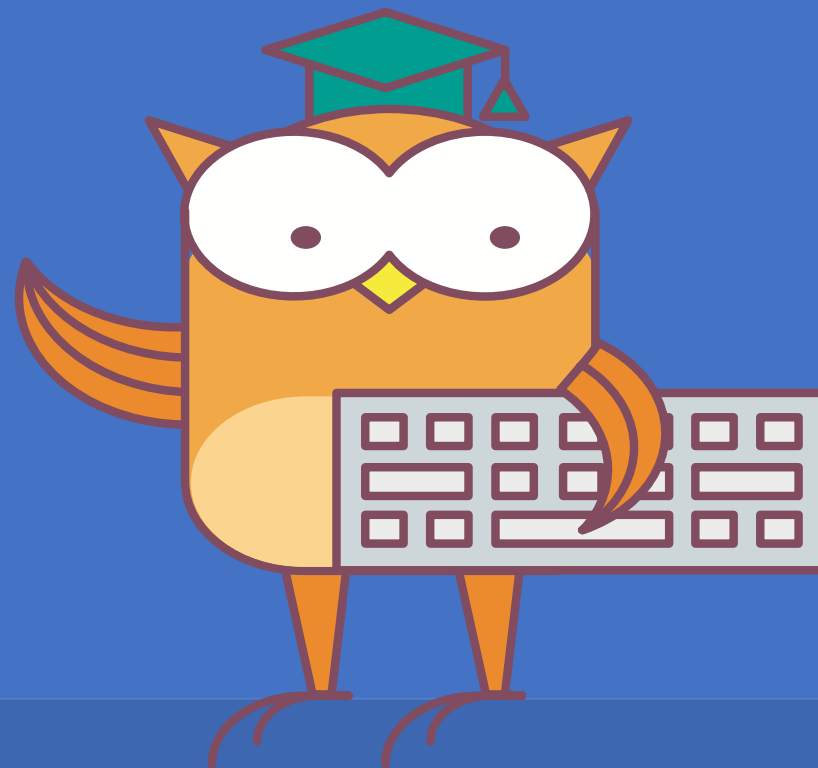
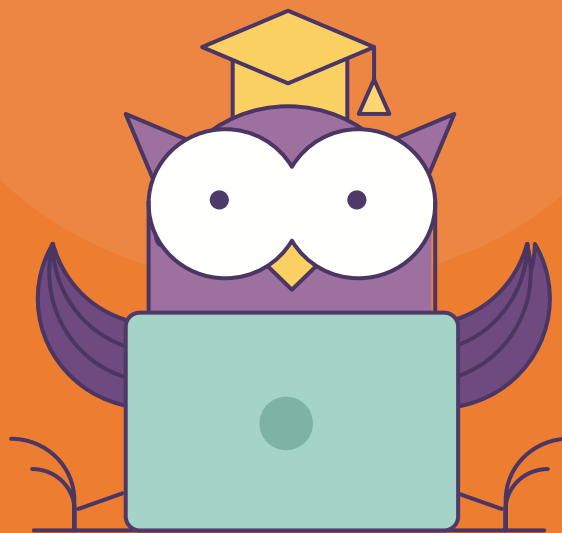


Паттерны декомпозиции сервисов

Архитектор ПО



Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте  если все хорошо

Преподаватель



Непомнящий Евгений

- 15 лет программировал контроллеры на C++ и руководил отделом разработки
- 3 года пишу на Java
- Последнее время пишу микросервисы на Java в Мвидео
- Телеграм @EvgeniyN

Правила вебинара



Активно участвуем



Задаем вопросы в чат



Off-topic обсуждаем в Slack #канал группы или #general



Вопросы вижу в чате, могу ответить не сразу

Карта курса



Опрос по программе - каждый месяц в ЛК

Начало пути

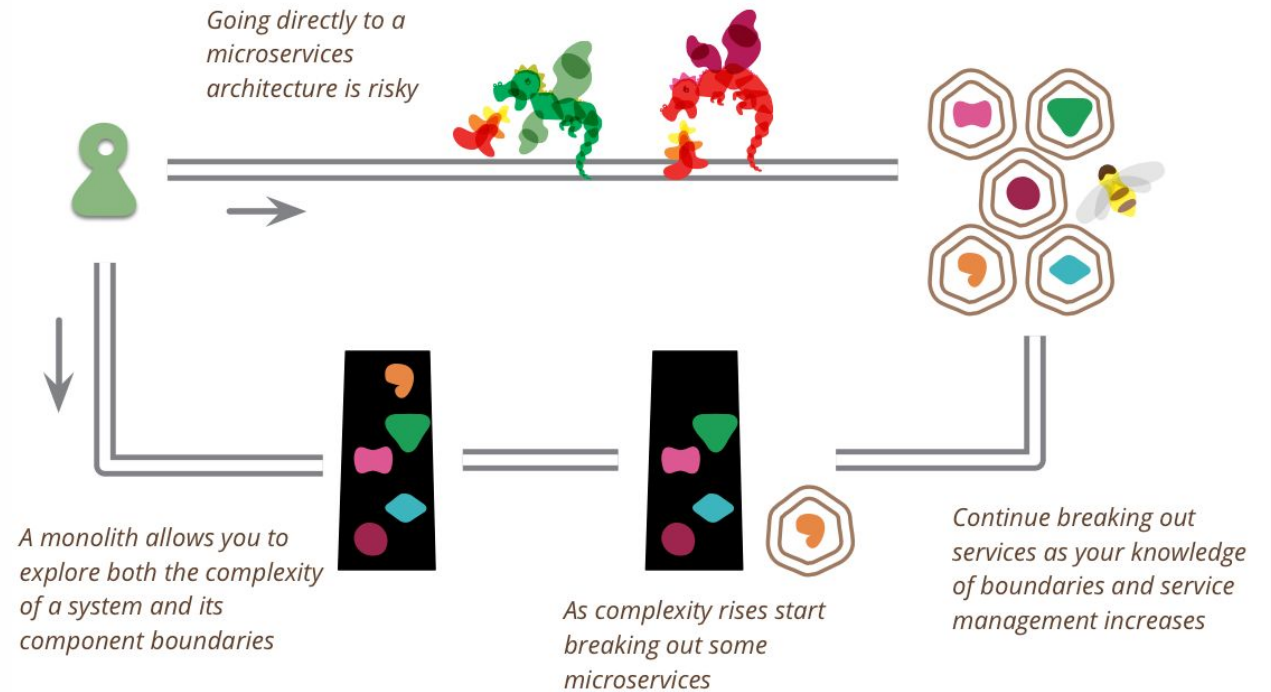
1. “Хотелки” бизнеса
2. Аналитика
3. ... ← *мы здесь*
4. Разработка
5. Тестирование
6. Готовое приложение

01

Monolith first

Monolith first

1. Относительно небольшая команда
2. Относительно небольшая (или равномерная) нагрузка
3. Высокая неопределенность



<https://martinfowler.com/bliki/MonolithFirst.html>

Monolith first

- Почти все успешные истории микросервисов начались с монолита, который стал слишком большим и был разрушен.
- Почти во всех известных мне случаях, когда микросервисная система строилась с нуля, это привело к серьезным проблемам.
- Не следует начинать новый проект с микросервисами, даже если вы уверены, что ваше приложение будет достаточно большим, чтобы оно того стоило.

Мартин Фаулер



Причины начать с монолита

- YAGNY - Зачем сразу создавать сложную систему. Вдруг она окажется бесполезной.
- Ограниченные контексты
 - Успех микросервисной архитектуры во многом зависит от того насколько правильно вы определили границы сервисов.
 - На первых этапах работы сделать это затруднительно.
 - Проводить рефакторинг удобнее в монолите.

Что дальше делать с монолитом

- Тщательно спроектировать монолит и в дальнейшем разбить его на куски.
- Отщипывать с краев - выделять микросервисы из монолита по мере их выявления.
- Жертвенная архитектура (Sacrificial Architecture), прототип.
 - Полностью заменить имеющийся монолит на микросервисы.
 - Рассматривать монолит как реализацию, которую мы должны будем рано или поздно выбросить, но при этом ничего лучше в текущий момент создать не можем.

Более подробно - на следующем занятии

02

Декомпозиция на микросервисы

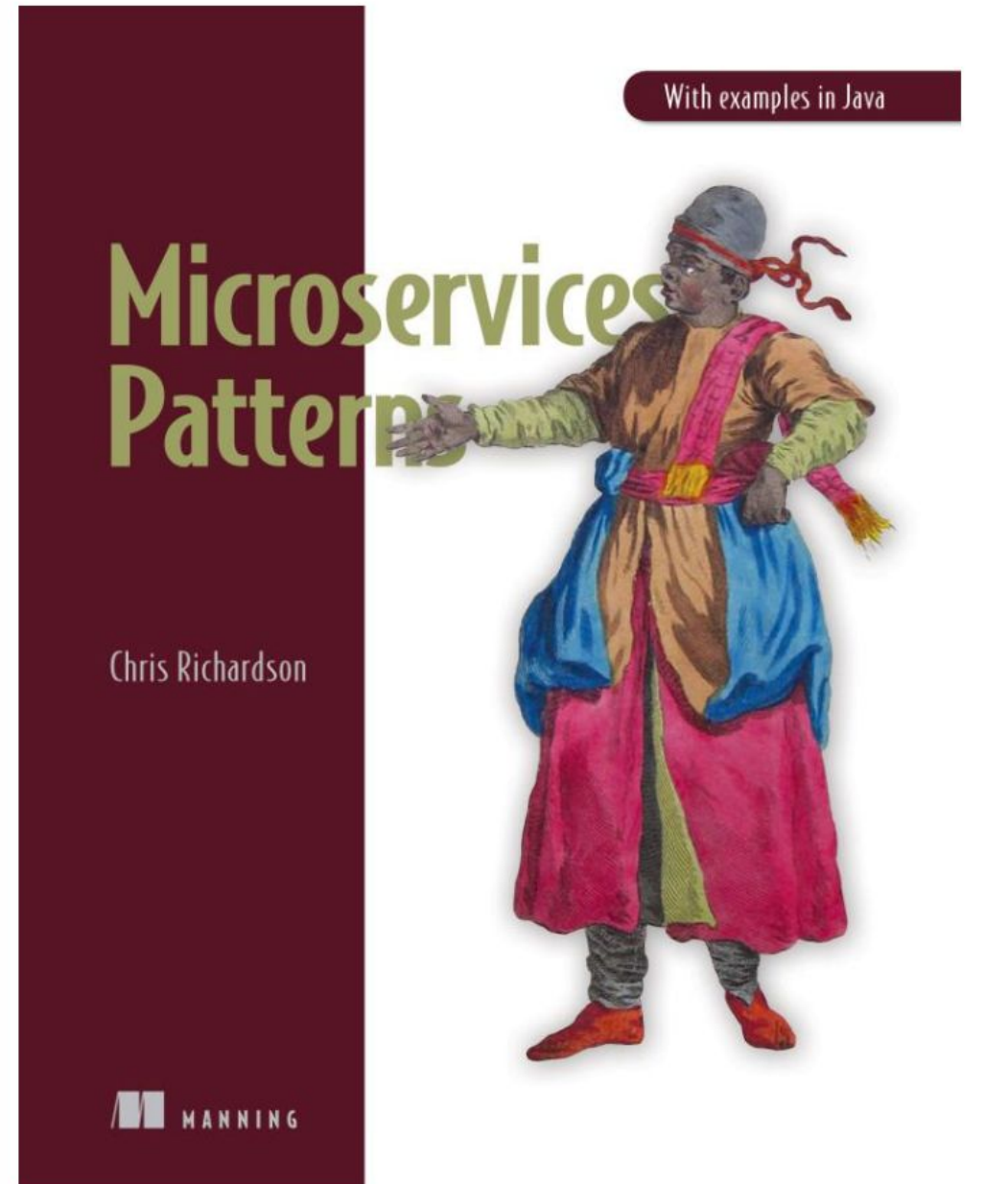
Как декомпозировать на сервисы?

