

# РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Модель «Клиент-Сервер»

- ◎ К какому уровню модели OSI относятся следующие протоколы:
  - ◎ TCP
  - ◎ IP
  - ◎ HTTP
  - ◎ Bluetooth
- ◎ Что такое сокет? Как определяется сокет TCP/IP?
- ◎ Как использование менеджеров сообщений позволяет обеспечить масштабируемость и эластичность PBC?
- ◎ Что означает децентрализованная доступность ресурсов?
- ◎ Зачем была придумана технология RPC, если уже существовала технология сокетов?
- ◎ Зачем была придумана технология RMI, если уже существовала технология RMI?

# 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

7

- ◎ 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

8

- ◎ 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.*

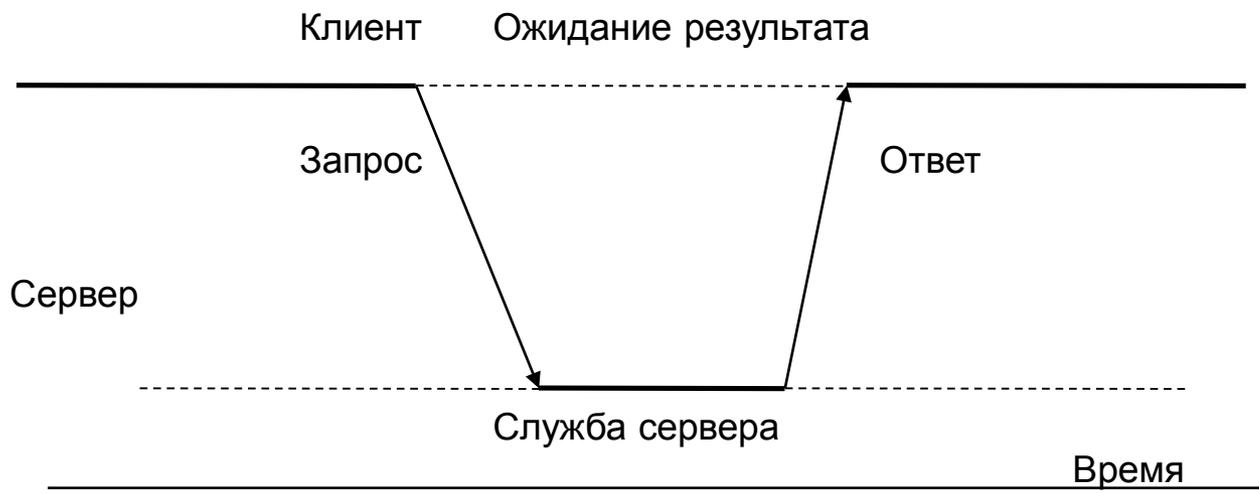
# АЛГОРИТМ РАБОТЫ КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЫ

