СТАЧКА

Визуальное **проектирование** масштабируемых приложений

Максим Цепков

IT-архитектор и бизнес-аналитик,
Навигатор и эксперт по миру Agile,
методов самоуправления и моделей soft skill

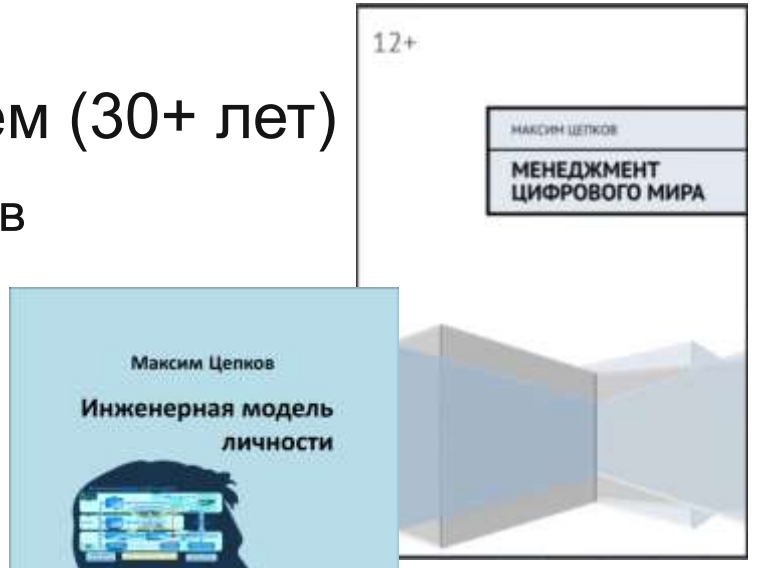
mtsepkov mtsepkov.org

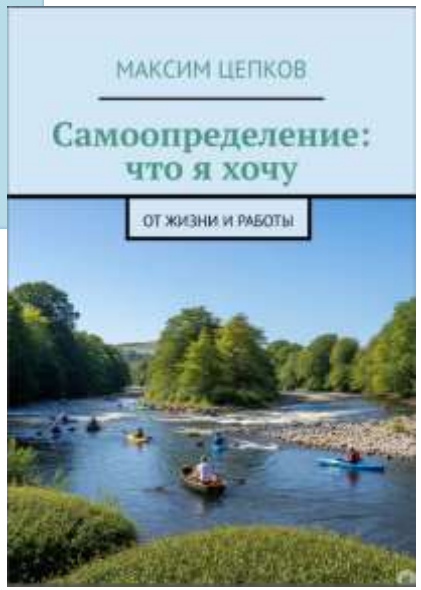
Стачка | Ульяновск
10-11 апреля 2026

Немного обо мне

Создание и внедрение больших корпоративных систем (30+ лет)

- Знание практик операционного управления и ведения проектов в коммерческих и государственных организациях и банках.
- Опыт управления проектами в ИТ — от инженерного подхода и PMBOK к современным Agile-методам (с 2007 года).
- Опыт перестройки организаций при внедрении систем.





Навигация в менеджменте цифрового мира

- Agile и самоуправление: бирюзовые организации, холакратия и социократия (**книга**, статьи и выступления).
- Модель спиральной динамики (с 2013 года) и другие модели soft skills, **модели личности** (**книга**) и **самоопределения** (**книга**).

Китай как лидер новой волны технологических революций

На сайте mtsepkov.org мои выступления и много других материалов.

О чём это выступление?

- Визуальные схемы – эффективное представление архитектуры софта
- Традиционные схемы созданы в эпоху монолитов, когда транзакции обеспечивали устойчивость, а масштабирование давало железо
- Сервисной архитектуре нужны новые модели и схемы для проектирования масштабирования и устойчивости приложений – и я покажу свой вариант

Мои выступления по этой и смежным темам

- Визуальное проектирование масштабируемых приложений (TechLead-2021)
- Архитектура софта и бизнеса в сложном ИТ-ландшафте (AnalystDays-2025)
- Постановка от модели бизнеса до детального дизайна требований: как делать и кому (13.03.2025)
- DDD: модели вместо требований 9 лет спустя (ЛАФ-2023)
- Обеспечиваем устойчивость интеграции (SQAdays-2025)

Визуальное проектирование

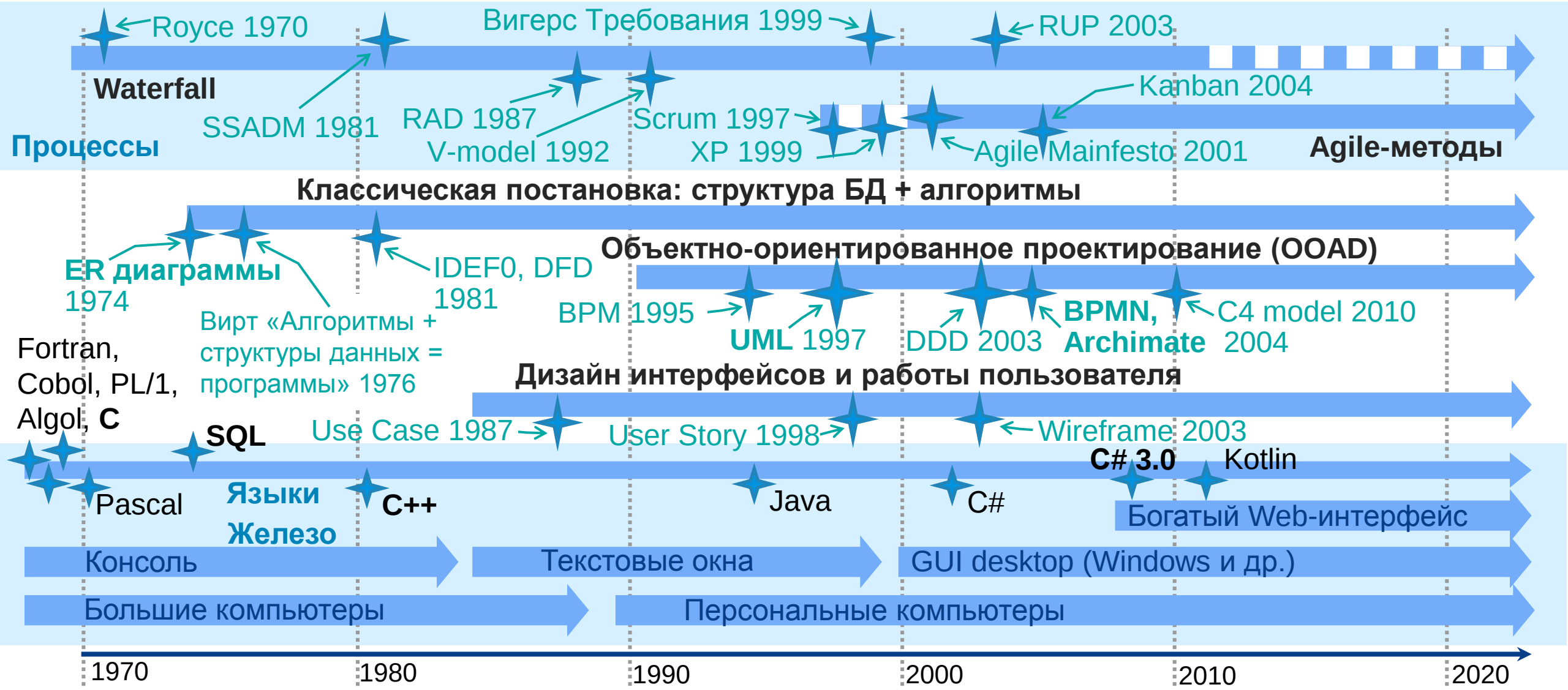
Визуальное проектирование приложений

- Блок-схемы для описания алгоритмов и ER-диаграммы базы данных появились в 1960-70-е годы, но это был лишь способ нарисовать диаграмму
- С развитием персоналок появились системы визуального конструирования приложений: PowerBuilder (1991), Delphi (1995), Erwin (1993) для модели БД
- В 1997 создан **UML**, замысел – **графический язык создания приложений**
  - Activity diagram для бизнес-процессов
  - Class diagram для структур данных
  - State diagram для workflow документов
  - Sequence диаграмм для взаимодействия процессов
  - и так далее
- Rational Rose, Oracle Designer 2000 – попытка чисто визуальной разработки
- Enterprise Architect (2000), Visual Paradigm (2002), Activiti (2010) – живет, но создание кода решает ограниченные задачи: структуры данных, BPMN

