CUSTIS® ИТ-РЕШЕНИЯ ДЛЯ РАЗВИТИЯ

TechLead 2021

Визуальное проектирование масштабируемых приложений

Максим Цепков

Главный архитектор решений CUSTIS

Навигатор в мире Agile, бирюзовых организаций и спиральной динамики

TechLead Conf

Москва, 1 июля 2021

CUSTIS

Немного истории: почему нужны новые модели?

Классические приложения

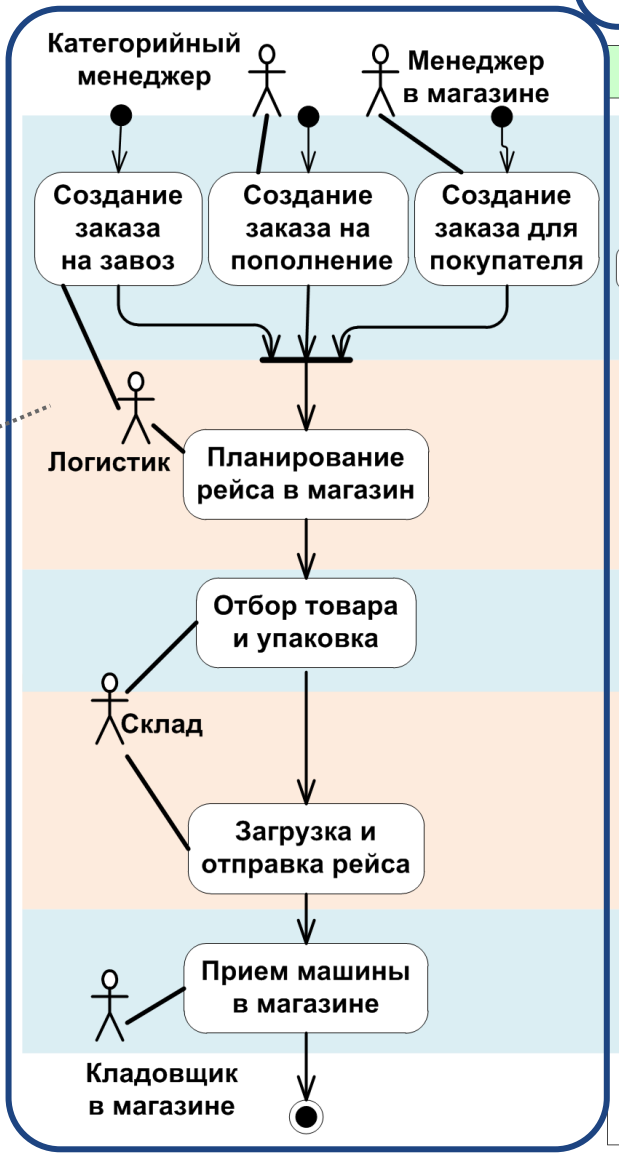
- Клиент-сервер, на сервере — СУБД, она обеспечивает конкурентную работу, транзакционность и консистентность данных в рамках обработки запроса пользователя
- Трехзвенная архитектура не принесла принципиальных изменений, она позволила писать бизнес-логику на сервере на объектном языке
- Масштабирование — за счет железа и системного софта
- Классическое проектирование: **процедурный** и **объектный подход**
- Для классических постановок есть хорошие средства визуального проектирования: ER-модель и диаграммы классов, потоков данных, модулей, последовательности и другие

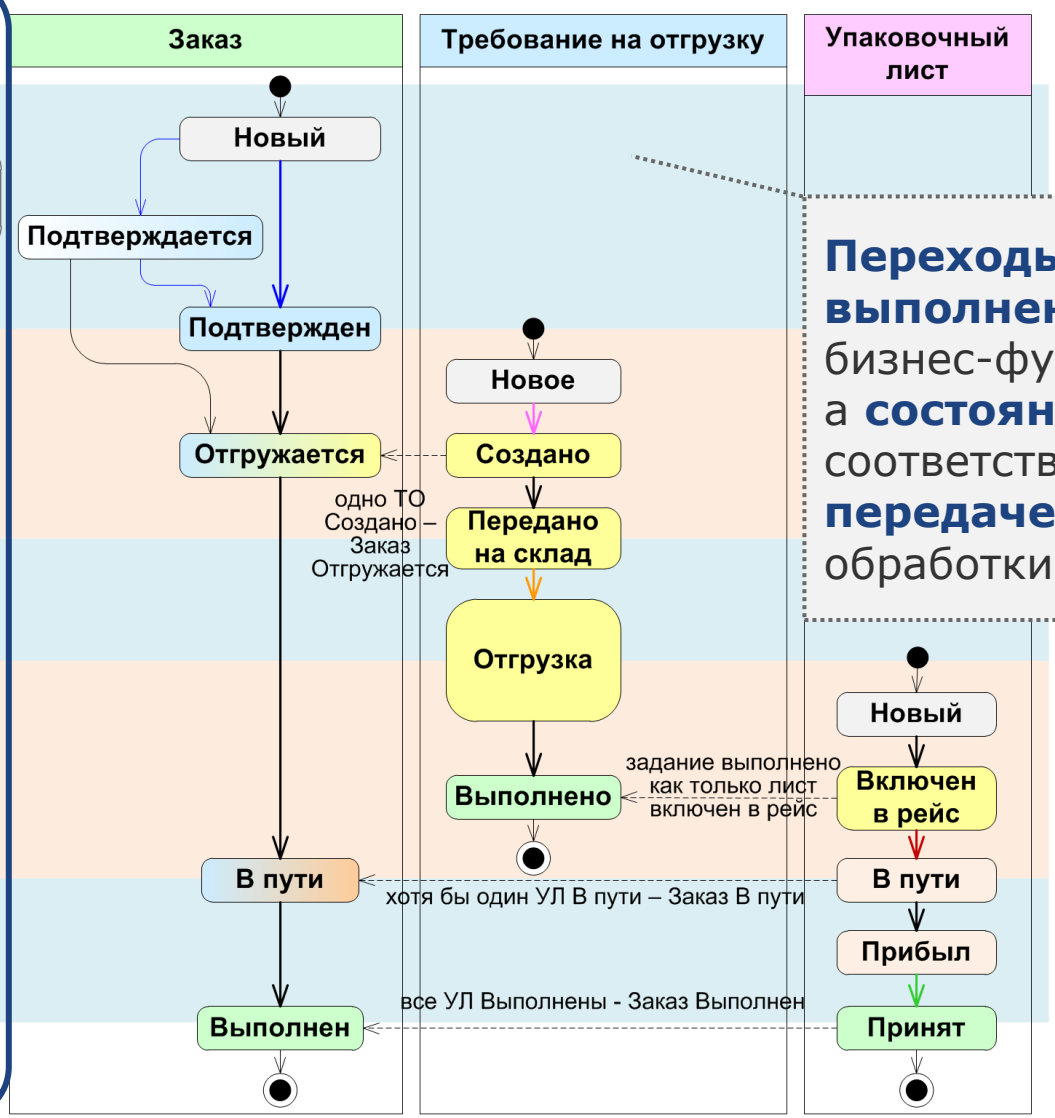
Классическая постановка

Снабжение магазинов

Выделяем объекты-документы

Бизнес-процесс: диаграмма активности





Переходы фиксируют **выполнение** бизнес-функции, а **состояния** соответствуют **передаче по этапам** обработки

Процедурный и объектный подход: пример

Задача — интернет-магазин: заказы, оплата, склад, отгрузка, доставка

Процедурный подход

- Таблицы товаров, заказов, платежей, остатков на складе, курьеров, доставок
- Алгоритмы: фиксация оплаты, назначение даты доставки, планирование курьеров
- Интерфейсы: какие экраны, какие данные показываем и действия выполняем

Объектный подход

- Объекты: товар, заказ, платеж, курьер. Доставка — отдельный объект в заказе?
- Алгоритмы: в методах, инкапсуляция внутренней логики объектов
- Интерфейсы: витрины каких объектов представляем, какие методы доступны

