MICROSERVICE ARCHITECTURE AND MESSAGE QUEUES

ARTURO CALVO
Trondheim, July 12th 2018
hello@arturocalvo.com / @artucalvo

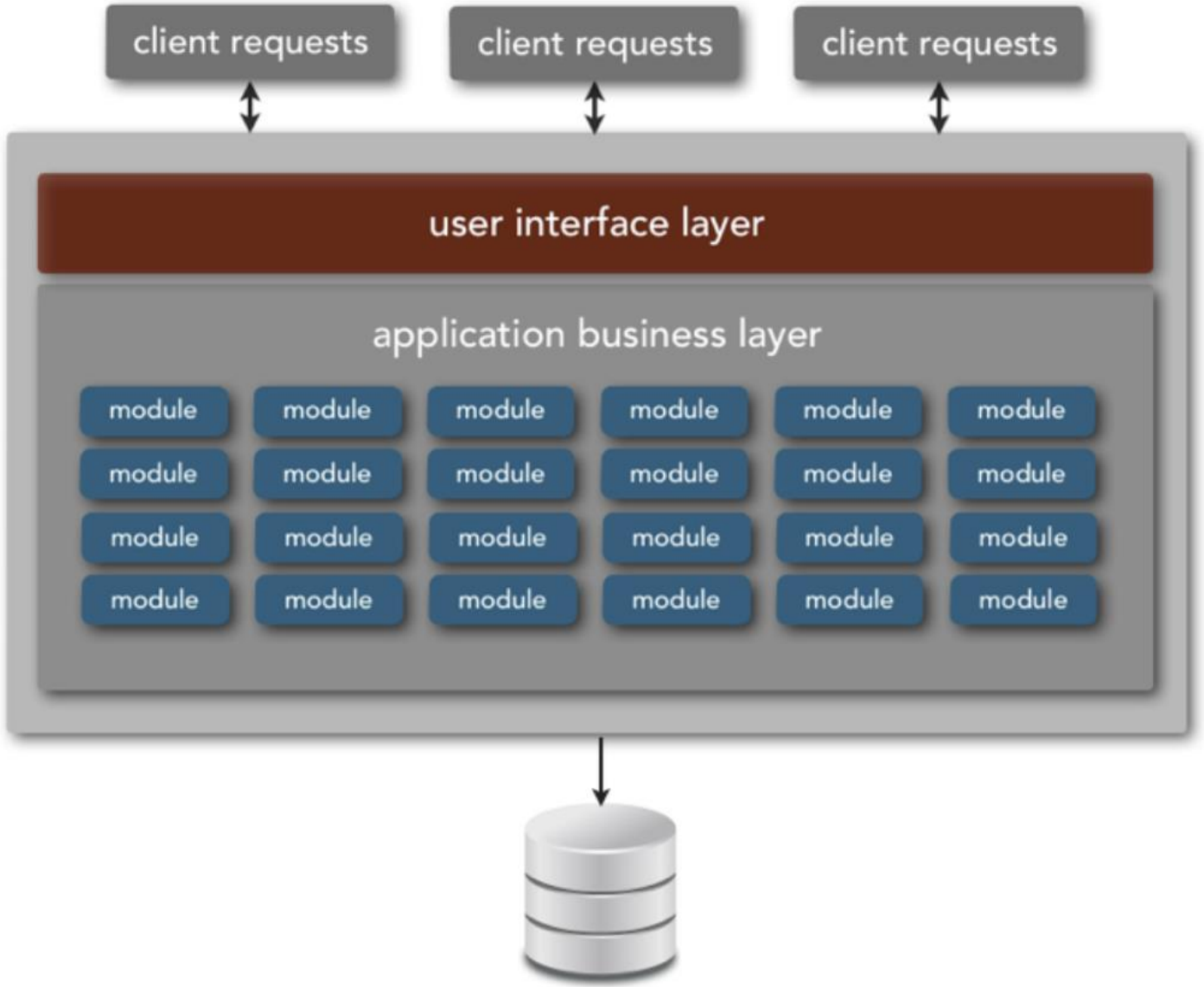
accenture digital


449.000
120
accenture digital

AGENDA

1. Monolithic Architecture
2. What is a Microservice?
3. Characteristics of Microservices
4. Service-Based Architecture
5. Trade-offs of architecture styles
6. Communication via REST
7. Communication via Message Queues
8. Message Broker comparision
9. DEMO

MONOLITHIC ARCHITECTURE



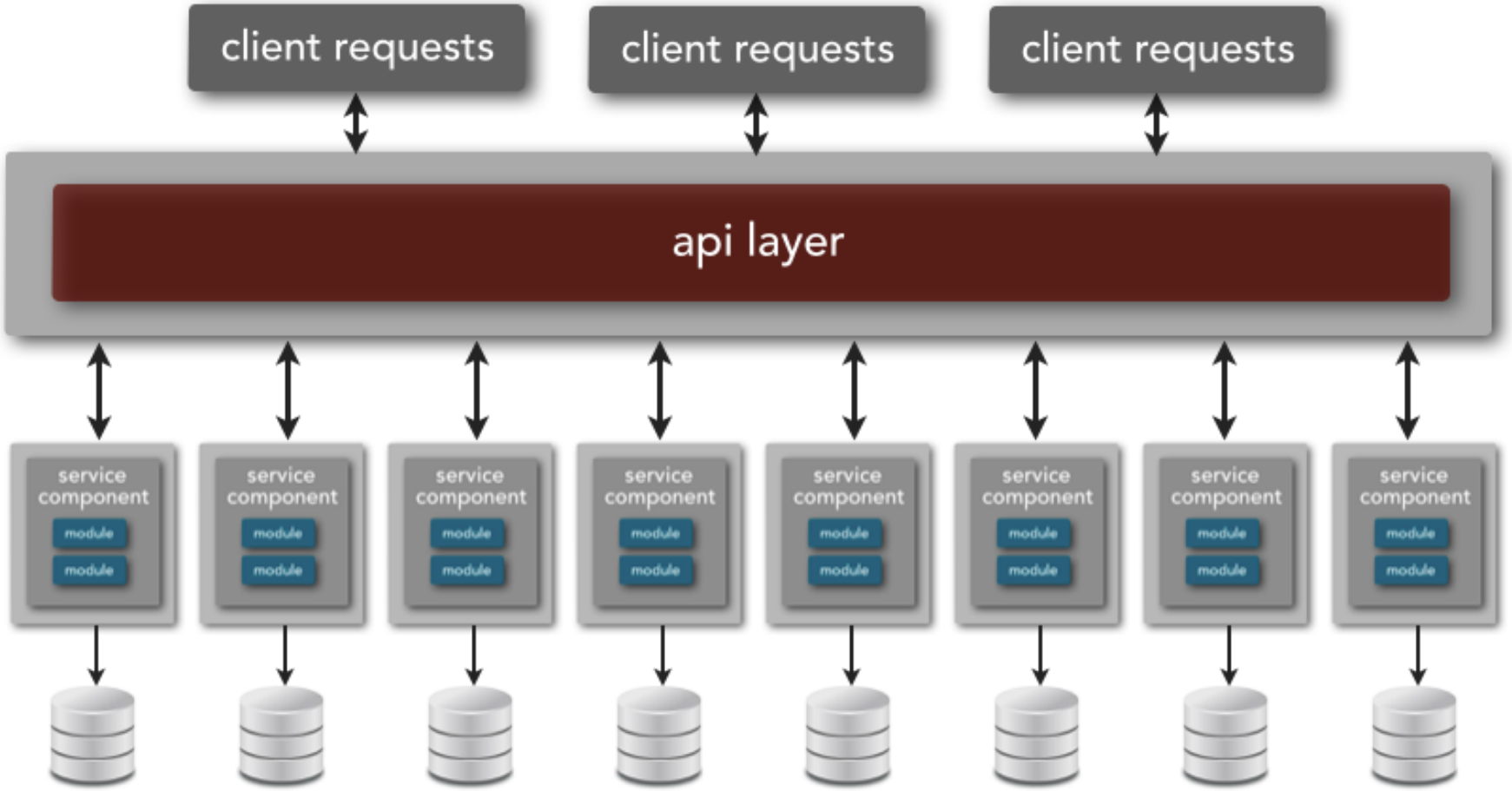
WHAT IS A MICROSERVICE?

*"The microservice architectural style is an approach to developing a single application as a suite of **small services**, each running in its **own process** and communicating with **lightweight** mechanism, often an HTTP resource API".*

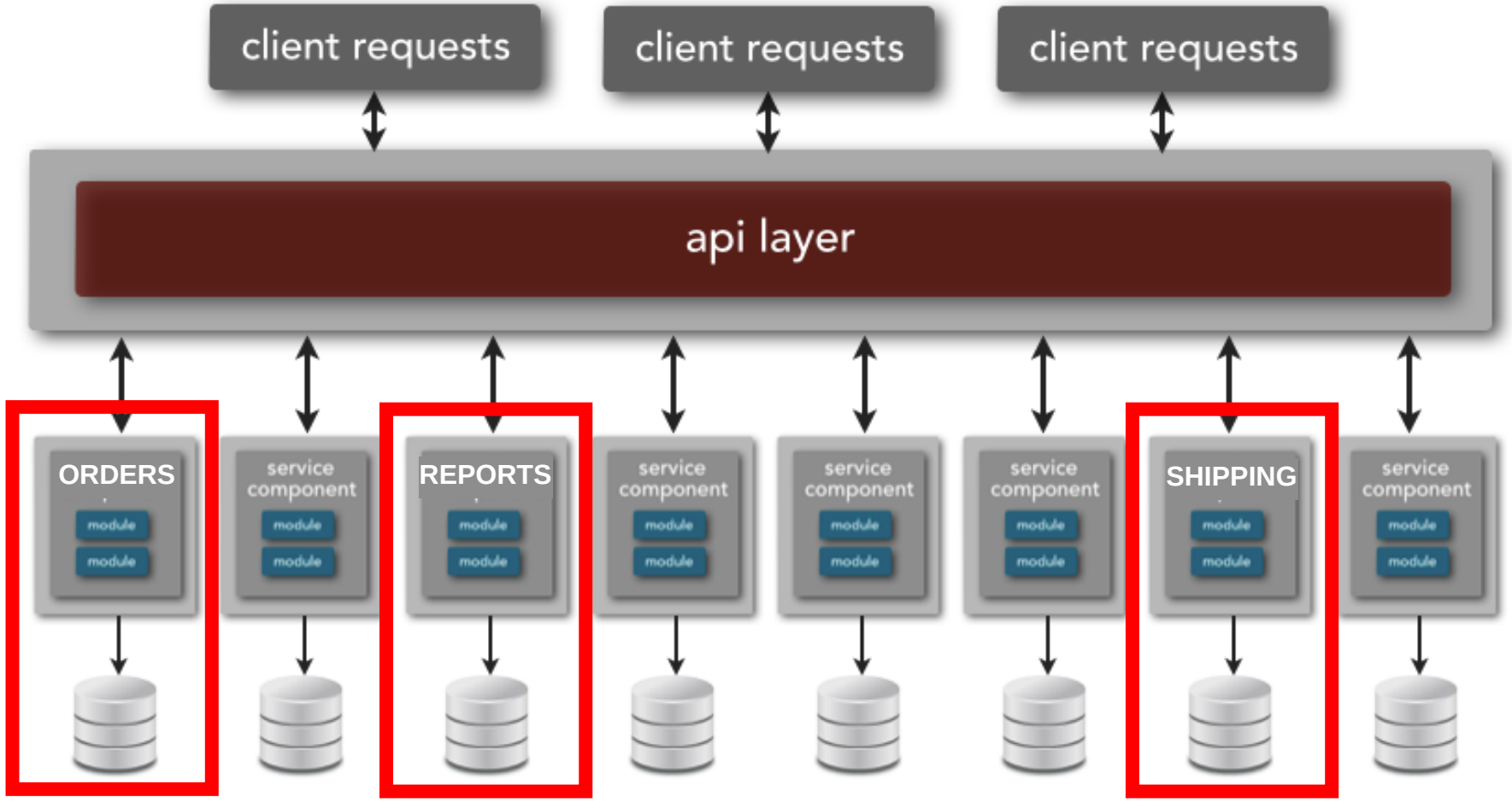
Martin Fowler

*"A **domain-centric** service-based architecture with modern **DevOps** practices".*

