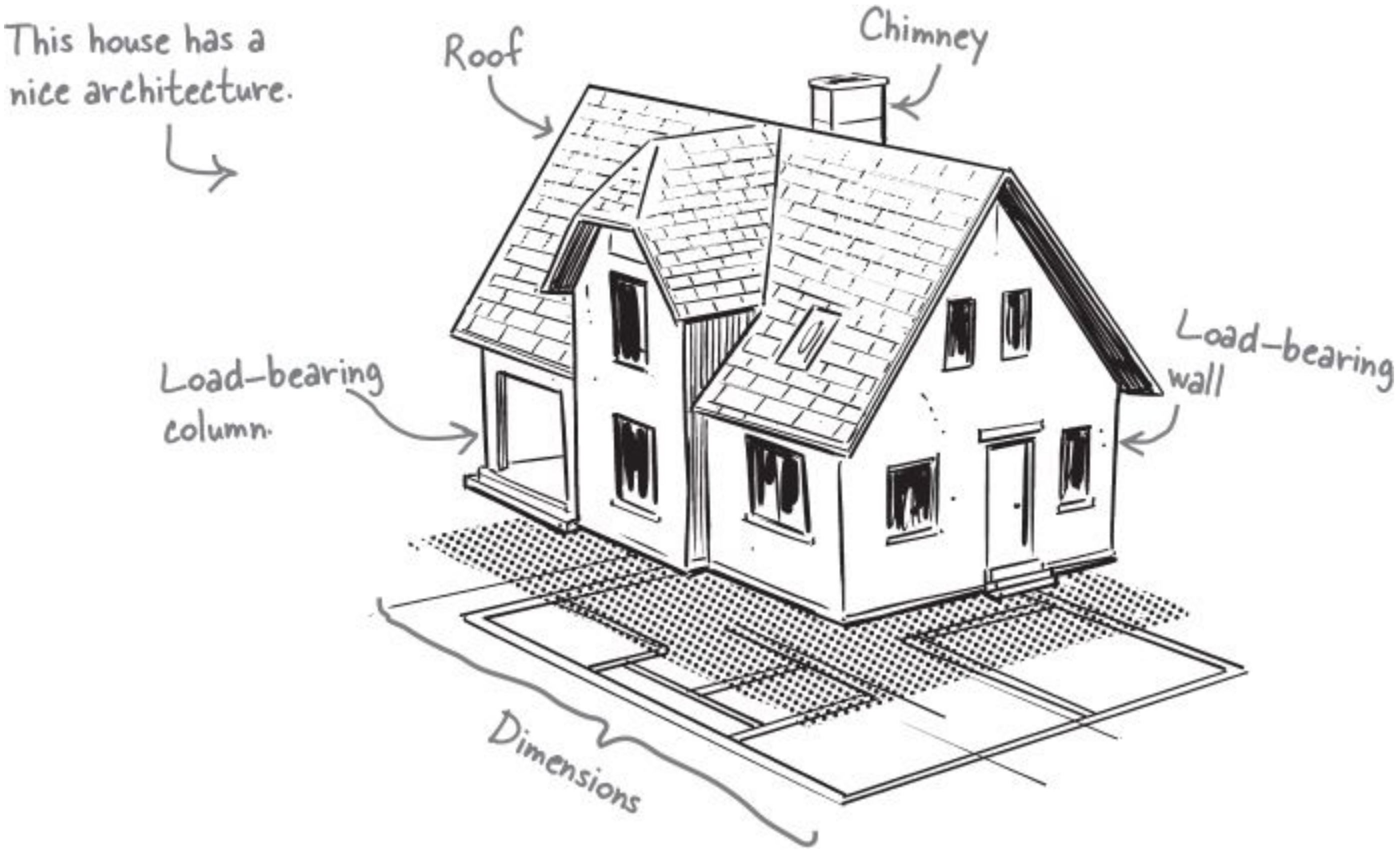
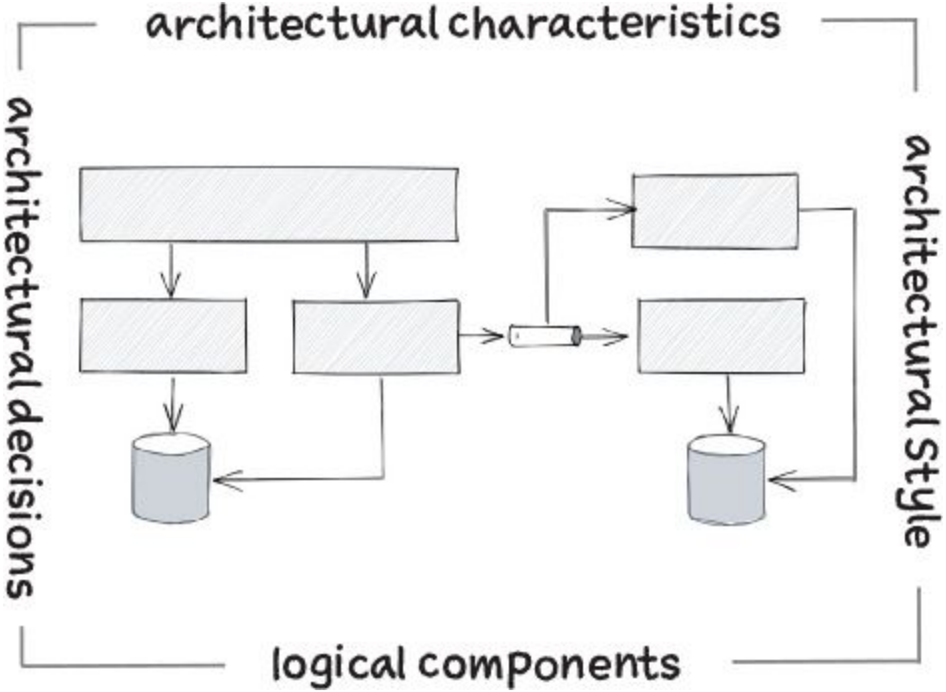
Livro: head first software architecture



This house has a nice architecture.

Not only is this house ugly, it's not very functional either.



architectural characteristics
architectural decisions
architectural style
logical components

Architectural characteristics

This dimension describes what aspects of the system the architecture needs to support—things like scalability, testability, availability, and so on.

Architectural decisions

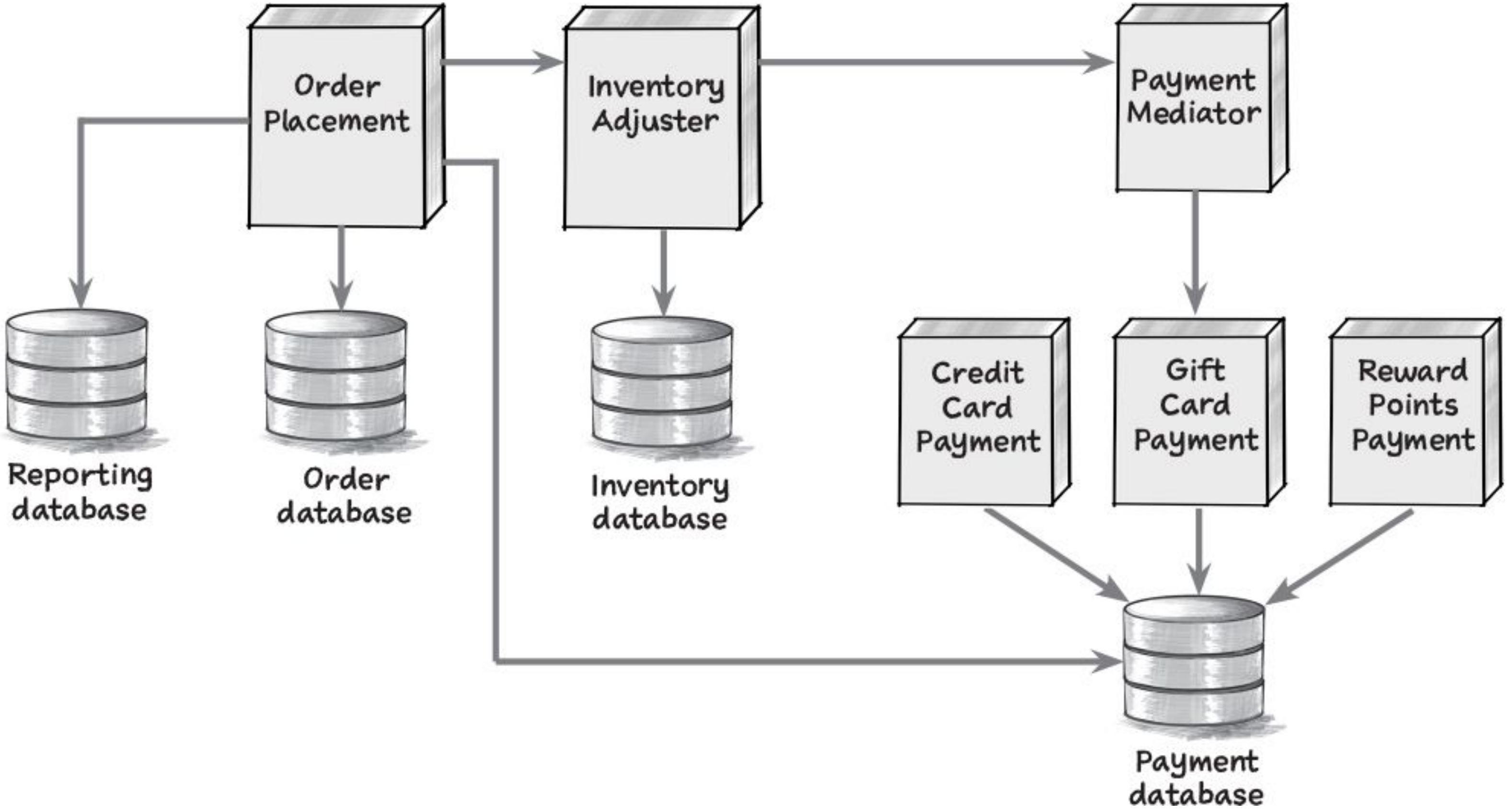
This dimension includes important decisions that have long-term or significant implications for the system—for example, the kind of database it uses, the number of services it has, and how those services communicate with each other.

Logical components

This dimension describes the building blocks of the system's functionality and how they interact with each other. For example, an e-commerce system might have components for inventory management, payment processing, and so on.

Architectural style

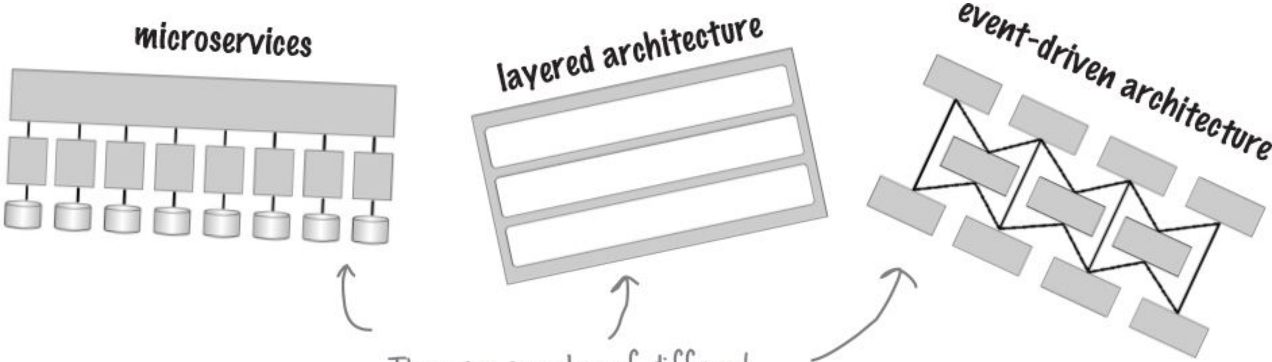
This dimension defines the overall physical shape and structure of a software system in the same way a building plan defines the overall shape and structure of your home.



