Распределенные вычислительные системы

Архитектура клиент-серверных распределенных приложений.

Модель взаимодействия клиент-сервер

Проблема: централизация vs децентрализация

- Это одна из старейших проблем в IT
- Пример:
  - *King J.L. Centralized versus decentralized computing: organizational considerations and management options // ACM Computing Surveys. Vol. 15, Issue 4. 1983. P. 319-349.*

До 70-х годов: централизованная модель

- До середины 70-х годов прошлого века доминировала централизованная модель:
  - Высокая стоимость телекоммуникационного оборудования
  - Слабая мощность вычислительных систем

80-Е – 90-Е: МЕЙНФРЕЙМЫ

- Появление систем разделения времени и удаленных терминалов - предпосылка возникновения клиент-серверной архитектуры.
- Ресурсы мейнфреймов предоставлялись конечным пользователям посредством удаленного соединения.
- Дальнейшее развитие телекоммуникационных систем и появление персональных компьютеров дало толчок развитию клиент-серверной парадигме обработки данных

Клиент-серверная архитектура

- Согласно парадигме клиент-серверной архитектуры:
  - один или несколько клиентов и один или несколько серверов
  - совместно с базовой операционной системой
  - и средой взаимодействия
  - образуют единую систему, обеспечивающую распределенные вычисления, анализ и представление данных

Применение клент-серверной модели

- Процесс разработки распределенных приложений достаточно сложен и одной из наиболее важных задач является решение того, как
  - *функциональность приложения должна быть распределена между клиентской и серверной частью.*

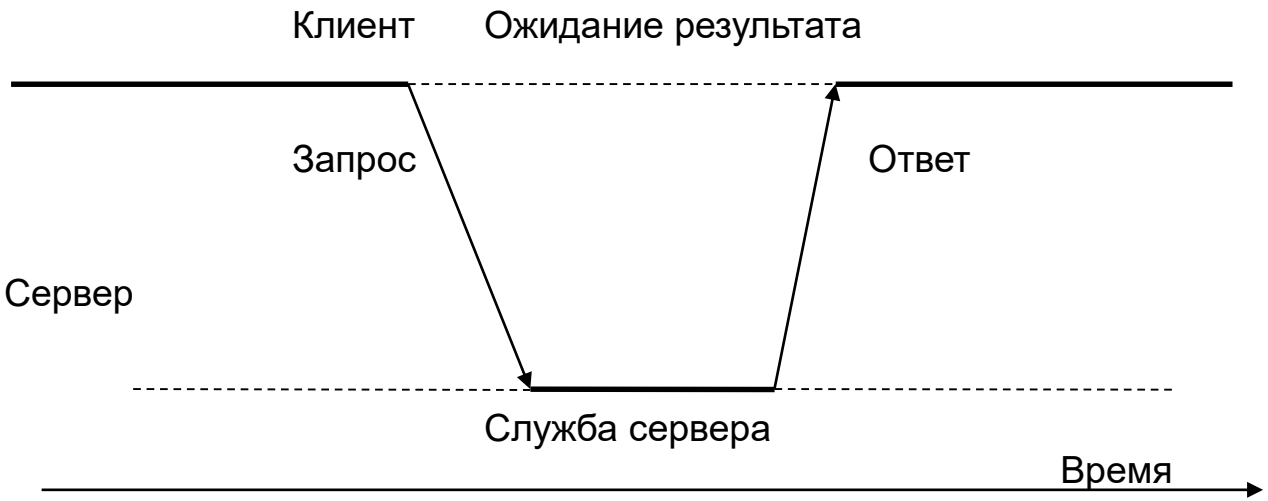
Алгоритм работы клиент-серверной архитектуры

В классическом случае данная схема функционирует следующим образом:

- клиент формирует и посылает запрос на сервер;
- сервер производит необходимые манипуляции с данными, формирует результат и передаёт его клиенту;
- клиент получает результат, отображает его на устройстве вывода и ждет дальнейших действий пользователя.

Цикл повторяется, пока пользователь не закончит работу с сервером.

Обобщение взаимодействия «Клиент-Сервер»



Надежность сети

- Если базовая сеть надежна как локальная сеть, взаимодействие может быть реализовано простым протоколом, без установления соединения (выигрыш эффективности)
- Что, если сообщения пропадают? Если теряется ответ?
  - Отправлять повторно?
  - Надеяться на лучшее?
- А если это операция перевода 10 000$ с одного счета на другой?

Надежные протоколы соединения

- В прикладных протоколах используется TCP/IP:
  - До посылки запроса клиент должен установить соединение
  - Сервер использует то же соединение для посылки ответа

