

# Философия микросервисов



[vk.com/javaenjoy](https://vk.com/javaenjoy)

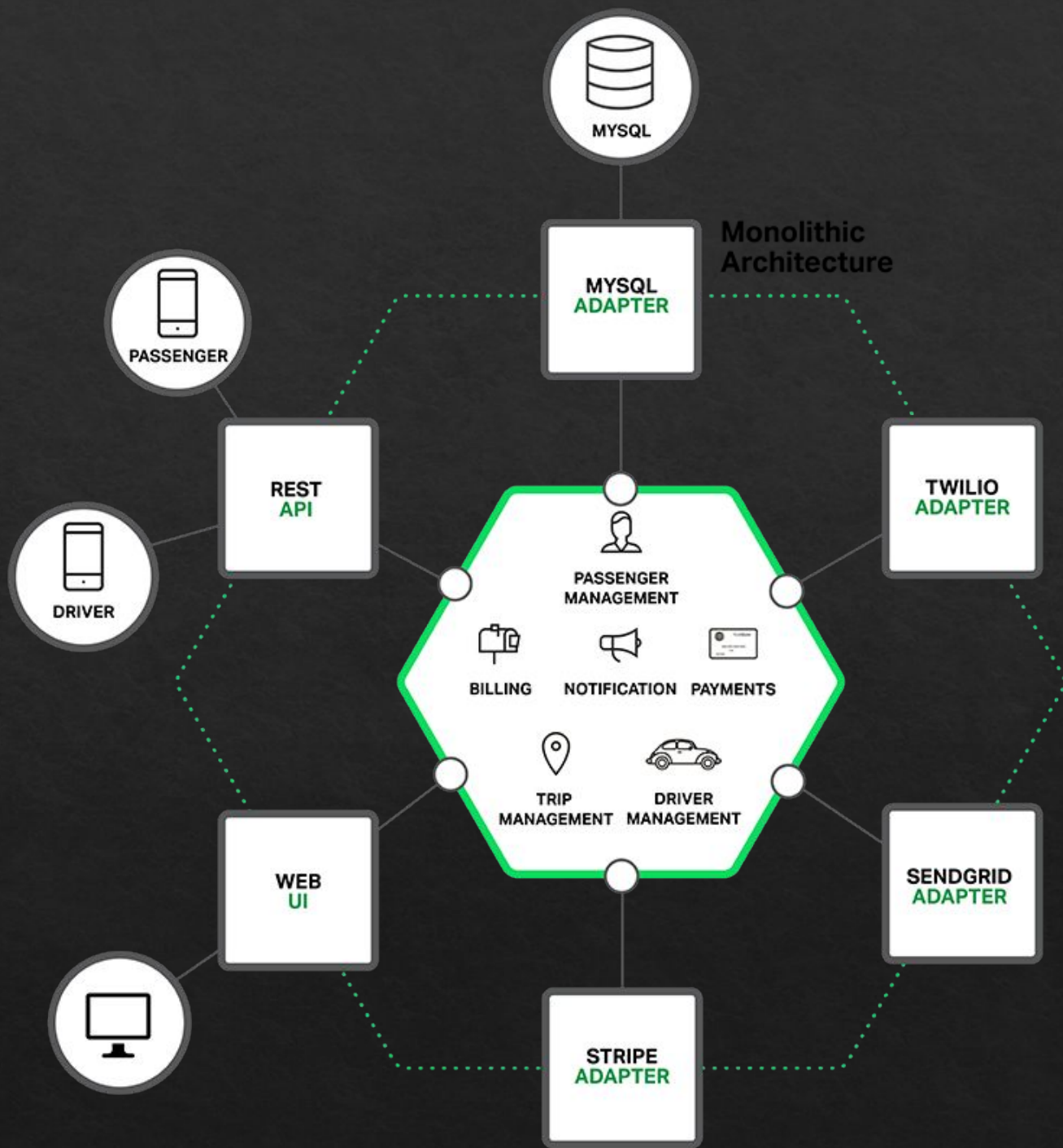


[artplastika.r  
u](https://artplastika.ru)