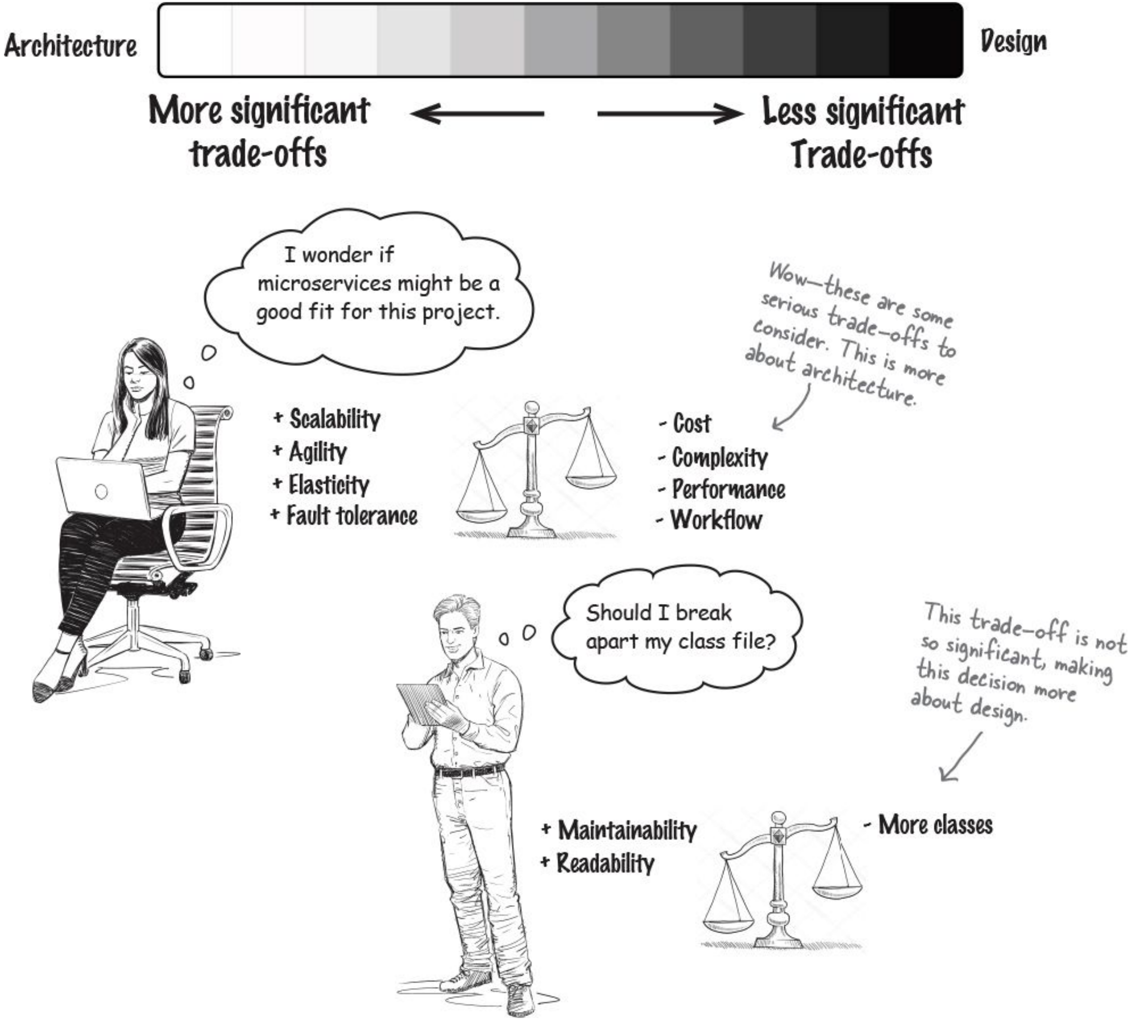
Here's a hint—do you have questions about why certain things are done the way they are?

What style home do you live in?

Each region of the world has its own set of home styles—check 'em out at https://en.wikipedia.org/wiki/List_of_house_styles



There are a number of different architectural styles, but fortunately not as many as there are house styles.



Architecture

Design

More significant trade-offs

Less significant Trade-offs

I wonder if microservices might be a good fit for this project.

+ Scalability
+ Agility
+ Elasticity
+ Fault tolerance

- Cost
- Complexity
- Performance
- Workflow

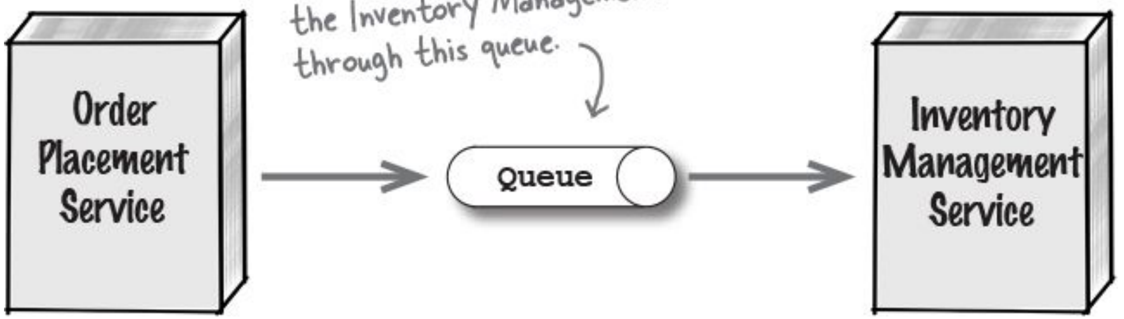
Wow—these are some serious trade-offs to consider. This is more about architecture.

Should I break apart my class file?

+ Maintainability
+ Readability

- More classes

This trade-off is not so significant, making this decision more about design.



Inventory adjustments are sent to the Inventory Management service through this queue.

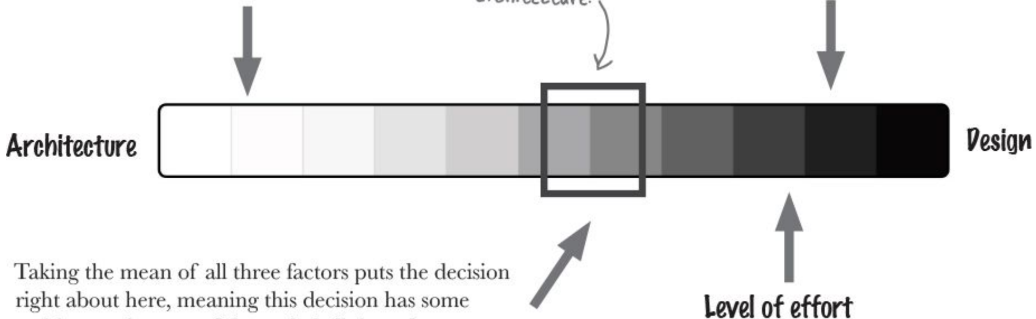
Significance of trade-offs

Using a queue will increase responsiveness when placing an order, but inventory may not be updated in a timely manner, likely creating back-order conditions. These are pretty significant trade-offs.

The significant trade-offs push this decision closer to architecture.

Strategic or tactical

Not many people need to be involved in this decision, and it doesn't involve long-term planning, so it's more tactical.

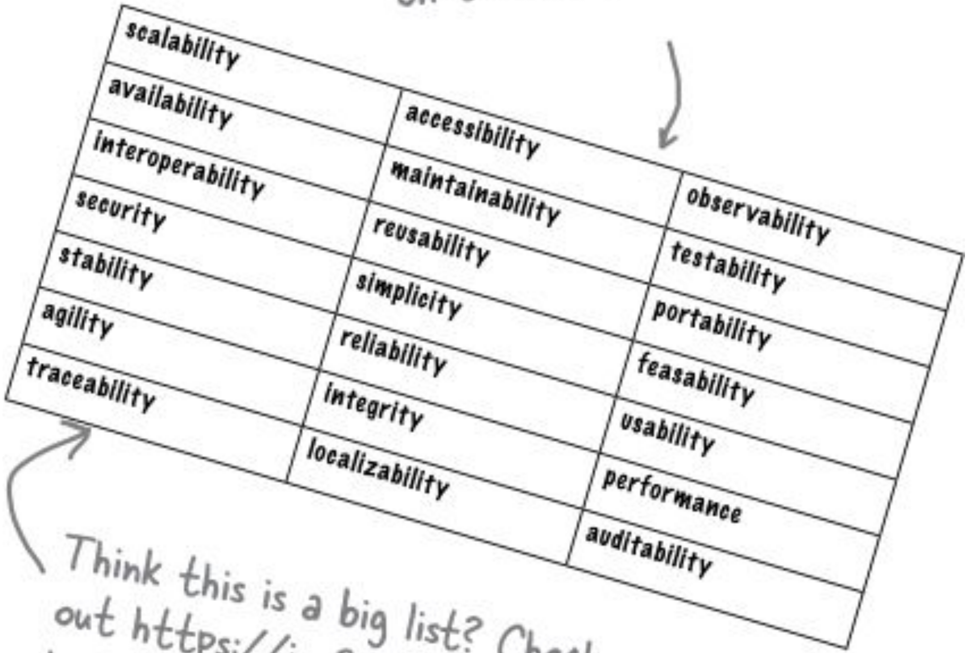


Taking the mean of all three factors puts the decision right about here, meaning this decision has some architectural aspects. We needed all three factors to determine whether this decision was more about architecture or design.

Level of effort

It doesn't take a whole lot of effort to send a message to another service. This is pretty standard stuff.

Don't worry—we define many of the more mysterious terms on the next few pages.



- scalability
- availability
- interoperability
- security
- stability
- agility
- traceability
- accessibility
- maintainability
- reusability
- simplicity
- reliability
- integrity
- localizability
- observability
- testability
- portability
- feasability
- usability
- performance
- auditability

Think this is a big list? Check out https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

modularity

The degree to which the software is composed of discrete components. Modularity affects how architects partition behavior and organize logical building blocks.

testability

How complete the system's testing is and how easy these tests are to run, including unit, functional, user-acceptance, exploratory, and other forms of testing.

agility

A composite architectural characteristic that encompasses testability, deployability, modularity, and a host of other architectural characteristics that facilitate and enable agile software development practices.

"Testability" refers to testing at development time (such as unit testing), rather than formal quality assurance..

Yep, it shows up twice. Many architectural characteristics cut across categories, as you'll see in the next few pages.

Agility is a composite architectural characteristic we'll discuss later in this chapter—stay tuned!

extensibility

How easy it is for developers to extend the system. This may encompass architectural structure, engineering practices, internal design, and governance.

Yes, we know this is a made-up word. That happens a lot in software architecture!

decouple-ability

Coupling describes how parts of the system are joined together; some architectures define how to *decouple* parts in specific ways to achieve certain benefits, which this architecture characteristic measures.

This is one of the many architectural characteristics that make up "agility.".

deployability

How easy and efficient it is to deploy the software system.

When business analysts and subject-matter experts say:

"Our business is constantly changing to meet new marketplace demands."

"Due to new regulatory requirements, it is imperative that we complete daily processing by the end of each day."

"Our plan is to engage heavily in mergers and acquisitions in the next three years."

Of course, no one would *ever* ask for this impossible combination...ahem.

"We have a very tight timeframe and a fixed scope and budget for this project."

Software architects translate:

Good modularity allows for faster change without rippling side effects.

- Agility
- Modularity
- Extensibility

More of an architect-the-person characteristic

We must perform well but also recover quickly in case of error.

- Performance
- Recoverability
- Scalability

"The ability to update your resume..." Many people would rather not work in a place undergoing constant mergers.

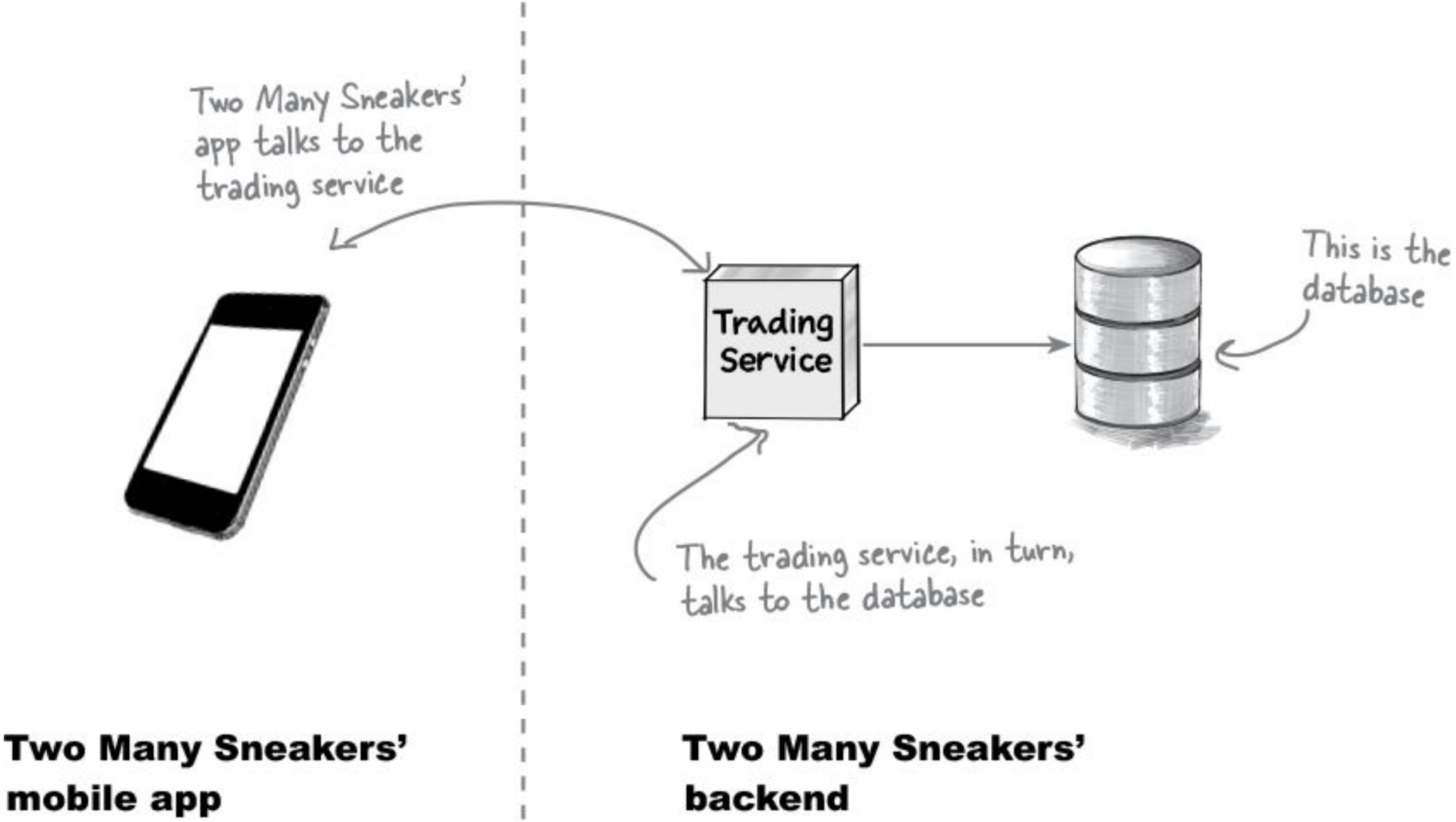
- Resume-ability
- Integratability
- Interoperabilityi

