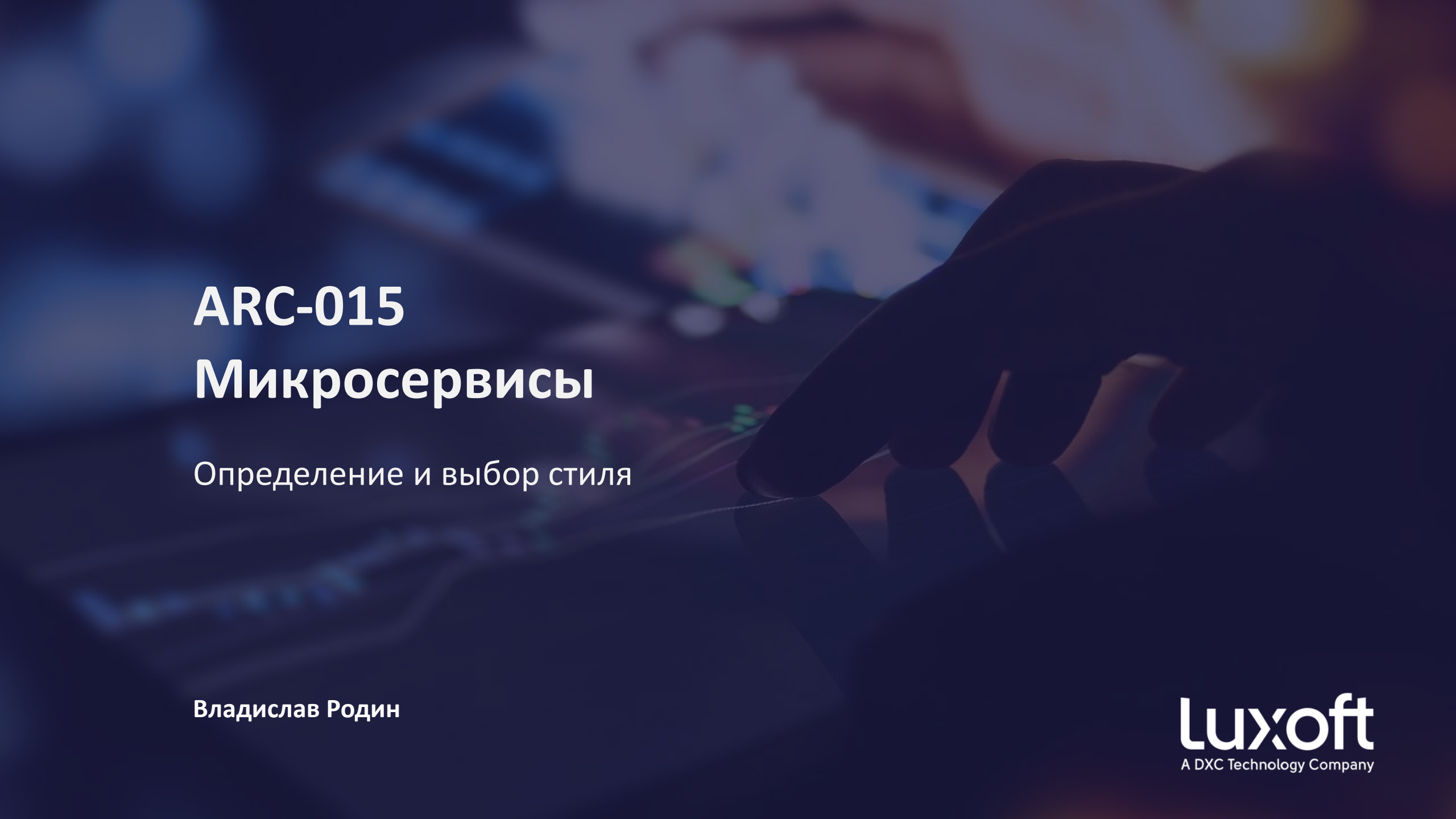


ARC-015
Микросервисы

Определение и выбор стиля

Владислав Родин

Luxoft
A DXC Technology Company

История термина

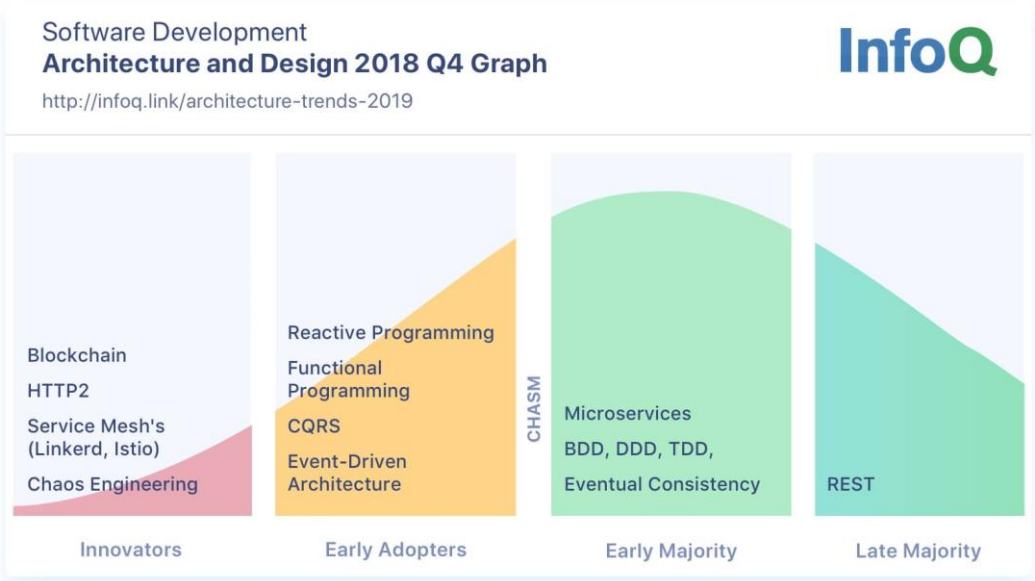
- В 2011 году Джеймс Льюис, анализируя работы различных компаний, обратил внимание на появления нового паттерна оптимизирующего SOA с целью ускорения развертывания сервисов.
- Этот паттерн был назван “micro-app”.
- Позже, в 2012 году на архитектурном саммите паттерн был переименован в **микросервис** (microservice).

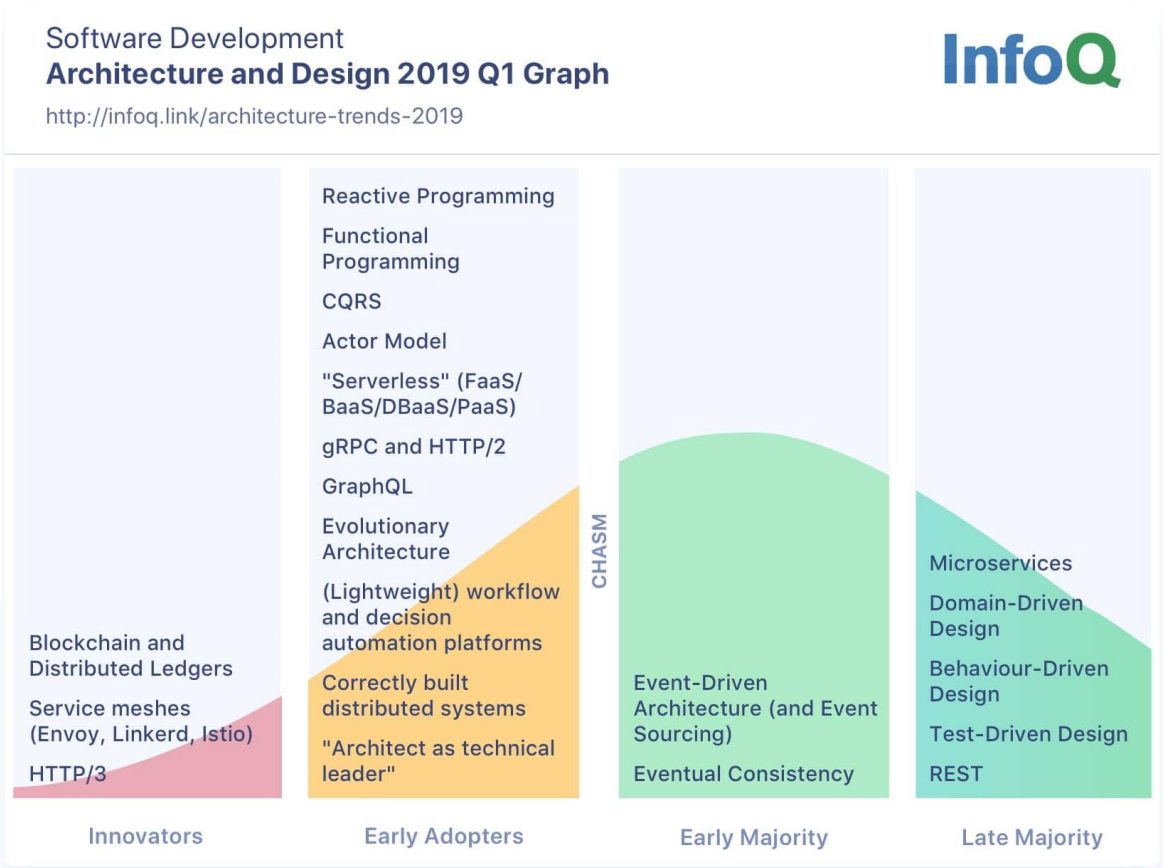
Время выхода на рынок (Time to market)

- Первоначальной целью внедрения микросервисов была цель уменьшения времени выхода на рынок

- Фазы продуктового инкремента
  - Анализ и проектирование
  - Разработка (улучшаем модифицируемость)
  - Тестирование (уменьшаем изменяемую область)
  - Поставка (уменьшаем количество развертываемых элементов)

Современная тенденция





Определение микросервиса

Microservices are autonomous services that work together, modeled around a business domain, decentralized, and hides implementation detail

Микросервисы - это автономные сервисы, которые взаимодействуя моделируют предметную область, децентрализованы и скрывают детали реализации

Sam Newman

Основной конкурент MSA – Монолит (все в одном)

Проблемы

- Размер
- Связанность
- Развертывание
- Масштабируемость
- Надежность
- Косность

Монолит: Размер

- **Приложение слишком велико, чтобы его мог понять один разработчик.**

- **С монолитом хорошо могут работать только те, кто провел за этим кодом много времени.**

  Новички же потратят кучу времени на выяснение того, как программа работает.

  И не факт, что разберутся.

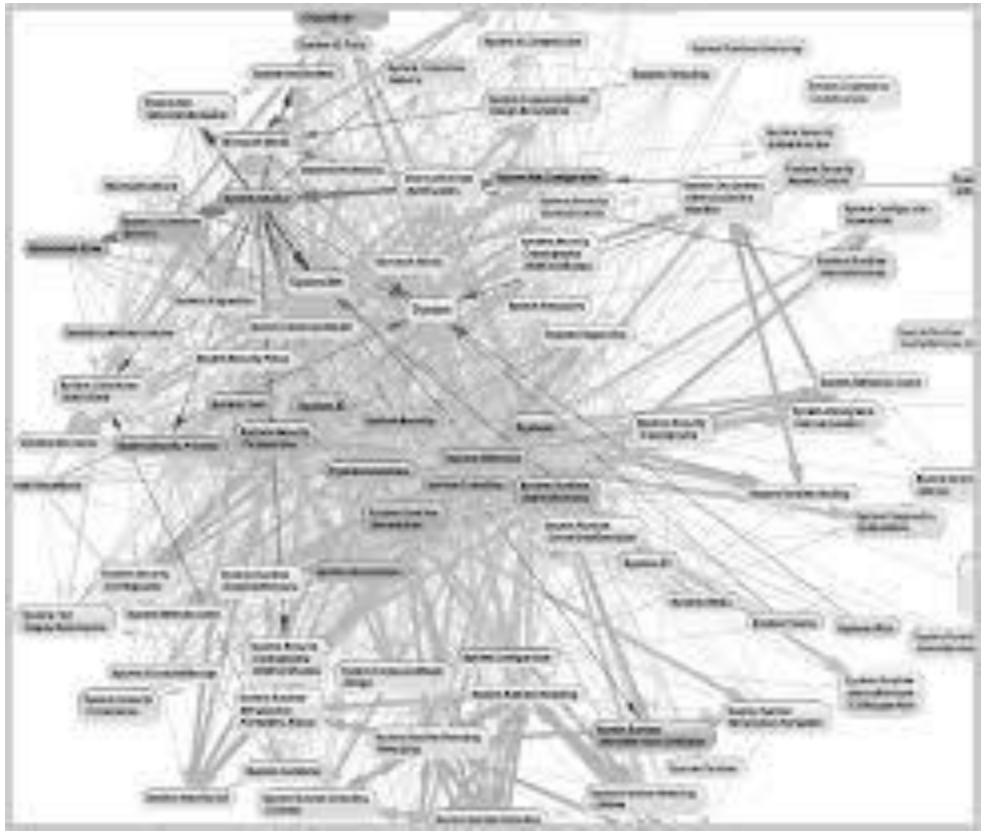
- **Уход ключевого специалиста – гибель монолита.**

Монолит: Связанность

- Большой ком грязи (Big Ball of Mud), изменения в котором приводят к непредсказуемым последствиям.

