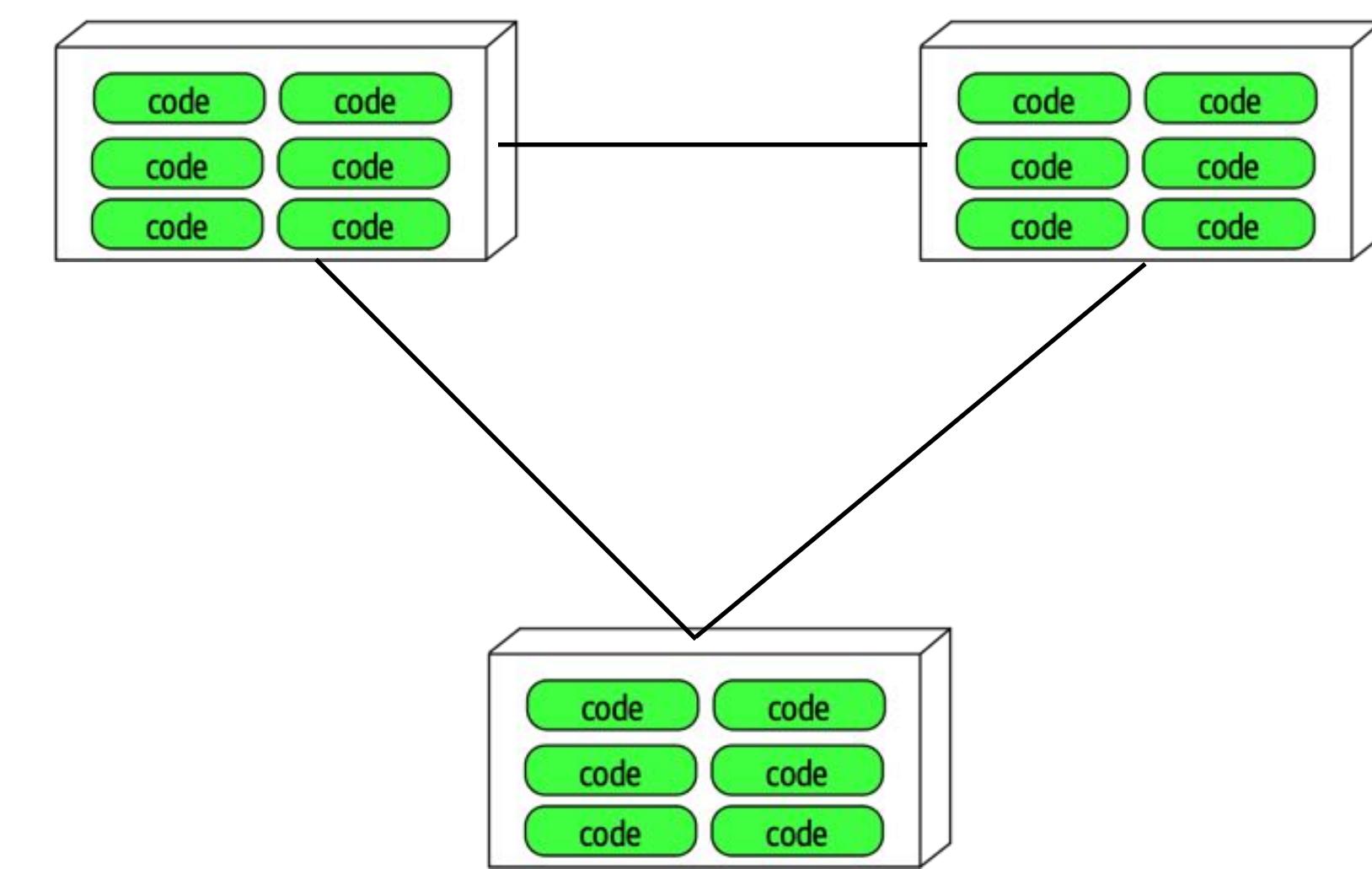
monolith | distributed ?

(semantic | syntactic) coupling ?

coupling

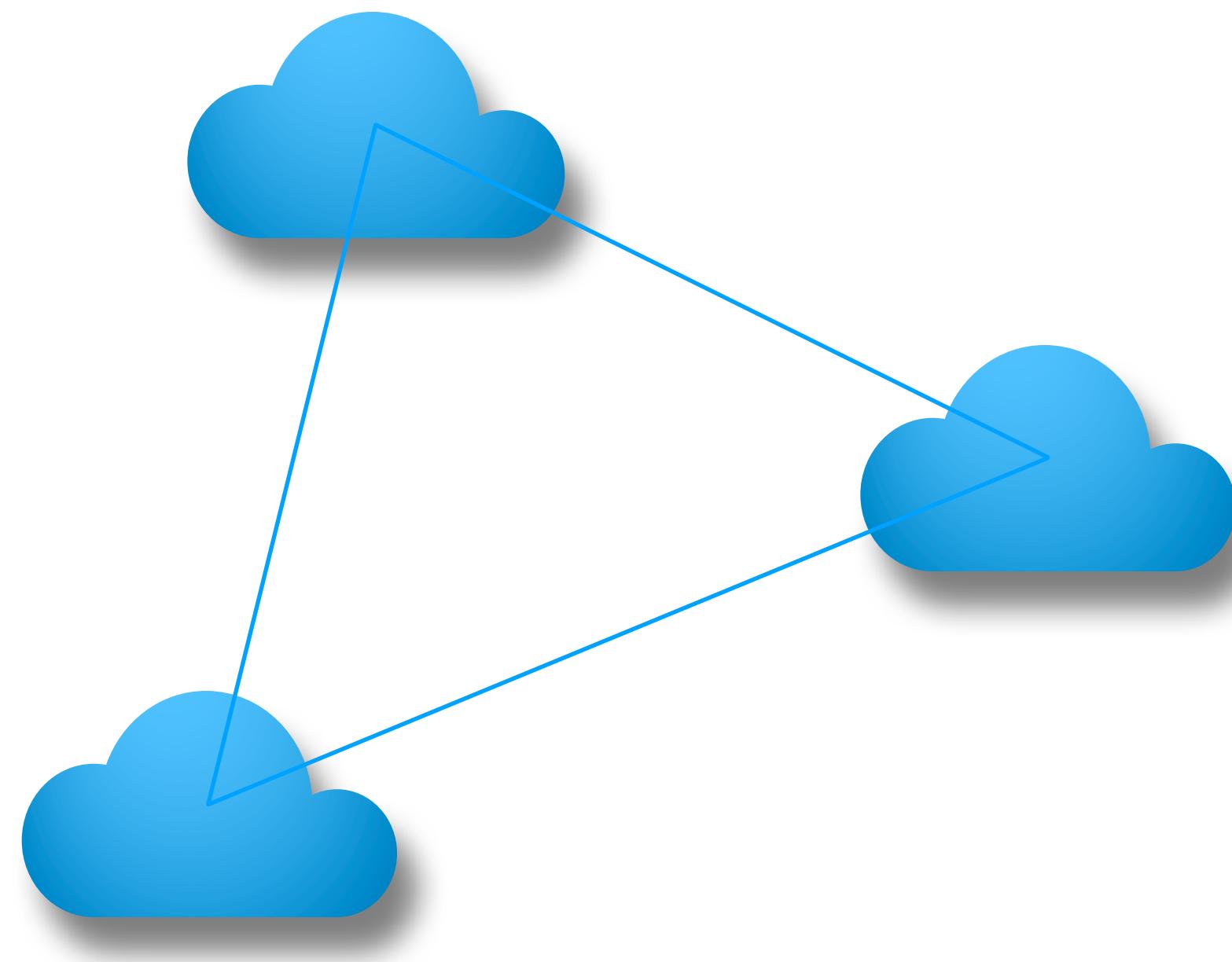


semantic

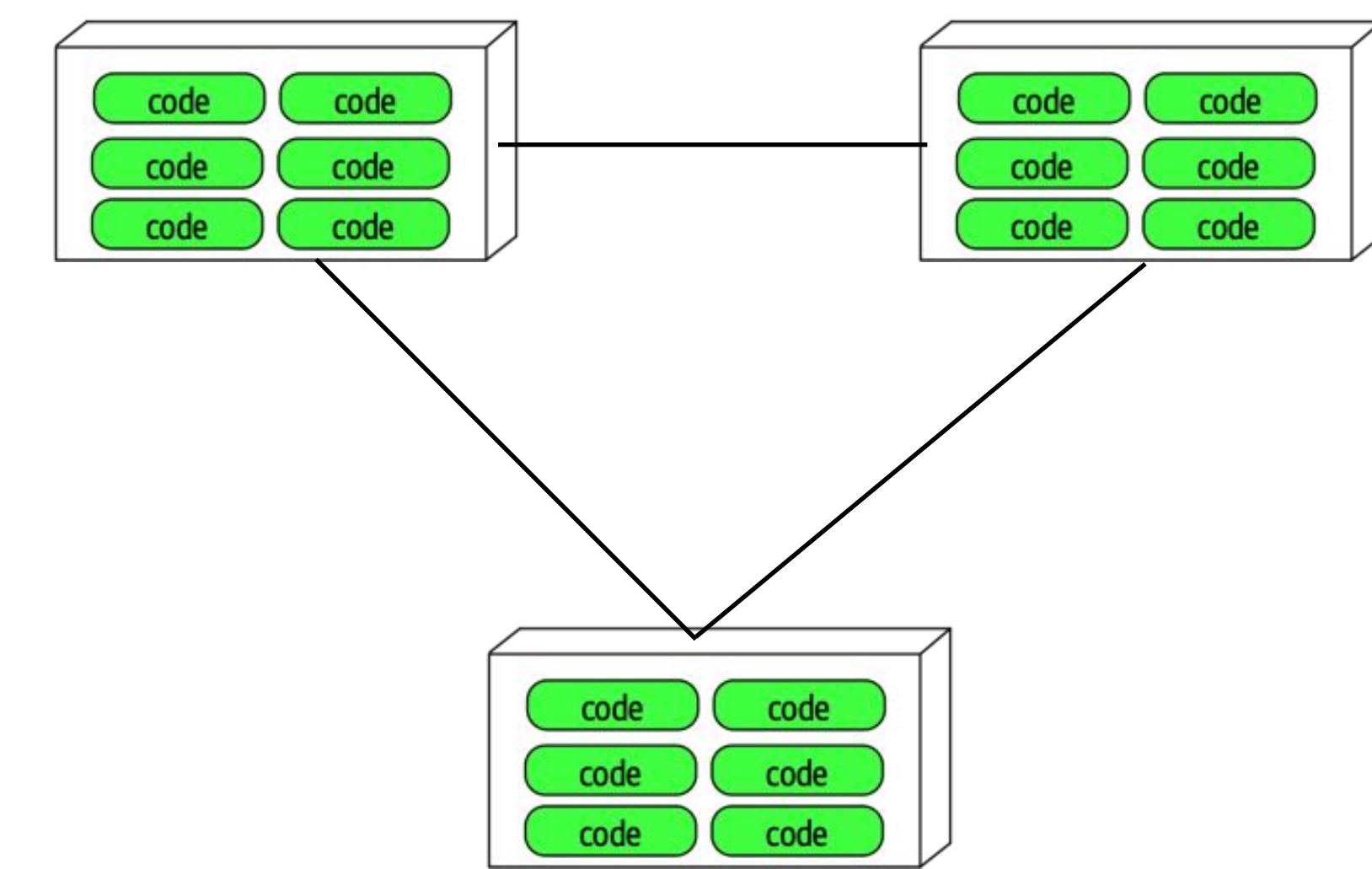


syntactic

coupling

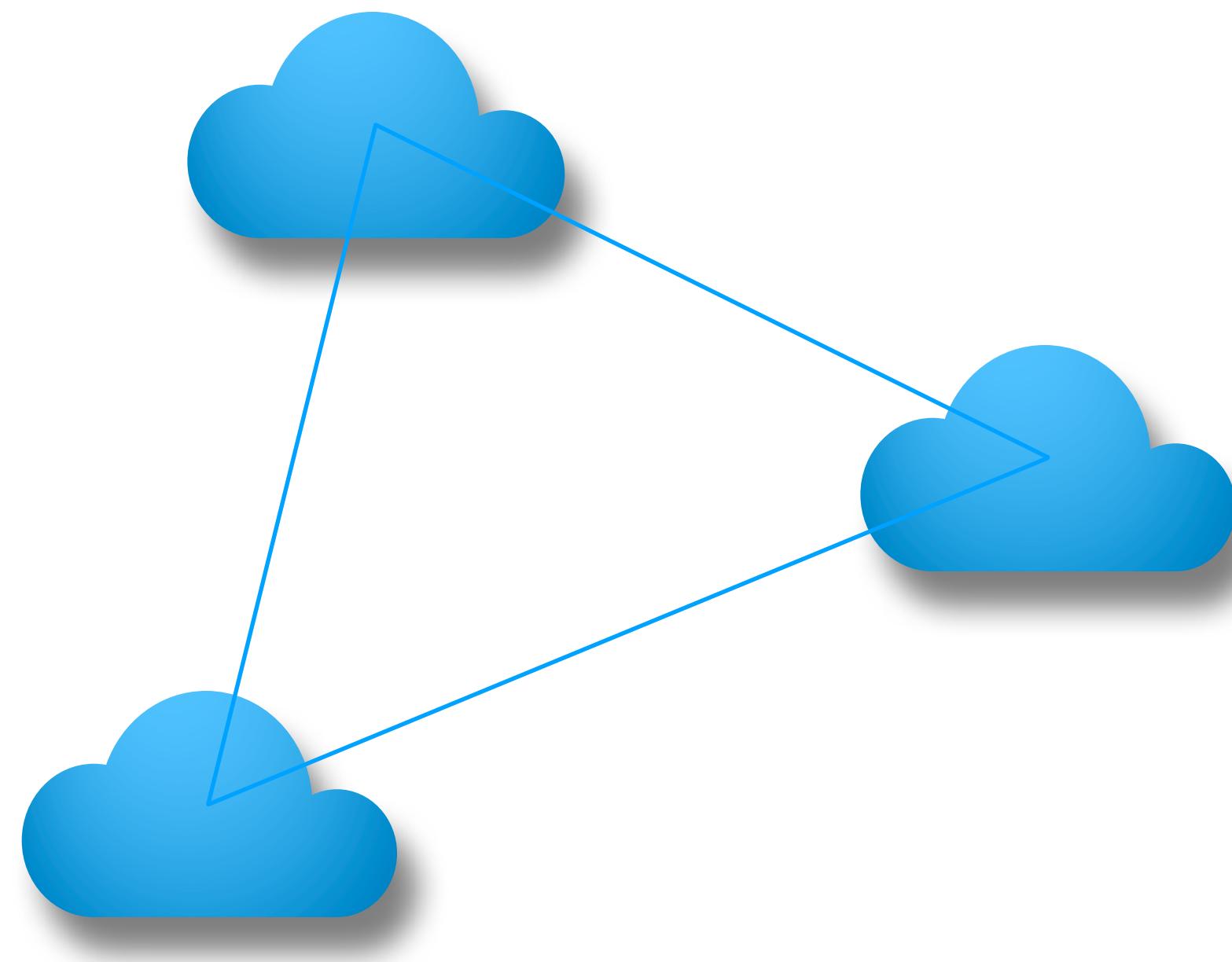


semantic

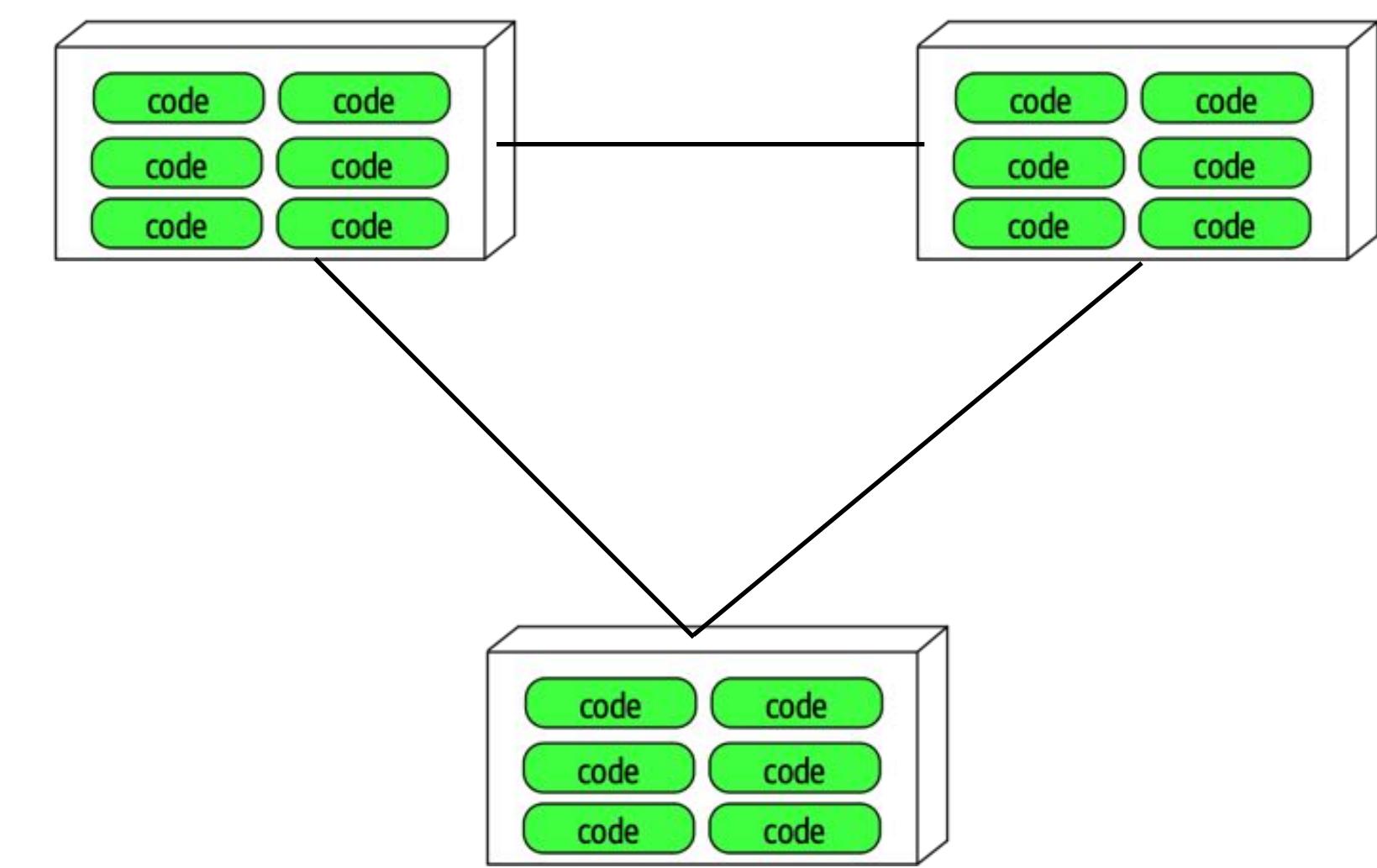


syntactic

coupling



semantic



syntactic

connascence

<https://connascence.io/name.html>



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

What is Connascence?

Connascence is a software quality metric & a taxonomy for different types of coupling. This site is a handy reference to the various types of connascence, with examples to help you improve your code.

Subject to Change

All code is subject to change. As the real world changes, so too must our code. Connascence gives us an insight into the long-term impact our code will have on flexibility, as we write it. Maintaining a flexible codebase is essential for maintaining long-term development velocity.

A Flexible Metric

Connascence is a metric, and like all metrics is an imperfect measure. However, connascence takes a more holistic approach, where each instance of connascence in a codebase must be considered on three separate axes:

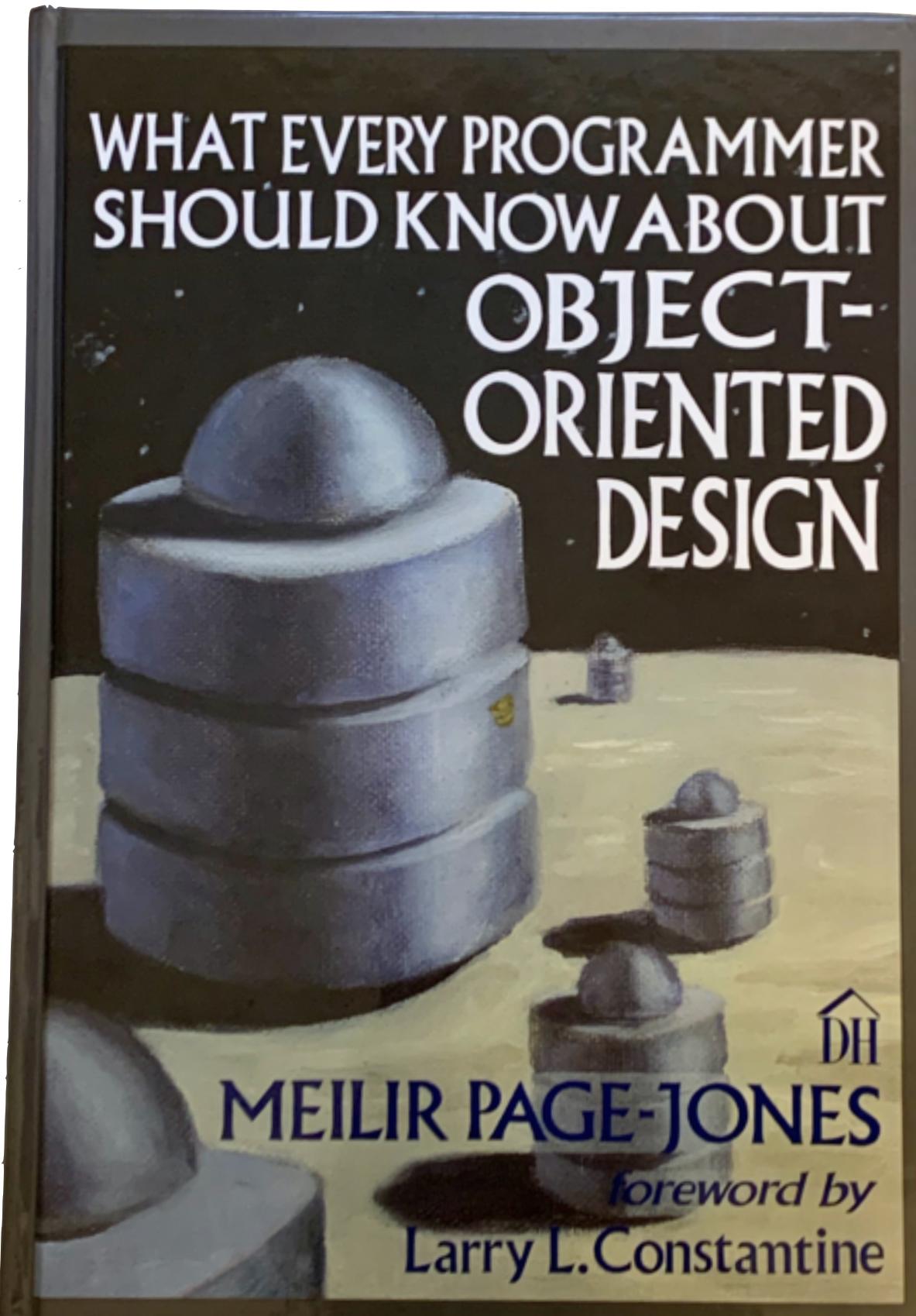
1. Strength. Stronger connascences are harder to discover, or harder to refactor.
2. Degree. An entity that is connascent with thousands of other entities is likely to be a larger issue than one that is connascent with only a few.
3. Locality. Connascent elements that are close together in a codebase are better than ones that are far apart.

The three properties of Strength, Degree, and Locality give the programmer all the tools they need in order to make informed decisions about when they will permit certain types of coupling, and when the code ought to be refactored.

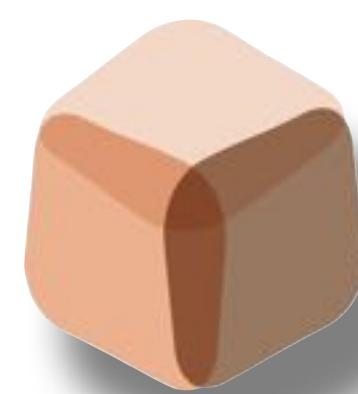
A Vocabulary for Coupling

Arguably one of the most important benefits of connascence is that it gives developers a vocabulary to talk about different types of coupling. Connascence codifies what many experienced engineers have learned by trial and error: Having a common set of nouns to refer to different types of coupling allows us to share that experience more easily.

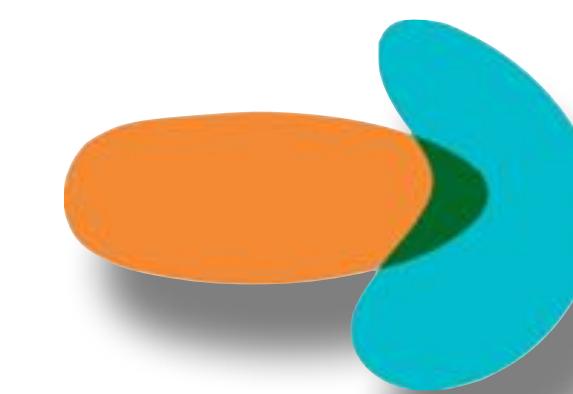
connascence



Two components are connascent if a change in one would require the other to be modified in order to maintain the overall correctness of the system.



static



dynamic

connascence properties

Strength



name

type

meaning

algorithm

position

execution order

timing

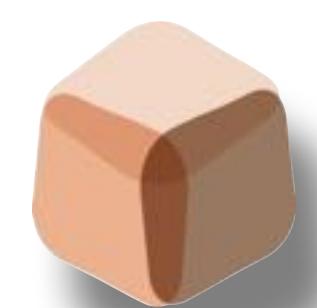
value

identity



static

dynamic

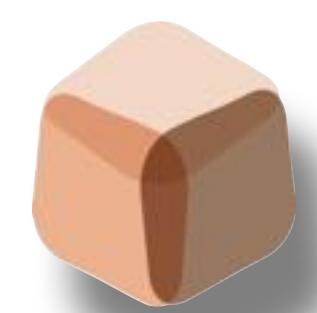


static

connascence of meaning

multiple components must agree on the meaning of particular values

```
def valid_credit_card_number?(cc_number)
  # check for "test" credit card numbers
  if cc_number == "9999-9999-9999-9999"
    # test verification
  else
    # some more code
  end
end
```



static

connascence of meaning

multiple components must agree on the meaning of particular values

```
def get_user_role(username):
    user = database.get_user_object_for_username(username)
    if user.is_admin:
        return 2
    elif user.is_manager:
        return 1
    else:
        return 0
```

constants!

...move from connascence of *meaning* to
connascence of *name*

```
if get_user_role(username) != 2:
    raise PermissionDenied("You must be an administrator")
```

connascence properties

Strength



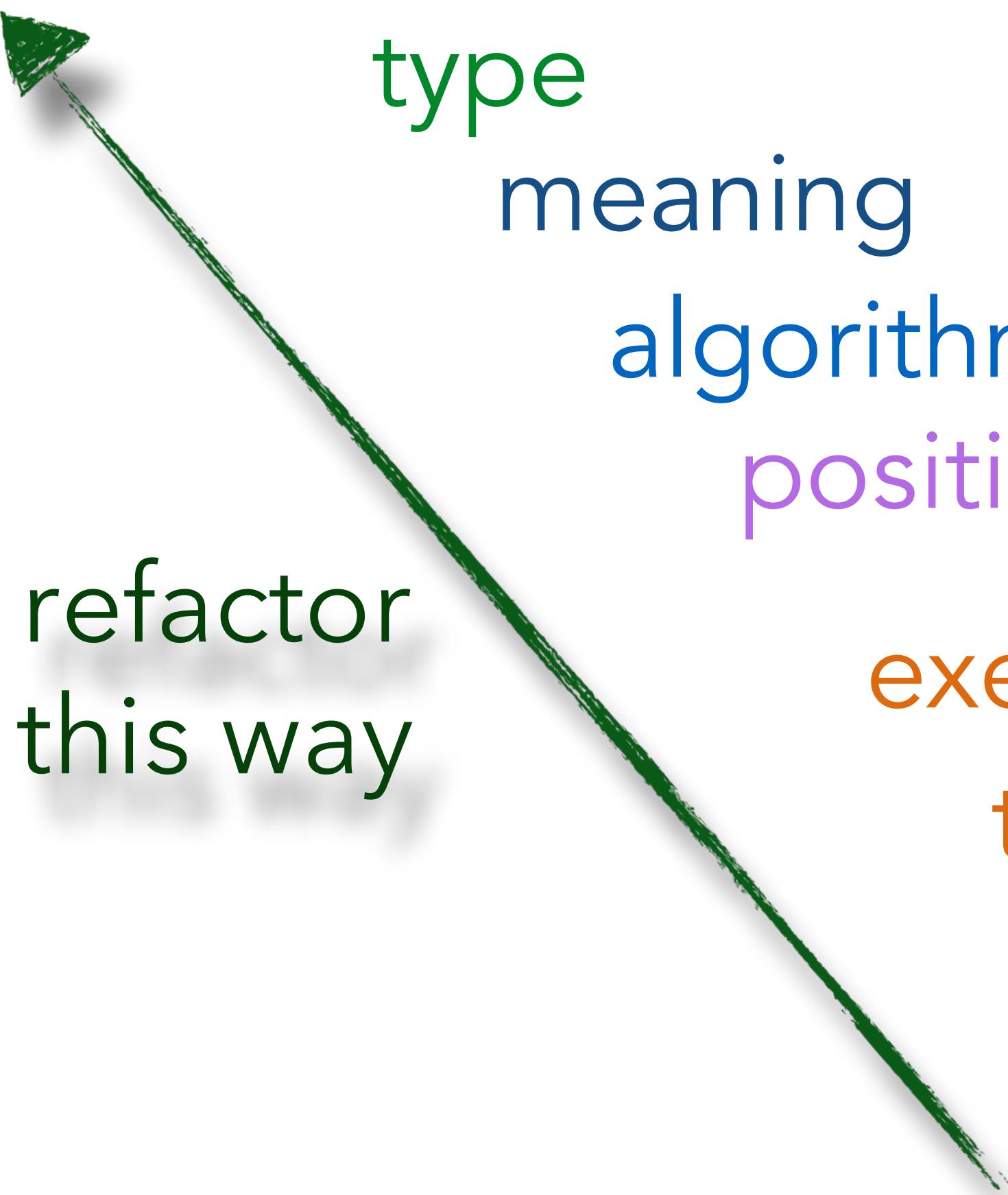
refactor
this way

name
type
meaning
algorithm
position
execution order
timing
value
identity



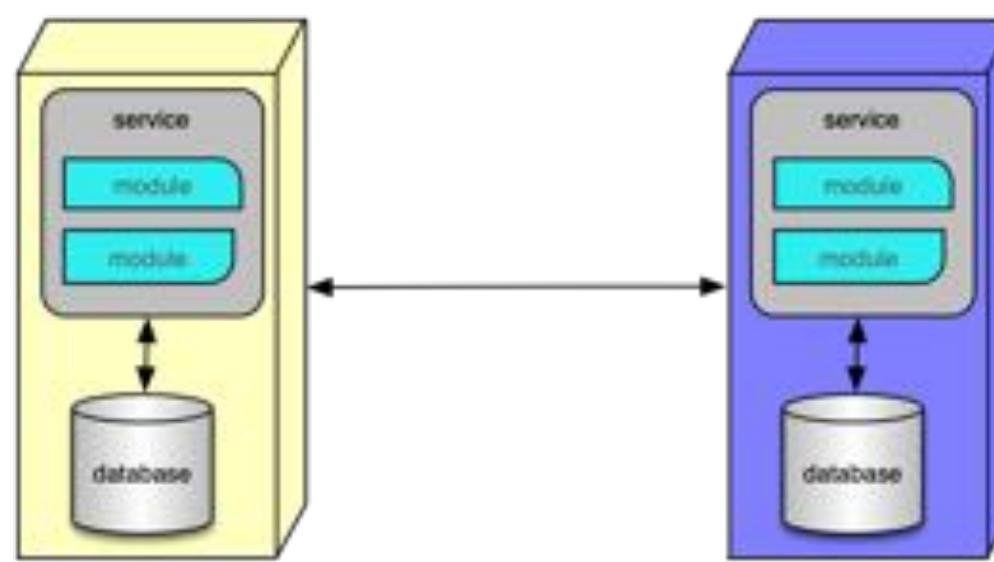
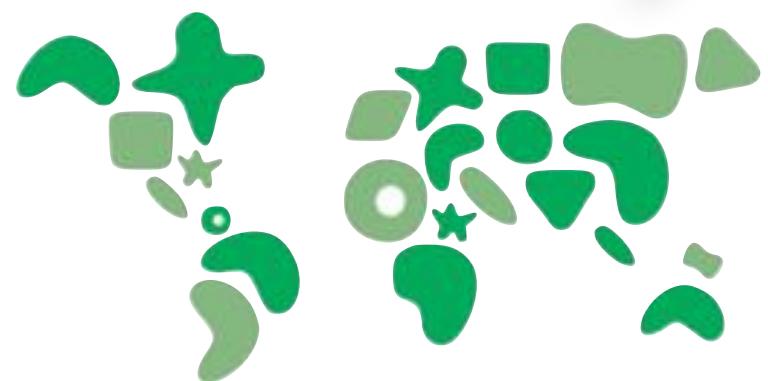
static

dynamic



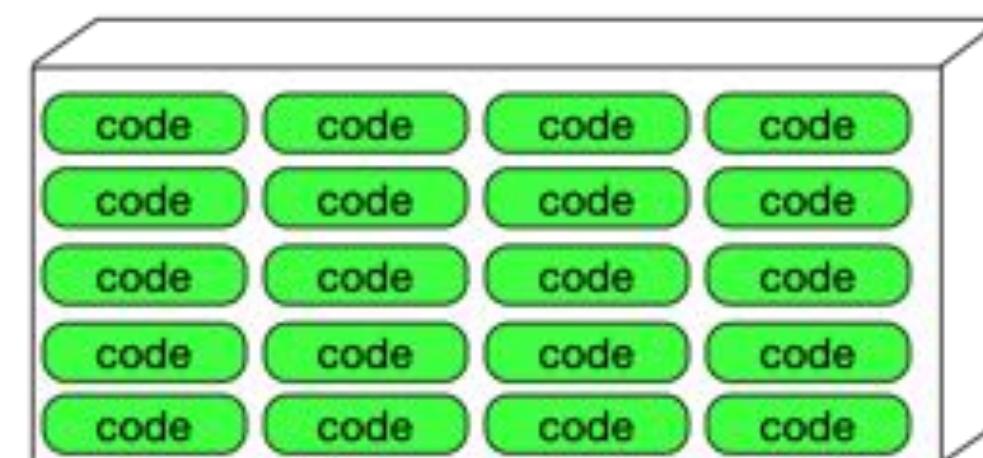
connascence properties

Locality



static

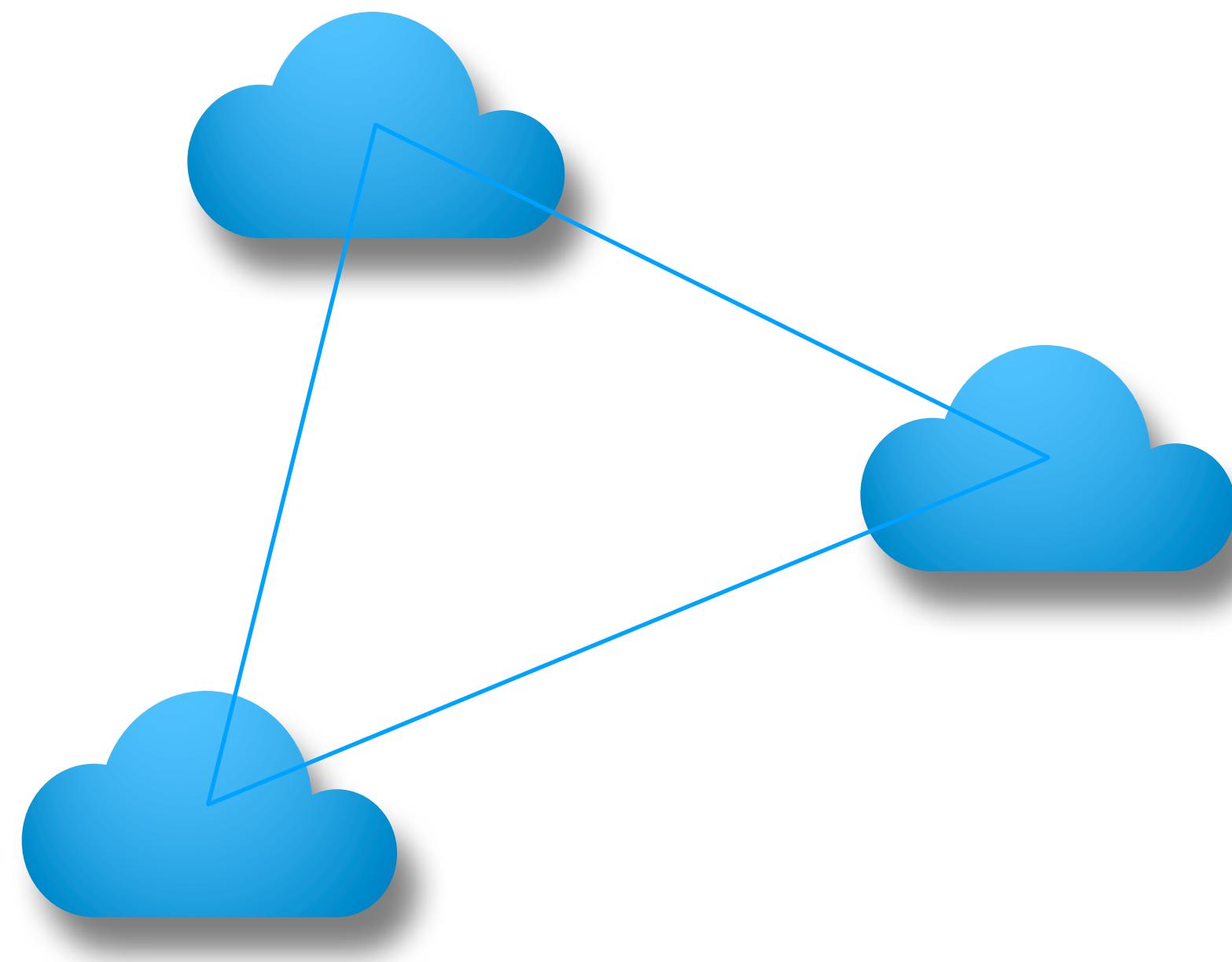
Proximal code (in the same module, class, or function) should have more & higher forms of connascence than less proximal code (in separate modules, or even codebases).



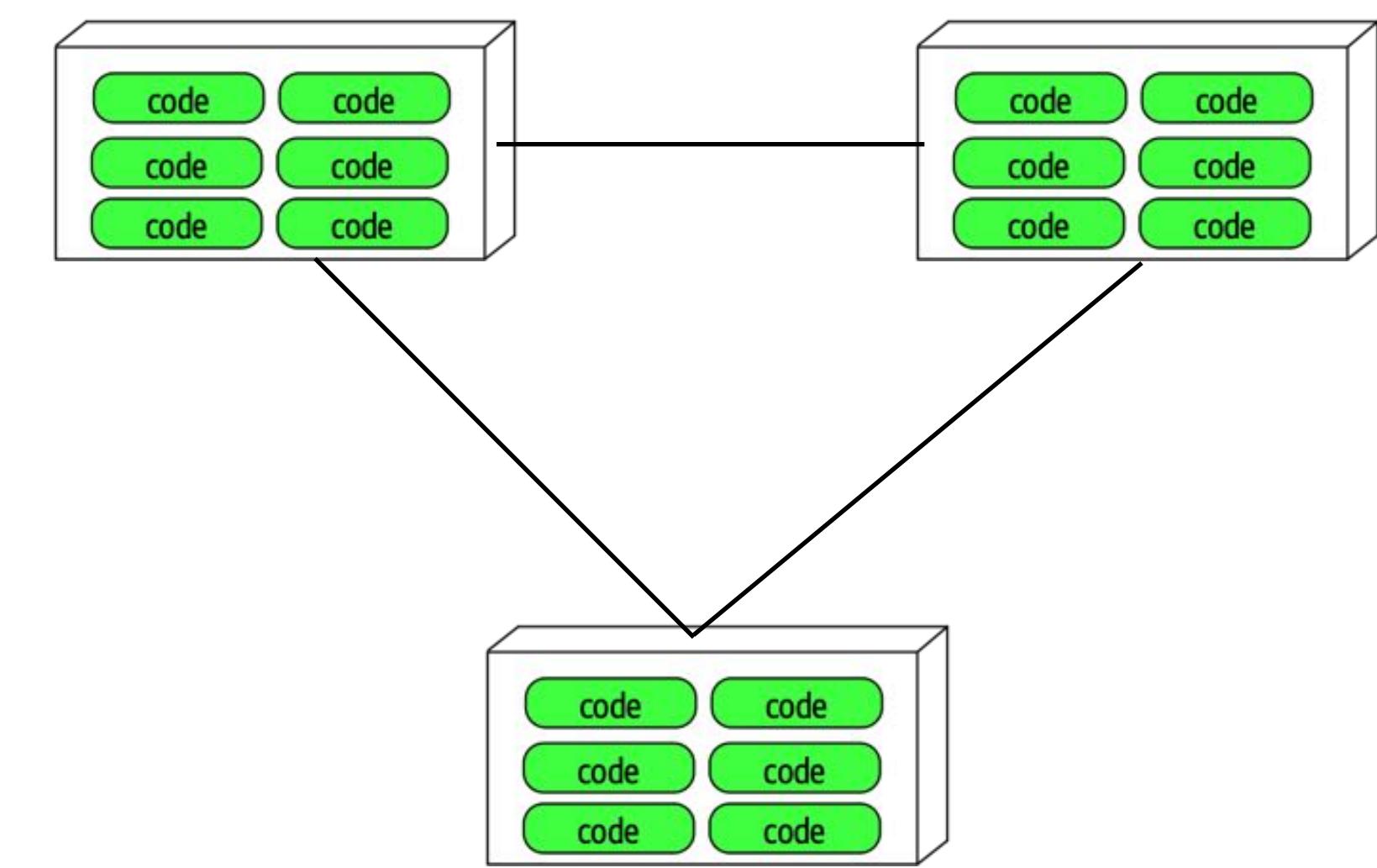
dynamic



coupling



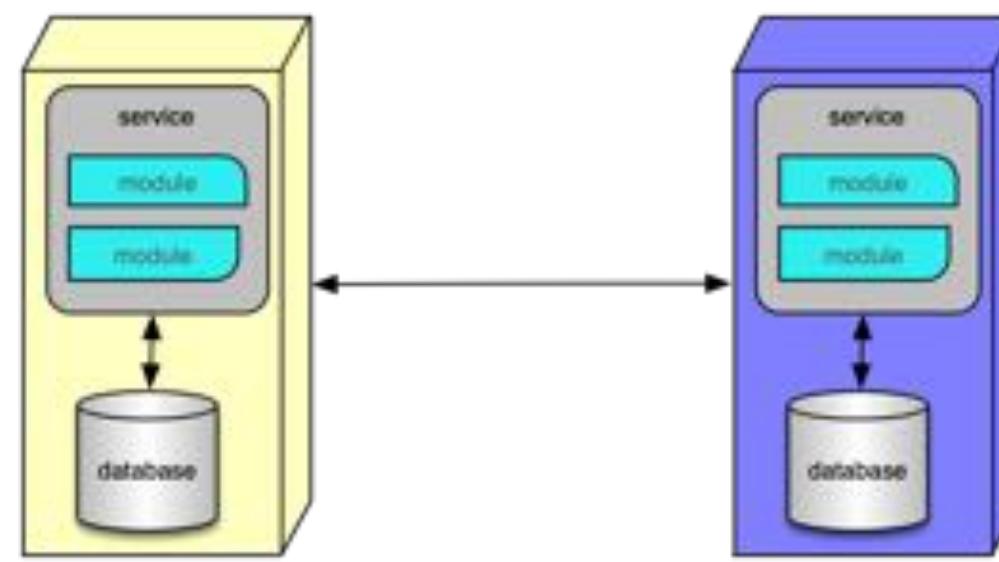
semantic



syntactic

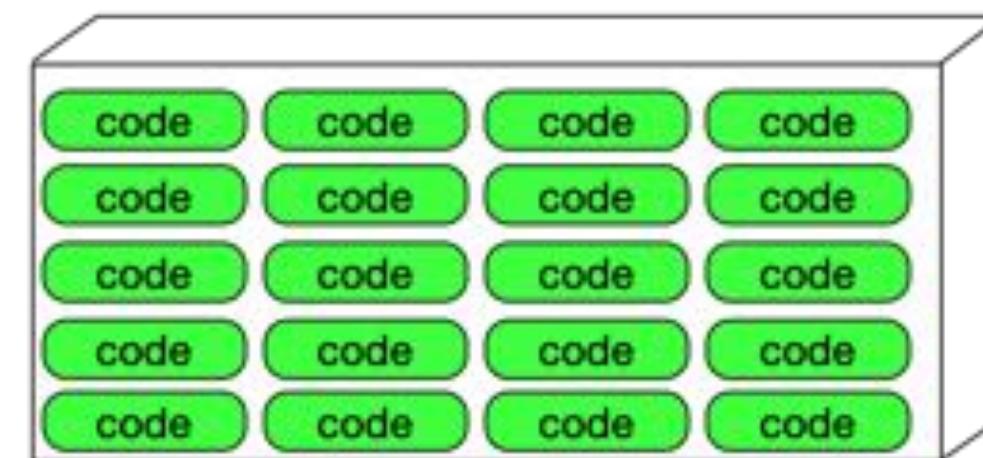
connascence properties

Locality



static

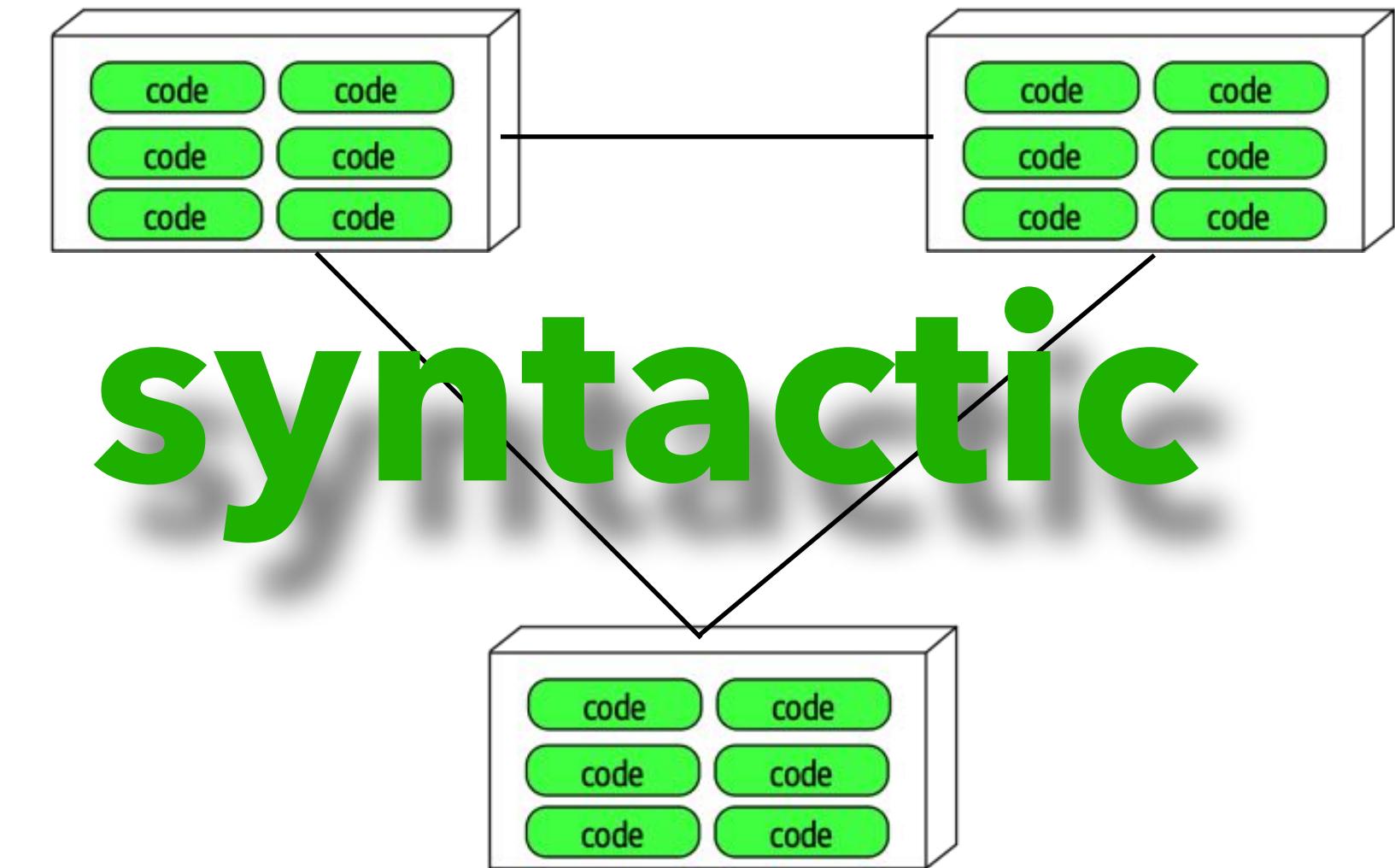
Proximal code (in the same module, class, or function) should have more & higher forms of connascence than less proximal code (in separate modules, or even codebases).



dynamic

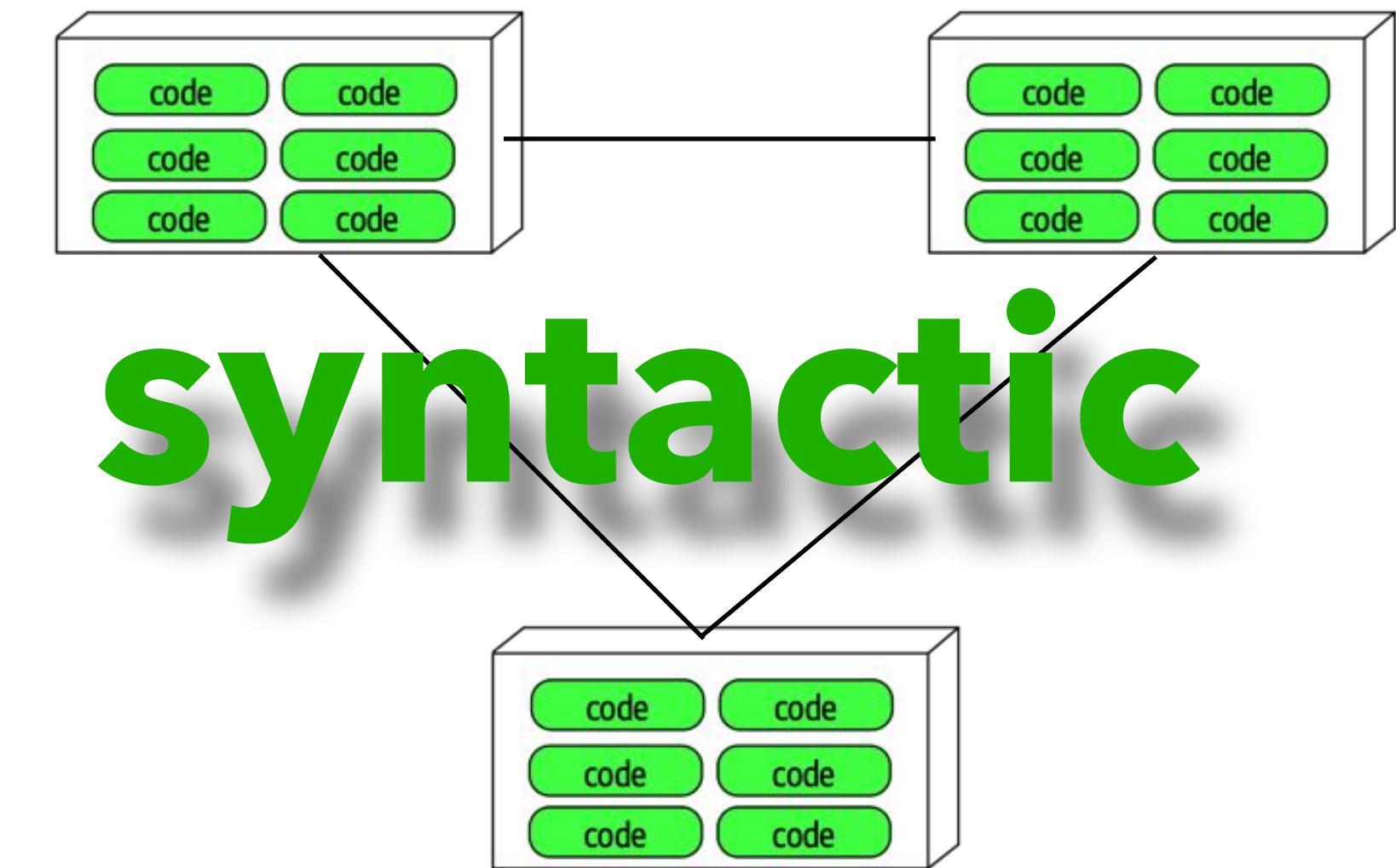


coupling



For highly semantically coupled problem domains,
limit the scope of syntactic coupling.

coupling



For highly semantically coupled problem domains,
limit the scope of syntactic coupling.

duce the amount of semantic coupling via the application

connascence properties

Strength



name

type

meaning

algorithm

position

execution order

timing

value

identity

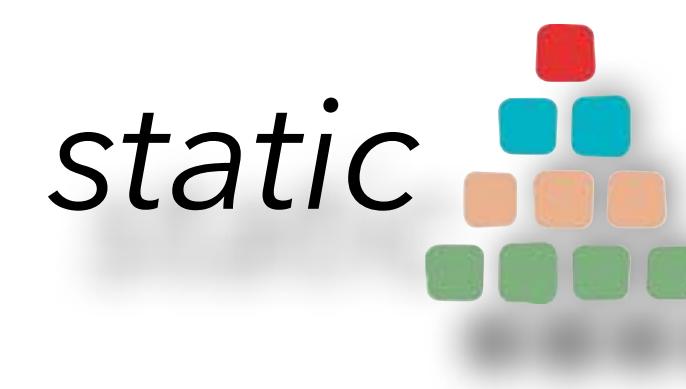


static

dynamic

quantum connascence

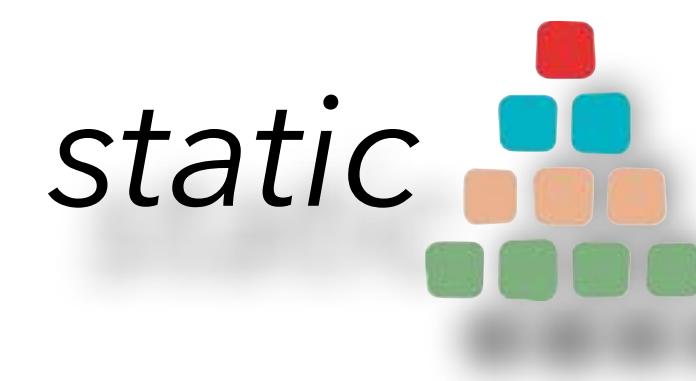
quantum connascence



dynamic



quantum connascence



static

dynamic



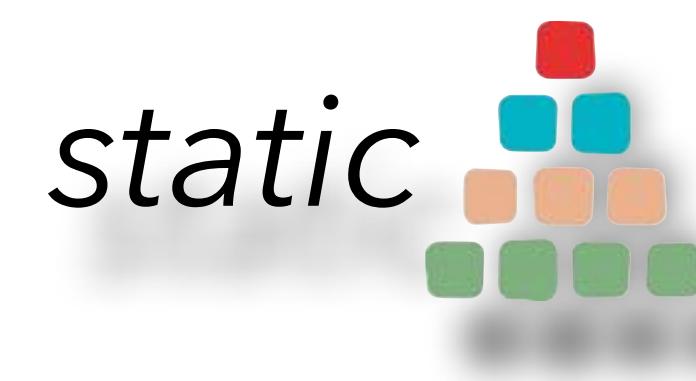
contracts



tight

loose

quantum connascence



static

contracts



tight

loose

synchronous



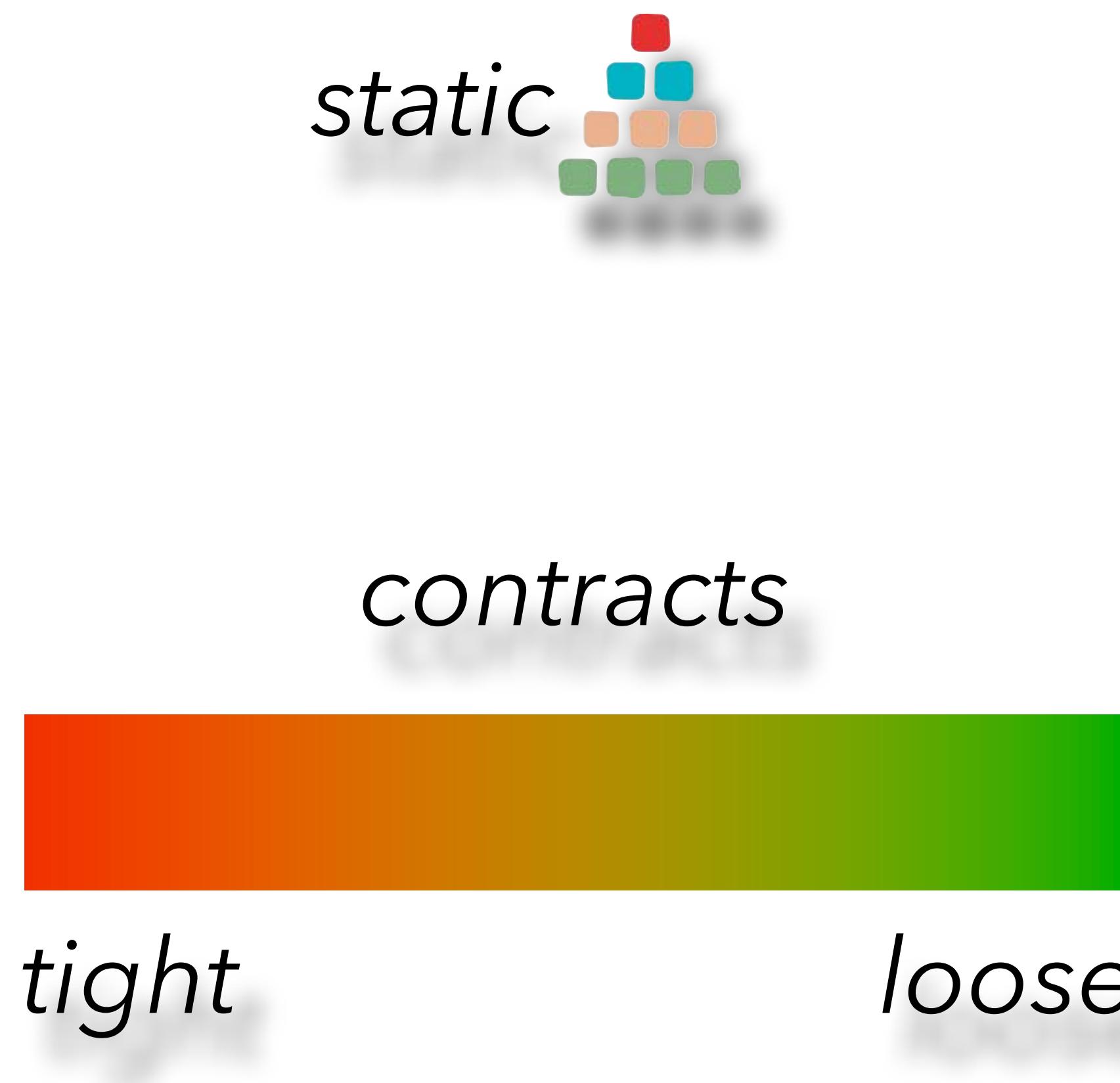
dynamic



asynchronous



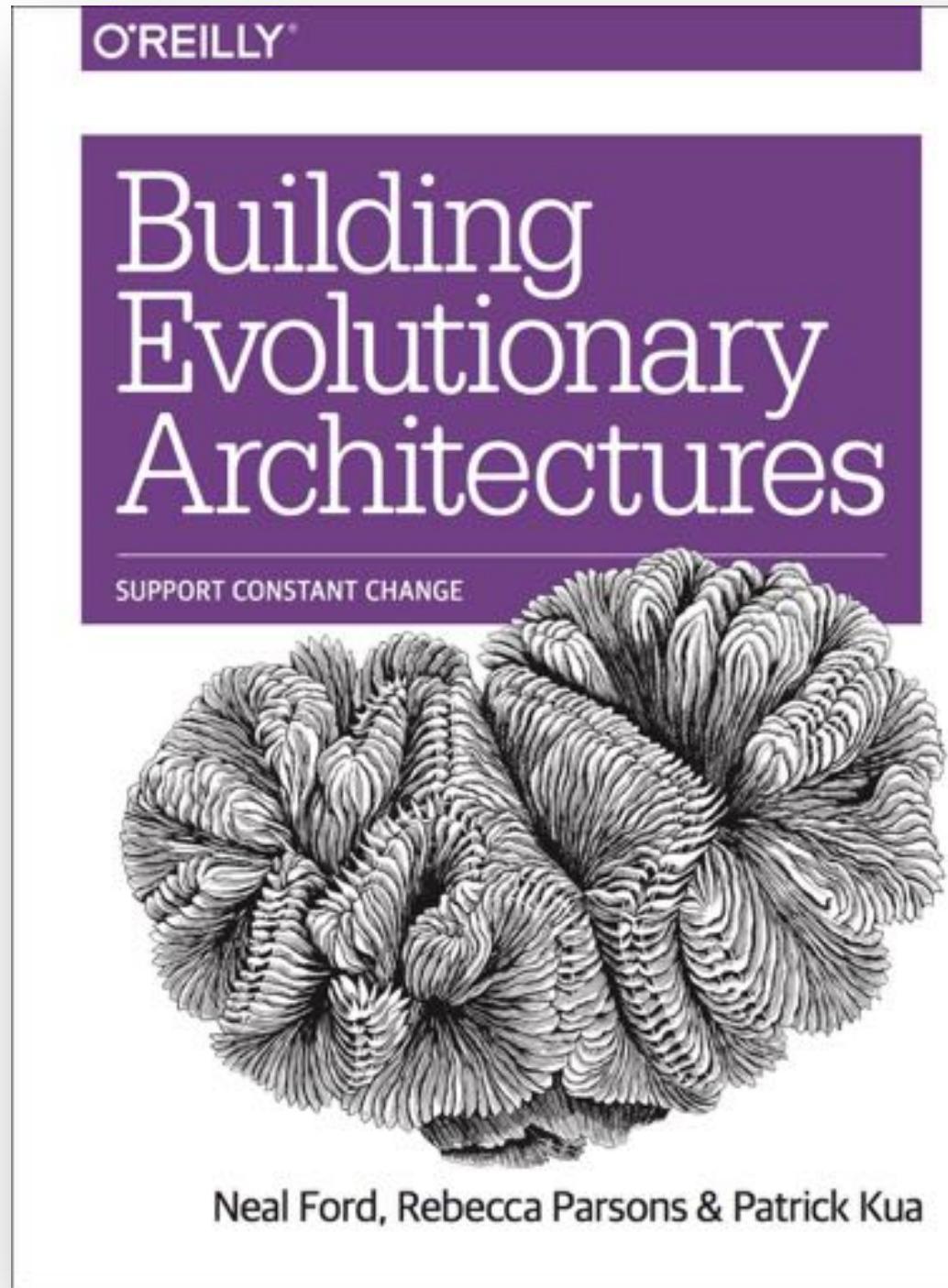
quantum connascence



impact on operational
(and other) architecture characteristics

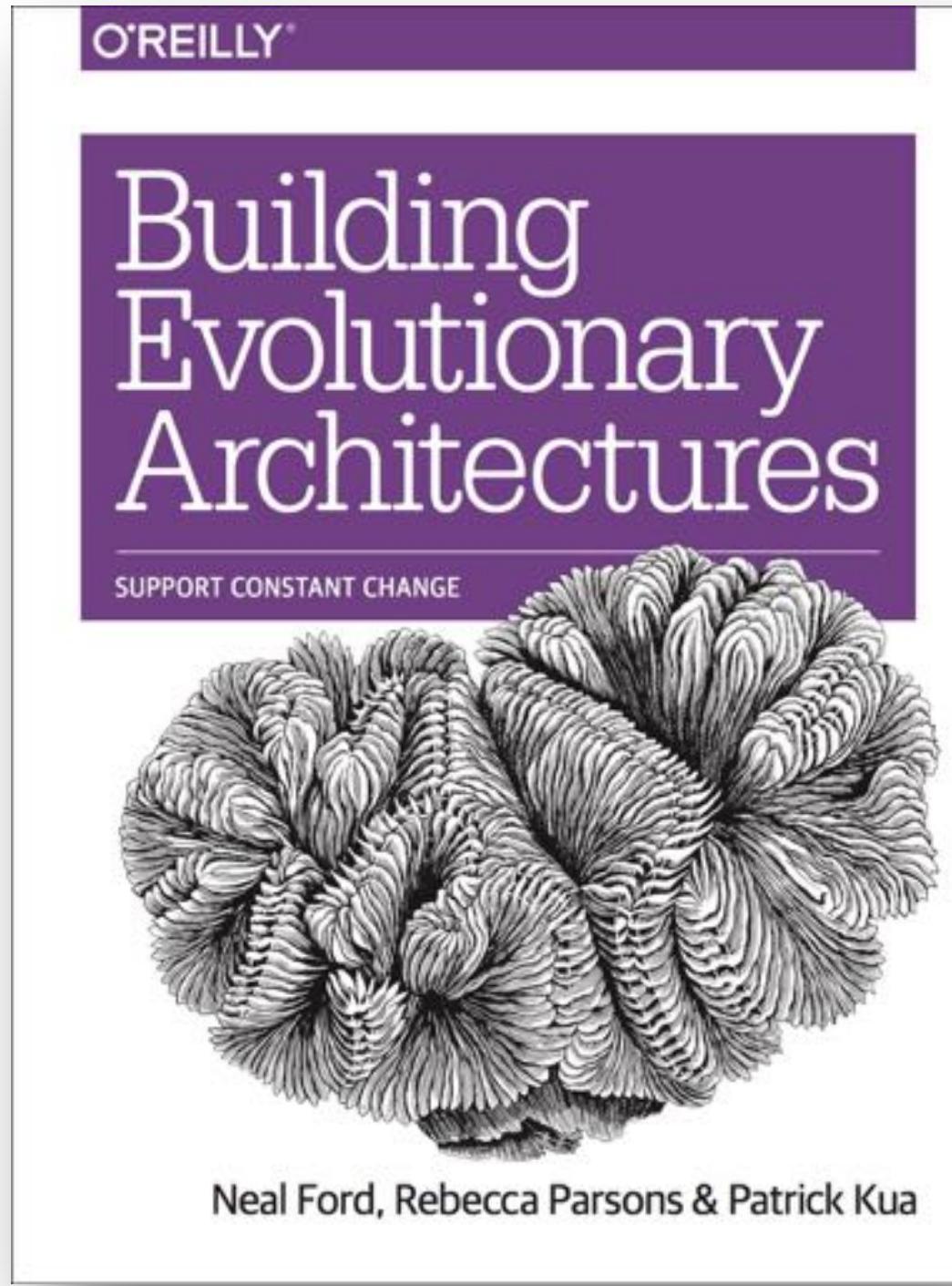
useful for hybrid architecture design,
architecture migration, integration, etc.

architectural quantum



An *architectural quantum* is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

architectural quantum



An architectural quantum is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

*architectural characteristics
live at the quantum level*

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

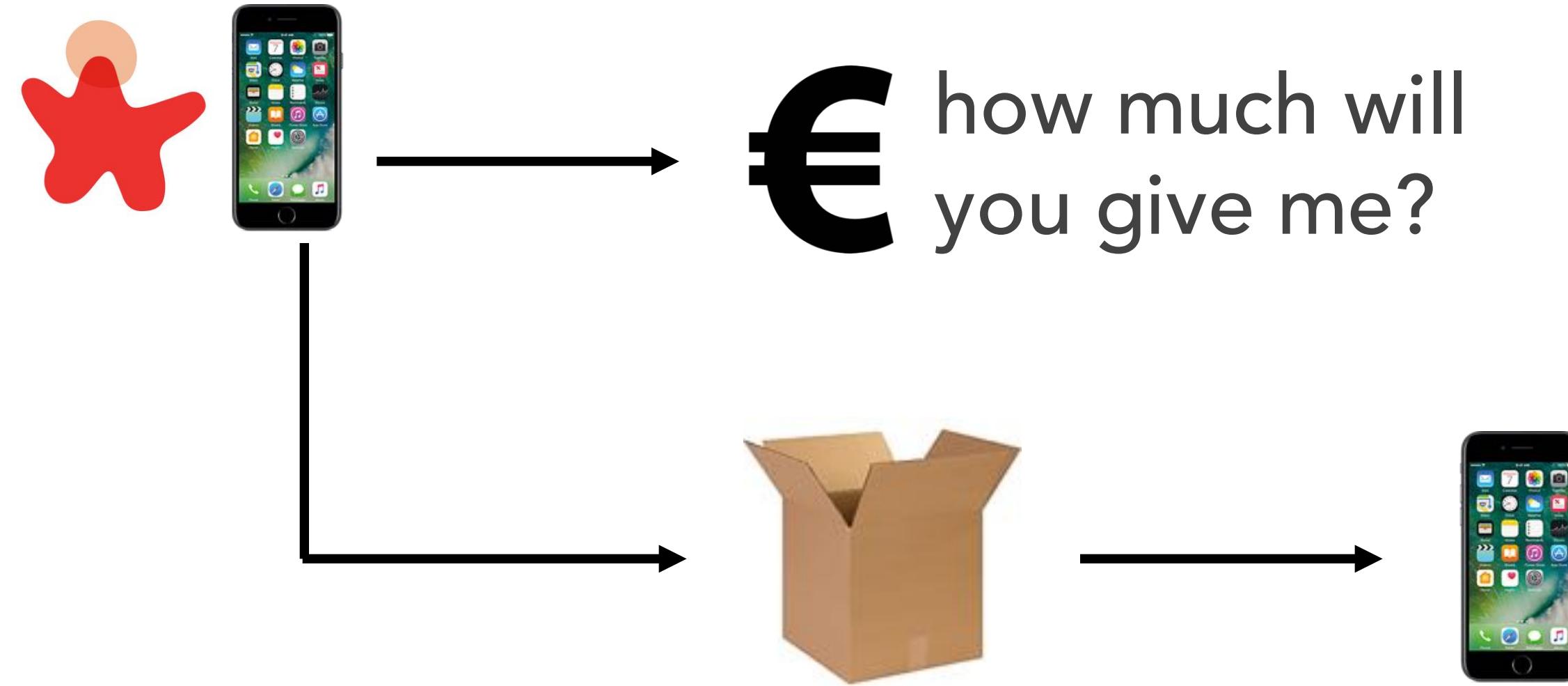
architectural quantum

electronics recycling



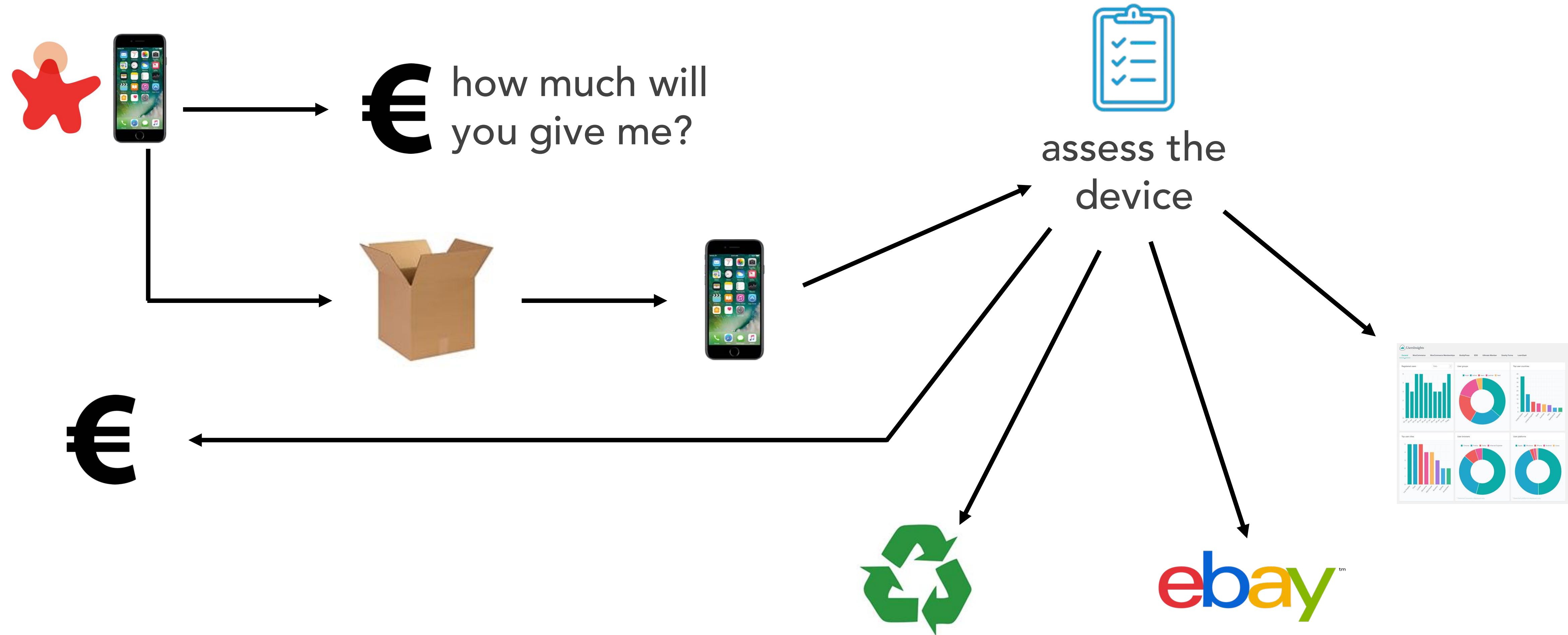
architectural quantum

electronics recycling



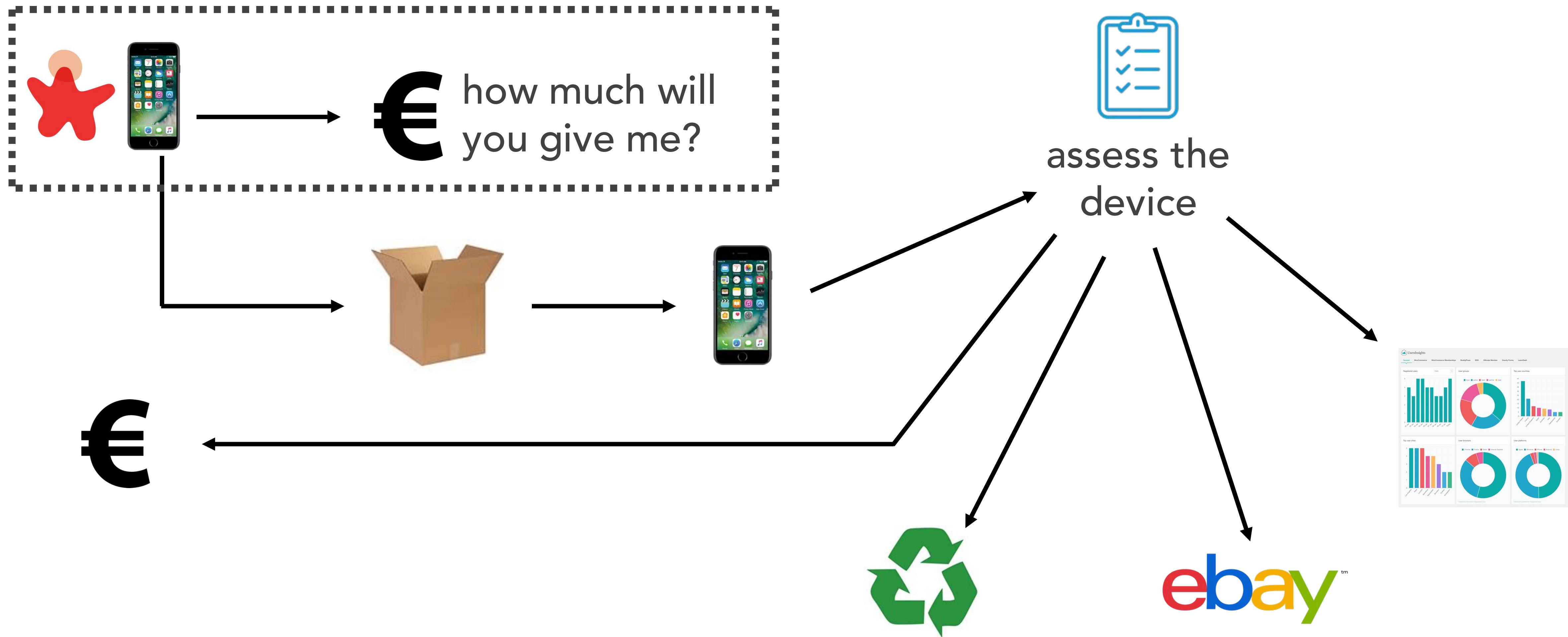
architectural quantum

electronics recycling



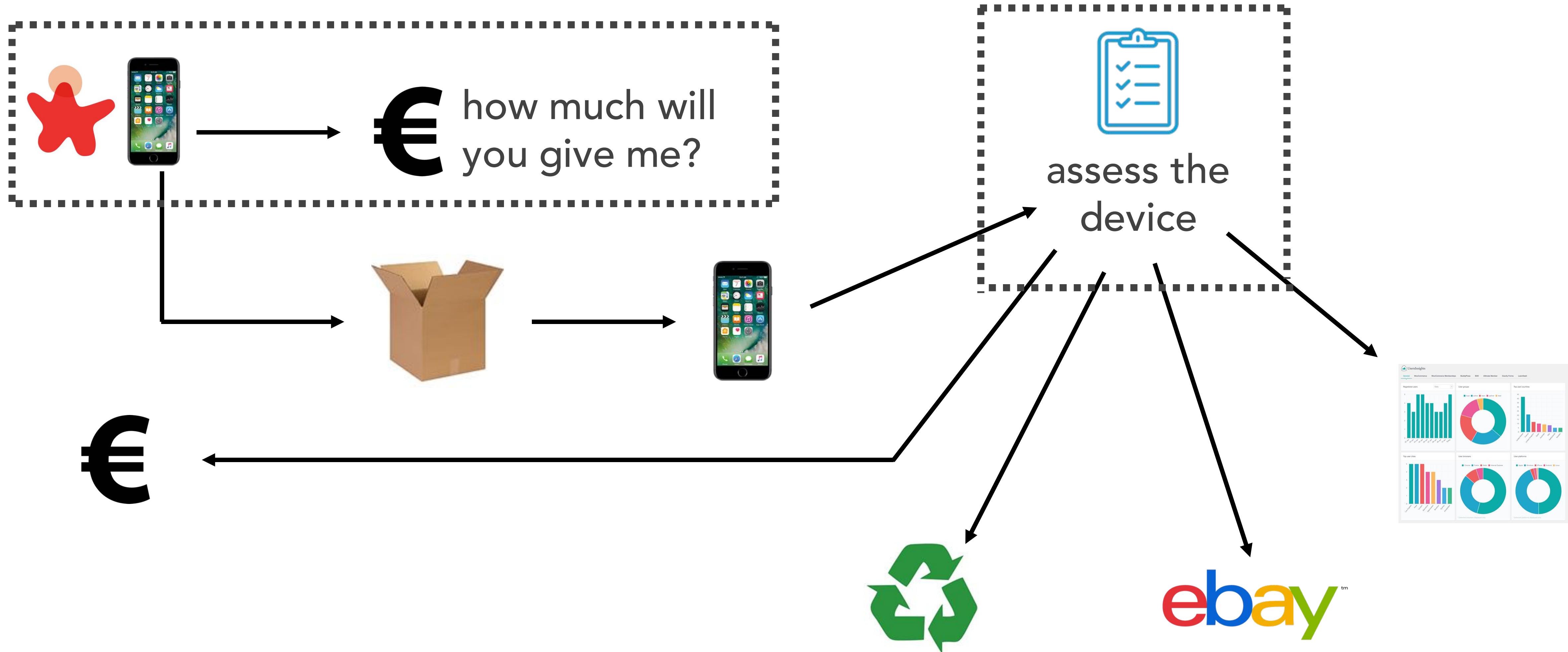
architectural quantum

electronics recycling



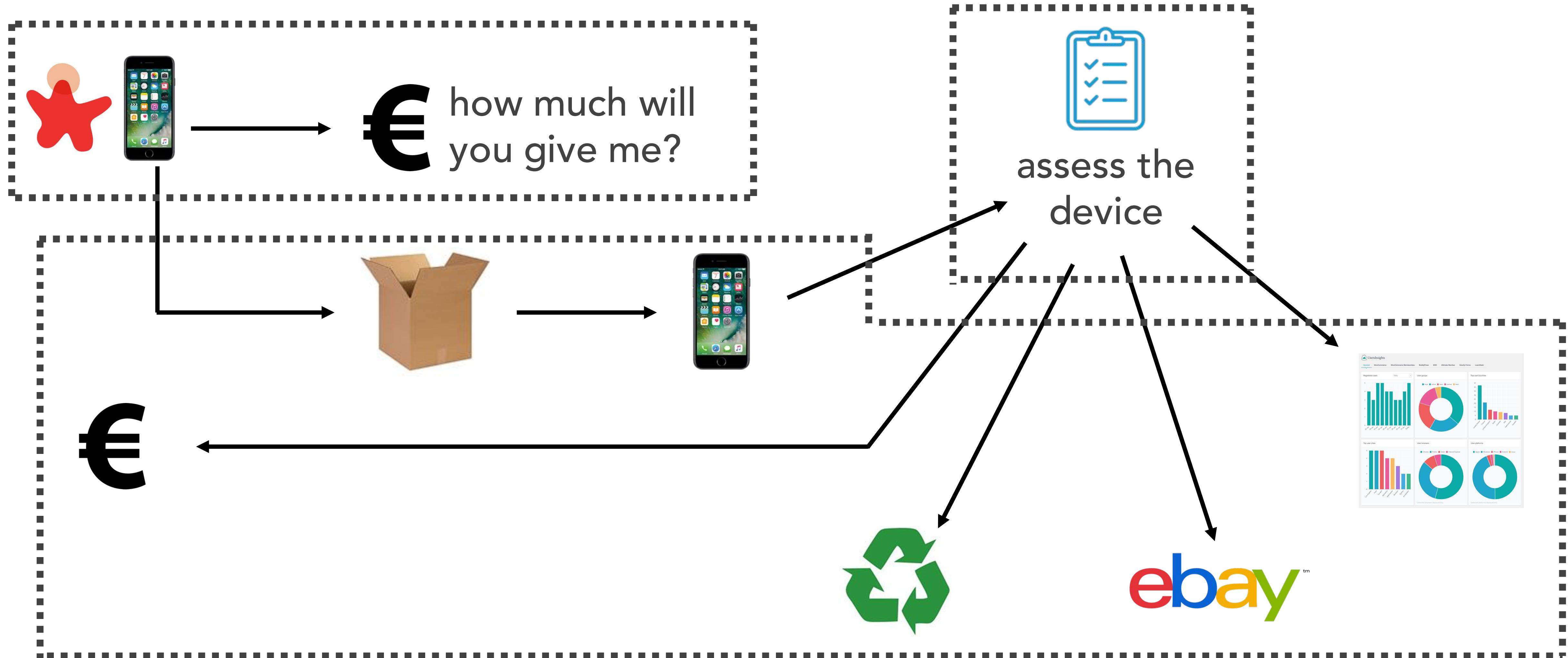
architectural quantum

electronics recycling



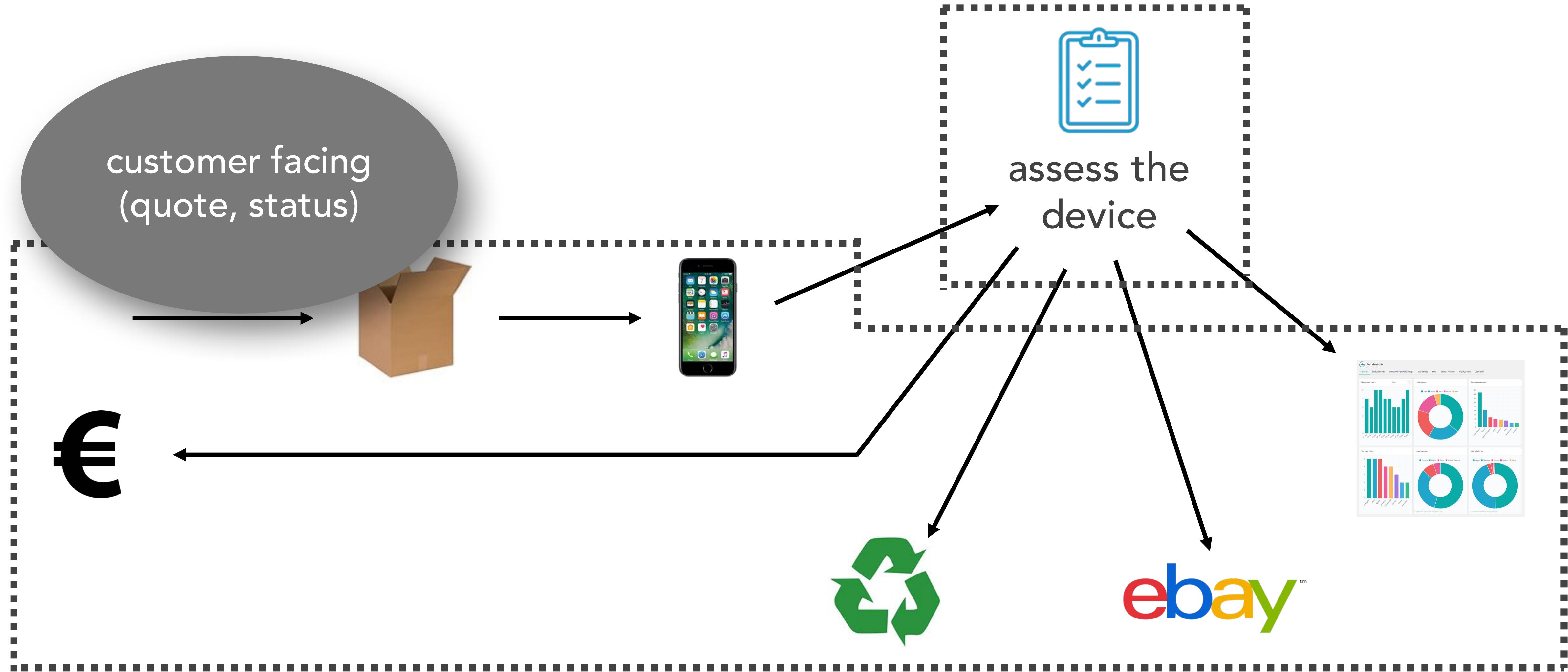
architectural quantum

electronics recycling



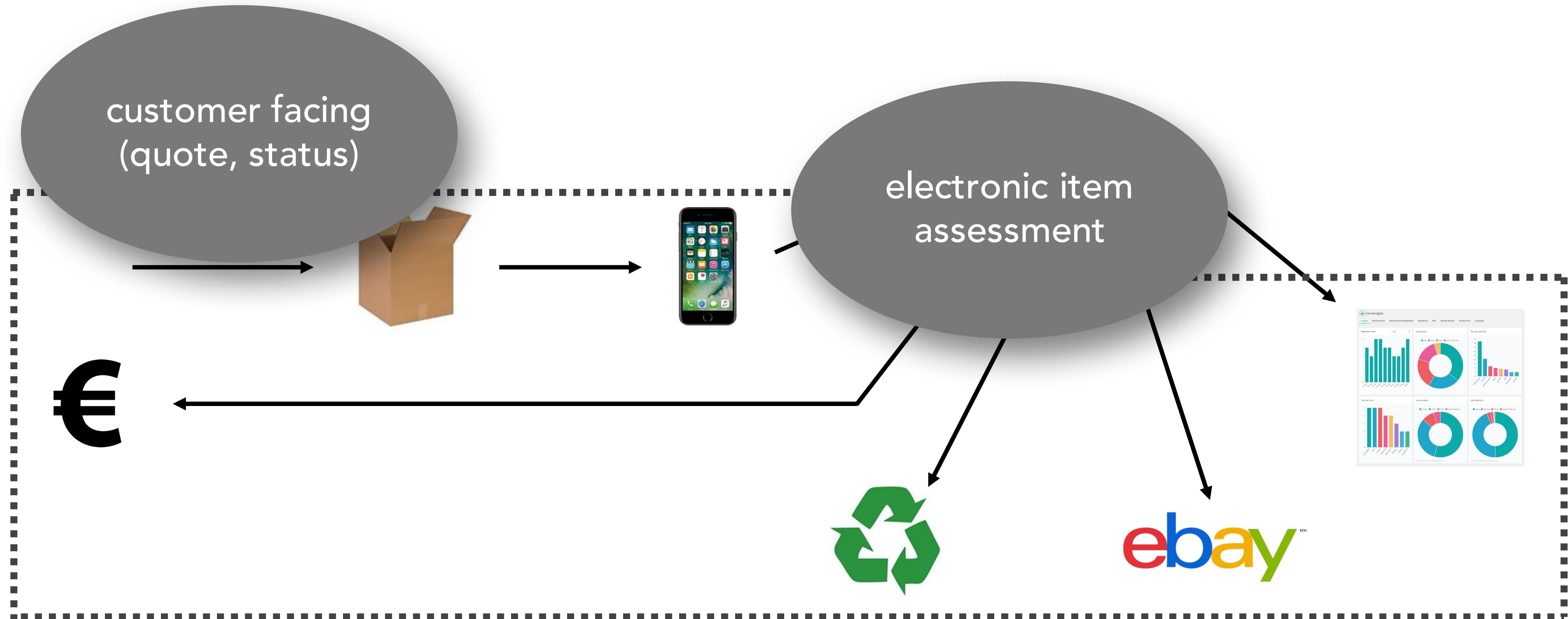
architectural quantum

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

electronic item
assessment

maintainability
deployability
testability
agility

recycling, reporting,
and accounting

architectural quantum

electronics recycling

customer facing
(quote, status)

scalability
availability
agility

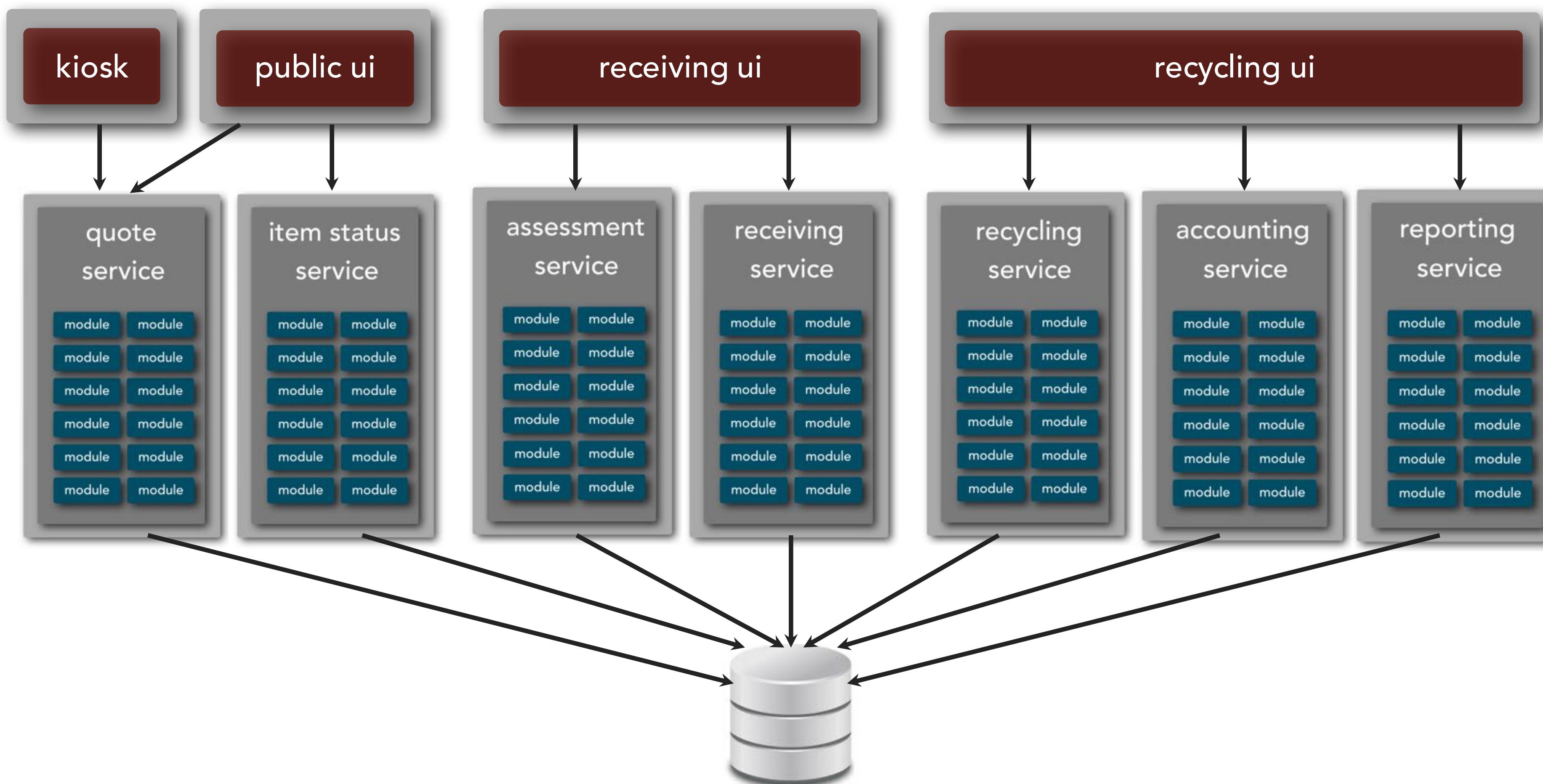
electronic item
assessment

maintainability
deployability
testability
agility

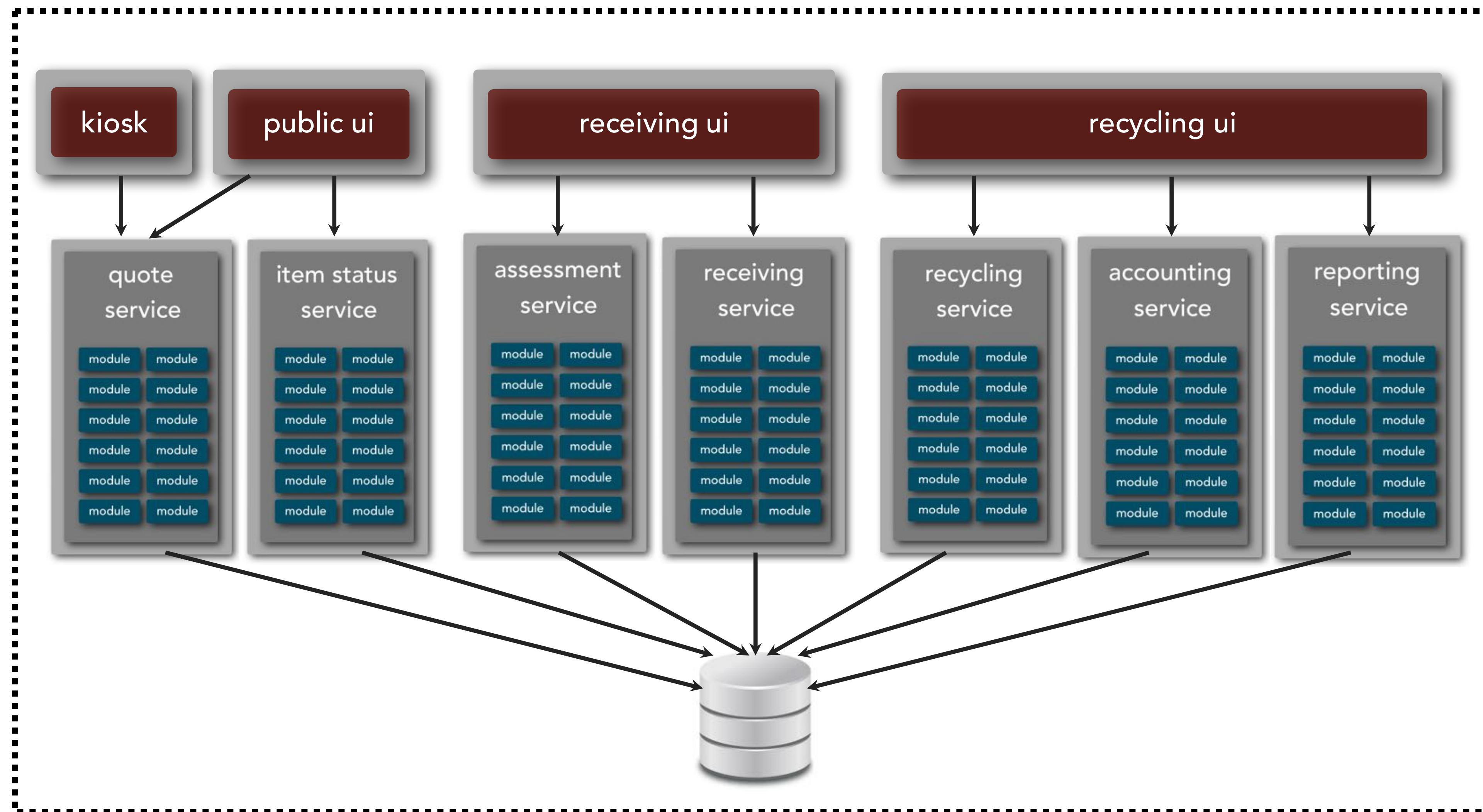
security
data integrity
auditability

recycling, reporting,
and accounting

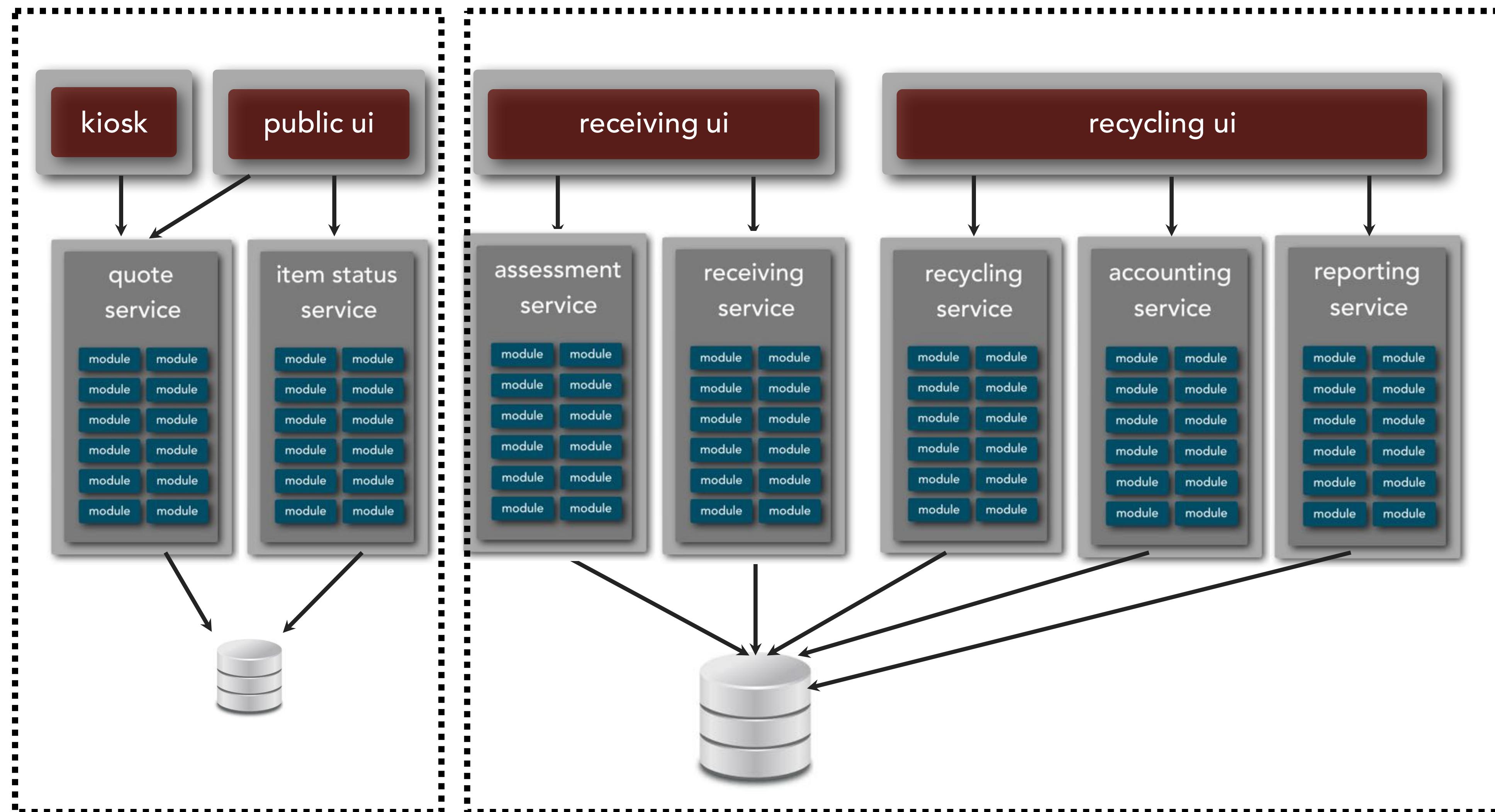
architectural quantum ?



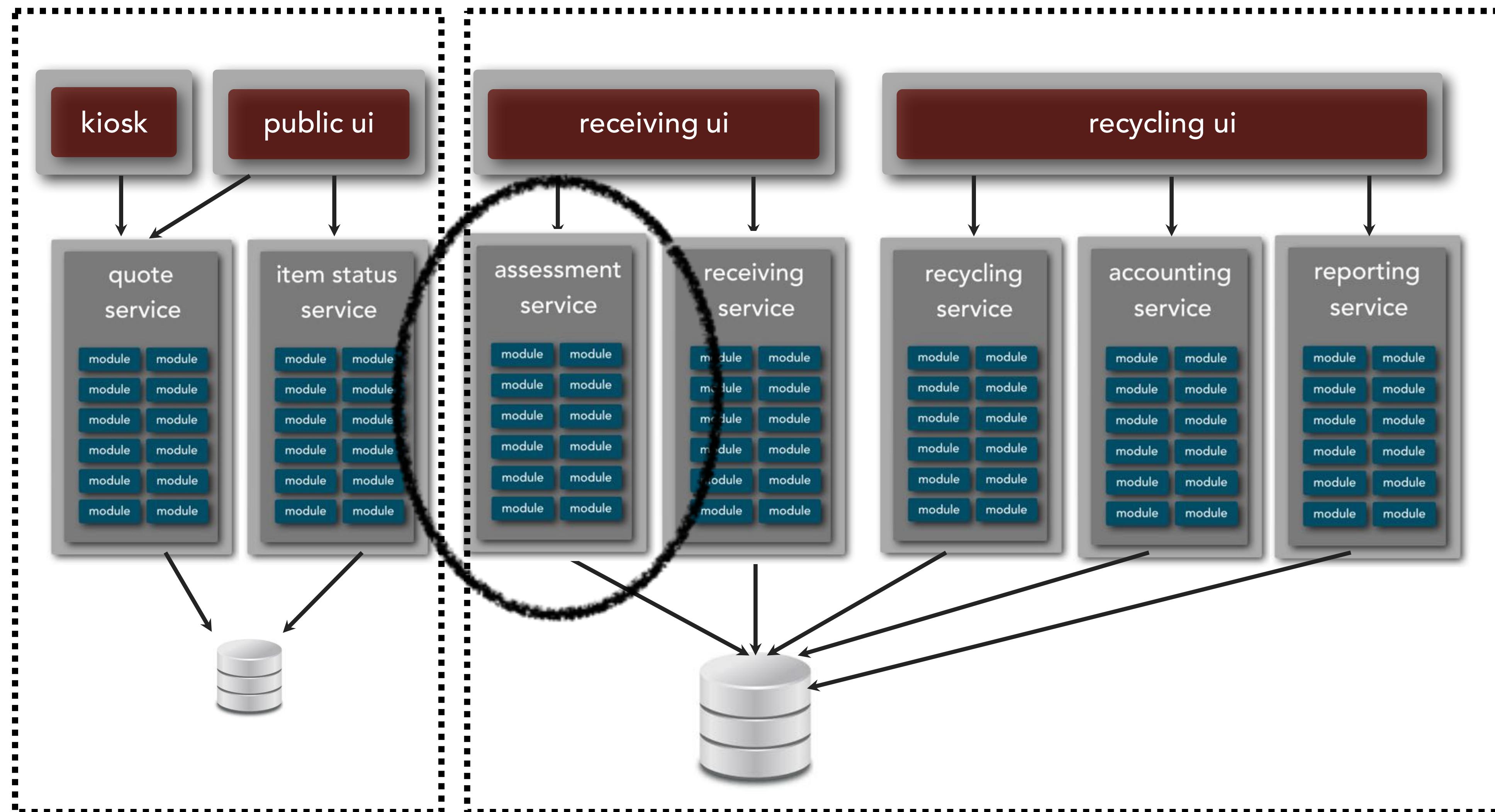
architectural quantum



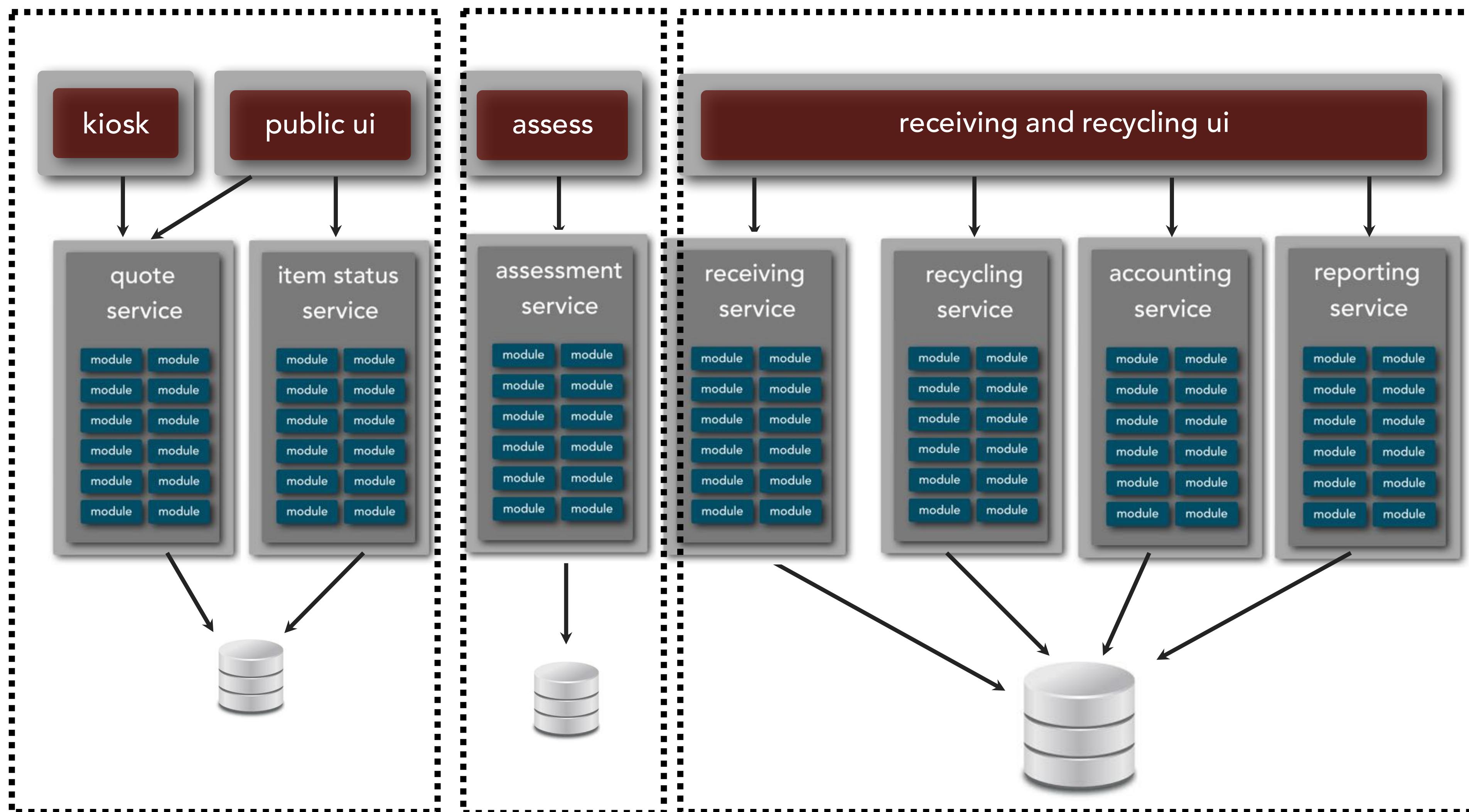
electronics recycling application



electronics recycling application

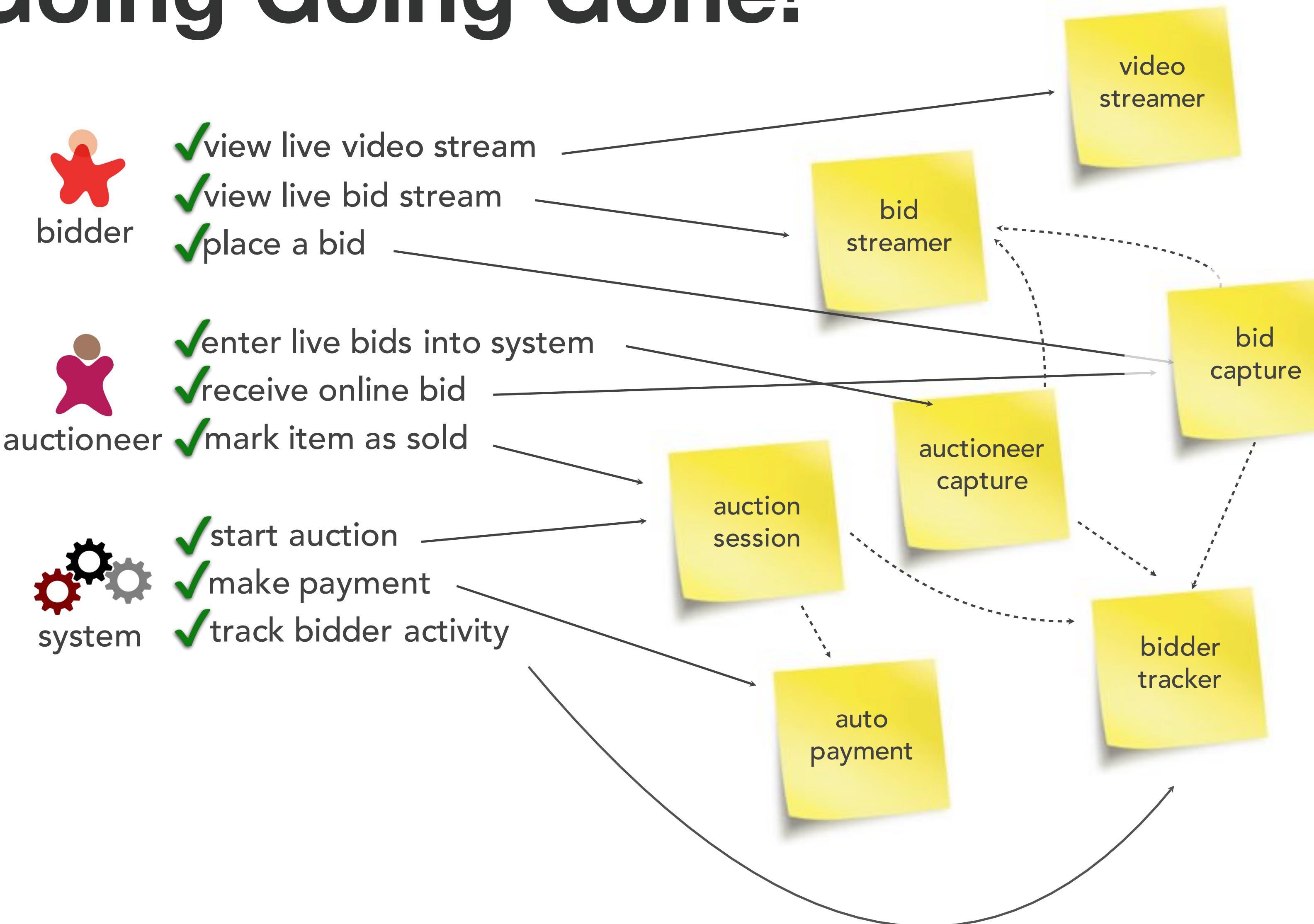


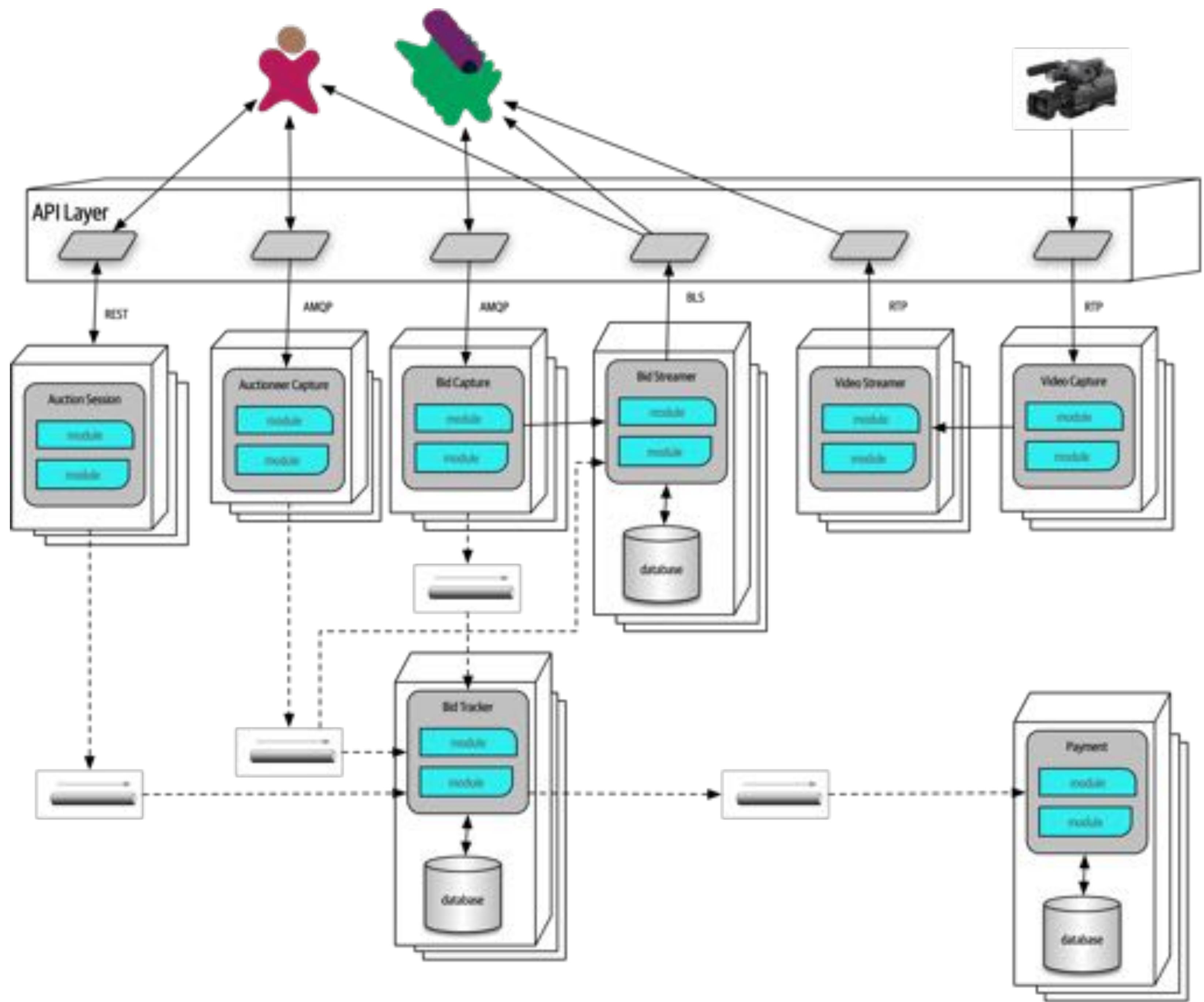
electronics recycling application

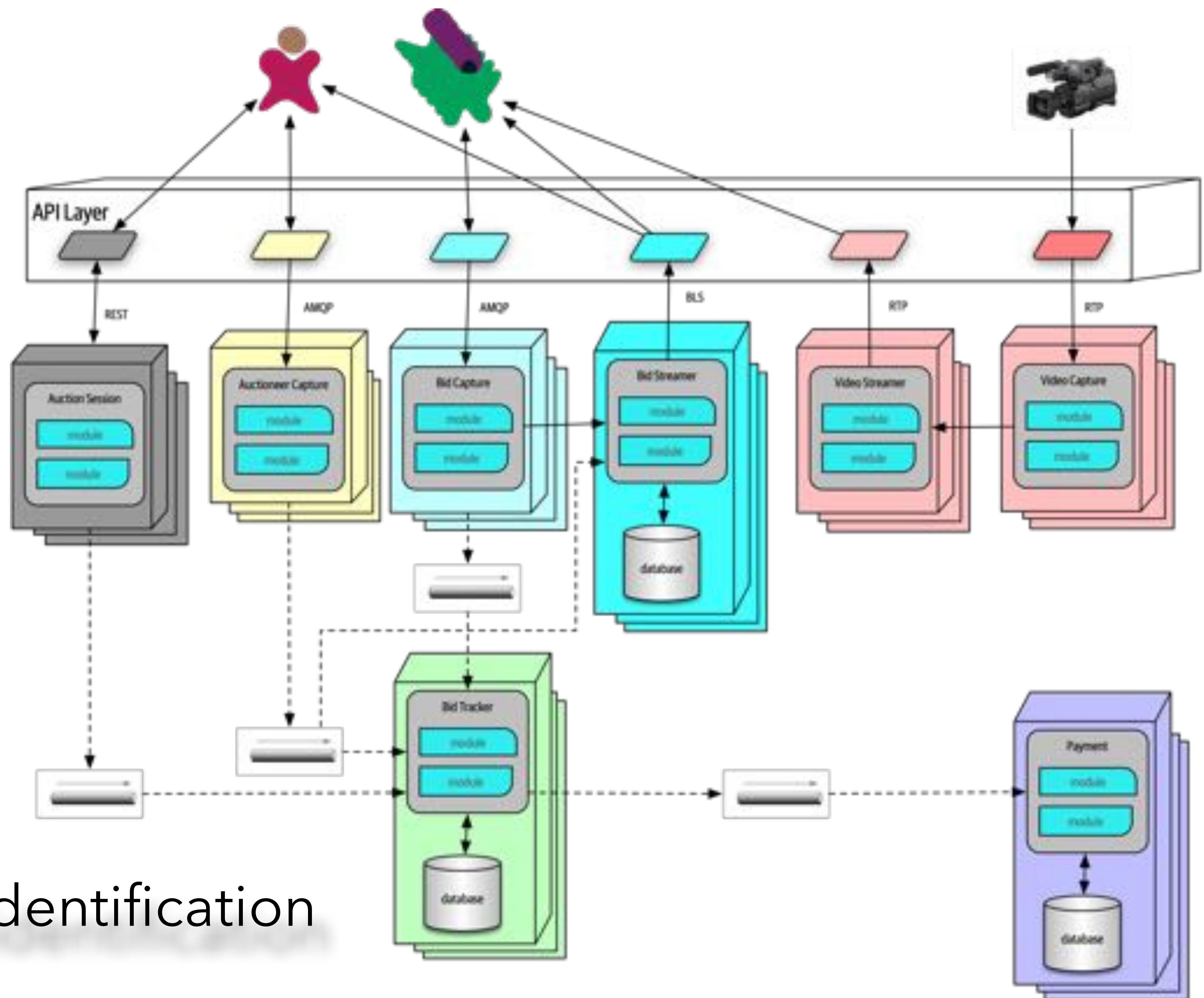


Your Architectural Kata is...