[technovillage.  
ru](https://technovillage.ru)

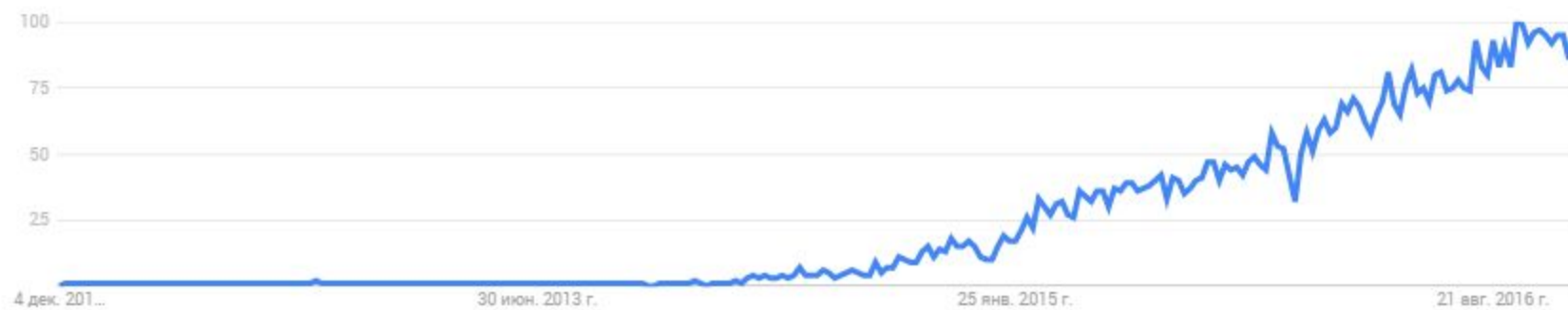
**Виктор Глушенков,**  
независимый разработчик,  
программист-прагматик с 2001  
года



# Недостатки

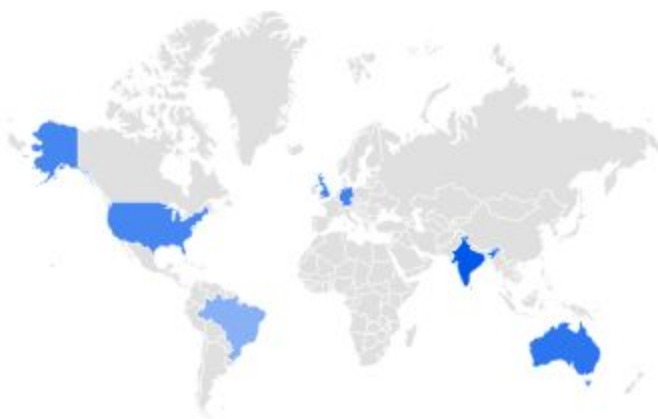
- ◆ Долгосрочная привязка к технологиям
- ◆ Сложность внесения изменений
- ◆ Сложность тестирования
- ◆ Медленное развёртывание
- ◆ Ресурсоёмкое масштабирование

## Динамика популярности ?



## Популярность по регионам ?

Регион ▾ ⋮



1	Индия	100	<div style="width: 100%;"><div style="width: 100%;"></div></div>
2	Австралия	74	<div style="width: 74%;"><div style="width: 74%;"></div></div>
3	Великобритания	66	<div style="width: 66%;"><div style="width: 66%;"></div></div>
4	Германия	63	<div style="width: 63%;"><div style="width: 63%;"></div></div>
5	Соединенные Штаты	60	<div style="width: 60%;"><div style="width: 60%;"></div></div>

# SmallTalk

- ◆ OOP to me means only **messaging**, local retention and protection and hiding of state-process, and extreme late-binding of all things.
- ◆ I thought of objects being like biological cells and/or **individual computers on a network**, only able to communicate with messages
- ◆ However, doing encapsulation right is a commitment not just to abstraction of state, but to **eliminate state oriented metaphors** from programming.