Architects often have a unique perspective as to what's possible within a given timeframe.

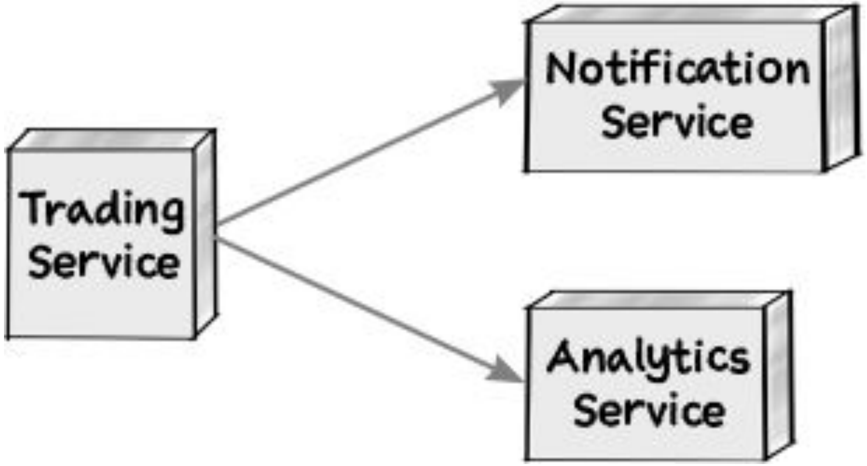
Feasability—evaluating whether something is possible—is an underutilized architecture "ility".

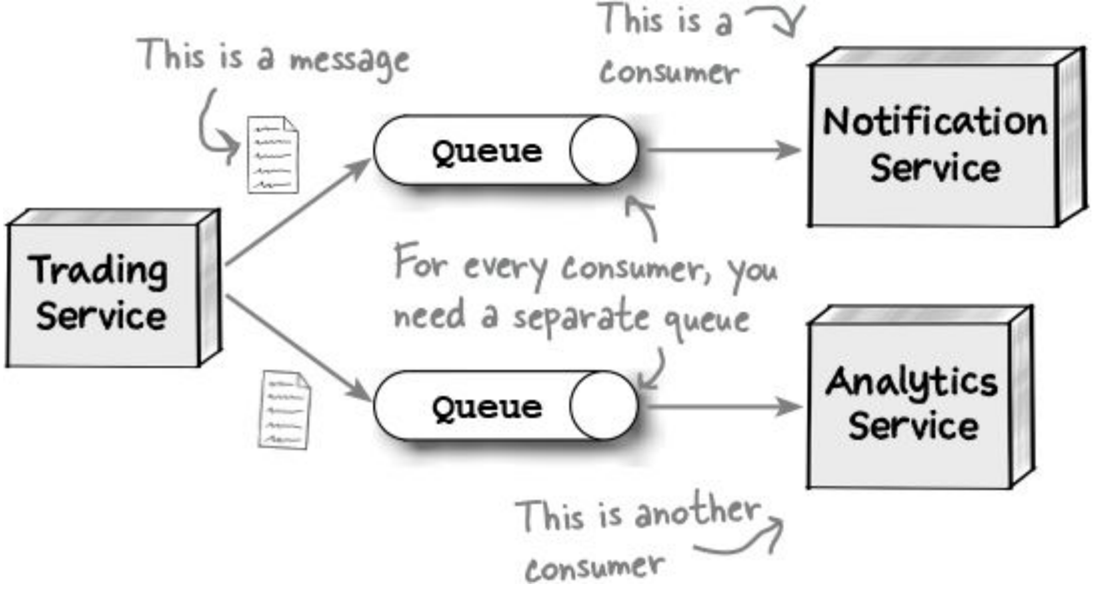
- Feasibility
- Simplicity

Exercício

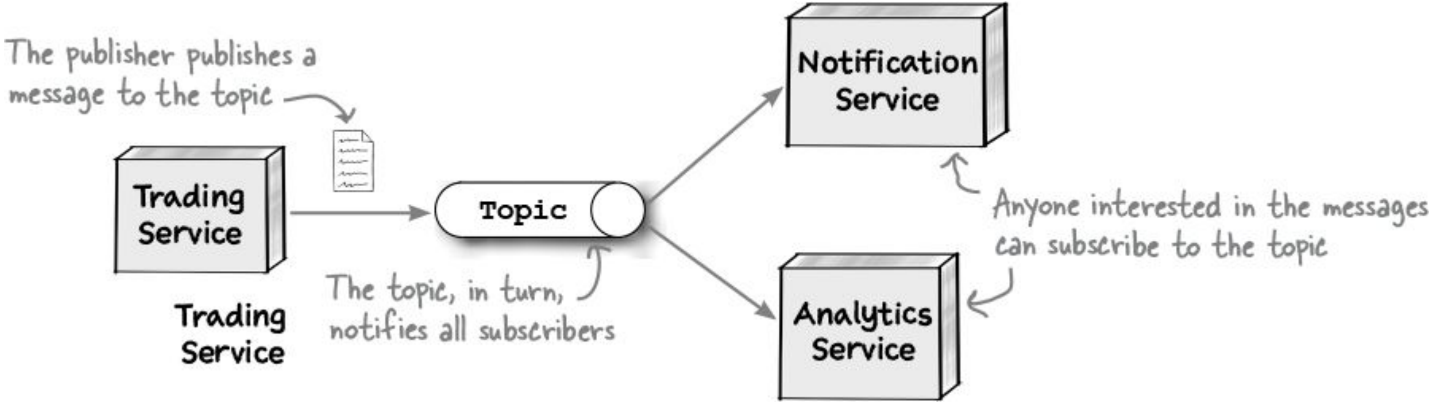

Two Many Sneakers' app talks to the trading service
Trading Service
This is the database
The trading service, in turn, talks to the database
Two Many Sneakers' mobile app
Two Many Sneakers' backend

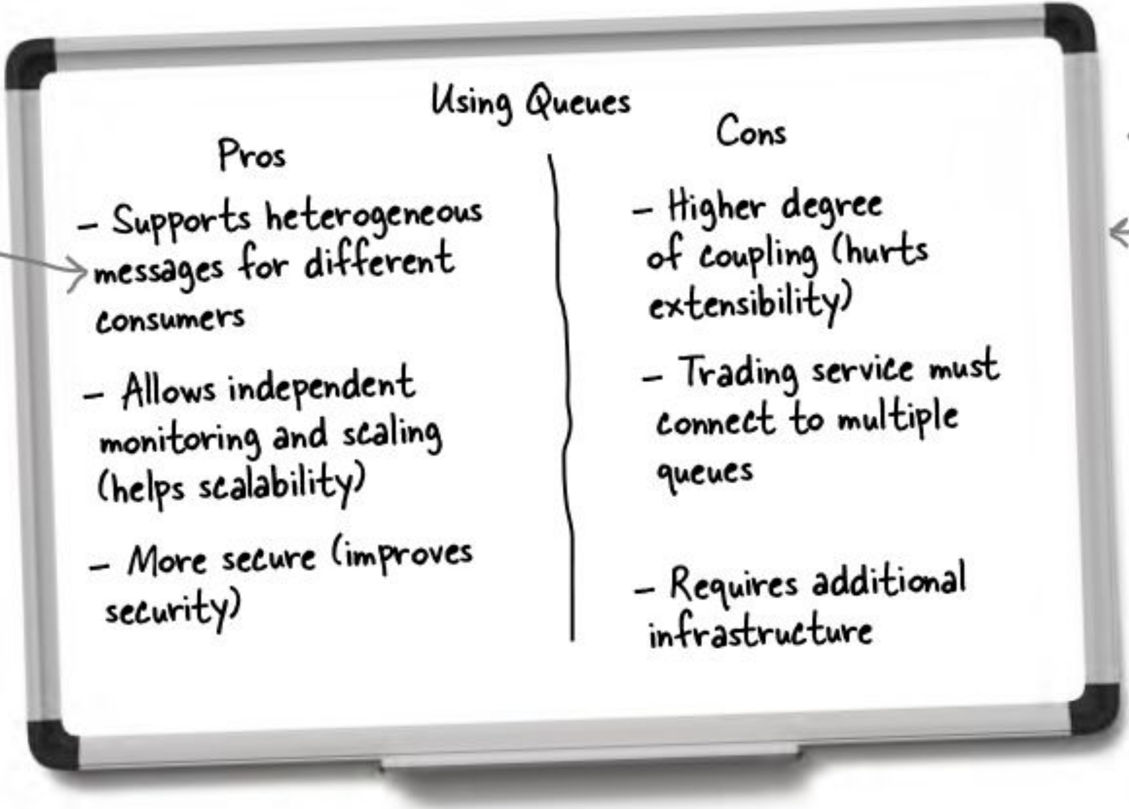
Como vc faz???




This is a message
This is a consumer
Queue
Notification Service
Trading Service
For every consumer, you need a separate queue
Queue
Analytics Service
This is another consumer

Versus


The publisher publishes a message to the topic
Notification Service
Trading Service
Topic
Anyone interested in the messages can subscribe to the topic
Trading Service
The topic, in turn, notifies all subscribers
Analytics Service



Using Queues

Pros

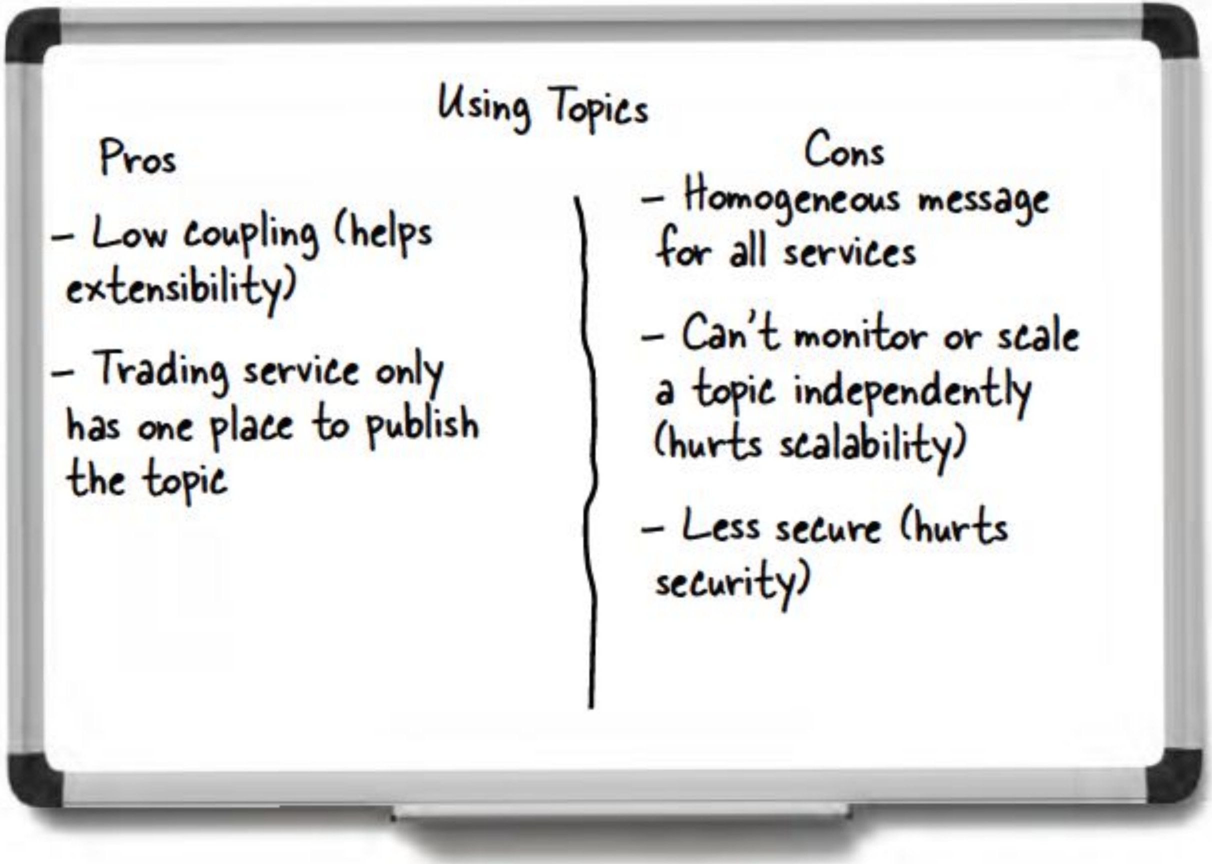
- Supports heterogeneous messages for different consumers
- Allows independent monitoring and scaling (helps scalability)
- More secure (improves security)

