

AP



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

Architecture: The Hard Parts

How do I choose an appropriate architecture?

How do I determine the appropriate level of service granularity?

How do I automate architectural governance?

How do I choose between choreography and orchestration for my workflow?

How do I create systems with high semantic coupling but low syntactic coupling?

How do I access data I don't own in a distributed system?

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.



Sysops Squad - A Bad Situation...

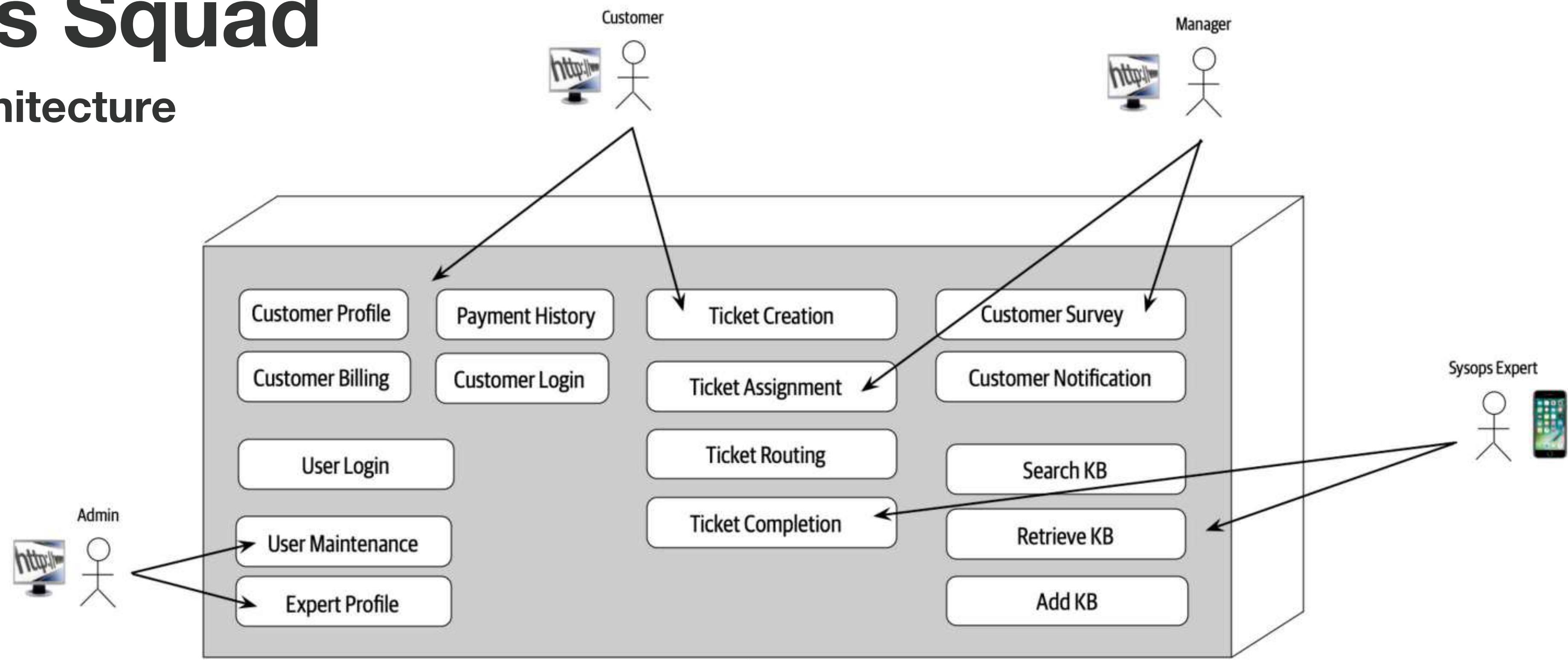
Things have not been good with the Sysops Squad lately. The current trouble ticket system is a large monolithic application that was developed many years ago. Customers are complaining that consultants are never showing up due to lost tickets, and often times the wrong consultant shows up to fix something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

Current process in the monolithic system:

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

Sysops Squad

Current Architecture



contains sysops squad profiles (skill, location, etc.)

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]

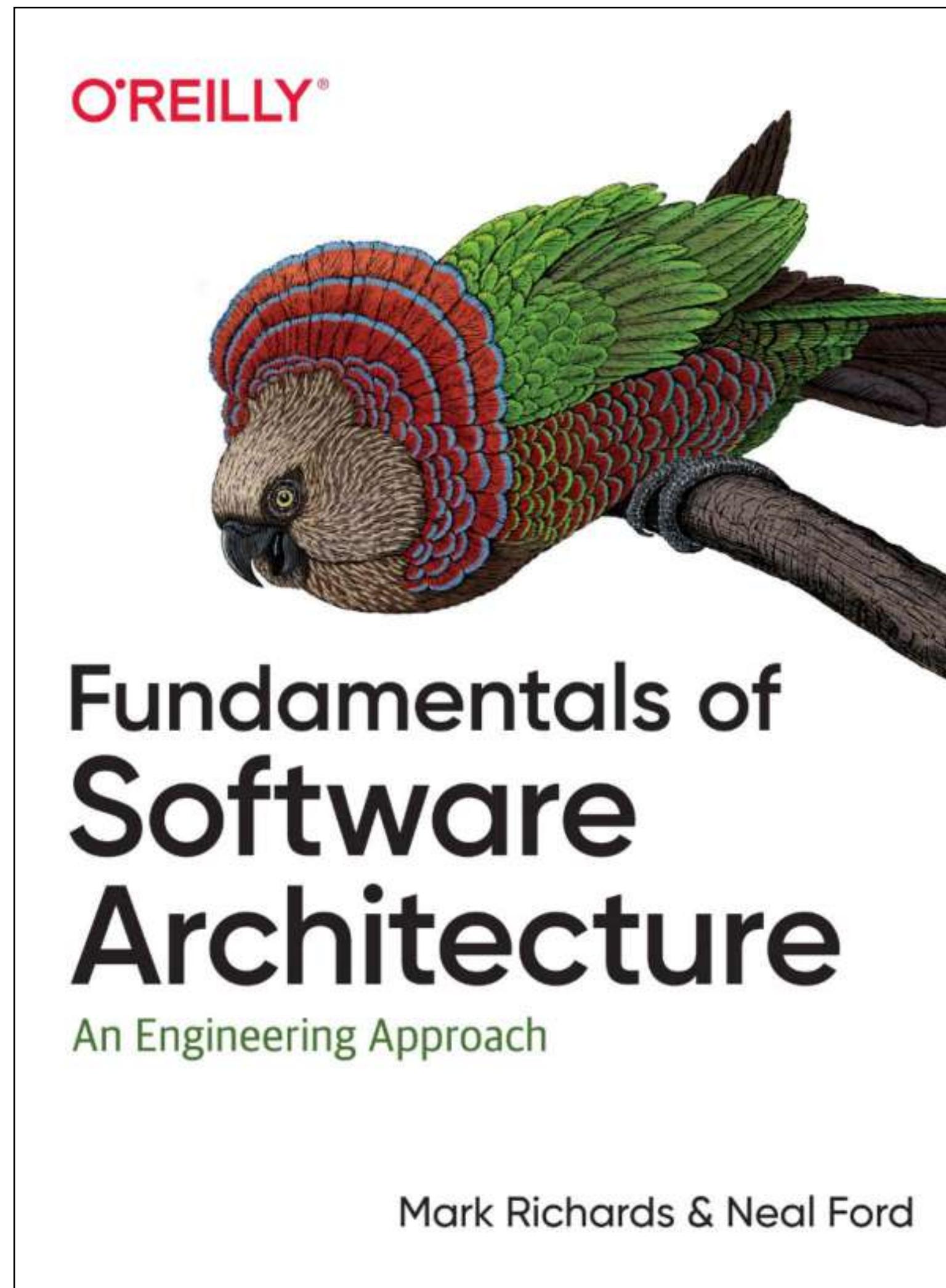
contains survey results from customers

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

documenting architecture decisions



Second Law of Software Architecture

“Why is more important than how”

tradeoffs

the +'s and –'s of *one* thing

+ positive tradeoffs

– negative tradeoffs

tradeoffs

this ***OR*** that

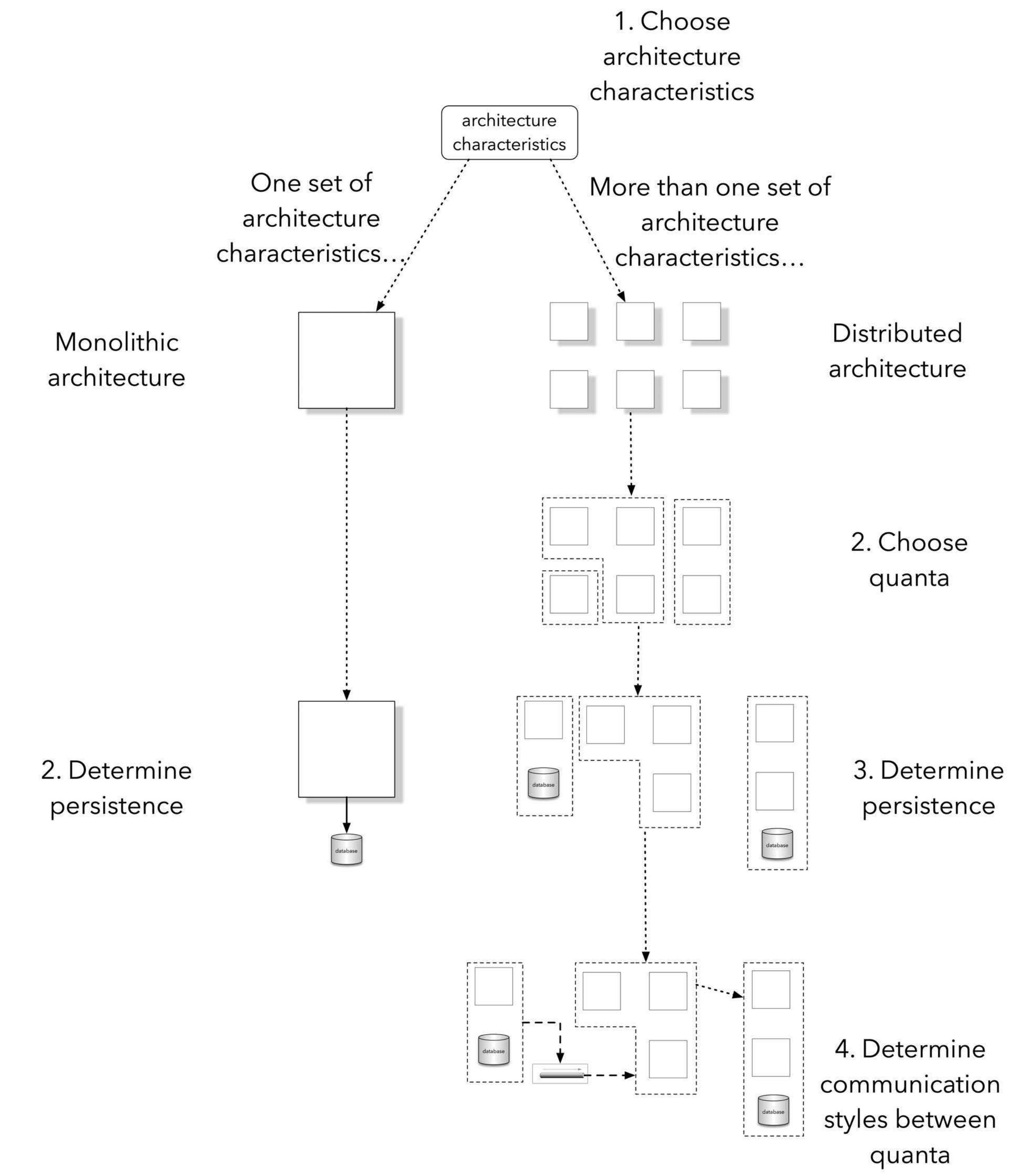
* stuff about ***this***

* stuff about ***that***



*How do I choose an
appropriate architecture?*

choosing an architecture





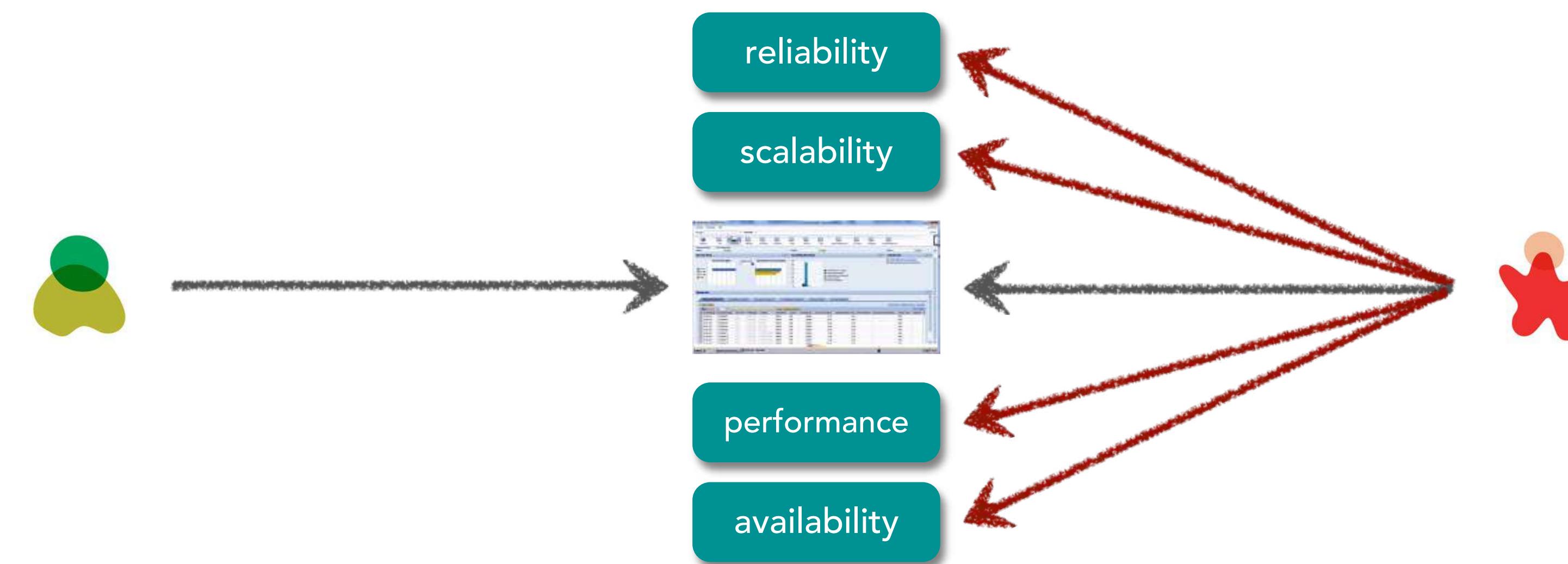
How do I choose an appropriate architecture?

1. Identify architecture characteristics
2. Identify the scope and number of distinct architecture characteristics

architecture characteristics



architecture characteristics



architecture characteristics

1. synergistic



2. ill-defined



3. voluminous



architecture characteristics

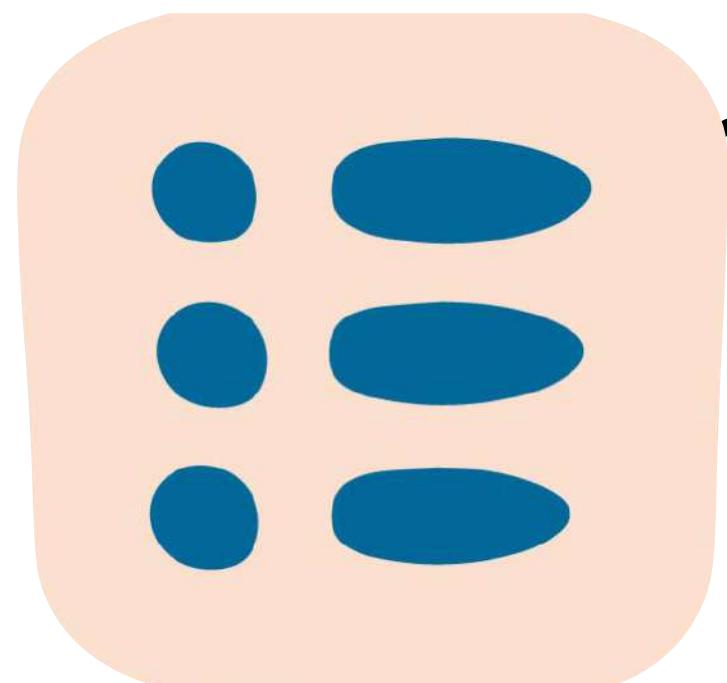
1. synergistic



2. ill-defined

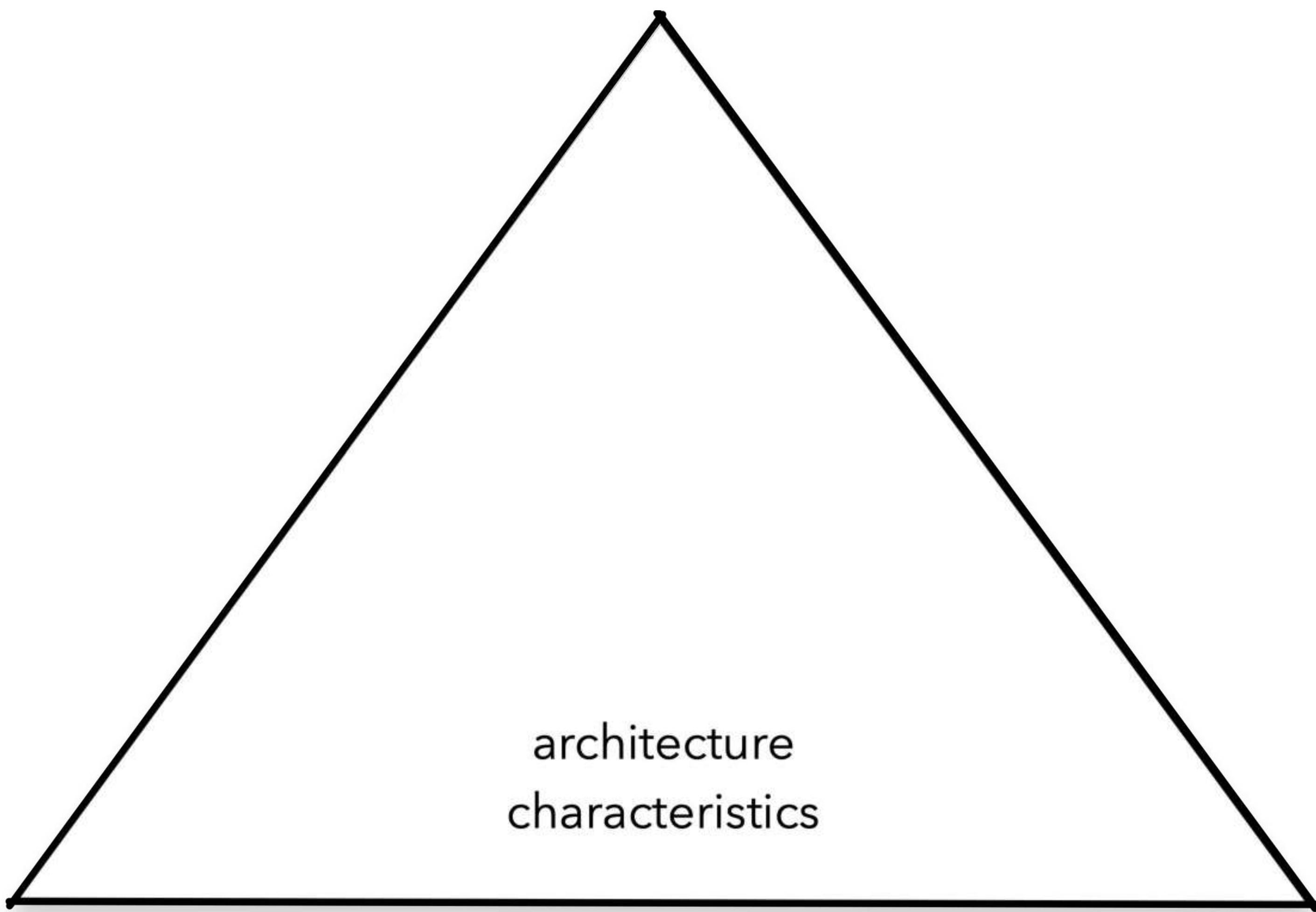


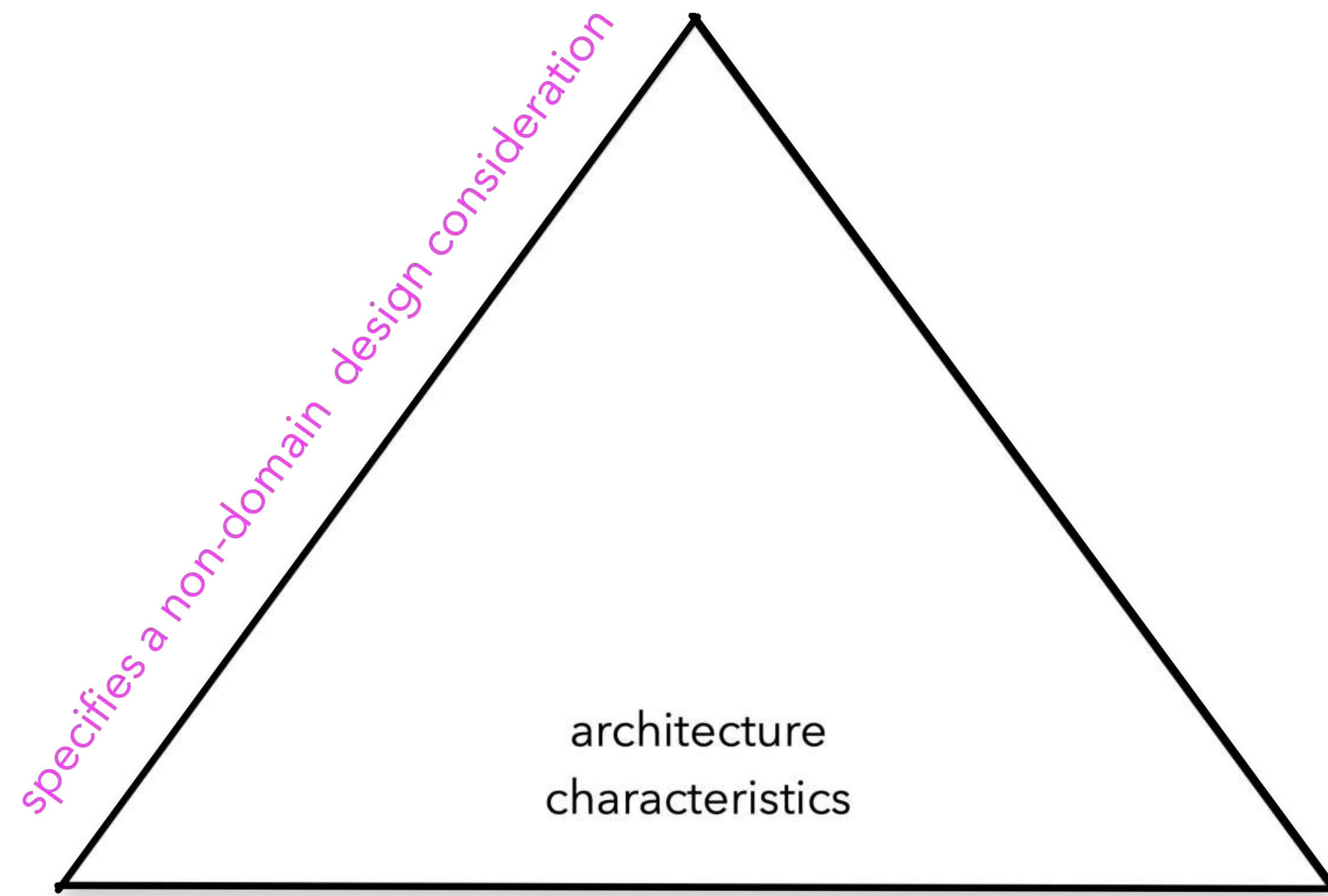
4. numerous categories

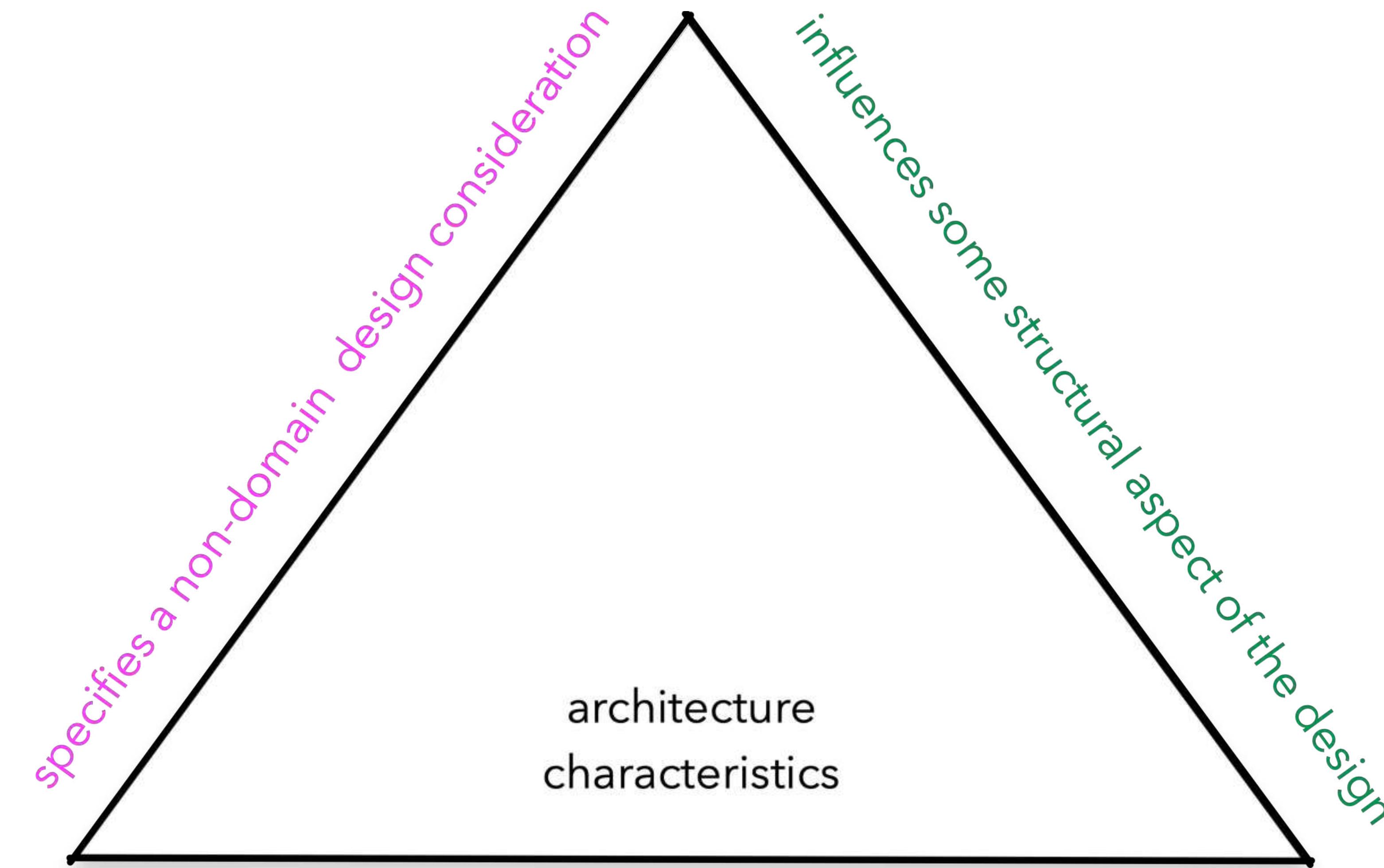


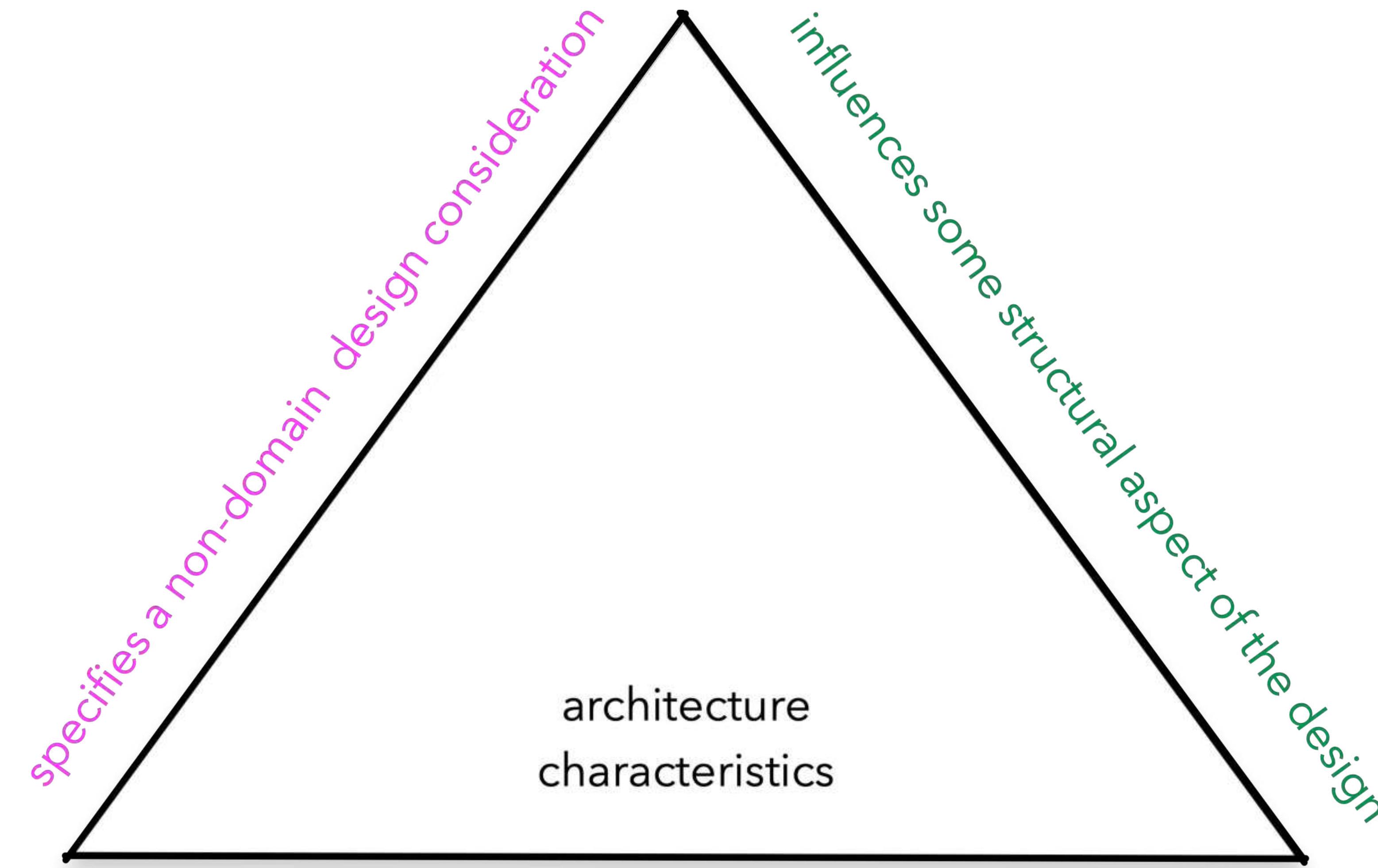
3. voluminous



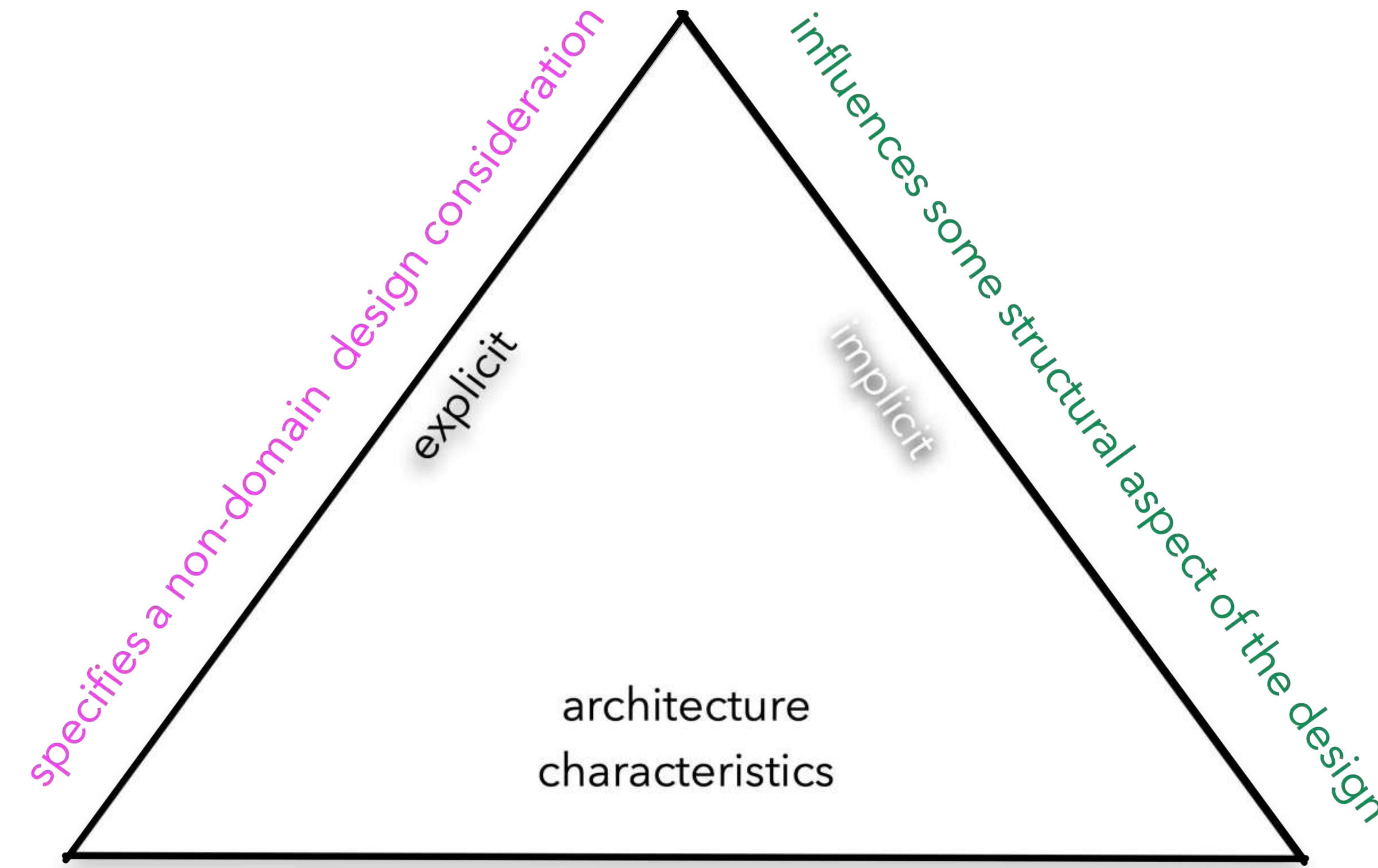






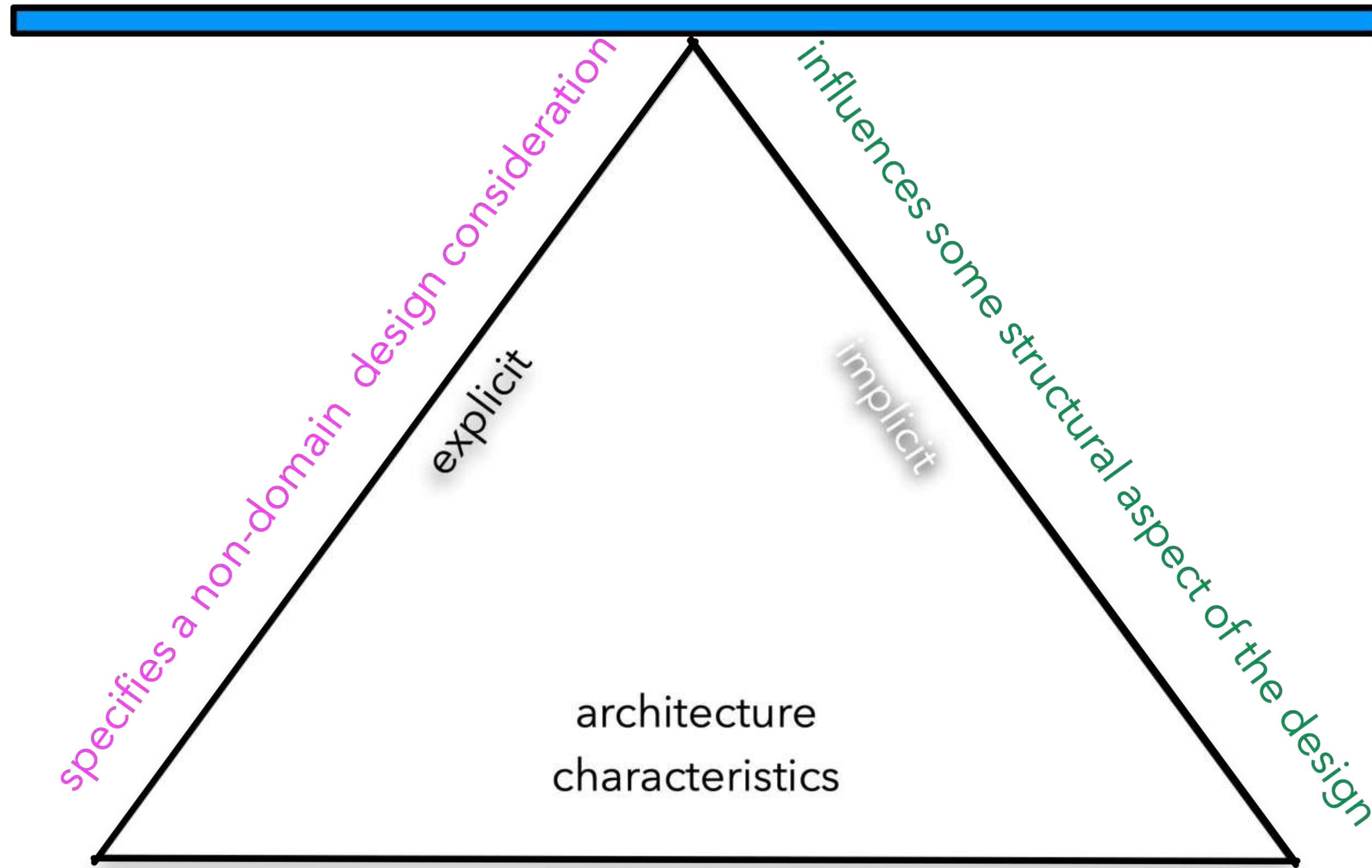


critical or important to application success

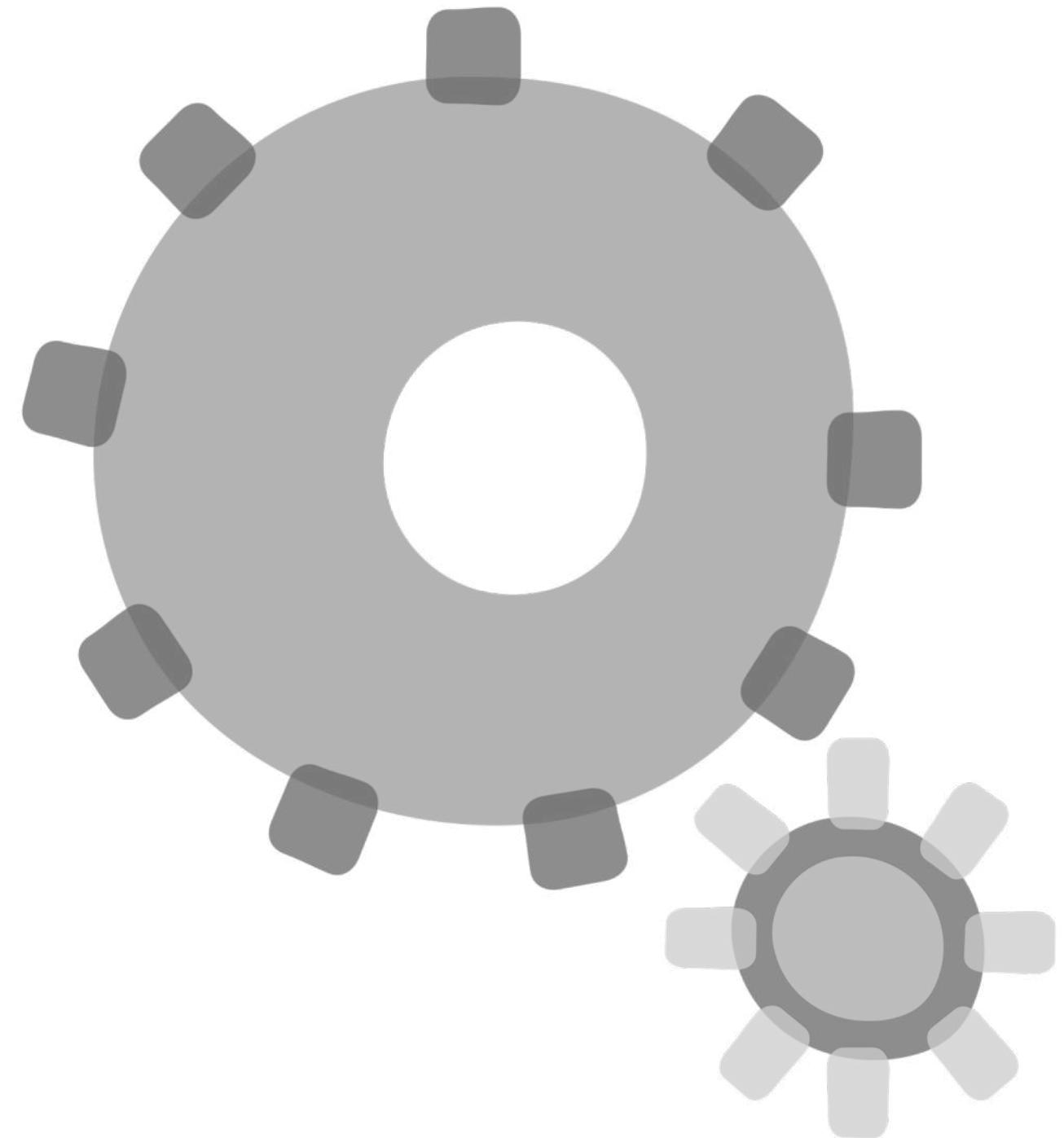


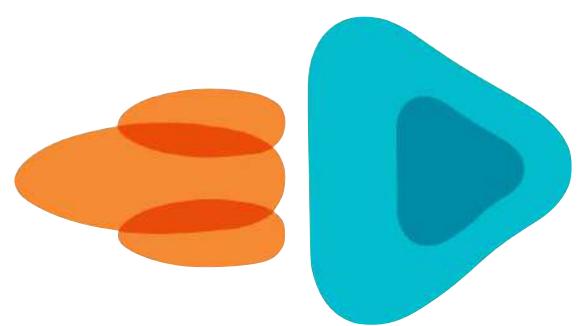
critical or important to application success

design



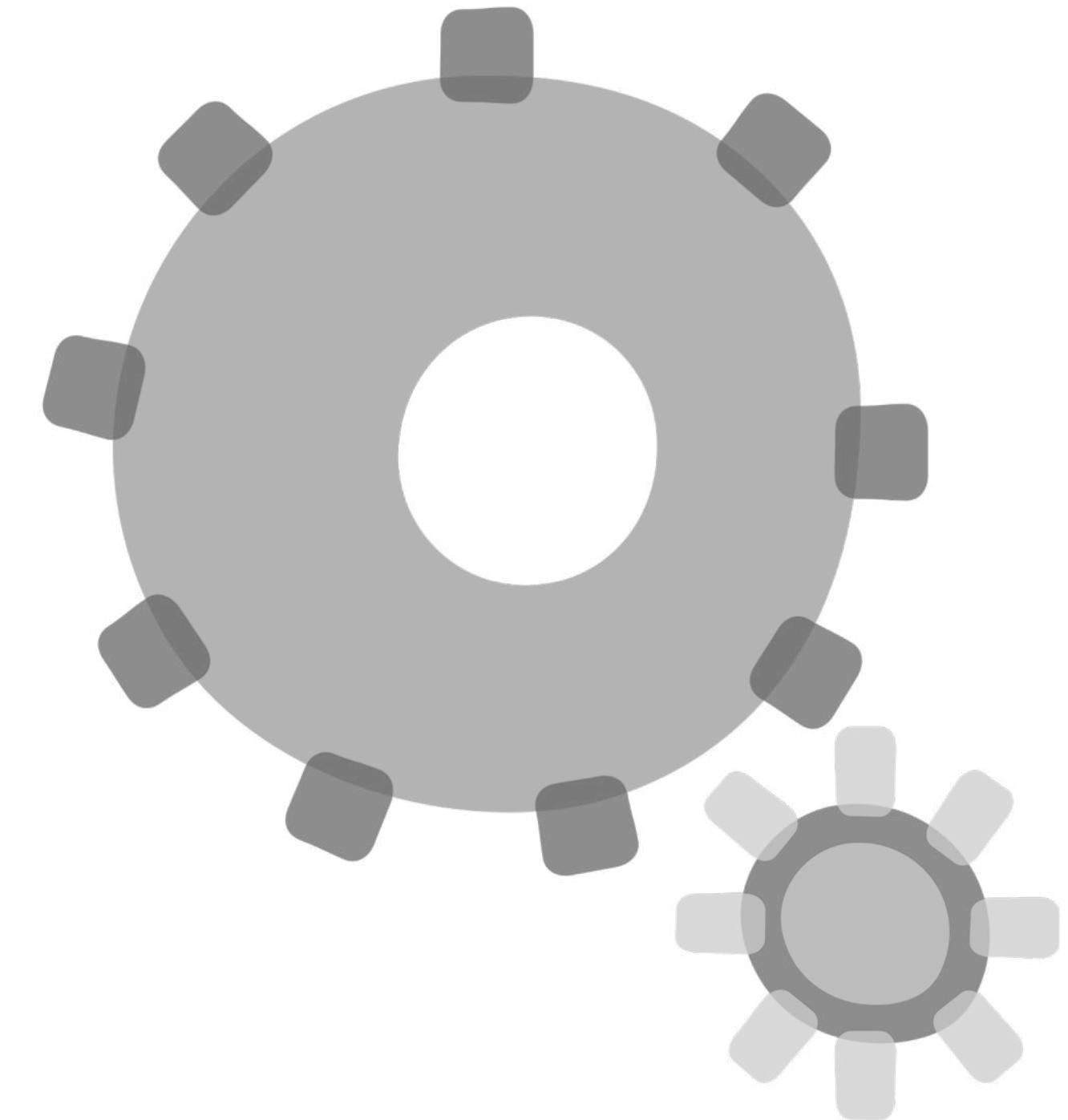
Operational Architecture Characteristics

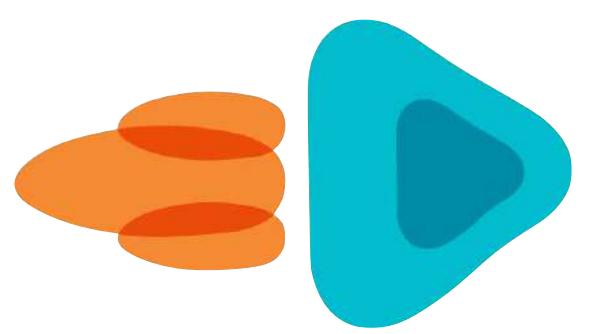




performance

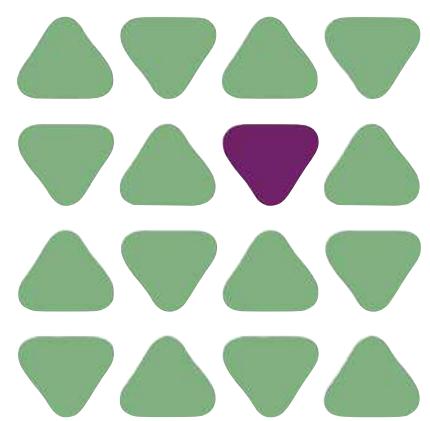
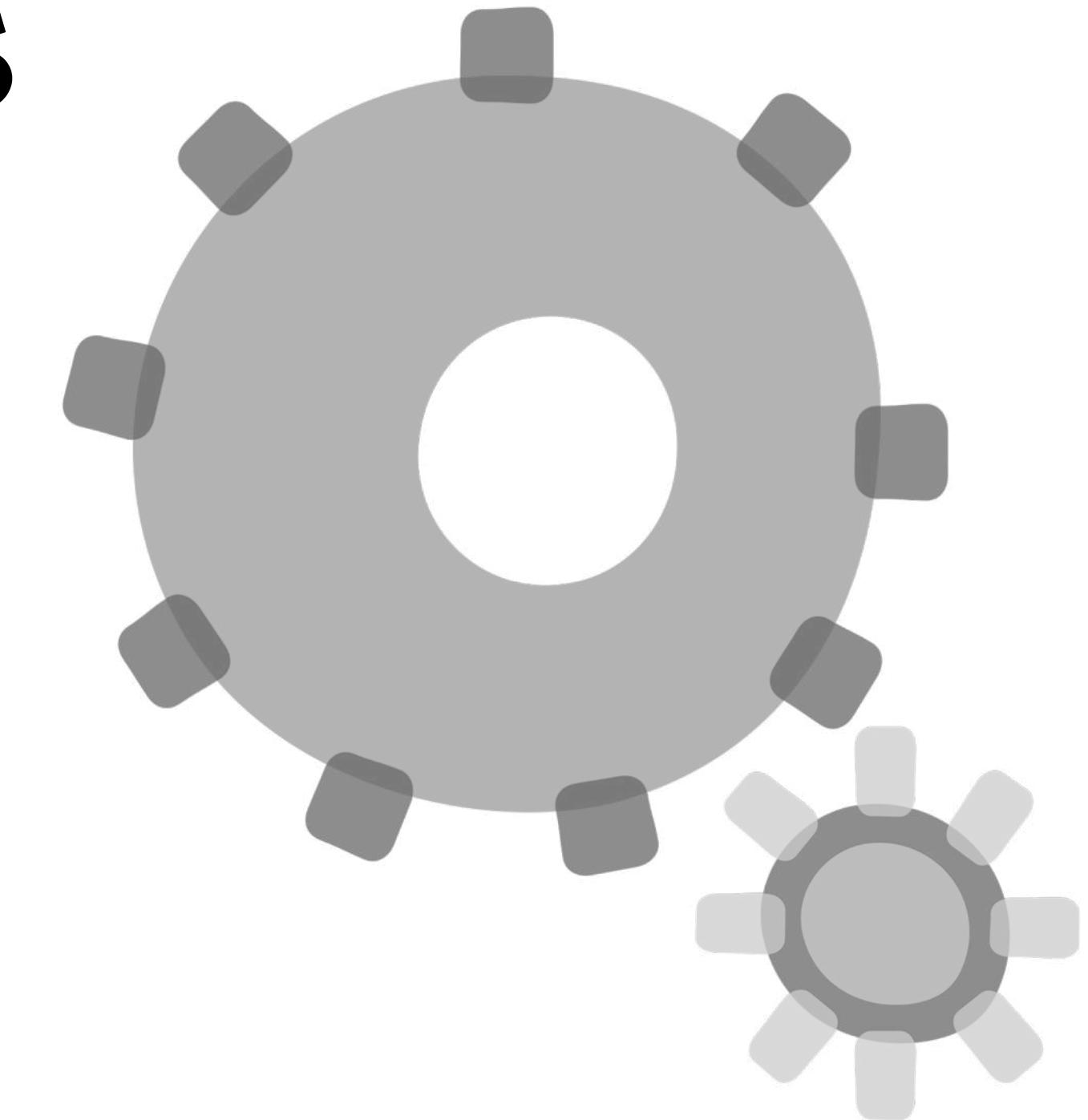
Operational Architecture Characteristics



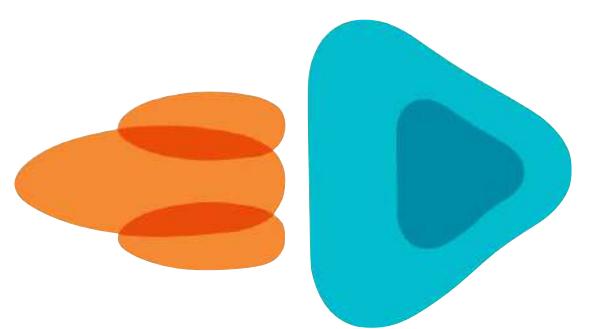


performance

Operational Architecture Characteristics

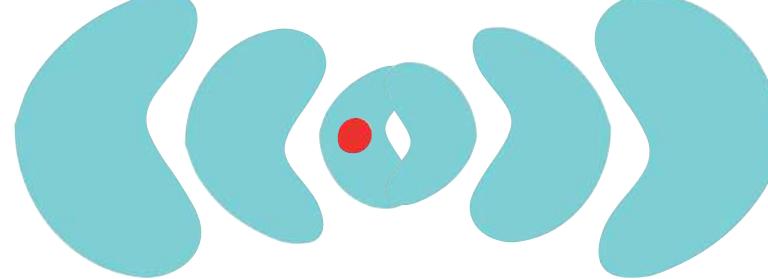


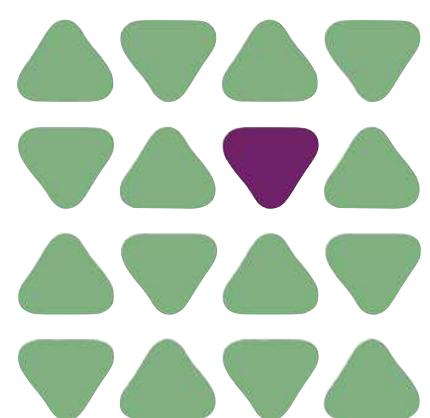
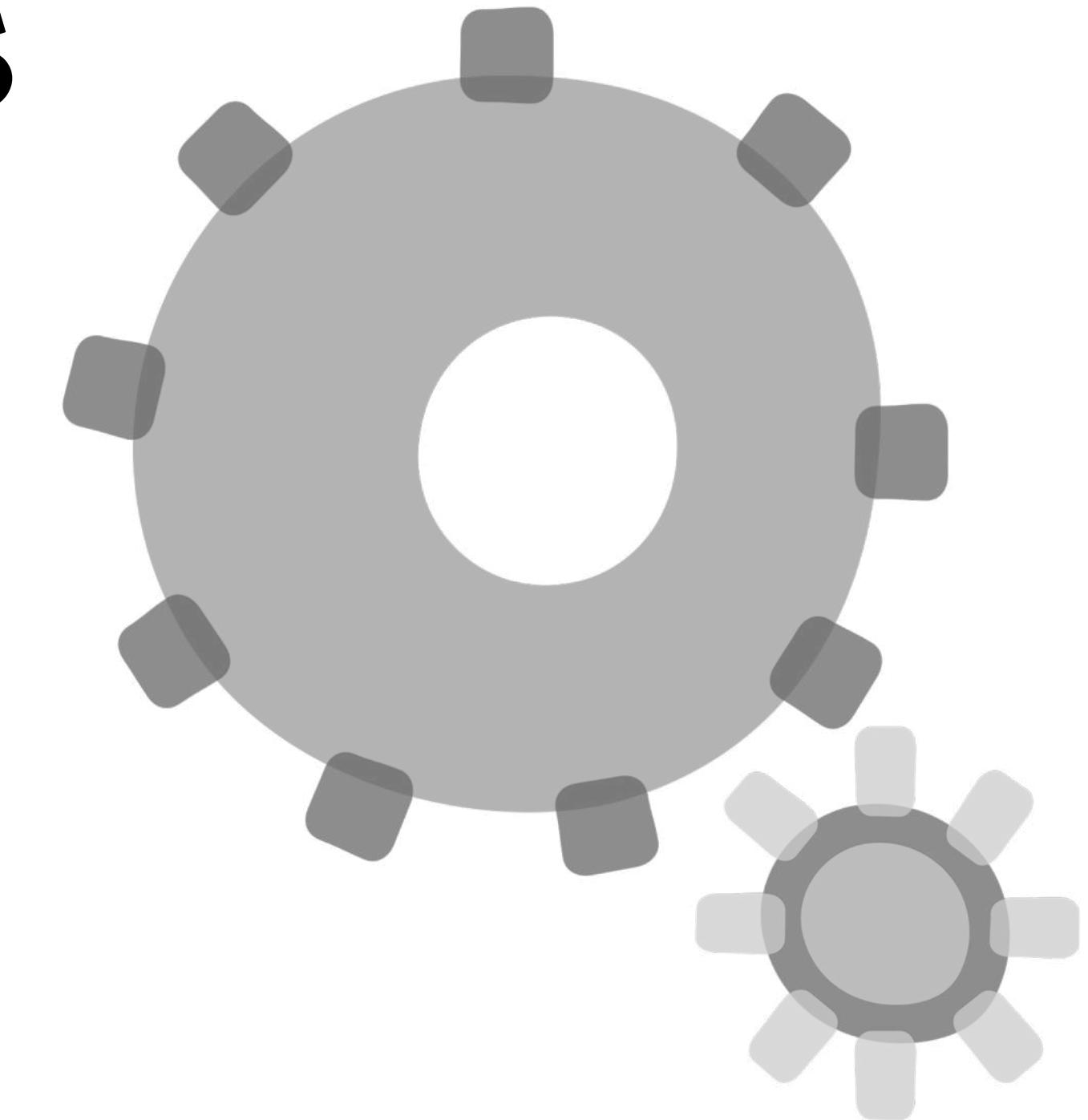
scalability



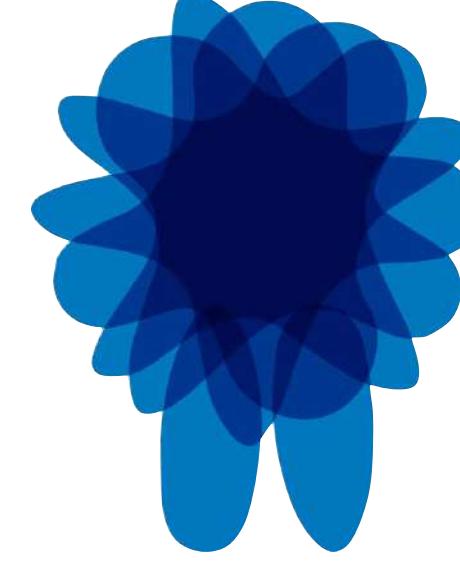
performance

Operational Architecture Characteristics

 *elasticity*

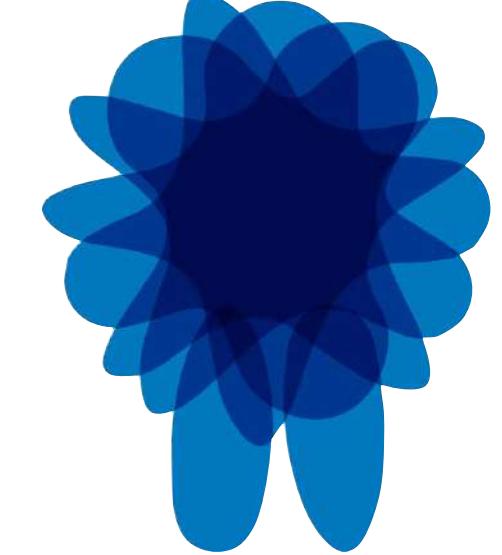


scalability



reliability

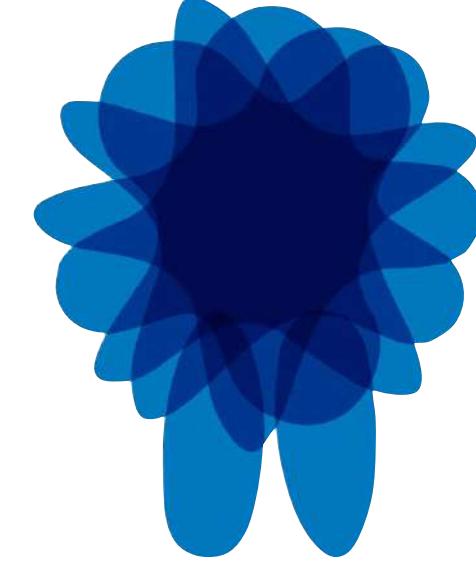
the ability of a system or component to function under stated conditions for a specified period of time.



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*?



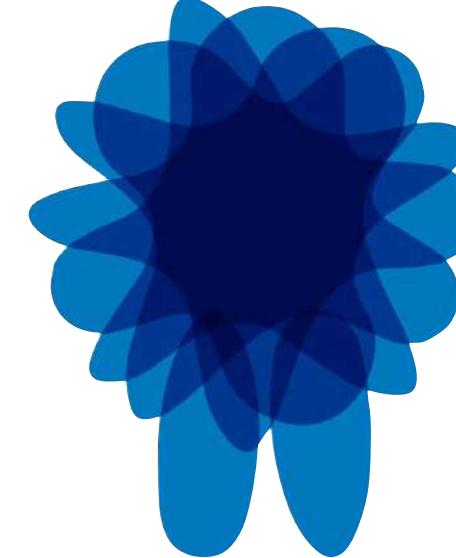
availability

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

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:

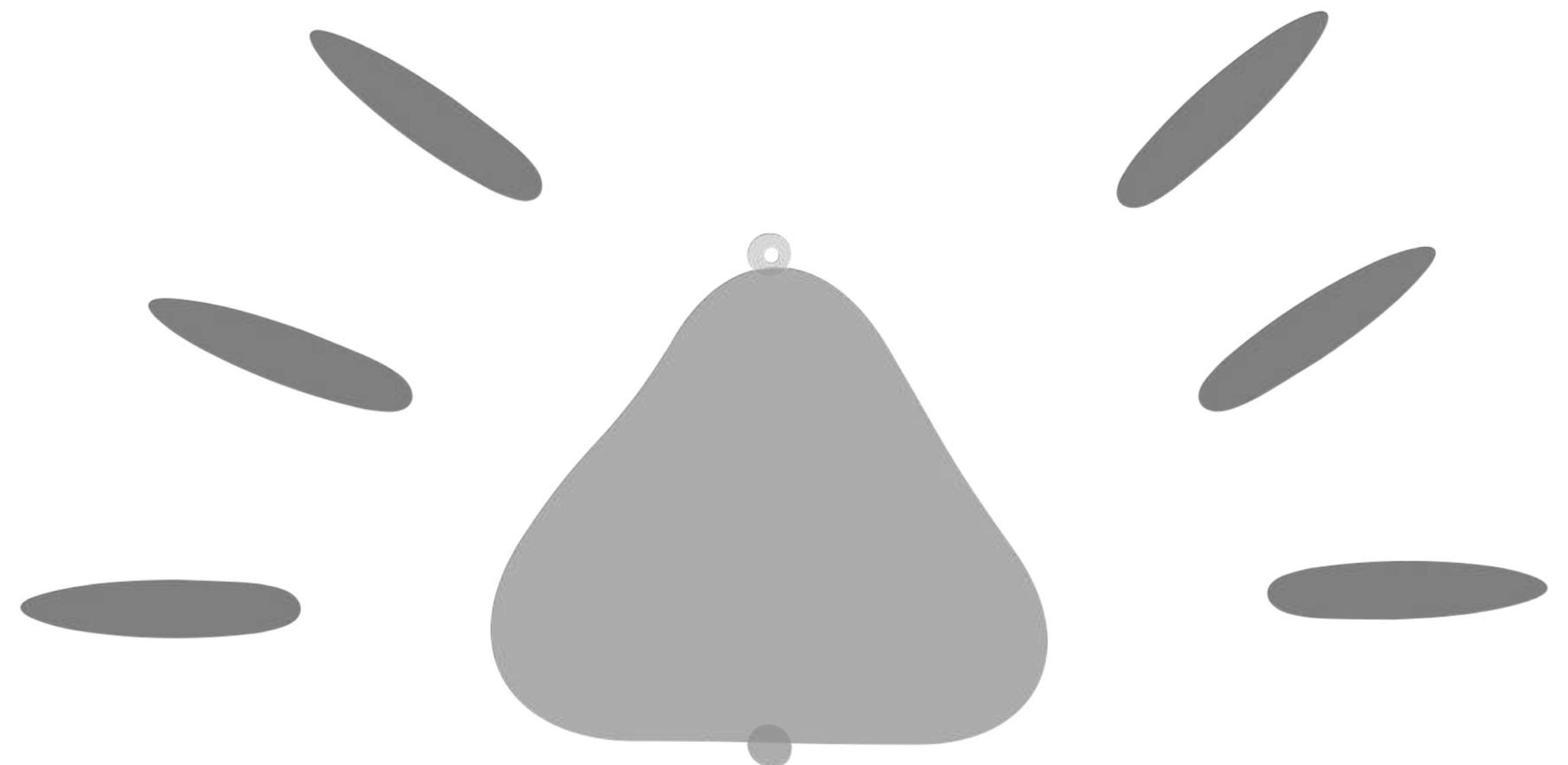
$$A_\infty = \lim_{c \rightarrow \infty} A_c = \lim_{c \rightarrow \infty} \frac{1}{c} \int_0^c A(t) dt, \quad c > 0$$

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 microseconds	2.63 microseconds	604.80 microseconds	86.40 microseconds

Internal Quality Architecture Characteristics



Scenario Exercise - Architecture Characteristics

Identify which of the following architecture characteristics are critical for the success of the sysops squad system (select no more than 7)

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

- | | | |
|---|--|---|
| <input type="checkbox"/> deployability | <input type="checkbox"/> adaptability | <input type="checkbox"/> scalability |
| <input type="checkbox"/> learnability | <input type="checkbox"/> fault tolerance | <input type="checkbox"/> recoverability |
| <input type="checkbox"/> availability | <input type="checkbox"/> workflow | <input type="checkbox"/> reliability |
| <input type="checkbox"/> performance | <input type="checkbox"/> maintainability | <input type="checkbox"/> elasticity |
| <input type="checkbox"/> extensibility | <input type="checkbox"/> responsiveness | <input type="checkbox"/> interoperability |
| <input type="checkbox"/> data integrity | <input type="checkbox"/> testability | <input type="checkbox"/> cost |

Scenario Exercise - Architecture Characteristics (Instructor)

Identify which of the following architecture characteristics are critical for the success of the sysops squad system (select no more than 7)

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

- | | | |
|---|--|---|
| <input type="checkbox"/> deployability | <input type="checkbox"/> adaptability | <input type="checkbox"/> scalability |
| <input type="checkbox"/> learnability | <input type="checkbox"/> fault tolerance | <input type="checkbox"/> recoverability |
| <input type="checkbox"/> availability | <input type="checkbox"/> workflow | <input type="checkbox"/> reliability |
| <input type="checkbox"/> performance | <input type="checkbox"/> maintainability | <input type="checkbox"/> elasticity |
| <input type="checkbox"/> extensibility | <input type="checkbox"/> responsiveness | <input type="checkbox"/> interoperability |
| <input type="checkbox"/> data integrity | <input type="checkbox"/> testability | <input type="checkbox"/> cost |

Scenario Exercise - Architecture Characteristics (Instructor)

Identify which of the following architecture characteristics are critical for the success of the sysops squad system (select no more than 7)

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

- | | | |
|--|--|---|
| <input type="checkbox"/> deployability | <input type="checkbox"/> adaptability | <input type="checkbox"/> scalability |
| <input type="checkbox"/> learnability | <input type="checkbox"/> fault tolerance | <input type="checkbox"/> recoverability |
| <input type="checkbox"/> availability | <input checked="" type="checkbox"/> workflow | <input checked="" type="checkbox"/> reliability |
| <input type="checkbox"/> performance | <input type="checkbox"/> maintainability | <input type="checkbox"/> elasticity |
| <input type="checkbox"/> extensibility | <input type="checkbox"/> responsiveness | <input type="checkbox"/> interoperability |
| <input checked="" type="checkbox"/> data integrity | <input type="checkbox"/> testability | <input type="checkbox"/> cost |

Scenario Exercise - Architecture Characteristics (Instructor)

Identify which of the following architecture characteristics are critical for the success of the sysops squad system (select no more than 7)

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

- | | | |
|--|--|---|
| <input type="checkbox"/> deployability | <input type="checkbox"/> adaptability | <input type="checkbox"/> scalability |
| <input type="checkbox"/> learnability | <input type="checkbox"/> fault tolerance | <input type="checkbox"/> recoverability |
| <input checked="" type="checkbox"/> availability | <input checked="" type="checkbox"/> workflow | <input checked="" type="checkbox"/> reliability |
| <input type="checkbox"/> performance | <input type="checkbox"/> maintainability | <input type="checkbox"/> elasticity |
| <input type="checkbox"/> extensibility | <input type="checkbox"/> responsiveness | <input type="checkbox"/> interoperability |
| <input checked="" type="checkbox"/> data integrity | <input type="checkbox"/> testability | <input type="checkbox"/> cost |

Scenario Exercise - Architecture Characteristics (Instructor)

Identify which of the following architecture characteristics are critical for the success of the sysops squad system (select no more than 7)

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

- | | | |
|--|---|---|
| <input type="checkbox"/> deployability | <input type="checkbox"/> adaptability | <input type="checkbox"/> scalability |
| <input type="checkbox"/> learnability | <input type="checkbox"/> fault tolerance | <input type="checkbox"/> recoverability |
| <input checked="" type="checkbox"/> availability | <input checked="" type="checkbox"/> workflow | <input checked="" type="checkbox"/> reliability |
| <input type="checkbox"/> performance | <input checked="" type="checkbox"/> maintainability | <input type="checkbox"/> elasticity |
| <input type="checkbox"/> extensibility | <input type="checkbox"/> responsiveness | <input type="checkbox"/> interoperability |
| <input checked="" type="checkbox"/> data integrity | <input checked="" type="checkbox"/> testability | <input type="checkbox"/> cost |

Scenario Exercise - Architecture Characteristics (Instructor)

Identify which of the following architecture characteristics are critical for the success of the sysops squad system (select no more than 7)

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

- | | | |
|--|---|---|
| <input type="checkbox"/> deployability | <input type="checkbox"/> adaptability | <input checked="" type="checkbox"/> scalability |
| <input type="checkbox"/> learnability | <input type="checkbox"/> fault tolerance | <input type="checkbox"/> recoverability |
| <input checked="" type="checkbox"/> availability | <input checked="" type="checkbox"/> workflow | <input checked="" type="checkbox"/> reliability |
| <input type="checkbox"/> performance | <input checked="" type="checkbox"/> maintainability | <input type="checkbox"/> elasticity |
| <input type="checkbox"/> extensibility | <input type="checkbox"/> responsiveness | <input type="checkbox"/> interoperability |
| <input checked="" type="checkbox"/> data integrity | <input checked="" type="checkbox"/> testability | <input type="checkbox"/> cost |

Quality Architecture Characteristics



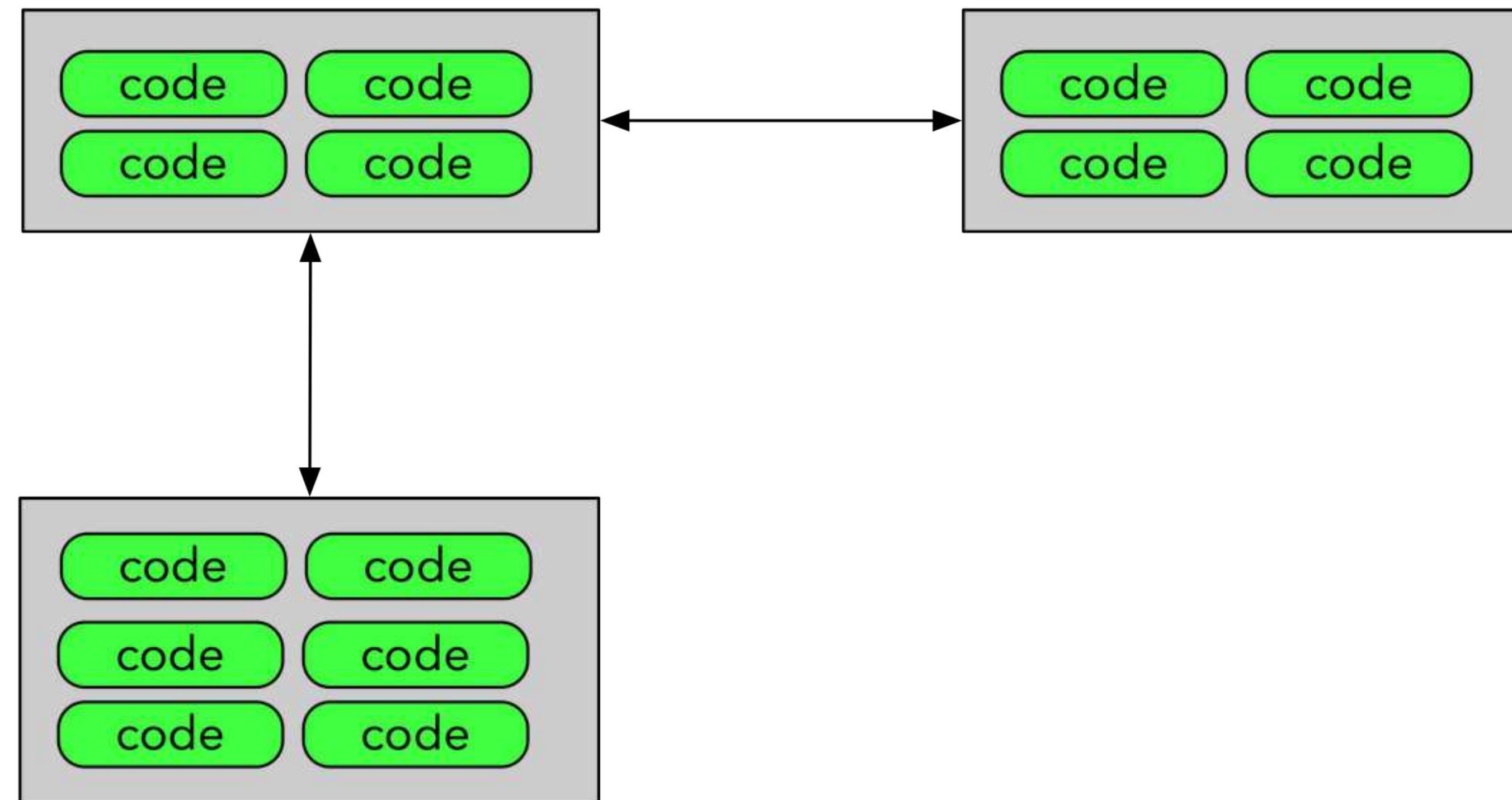


How do I choose an appropriate architecture?

1. Identify architecture characteristics
2. Identify the scope and number of distinct architecture characteristics

component coupling

the extent to which components know
about each other



connascence

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



connascence.io

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.

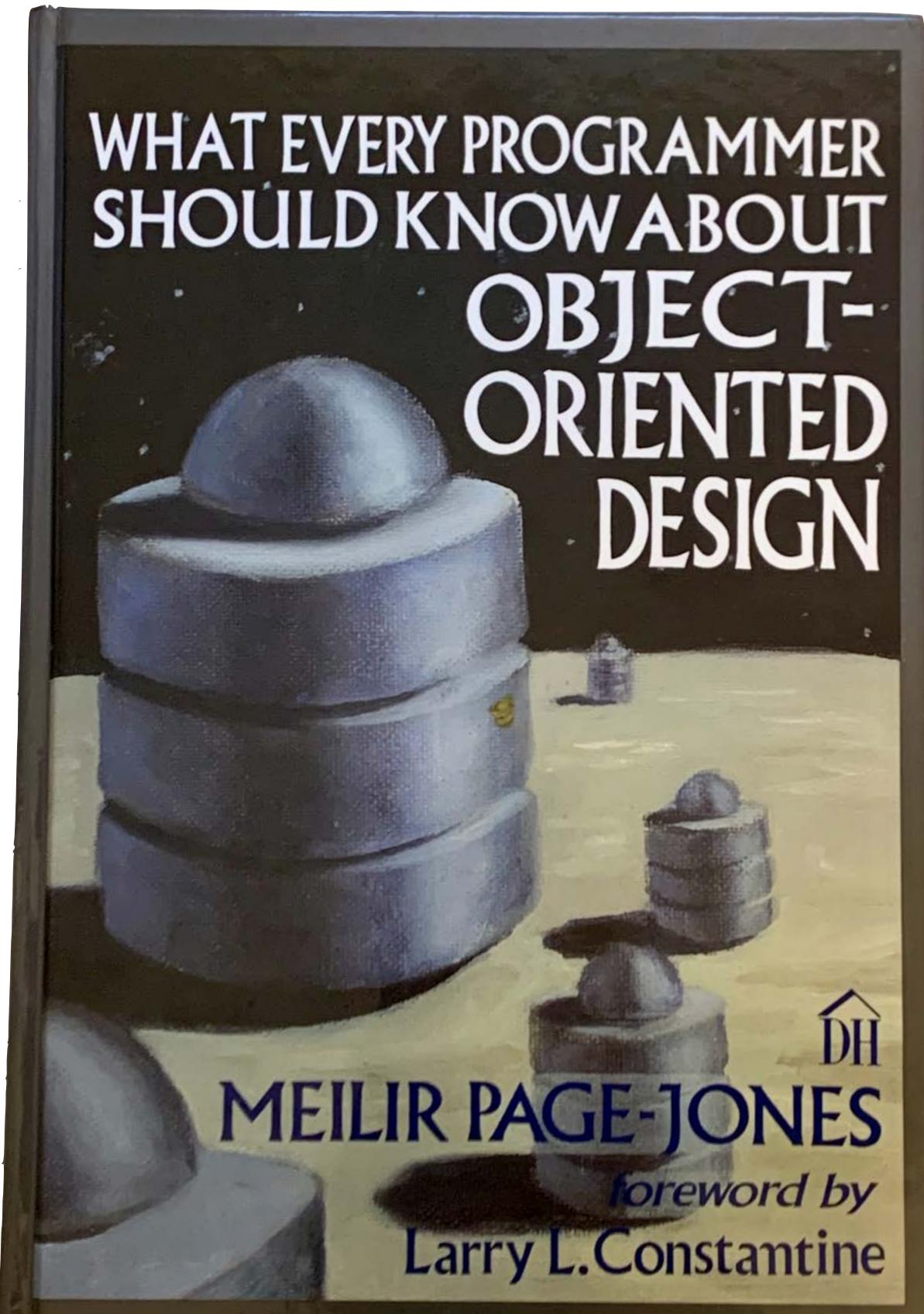
The three properties of Strength, Degree, and Locality give the programmer all the tools they need in order to make informed decisions about when they will permit certain types of coupling, and when the code ought to be refactored.

A Vocabulary for Coupling

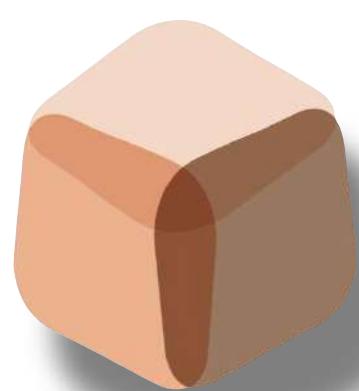
Arguably one of the most important benefits of connascence is that it gives developers a vocabulary to talk about different types of coupling. Connascence codifies what many experienced engineers have learned by trial and error: Having a common set of nouns to refer to different types of coupling allows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

(CC) BY-SA This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

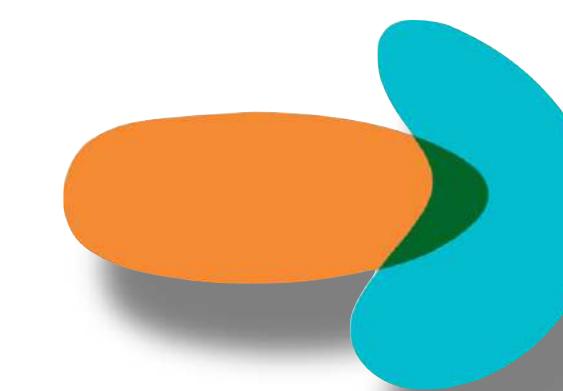
connascence



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

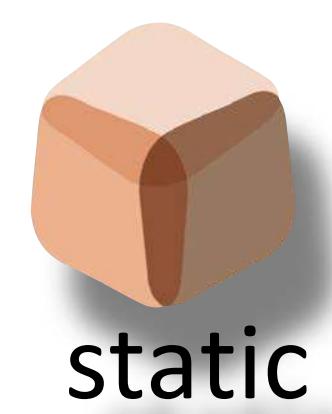
value

identity



static

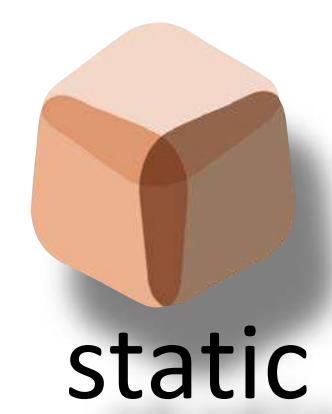
dynamic



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

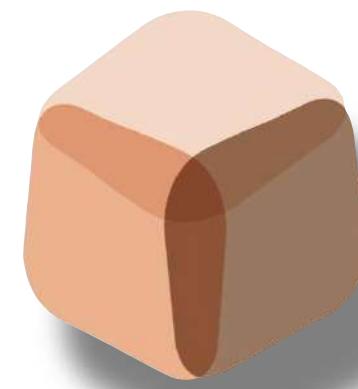
connascence of meaning

multiple components must agree on the meaning of particular values

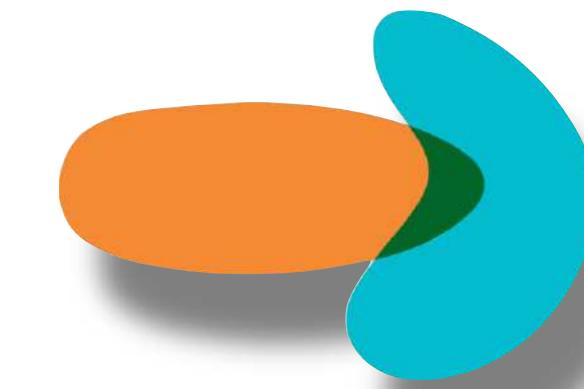
```
def valid_credit_card_number?(cc_number)
  # check for "test" credit card numbers
  if cc_number == "9999-9999-9999-9999"
    # test verification
  else
    # some more code
  end
end
```

How would you fix this ?!?

connascence properties



static

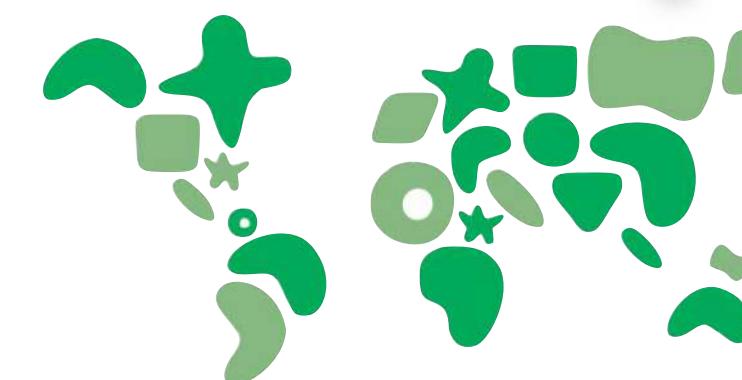


dynamic

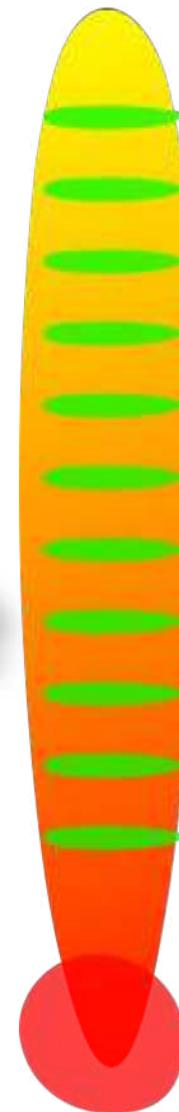
Strength



Locality



Degree



connascence properties

Strength



name
type
meaning
algorithm
position
execution order
timing
value
identity

refactor
this way

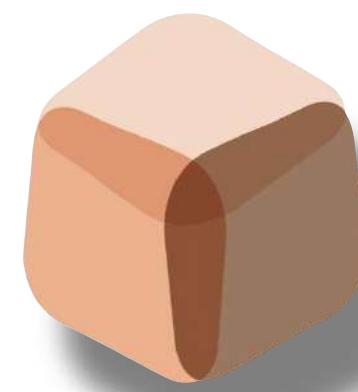
A green arrow points diagonally upwards from the bottom-left towards the top-right, passing through the word "refactor".



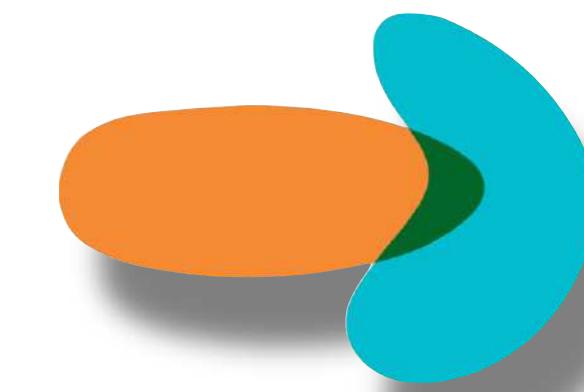
static

dynamic

connascence properties



static

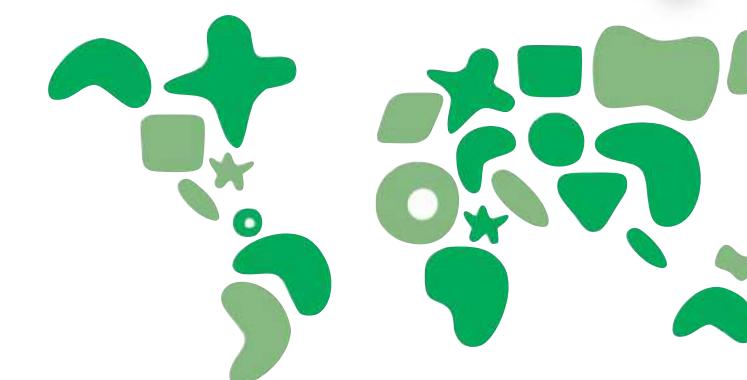


dynamic

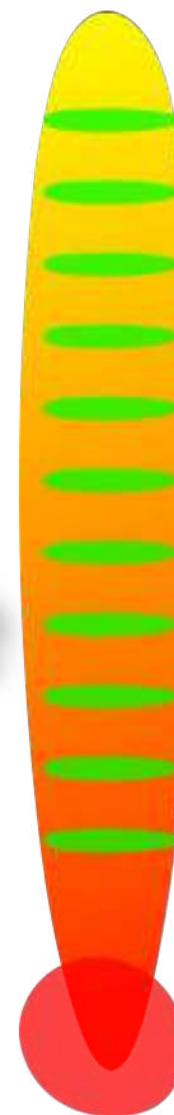
Strength



Locality

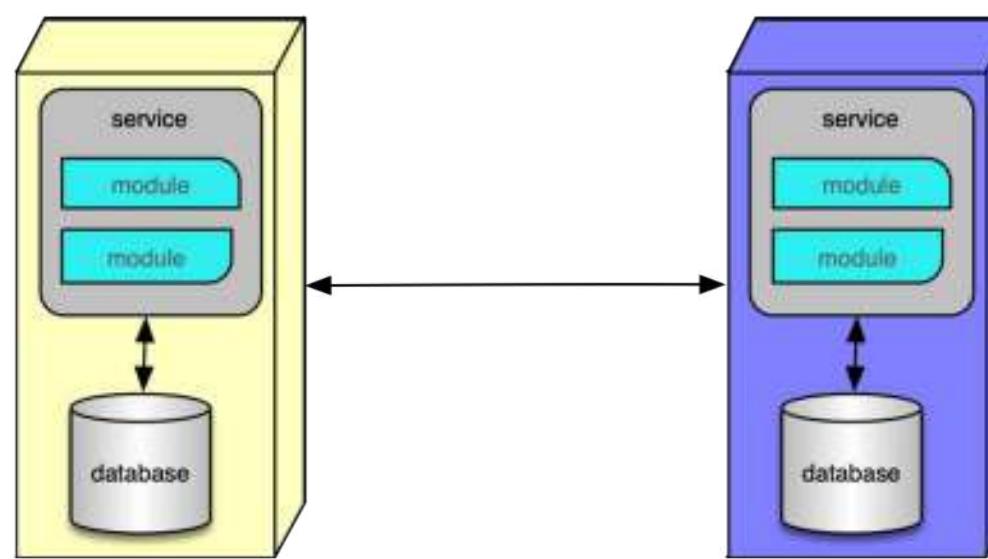
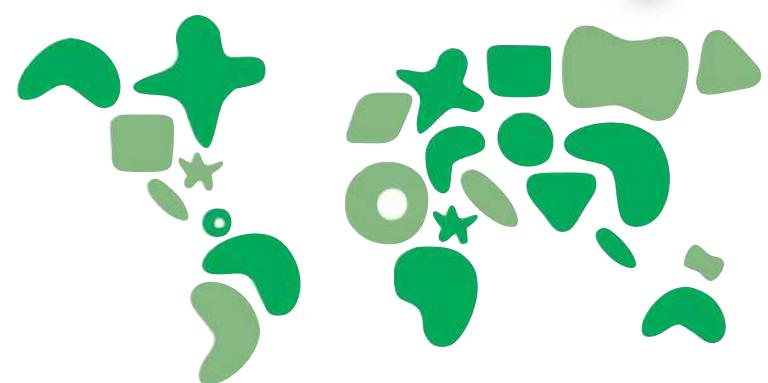


Degree



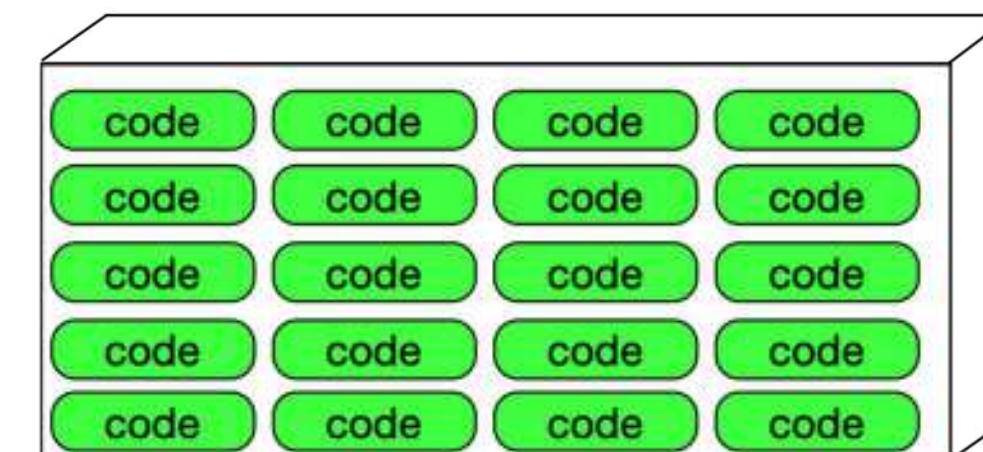
connascence properties

Locality

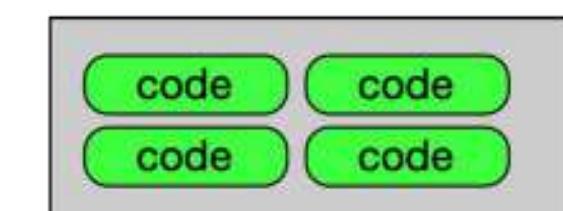


static

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).

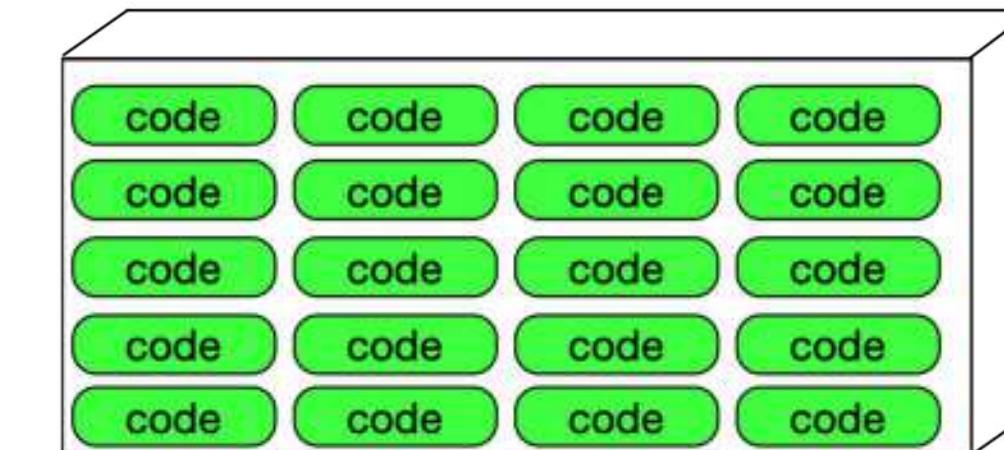
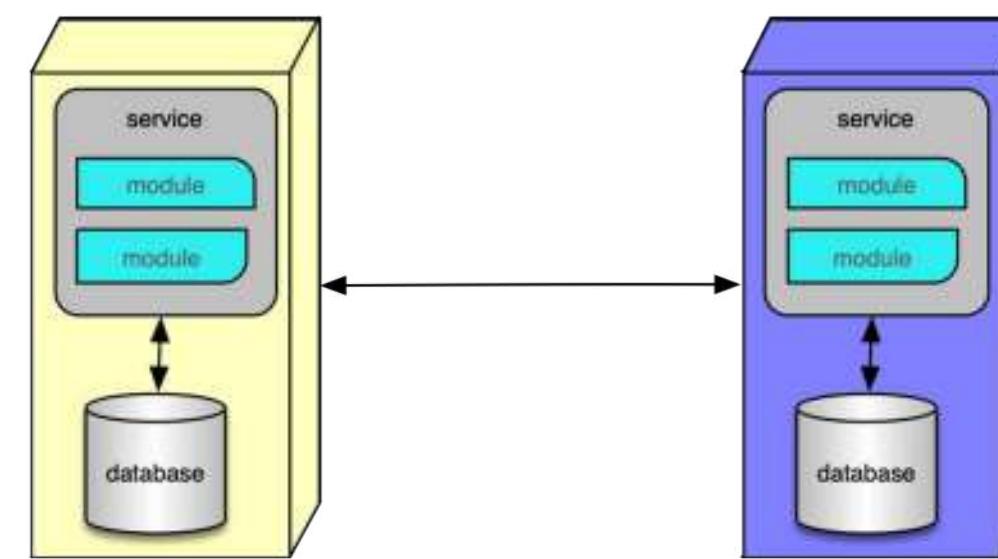
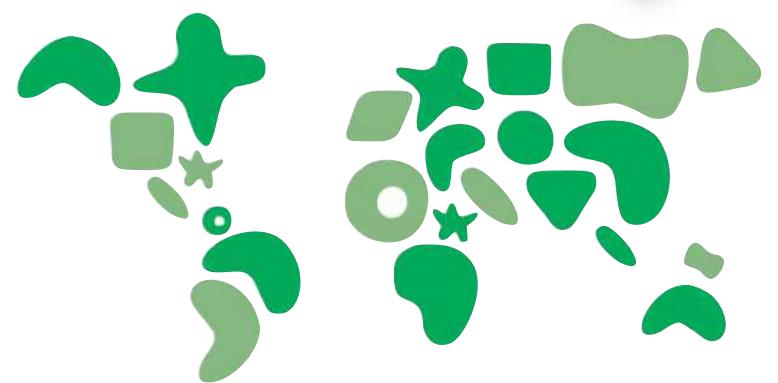


dynamic

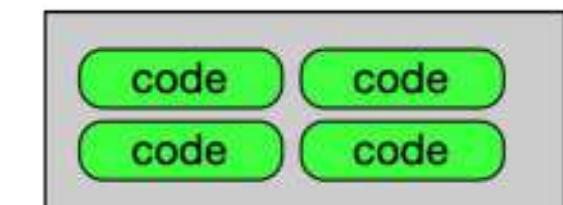


connascence properties

Locality



locality matters!



connascence

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



connascence.io

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.

(CC) BY-SA This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

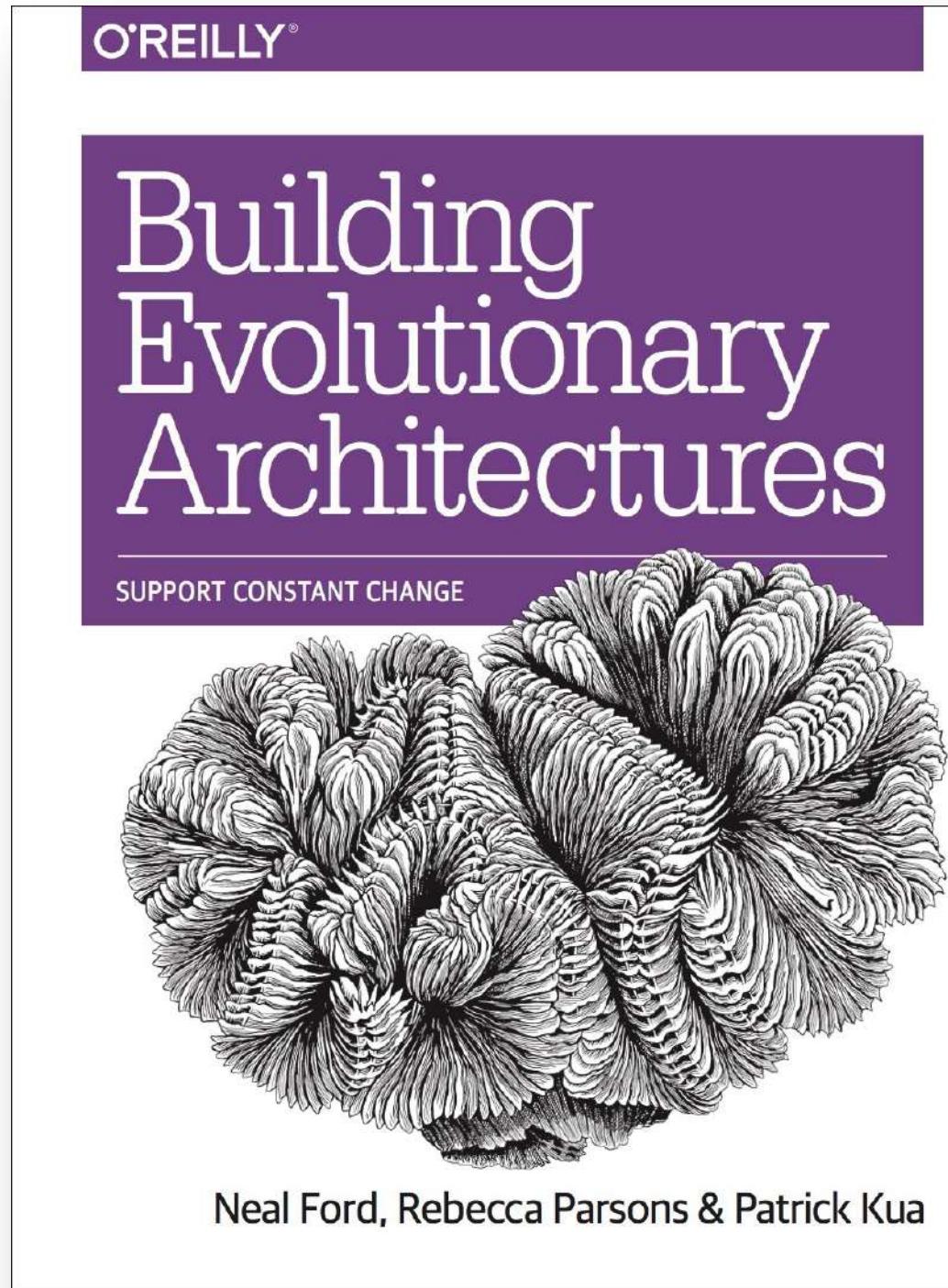
the problems w/ 90's connasence



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

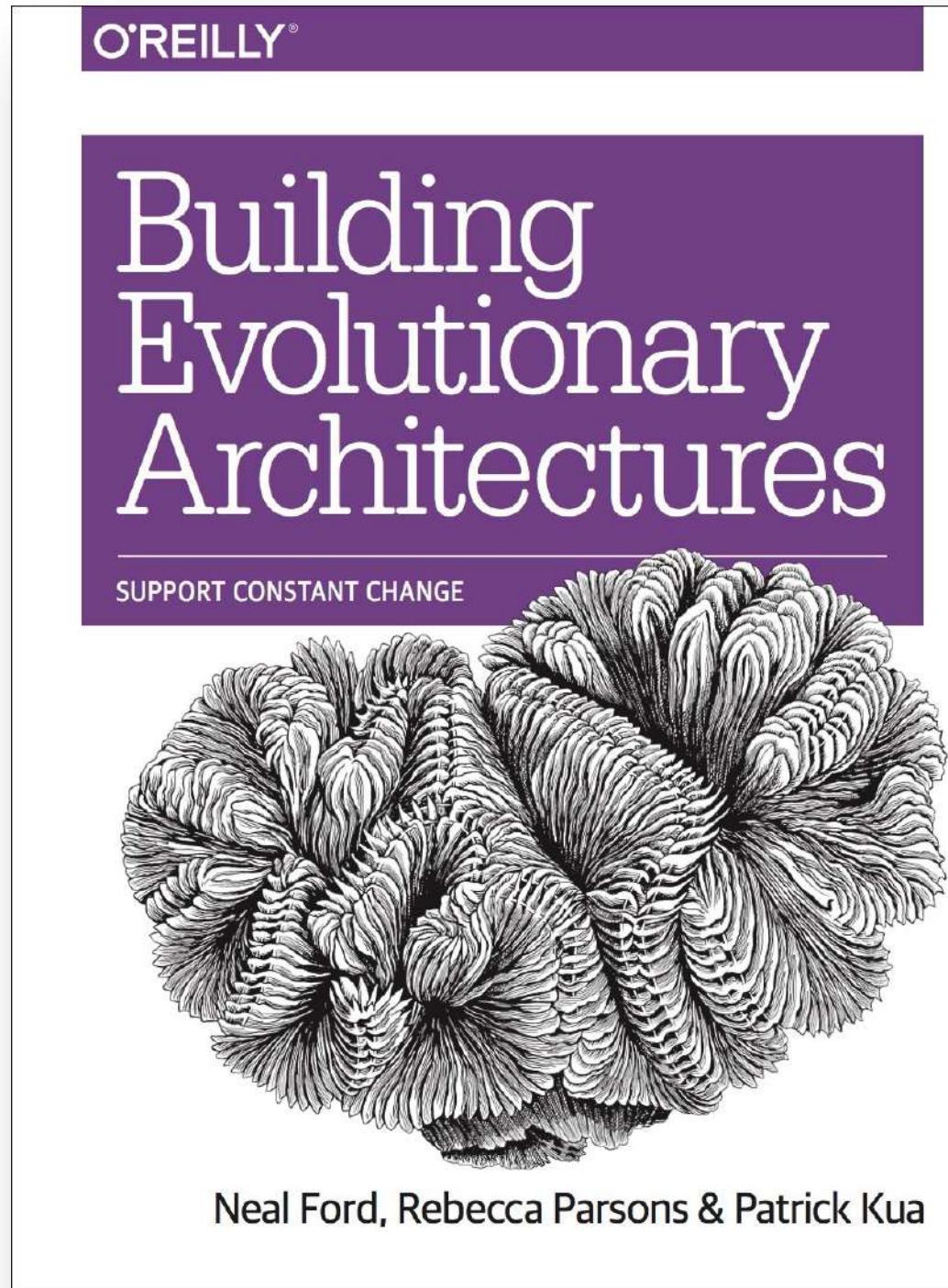
quantum connascence

architectural quantum



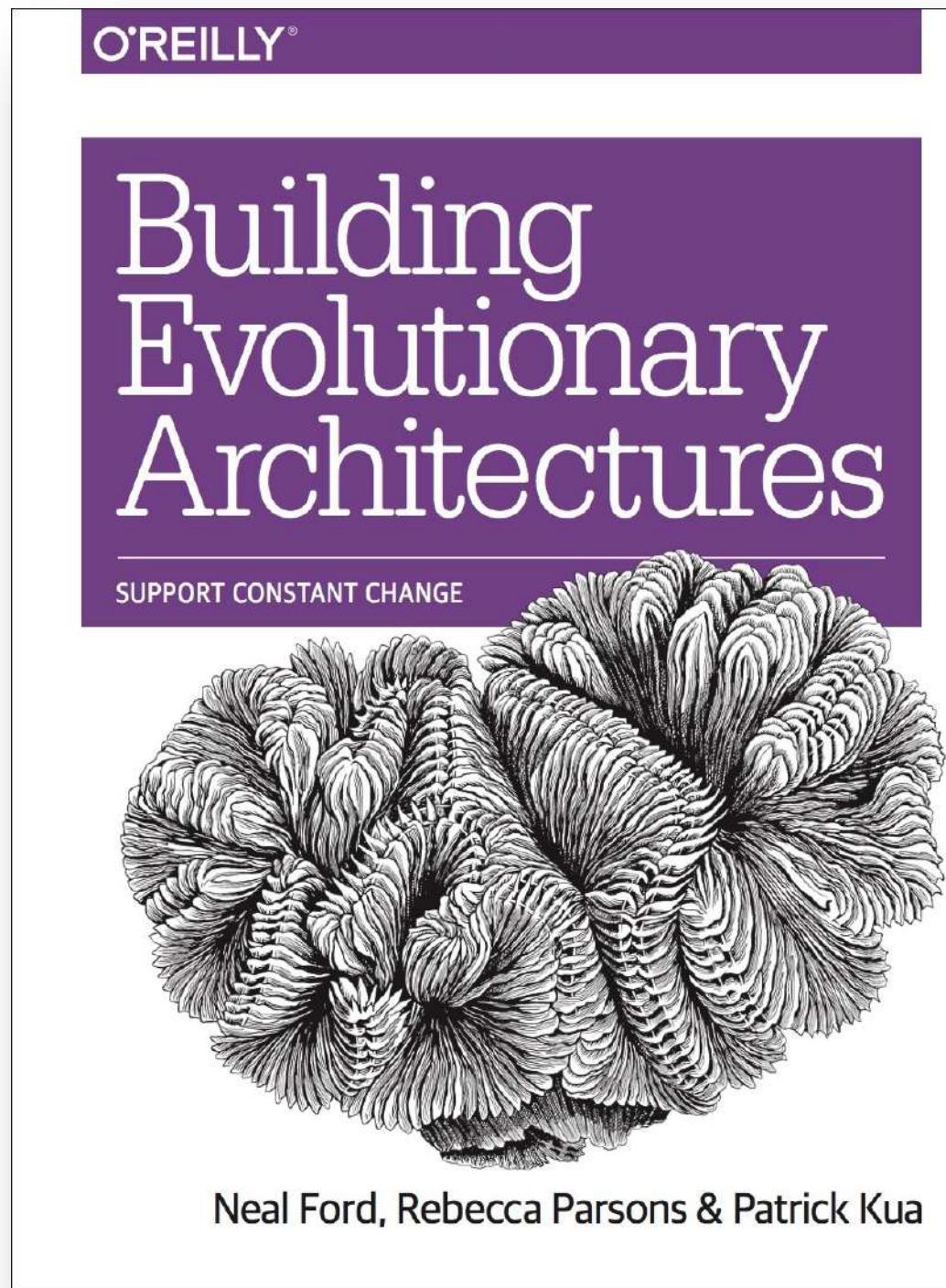
An *architectural quantum* is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

architectural quantum



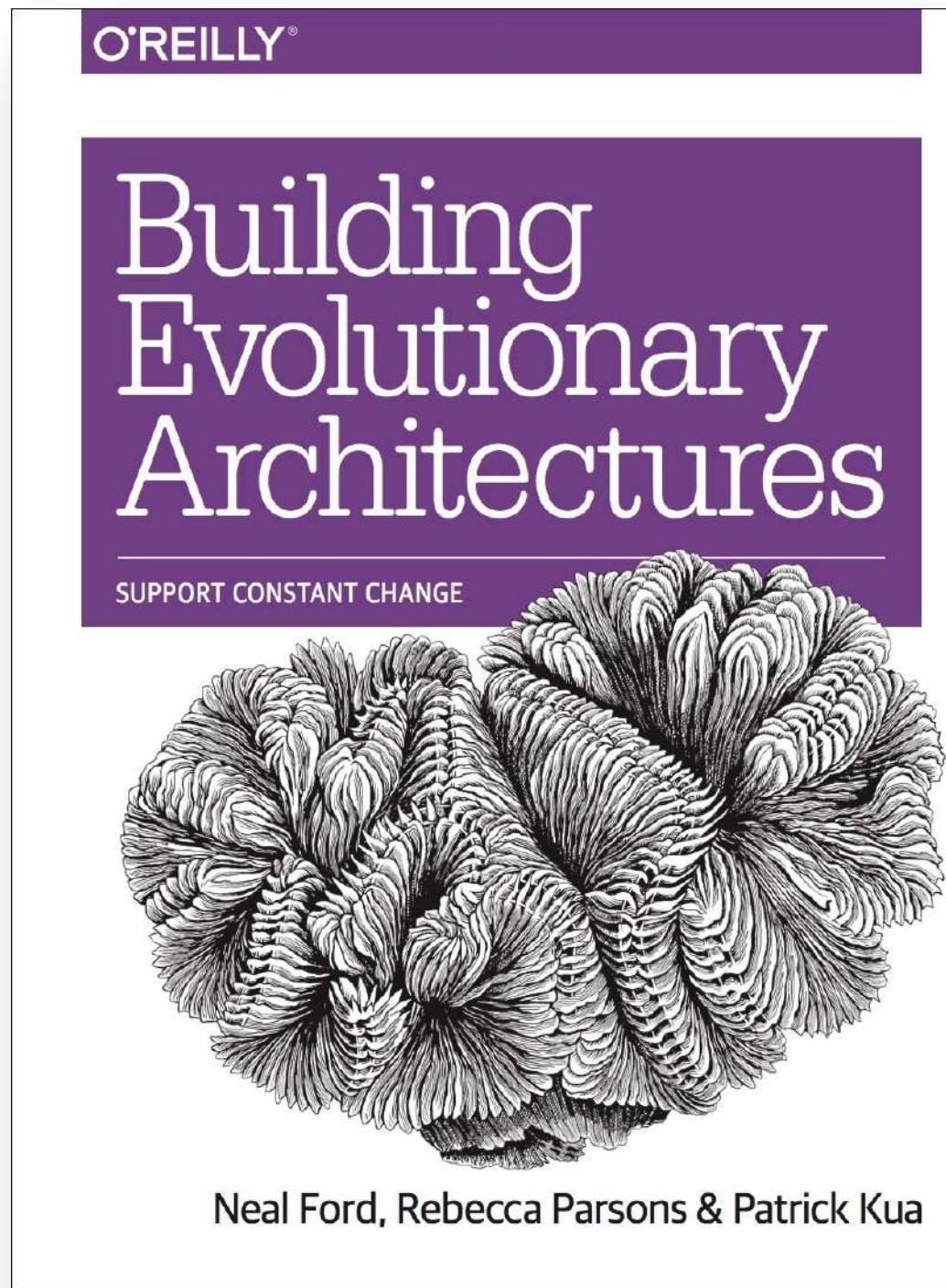
An architectural quantum is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

architectural quantum



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

architectural quantum

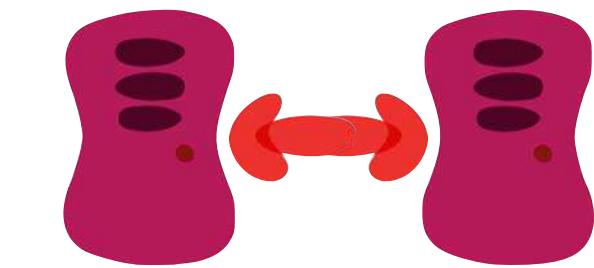


An architectural quantum is an independently deployable component with high functional cohesion and synchronous dynamic quantum connascence.

quantum connascence



dynamic



quantum connascence



static

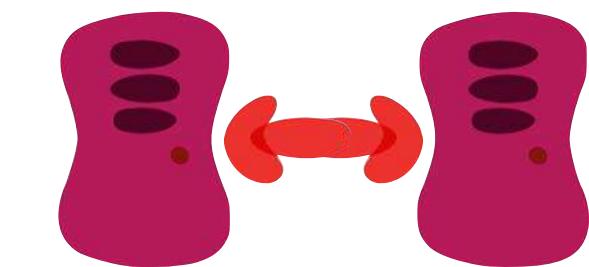
contracts



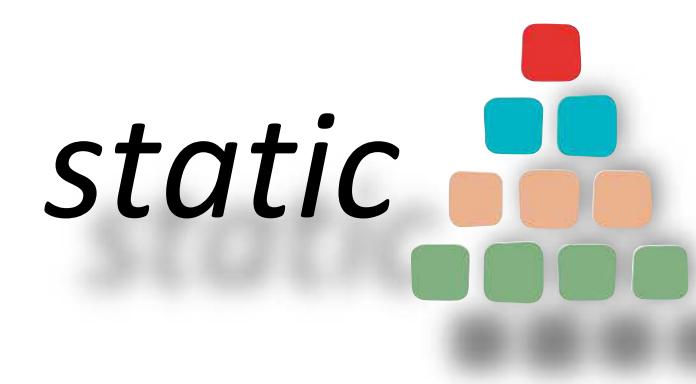
tight

loose

dynamic



quantum connascence



static

contracts



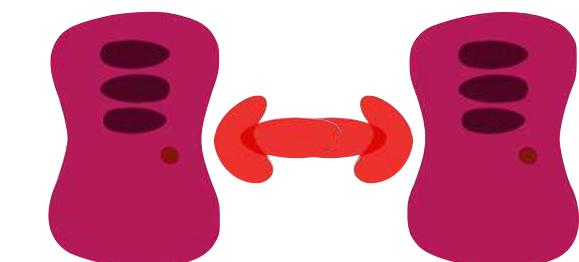
tight

loose

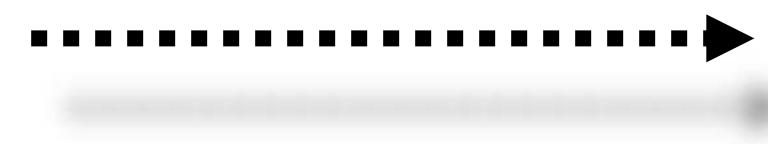
synchronous



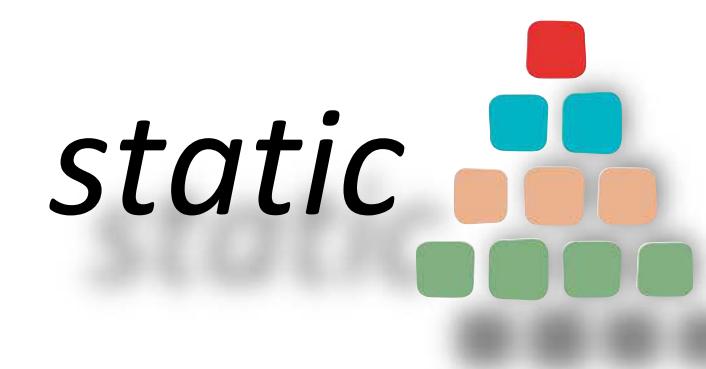
dynamic



asynchronous



quantum connascence



static

contracts



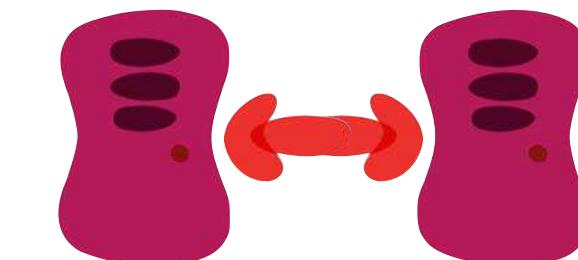
tight

loose

synchronous

quantum

asynchronous

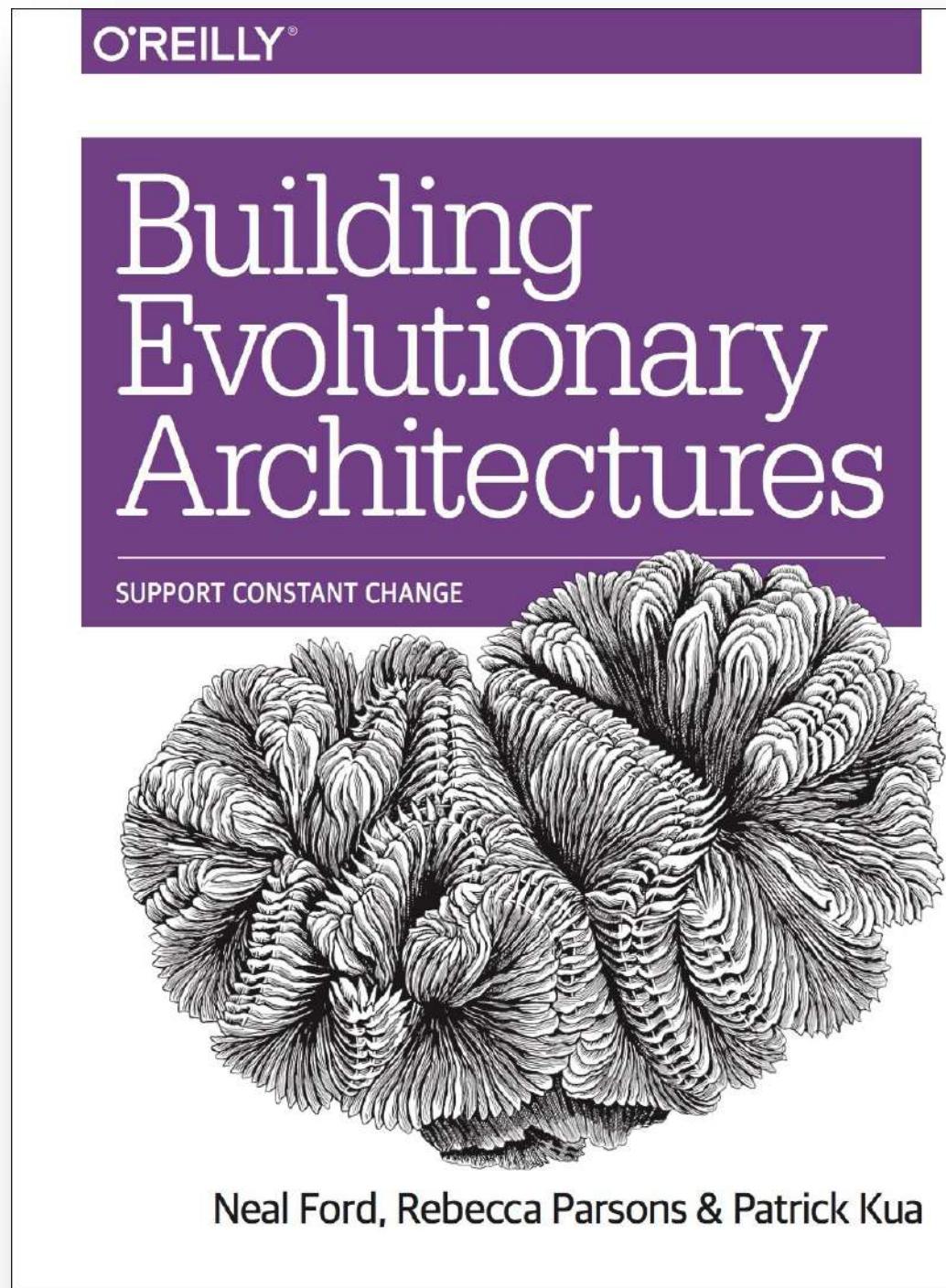


how quanta communicate

impact on operational
(and other) architecture characteristics

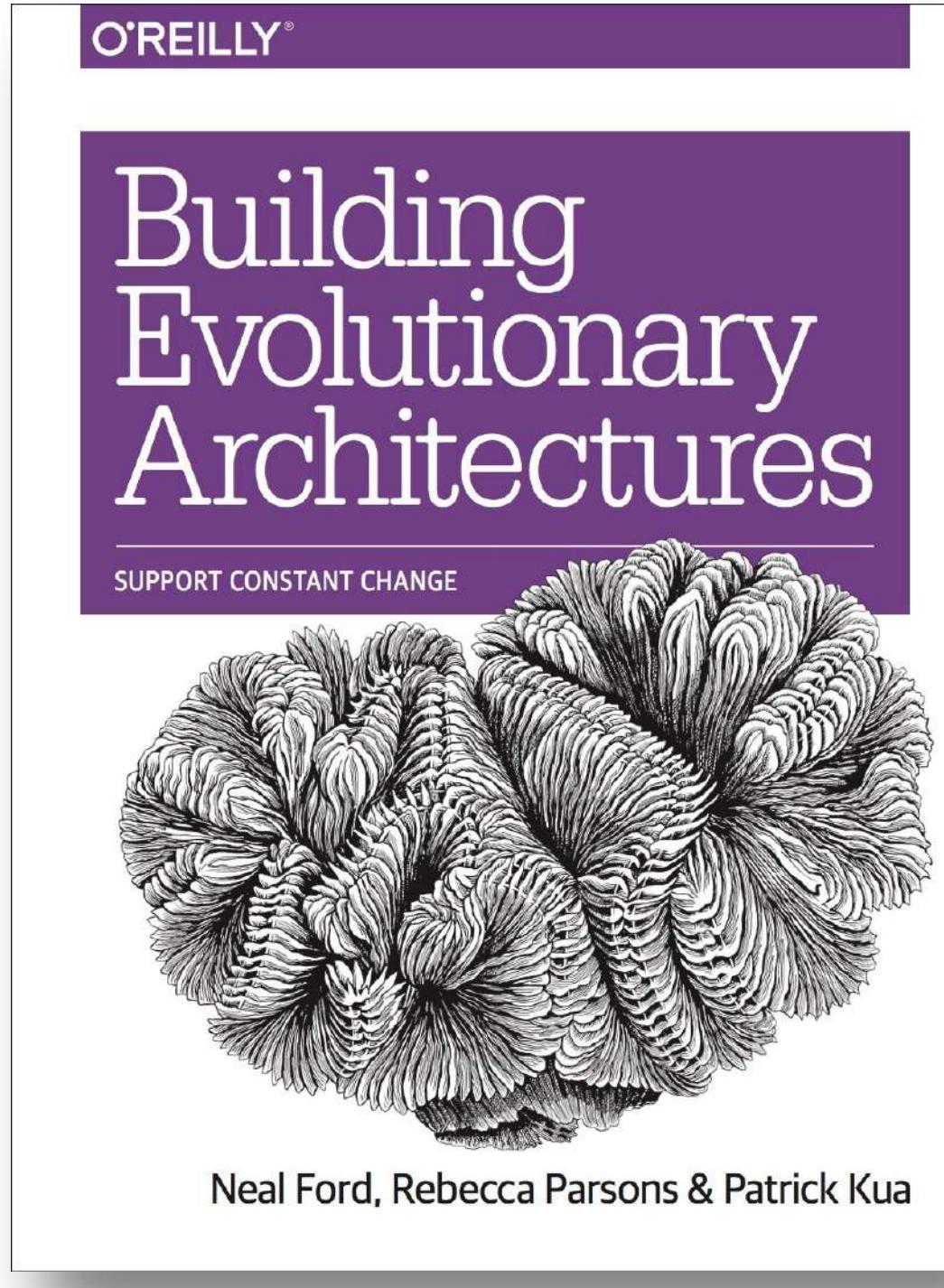
useful for hybrid architecture design,
architecture migration, integration, etc.

