Mark Richards
Independent Consultant
Hands-on Software Architect, Published Author
Founder, DeveloperToArchitect.com
@markrichardssa

Architecture: The Hard Parts
Scenarios

Neal Ford
ThoughtWorks
Director / Software Architect / Meme Wrangler
http://www.nealford.com
@neal4d

v2020-06-22

Architecture: The Hard Parts

1. How do I effectively analyze trade-offs and document architecture decisions?
2. How do I choose an appropriate architecture?
3. How do I determine the appropriate level of granularity?
4. How do I handle workflow and transactions in a distributed architecture?
5. How do I automate architectural governance?
6. How do I create systems with high semantic coupling but low syntactic coupling?
7. How do I manage data in a distributed system?
8. How do I achieve high levels of scalability and elasticity in a system?

The Sysops Squad

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

the +'s and –'s of *one* thing

– negative tradeoffs

+ positive tradeoffs

tradeoffs

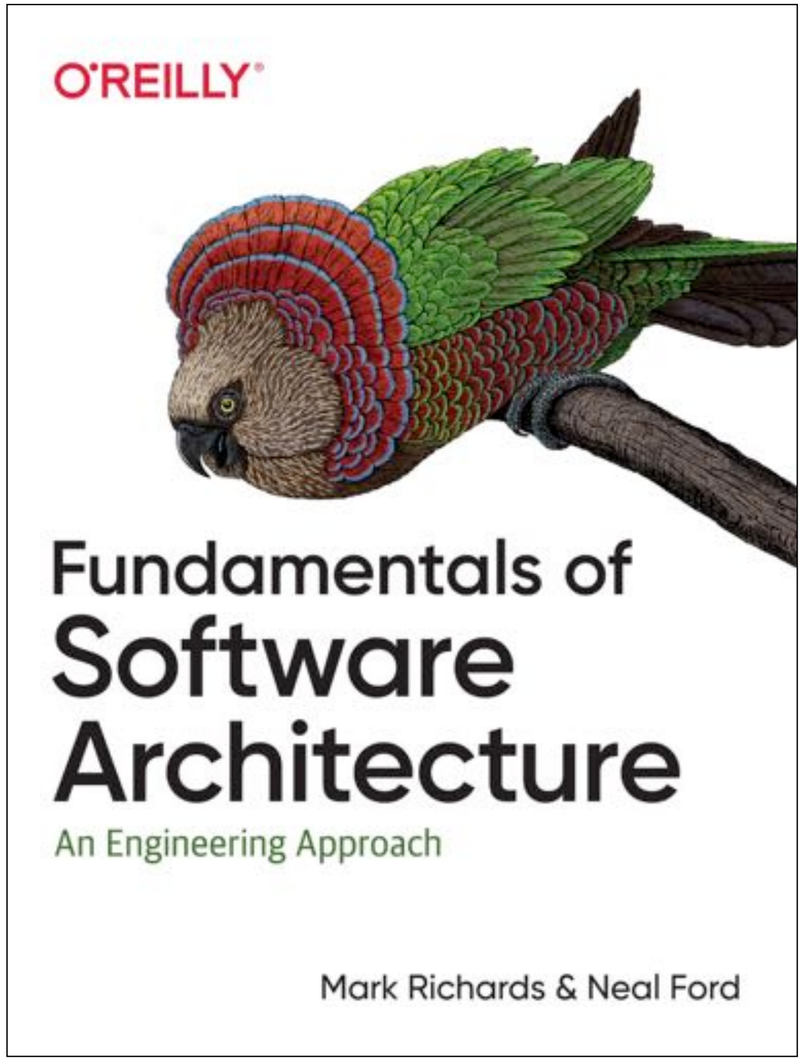
this ***OR*** that

* stuff about ***this***

* stuff about ***that***

How do I effectively analyze trade-offs and document architecture decisions?

documenting architecture decisions



Second Law of Software Architecture

*"**Why** is more important than **how**"*

documenting architecture decisions

"We will keep a collection of records for *architecturally significant decisions*: those that affect the structure, non-functional characteristics, dependencies, interfaces, or construction techniques."

- Michael Nygard

http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions

documenting architecture decisions

short text file; 1-2 pages long, one file per decision
markdown, textile, asciidoc, plaintext, etc.

short noun phrase

proposed, accepted, superseded

forces at play

response to forces

context after decision is applied

documenting architecture decisions

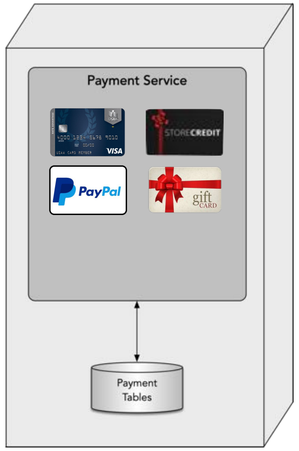
short text file; 1-2 pages long, one file per decision
markdown, textile, asciidoc, plaintext, etc.

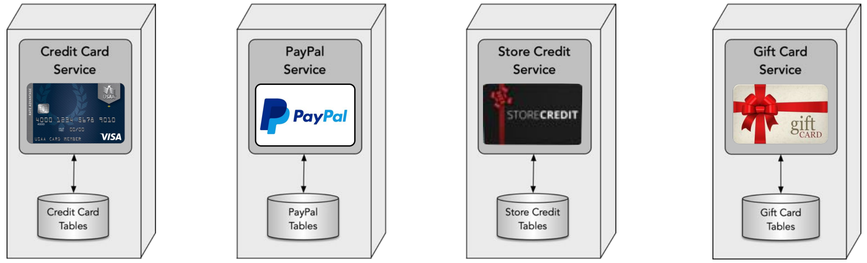
description of the problem and alternative solutions available (documentation)

justification (the "why")

tradeoffs and impact of decision

documenting architecture decisions





maintenance, testing, and deployment for changes and additions

transactions and data integrity when combining payment types

customer must wait until payment is applied when placing order

4. Single consolidated payment service

Context

A customer can use any payment type combination (credit card, paypal, store credit, and gift cards) when placing an order. Payment is required when pacing an order (synchronous activity). The payment logic can be either a single consolidated service or a separately deployed service for each payment type.

Decision

We will use a single consolidated payment service for all payment types.

Separating the services would require orchestration, which incurs both network and security latency. A single payment service will be more responsive since it does not require orchestration of each payment, therefore improving customer satisfaction when placing orders.

Error handling and transactions are maintained in a single consolidated service, improving overall data integrity, data consistency, and performance when errors occur. This also improves overall customer satisfaction when placing orders.

Consequences

The tradeoff of this decision is that maintenance, testing, and deployment will be impacted when changes are made to any of the payment types or another payment type is added. Testing scope is increased, requiring more time for complete testing. Deployment risk is increased as deploying a single service requires deployment of all payment types.

The maintenance impact tradeoff was accepted by the business stakeholders in favor of better performance and data consistency when

4. Single consolidated payment service

Status

Proposed

Context

A customer can use any payment type combination (credit card, paypal, store credit, and gift cards) when placing an order. Payment is required when pacing an order (synchronous activity). The payment logic can be either a single consolidated service or a separately deployed service for each payment type.

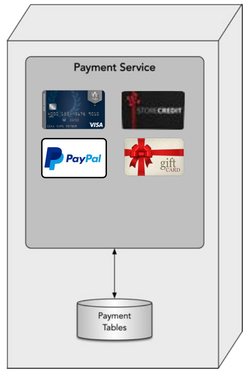


Figure 1: Single service for payment

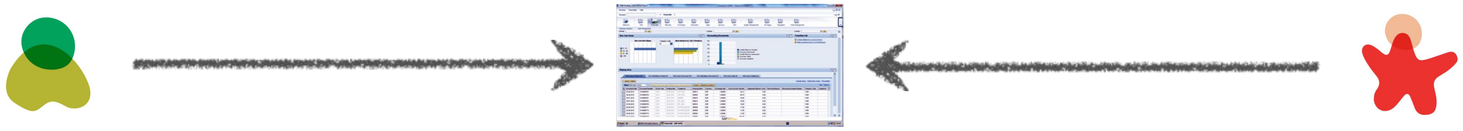
Decision

We will use a single consolidated payment service for all payment types.

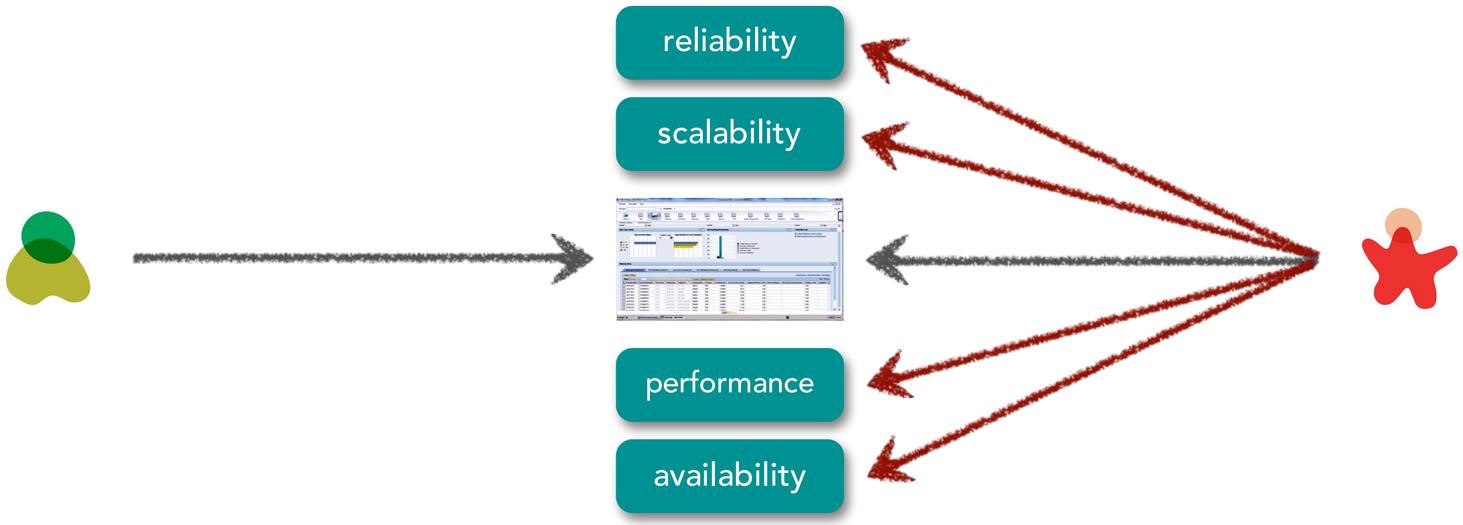
Separating the services would require orchestration, which incurs both network and security latency. A single payment service will be more responsive since it does not require orchestration of each payment, therefore

How do I choose an appropriate architecture?

architecture characteristics

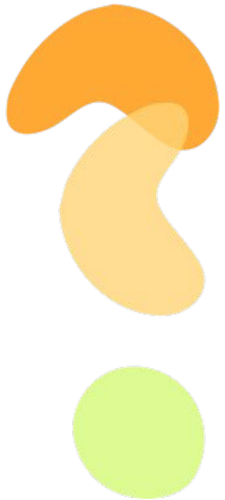


architecture characteristics



architecture characteristics

1. synergistic



2. ill-defined

3. voluminous

architecture characteristics

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

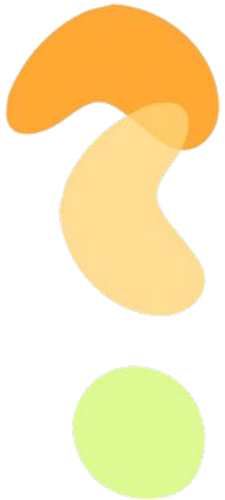
evolvability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
interchangeability
interoperability
learnability
maintainability
manageability
mobility
modifiability
modularity
operability
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

architecture characteristics

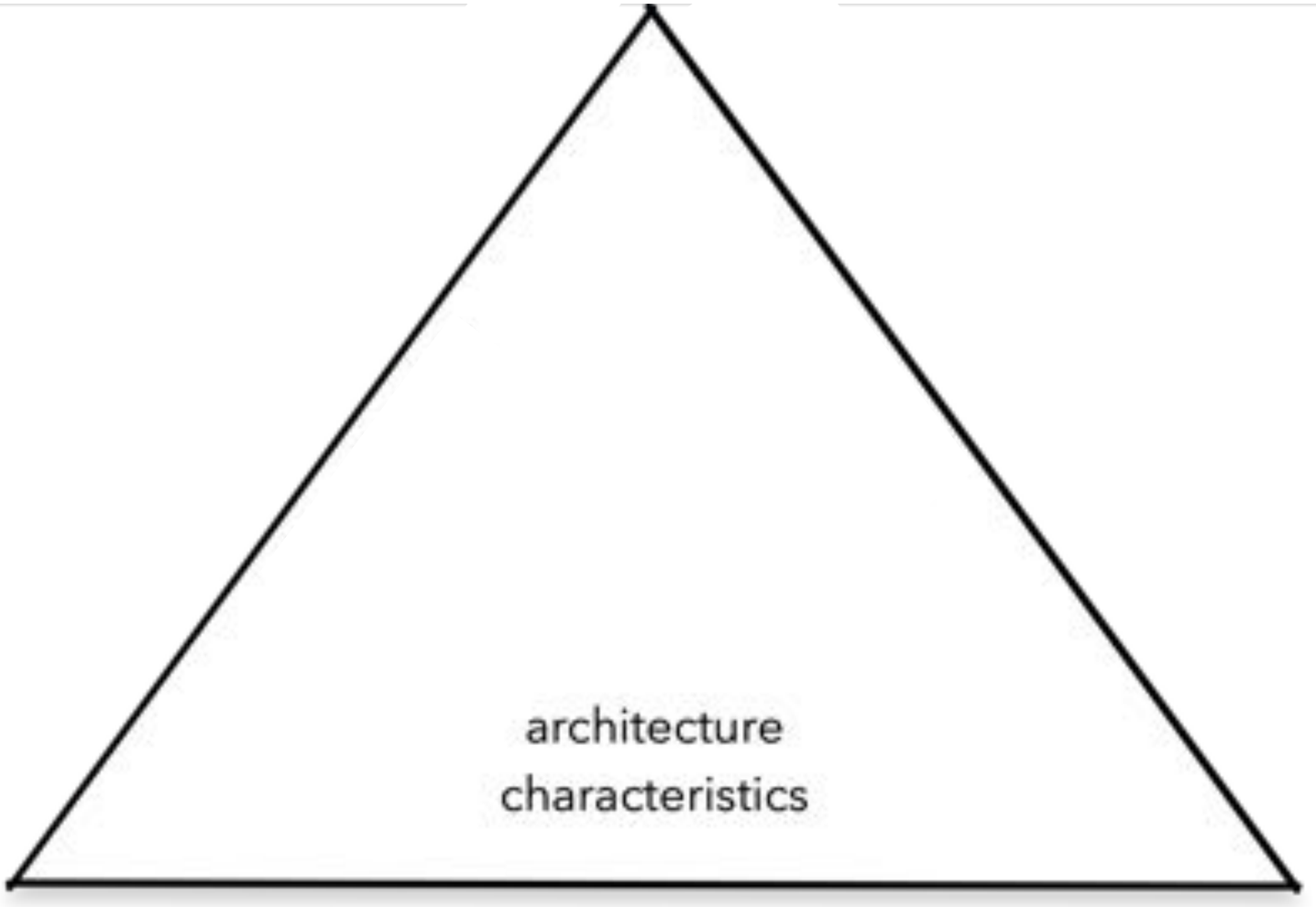
1. synergistic

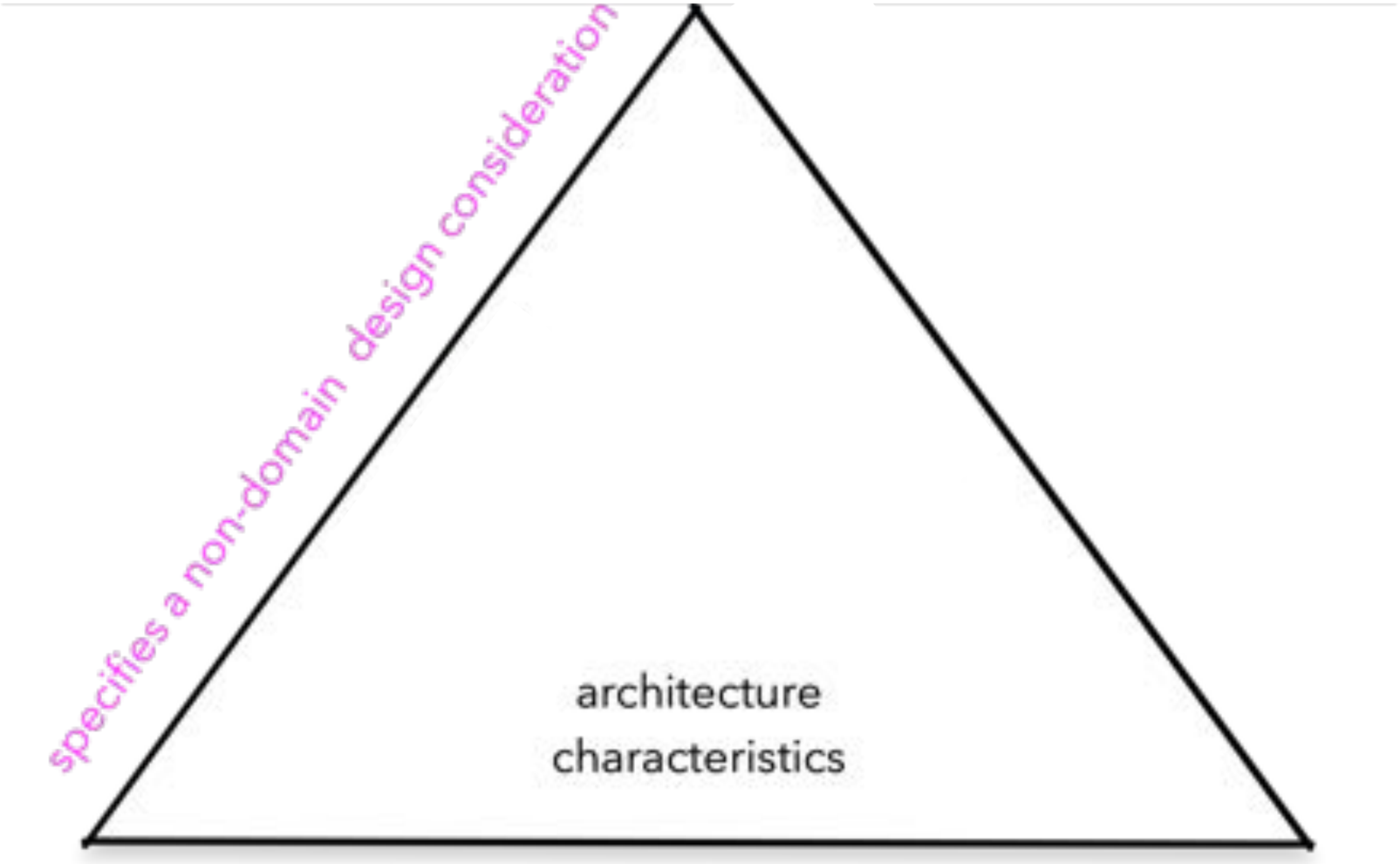


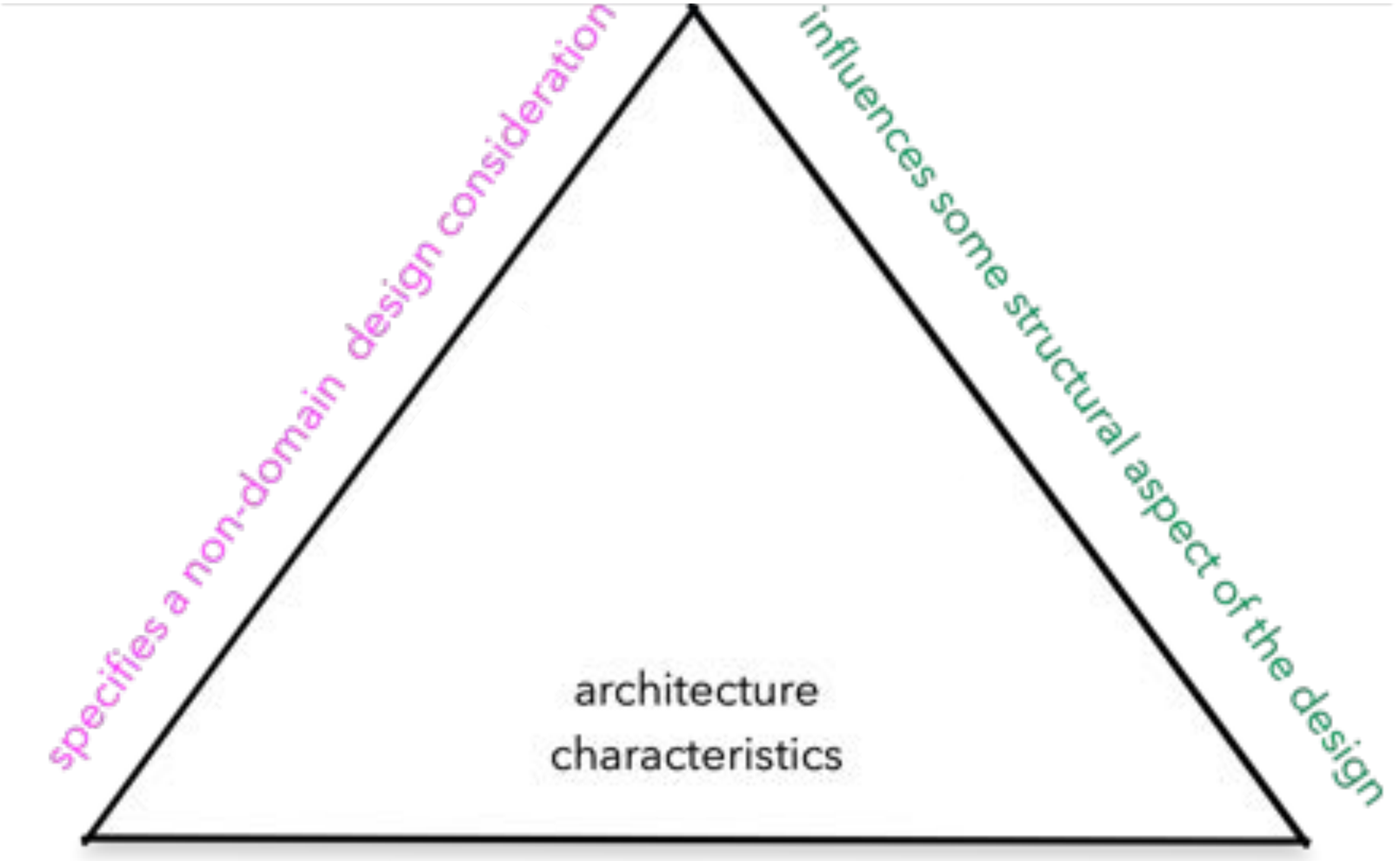
2. ill-defined

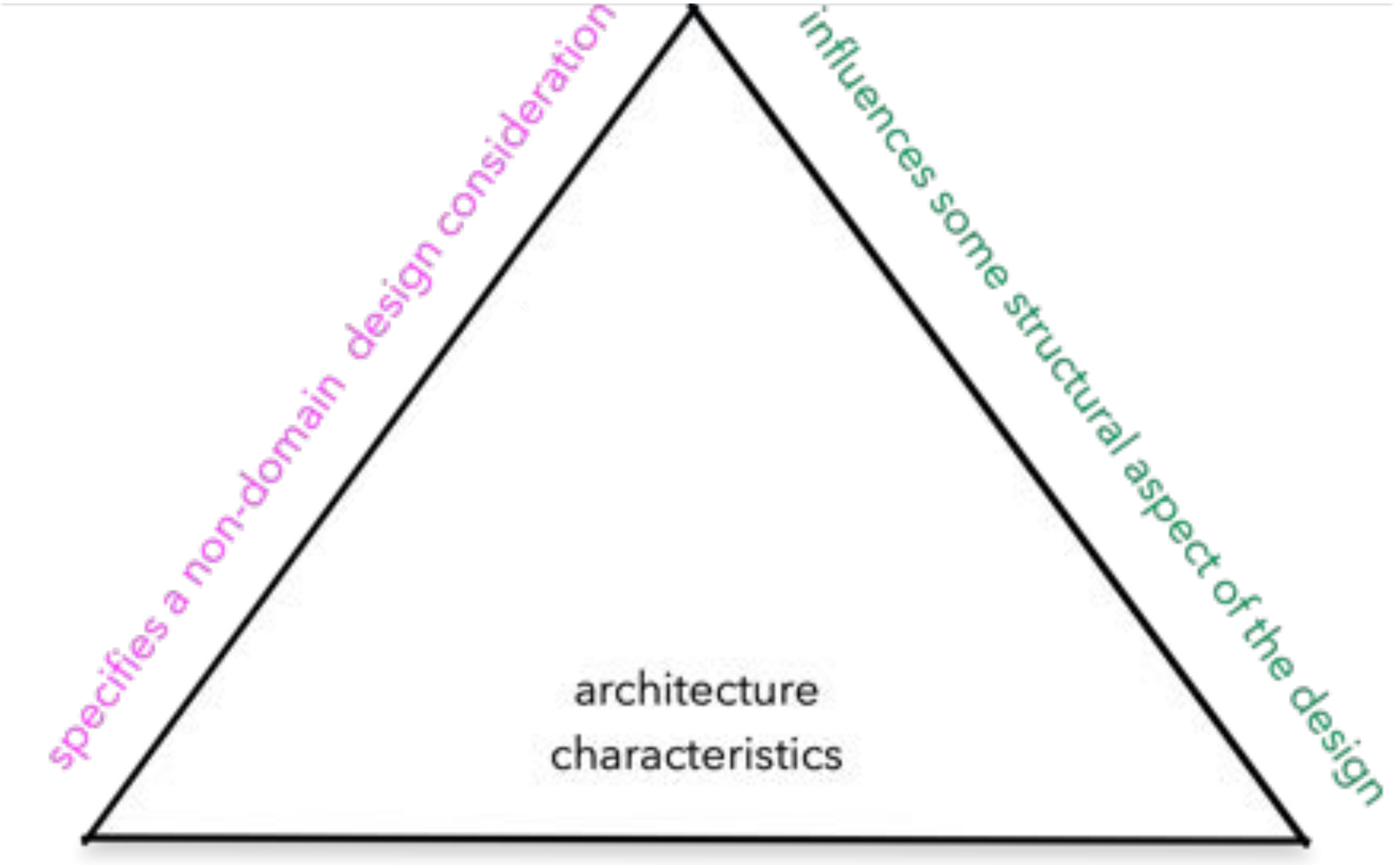
4. numerous categories

3. voluminous

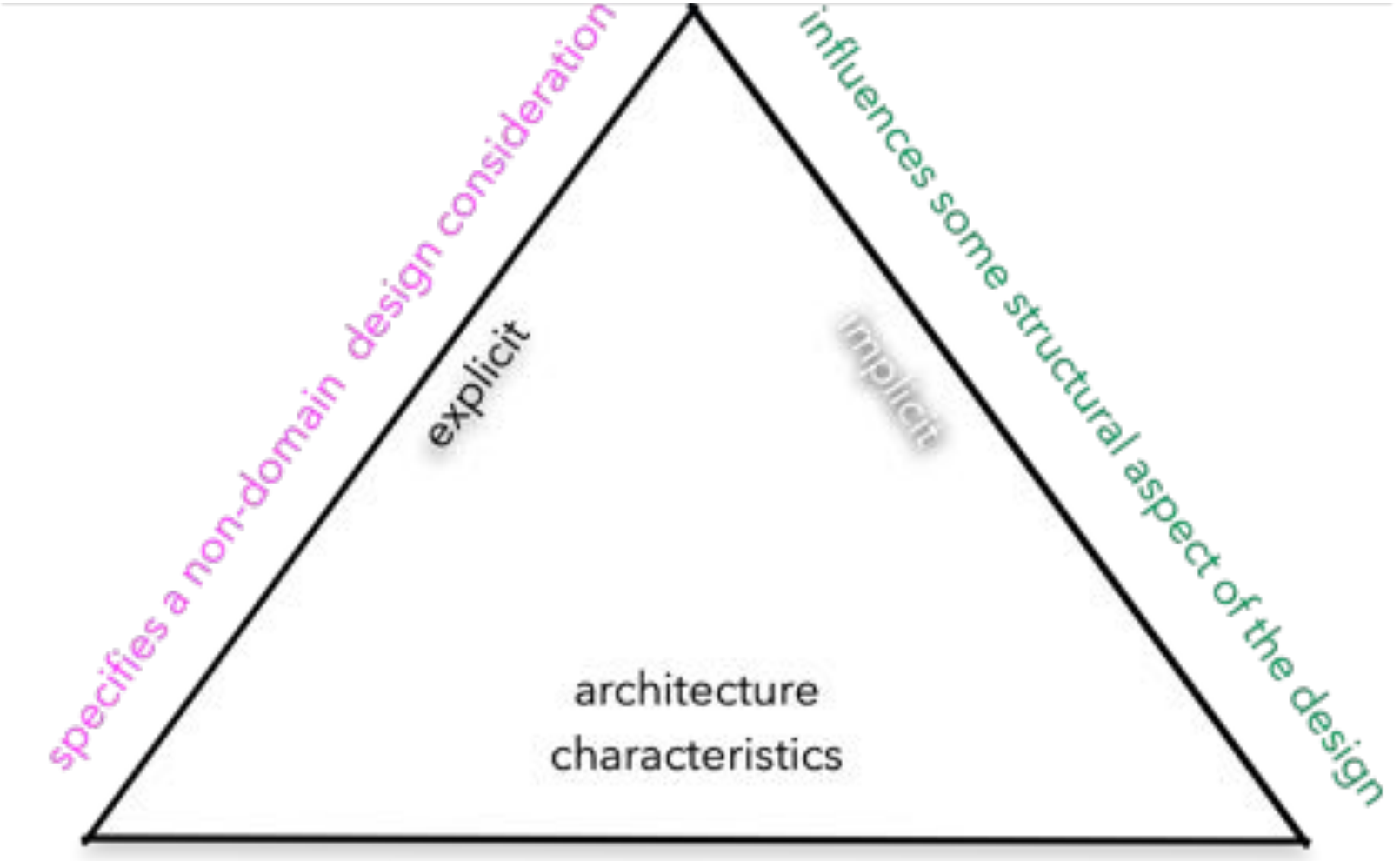

architecture
characteristics


specifies a non-domain design consideration
architecture
characteristics

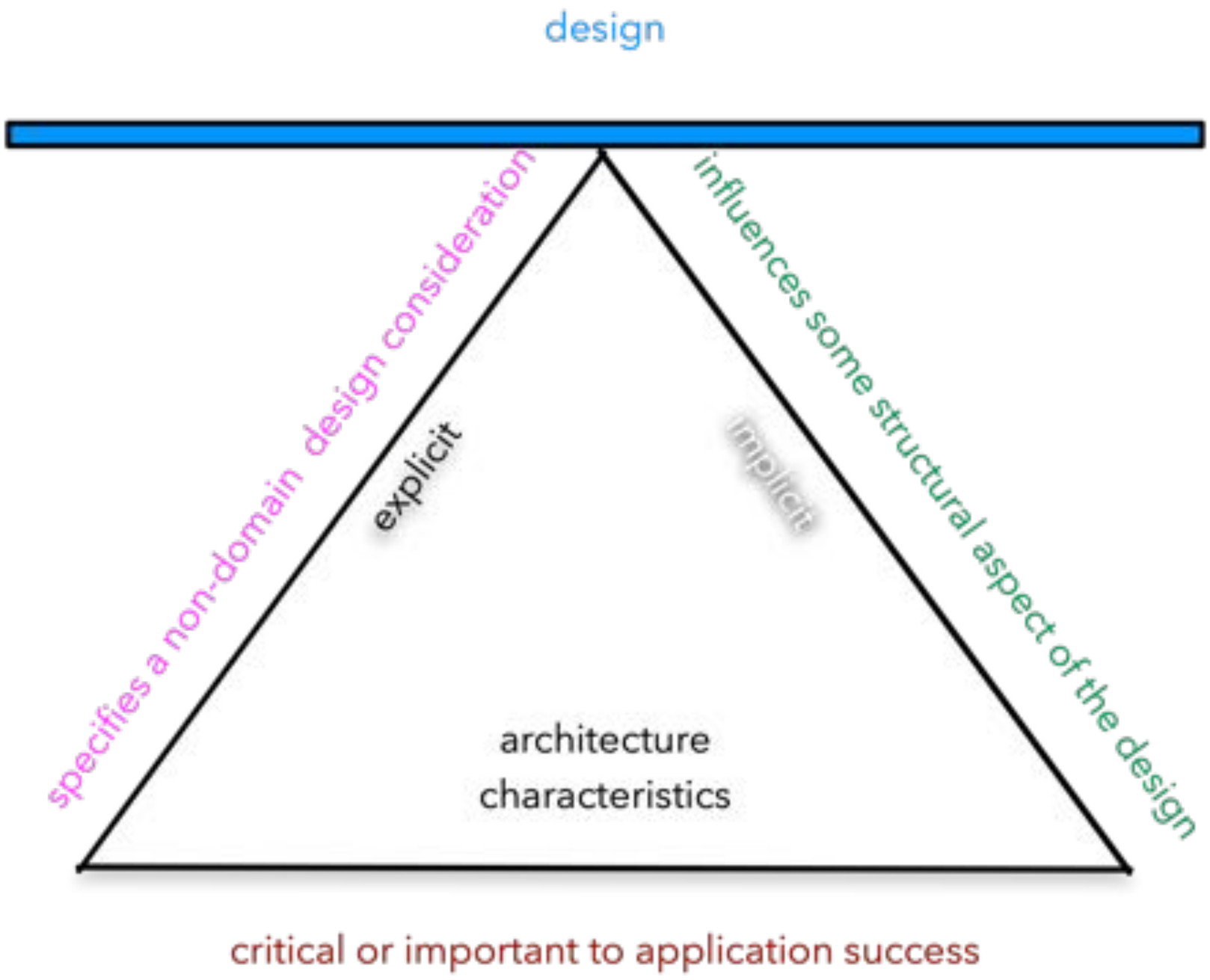

specifies a non-domain design consideration
influences some structural aspect of the design
architecture
characteristics



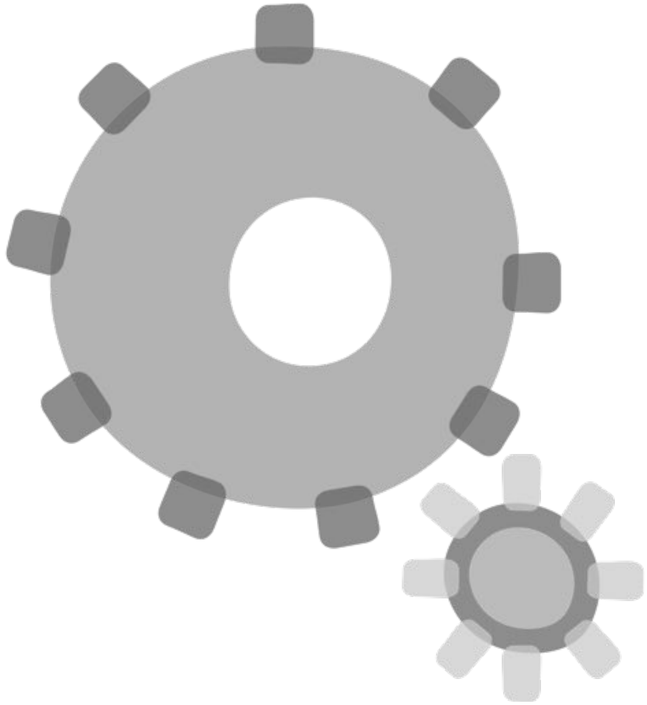
critical or important to application success



critical or important to application success

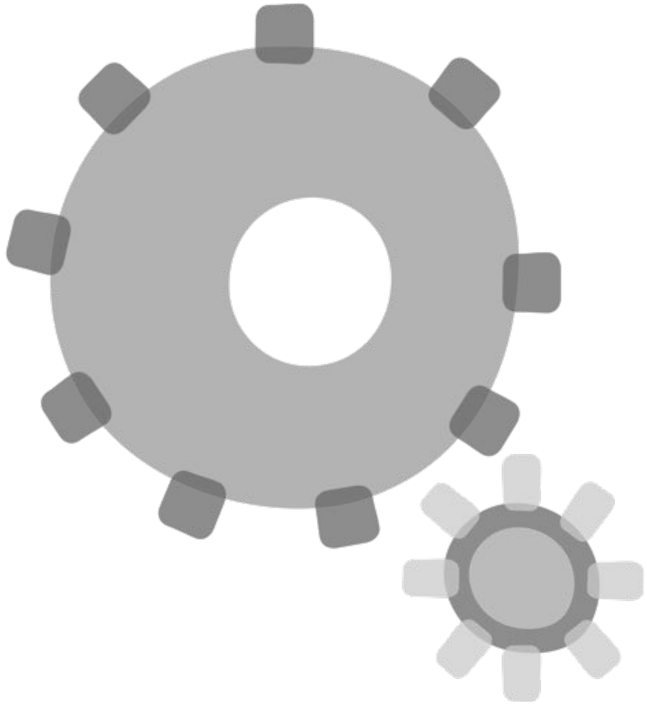

design
specifies a non-domain design consideration
explicit
implicit
influences some structural aspect of the design
architecture
characteristics
critical or important to application success

Operational Architecture Characteristics



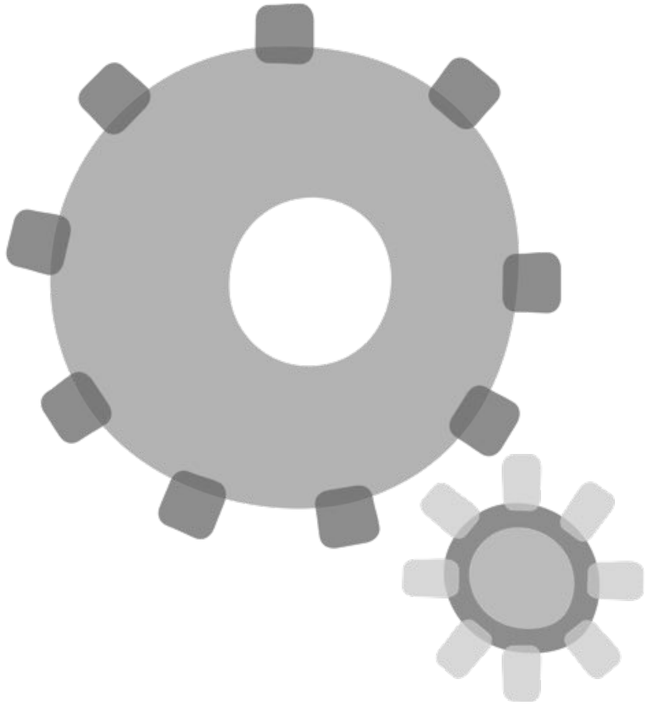
performance

Operational Architecture Characteristics

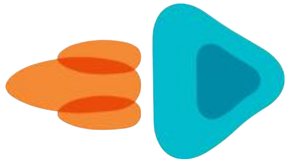


performance

Operational Architecture Characteristics

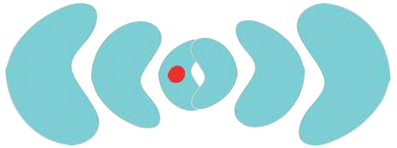


scalability

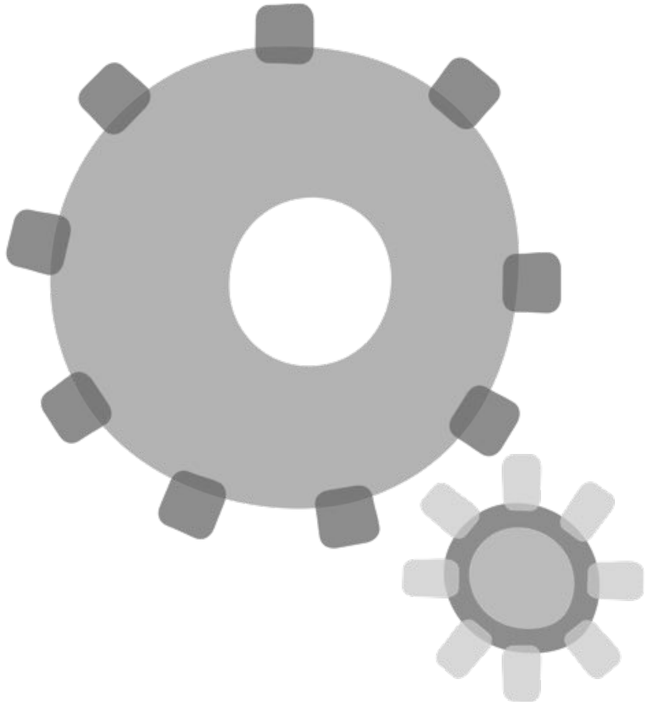


performance

Operational Architecture Characteristics



elasticity



scalability

Quality Architecture Characteristics

reliability

the ability of a system or component to function
under stated conditions for a specified period of time.

reliability ← composite

the ability of a system or component to function
under stated conditions for a specified period of time.

Q: How many ways can engineers measure *reliability*?

availability

$$X(t) = \begin{cases} 1, & \text{sys functions at time } t \\ 0, & \text{otherwise} \end{cases}$$

The availability A(t) at time t > 0 is:

$$A(t) = \Pr[X(t) = 1] = E[X(t)].$$

$$A_c = \frac{1}{c} \int_0^c A(t)\, dt.$$

availability

Limiting average availability:

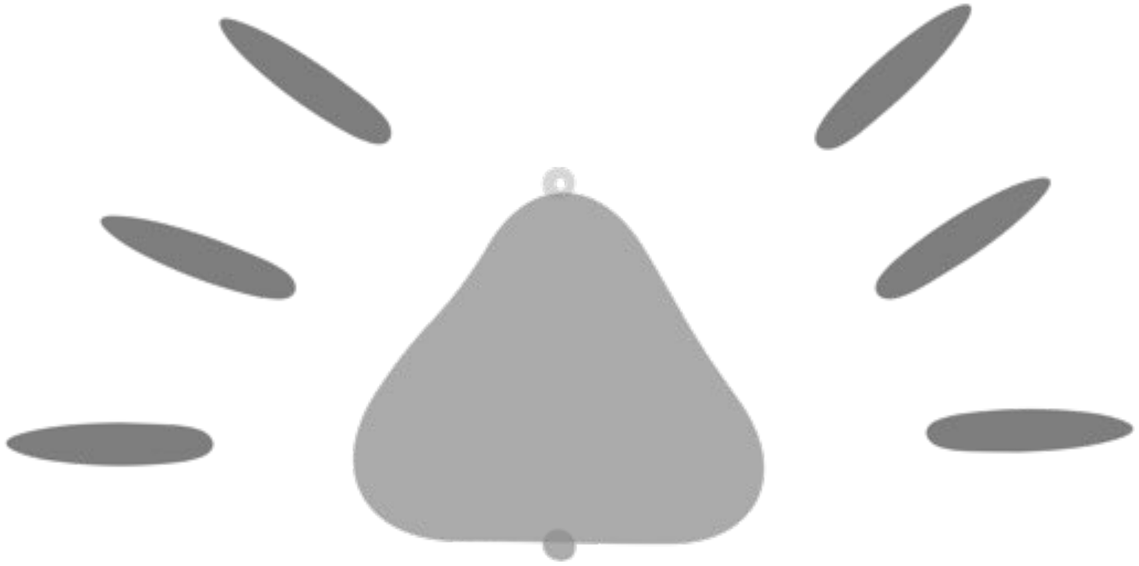
$$A_\infty = \lim_{c\to\infty} A_c = \lim_{c\to\infty} \frac{1}{c}\int_0^c A(t)\,dt, \quad c > 0$$

Q: How do engineers typically measure *availability*?

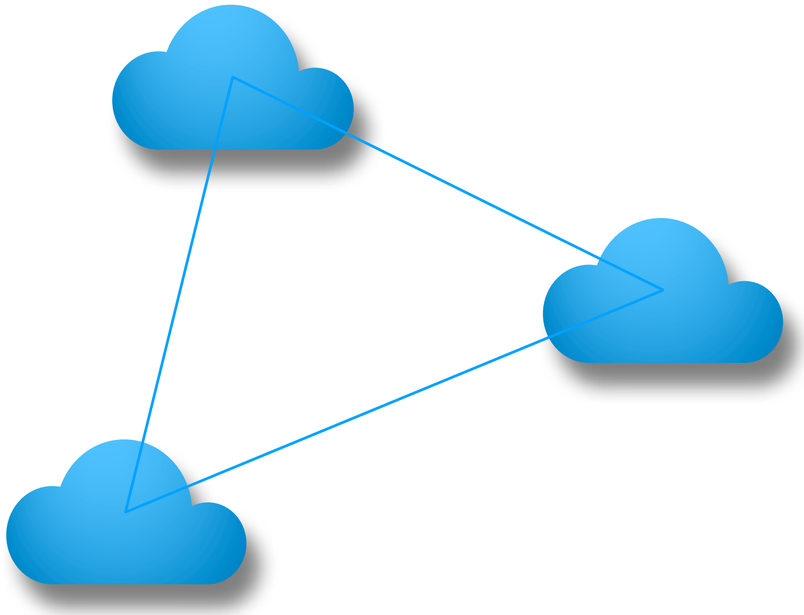
high availability

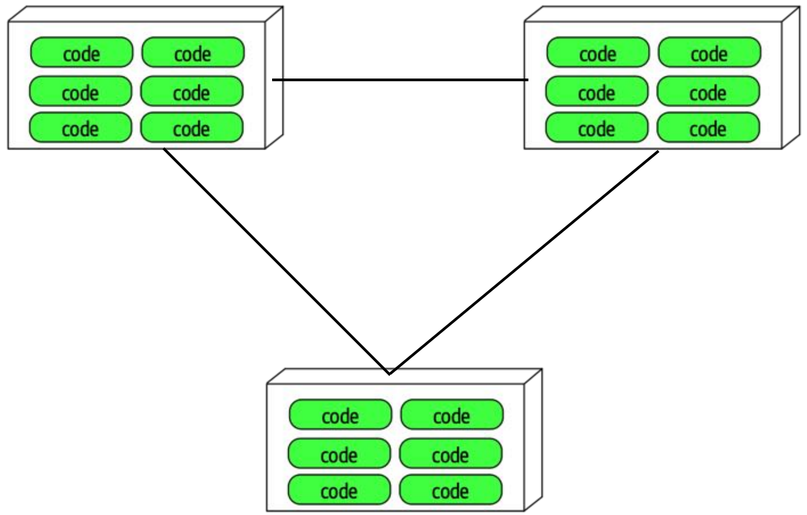
Availability %	Downtime per year[note 1]	Downtime per month	Downtime per week	Downtime per day
55.5555555% ("nine fives")	162.33 days	13.53 days	74.92 hours	10.67 hours
90% ("one nine")	**36.53 days**	**73.05 hours**	**16.80 hours**	**2.40 hours**
95% ("one and a half nines")	18.26 days	36.53 hours	8.40 hours	1.20 hours
97%	10.96 days	21.92 hours	5.04 hours	43.20 minutes
98%	7.31 days	14.61 hours	3.36 hours	28.80 minutes
99% ("two nines")	**3.65 days**	**7.31 hours**	**1.68 hours**	**14.40 minutes**
99.5% ("two and a half nines")	1.83 days	3.65 hours	50.40 minutes	7.20 minutes
99.8%	17.53 hours	87.66 minutes	20.16 minutes	2.88 minutes
99.9% ("three nines")	**8.77 hours**	**43.83 minutes**	**10.08 minutes**	**1.44 minutes**
99.95% ("three and a half nines")	4.38 hours	21.92 minutes	5.04 minutes	43.20 seconds
99.99% ("four nines")	**52.60 minutes**	**4.38 minutes**	**1.01 minutes**	**8.64 seconds**
99.995% ("four and a half nines")	26.30 minutes	2.19 minutes	30.24 seconds	4.32 seconds
99.999% ("five nines")	**5.26 minutes**	**26.30 seconds**	**6.05 seconds**	**864.00 milliseconds**
99.9999% ("six nines")	**31.56 seconds**	**2.63 seconds**	**604.80 milliseconds**	**86.40 milliseconds**
99.99999% ("seven nines")	**3.16 seconds**	**262.98 milliseconds**	**60.48 milliseconds**	**8.64 milliseconds**
99.999999% ("eight nines")	**315.58 milliseconds**	**26.30 milliseconds**	**6.05 milliseconds**	**864.00 microseconds**
99.9999999% ("nine nines")	**31.56 milliseconds**	**2.63 milliseconds**	**604.80 microseconds**	**86.40 microseconds**

Internal Quality Architecture Characteristics



coupling

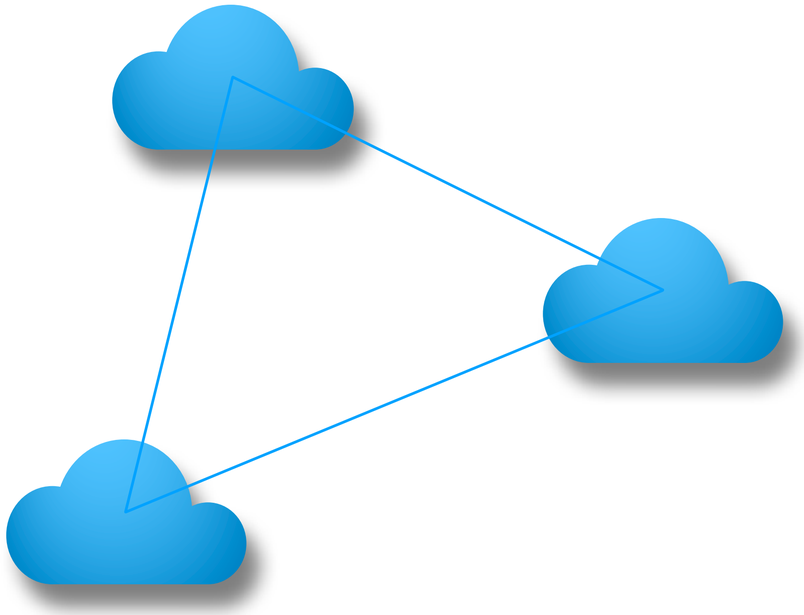


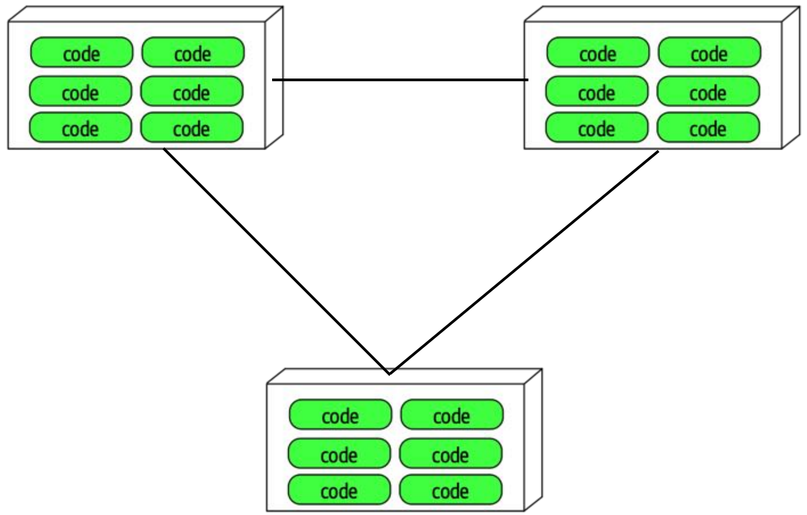


semantic

syntactic

coupling

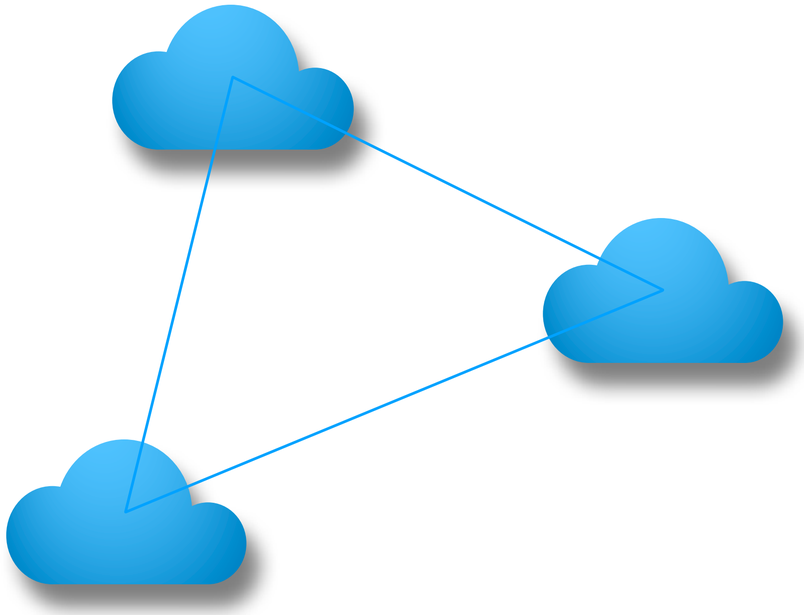


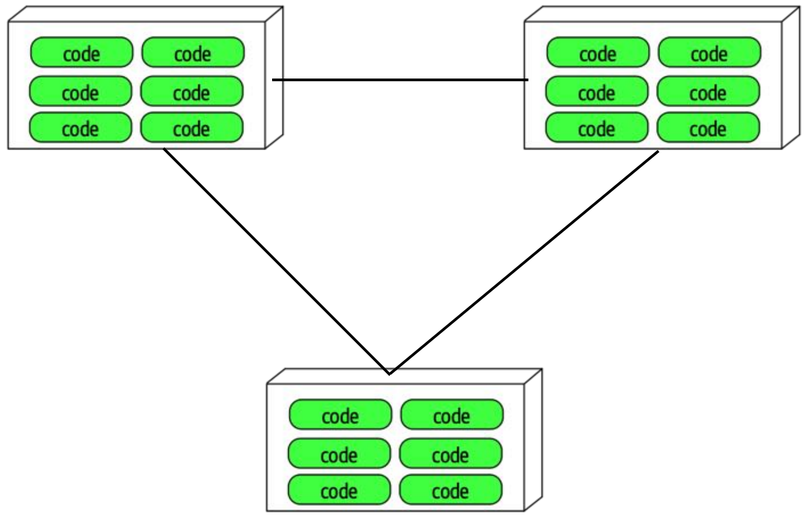


semantic

syntactic

coupling



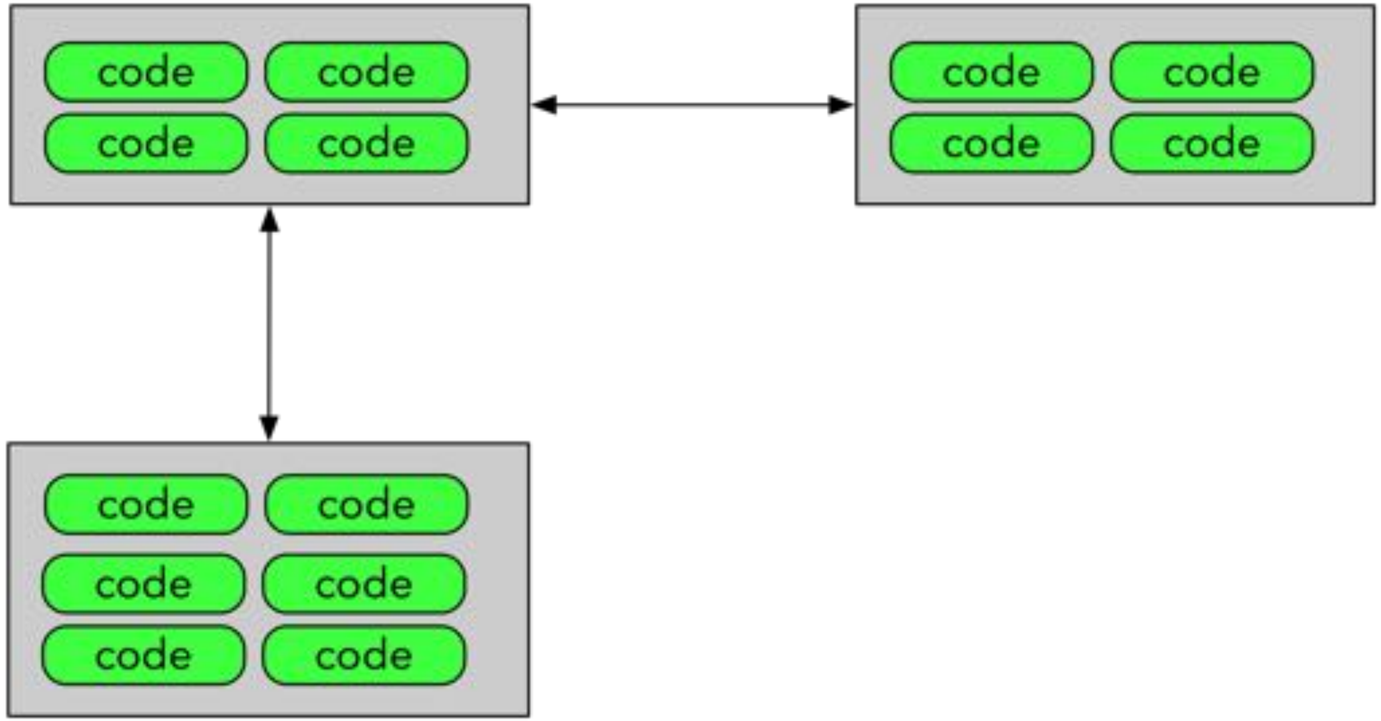


semantic

syntactic

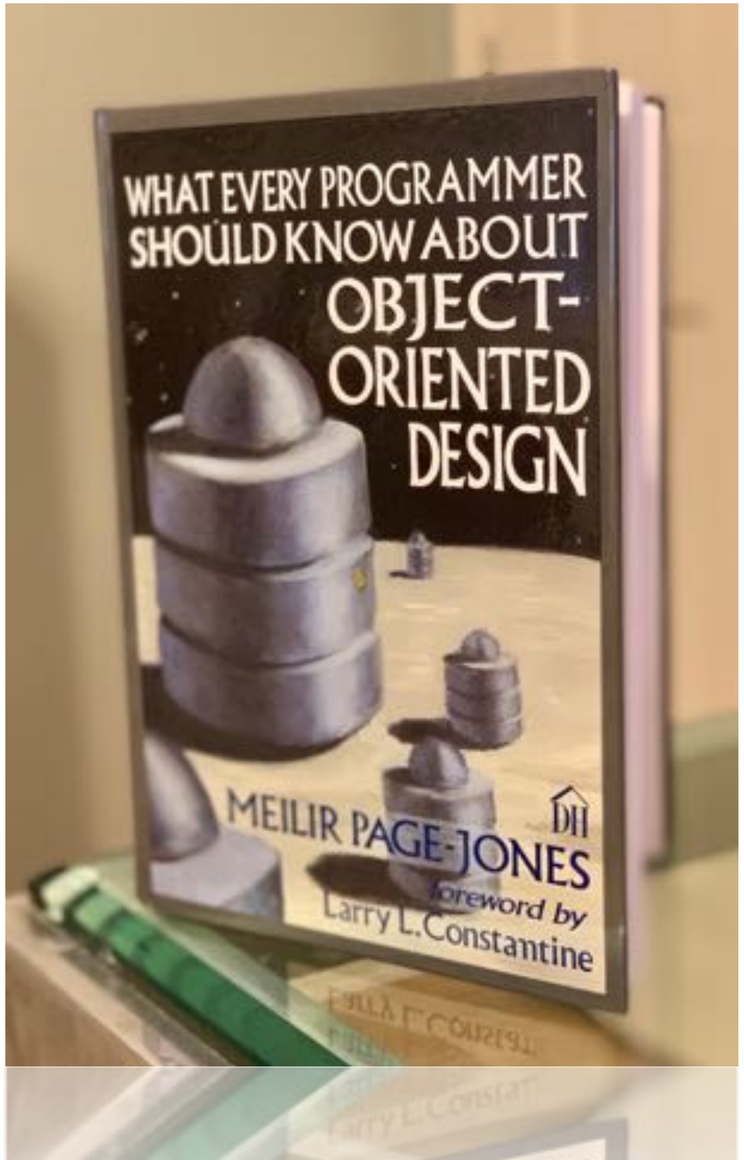
component coupling

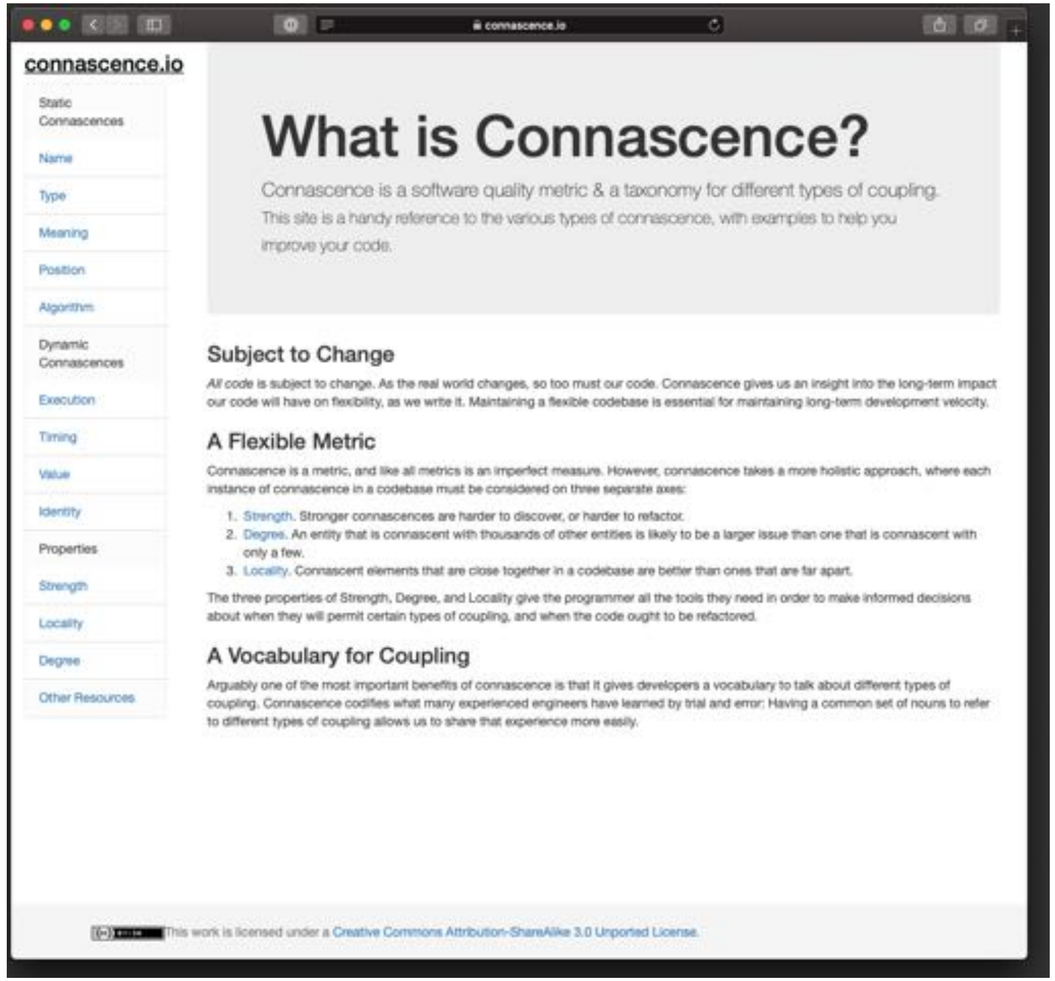
the extent to which components know
about each other



connascence

https://connascence.io/name.html





connascence.io

Static Connascences
- Name
- Type
- Meaning
- Position
- Algorithm

Dynamic Connascences
- Execution
- Timing
- Value
- Identity

Properties
- Strength
- Locality
- Degree

Other Resources

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.

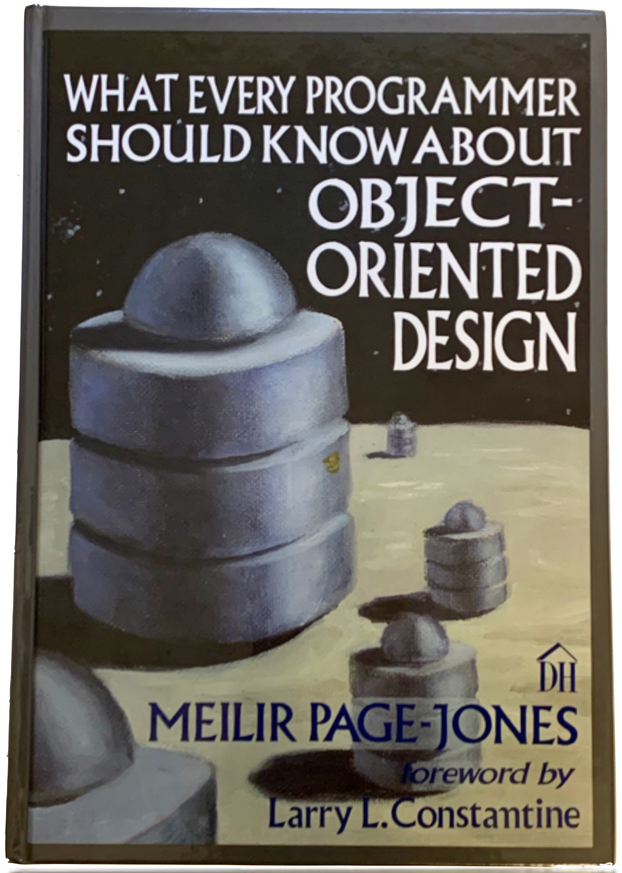
The three properties of Strength, Degree, and Locality give the programmer all the tools they need in order to make informed decisions about when they will permit certain types of coupling, and when the code ought to be refactored.

A Vocabulary for Coupling

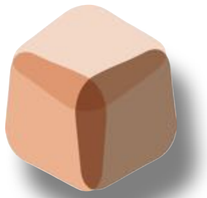
Arguably one of the most important benefits of connascence is that it gives developers a vocabulary to talk about different types of coupling. Connascence codifies what many experienced engineers have learned by trial and error: Having a common set of nouns to refer to different types of coupling allows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

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

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

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

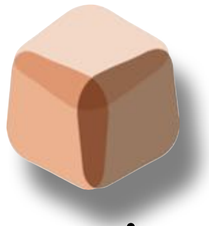
connascence of meaning

multiple components must agree on the meaning of particular values

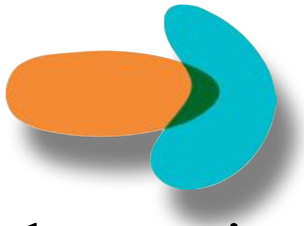
```
def valid_credit_card_number?(cc_number)
  # check for "test" credit card numbers
  if cc_number == "9999-9999-9999-9999"
    # test verification
  else
    # some more code
  end
end
```

How would you fix this ?!?

connascence properties



static

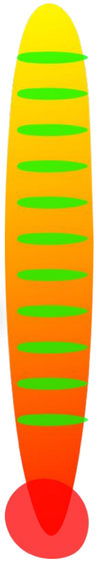


dynamic

Strength

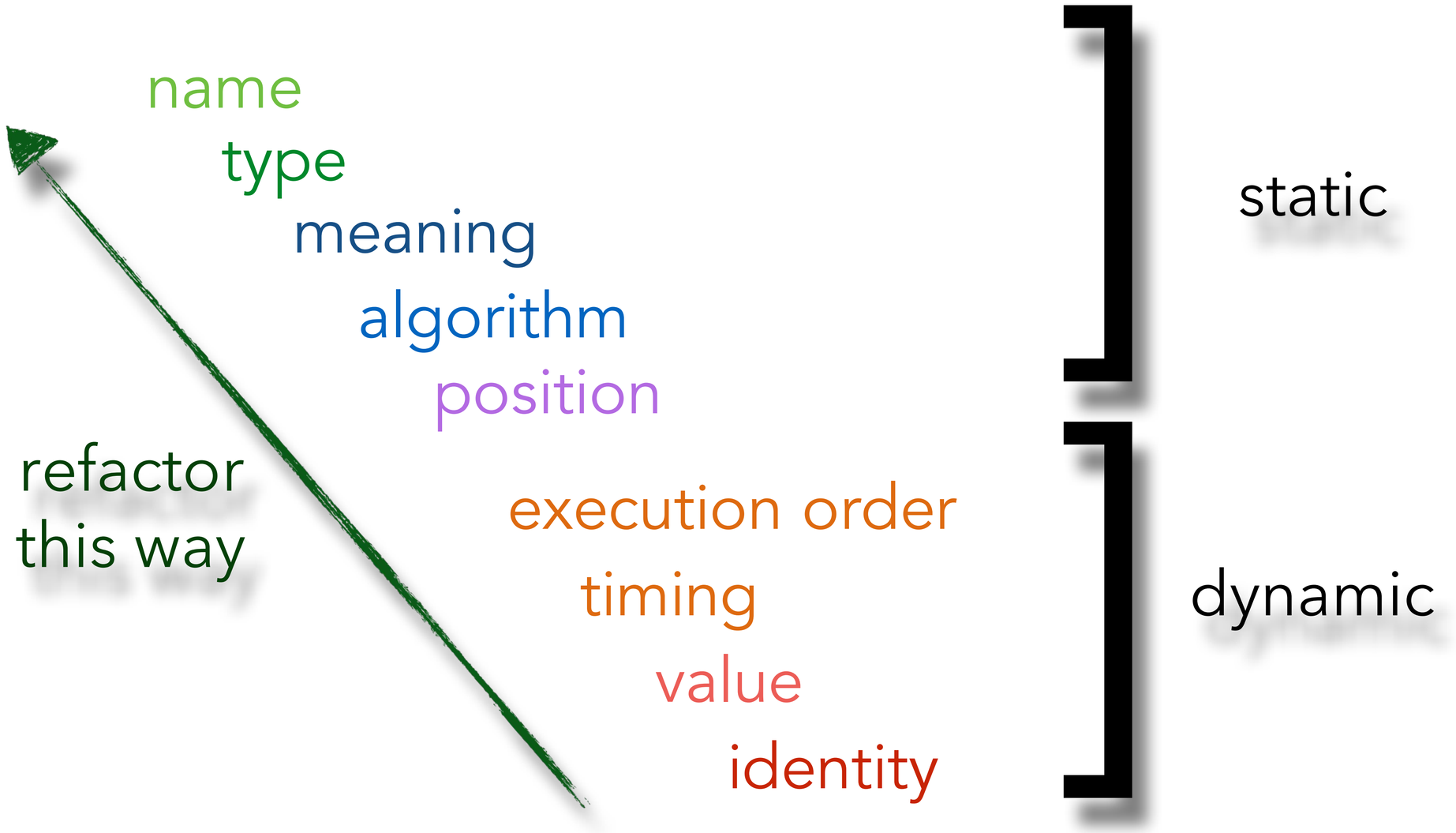
Locality

Degree

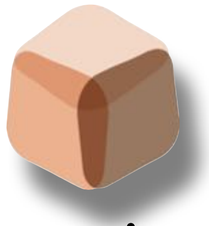


connascence properties

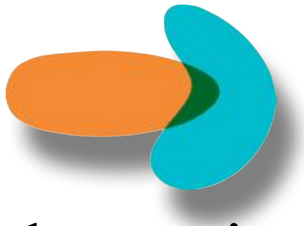
Strength



connascence properties



static

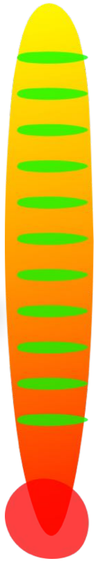


dynamic

Strength

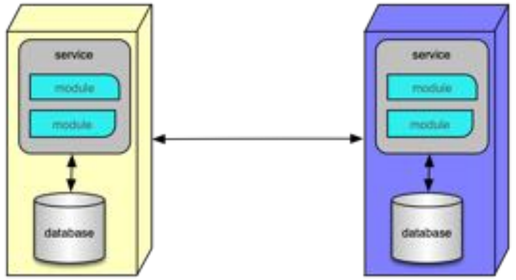
Locality

Degree



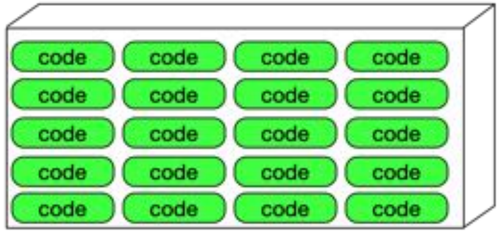
connascence properties

Locality



static

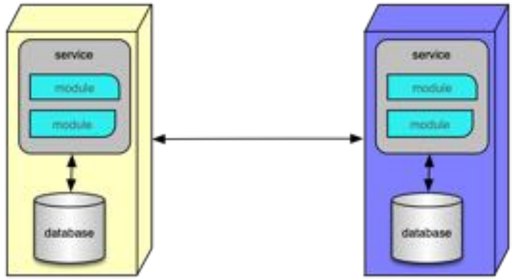
Proximal code (in the same module, class, or function) should have more & higher forms of connascence than less proximal code (in separate modules, or even codebases).

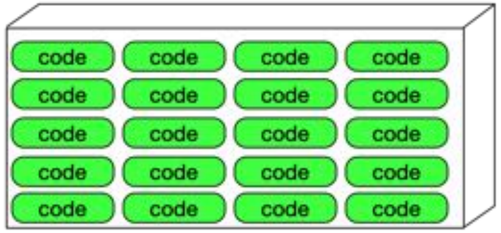


dynamic

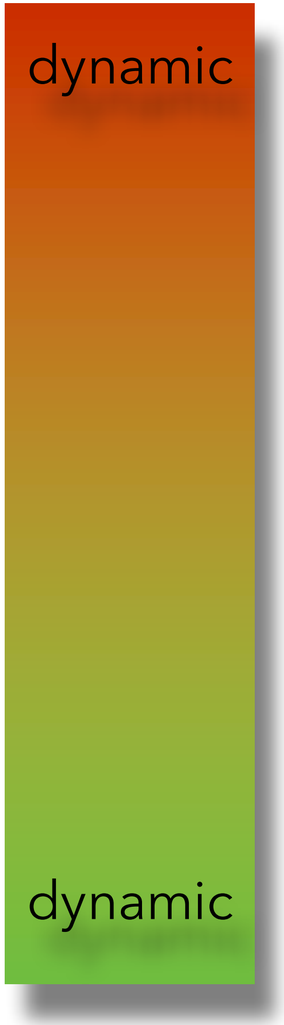
connascence properties

Locality

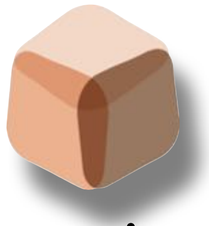




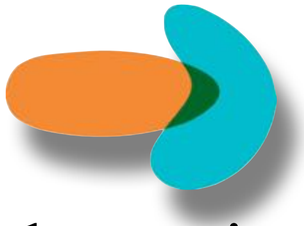
locality matters!



connascence properties



static

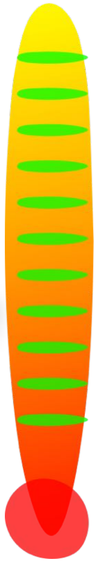


dynamic

Strength

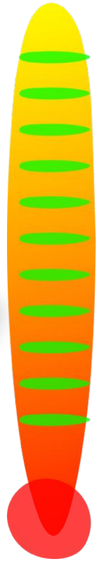
Locality

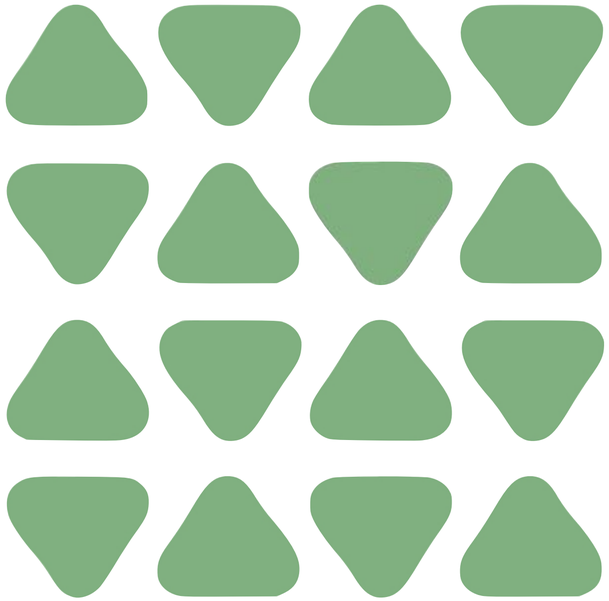
Degree

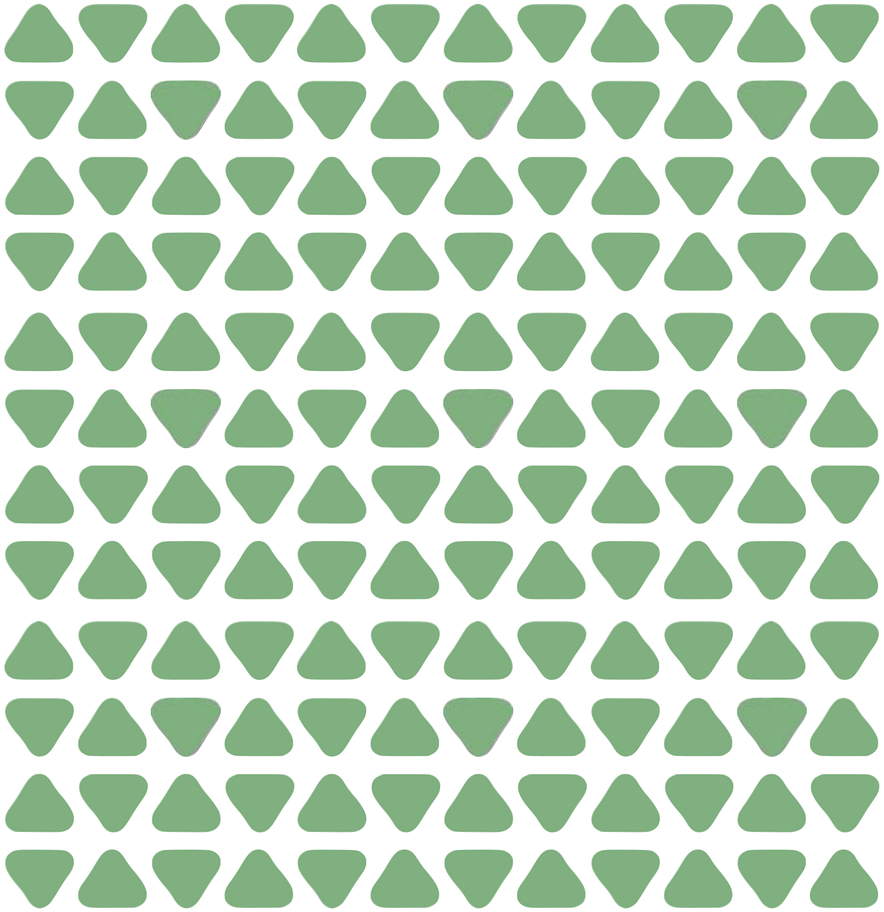


connascence properties

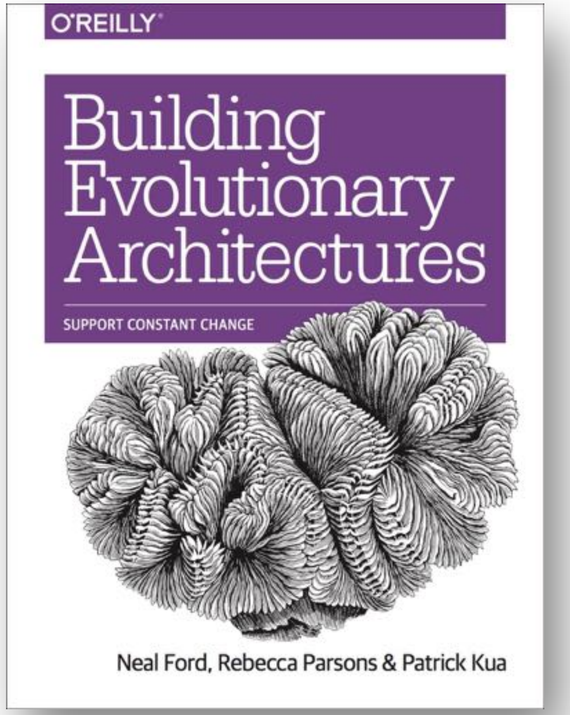
Degree





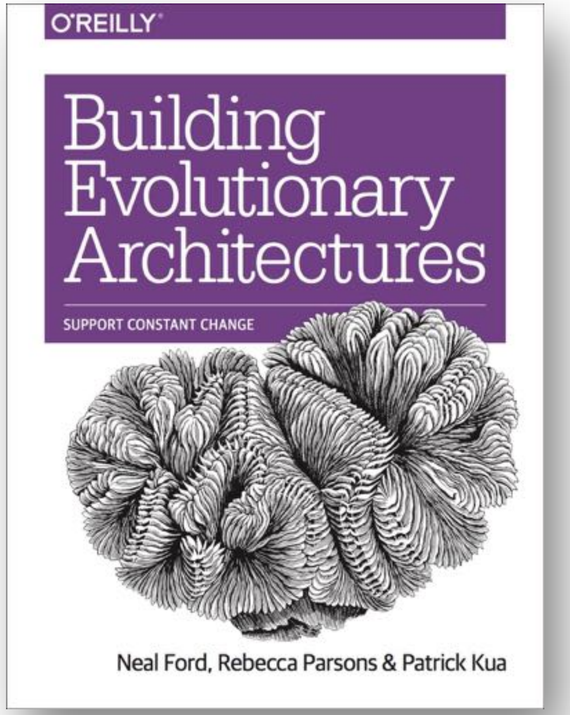


architectural quantum



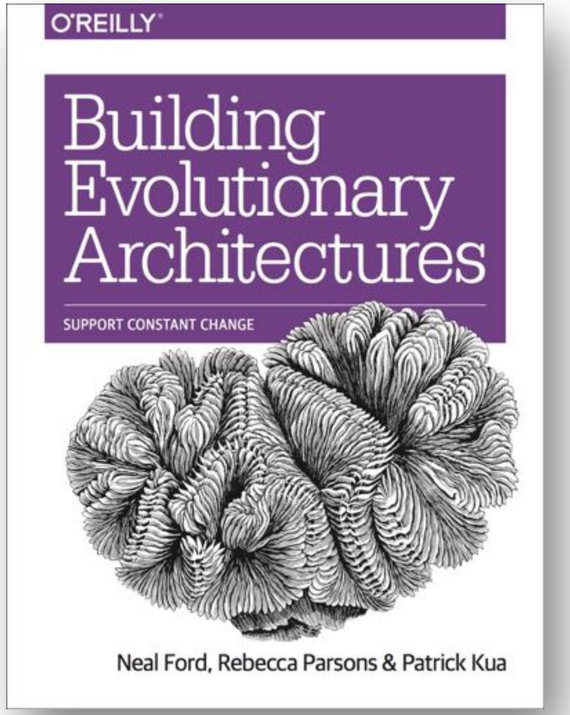
An *architectural quantum* is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

architectural quantum



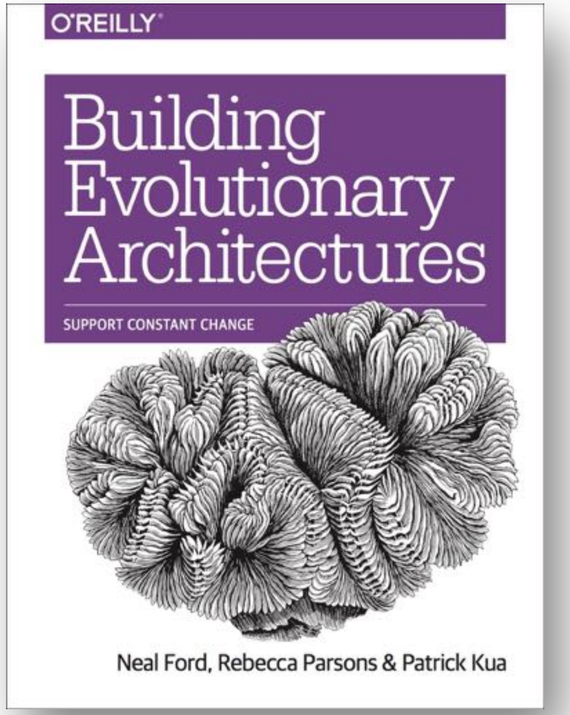
An *architectural quantum* is an <u>*independently deployable component*</u> with high functional cohesion and synchronous dynamic quantum connascence.

architectural quantum



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

architectural quantum



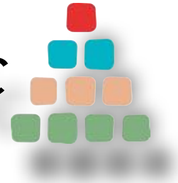
An *architectural quantum* is an independently deployable component with high functional cohesion and *synchronous dynamic quantum connascence*.

