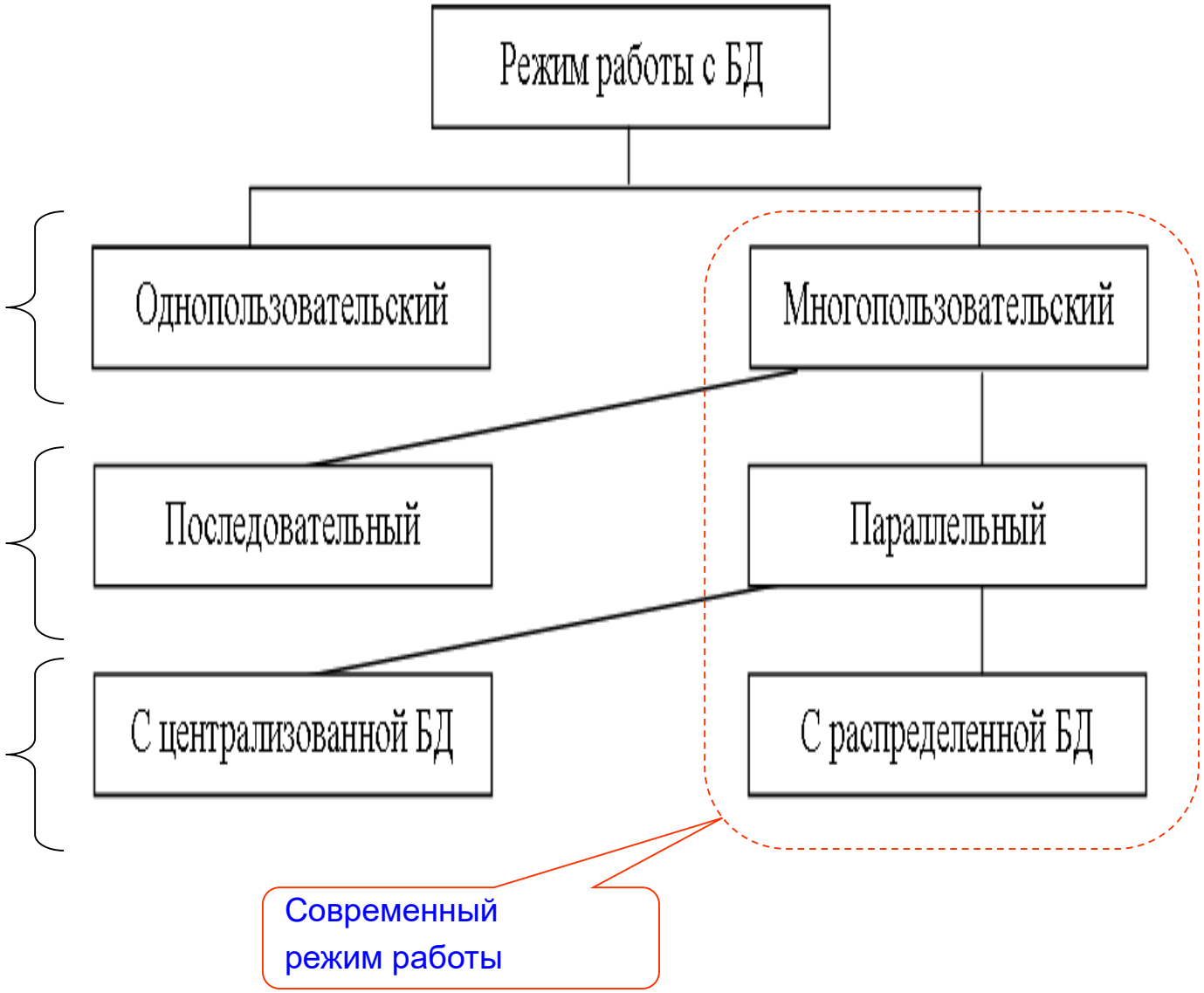
# Разработка проекта распределенной обработки

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ

# Режимы работы с базами данных

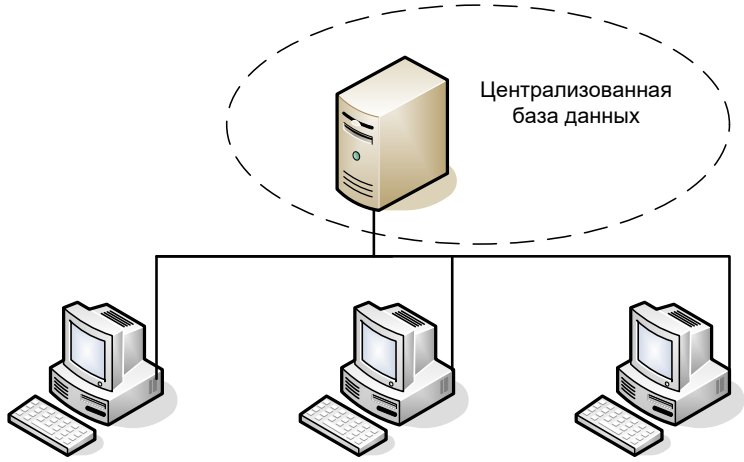
Классификационный признак, отличия:



## Система распределенной обработки данных

Режим распределенного доступа к централизованной БД:

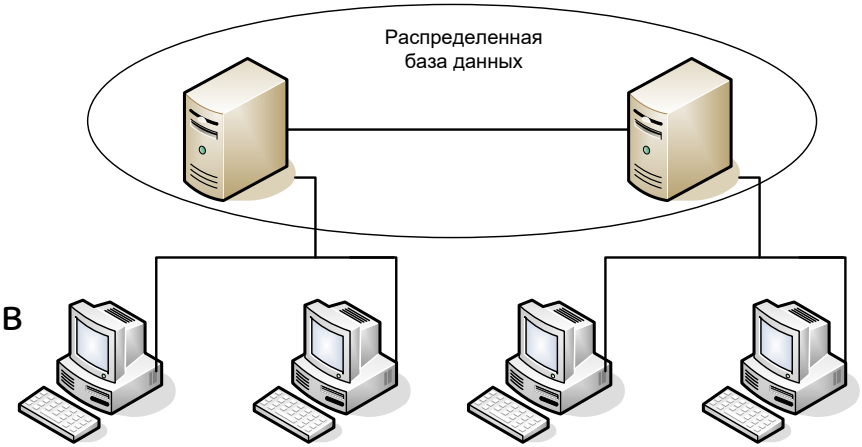
- параллельный доступ к одной базе данных нескольких пользователей;
- база данных физически расположена на одной машине.



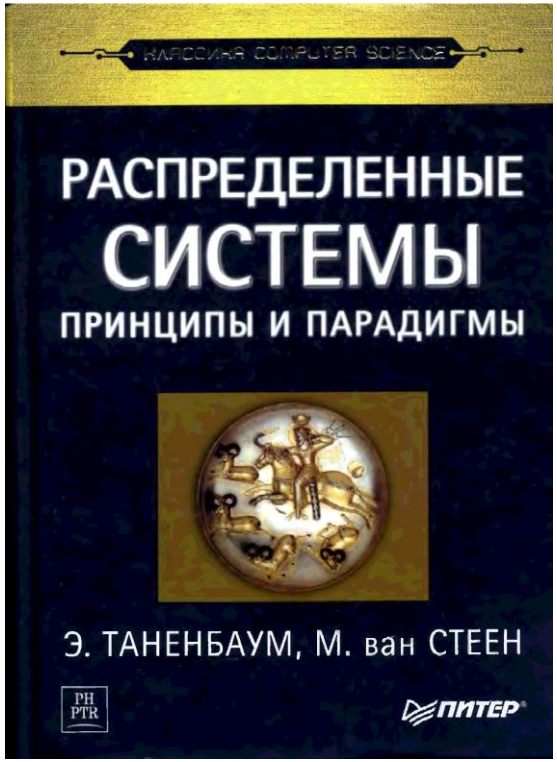

## Система распределенных баз данных

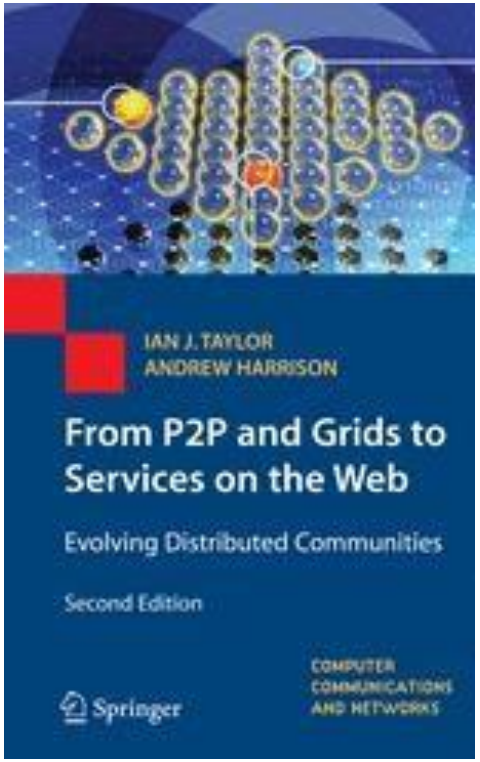
Режим параллельного доступа к распределенной БД:

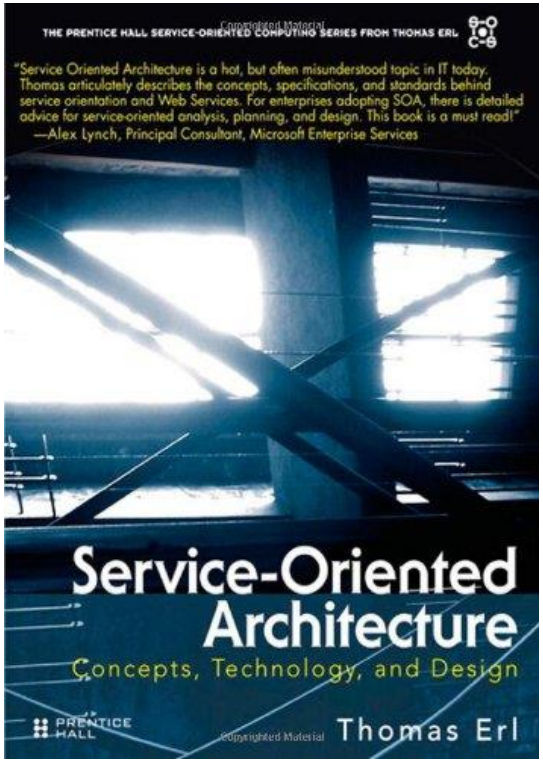
- база данных физически распределена по нескольким компьютерам, расположенным в сети;
- к базе данных возможен параллельный доступ нескольких пользователей.