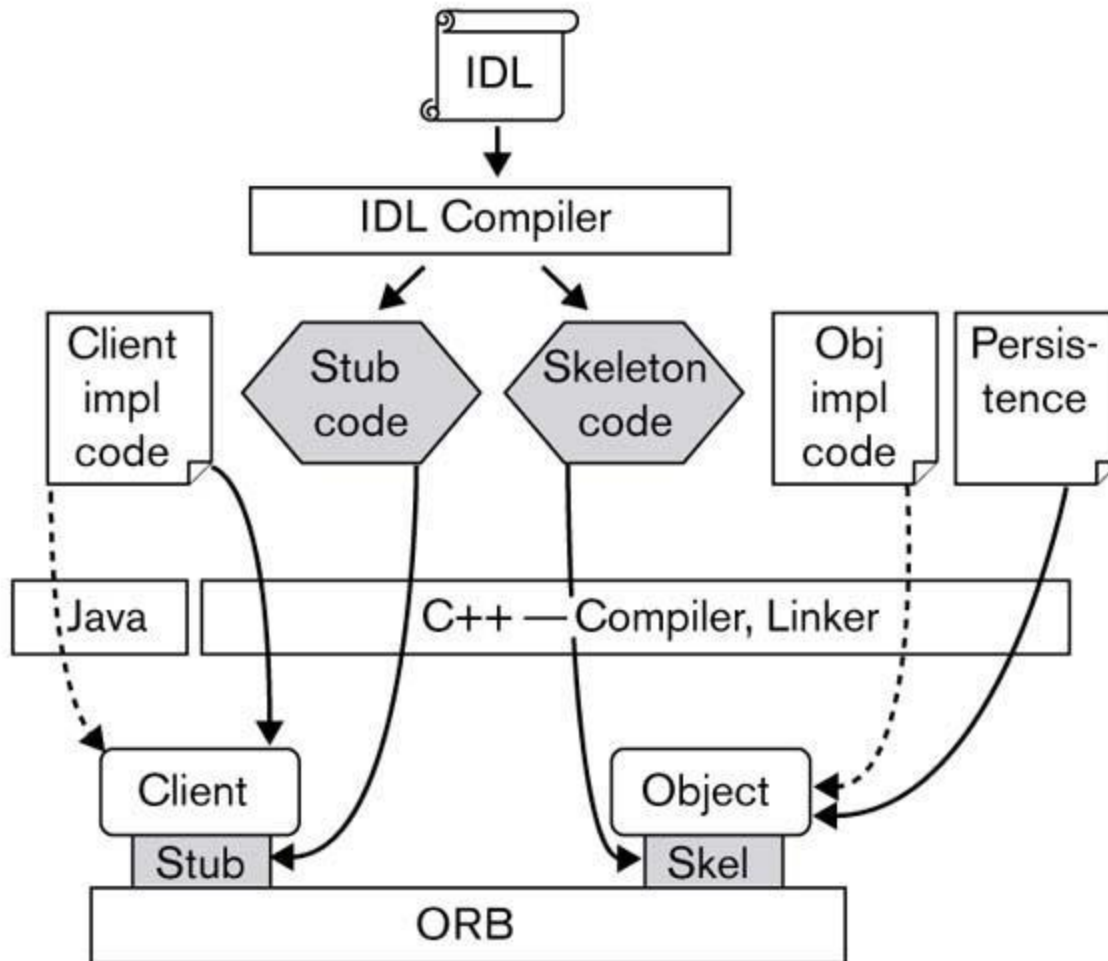


# Object-Oriented Design

- ◆ Decomposition & Abstraction
  - ◆ Single Responsibility Principle
  - ◆ Open/Closed Principle
  - ◆ Low Coupling (связанность)
  - ◆ High Cohesion (зацепление)
  - ◆ Encapsulation
- SOLID
- GRASP
-