- Внеся изменения в одном месте монолита, можно вызвать повреждения в других ее частях.

  Все дело в том, что компоненты в монолите могут иметь очень сложные и неочевидные взаимосвязи (спагетти-код).



Монолит: Развертывание

- **Развертывание монолита – долгий процесс со своим ритуалом.**

Монолит: Масштабирование

- **Модули имеют конфликтующие потребности в ресурсах.**

  Компромисс при выборе железа.

Монолит: Надежность

- **Все службы в одном процессе.**

  Ошибка в модуле – отказ всего приложения.

- **Даже если ошибка была в коде маленькой и незначительной операции, которая нужна 0.01% пользователей, — в случае сбоя может прилечь вся ваша система.**

Монолит: Косность

- Трудно перейти на новые технологии.
- И как следствие – проблема удержать единственного специалиста.
- Выбранный на старте проекта технологический стек, становится блоком (lock), не позволяющим развивать продукт.

Компонентно-ориентированное программирование (component-oriented programming, COP)

парадигма программирования, существенным образом опирающаяся на понятие компонента — независимого модуля исходного кода программы, предназначенного для повторного использования и развёртывания и реализующегося в виде множества языковых, объединённых по общему признаку и организованных в соответствии с определёнными правилами и ограничениями.

- **История:**
  - **1987 г — Никлаус Вирт предложил для языка «Оберон» паттерн написания блоков. Паттерн заключался в том, что компонент компилируется отдельно от других, а на стадии выполнения — необходимые компоненты подключаются динамически.**
  - **1989 г — Бертран Мейер предложил идею единого взаимодействия между вызываемым и вызывающим компонентами. Эта идея воплотилась в виде готовых решений: CORBA, COM, SOAP**

Компонентная архитектура

- **Компонентная архитектура – архитектура системы разбивается на компоненты, которые можно использовать как в текущем, так и в будущих проектах.**

RUP 1998

- **Разбиваем систему на компоненты, учитывая**
  - **пригодность для повторного использования;**
  - **замещаемость;**
  - **независимость от контекста;**
  - **расширяемость;**
  - **инкапсуляцию;**
  - **независимость.**
- **Примеры**
  - **DCOM, EJB и т. п.**

Характеристики компонентной архитектуры

- **Разбиение на компоненты решает проблемы большого кома грязи**
  - **Большой размер**
  - **Высокая связанность**

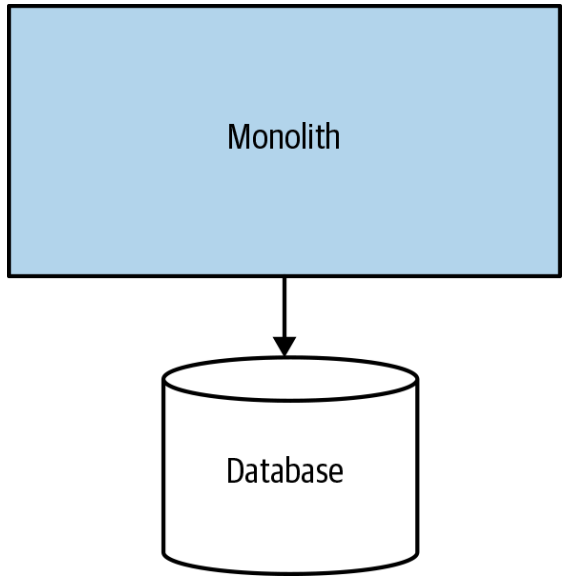
- **Компоненты могут быть выполнены в виде**
  - **модулей, библиотек (единицы сборки),**
  - **сервисов (единицы развертывания)**
    - **Некоторые считают, что сервис отображается на выполняемый процесс. Это ошибка. Сервис может отображаться на несколько процессов, которые будут разрабатываться и развертываться совместно. Например, веб приложение и база данных.**

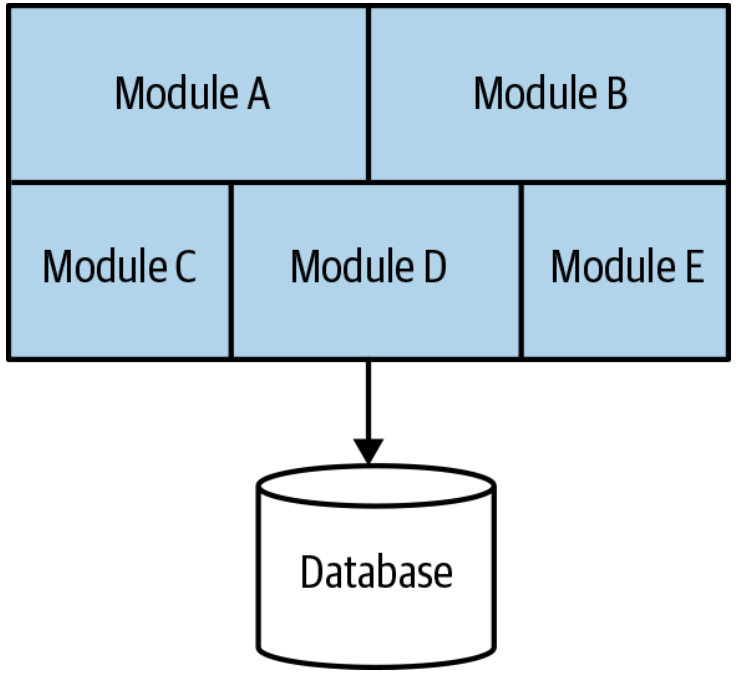
Проблемы модульного подхода

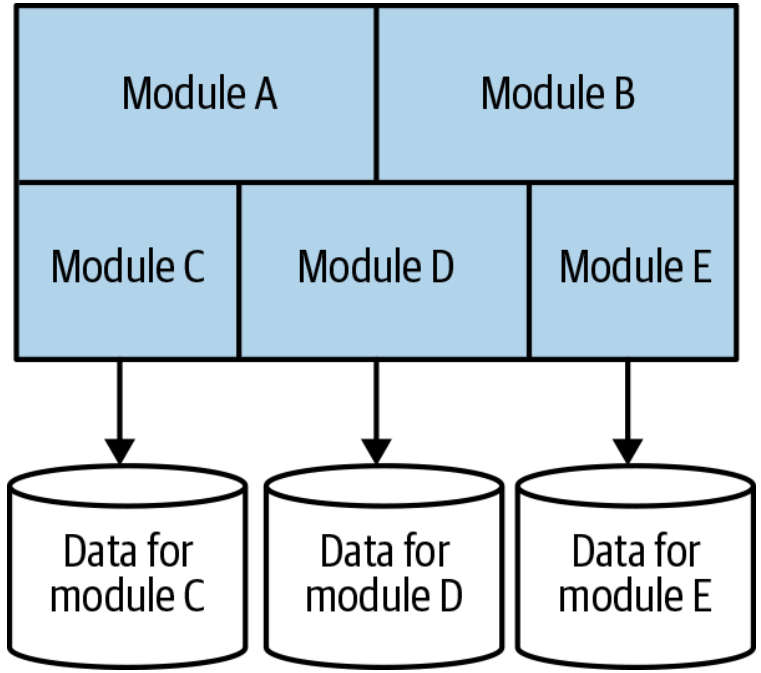
- Модули позволяют осуществлять независимую разработку компонент, но не решают проблемы
  - Независимого развертывания
  - Независимого масштабирования
  - Надежности (отказоустойчивости)
  - Независимости от общего технологического стека (lock)
- Разделение на модули внутри приложения логическое, и может быть нарушено разработчиками

- Модуль – частично независимая компонента

Разновидности монолитов

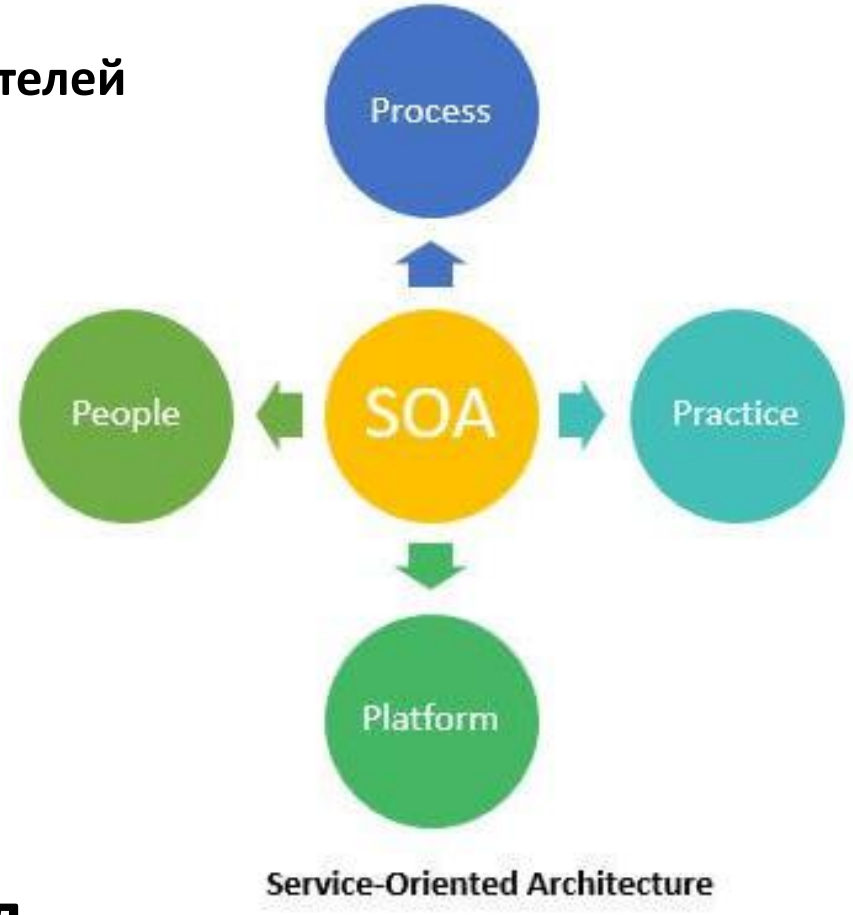






Сервис-ориентированная архитектура (SOA)

- **Принципы SOA:**
  - **Сочетаемость приложений, ориентированных на пользователей**
  - **Многократное использование бизнес-сервисов**
  - **Независимость от набора технологий**
  - **Автономность**
    (независимые эволюция, масштабируемость и развёртываемость)



- **Примеры:**
  - **COBRA, Web-сервисы на основе SOAP и т. д.**

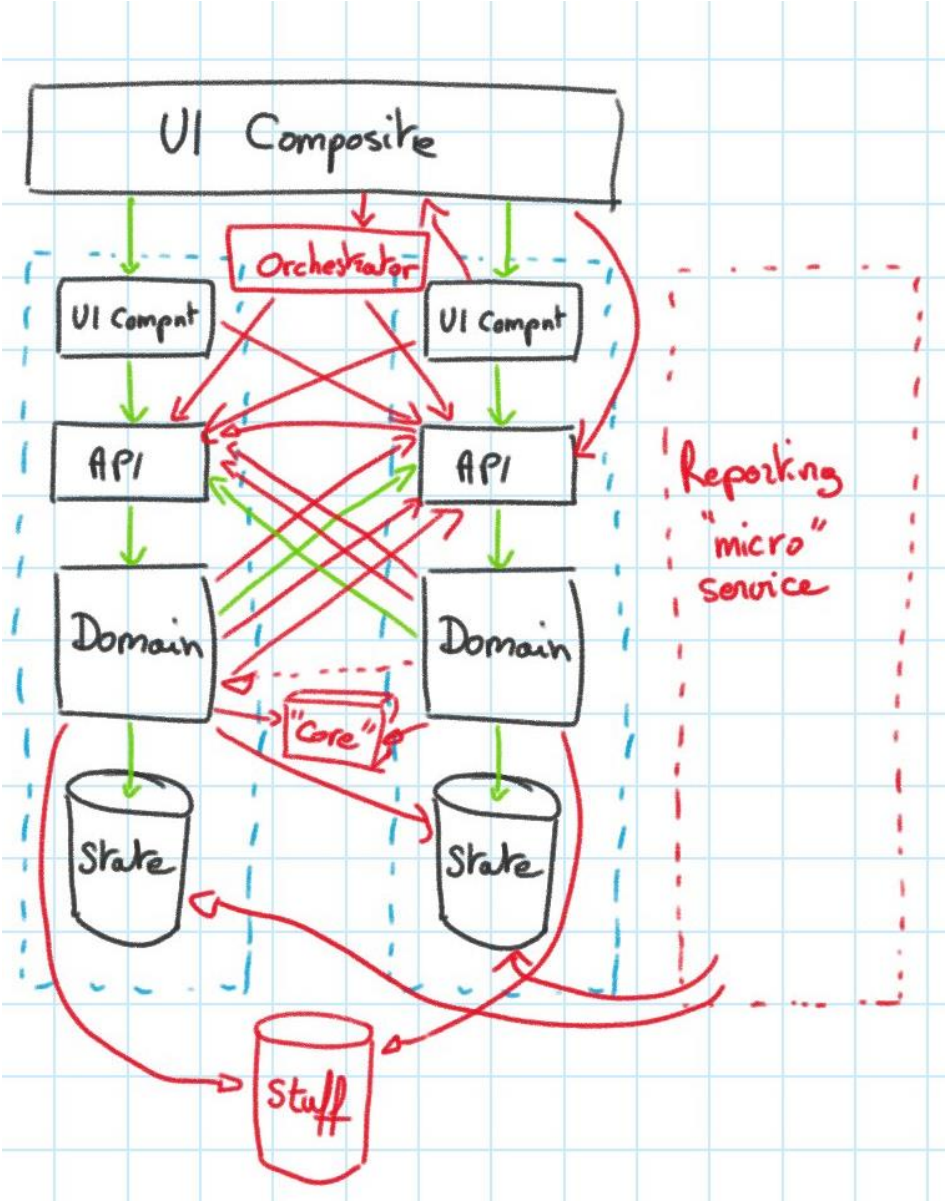
Результат

- **Архитектура SOA решает все проблемы монолита:**
  - **При изменении требуется развертывание только одной службы.**
  - **Четко определенные интерфейсы служб (API) поддерживают инкапсуляцию компонент.**