Описание через диаграммы – востребовано, а вот приложения создают с помощью кода

Не взлетело

Проектирование и его окружение



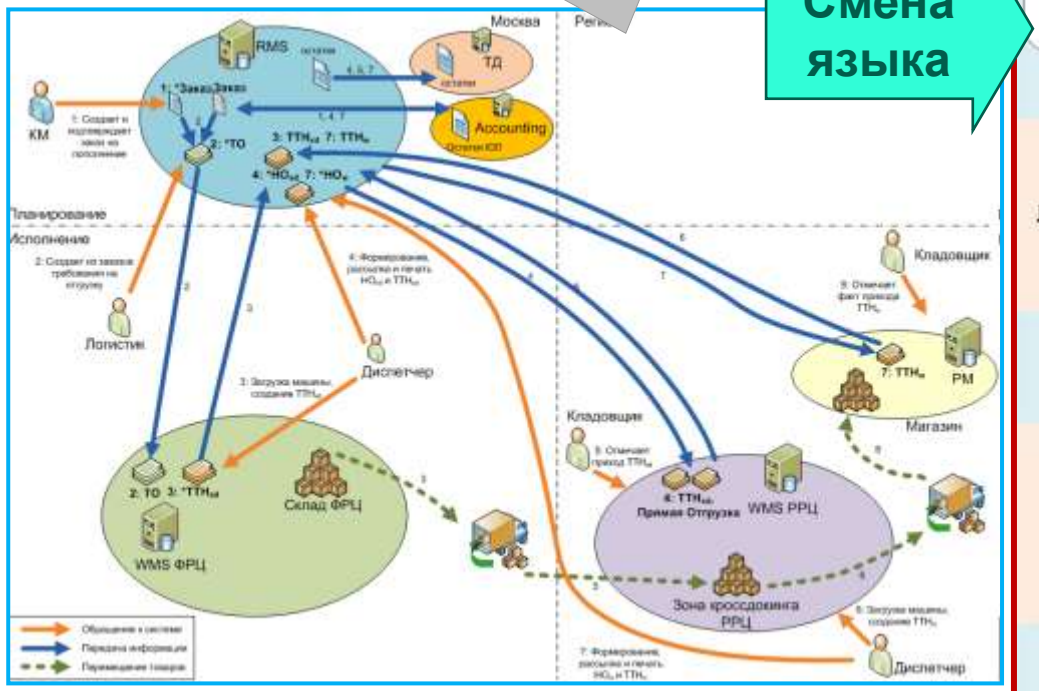
Формализация снабжения магазинов

Объекты — Class Diagram

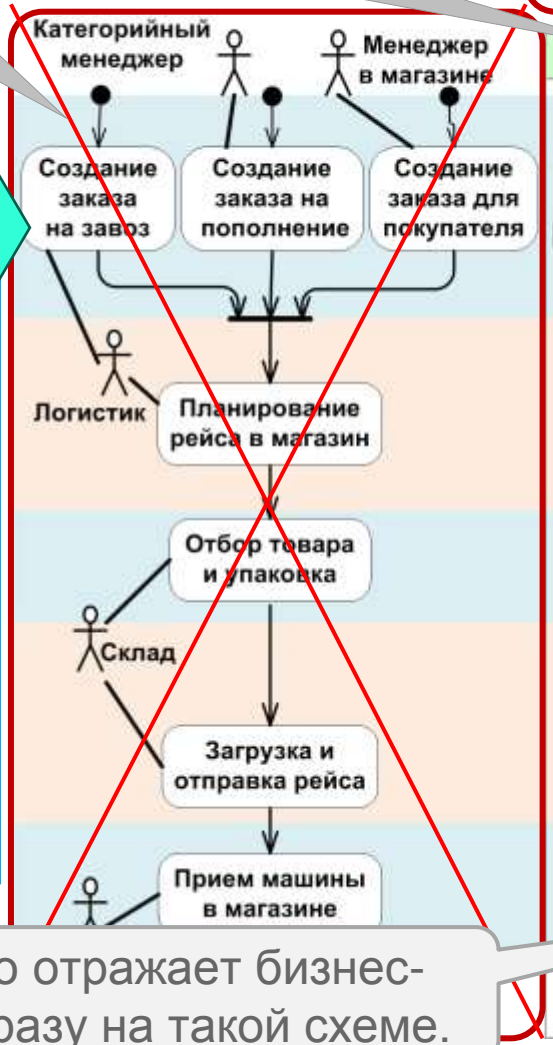
Бизнес-процесс — Activity Diagram

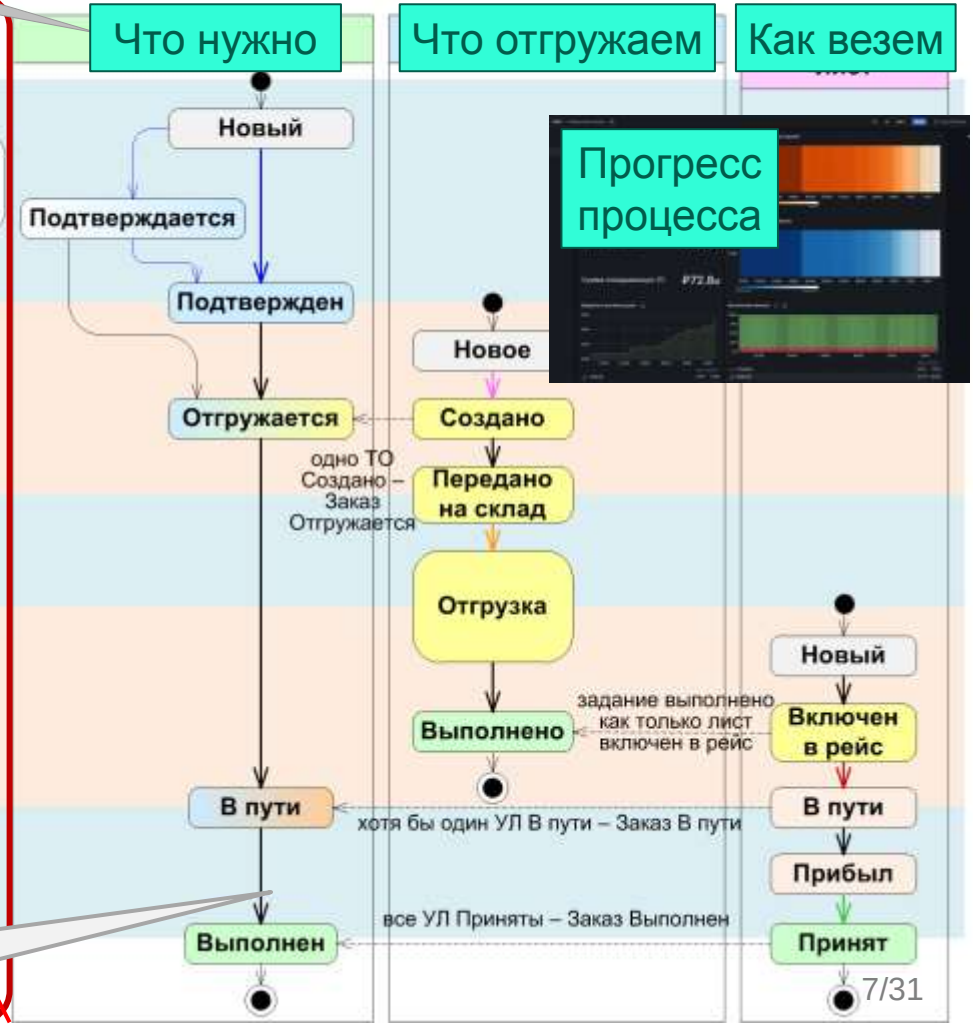
Состояния документов — State Diagram

Неформальная схема деятельности и её отражение в существующих системах



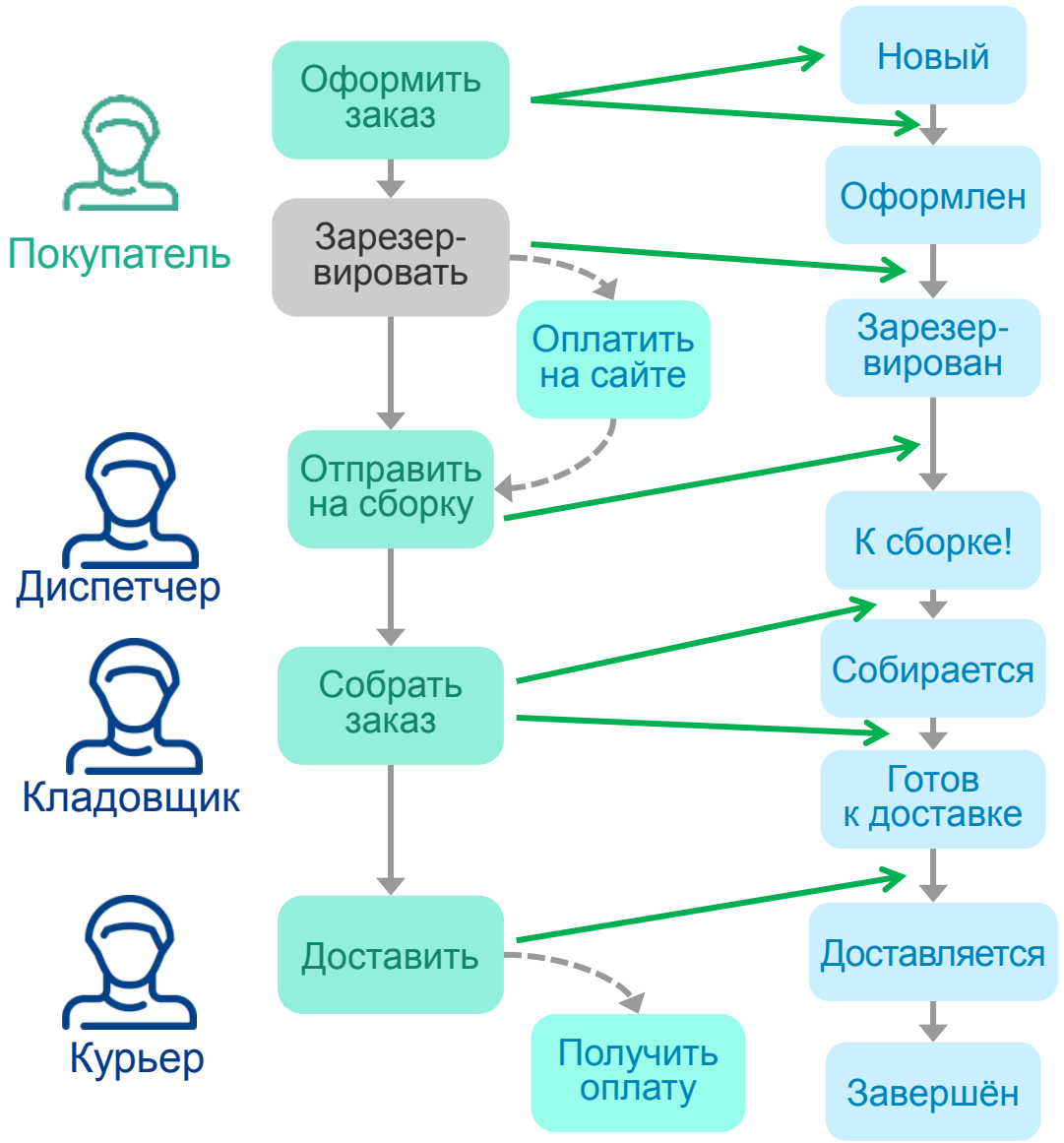
Смена языка

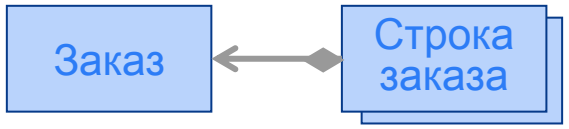




Если workflow документов прозрачно отражает бизнес-процессы, то можно обсуждать их сразу на такой схеме.

Классика — процесс через workflow

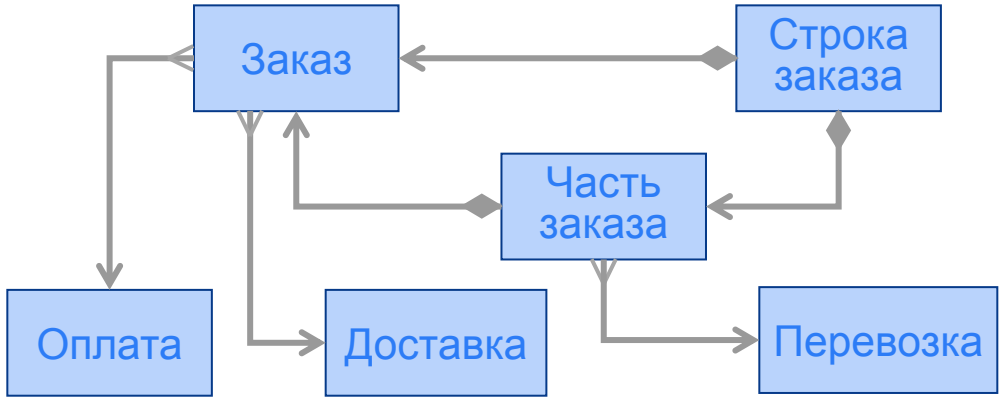




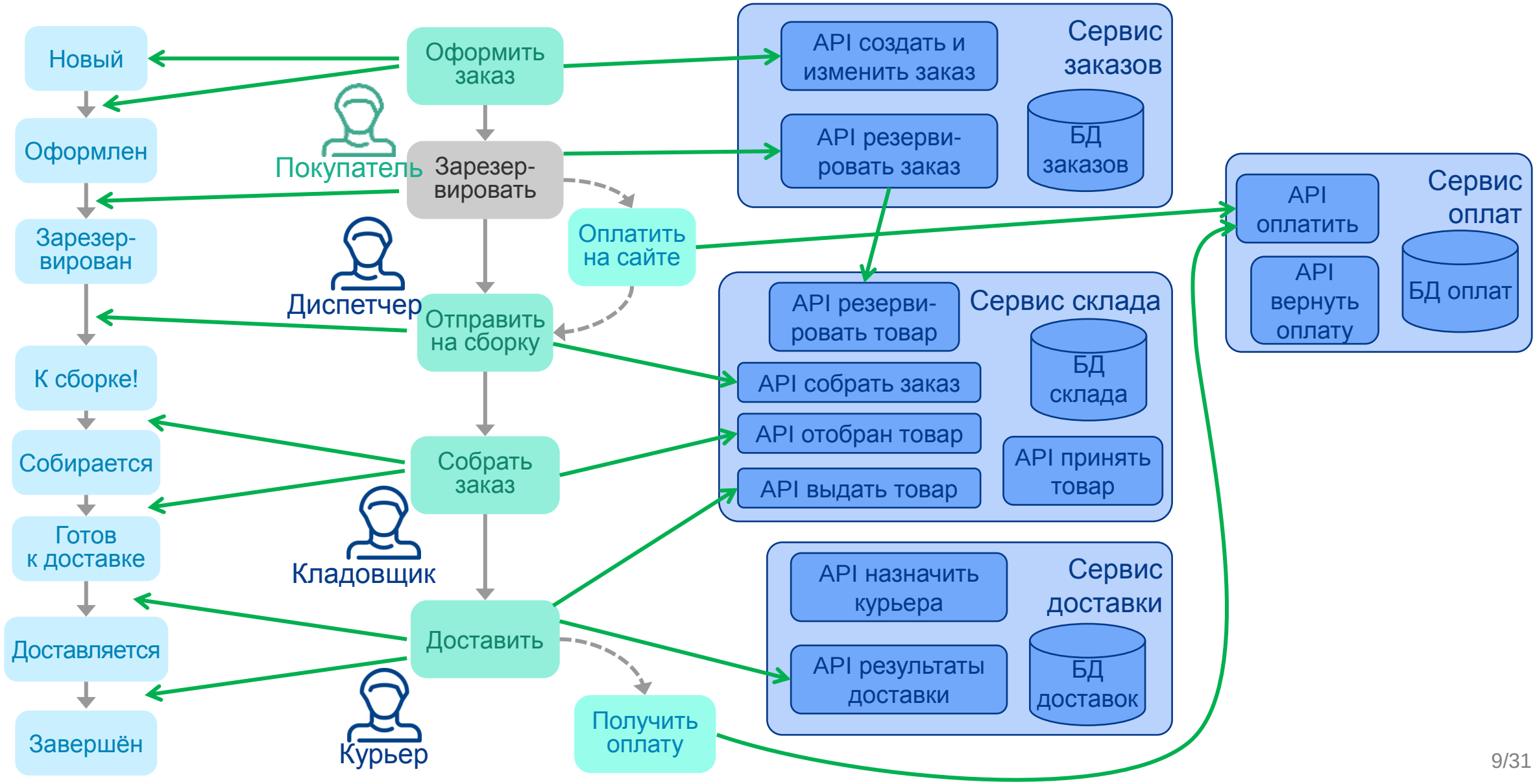
- Состояние заказа — единственный атрибут
- Отражение оплаты — две суммы в заказе

Усложнения:

- Собирают на складе несколько кладовщиков
- Собирают на нескольких складах, объединяют для выдачи покупателю или курьеру
- Заказ делят — оплата относится к нескольким



Монолит vs сервисы: workflow vs API



Сервисы: API вместо workflow

- Заказ передаётся между сервисами Заказов, Оплат, Склада и Доставки (хореография), или сервис Заказов оркестрирует исполнение, или гибрид: основная цепочка Заказ — Склад— Доставка, а Оплаты — вызываются?
- Транспортный объект: полная структура заказа или то, что нужно сервису? Например, цены и полный адрес со схемой проезда не нужны складу
- Транспорт должен быть устойчив к расширению возможностей систем!
- Какое API предоставляют справочники?
  - Денормализация: названия товаров хранятся в заказе или каждый раз запрашиваются?
  - Какие у нас сервисы ведения цен, скидок и рекламных акций? Кто вычисляет цену позиции заказа с учётом скидок и акций, учитывая что возможные комплекты и сложные условия (бесплатная доставка, если заказ больше 1000 рублей и для крупных товаров)?
- Планирование доставки и её исполнение — один сервис, или два с общей базой, или два с разными базами, плюс сервис база адресов и геоданных?