Если есть доставка самовывозом и курьером, то в процедурном подходе особенности во всех алгоритмах, а в объектном — делаем подтипы

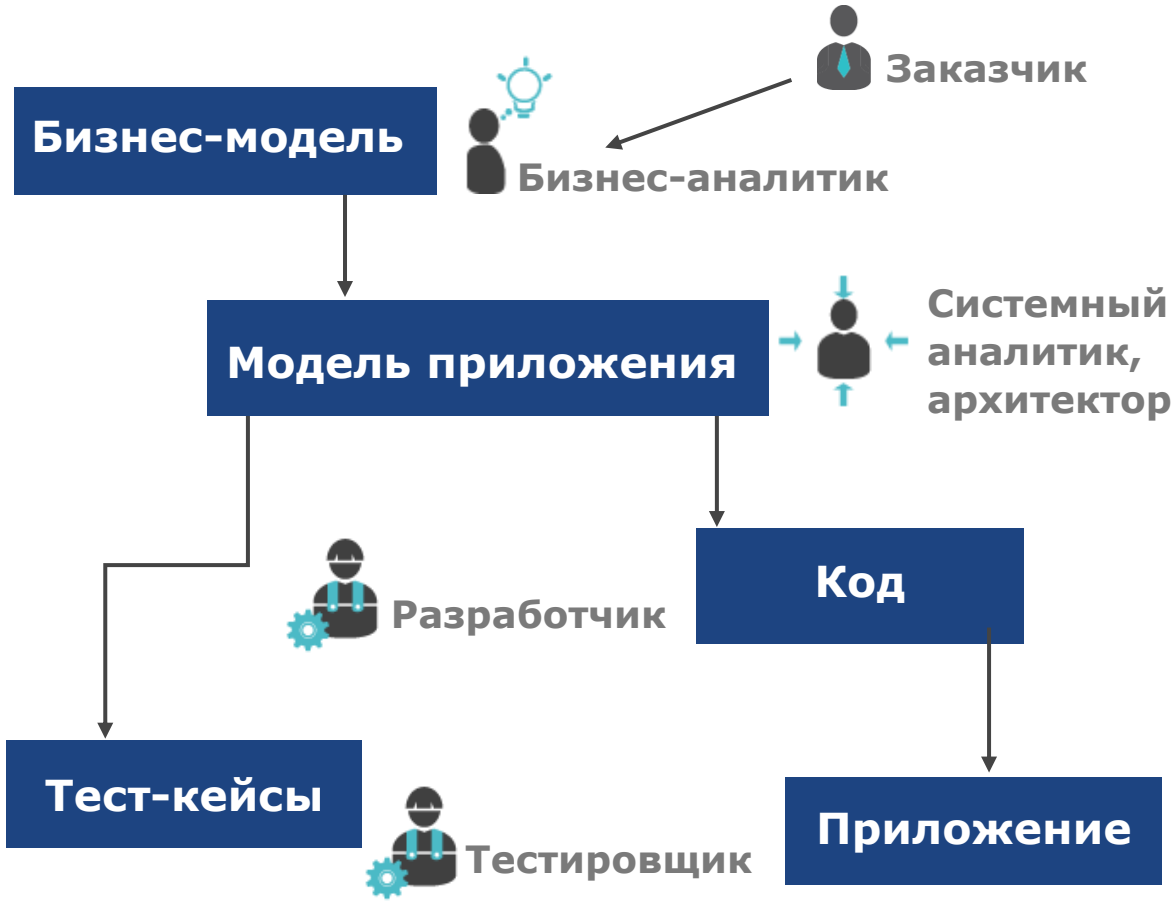
Процедурный и объектный подход

- Процедурный подход: проектируем структуру БД, интерфейсы и алгоритмы обработки. Иногда — процедуры API backend и API RPC
- Объектный подход: типы и статусы объектов, методы бизнес-логики, интерфейсы в стиле **naked object** — витрины объектов и методы, REST API
- Разница в том, где бизнес-логика — в методах объектов или отдельно
- В реализации может быть анемичная модель транспортных объектов и соответствующие тем же объектам контролеры для бизнес-логики
- **DDD** распространил объектный подход на модель предметной области: вместо словаря мы делаем онтологию понятий и связей, декомпозируя область на фрагменты и применяя методы инкапсуляции, наследования и другие, — концепция **bounded context**