# CORBA / RMI

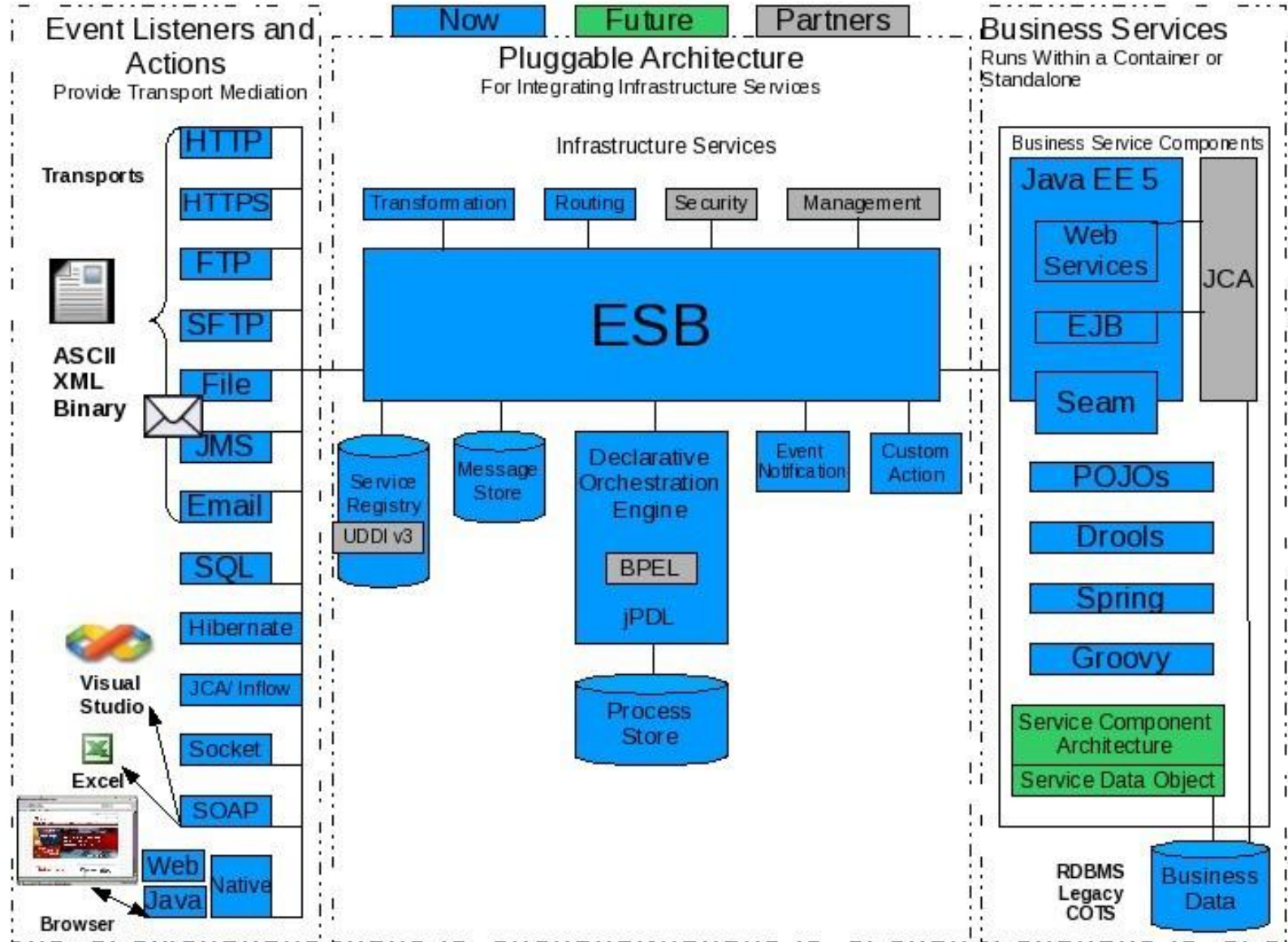
CORBA is particularly important in large organizations, where **many systems** must interact with each other, and legacy systems can't yet be retired. CORBA provides the **connection between** one language and **platform** and another.