Going Going Gone!







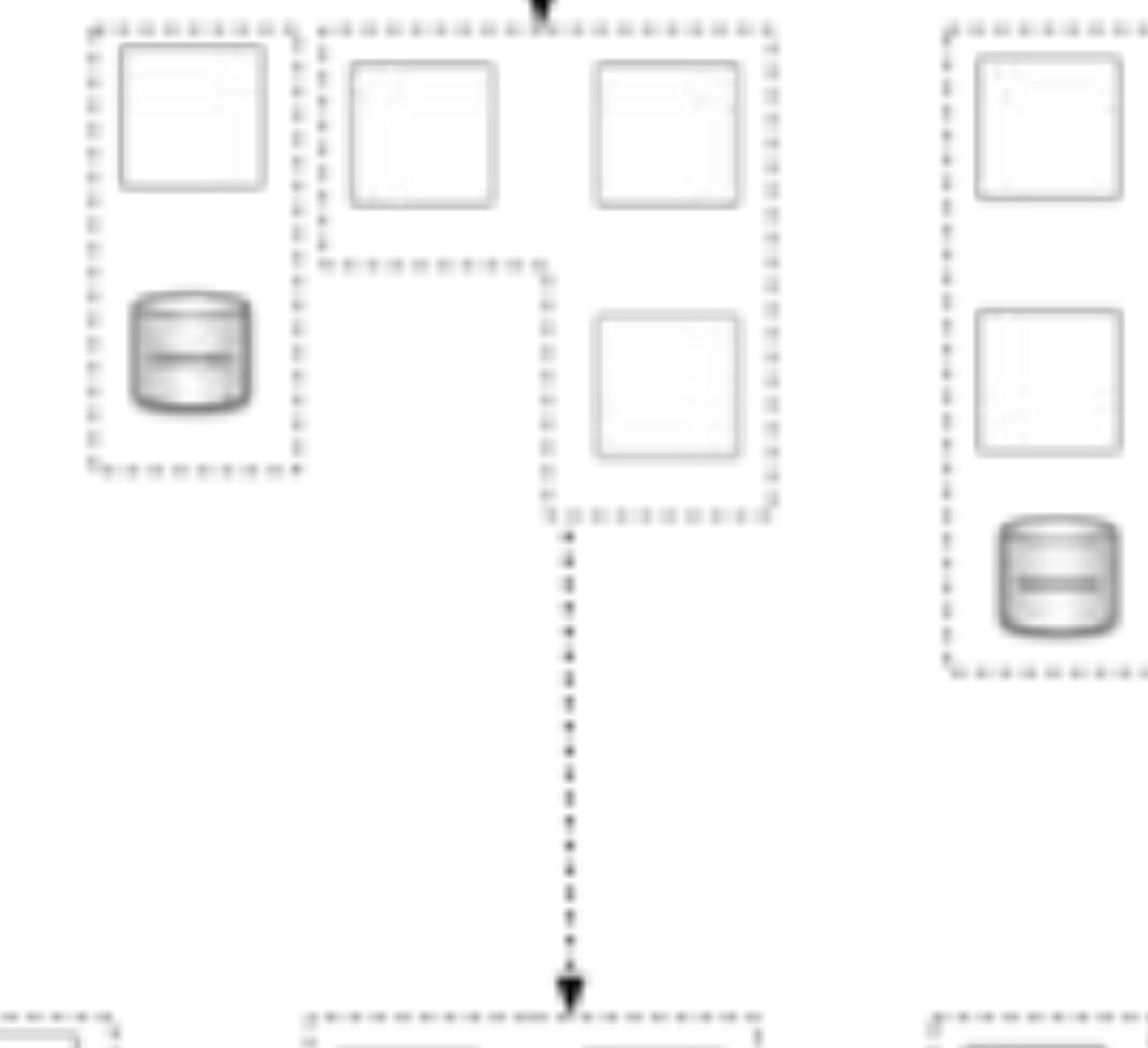
quanta identification

monolith | distributed?

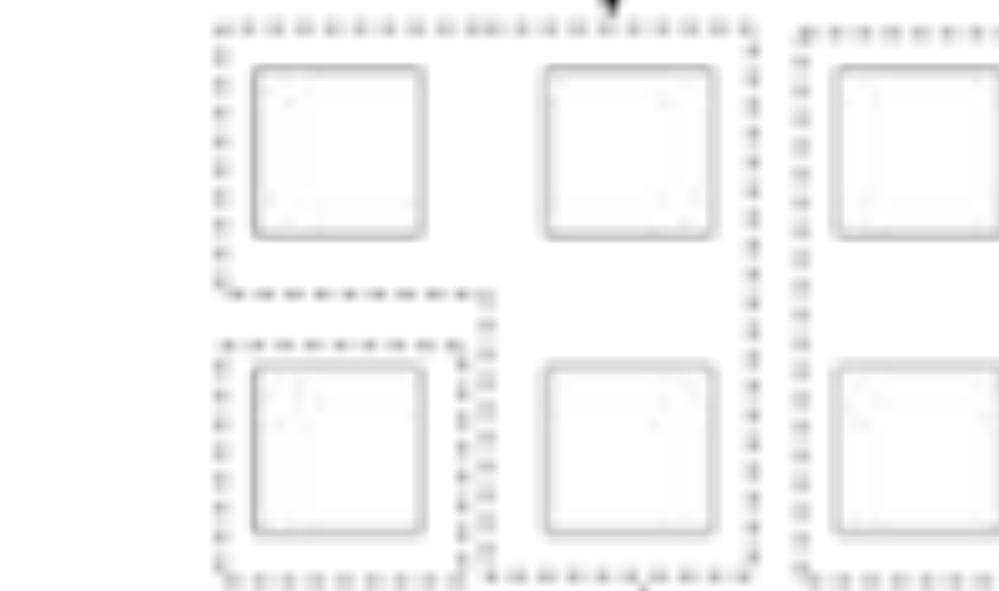
2. Determine persistence



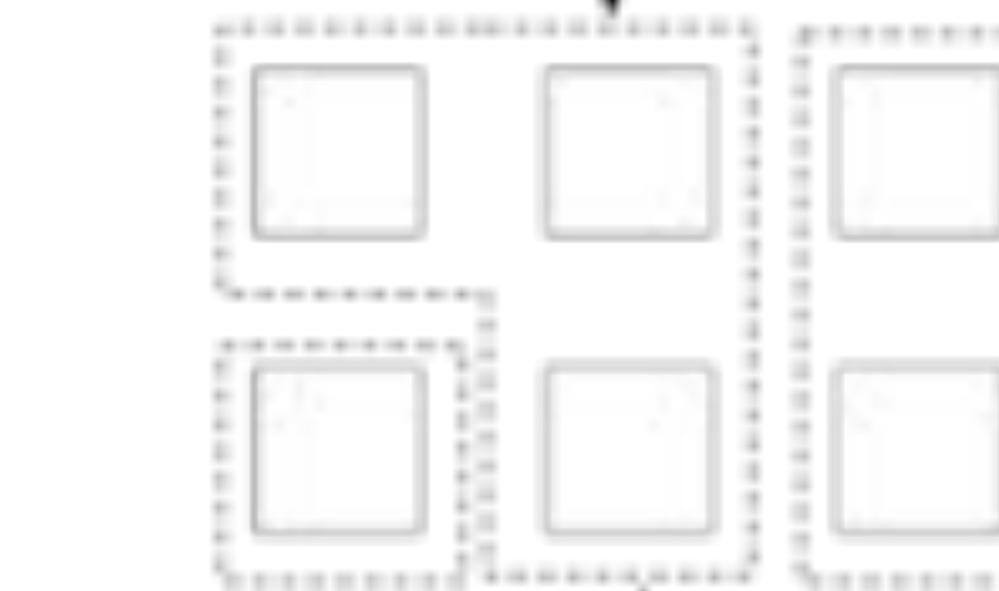
Monolithic architecture



3. Determine persistence



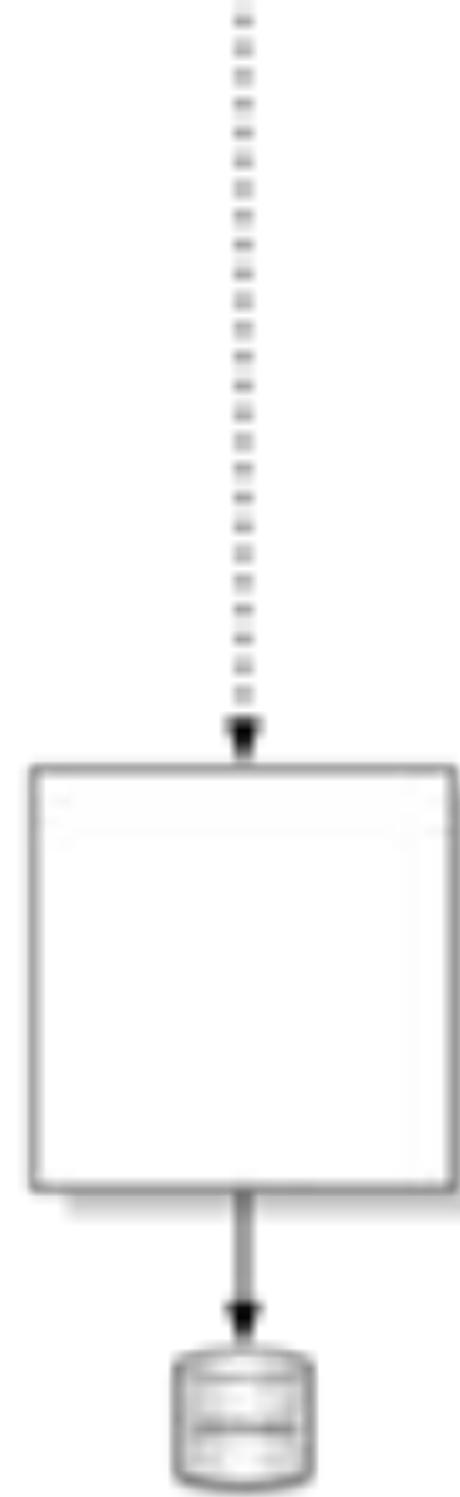
2. Choose quanta



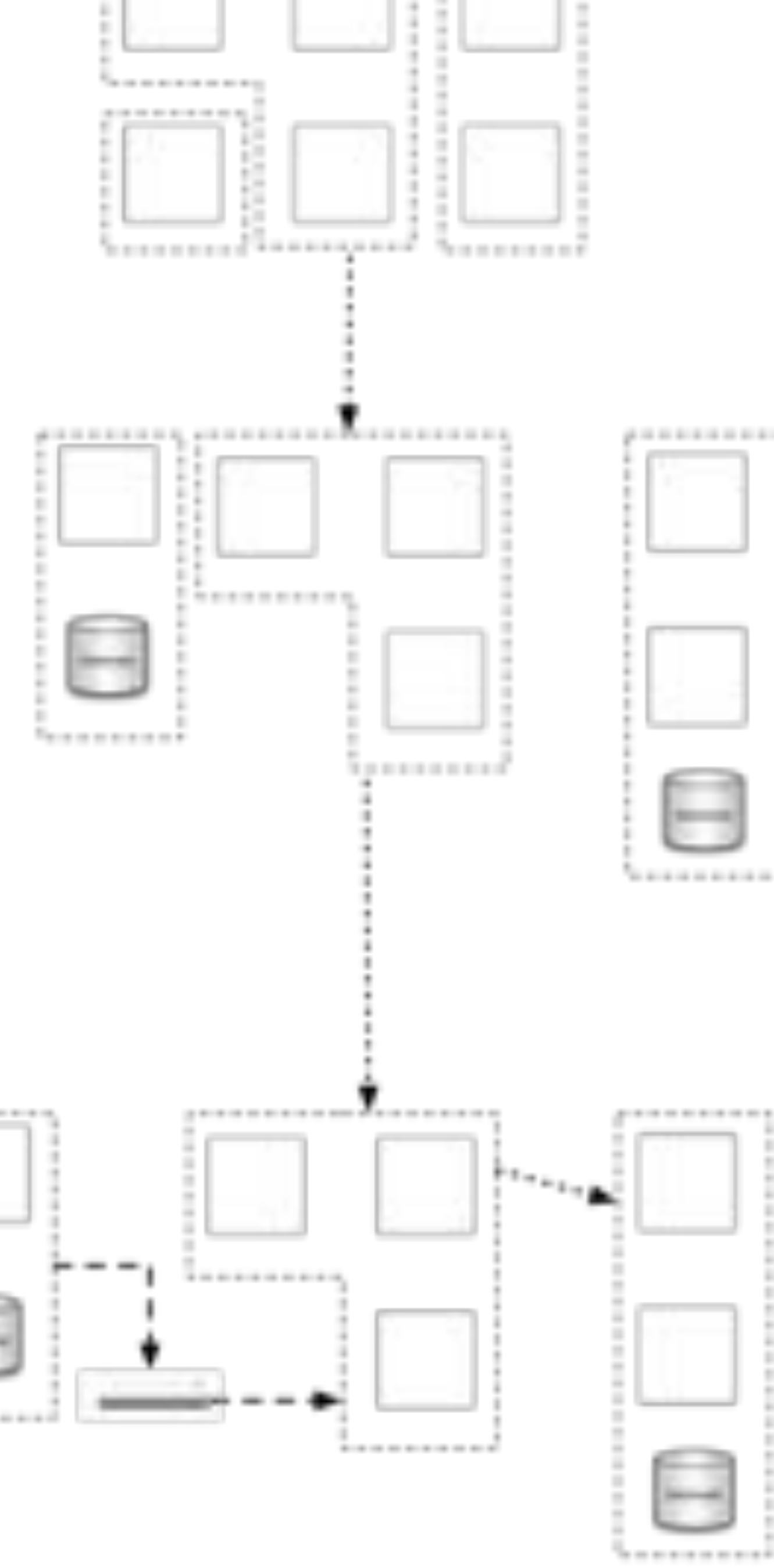
Distributed architecture

monolith | distributed?

2. Determine persistence

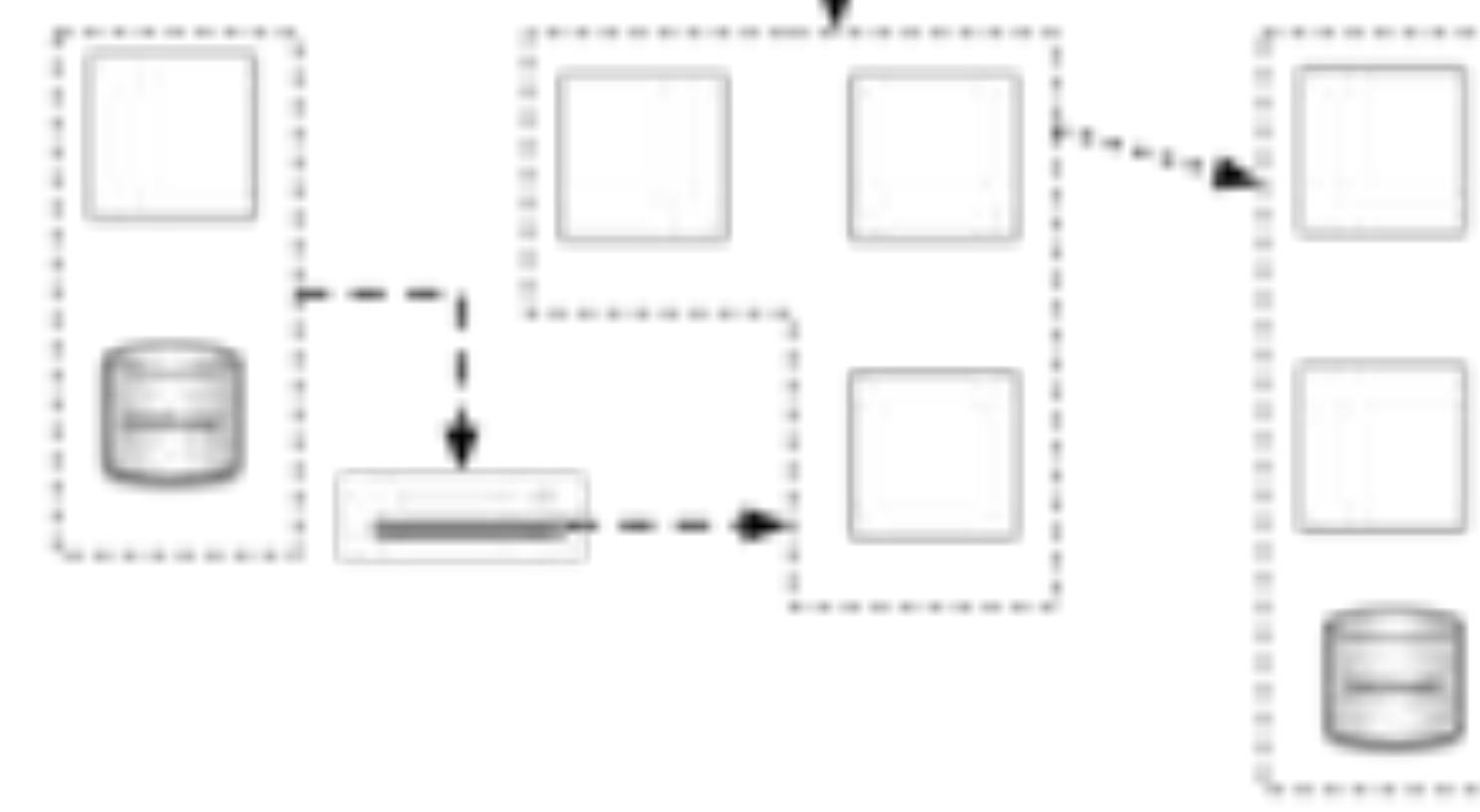


2. Choose quanta

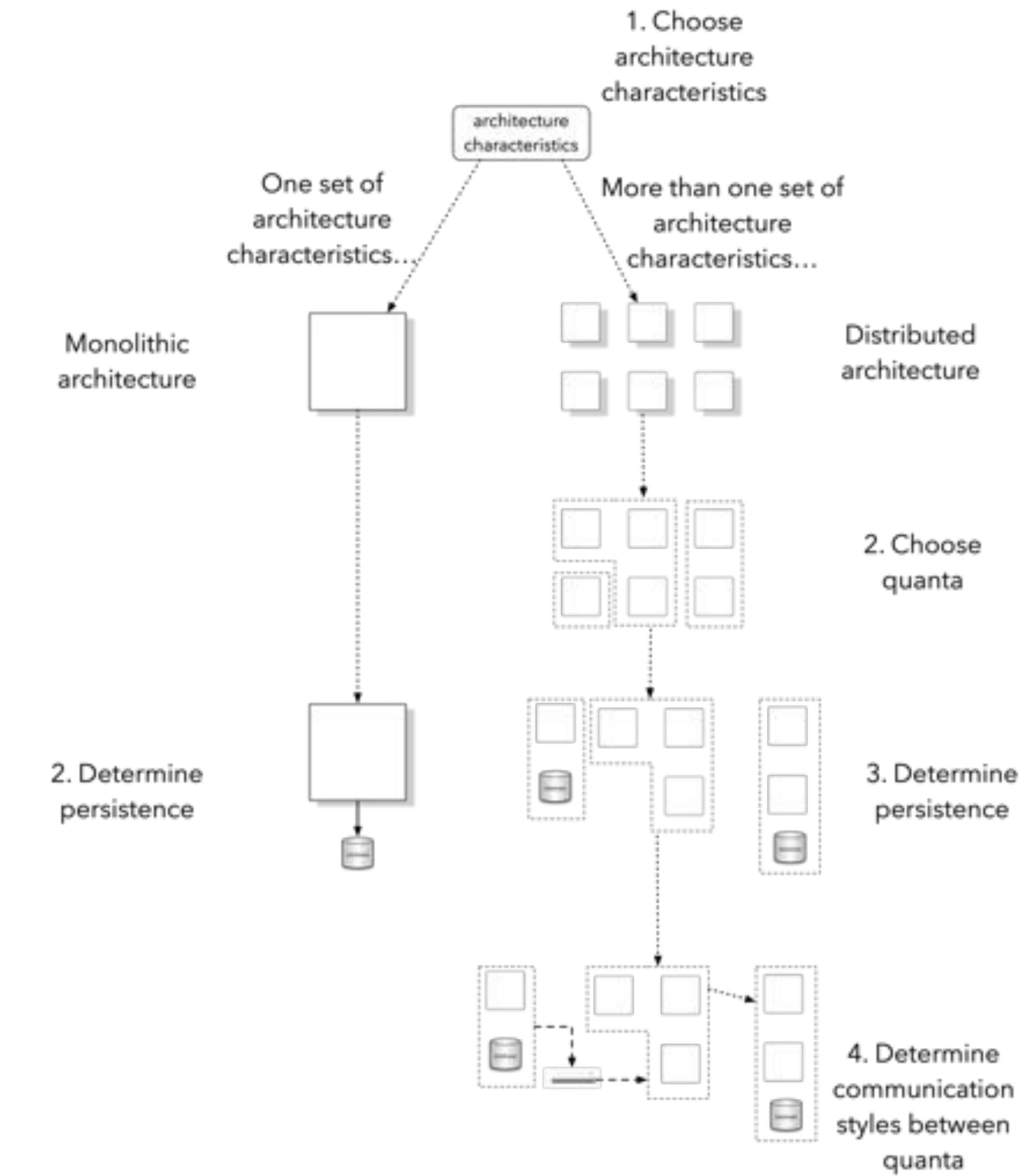


3. Determine persistence

4. Determine communication styles between quanta



monolith | distributed ?



monolith | distributed ?

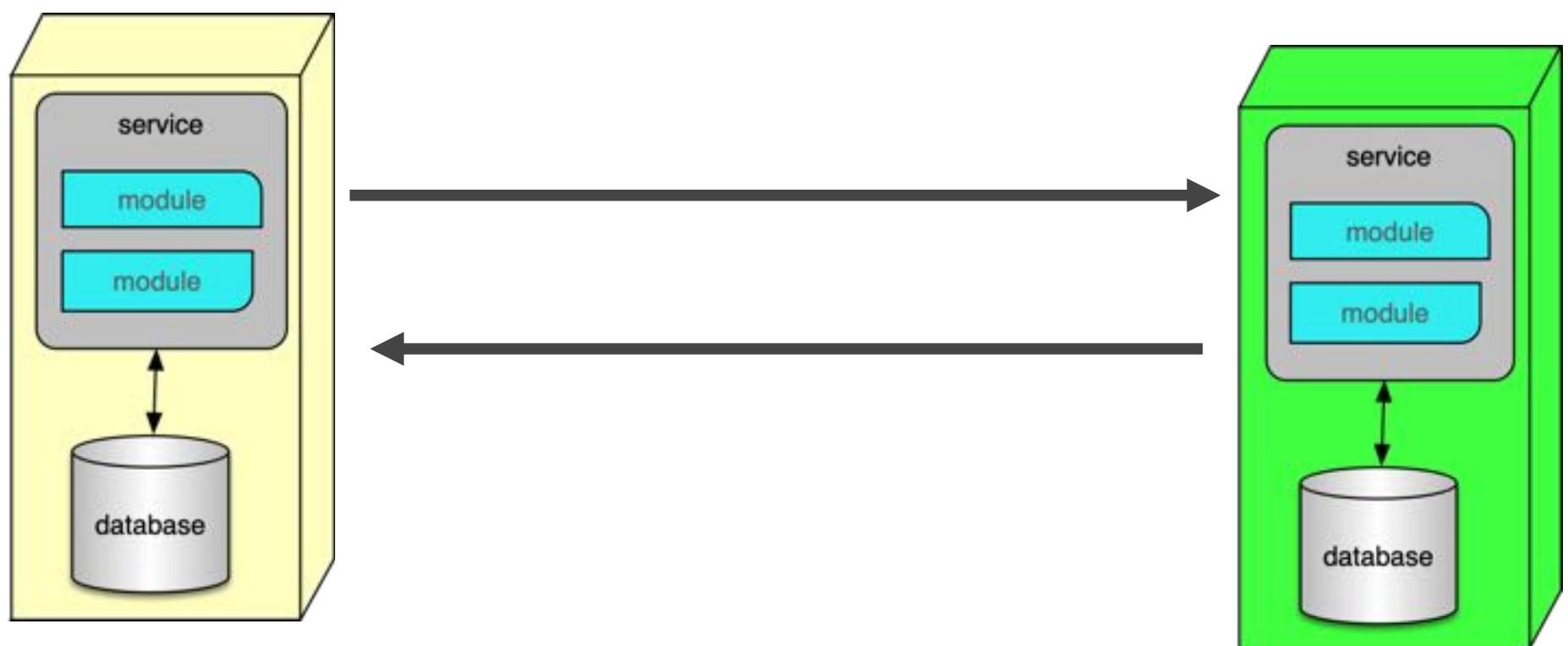
distributed architecture
building blocks

synchronous | asynchronous ?

sync | async ?

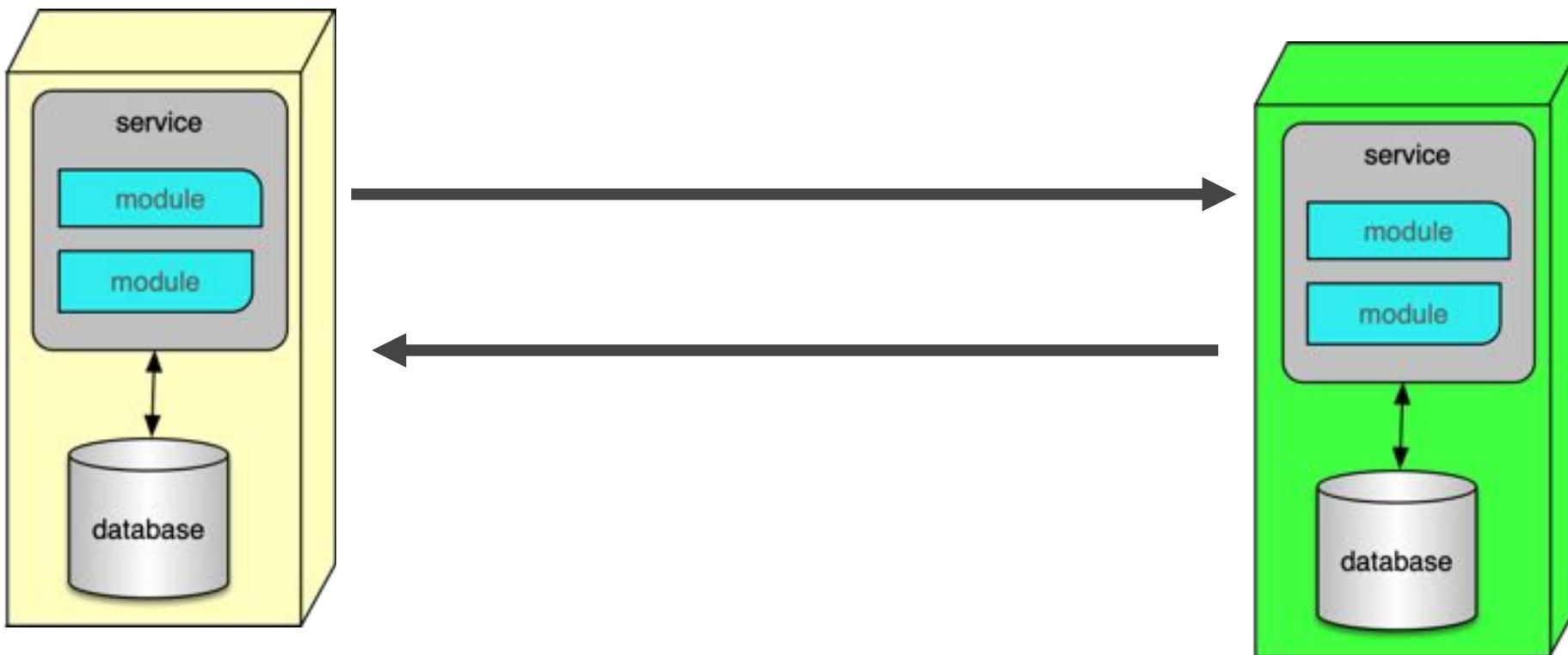
sync | async ?

Caller waits (blocks) for response

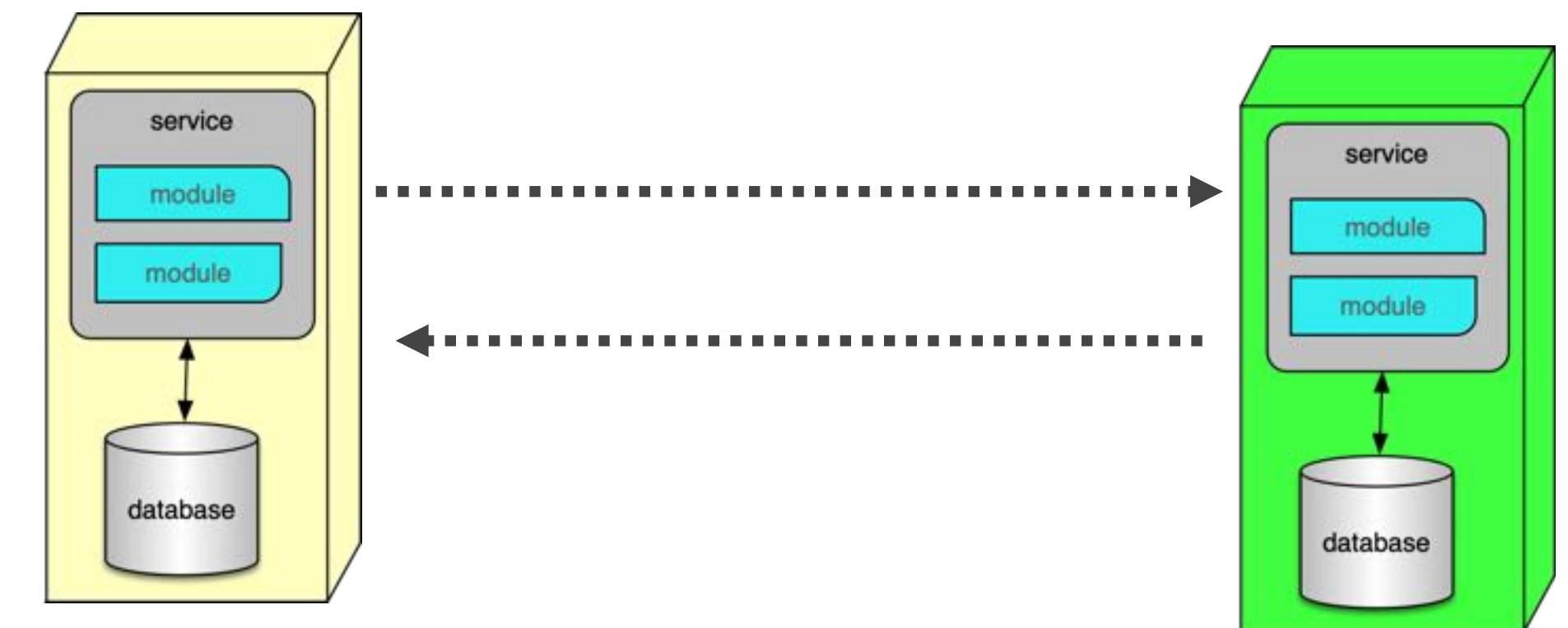


sync | async ?

Caller waits (blocks) for response

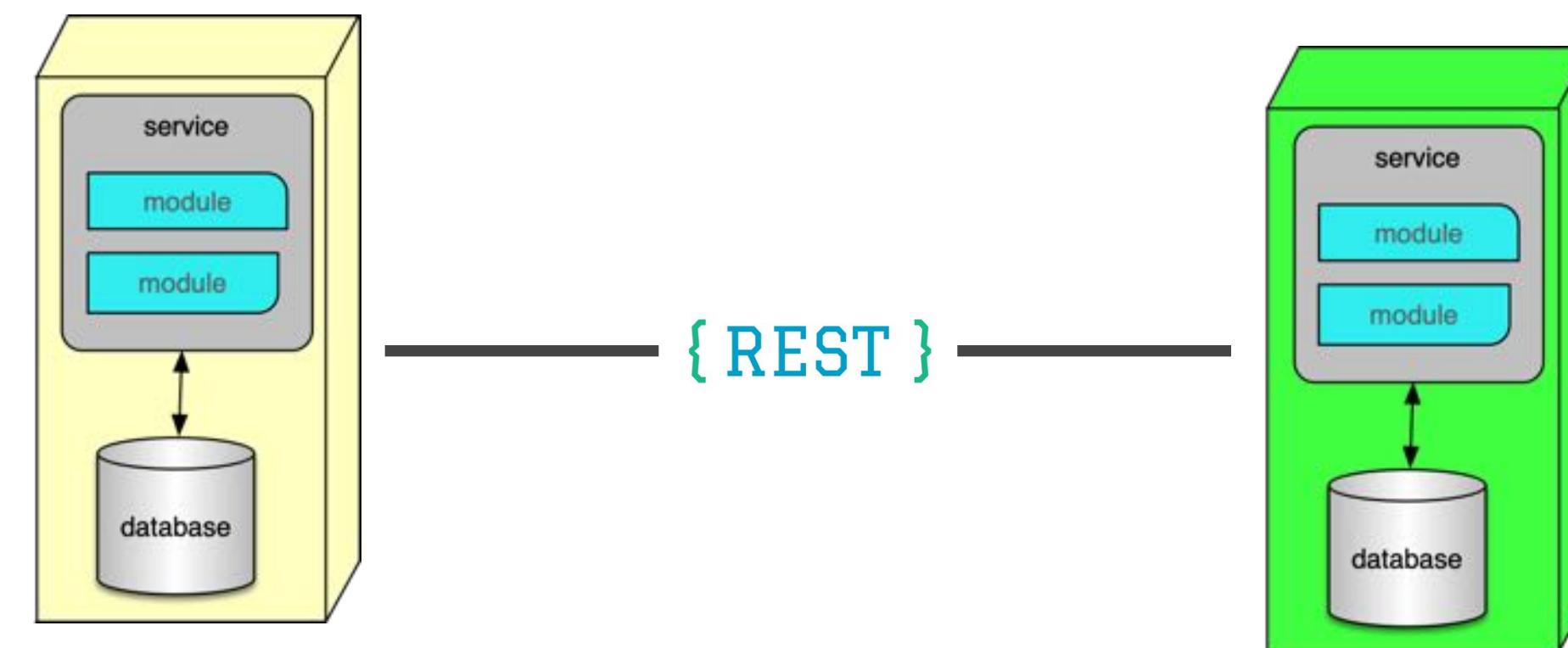


Caller receives response asynchronously



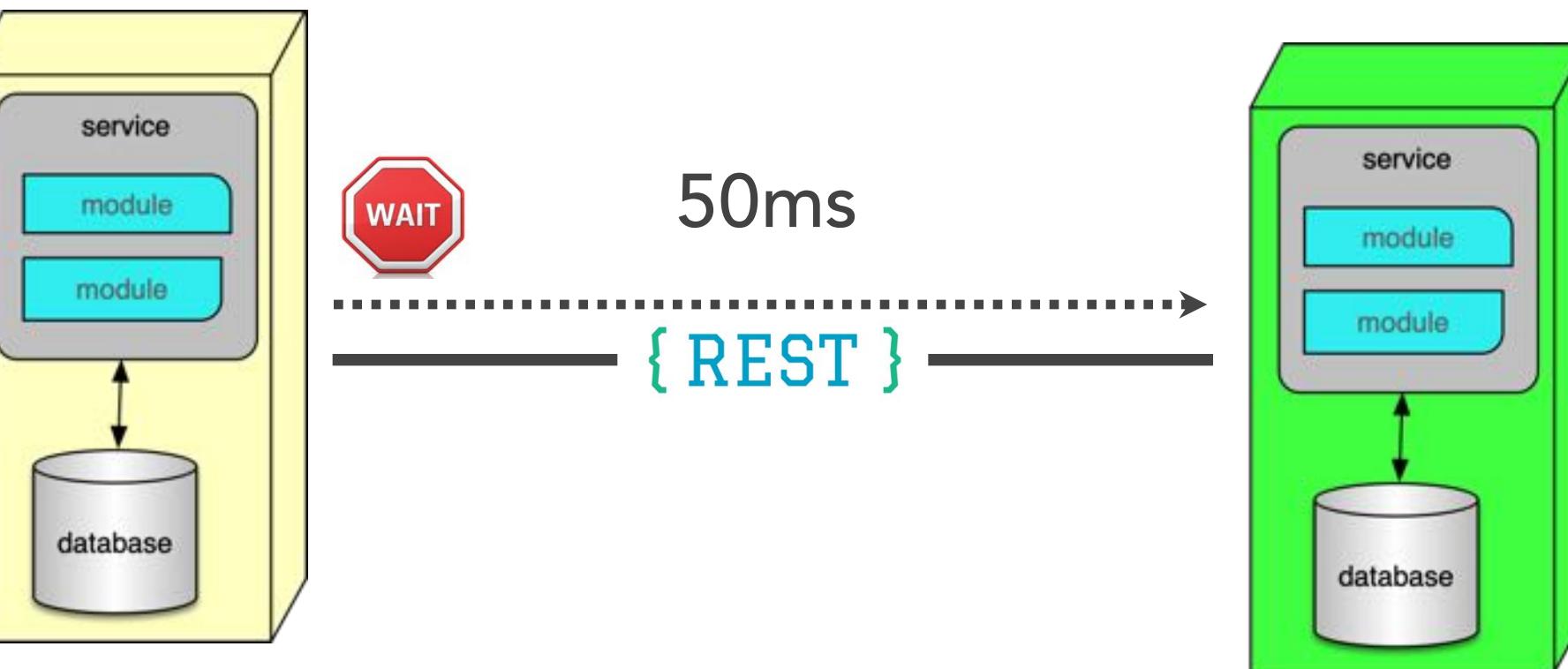
sync | async ?

synchronous



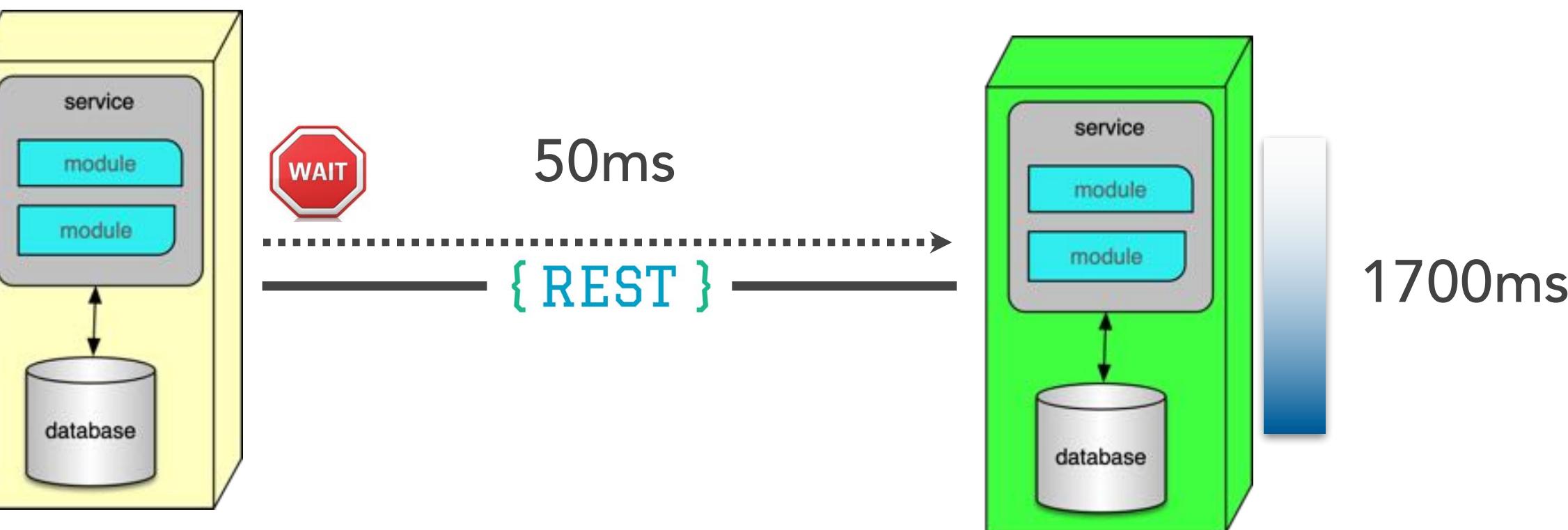
sync | async ?

synchronous



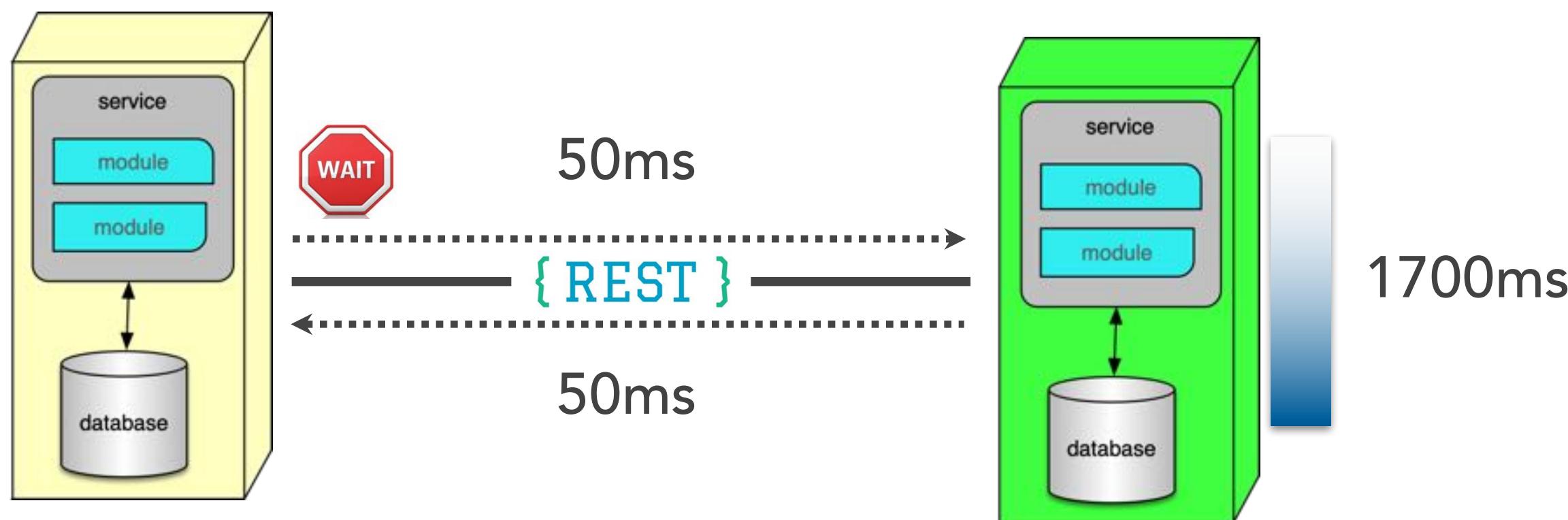
sync | async ?

synchronous



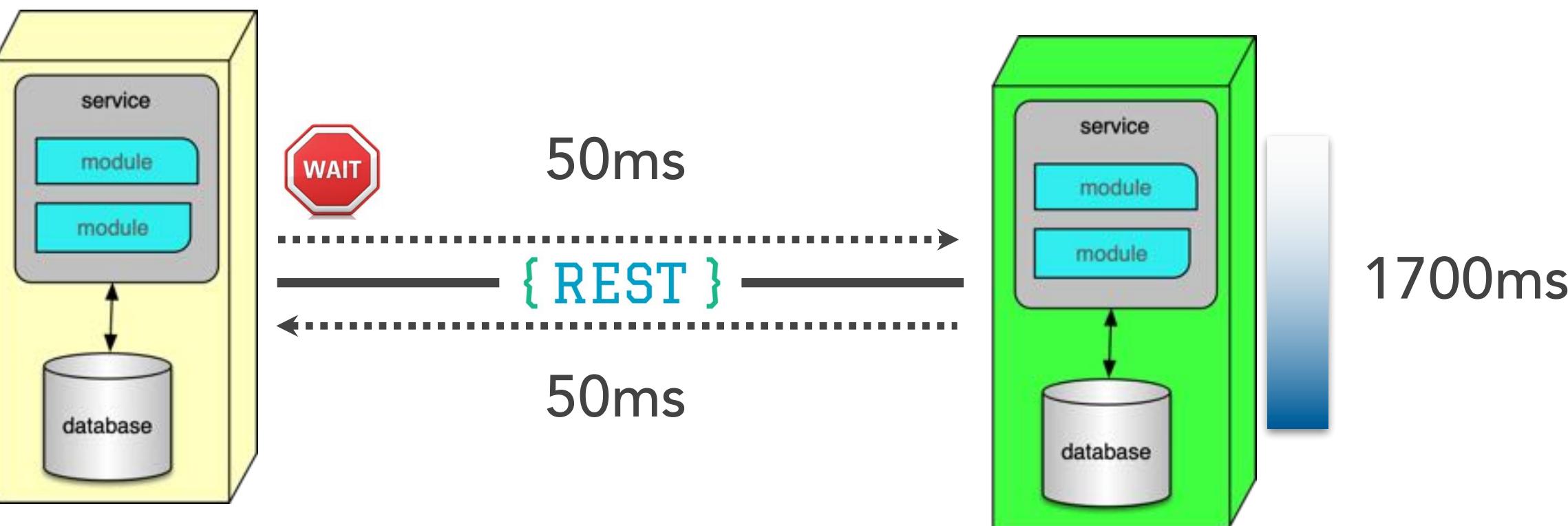
sync | async ?

synchronous



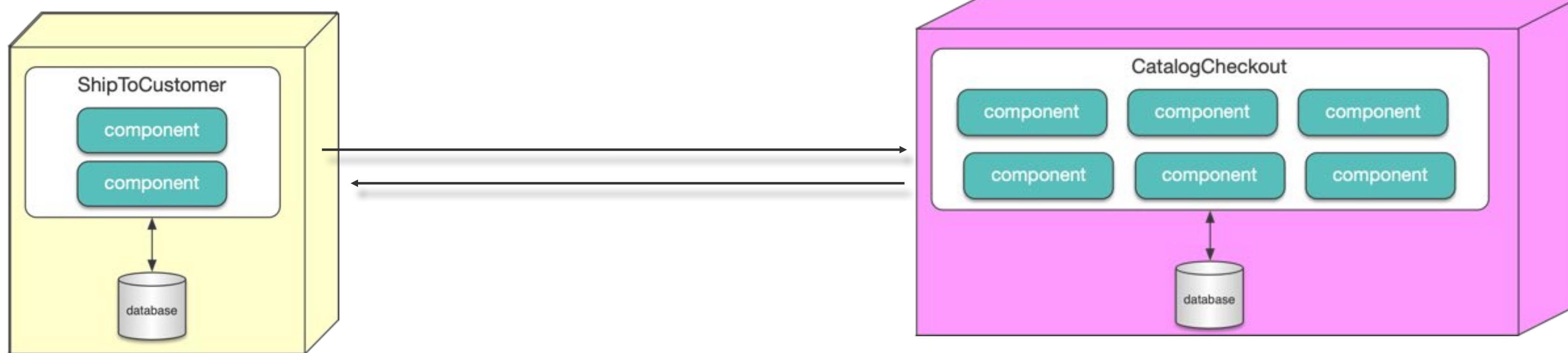
sync | async ?

synchronous



sync | async ?

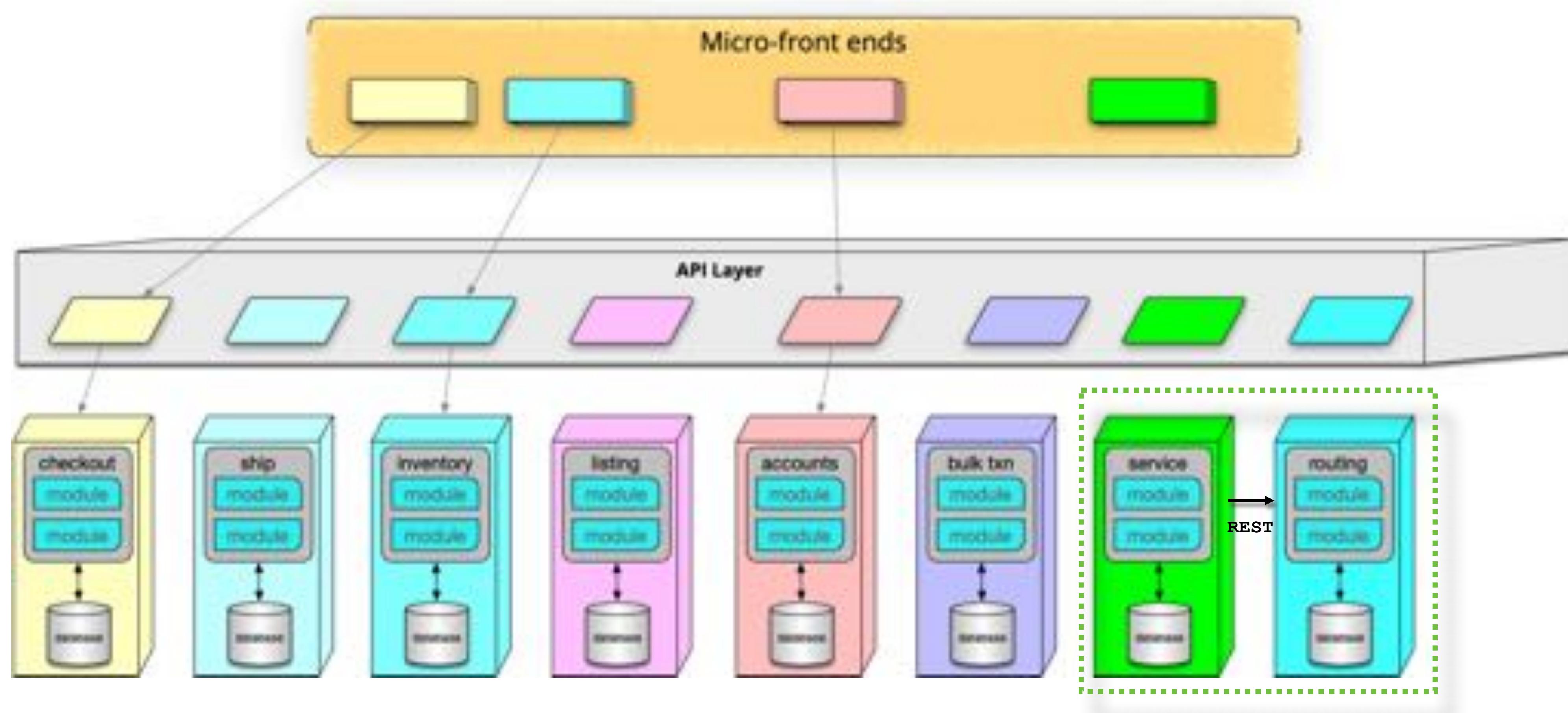
synchronous



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

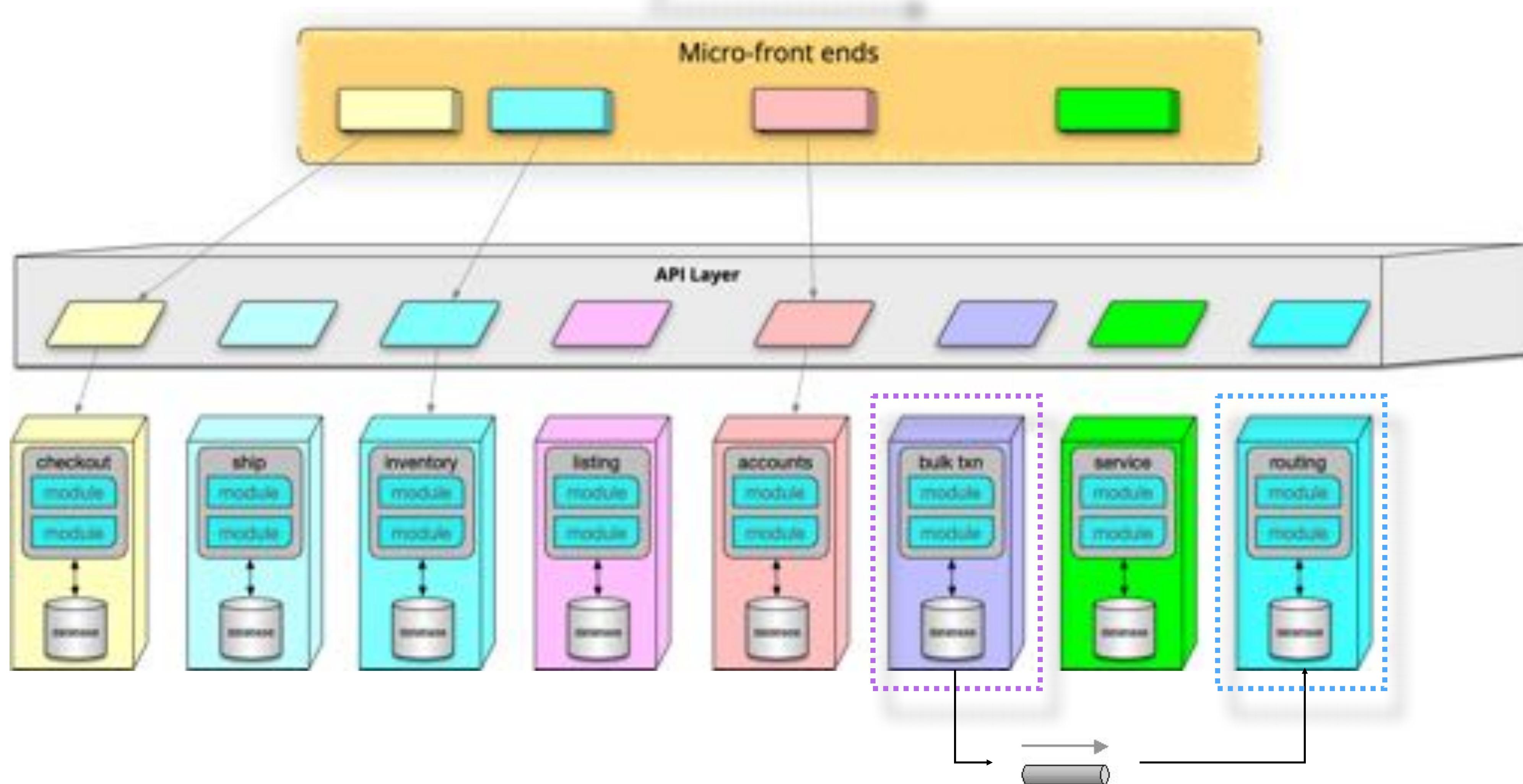
sync | async ?

synchronous



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 ?

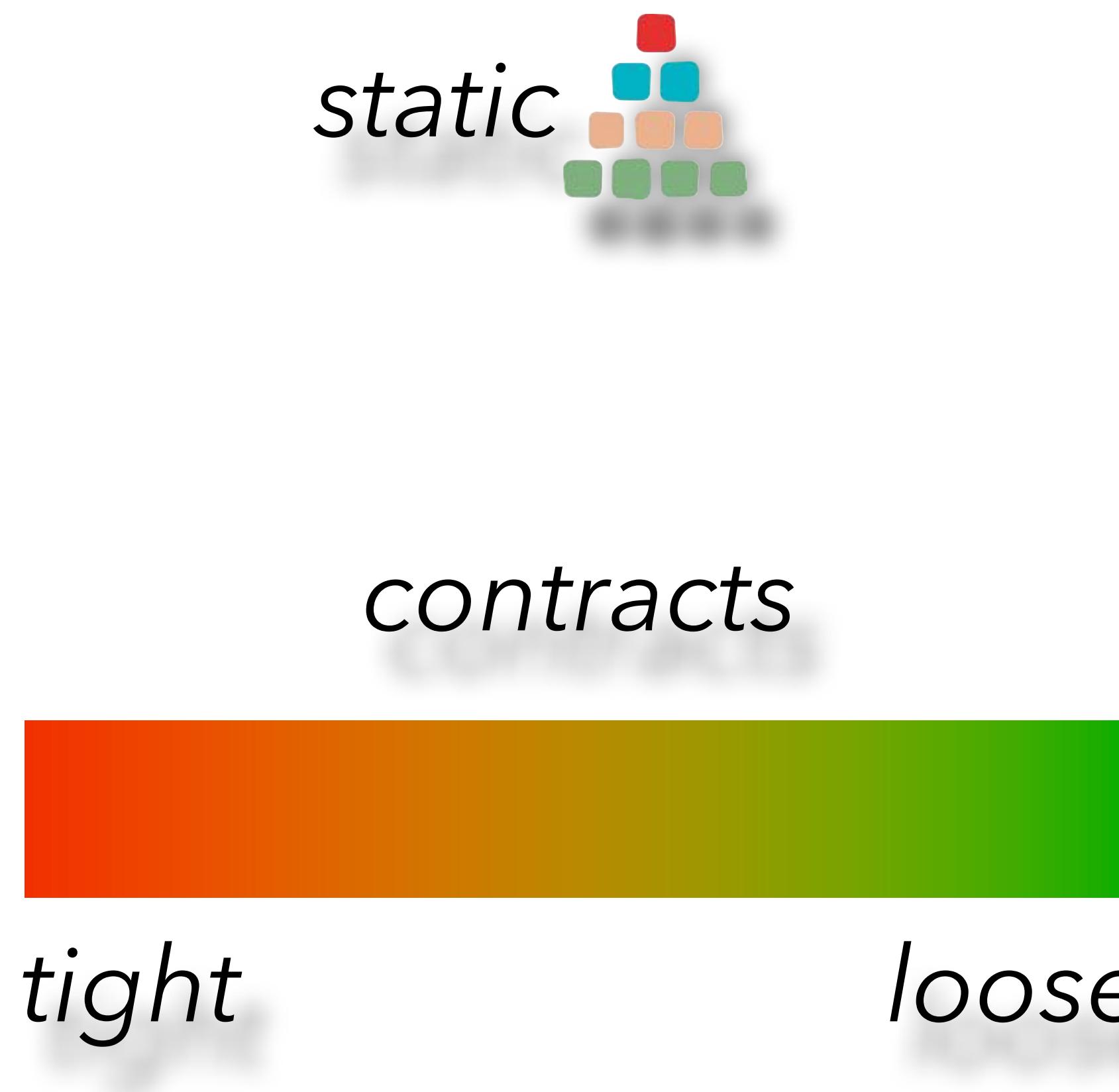


- architecture concern: synchronous versus asynchronous
 - exception: implementation detail is important/unique
- design: how to implement the architecture concern
- use layers in drawing tools to differentiate

contracts | versions ?

contracts | versions ?

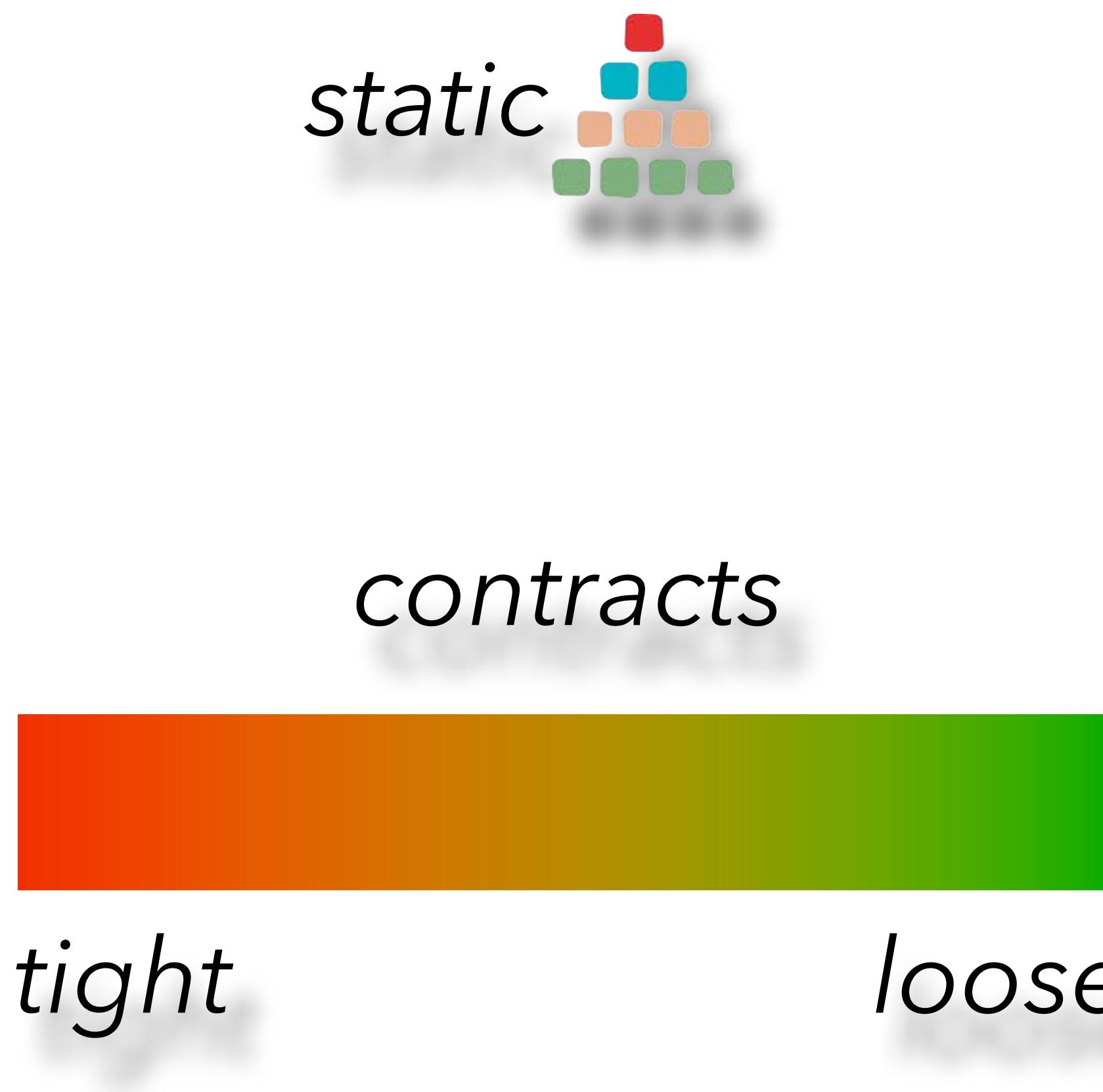
quantum connascence



impact on operational
(and other) architecture characteristics

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

quantum connascence



how quanta communicate

impact on operational
(and other) architecture characteristics

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

contracts

tight



loose

method signatures

resources

name/value pairs

contracts

tight



loose

method signatures

resources

name/value pairs

SOAP

contracts

tight

loose

method signatures

resources

name/value pairs

SOAP

Buffer Protocols (gRPC)

RPC*

contracts

tight



loose

method signatures

resources

name/value pairs

SOAP

Buffer Protocols (gRPC)

RPC*

contracts

tight

loose

method signatures

resources

name/value pairs

SOAP

REST

Buffer Protocols (gRPC)

RPC*

contracts

tight

loose

method signatures

resources

name/value pairs

SOAP

REST

Buffer Protocols (gRPC)

GraphQL

RPC*

contracts

tight

loose

method signatures

resources

name/value pairs

SOAP

REST

JSON

Buffer Protocols (gRPC)

GraphQL

RPC*

Some GraphQL proponents claim that GraphQL is “a better REST”.

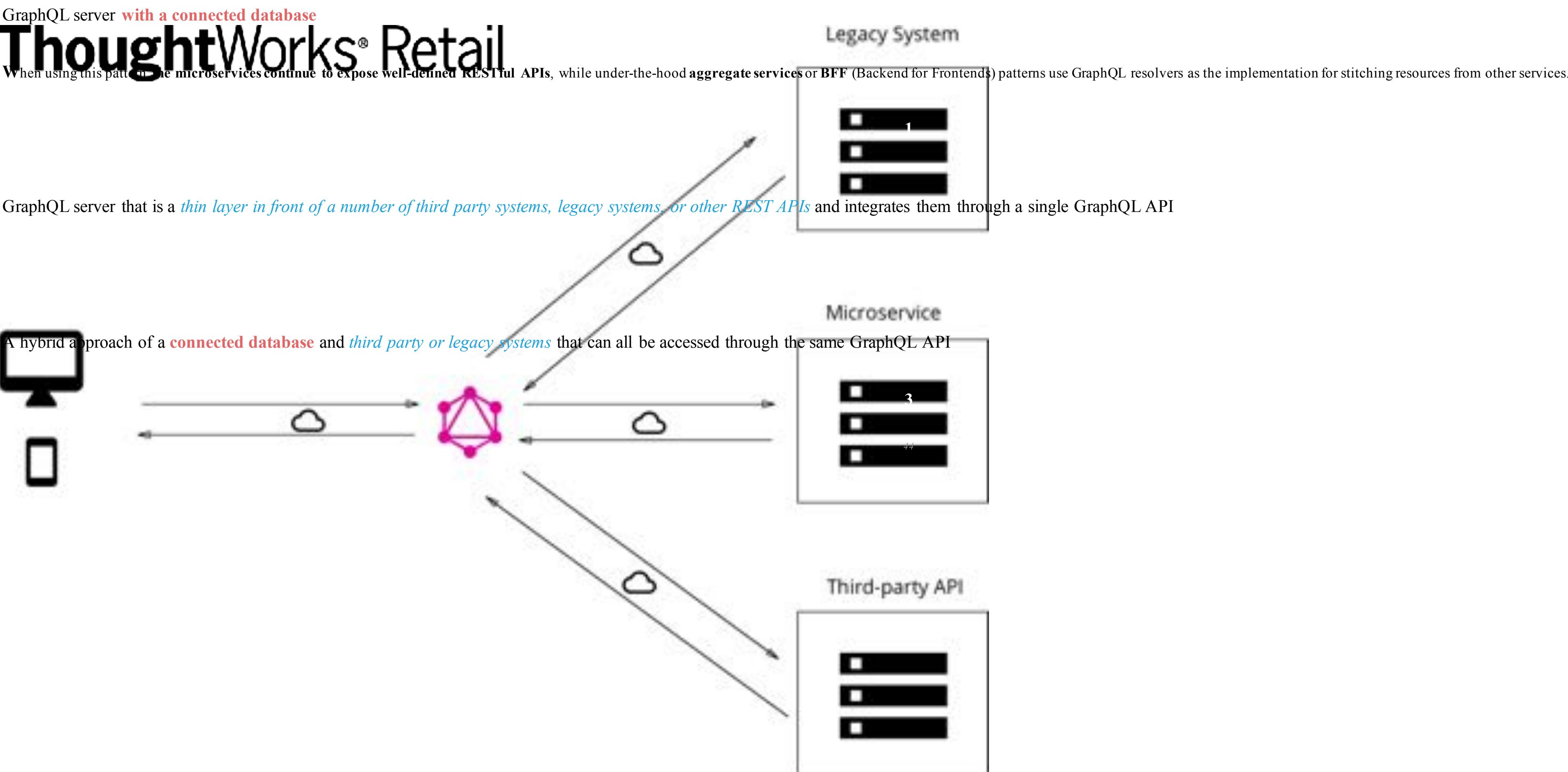
We have seen some criticism towards usage of REST versus GraphQL, due to possibly chatty, inefficient interactions among systems and failing to adapt as client needs evolve. **However, in our experience, it isn't REST that causes these problems.** Rather, they stem from a failure to properly model data.

In other words, naively developing services that simply expose static, hierarchical data models via templated URLs result in an **anemic REST** implementation. GraphQL is not the solution for this problem.

ESTFUL API or GraphQL

We recommend the use of GraphQL for server-side resource aggregation only.

Broadly speaking, there are 3 use cases that GraphQL may be used for:



Backend for Frontend (BFF)

The Backend for Frontend (BFF) Pattern was originated at SoundCloud. They originally had a monolithic API serving all downstream (mostly user-facing) applications. As the needs of different downstream applications started to diverge and any changes made to the monolithic API could affect other downstream applications, this approach started to become problematic.

For specific applications or features that needed fine-tuned APIs to deliver a good user experience, they then decided to build a backend API for those specific use cases.

In SoundCloud's original implementation,

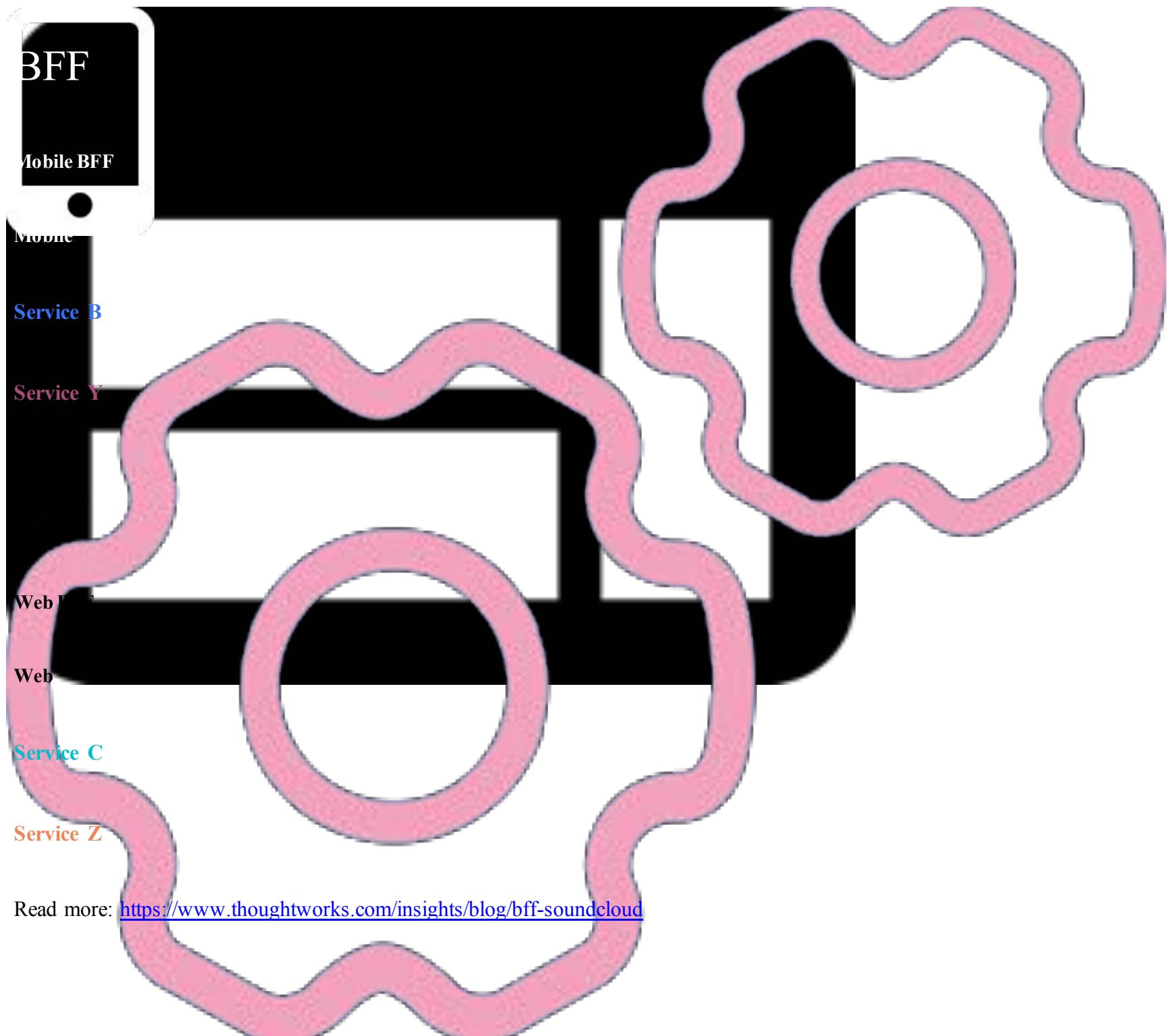
The BFF isn't an API used by the frontend application.

The BFF is considered part of the frontend, owned by the same team.

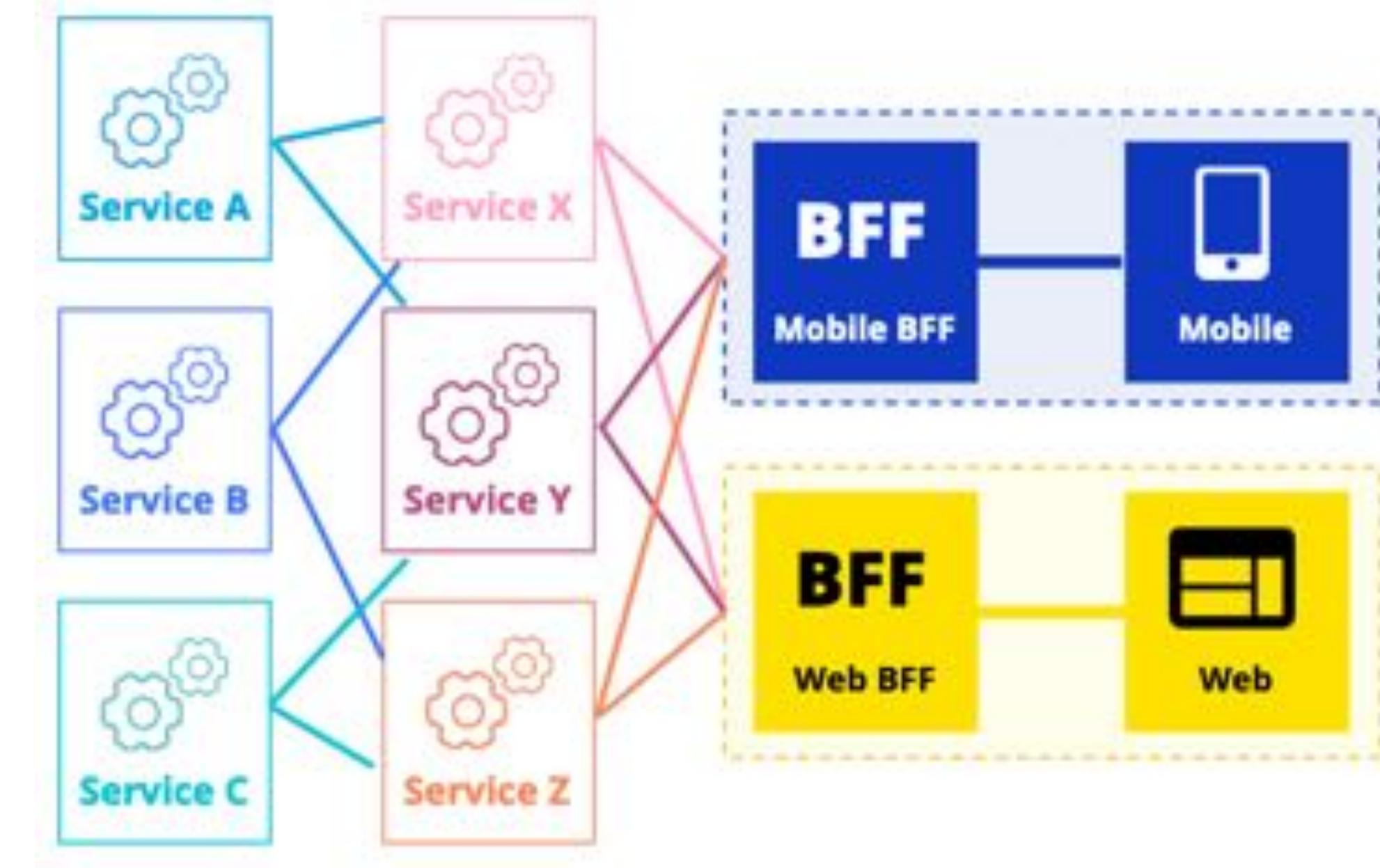
Having an independent backend the application team controls that fits their specific needs means they can deliver value without the friction introduced by the formerly monolithic API.

Service A

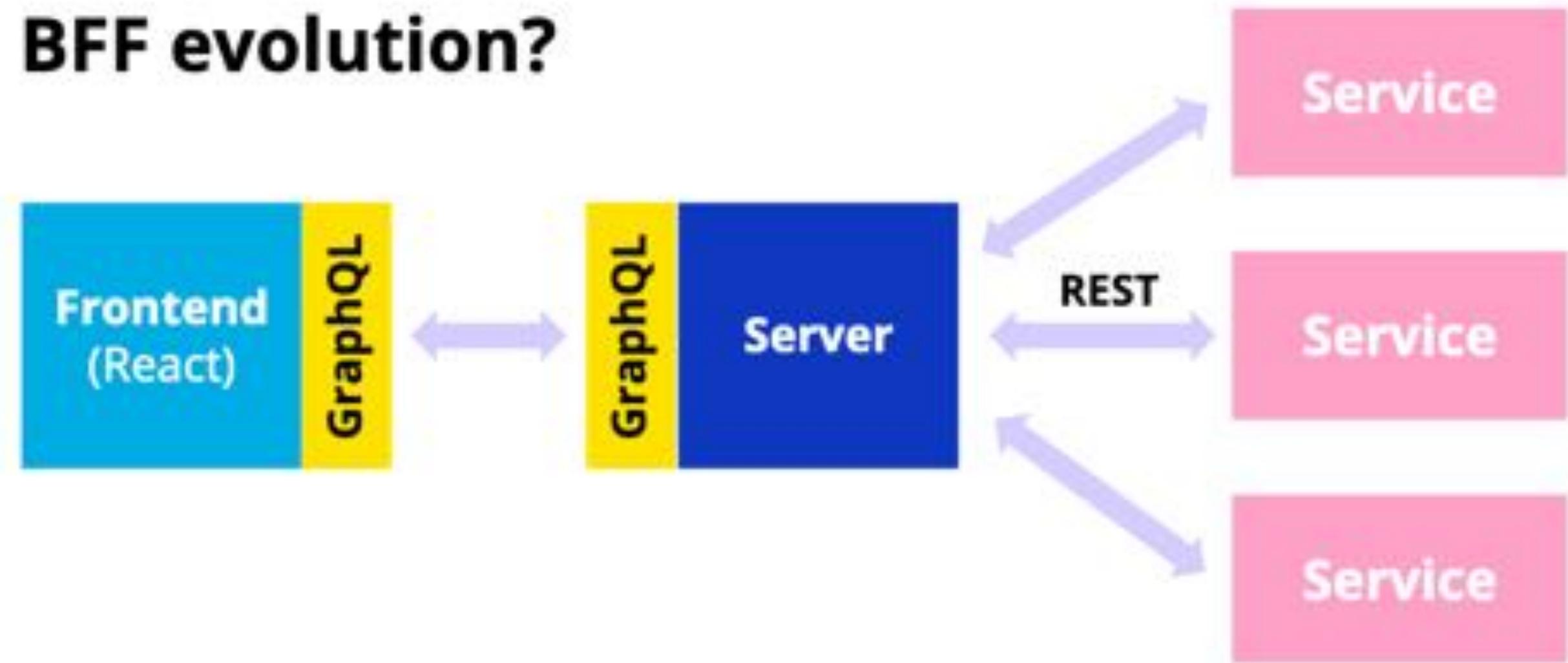
Service X

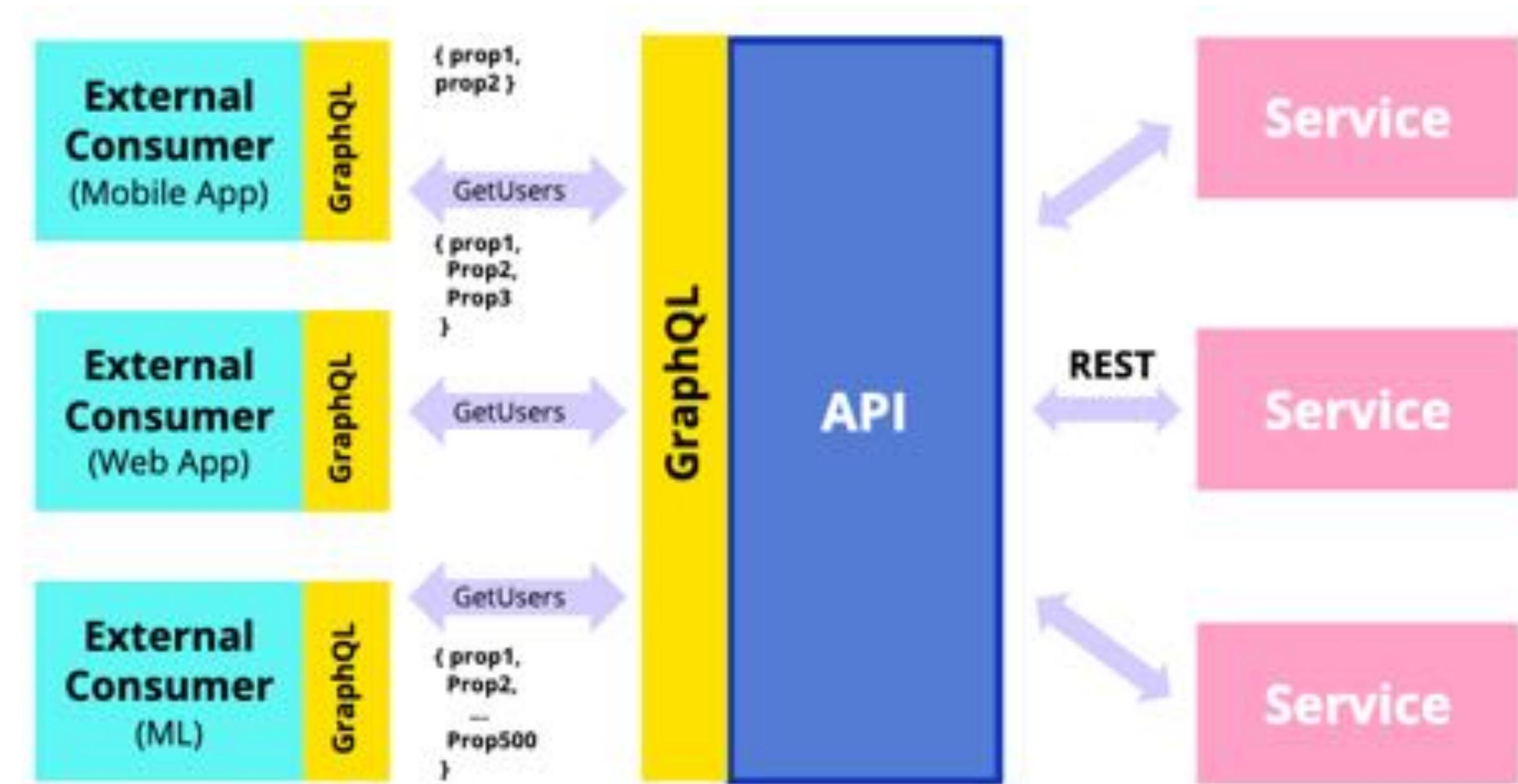


Read more: <https://www.thoughtworks.com/insights/blog/bff-soundcloud>



BFF evolution?







Main benefits:

- 1 Flexibility - different clients can specify what data they need from the provider
- 2 The provider gets to find out which fields each client cares about



Even with a specific use case in mind (such as server-side aggregation), technologies like GraphQL comes with its own learning curve.

What to watch out for:

- 1** Over-fetching data on the server side due to resolver implementation
- 2** HTTP caching - it takes more effort to make this work for GraphQL endpoints
- 3** Error handling
- 4** Security (e.g. limiting query depth)
- 5** Contract testing is a bit different

```
Detected the following changes (6) between schemas:  
  
✖ Field posts was removed from object type Query  
✖ Field modifiedAt was removed from object type Post  
✓ Field Post.id changed type from ID to ID!  
✓ Deprecation reason on field Post.title has changed from No more used to undefined  
✓ Field Post.title changed type from String to String!  
✓ Field Post.createdAt changed type from String to String!  
  
error Detected 2 breaking changes
```

tight

loose

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

tight

loose

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

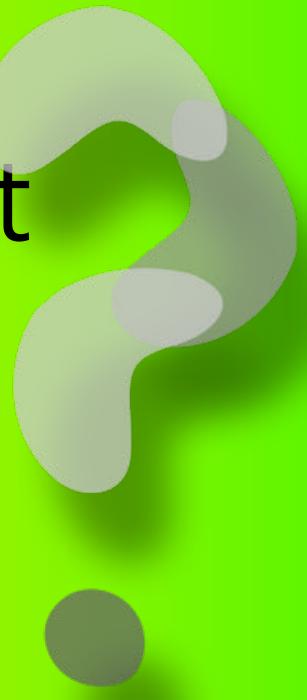
- * decoupled...
- * ...therefore better for decoupled architectures
- * contract management

tight

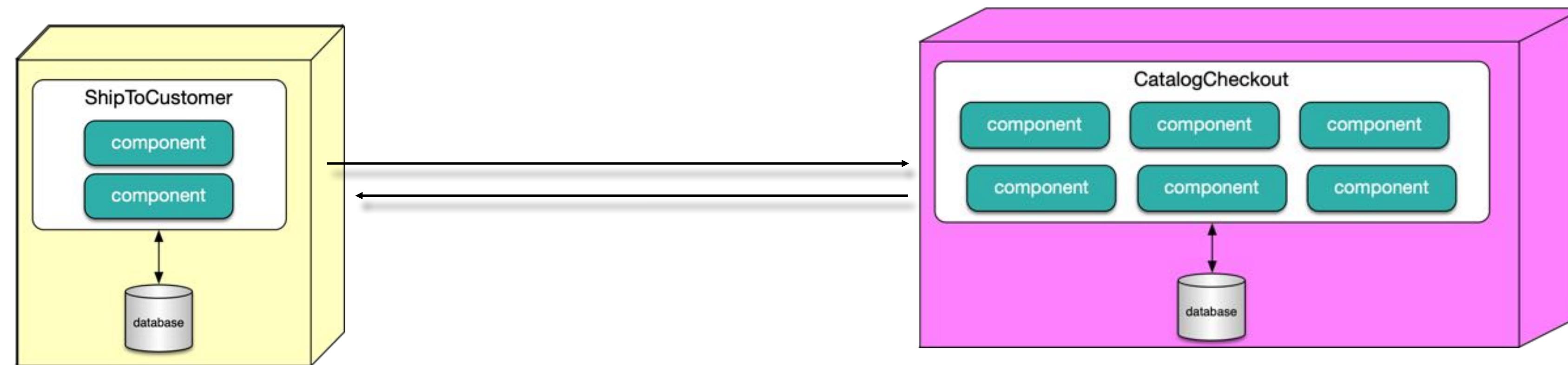
loose

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

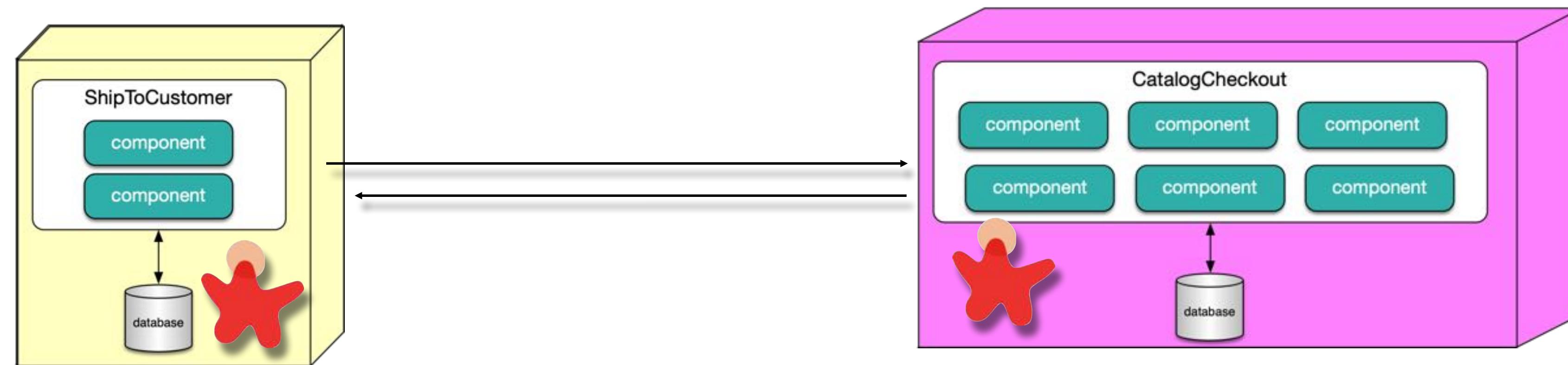
- * decoupled...
- * ...therefore better for decoupled architectures
- * contract management



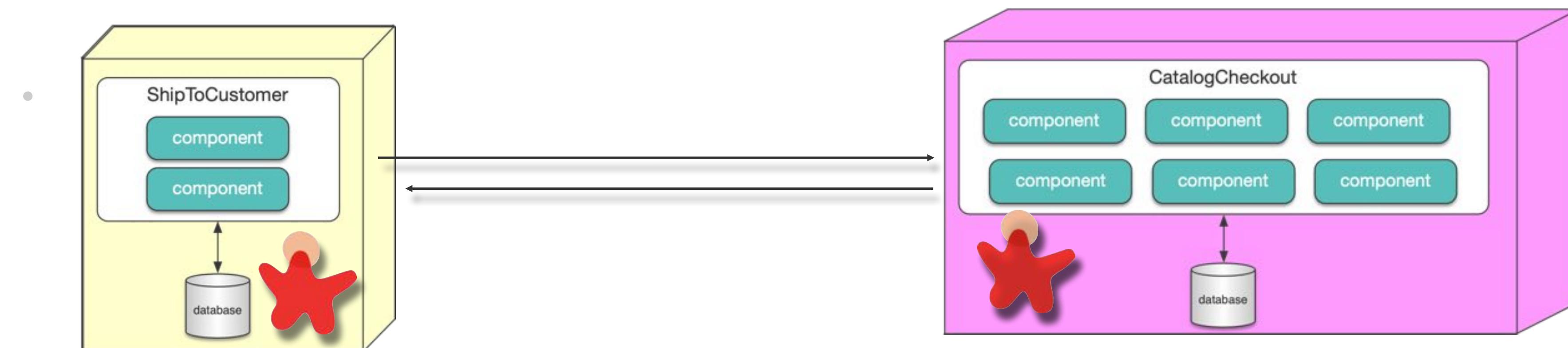
contracts in microservices



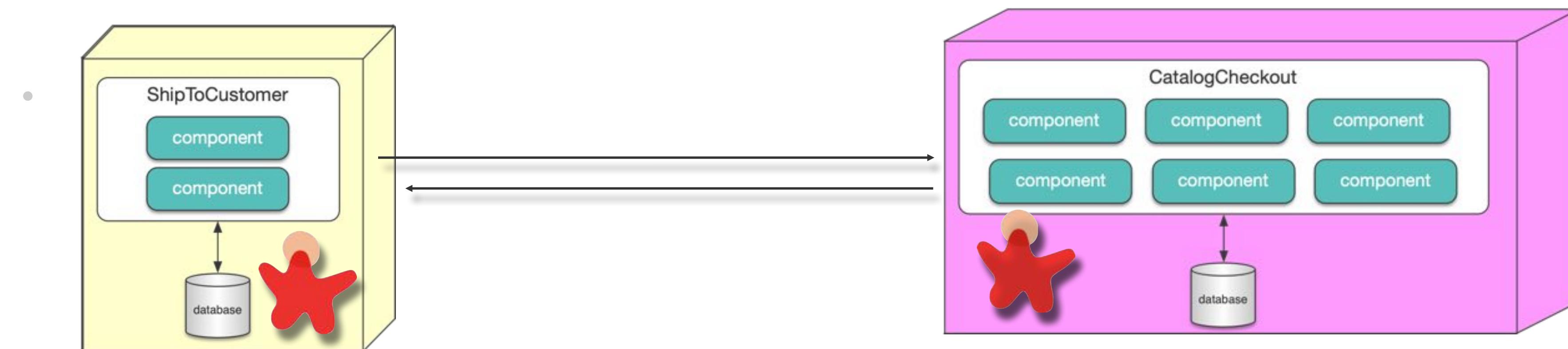
contracts in microservices



contracts in microservices

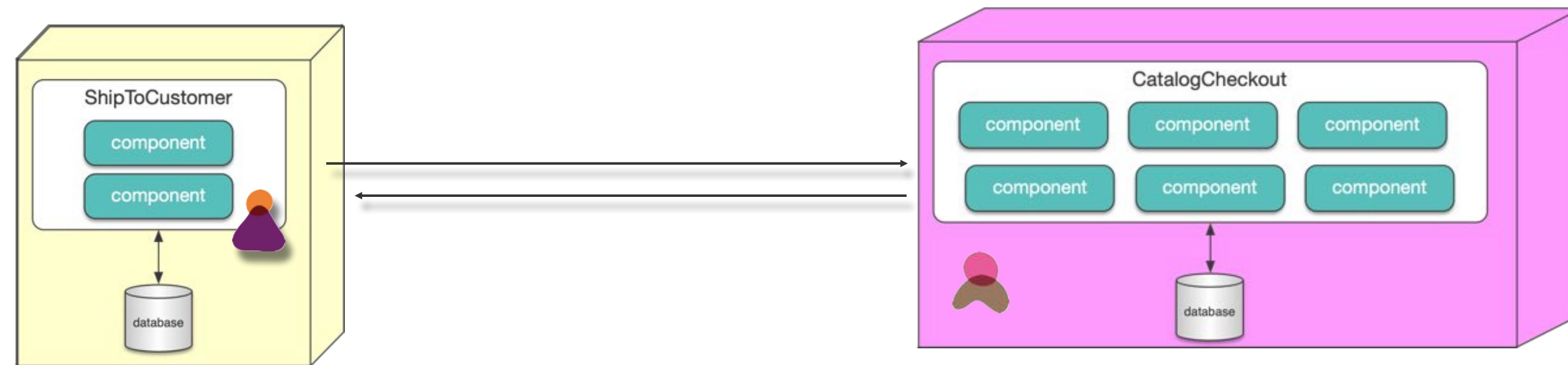


contracts in microservices

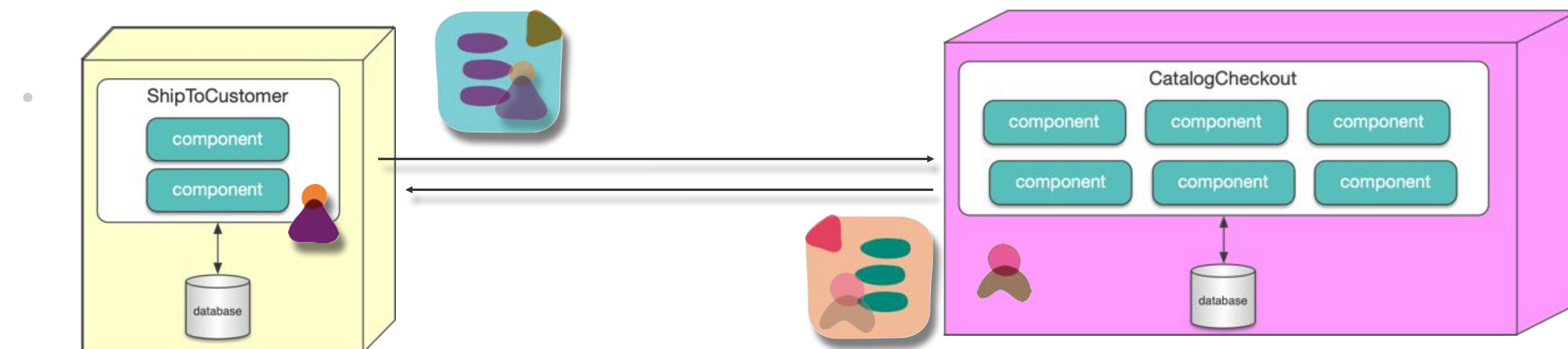


tight

contracts in microservices

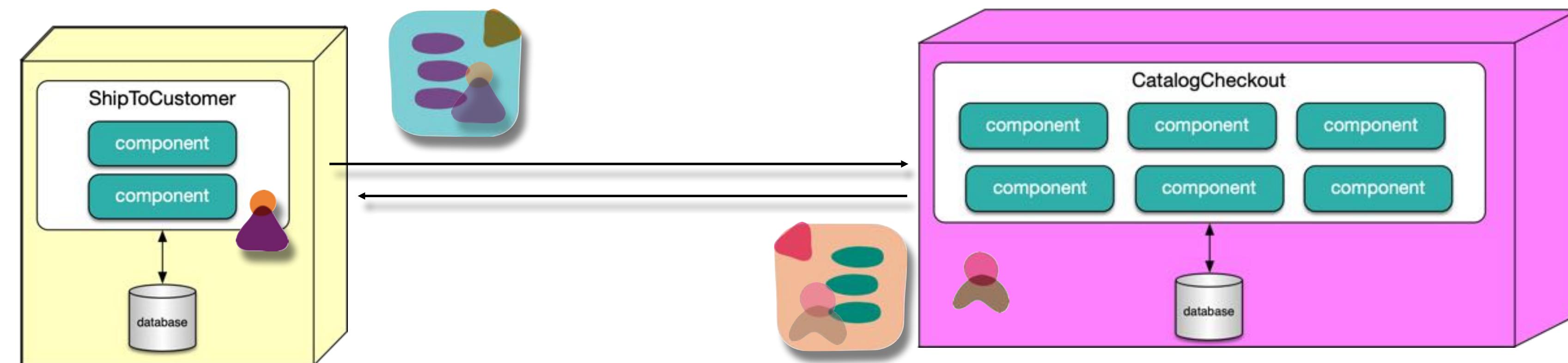


contracts in microservices



contracts in microservices

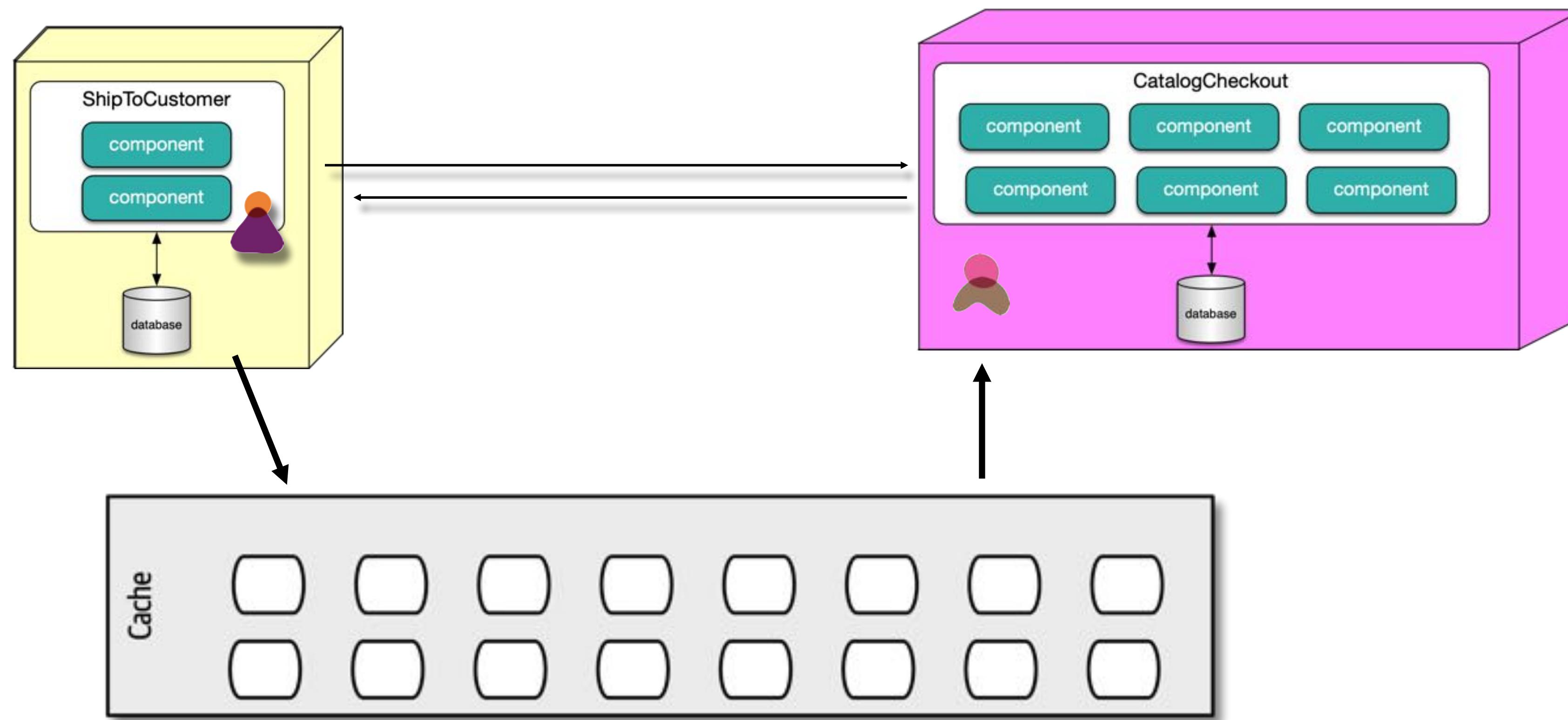
Transfer values, not types.



loose

contracts in microservices

Transfer values, not types.