CHARACTERISTICS OF MICROSERVICES

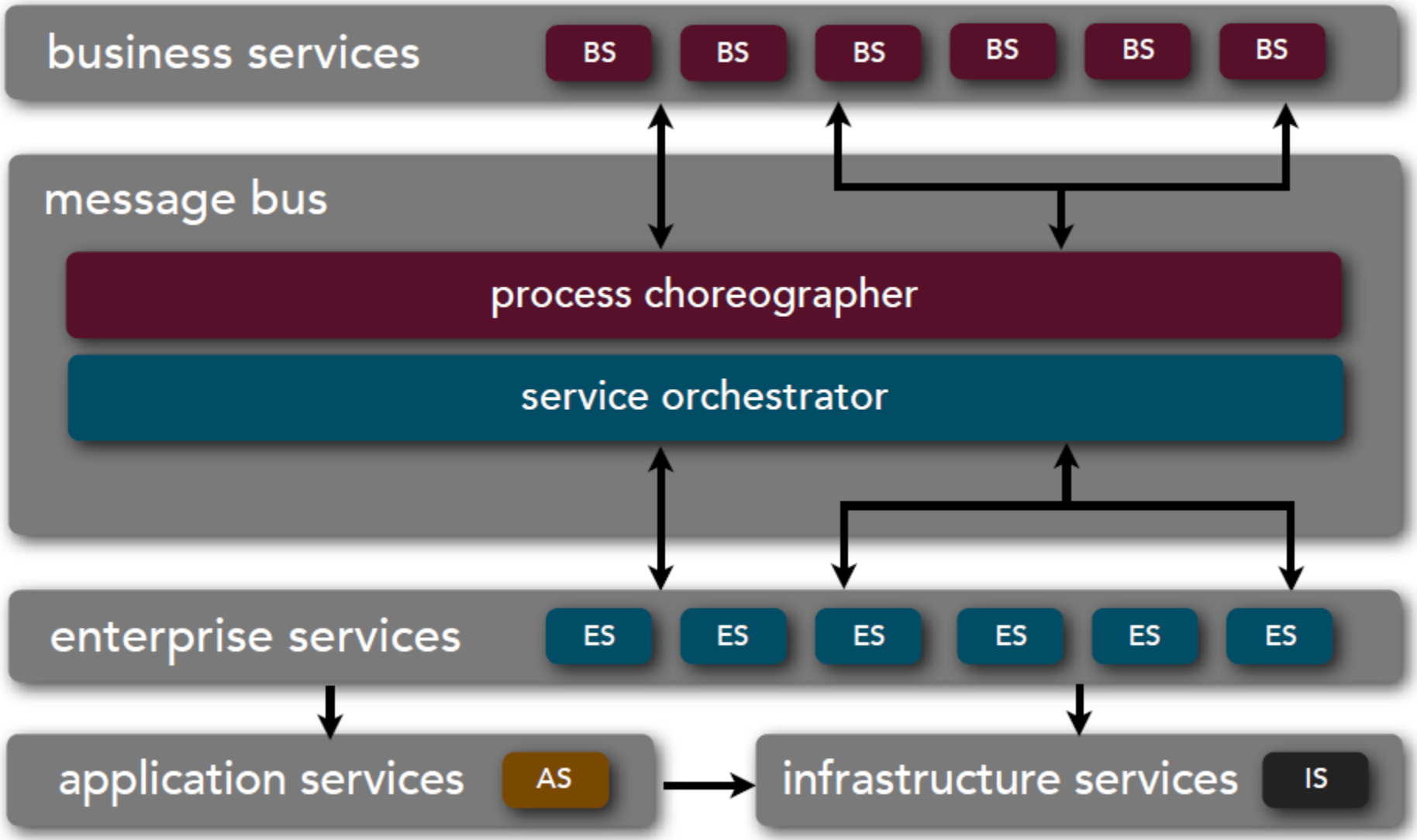


CHARACTERISTICS OF MICROSERVICES

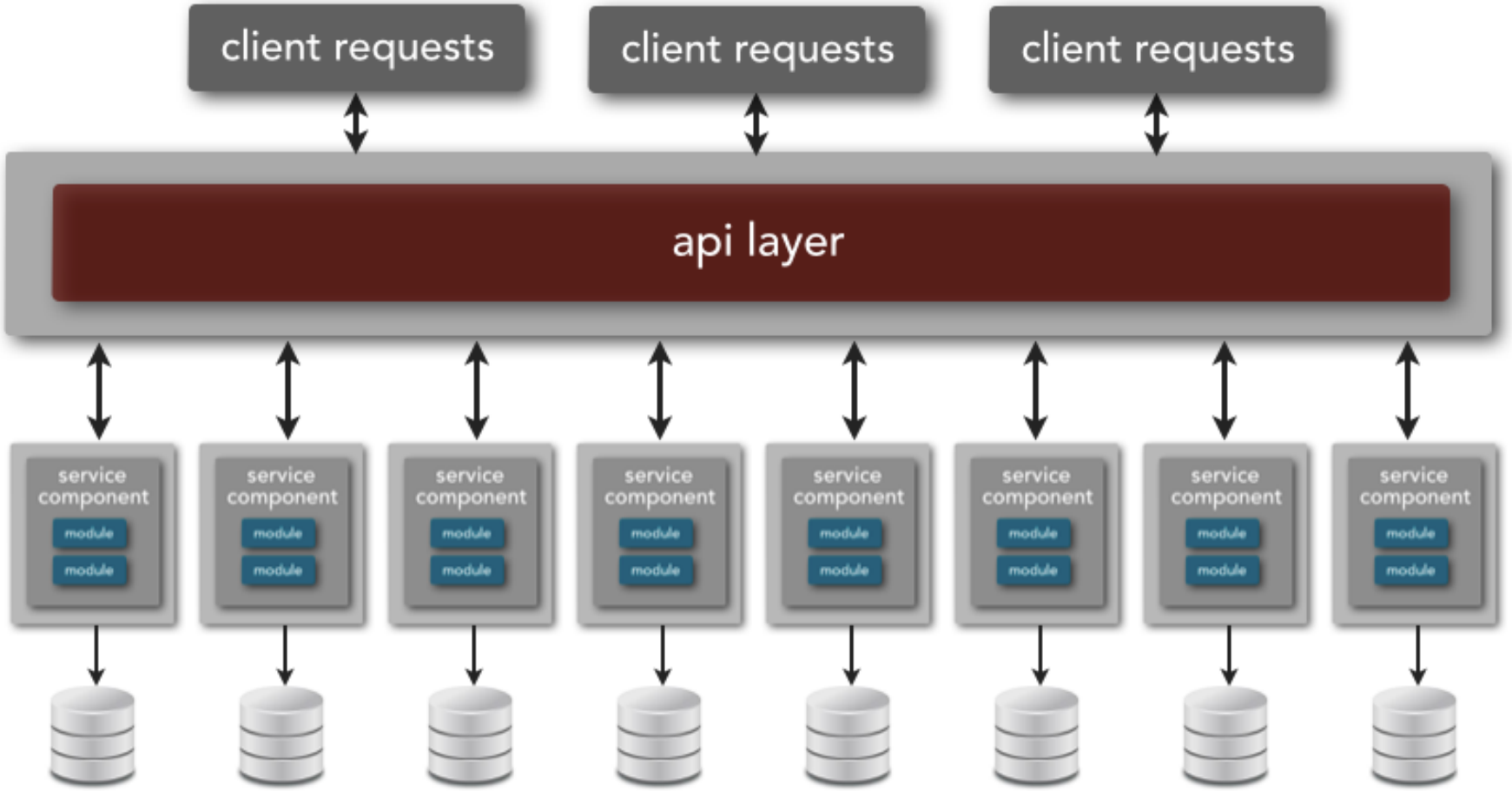


- Modularity and encapsulation into services, not libraries
- Organised around business capabilities vs infrastructure