# Литература


КЛАССИКА COMPUTER SCIENCE
РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ
ПРИНЦИПЫ И ПАРАДИГМЫ
Э. ТАНЕНБАУМ, М. ван СТЕЕН
PH PTR
ПИТЕР


IAN J. TAYLOR
ANDREW HARRISON
From P2P and Grids to Services on the Web
Evolving Distributed Communities
Second Edition
COMPUTER COMMUNICATIONS AND NETWORKS
Springer


THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL
"Service Oriented Architecture is a hot, but often misunderstood topic in IT today. Thomas articulately describes the concepts, specifications, and standards behind service orientation and Web Services. For enterprises adopting SOA, there is detailed advice for service-oriented analysis, planning, and design. This book is a must read!"
—Alex Lynch, Principal Consultant, Microsoft Enterprise Services
Service-Oriented Architecture
Concepts, Technology, and Design
PRENTICE HALL
Thomas Erl

# Распределенная система

***«Распределенной вычислительной системой** называется такая система, в которой отказ компьютера, о существовании которого вы даже не подозревали, может сделать ваш собственный компьютер непригодным к использованию».*

Лесли Лампорт (Microsoft Corporation)

***«Распределенная вычислительная система (РВС)** – это набор соединенных каналами связи независимых компьютеров, которые с точки зрения пользователя некоторого программного обеспечения выглядят единым целым».*

Э. Таненбаум

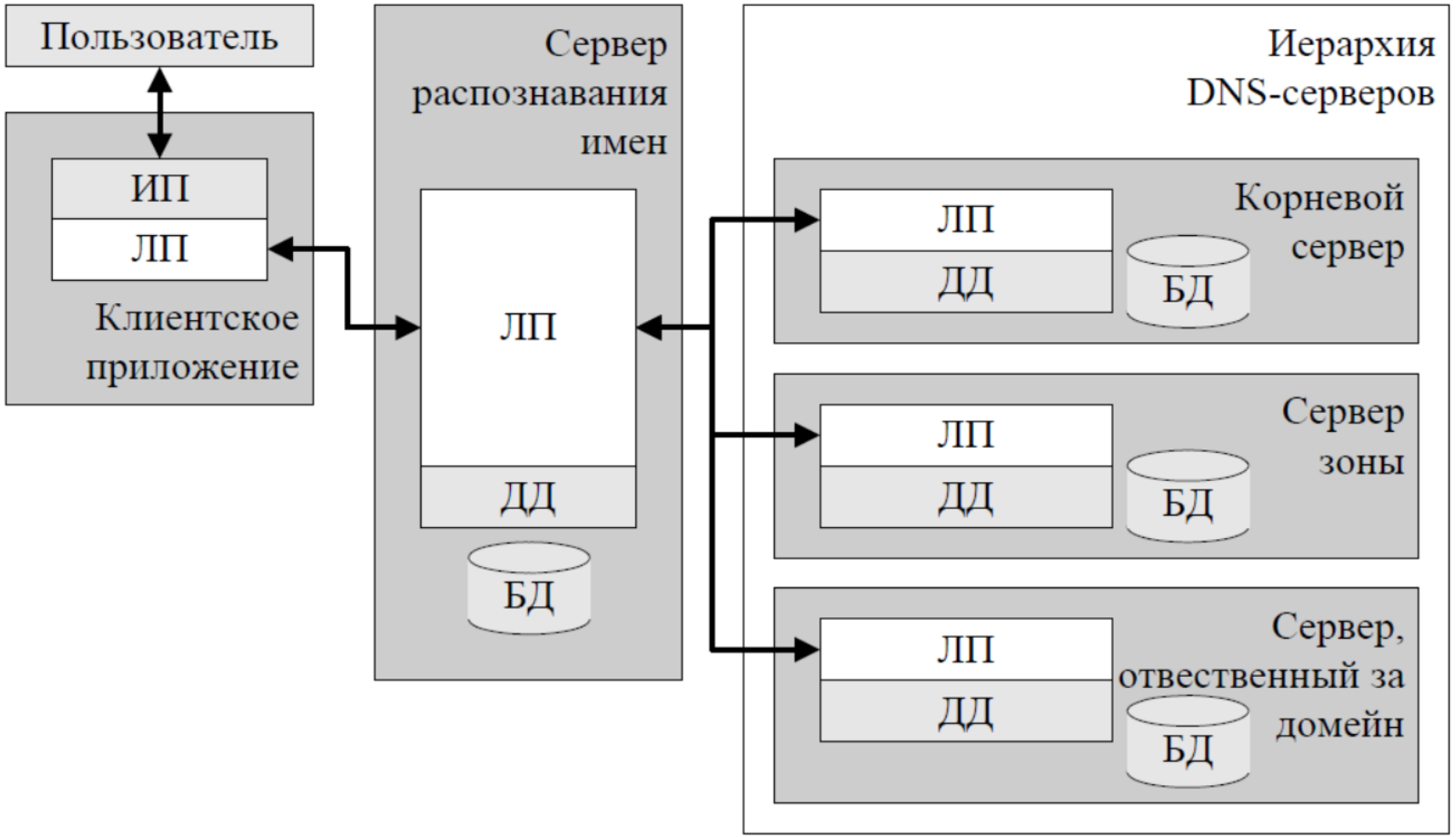
# Распределенная система

Распределенная система – это такая система, в которой взаимодействие и синхронизация программных компонентов, выполняемых на независимых сетевых компьютерах, осуществляется посредством передачи сообщений.

Распределенная система – набор независимых компьютеров, не имеющих общей совместно используемой памяти и общего единого времени (таймера) и взаимодействующих через коммуникационную сеть посредством передачи сообщений, где каждый компьютер использует свою собственную оперативную память и на котором выполняется отдельный экземпляр своей операционной системы. Однако эти операционные системы функционируют совместно, предоставляя свои службы друг другу для решения общей задачи.

Термин "распределенная система" описывает широкий спектр систем от слабо связанных многомашинных комплексов, представляемых, например, набором персональных компьютеров, объединенных в сеть, до сильно связанных многопроцессорных систем.

# Пример

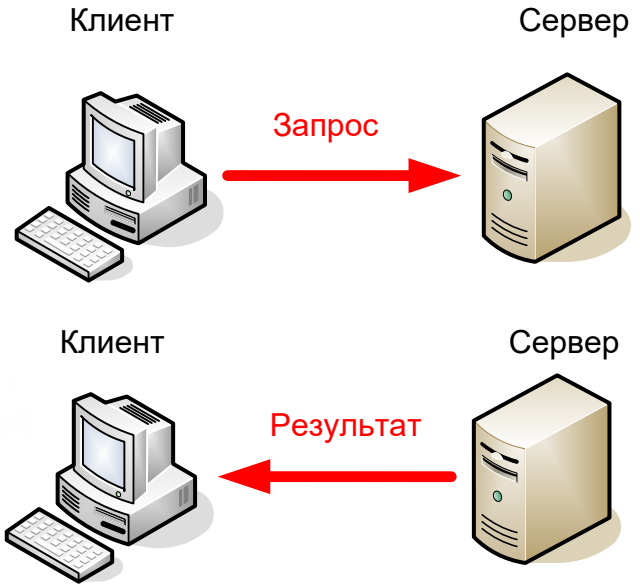

Пользователь
ИП
ЛП
Клиентское приложение
Сервер распознавания имен
ЛП
ДД
БД
Иерархия DNS-серверов
ЛП
ДД
БД
Корневой сервер
ЛП
ДД
БД
Сервер зоны
ЛП
ДД
БД
Сервер, ответственный за домейн

# Модель взаимодействия клиент-сервер

## Основные программные процессы:

- **«Клиент»**
  сторона, запрашивающая функции/обслуживание

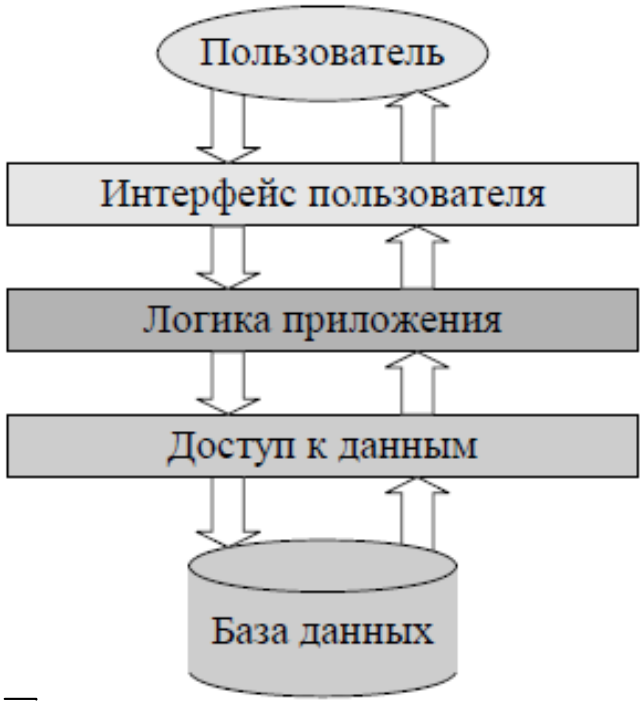
- **«Сервер»**
  сторона, предоставляющая функции/обслуживание

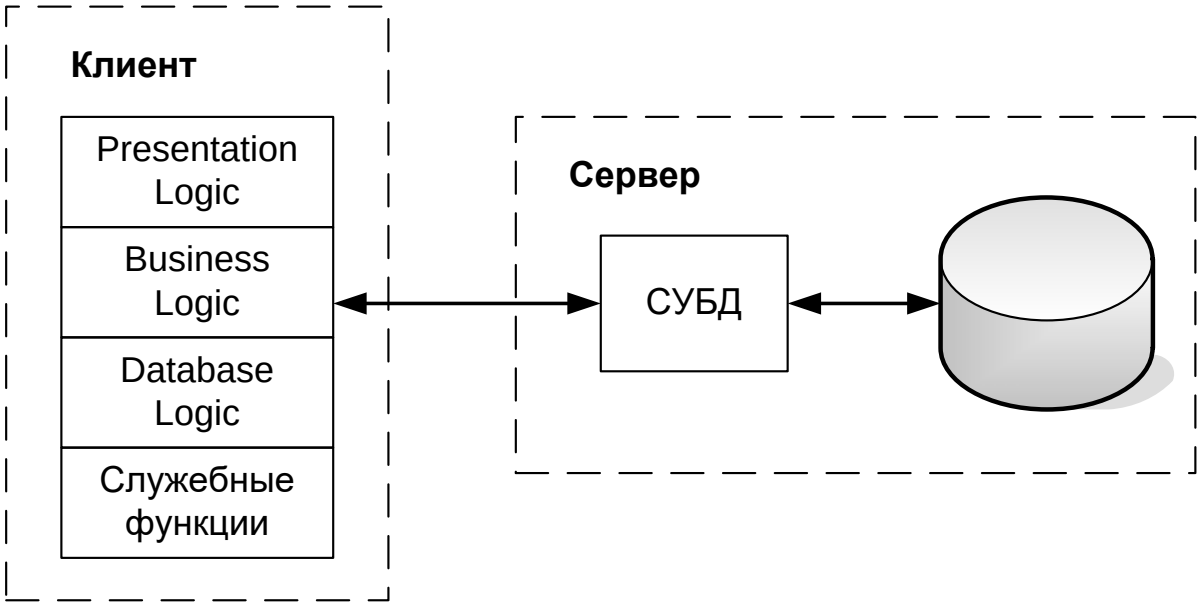



**Особенность**

На уровне программного обеспечения разделение системы на клиента и сервер является **логическим**, а именно процессы клиента и сервера могут **физически** размещаться как на одной, так и на разных машинах.

