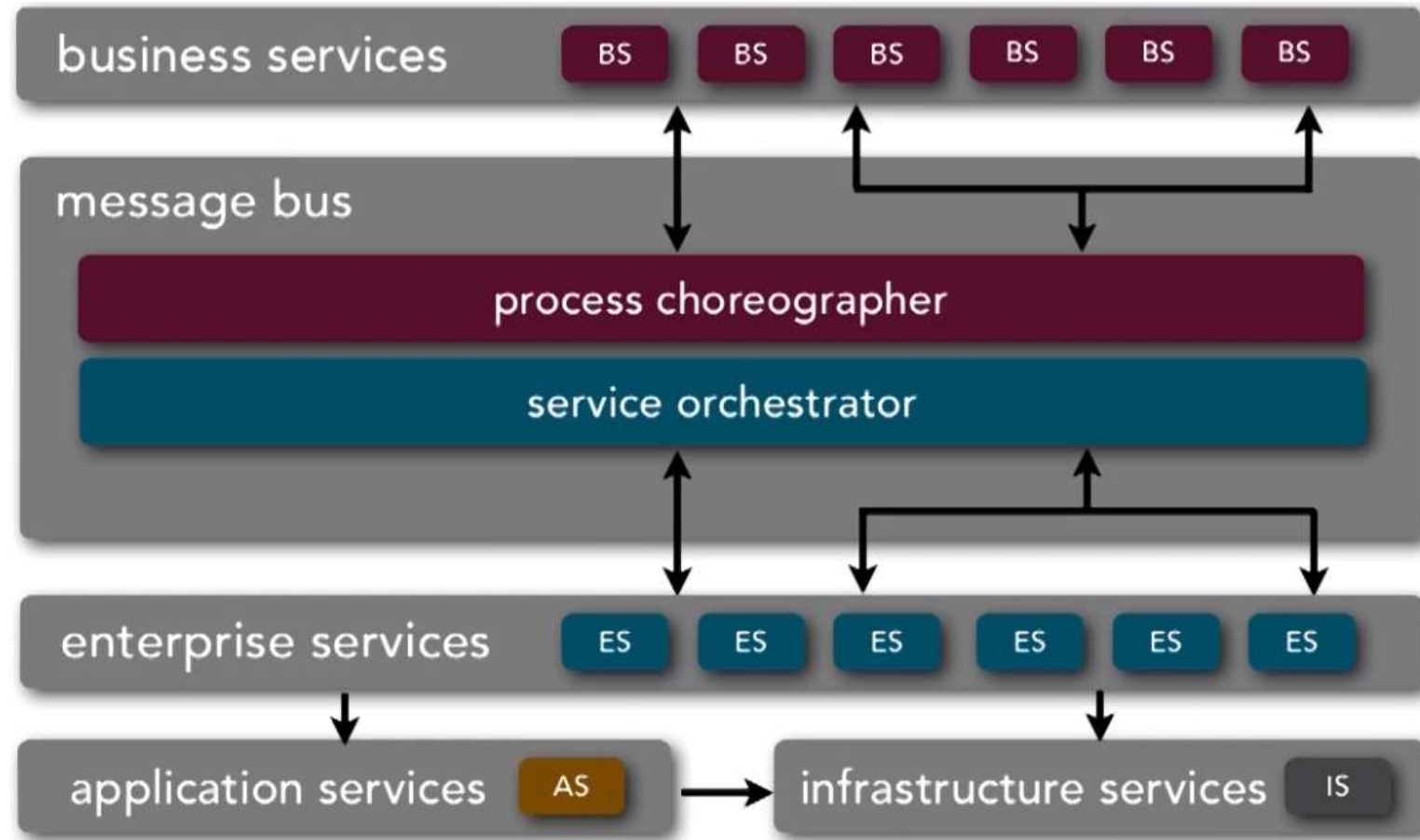