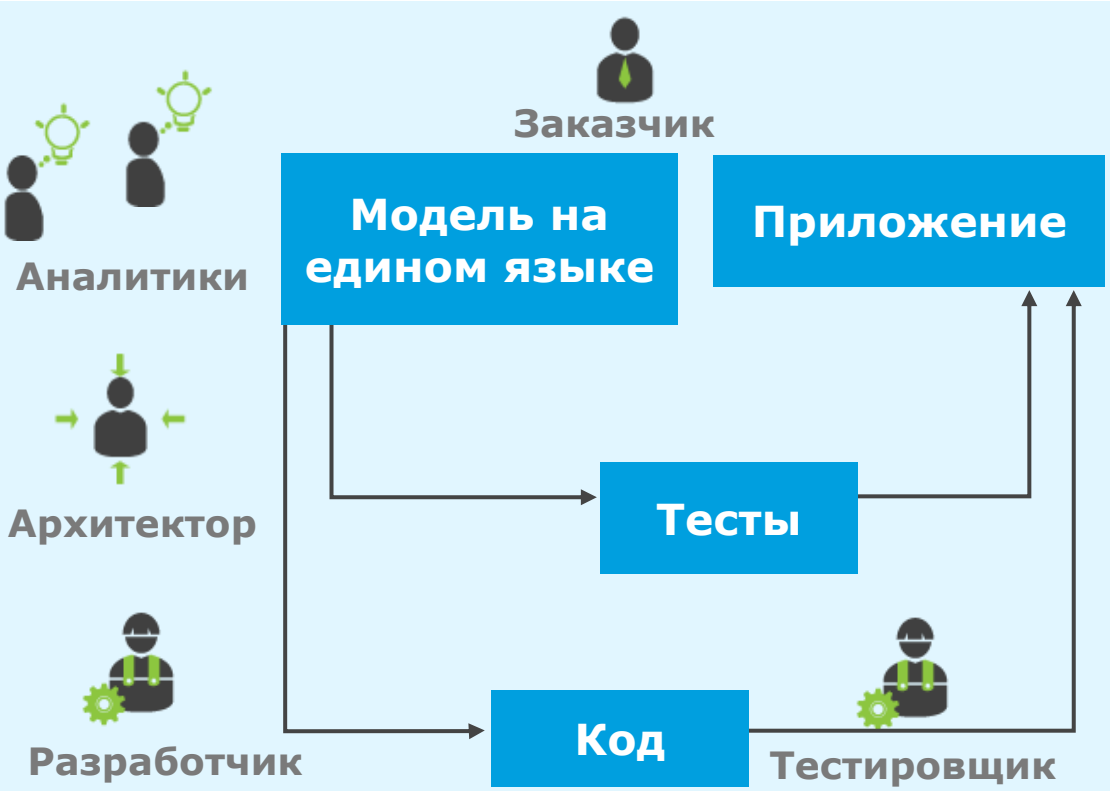
CUSTIS

DDD — единый язык и единая модель приложения

Раньше



DDD



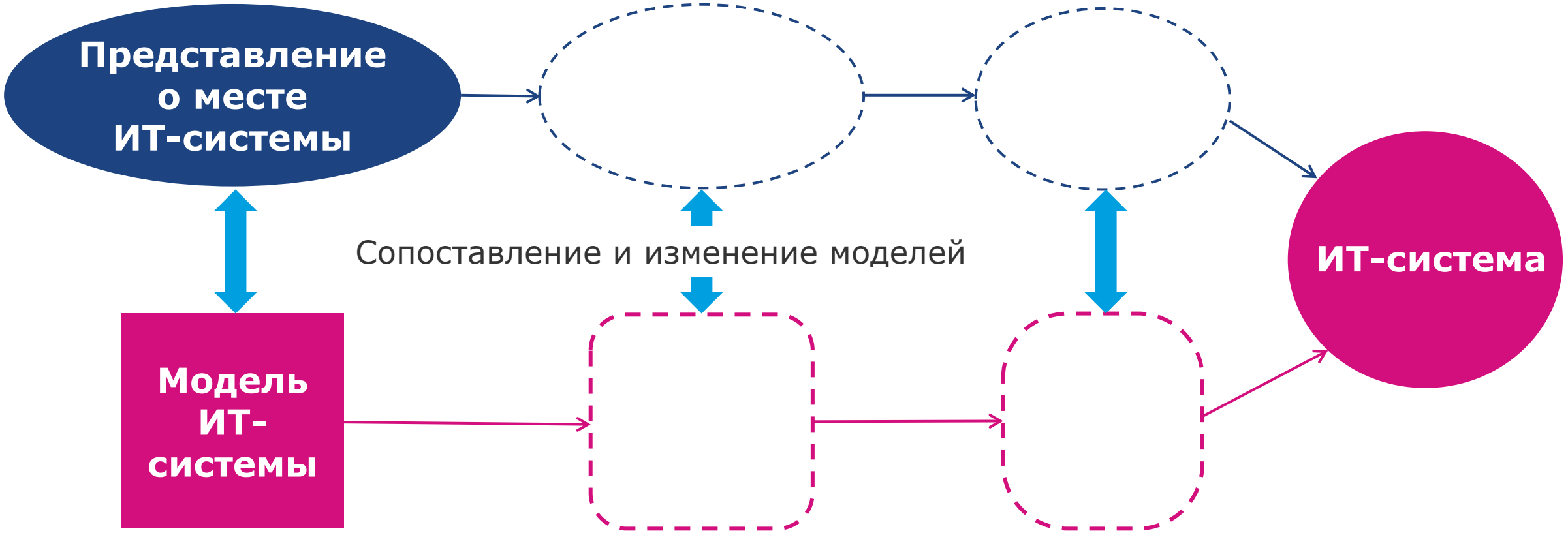
Единый язык создает общее пространство общения

Зачем нужен единый язык?

Модель системы **не соответствует** представлению бизнеса о ее месте в модели предприятия

Не то чтобы совсем не попал,
но только не попал в шарик…

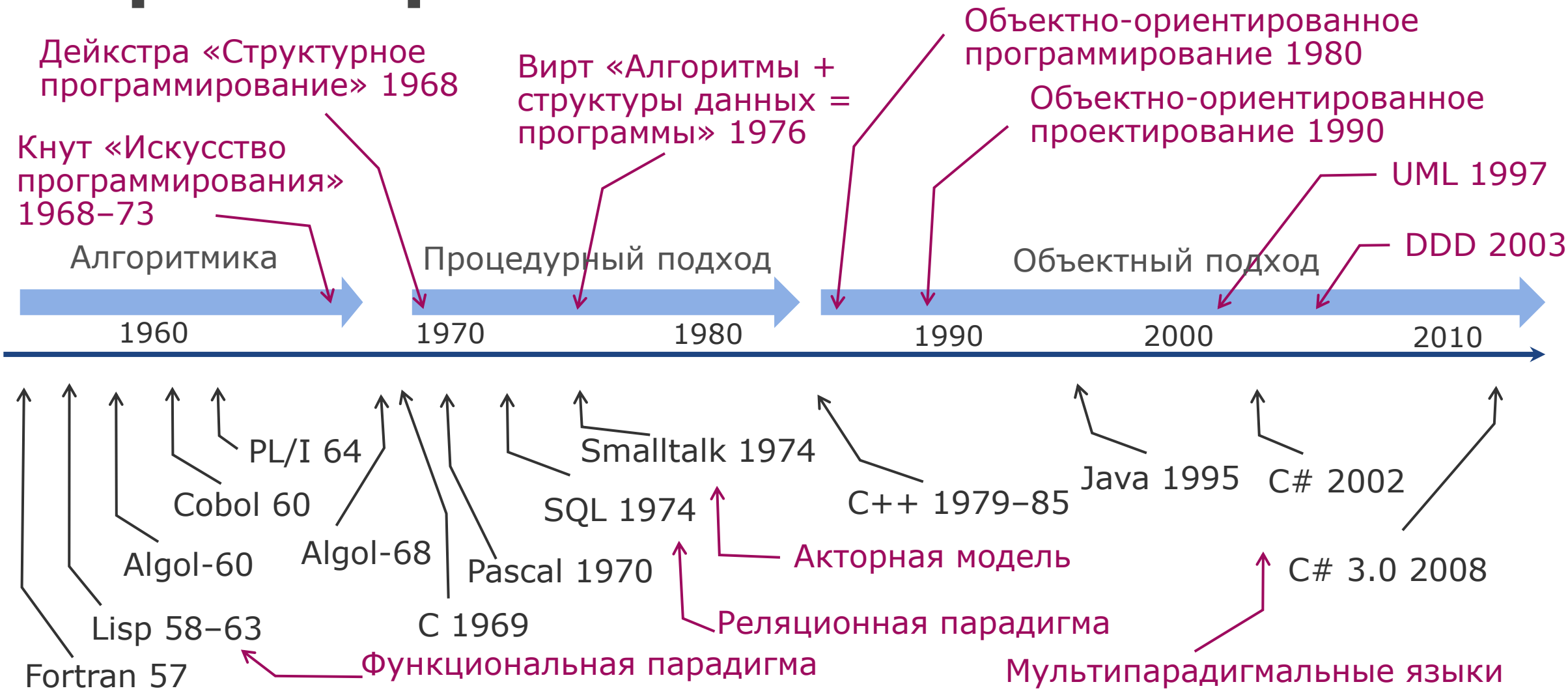
Итерационное развитие модели



Единый язык позволяет совместить модель системы
с представлениями бизнеса о ее месте

История программирования и проектирования

CUSTIS



Мои статьи «История IT. Когда компьютеры были большими...» и «История IT. ООП»

Public web потребовал масштабирования

Уход от реляционных СУБД

- NoSQL-базы данных, использование многих БД одним приложением
- Кластерное развертывание с независимым хранением на узлах
- Транзакционность и консистентность обеспечиваются в приложении
- При восстановлении узла кластера данные неконсистентны

Сервисные и микросервисные архитектуры

- Каждый бизнес-запрос обрабатывает много сервисов
- Много экземпляров одного сервиса для масштабирования
- Экземпляры сервисов падают по ошибкам или блокировкам
- Асинхронные сообщения, очереди выравнивают производительность