1. Описать сценарии использования
2. Определить системные действия с приложением
3. Построить доменную модель (модели)
4. Построить сервисную модель
5. Протестировать сервисную модель
6. Повторить п3 – п5 пока не станет хорошо
7. Ничего ли мы не забыли?

Пример

Система вроде “яндекс-еда” -
рестораны, курьеры, заказ блюд

FTGO



Как декомпозировать на сервисы?

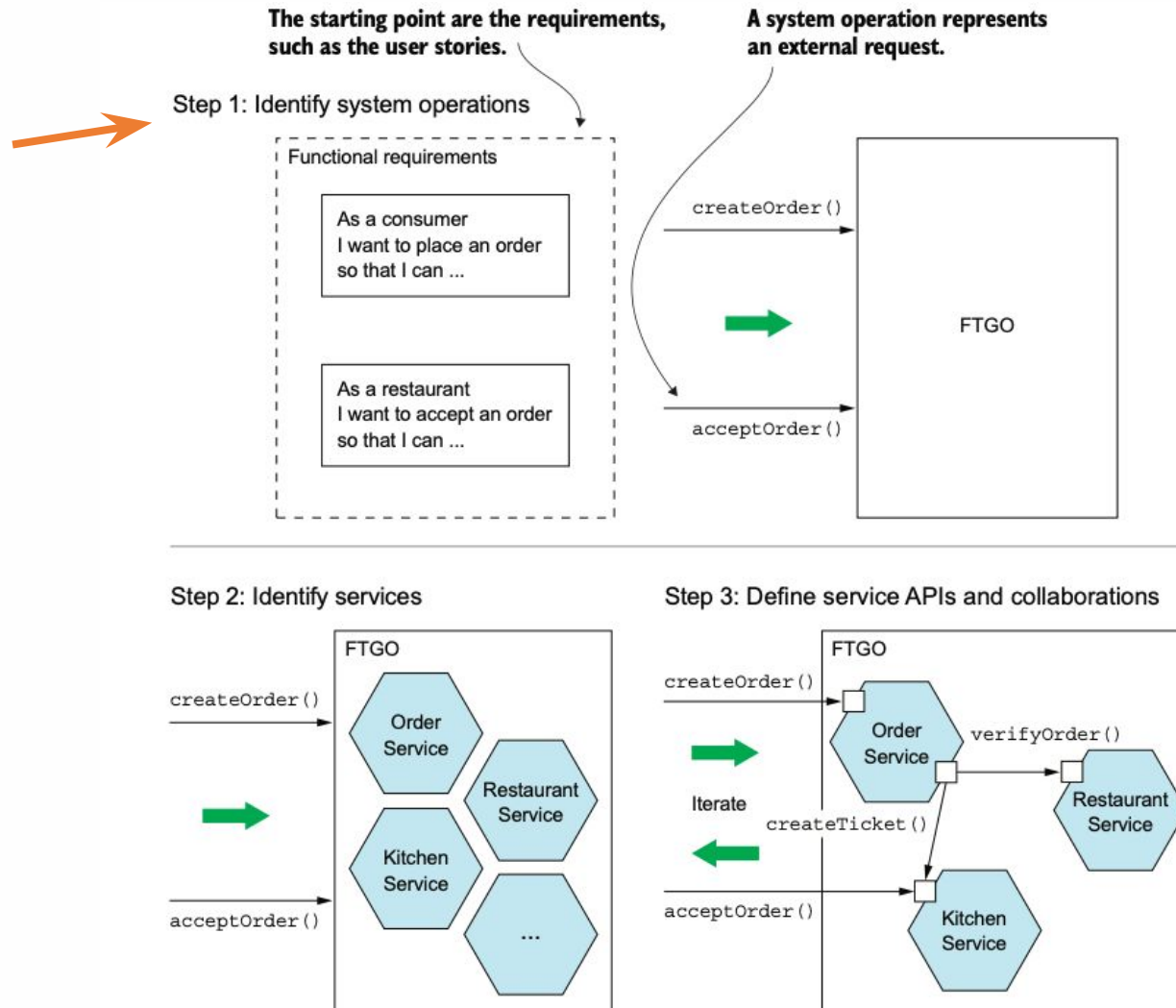


Figure 2.5 A three-step process for defining an application's microservice architecture

03

Пользовательский сценарий

Пользовательский сценарий

Чаще всего пользовательские сценарии описывают в формате BDD/User Story

[википедия](#)

Пользовательский сценарий

Есть авторизованный клиент

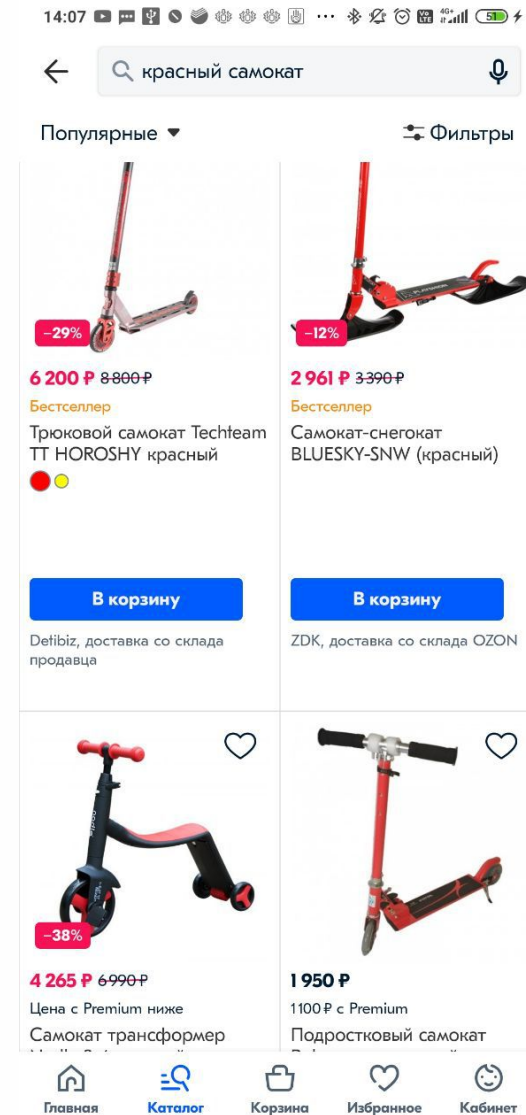
И есть интернет-магазин "Гозон"

И клиент вводит в строке поиска "Красный самокат"

Тогда появляется список товаров, подходящих под запрос клиента

И в карточке товара пользователь видит

- 1) название
- 2) цену
- 3) изображение
- 4) скидку
- 5) доступность в разных магазинах



Пользовательский сценарий

Когда клиент нажимает карточку на товар
Тогда клиент переходит на страничку с описанием товара



04

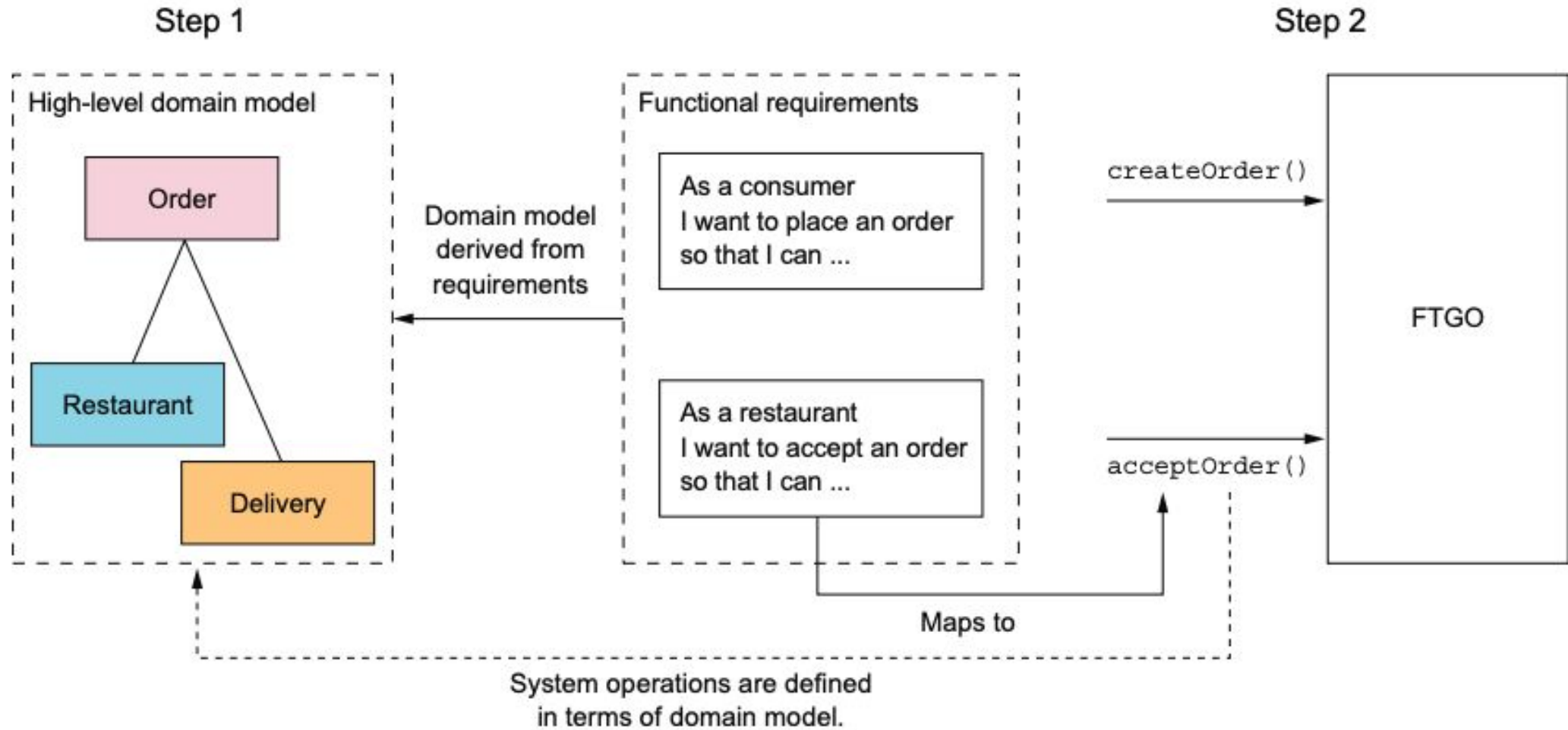
Модель предметной области и
системные операции

Системные действия

Системные действия – это действия, которые совершает пользователь и приложение в рамках пользовательского сценария.