connascence properties

Strength



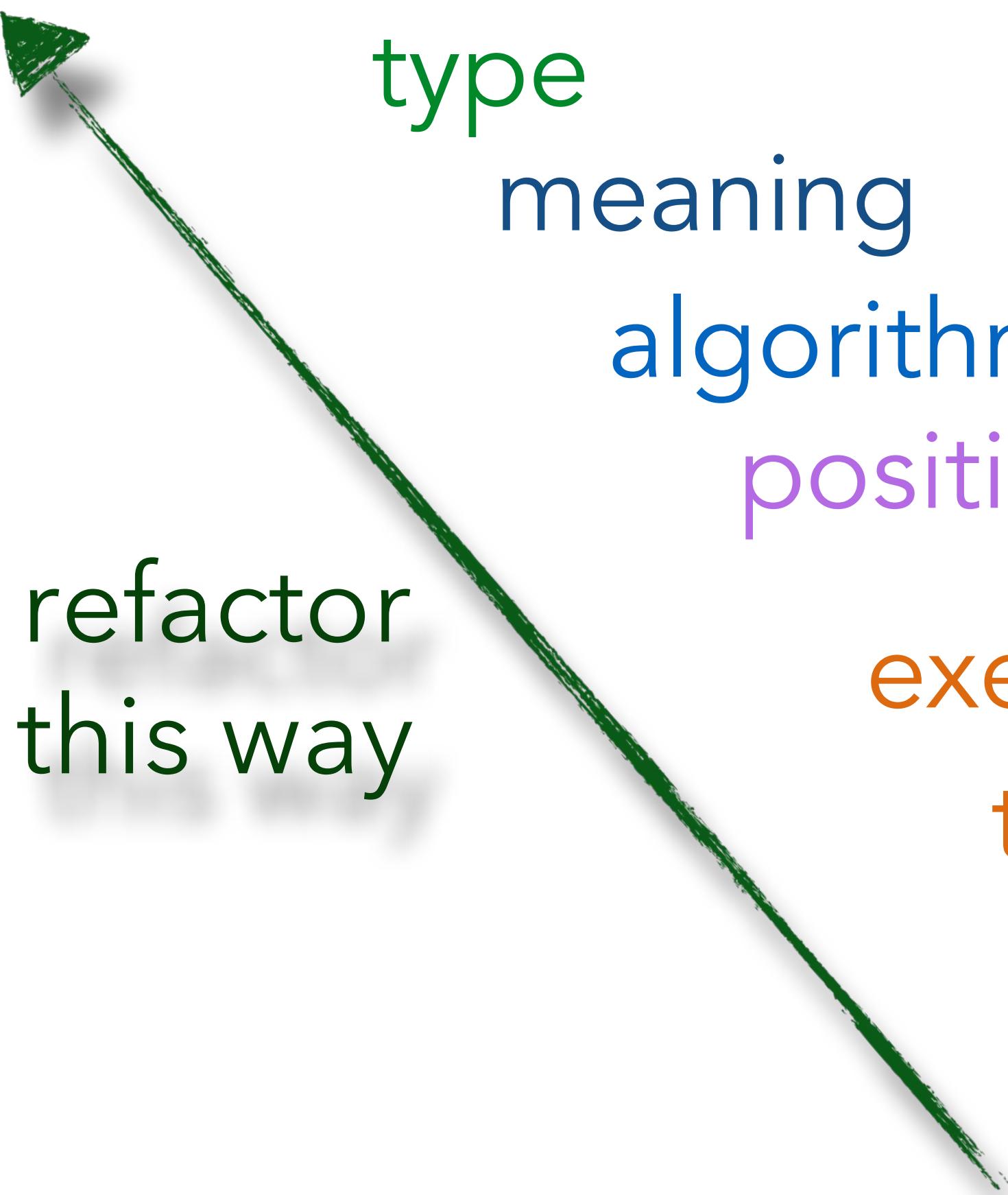
refactor
this way

name
type
meaning
algorithm
position
execution order
timing
value
identity



static

dynamic

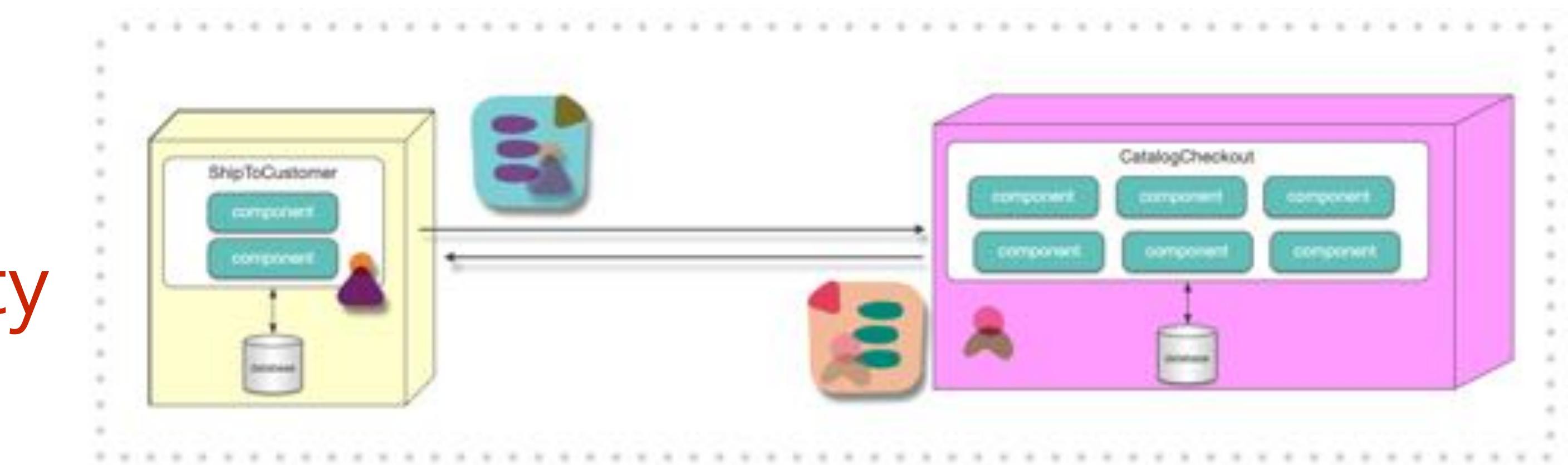
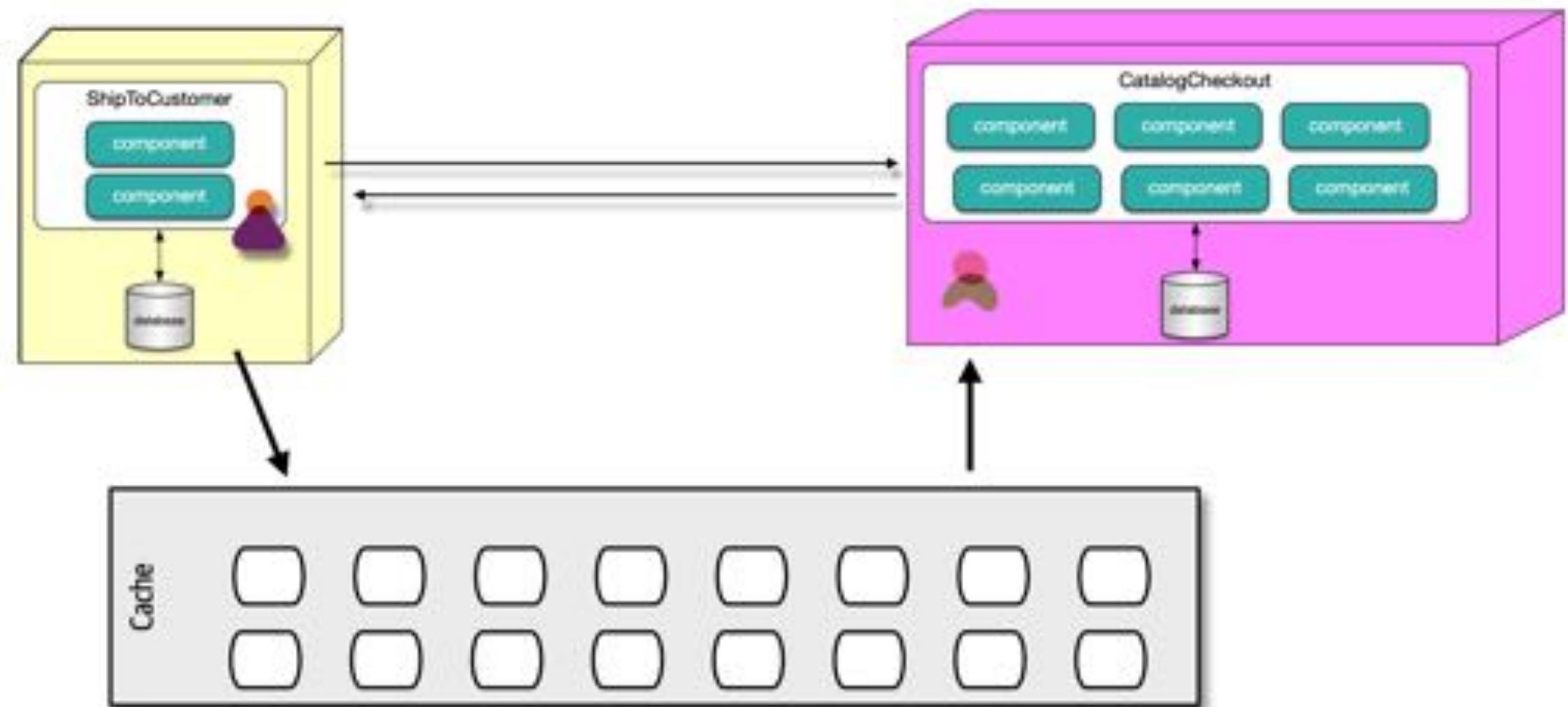
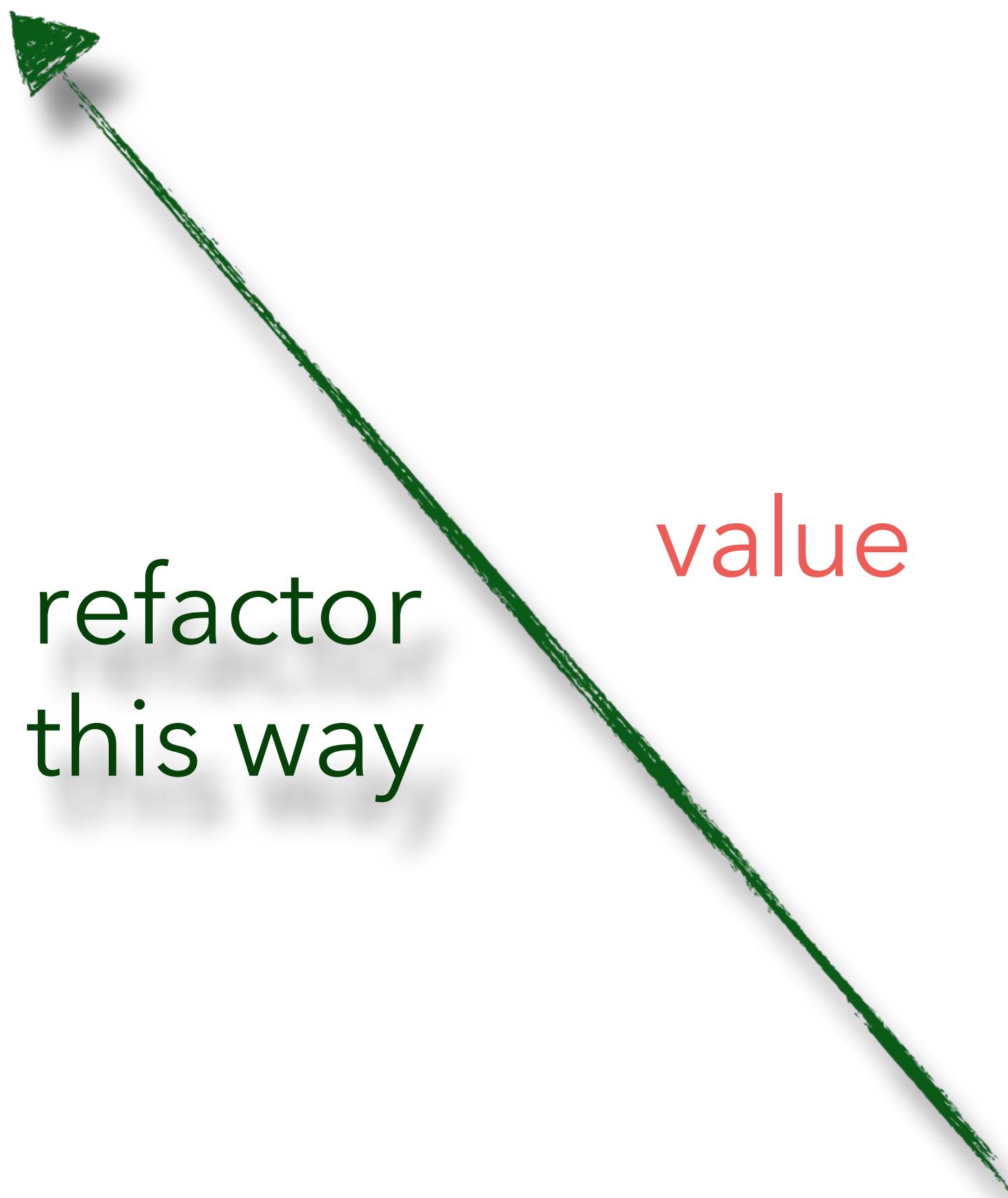


improving connascence

refactor
this way

value

identity



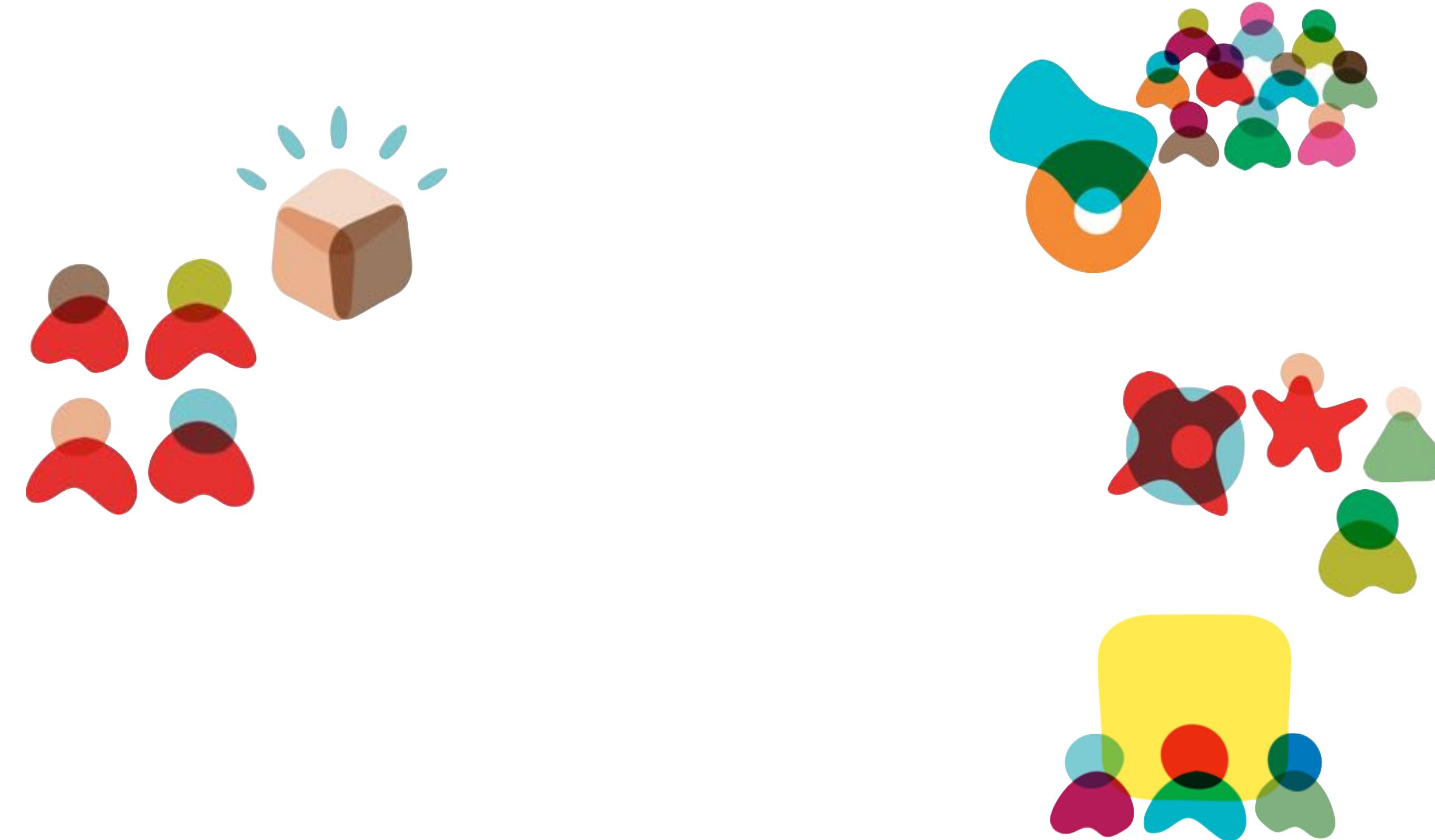
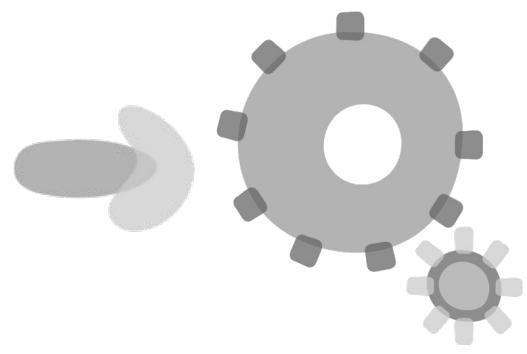
tradeoffs

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

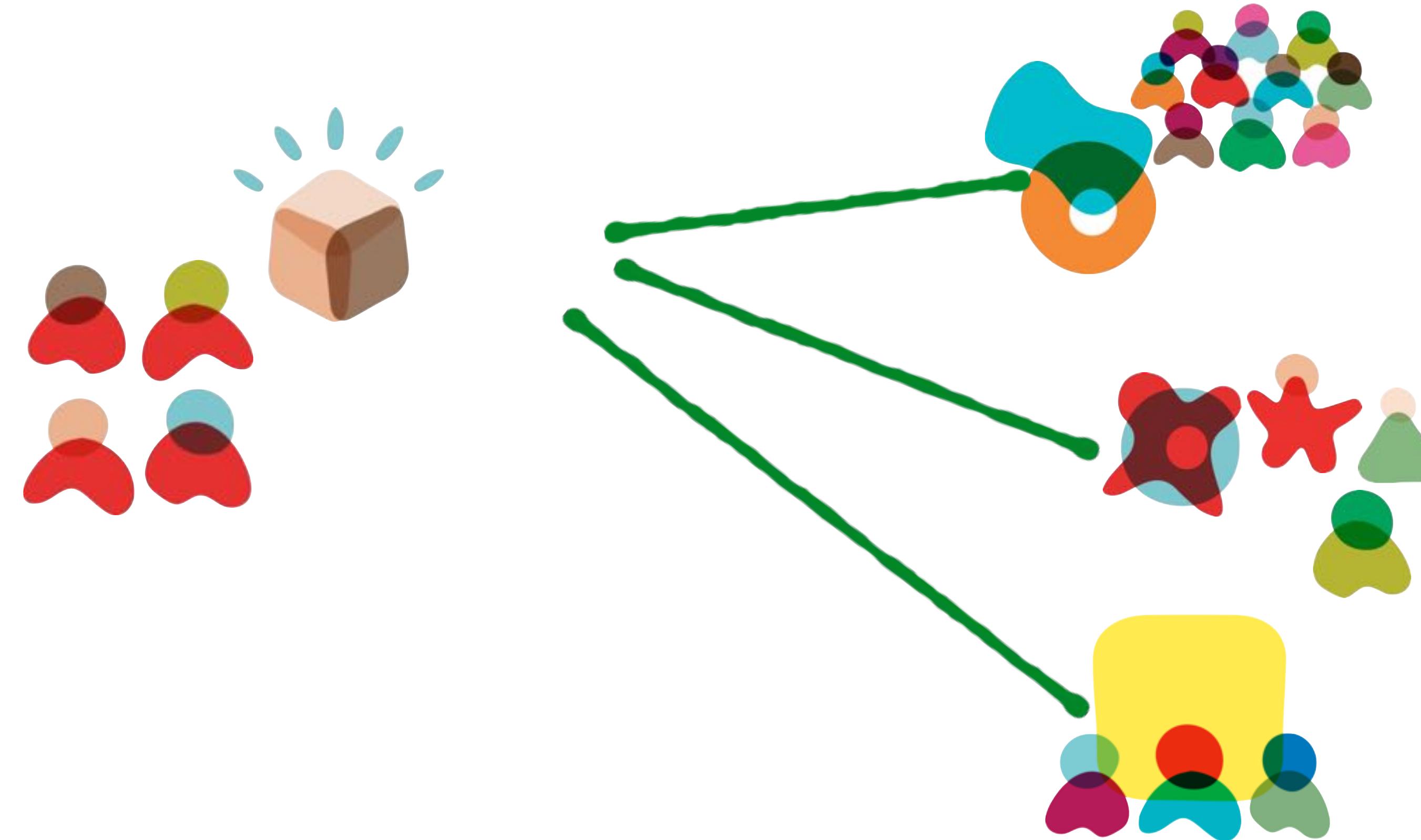
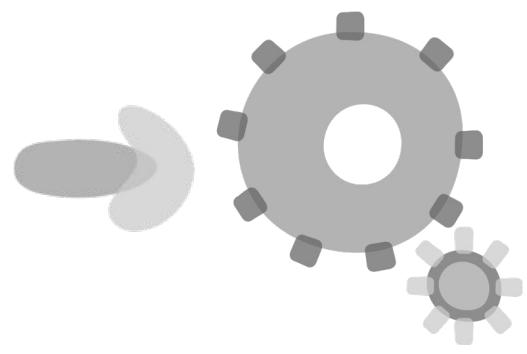


contract fitness function



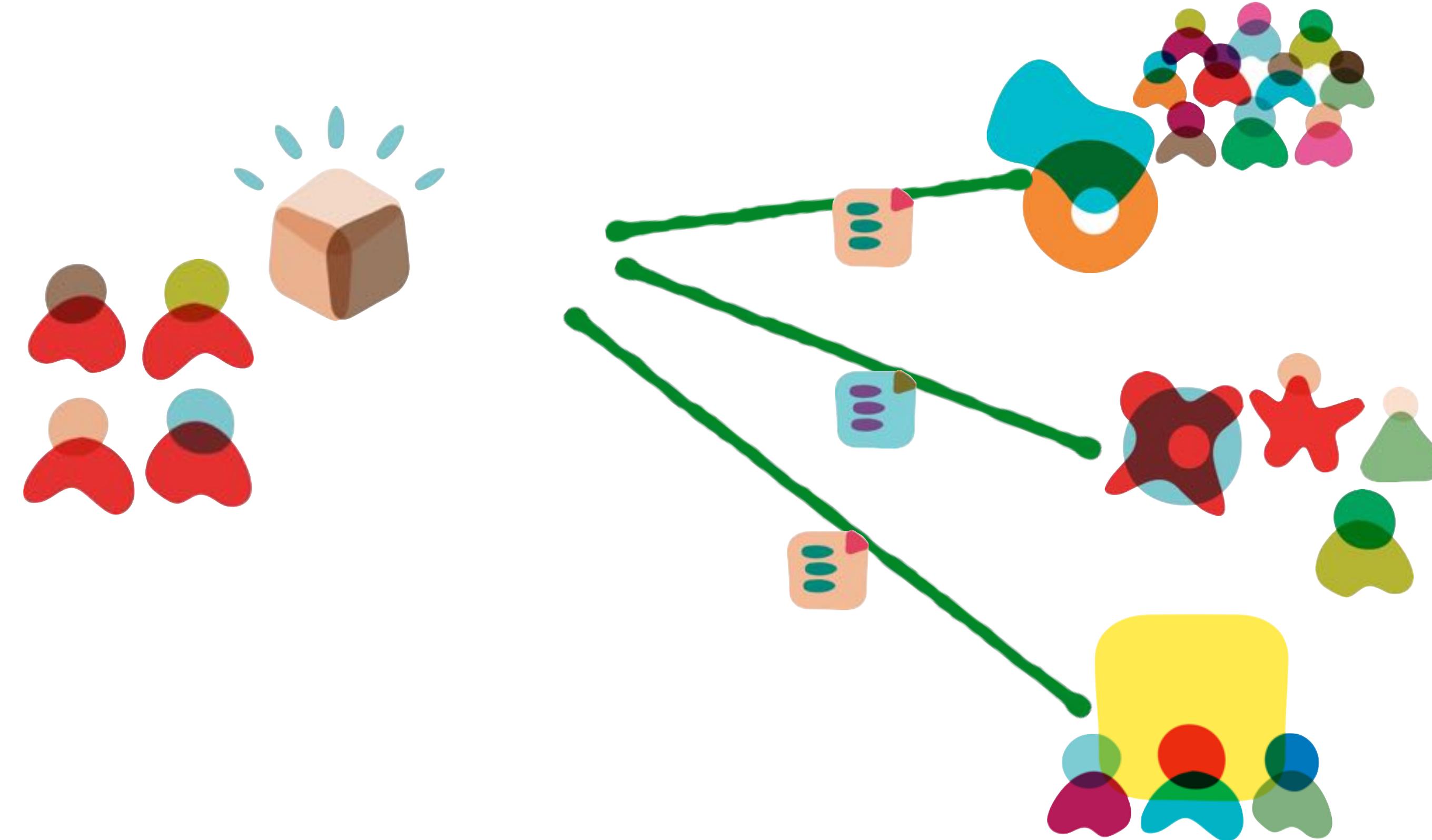
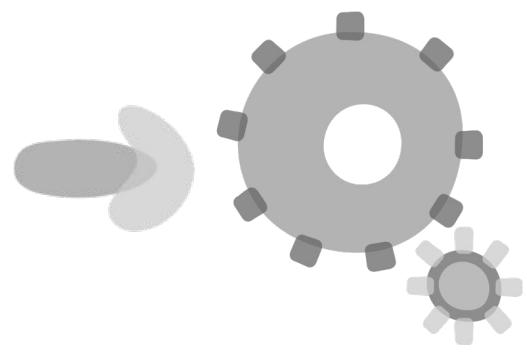
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function

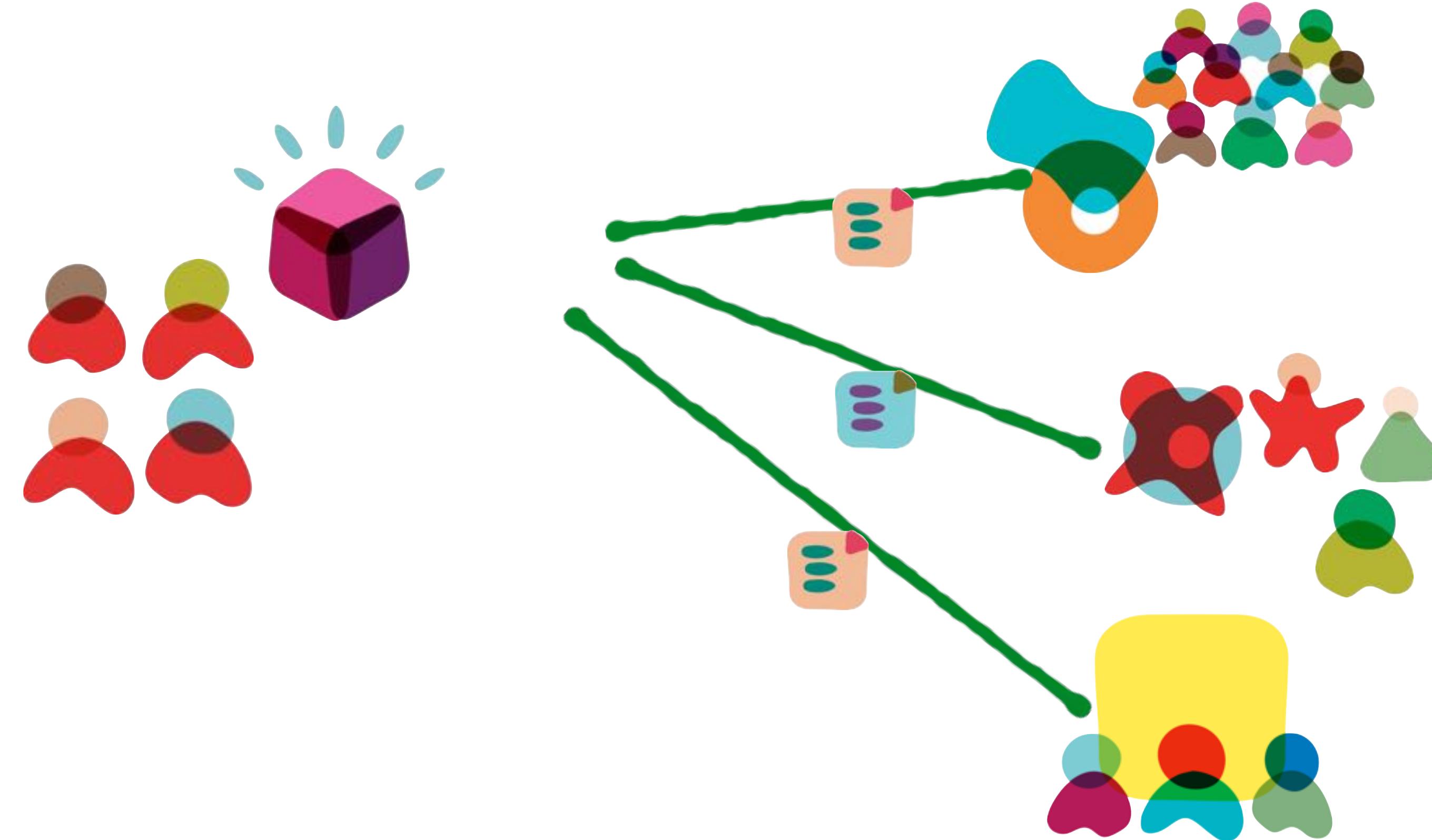
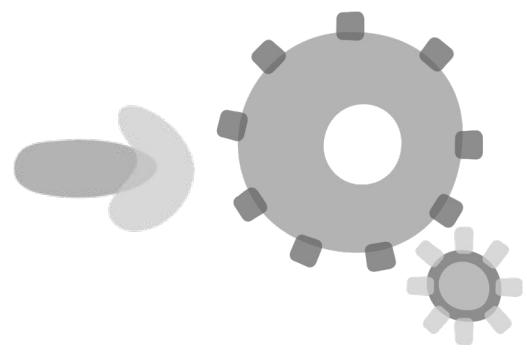


martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function

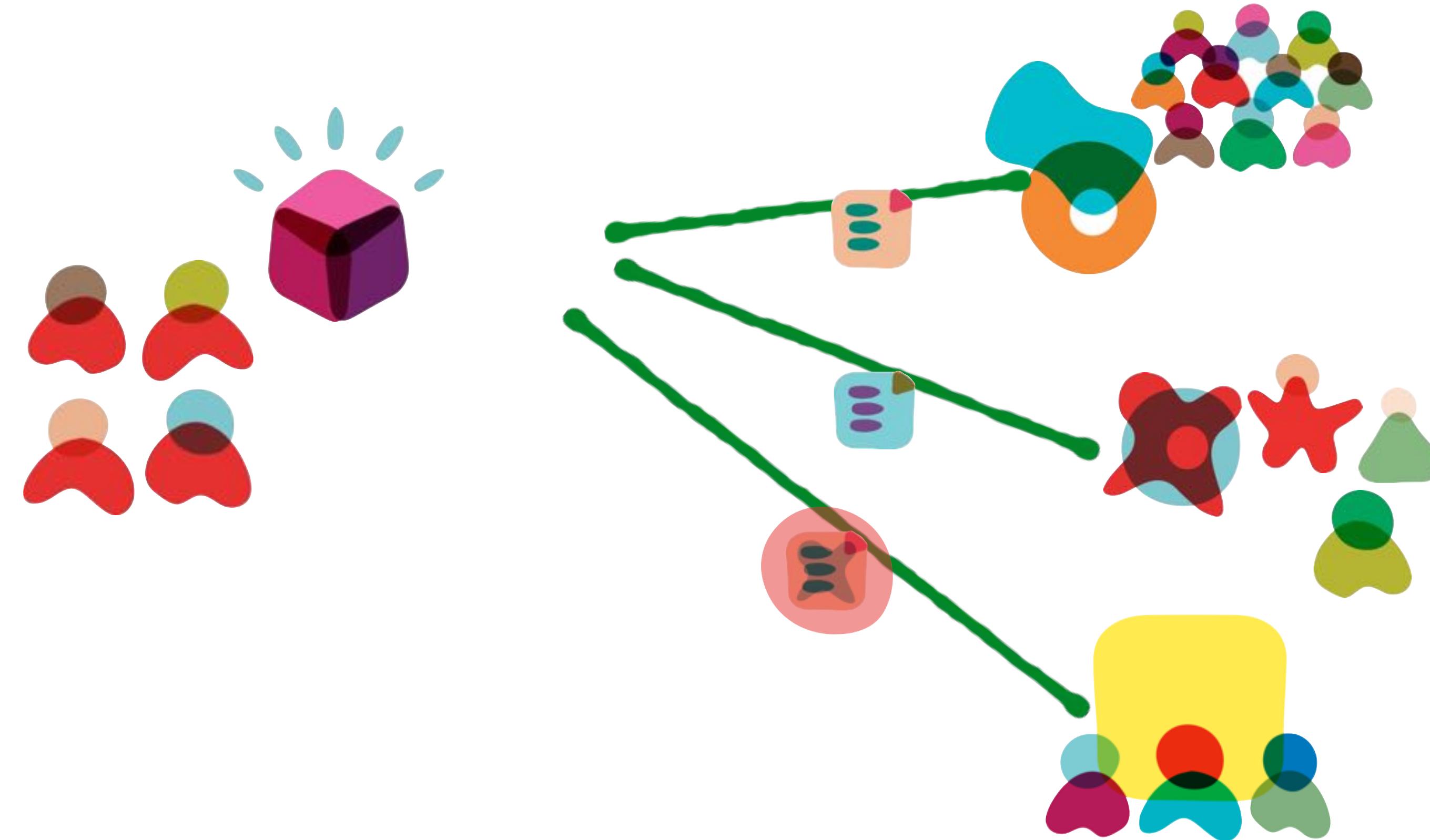
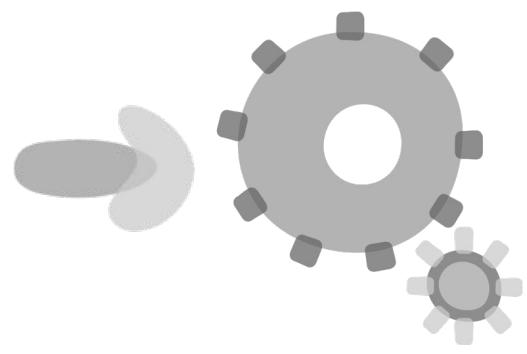


contract fitness function



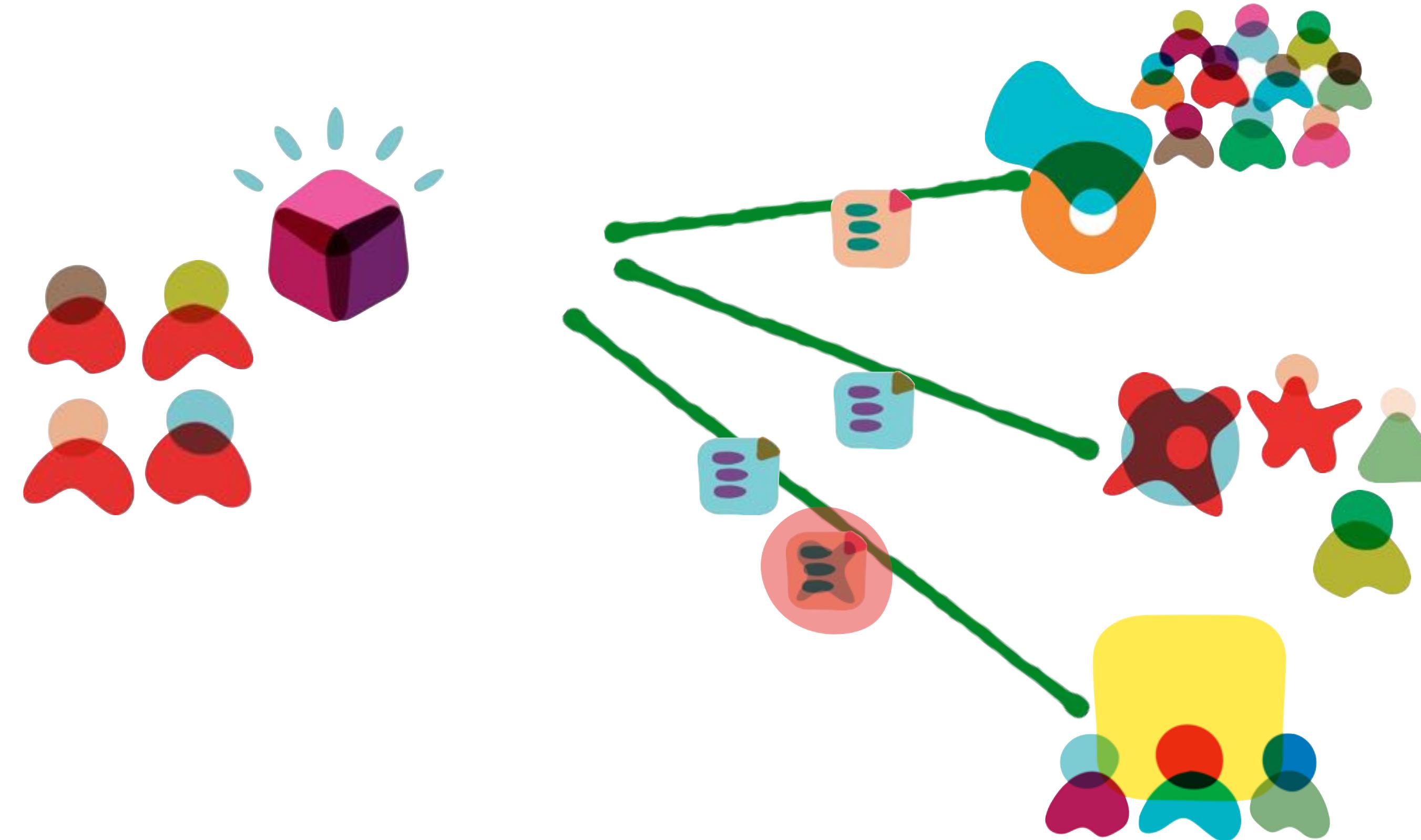
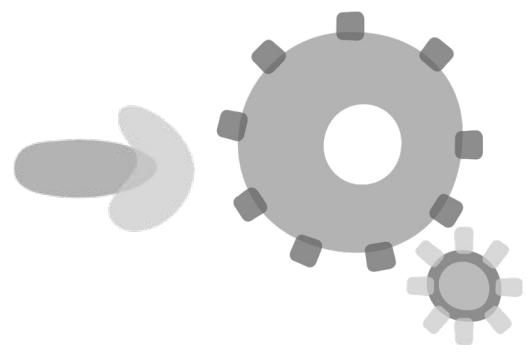
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



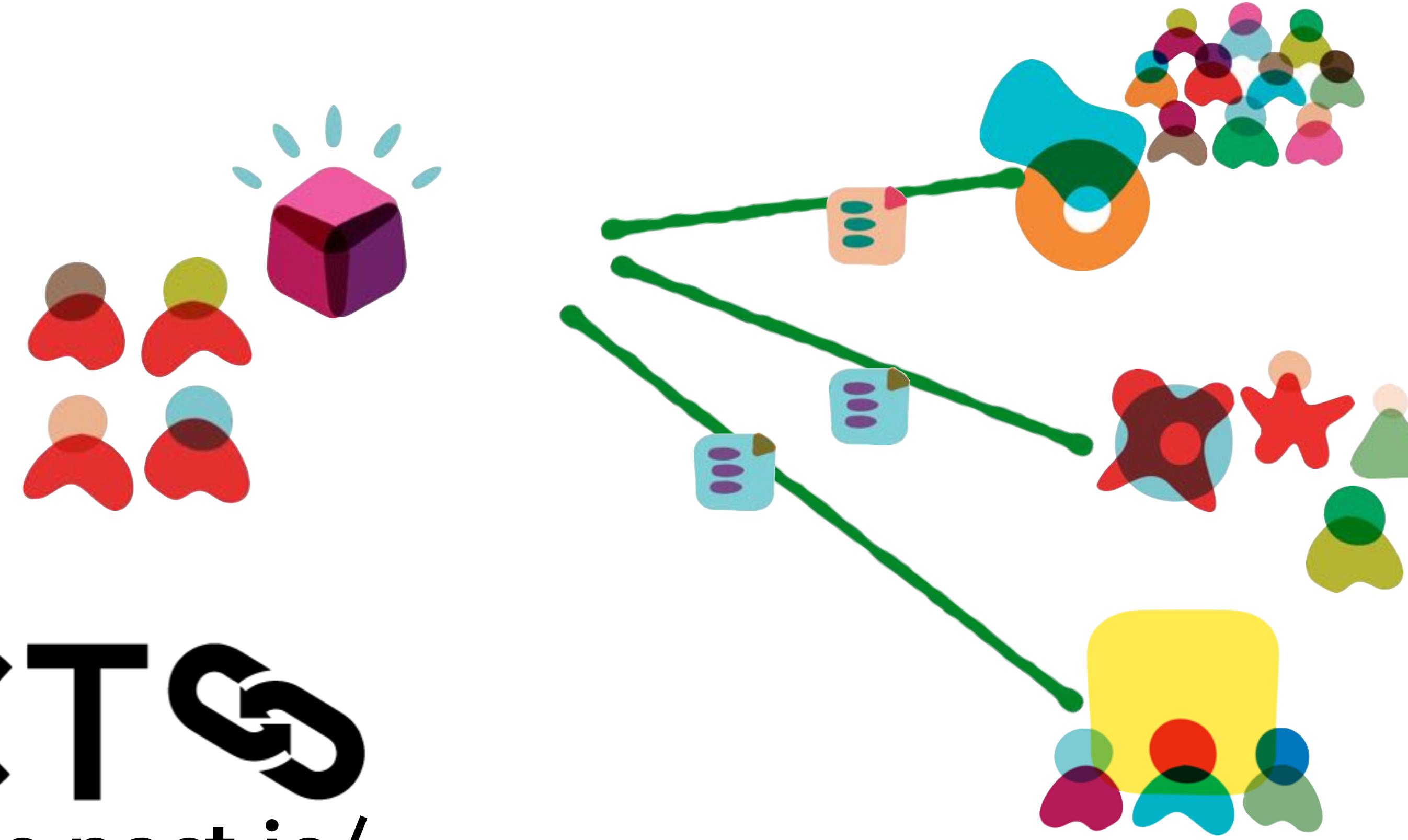
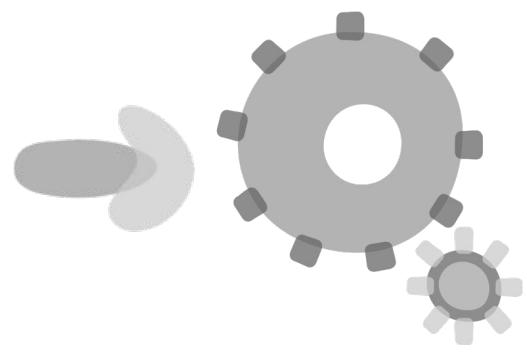
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



martinfowler.com/articles/consumerDrivenContracts.html

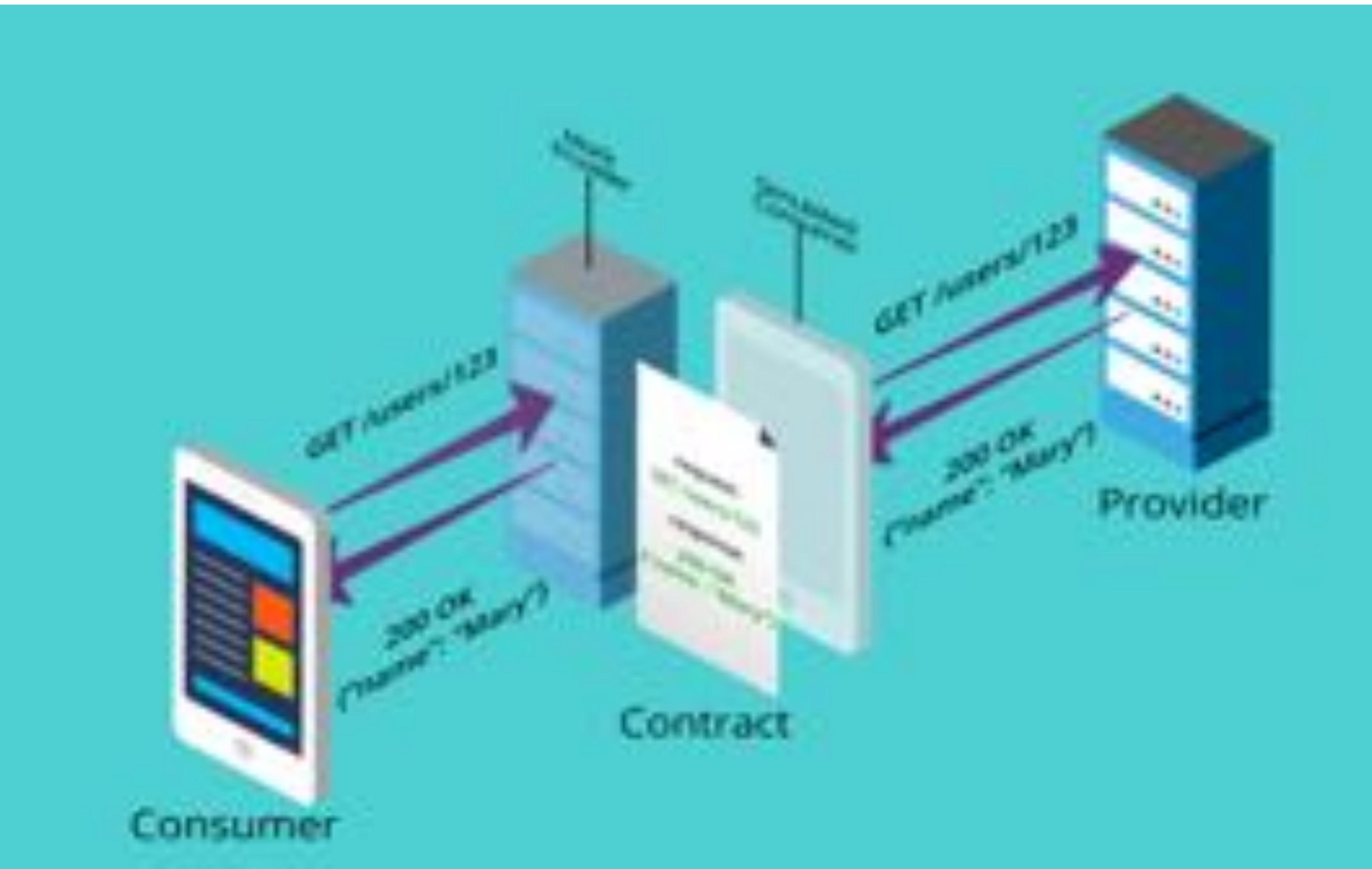
contract fitness function



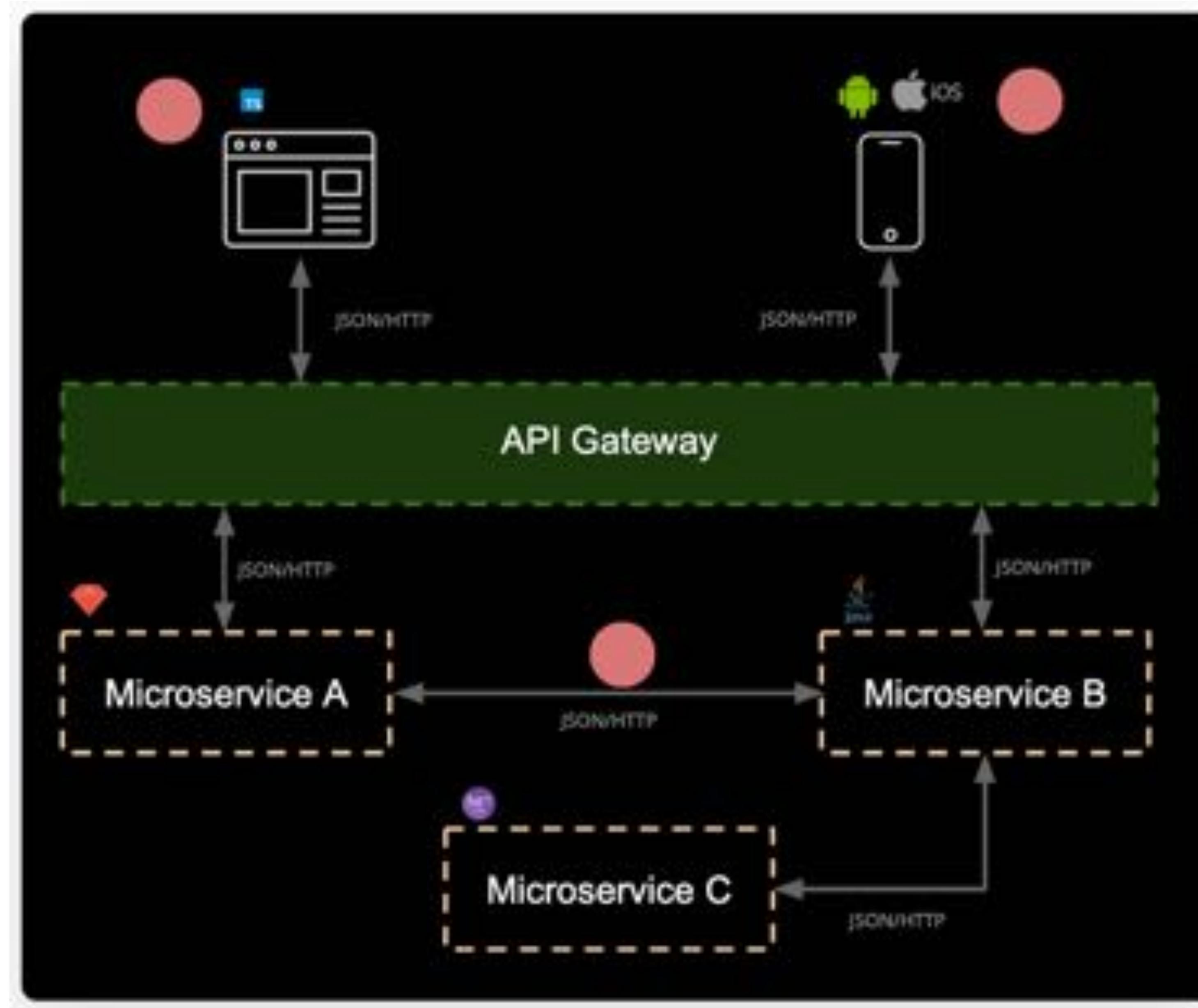
PACTS
<https://docs.pact.io/>

martinfowler.com/articles/consumerDrivenContracts.html

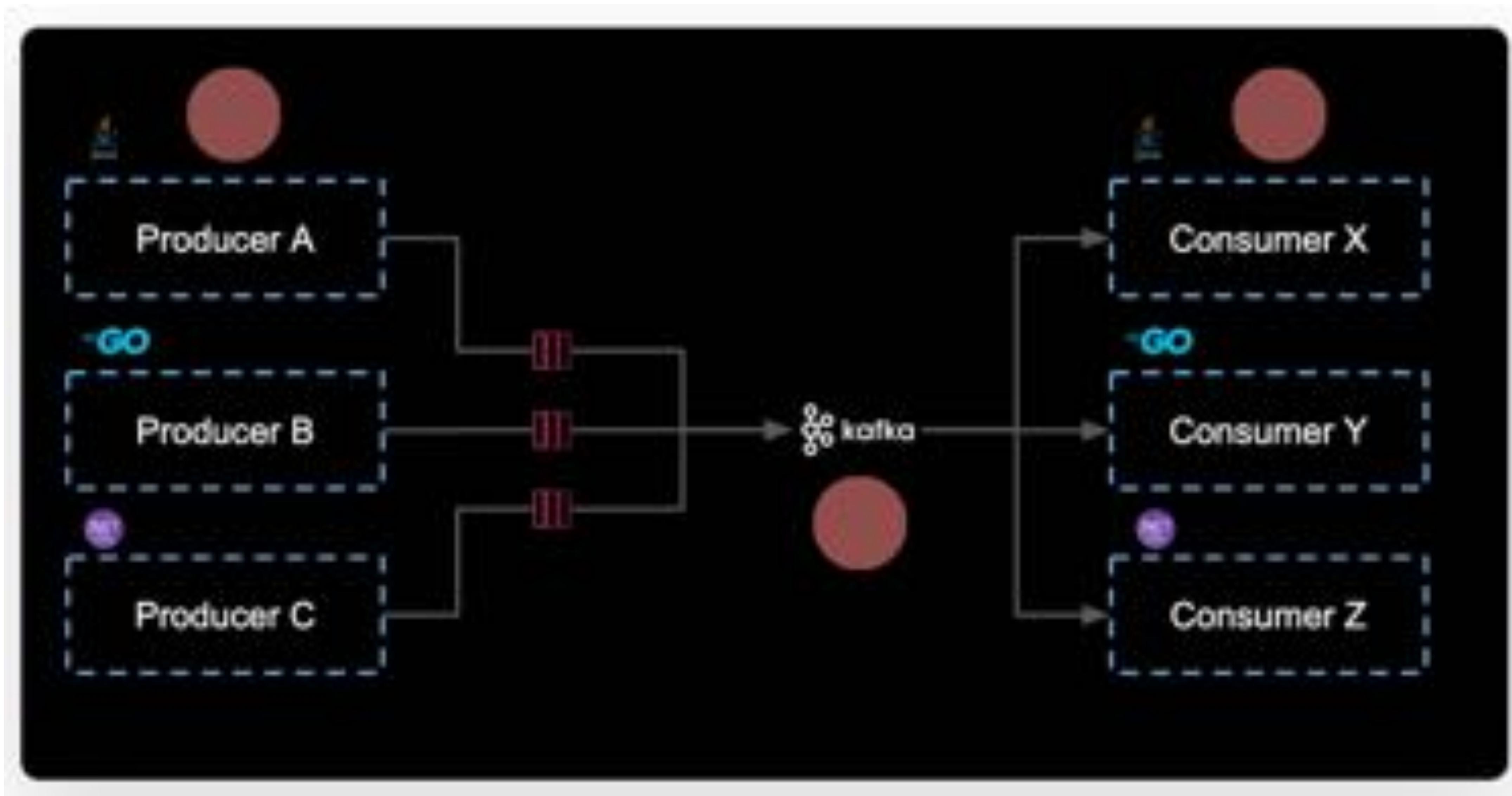
<https://docs.pact.io>



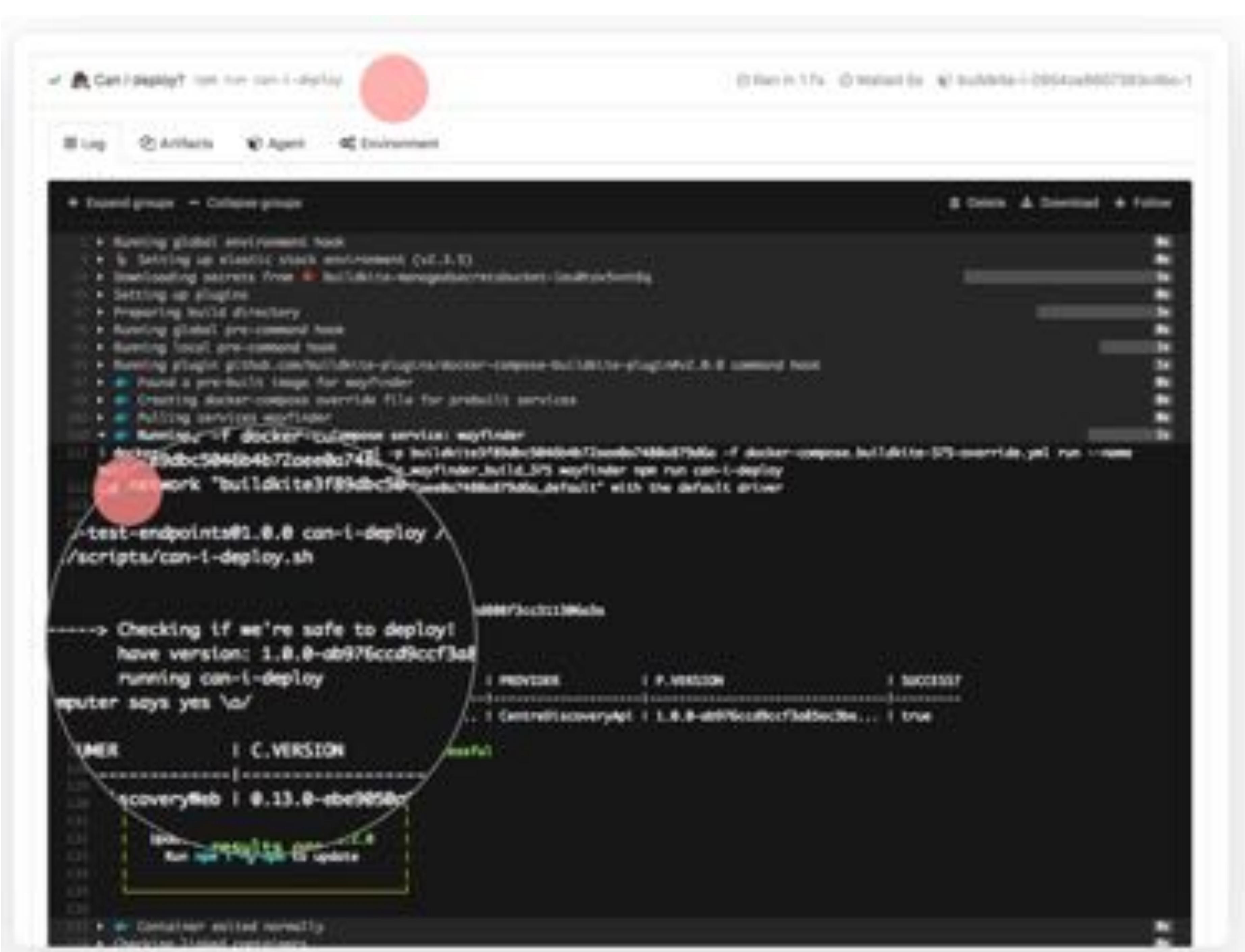
Pact + REST



Pact + EDA



D
C
I
+
Pact



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_thenReturn200WithProperHeaderAndBody() {  
  
    // 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
- requires versioning
- + guaranteed type fidelity

tradeoffs

value-based contracts

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

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

contracts | versions ?

the best version...

the best version...



the best version...



strive for versionless

tradeoffs

versionless

- harder to evolve behavior
- + follows natural evolution of web resources
- + never breaks backwards compatibility

versionless **OR** versions



since 2008



currently v. 20...
and incrementing

building versionless resources

- think from a consumer's standpoint
- (contract | fitness function) first design
- require tolerant readers
- add indirection
 - For example: do not return raw arrays in JSON

resource versioning

`http://api.example.com/v1/things/foo`

resource versioning

`http://api.example.com/v1/things/foo`

`http://api.example.com/v2/things/foo`

resource versioning

`http://api.example.com/v1/things/foo`

`http://api.example.com/v2/things/foo`

`http://api2.example.com/things/foo`

resource versioning

`http://api.example.com/v1/things/foo`

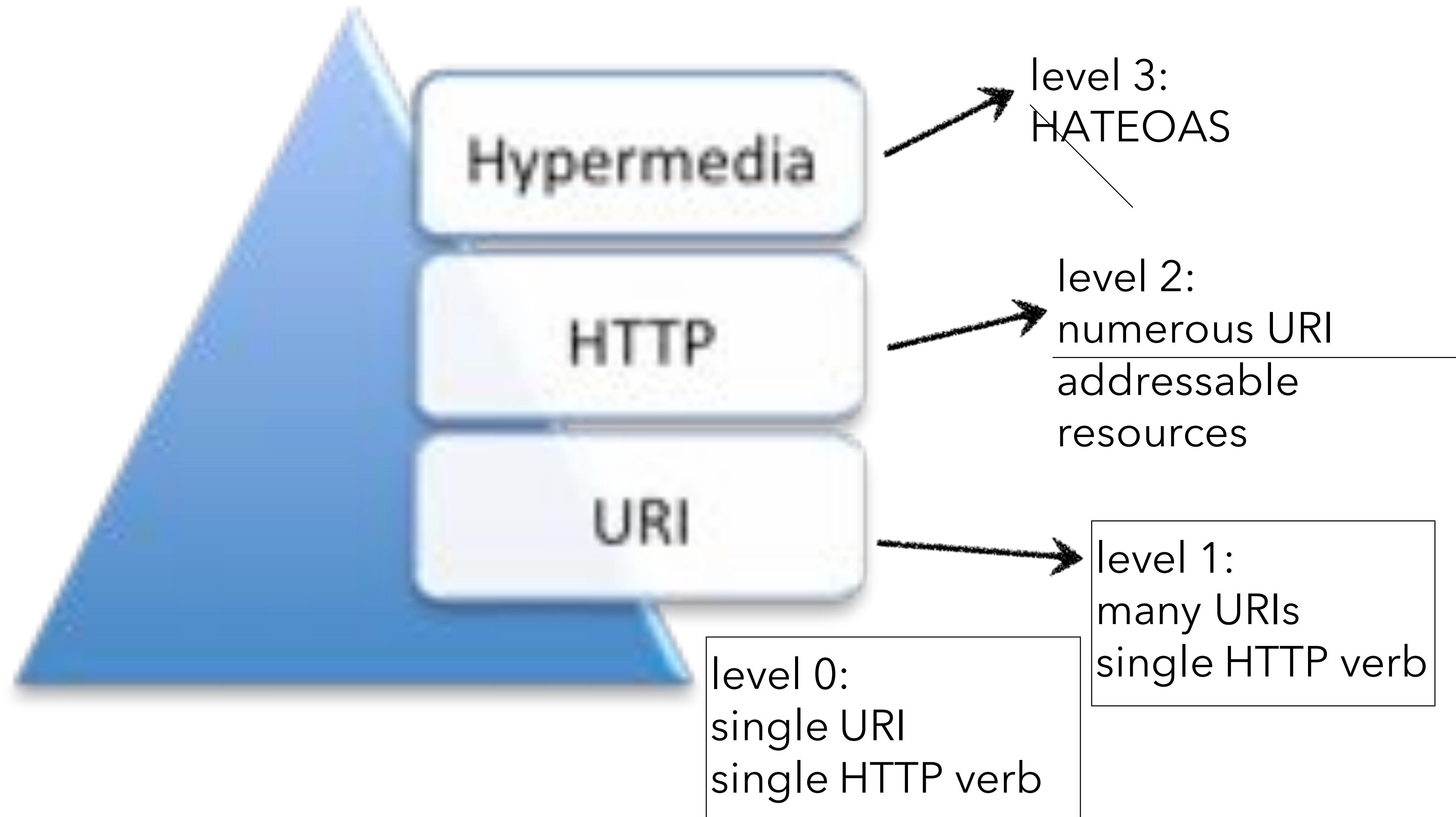
`http://api.example.com/v2/things/foo`

`http://api2.example.com/things/foo`

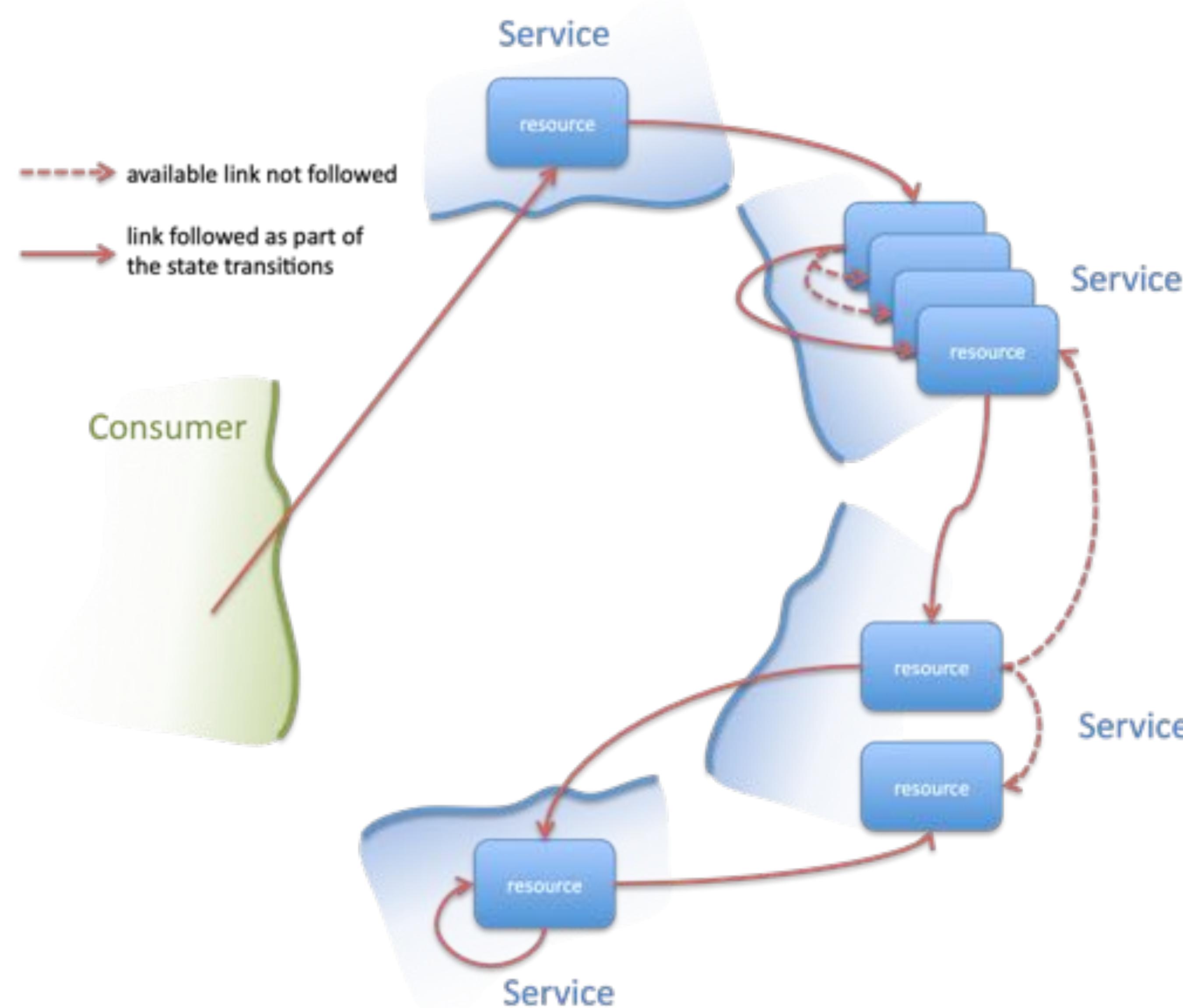
HATEAOS

-Hypermedia
-As
-The
-Engine
-Of
-Application
-State

Richardson's restful maturity



hypermedia tenet



HATEOAS example

```
GET /accounts/12345 HTTP/1.1
```

```
Host: bank.example.com
```

```
Accept: application/vnd.acme.account+json
```

HATEOAS example

```
GET /accounts/12345 HTTP/1.1
```

```
Host: bank.example.com
```

```
Accept: application/vnd.acme.account+json
```

HTTP/1.1 200 OK

Content-Type: application/vnd.acme.account+json

Content-Length: ...

```
{  
    "account": {  
        "account_number": 12345,  
        "balance": {  
            "currency": "usd",  
            "value": 100.00  
        },  
        "links": {  
            "deposit": "/accounts/12345/deposit",  
            "withdraw": "/accounts/12345/withdraw",  
            "transfer": "/accounts/12345/transfer",  
            "close": "/accounts/12345/close"  
        }  
    }  
}
```

HATEOAS example

```
GET /accounts/12345 HTTP/1.1  
Host: bank.example.com  
Accept: application/vnd.acme.account+json
```

```
HTTP/1.1 200 OK  
Content-Type: application/vnd.acme.account+json  
Content-Length: ...  
  
{  
  "account": {  
    "account_number": 12345,  
    "balance": {  
      "currency": "usd",  
      "value": 100.00  
    },  
    "links": {  
      "deposit": "/accounts/12345/deposit",  
      "withdraw": "/accounts/12345/withdraw",  
      "transfer": "/accounts/12345/transfer",  
      "close": "/accounts/12345/close"  
    }  
  }  
}
```

hypermedia (endpoint + media type) for
next steps in domain process

resource versioning

`http://api.example.com/v1/things/foo`

`http://api.example.com/v2/things/foo`

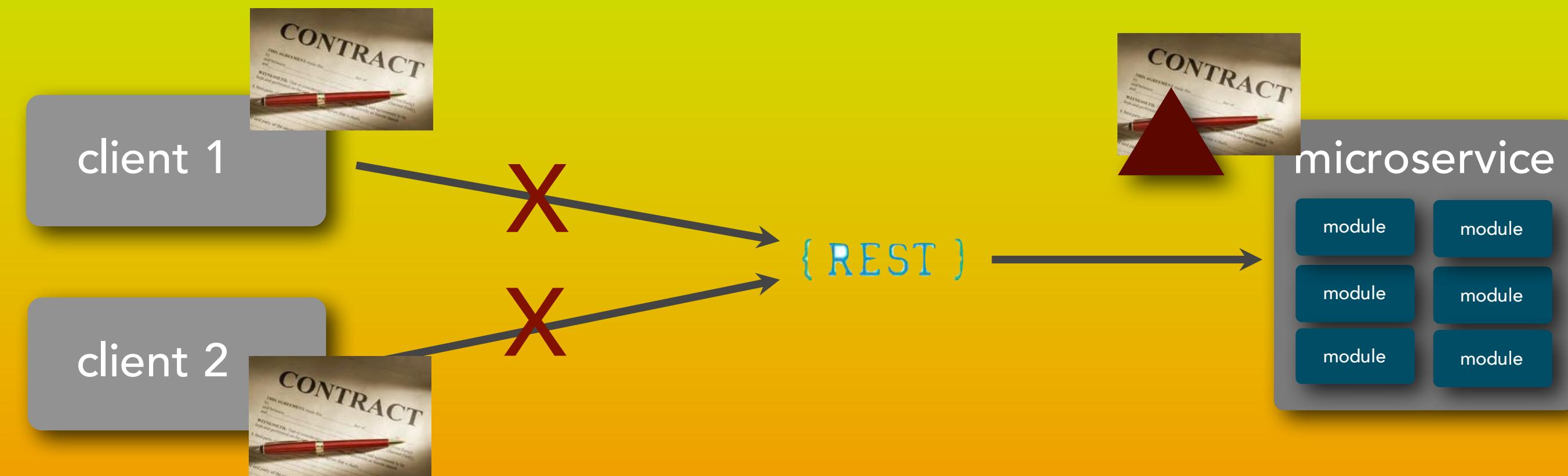
`http://api2.example.com/things/foo`

```
HTTP/1.1 200 OK
Content-Type: application/vnd.acme.account+json
Content-Length: ...

{
    "account": {
        "account_number": 12345,
        "balance": {
            "currency": "usd",
            "value": 100.00
        },
        "links": {
            "deposit": "/accounts/12345/deposit",
            "withdraw": "/accounts/12345/withdraw",
            "transfer": "/accounts/12345/transfer",
            "close": "/accounts/12345/close"
        }
    }
}
```

tradeoffs

contract creation, maintenance, versioning,
and coordination



<https://www.developertoarchitect.com/lessons/lesson41.html>

choreography | orchestration |
synchronization | write-ahead log
| sagas ?

orchestration and workflow

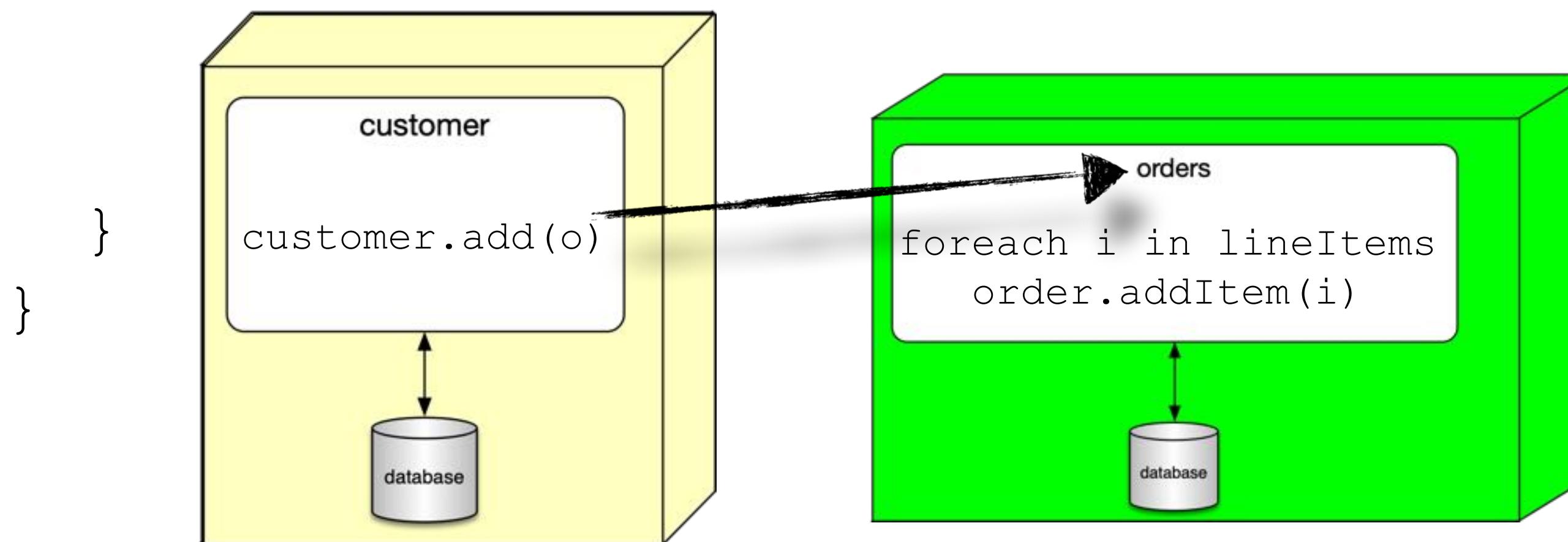


The Origin of Orchestration

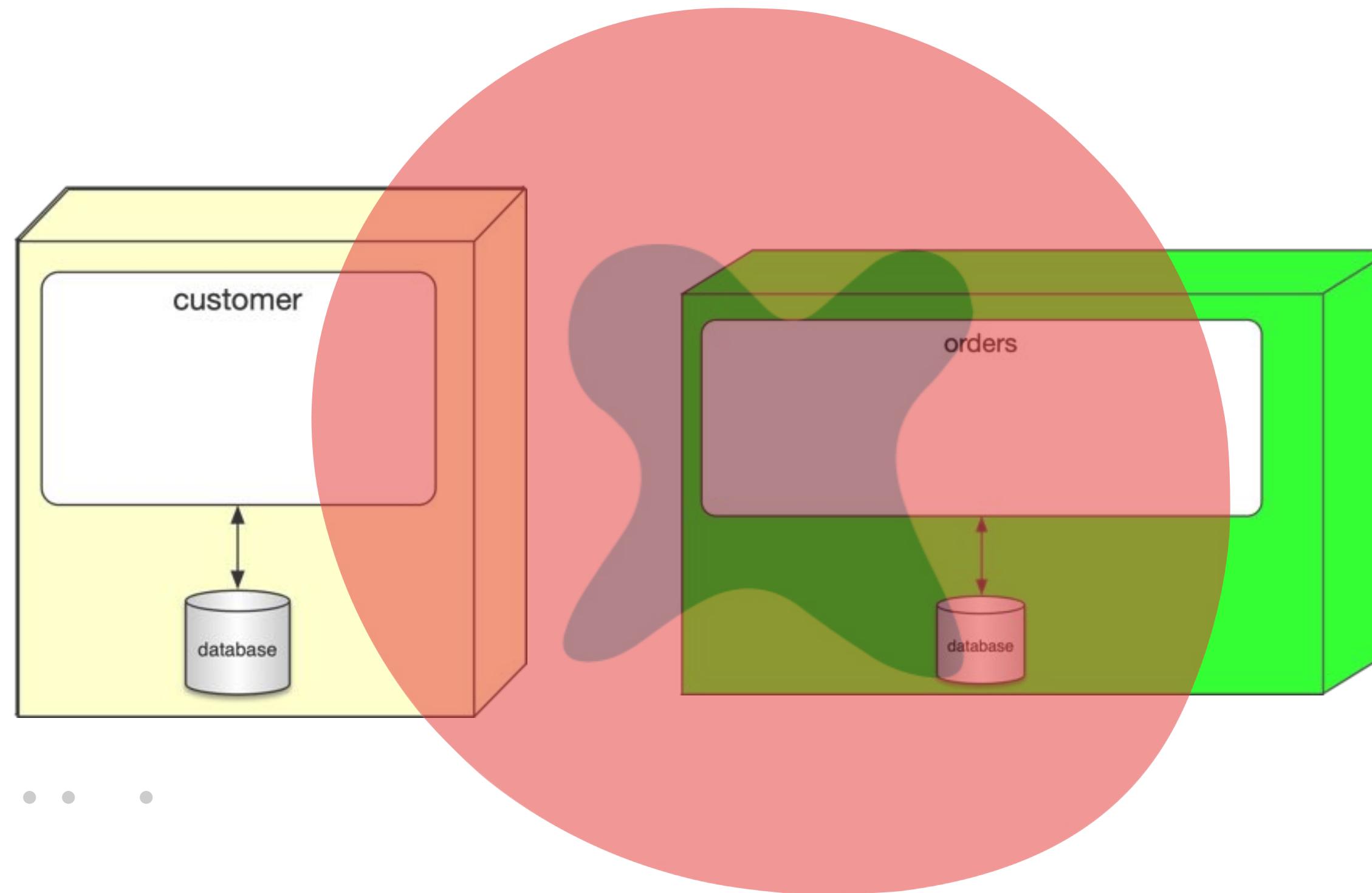
```
placeOrder(customer, order, lineItems) {  
    startTransaction {  
        foreach i in lineItems  
            order.addItem(i)  
        customer.add(o)  
    }  
}
```

The Origin of Orchestration

```
placeOrder(customer, order, lineItems) {  
    startTransaction {
```

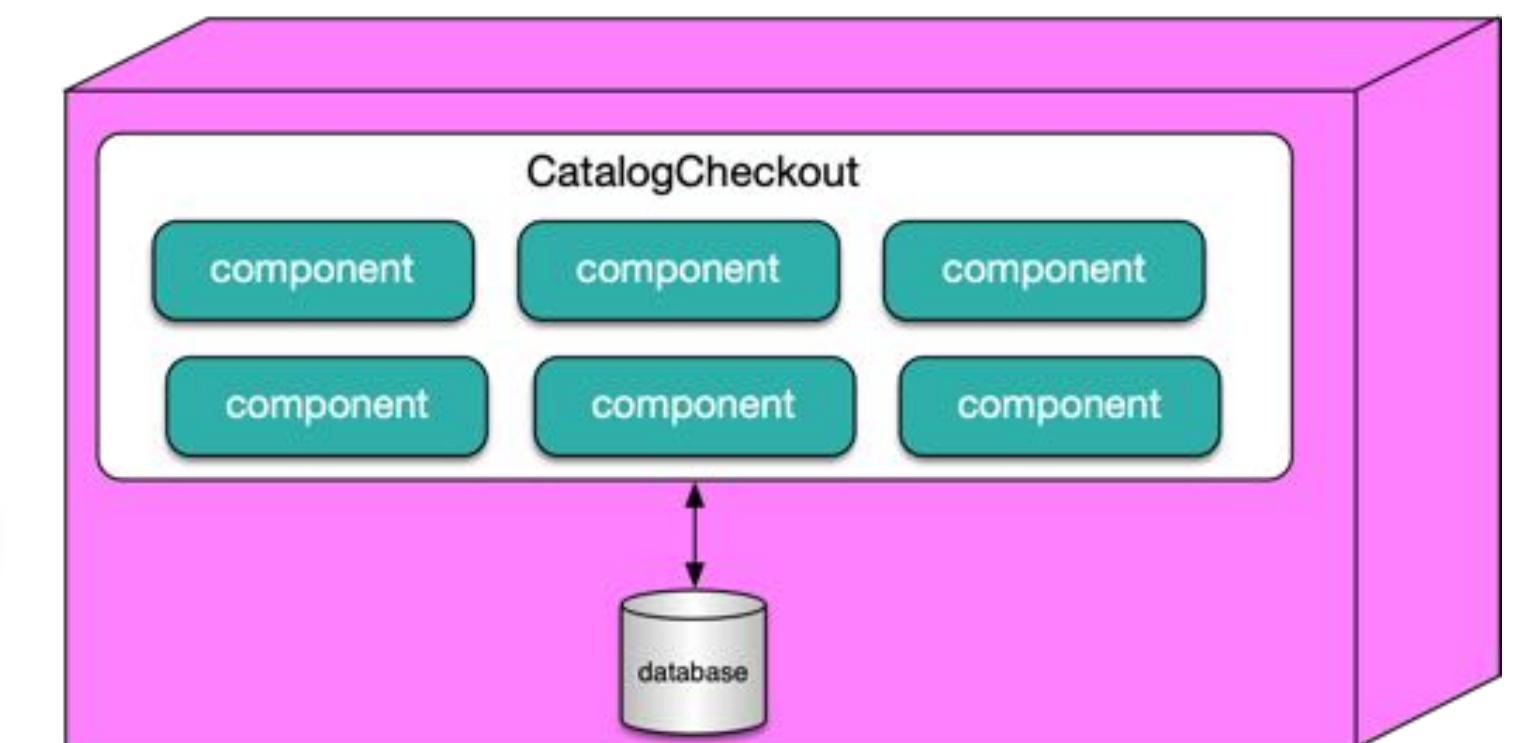


Orchestration in Microservices

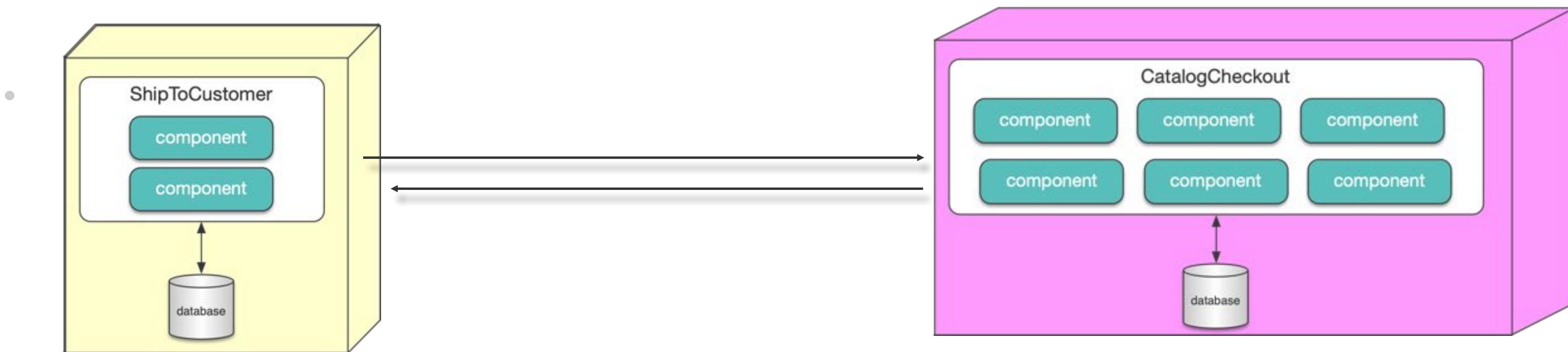


The best orchestration is
no orchestration.

Fix your granularity instead.

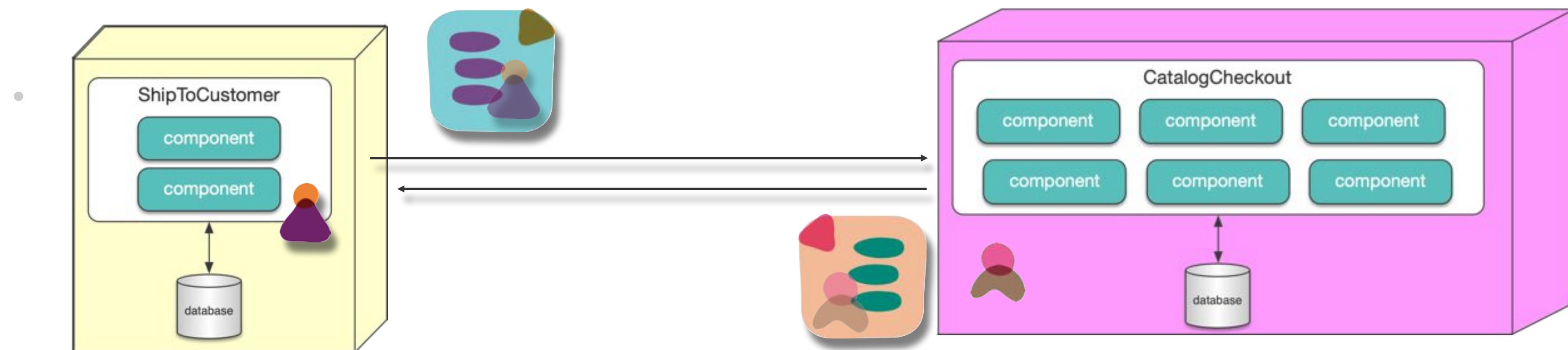


Orchestration in Microservices



Synchronous calls expand the scope of quanta.

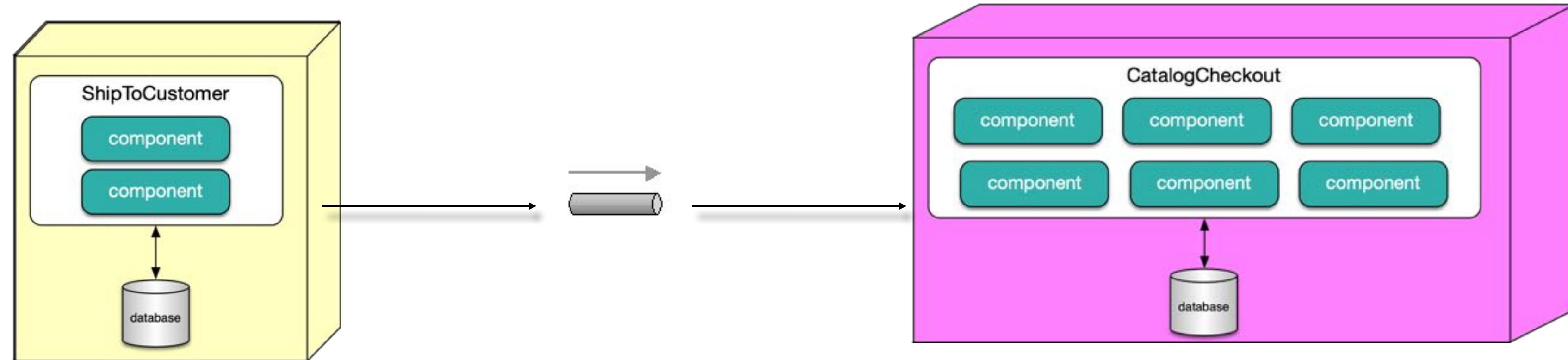
Orchestration in Microservices



Transfer values, not types.

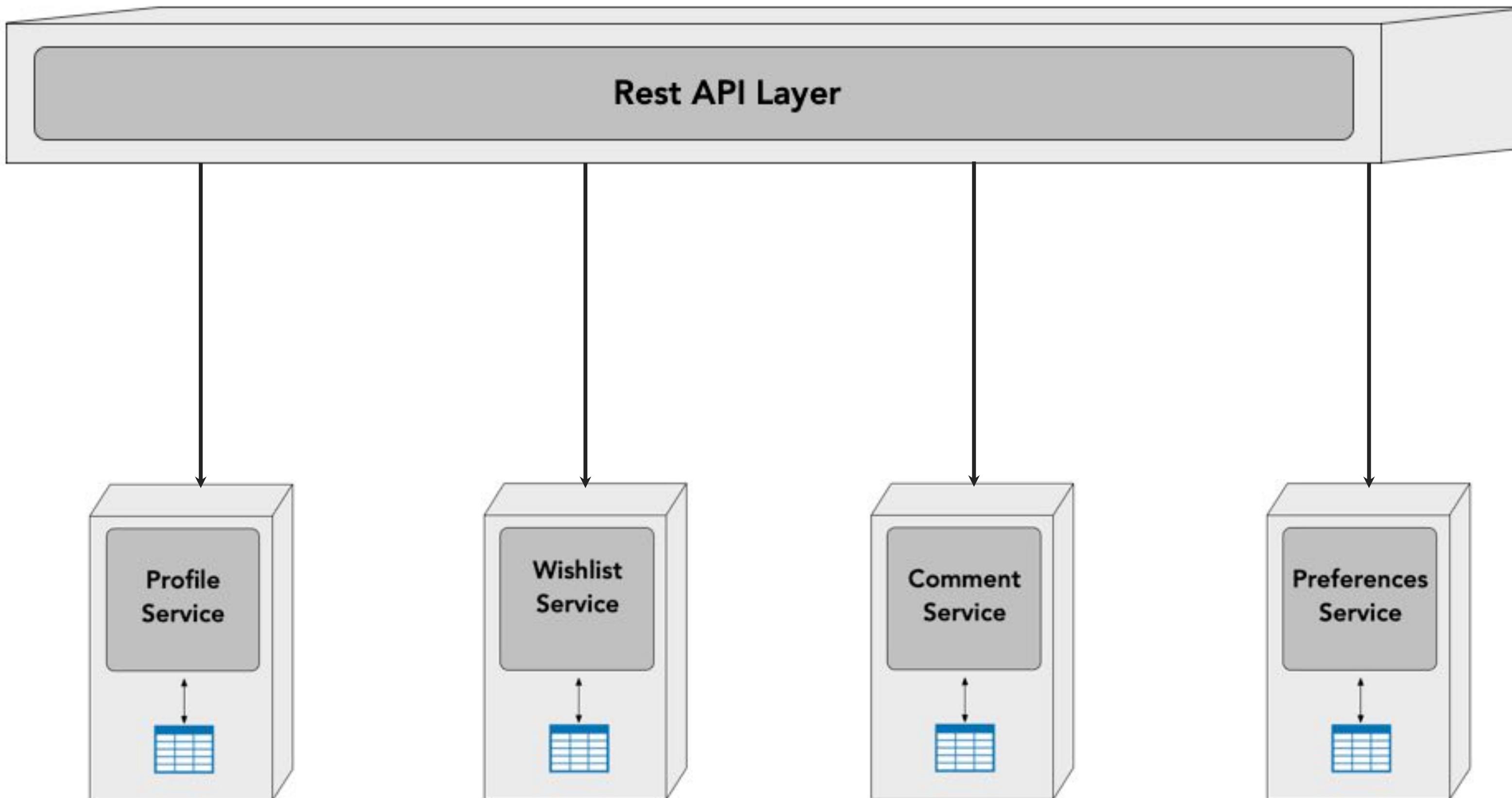
Orchestration in Microservices

broker
event driven architecture



Broker EDA is the same “shape” as microservices.

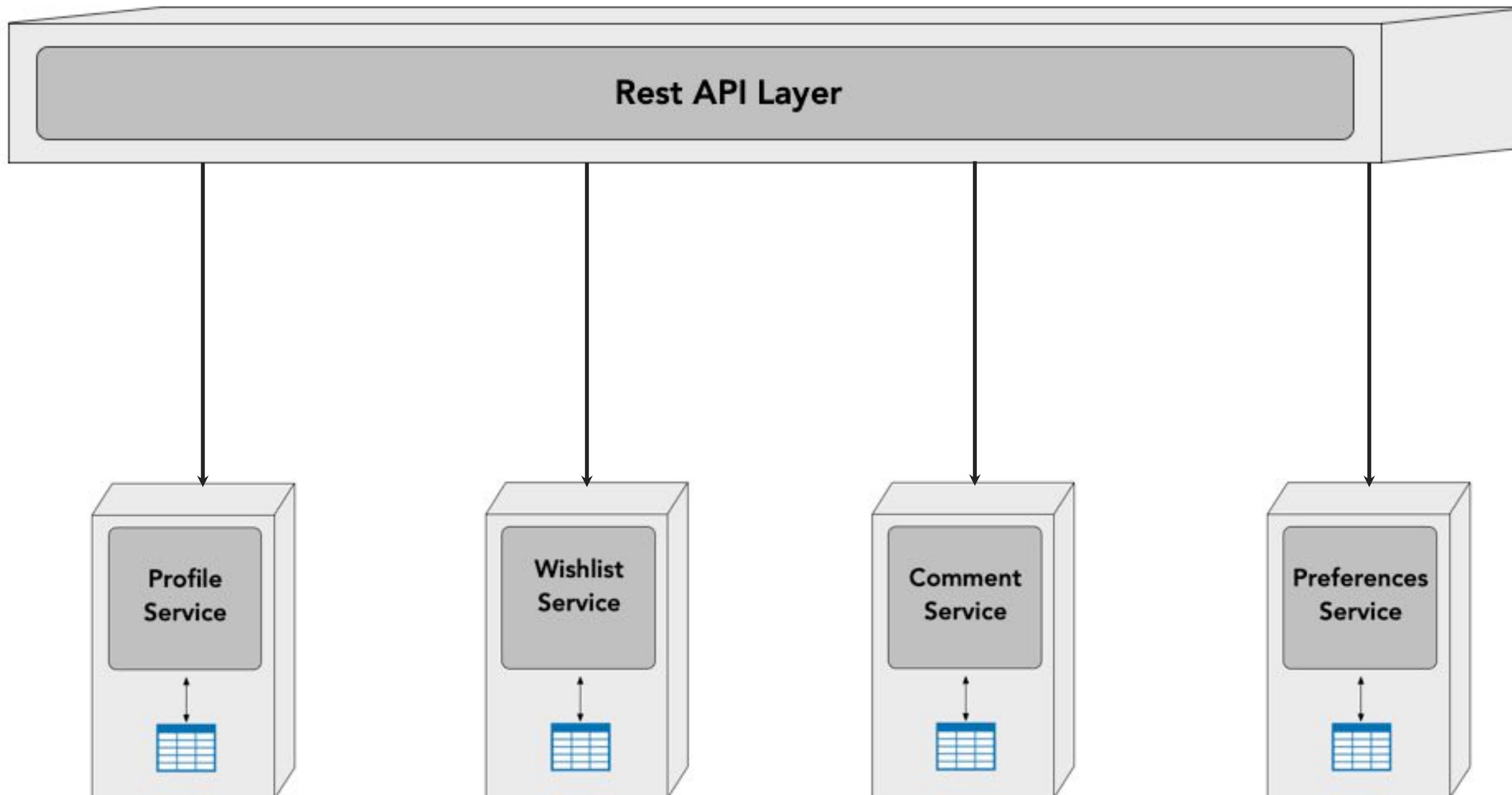
orchestration vs. choreography



orchestration vs. choreography

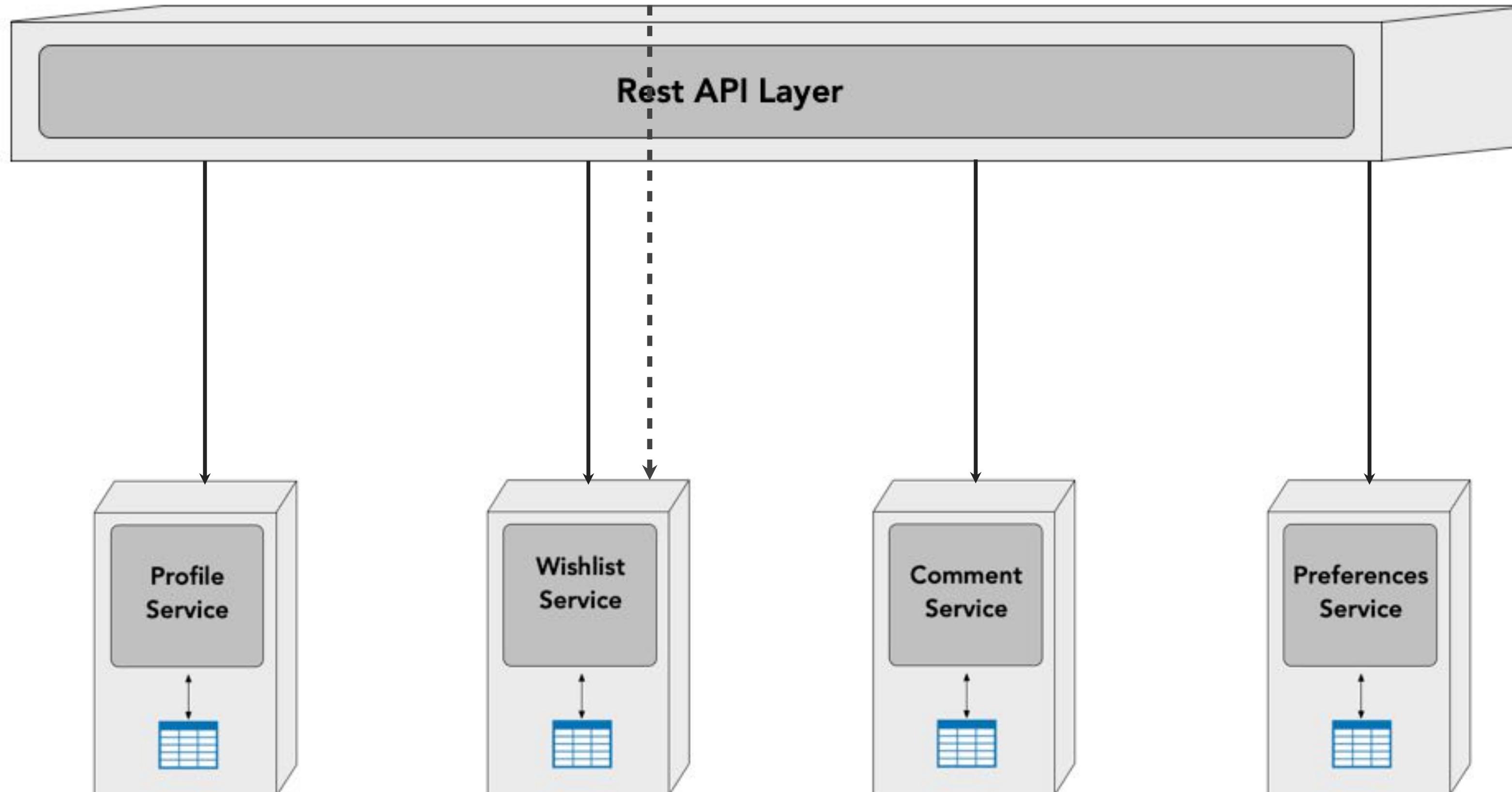


“get wishlist for customer 123”



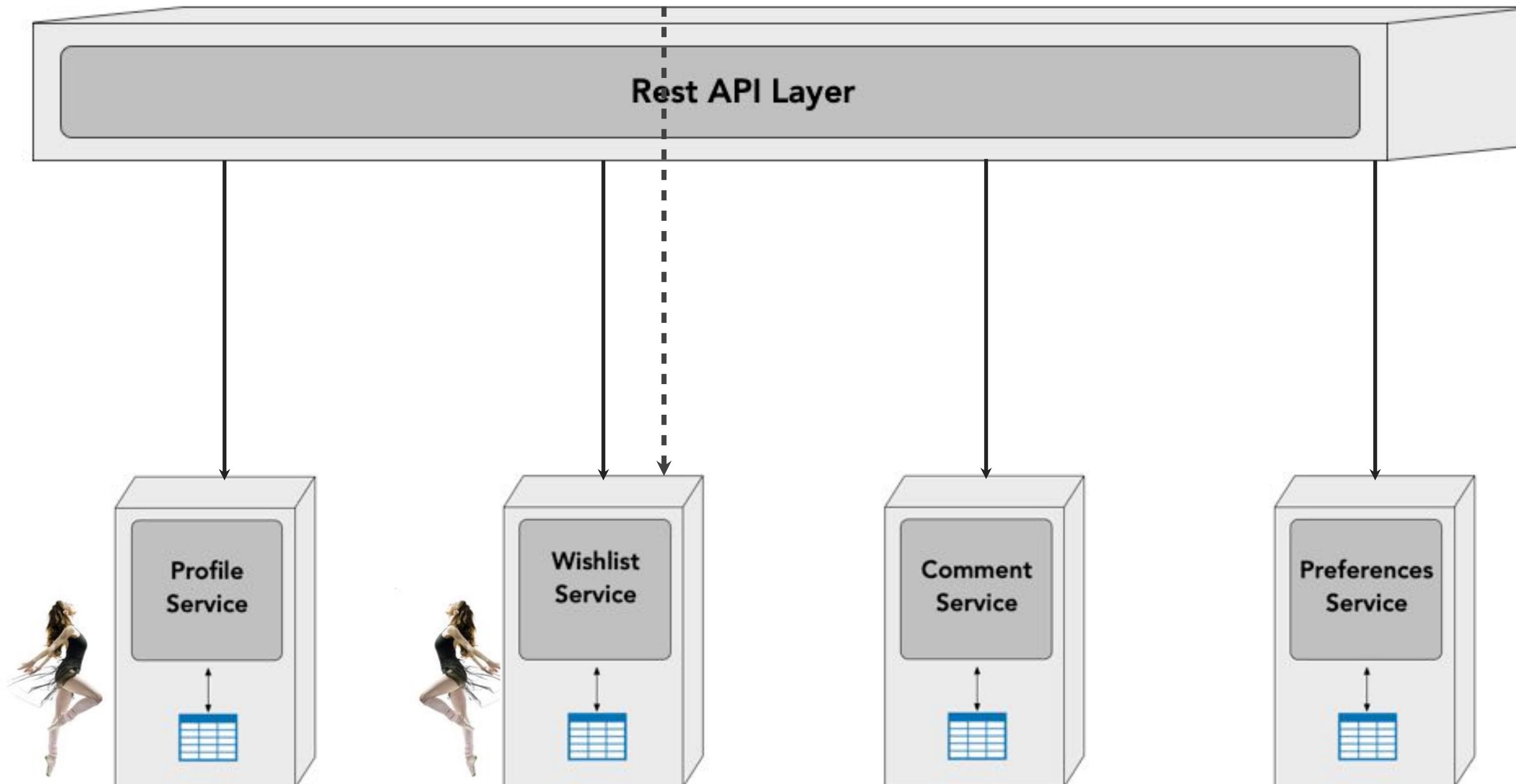
orchestration vs. choreography

 "get wishlist for customer 123" -----

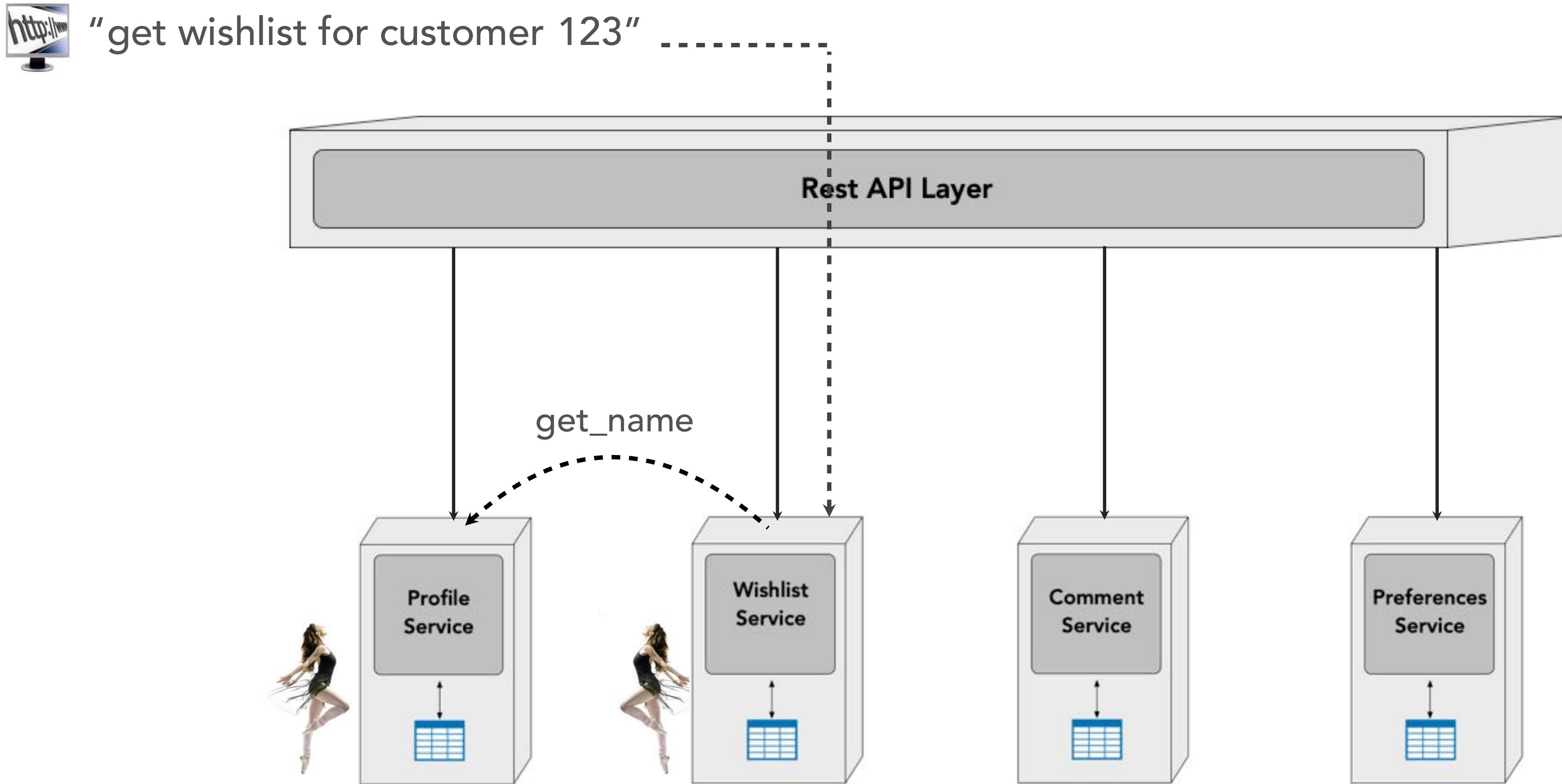


orchestration vs. choreography

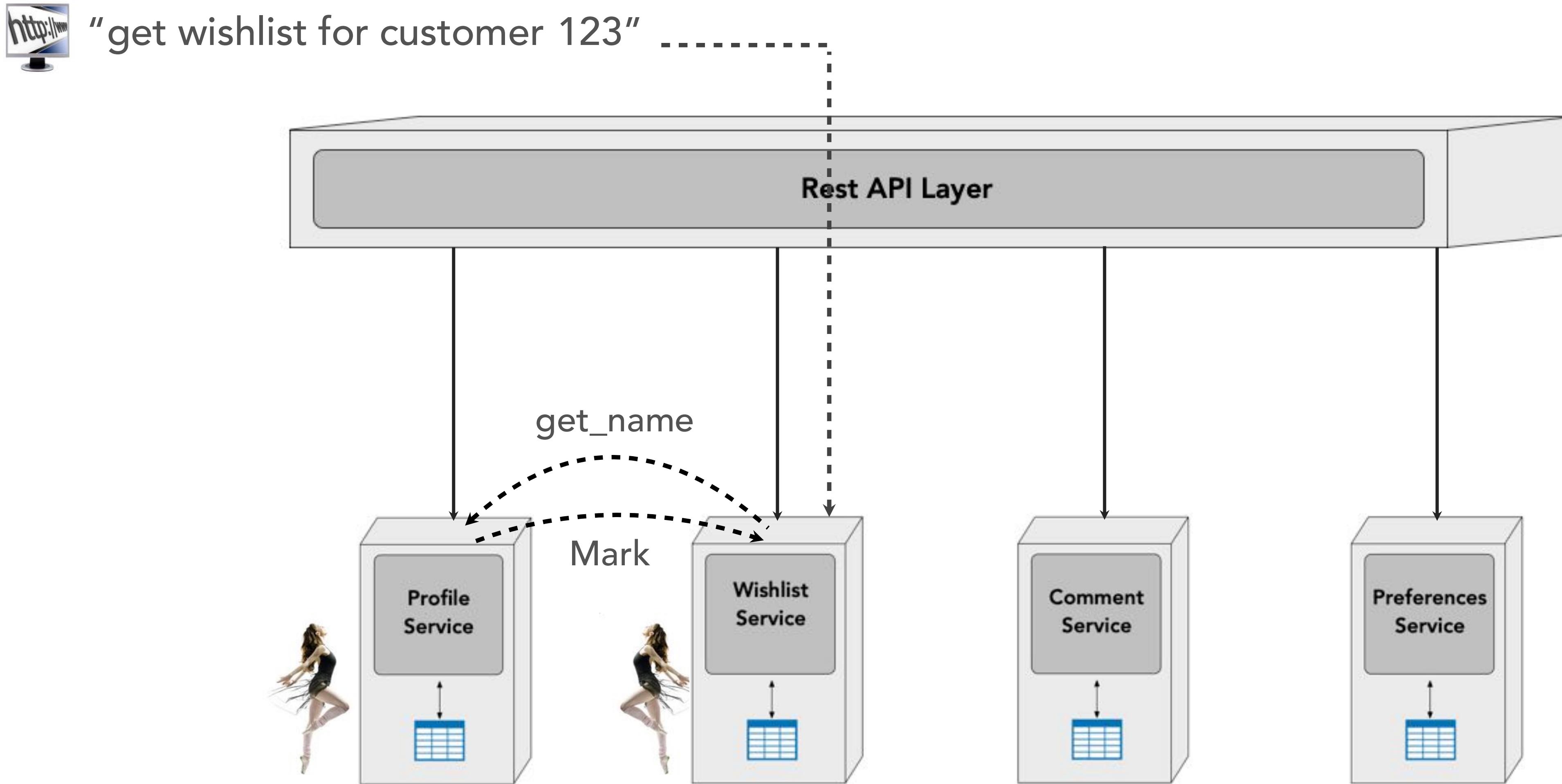
 "get wishlist for customer 123" -----



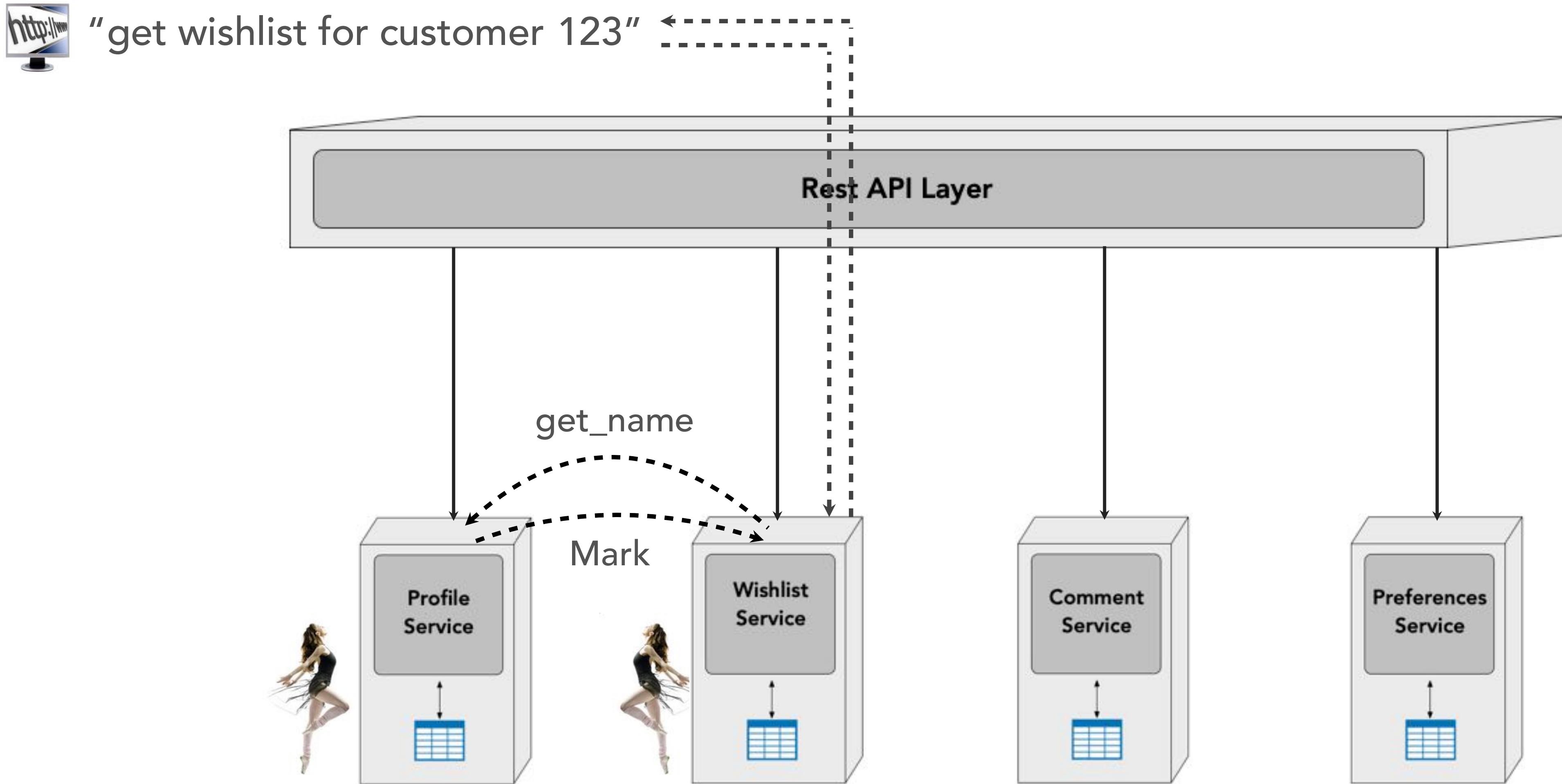
orchestration vs. choreography



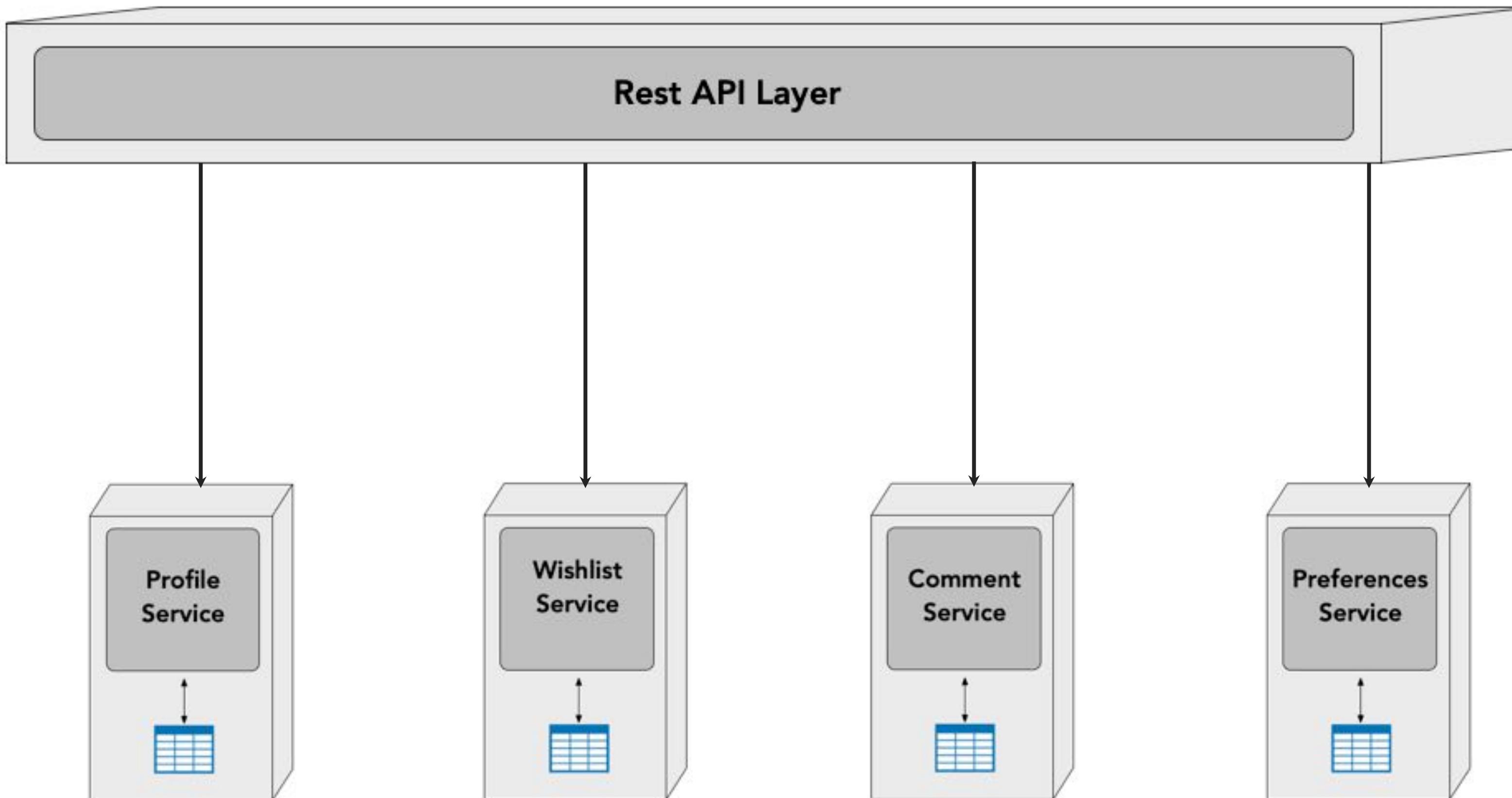
orchestration vs. choreography



orchestration vs. choreography



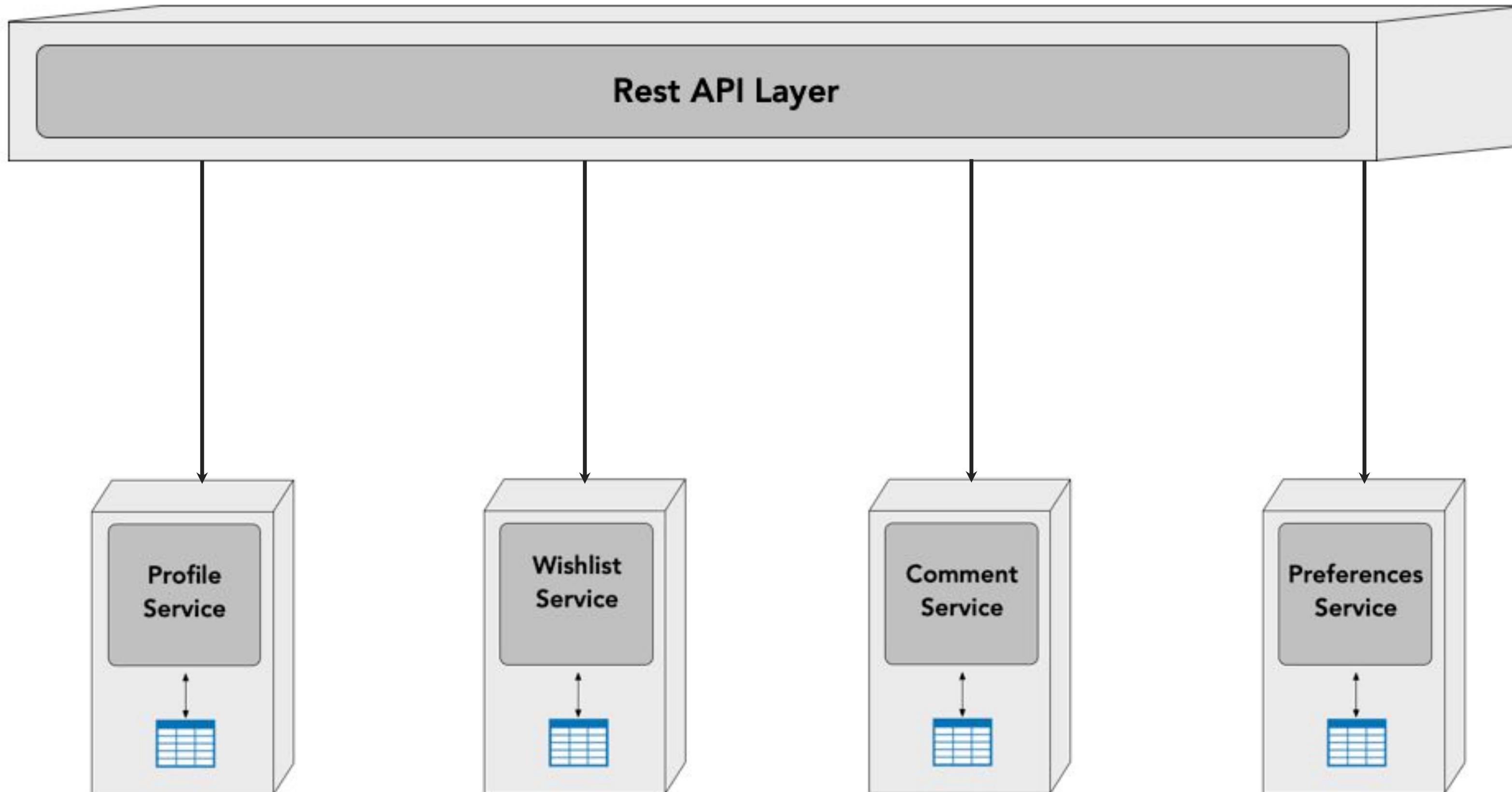
orchestration vs. choreography



orchestration vs. choreography



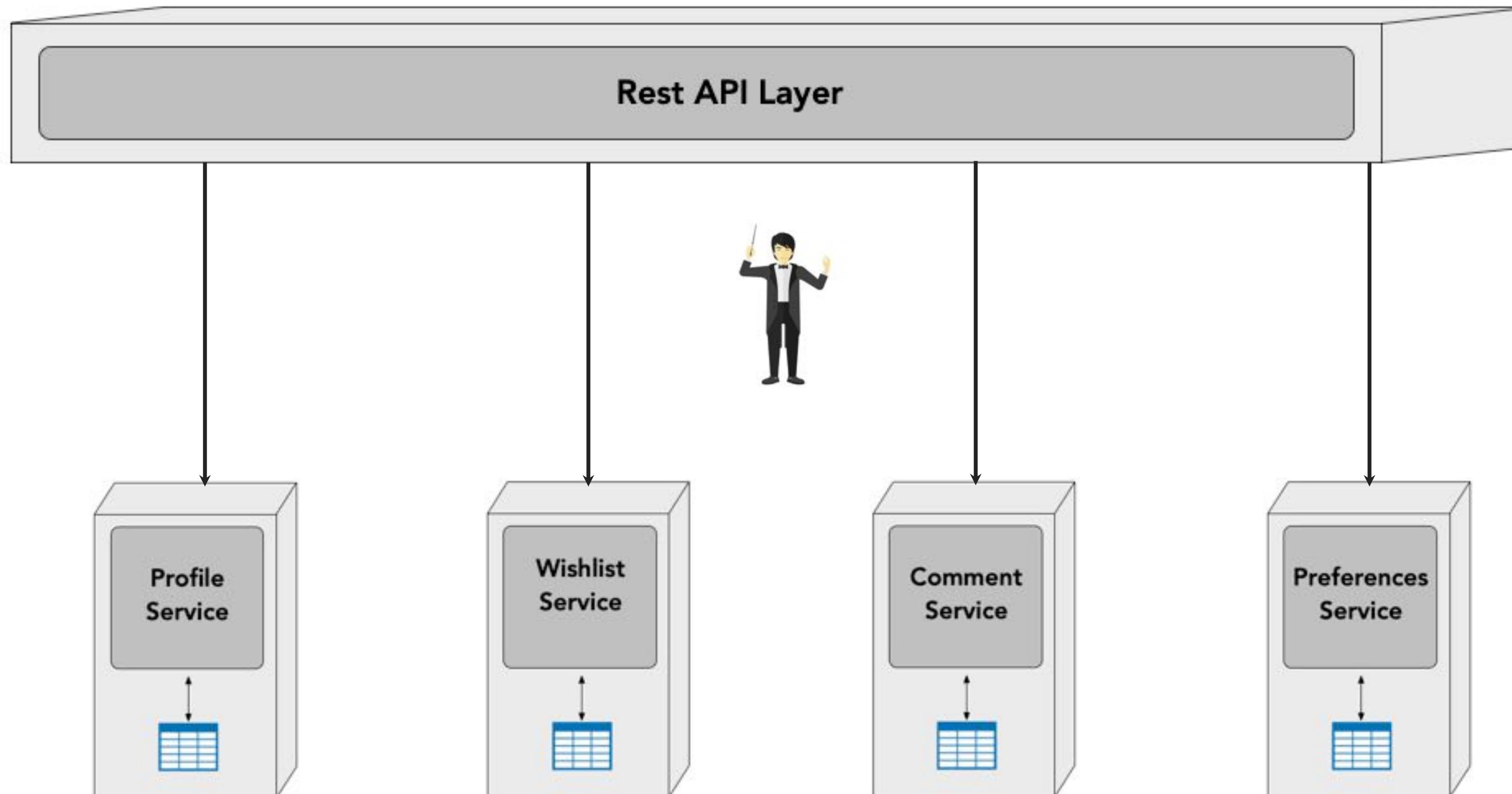
“get all data for customer 123”



orchestration vs. choreography



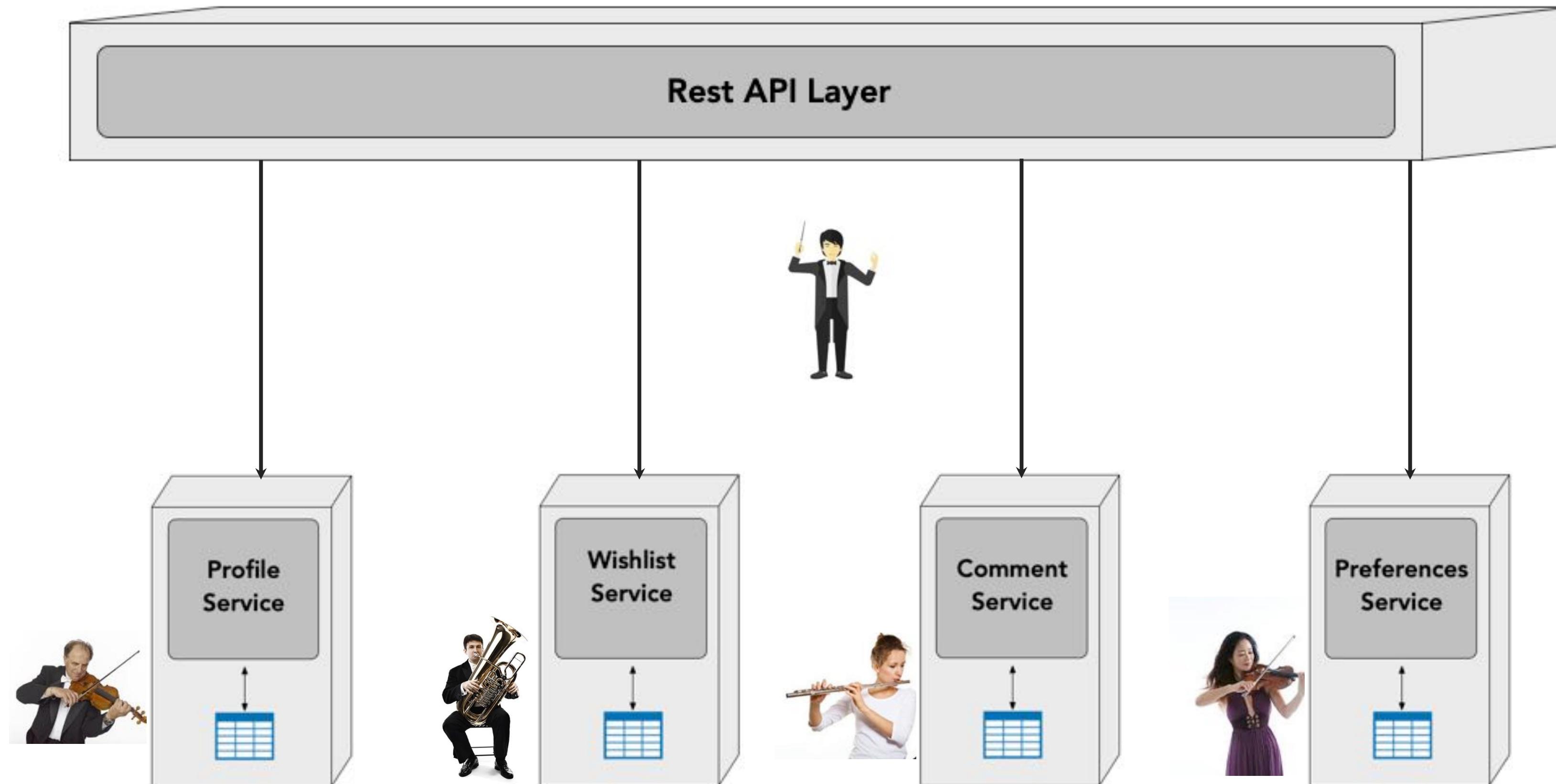
“get all data for customer 123”