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

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

Гномики для интернет-магазина

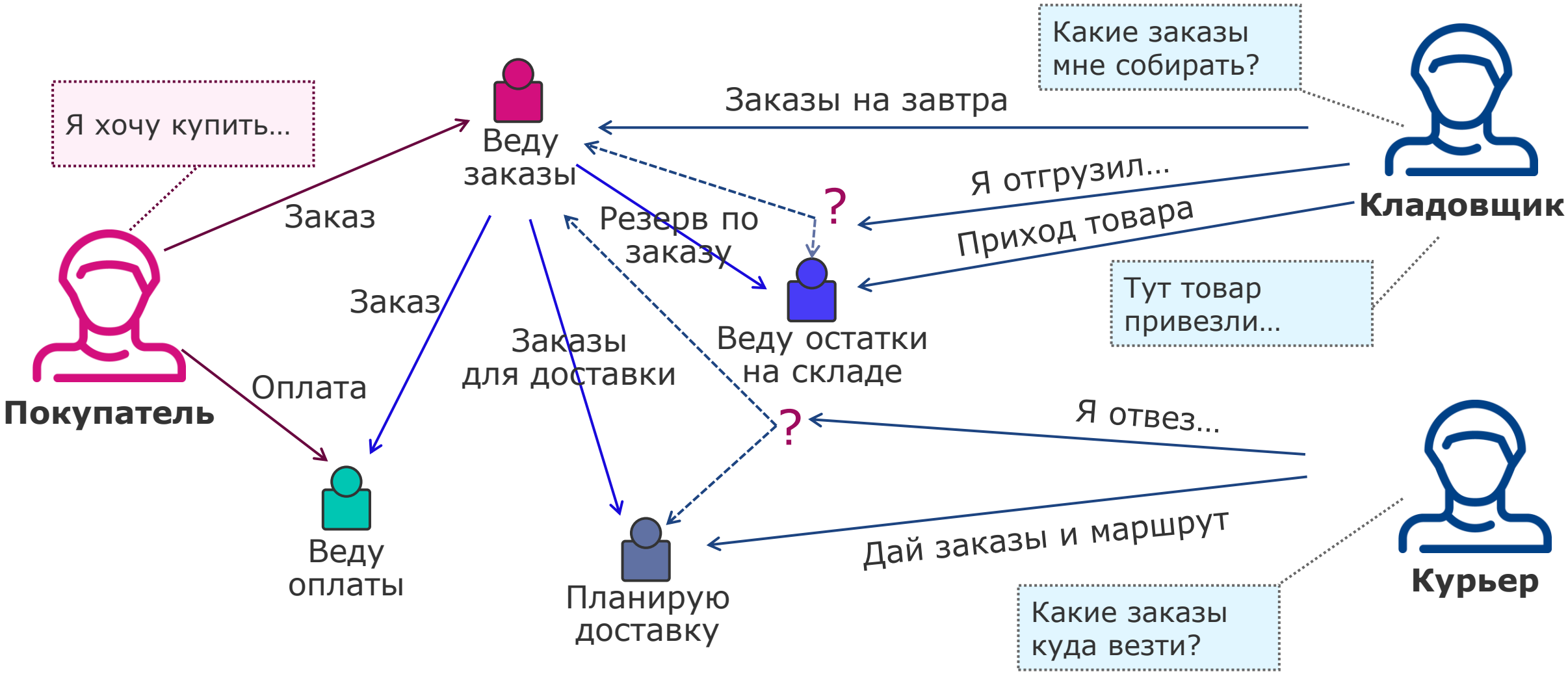
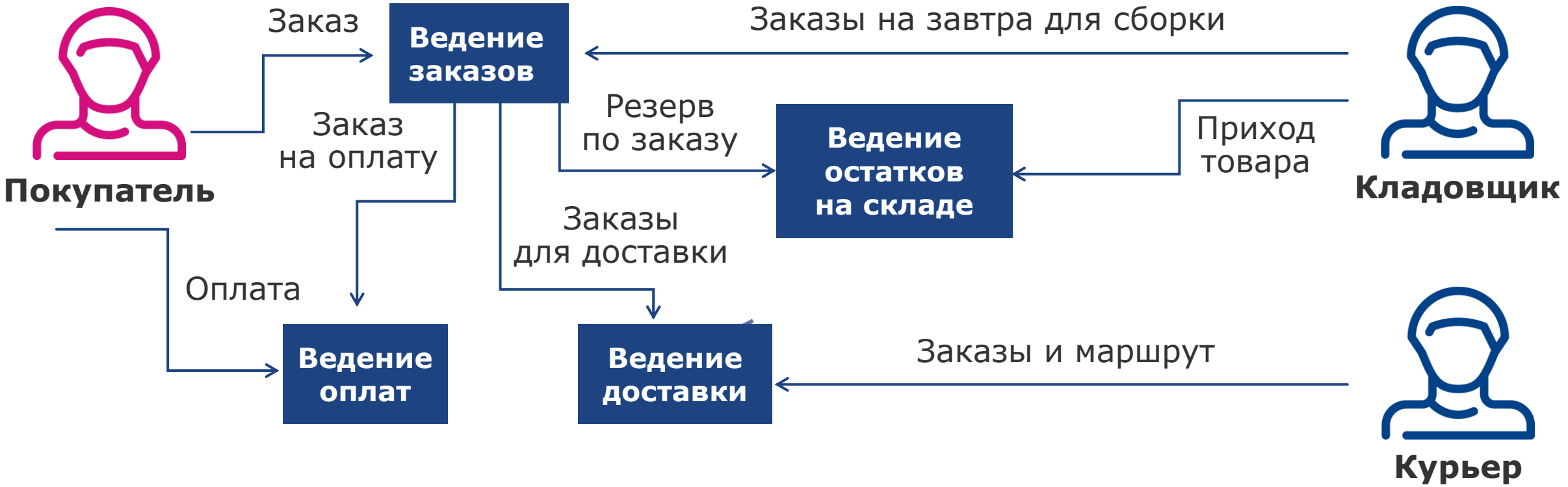
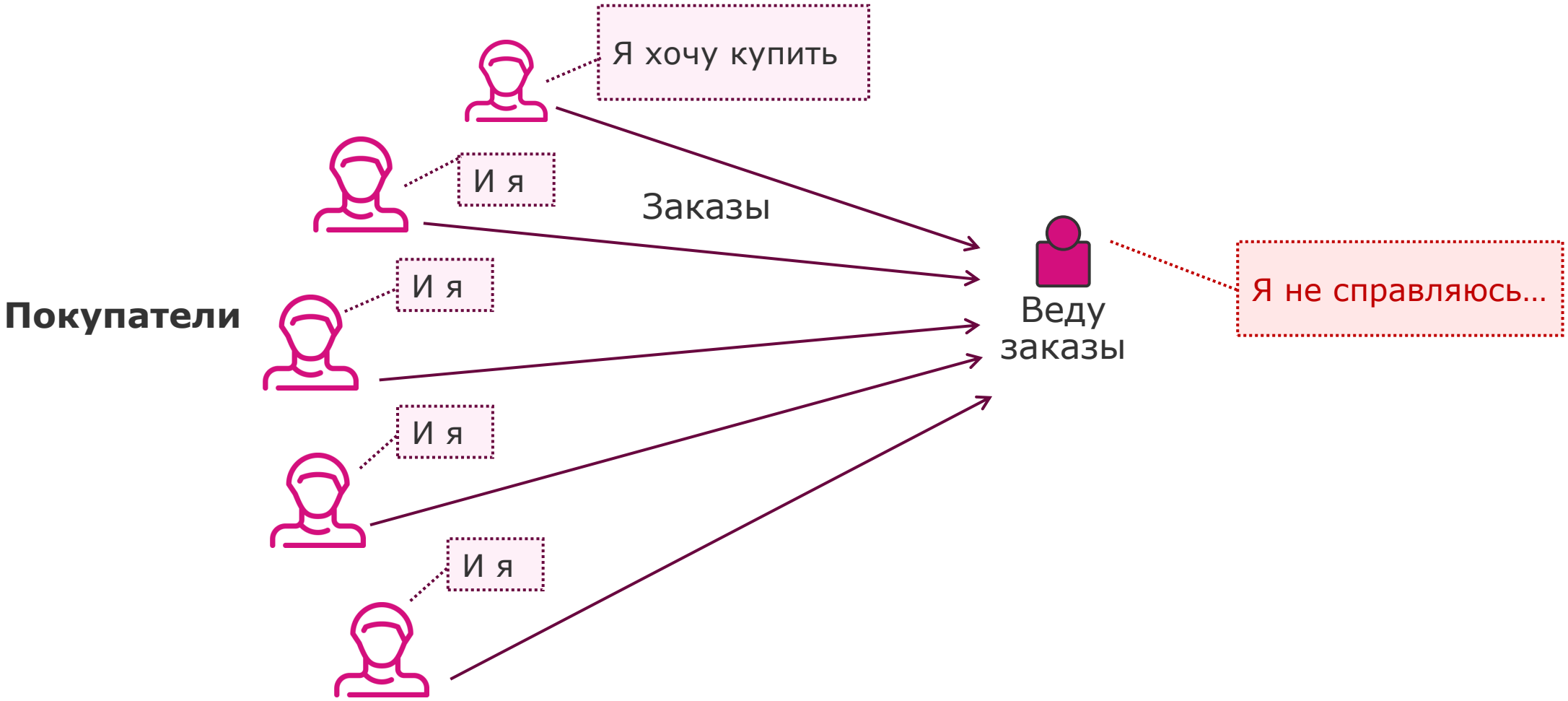