# SOA / ESB / JBI

- ◆ For some SOA is about **exposing** software through **web services**
- ◆ For some SOA implies an architecture where **applications disappear**
- ◆ For some SOA is about allowing systems to **communicate** over some form of standard structure (usually XML based) **with other applications**
- ◆ For some SOA is all about using (mostly) asynchronous messaging to transfer documents between different systems.



# OSGi

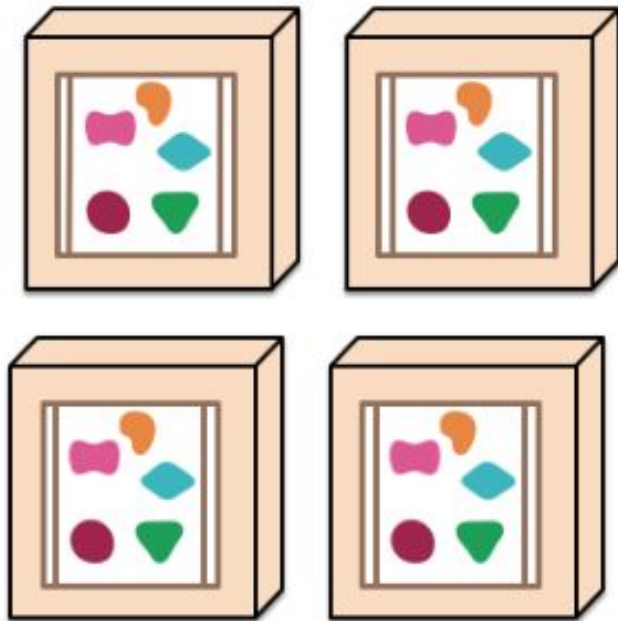
- ◆ Development model where applications are (**dynamically**) composed of many different (**reusable**) components.
- ◆ Components hide their implementations from other components while **communicating through services**, which are objects that are specifically **shared between components**.



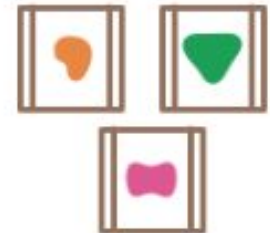
*A monolithic application puts all its functionality into a single process...*



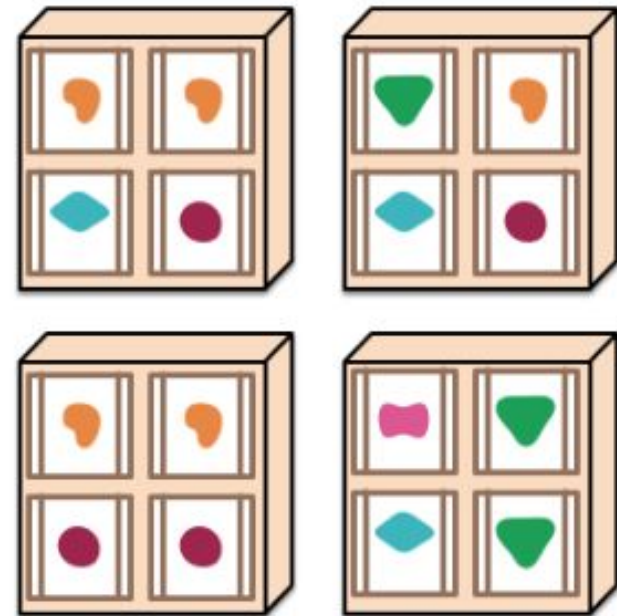
*... and scales by replicating the monolith on multiple servers*



*A microservices architecture puts each element of functionality into a separate service...*



*... and scales by distributing these services across servers, replicating as needed.*