- **Но создает новые:**
  - **Удаленные вызовы дороже чем локальные.**
  - **Перераспределение обязанностей между компонентами затруднено.**

Распределенный монолит

- Возможность независимого развертывания важная особенность сервиса
- Если сервисы системы должны развертываться совместно, и, даже возможно, в определенной последовательности, то мы не можем называть эту систему сервис-ориентированной.
- Подобную систему принято называть **распределенным монолитом** (distributed monolith)



Особенности SOA

- **Поддержка**
  - **Поддержка архитектурного комьюнити**
  - **Великолепная поддержка вендоров**
    - **Обучение и сертификация**
    - **Решения**
- **Хорошо проработанные паттерны**
  - **В числе которых сервисная шина предприятия (ESB)**

    Но, SOA ≠ ESB

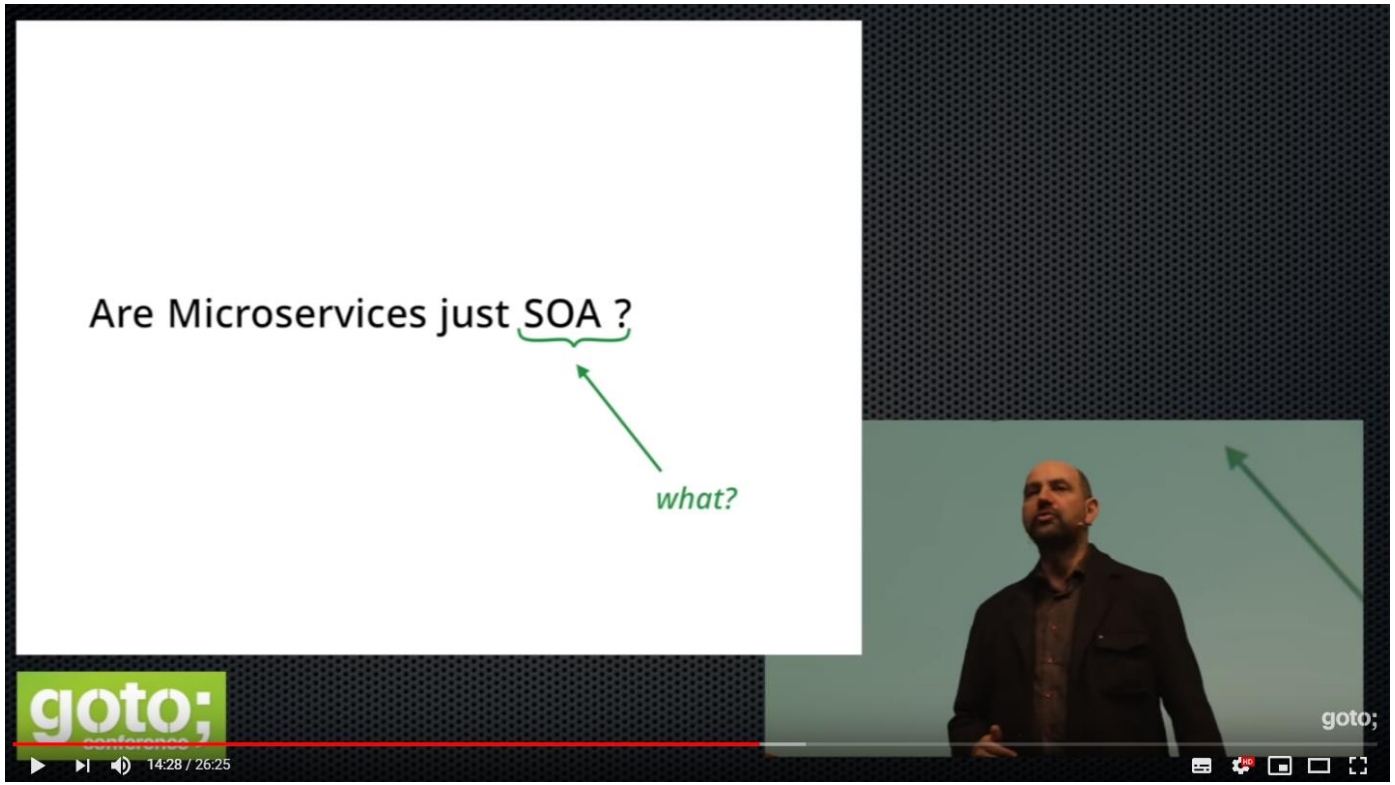
Сервисно-ориентированная неопределенность (Service Oriented Ambiguity)

- **Всякий раз, когда ThoughtWorks опрометчиво выпускает меня к клиенту, мне непременно задают один и тот же вопрос: «что вы думаете о SOA?»**
- **На этот вопрос практически невозможно ответить, потому что SOA означает разное для разных людей.**

Martin Fowler

MSA vs SOA

Микросервисы это SOA



Мартин Фаулер

Отношение архитектур

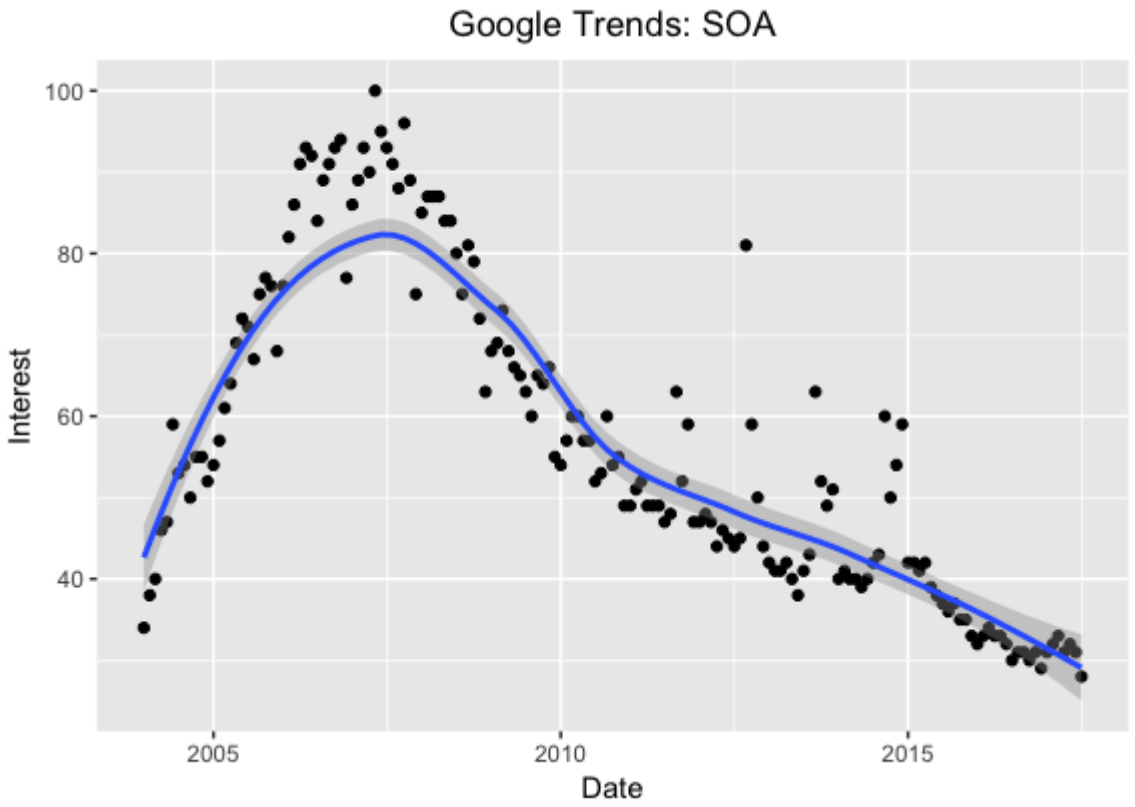
- **Микросервисы это сервисы и компоненты,**

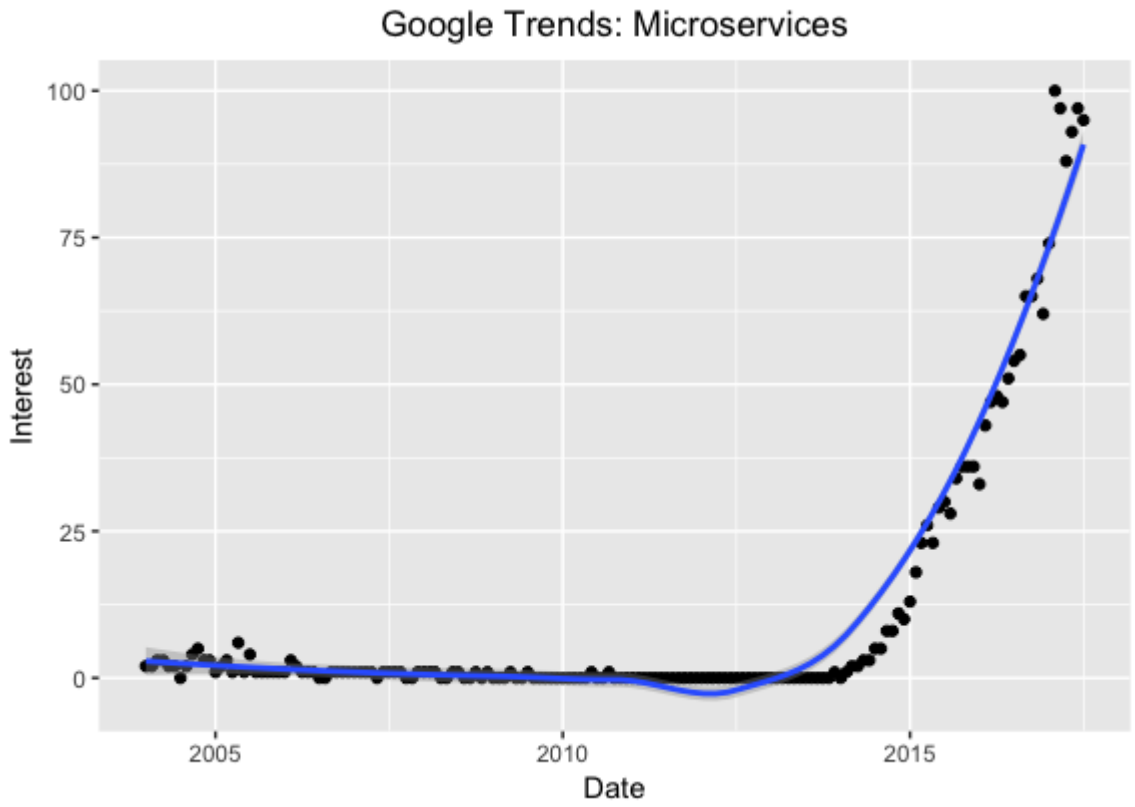
  но не любой сервис

  и, тем более, компонент микросервис.