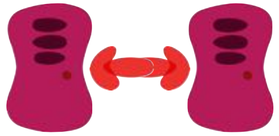
the problems w/ 90's connasence

- still good for static code analysis
- increasingly invalid axiom for dynamic communication
- too low level for architecture

quantum connascence

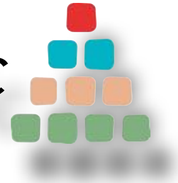
quantum connascence

static 

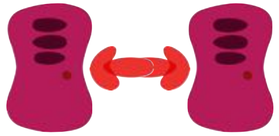
dynamic 

quantum connascence

static



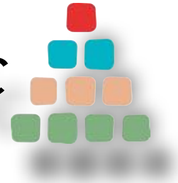
dynamic

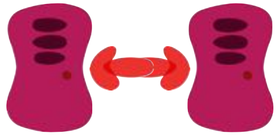


contracts

tight *loose*

quantum connascence

static 

dynamic 

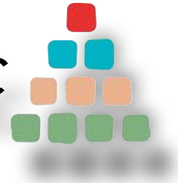
synchronous *asynchronous*

contracts

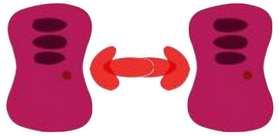
tight *loose*

quantum connascence

static



quantum



synchronous → *asynchronous* ⇢

how quanta communicate

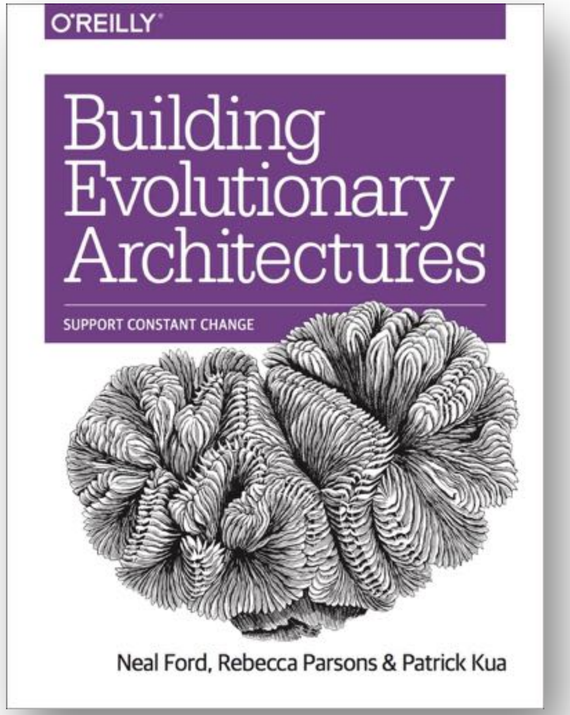
contracts

tight *loose*

impact on operational
(and other) architecture characteristics

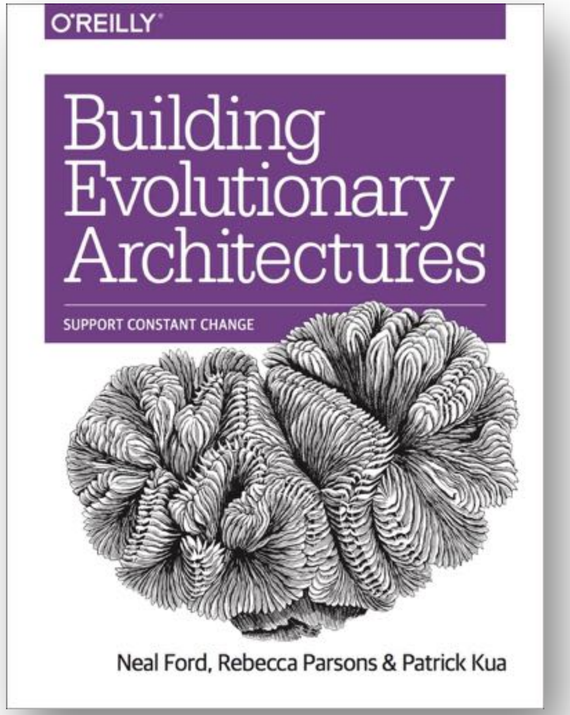
useful for hybrid architecture design,
architecture migration, integration, etc.

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

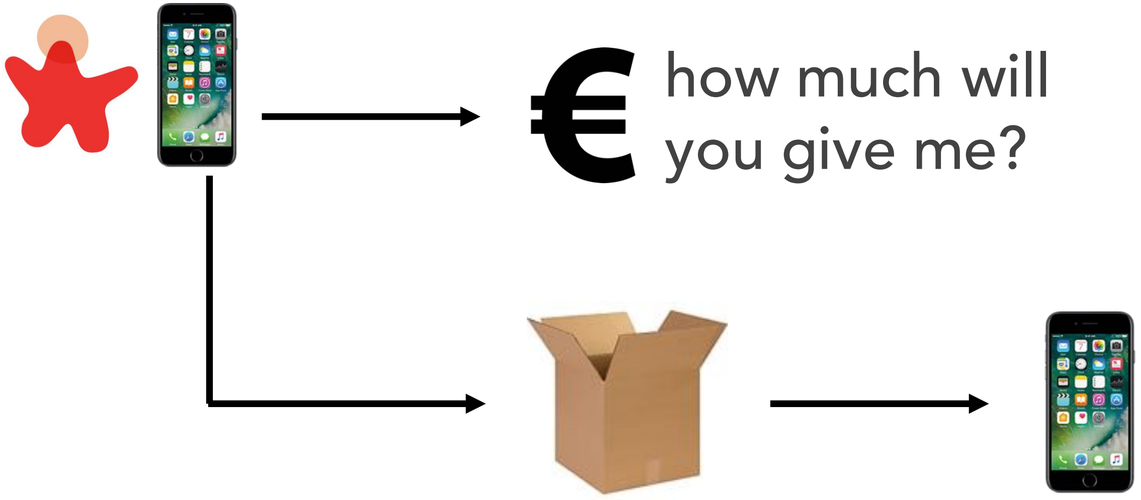
Users: Hundreds, hopefully thousands to millions

architectural quantum

electronics recycling

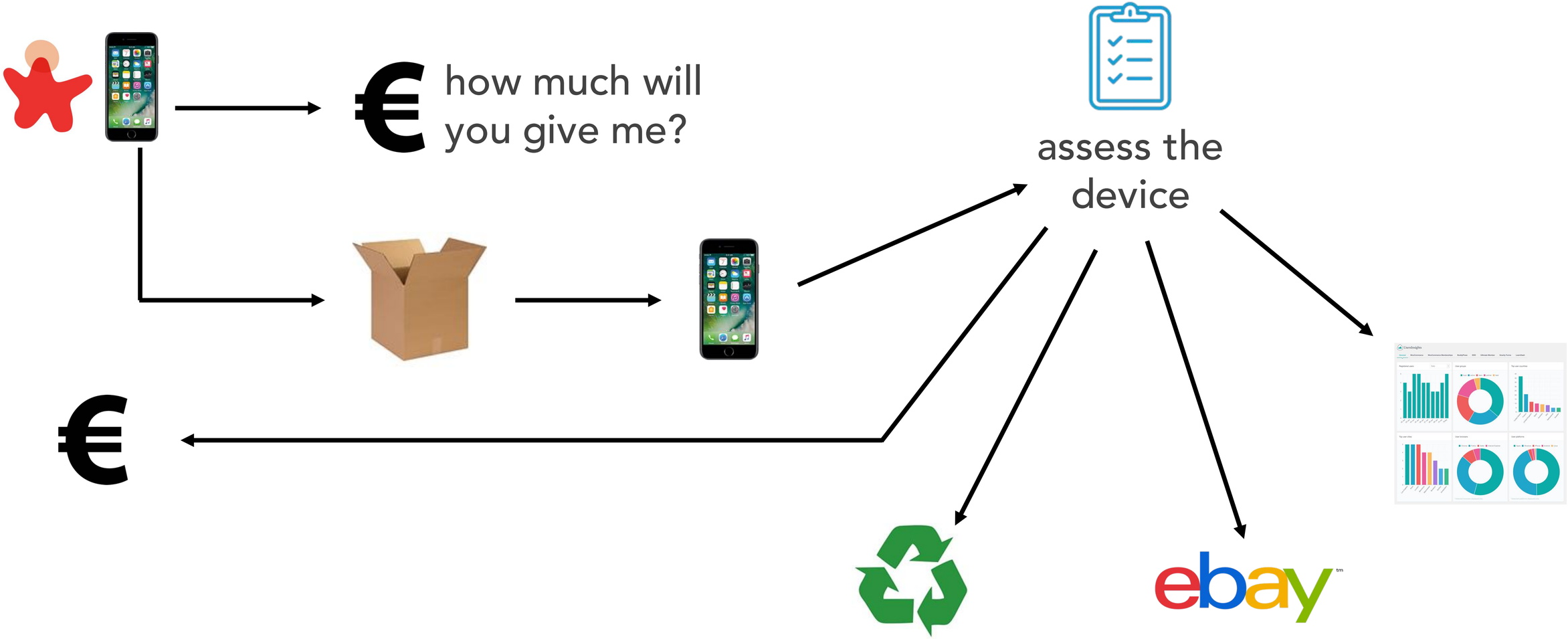
architectural quantum

electronics recycling



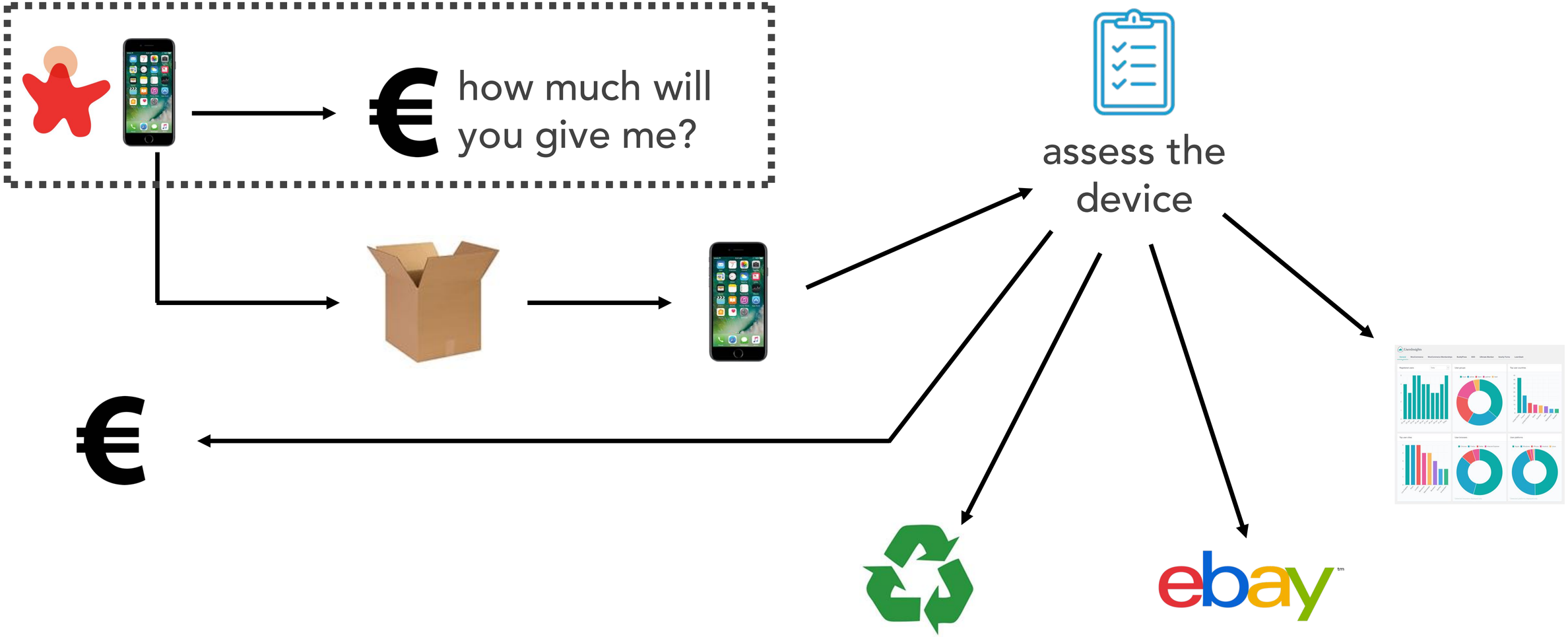
architectural quantum

electronics recycling



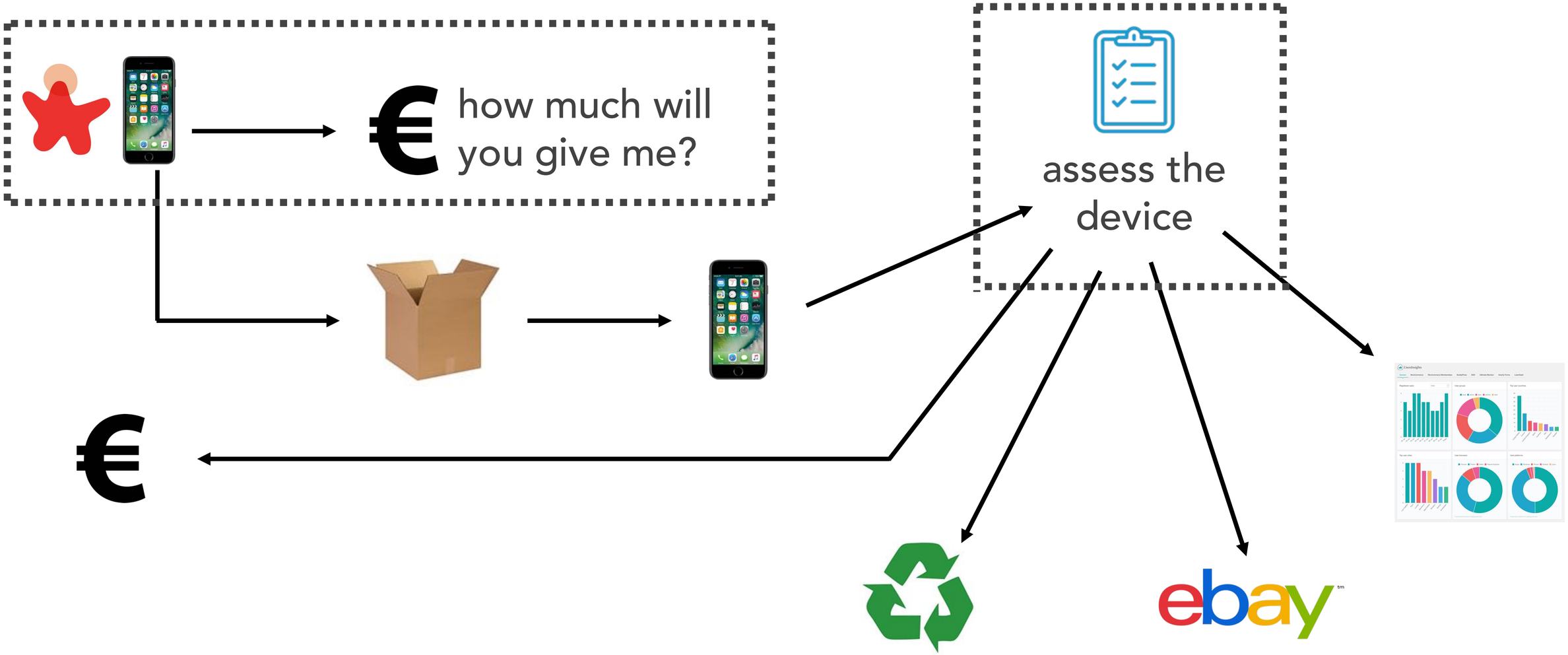
architectural quantum

electronics recycling



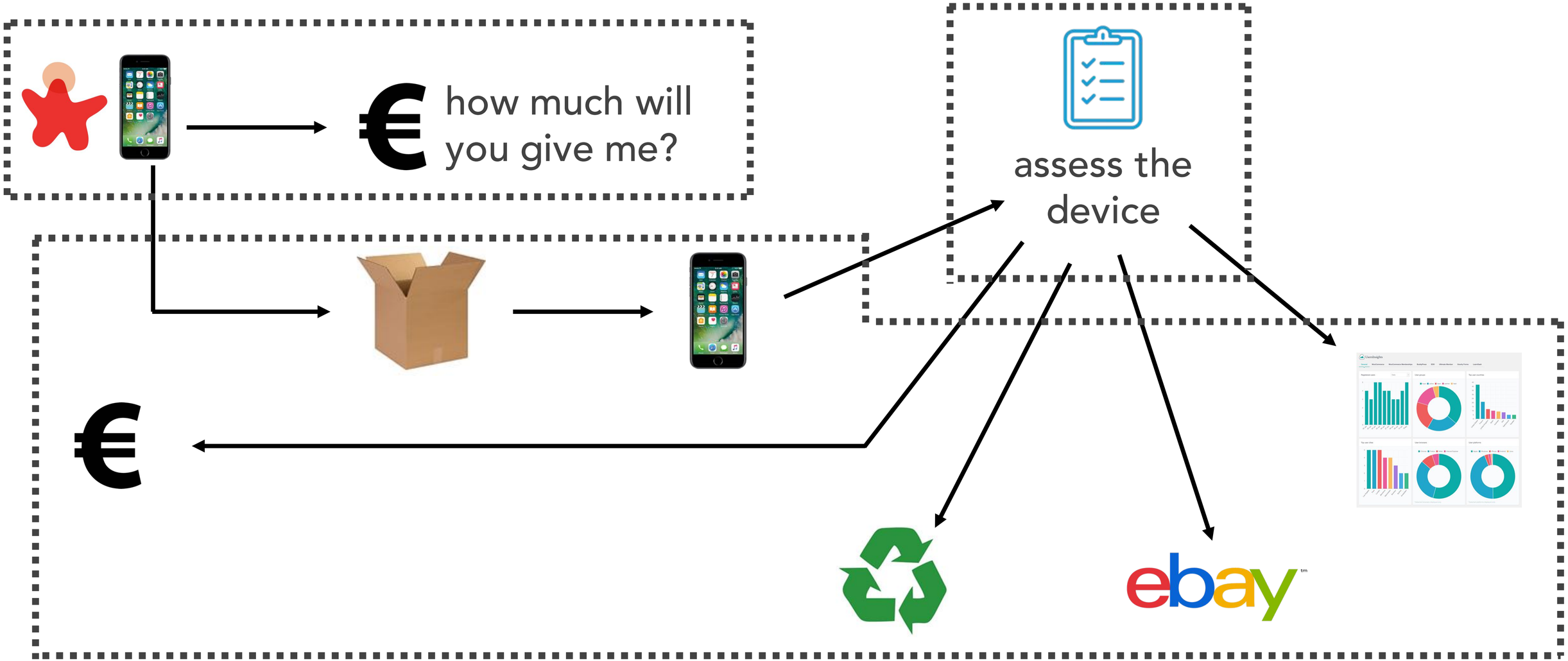
architectural quantum

electronics recycling



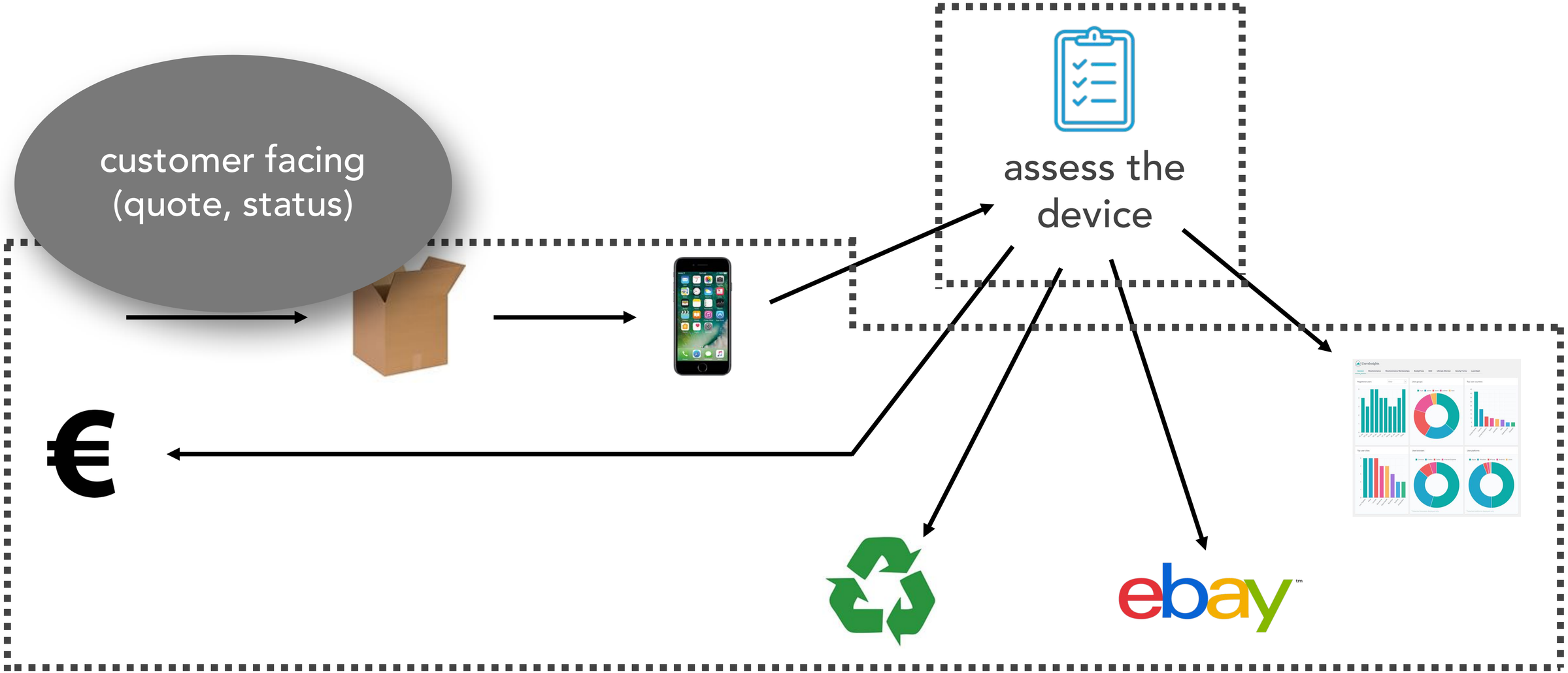
architectural quantum

electronics recycling



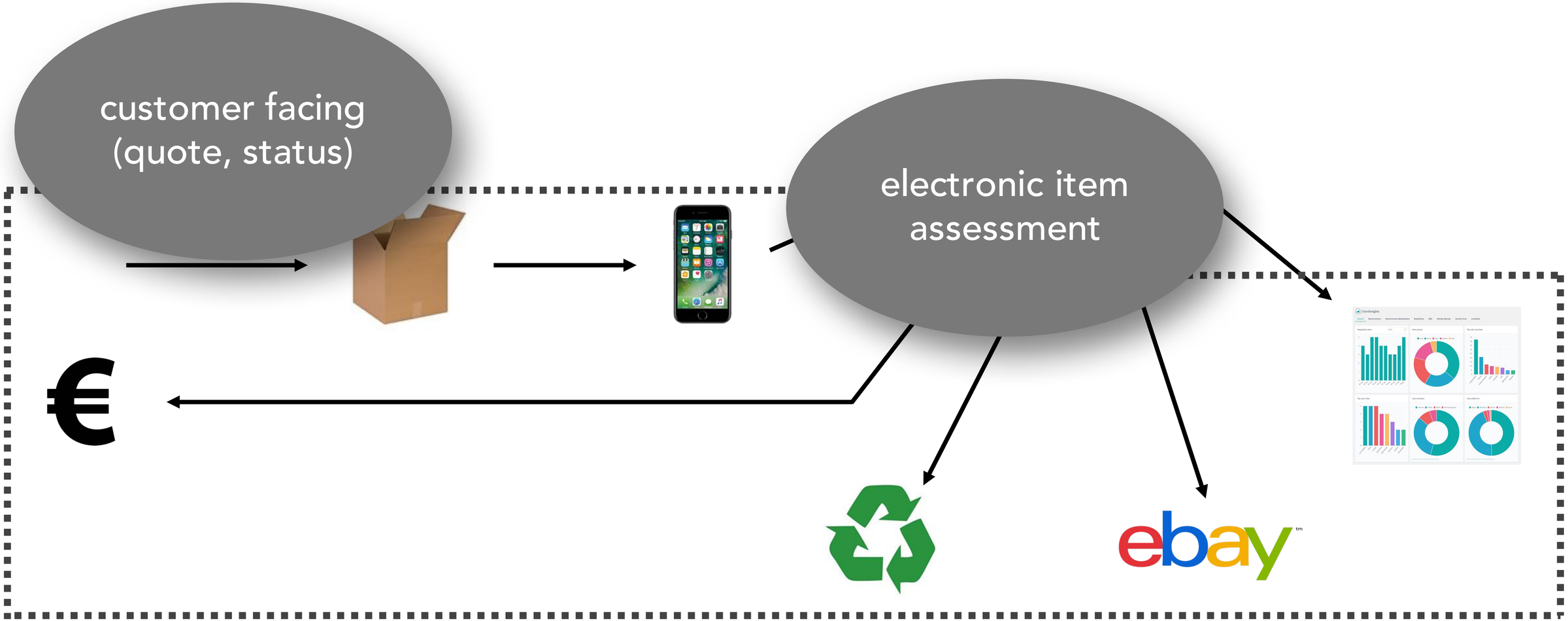
architectural quantum

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

recycling, reporting,
and accounting

security
data integrity
auditability

Your Architectural Kata is...

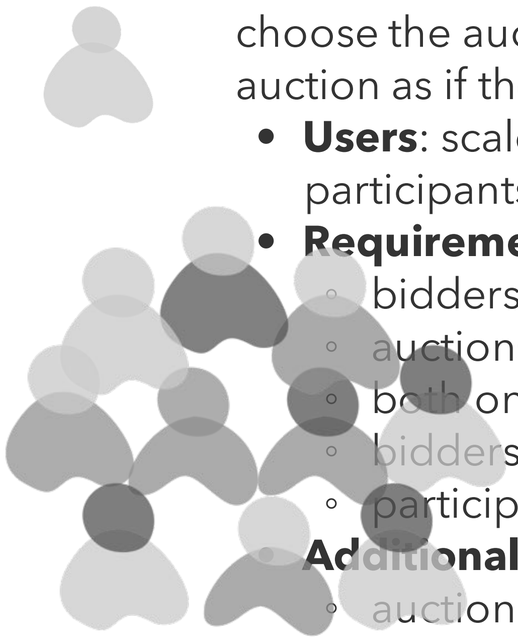
Going Going Gone!

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

Your Architectural Kata is...

Going Going Gone!

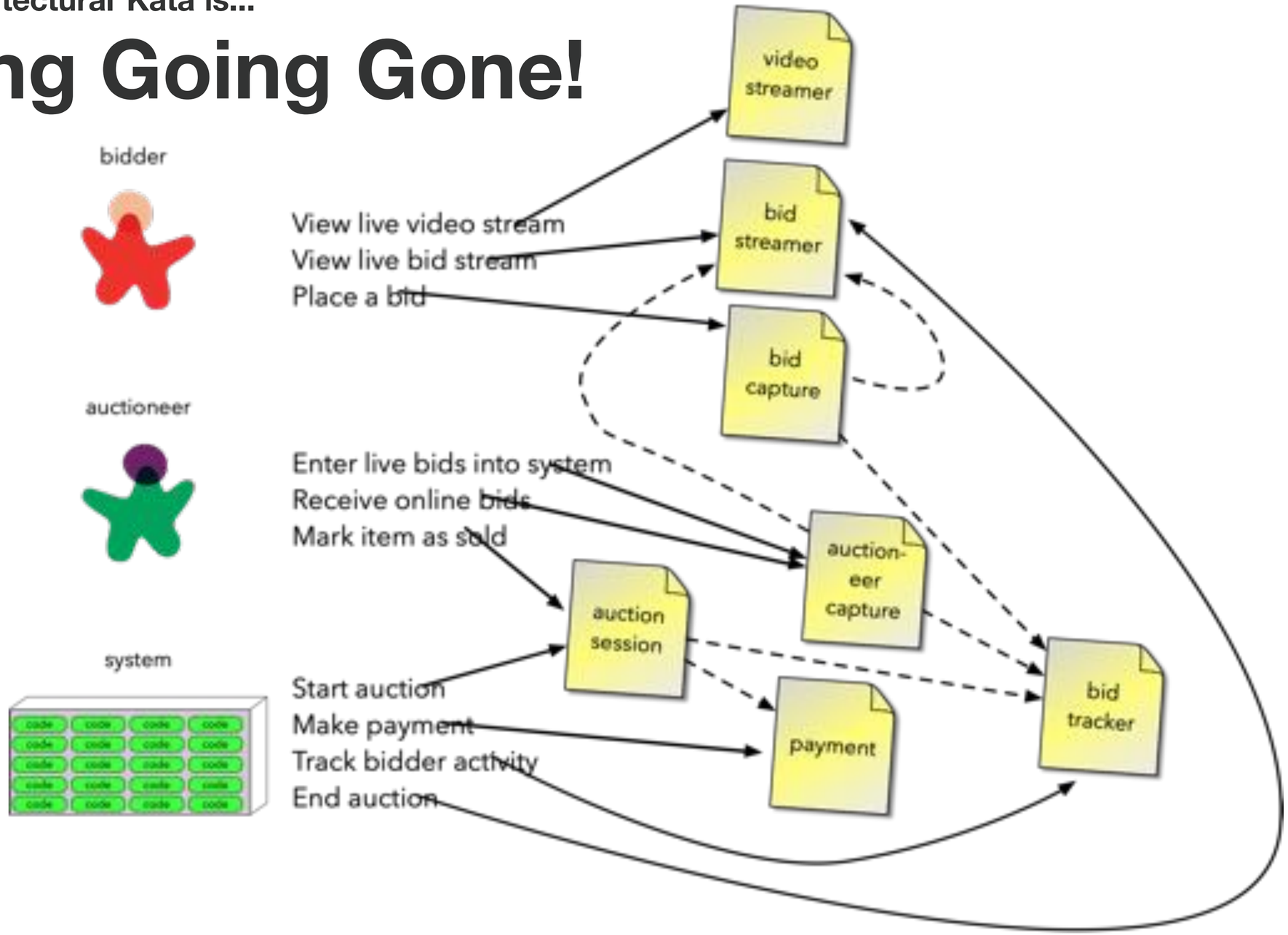


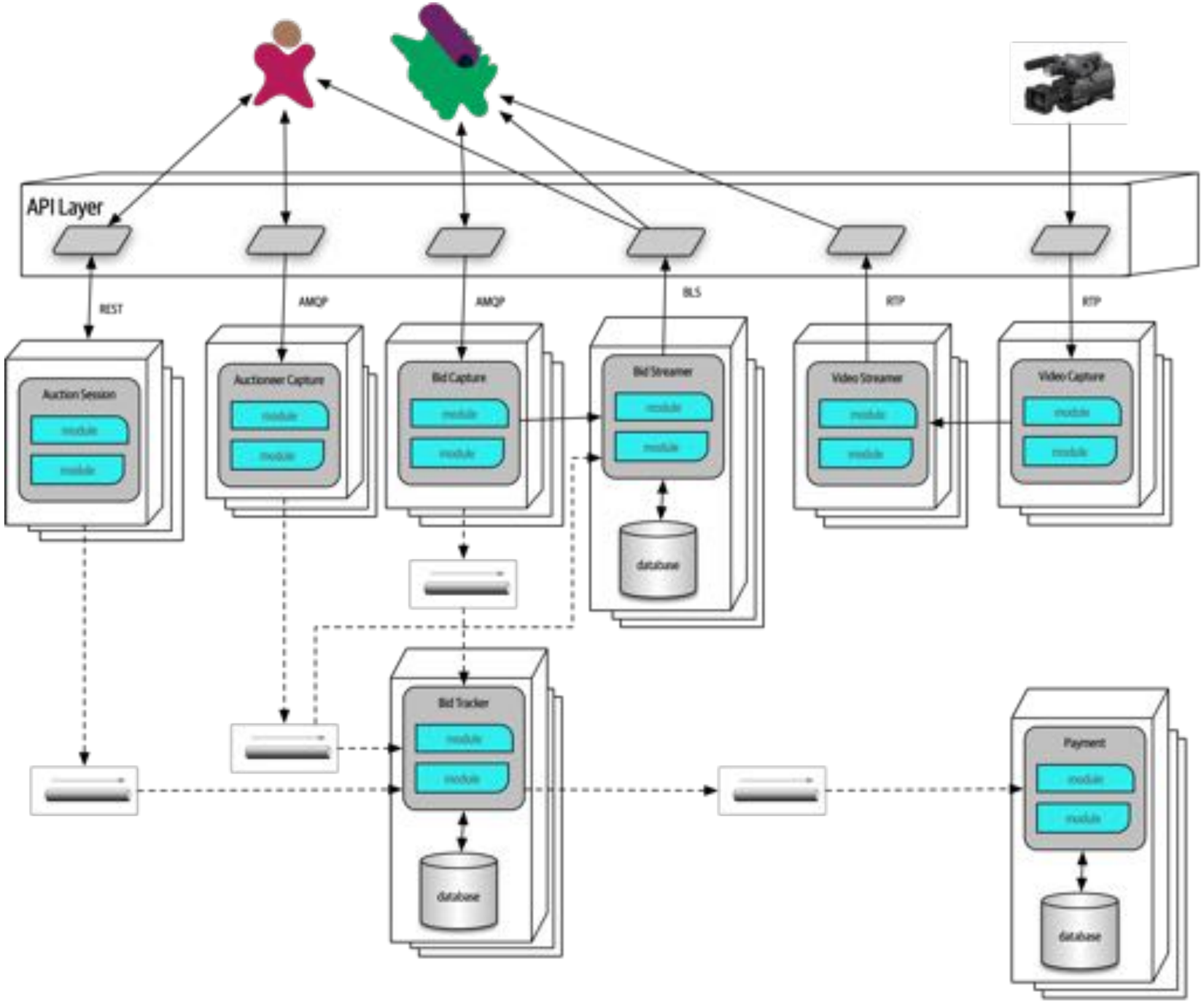
An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

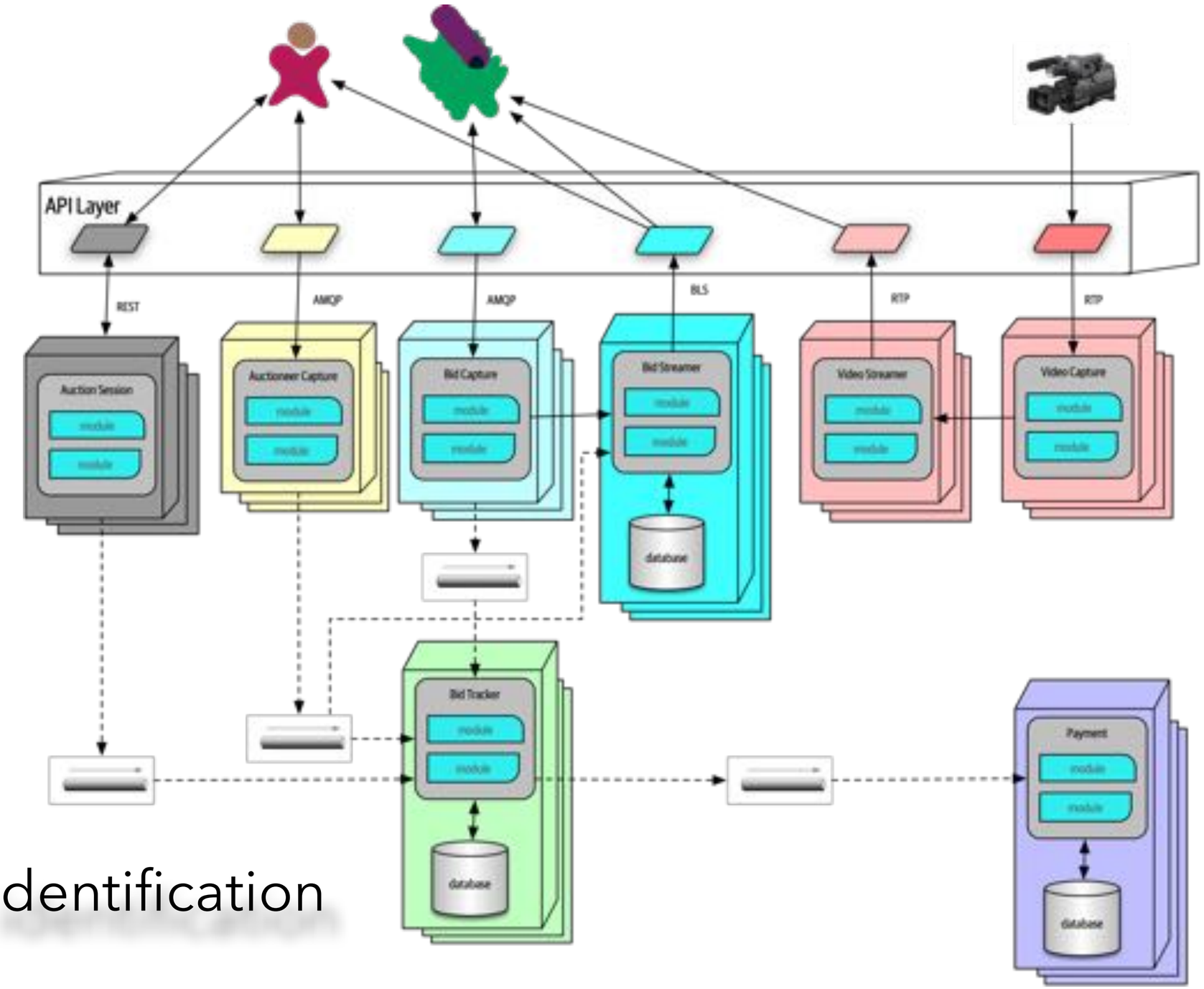
- **Users**: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - auctions must be as real-time as possible
  - 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

Your Architectural Kata is...

Going Going Gone!



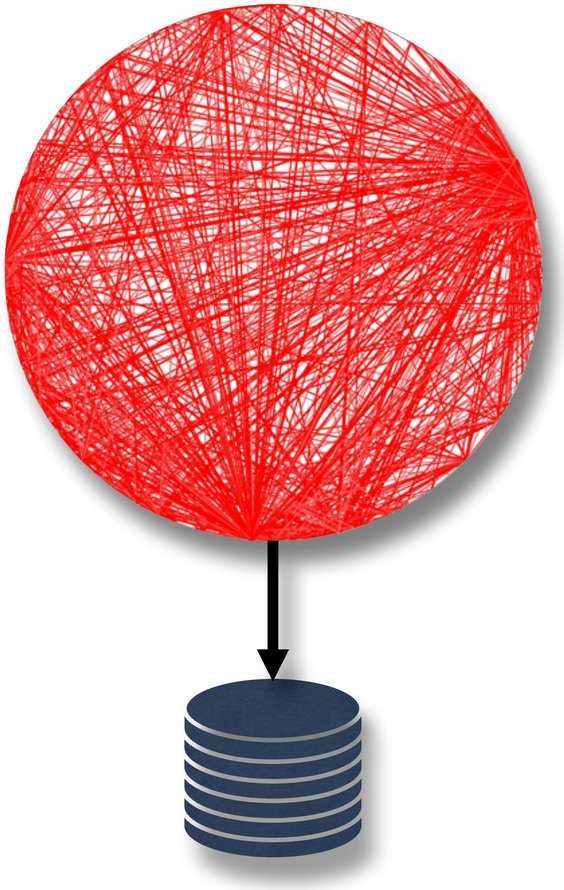

API Layer
REST
AMQP
AMQP
BLS
RTP
RTP
Auction Session
module
module
Auctioneer Capture
module
module
Bid Capture
module
module
Bid Streamer
module
module
database
Video Streamer
module
module
Video Capture
module
module
Bid Tracker
module
module
database
Payment
module
module
database



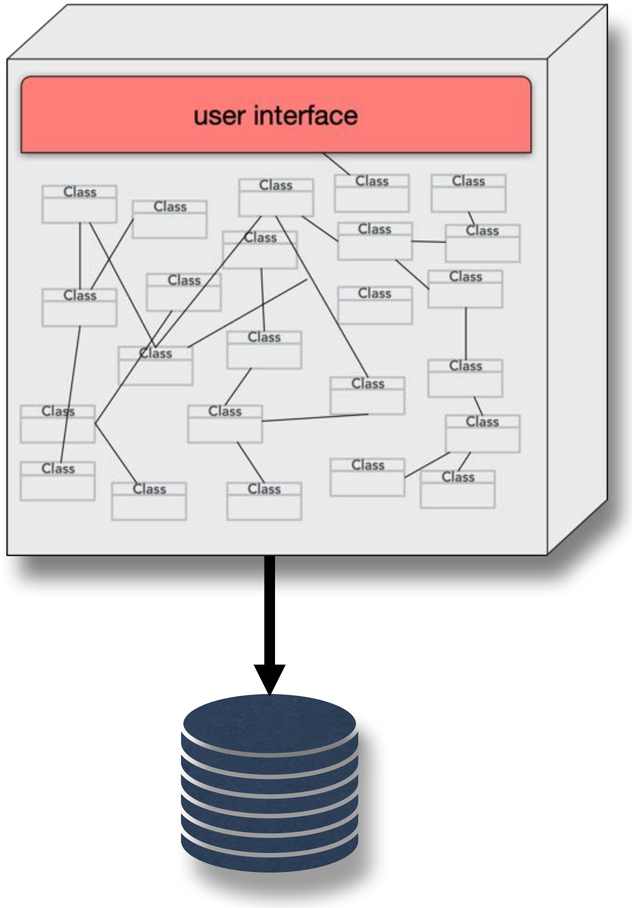
quanta identification

Domain Partitioning

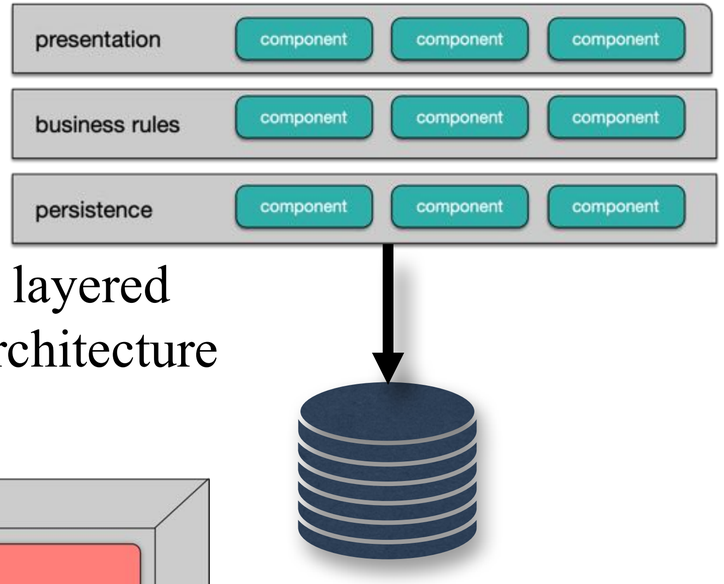
monoliths



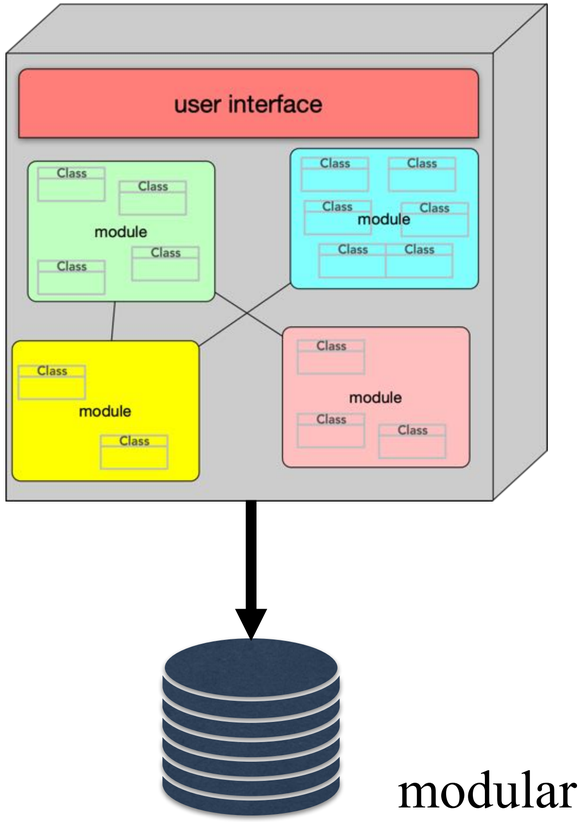
“Big Ball of Mud”
https://en.wikipedia.org/wiki/Big_ball_of_mud



unstructured monolith



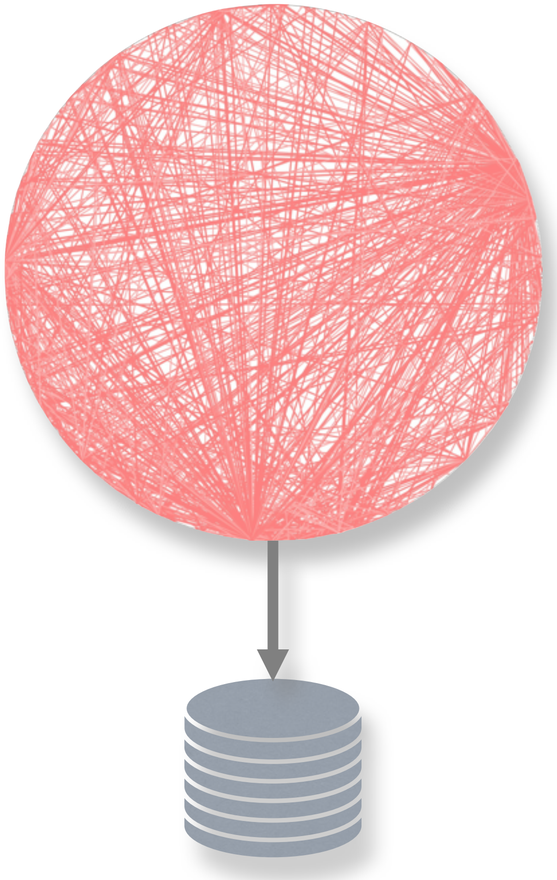
layered architecture



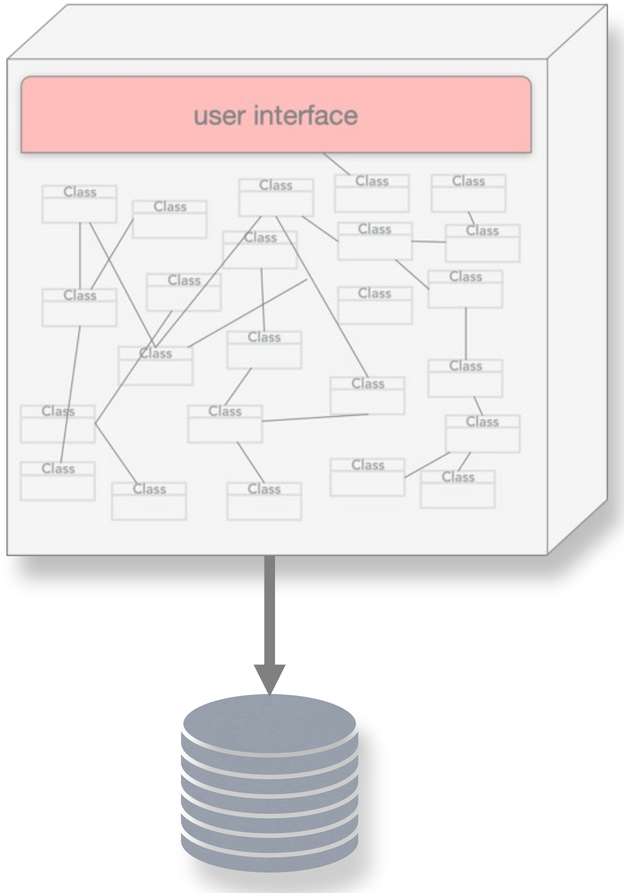
modular monolith

http://www.codingthearchitecture.com/presentations/sa2015-modular-monoliths

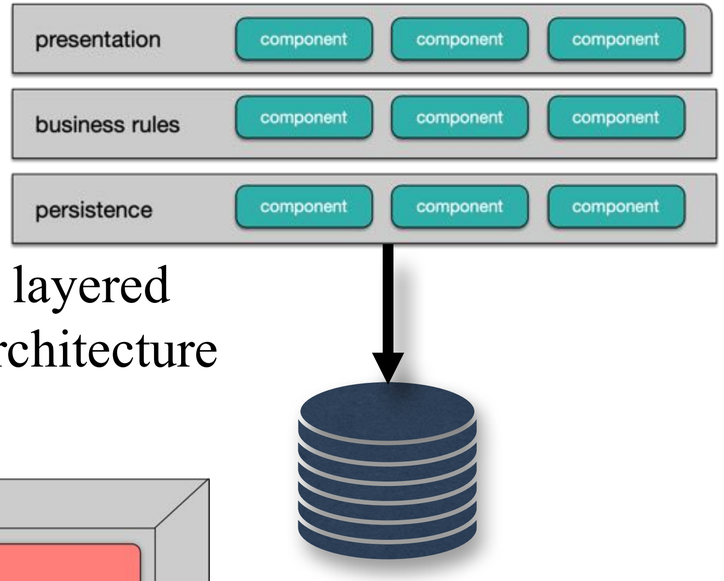
monoliths



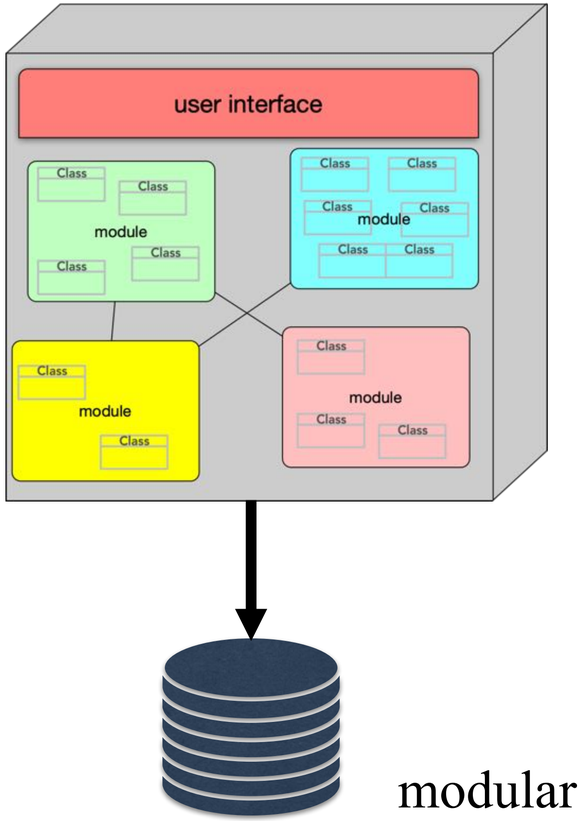
"Big Ball of Mud"
https://en.wikipedia.org/wiki/Big_ball_of_mud



unstructured monolith



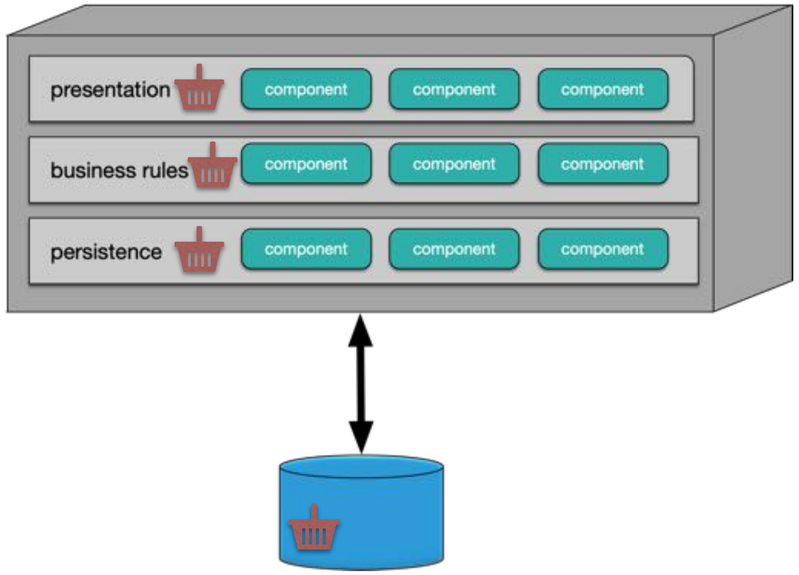
layered architecture



modular monolith
http://www.codingthearchitecture.com/presentations/sa2015-modular-monoliths

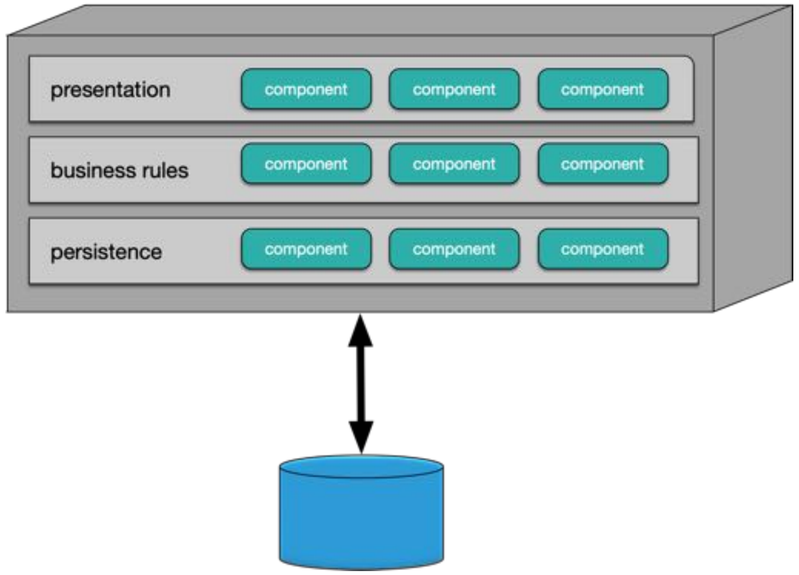
top level partitioning

technical partitioning

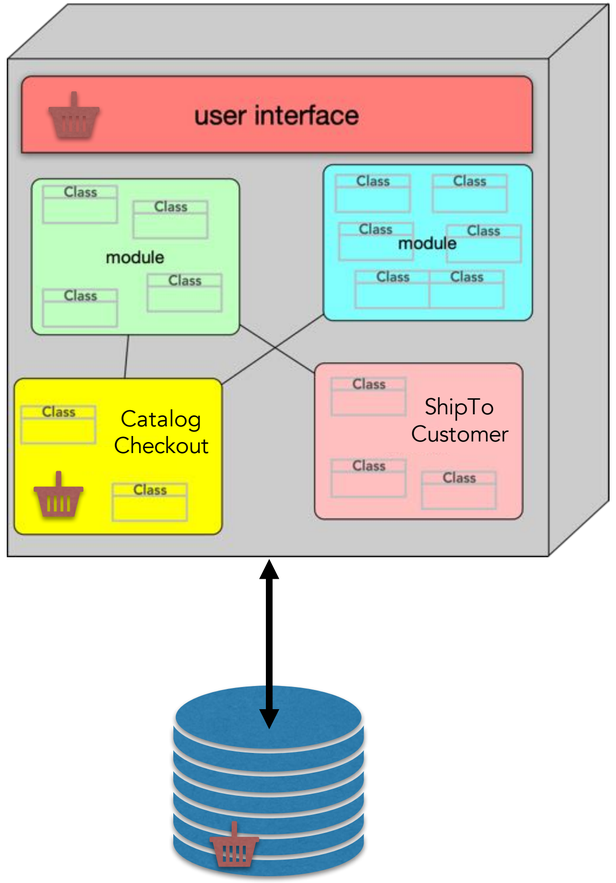


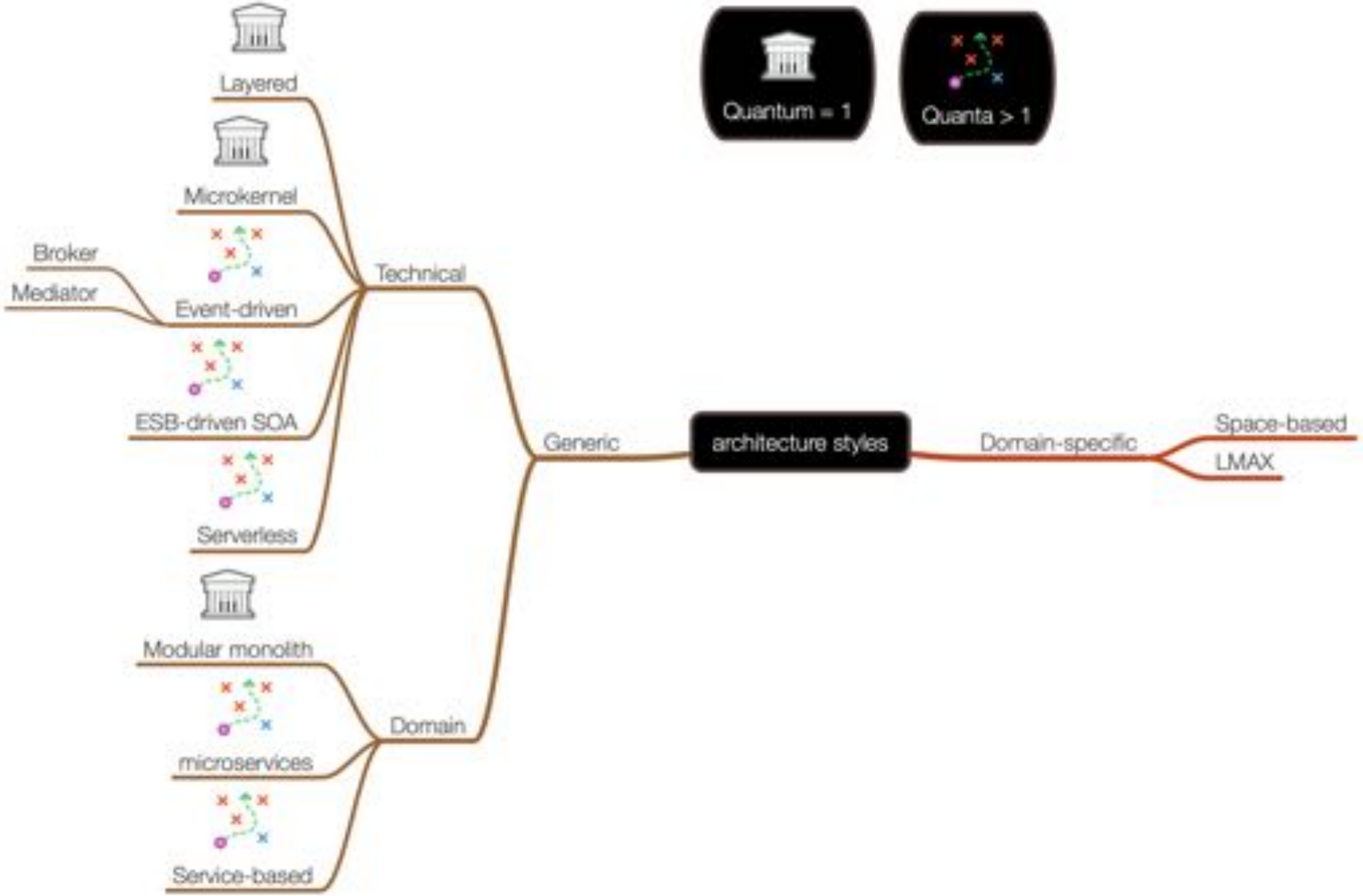
top level partitioning

technical partitioning



domain partitioning




Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX

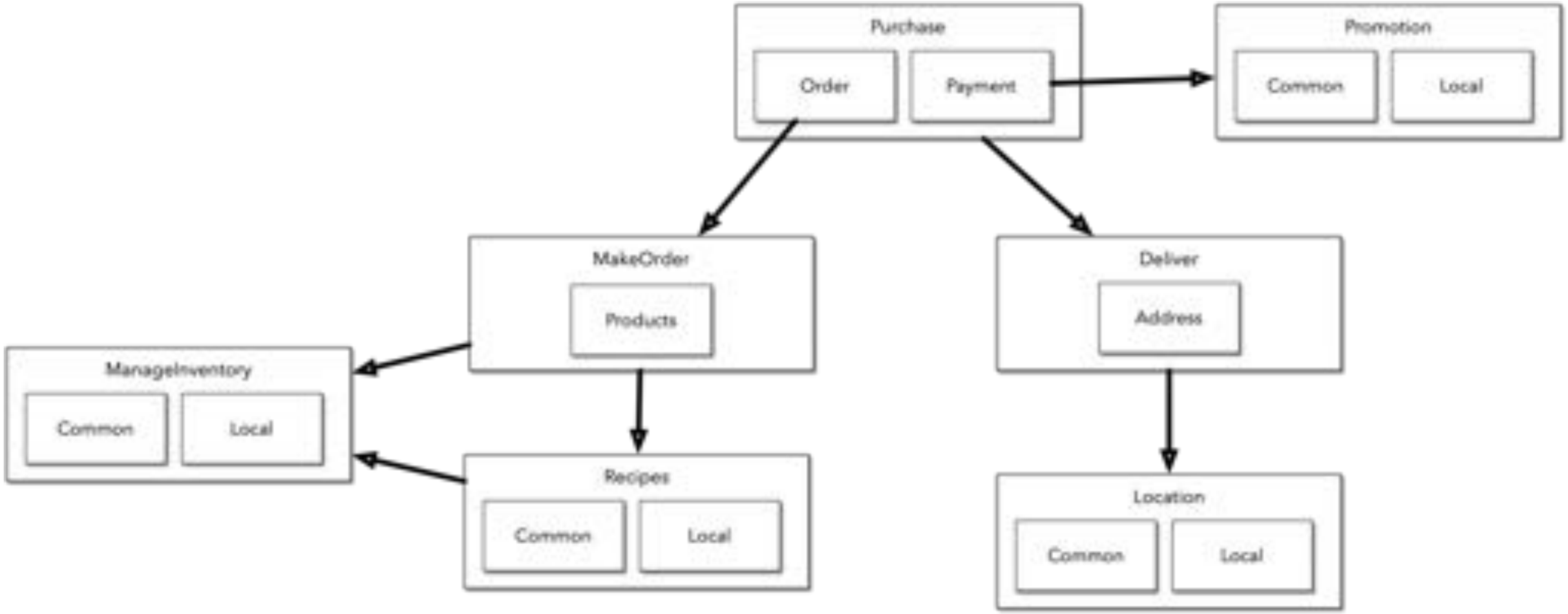
Silicon Sandwiches

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

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

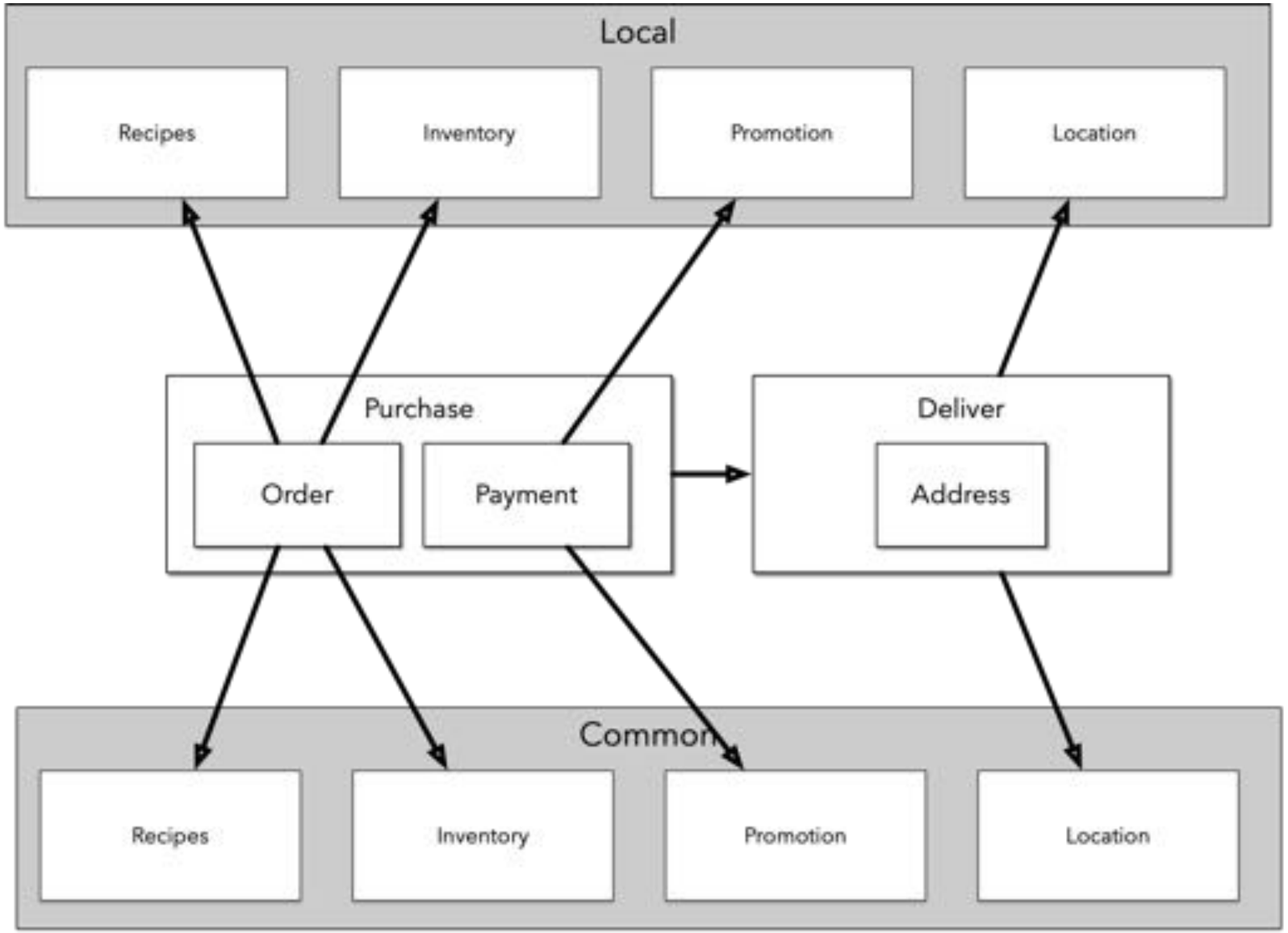
Your Architectural Kata is...

Silicon Sandwiches



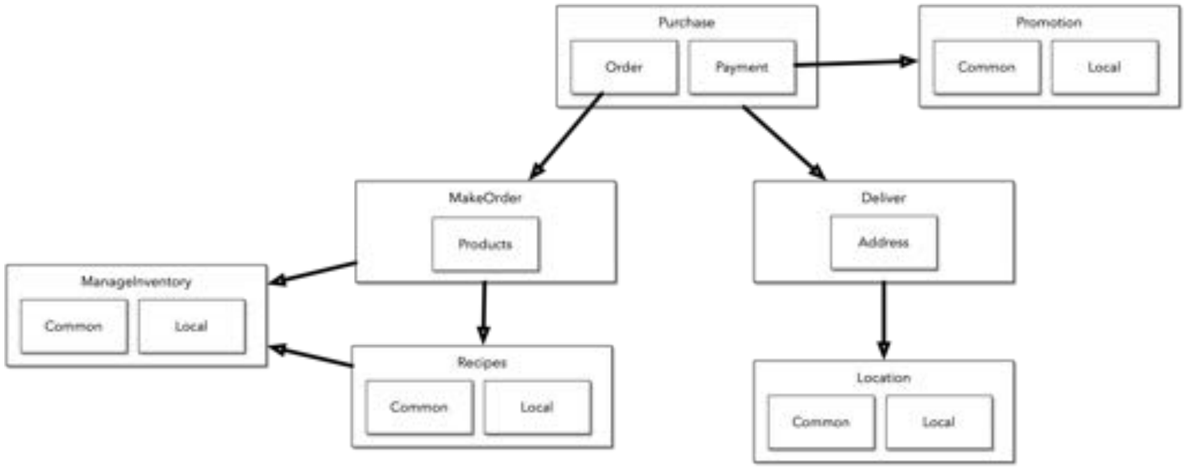
Your Architectural Kata is...

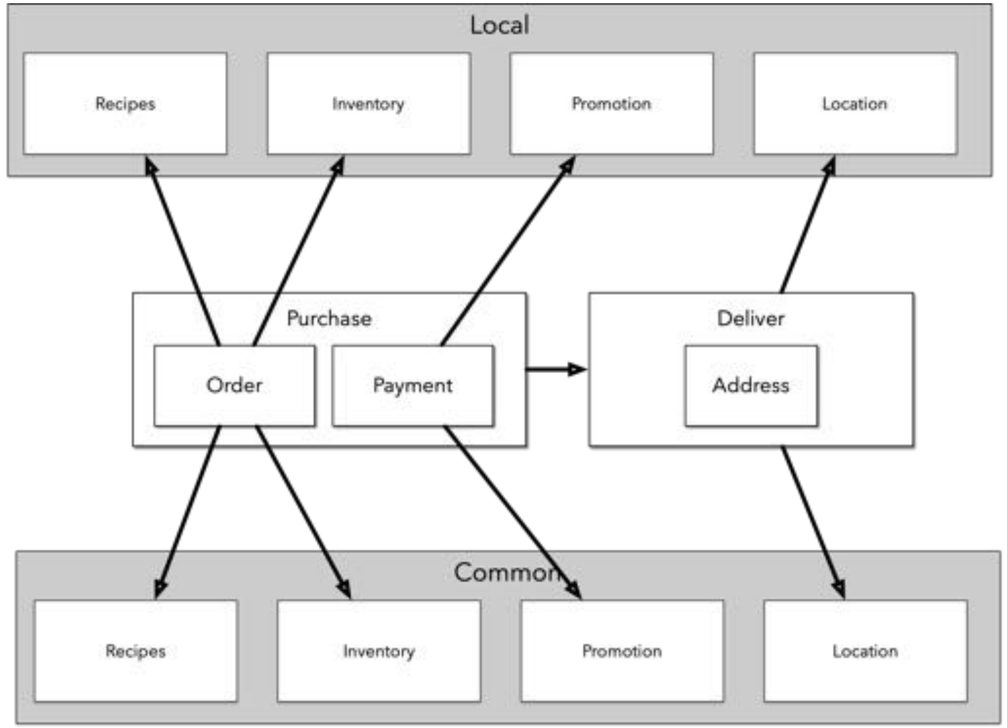
Silicon Sandwiches



Your Architectural Kata is...

Silicon Sandwiches





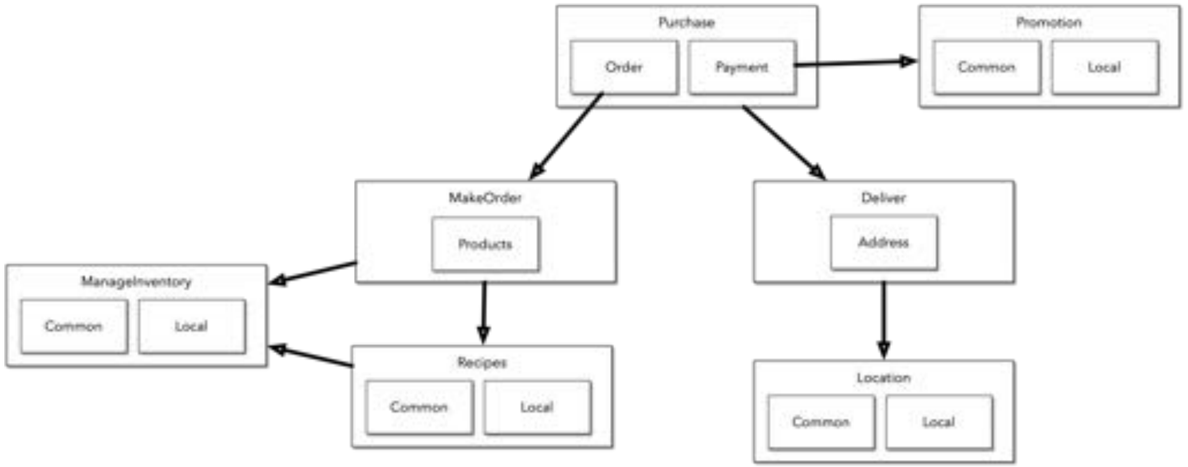
domain partitioned

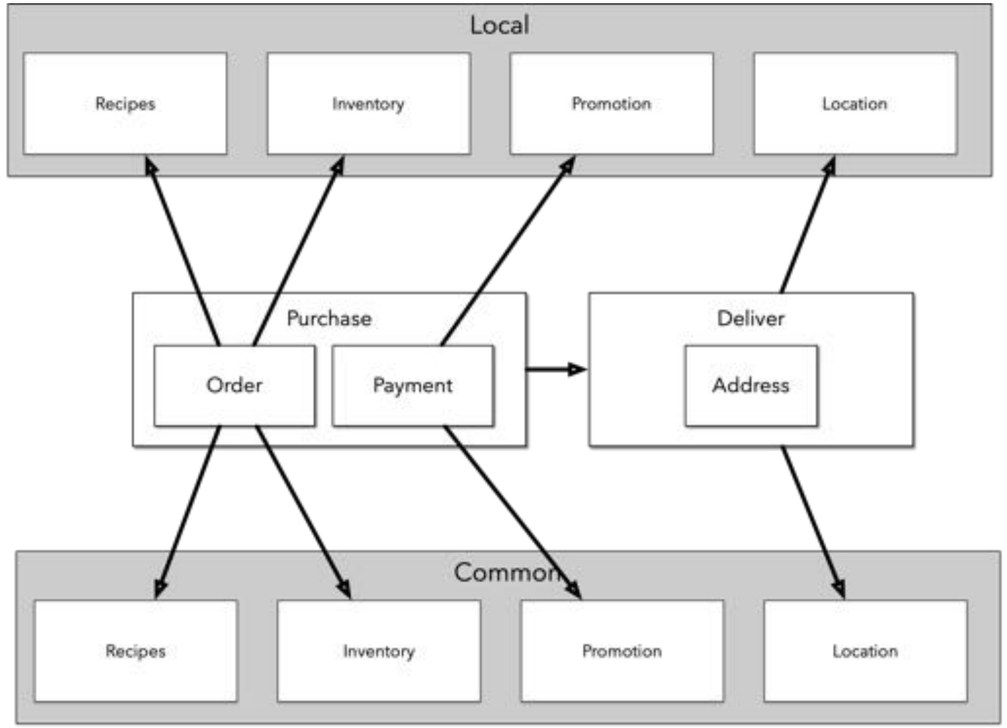
technically partitioned

Which one is better?

Your Architectural Kata is...

Silicon Sandwiches





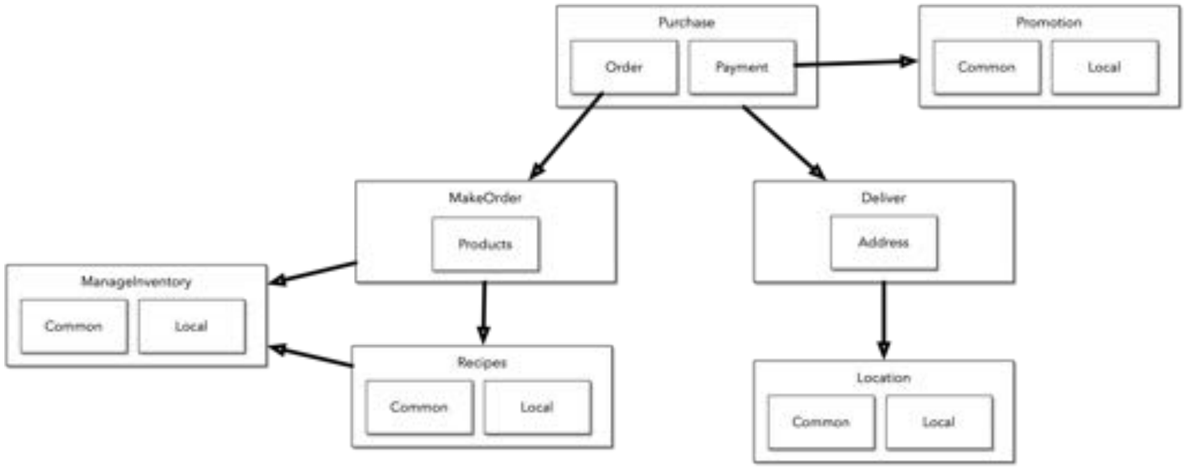
domain partitioned

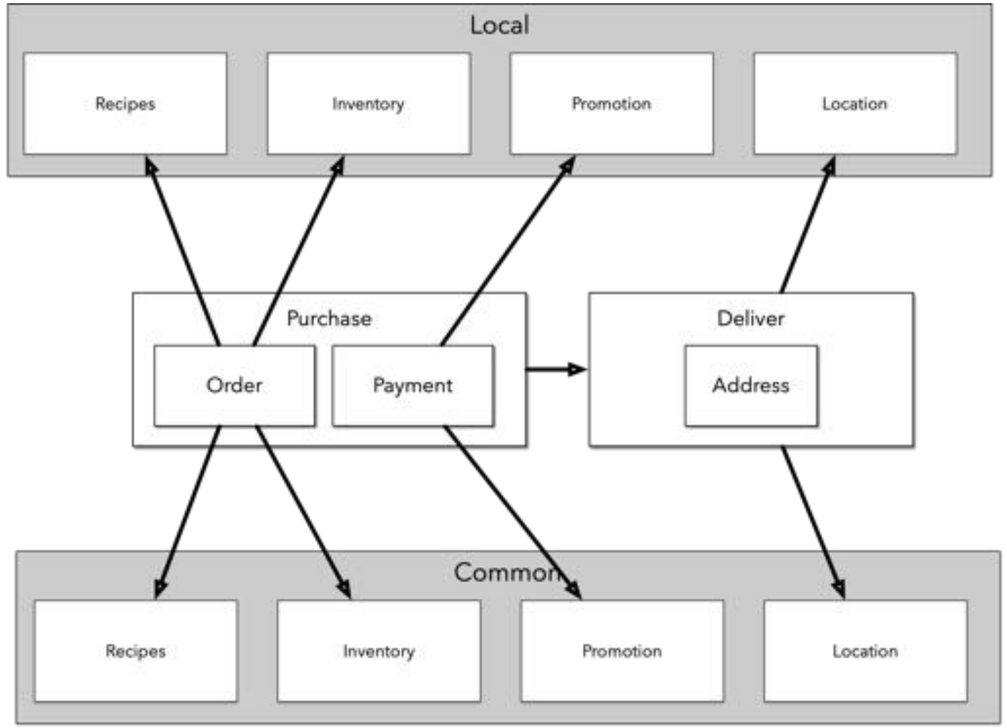
technically partitioned

Which one is "better"?