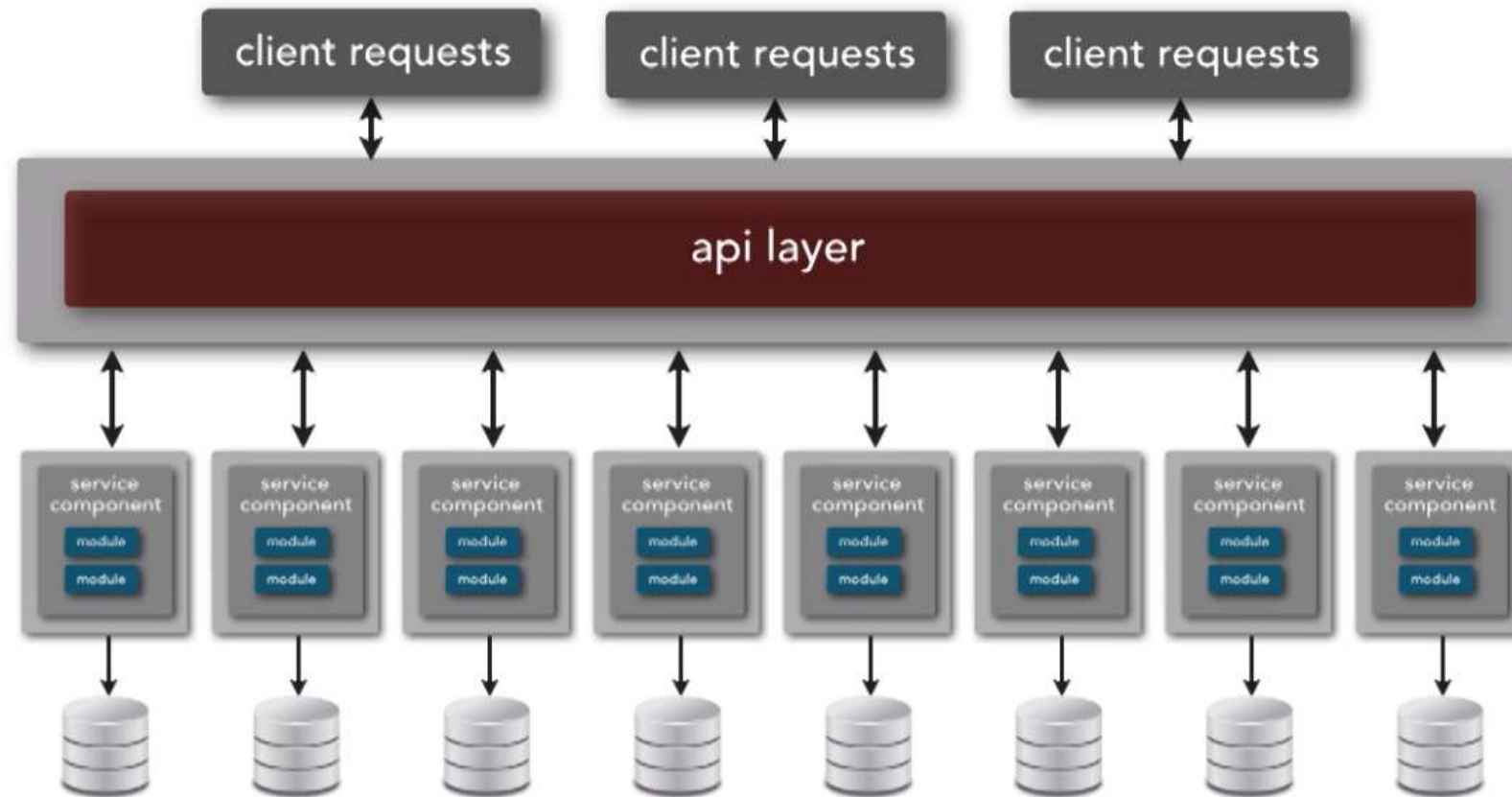