C4 Model – декомпозиция для сервисов

- Основные диаграммы – четыре уровня декомпозиции:
  - System context diagram – работа системы в окружении
  - Container diagram – декомпозиция системы на контейнеры, размещаемые на одной ноде
  - Component diagram – декомпозиция контейнера на сервисы
  - Code diagram – **диаграмма классов** для отдельного сервиса
- Дополнительные диаграммы
  - System landscape diagram – взаимодействие многих систем в ландшафте
  - Dynamic diagram – динамика обработки: sequence или collaboration
  - Deployment diagram – физическое размещение контейнеров

Проблемы

- Диаграммы показывают способ декомпозиции, соответствующий сервисной архитектуре, **но не отражают аспектов масштабирования**
- Диаграммы для разработчика – **смешаны бизнес и технические аспекты**

Что изменилось в сервисной архитектуре

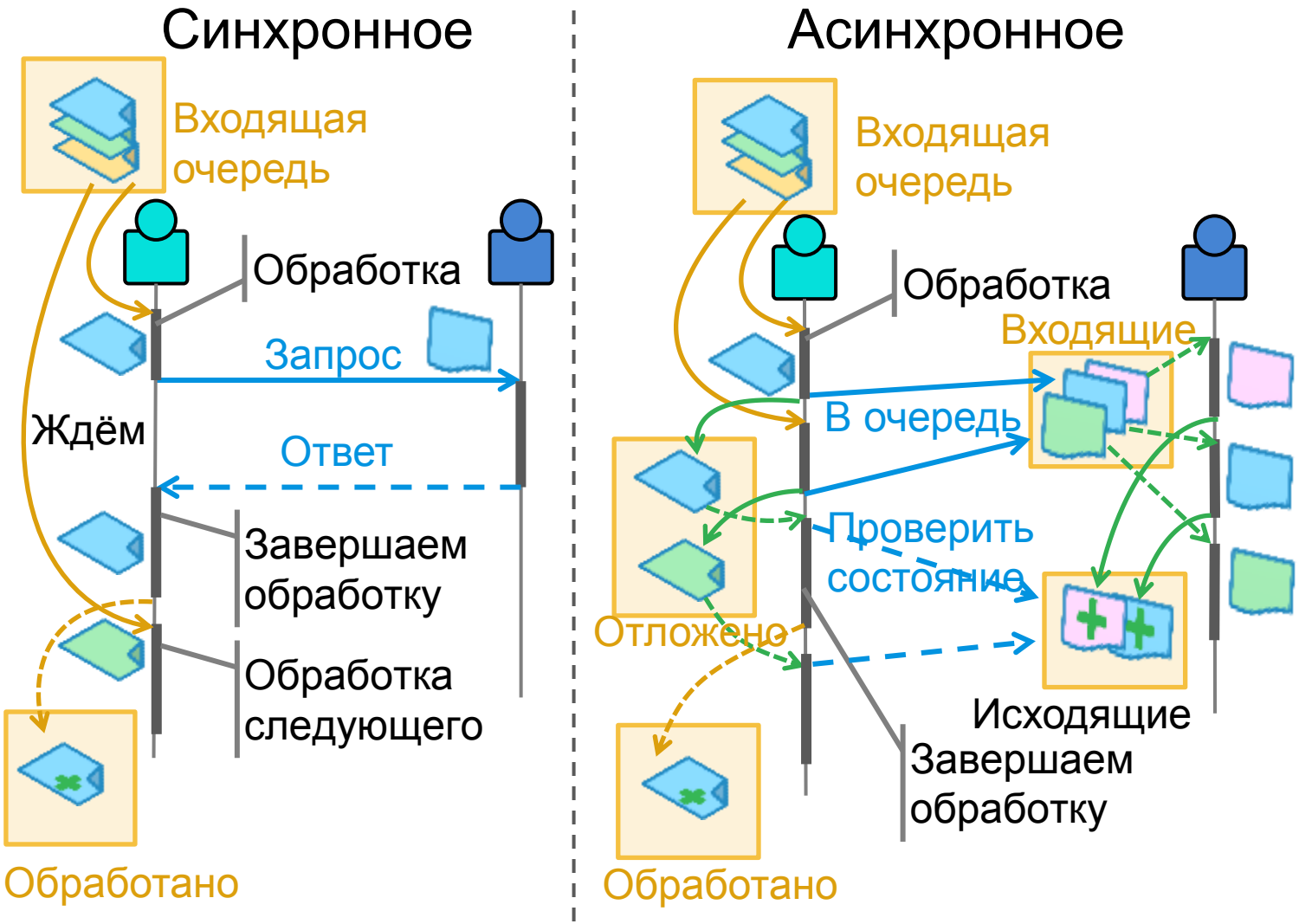
Микросервисы: масштабируемость и быстрые доработки приложений

- Каждый бизнес-запрос обрабатывает много сервисов, нет общей транзакции
- Много экземпляров одного сервиса для масштабирования, каждый может упасть по ошибкам или блокировкам, а система должна работать устойчиво
- Асинхронные сообщения, очереди выравнивают производительность

Отказ от единой реляцинной СУБД – она не справляется

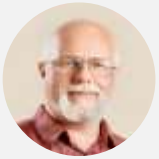
- NoSQL-базы данных, использование многих БД одним приложением, In-memory хранение в БД и очередях, отложенный сброс в хранилища
- Кластерное развертывание с независимым хранением на узлах
- Транзакционность и консистентность обеспечивает приложение, а не БД
- При восстановлении узла кластера данные неконсистентны

Варианты межсервисного взаимодействия



Устойчивость взаимодействия

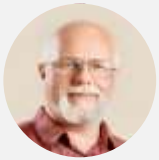
- **Протоколов передачи, обеспечивающих exactly once, не существует!**
- Идемпотентные операции обеспечивают устойчивость обработки:
  - Не Insert + Update (CRUD и REST), а **Upsert**, уникальный ключ создаёт клиент
  - Распределённые транзакции убивают скорость, проектируем без них
- Учитываем: при падении инстанса портятся связанные с ним очереди, и даже при сохранении информации может нарушаться порядок сообщений
- Обеспечиваем устойчивость на большом потоке сообщений
- Инциденты неизбежны — **нужна хорошая админка** для быстрого разбора



Устойчивость приложения обеспечивает проектирование.
Кейс: при внедрении хранилища данных за счет сверки информации от разных систем **нашли тысячи коробок**, заблокированных на складе в результате ошибок интеграции

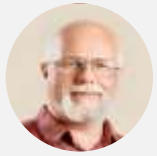
Нужна новая модель проектирования

- Показывать способы масштабирования для отдельных сервисов
- Отражать способы взаимодействия между множествами сервисов
- Моделировать поведение при падении экземпляров сервисов, проектировать устойчивость системы в целом
- Рассматривать восстановление при сбоях узлов кластера и дата-центров — техника и базовый софт не обеспечивают консистентного восстановления



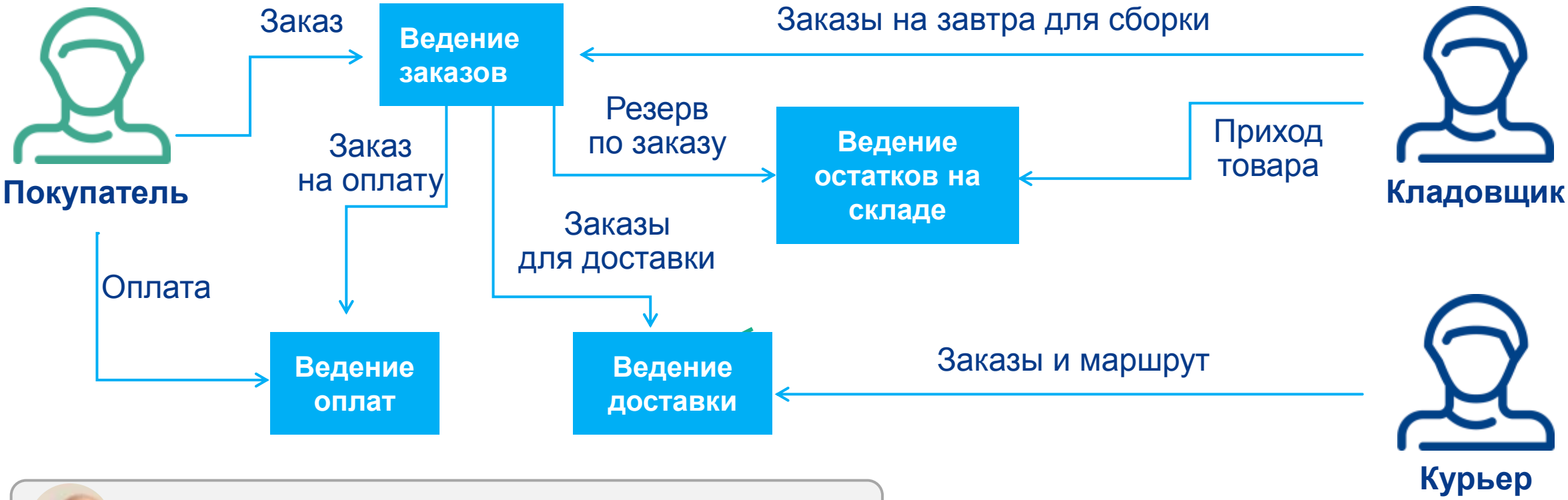
Раньше мы проектировали системы, в которых падение одного элемента было форс-мажором.
Теперь надо проектировать системы, в которых падение элемента-сервиса – норма.
Система должна восстанавливать упавшие элементы и устойчиво работать.