# Логические уровни приложения



# Типовые функциональные части приложений

1. Средства представления данных на экране (графический пользовательский интерфейс).
2. Логика представления данных на экране – описание правил и возможностей сценариев взаимодействия пользователя с приложением.
3. Прикладная логика – набор правил для принятия решения, вычислительные процедуры и операции.
4. Логика данных – операции с данными, хранящимися в базе данных, которые нужно выполнить для реализации прикладной логики.
5. Внутренние операции базы данных – действия СУБД в ответ на запросы логики данных (поиск записей по определенным признакам).
6. Файловые операции – стандартные операции над файлами и файловой системой.

# Интерфейс пользователя - Презентационная логика (Presentation Logic)

Это часть кода приложения, которая определяет ***что*** пользователь видит на своем экране, когда работает приложение.

**Основные задачи:**

- формирование экранных изображений;
- чтение и запись в экранные формы информации;
- управление экраном;
- обработка движений мыши и нажатие клавиш клавиатуры.

# Бизнес-логика, или логика собственно приложений (Business Processing Logic)

Это часть кода приложения, которая определяет алгоритмы решения конкретных задач приложения.

В зависимости от конкретных функциональных требований и сложности задач может оказаться полезным подразделить эту часть на несколько компонентов (бизнес-правила, правила целостности и др.).

## Реализация

Конкретная реализация каждого компонента может быть представлена в виде набора процедур (библиотек), класса или классов объектов, отдельных программ.

Как правило, код компонентов пишут с использованием различных языков программирования, таких как C, C++, C#, Pascal, Visual Basic и др.

# Логика обработки данных, или логика доступа к данным (Database Logic)

Это часть кода приложения, которая связана с обработкой данных внутри приложения.

## Назначение

Осуществляет перевод специфических для конкретного приложения запросов на язык SQL (интерфейс к СУБД), получение результатов и перевод этих результатов обратно в специфические для конкретного приложения структуры данных.

## Процессор управления данными (Database Manager System Processing)

Это собственно СУБД, которая обеспечивает хранение и управление базами данных.

В идеале функции СУБД должны быть скрыты от бизнес-логики приложения, однако для рассмотрения архитектуры приложения их выделяют в отдельную часть приложения.

# Варианты архитектуры приложения

**Централизованная**

Все части приложения располагаются в единой среде и комбинируются внутри одной исполняемой программы.

**Децентрализованная**

Все задачи могут быть по-разному распределены между серверным и клиентским процессами.

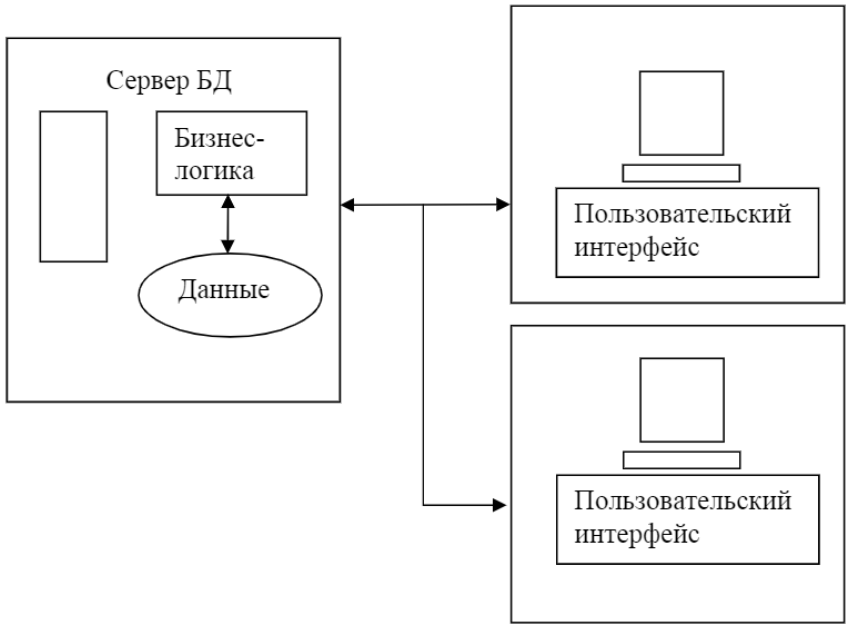
**Различие архитектурных реализаций** определяется тем:

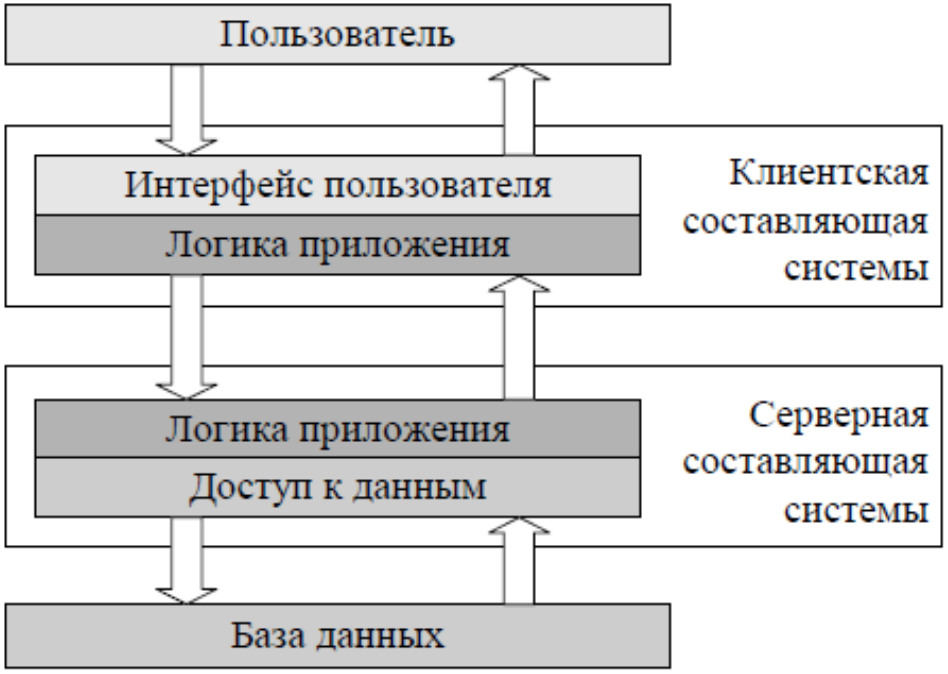
- как логические компоненты базы данных распределены в сети;
- какие механизмы используются для связи компонентов между собой.

**Варианты (модели) построения архитектуры приложения:**

- **двухуровневые**
  - модель файлового сервера (модель удаленного управления данными);
  - модель удаленного доступа к данным;
  - модель сервера баз данных;
- **трехуровневая**
  - модель сервера приложений.