Your Architectural Kata is...

Silicon Sandwiches





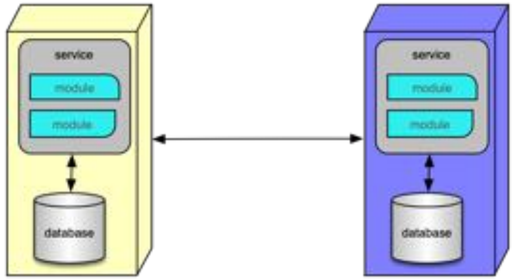
domain partitioned

technically partitioned

Which one is less worse?

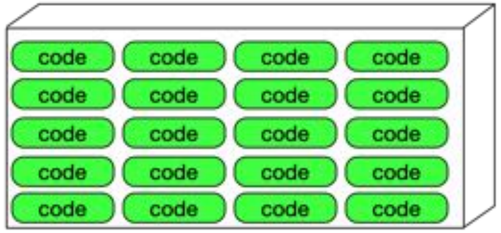
connascence properties

Locality



static

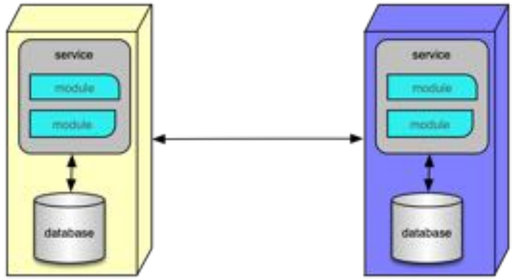
Proximal code (in the same module, class, or function) should have more & higher forms of connascence than less proximal code (in separate modules, or even codebases).



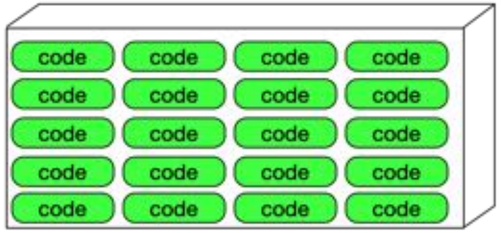
dynamic

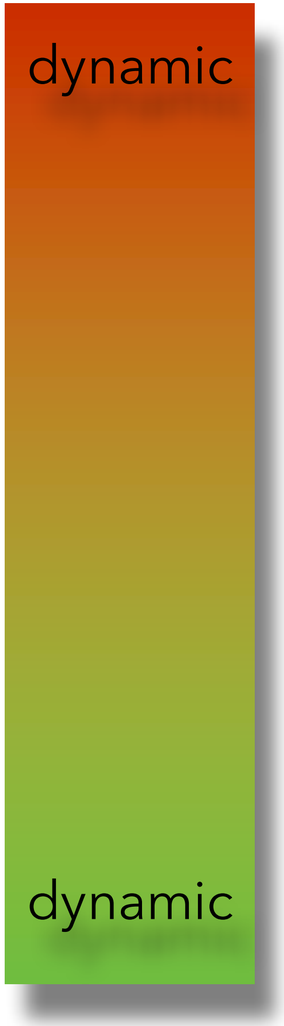
connascence properties

Locality



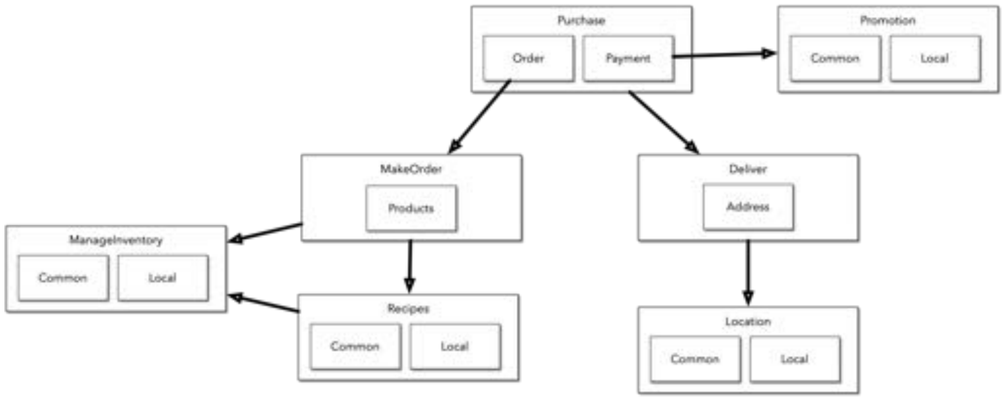
locality matters!



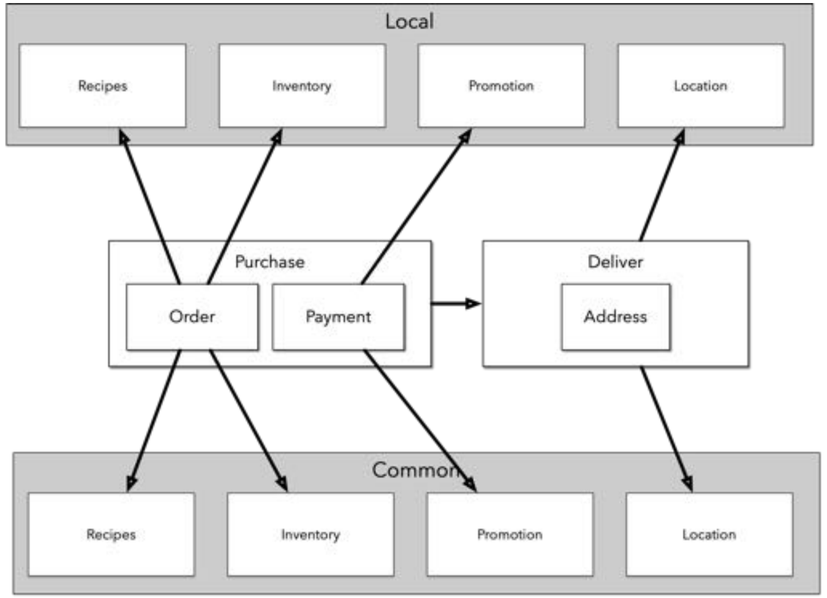


Your Architectural Kata is...

Silicon Sandwiches



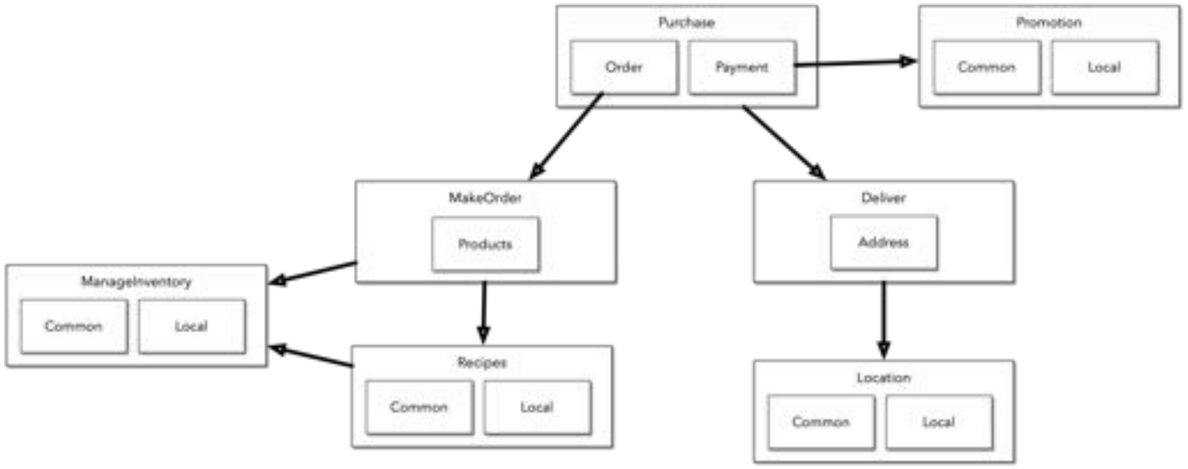
domain partitioned



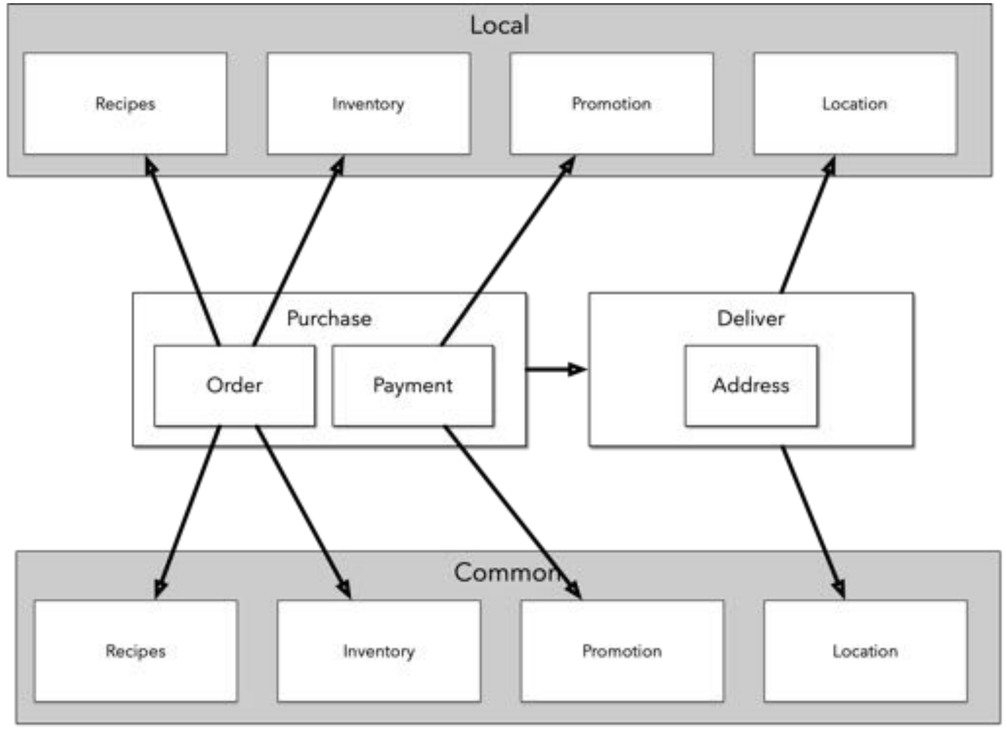
technically partitioned

Your Architectural Kata is...

Silicon Sandwiches



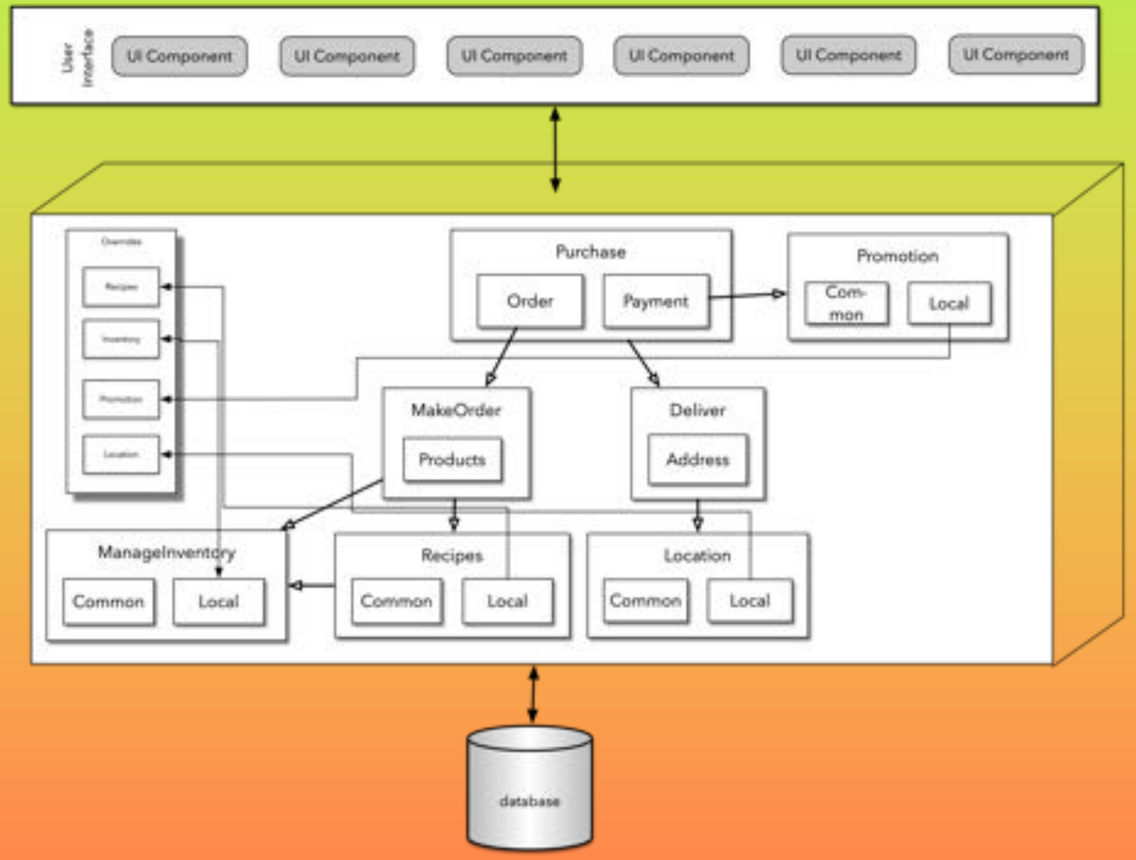
domain partitioned

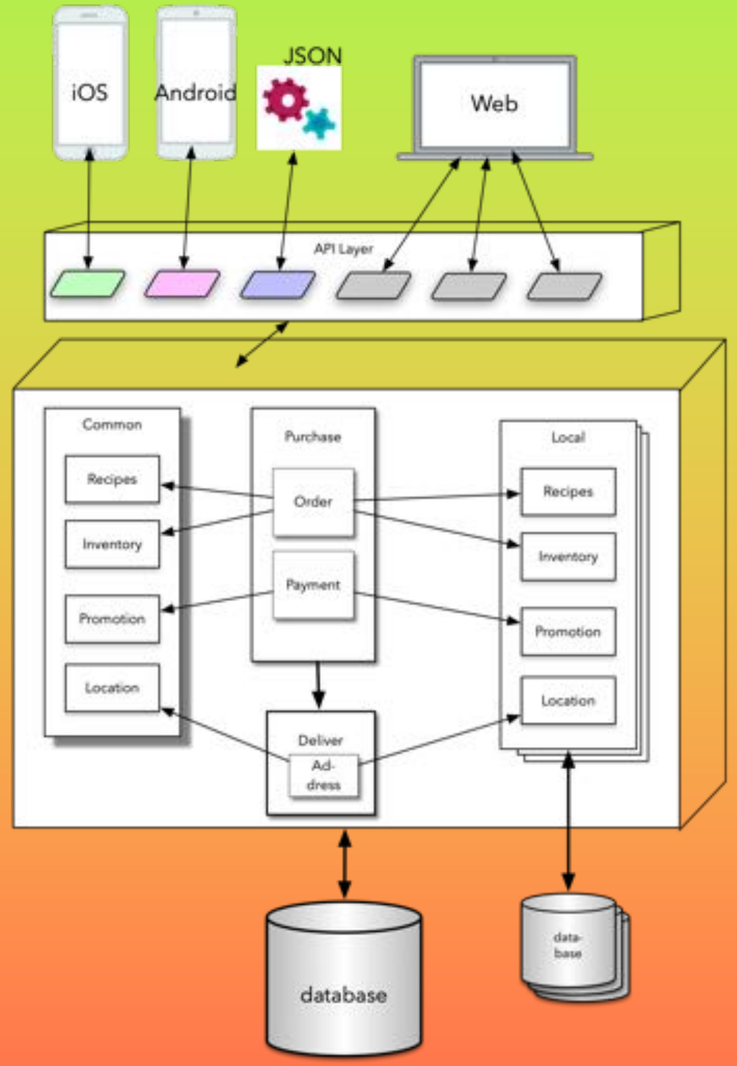


technically partitioned

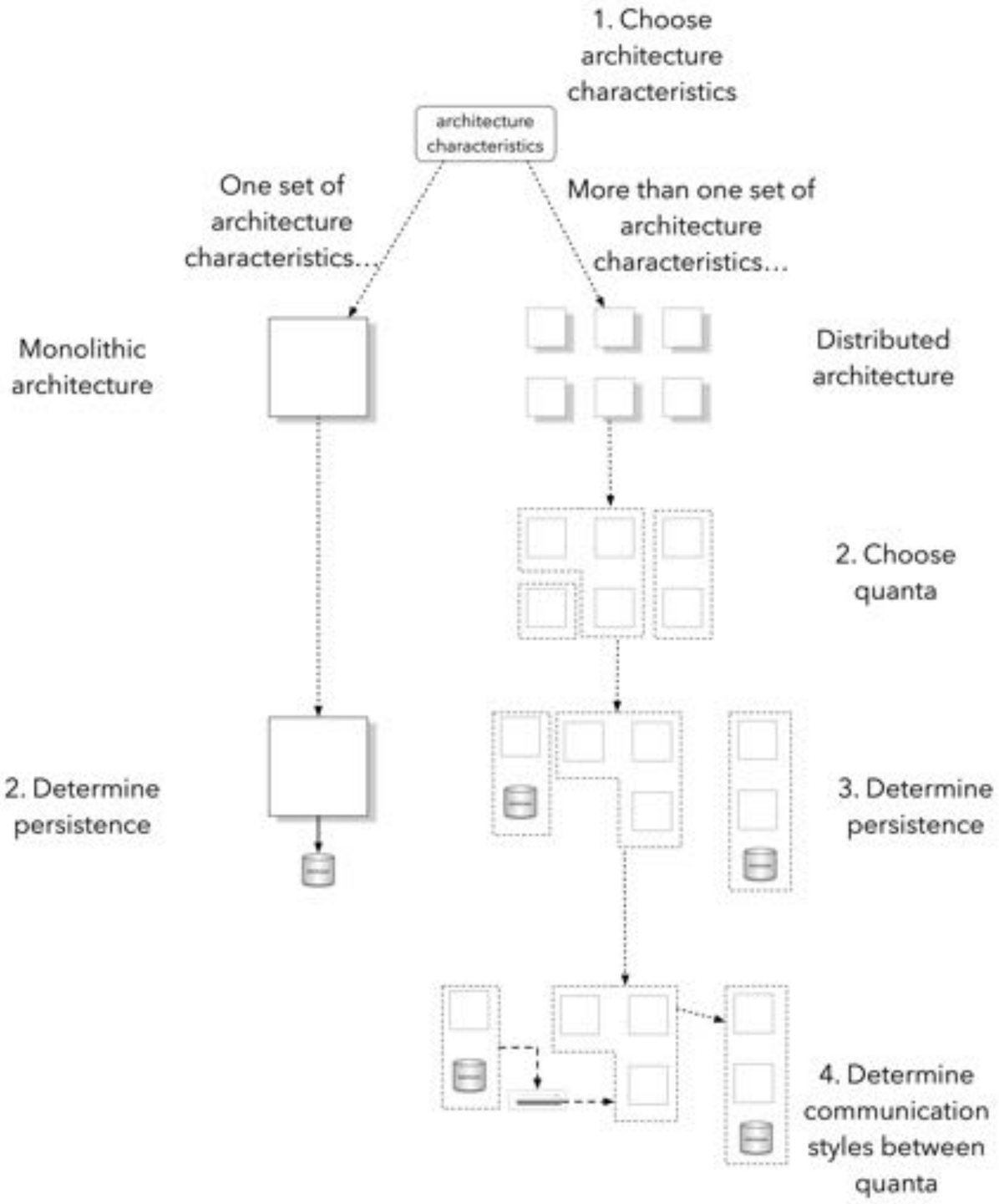
tradeoffs

domain VS technical partitioning

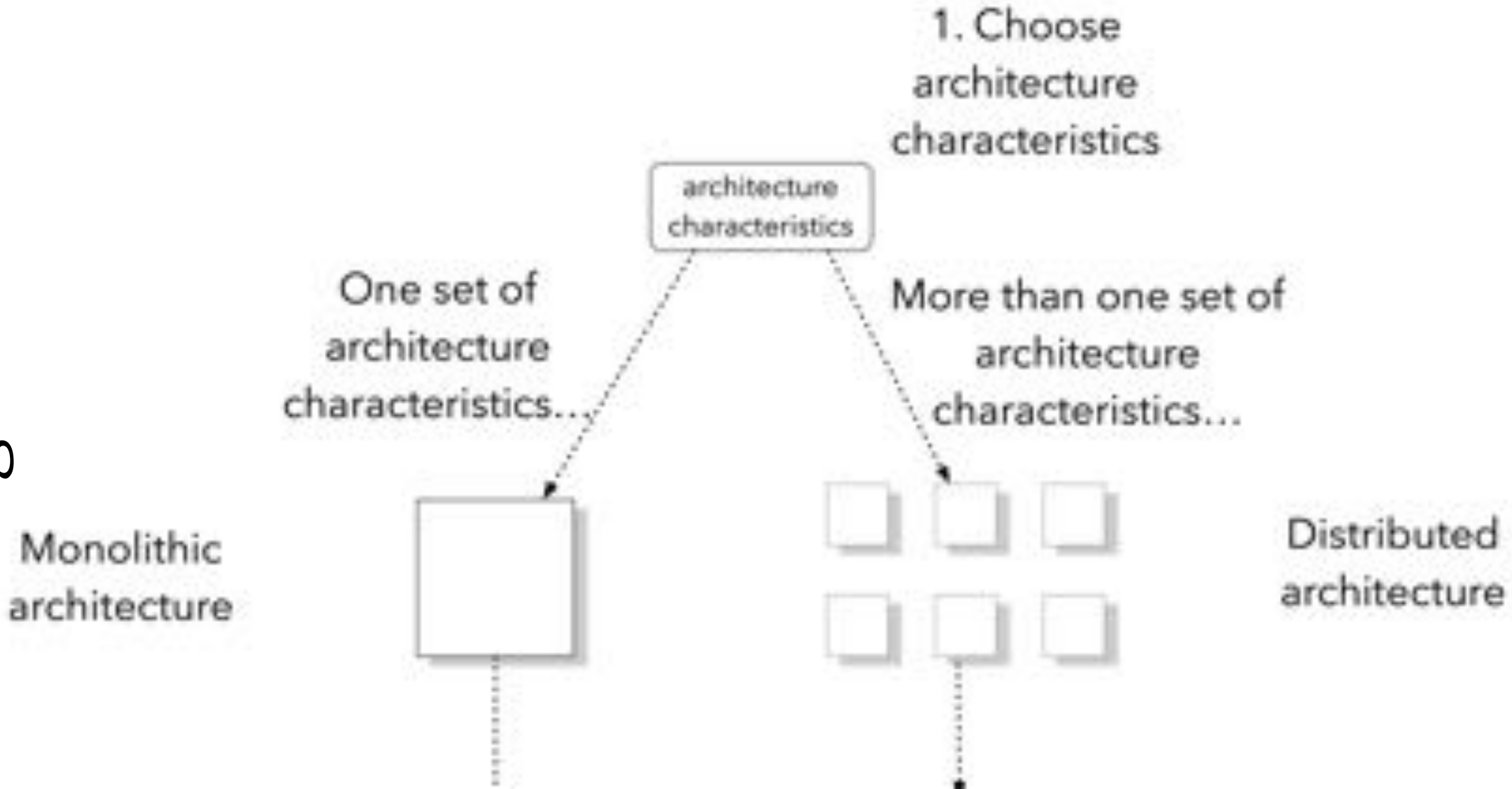




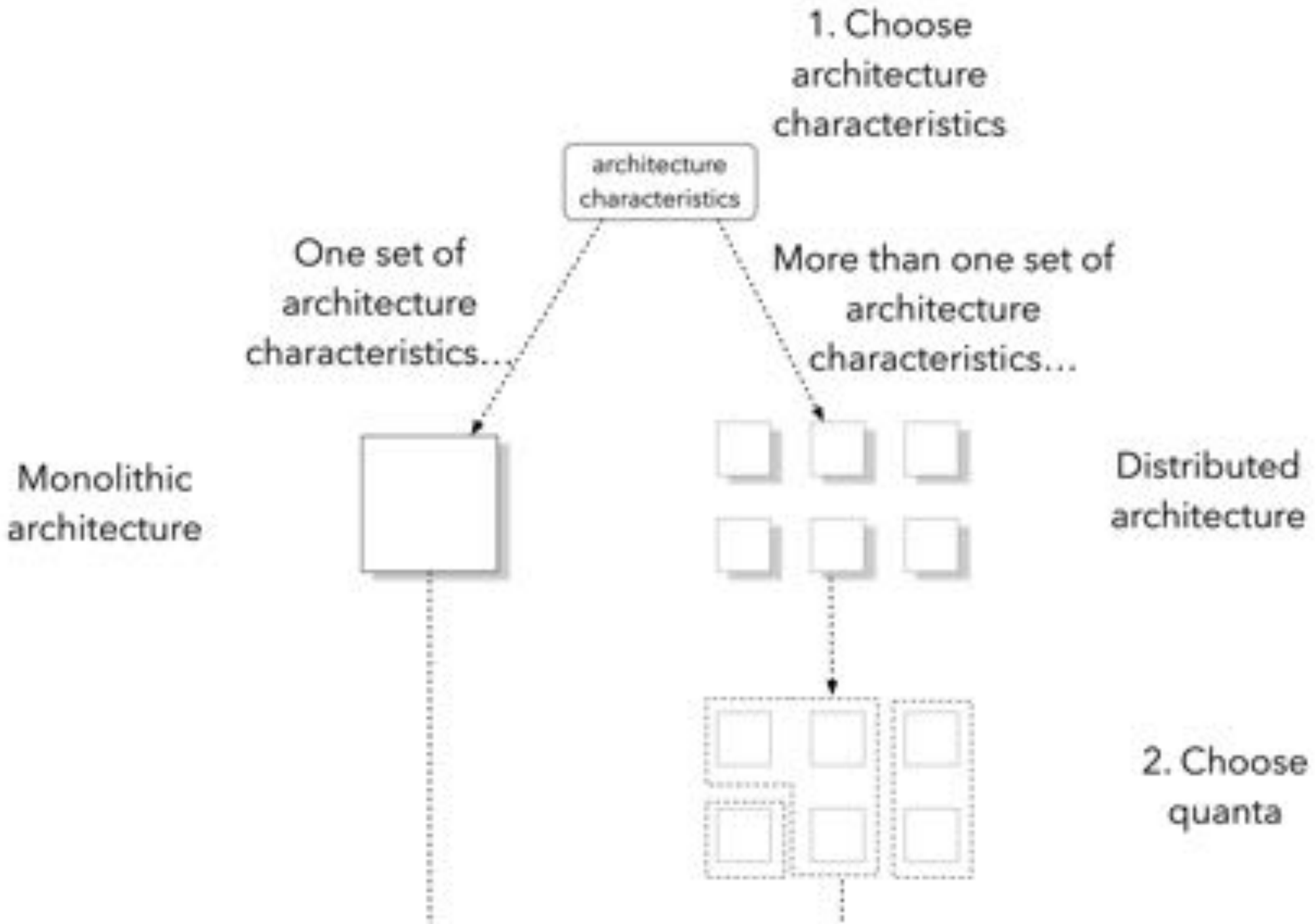
choosing an architecture



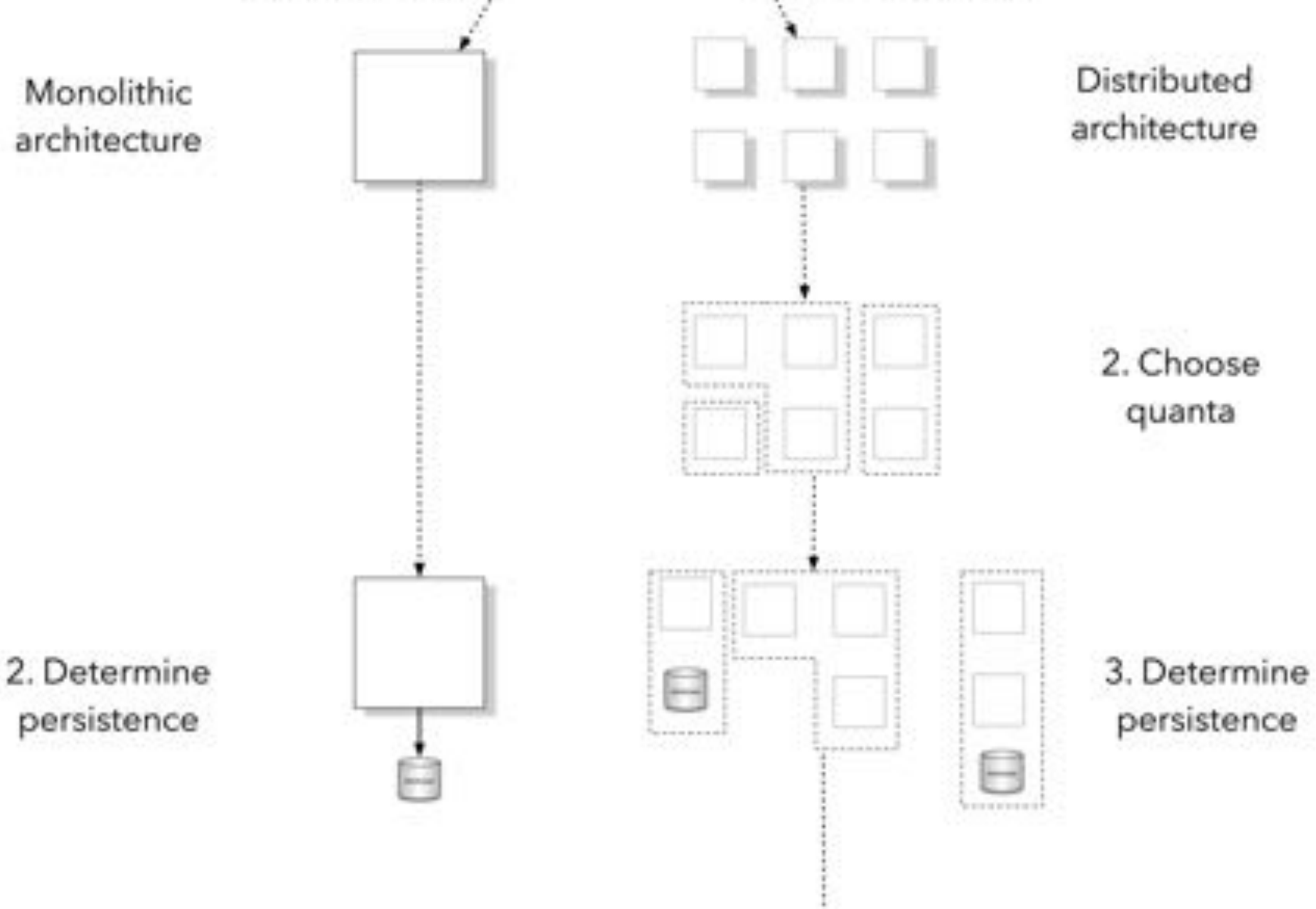
choosing an architecture



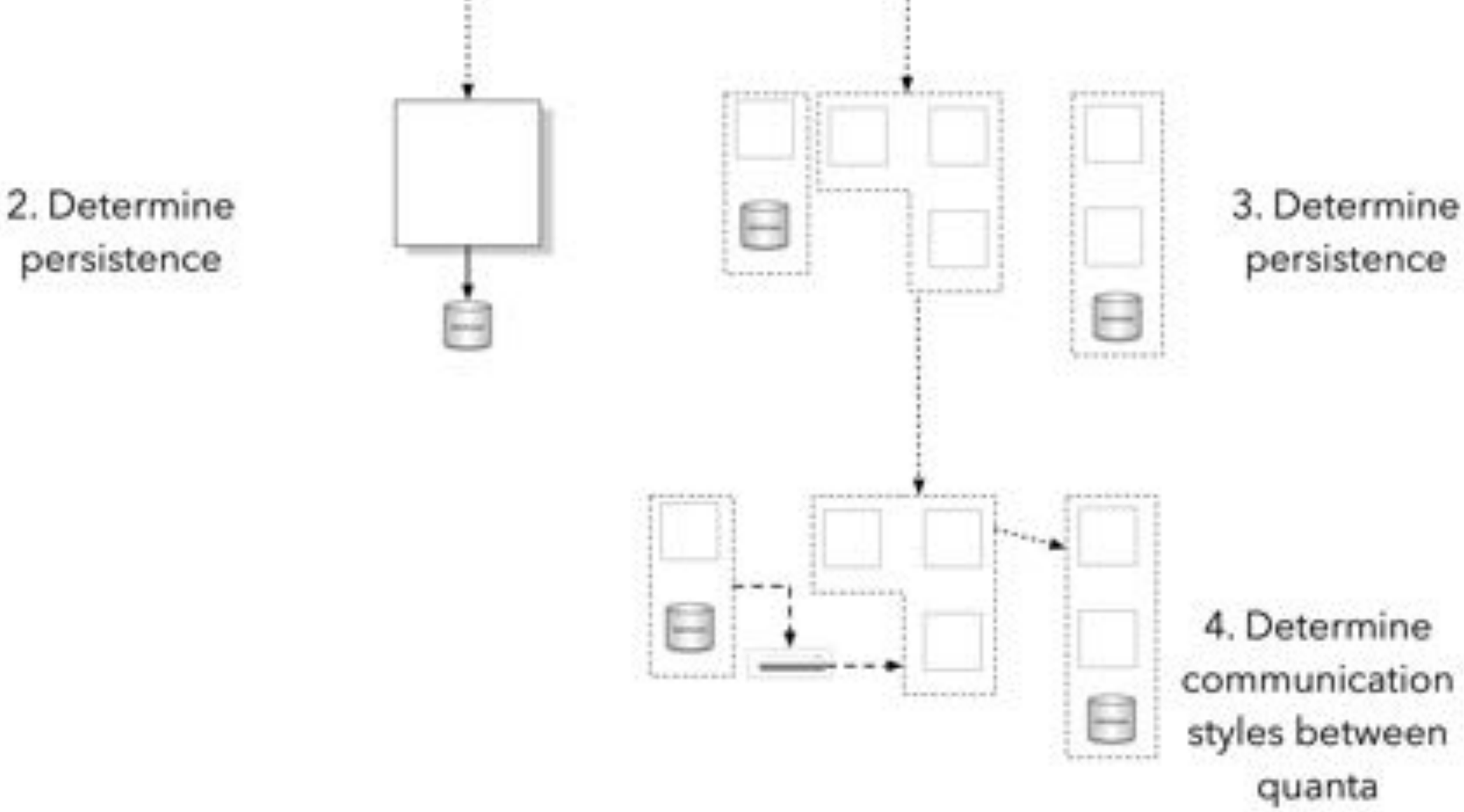
choosing an architecture



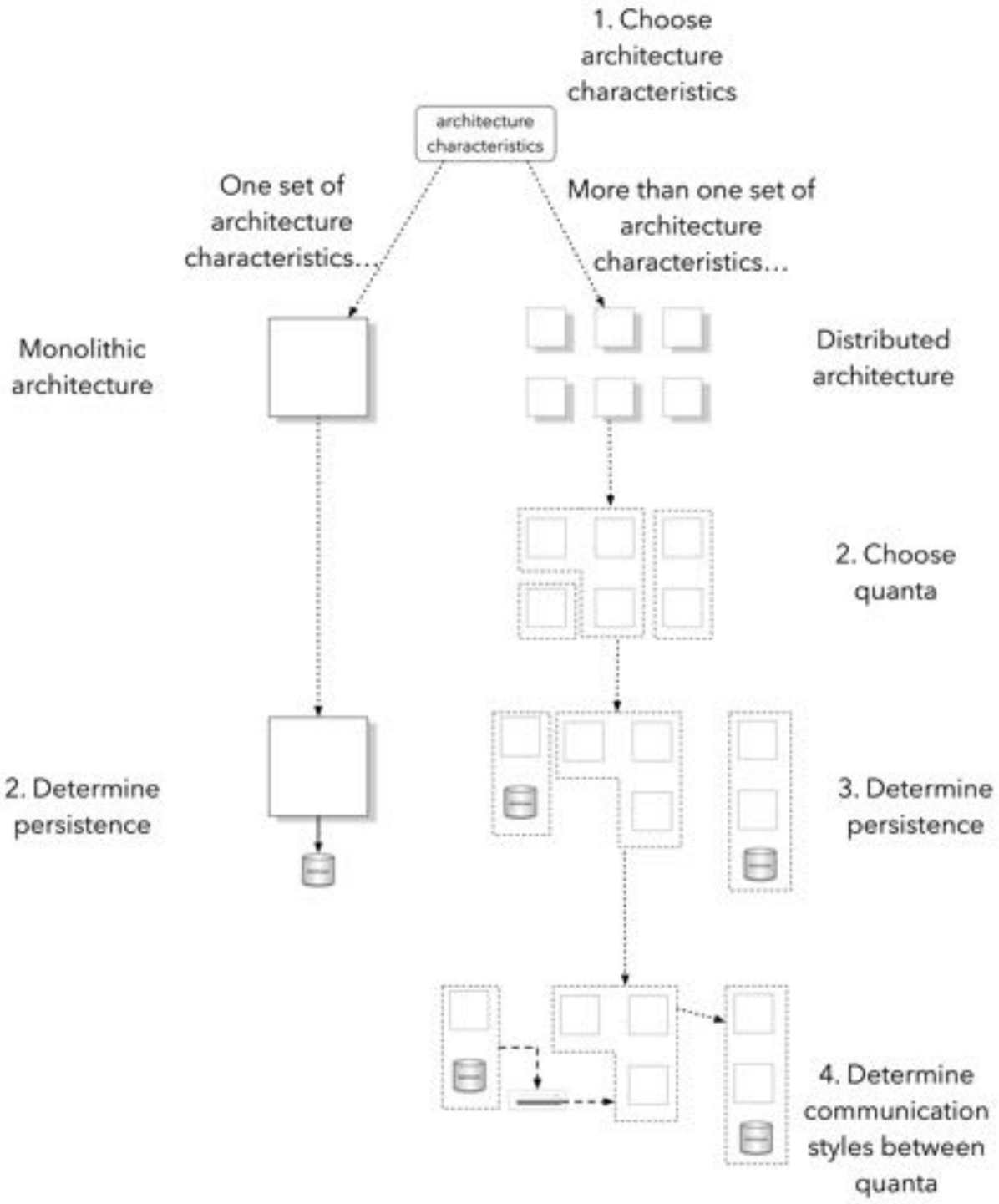
choosing an architecture



choosing an architecture



choosing an architecture



Sysops Squad

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

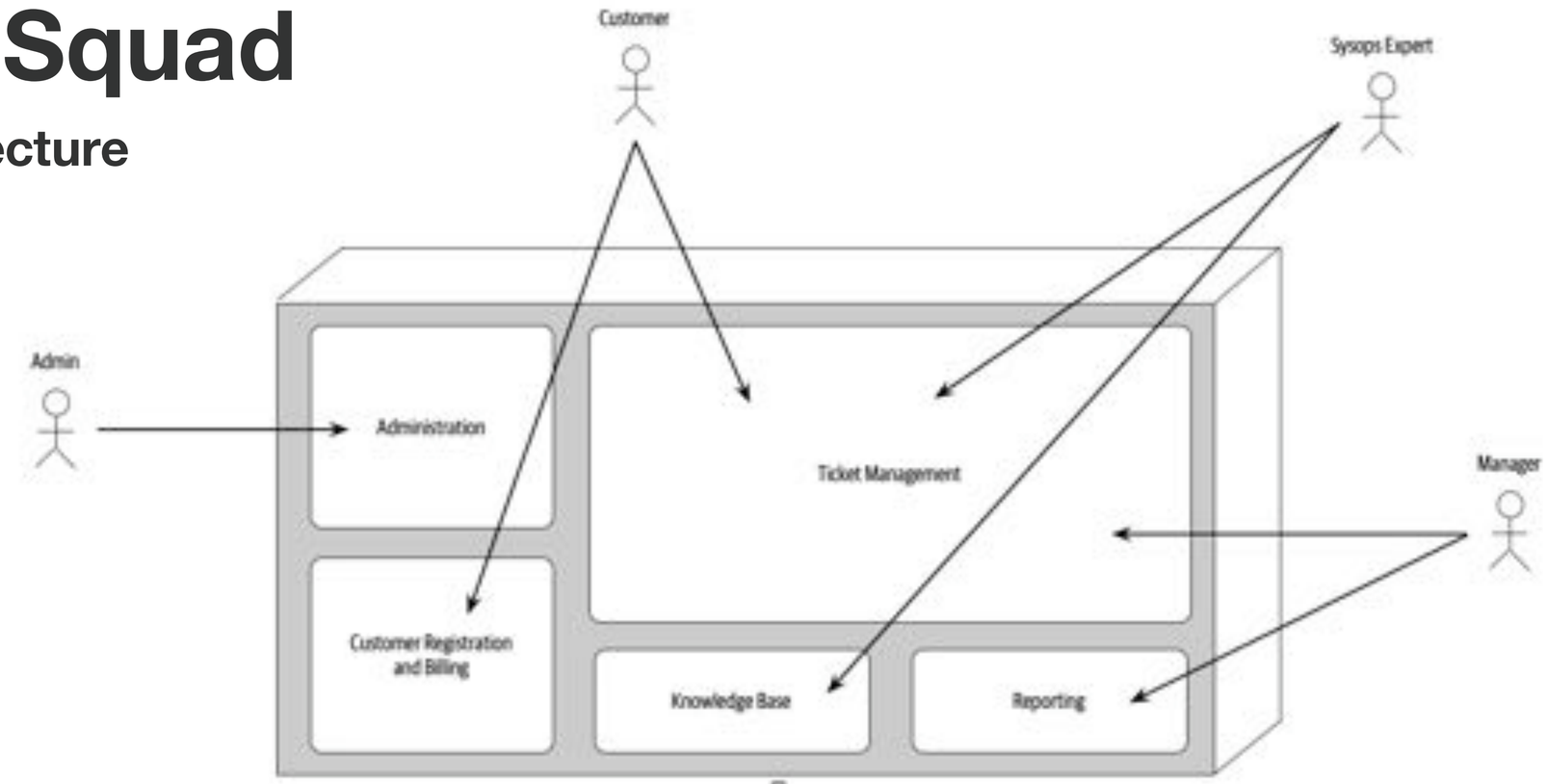
Things have not been good with the Sysops Squad lately. The current trouble ticket system is a monolithic application that was developed many years ago. Customers are complaining that consultants are never showing up due to lost tickets, and often times the wrong consultant shows up to fix something they know nothing about. Customers and call-center staff have been complaining that the system is not always available for web-based or call-based problem ticket entry. This is because whenever a change is made to the system, something else usually breaks or the monolithic system keeps crashing after a deployment. If something isn't done soon, Best Electronics will be forced to abandon this very lucrative business line and fire all of the experts (including you, the architect).

Current process in the monolithic system:

1. Sysops squad experts are added and maintained in the system through an administrator, who enters in their service area, availability, and skills.
2. Customers who have purchased the support plan can enter a problem ticket using the sysops squad website. Customer registration for the support service is part of the system. The system bills the customer on an annual basis when their support period ends by charging their registered credit card.
3. Once a trouble ticket is entered in the system, the system then determines which sysops squad expert would be the best fit for the job based on skills, current location, service area, and availability (free or currently on a job).
4. The sysops squad expert is then notified via a text message, along with the trouble ticket number. Once this happens an email or SMS text message is sent to the customer (based on their profile preference) that the expert is on their way.
5. The sysops squad expert then uses a custom application on their phone to access the ticketing system to retrieve the ticket information and location. The sysops squad expert can access the knowledge base through the mobile app to find out what things have been done in the past to fix the problem.
6. Once the sysops squad expert fixes the problem, they mark the ticket as "complete". The system then sends the customer an email or text with a survey to fill out. Customers are asked to give the expert a rating of 1 to 5 stars.
7. Optionally, the sysops squad expert can then add information about the problem and fix to the knowledge base. This is rarely done due to the slowness and complexity of adding information about the ticket. experts say "they can't be bothered", which drives management crazy.
8. Once the survey is received, it is added to the system. The sysops squad expert star rating is then added to his or hers profile. If an expert gets more than five 1-3 star ratings, they will show up on a probation report viewed by the sysops squad manager.
9. The sysops squad manager can access various reports using the system.

Sysops Squad

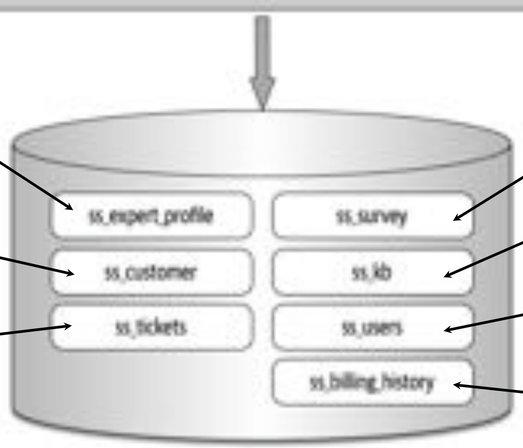
Current Architecture



contains sysops squad expert profiles

contains registered customer profile information (user id, password, address, email, text, preferences, credit card info, payment info, etc.)

contains all ticket information and the current status of each:
[entered | assigned | in progress | completed]



contains survey results from customers and expert star ratings

contains the knowledge base entered by experts after a job describing fixes to various problems

contains user id and password information for managers, admin staff, and sysops squad experts

contains customer billing history and statements

Kata Exercise 1 - Components and Quanta

Identify the major components (building blocks) of the monolith and what the appropriate architectural quanta should be for the new distributed system.

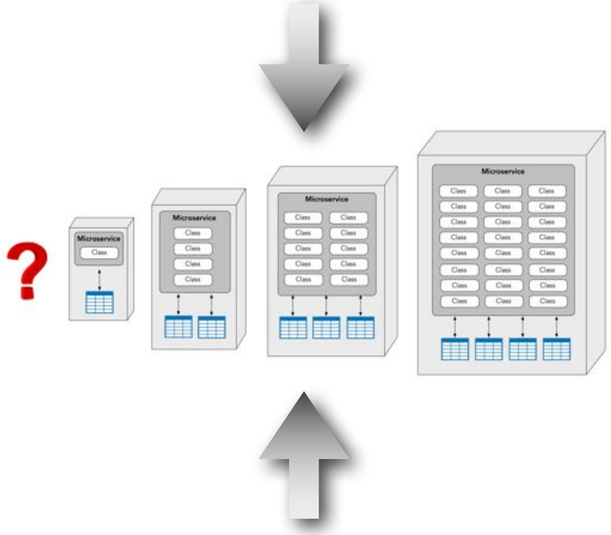
Current Domain Areas	Architectural Quantum	Core Components	
Administration			
Customer Registration and Billing			
Ticket Management			
Knowledge Base			
Reporting			

How do I determine the appropriate level of granularity?

service granularity

granularity drivers

"when should I consider breaking apart a service?"



granularity factors

"what factors influence service granularity?"

service granularity

granularity drivers

"when should I consider breaking apart a service?"

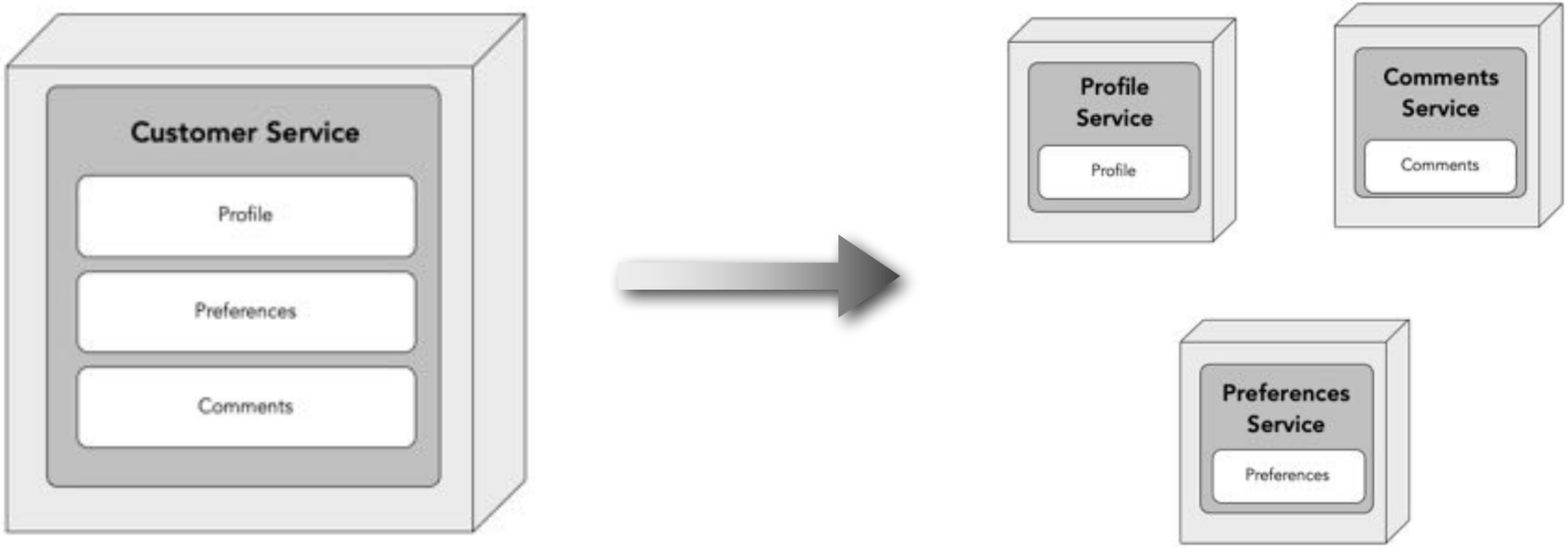
service
functionality

service granularity

granularity drivers

"when should I consider breaking apart a service?"

service functionality



service granularity

granularity drivers

"when should I consider breaking apart a service?"

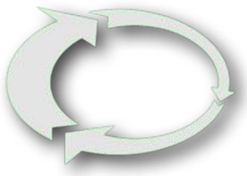
service
functionality

code
volatility

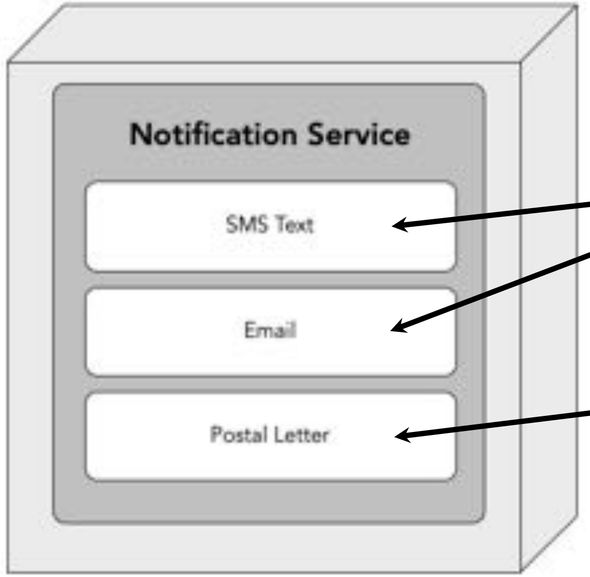
service granularity

granularity drivers

"when should I consider breaking apart a service?"



code
volatility



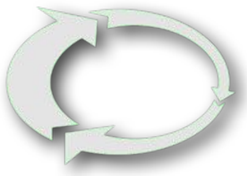
code rarely changes

code always changes

service granularity

granularity drivers

"when should I consider breaking apart a service?"



code
volatility

code rarely changes

code always changes

service granularity

granularity drivers

"when should I consider breaking apart a service?"

service functionality

code volatility

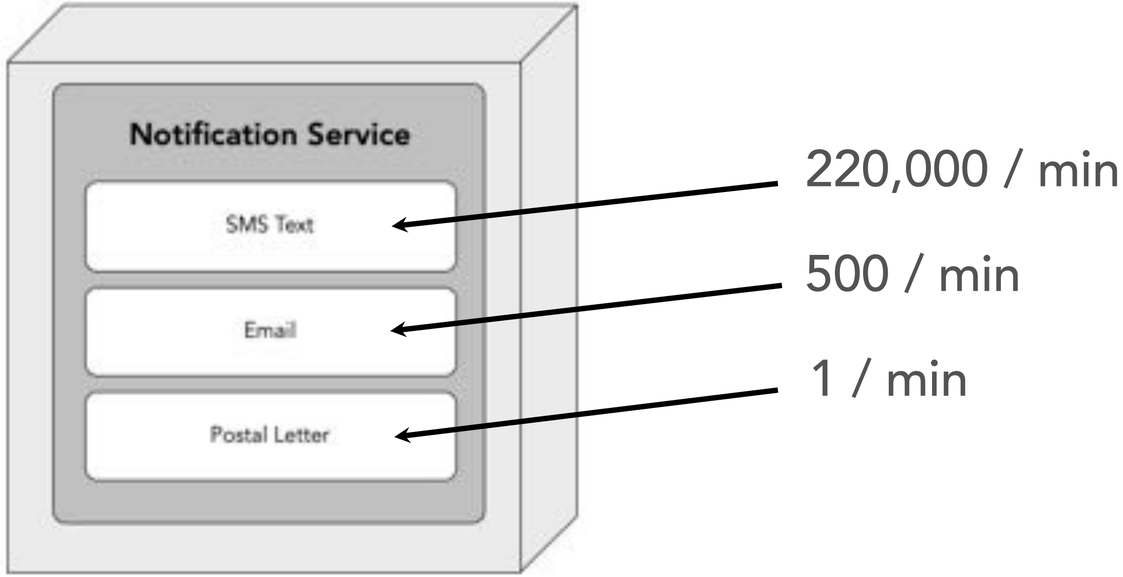
scalability and throughput

service granularity

granularity drivers

"when should I consider breaking apart a service?"

scalability and throughput



service granularity

granularity drivers

"when should I consider breaking apart a service?"

scalability and throughput

220,000 / min

500 / min

1 / min

service granularity

granularity drivers

"when should I consider breaking apart a service?"

service functionality

code volatility

scalability and throughput

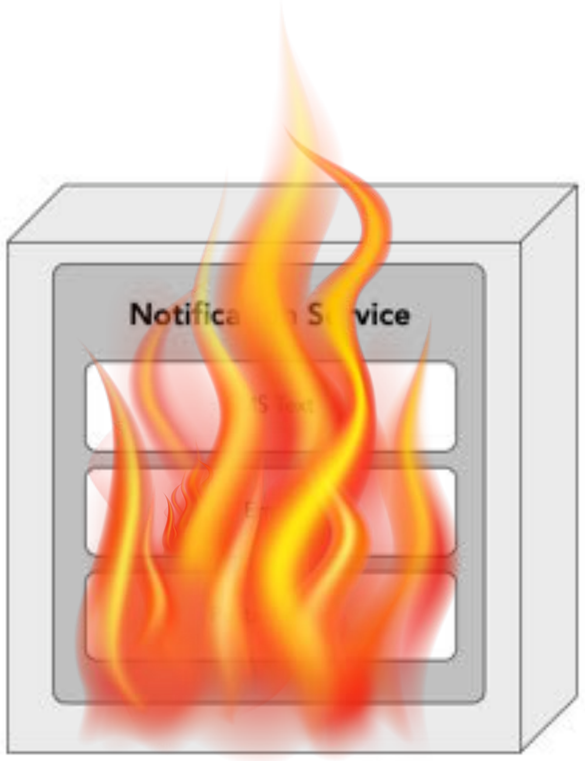
fault tolerance

service granularity

granularity drivers

"when should I consider breaking apart a service?"

fault tolerance

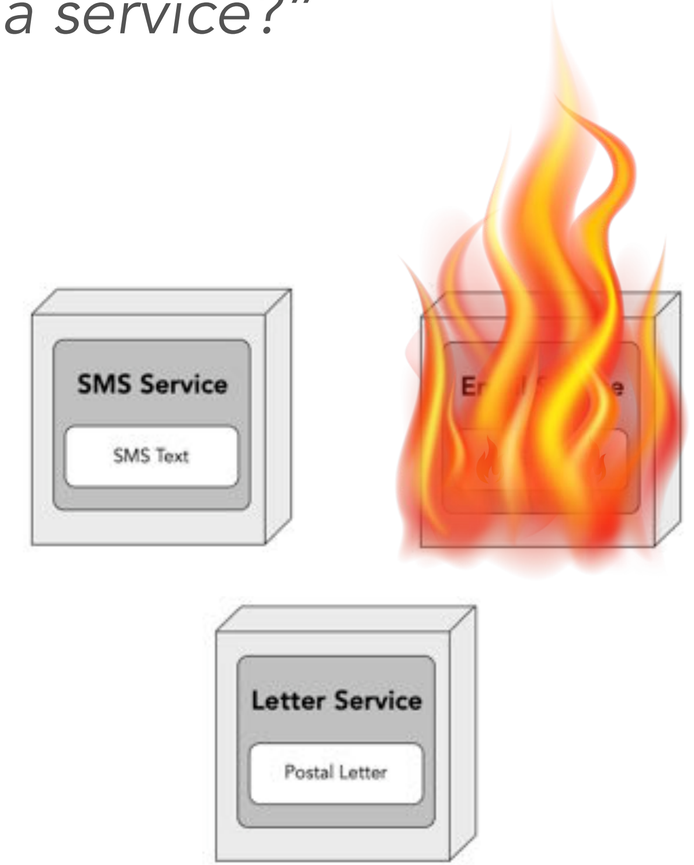


service granularity

granularity drivers

"when should I consider breaking apart a service?"

fault tolerance



service granularity

granularity drivers

"when should I consider breaking apart a service?"

service functionality

code volatility

scalability and throughput

fault tolerance

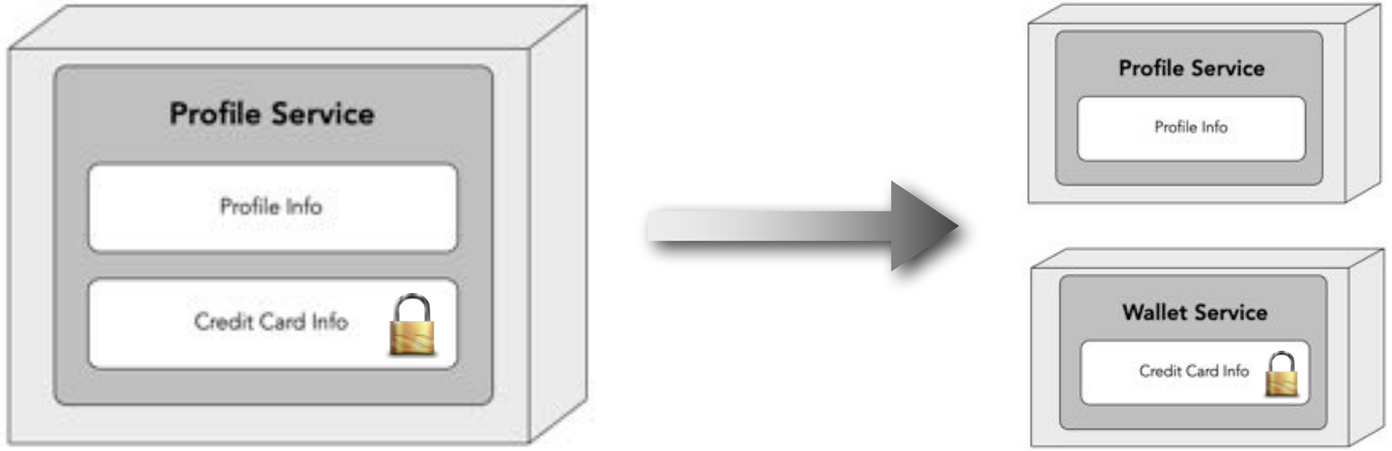
data security

service granularity

granularity drivers

"when should I consider breaking apart a service?"

data security



service granularity

granularity drivers

"when should I consider breaking apart a service?"

service functionality

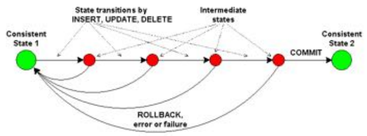
code volatility

scalability and throughput

fault tolerance

data security

service granularity

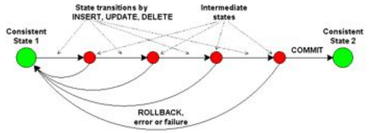


database
transactions

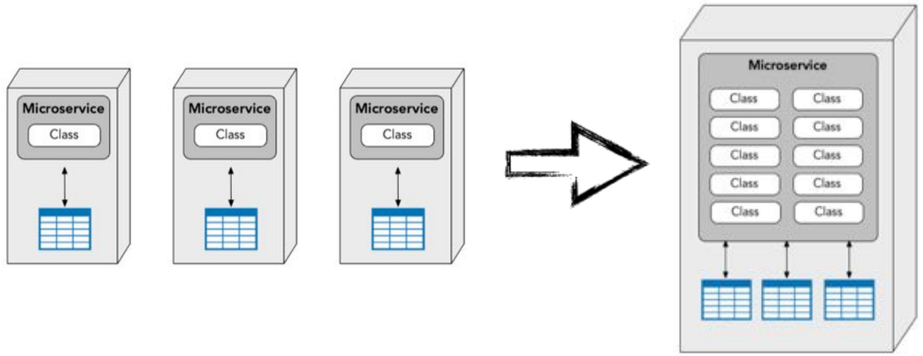
granularity factors

"what factors influence service granularity?"

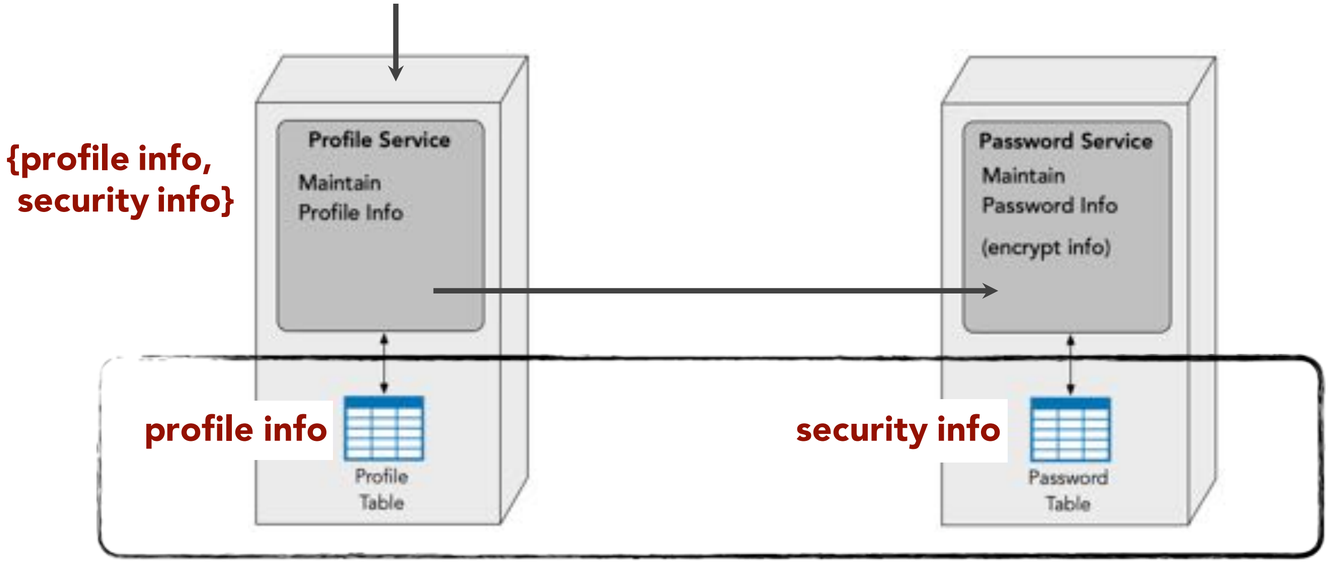
service granularity



database
transactions

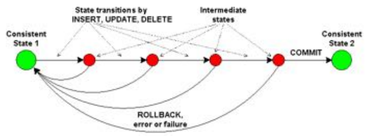


"register a new customer"



no acid
transaction

service granularity



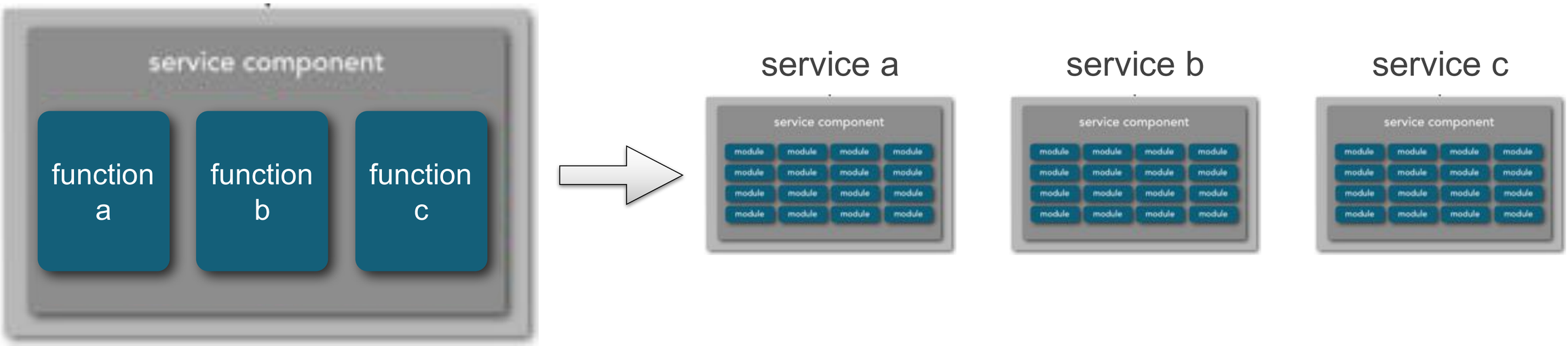
database
transactions

data
dependencies

granularity factors

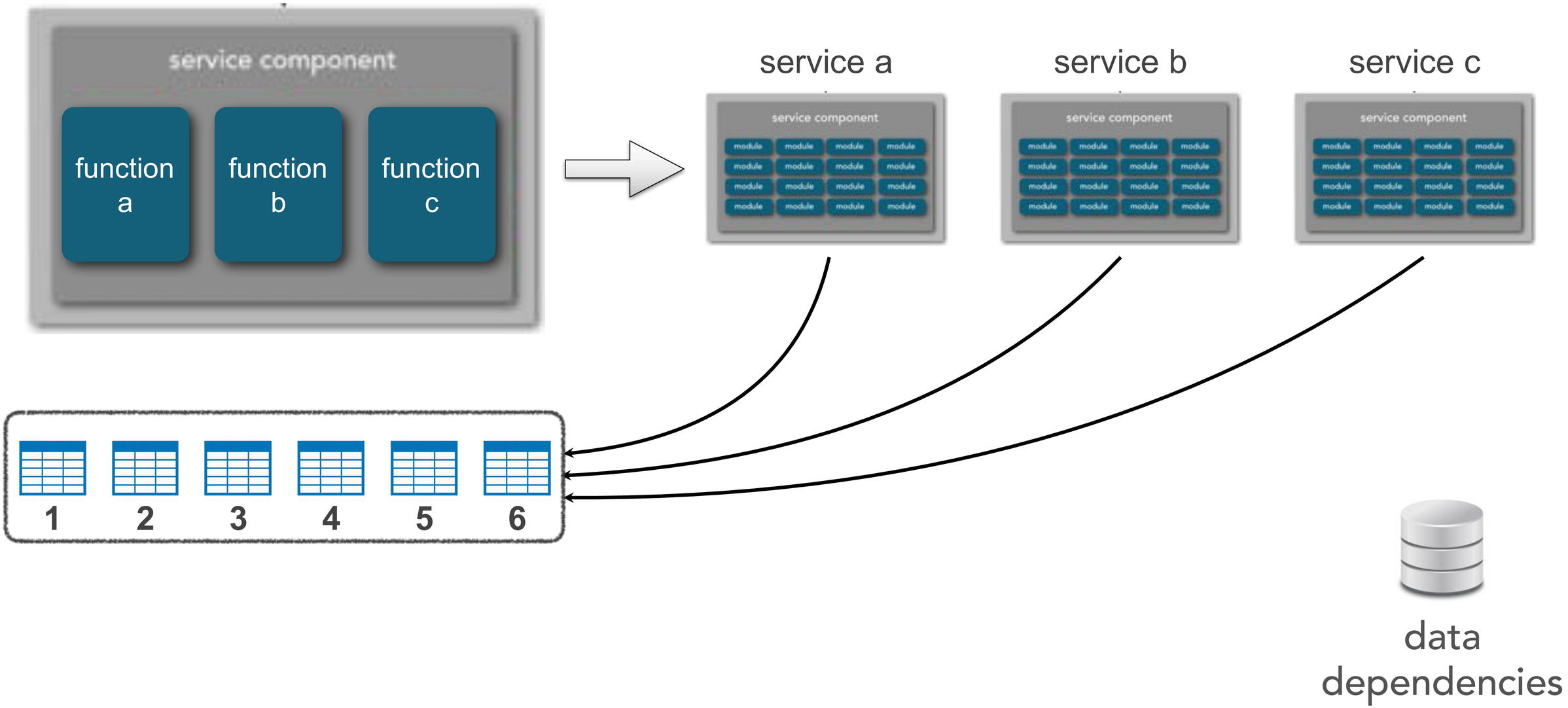
"what factors influence service granularity?"