Концепции организации серверной архитектуры

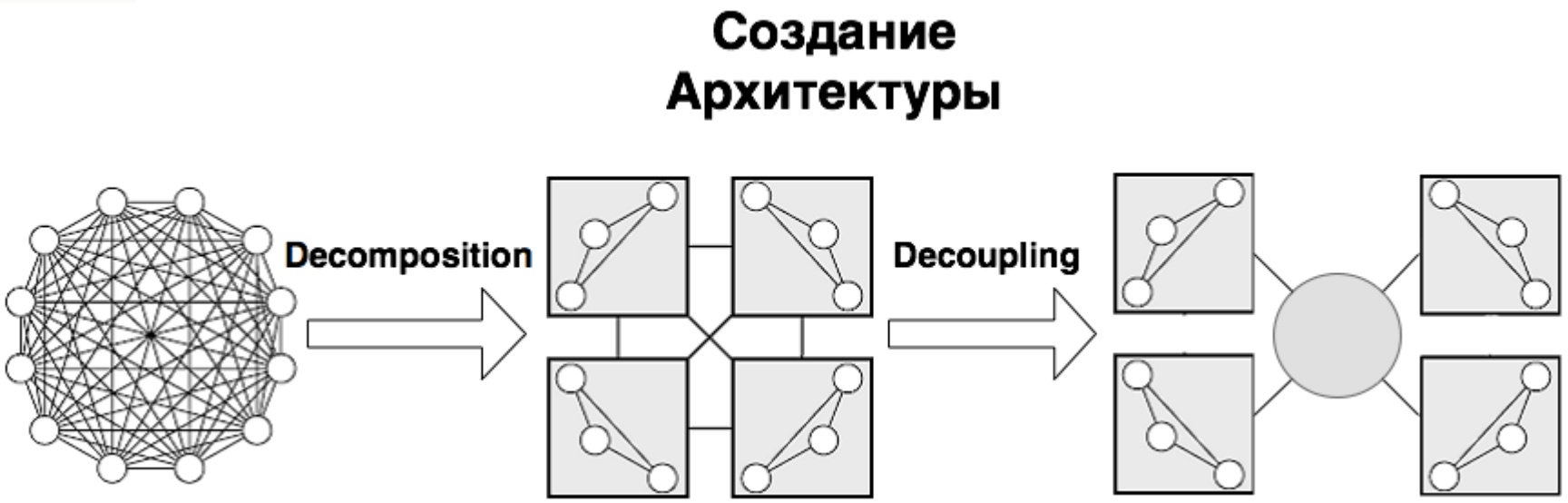
ЧТО ТАКОЕ ПРОЕКТИРОВАНИЕ И АРХИТЕКТУРА?

- **Проектирование ПО** – это осознанный выбор решений о логической организации составных частей программного комплекса.
- Проектирование – это творческий процесс, который тяжело описать в виде формальных структурированных методов.
- **Архитектура** — это **организация системы**, воплощенная в ее **компонентах**, их **отношениях** между собой и с **окружением**, а также определяет принципы ее проектирования и развития. (IEEE 1471)

ВЛИЯНИЕ НЕФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ НА АРХИТЕКТУРУ ПО

- **Производительность и эффективность** – локализация критически-важных операций
- **Безопасность** – слоистая архитектура
- **Высокая доступность** – избыточность используемых компонентов
- **Расширяемость и гибкость** – мелкомодульная архитектура слабосвязанных компонентов

Создание модульной архитектуры

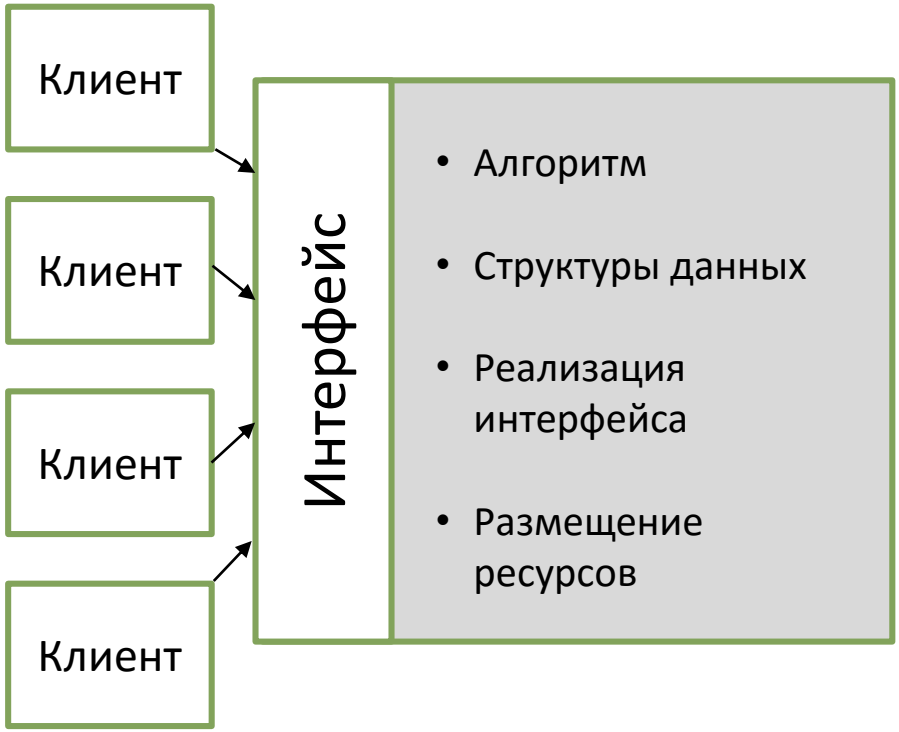


- Главная задача при разработке больших систем - снижение сложности.
- Принцип: **«разделяй и властвуй» (divide et impera)**, но по сути речь идет об иерархической декомпозиции.
- Помимо снижения сложности, обеспечивается гибкость системы, возможности для масштабирования, повышается устойчивость за счет дублирования критически важных частей.