MSA vs SOA: тенденции

Микросервисы это SOA, но без надоевшего багажа от вендоров





Характеристики MSA

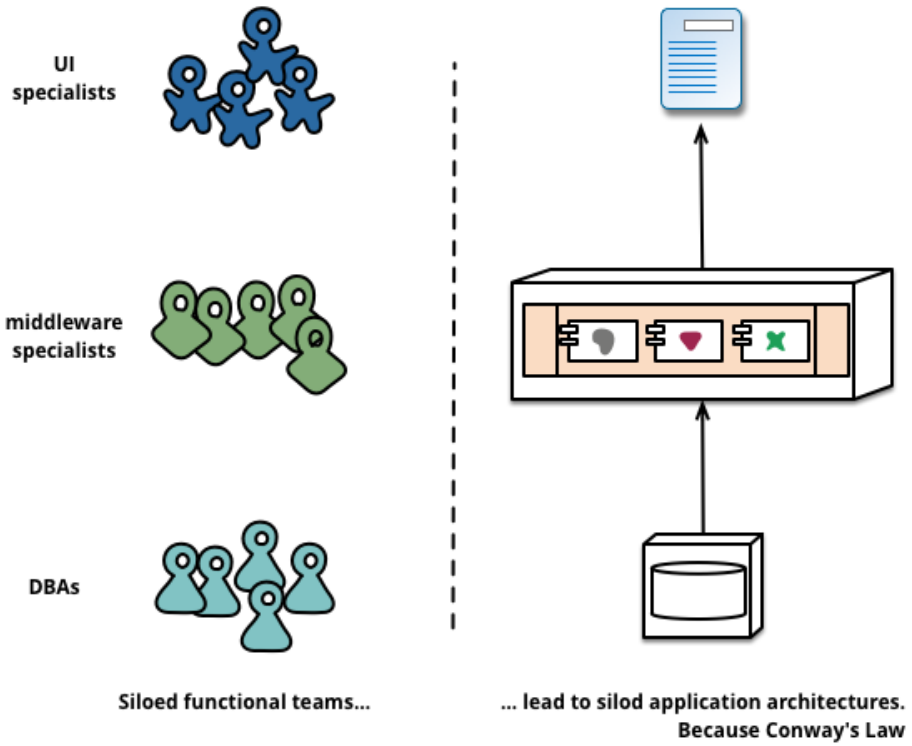
- Компонентное представление через сервисы (Componentization via Services)
- Организация в соответствии с бизнес-возможностями (Organized around Business Capabilities)
- Продукты, а не проекты (Products not Projects)
- Умные точки входа и глупые каналы (Smart endpoints and dumb pipes)
- Децентрализованное управление (Decentralized Governance)
- Децентрализованное управление данными (Decentralized Data Management)
- Автоматизация инфраструктуры (Infrastructure Automation)
- Страховка от сбоев (Design for failure)
- Архитектура с эволюционным развитием (Evolutionary Design)

Отличие от SOA

James Lewis

Martin Fowler

Организация человеческих ресурсов в соответствии с возможностями бизнеса (1 из 2)



Когда-то внутри команд разработчиков самоорганизовывались группы на основе используемых технологий.

В результате проект создавали команда по DBA, команда разработки серверной части и команда разработки интерфейса, действовавшие независимо друг от друга.

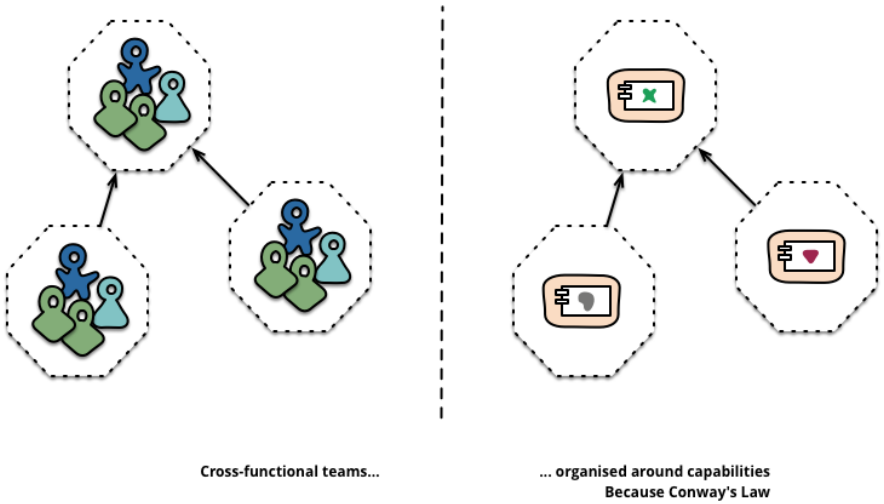
Любые изменения требуют участия нескольких групп разработчиков

Закон Конвея

- **Организации, разрабатывающие системы… создают архитектуры, которые копируют структуры взаимодействий внутри этих организаций.**

Мелвин Конвей, 1967

Организация человеческих ресурсов в соответствии с возможностями бизнеса (2 из 2)



При микросервисном подходе команды должны организовываться на основе бизнес-возможностей: например команда заказов, отгрузки, каталога и т. д.

В каждой команде должны быть специалисты по всем необходимым технологиям (интерфейс, серверная часть, DBA, QA...).

Это даст каждой команде достаточный объём знаний, чтобы сосредоточиться на создании конкретных частей приложения

Продукты, а не проекты

- Проектный подход, команда создает продукт и передает его другой команде.
- В случае MSA команда должна поддерживать продукт на протяжении всего жизненного цикла.

«вы создаете продукт, и вы же запускаете его» ("you build, you run it")

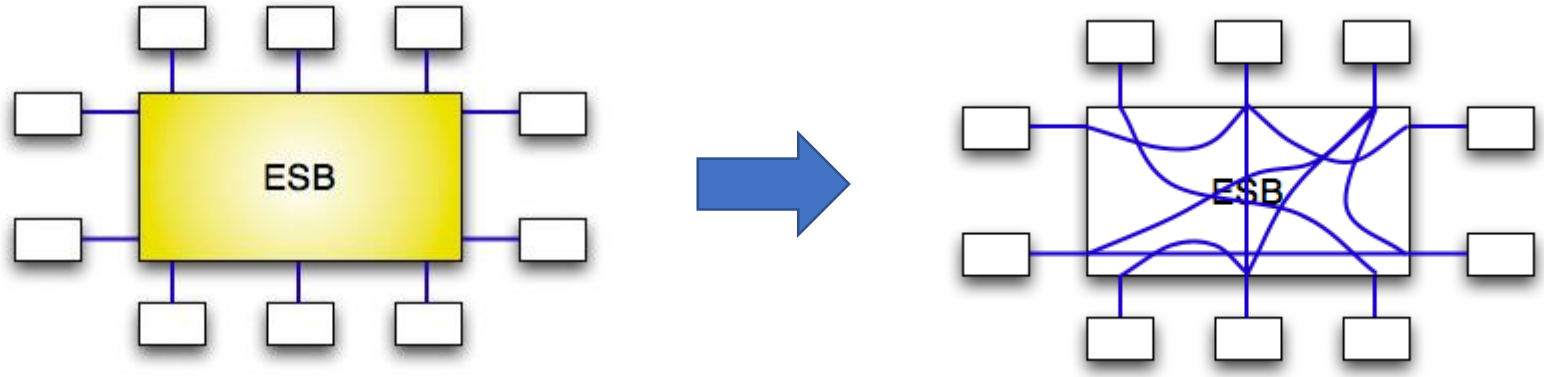
Amazon

- Продуктовый подход позволяет команде почувствовать потребности бизнеса.

Умные точки входа и глупые каналы

- SOA архитектура большое внимание уделяла каналам связи, в частности Enterprise Service Bus (сервисная шина).

  Что зачастую приводит к Erroneous Spaghetti Box.



- Сложность монолита не должна переходить в сложность связей между сервисами.
- Используем только простые способы взаимодействия.

Децентрализованное управление

- **Одним из последствий централизованного управления является тенденция к стандартизации.**

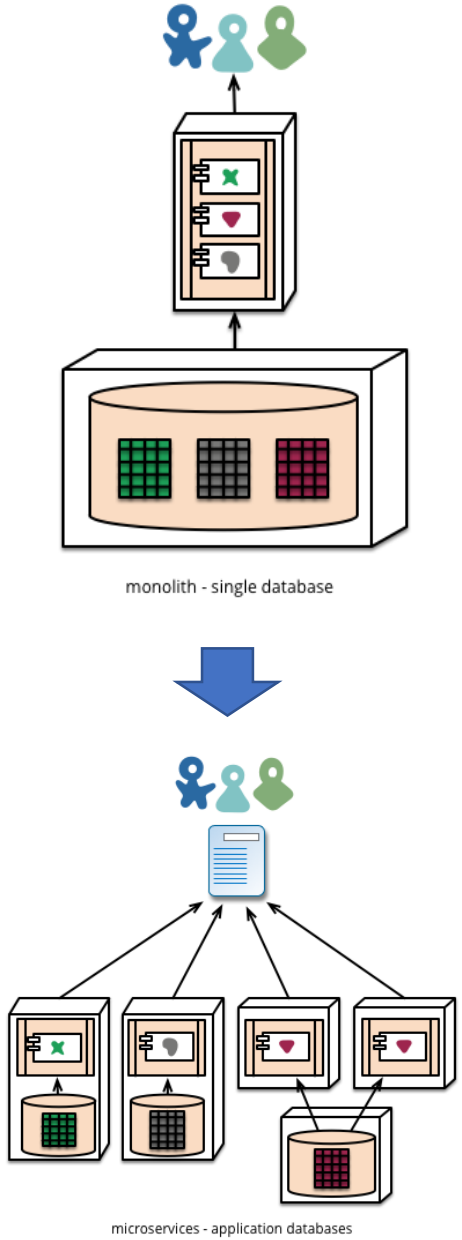
  Если я молоток, то все кругом гвозди

- **Ключевые решения по микросервисам должны принимать люди, которые действительно разрабатывают микросервисы.**

  Здесь под ключевыми решениями подразумевается выбор языков программирования, методологии развёртывания, контрактов публичных интерфейсов и т. д.

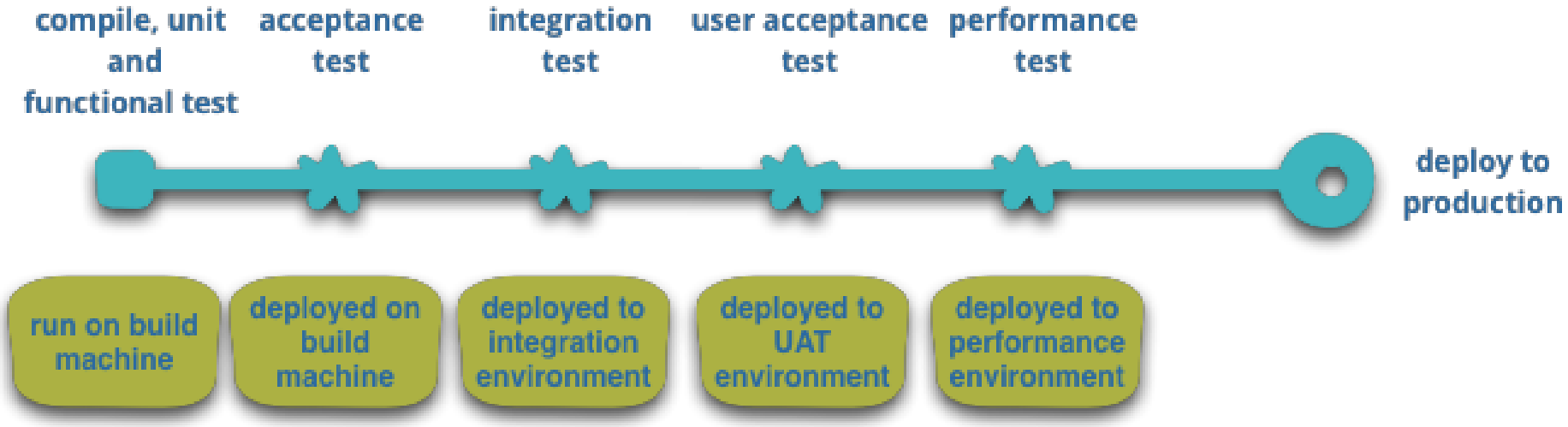
Децентрализованное управление данными

- Стандартный подход, в котором приложение опирается на одну базу данных, не может учитывать специфики каждого конкретного сервиса
- MSA предполагает децентрализованное управление данными, вплоть до применения различных технологий (Polyglot Persistence)
- С точки зрения Фреда Джорджа, это первый вызов на пути к микросервисной архитектуре.