service granularity

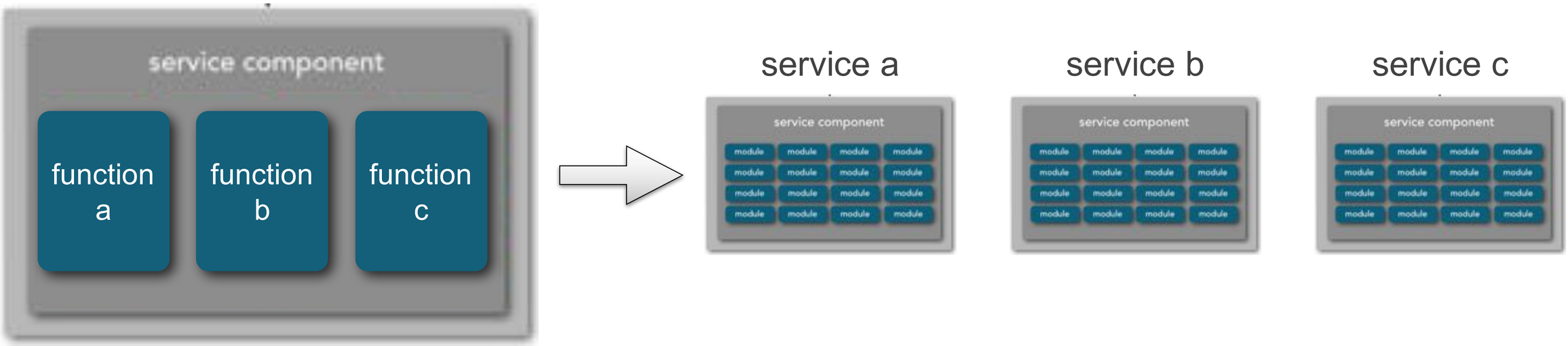


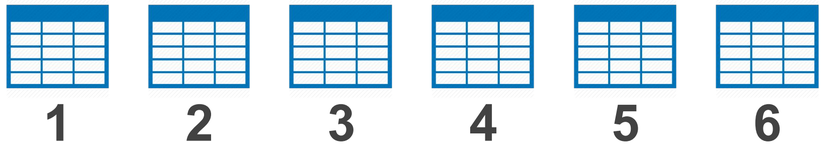
data dependencies

service granularity



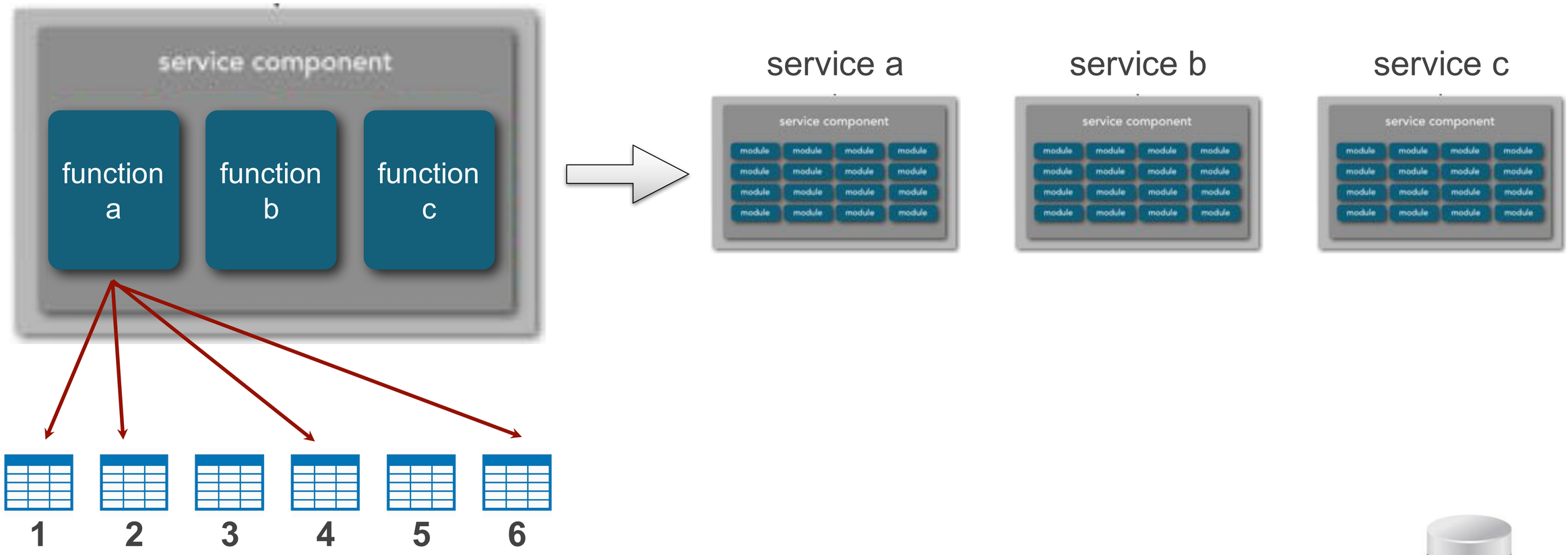
service granularity





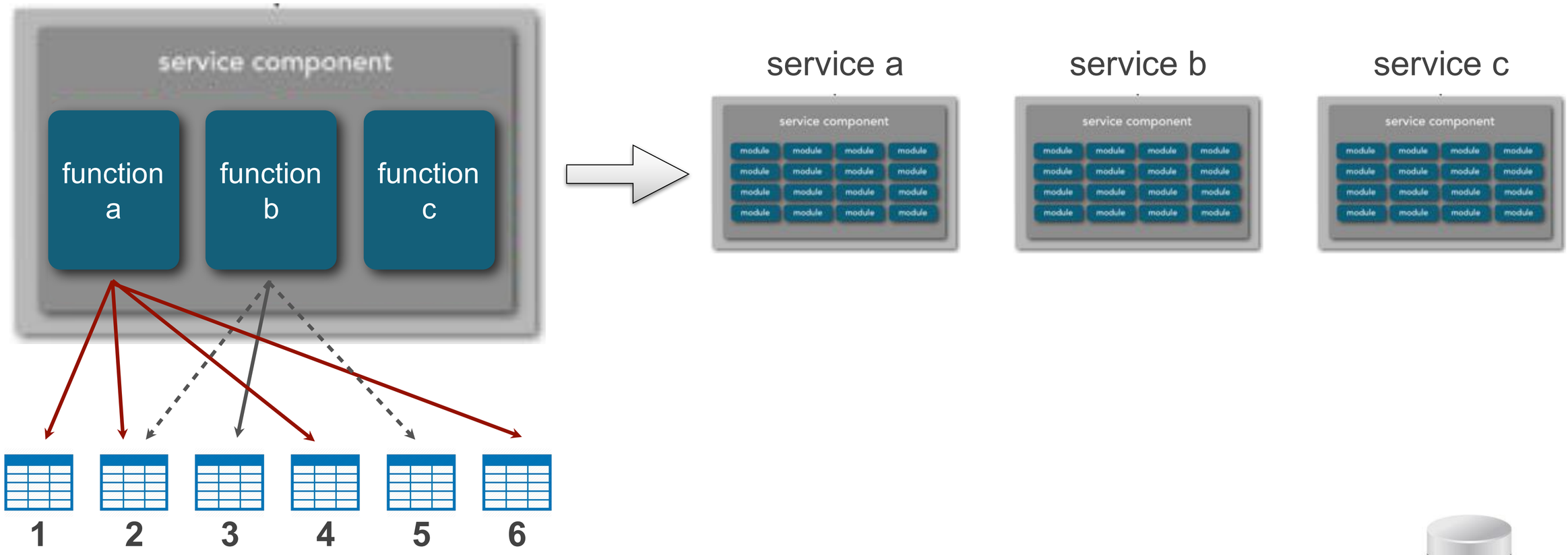
data dependencies

service granularity



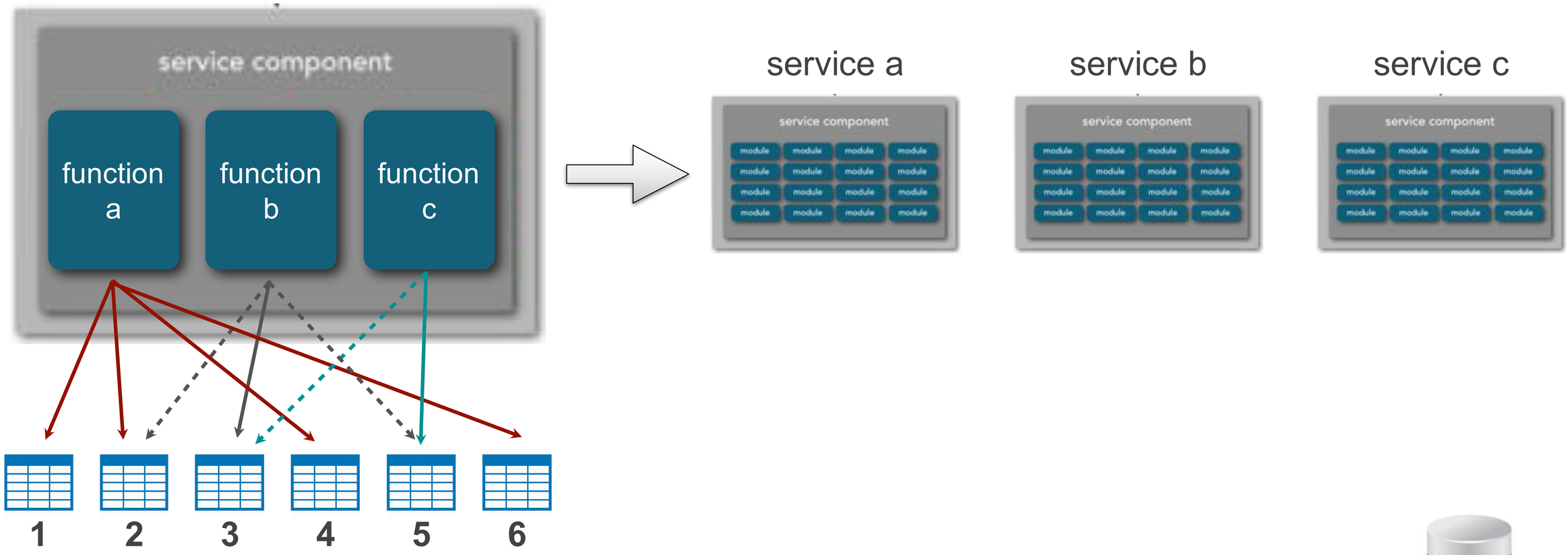
data dependencies

service granularity



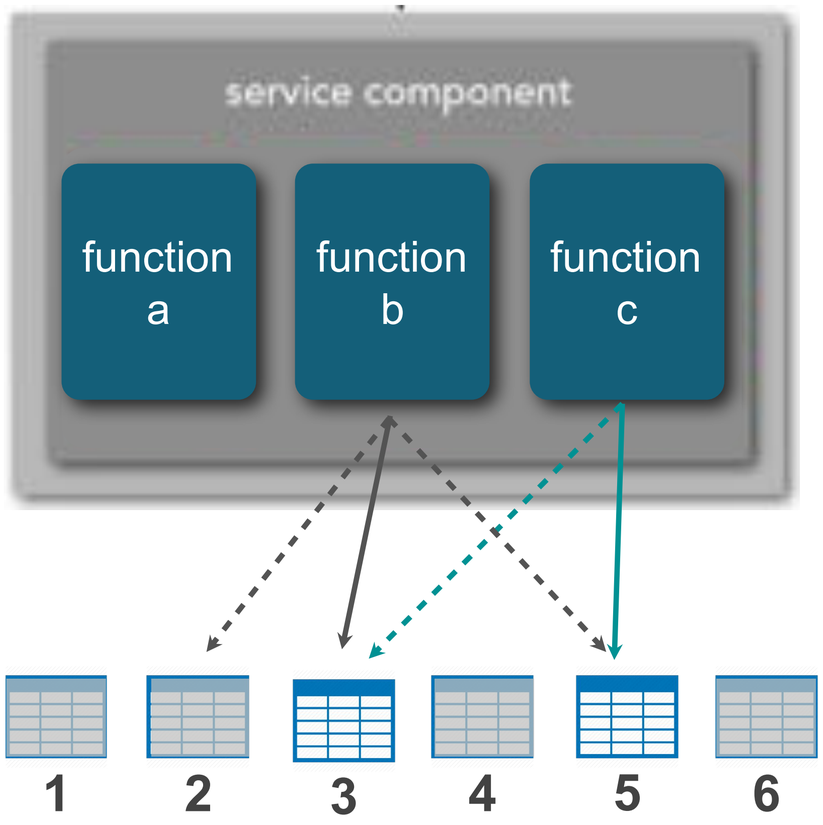
data dependencies

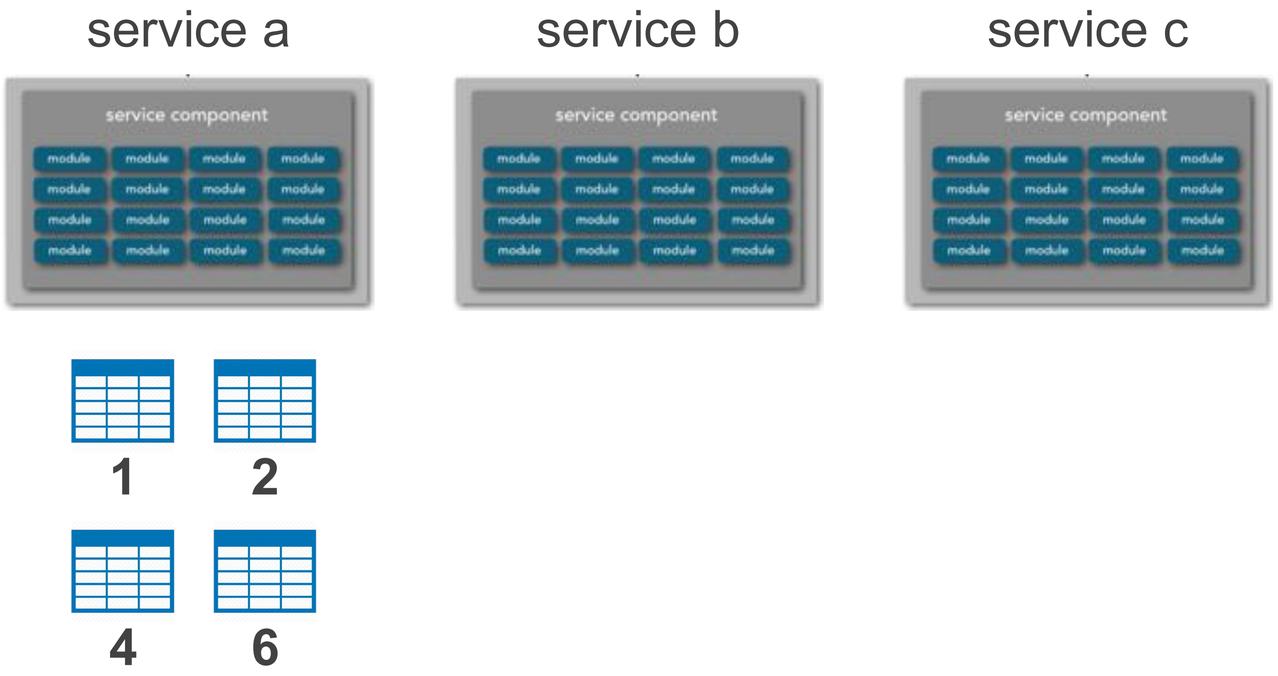
service granularity



data dependencies

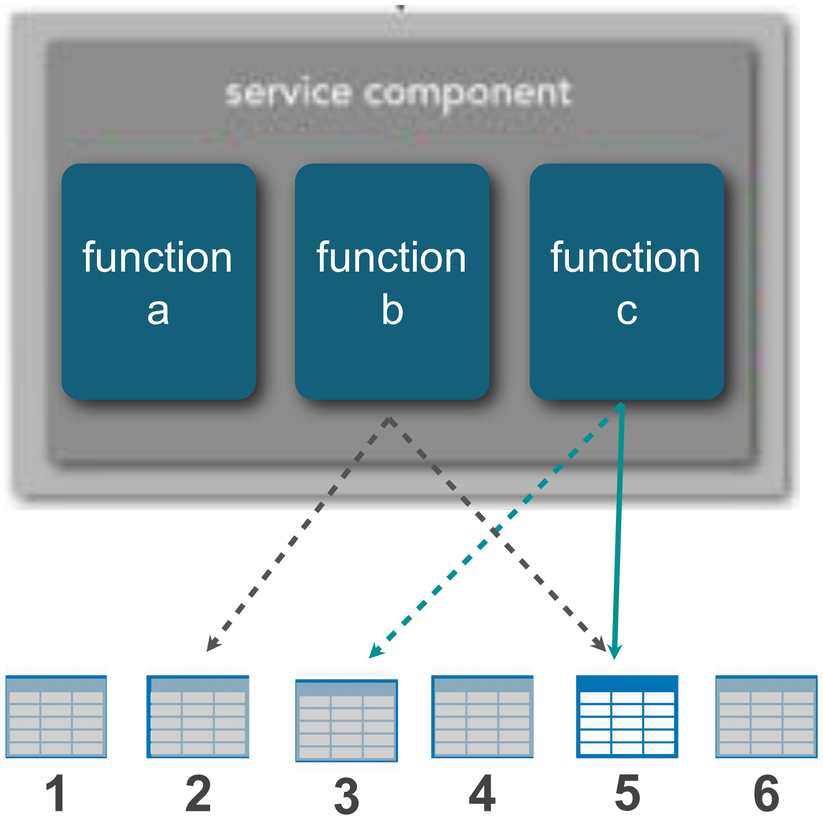
service granularity

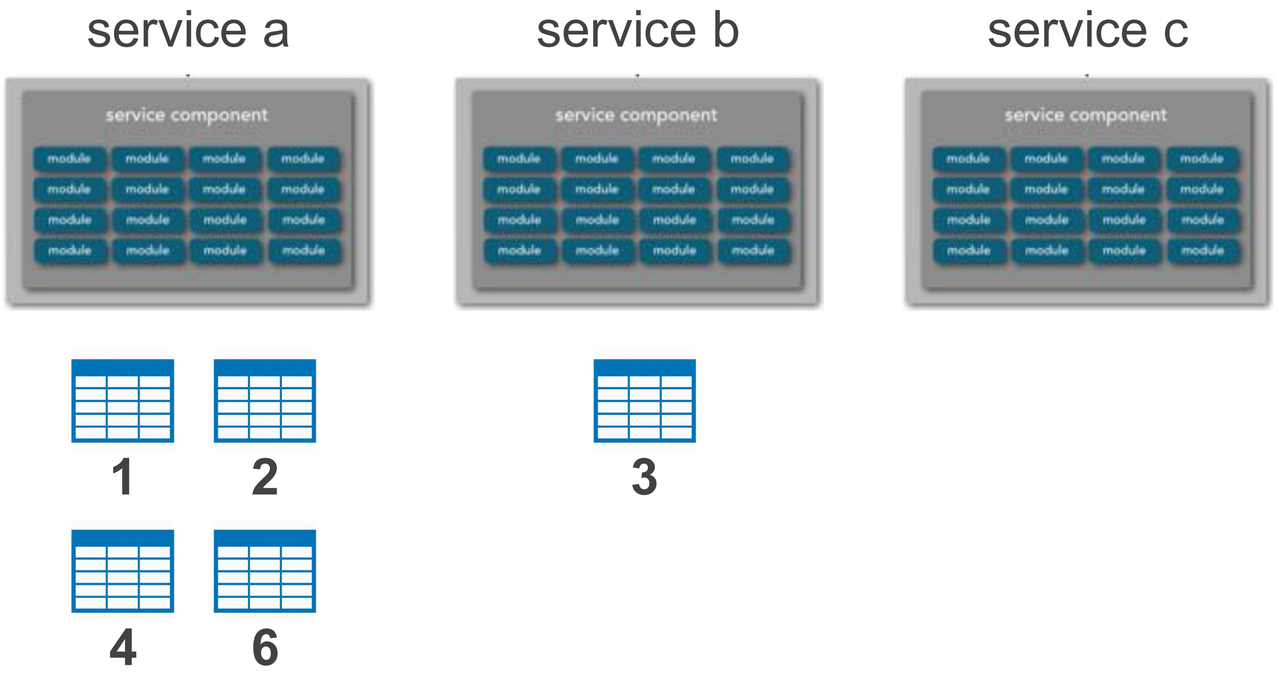




data dependencies

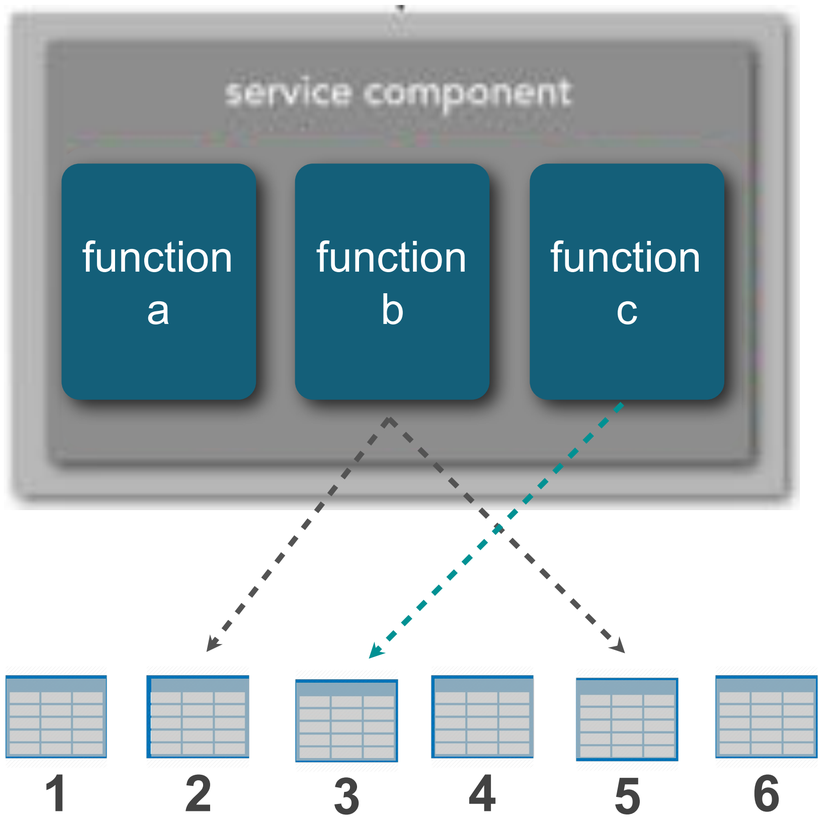
service granularity

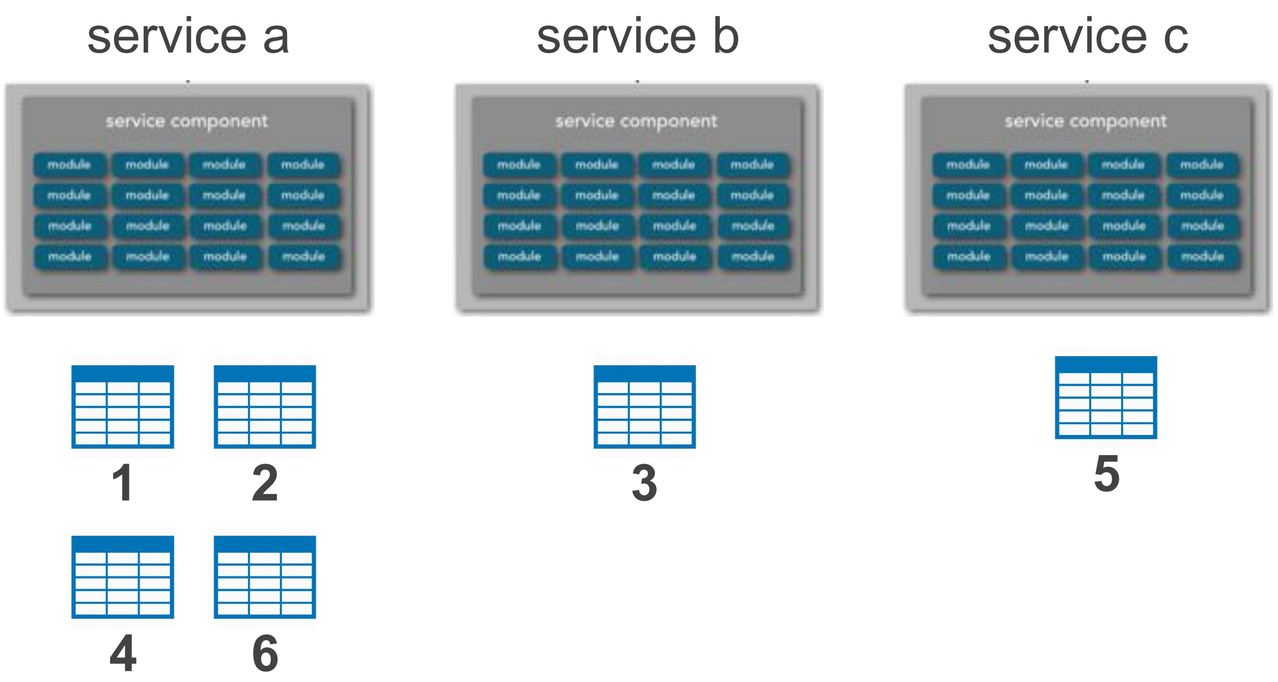




data dependencies

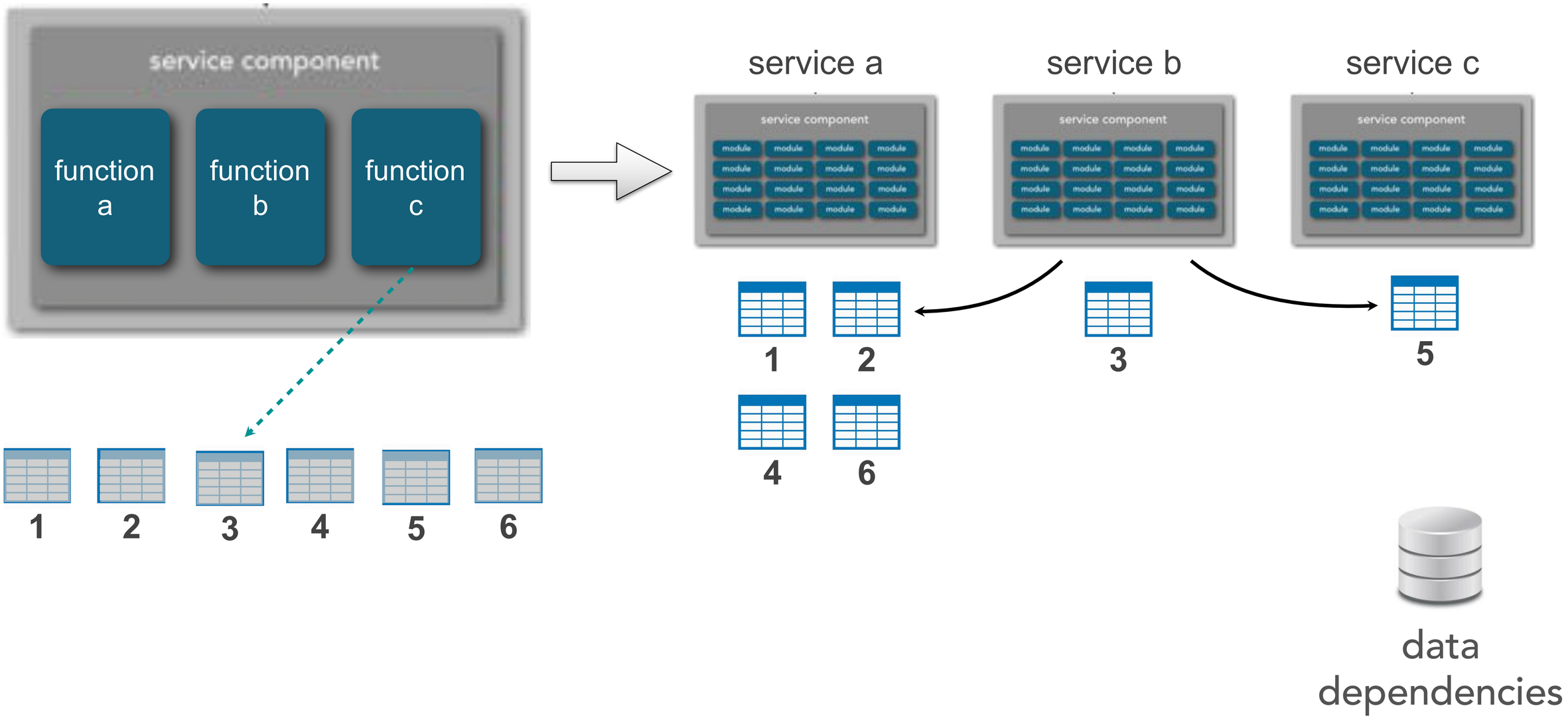
service granularity



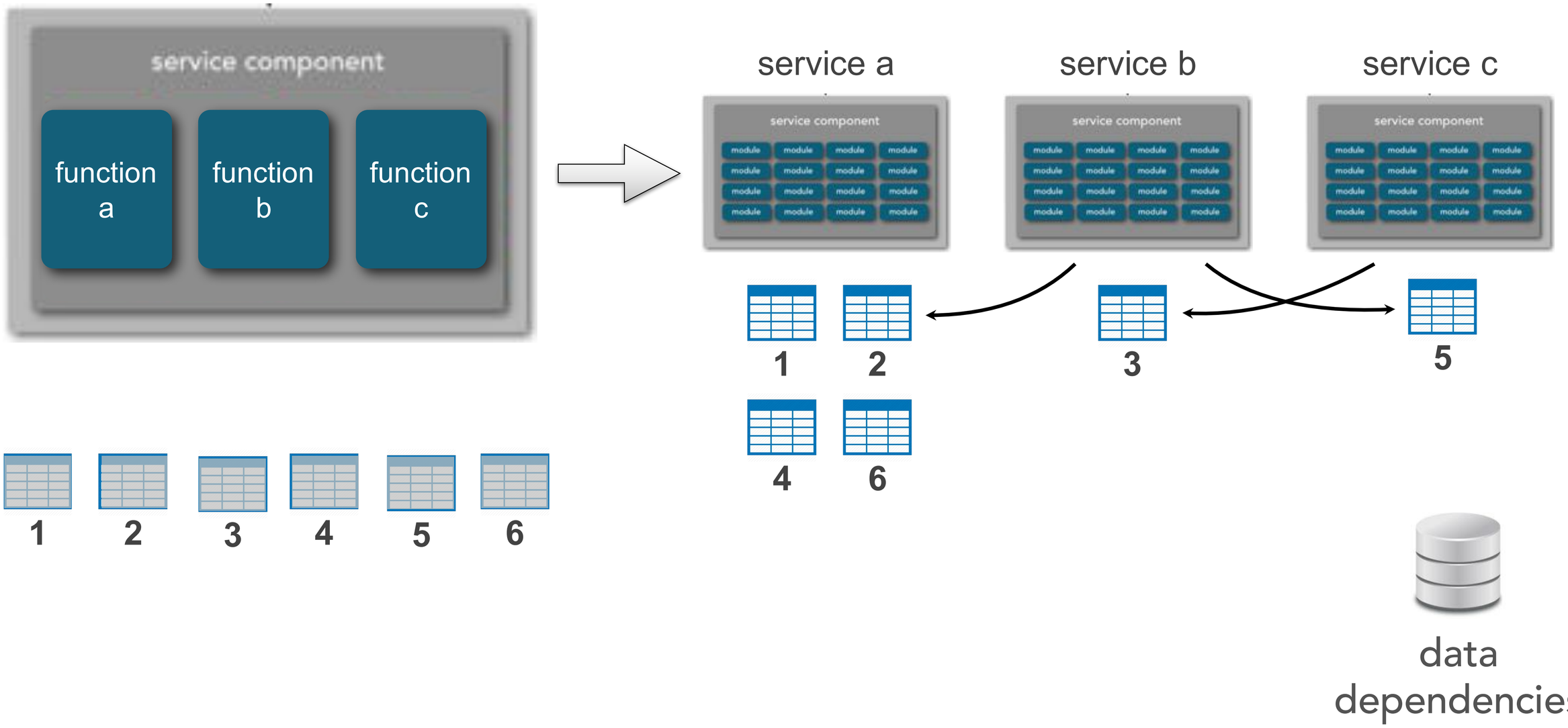


data dependencies

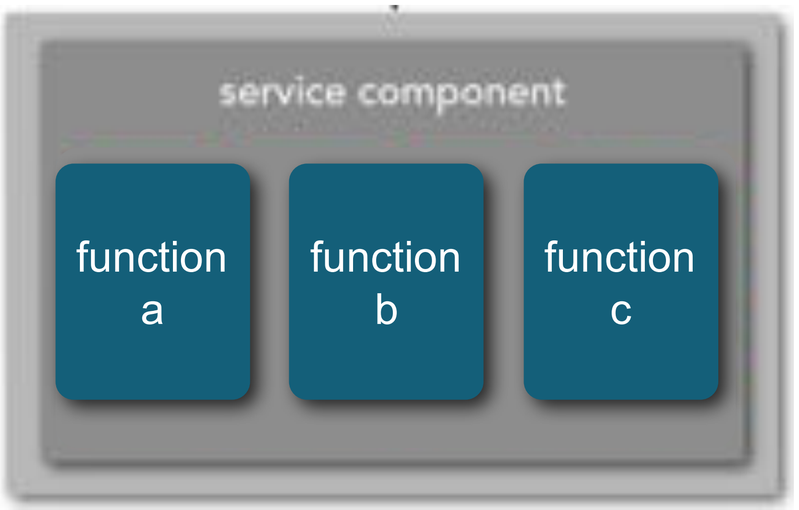
service granularity

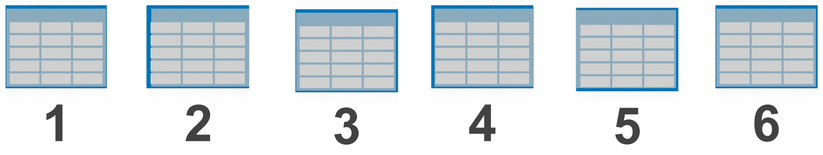


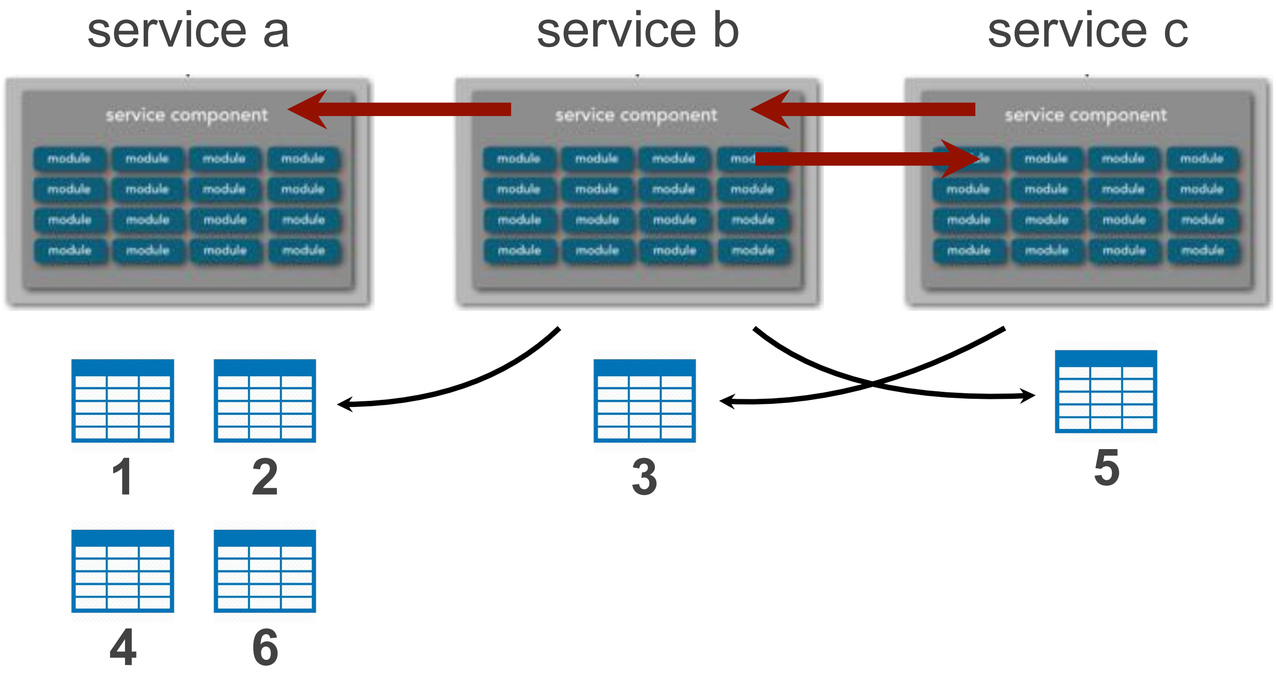
service granularity



service granularity

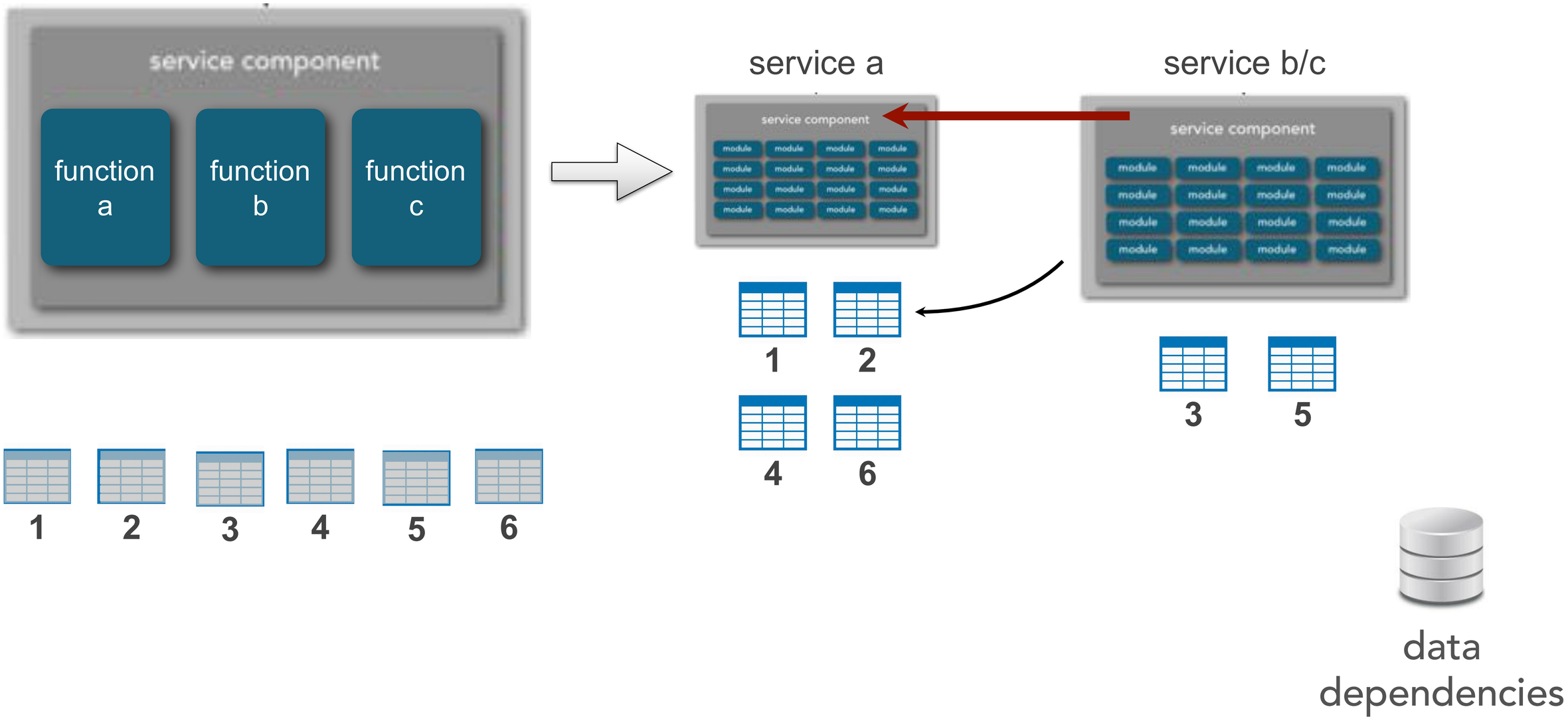




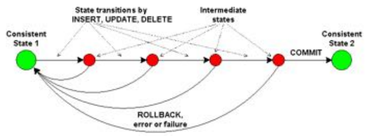


data dependencies

service granularity



service granularity



database
transactions

data
dependencies

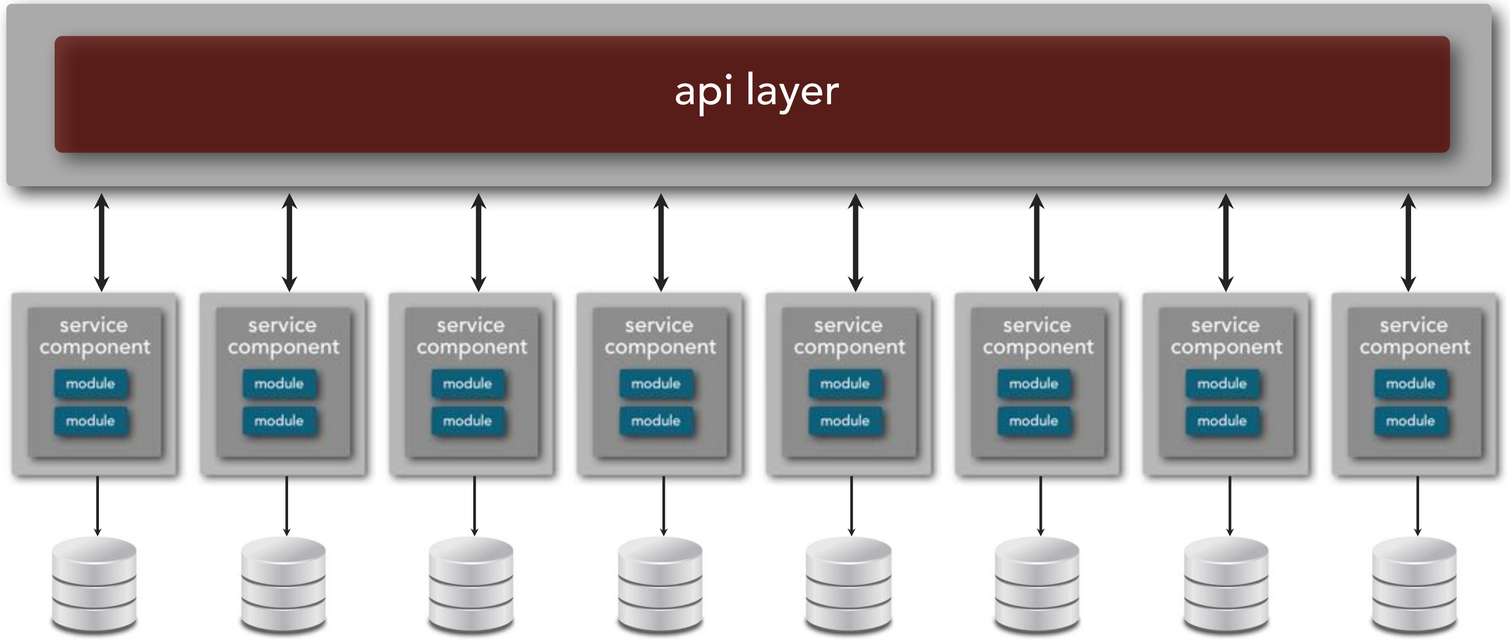
workflow and
choreography

granularity factors

"what factors influence service granularity?"

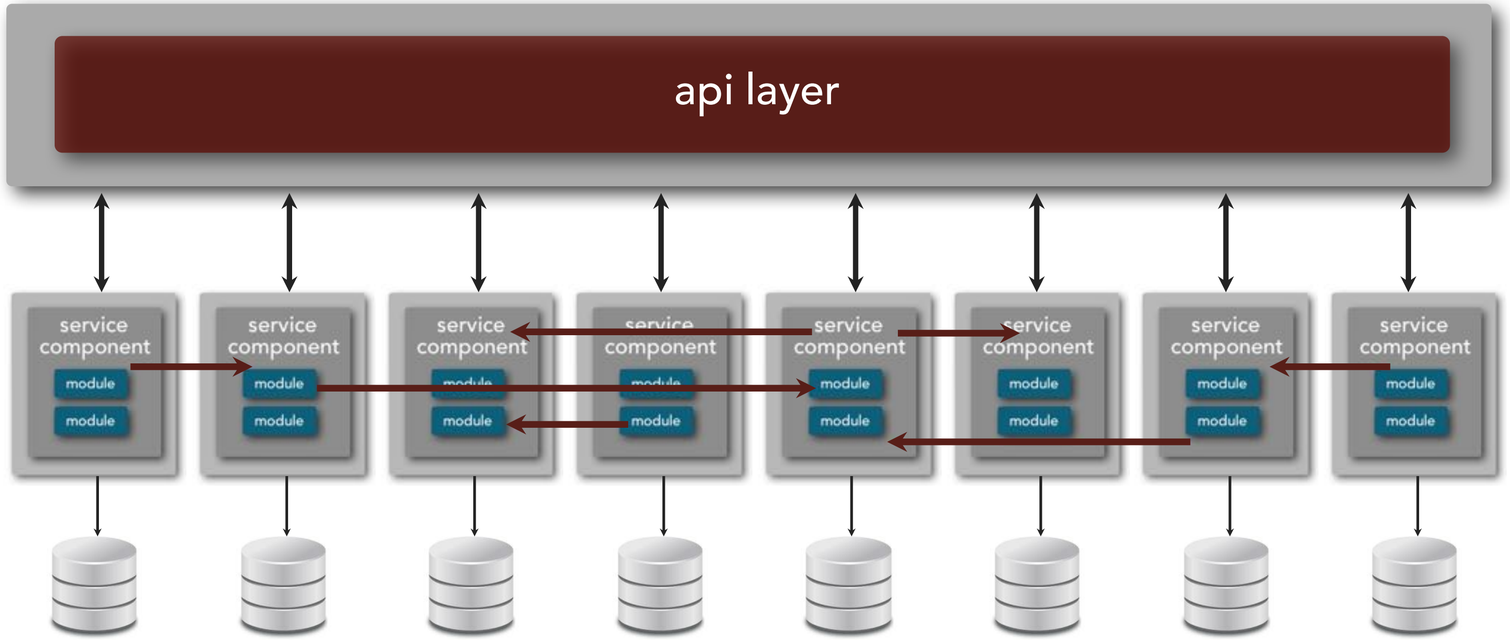
service granularity

workflow and choreography



service granularity

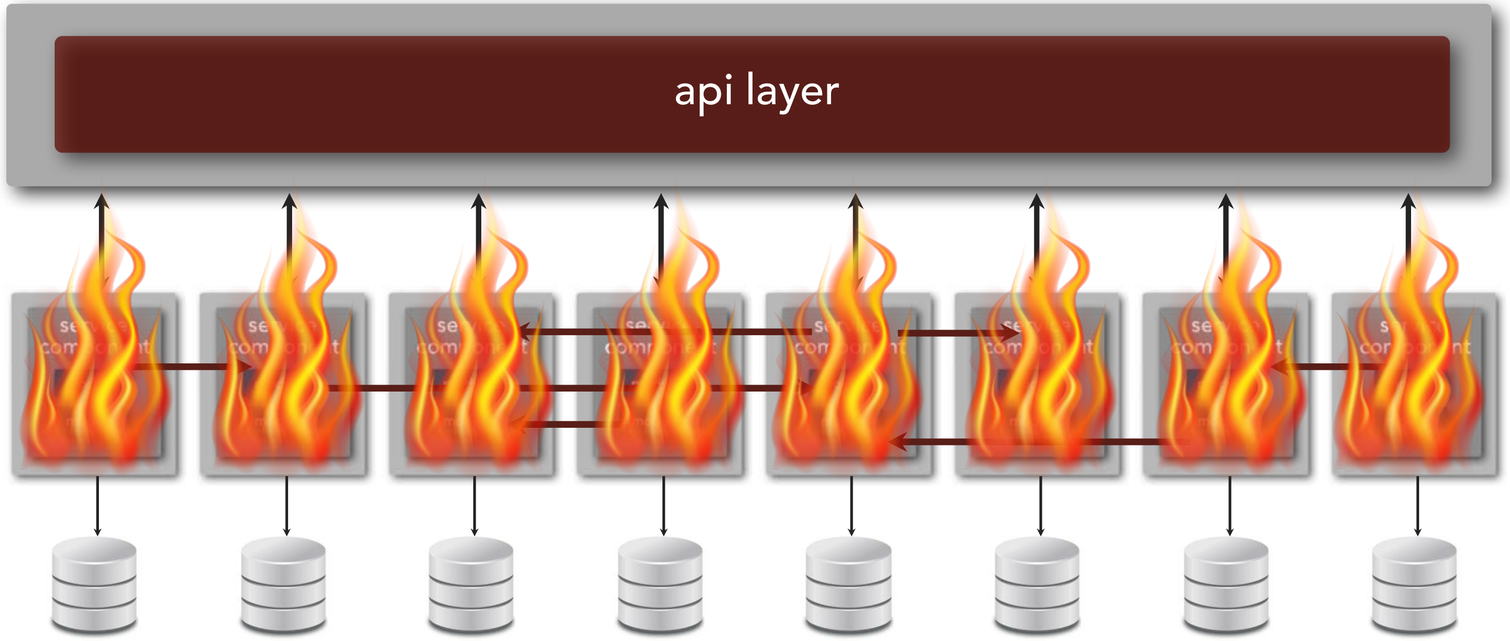
workflow and choreography



service granularity

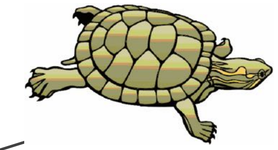
workflow and choreography

fault tolerance (availability)

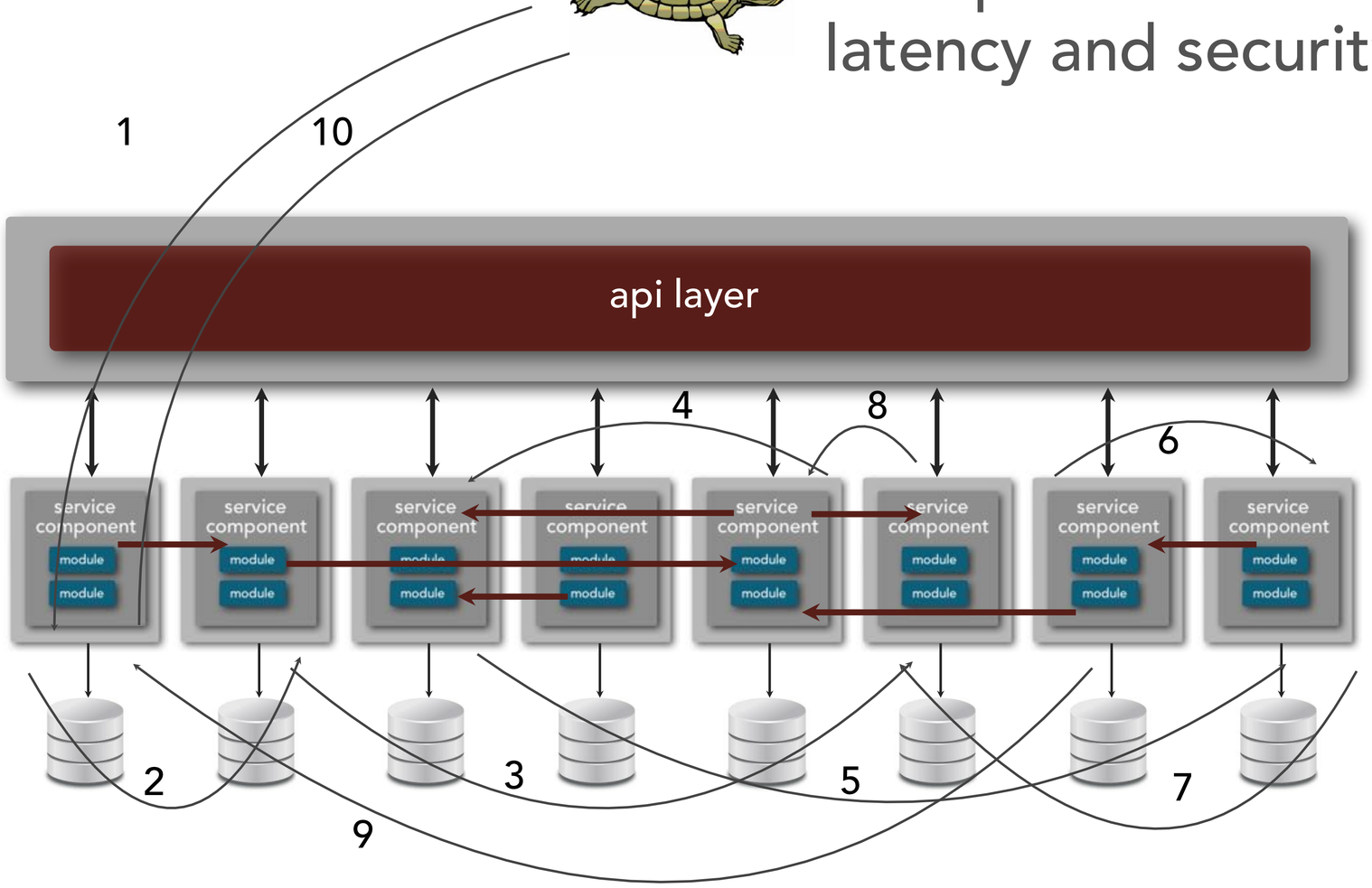


service granularity

workflow and choreography



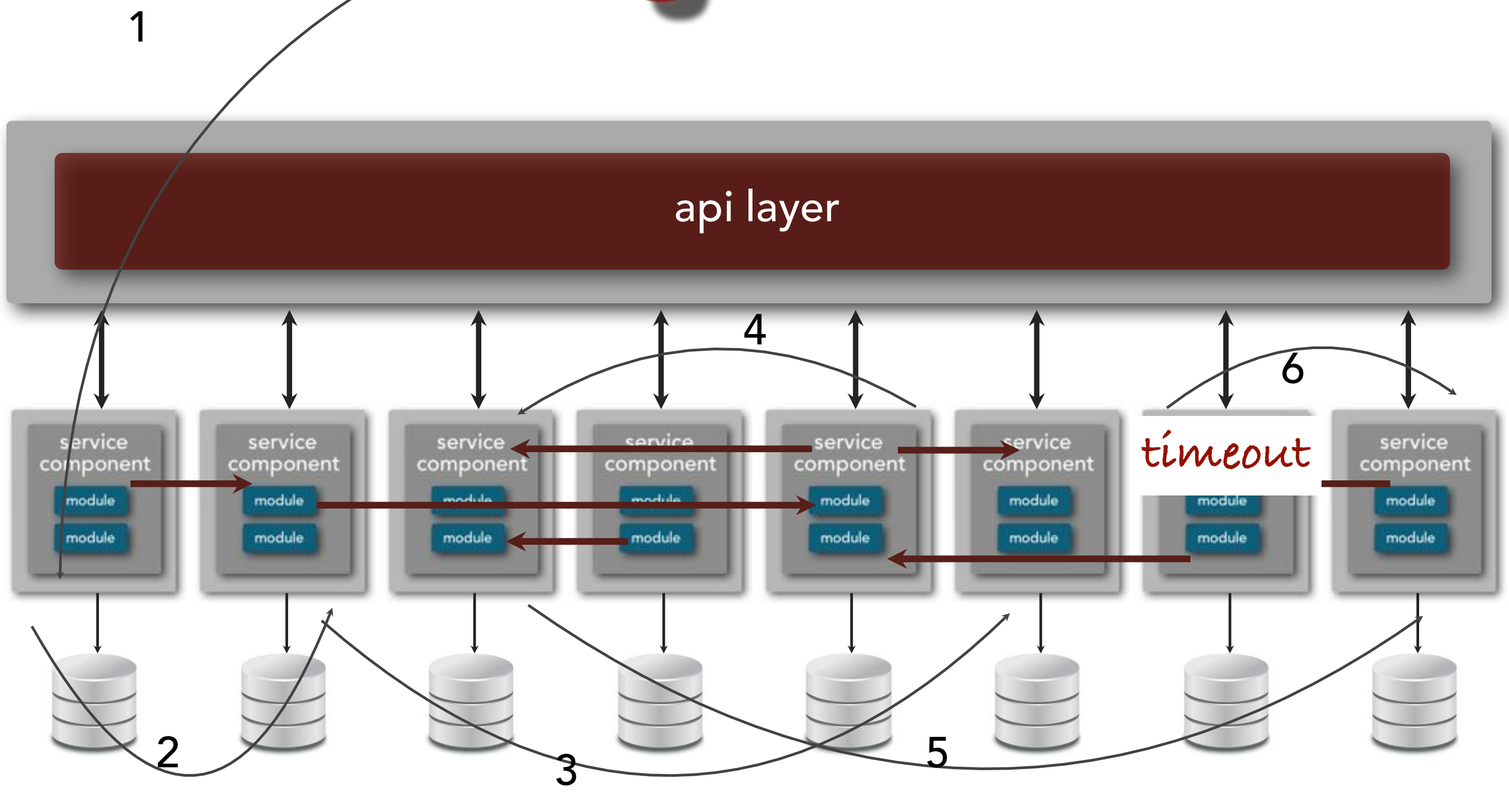
slow performance due to latency and security



service granularity

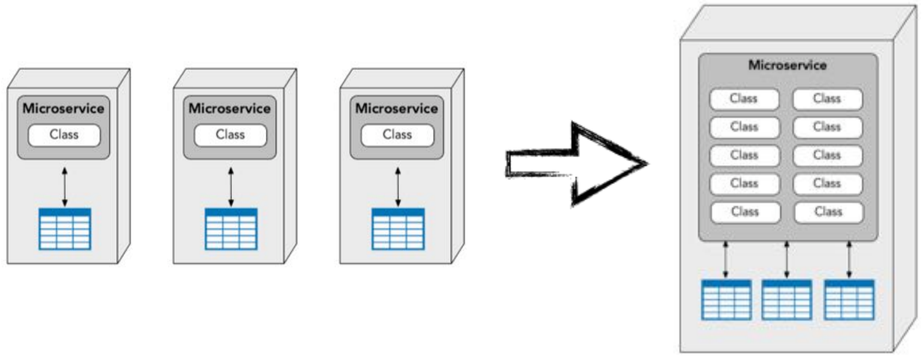
workflow and choreography

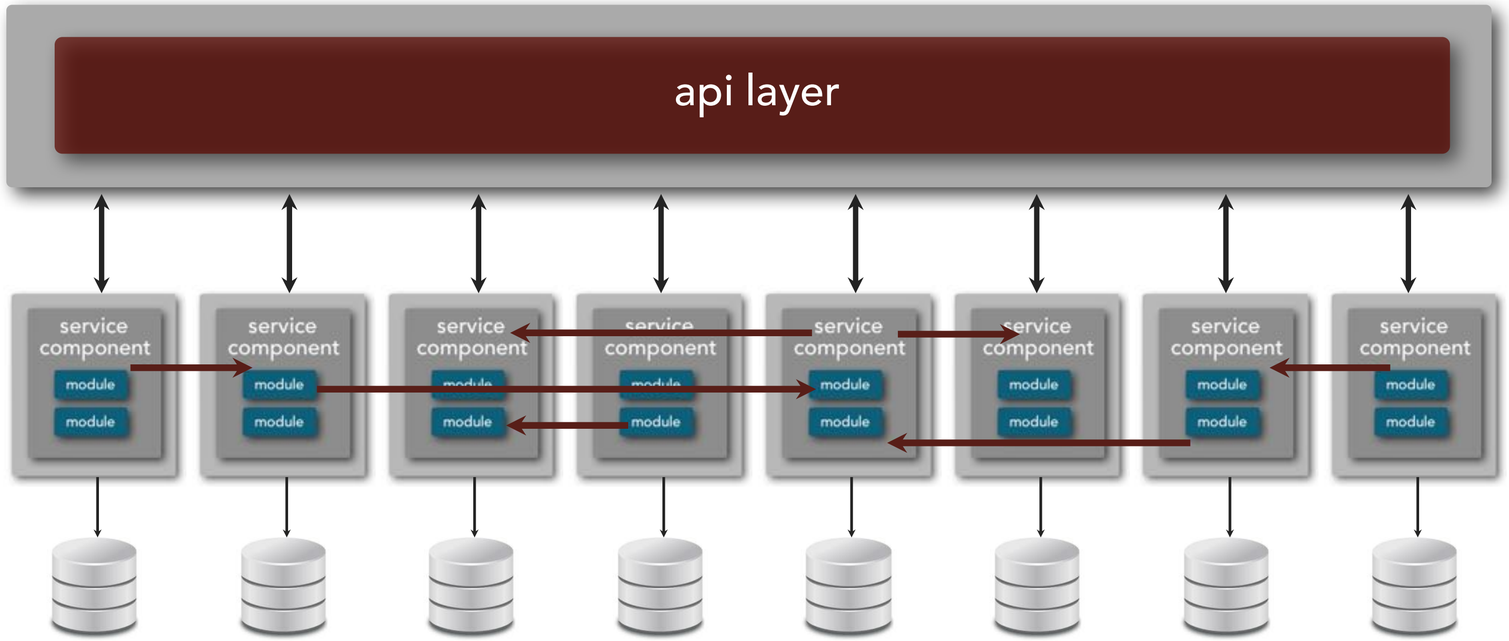
reliability and data consistency



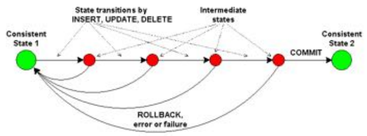
service granularity

workflow and choreography





service granularity



database
transactions

data
dependencies

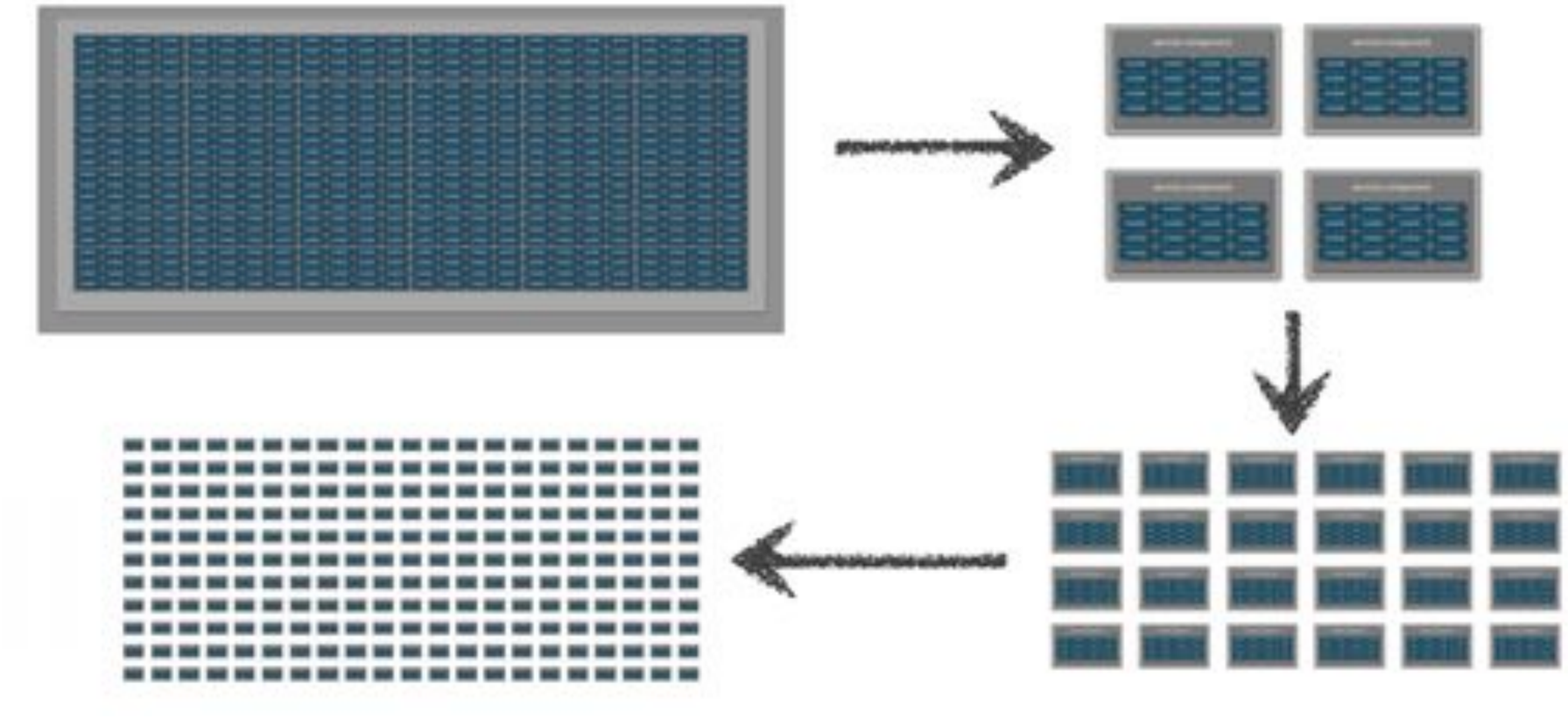
workflow and
choreography

granularity factors

"what factors influence service granularity?"

service granularity

get granularity right by iterating



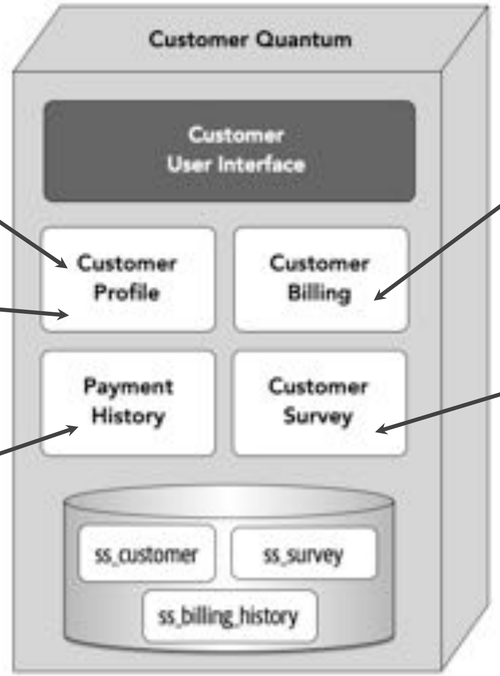
Kata Exercise - Service Granularity

Based on the following components below, identify the services for the customer functionality in the Sysops Squad application.

Customers can register for the service

Customers can view and edit their profile

Customers can view their payment history



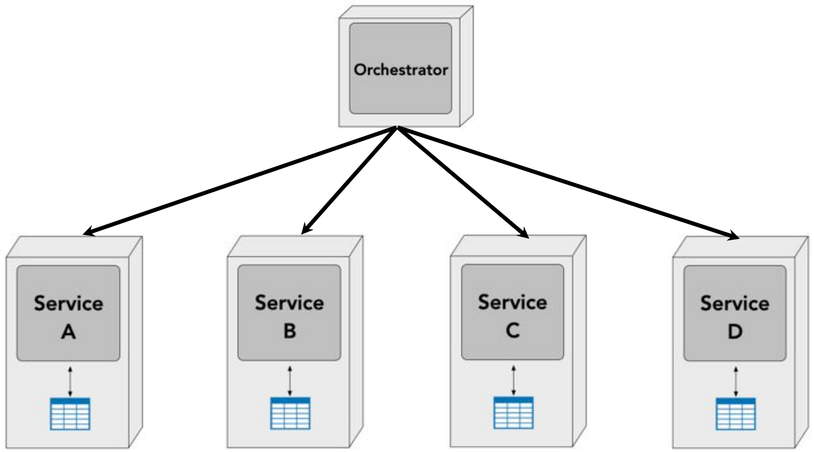
Customers are automatically charged annually for the service based on a credit card stored as part of their profile

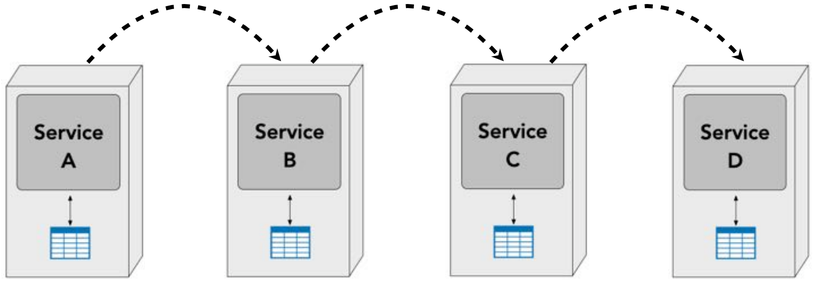
Once a ticket is complete the customer can go online to fill out a survey

How do I handle workflow and transactions in a distributed architecture?

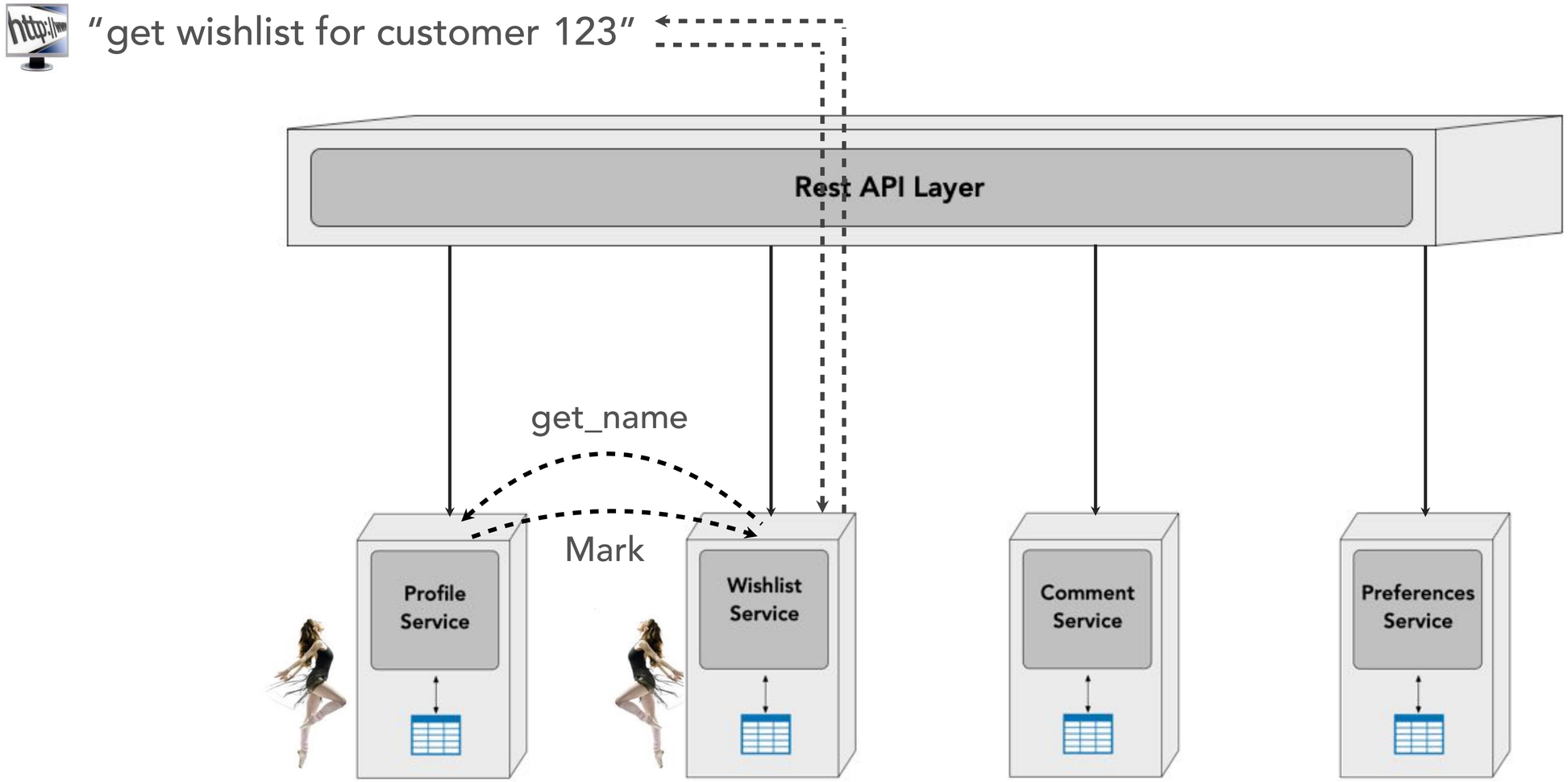
orchestration and workflow

"when should I consider using orchestration vs. choreography?"