Решаем проблему: метафора гномиков — человечков, которые все делают



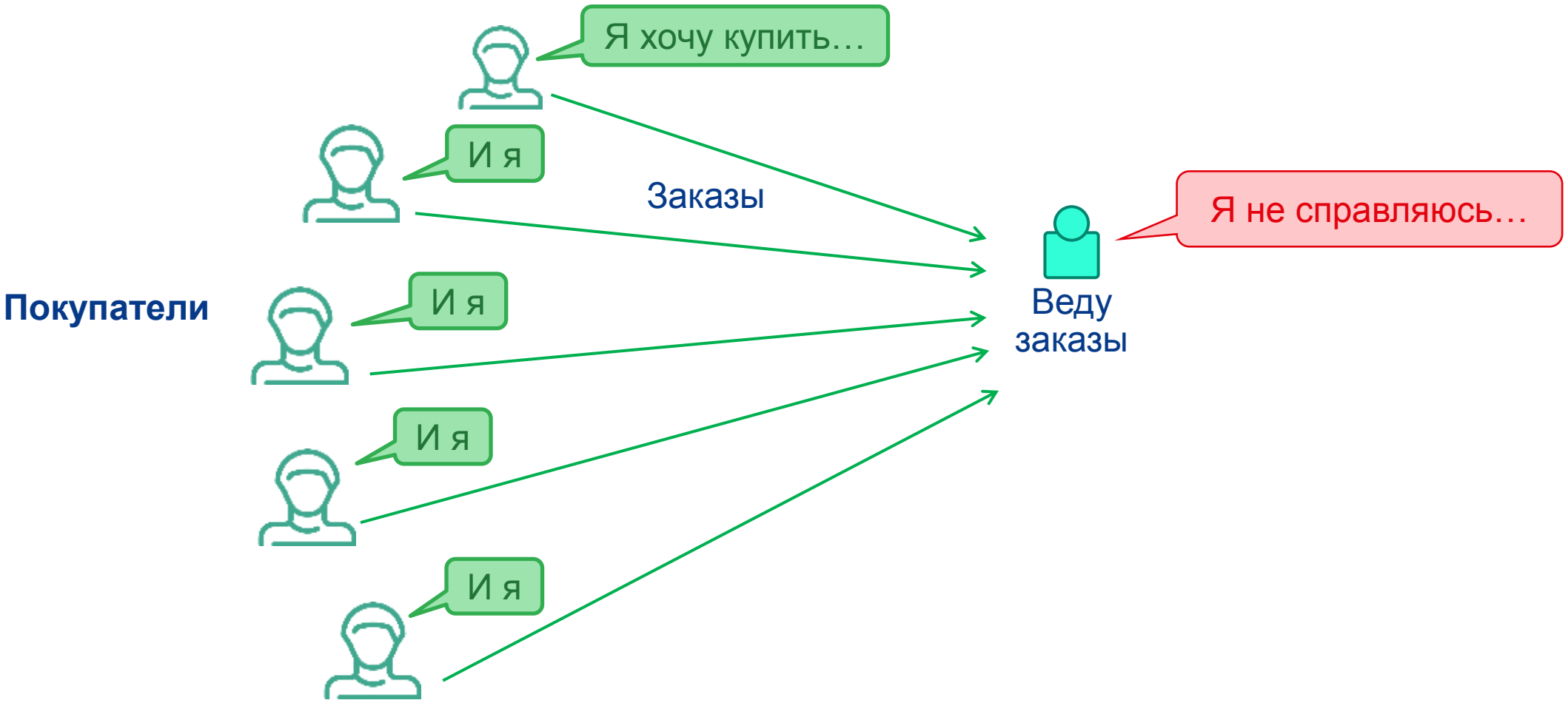
Гномик представляет актора, а олицетворение делает устройство системы понятным не только разработчикам, но и аналитикам, тестировщикам, бизнес-заказчикам

Схема сервисов для интернет-магазина

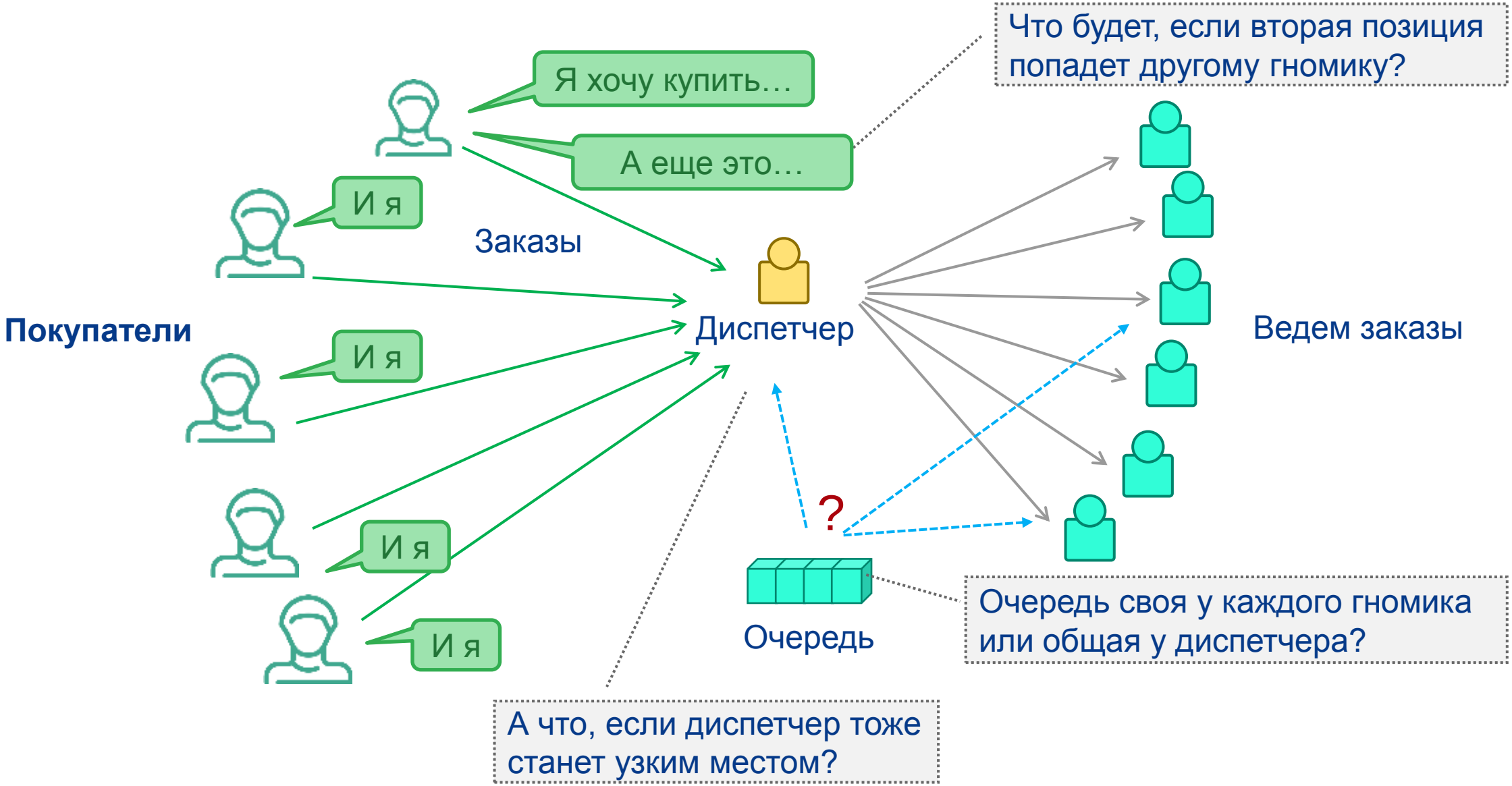


Но что происходит при масштабировании?