Модель предметной области – это набор инвариантов и ограничений предметной области, описанных в том или ином виде

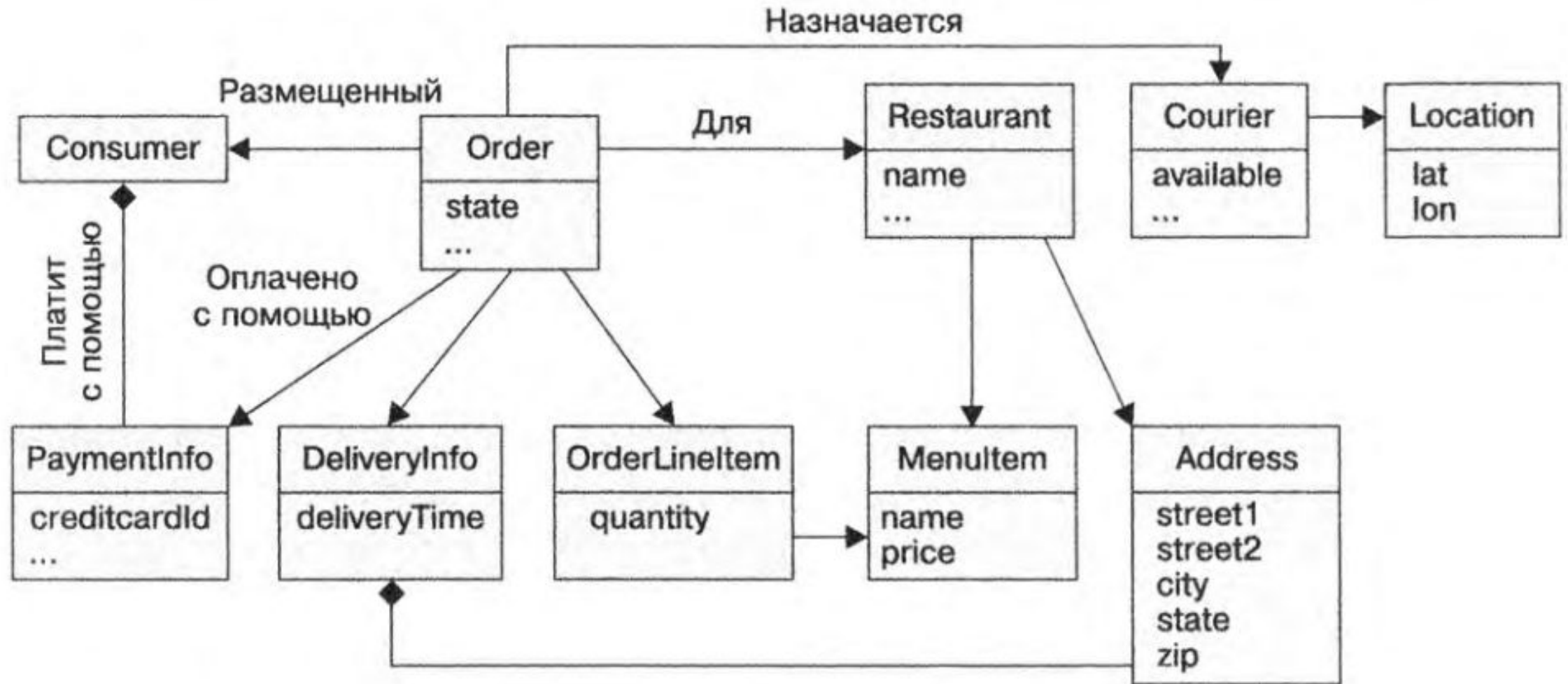
Системные действия



Доменная модель - существительные

- Клиент размещает заказ
- Клиент получает данные о продукте
- Клиент добавляет продукт в заказ
- Клиент может убрать продукт из заказа
- Клиент может оформить заказ
- Клиент выбирает место доставки в заказе
- Клиент выбирает способ оплаты в заказе
- Клиент выбирает время доставки в заказе
- Клиент проводит оплату заказа
- Ресторан принимает заказ
- Приложение резервирует курьера для доставки заказа

Доменная модель - схема



Системные операции

- *команды* — системные операции для создания, обновления и удаления данных;
- *запросы* — системные операции для чтения (запрашивания) данных

Системные операции - глаголы

- Клиент размещает заказ
- Клиент получает данные о продукте
- Клиент добавляет продукт в заказ
- Клиент может убрать продукт из заказа
- Клиент может оформить заказ
- Клиент выбирает место доставки в заказе
- Клиент выбирает способ оплаты в заказе
- Клиент выбирает время доставки в заказе
- Клиент проводит оплату заказа
- Ресторан принимает заказ
- Приложение резервирует курьера для доставки заказа

Системные операции

Действующее лицо	История	Команда	Описание
Клиент	Create Order	createOrder()	Создает заказ
Ресторан	Accept Order	acceptOrder()	Указывает на то, что ресторан принял заказ и обязуется приготовить его к заданному времени
	Order Ready for Pickup	noteOrderReadyForPickup()	Указывает на то, что заказ готов к доставке
Курьер	Update Location	noteUpdatedLocation()	Обновляет текущее местоположение курьера
	Delivery picked up	noteDeliveryPickedUp()	Указывает на то, что курьер взял заказ
	Delivery delivered	noteDeliveryDelivered()	Указывает на то, что курьер доставил заказ

Системные операции - детализация

Операция	createOrder (ID клиента, способ оплаты, адрес доставки, время доставки, ID ресторана, позиции заказа)
Возвращает	orderId...
Предварительные условия	Клиент существует и может размещать заказы. Позиции заказа соответствуют пунктам меню ресторана. Адрес и время доставки выполнимы для ресторана
Окончательные условия	Банковская карта клиента позволила снять сумму заказа. Заказ был создан в состоянии PENDING_ACCEPTANCE

Операция	acceptOrder (restaurantId, orderId, readyByTime)
Возвращает	—
Предварительные условия	order.status равно PENDING_ACCEPTANCE. Курьер доступен для доставки заказа
Окончательные условия	Состояние order.status поменялось на ACCEPTED. Время order.readyByTime поменялось на readyByTime. Курьер назначен для доставки заказа

Как декомпозировать на сервисы?

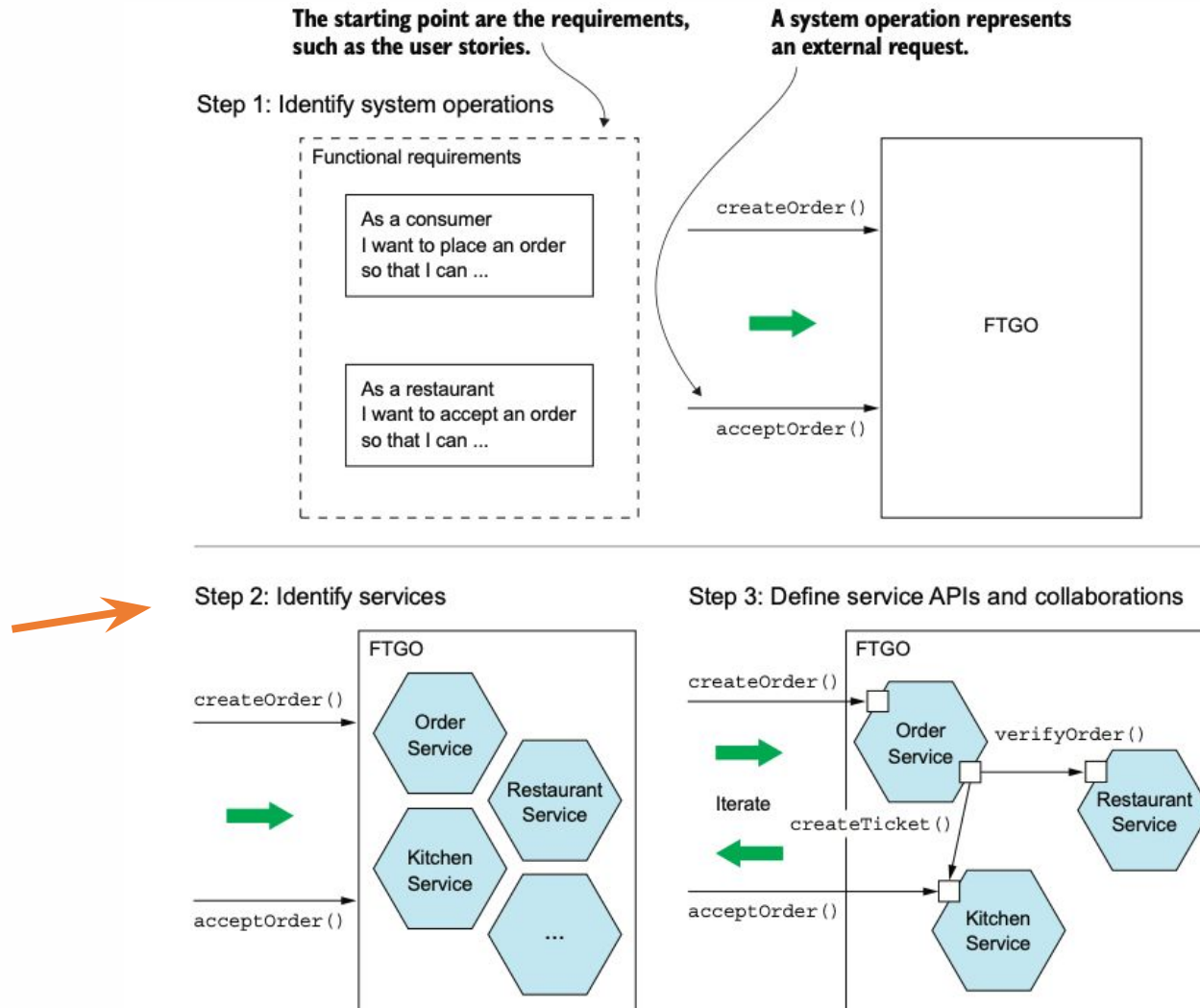
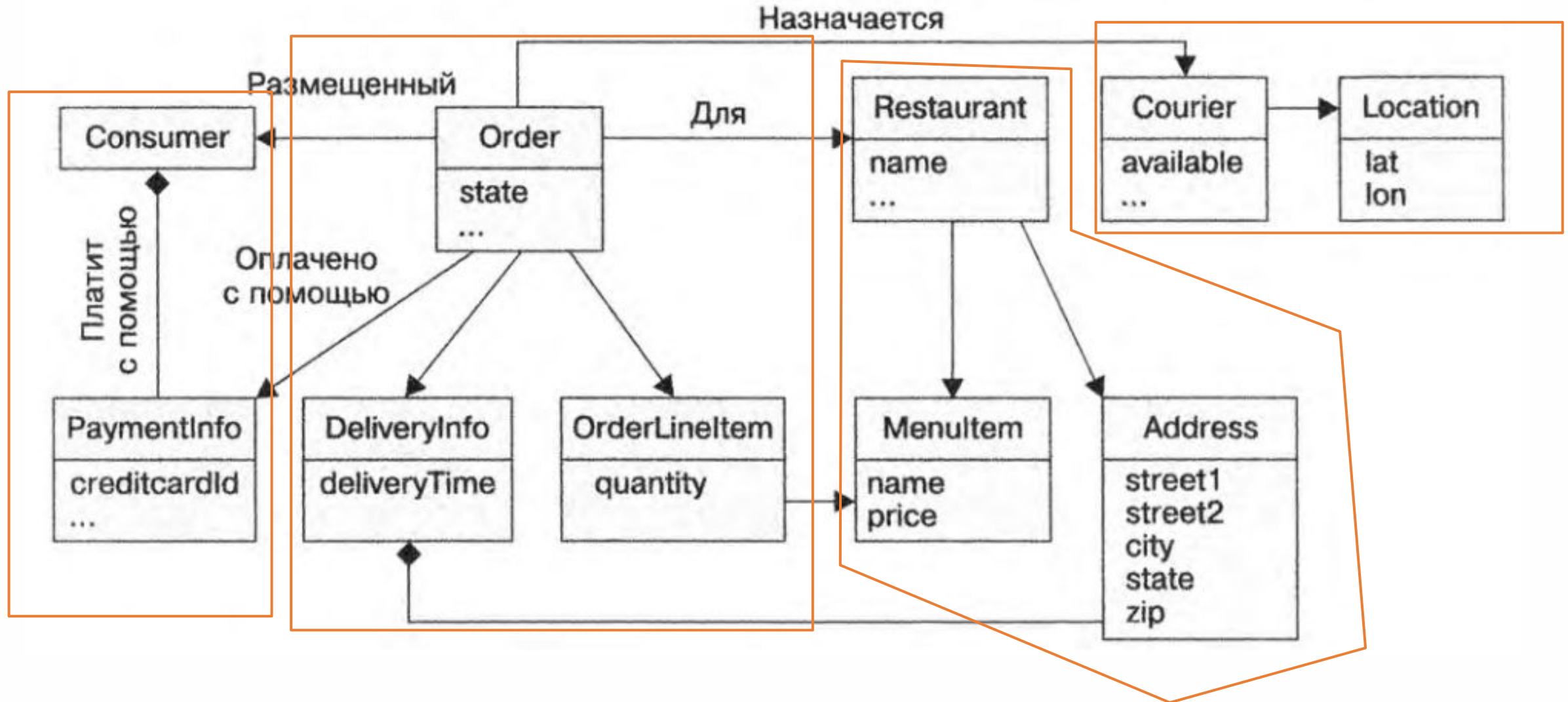


Figure 2.5 A three-step process for defining an application's microservice architecture