Автоматизация инфраструктуры (1 из 2)

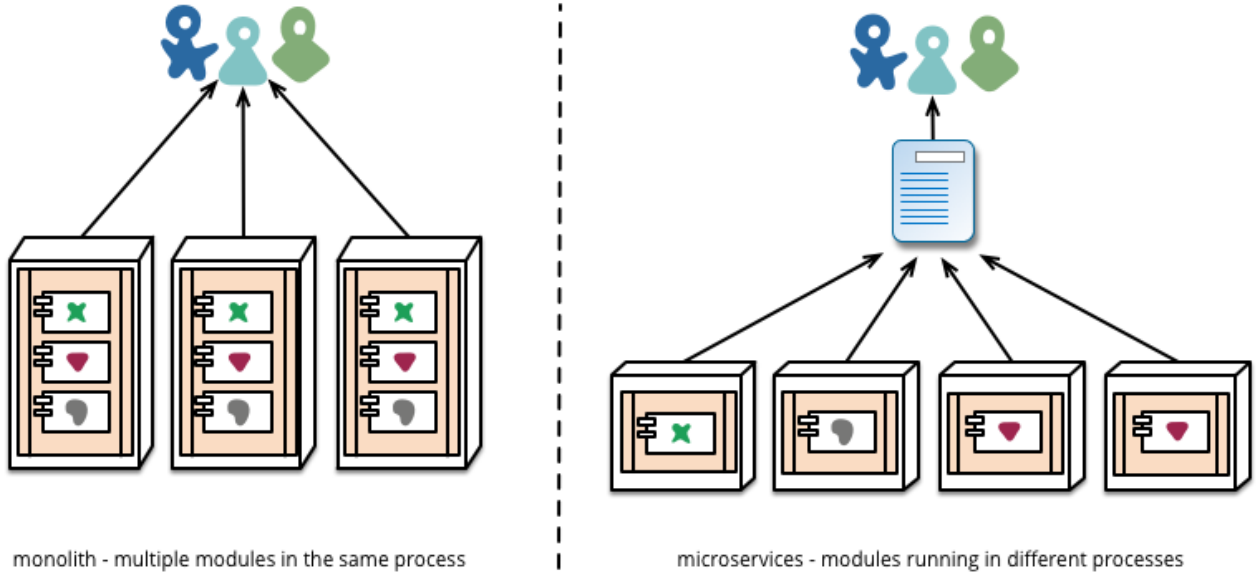
- MSA поддерживает процессы непрерывного развертывания и поставки.
- Осуществить это возможно только путем автоматизации процессов.



- При этом, развертывание большого количества сервисов уже не выглядит чем-то страшным.
- Процесс развертывания должен стать скучным.

Автоматизация инфраструктуры (2 из 2)

- Второй аспект связан с управлением сервисами в продуктовой среде
- Без автоматизации, управления процессами запущенными в разных операционных средах становится невозможным



Страховка от сбоев

- Многочисленные сервисы MSA подвержены сбоям
- При этом, обработка ошибок в распределенной системе весьма не тривиальная задача
- Архитектура приложений должна быть устойчива к таким сбоям

- Ребекка Парсонс считает очень важным, что мы больше не используем даже внутрипроцессное взаимодействие между сервисами, вместо этого для связи мы прибегаем к HTTP, который и близко не бывает столь же надёжен

Архитектура с эволюционным развитием

- Архитектура MSA системы должна развиваться эволюционно.
- Желательно ограничить необходимые изменения границами одного сервиса.
- Необходимо так же учитывать влияние на другие сервисы.
  - Традиционный подход состоит в том, чтобы попытаться решить эту проблему с помощью управления версиями, но MSA предполагает использовать управление версиями в качестве крайней меры.

Стиль «микросервисная архитектура» (MSA) это

- …подход к разработке отдельного приложения в виде набора небольших сервисов, каждый из которых работает в своем собственном процессе и взаимодействует посредством облегченных механизмов, часто API-интерфейсом HTTP-ресурсов.
- Эти сервисы построены на бизнес-возможностях и могут быть развернуты независимо с помощью полностью автоматизированного механизма развертывания.
- Существует минимальный уровень централизованного управления этими сервисами, которые могут быть написаны на разных языках программирования и использовать разные технологии хранения данных.

Microservices
a definition of this new architectural term (2014)

James Lewis

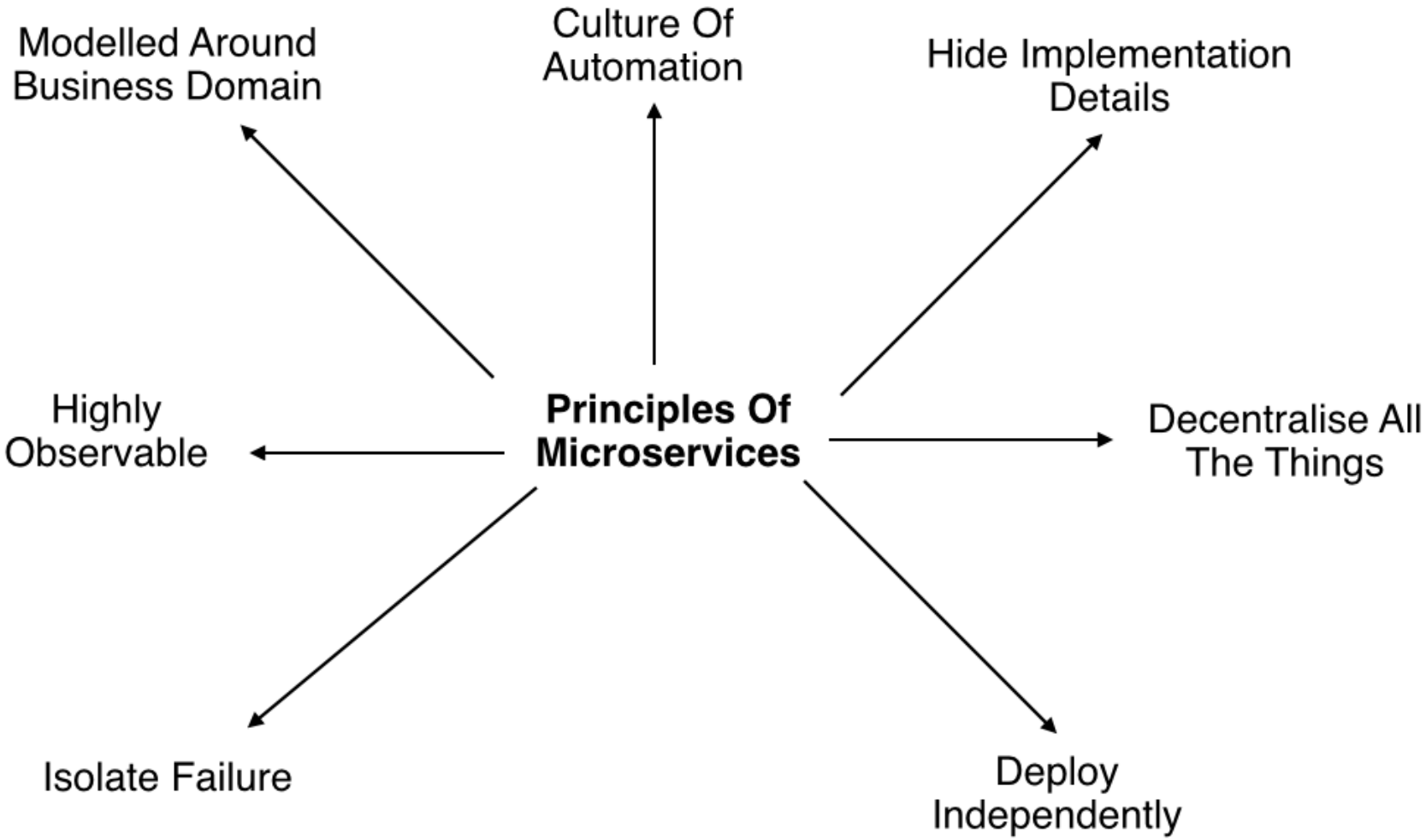
Martin Fowler

Крис Ричардсон MSA это

- MSA это архитектурный стиль представляюший коллекцию сервисов которые
  - Легки в поддержке и тестировании (Highly maintainable and testable)
  - Слабо связанны (Loosely coupled)
  - Независимо разворачиваемы (Independently deployable)
  - Организованы вокруг бизнес-возможностей (Organized around business capabilities)
  - Принадлежат небольшой команде (Owned by a small team)

Chris Richardson

Сэм Ньюмен
MSA это

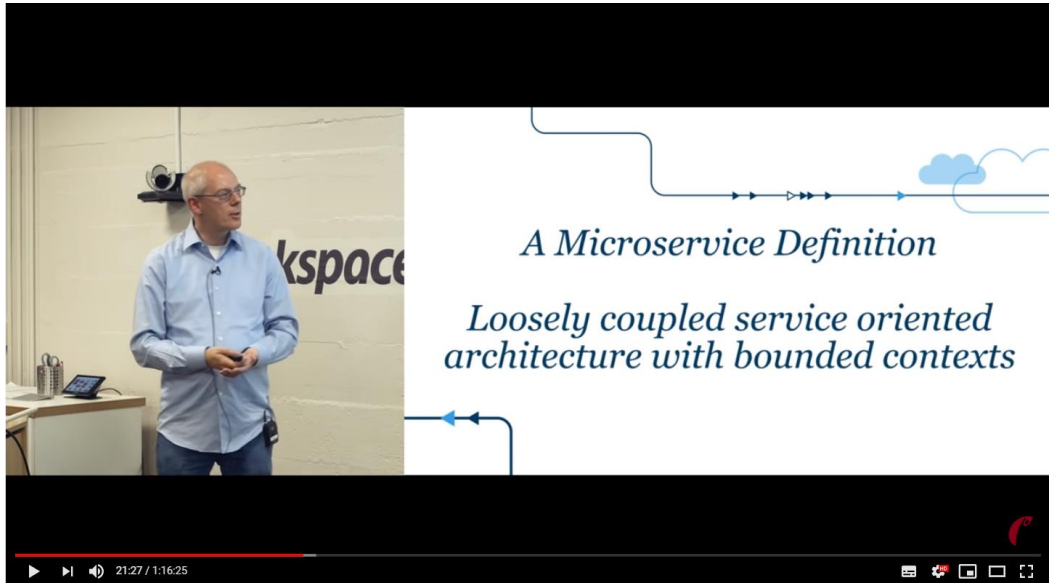


Sam Newman

Эдриан Кокрофт
MSA это

Архитектура на основе свободно сопряжённых сервисов с ограниченными контекстами

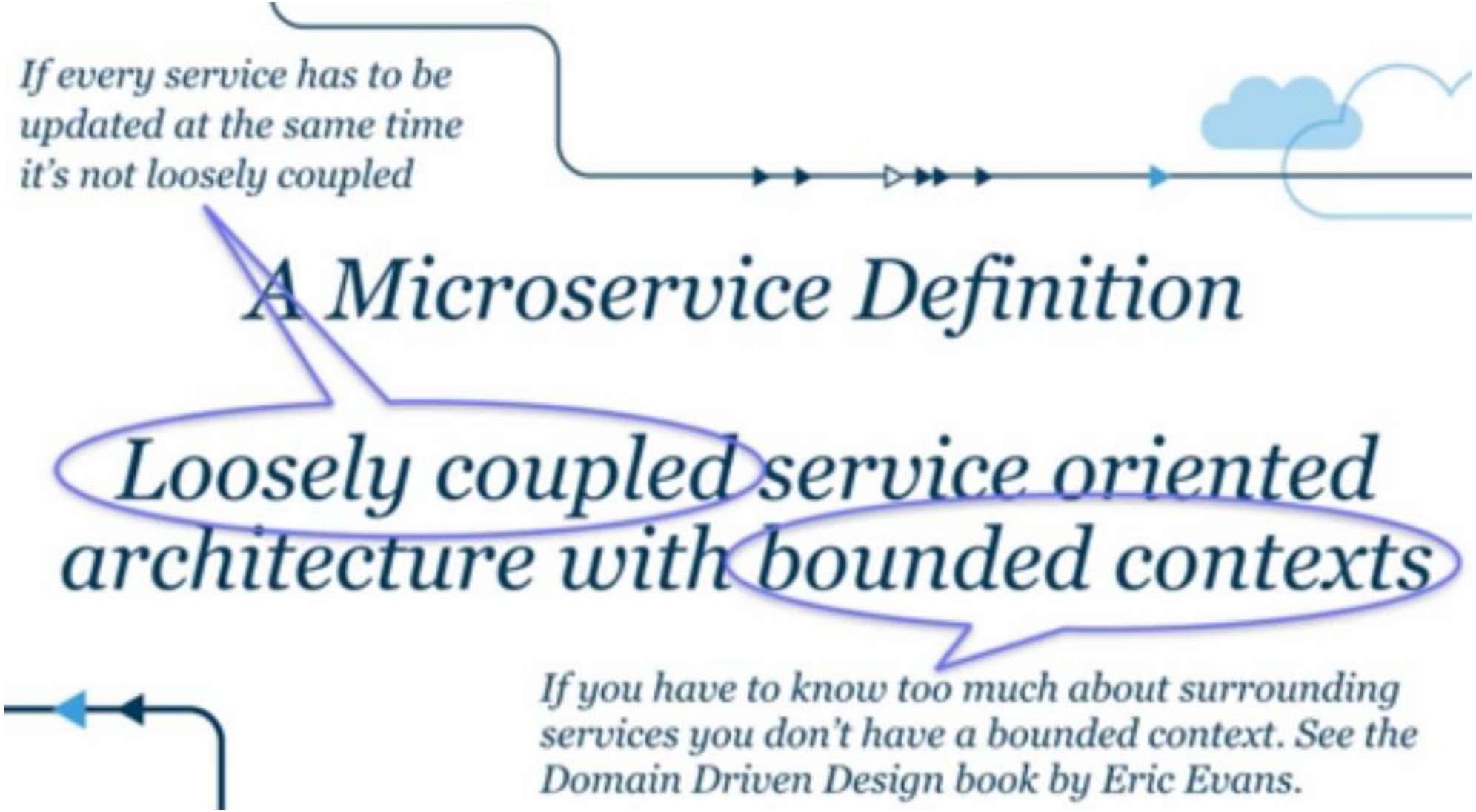
(Loosely coupled service-oriented architecture with bounded contexts)