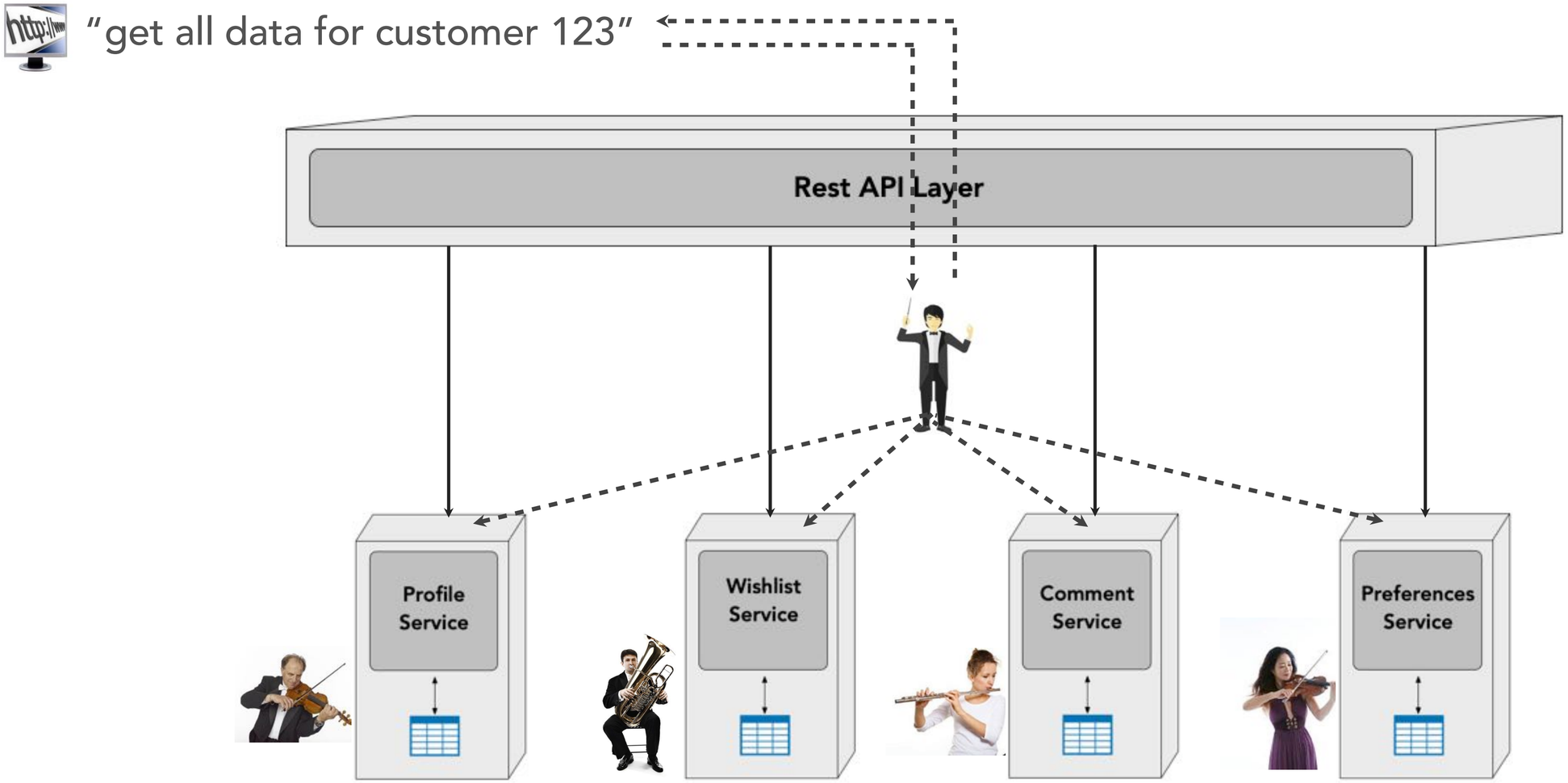




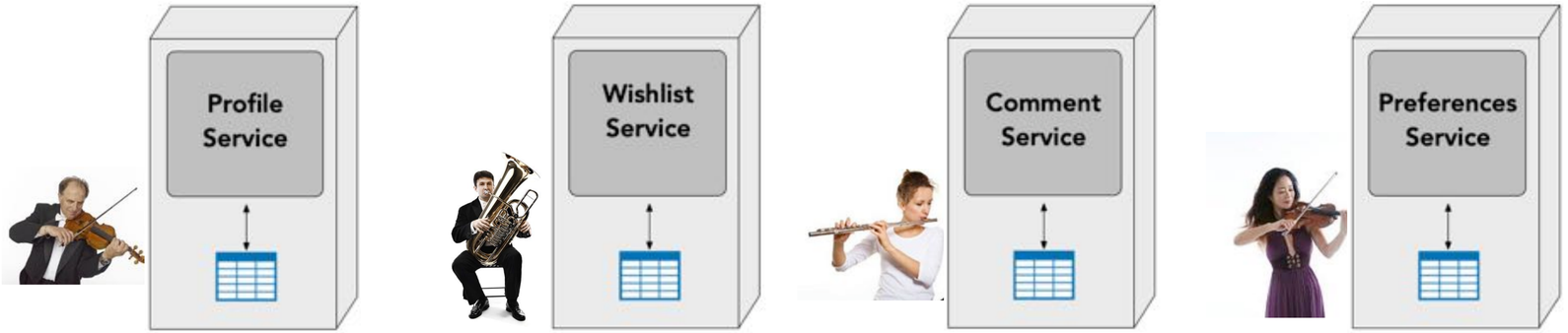
orchestration vs. choreography



orchestration vs. choreography

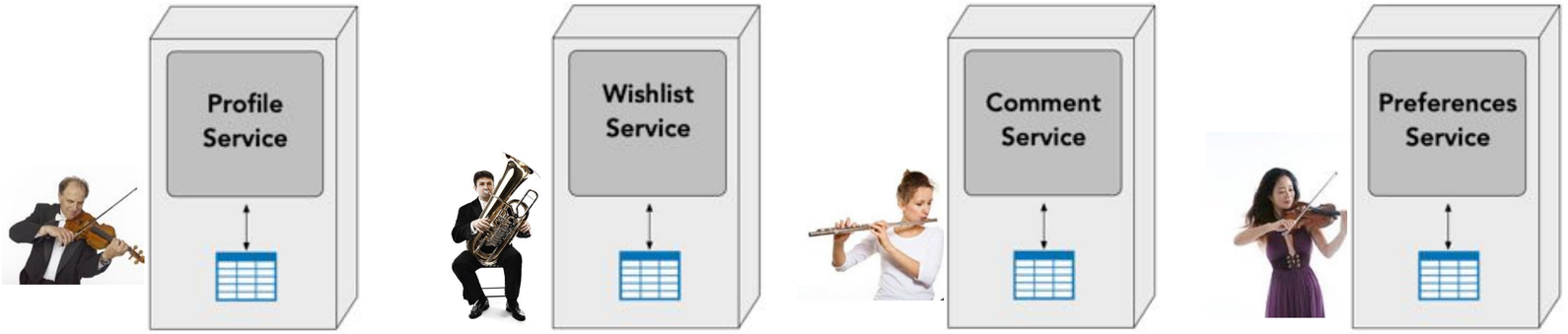


orchestration vs. choreography



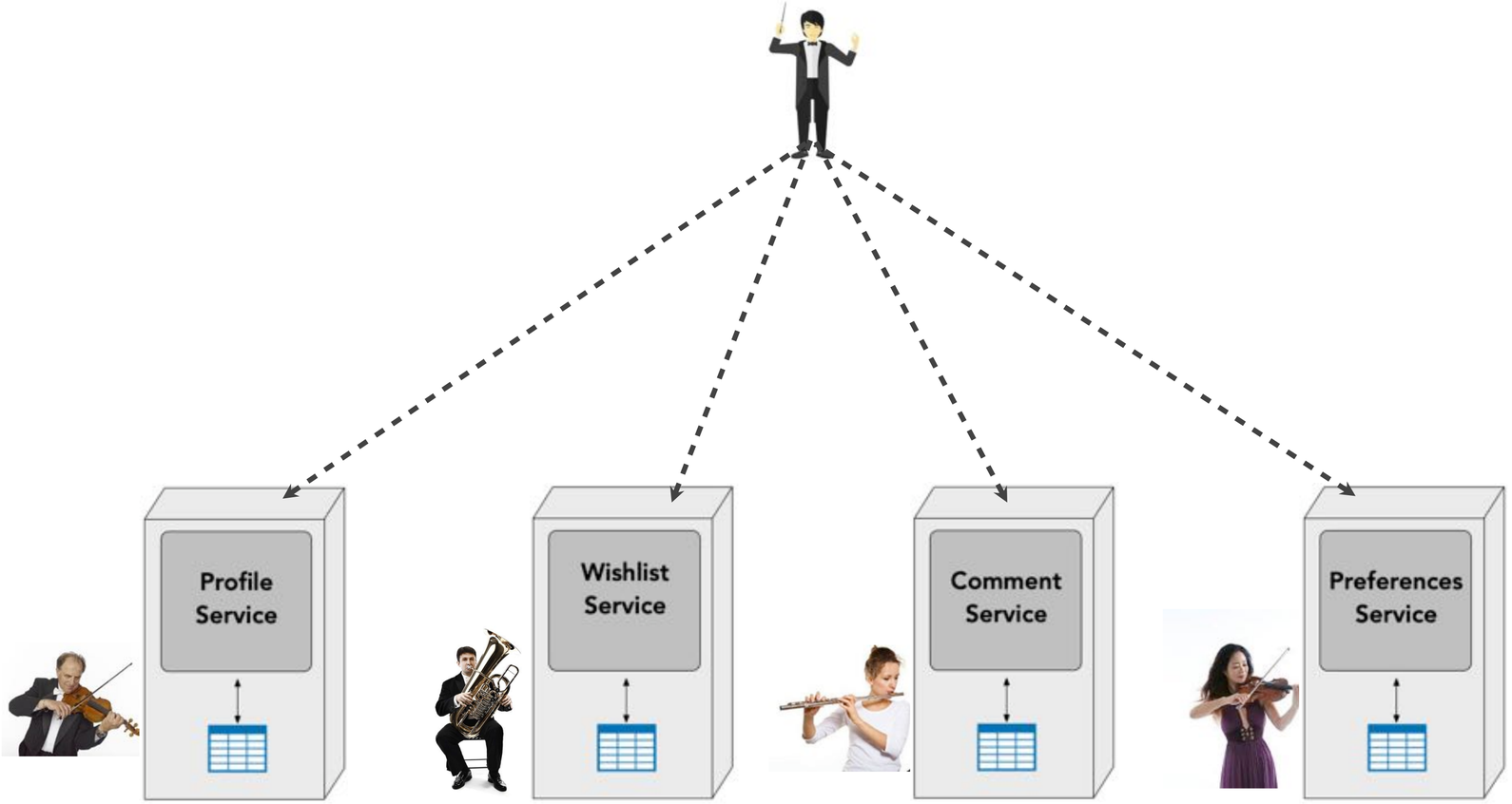
orchestration vs. choreography

1. multicasting logic



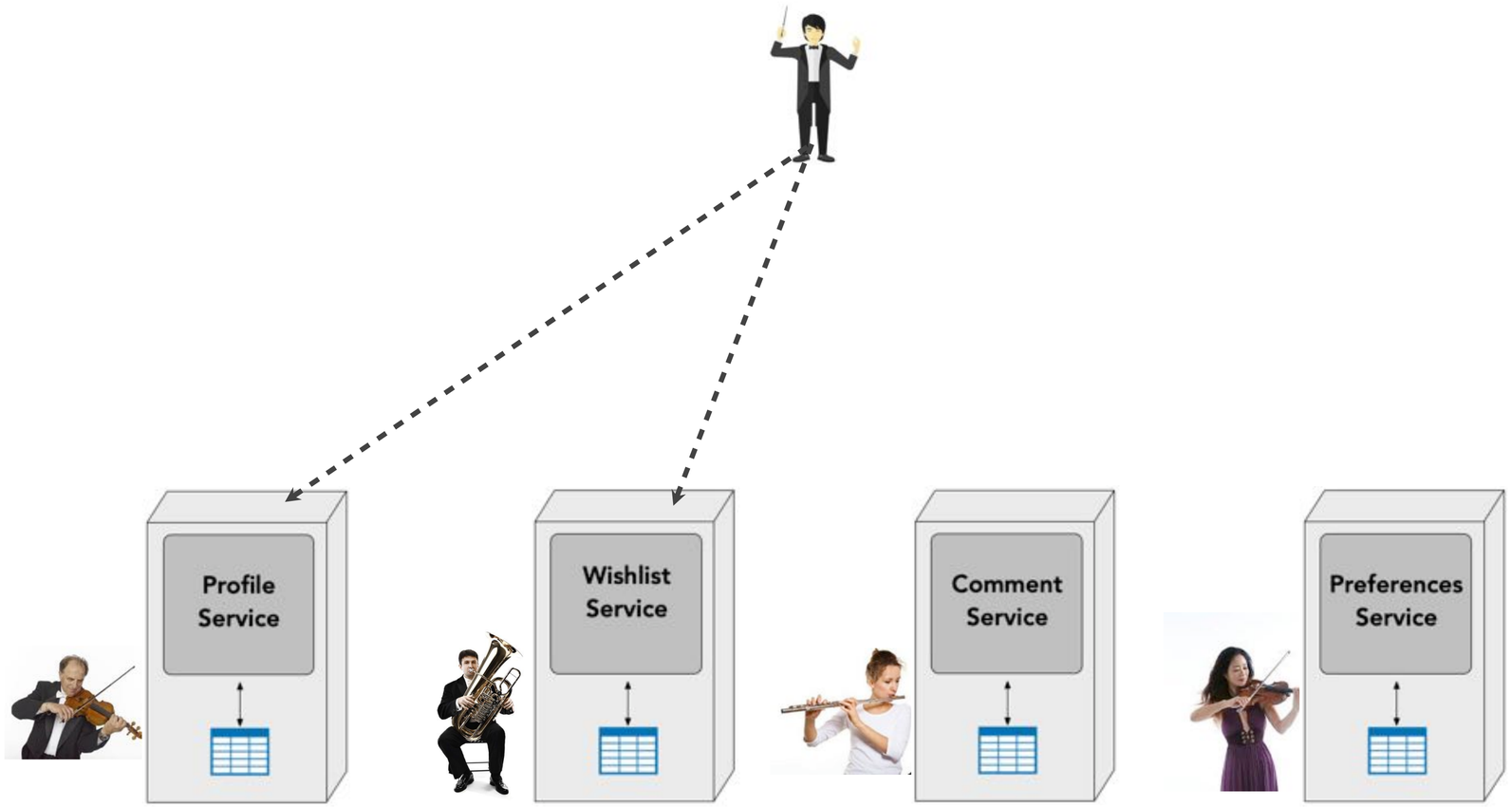
orchestration vs. choreography

1. multicasting logic



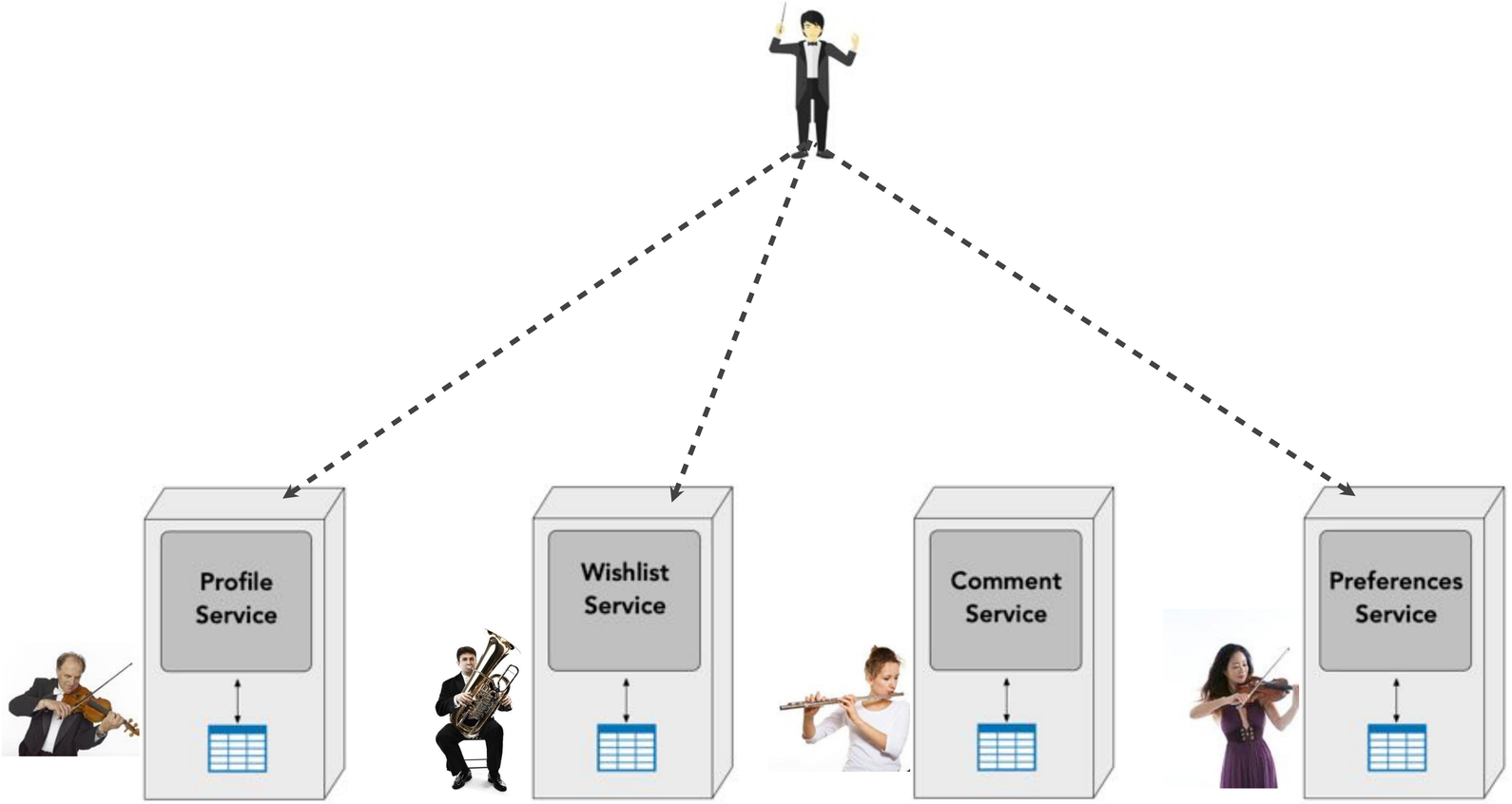
orchestration vs. choreography

1. multicasting logic



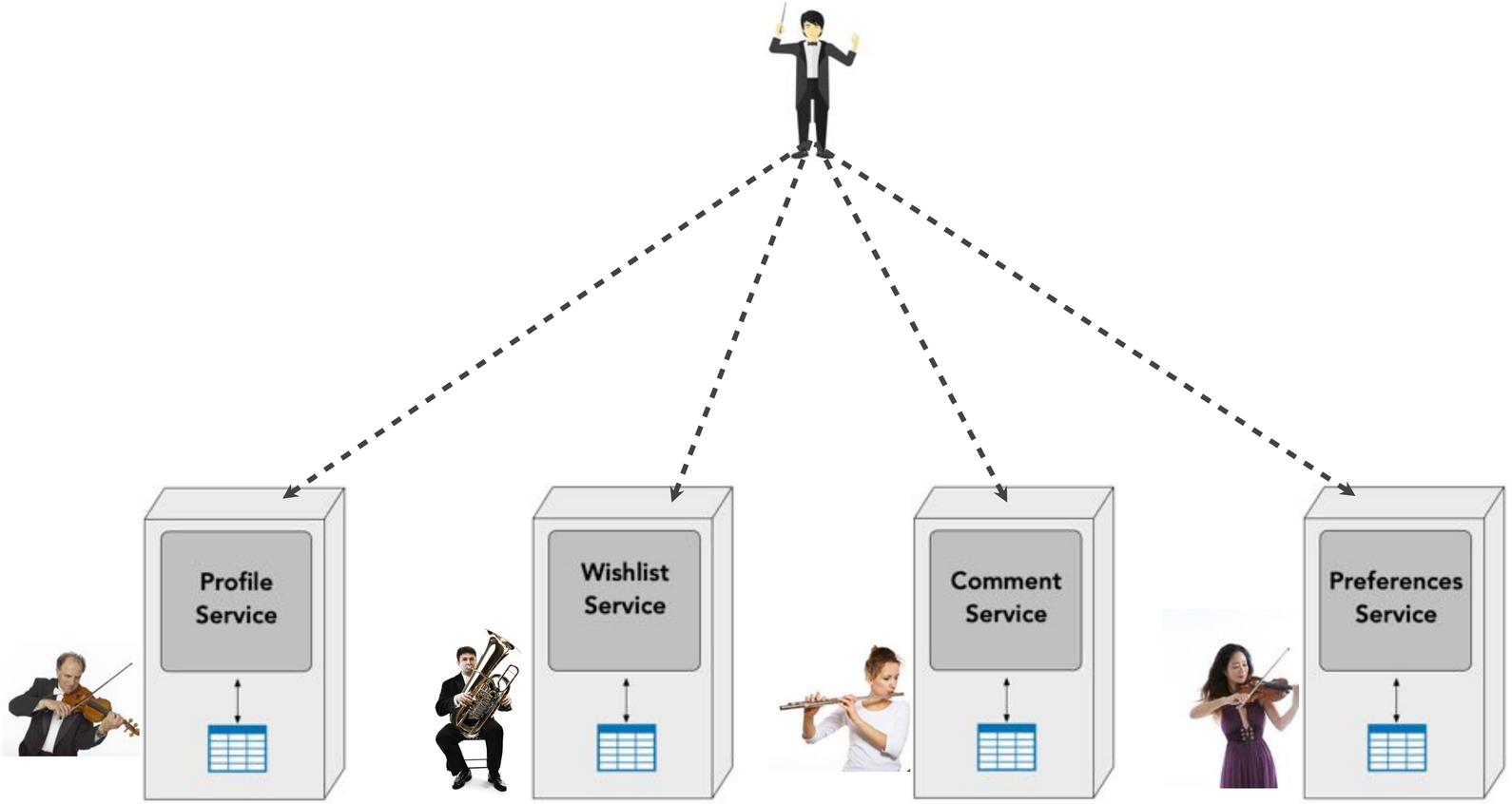
orchestration vs. choreography

1. multicasting logic



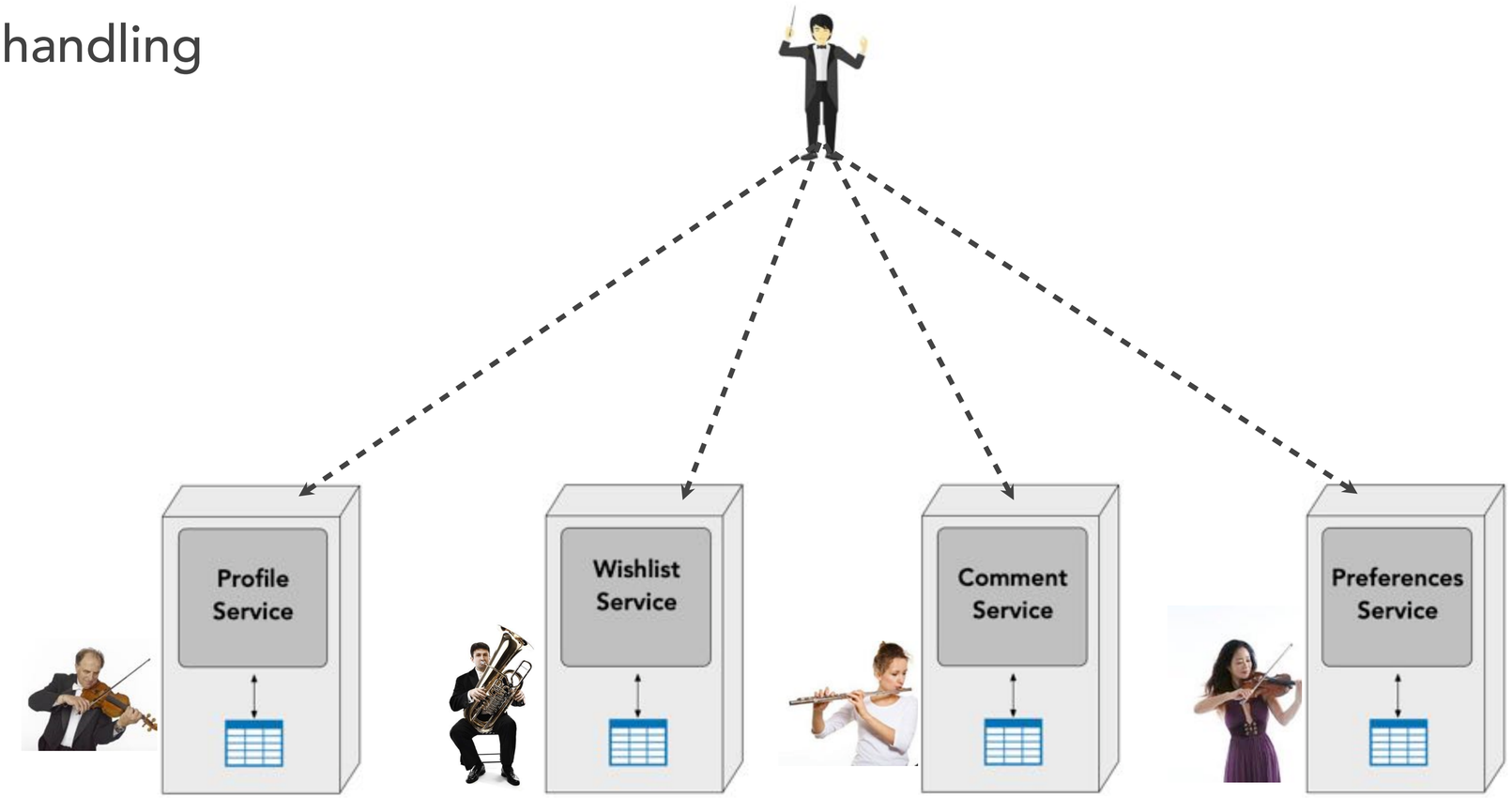
orchestration vs. choreography

1. multicasting logic



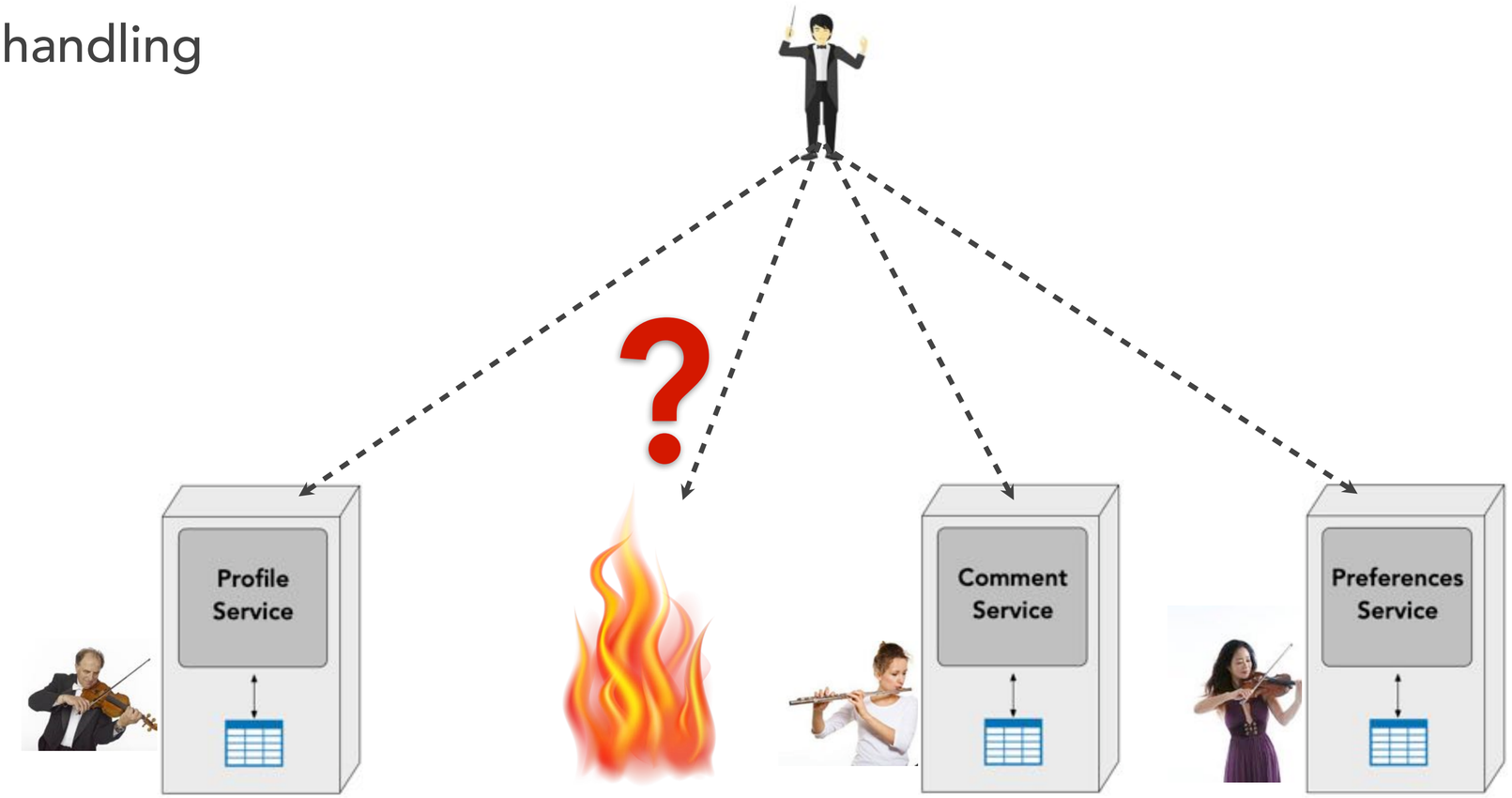
orchestration vs. choreography

1. multicasting logic
2. error handling



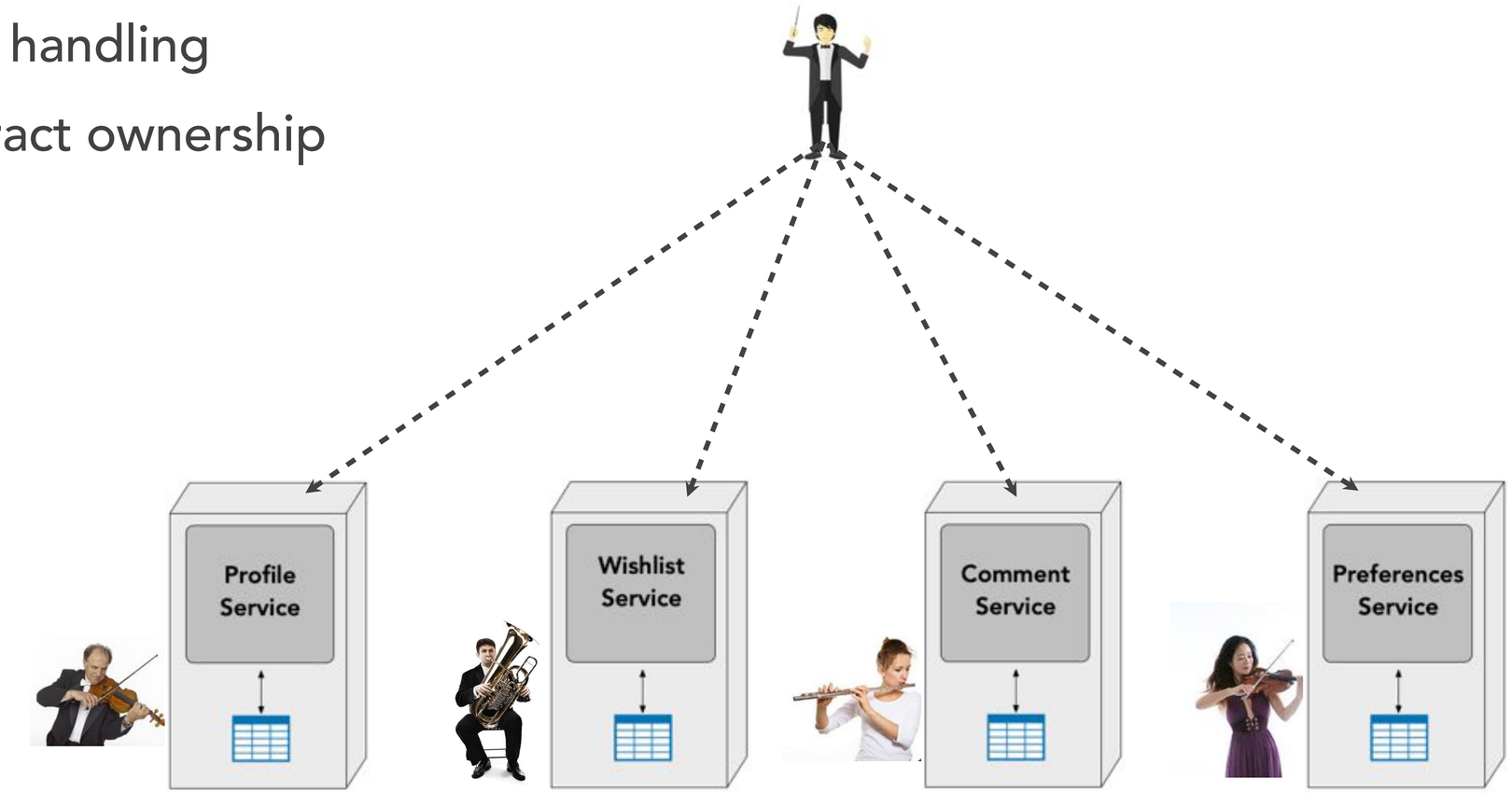
orchestration vs. choreography

1. multicasting logic
2. error handling



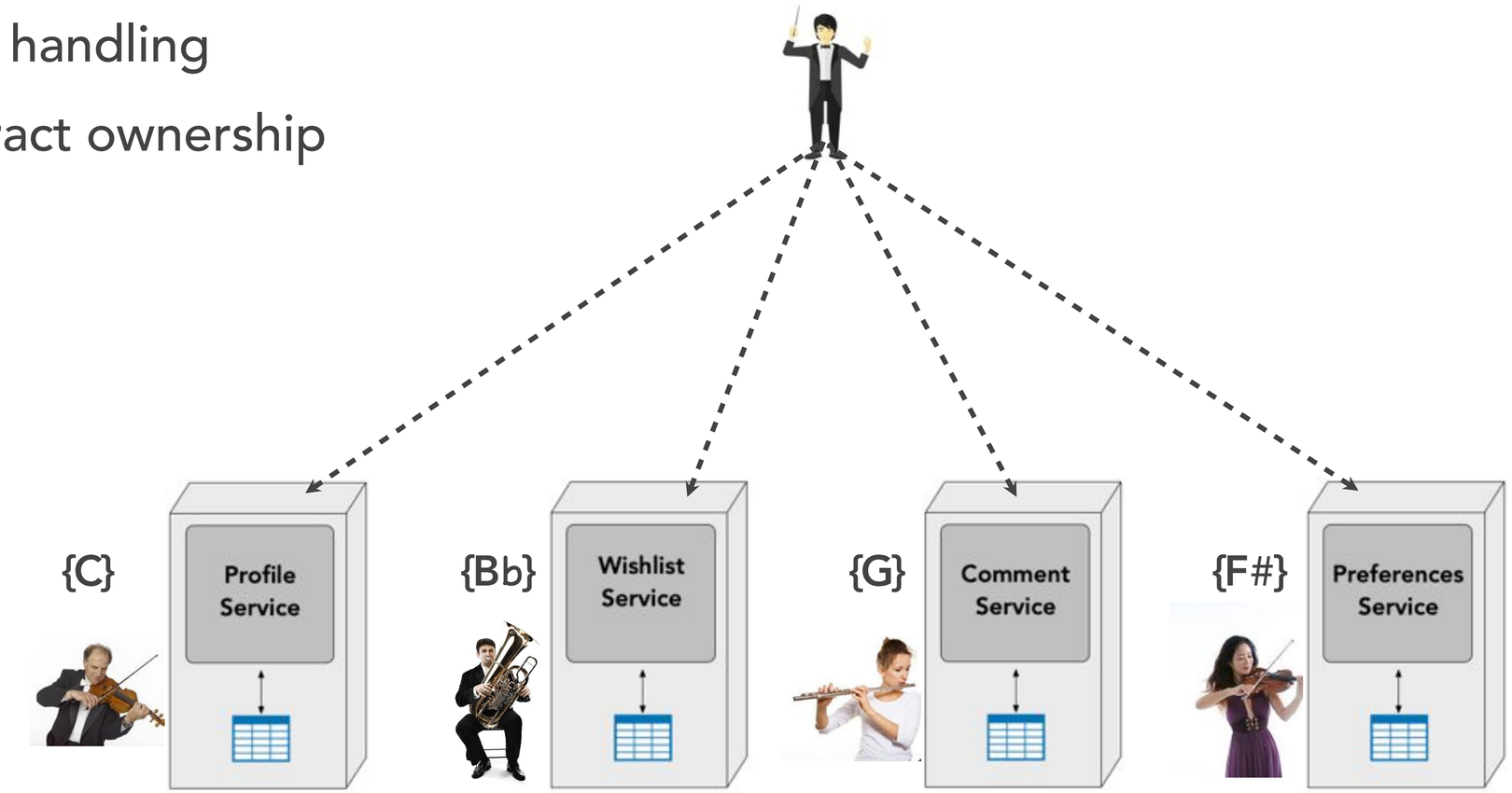
orchestration vs. choreography

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



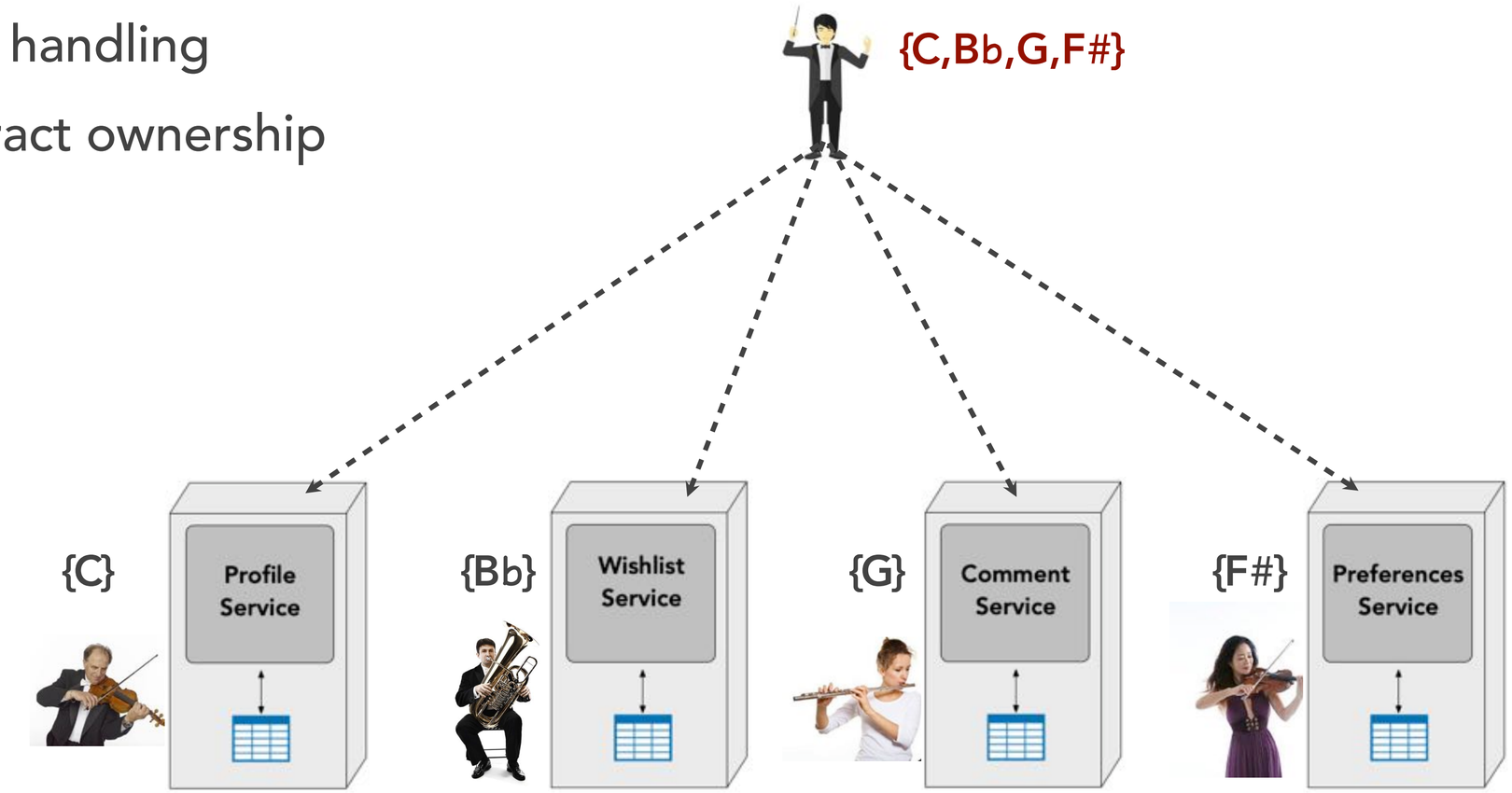
orchestration vs. choreography

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



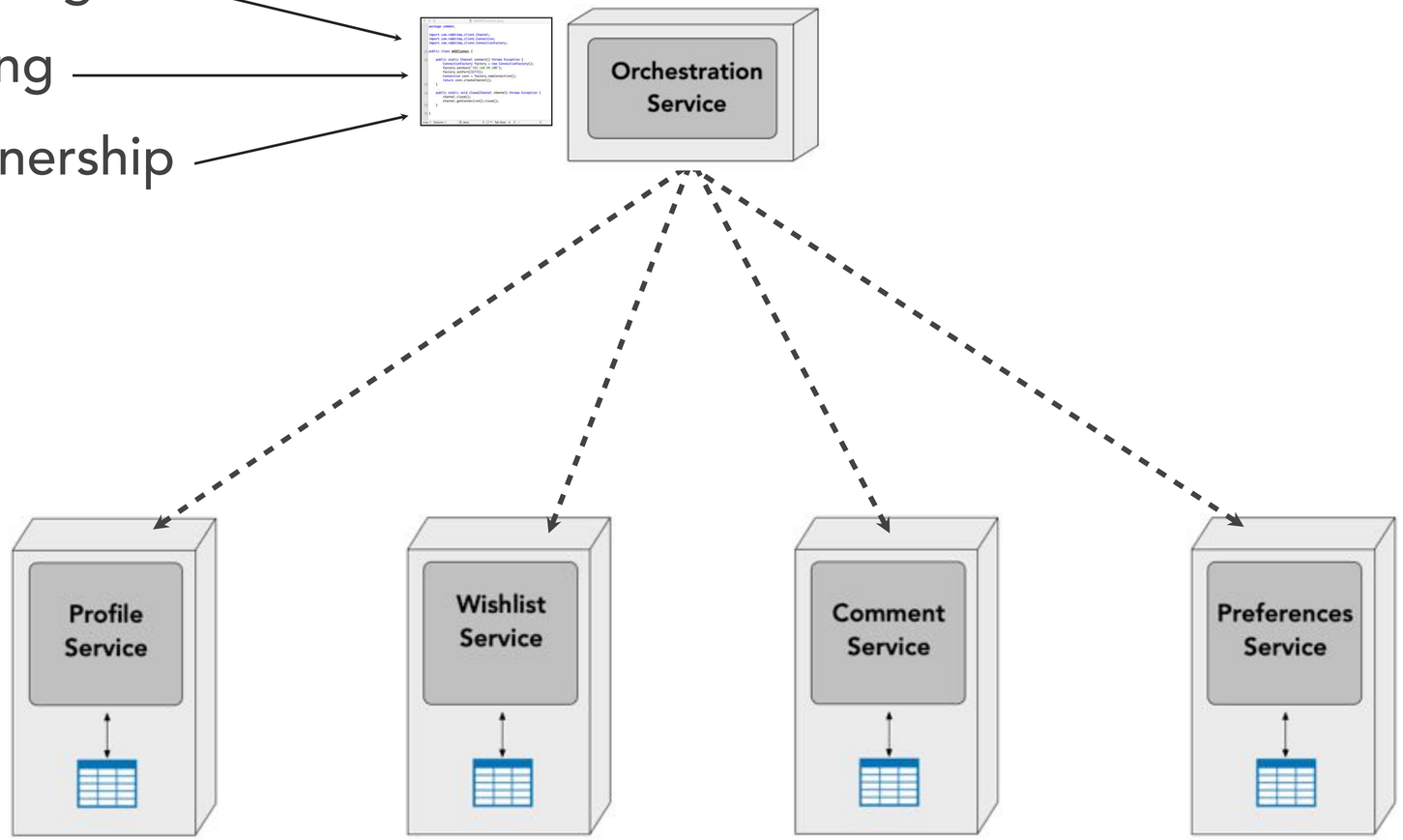
orchestration vs. choreography

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



orchestration vs. choreography

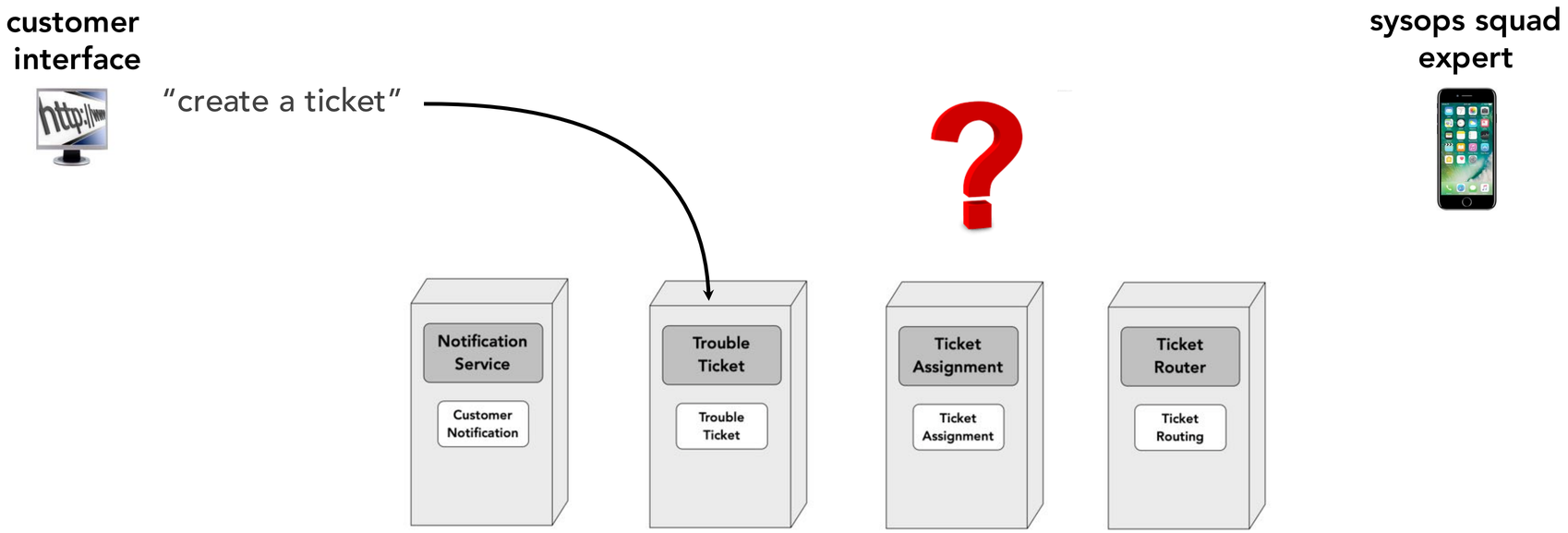
1. multicasting logic
2. error handling
3. contract ownership



Kata Exercise - Communication and Workflow

Identify the communication flow between the 4 services below. Orchestration? Choreography? Async? Sync? The flow for a trouble ticket is as follows:

1. Customers create a trouble ticket through the ***Ticket Management*** service and receive a ticket number.
2. The ***Ticket Assignment*** service finds the right sysops expert for the trouble ticket.
3. The ***Ticket Routing*** service routes the trouble ticket to the systems experts mobile device.
4. The customer is notified via SMS or email that the sysops expert is on their way to fix the problem.
5. The expert fixes the issue and marks the ticket complete in the mobile app, which is send to the system.



How do I automate architecture governance?

Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.

Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.

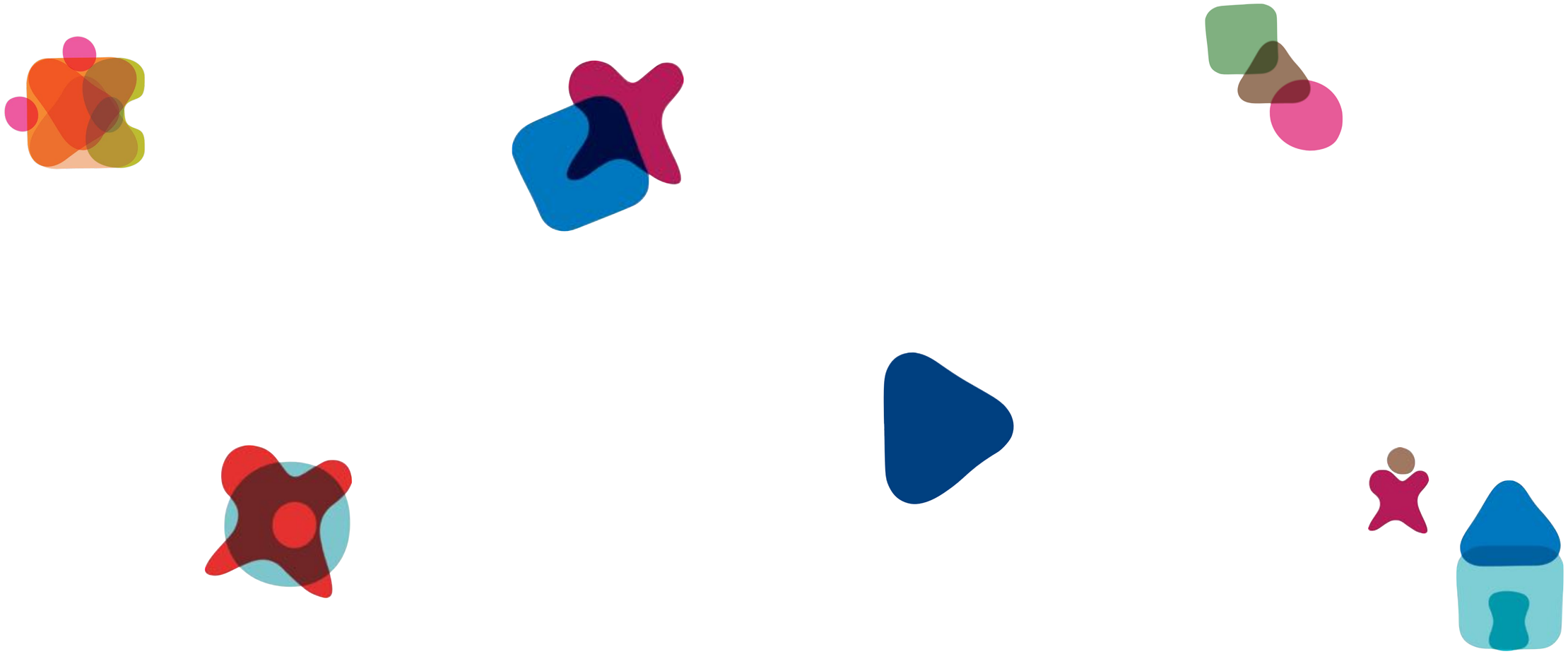
guided

evolutionary computing fitness function:

a particular type of objective function that is used to summarize…how close a given design solution is to achieving the set aims.

traveling salesperson problem

traveling salesperson problem



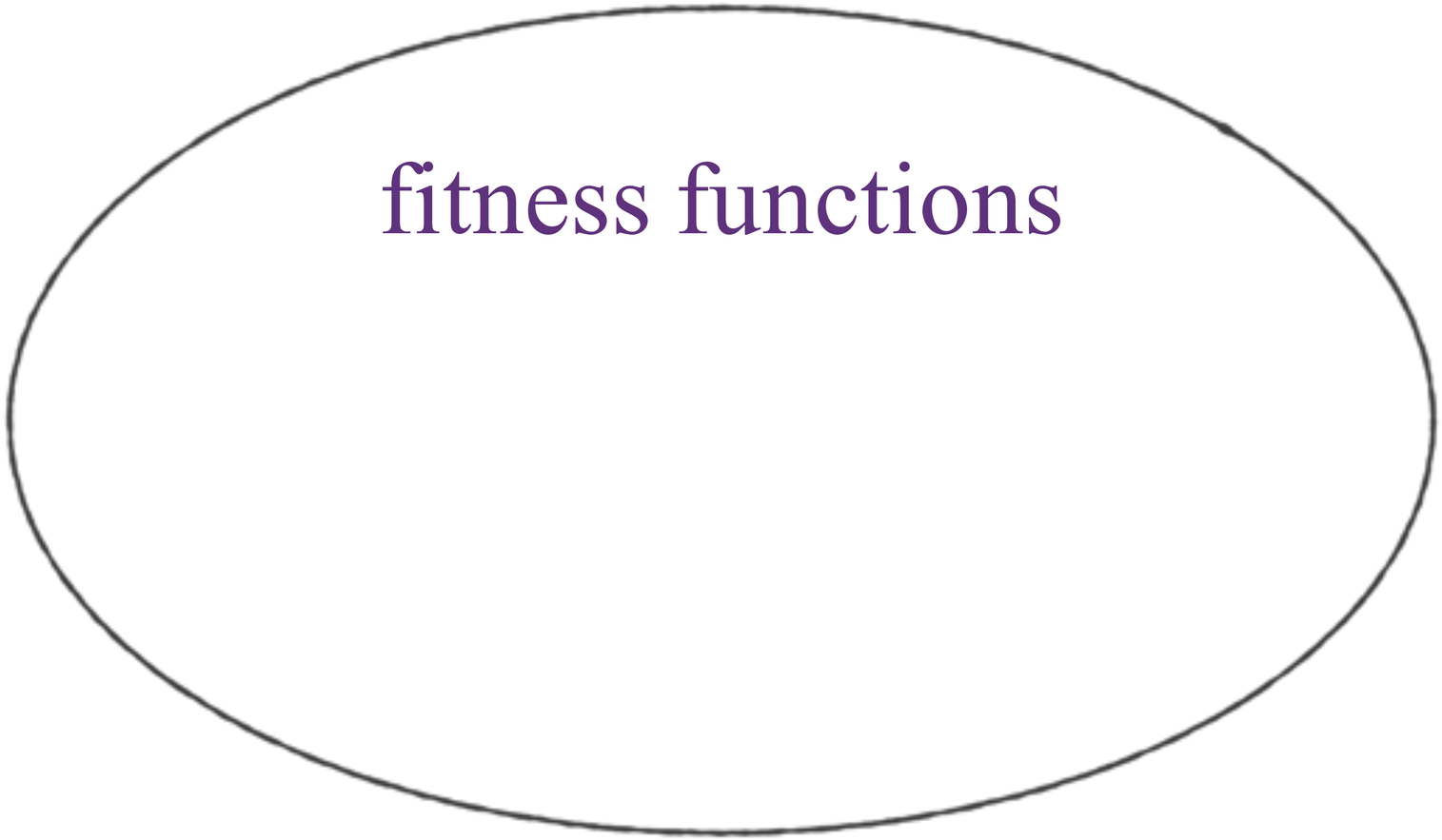
fitness function = length of route

guided

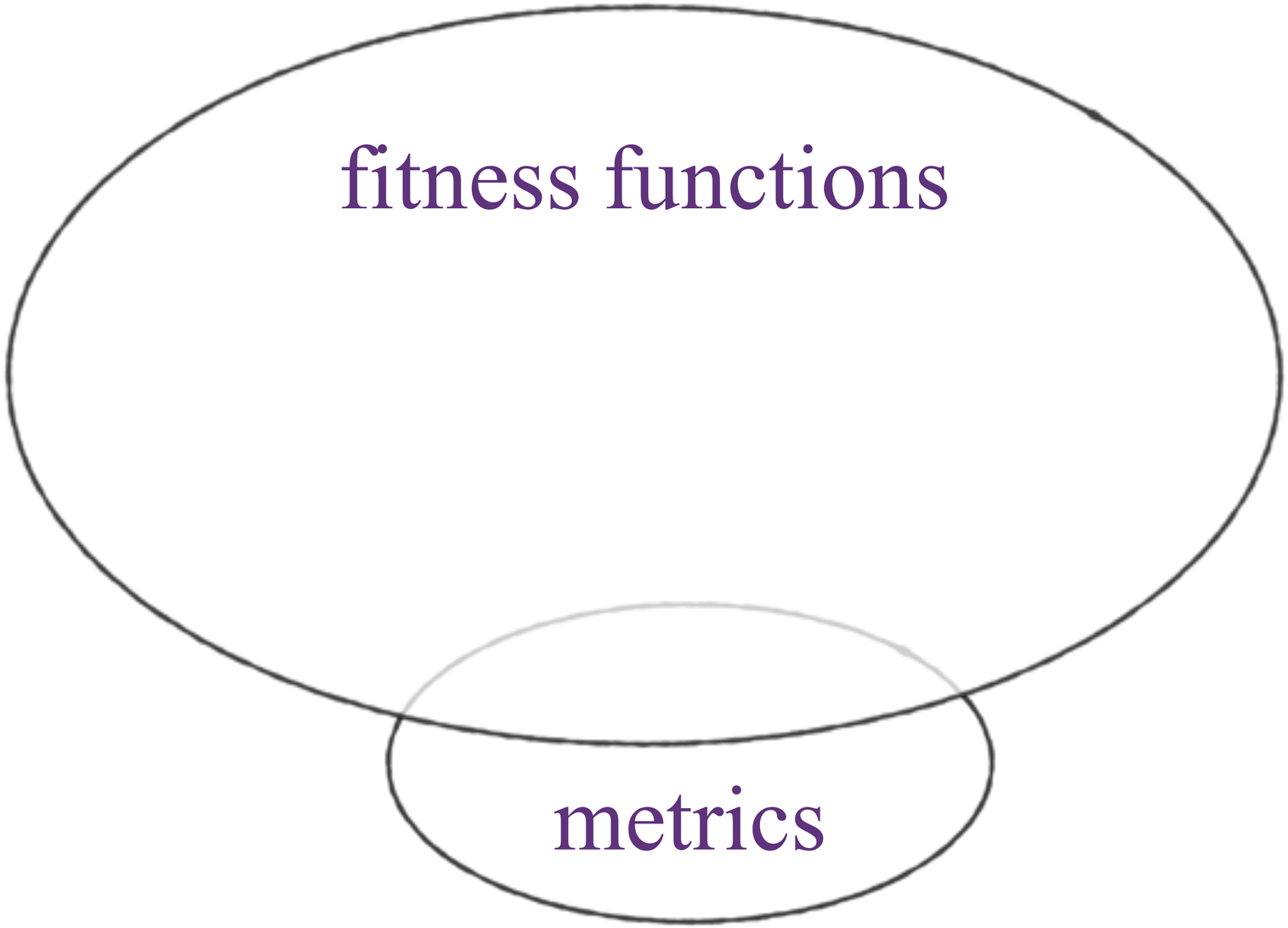
***architectural** fitness function*:

any mechanism that provides an objective integrity assessment of some architectural characteristic(s).

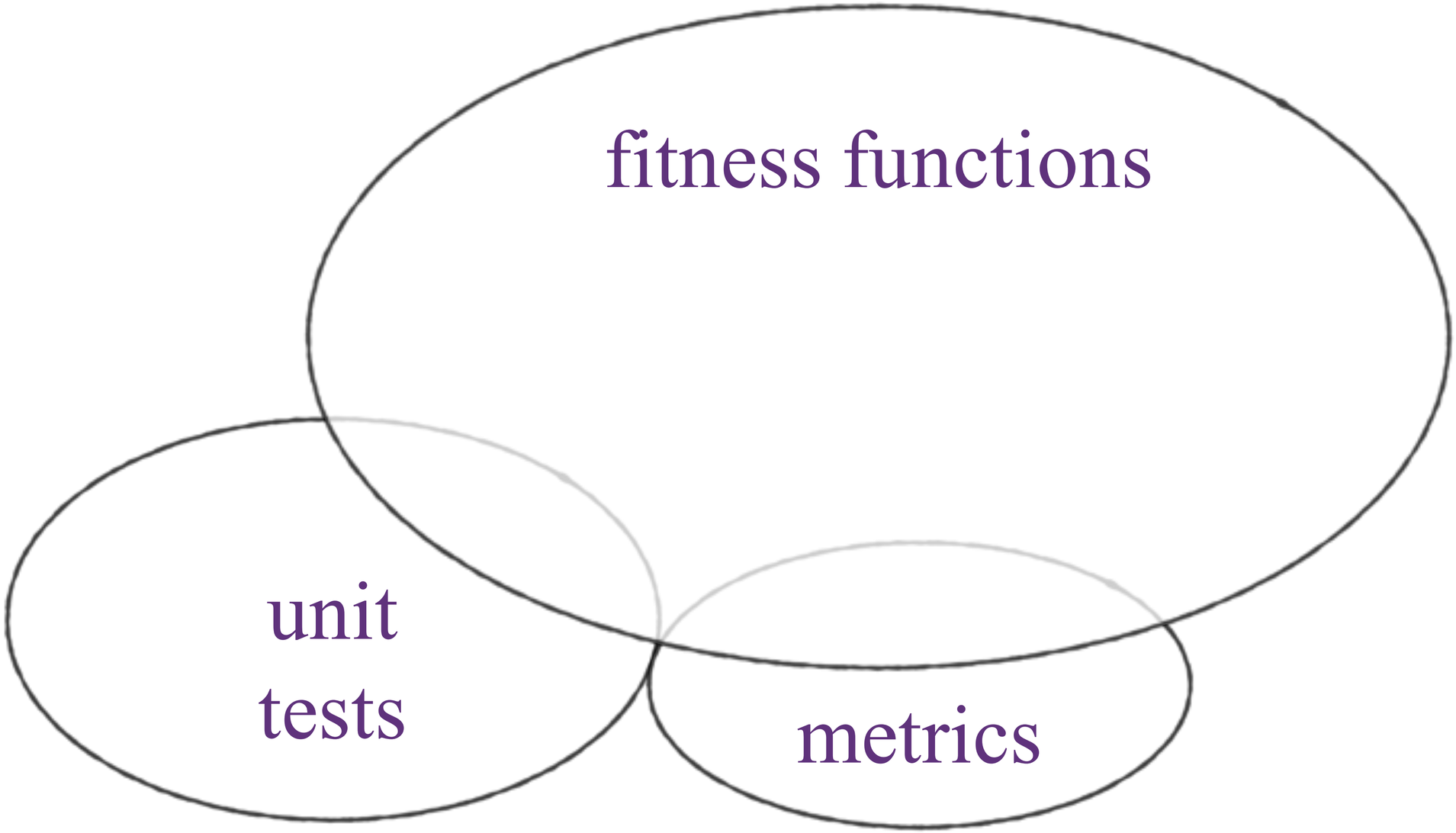
fitness functions



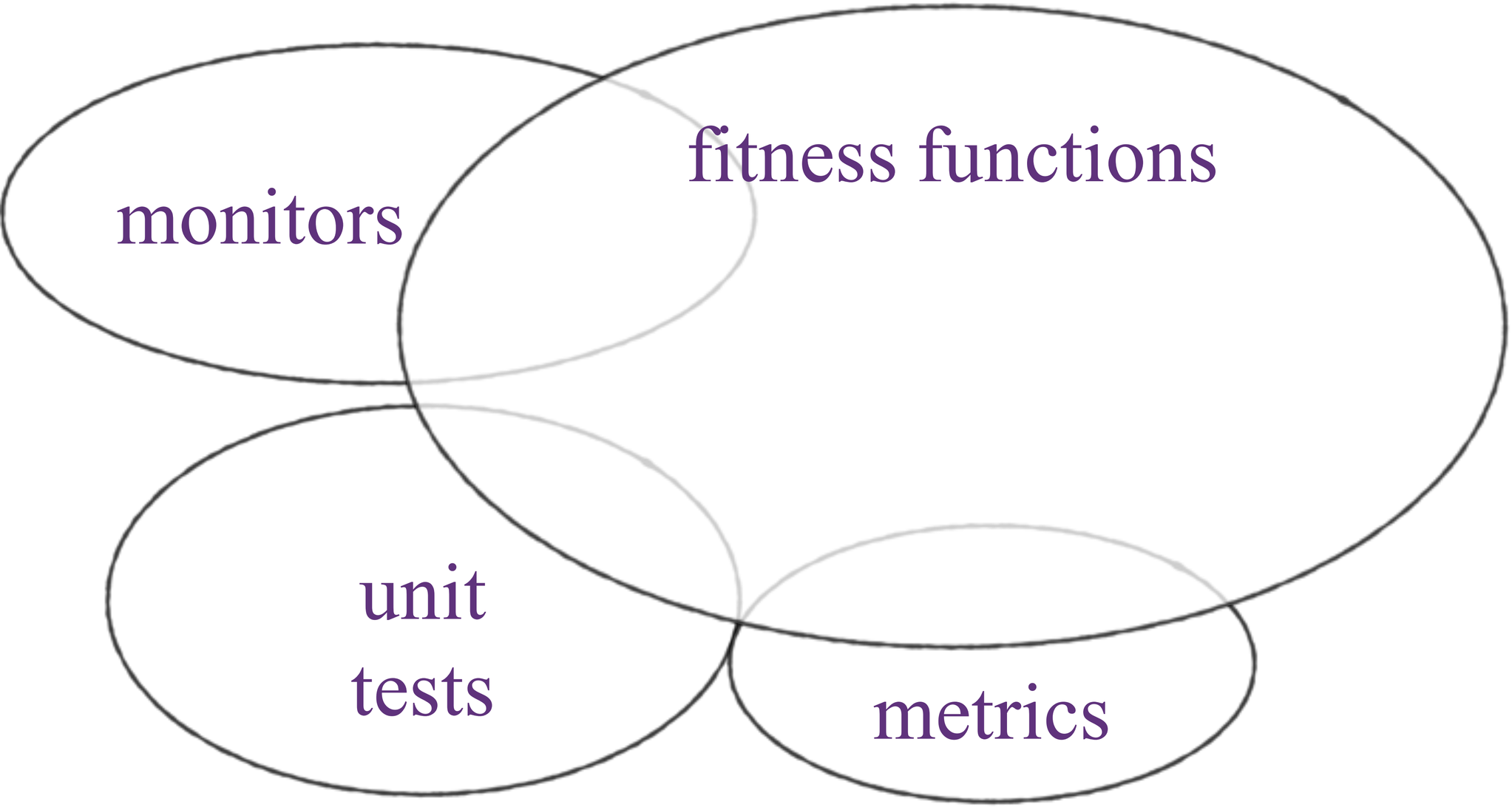
fitness functions



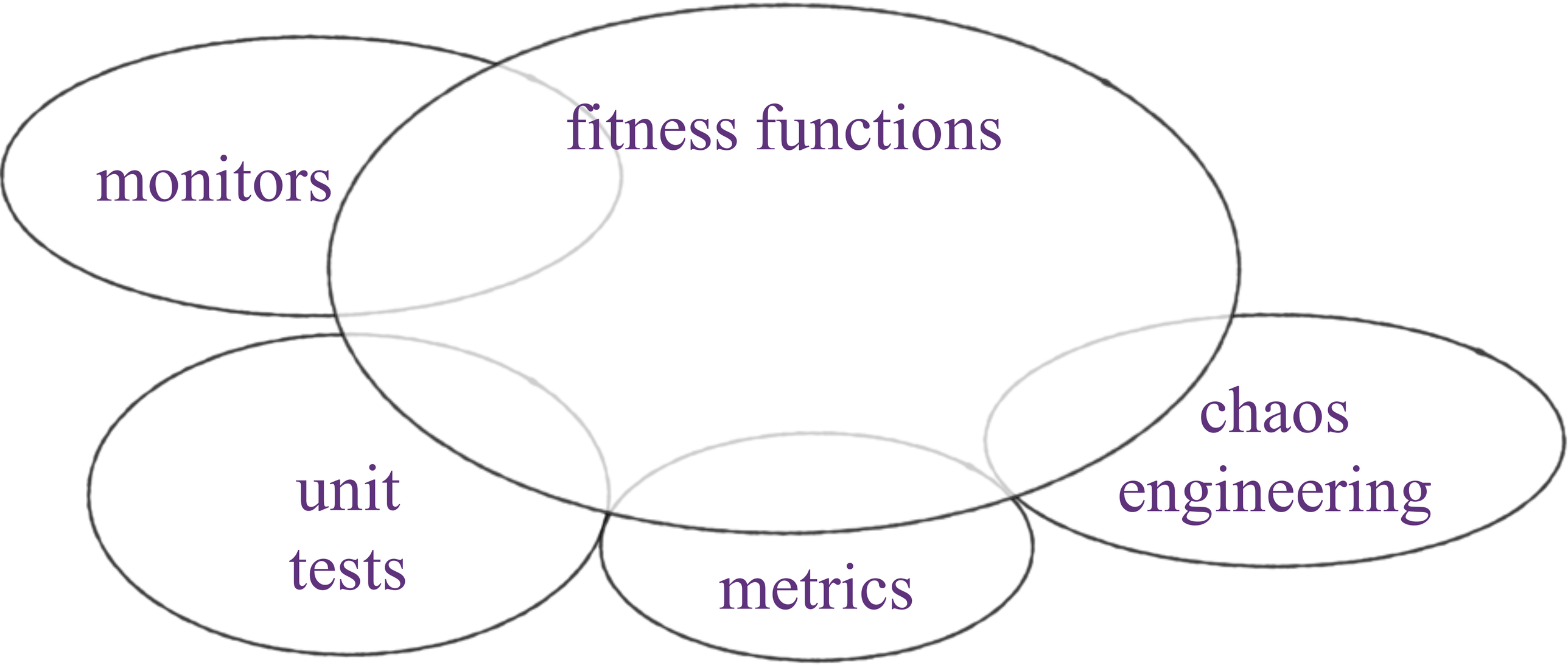
fitness functions



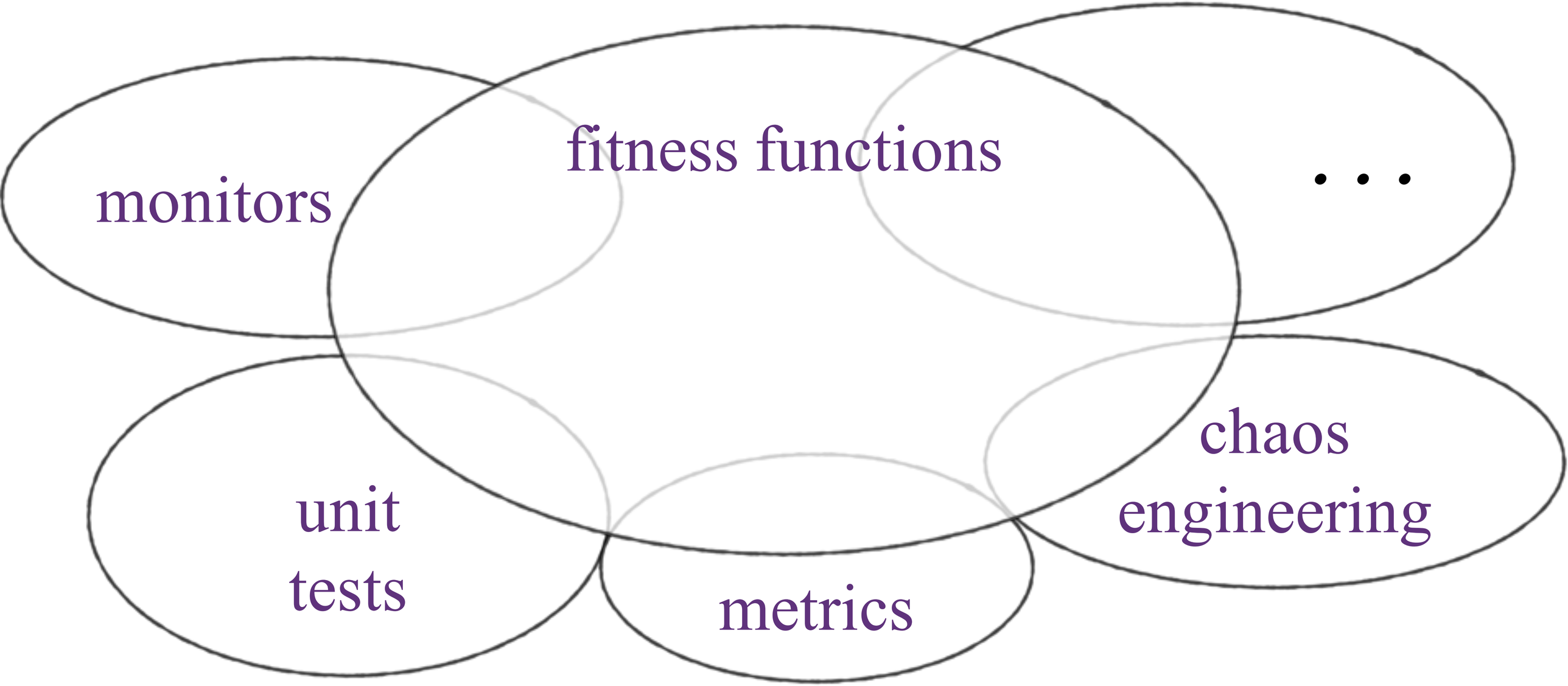
fitness functions



fitness functions

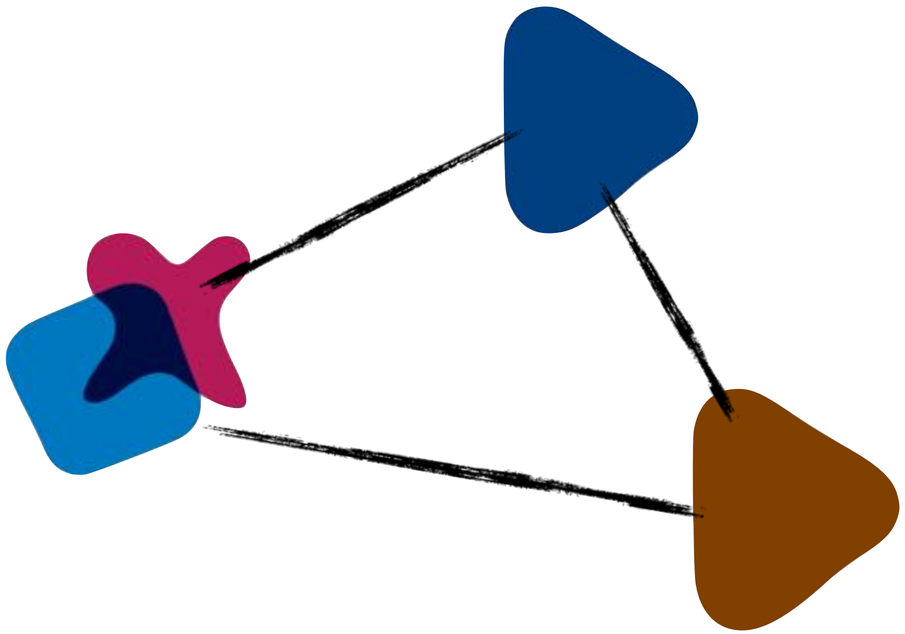


fitness functions

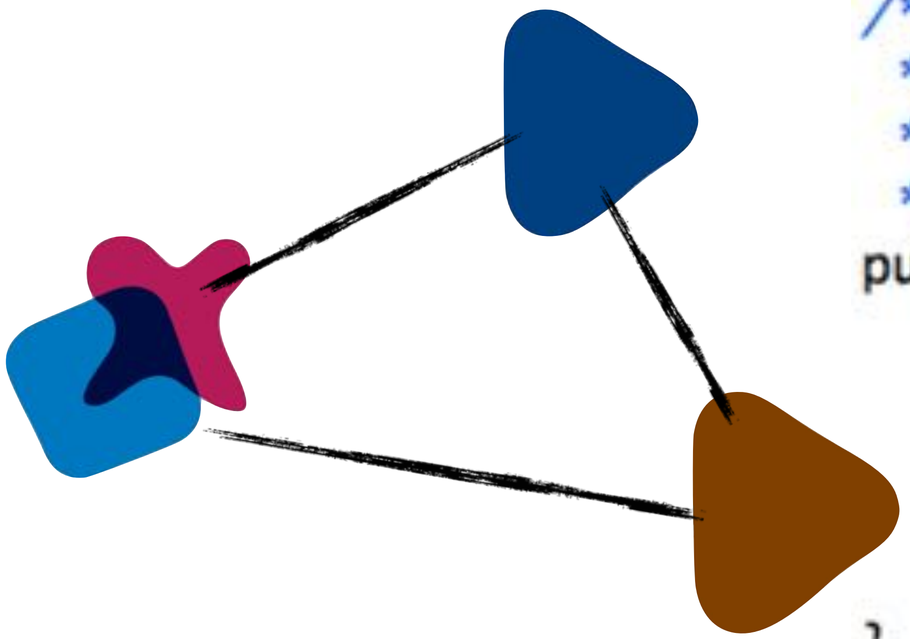


cyclic dependency function

cyclic dependency function



cyclic dependency function



```
/**
 * Tests that a package dependency cycle does not
 * exist for any of the analyzed packages.
 */
public void testAllPackages() {

    Collection packages = jdepend.analyze();

    assertEquals("Cycles exist",
                 false, jdepend.containsCycles());
}
```

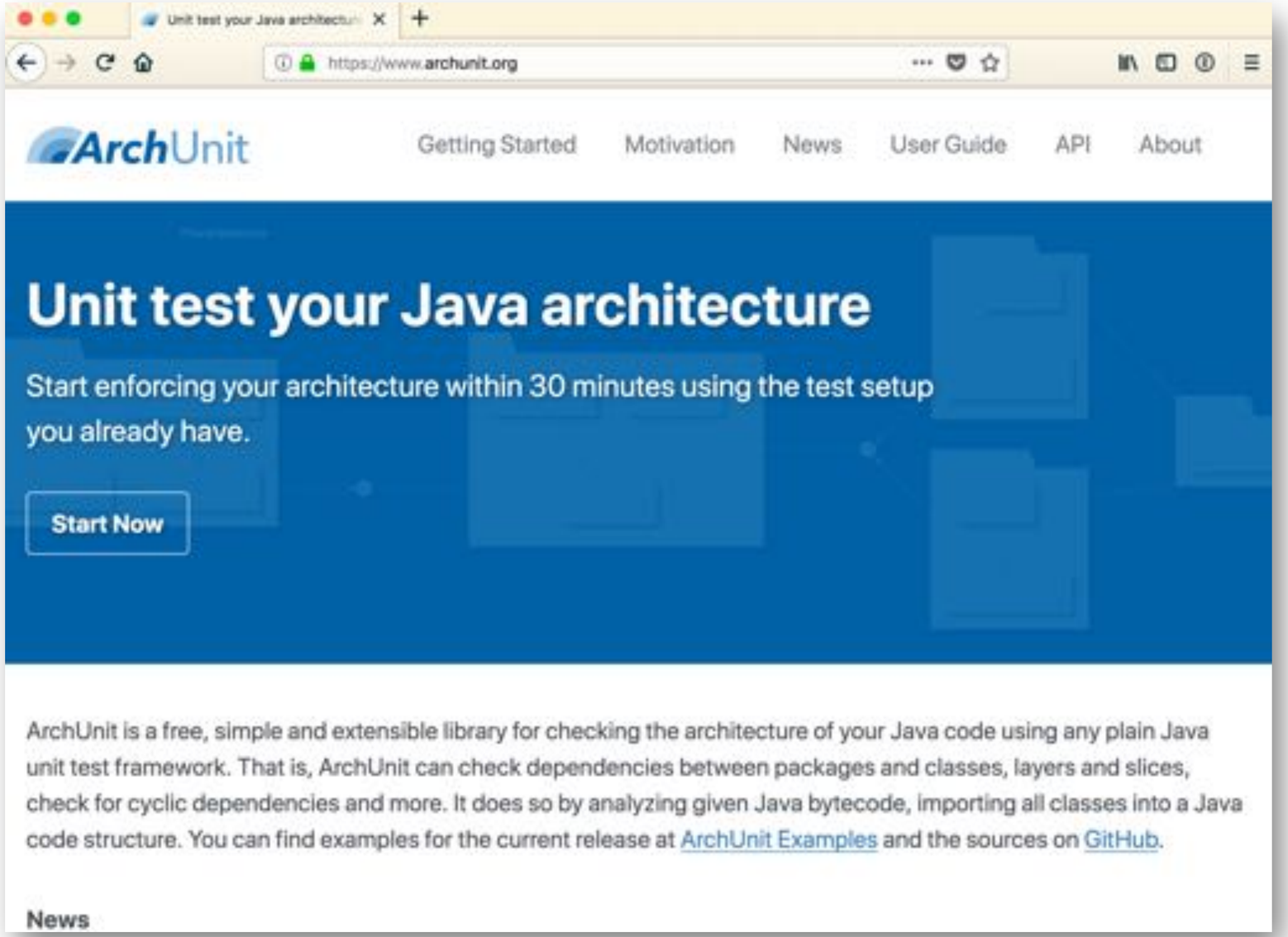
clarkware.com/software/JDepend.html

guided

***architectural** fitness function*:

any mechanism that provides an objective integrity assessment of some architectural characteristic(s).

https://www.archunit.org/



https://www.archunit.org/

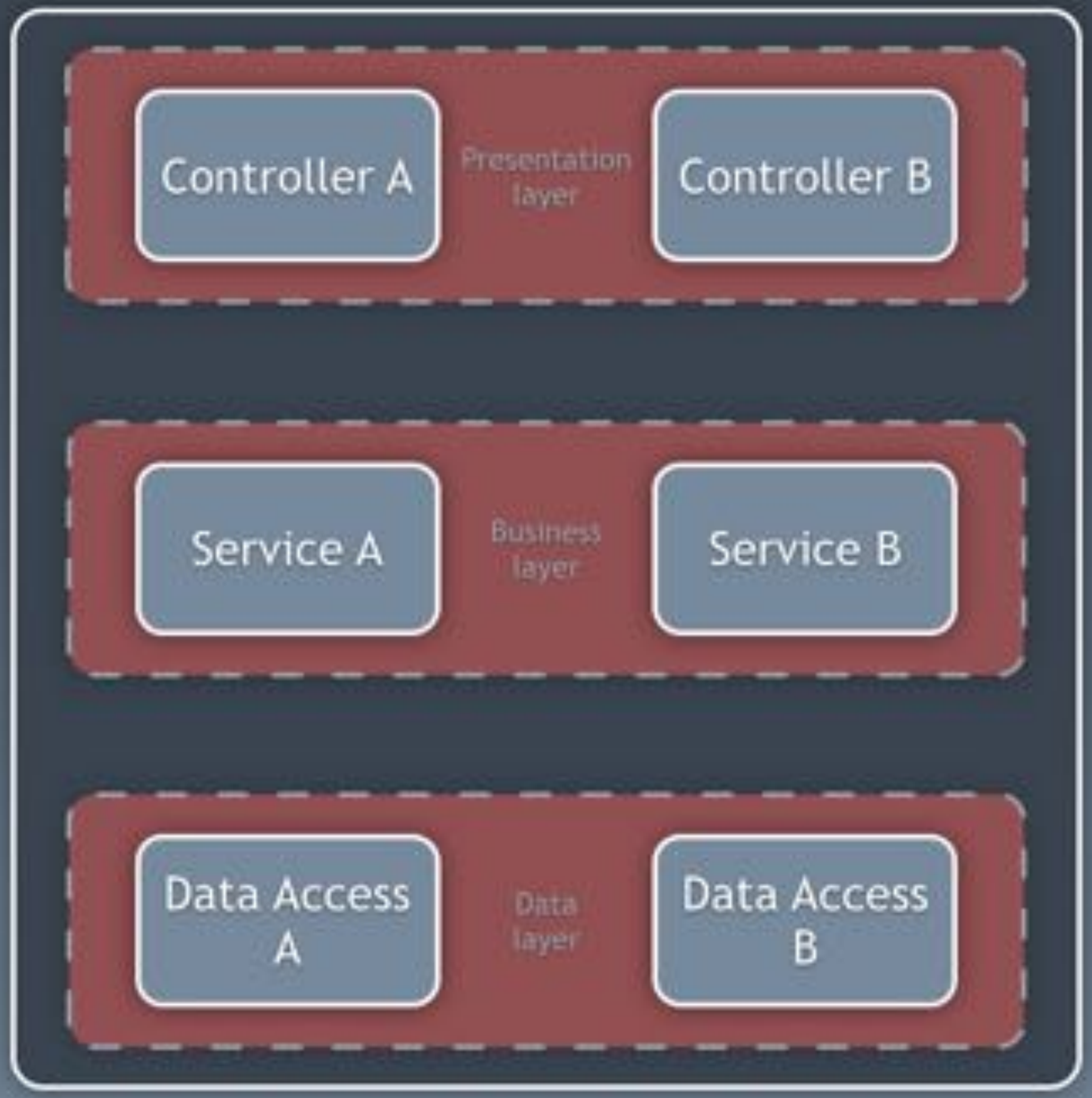
```java
import static com.tngtech.archunit.lang.syntax.ArchRuleDefinition.noClasses;
import static com.tngtech.archunit.library.GeneralCodingRules.ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING;

public class CodingRulesTest {
    private JavaClasses classes;

    @Before
    public void setUp() throws Exception {
        classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);
    }

    @Test
    public void classes_should_not_access_standard_streams_defined_by_hand() {
        noClasses().should(ACCESS_STANDARD_STREAMS).check(classes);
    }

    @Test
    public void classes_should_not_access_standard_streams_from_library() {
        NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS.check(classes);
    }

    @Test
    public void classes_should_not_throw_generic_exceptions() {
        NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS.check(classes);
    }

    @Test
    public void classes_should_not_use_java_util_logging() {
        NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING.check(classes);
    }
}
```

coding rules

https://www.archunit.org/

```java
public class InterfaceRules {

    @Test
    public void interfaces_should_not_have_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
                SomeBusinessInterface.class,
                SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveNameMatching(".*Interface").check(classes);
    }

    @Test
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
                SomeBusinessInterface.class,
                SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveSimpleNameContaining("Interface").check(classes);
    }

    @Test
    public void interfaces_must_not_be_placed_in_implementation_packages() {
        JavaClasses classes = new ClassFileImporter().importPackagesOf(SomeInterfacePlacedInTheWrongPackage.class);

        noClasses().that().resideInAPackage("..impl..").should().beInterfaces().check(classes);
    }
}
```

interface rules



Package by layer (horizontal slicing)

https://www.archunit.org/

```java
public class LayerDependencyRulesTest {
    private JavaClasses classes;

    @Before
    public void setUp() throws Exception {
        classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);
    }

    @Test
    public void services_should_not_access_controllers() {
        noClasses().that().resideInAPackage("..service..")
                .should().accessClassesThat().resideInAPackage("..controller..").check(classes);
    }

    @Test
    public void persistence_should_not_access_services() {
        noClasses().that().resideInAPackage("..persistence..")
                .should().accessClassesThat().resideInAPackage("..service..").check(classes);
    }

    @Test
    public void services_should_only_be_accessed_by_controllers_or_other_services() {
        classes().that().resideInAPackage("..service..")
                .should().onlyBeAccessed().byAnyPackage("..controller..", "..service..").check(classes);
    }
}
```

layer dependency

https://www.archunit.org/

```
@Test
public void third_party_class_should_only_be_instantiated_via_workaround() {
    classes().should(notCreateProblematicClassesOutsideOfWorkaroundFactory()
            .as(THIRD_PARTY_CLASS_RULE_TEXT))
            .check(classes);
}

private ArchCondition<JavaClass> notCreateProblematicClassesOutsideOfWorkaroundFactory() {
    DescribedPredicate<JavaCall<?>> constructorCallOfThirdPartyClass =
            target(is(constructor())).and(targetOwner(is(assignableTo(ThirdPartyClassWithProblem.class))));

    DescribedPredicate<JavaCall<?>> notFromWithinThirdPartyClass =
            originOwner(is(not(assignableTo(ThirdPartyClassWithProblem.class)))).forSubType();

    DescribedPredicate<JavaCall<?>> notFromWorkaroundFactory =
            originOwner(is(not(equivalentTo(ThirdPartyClassWorkaroundFactory.class)))).forSubType();

    DescribedPredicate<JavaCall<?>> targetIsIllegalConstructorOfThirdPartyClass =
            constructorCallOfThirdPartyClass.
                    and(notFromWithinThirdPartyClass).
                    and(notFromWorkaroundFactory);

    return never(callCodeUnitWhere(targetIsIllegalConstructorOfThirdPartyClass));
}
```

governance

NetArchTest

https://github.com/BenMorris/NetArchTest/

```
// Controllers should not directly reference repositories
var result = Types.InCurrentDomain()
    .That()
    .ResideInNamespace("NetArchTest.SampleLibrary.Presentation")
    .ShouldNot()
    .HaveDependencyOn("NetArchTest.SampleLibrary.Data")
    .GetResult().IsSuccessful;
```

NetArchTest

https://github.com/BenMorris/NetArchTest/

```
// Only classes in the data namespace can have a dependency on System.Data
result = Types.InCurrentDomain()
    .That().HaveDependencyOn("System.Data")
    .And().ResideInNamespace(("ArchTest"))
    .Should().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .GetResult().IsSuccessful;
```

NetArchTest

https://github.com/BenMorris/NetArchTest/

```
// All the classes in the data namespace should implement IRepository
result = Types.InCurrentDomain()
    .That().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .And().AreClasses()
    .Should().ImplementInterface(typeof(IRepository<>))
    .GetResult().IsSuccessful;

// Classes that implement IRepository should have the suffix "Repository"
result = Types.InCurrentDomain()
    .That().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .And().AreClasses()
    .Should().HaveNameEndingWith("Repository")
    .GetResult().IsSuccessful;

// Classes that implement IRepository must reside in the Data namespace
result = Types.InCurrentDomain()
    .That().ImplementInterface(typeof(IRepository<>))
    .Should().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .GetResult().IsSuccessful;

// All the service classes should be sealed
result = Types.InCurrentDomain()
    .That().ImplementInterface(typeof(IWidgetService))
    .Should().BeSealed()
    .GetResult().IsSuccessful;
```

amazon
CHAOS
MONKEY

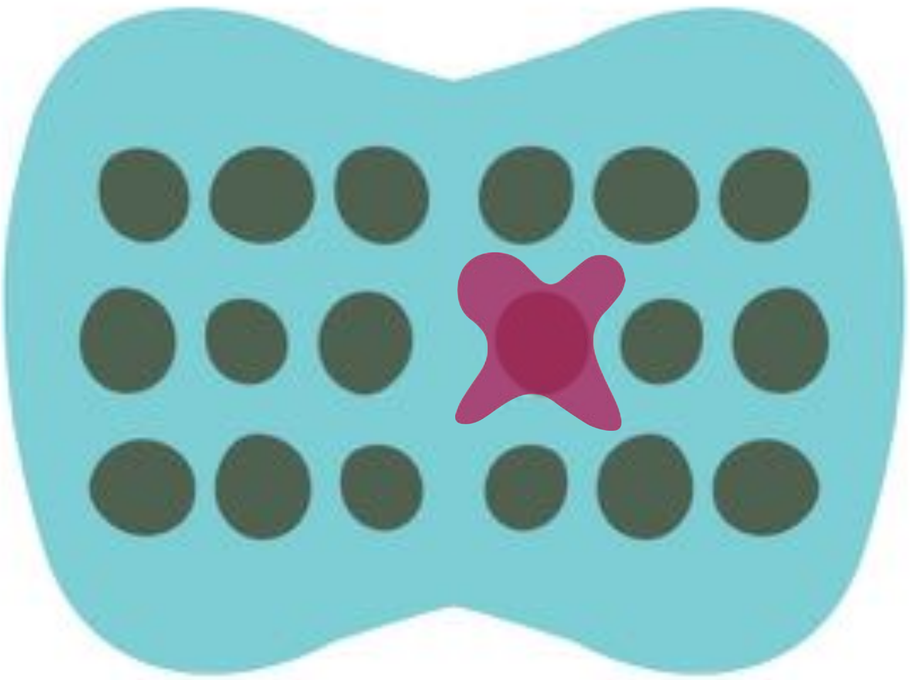
amazon
web services
CHAOS
MONKEY

SIMIAN
ARMY

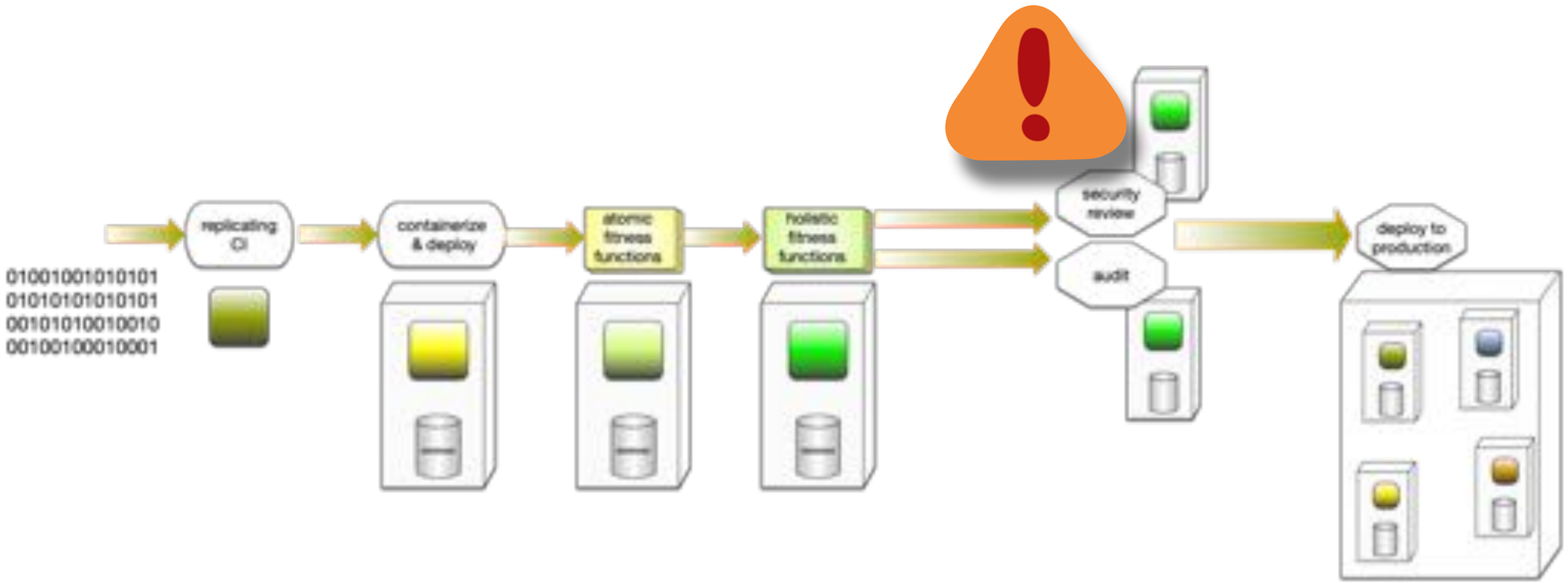
conformity monkey
SIMIAN
ARMY

security monkey

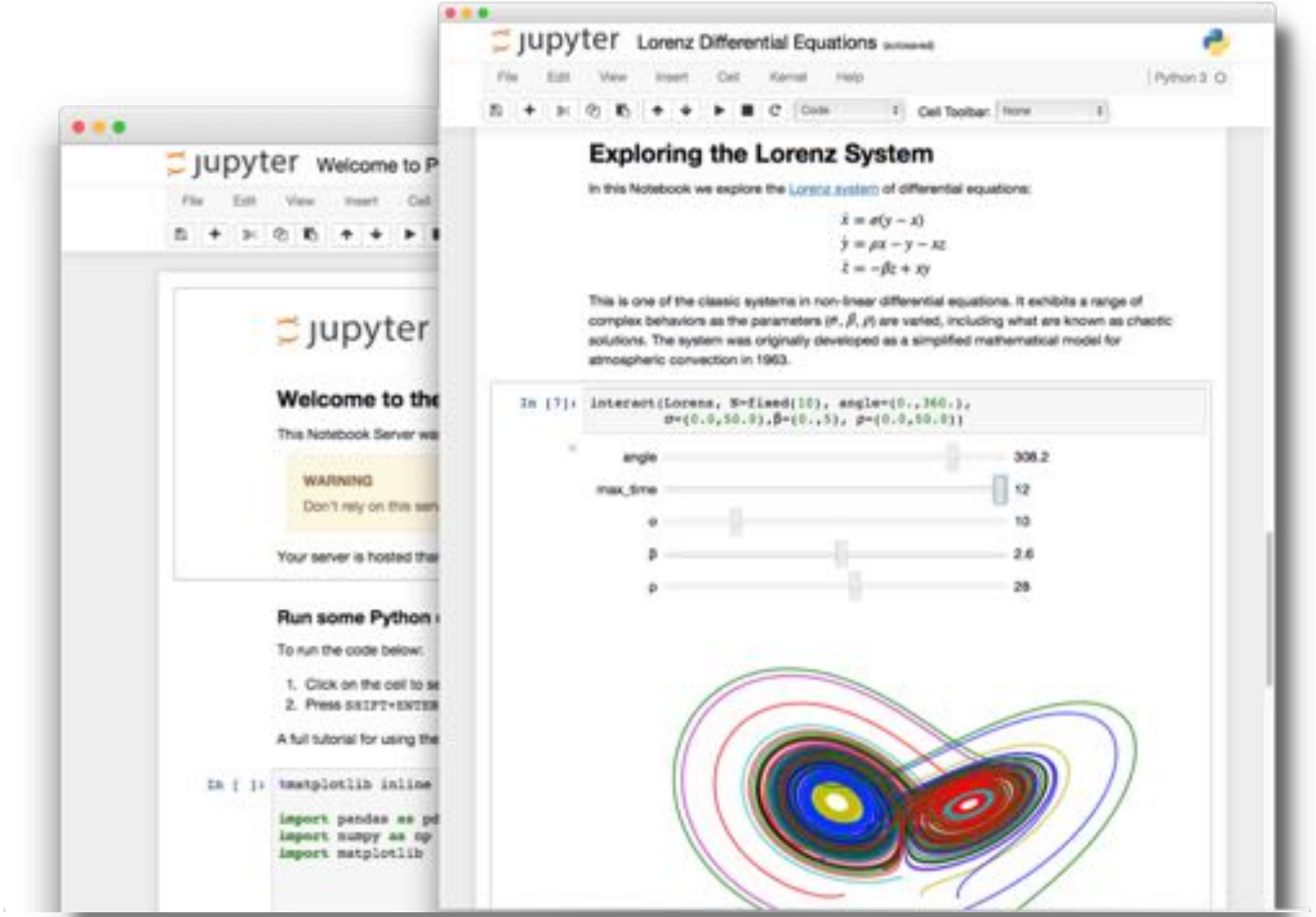
Zero-day Security Check



Zero-day Security Check



Jupyter Notebooks



www.feststelltaste.de/checking-ar

feststelltaste

About Legacy Systems, Software Analytics and the Fundamental Problems of Software Engineering

Home Now About me Privacy Contact

Checking Architecture Governance with jQAssistant, Neo4j and Jupyter

Introduction

Software architects have to make sure that the communicated software architecture blueprints exist in the real world. For this, manual inspections as well as automated measurements are needed to avoid surprises.