SOA vs Microservices vs SBA

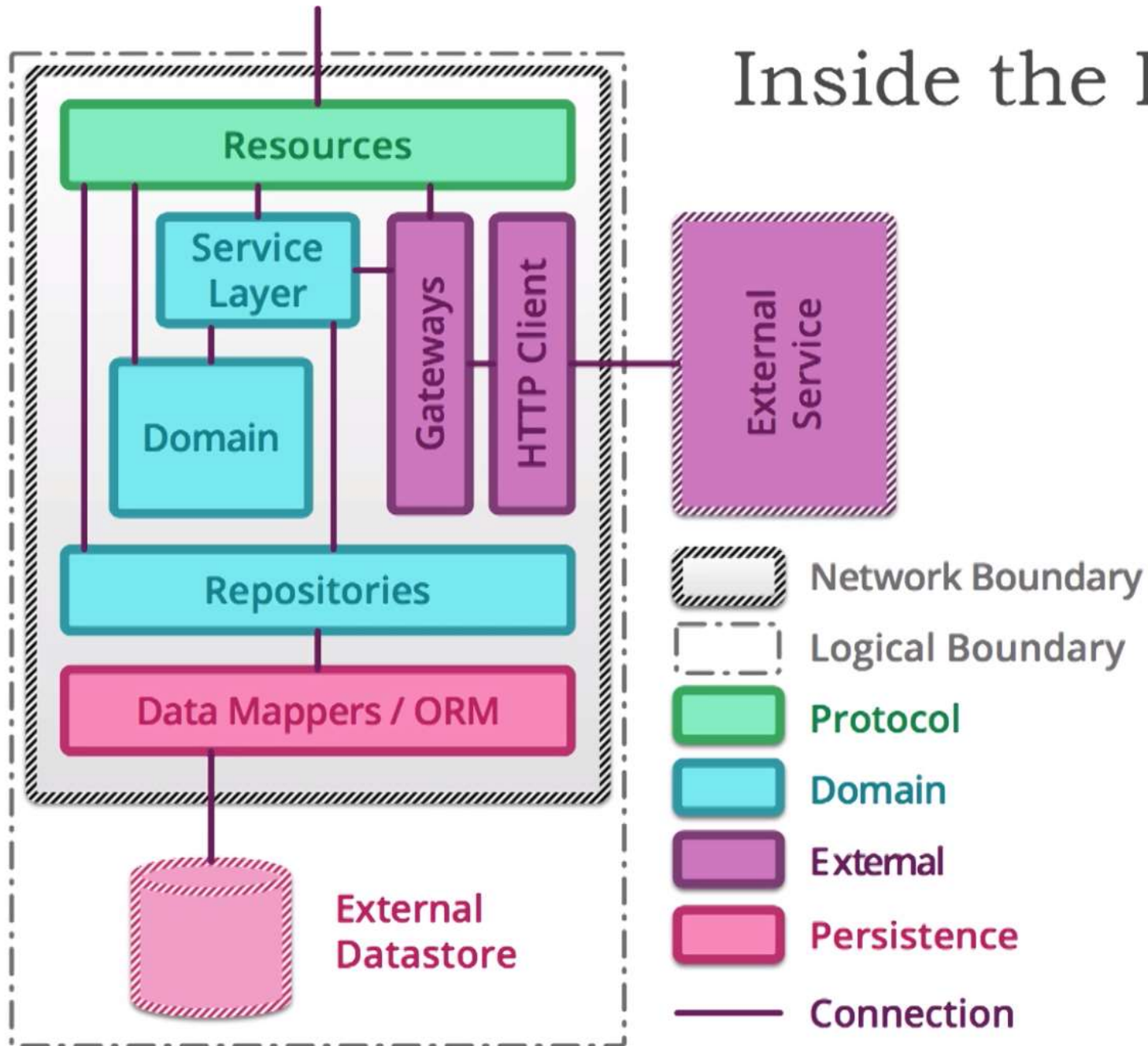
Service-Oriented Architecture (SOA)