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*



How do I choose an appropriate architecture?

1. Identify architecture characteristics
2. Identify the scope and number of distinct architecture characteristics

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

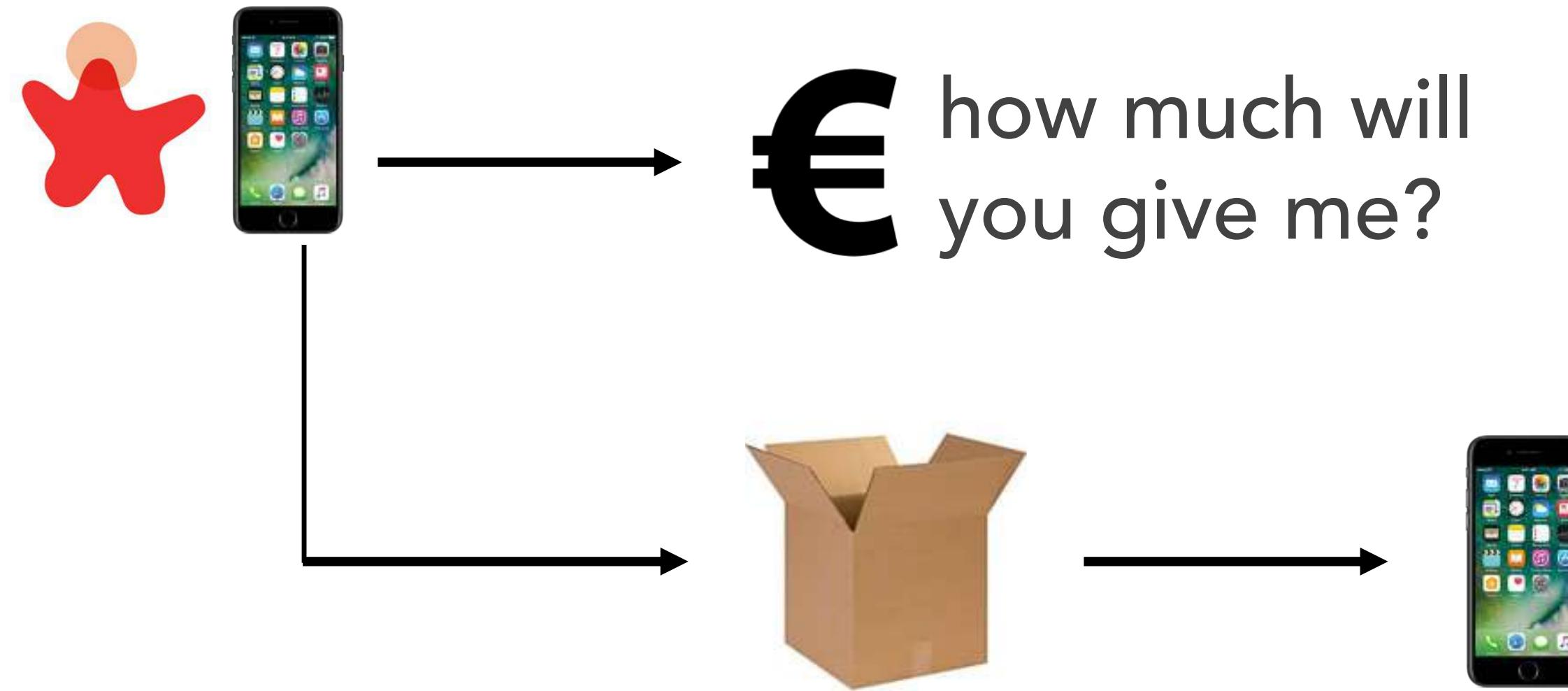
architectural quantum

electronics recycling



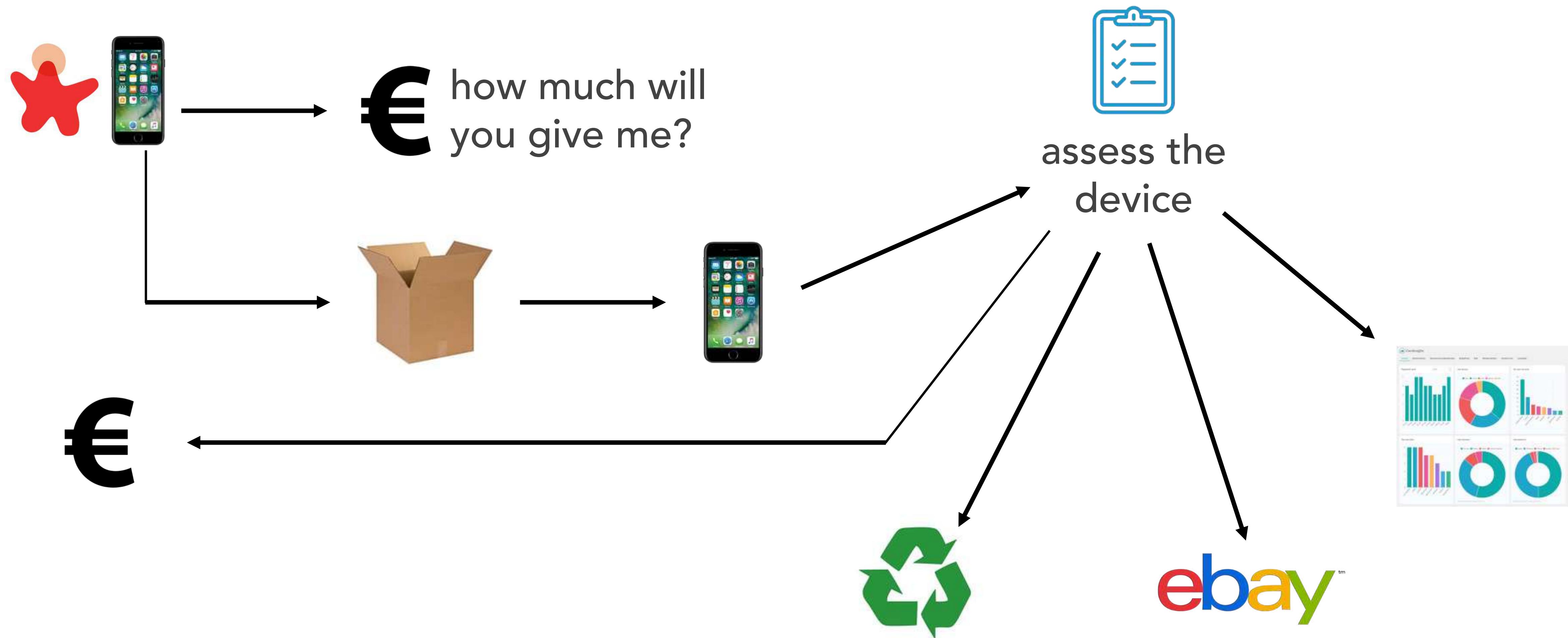
architectural quantum

electronics recycling



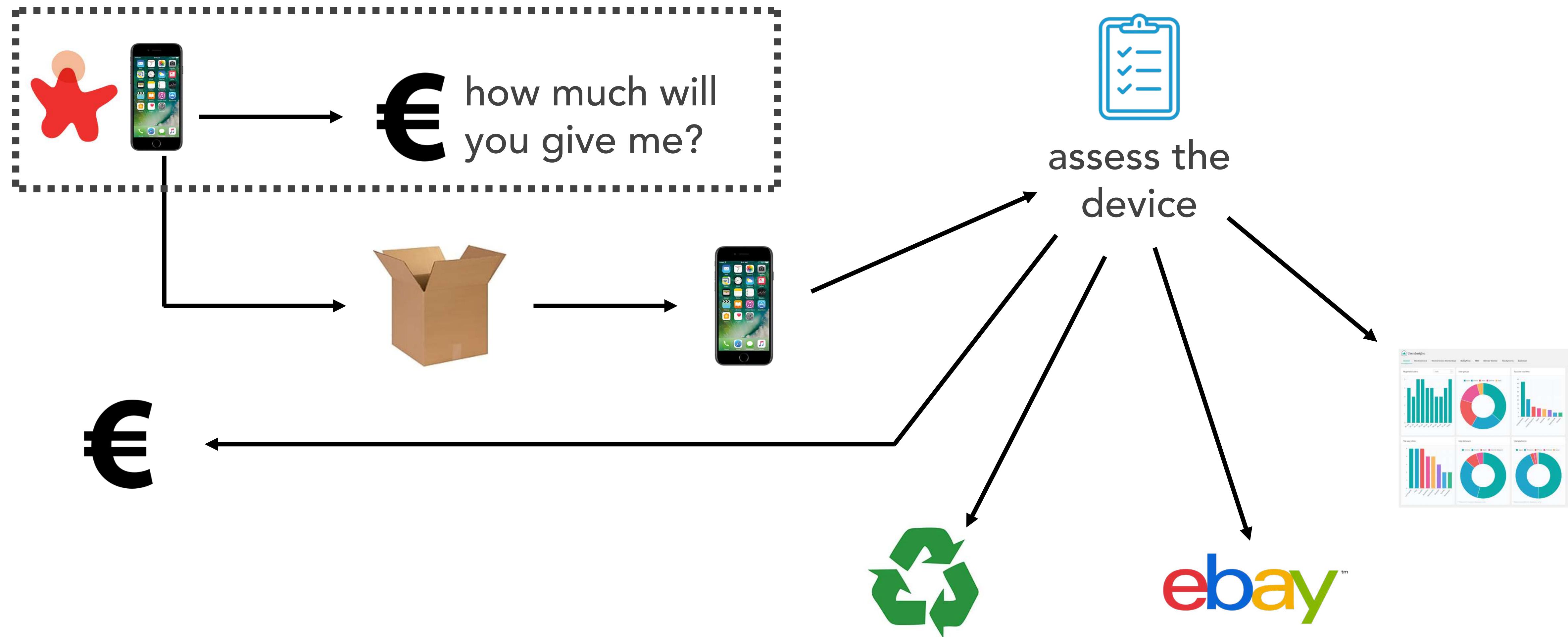
architectural quantum

electronics recycling



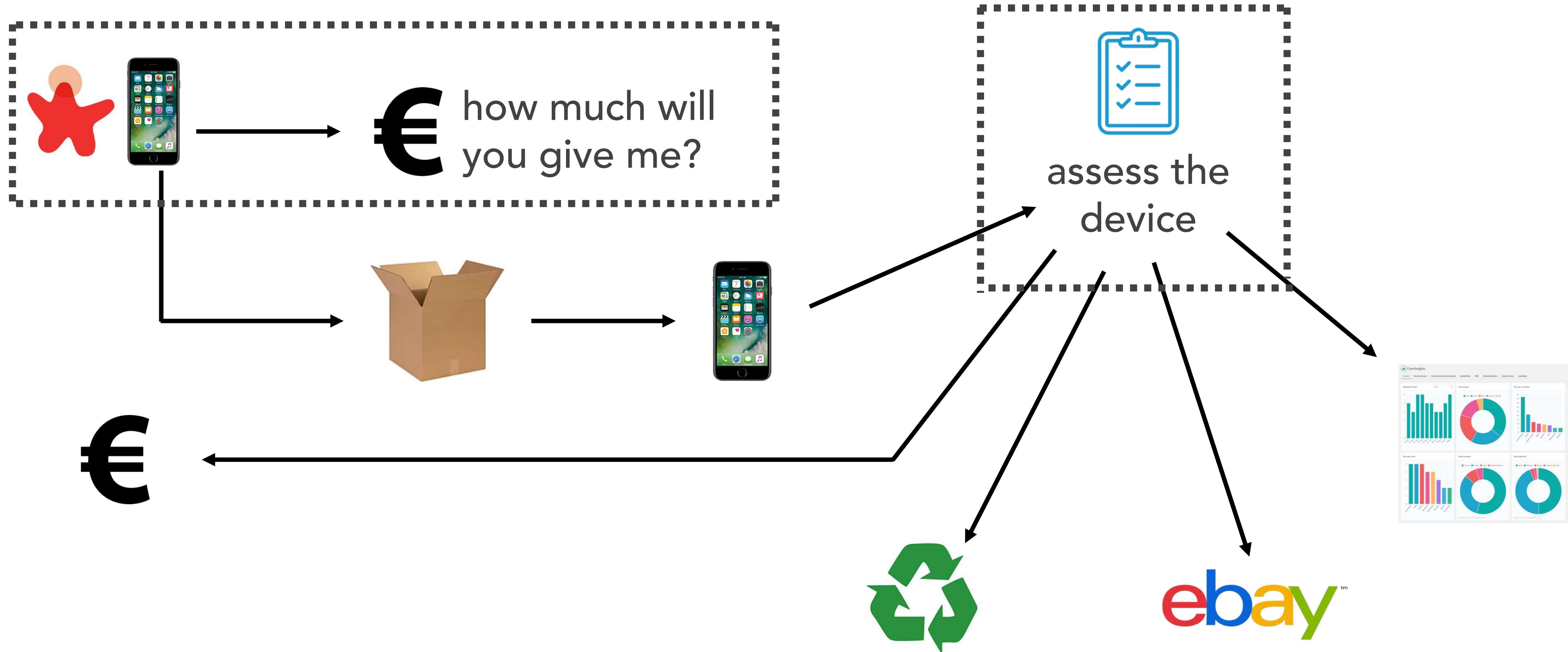
architectural quantum

electronics recycling



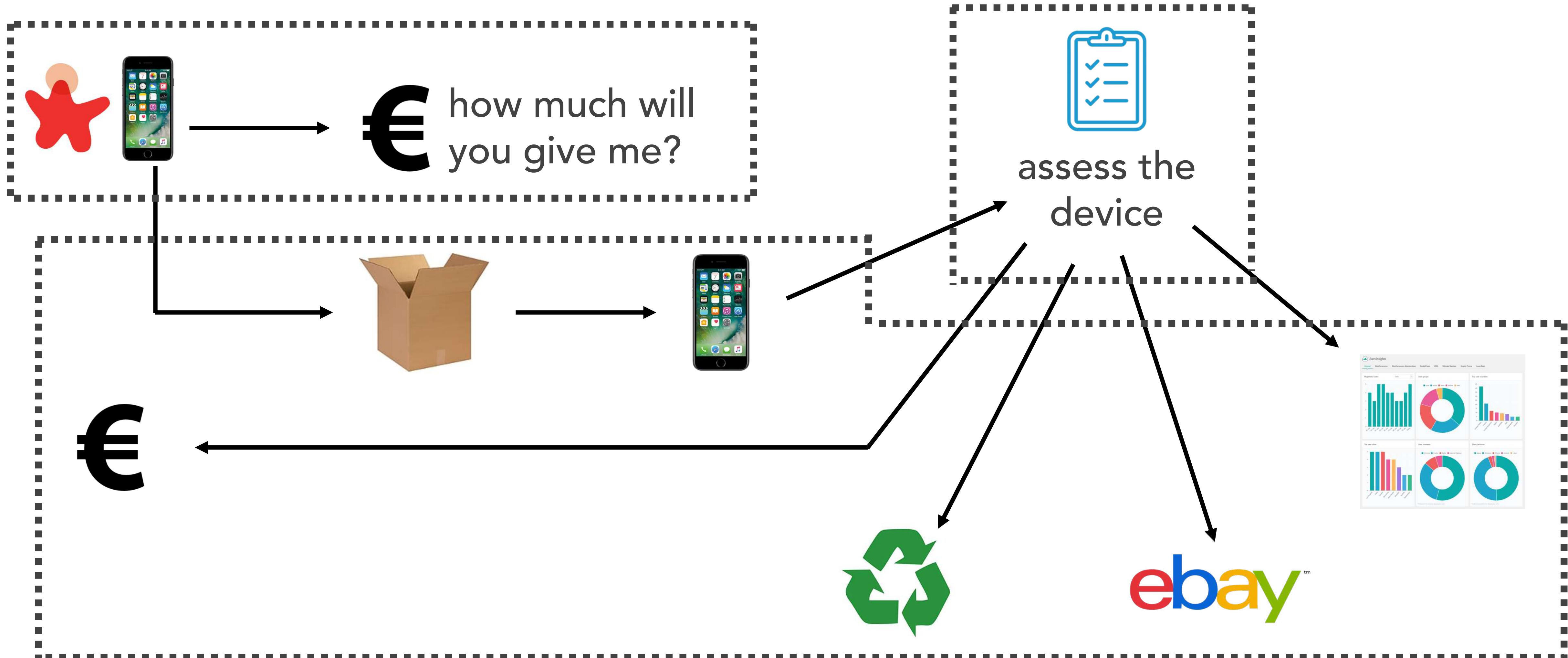
architectural quantum

electronics recycling



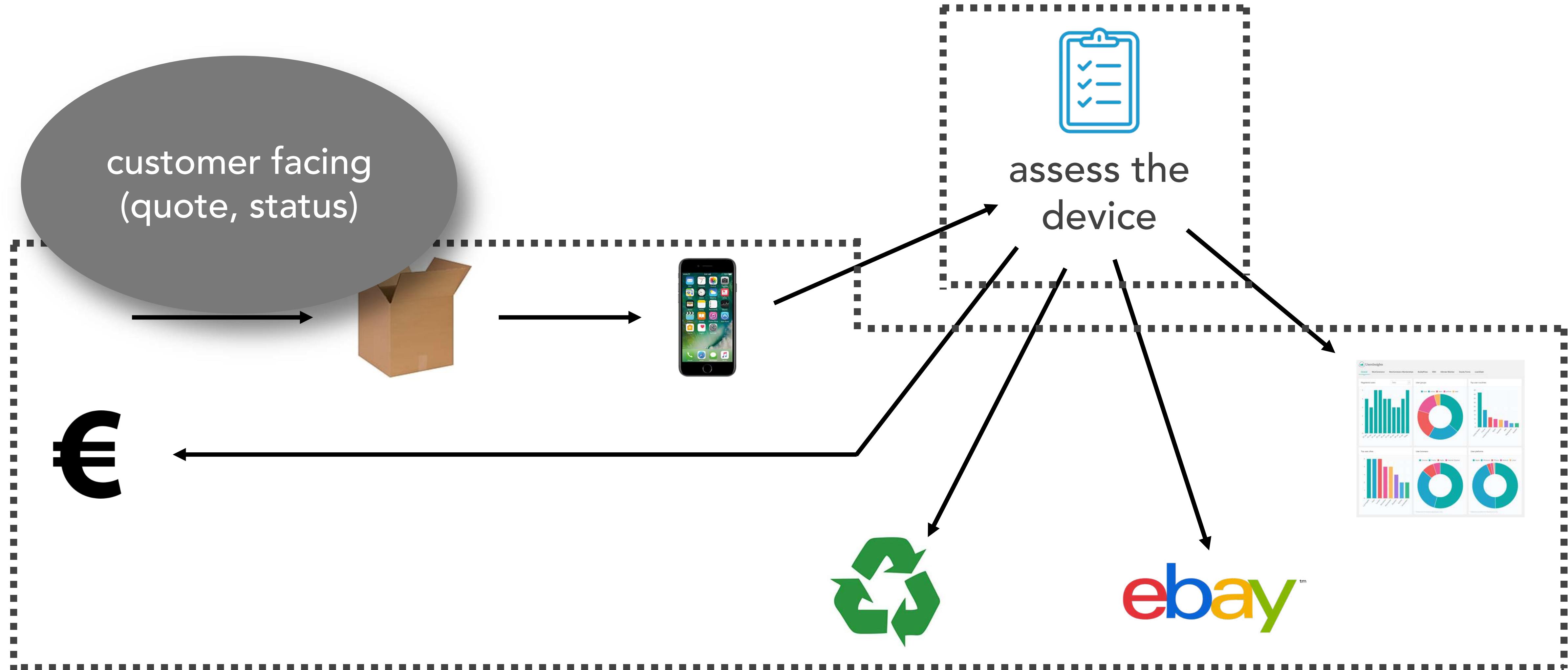
architectural quantum

electronics recycling



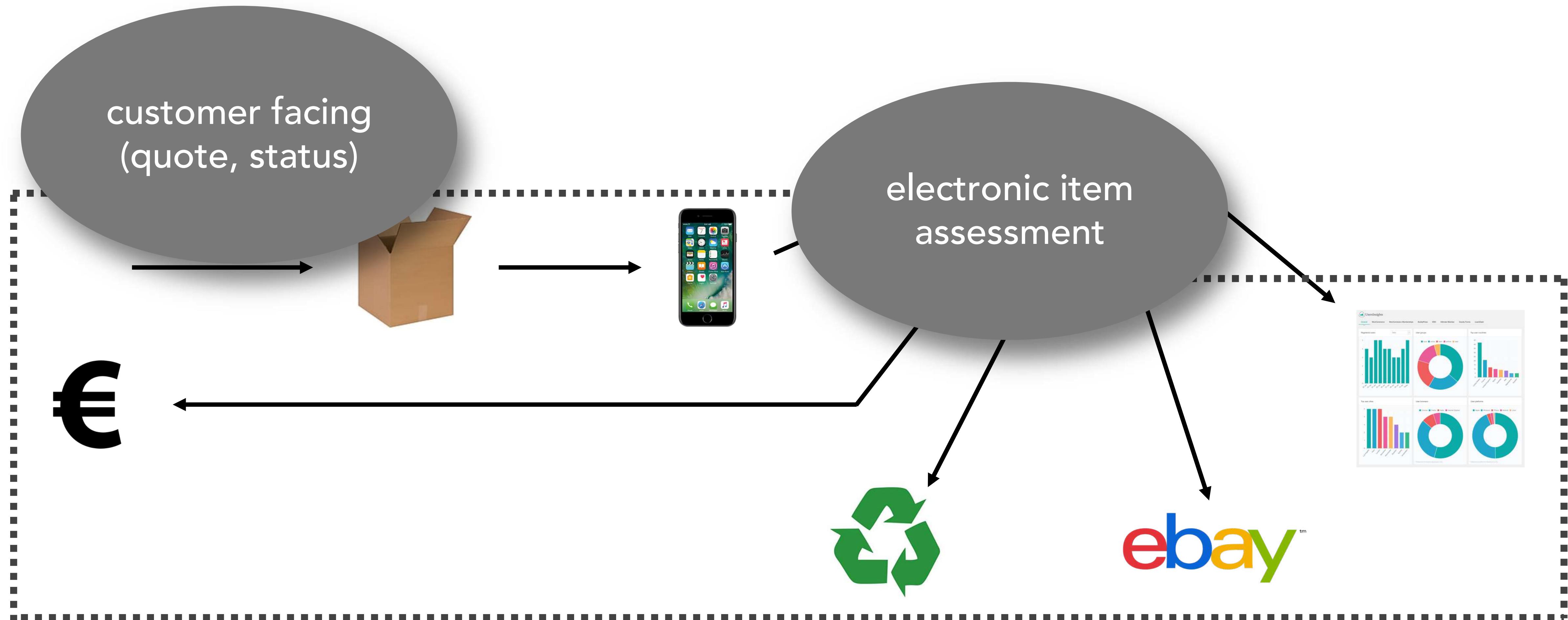
architectural quantum

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

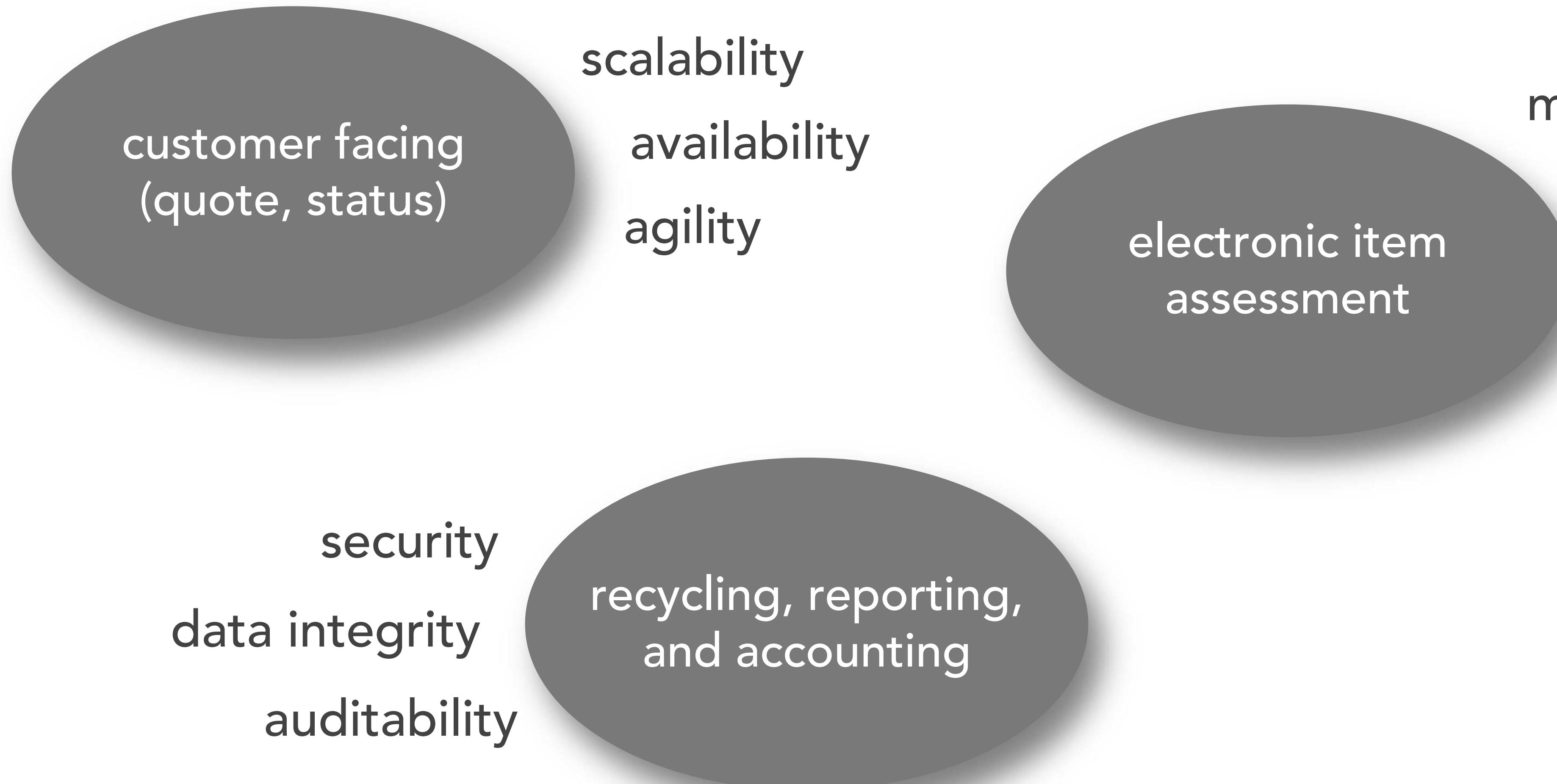
electronic item
assessment

maintainability
deployability
testability
agility

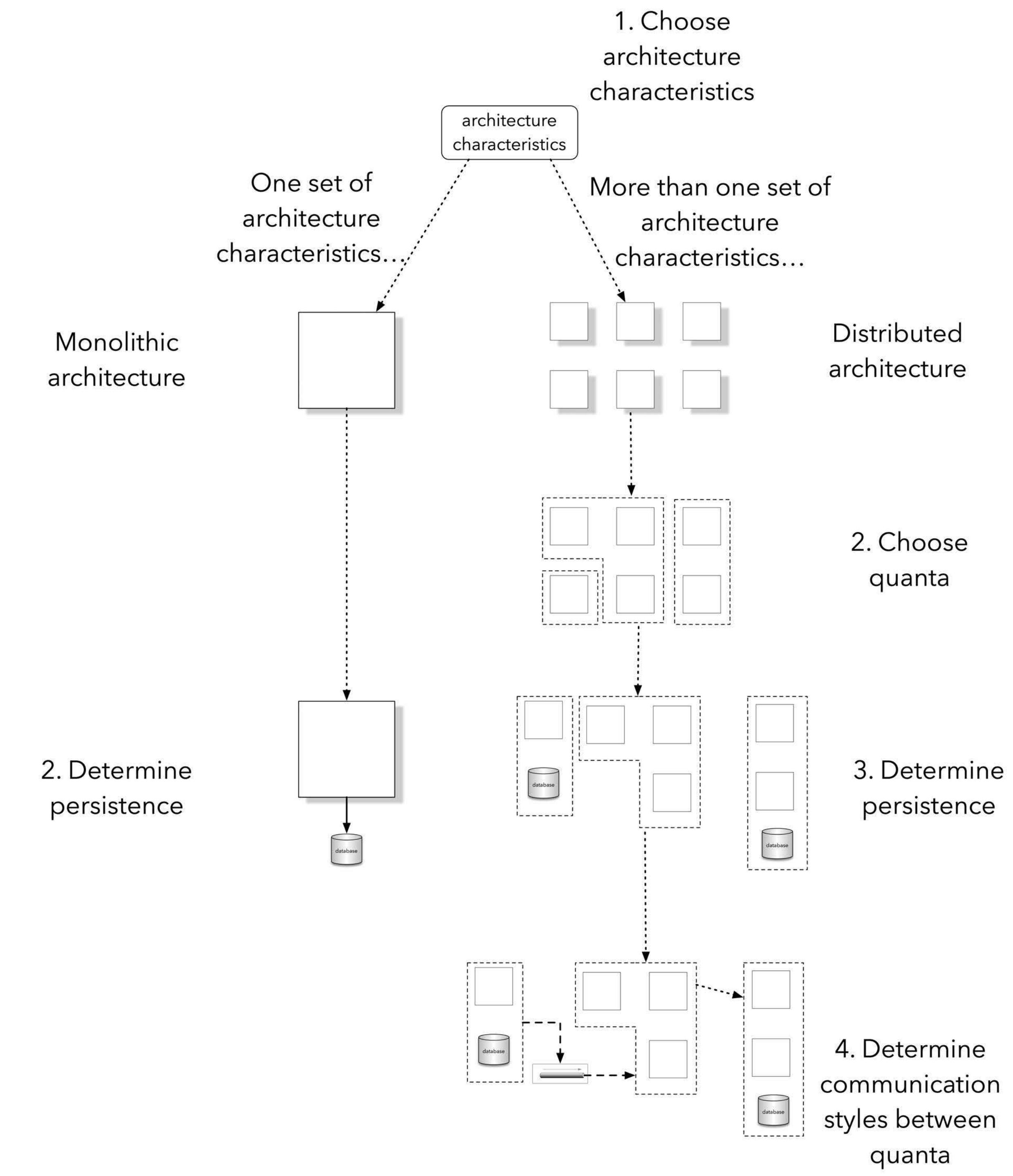
recycling, reporting,
and accounting

architectural quantum

electronics recycling



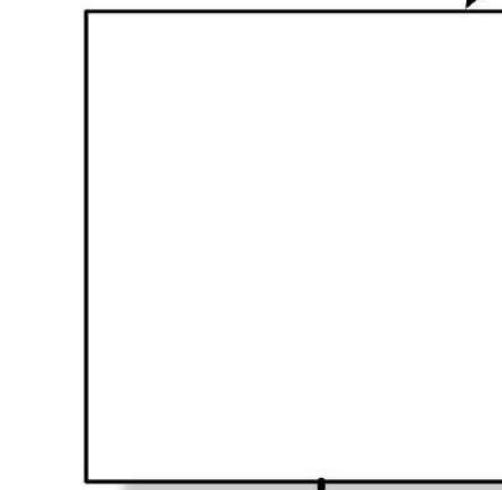
choosing an architecture



choosing an architecture

Monolithic
architecture

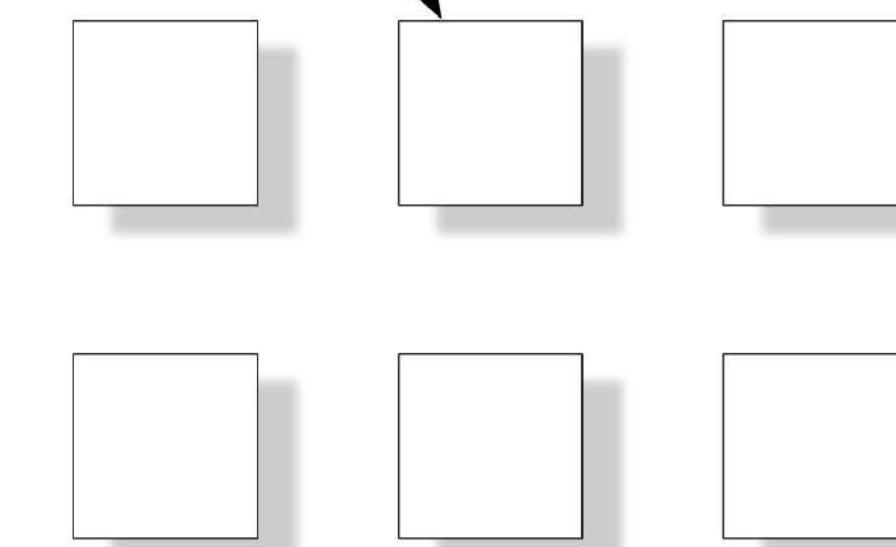
One set of
architecture
characteristics...



architecture
characteristics

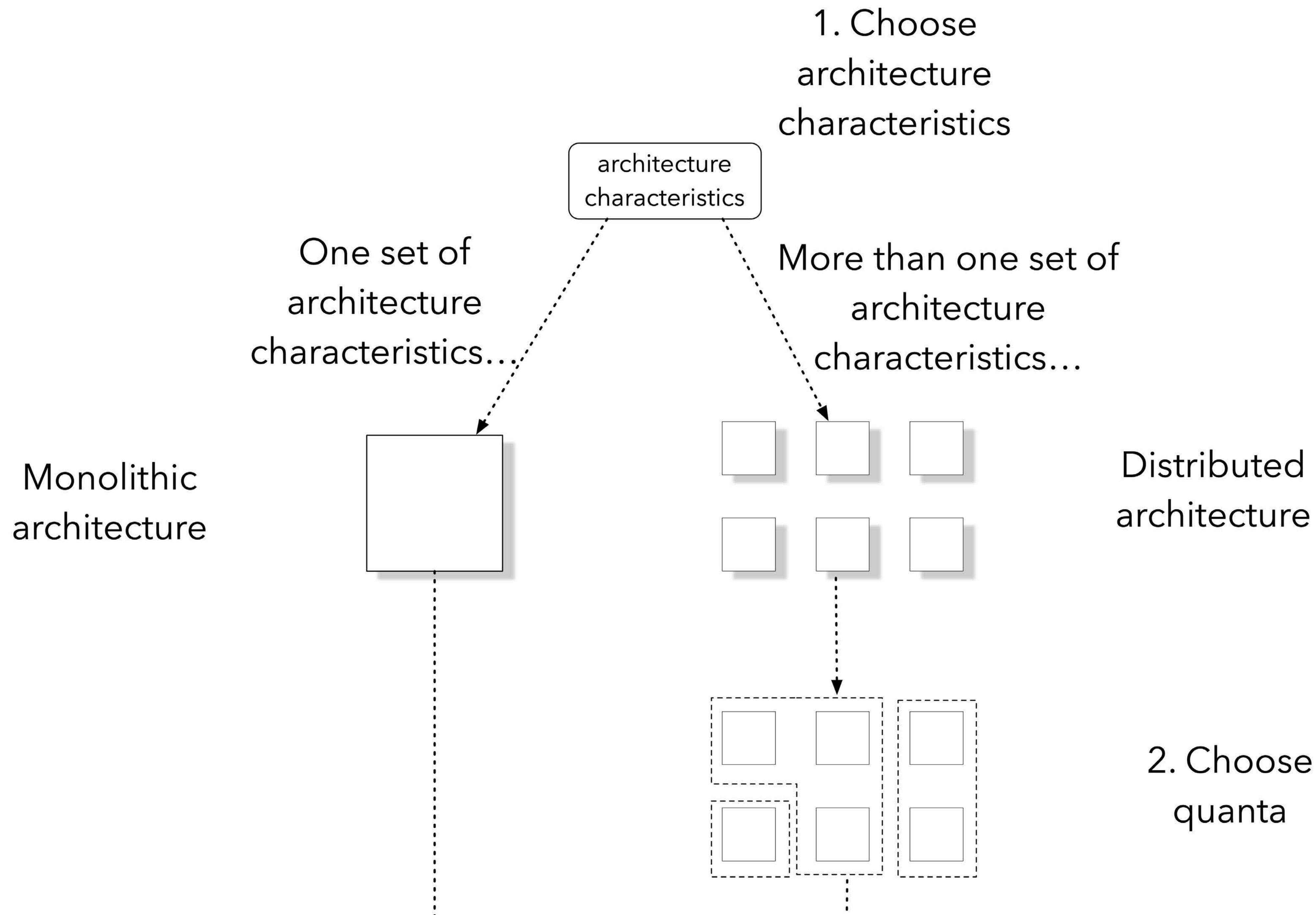
1. Choose
architecture
characteristics

More than one set of
architecture
characteristics...



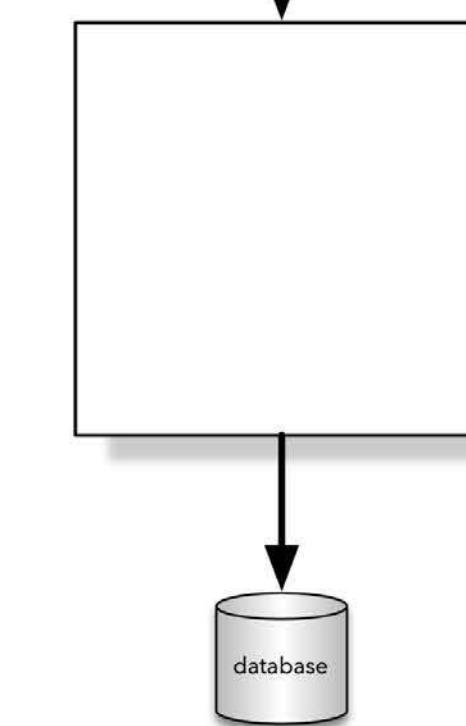
Distributed
architecture

choosing an architecture

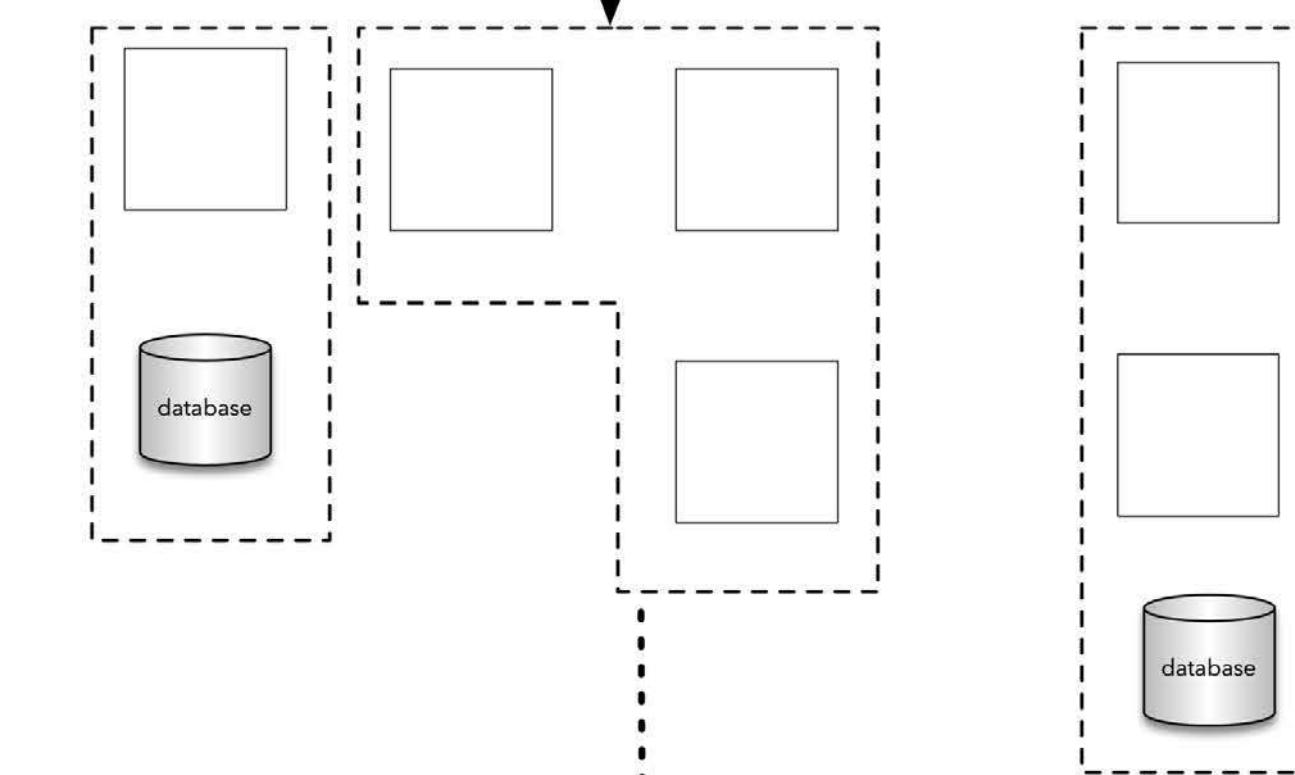


choosing an architecture

2. Determine persistence



Monolithic architecture



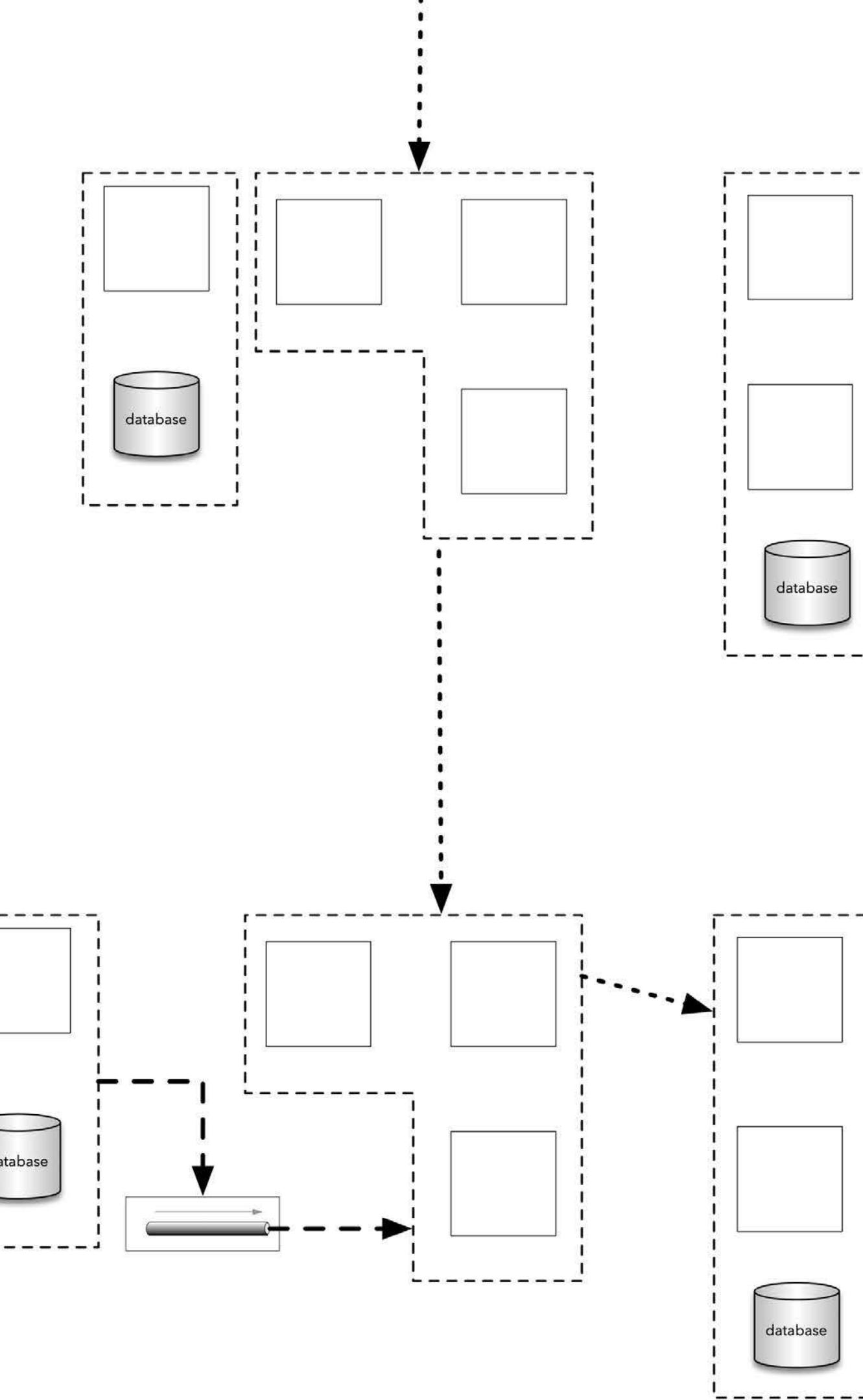
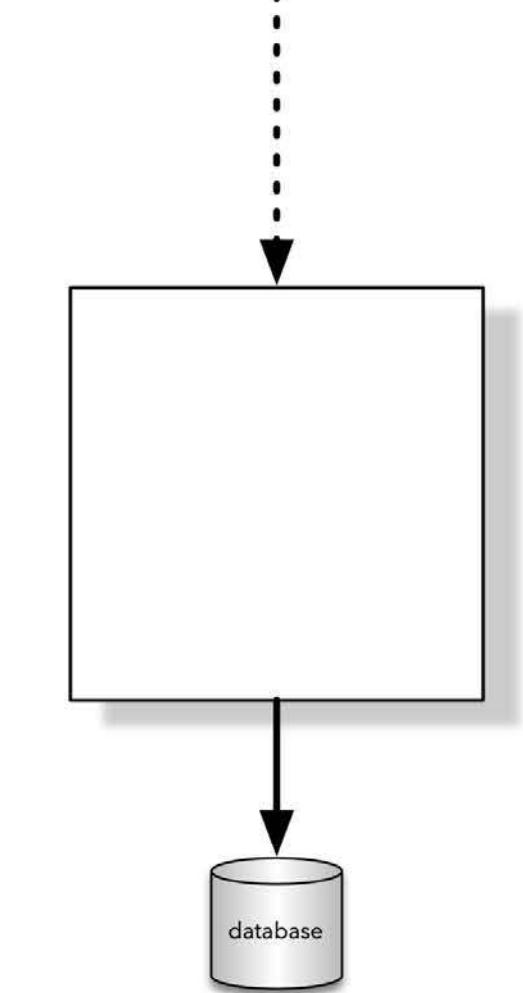
3. Determine persistence

Distributed architecture

2. Choose quanta

choosing an architecture

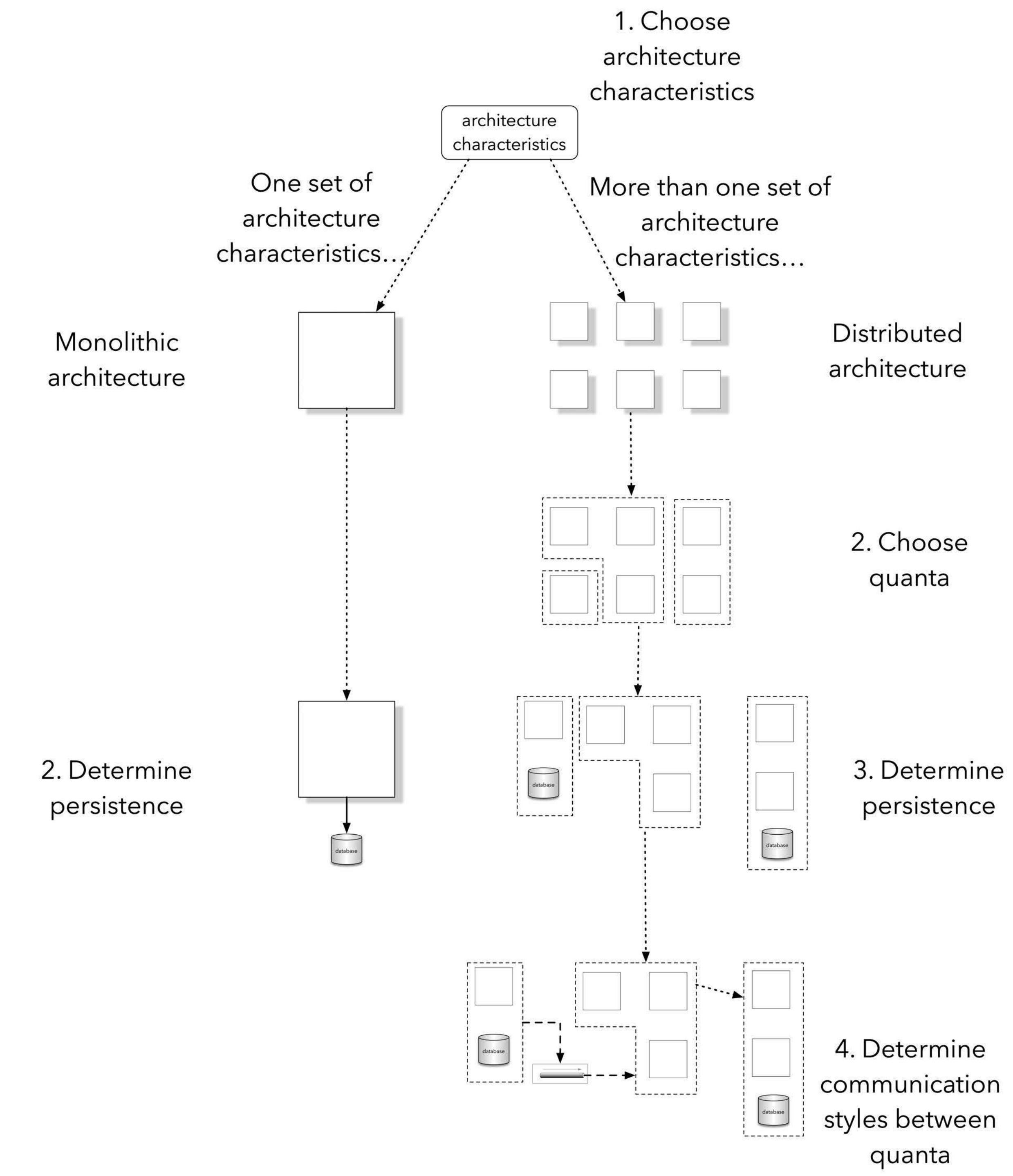
2. Determine persistence



3. Determine persistence

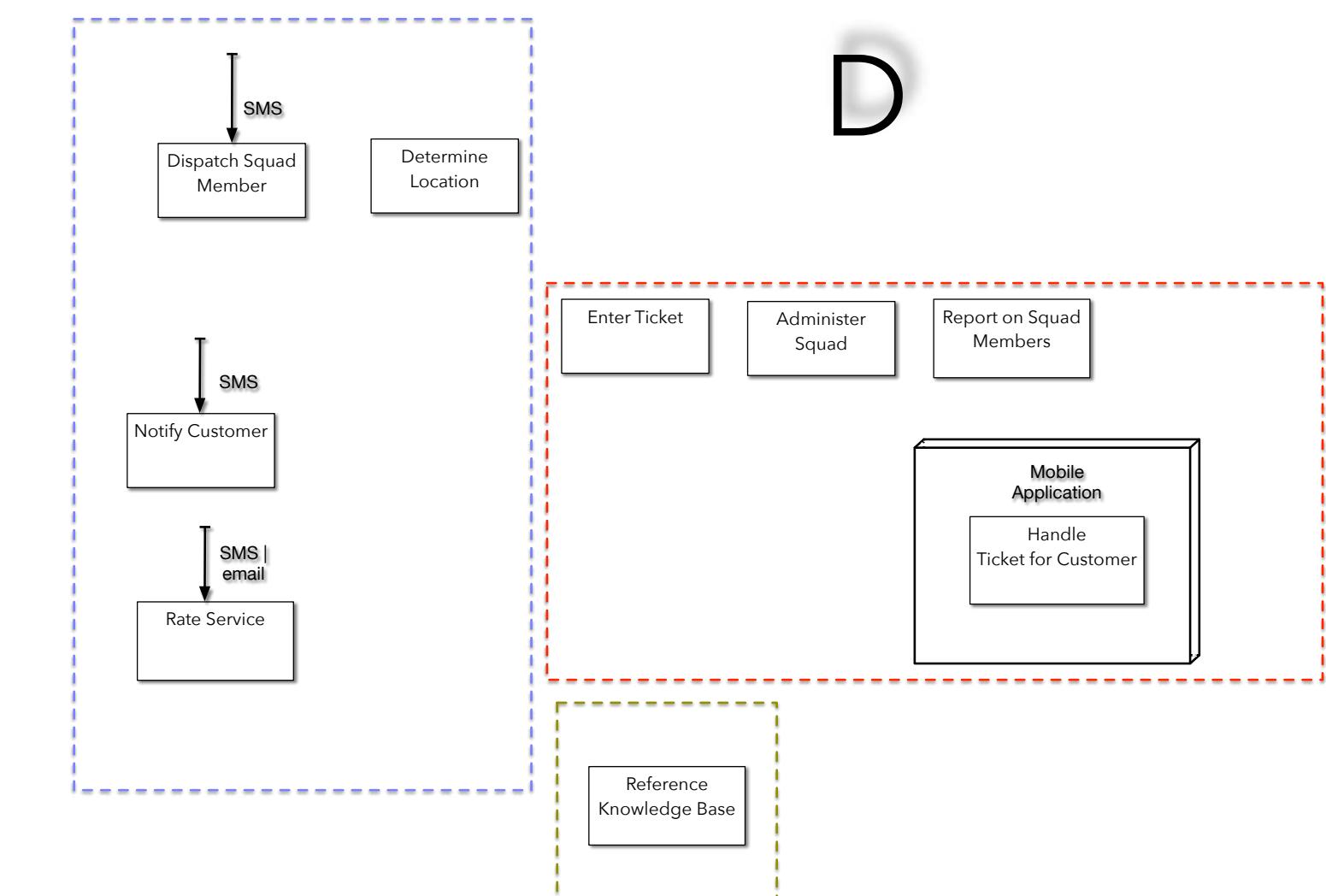
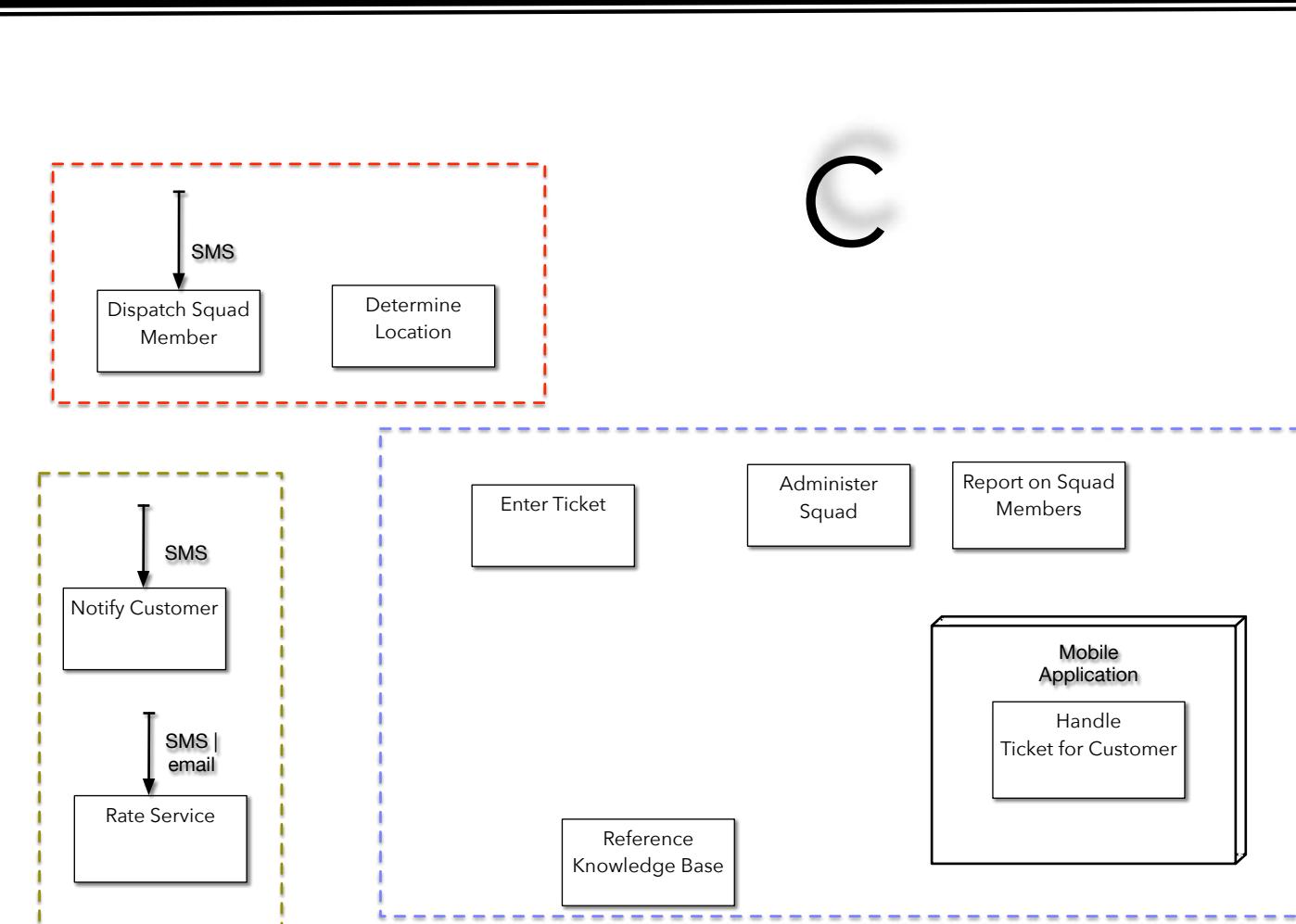
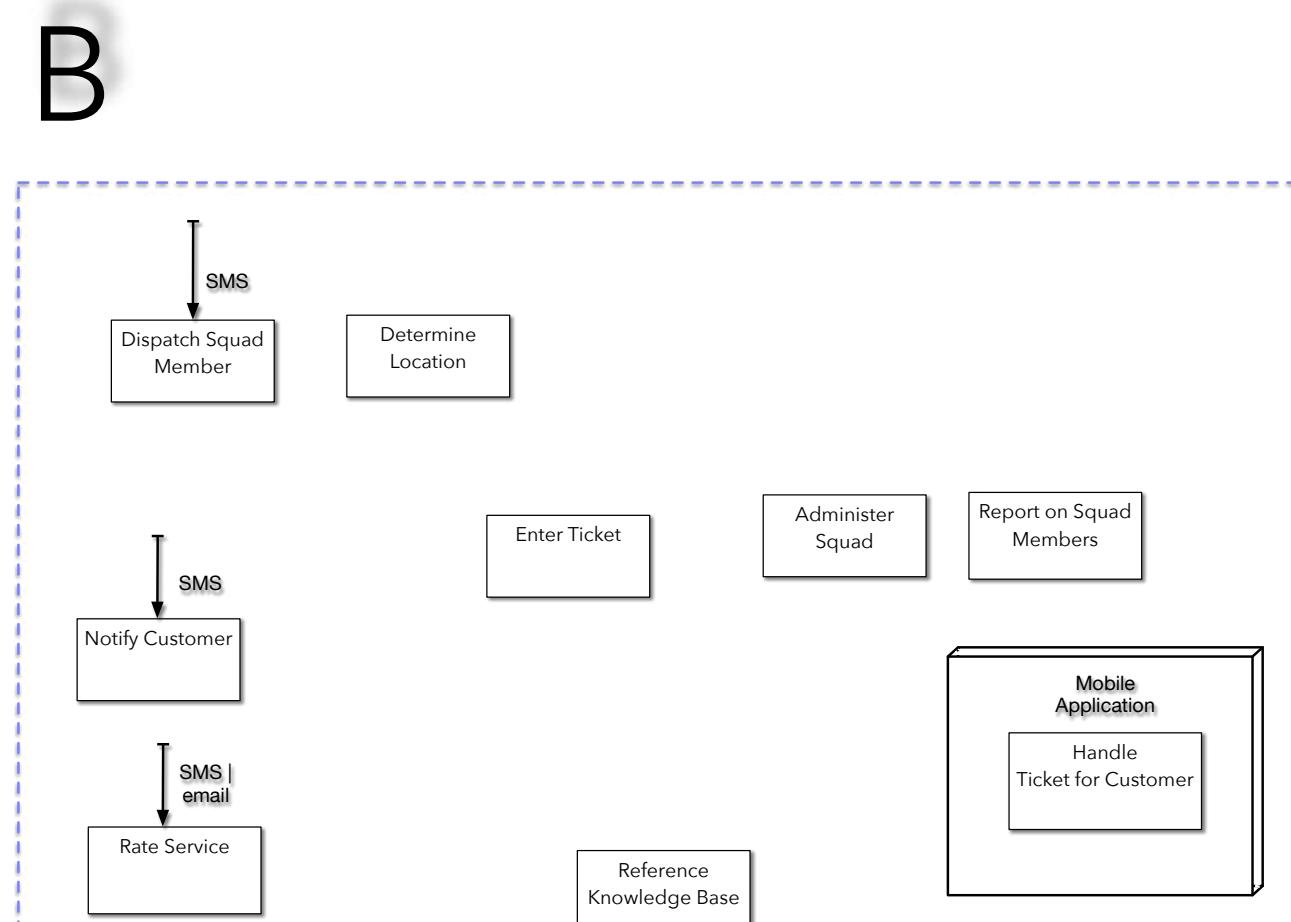
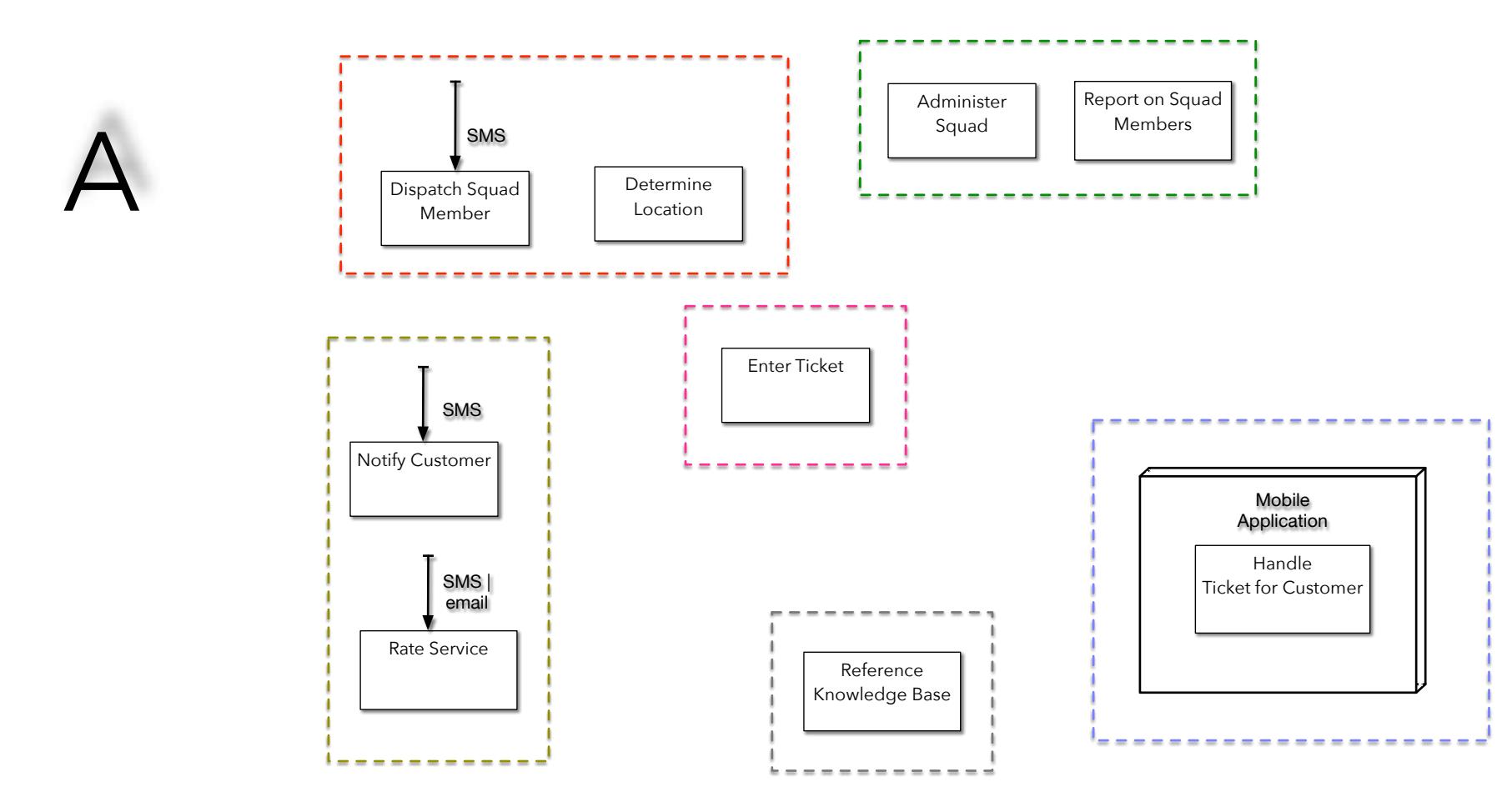
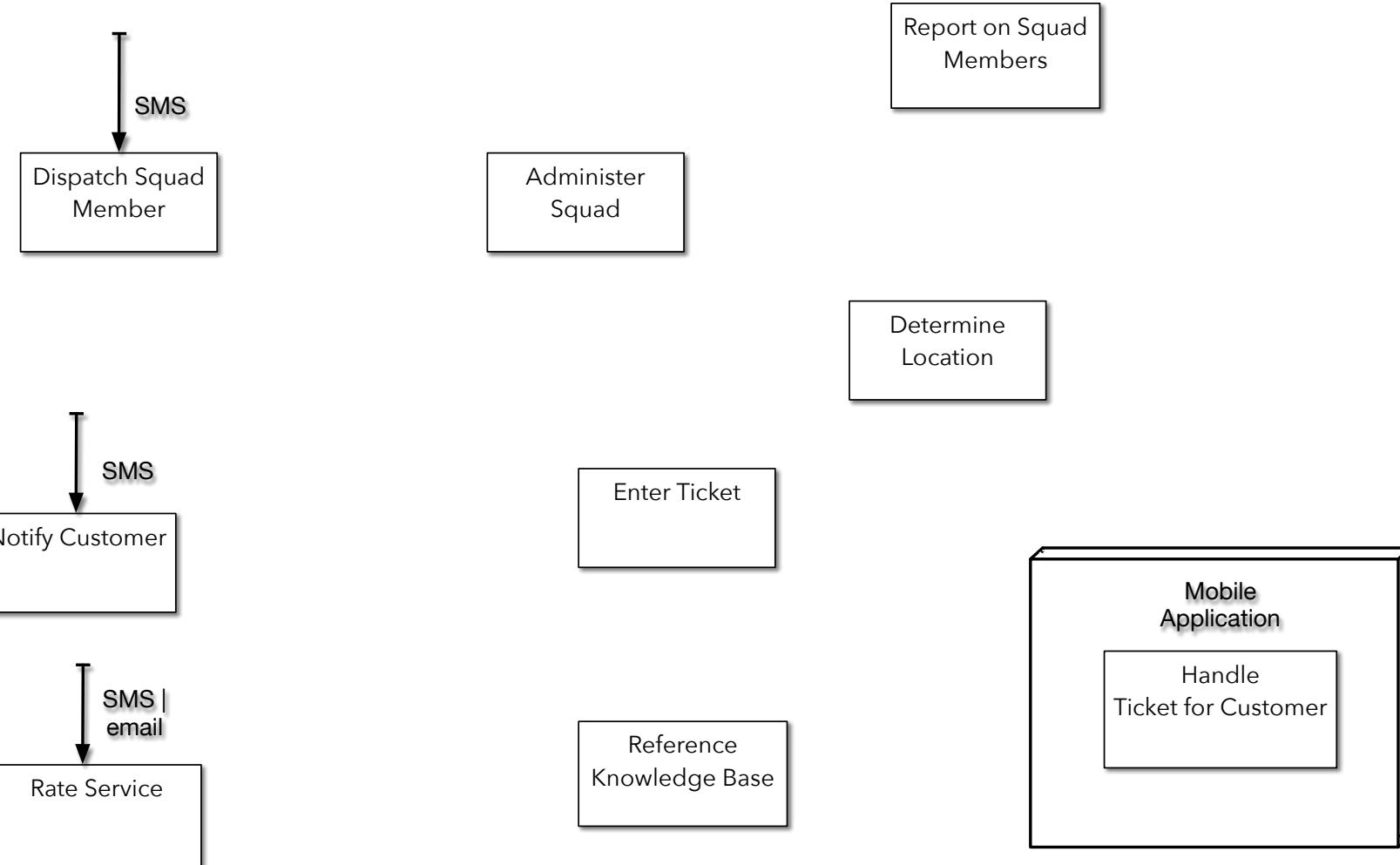
4. Determine communication styles between quanta

choosing an architecture



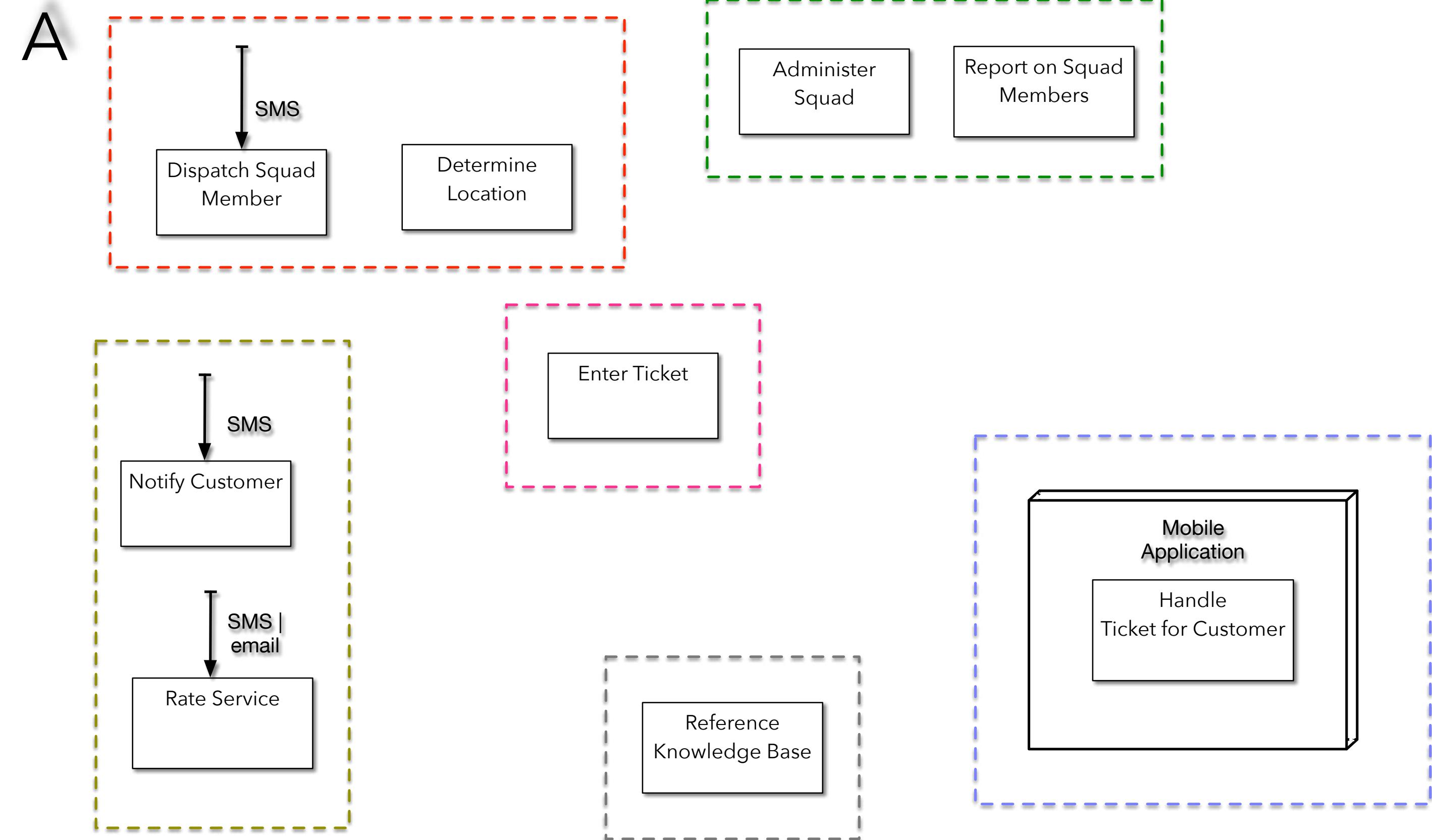
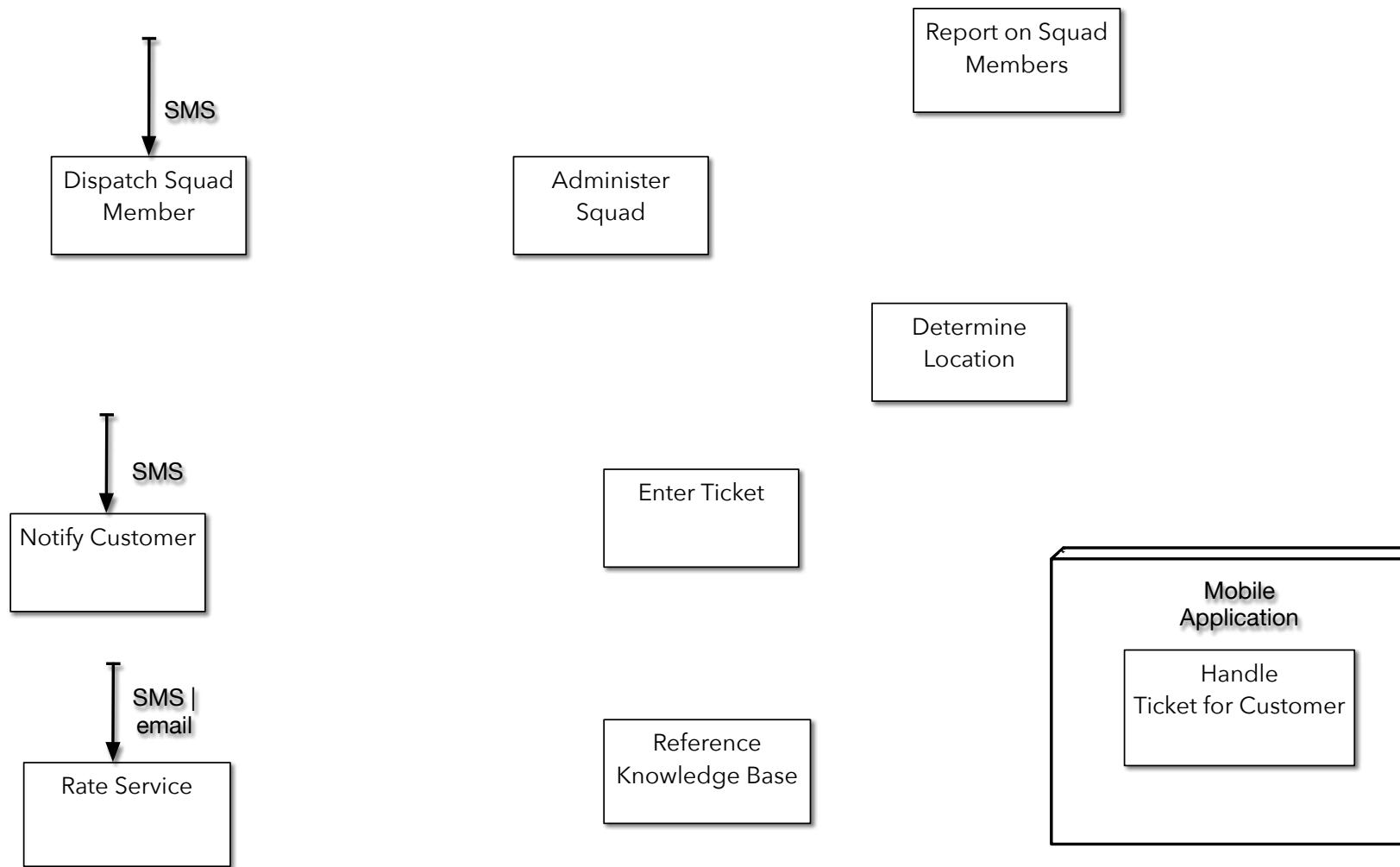
Scenario Exercise - Architecture Quanta

Given the set of components identified below, where should the quantum boundaries lie?



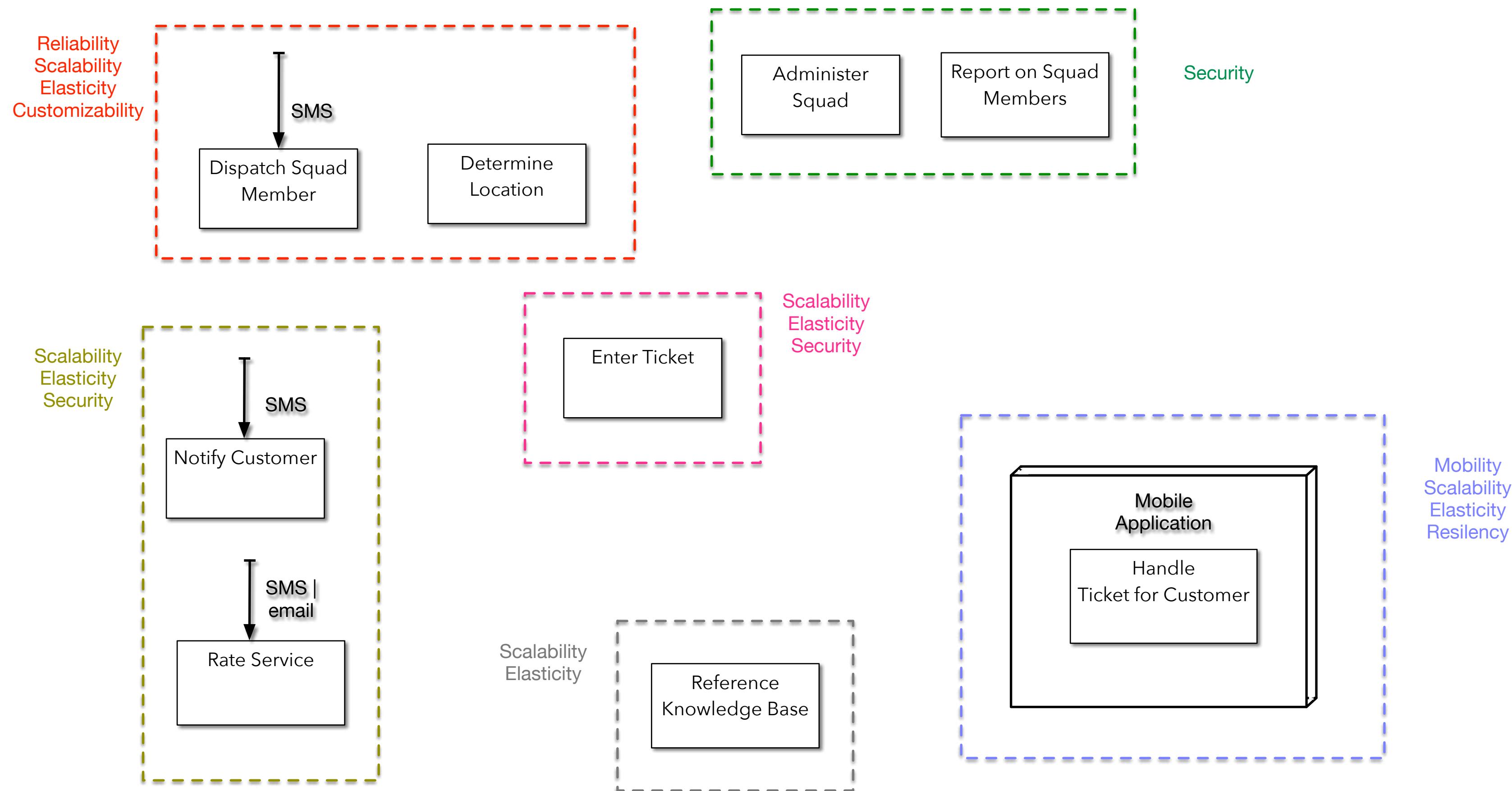
Scenario Exercise - Architecture Quanta(Instructor)

Given the set of components identified below, where should the quantum boundaries lie?



Scenario Exercise - Architecture Quanta(Instructor)

Given the set of components identified below, where should the quantum boundaries lie?



Scenario Exercise – Choosing Monolithic versus Distributed Architecture

What style of architecture would be most appropriate for each set of quanta?

A monolithic

distributed

B monolithic

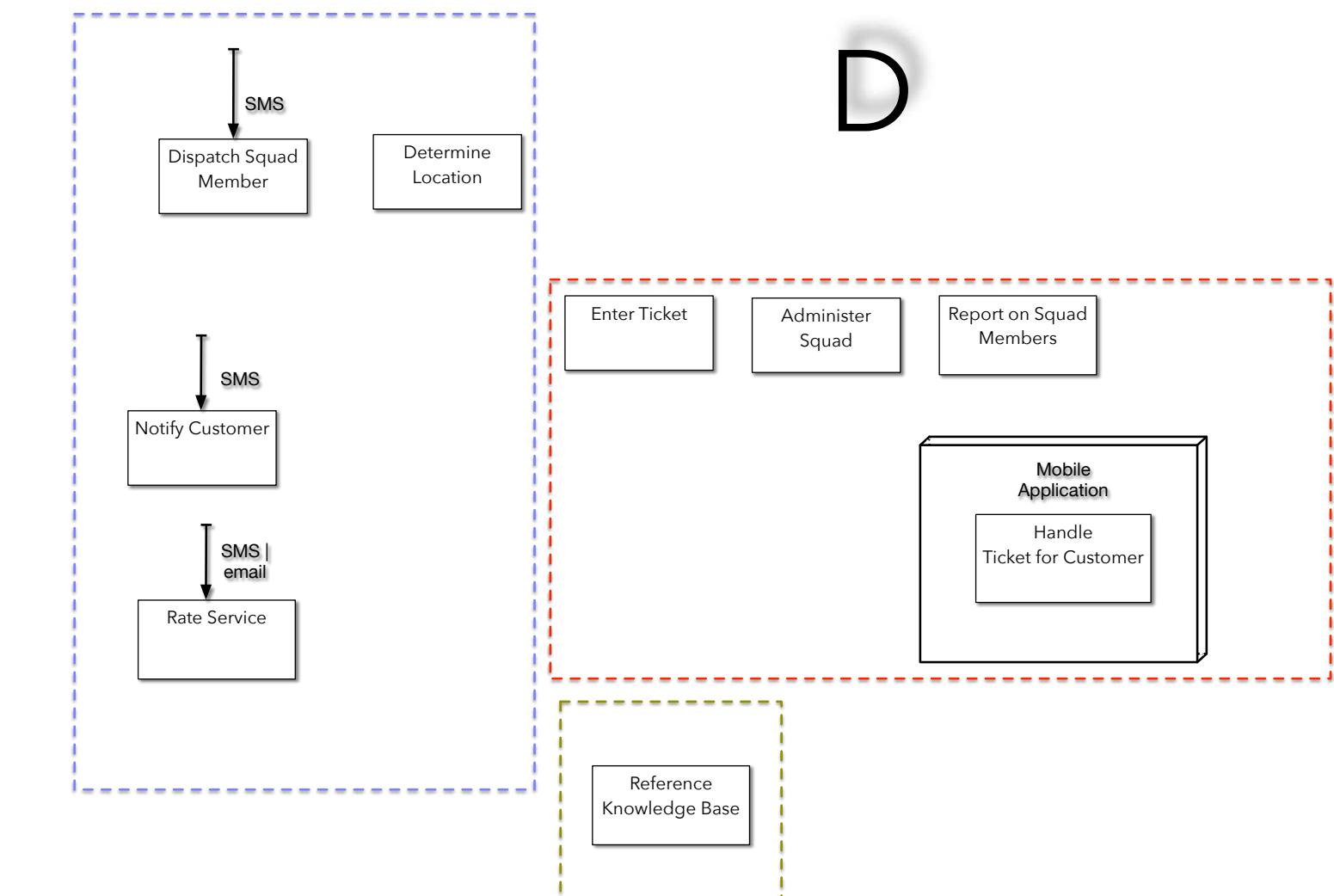
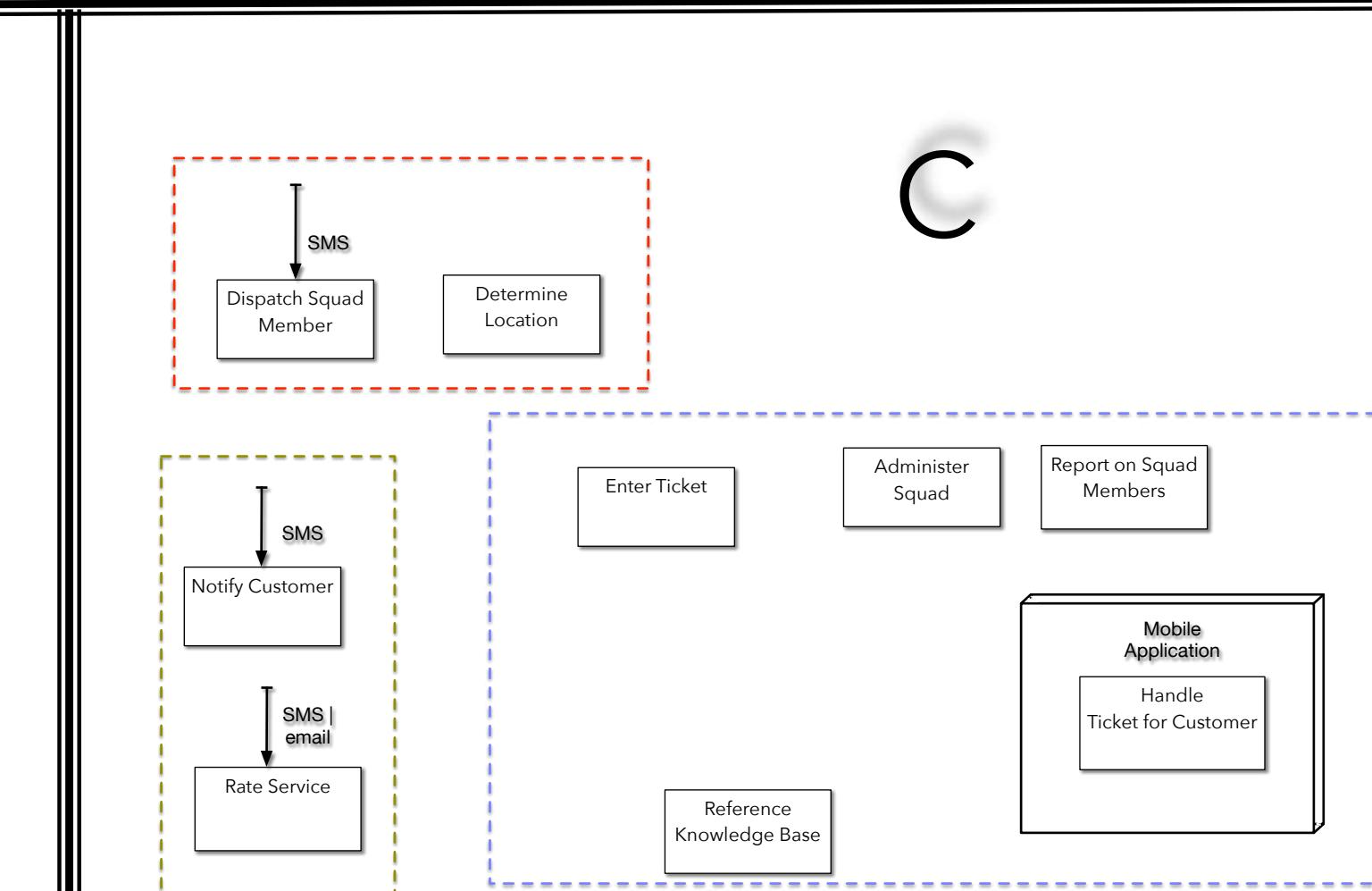
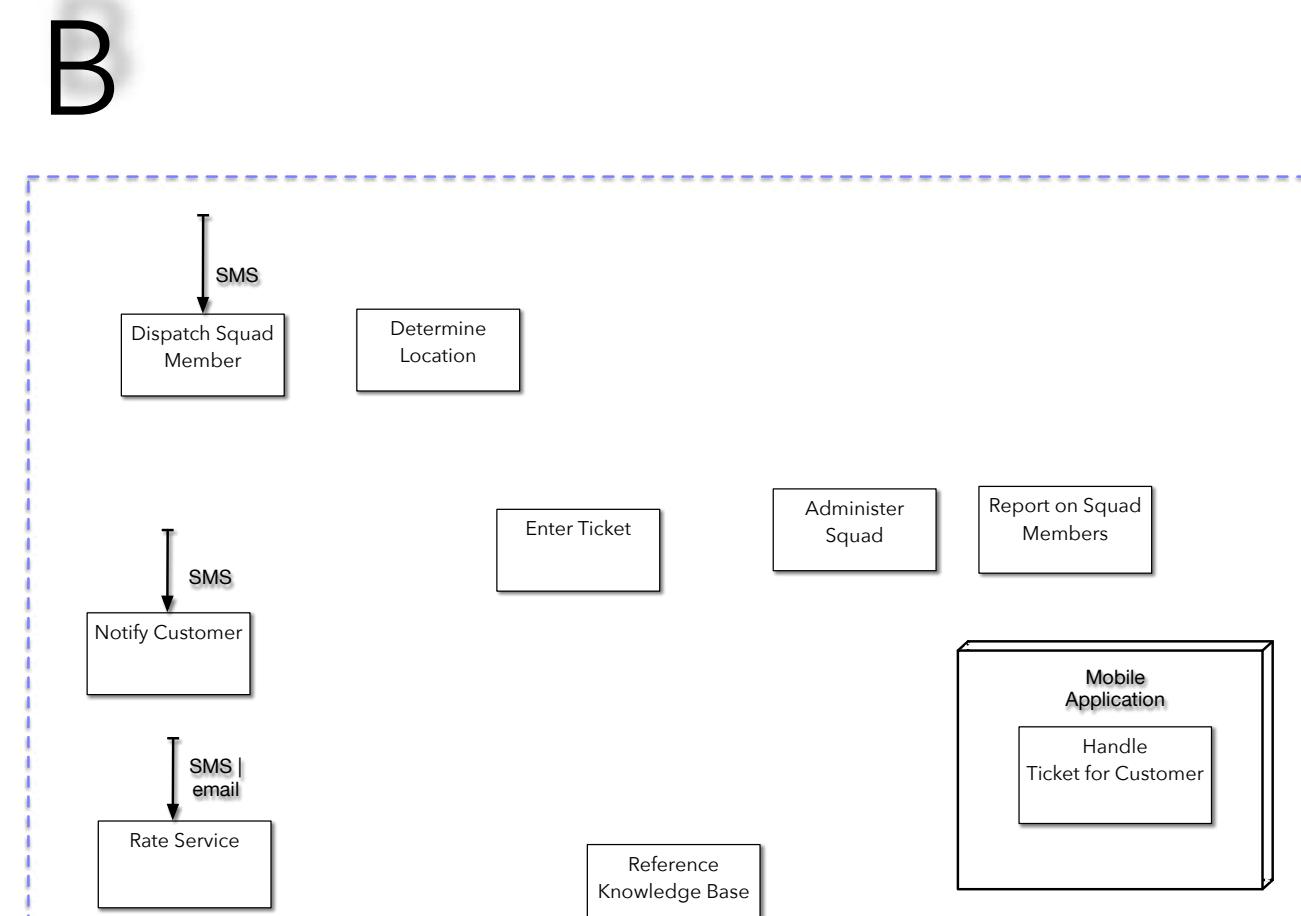
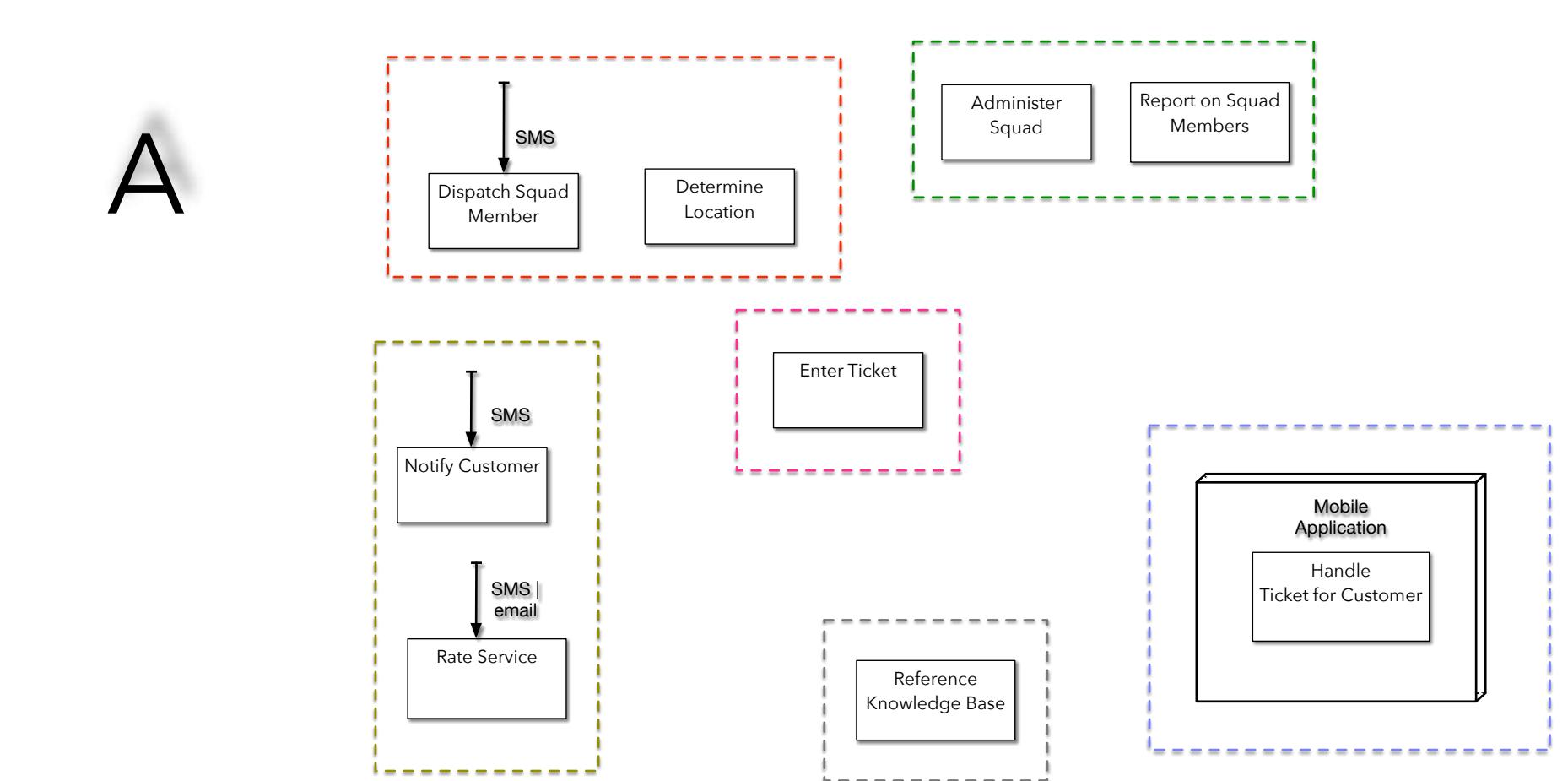
distributed

C monolithic

distributed

D monolithic

distributed



Scenario Exercise – Choosing Monolithic versus Distributed Architecture

What style of architecture would be most appropriate for each set of quanta?

A monolithic

B monolithic

C monolithic

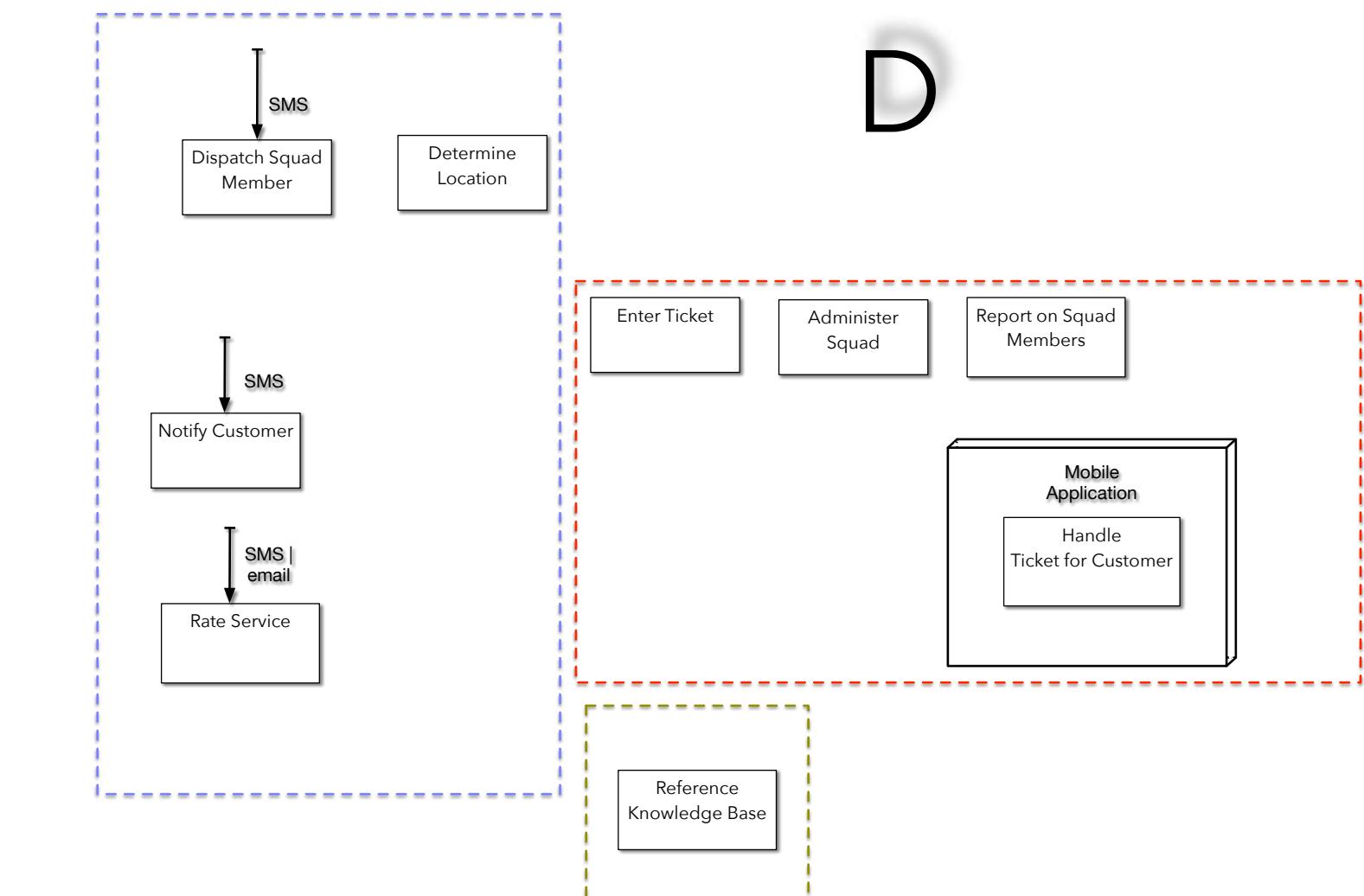
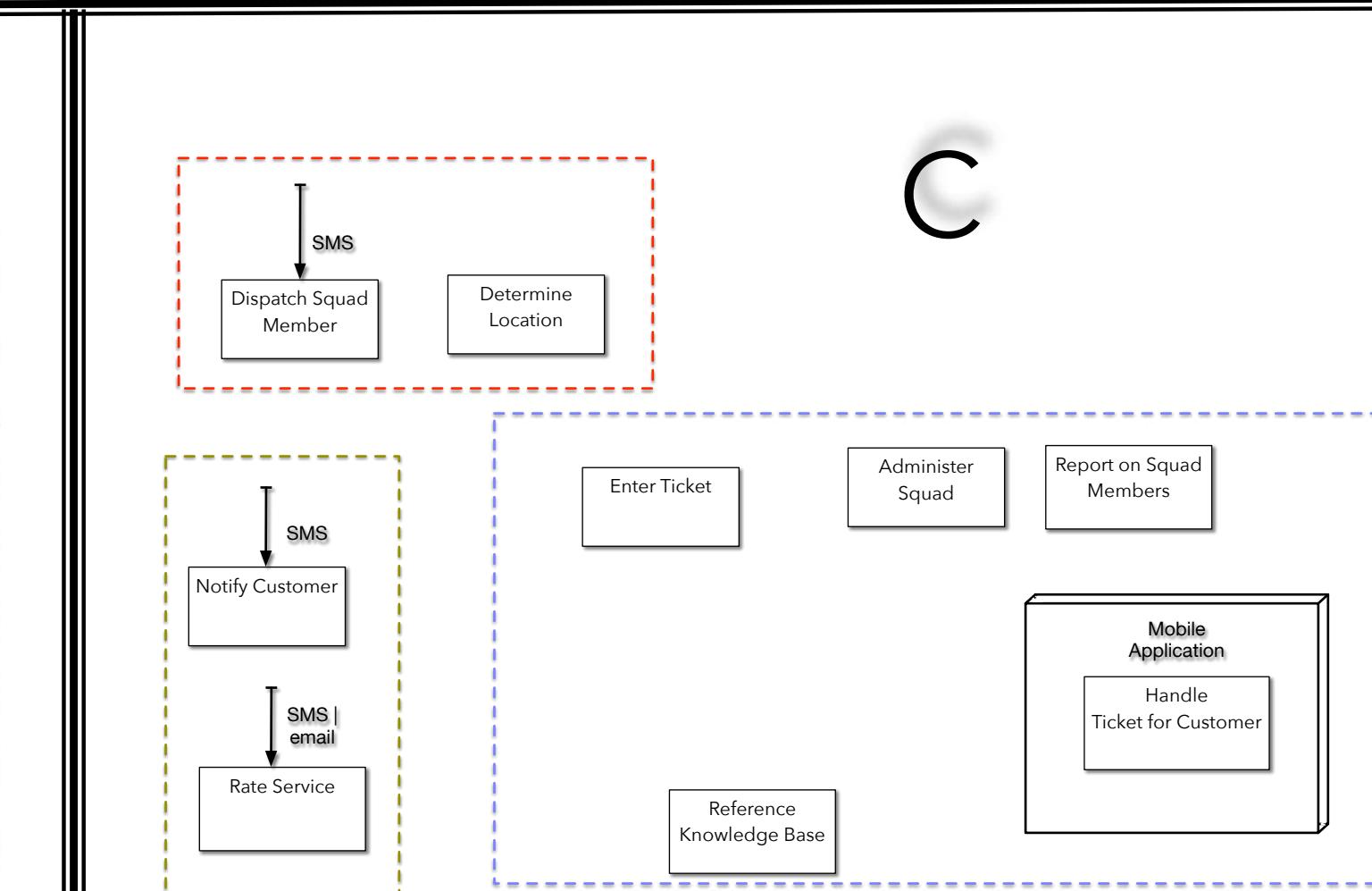
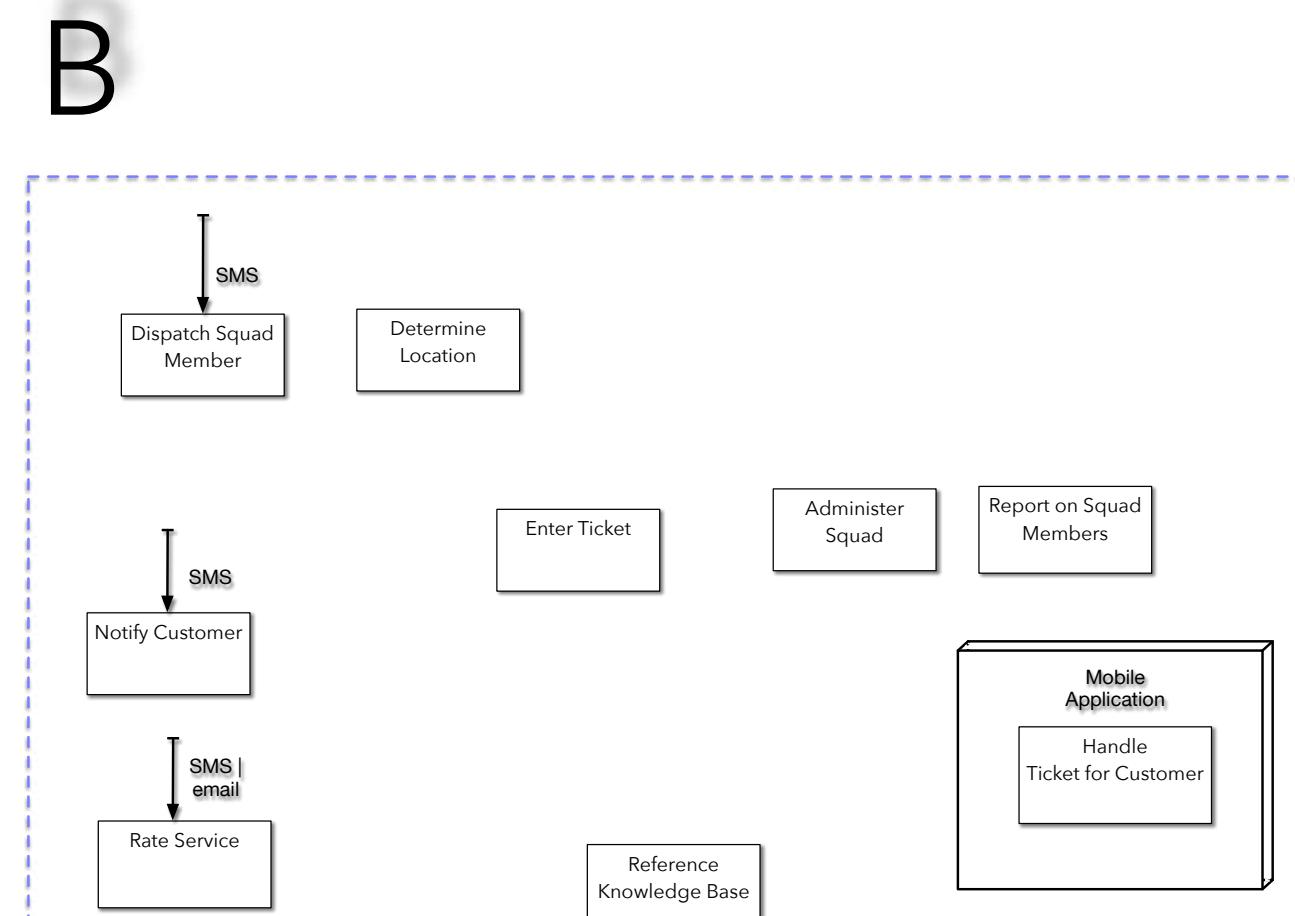
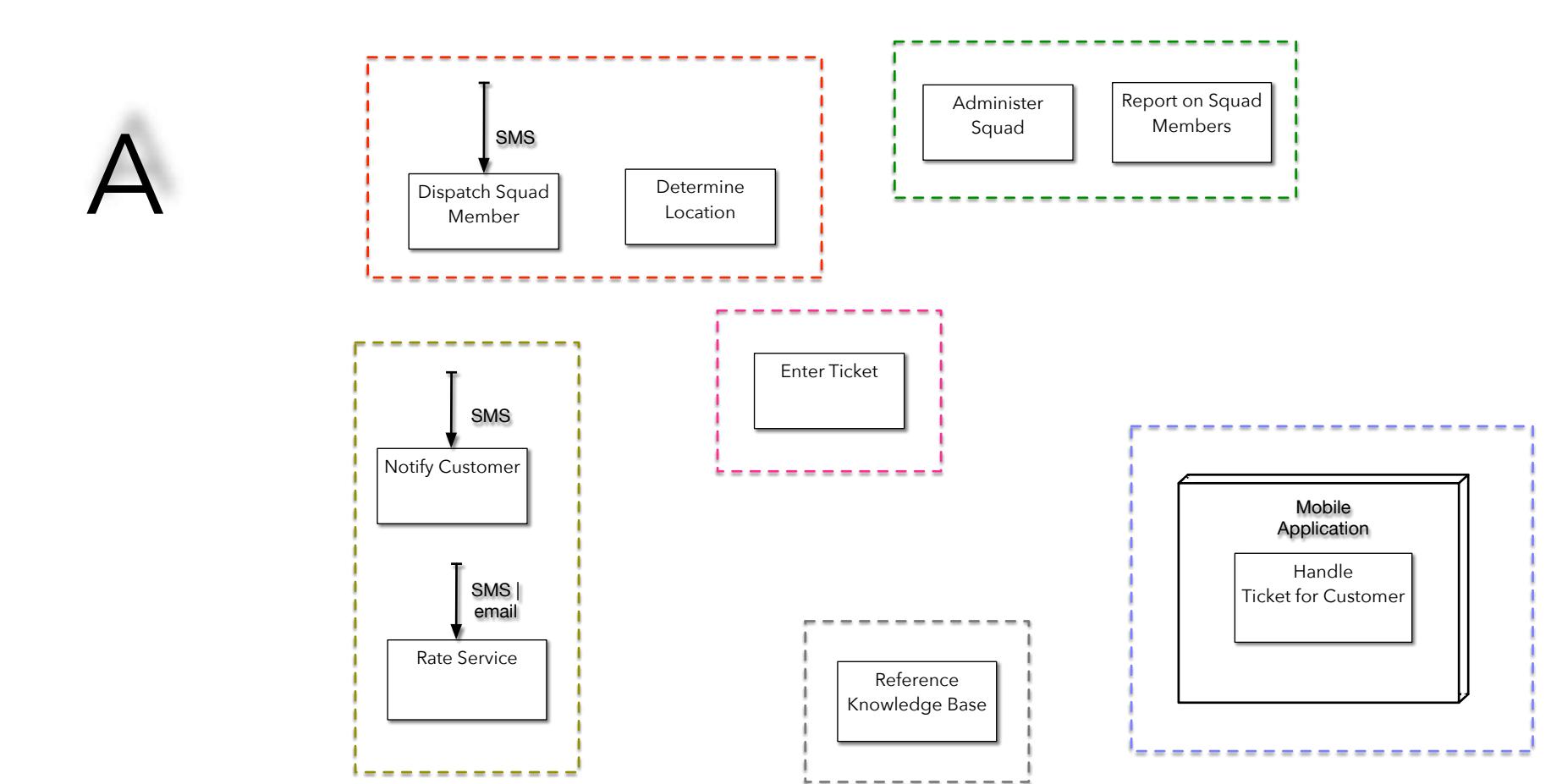
D monolithic

distributed

distributed

distributed

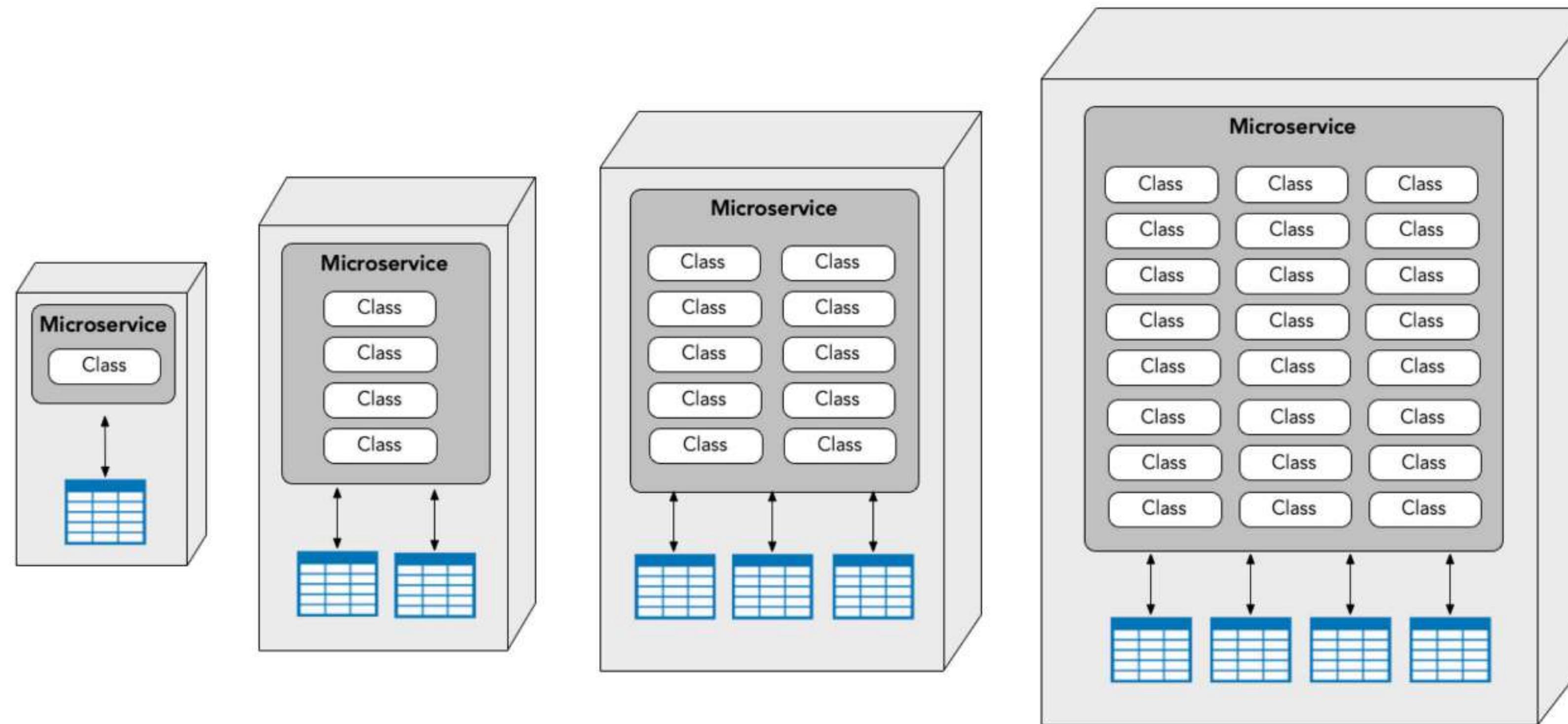
distributed



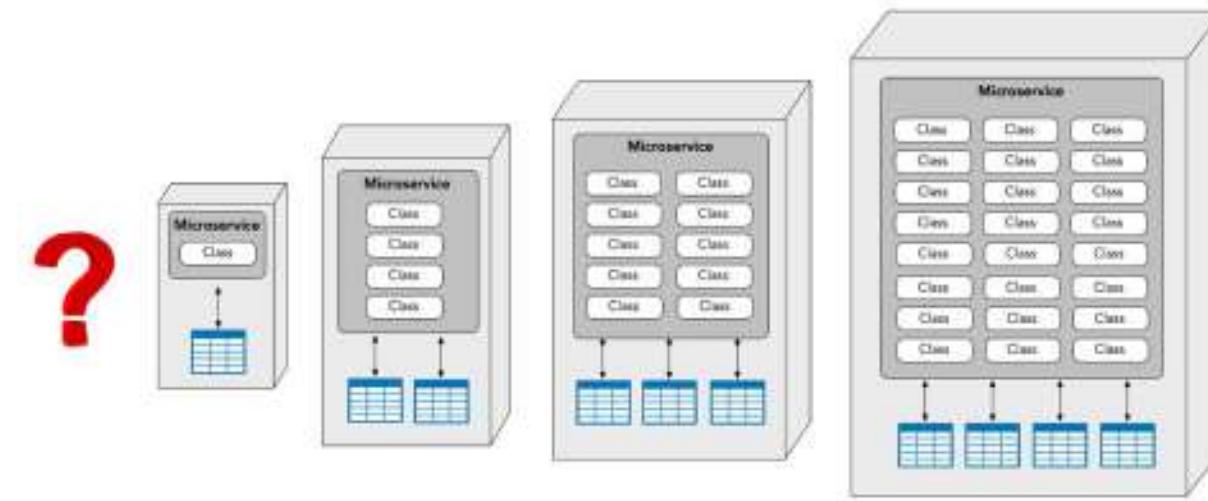


How do I determine the appropriate level of granularity?

what is the right level of granularity for a service?



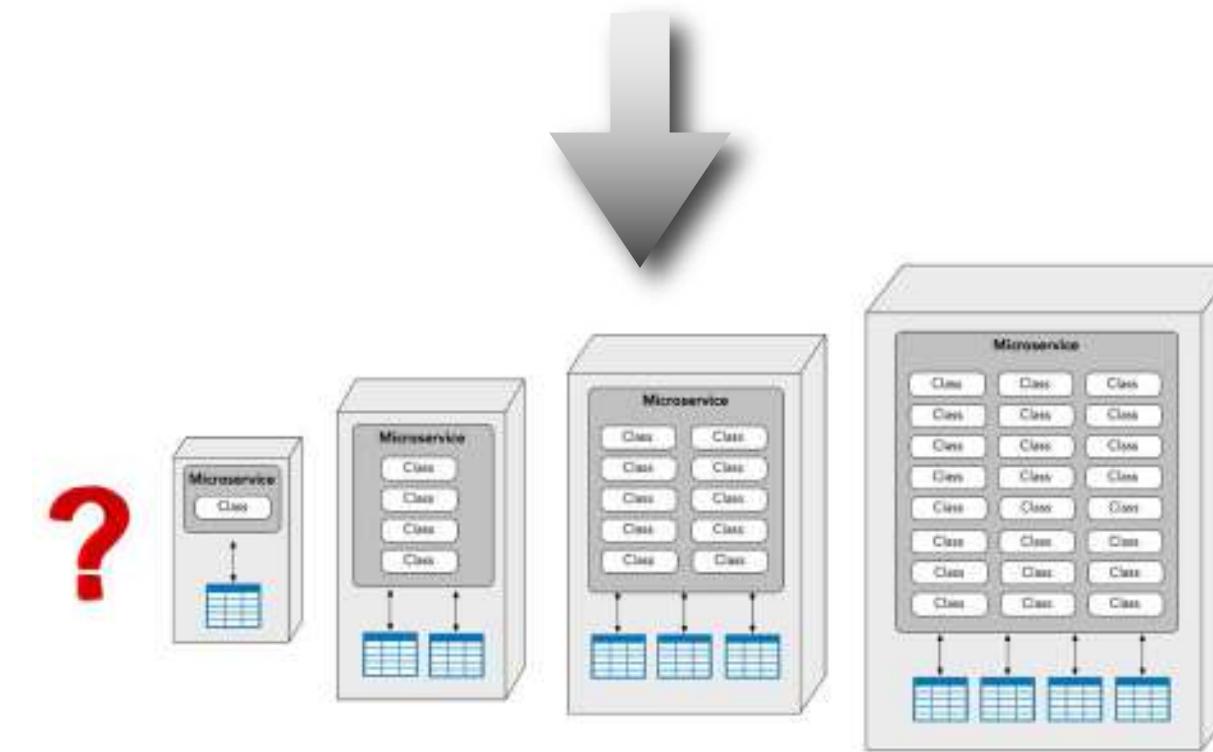
service granularity



service granularity

granularity drivers

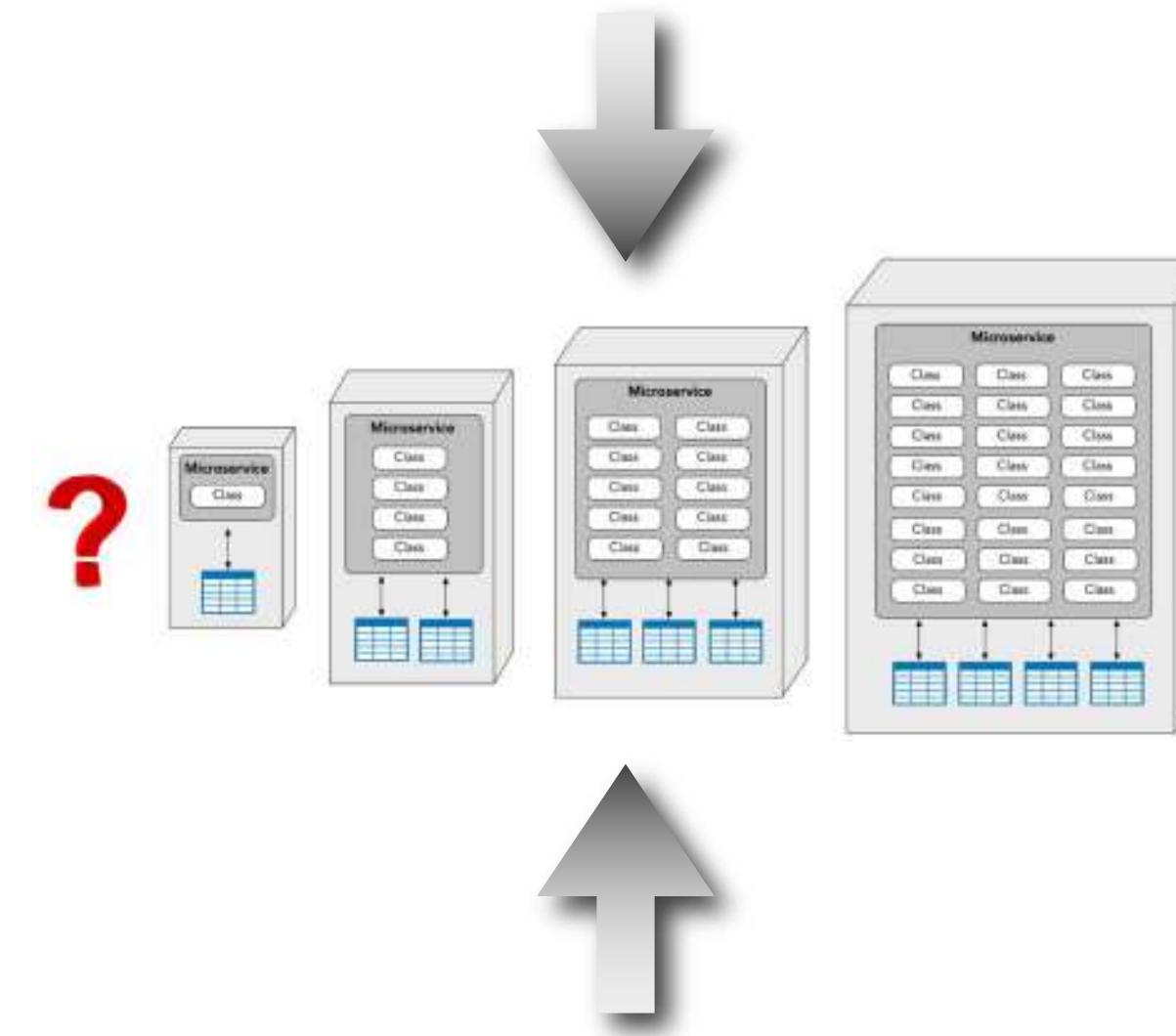
“when should I consider breaking apart a service?”



service granularity

granularity drivers

“when should I consider breaking apart a service?”