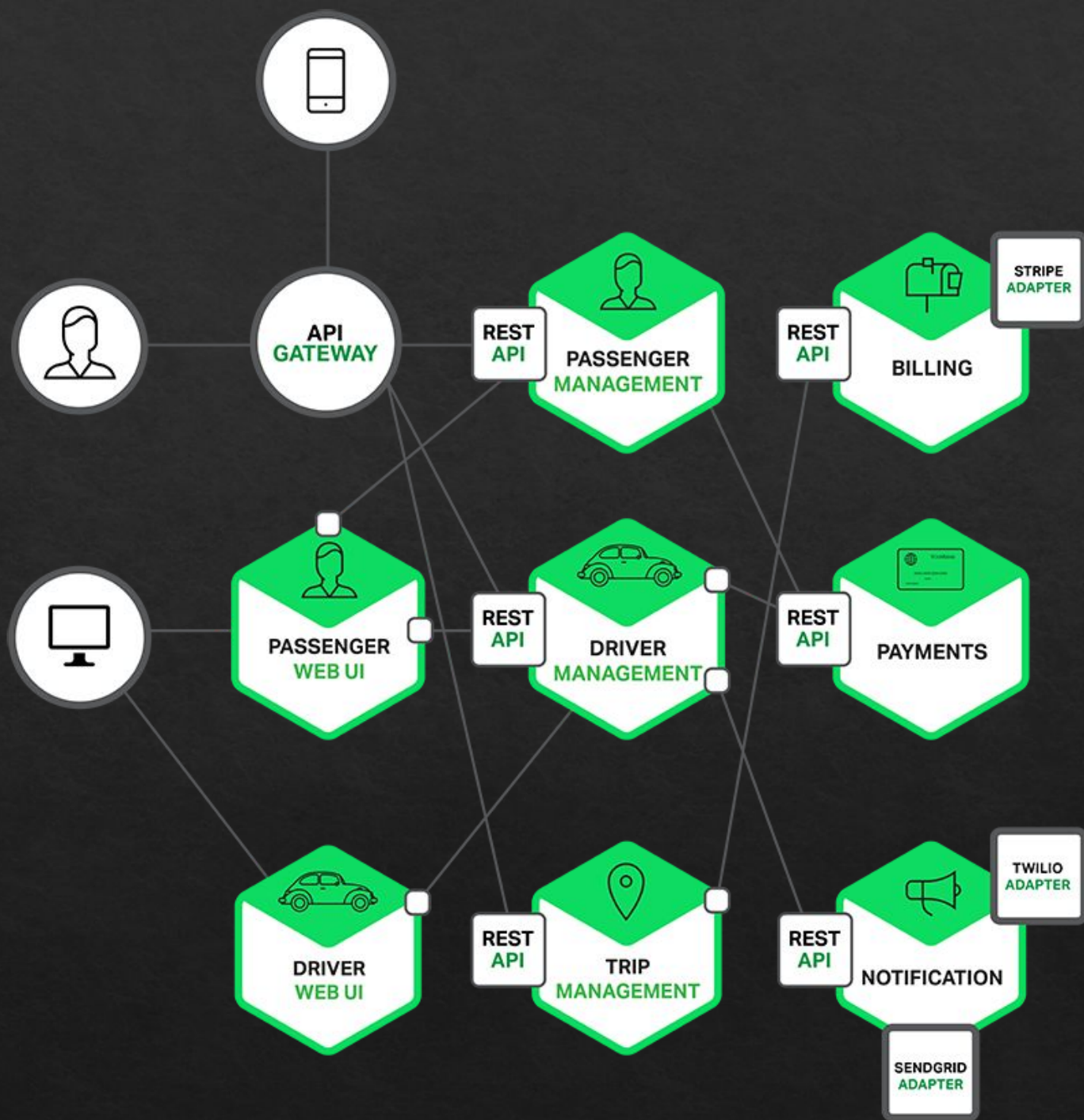


Микросервисы — это  
небольшие, автономные,  
совместно работающие  
сервисы



# Свойства архитектуры

1. Разбиение на компоненты через сервисы
2. Организация вокруг потребностей бизнеса
3. Продукты, а не проекты
4. Умные приёмники и глупые каналы передачи
5. Децентрализованное руководство
6. Децентрализованное управление данными
7. Автоматизация инфраструктуры
8. Проектирование под отказ
9. Эволюционный дизайн