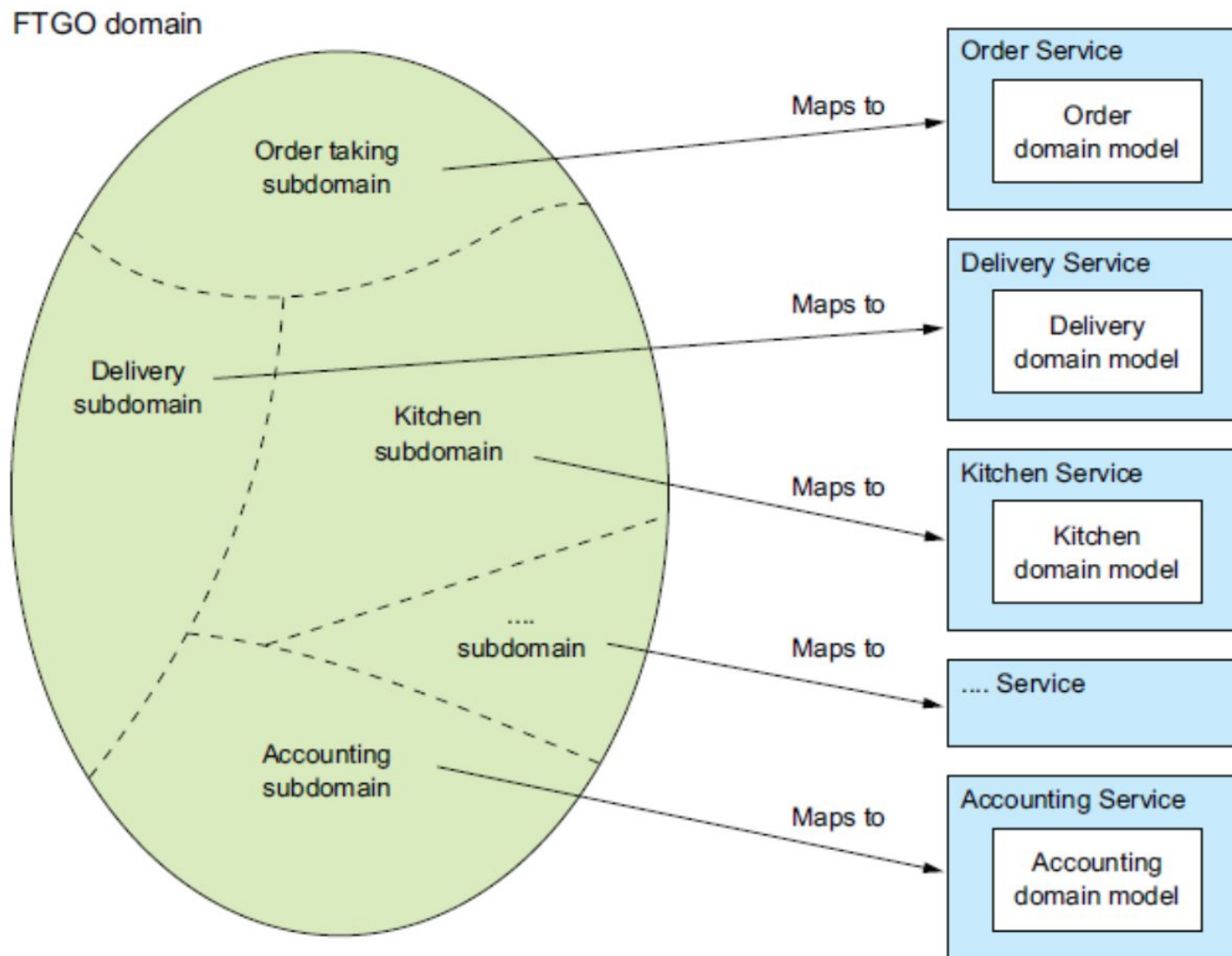
05

Разбиение по агрегатам

Доменная модель - схема



Нулевая итерация модели предметной области



Эрик Эванс - DDD

В целом мы можем добавлять сущностей или делать модель более детализированной.

Чтобы перейти к моделированию сервисов, для сложных предметных областей, мы должны каким-то образом провести границы, по которым сервисы будут проходить.

В DDD есть соответствующие паттерны для проведения таких границ – модули, агрегаты и сервисы.

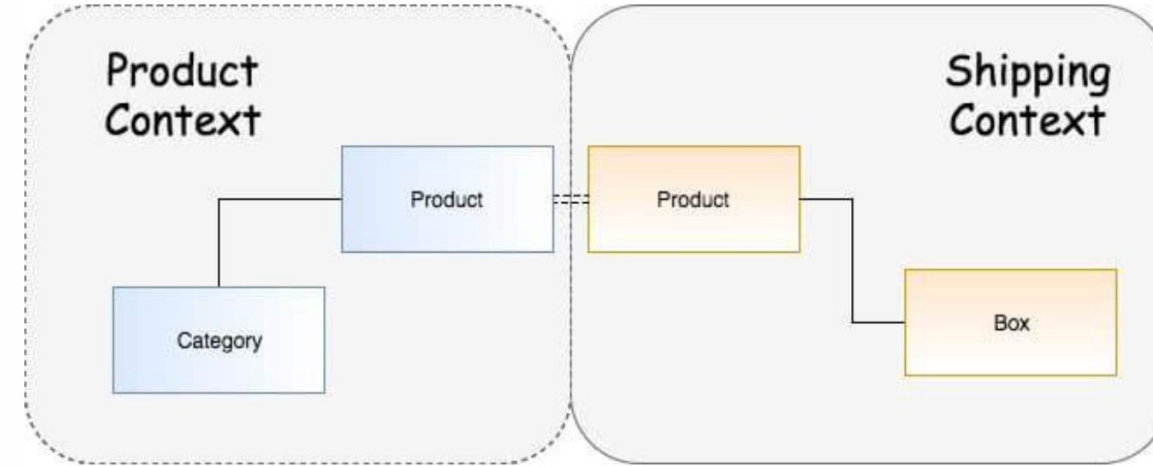
Нарисовать единую картину предметной области - очень сложно и часто невыполнимо

Давайте разобьем на подобласти и в каждой из них будет свой язык (Ubiquitous Language) и свои части данных

DDD у вас будет более подробно в следующем модуле

Ограниченный контекст (Bounded context)

- Выделенные поддомены ограничивают при реализации системы (ограниченный контекст).
- Ограниченные контексты используются для создания переборок в больших и сложных системах.
- Подобно переборкам в корпусе корабля, которые предотвращают затопление всего корабля, переборка в системе предотвращает утечку ненужной сложности за пределы контекстной границы.



06

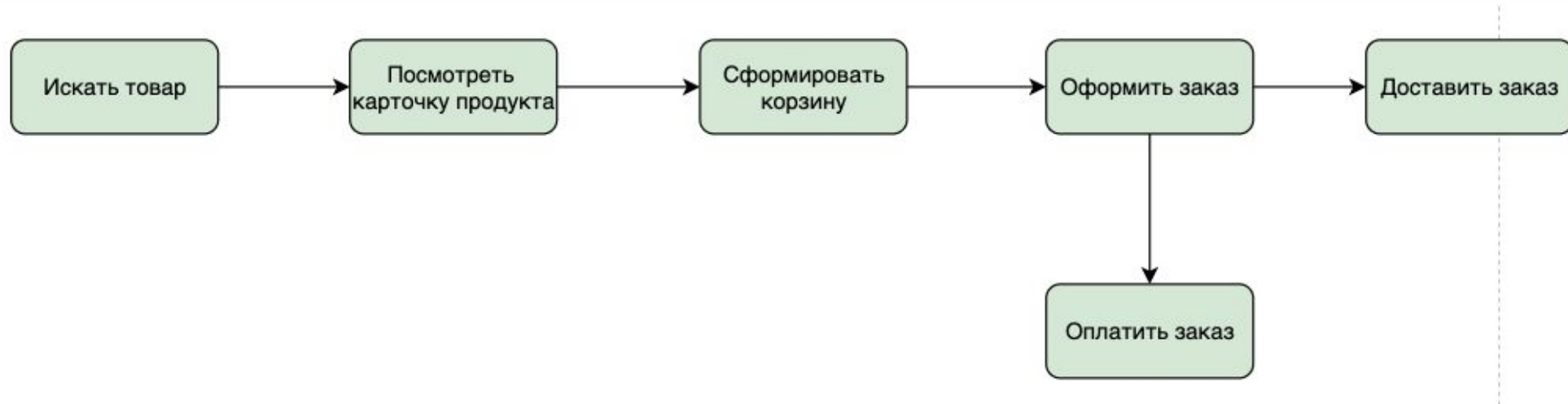
Функциональная модель

Функциональное моделирование

Моделировать можно не только как набор сущностей, но и как просто некий флоу – в «функциональном»/процедурном стиле.

Функциональное моделирование

В качестве действий использовать использовать какие-то «крупные» бизнес-действия.



Функциональное моделирование

В качестве первого приближения для разбиения на сервисы использовать каждый блок – как отдельный сервис или набор связанных сервисов.

В данном случае сервисами в сервисной модели будут:

- Поиск блюда
- Каталог блюд
- Сформировать корзину
- Оформление заказа
- Доставка заказа
- Оплата заказа
- Отмена заказа

07

Разбиение по бизнес-
ВОЗМОЖНОСТЯМ

Бизнес-возможности

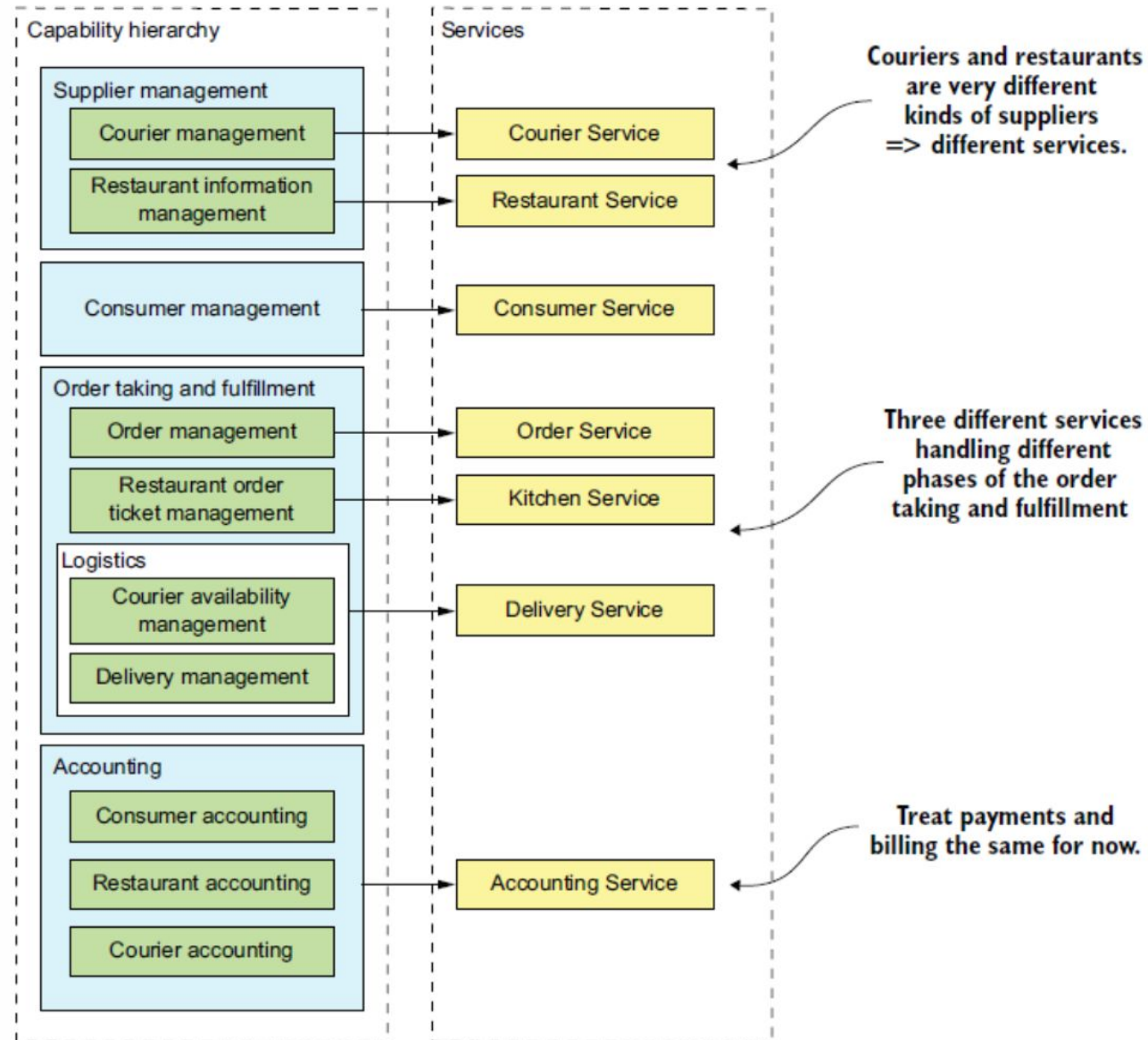
Бизнес-возможности организации описывают то, чем она является.