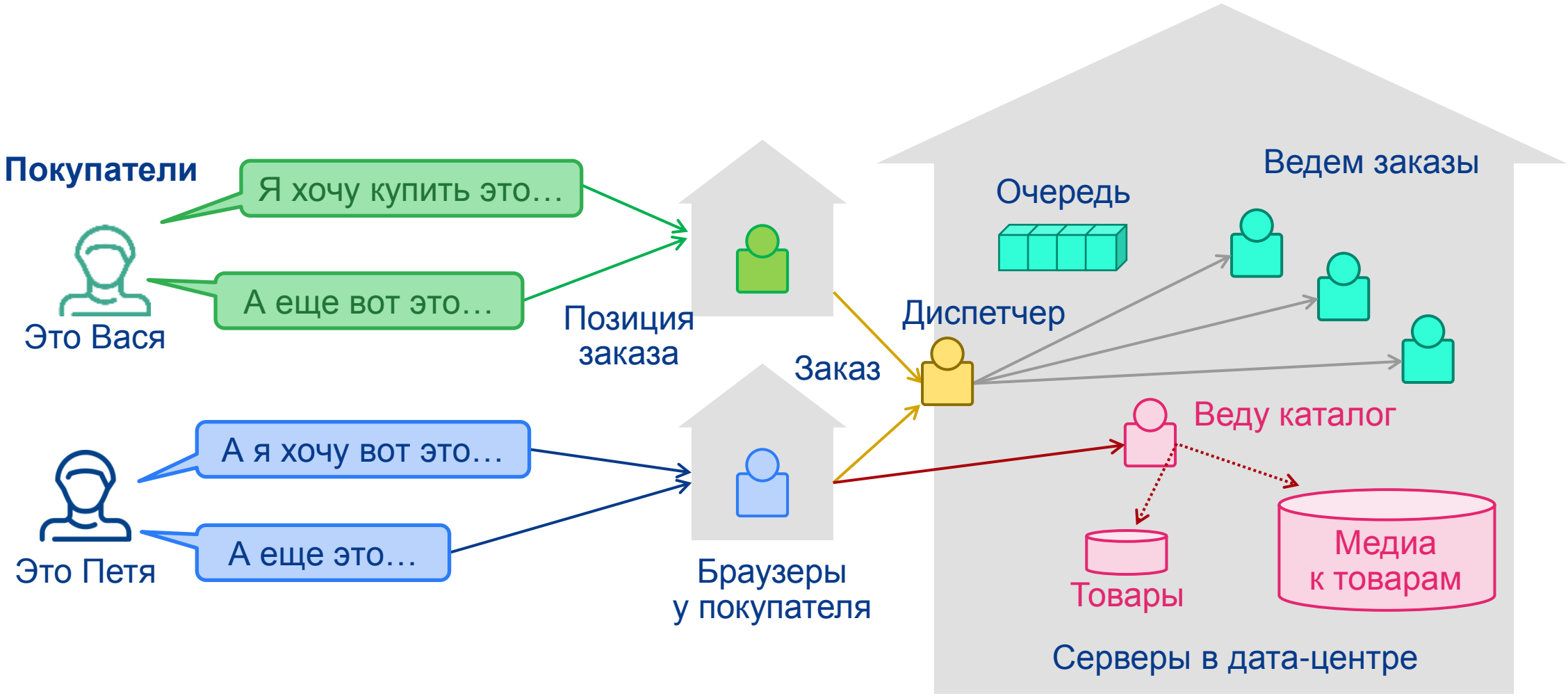
У нас много покупателей…



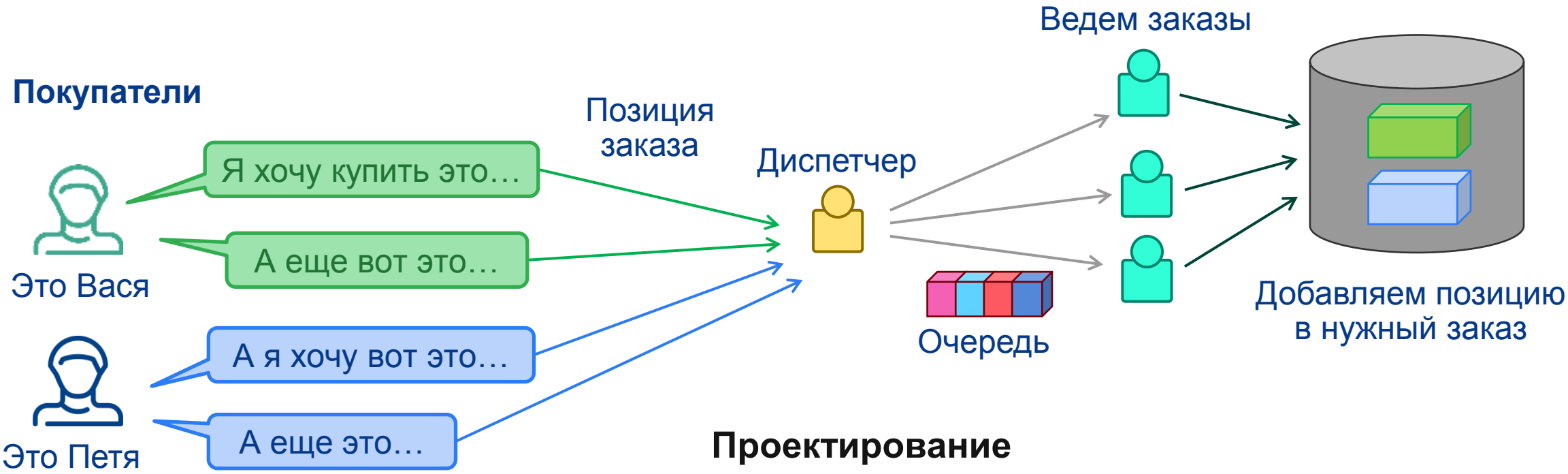
Делаем кластер сервисов приемки заказов



Собираем заказ в браузере



Общая база данных

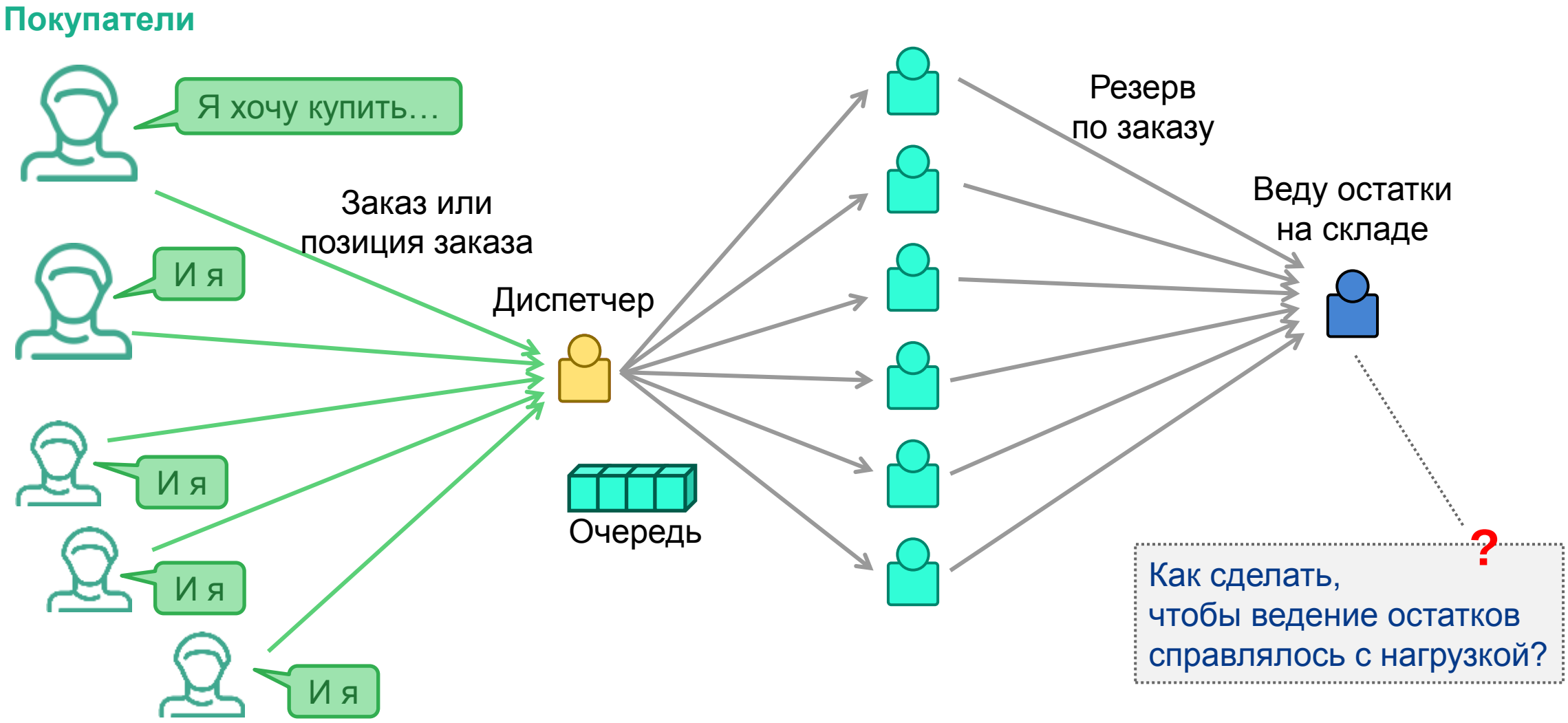


Идентификация — через авторизацию или токены

Проектирование

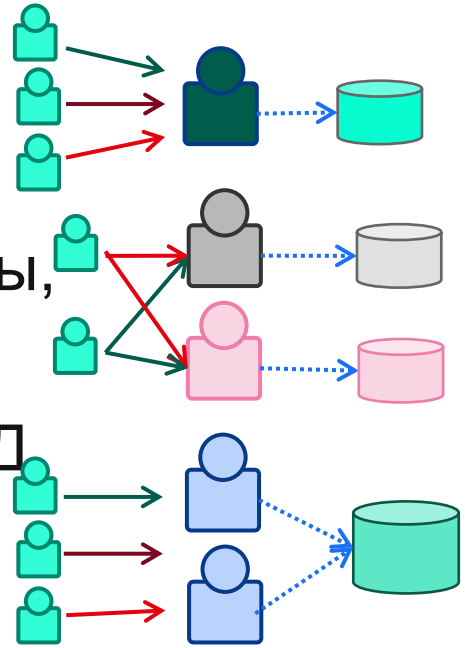
- Как покупатель, начав делать заказ, увидит корзину с другого устройства?
- Как брошенные покупателями корзины не останутся вечным мусором?