# Вселяет оптимизм (Pros)

- ◆ Чёткие границы сервисов
- ◆ Простота добавления и переделки функционала
- ◆ Простота тестирования
- ◆ Решение организационных вопросов
- ◆ Отказоустойчивость
- ◆ Масштабирование
- ◆ Повторное использование
- ◆ Простота развёртывания
- ◆ Технологическая разнородность

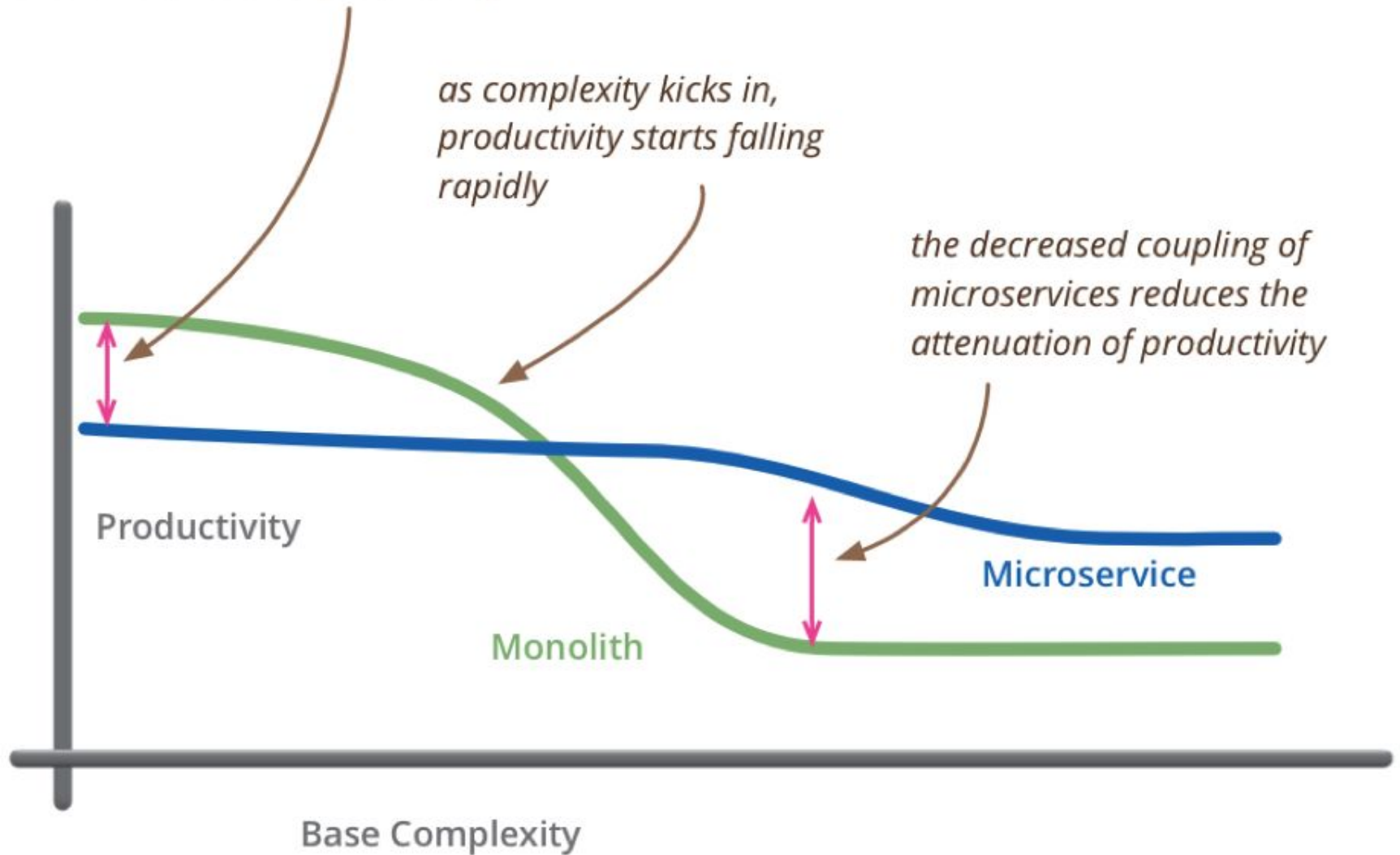
# Вызывает скептицизм (Cons)