Cons

- Higher degree of coupling (hurts extensibility)
- Trading service must connect to multiple queues
- Requires additional infrastructure

Heterogeneous is just a fancy way of saying "different"

Whiteboards are great for brainstorming trade-offs with your team



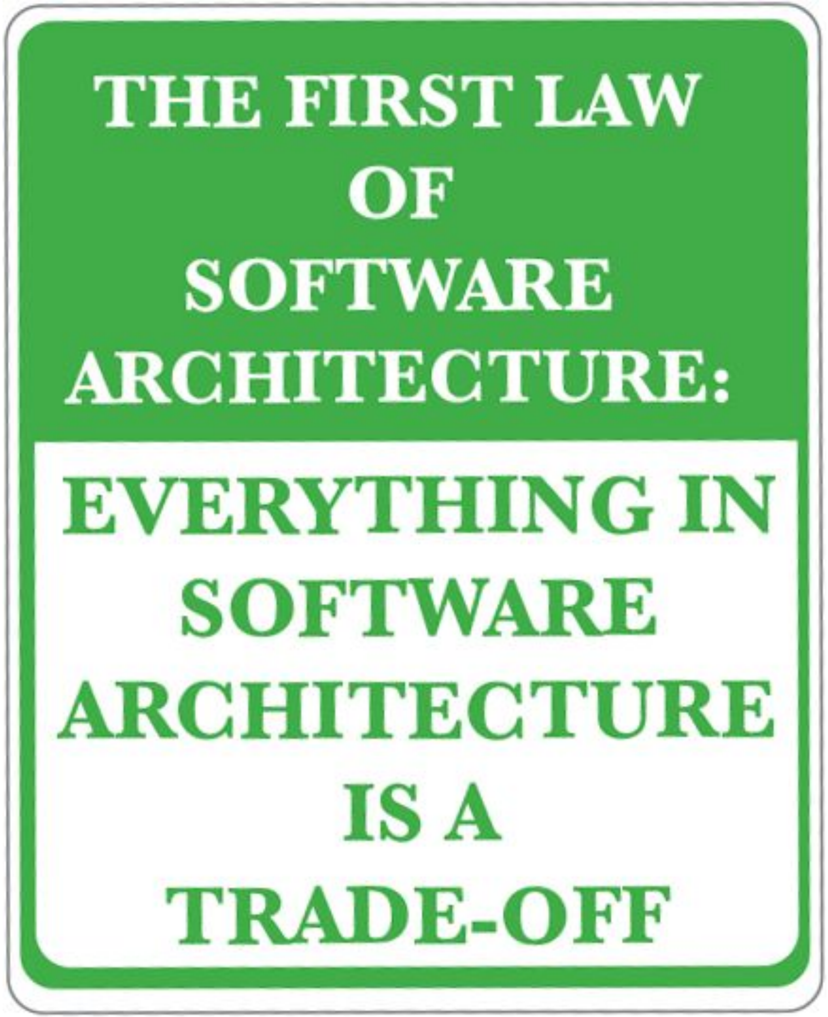
Using Topics

Pros

- Low coupling (helps extensibility)
- Trading service only has one place to publish the topic

Cons

- Homogeneous message for all services
- Can't monitor or scale a topic independently (hurts scalability)
- Less secure (hurts security)



THE FIRST LAW OF SOFTWARE ARCHITECTURE:

EVERYTHING IN SOFTWARE ARCHITECTURE IS A TRADE-OFF

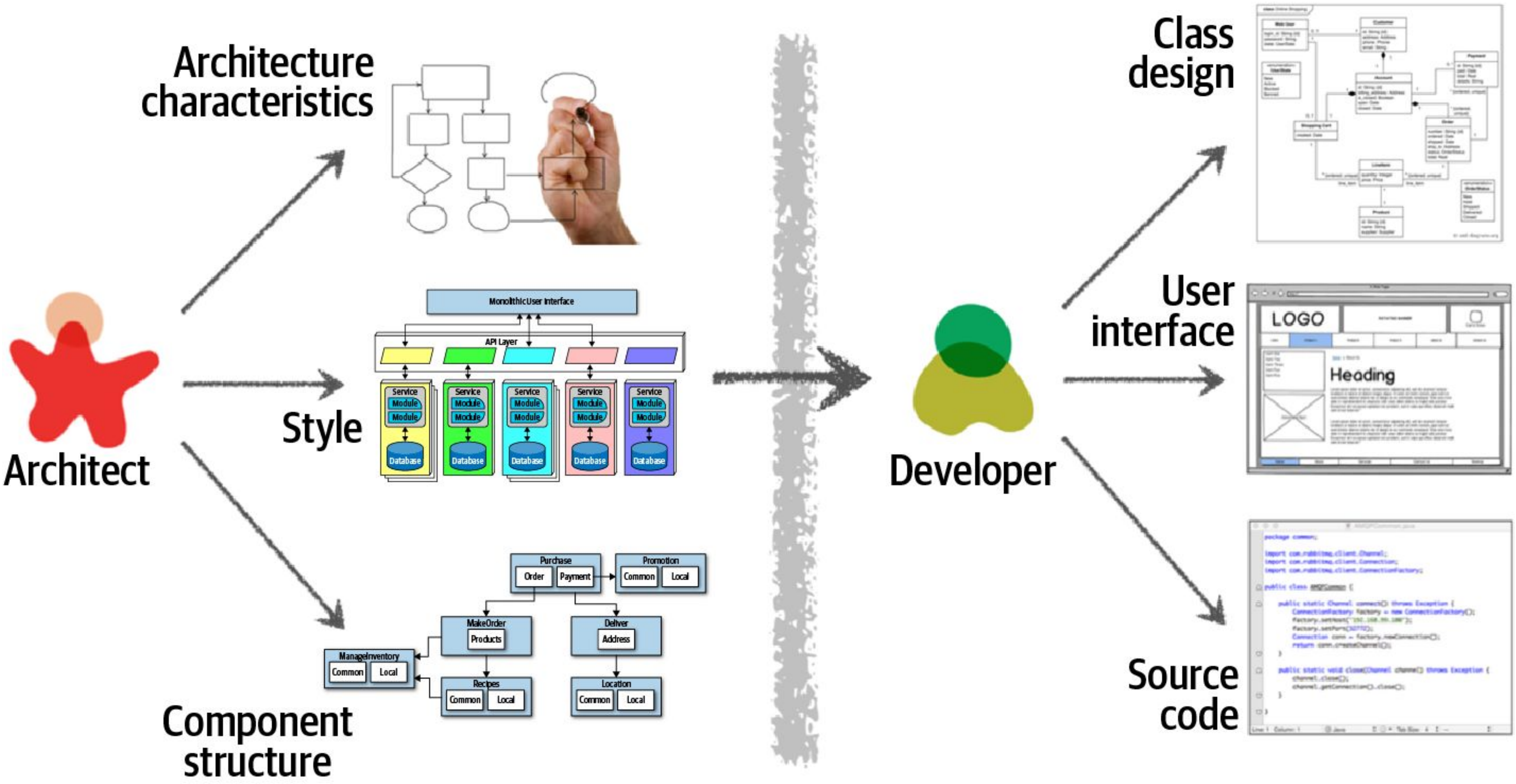
THE SECOND LAW OF SOFTWARE ARCHITECTURE:

WHY IS MORE IMPORTANT THAN HOW

```
We will use a cache to reduce the load on
the database and improve performance.
```

Notice how this decision introduces an additional piece of infrastructure. It's also something the implementing team must keep in the back of their minds when accessing or writing data.

Livro: Fundamentals of Software Architecture



Architect

Architecture characteristics

Style

Component structure

Developer

Class design

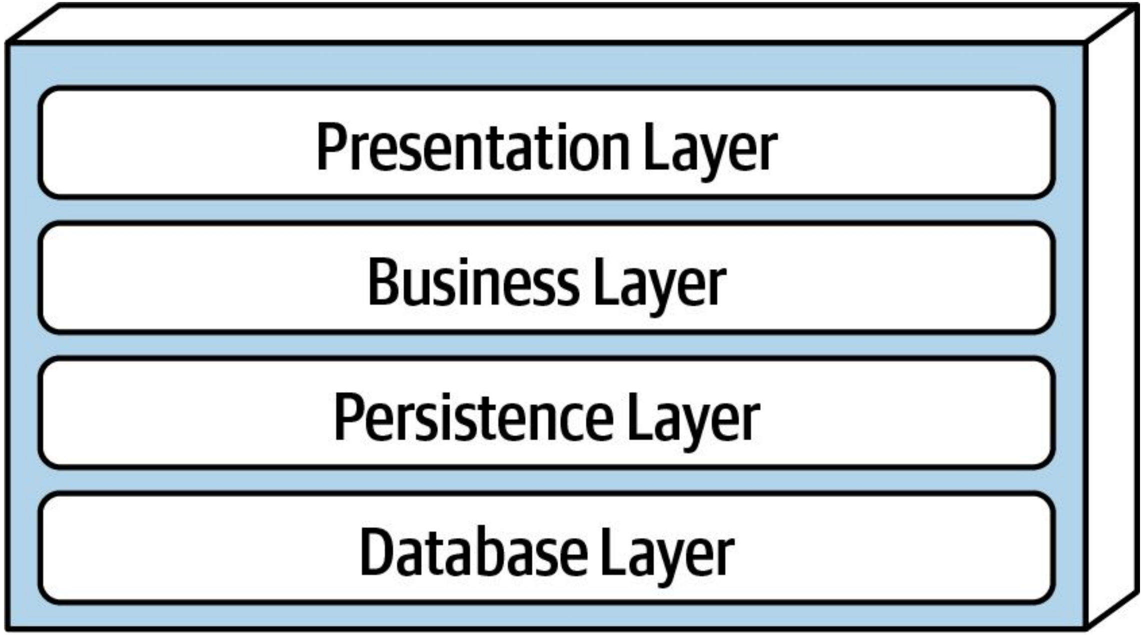
User interface

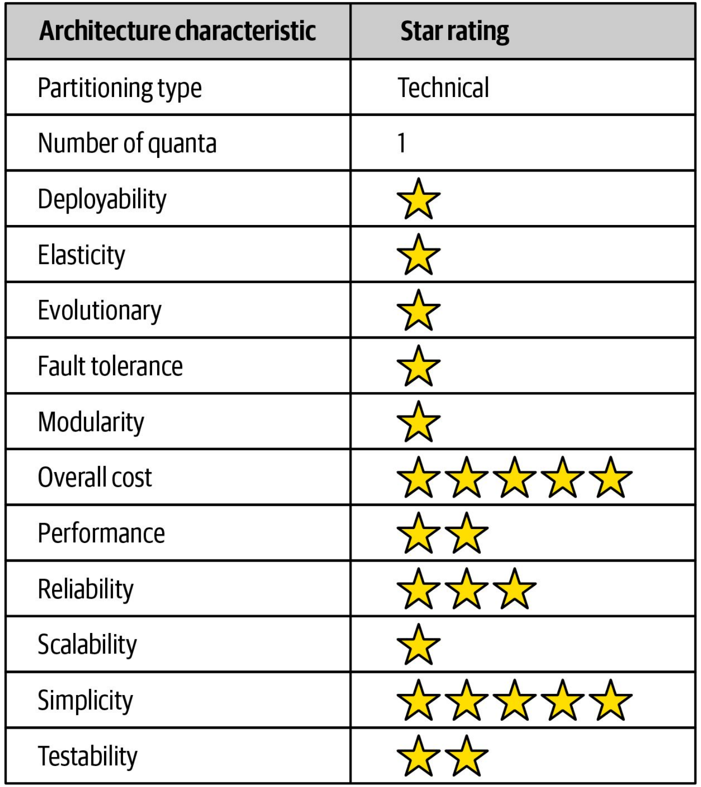
Source code

```
package common;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

public class AMQPCommon {

    public static Channel connect() throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("192.168.99.100");
        factory.setPort(32772);
        Connection conn = factory.newConnection();
        return conn.createChannel();
    }

    public static void close(Channel channel) throws Exception {
        channel.close();
        channel.getConnection().close();
    }

}
```

Programmers know the benefits of everything and the trade-offs of nothing. Architects need to understand both.