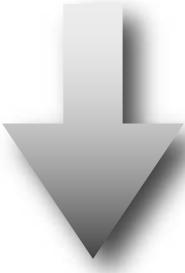
granularity factors

“what factors influence service granularity?”

service granularity

granularity drivers

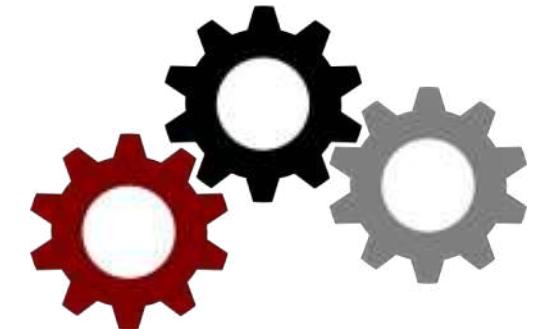
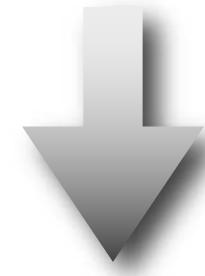
“when should I consider breaking apart a service?”



service granularity

granularity drivers

“when should I consider breaking apart a service?”

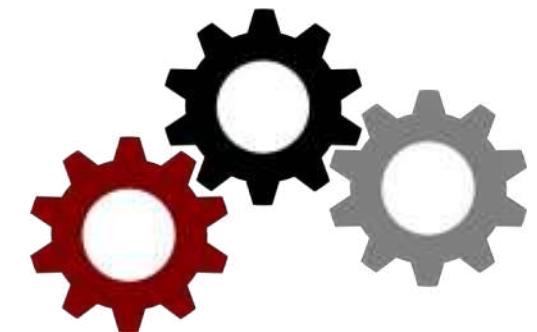


service
functionality

service granularity

granularity drivers

“when should I consider breaking apart a service?”



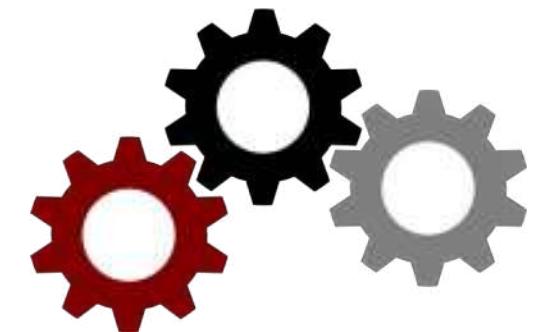
service
functionality



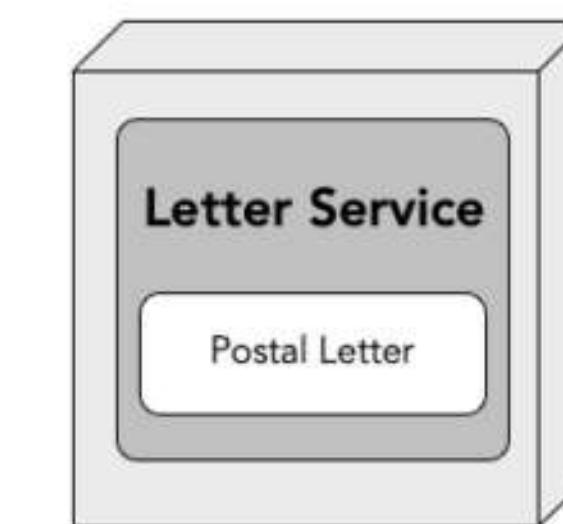
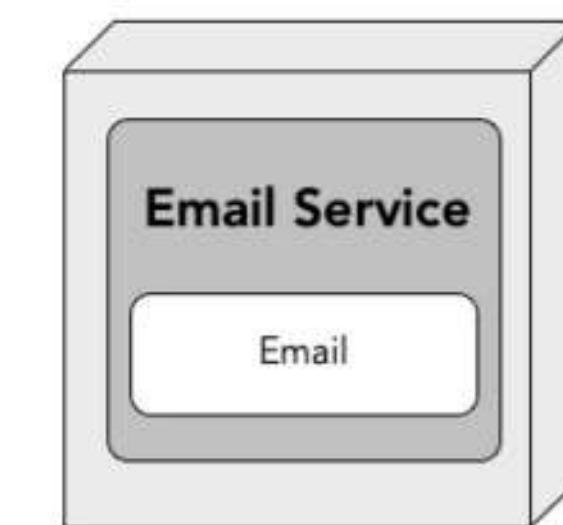
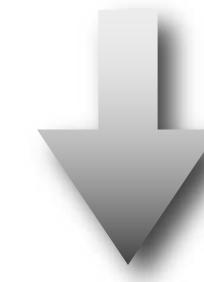
service granularity

granularity drivers

“when should I consider breaking apart a service?”



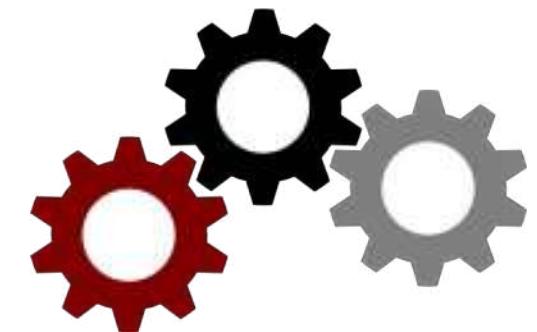
service
functionality



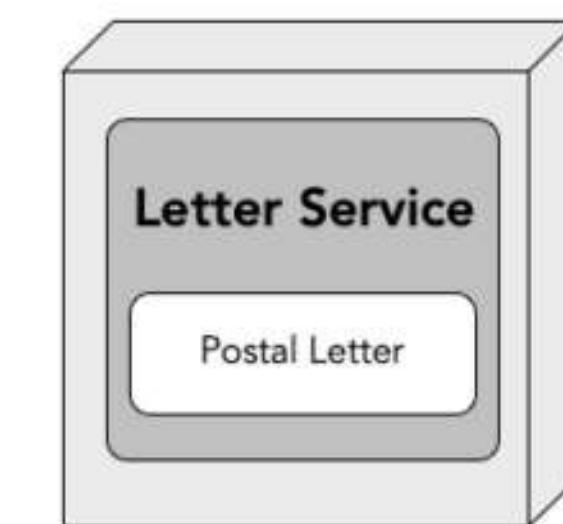
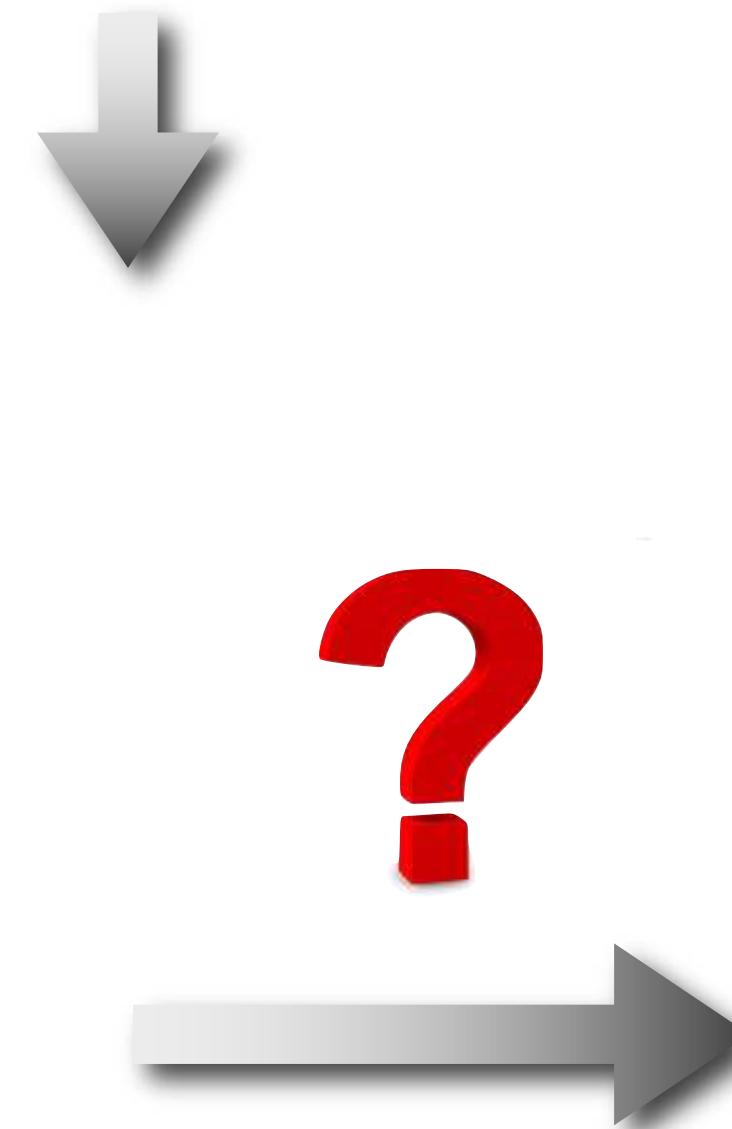
service granularity

granularity drivers

“when should I consider breaking apart a service?”



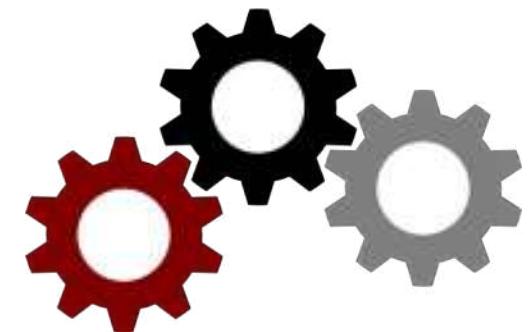
service
functionality



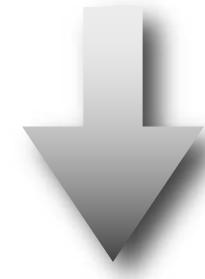
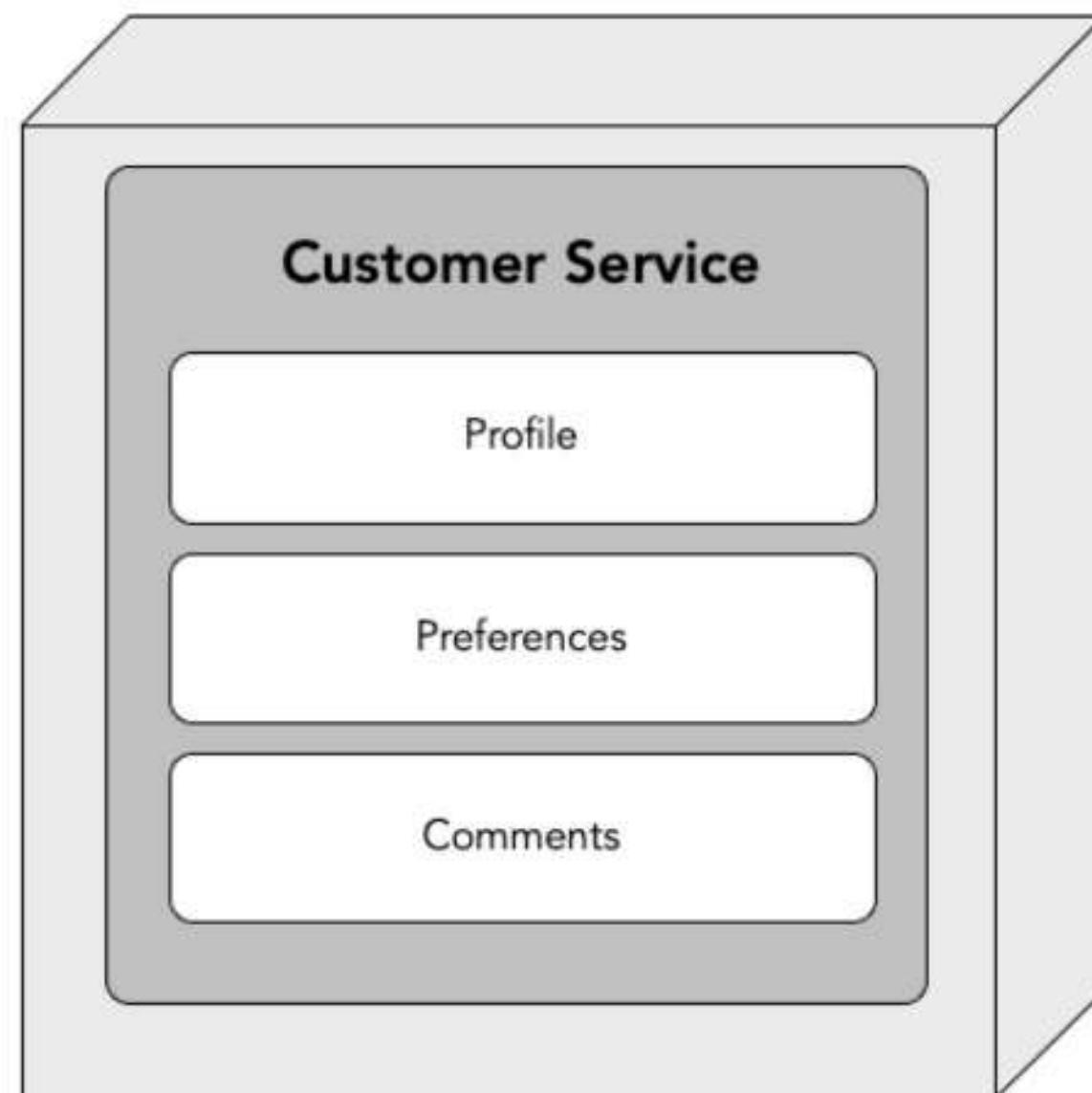
service granularity

granularity drivers

“when should I consider breaking apart a service?”



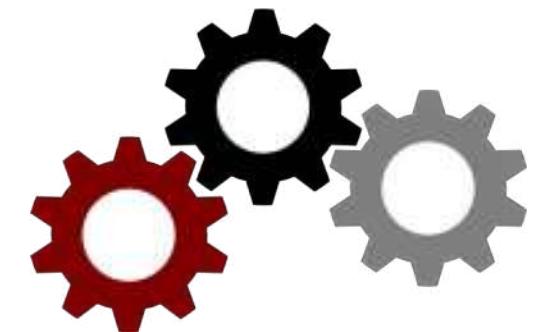
service
functionality



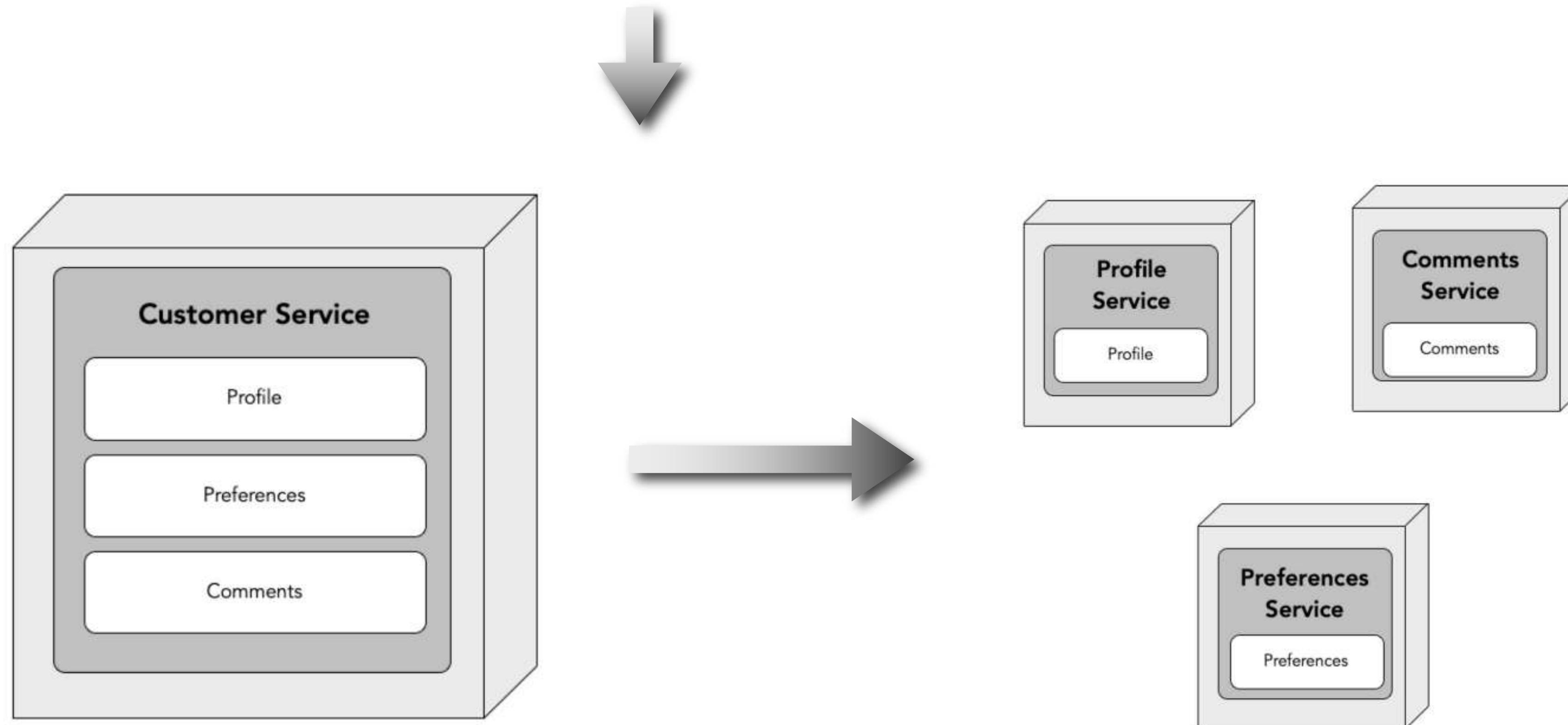
service granularity

granularity drivers

“when should I consider breaking apart a service?”



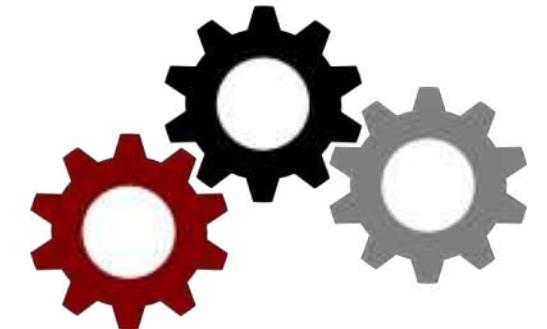
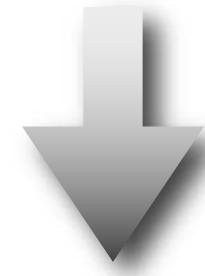
service
functionality



service granularity

granularity drivers

“when should I consider breaking apart a service?”

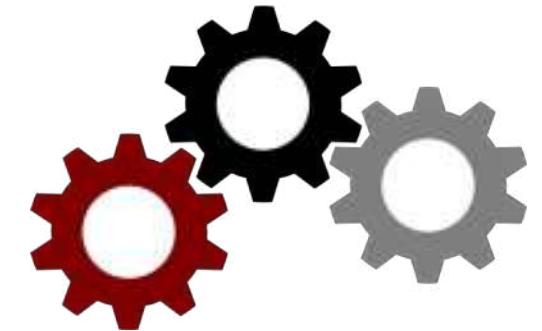
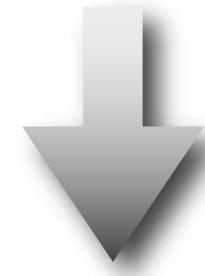


service
functionality

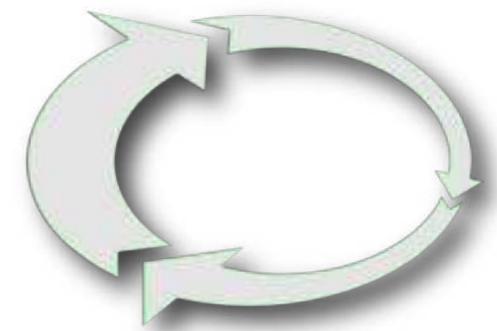
service granularity

granularity drivers

“when should I consider breaking apart a service?”



service
functionality

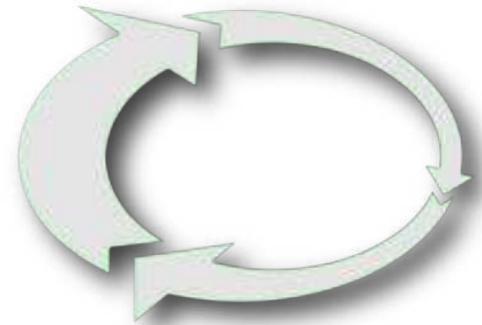


code
volatility

service granularity

granularity drivers

“when should I consider breaking apart a service?”

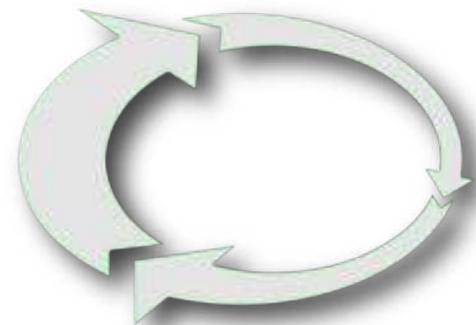


code
volatility

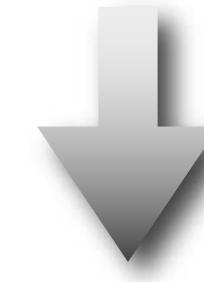
service granularity

granularity drivers

“when should I consider breaking apart a service?”



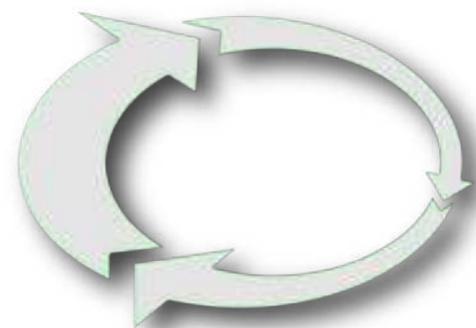
code
volatility



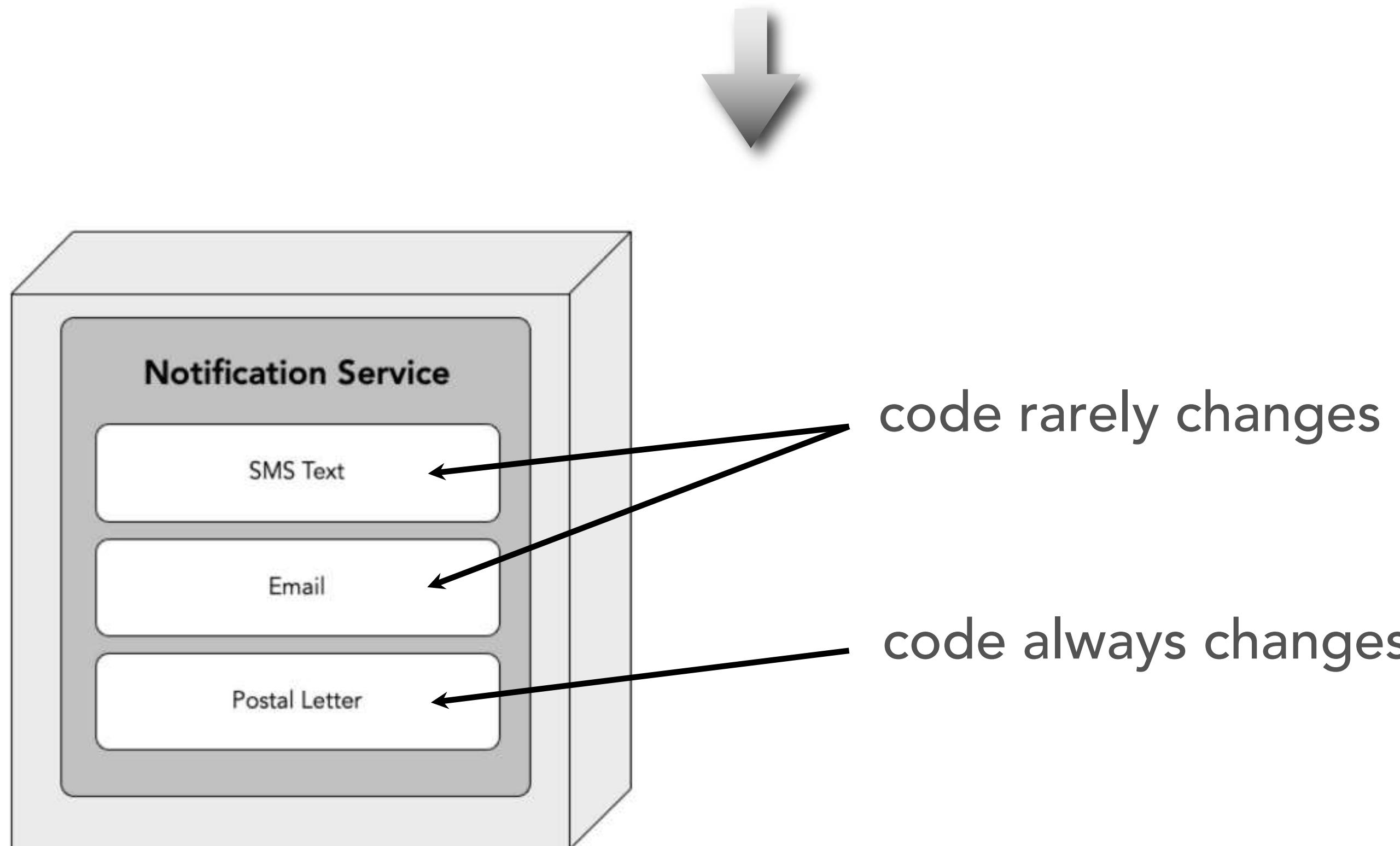
service granularity

granularity drivers

“when should I consider breaking apart a service?”



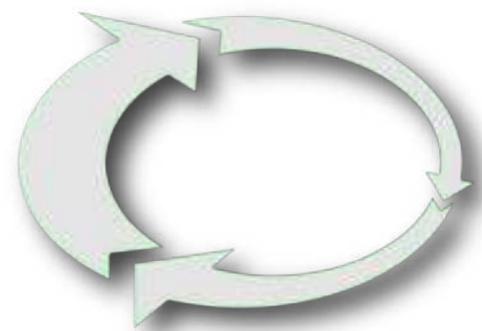
code
volatility



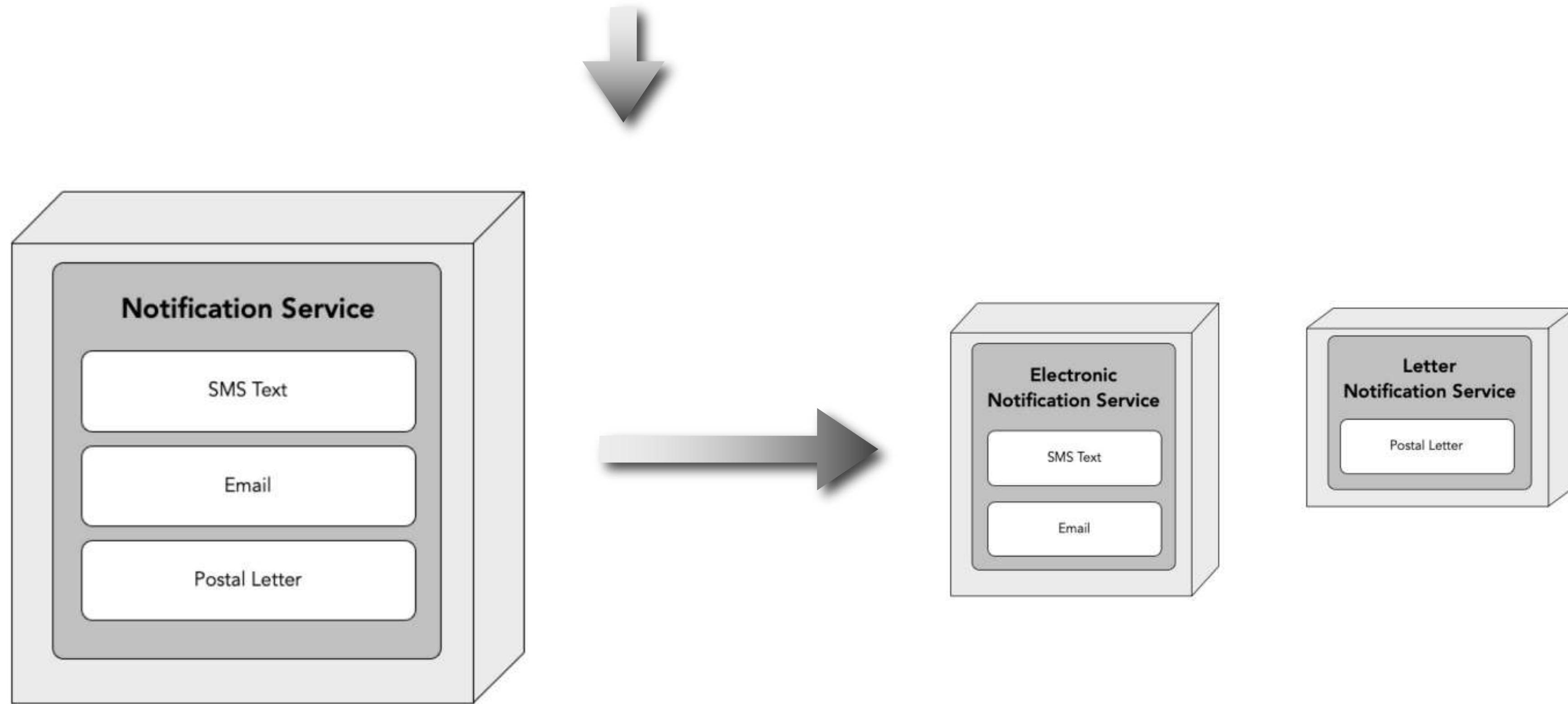
service granularity

granularity drivers

“when should I consider breaking apart a service?”



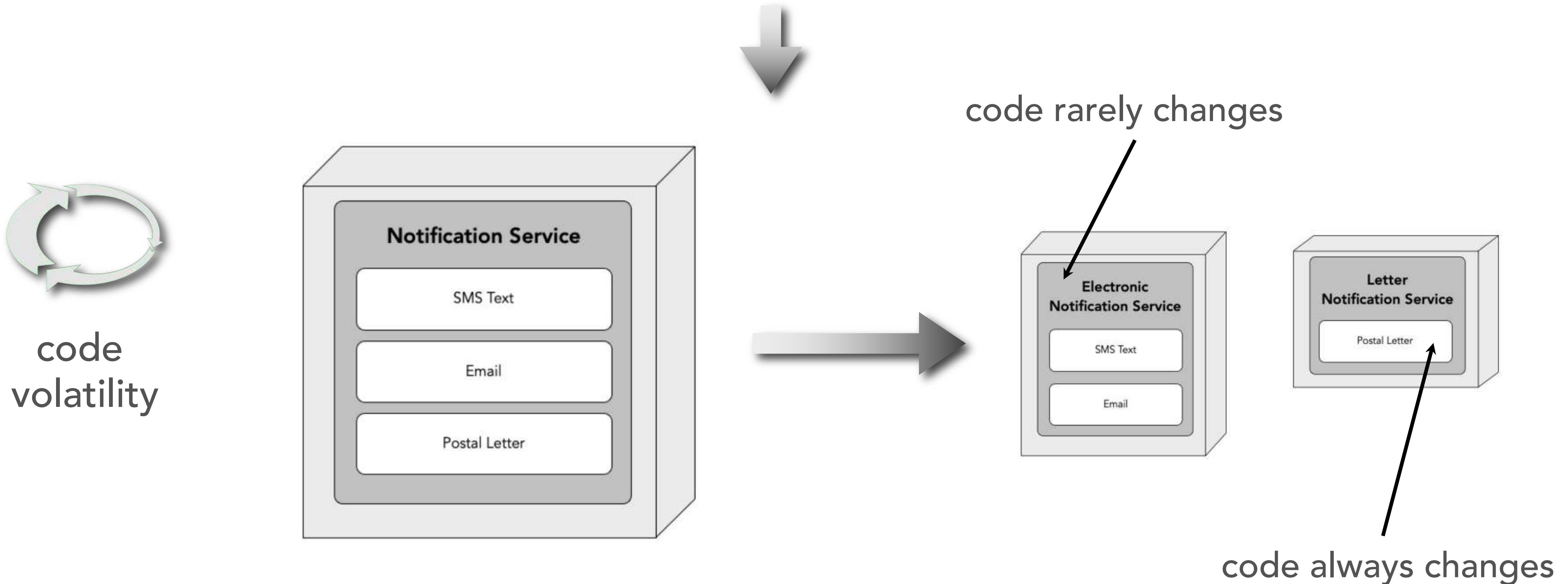
code
volatility



service granularity

granularity drivers

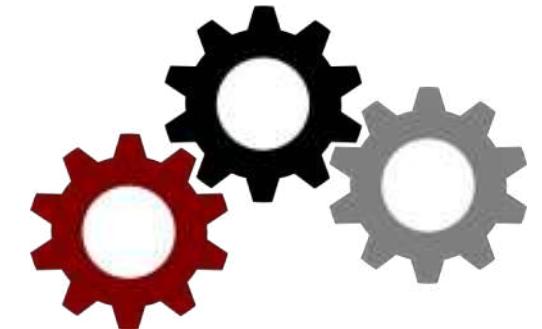
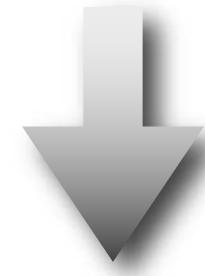
“when should I consider breaking apart a service?”



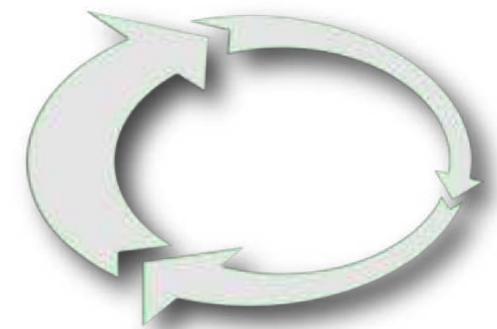
service granularity

granularity drivers

“when should I consider breaking apart a service?”



service
functionality

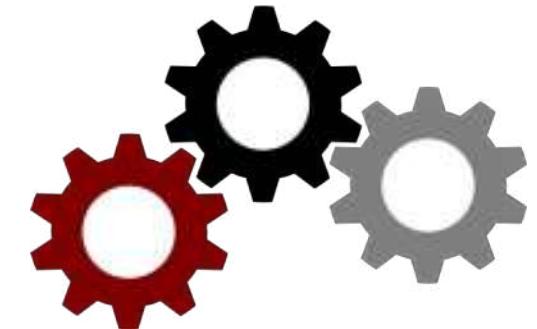


code
volatility

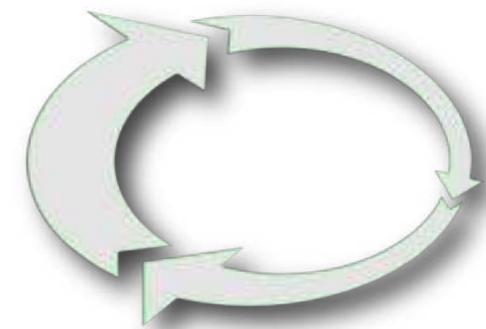
service granularity

granularity drivers

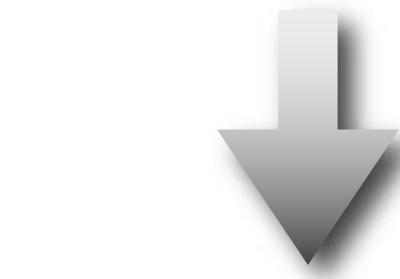
“when should I consider breaking apart a service?”



service
functionality



code
volatility

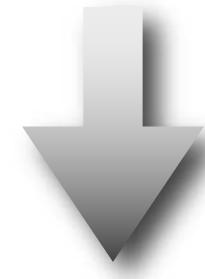


scalability and
throughput

service granularity

granularity drivers

“when should I consider breaking apart a service?”



scalability and
throughput

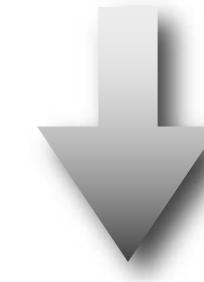
service granularity

granularity drivers

“when should I consider breaking apart a service?”



scalability and
throughput



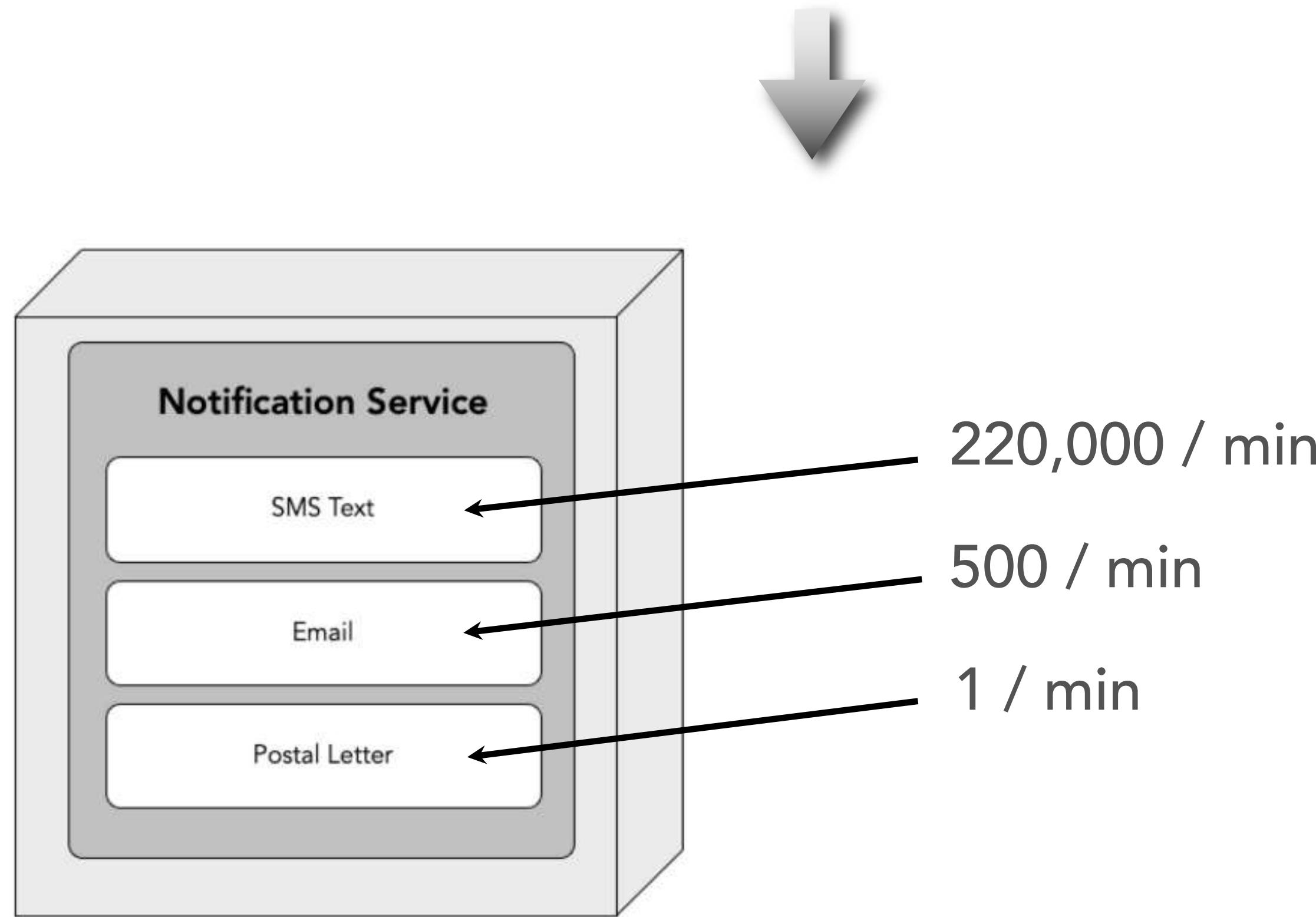
service granularity

granularity drivers

“when should I consider breaking apart a service?”



scalability and
throughput



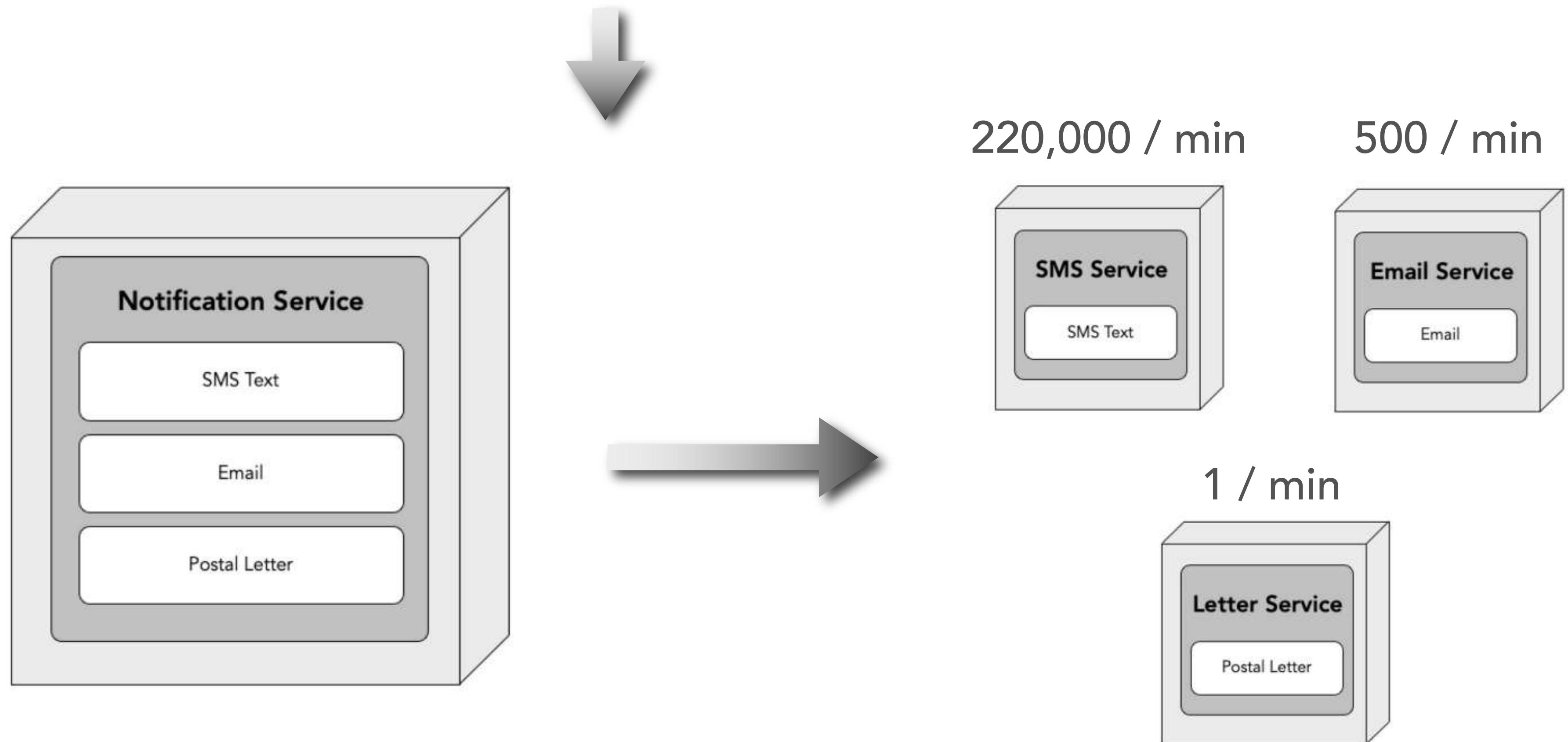
service granularity

granularity drivers

“when should I consider breaking apart a service?”



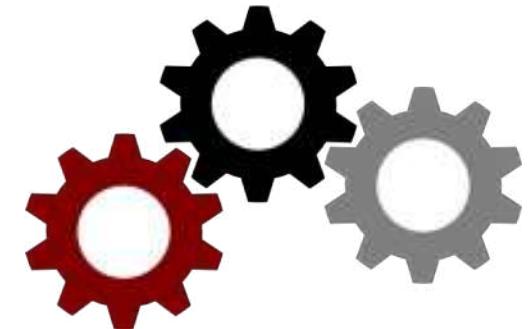
scalability and
throughput



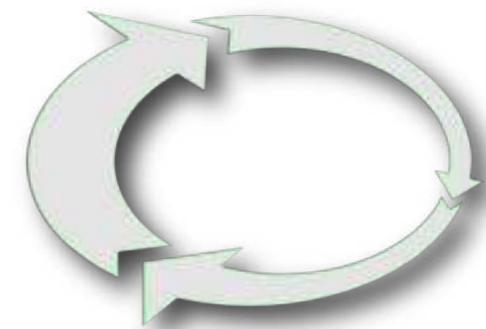
service granularity

granularity drivers

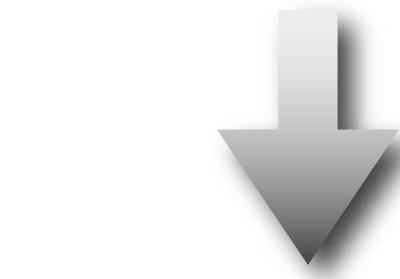
“when should I consider breaking apart a service?”



service
functionality



code
volatility

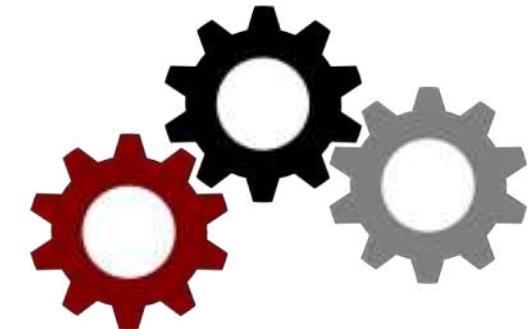


scalability and
throughput

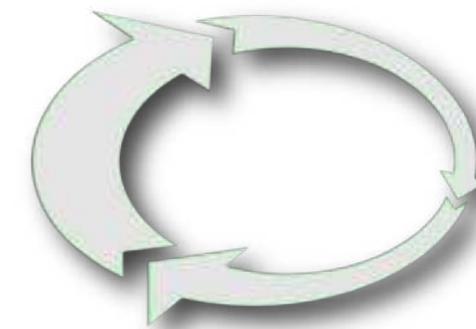
service granularity

granularity drivers

“when should I consider breaking apart a service?”



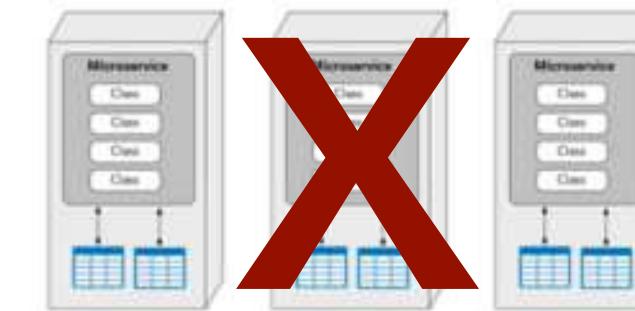
service
functionality



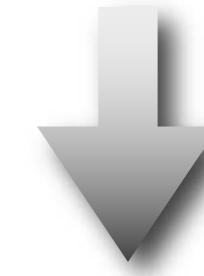
code
volatility



scalability and
throughput



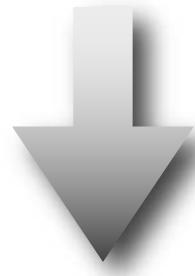
fault
tolerance



service granularity

granularity drivers

“when should I consider breaking apart a service?”

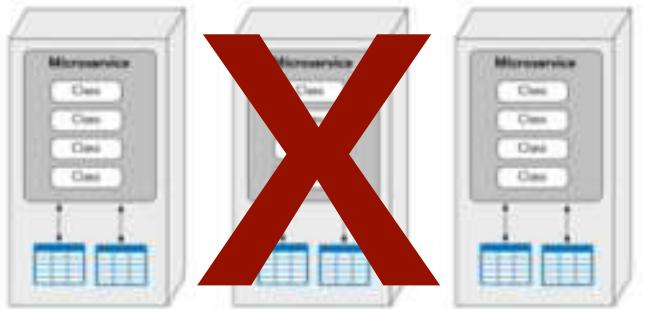
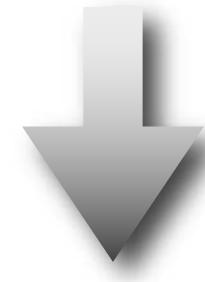


fault
tolerance

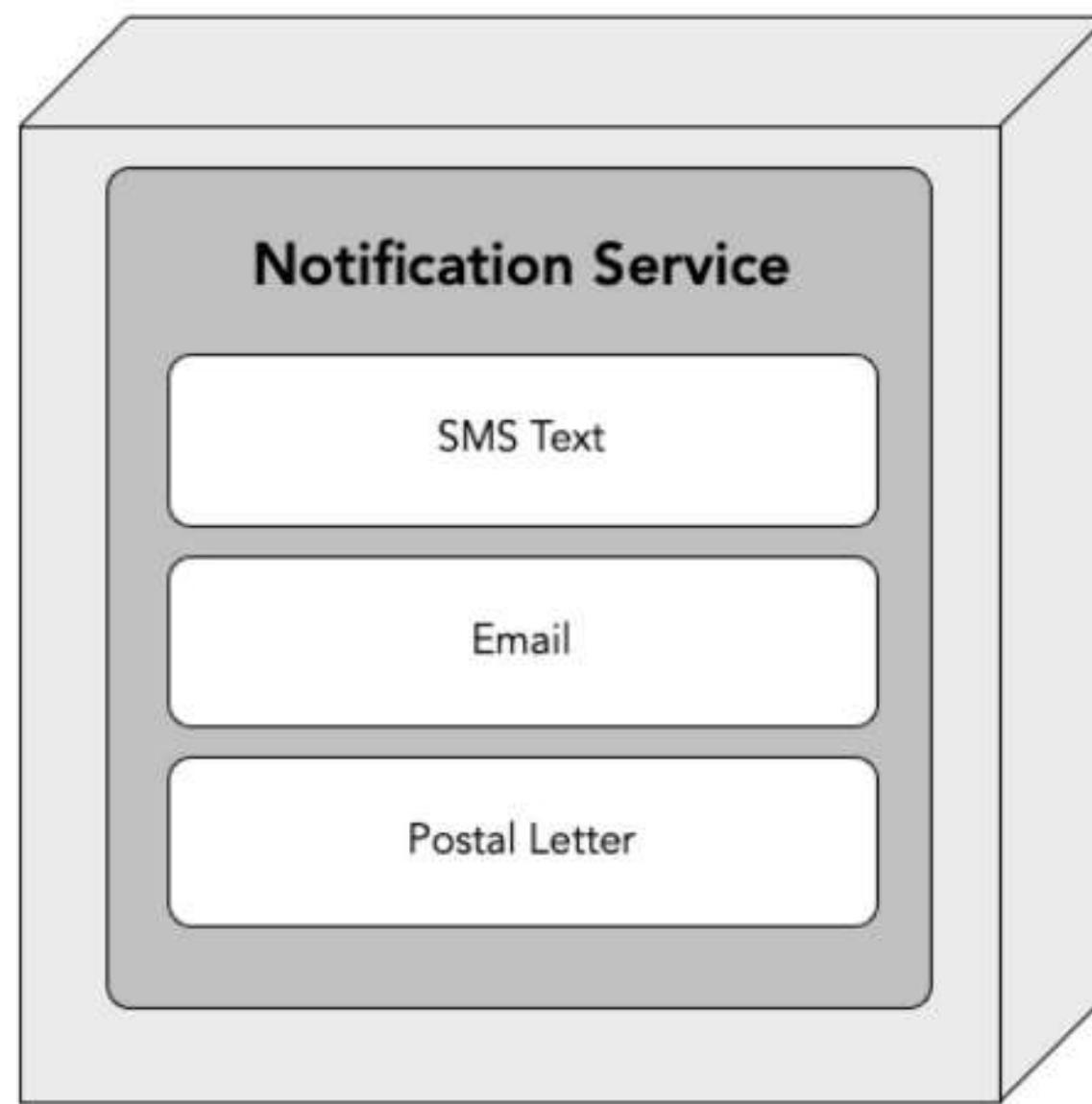
service granularity

granularity drivers

“when should I consider breaking apart a service?”



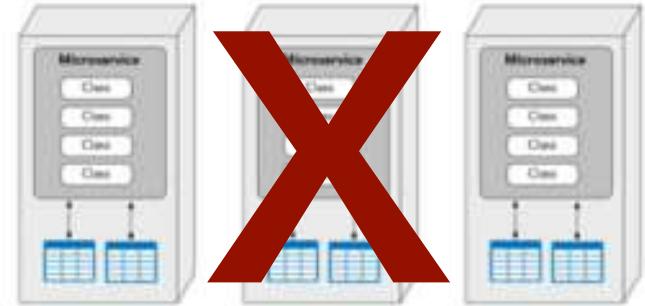
fault
tolerance



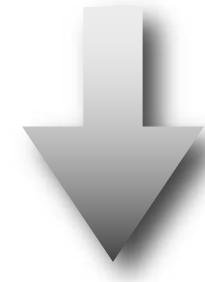
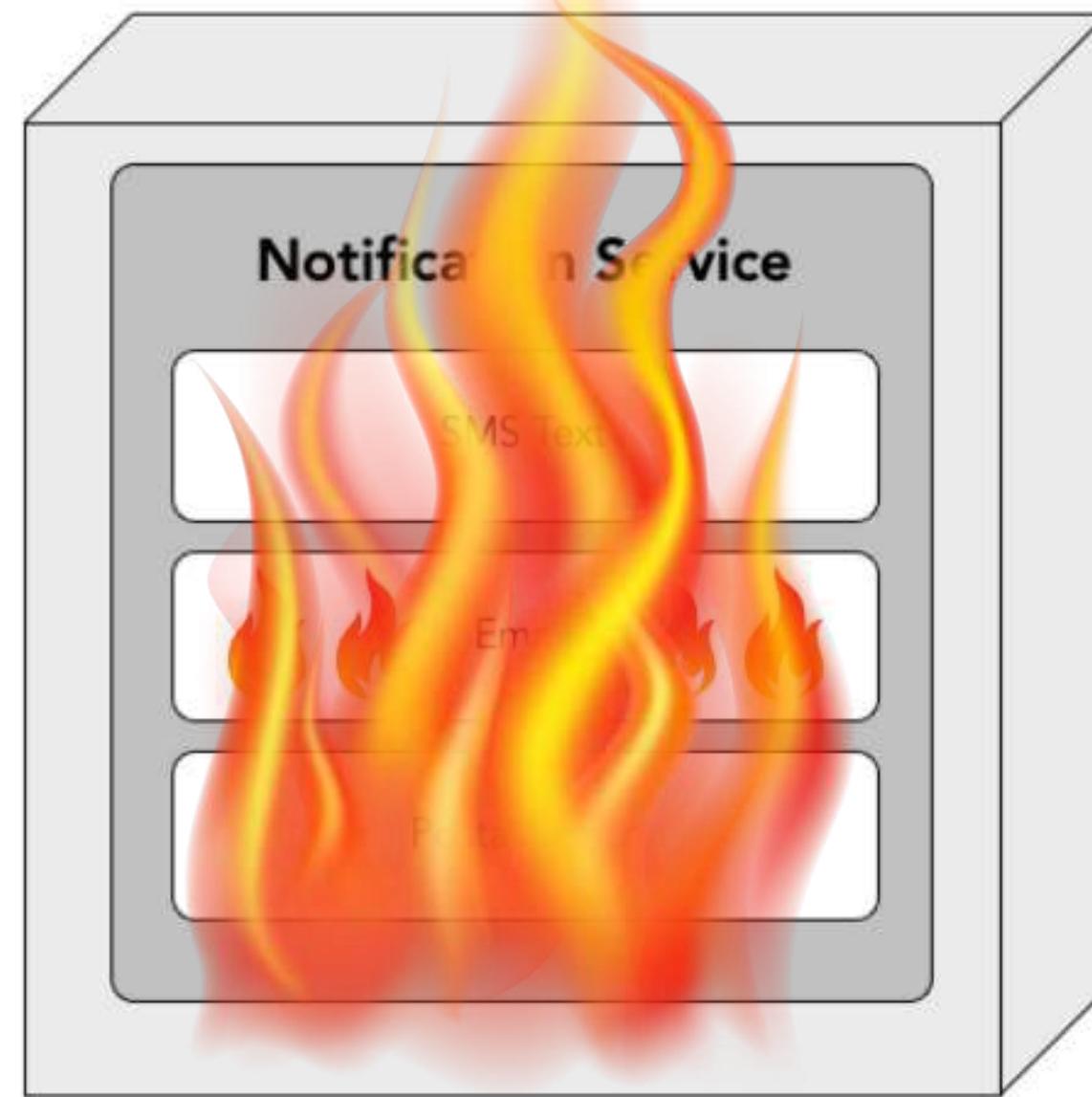
service granularity

granularity drivers

“when should I consider breaking apart a service?”



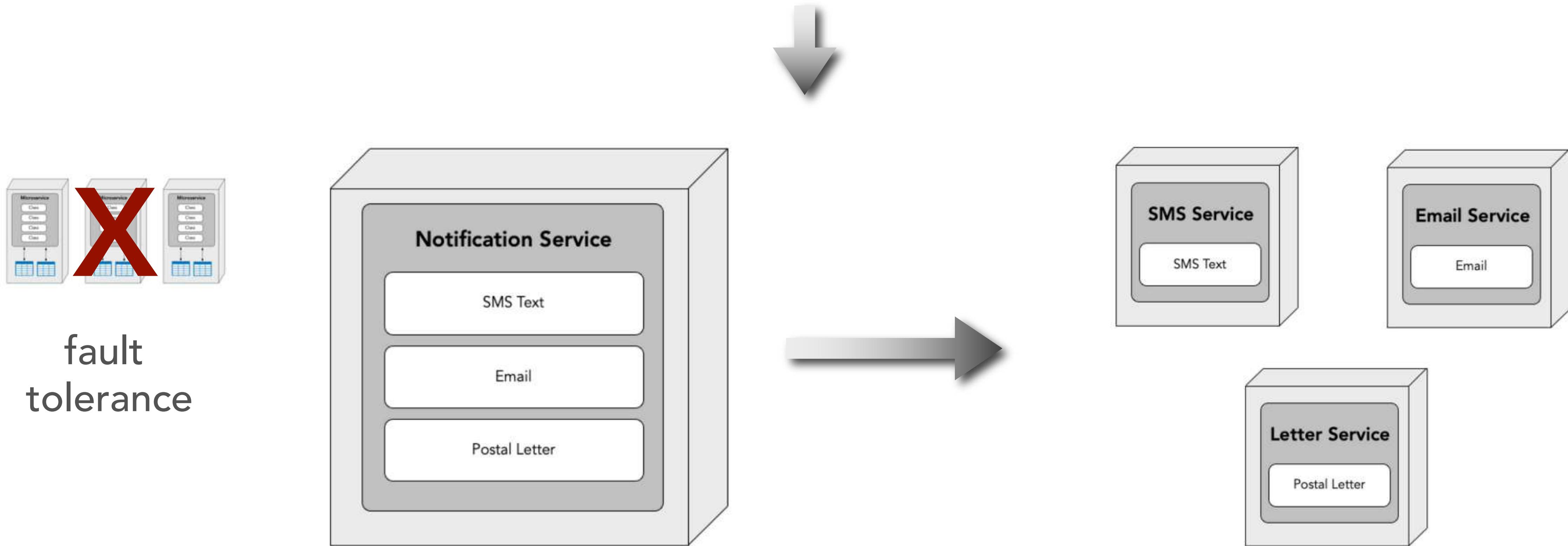
fault
tolerance



service granularity

granularity drivers

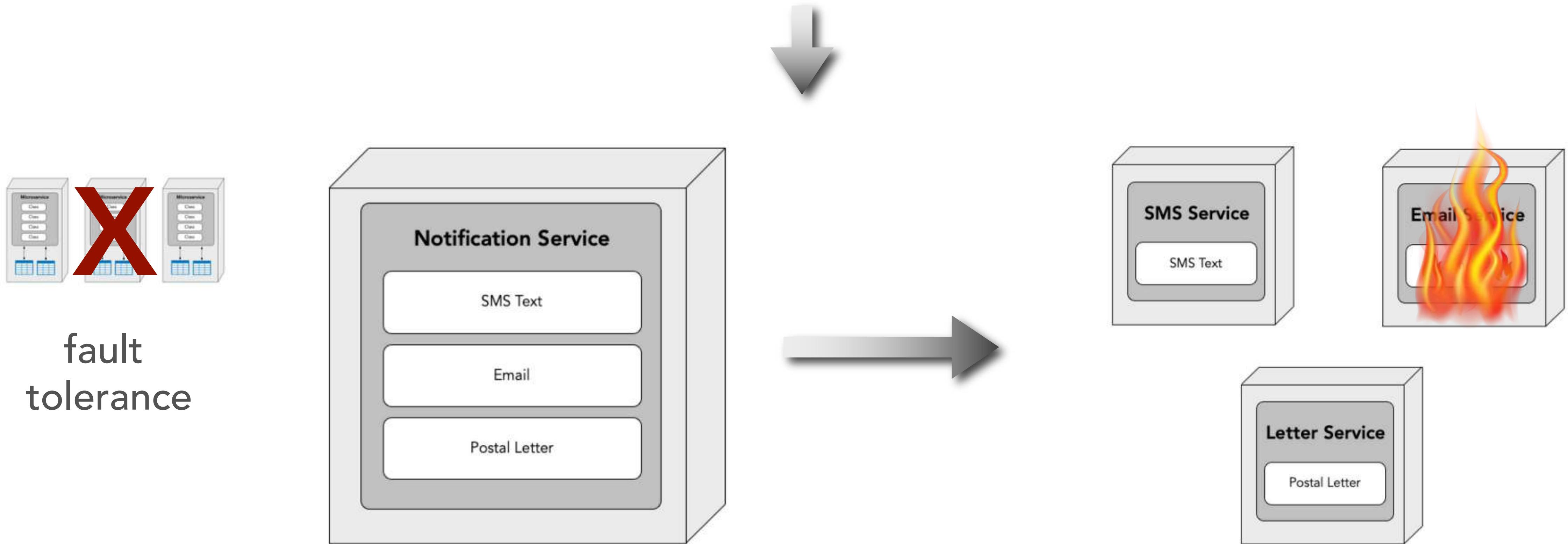
“when should I consider breaking apart a service?”



service granularity

granularity drivers

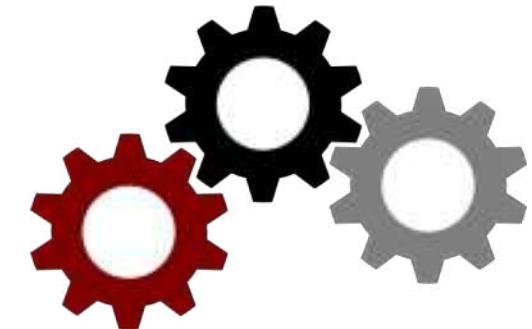
“when should I consider breaking apart a service?”



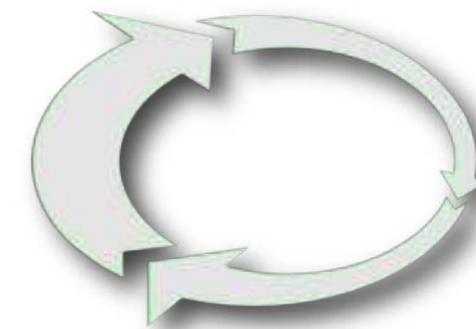
service granularity

granularity drivers

“when should I consider breaking apart a service?”



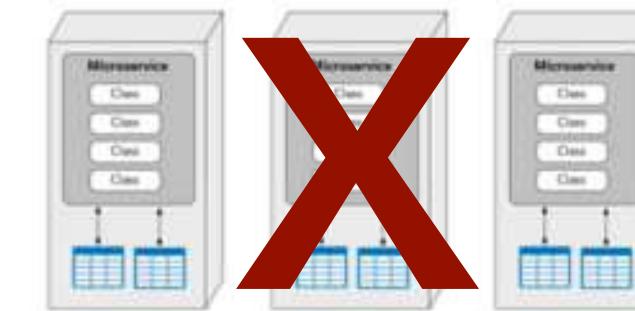
service
functionality



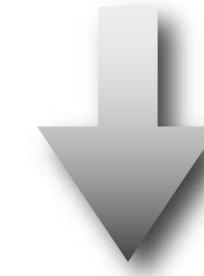
code
volatility



scalability and
throughput



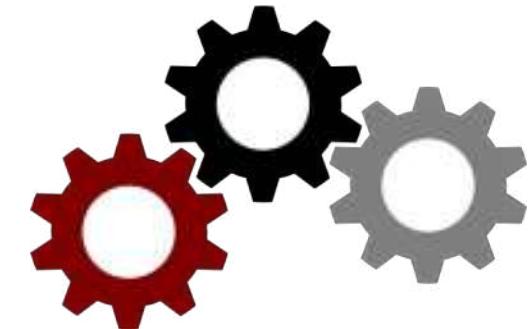
fault
tolerance



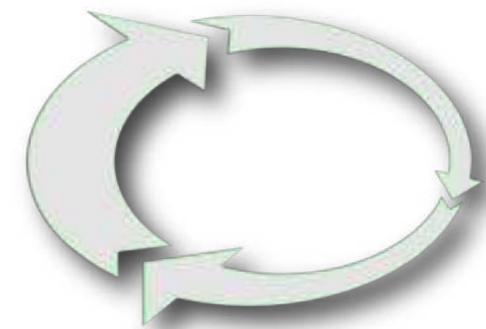
service granularity

granularity drivers

“when should I consider breaking apart a service?”



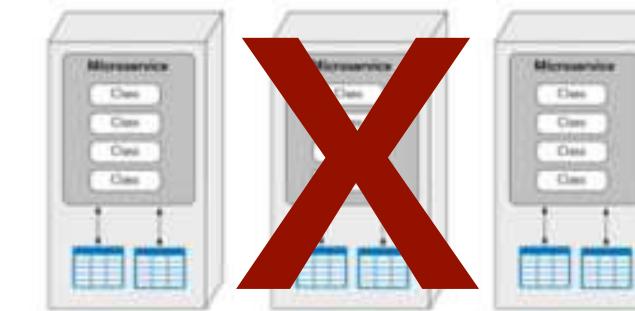
service
functionality



code
volatility



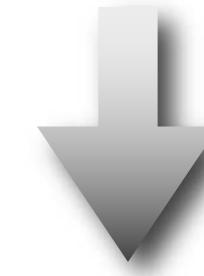
scalability and
throughput



fault
tolerance



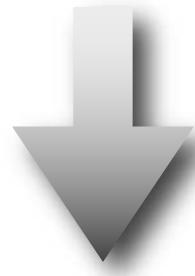
data
security



service granularity

granularity drivers

“when should I consider breaking apart a service?”

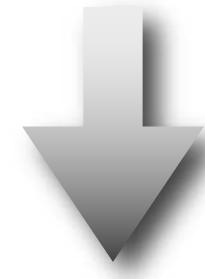


data
security

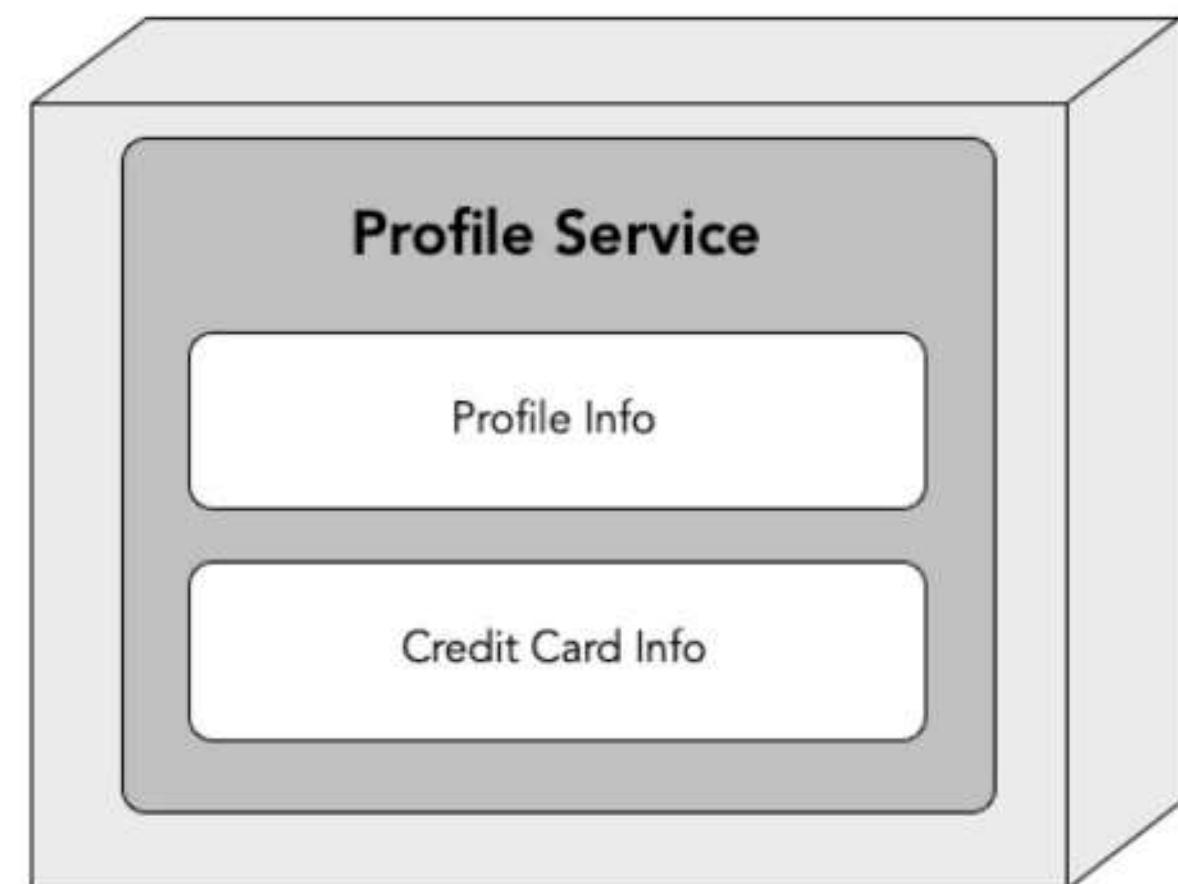
service granularity

granularity drivers

“when should I consider breaking apart a service?”



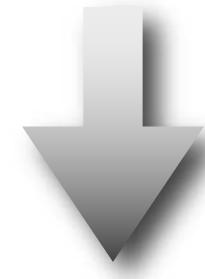
data
security



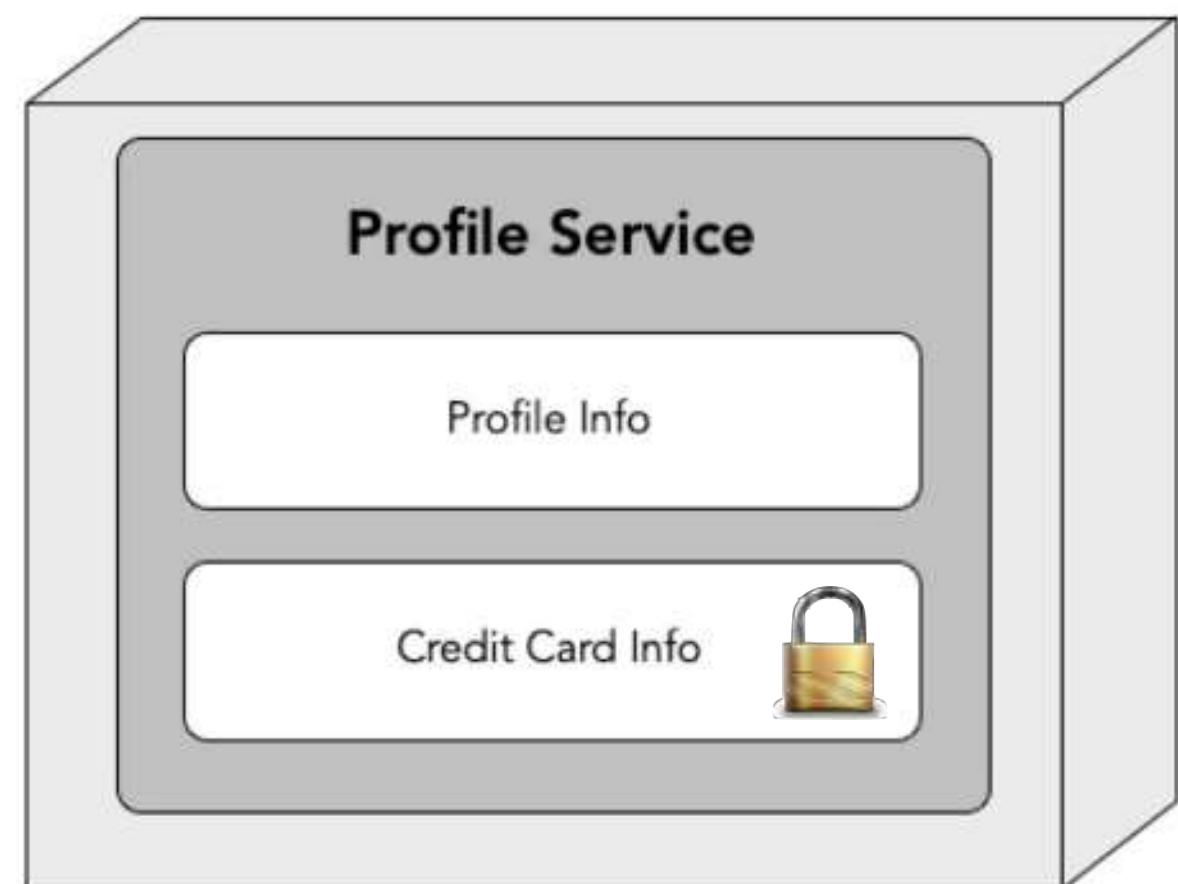
service granularity

granularity drivers

“when should I consider breaking apart a service?”



data
security



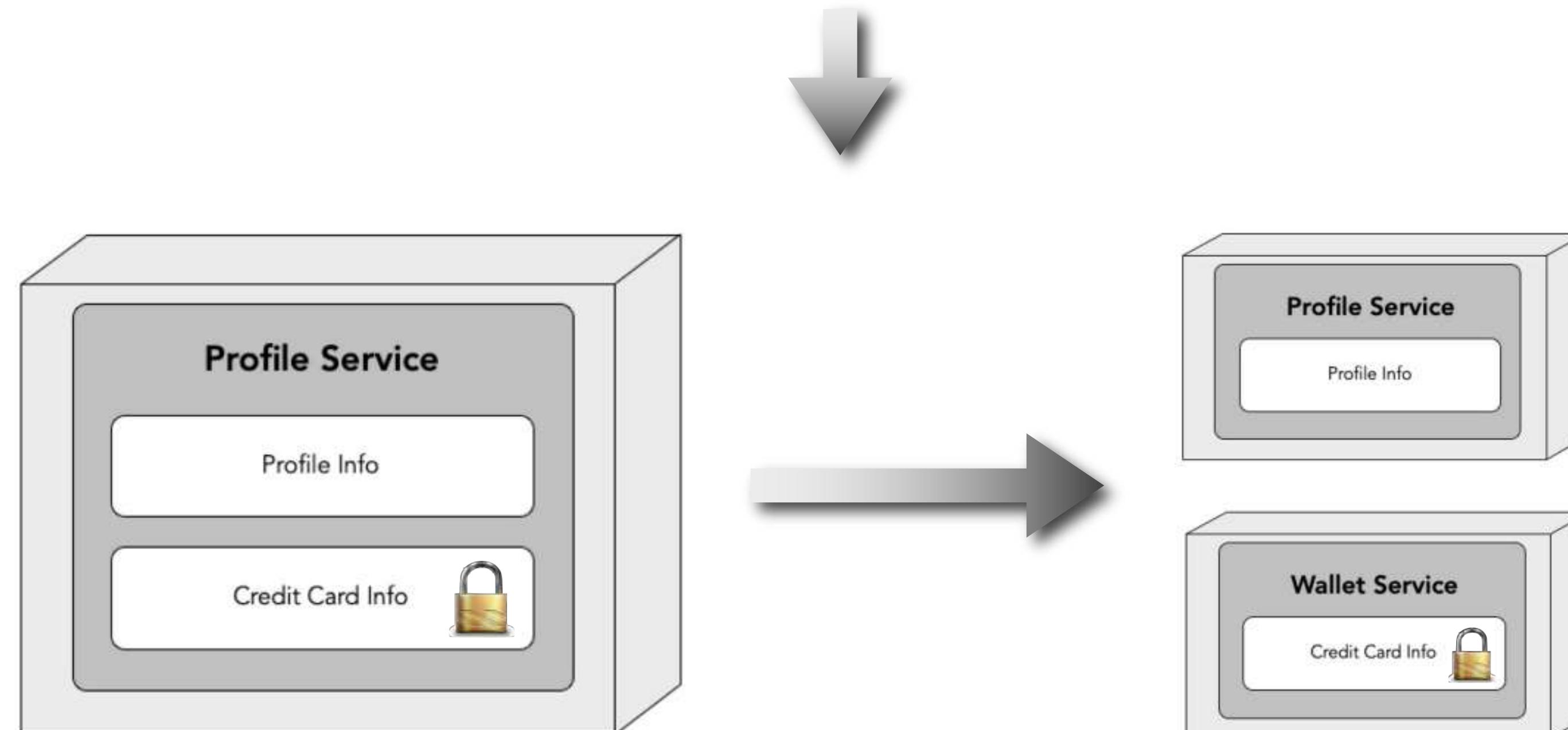
service granularity

granularity drivers

“when should I consider breaking apart a service?”



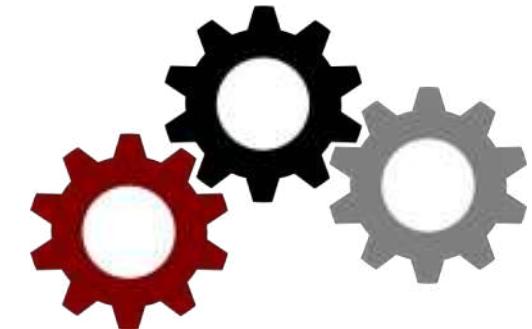
data
security



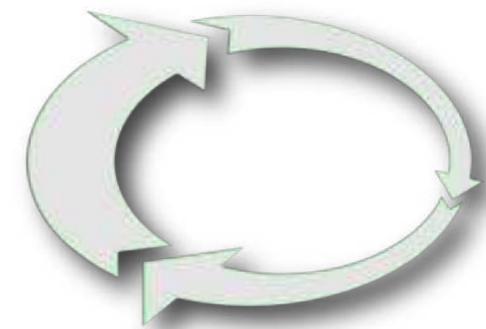
service granularity

granularity drivers

“when should I consider breaking apart a service?”



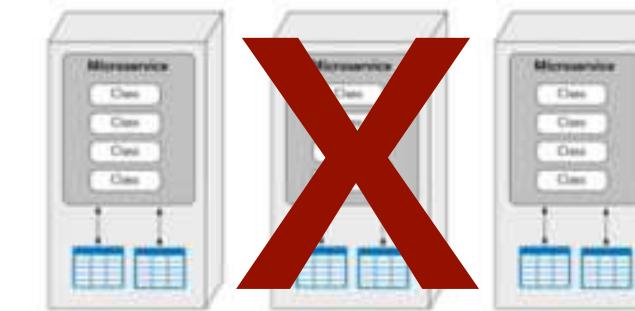
service
functionality



code
volatility



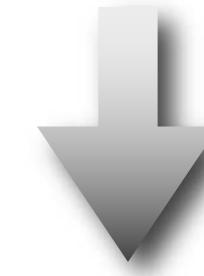
scalability and
throughput



fault
tolerance



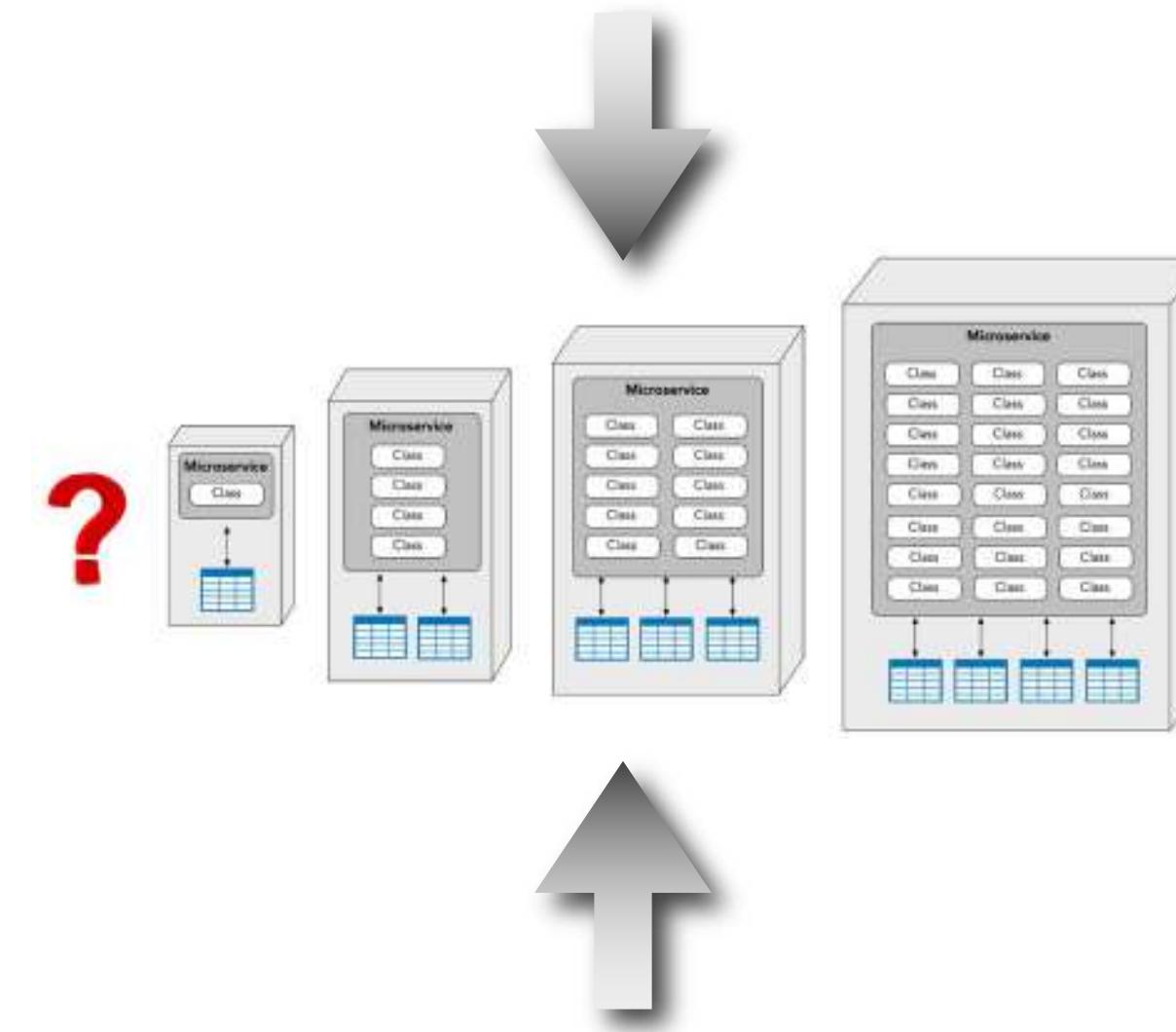
data
security



service granularity

granularity drivers

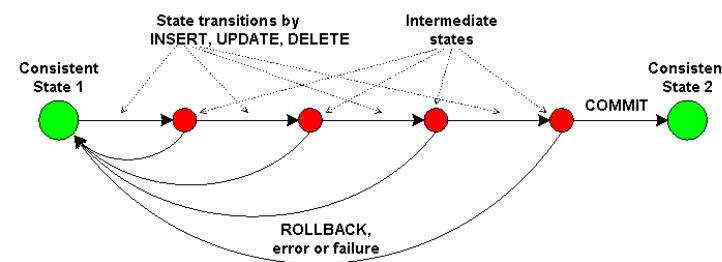
“when should I consider breaking apart a service?”



granularity factors

“what factors influence service granularity?”

service granularity

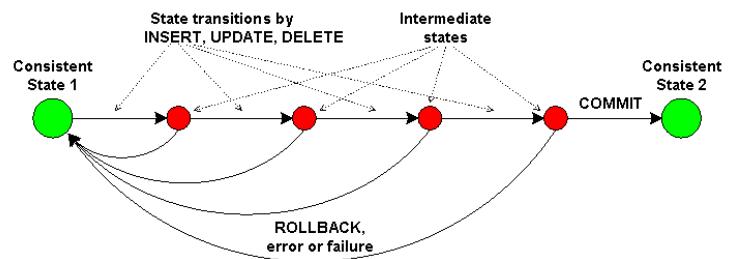


database
transactions

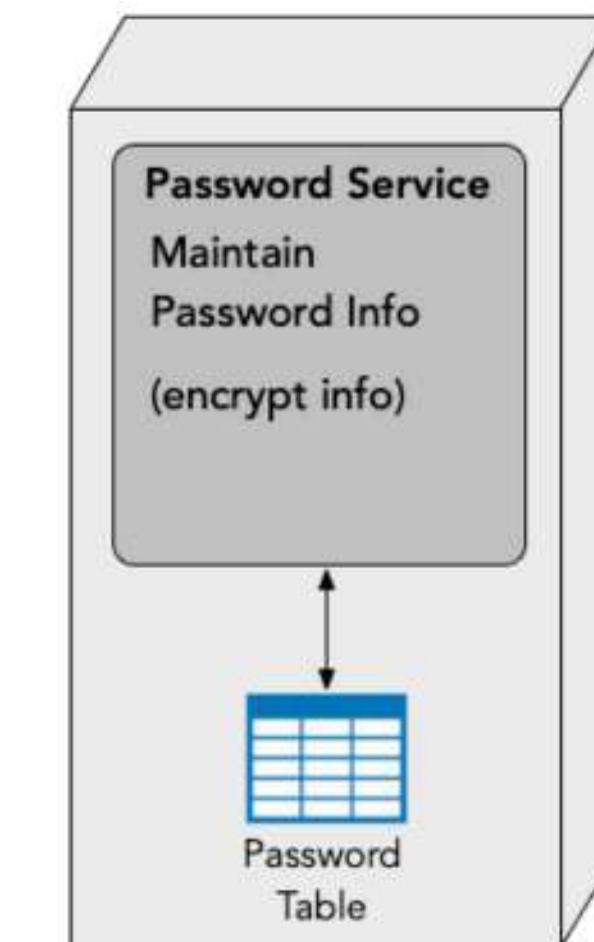
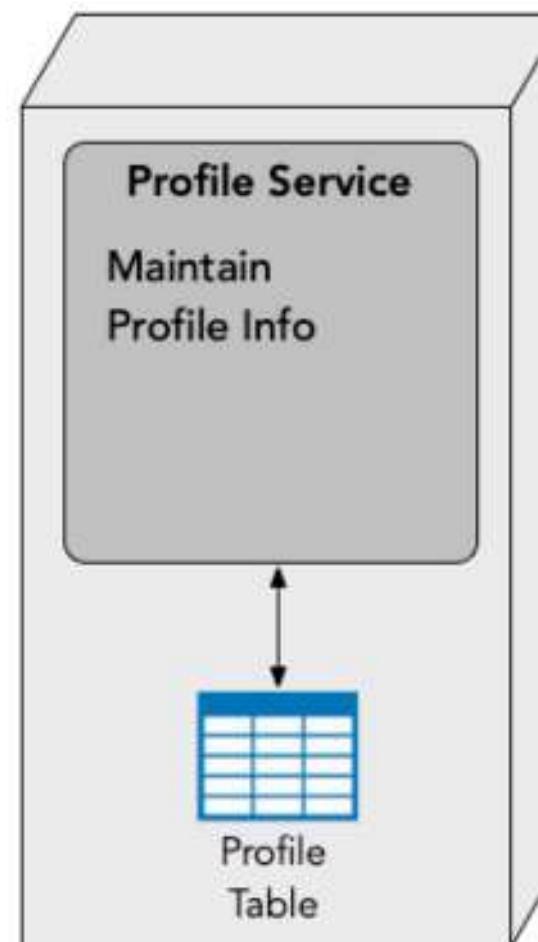


granularity factors
“what factors influence service granularity?”

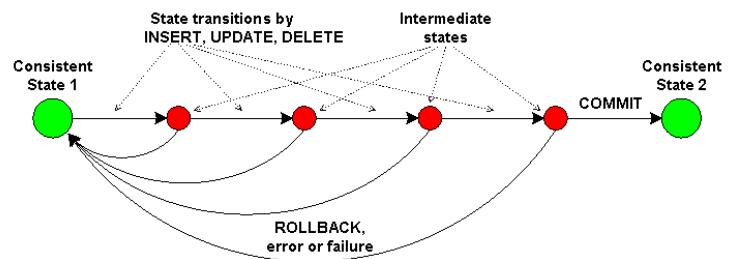
service granularity



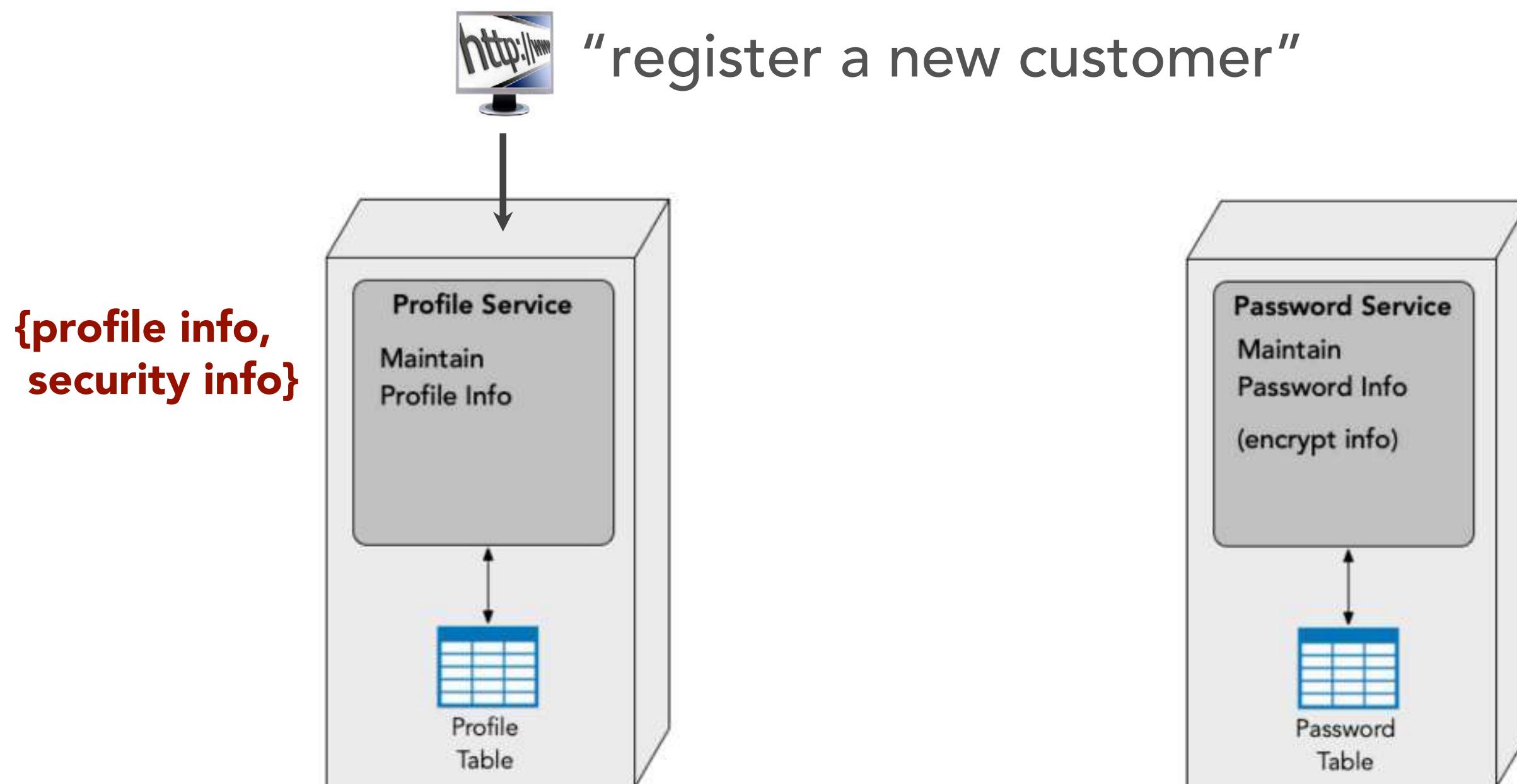
database
transactions



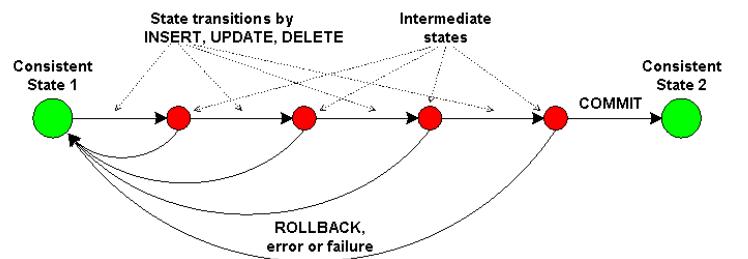
service granularity



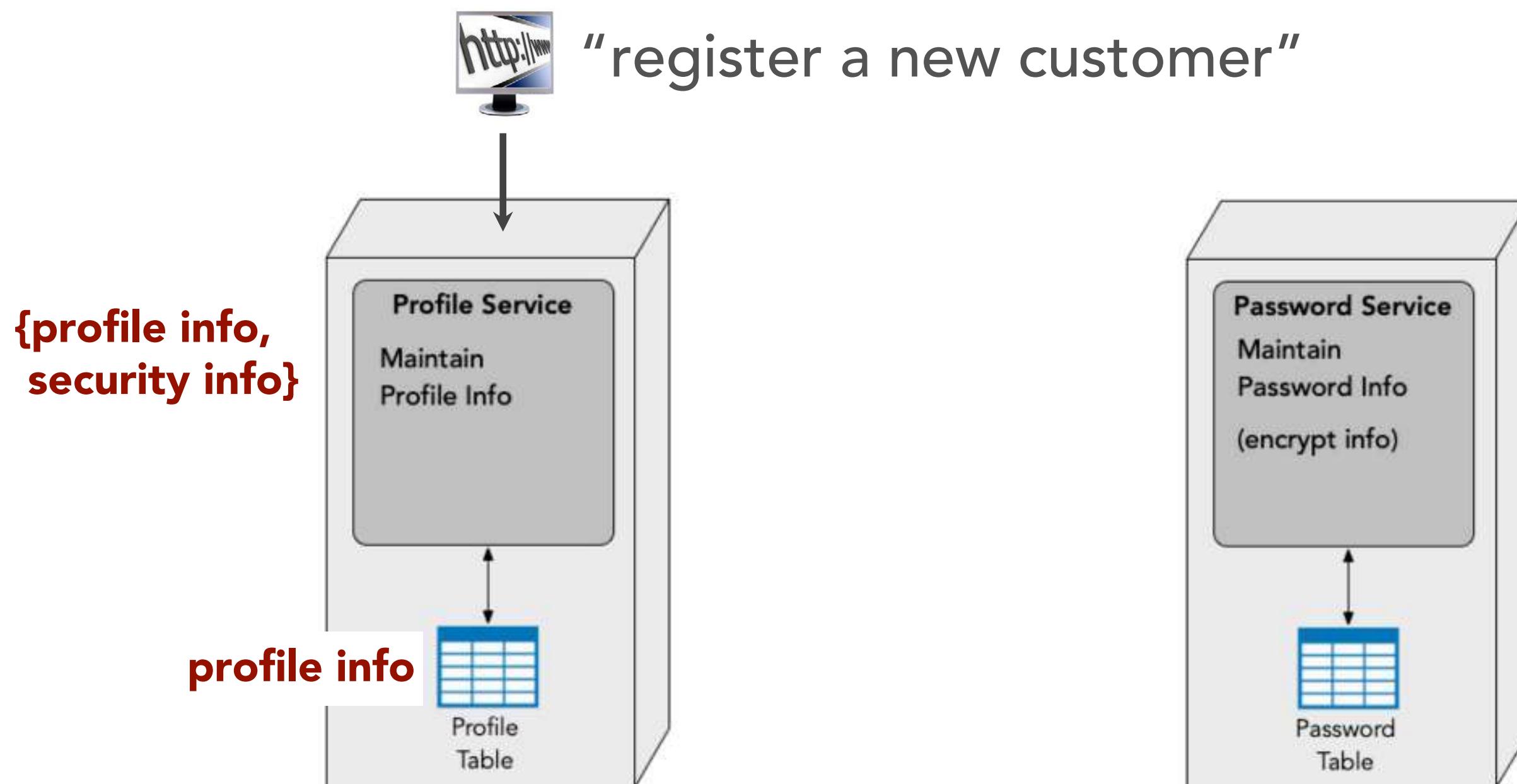
database
transactions



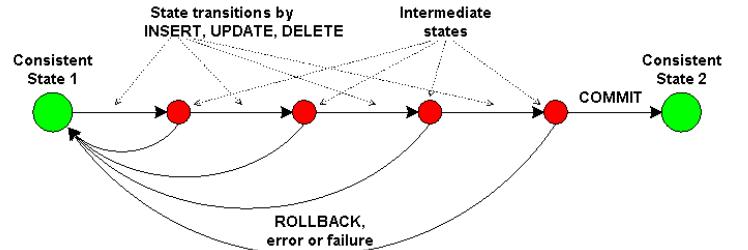
service granularity



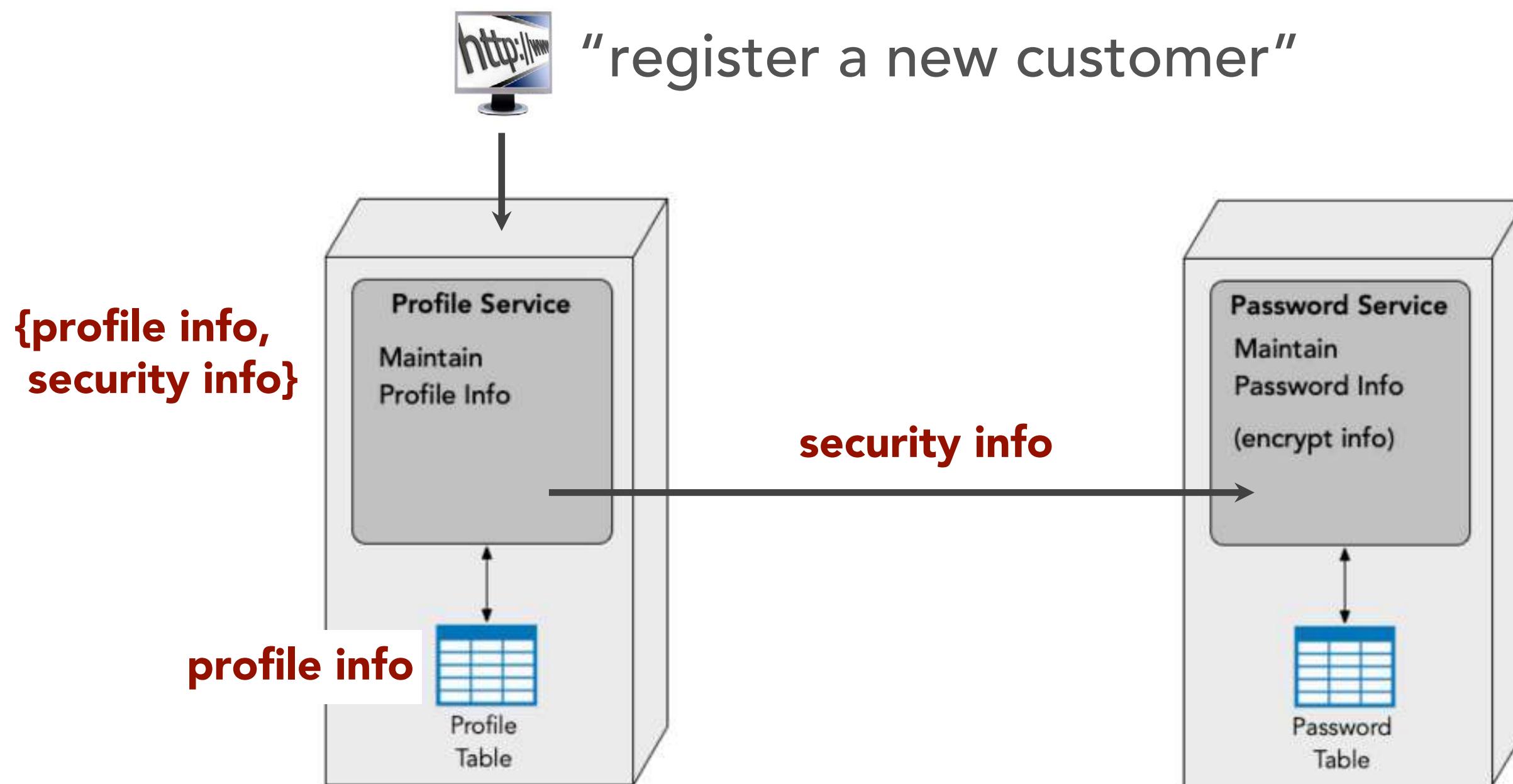
database
transactions



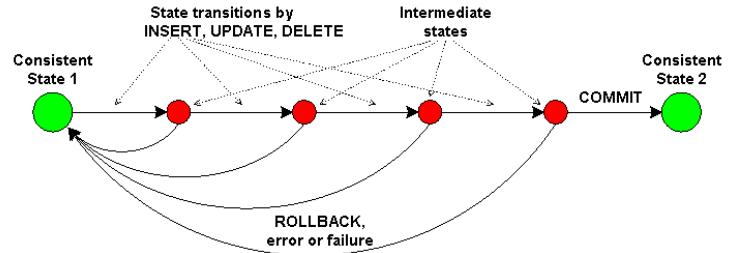
service granularity



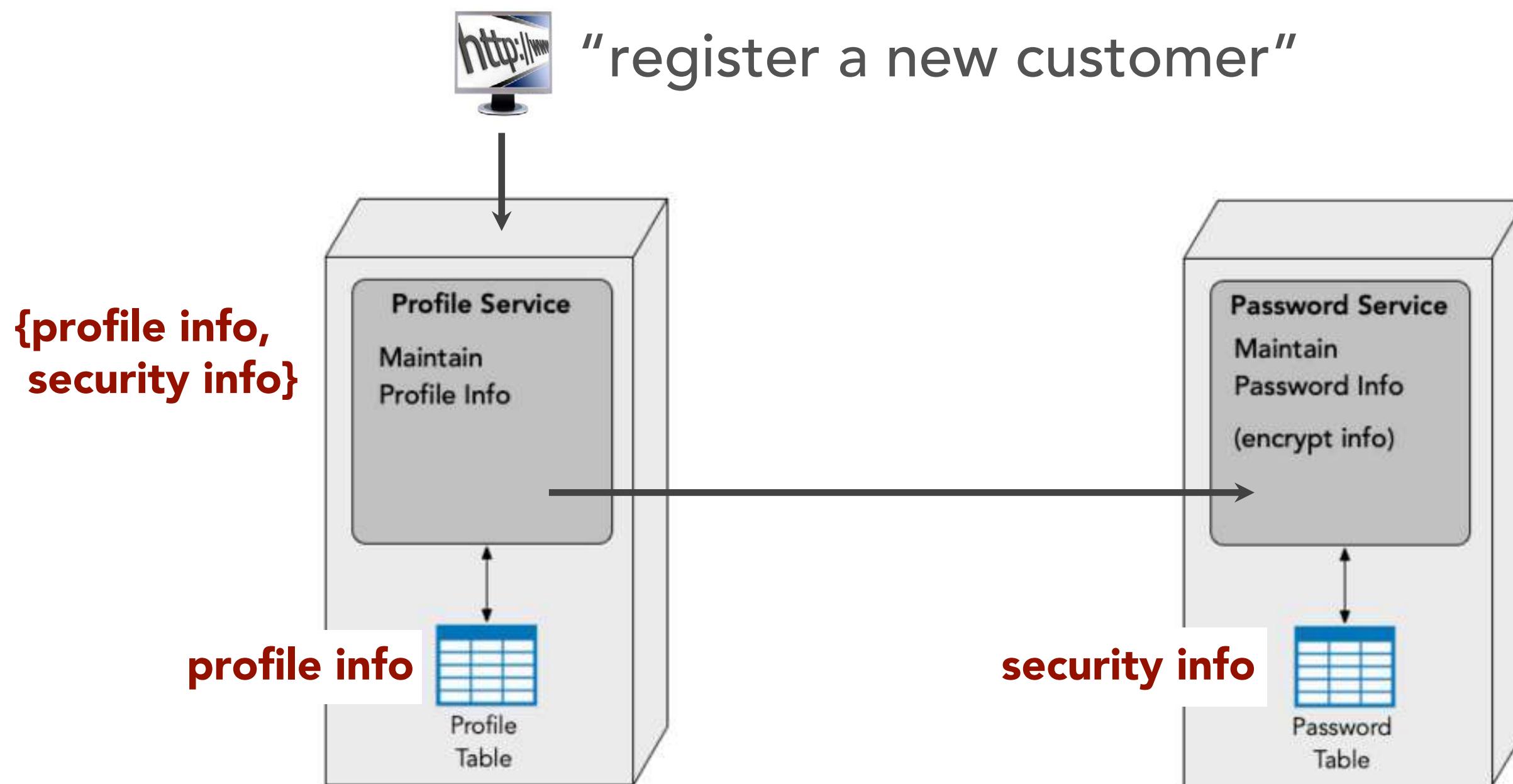
database
transactions



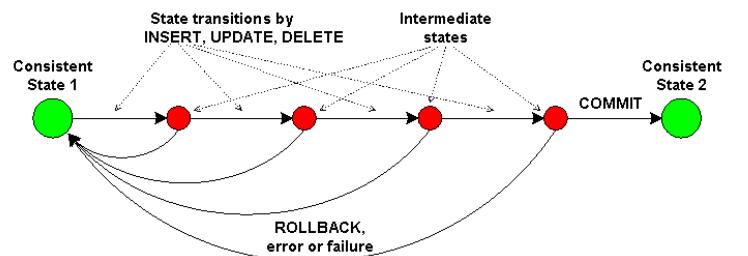
service granularity



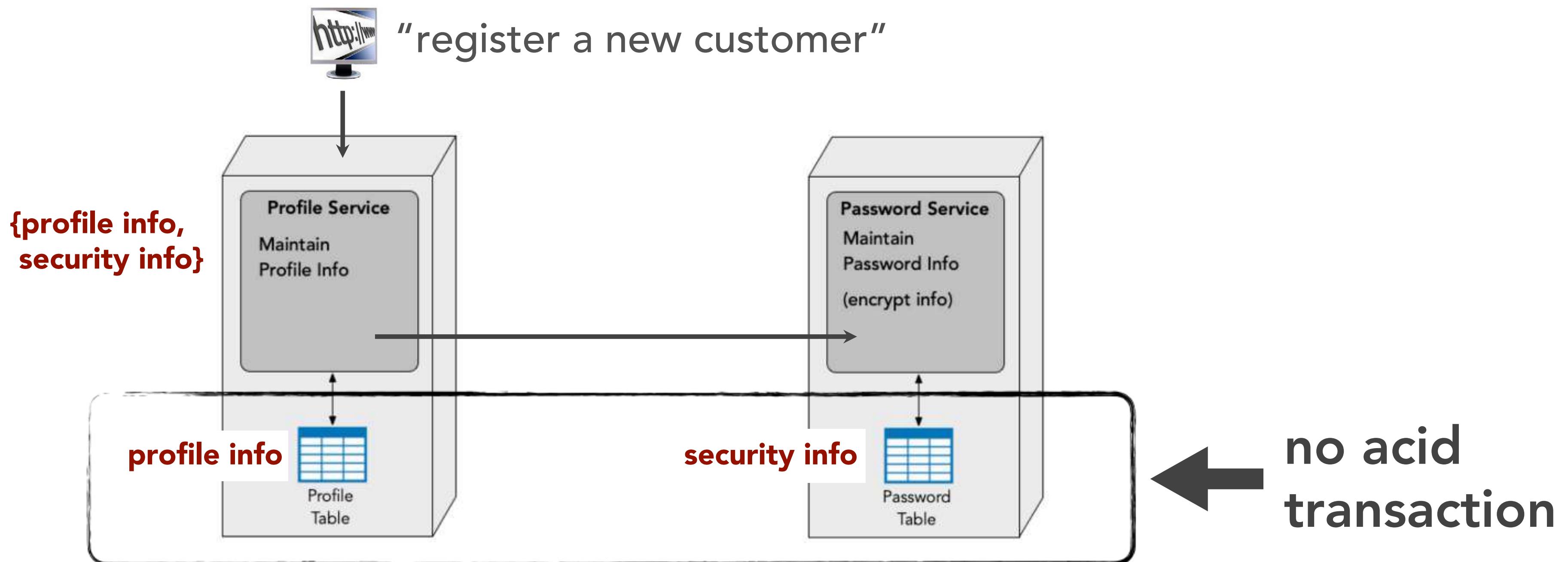
database
transactions



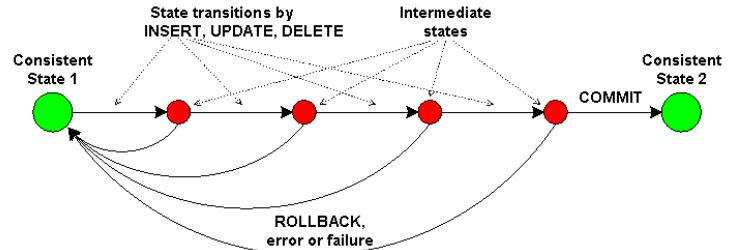
service granularity



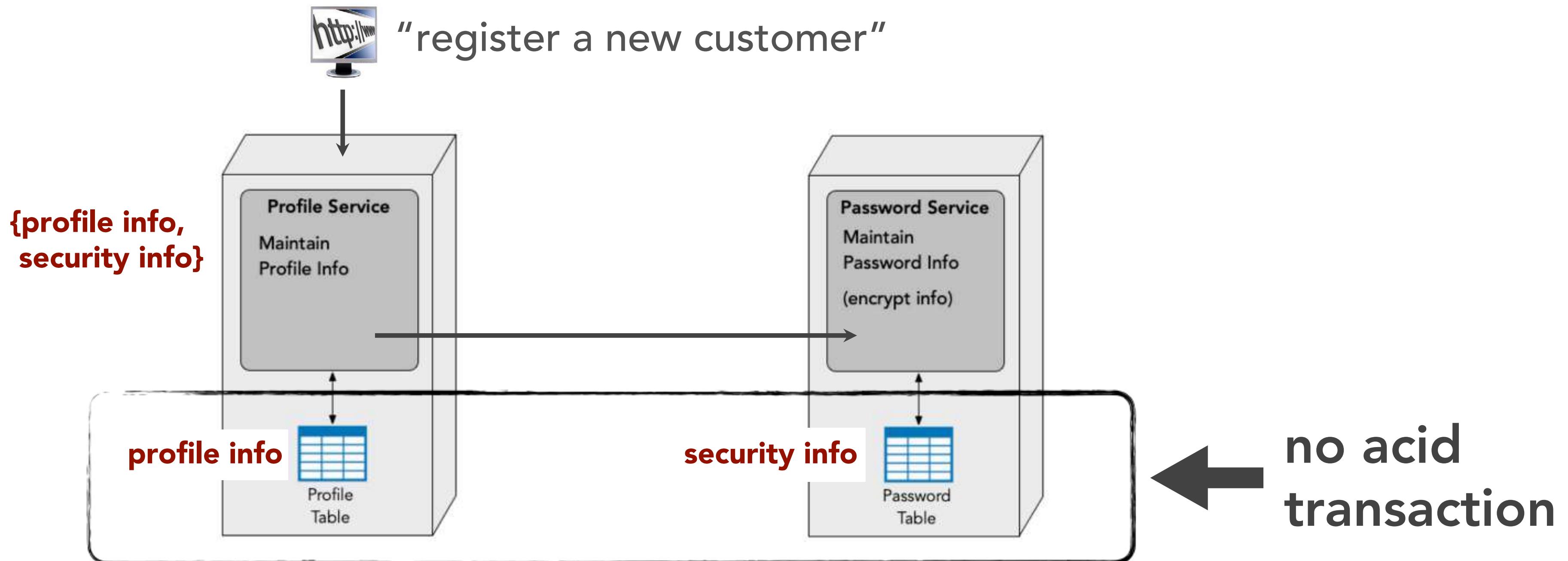
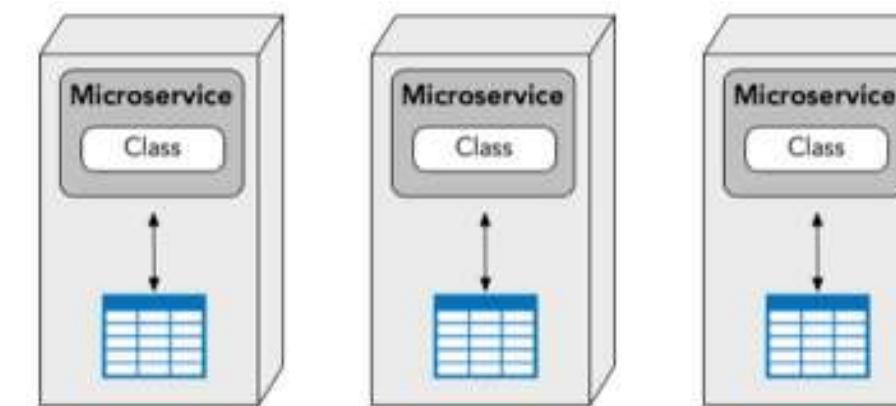
database
transactions



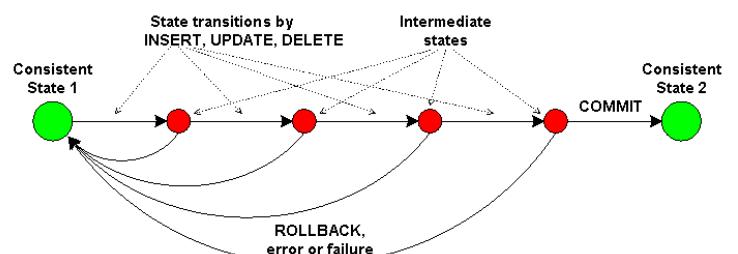
service granularity



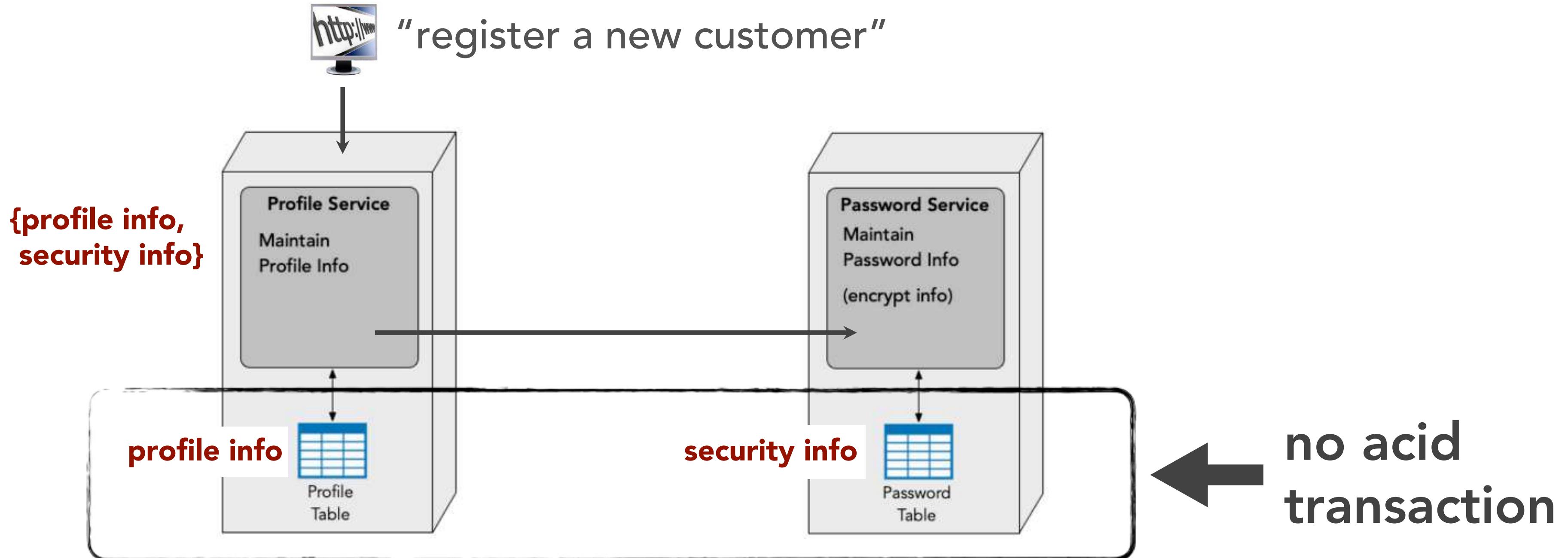
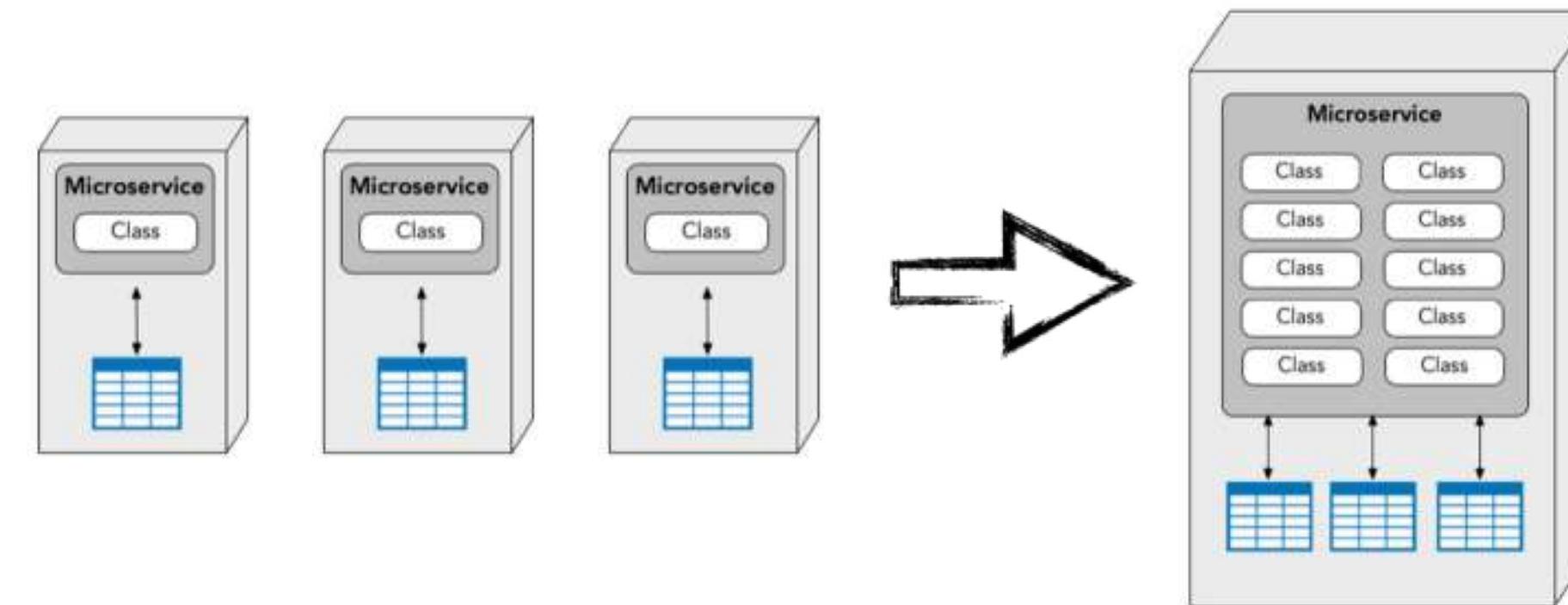
database
transactions



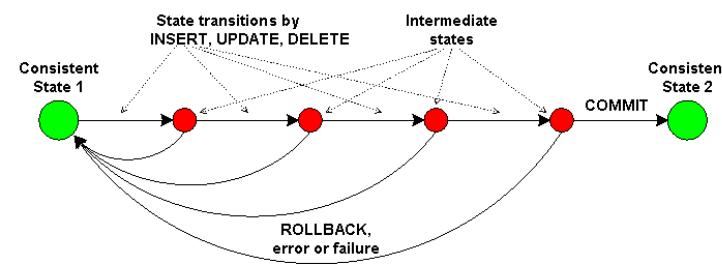
service granularity



database
transactions



service granularity

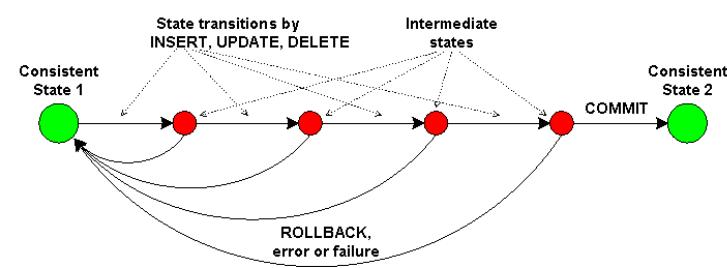


database
transactions



granularity factors
“what factors influence service granularity?”