Обычно они стабильны, в отличие от того, как организация ведет свой бизнес (этот аспект со временем меняется, иногда до неузнаваемости).

Бизнес-возможности

- Управление поставщиками:
 - управление курьерами
 - управление информацией о ресторанах
- Управление клиентами
- Прием и выполнение заказов:
 - управление заказами со стороны клиентов
 - управление заказами в ресторане
 - логистика
 - управление доступностью курьеров
- Бухучет

Бизнес-возможности



08

Разбиение по транзакциям

Транзакции и MSA

- Важное ограничение MSA: с базой данных связан только один сервис
- Если транзакция затрагивает несколько микросервисов, то возникает техническая проблема организации транзакции
- Разбиваем систему на микросервисы таким образом, чтобы любая транзакция осуществлялась в пределах одного компонента (сцепленность по транзакциям)

Как обойти - занятие по распределенным транзакциям

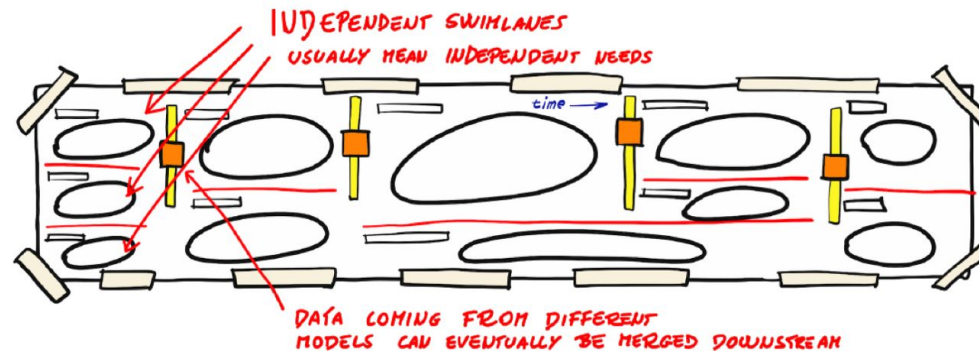
09

Event Storming

Event storming

Для относительно небольших проектов хватает функционального и ООП моделирования, чтобы перейти к модели сервисов.

Для крупных проектов – обычно не хватает. В этом случае может подойти «гибридный» и более современный подход – event storming.



<https://www.eventstorming.com/book/>

Event storming

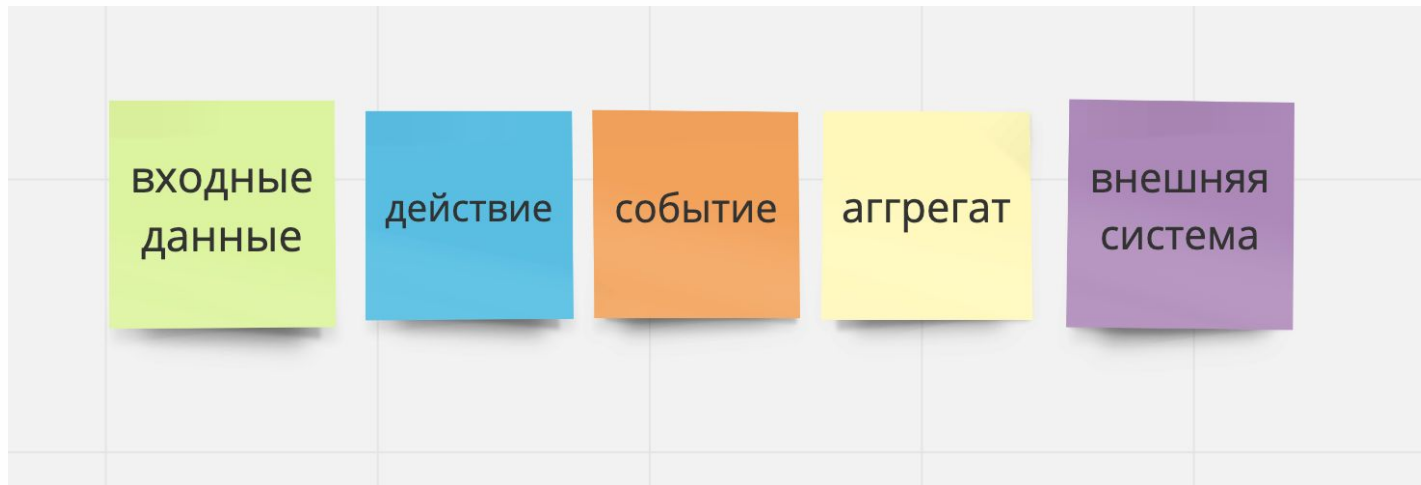
Event storming – это собрание, на котором разработчики и бизнес вместе выясняют, как же выглядит предметная область.

Обычно выделяется большое помещение, **ОЧЕНЬ** большая доска (или стена), **ОЧЕНЬ** много стикеров и маркеров.

И в течение дня или двух дней происходит моделирование предметной области.

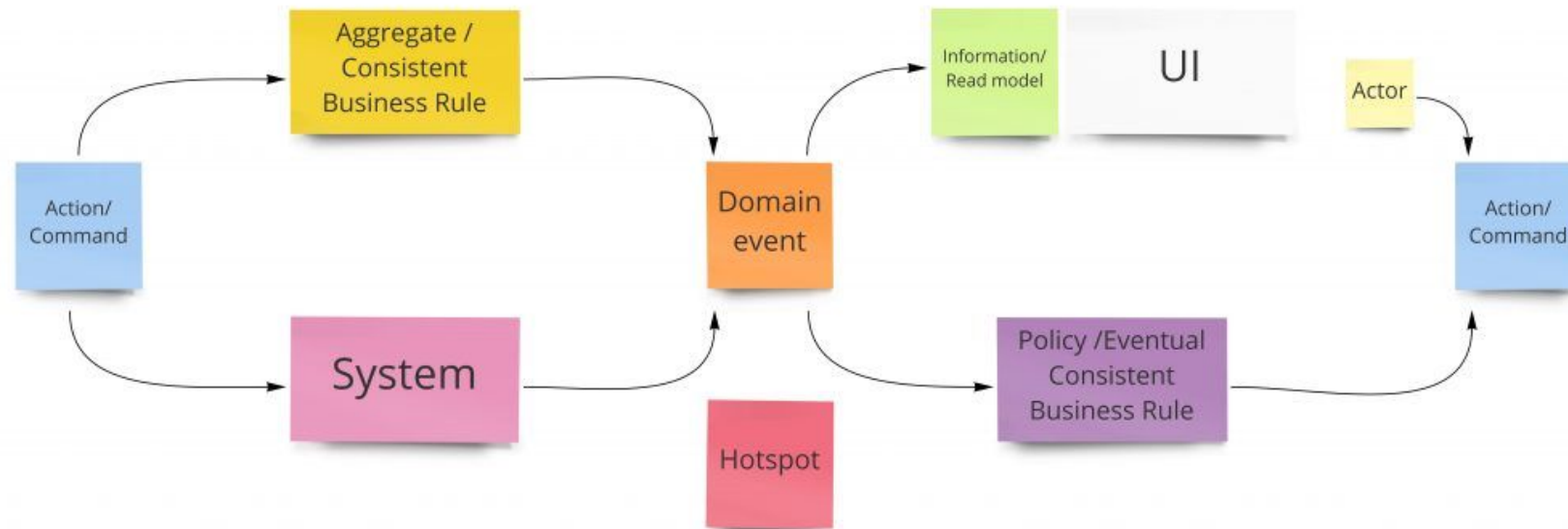
Event storming

Отражаем на доске бизнес процесс в виде набора стикеров



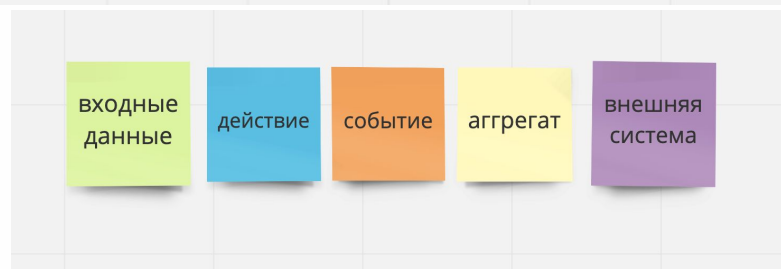
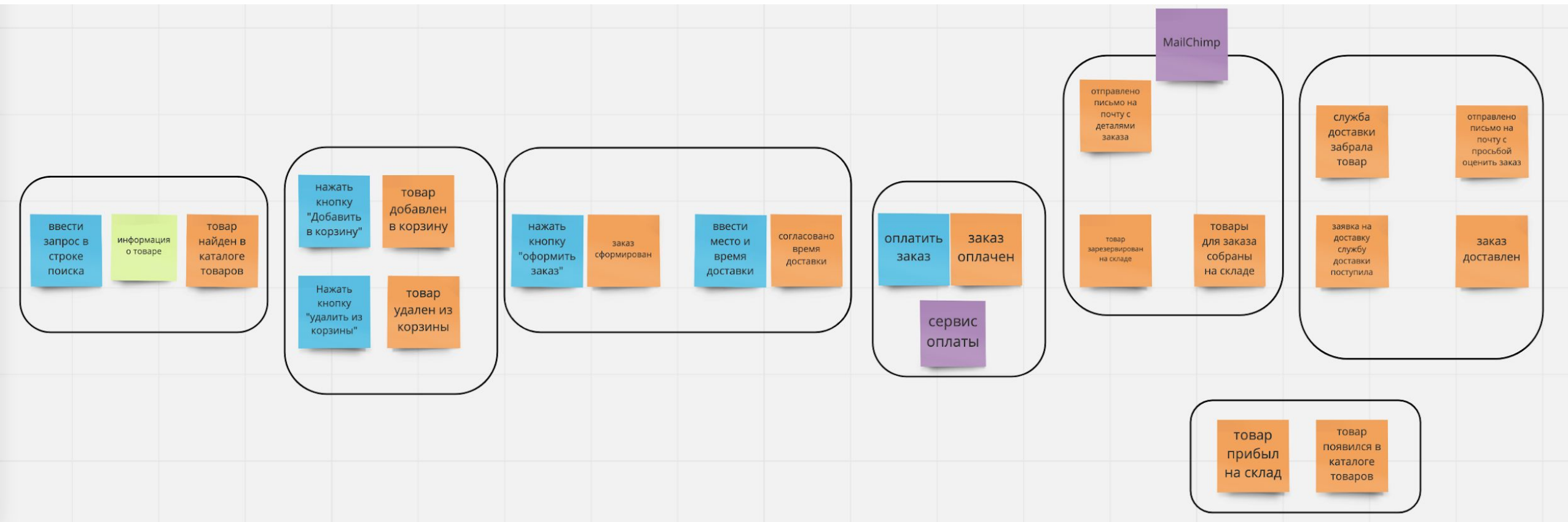
Event storming

- Представляем бизнес процесс в виде набора действий, запросов, событий, акторов, агрегатов и внешних систем.
- Пытаемся провести границы вокруг сущностей, которые потом станут границами команд и микросервисов (или группы микросервисов).
- В ограниченный контекст включается: описание, бизнес правила, запроса + команды + события, зависимости от других контекстов



Event storming

https://miro.com/app/board/o9J_kvo2IAo=



Event storming. Проблемы и особенности

- Очень многословен. Фактически для создания подробного event storming-а требуется написать спецификацию всего кода в терминах DDD. Даже для небольших проектов он может быть многословен. (<https://github.com/ddd-by-examples/library>)
- Может содержать ошибки. Кто-то какие-то события не вспомнил, кто-то что-то неправильно написал и т.д. В целом это больше инструмент коммуникации, нежели подробного и точного моделирования.
- Результаты моделирования остаются на бумаге (точнее на стене) и ими дальше не руководствуются в принятии архитектурных решений.