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

12

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

# УРОВНИ КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЫ

# УРОВНИ КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЫ

14

- ◎ 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

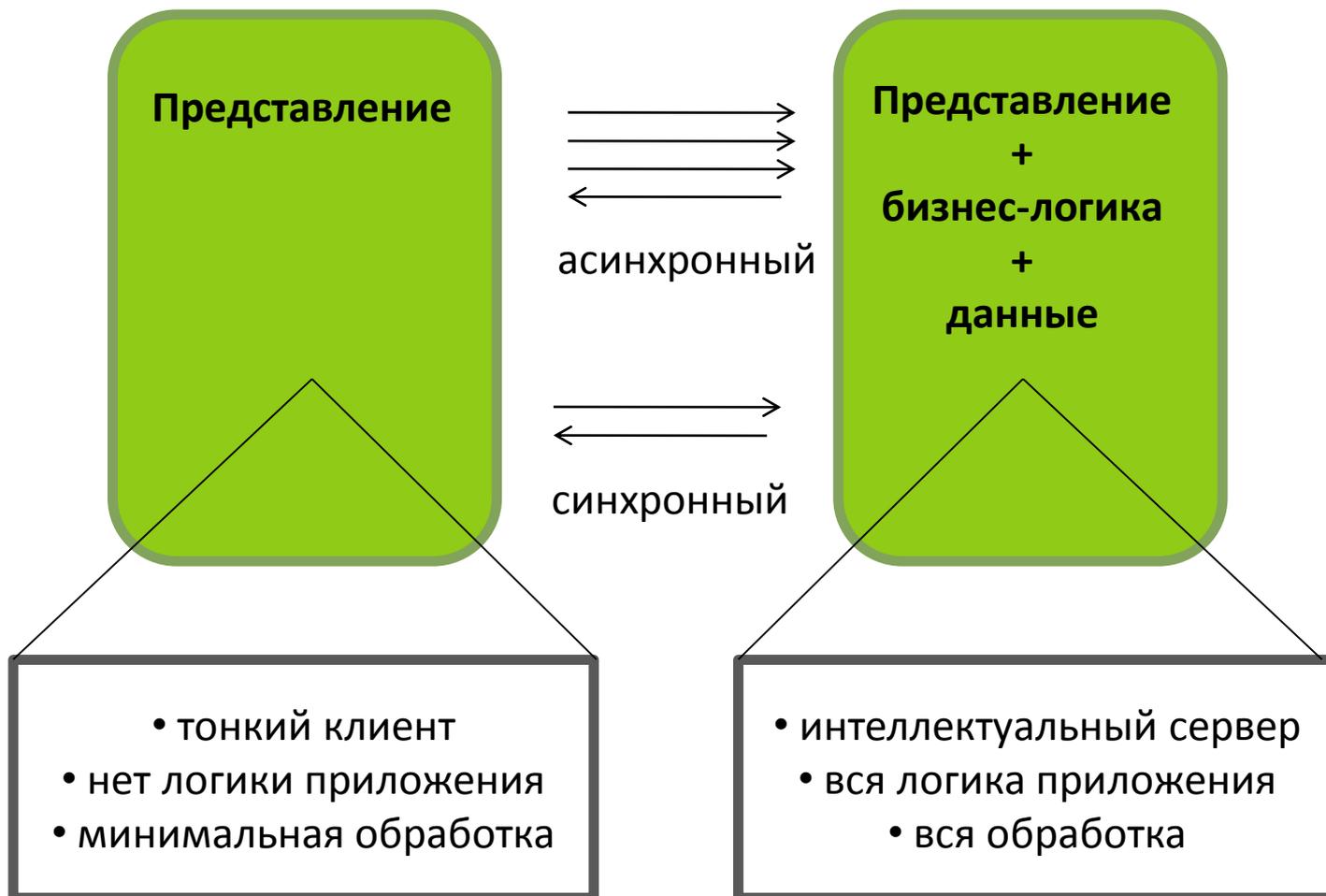
- ◎ Бизнес-логика – это совокупность правил, принципов и зависимостей поведения объектов предметной области системы.
- ◎ Синоним: логика предметной области (Domain Logic).
- ◎ Пример:
  - ◎ формула расчета зарплаты + налоги;
  - ◎ оценка качества обучения на основе оценок студента;
  - ◎ отказ от отеля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