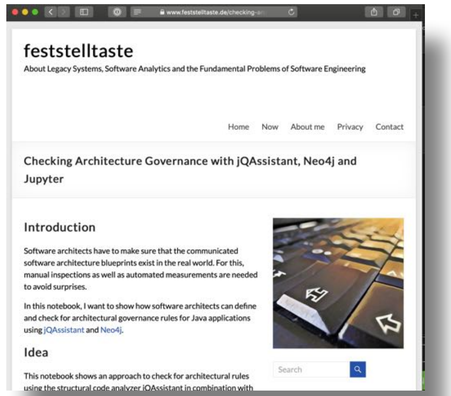
In this notebook, I want to show how software architects can define and check for architectural governance rules for Java applications using jQAssistant and Neo4j.

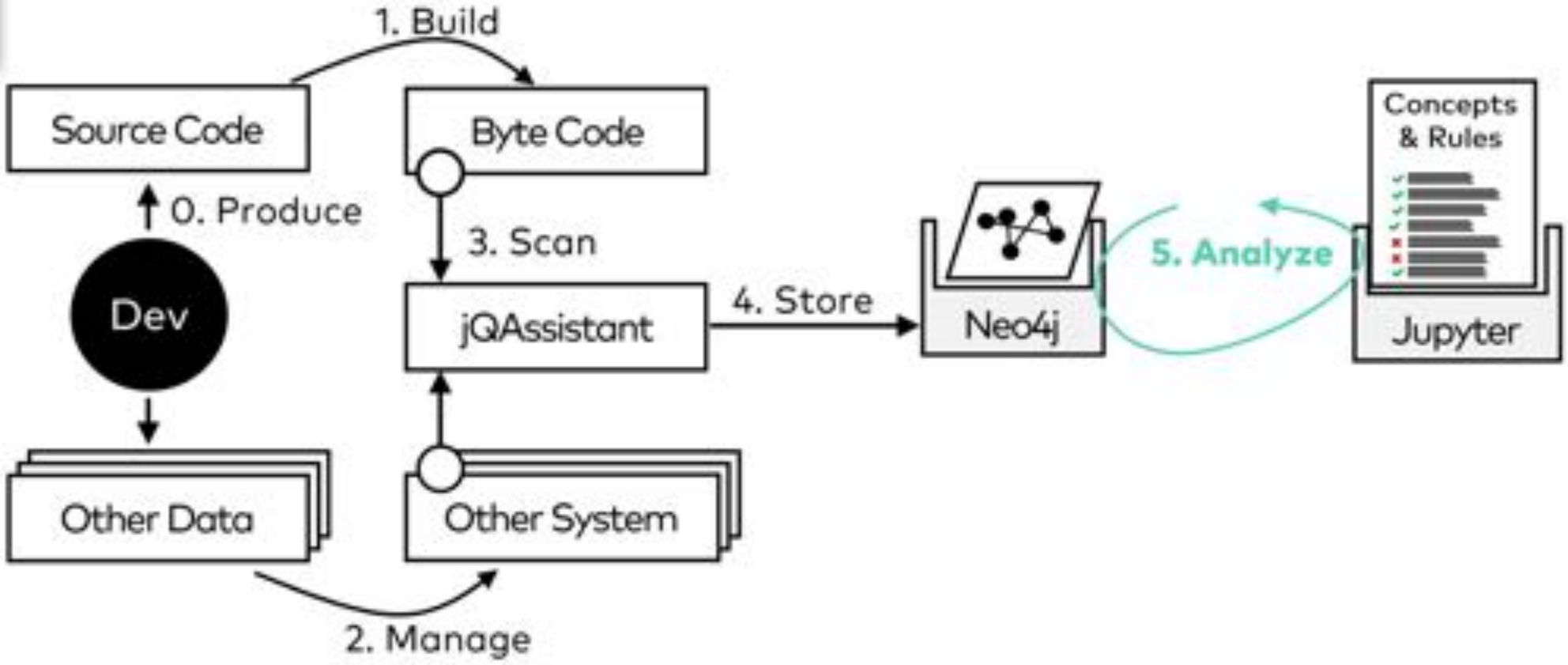
Idea

This notebook shows an approach to check for architectural rules using the structural code analyzer jQAssistant in combination with

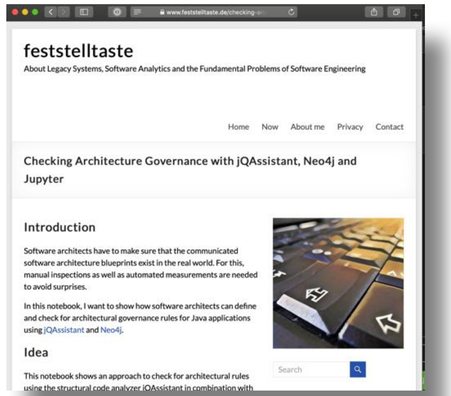
Search

https://www.feststelltaste.de/checking-architecture-governance-with-jqassistant-neo4j-and-jupyter/





https://www.feststelltaste.de/checking-architecture-governance-with-jqassistant-neo4j-and-jupyter/



Rule Definition

Because we want to enable developers to quickly identify Entities in the application, we use the Java package names to define were all the Entities are placed. Thus, Entities must be placed in a package named `model`.

The following query lists all existing Entities in the scanned codebase:

In [3]:
```
%%cypher
MATCH (e:Entity)<-[:CONTAINS]-(p:Package)
WHERE p.name = "model"
RETURN e.fqn as Entity, p.name as Package
```

5 rows affected.

Out[3]:

Entity	Package
org.springframework.samples.petclinic.model.Owner	model
org.springframework.samples.petclinic.model.Pet	model
org.springframework.samples.petclinic.model.Visit	model
org.springframework.samples.petclinic.model.Specialty	model
org.springframework.samples.petclinic.model.Vet	model

Rule Violations

The following query lists all Enties that doesn't comply to the rule above:

In [4]:
```
%%cypher
MATCH (e:Entity)<-[:CONTAINS]-(p:Package)
WHERE p.name <> "model"
RETURN e.fqn as MisplacedEntity, p.name as WrongPackage
```

1 rows affected.

Out[4]:

MisplacedEntity	WrongPackage

Kata Exercise - Automating Architecture Governance

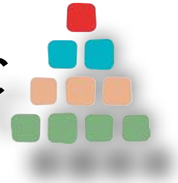
Sysop Squad has had a problem with lost / mis-routed tickets, and the architects have narrowed it down to lost messages in a message queue because of a hard to reproduce error.

1. What kind of fitness function can the Sysop Squad architects write to track down the root cause?
2. What fitness function could they write to prevent this from happening in the future?

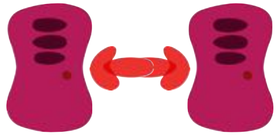
How do I create systems with high <u>semantic</u> coupling but low <u>syntactic</u> coupling?

quantum connascence

static



quantum



synchronous → *asynchronous* ⇢

how quanta communicate

contracts

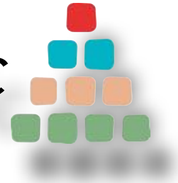
tight *loose*

impact on operational
(and other) architecture characteristics

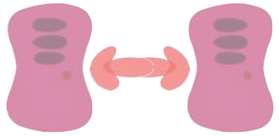
useful for hybrid architecture design,
architecture migration, integration, etc.

quantum connascence

static



quantum



synchronous → *asynchronous* ⇢

how quanta communicate

contracts

tight *loose*

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*

tradeoffs

tight — *loose*

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

tradeoffs

tight — *loose*

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

* decoupled…
* …therefore better for decoupled architectures
* contract management

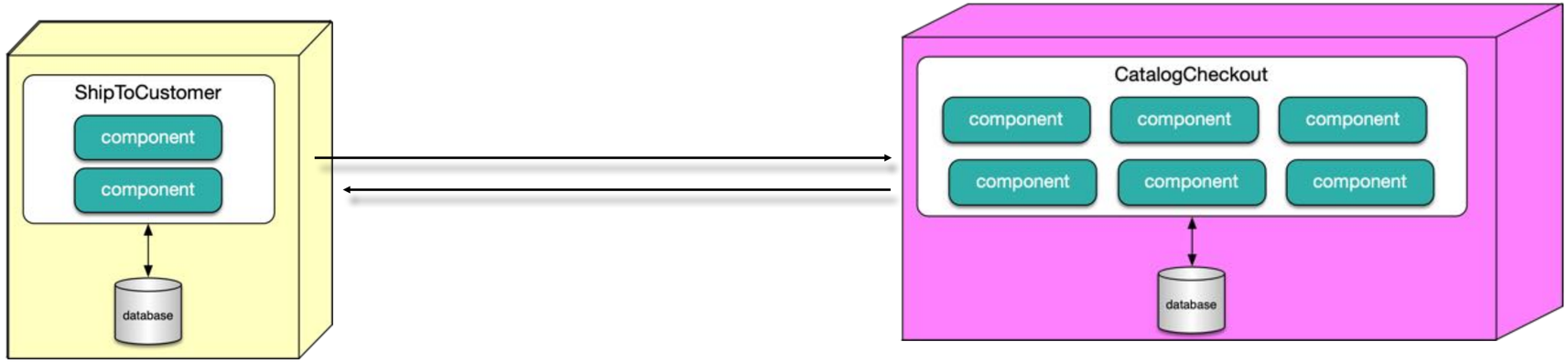
tradeoffs

tight — *loose*

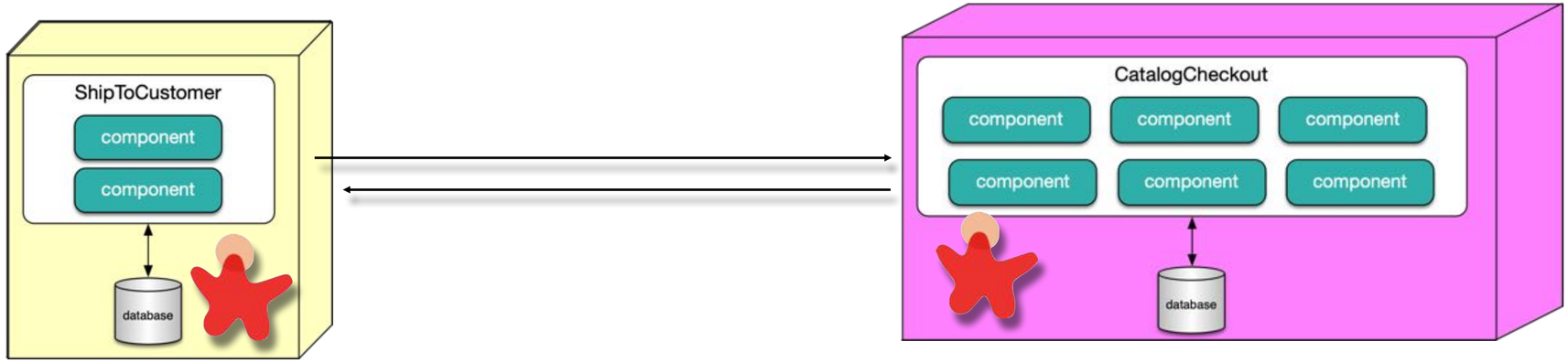
* guaranteed contract fidelity
* distinct versions (+ or –?)
* build-time contracts

* decoupled…
* …therefore better for decoupled architectures
* contract management

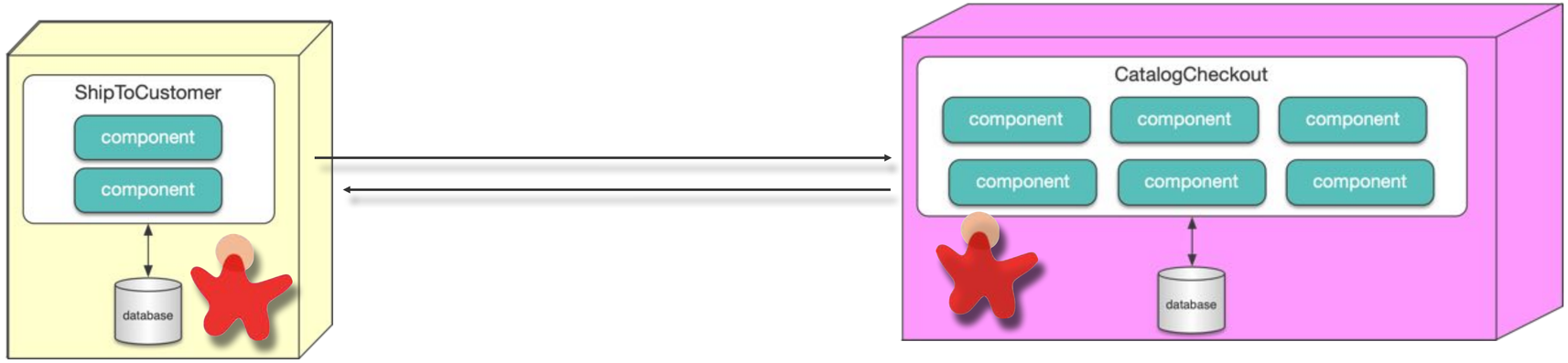
contracts in microservices



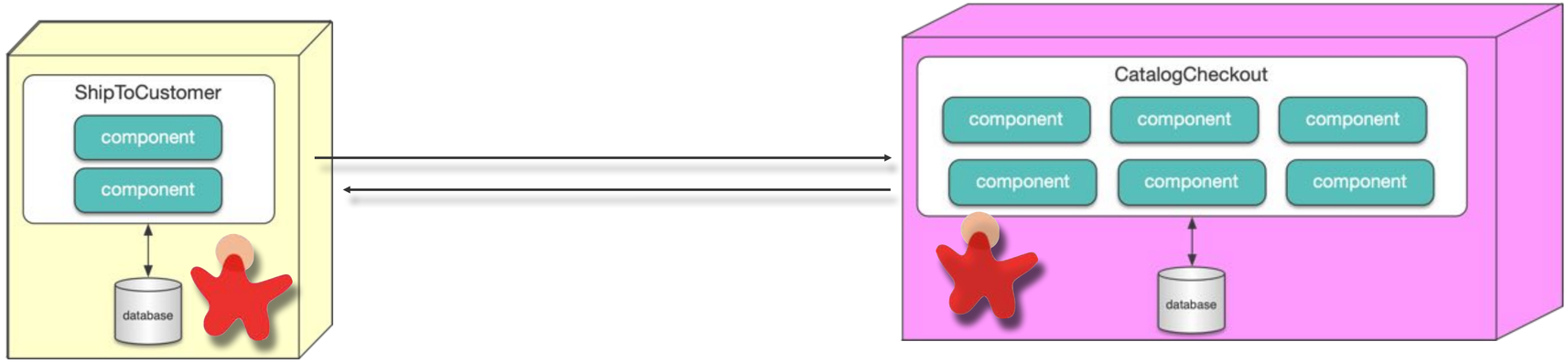
contracts in microservices



contracts in microservices

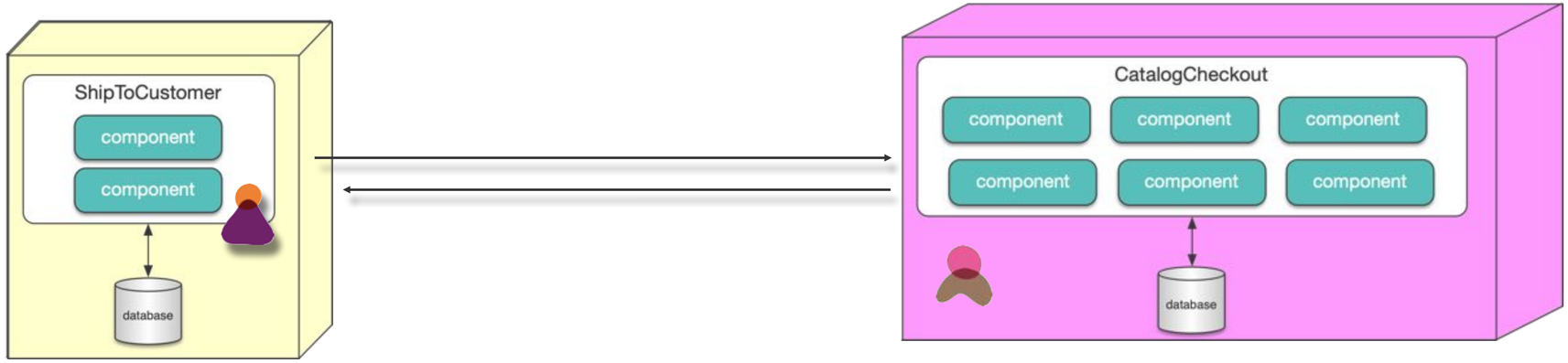


contracts in microservices

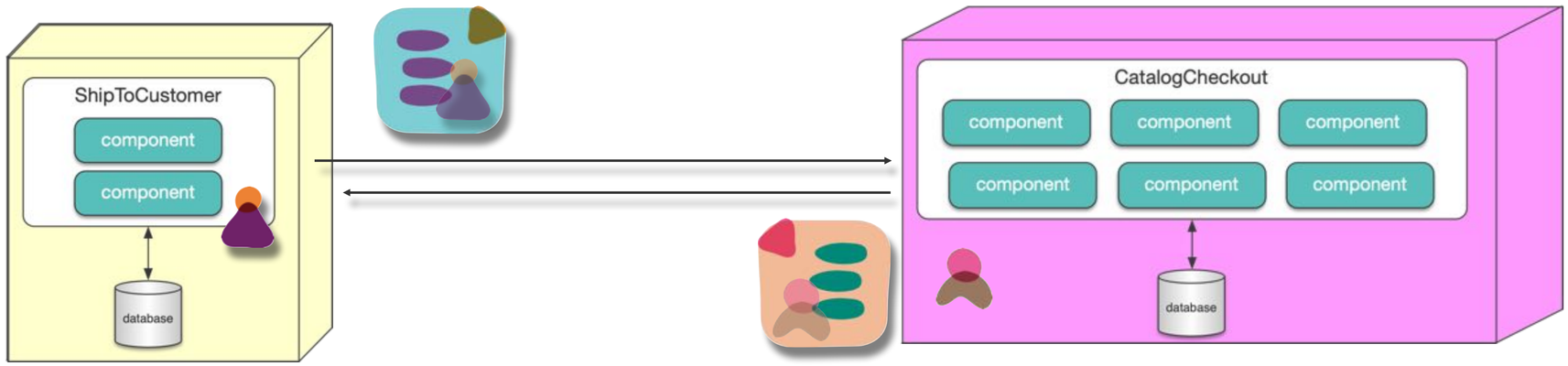


tight

contracts in microservices

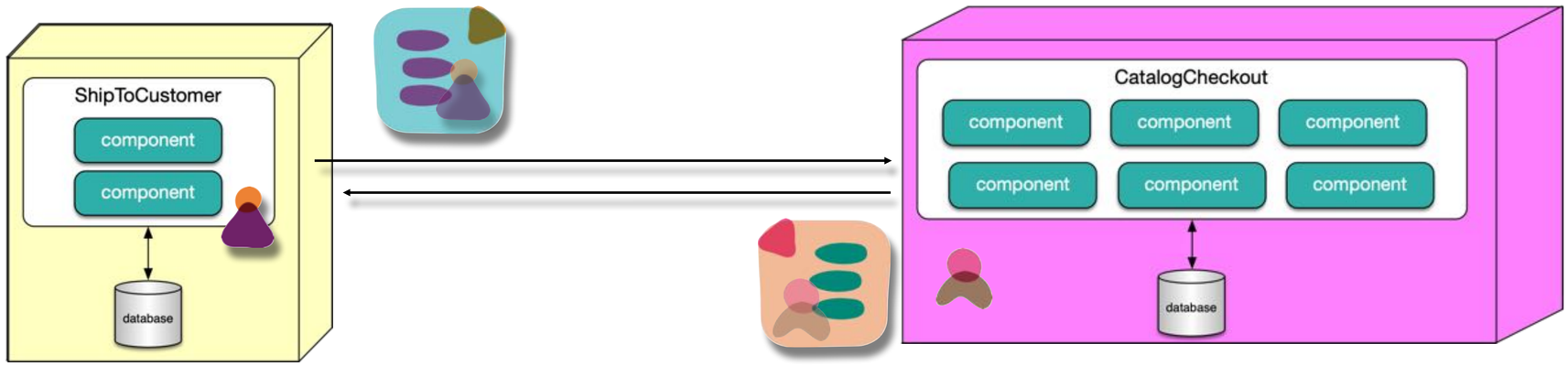


contracts in microservices



contracts in microservices

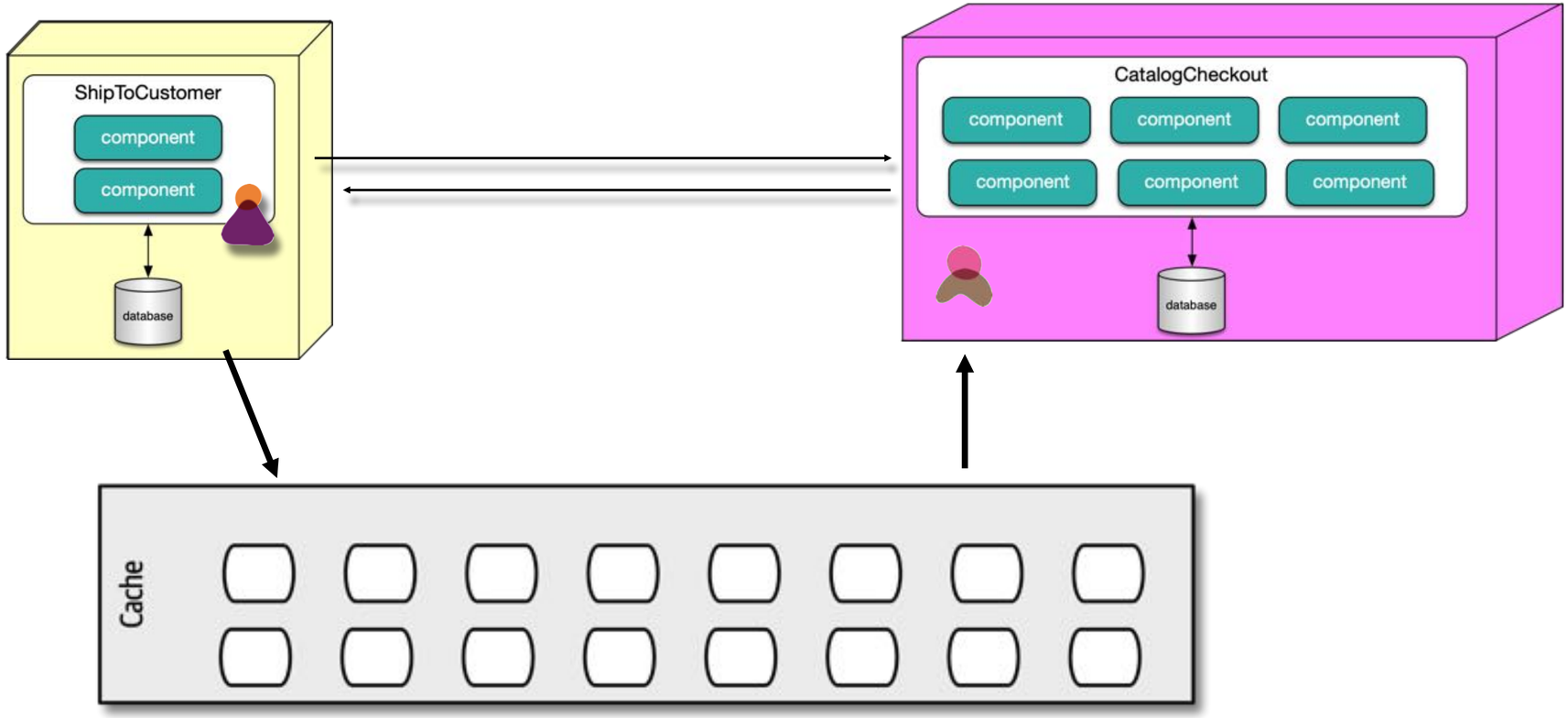
Transfer values, not types.



loose

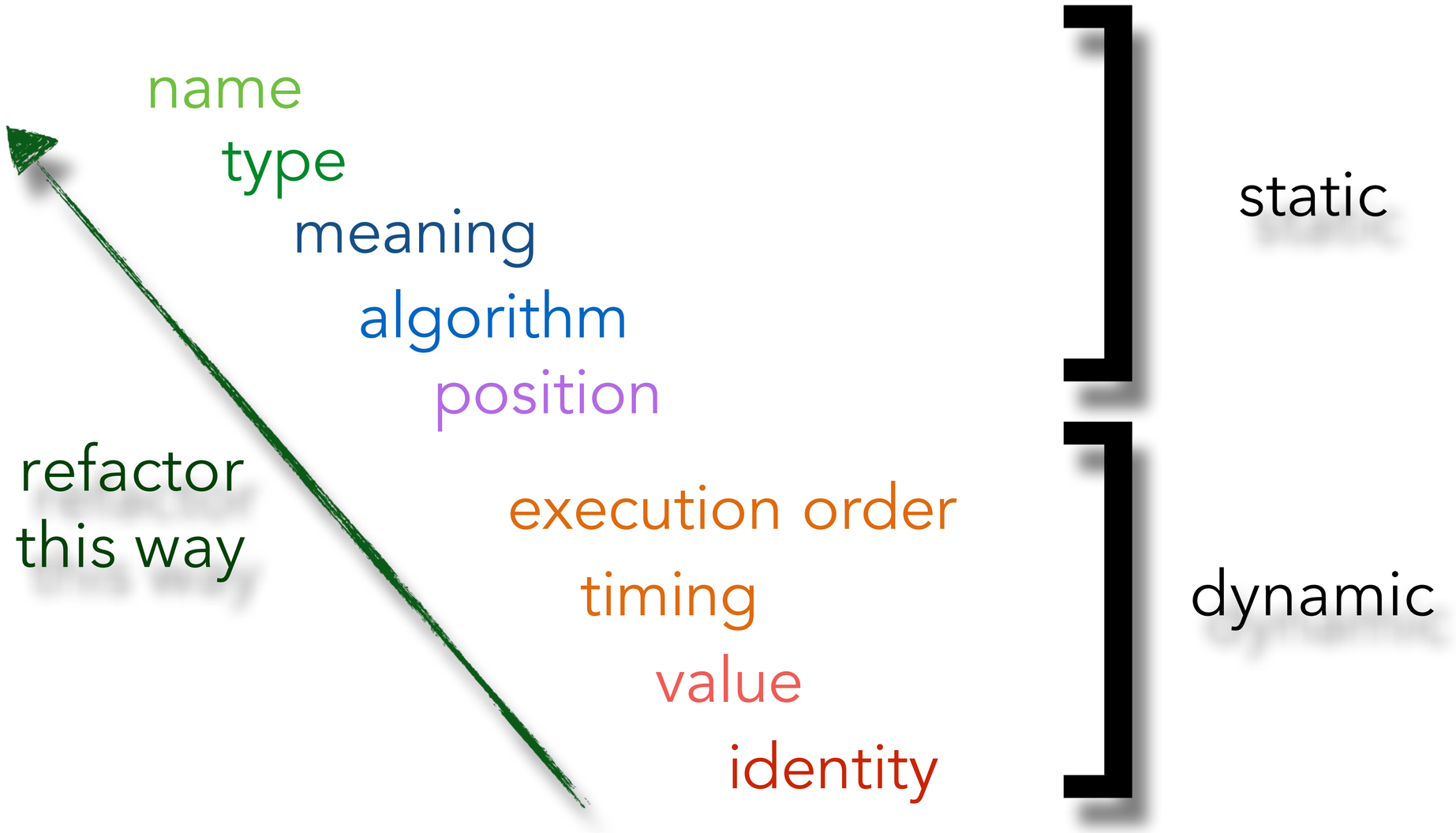
contracts in microservices

Transfer values, not types.

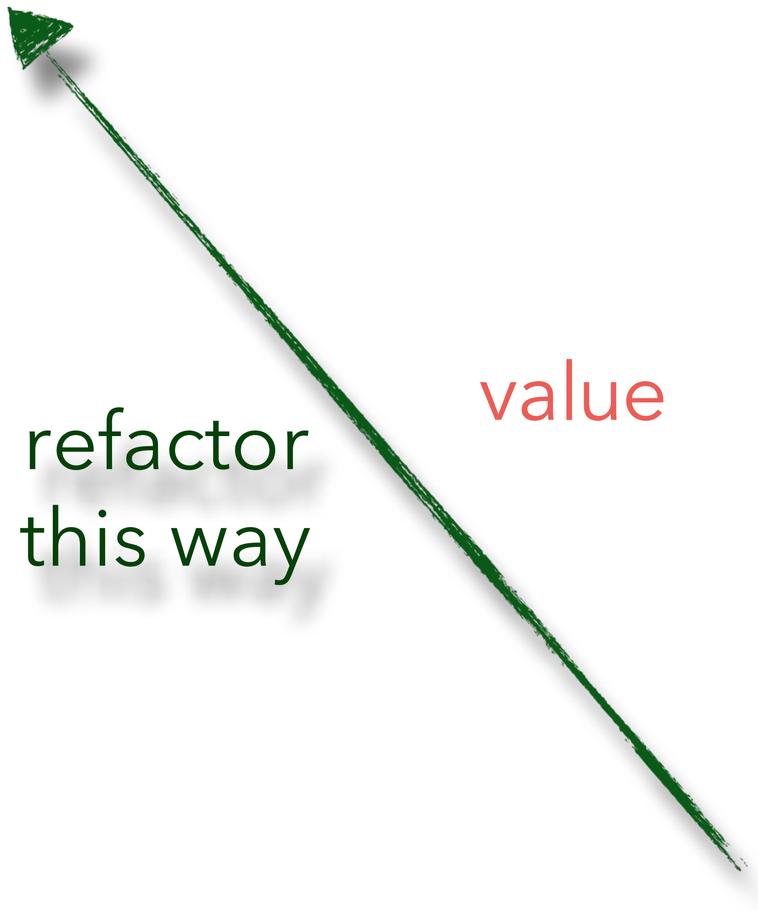


connascence properties

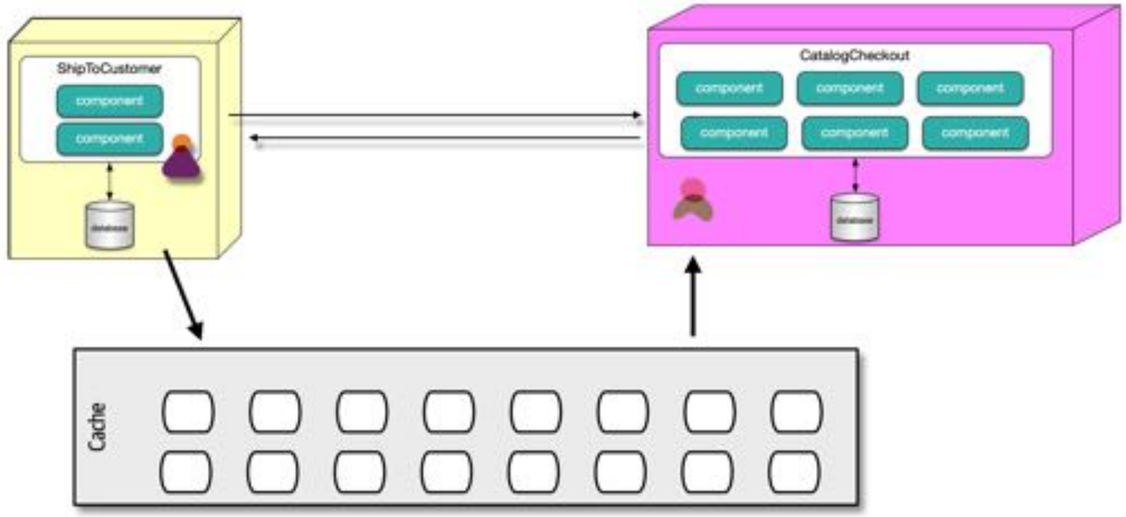
Strength



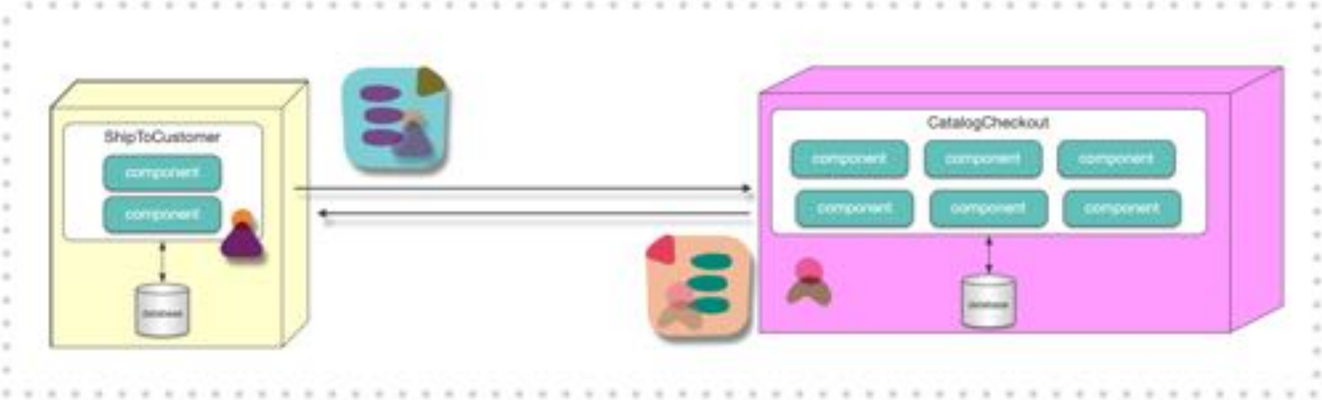
improving connascence



value



identity



value-based contracts

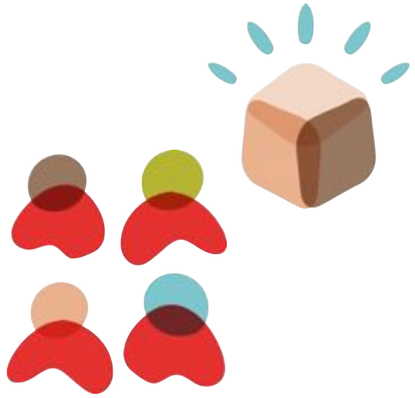
+ more malleable integration points

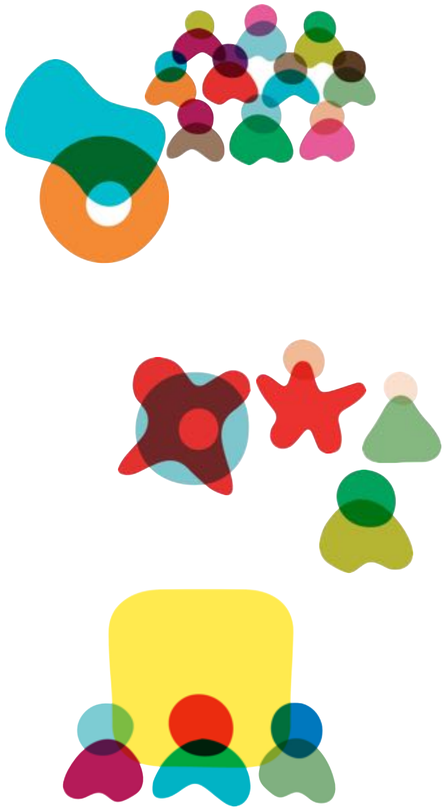
+ easier to evolve

+ decouples integration from implementation platform

– the “contract” part of contract

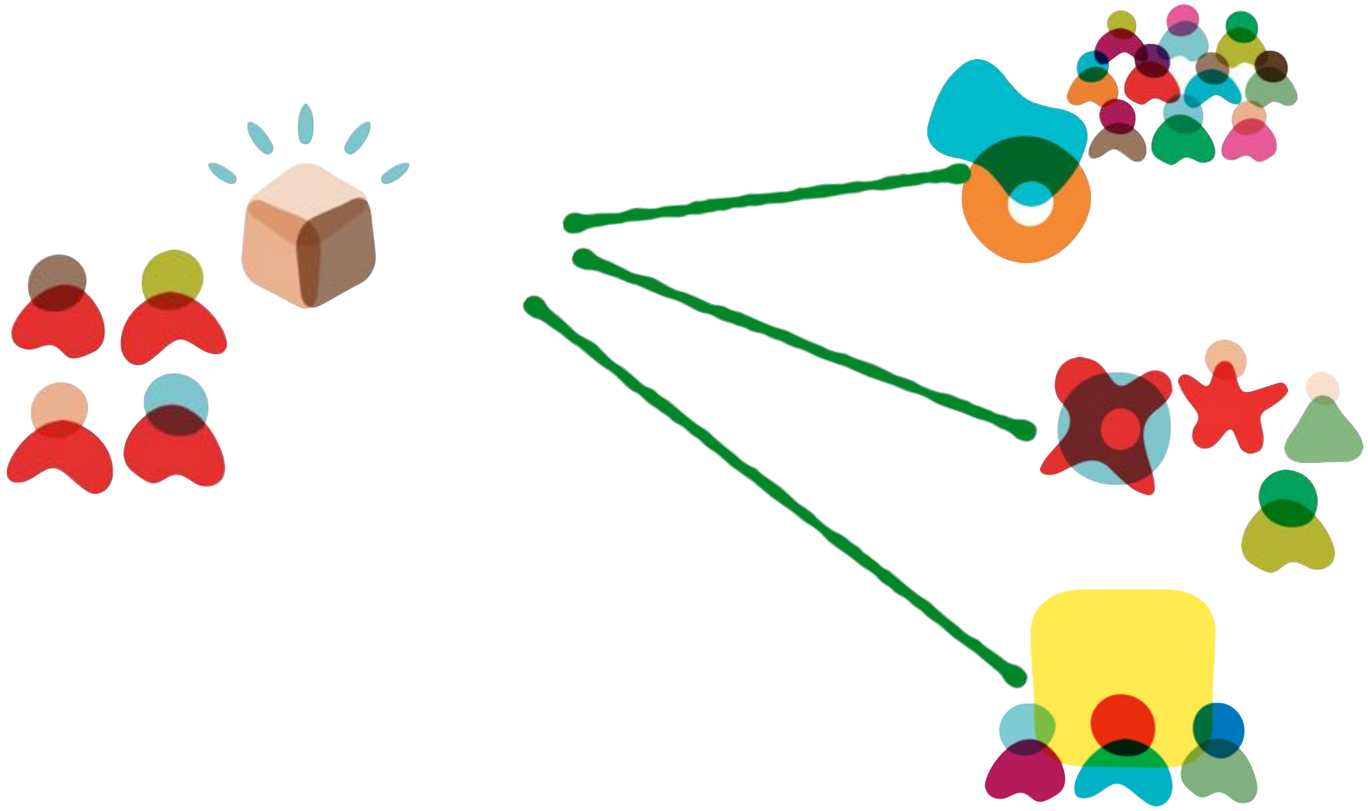
contract fitness function





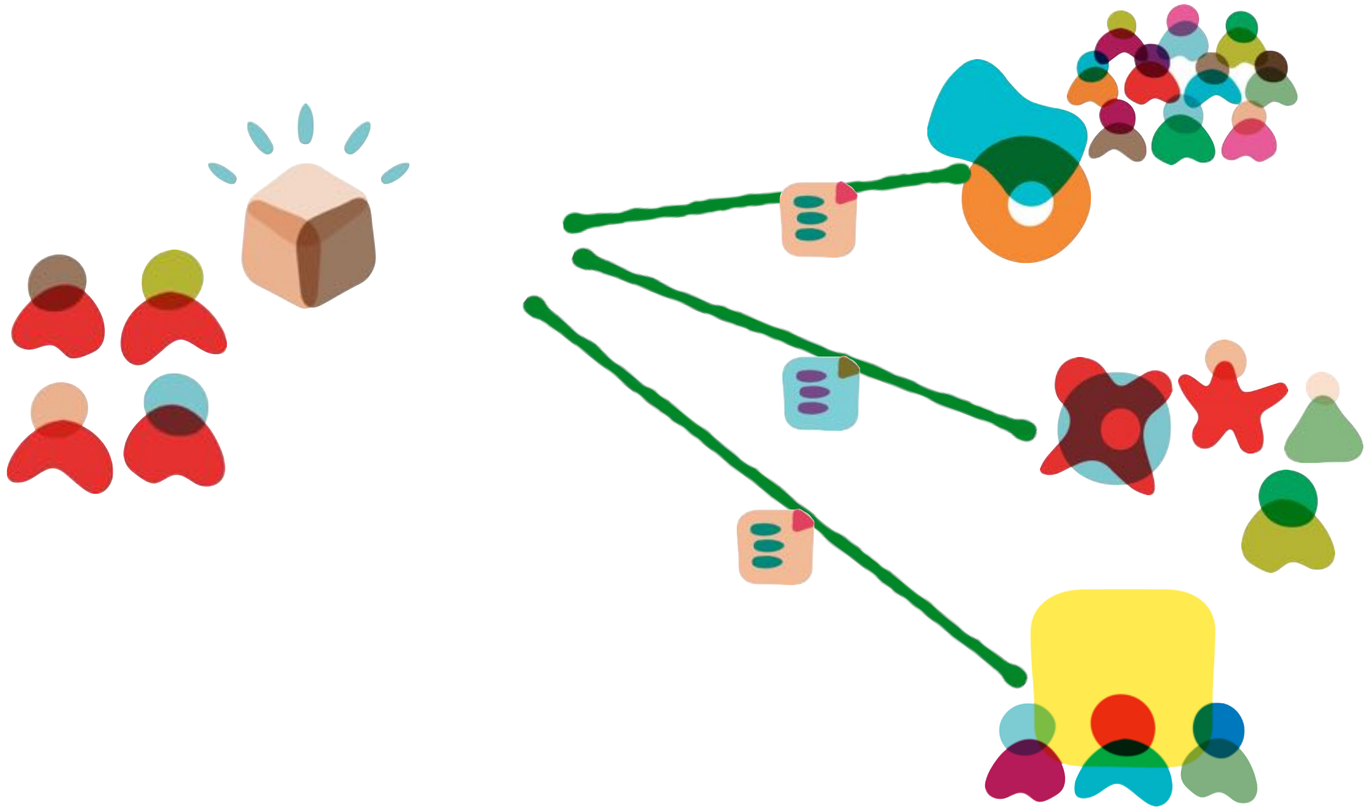
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



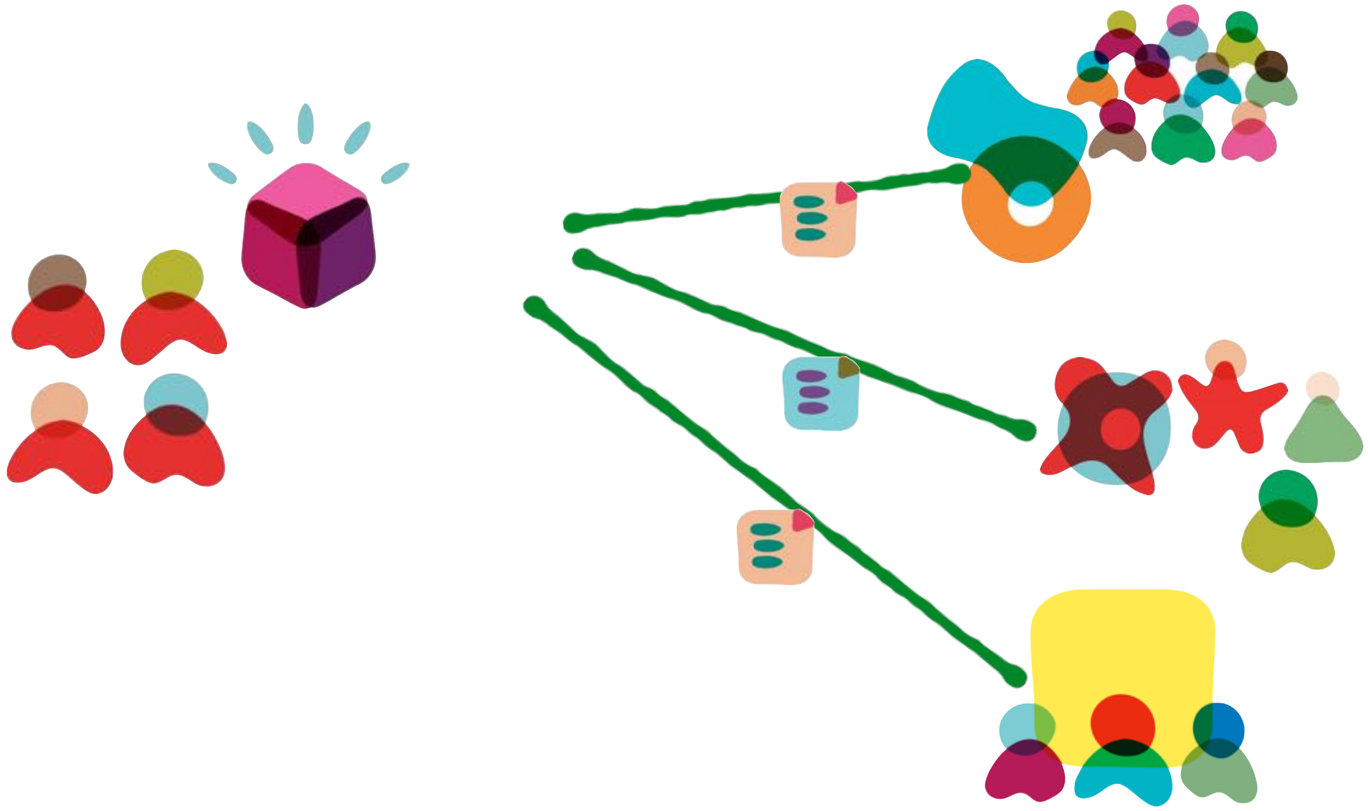
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



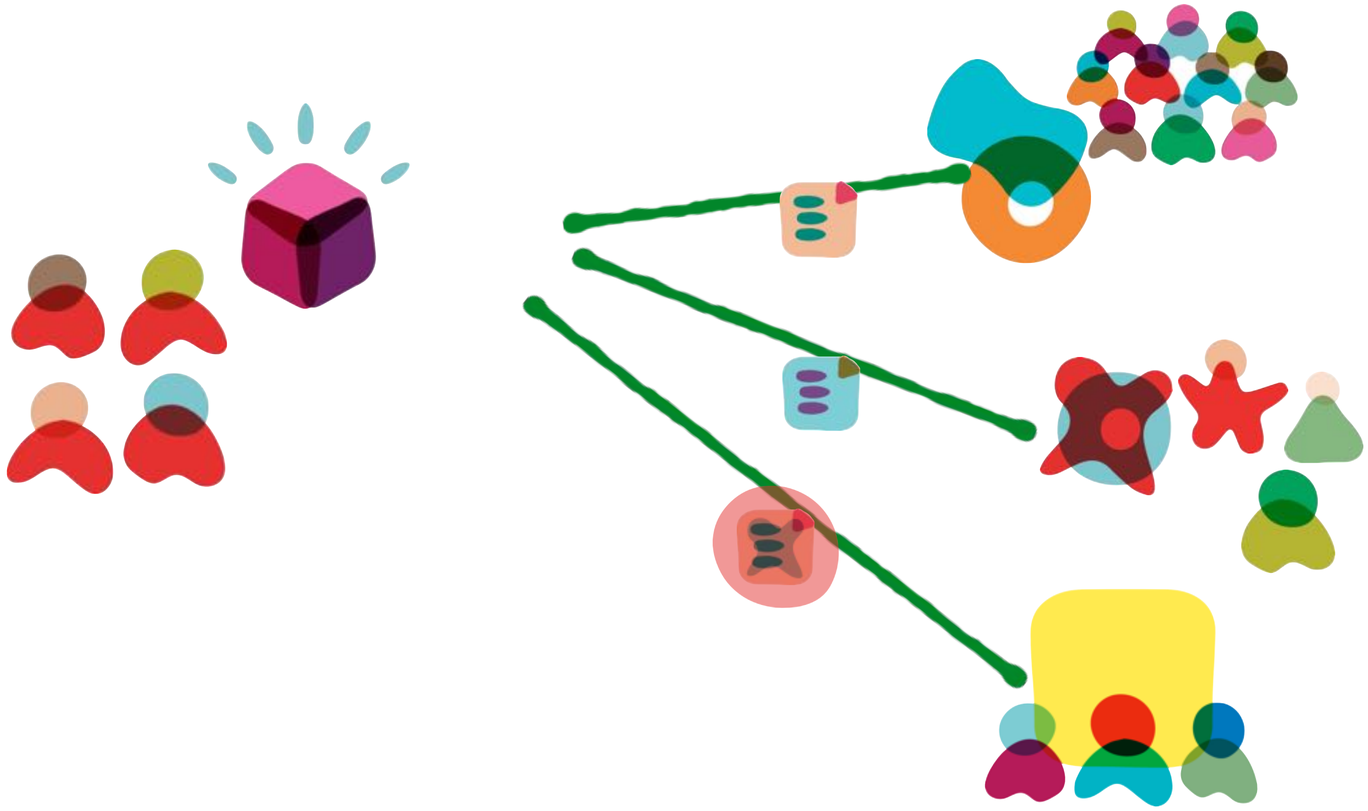
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



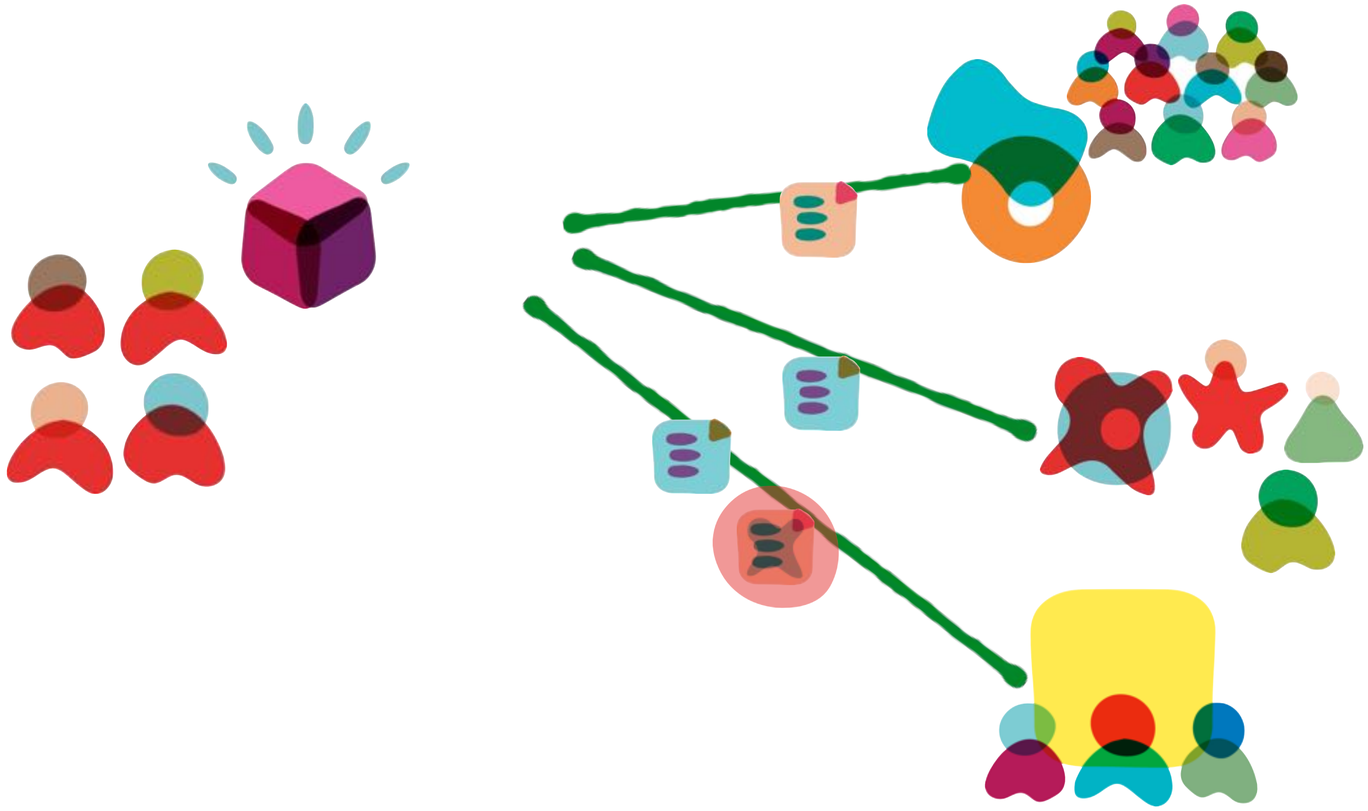
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



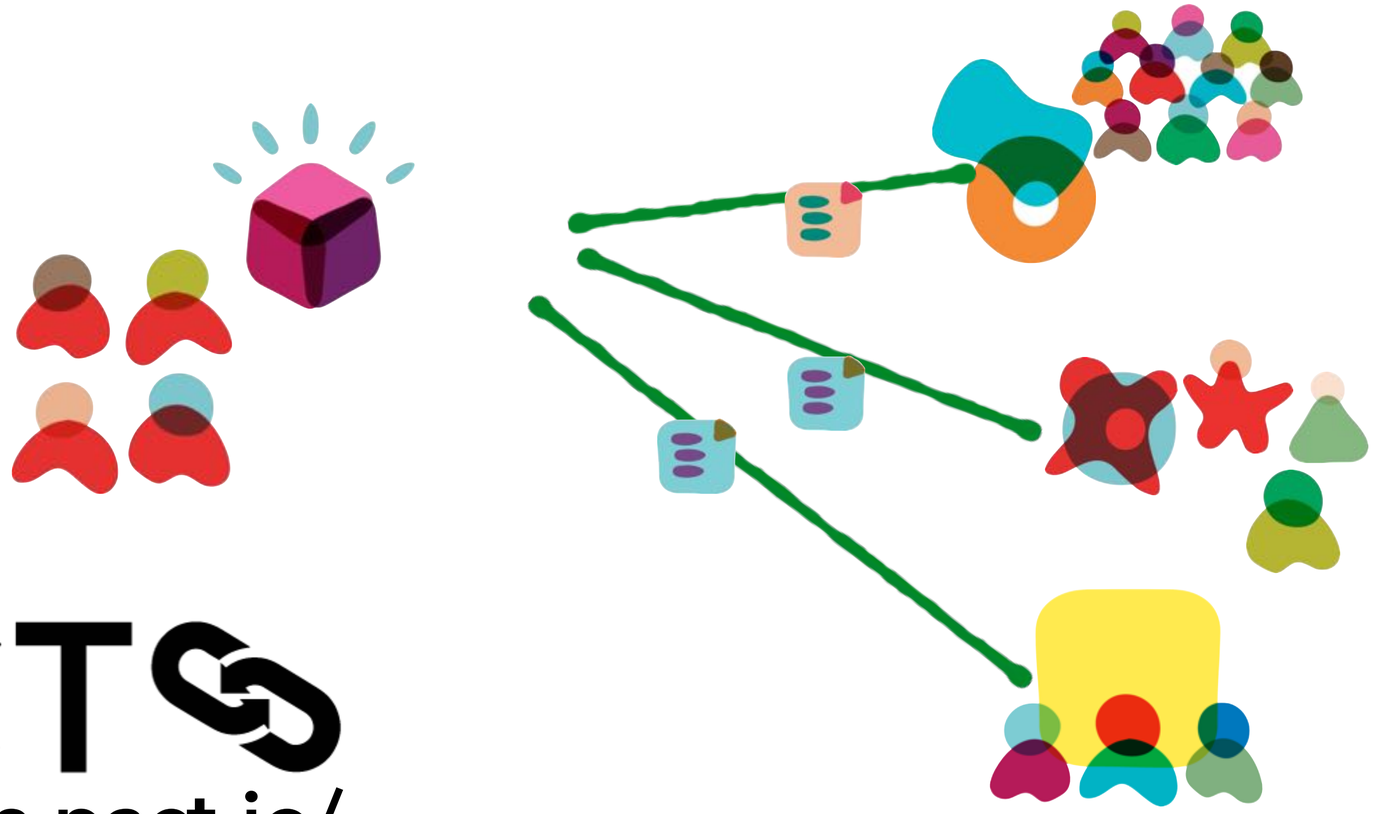
martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function



martinfowler.com/articles/consumerDrivenContracts.html

contract fitness function

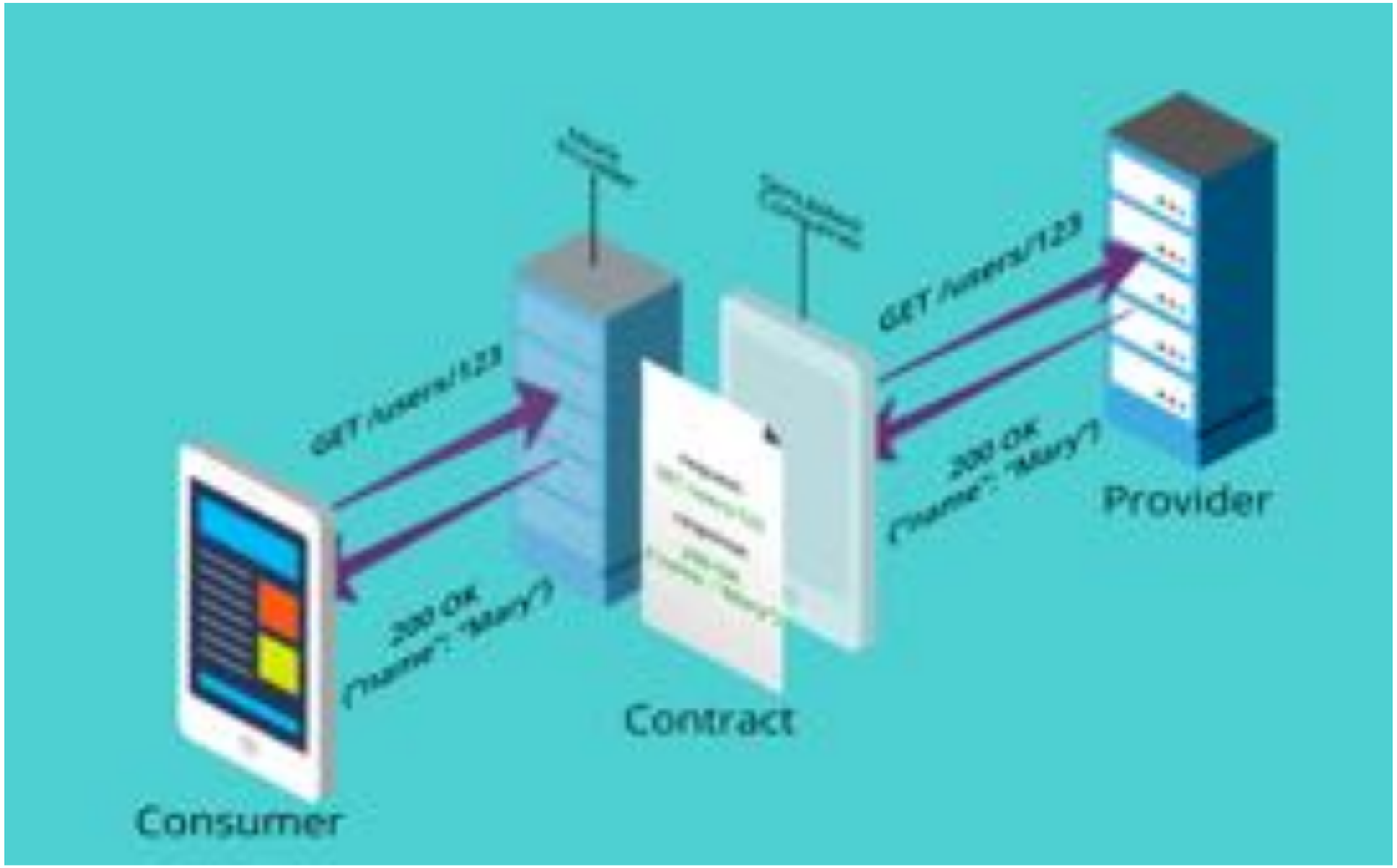


PACT

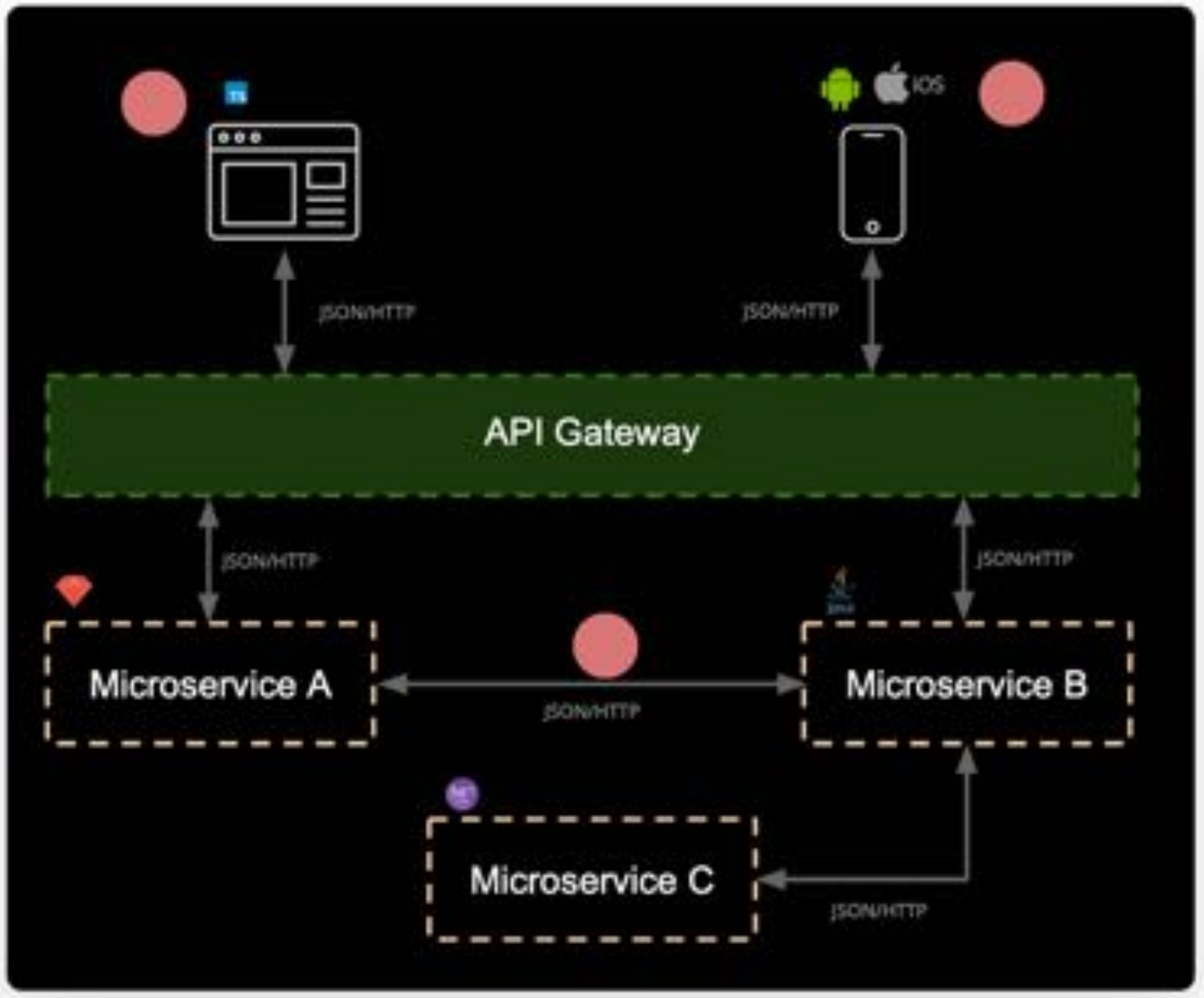
https://docs.pact.io/

martinfowler.com/articles/consumerDrivenContracts.html

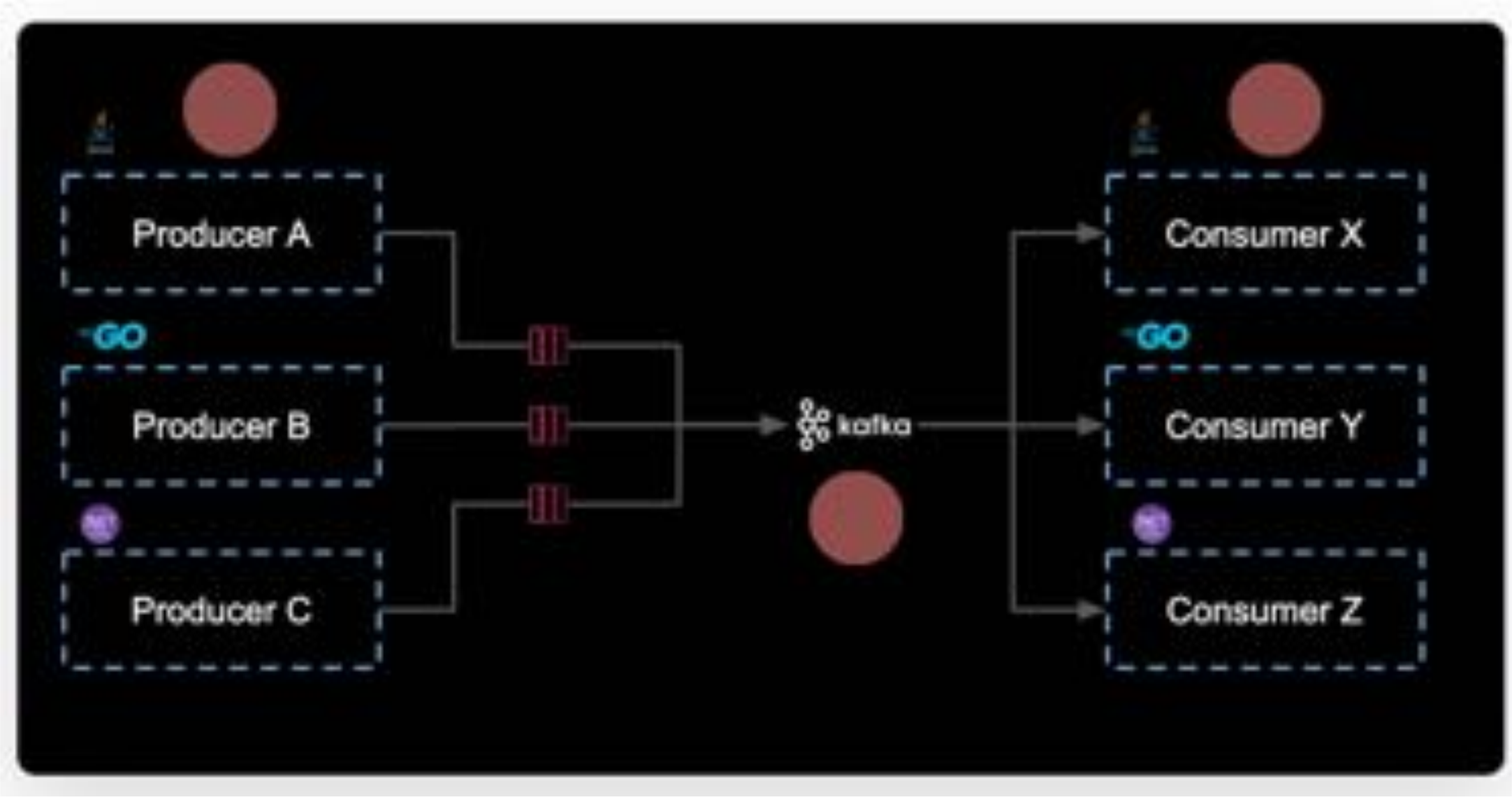
https://docs.pact.io



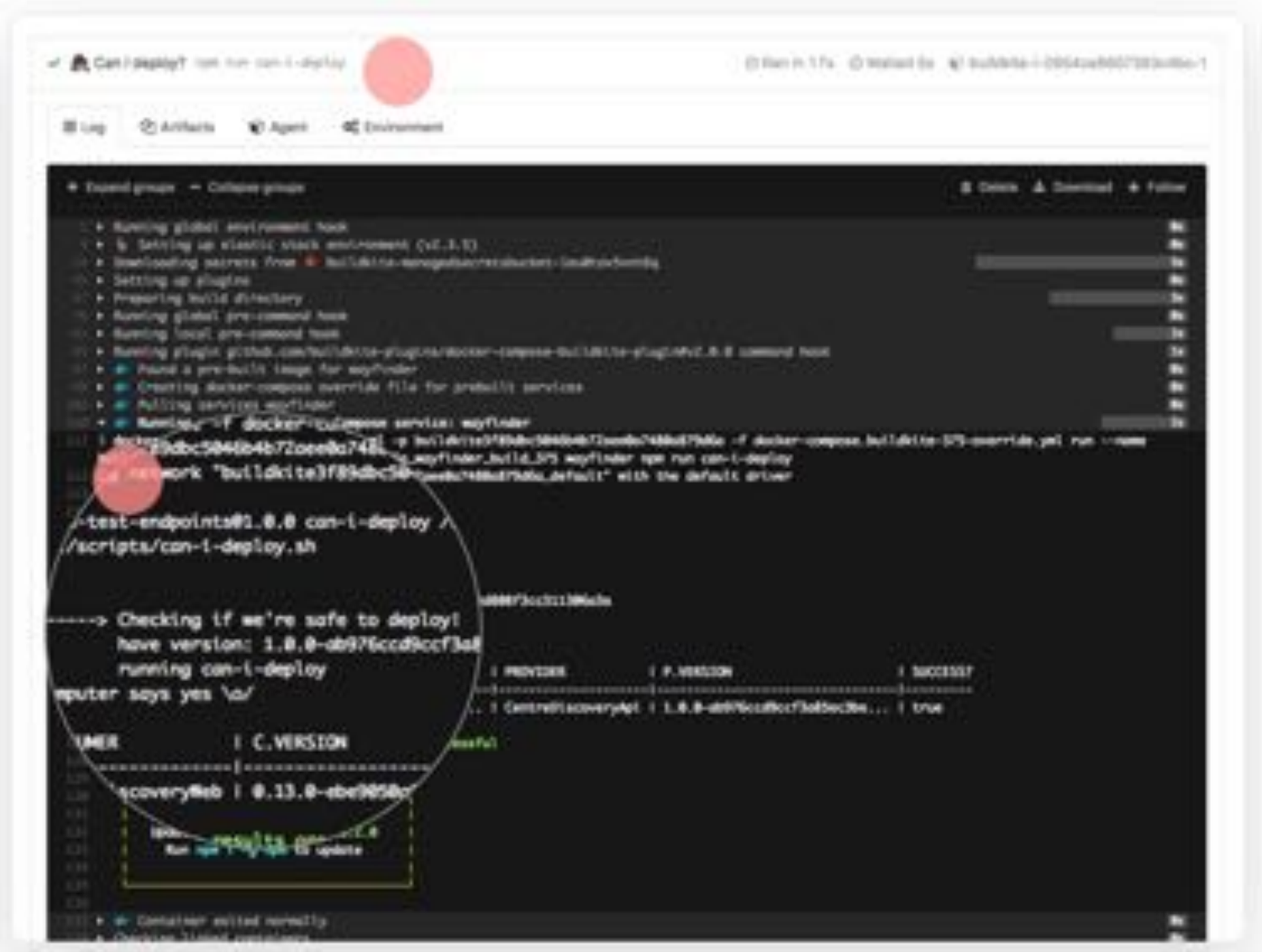
Pact + REST


iOS
JSON/HTTP
JSON/HTTP
API Gateway
JSON/HTTP
JSON/HTTP
Microservice A
JSON/HTTP
Microservice B
Microservice C
JSON/HTTP

Pact + EDA


Producer A
GO
Producer B
Producer C
kafka
Consumer X
GO
Consumer Y
Consumer Z

Pact + CI/CD


Log
Artifacts
Agent
Environment
Checking if we're safe to deploy!
have version: 1.0.0-ab976ccd9ccf3a8
running can-i-deploy
PROVIDER
P.VERSION
SUCCESS
C.VERSION

strict contracts

– brittle integration points

– requires versioning

+ guaranteed type fidelity

value-based contracts

– less certainty in contracts (requires fitness functions)

– requires more developer discipline

+ extremely loose coupling

+ immune from implementation change

tradeoffs

strict contracts **OR** value-based contracts

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

* requires fitness functions
* requires documentation
* extremely loose coupling

Kata Exercise - Semantic versus Syntactic Coupling

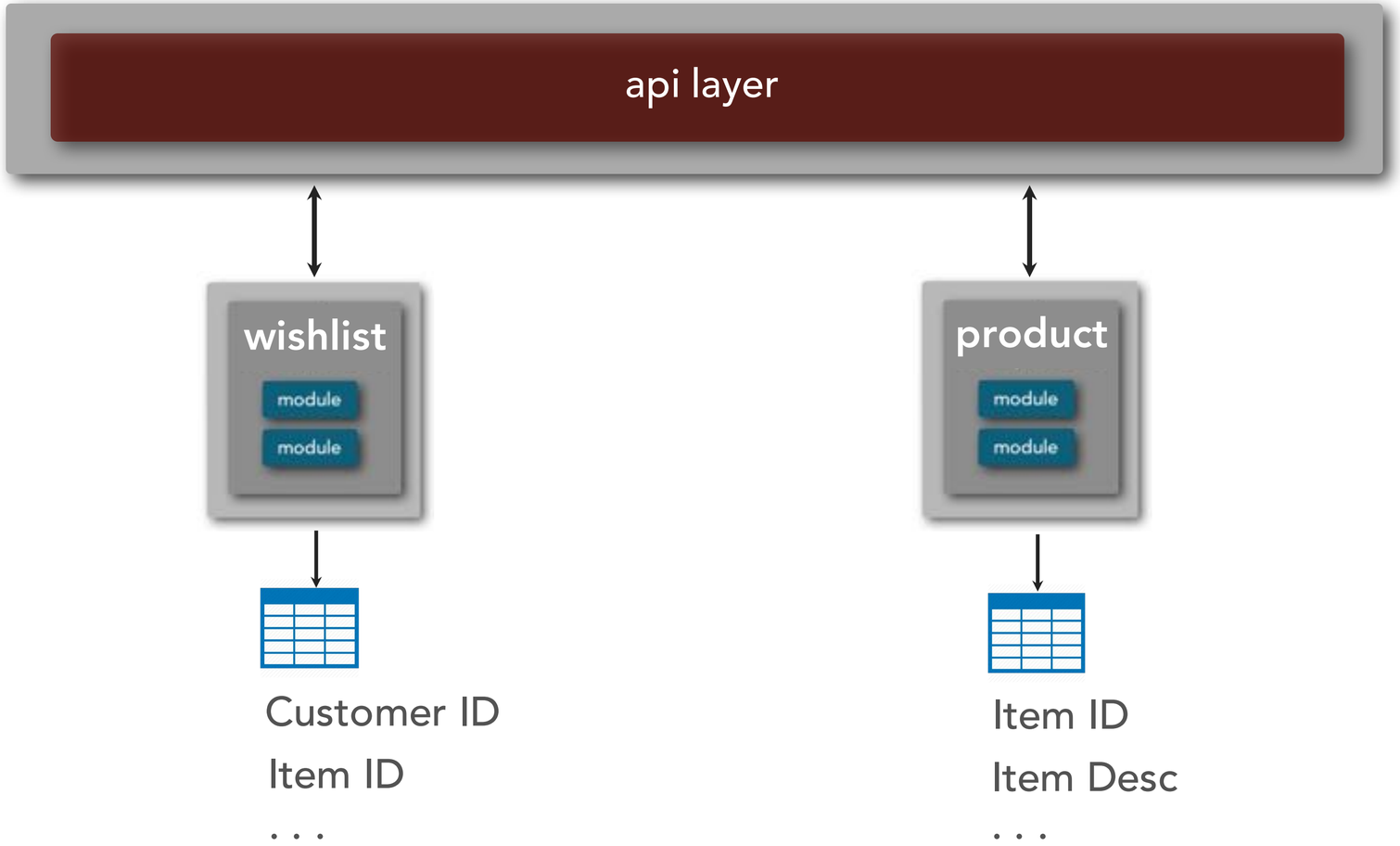
For each quanta identified earlier, determine:

1. What is the level of semantic coupling within and between quanta? (Is it time to refactor?)
2. For each quanta, determine the strictness of contracts needed and the tradeoffs for each choice.
3. For loose contract, implement fitness functions to restore the necessary semantic coupling contract.

How do I manage data in a distributed system?

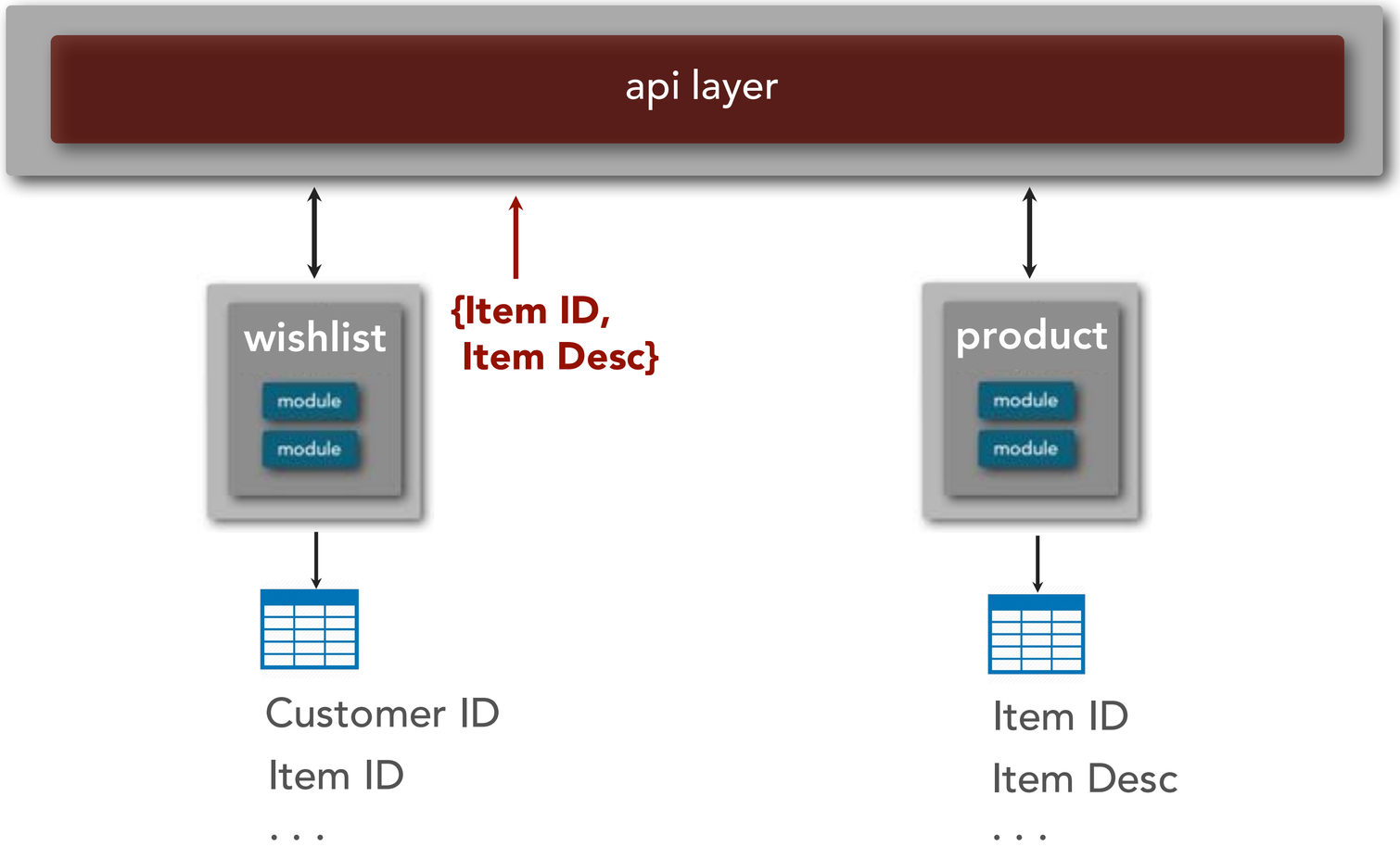
data access and sharing

the issue...