service granularity



database
transactions

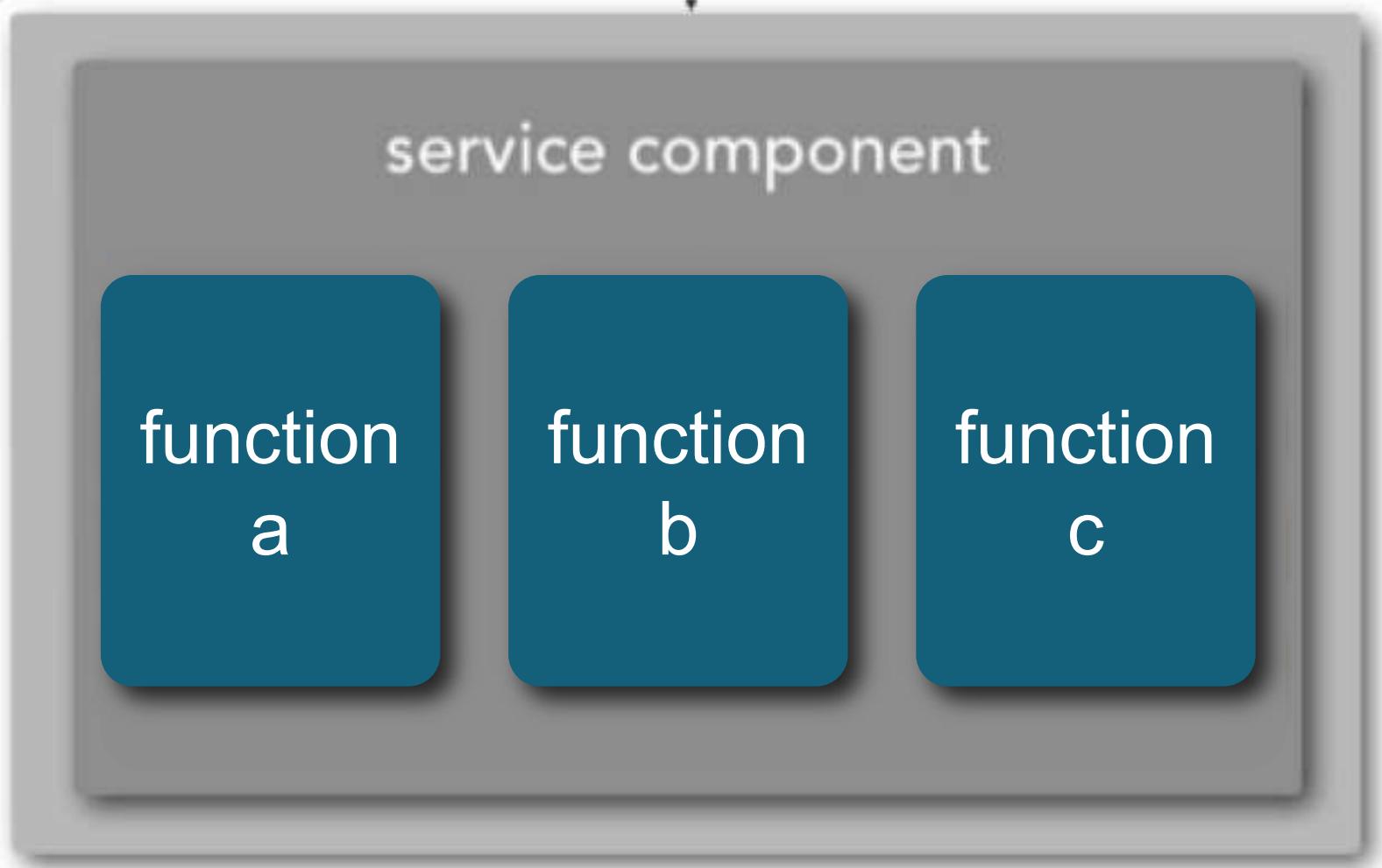
data
dependencies



granularity factors

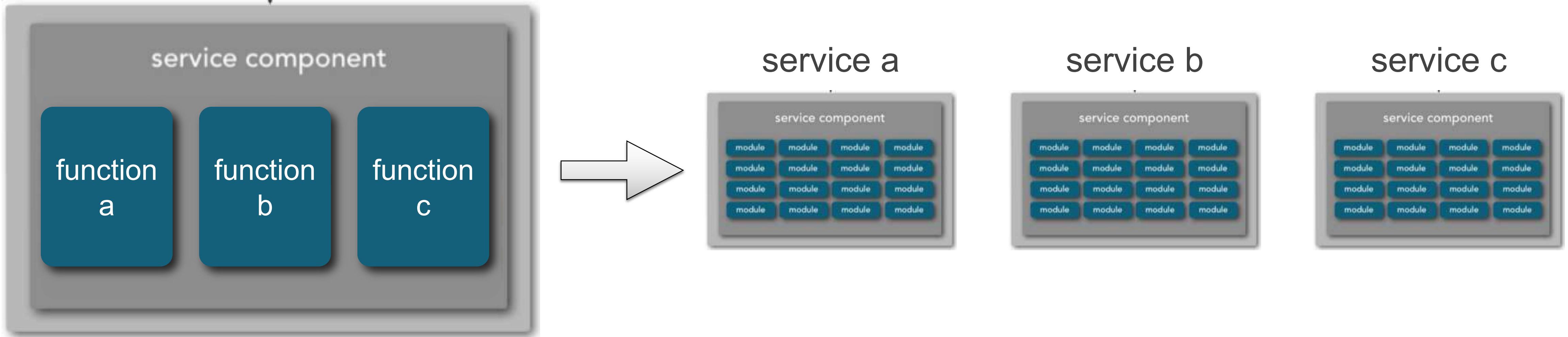
“what factors influence service granularity?”

service granularity



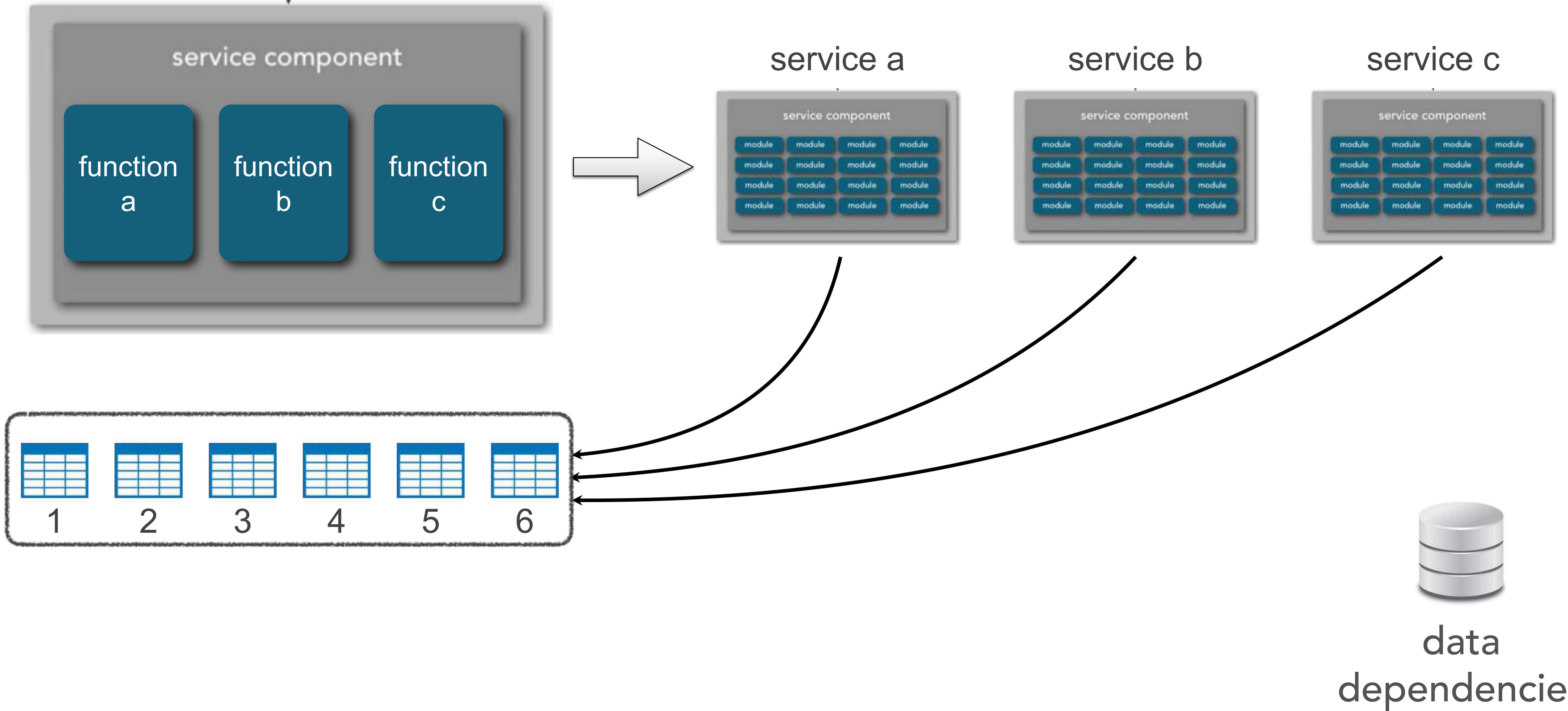
data
dependencies

service granularity

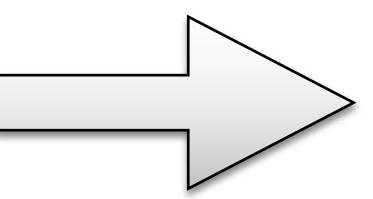
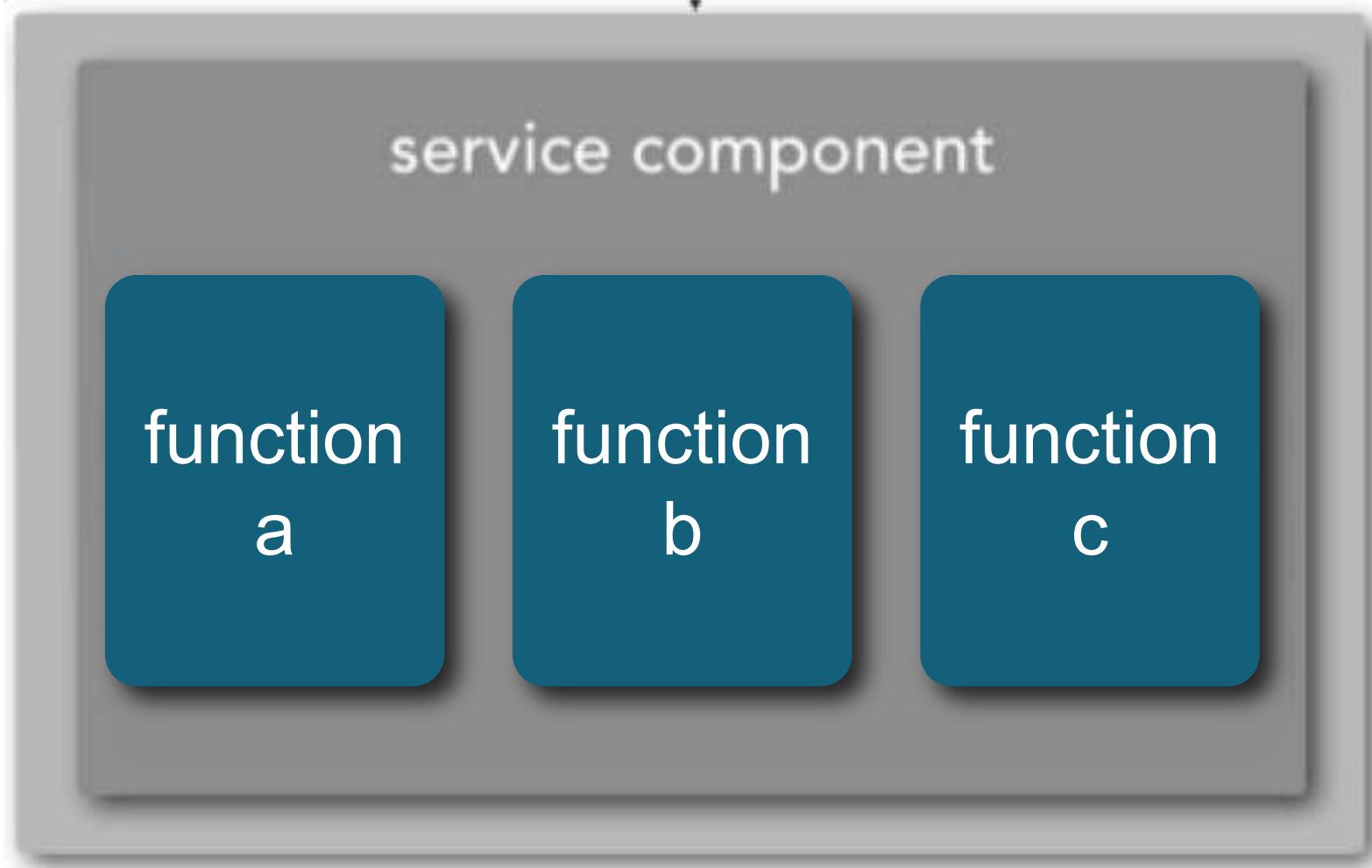


data
dependencies

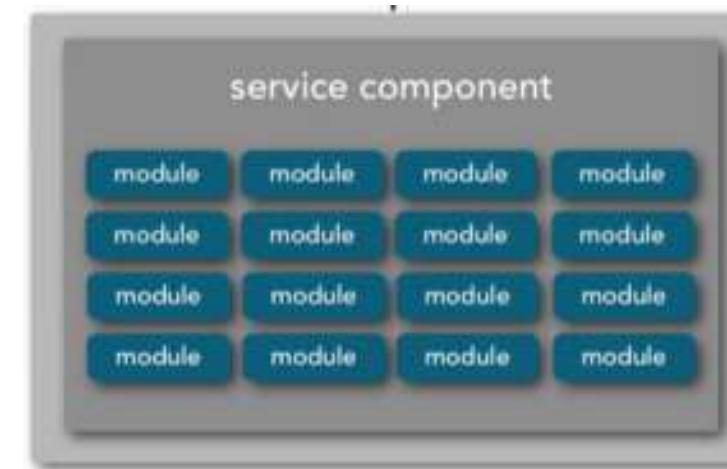
service granularity



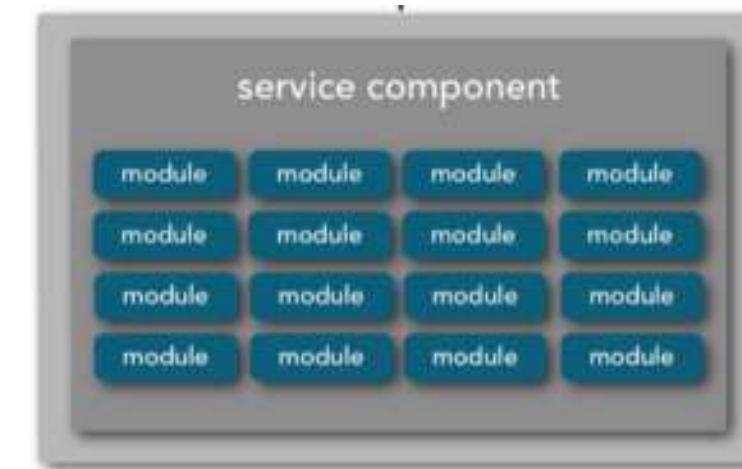
service granularity



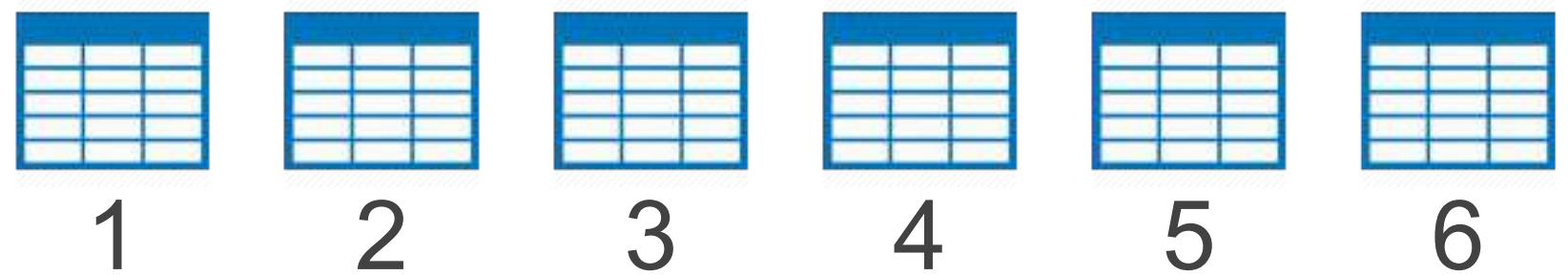
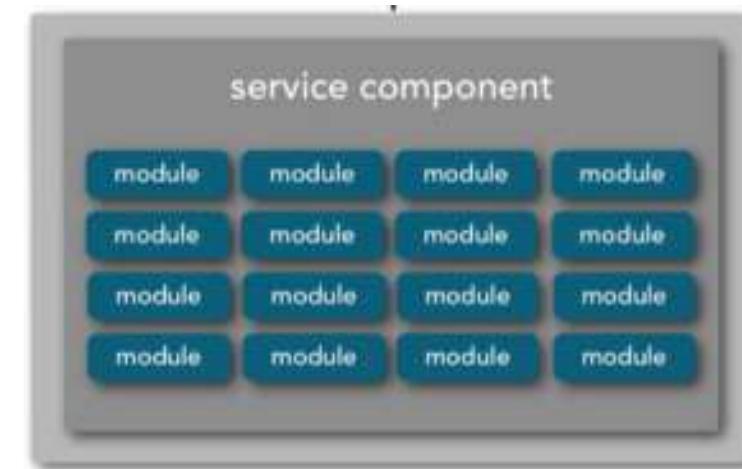
service a



service b

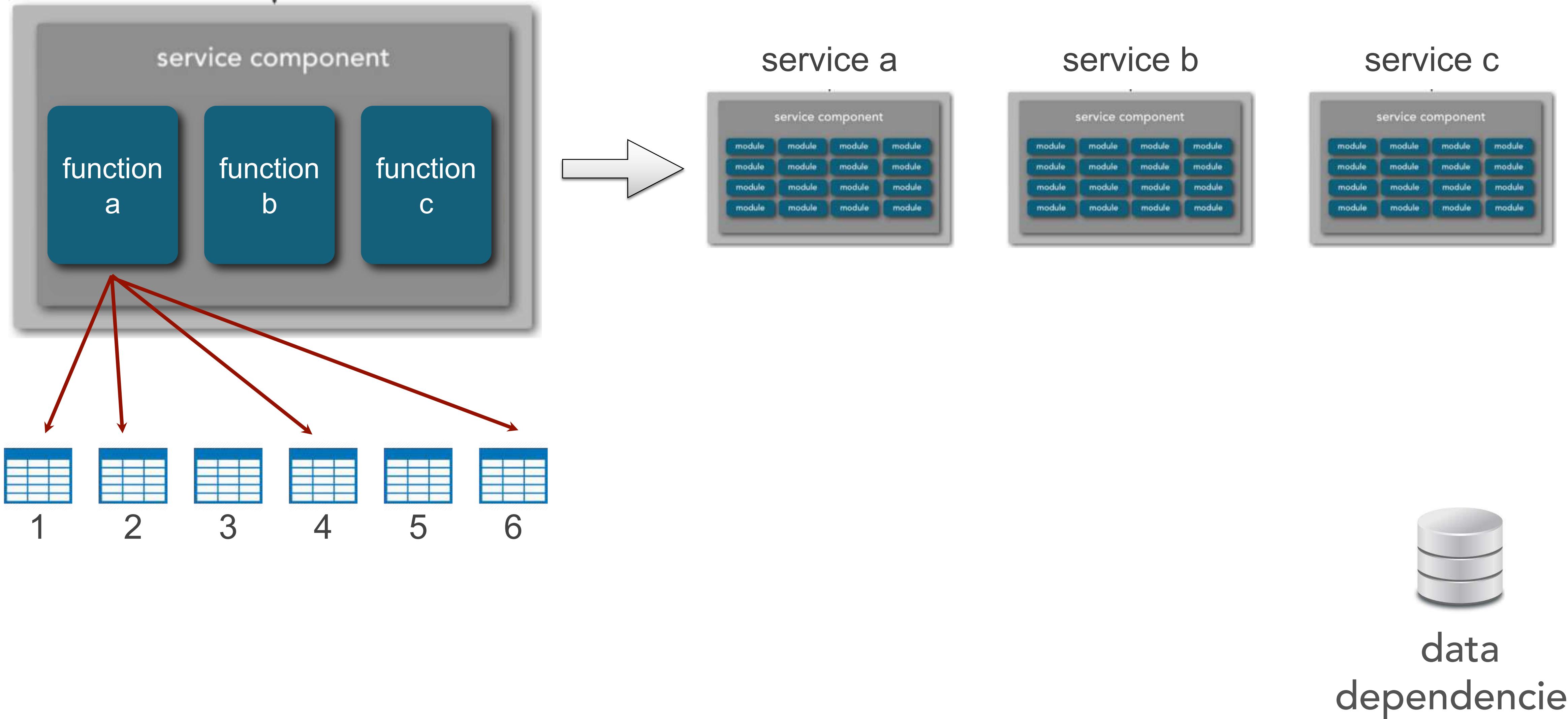


service c

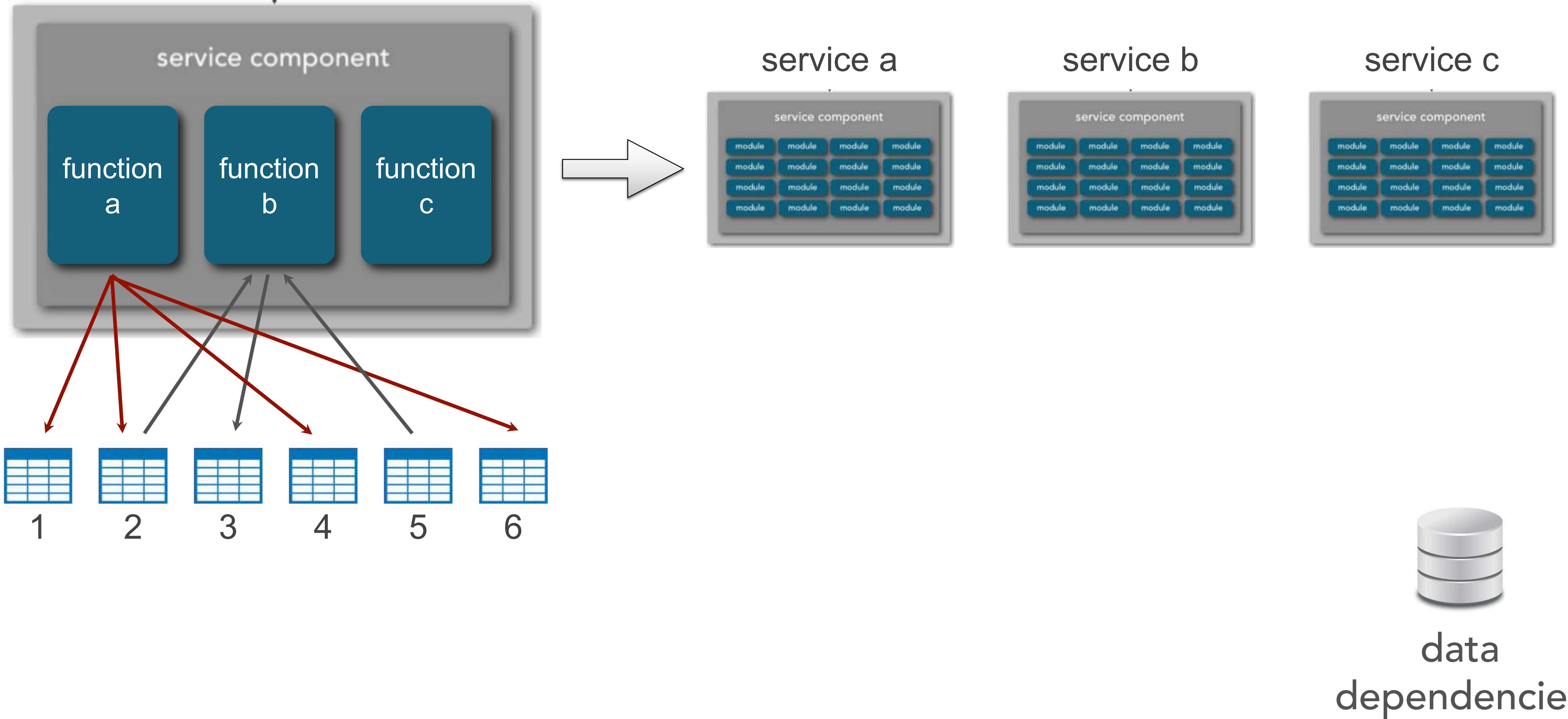


data
dependencies

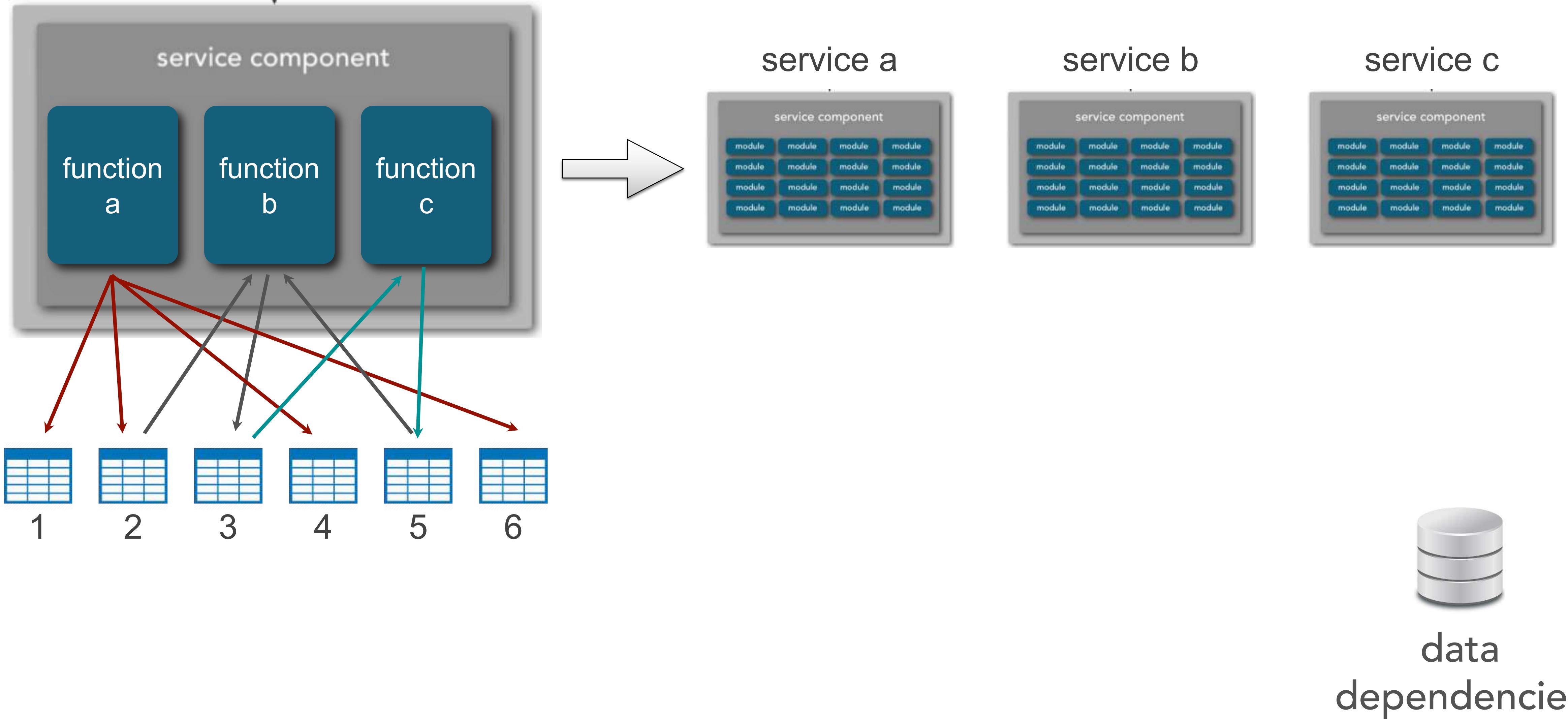
service granularity



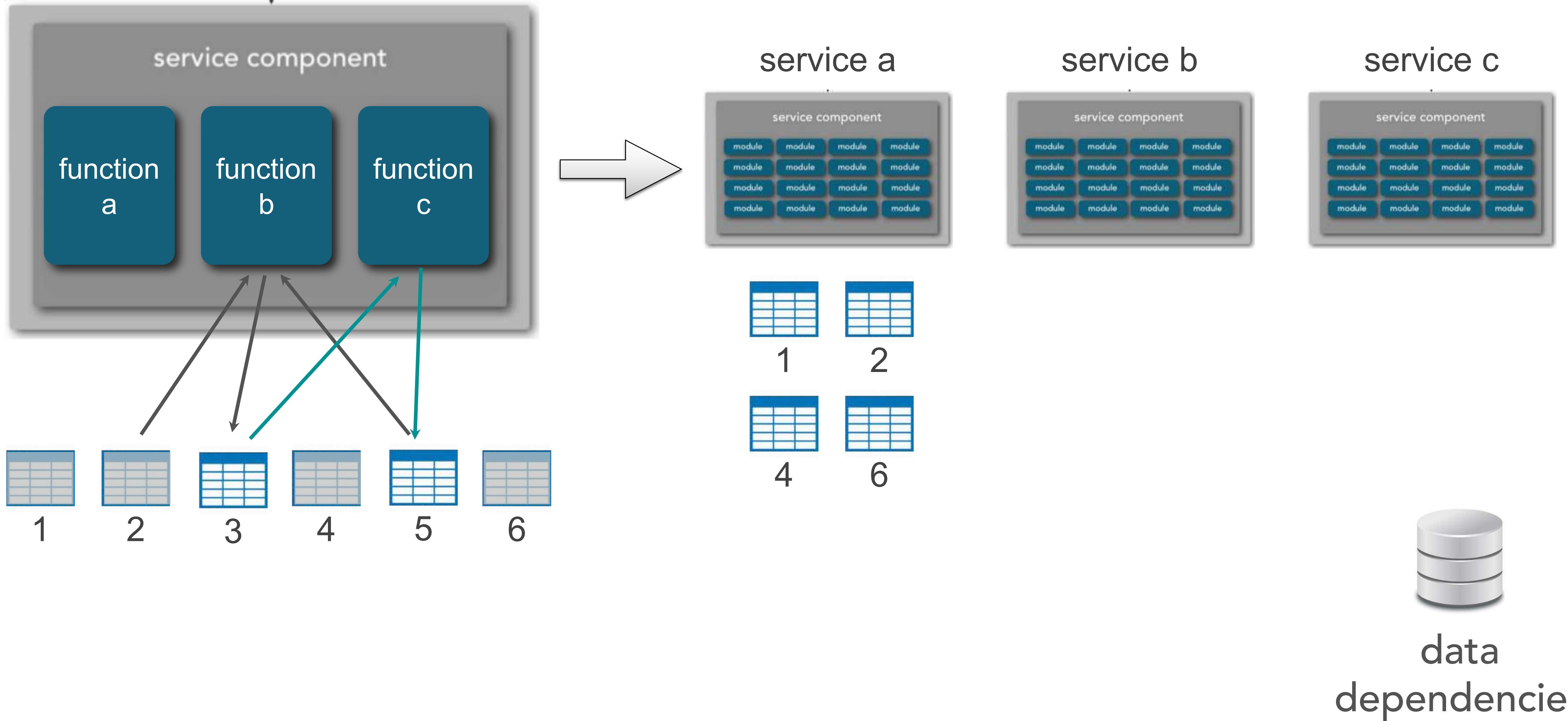
service granularity



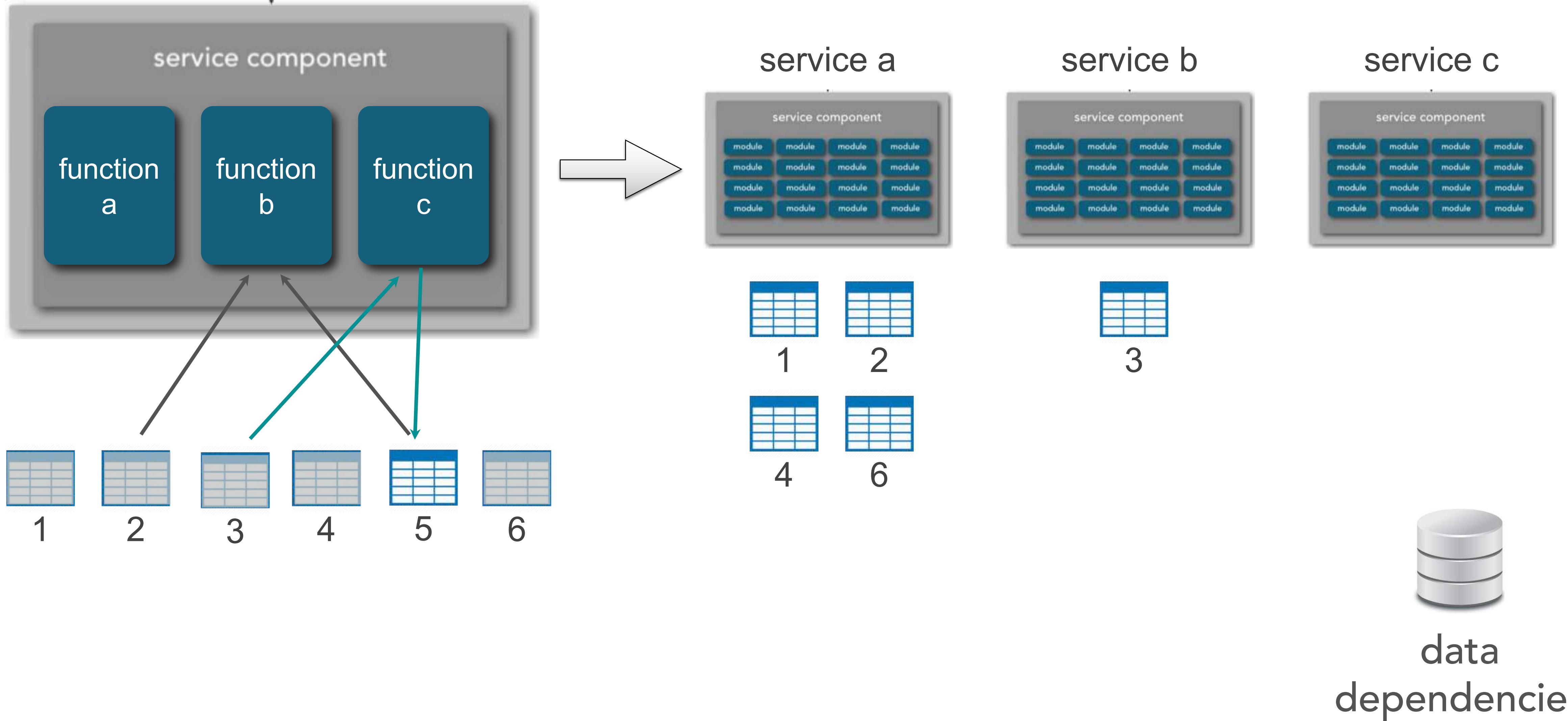
service granularity



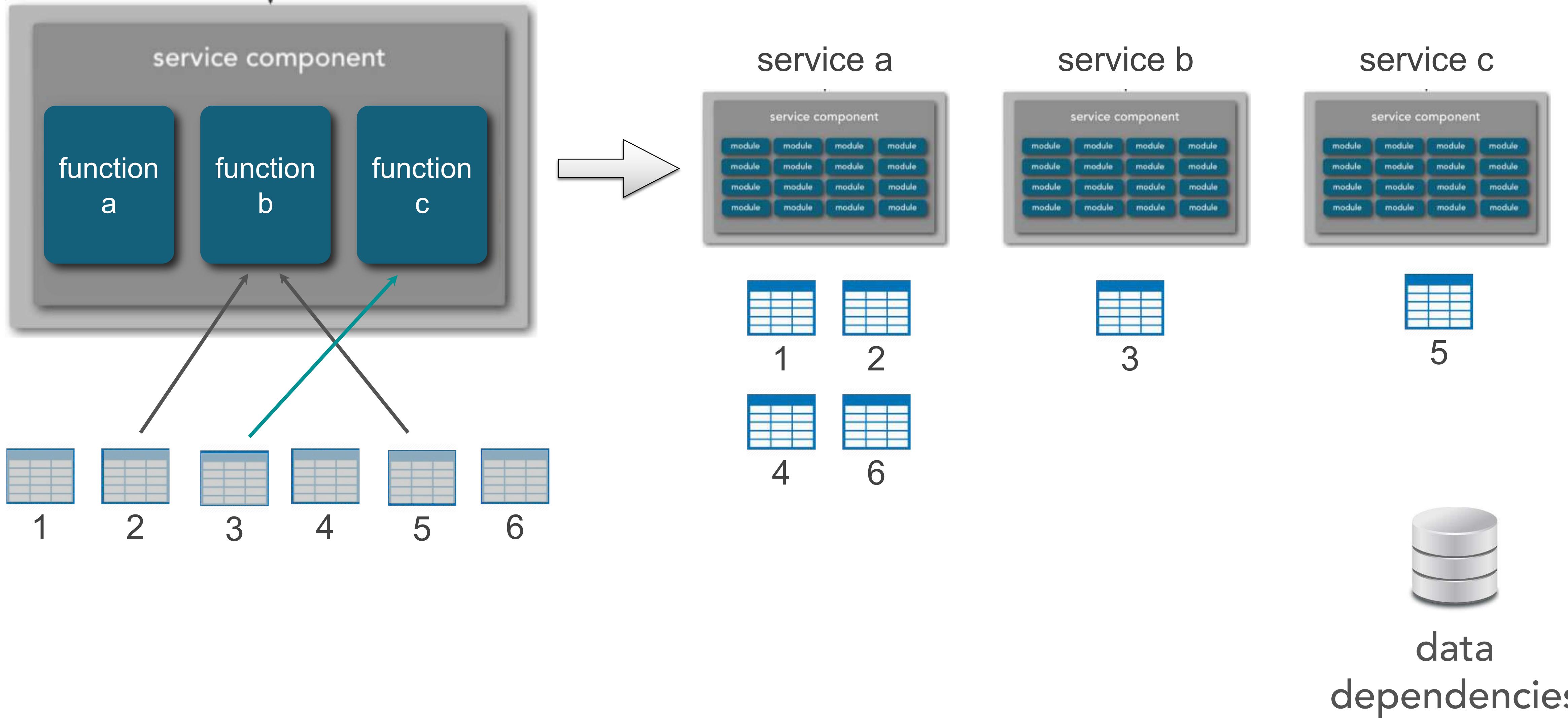
service granularity



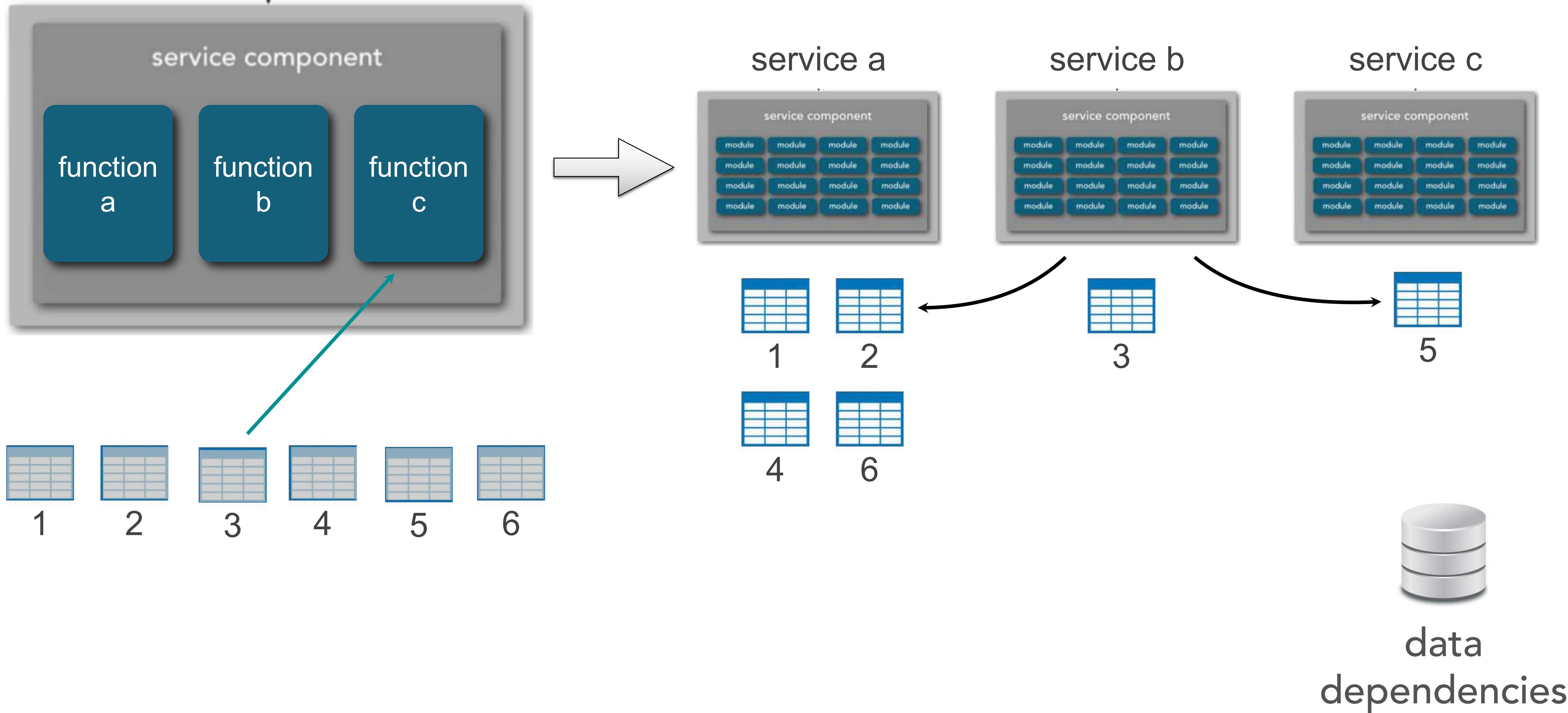
service granularity



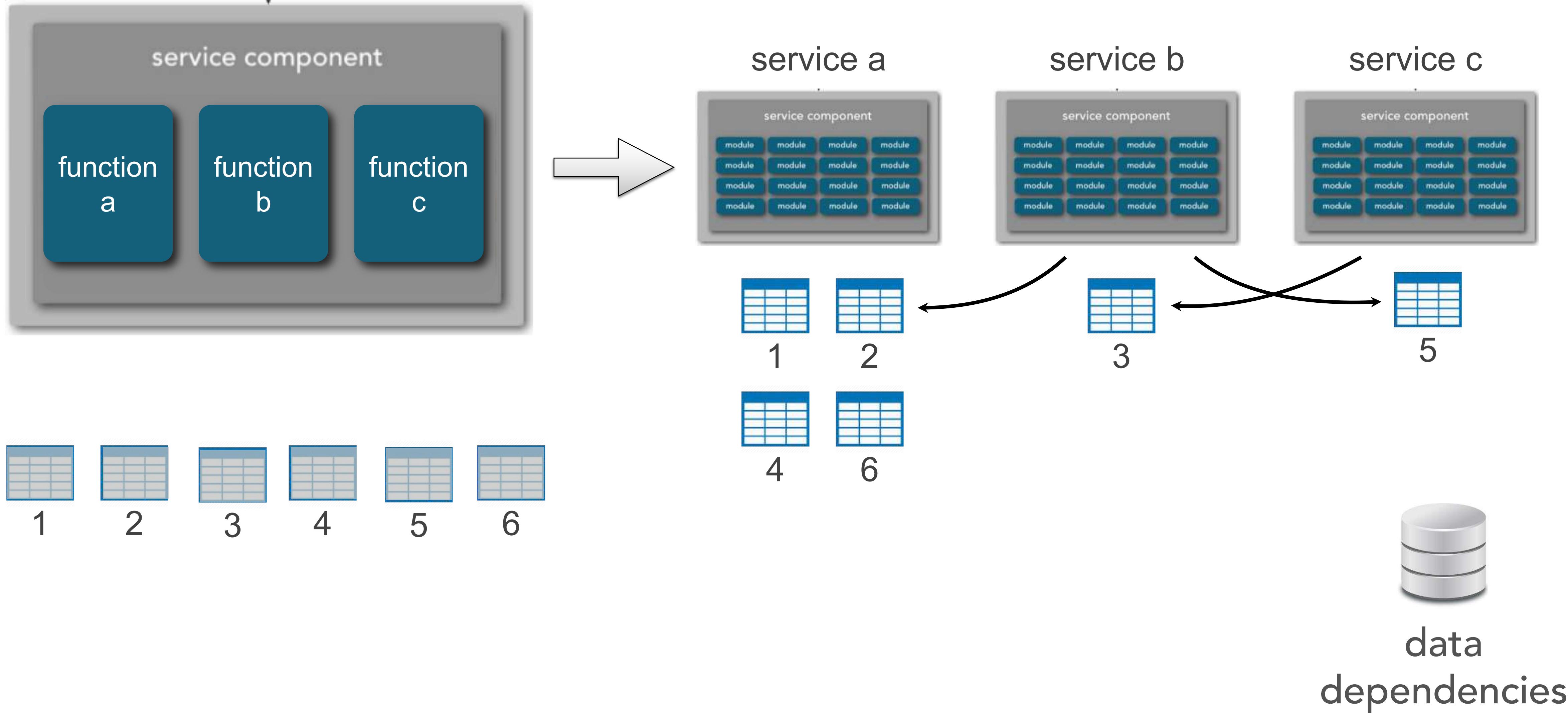
service granularity



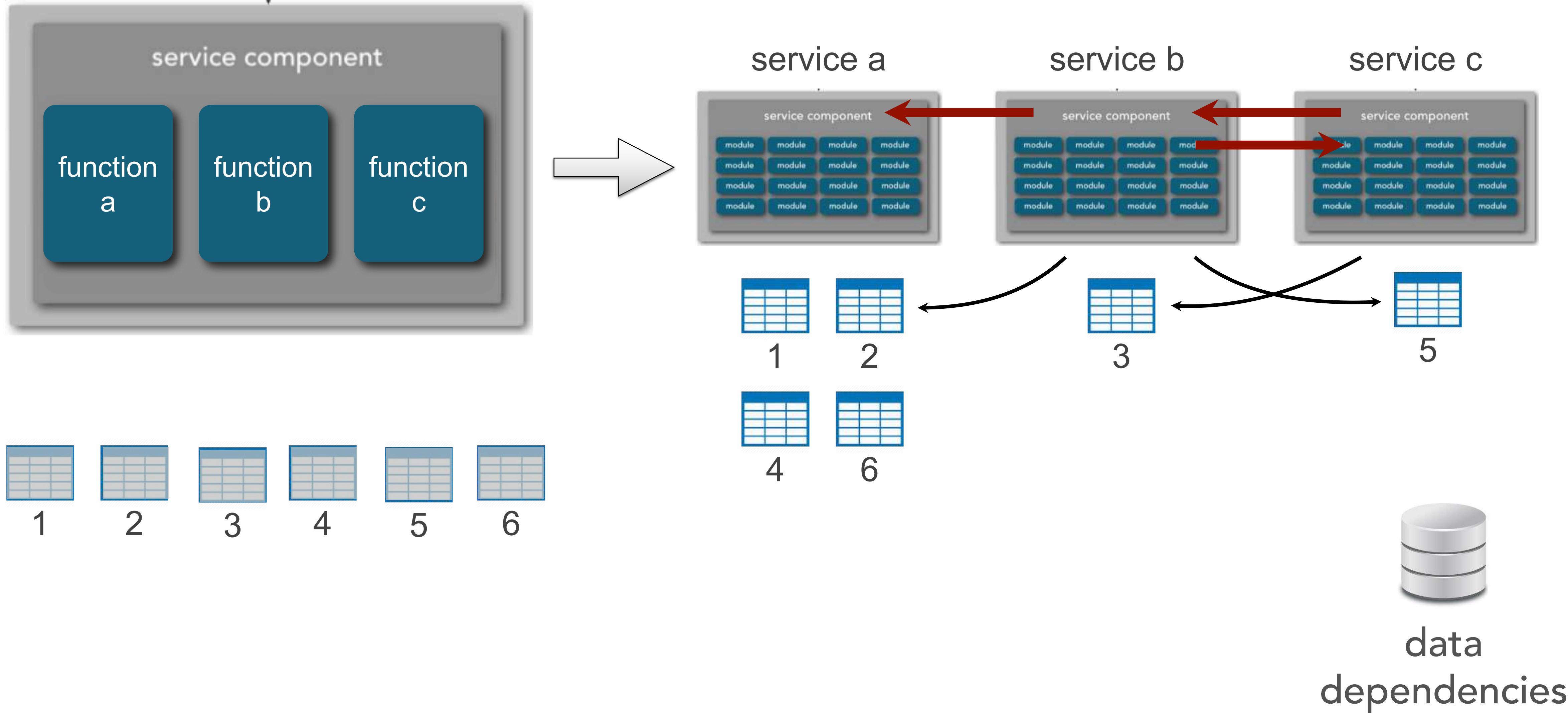
service granularity



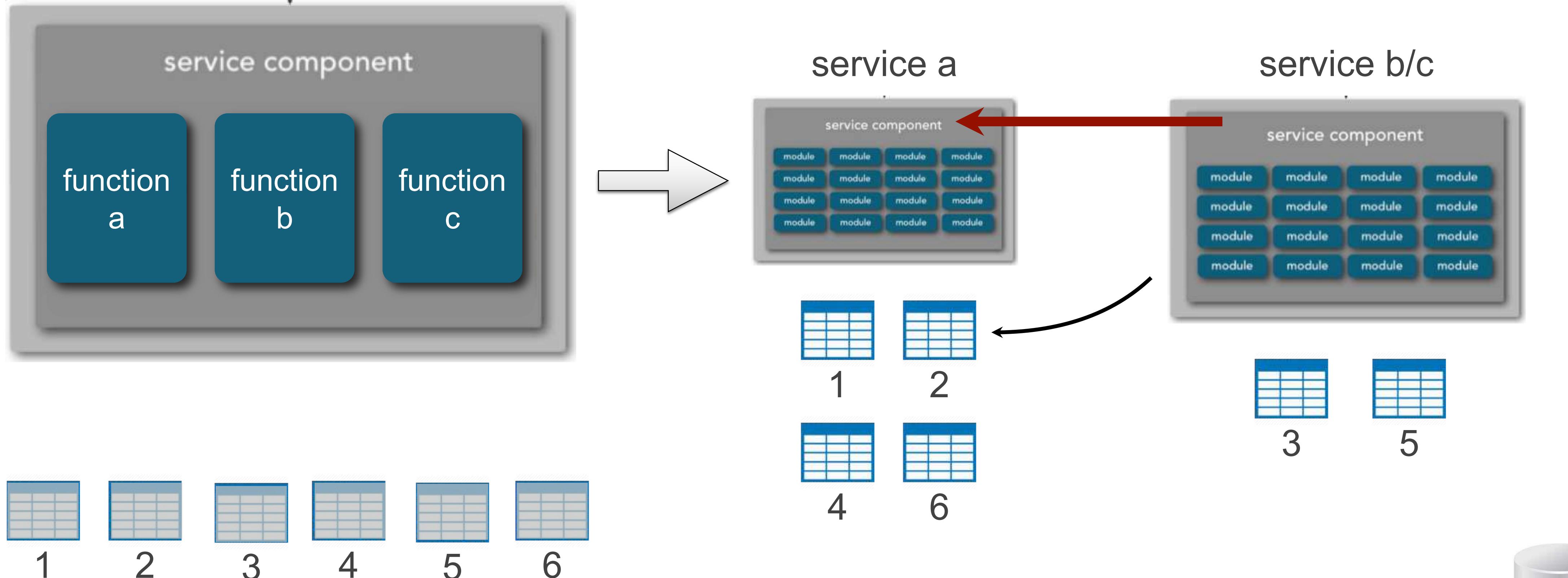
service granularity



service granularity

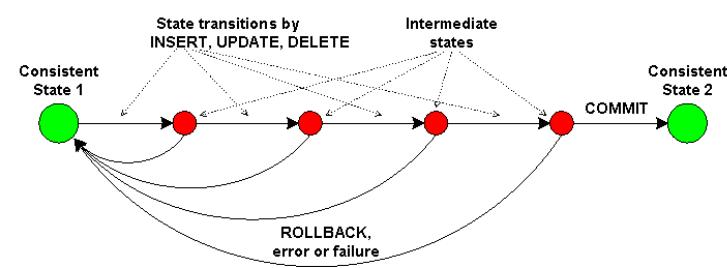


service granularity



data
dependencies

service granularity



database
transactions

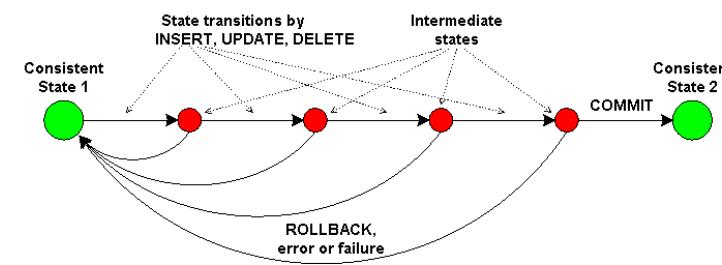
data
dependencies



granularity factors

“what factors influence service granularity?”

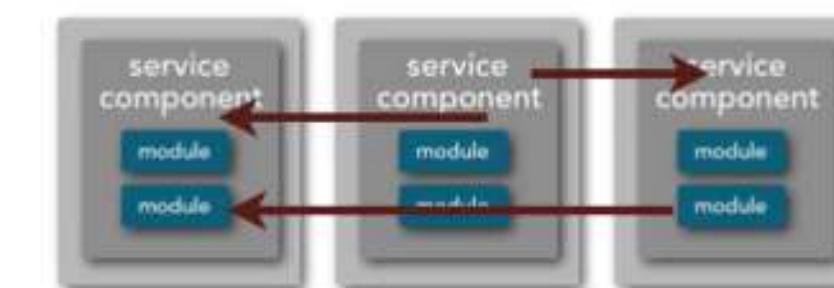
service granularity



database
transactions



data
dependencies



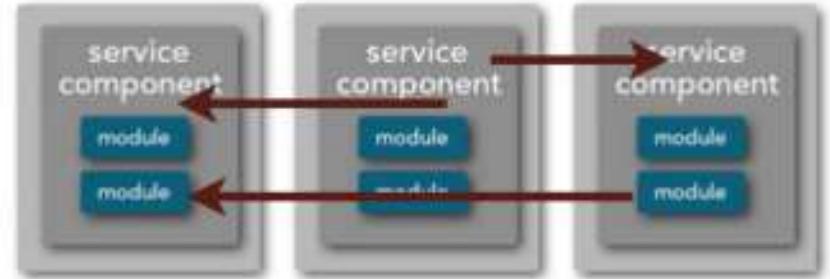
workflow and
choreography



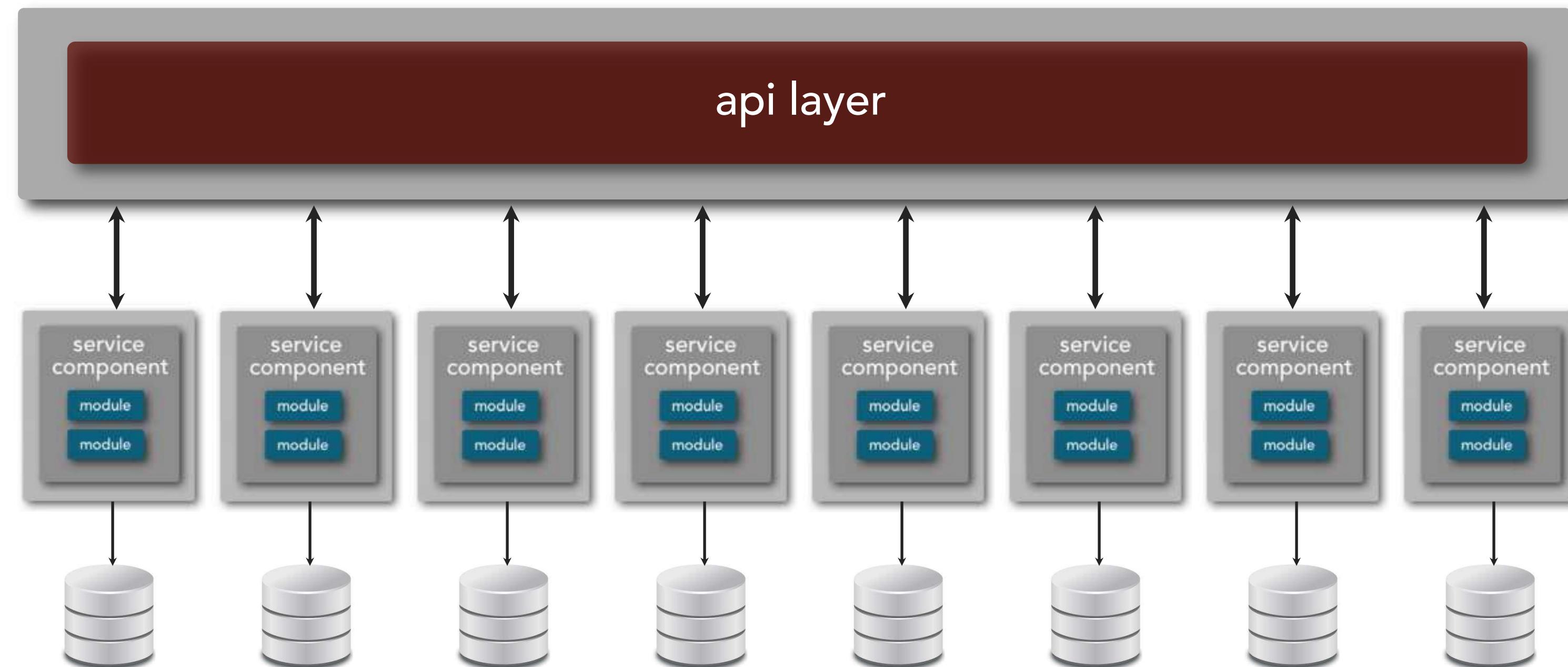
granularity factors

“what factors influence service granularity?”

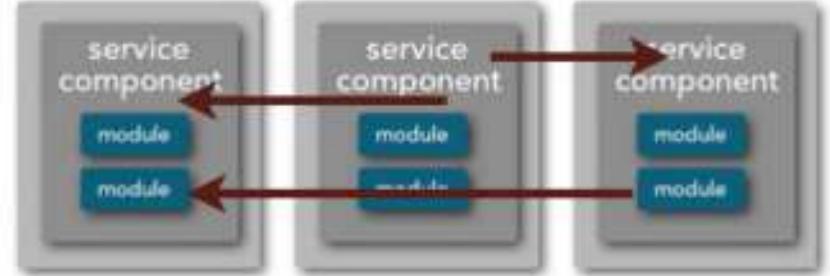
service granularity



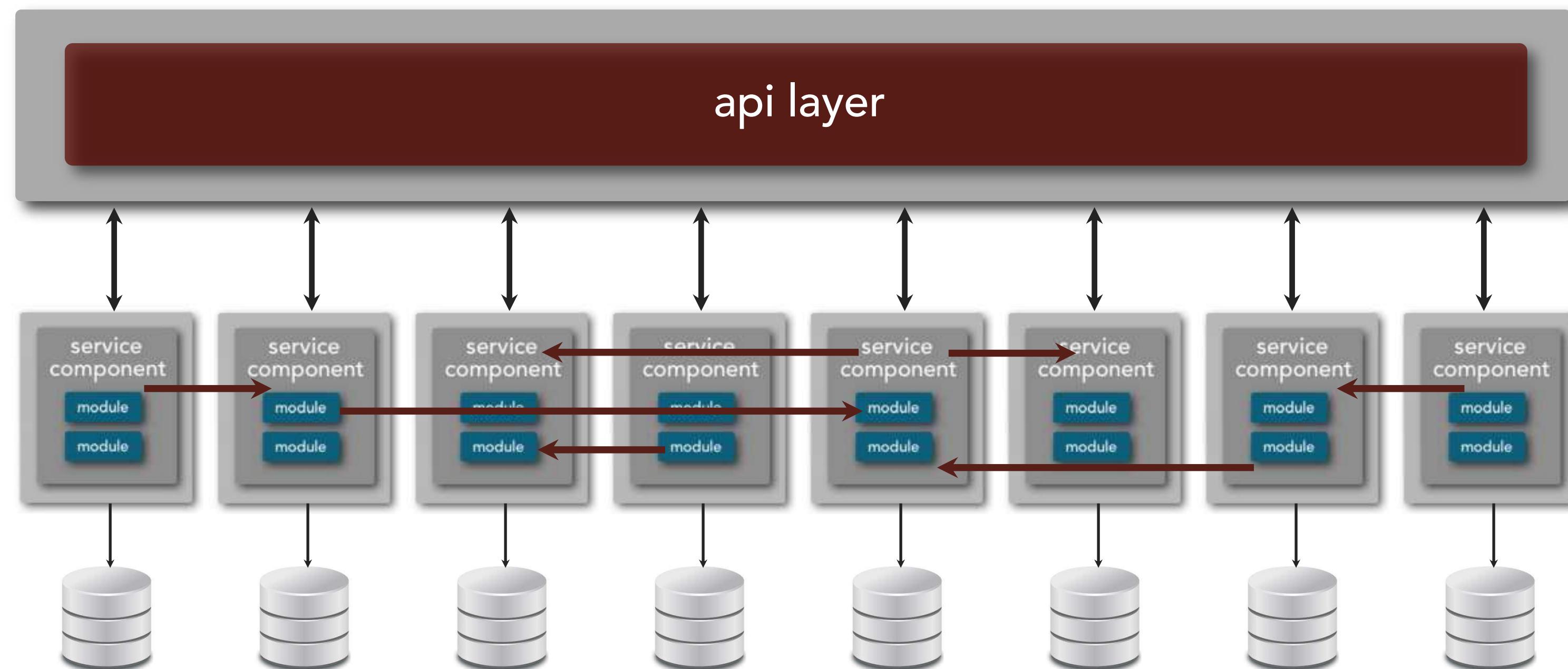
workflow and
choreography



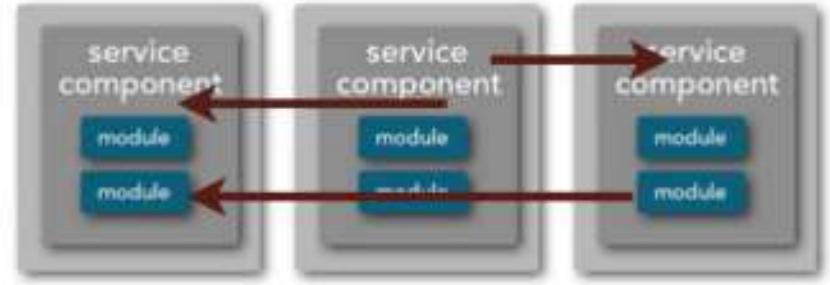
service granularity



workflow and
choreography

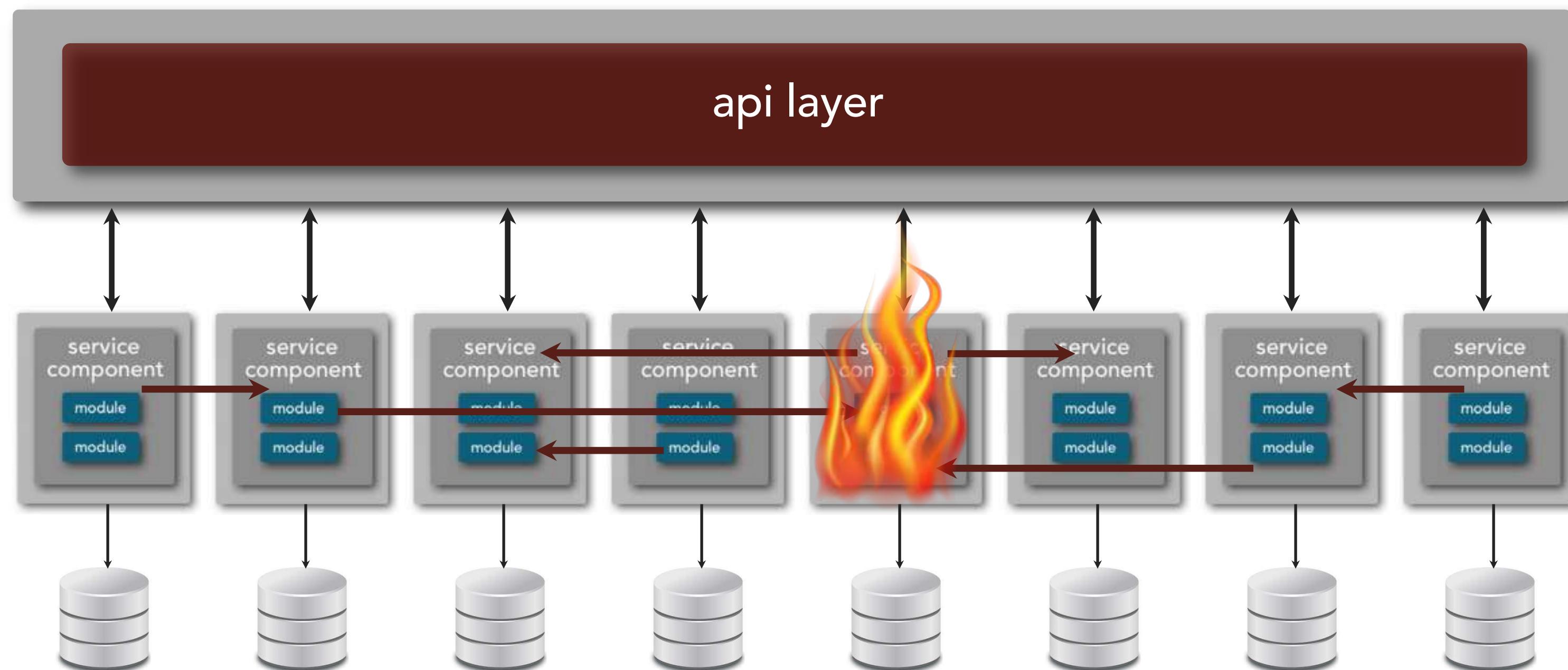


service granularity

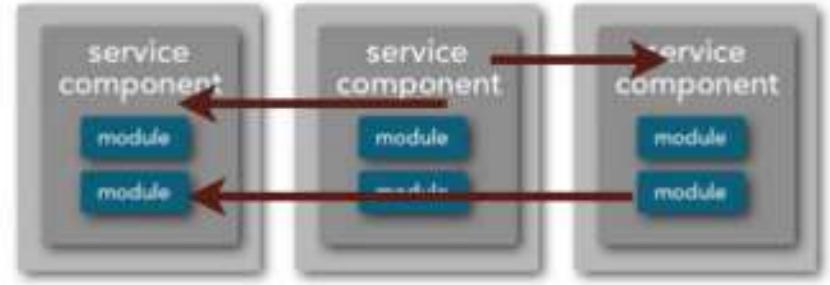


workflow and
choreography

fault tolerance (availability)

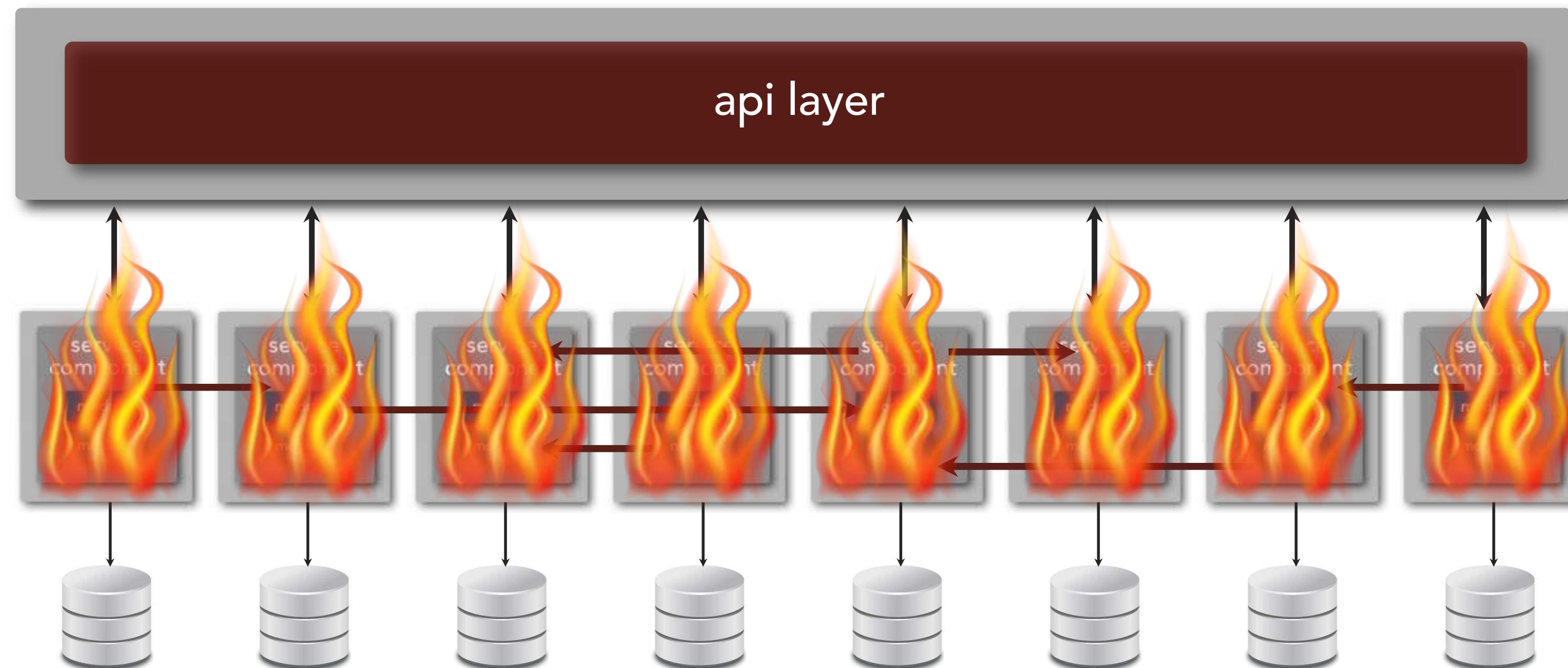


service granularity

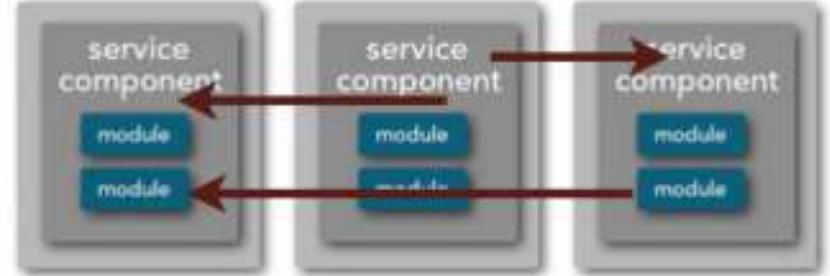


workflow and
choreography

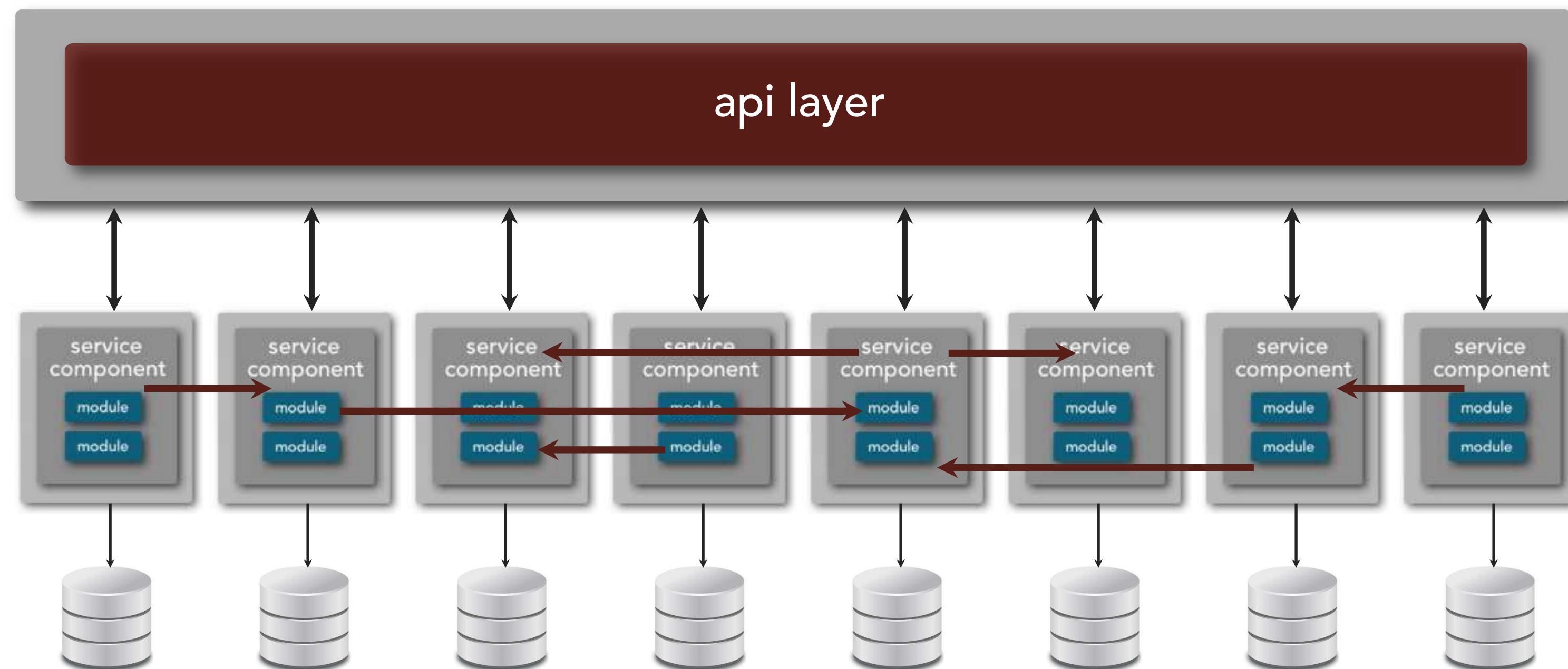
fault tolerance (availability)



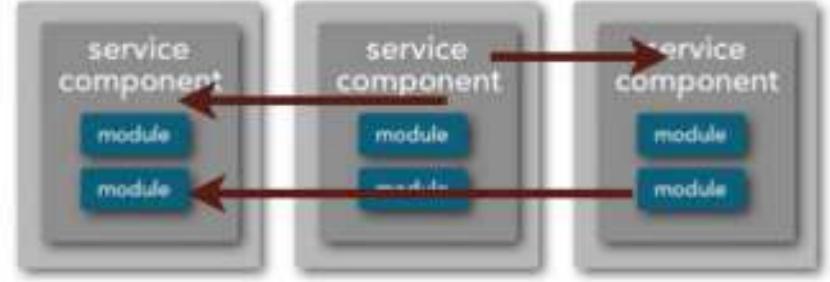
service granularity



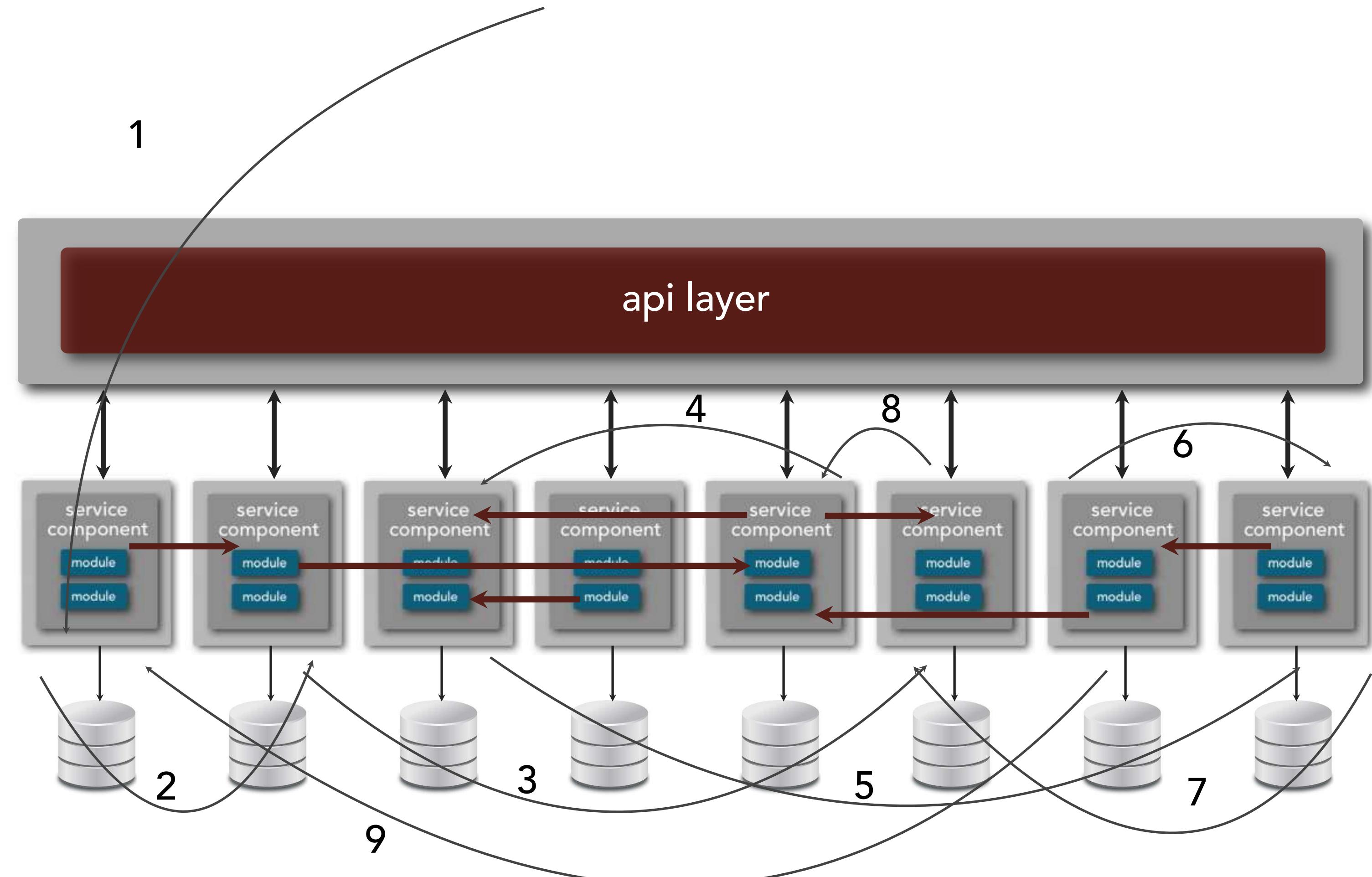
workflow and
choreography



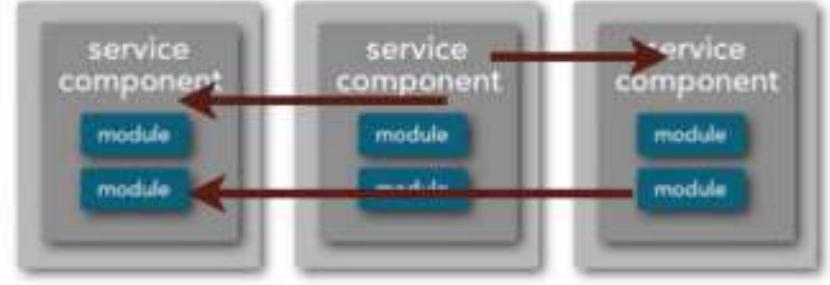
service granularity



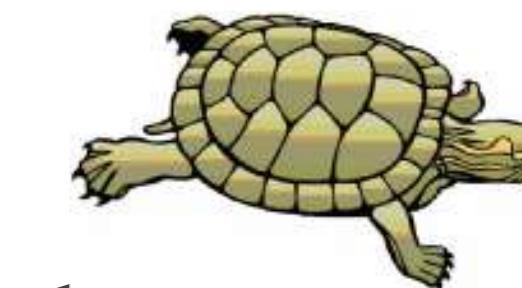
workflow and
choreography



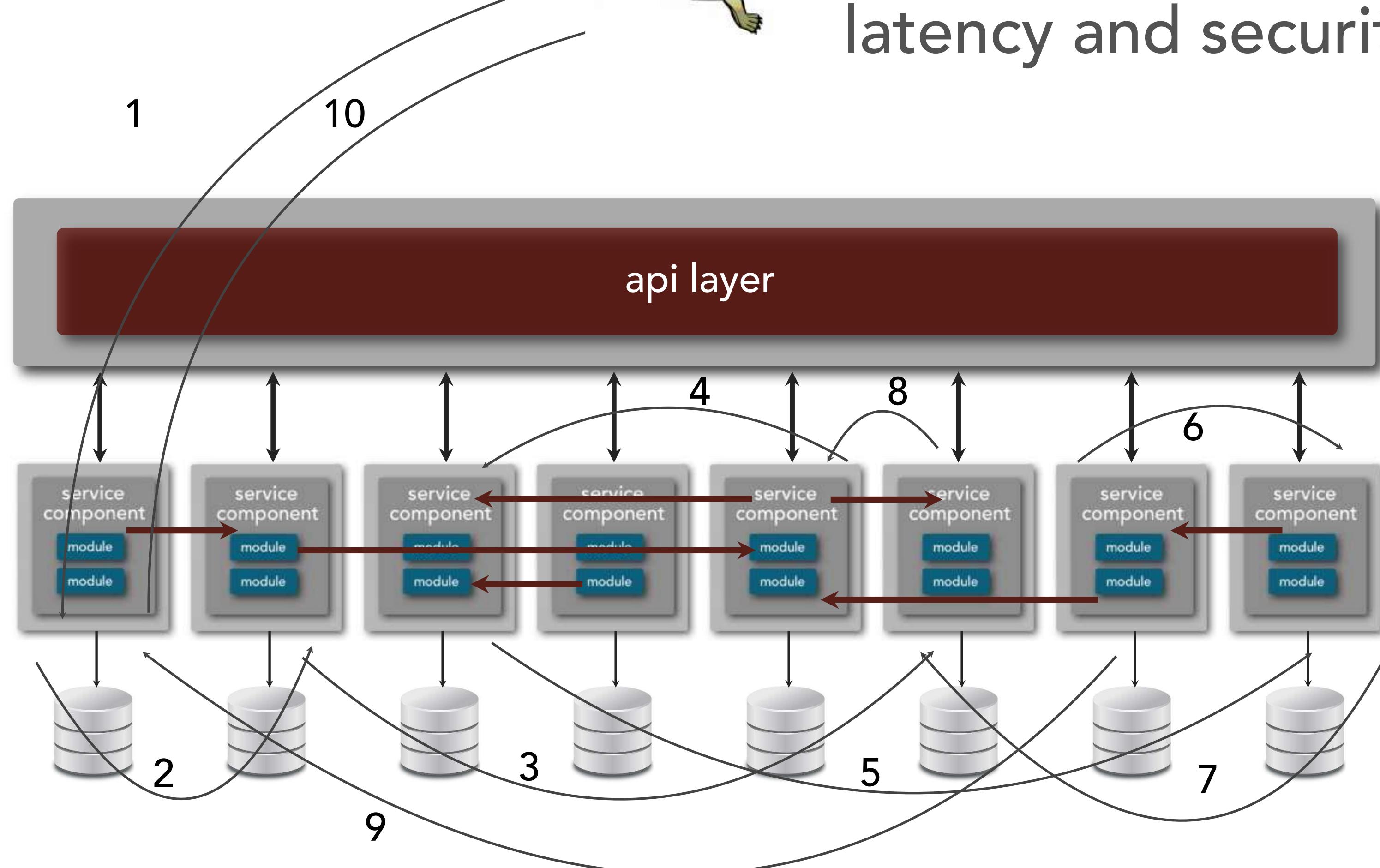
service granularity



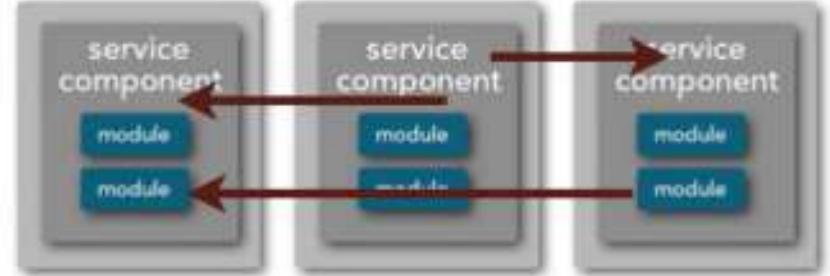
workflow and
choreography



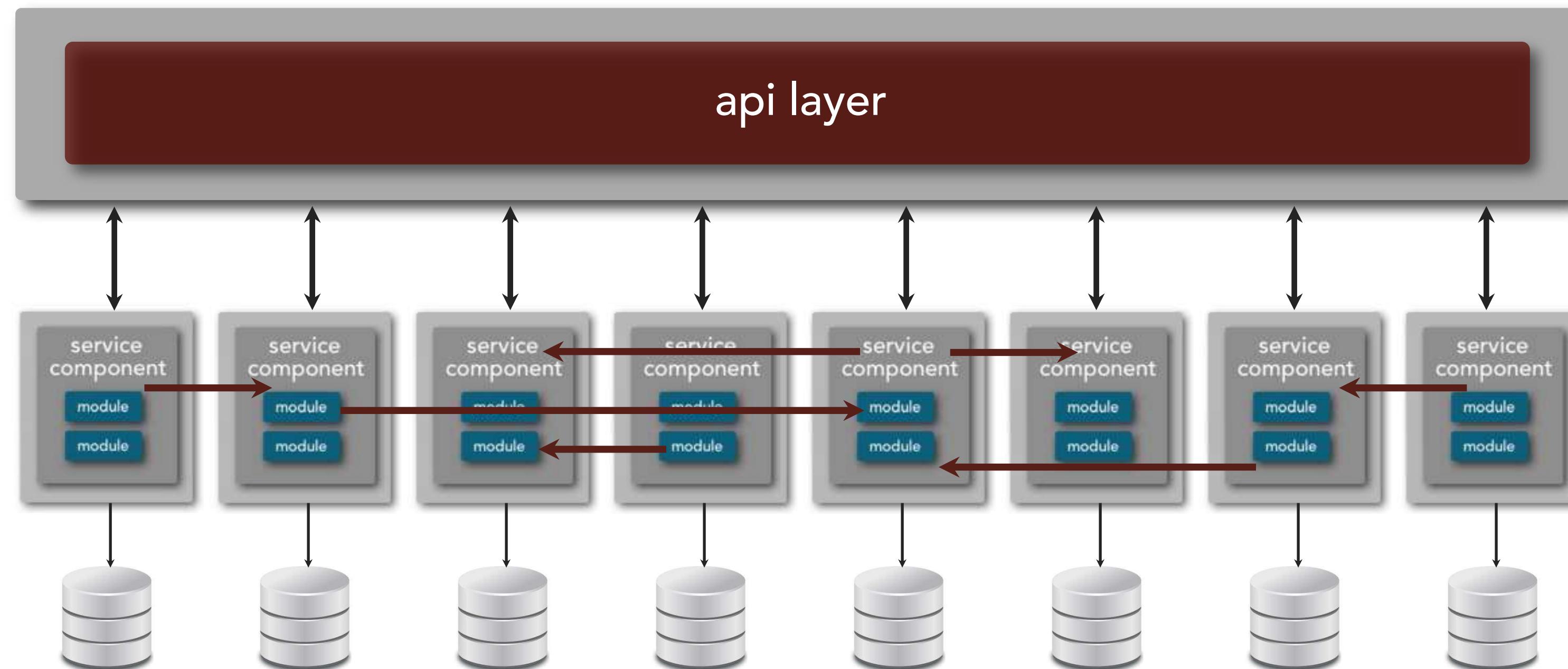
slow performance due to
latency and security



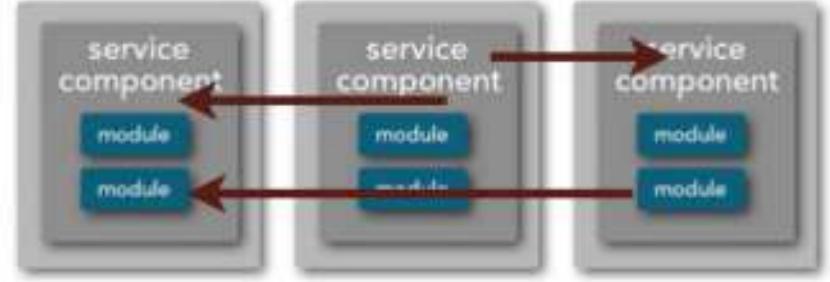
service granularity



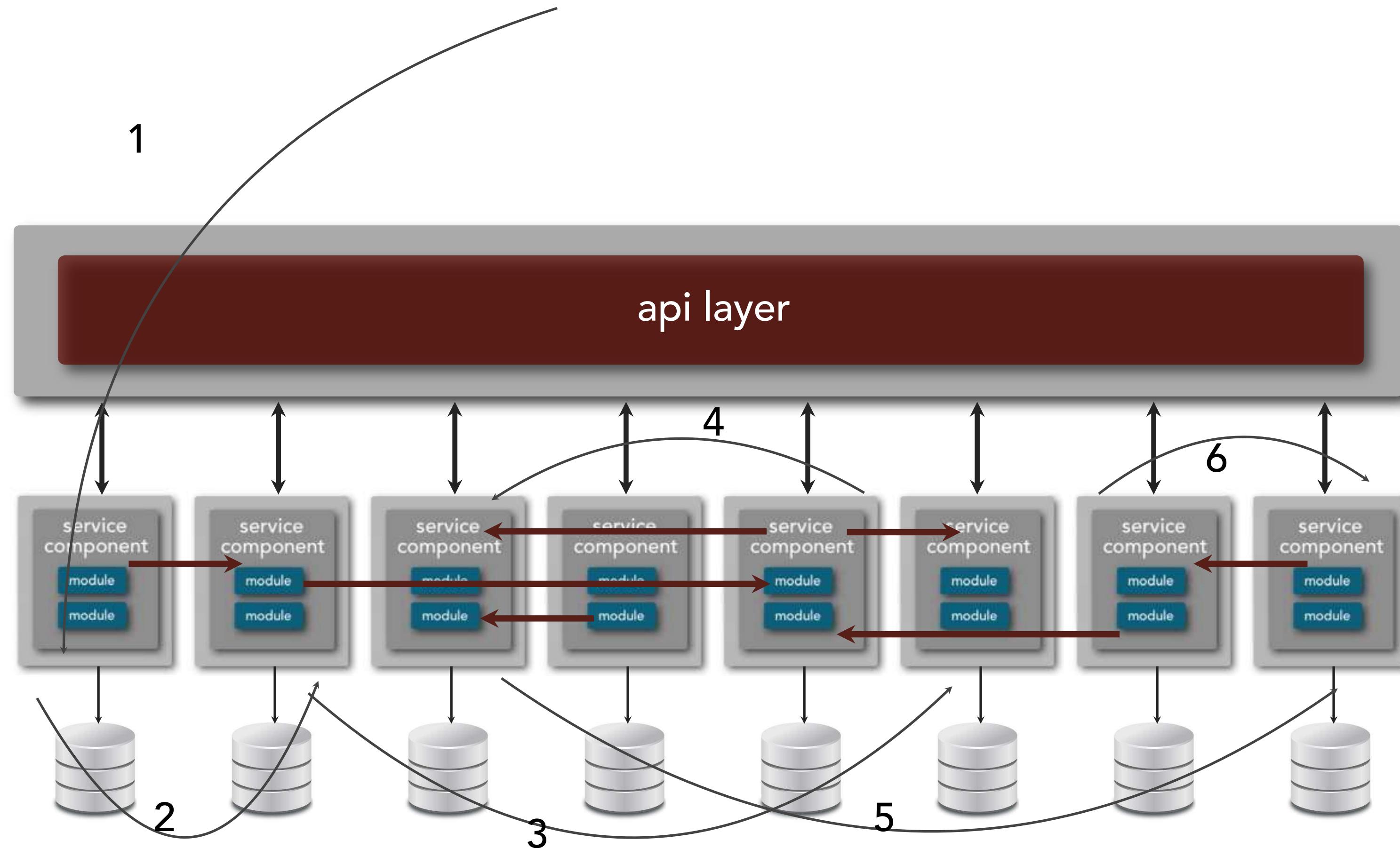
workflow and
choreography



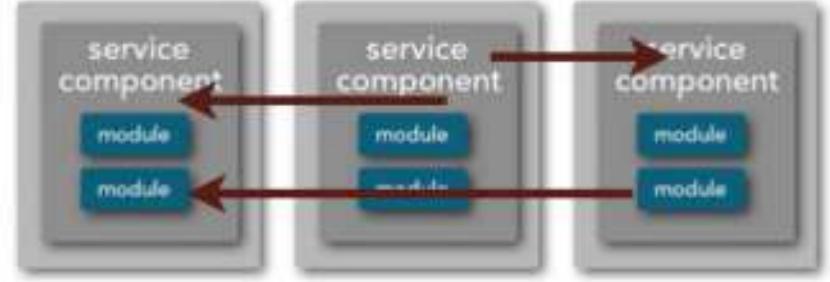
service granularity



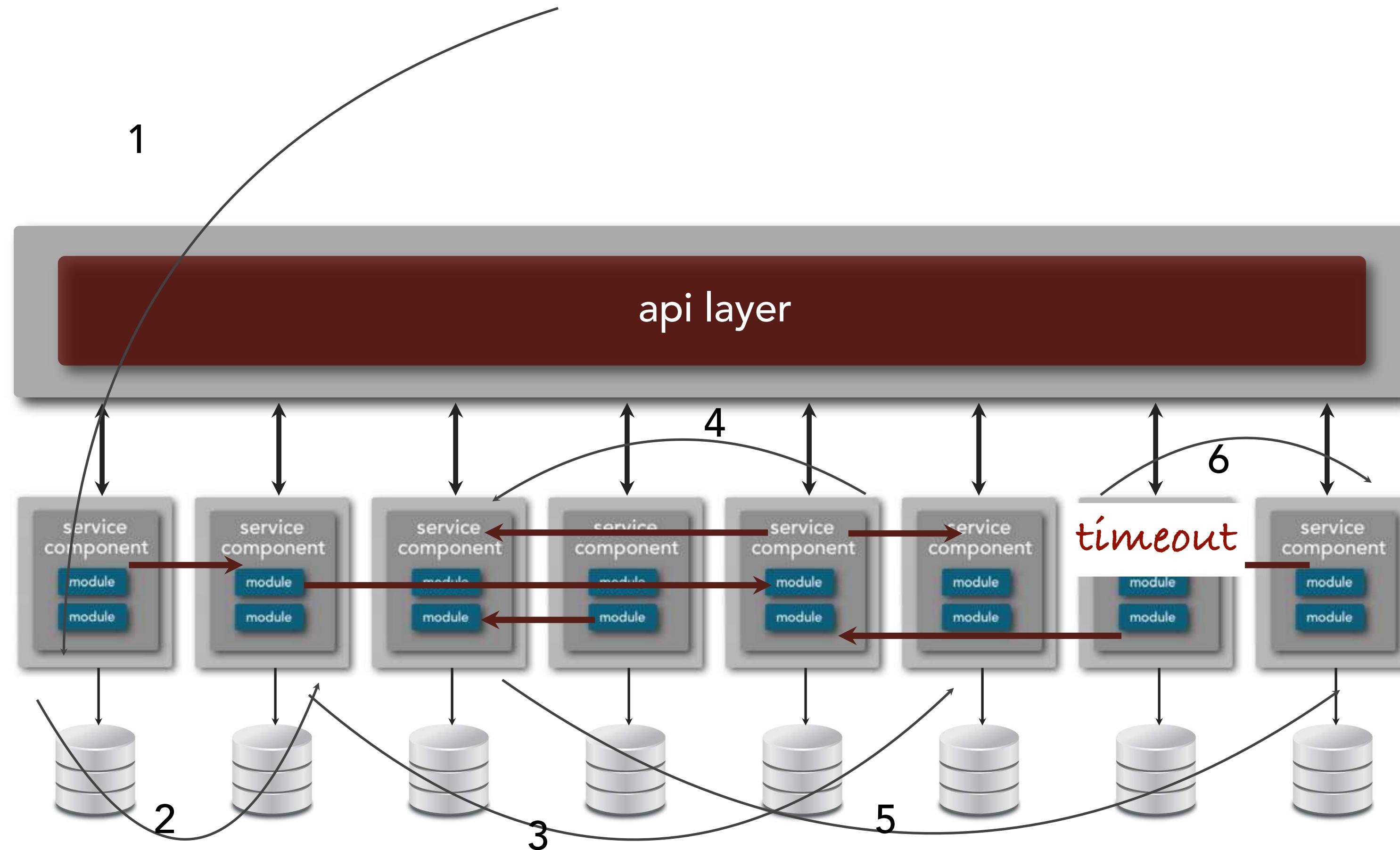
workflow and
choreography



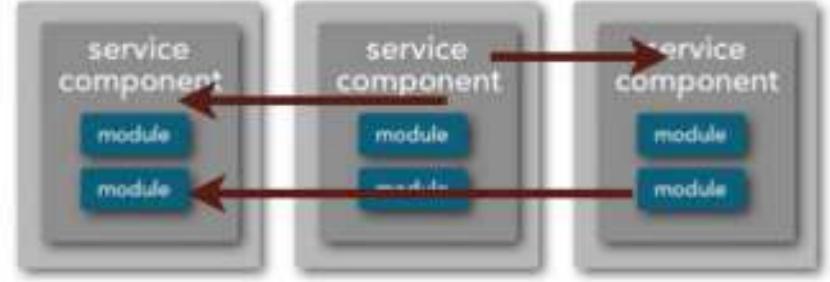
service granularity



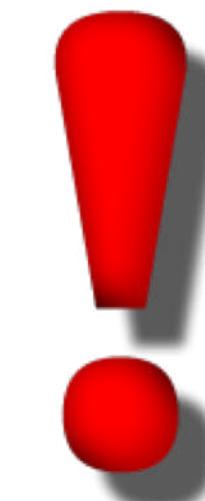
workflow and
choreography



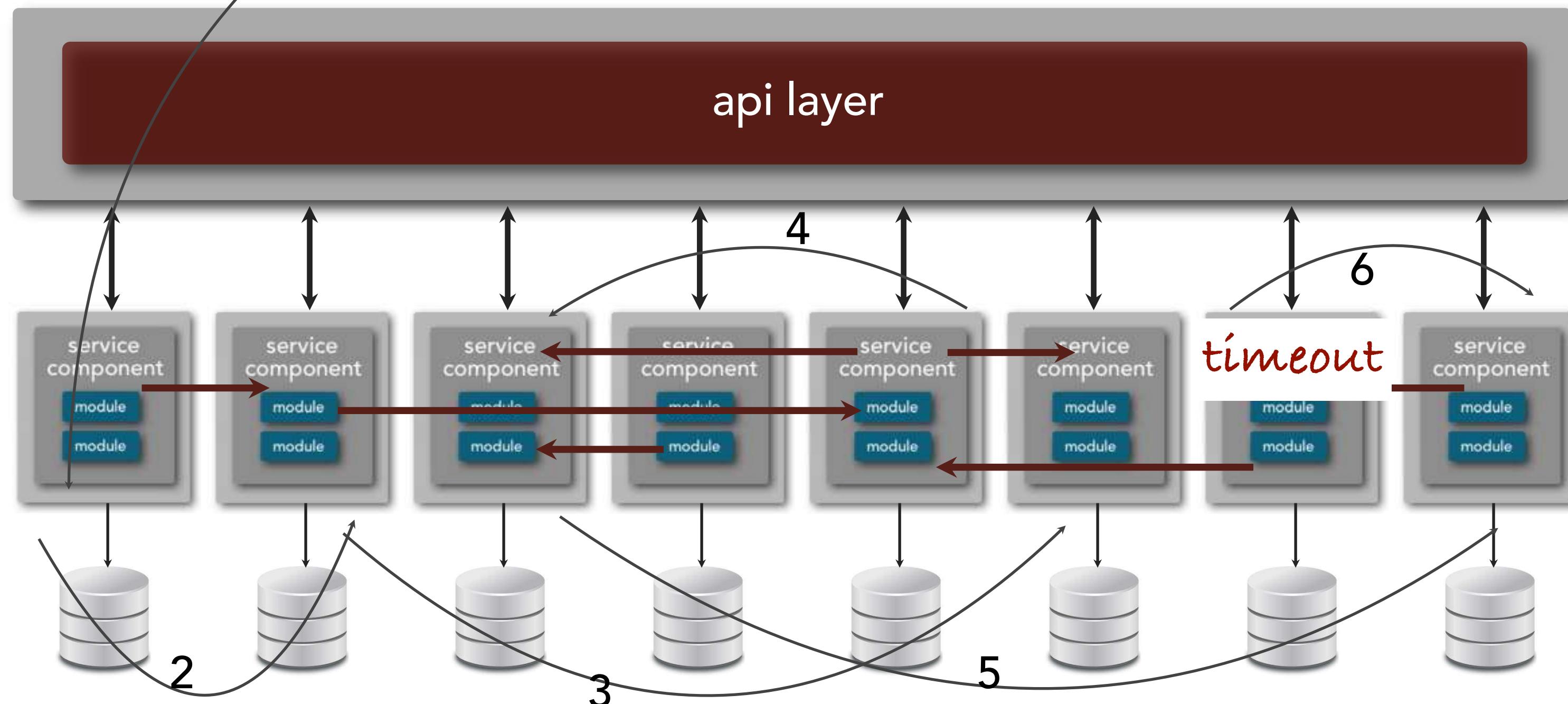
service granularity



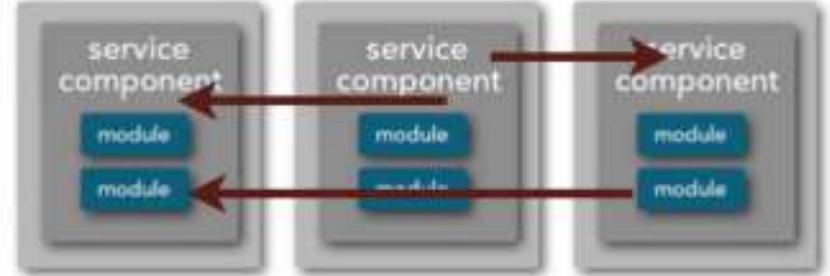
workflow and
choreography



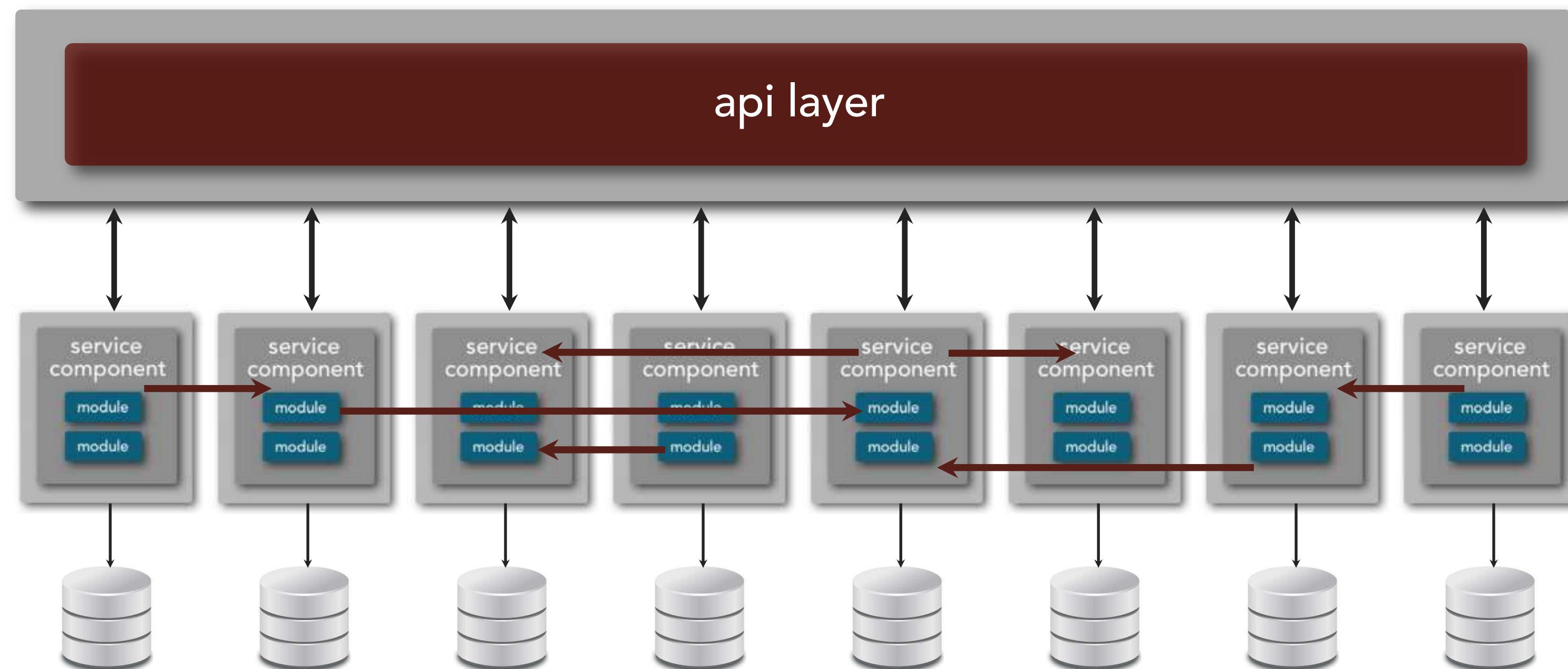
reliability and data consistency



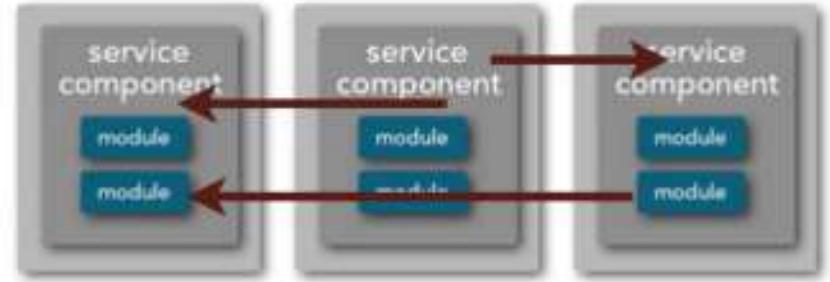
service granularity



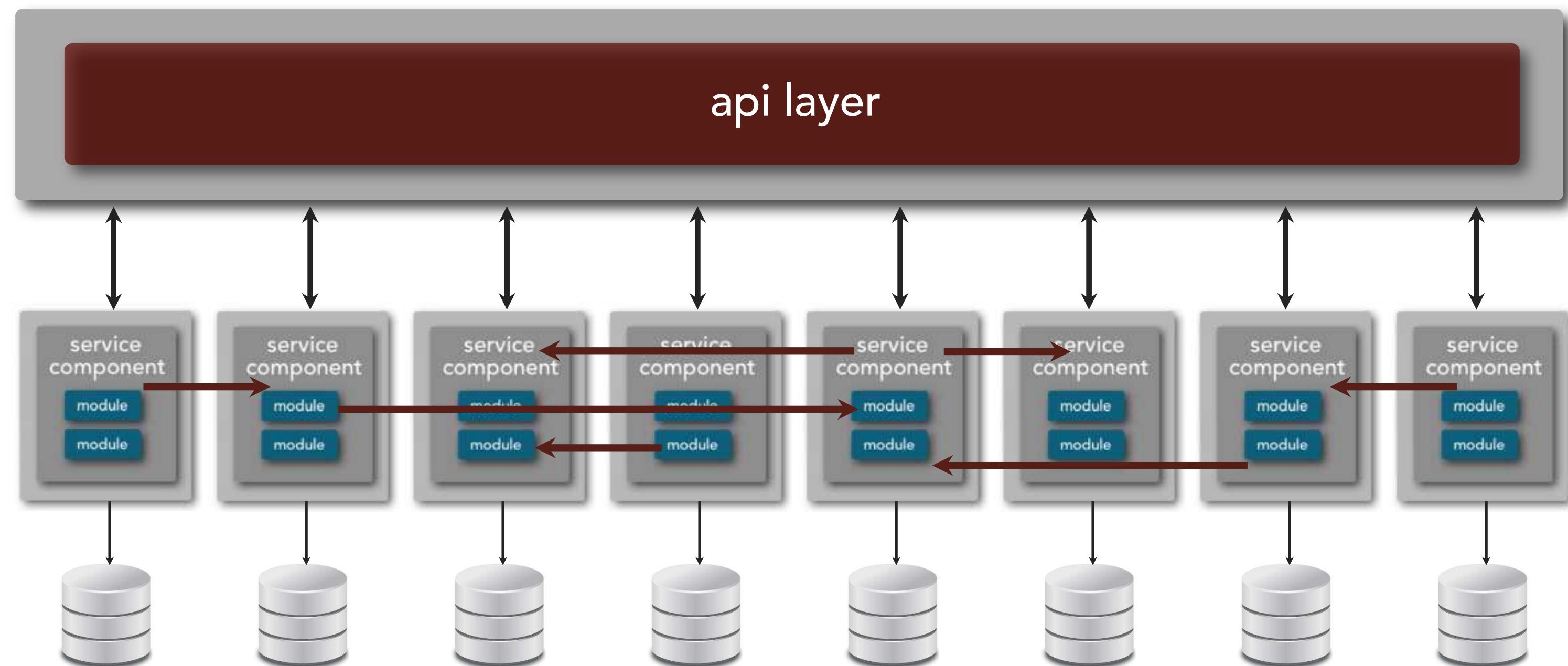
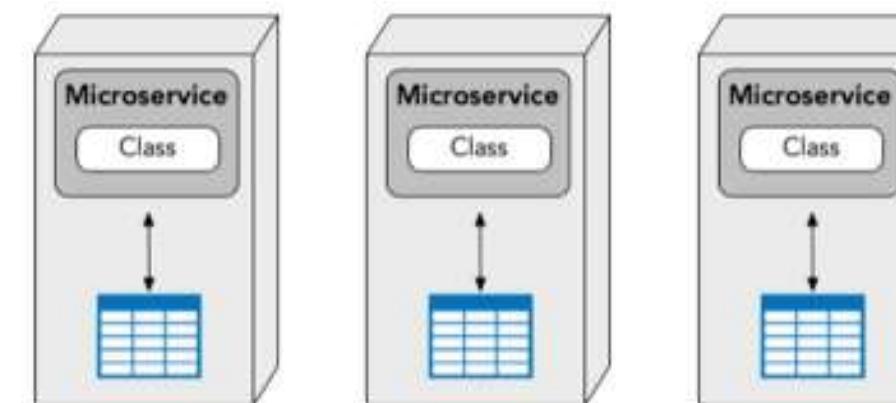
workflow and
choreography



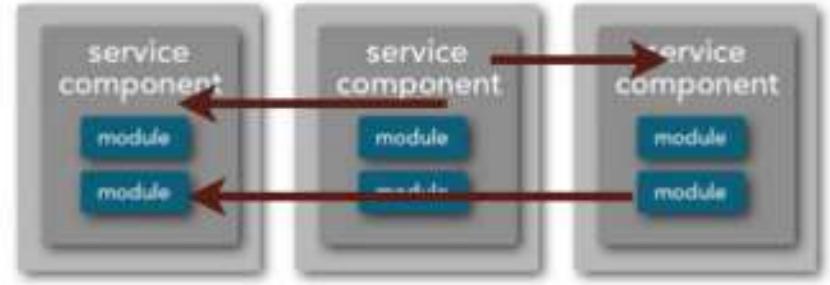
service granularity



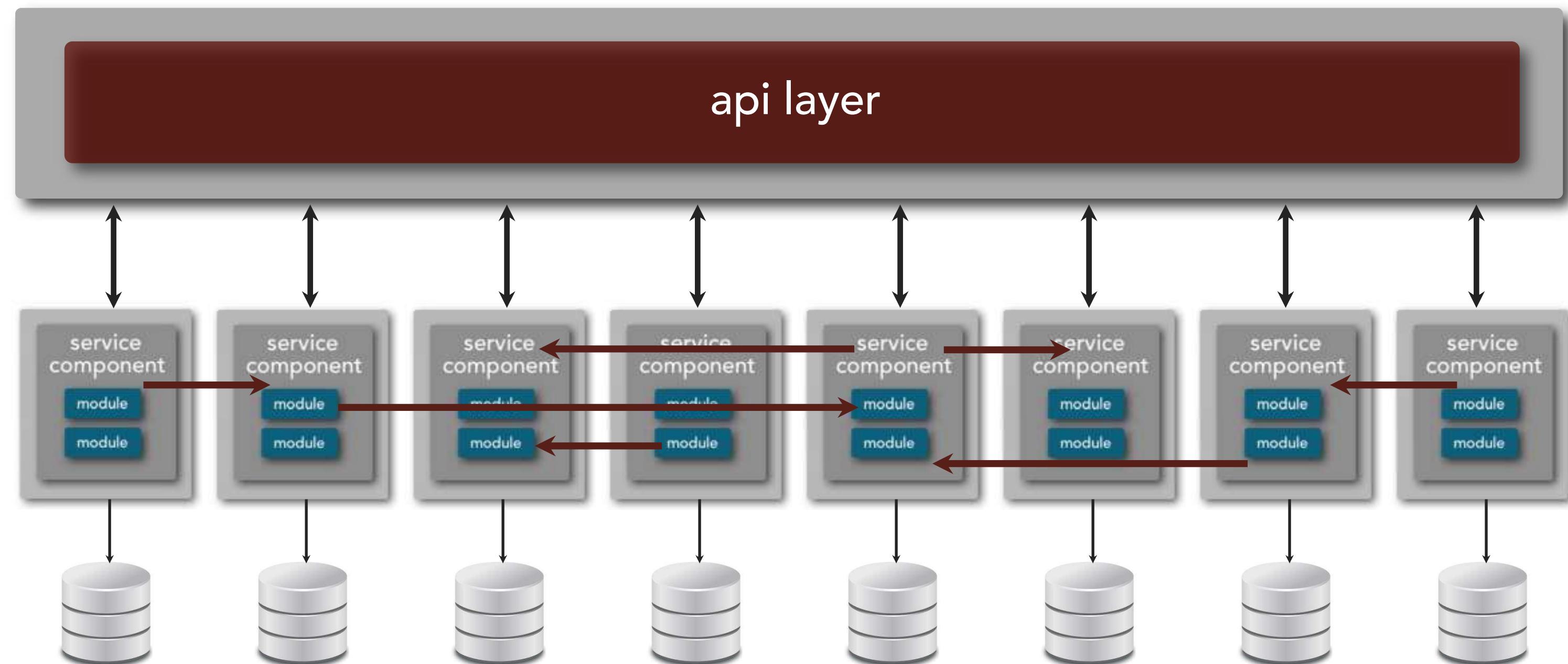
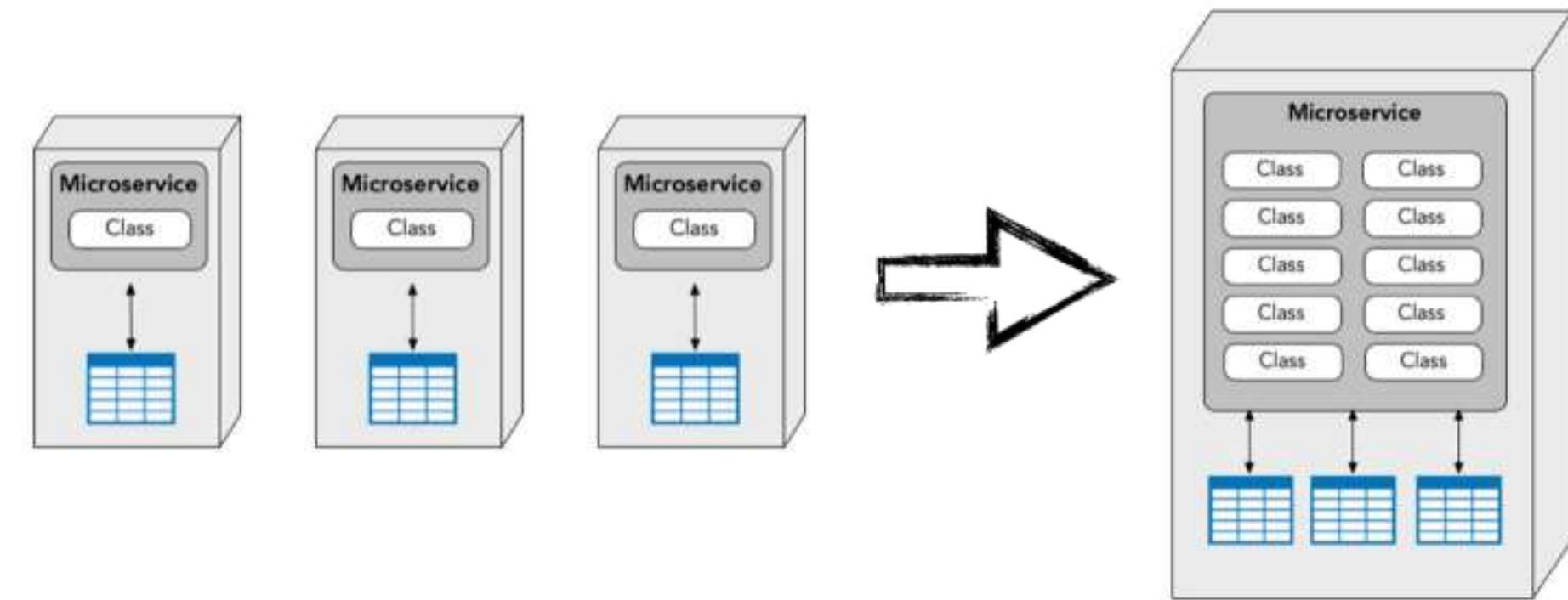
workflow and
choreography



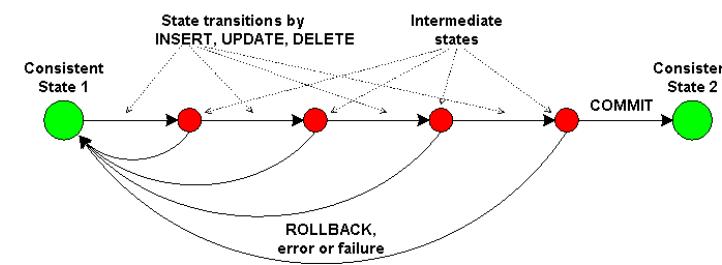
service granularity



workflow and
choreography



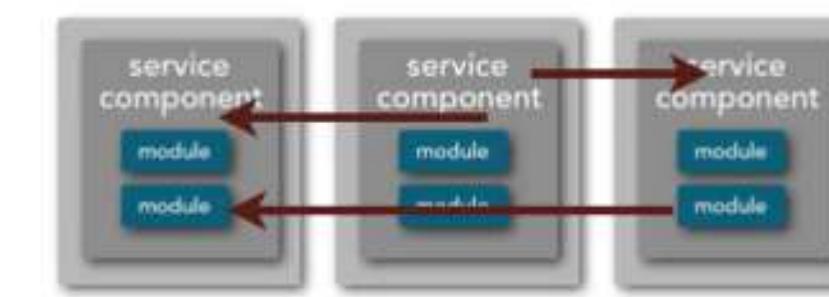
service granularity



database
transactions



data
dependencies



workflow and
choreography



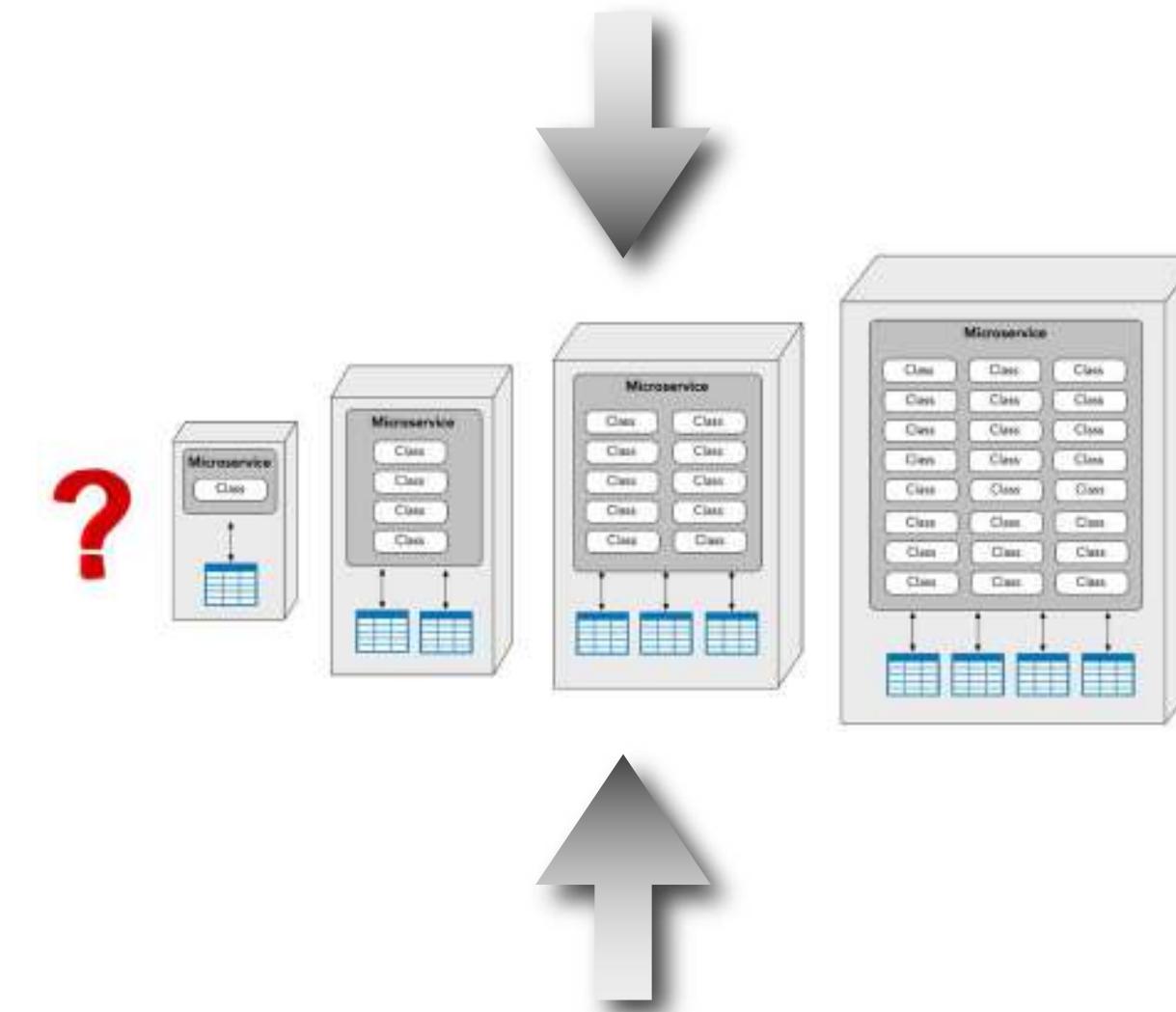
granularity factors

“what factors influence service granularity?”

service granularity

granularity drivers

“when should I consider breaking apart a service?”