SERVICE-ORIENTED ARCHITECTURE

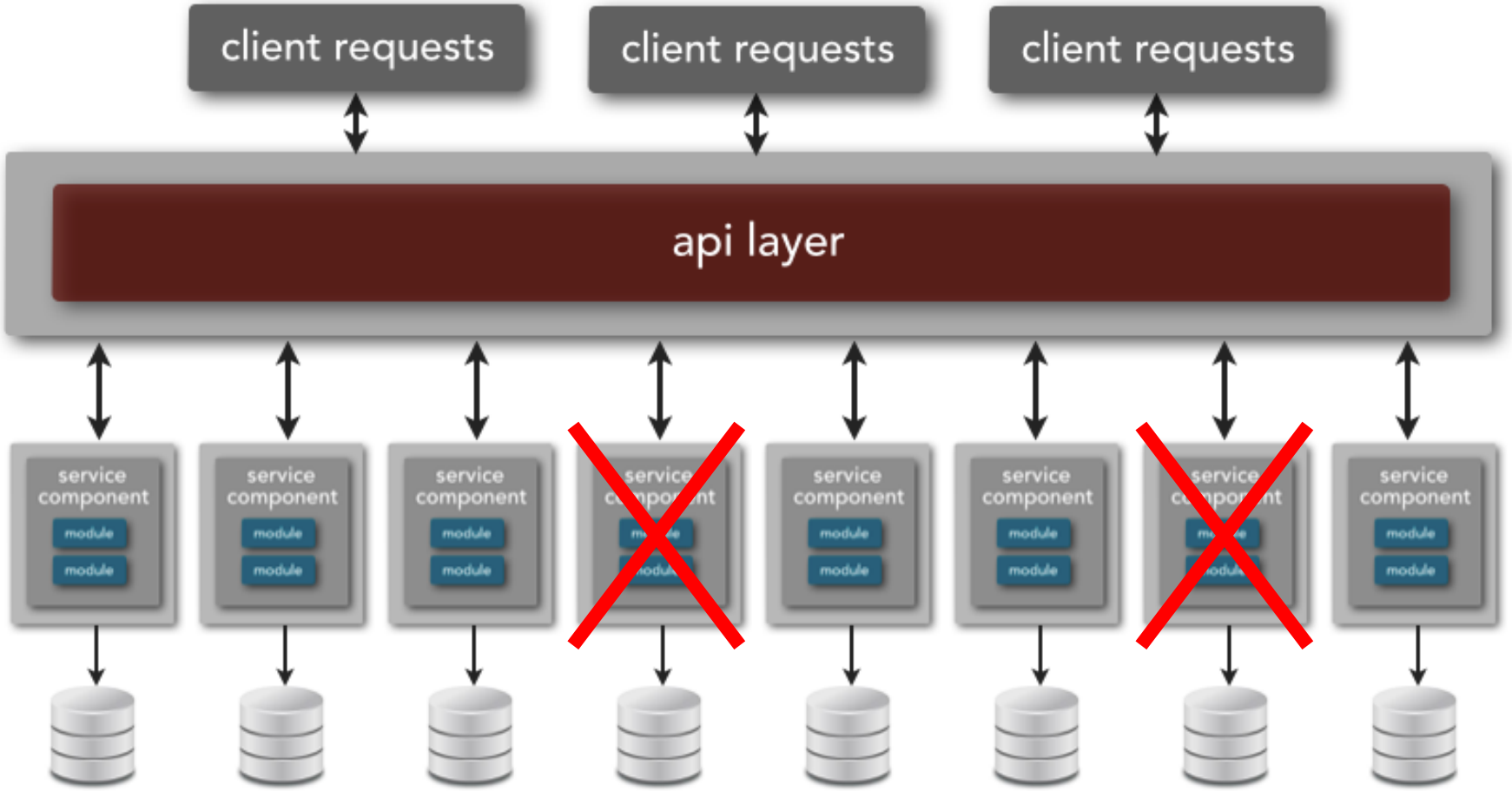


CHARACTERISTICS OF MICROSERVICES



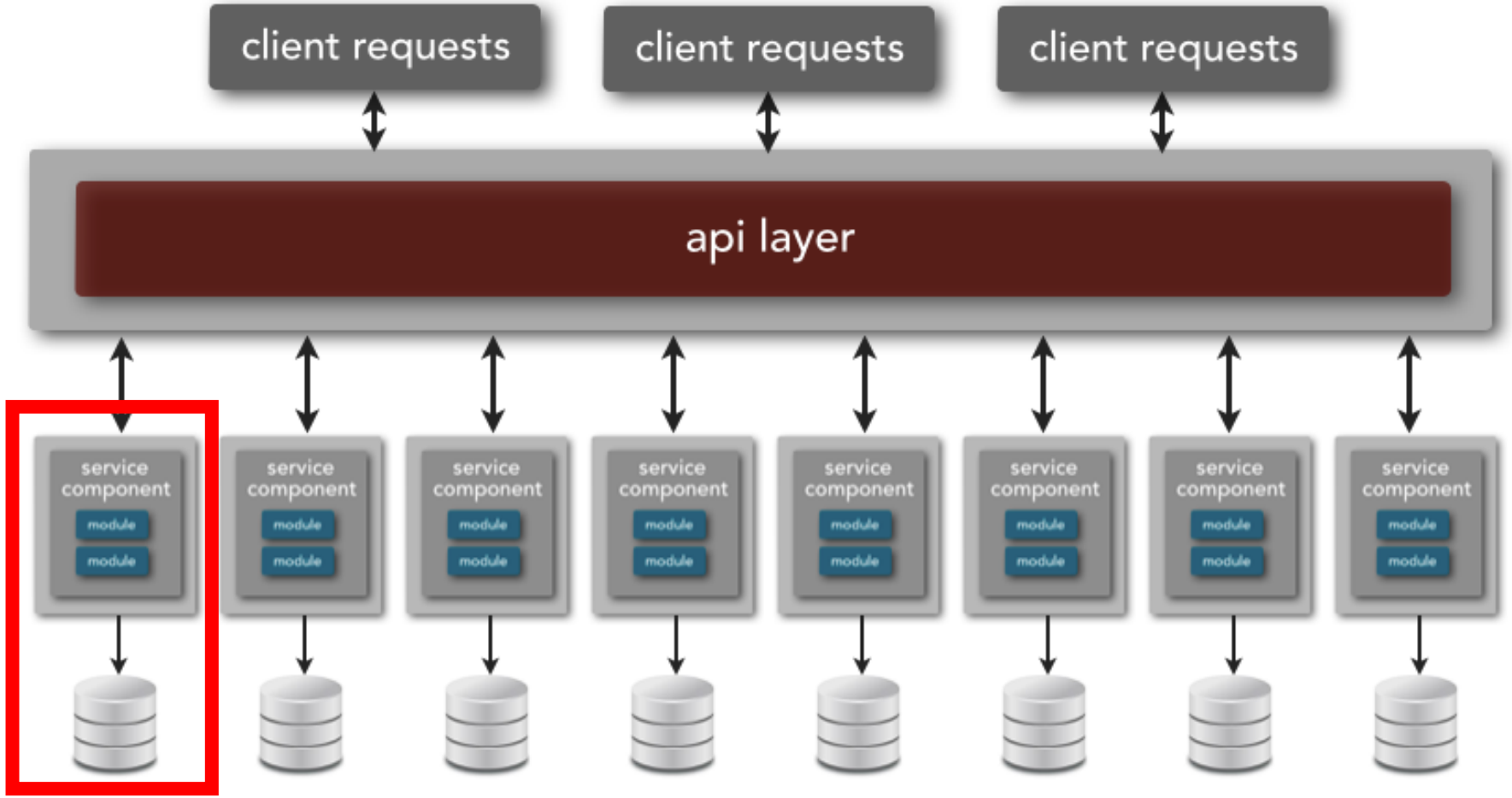
- Smart endpoints and dumb pipes

CHARACTERISTICS OF MICROSERVICES



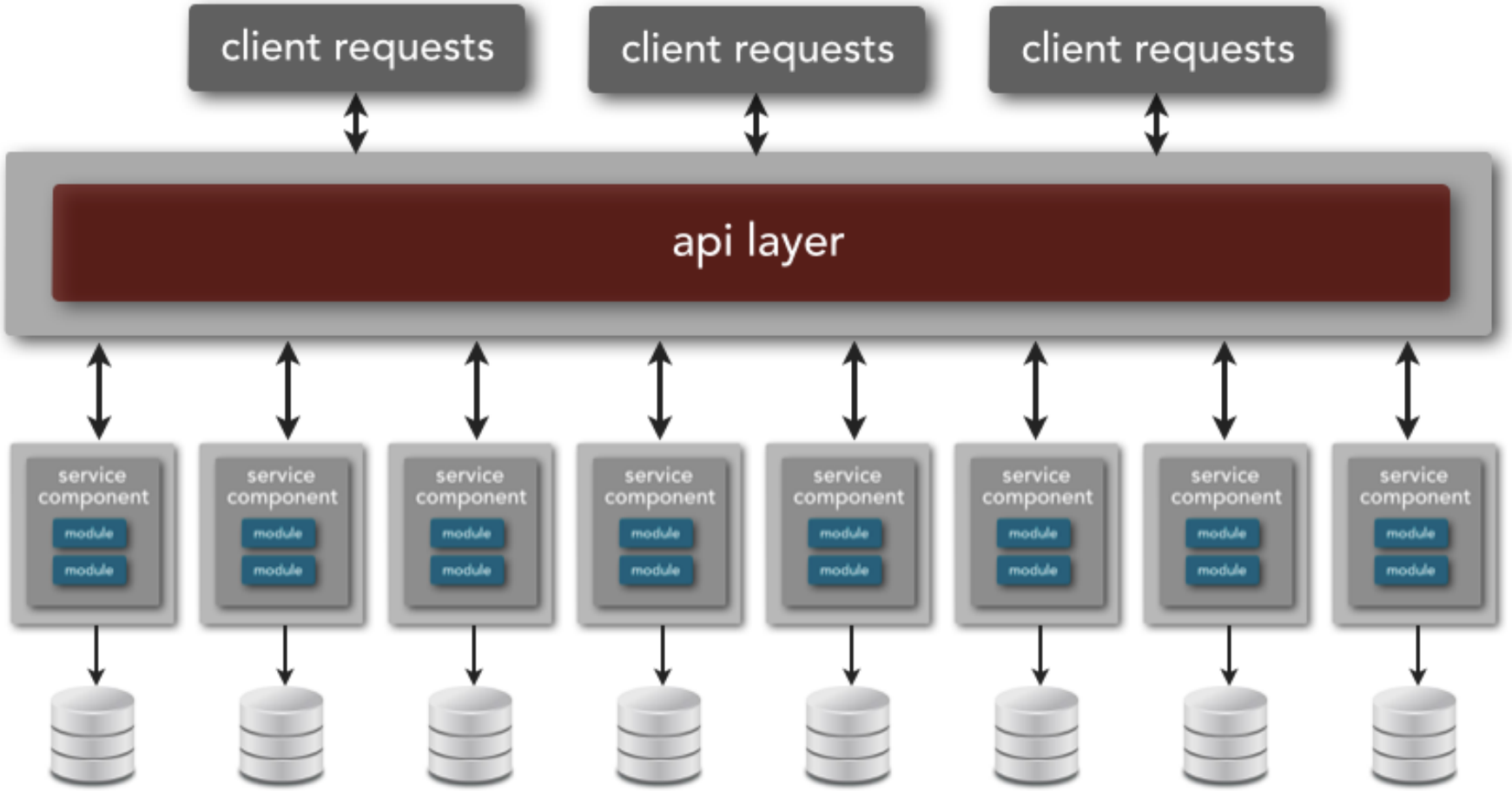
- Higher availability due to fault isolation

CHARACTERISTICS OF MICROSERVICES



- Services with bounded context and private database
- Decentralized governance and data management

CHARACTERISTICS OF MICROSERVICES



- 100s of independent and distributed service components

HOW SMALL SHOULD THEY BE?

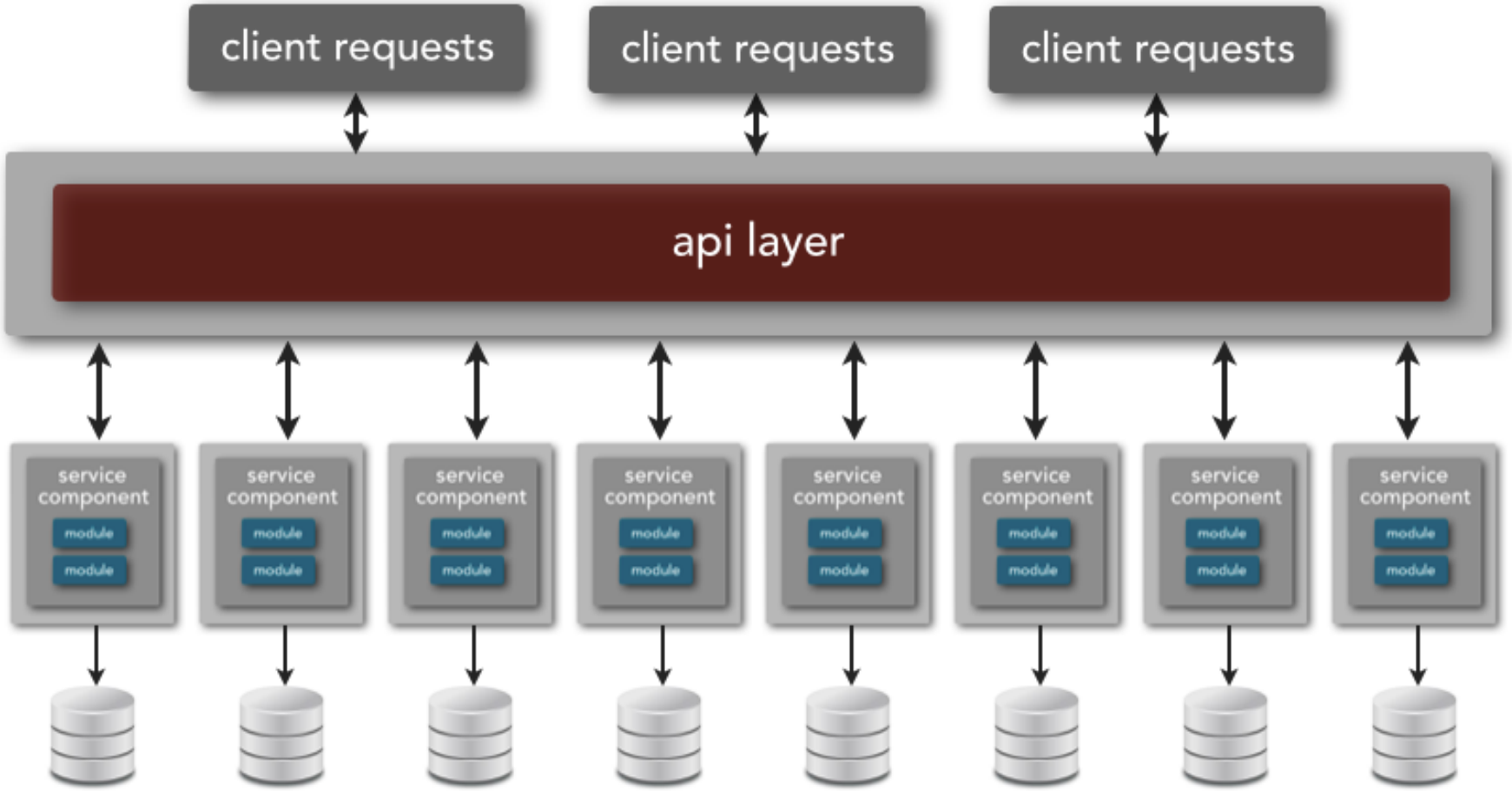
“Applications that fit in your head”.