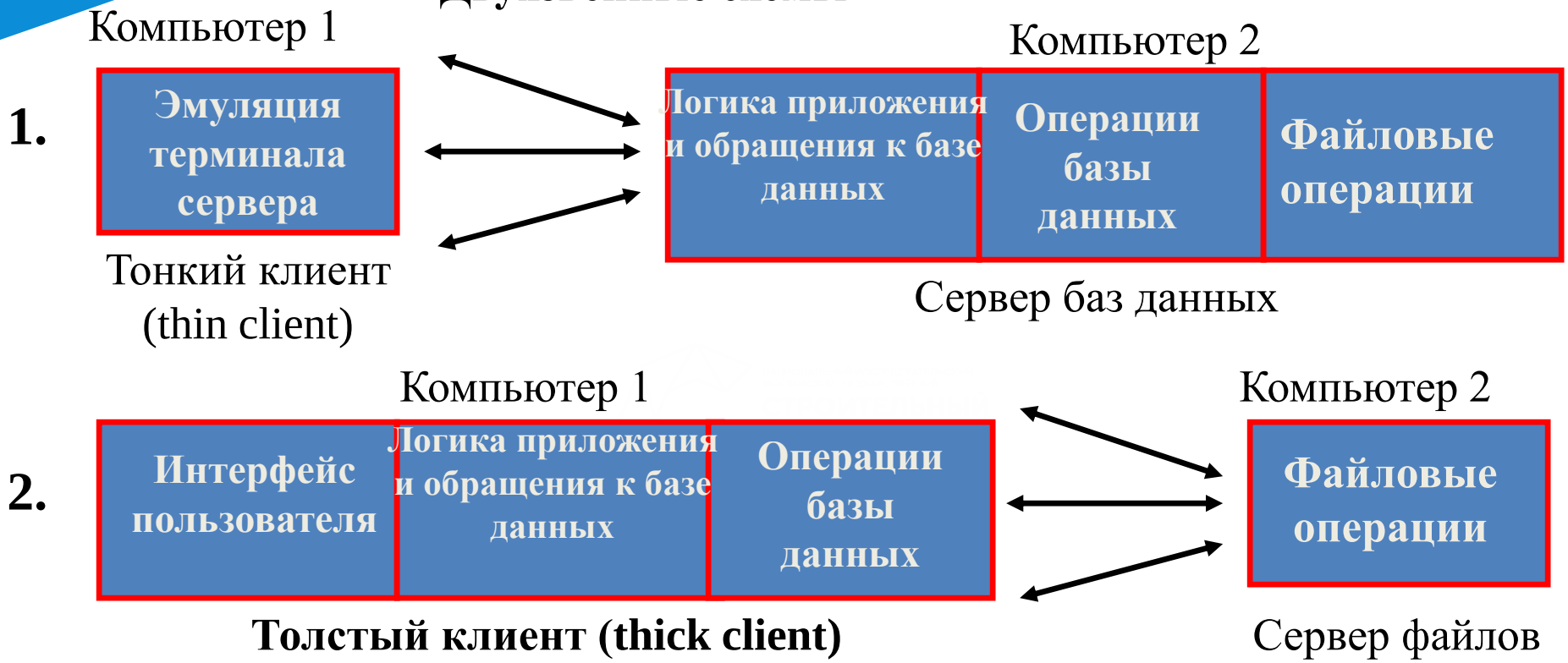
# Двухзвенная архитектура



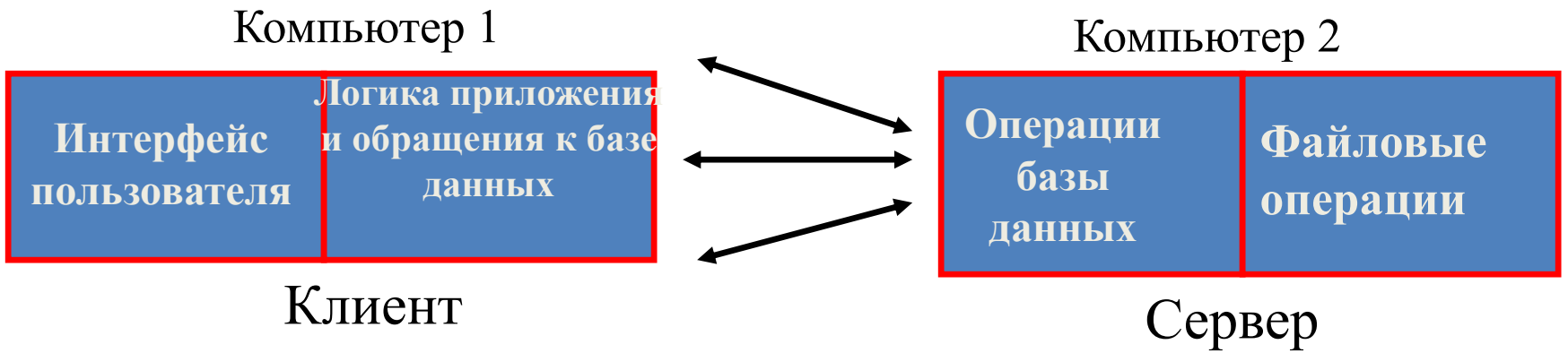
# Модели распределенных приложений

## Двухзвенные схемы



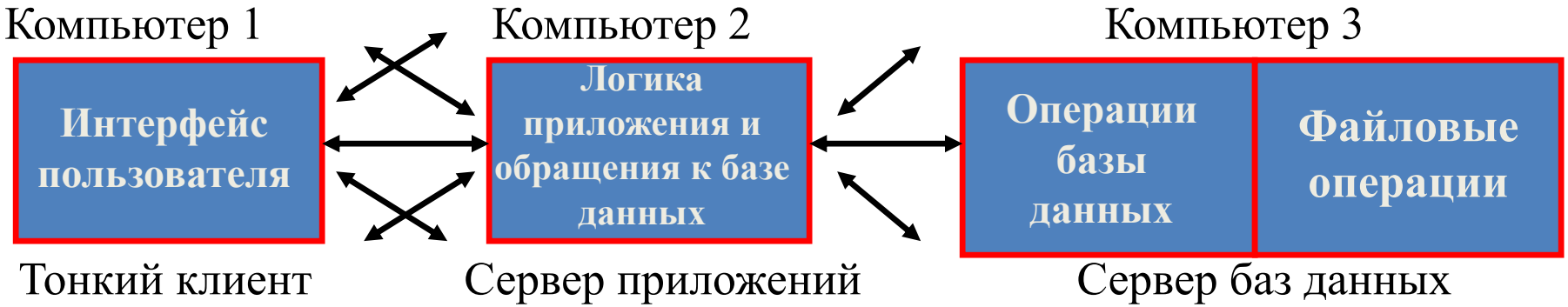

**1. Недостаточная масштабируемость и отсутствие отказоустойчивости, ограничение количества клиентов, простота обновления приложений.**
**2. Хорошая масштабируемость, рост сетевой нагрузки, необходимость обновления приложений на всех клиентских компьютерах.**

3.




3. Оптимальное использование сильных сторон сервера и клиента

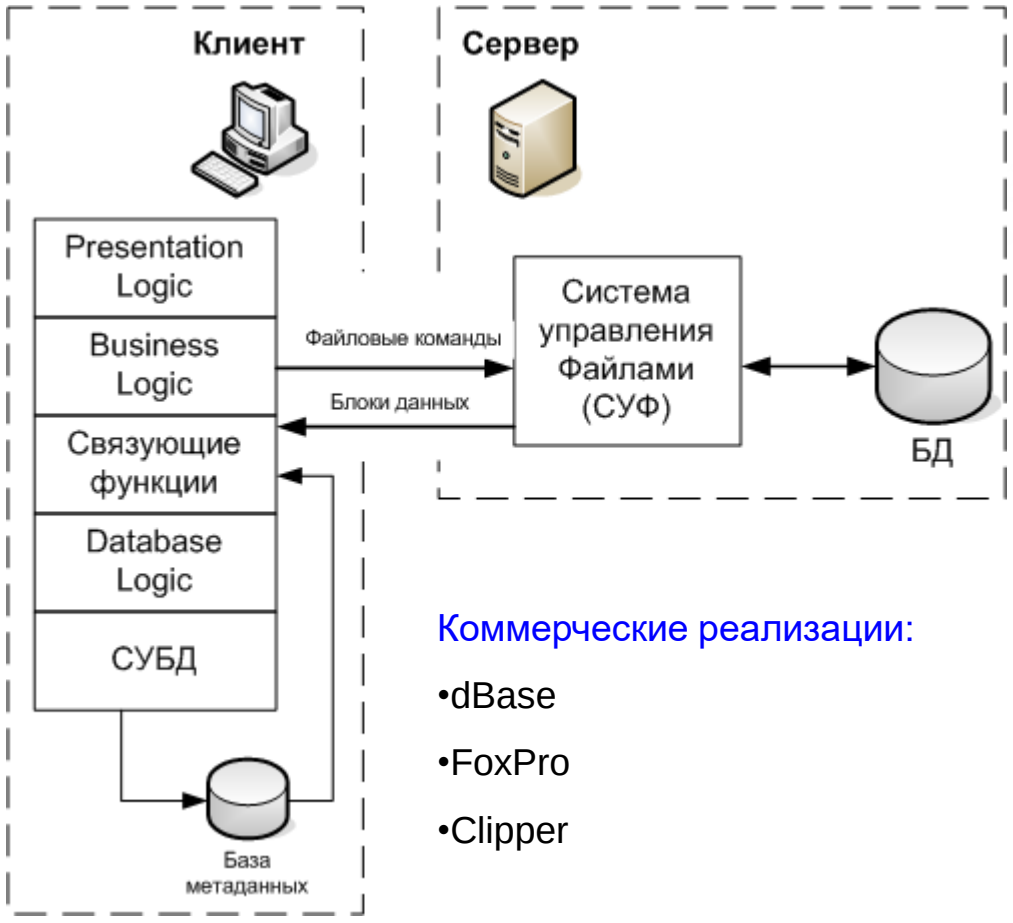
## Трехзвенные схемы




**Трехзвенная схема применяется для централизованной реализации в сети общих для распределенных приложений функций, отличных от файлового сервиса и управления базами данных. Программные модули, выполняющие эти функции,относятся к к классу middleware (промежуточному слою). Цель – позволить приложению (клиенту) получить доступ к различным серверным сервисам, не беспокоясь о различиях между серверами.**

# Двухуровневые модели

## Модель файлового сервера (File Server, FS) (модель удаленного управления данными)




Коммерческие реализации:

- dBase
- FoxPro
- Clipper

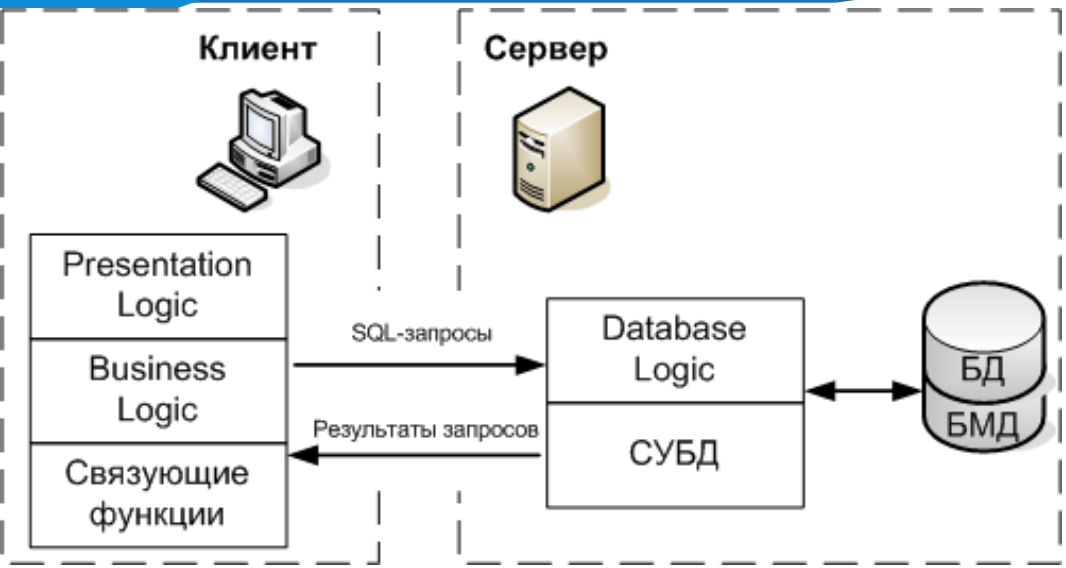
## Преимущества

- разделение монопольного приложения на два взаимодействующих процесса;
- сервер (серверный процесс) может обслуживать множество клиентов

## Недостатки

- высокий сетевой трафик, который связан с передачей по сети множества блоков и файлов, необходимых приложению;
- узкий спектр операций манипулирования с данными, который определяется только файловыми командами. Драйвер «умеет» обрабатывать только простые запросы;
- отсутствуют механизмы оптимизаций запросов и кэширования;
- отсутствие адекватных средств безопасности доступа к данным (защита только на уровне файловой системы).

# Двухуровневые модели




## Модель удаленного доступа к данным (Remote Data Access, RDA)

Коммерческие реализации:

Microsoft Access

## Преимущества

- резко уменьшается загрузка сети, так как по ней от клиентов к серверу передаются не запросы на ввод-вывод в файловой терминологии, а запросы на SQL, и их объем существенно меньше. В ответ на запросы клиент получает только данные, релевантные (соответствующие смыслу) запросу, а не блоки файлов, как в FS-модели,
- более гибкое распределение доступа к данным (на уровне отдельных записей);
- унификация (стандартизация) интерфейса «клиент-сервер», стандартом при общении приложения-клиента и сервера становится язык SQL.

## Недостатки

- запросы на языке SQL при интенсивной работе клиентских приложений могут существенно загрузить сеть;
- сложность администрирования при изменении бизнес-правил, вызванная излишним дублированием кода приложений;
- сервер играет пассивную роль, поэтому функции управления информационными ресурсами должен выполнять клиент.