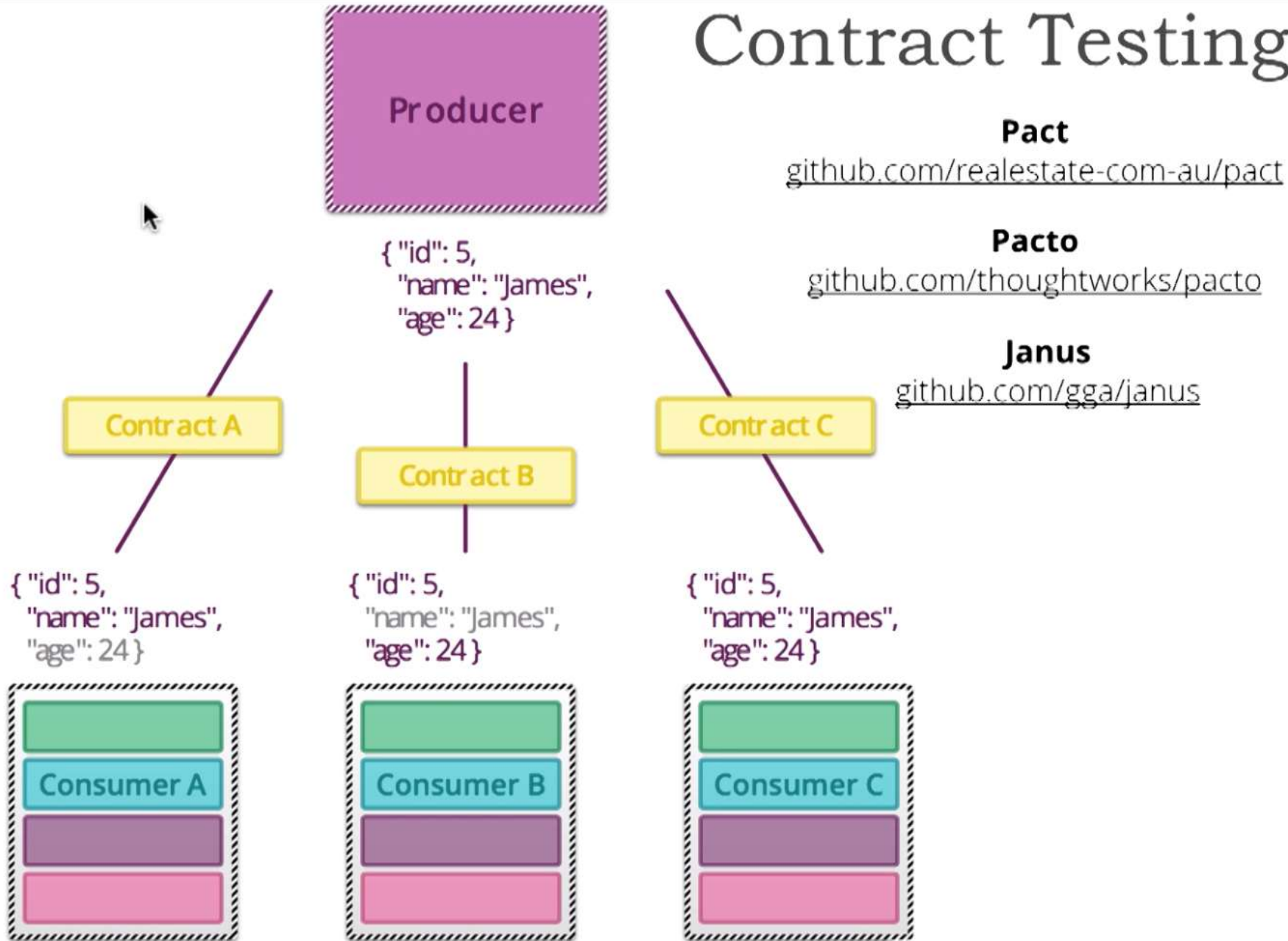
Microservices Architecture



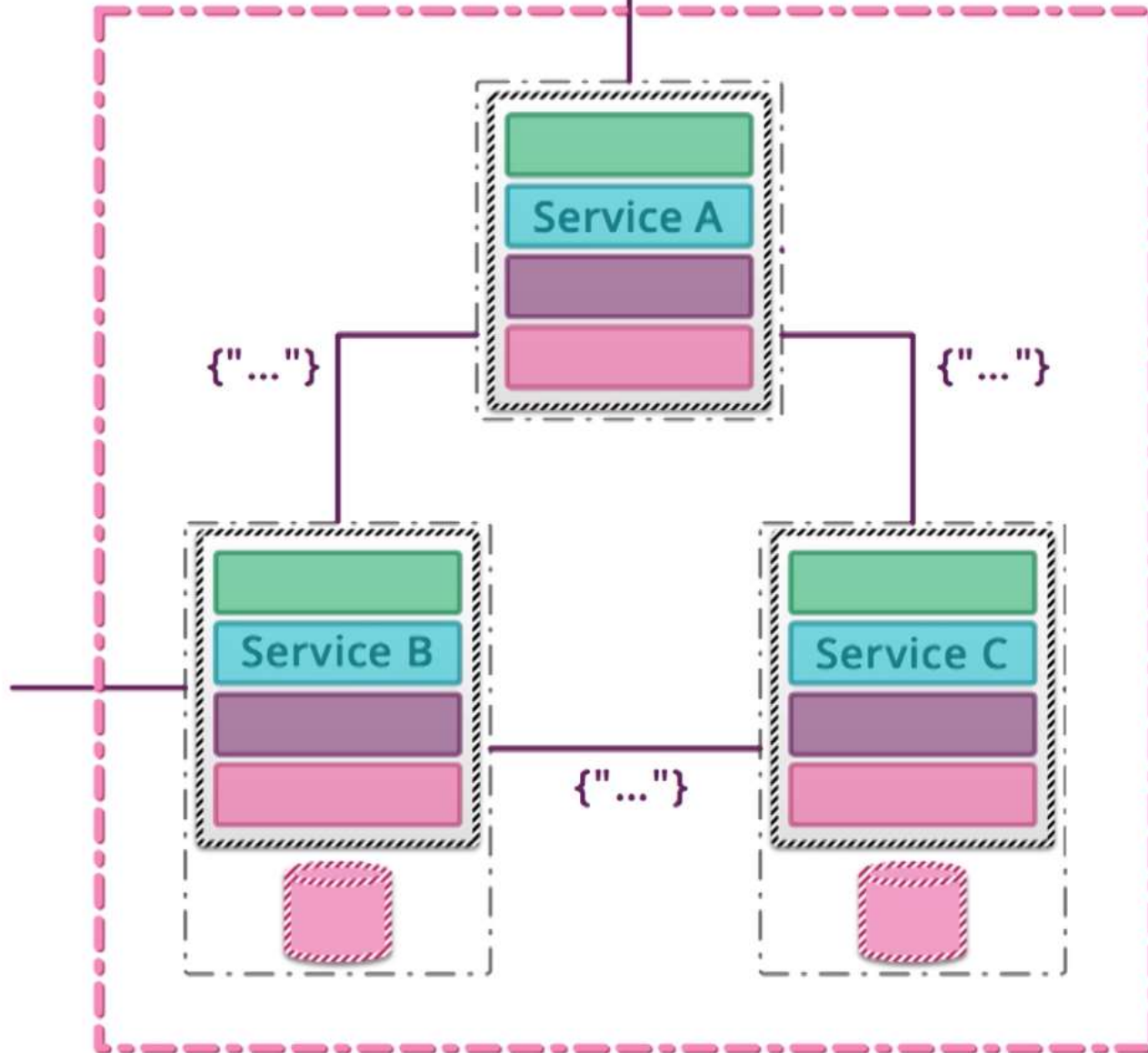
Inside the Box



Contract Testing



End-to-End Testing



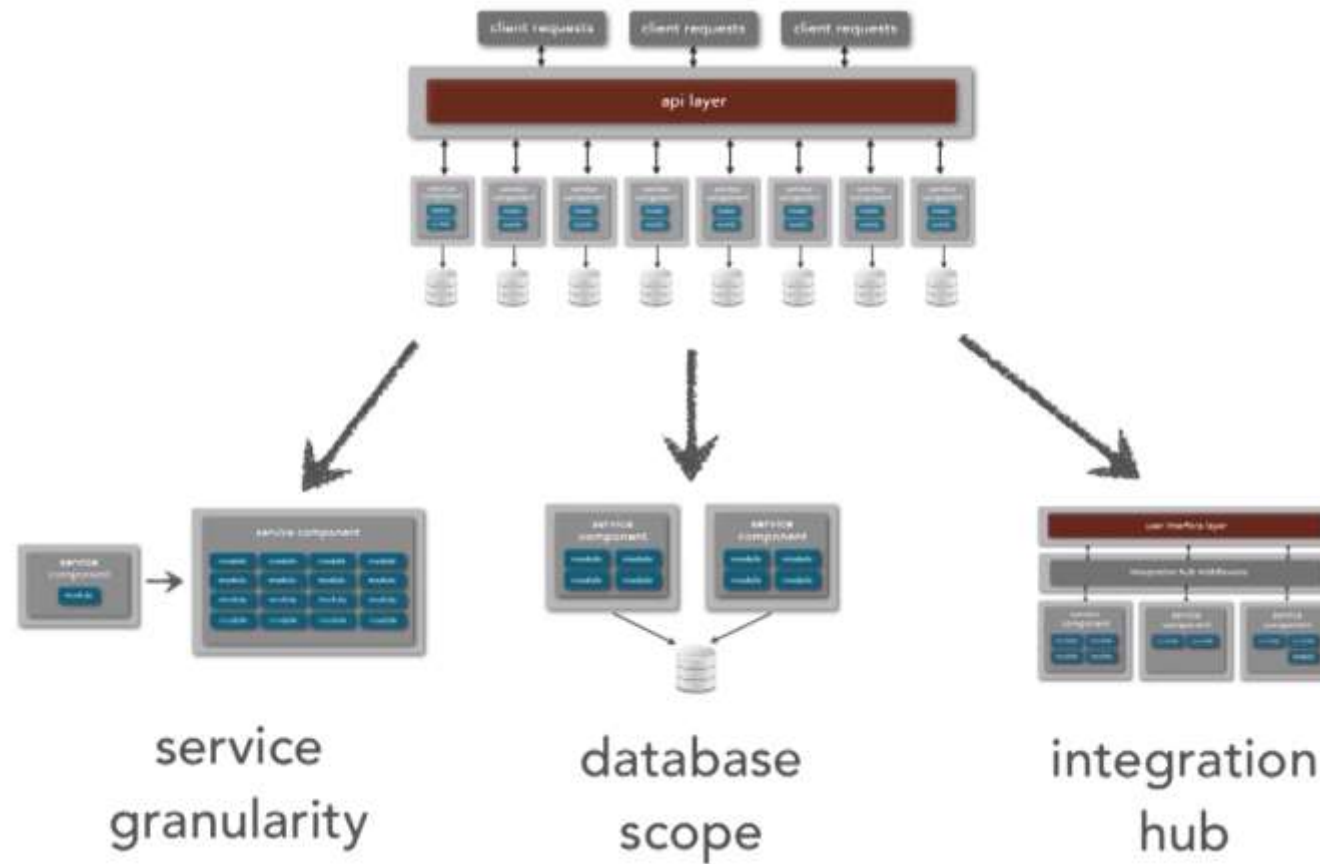
Microservices Architecture Benefits

- **Each microservice is relatively small**
 - Easier for a developer to understand
 - The IDE is faster making developers more productive
 - The web container starts faster, which makes developers more productive, and speeds up deployments
- **Each service can be deployed independently of other services** - easier to deploy new versions of services frequently
- **Easier to scale development.** It enables you to organize the development effort around multiple teams. Each team is responsible a single service. Each team can develop, deploy and scale their service independently of all of the other teams.
- Improved fault isolation. For example, if there is a memory leak in one service then only that service will be affected. The other services will continue to handle requests.
- Eliminates any long-term commitment to a technology stack


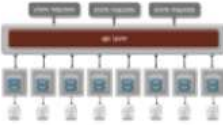

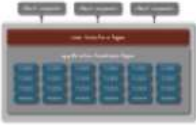
Microservices Architecture Drawbacks

- **Developers must deal with the additional complexity of creating a distributed system.**
 - Developer tools/IDEs are oriented on building monolithic applications and don't provide explicit support for developing distributed applications.
 - Testing is more difficult
 - **Developers must implement the inter-service communication mechanism.**
 - Implementing use cases that span multiple services without using distributed transactions is difficult
 - Implementing use cases that span multiple services requires careful coordination between the teams
- **Deployment complexity.** In production, there is also the operational complexity of deploying and managing a system comprised of many different service types.
- **Increased memory consumption.** The microservices architecture replaces N monolithic application instances with $N \times M$ services instances. If each service runs in its own JVM (or equivalent), which is usually necessary to isolate the instances, then there is the overhead of M times as many JVM runtimes. Moreover, if each service runs on its own VM (e.g. EC2/Azure instance), as is the case at Netflix, the overhead is even higher.