James Lewis

“Start out more coarse-grained and move to fine-grained as you learn more about the service”.

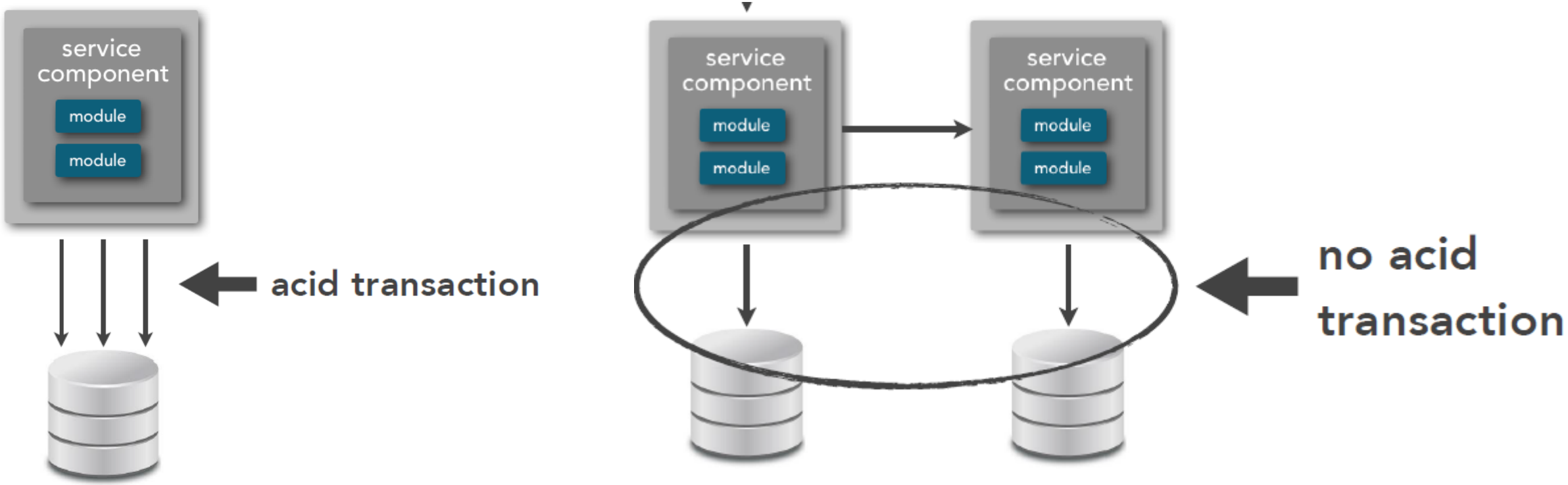
Sam Newman

CHARACTERISTICS OF MICROSERVICES



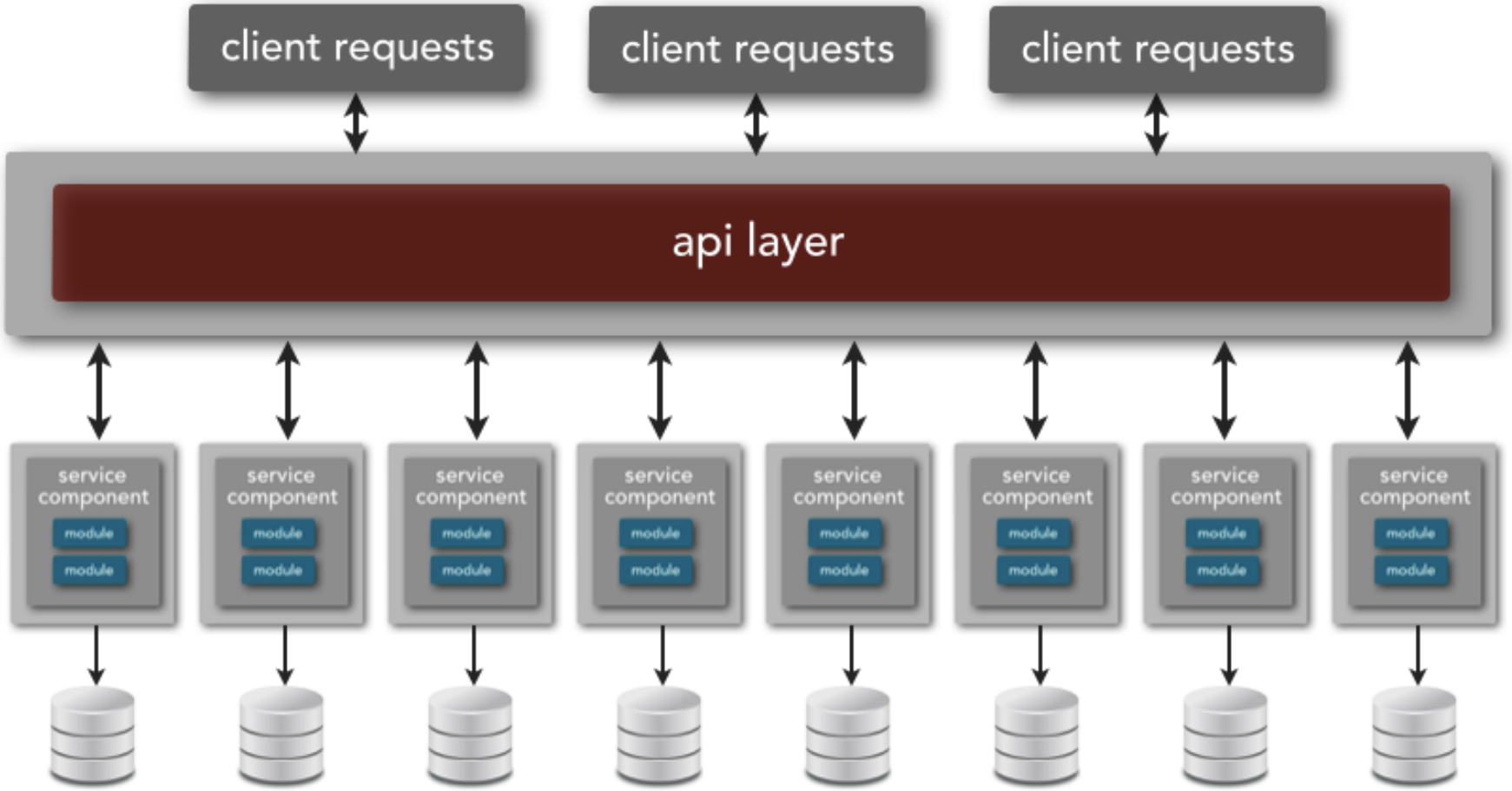
- Granularity: single purpose, choreography feasible and no ACID transactions

CHARACTERISTICS OF MICROSERVICES



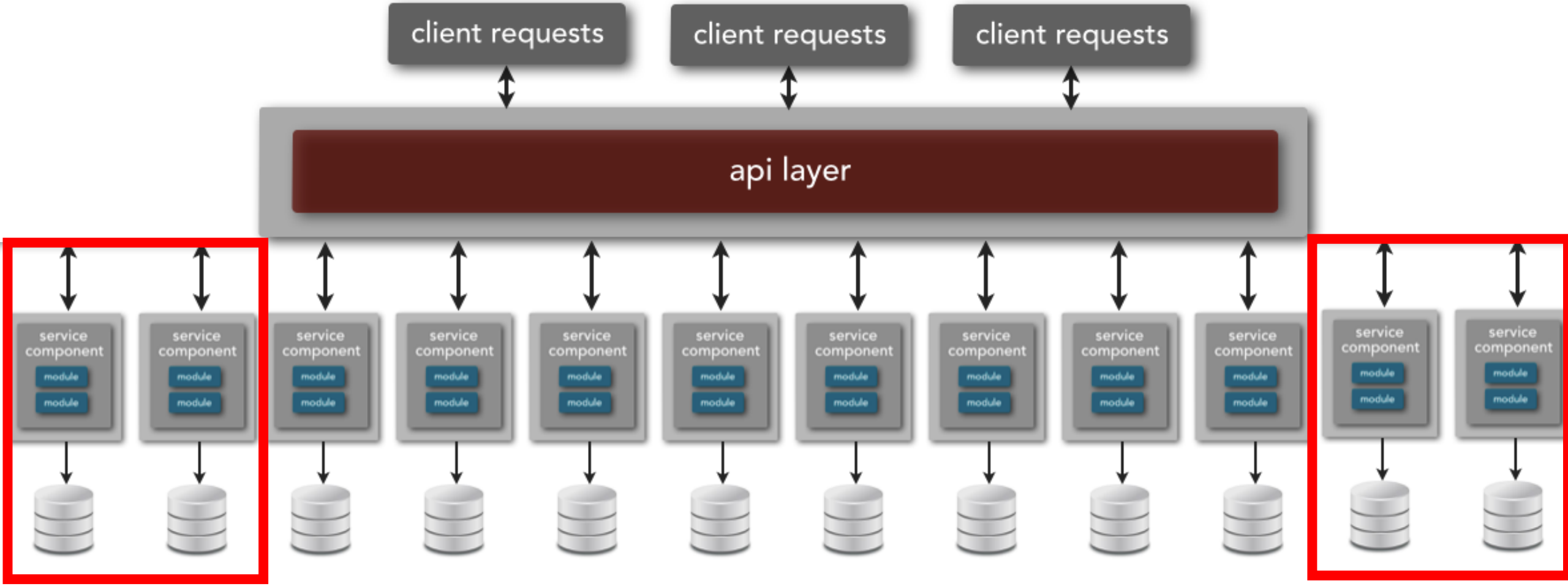
- ACID - atomicity, consistency, isolation, durability
- BASE - basic availability, soft state, eventual consistency