# УРОВЕНЬ ДАННЫХ

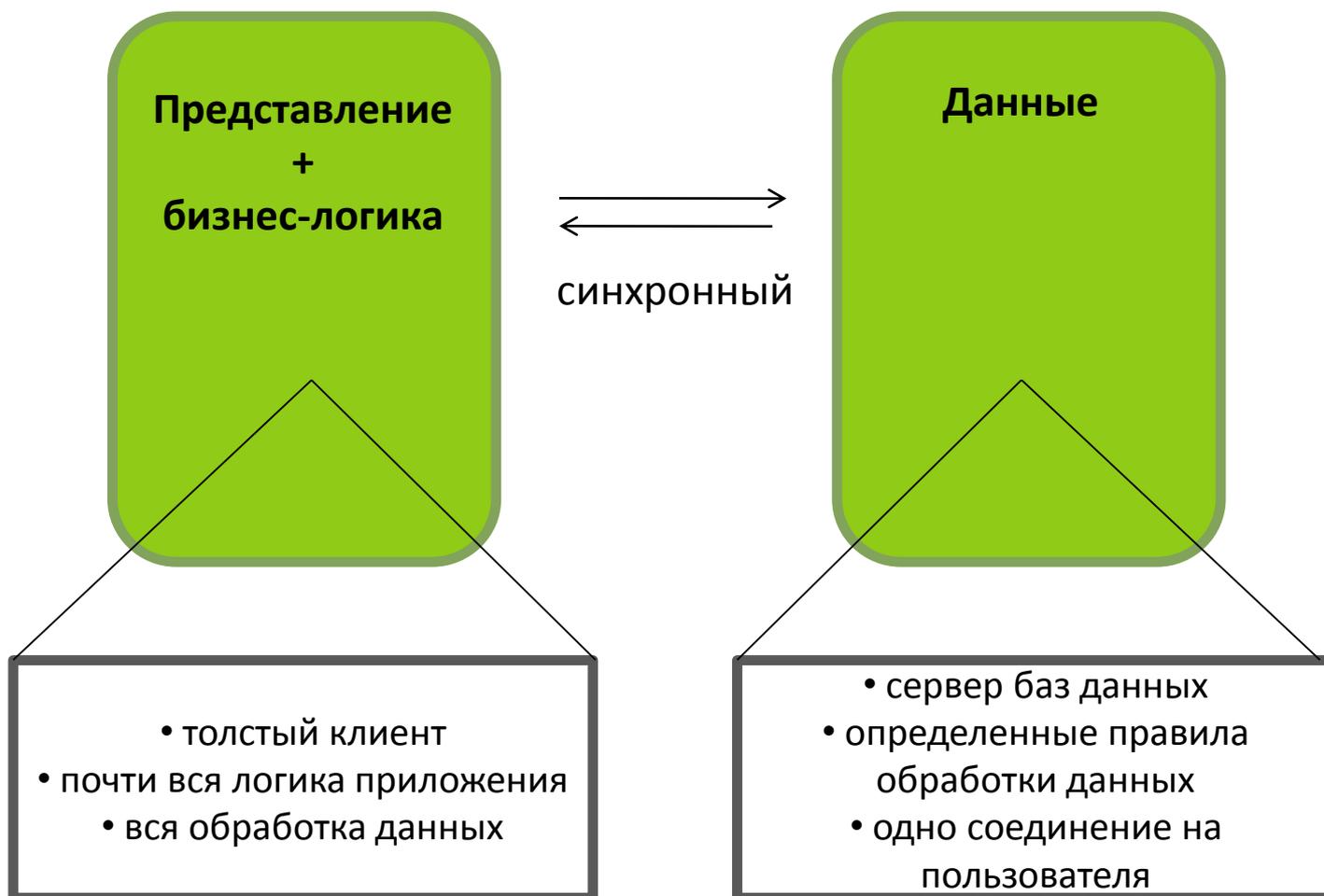
- ⊙ Программы, которые предоставляют данные обрабатывающим приложениям
- ⊙ требование СОХРАННОСТИ: когда приложение не работает, данные должны сохраняться в определенном месте;
- ⊙ требование ЦЕЛОСТНОСТИ: метаданные (описание таблиц, ограничения и т.п.) должны исполняться и проверяться на этом уровне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