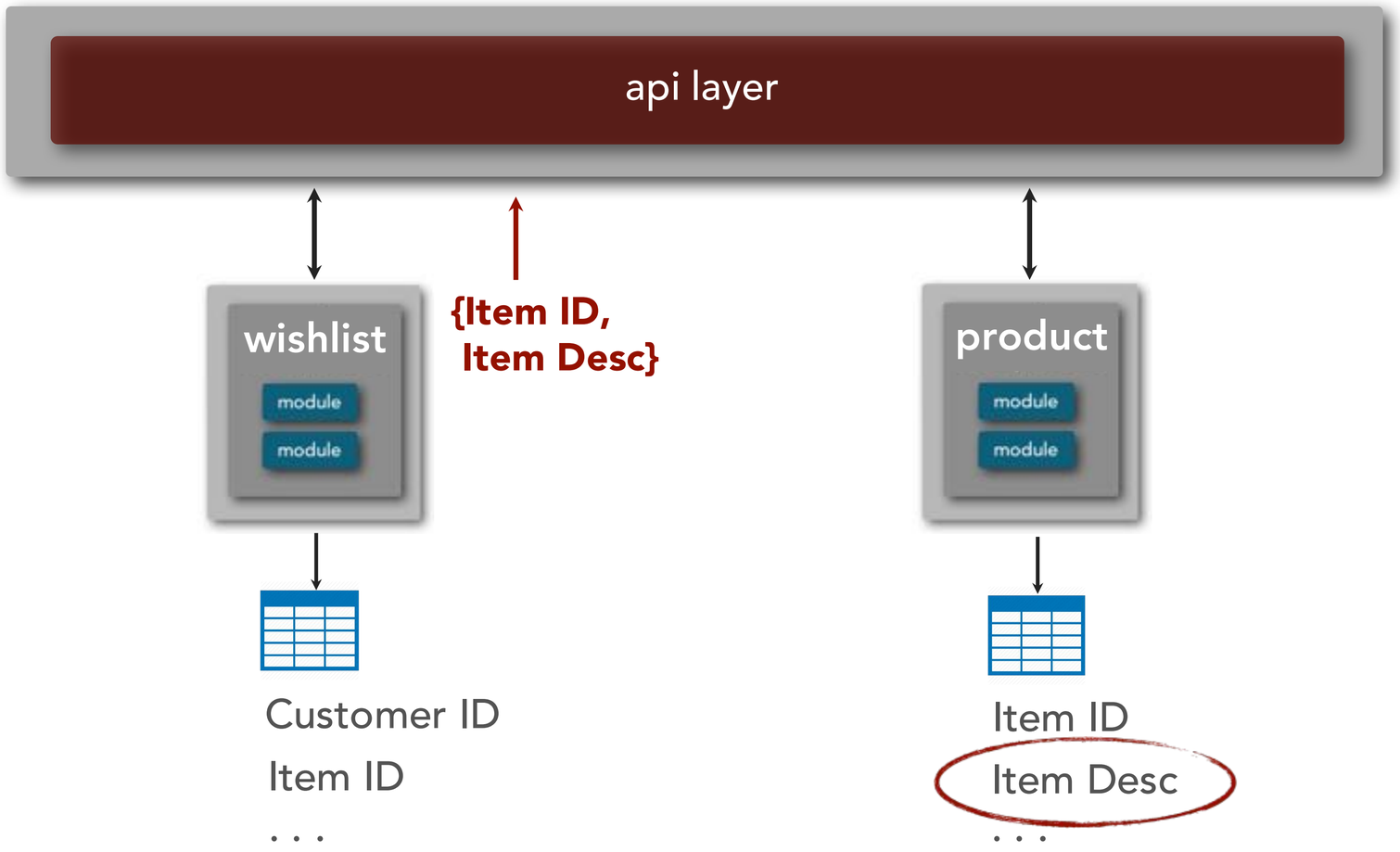
data access and sharing

the issue...



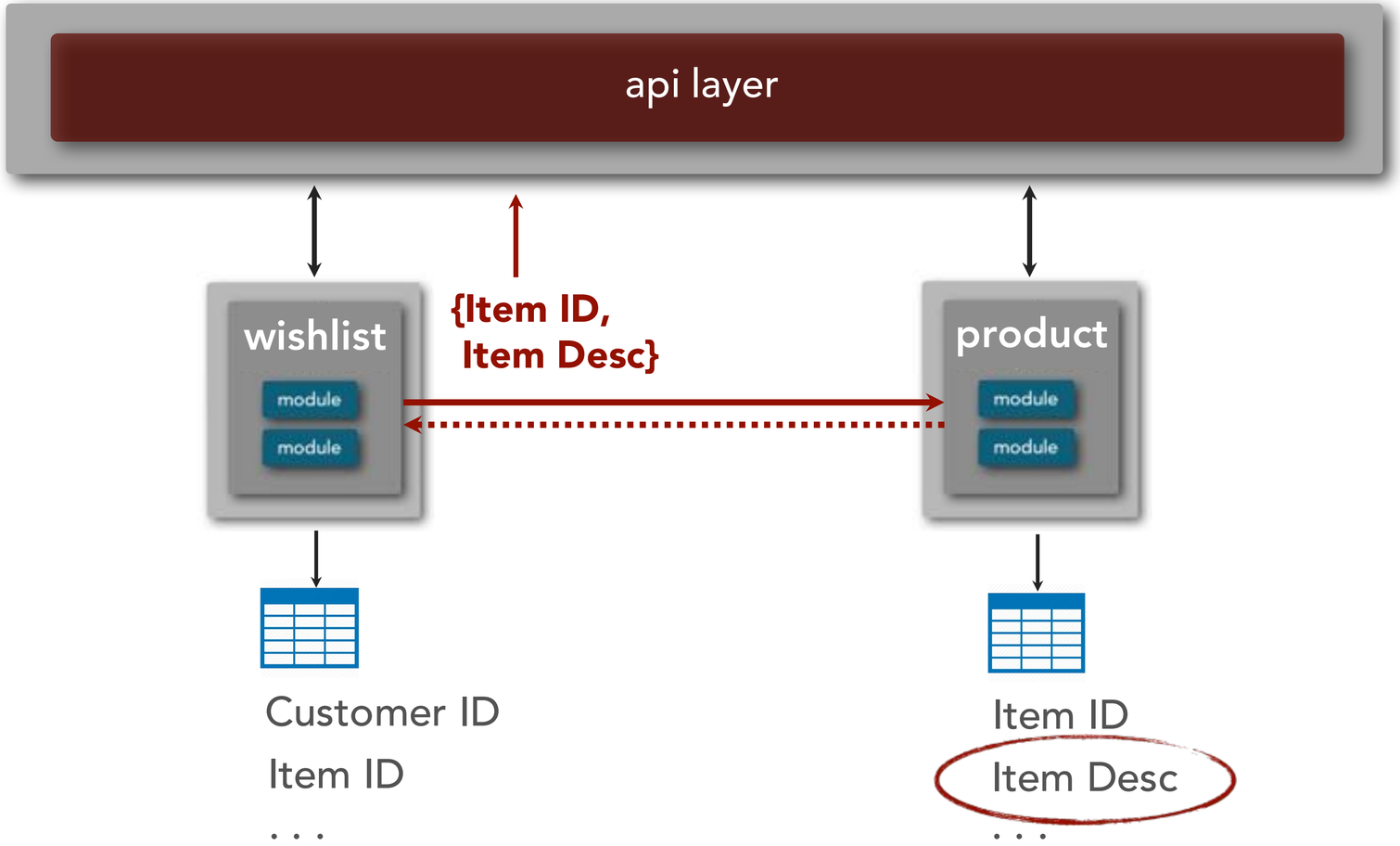
data access and sharing

the issue...



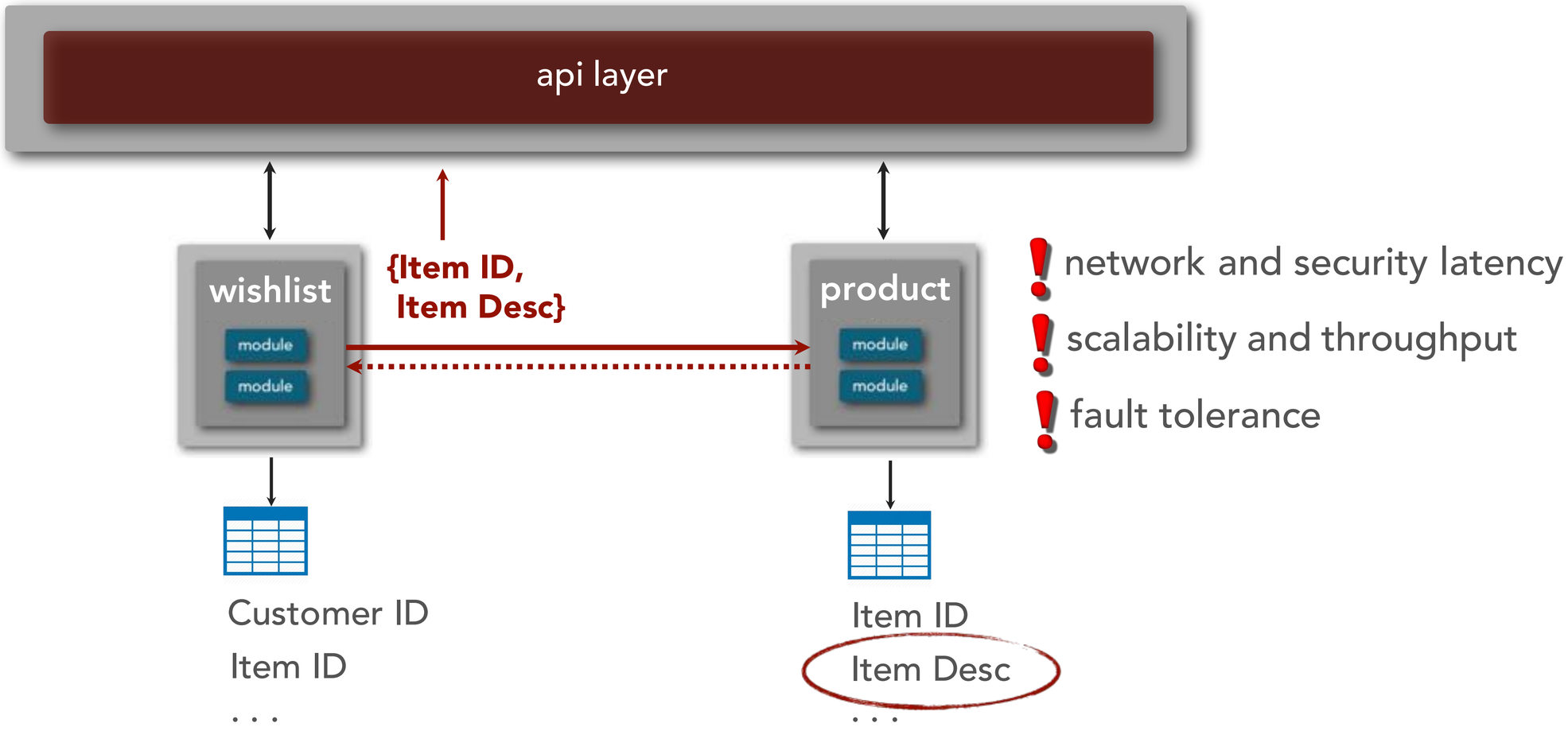
data access and sharing

option 1: interservice communication



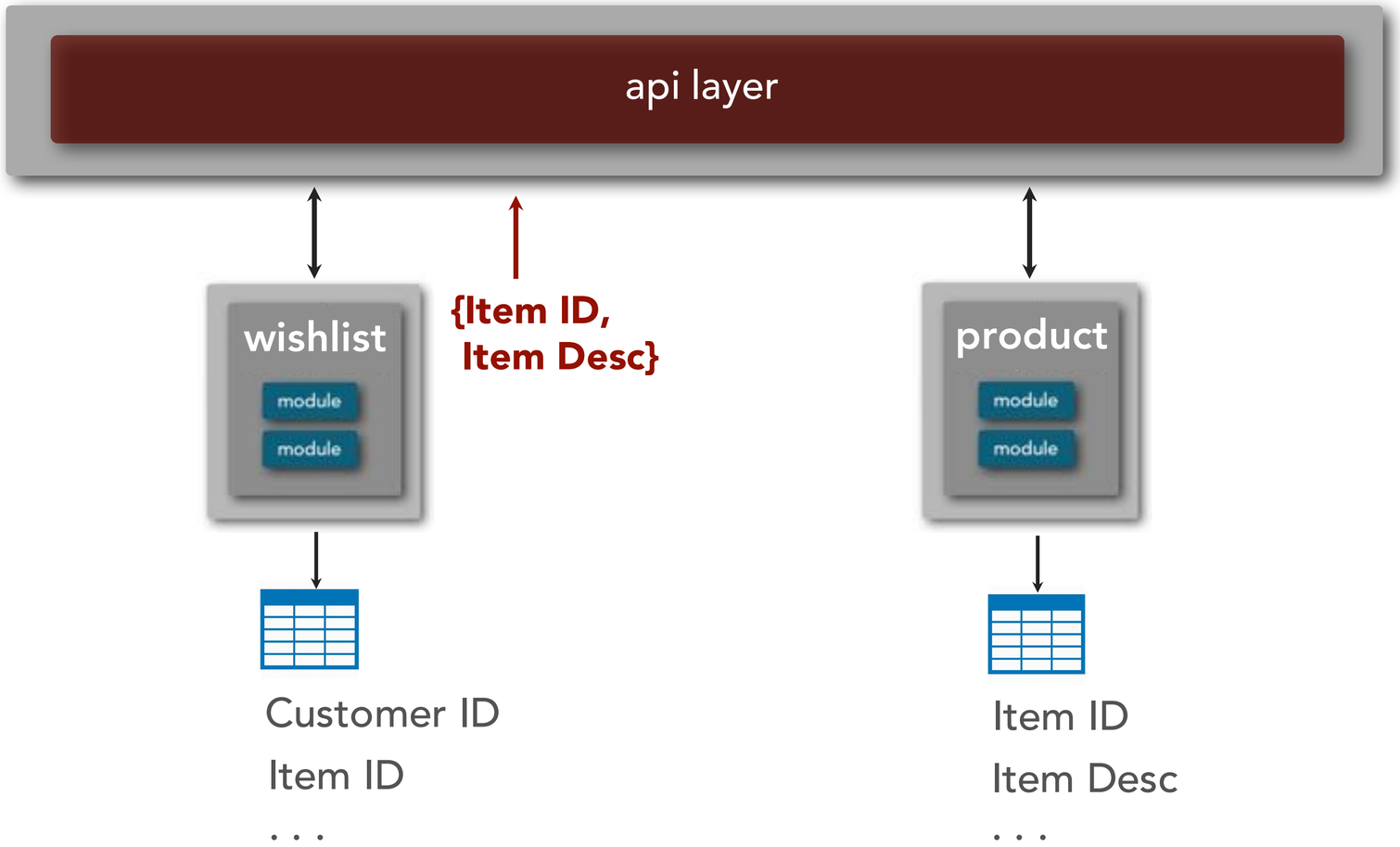
data access and sharing

option 1: interservice communication



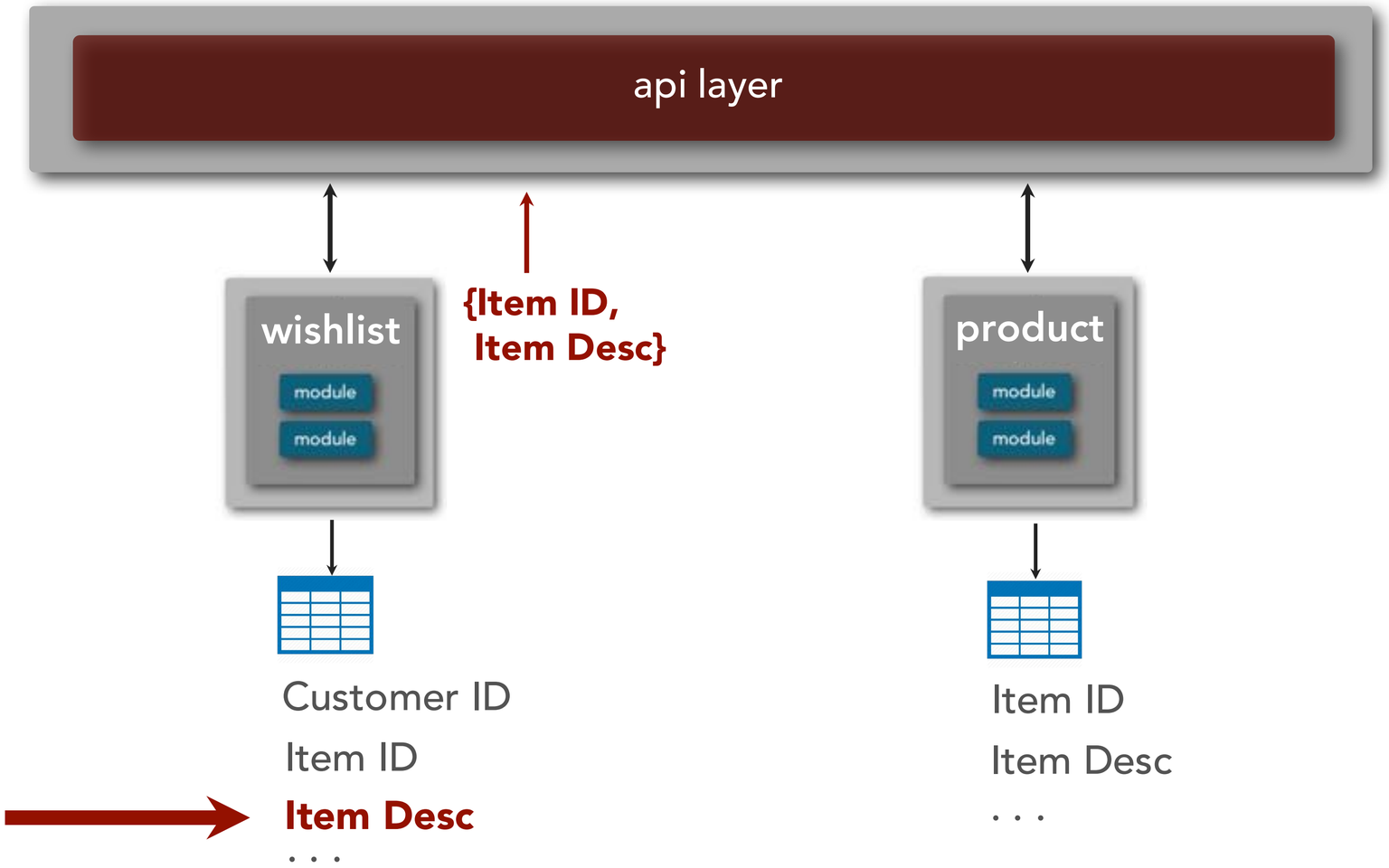
data access and sharing

option 2: data replication



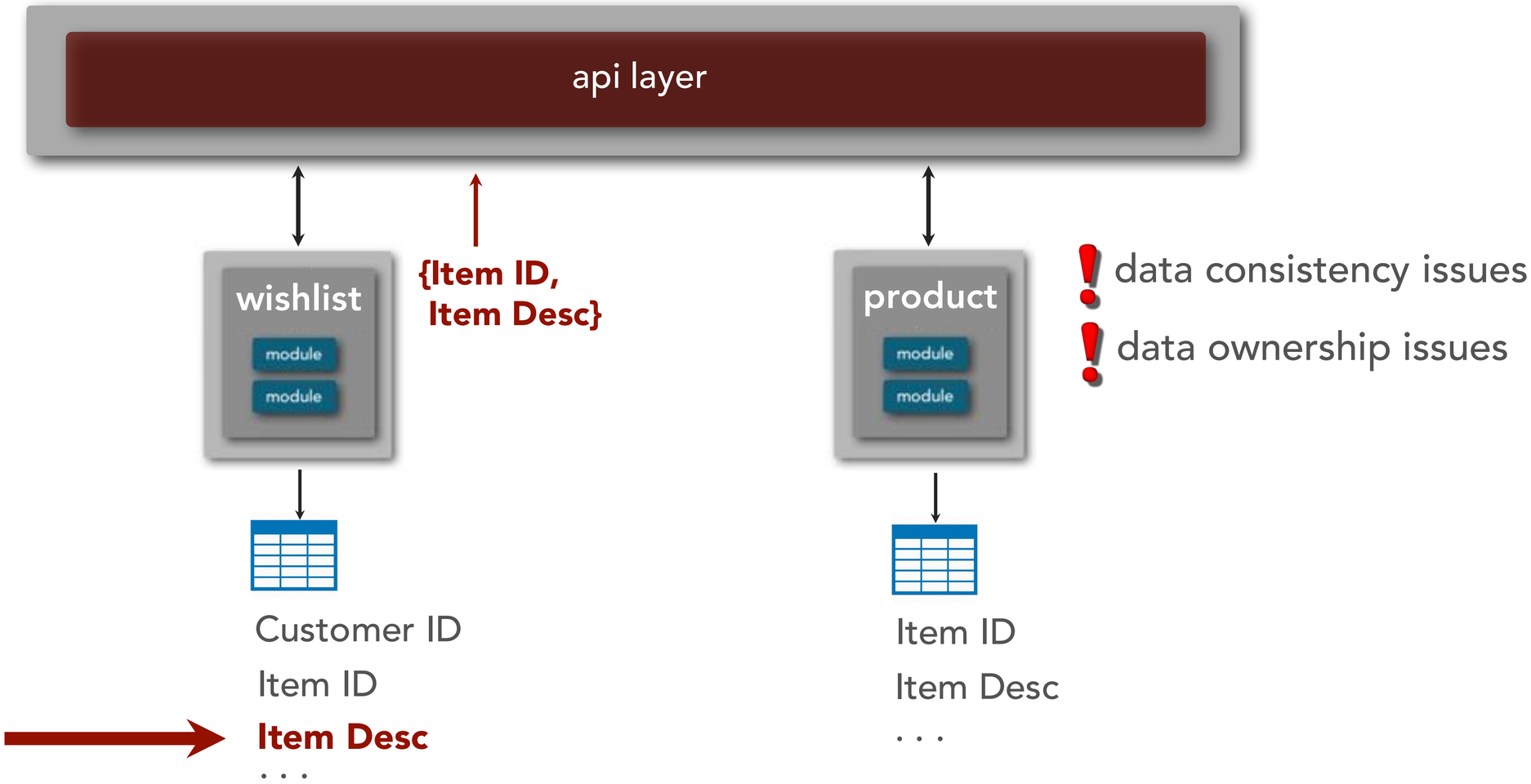
data access and sharing

option 2: data replication



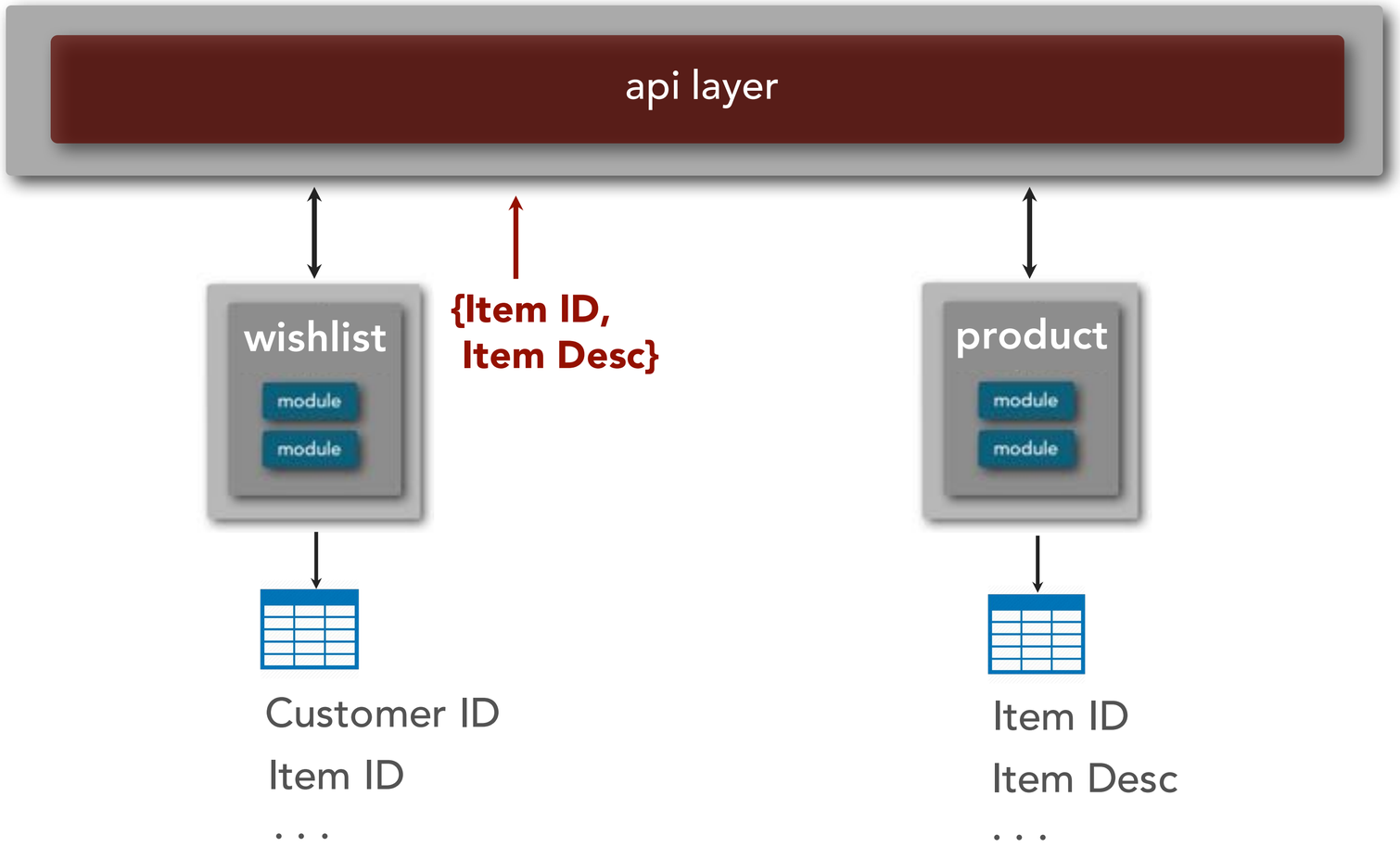
data access and sharing

option 2: data replication



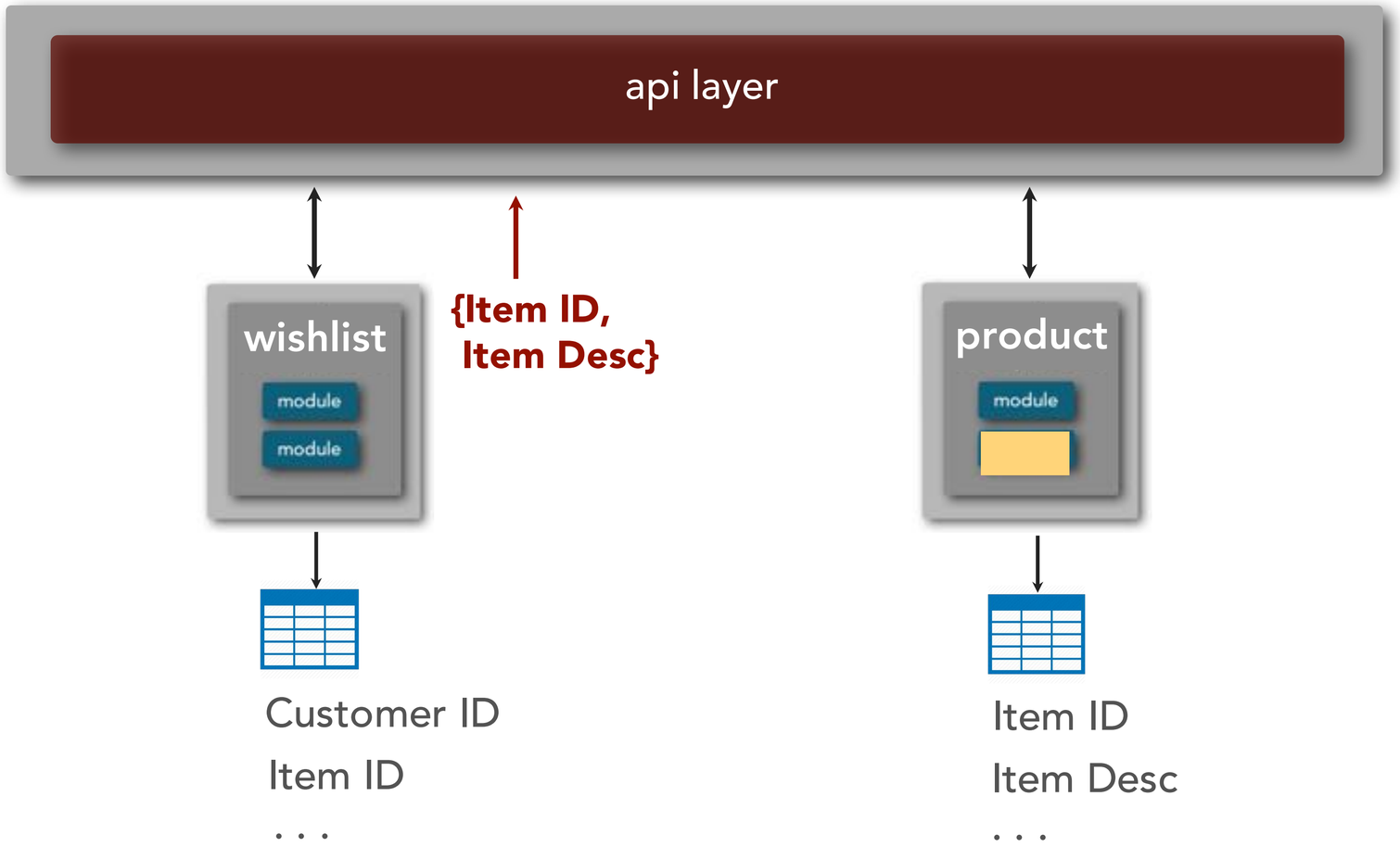
data access and sharing

option 3: in-memory replicated cache



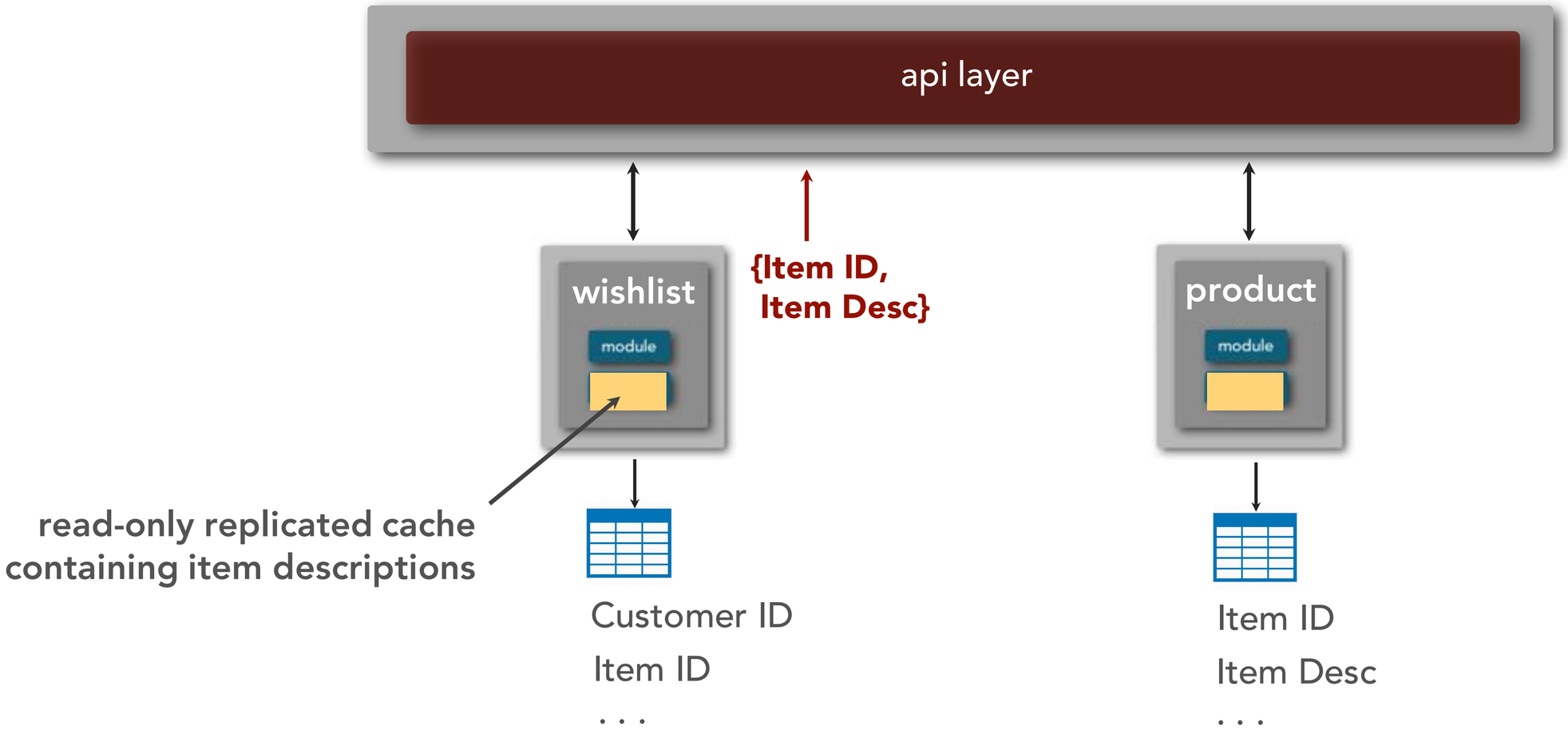
data access and sharing

option 3: in-memory replicated cache



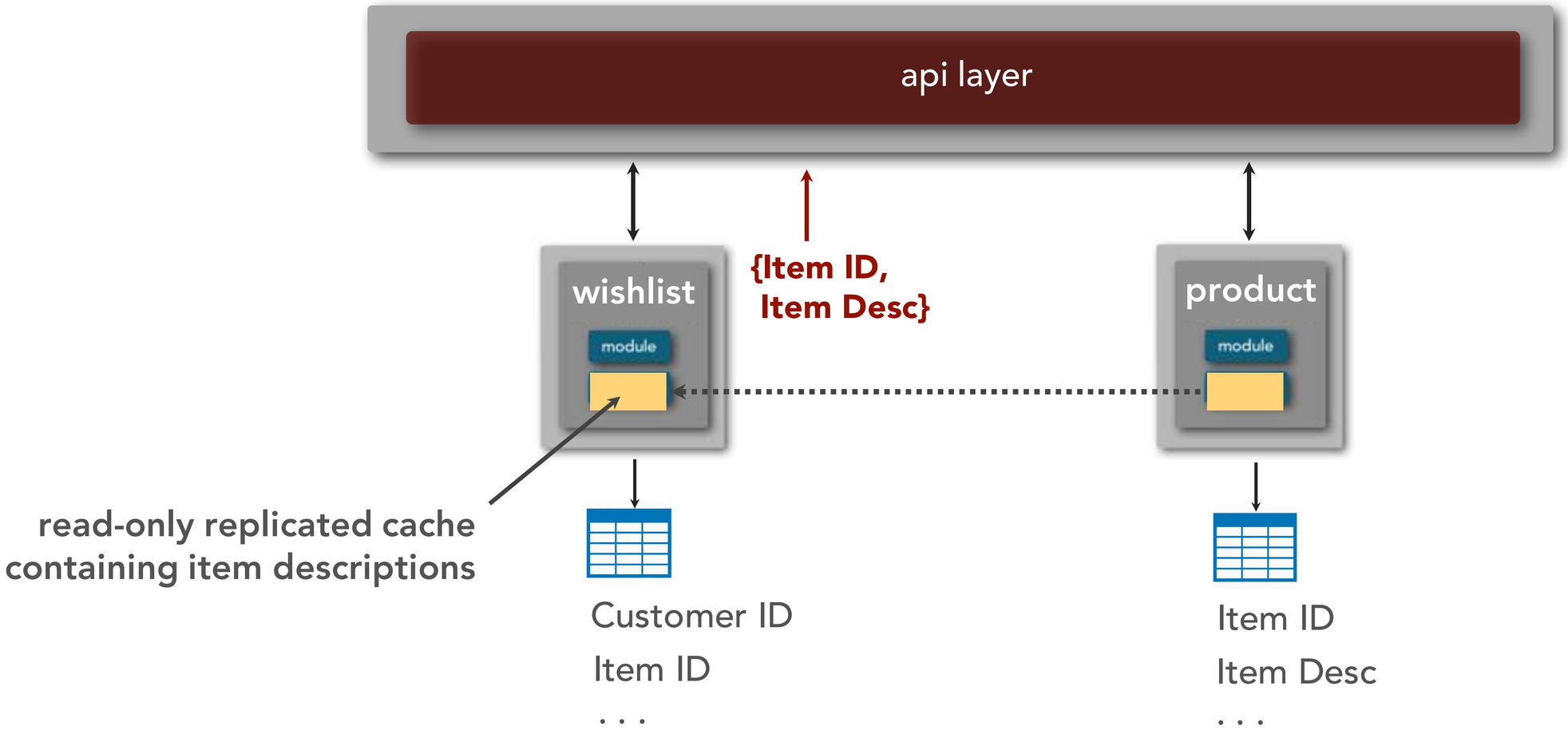
data access and sharing

option 3: in-memory replicated cache



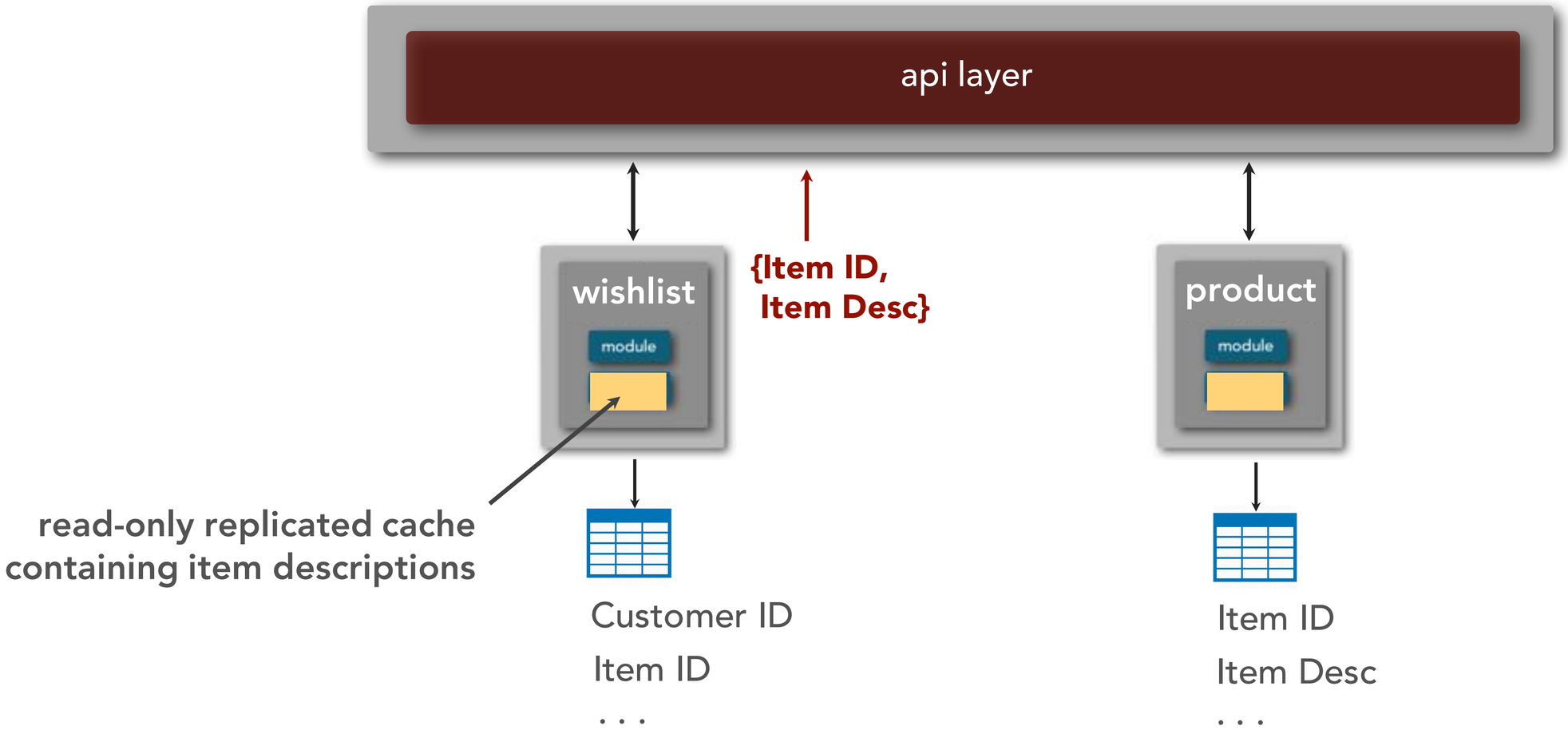
data access and sharing

option 3: in-memory replicated cache



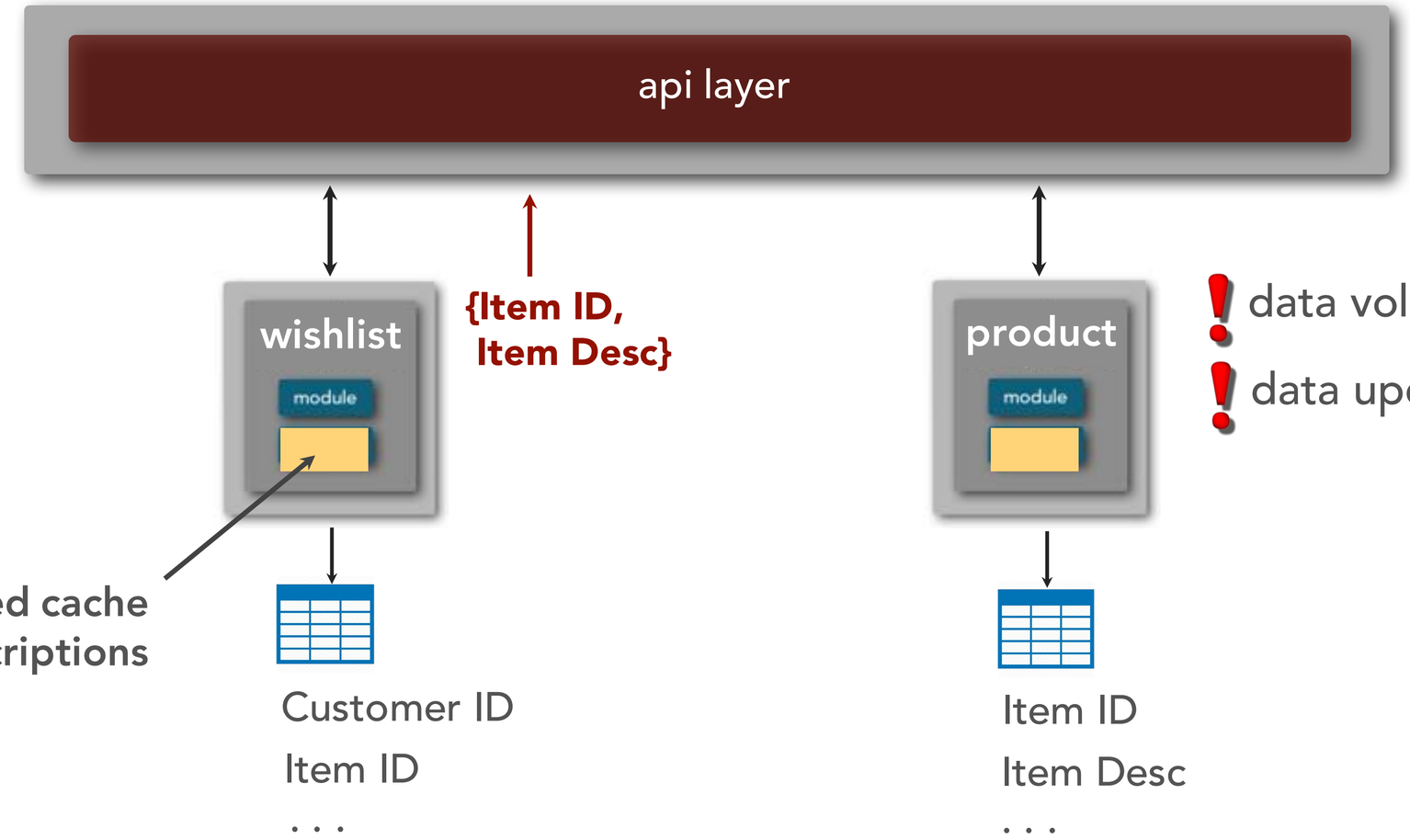
data access and sharing

option 3: in-memory replicated cache



data access and sharing

option 3: in-memory replicated cache

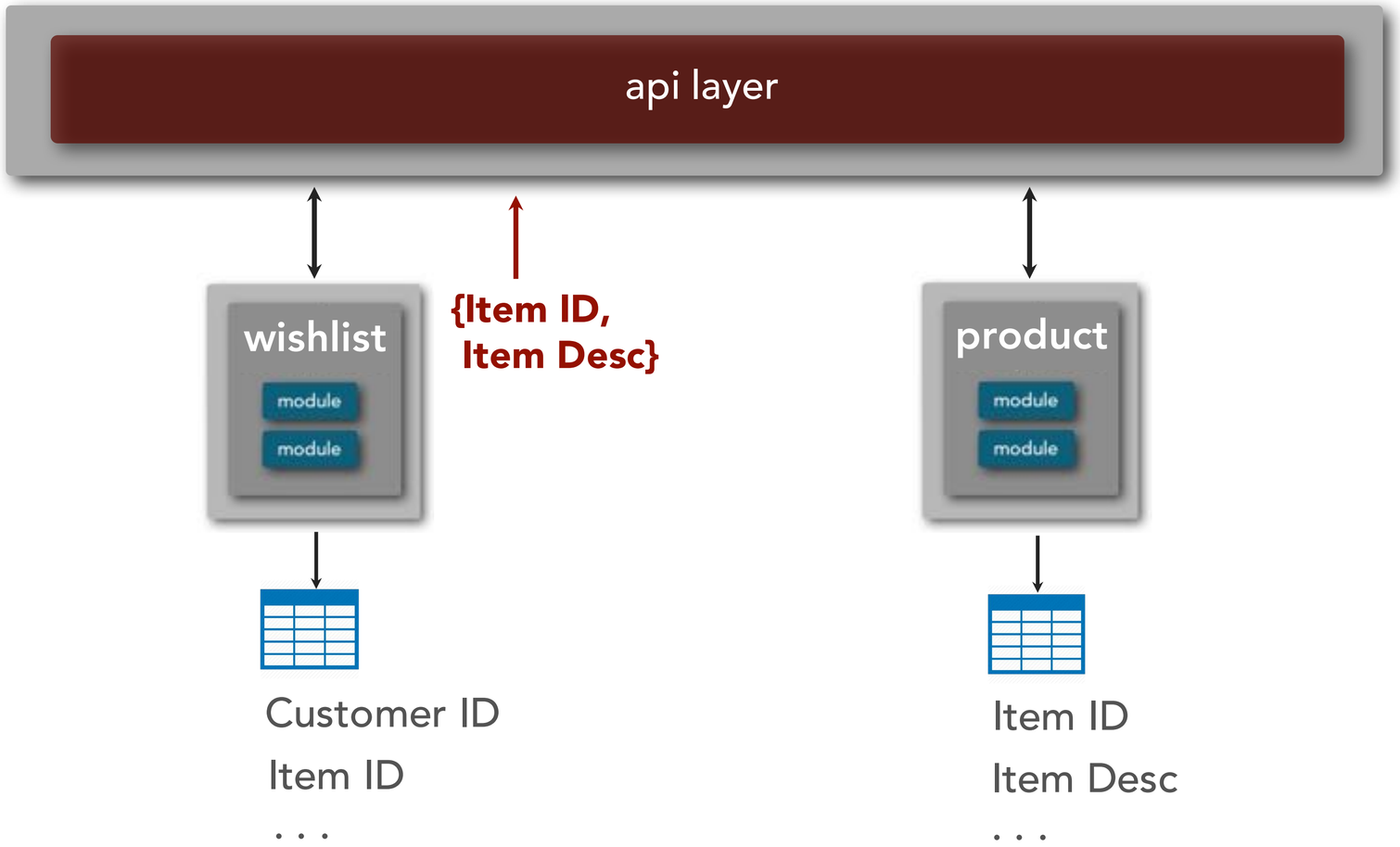


! data volume issues

! data update rate issues

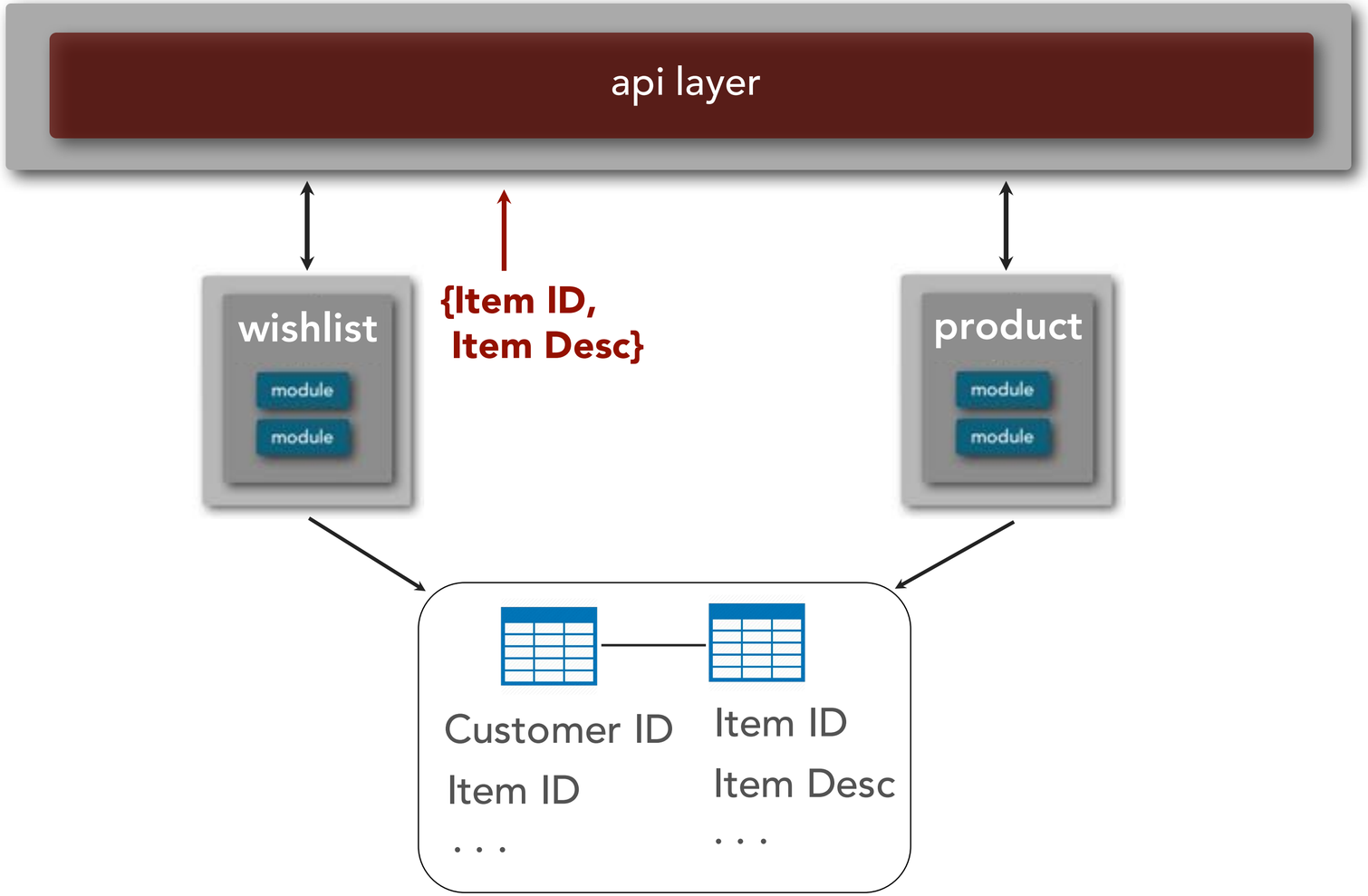
data access and sharing

option 4: data domain



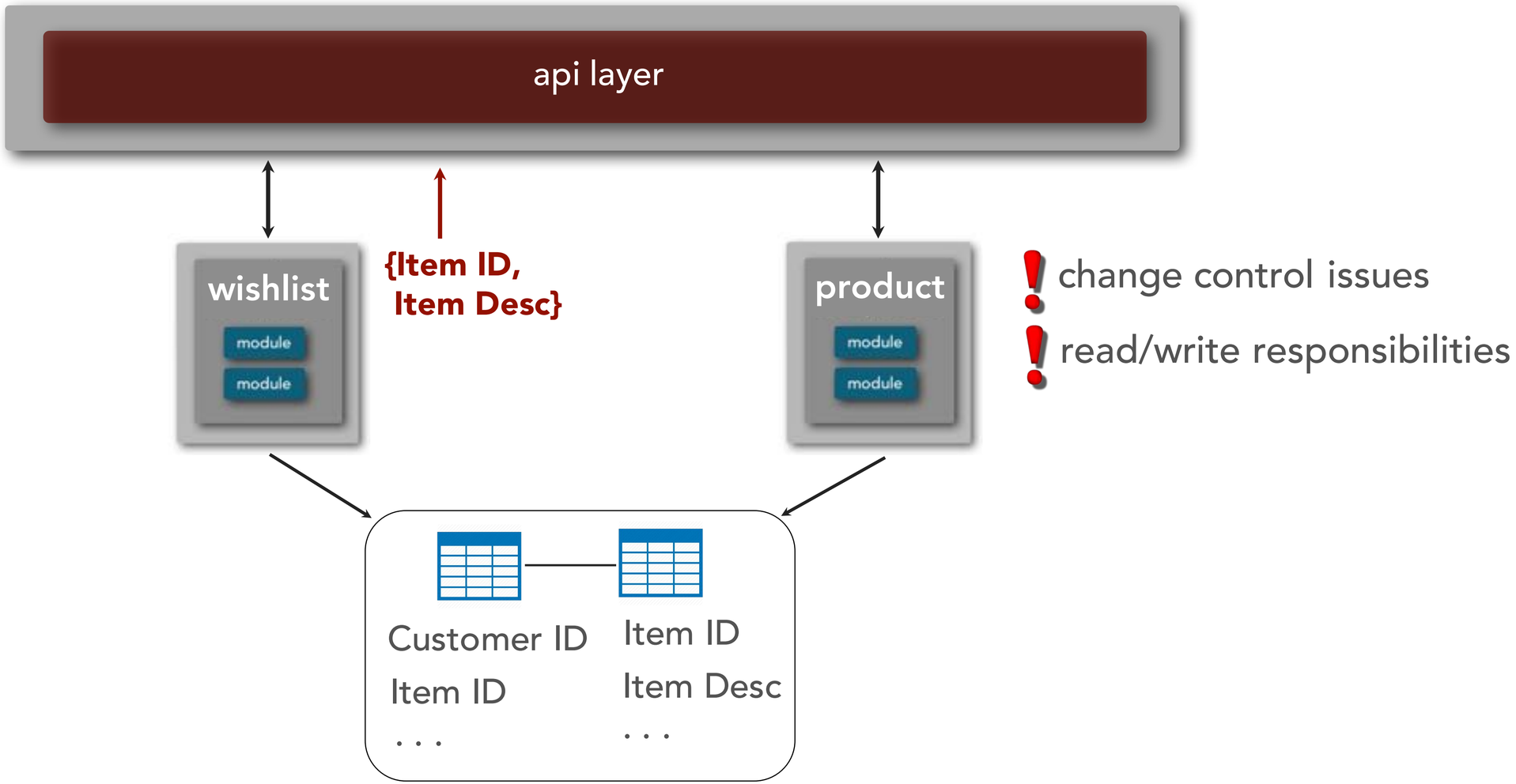
data access and sharing

option 4: data domain



data access and sharing

option 4: data domain



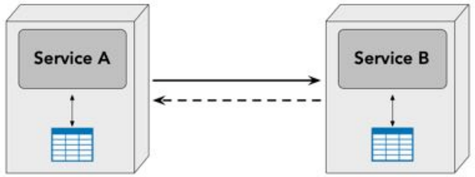
! change control issues

! read/write responsibilities

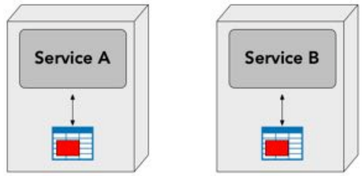
data access and sharing

which one is better?

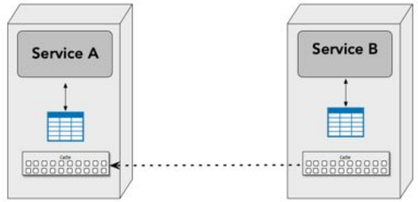
option 1:
interservice
communication



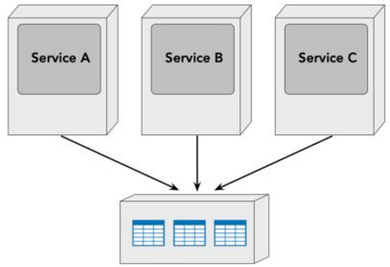
option 2:
data schema
replication



option 3:
in-memory
replicated cache

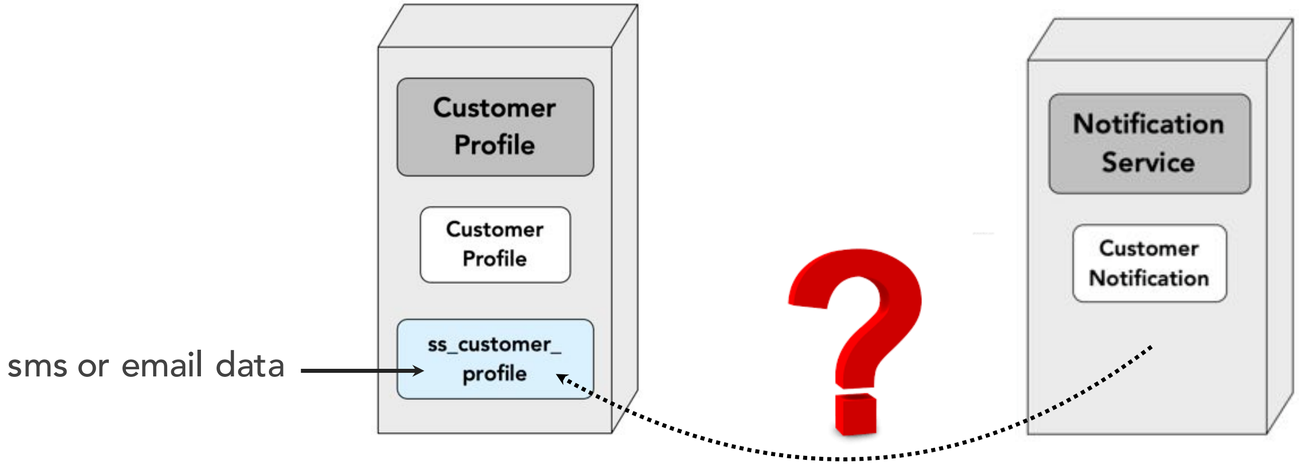


option 4:
data domains
(shared tables)



Kata Exercise - Data Access

The notification service needs the customers sms or email to notify them that the expert is on their way. How should the notification service get this information? (the notification service has the customer id)

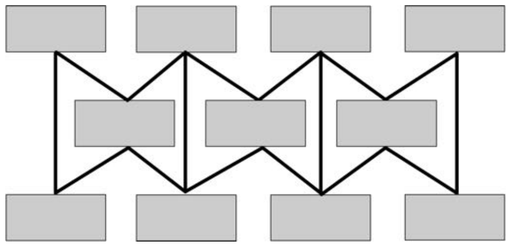


How do I achieve high levels of scalability and elasticity in a system?

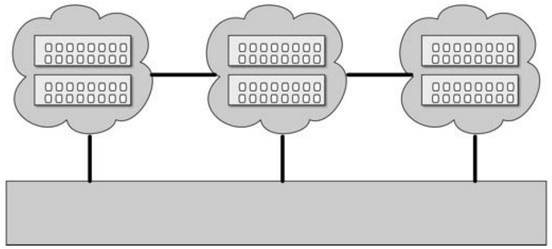
scenario: ↑ volume concert □'s

- Lots of concurrent users
- huge bursts of users when new shows go on sale
- highly transactional !

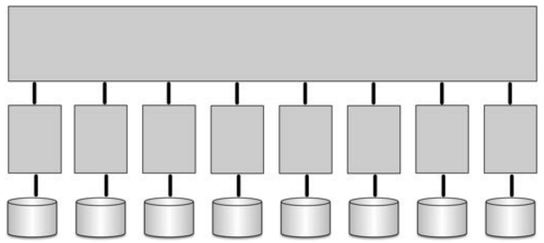
scalability ∪ elasticity



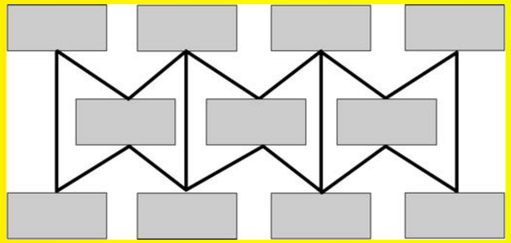
Event-driven architecture



Space-based architecture



Microservices architecture



– very complex for simple things
  (instancing, discovery, etc)

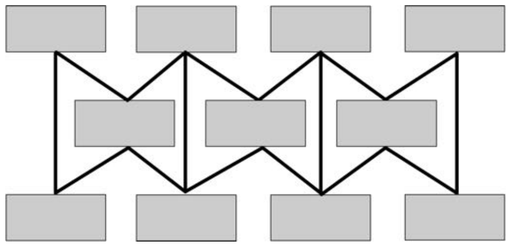
– traditional headaches of
  message-based systems

– granularity issues with elasticity

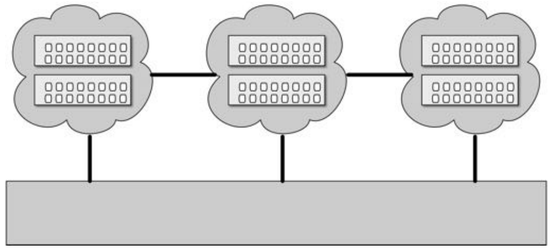
– too much build-it-yourself

+ highly performant and scalable

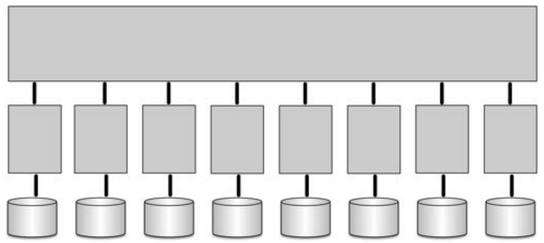
scalability ∪ elasticity



Event-driven architecture

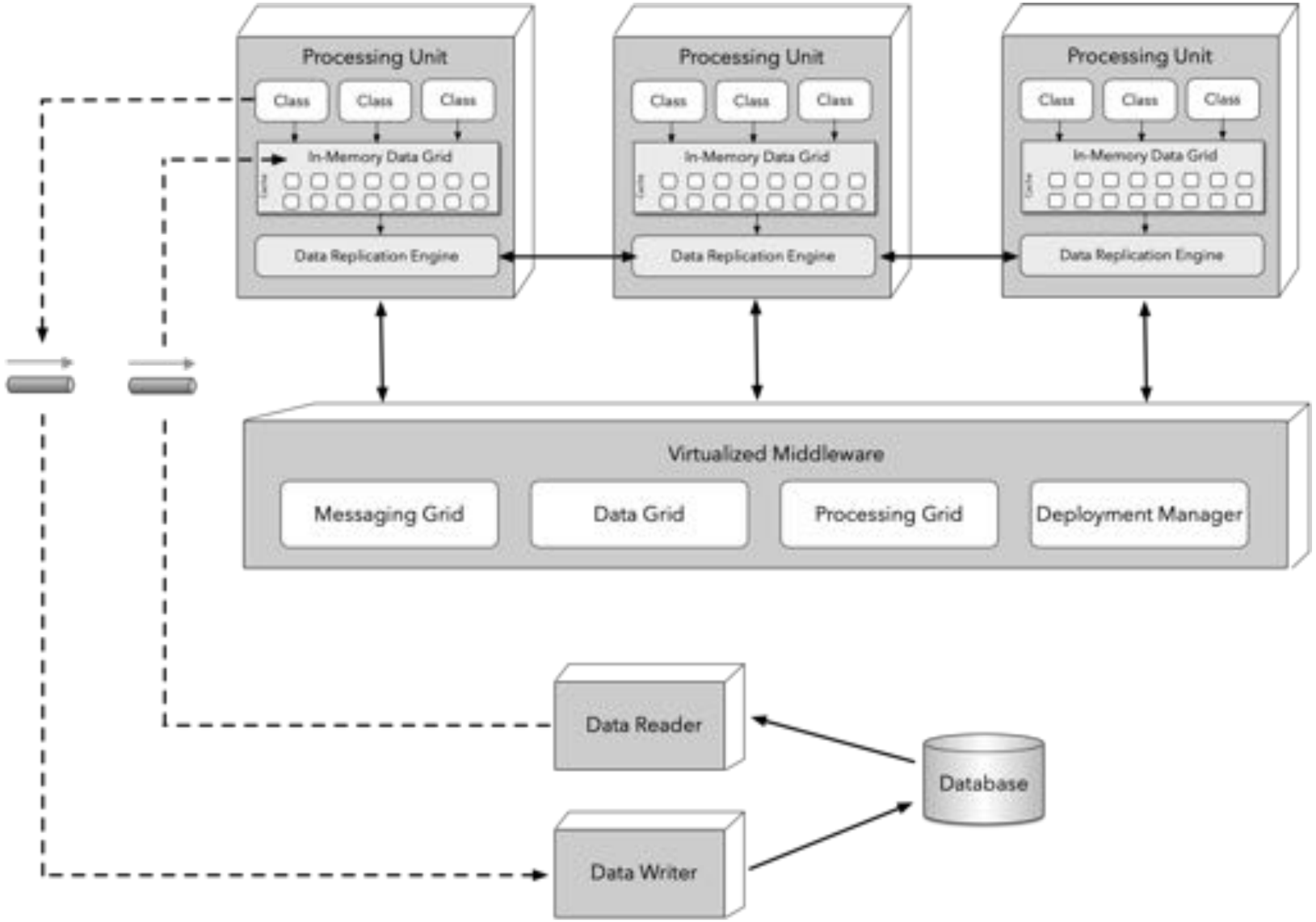


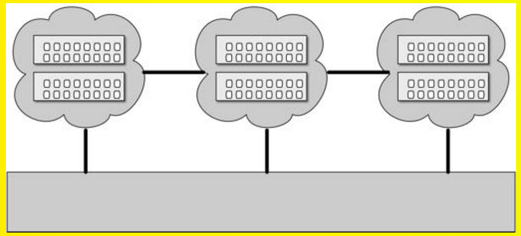
Space-based architecture



Microservices architecture

space-based architecture

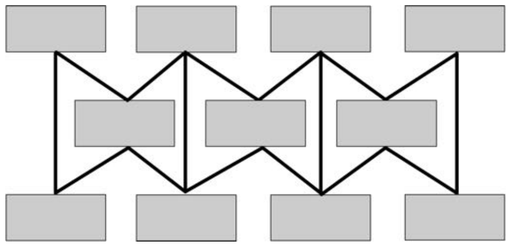




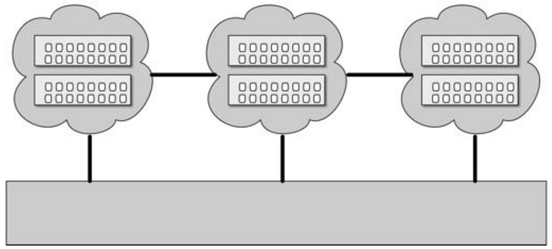
- architects must build transactional behavior
- highly operationally complex
- expensive
- higher incidental coupling because of technical partitioning

+ hyper-optimized for elastic scale

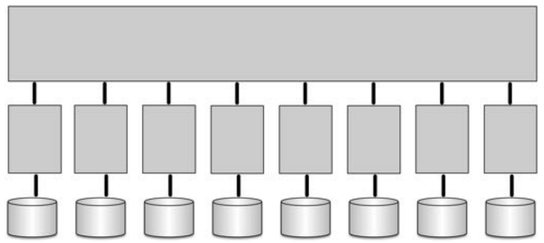
scalability ∪ elasticity



Event-driven architecture

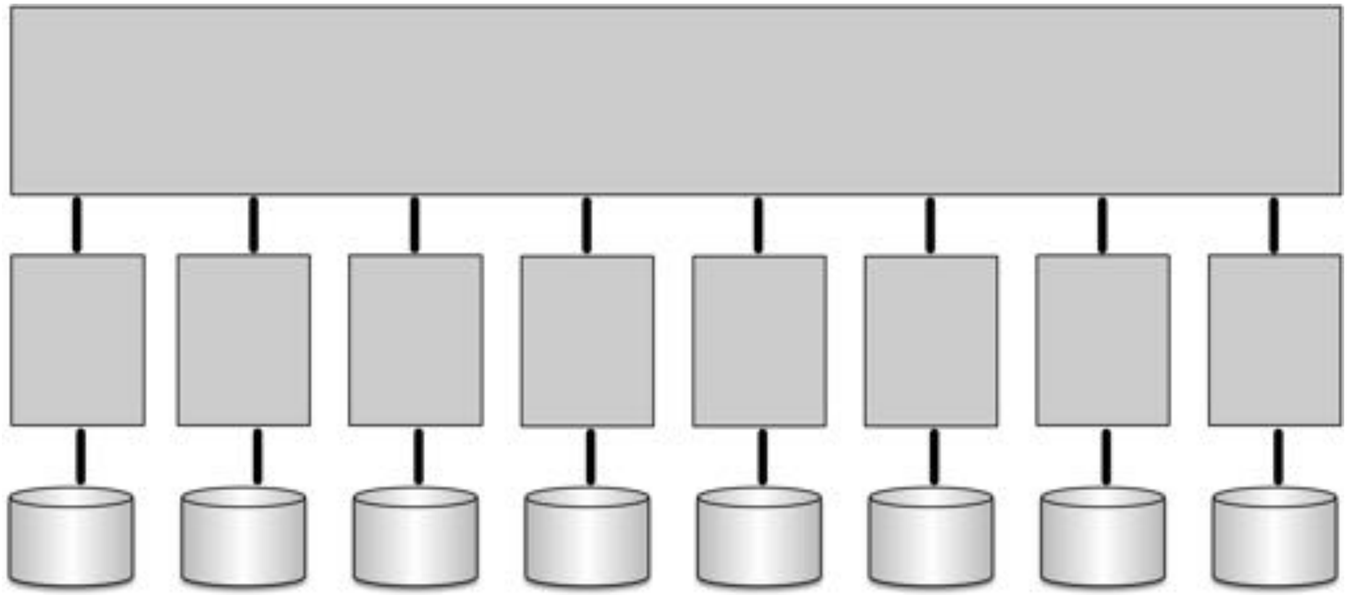


Space-based architecture

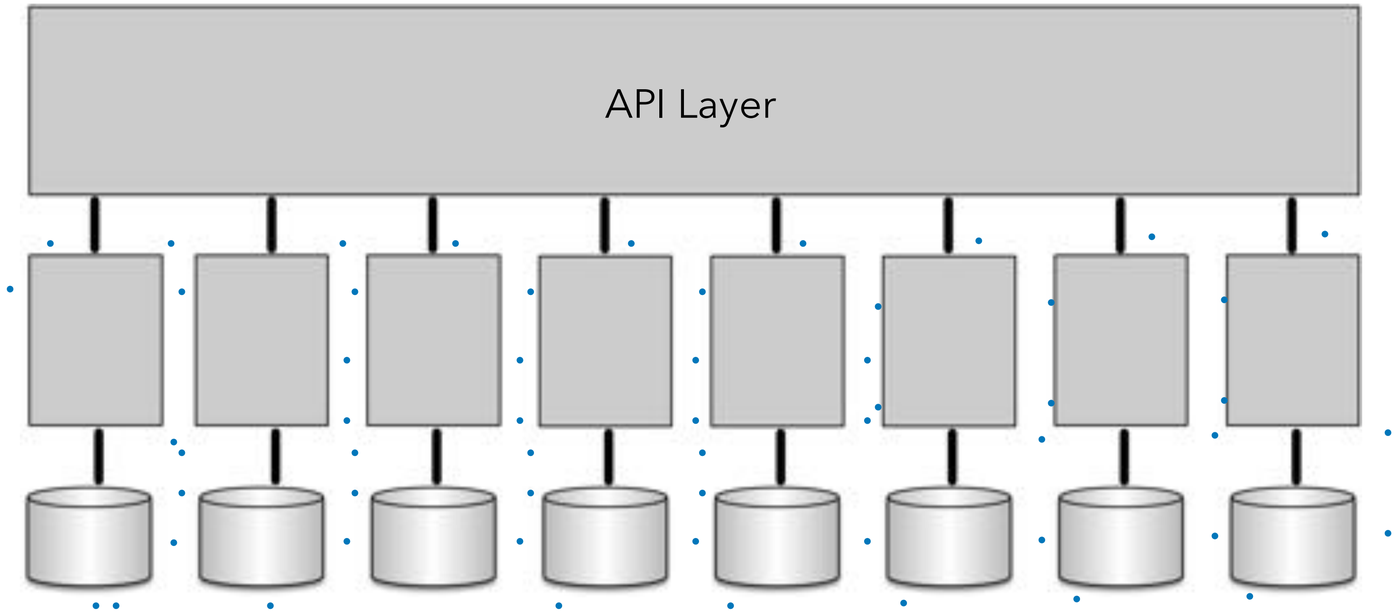


Microservices architecture

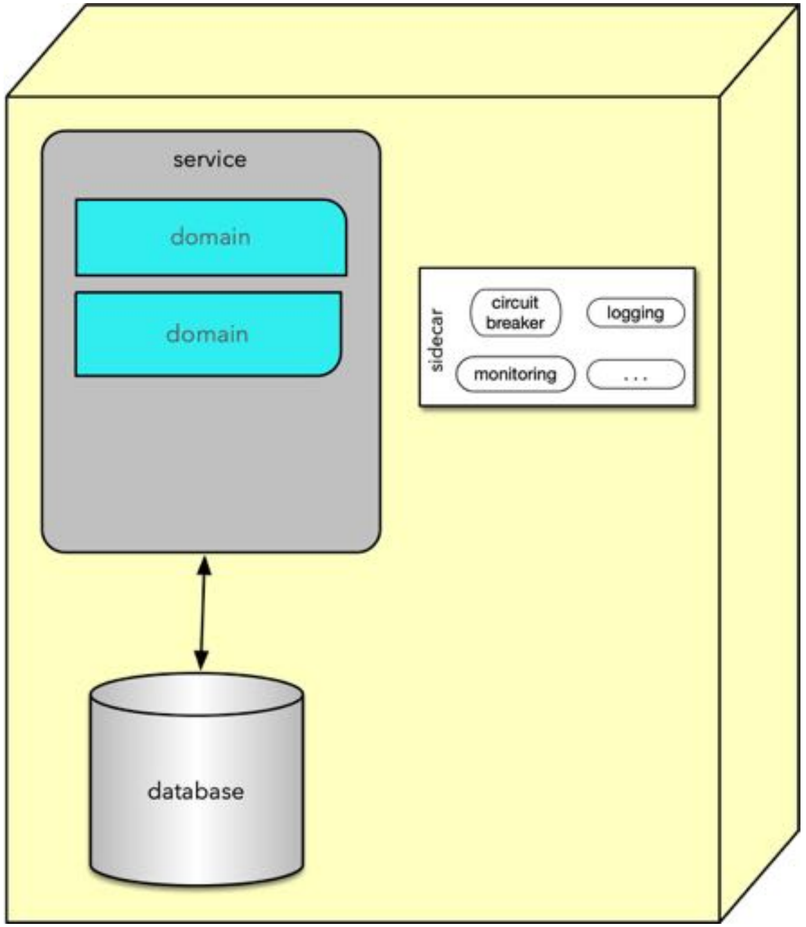
elasticity in microservices



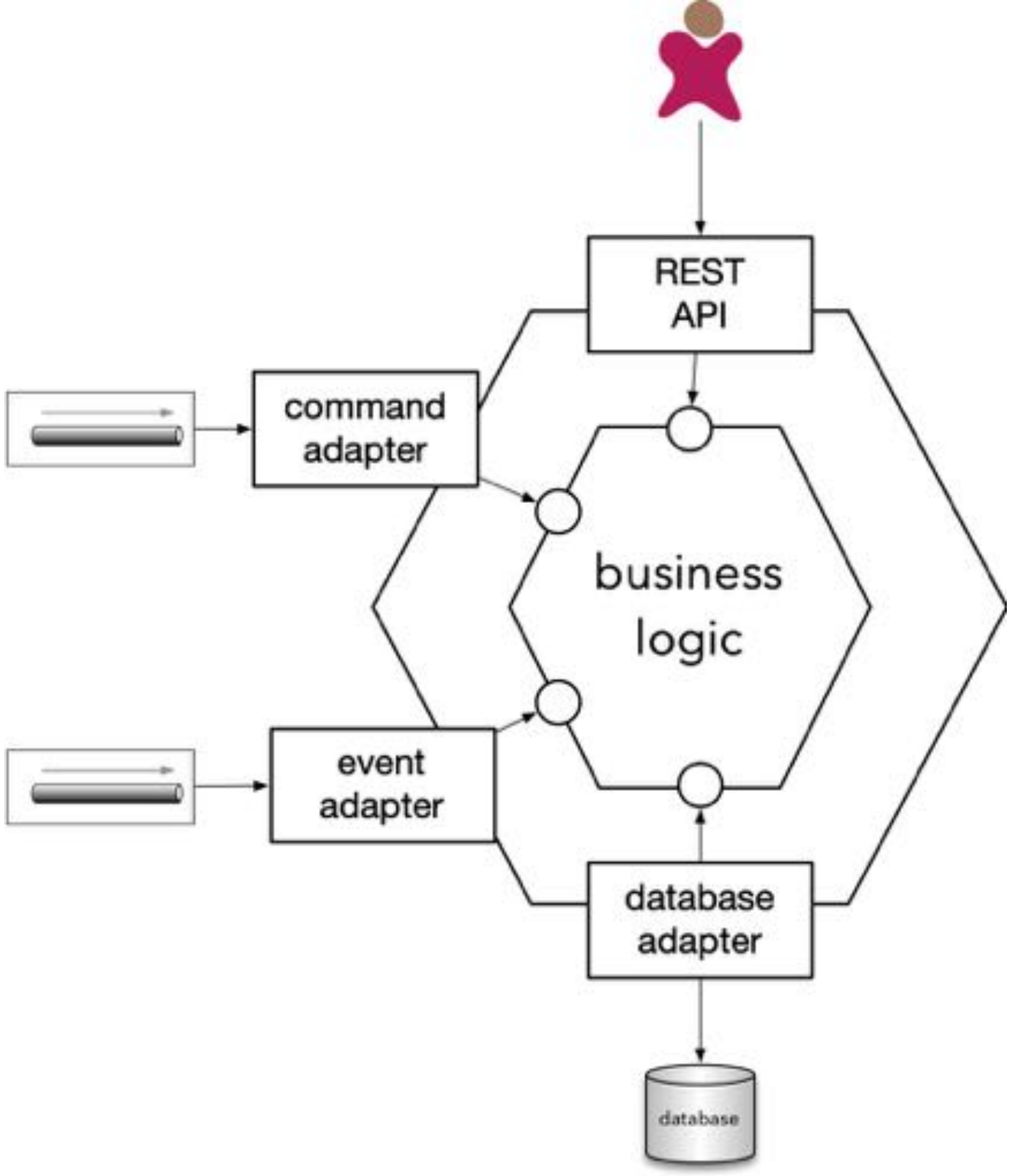
elasticity in microservices



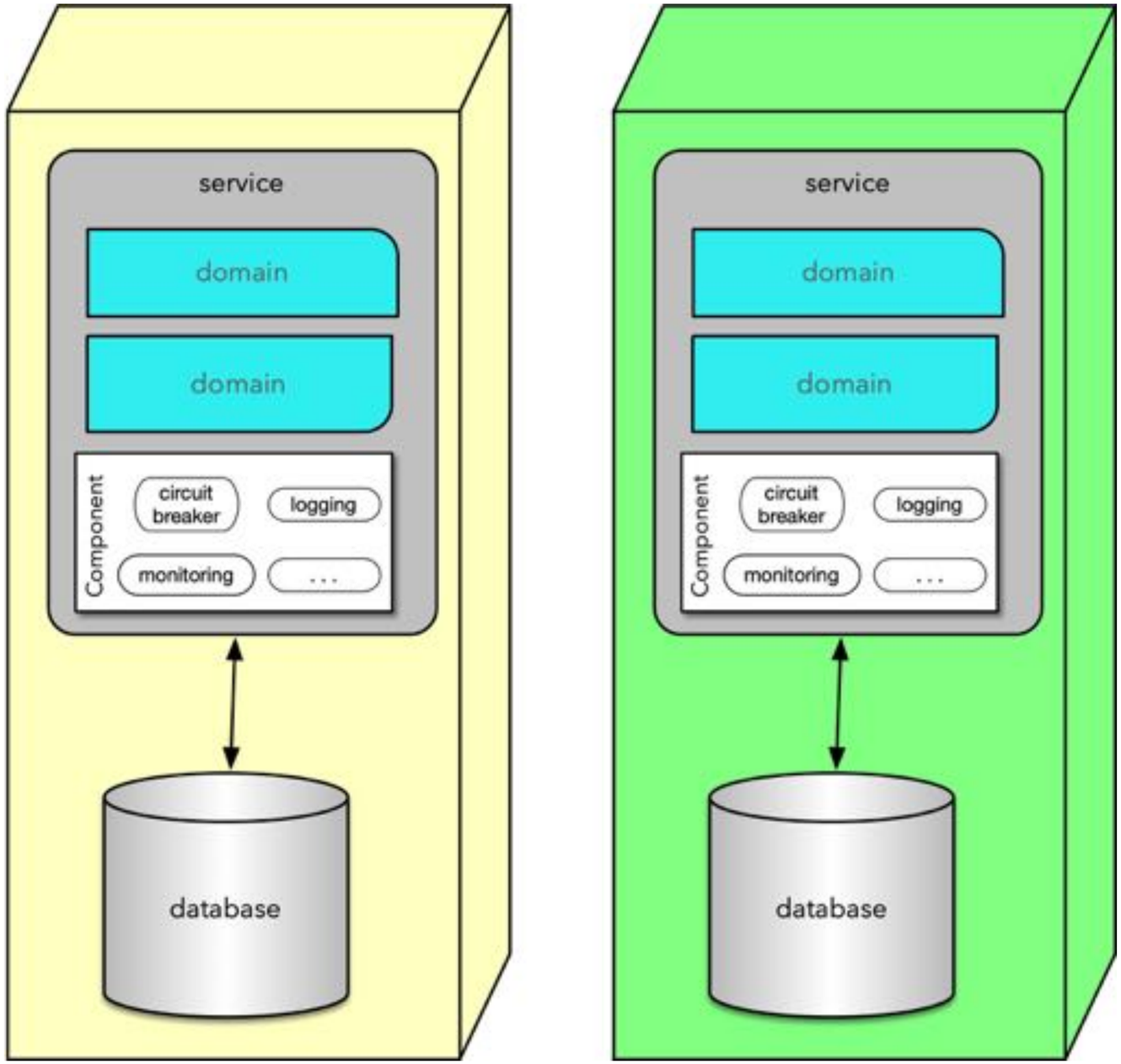
operational reuse in microservices: the sidecar pattern