CHARACTERISTICS OF MICROSERVICES



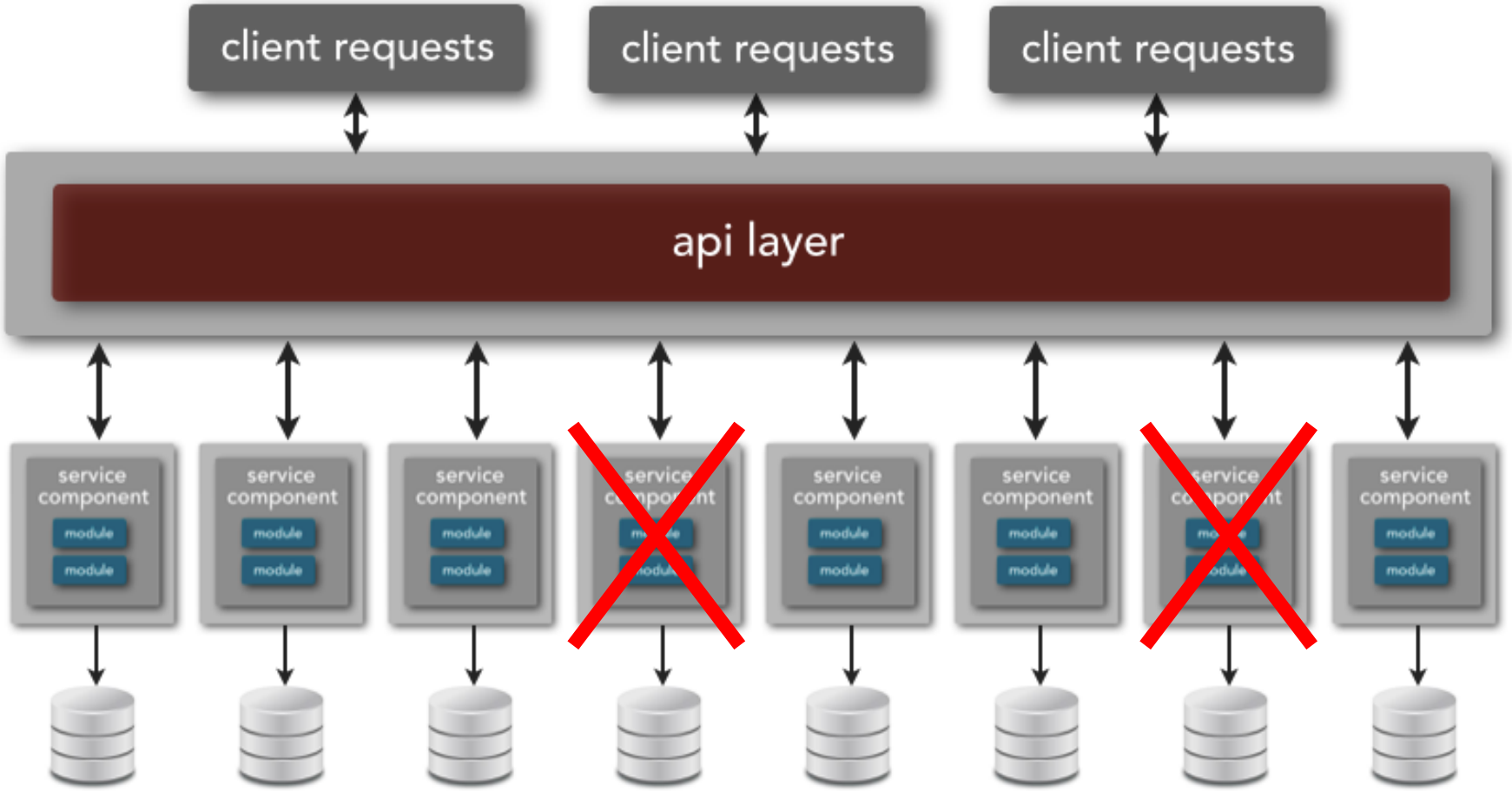
- Easier to develop: multi-language and multi-platform
- Easier to maintain: updated and deployed independently

CHARACTERISTICS OF MICROSERVICES



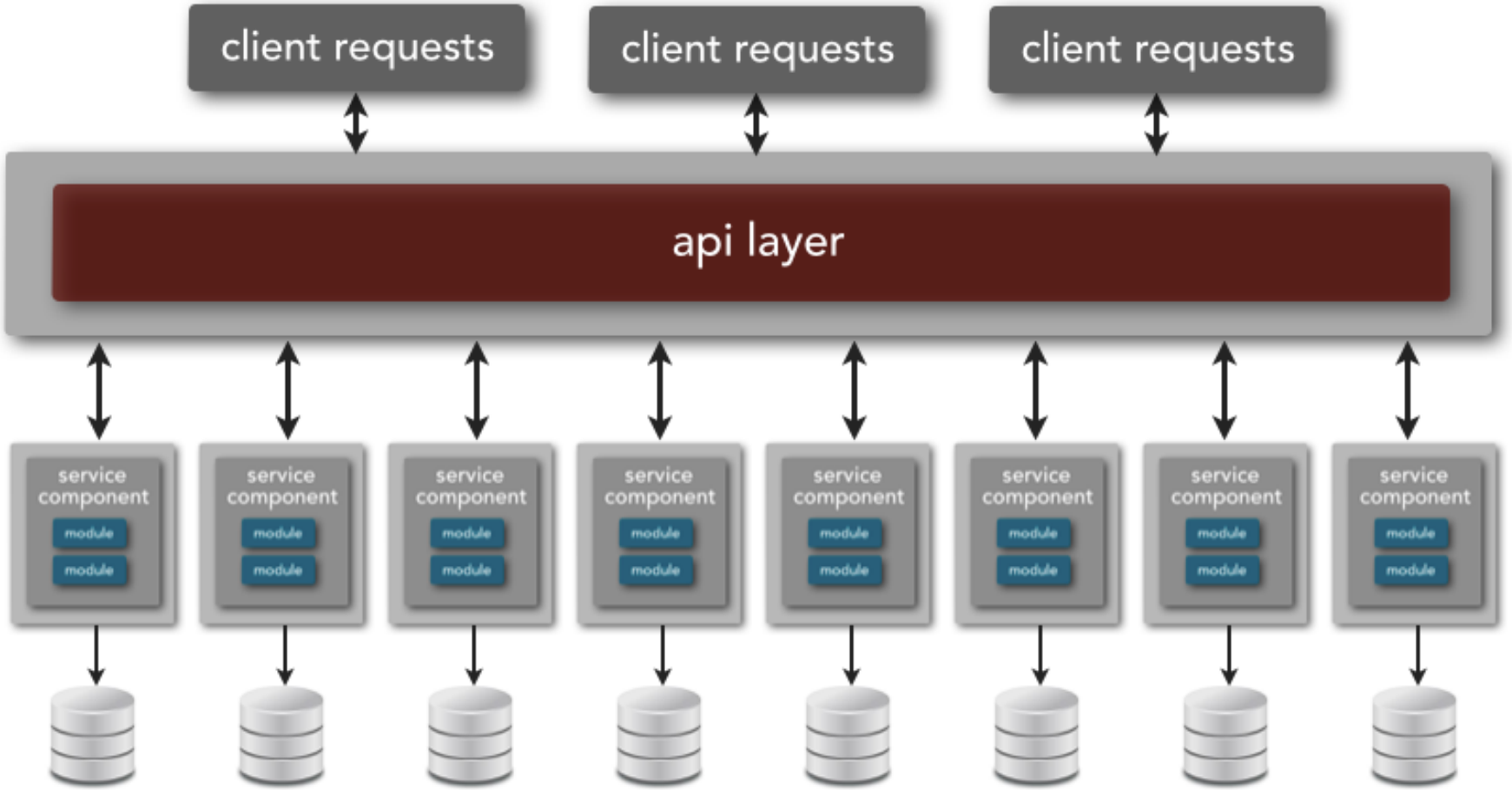
- Horizontal scaling, workload partitioning and reusability