# ИСТОРИЯ И ТИПЫ КЛИЕНТ- СЕРВЕРНОЙ АРХИТЕКТУРЫ

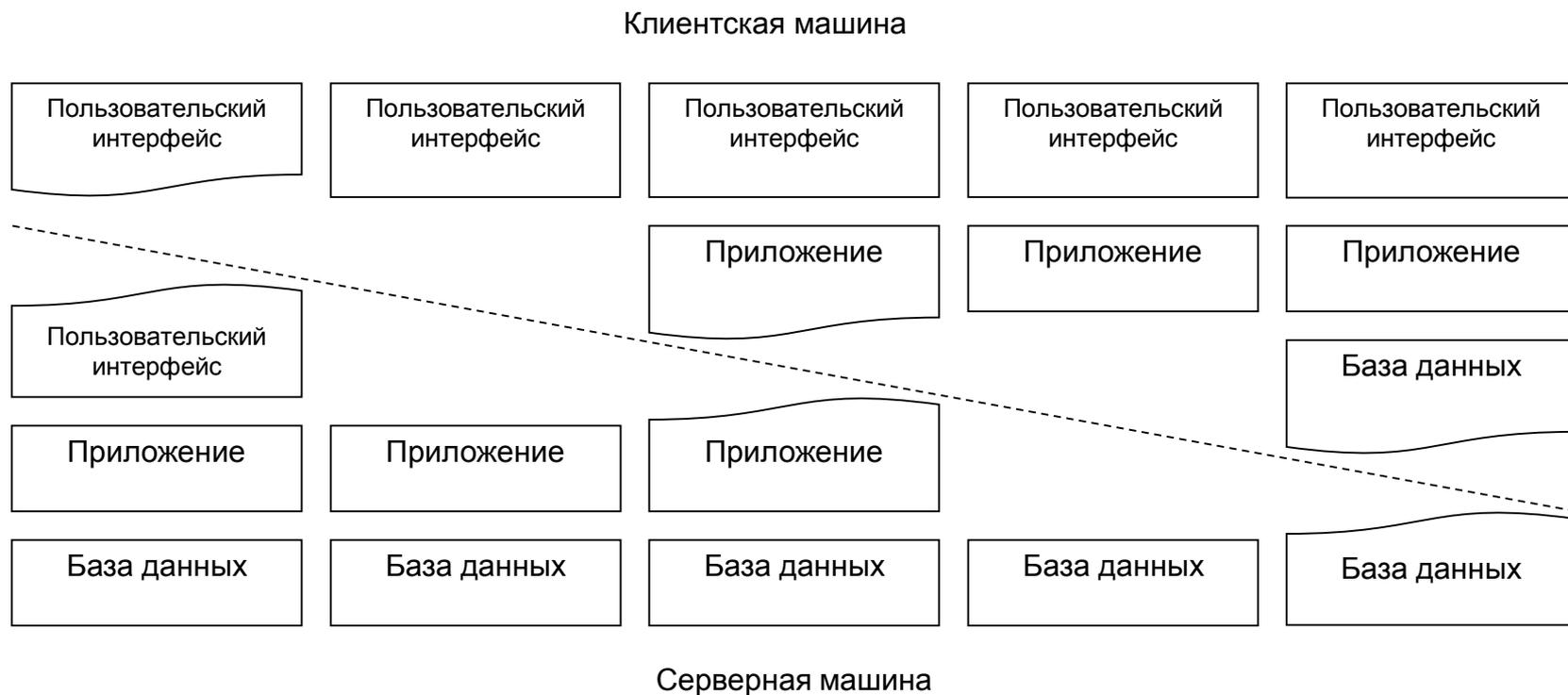
# ОДНОЗВЕННАЯ КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА



# ДВУЗВЕННАЯ АРХИТЕКТУРА



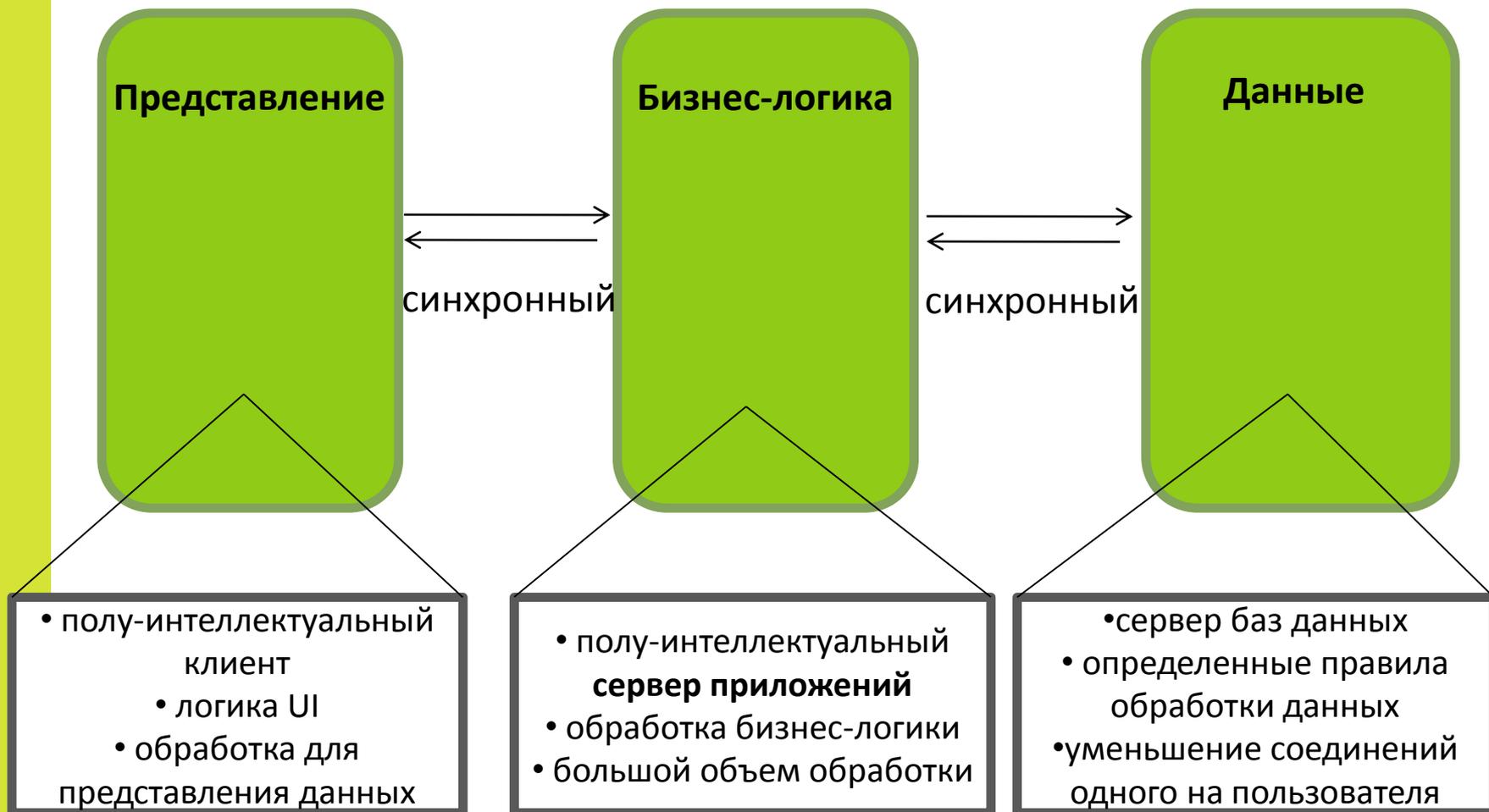
# АЛЬТЕРНАТИВНЫЕ ВАРИАНТЫ ДВУЗВЕННОЙ АРХИТЕКТУРЫ