# Двухуровневые модели

## Модель сервера баз данных (Server DataBase)

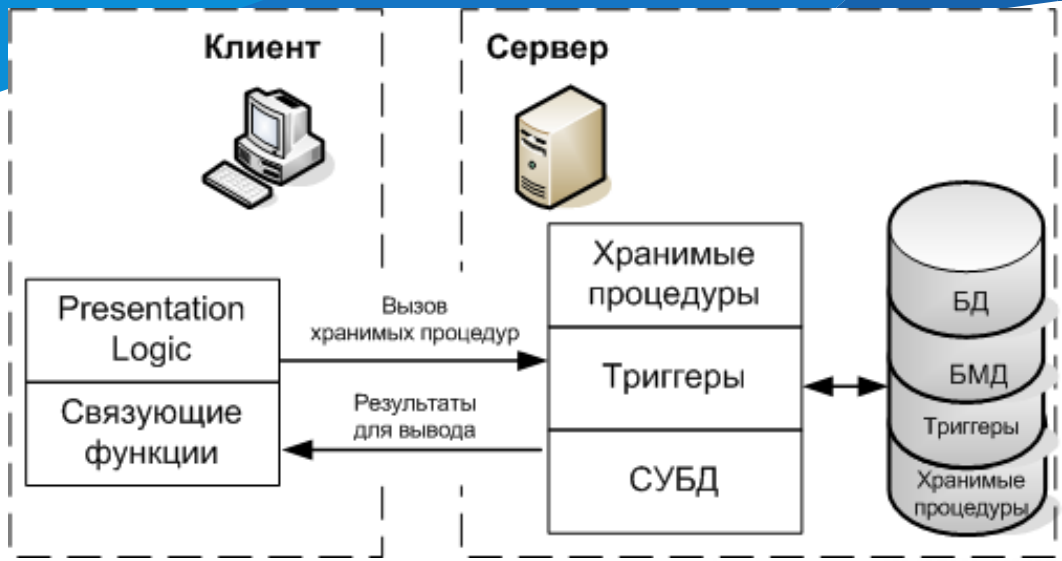
### Требования к серверу

- База данных должна в каждый момент отражать текущее состояние предметной области. Непротиворечивость хранимых в базе данных
- База данных должна отражать некоторые общие правила предметной области (бизнес-правила)
- Необходим постоянный контроль за состоянием базы данных, отслеживание всех изменений и адекватная реакция на них
- Необходимо, чтобы возникновение некоторой ситуации в базе данных четко и оперативно влияло на ход выполнения прикладной задачи.
- Необходимо контролировать в базе данных семантическую составляющую бизнес-правил.

**Механизмы на сервере для реализации требований**

- Хранимые процедуры (Stored Procedure)
- Триггеры (Trigger)

# Двухуровневые модели




## Модель сервера баз данных (Server DataBase)

Коммерческие реализации:

- Oracle
- Microsoft SQL Server

## Преимущества

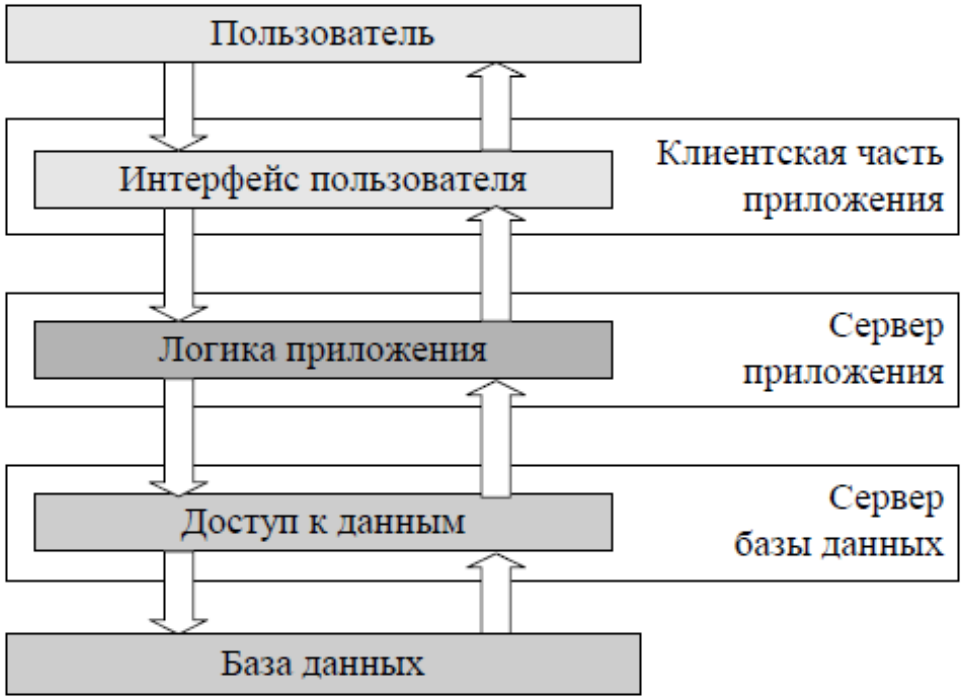
- снижение сетевого трафика;
- реализация общей для клиентов бизнес-логики средствами сервера, что существенно уменьшает дублирование алгоритмов обработки данных в разных клиентских приложениях;
- упрощение администрирования сервера БД (наличие встроенных механизмов решения административных задач по обслуживанию сервера, например репликации данных между серверами, автоматического запуска задач, оповещения и др.).

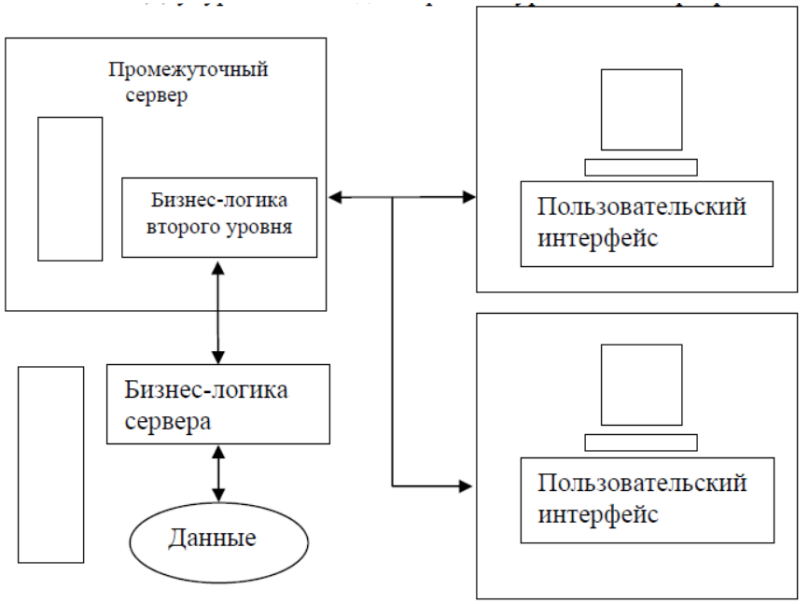
## Недостатки

- очень большая загрузка сервера;
- в настоящее время в коммерческих СУБД нет удобного инструмента для написания и отладки хранимых процедур и триггеров;
- отсутствие стандартов на хранимые процедуры.

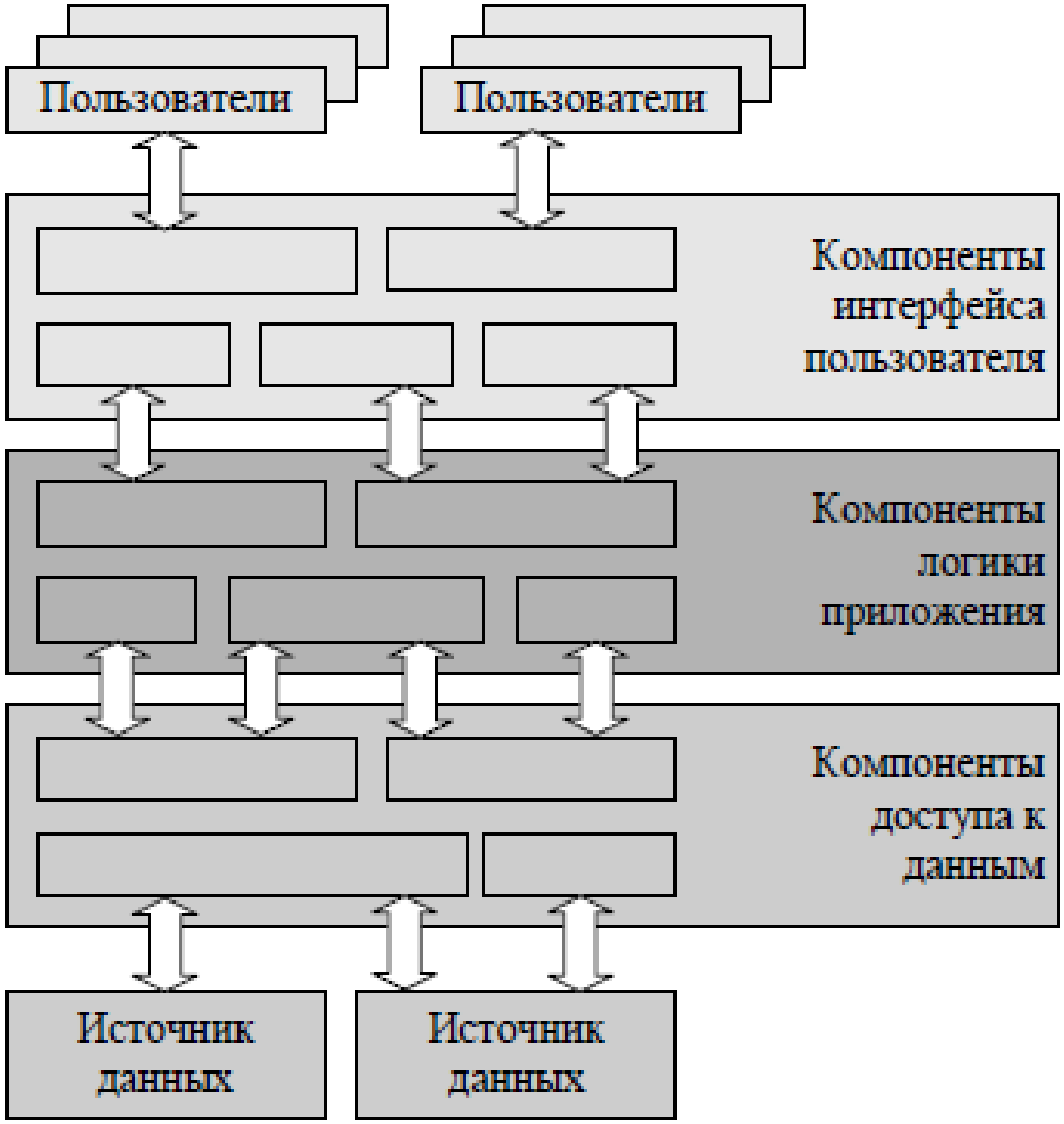

Компоненты приложения
Функции логики представления
Функции бизнес-логики
Функции управления данными
DP
RP
DBL
RDM
DDM
Распределенное представление (DP)
Удаленное представление (RP)
Распределенная бизнес-логика (DBL)
Удаленное управление данными (RDM)
Распределенное управление данными (DDM)
Совмещение DBL DDM
Клиент Сервер
Клиент Сервер
Клиент Сервер
Клиент Сервер
Клиент Сервер
Клиент Сервер

# Трехуровневые модели



# Компоненты распределенной системы



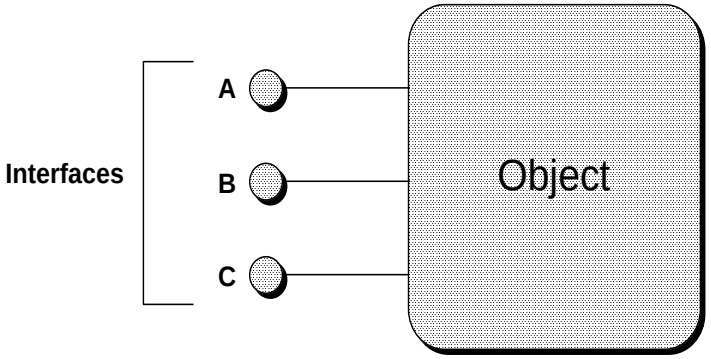
# Понятие компонентных технологий

Для описания и реализации распределенных систем введено понятие программного компонента.