Эдриан Кокрофт

один из ключевых специалистов в Netflix Cloud и пионер в освоении микросервисов

Уточнение определения



И еще…

Loosely coupled service oriented architecture

with bounded contexts

Adrian Cockcroft

Applications that fit in your head

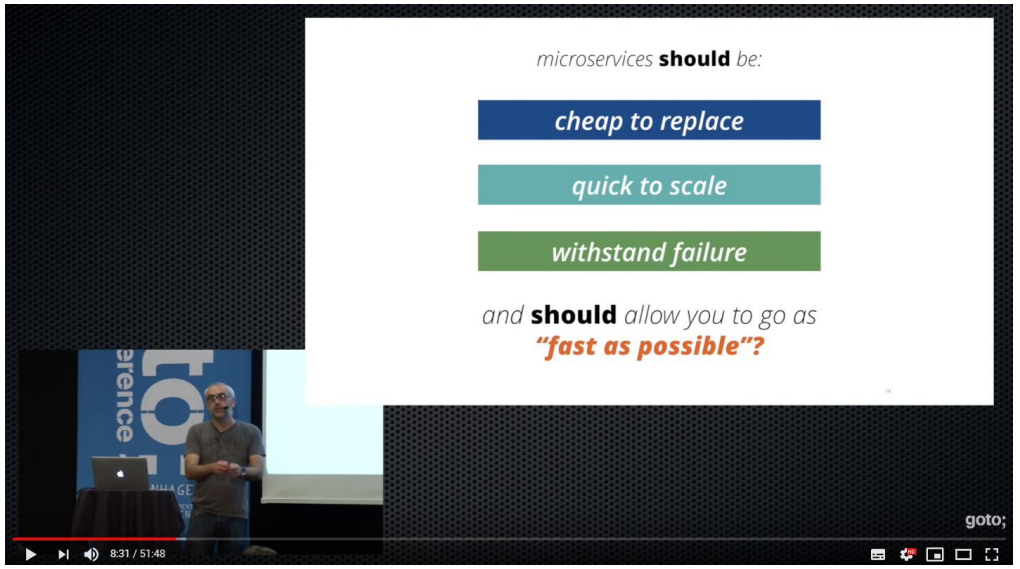
James Lewis

Bounded context + events

Indu Alagarsamy

Микросервис должен

- дёшево заменяться;
- быстро масштабироваться;
- быть устойчивыми к сбоям;
- никоим образом не замедлять нашу работу.



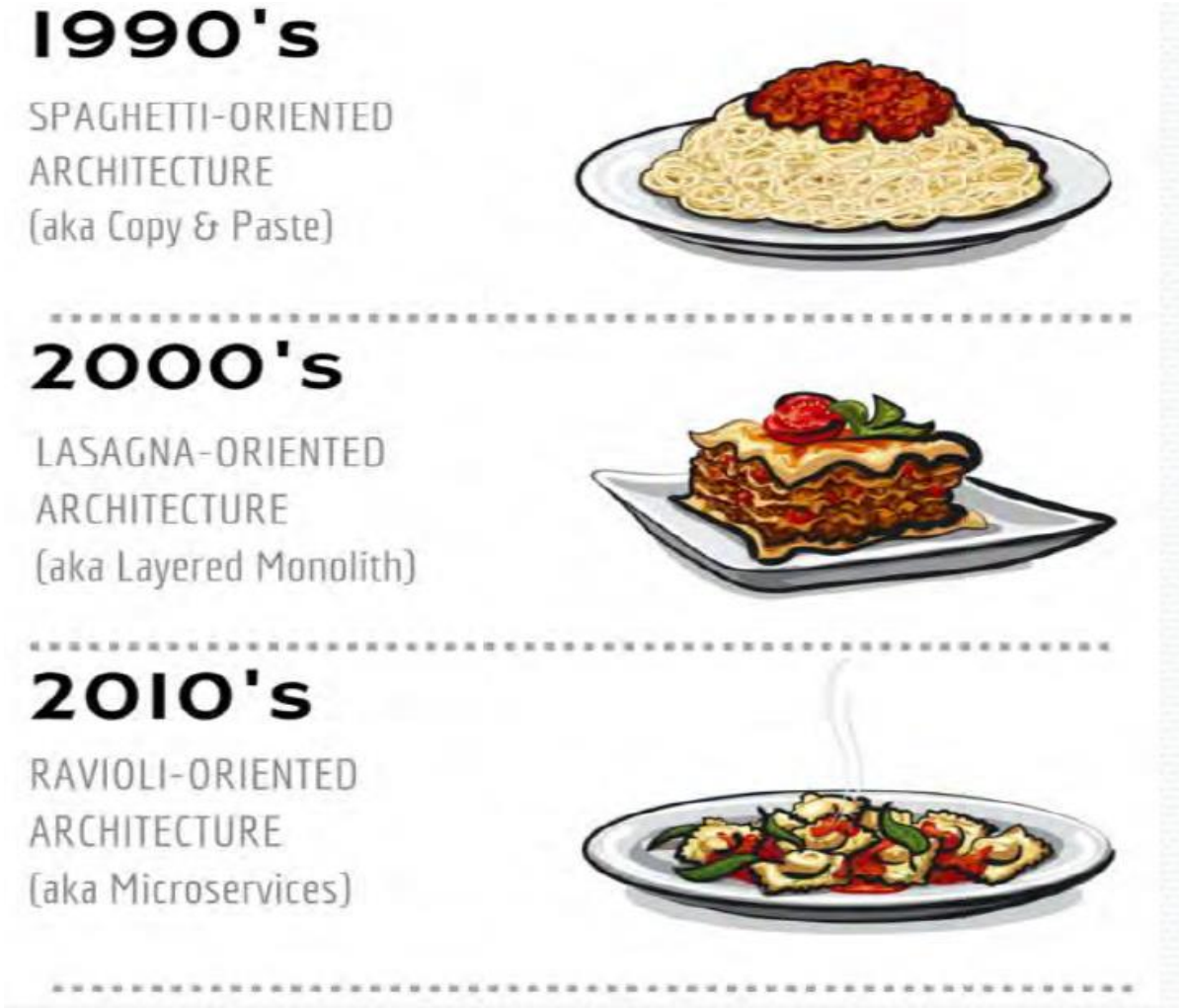
Джеймс Льюис

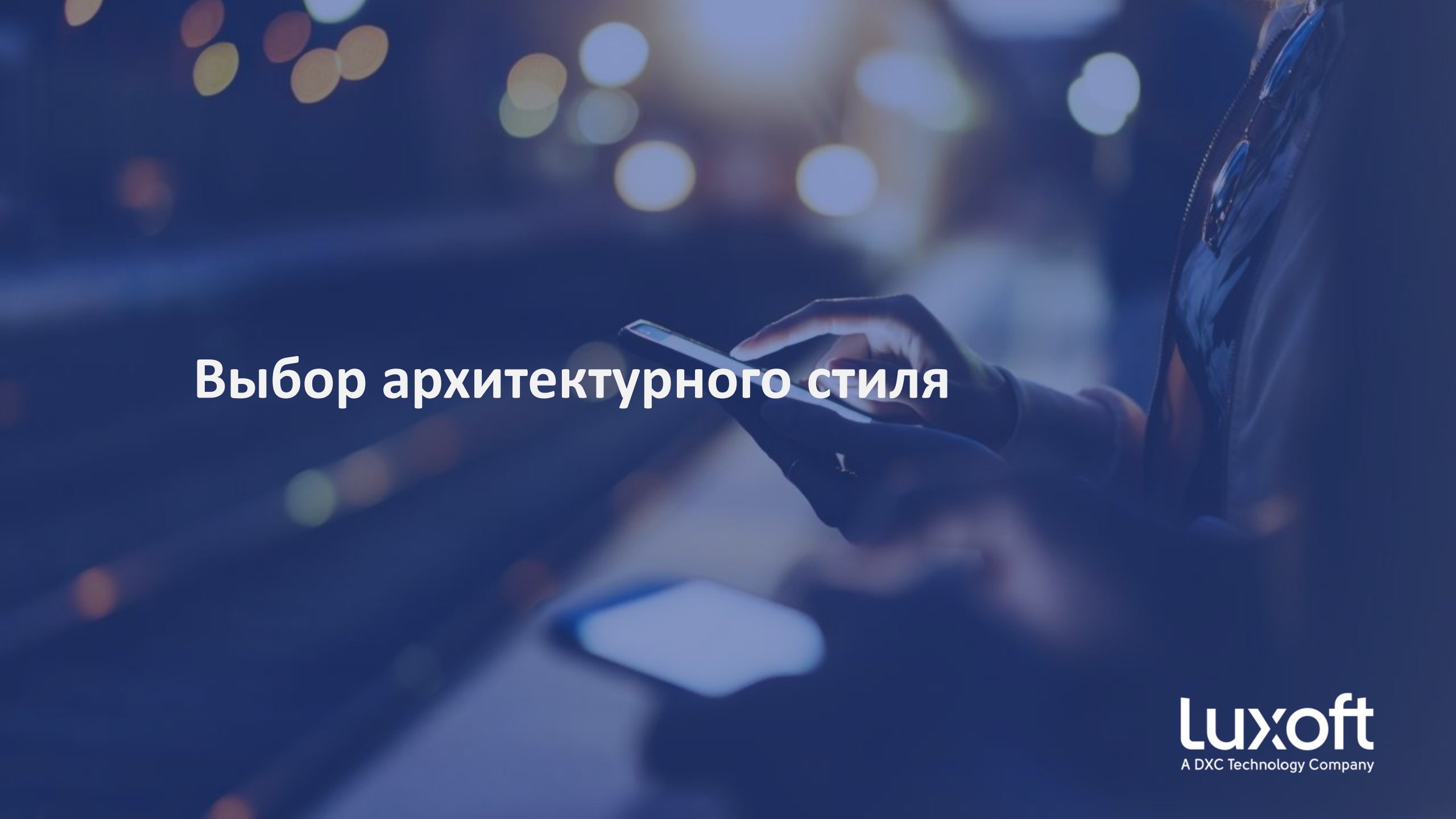
один из ключевых специалистов в Netflix Cloud и пионер в освоении микросервисов

Резюме
MSA это

- **Приложение MSA это**
  - **Набор сервисов в котором каждый сервис**
    - **работает в своем собственном процессе;**
    - **взаимодействует посредством облегченных механизмов;**
    - **построен на бизнес-возможностях;**
    - **может быть развернут независимо с помощью полностью автоматизированного механизма развертывания.**
  - **минимальный уровень централизованного управления этими сервисами, который позволяет**
    - **использовать разные языки программирования;**
    - **использовать разные технологии хранения данных.**

Архитектура равиоли





Выбор архитектурного стиля

Luxoft
A DXC Technology Company

Programmers know the benefits of everything and

the trade-offs of nothing

Rich Hickey (Designer of Clojure)

Проблема выбора

- **Микросервисная архитектура не является серебряной пулей**
  - **Решает вполне конкретные задачи**
  - **Влечет большое количество проблем**

- **Когда стоит использовать MSA и когда стоит отказаться от этого решения?**

Преимущества MSA

- Независимость проектирования, разработки, тестирования, развертывания и поставки
  - Улучшение масштабирования
  - Возможности улучшения надежности (отказоустойчивости)
- Привлечение большого количества независимых разработчиков
- Разработчики понимают свою часть системы
- Изоляция процессов позволяет варьировать технологический стек
  - языки
  - стили
  - платформы
  - базы данных
- Независимость в процессе разработки обеспечивает гибкость