Создание архитектуры программы или как проектировать табуретку
https://habrahabr.ru/post/276593/

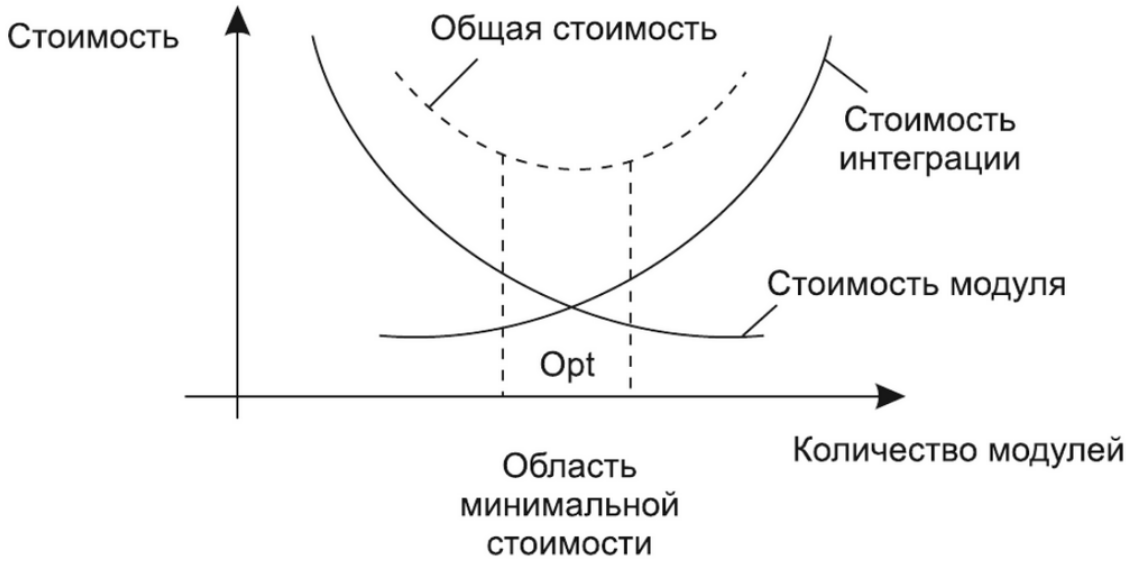
Модульность

- Программная система делится *на именуемые и адресуемые компоненты*, часто называемые *модулями*, которые затем интегрируются для совместного решения проблемы.
- **Модуль** - фрагмент программного текста, являющийся строительным блоком для физической структуры системы.
- Как правило, модуль состоит из интерфейсной части и части-реализации.



Модульность

- Модульность — свойство ПО, обеспечивающее интеллектуальную возможность создания сколь угодно сложной программы.
- Но необходимо учитывать затраты на дальнейшую интеграцию модулей.



Информационная закрытость

- Информационная закрытость означает, что:
  - Все модули независимы и обмениваются информацией только необходимой для работы
  - Доступ к операциям и структурам данных модуля ограничен.
- Это позволяет:
  - Обеспечить разработку модулей различными независимыми коллективами;
  - Обеспечить легкую модификацию системы
- Идеальный модуль – это черный ящик, содержимое которого не видно клиенту.

Достоинства хорошей архитектуры

- **Масштабируемость (Scalability)**
  возможность расширять систему и увеличивать ее производительность, за счет добавления новых модулей.
- **Ремонтопригодность (Maintainability)**
  изменение одного модуля не требует изменения других модулей
- **Заменимость модулей (Swappability)**
  модуль легко заменить на другой
- **Возможность тестирования (Unit Testing)**
  модуль можно отсоединить от всех остальных и протестировать / починить
- **Переиспользование (Reusability)**
  модуль может быть переиспользован в других программах и другом окружении
- **Сопровождаемость (Maintenance)**
  разбитую на модули программу легче понимать и сопровождать

Создание архитектуры программы или как спроектировать табуретку
https://habrahabr.ru/post/276593/

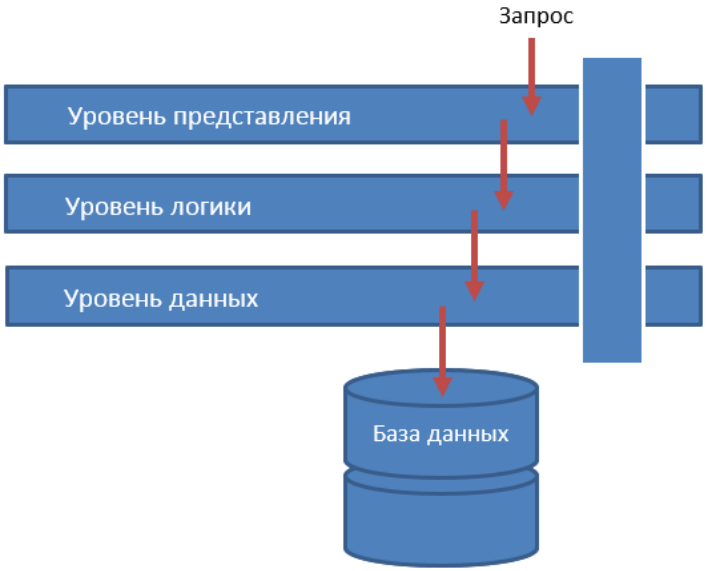
АРХИТЕКТУРНЫЕ ПАТТЕРНЫ

- Не существует формального перечня существующих архитектурных паттернов
- Но на сегодняшний день можно выделить основные (часто-используемые) подходы:
  - Многоуровневая архитектура
  - Архитектура, управляемая событиями (Event-Driven Architecture)
  - Сервис-ориентированая (микросервисная) архитектура

Немного об архитектурах программного обеспечения
https://habrahabr.ru/company/it-grad/blog/276297/

Многоуровневая (слоистая) архитектура

- Является одной из самых известных архитектур, в которой каждый слой выполняет определенную функцию. В зависимости от ваших нужд вы можете реализовать любое количество уровней.
- Является одной из самых известных архитектур, в которой каждый слой выполняет определенную функцию. В зависимости от ваших нужд вы можете реализовать любое количество уровней.