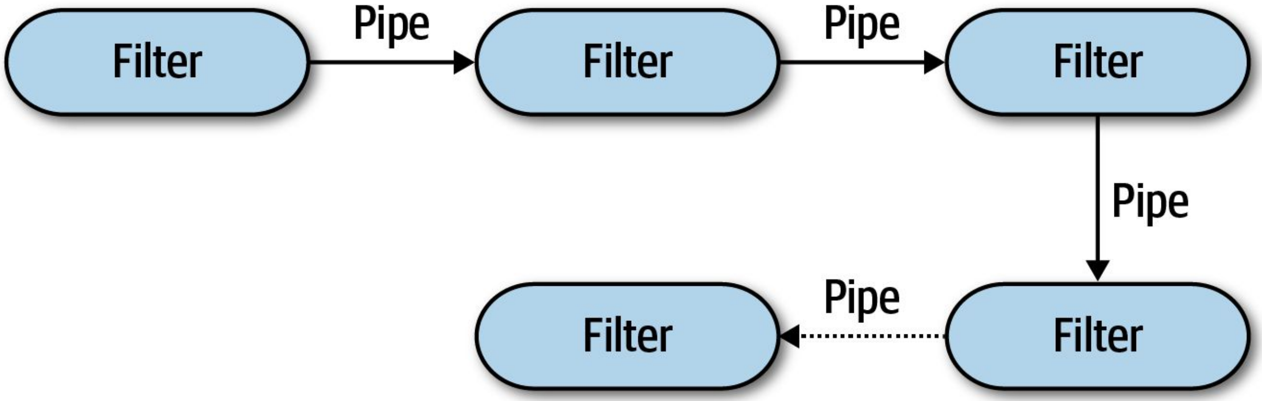
Layered Architecture Style





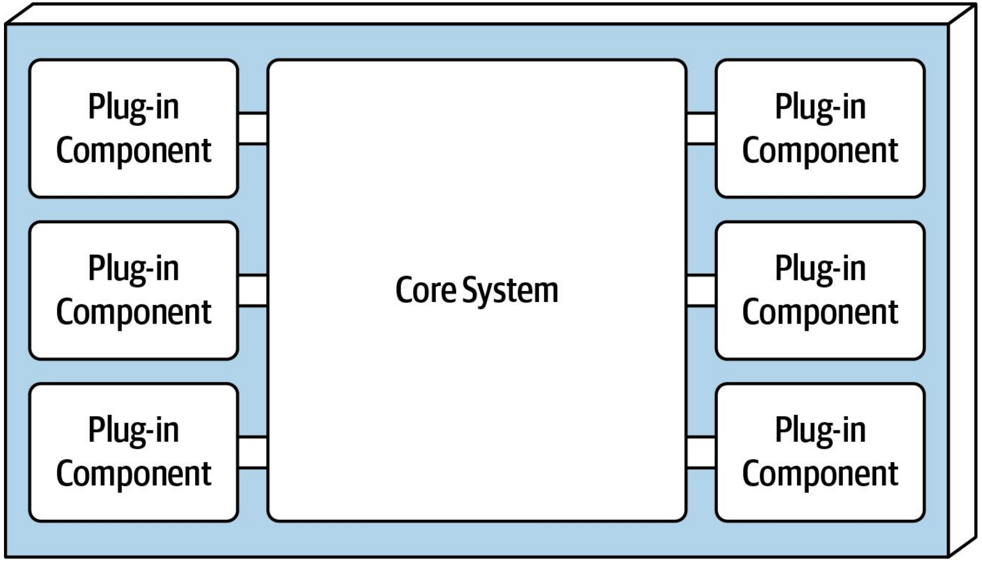
Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1
Deployability	★
Elasticity	★
Evolutionary	★
Fault tolerance	★
Modularity	★
Overall cost	★★★★★
Performance	★★
Reliability	★★★
Scalability	★
Simplicity	★★★★★
Testability	★★

Pipeline Architecture Style



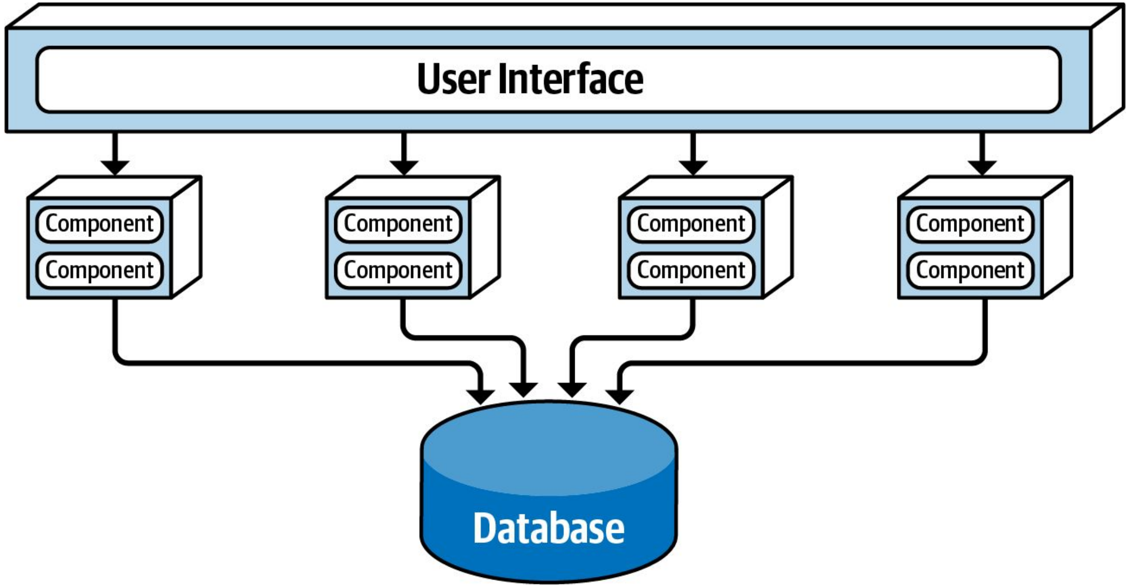
Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1
Deployability	★★
Elasticity	★
Evolutionary	★★★
Fault tolerance	★
Modularity	★★★
Overall cost	★★★★★
Performance	★★
Reliability	★★★
Scalability	★
Simplicity	★★★★★
Testability	★★★

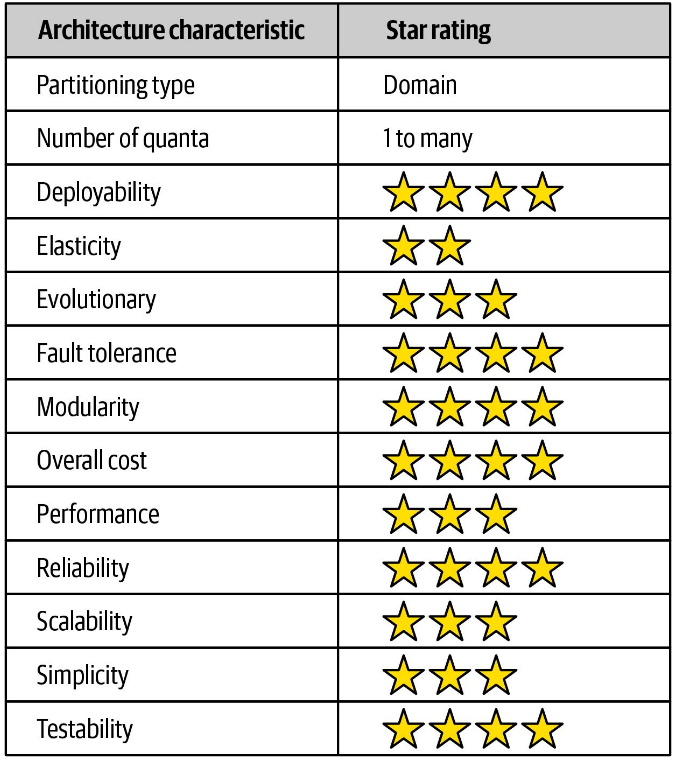
Microkernel Architecture Style



Architecture characteristic	Star rating
Partitioning type	Domain and technical
Number of quanta	1
Deployability	★★★
Elasticity	★
Evolutionary	★★★
Fault tolerance	★
Modularity	★★★
Overall cost	★★★★★
Performance	★★★
Reliability	★★★
Scalability	★
Simplicity	★★★★
Testability	★★★

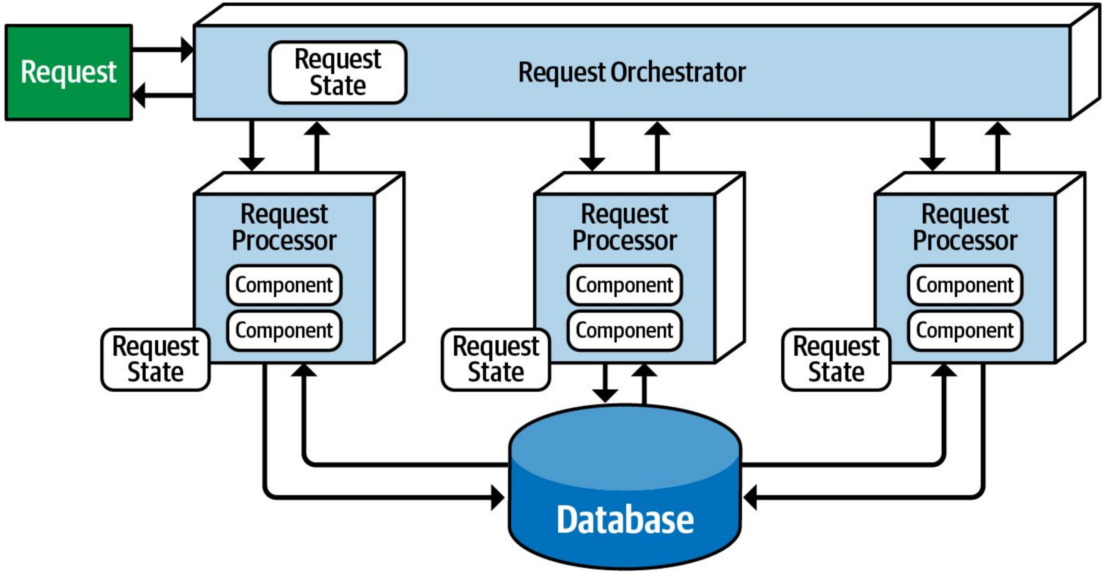
Service-Based Architecture Style





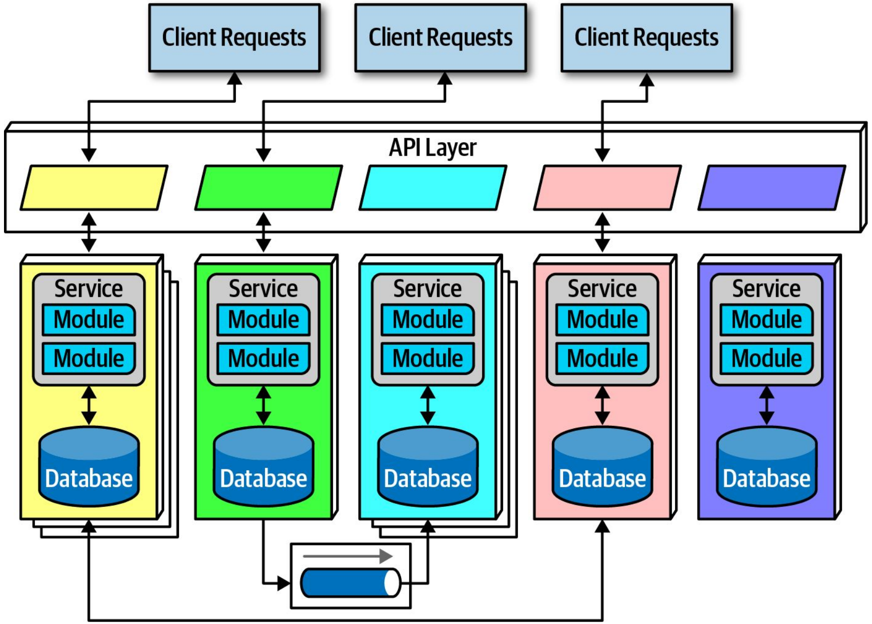
Architecture characteristic	Star rating
Partitioning type	Domain
Number of quanta	1 to many
Deployability	★★★★
Elasticity	★★
Evolutionary	★★★
Fault tolerance	★★★★
Modularity	★★★★
Overall cost	★★★★
Performance	★★★
Reliability	★★★★
Scalability	★★★
Simplicity	★★★
Testability	★★★★