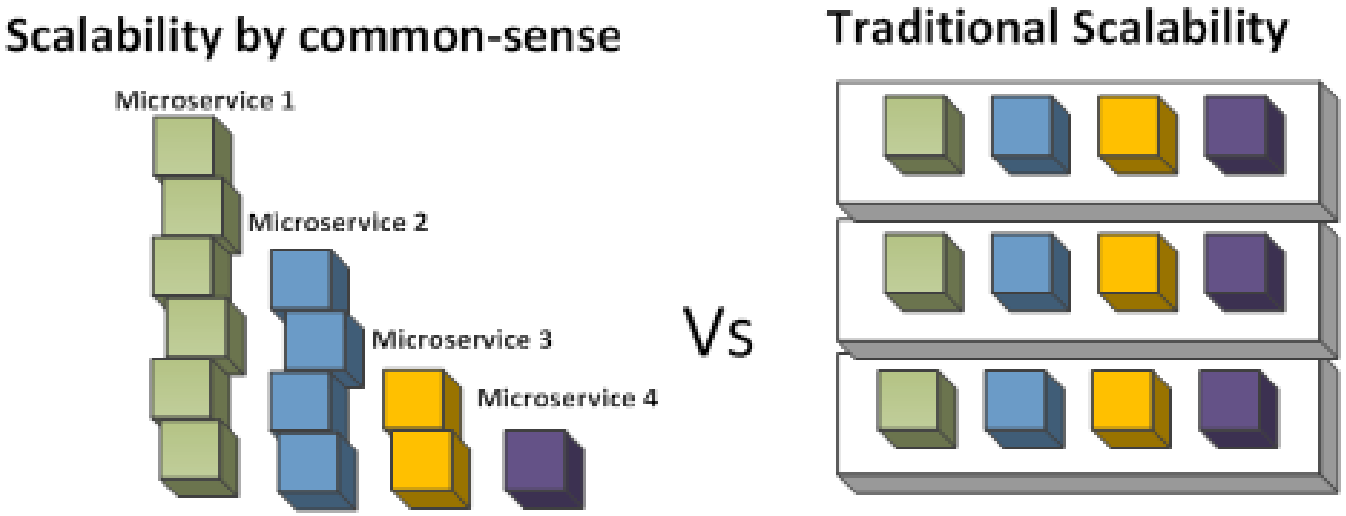
Независимое развертывание

- Нет нужды координировать поставку и развертывание разного функционала
- Быстрый запуск
- Сбой при развертывании не приводит к отказу системы
- Возможно непрерывное развертывание (CD)

- Но, в связи с большим количеством разнородных компонент, процесс развертывания в целом становится достаточно сложным и требует автоматизации
- Кроме того усложняются процессы тестирования и сопровождения

Улучшение масштабирования

- Каждая служба может быть развернута в том количестве, которое действительно необходимо
- Специализация сервисов позволяет предоставлять им только необходимые ресурсы



- Но, переход от гигантских мэйнфреймов к множеству слабых машин приводит к сетевым проблемам

  Соединение по сети медленно и ненадежно

First Law of Distributed Object Design

Don't distribute your objects

Martin Fowler

Улучшение отказоустойчивости

- **Возможность независимого масштабирования и развертывания позволяет повышать избыточность отдельного сервиса и при необходимости перезапускать его без остановки всей системы**

- **Но, необходимо учитывать то, что**
  - **каждый сетевой вызов может закончится неудачей**
  - **любая служба, с которой вы взаимодействуете, может перестать отвечать**
- **Кроме этого распределение данных в сети делает проблемой обеспечение согласованности данных**

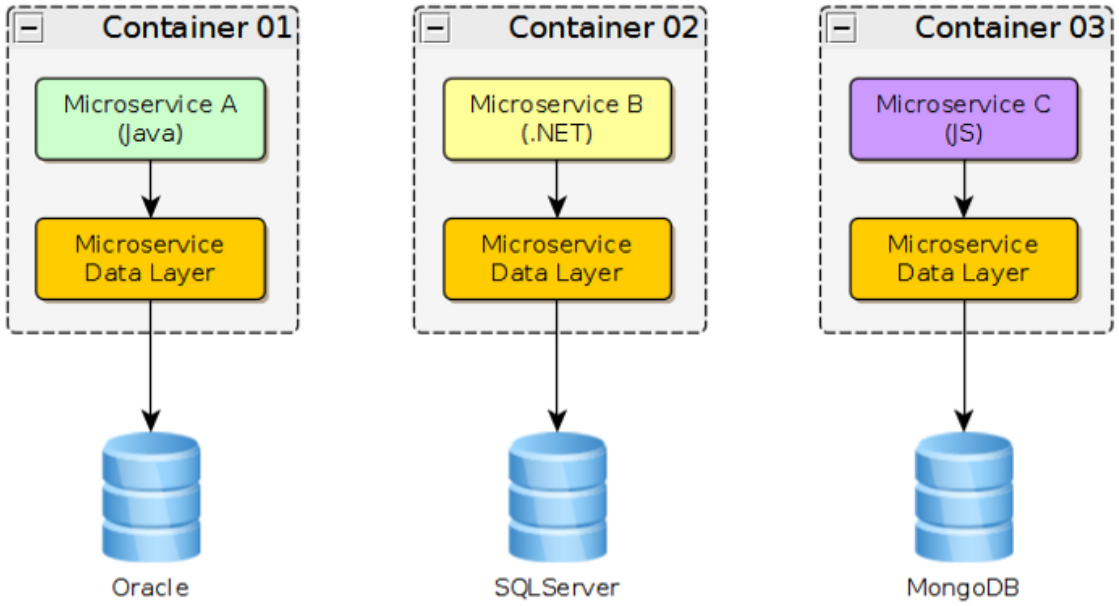
Гибкость и ясность

- **Жесткие физические границы сервиса обеспечивают**
  - **Низкую связанность**
  - **Управляемые части**
  - **Независимую разработку**
- **Разработчики хорошо понимают свою часть системы**

- **Но, независимость разработчиков приводит к эффекту колодца, когда никто не видит картины целиком**

Варьирование технологического стека

- Каждый сервис может использовать наиболее подходящие под его условия технологии и стили
- Ошибочное архитектурное решение не фатально, так как может быть исправлено в кратчайшие сроки



- Но, наличие множества подходов и технологий делает сервисы плохо отчуждаемыми
- Кроме того, каждая новая технология эта новая возможность выстрелить себе в ногу

Техническая сложность

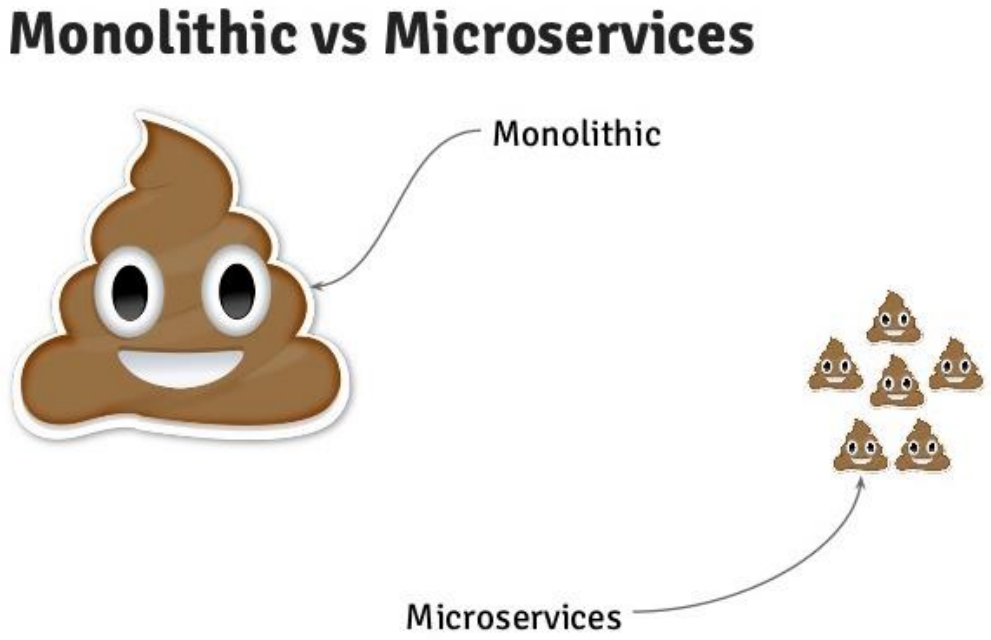
- **Исходя из всего вышесказанного можно сделать вывод, что разработка микросервисов сложная техническая задача, требующая особых навыков**
  - **разработчиков,**
  - **технических специалистов (DevOps),**
  - **тестировщиков.**
- **А так же, определенной организационной перестройки**

Microservices – Not A Free Lunch!

Benjamin Wootton, CTO of Contino

Распространённые заблуждения

- Код будет чище.
- Писать модули, решающие одну задачу — легче.
- Это работает быстрее, чем монолит.
- Инженерам проще, если не нужно работать с единой кодовой базой.
- Это самый простой способ обеспечить автоматическое масштабирование.
  - И тут где-то замешан Докер.



Microservices buy you options

James Lewis (Principal Consultant at ThoughtWorks)

Выбор стиля

- **является ли микросервисная архитектура хорошим выбором для системы, над которой вы работаете?**

any decent answer to an interesting question begins, "it depends..."

Kent Beck

Кому стоит переходить на микросервисы

- **Если ваша система достаточна сложна и очень важно обеспечить контроль на ее компонентами.**
- **Если существенное значение имеет время выхода на рынок очередной функции (time to market).**
- **Если вы работает в agile процессе и практикуете эволюционную архитектуру.**

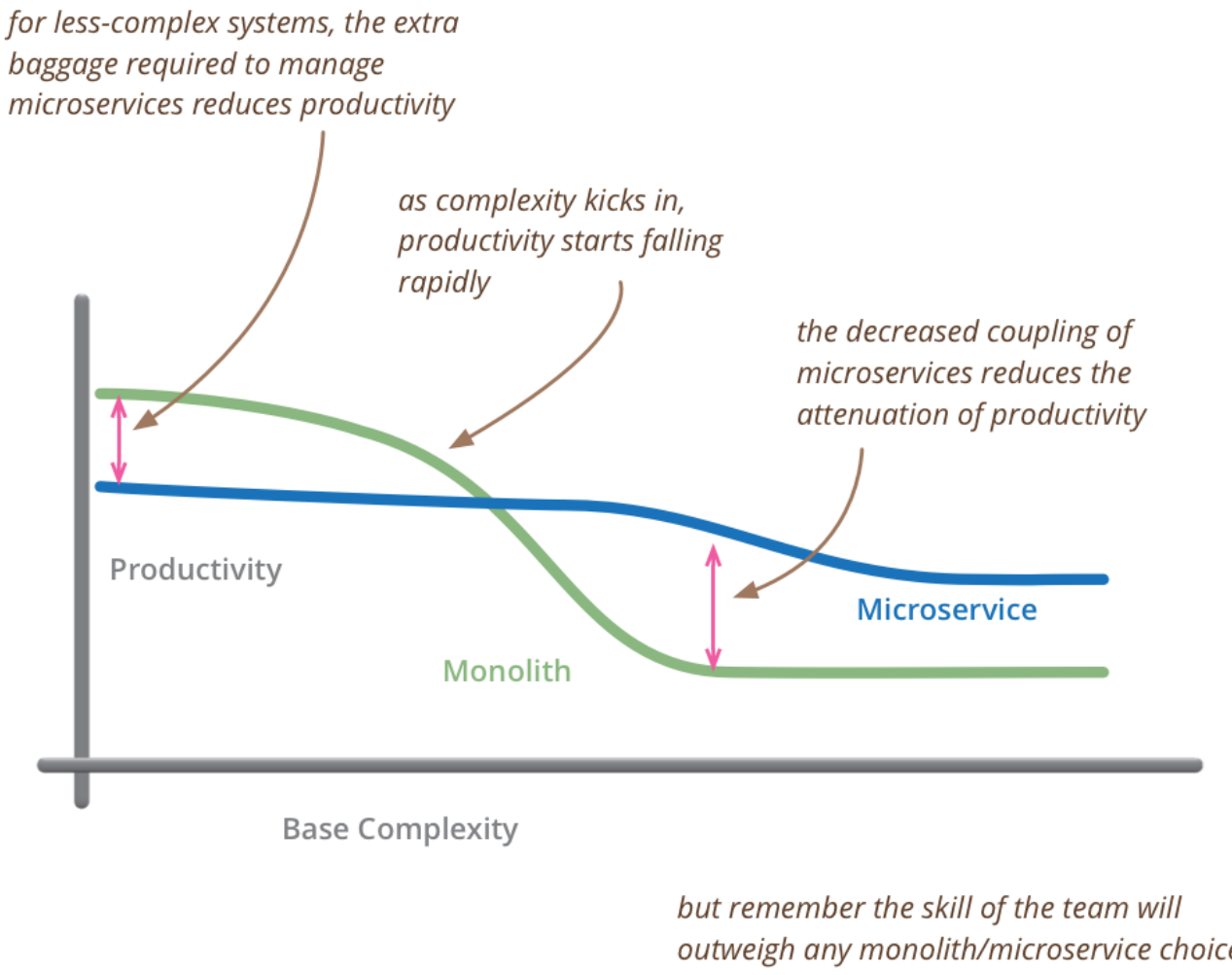
Когда не стоит использовать микросервисы ?

- Приложение слишком мало по объему.
- Предметная область не ясна и не определена.
- Организация не готова к перестройке.
- Вы не готовы идти на компромиссы по производительности.
- Нет специалистов, понимающих концепты MSA (в частности концепты DDD).
- Команда не созрела для такой работы.