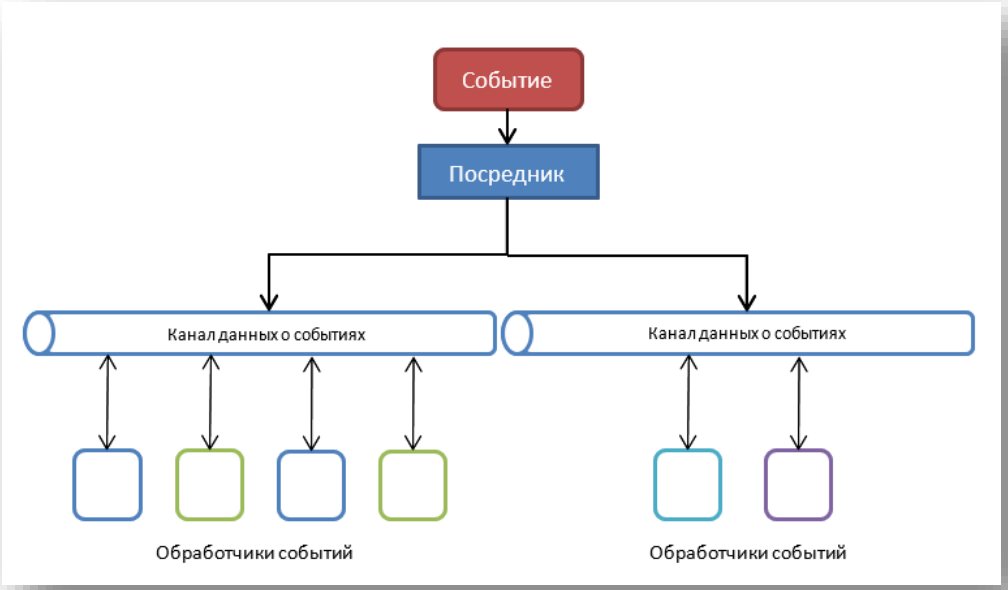


- Достоинствами применения такой архитектуры являются простота разработки (в основном из-за того, что этот вид архитектуры всем знаком) и простота тестирования.
- Среди недостатков можно выделить возможные сложности с производительностью и масштабированием – всему виной необходимость прохождения запросов и данных по всем уровням (опять же, в том случае, если все слои являются закрытыми).

Немного об архитектурах программного обеспечения https://habrahabr.ru/company/it-grad/blog/276297/

Архитектура, управляемая событиями (Event-Driven Architecture)

- Широко используется для создания масштабируемых систем.
- Обработчики являются изолированными независимыми компонентами, отвечающими (в идеале) за какую-нибудь одну задачу, и содержат бизнес-логику, необходимую для работы.

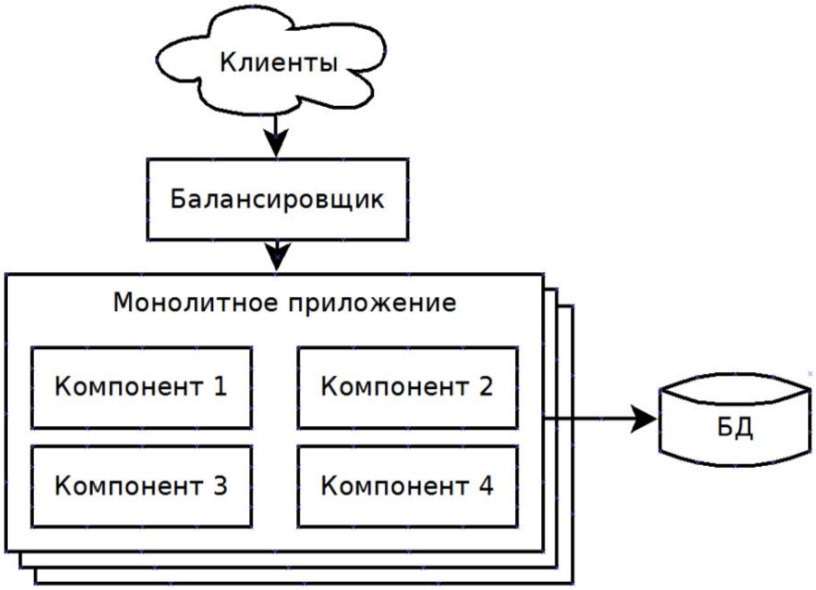


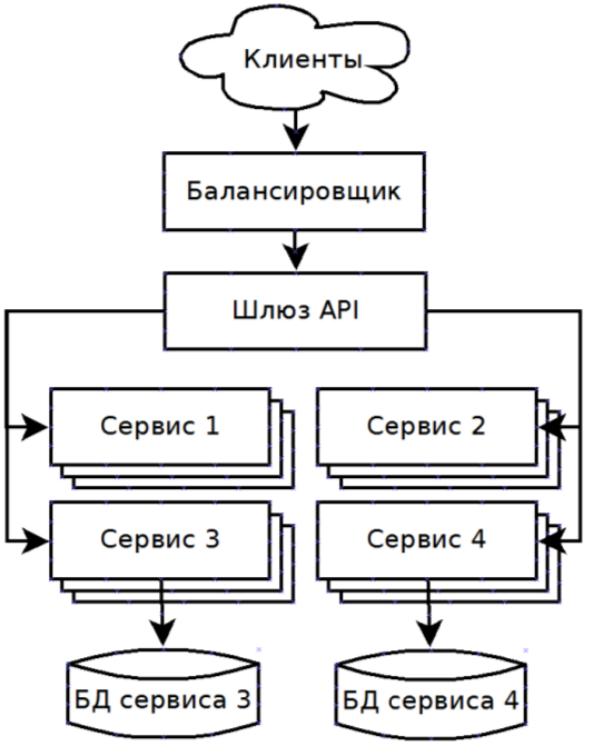
- Архитектура, управляемая событиями – это относительно сложный паттерн. Причиной тому – его **распределенная и асинхронная природа**. Вам придется решать проблемы фрагментации сети, обрабатывать ошибки очереди событий и так далее.
- Плюсами этой архитектуры могут служить **высокая производительность**, **легкость развертки** и **поразительные возможности масштабирования**. Однако возможно усложнение процесса тестирования системы.