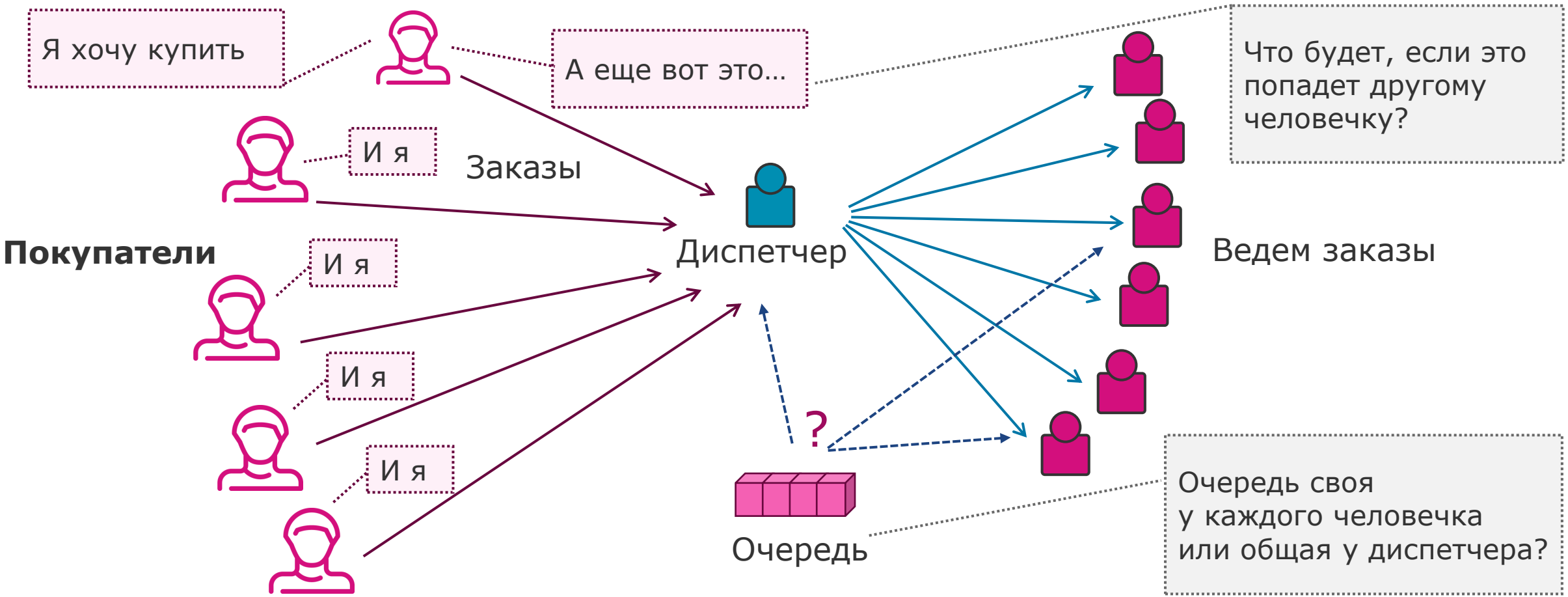
Схема сервисов для интернет-магазина



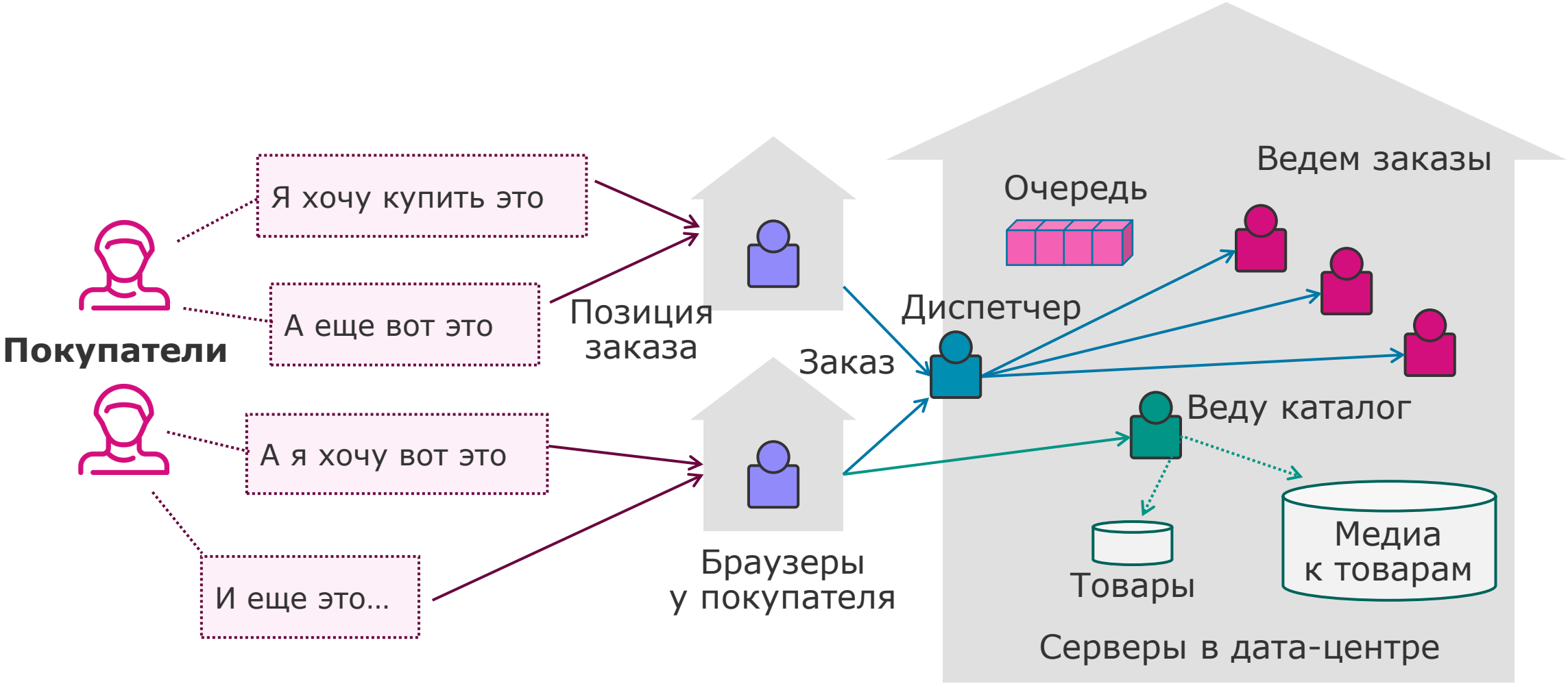
Но у нас много покупателей...