Event storming. Проблемы и особенности

- В рамках Event Storming лучше моделировать большие bounded context-ы (в среднем 1 на 1 команду).
- Моделирование маленьких ограниченных контекстов и деление на сервисы/микросервисы в рамках сессии event storming кажется не продуктивным.

Как декомпозировать на сервисы?

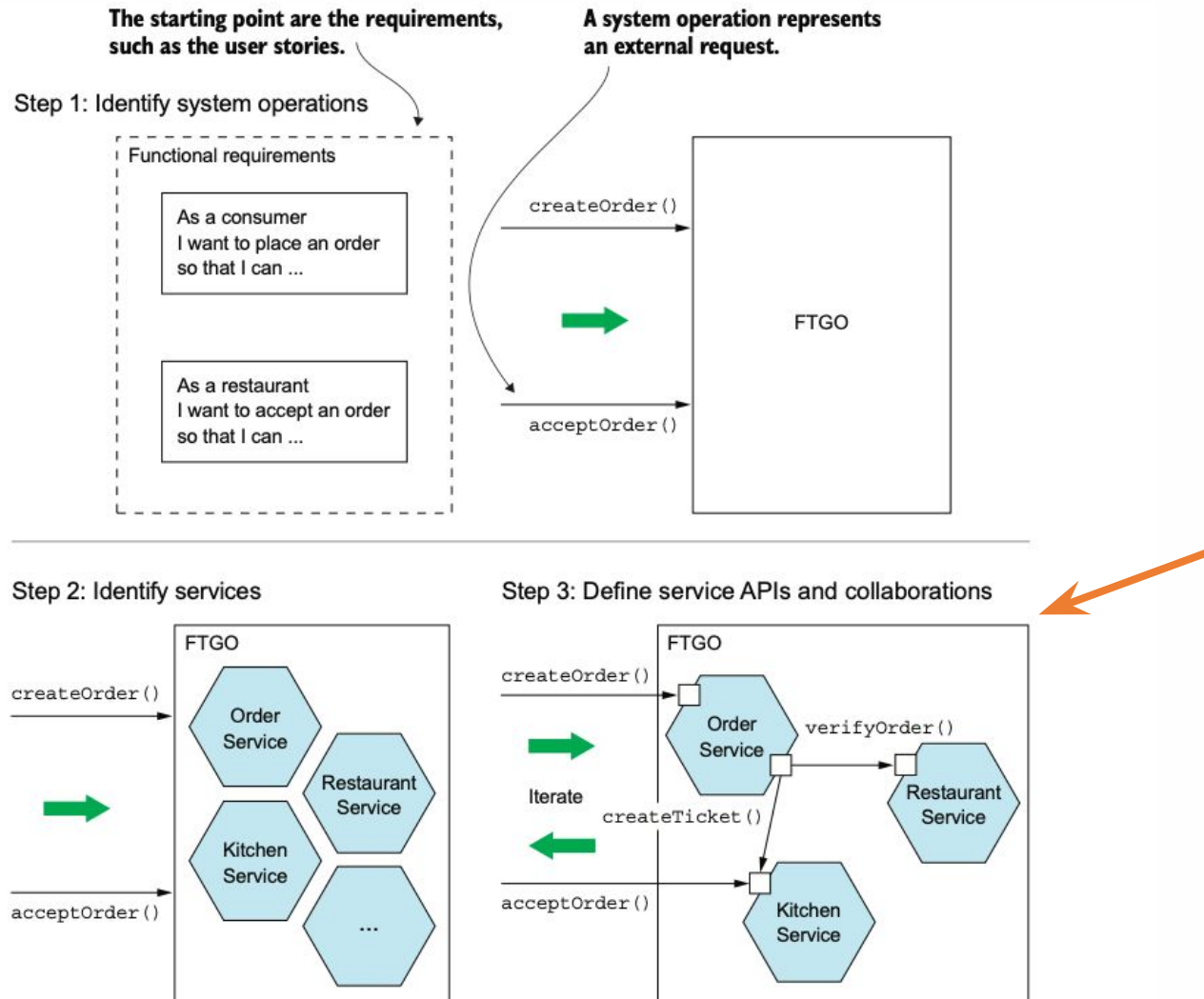


Figure 2.5 A three-step process for defining an application's microservice architecture

10

API и описание сервисов

Моделирование.

Какой бы способ мы ни выбрали, в любом случае надо понимать, что из модели мы получаем лишь первое приближение для разбиения сервисов.

И в дальнейшем разделение на сервисы мы можем делать исходя из нефункциональных требований – требований производительности, безопасности, масштабируемости и т.д и т.п.

Как моделируем сервисы?

- Прописываем системные действия для каждого сервиса уже в виде API
- Пробегаемся по основным сценариям и смотрим, не забыли ли мы что-нибудь и насколько хорошее и полное API у нас получается.
- Проходим по всем сервисам и определяем ответственность сервиса, зависимости между сервисами, пытаемся прописать конкретное API и выбираем протокол для взаимодействия
- Находим проблемы в текущей модели разбиения
- Корректируем модель и разбиение.
- Повторяем

Системные действия в модели сервисов

Клиент

GET /api/v1/client/ { delivery_address, payment_method }

Продукт

GET /api/v1/products/search?q=Гамбургер&limit=20

GET /api/v1/products/{id}/

Корзина

POST /api/v1/cart/products/ {id, quantity: 1}

DELETE /api/v1/cart/products/{id}/

PUT /api/v1/cart/products/{id}/ {quantity: 2}

Заказ

POST /api/v1/order/ { delivery_time, delivery_address, payment_method }

POST /api/v1/order/pay/ - редирект на сервис оплаты

Ресторан

POST /api/v1/restourant/prepare/ { order_id }

Какие методы еще нужны исходя из сценария?

Продукт

GET /api/v1/products/catalog/?category_id=23&subcategory_id=43&q=Бургер

POST /api/v1/products/catalog/filter

Корзина

GET /api/v1/cart/products/ - забыли получить все продукты в корзине

Заказ

GET /api/v1/orders/ - список заказов

Словари

GET /api/v1/delivery_types/ - список типа доставок

GET /api/v1/payment_methods/ - список типа платежей

Описание сервисов

Для более формального описания сервисов может подойти один из шаблонов для описания:

- `MicroservicesCanvas`
- `BoundedContextCanvas`






Microservices Canvas

Формат описания сервисов от launchany (<https://launchany.com/microservice-design-canvas/>)

Microservice Summary		
<div><div></div><div>Service Name: (name)</div></div> <div>Description: (description)</div>		
<div><div></div><div>Consumer Tasks: • (task)</div></div>		
Dependencies	Implementation	Interface
<div><div></div><div>Service Dependencies</div><div>• (service dependency)</div></div>	<div><div></div><div>Data Sources</div><div>• (data)</div></div>	<div><div></div><div>Queries</div><div>• (query)</div></div>
<div><div></div><div>Event Subscriptions</div><div>• (event subscription)</div></div>	<div><div></div><div>Logic/Rules</div><div>• (logic)</div></div>	<div><div></div><div>Commands</div><div>• (command)</div></div>
Architecture		
<div><div></div><div>Qualities</div><div>• (quality)</div></div>		<div><div></div><div>Events Published</div><div>• (event published)</div></div>
Microservice Design Canvas v2 — https://launchany.com/canvas/		

Microservices Canvas

Более простая форма описания сервиса

 Service Name: (name)	
 Service Dependencies <ul style="list-style-type: none">• (service)	 Queries and Commands <ul style="list-style-type: none">• (query/command)
 Event Subscriptions <ul style="list-style-type: none">• (event)	 Events Published <ul style="list-style-type: none">• (event)
Microservice Design Card v2 — https://launchany.com/canvas/	

Bounded Context Canvas

Bounded Context Canvas – похож на канвас про микросервисы, но более DDD ориентированный и построение bounded context-ов, его можно использовать для описания сервисов

<div>Название</div> <div>Реестр налогов на имущество предприятий</div>	<div>Возможности и обязанности</div> <div>(Сервисы/услуги, предоставляемые потребителям)</div>											
<div>Стратегическая классификация</div> <div>(основной, поддерживающий, обобщенный и др.)</div> <div>Основной: Необходимо, чтобы правительство могло оказывать важные услуги законно и справедливо</div>	<div>Информационные</div> <div>(запросы, события, отчеты...)</div>		<div>Действия</div> <div>(вызываемые команды, запланированные задачи...)</div>									
<div>Описание</div> <div>(краткое описание цели и обязанностей - не реализация)</div> <div>Каталог с подробным описанием сумм налога на имущество, уплаченных каждым предприятием и критериев, используемых для определения этих сумм за последние 20 лет</div>												
<div>Бизнес-правила</div> <div>(ключевые бизнес-правила и политики)</div>	<div>Зависимости</div> <div>(Взаимодействие с другими bounded contexts и службами)</div> <table><tr><td>Название</td><td>Причина зависимости</td><td>Внутренняя или внешняя система?</td><td>Отношение (Входящая, исходящая, двунаправленная, UI)</td></tr><tr><td></td><td></td><td></td><td></td></tr></table>				Название	Причина зависимости	Внутренняя или внешняя система?	Отношение (Входящая, исходящая, двунаправленная, UI)				
Название					Причина зависимости	Внутренняя или внешняя система?	Отношение (Входящая, исходящая, двунаправленная, UI)					
<div>Ubiquitous Language</div> <div>(ключевая терминология домена)</div>												

<https://habr.com/ru/company/oleg-bunin/blog/500506/>

Сервис «Продукт»

Название:

Продукт

Запросы:

- Поиск продукта GET /api/v1/search/?q=...
- Информация о продукте GET /api/v1/products/{id}
- Информация о списке продуктов
- ...

Команды:

- Изменение товарной номенклатуры
POST on /api/v1/products/{id}

События:

- -

Зависимости:

- Слушает события ProductReserveChanged от Склада.

Вопросы:

- Атрибуты для поиска продуктов и технология – Elastic Search/Solr для поиска использовать?

Сервис «Корзина»

Название:

Корзина

Запросы:

- Список продуктов в корзине
GET /api/v1/cart/products/ { [product_id , quantity_id] }

Команды:

- Изменение состава продуктов в корзине
POST /api/v1/cart/products/ {id, quantity: 1}
DELETE /api/v1/cart/products/{id}/
PUT /api/v1/cart/products/{id}/ {quantity: 2}

События:

- -

Зависимости:

Вопросы:

- В какой момент очищается корзина? Ее в явном виде очищает создание заказа? Типа когда создан заказ, тогда очищать? Можно ли ее сделать на клиенте?
- Зависимость от остатка на складе

Сервис «Заказ»

Название:

Заказ. Оформление заказа – оплата, доставка и т.д.

Запросы:

- Получить текущий заказ
GET /api/v1/orders/current/ { status, deliveryTime, deliveryAddress, paymentMethod }
- Получить список прошлых заказов
GET /api/v1/orders/ [{ ... }]

Команды:

- Изменить параметры (время доставки, адрес, метод оплаты)
POST/PUT /api/v1/orders/current/ { deliveryTime, deliveryAddress, paymentMethod }
- Отменить заказ
DELETE /api/v1/orders/current
- Оплатить заказ
POST /api/v1/orders/current/pay

События:

- Заказ оплачен (оформлен).
OrderCreated (id, deliveryTime, ..., productItems)
- Заказ отменен
OrderCancelled(...)