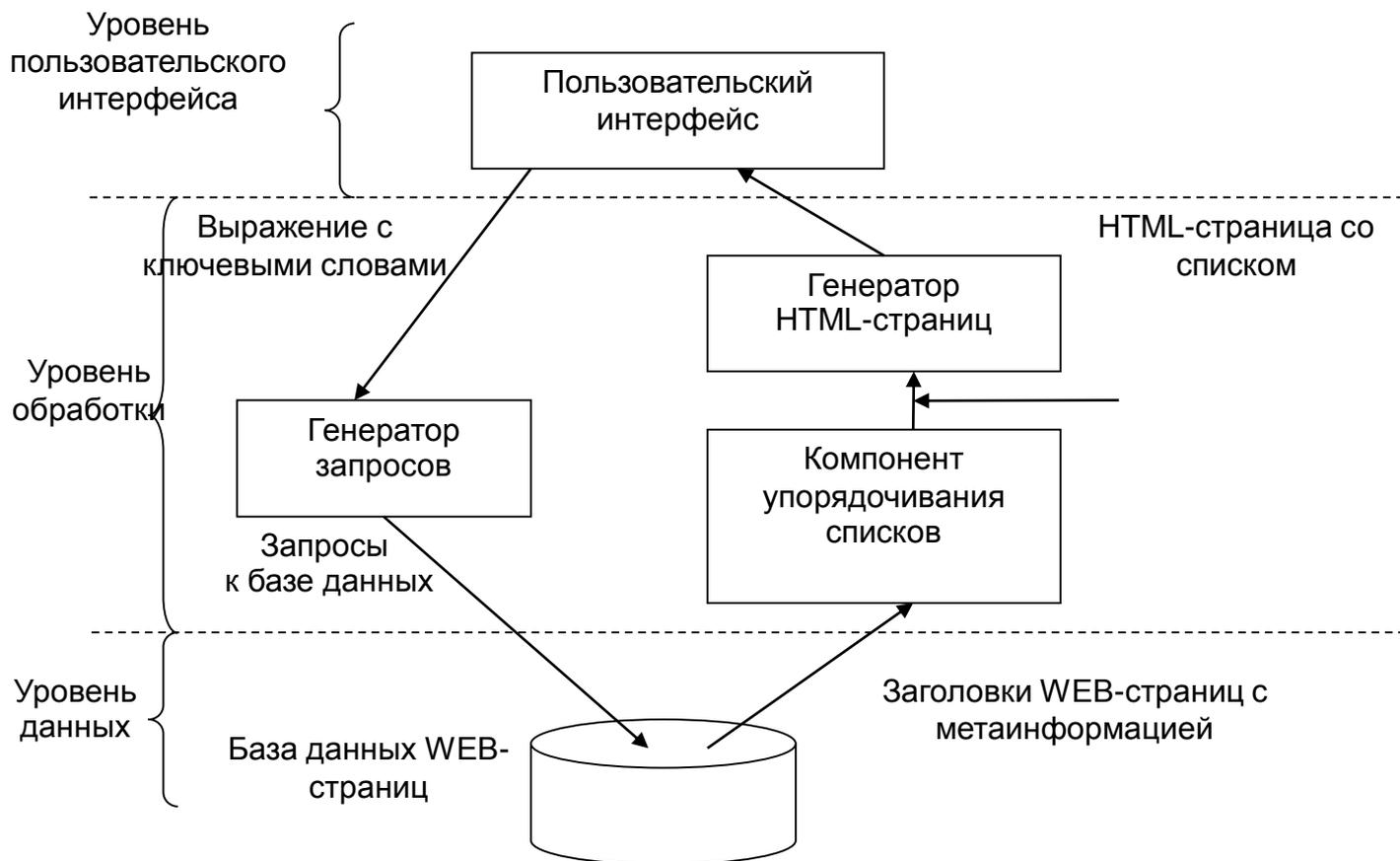
# МИНУСЫ ДВУЗВЕННОЙ АРХИТЕКТУРЫ

- ◎ 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