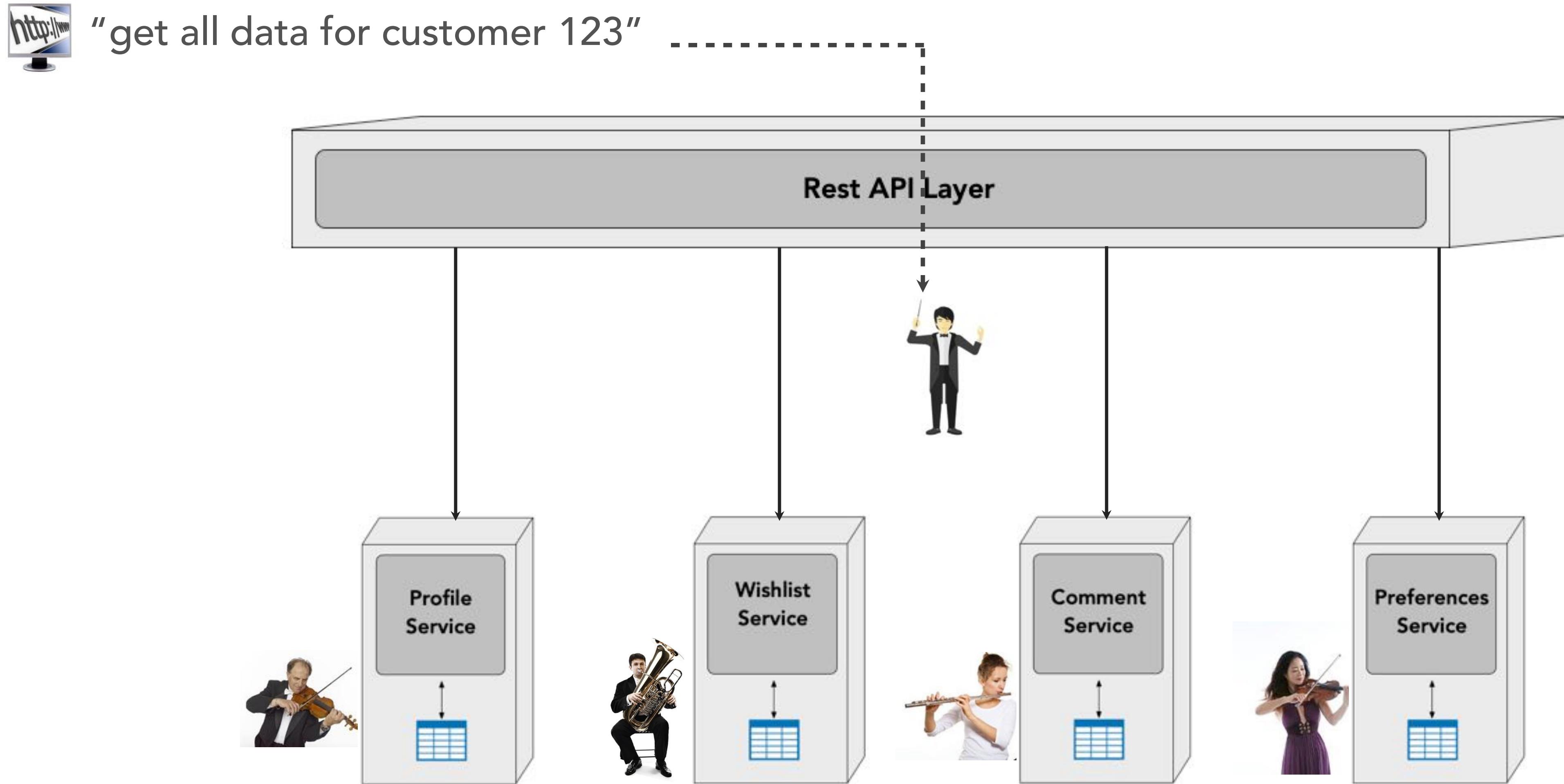
orchestration vs. choreography



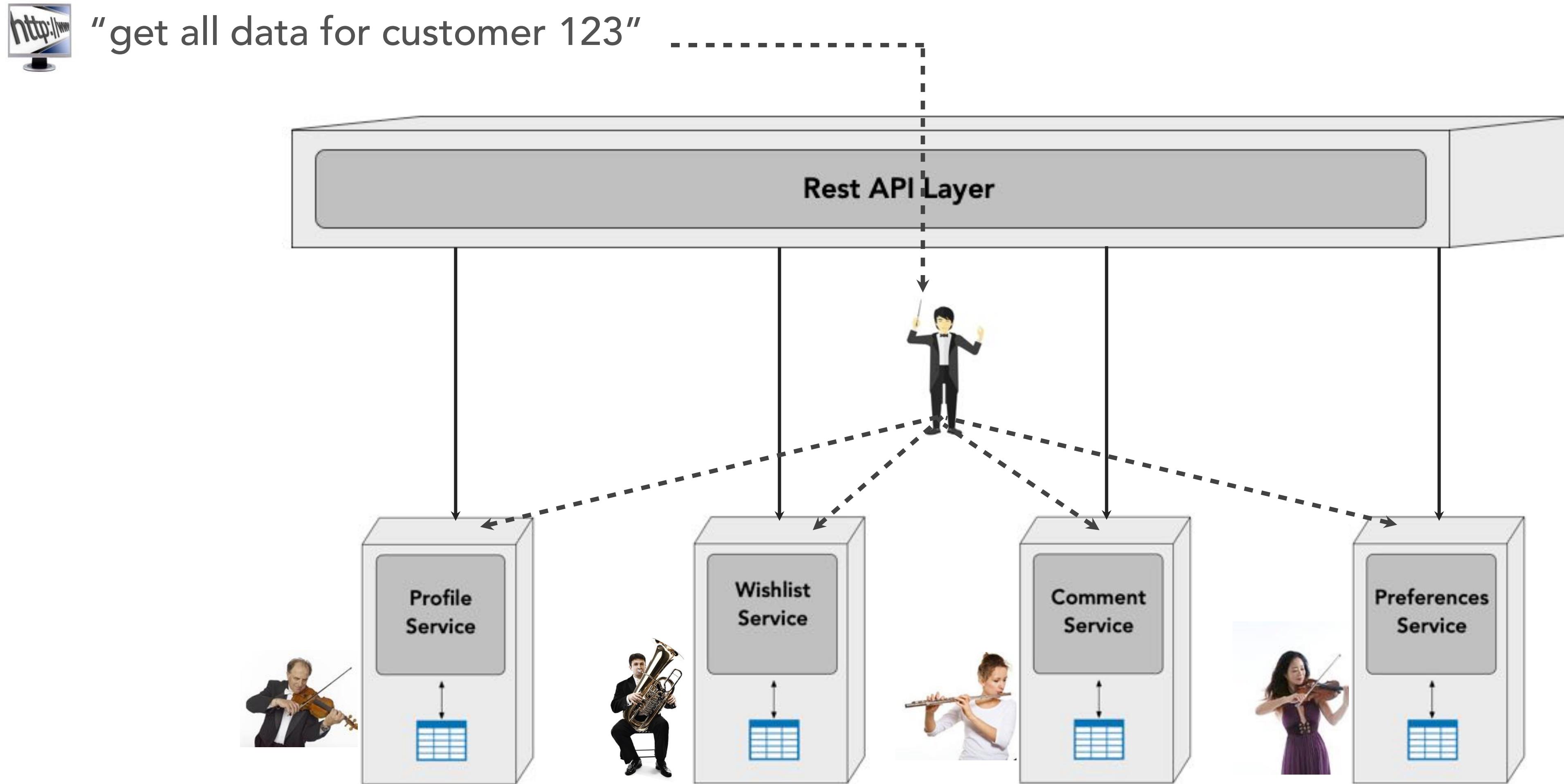
“get all data for customer 123”



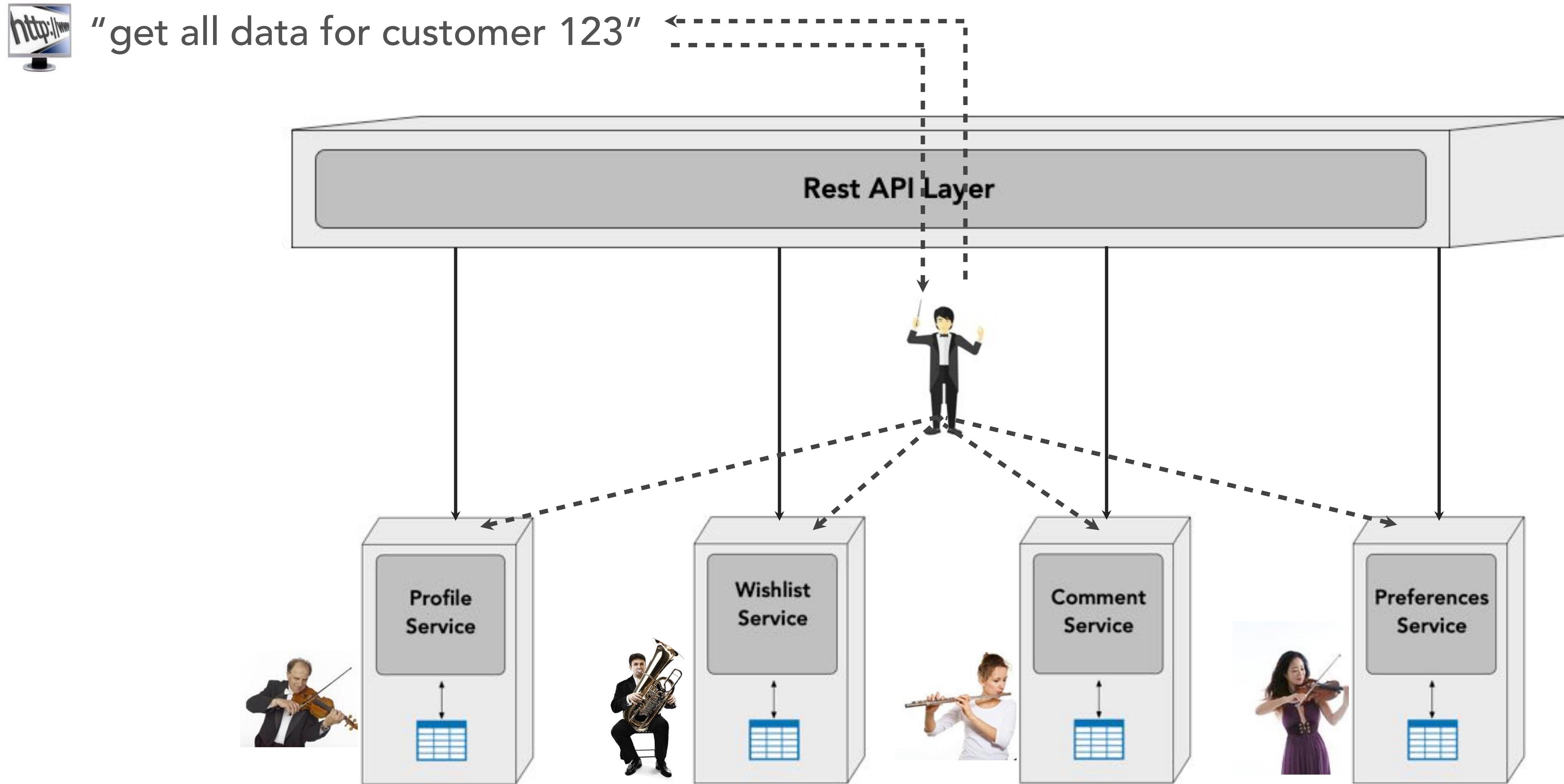
orchestration vs. choreography



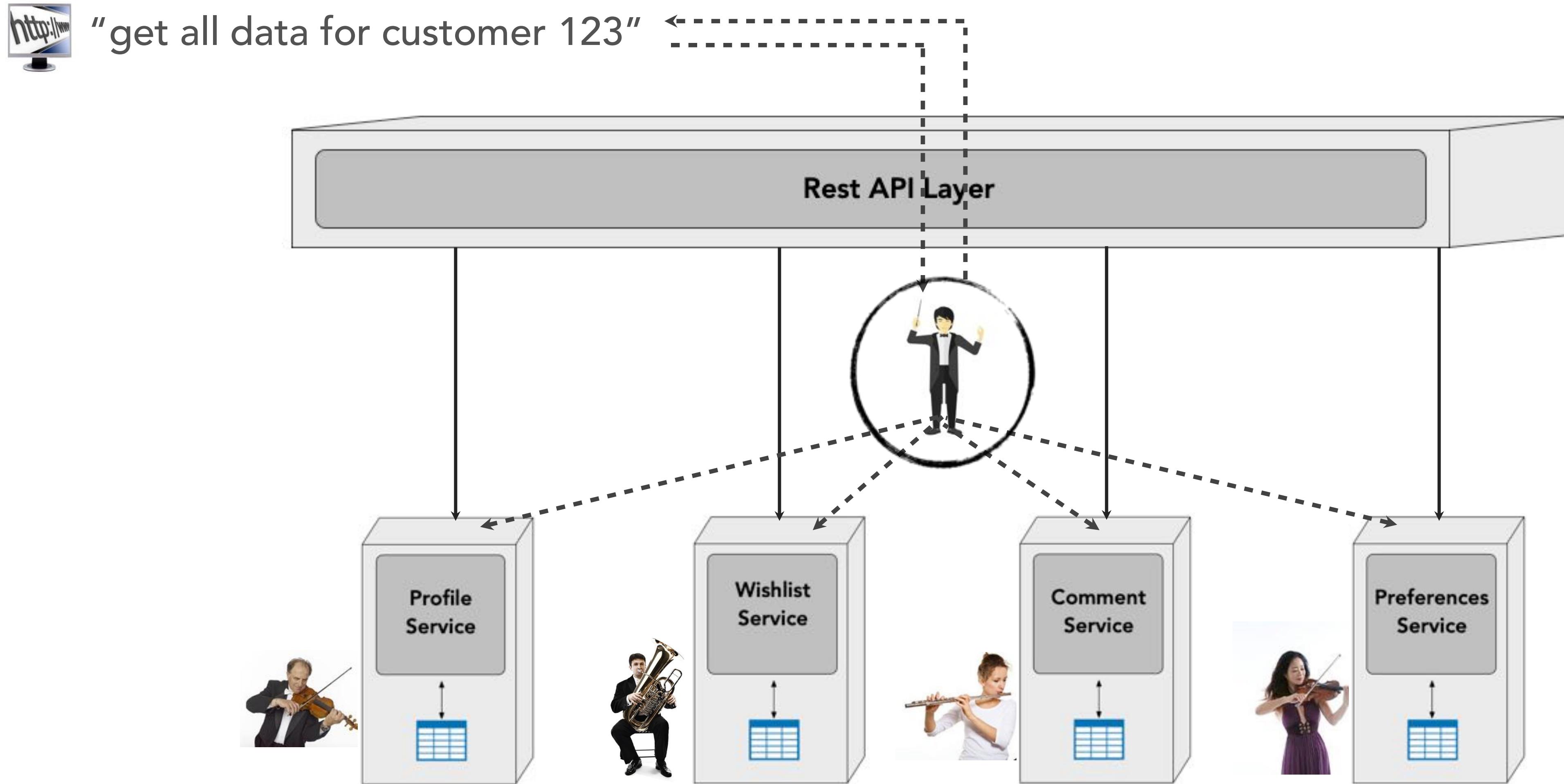
orchestration vs. choreography



orchestration vs. choreography



orchestration vs. choreography



orchestration vs. choreography



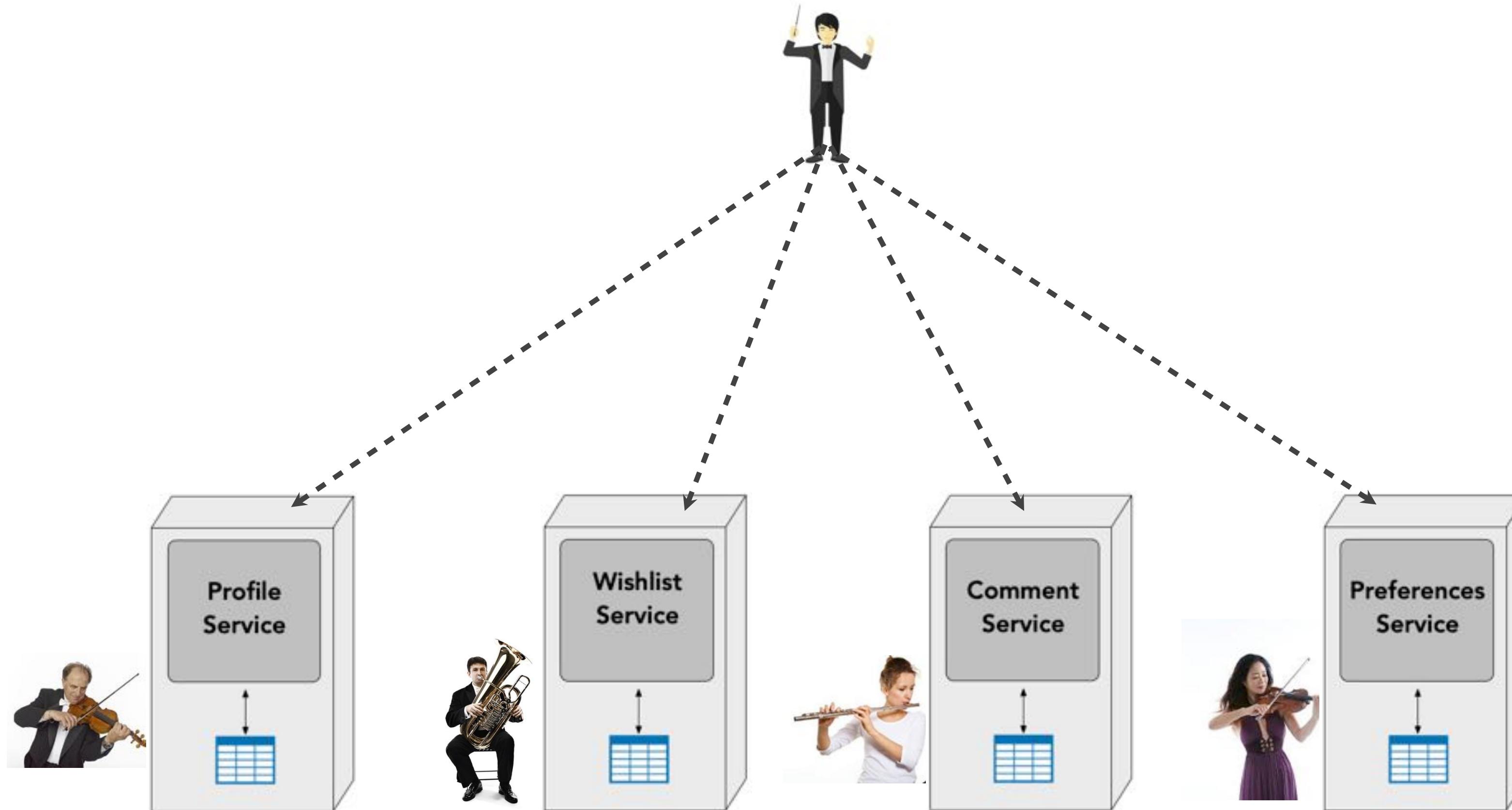
orchestration vs. choreography

1. multicasting logic



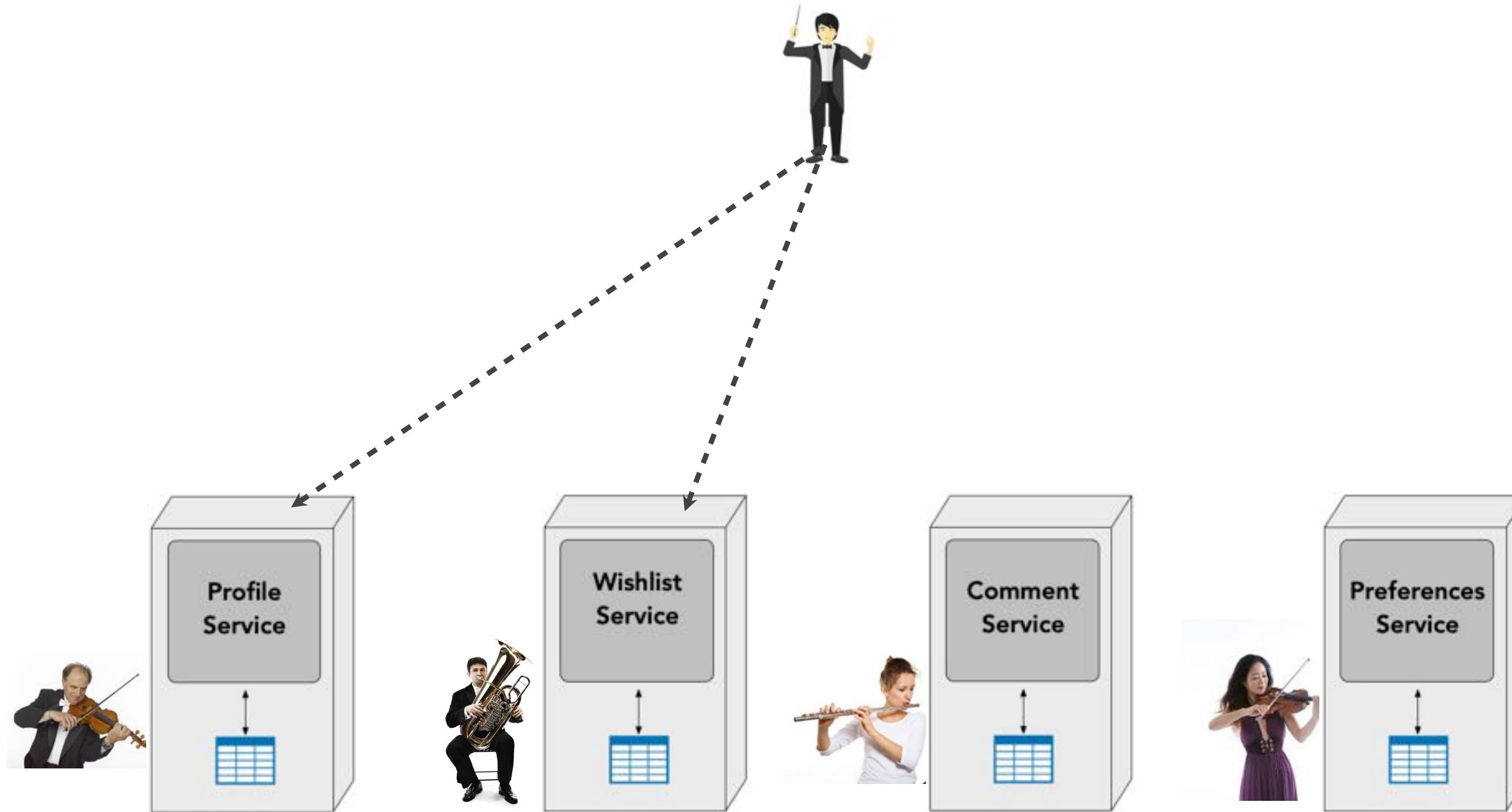
orchestration vs. choreography

1. multicasting logic



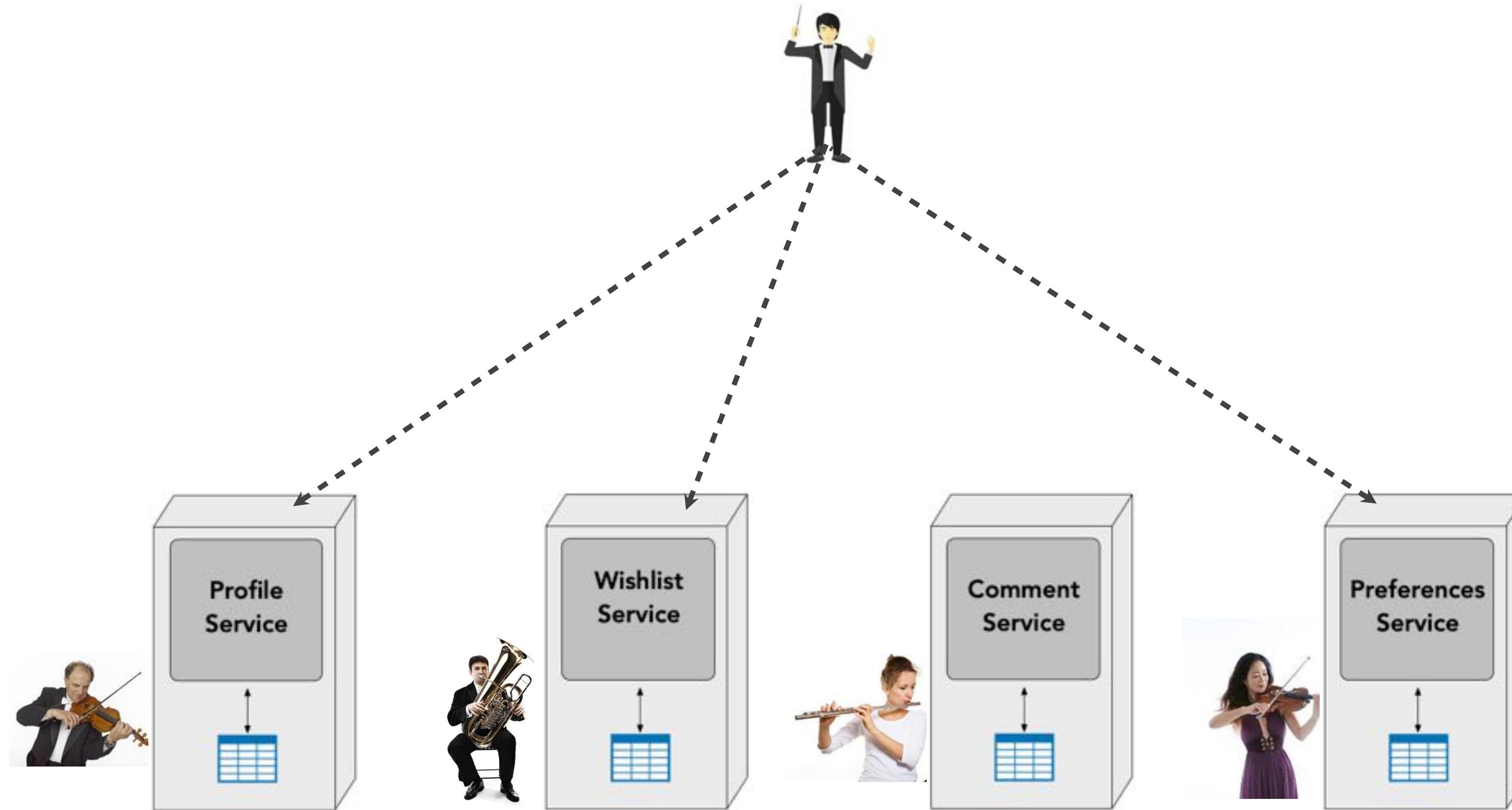
orchestration vs. choreography

1. multicasting logic



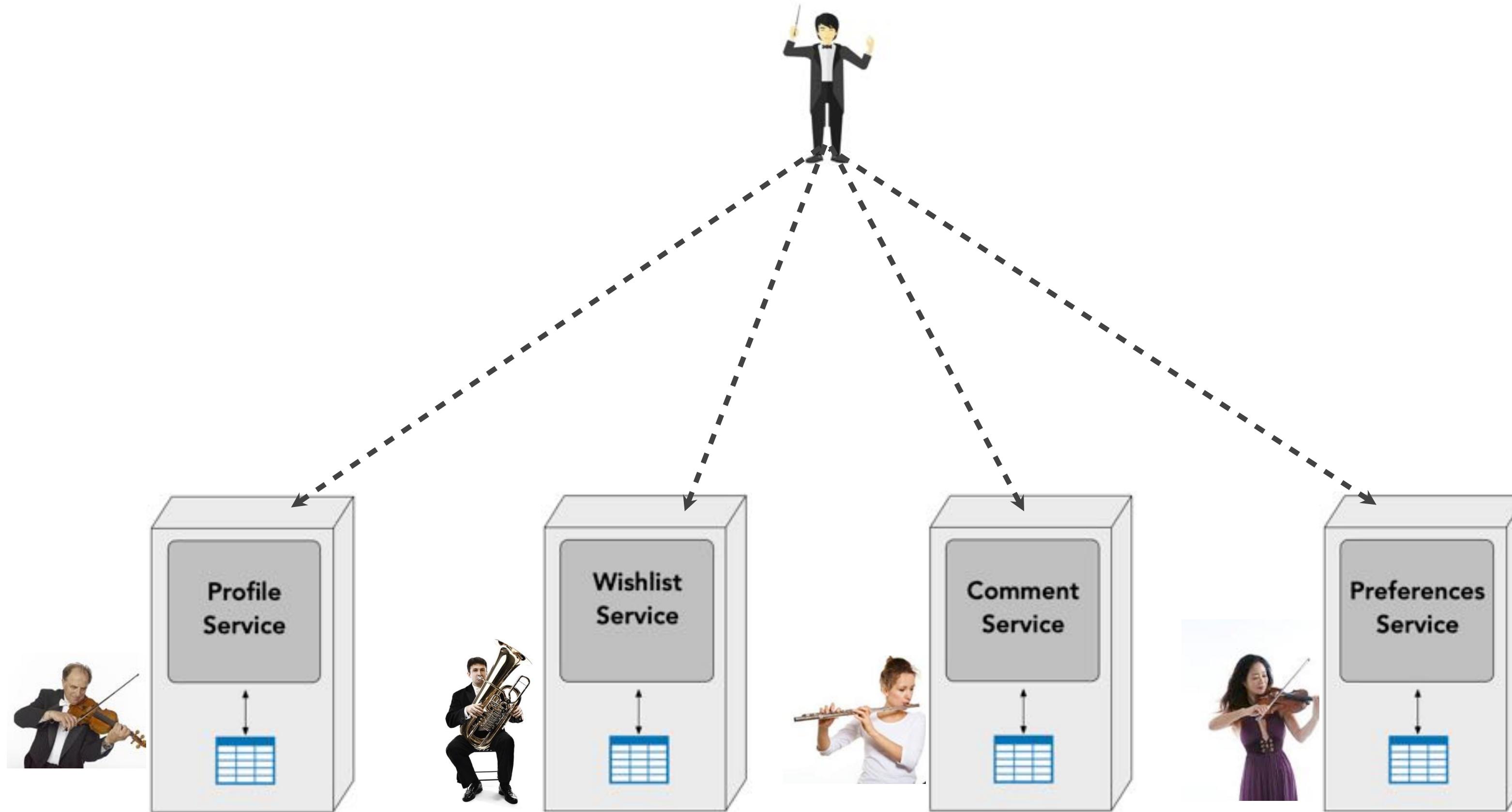
orchestration vs. choreography

1. multicasting logic



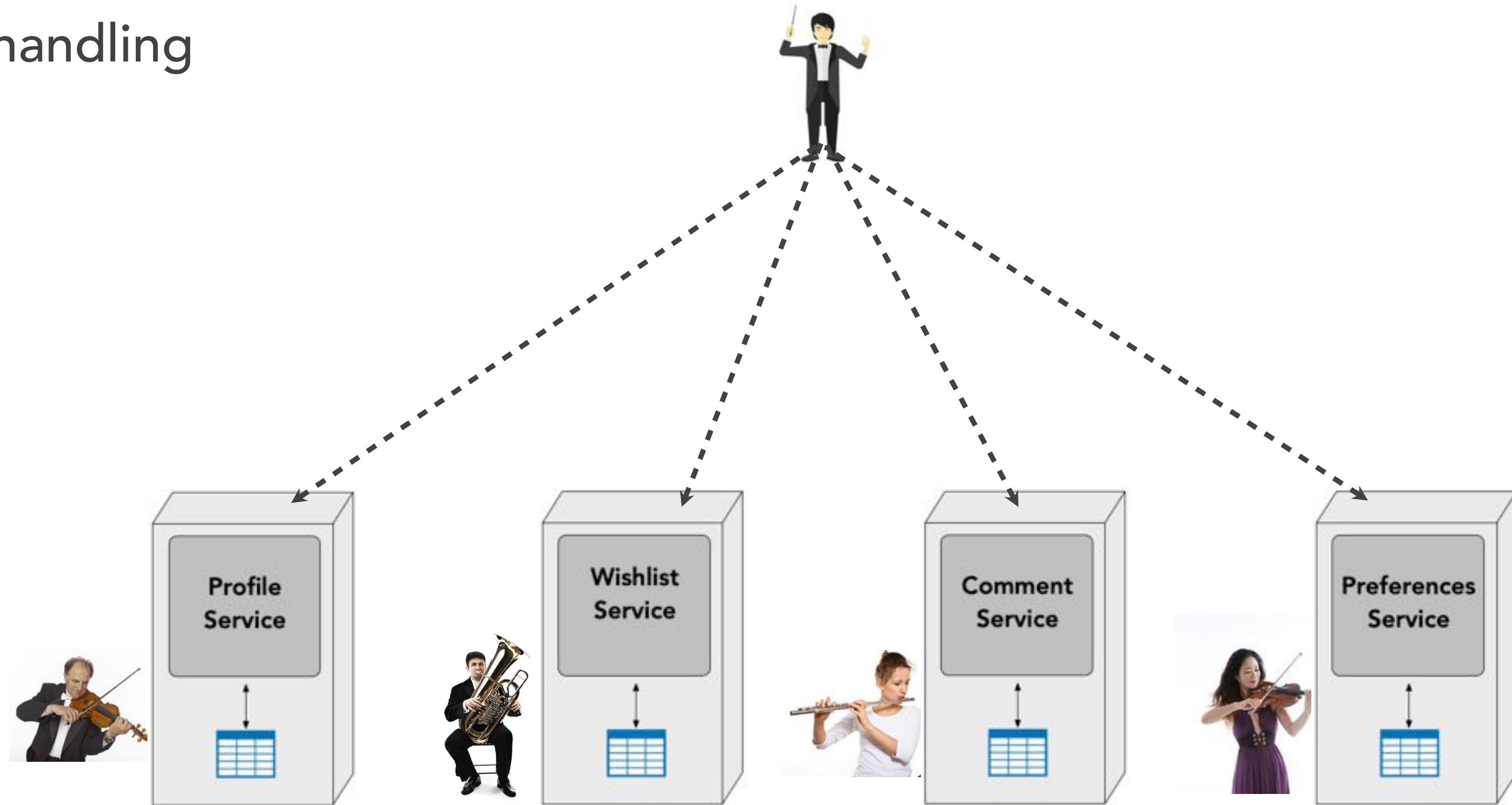
orchestration vs. choreography

1. multicasting logic



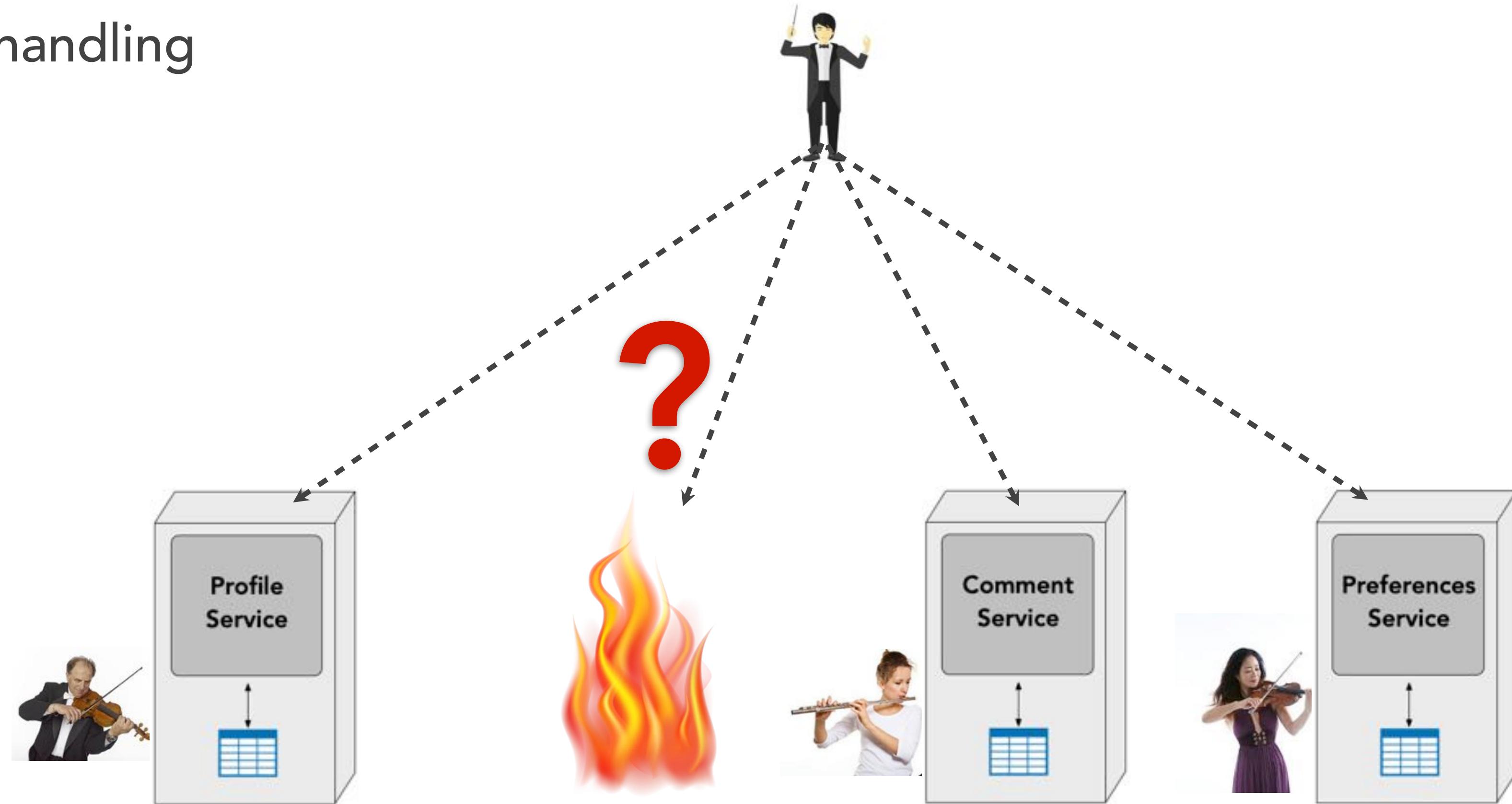
orchestration vs. choreography

1. multicasting logic
2. error handling



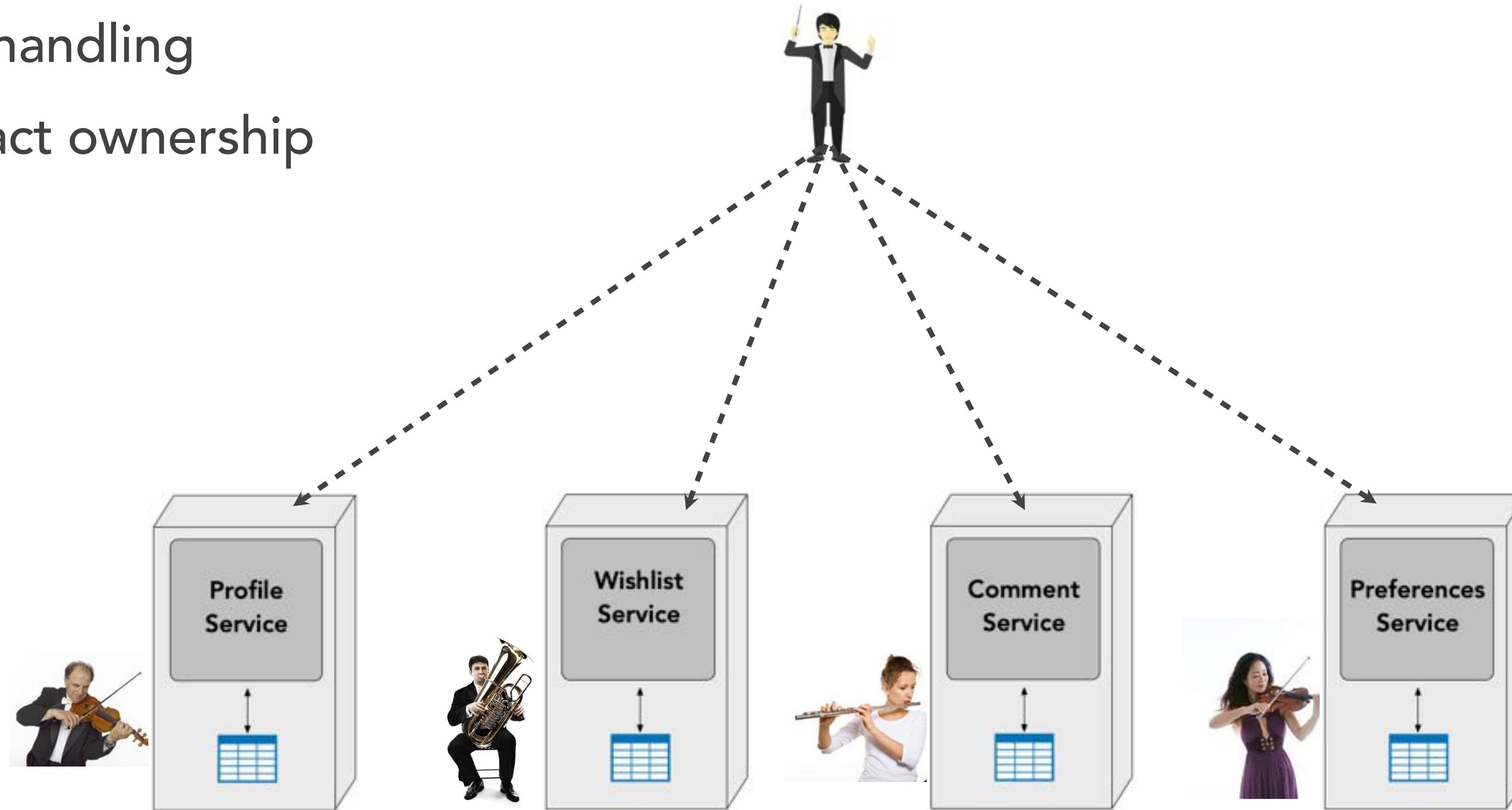
orchestration vs. choreography

1. multicasting logic
2. error handling



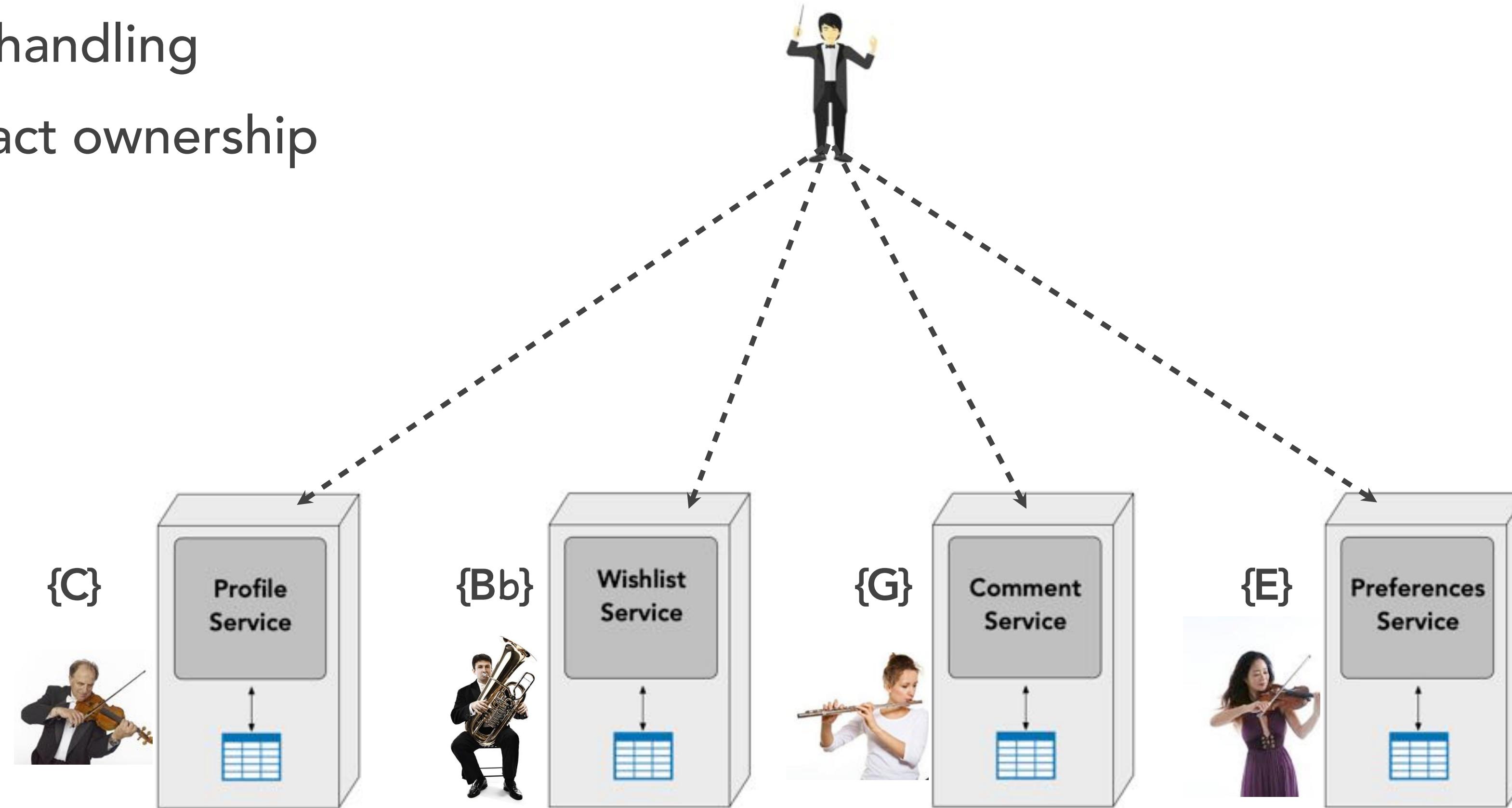
orchestration vs. choreography

1. multicasting logic
2. error handling
3. contract ownership



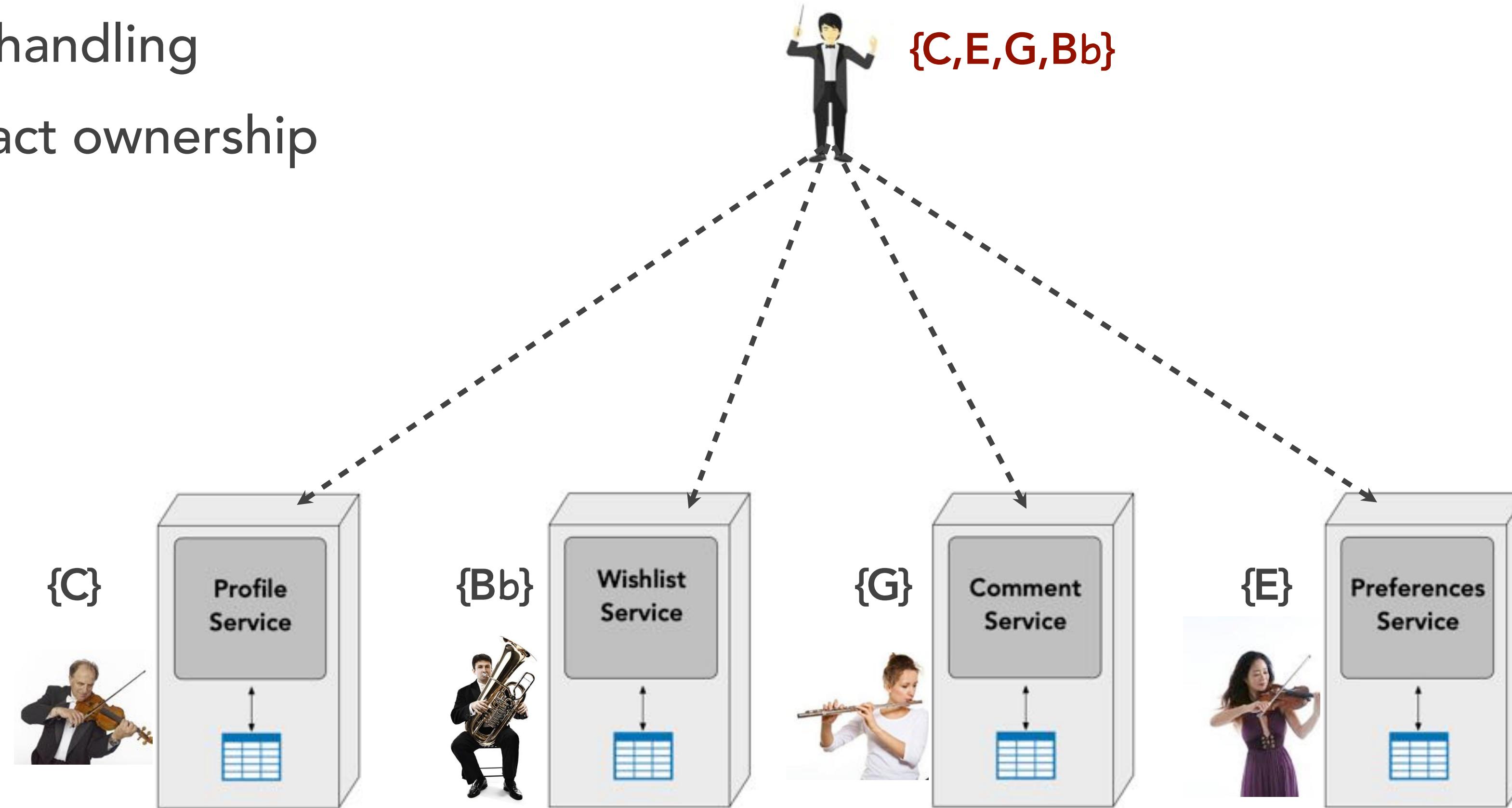
orchestration vs. choreography

1. multicasting logic
2. error handling
3. contract ownership



orchestration vs. choreography

1. multicasting logic
2. error handling
3. contract ownership

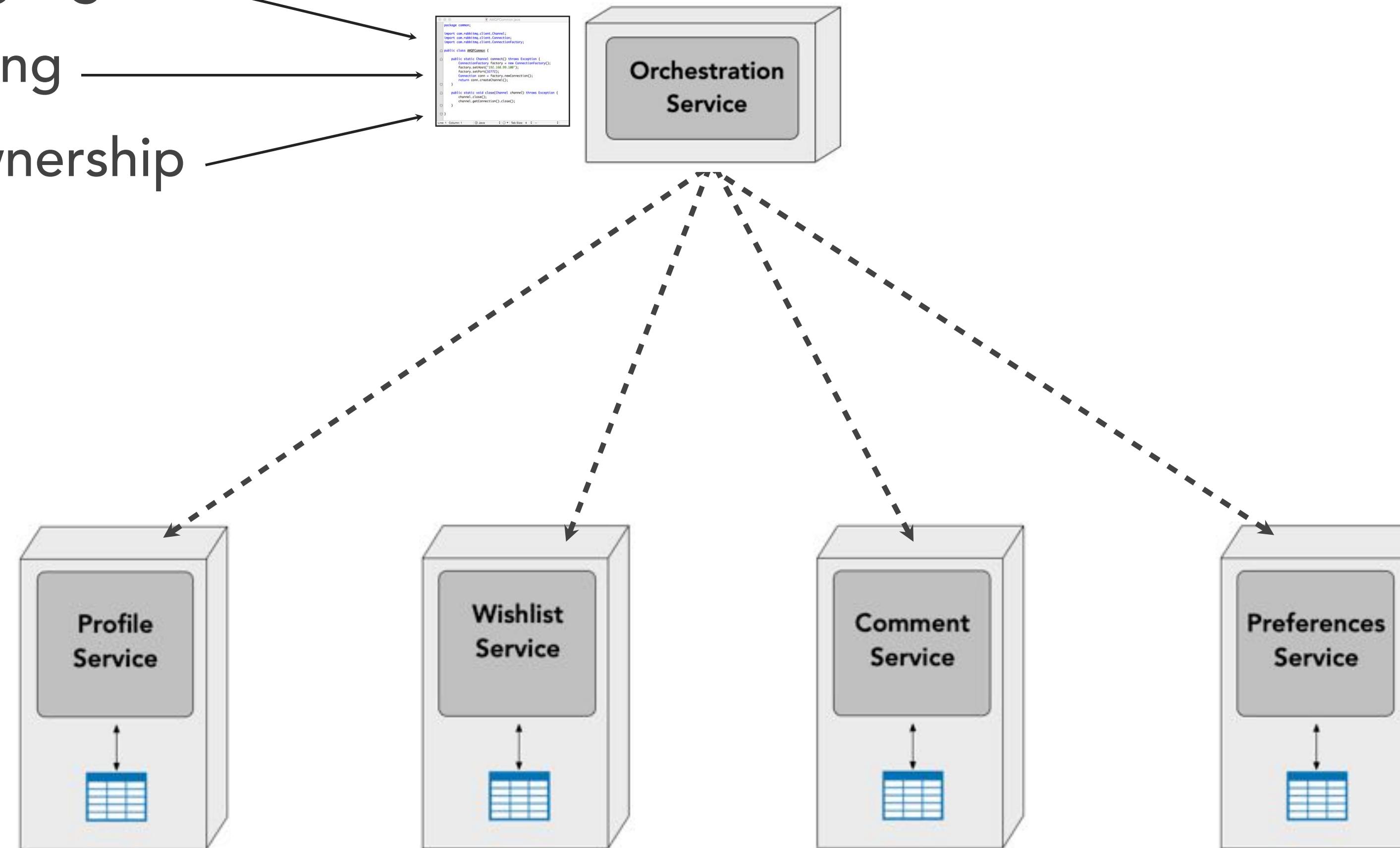


orchestration vs. choreography

1. multicasting logic

2. error handling

3. contract ownership

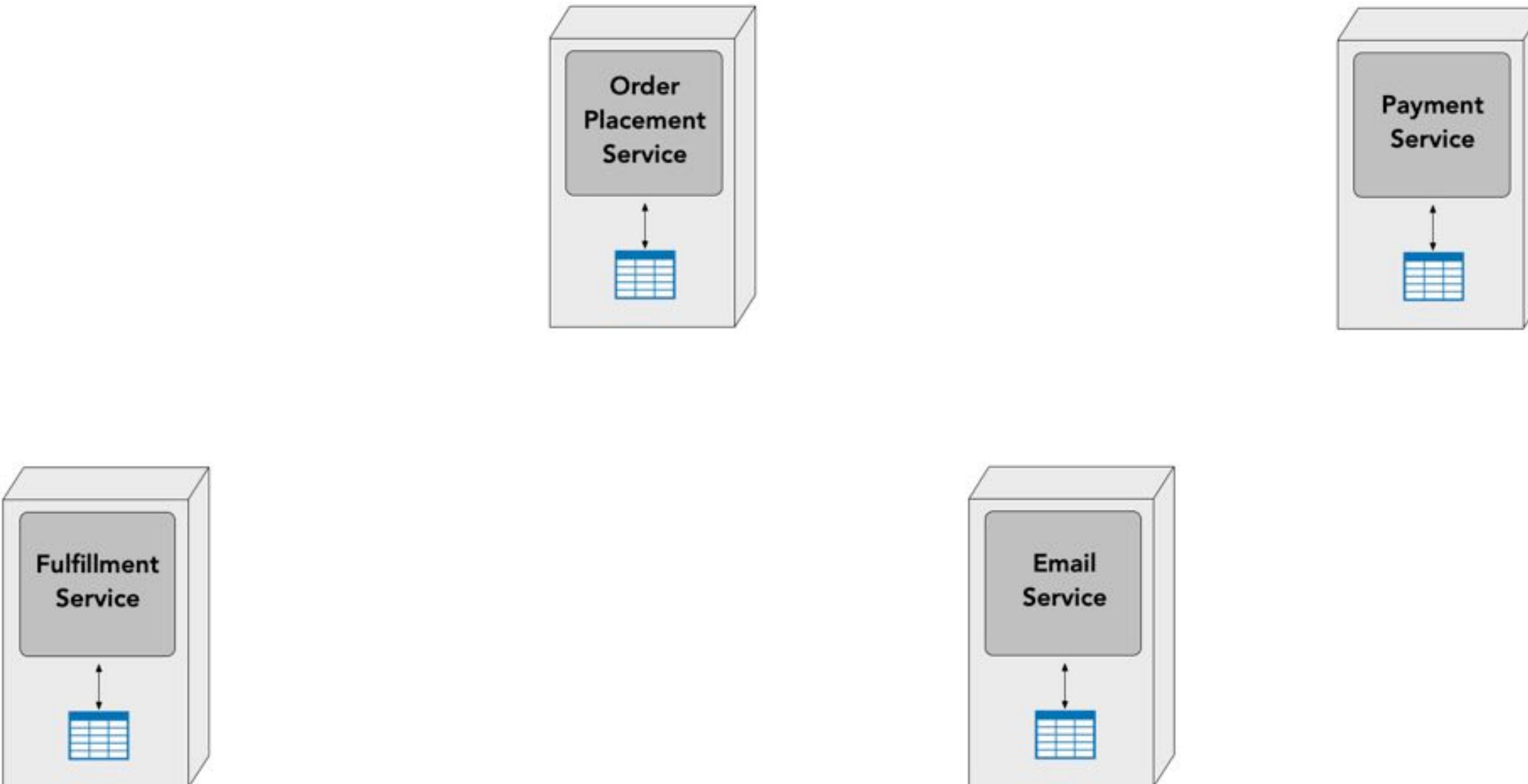


Sagas

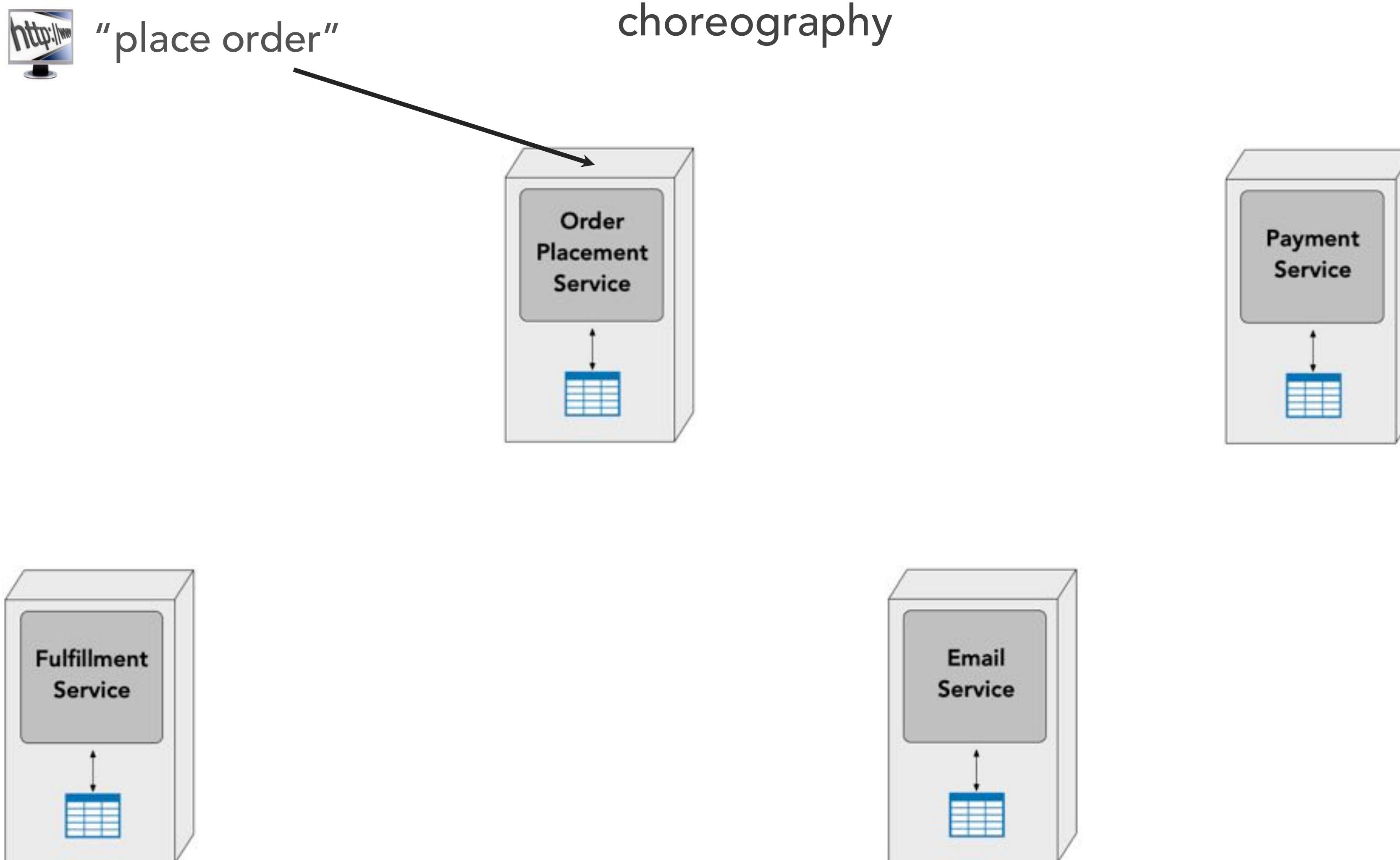
```
placeOrder(customer, order, lineItems) {  
    startTransaction {  
        foreach i in lineItems  
            order.addItem(i)  
        customer.add(o)  
    }  
}
```

workflow patterns

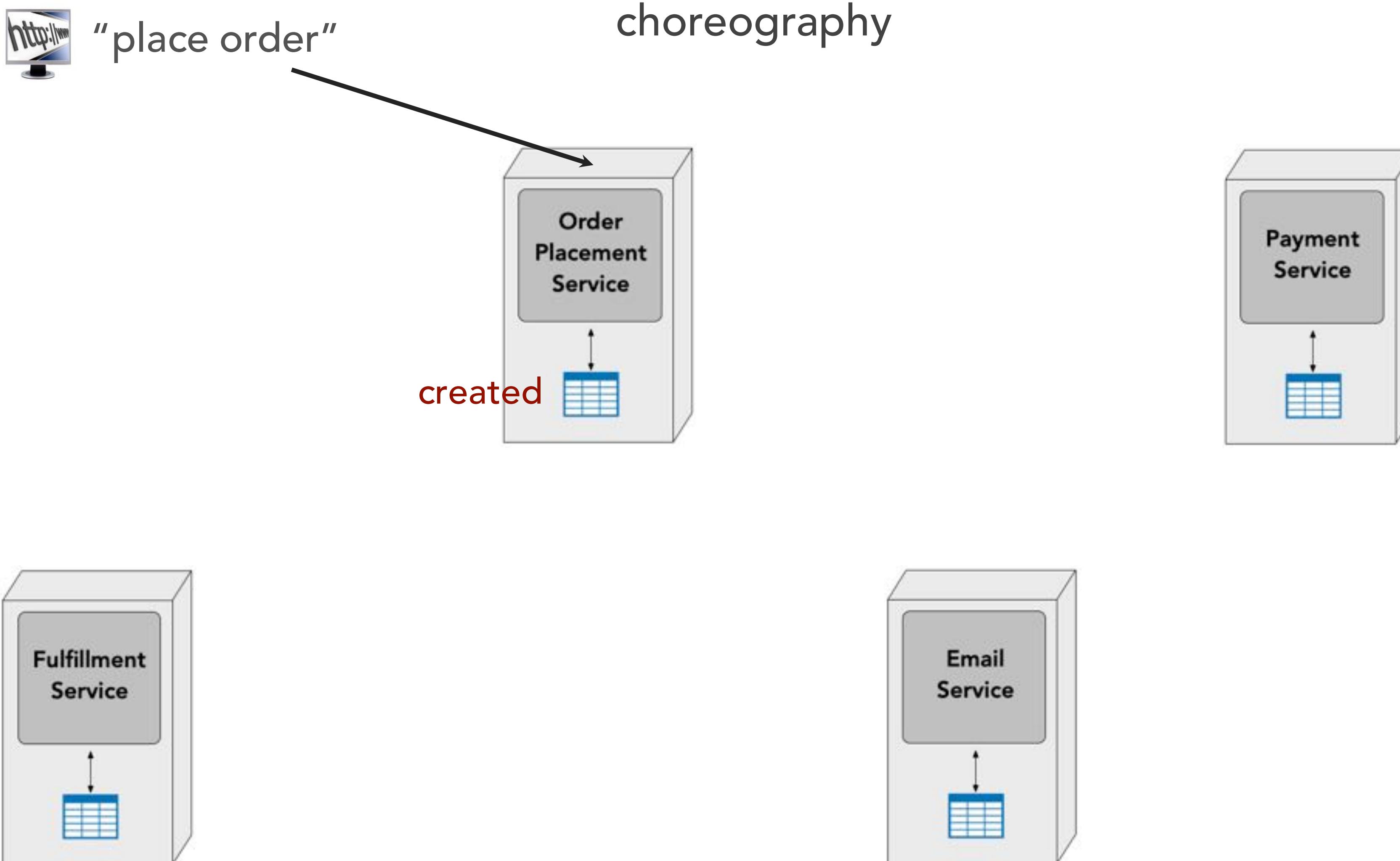
choreography



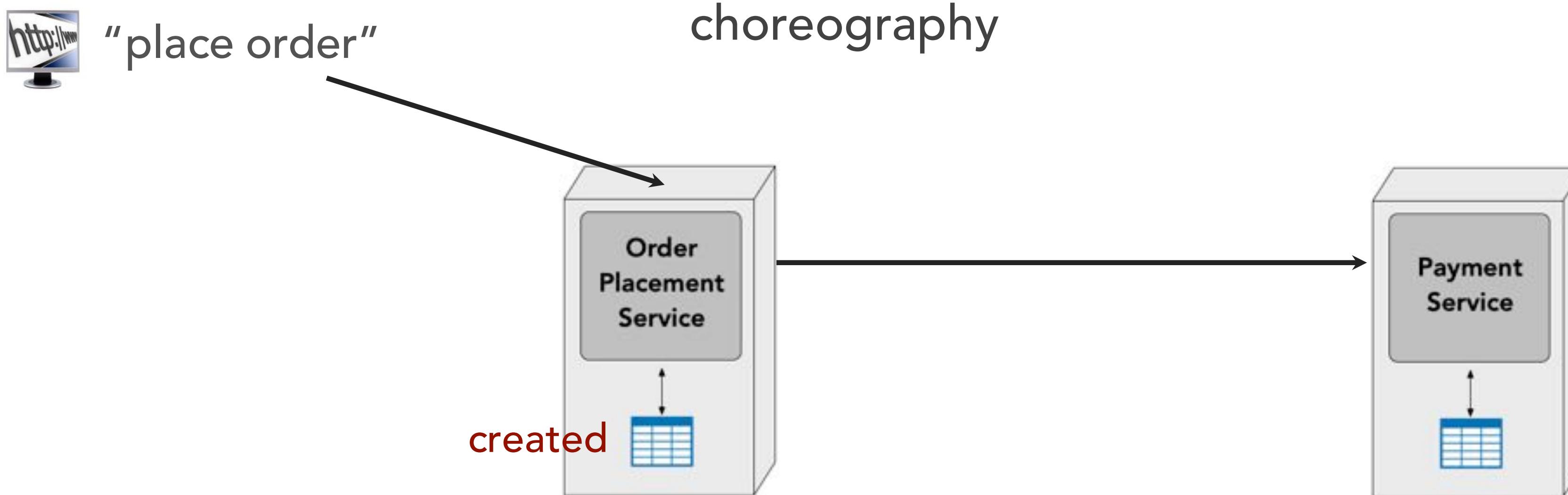
workflow patterns



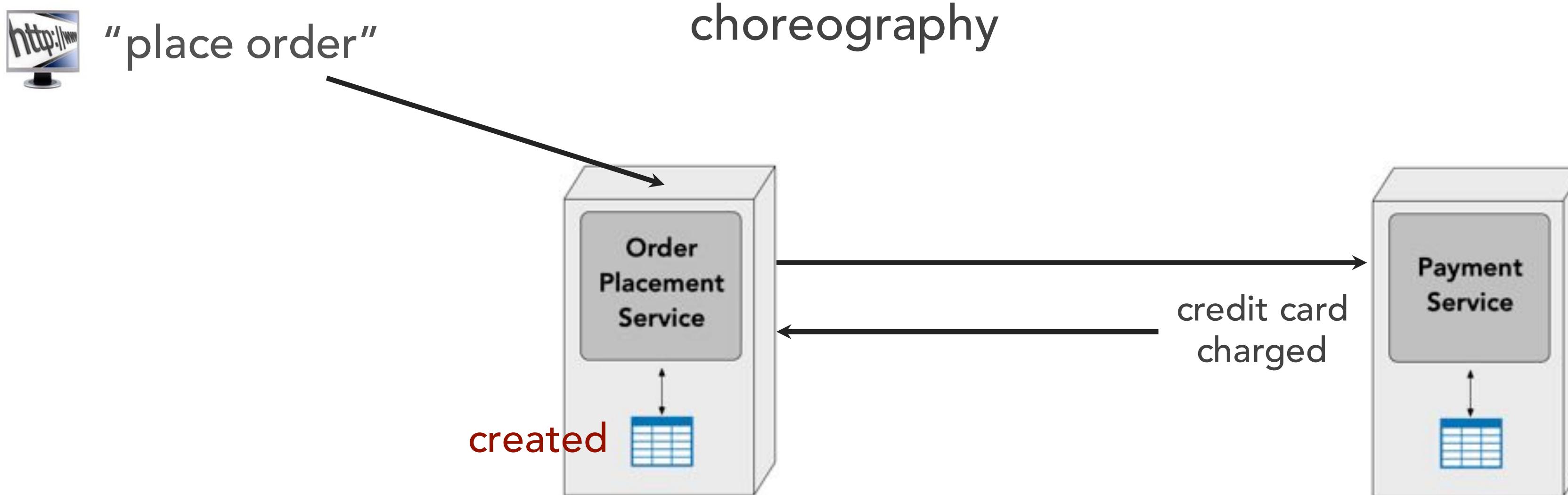
workflow patterns



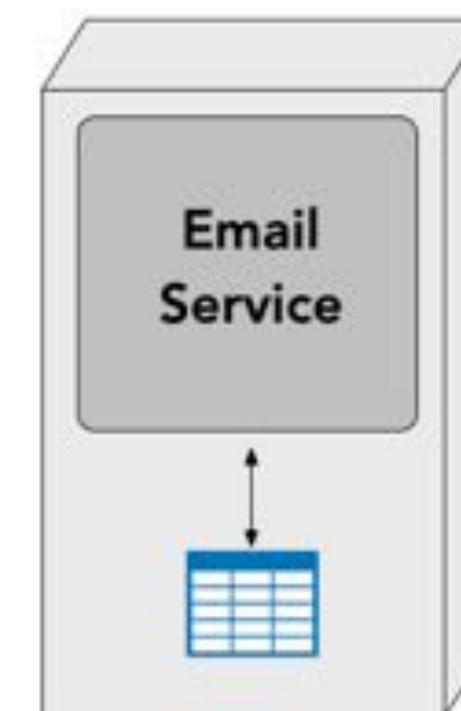
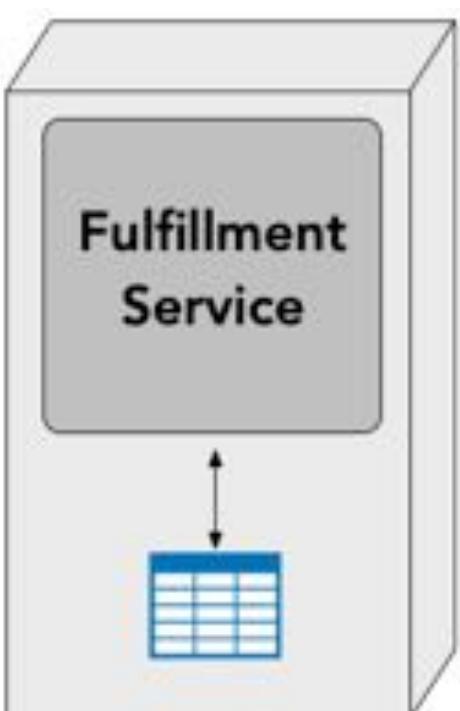
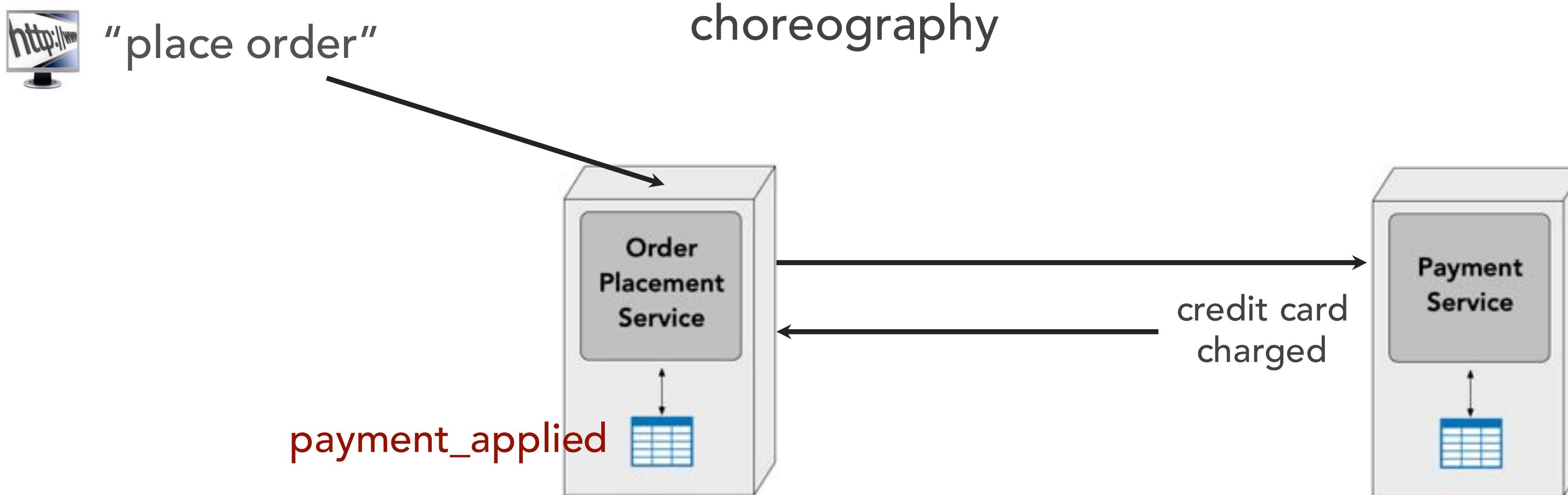
workflow patterns



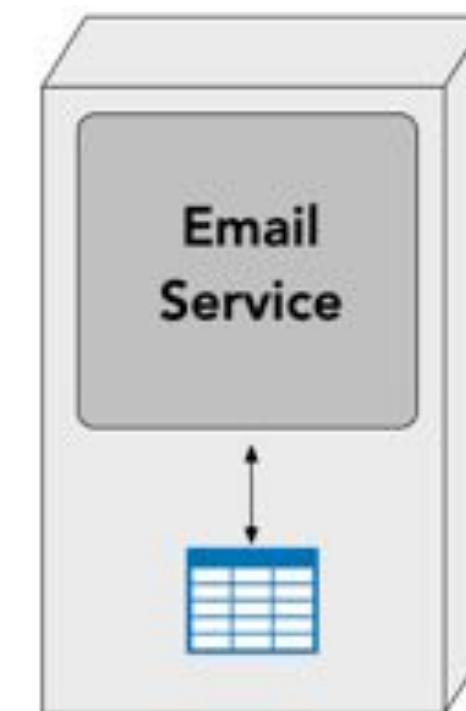
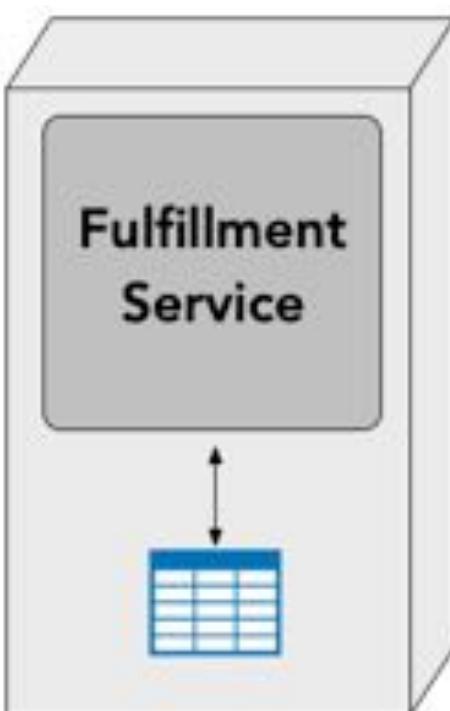
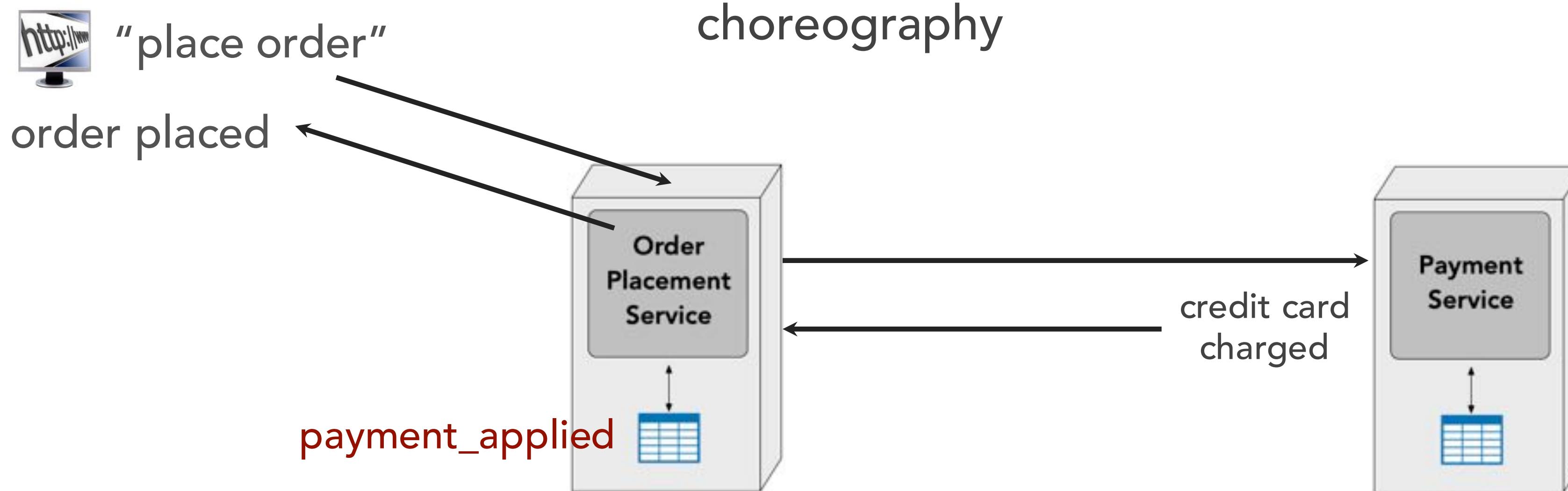
workflow patterns