Проблема: ведение остатка на складе



Варианты ведения остатка на складе

- Очень быстрый гномик: высокопроизводительная БД и железо под узкоспециализированную логику ведения остатков
- Шардирование: несколько гномиков, каждый ведет свои товары, поделить надо равномерно
- Много гномиков логики остатков и высокопроизводительная БД, если критичны ресурсы процессора, например, встроена проверка полномочий агента на использование остатка

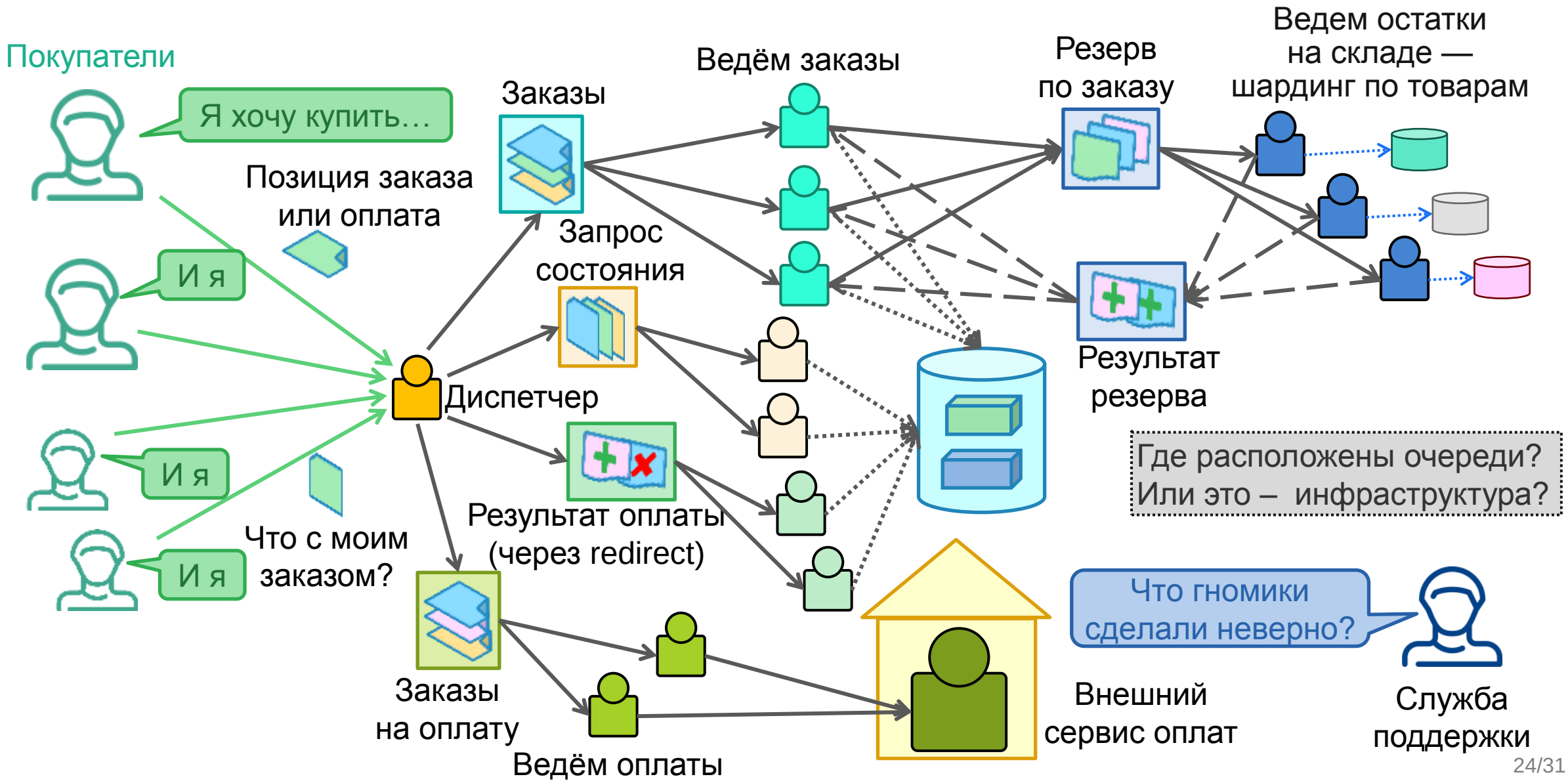


Транзакций нет, а работа асинхронная, и это порождает вопросы:

- Что делать, если зарезервировали не весь заказ?
- Что делать, если резервирование идёт долго?
- Вернее так: переводить ли заказ на оплату, если резерва долго нет?
- А что если сервис заказа упал, а резервы остались?

Запросы – через очередь (не показана)

Ведение остатка на складе и оплаты



Кейсы для проектирования

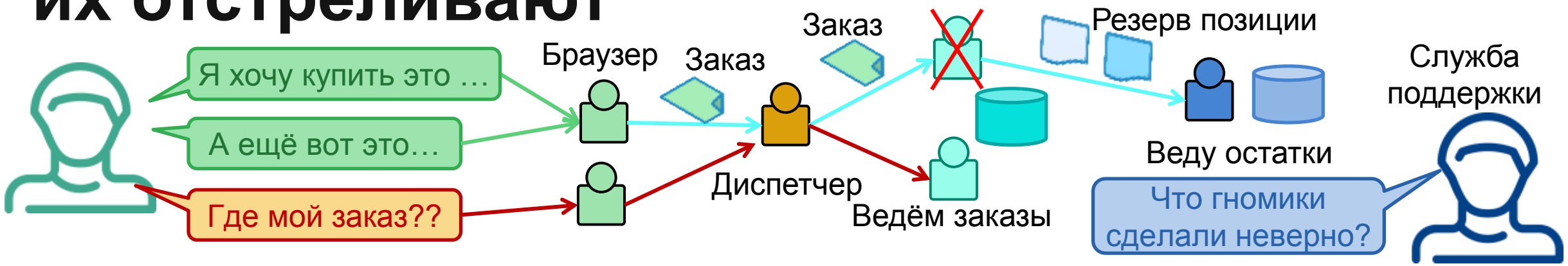
Устойчивость

- Как обрабатывается ситуация, когда покупатель очень быстро добавил позиции, и они попали разным обработчикам одновременно?
- Покупатель нажал «Оплатить», резервирование идет долго, страница оплаты не появляется — что происходит?
- Как решаются ситуации, когда результата оплаты нет?
  Можно ли отправить заказ на оплату повторно?

Масштабирование

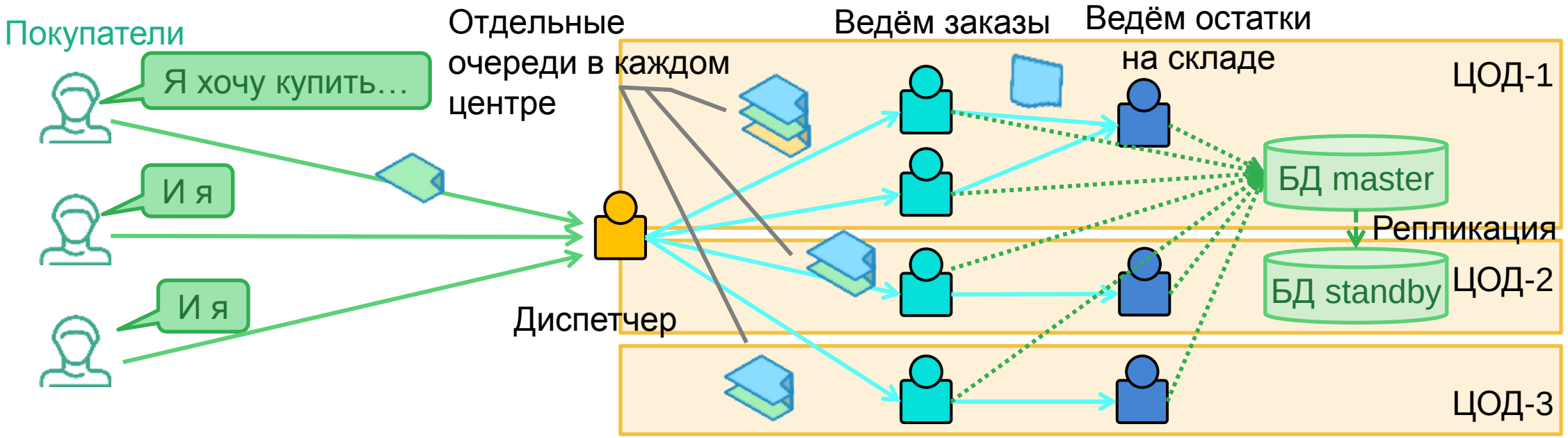
- Как происходит масштабирование по увеличению числа заказов
- Как избегают перегрузки диспетчера?
- Как происходит масштабирование по увеличению числа товаров