Сервис «Заказ»

Название:

Заказ. Оформление заказа – оплата, доставка и т.д.

Зависимости:

- Нужно получить список продуктов с количеством из корзины
- Получить цены продуктов из сервиса Продукты
- Сделать редирект на платежный шлюз и получить от него ответ
- Зарезервировать продукты на складе с помощью сервиса Склад
- Зарезервировать курьерскую доставку

Вопросы:

- В какой момент очищается корзина?
- Если заказ отменили, то нужно еще событие об этом для Склада?
- Слишком сложный код?

11

Возможные изменения

Возможные изменения

- Сервис Продукт имеет смысл разделить на 2.
- Формирование корзины и оформление заказа кажется, что про одно и то же. И сервисы получаются сильно связанные. Имеет смысл эти сервисы объединить
- Сервис Заказ много с кем связан и интегрирует в себя много логики, имеет смысл отдельно выделить сервис оплаты

Нужен ли отдельный сервис оплаты?

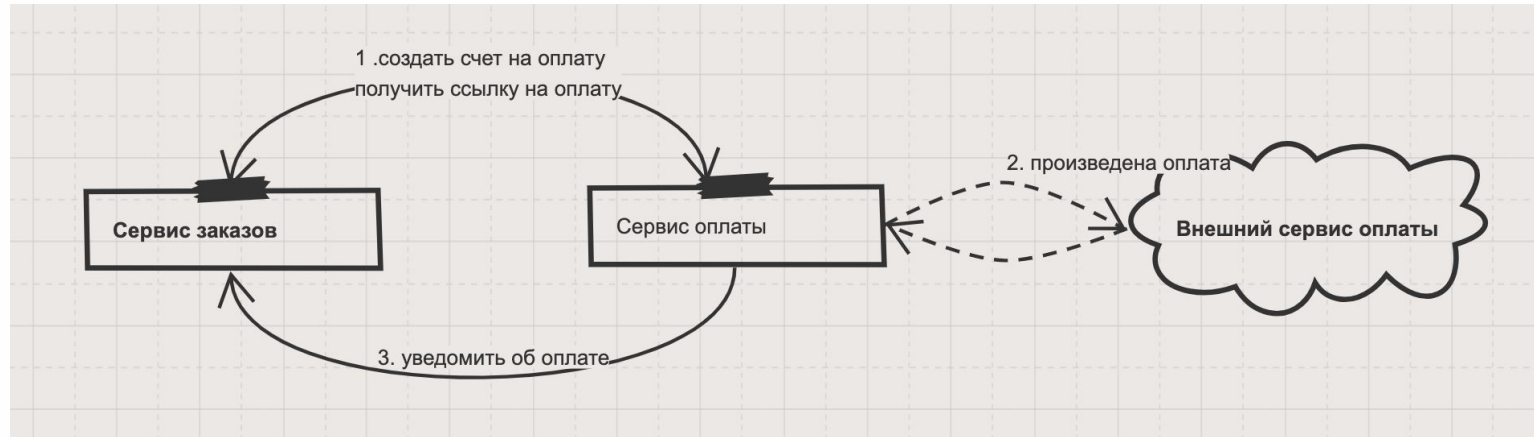


Что будет, если из сервиса Заказов сразу ходить во внешний сервис?

Сервис Заказов будет «знать» про внутренности внешнего сервиса оплаты.

Если понадобится оплата вне сервиса заказов (например, если надо будет пополнять счет в личном кабинете), то мы не сможем переиспользовать код.

Как связать сервис оплаты и сервис заказов?



Что будет, если мы сделаем прослойку в виде Сервиса оплаты?
Создать счет и получить ссылку на оплату можно по API

POST /api/v1/invoices/ { payment_type_id amount {order_id, redirect_url}}

Без ссылки на оплату флоу работы клиента заблокирован.

Как можно уведомить сервис Заказов, что оплата прошла?

POST /srv/api/v1/orders/{order_id}/invoice_payed/

В этом случае «Сервис оплаты» явно знает про «Сервис заказов» и получаем похожую проблему.

Как связать сервис оплаты и сервис заказов?



Что будет, если мы сделаем очередь событий «Произошла оплата»?

Сервис оплаты теперь не будет знать явно про Сервис заказов.
Но в этом случае увеличится latency, насколько будет комфортно пользователю, что сообщение об оплате может появиться не сразу?

Кажется, что ок.

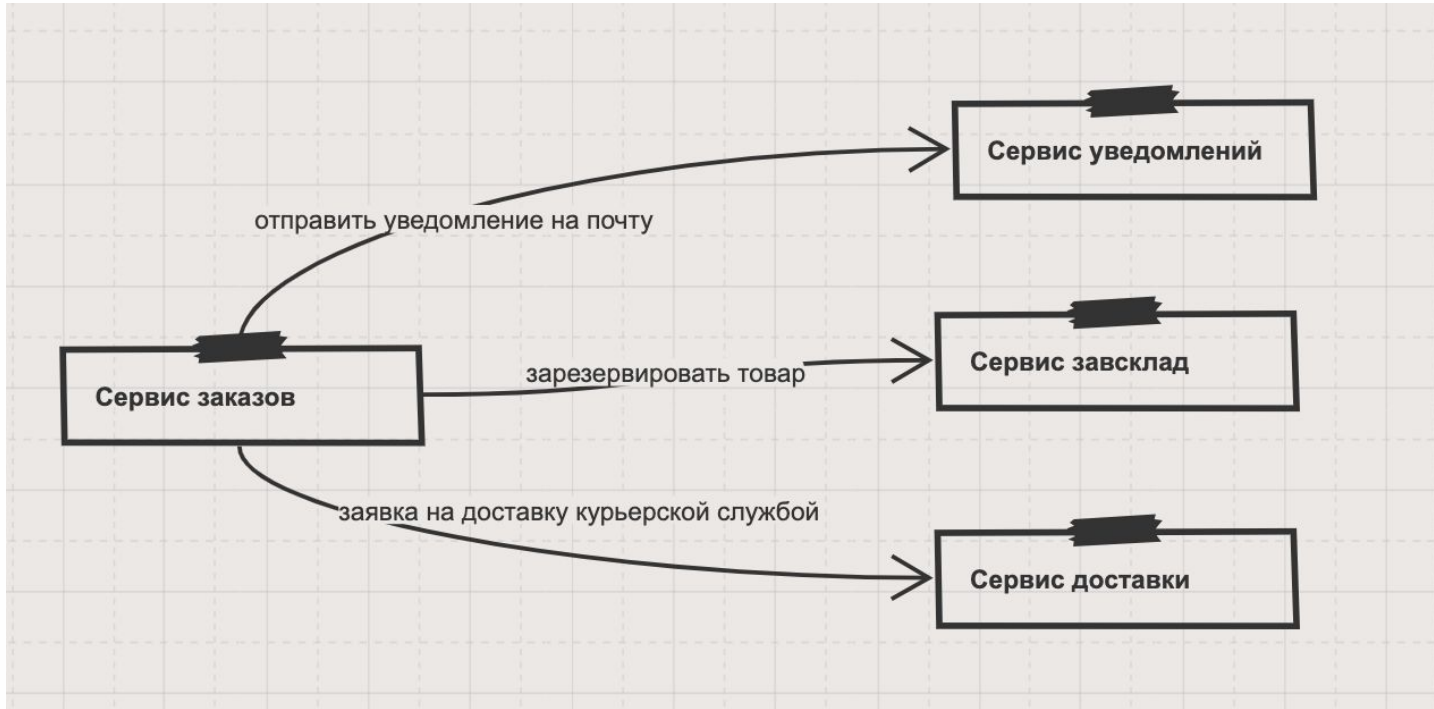
Как связать сервис заказов и сервис доставки?



Когда оплата завершилась, нам надо еще

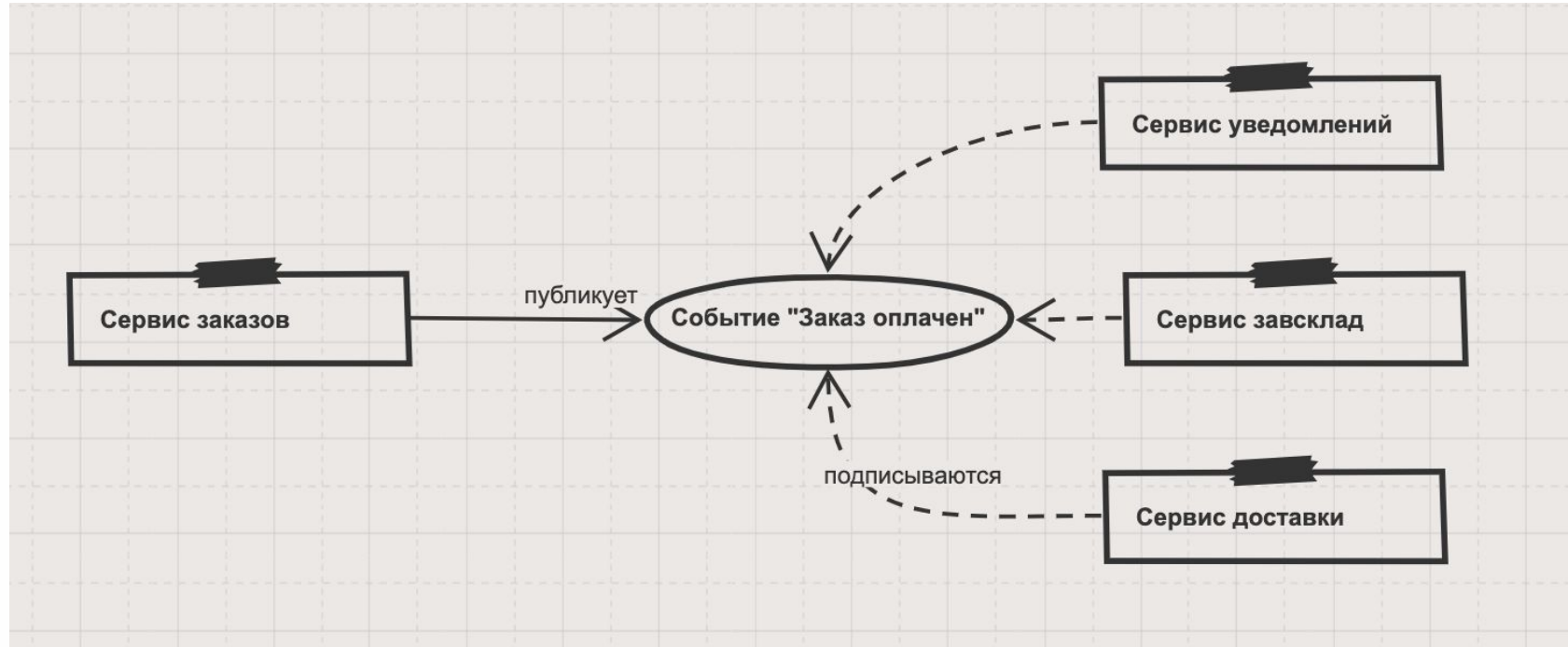
- 1) послать уведомление на почту
- 2) зарезервировать товар на складе /
- 3) сделать заявку на доставку в курьерскую службу

Как связать сервис заказов и сервис доставки?