Service-Based Architecture (SBA - SOA and Microservices Hybrid)



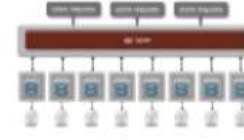
Characteristics Differences

ratings relative to each pattern	 service-oriented architecture	 microservices architecture	 service-based architecture	 monolithic architecture
overall agility	L	H	M	L
deployment	L	H	M	L
testability	L	H	M	M
performance	L	M	M	H
scalability	M	H	M	L
overall simplicity	L	M	M	H

Capabilities Differences



service-oriented
architecture



microservices
architecture



service-based
architecture

location transparency	✓	✓	✓
name transparency	✓	✓	✓
implement transparency	✓	✓	✓
access decoupling	✓	✗	✗*
contract decoupling	✓	✗	✗*

Service Differences



service-oriented
architecture



microservices
architecture



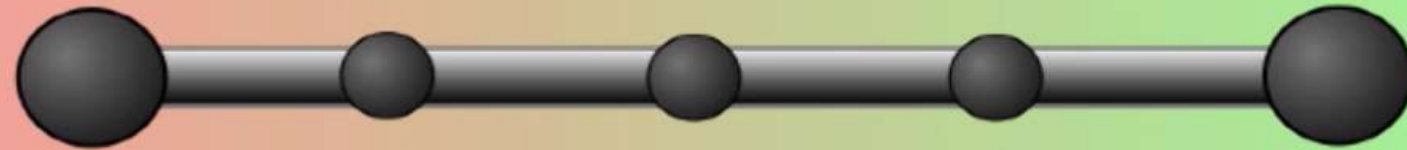
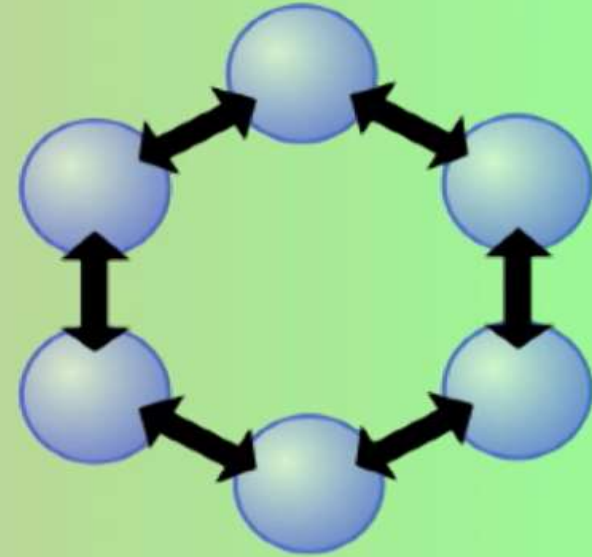
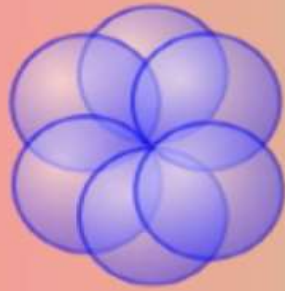
service-based
architecture

service categorization	yes	no	no
service granularity	small to very large	very small	small to medium
service orchestration	yes	minimal	generally no
service numbers	hundreds	thousands	dozens

Coupling And Cohesion

Coupling Types in Software Architecture

- Content Coupling (Pathological Coupling)
 - One module modifies or relies on the internal implementation of another module
- Common Coupling (Global coupling)
 - Two modules share the same global data (e.g., a global variable)
- External Coupling
 - Two modules share an externally imposed data format, communication protocol or device interface
- Control Coupling
 - One module controlling the flow of another, by passing it an information on what to do (e.g., passing what-to-do flags/codes)
- Stamp Coupling (Data-structured Coupling)
 - Modules share a composite data structure and use only a part of it, possibly a different part (e.g., passing the whole JSON structure to a function that only uses specific fields from it)
- Data Coupling
 - Data sharing through parameters, for example. Each peace piece of information is elementary and the only shared data (like SOAP)
- Message Coupling
 - Data is shared through the message-based exchange (like REST)



Content

Common

Control

Stamp

Data

Tight

Loose

More interdependency
More coordination
More information flow

Less interdependency
Less coordination
Less information flow

Component Coupling Calculation

$$C = 1 - \frac{1}{d_i + 2c_i + d_o + 2c_o + f_o + f_i}$$

d_i = number of input data parameters

c_i = number of input data control flow parameters

d_o = number of output data parameters

c_o = number of output data control flow parameters

f_o = number of components called from A (fan-out)

f_i = number of components calling A (fan-in)

$C = 0.7$ (low coupling) to 1.0 (high coupling)