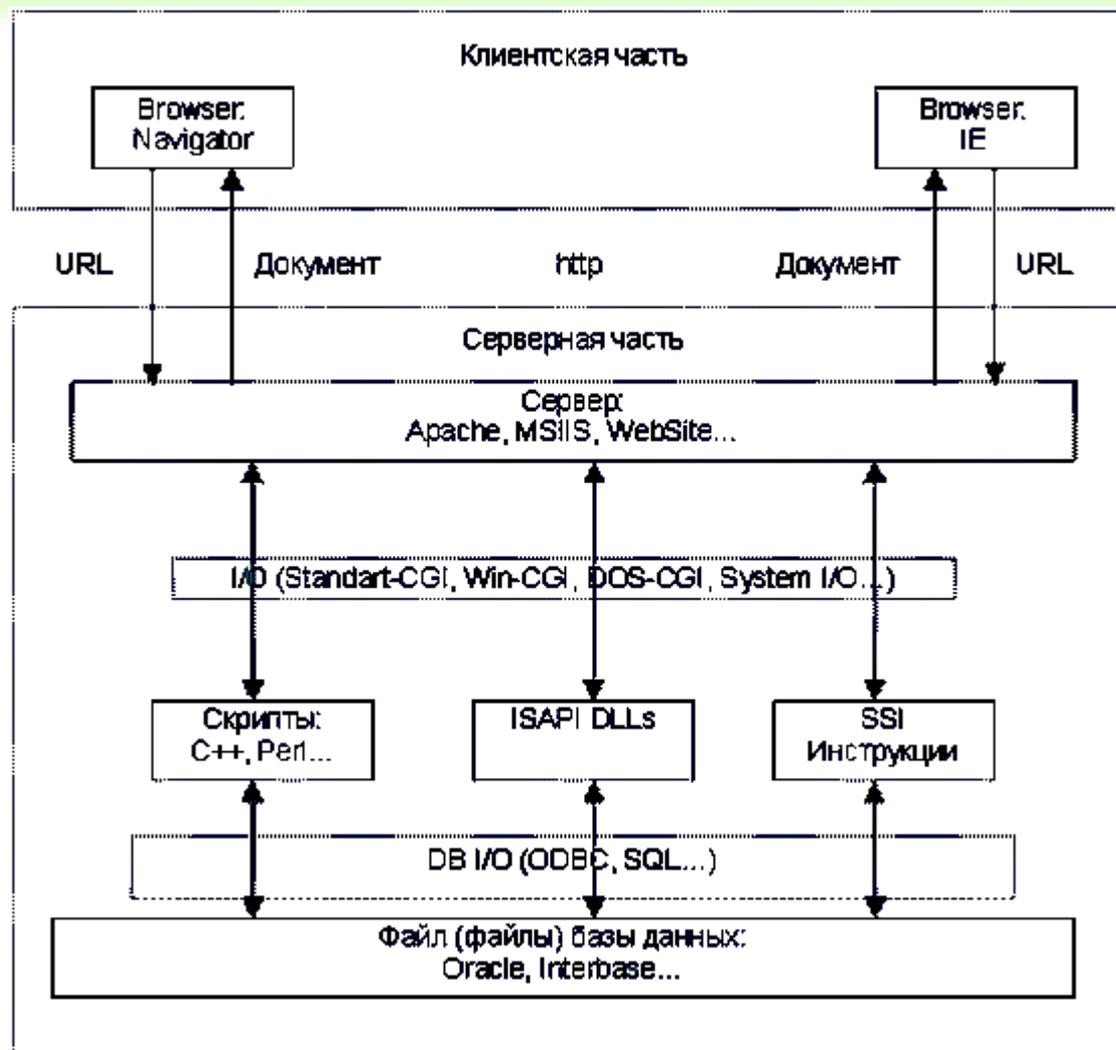
# ПРИМЕР ТРЕХЗВЕННОЙ АРХИТЕКТУРЫ - ПОИСКОВАЯ МАШИНА