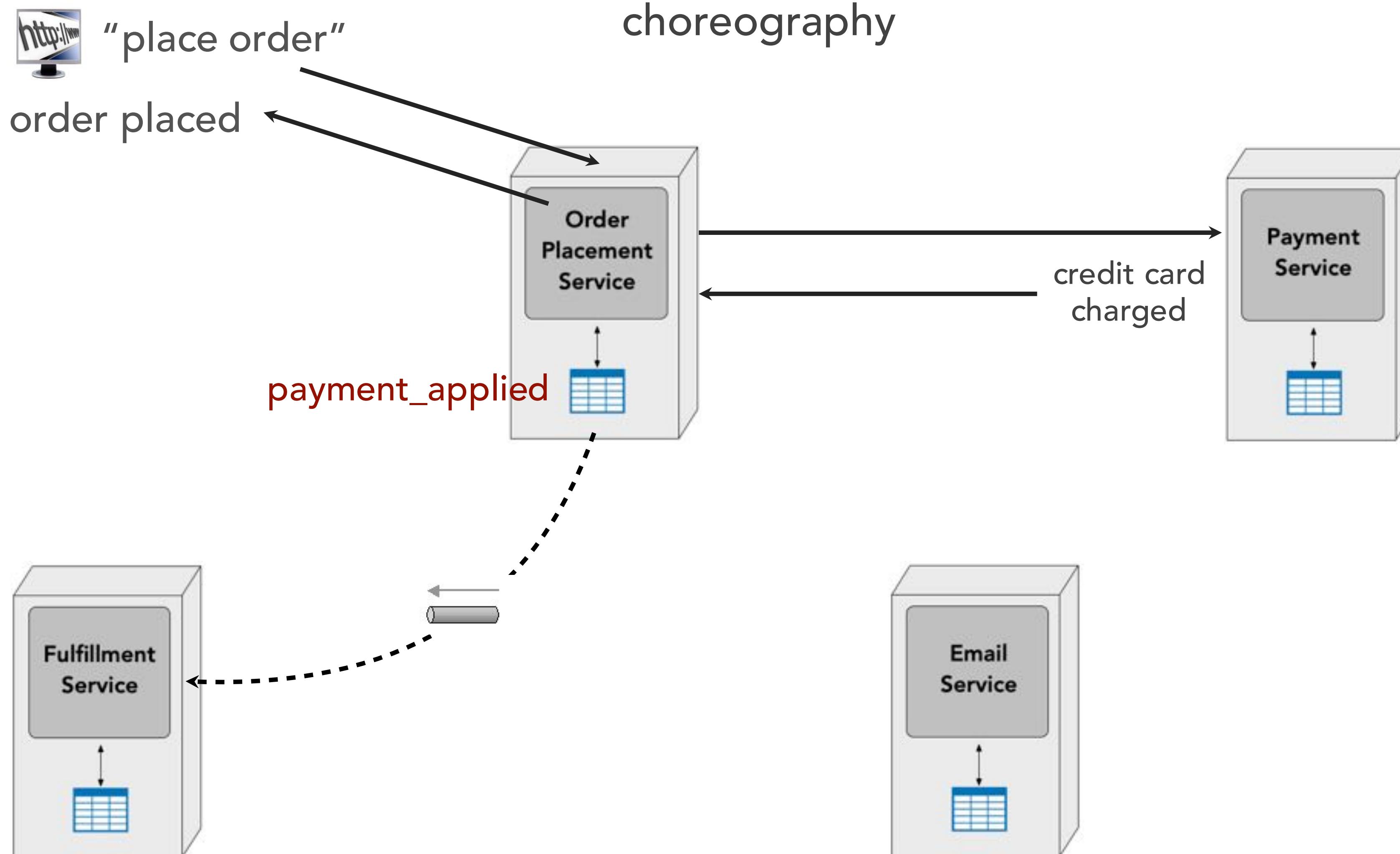
workflow patterns



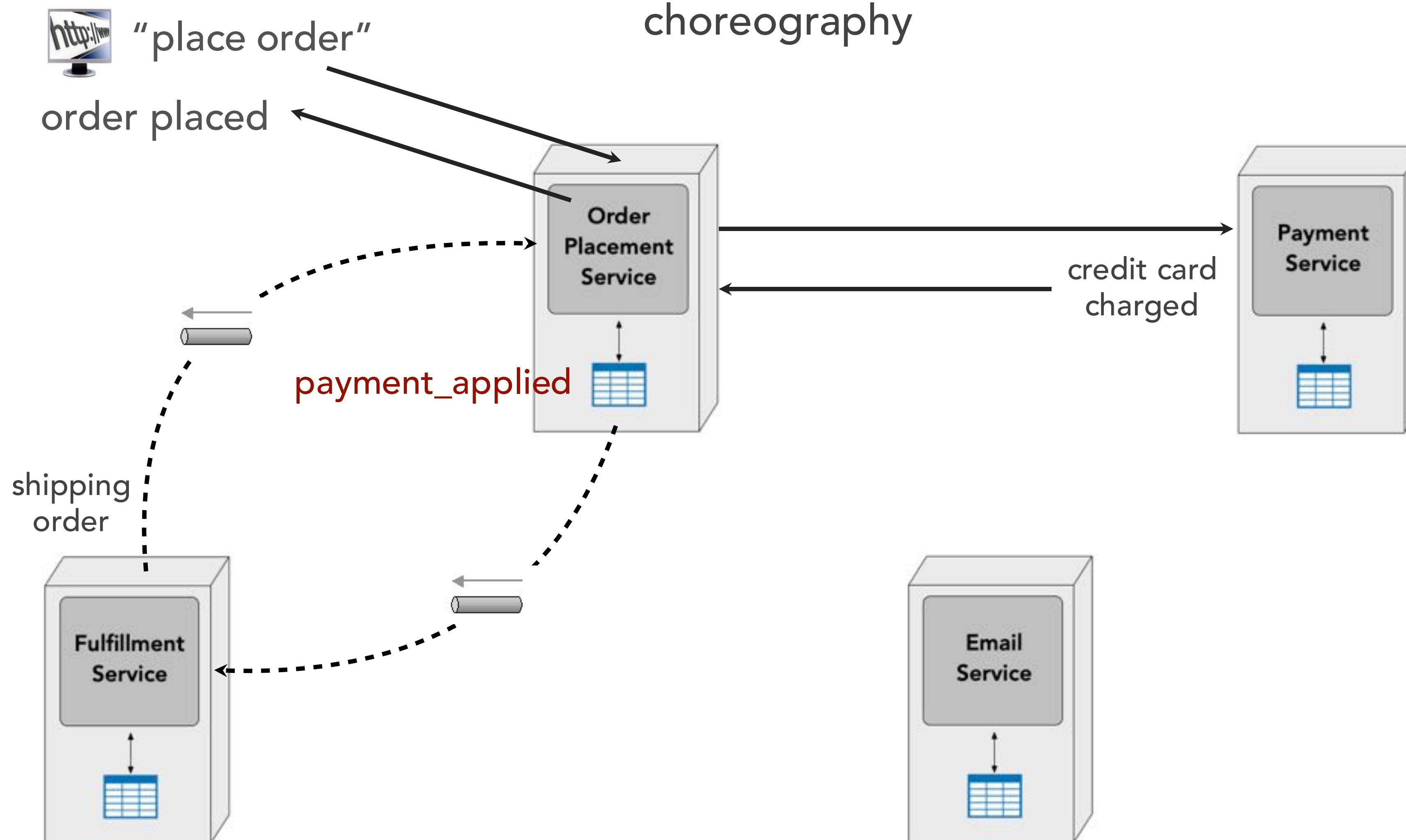
workflow patterns



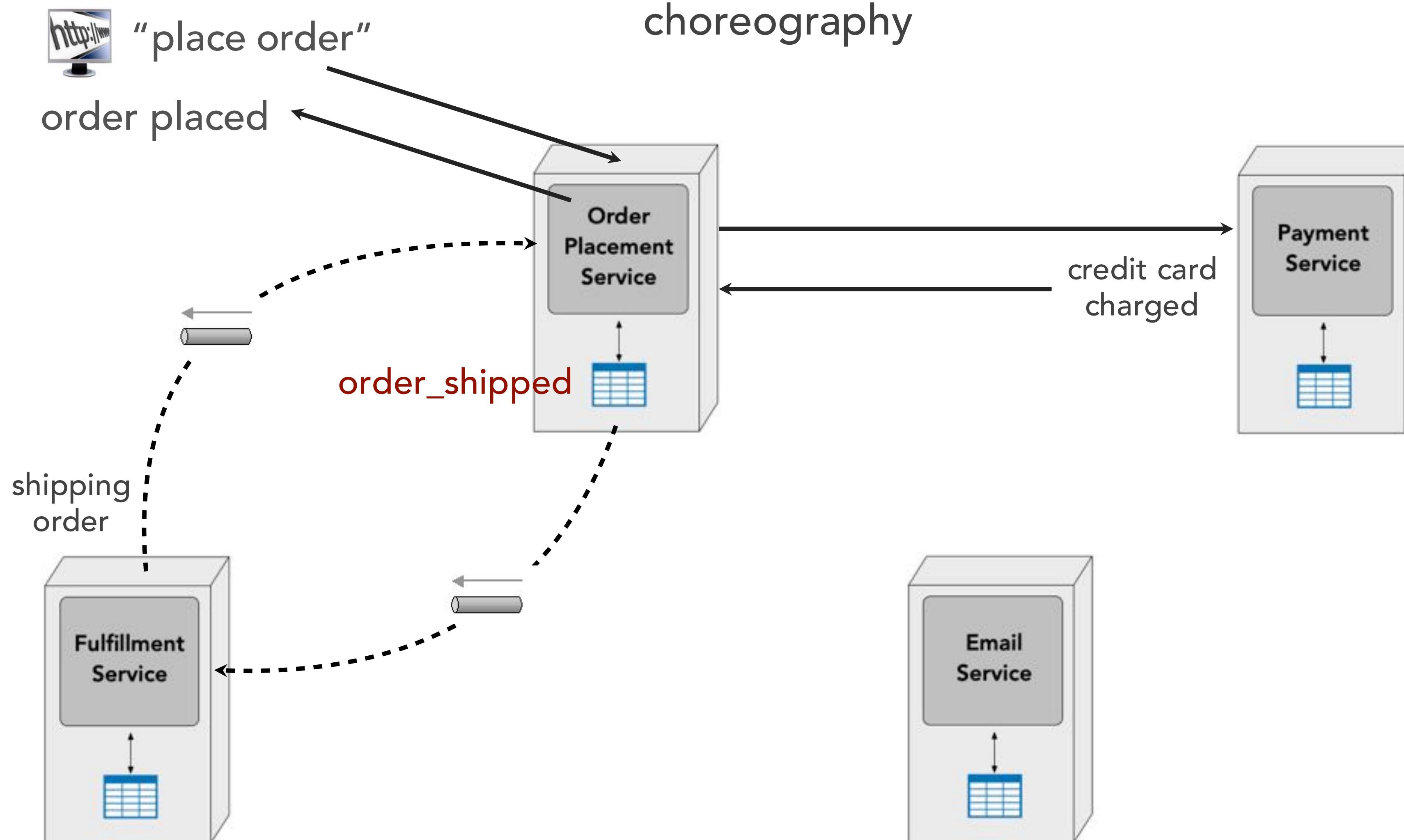
workflow patterns



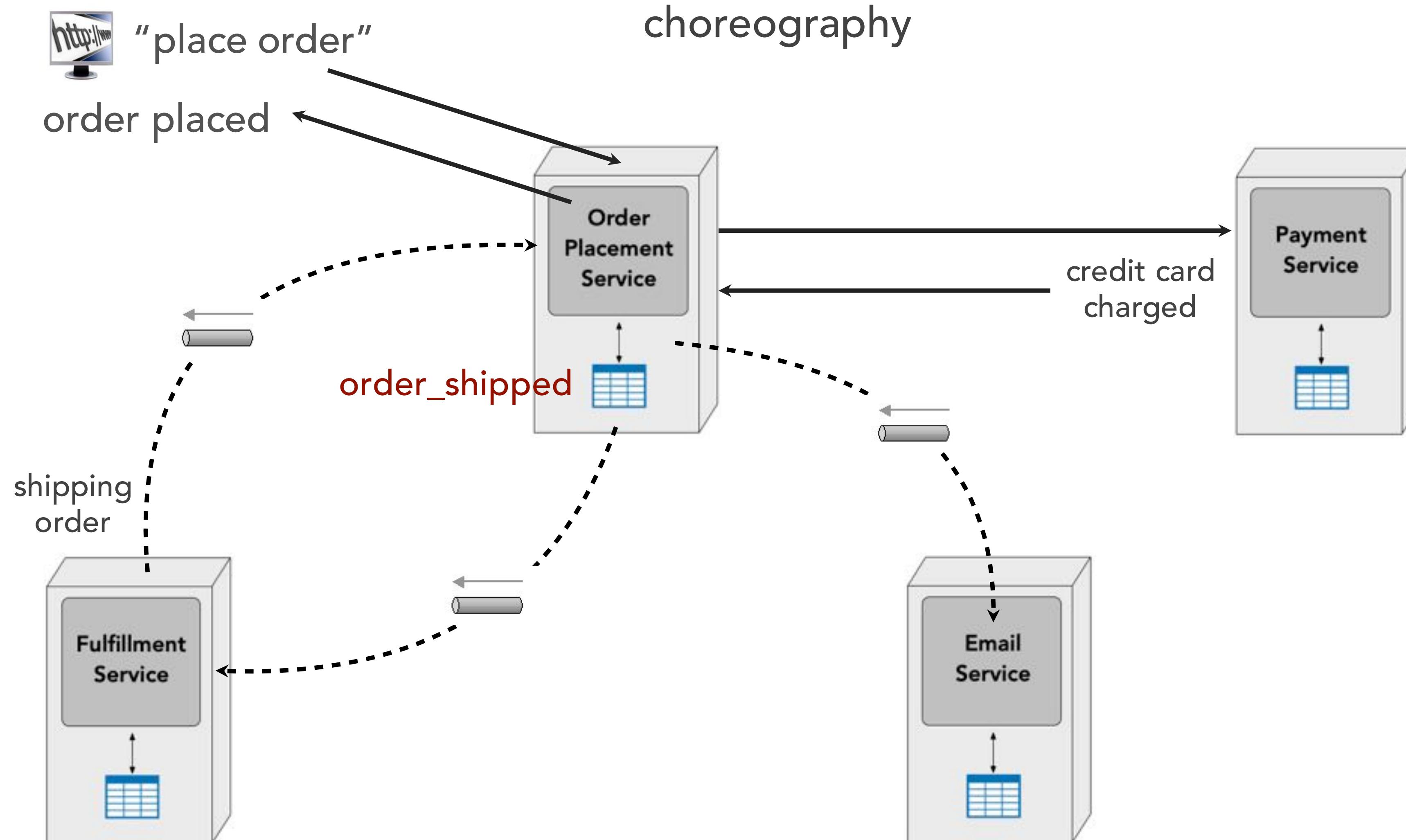
workflow patterns



workflow patterns

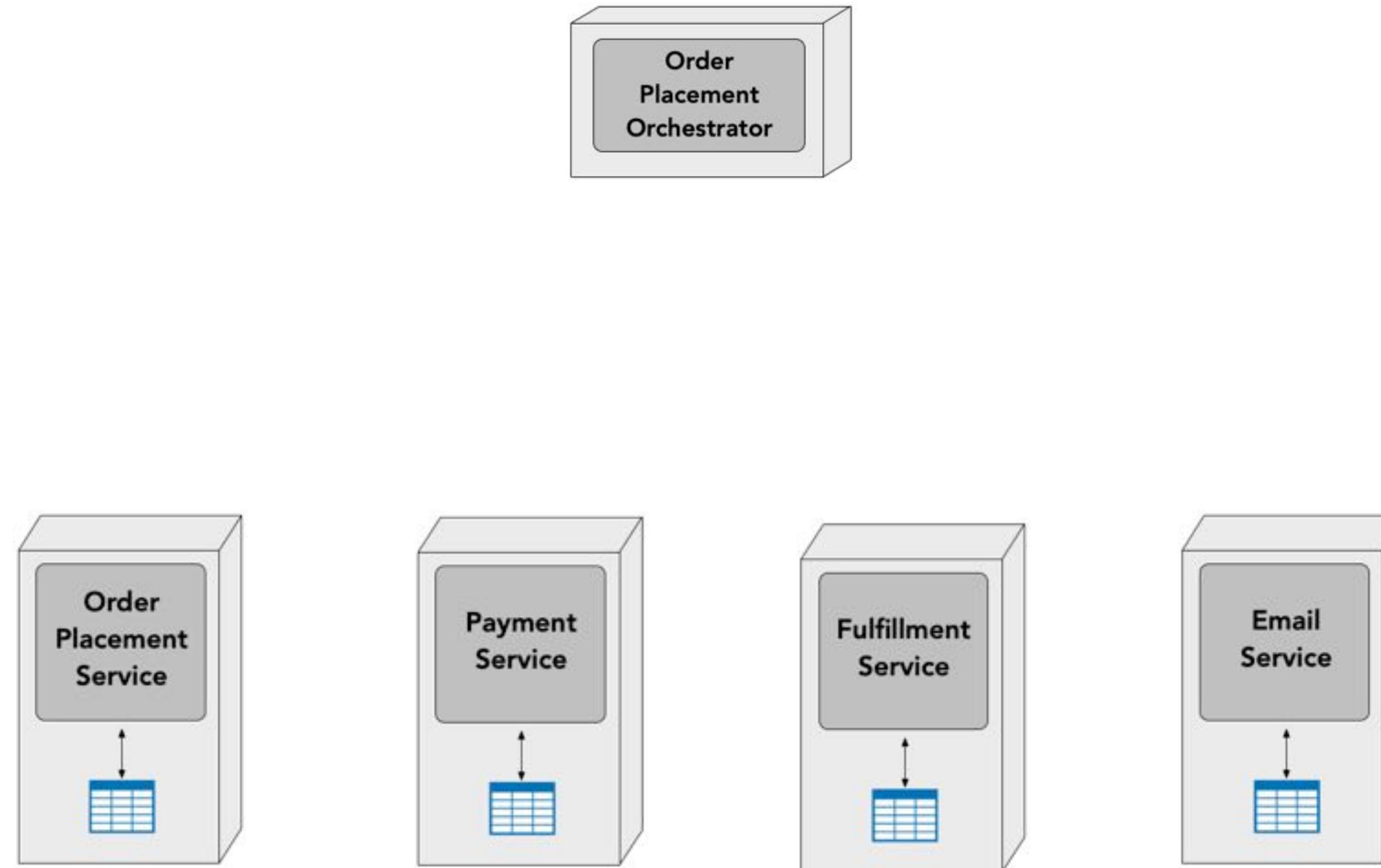


workflow patterns



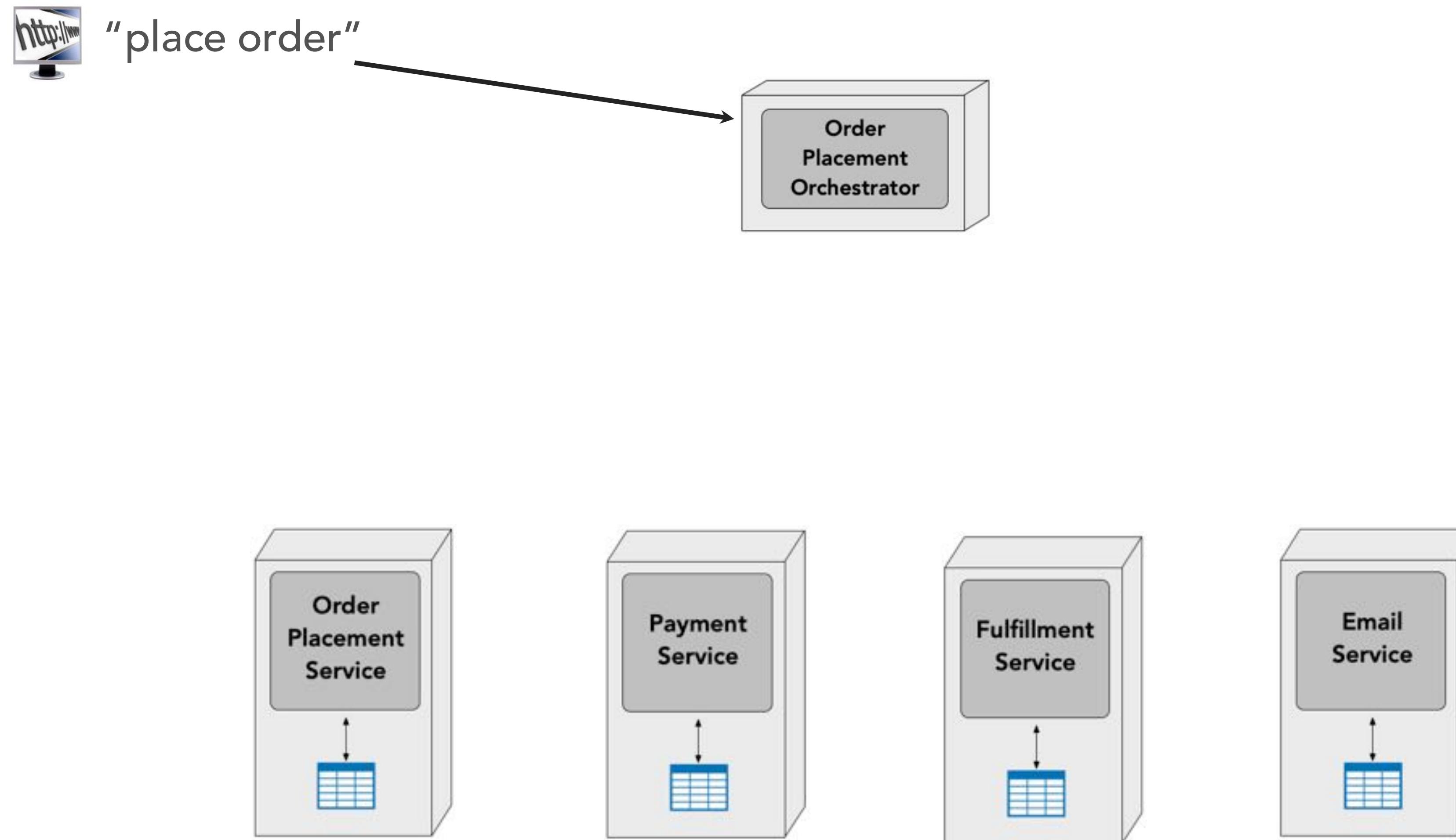
workflow patterns

stateless orchestration



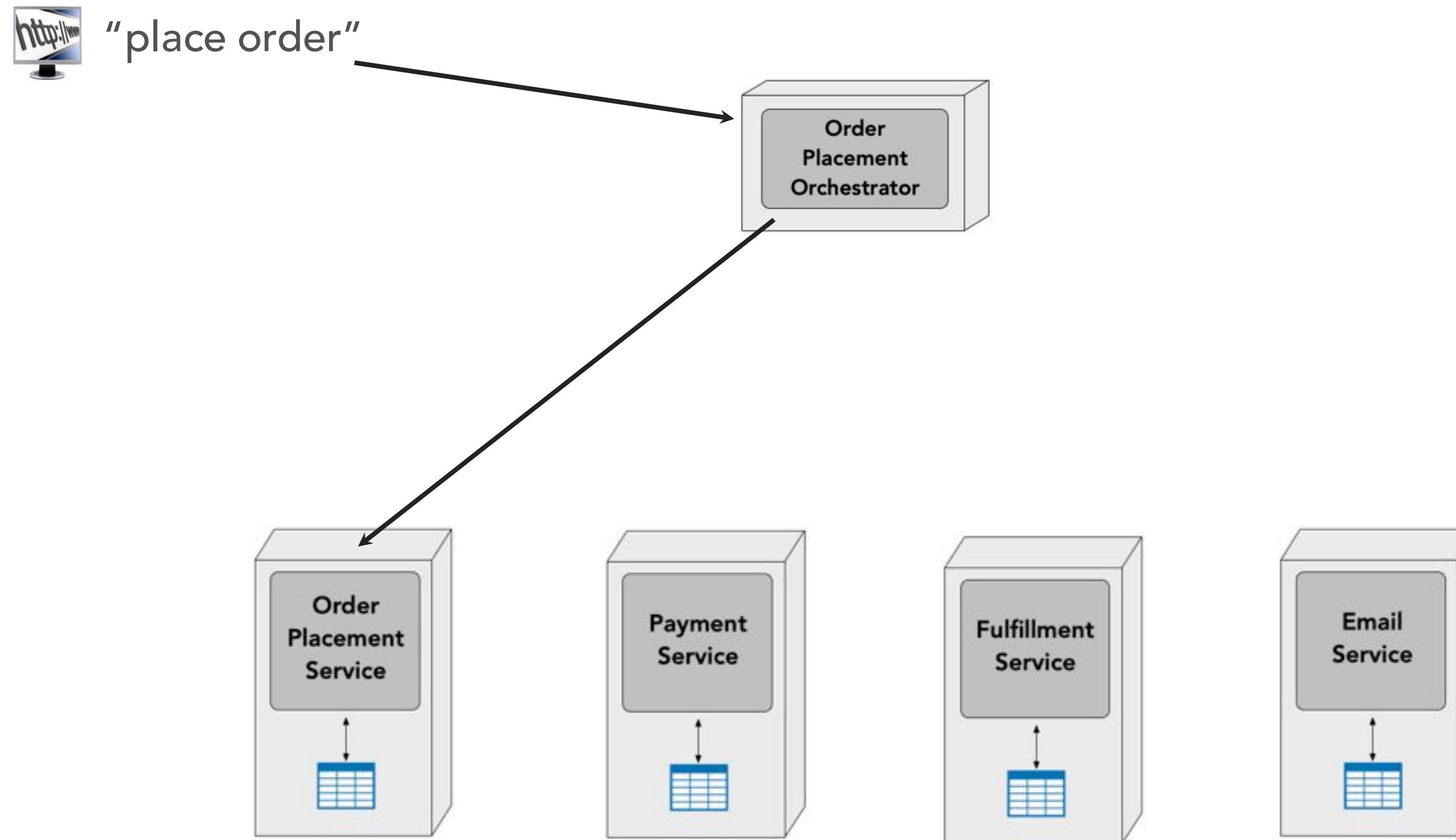
workflow patterns

stateless orchestration



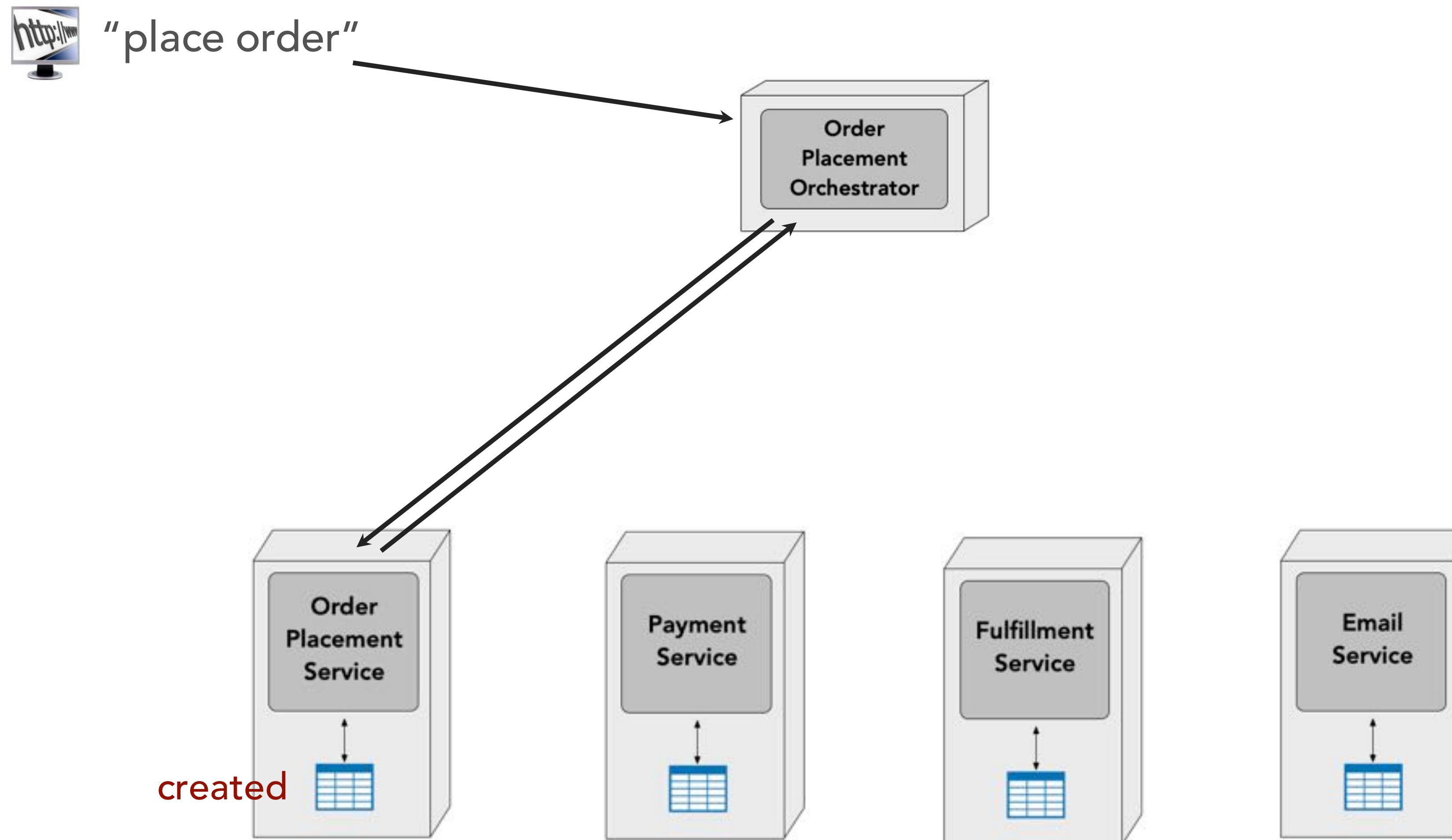
workflow patterns

stateless orchestration



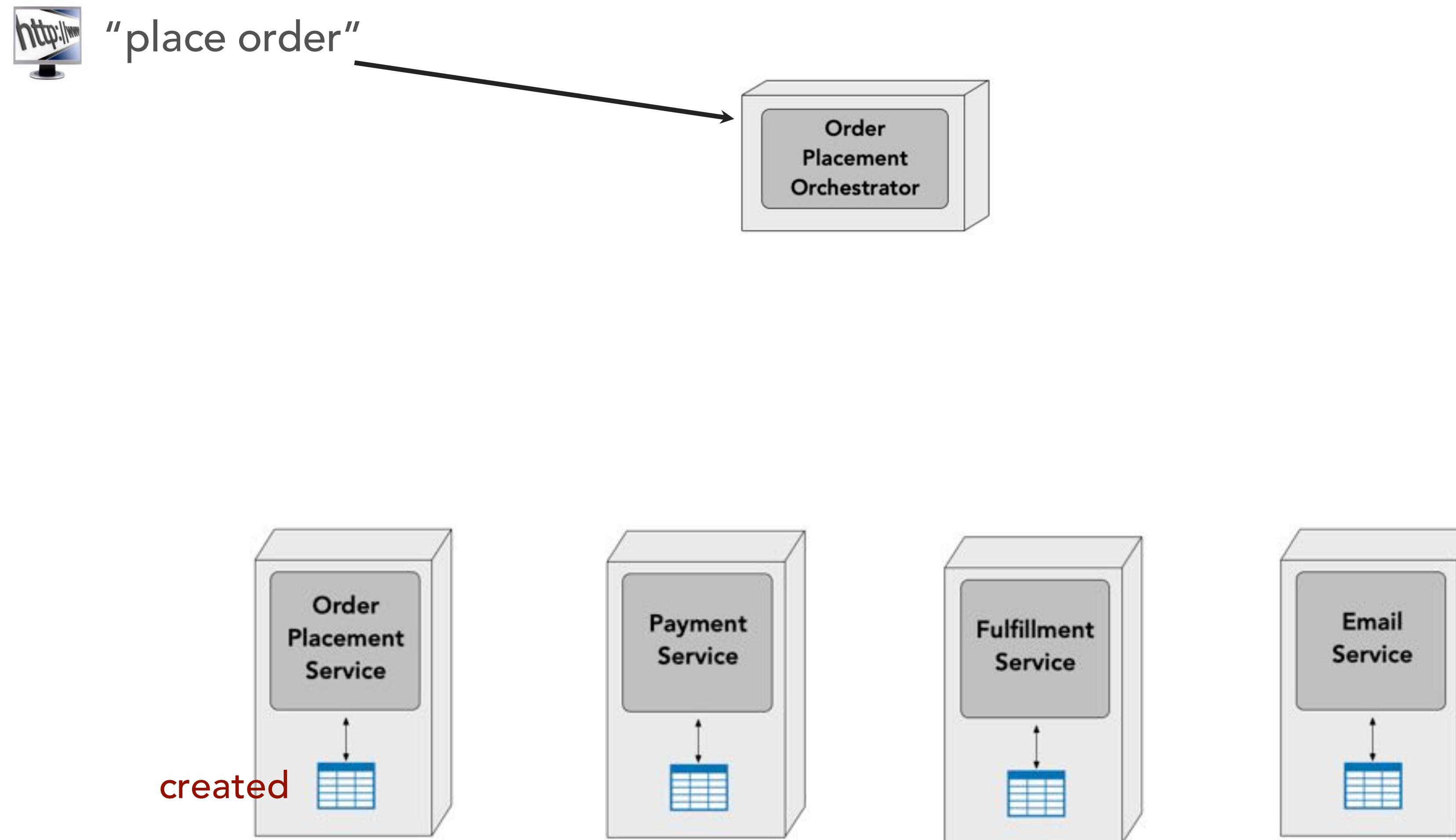
workflow patterns

stateless orchestration



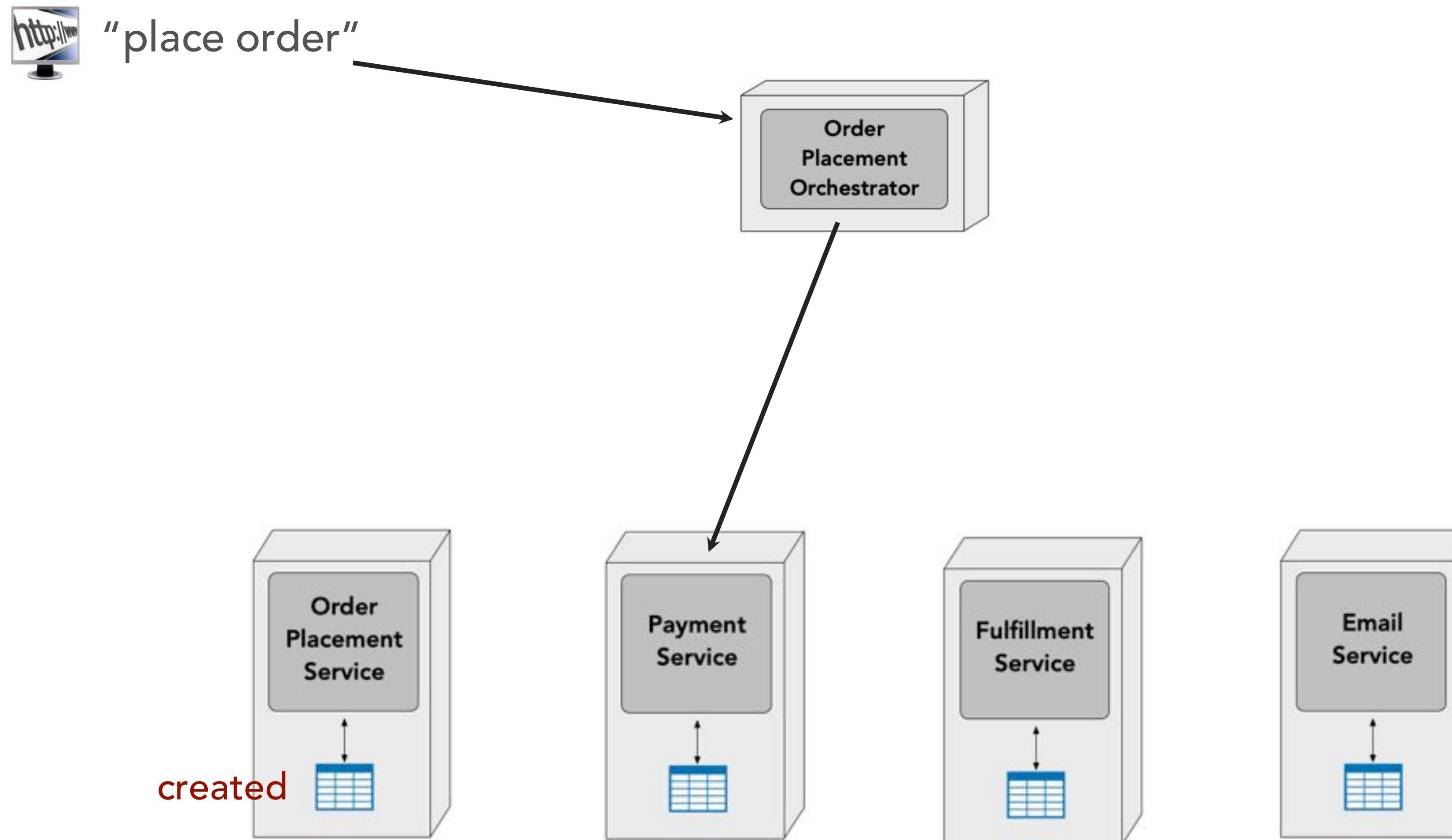
workflow patterns

stateless orchestration



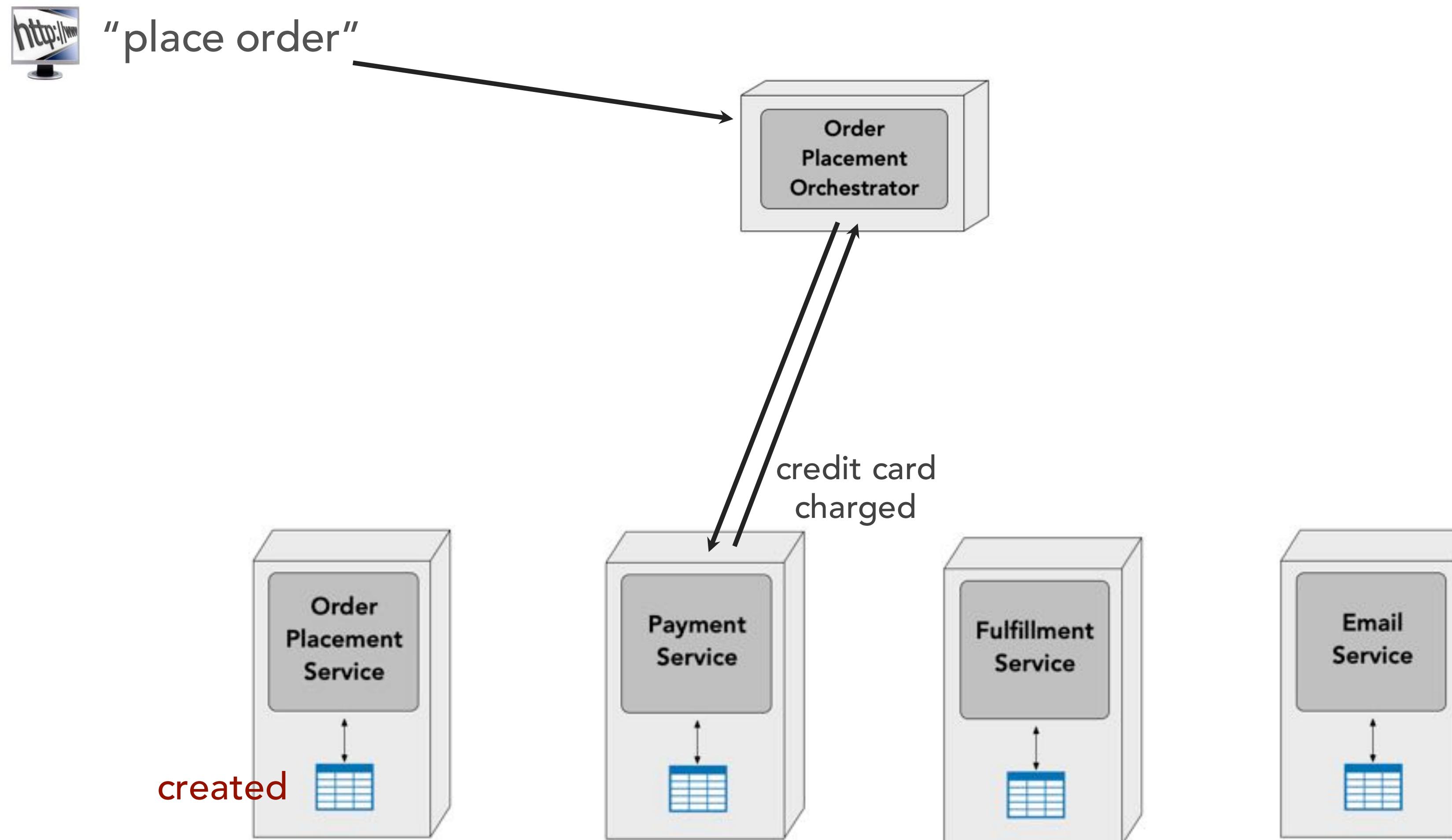
workflow patterns

stateless orchestration



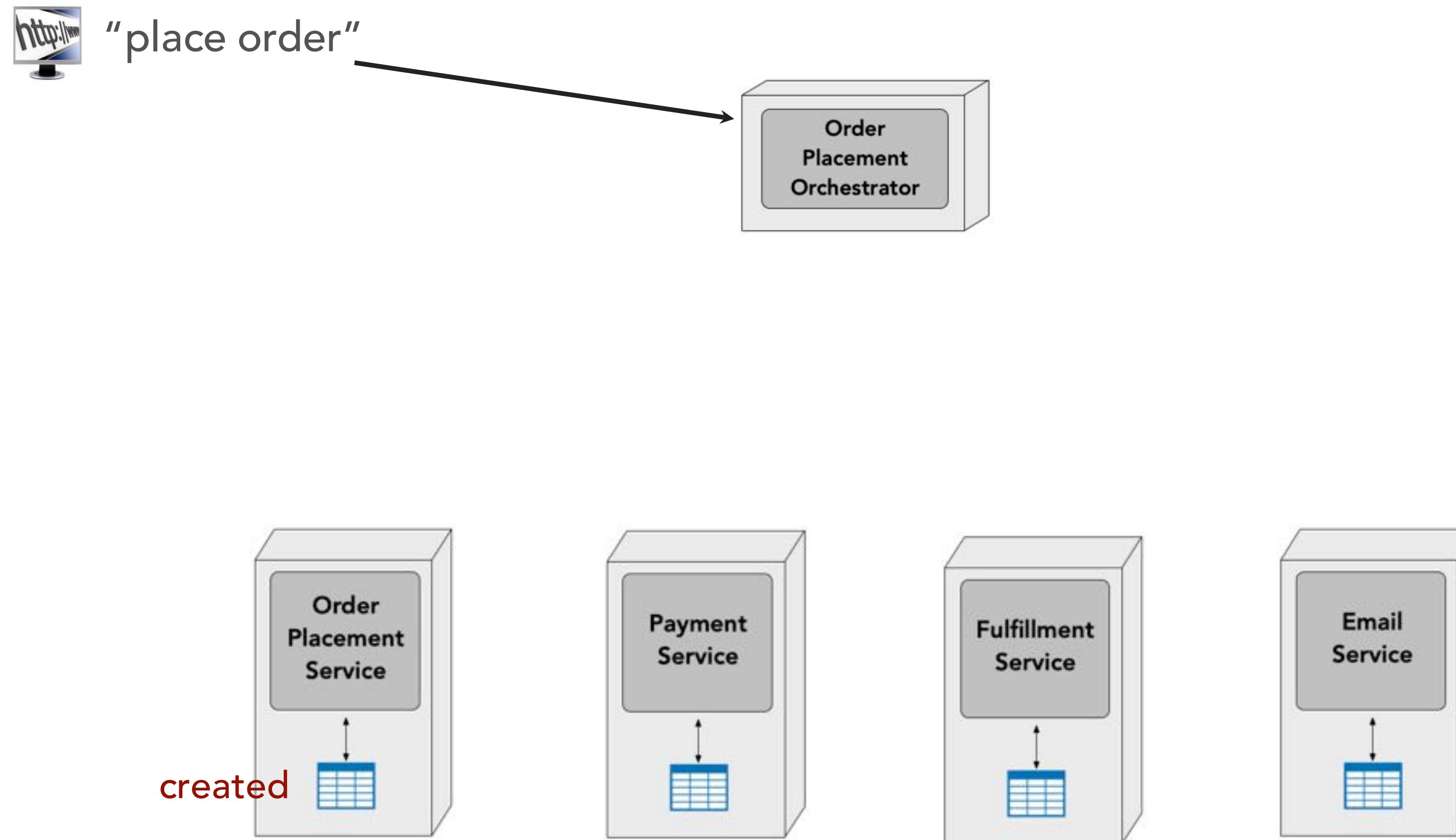
workflow patterns

stateless orchestration



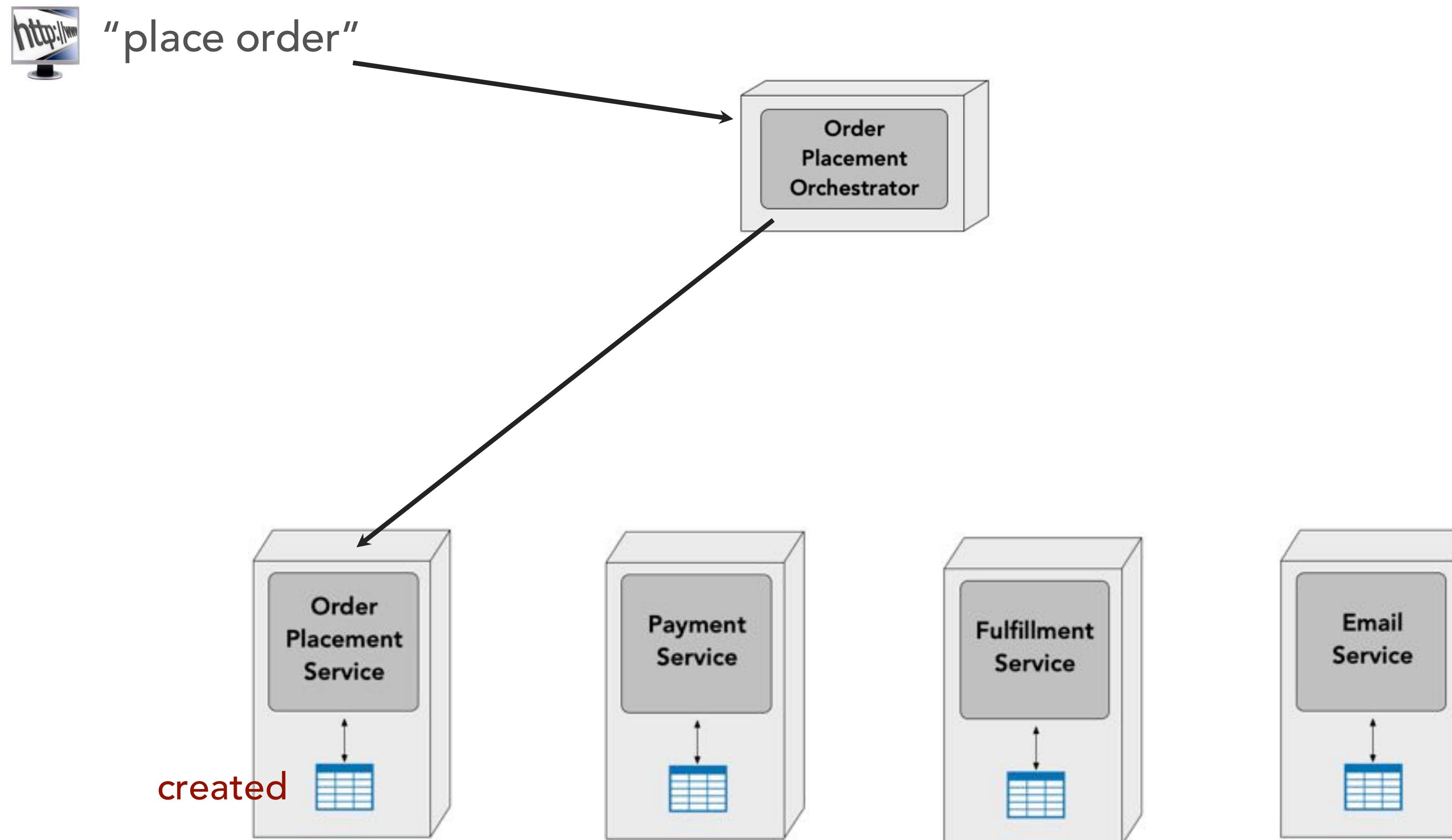
workflow patterns

stateless orchestration



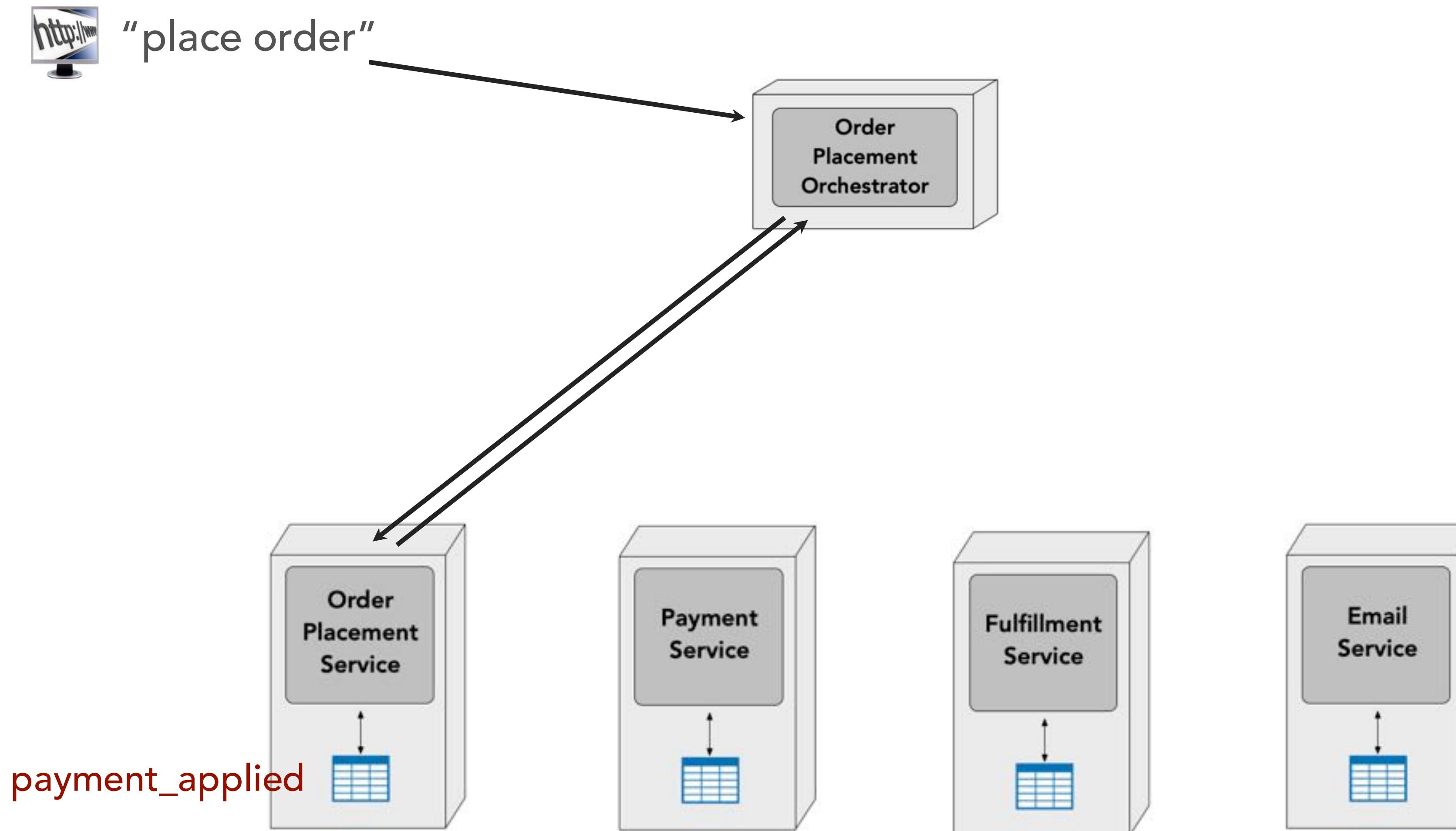
workflow patterns

stateless orchestration



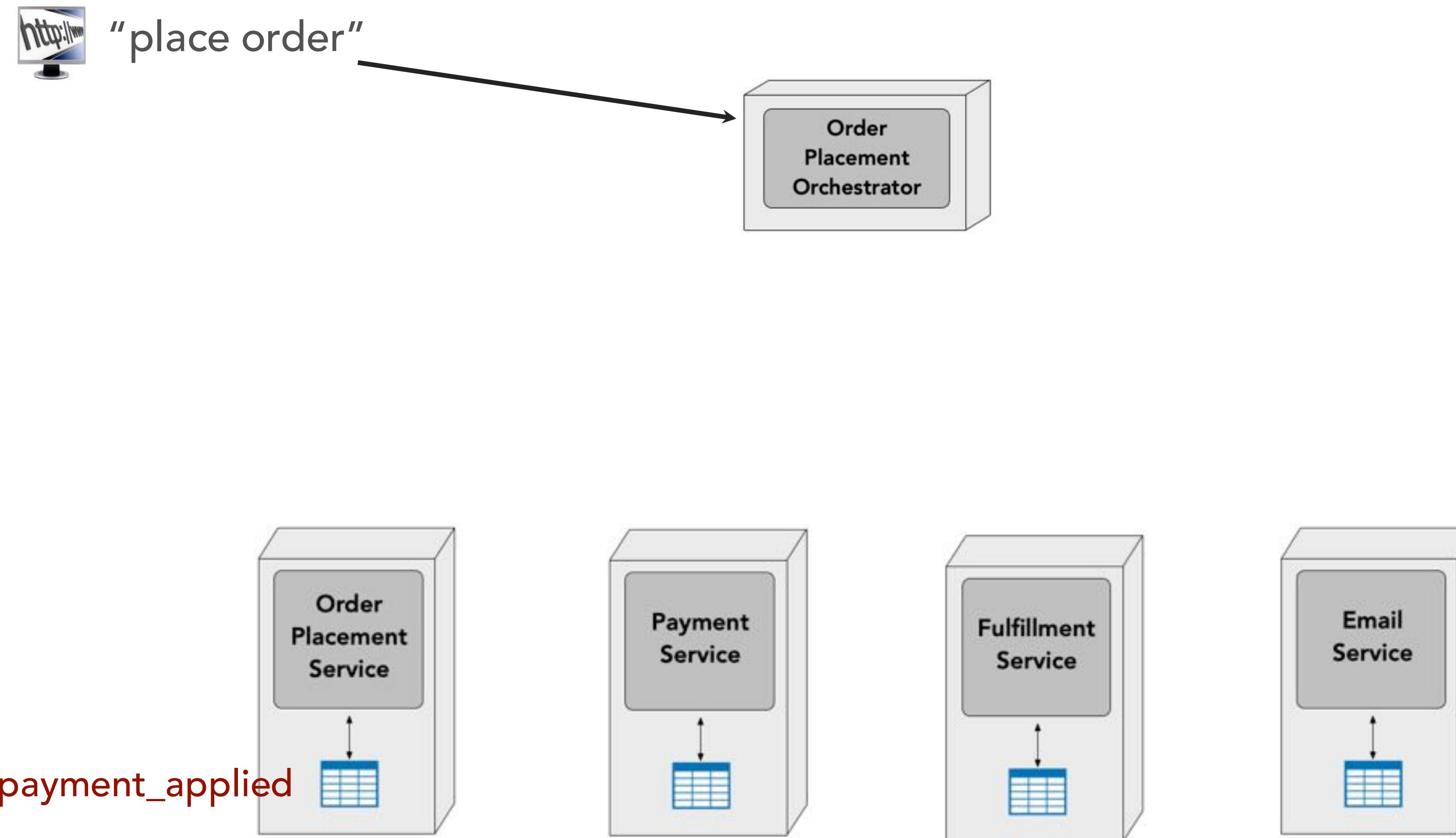
workflow patterns

stateless orchestration



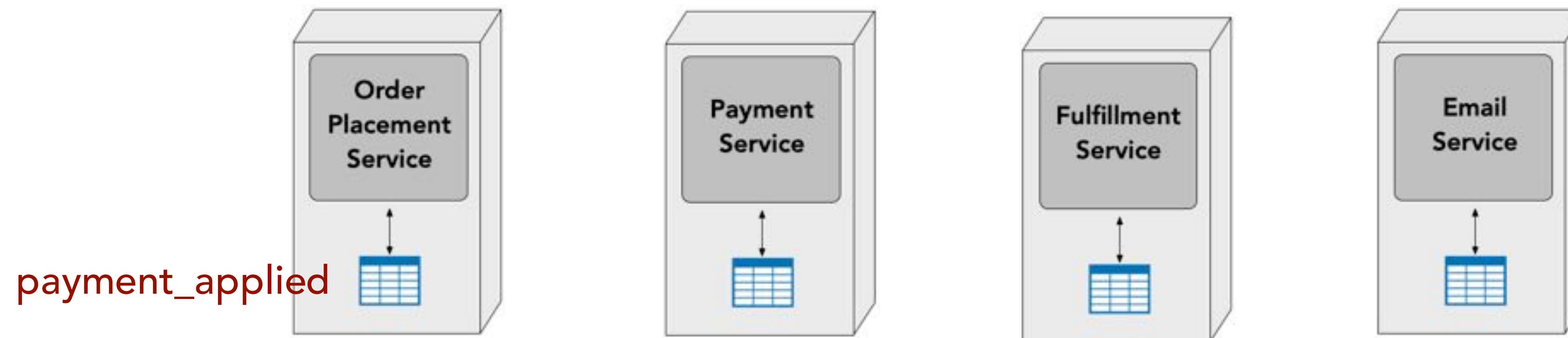
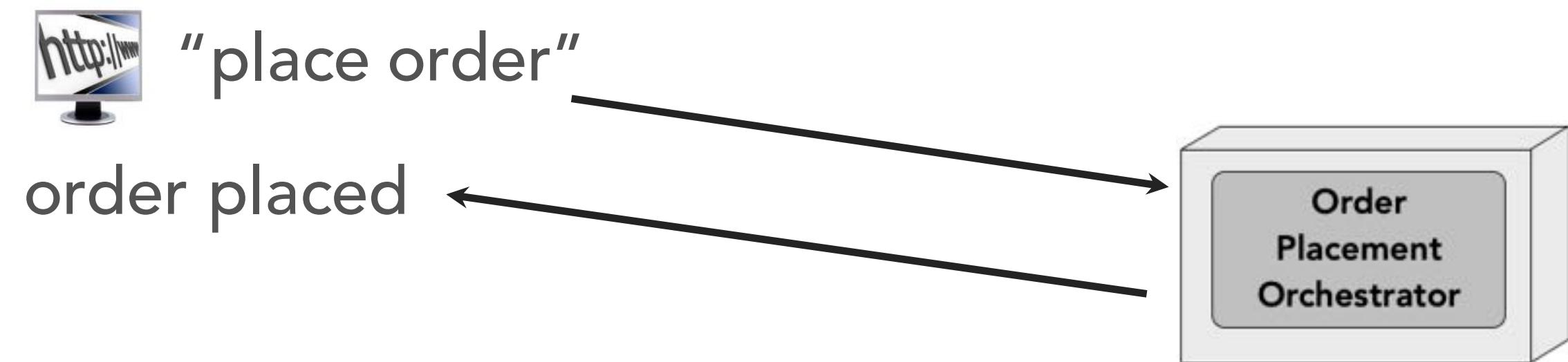
workflow patterns

stateless orchestration



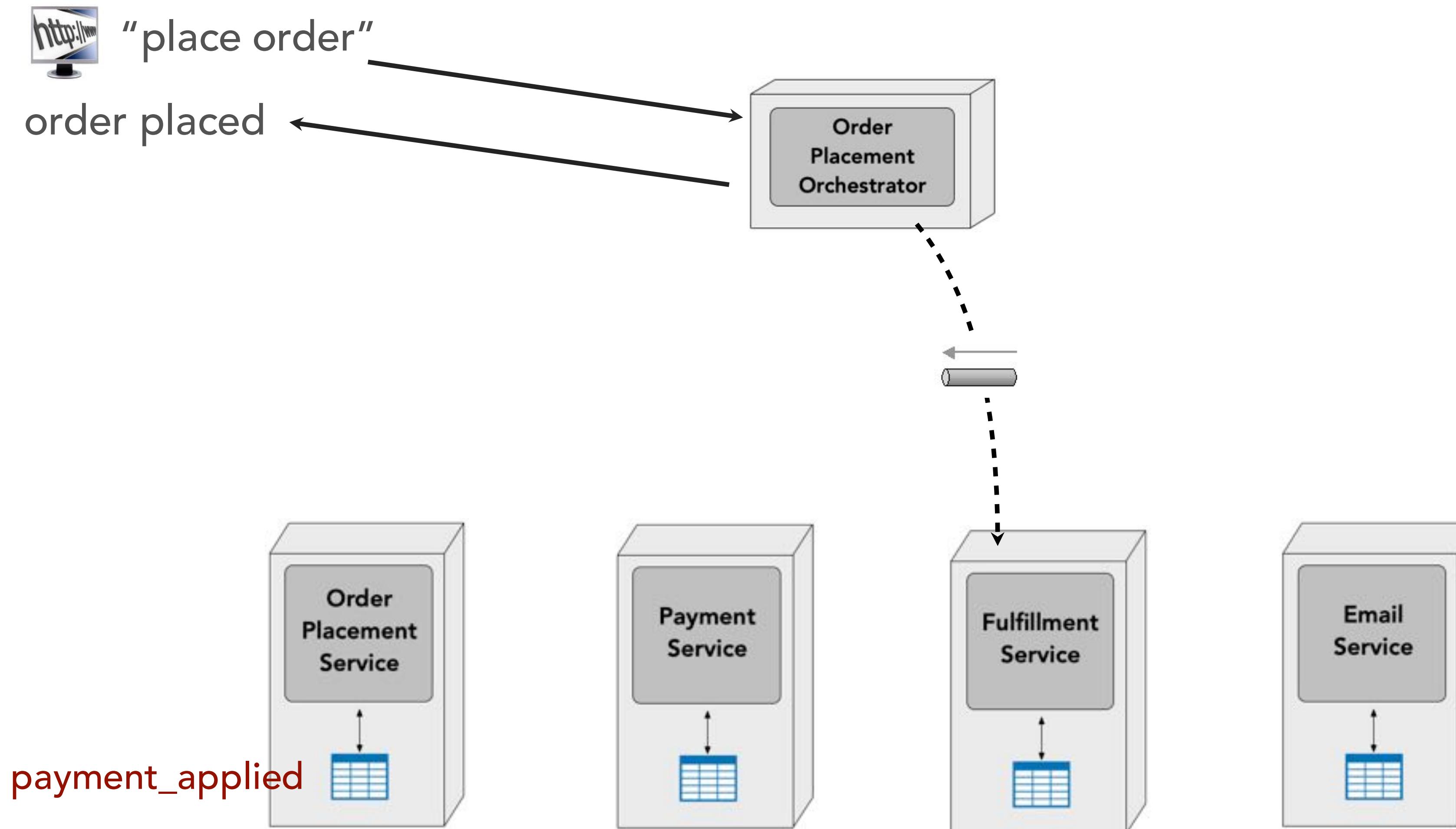
workflow patterns

stateless orchestration



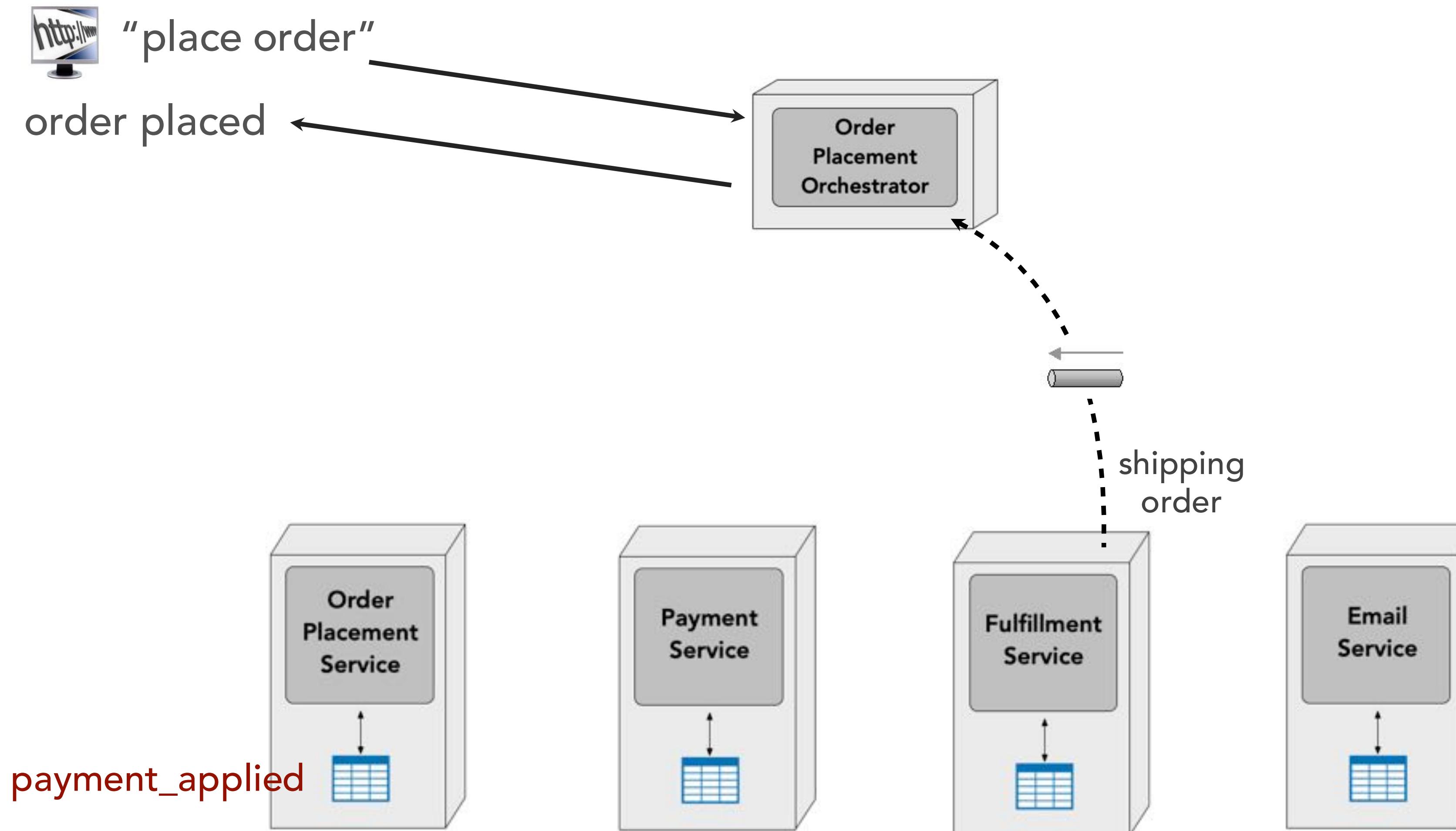
workflow patterns

stateless orchestration



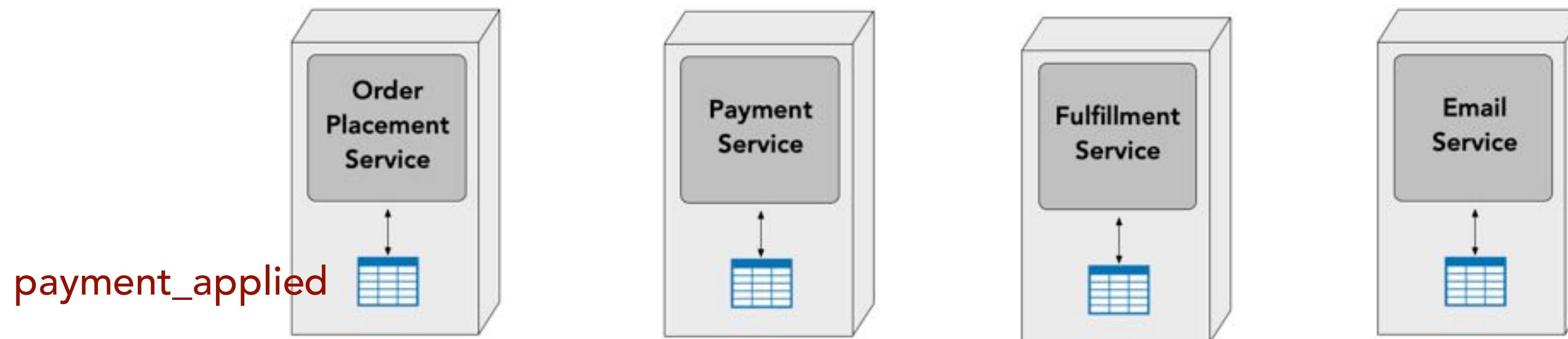
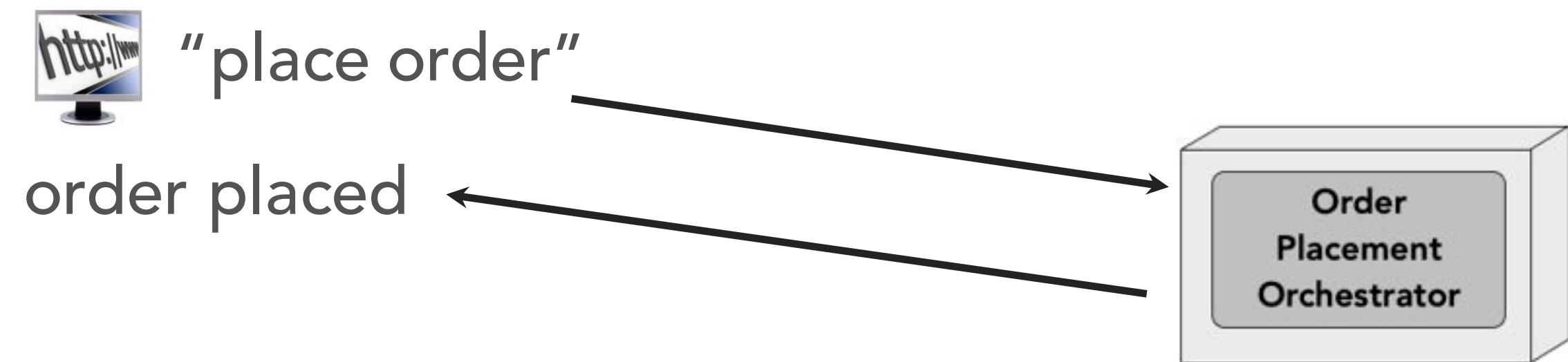
workflow patterns

stateless orchestration



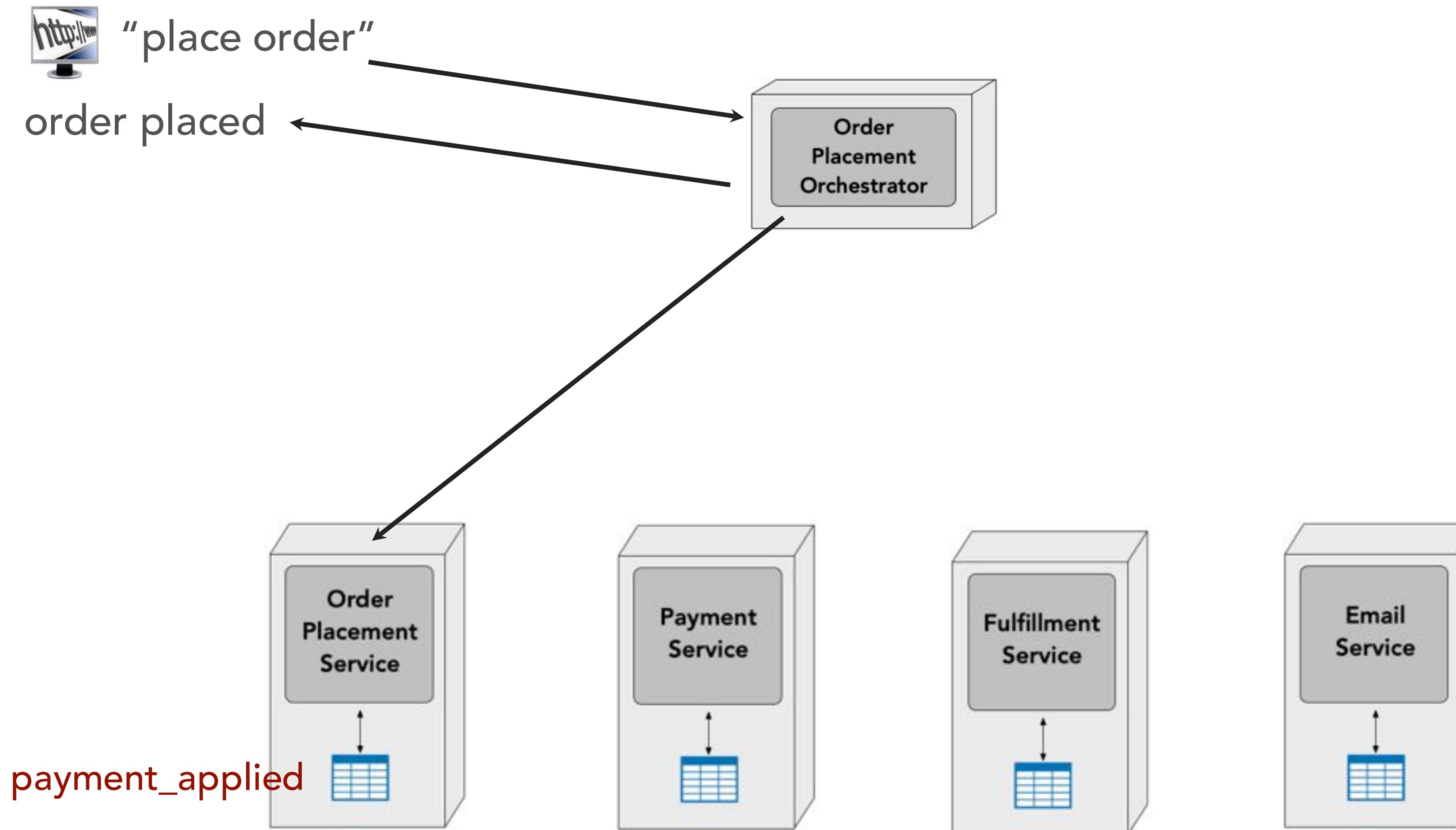
workflow patterns

stateless orchestration



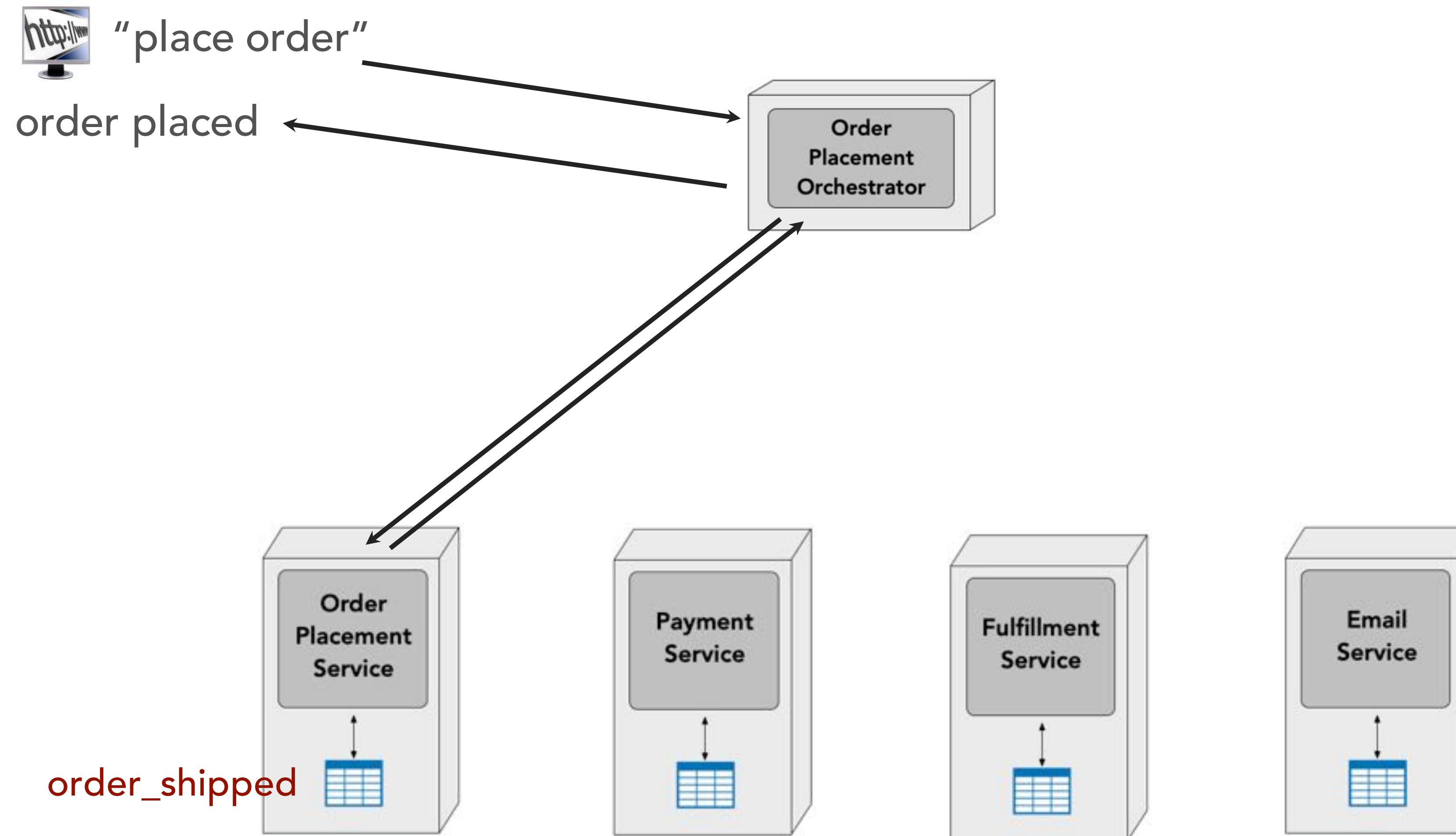
workflow patterns

stateless orchestration



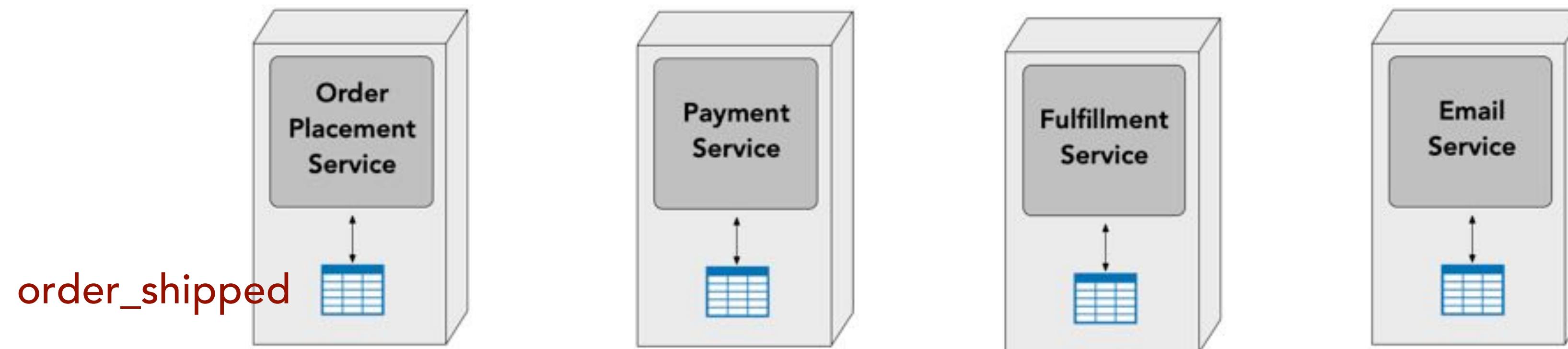
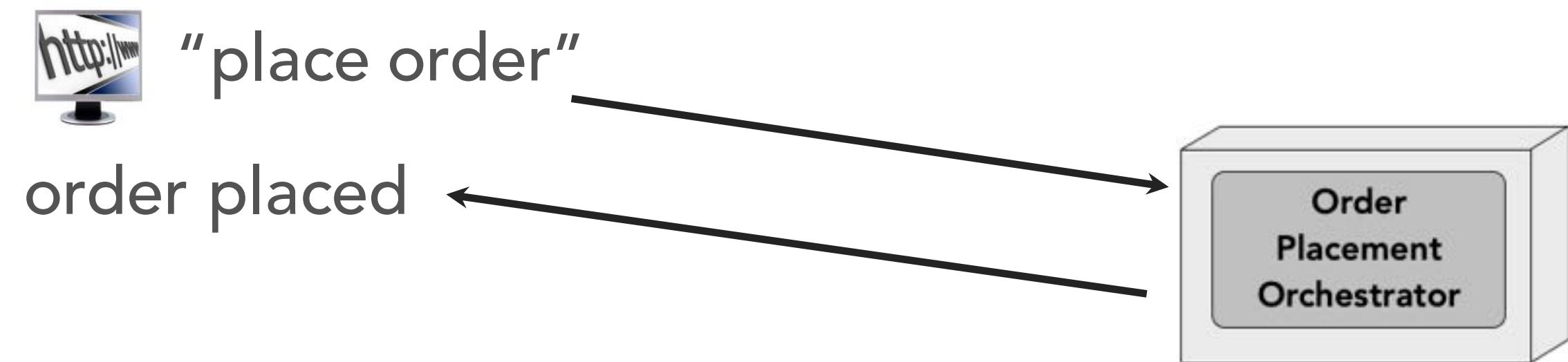
workflow patterns

stateless orchestration



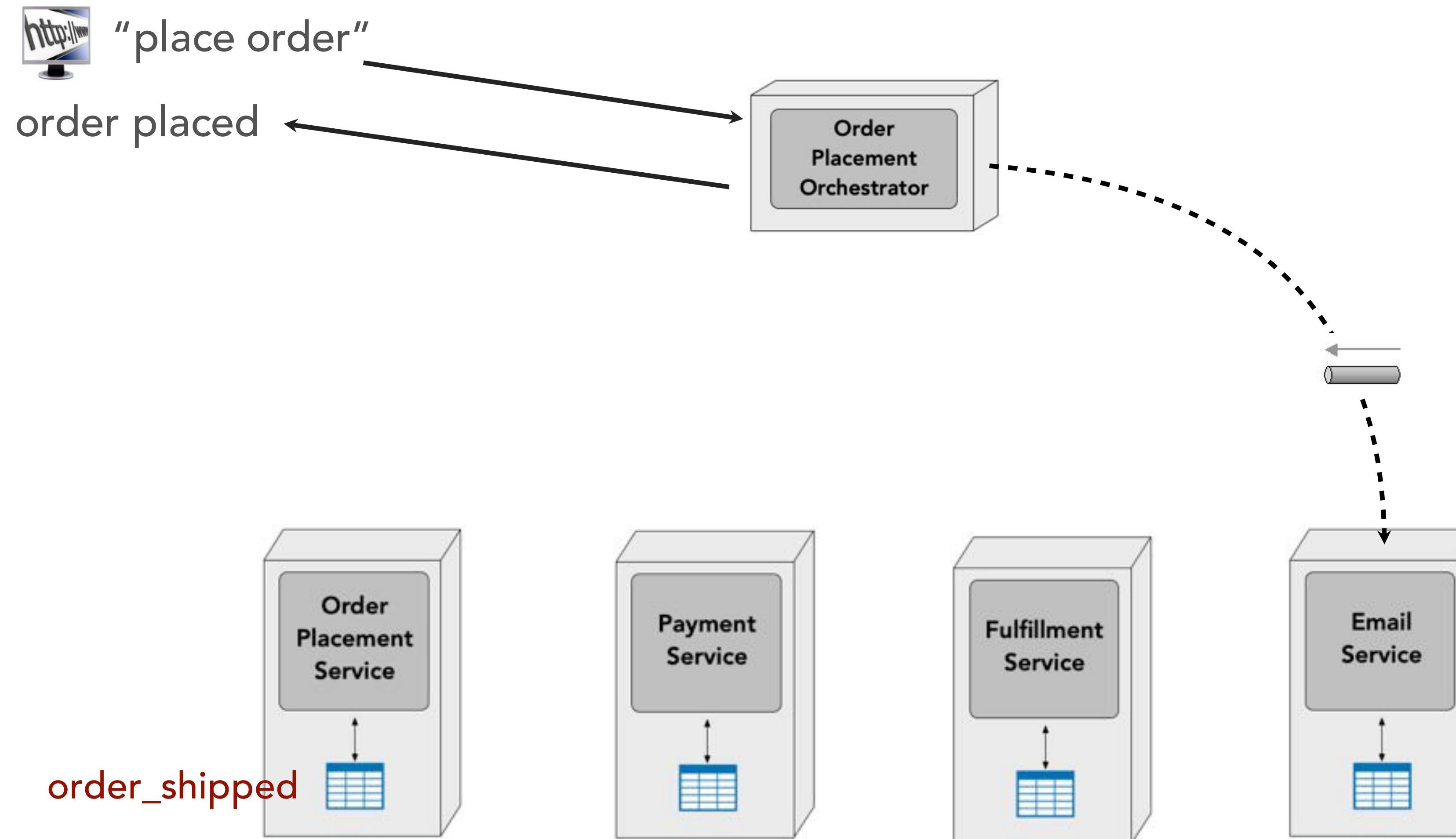
workflow patterns

stateless orchestration



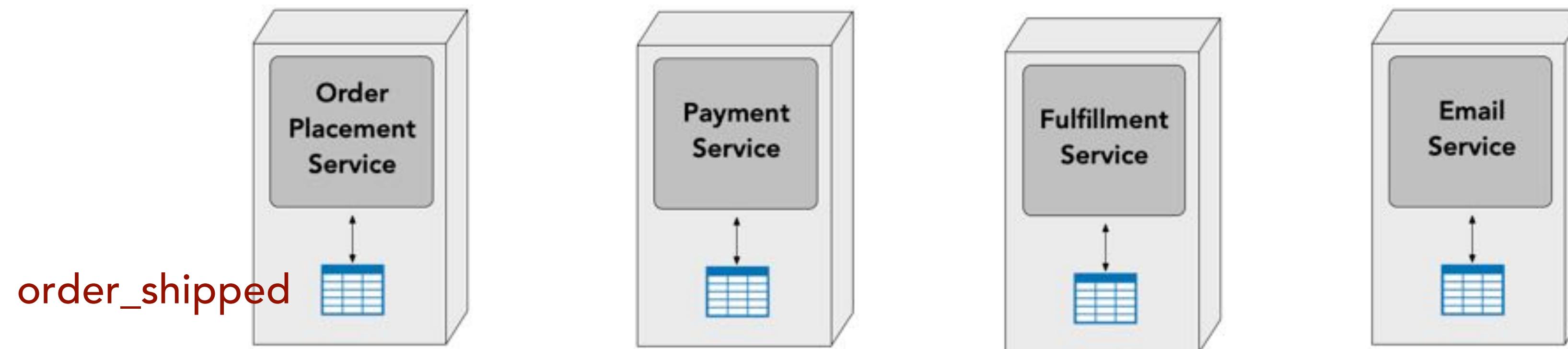
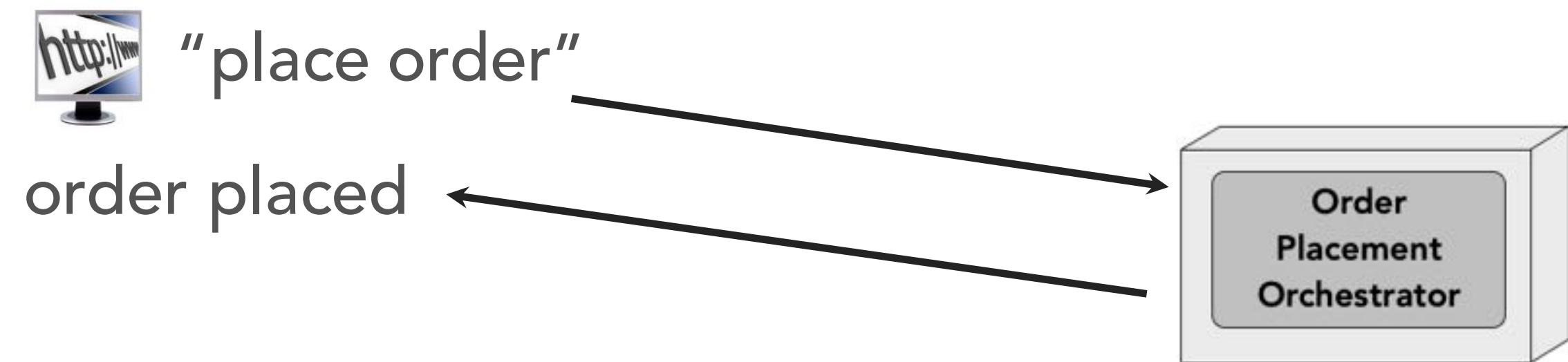
workflow patterns

stateless orchestration



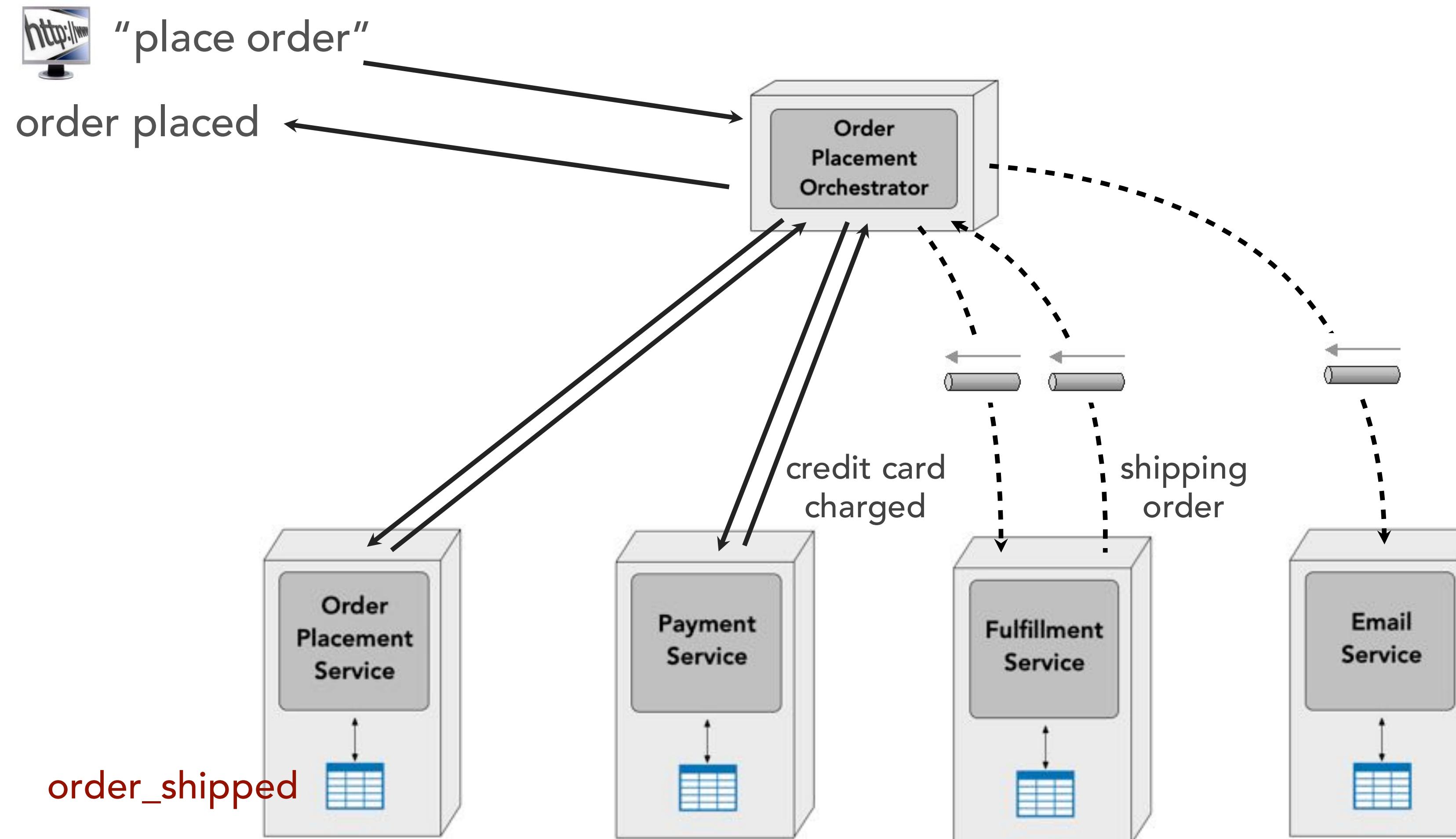
workflow patterns

stateless orchestration



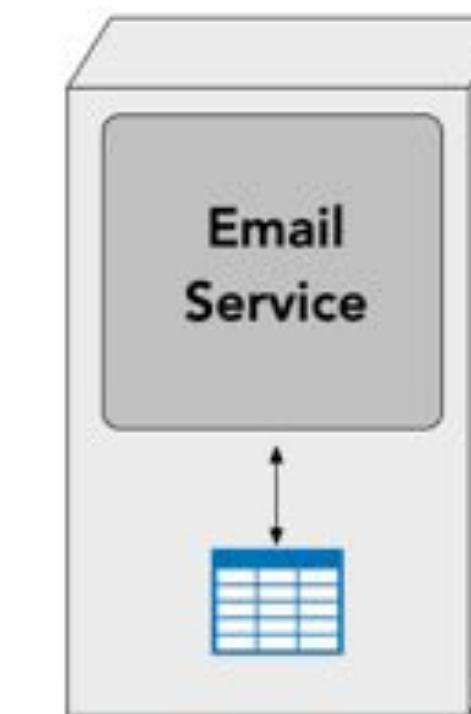
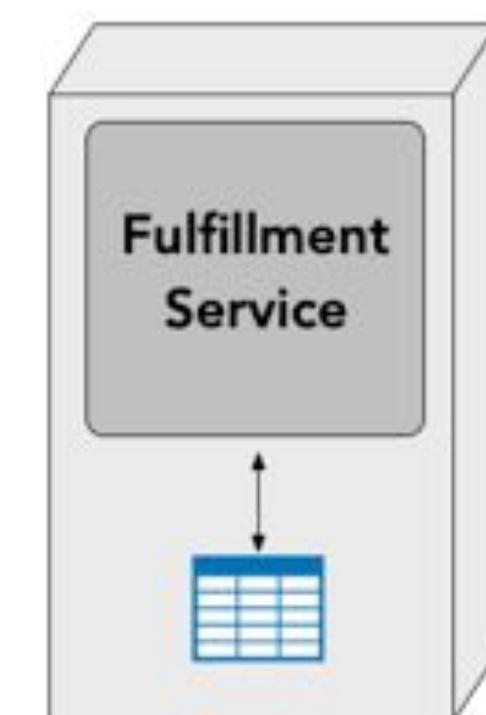
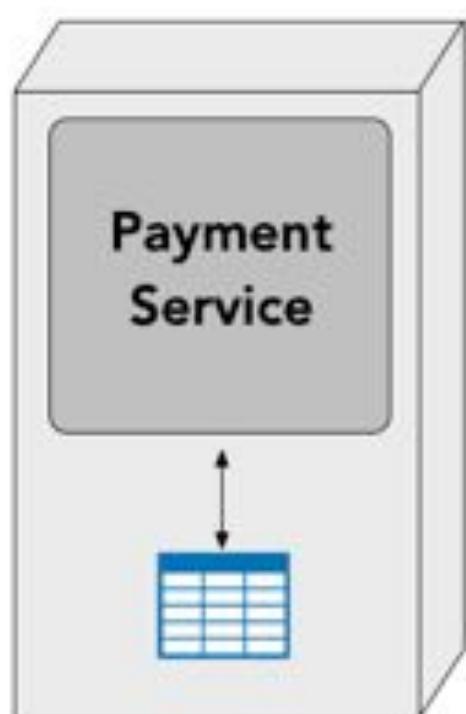
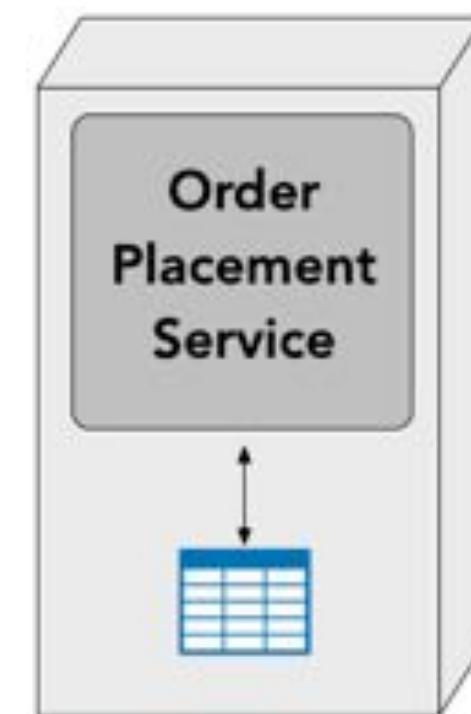
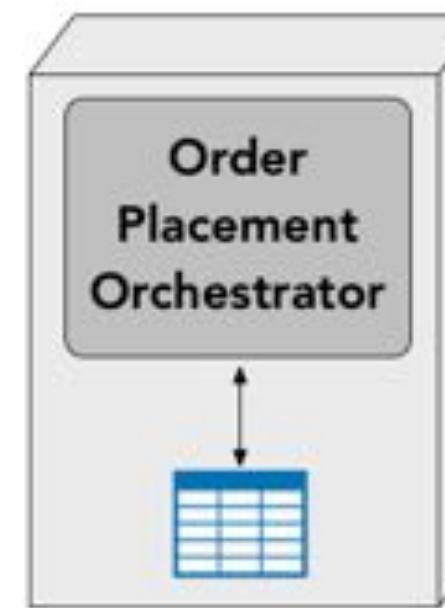
workflow patterns

stateless orchestration



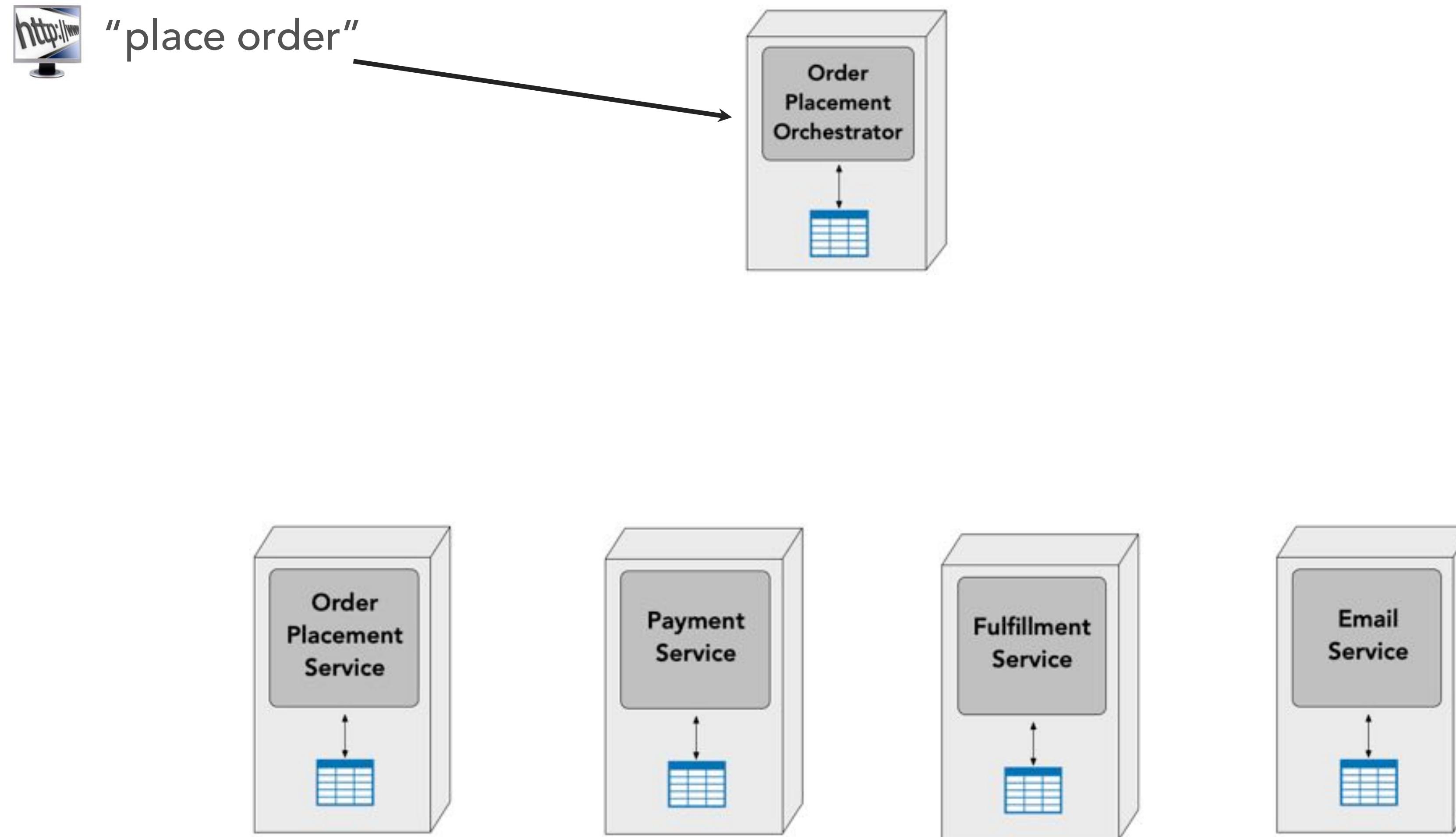
workflow patterns

stateful orchestration



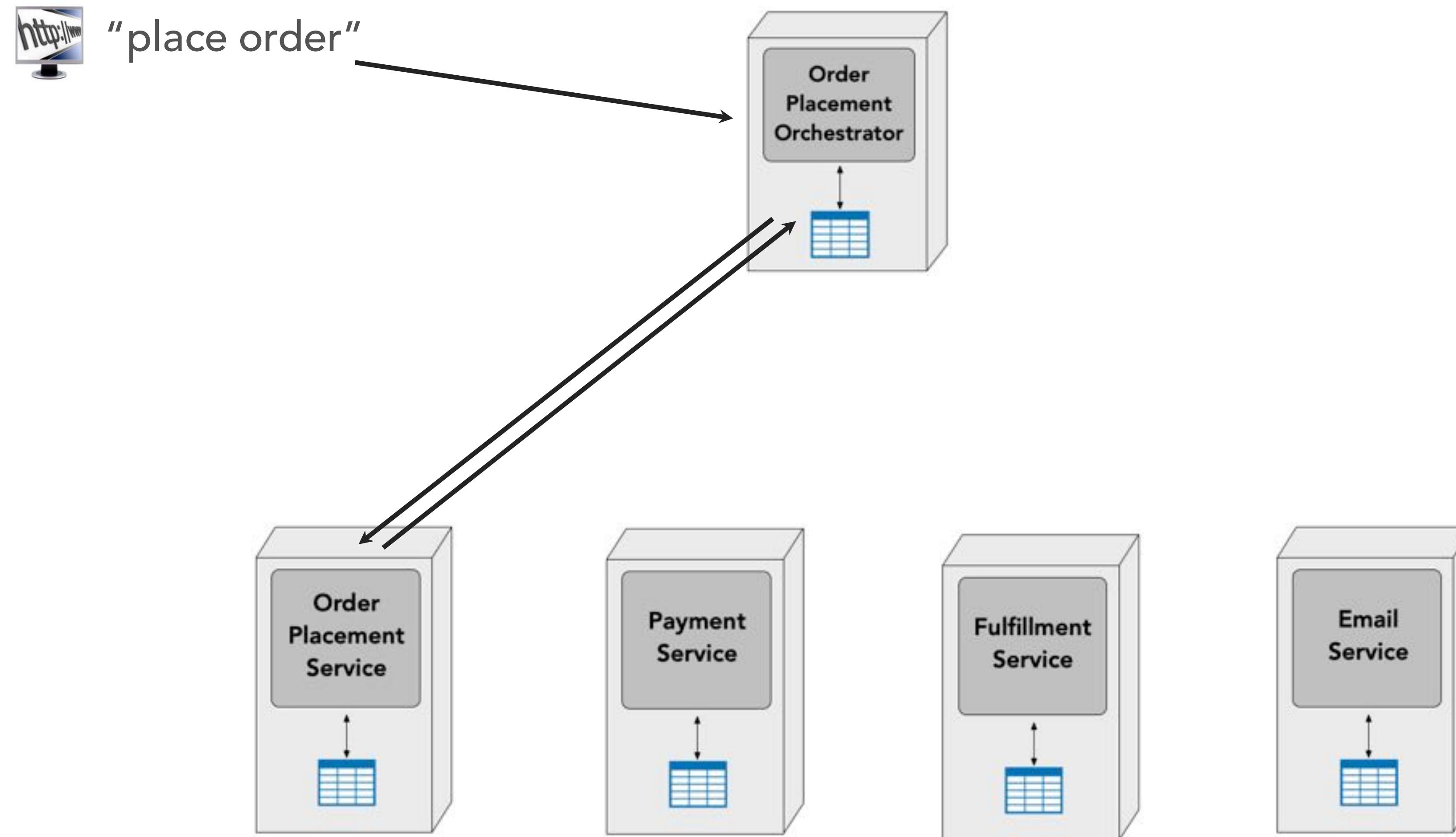
workflow patterns

stateful orchestration



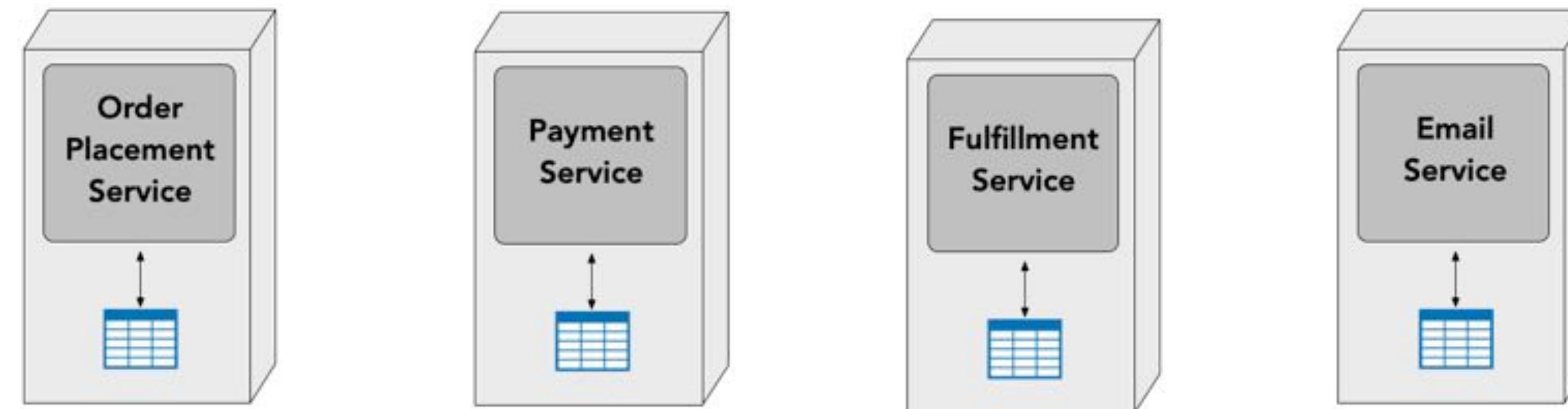
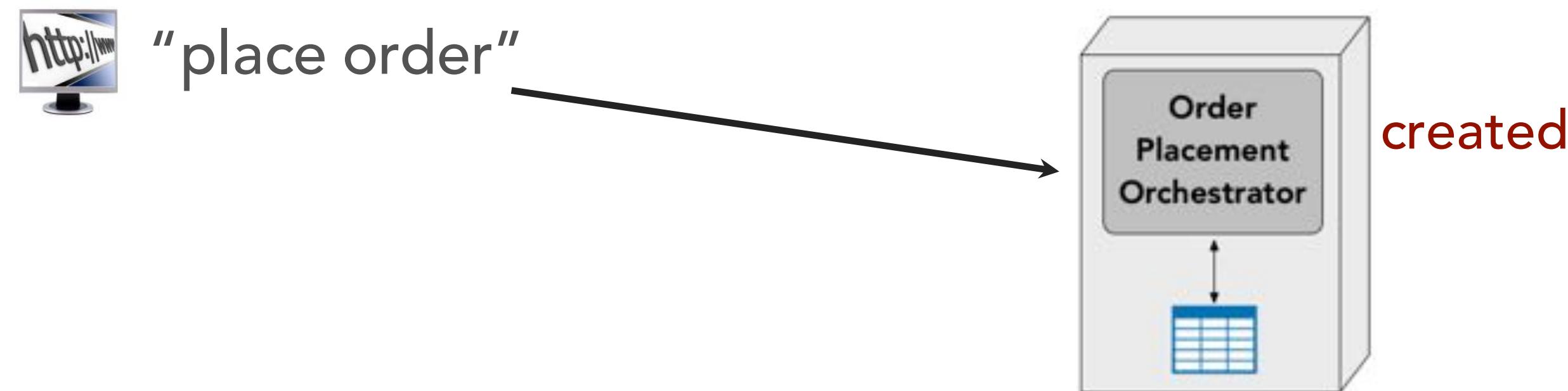
workflow patterns

stateful orchestration



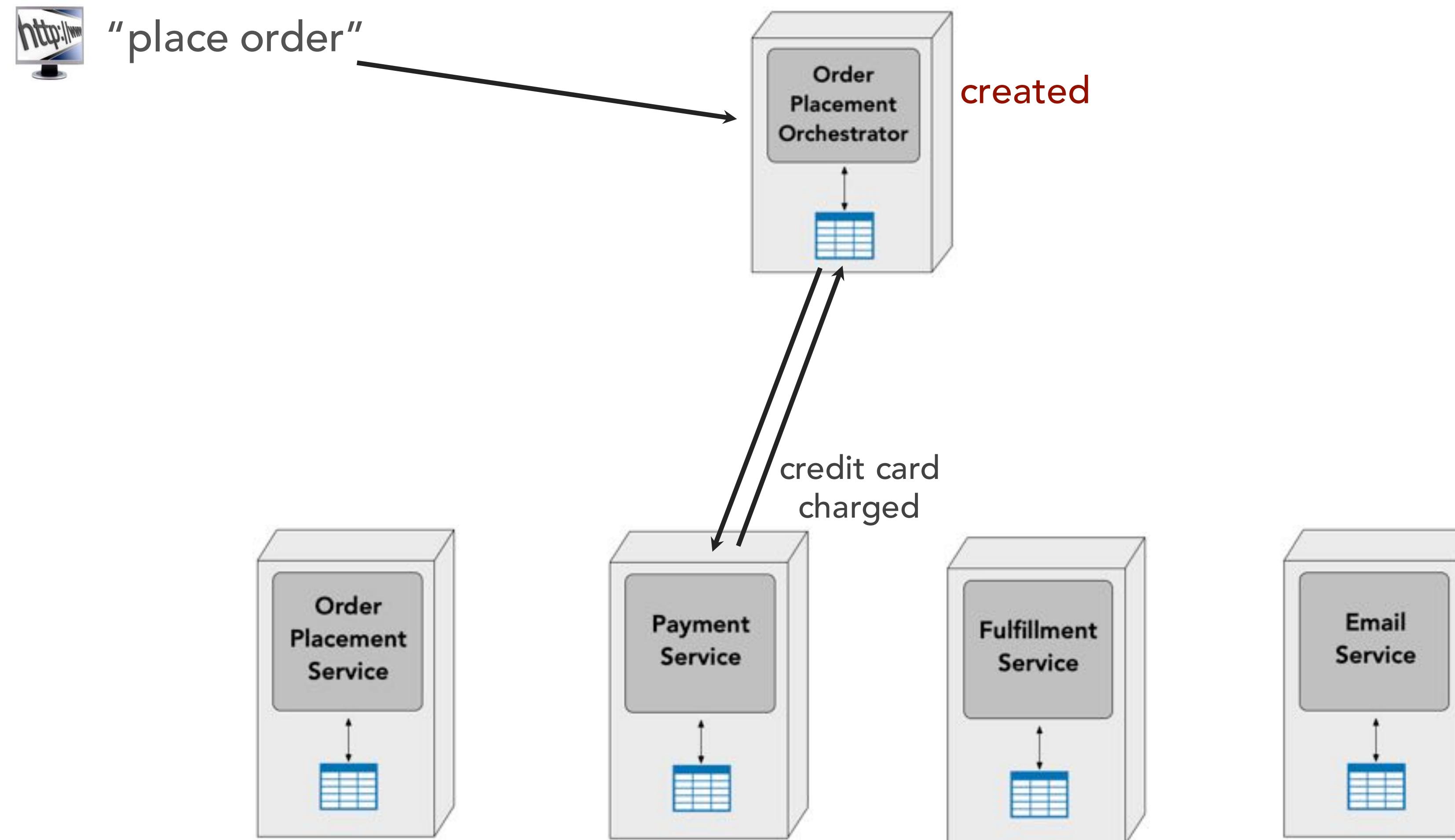
workflow patterns

stateful orchestration



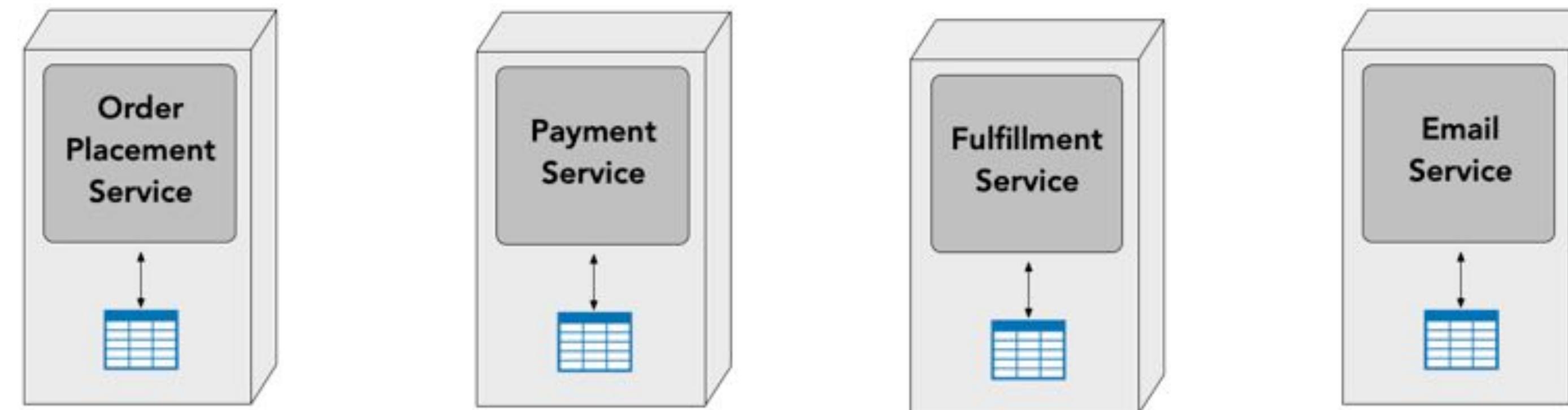
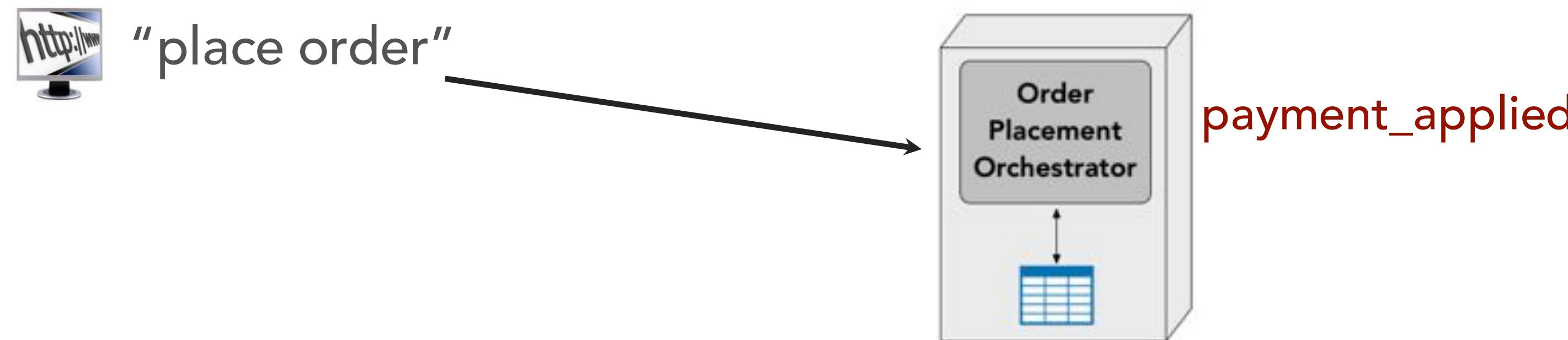
workflow patterns

stateful orchestration



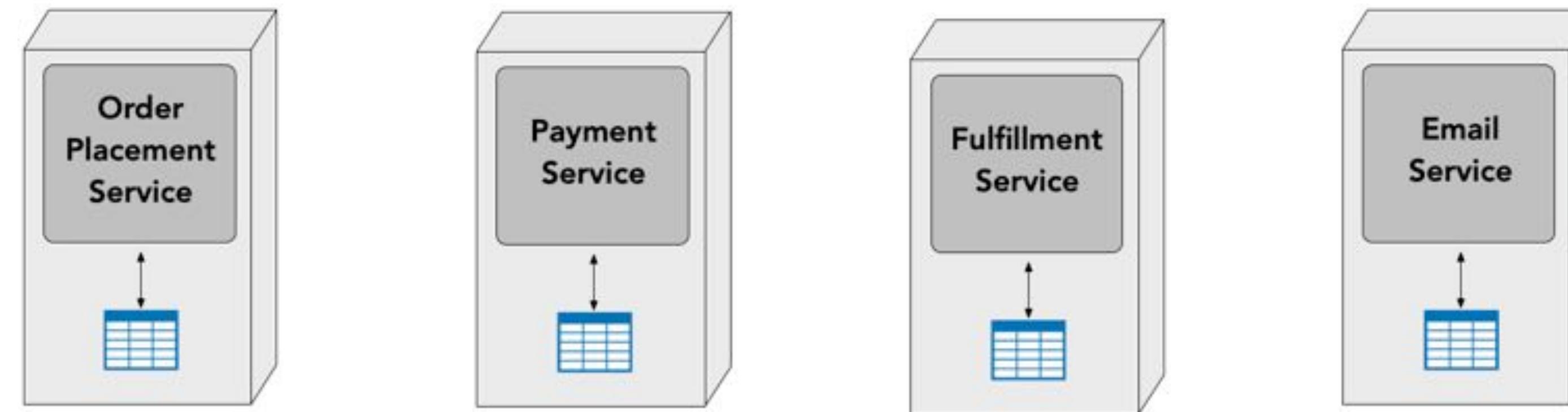
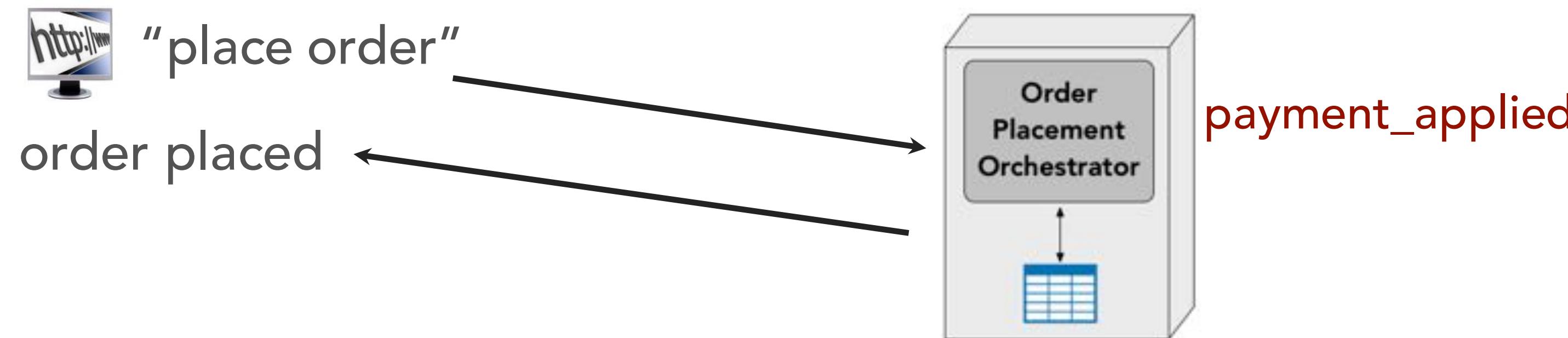
workflow patterns

stateful orchestration



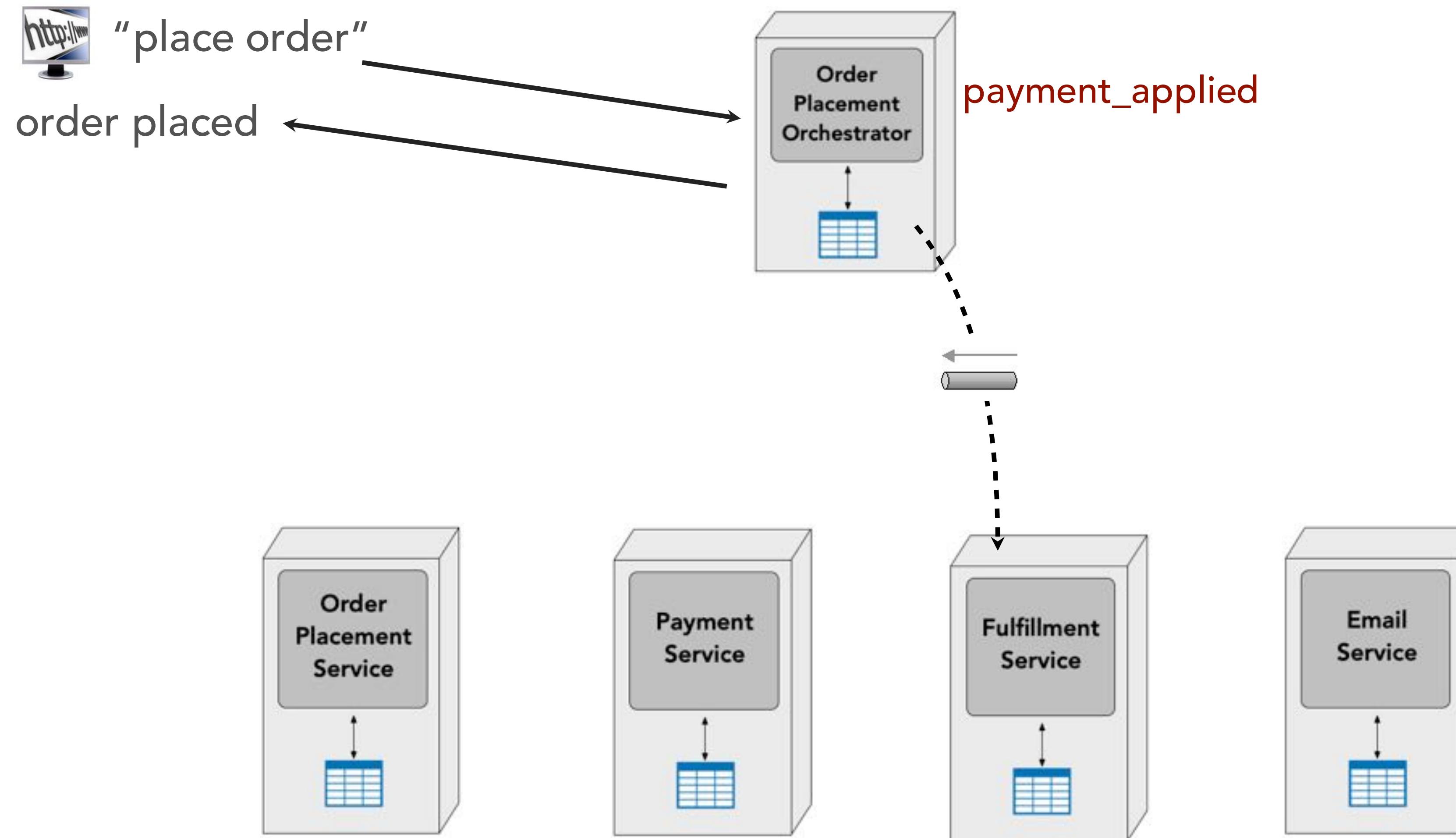
workflow patterns

stateful orchestration



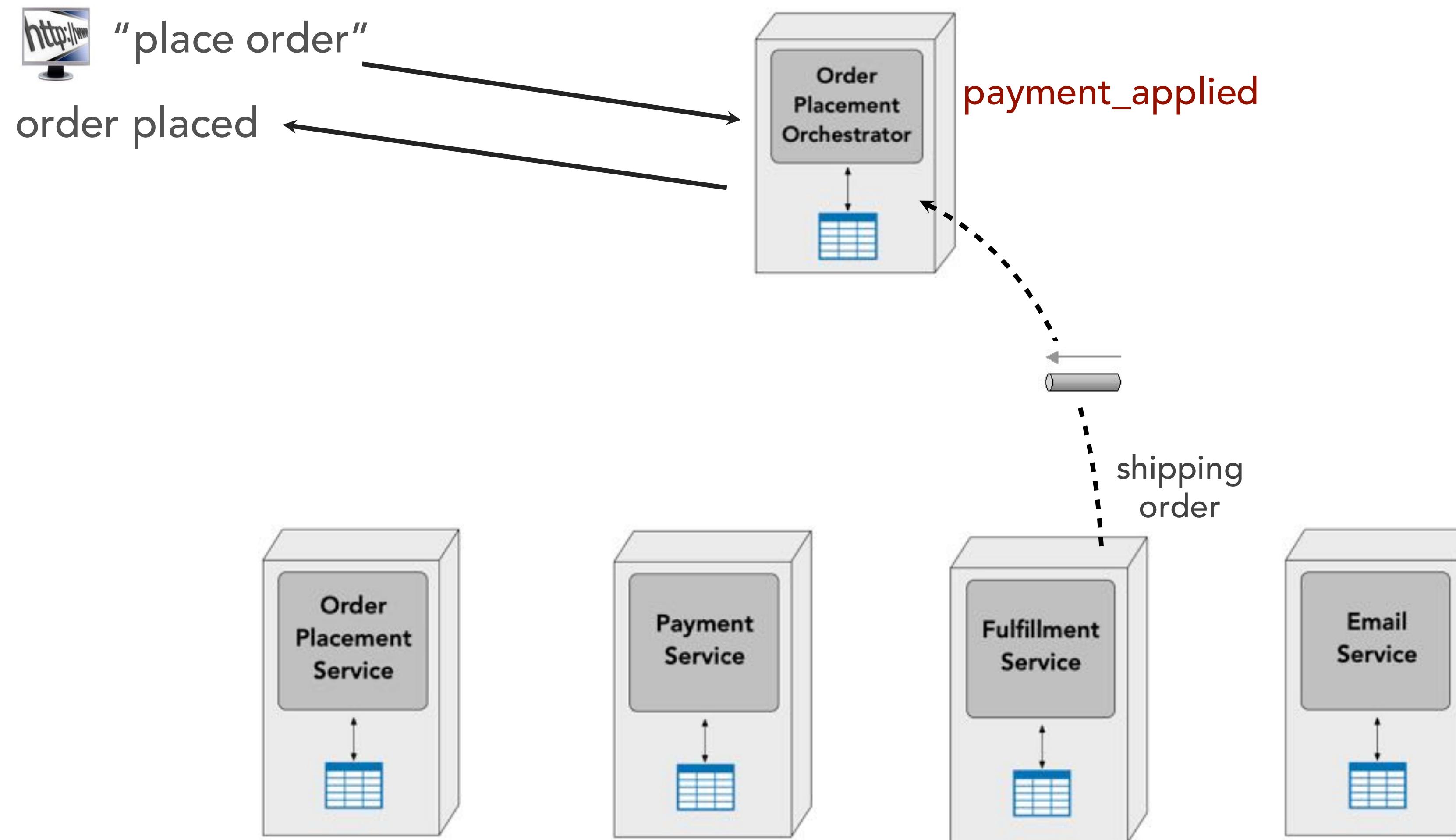
workflow patterns

stateful orchestration



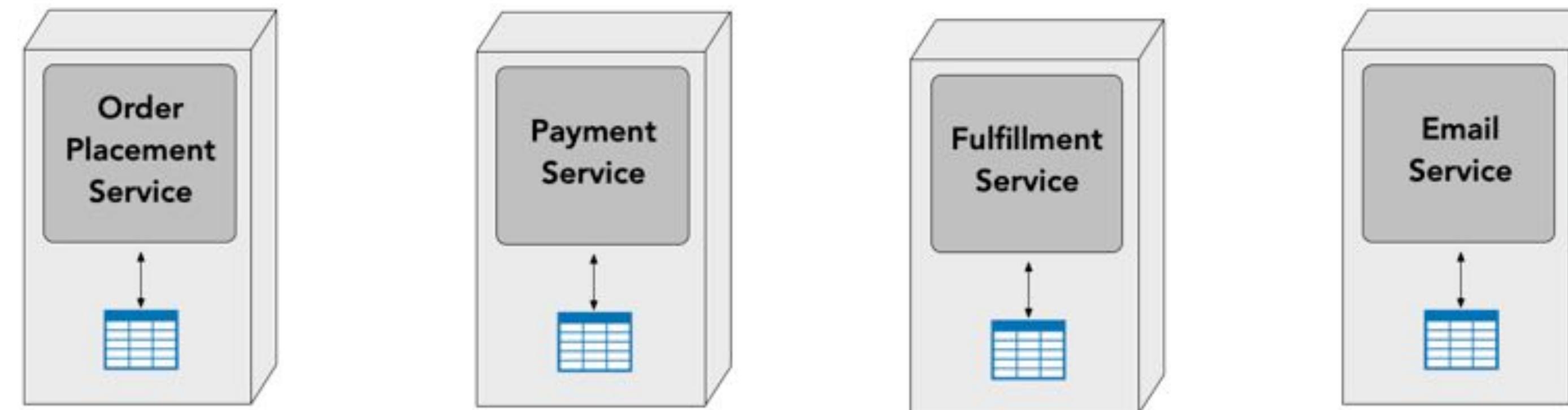
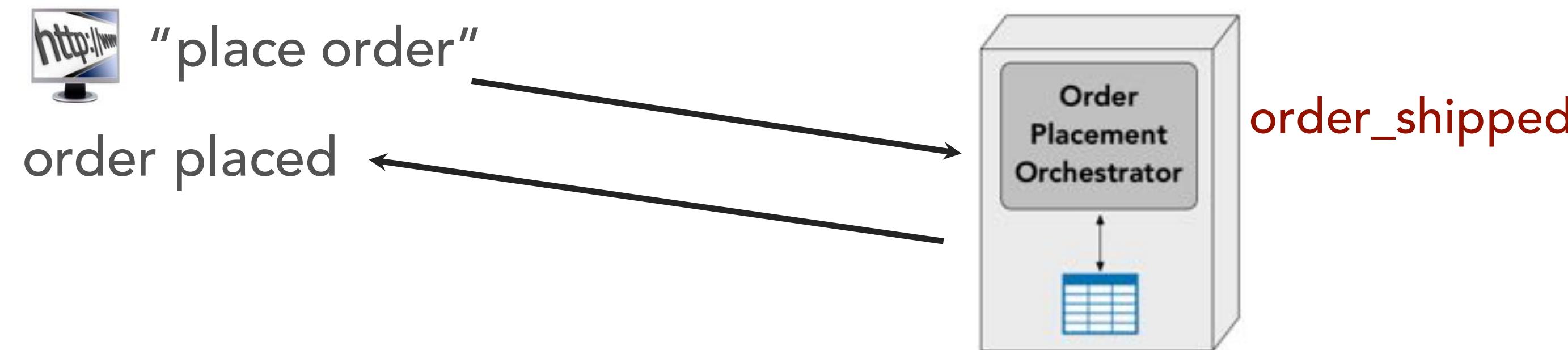
workflow patterns