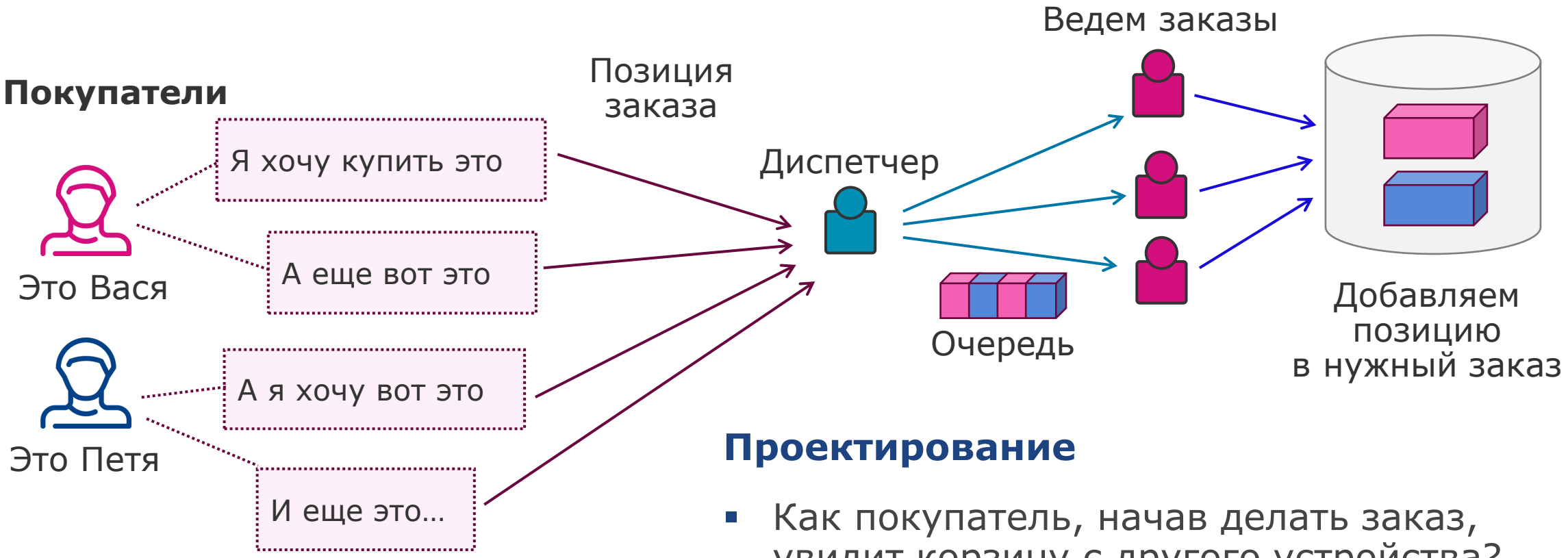
Делаем кластер сервисов приемки заказов



Собираем заказ в браузере



Общая база данных

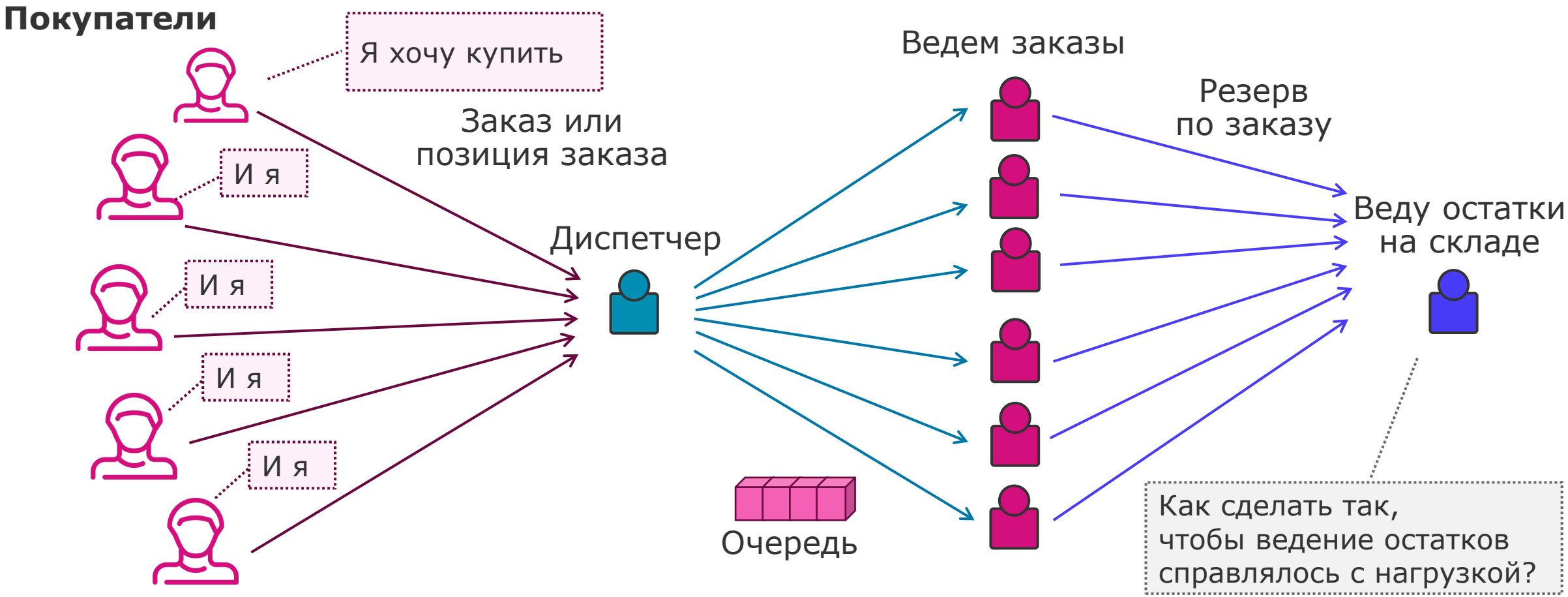


Идентификация — через авторизацию или токены.

Проектирование

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

Проблема: ведение остатка на складе



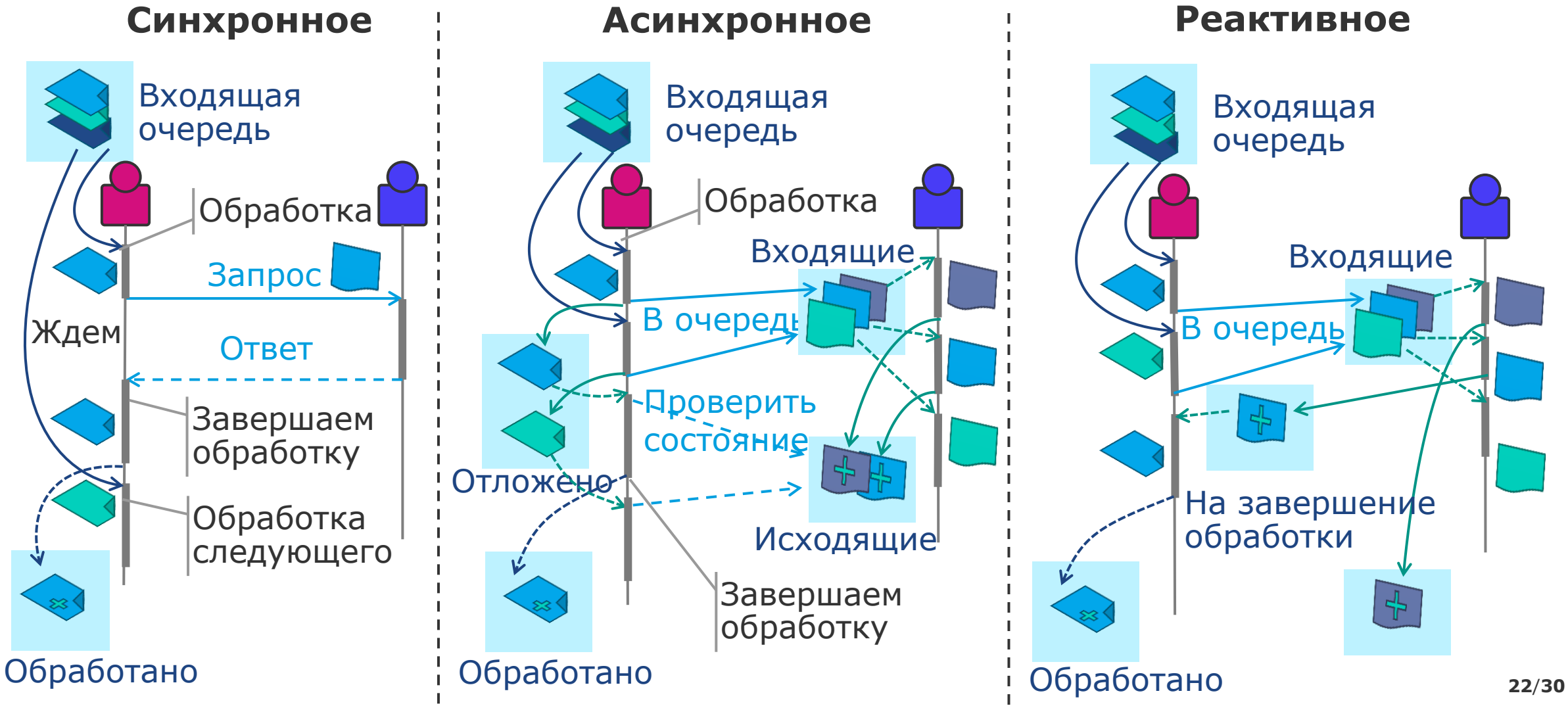
Варианты ведения остатка на складе

- 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 быстрая специализированная БД