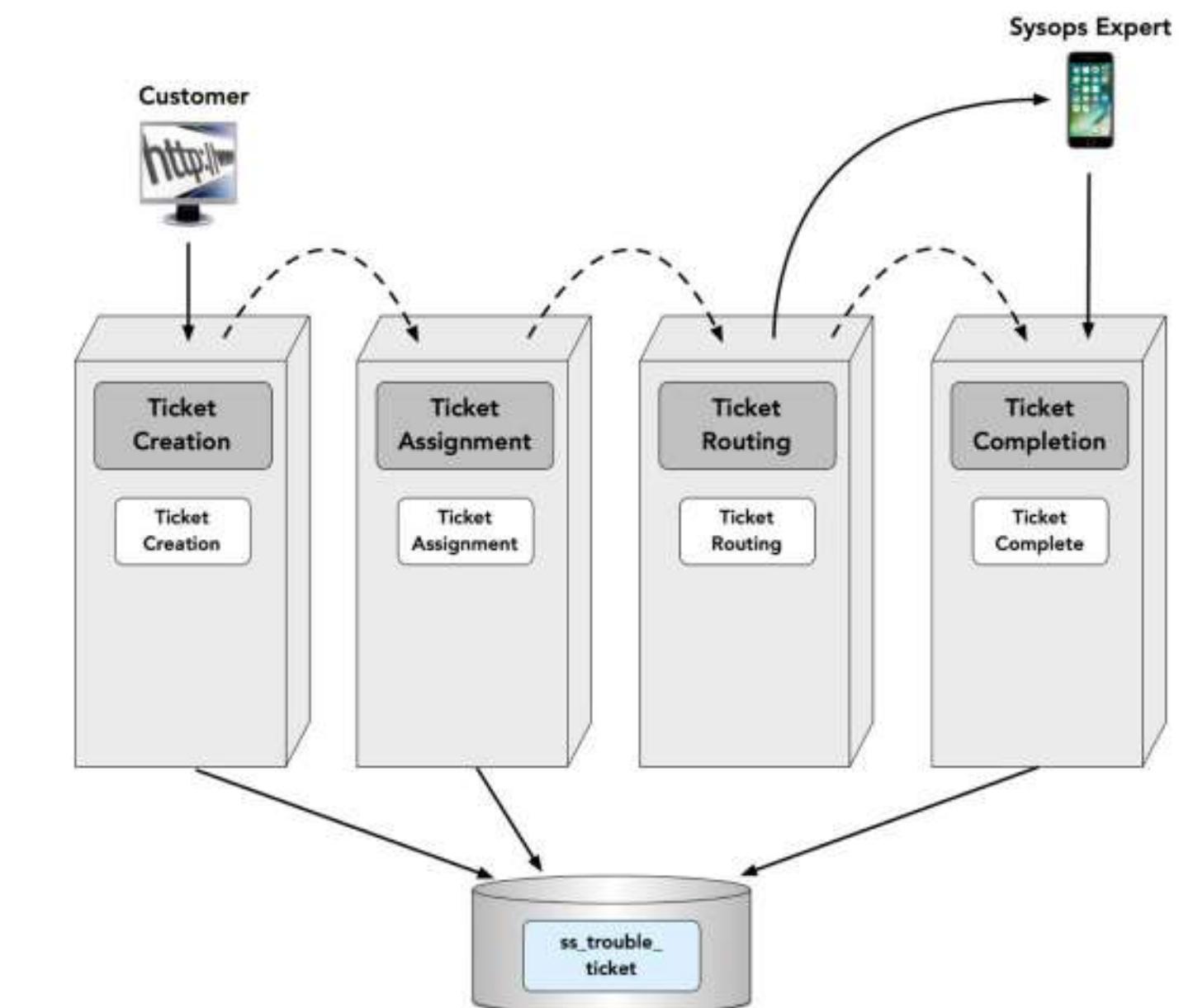
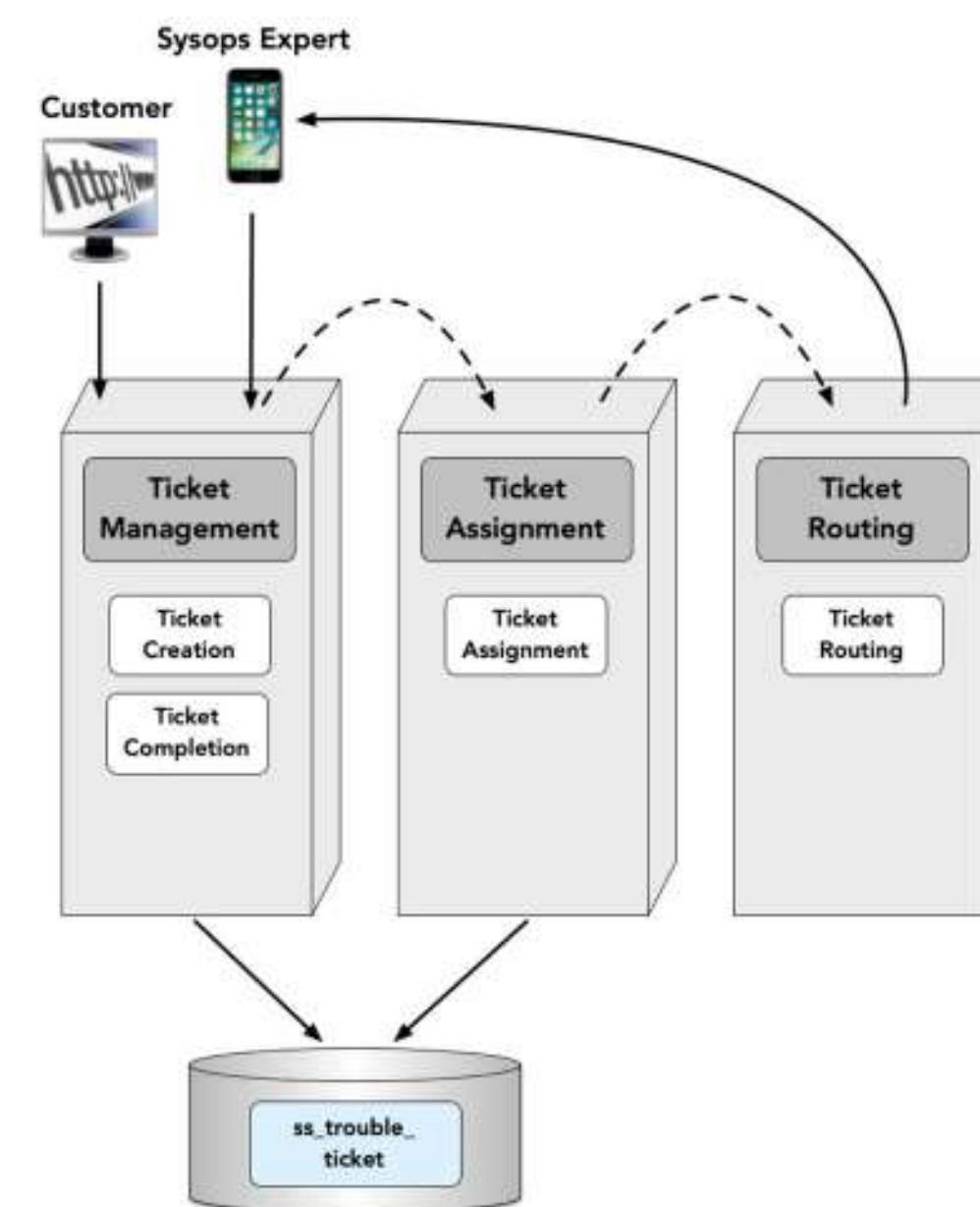
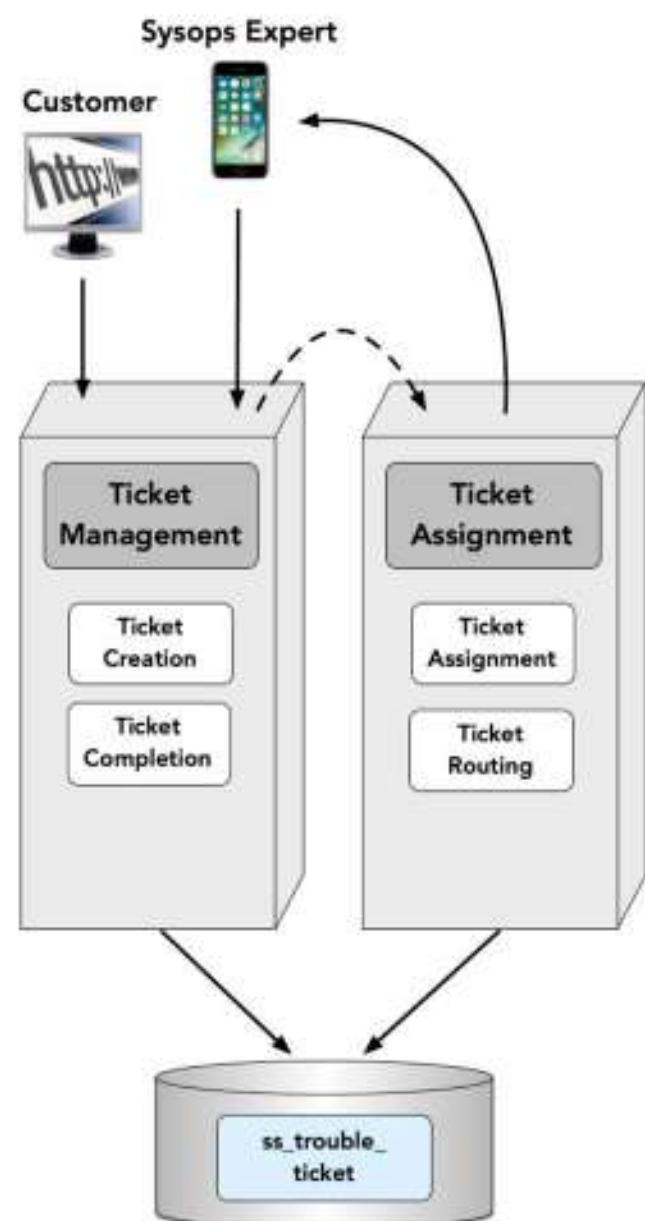
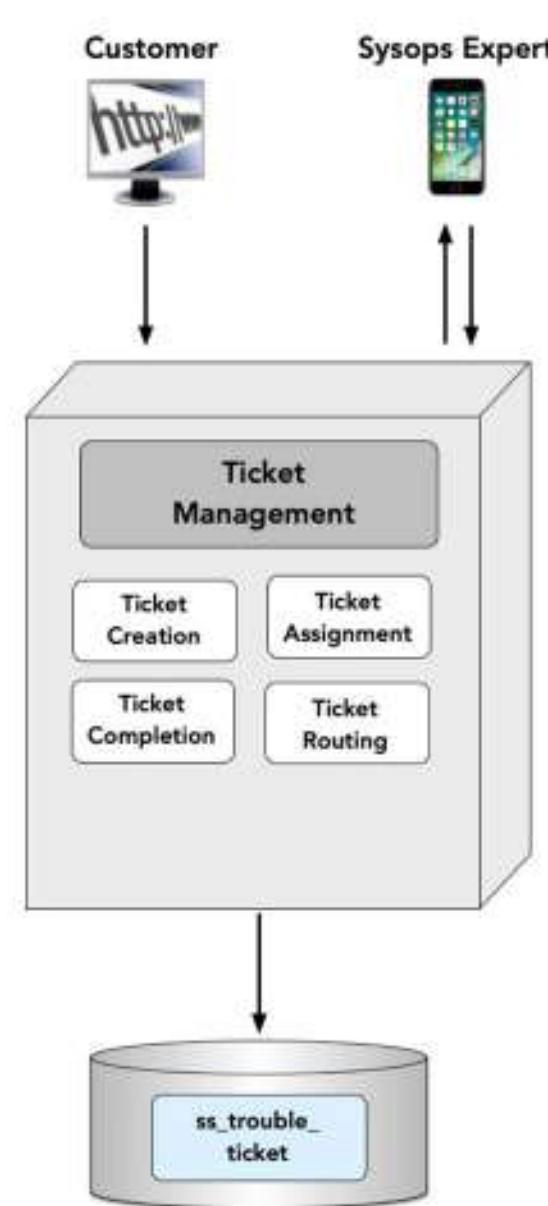
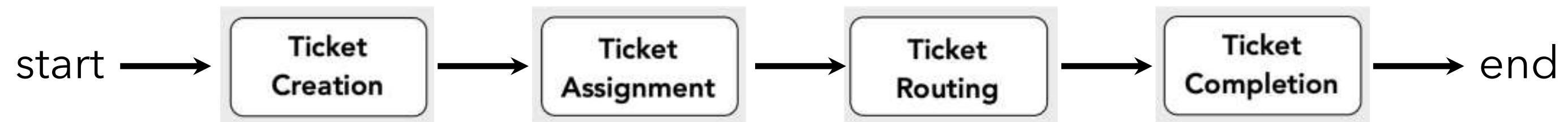


granularity factors

“what factors influence service granularity?”

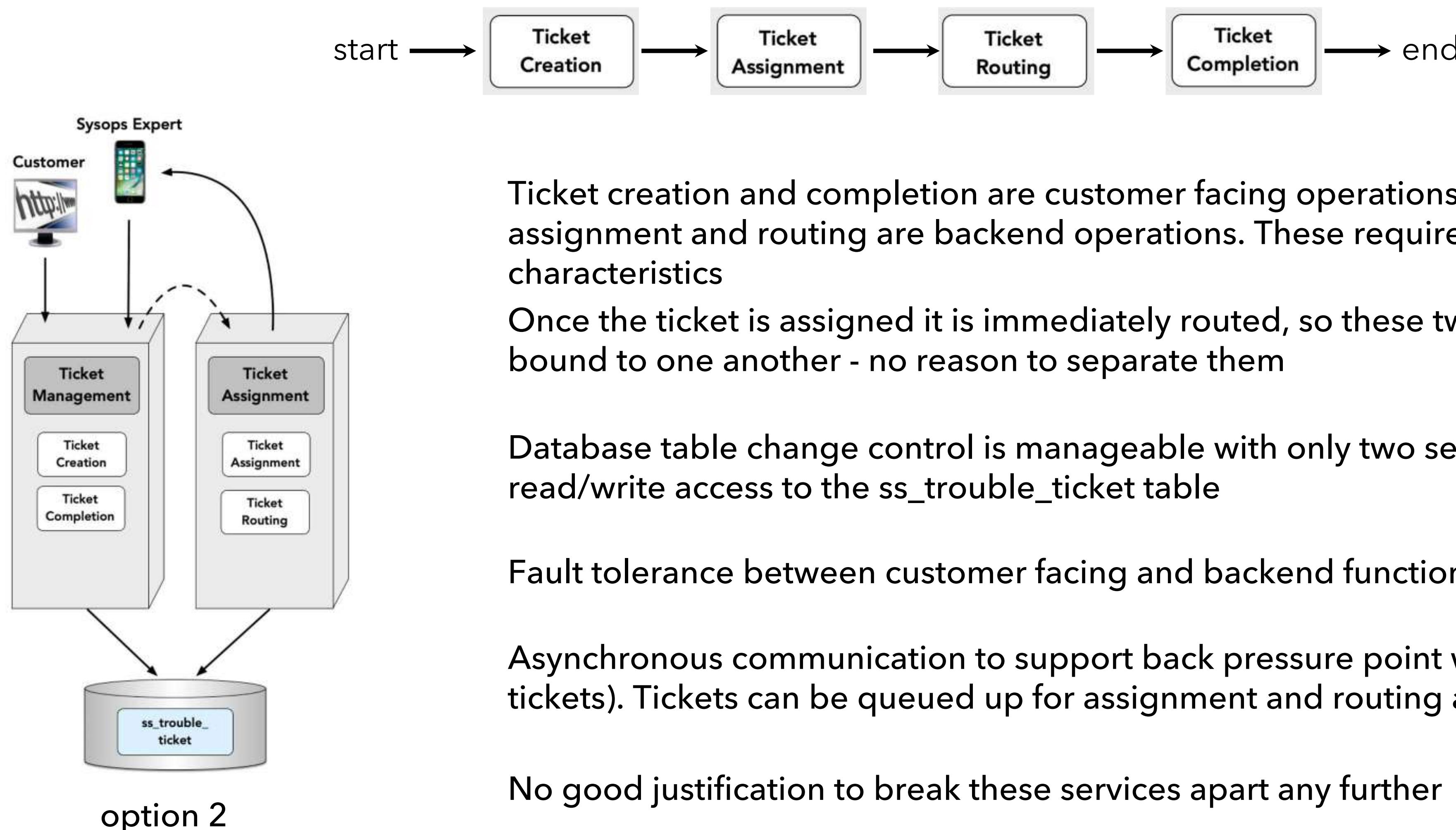
Scenario Exercise - Service Granularity

The primary ticket workflow uses the four components below. What service option would you choose and why?



Scenario Exercise - Service Granularity (Instructor)

The primary ticket workflow uses the four components below. What service option would you choose and why?





*How do I automate
architecture governance?*

Evolutionary Architecture

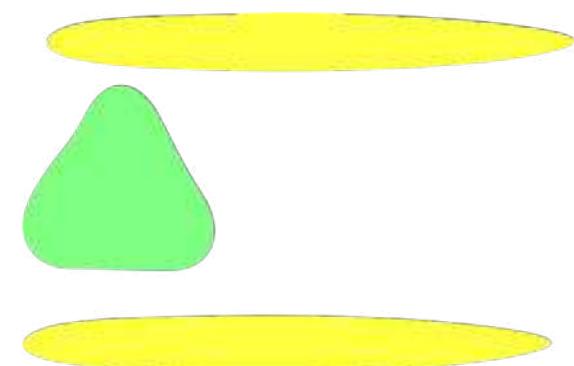
An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



Evolutionary Architecture

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



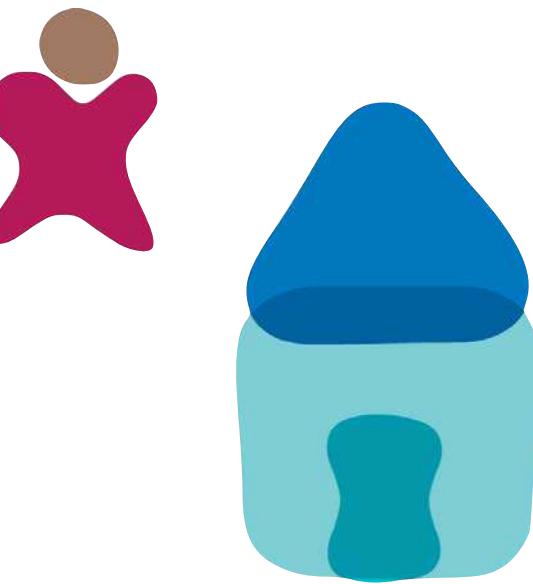
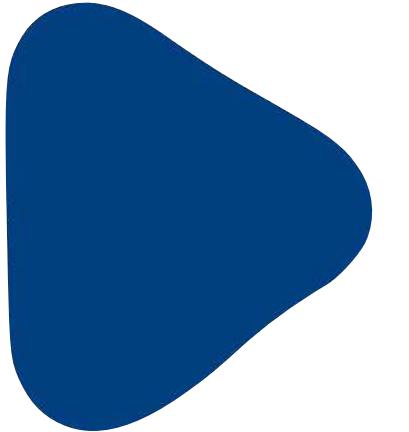
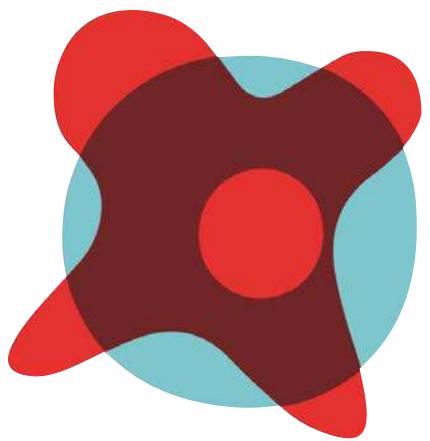
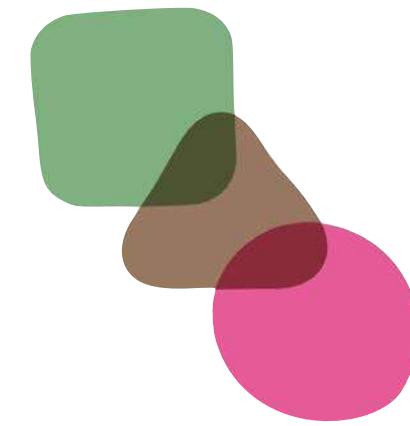
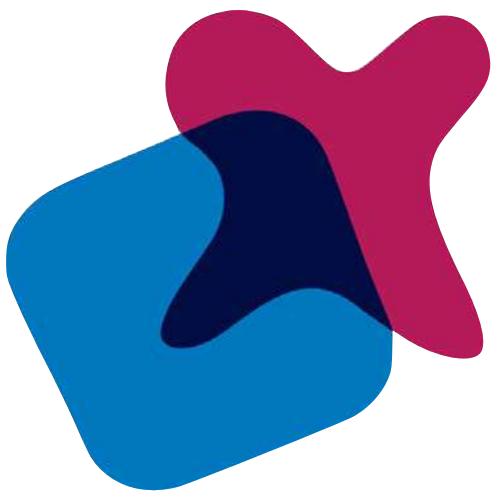
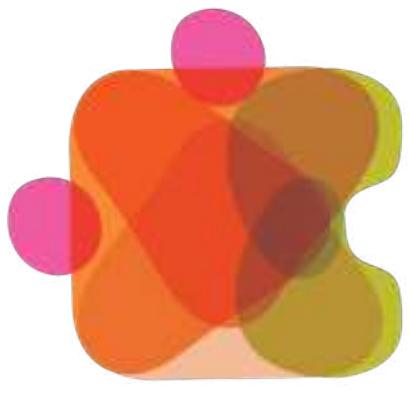


guided

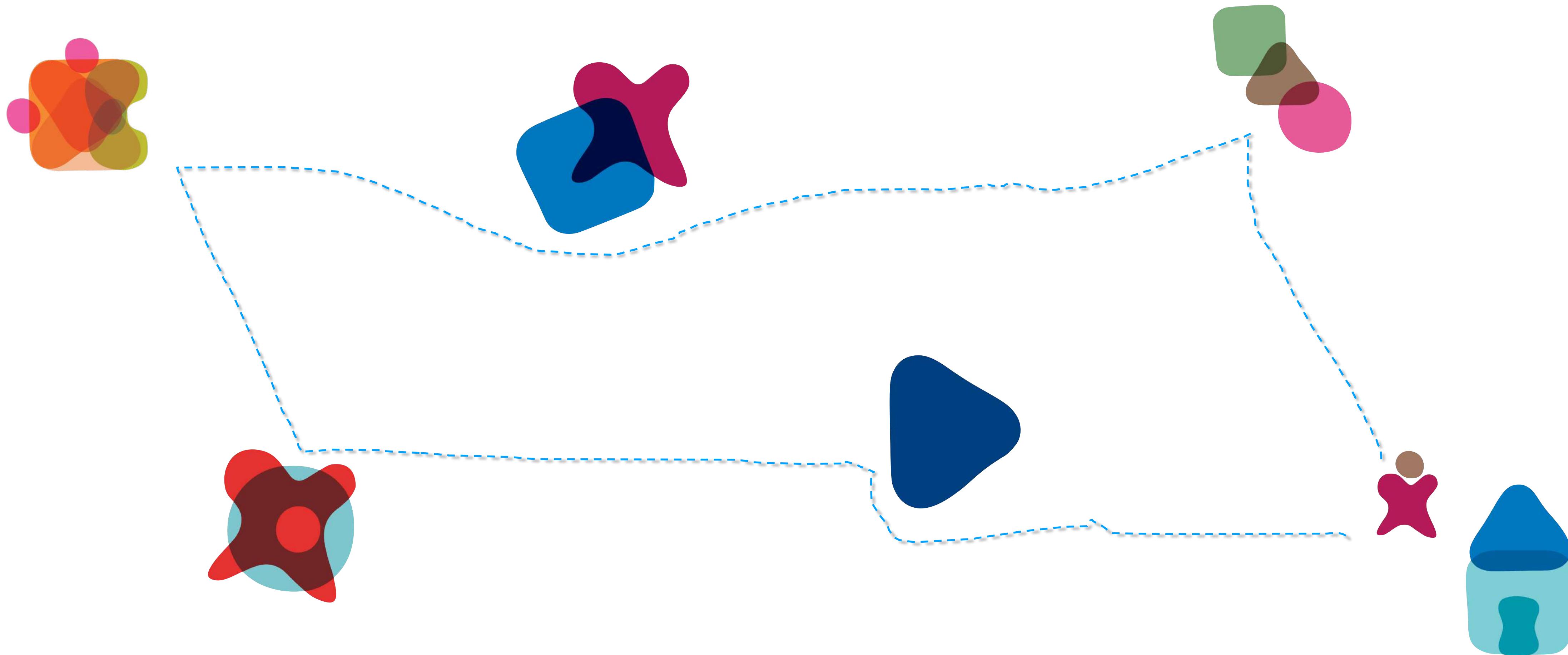
evolutionary computing fitness function:

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

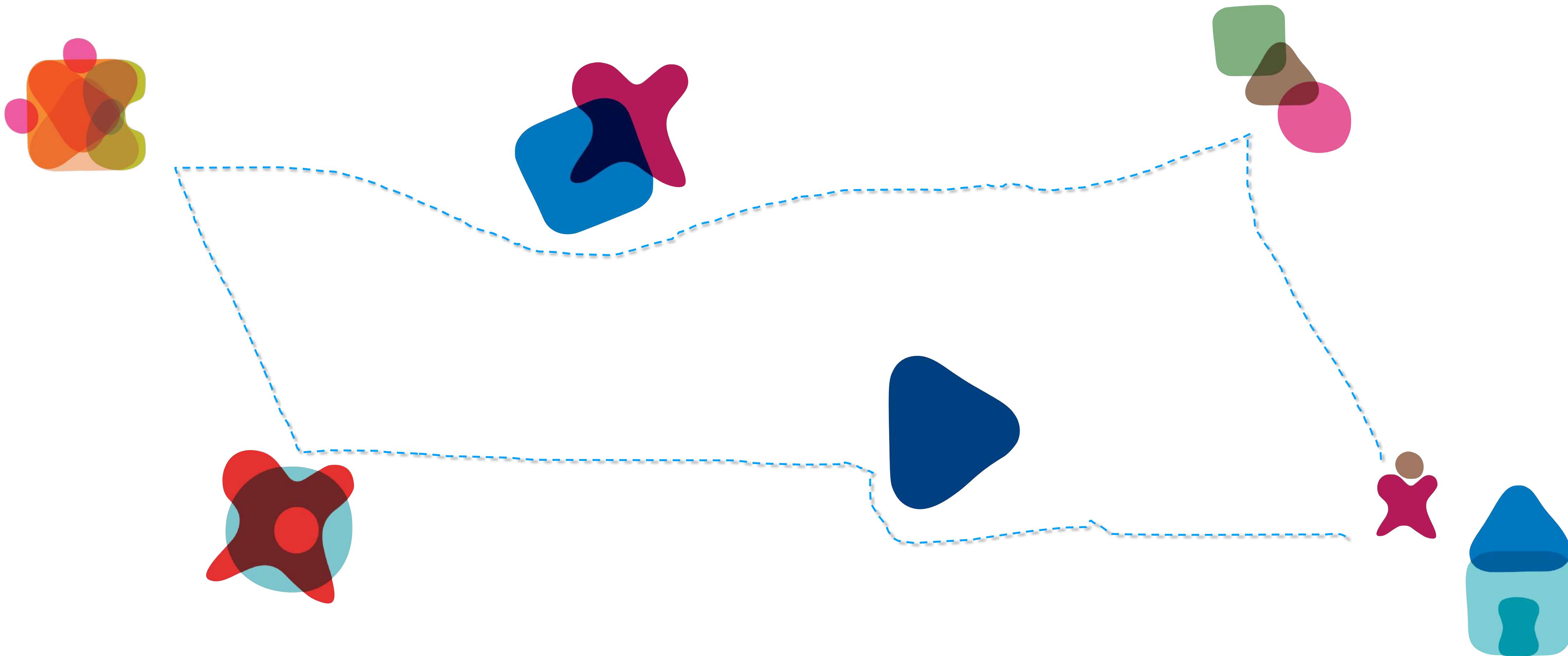
traveling salesperson problem



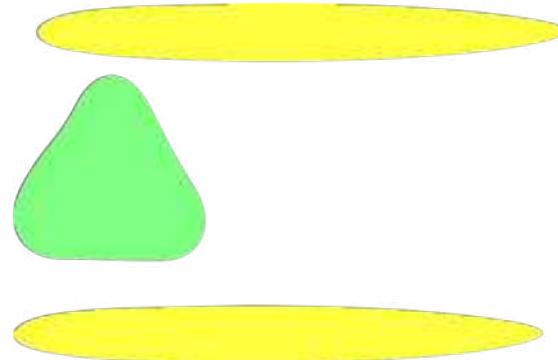
traveling salesperson problem



traveling salesperson problem



fitness function = length of route

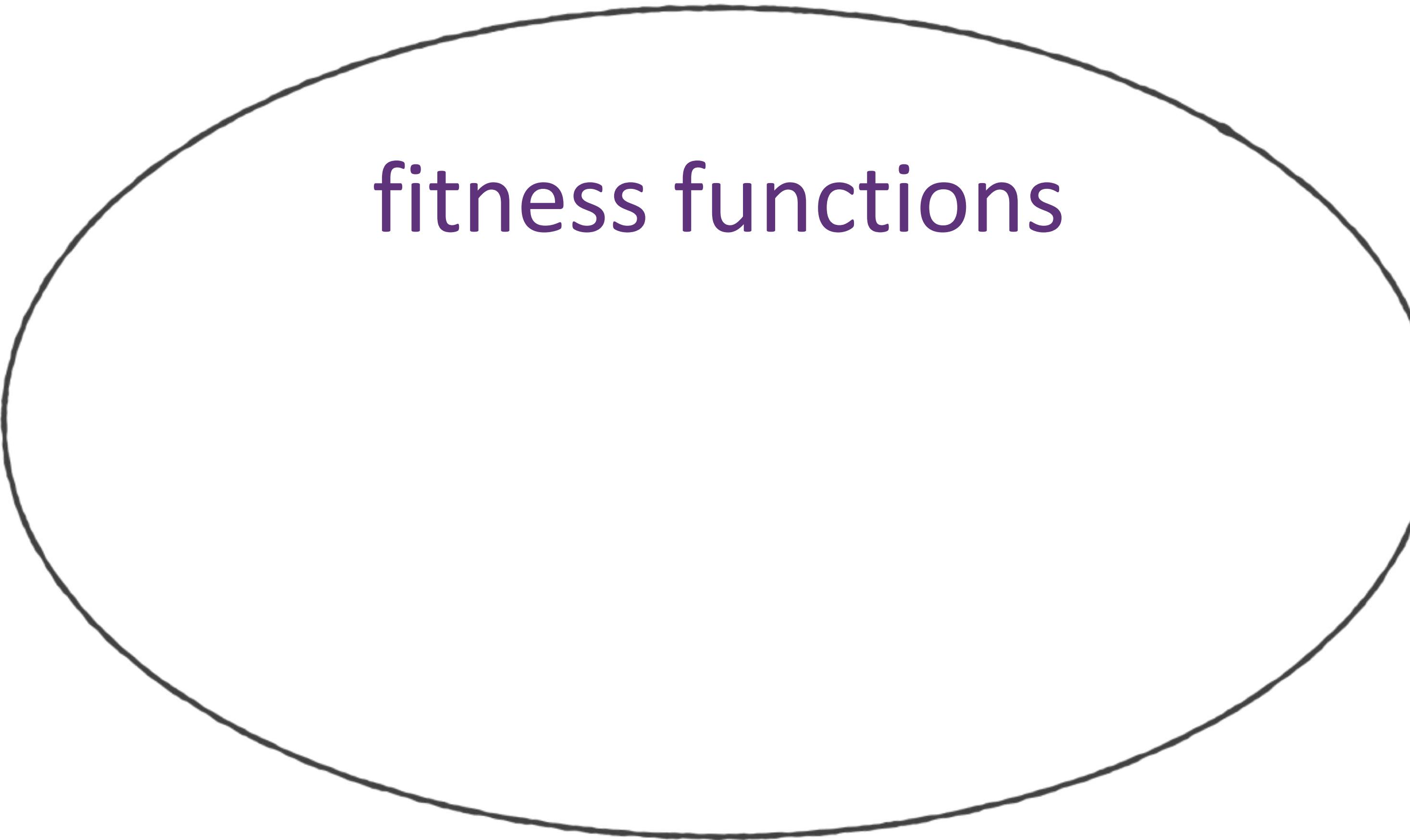


guided

architectural fitness function:

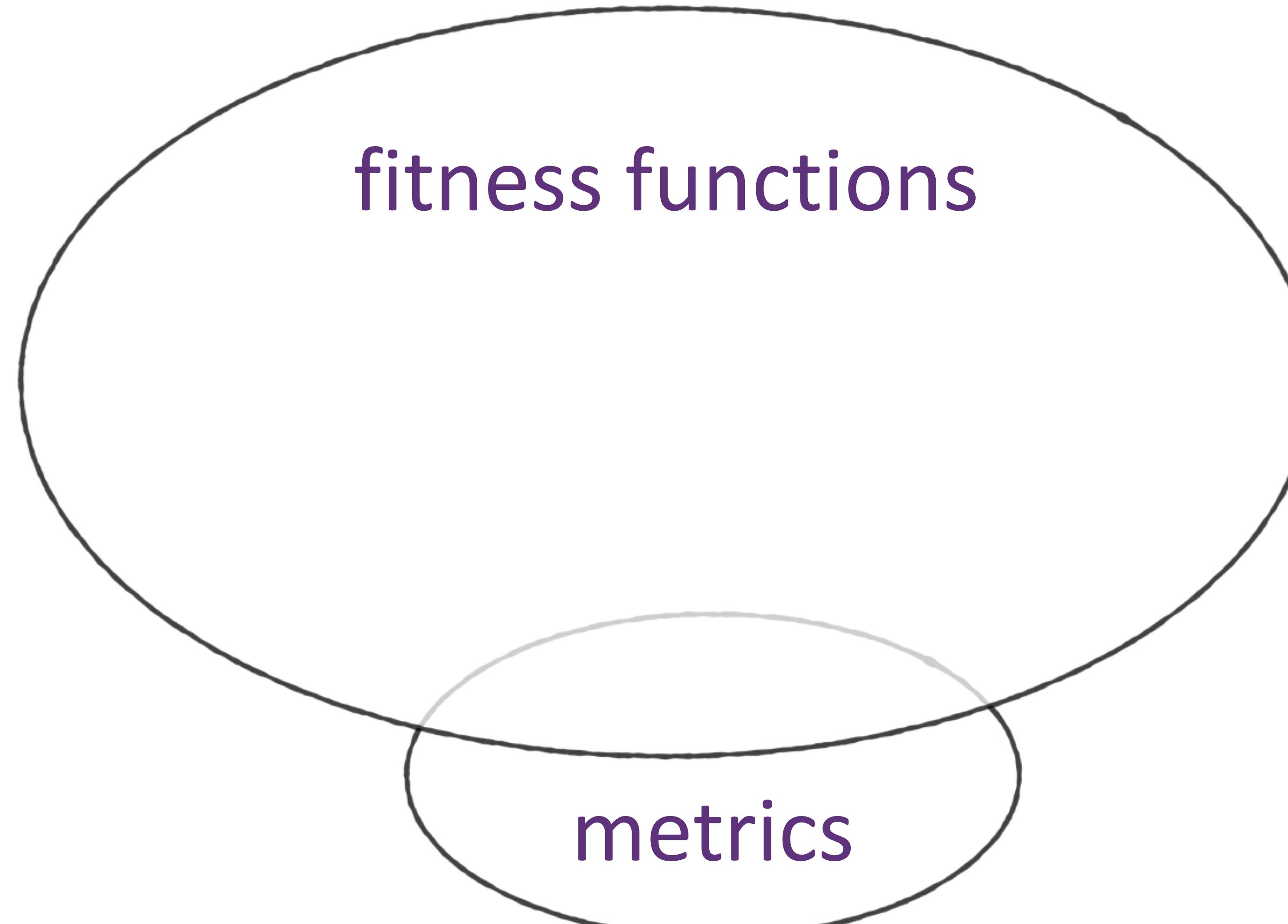
any mechanism that provides an objective integrity assessment of some architectural characteristic(s).

fitness functions



fitness functions

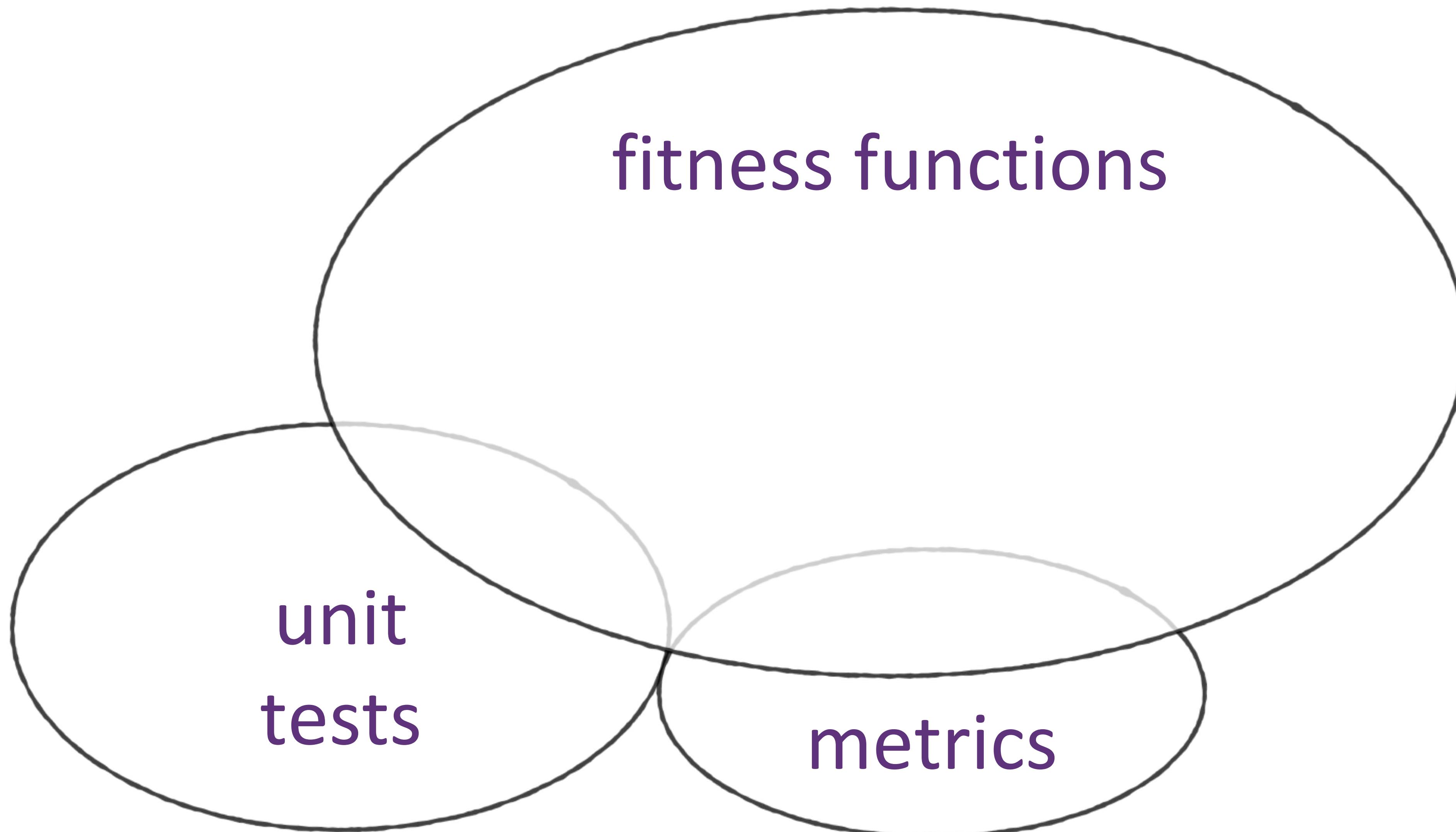
fitness functions



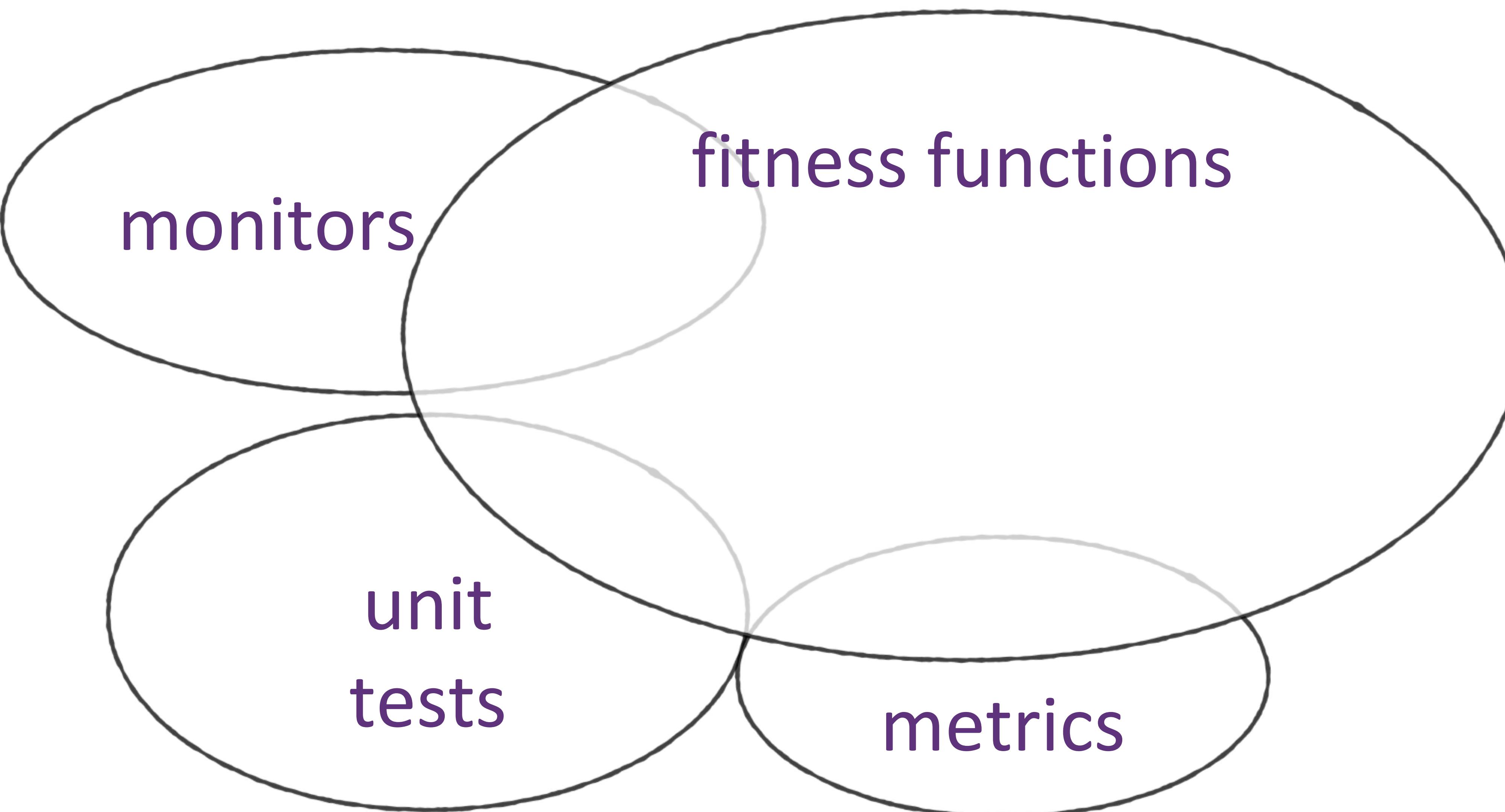
fitness functions

metrics

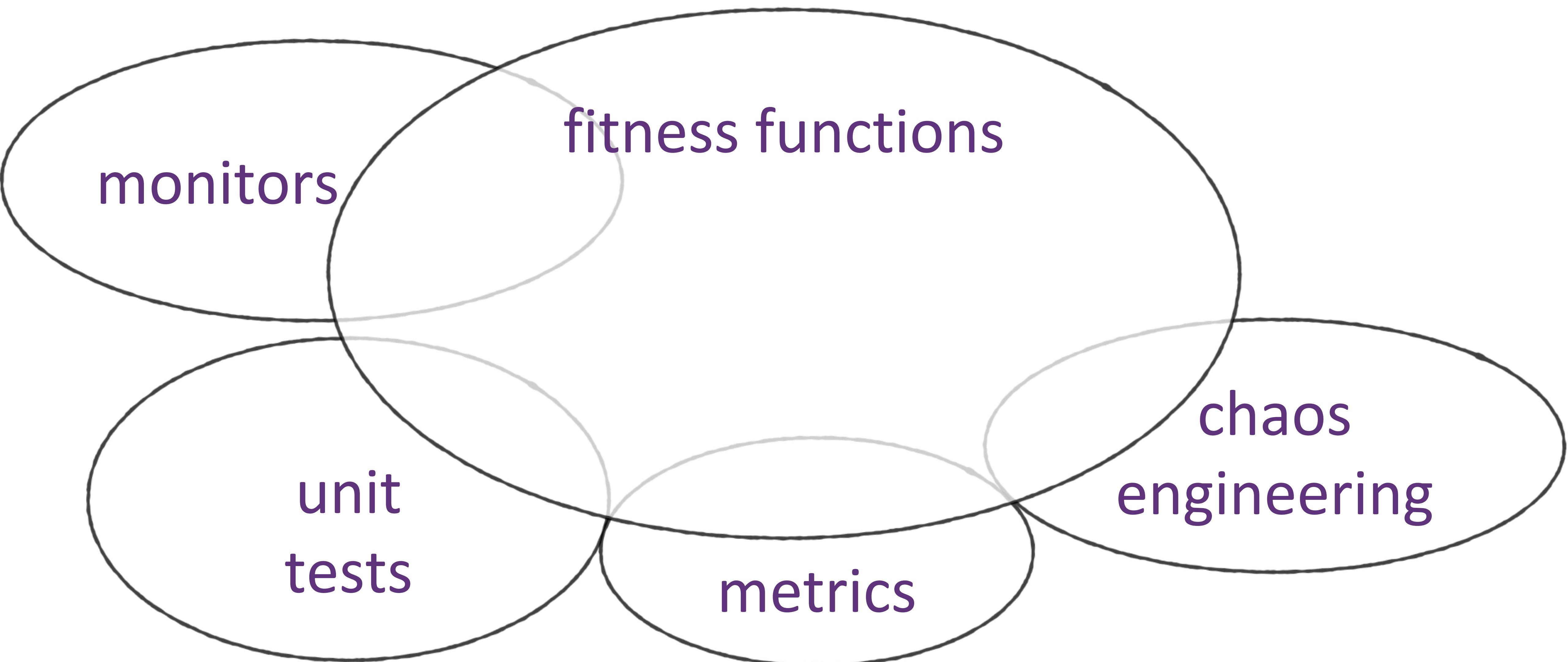
fitness functions



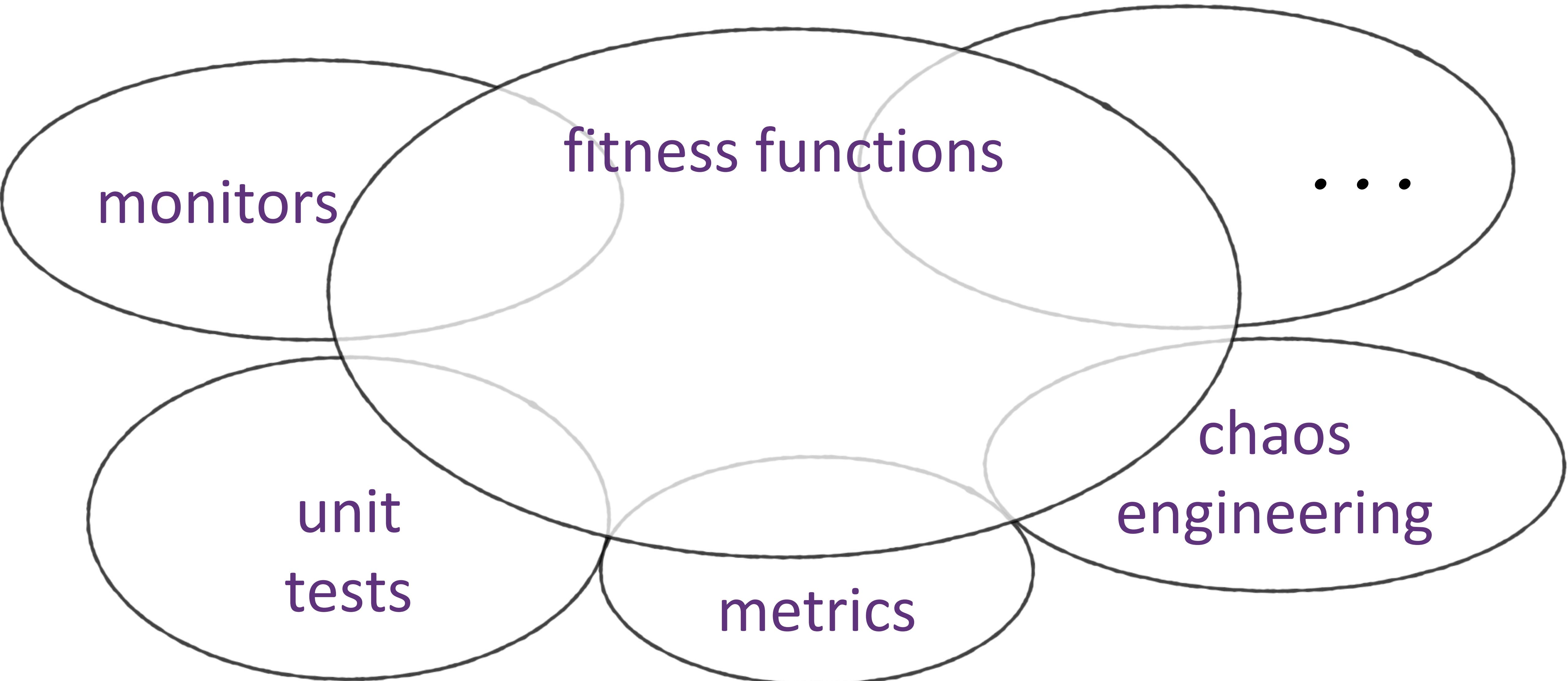
fitness functions



fitness functions

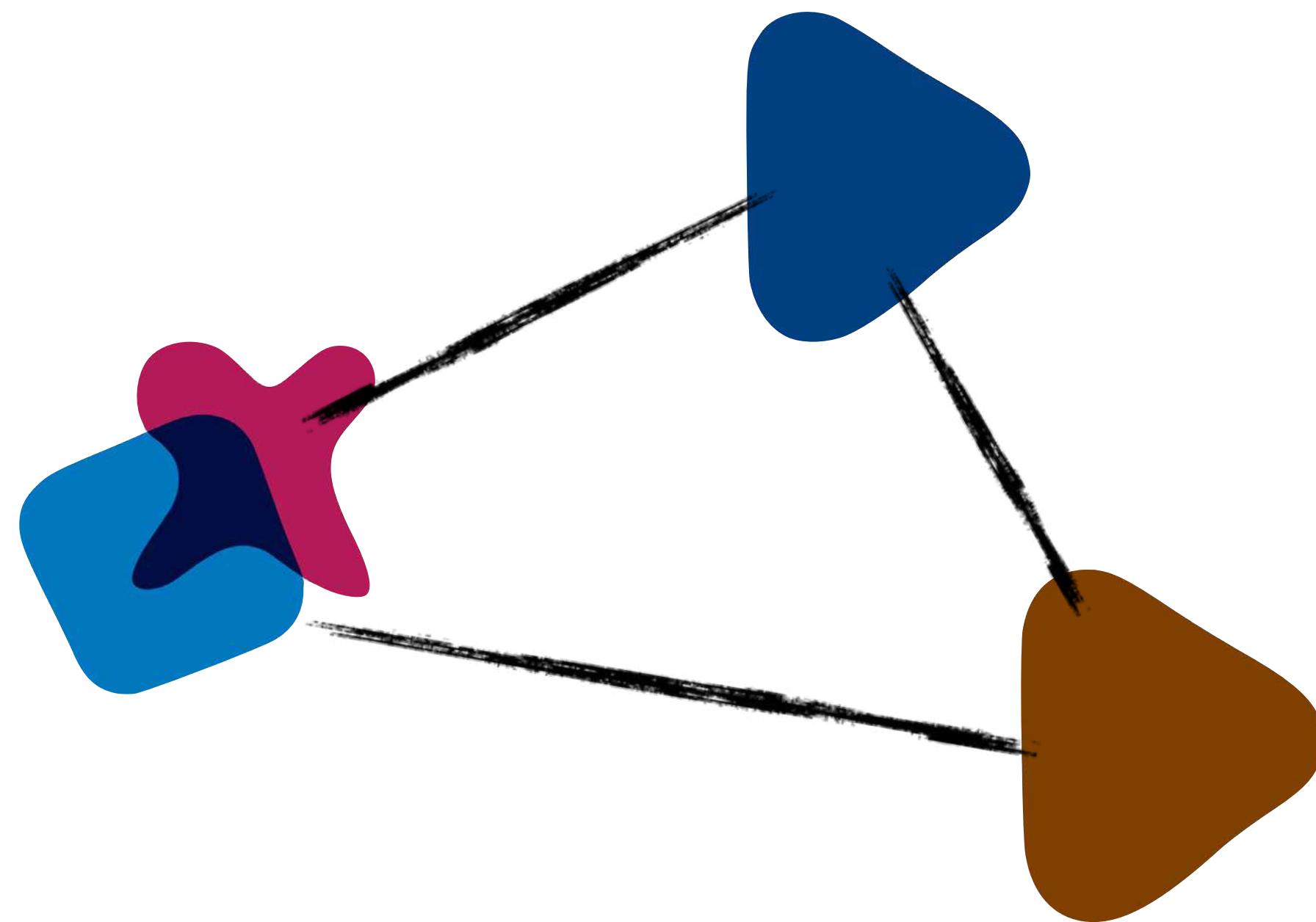


fitness functions

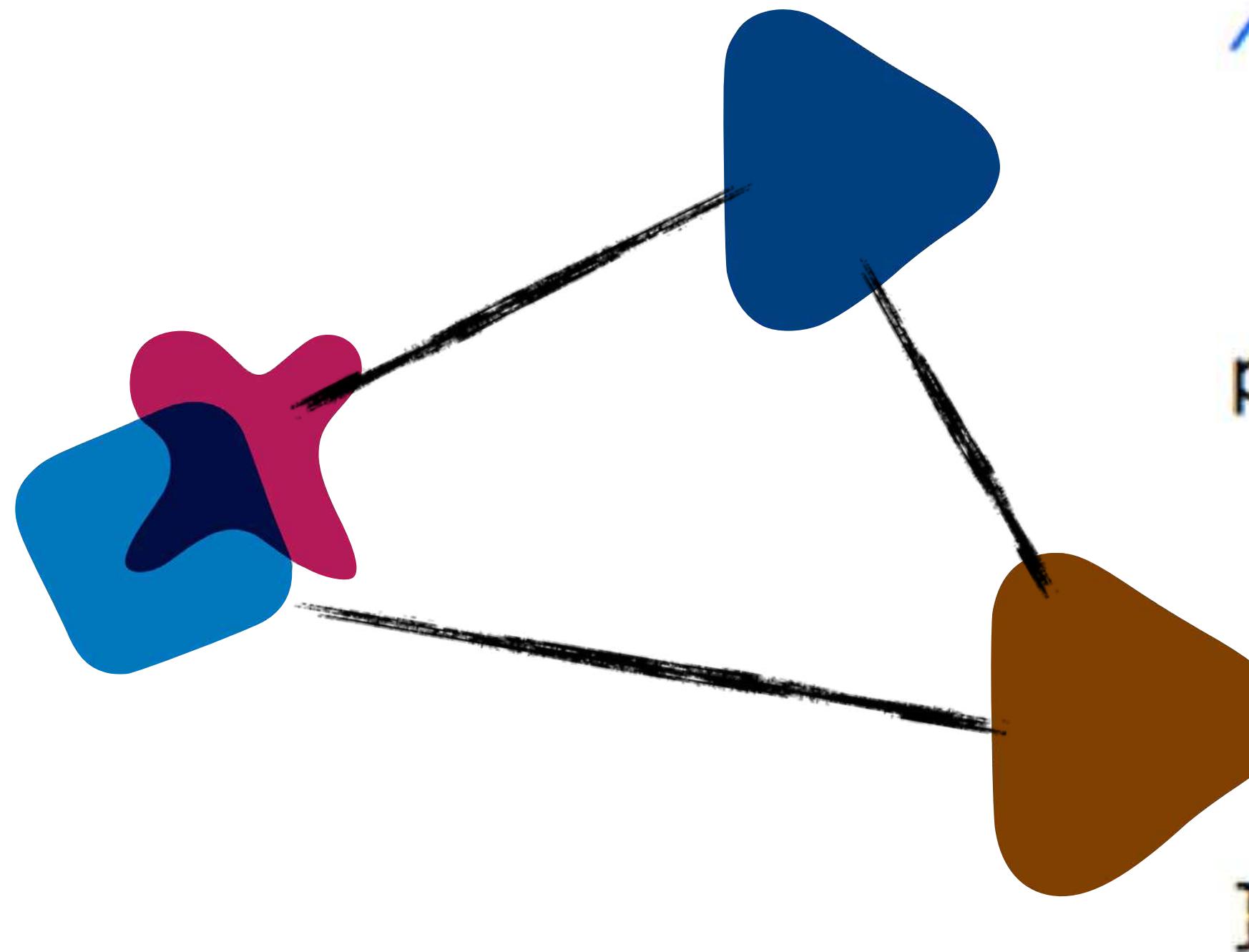


cyclic dependency function

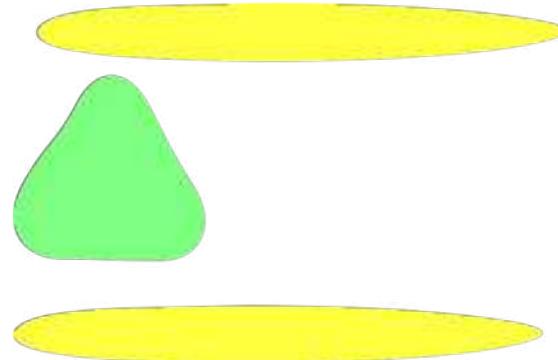
cyclic dependency function



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());  
}
```



guided

architectural fitness function:

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

<https://www.archunit.org/>

The screenshot shows a web browser window with the URL <https://www.archunit.org/> in the address bar. The page itself has a blue header with the ArchUnit logo and navigation links for Getting Started, Motivation, News, User Guide, API, and About. The main content area features a large white title "Unit test your Java architecture" and a subtext "Start enforcing your architecture within 30 minutes using the test setup you already have." Below this is a "Start Now" button. The background of the main section is a blue gradient with abstract architectural icons like files and connections.

Persistence

Unit test your Java architecture

Start enforcing your architecture within 30 minutes using the test setup you already have.

Start Now

ArchUnit is a free, simple and extensible library for checking the architecture of your Java code using any plain Java unit test framework. That is, ArchUnit can check dependencies between packages and classes, layers and slices, check for cyclic dependencies and more. It does so by analyzing given Java bytecode, importing all classes into a Java code structure. You can find examples for the current release at [ArchUnit Examples](#) and the sources on [GitHub](#).

News

<https://www.archunit.org/>

coding rules

```
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);
    }
}
```

<https://www.archunit.org/>

```
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);
    }

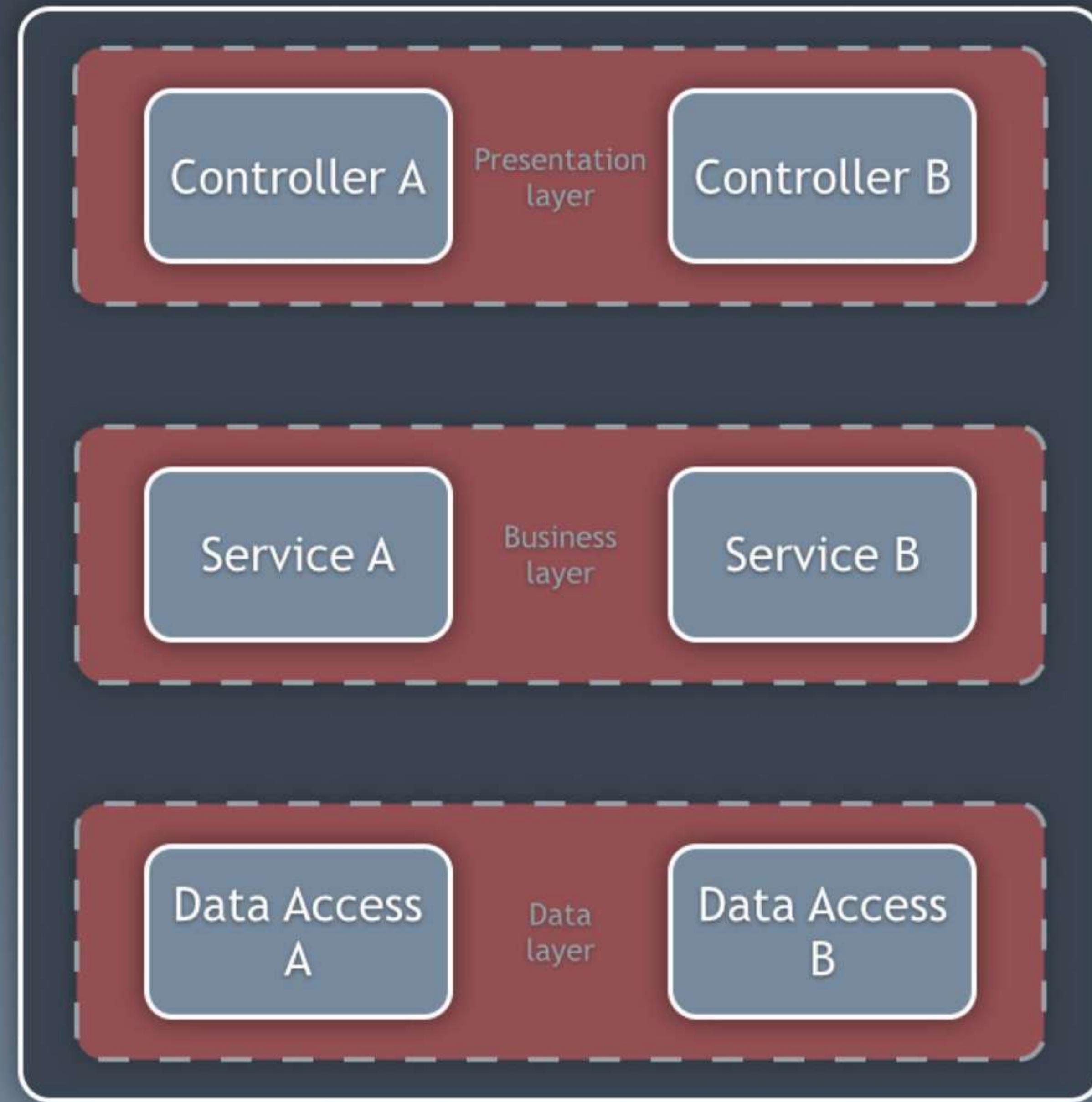
    @Test
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
            SomeBusinessInterface.class,
            SomeDao.class
        );

        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/>

```
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;
```

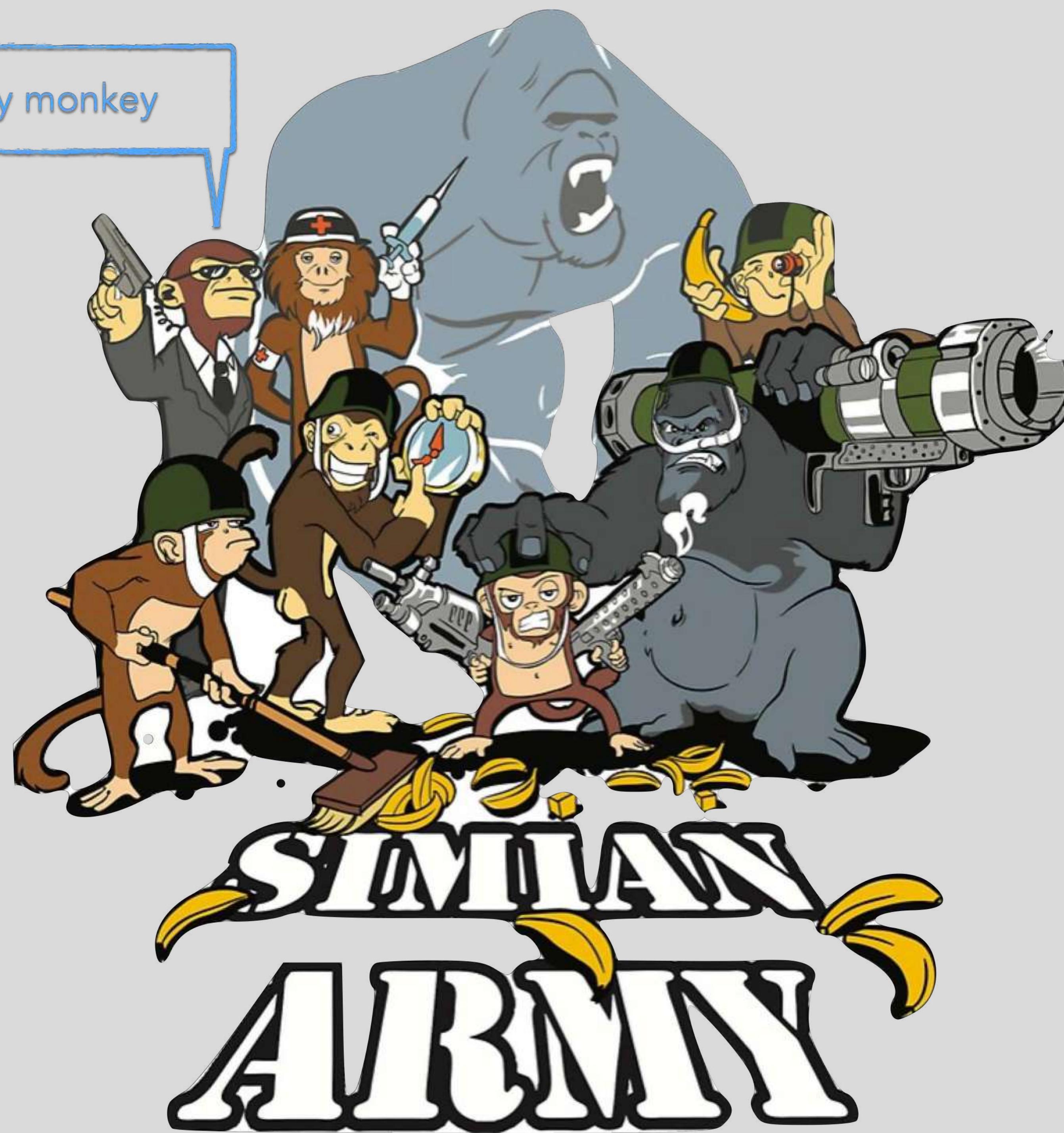




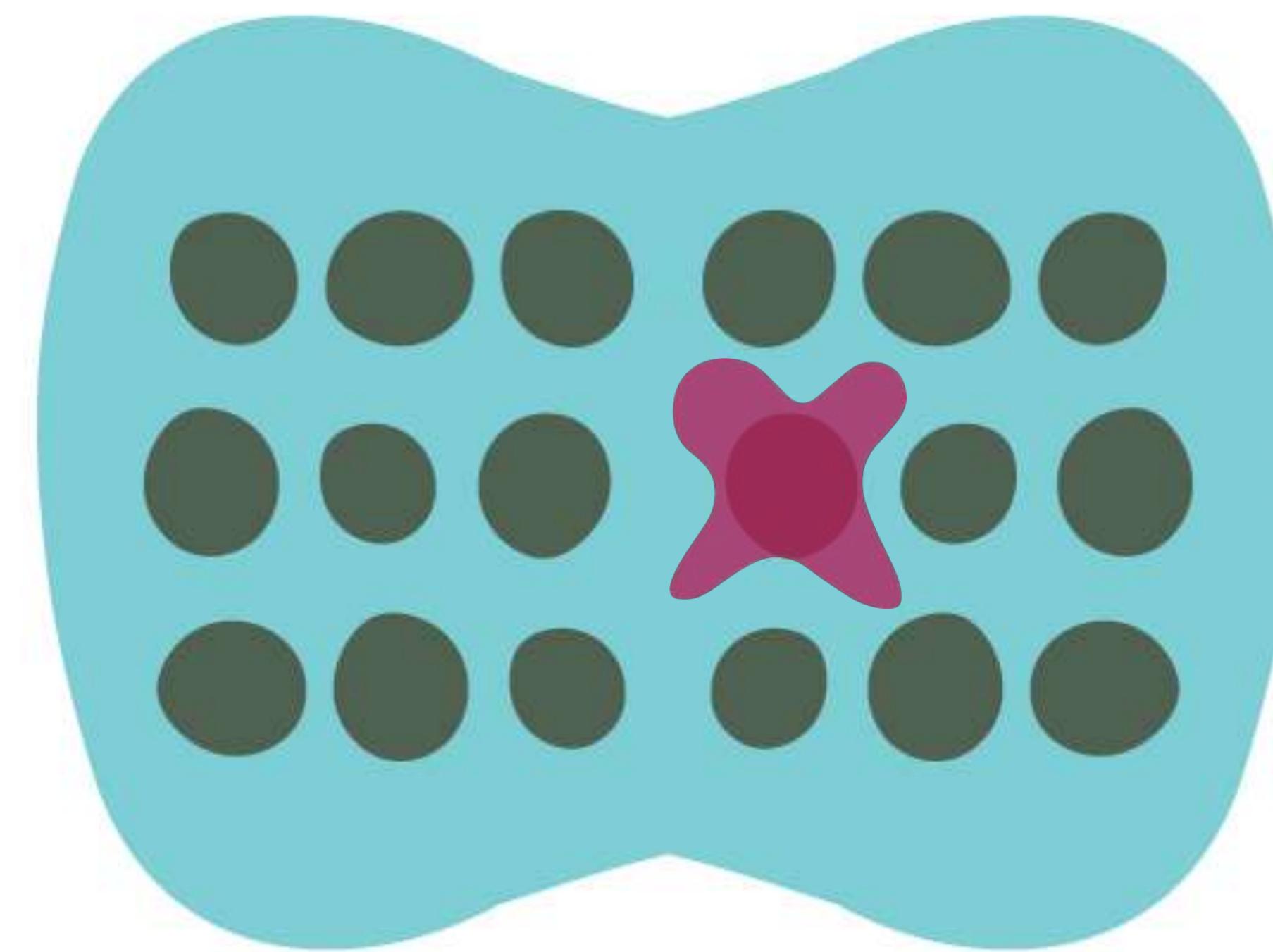


conformity monkey

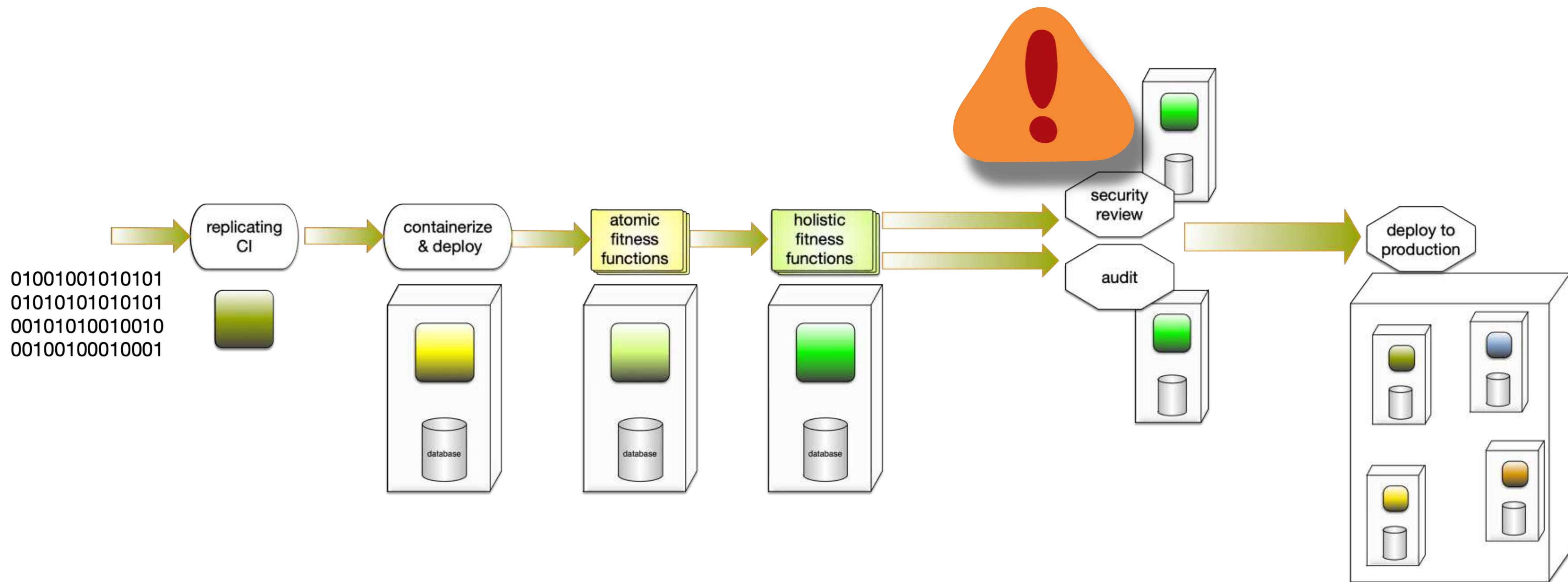
security monkey



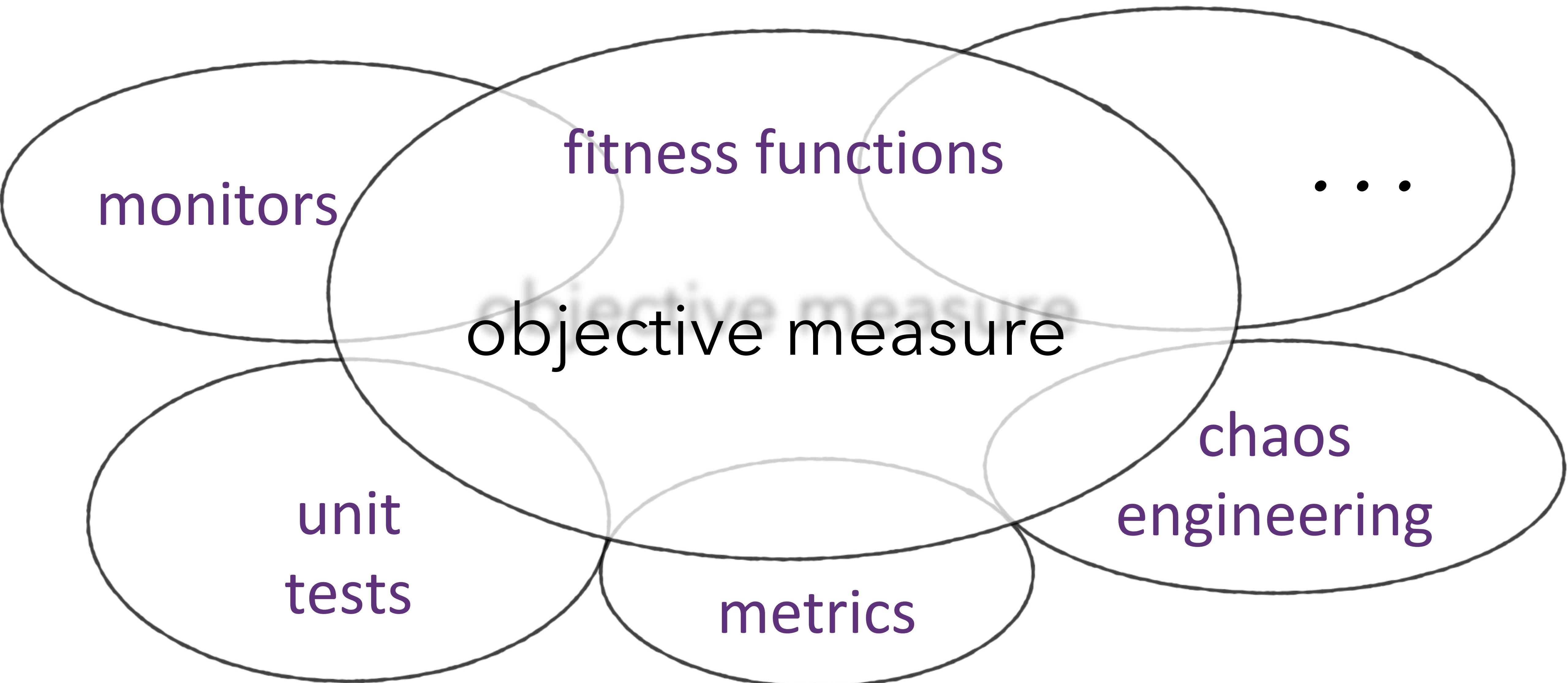
Zero-day Security Check



Zero-day Security Check



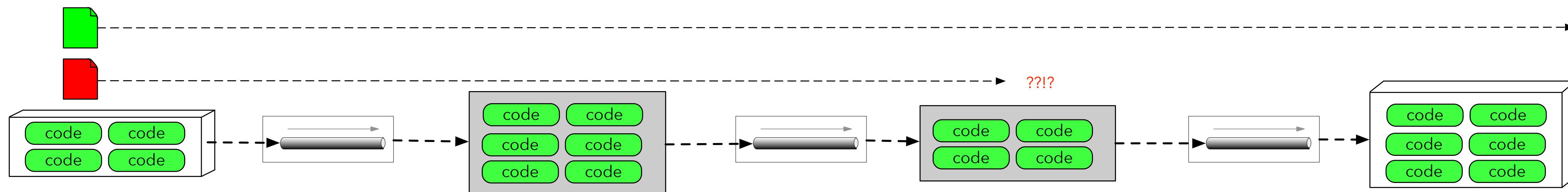
fitness functions



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.

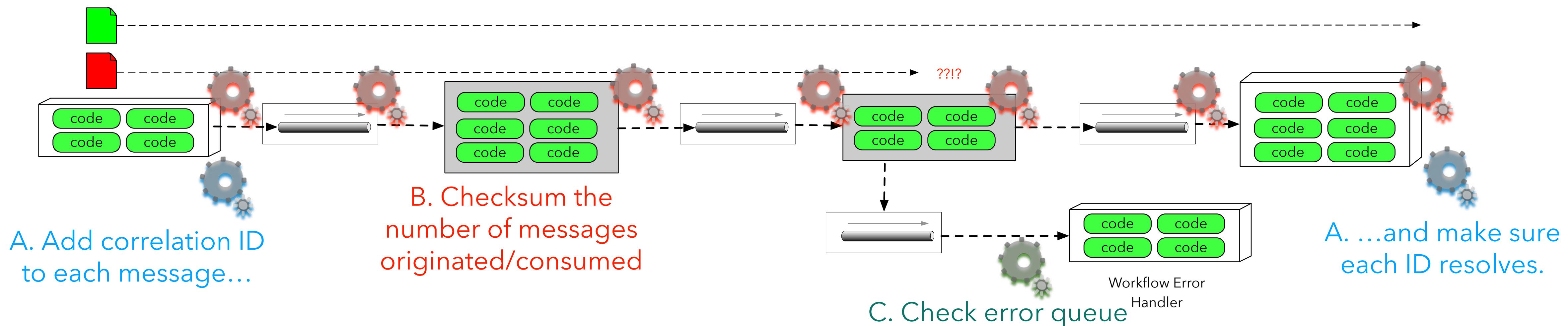
1. What kind of fitness function can the Sysop Squad architects write to track down the root cause?



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.

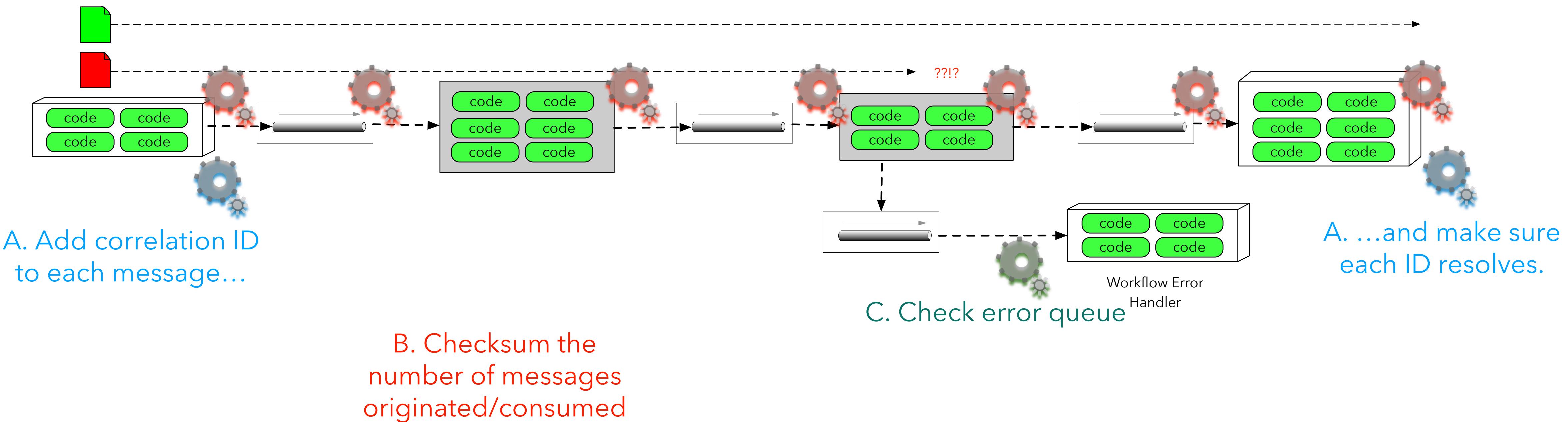
1. What kind of fitness function can the Sysop Squad architects write to track down the root cause?



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.

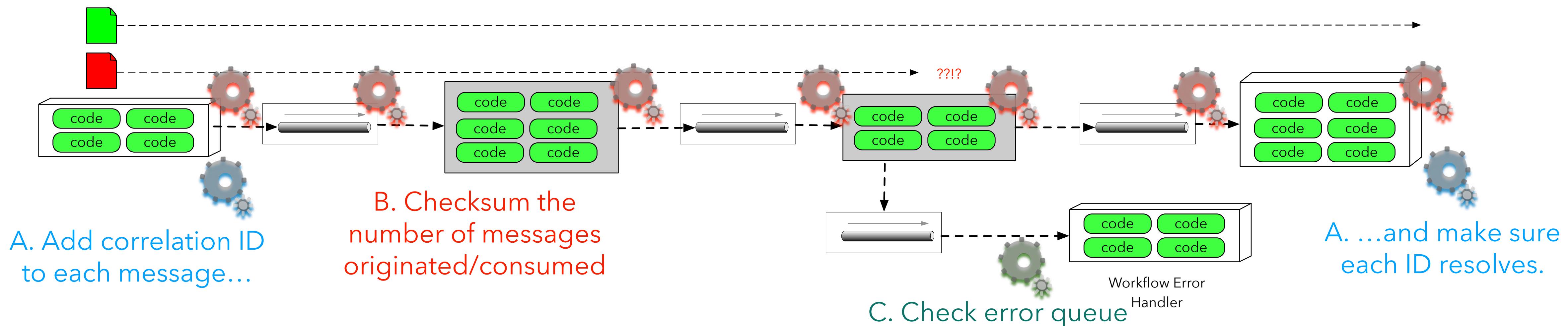
1. What kind of fitness function can the Sysop Squad architects write to track down the root cause?



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.

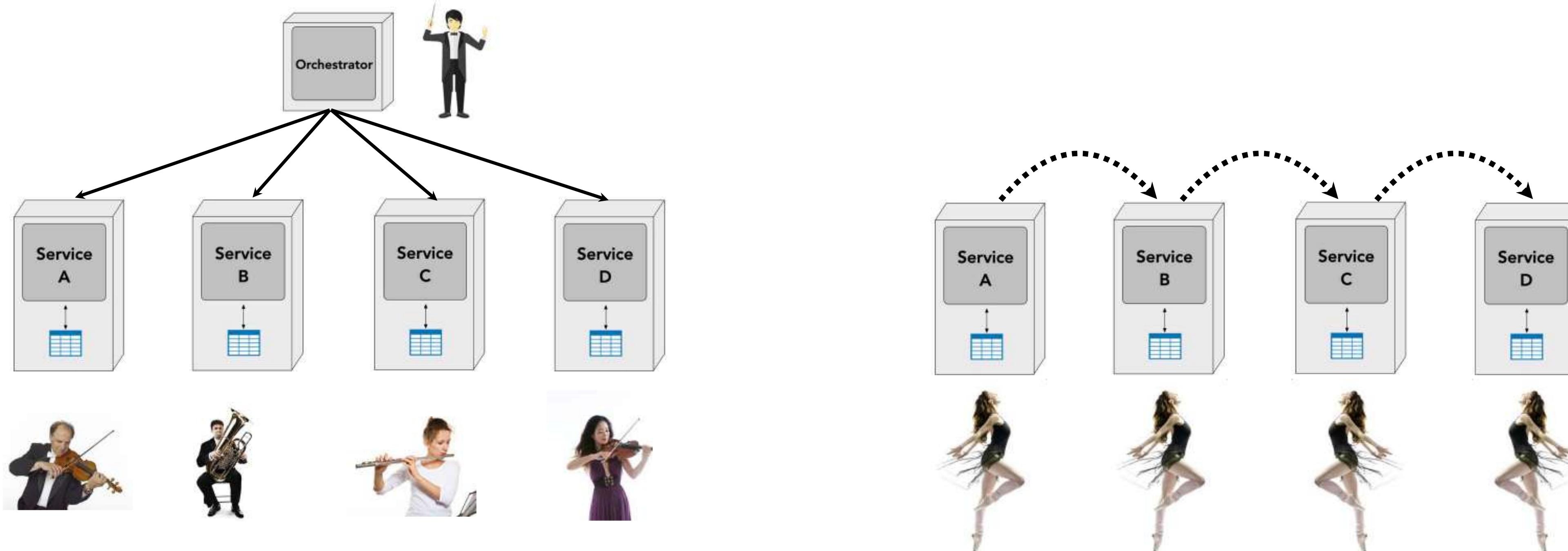
2. What fitness function could they write to prevent this from happening in the future?