stateful orchestration



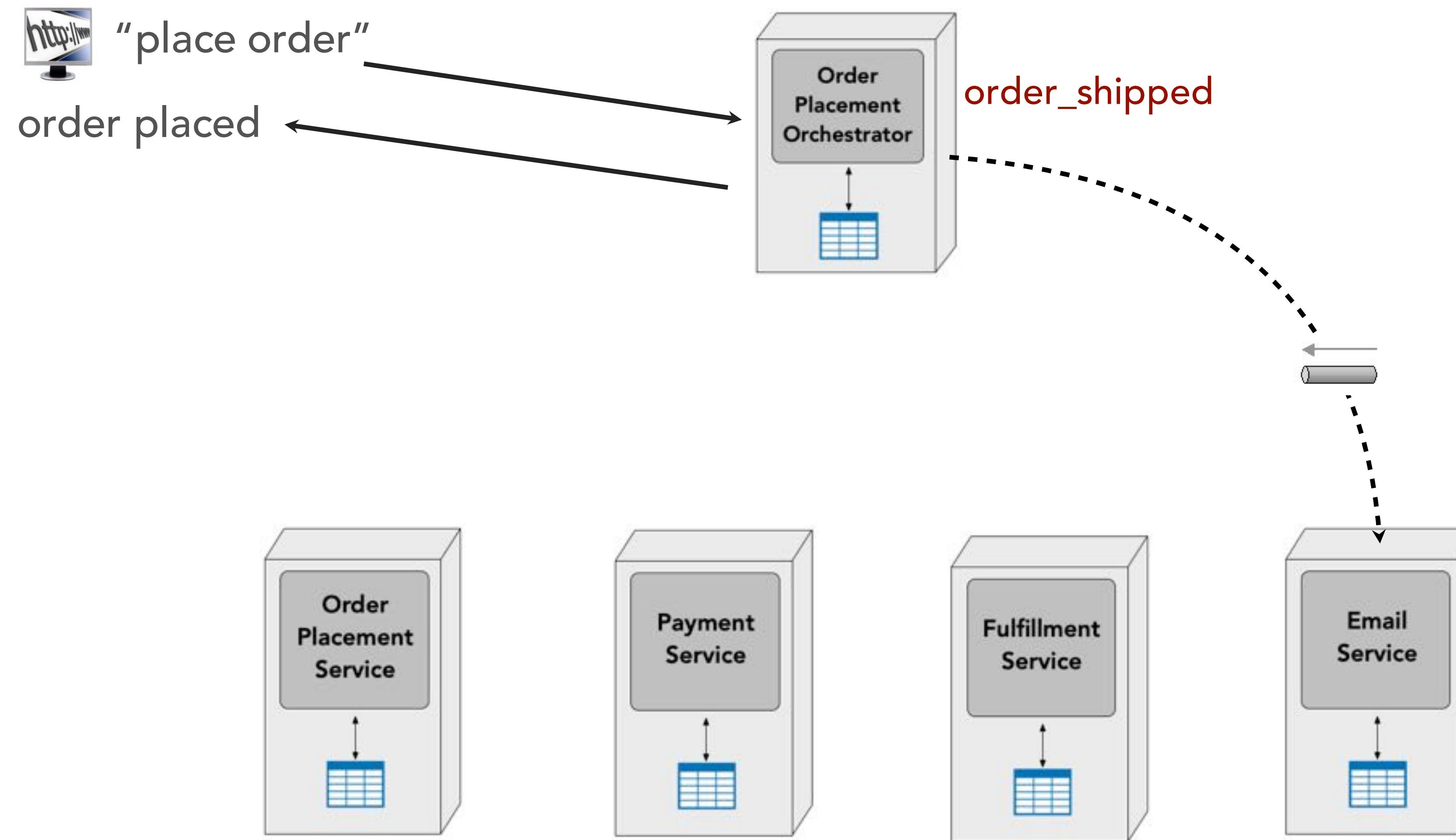
workflow patterns

stateful orchestration



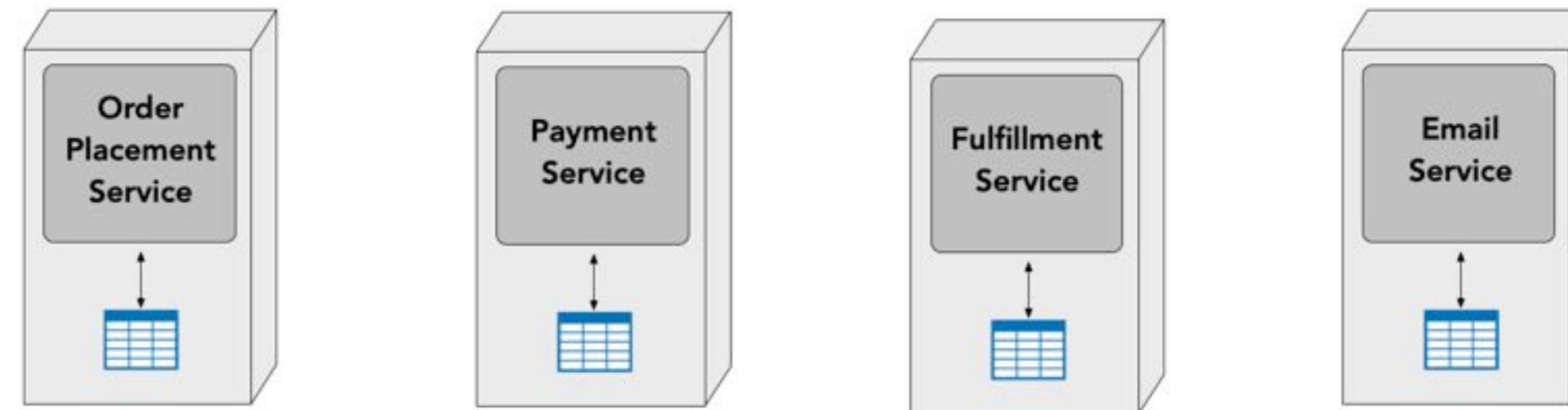
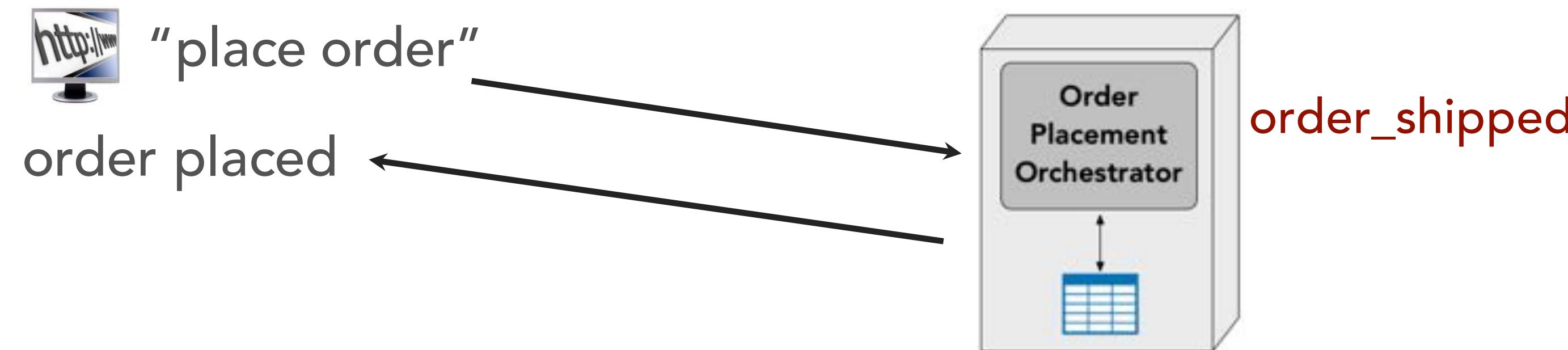
workflow patterns

stateful orchestration



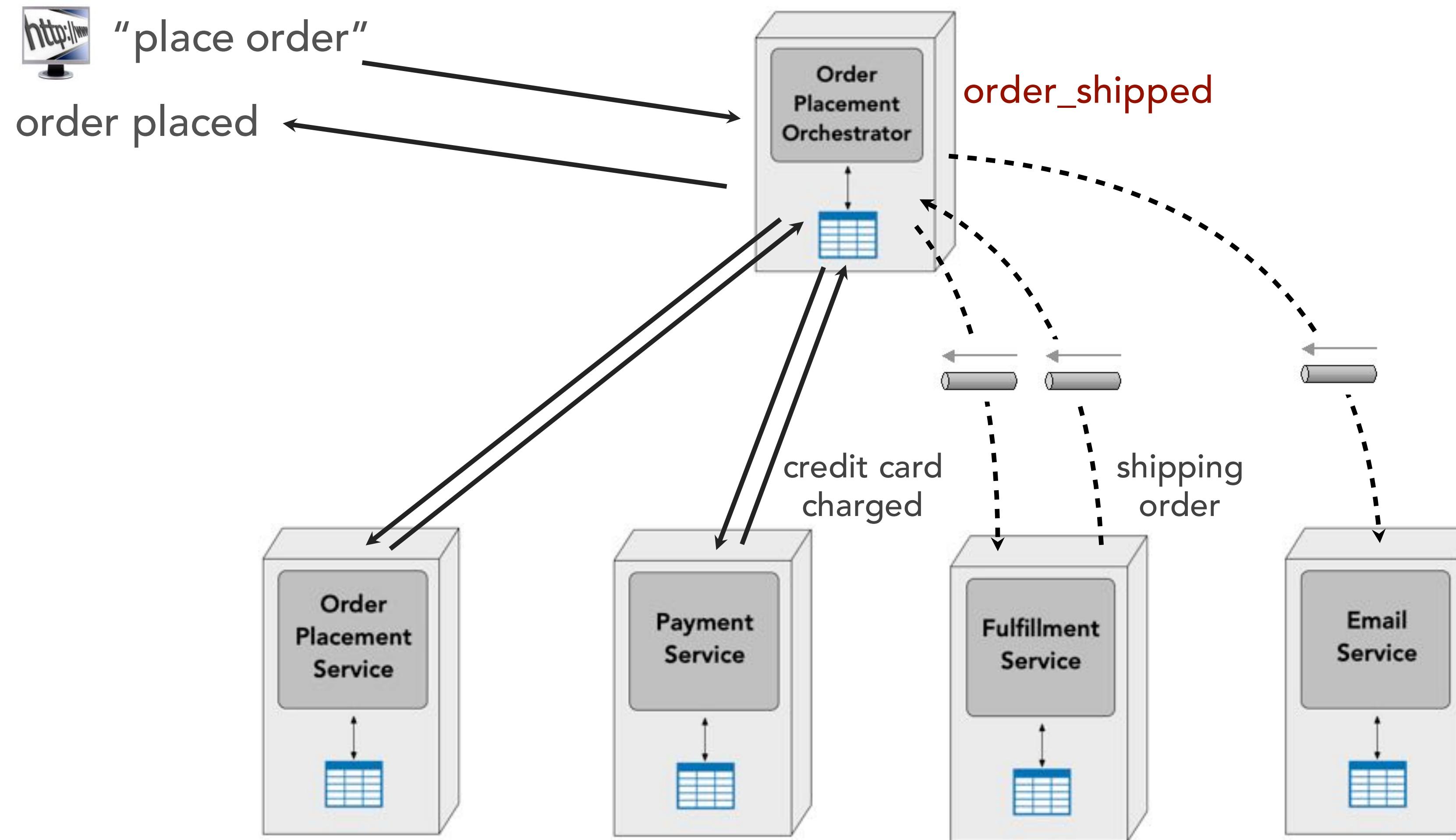
workflow patterns

stateful orchestration



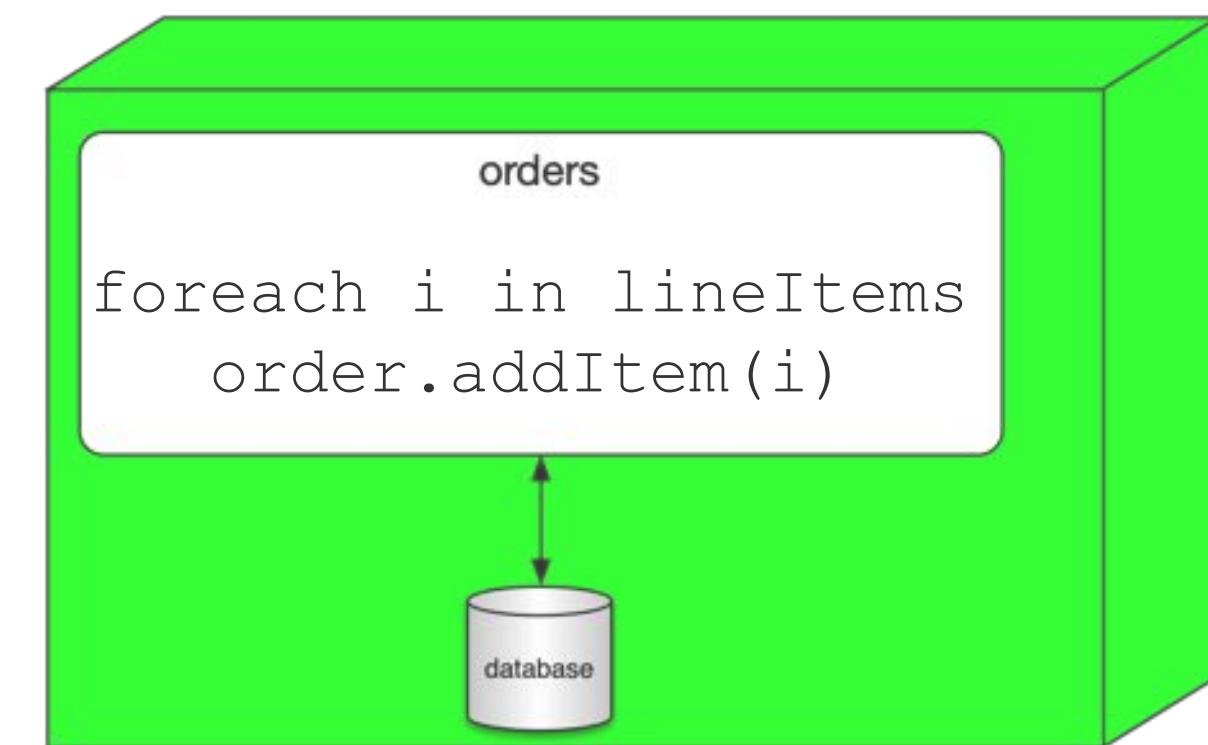
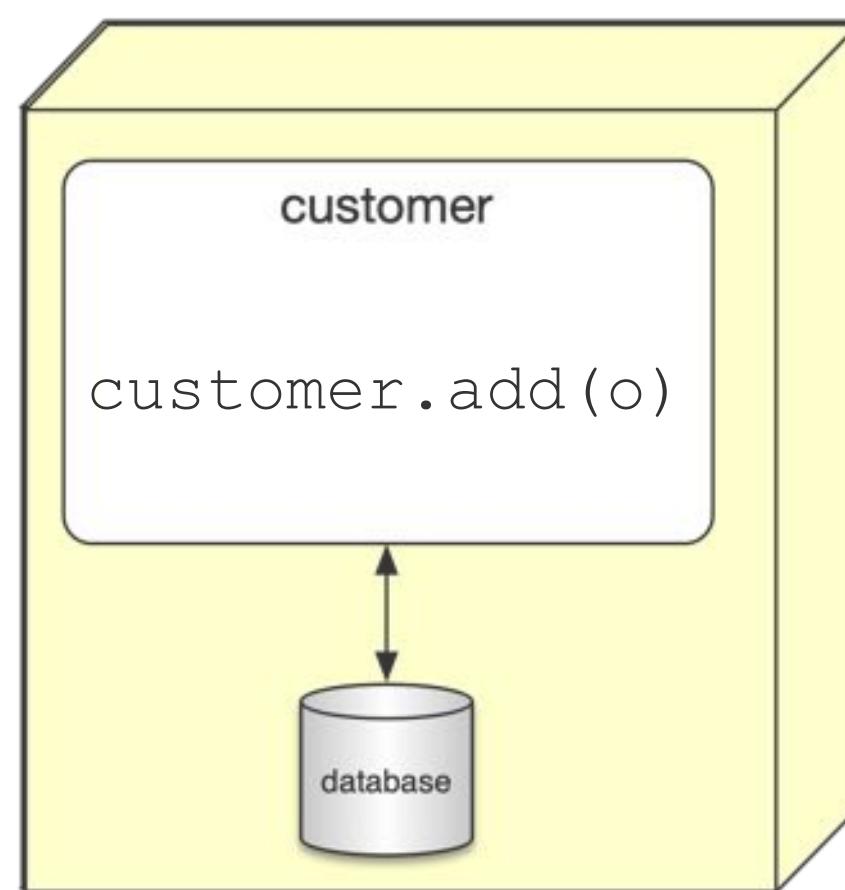
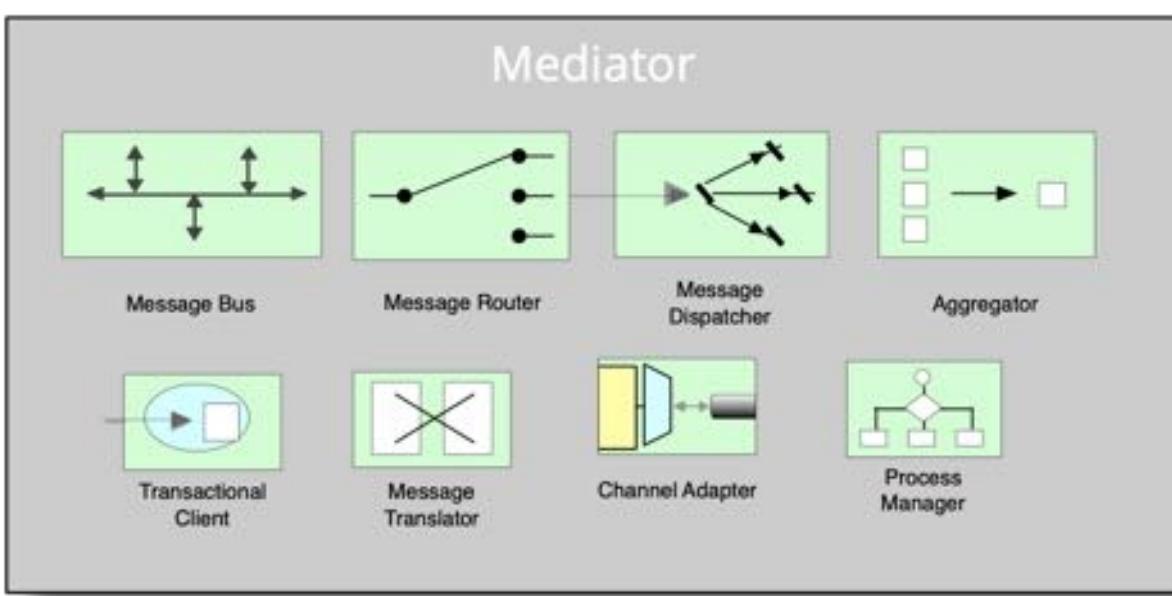
workflow patterns

stateful orchestration

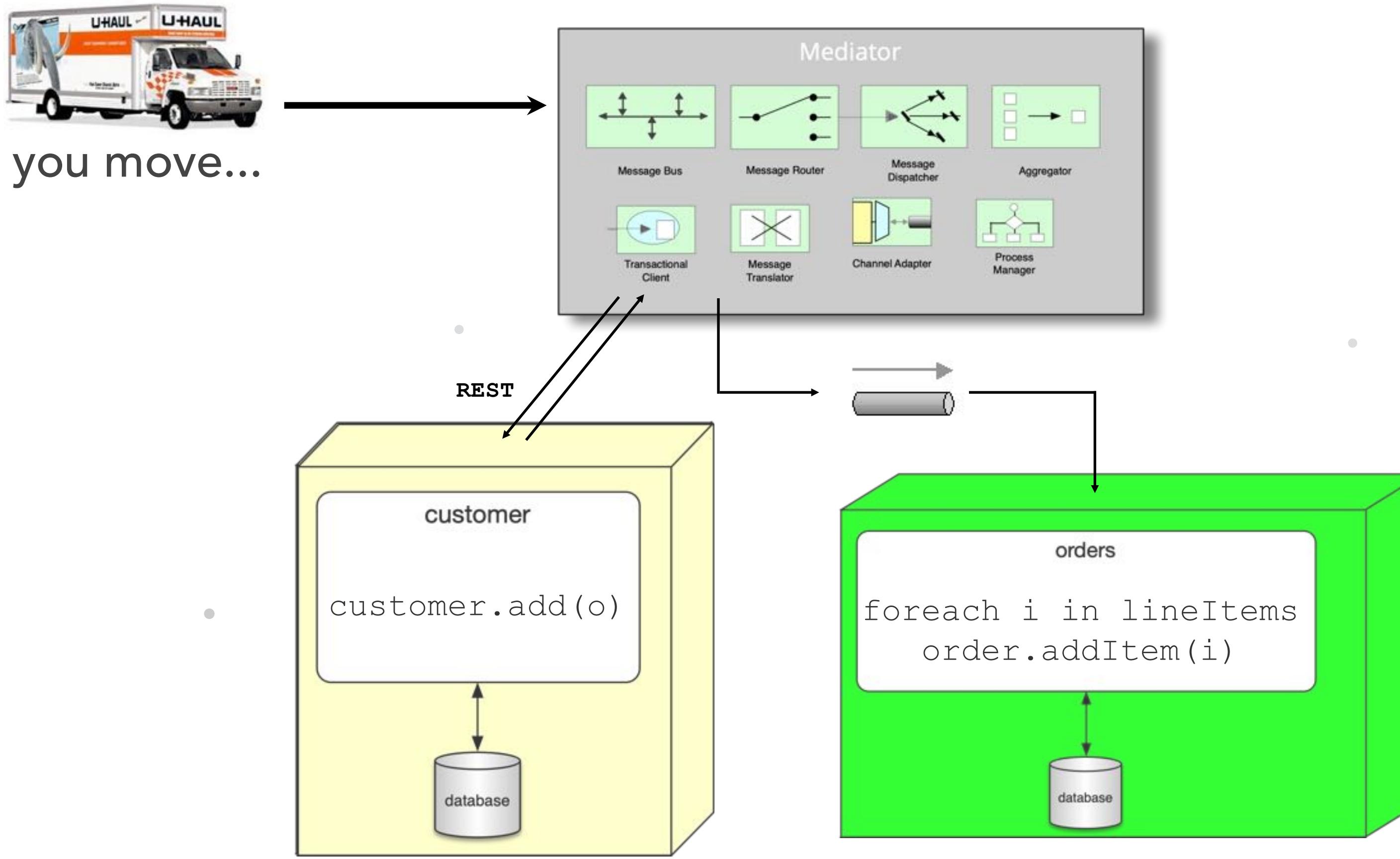


Sagas

mediator
event driven architecture

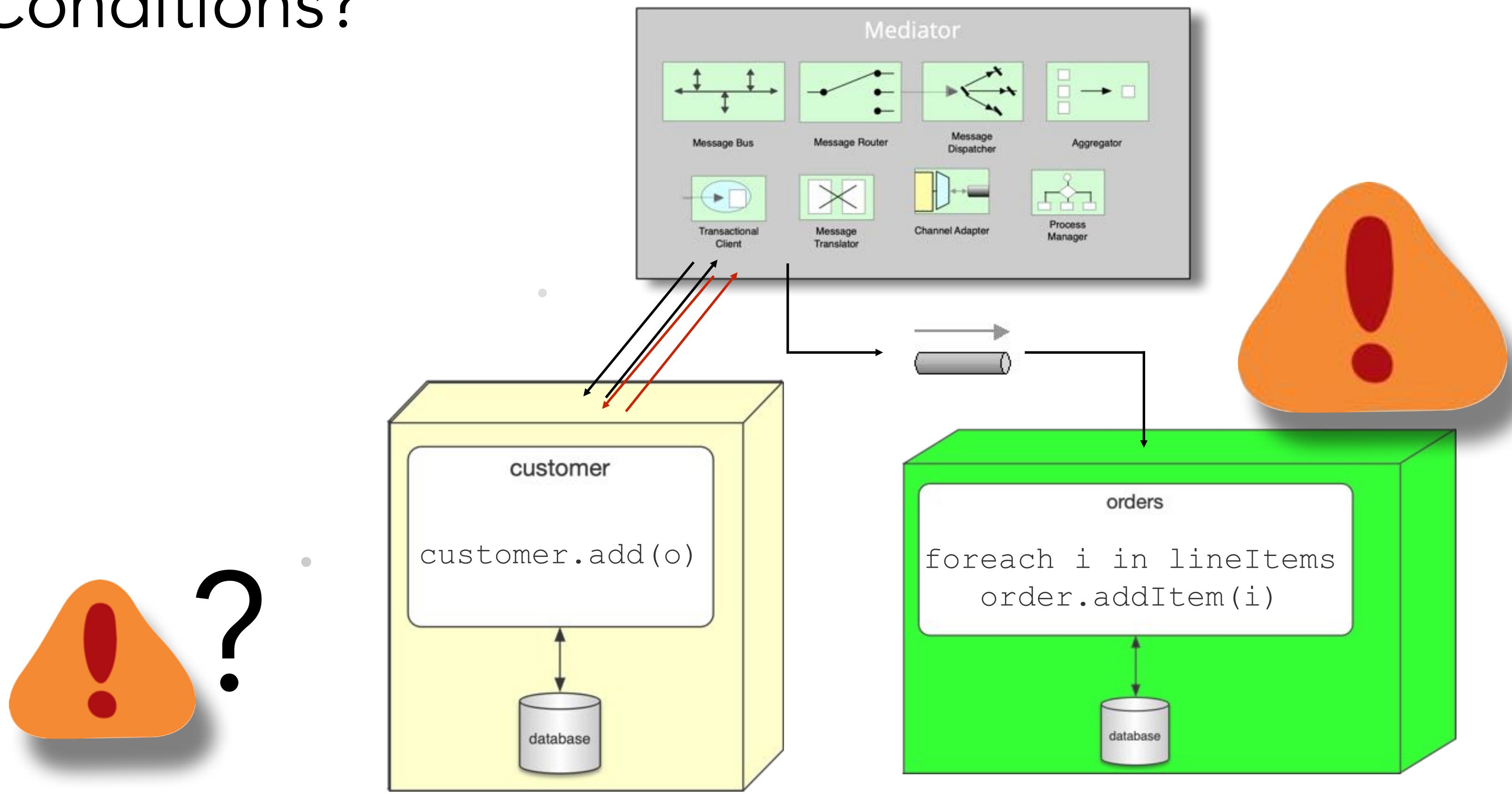


Orchestration via Sagas



Orchestration via Sagas

Error Conditions?

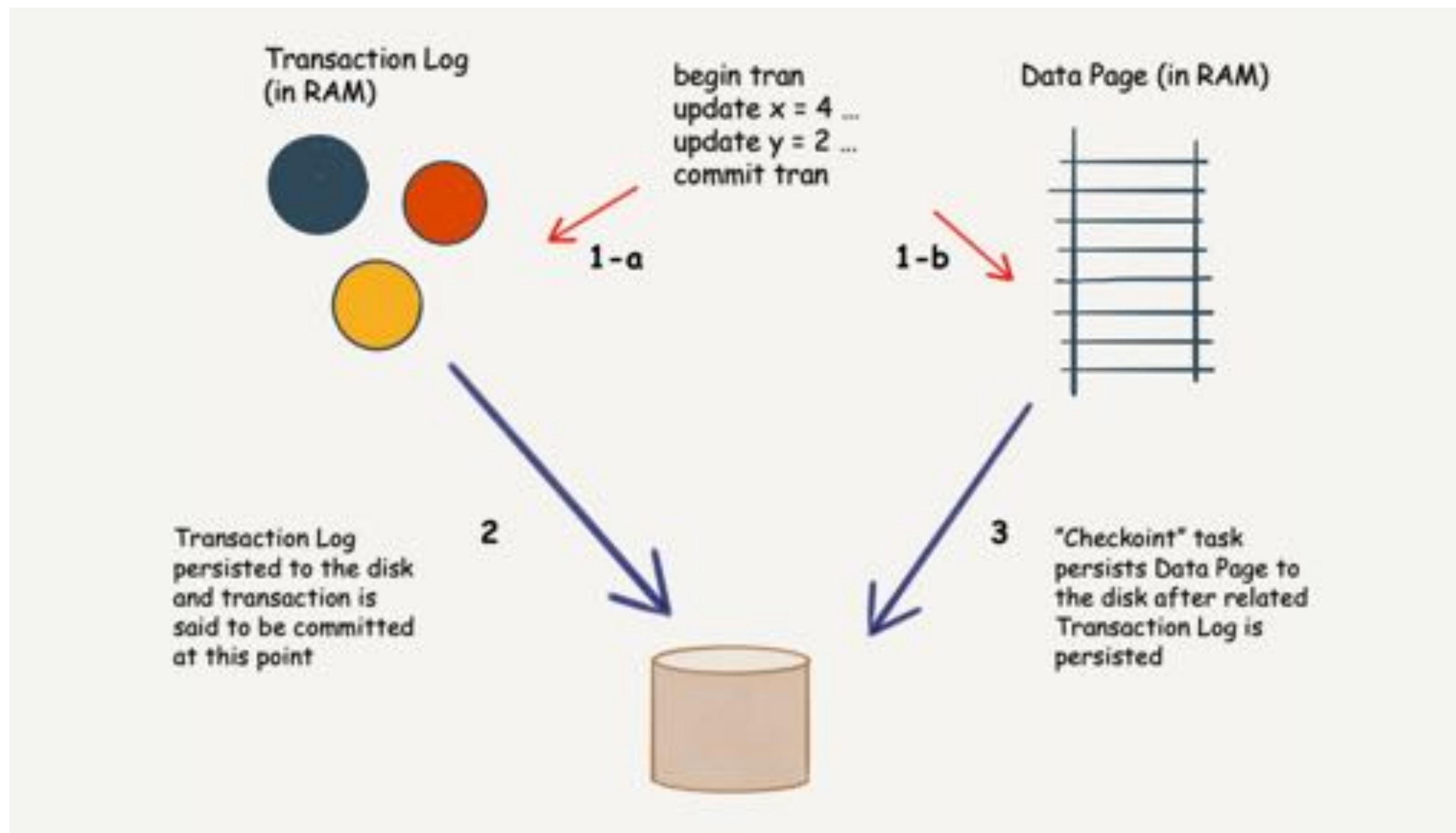


compensating transactions

Data Consistency in Microservices

- Sagas and compensating transactions
- Write-ahead log

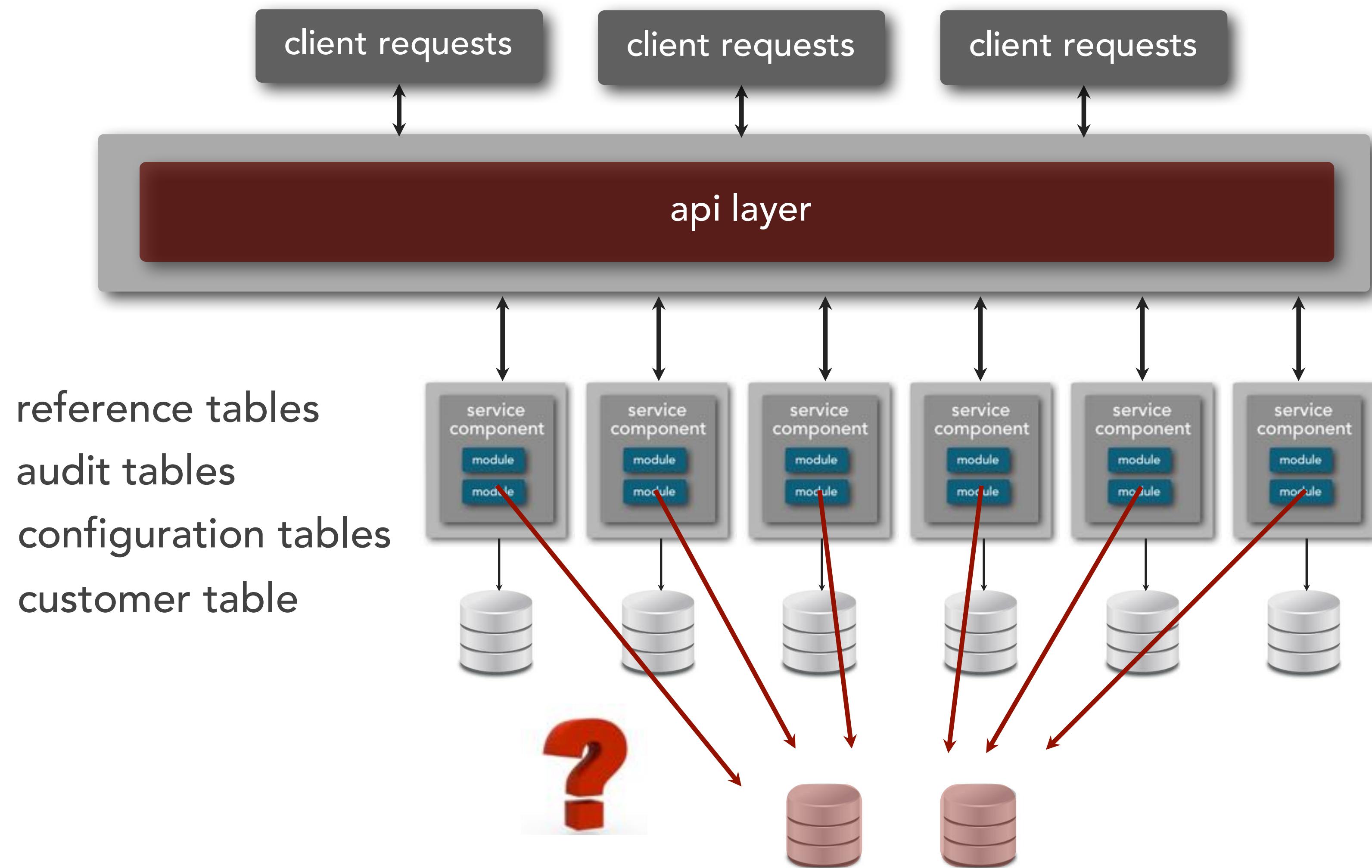
Write-ahead Log



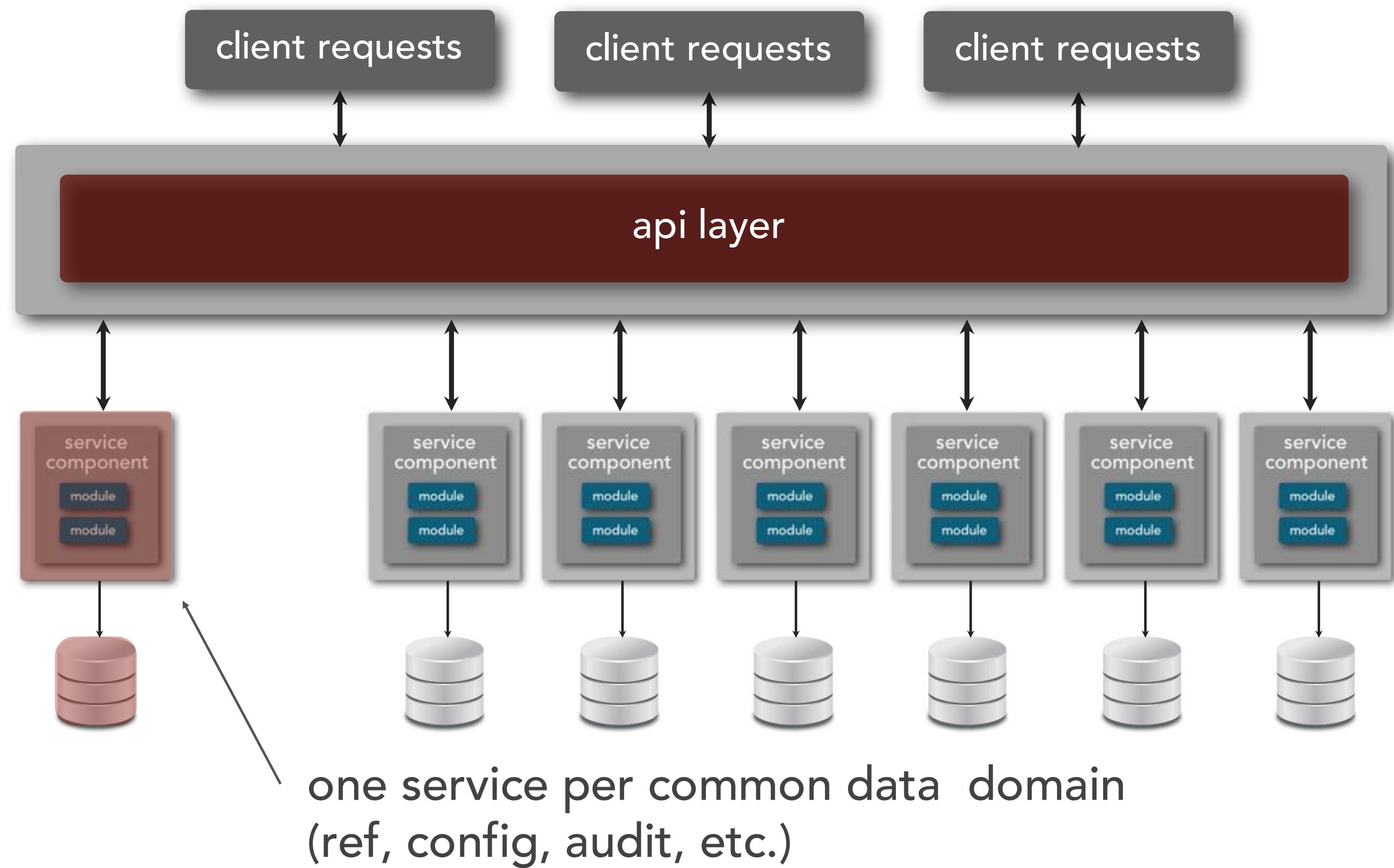
Data Consistency in Microservices

- Sagas and compensating transactions
- Write-ahead log
- Data reconciliation
- Data replication

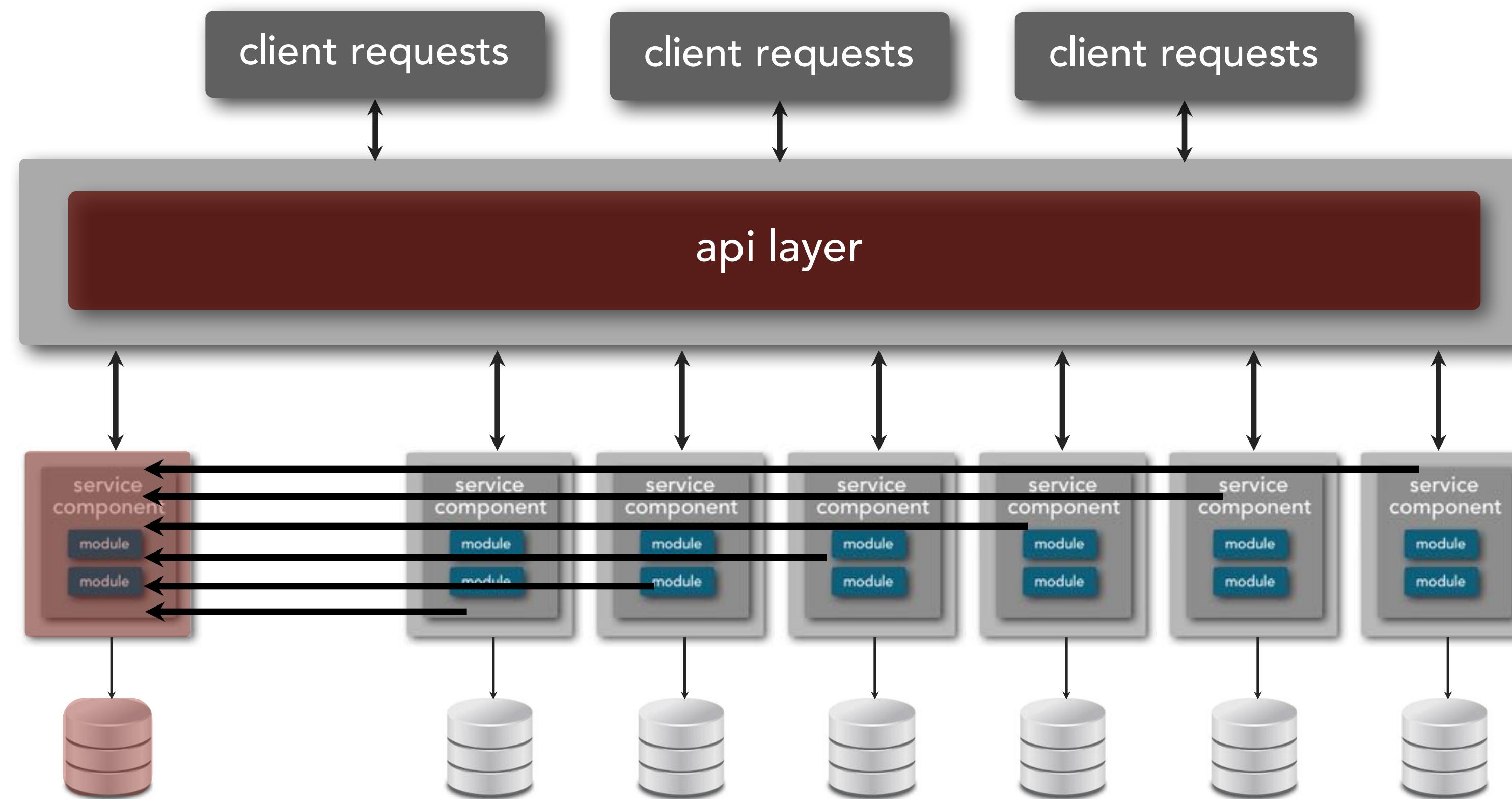
data replication



data replication

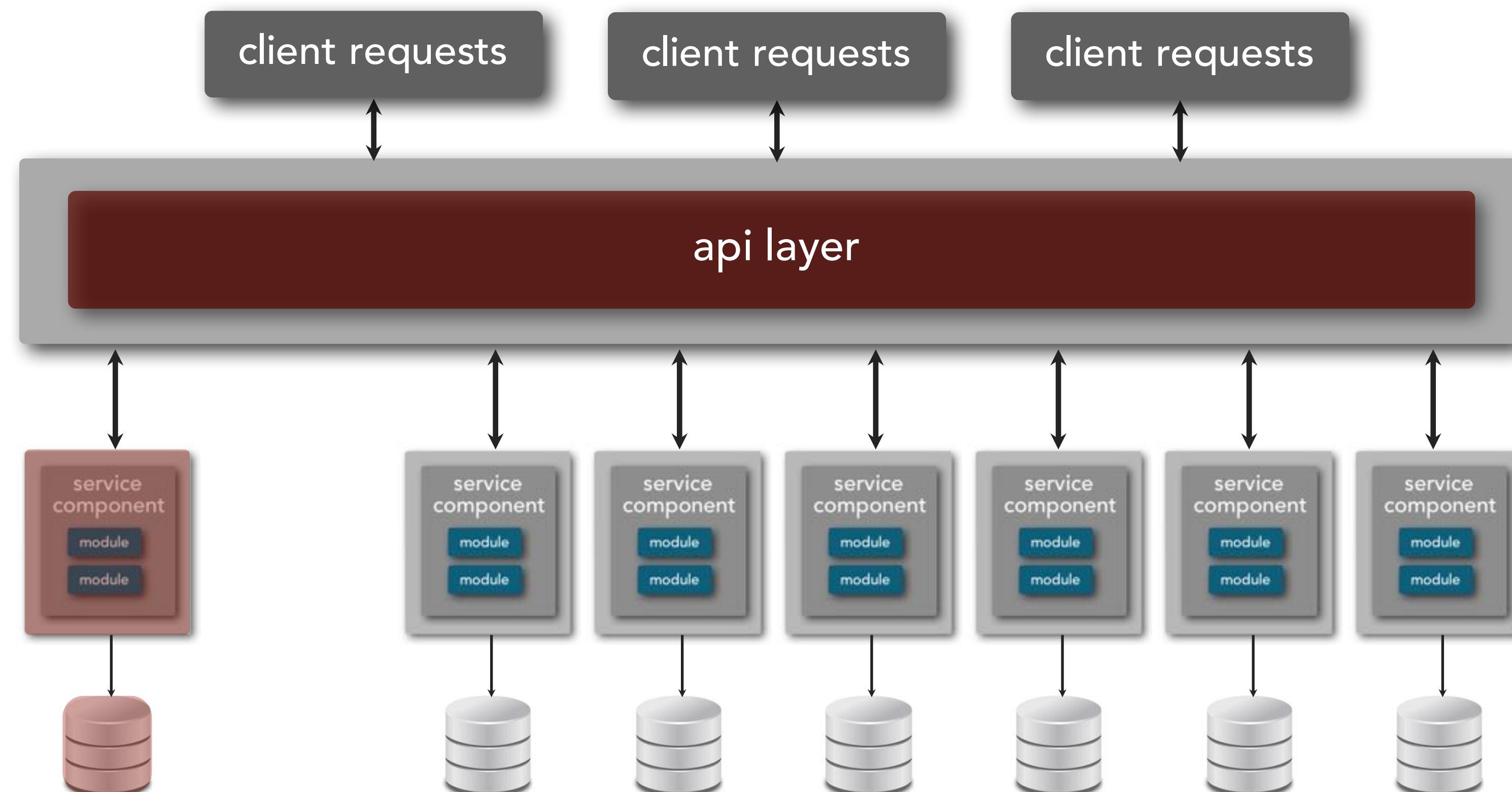


data replication

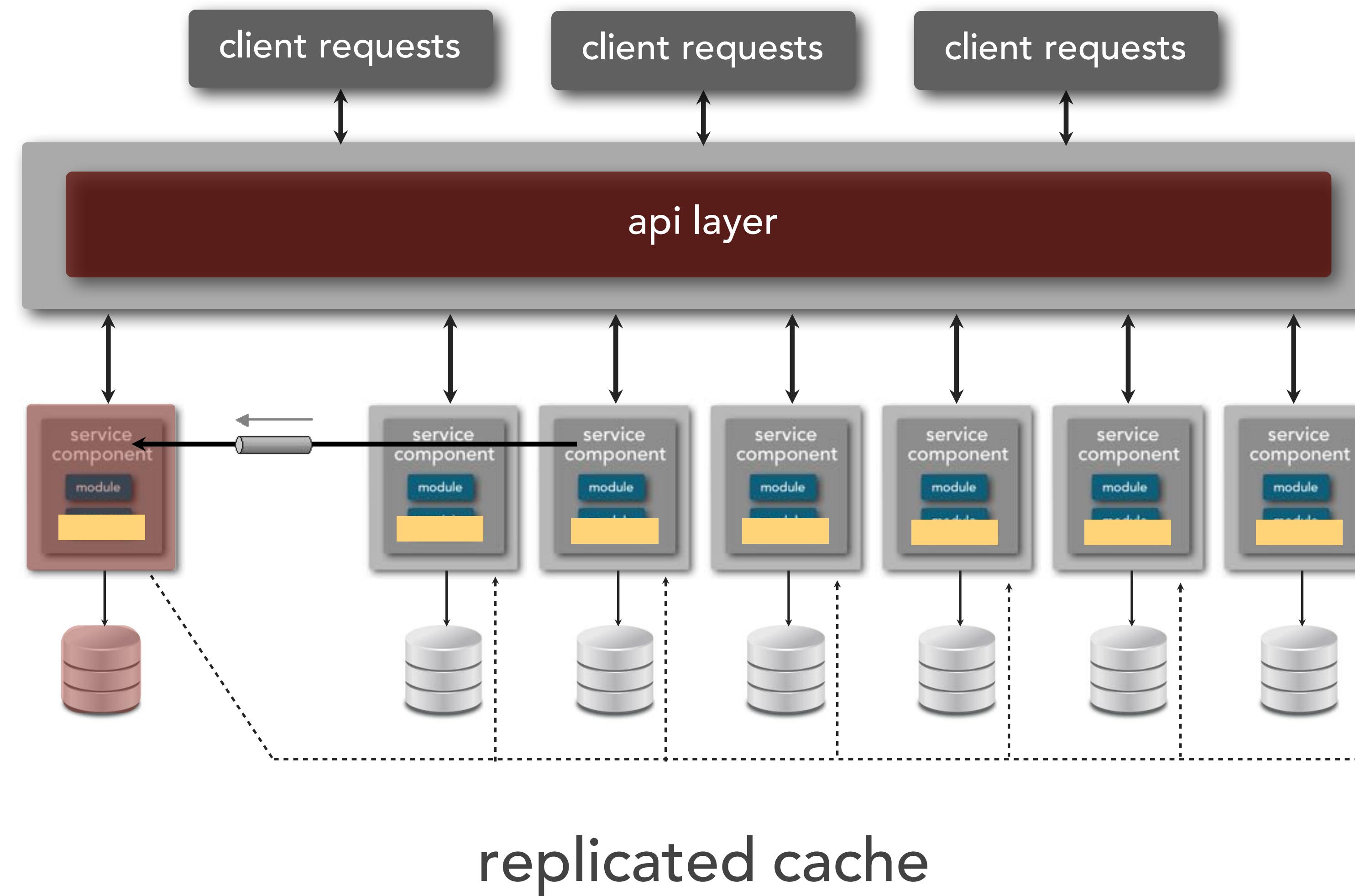


inter-service communication

data replication



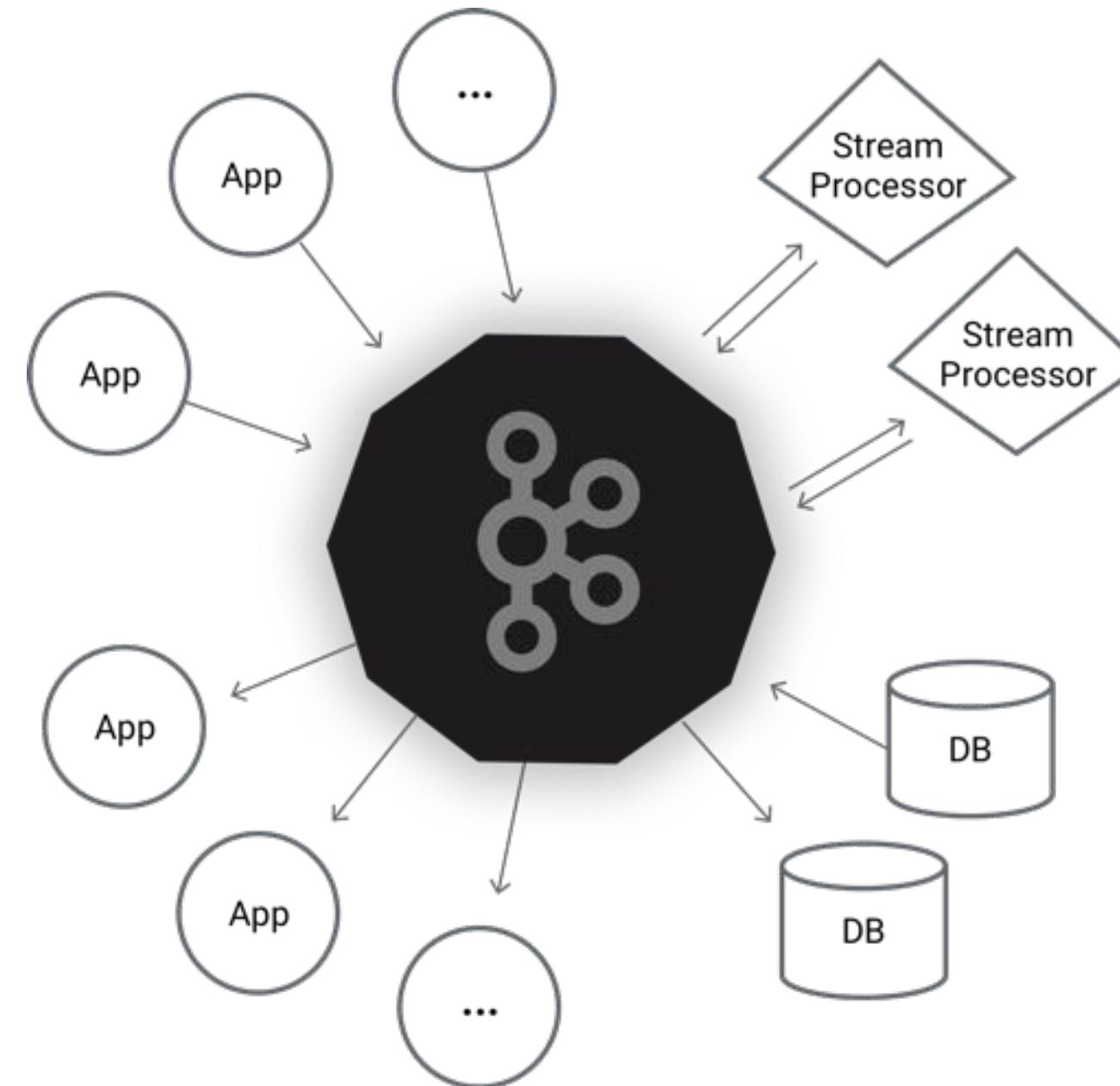
data replication



Data Consistency in Microservices

- Sagas and compensating transactions
- Write-ahead log
- Data reconciliation
- Data replication
- Events as a single source of truth

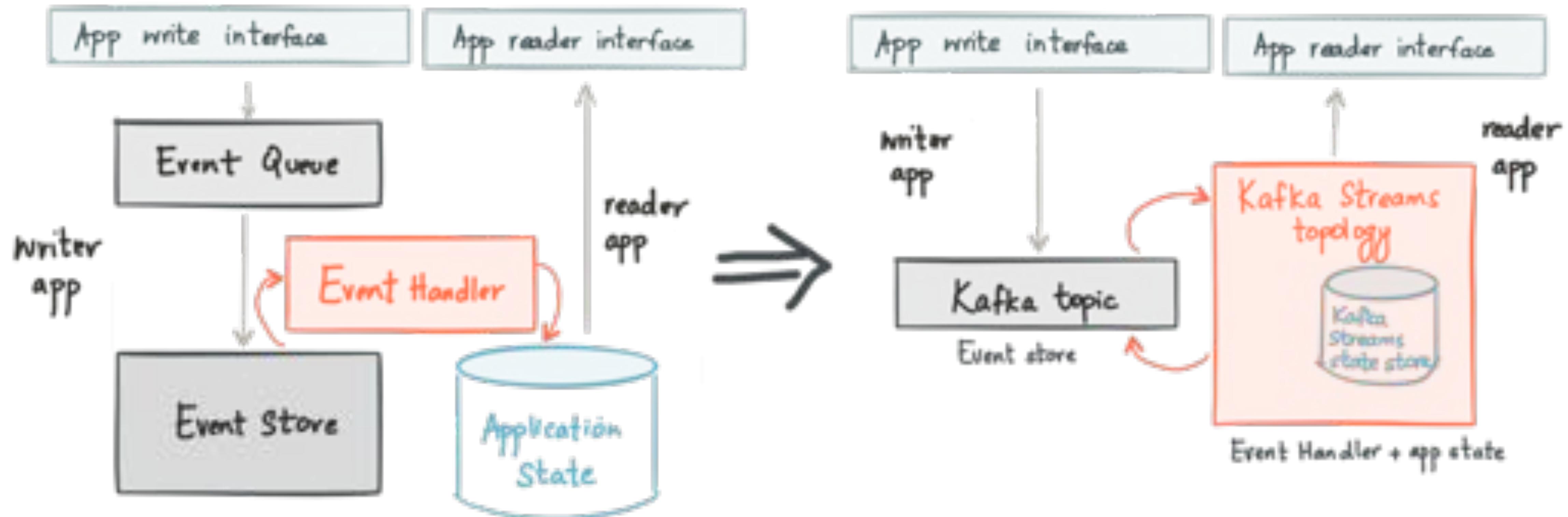
Event Streams == Truth



<https://kafka.apache.org/>

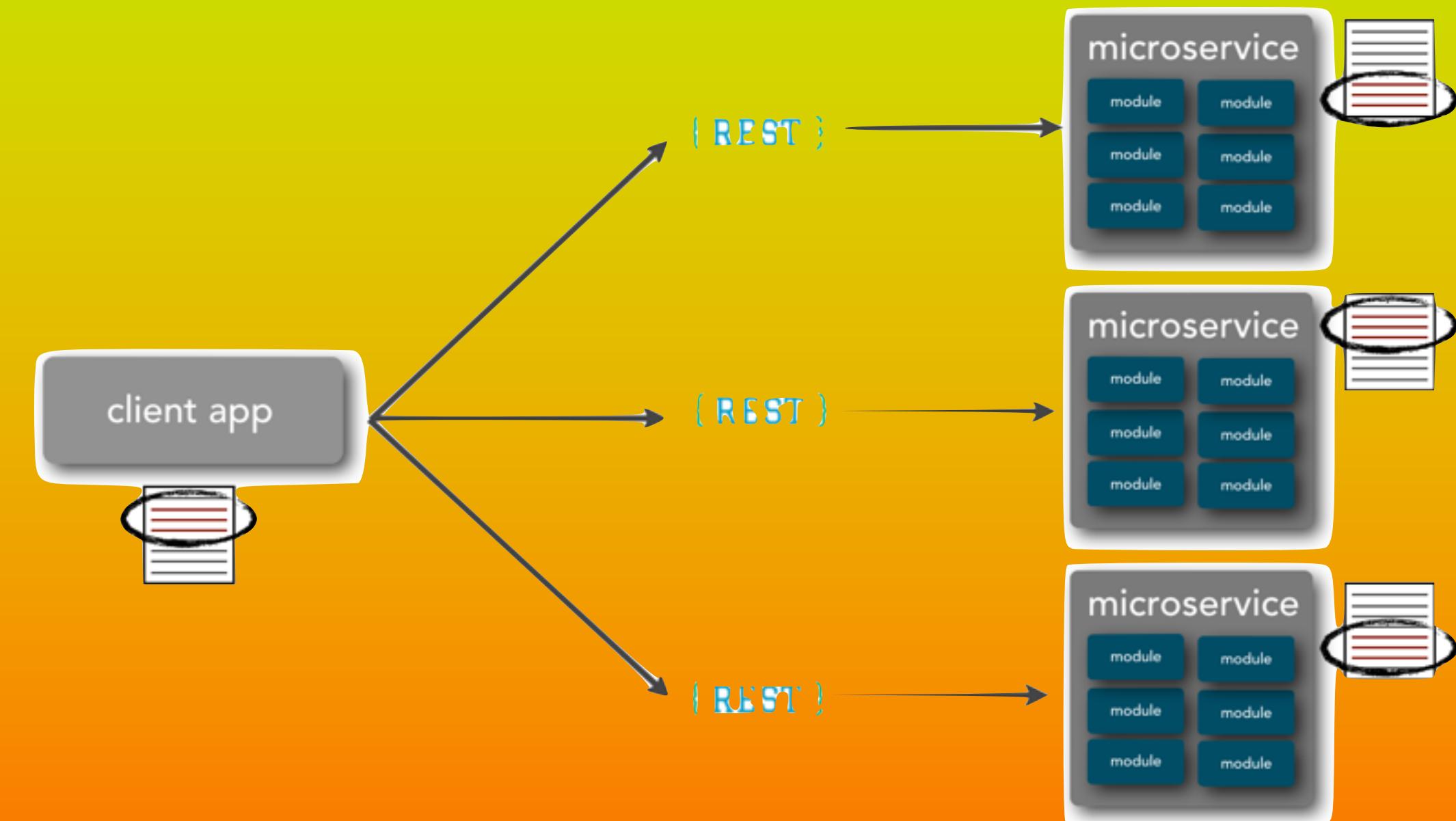
Is Kafka a database?

Kafka CQRS



tradeoffs

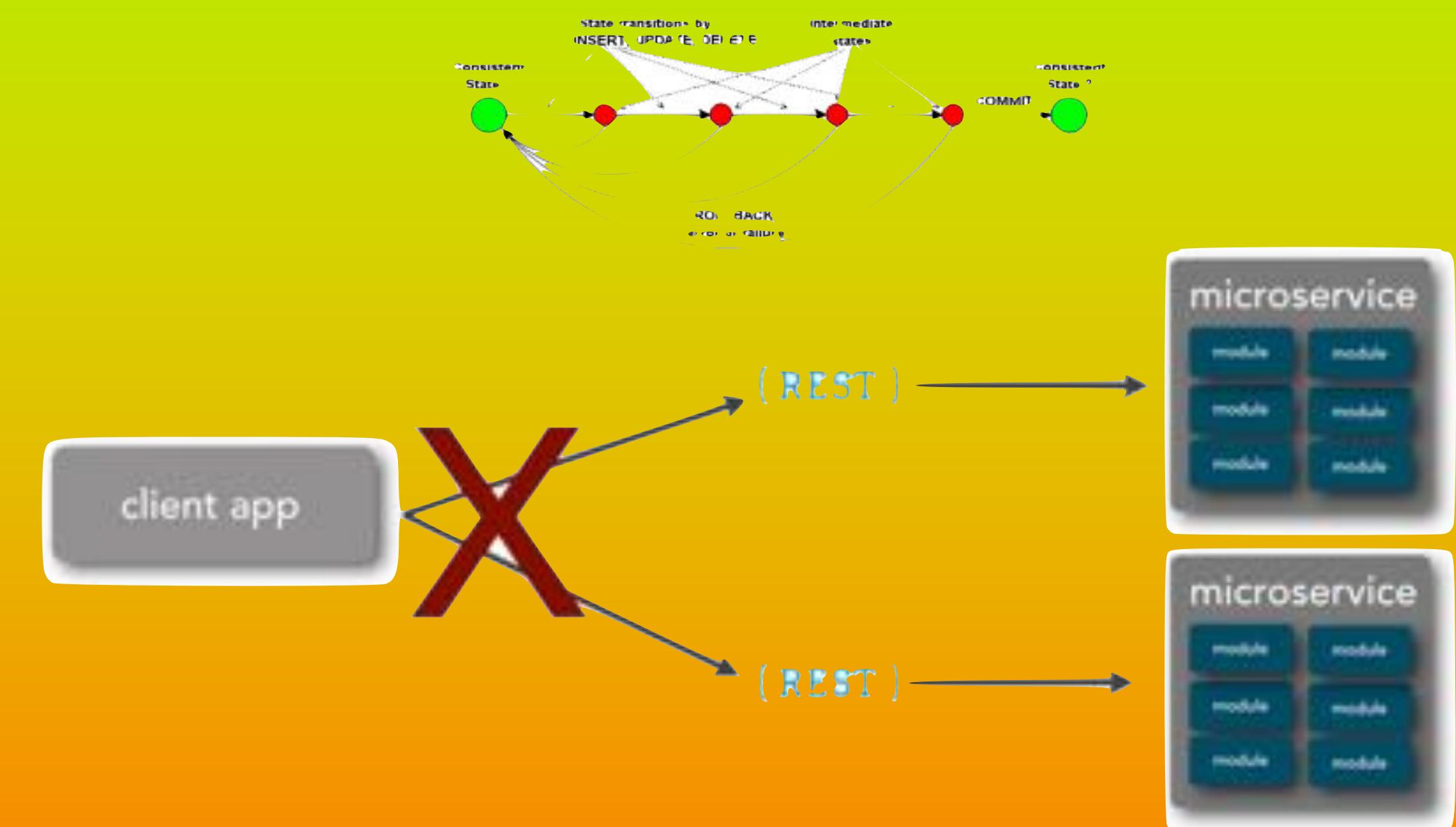
distributed logging facilities to provide a holistic view of a transaction



<https://www.developertoarchitect.com/lessons/lesson4.html>

tradeoffs

distributed transaction management



<https://www.developertoarchitect.com/lessons/lesson53.html>

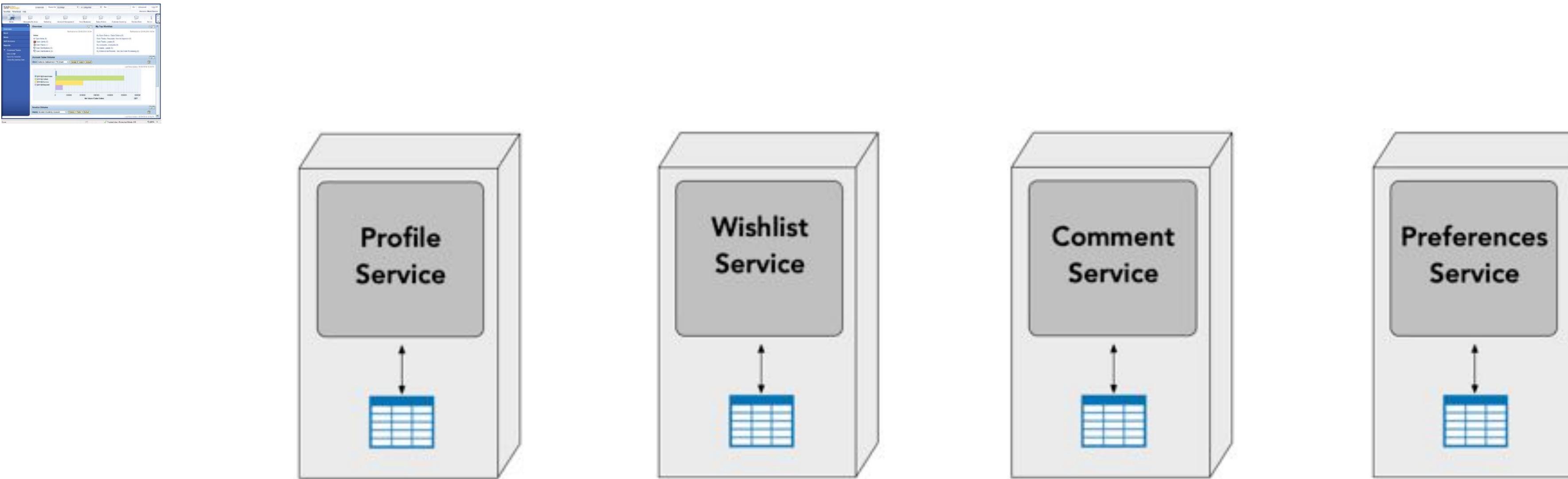
operational | analytical ?

operational | analytical ?

ACID | BASE ?

eventual consistency

the problem...



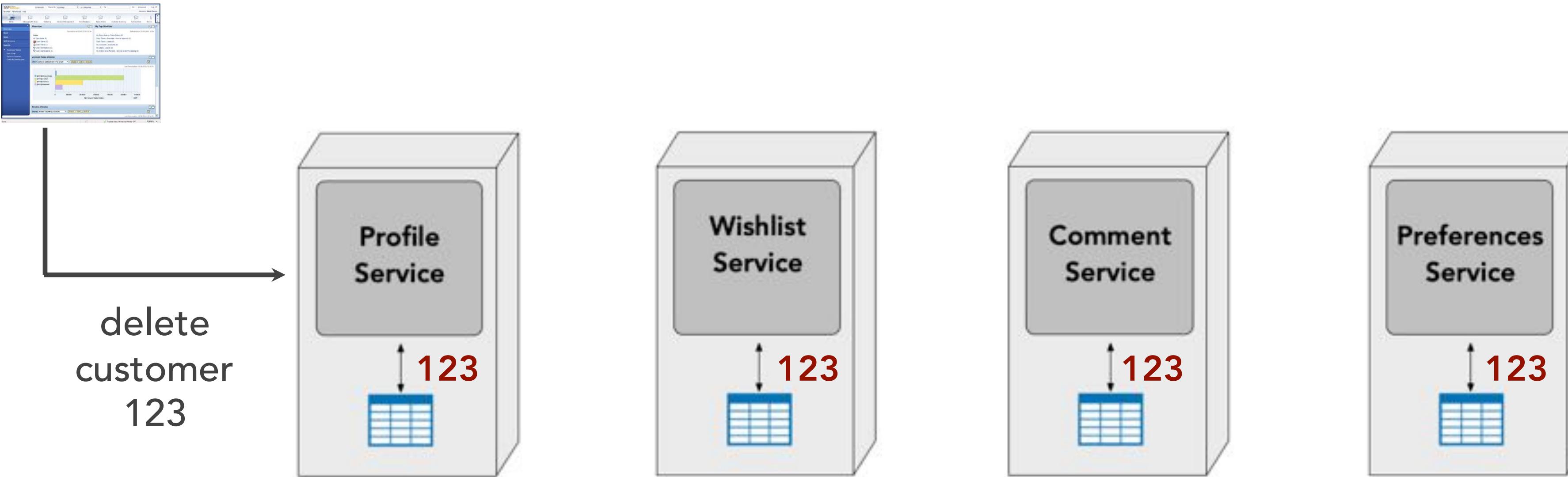
eventual consistency

the problem...



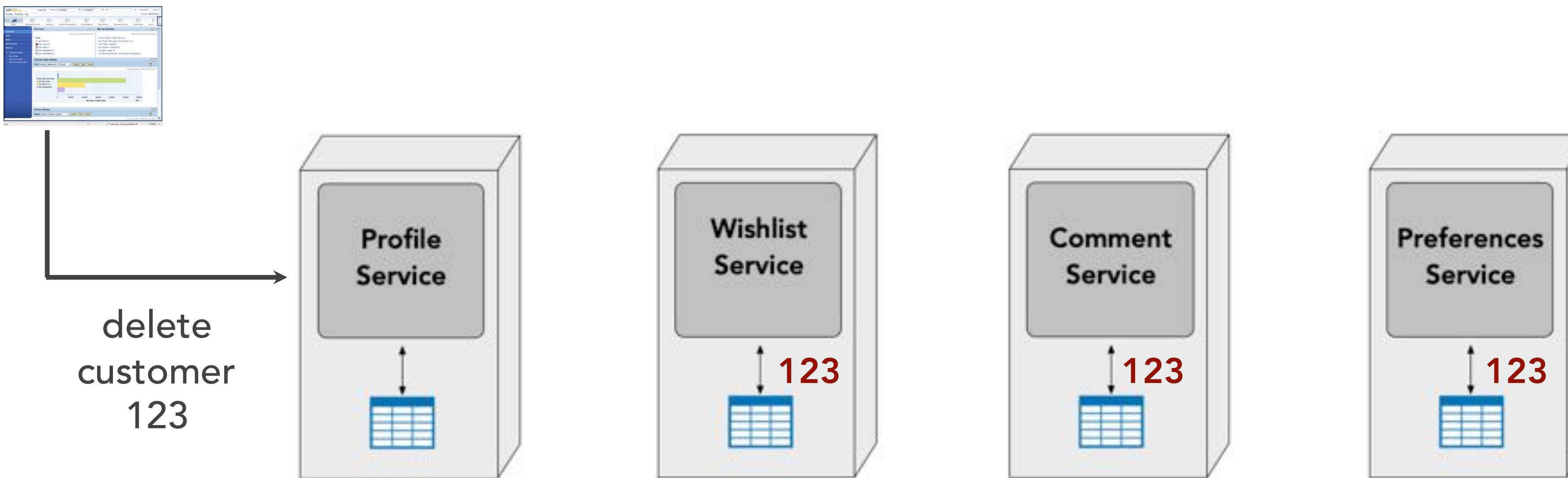
eventual consistency

the problem...



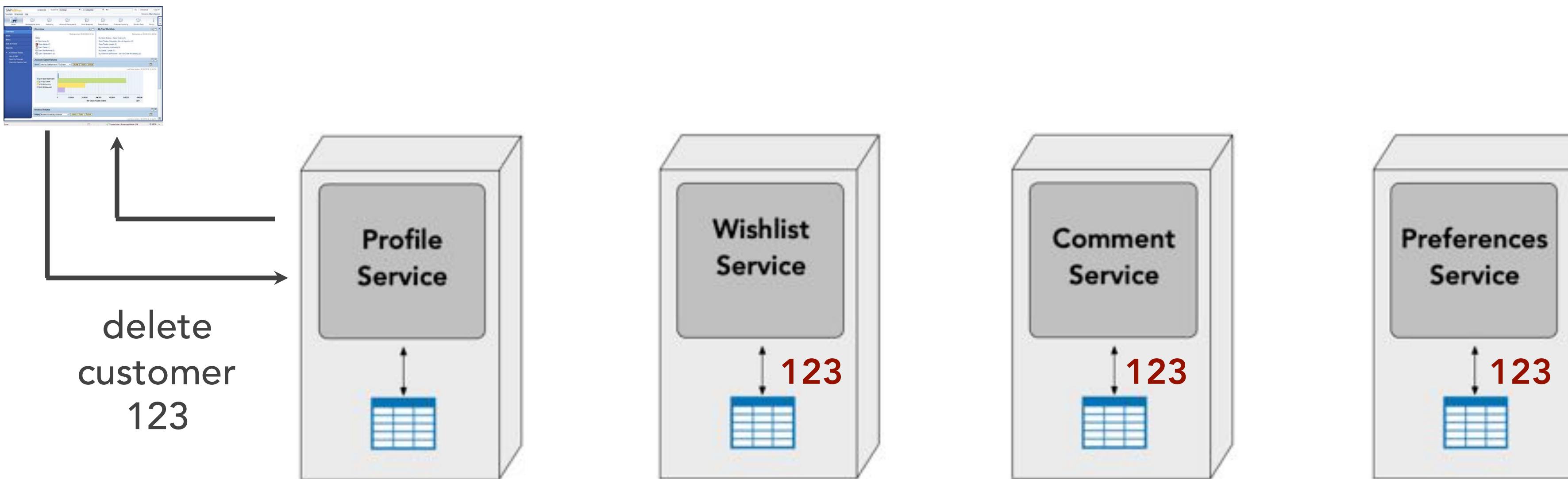
eventual consistency

the problem...



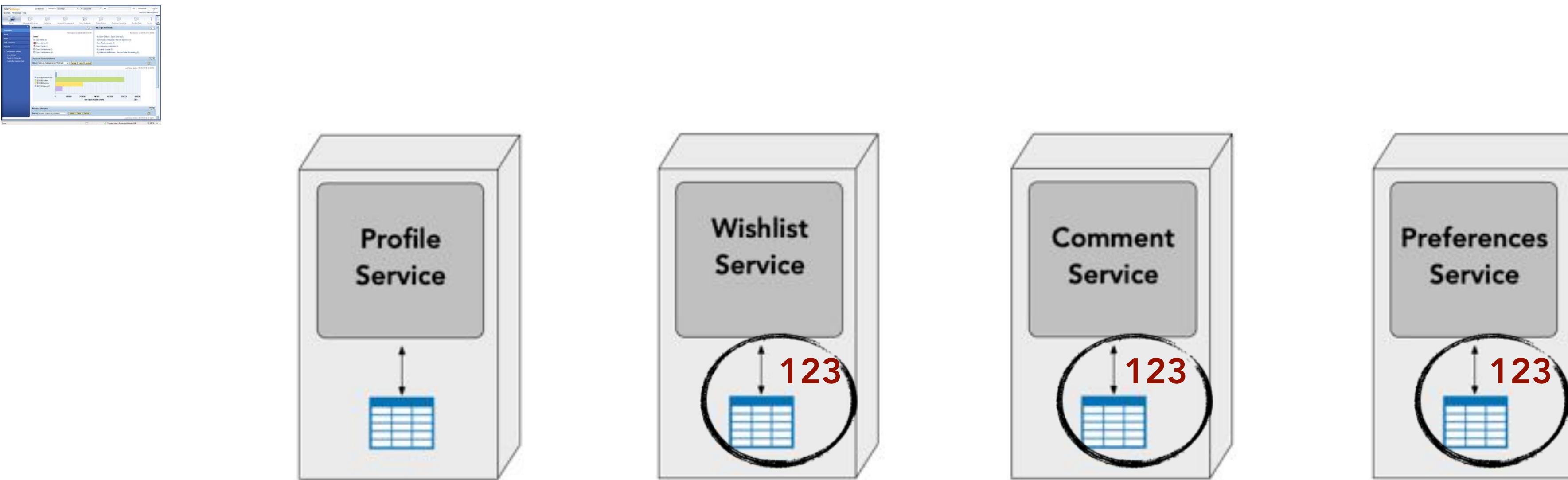
eventual consistency

the problem...



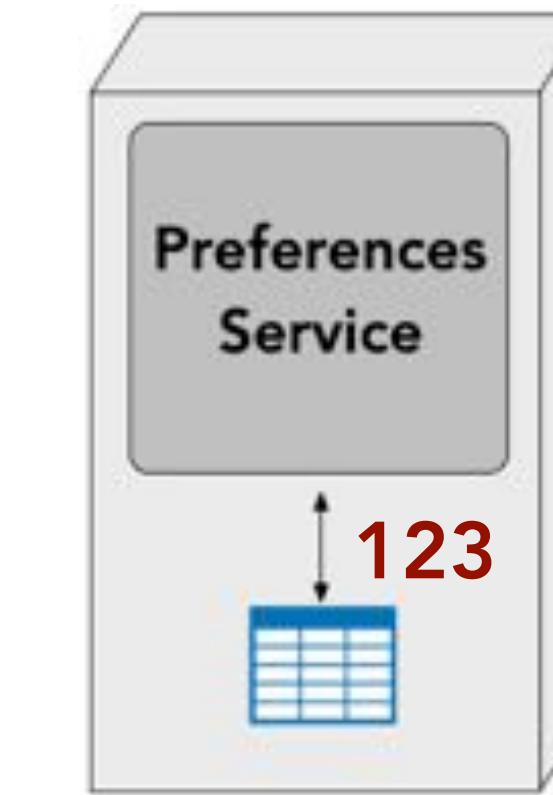
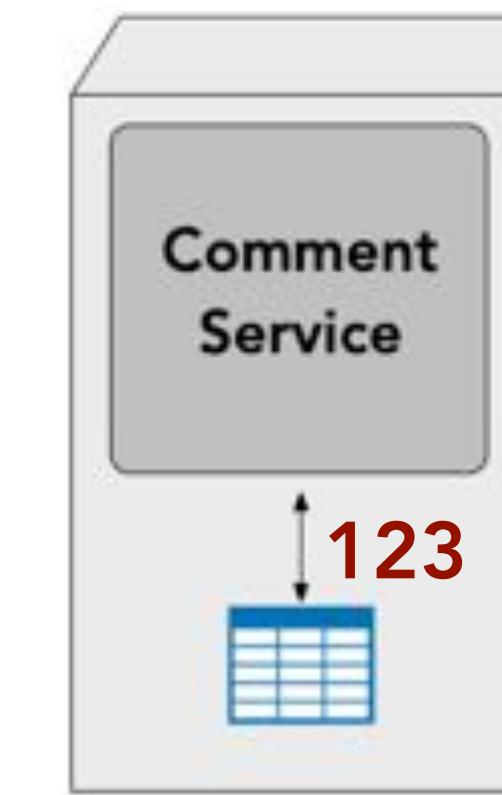
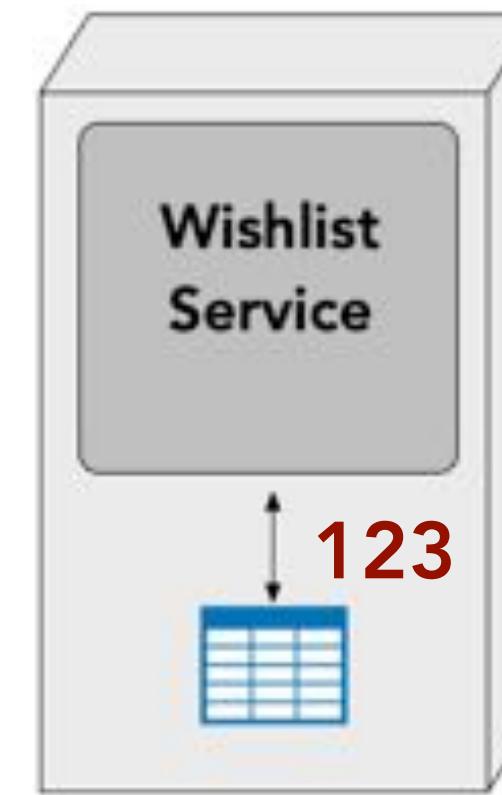
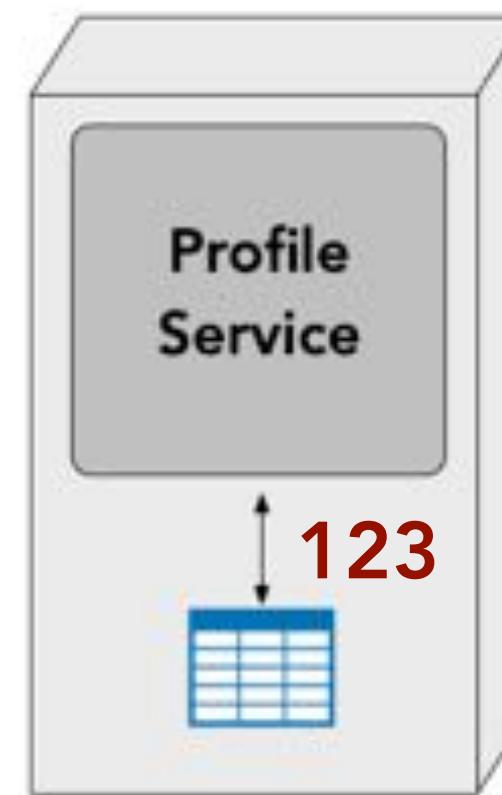
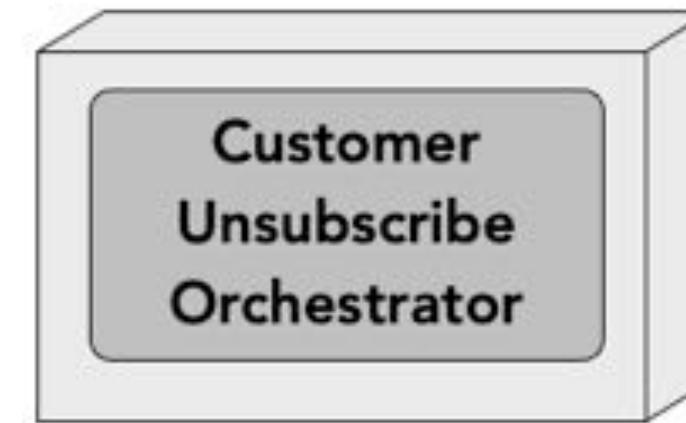
eventual consistency

the problem...



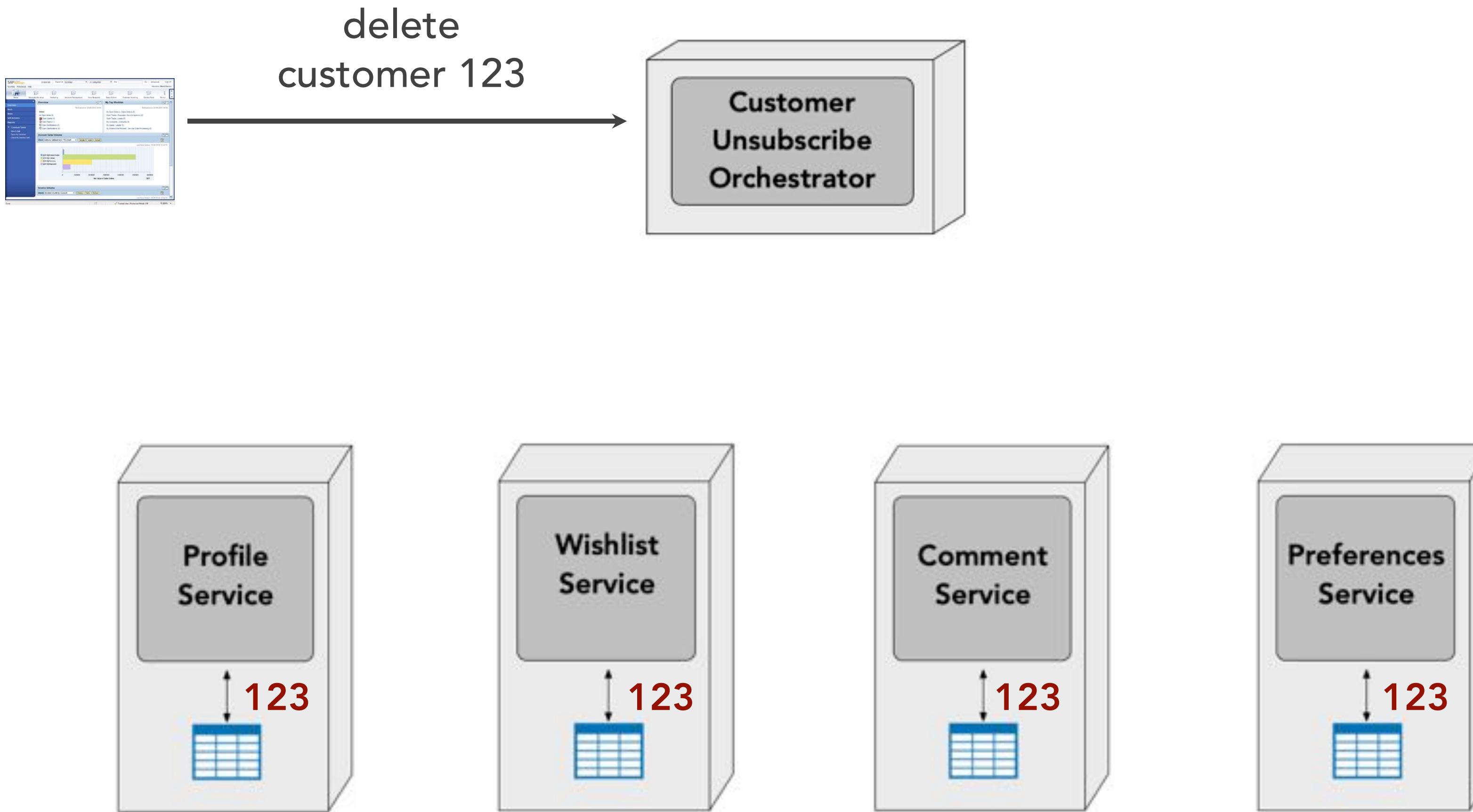
eventual consistency

orchestration



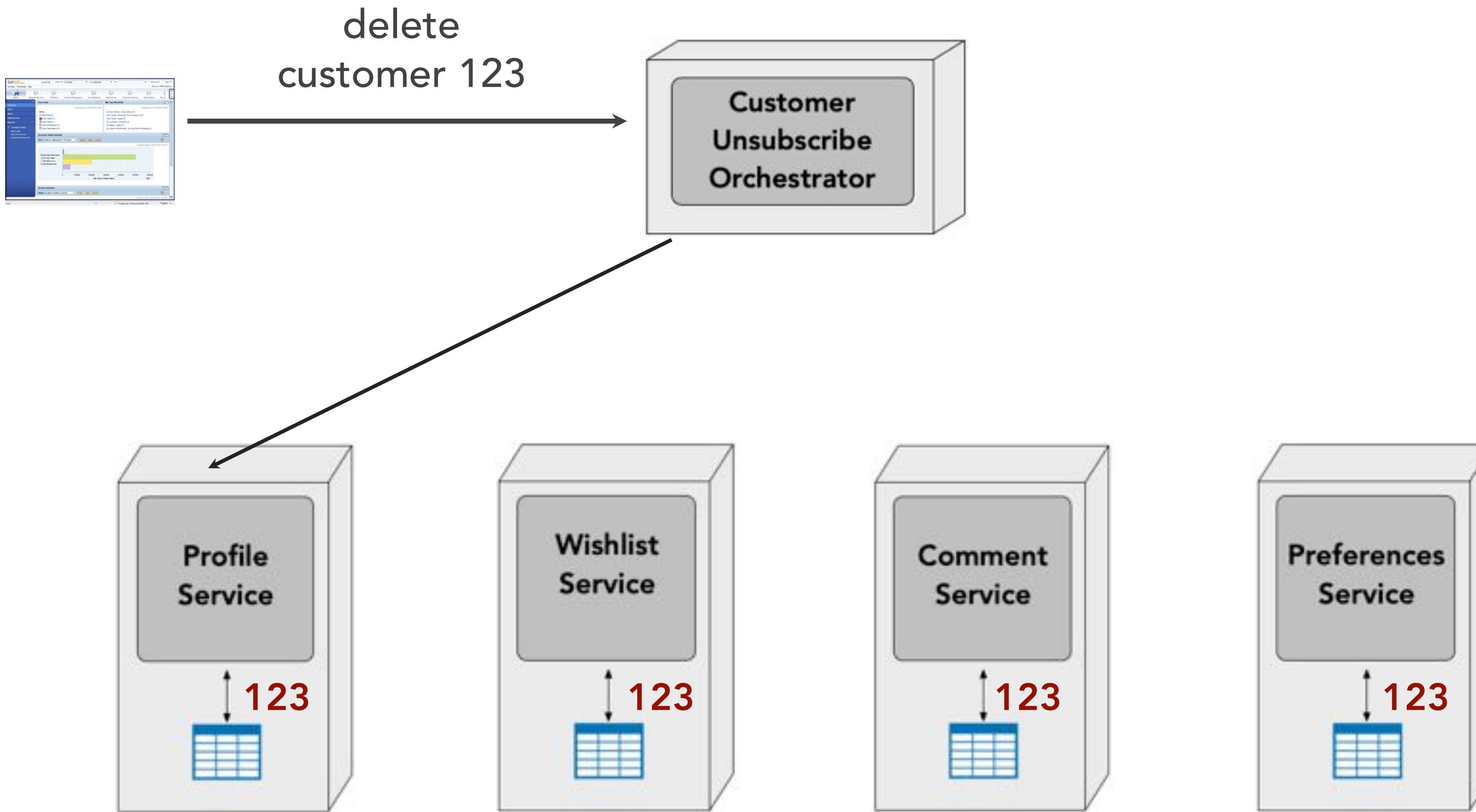
eventual consistency

orchestration



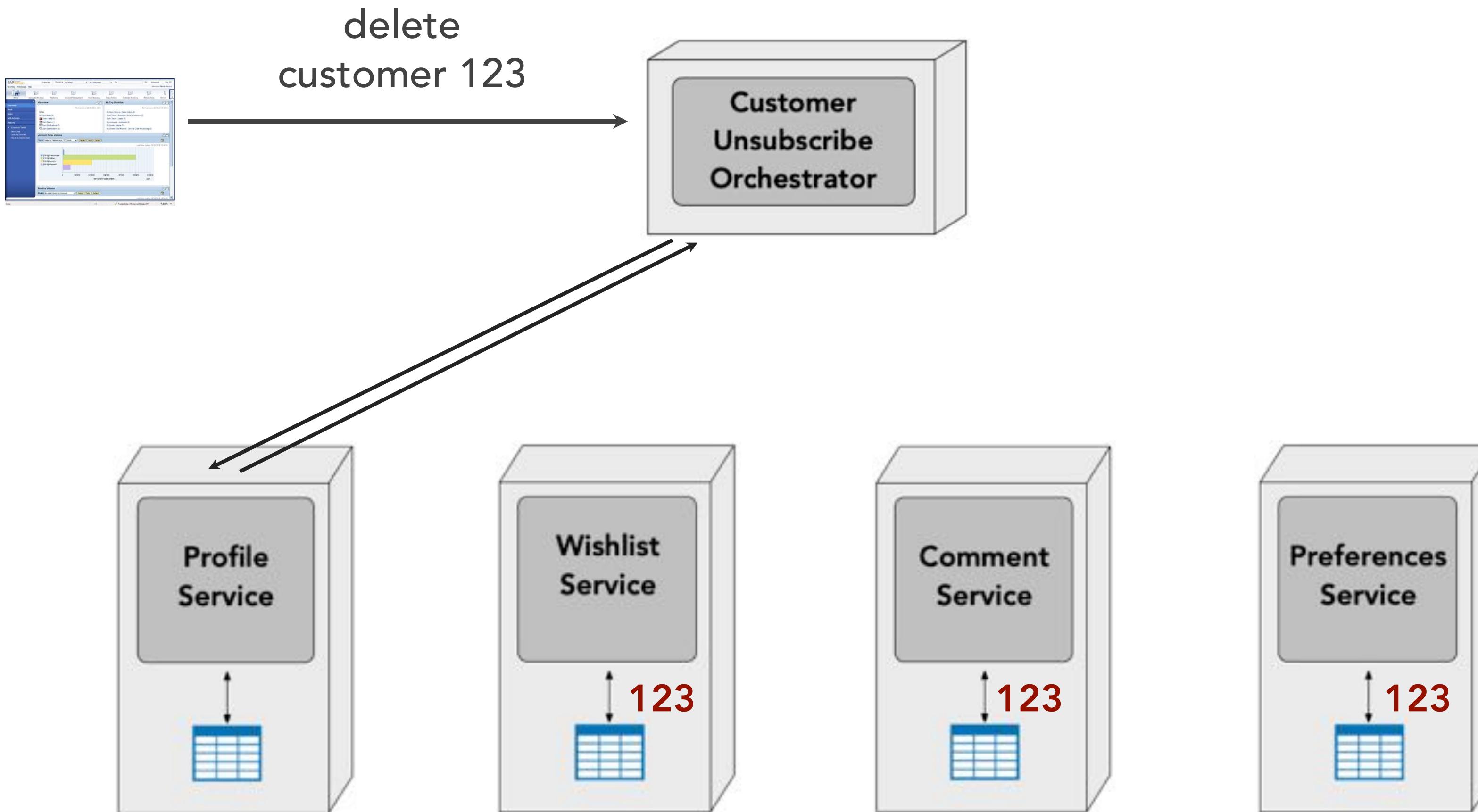
eventual consistency

orchestration



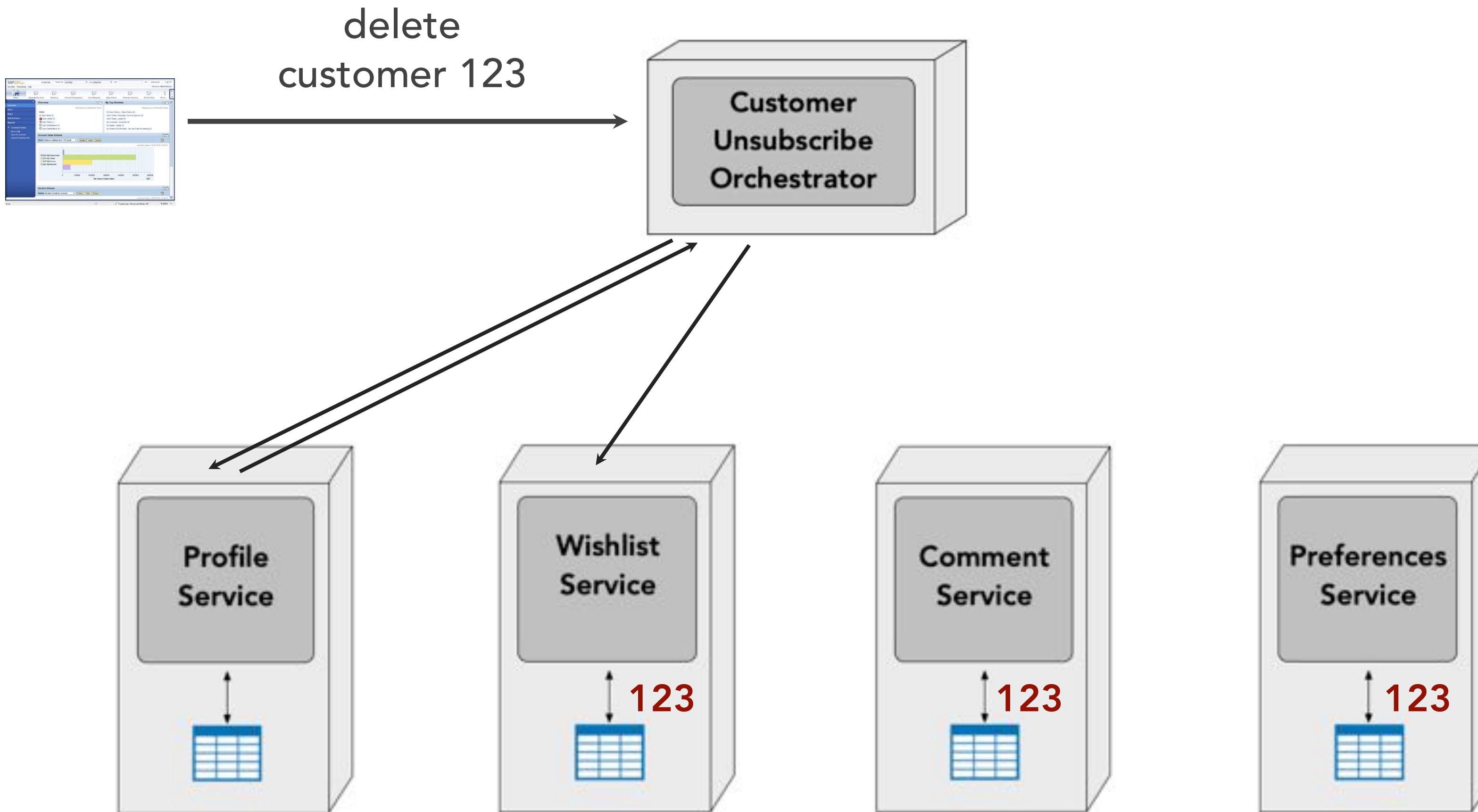
eventual consistency

orchestration



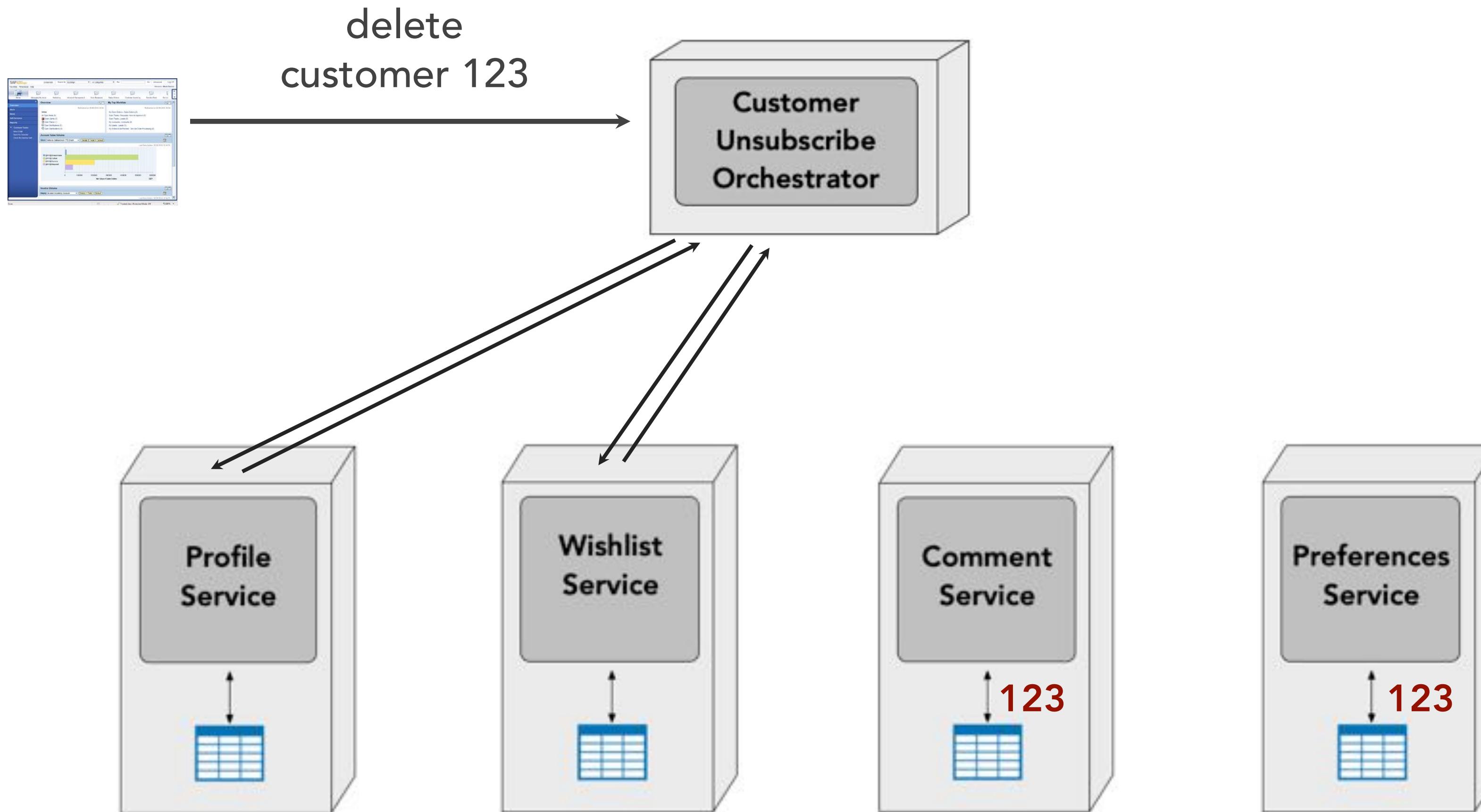
eventual consistency

orchestration



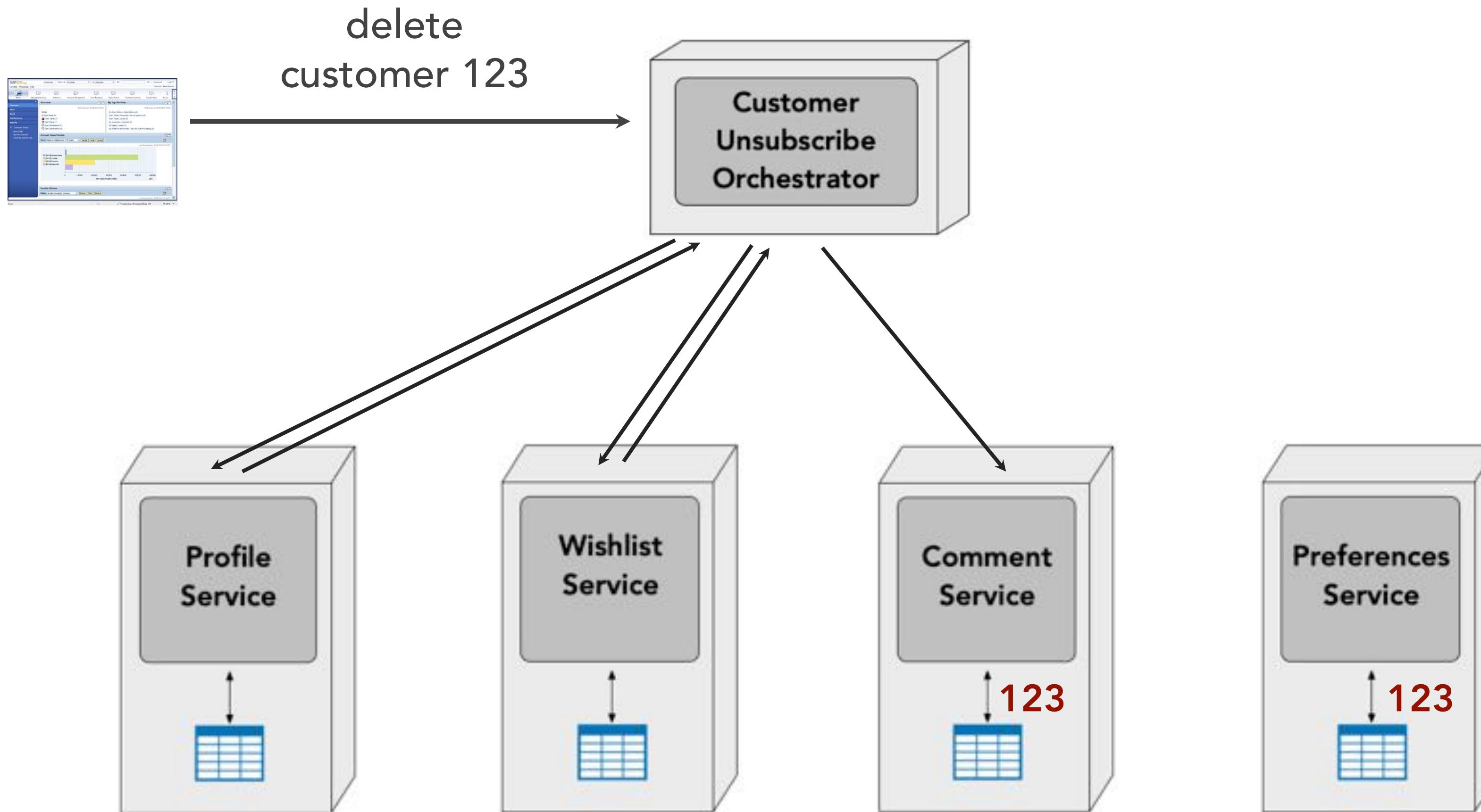
eventual consistency

orchestration



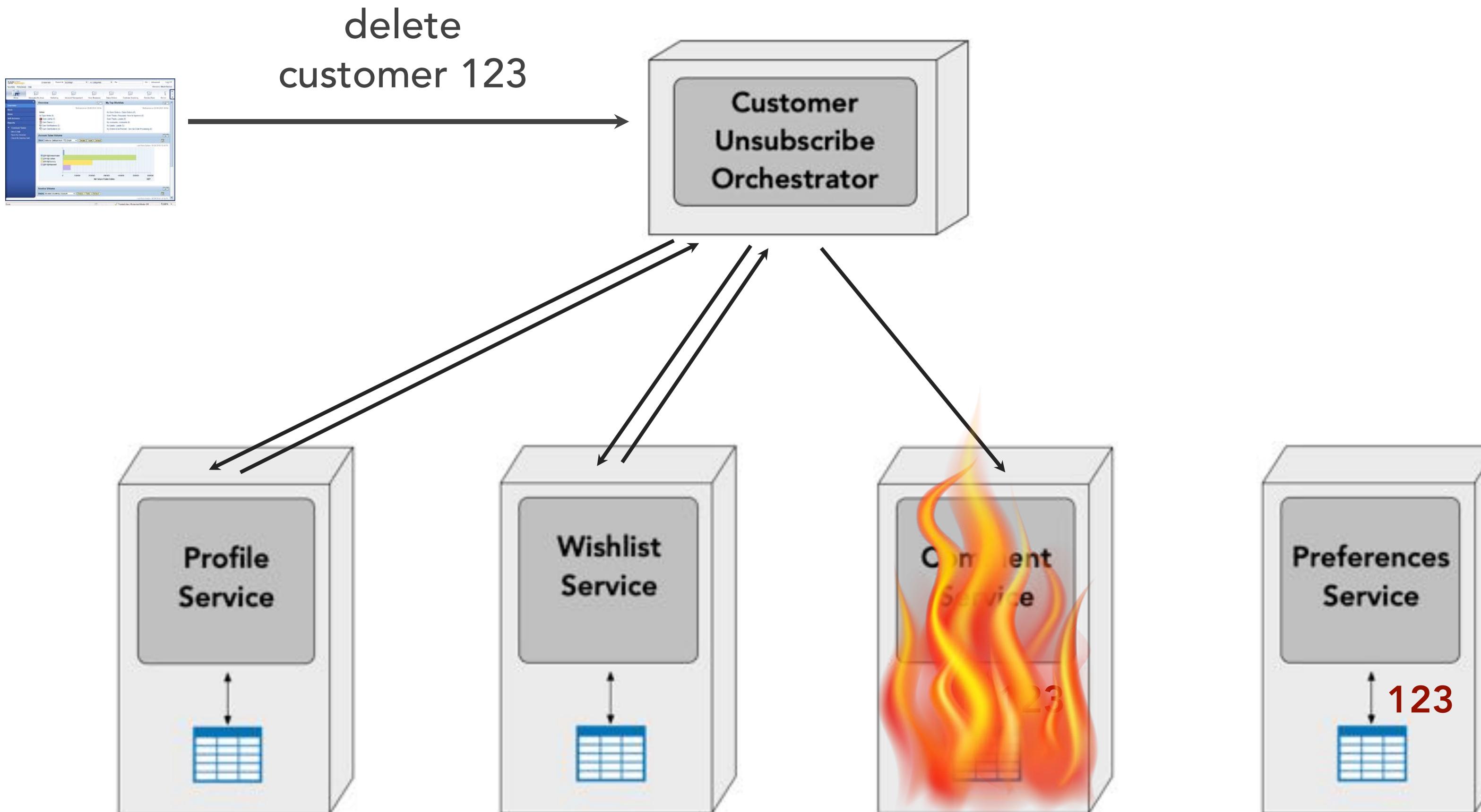
eventual consistency

orchestration



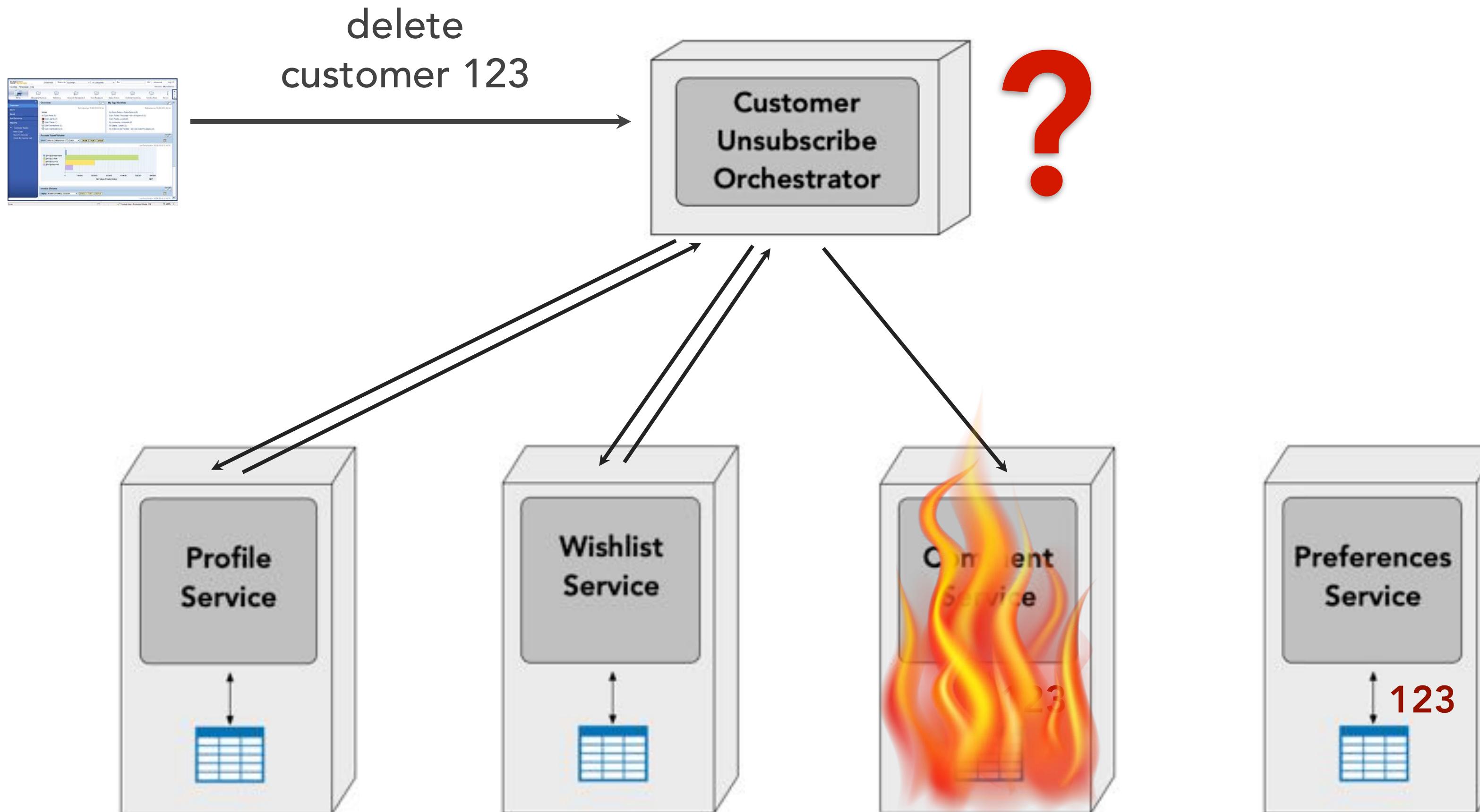
eventual consistency

orchestration



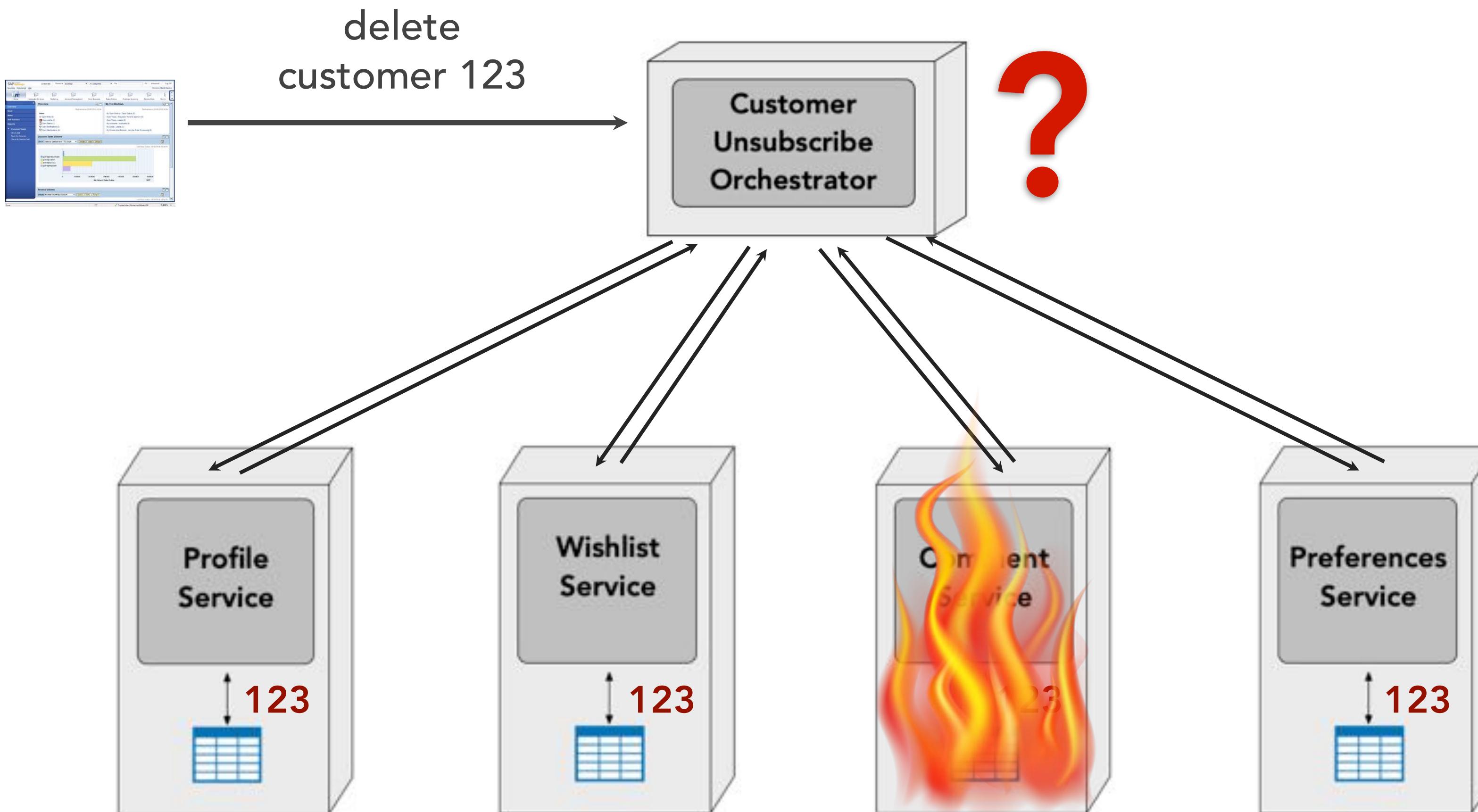
eventual consistency

orchestration



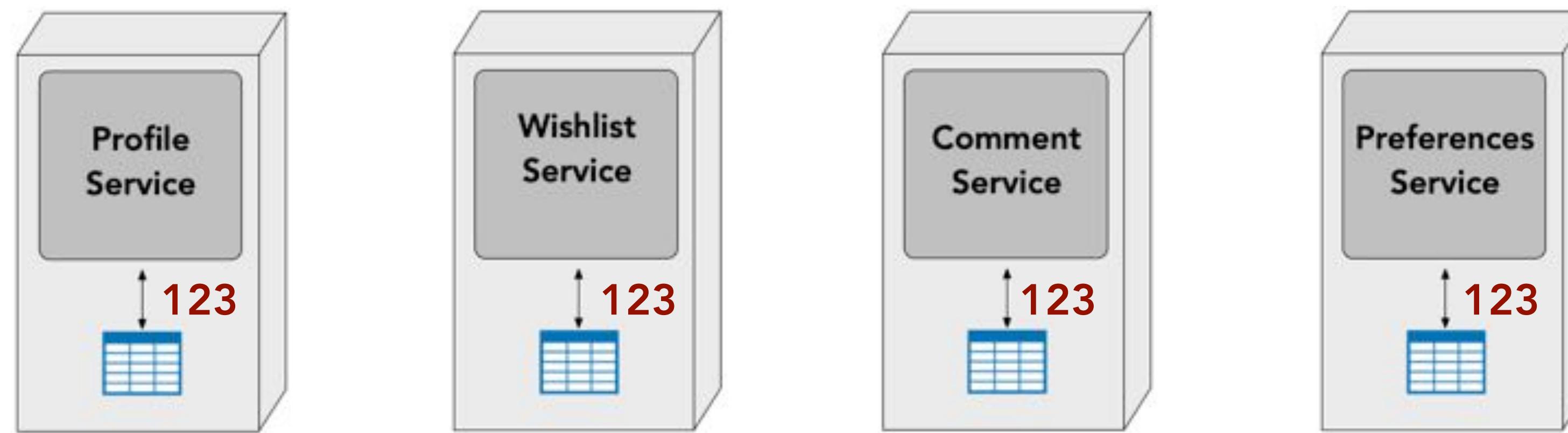
eventual consistency

orchestration



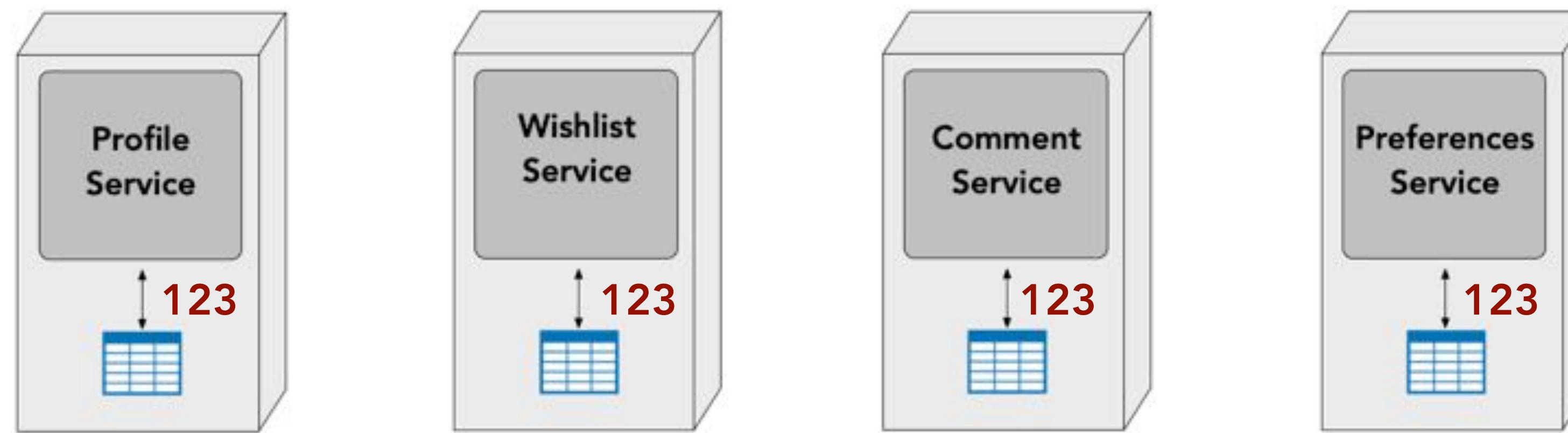
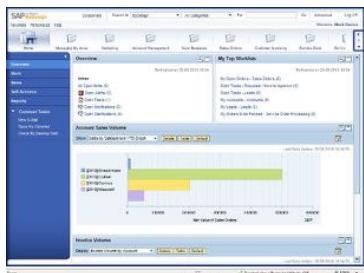
eventual consistency