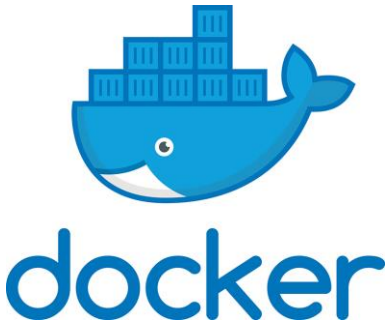
CHARACTERISTICS OF MICROSERVICES



- Designed for failure: monitor and recover

CHARACTERISTICS OF MICROSERVICES

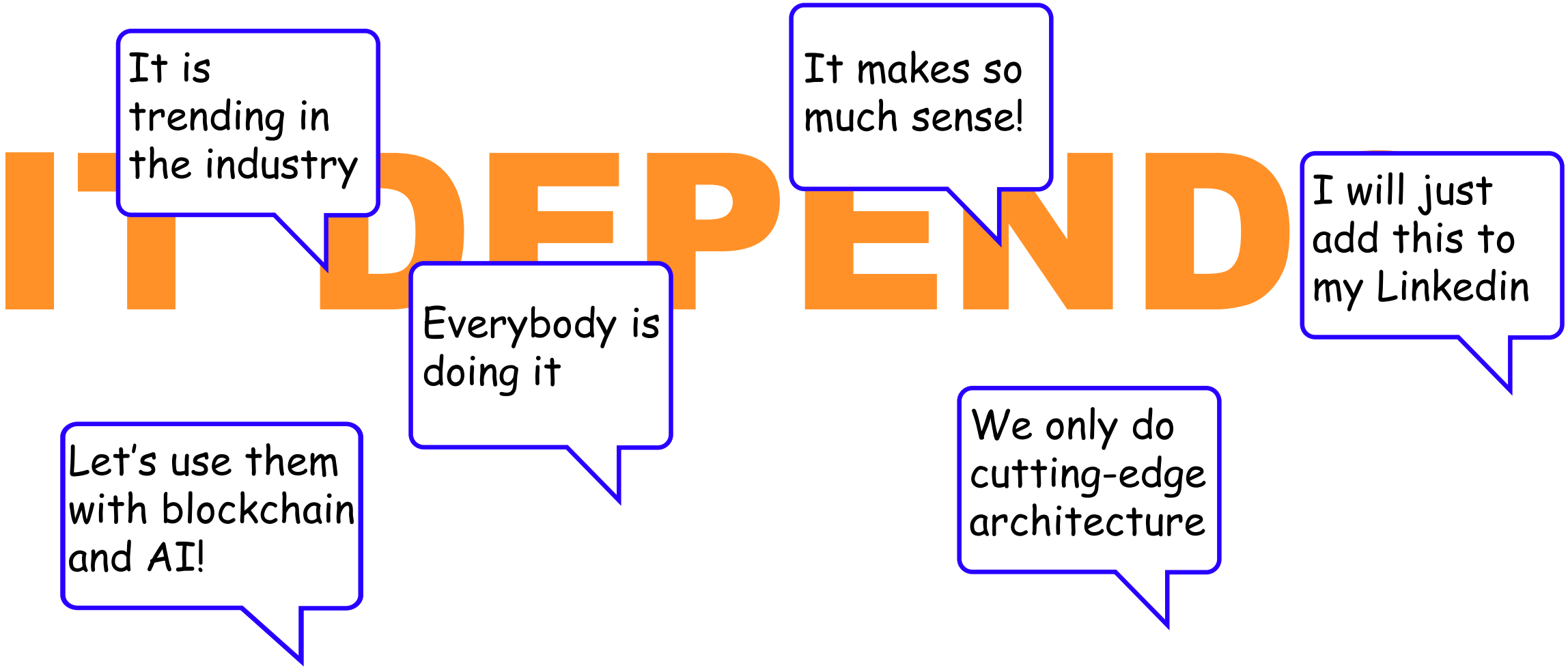




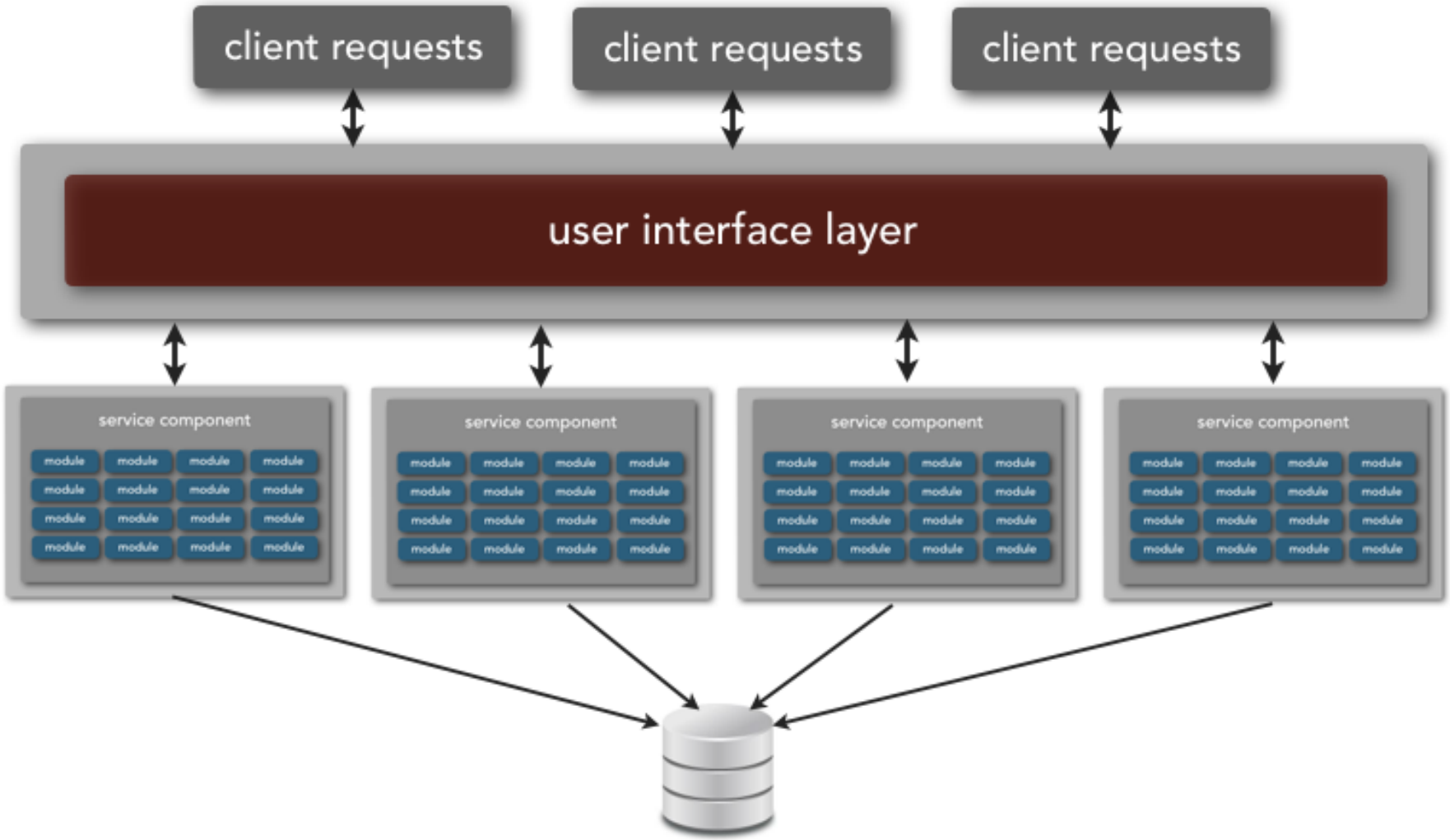
kubernetes

- Devops culture: automate all the infrastructure from compile & build to tests

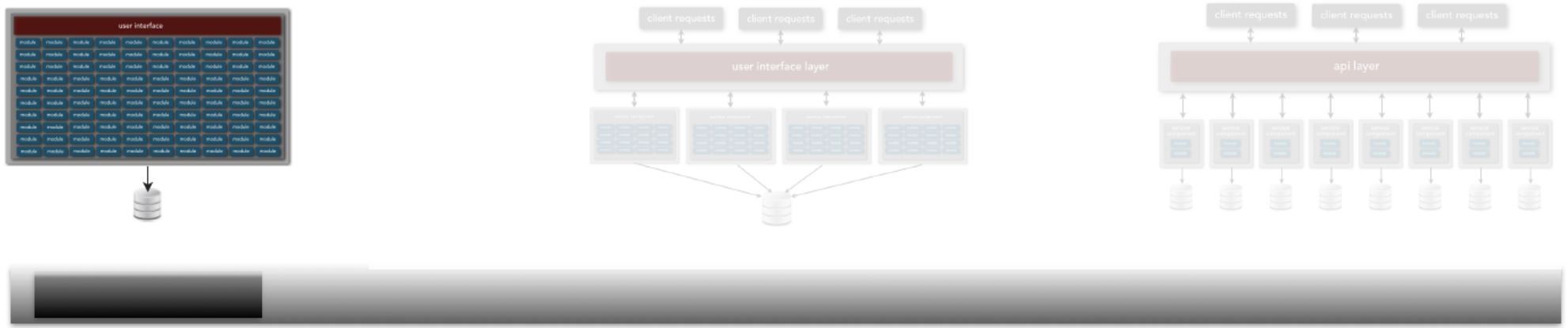
SO... SHOULD I USE MICROSERVICES?