Микросервисная архитектура

- **Микросервисная архитектура** – это паттерн проектирования облачных приложений, подразумевающий, что сложное приложение разделяется на ряд небольших независимых сервисов, взаимодействующих друг с другом посредством кроссплатформенного API.
- Каждый микросервис включает в себя бизнес-логику и представляет собой совершенно независимый компонент. Сервисы одной системы могут быть написаны на различных языках программирования и общаться друг с другом, используя различные протоколы.
- *Отличительные особенности микросервисов:*
  - Микросервисы независимы;
  - Микросервисы общаются друг с другой только при помощи сообщений;
  - Каждый микросервис может быть развернут, приостановлен, дублирован или перемещен независимо от других.

Микросервисная архитектура VS Монолитная архитектура





- Достоинства
  - сверхвысокая масштабируемость
  - легкость распределения задач
- Недостатки
  - необходимость передачи большого объема данных
  - необходимость автоматизации развертывания и тестирования

Многоуровневая клиент-серверная архитектура

Уровни клиент-серверной архитектуры

- Сегодня выделяют 3 основных уровня клиент-серверной архитектуры:
  - Уровень представления (пользовательского интерфейса)
  - Уровень бизнес-логики (обработки)
  - Уровень данных

Уровень представления

- Обычно реализуется на клиентах
- Организует методы взаимодействия с приложением
- Простейший вариант:
  - символьный дисплей (терминал) к мейнфрейму

Уровень бизнес-логики

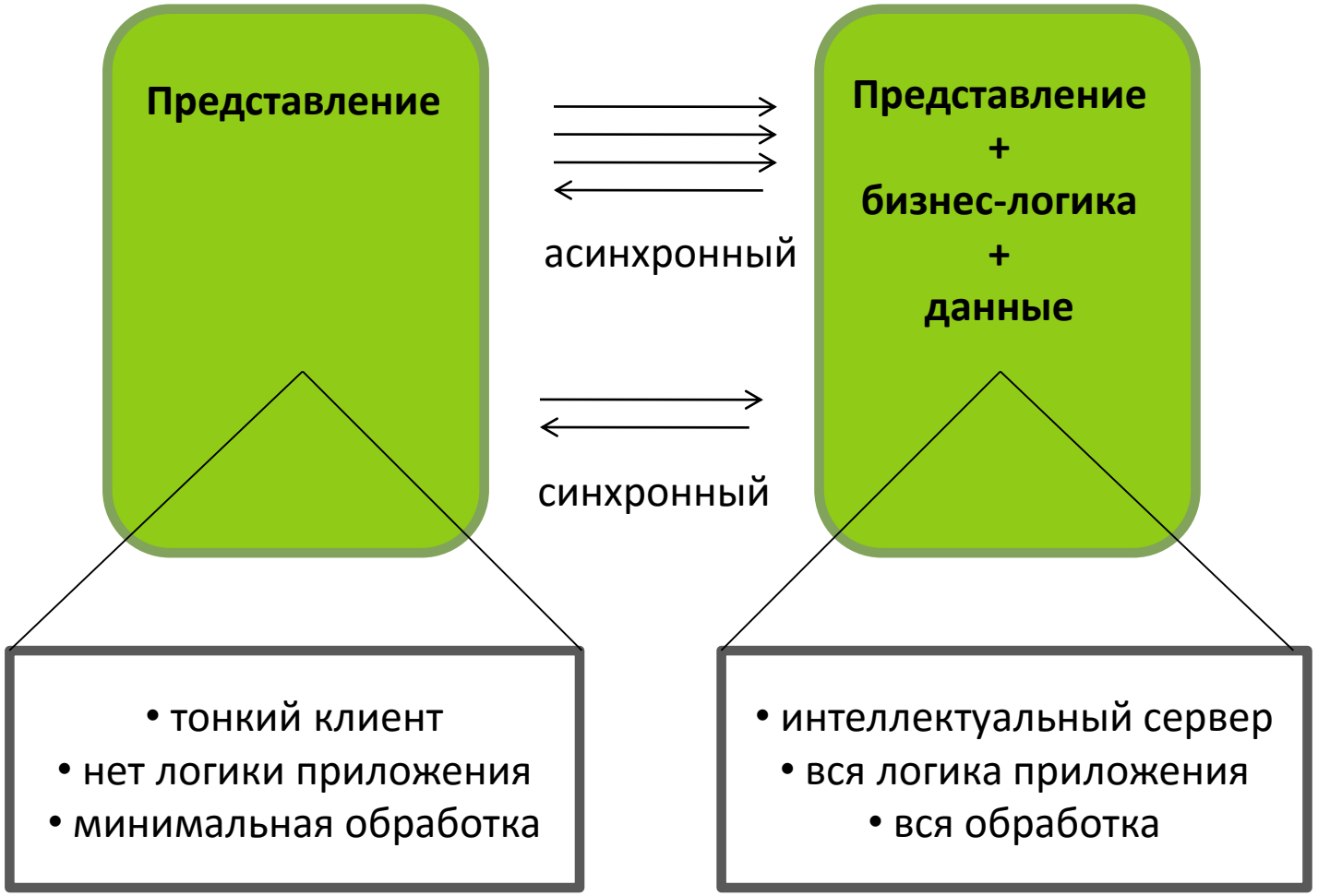
- Бизнес-логика – это совокупность правил, принципов и зависимостей поведения объектов предметной области системы.
- Синоним: логика предметной области (Domain Logic).
- Пример:
  - формула расчета зарплаты + налоги;
  - оценка качества обучения на основе оценок студента;
  - отказ от отеля при отмене рейса авиакомпанией.