Устойчивость: инстансы умирают, их отстреливают



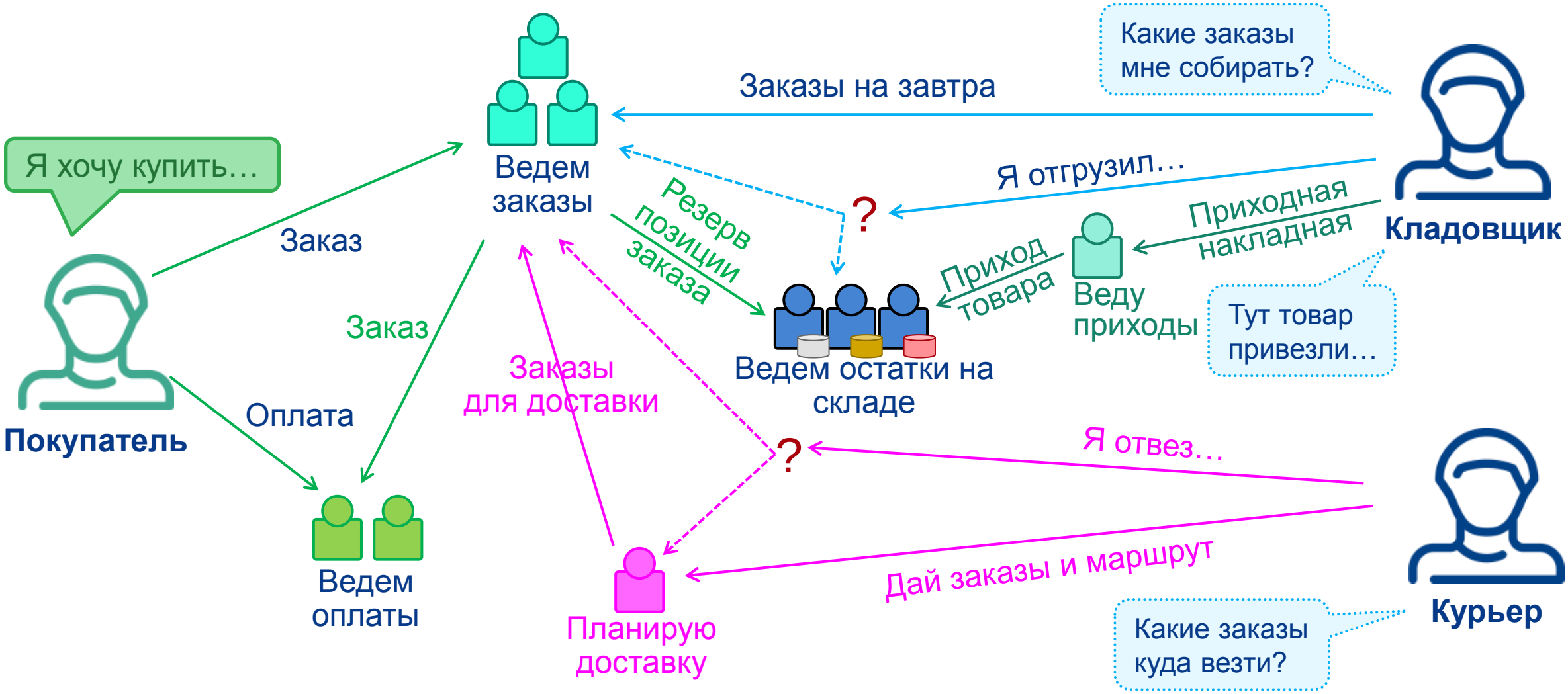
- **Ситуация**: идёт обработка и резервирование заказа, и в этот момент инстанс заказов, ведущий резервирование, падает или его убивают…
  - Как при новом обращении подхватить имеющееся резервирование?
  - Как сделать так, чтобы резервирования не зависали?
- Дополнительные сценарии для проектирования:
  - Инстанс ведения остатков не прислал ответ, но резерв при этом мог быть установлен
  - Падает инстанс остатков – как обеспечивают консистентность при восстановлении
  - Покупатель начал работу из другого браузера, пока запрос первого обрабатывается

Надёжность: ноды в разных дата-центрах

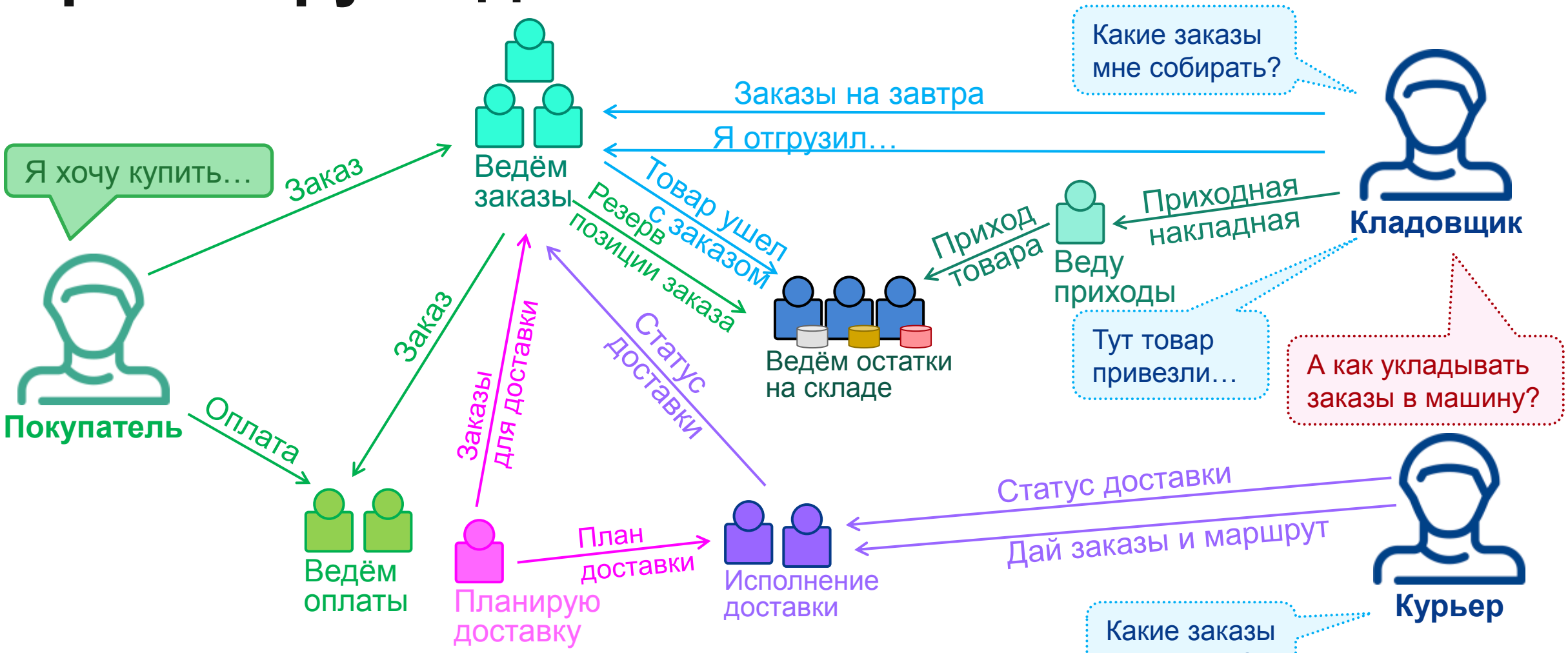


- Метафора: ЦОД — дома для гномиков, а ноды — комнаты
- Обращение в соседнее помещение дольше или невозможно
- Надо 3 ноды или ЦОДа, чтобы отличить пропажу связи от падения, в метафоре: соседний дом сгорел или телефон не работает
- Учесть: при переключении на standby часть сообщений может потеряться

Проектируем дальше: сборка и доставка…

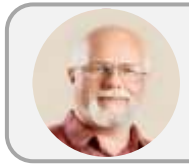


Проектируем дальше – новые кейсы



Чек-лист проектирования

- Как масштабируется каждый из сервисов под нагрузкой?
- Используется общая БД или отдельные?
- Где в БД возникают блокировки обработки запросов пользователей?
- Как взаимодействуют сервисы, где и какие очереди, что происходит с записями в очереди, когда экземпляры сервисов и ноды падают?
- Как обеспечивается устойчивость при падении экземпляров сервисов?
- Как обеспечивается устойчивость при падении нод и дата-центров?
- Как обеспечивается работа, когда пользователь едет в «Сапсане» или в другом месте с плохой связью, соединения рвутся, страницу перегружают?
- Какой мусор остается, если пользователь ушел, и как его чистят?



Лучше, если эти решения принимают не по каждому сервису отдельно, а с помощью технологической политики а использования общих шаблонов

И в заключение

СТАЧКА

- Мир изменился, старые способы описания приложений не описывают важные аспекты архитектуры: масштабирование и устойчивость
- Метафора гномиков позволяет эффективно проектировать приложения с многими сервисами в акторной модели
- **Метафора понятна** не только разработчикам, но и аналитикам, тестировщикам **и бизнесу**
- Конечно, могут потребоваться другие метафоры – ищите их

Обратная связь

Максим Цепков

http://mtsepkov.org

@MaximTsepkov

На сайте много материалов по анализу и архитектуре, Agile, ведению проектов, управлению знаниями, мои выступления, статьи и конспекты книг