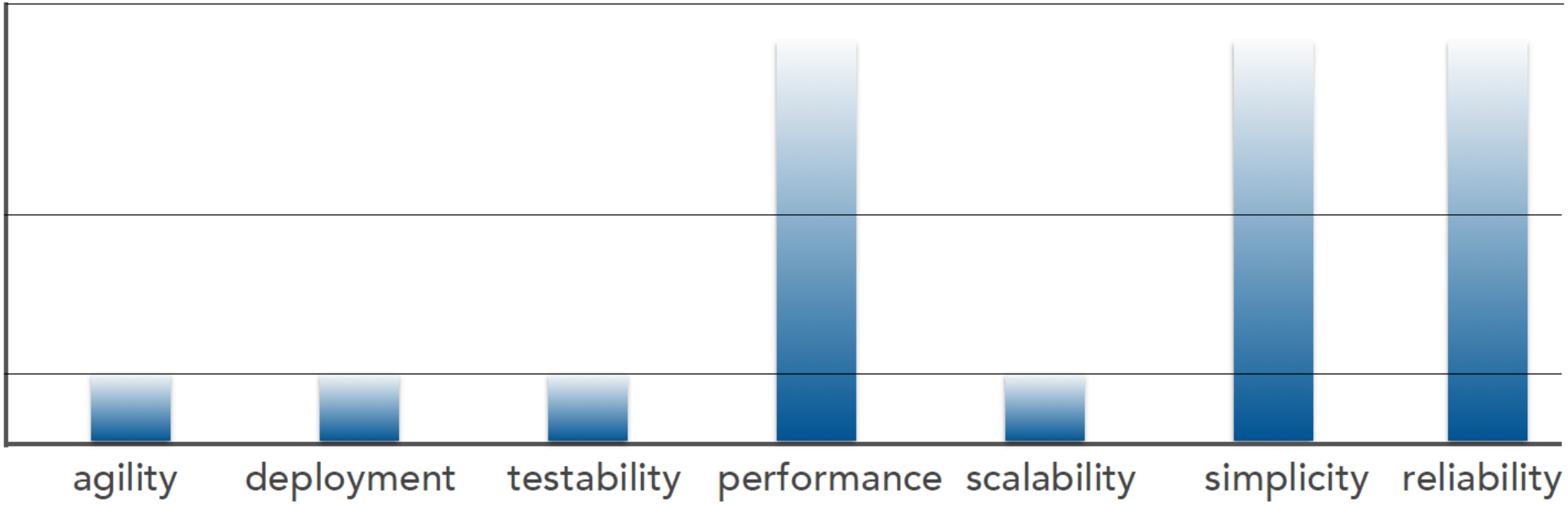


SERVICE-BASED ARCHITECTURE

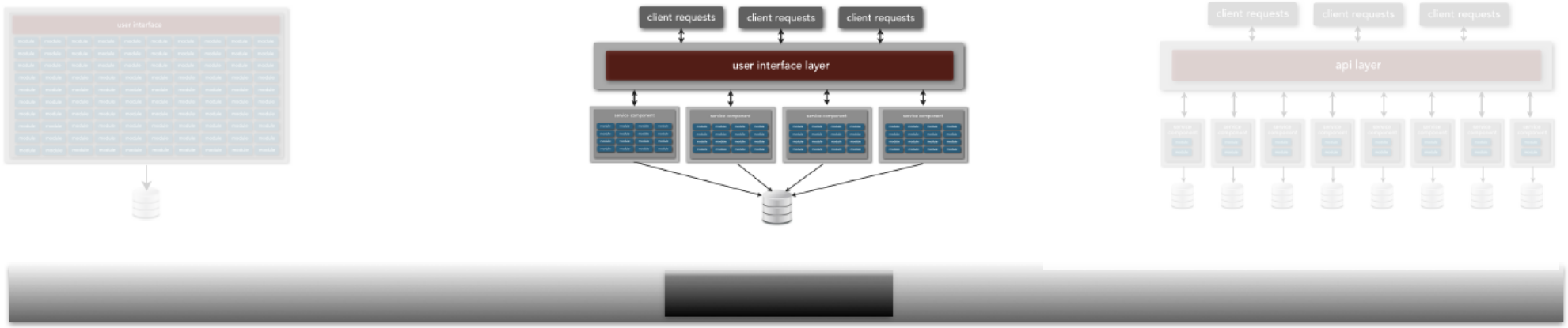


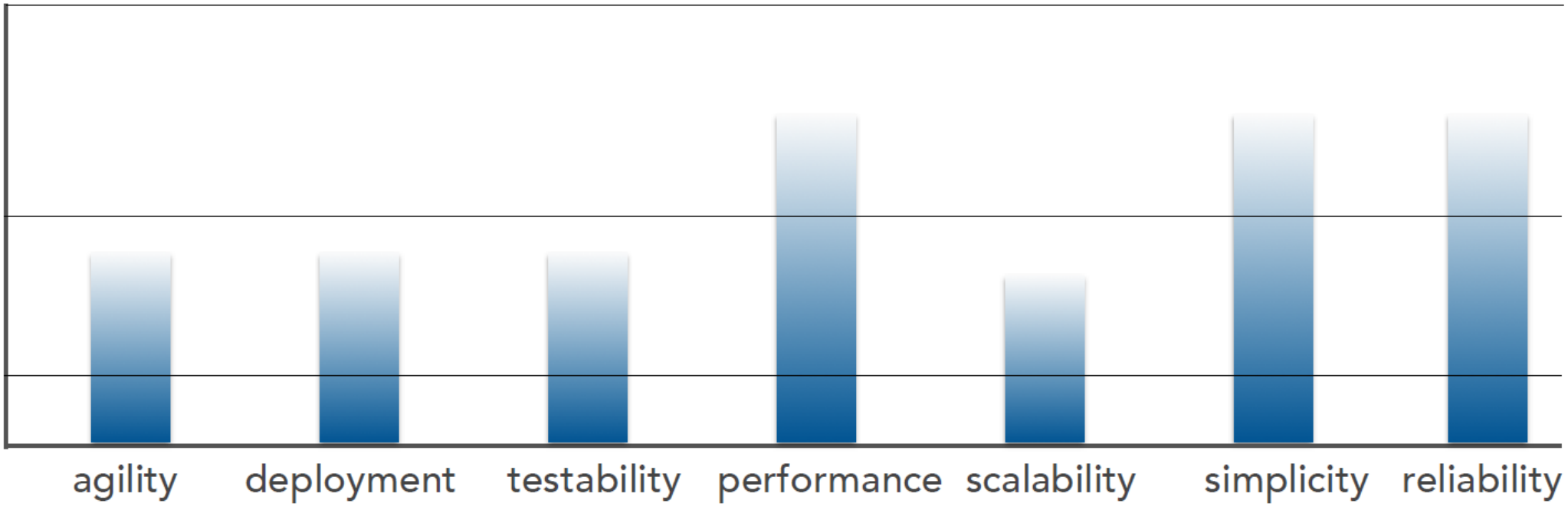
TRADE-OFFS: MONOLITHIC



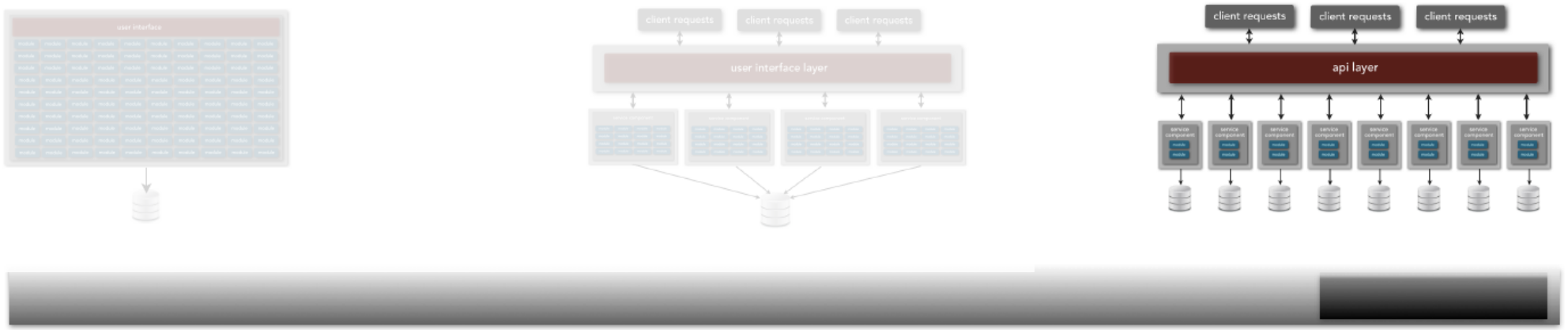


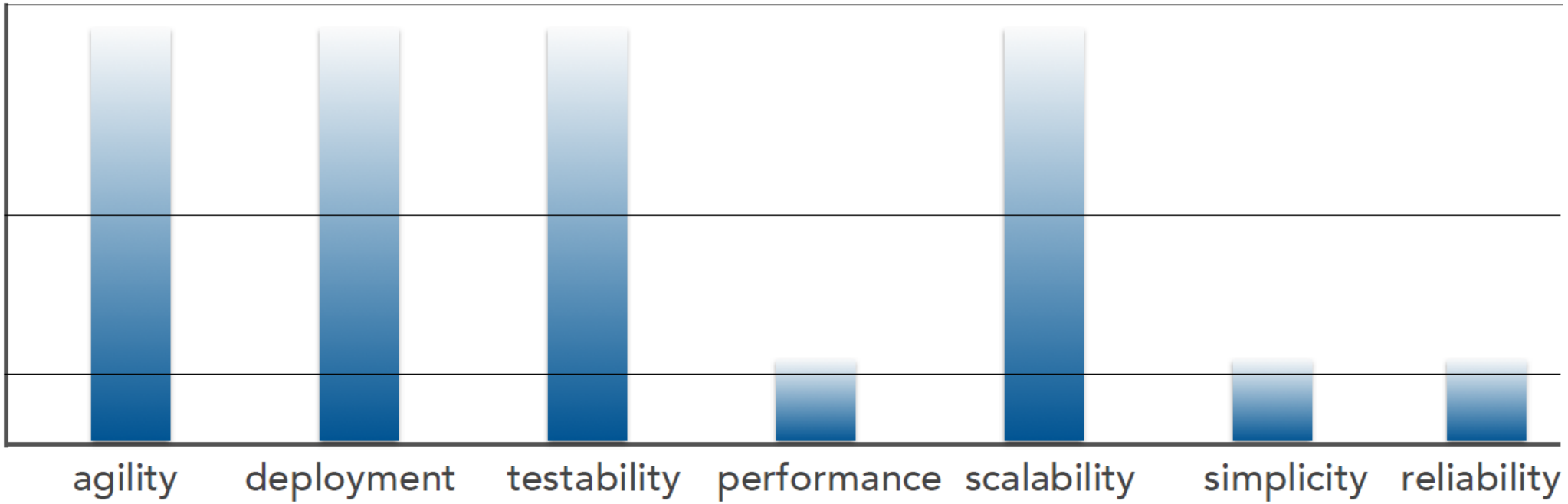
TRADE-OFFS: SBA



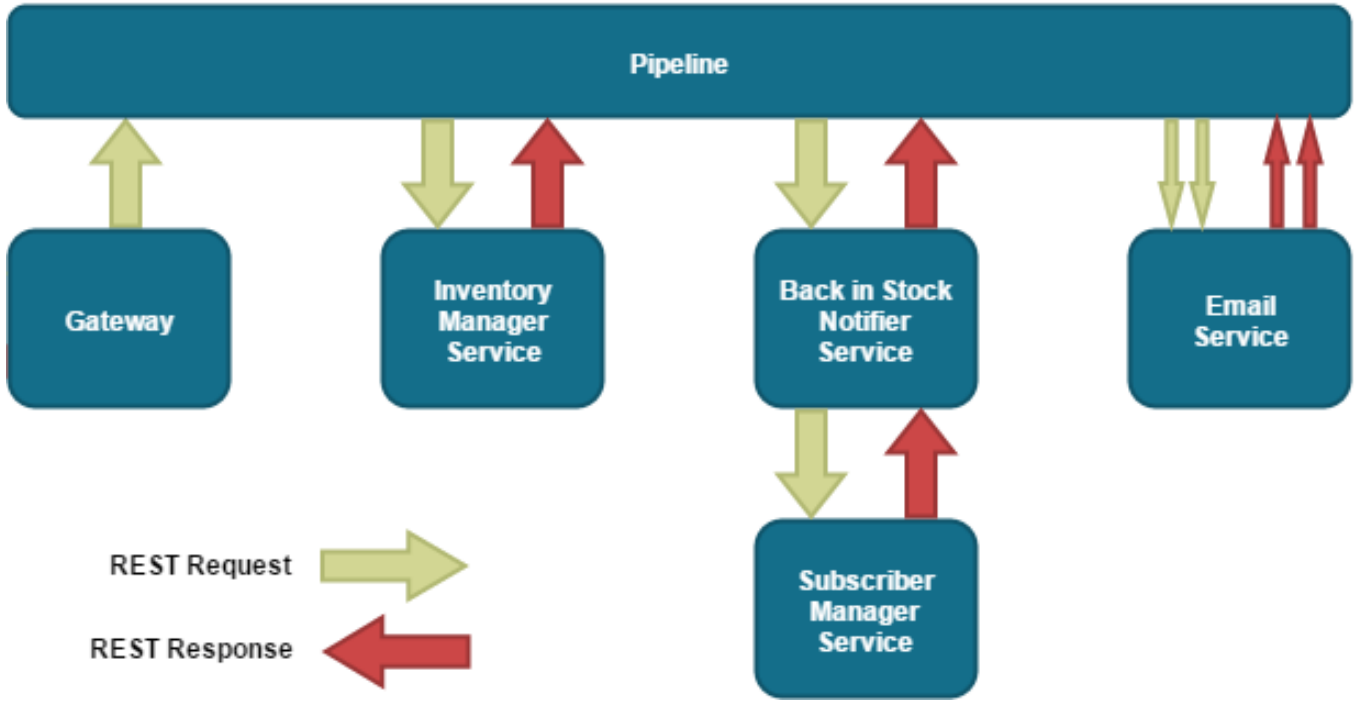


TRADE-OFFS: MICROSERVICES



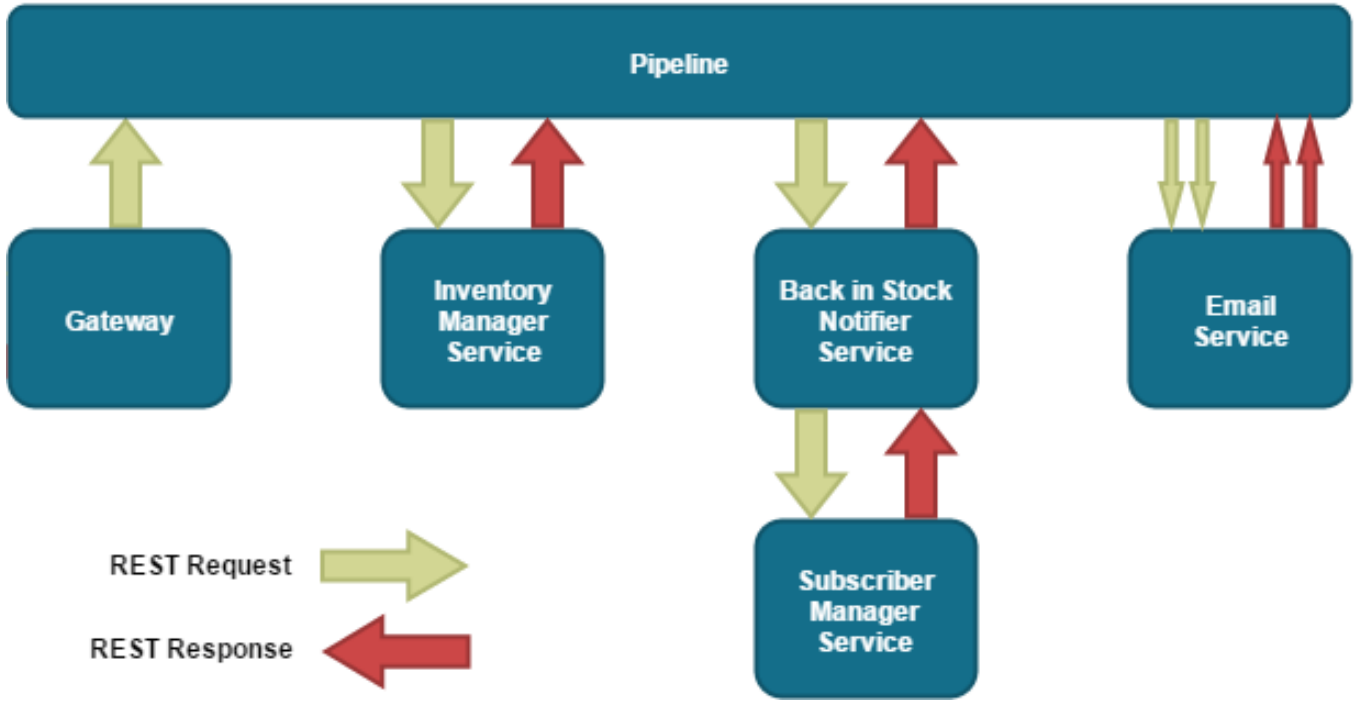


COMMUNICATION VIA REST



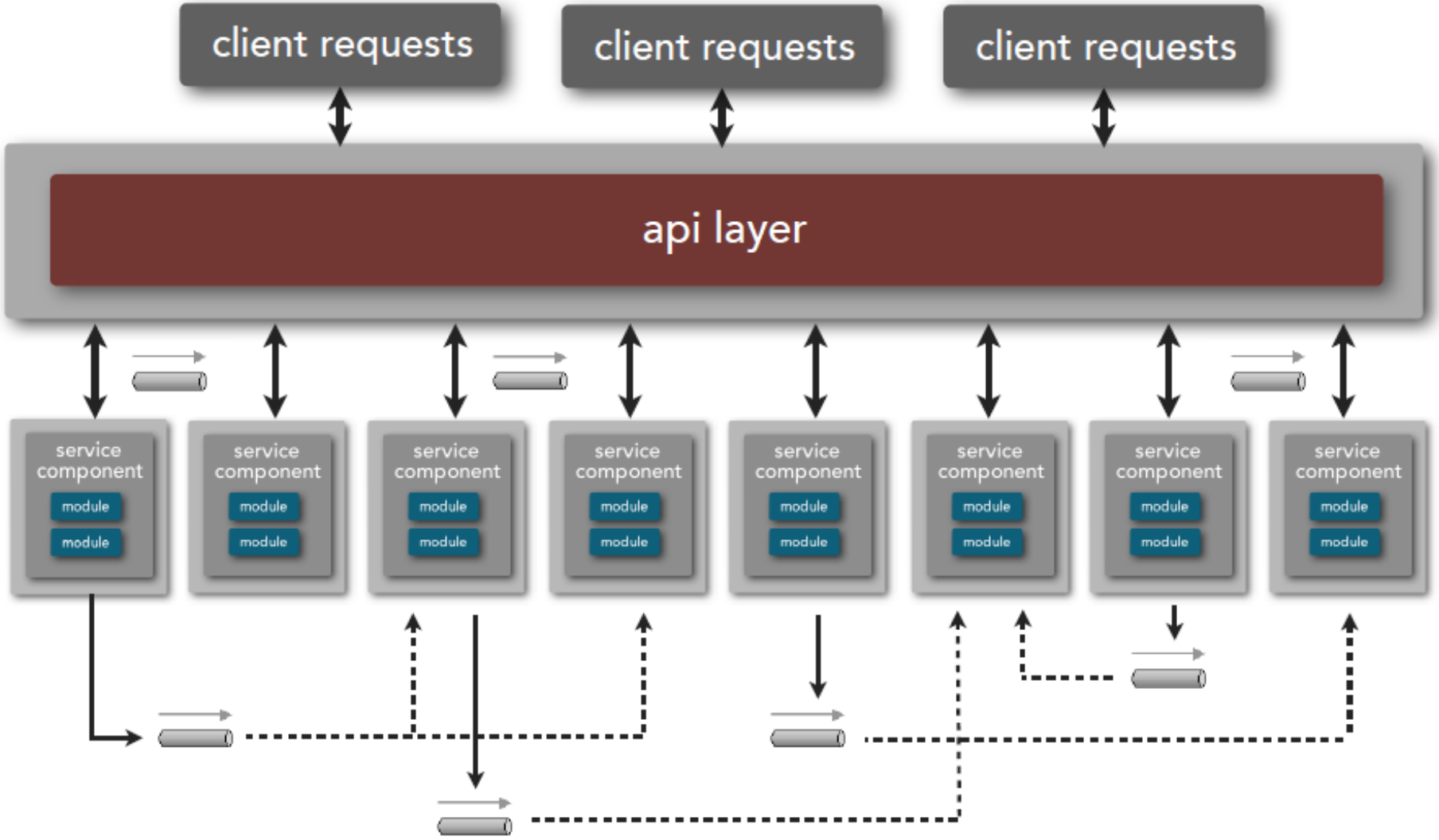
- No need for any additional infrastructure as it uses **HTTP** protocol
- Direct and **synchronous** communication between services
- Server does not respond anything until completed, which can be a problem with **time-consuming** operations
- It can handle **asynchronous** calls too