# 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

# КЛИЕНТ-СЕРВЕР В НТТР



# ГОРИЗОНТАЛЬНОЕ РАСПРЕДЕЛЕНИЕ

# МЕТОДЫ РАЗБИЕНИЯ НА УРОВНИ

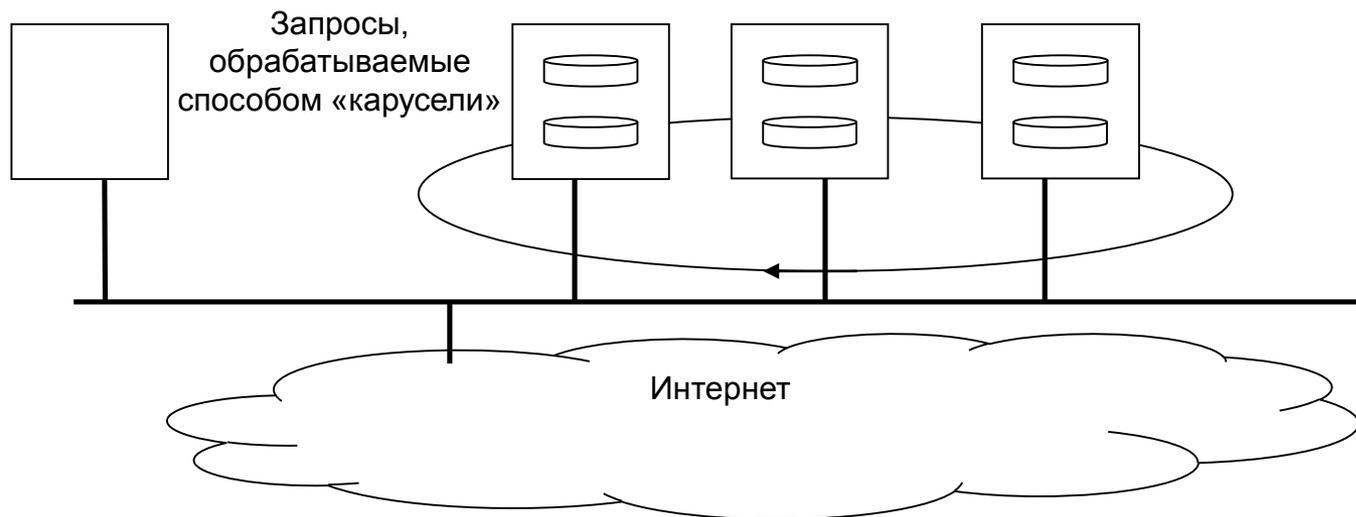
28

- ◎ Вертикальное деление:
  - ◎ UI
  - ◎ бизнес-логика
  - ◎ данные
  
- ◎ Горизонтальное деление:
  - ◎ Клиент или сервер может содержать физически разделенные части логически однородного модуля

# ПРИМЕР ГОРИЗОНТАЛЬНОГО РАСПРЕДЕЛЕНИЯ

Встреча и  
обработка  
входящих запросов

Реплицированные WEB-серверы,  
содержащие одни и те же WEB-  
страницы



- ◎ Клиент-серверная архитектура появилась вместе с появлением первых методов удаленного доступа к вычислительным системам.
- ◎ Выделяют от 1 до 3-х уровней клиент-серверной архитектуры: UI, бизнес-логики, данных.
- ◎ Существуют методы вертикального (по слоям) и горизонтального (в рамках одного слоя) распределения серверов.