Уровень данных

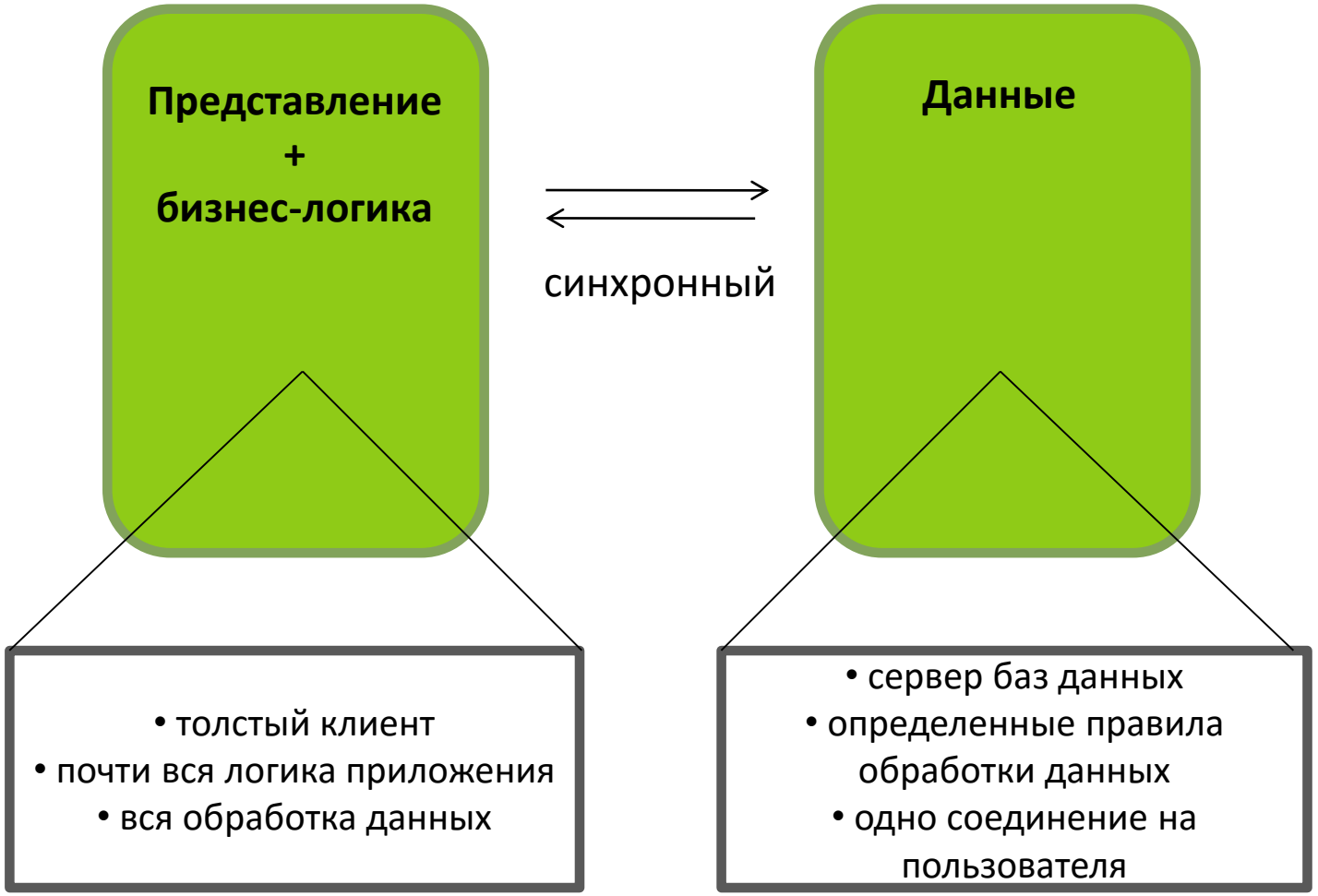
- Программы, которые предоставляют данные обрабатывающим приложениям
- требование СОХРАННОСТИ: когда приложение не работает, данные должны сохранятся в определенном месте;
- требование ЦЕЛОСТНОСТИ: метаданные (описание таблиц, ограничения и т.п.) должны исполнятся и проверяться на этом уровне
- Обычно реализуется реляционной БД