Между гномиками заказов и гномиками остатков будет очередь на резервирование для выравнивания нагрузки

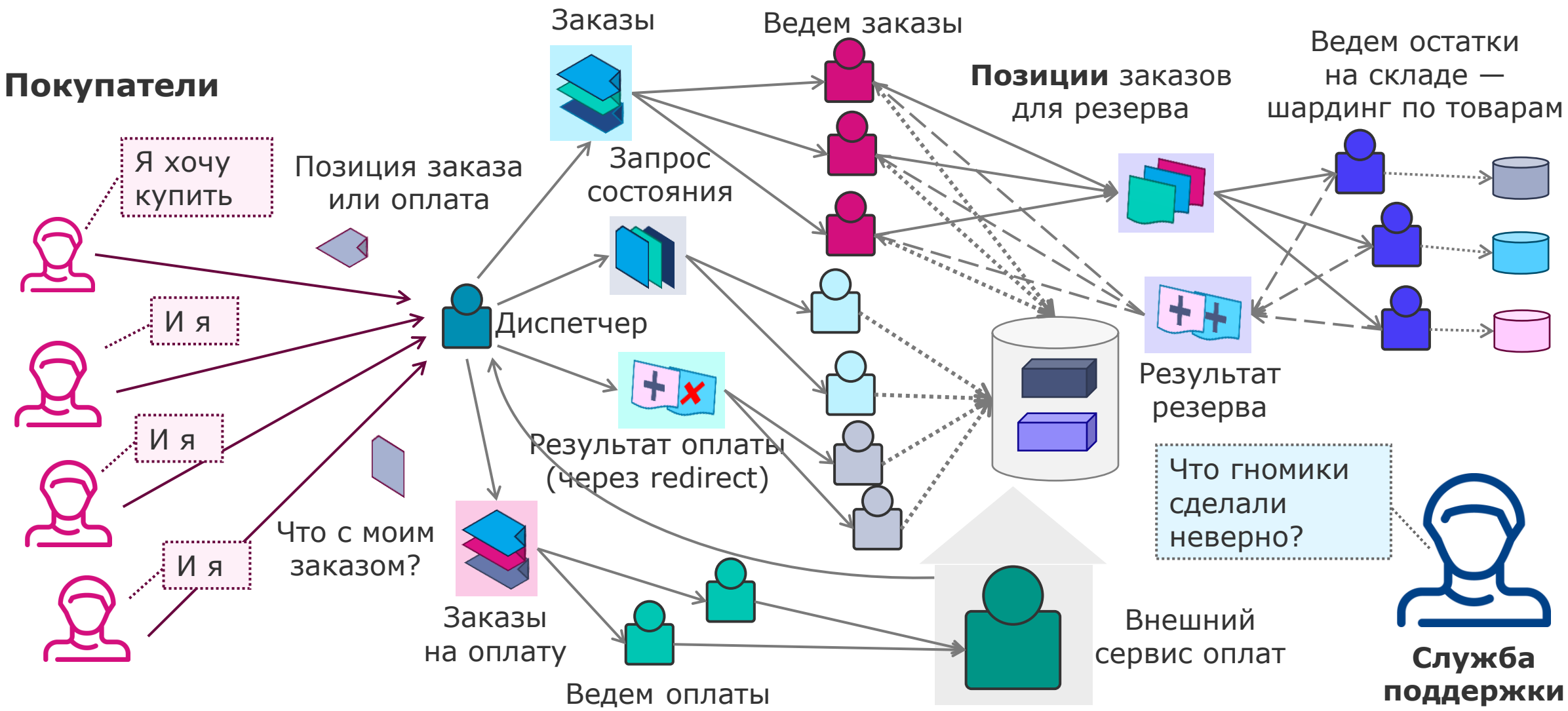
Варианты межсервисного взаимодействия

CUSTIS



Ведение остатка на складе по товарам

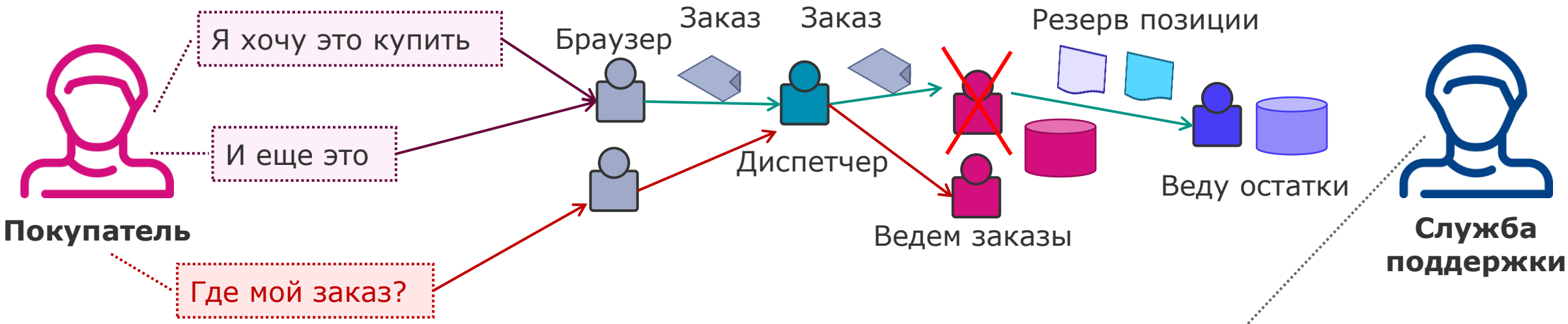
CUSTIS



Кейсы для проектирования

- 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?

Устойчивость: гномики умирают, их отстреливают



Ситуация: идет обработка и резервирование заказа, и в этот момент:

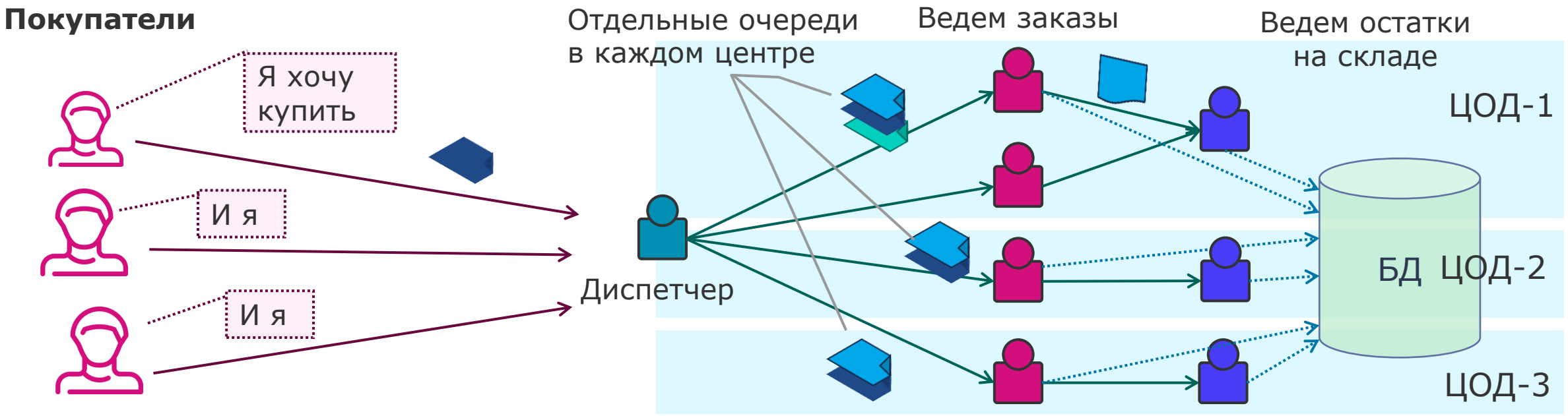
- Инстанс заказов, ведущий резервирование, падает или его убивают…
- Покупатель долго не видит ответа в браузере — и открывает новый

Вопросы:

- Как при новом обращении подхватить имеющееся резервирование?
- Как сделать так, чтобы резервирования не зависали?