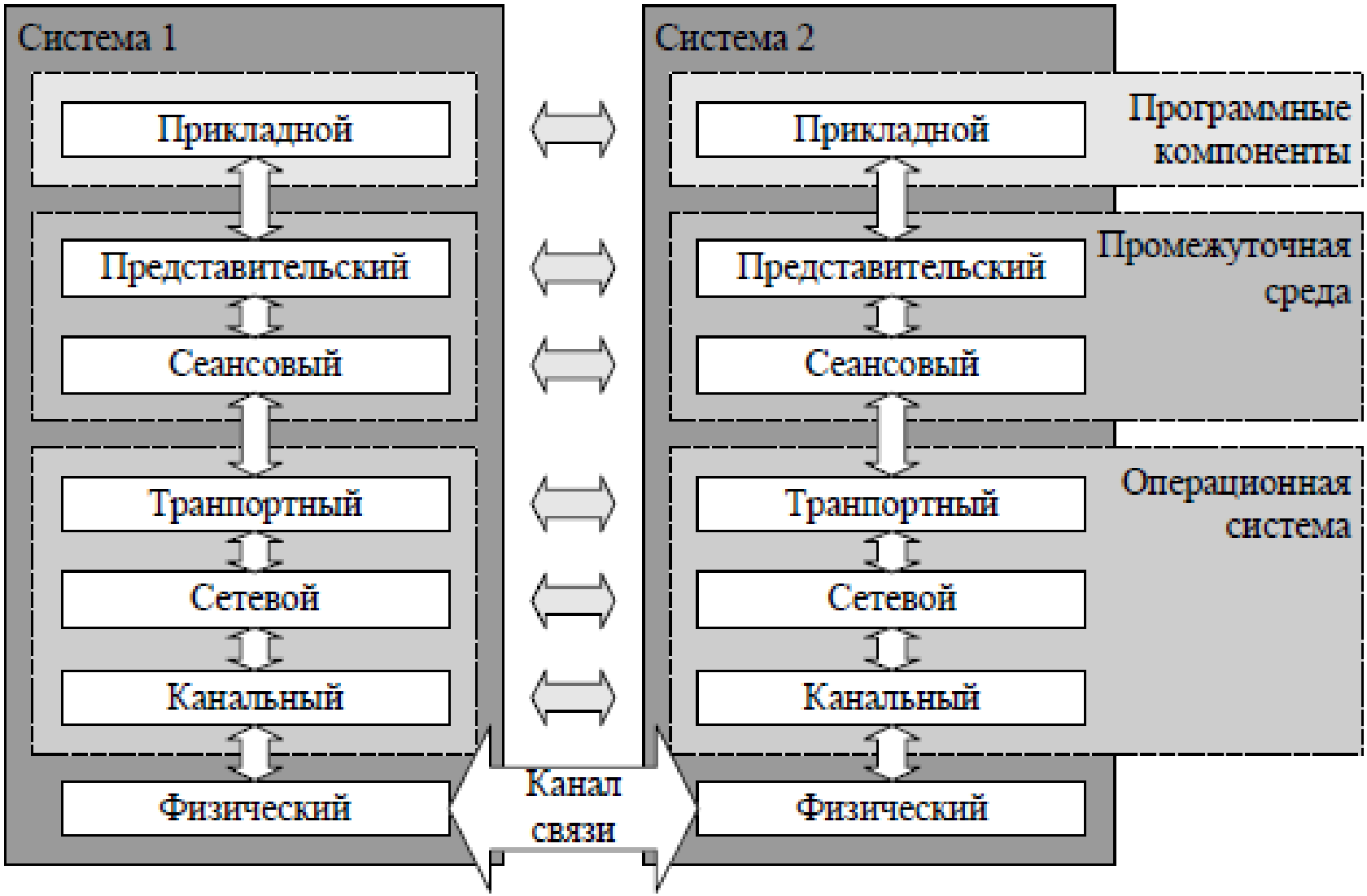
Программный компонент – это единица программного обеспечения, которая:

- исполняется на одном компьютере в пределах одного процесса,
- предоставляет некоторый набор сервисов,
- взаимодействует с другими компонентами через ее внешний интерфейс,
- выполняется на этом же компьютере, так и на удаленных компьютерах.

- через графический пользовательский интерфейс предоставляют свой сервис конечному пользователю



# Модель взаимодействия вычислительных систем



# Модель OSI (Open System Interconnection)

**Модель OSI состоит из семи уровней, каждый из которых представляет собой определенный этап процесса сетевой коммуникации. Каждый уровень выполняет определенную задачу процесса коммуникации, а затем передает данные вверх или вниз на следующий уровень.**

**Прикладной уровень обеспечивает взаимодействие пользовательским приложением и сетью: FTP , Telnet , SMTP , HTTP (передача гипертекста) и др.**

**Уровень представления (сжатие данных, шифрование, трансляция протоколов). На этом уровне работают шлюзы для соединения различных сетей и Редиректоры – программы, определяющая, чем должен быть обработан запрос – локальным компьютером или устройством сети**

**Сеансовый уровень – устанавливает и разрывает диалоги приложений разных компьютеров сети в дуплексном или полудуплексном режиме по установленным правилам обмена, обеспечивает безопасность и распознавание имен.**

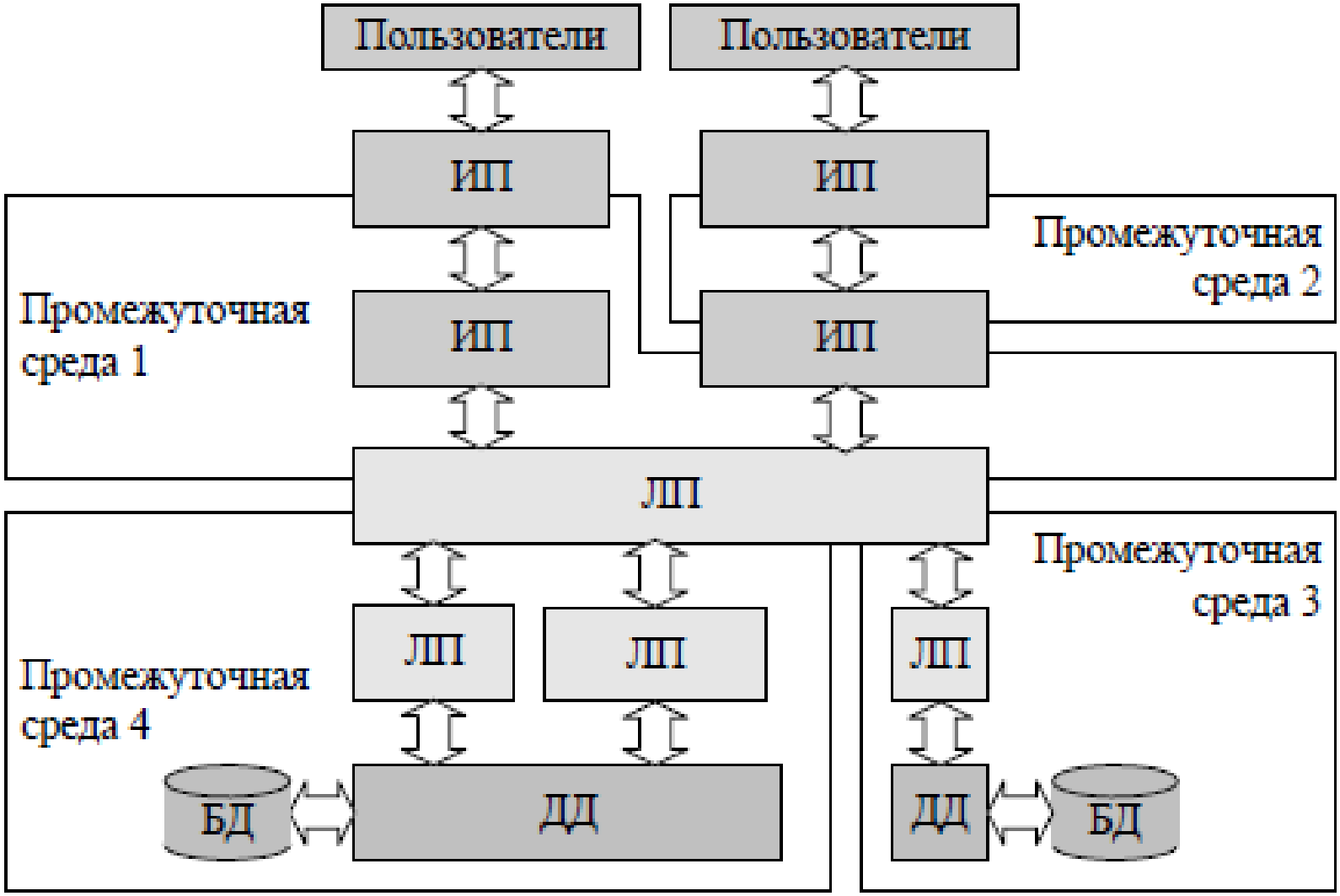
**Транспортный уровень – обеспечение целостности и последовательности пакетов данных. На этом уровне работает DNS, организуются порты и Сокеты приложений.**