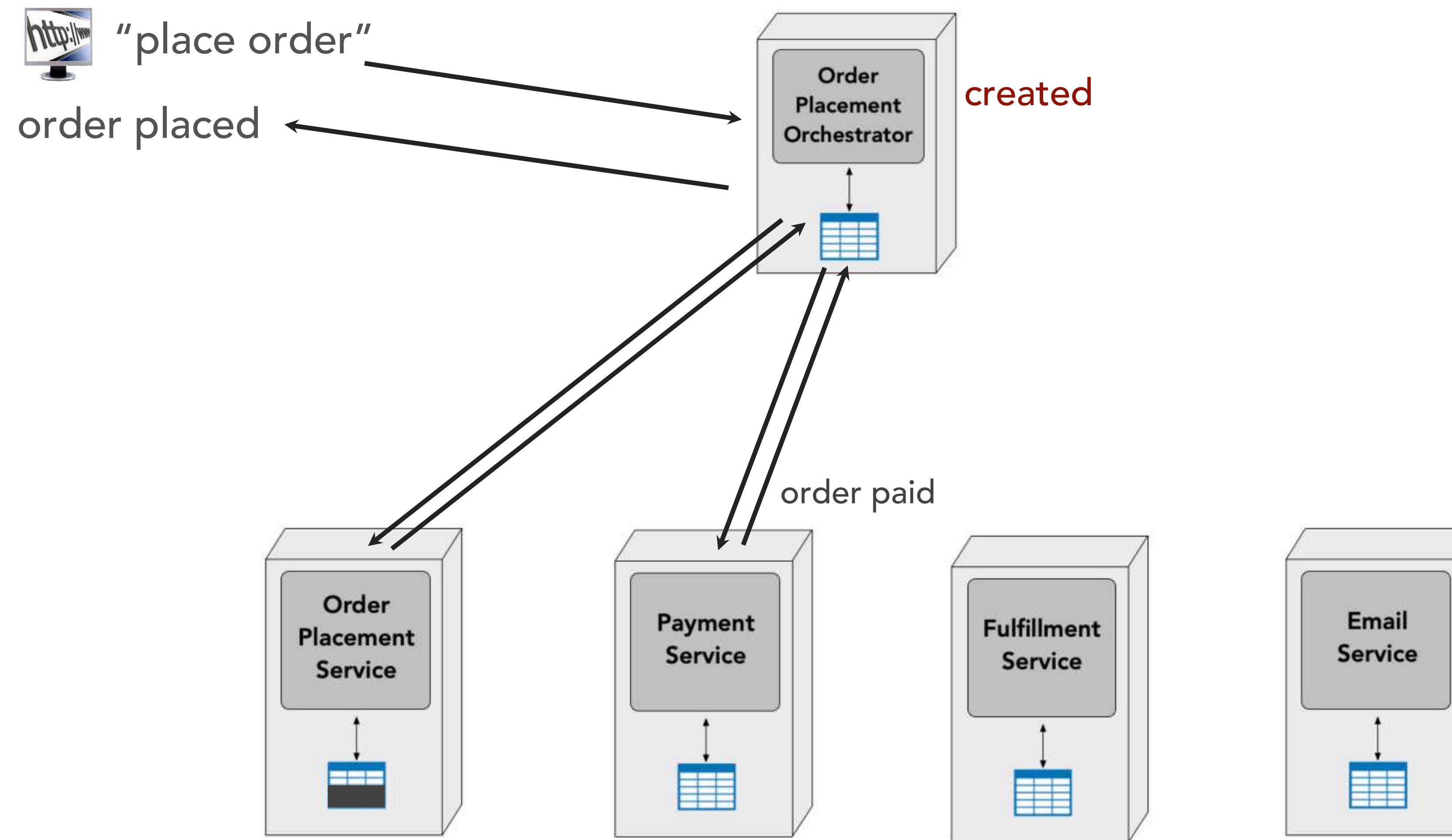


*How do I choose between
choreography and
orchestration for my
workflow?*

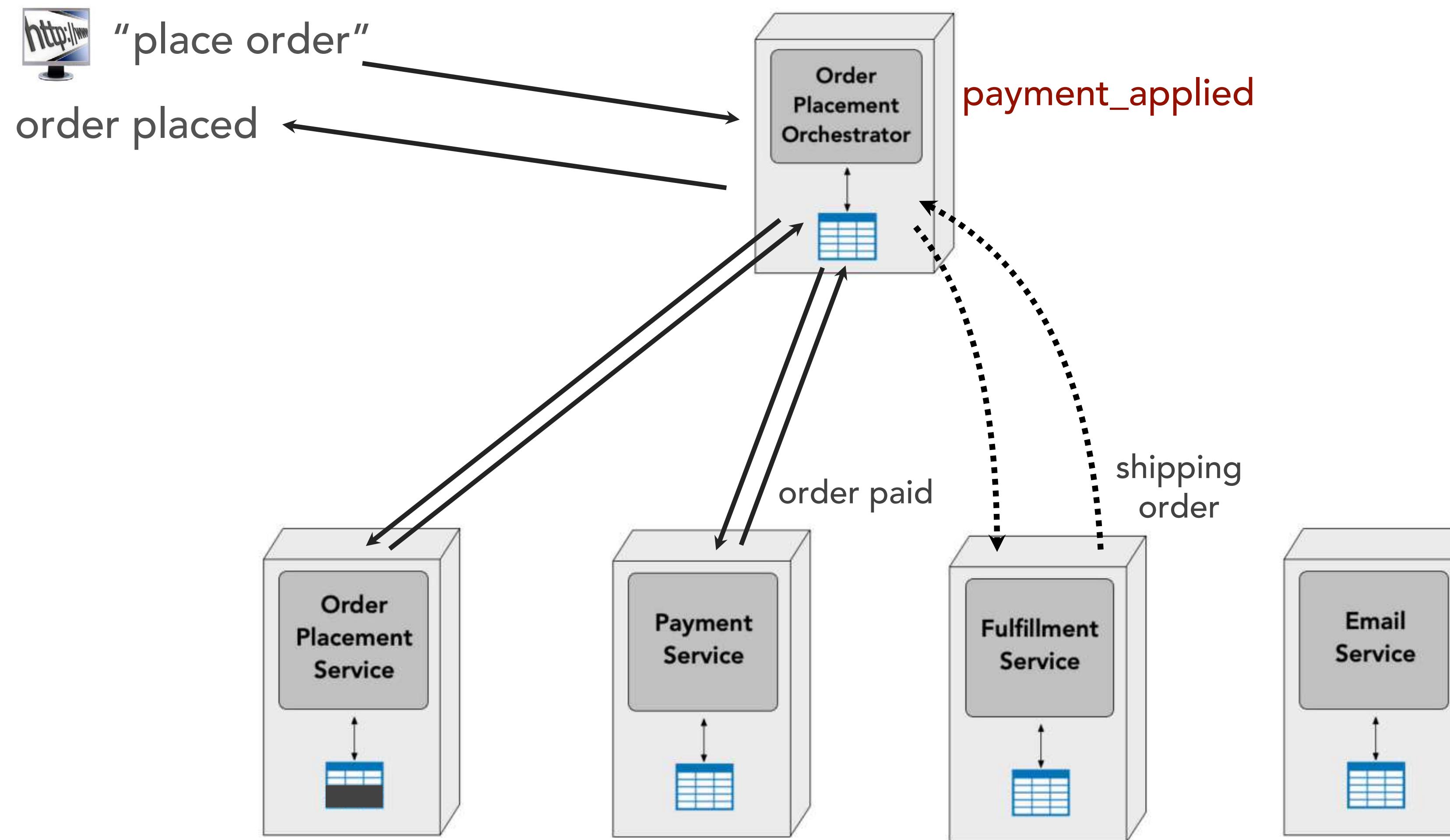
managing workflows



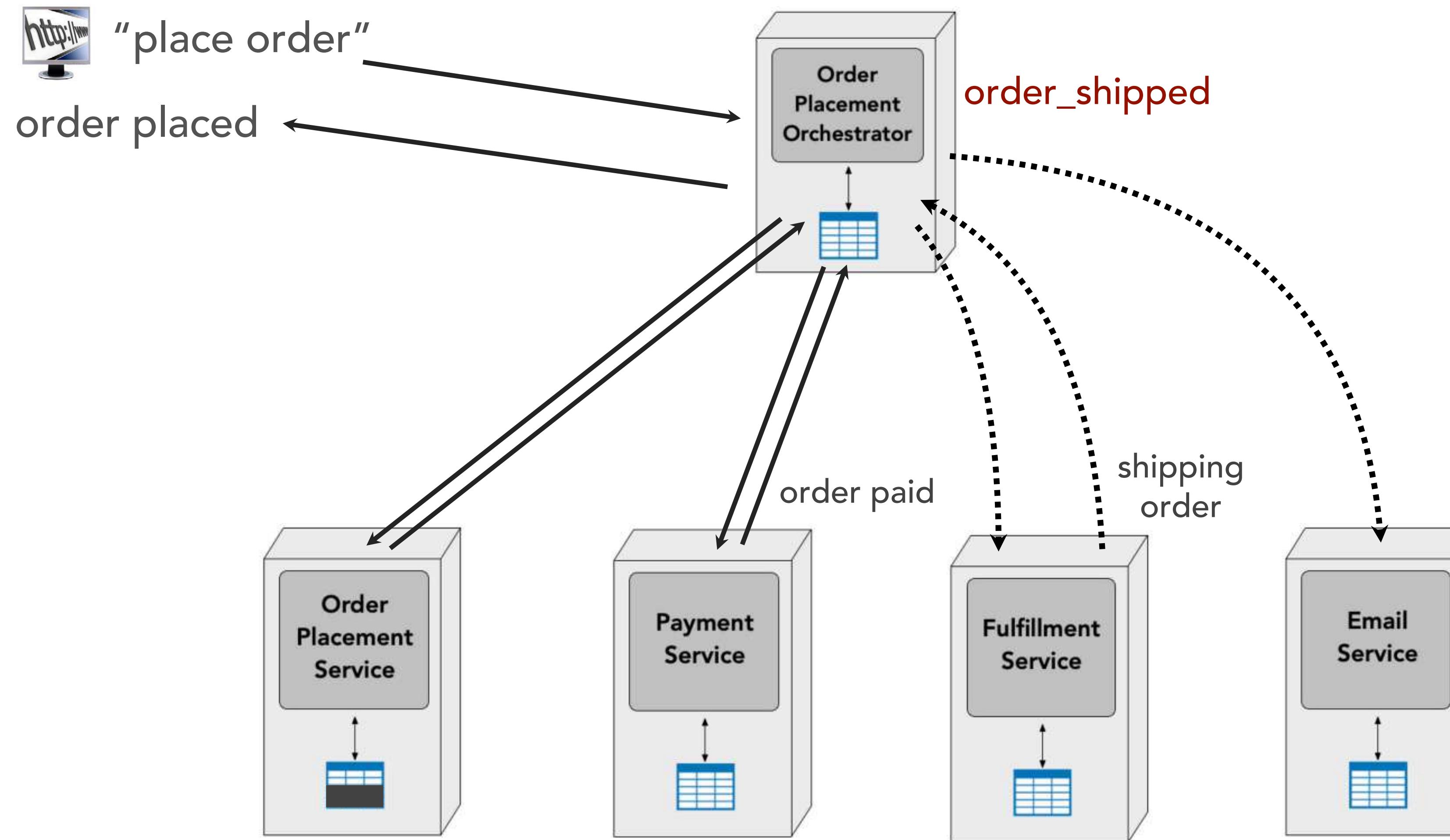
managing workflows



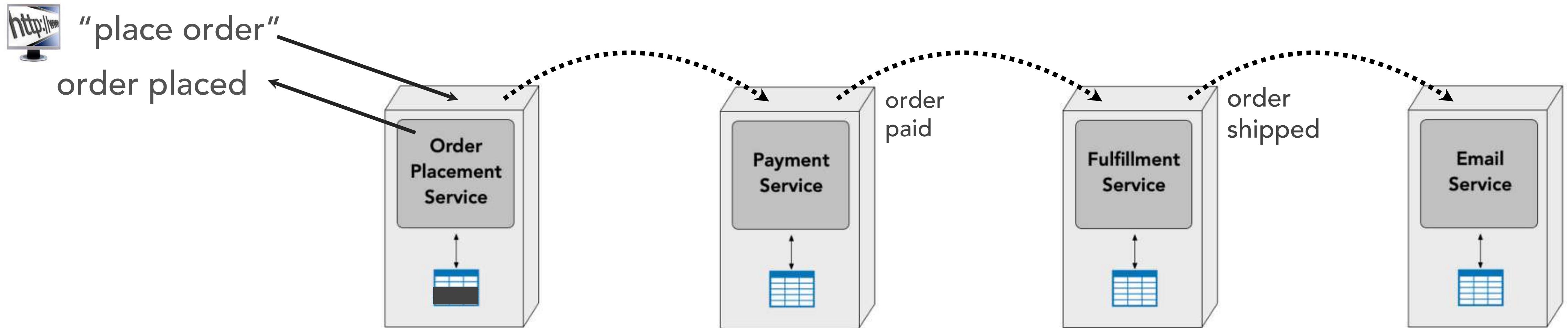
managing workflows



managing workflows

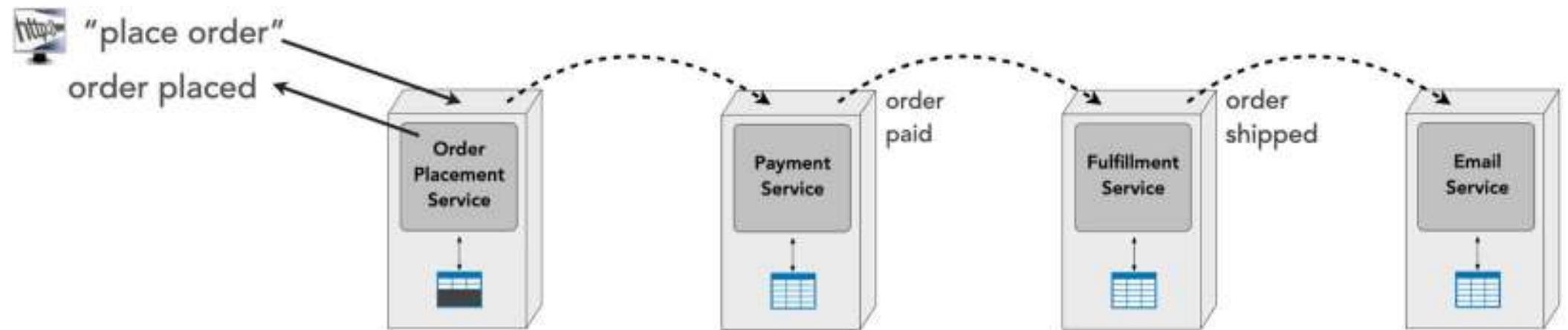


managing workflows

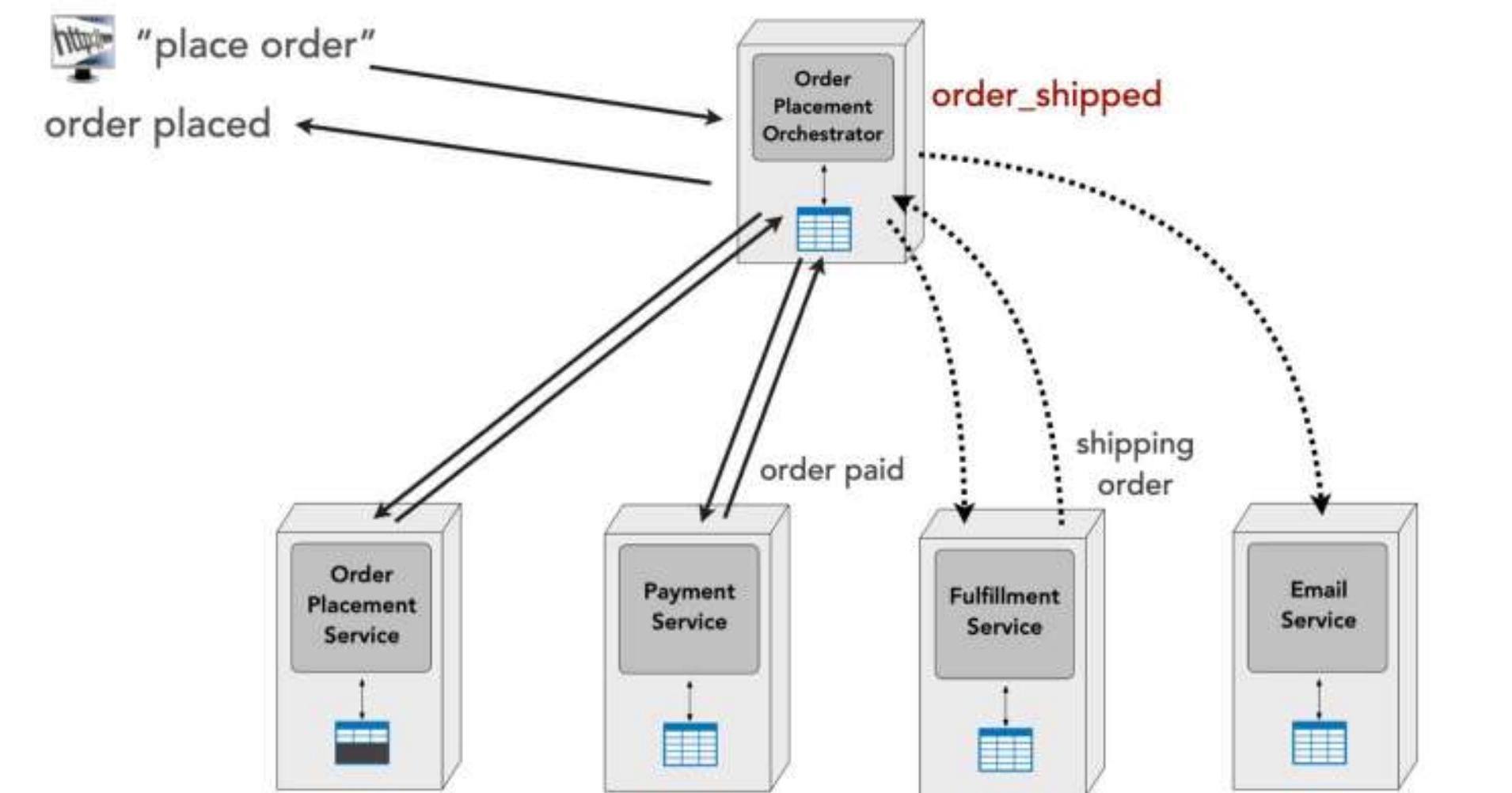


managing workflows

choreography

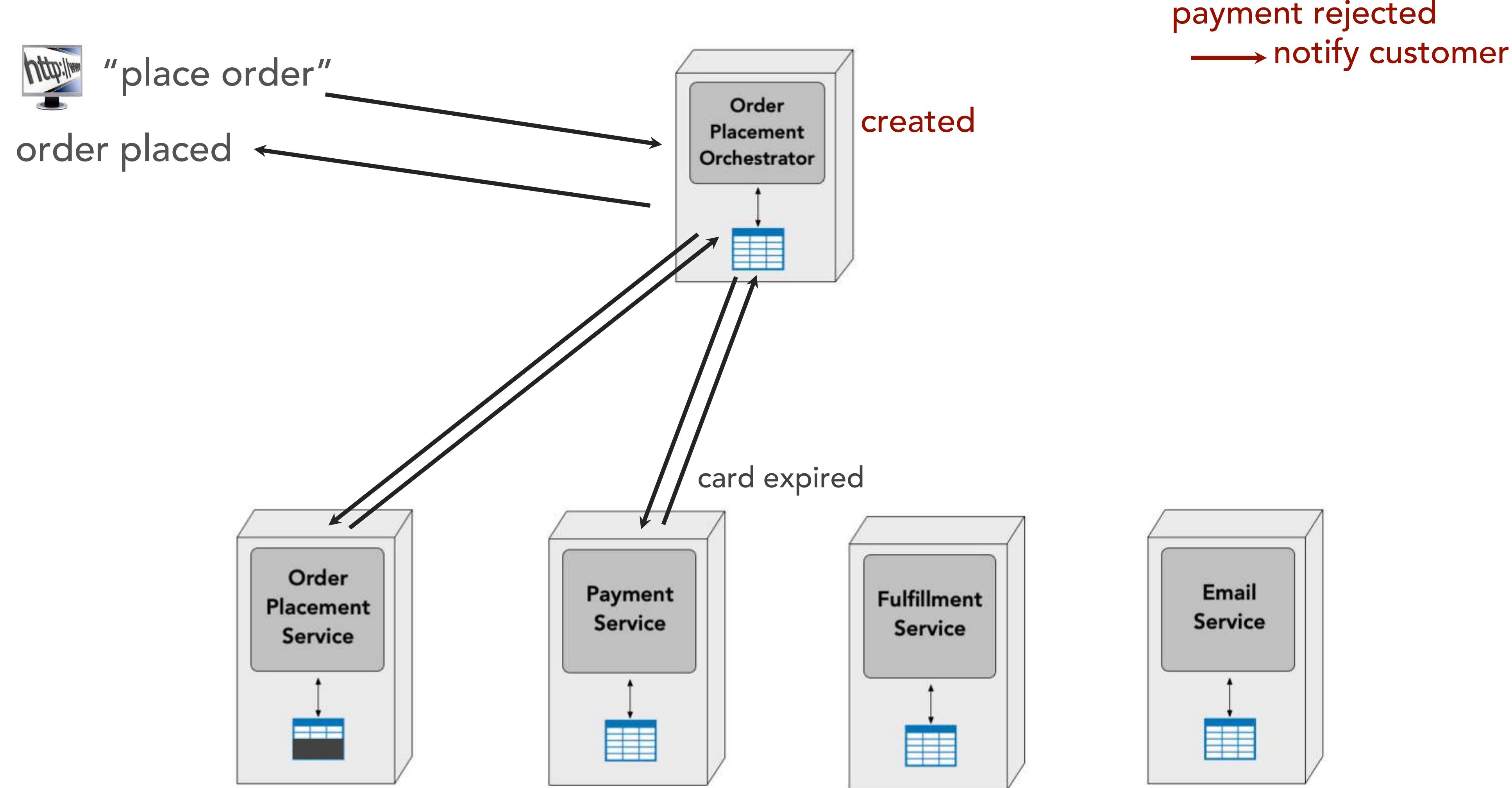


orchestration



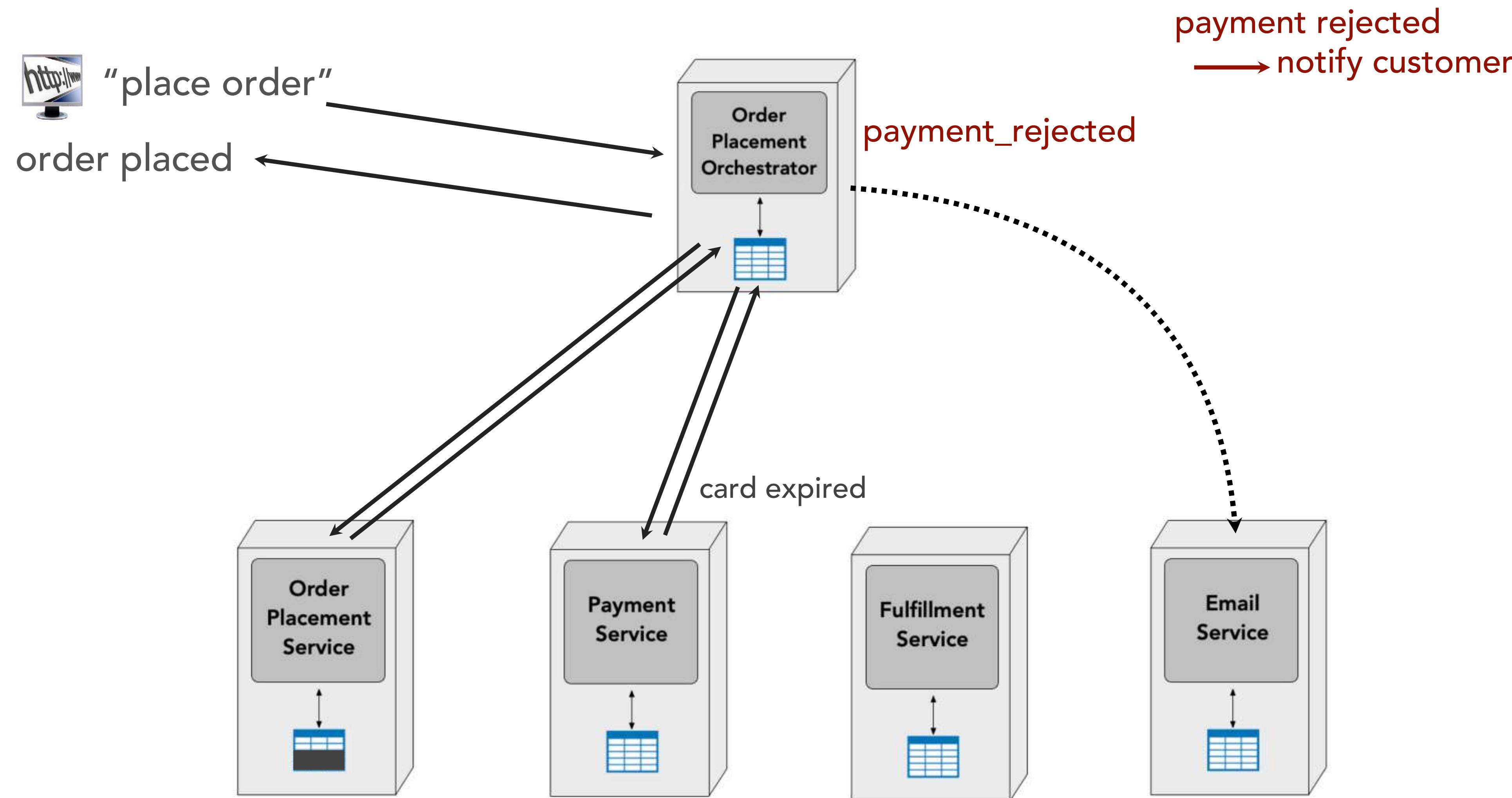
managing workflows

error handling



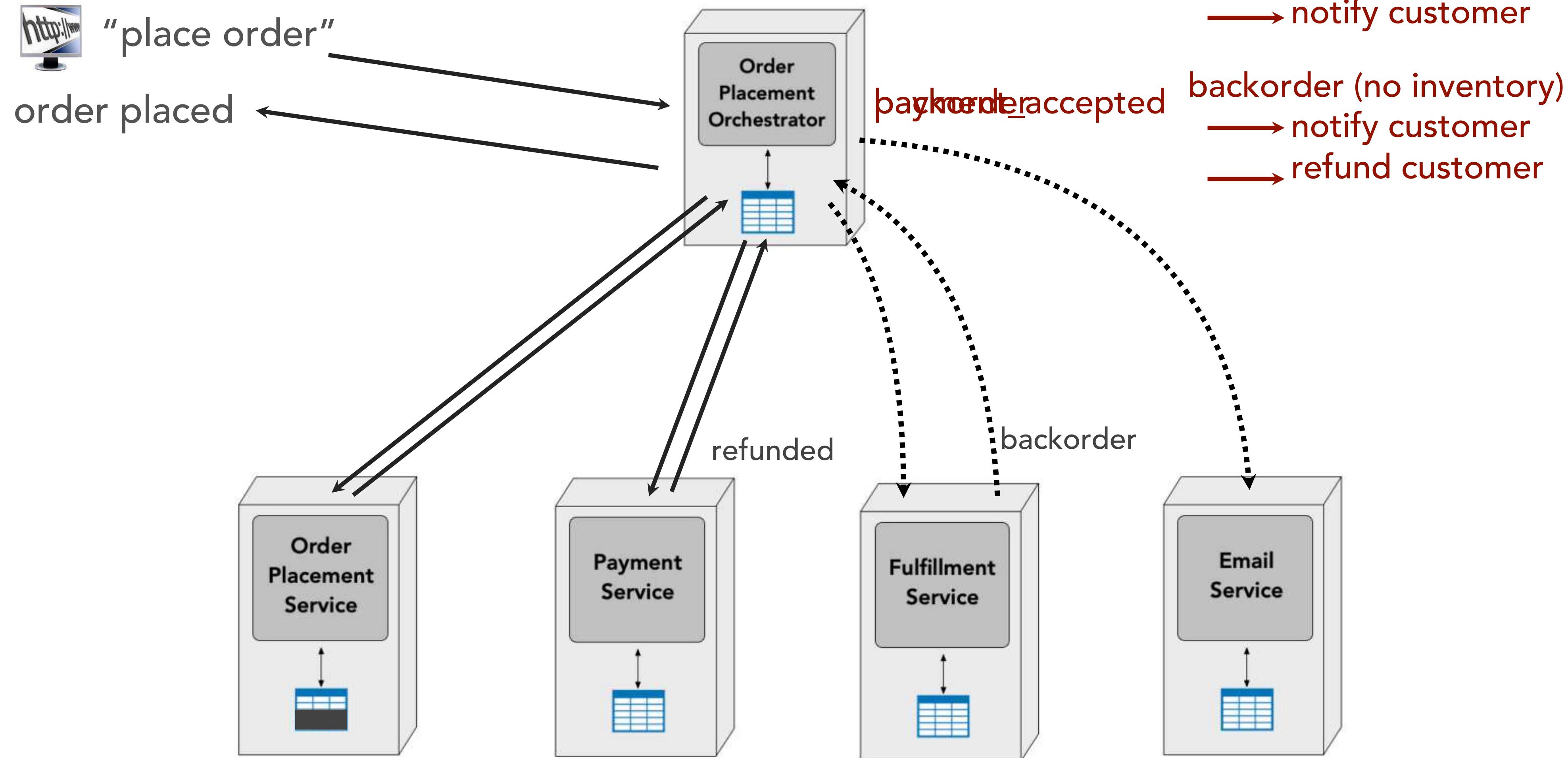
managing workflows

error handling



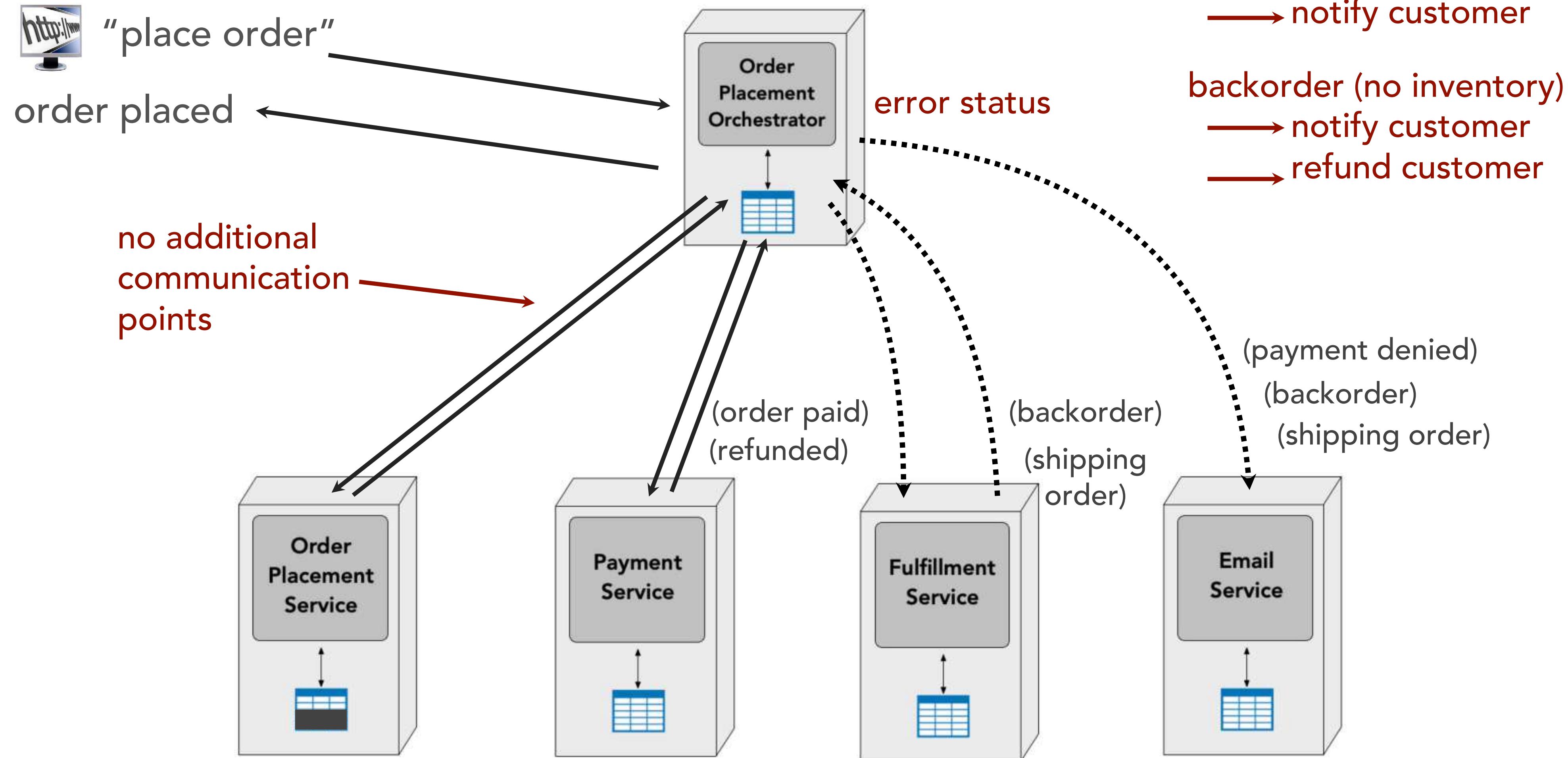
managing workflows

error handling



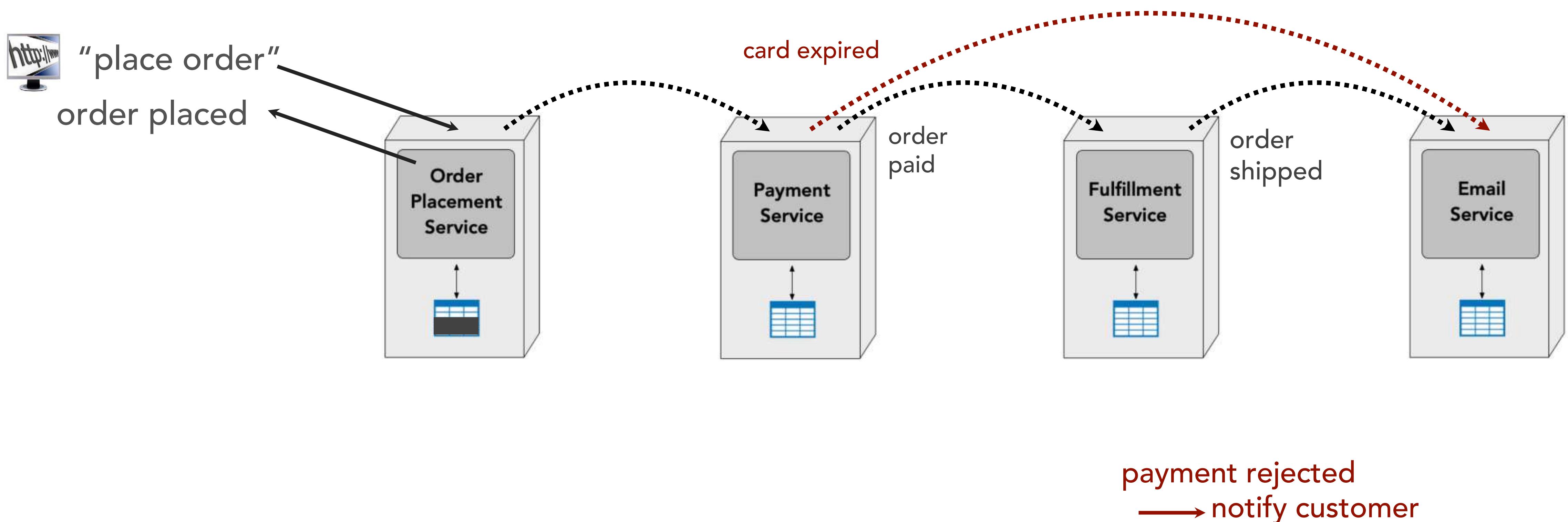
managing workflows

error handling



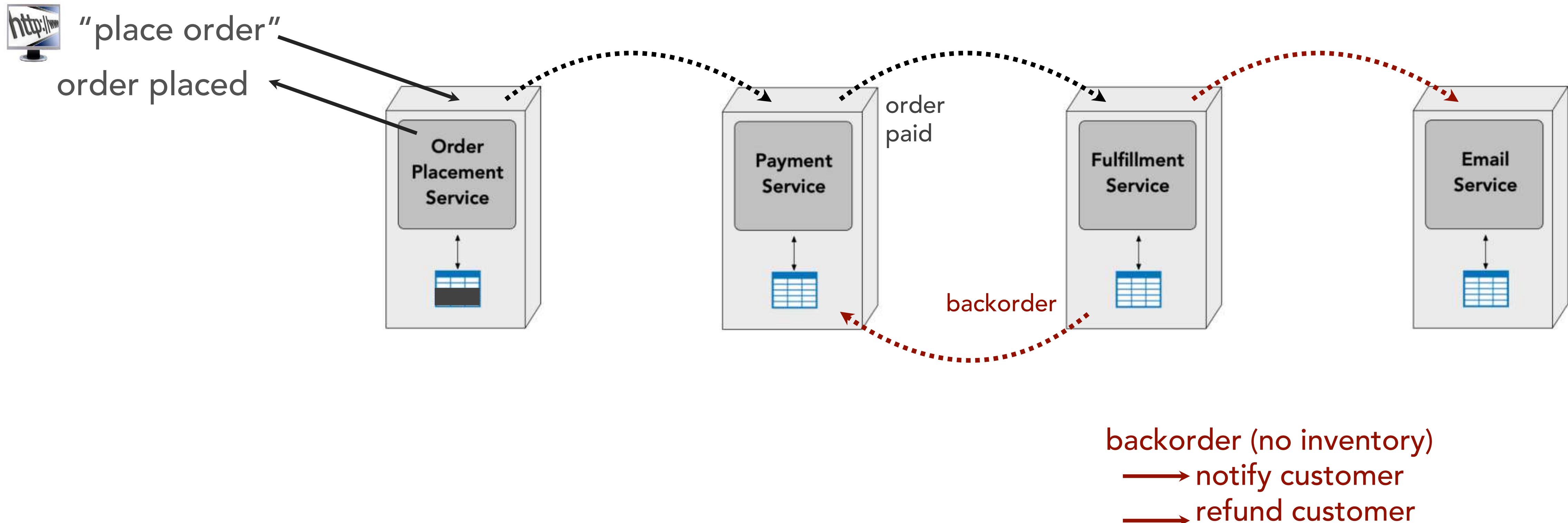
managing workflows

error handling



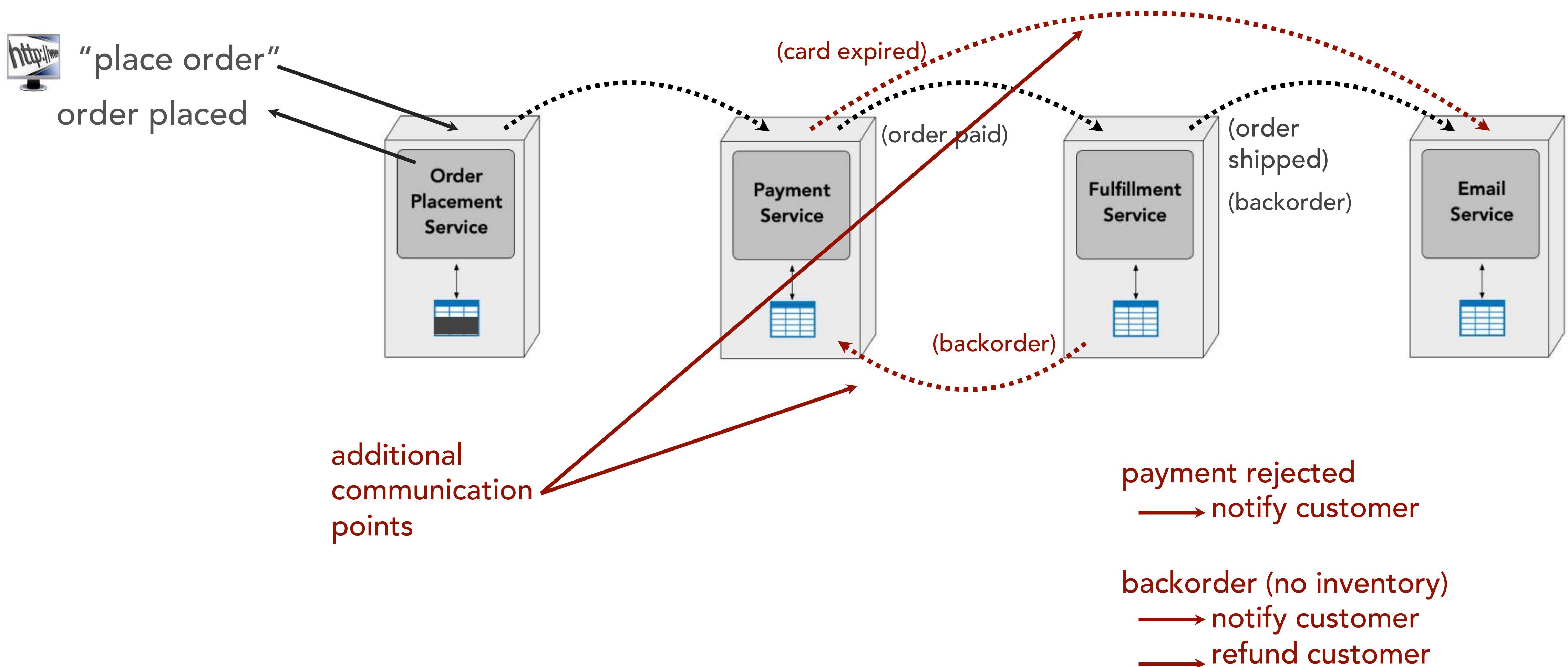
managing workflows

error handling

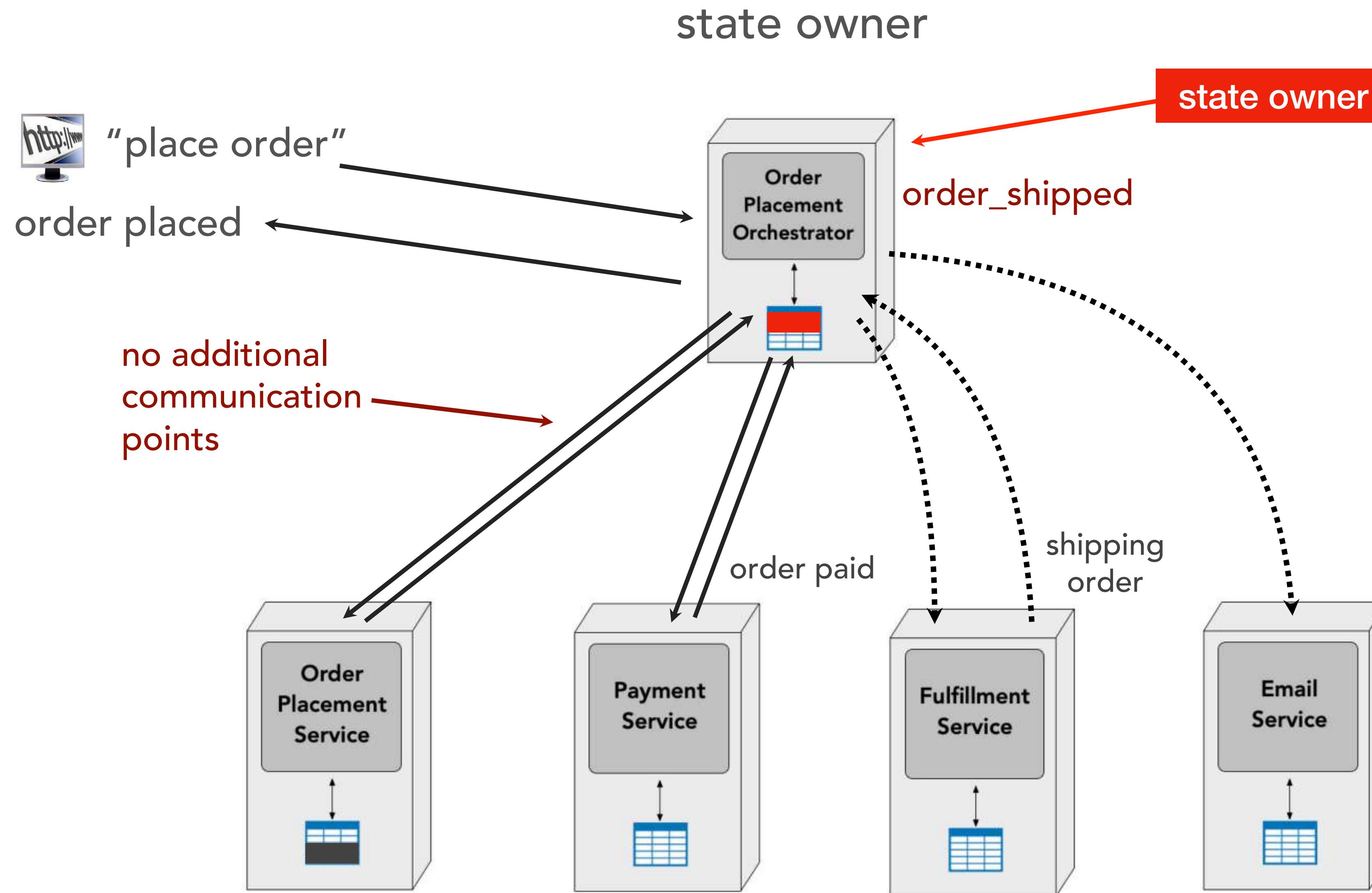


managing workflows

error handling

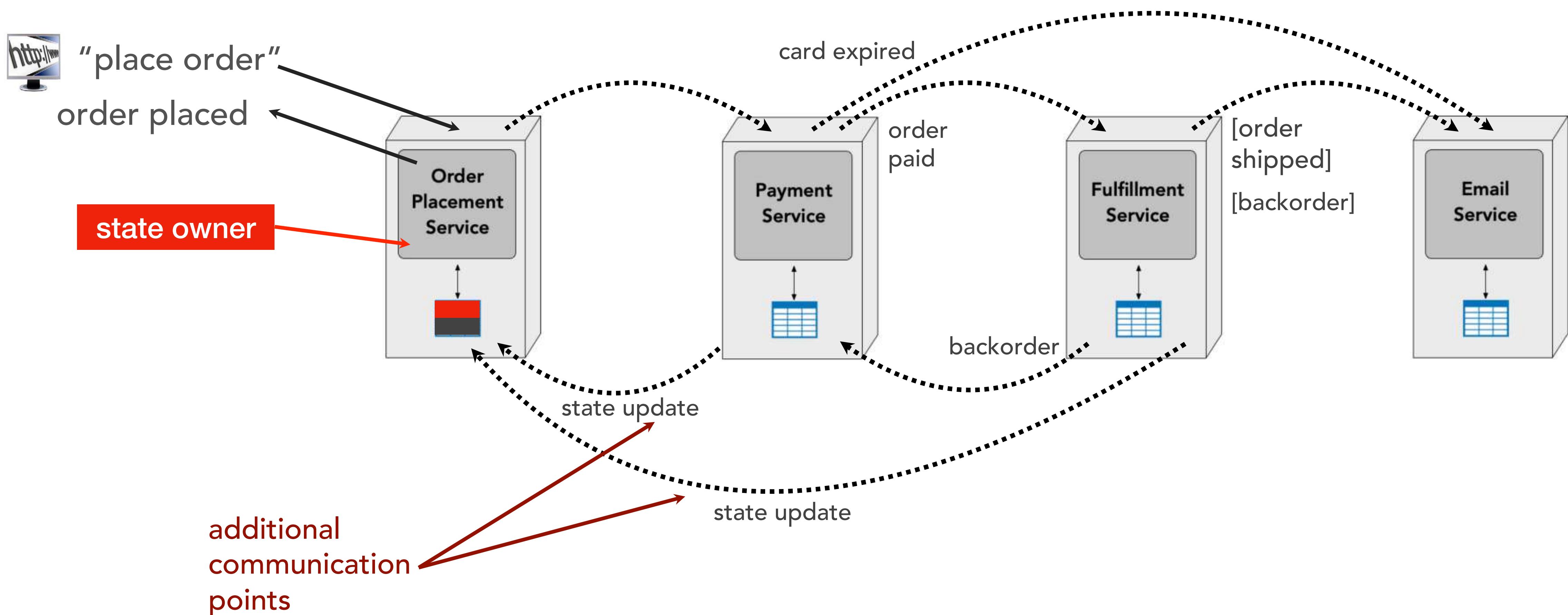


managing workflows



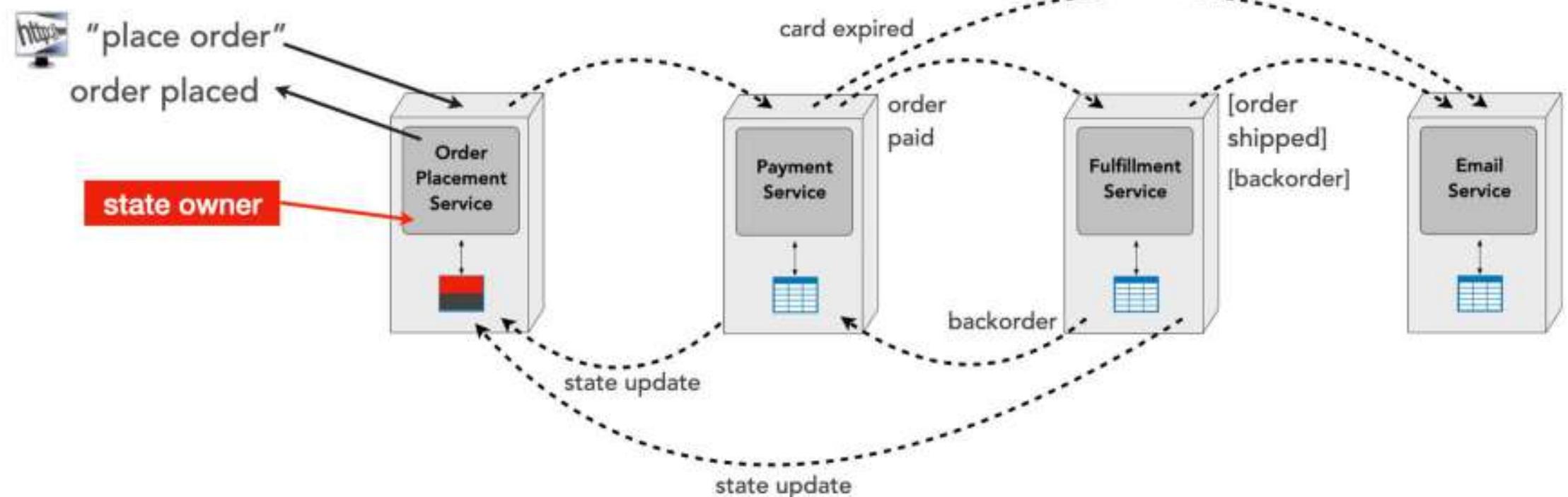
managing workflows

state owner

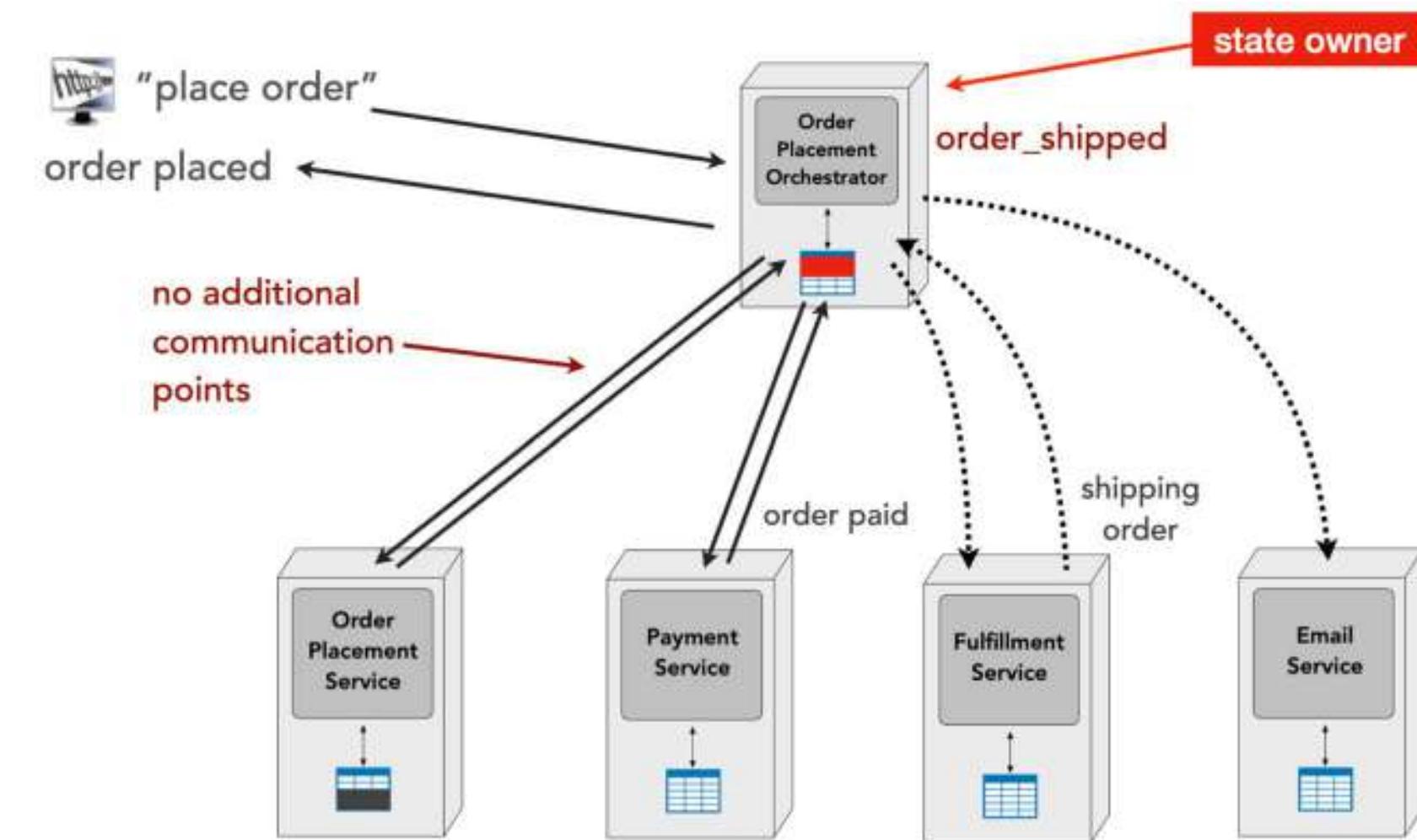


managing workflows

choreography

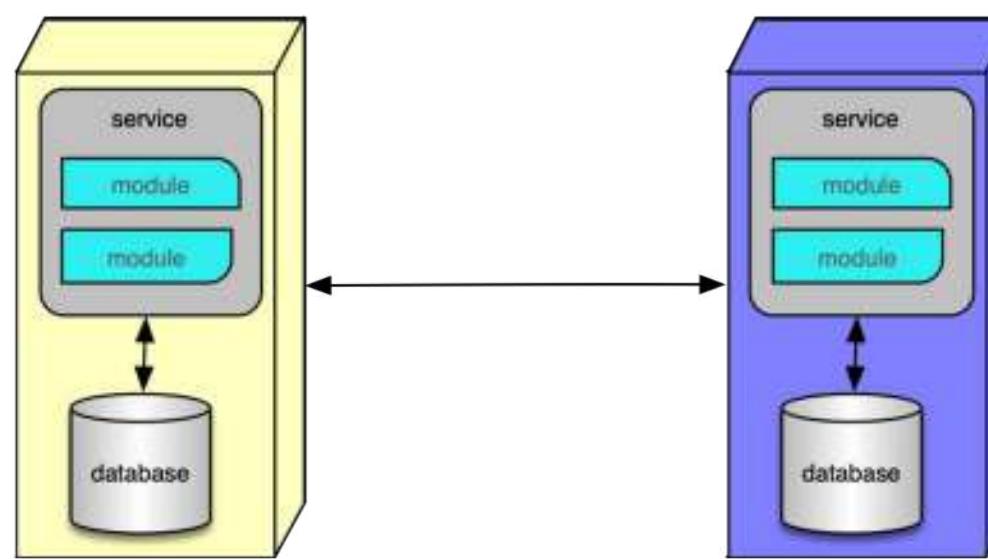
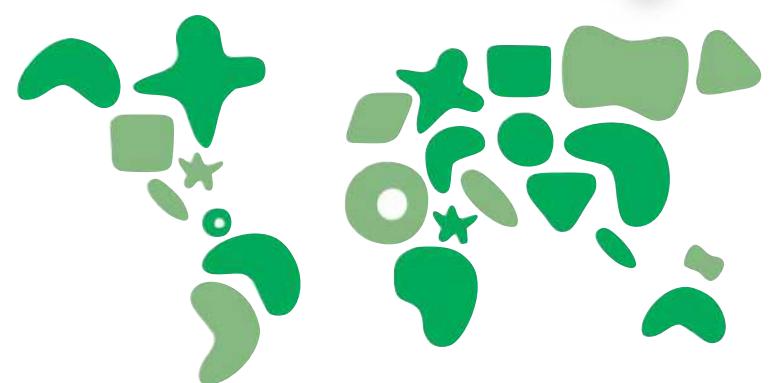


orchestration



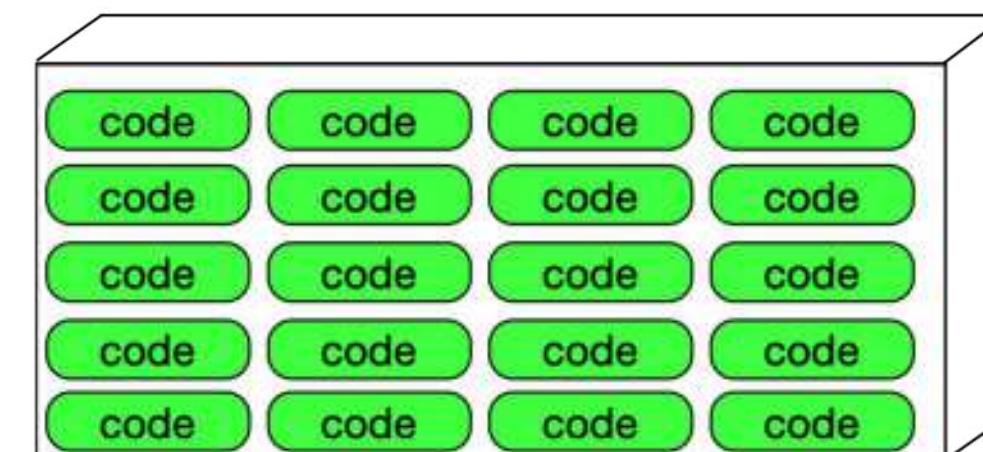
connascence properties

Locality

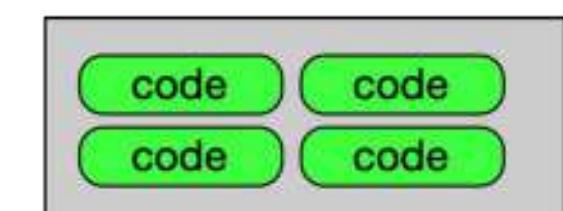


static

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



Scenario Exercise - Communication and Workflow

The primary ticket flow involves four services and is as follows:

Customer Facing Operations:

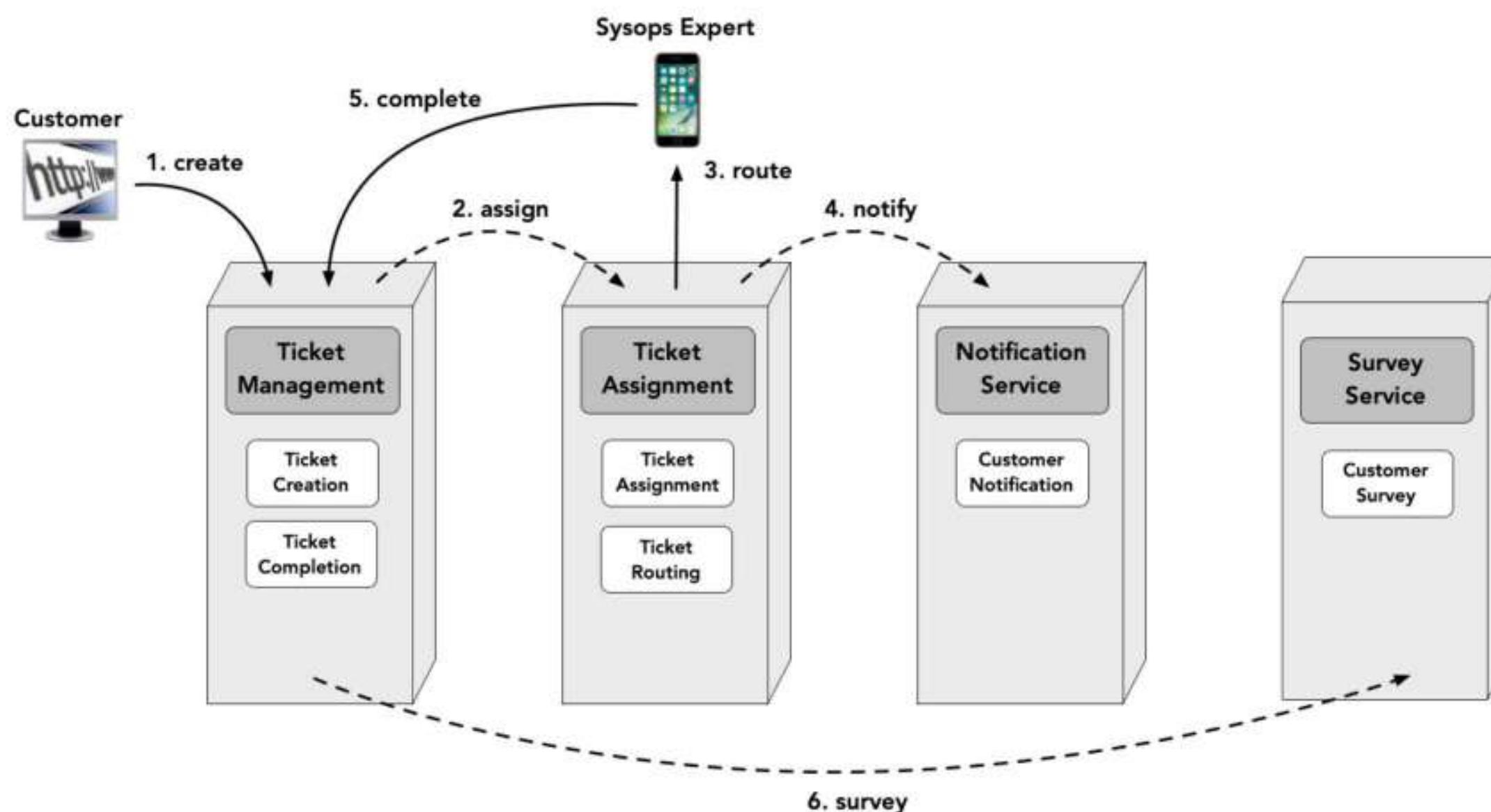
1. Customers submits a trouble ticket through the ***Ticket Management*** service and receive a ticket number.

Background Operations:

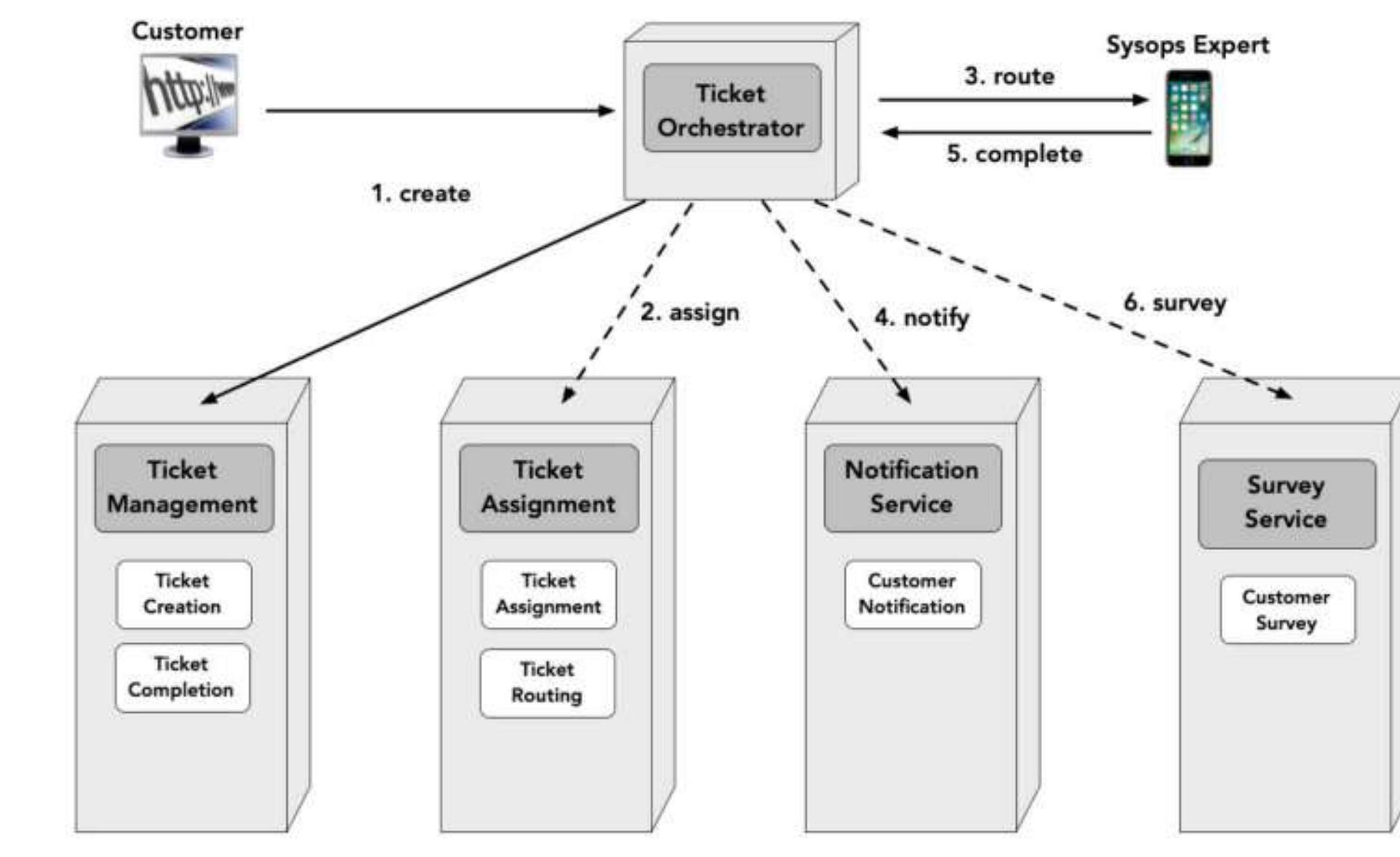
2. The ***Ticket Assignment*** service finds the right sysops expert for the trouble ticket.
3. The ***Ticket Assignment*** service routes the trouble ticket to the systems experts mobile device.
4. The customer is notified via the ***Notification Service*** that the sysops expert is on their way to fix the problem.
5. The expert fixes the problem and marks the ticket as complete, which is sent to the ***Ticket Management*** service.
6. The ***Ticket Management*** service communicates with the ***Survey Service*** to tell the customer to fill out the survey.

Scenario Exercise - Communication and Workflow

Which of the following communication styles would you choose and why? (solid lines are synchronous communication, dotted lines are asynchronous).



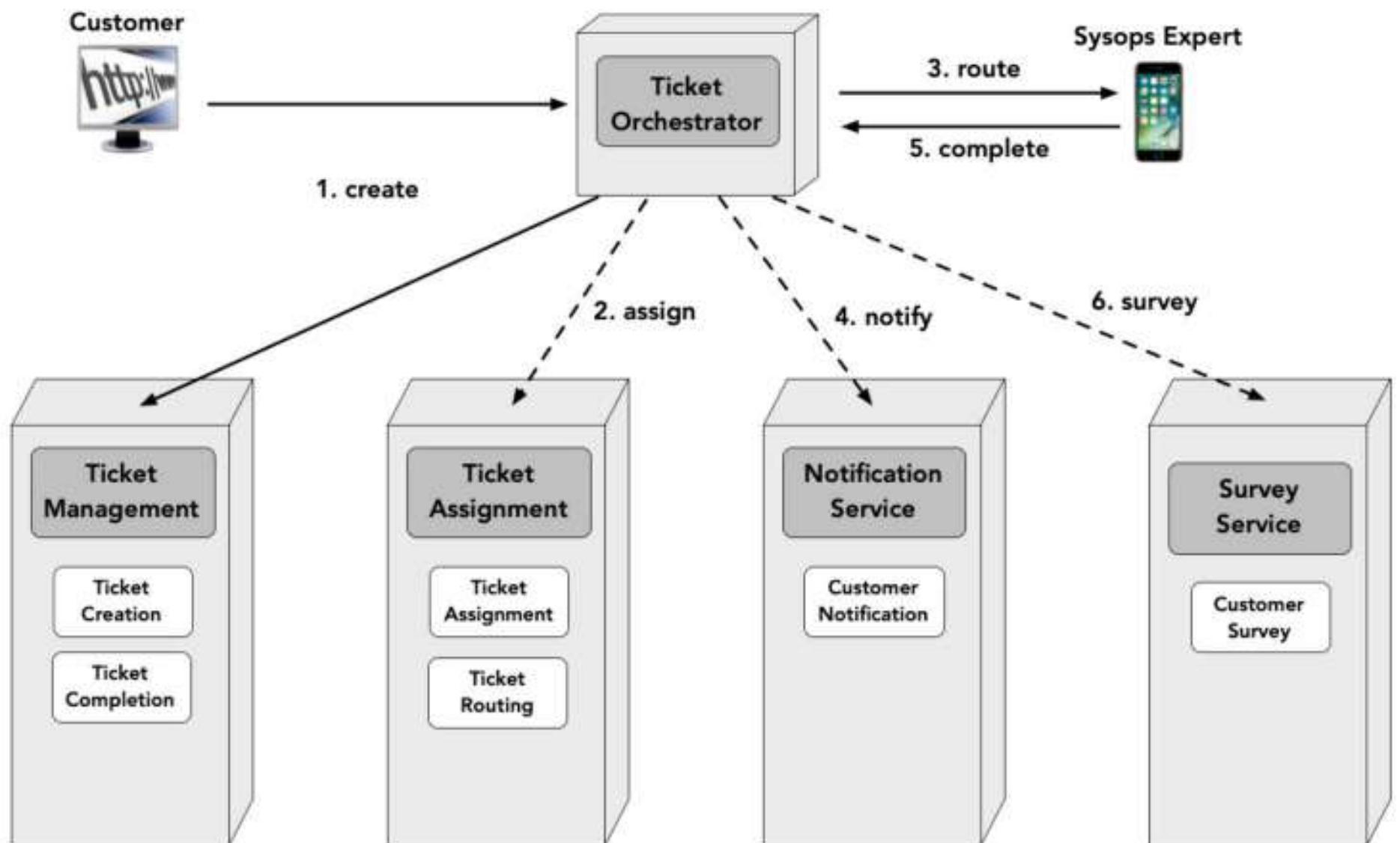
option 1: choreography



option 2: orchestration

Scenario Exercise - Communication and Workflow (Instructor)

Which of the following communication styles would you choose and why? (solid lines are synchronous communication, dotted lines are asynchronous).



option 2: orchestration

Justification:

Desire to always know ticket state (existing complaints about lost tickets and the wrong expert arriving for the job)

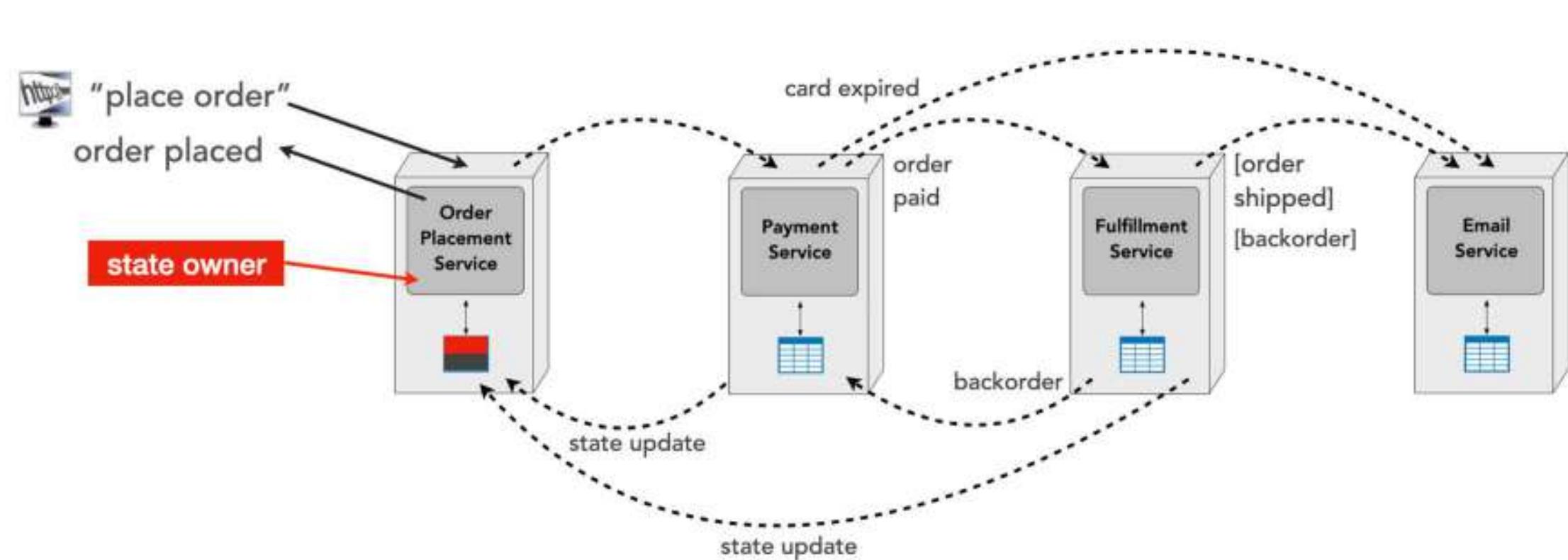
Error handling for reassignment, ticket cancellation, resend survey, lost connection to sysops expert on mobile phone (out of coverage area or dropped connection)



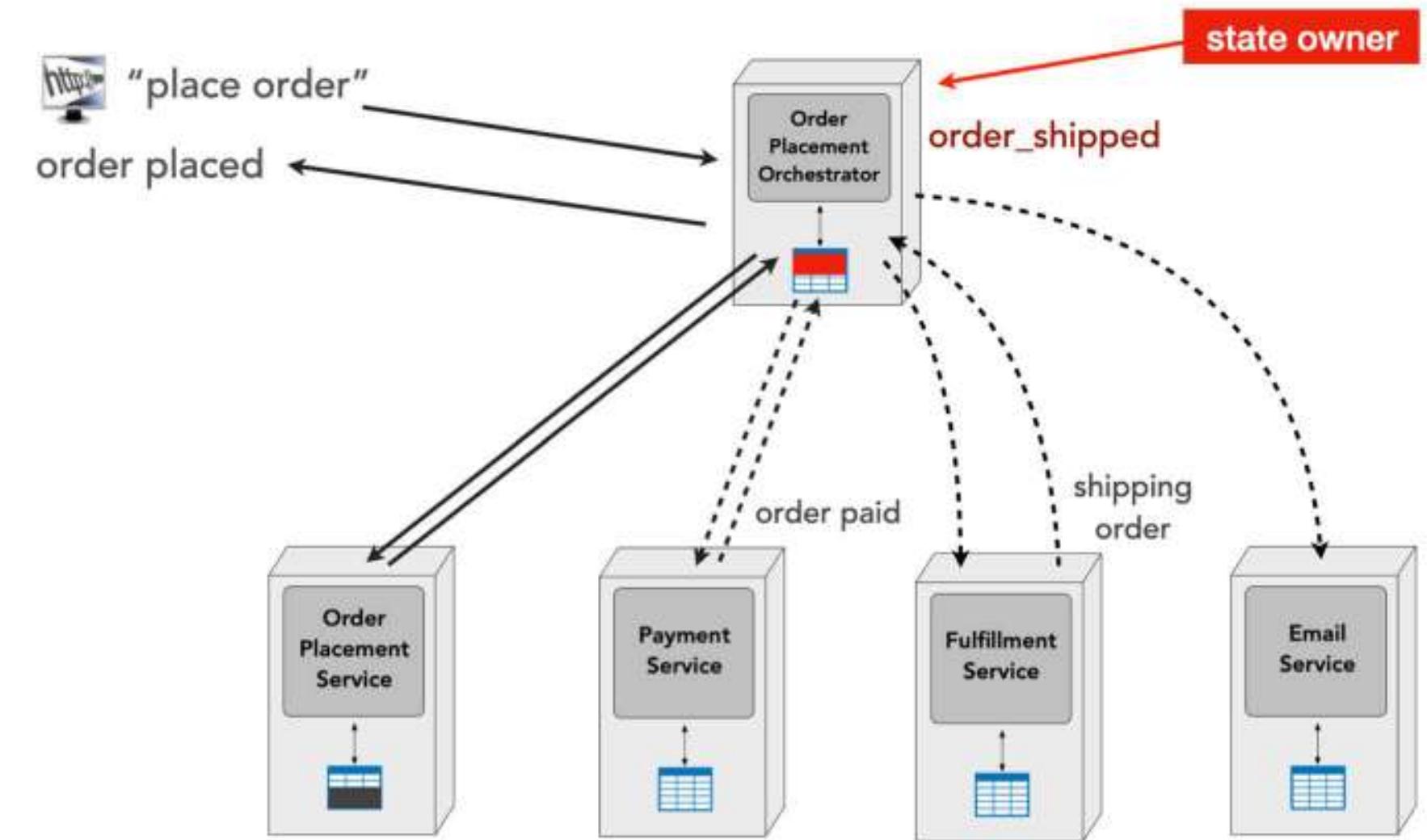
*How do I create systems with
high semantic coupling but
low syntactic coupling?*

managing workflows

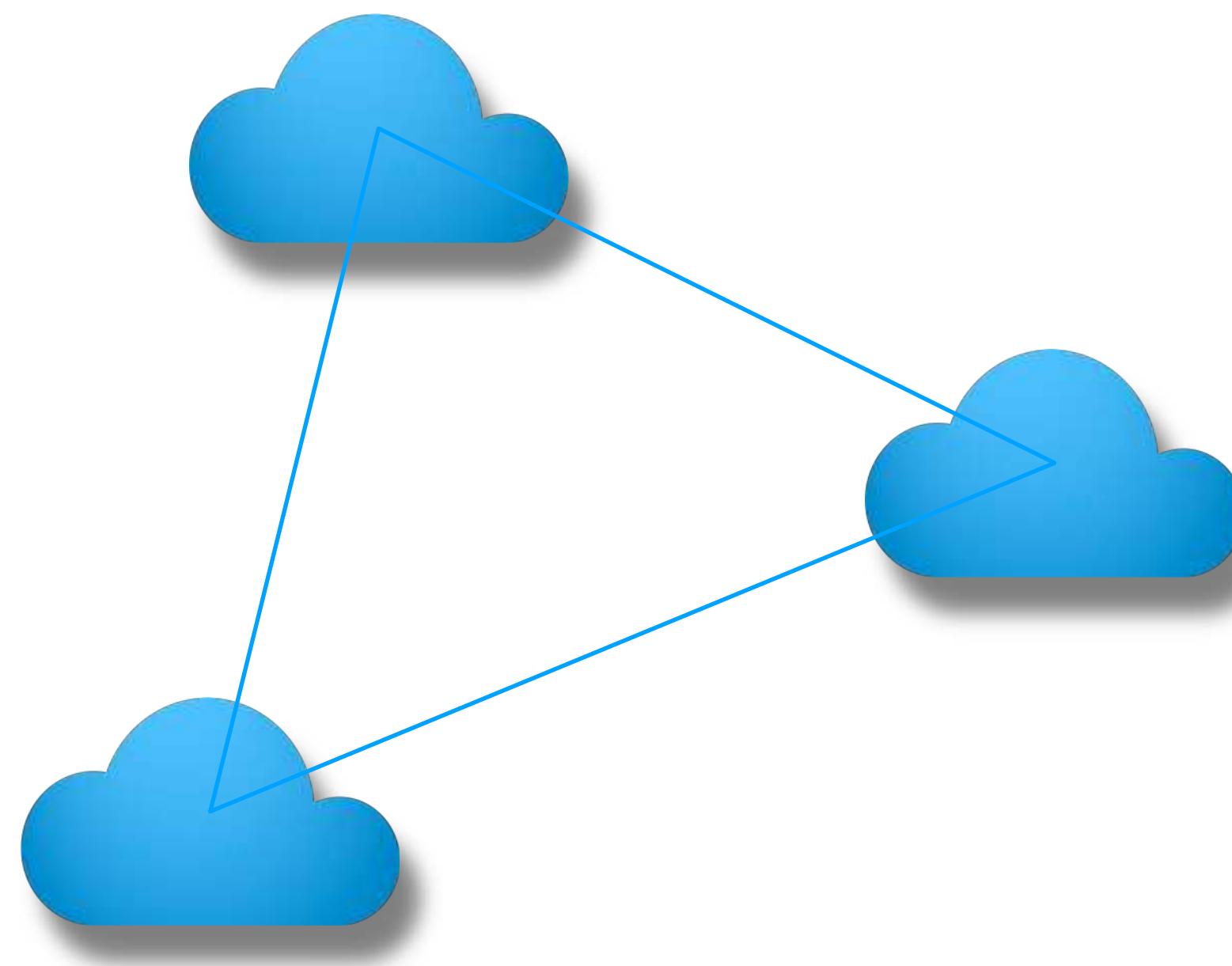
choreography



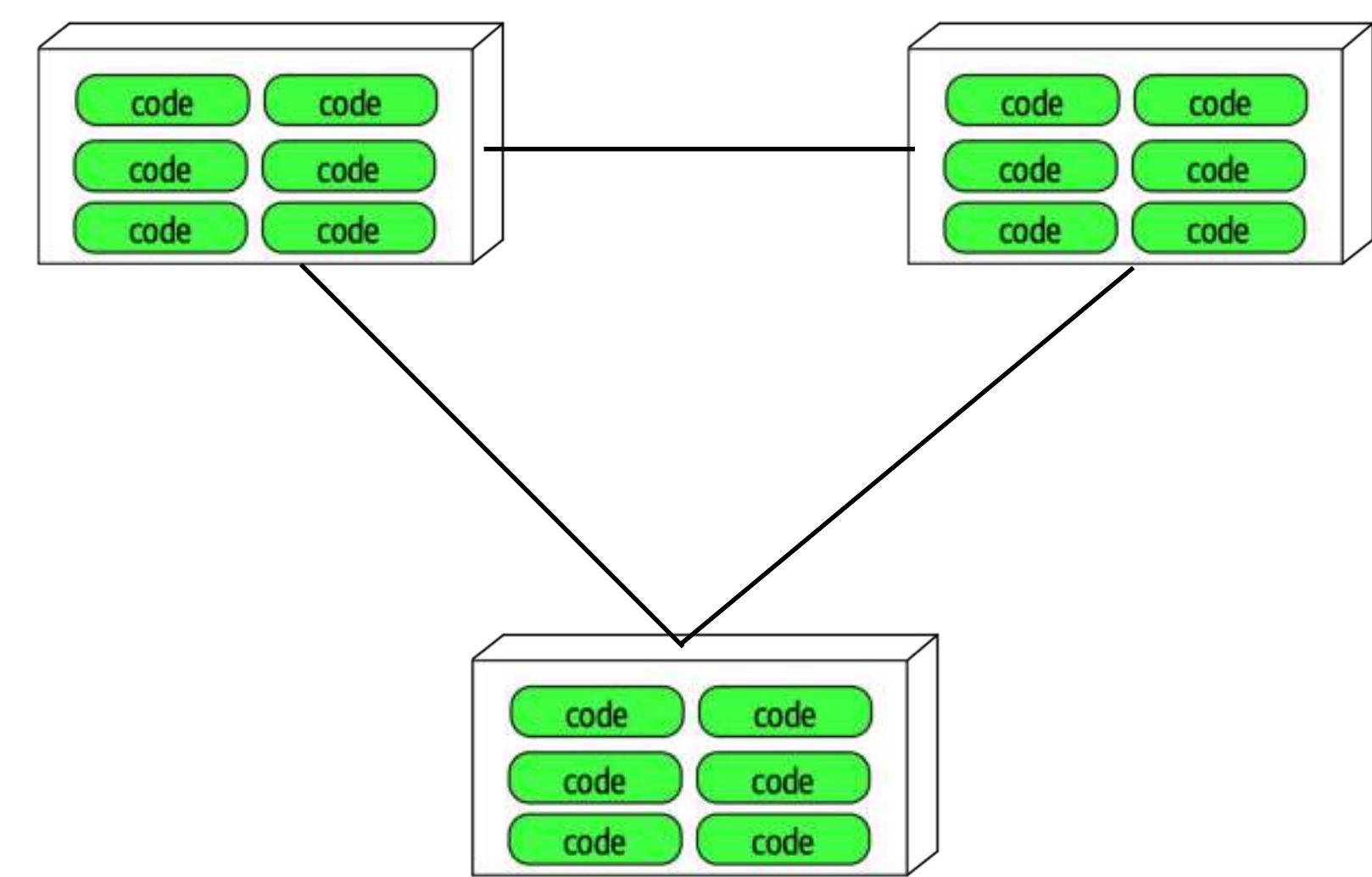
orchestration



coupling

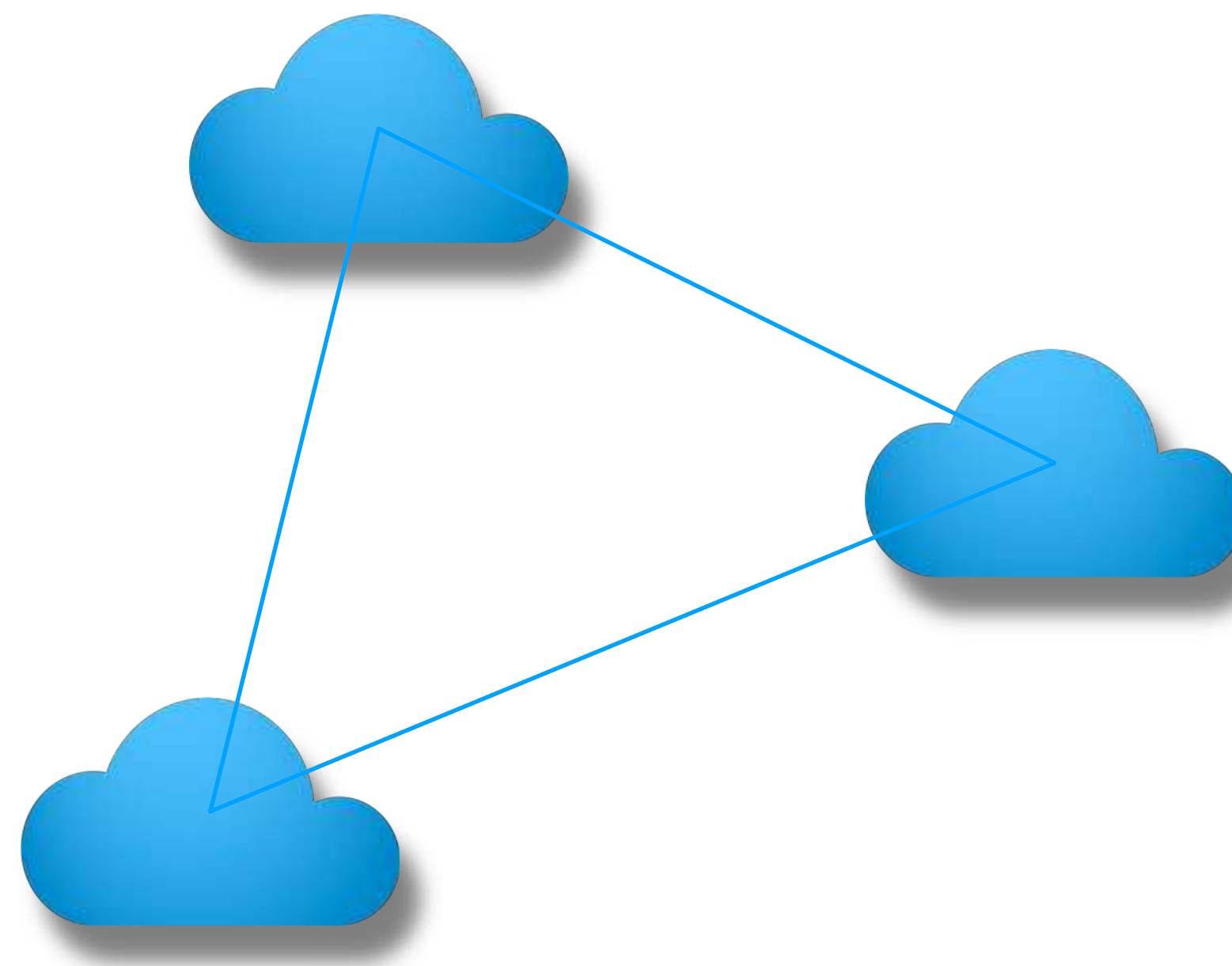


semantic

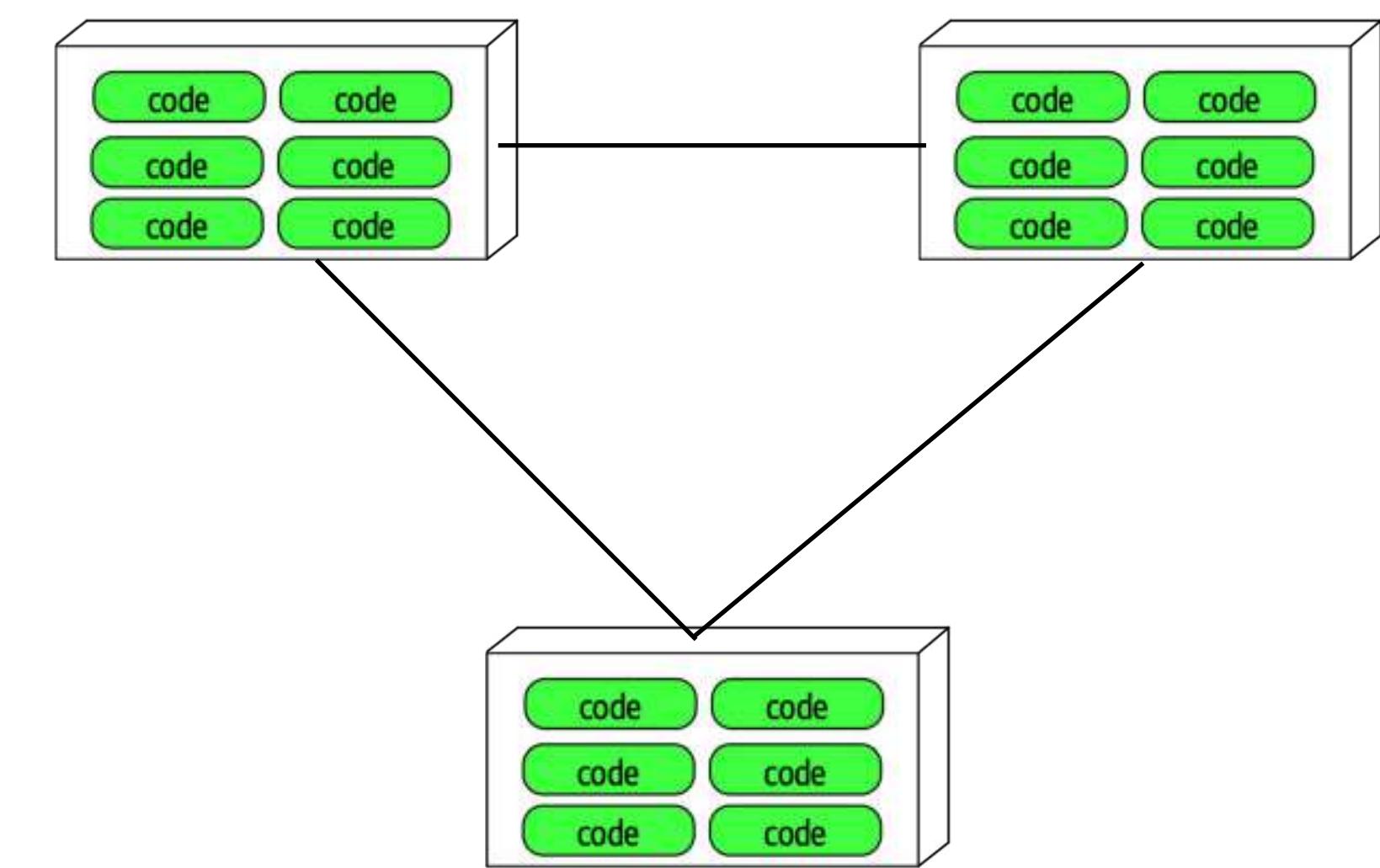


syntactic

coupling

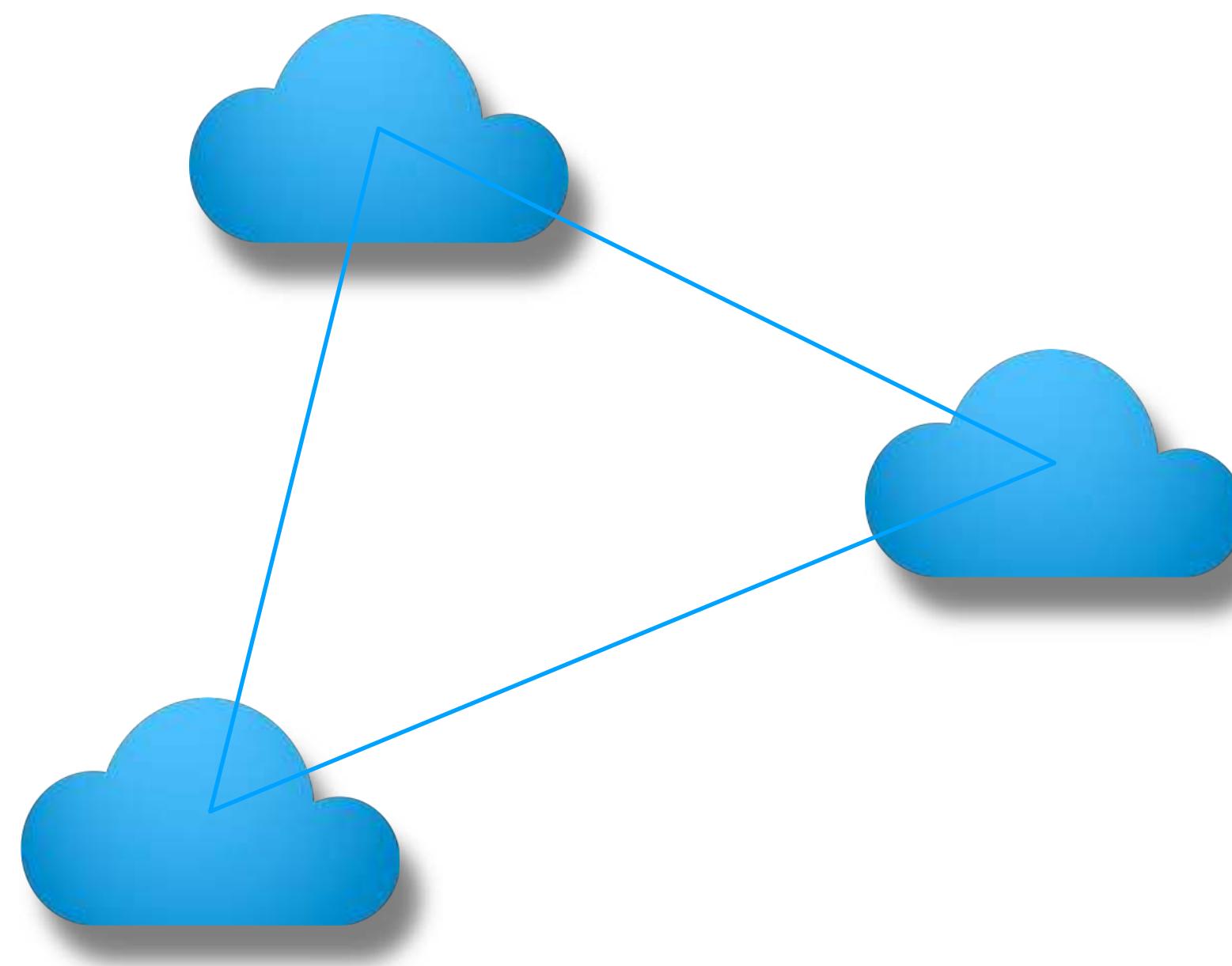


semantic
coupling



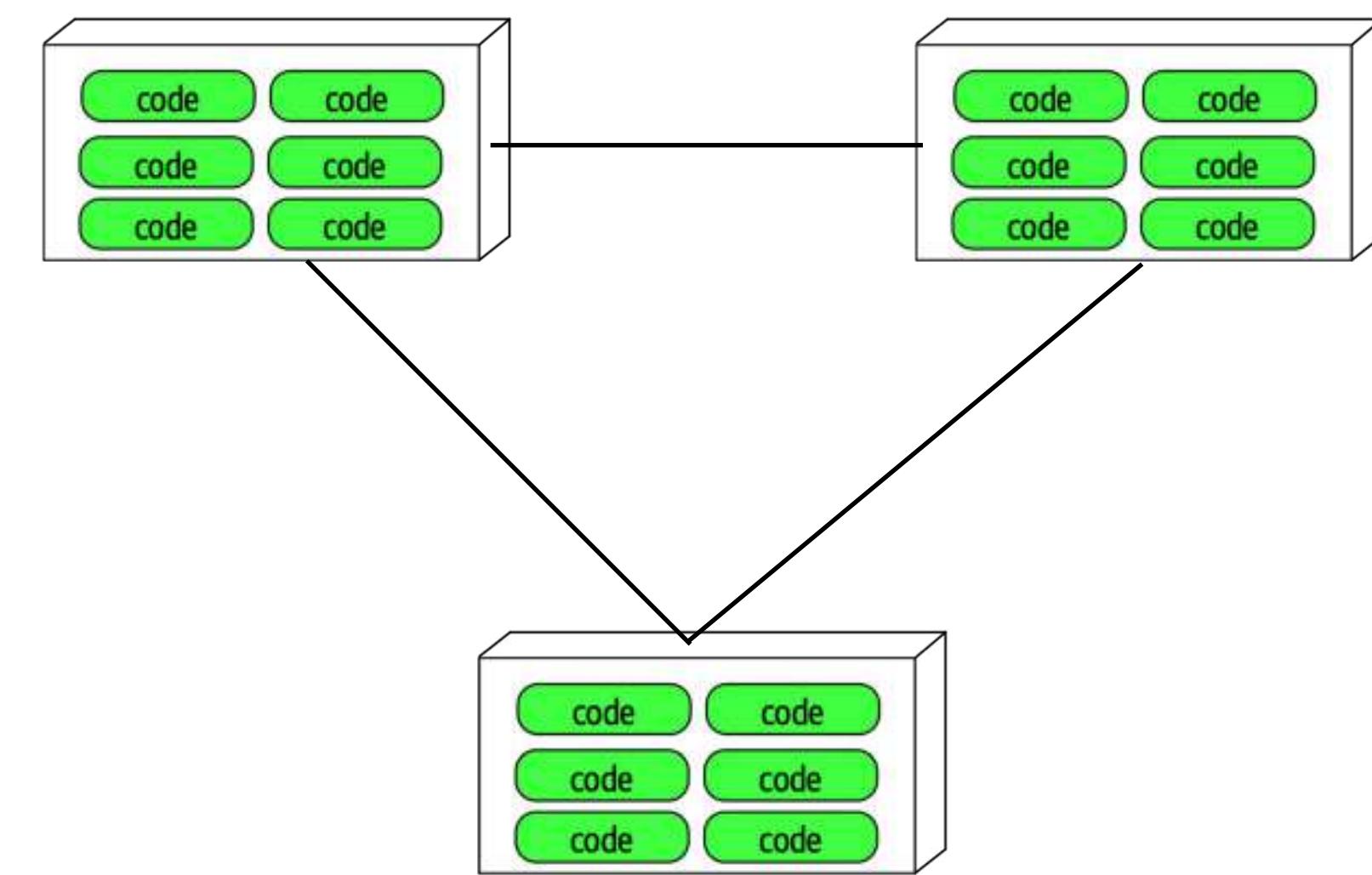
syntactic
coupling

coupling



semantic

syntactic



quantum connascence



static

contracts



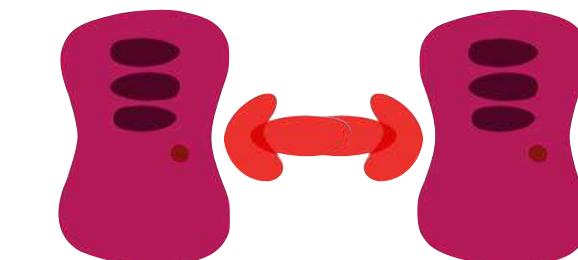
tight

loose

synchronous

quantum

asynchronous



how quanta communicate

impact on operational
(and other) architecture characteristics

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

quantum connascence



static

contracts



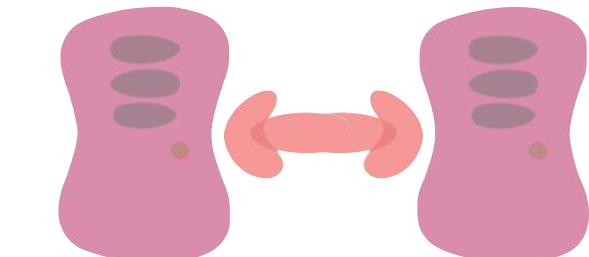
tight

loose

synchronous



quantum



asynchronous



how quanta communicate

impact on operational
(and other) architecture characteristics

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

contracts

tight



loose

method signatures

resources

name/value pairs

contracts

tight



loose

method signatures

resources

name/value pairs

SOAP

contracts

tight



loose

method signatures

resources

name/value pairs

SOAP

Buffer Protocols (gRPC)

RPC*

contracts

tight



loose

method signatures

resources

name/value pairs

SOAP

Buffer Protocols (gRPC)

RPC*

contracts

tight

loose

method signatures

resources

name/value pairs

SOAP

REST

Buffer Protocols (gRPC)

RPC*

contracts

tight

loose

method signatures

resources

name/value pairs

SOAP

REST

Buffer Protocols (gRPC)

GraphQL

RPC*

contracts

tight

loose

method signatures

resources

name/value pairs

SOAP

REST

JSON

Buffer Protocols (gRPC)

GraphQL

RPC*

tradeoffs

tight

loose

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

tradeoffs

tight

loose

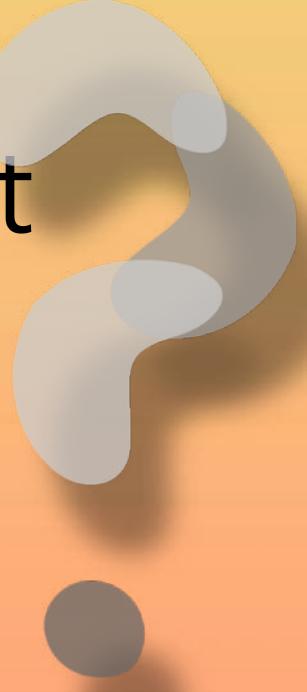
- * guaranteed contract fidelity
- * distinct versions (+ or -?)
- * build-time contracts
- * decoupled...
- * ...therefore better for decoupled architectures
- * contract management

tradeoffs

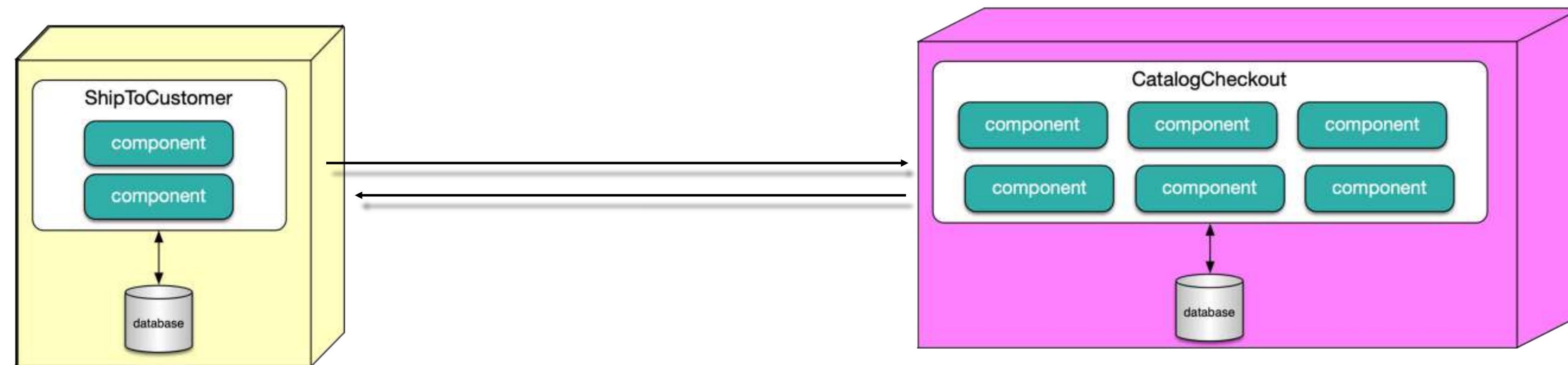
tight

loose

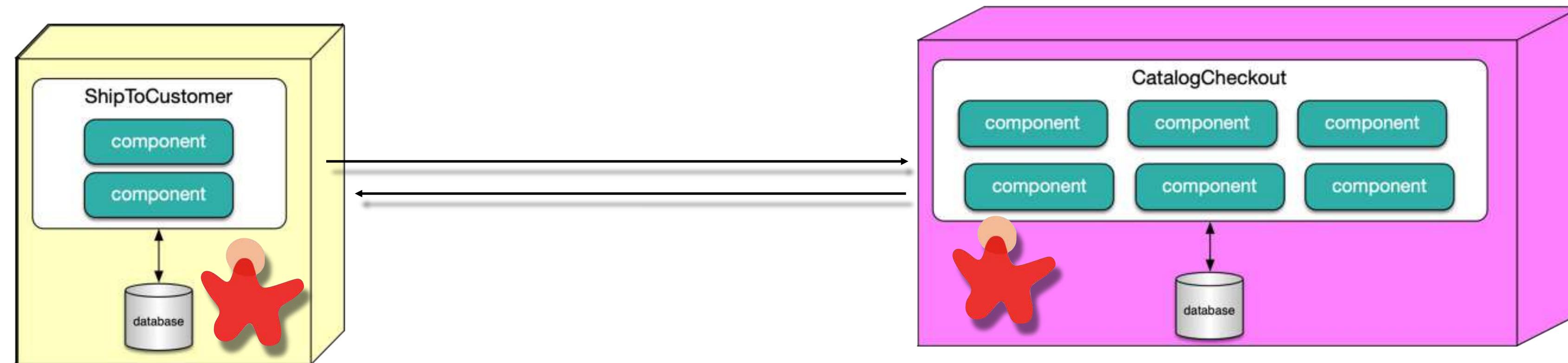
- * guaranteed contract fidelity
- * distinct versions (+ or -?)
- * build-time contracts
- * decoupled...
- * ...therefore better for decoupled architectures
- * contract management



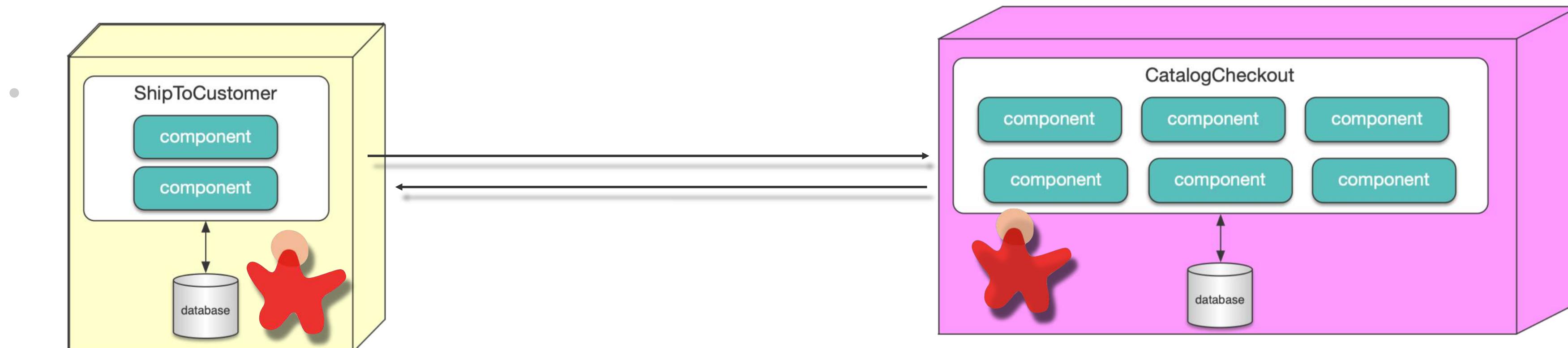
contracts in microservices



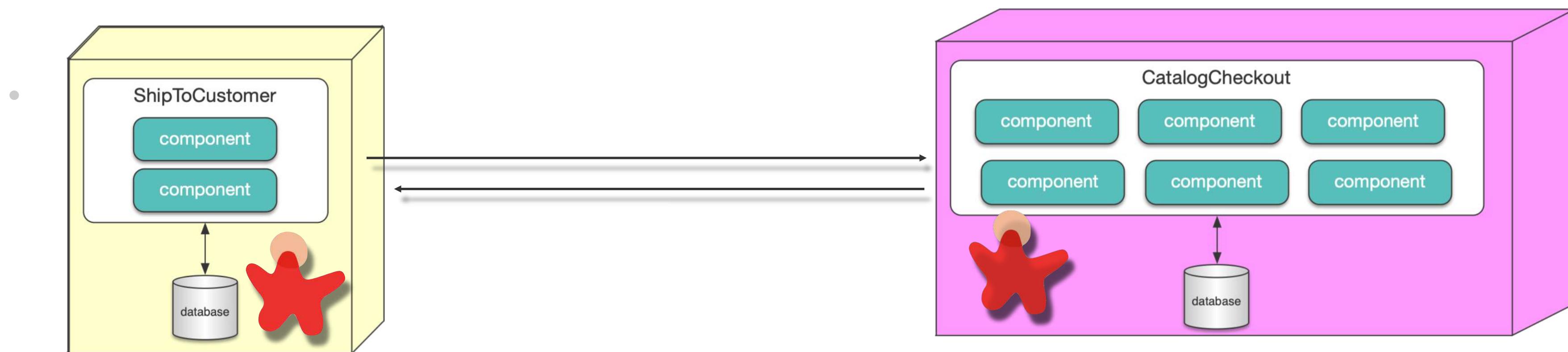
contracts in microservices



contracts in microservices



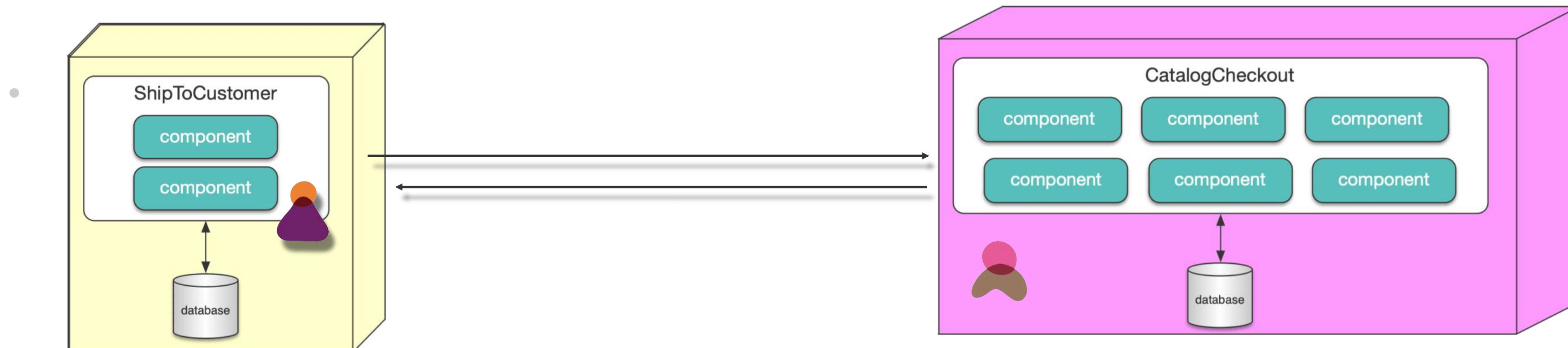
contracts in microservices



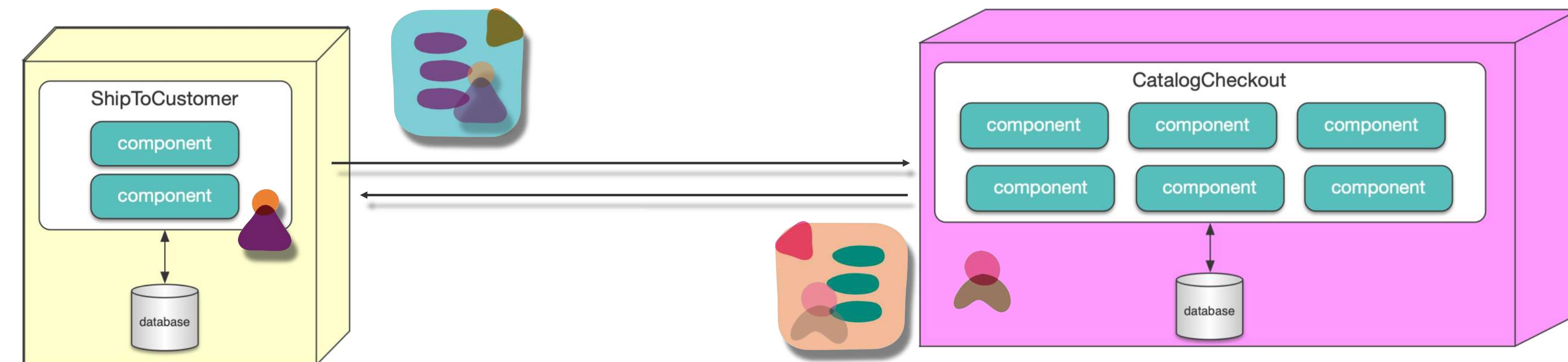
tight

loose

contracts in microservices

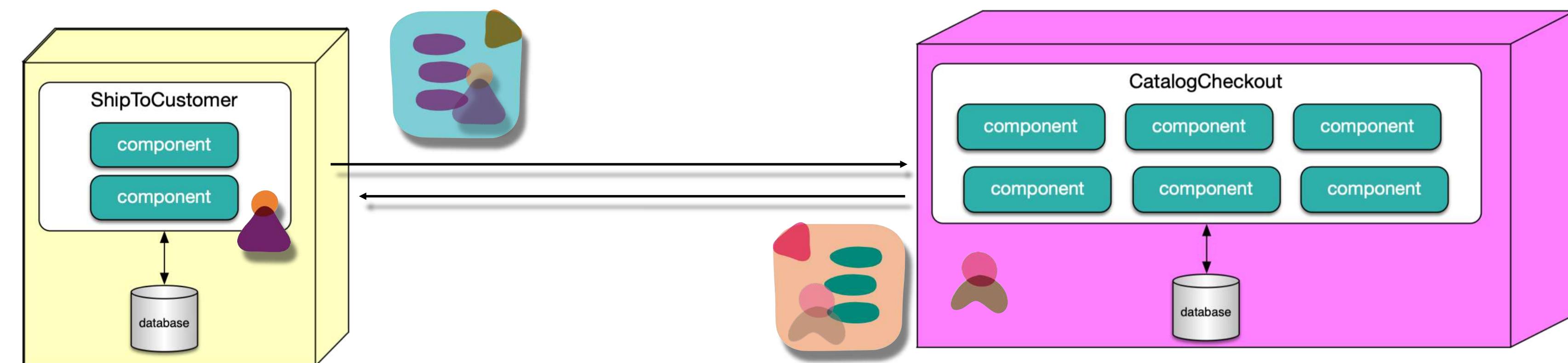


contracts in microservices



contracts in microservices

Transfer values, not types.



tight

loose

connascence properties

Strength



name
type
meaning
algorithm
position
execution order
timing
value
identity

refactor
this way

A large green arrow points diagonally upwards and to the right, starting from the bottom left and ending near the top right of the text area.



static

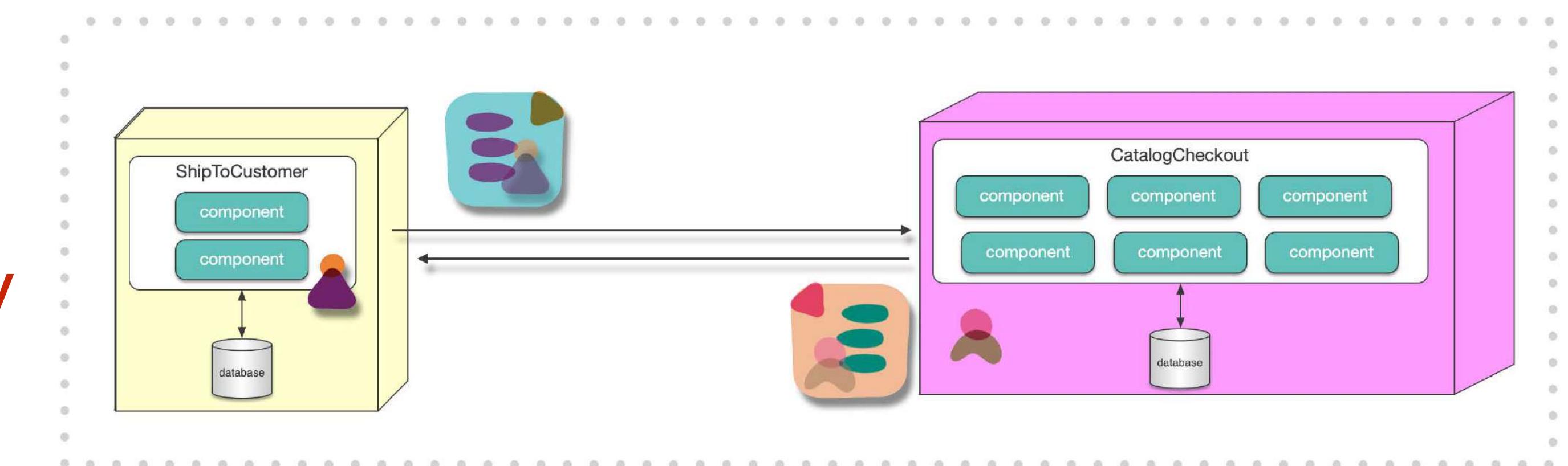
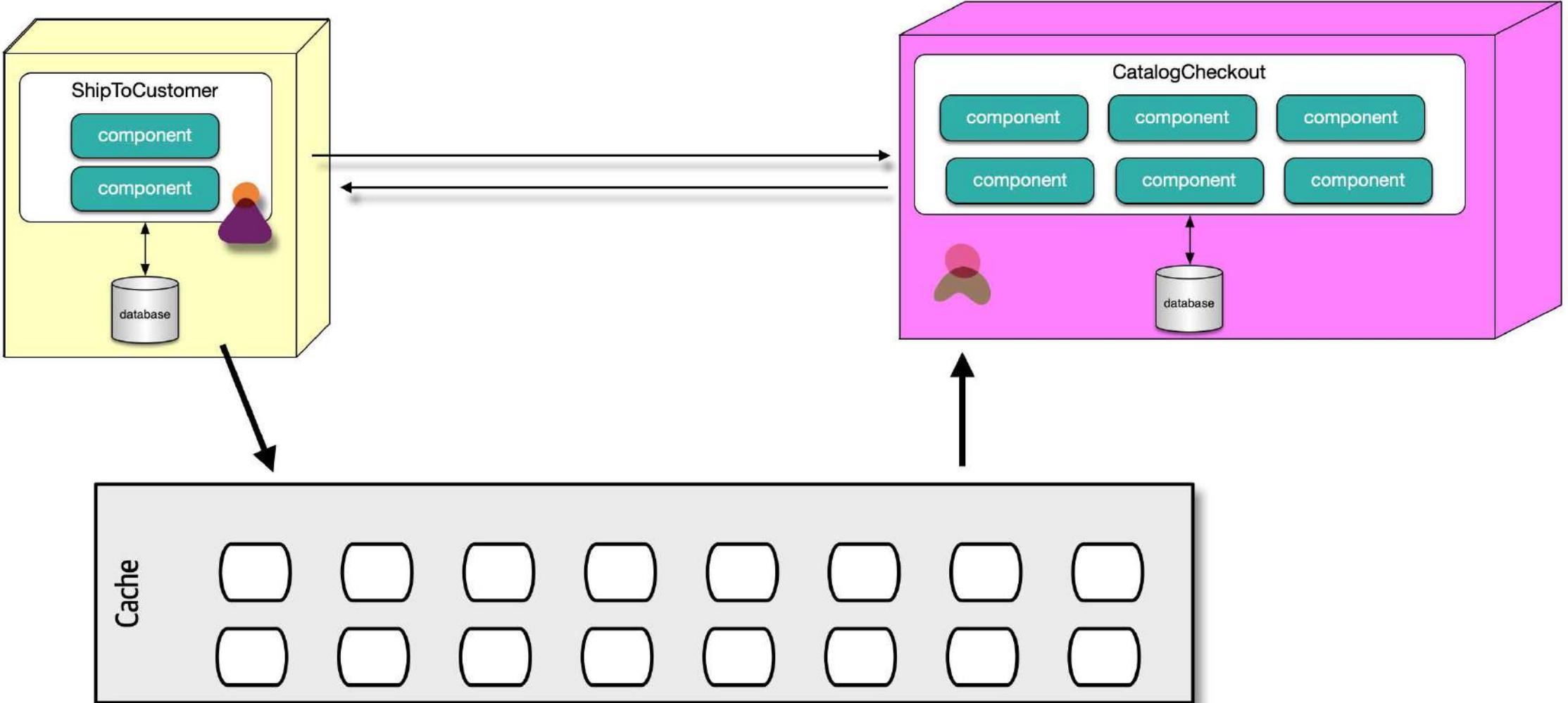
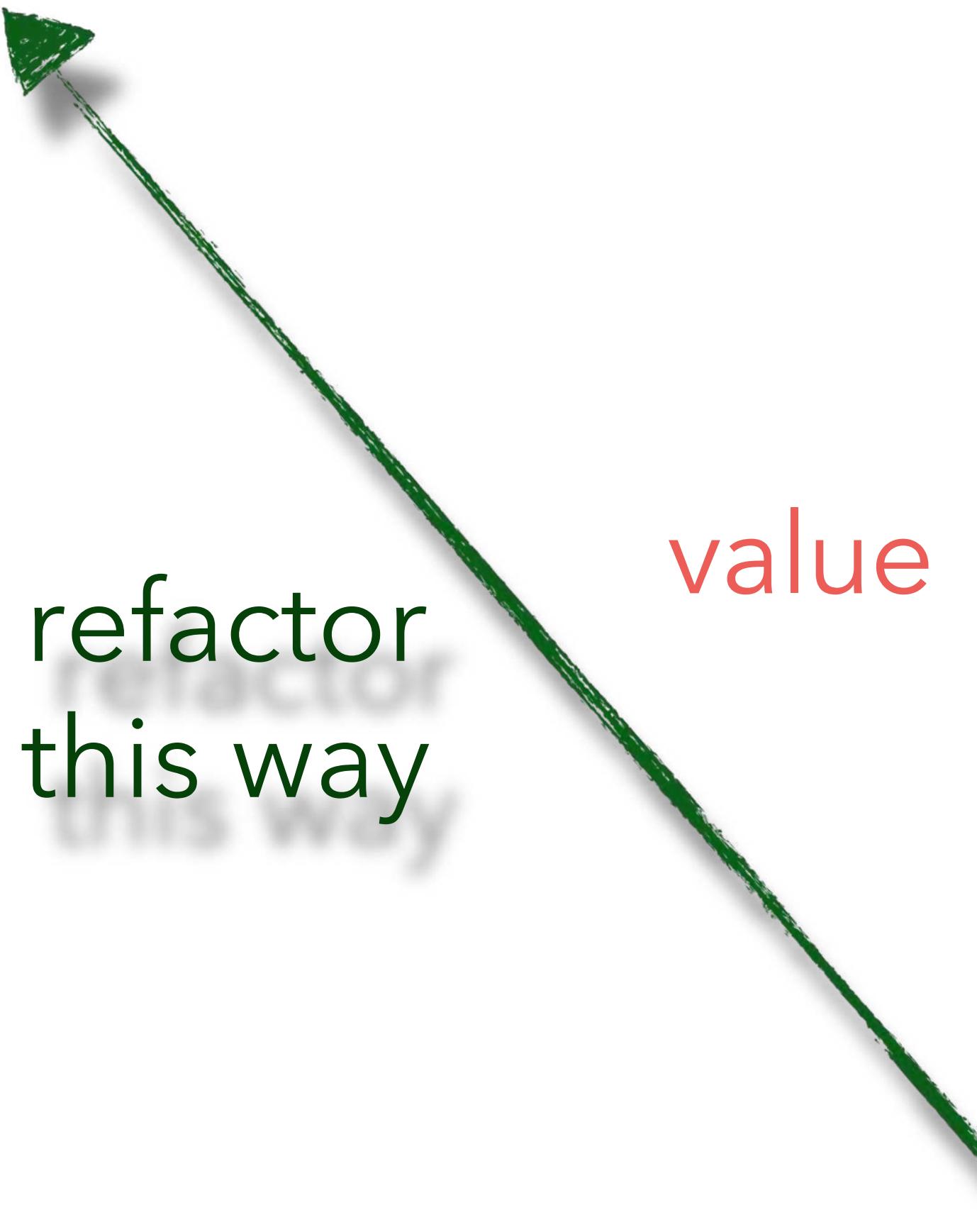
dynamic

improving connascence

refactor
this way

value

identity

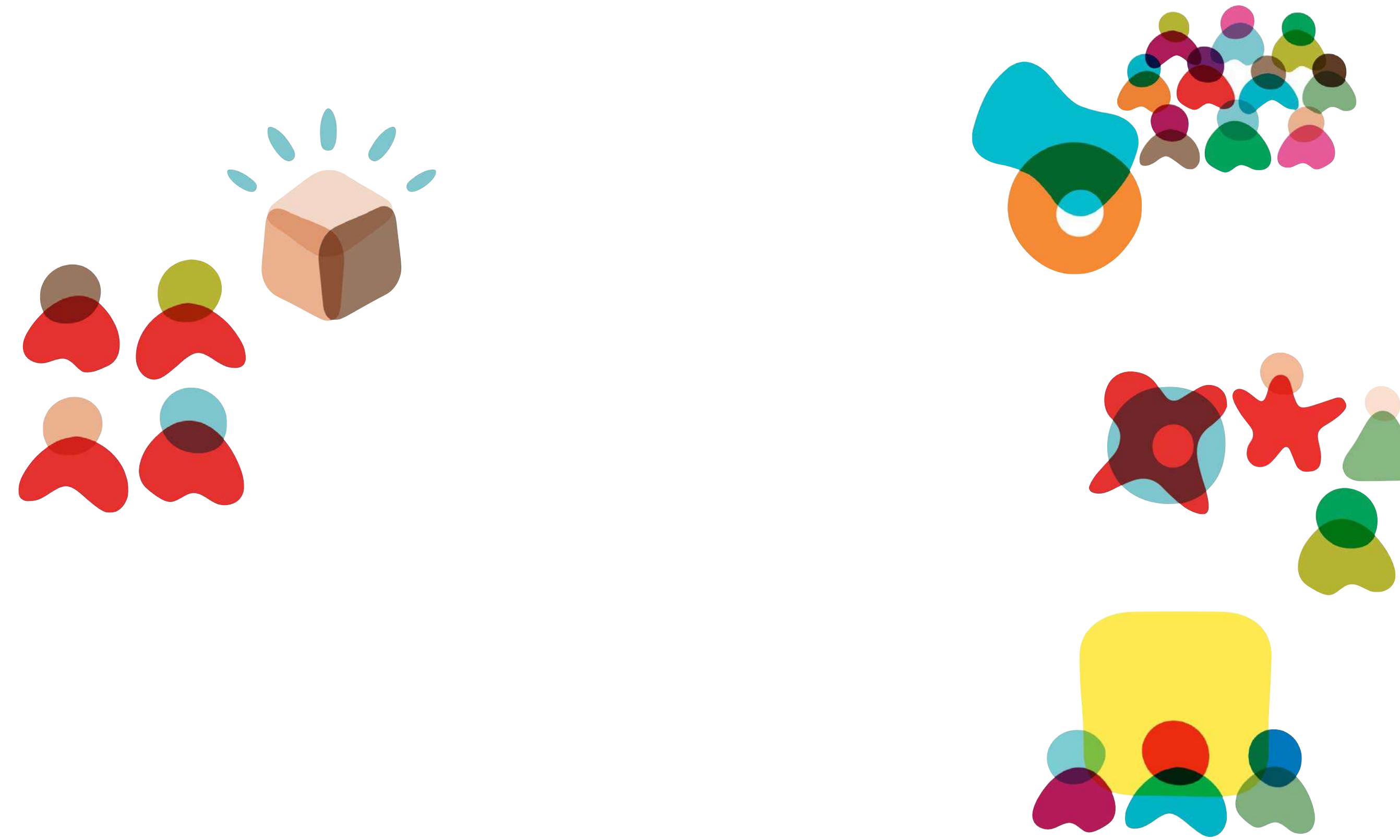


tradeoffs

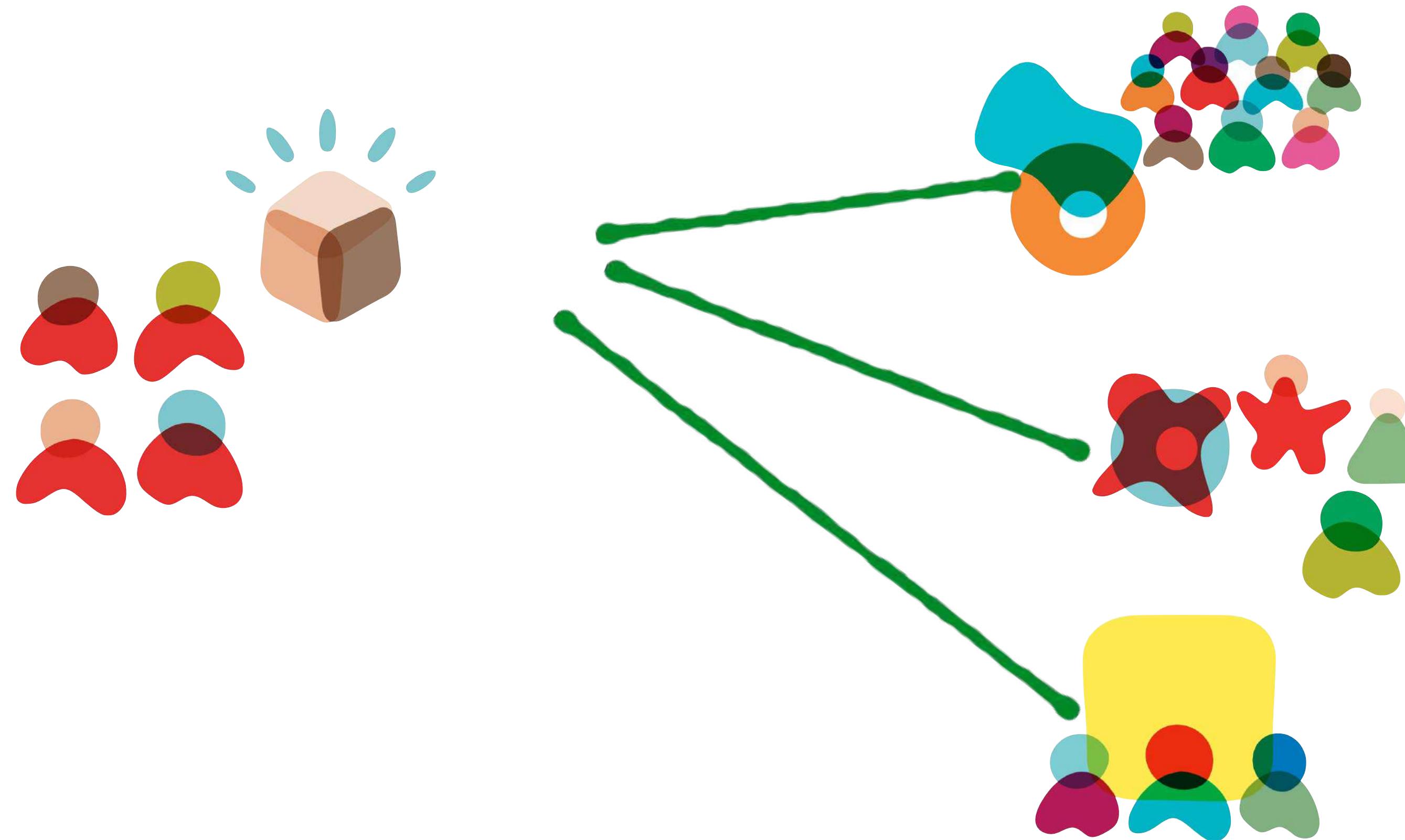
value-based contracts

- + more malleable integration points
- + easier to evolve
- + decouples integration from implementation platform
- the “contract” part of contract

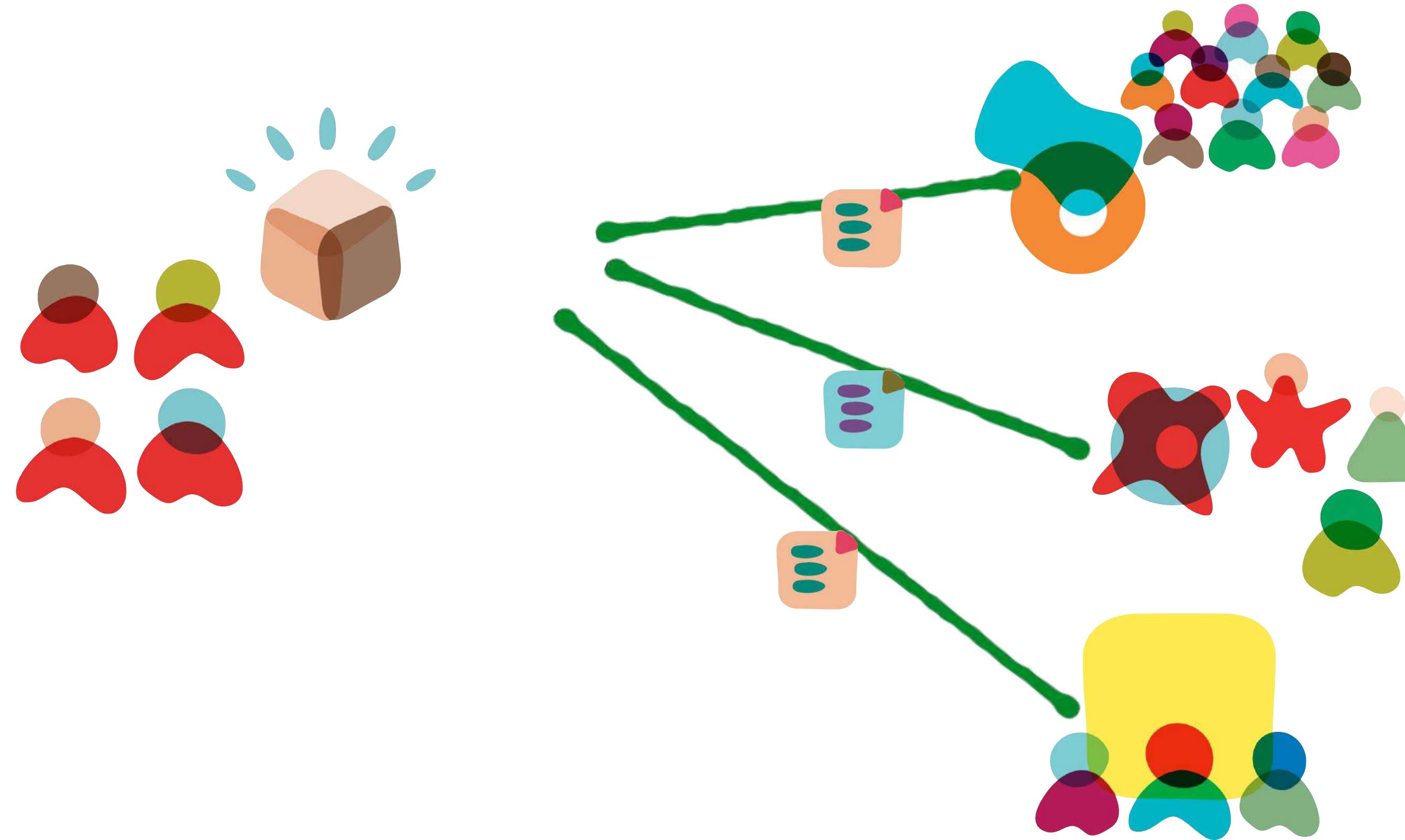
contract fitness function



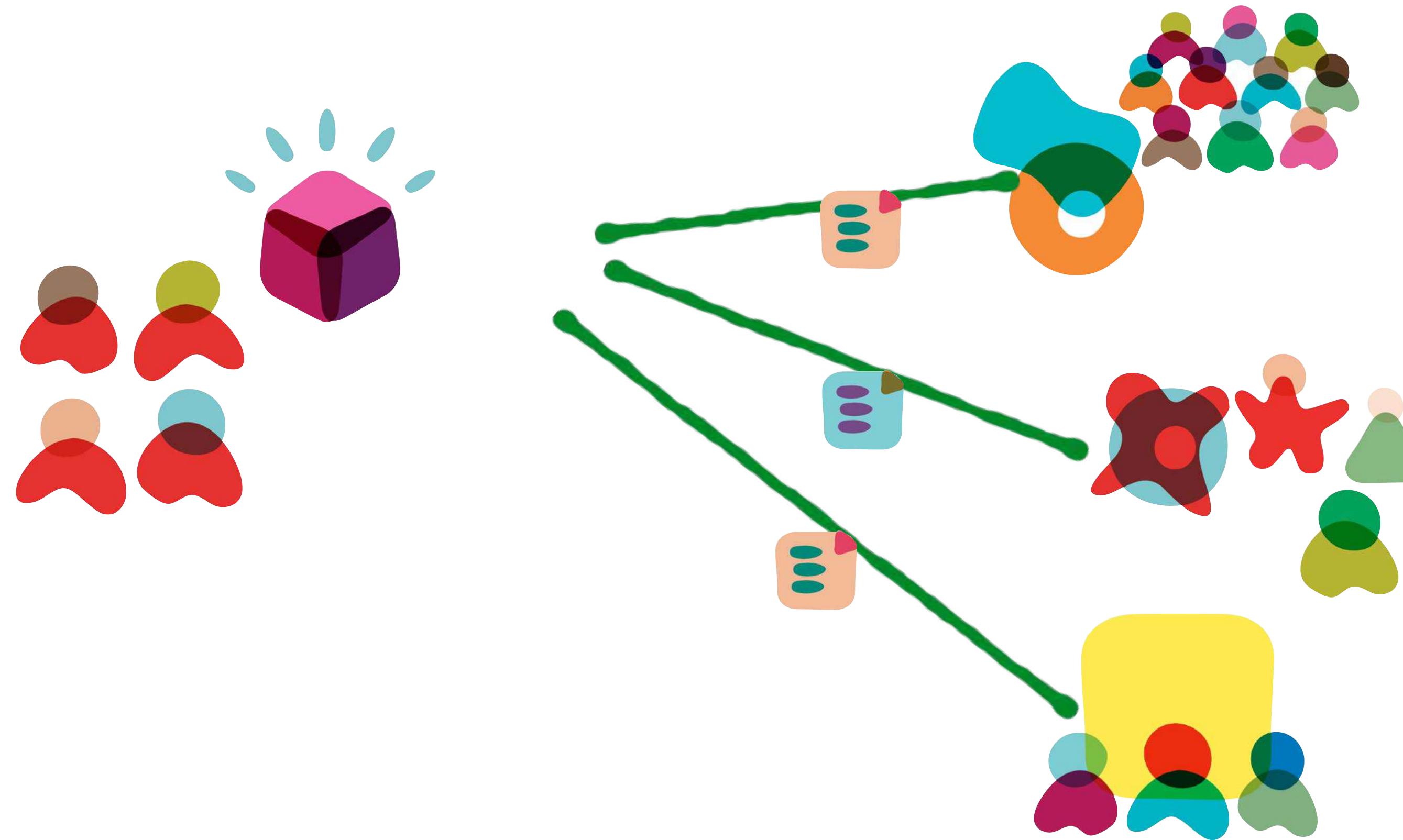
contract fitness function



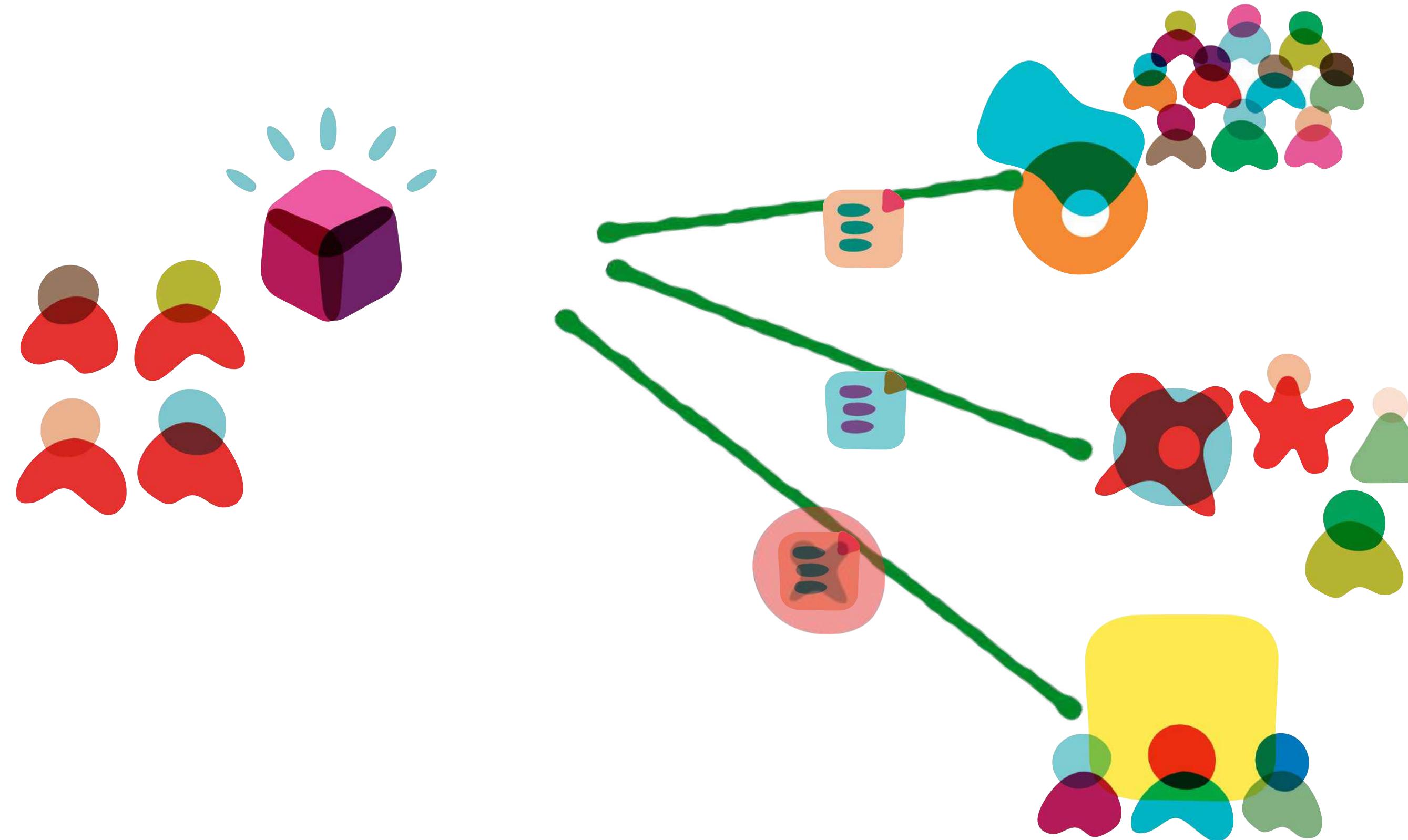
contract fitness function



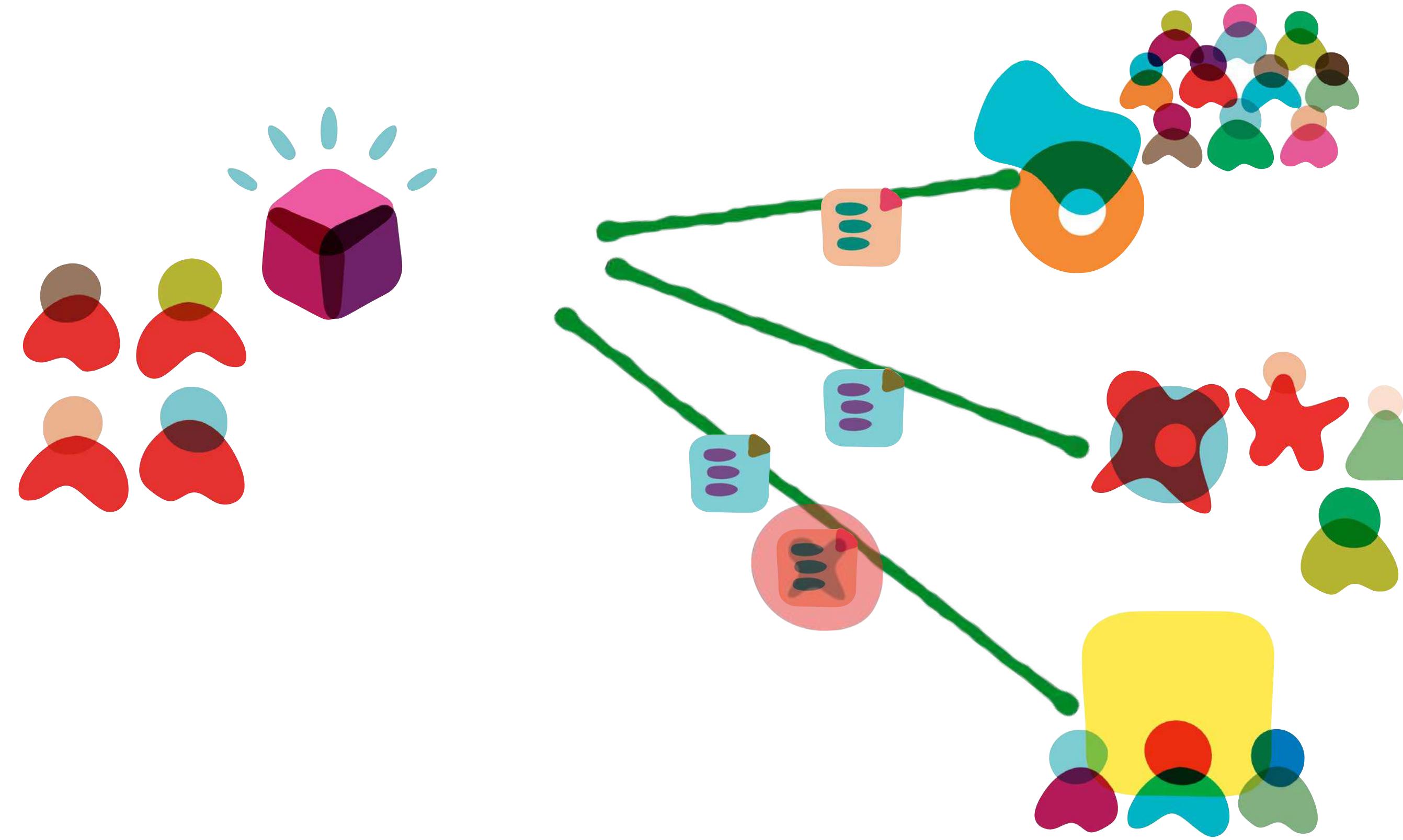
contract fitness function



contract fitness function

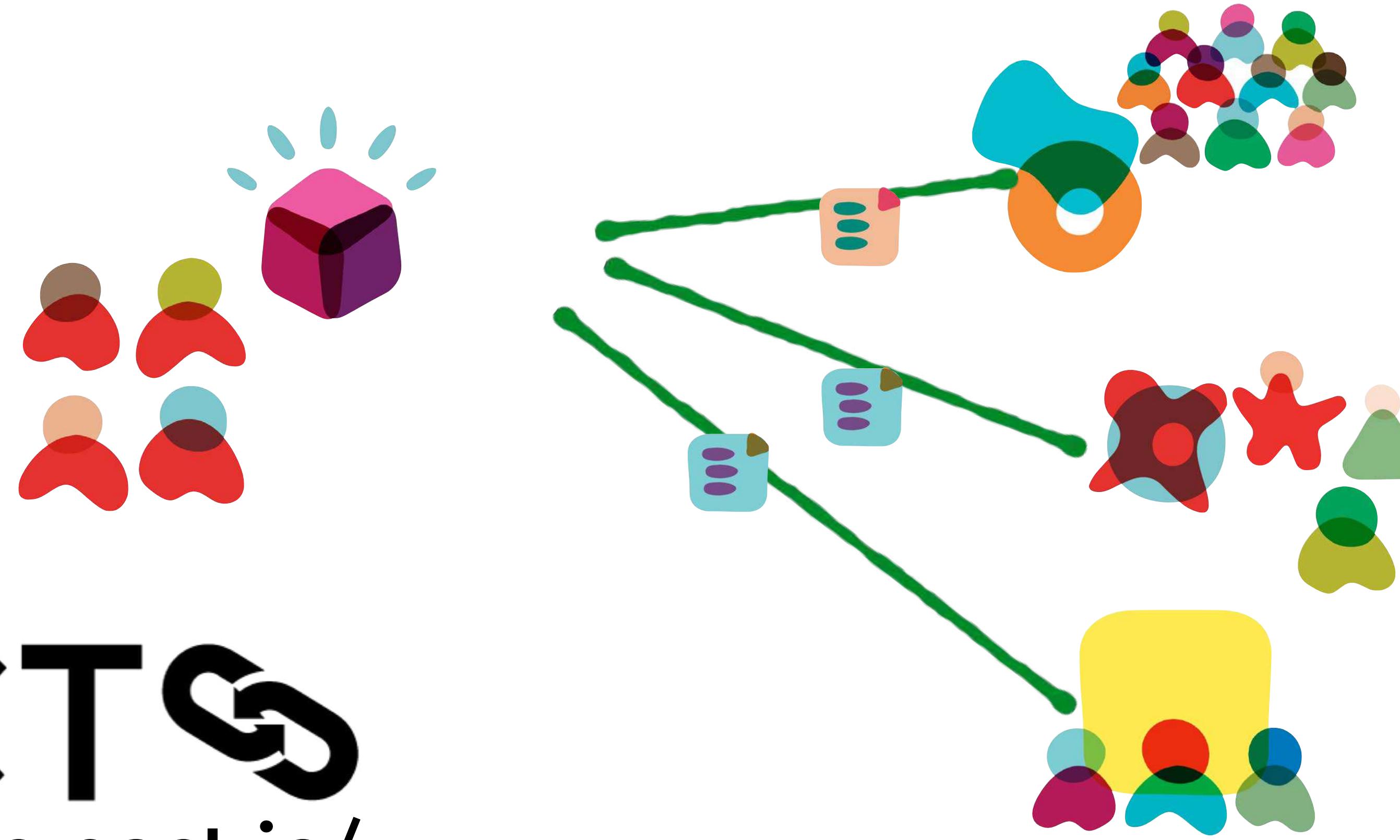


contract fitness function



martinfowler.com/articles/consumerDrivenContracts.html

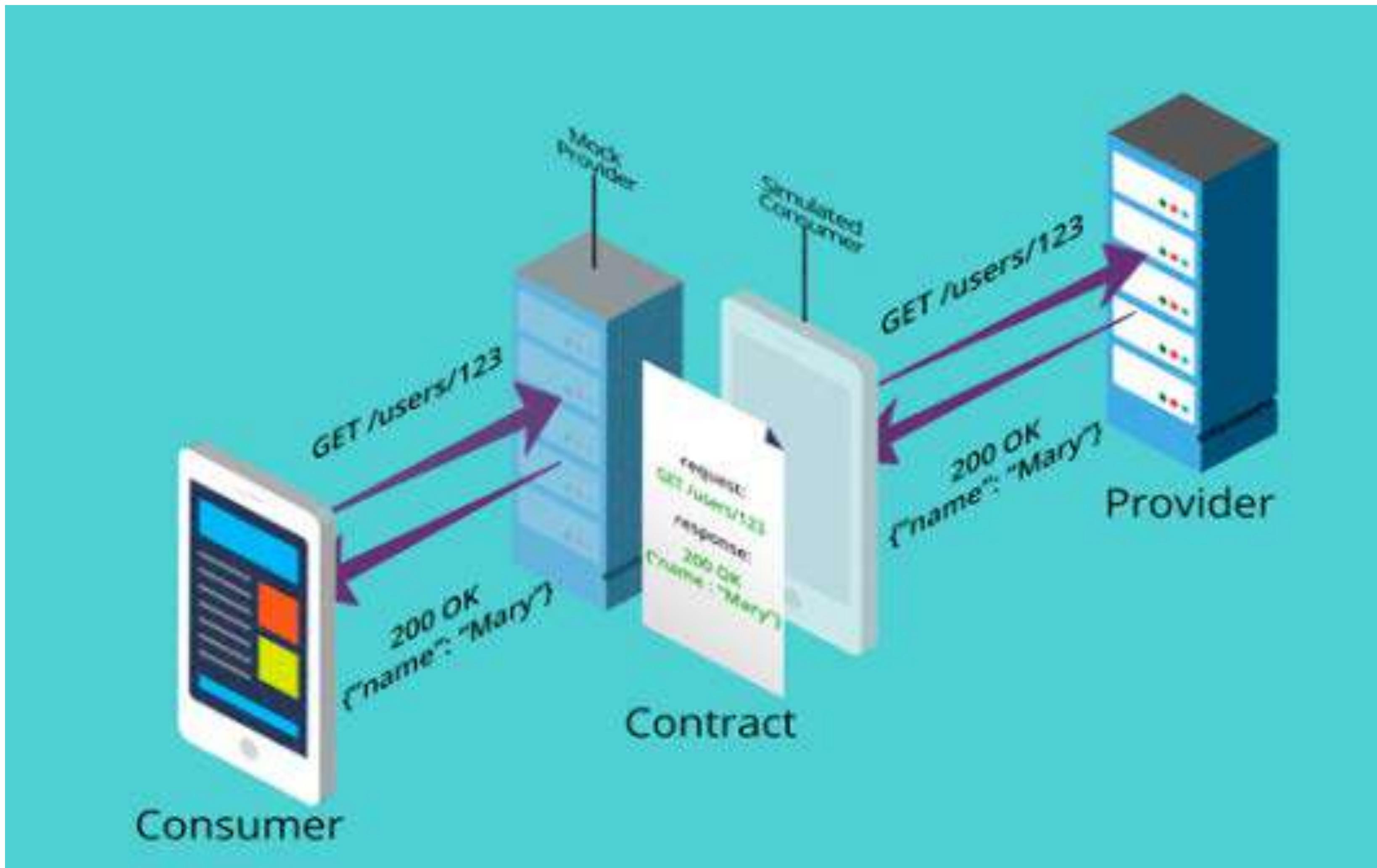
contract fitness function



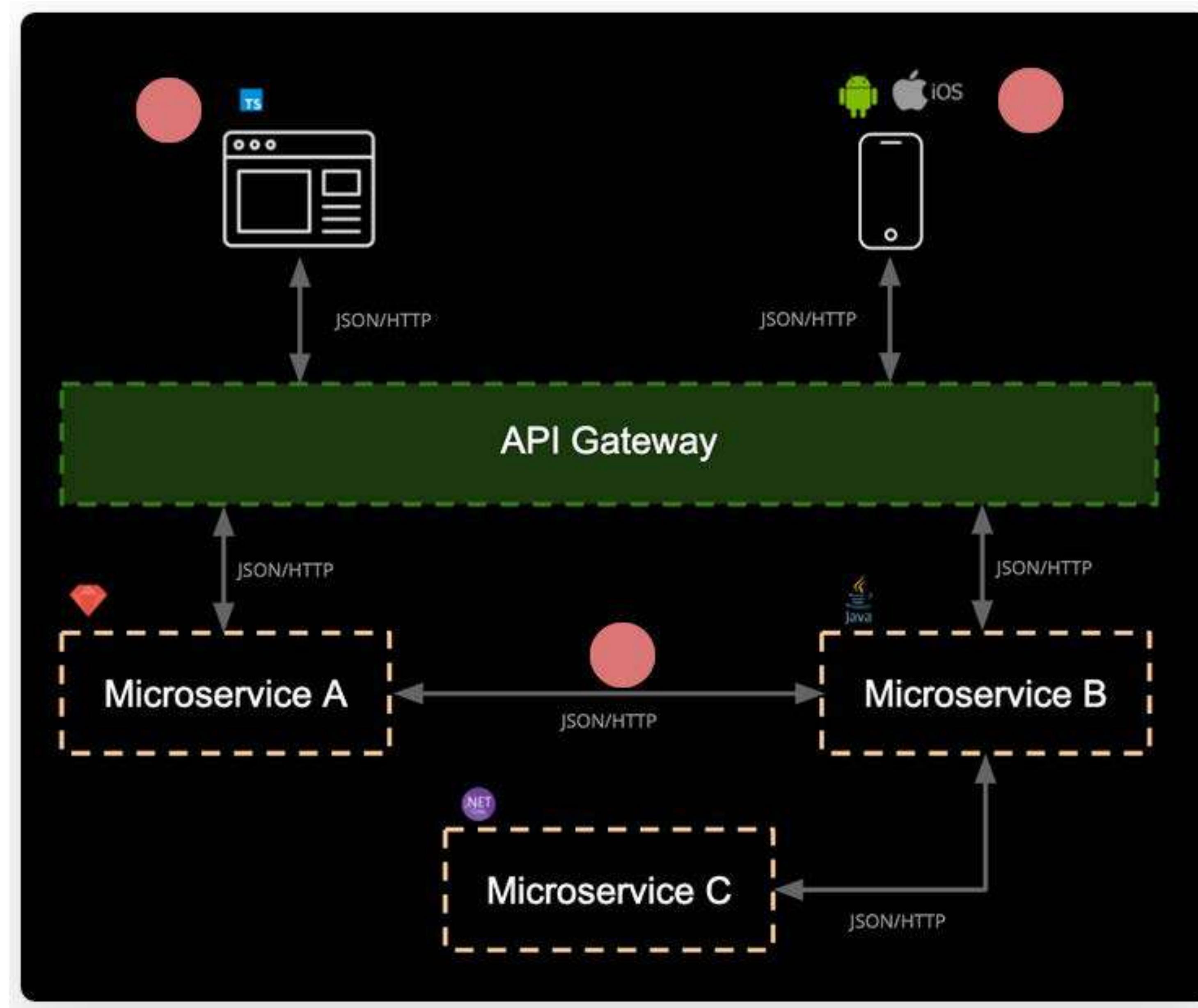
PACTS
<https://docs.pact.io/>

martinfowler.com/articles/consumerDrivenContracts.html

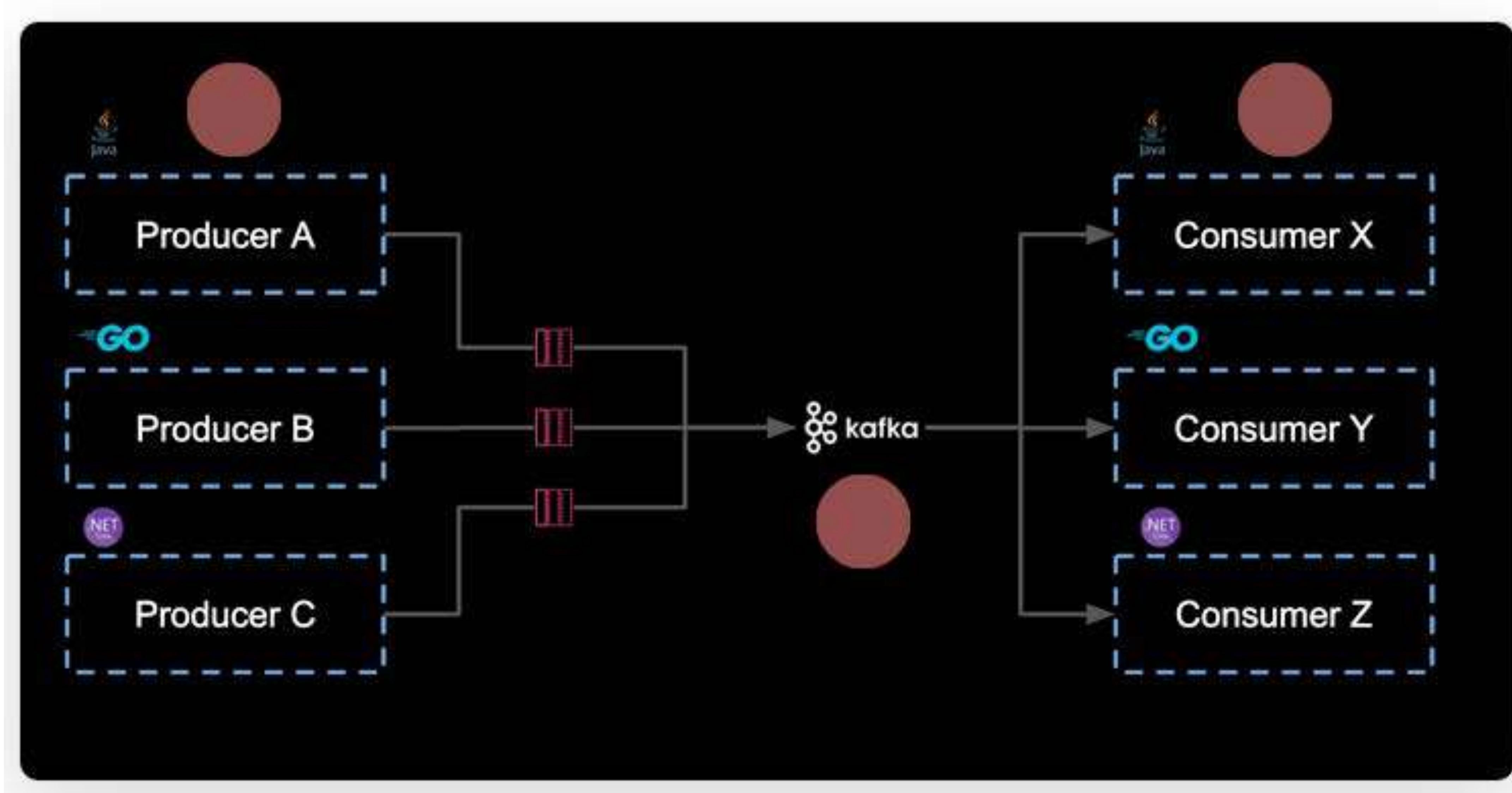
<https://docs.pact.io>



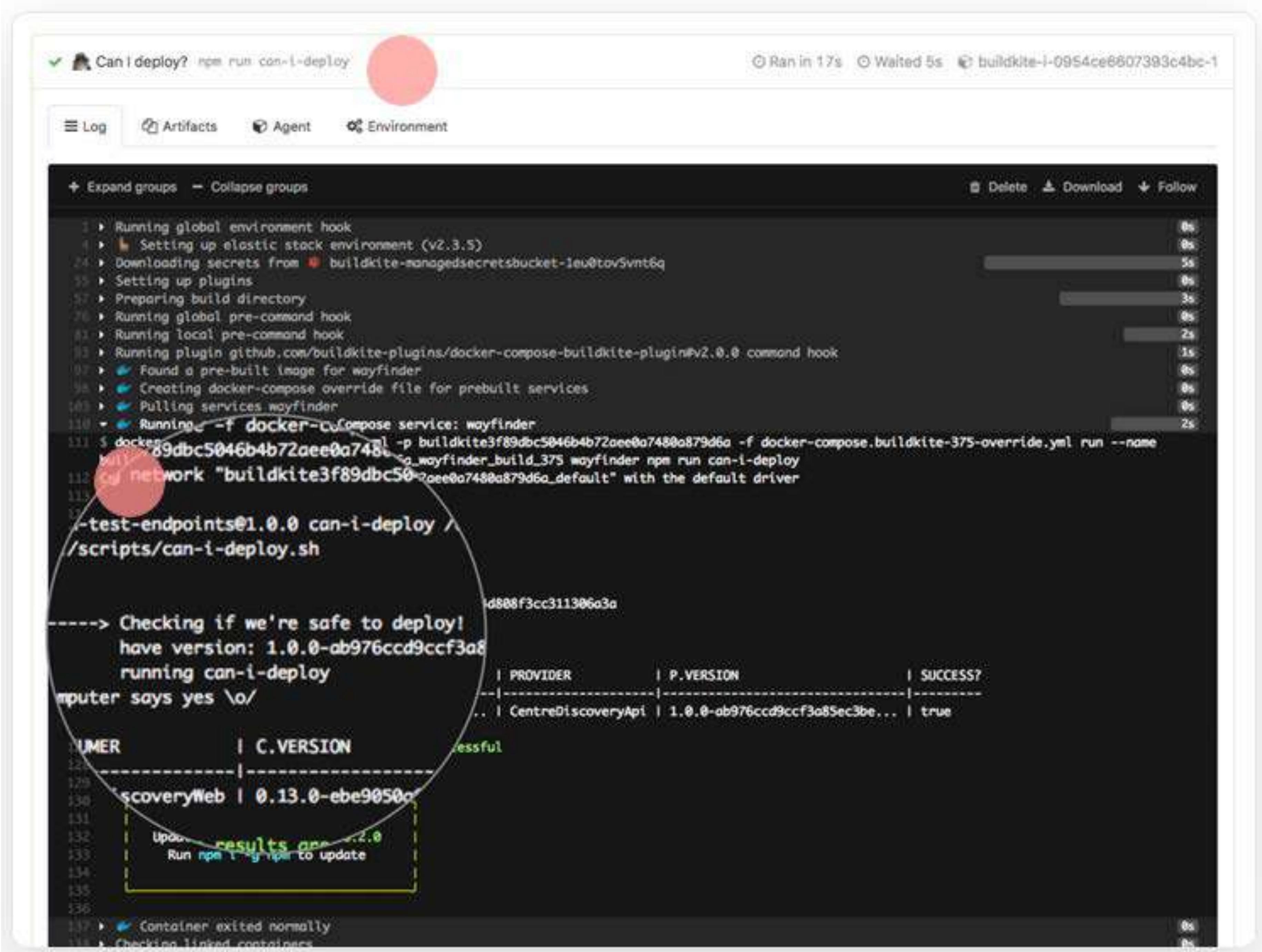
Pact + REST



Pact + EDA



A
C
I
C
+
Pact



tradeoffs

strict contracts

+ guaranteed type fidelity

- brittle integration points
- requires versioning

tradeoffs

value-based contracts

- + extremely loose coupling
- + immune from implementation change

- less certainty in contracts
(requires fitness functions)
- requires more developer discipline

tradeoffs

strict contracts

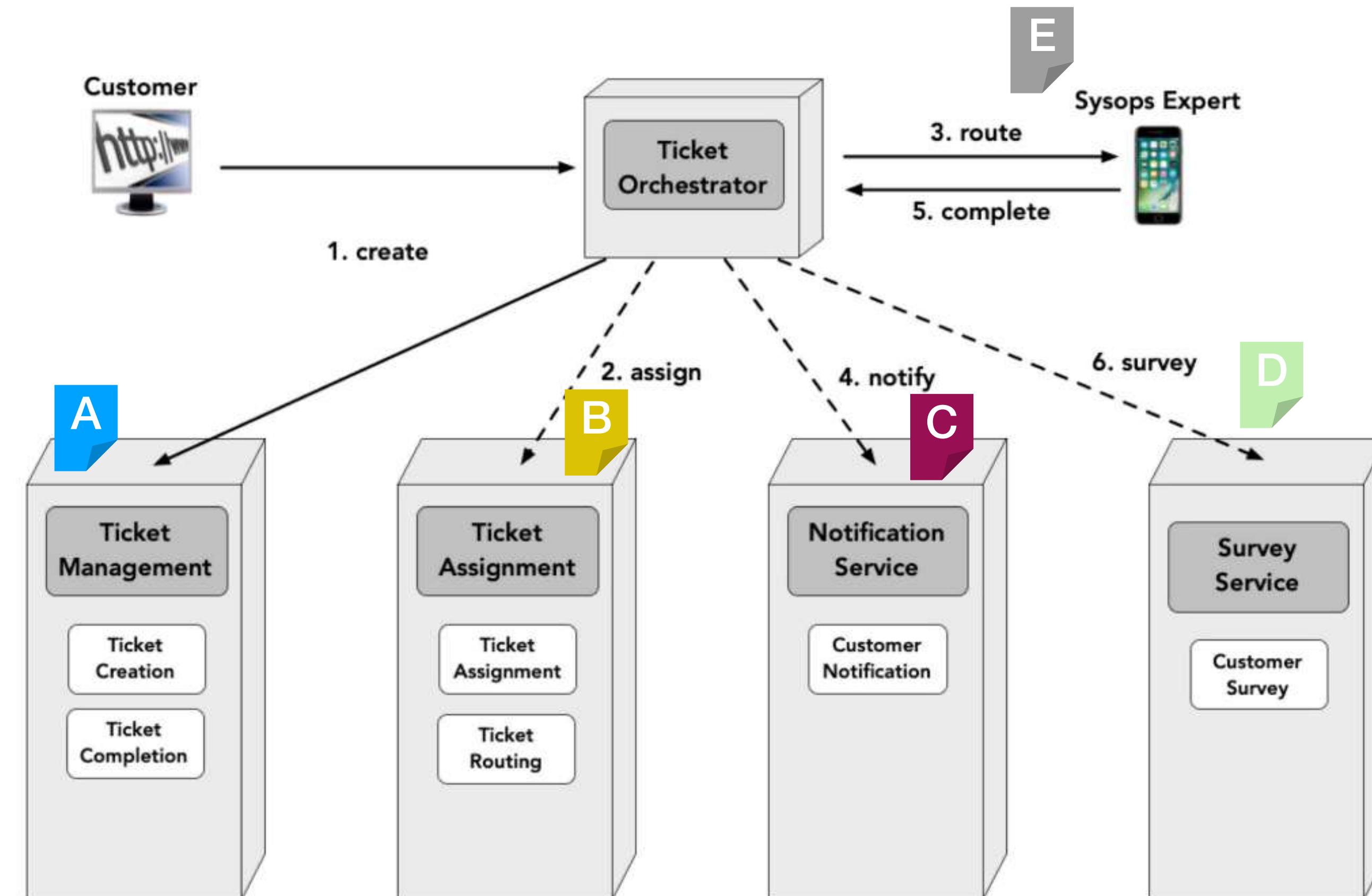
OR

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

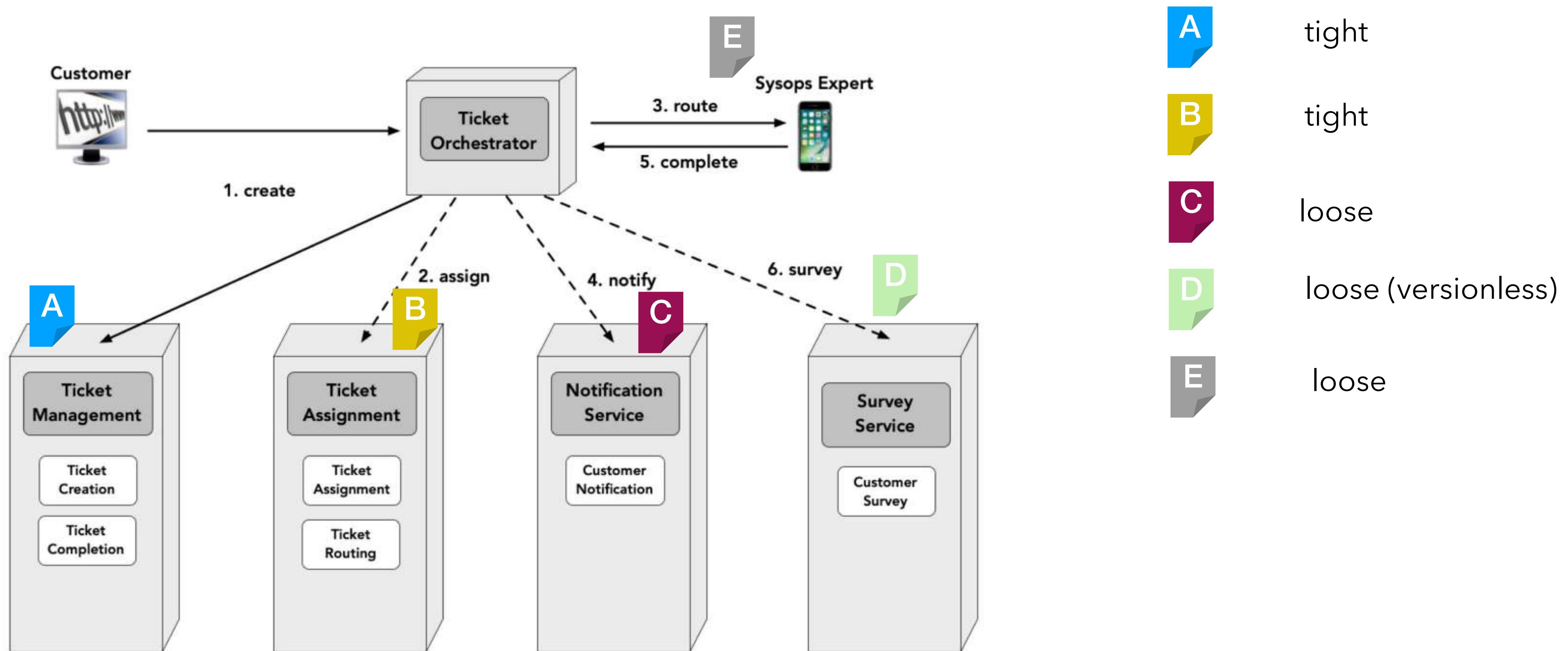
Scenario Exercise - Semantic versus Syntactic Coupling

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



Scenario Exercise - Semantic versus Syntactic Coupling

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

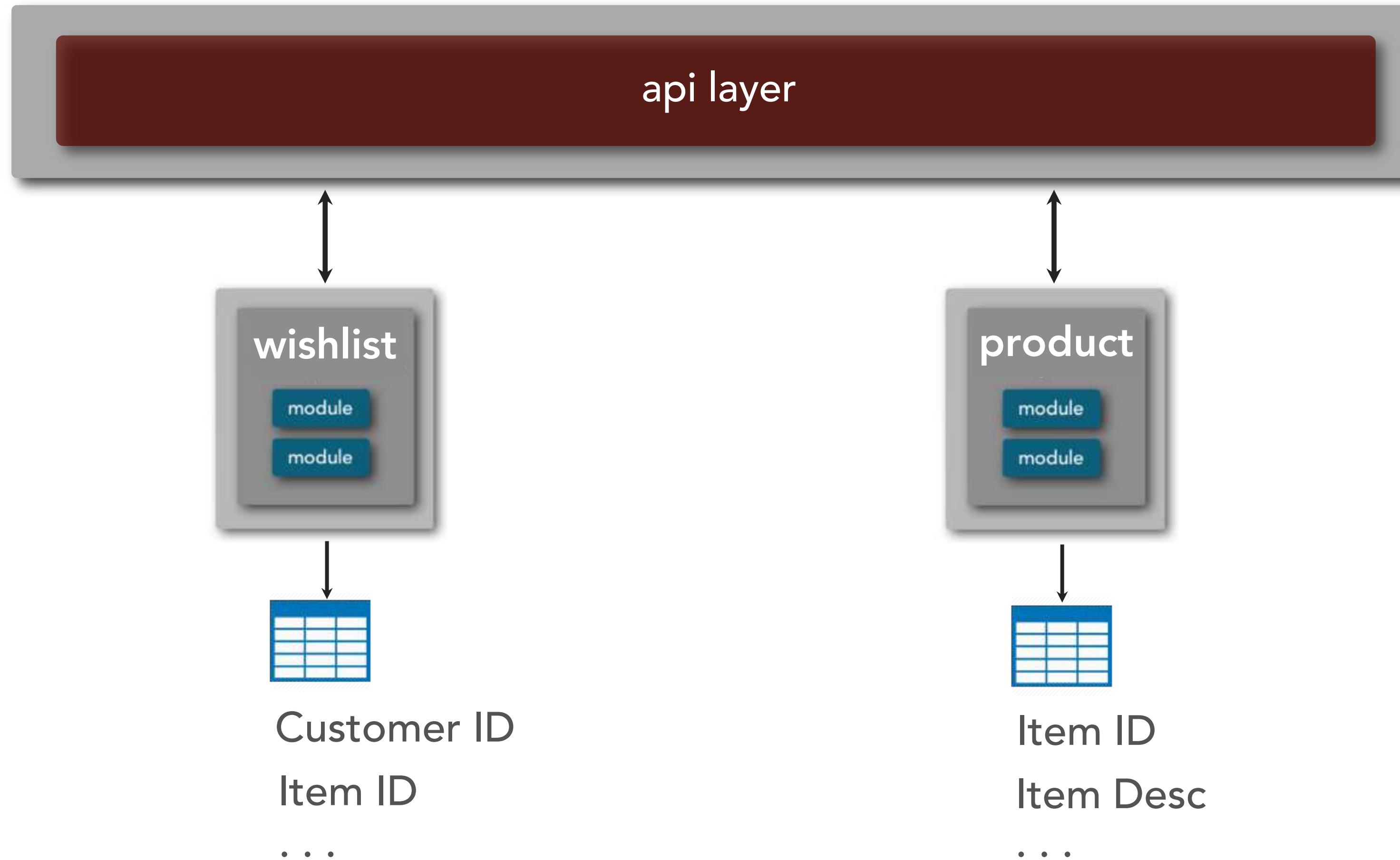




How do I access data I don't own in a distributed system?