Приложение слишком мало по объему

даже не рассматривайте микросервисы, если у вас нет системы, которая слишком сложна, чтобы управлять ею как монолитом

Мартин Фаулер

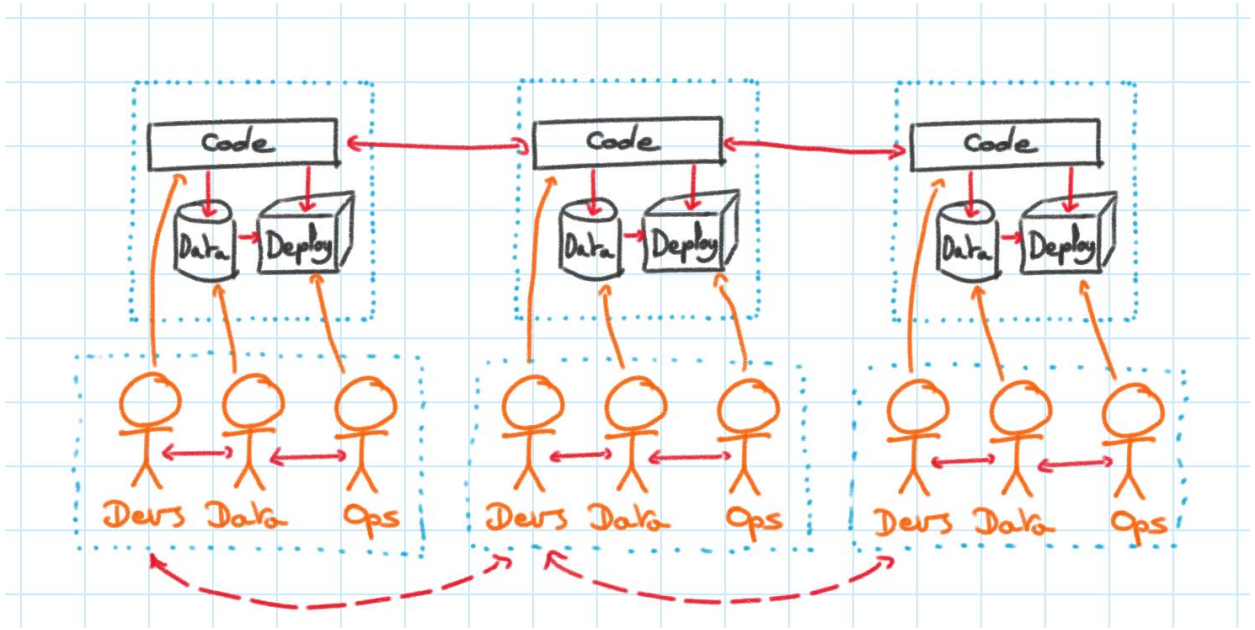


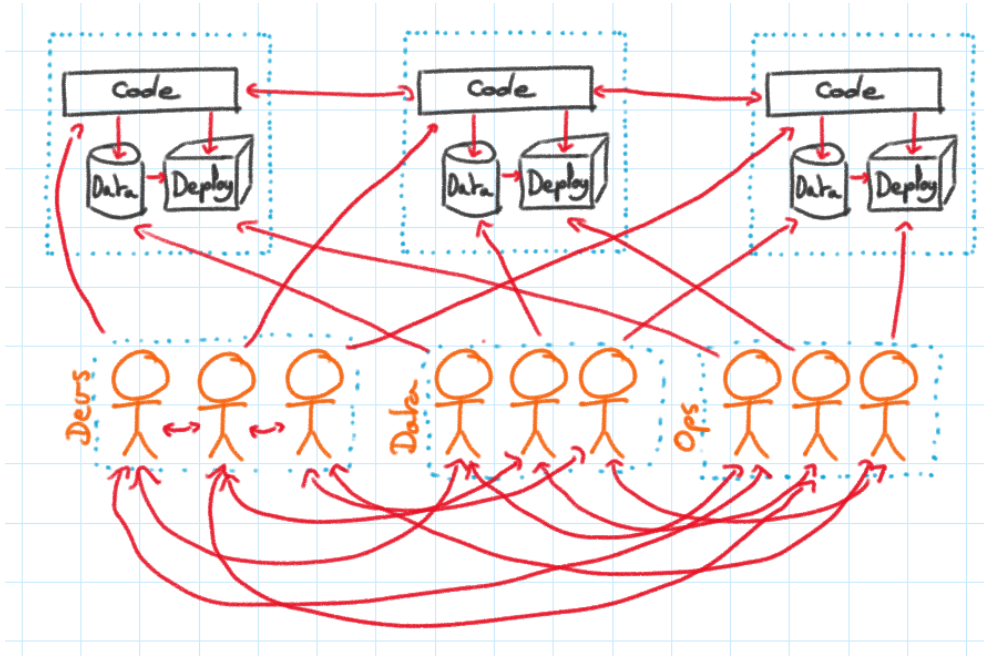
Предметная область не ясна и не определена

- Если предметная область является не ясной, невозможно правильно определить границы сервисов
- Перераспределение ответственности между сервисами, выполняющихся в разных средах и поддерживаемых разными командами – очень сложная задача
- В это случае проще начать с монолита

Организация не готова к перестройке

- **Распределение ответственностей в разных проектах**





Нет специалистов понимающих концепты MSA

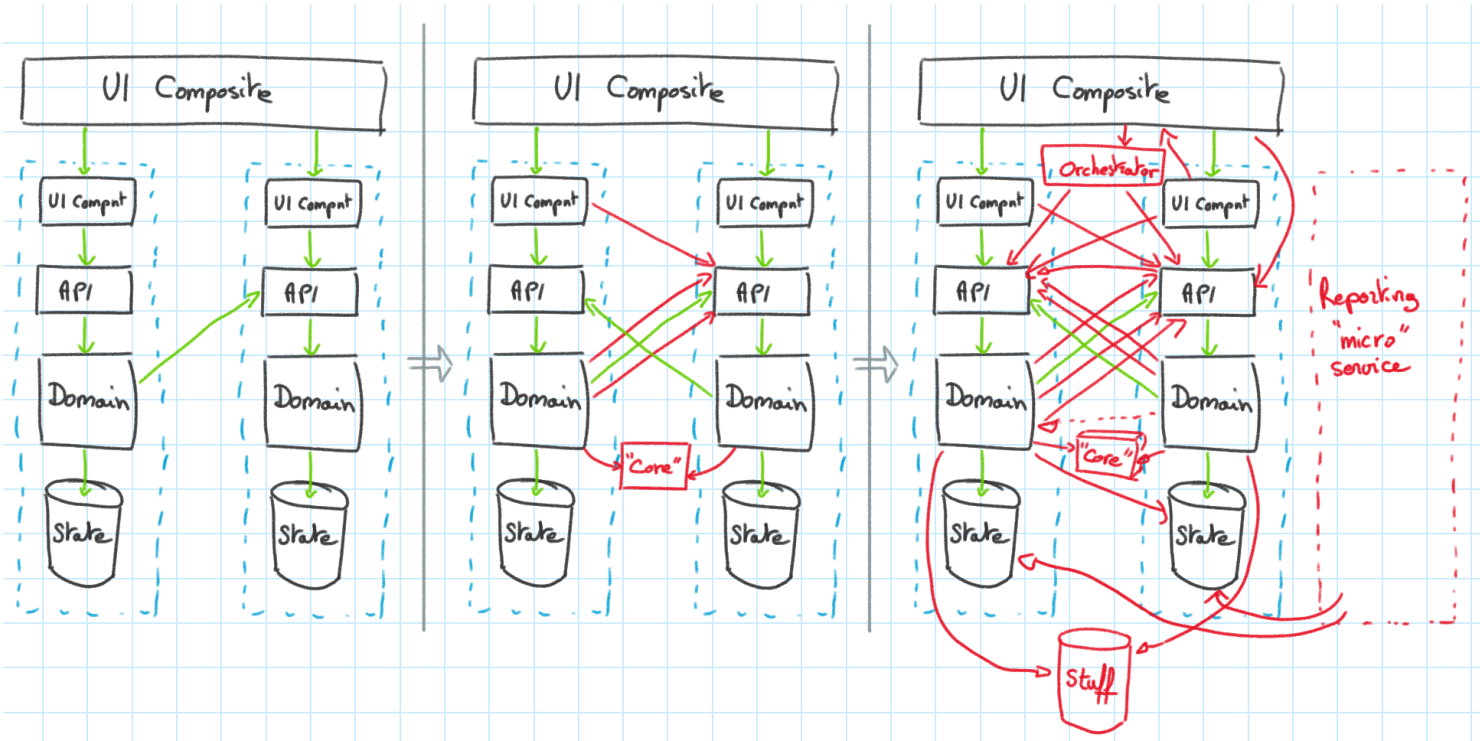
- Зачастую никто даже не может определить термин «микросервис»
- Зачастую это не является сдерживающим фактором
- Важно
  - Готовы ли специалисты к обучению
  - Готова ли организация вкладывать средства в развитие специалистов

Fail often. Fail early. Fail fast. Embracing failure is a means to a successful end. Instead of fearing failure, become empowered by it.

Gary Burnison, CEO of Korn Ferry

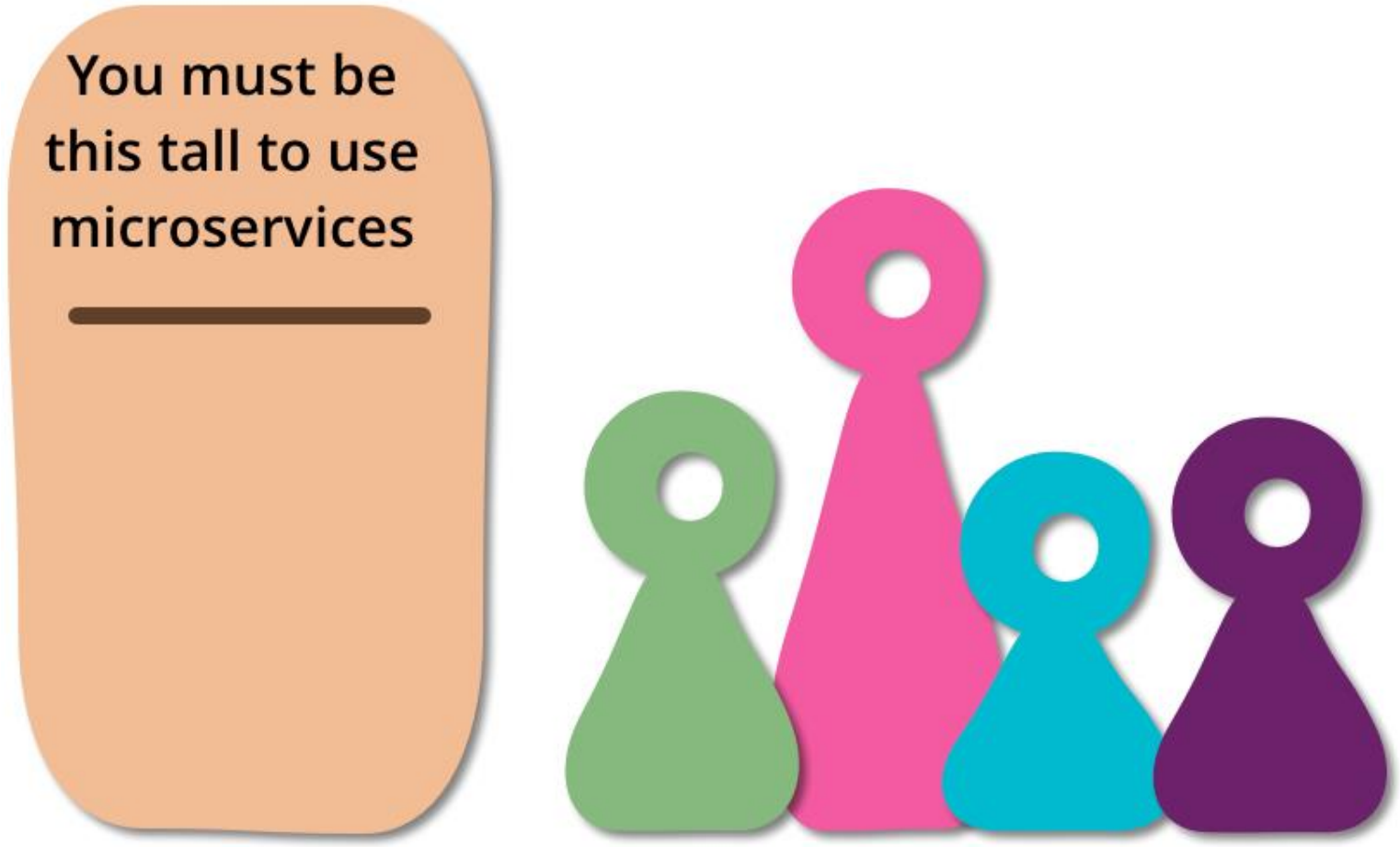
Команда не созрела

- Низкая степень зрелости команды приводит к частому желанию «срезать углы»
- Архитектурные решения откатываются к хорошо знакомым и опробованным шаблонам
- В случае микросервисов это часто приводит к анти-паттерну «распределенный монолит»



Когда стоит использовать микросервисы ?

Прежде всего *«Когда вы, как инженерная организация, будете готовы»*



Буду рад ответить на ваши вопросы