**Сетевой уровень – маршрутизация пакетов. Устройства этого уровня – маршрутизаторы и коммутаторы сетевого уровня.**

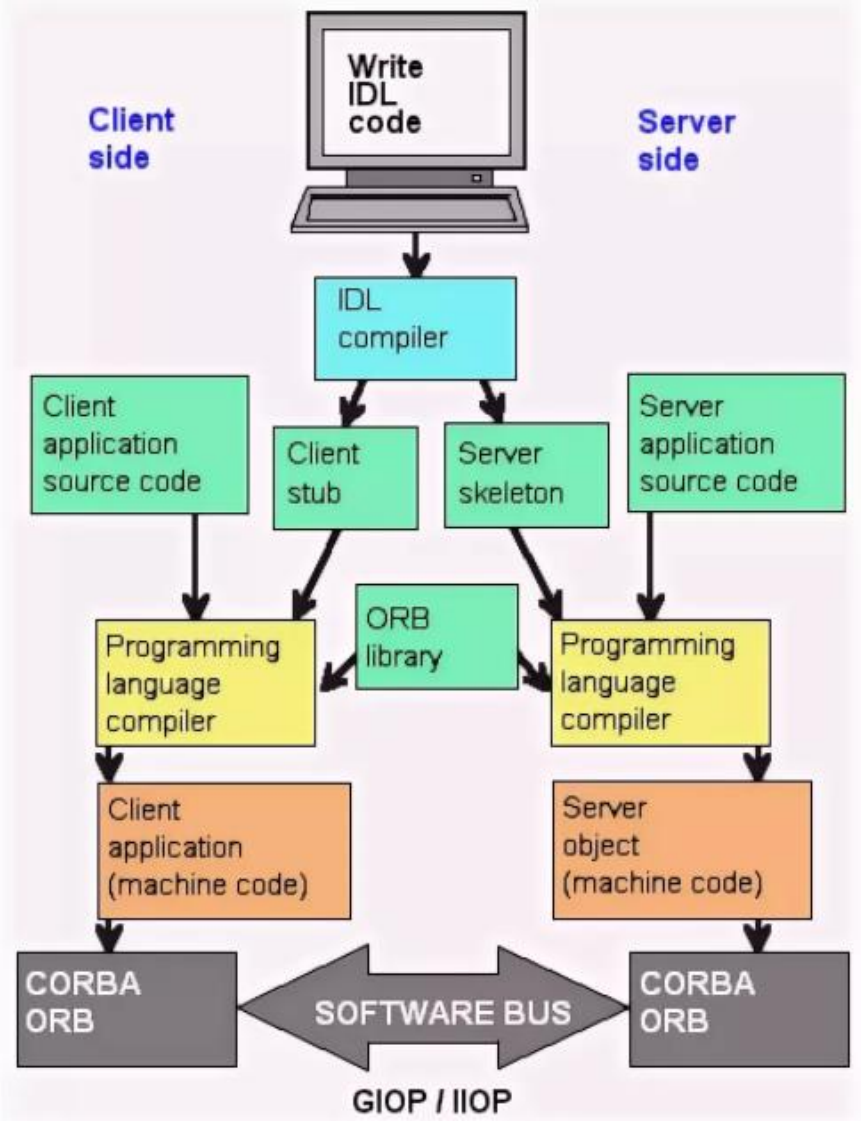
**Канальный уровень: MAC – управление доступом к сети (48-разрядный двоичный адрес сетевой карты) и LLC – управление логическими связями (определяет логическую топологию сети). На этом уровне работают мосты и коммутаторы уровня 2**

**Физический уровень –формирование аналоговых, цифровых, модулированных и других сигналов, синхронная и асинхронная передача и др.**