background synchronization



eventual consistency

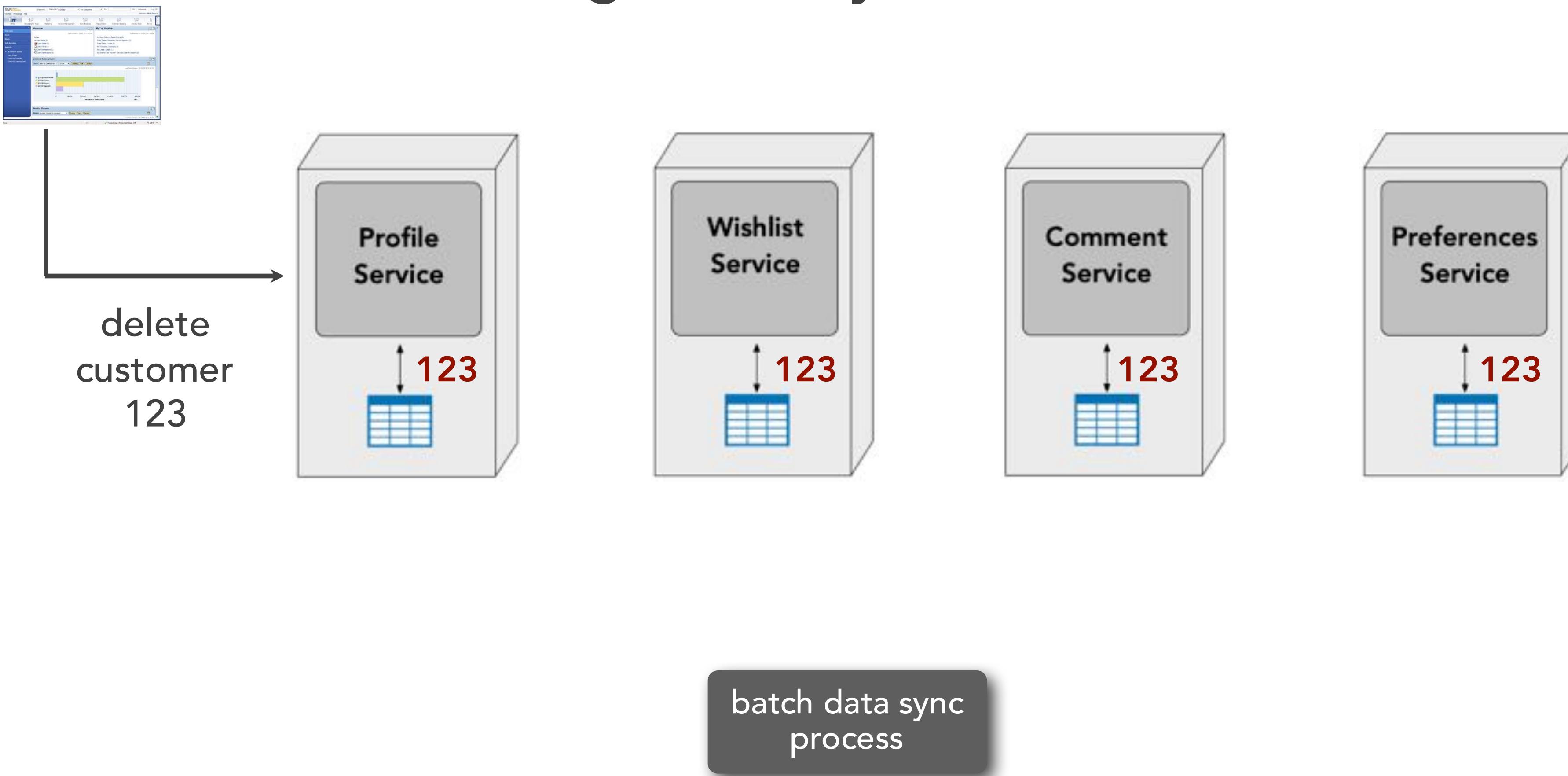
background synchronization



batch data sync
process

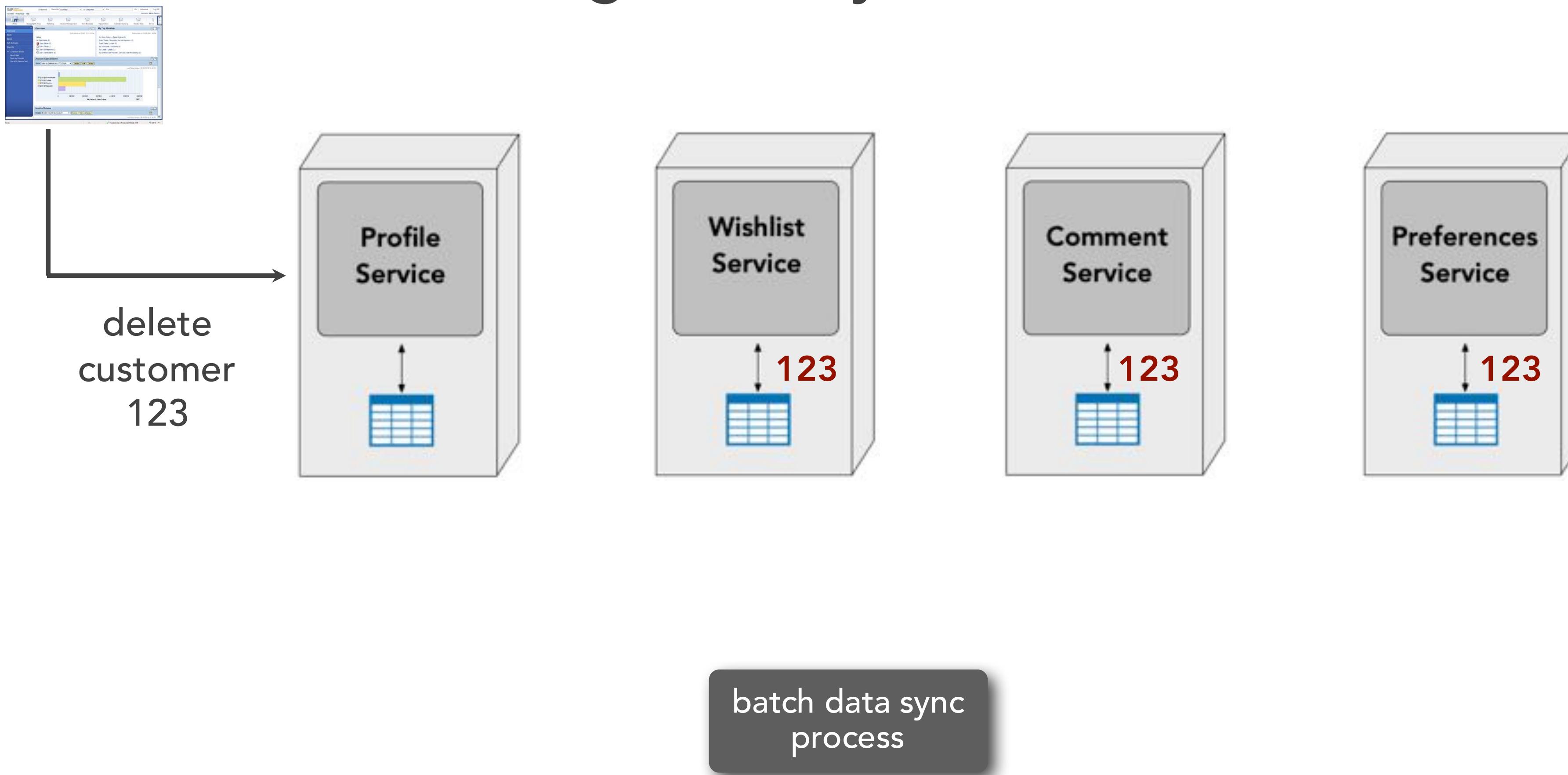
eventual consistency

background synchronization



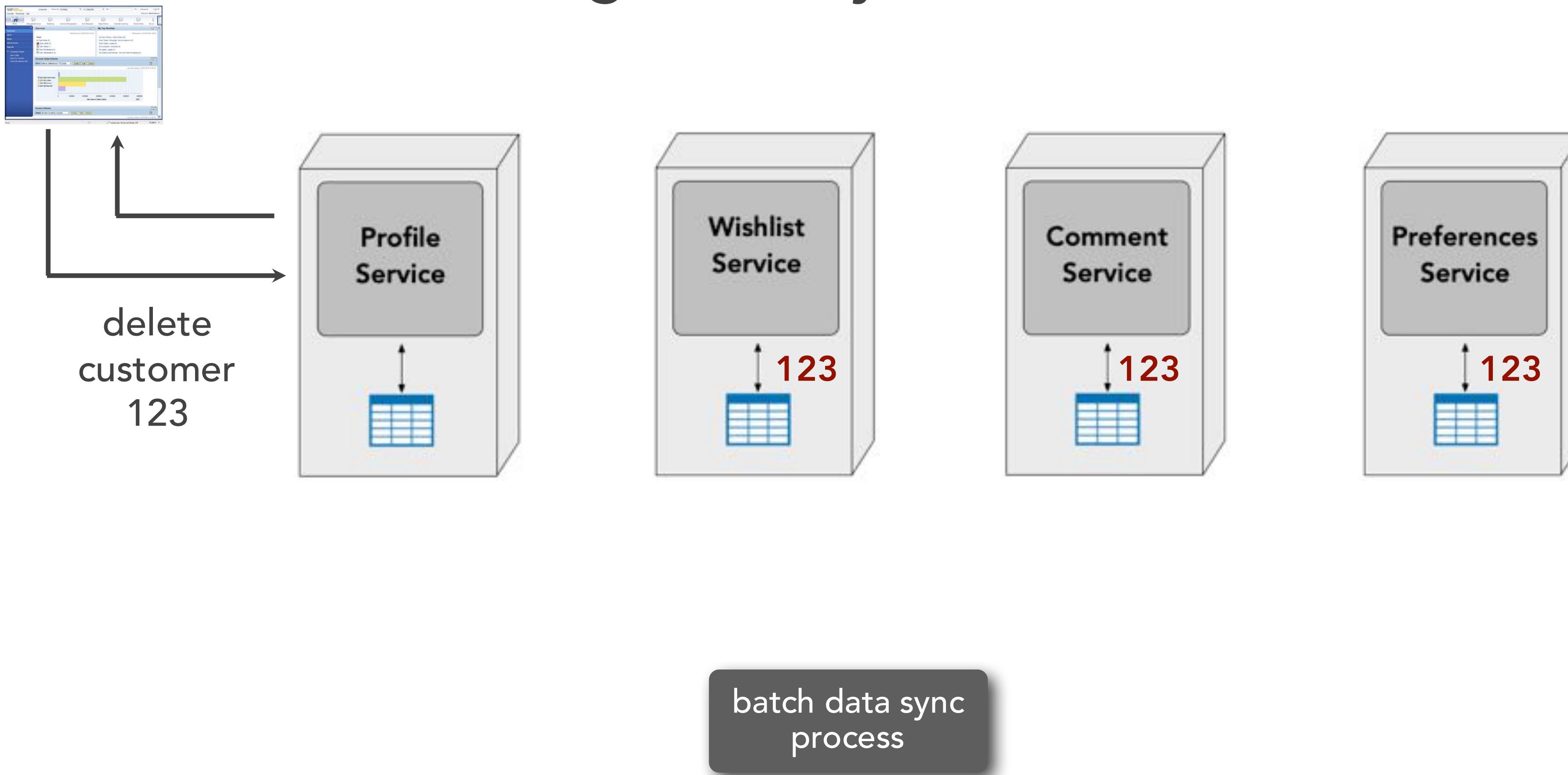
eventual consistency

background synchronization



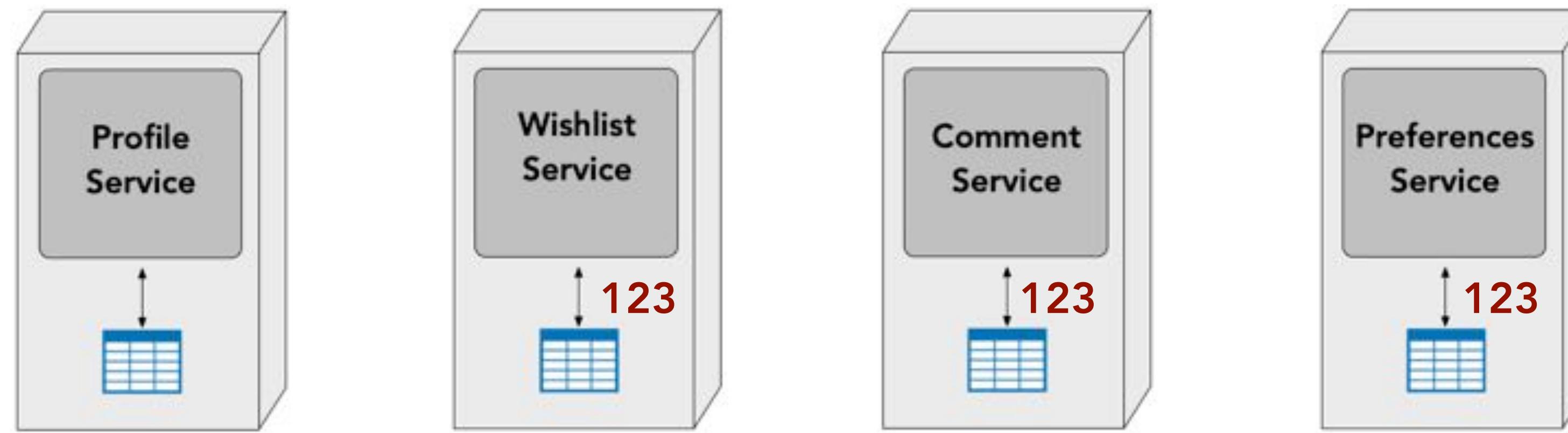
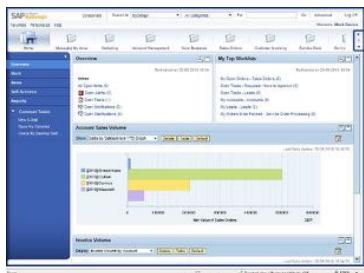
eventual consistency

background synchronization



eventual consistency

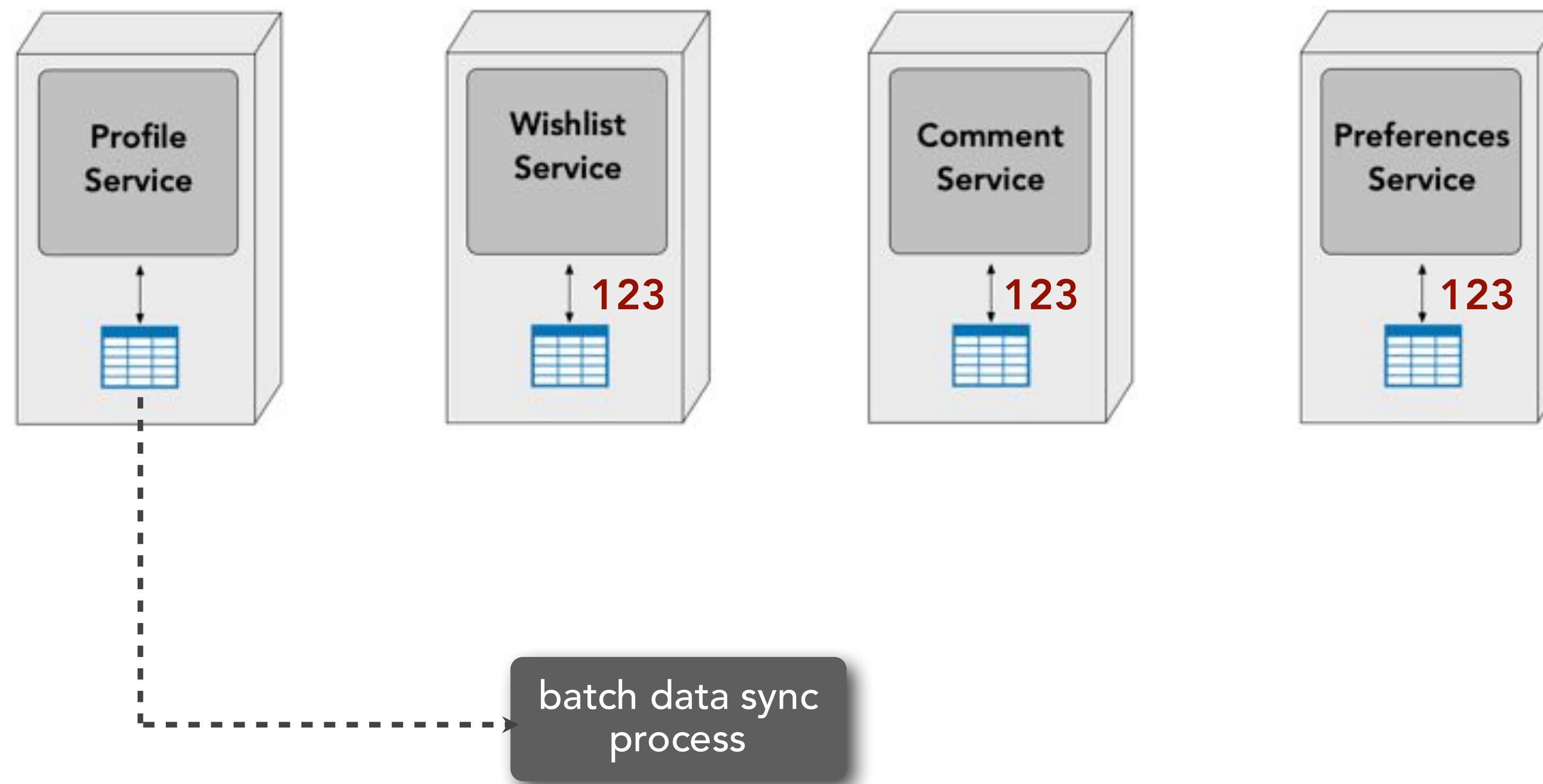
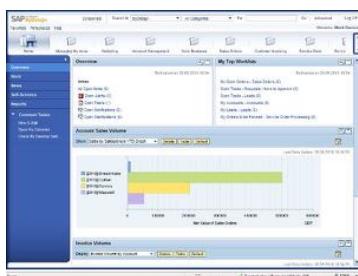
background synchronization



batch data sync
process

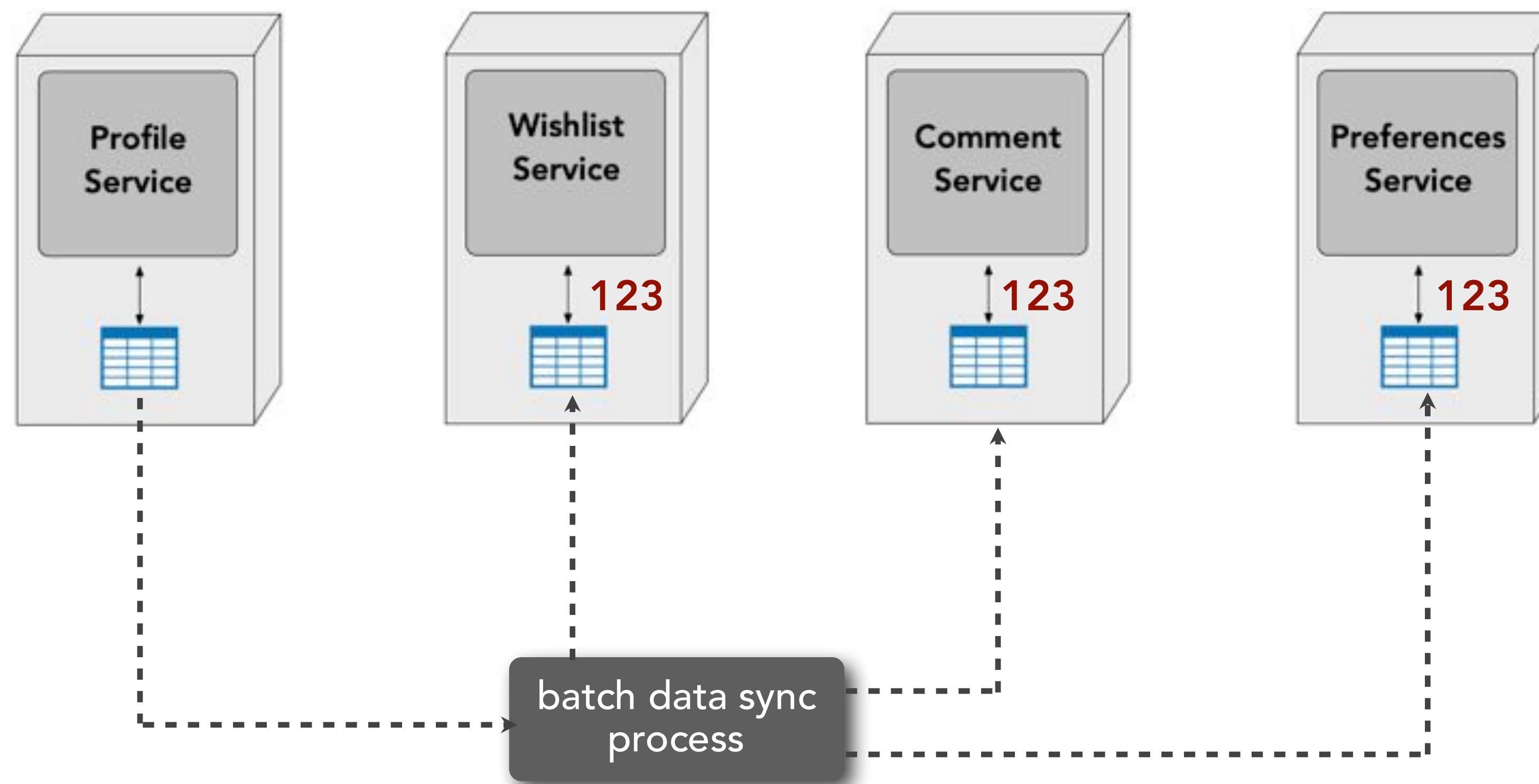
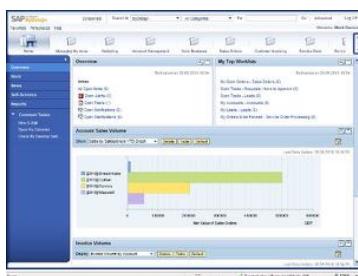
eventual consistency

background synchronization



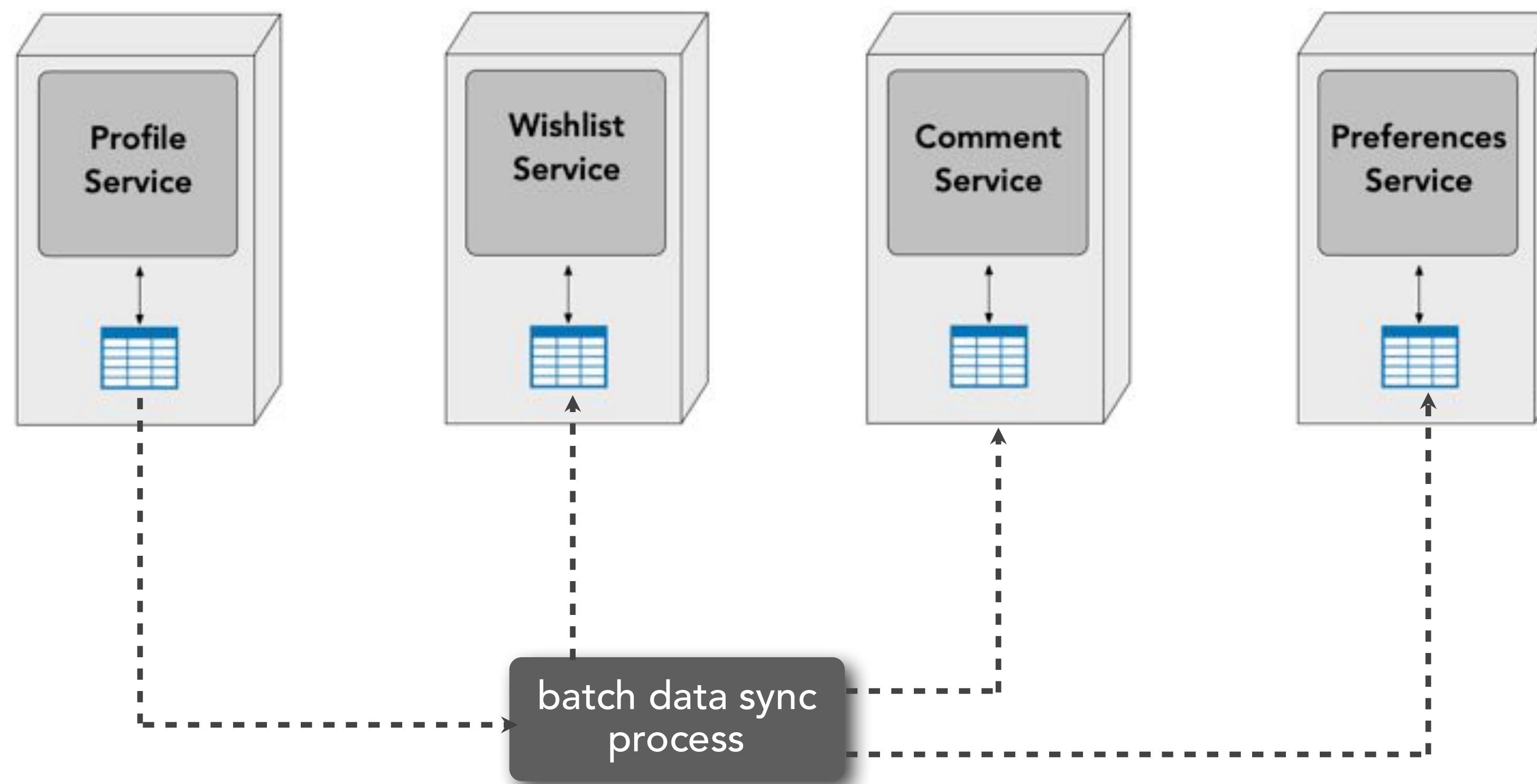
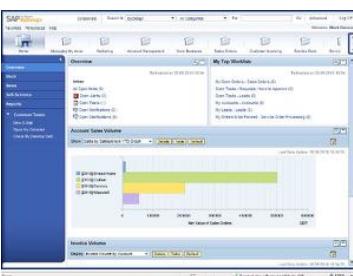
eventual consistency

background synchronization



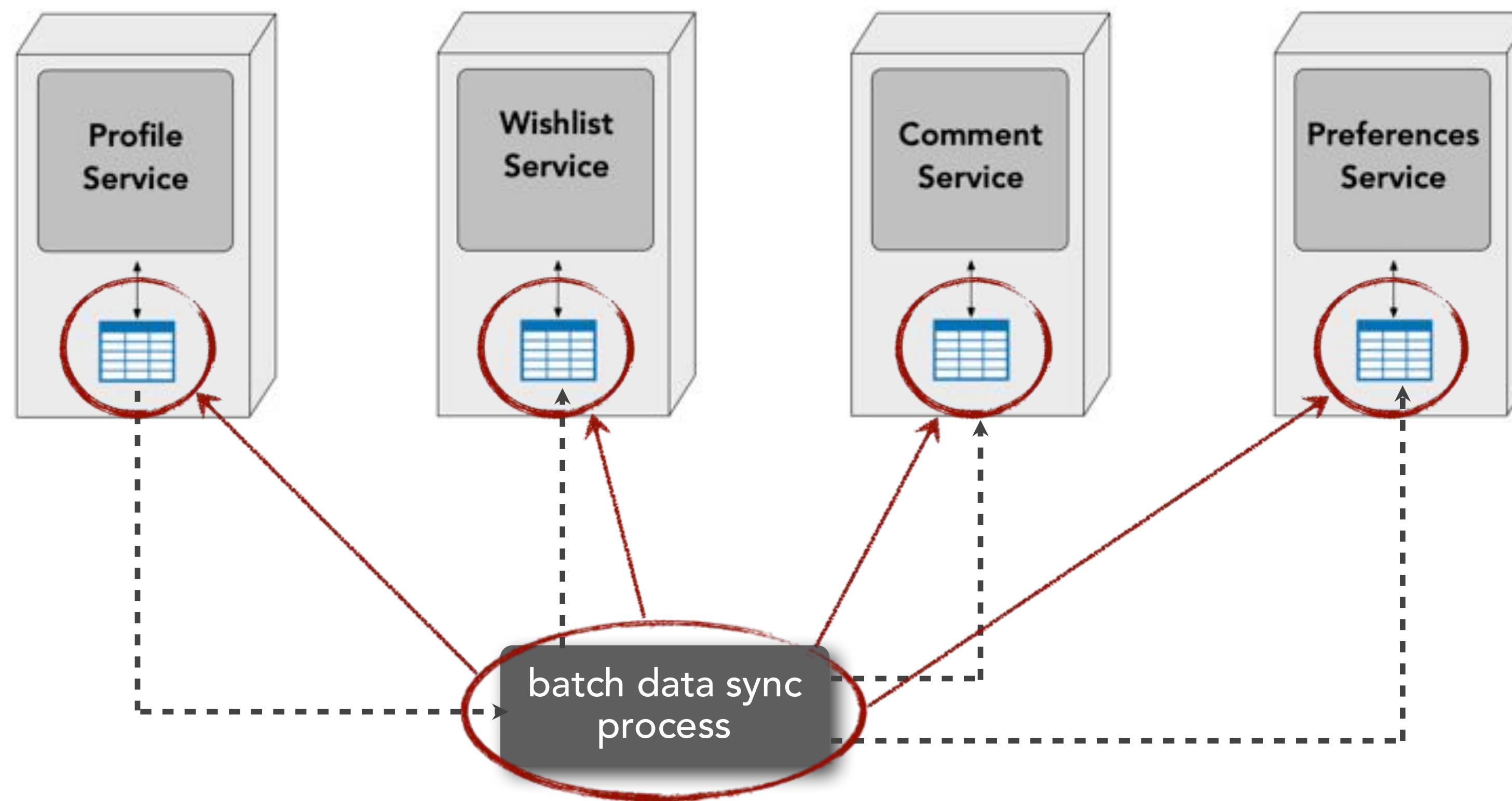
eventual consistency

background synchronization



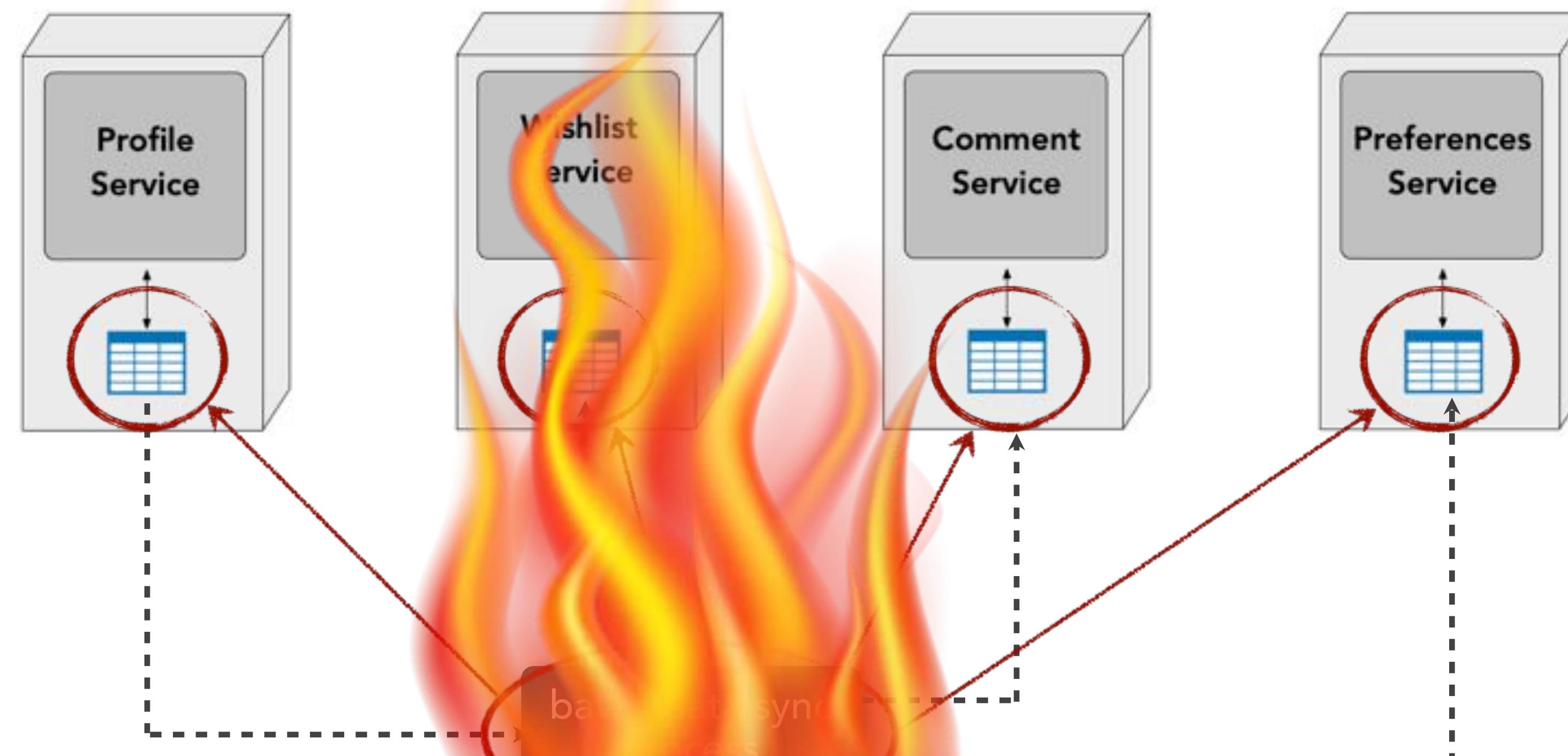
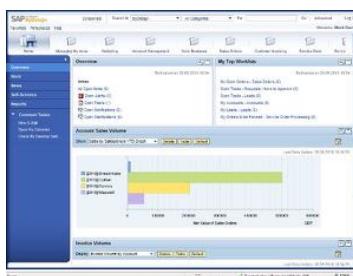
eventual consistency

background synchronization



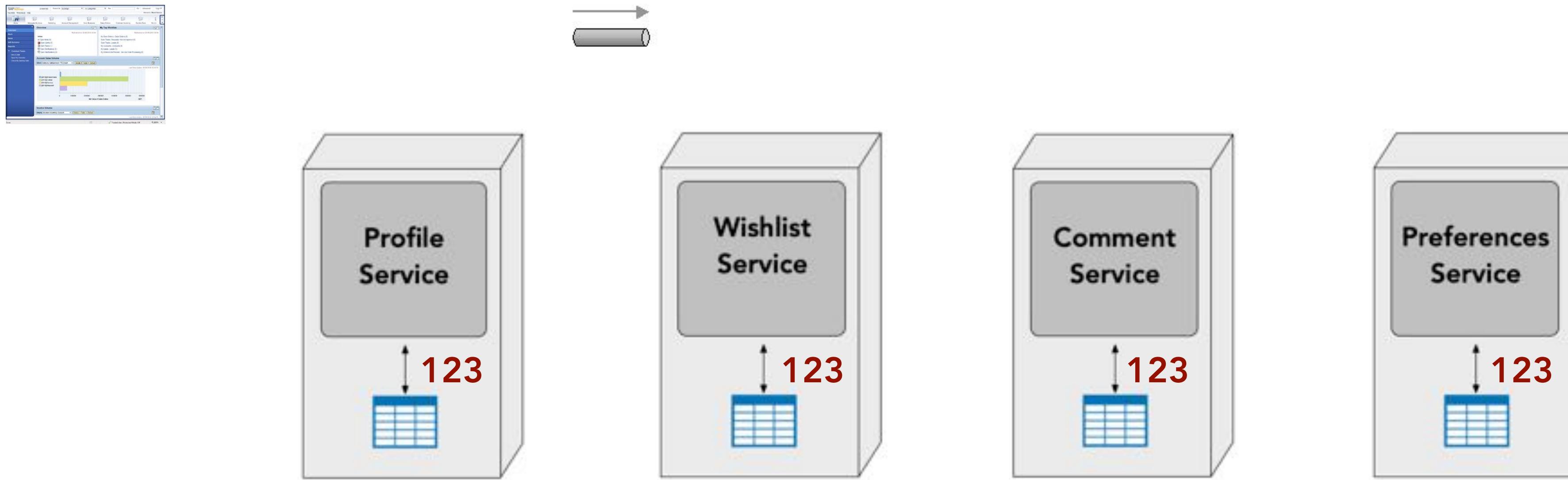
eventual consistency

background synchronization



eventual consistency

event-based synchronization



eventual consistency

event-based synchronization



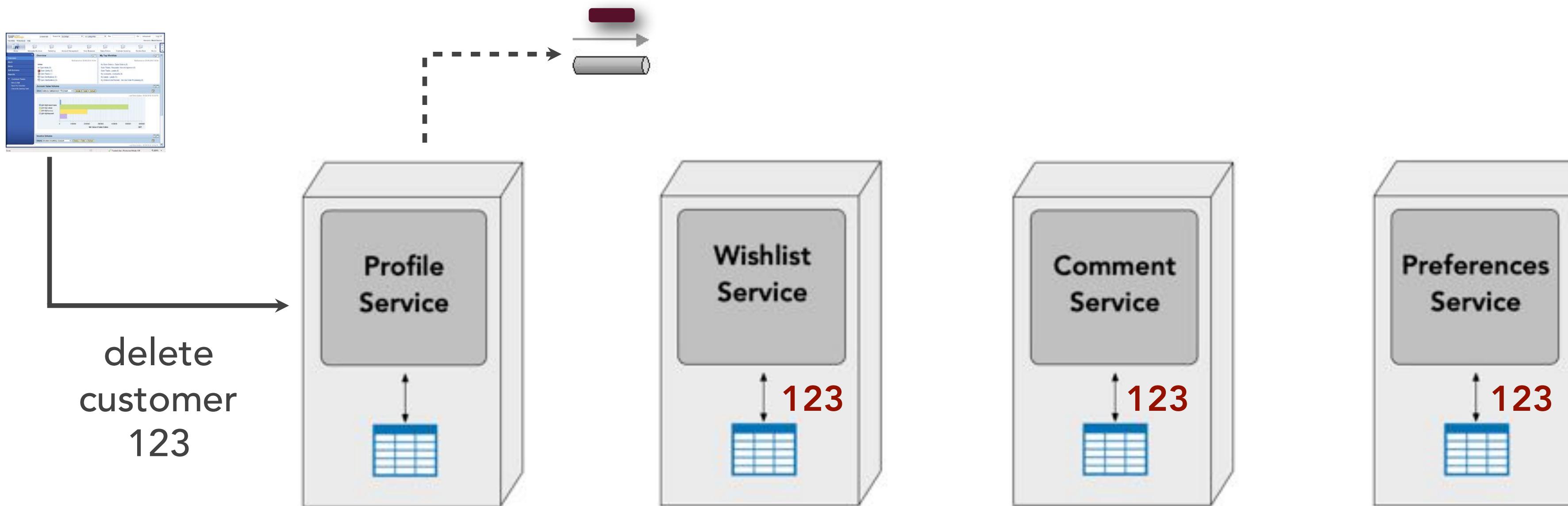
eventual consistency

event-based synchronization



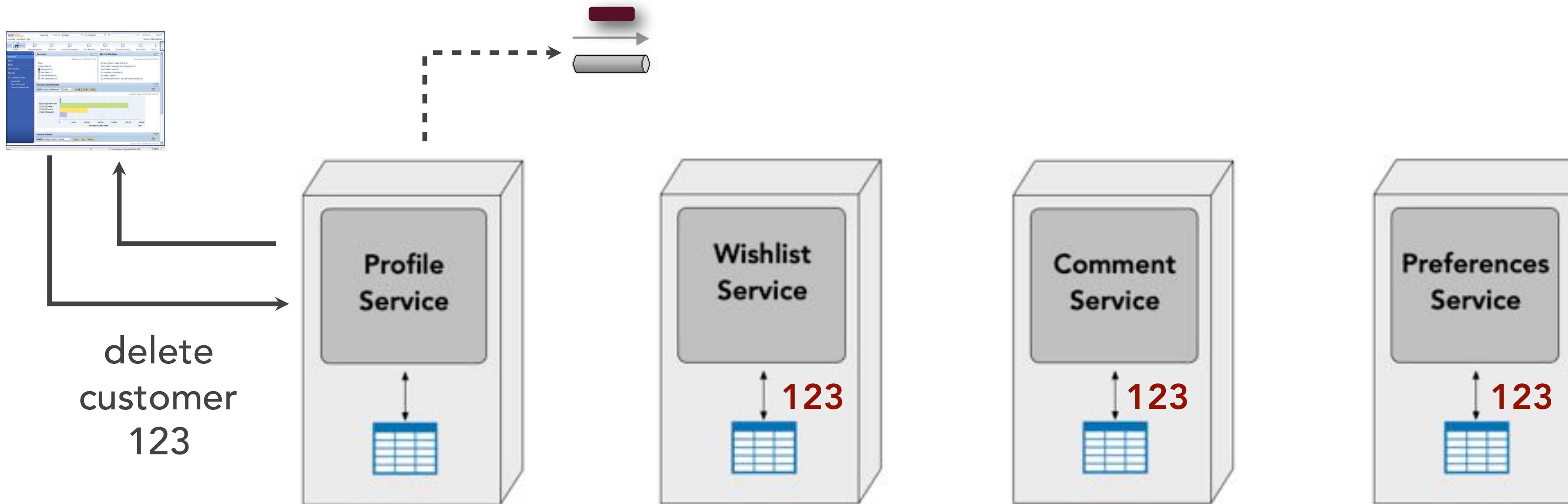
eventual consistency

event-based synchronization



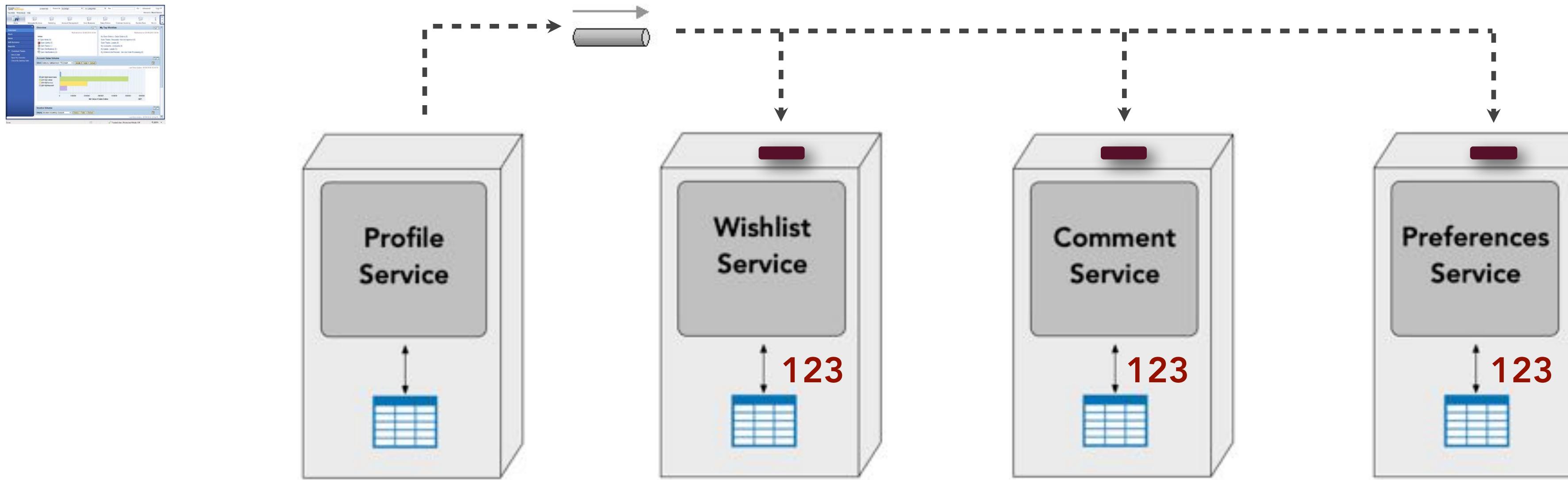
eventual consistency

event-based synchronization



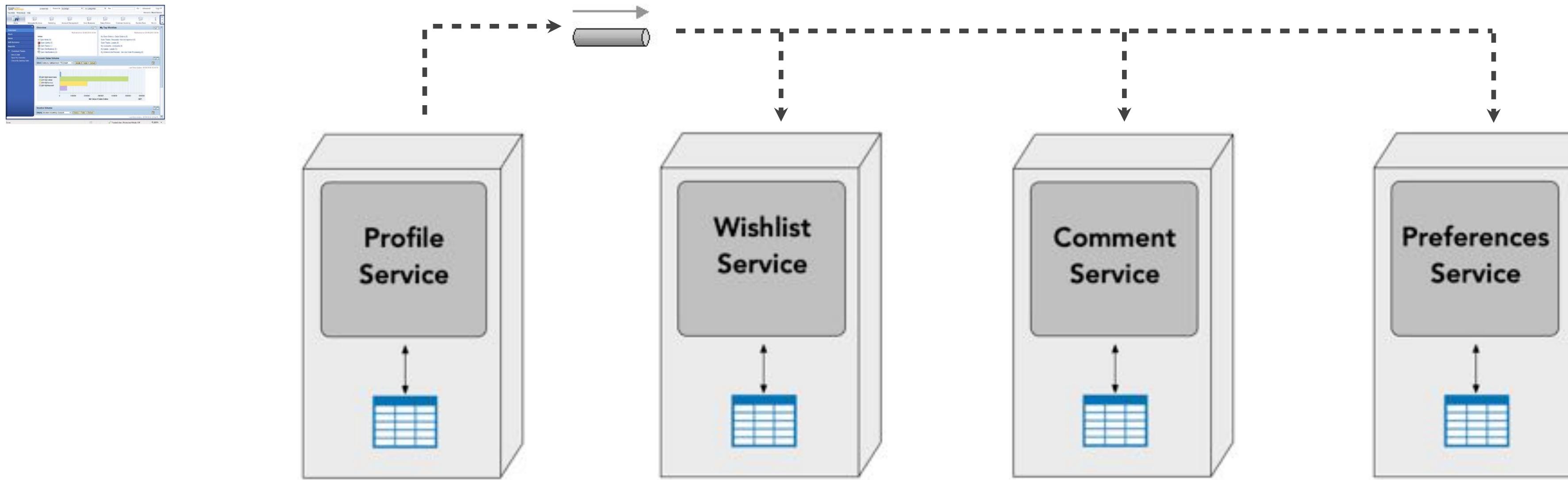
eventual consistency

event-based synchronization



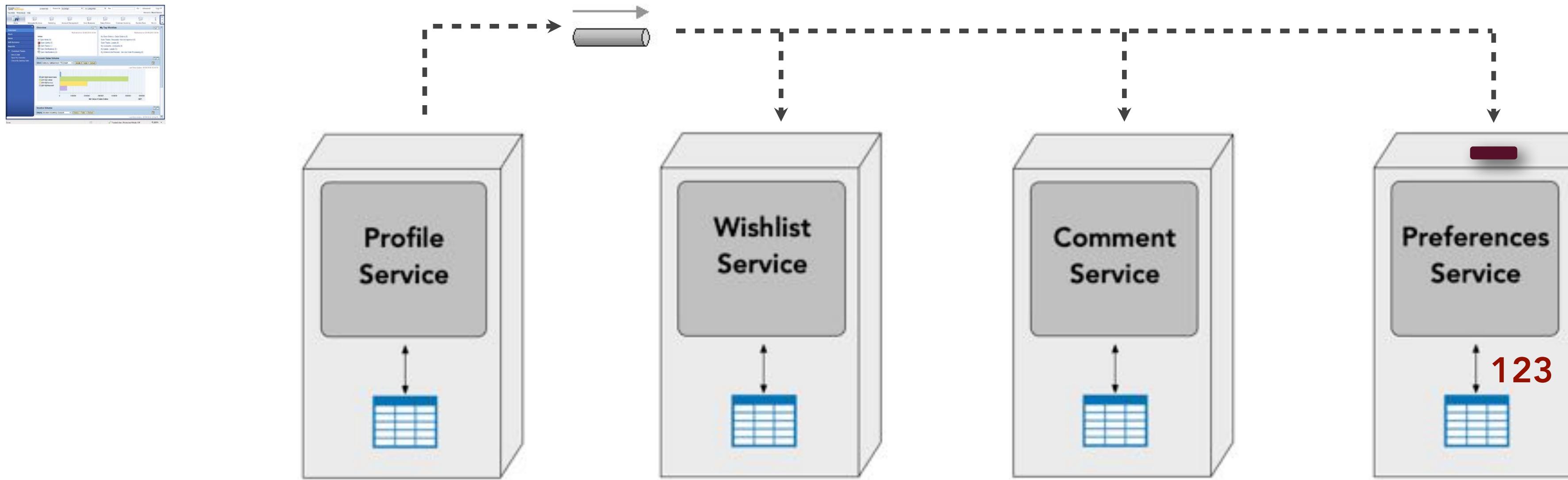
eventual consistency

event-based synchronization



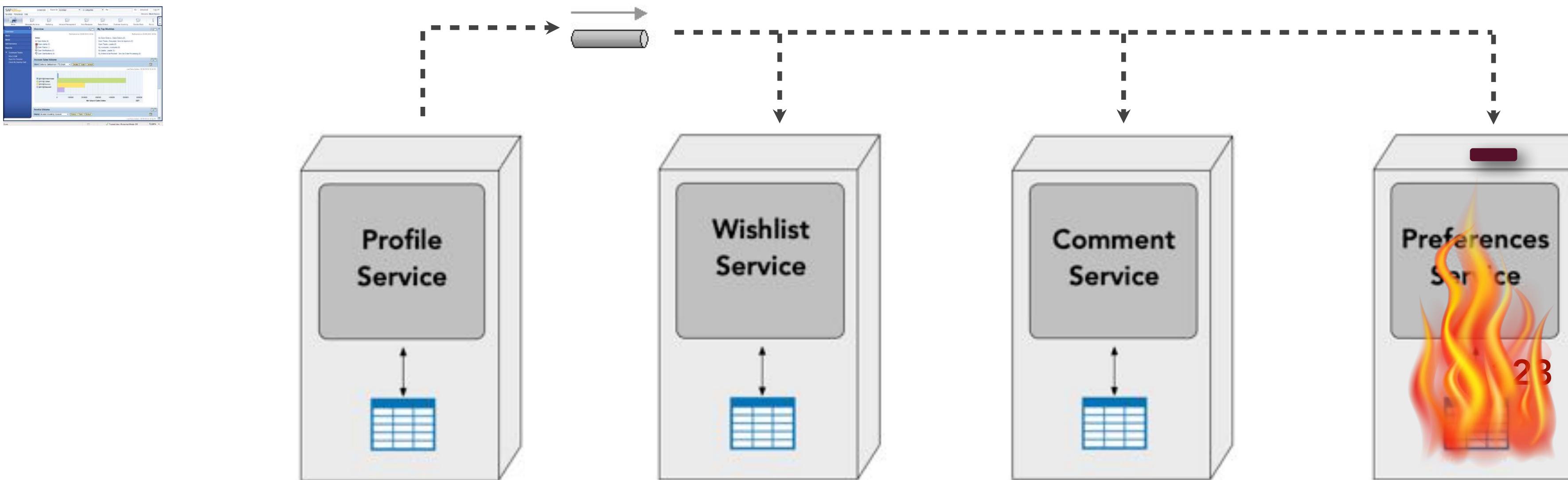
eventual consistency

event-based synchronization



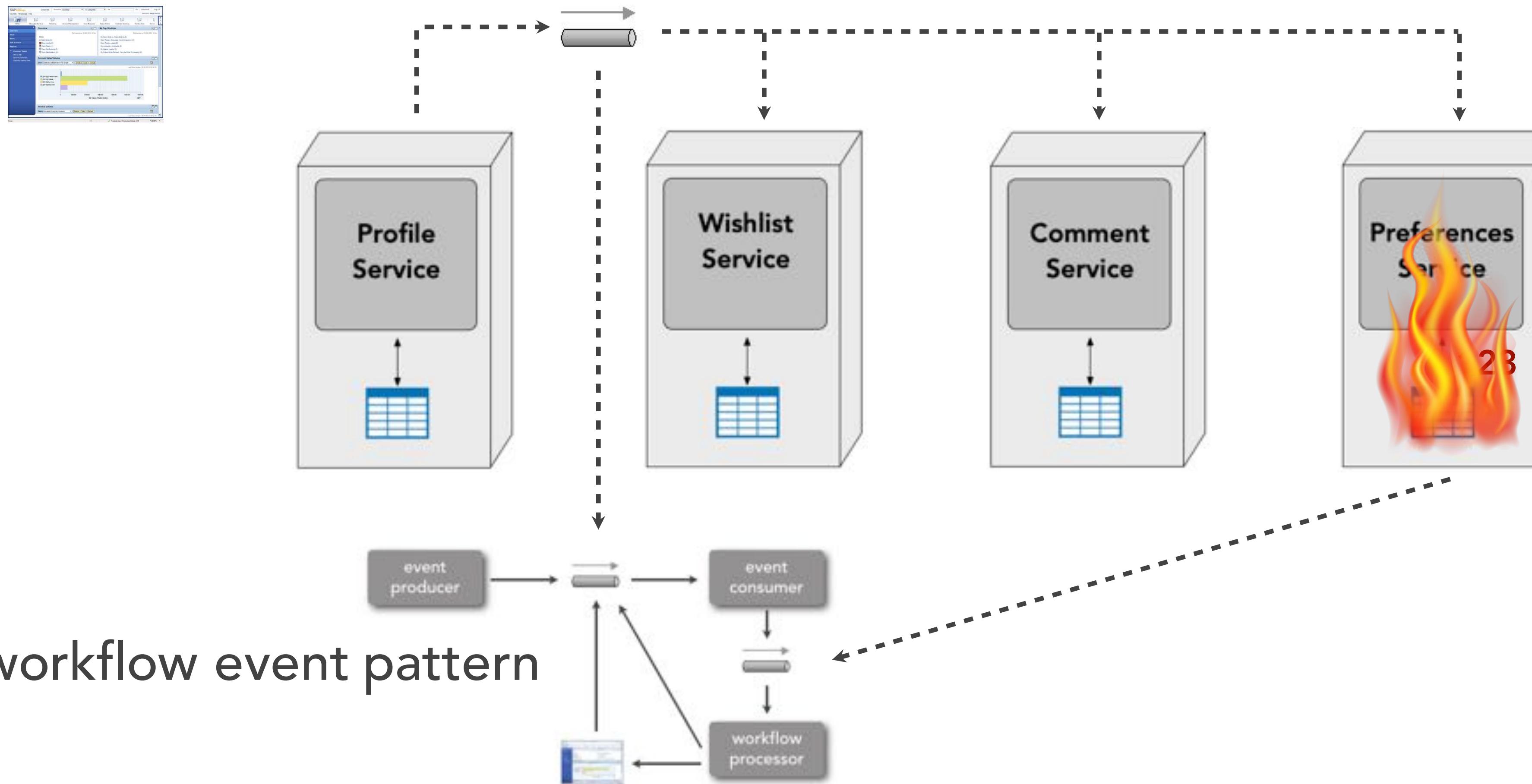
eventual consistency

event-based synchronization



eventual consistency

event-based synchronization



transactions | X ?

operational | analytical ?

Zhamak Dehghani

ThoughtWorks

Principle Consultant



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

@zhamakd

Data Mesh



WHAT'S BEYOND
THE LAKE?

Big Data | Analytics | ML Architecture

CURRENT APPROACHES

Running the Business

Capabilities & Services

Operational Data

Products & Applications

Communications

Infrastructure



Optimizing the Business

Insights

Analytical Data

Reports | ML Models

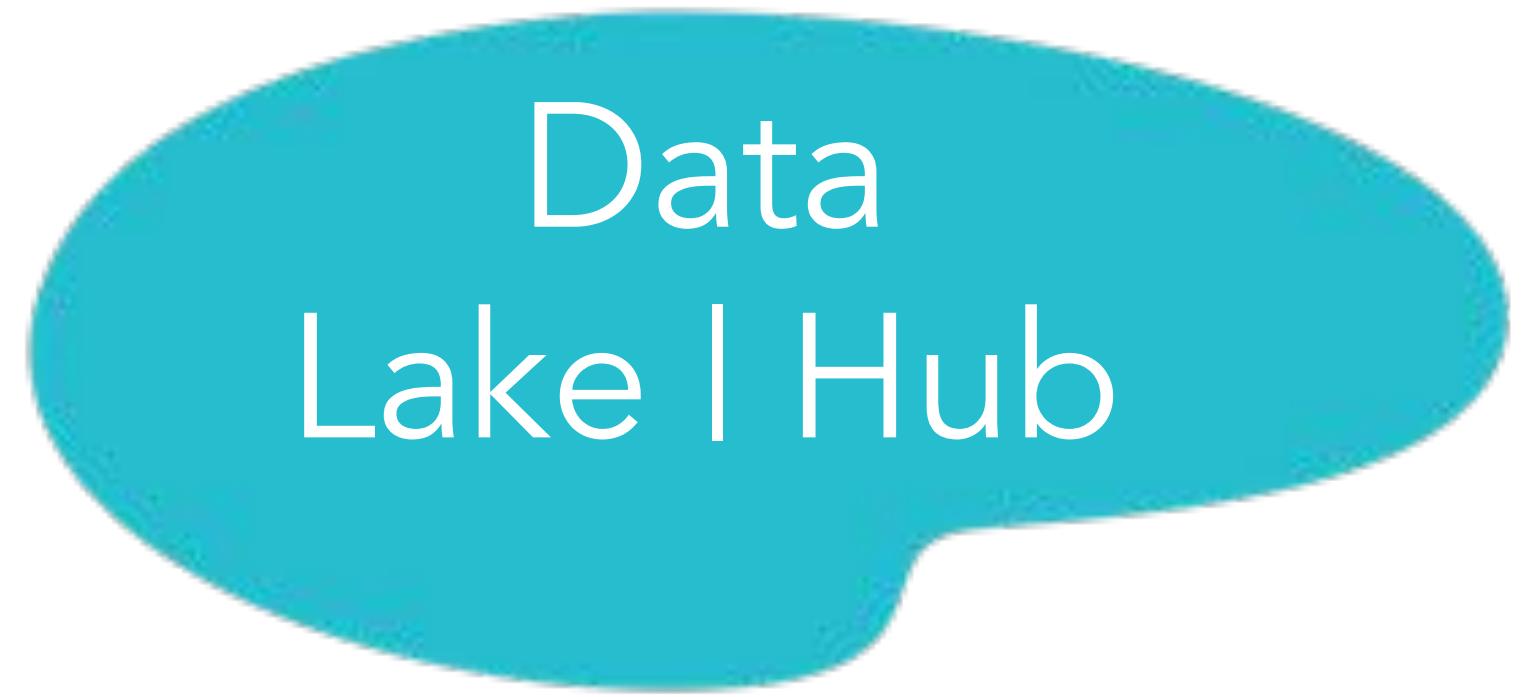
Data Transfer & Copying

Data Pipelines & Storage Infra

CURRENT GENERATIONS ...



Data
Warehouse



Data
Lake | Hub



Data
Lake on Cloud

Data Extracted from many sources

Transformed to single schema

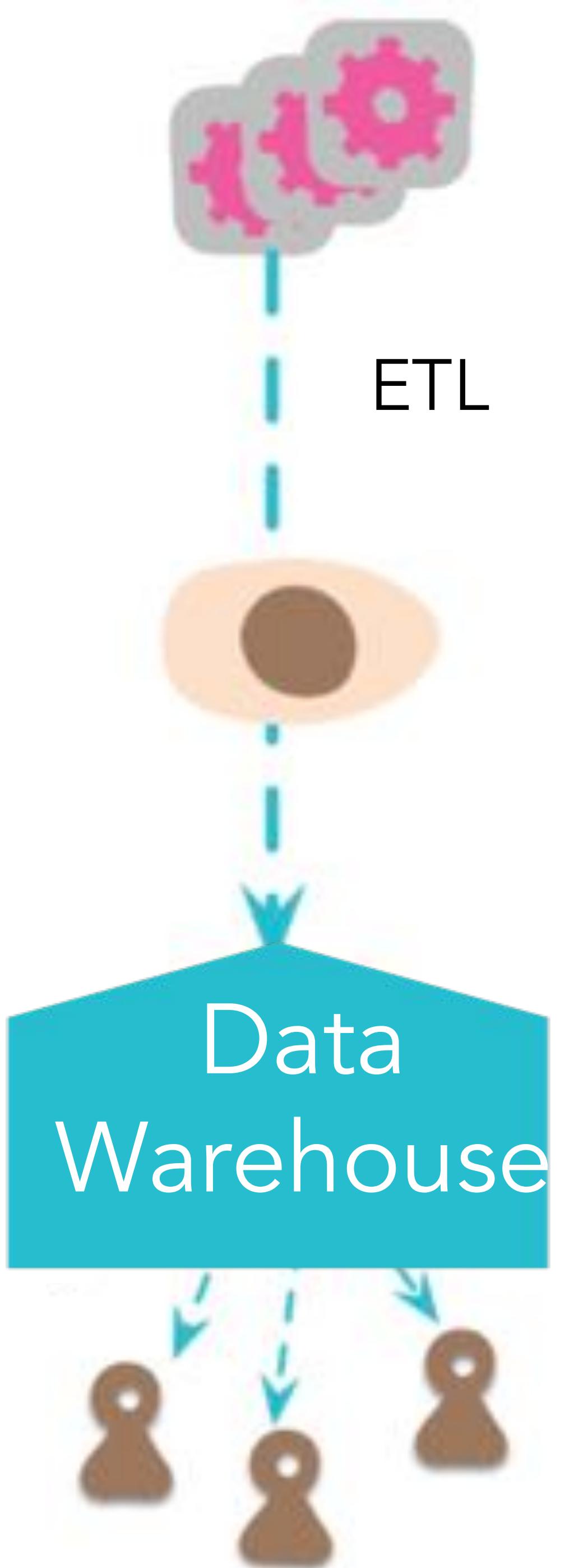
Loaded into warehouse

Analysis done on the warehouse

Used by data analysts

BI reports and dashboards

SQL-ish interface





TECHNOLOGIES ...



Google BigQuery



Data Extracted from many sources

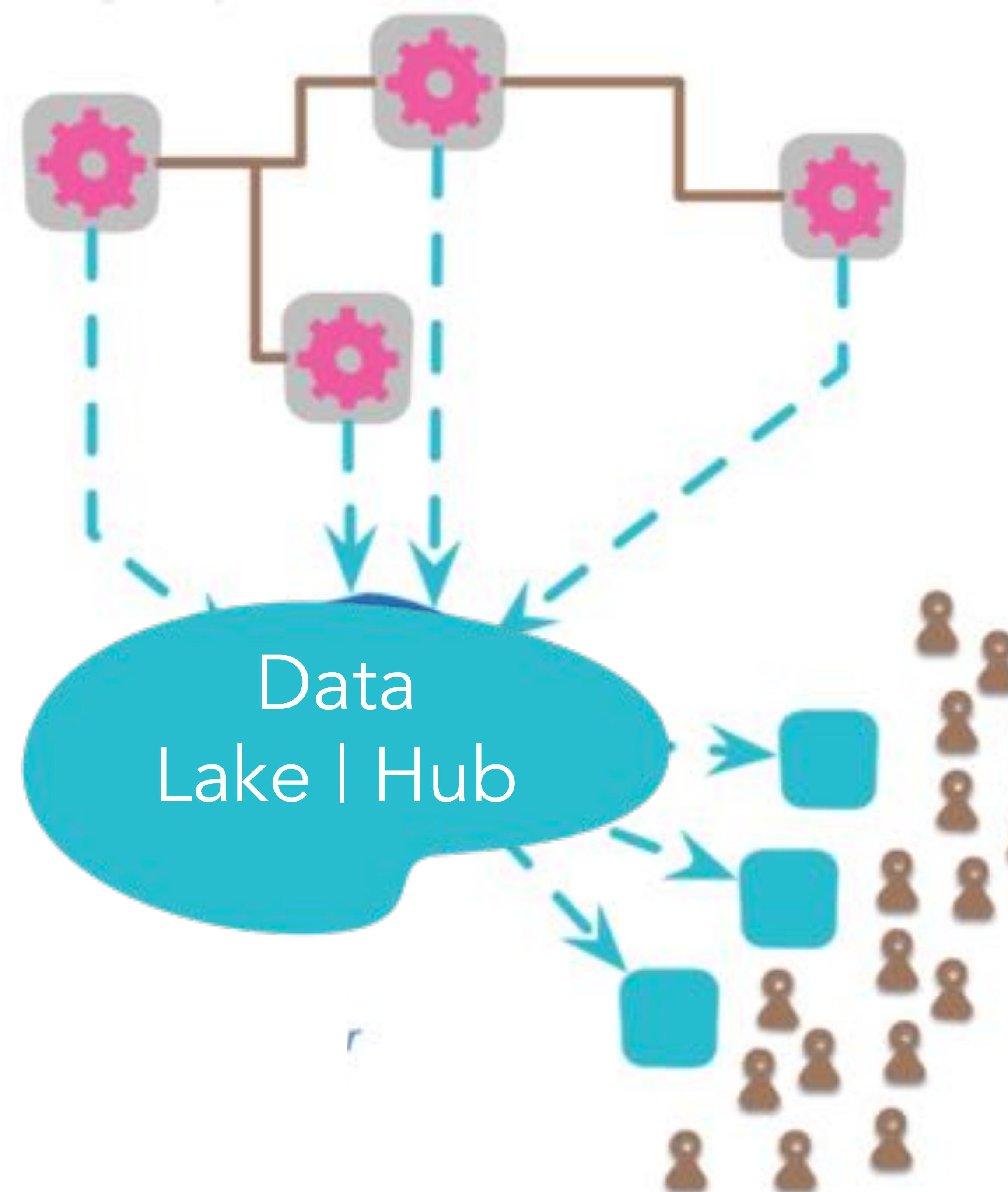
Loaded into the lake

Transformed to multiple lakeshore marts

Used by data scientists

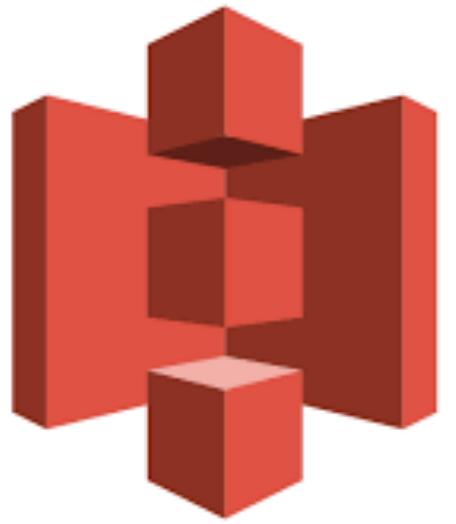
ML model training, data-driven apps

Raw file, API, or downstream DB access



Data
Lake | Hub

TECHNOLOGIES ...



Amazon S3



Azure Data Lake Storage



Google Cloud Storage

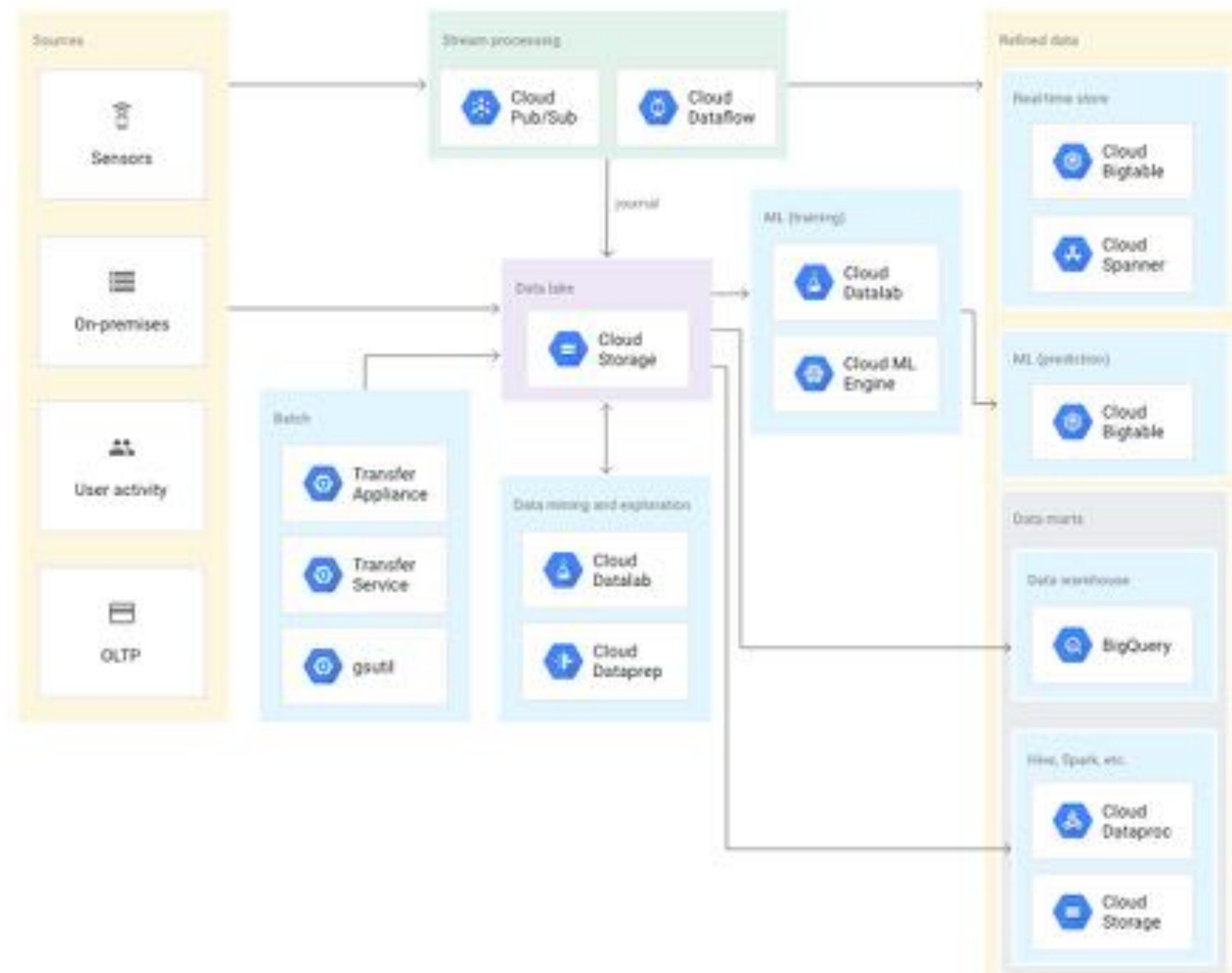


Azure Data Factory



Apache Airflow





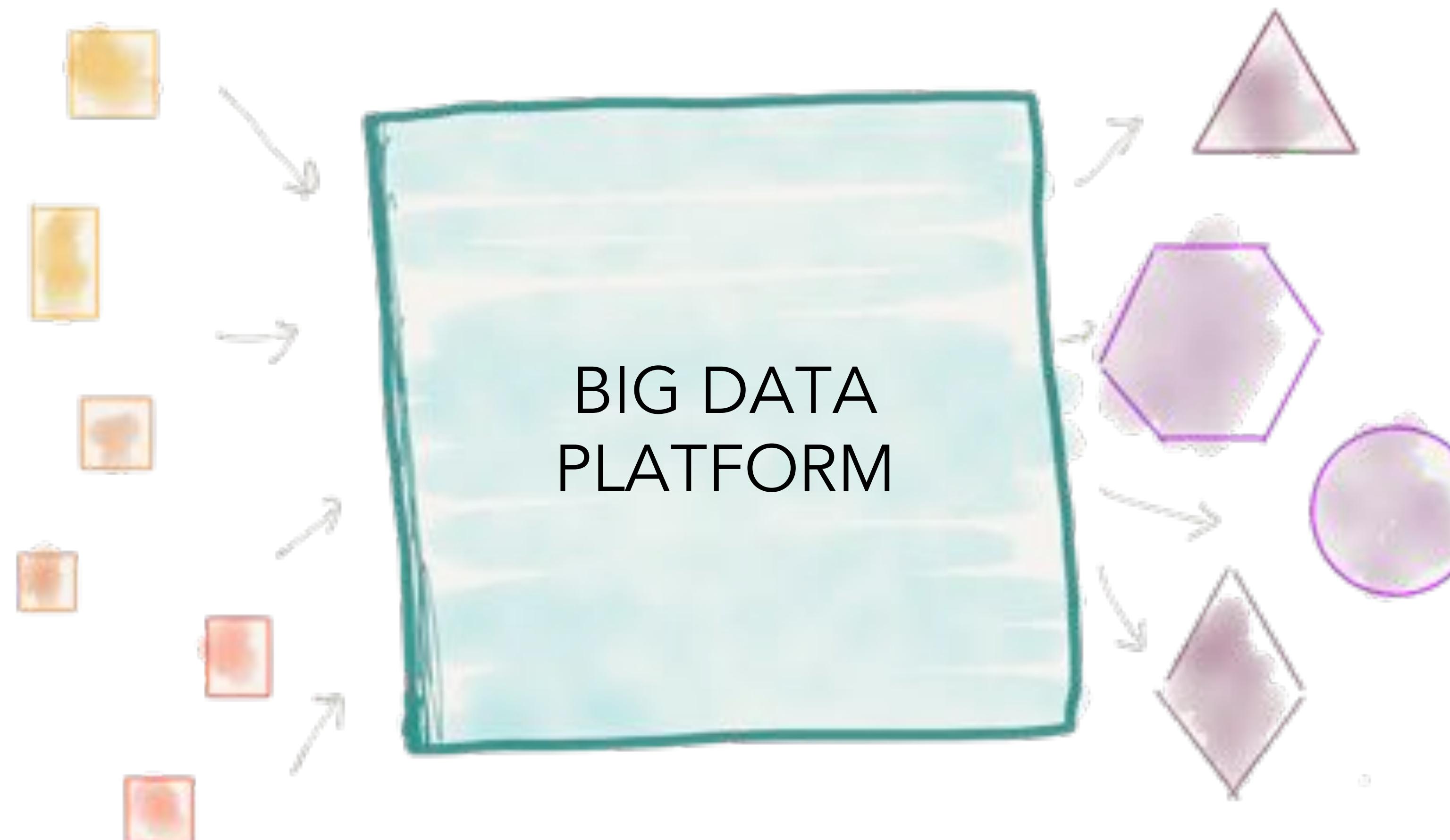
CENTRALIZED

SOURCES

TO INGEST

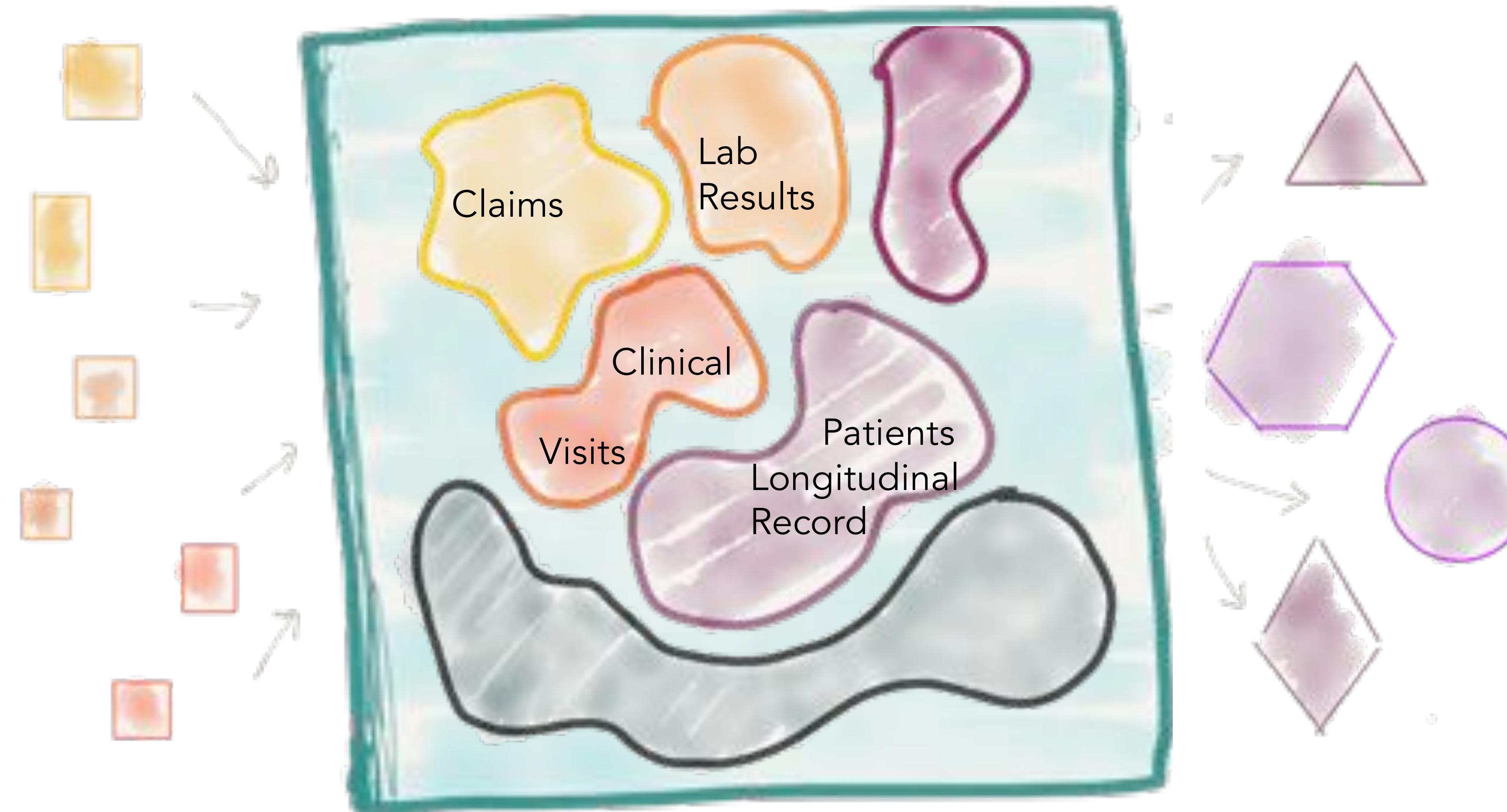
CONSUMERS

TO SERVE

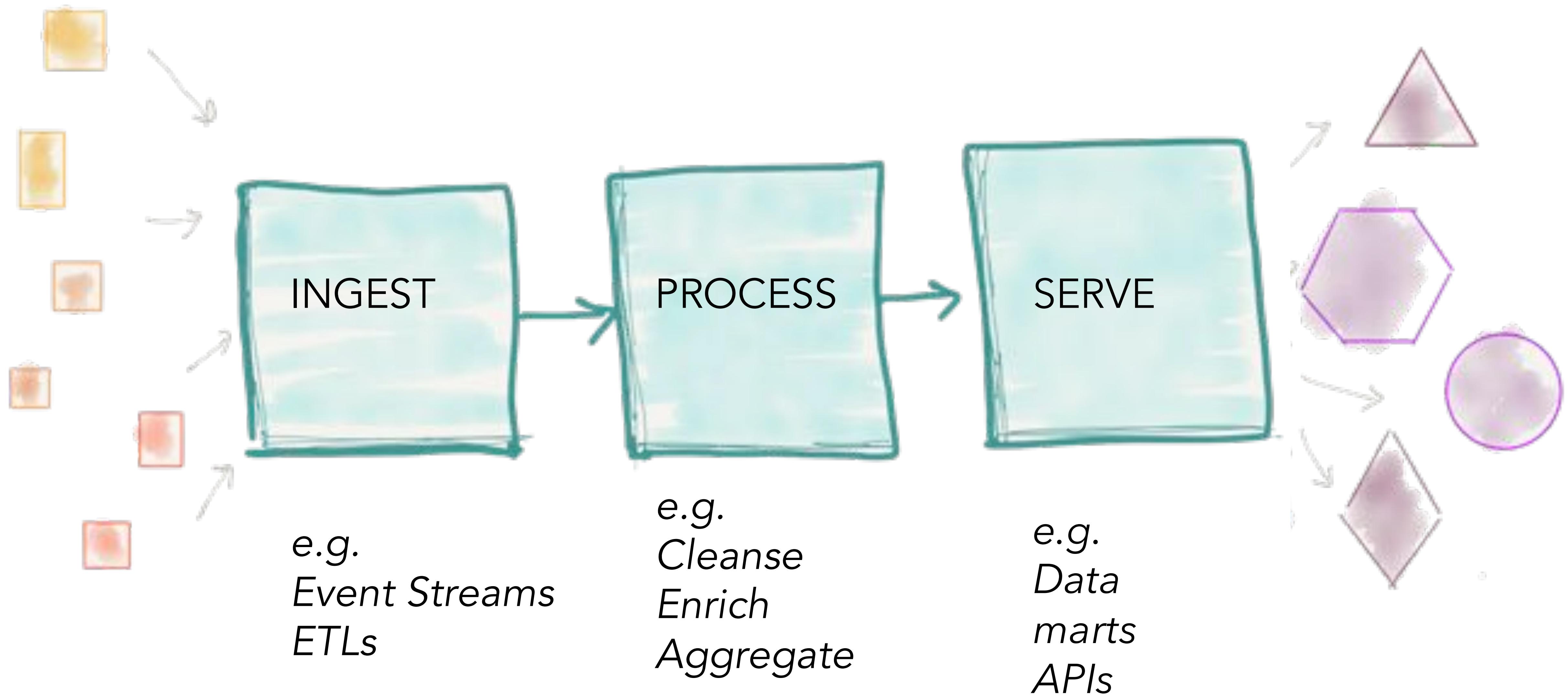


MONOLITHIC

No Domain Boundaries

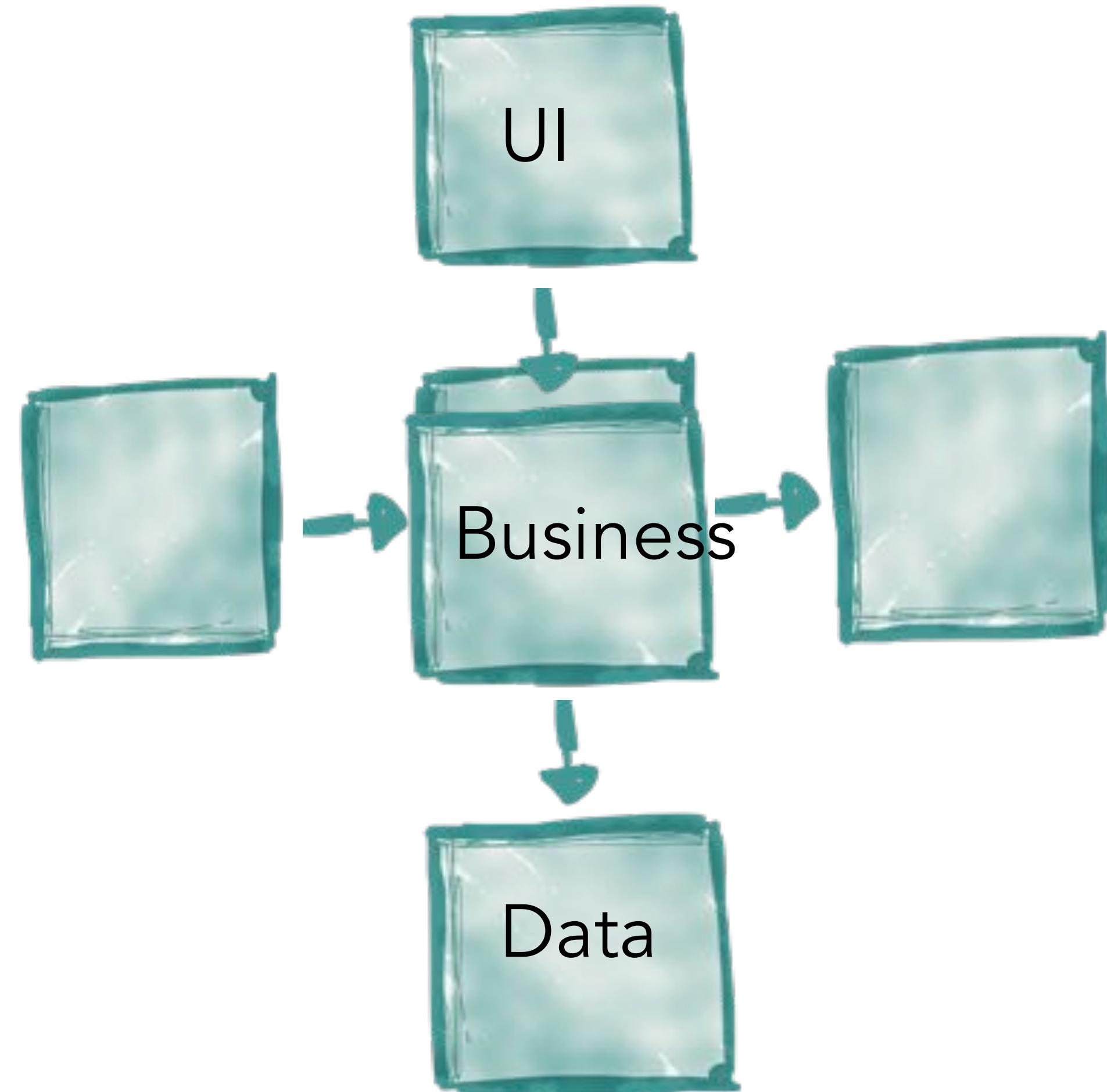


BIG DATA PLATFORM PIPELINE

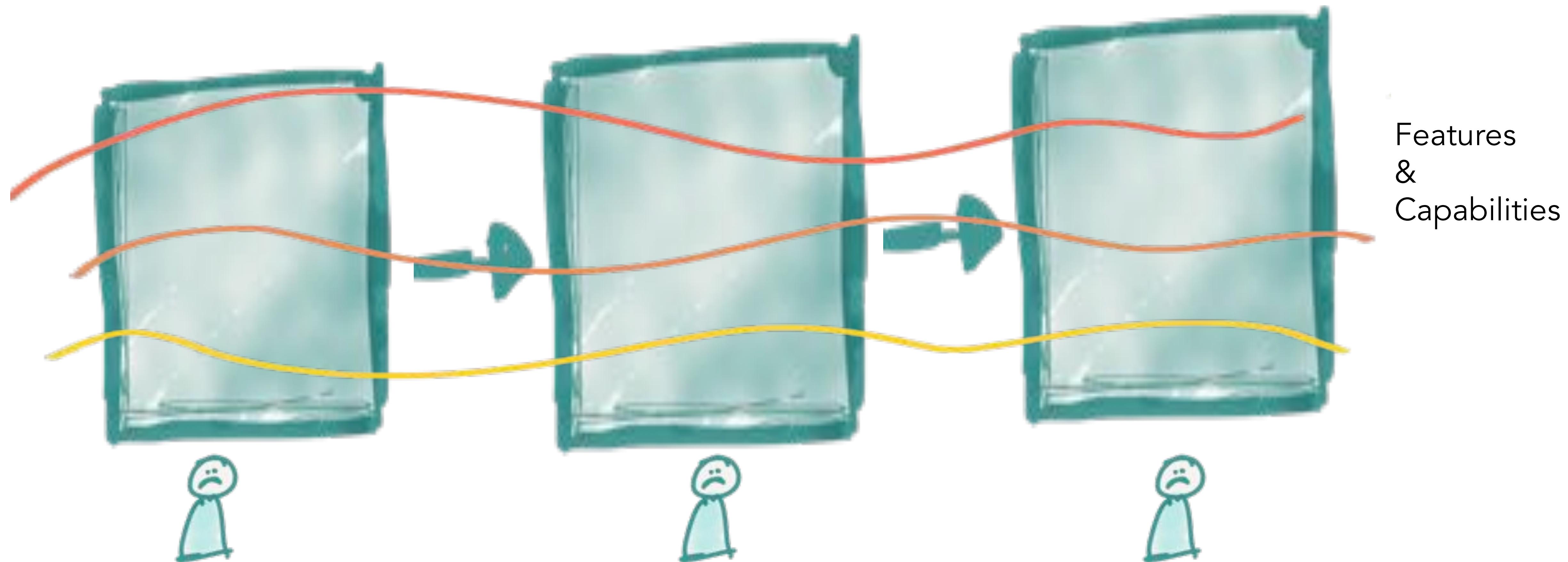


LAYERED

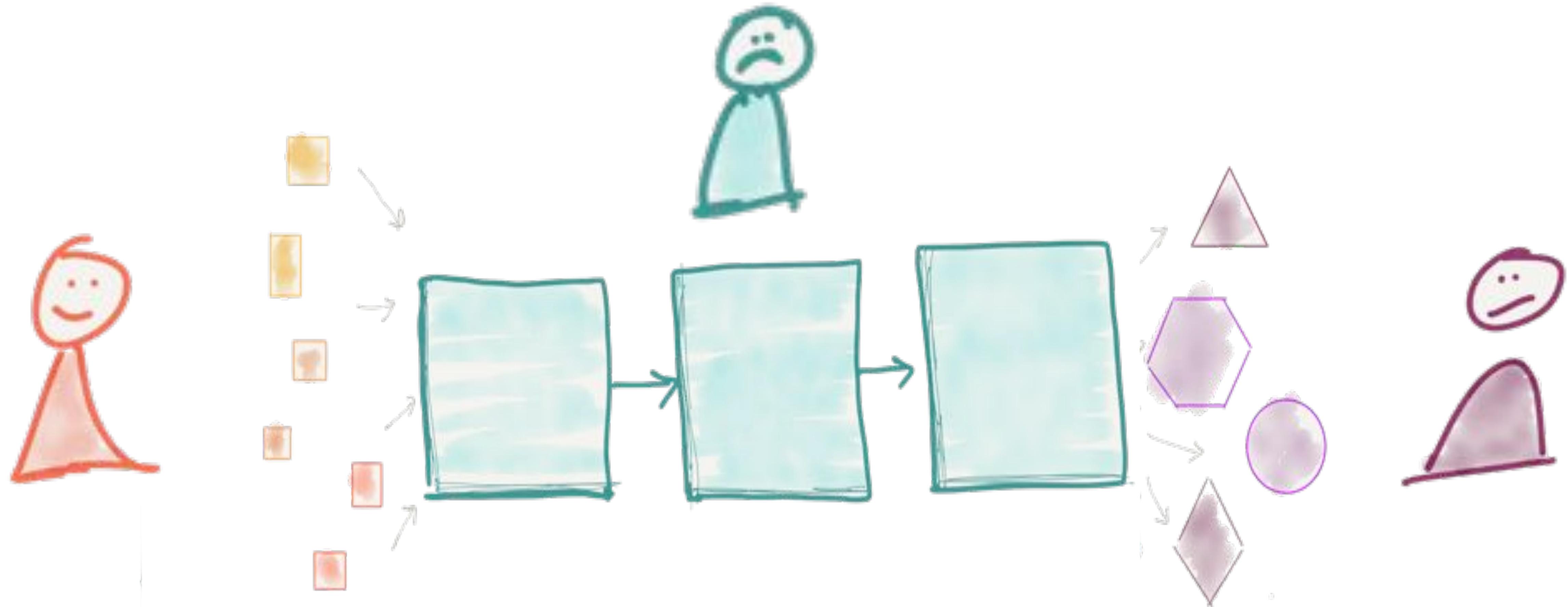
Top Level Technical Partitioning



Orthogonal to the Axis of Change!



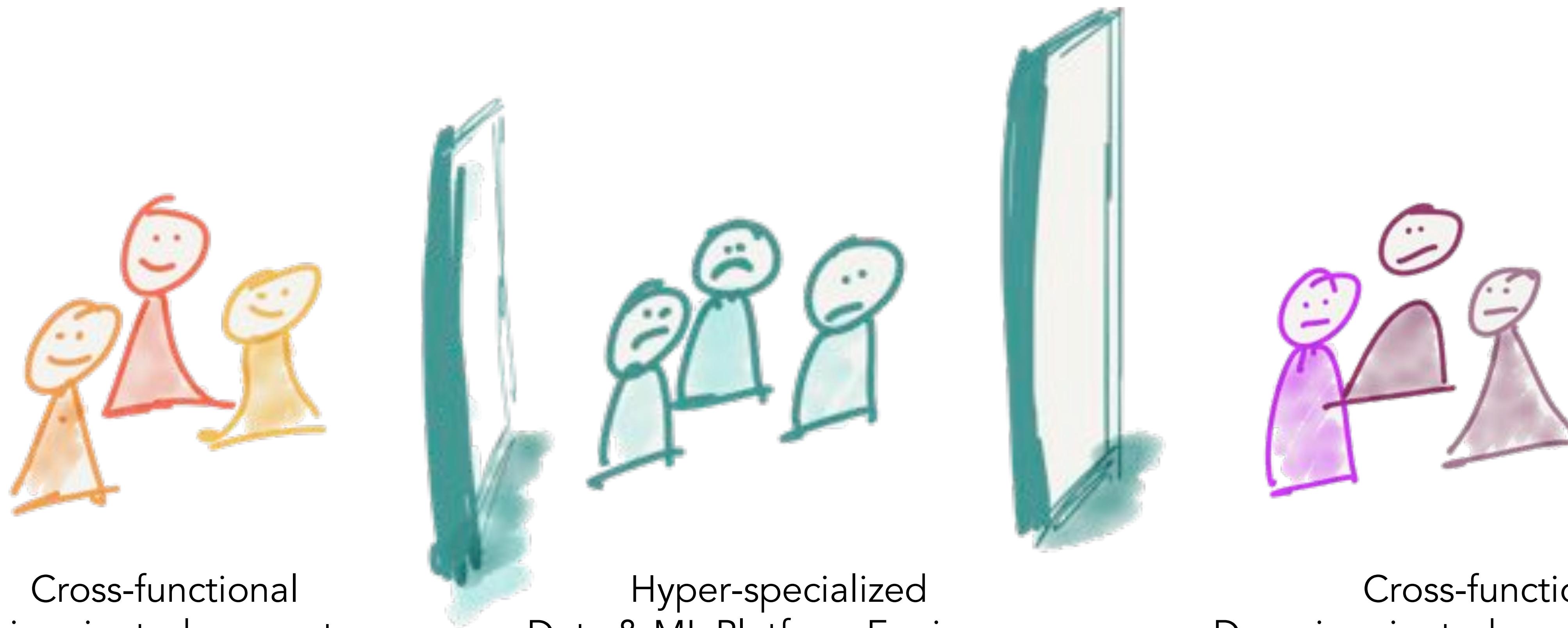
Data Platform Engineers



Domains' Operational
systems Teams

Data Scientists, BI teams, ...

HYPER-SPECIALIZED SILO



Cross-functional
Domain oriented source teams

Hyper-specialized
Data & ML Platform Engineers

Cross-functional
Domain oriented consumer teams

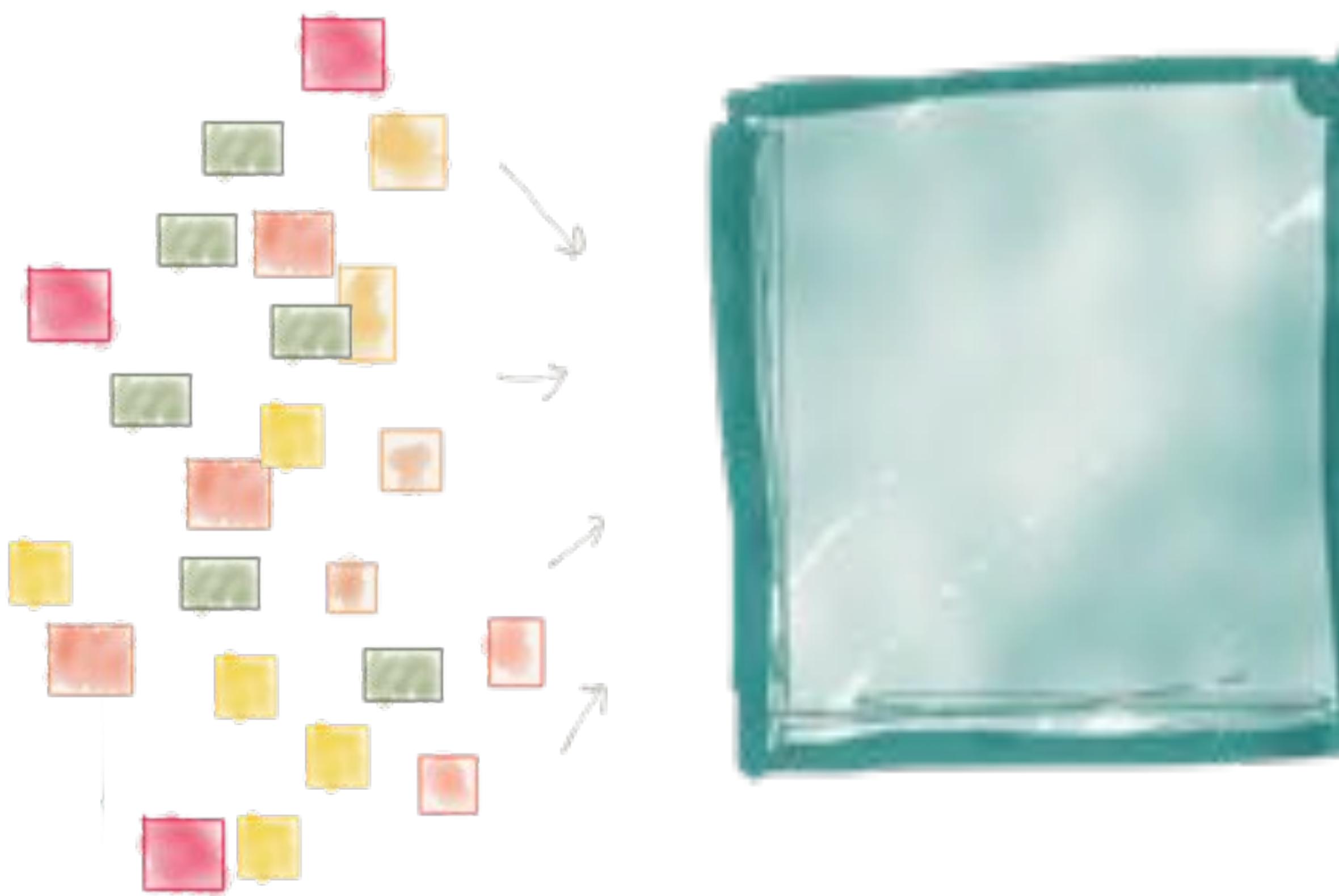


DEV



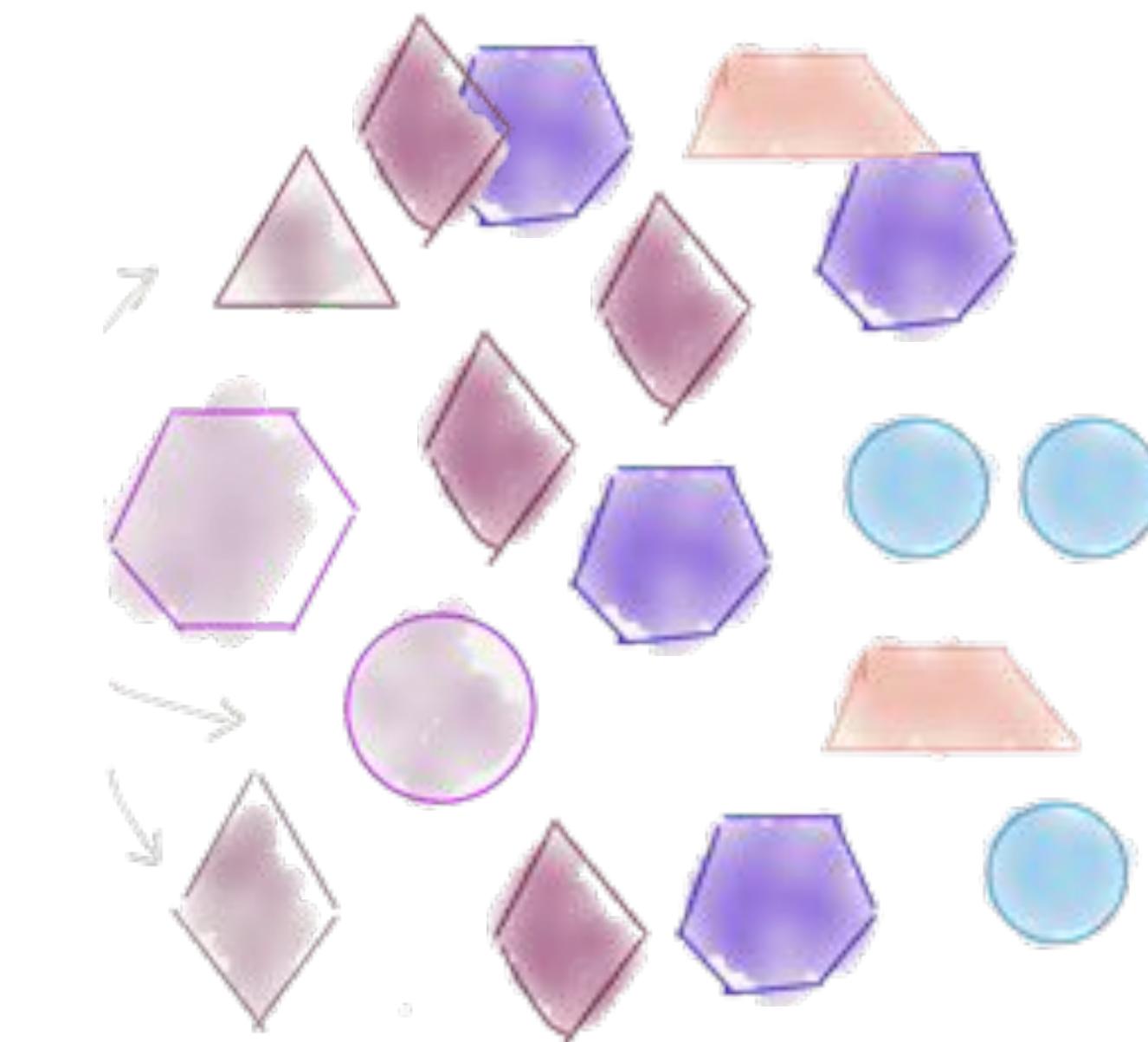
OPS

UBIQUITOUS DATA



SOURCE PROLIFERATION

INNOVATION AGENDA

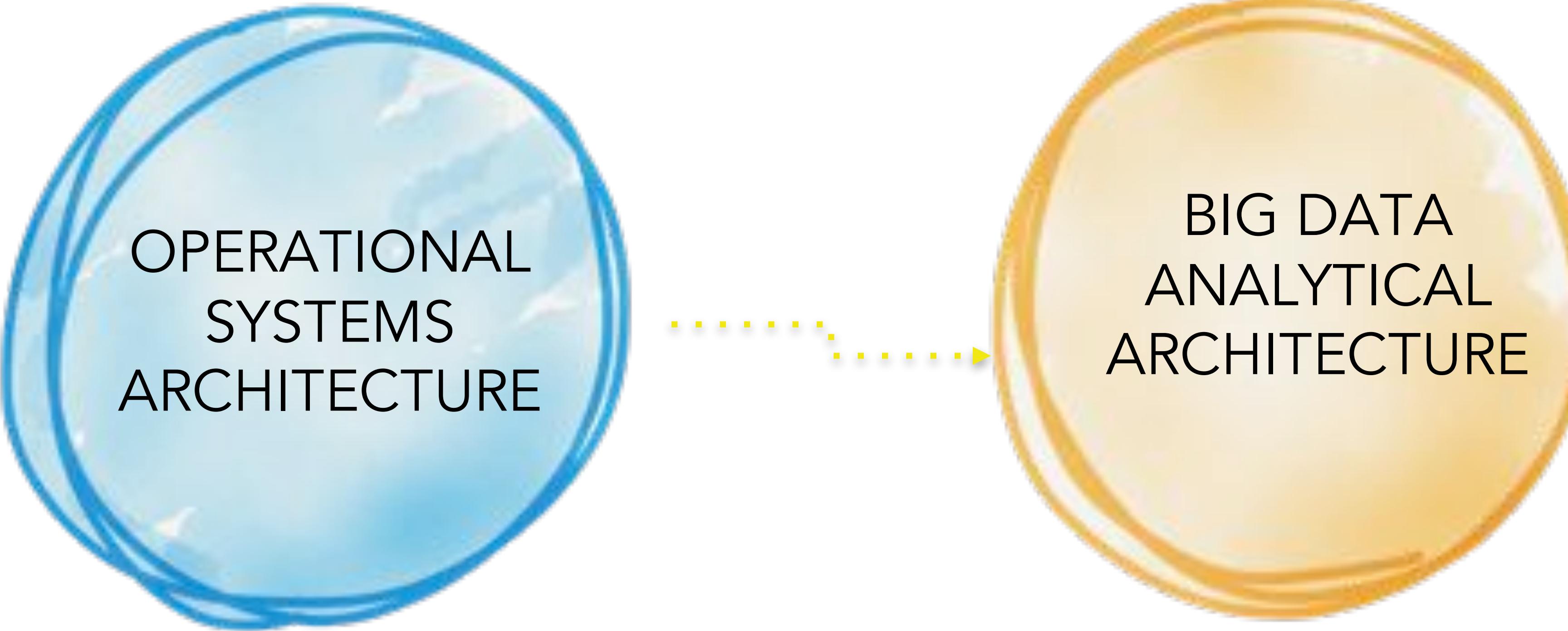


CONSUMER PROLIFERATION

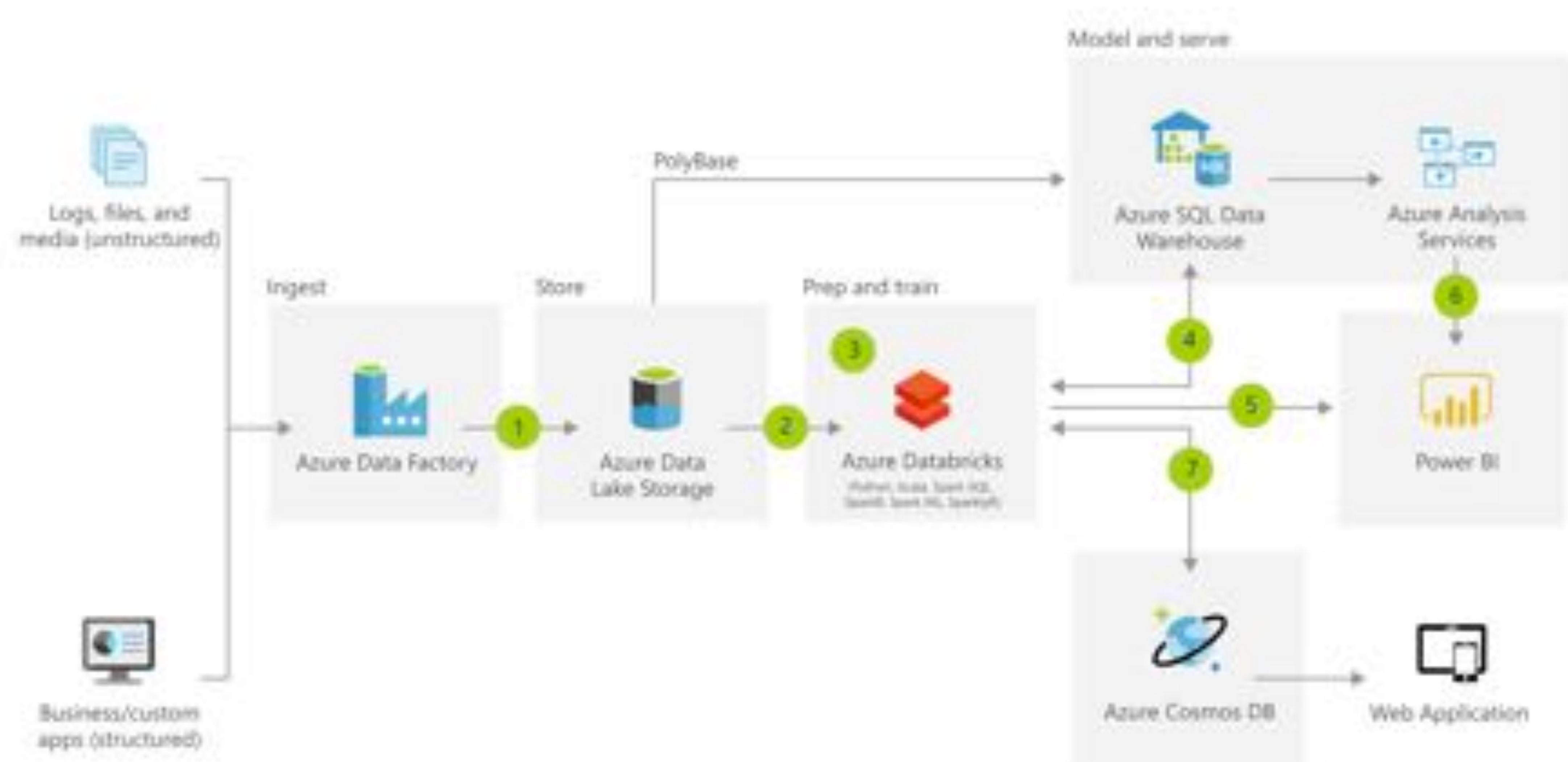
A hyper-specialized team
Building and Operating
Centralized Pipeline Solutions
Oriented around technical functionality

Does Not Scale!