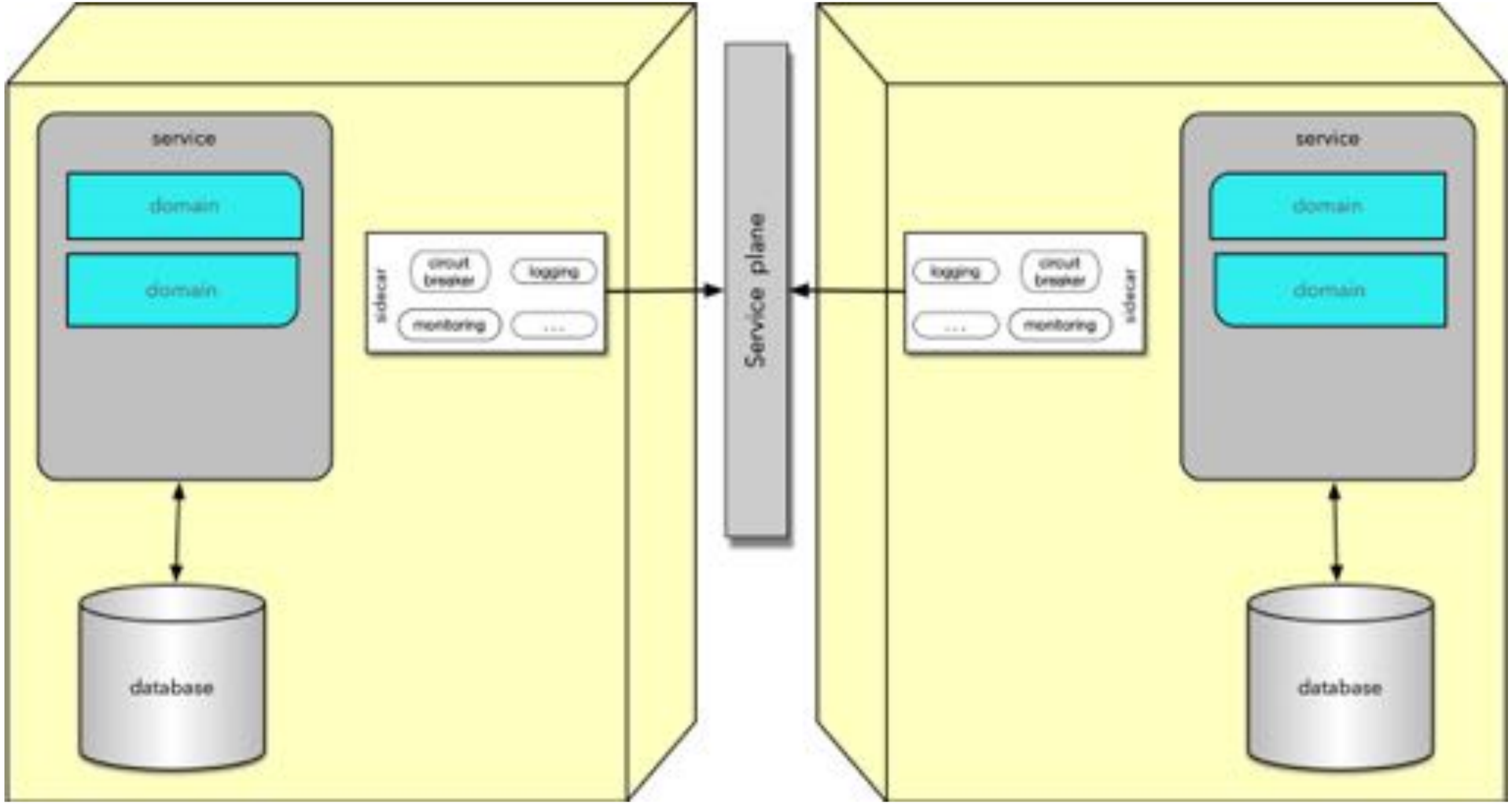
hexagonal pattern



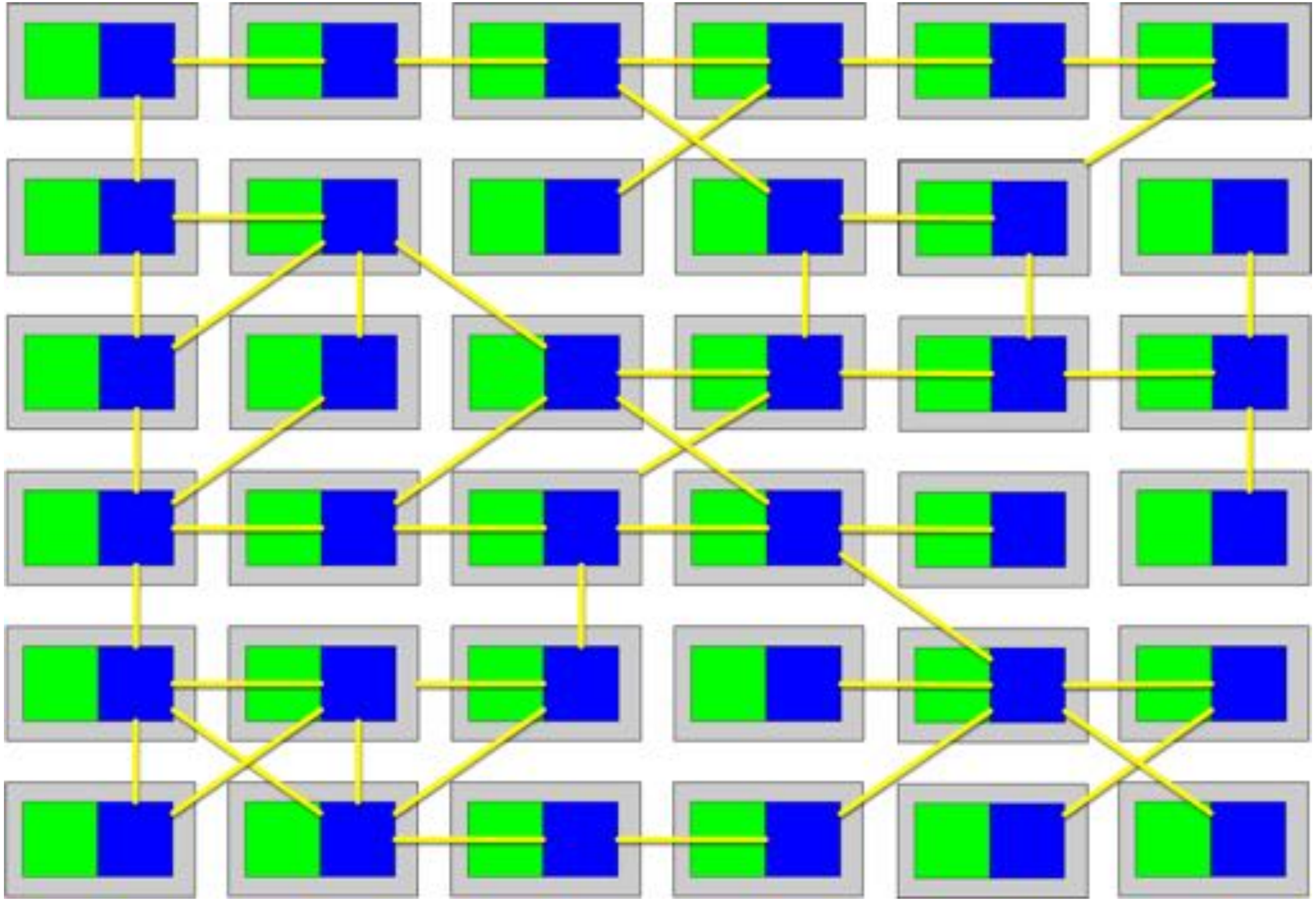
operational reuse



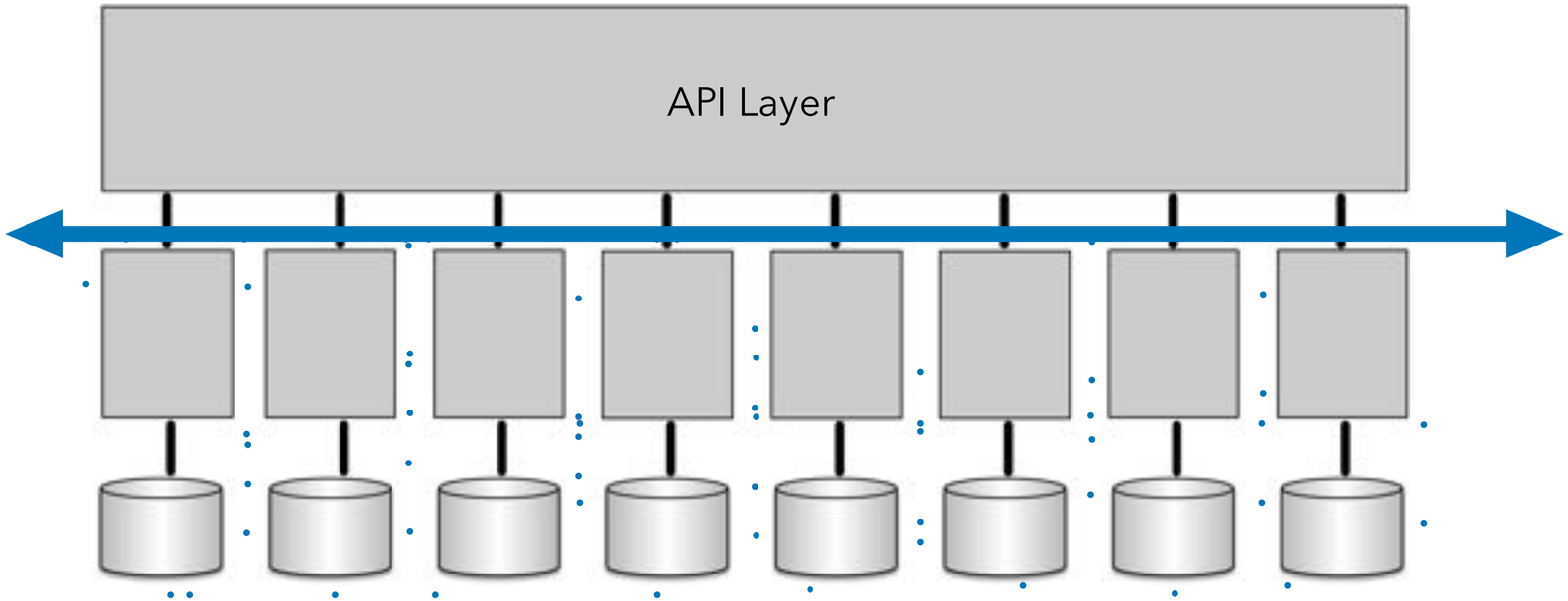
the sidecar pattern

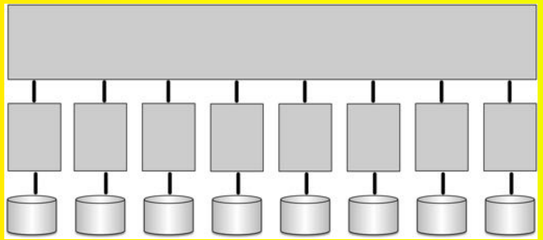


service mesh



elasticity in microservices





+ scalability achieved via service discovery + service mesh + orchestration

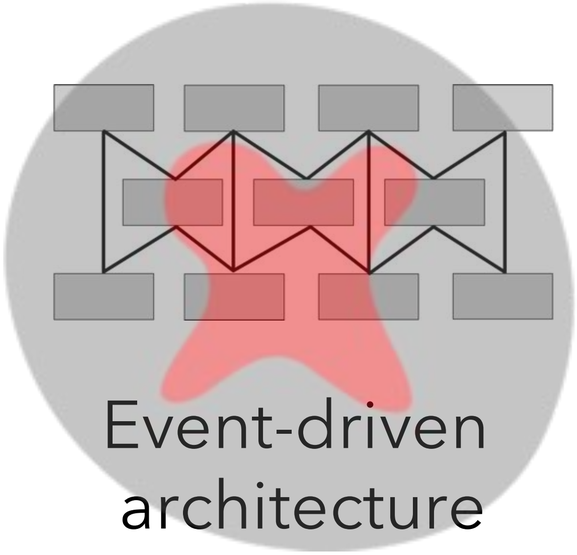
+ follows current trends

+ domain partitioned; low(est) incidental coupling

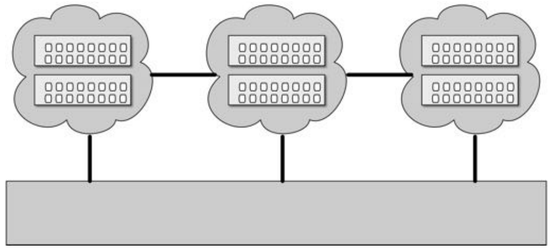
+ extremely good MTTS (mean-time-to-startup)

– architects must build transactional behavior

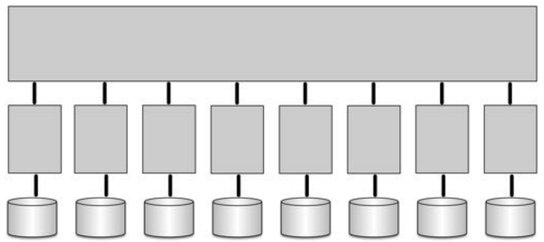
scalability ∪ elasticity



Event-driven architecture

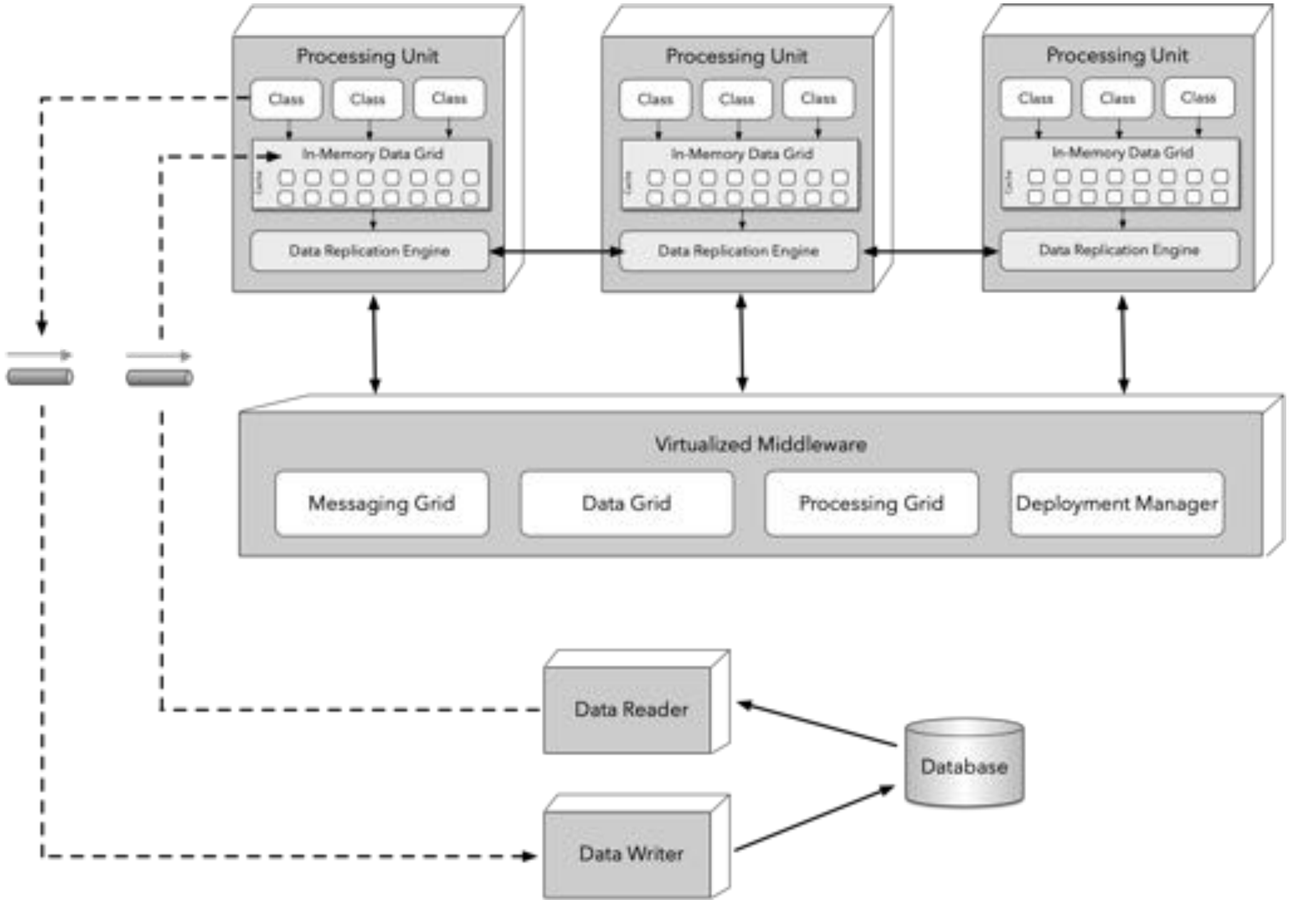


Space-based architecture

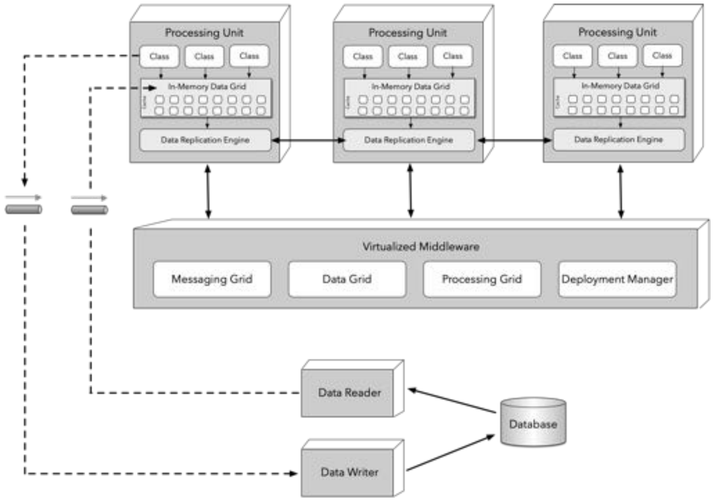


Microservices architecture

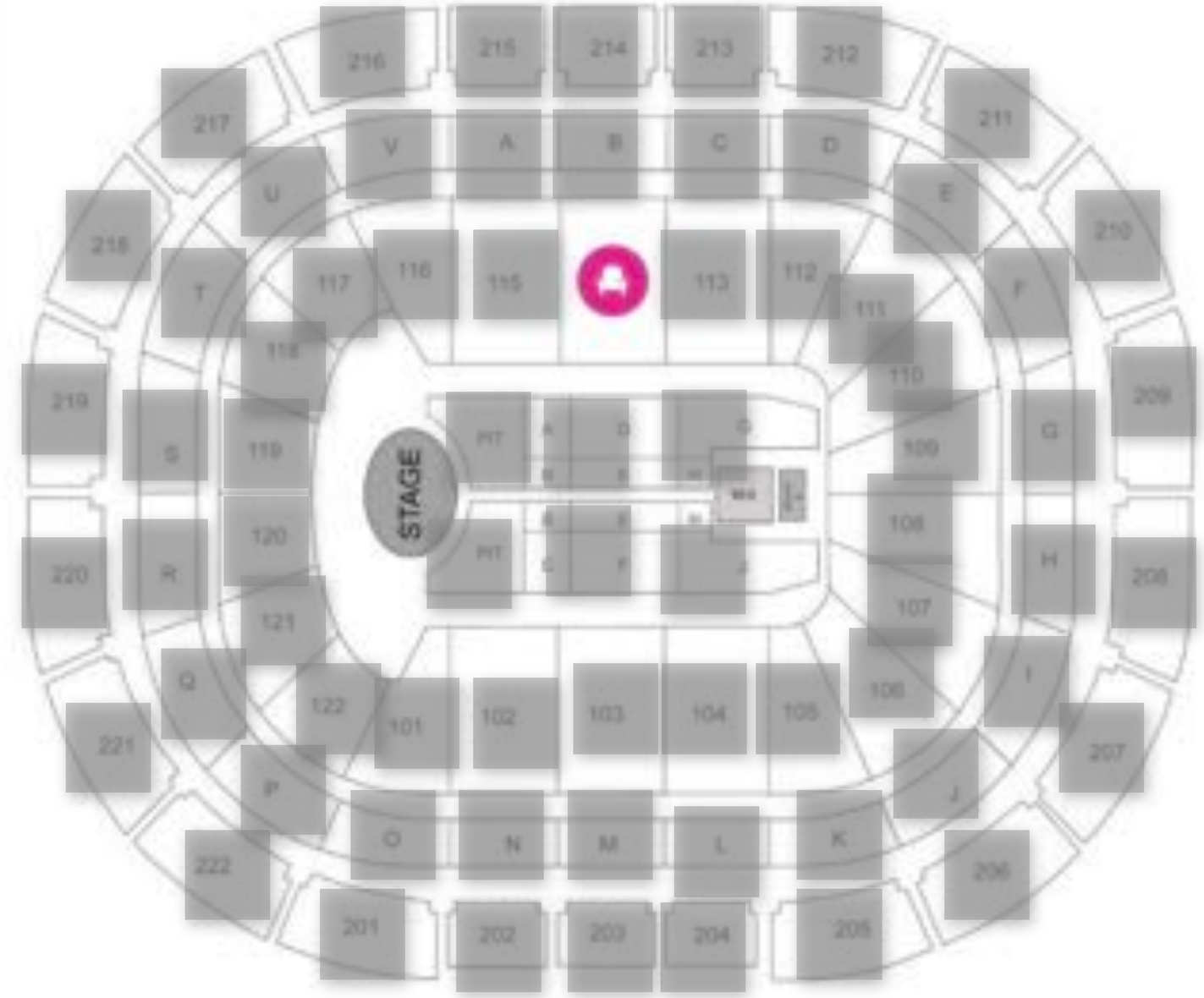
space-based ing: transactions

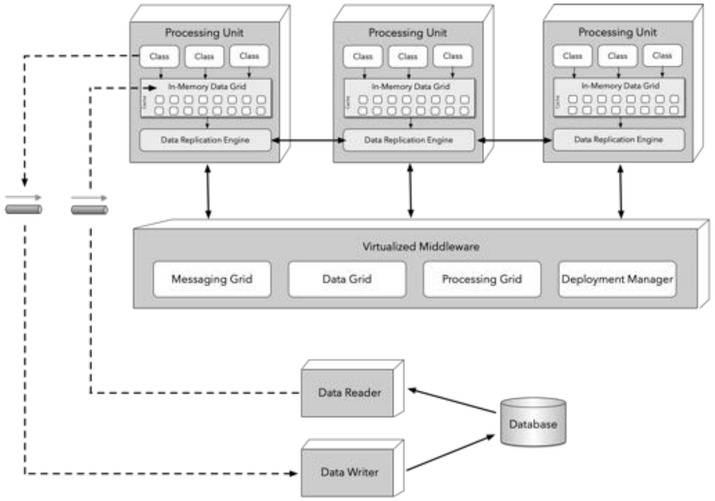


space-based □ing: transactions

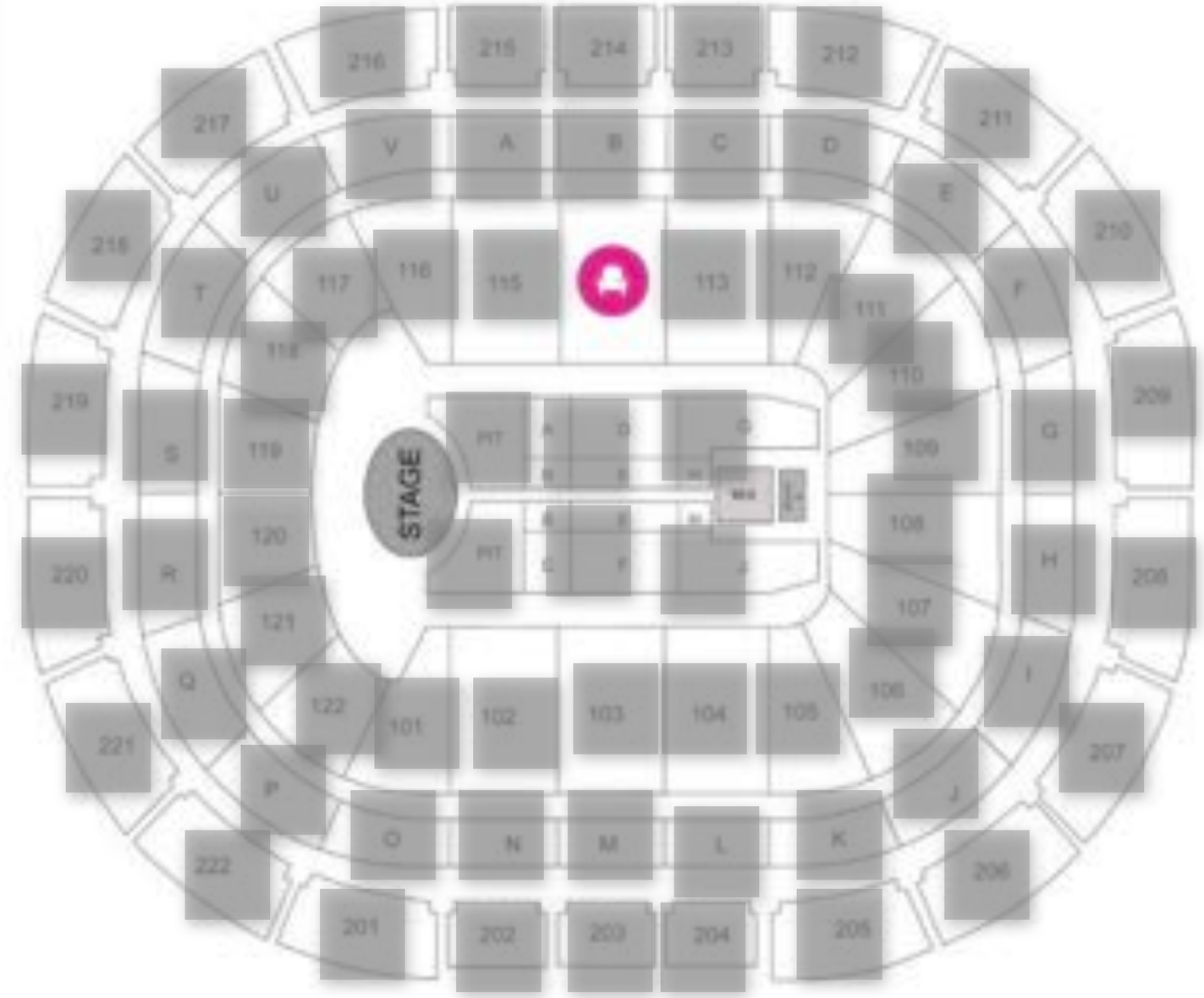


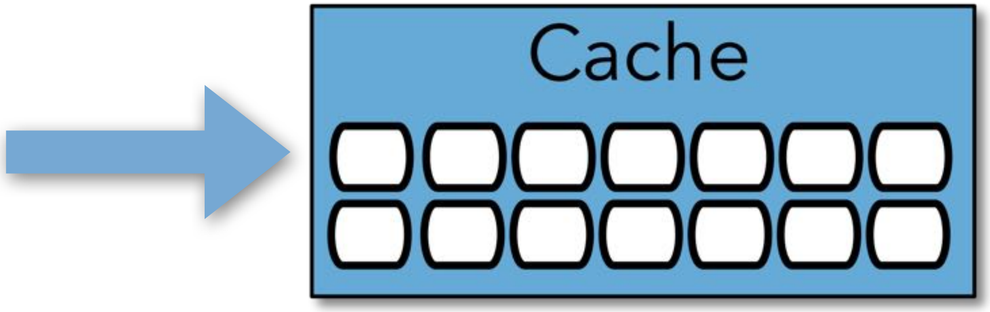
space-based ing: transactions

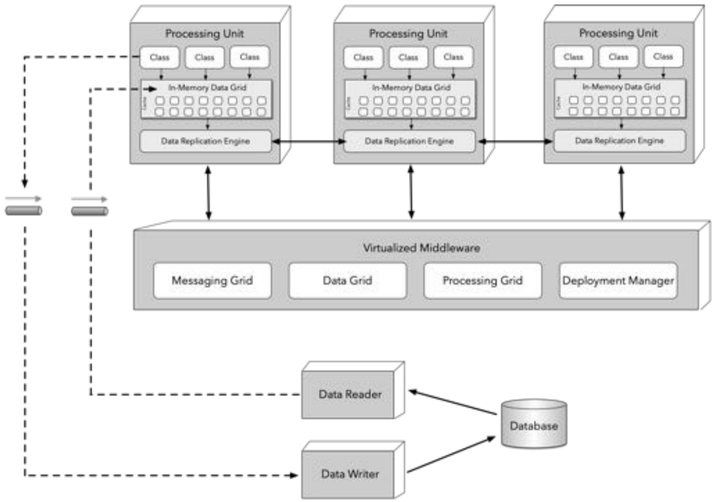




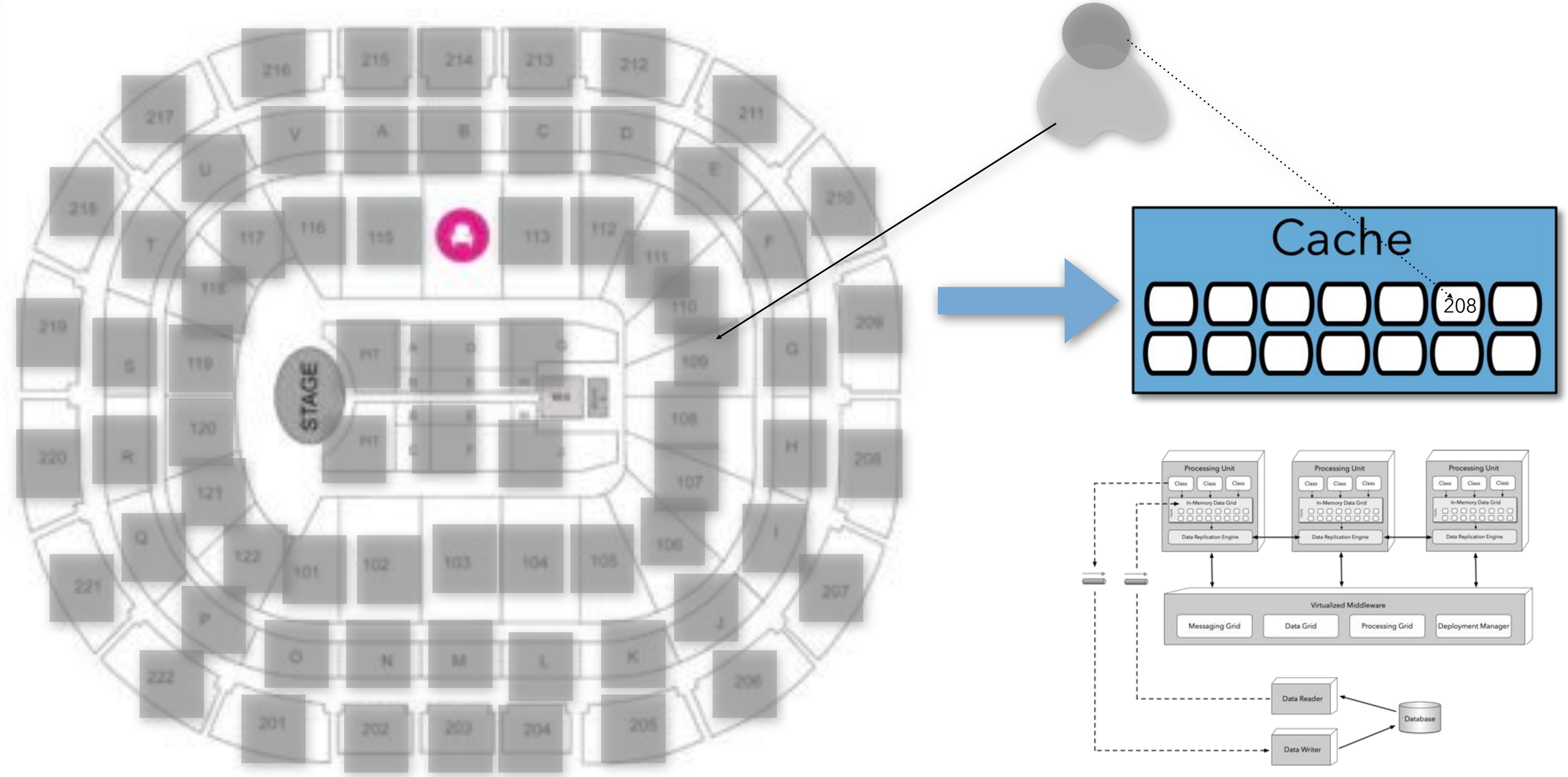
space-based □ing: preview



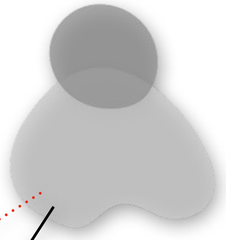




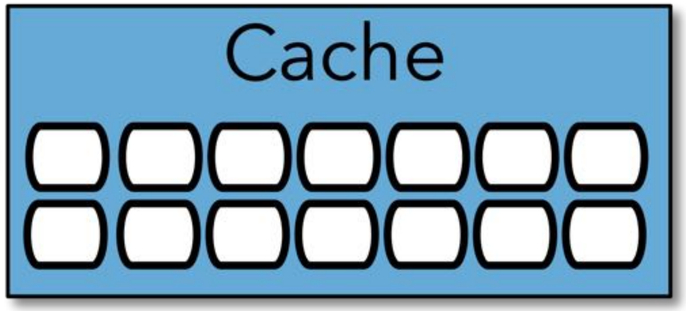
space-based □ing: preview

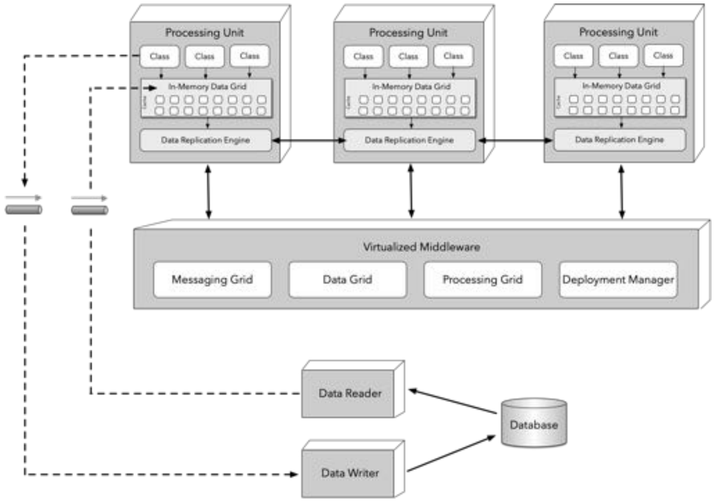


space-based □ing: preview

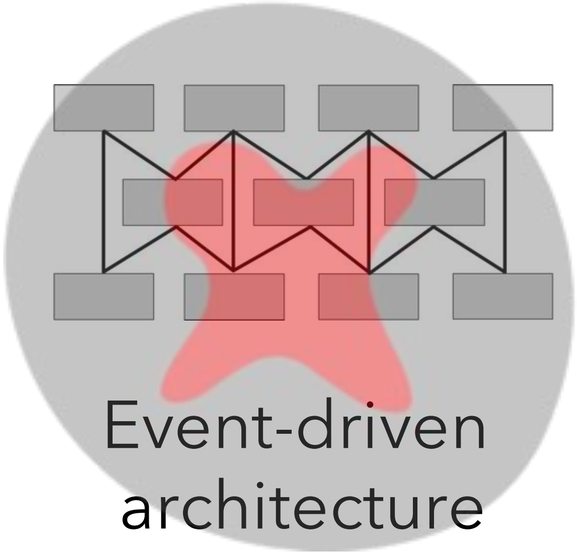


domain sharding

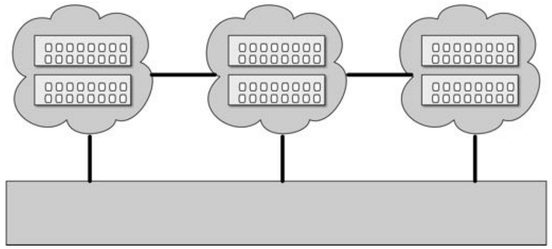




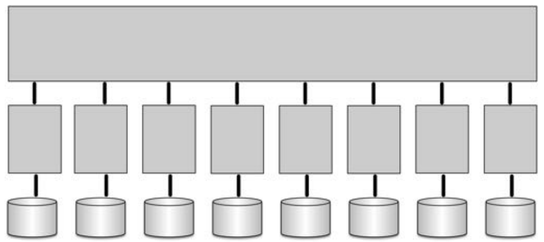
scalability ∪ elasticity



Event-driven architecture

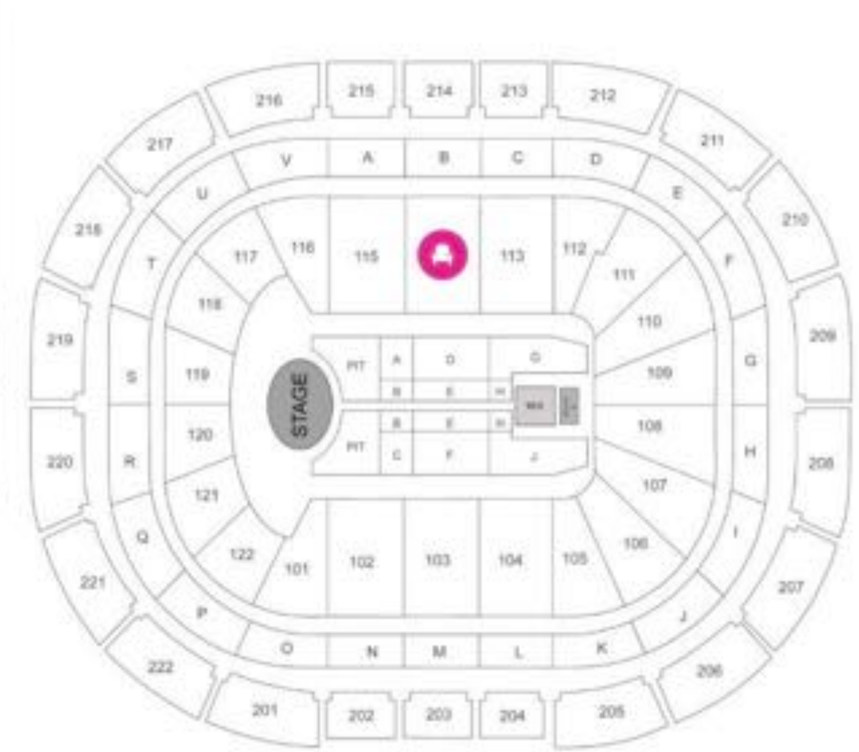


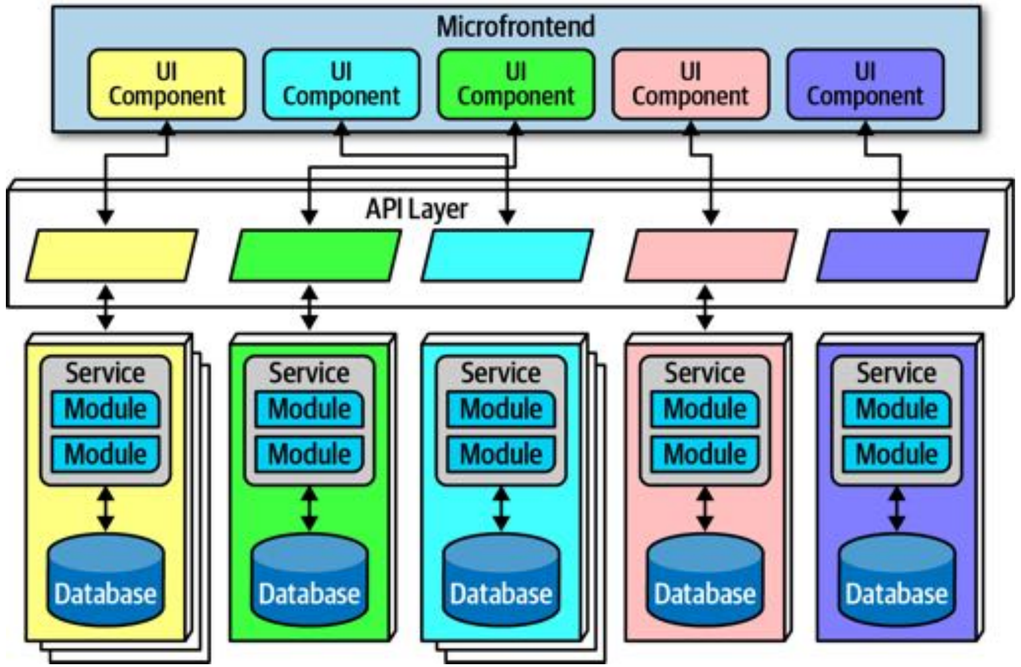
Space-based architecture



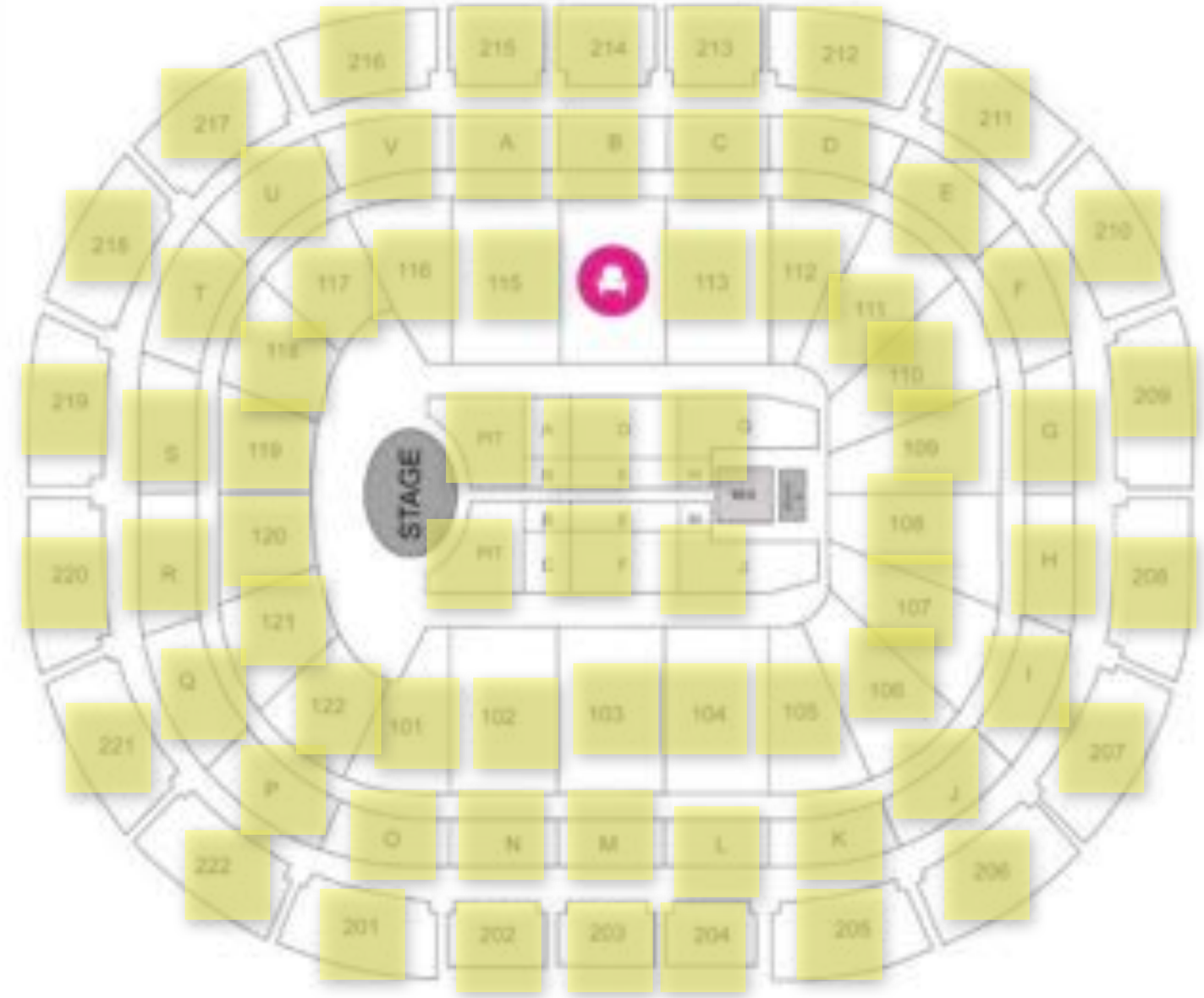
Microservices architecture

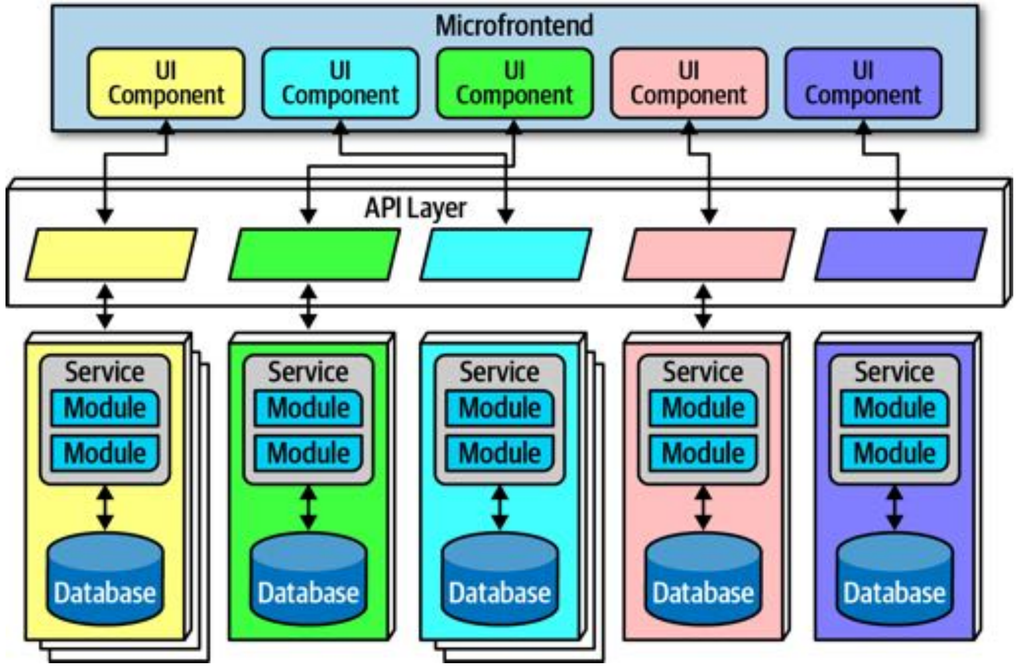
microservices □ing: transactions



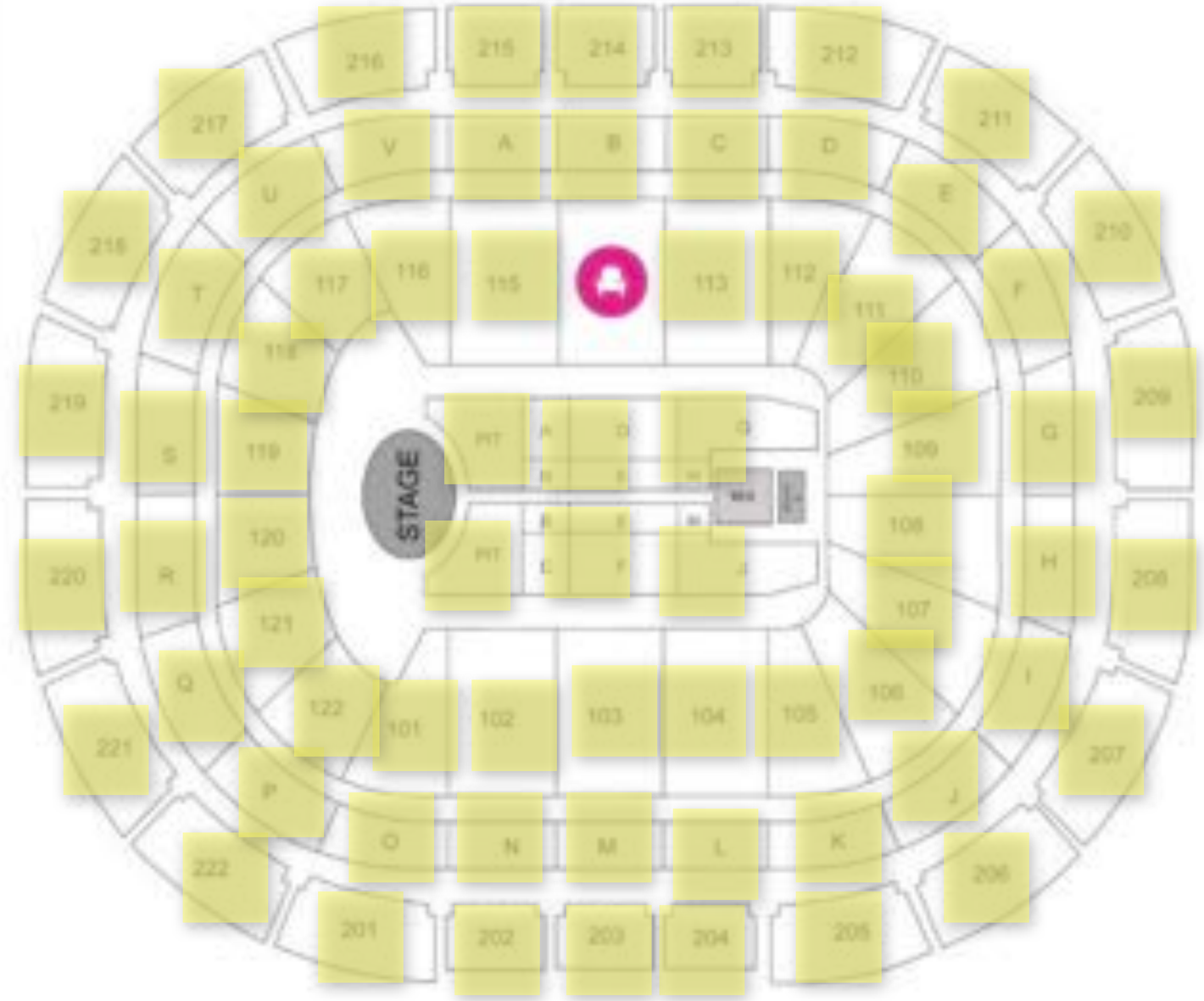


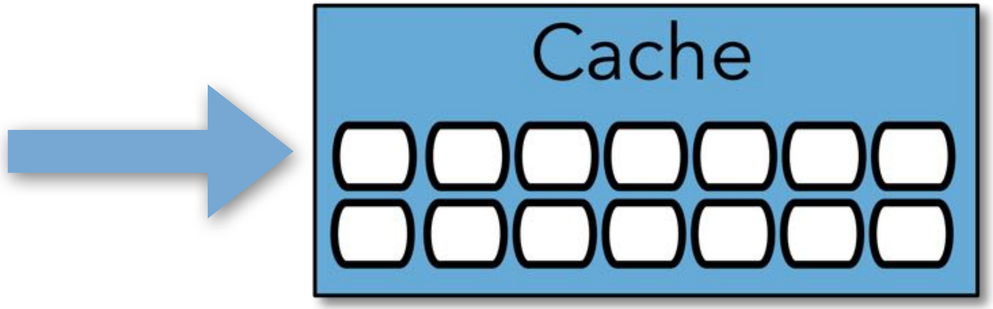
microservices □ing: transactions

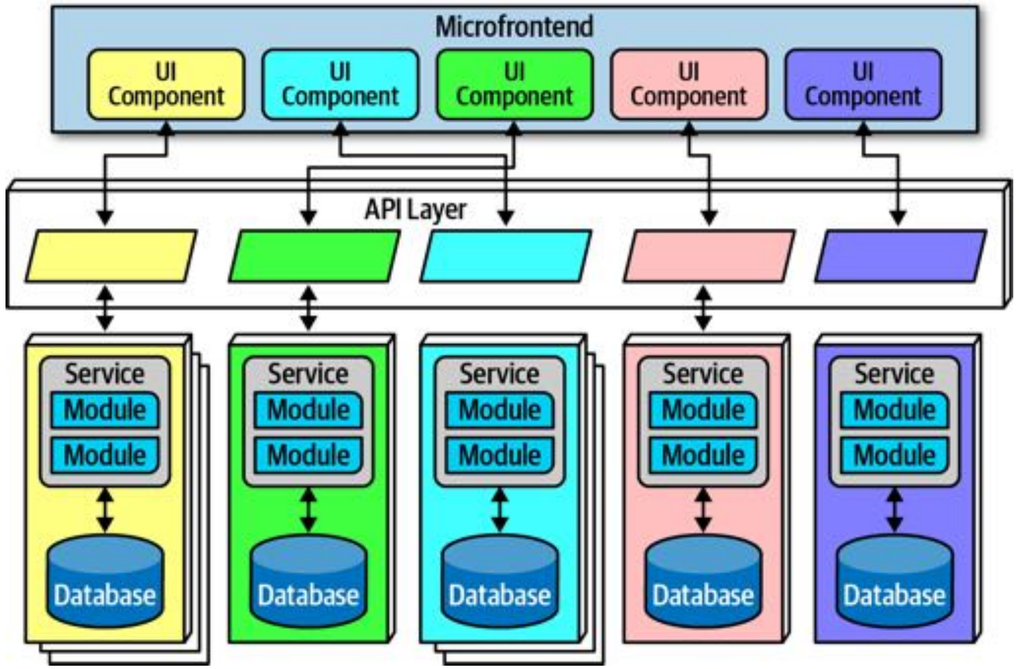




microservices □ing: preview







tradeoffs

space-based ***OR*** microservices

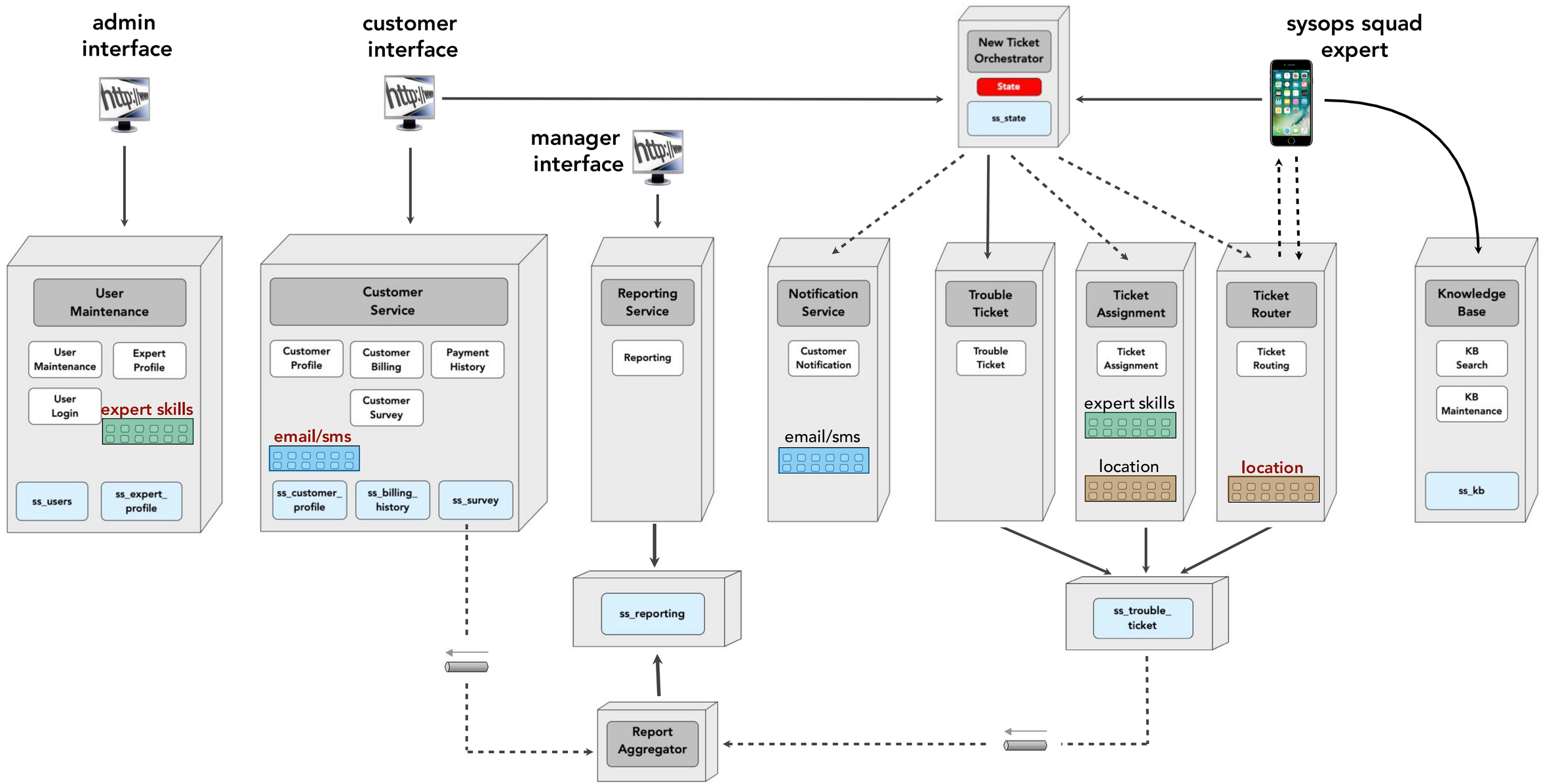
* more optimized for the problem space
* higher complexity
* higher incidental coupling

* more domain aligned
* current trend in architecture

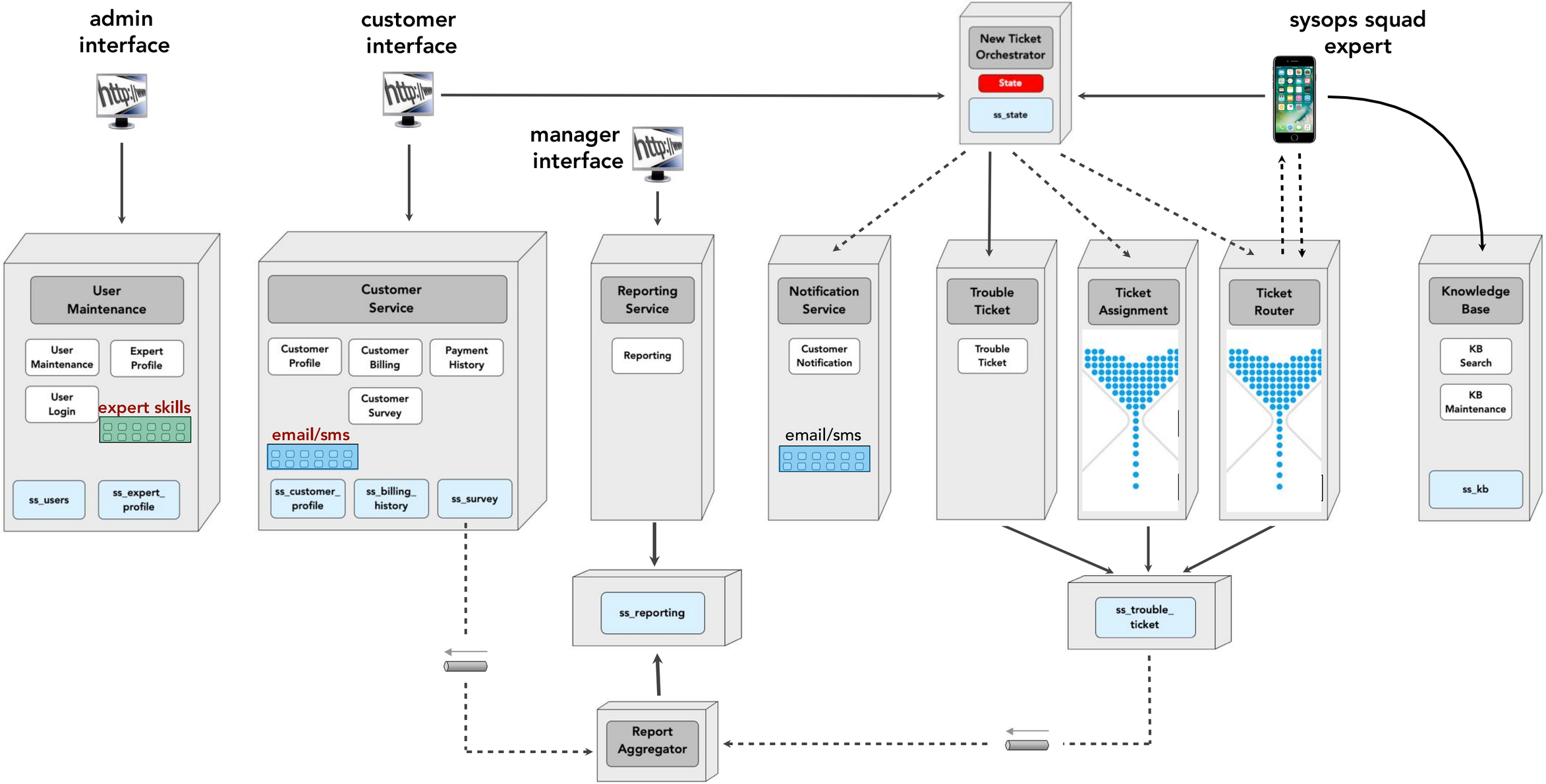
Kata Exercise - Handling Elasticity

Due to a recent pandemic world cup soccer will only be televised. As a result, sales have gone through the roof for new TV's, computers, and other electronic equipment. Your manager is very worried about a significant increase in trouble tickets, all within a very short period of time. How can you modify the architecture to handle this anticipated peak in trouble tickets?

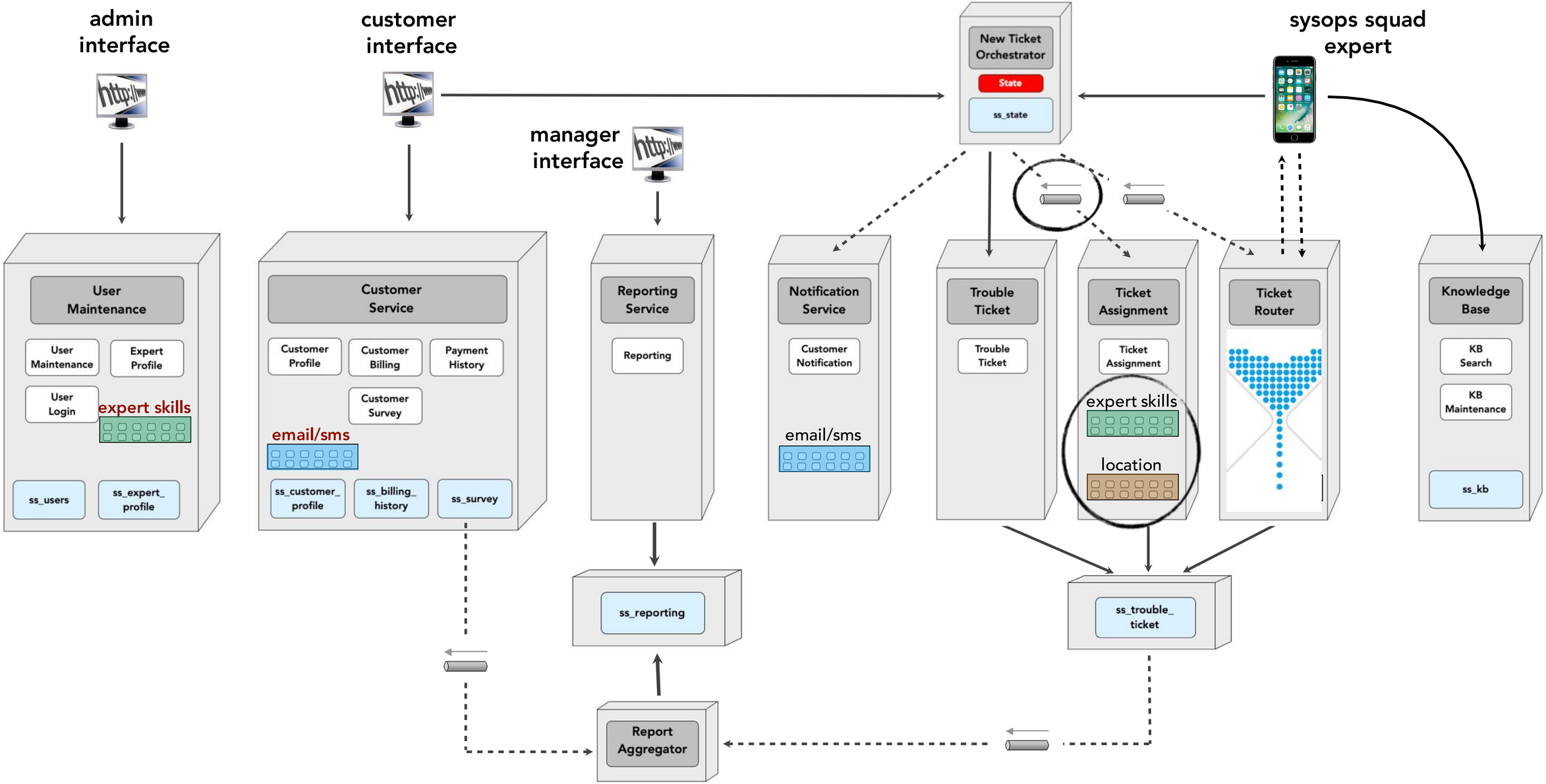
Kata Exercise - Handling Elasticity



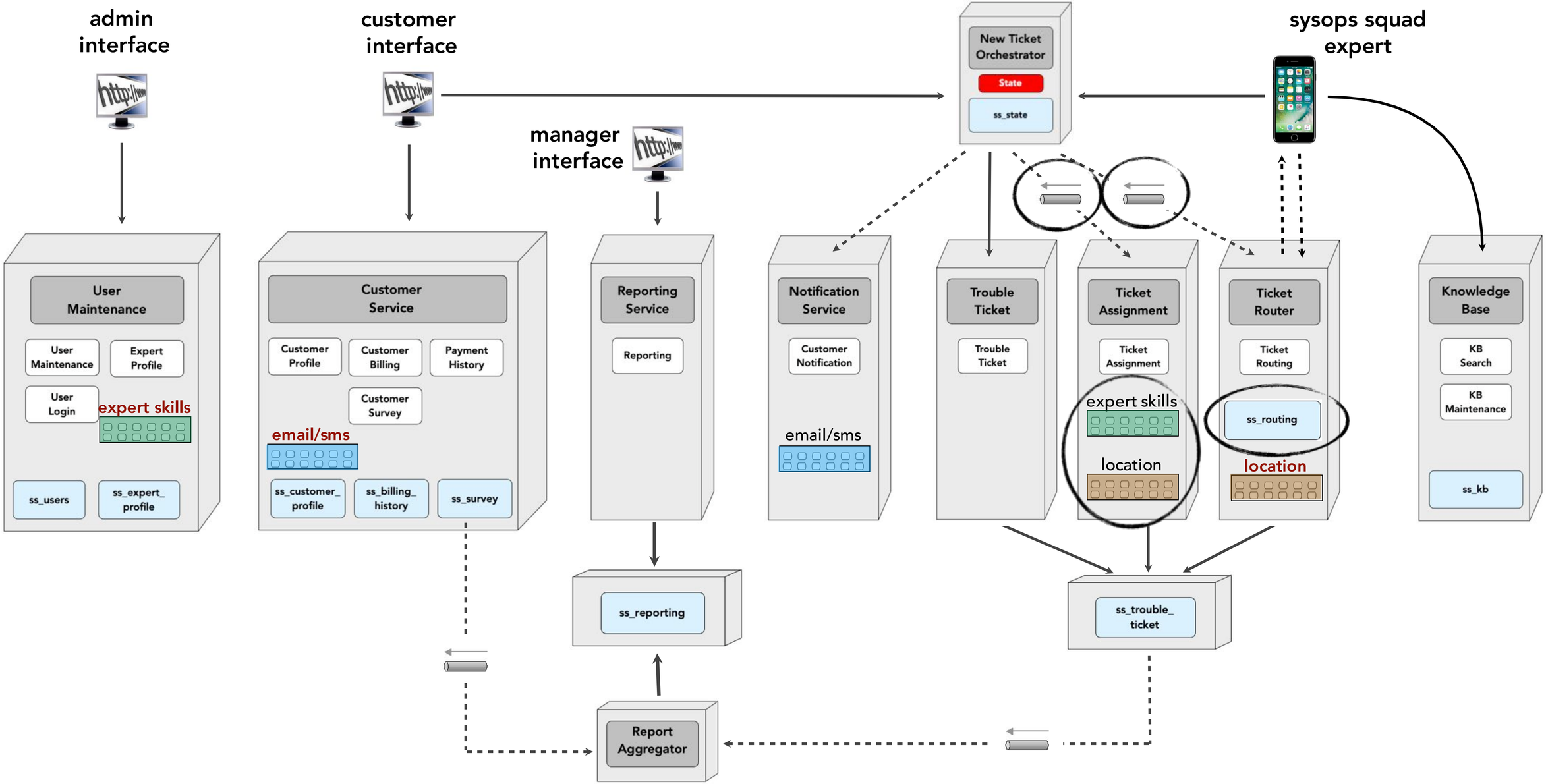
Kata Exercise - Handling Elasticity



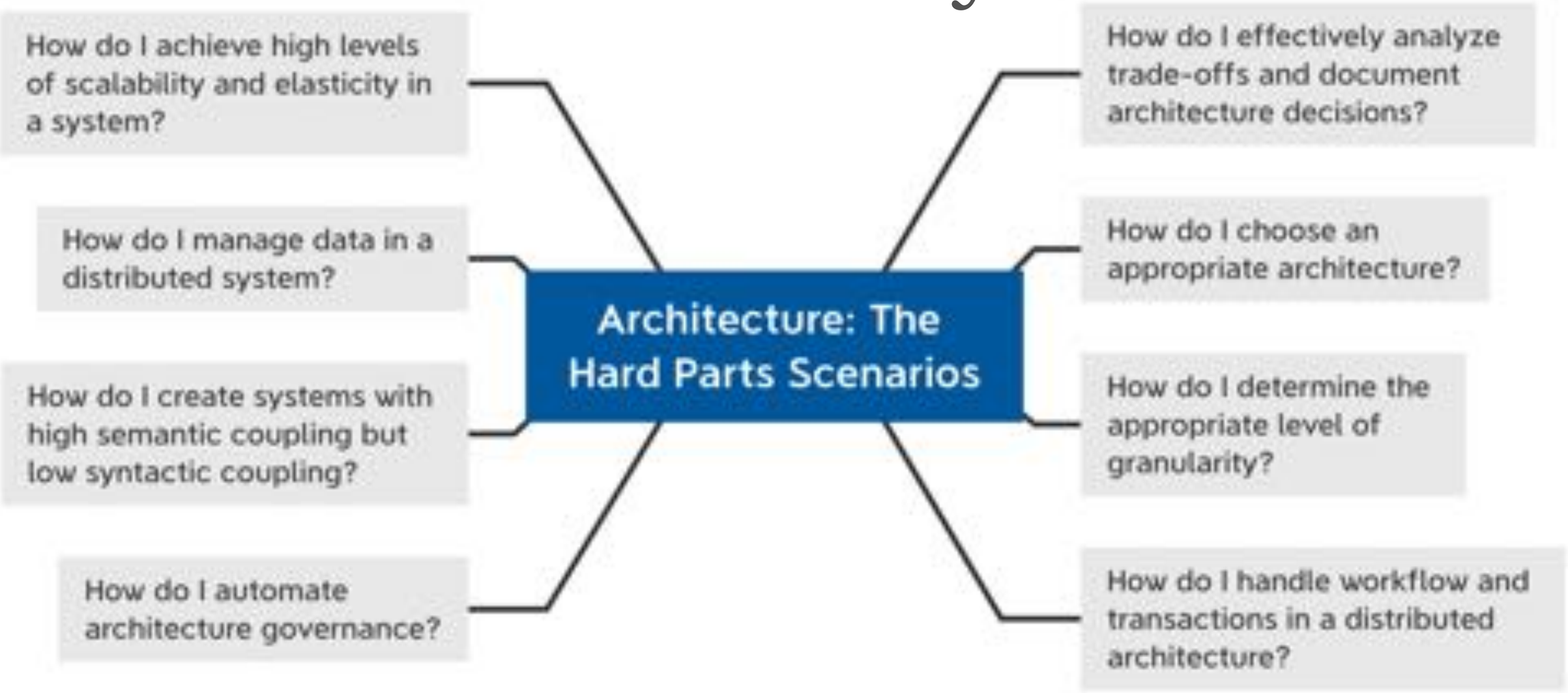
Kata Exercise - Handling Elasticity



Kata Exercise - Handling Elasticity

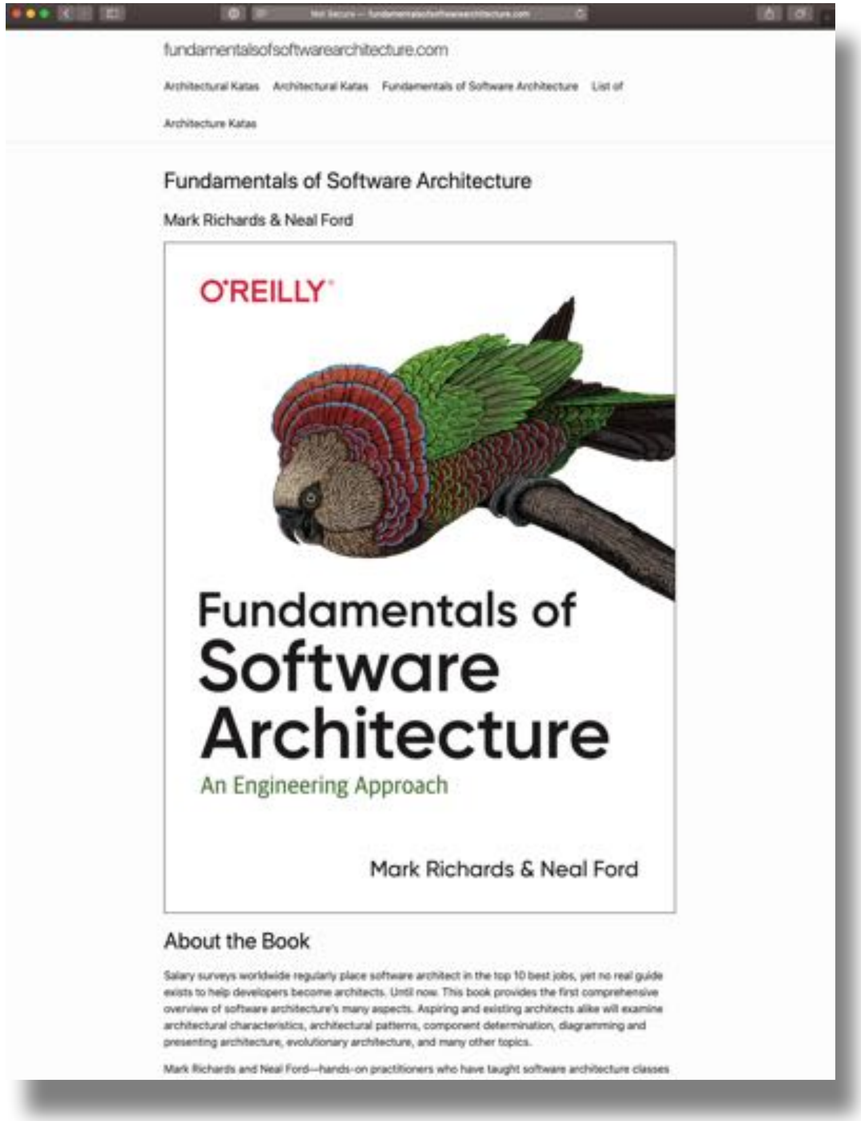


summary


Architecture: The Hard Parts Scenarios
How do I achieve high levels of scalability and elasticity in a system?
How do I manage data in a distributed system?
How do I create systems with high semantic coupling but low syntactic coupling?
How do I automate architecture governance?
How do I effectively analyze trade-offs and document architecture decisions?
How do I choose an appropriate architecture?
How do I determine the appropriate level of granularity?
How do I handle workflow and transactions in a distributed architecture?

Architecture: The Hard Parts

Fundamentals of Software Architecture



fundamentalsofsoftwarearchitecture.com

DeveloperToArchitect.com

HOME TRAINING LESSONS ARTICLES BOOKS VIDEOS CONTACT

Developer To Software Architect

"The journey from developer to software architect is a difficult and uncharted path filled with lots of challenges, pitfalls, and confusion. The purpose of DeveloperToArchitect.com is to provide resources and training to help you along the journey to becoming an effective software architect"

Mark Richards, Software Architect, Founder of DeveloperToArchitect.com

I am a hands-on software architect with over 30 years of experience in the industry. I have experience in the architecture and delivery of Microservices Architectures, Service-Based Architectures, and Service-Oriented Architectures in a variety of platforms. I am also a published author, conference speaker, and trainer.

Subscribe to RSS feed

Software Architecture Fundamentals Public 3-Day Class Dates and Locations 2018

Classes are held Friday, Saturday, and Sunday from 8am to 4pm

February 2-4, Reston, VA
March 9-11, Chicago, Il
April 5-7, Salt Lake City, UT*

Mark Richards
Independent Consultant
Hands-on Software Architect, Published Author
Founder, DeveloperToArchitect.com
@markrichardssa

Architecture: The Hard Parts
Scenarios

Neal Ford
ThoughtWorks
Director / Software Architect / Meme Wrangler
http://www.nealford.com
@neal4d