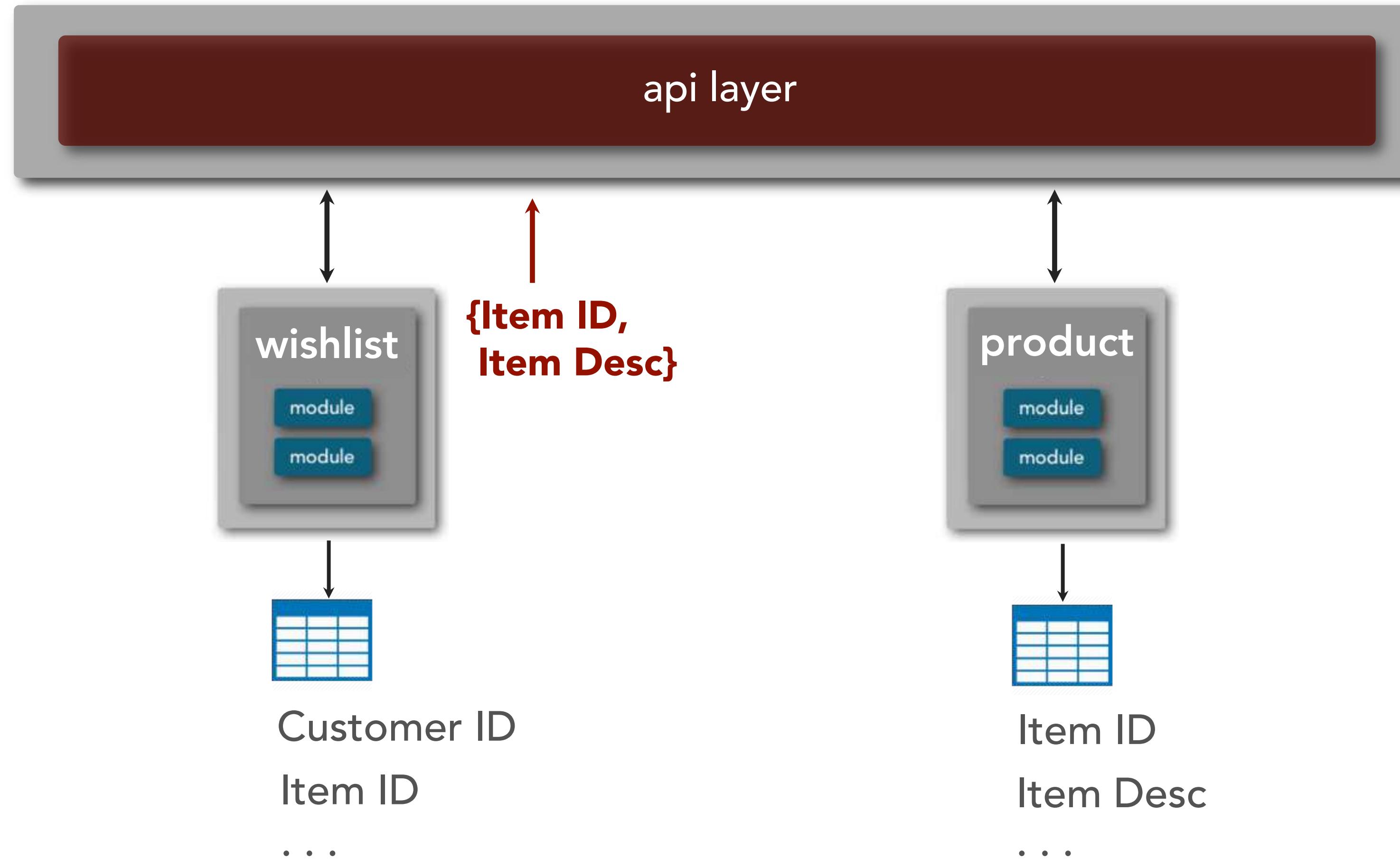
data access and sharing

the issue...



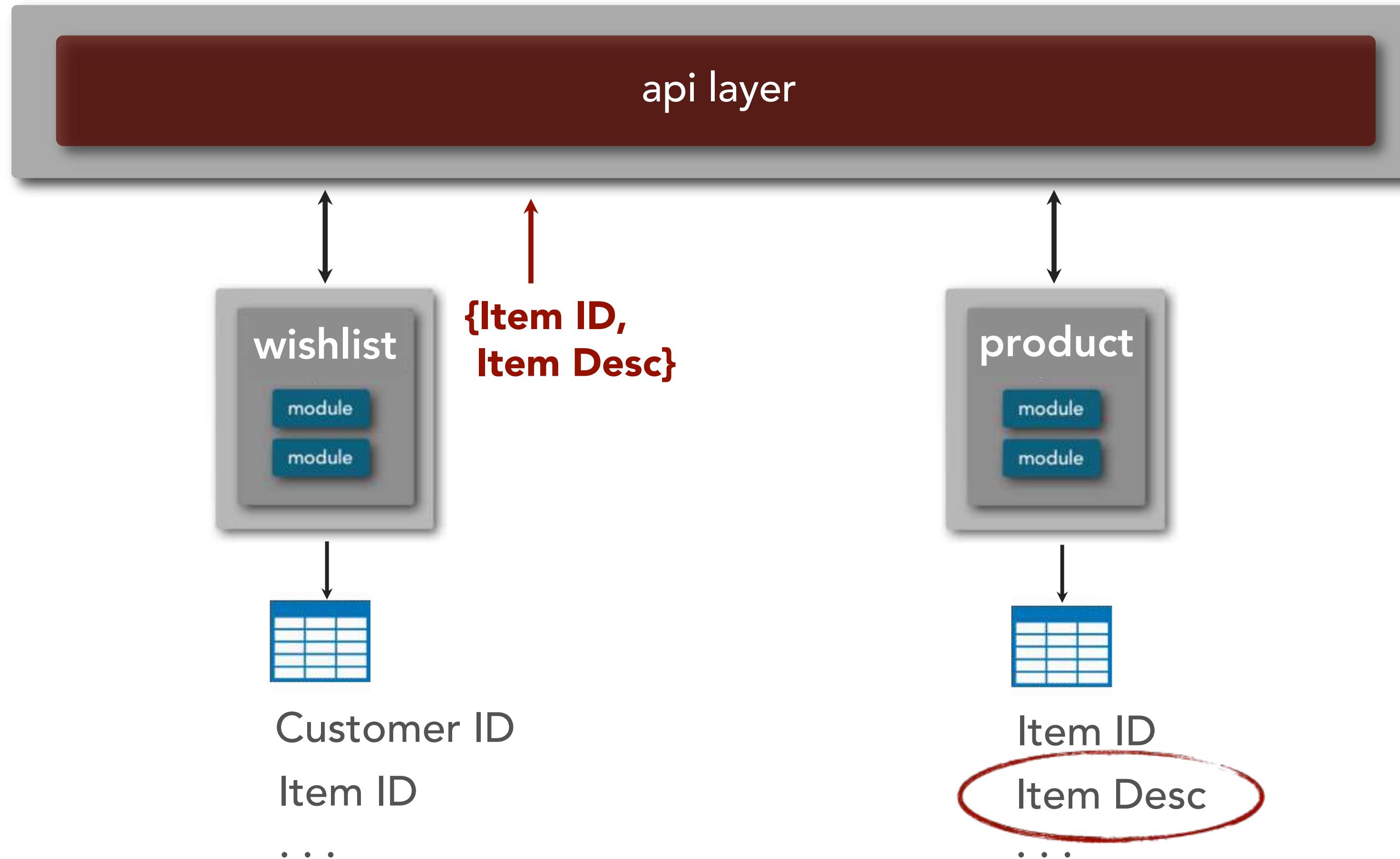
data access and sharing

the issue...



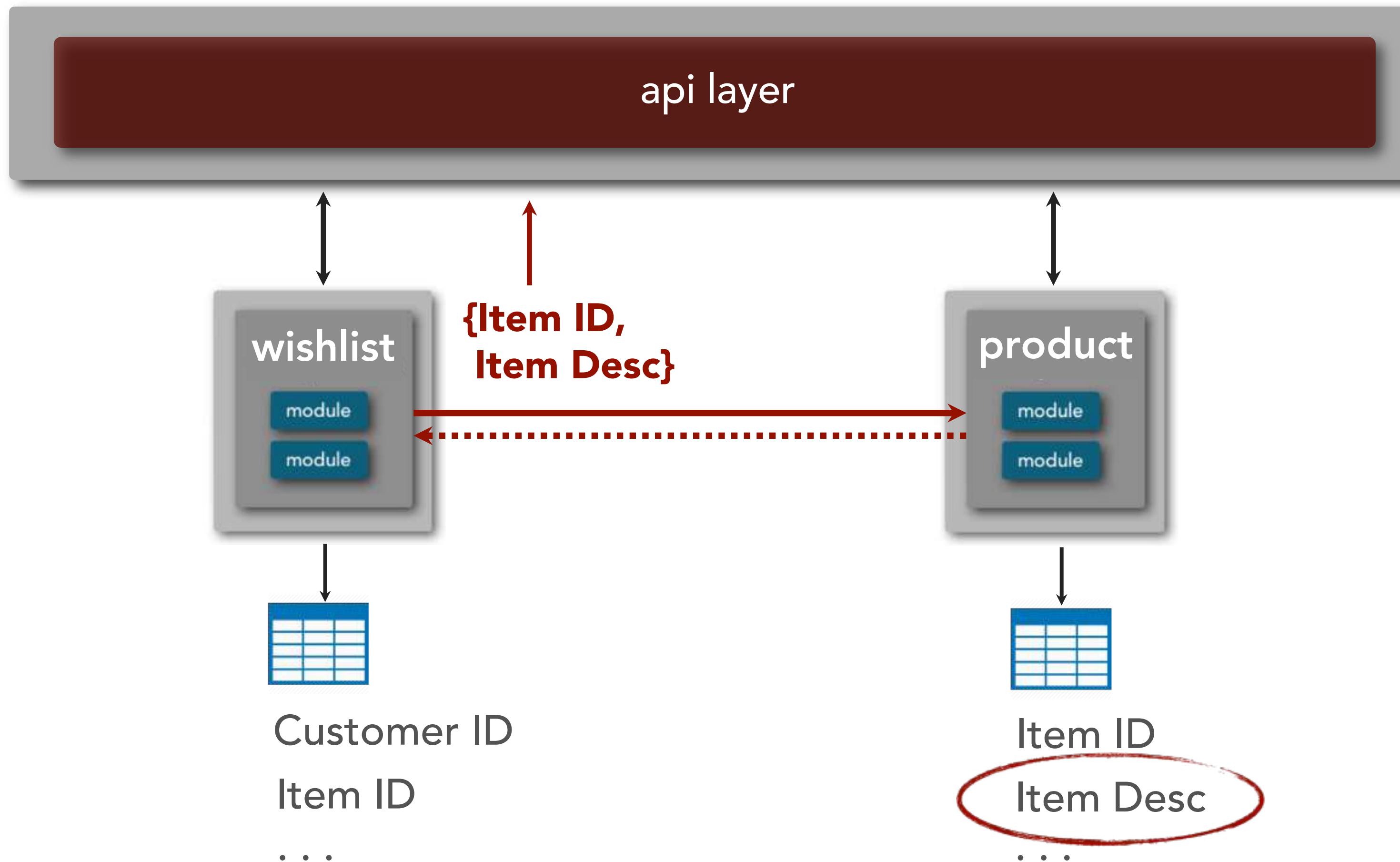
data access and sharing

the issue...



data access and sharing

option 1: interservice communication



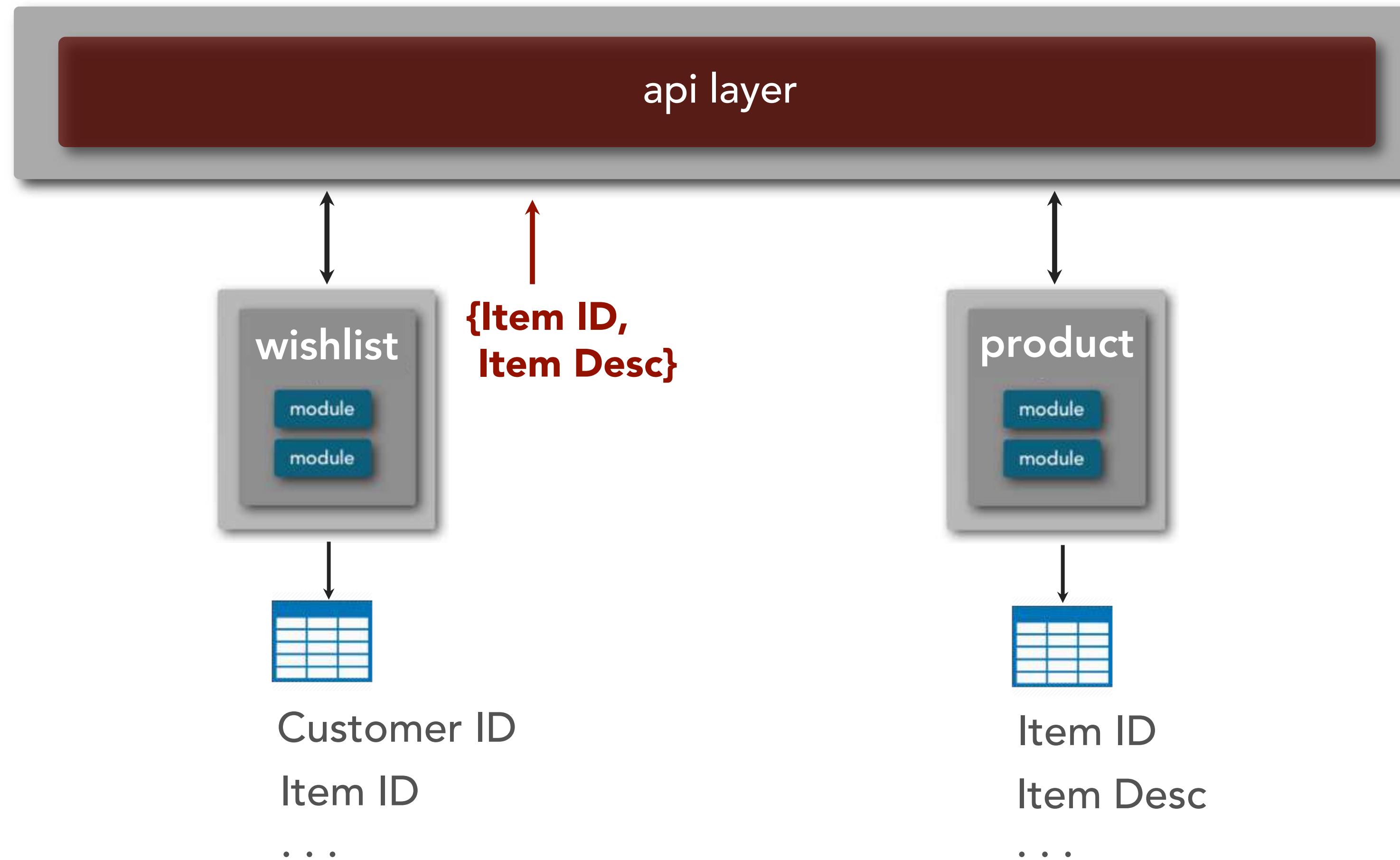
tradeoffs

option 1 - interservice communication

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

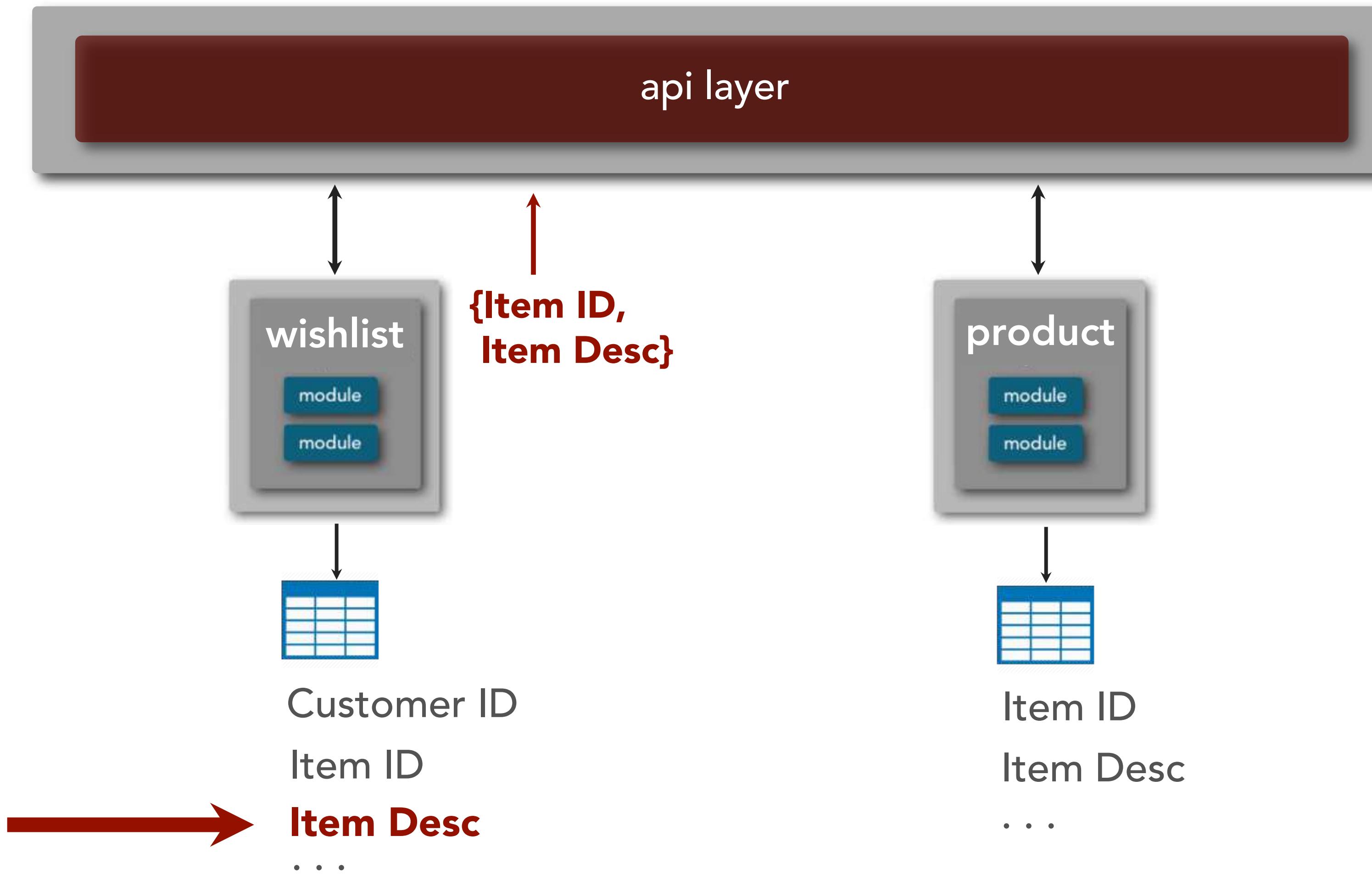
data access and sharing

option 2: data replication



data access and sharing

option 2: data replication



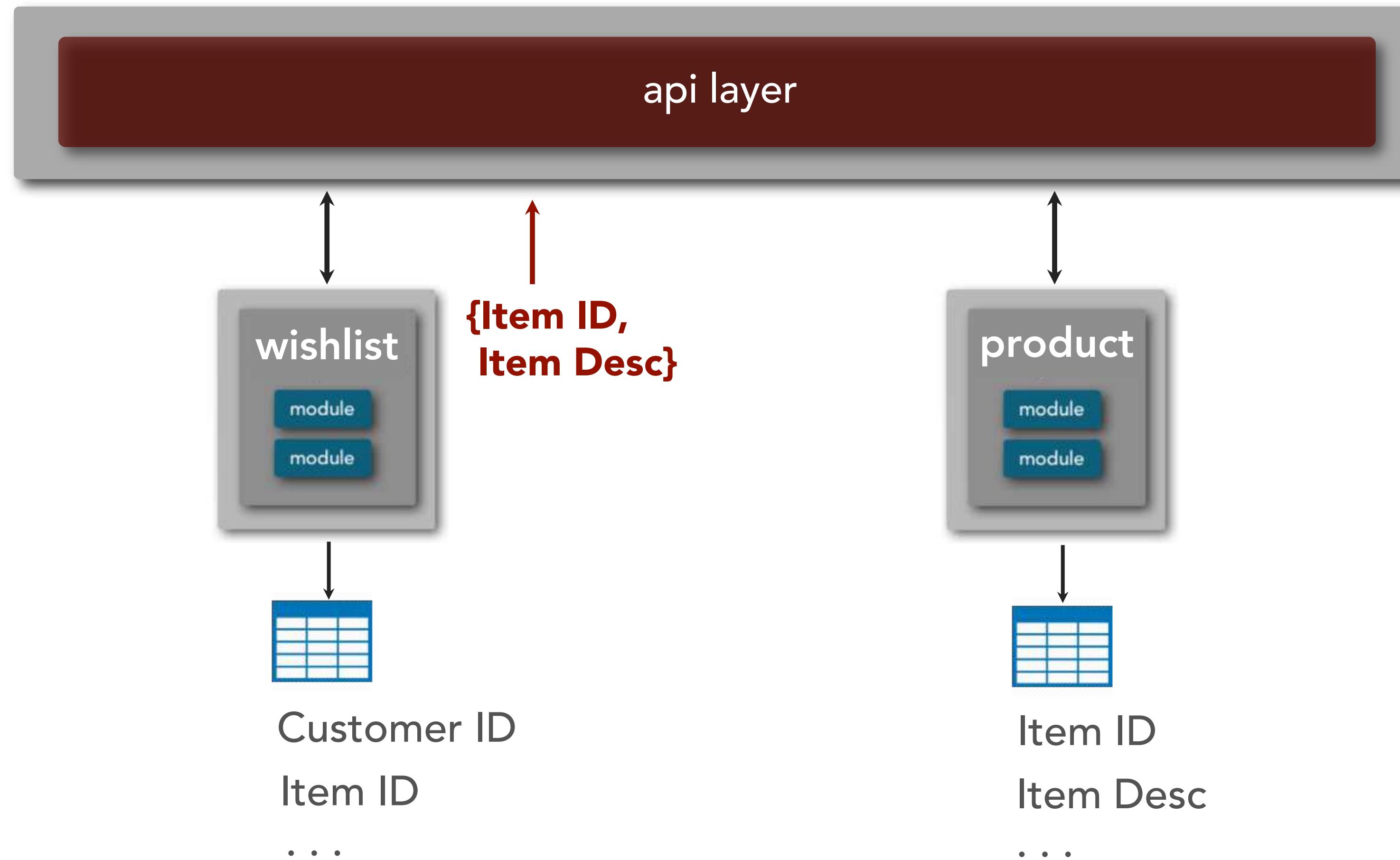
tradeoffs

option 2 - data replication

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

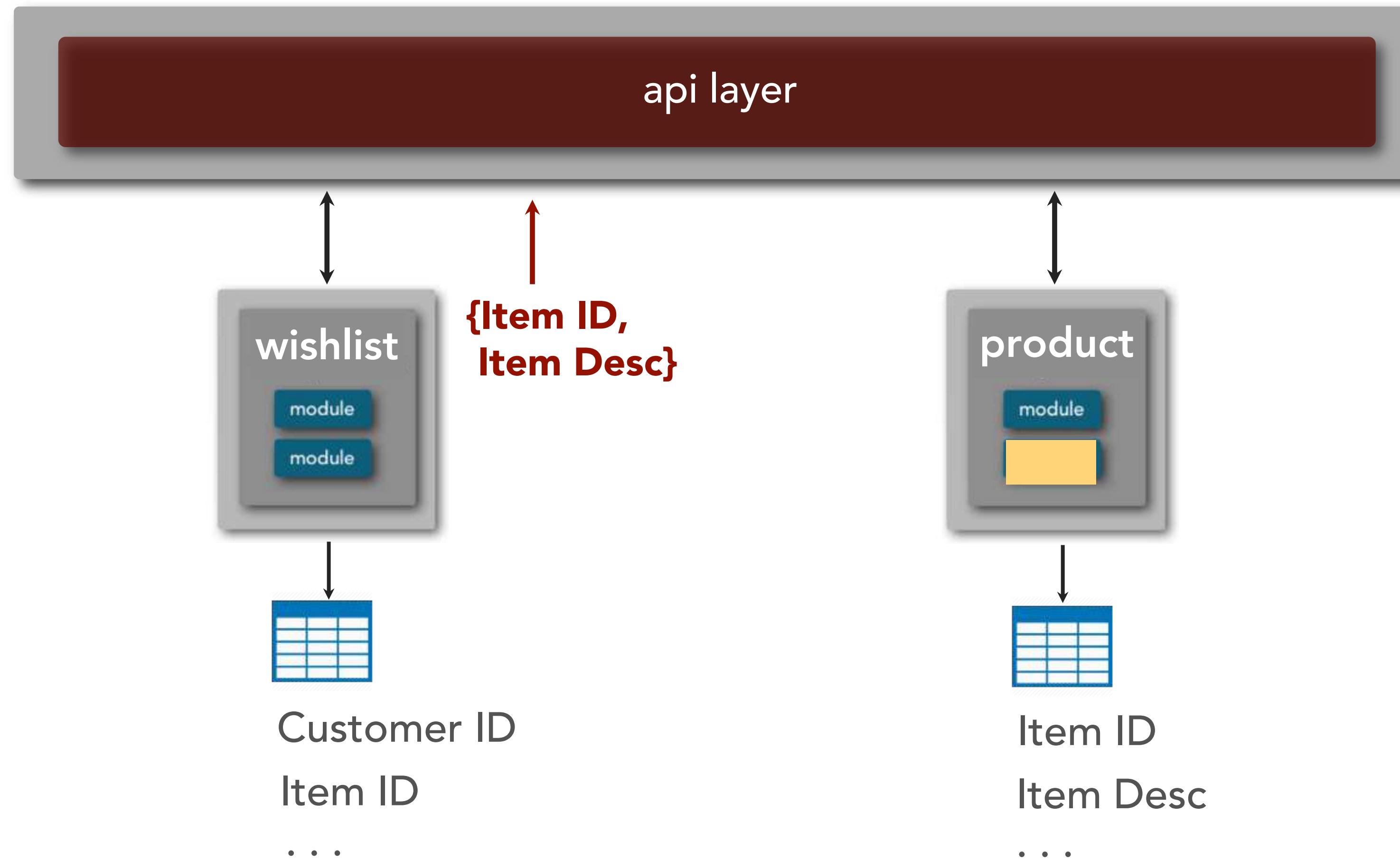
data access and sharing

option 3: in-memory replicated cache



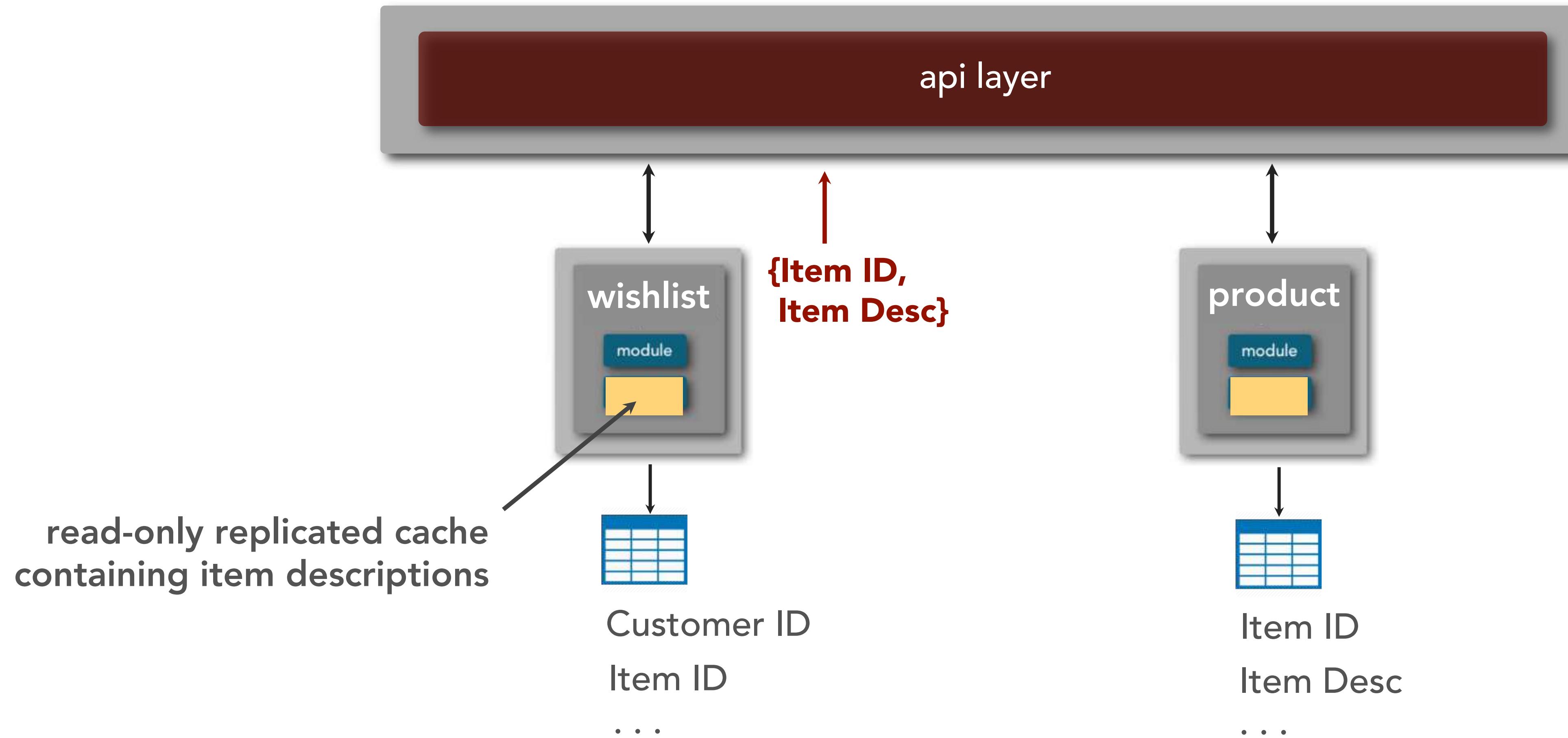
data access and sharing

option 3: in-memory replicated cache



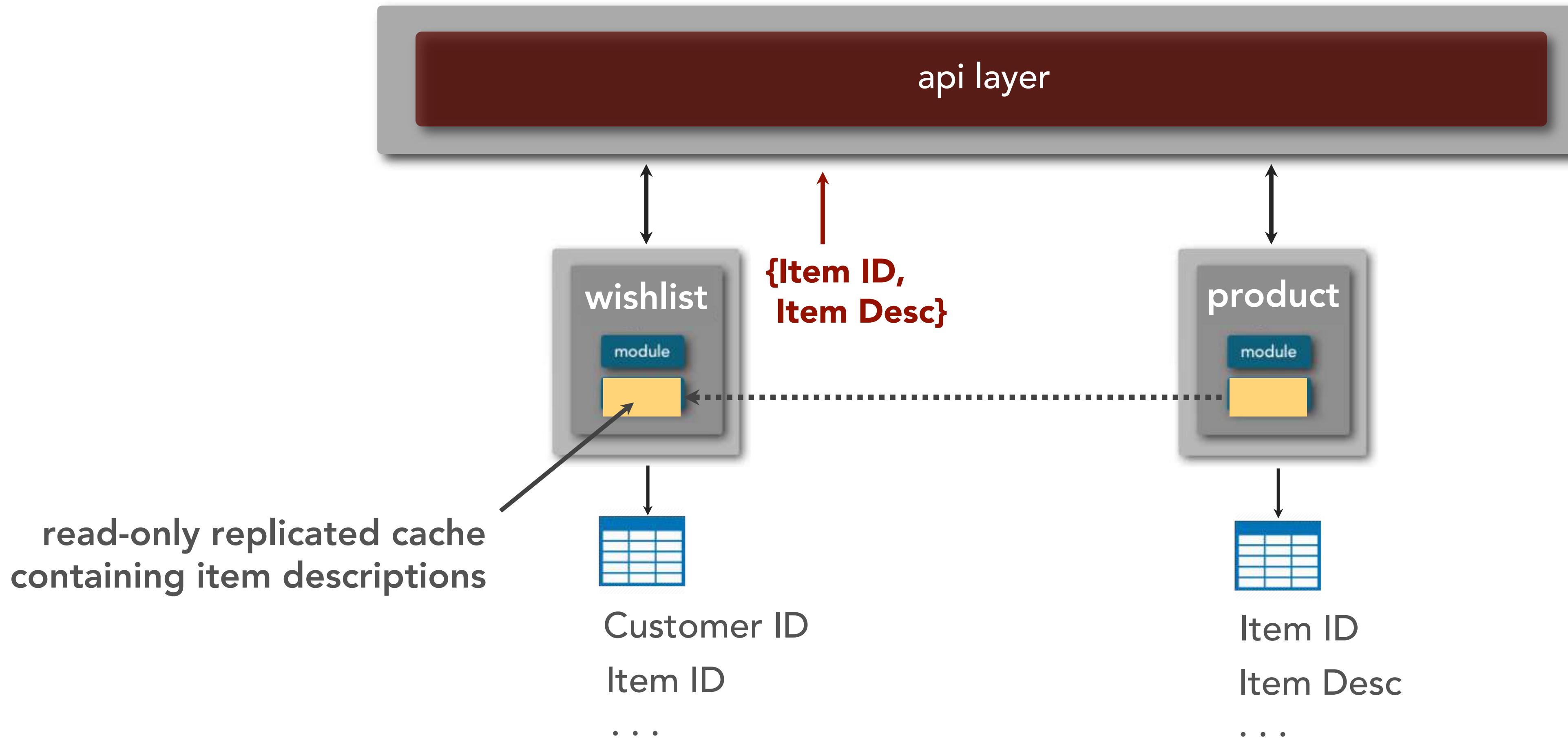
data access and sharing

option 3: in-memory replicated cache



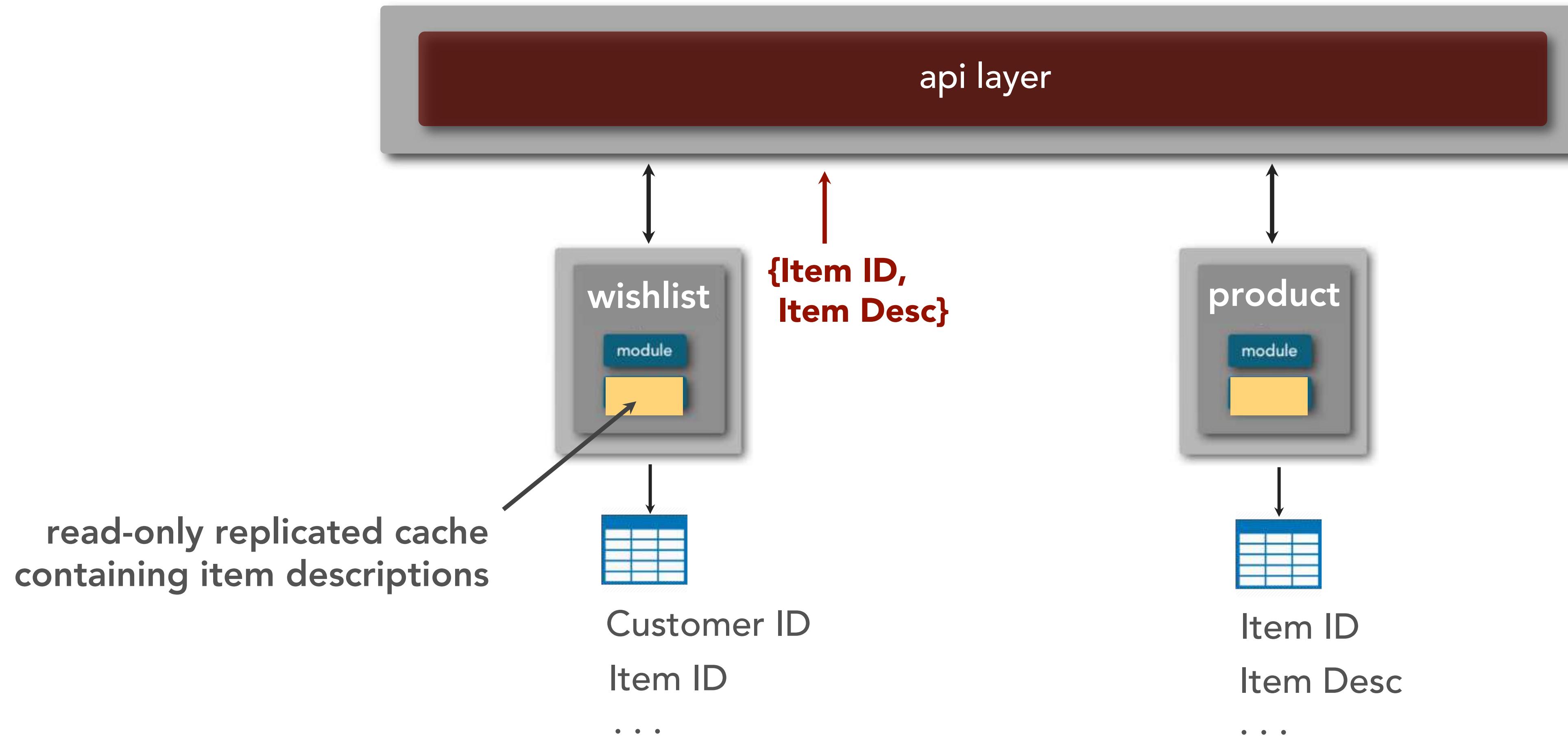
data access and sharing

option 3: in-memory replicated cache



data access and sharing

option 3: in-memory replicated cache



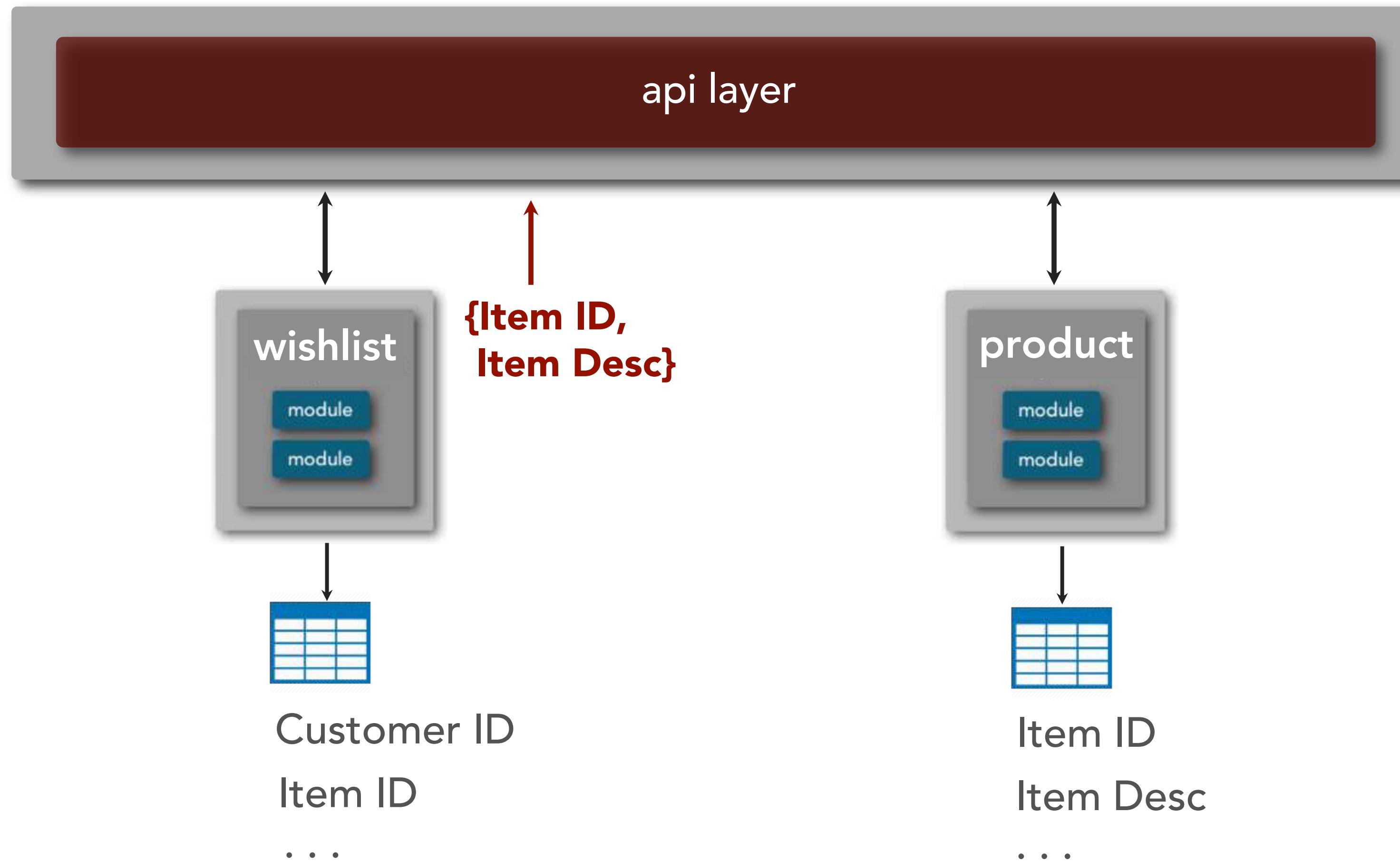
tradeoffs

option 3 - in-memory replicated cache

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

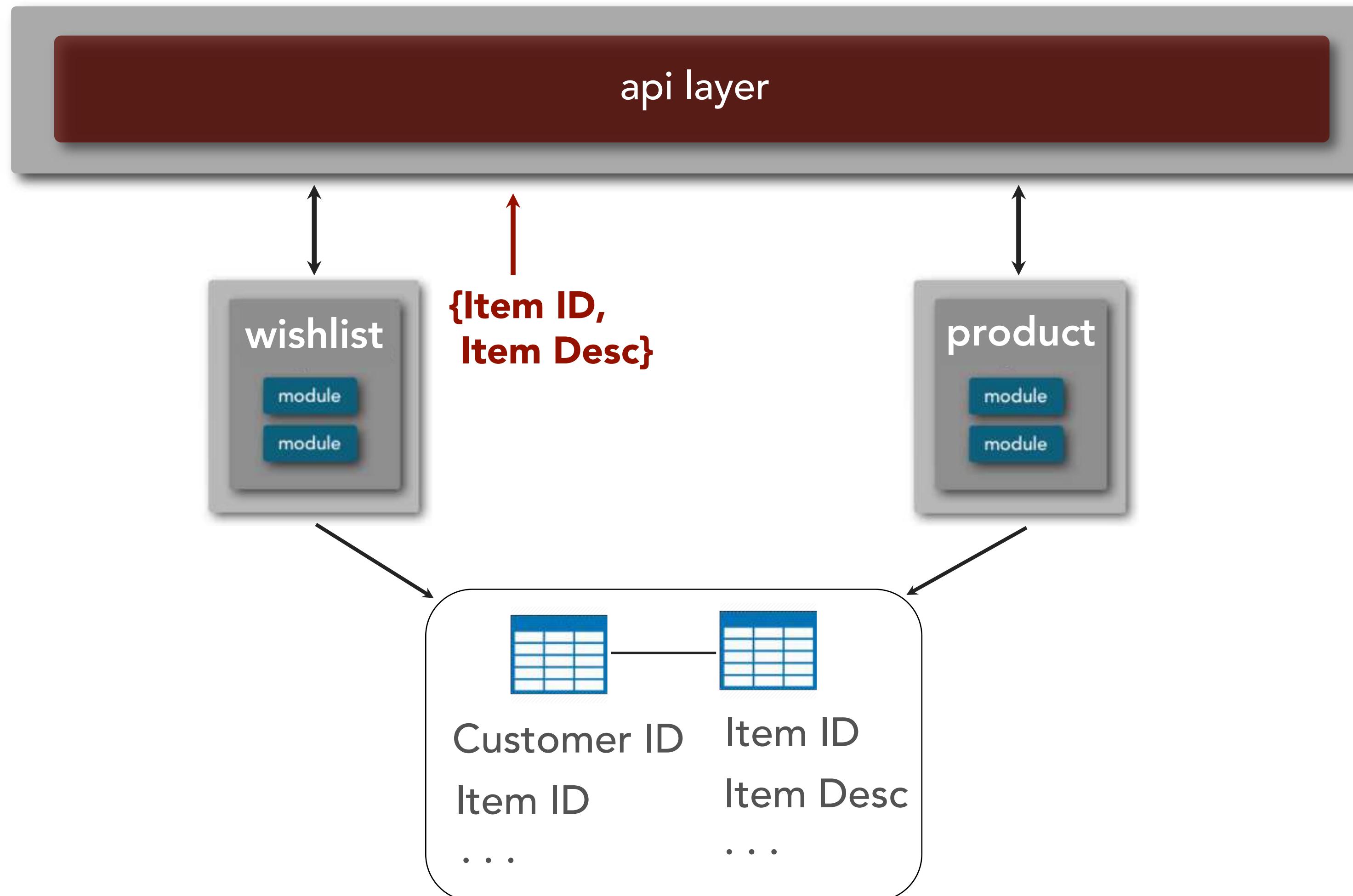
data access and sharing

option 4: data domain



data access and sharing

option 4: data domain



tradeoffs

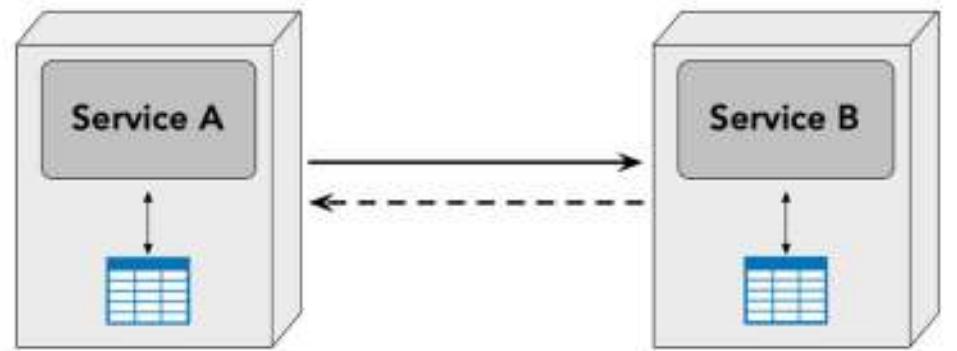
option 4 - data domain

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

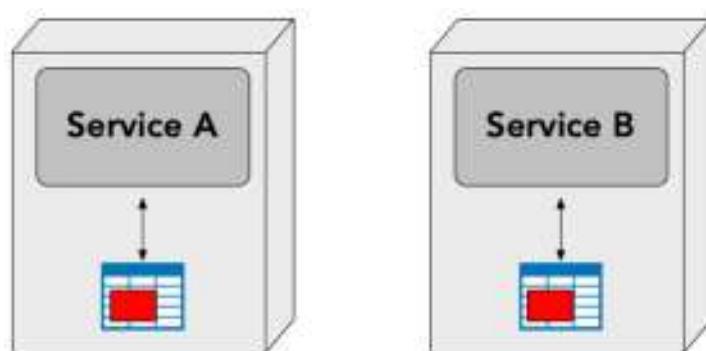
data access and sharing

which one is better?

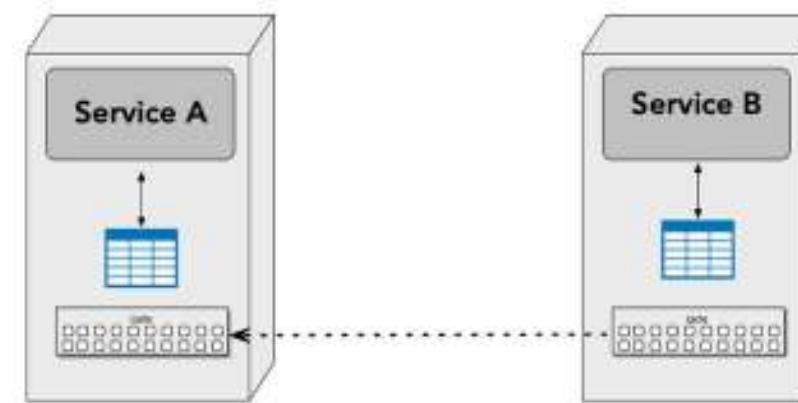
option 1:
interservice
communication



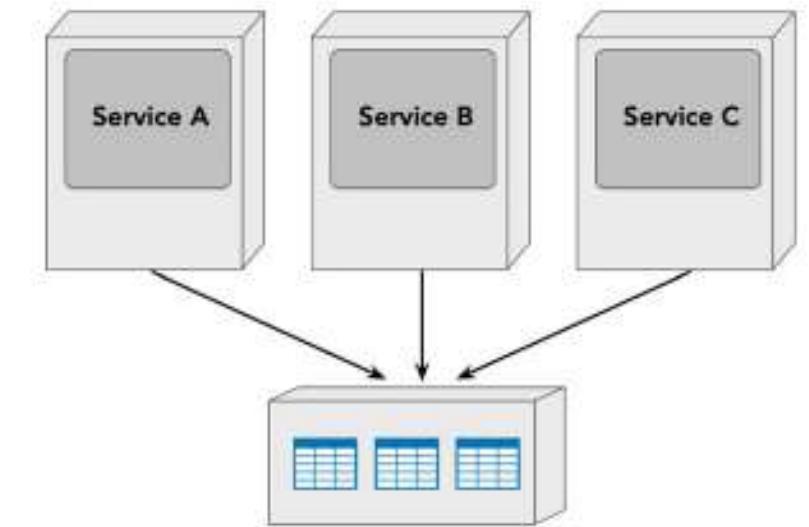
option 2:
data schema
replication



option 3:
in-memory
replicated cache



option 4:
data domains
(shared tables)



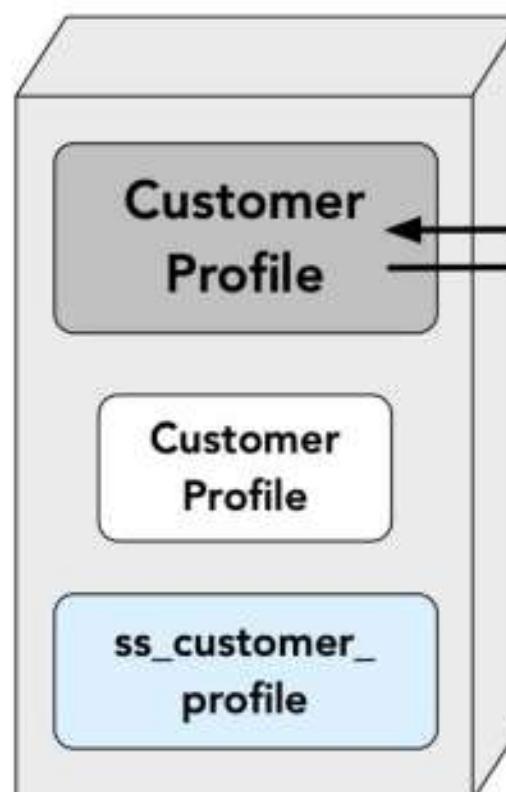
Scenario Exercise - Data Access

The notification service needs the customer's contact info and name to notify them that the expert is on their way. How should the notification service get this information?

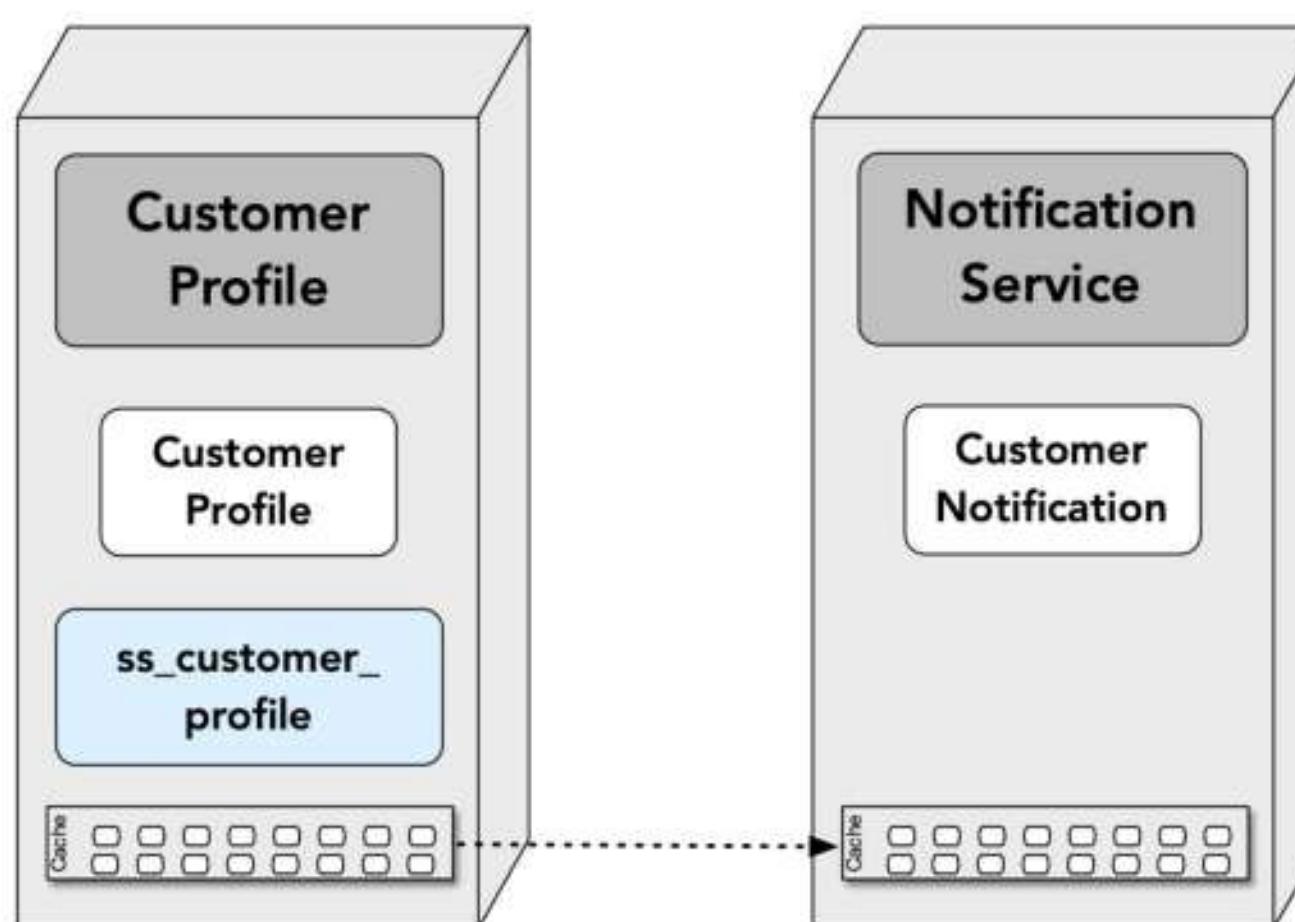
Additional Info: Number of customers: 10,000, growth rate 5%/year

Customer ID is passed into the notification service

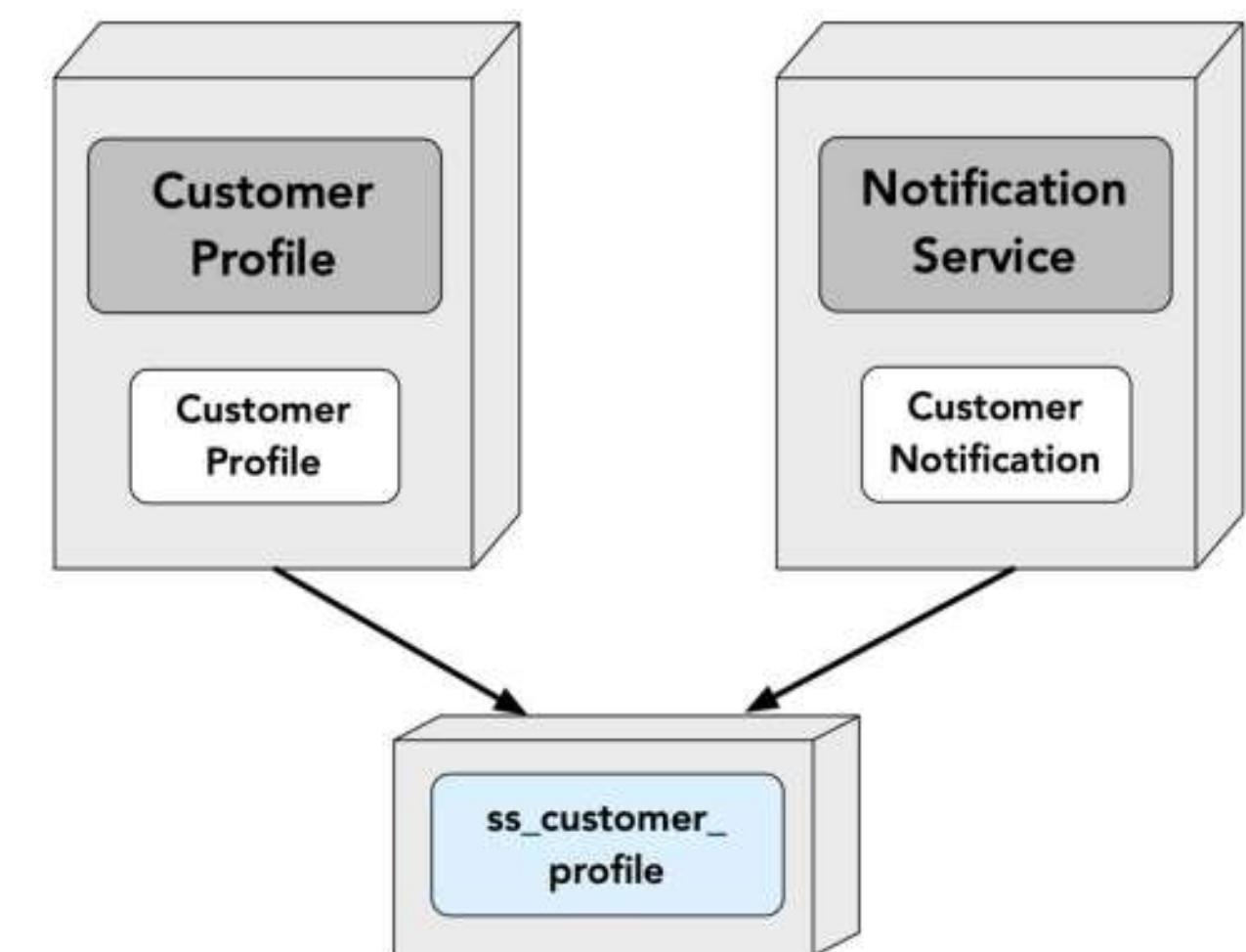
ss_customer_profile contains all name and contact info



option 1: interservice communication



option 2: replicated in-memory cache



option 3: data domain

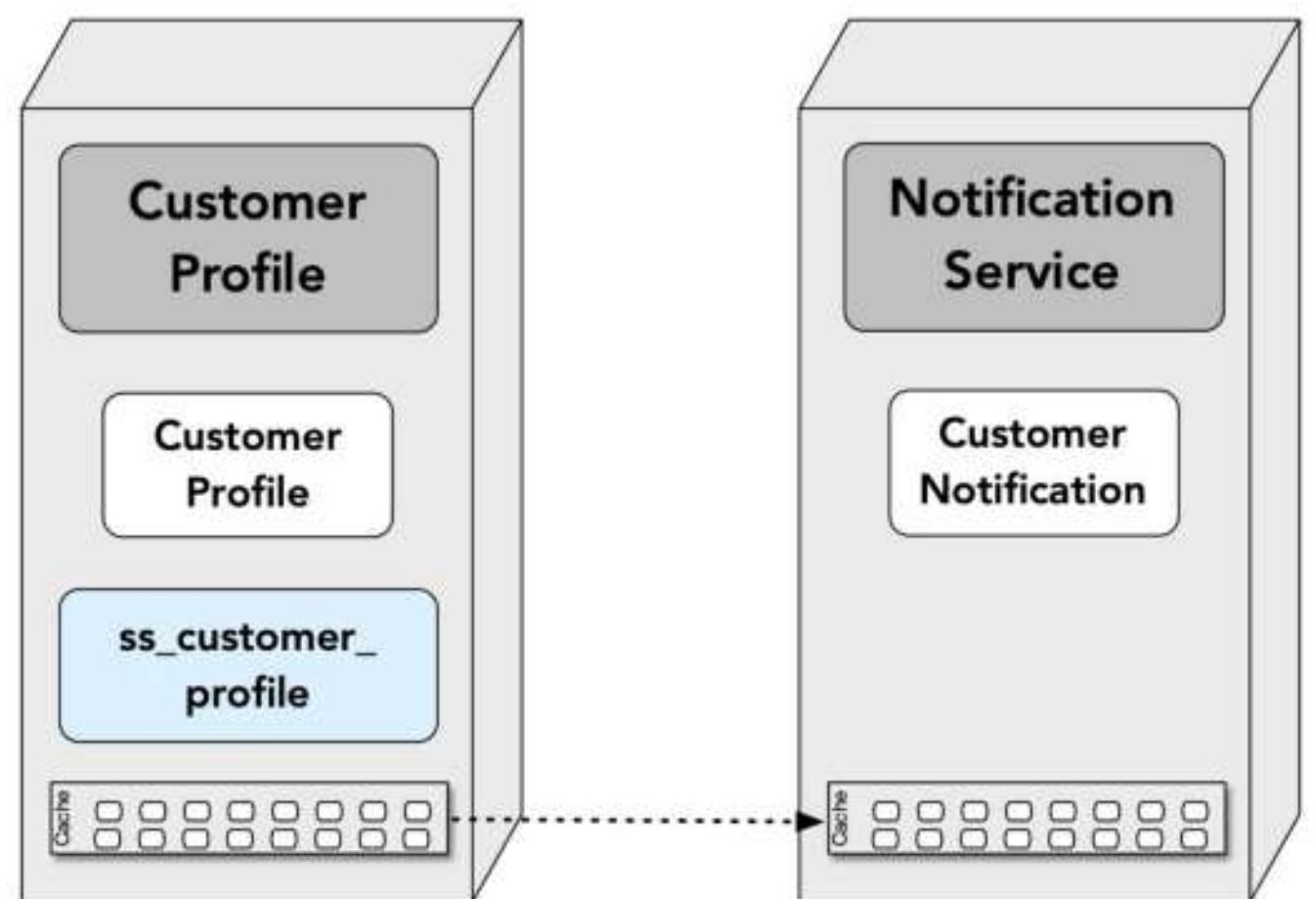
Scenario Exercise - Data Access (Instructor)

The notification service needs the customer's contact info and name to notify them that the expert is on their way. How should the notification service get this information?

Additional Info: Number of customers: 10,000, growth rate 5%/year

Customer ID is passed into the notification service

ss_customer_profile contains all name and contact info



Less service and data coupling, hence better responsiveness (no network and security latency) and fault tolerance

In-memory replicated cache creates a value-driven contract between the services

Relatively static data (low update rate), low data volume (~400MB)

option 2: replicated in-memory cache



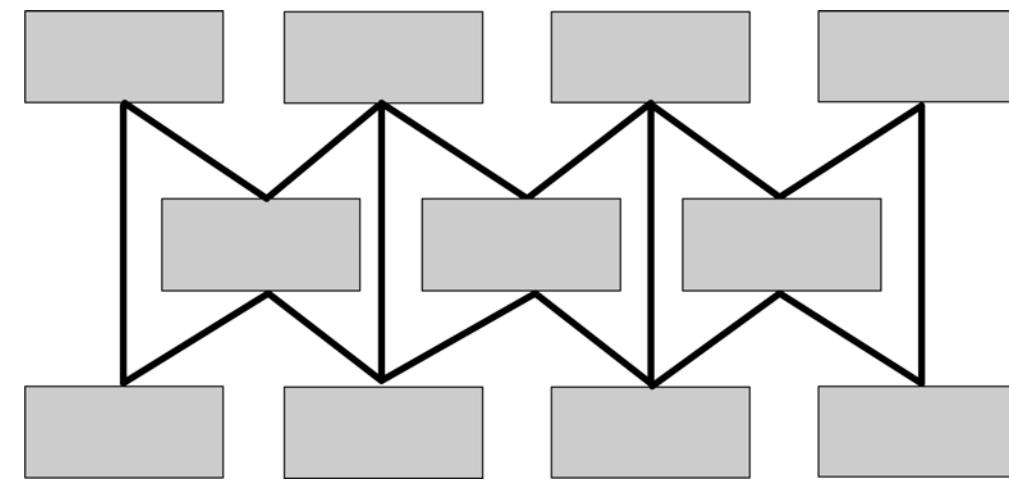
*How do I achieve high levels
of scalability and elasticity in
a system?*

scenario: ↑ volume concert

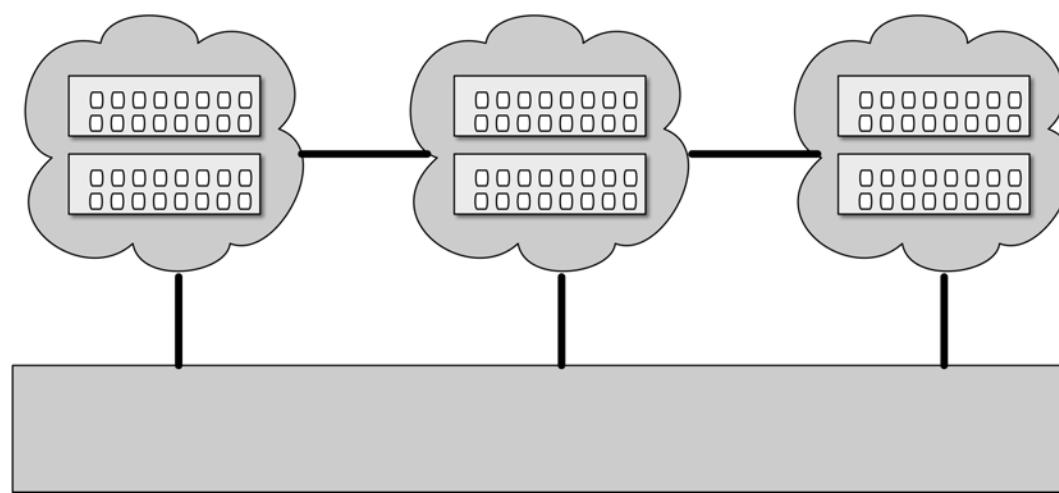


- Lots of concurrent users
- huge bursts of users when new shows go on sale
- highly transactional !

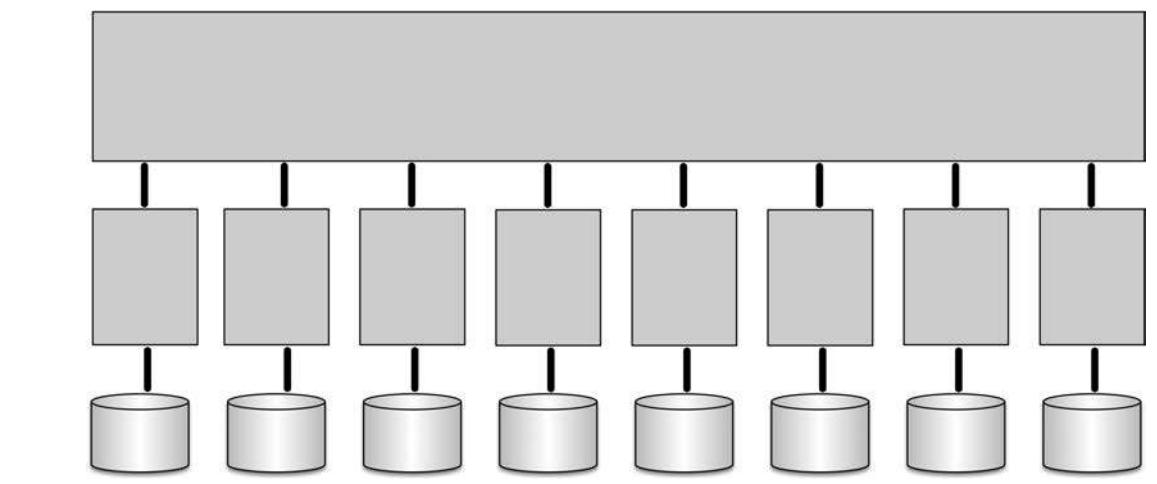
scalability ∪ elasticity



Event-driven
architecture

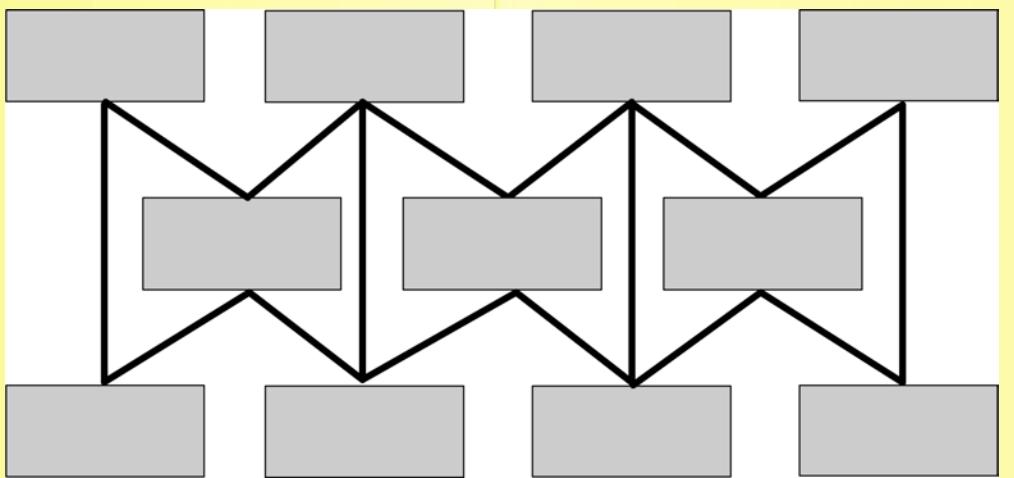


Space-based
architecture



Microservices
architecture

tradeoffs

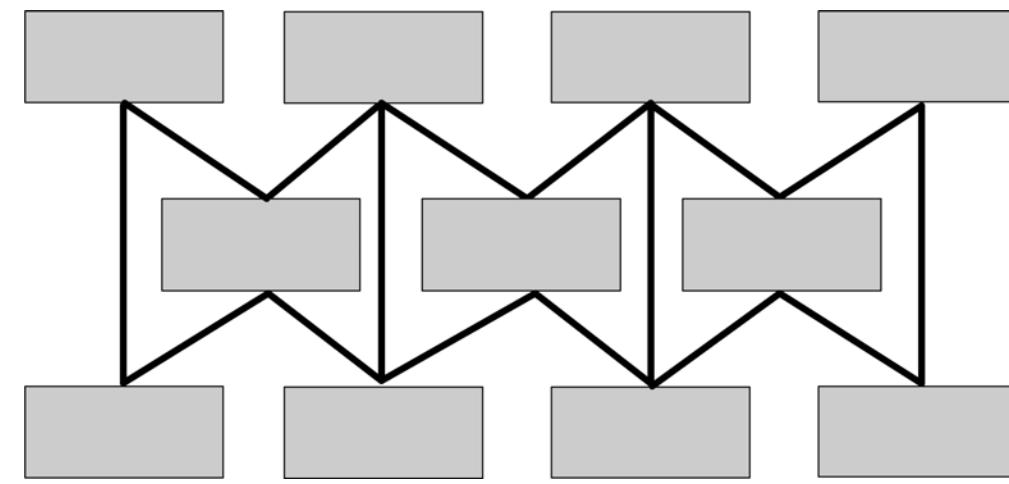


EDA

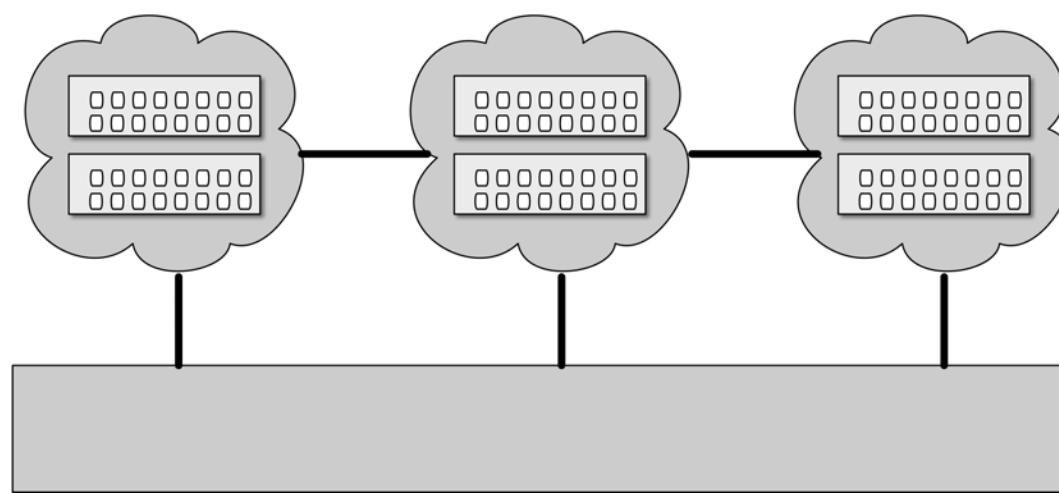
+ highly performant and scalable

- very complex for simple things
(instancing, discovery, etc)
- traditional headaches of
message-based systems
- granularity issues with elasticity
- too much build-it-yourself

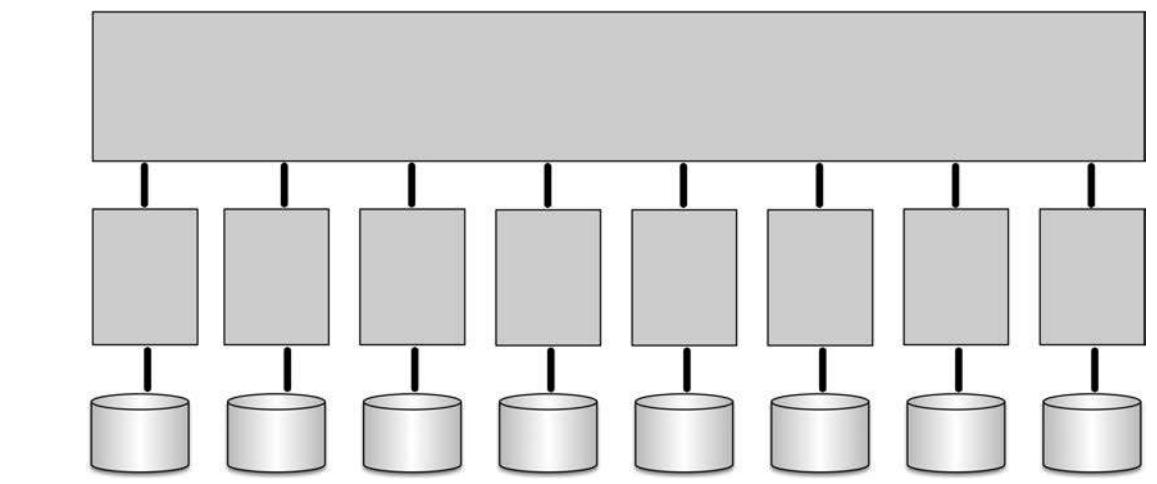
scalability ∪ elasticity



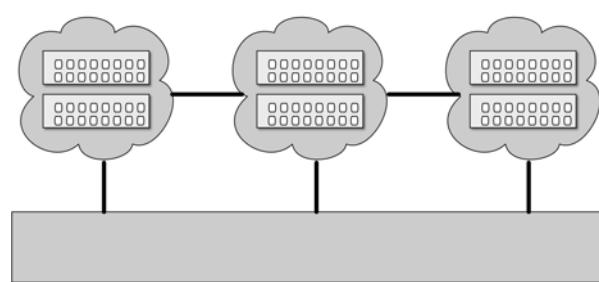
Event-driven
architecture



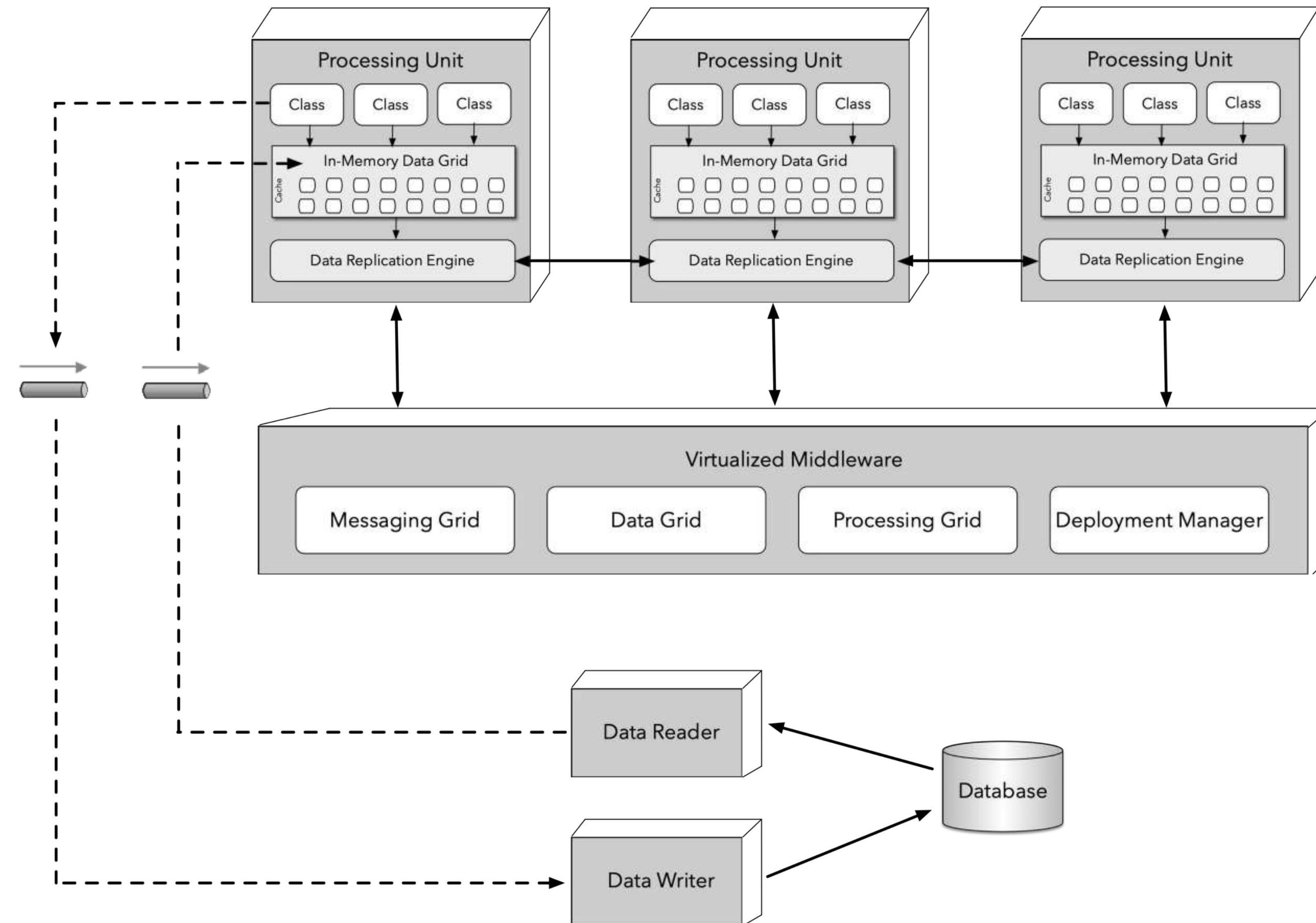
Space-based
architecture



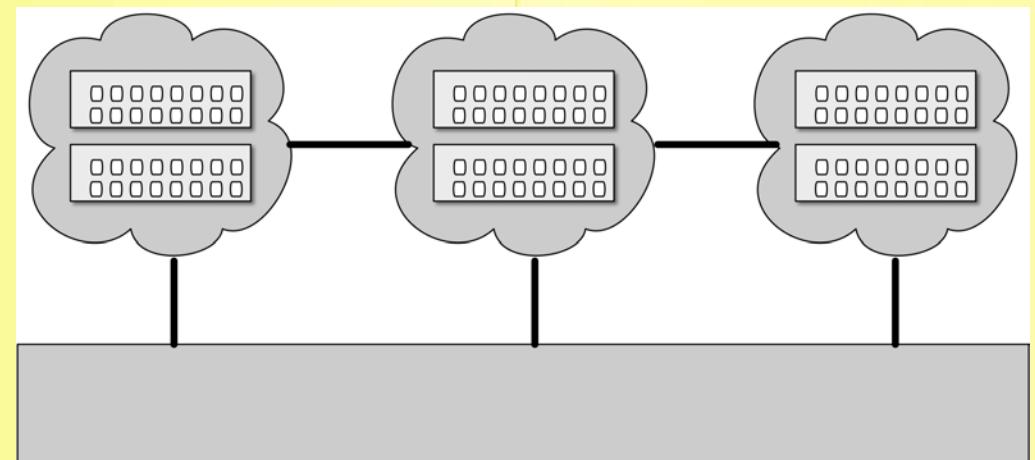
Microservices
architecture



space-based architecture



tradeoffs

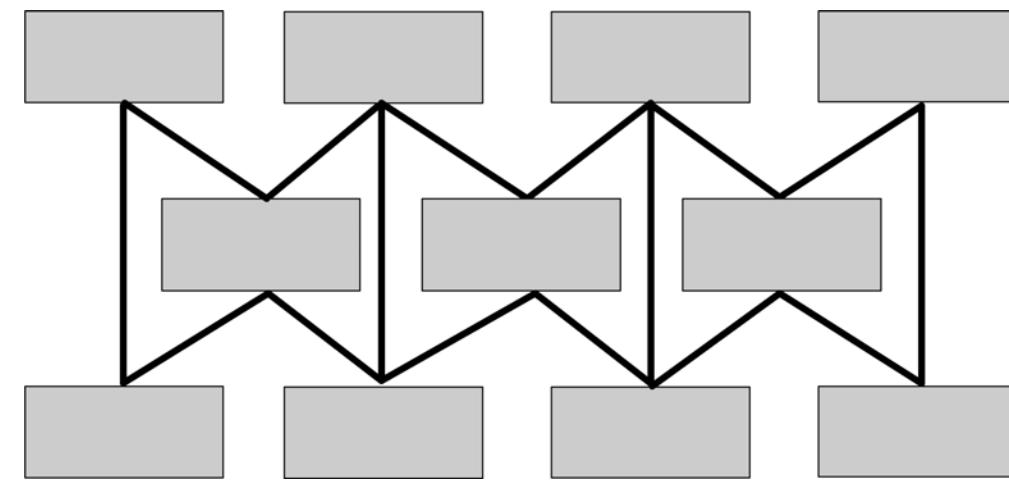


Space-based

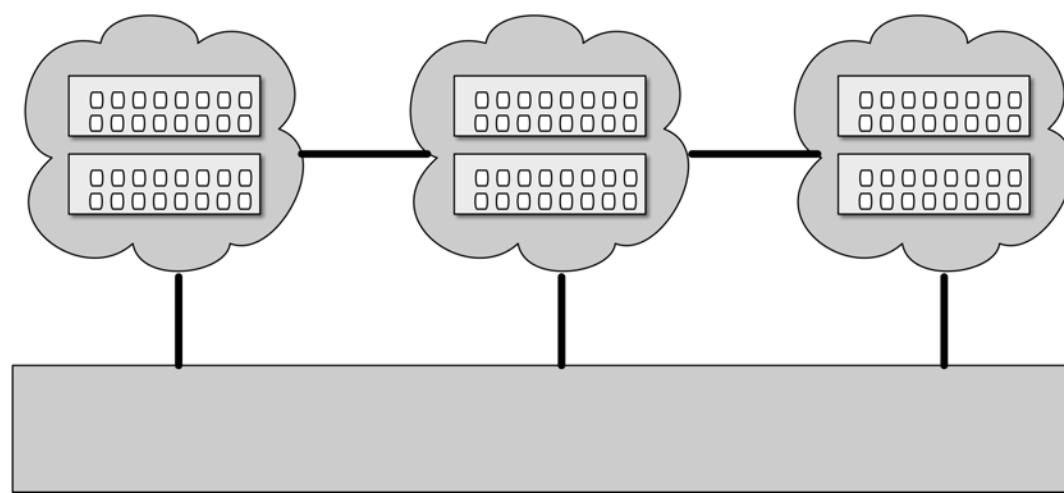
+ hyper-optimized for elastic scale

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

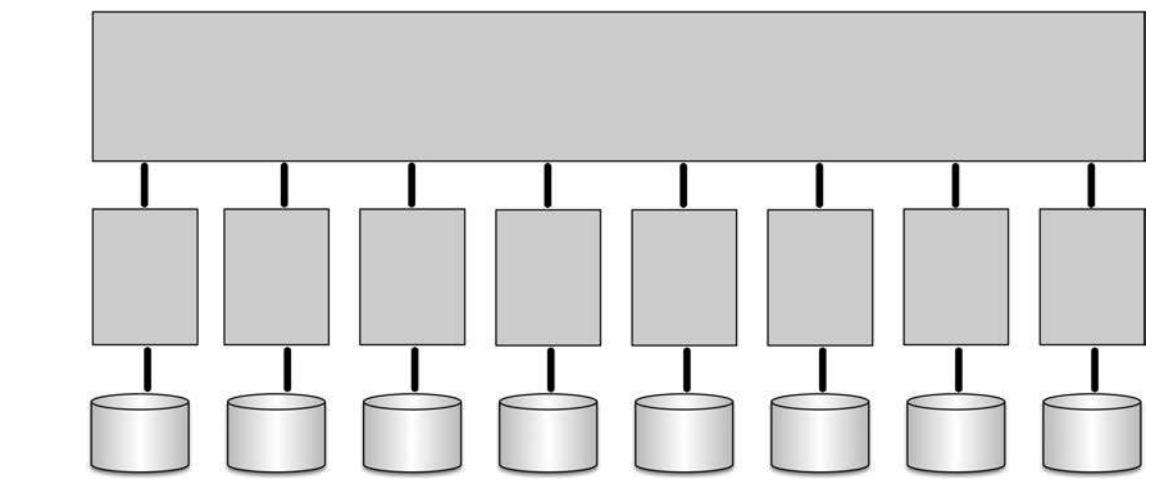
scalability ∪ elasticity



Event-driven
architecture

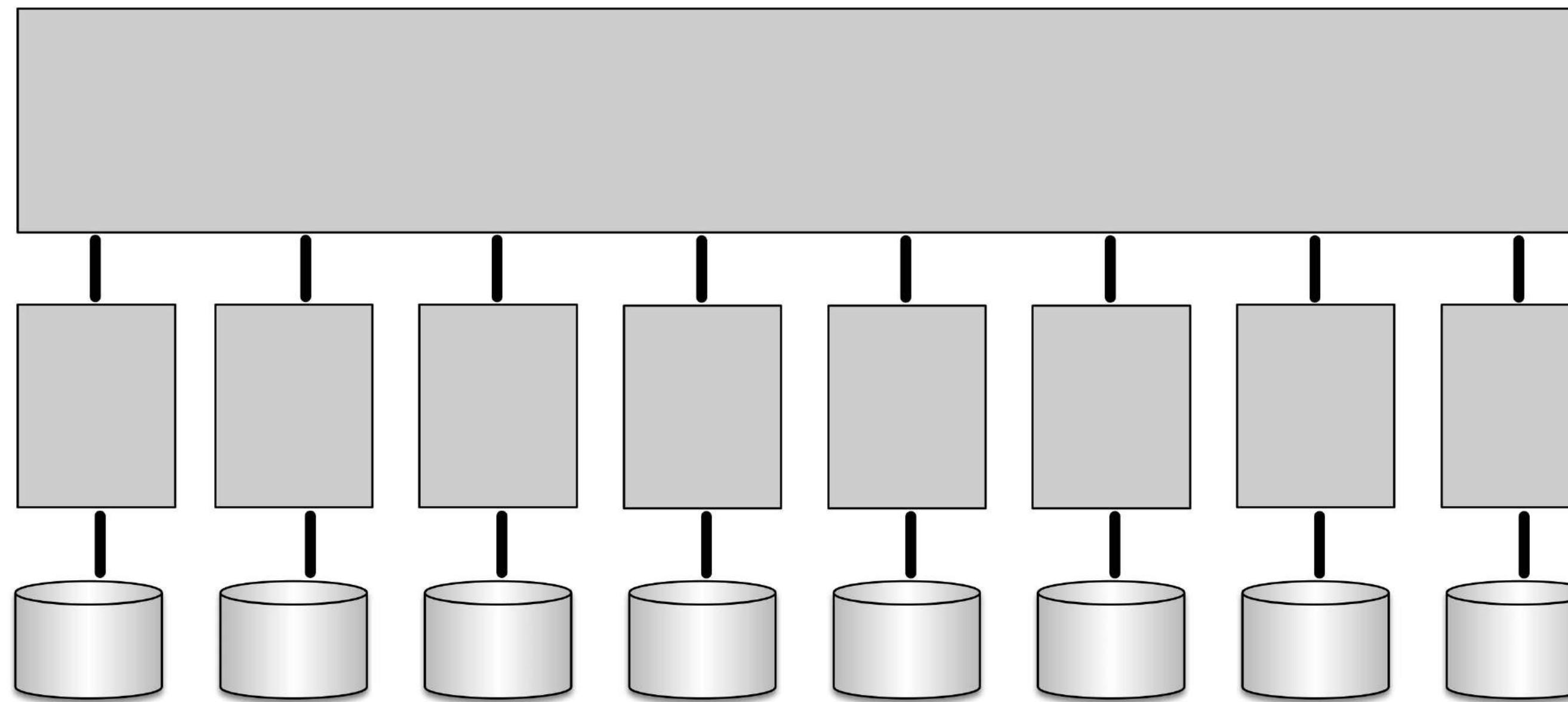


Space-based
architecture

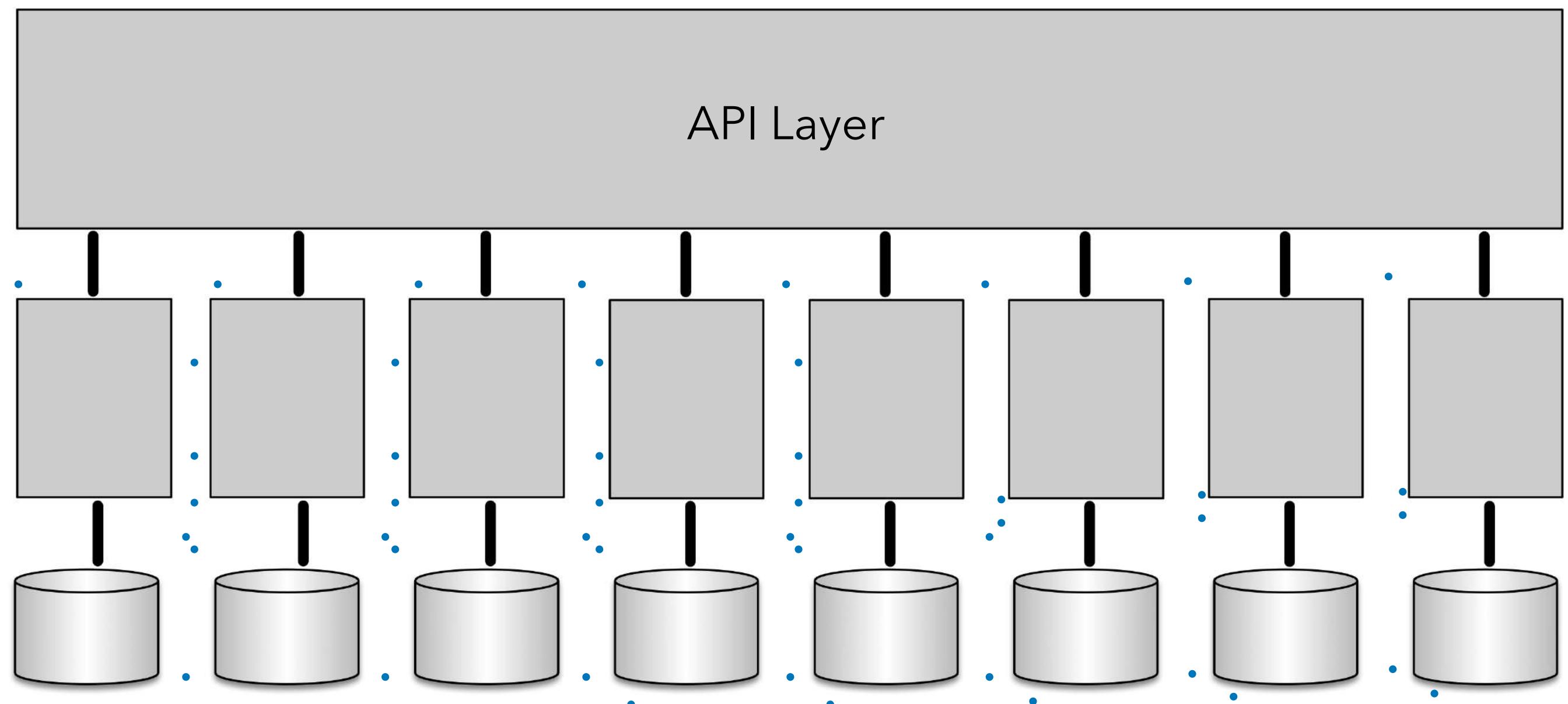


Microservices
architecture

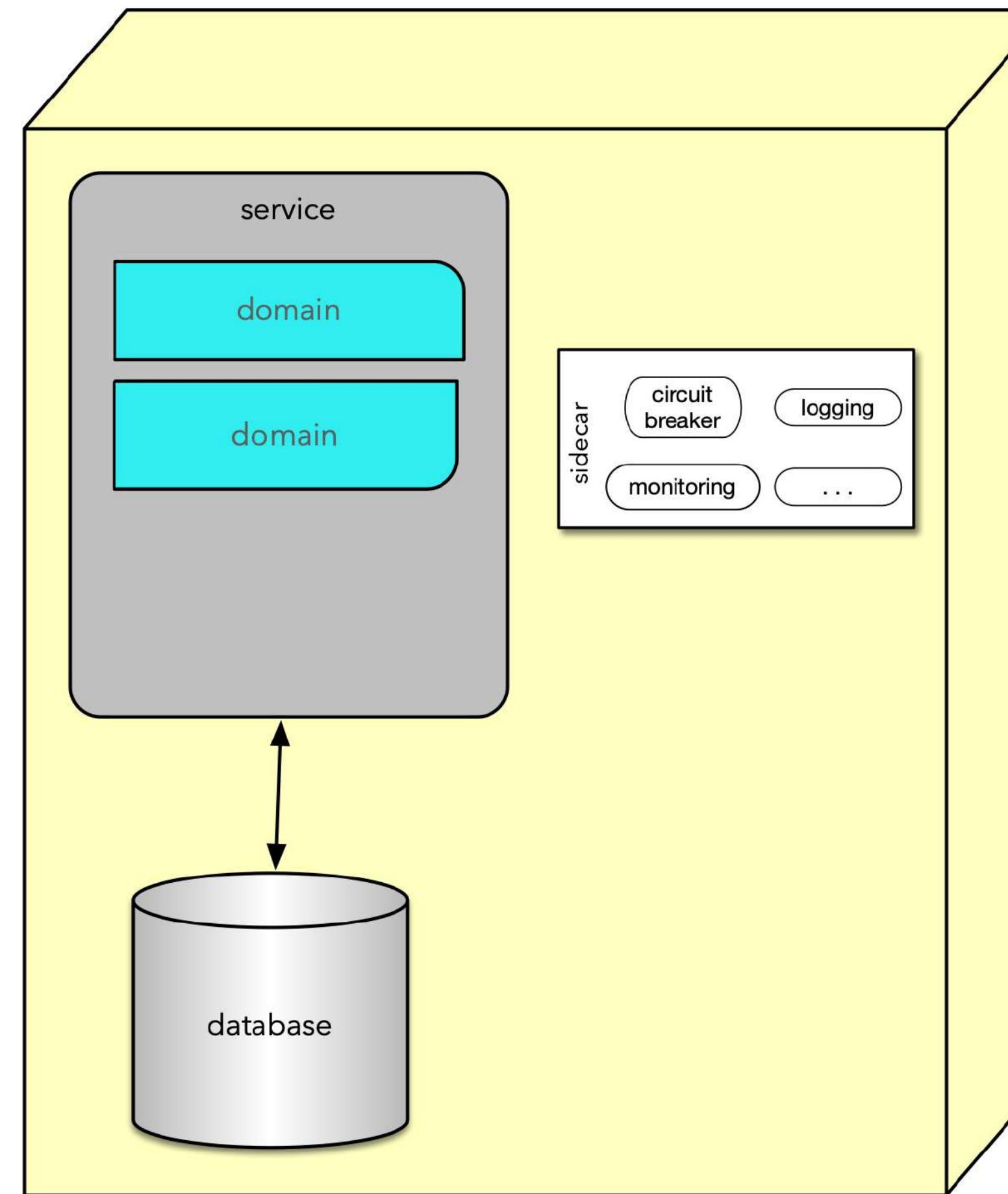
elasticity in microservices



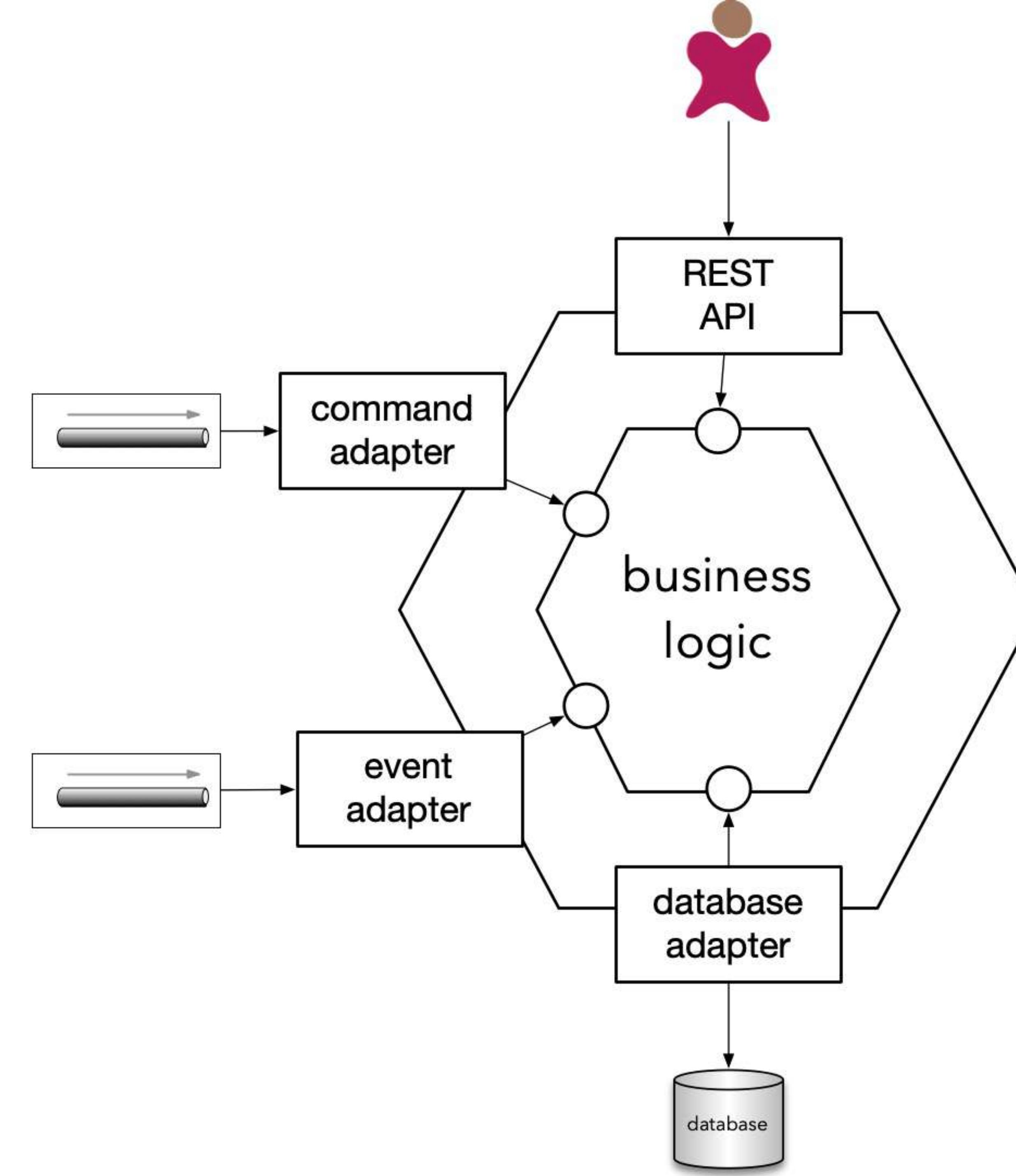
elasticity in microservices



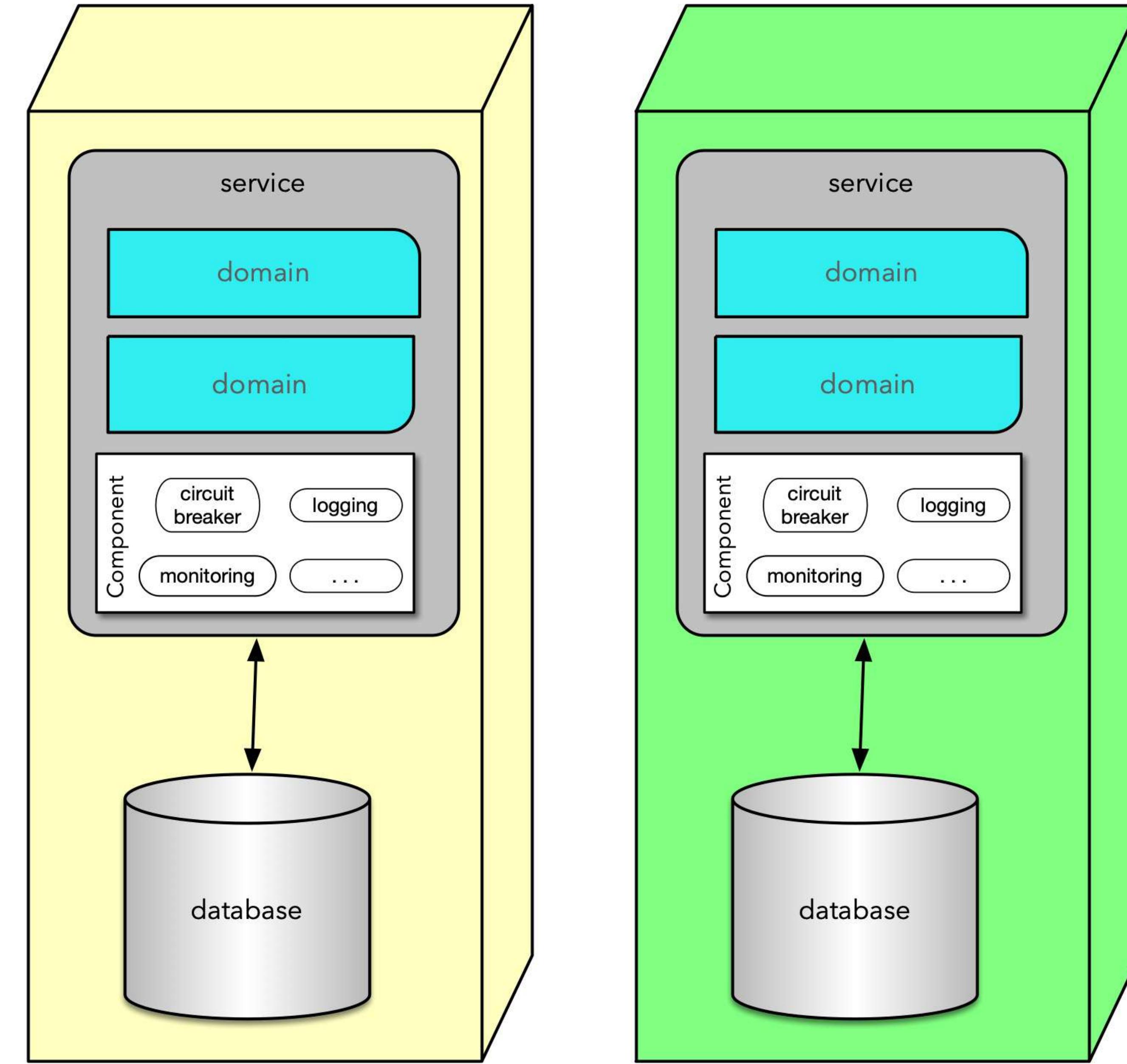
operational reuse in microservices: the sidecar pattern