Event-Driven Architecture Style



Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1 to many
Deployability	★★★
Elasticity	★★★
Evolutionary	★★★★★
Fault tolerance	★★★★★
Modularity	★★★★
Overall cost	★★★
Performance	★★★★★
Reliability	★★★
Scalability	★★★★★
Simplicity	★
Testability	★★

Microservices Architecture



Architecture characteristic	Star rating
Partitioning type	Domain
Number of quanta	1 to many
Deployability	★★★★
Elasticity	★★★★★
Evolutionary	★★★★★
Fault tolerance	★★★★
Modularity	★★★★★
Overall cost	★
Performance	★★
Reliability	★★★★
Scalability	★★★★★
Simplicity	★
Testability	★★★★

Livro soft arch hard parts

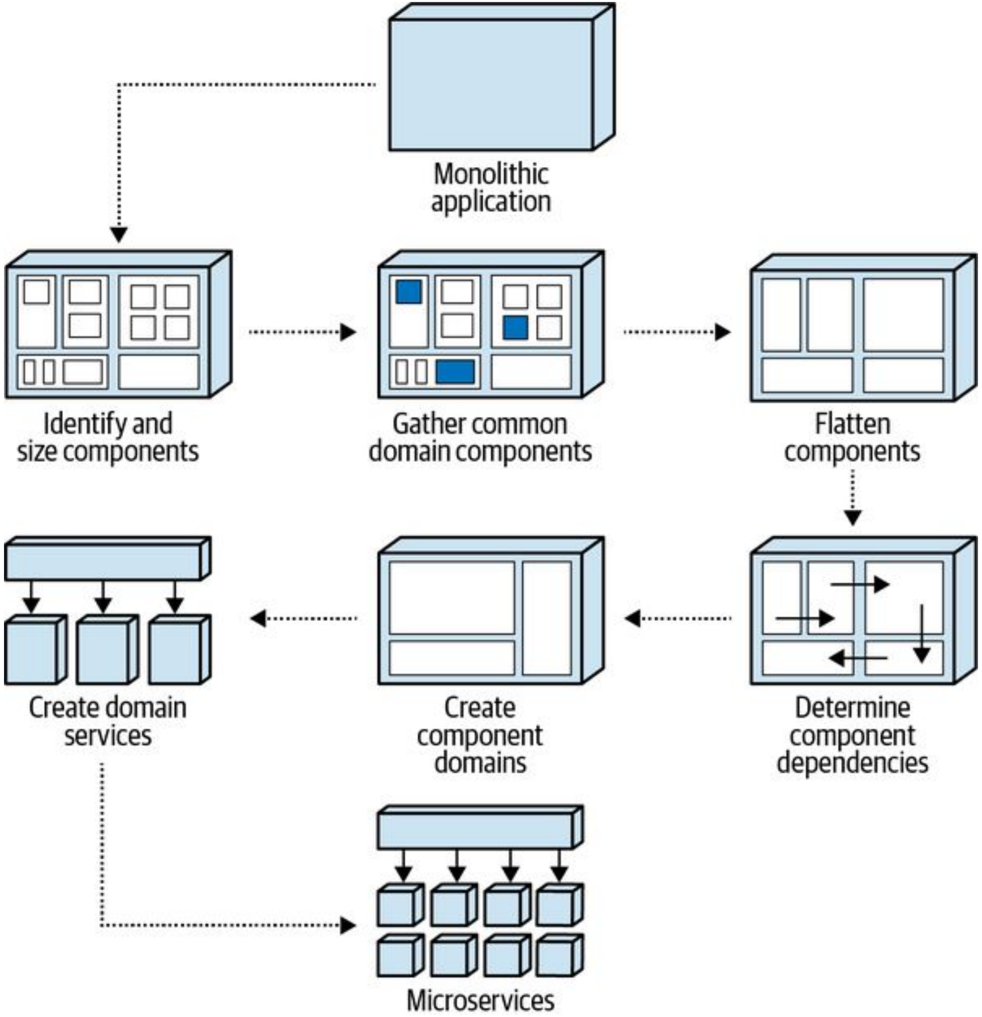
Data is a precious thing and will last longer

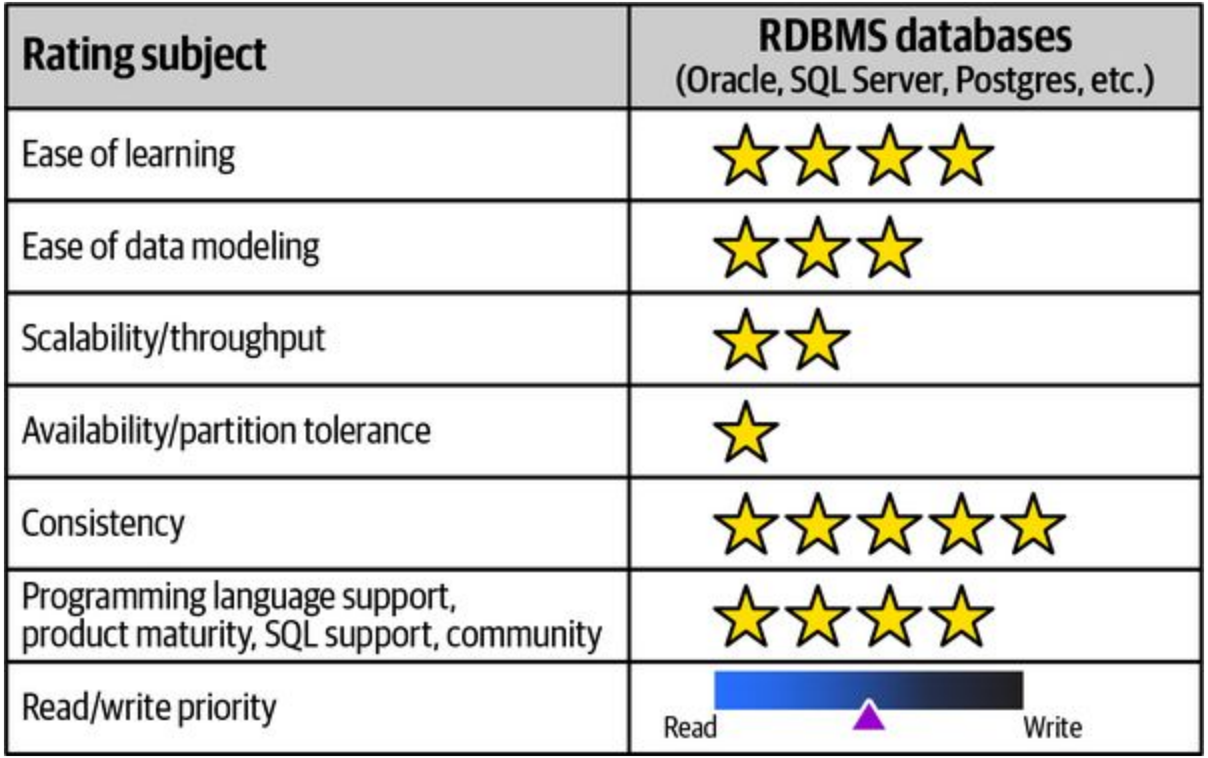
than the systems themselves.

Tim Berners-Lee

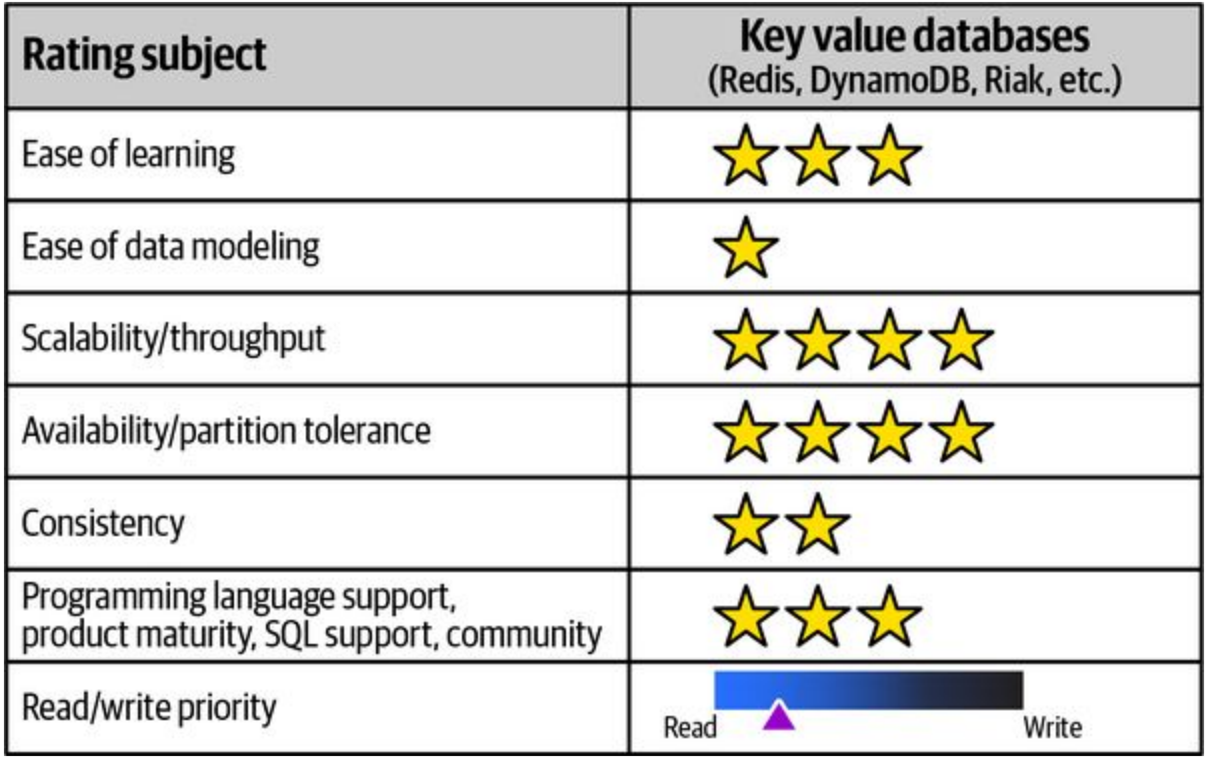
All things are poison, and nothing is without poison; the dosage alone makes it so a thing is not a poison.