История и типы клиент-серверной архитектуры

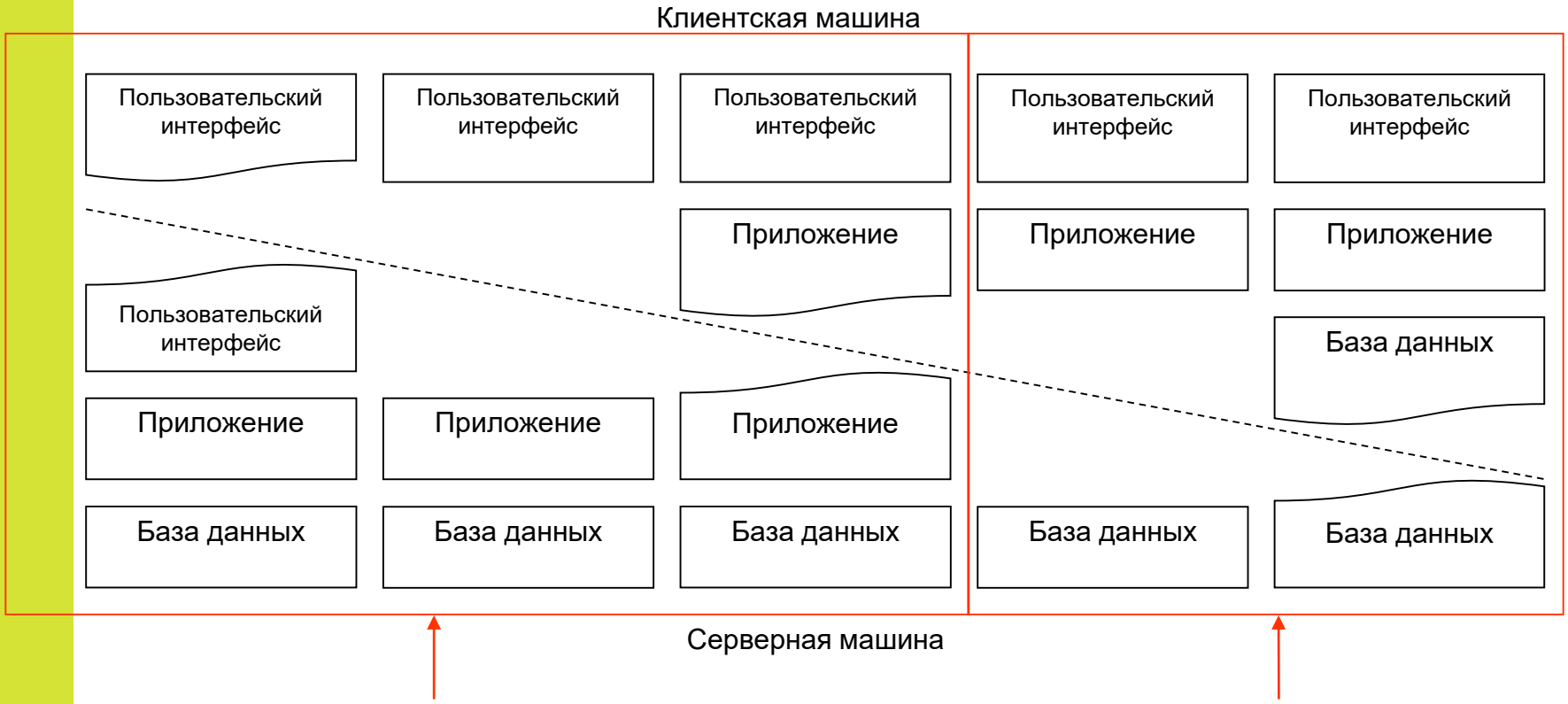
Однозвенная клиент-серверная архиетктура



Двузвенная архитектура



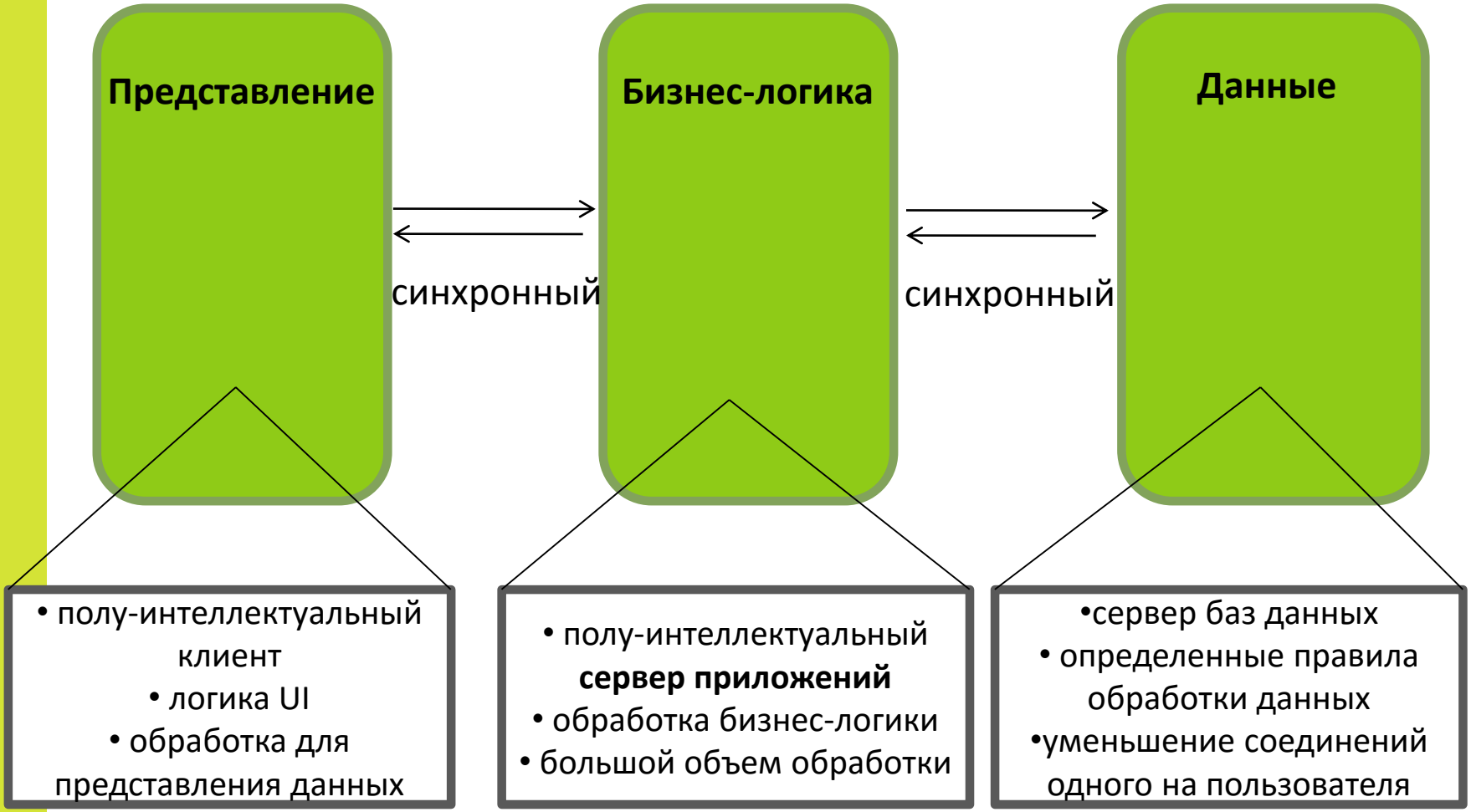
Альтернативные варианты двузвенной архитектуры