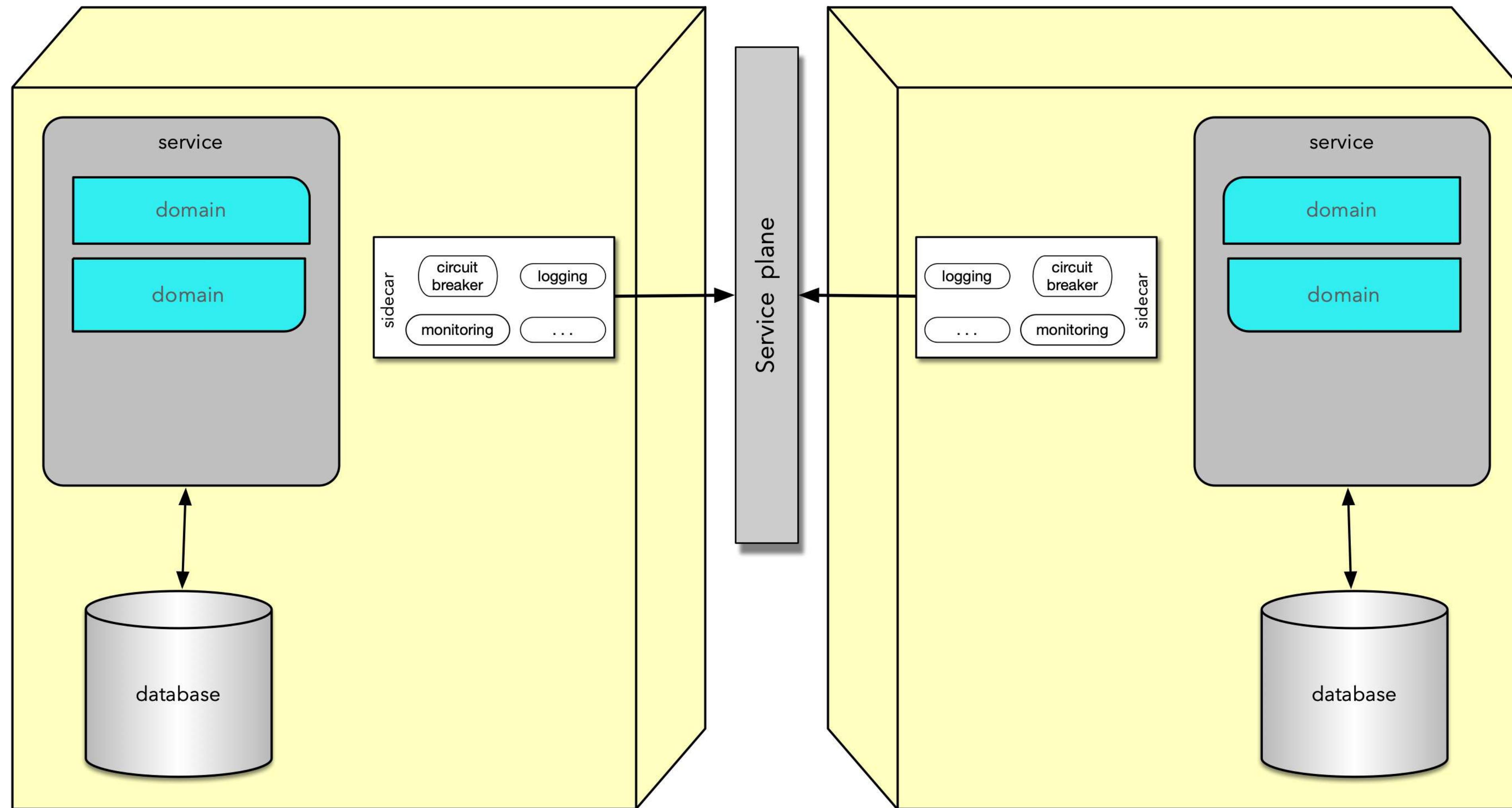
hexagonal pattern



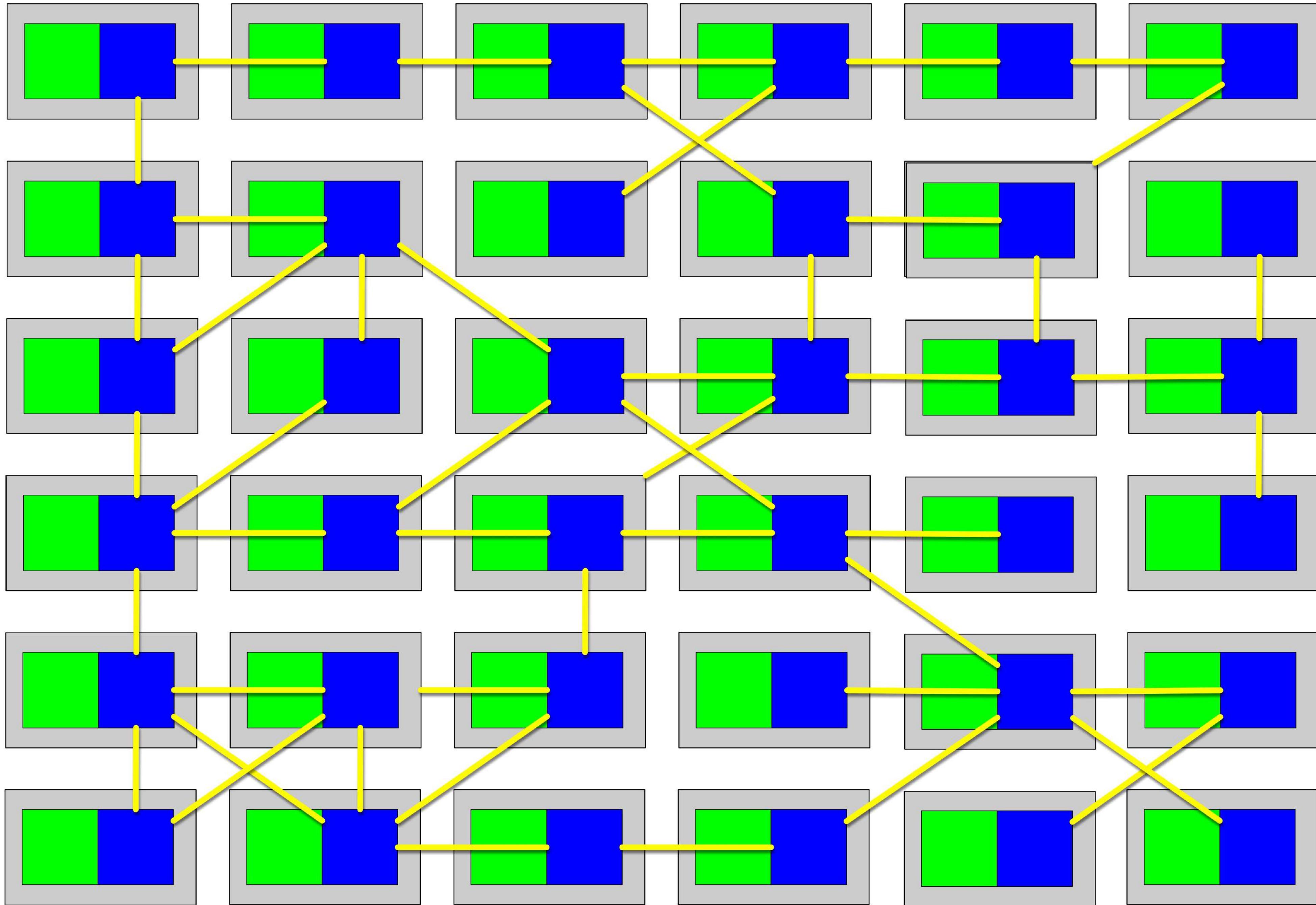
operational reuse



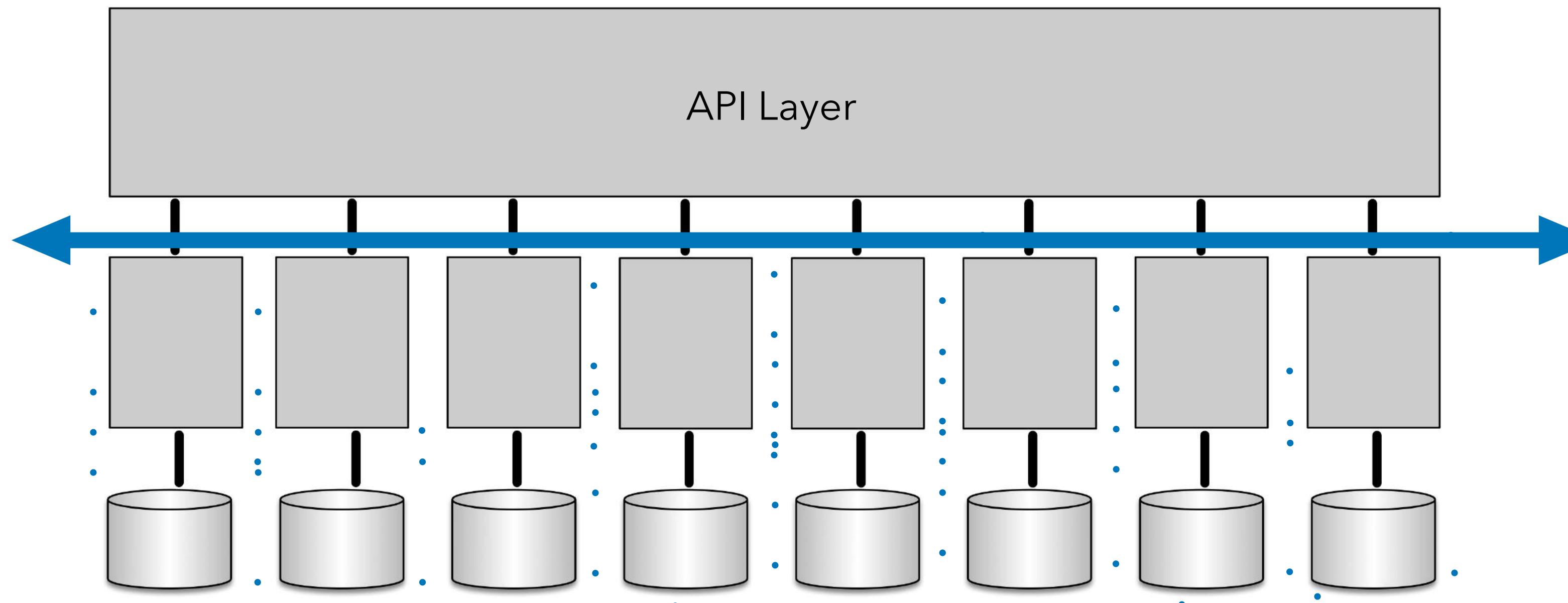
the Sidecar pattern



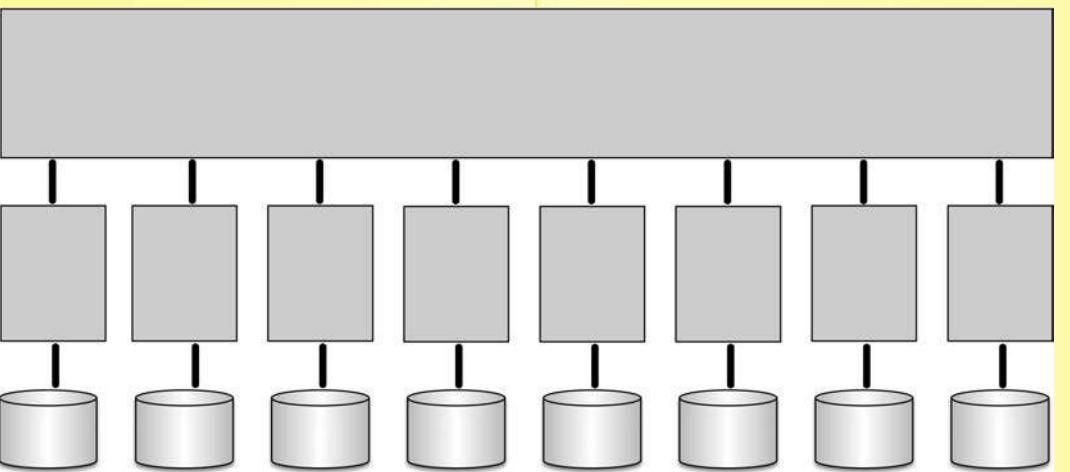
Service mesh



elasticity in microservices

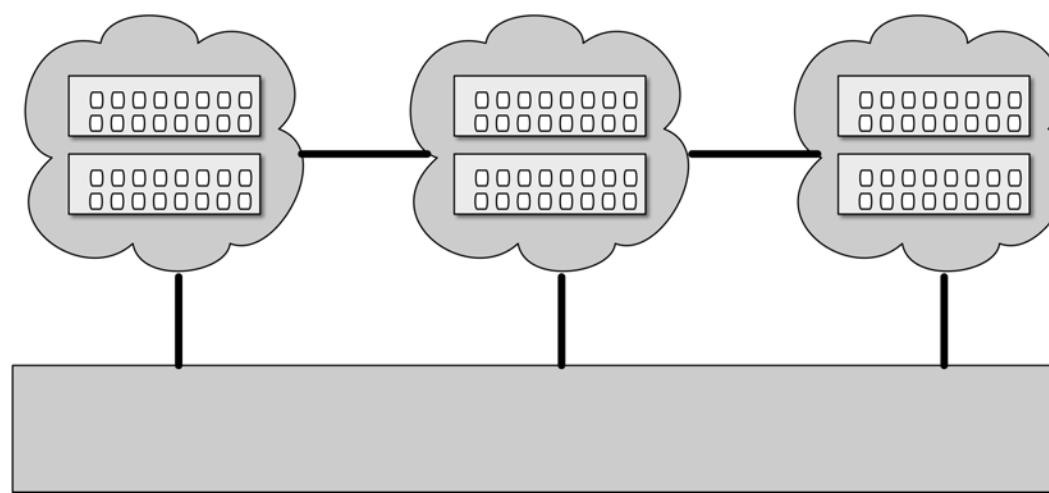
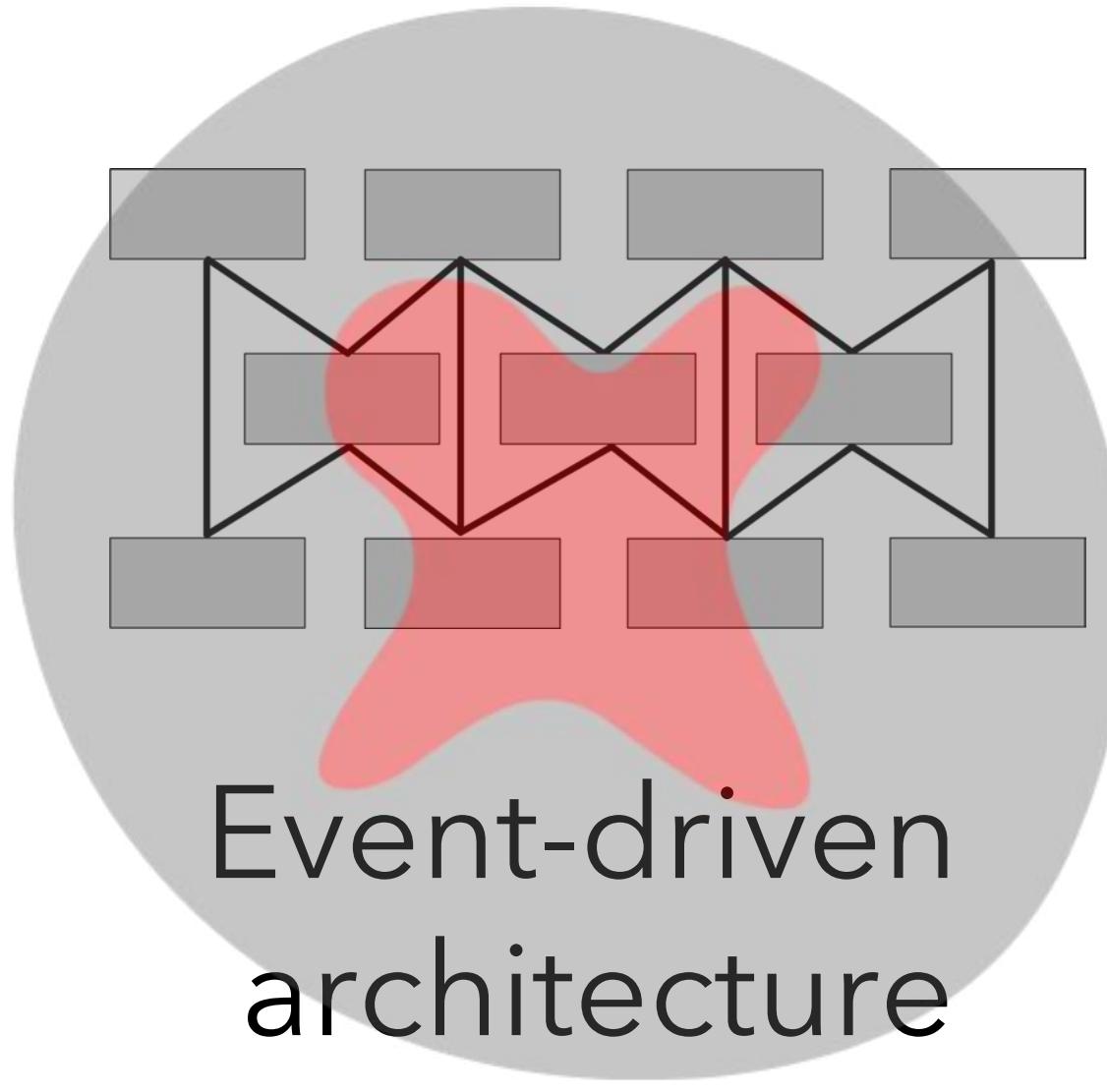


tradeoffs

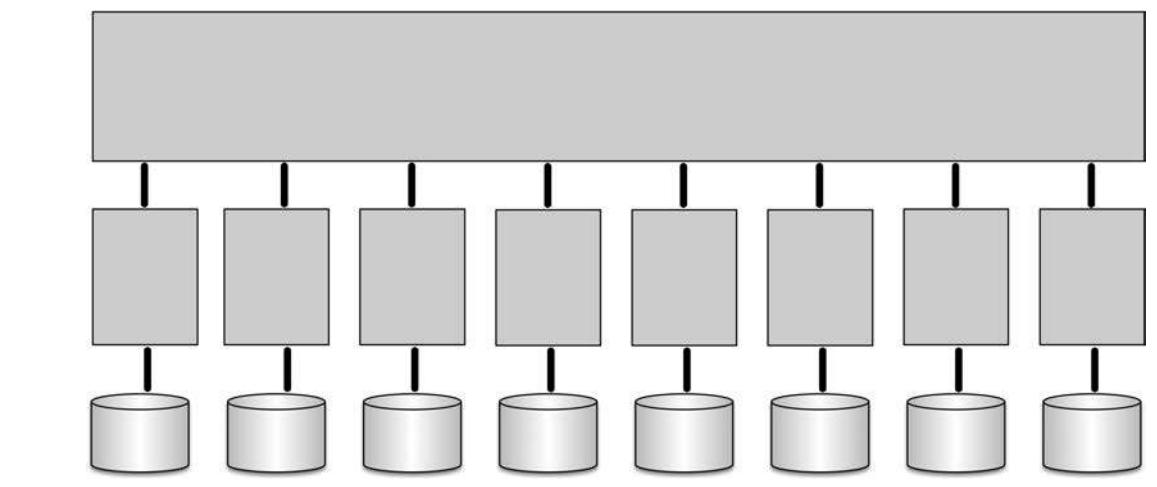


- + scalability achieved via service discovery + service mesh + orchestration
- + follows current trends
- + domain partitioned; low(est) incidental coupling
- + extremely good MTTS (mean-time-to-startup)
- architects must build transactional behavior

scalability ∪ elasticity



Space-based
architecture

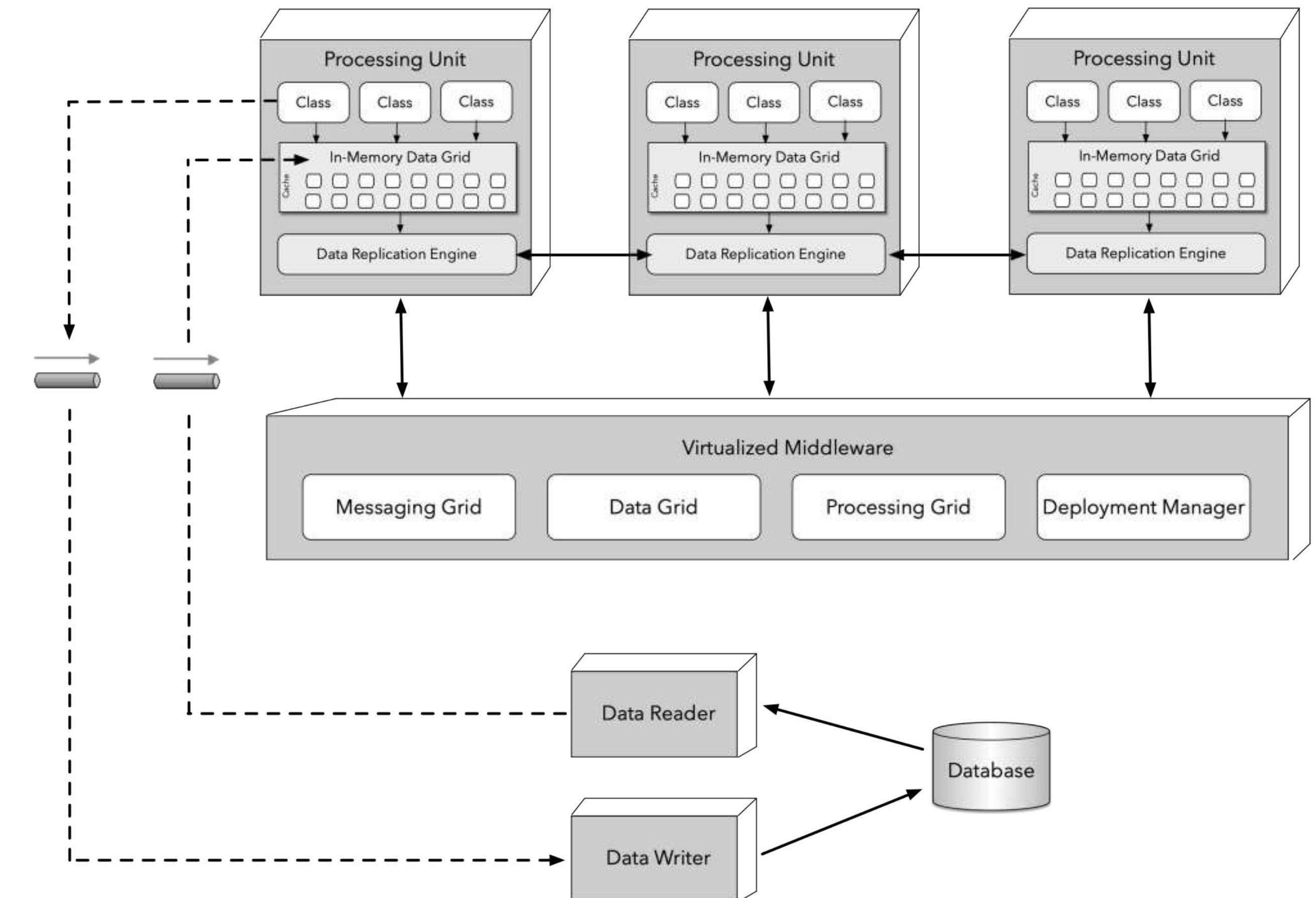
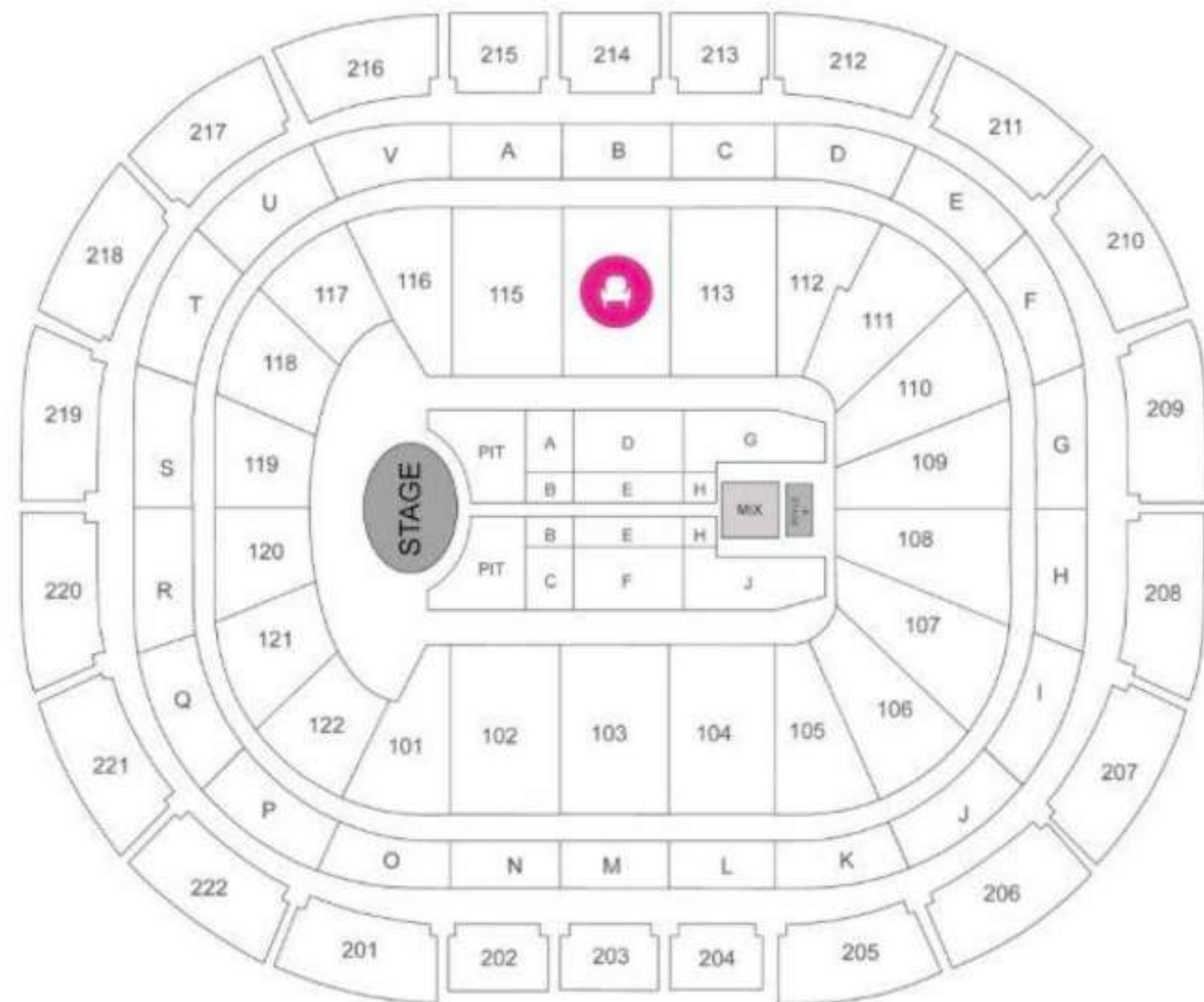


Microservices
architecture

space-based



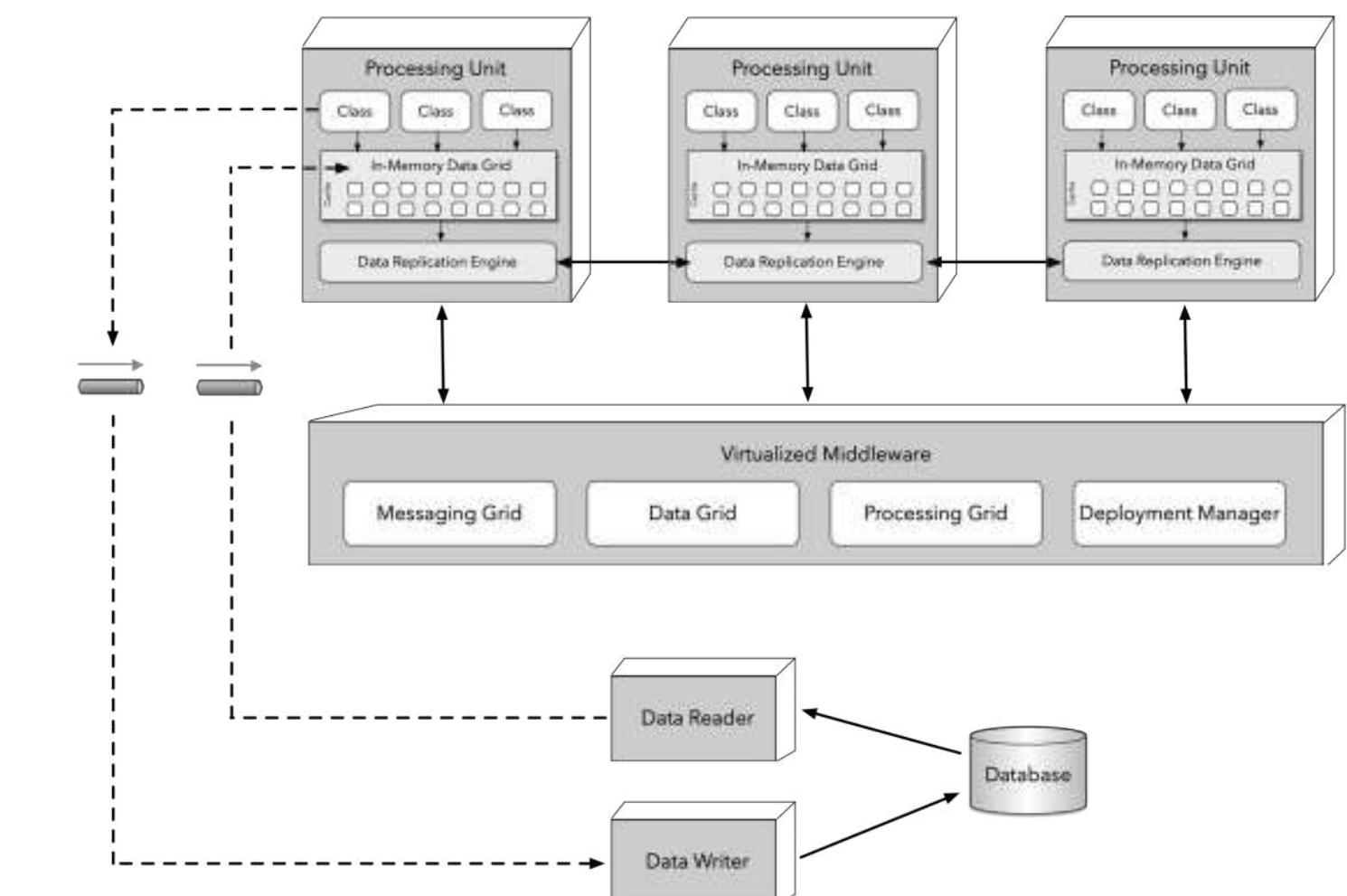
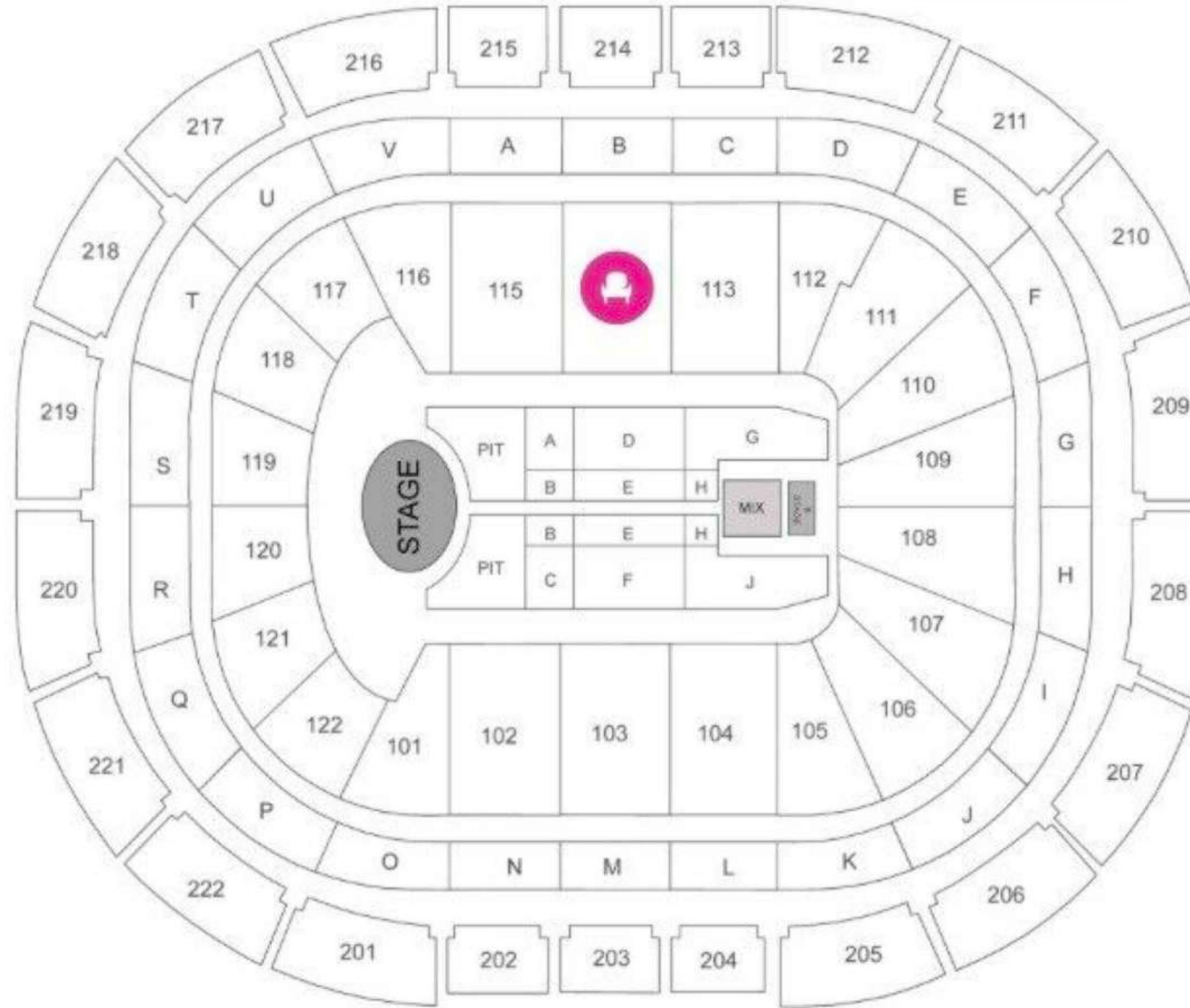
: transactions



space-based



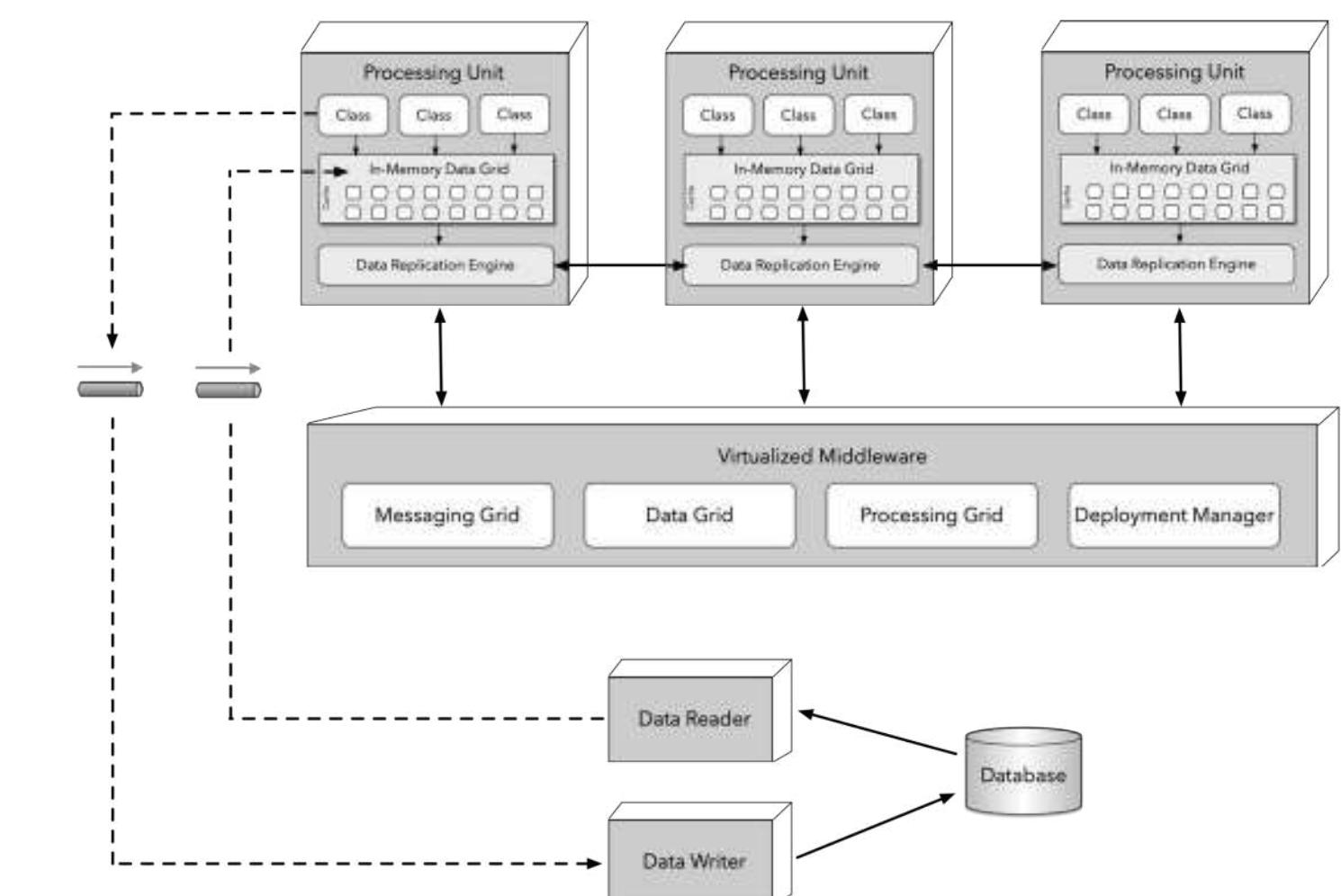
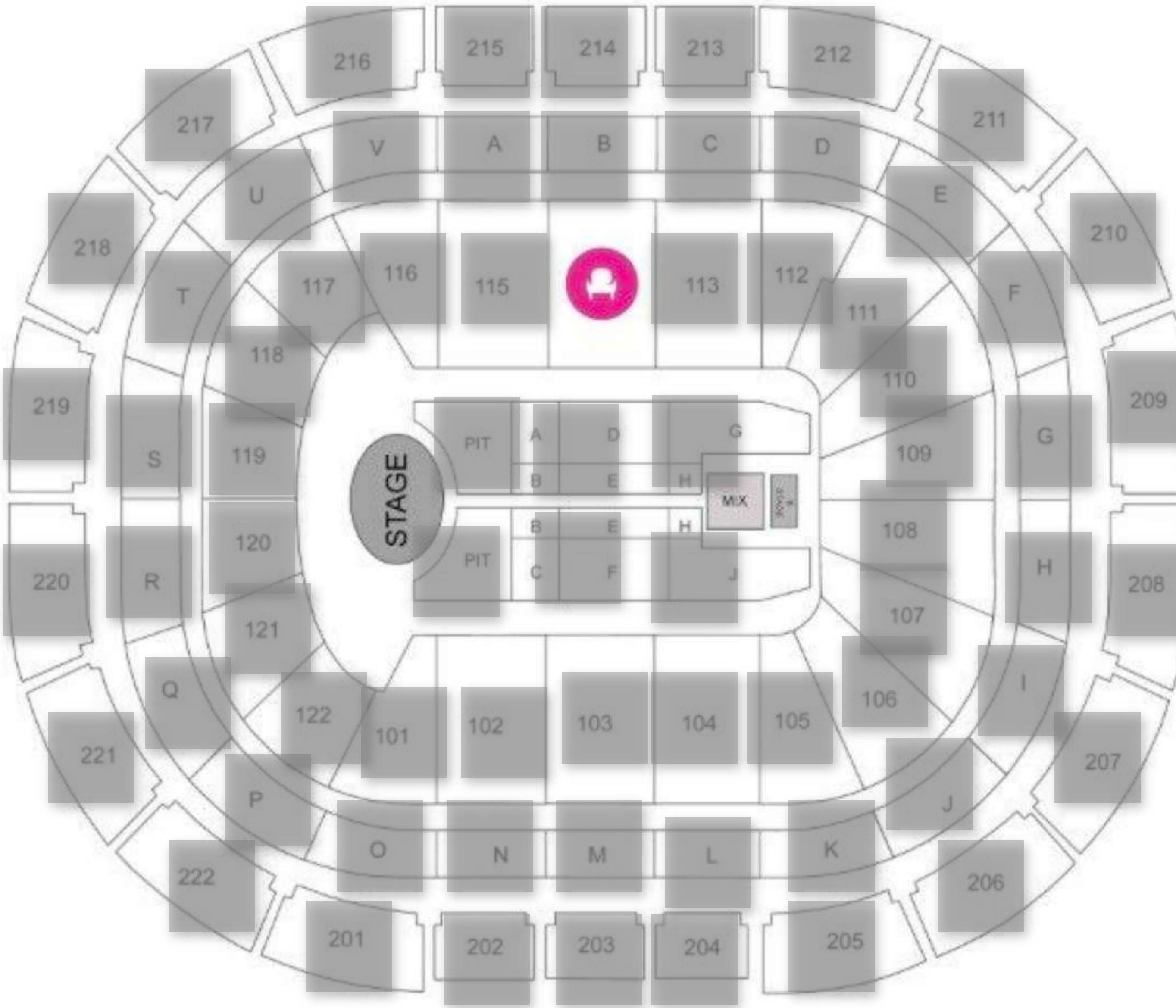
: transactions



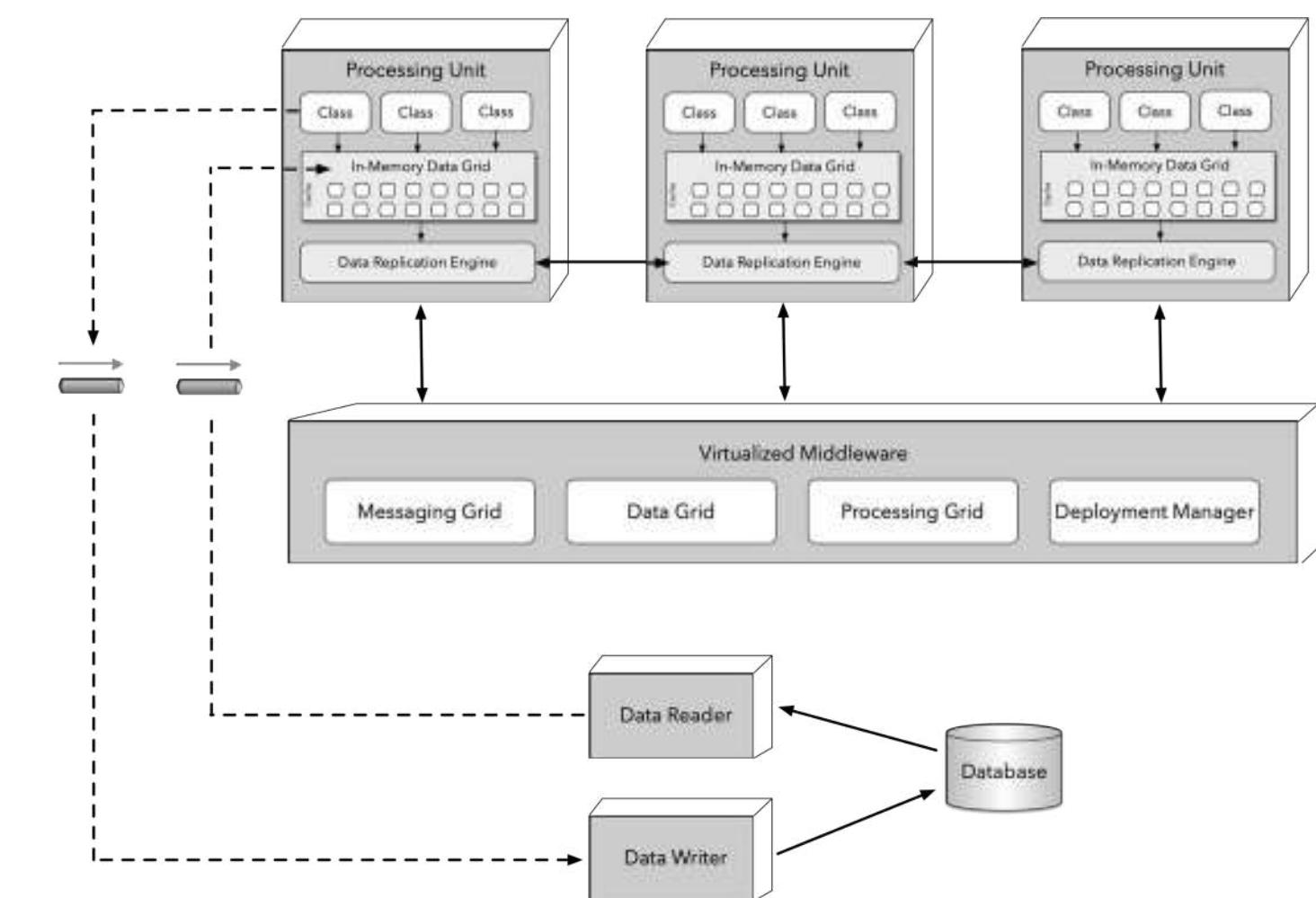
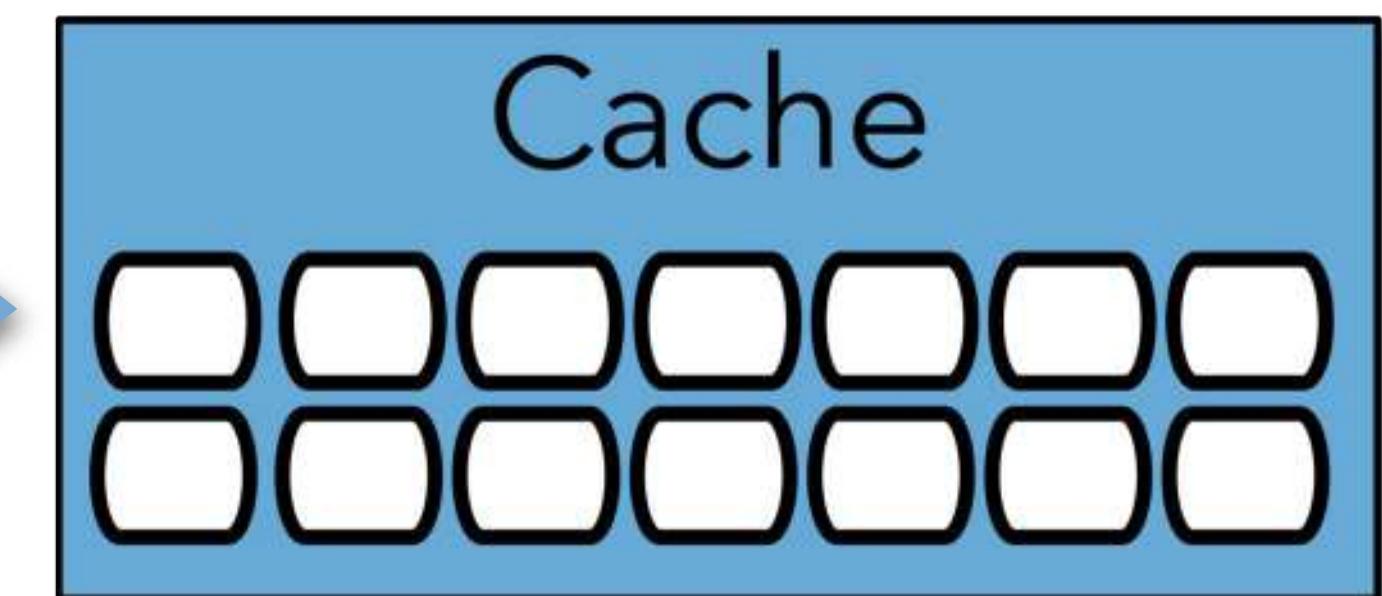
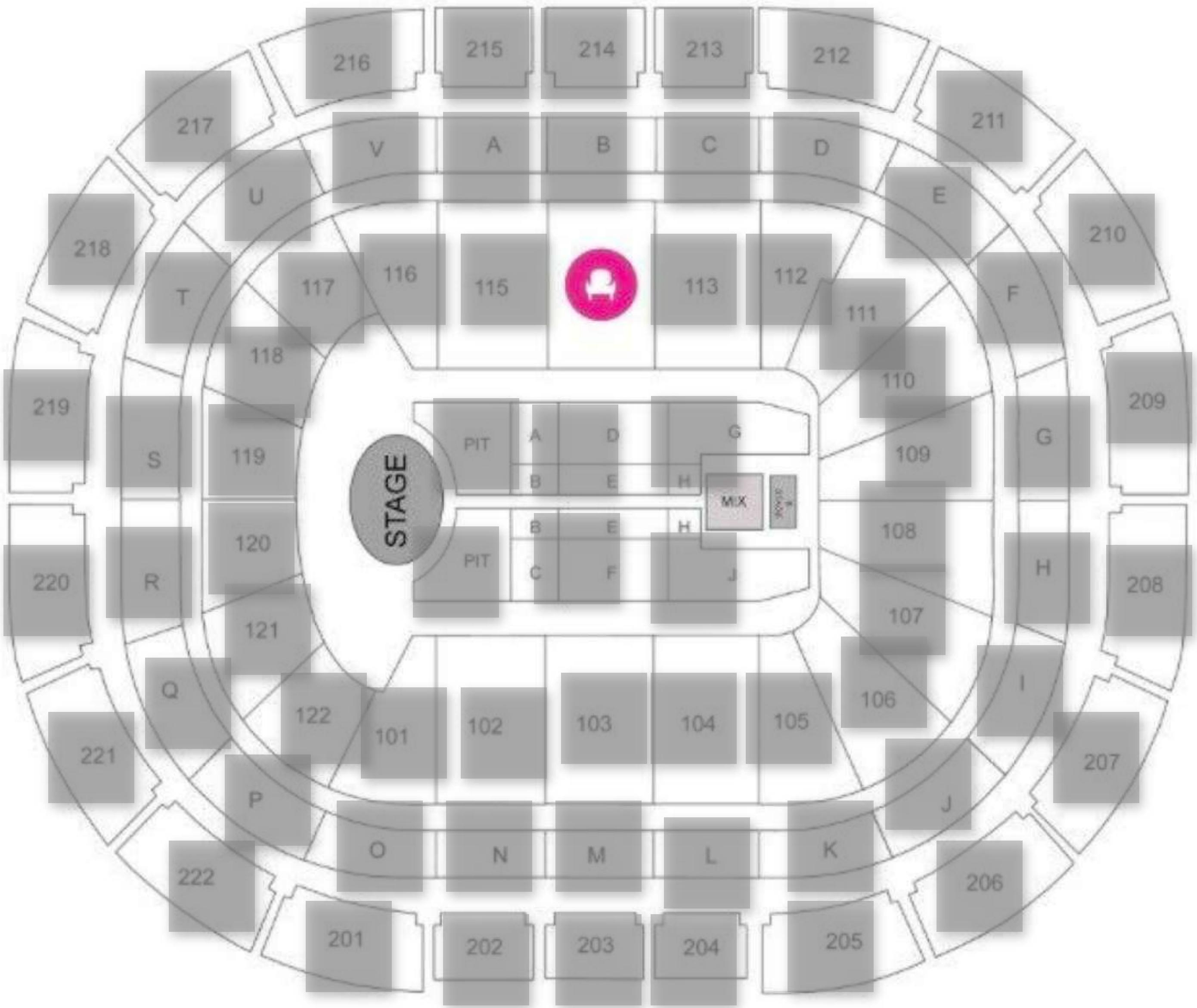
space-based



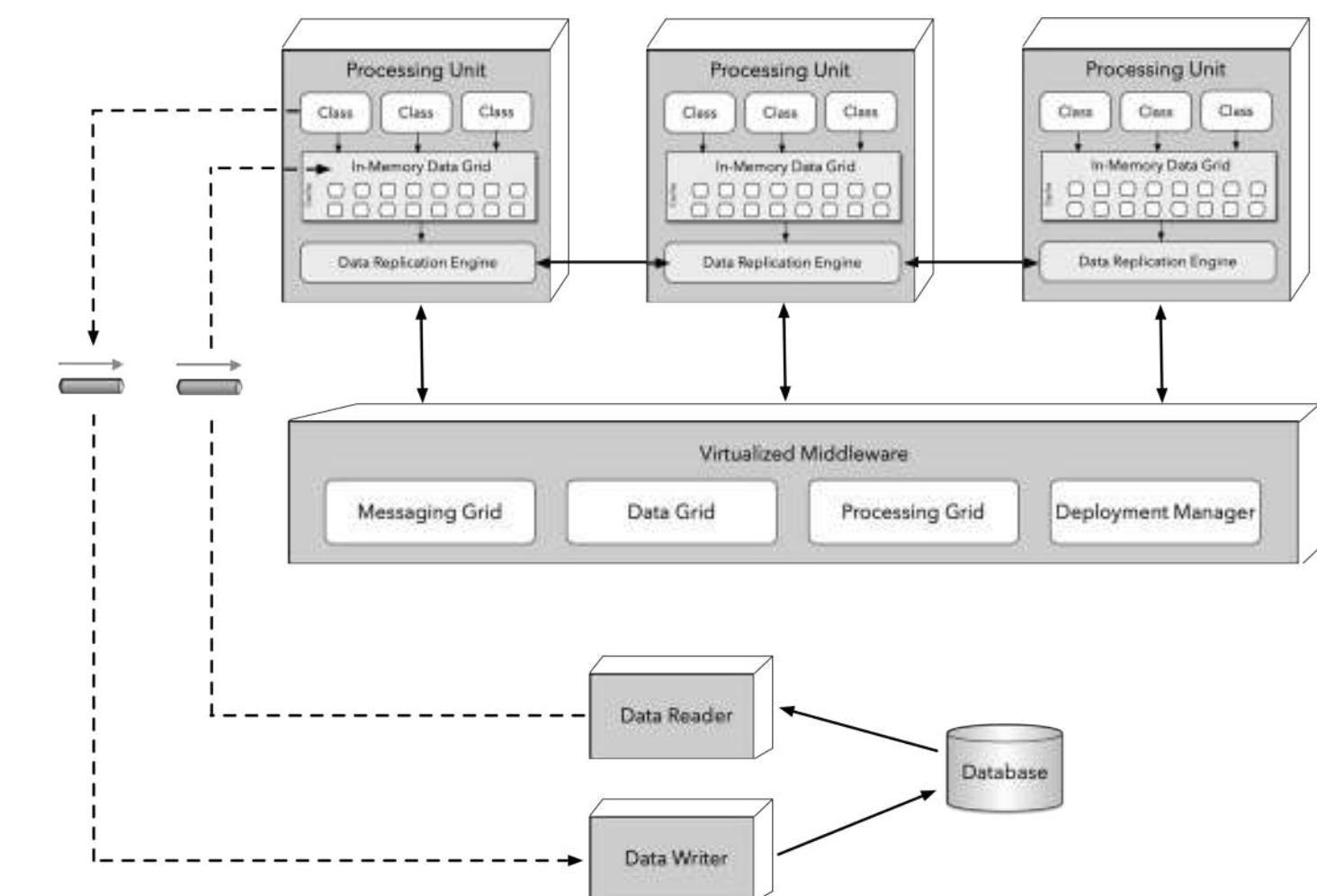
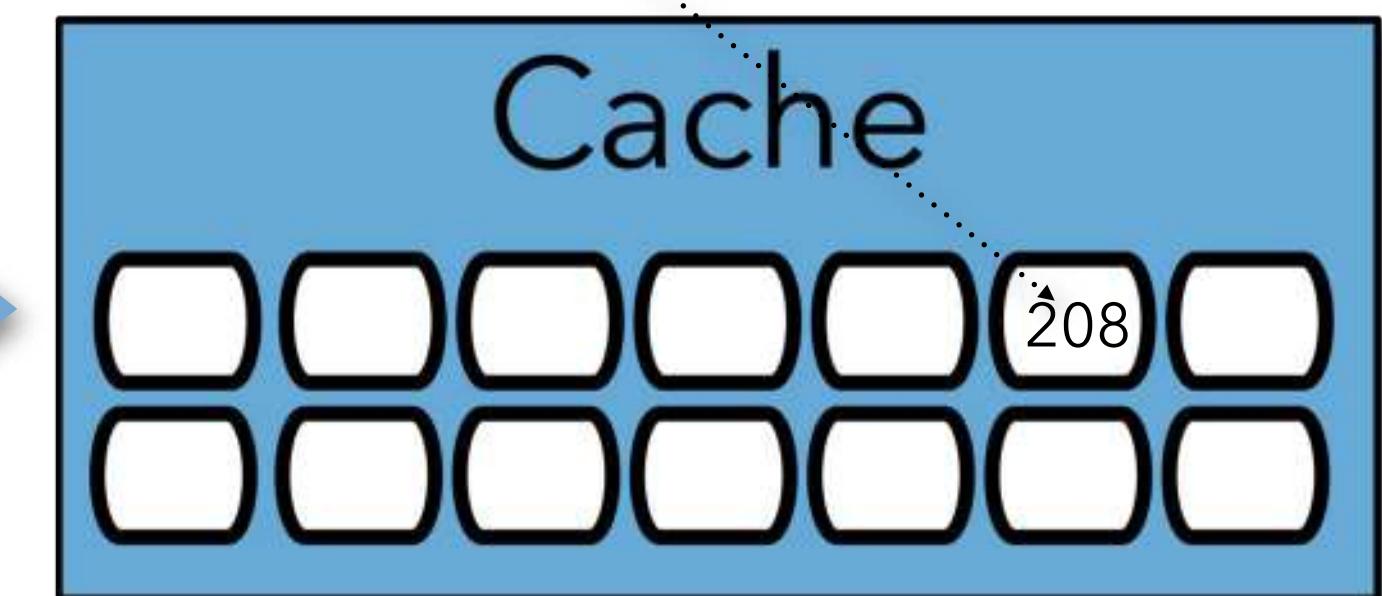
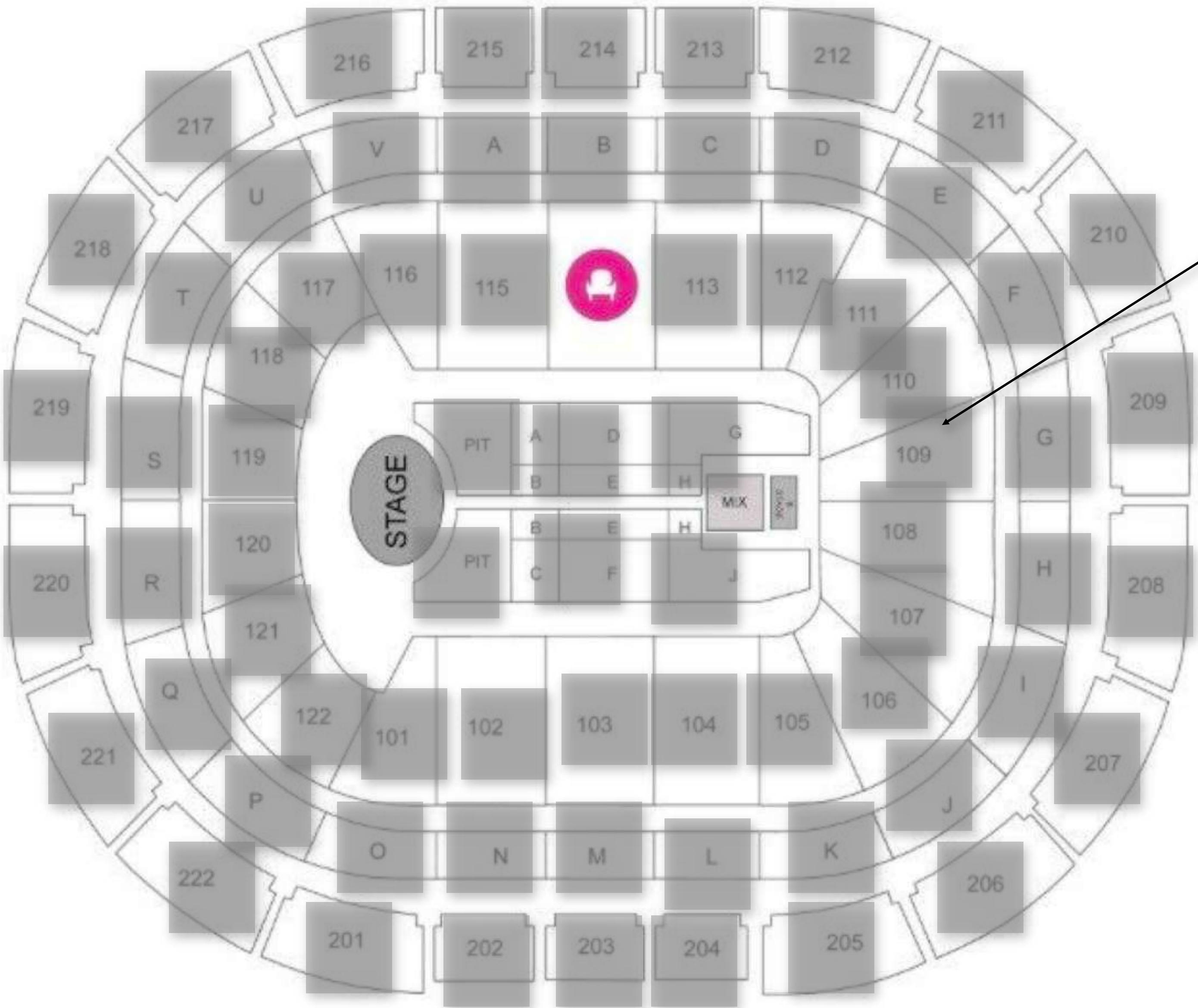
: transactions



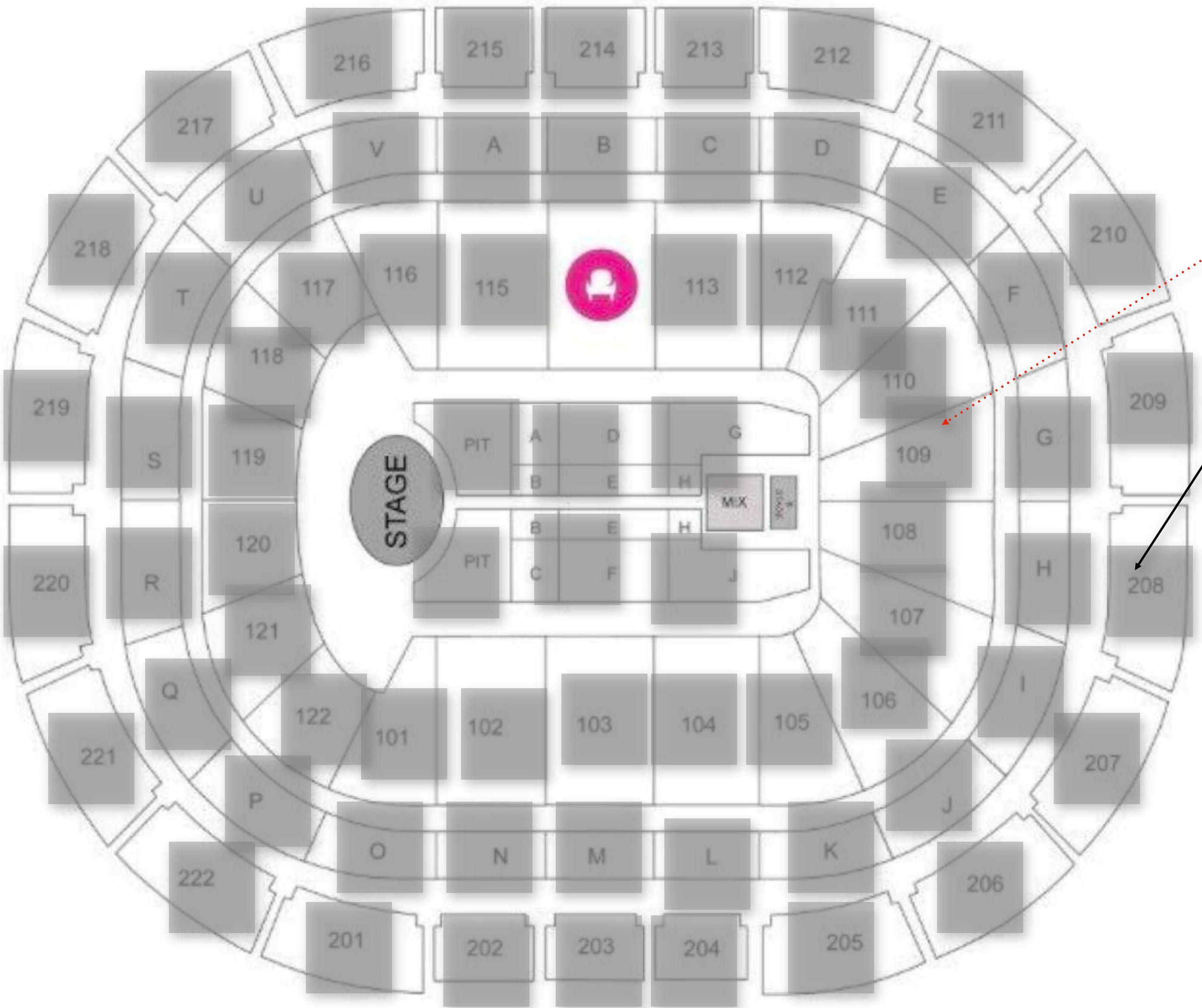
space-based : preview



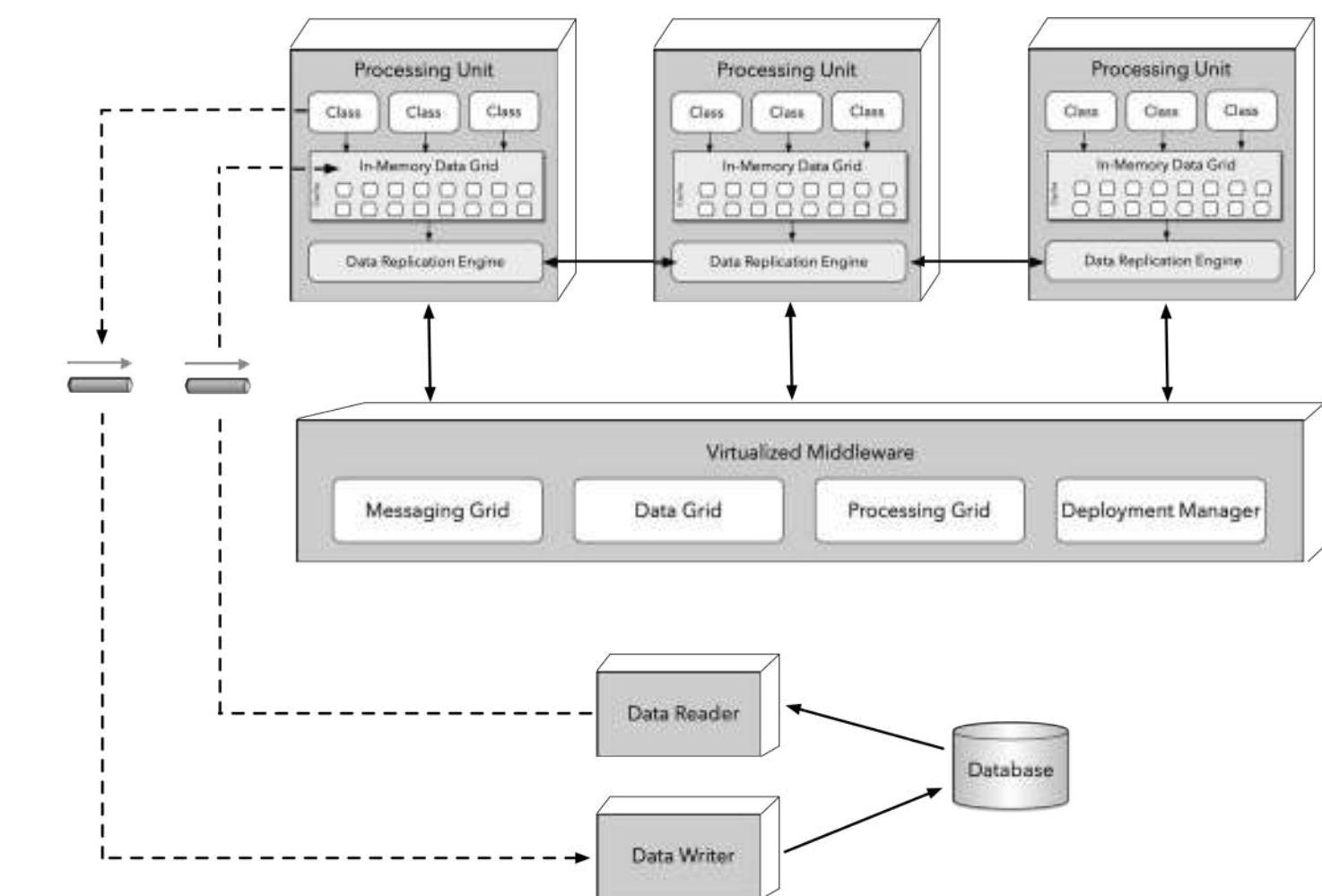
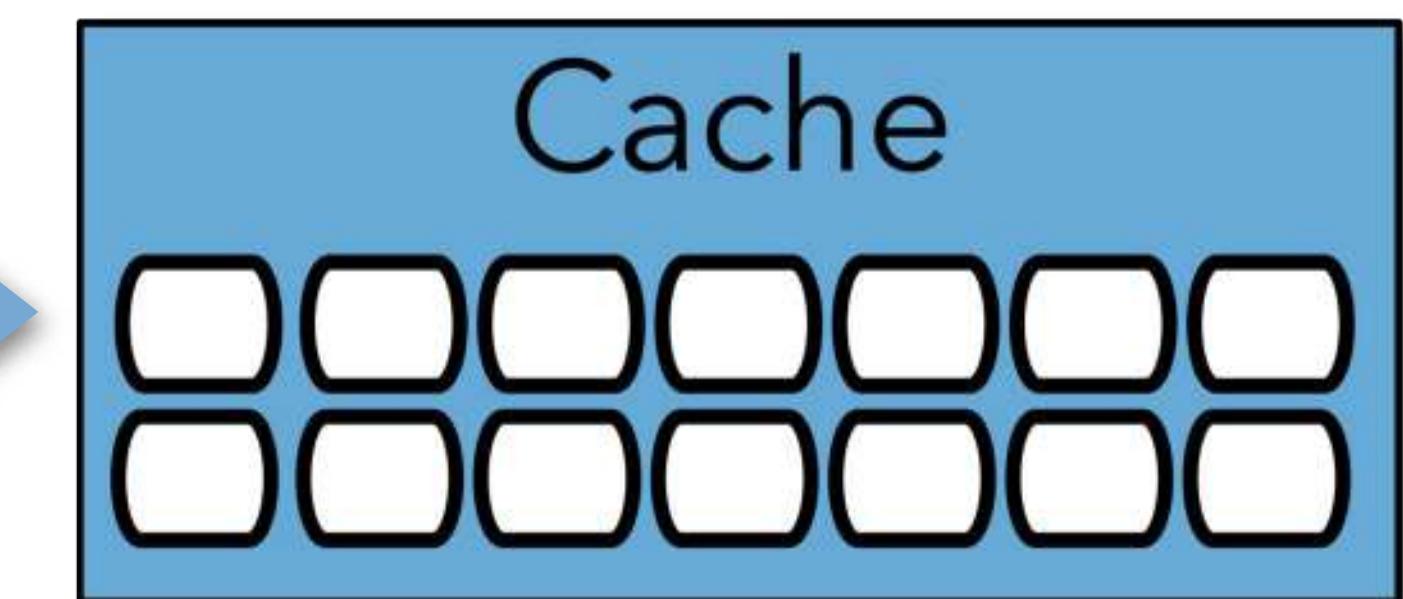
space-based : preview



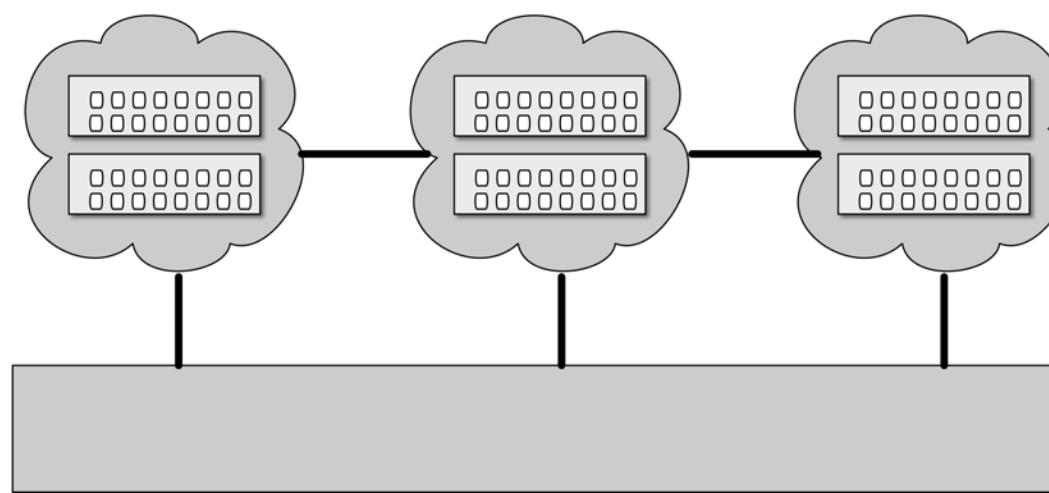
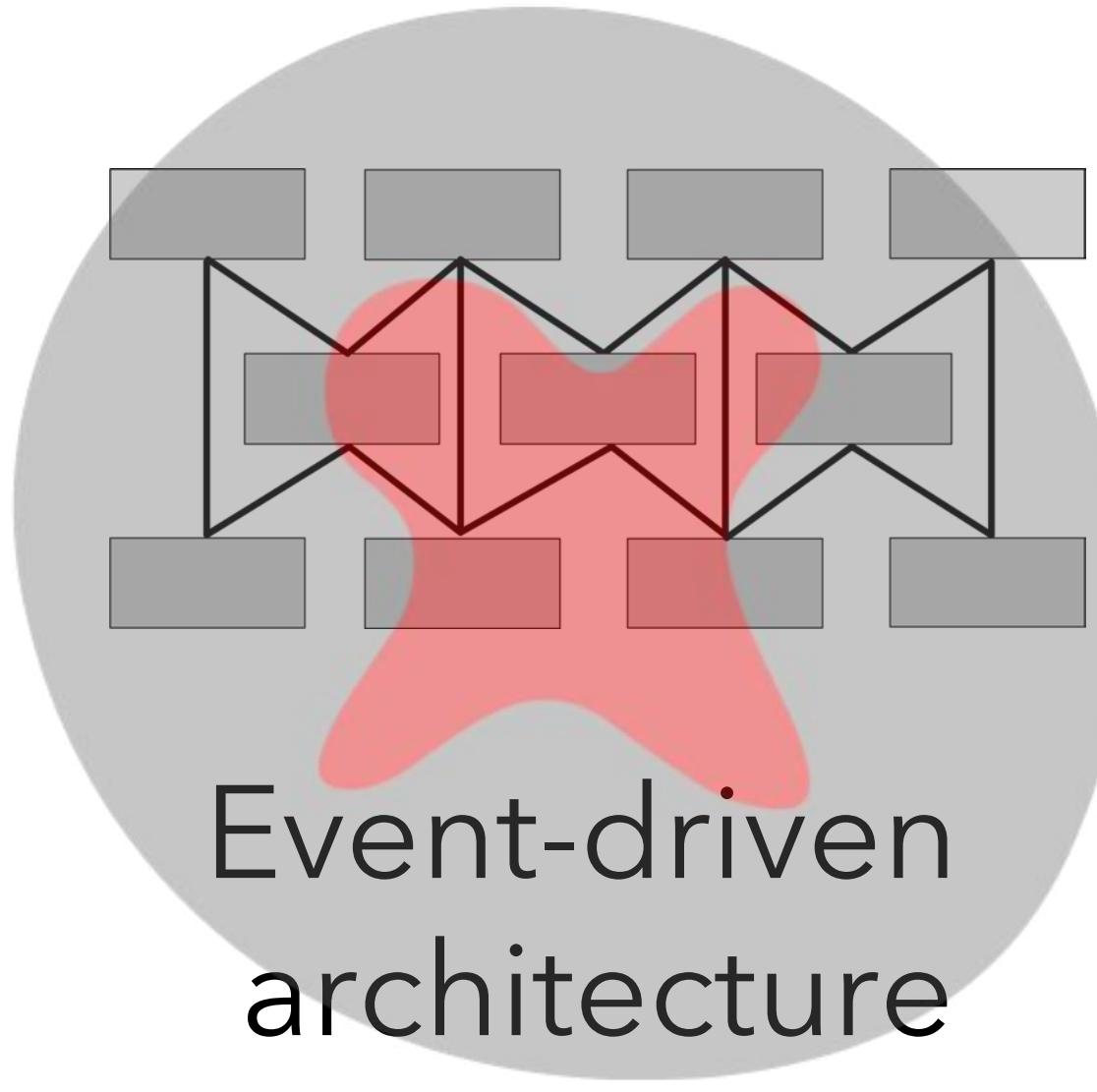
space-based : preview



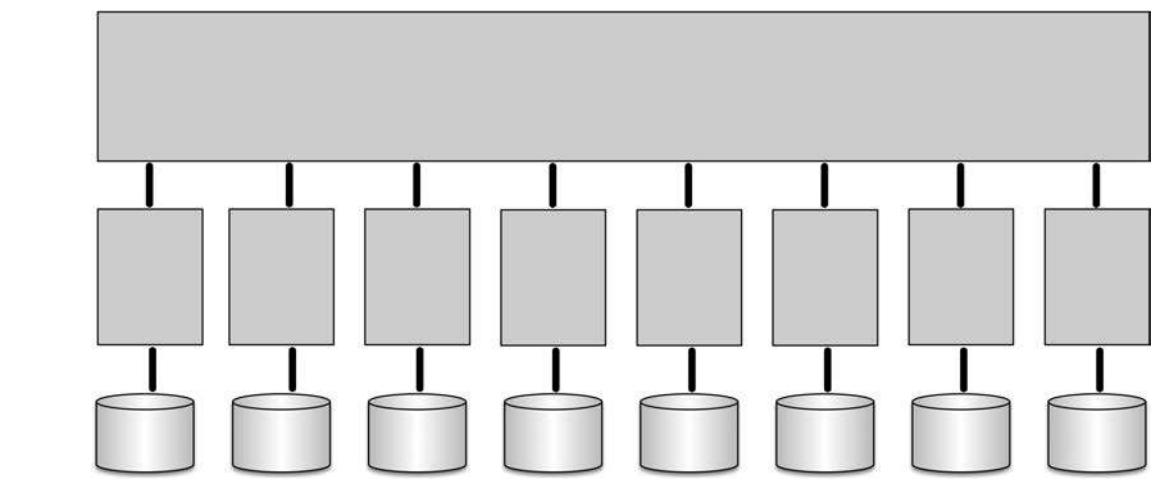
domain sharding



scalability ∪ elasticity

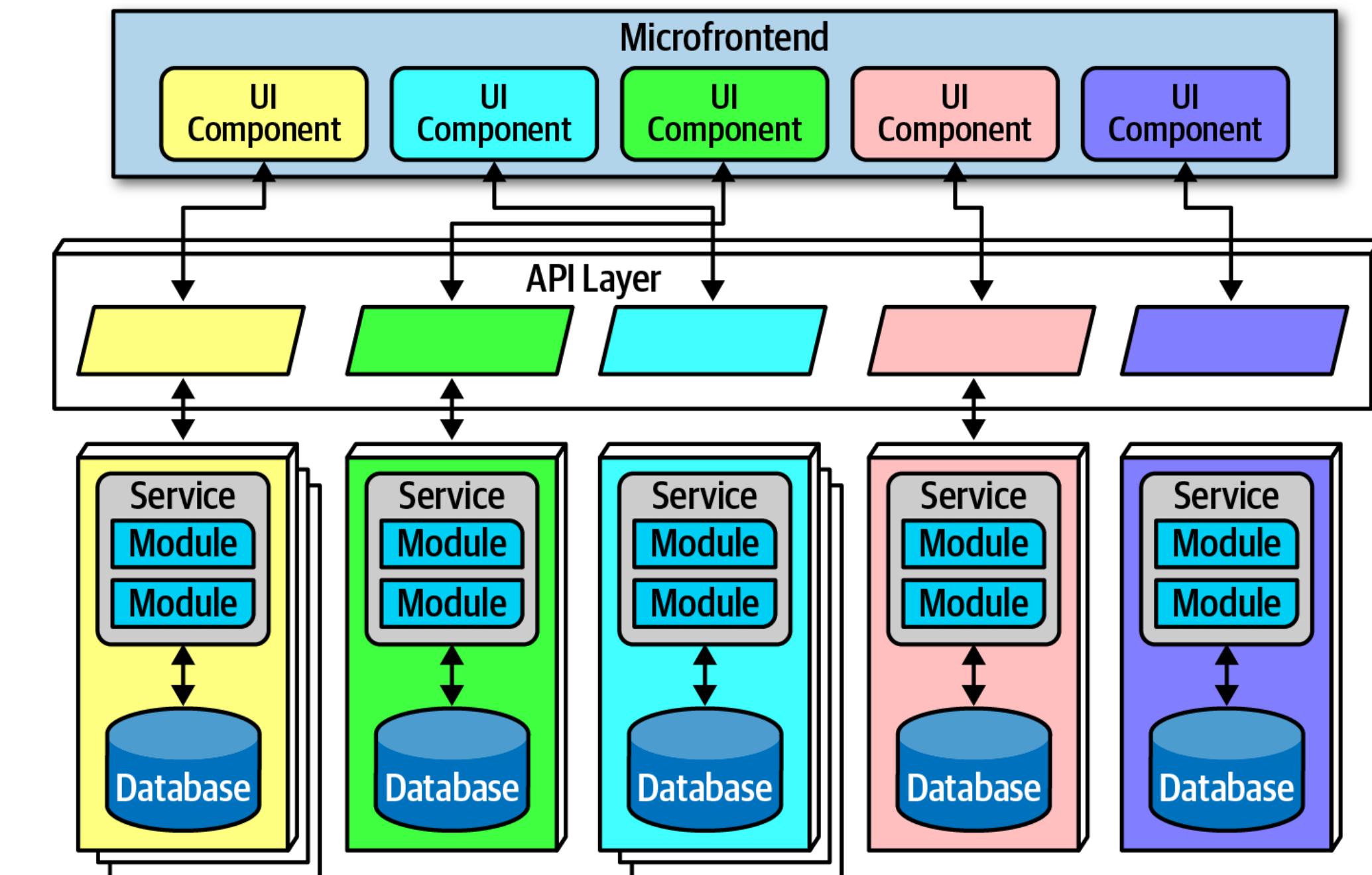
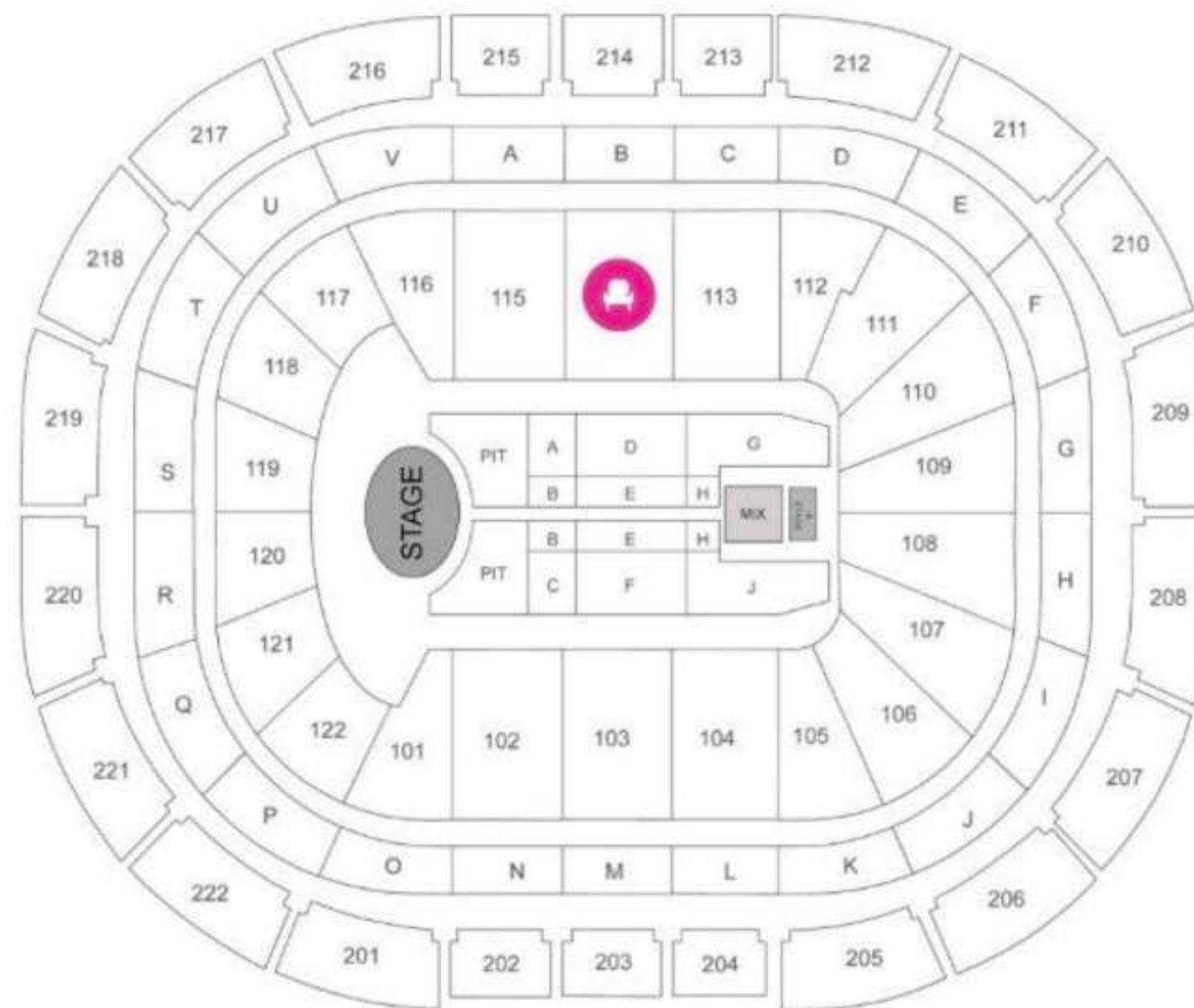


Space-based
architecture

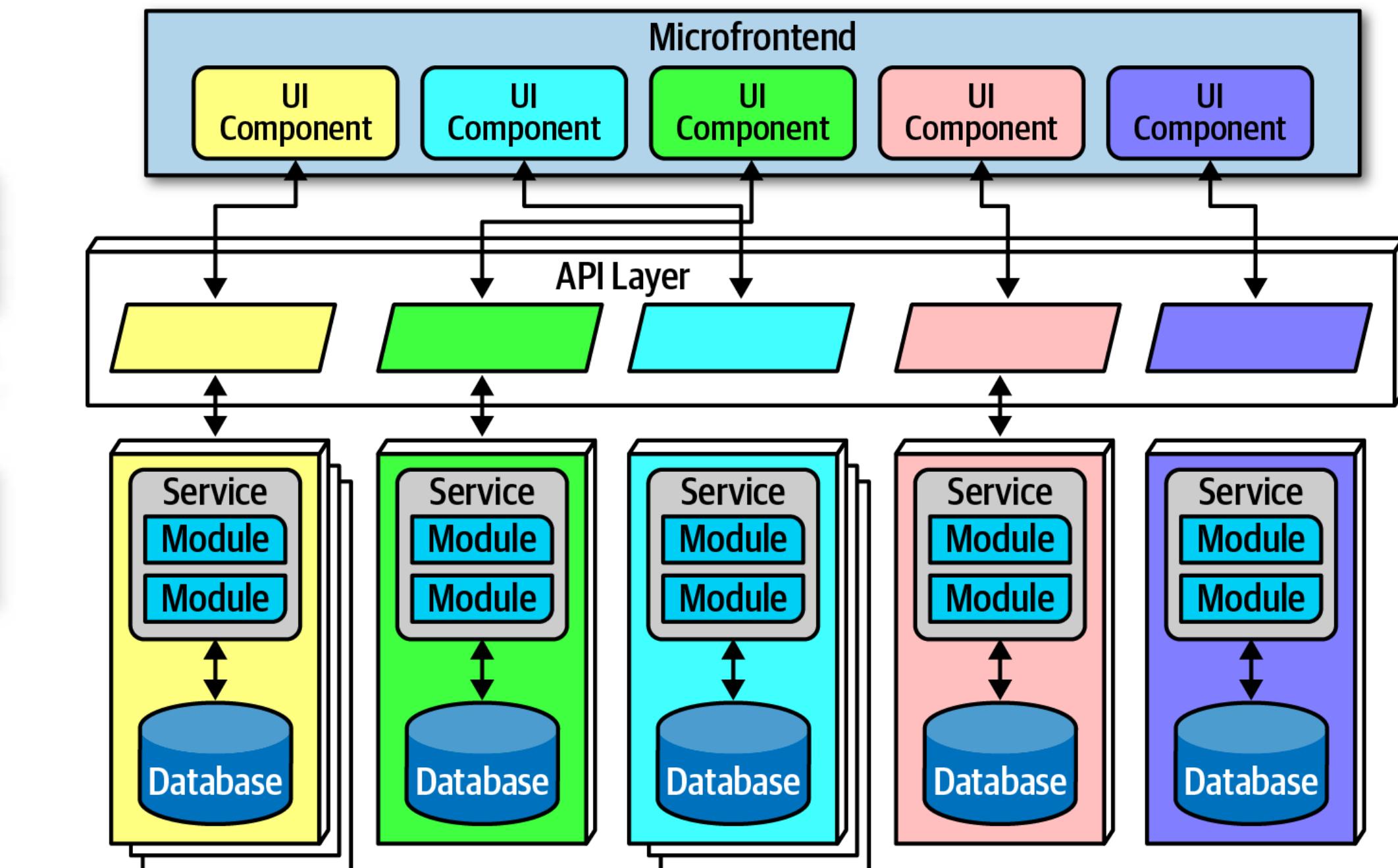
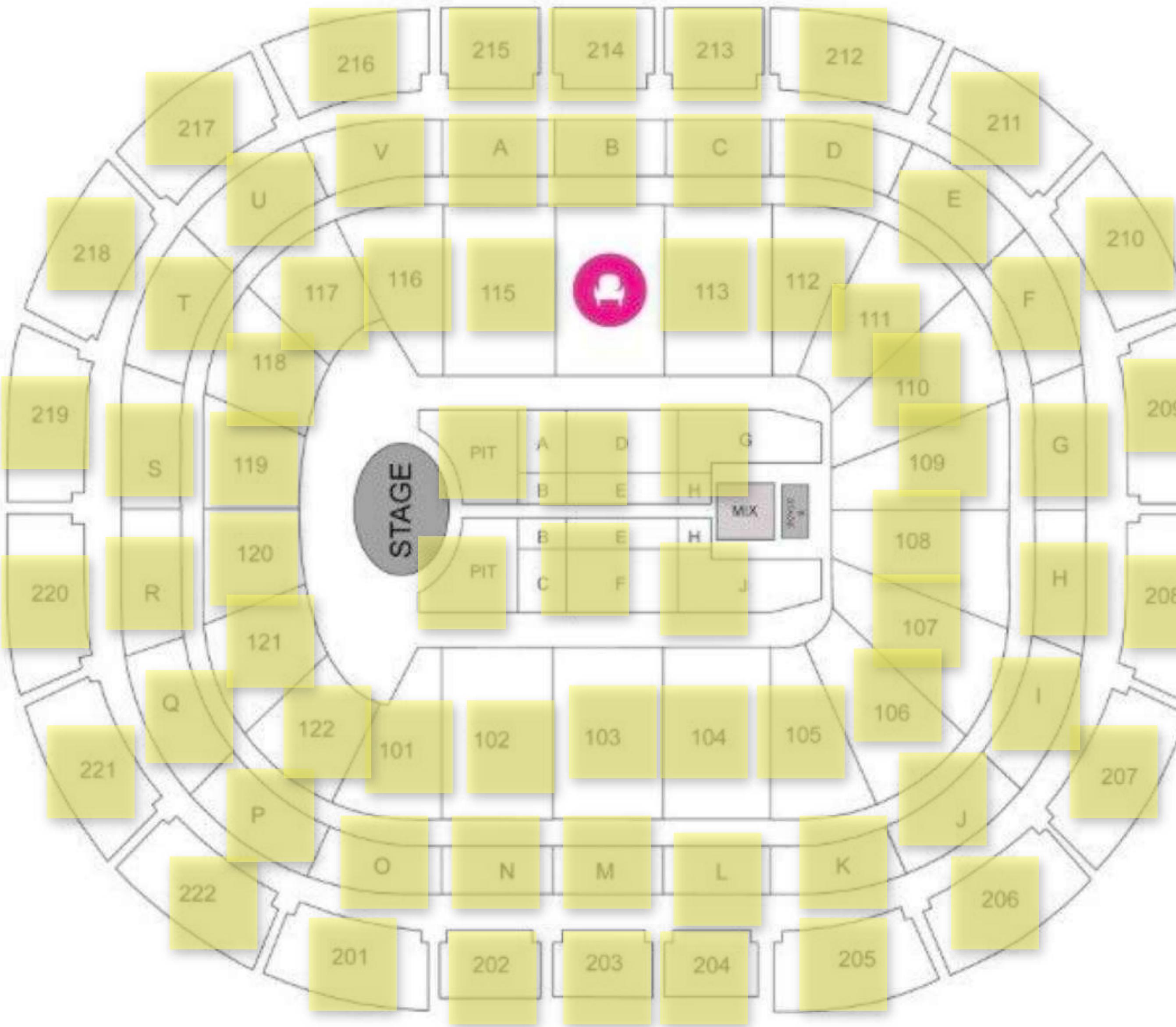


Microservices
architecture

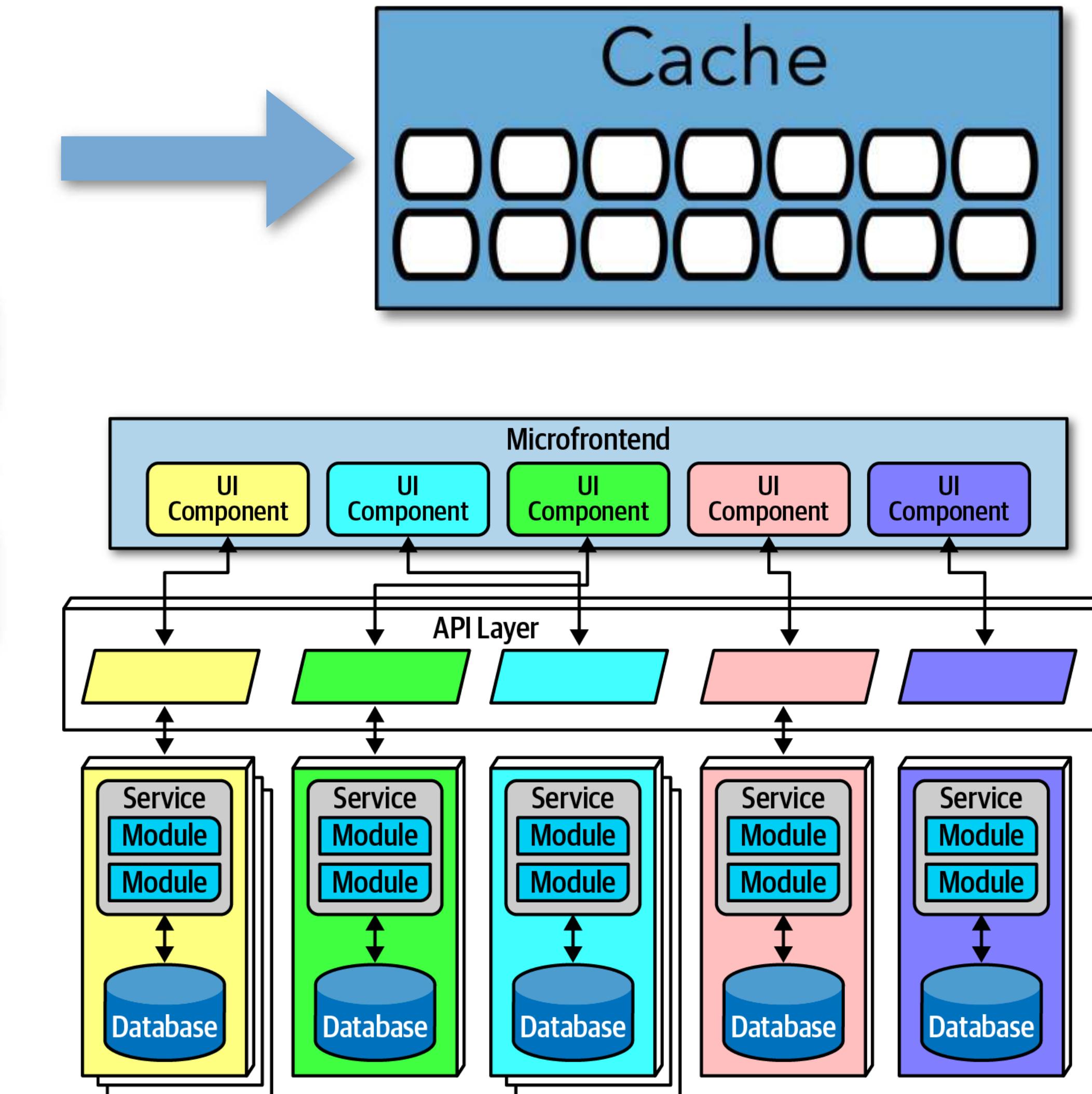
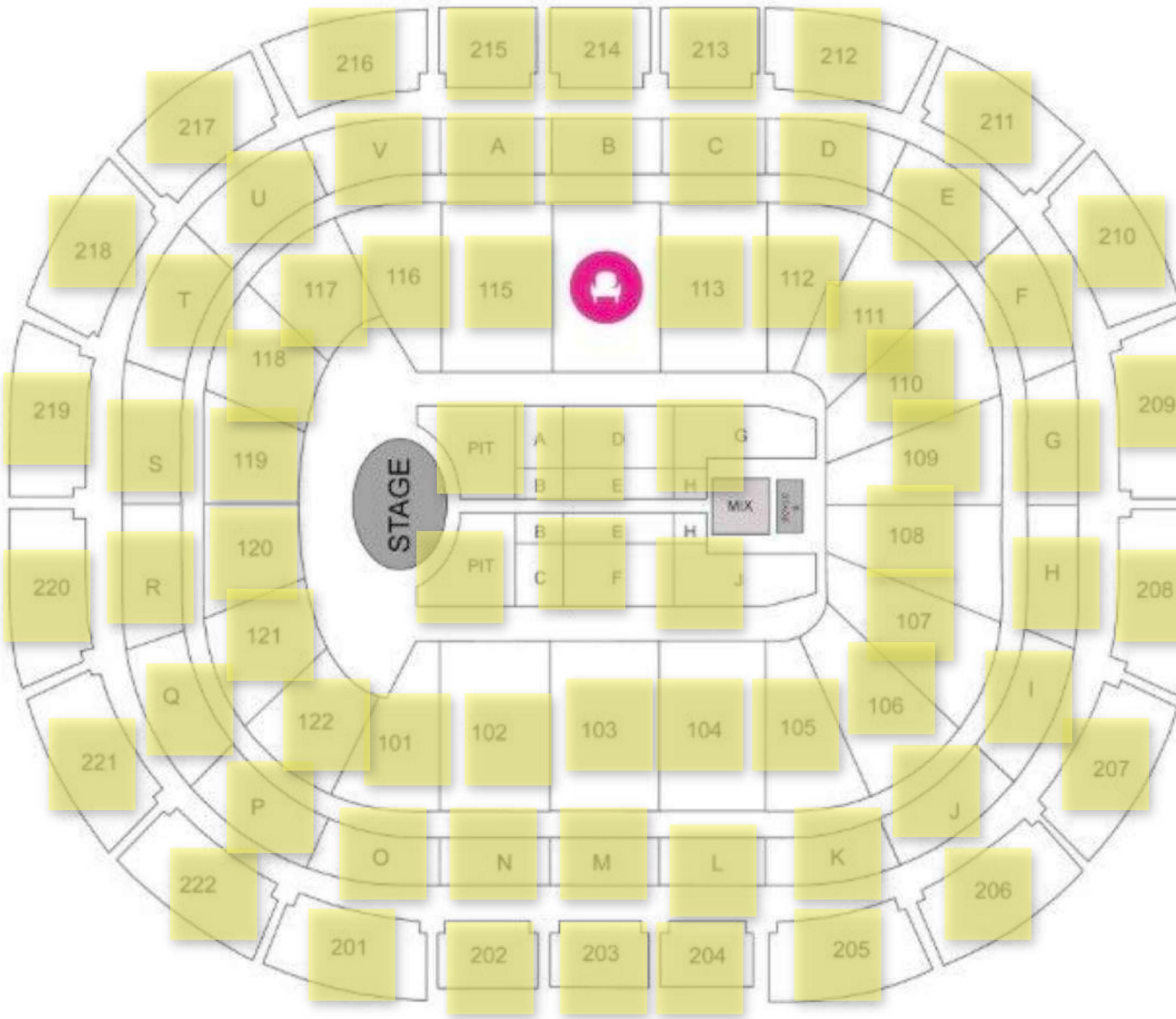
microservices: transactions



microservices: transactions



microservices: preview



tradeoffs

space-based *OR* microservices

- * more optimized for the problem space
- * higher complexity
- * higher incidental coupling
- * more domain aligned
- * current trend in architecture

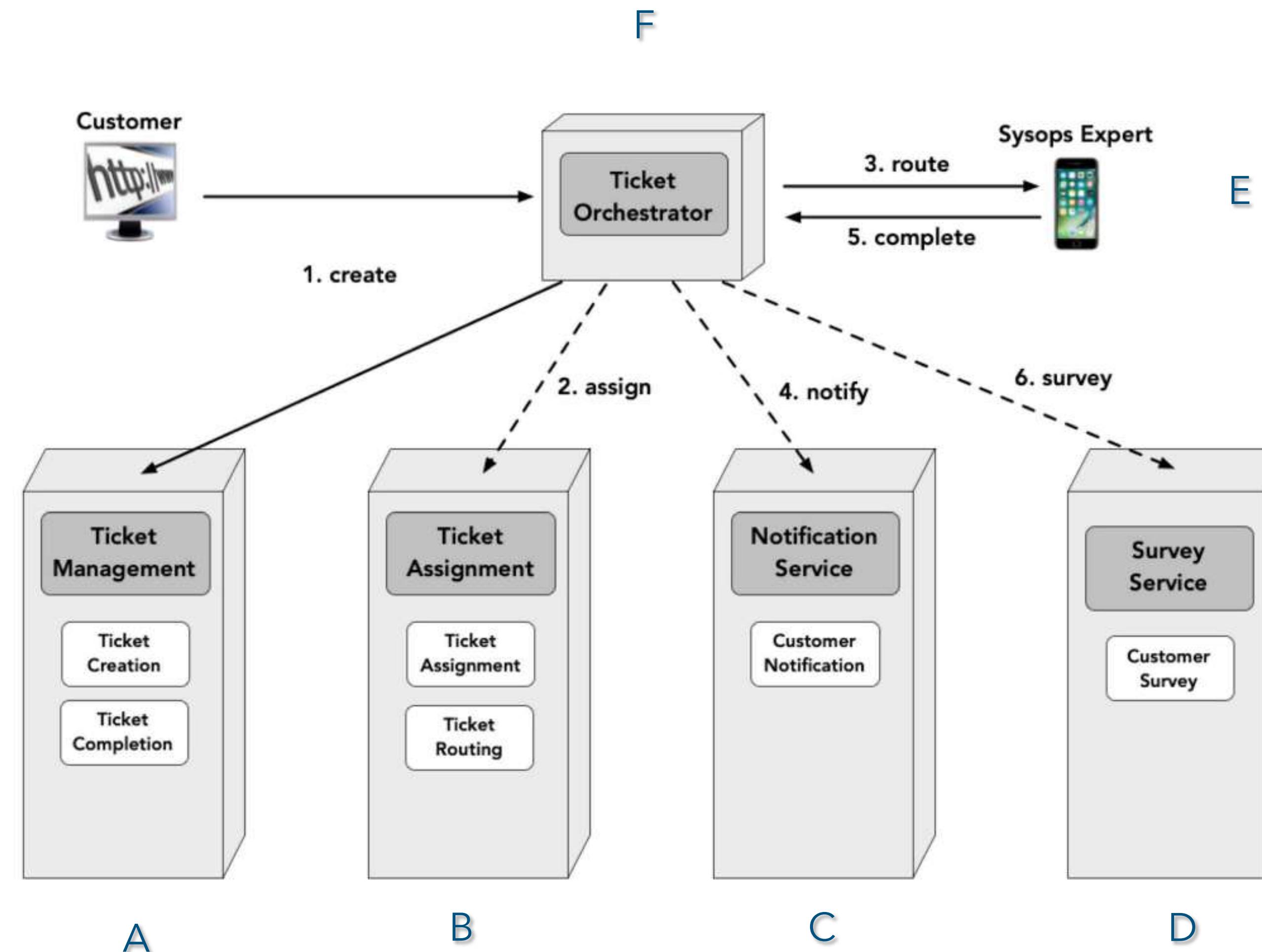
Kata Exercise - Handling Elasticity

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?



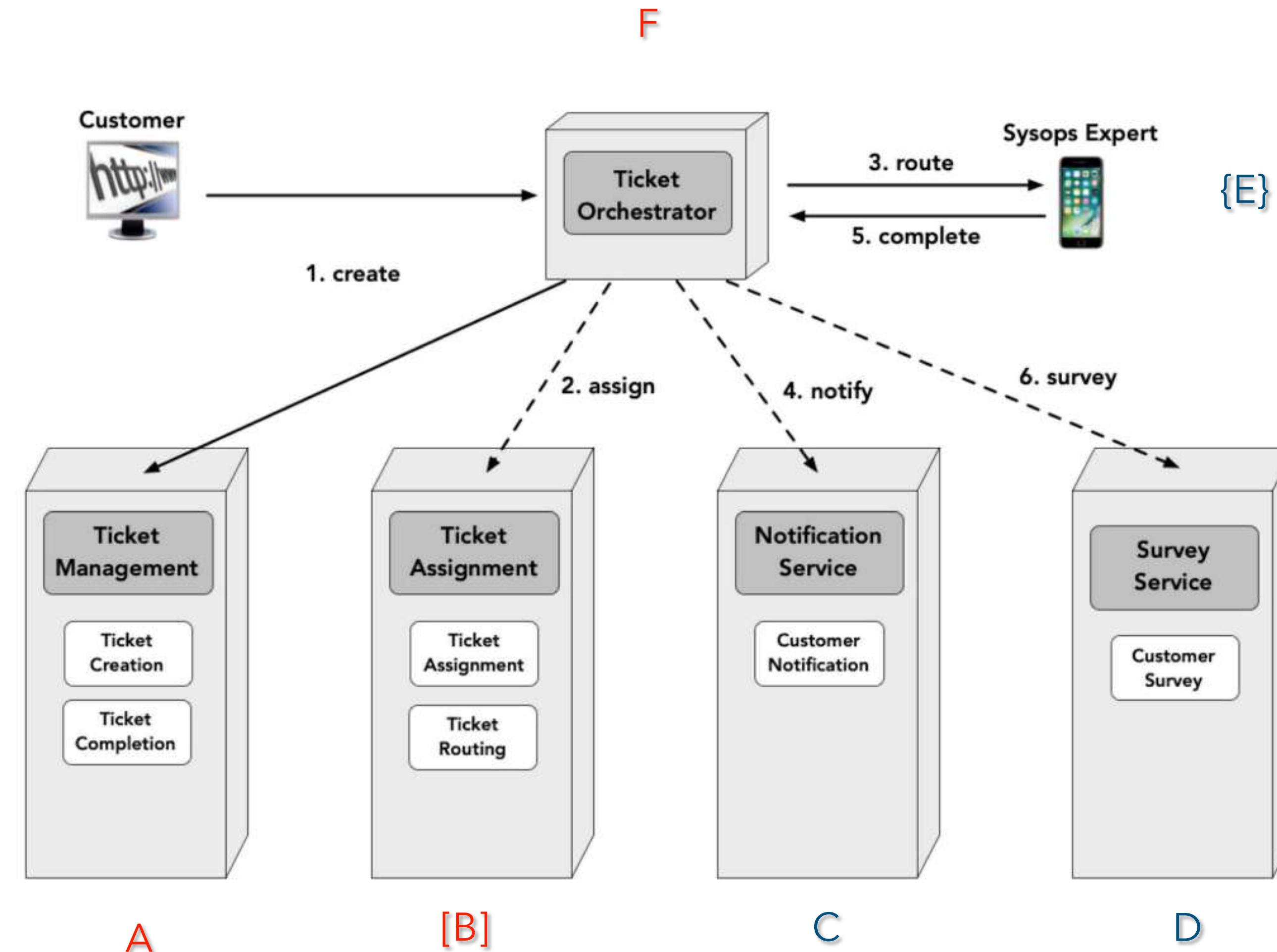
Scenario Exercise - Handling Elasticity

Which parts of the architecture should the architect add caching, scale instances, or do domain sharding?



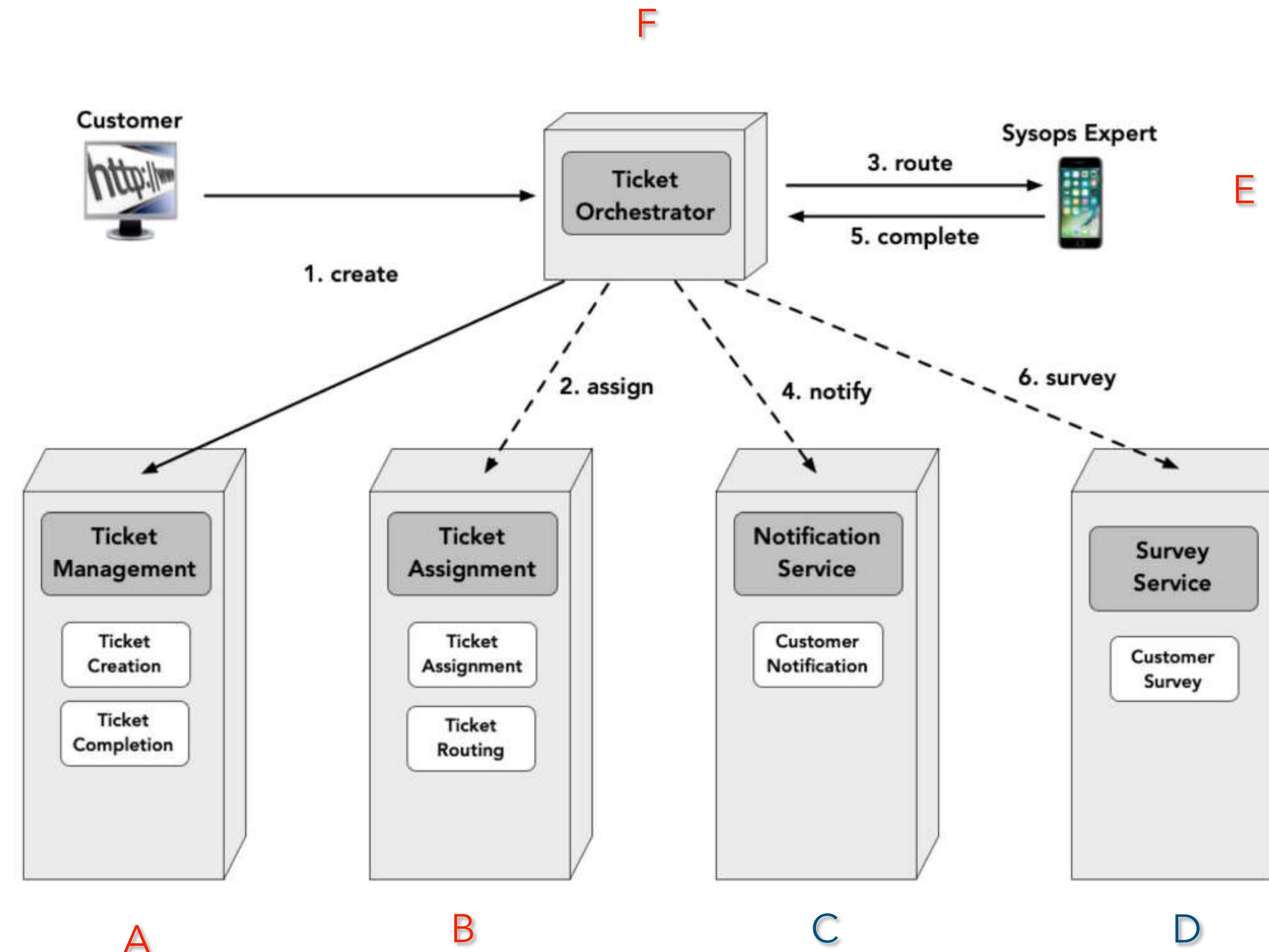
Scenario Exercise - Handling Elasticity

Which parts of the architecture should the architect add caching, scale instances, or do domain sharding?



Scenario Exercise - Handling Elasticity

Which parts of the architecture should the architect add caching, scale instances, or do domain sharding?



summary

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?

Architecture: The Hard Parts Scenarios

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?

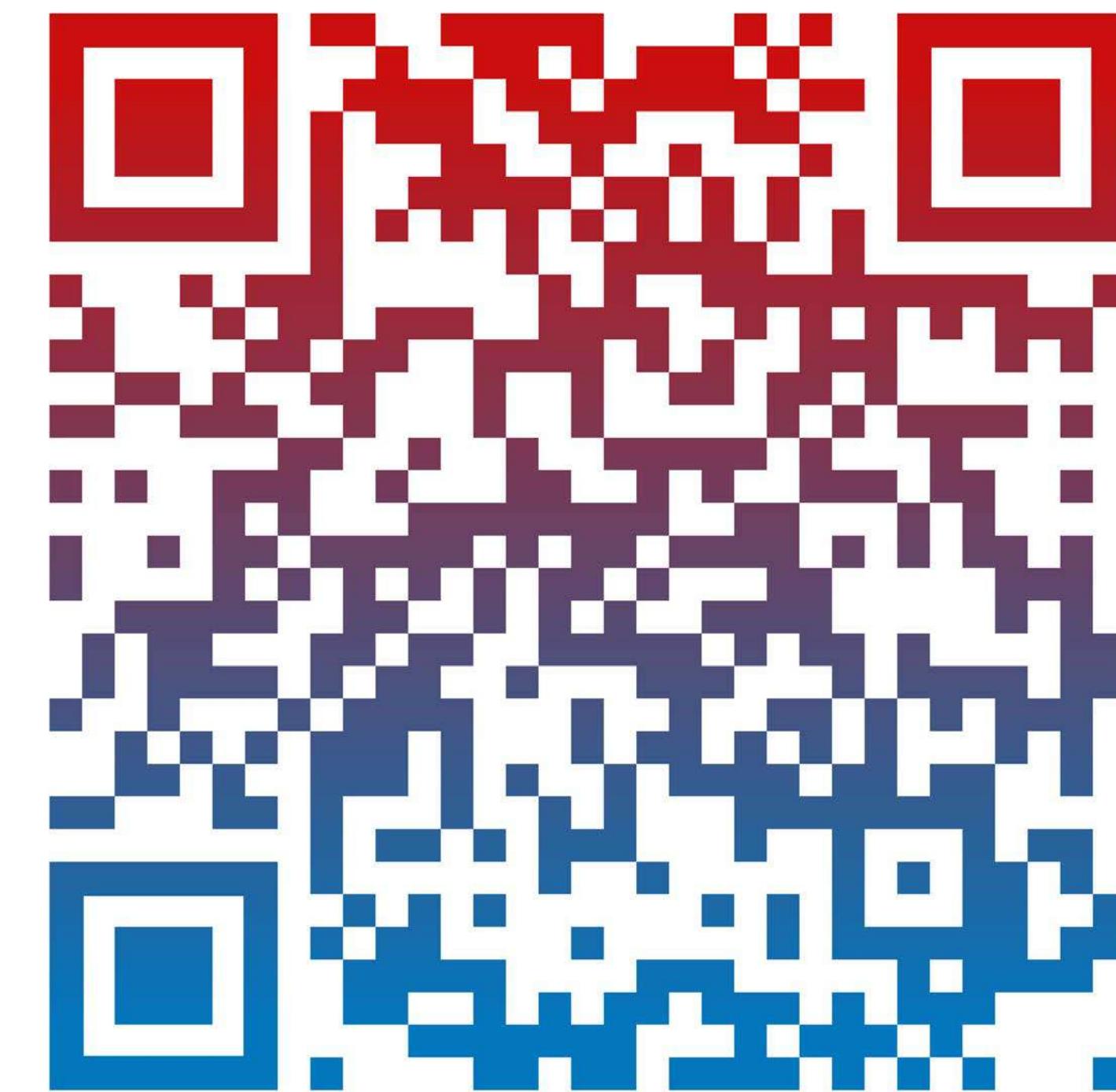
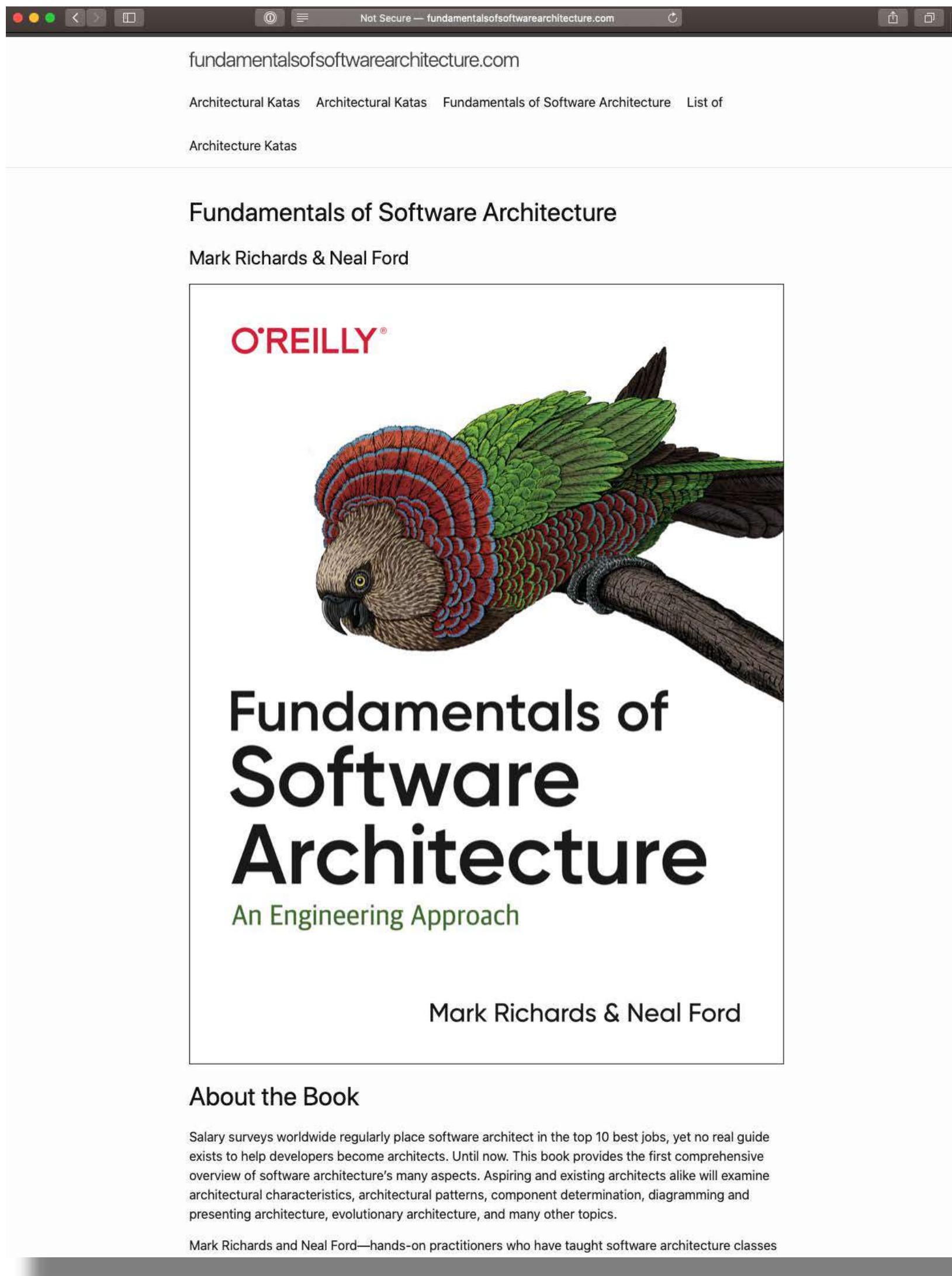
How do I handle workflow and transactions in a distributed architecture?

Architecture: The Hard Parts

A P



Fundamentals of Software Architecture



fundamentalsofsoftwarearchitecture.com

DeveloperToArchitect.com

The screenshot shows the homepage of DeveloperToArchitect.com. At the top, there is a navigation bar with links: HOME, TRAINING, LESSONS, ARTICLES, BOOKS, VIDEOS, FORUM, UPCOMING EVENTS, and ABOUT ME. Below the navigation is a large banner featuring a blue 3D pipe system on a grid background. The banner contains the title "Developer to Architect", a subtitle "Training and resources for the journey from software developer to software architect", and the author's name "Mark Richards, Software Architect and Founder". The main content area has two columns. The left column features a portrait of Mark Richards and a quote about the journey from developer to architect. It also mentions "Software Developer To Software Architect". The right column has sections for "Contact Me" (with information on how to reach Mark) and "Public Live Virtual Training" (with a thumbnail image of a video call interface).

HOME TRAINING LESSONS ARTICLES BOOKS VIDEOS FORUM UPCOMING EVENTS ABOUT ME

Developer to Architect

Training and resources for the journey from software developer to software architect

Mark Richards, Software Architect and Founder

Software 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 and goal 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 created this website to provide developers with resources and guidance in the long and difficult journey from software developer to software architect. In here you'll find helpful lessons, articles, books, videos, source code, and training classes I teach.



Software Architecture Monday is a free bi-weekly video lesson series on some aspect of software architecture. These 10 minute YouTube videos contain various aspects of application, integration, and enterprise architecture.

Contact Me

To contact me regarding any public and private software architecture training classes I offer, you can reach me (Mark Richards) directly at info@developertoarchitect.com



Public Live Virtual Training

I'm excited to announce that during our current social situation I am conducting live, hands-on virtual workshops. See my [Training](#) page and [Upcoming Events](#) page for a complete listing of classes, details, dates, and how to register.

AP



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