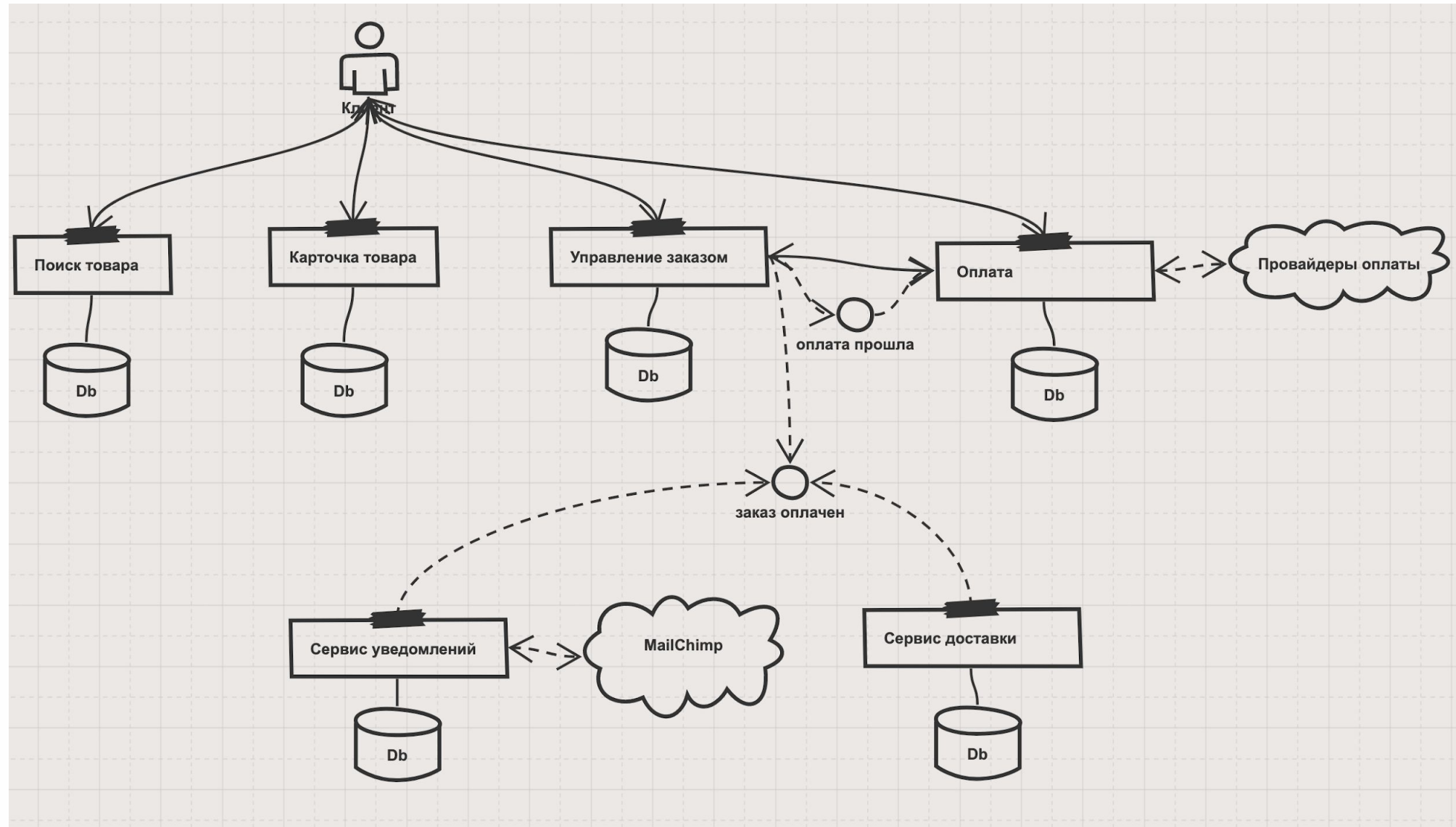
Сервис Заказов явным образом (синхронно) ходит во все остальные сервисы, когда получает сообщение из очереди, что заказ оплачен.

Как связать сервис заказов и сервис доставки?



Сервис Заказов публикует сообщение, что заказ оплачен, а остальные сервисы его слушают.

Как можно изменить разбиение сервисов?



11

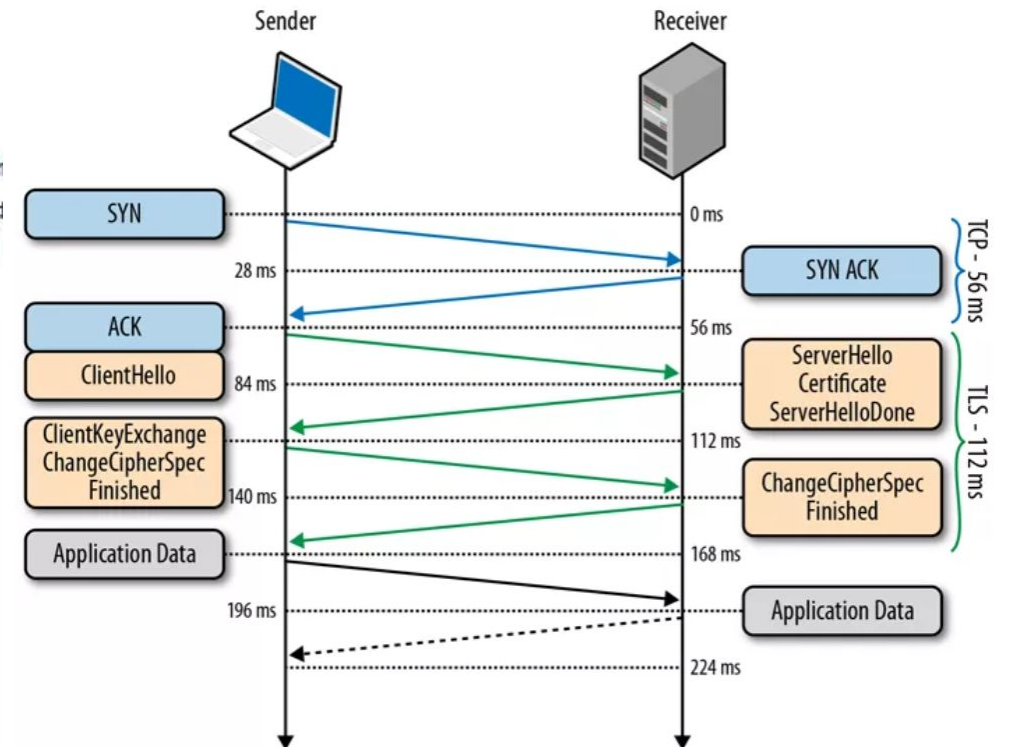
Что еще стоит учесть

Latency

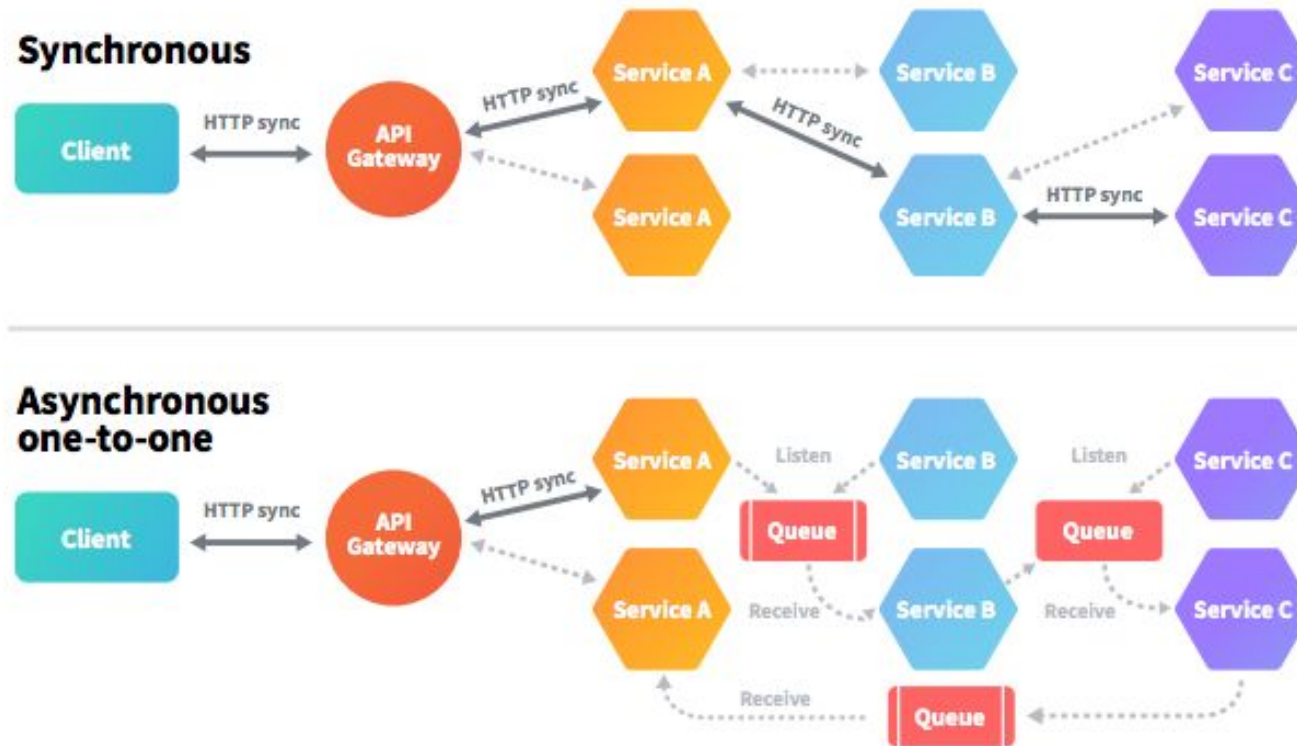
Вызов по сети на несколько порядков медленнее, чем вызов локальной функции

Latency Comparison Numbers (~2012)

L1 cache reference	0.5	ns			
Branch mispredict	5	ns			
L2 cache reference	7	ns	14x	L1 cache	
Mutex lock/unlock	25	ns			
Main memory reference	100	ns	20x	L2 cache, 200x	L1 cache
Compress 1K bytes with Zippy	3,000	ns	3	us	
Send 1K bytes over 1 Gbps network	10,000	ns	10	us	
Read 4K randomly from SSD*	150,000	ns	150	us	~1GB/sec SSD
Read 1 MB sequentially from memory	250,000	ns	250	us	
Round trip within same datacenter	500,000	ns	500	us	
Read 1 MB sequentially from SSD*	1,000,000	ns	1,000	us	1 ms ~1GB/sec SSD, 4X mem
Disk seek	10,000,000	ns	10,000	us	10 ms 20x datacenter round
Read 1 MB sequentially from disk	20,000,000	ns	20,000	us	20 ms 80x memory, 20X SSD
Send packet CA->Netherlands->CA	150,000,000	ns	150,000	us	150 ms



Синхронное взаимодействие ухудшает доступность



Согласованность и согласованное представление

Оформление заказа

1. Оплата
2. Формирование заказа в сервисе Kitchen
3. Планирование доставки в сервисе Delivery

Повествование - занятие по распределенным транзакциям

Согласованное представление

Сколько еще надо продуктов на оплаченные, но недоставленные заказы?

Рекомендации

1. Начинайте с основного пользовательского сценария
2. Исходя из пользовательского сценария с помощью одного из методов постройте модель предметной области
3. Из модели предметной области сделайте нулевую итерацию разбиения сервисов
4. Пройдитесь по всем сценариям и для каждого сервиса попробуйте конкретно описать его взаимодействие с клиентом, и с другими сервисами. Например, используя MicroserviceCanvas
5. Оцените сервисную модель: какие у нее есть слабые стороны, проблемы, какие возникли вопросы в рамках составления карточки сервисов? Как с точки зрения функциональных, так и не функциональных требований.
6. Придумайте решение проблем текущего разбиения, и переразбейте сервисы, по-другому организуйте взаимодействие (и т.д)
7. Повторяйте п 4-6 до полного просветления

Опрос

Домашнее задание

Разделите ваше приложение на несколько микросервисов с учетом будущих изменений.

Попробуйте сделать несколько вариантов разбиений и попробуйте их оценить. Выберите вариант, который вы будете реализовывать.

На выходе вы должны предоставить

- Пользовательские сценарии
- Общую схему взаимодействия сервисов.
- Для каждого сервиса опишите назначение сервиса и его зону ответственности.
- Опишите контракты взаимодействия сервисов друг с другом.

**Спасибо
за внимание!**