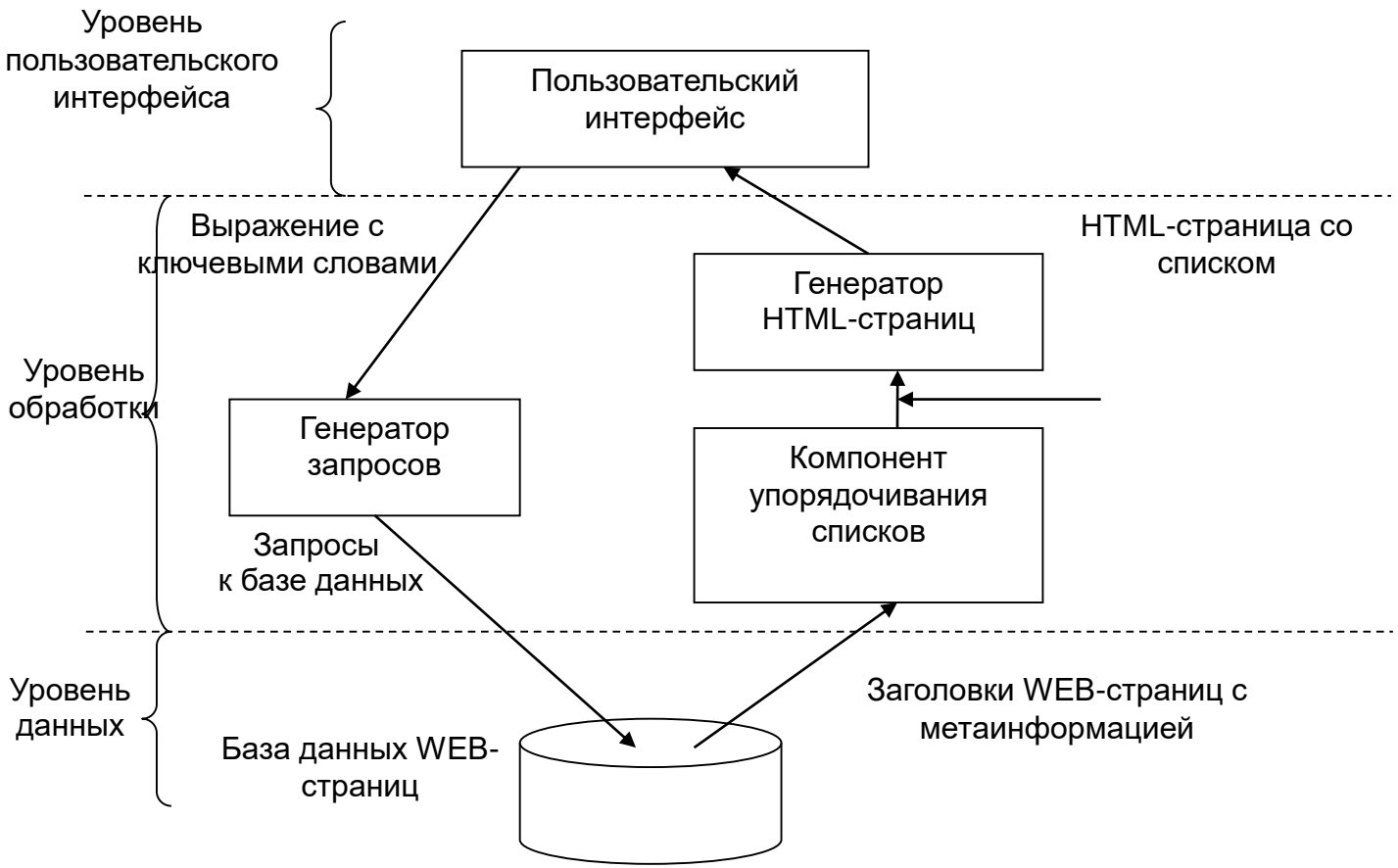
Минусы двузвенной архитектуры

- Чрезвычайные затраты на поддержание рабочих станций, которые должны обрабатывать бизнес-логику
- Чрезвычайная сложность обновления приложения при незначительном изменении бизнес-логики (необходимо переустановить все клиенты)
- Каждая рабочая станция – уникальный набор ПО, который может конфликтовать с клиентом и влиять на его работу

Трехзвенная архитектура



Пример трехзвенной архитектуры – поисковая машина



Современный пример многозвенной архитектуры

- 1. Браузер клиента->
  - 2. Сервер IIS->
    - 3. Исполняющая среда ASP.NET 2.0->
      - 4. Провайдер данных ADO.NET 2.0 ->
        - 5. Сервер MySQL ->
      - 6. Провайдер данных ADO.NET 2.0 ->
    - 7. Исполняющая среда ASP.NET 2.0->
  - 8. Сервер IIS ->
- 9. Браузер клиента