THE BIG DIVIDE

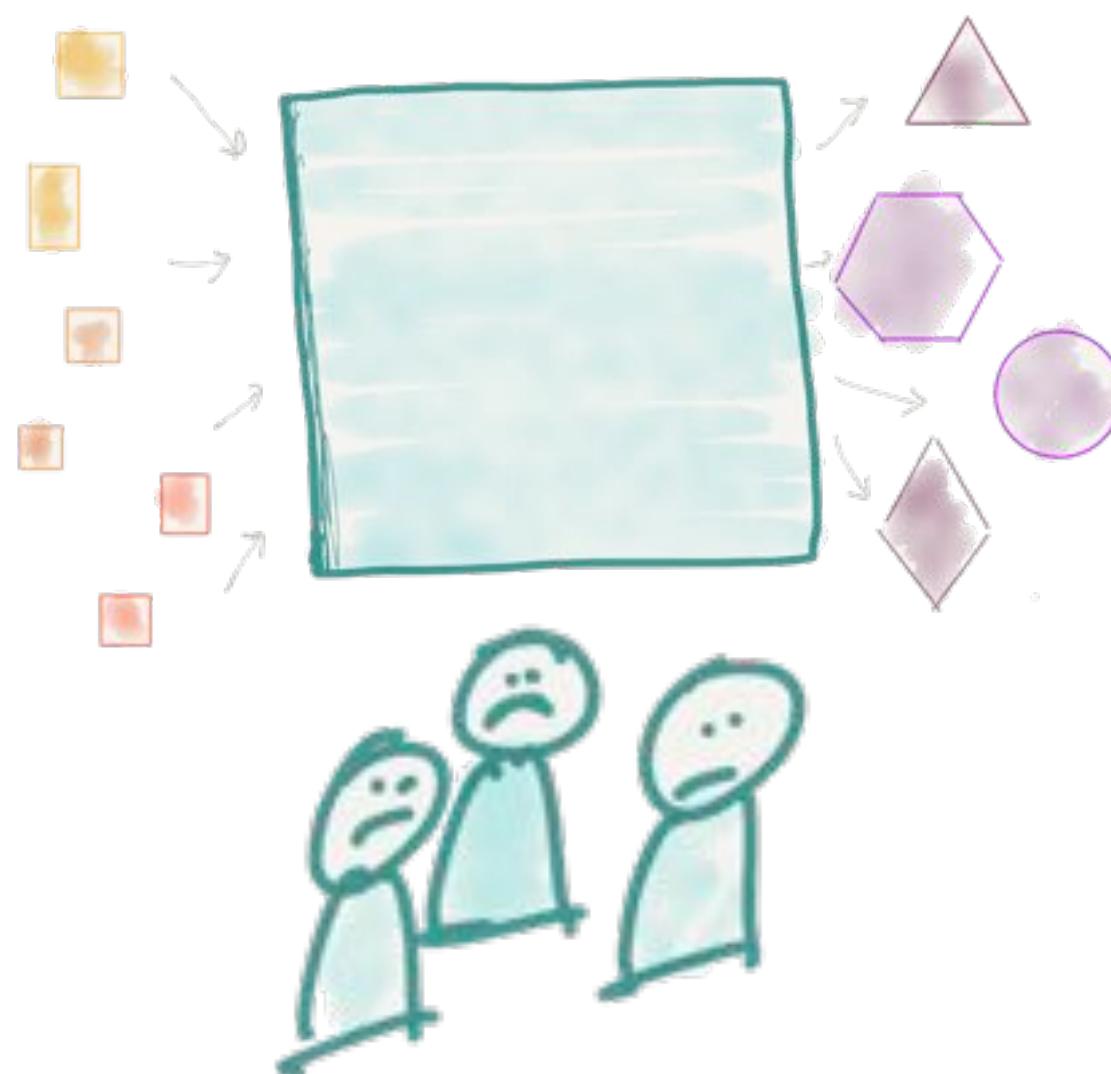


A typical data lake solution architecture ...



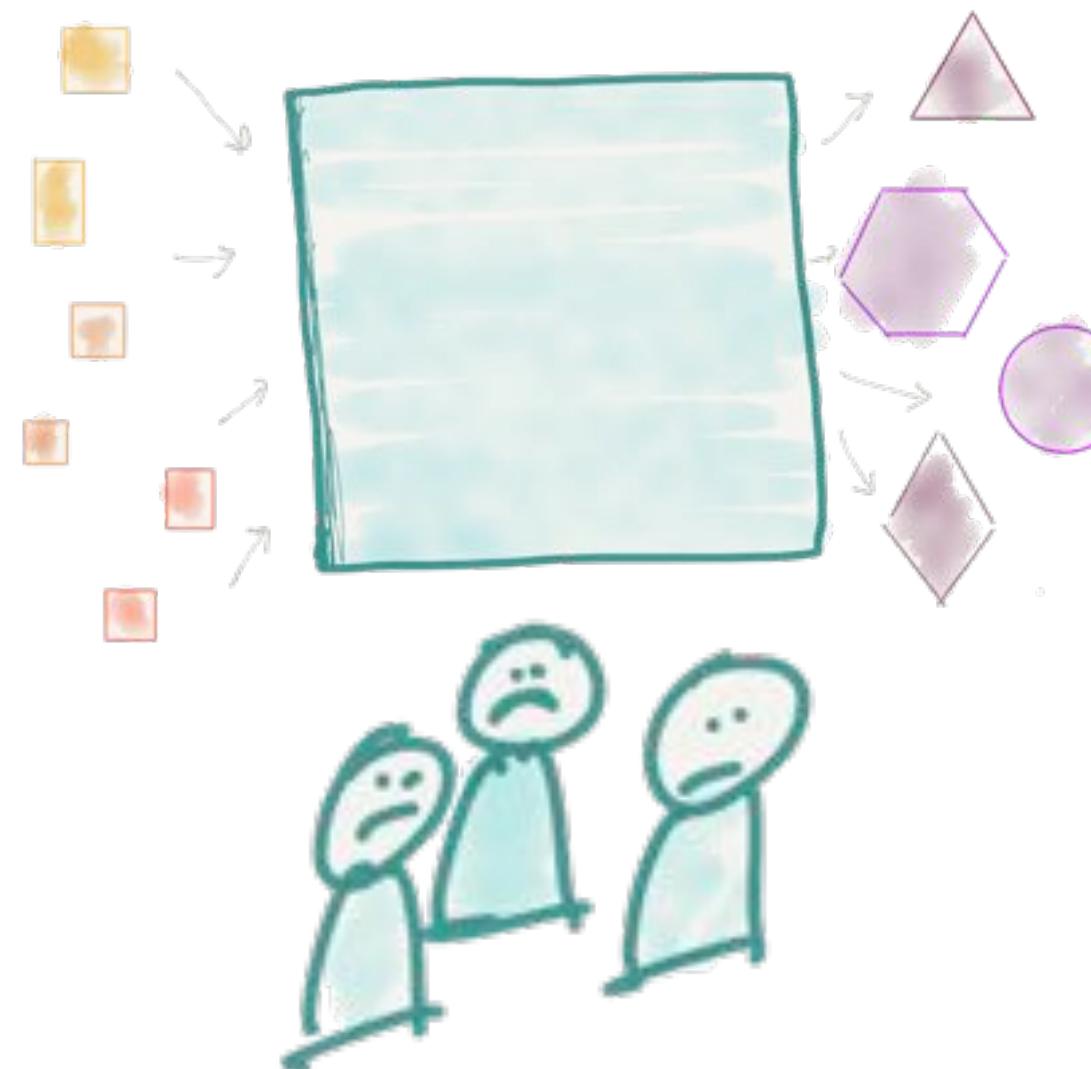
While wiring technology on paper looks convincing ...

CENTRALIZED SILOED

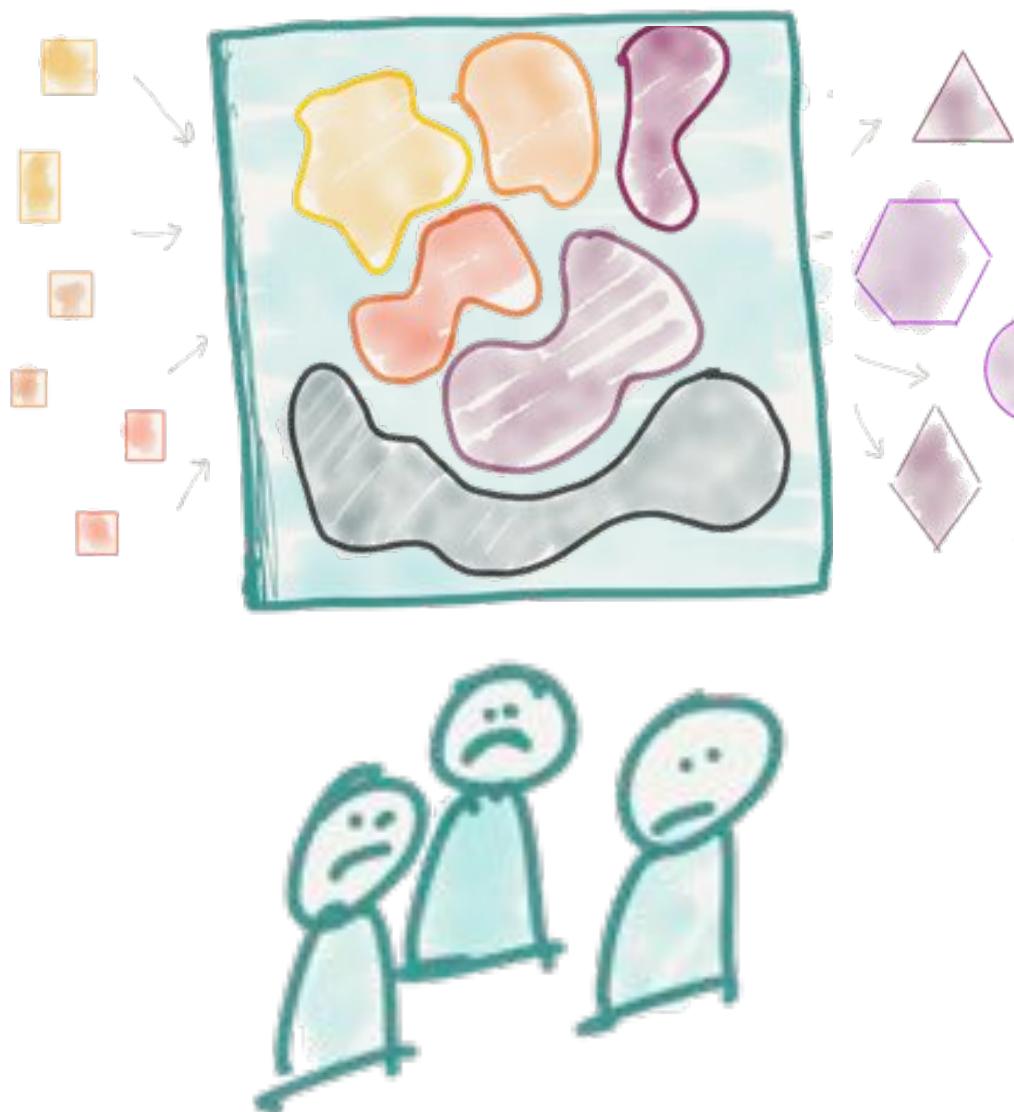


While wiring technology on paper looks convincing ...

CENTRALIZED
SILOED

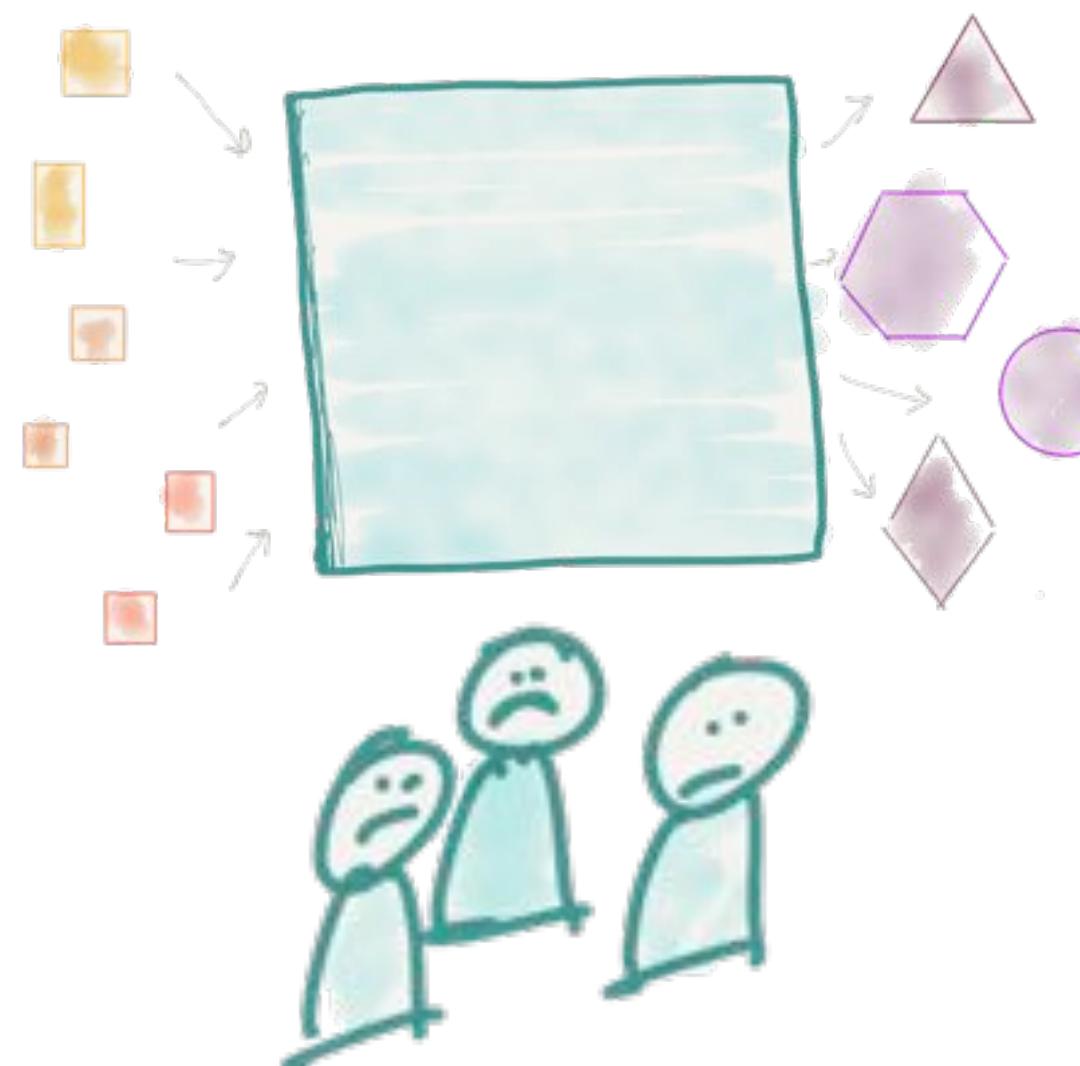


NO DOMAIN
BOUNDED
CONTEXT

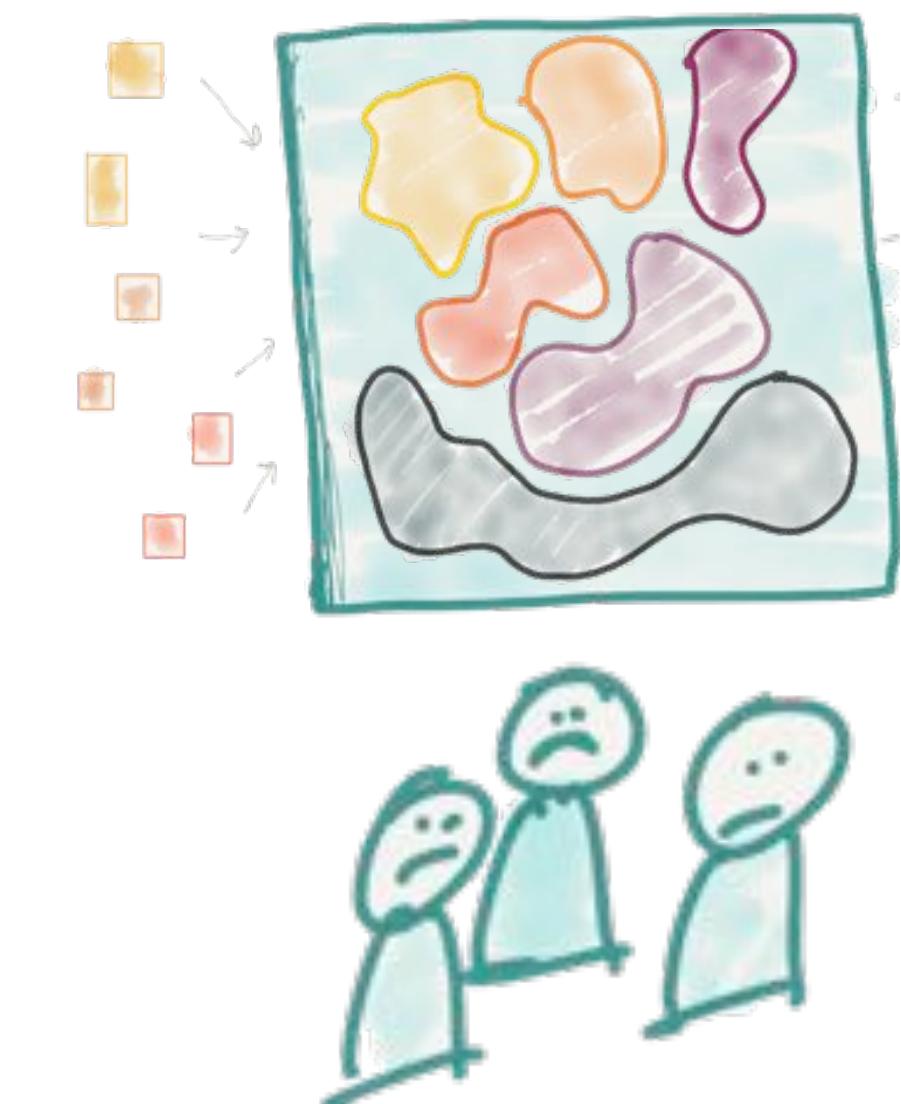


While wiring technology on paper looks convincing ...

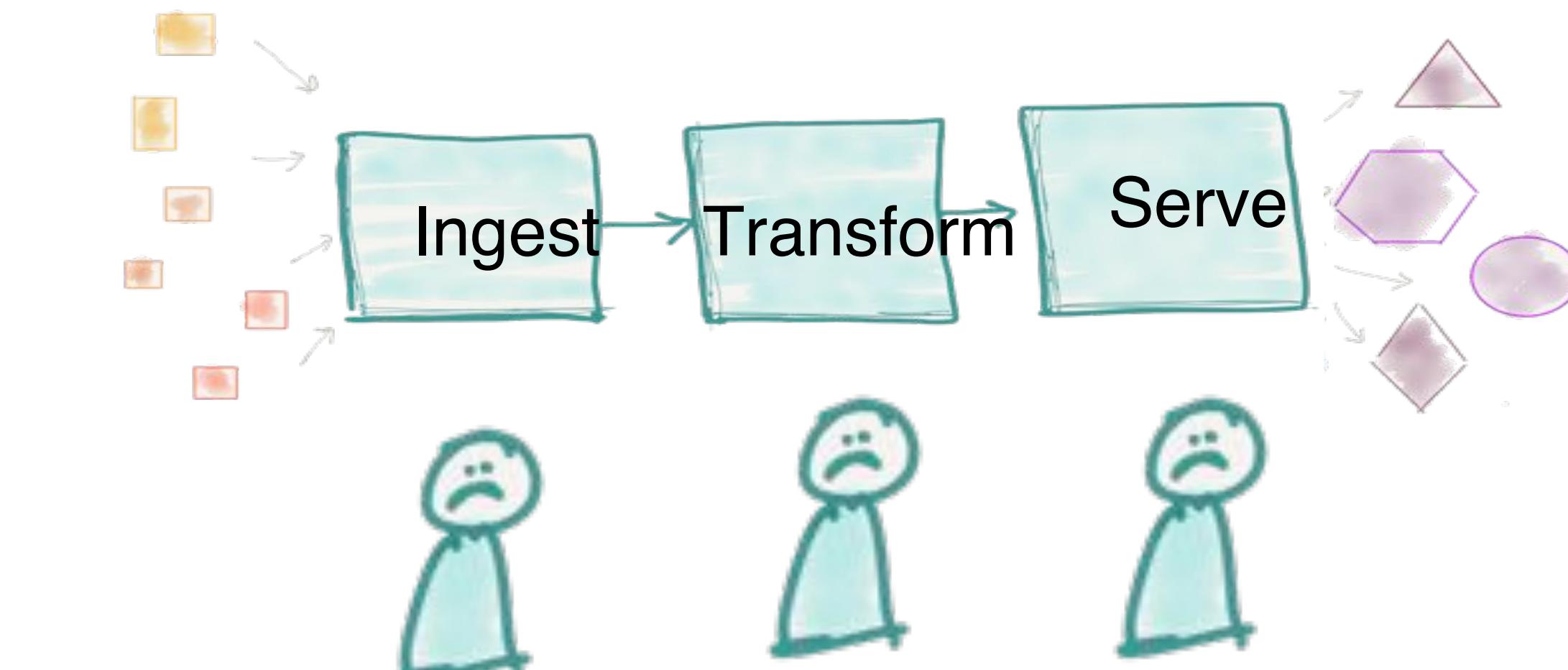
CENTRALIZED
SILOED



NO DOMAIN
BOUNDED
CONTEXT



TECHNOLOGY-DRIVEN
LAYERED
ARCHITECTURE



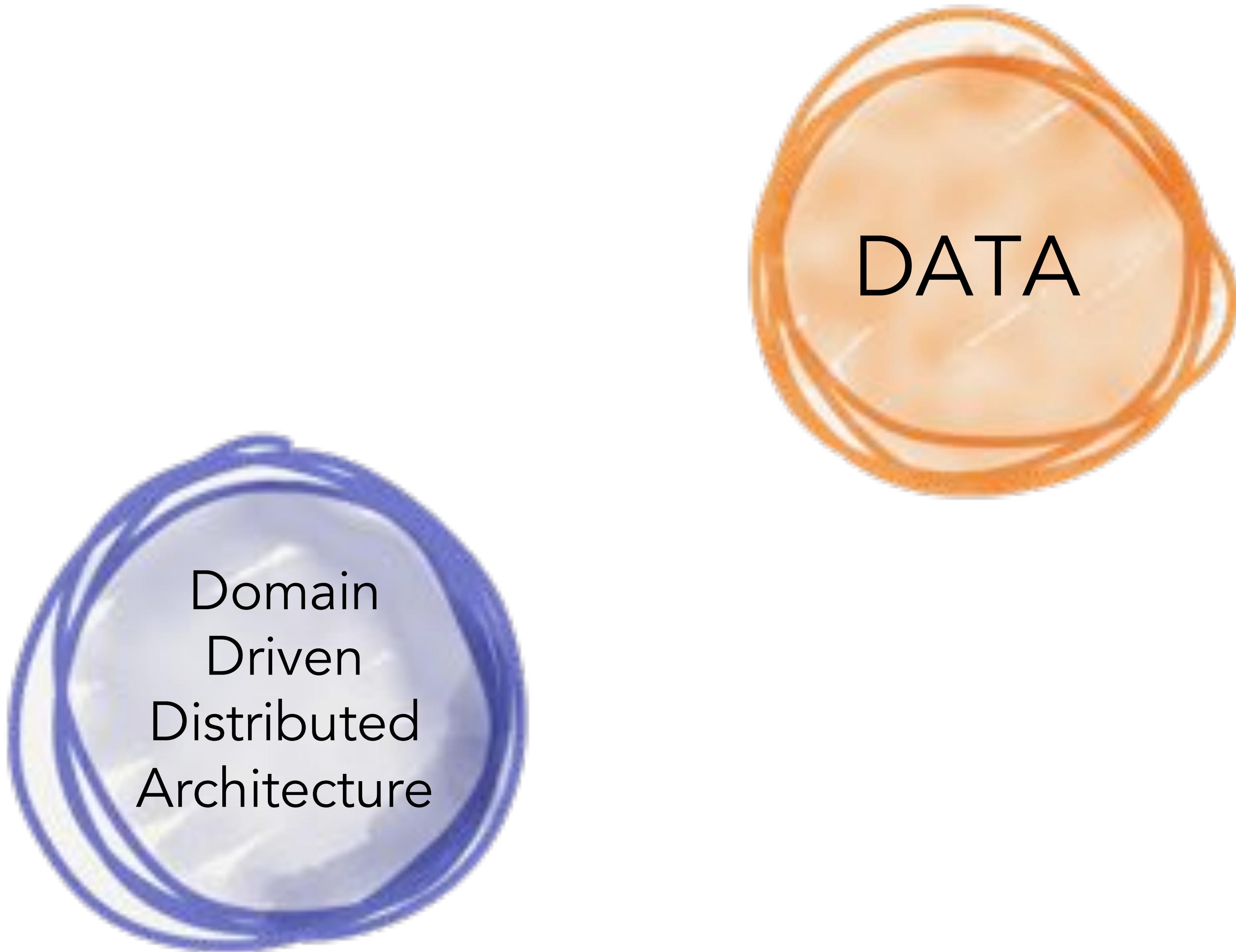
DATA MESH

ARCHITECTURE

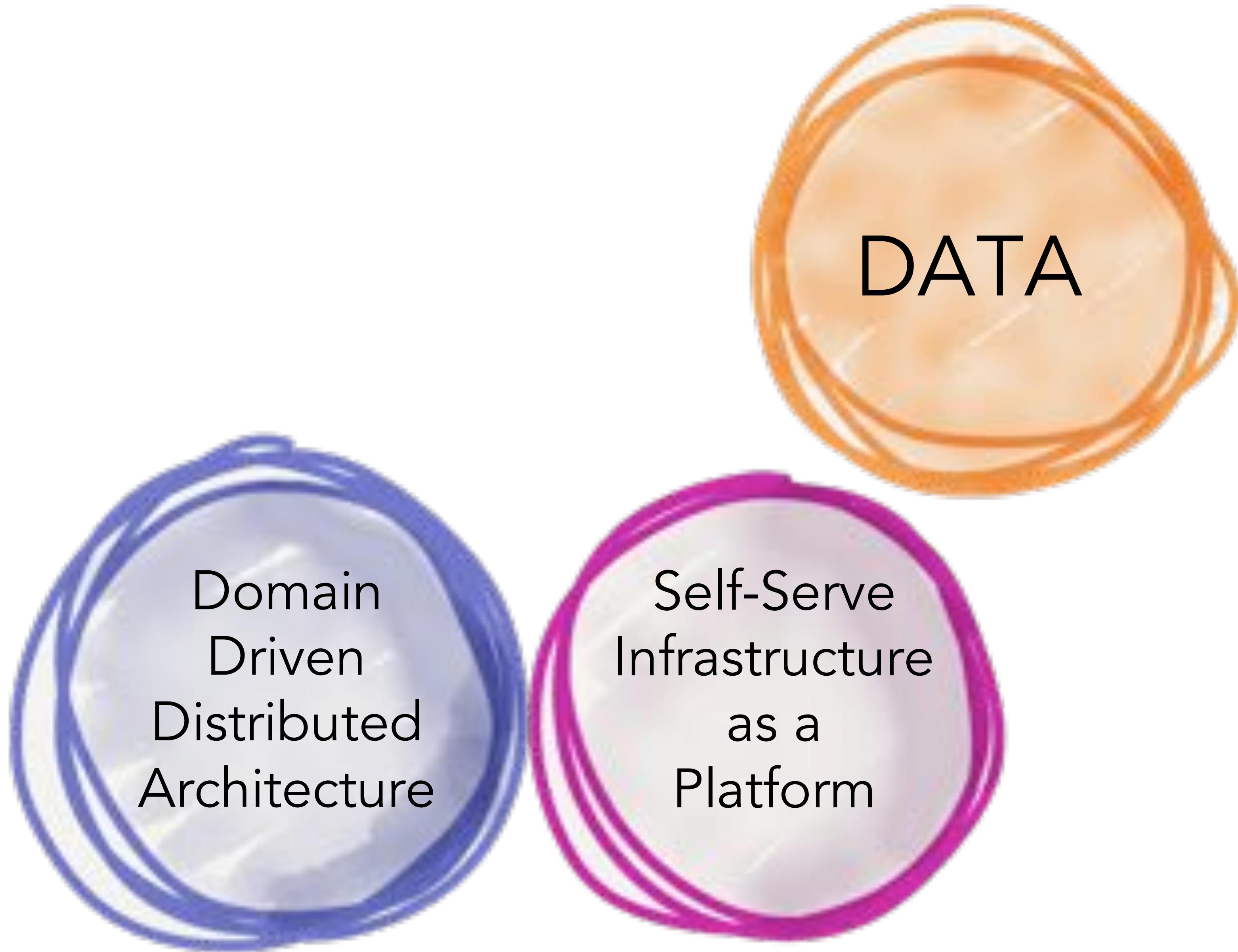
DATA MESH PRINCIPLES



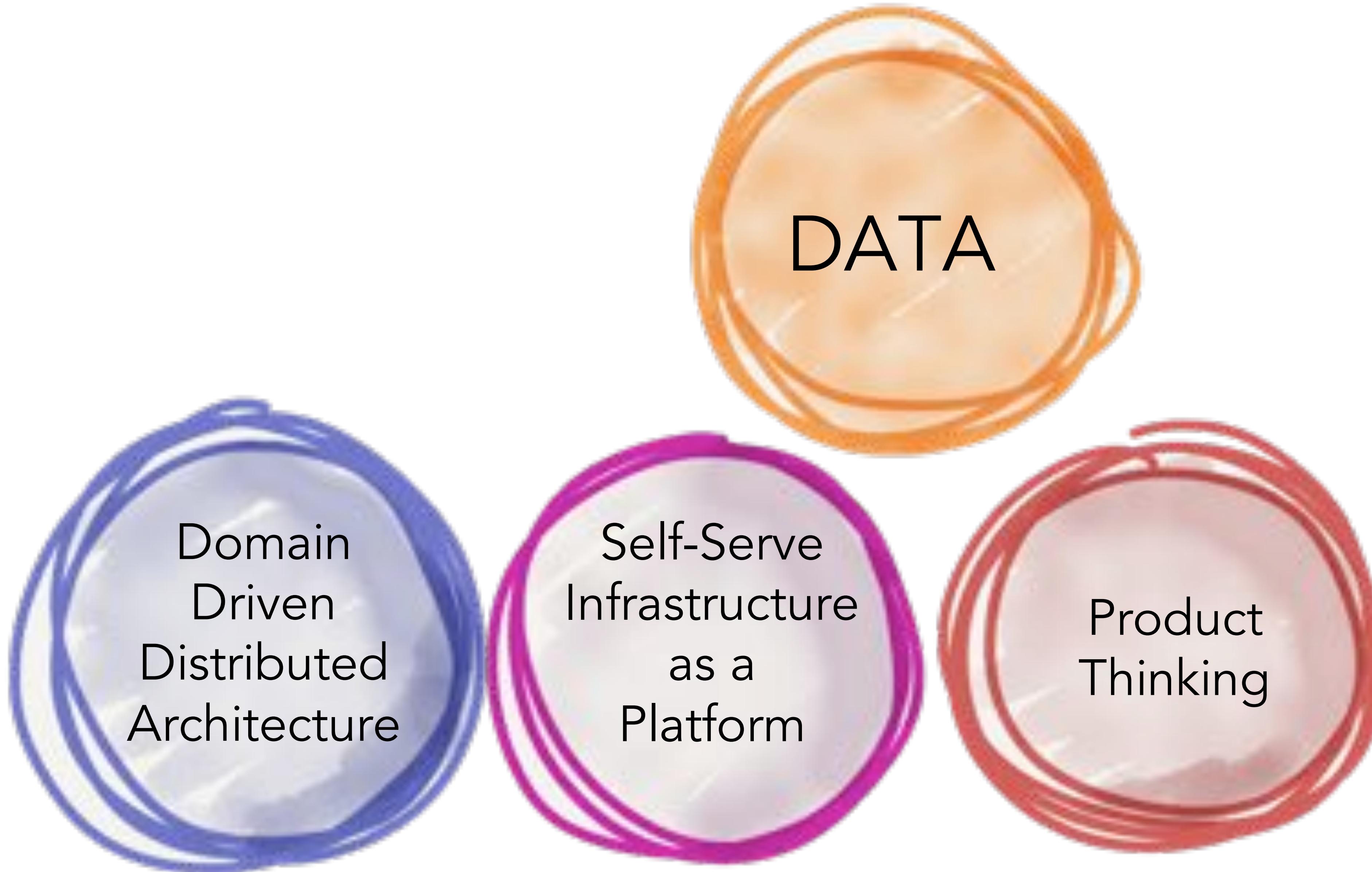
DATA MESH PRINCIPLES



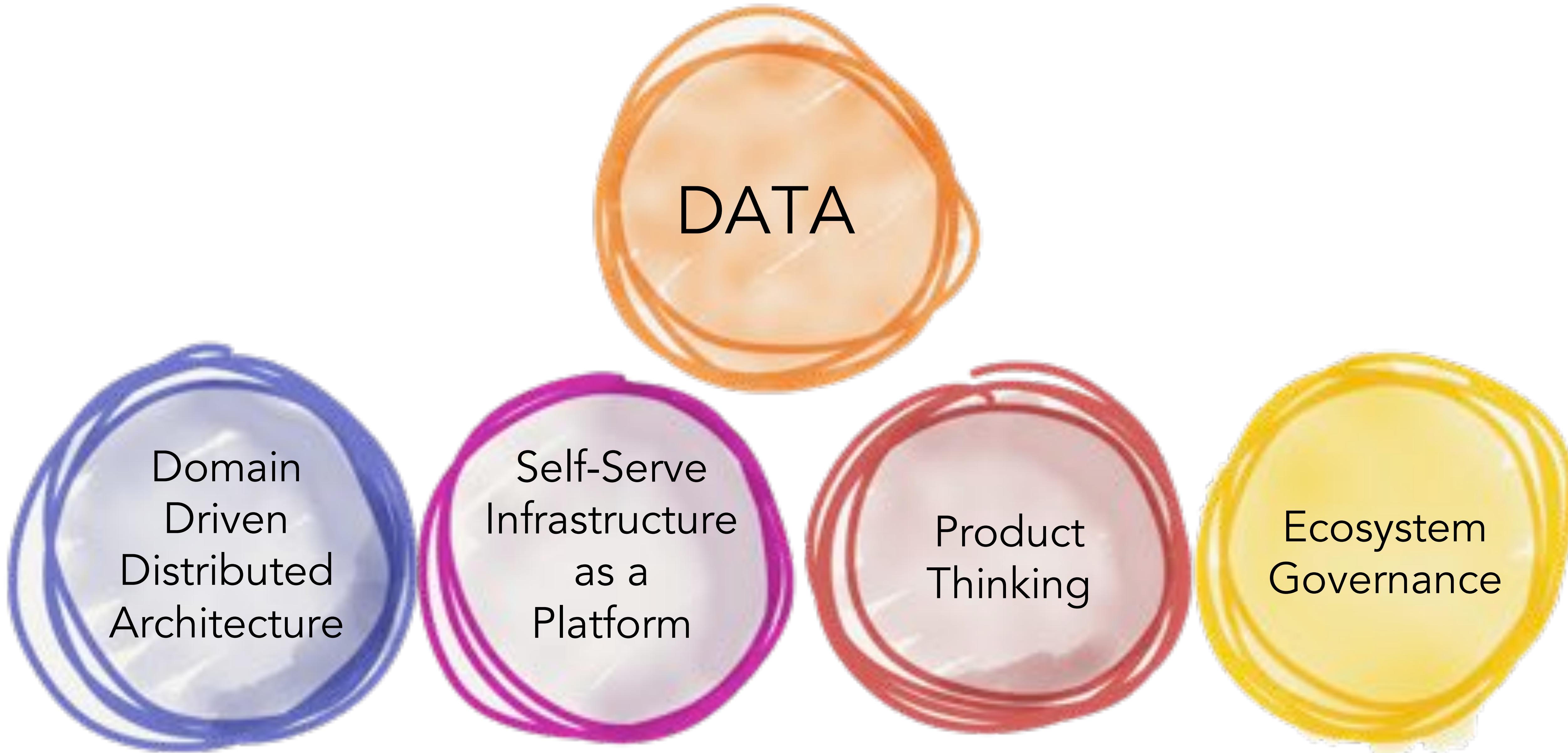
DATA MESH PRINCIPLES



DATA MESH PRINCIPLES



DATA MESH PRINCIPLES

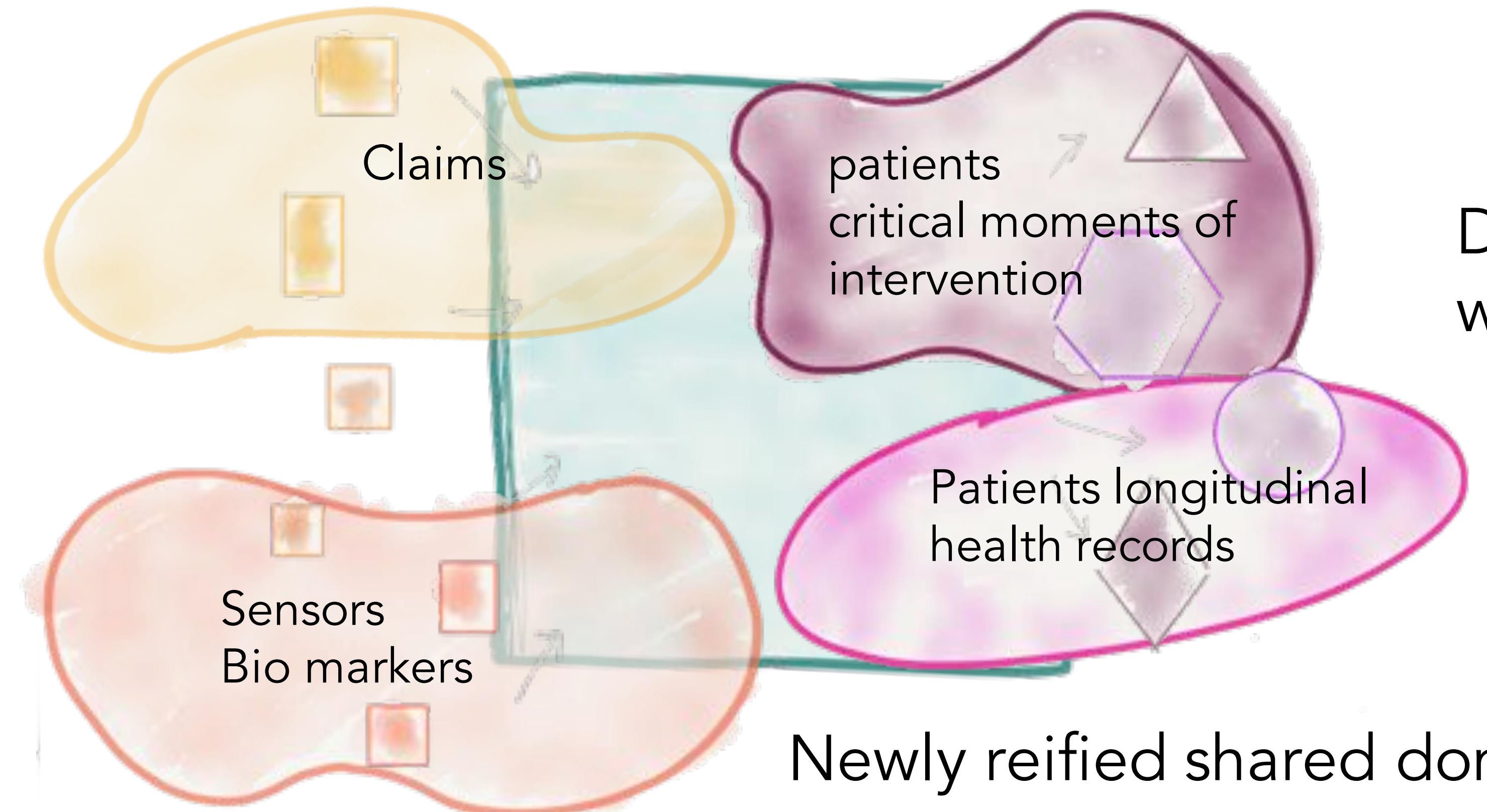




Domain Driven Distributed Architecture

DOMAIN ORIENTED DATA DECOMPOSITION & OWNERSHIP

Domains aligned
with the source



Domains aligned
with the consumption

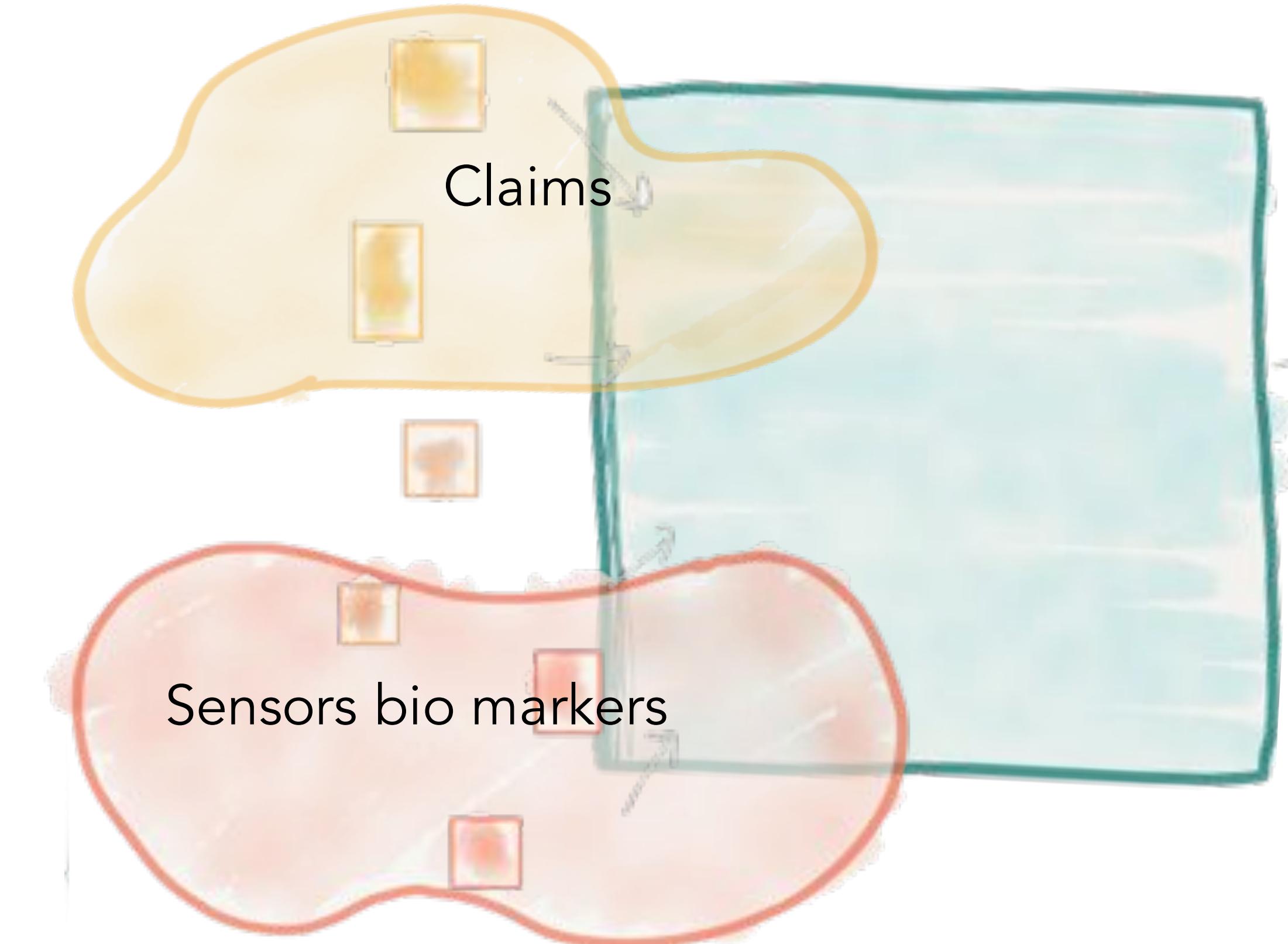
SOURCE ORIENTED (NATIVE) DOMAIN DATA

Facts & reality of business

Immutable timed events /
Historical snapshots

Change less frequently

Permanently captured



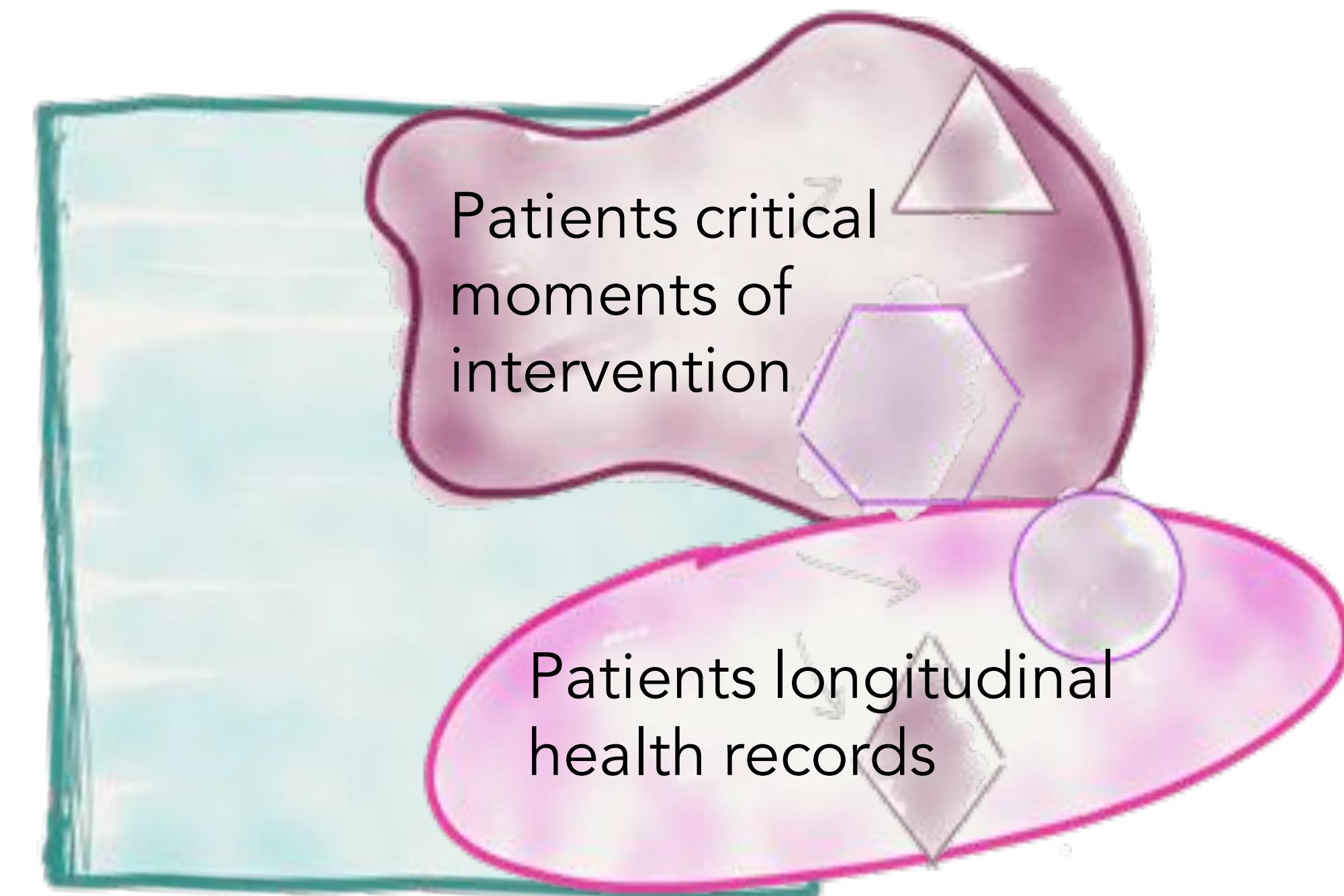
CONSUMER ORIENTED DOMAIN DATA

Fit for consumer purpose

Aggregation / Projections / Transformed

Change more often

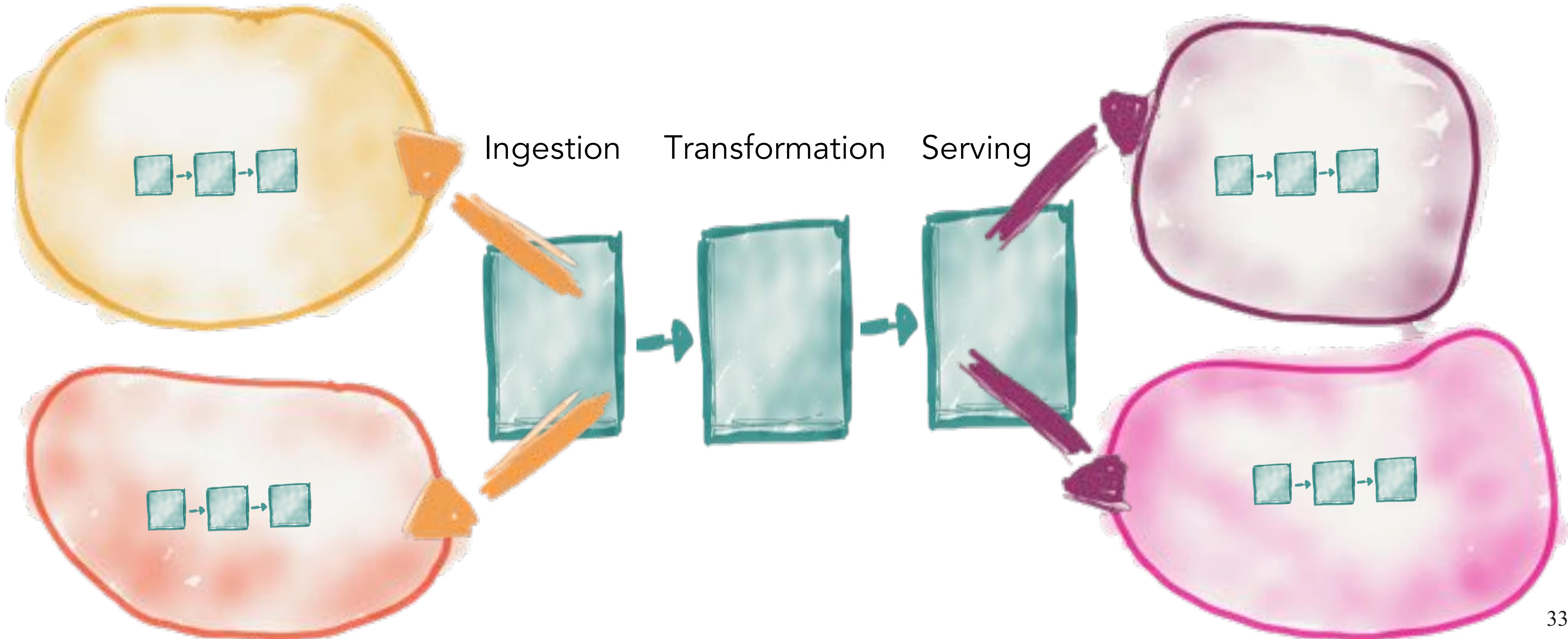
Can be recreated



DISTRIBUTED PIPELINES IN DOMAINS

More cleansing, integrity checks here

More aggregations, ML modeling here



Decomposition Heuristics

Aligned with source domains to retain business facts

Aligned with common consumptions to share modelled 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

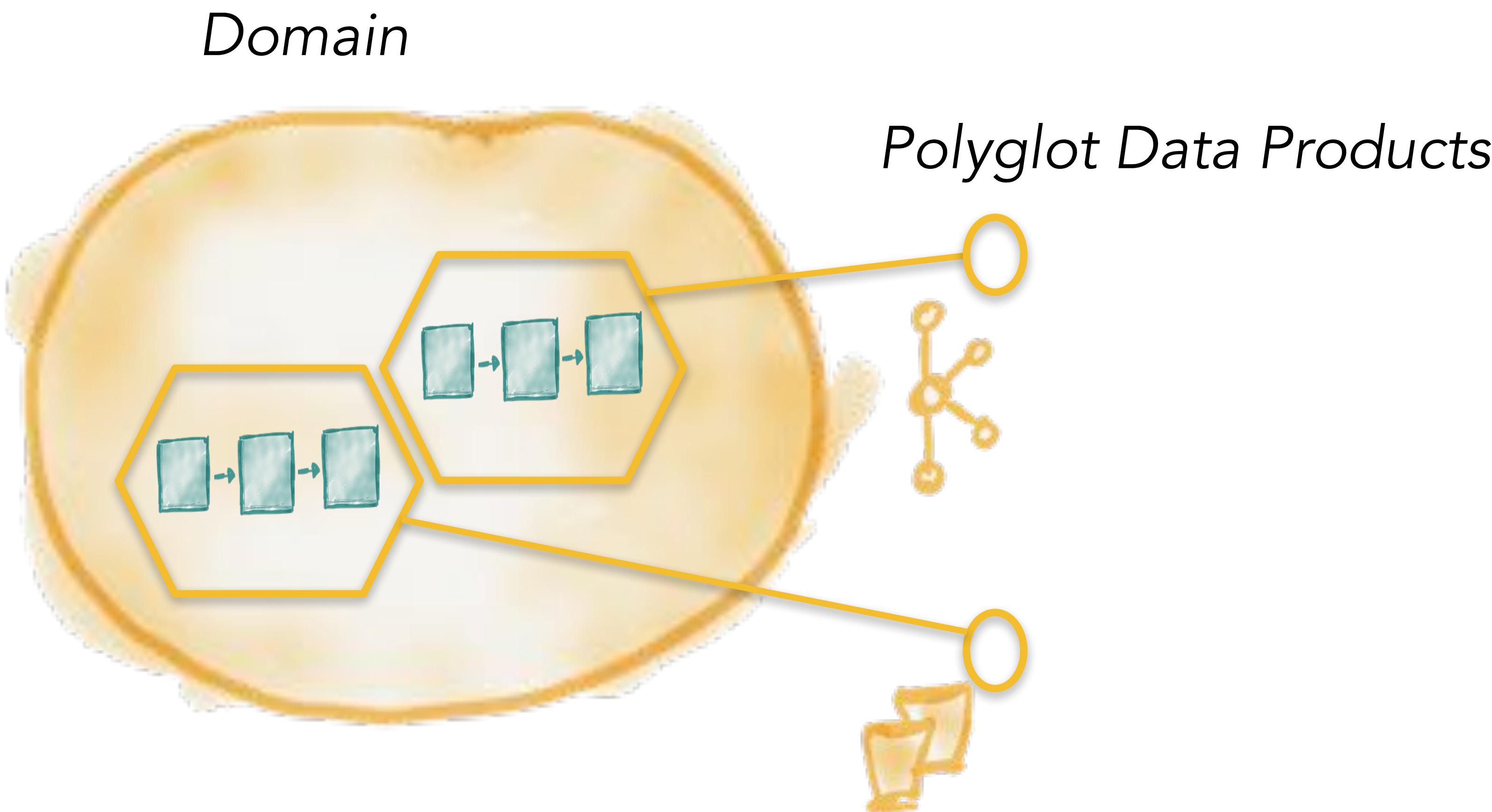
Architectural Quantum shifts from a
a pipeline to a domain (datasets)

Domain datasets are immutable (time series)



Product
Thinking

DOMAIN DATA AS A PRODUCT



DOMAIN DATA AS A PRODUCT

Aka Data Products



SHARED | DISCOVERABLE

DOMAIN DATA AS A PRODUCT

Aka Data Products



SHARED | DISCOVERABLE



ADDRESSABLE

DOMAIN DATA AS A PRODUCT

Aka Data Products



SHARED | DISCOVERABLE



ADDRESSABLE



TRUSTWORTHY
(DEFINED & MONITORED SLOs)

DOMAIN DATA AS A PRODUCT

Aka Data Products



SHARED | DISCOVERABLE



SELF-DESCRIBING



ADDRESSABLE



TRUSTWORTHY
(DEFINED & MONITORED SLOs)

DOMAIN DATA AS A PRODUCT

Aka Data Products



SHARED | DISCOVERABLE



SELF-DESCRIBING



ADDRESSABLE



INTER OPERABLE
(GOVERNED
BY GLOBAL STANDARDS)



TRUSTWORTHY
(DEFINED & MONITORED SLOs)

DOMAIN DATA AS A PRODUCT

Aka Data Products



SHARED | DISCOVERABLE



SELF-DESCRIBING



ADDRESSABLE



INTER OPERABLE
(GOVERNED
BY GLOBAL STANDARDS)



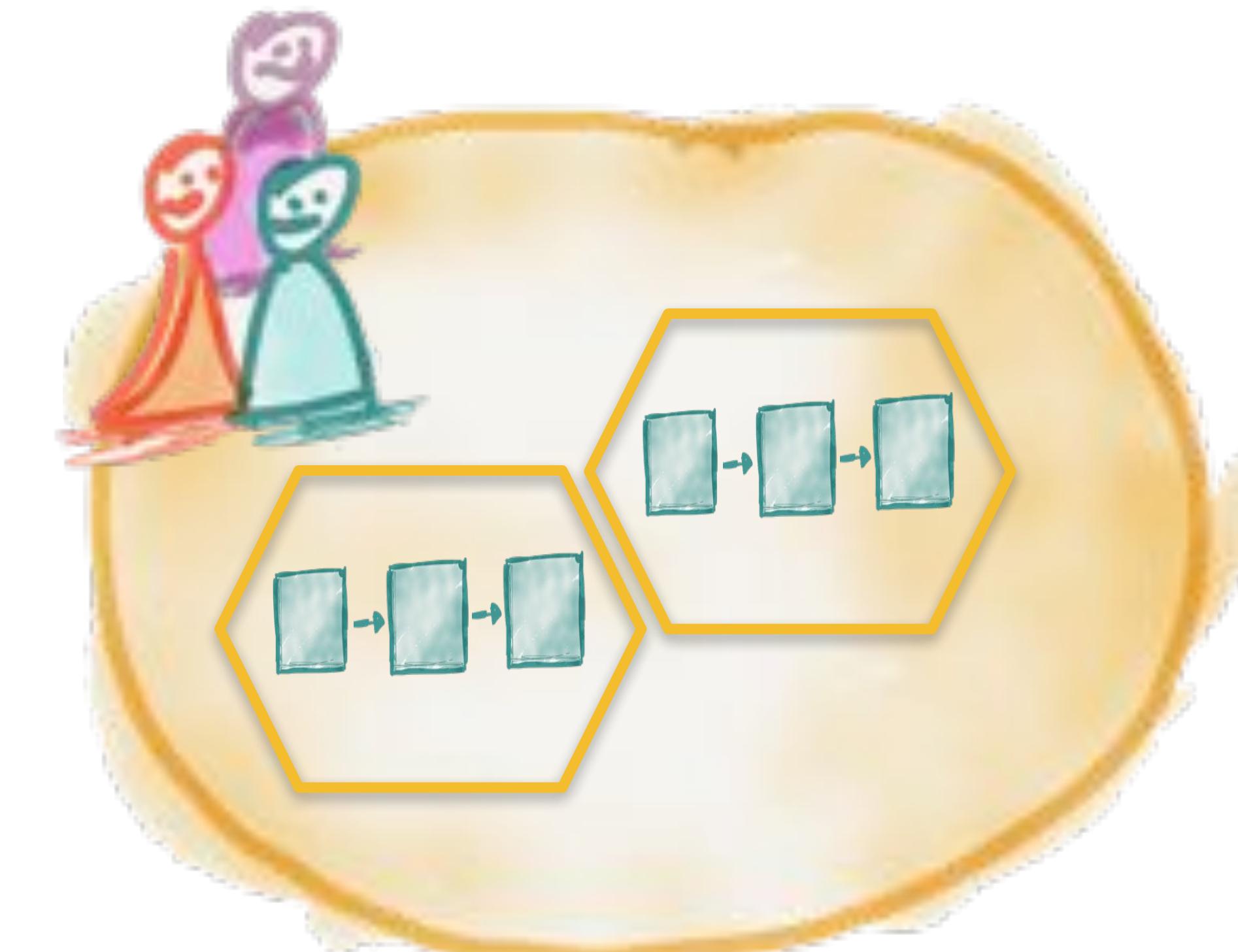
TRUSTWORTHY
(DEFINED & MONITORED SLOs)



SECURE
(GOVERNED
BY GLOBAL ACCESS
CONTROL)

CROSS-FUNCTIONAL DOMAIN TEAMS

Data Engineer
Domain Data Product Owner
Software Developer
Infra Engineer
...



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

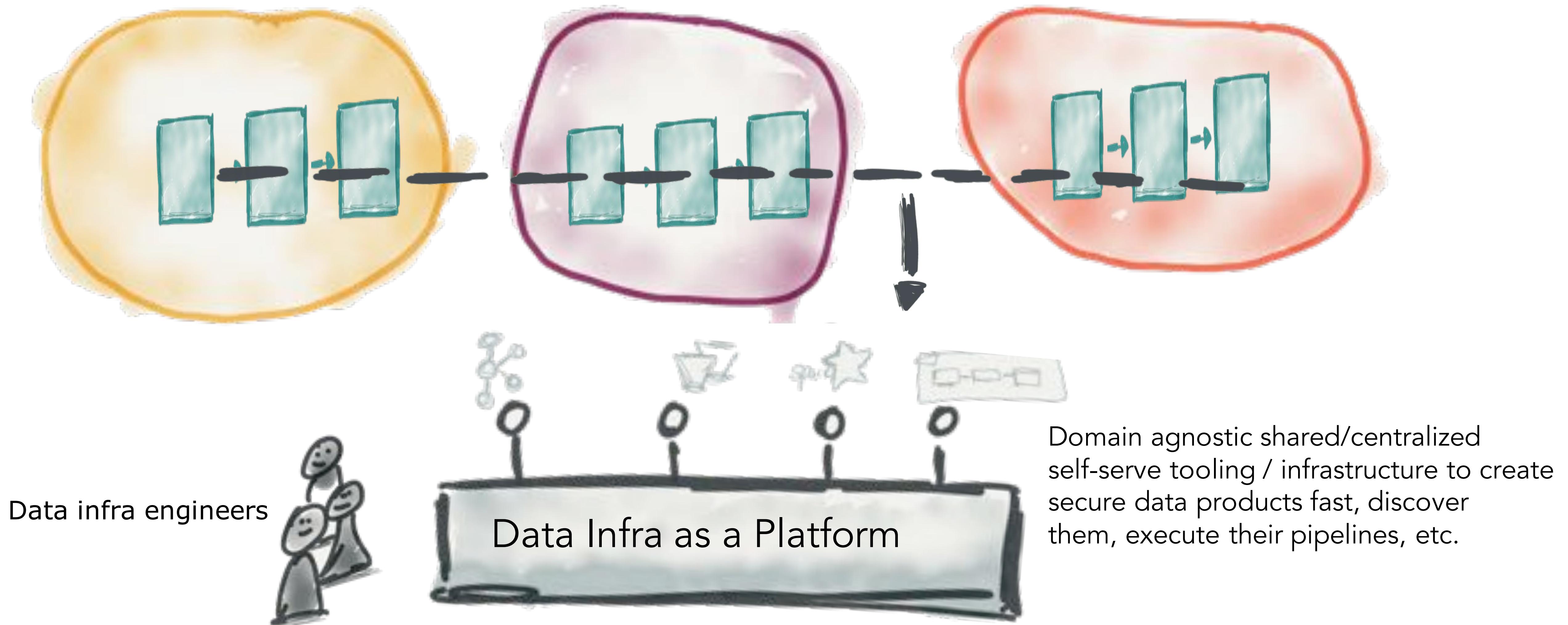
Success criteria: Decreased lead time to discover and consume a data product





Platform
Thinking

DATA INFR AS A PLATFORM



Scalable polyglot storage on demand

Encryption for data at rest and in motion

Unified data access control

Data product discoverability

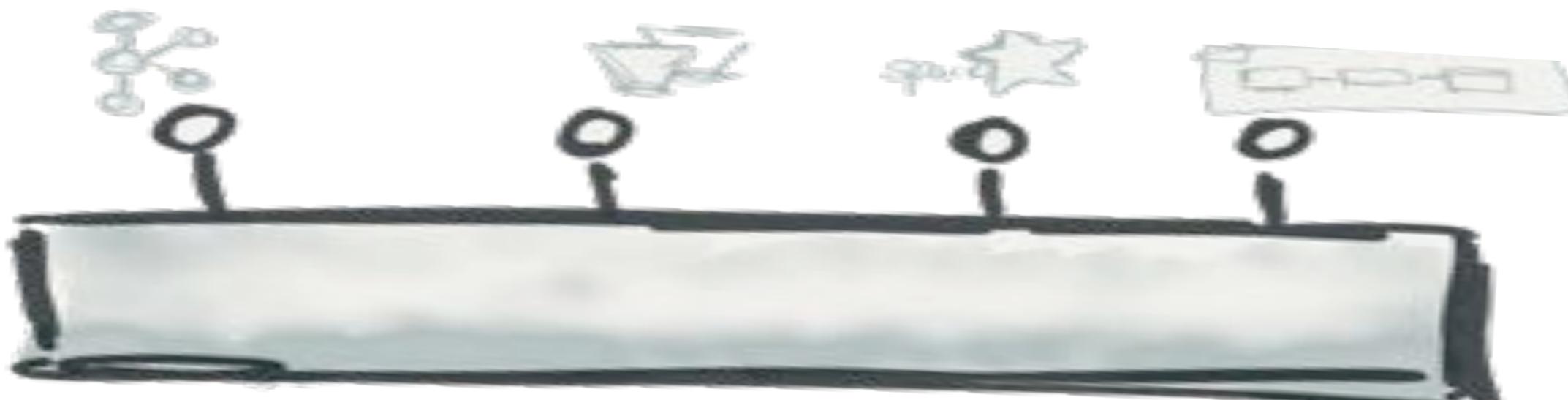
Data product SLO / metrics collection & sharing

Data pipeline orchestration / templates

Data Product CI/CD pipeline

Automate ecosystem governance
Guidelines

Data product scaffolding



Data infrastructure & Tooling (DataOps)

Shared, Self-serve, as a Platform

Domain agnostic

Owned by data infra and tooling team

At incubation it's opinionated

Ideally built on Cloud data services (despite lack of maturity)

Success criteria: reduced lead time to create new secure & discoverable data products



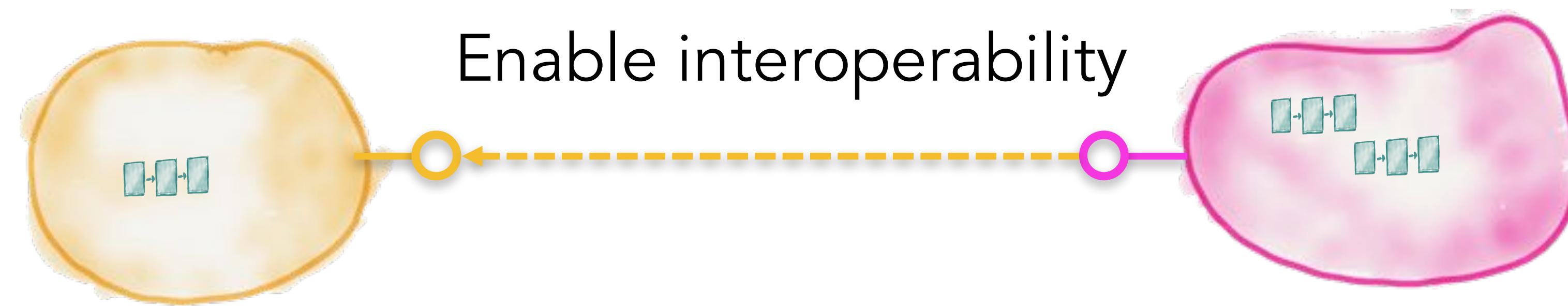


Ecosystem
Governance

Federated & Global Ecosystem Governance



Enable discoverability



Enable interoperability

Automated federated identity

Automated policy enforcement:
Encryption, access control, etc.

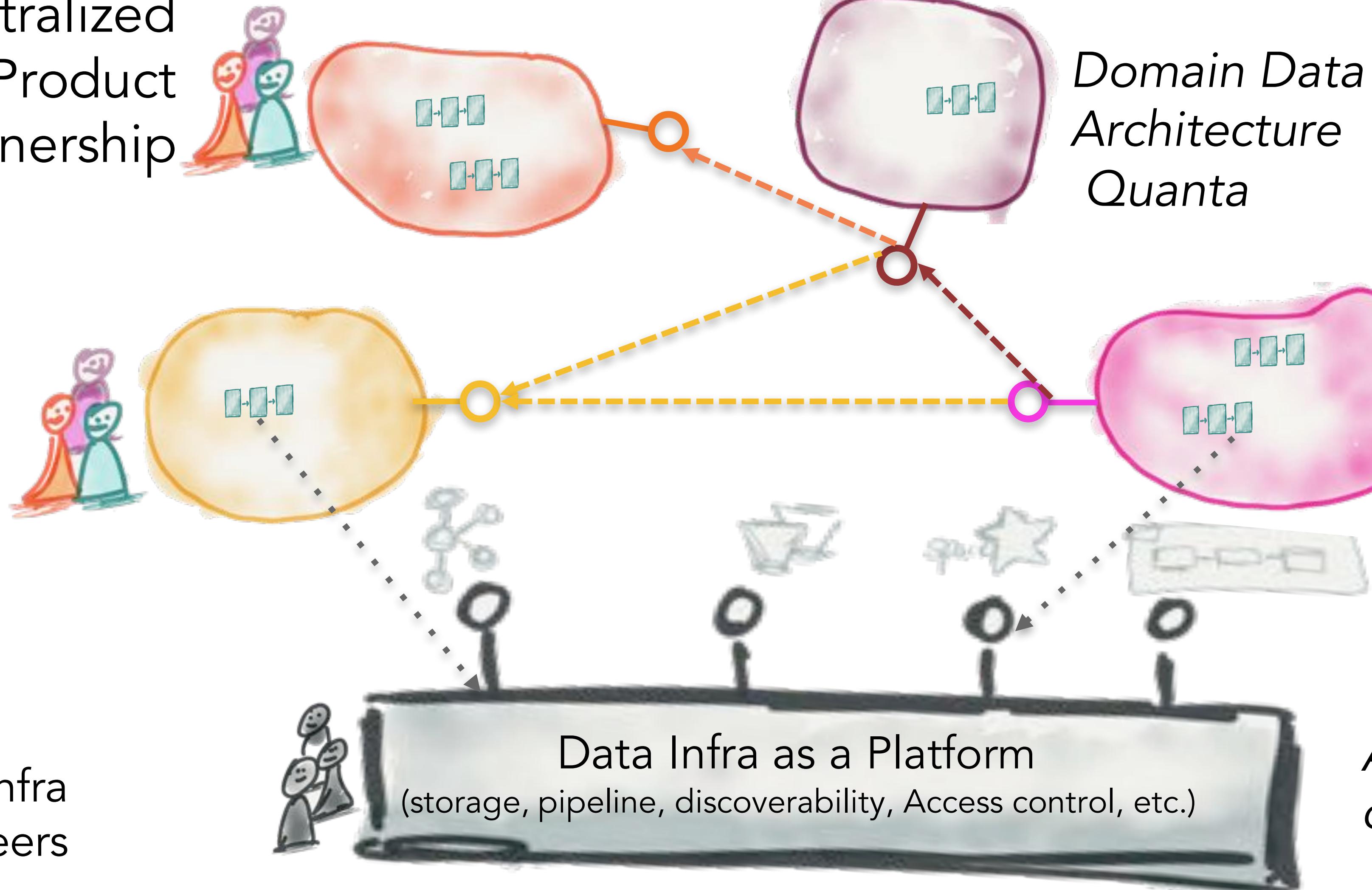
Governance enforced through automation

LET'S BRING IT TOGETHER

DATA MESH ARCHITECTURE

Federated Ecosystem Governance
(enable interoperability)

Decentralized
Data Product
ownership

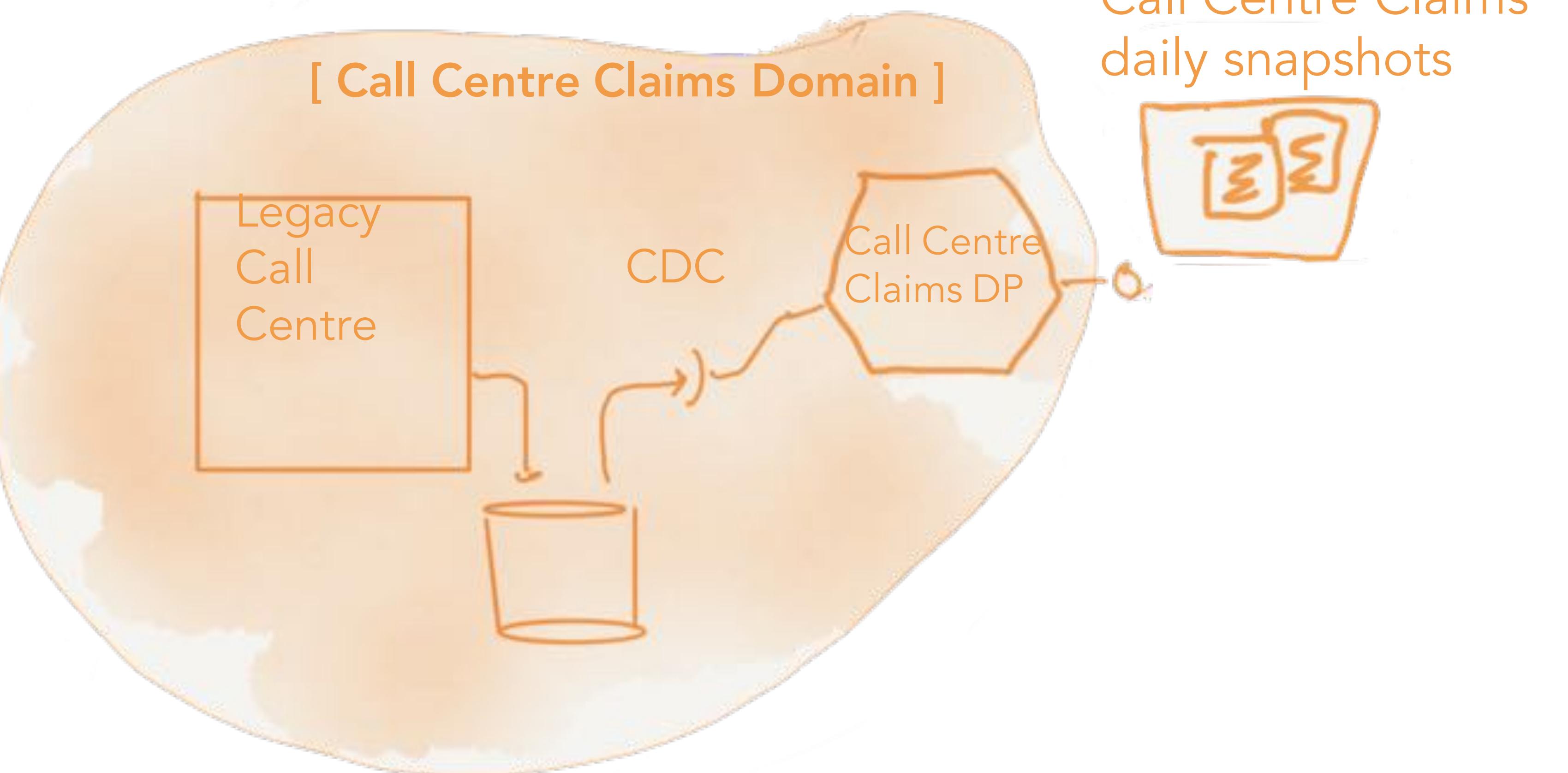


Data infra
engineers

DOMAIN DATA PRODUCTS

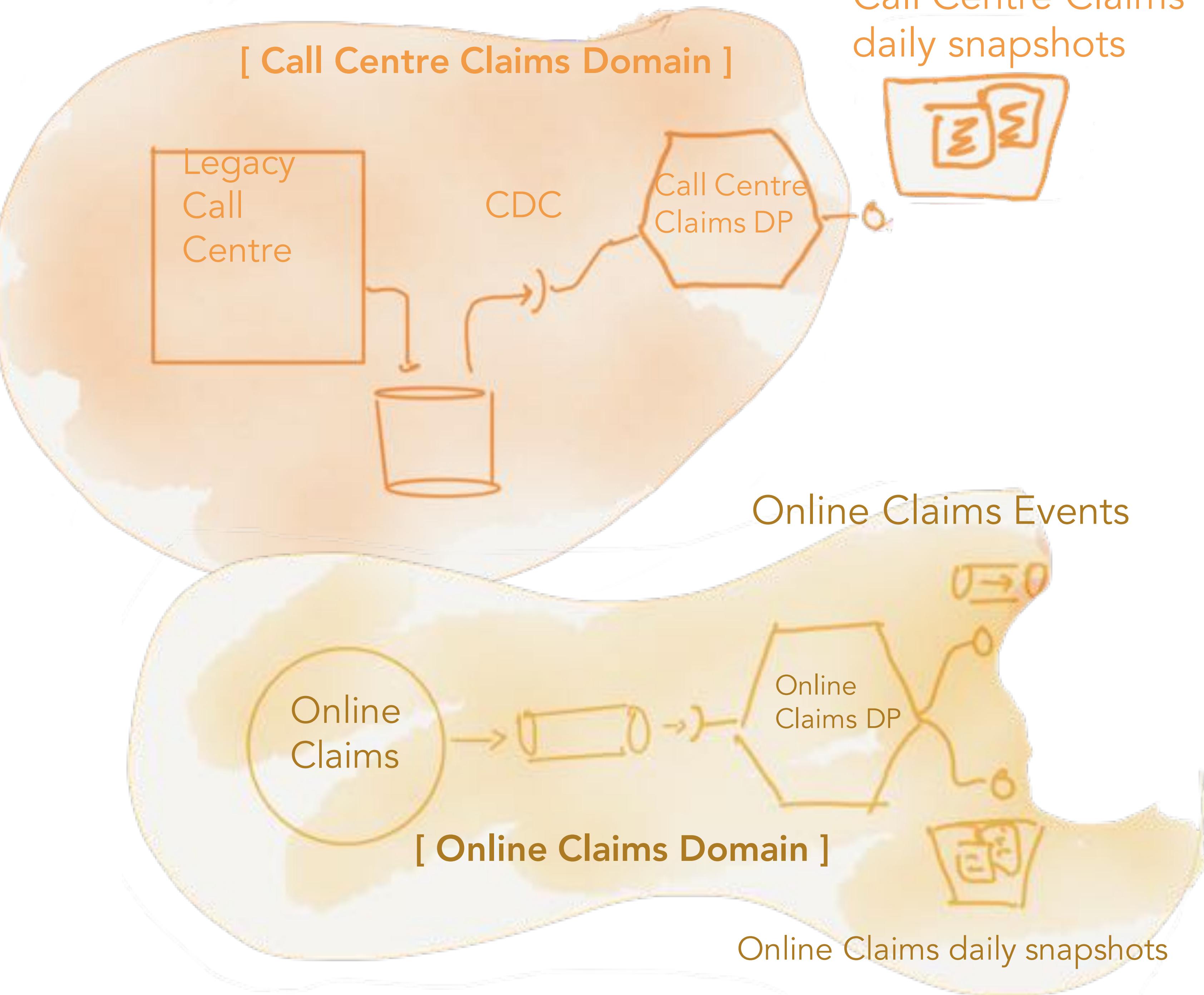
EXAMPLE

Native | Aggregate Data Products



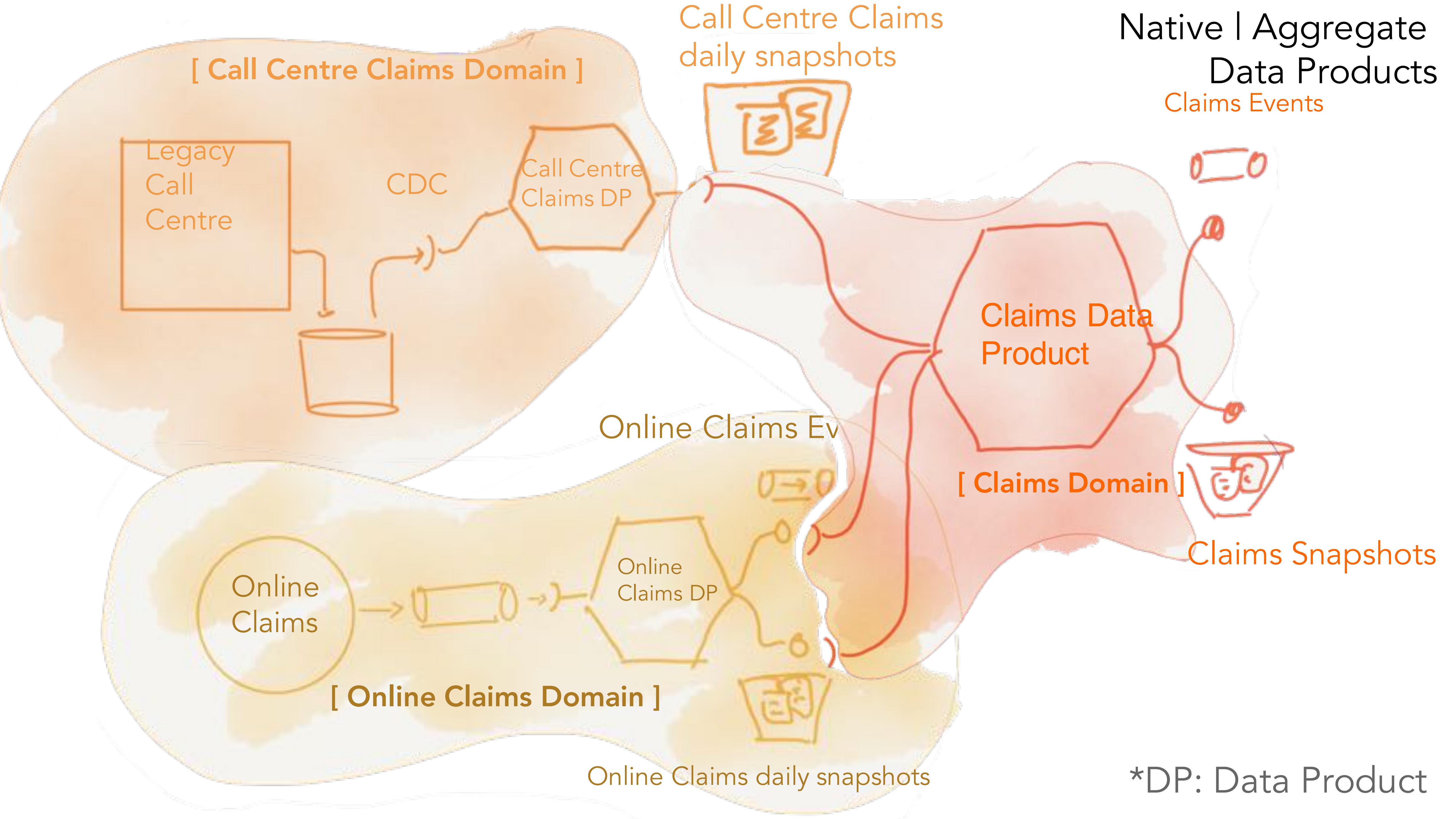
*DP: Data Product

Native | Aggregate Data Products



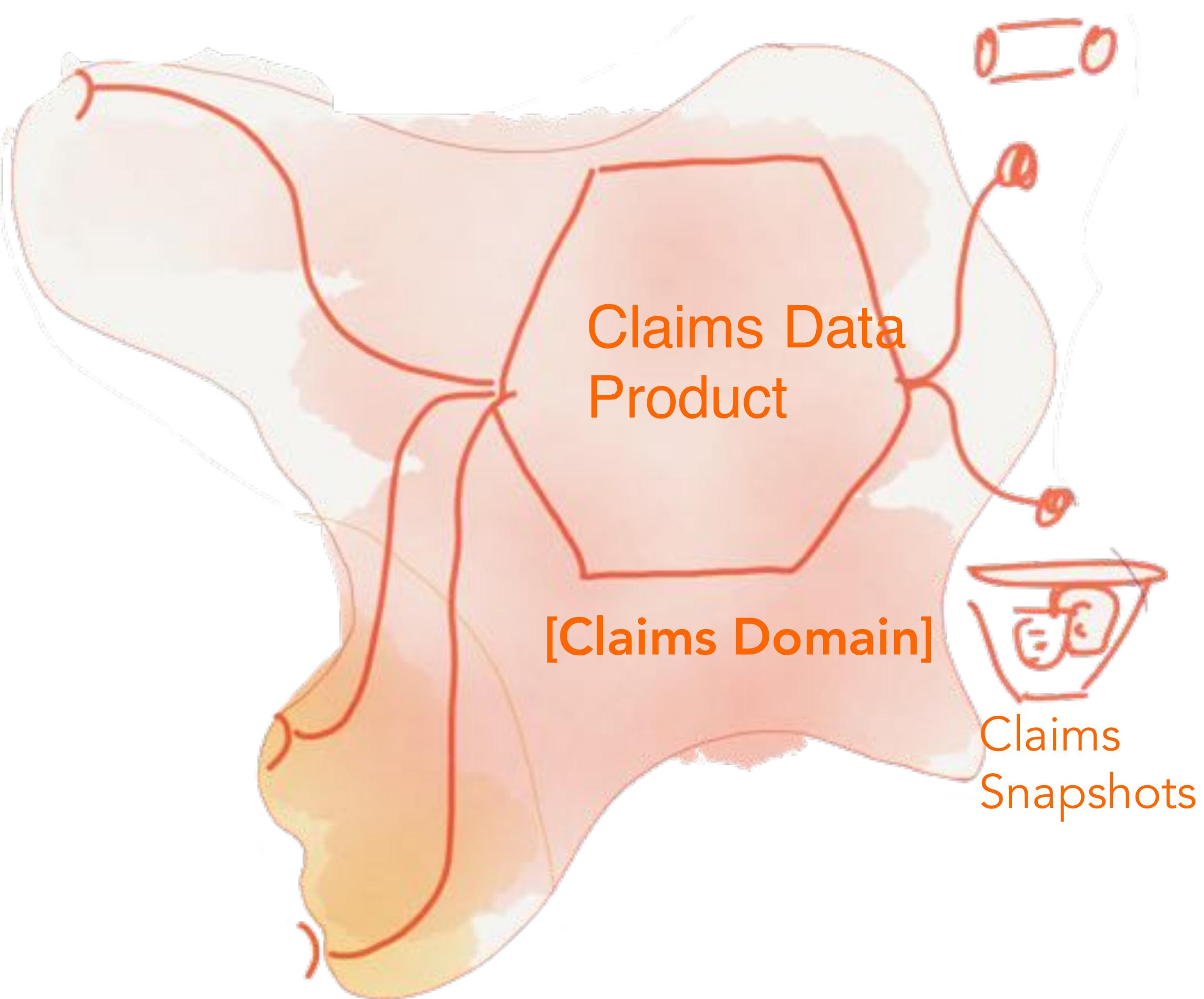
*DP: Data Product

Native | Aggregate Data Products



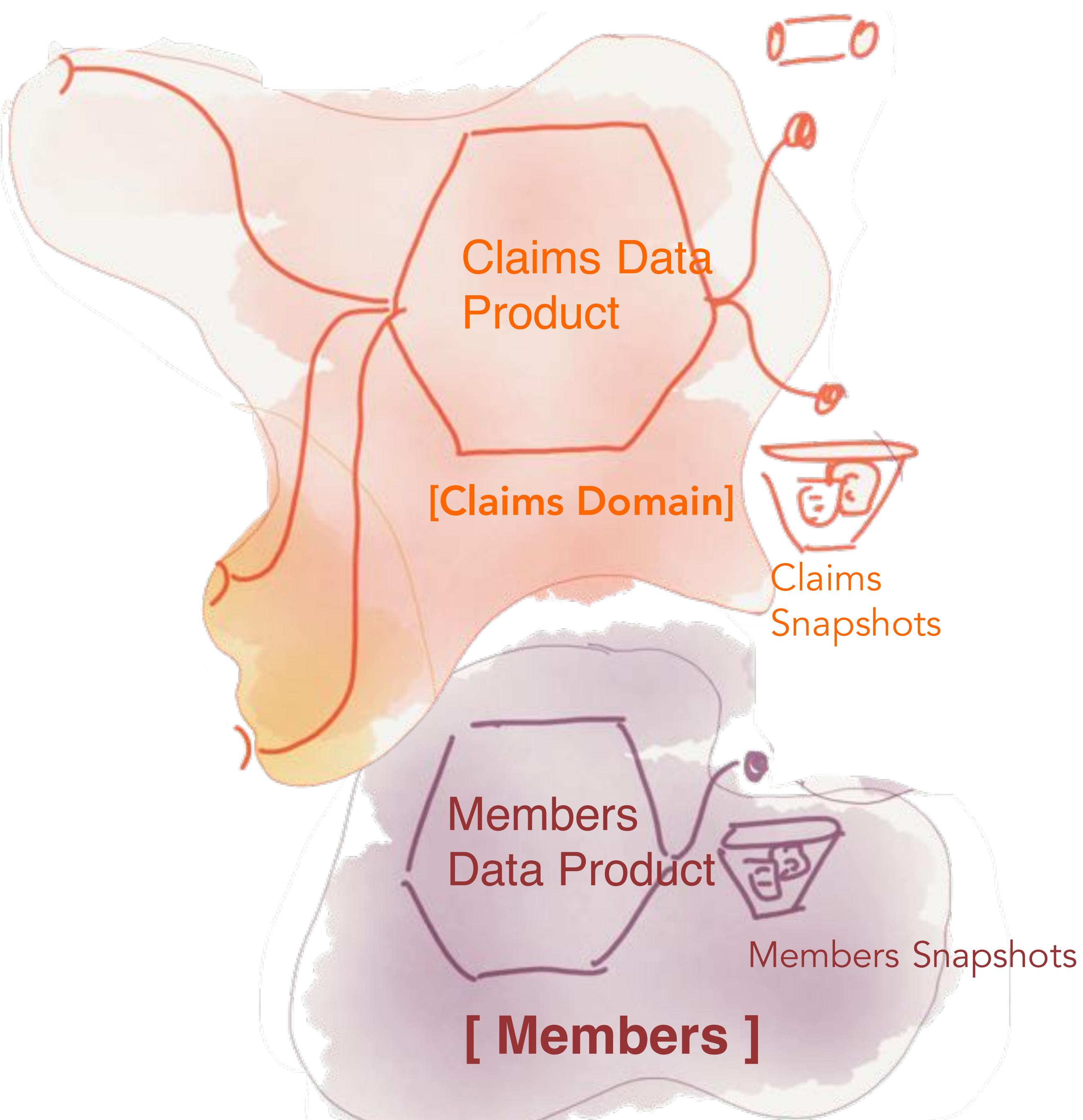
Claims Events

Aggregate | Fit-for-purpose
Data Products



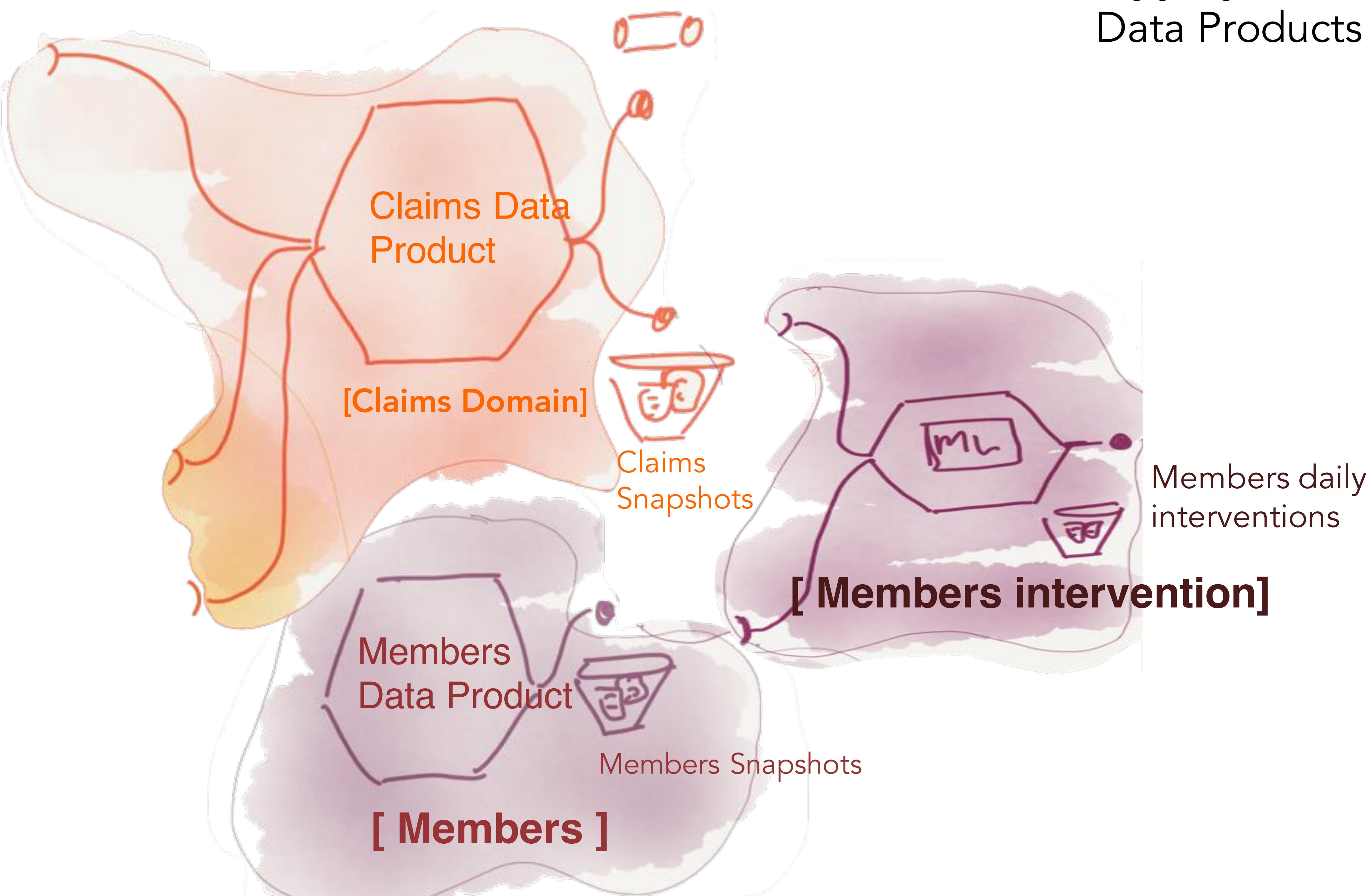
Claims Events

Aggregate | Fit-for-purpose
Data Products



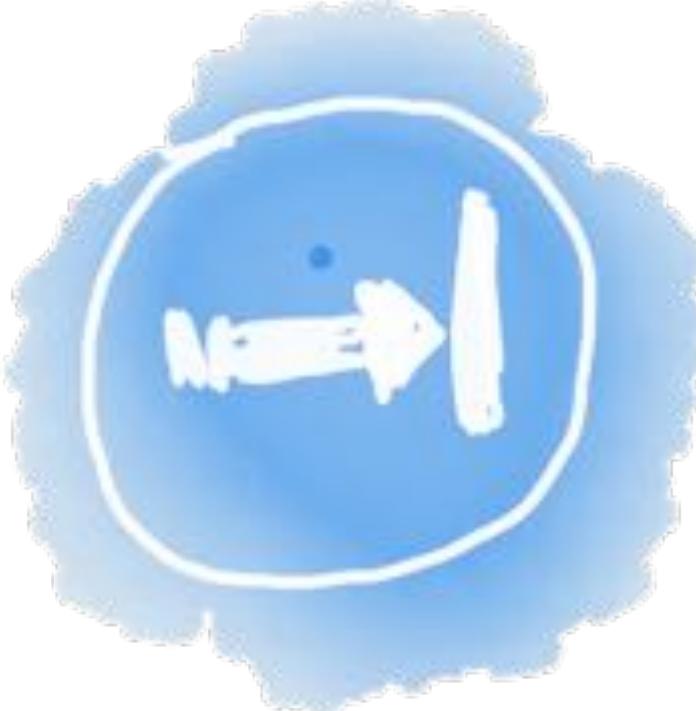
Claims Events

Aggregate | Fit-for-purpose
Data Products



DATA PRODUCT

DEEP DIVE



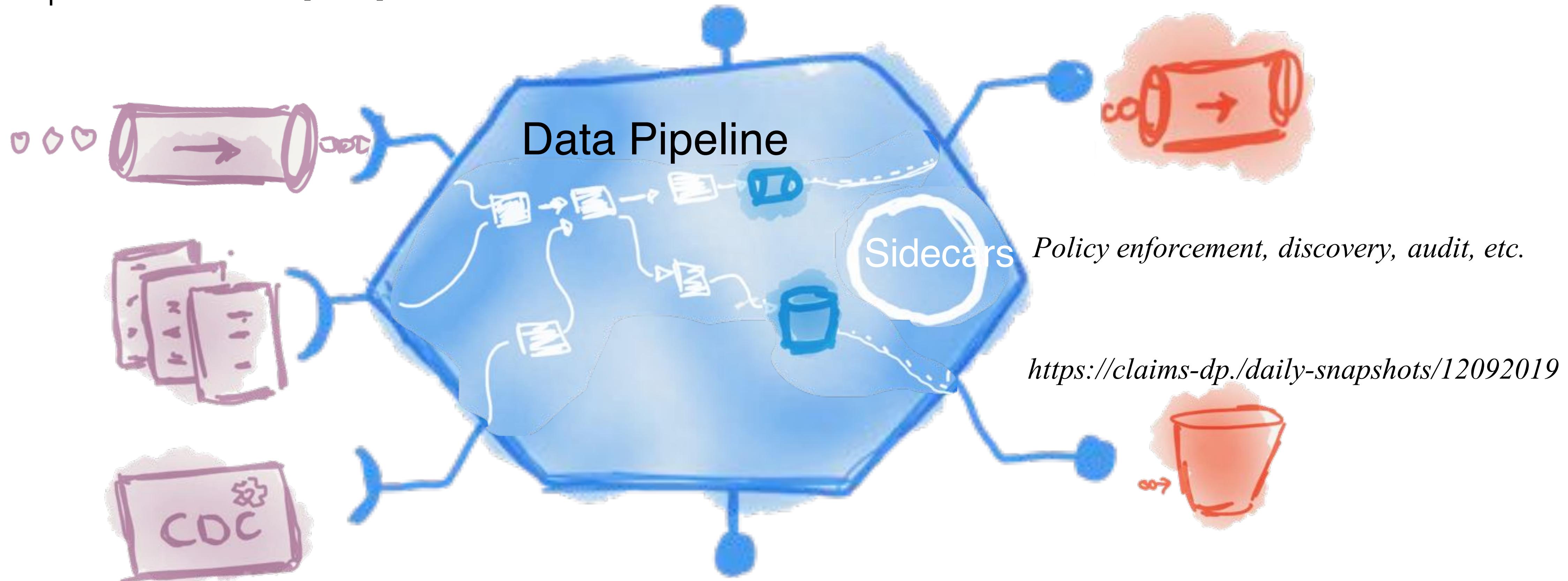
Polyglot Input Data Ports [1..n]



Control Ports



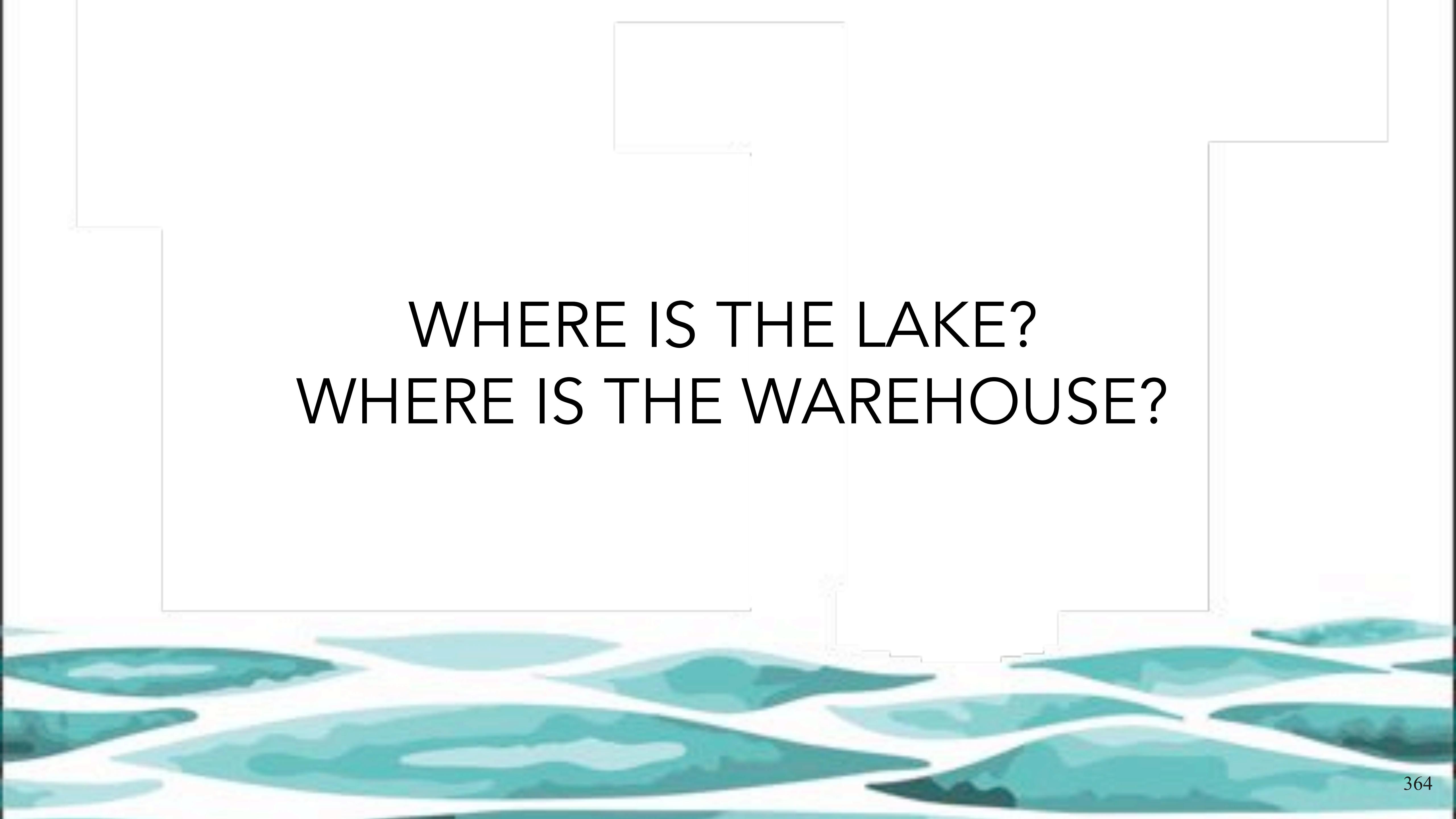
Polyglot Input Data Ports [1..m]



<https://members-dp/controls/describe>

On-prem to cloud

On-prem to cloud



**WHERE IS THE LAKE?
WHERE IS THE WAREHOUSE?**

PARADIGM SHIFT

FROM

TO

Centralized ownership

Decentralized ownership

Monolithic

Distributed

Pipeline first class concern

Domain first class concern

Data as a by-product

Data as a product

Siloed data engineering team

Cross-functional data domain teams





FROM

LANGUAGE

TO

Ingesting

Serving

Extracting & Loading

Discovering & Consuming

Flowing data through centralized Pipelines

Publishing output data ports

Centralized Data Lake | Warehouse | Platform

Ecosystem of Data Products

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

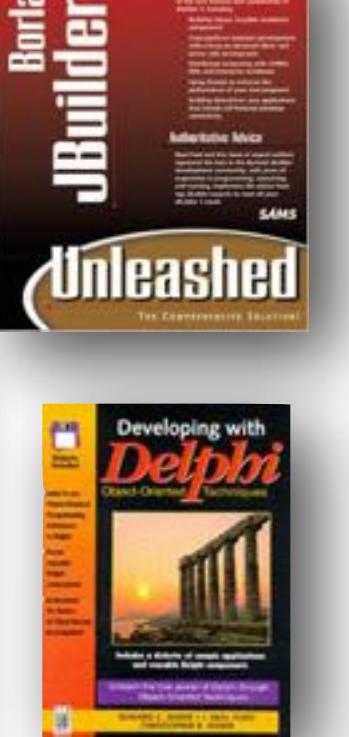
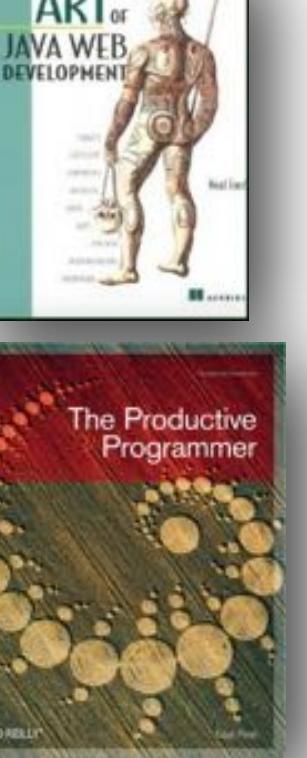
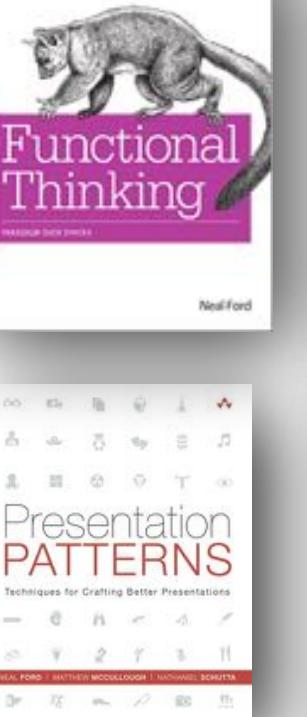
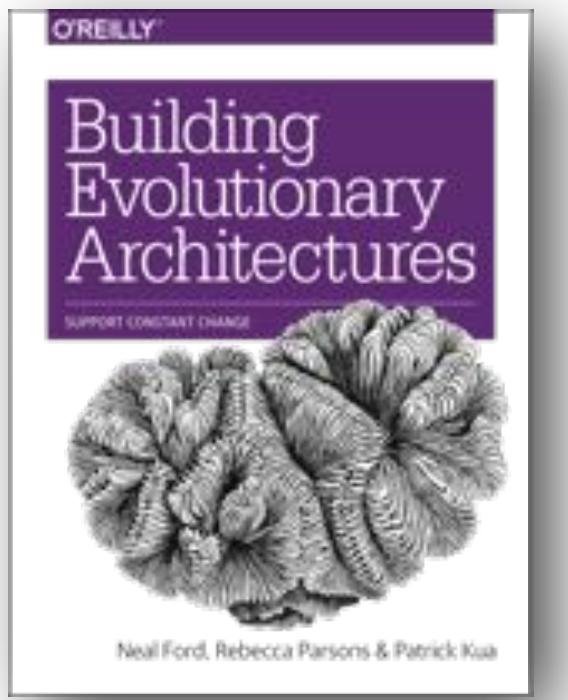
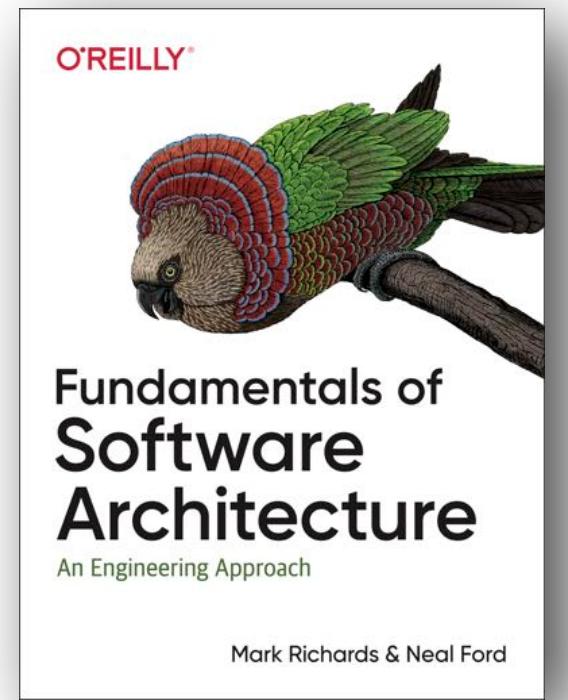
Many enterprises are investing in their next generation data lake, with the hope of democratizing data at scale to provide business insights and ultimately make automated intelligent decisions. Data platforms based on the data lake architecture have common failure modes that lead to unfulfilled promises 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](#)



www.thoughtworks.com/podcasts

evolutionaryarchitecture.com

learning.oreilly.com/live-training/

learning.oreilly.com/learning-paths/

fundamentalsofsoftwarearchitecture.com

ThoughtWorks®

NEAL FORD

Director / Software Architect / Meme Wrangler

@neal4d

<http://nealford.com>