- ◆ Сложность разработки распределённых систем
- ◆ Производительность при взаимодействии
- ◆ Версионность API сервисов
- ◆ Распределённые транзакции, авторизация
- ◆ Сложность рефакторинга (границы модулей)
- ◆ Сложность сопровождения (логирование, отладка)
- ◆ Необходим высокий уровень автоматизации
- ◆ Требуются DevOps-навыки и специалисты

*for less-complex systems, the extra baggage required to manage microservices reduces productivity*

*as complexity kicks in, productivity starts falling rapidly*

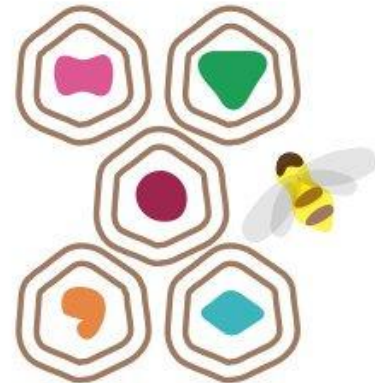
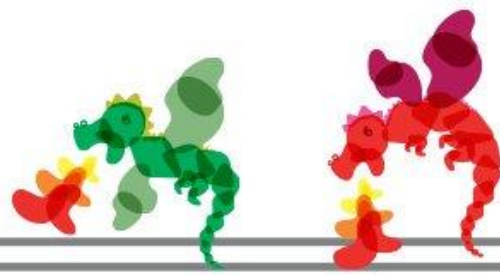
*the decreased coupling of microservices reduces the attenuation of productivity*



*but remember the skill of the team will outweigh any monolith/microservice choice*



*Going directly to a  
microservices  
architecture is risky*

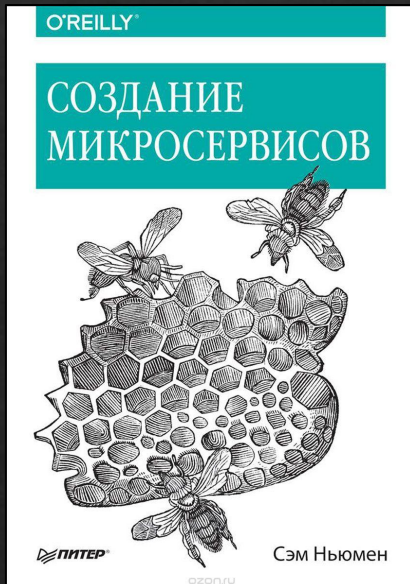


*A monolith allows you to  
explore both the complexity  
of a system and its  
component boundaries*

*As complexity rises start  
breaking out some  
microservices*

*Continue breaking out  
services as your knowledge  
of boundaries and service  
management increases*

# Литература



- [Microservices Resource Guide](#) by Martin Fowler & James Lewis
- [Microservices: From Design to Deployment](#) by Chris Richardson
- [«Народная библиотека» сообщества TechnoVillage](#) — МОЖНО ВЗЯТЬ ПОЧИТАТЬ КНИГИ

# Что дальше?

- ◆ — Работайте, братья!
- ◆ Spring Boot
- ◆ RxJava
- ◆ Continuous Delivery
- ◆ WebPurple MeetUp: «Level-Up By Community Growth-Hacking»