Paracelsus

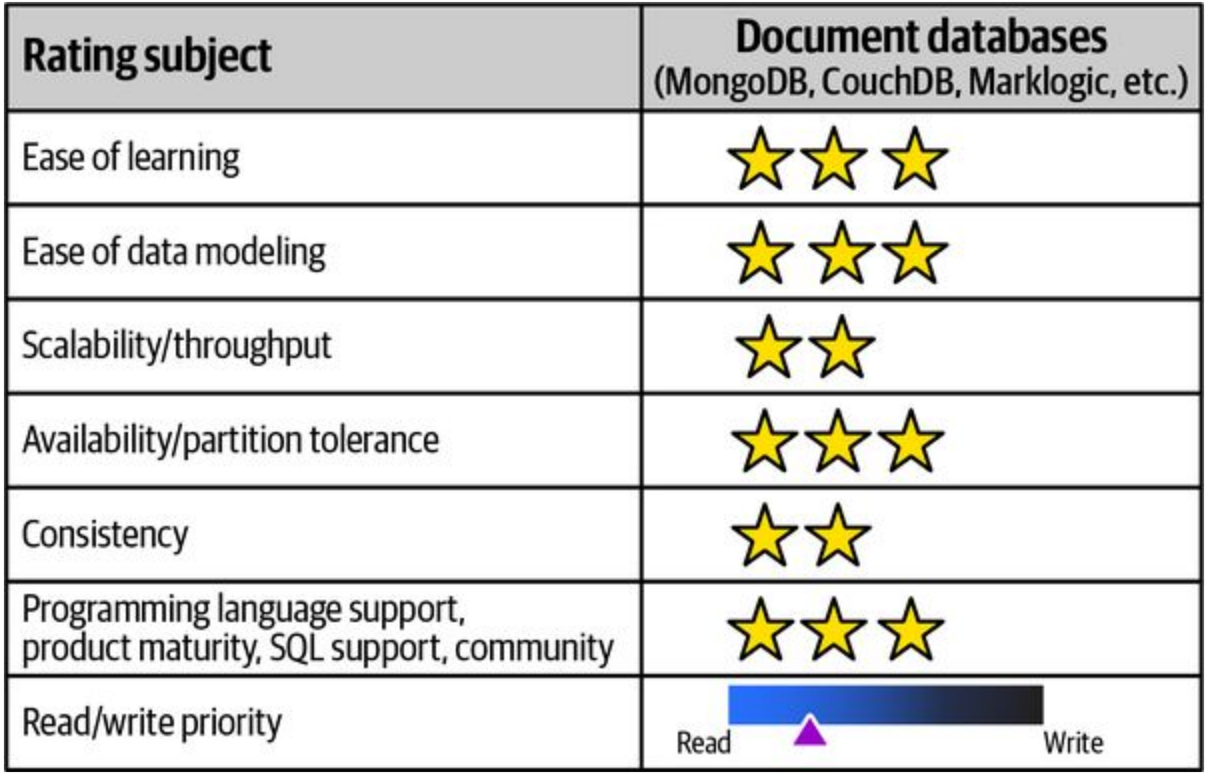

Monolithic application
Identify and size components
Gather common domain components
Flatten components
Determine component dependencies
Create component domains
Create domain services
Microservices



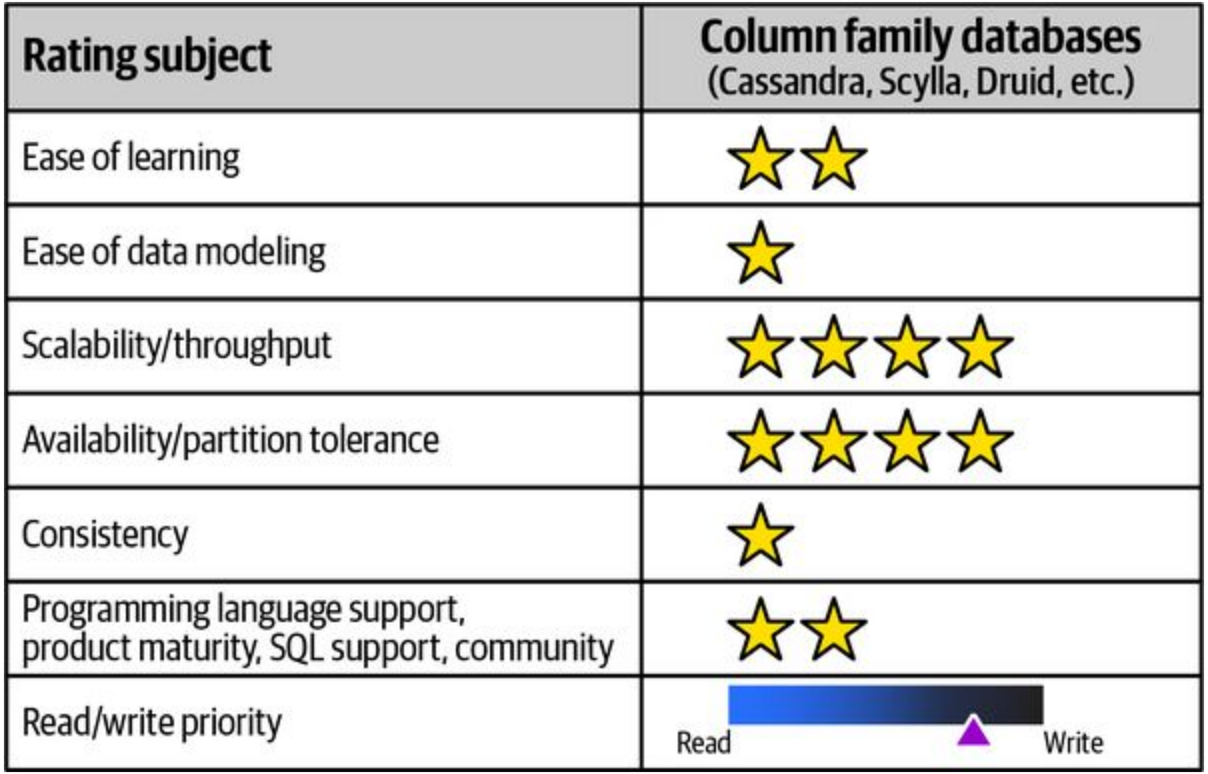
Rating subject	RDBMS databases (Oracle, SQL Server, Postgres, etc.)
Ease of learning	★★★★
Ease of data modeling	★★★
Scalability/throughput	★★
Availability/partition tolerance	★
Consistency	★★★★★
Programming language support, product maturity, SQL support, community	★★★★
Read/write priority	Read ▲ Write



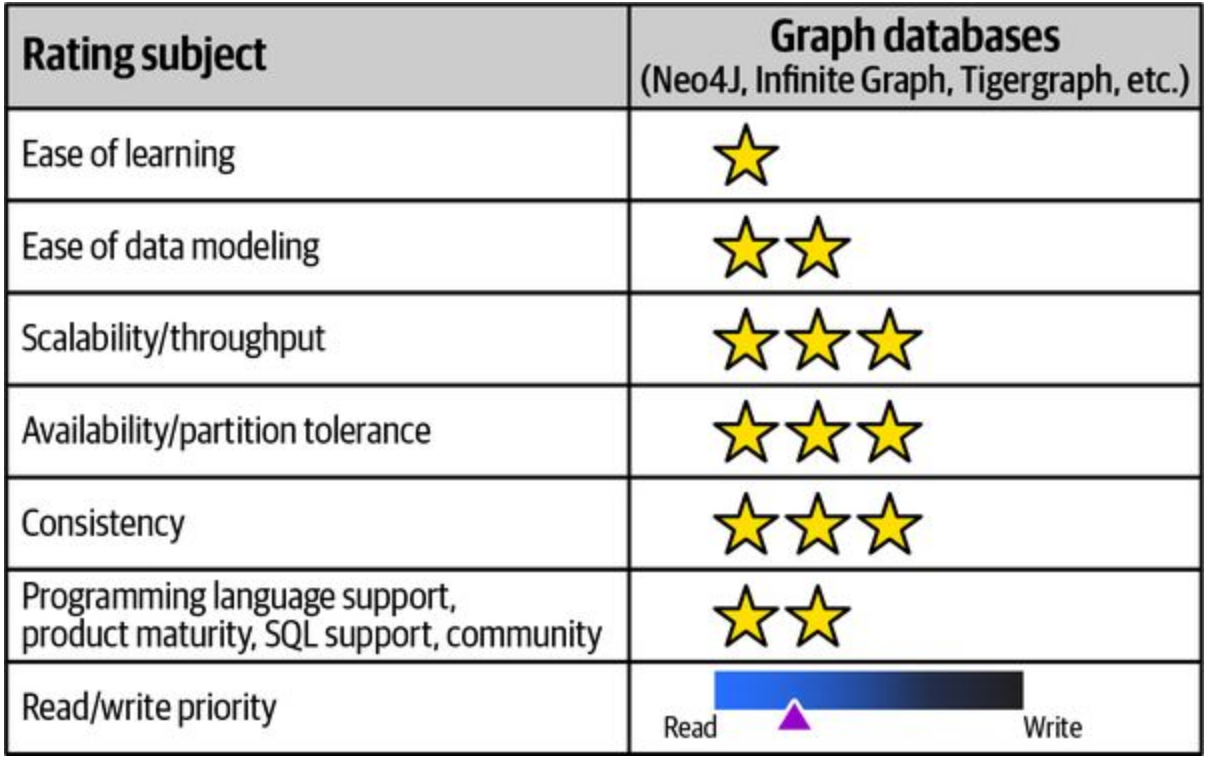
Rating subject	Key value databases (Redis, DynamoDB, Riak, etc.)
Ease of learning	★★★
Ease of data modeling	★
Scalability/throughput	★★★★
Availability/partition tolerance	★★★★
Consistency	★★
Programming language support, product maturity, SQL support, community	★★★
Read/write priority	Read ▲ Write



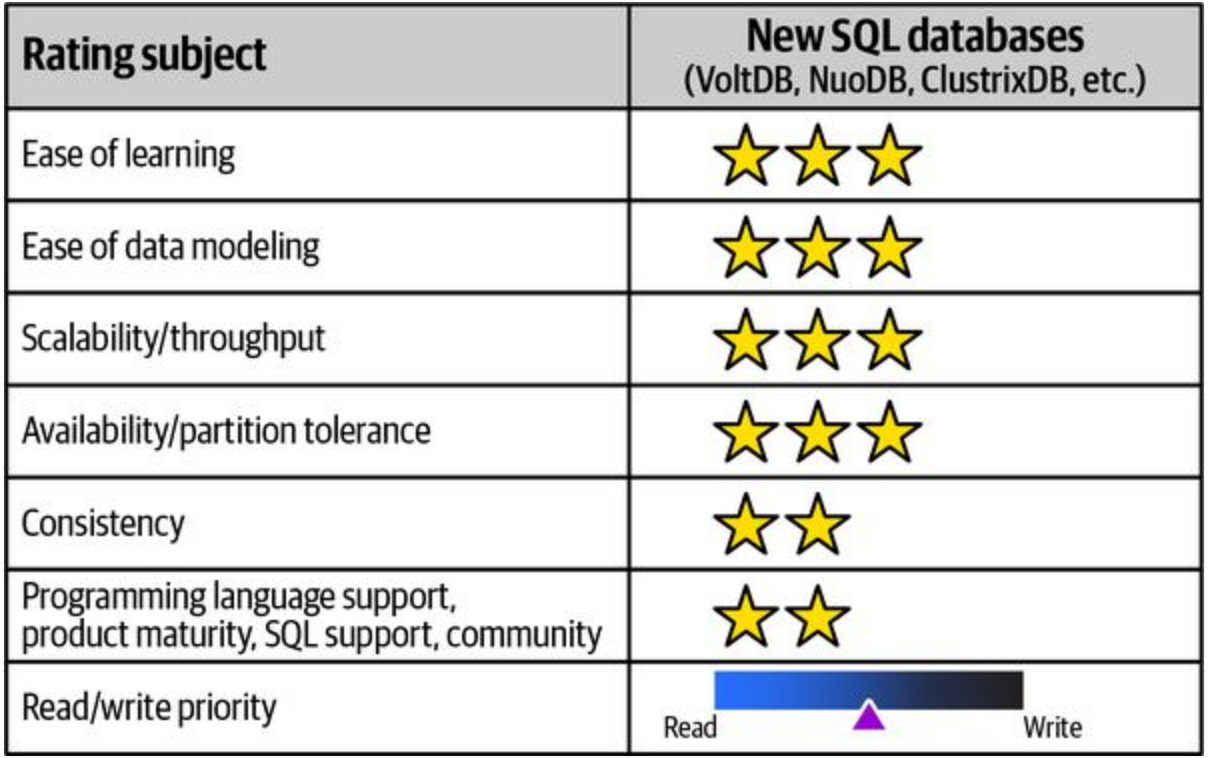
Rating subject	Document databases (MongoDB, CouchDB, Marklogic, etc.)
Ease of learning	★★★
Ease of data modeling	★★★
Scalability/throughput	★★
Availability/partition tolerance	★★★
Consistency	★★
Programming language support, product maturity, SQL support, community	★★★
Read/write priority	Read ▲ Write