COMMUNICATION VIA REST



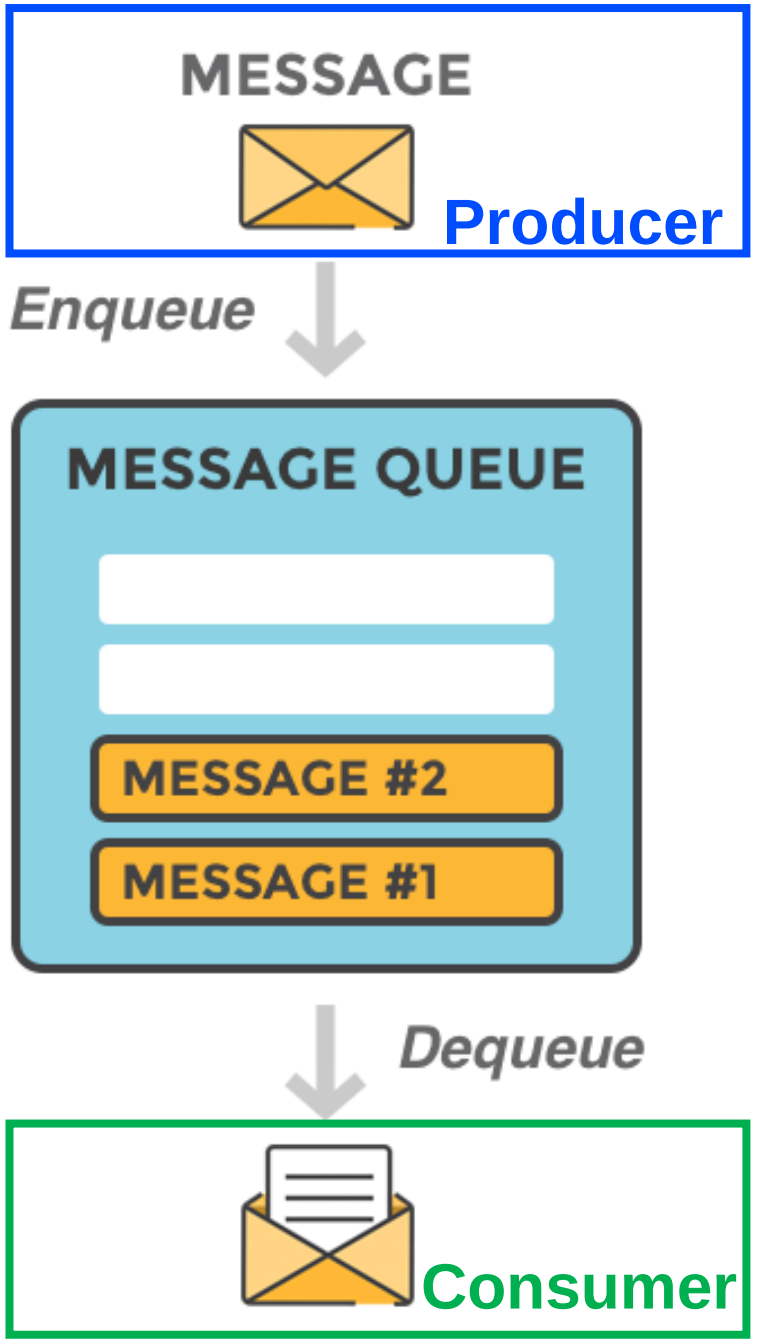
- **Tightly-coupled** services
- **Service Discovery** via Kubernetes, DNS, Eureka,...
- Client handles **server failure** and retries when it is back
- If server responds but **client fails**, the operation is lost

COMMUNICATION VIA MESSAGE QUEUES



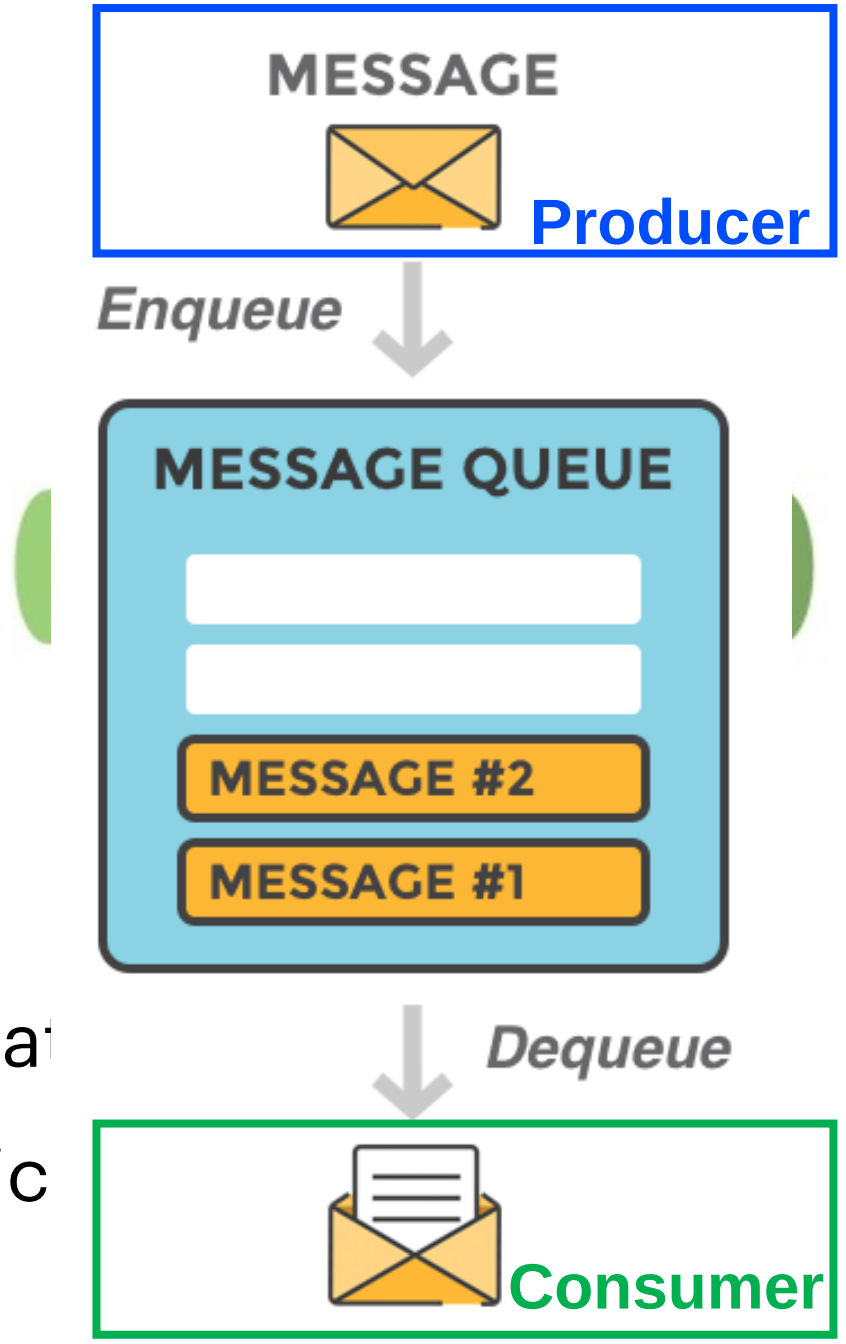
COMMUNICATION VIA MESSAGE QUEUES

- Communicate by sending messages between a ***producer*** and a ***consumer***
- **Asynchronous** communication protocol for highly decoupled systems
- Message is a **byte array** with some headers
- Provides temporary message **storage** when the recipient is not available



COMMUNICATION VIA MESSAGE QUEUES

- Message **delivery** is guaranteed
- High **workload** handled adding consumers
- Built-in **load balancing**: no infrastructure
- Broadcasting via **publish/subscribe**
- Consumers are **protected**: max requests
- **Synchronous** communication can be simulat
- **Debugging** message flow can be more diffic



MESSAGE BROKER COMPARISON

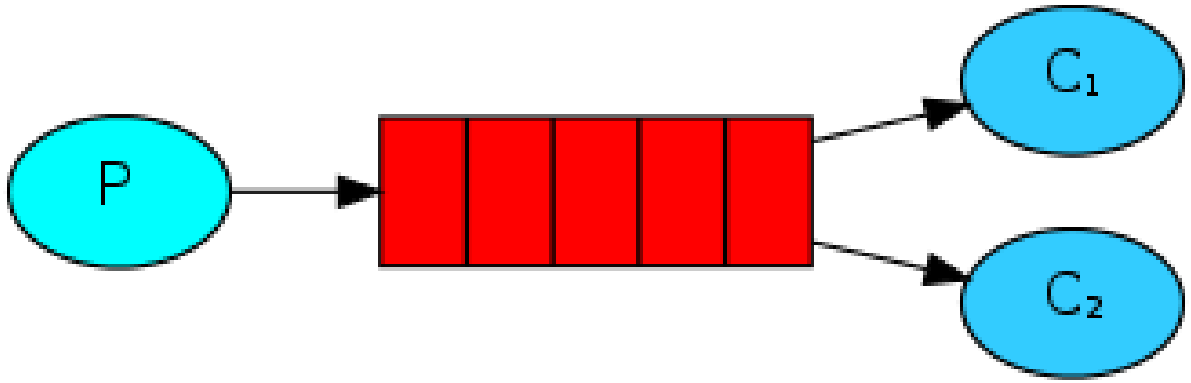
RabbitMQ. Messages queued in central node before sending to recipients. Higher latency and larger headers. Easy to use with advanced scenarios (load balancing, persistence)

ZeroMQ. Broker and P2P topologies. Very lightweight, high throughput and low latency. Broadly used in real-time environments. Difficult to implement advanced scenarios.

ActiveMQ. Broker and P2P topologies. Medium complexity for advanced scenarios, and decent performance.

SAMPLE APPLICATION WITH RABBITMQ

- **1 *producer* microservice** sending tasks to the queue
- **N *worker* microservices** consuming tasks
- Simulated resource-intensive tasks encapsulated in messages
- Run multiple workers in order to manage work load



ARTURO CALVO
Trondheim, July 12th 2018
hello@arturocalvo.com / @artucalvo

accenture digital