Что гномики сделали неверно?

Надежность: ноды в разных дата-центрах

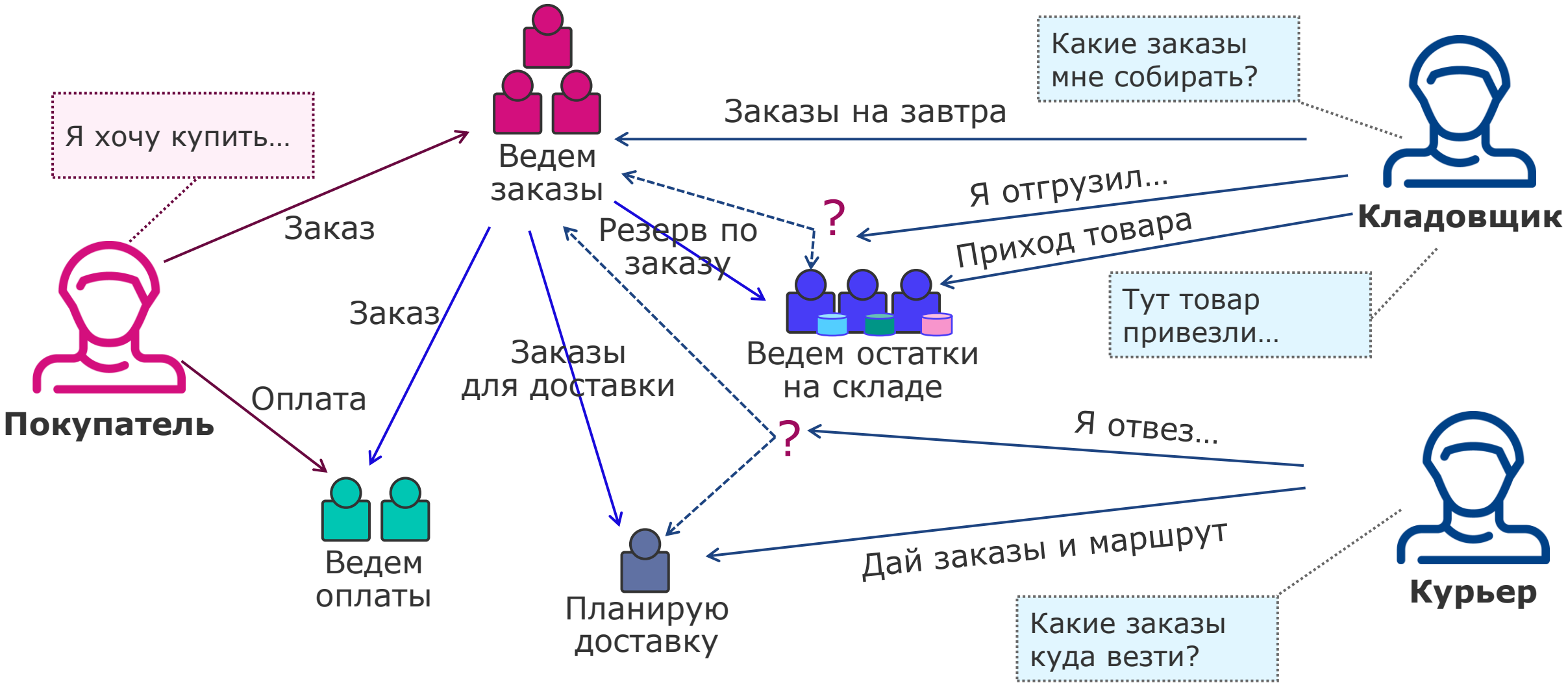


- Метафора: ЦОД — дома для гномиков, а ноды — комнаты
- Обращение в соседнее помещение — дольше или невозможно
- Надо 3 ноды или ЦОДа, чтобы отличить пропажу связи от падения, в метафоре: соседний дом сгорел или телефон не работает

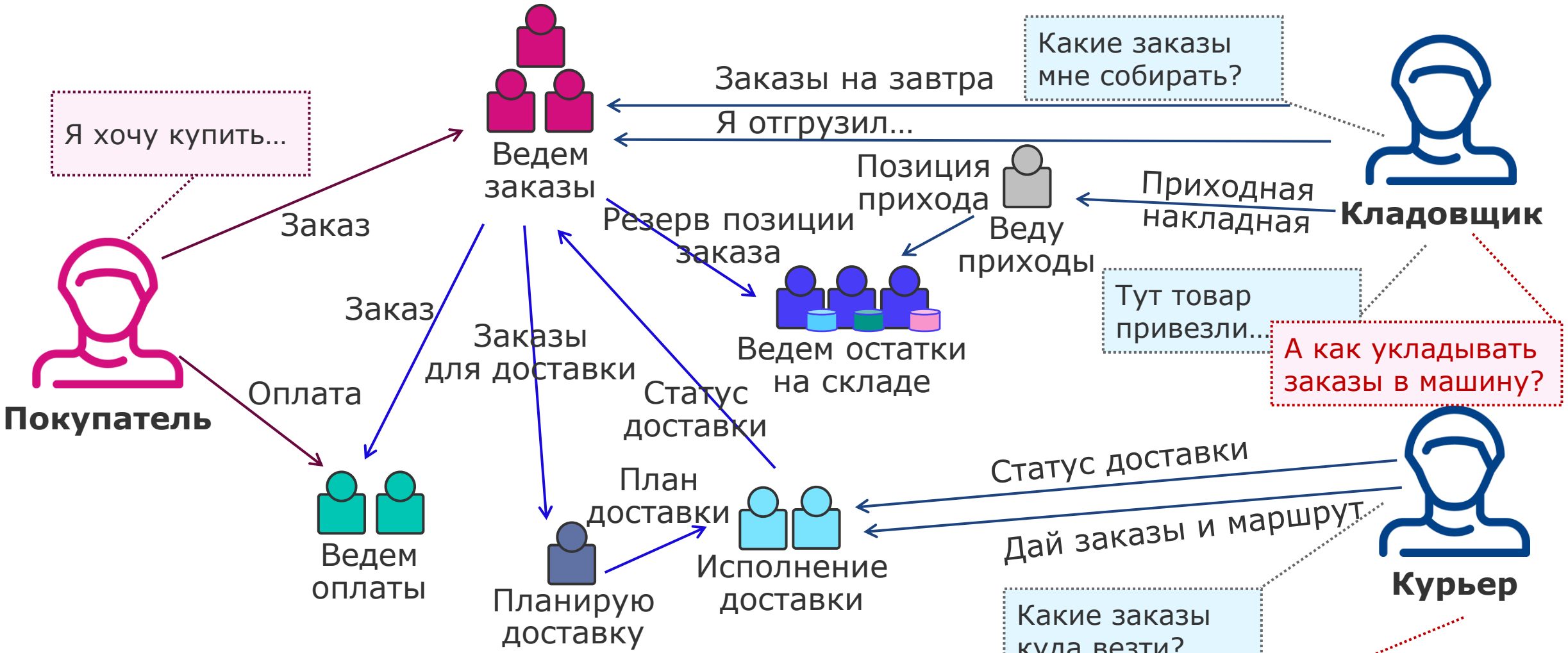
Чек-лист проектирования

- 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 в «Сапсане» или в другом месте с плохой связью, соединение рвется, и он переоткрывает страницу?
- Какой мусор остается, если пользователь ушел, и как его чистят?

Проектируем дальше



Проектируем дальше...



CUSTIS

И в заключение

- Мир изменился, и старые способы описания приложений не работают в современной архитектуре
- Метафора гномиков позволяет эффективно проектировать приложения в акторной модели и понятна не только разработчикам
- Но могут потребоваться и другие метафоры, ищите их

Максим Цепков

http://mtsepkov.org
Telegram: @MaximTsepkov

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

Вакансии техлидов

Пишите на hr@custis.ru, подходите с вопросами