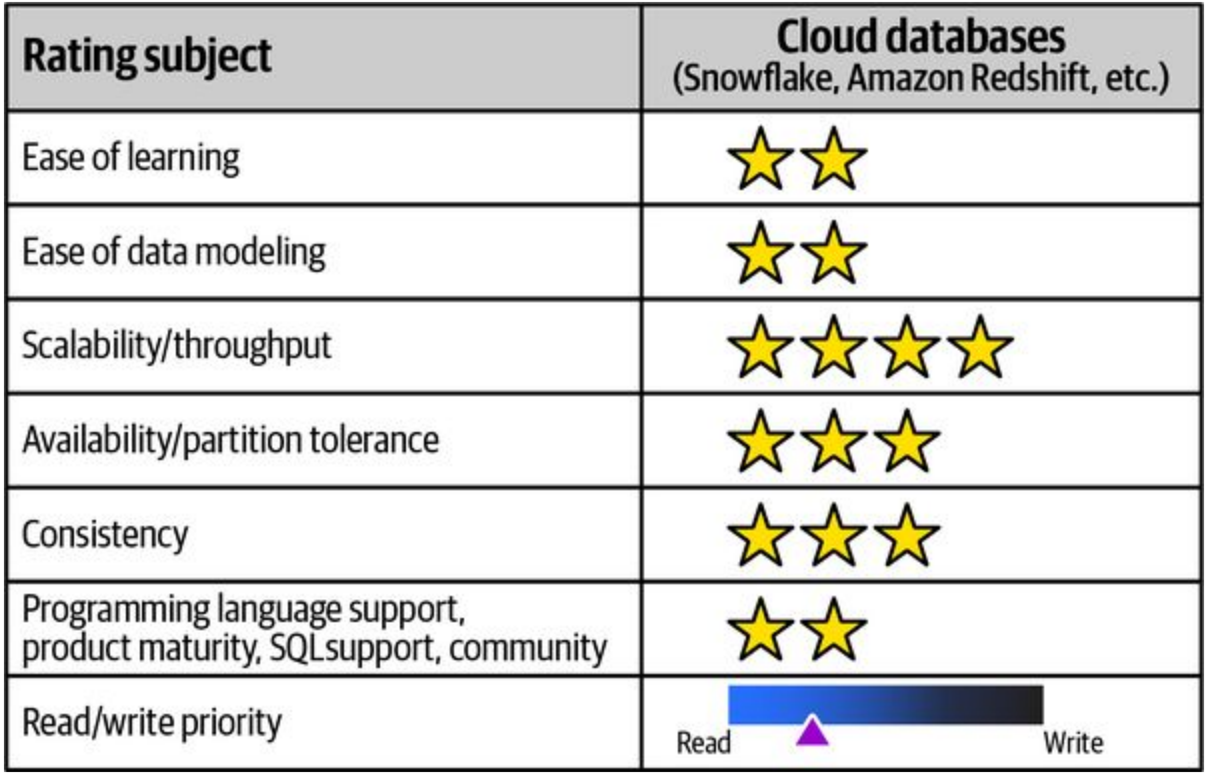
Rating subject	**Column family databases** (Cassandra, Scylla, Druid, etc.)
Ease of learning	★★
Ease of data modeling	★
Scalability/throughput	★★★★
Availability/partition tolerance	★★★★
Consistency	★
Programming language support, product maturity, SQL support, community	★★
Read/write priority	Read ▬▬▬▬▬▬▲▬ Write



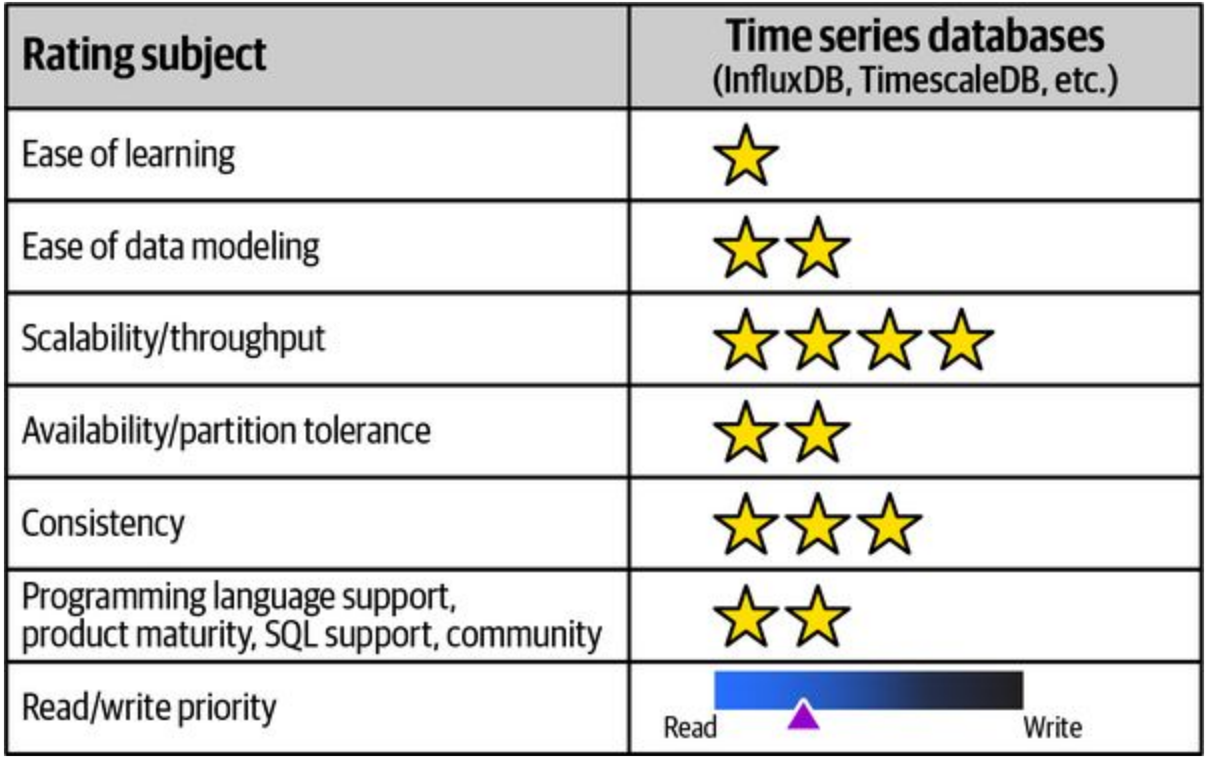
Rating subject	**Graph databases** (Neo4J, Infinite Graph, Tigergraph, etc.)
Ease of learning	★
Ease of data modeling	★★
Scalability/throughput	★★★
Availability/partition tolerance	★★★
Consistency	★★★
Programming language support, product maturity, SQL support, community	★★
Read/write priority	Read ▲ Write



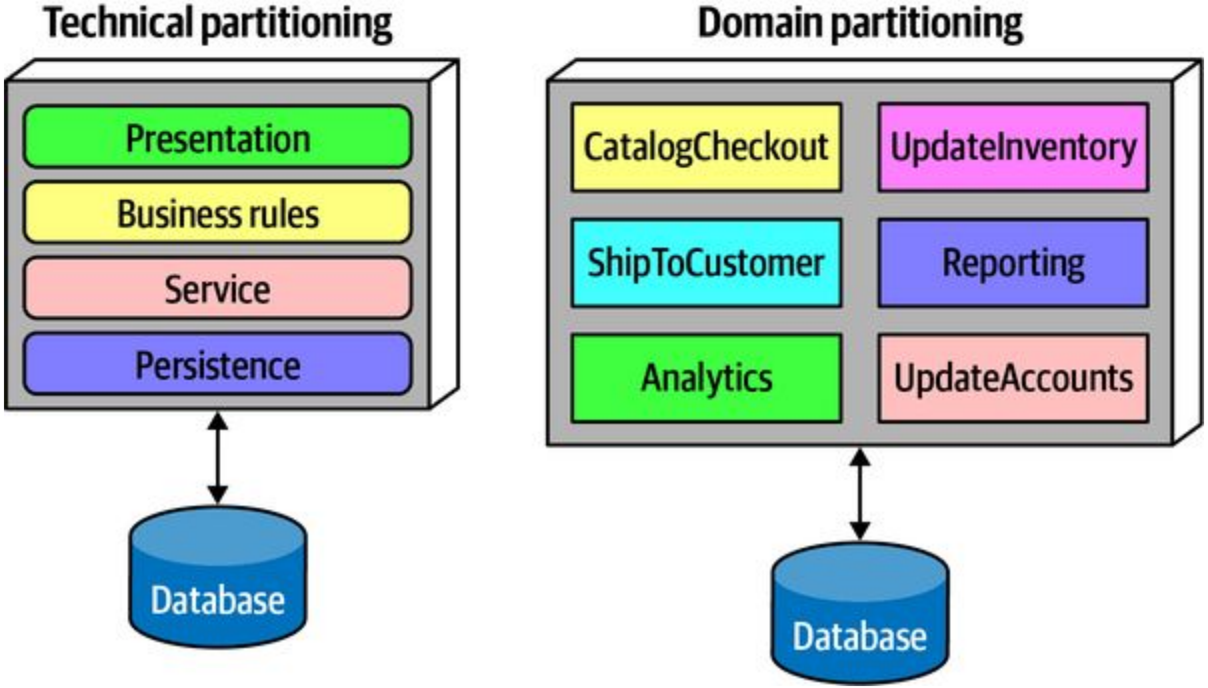
Rating subject	**New SQL databases** (VoltDB, NuoDB, ClustrixDB, etc.)
Ease of learning	★★★
Ease of data modeling	★★★
Scalability/throughput	★★★
Availability/partition tolerance	★★★
Consistency	★★
Programming language support, product maturity, SQL support, community	★★
Read/write priority	Read ▲ Write



Rating subject	Cloud databases (Snowflake, Amazon Redshift, etc.)
Ease of learning	★★
Ease of data modeling	★★
Scalability/throughput	★★★★
Availability/partition tolerance	★★★
Consistency	★★★
Programming language support, product maturity, SQLsupport, community	★★
Read/write priority	Read ▲ Write



Rating subject	Time series databases (InfluxDB, TimescaleDB, etc.)
Ease of learning	★
Ease of data modeling	★★
Scalability/throughput	★★★★
Availability/partition tolerance	★★
Consistency	★★★
Programming language support, product maturity, SQL support, community	★★
Read/write priority	Read ▲ Write



Technical partitioning

Presentation

Business rules

Service

Persistence

Database

Domain partitioning

CatalogCheckout

UpdateInventory

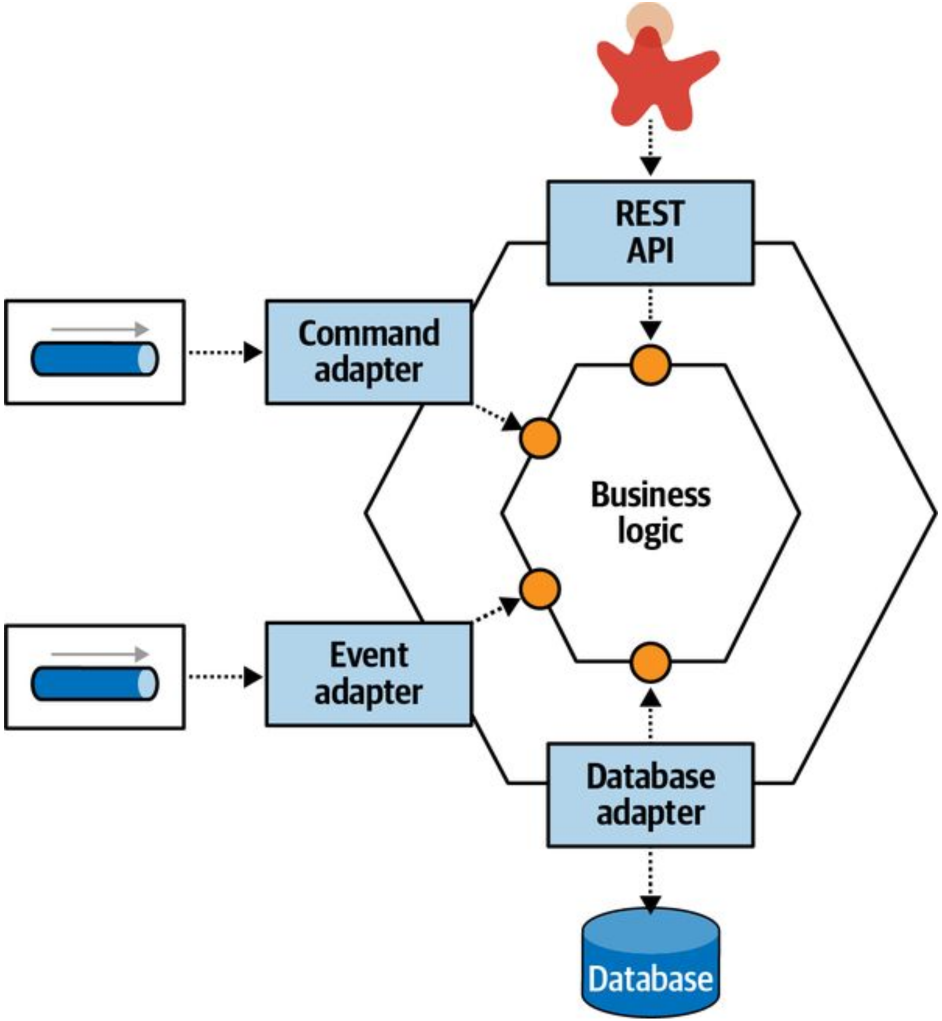
ShipToCustomer

Reporting

Analytics

UpdateAccounts

Database


REST API
Command adapter
Business logic
Event adapter
Database adapter
Database