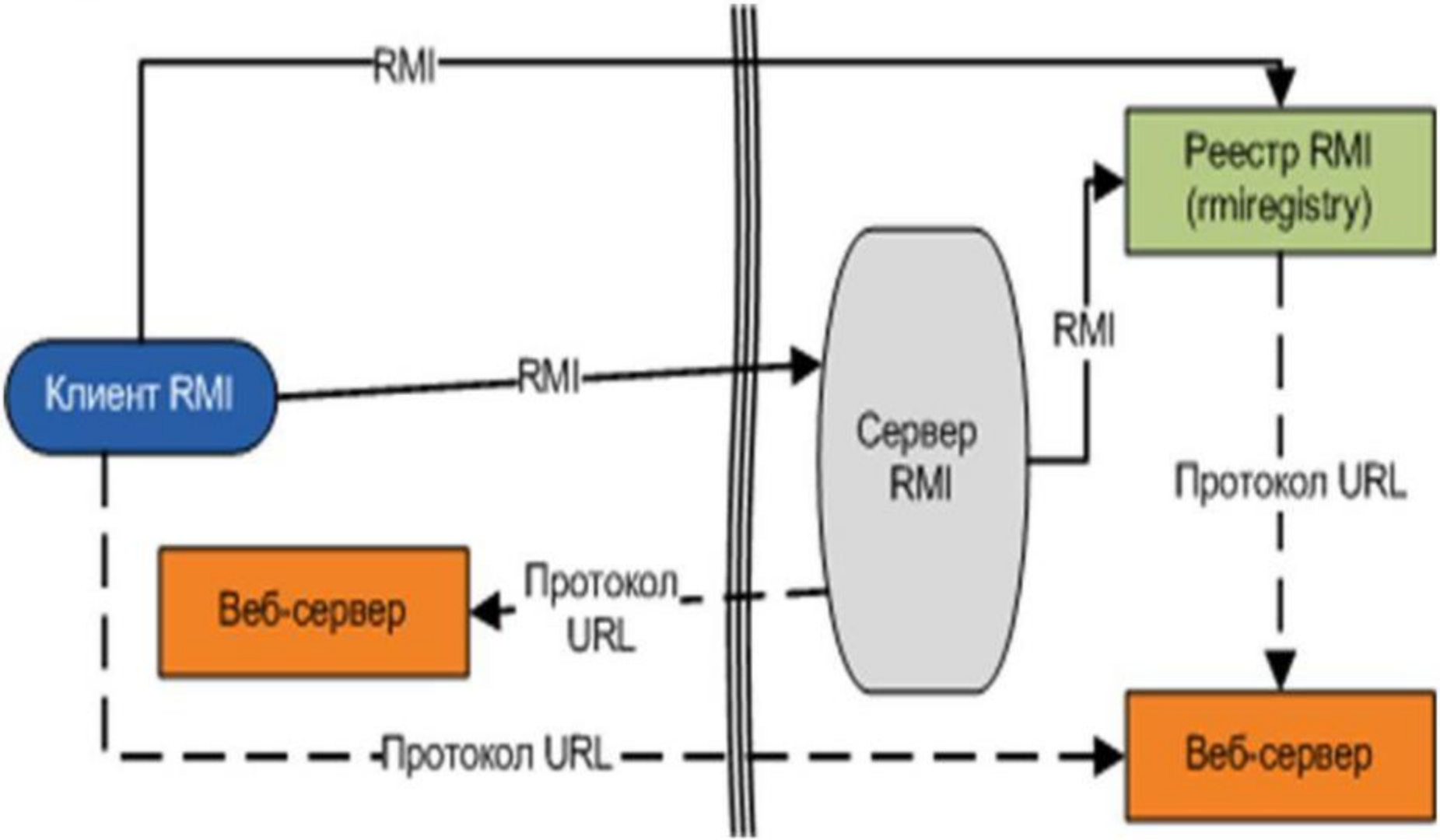
# Гетерогенная распределенная система



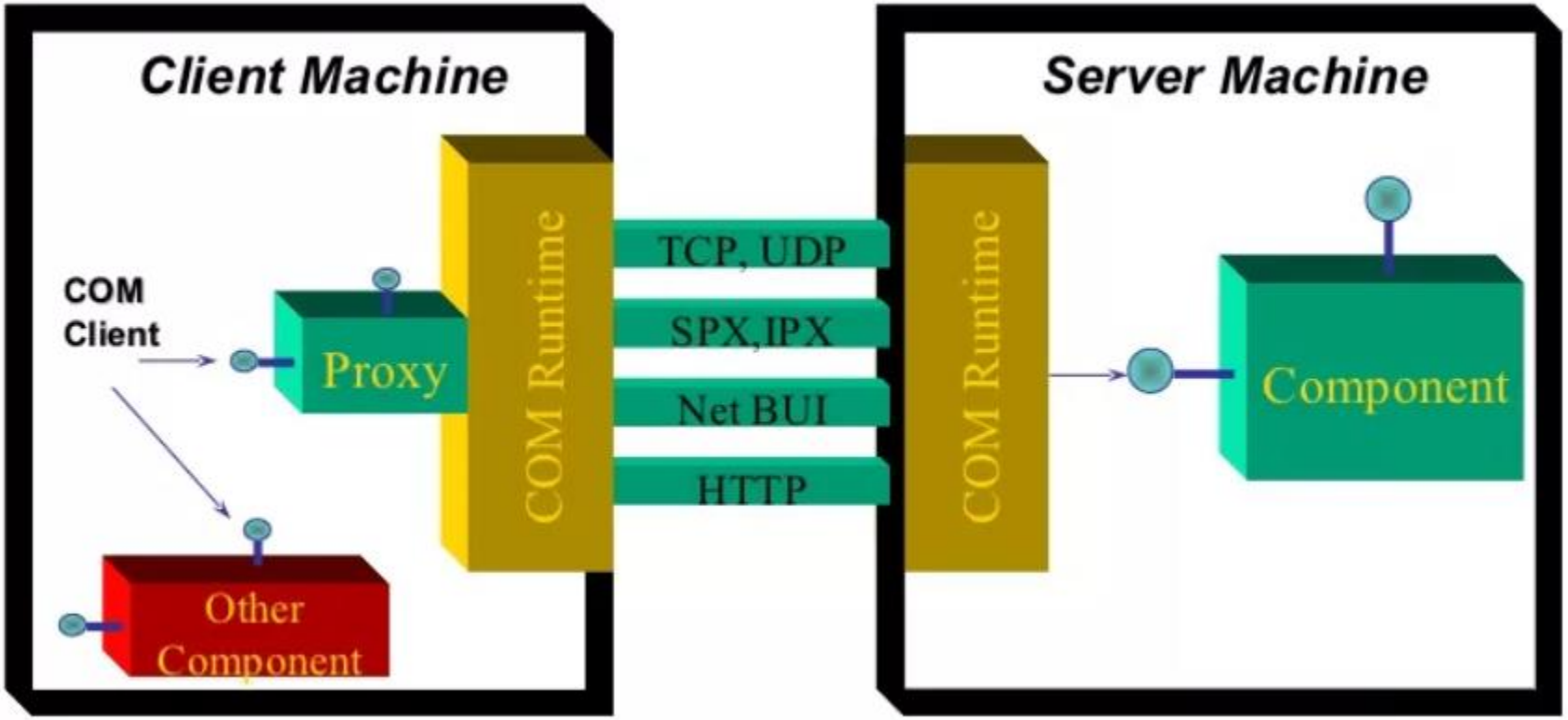
# Технология CORBA



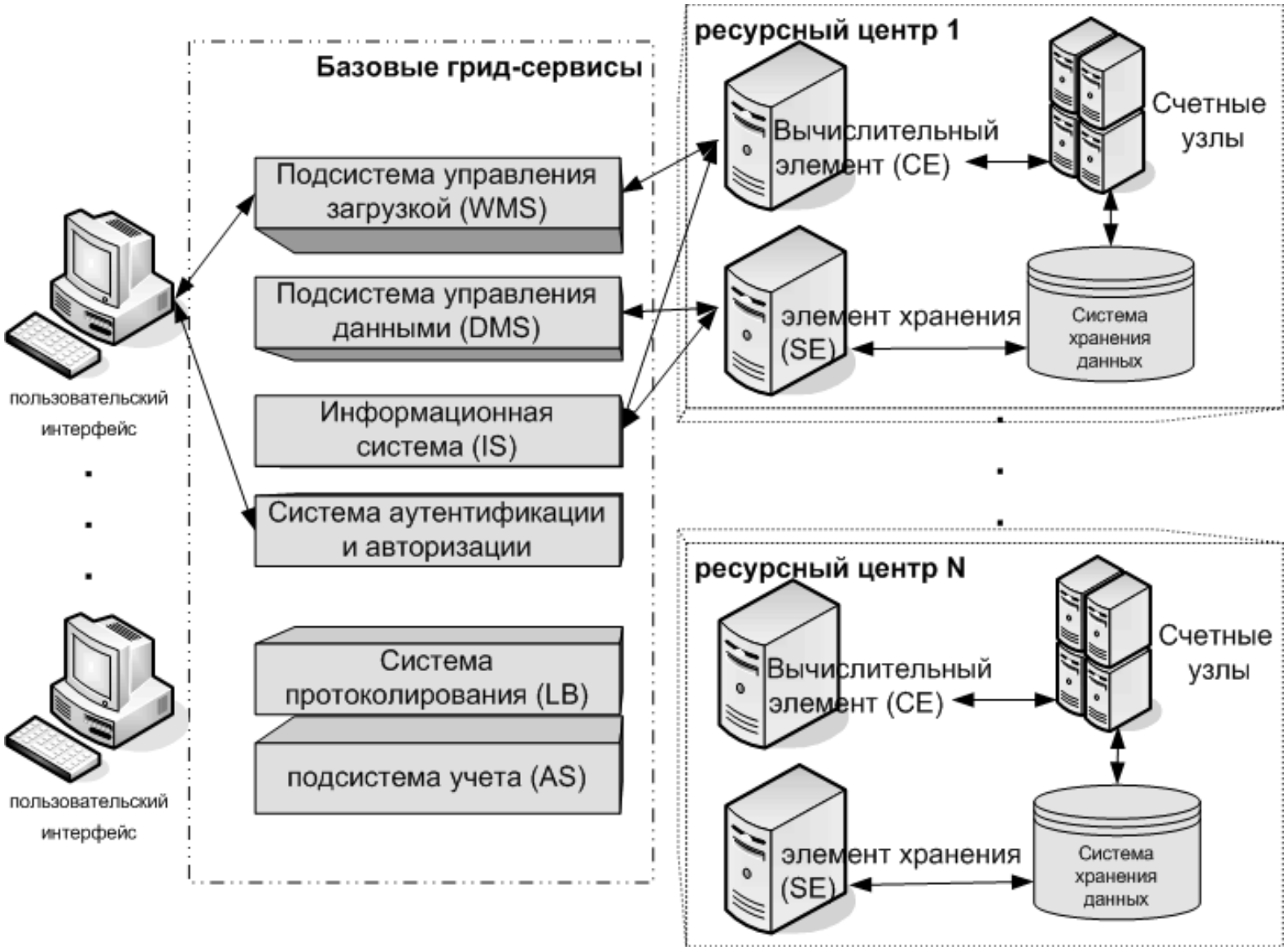
# Структура RMI приложения



# Технология DCOM



# Технология Grid



# Технология COM(1993)

Технология OLE –это набор API для создания и отображения документа. Обеспечивает совместное использование не только данных, но и кода
Объектная Модель Компонентов (Component Object Model - COM)

- Протоколы COM позволяют объединить множество компонентов для создания единого приложения: Например, текстовый редактор может сказать электронной таблице: “пользователь только что нажал на таблицу, поэтому необходимо запуститься, найти данные, и, как только закончишь работу –дай мне знать.”
- COM в настоящее время включает OLE как часть более широкой концепции OLE становится блоком стандартных интерфейсов COM

# Взаимодействие компонент распределенной системы

## Модели взаимодействия распределенных компонент

- Использование сообщений
- Удаленный вызов

# Передача сообщений в распределенных системах



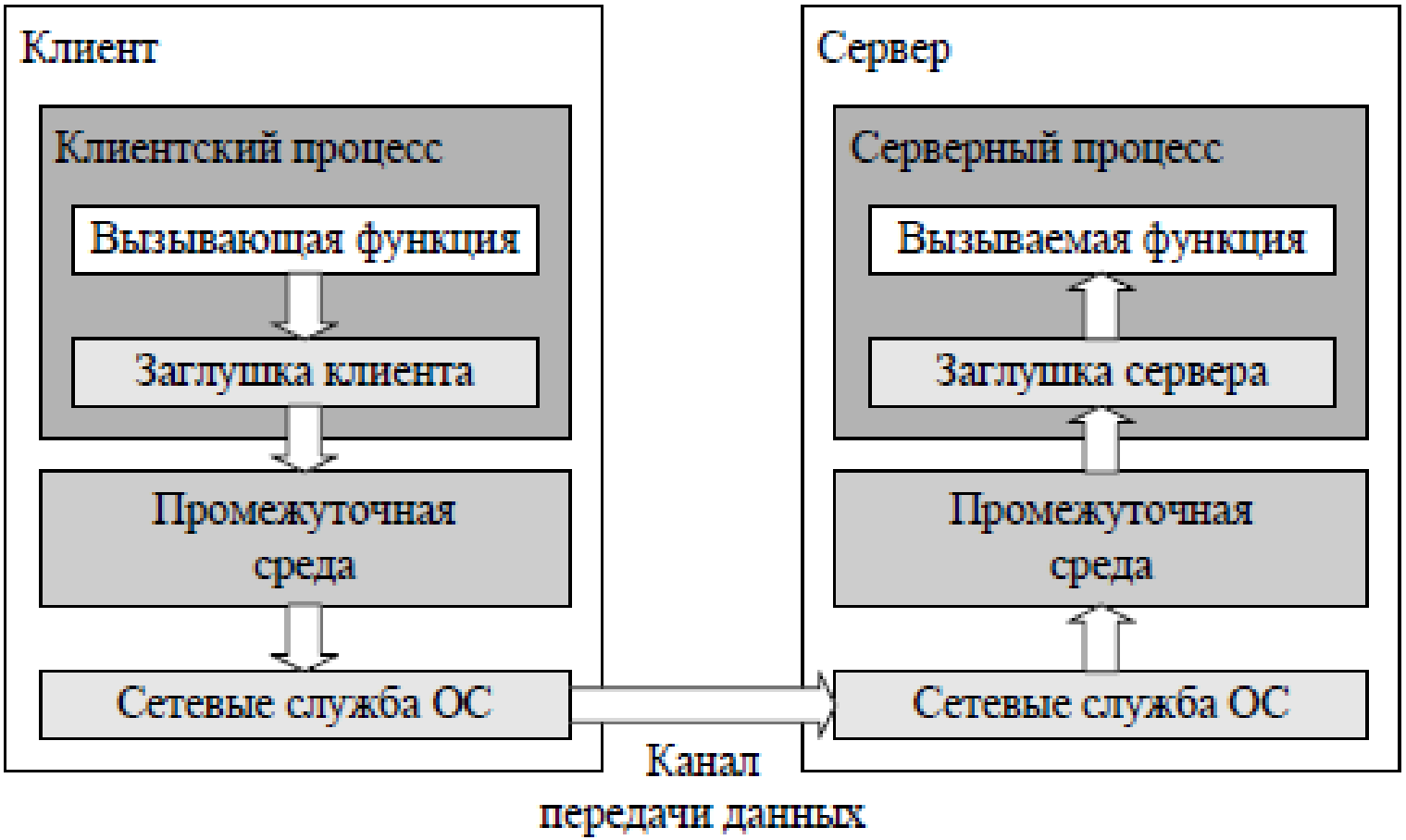
## Преимущества

– время функционирования сервера может быть не связано со временем работы клиентов;
– независимость промежуточной среды от средства разработки компонент и используемого языка программирования;
– считывать и обрабатывать заявки из очереди могут несколько независимых компонент, что дает возможность достаточно просто создавать устойчивые и масштабируемые системы.

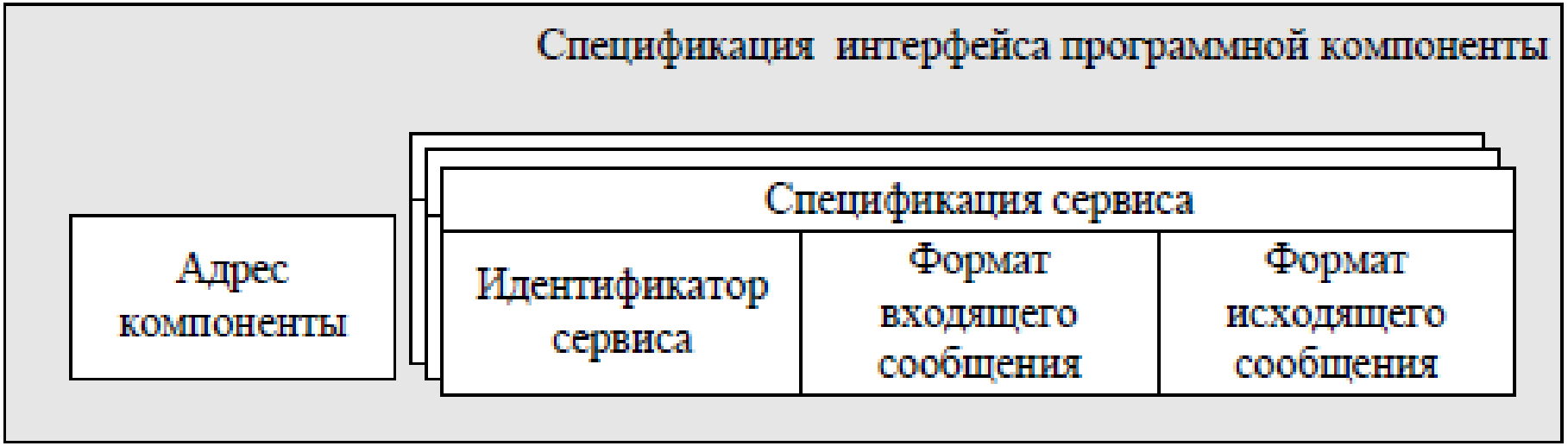
## Недостатки

– необходимость явного использования очередей распределенным приложением;
– сложность реализации синхронного обмена;
– определенные накладные расходы на использование менеджеров очередей;
– сложность получение ответа: передача ответа может потребовать отдельной очереди на каждый компонент, посылающий заявки.

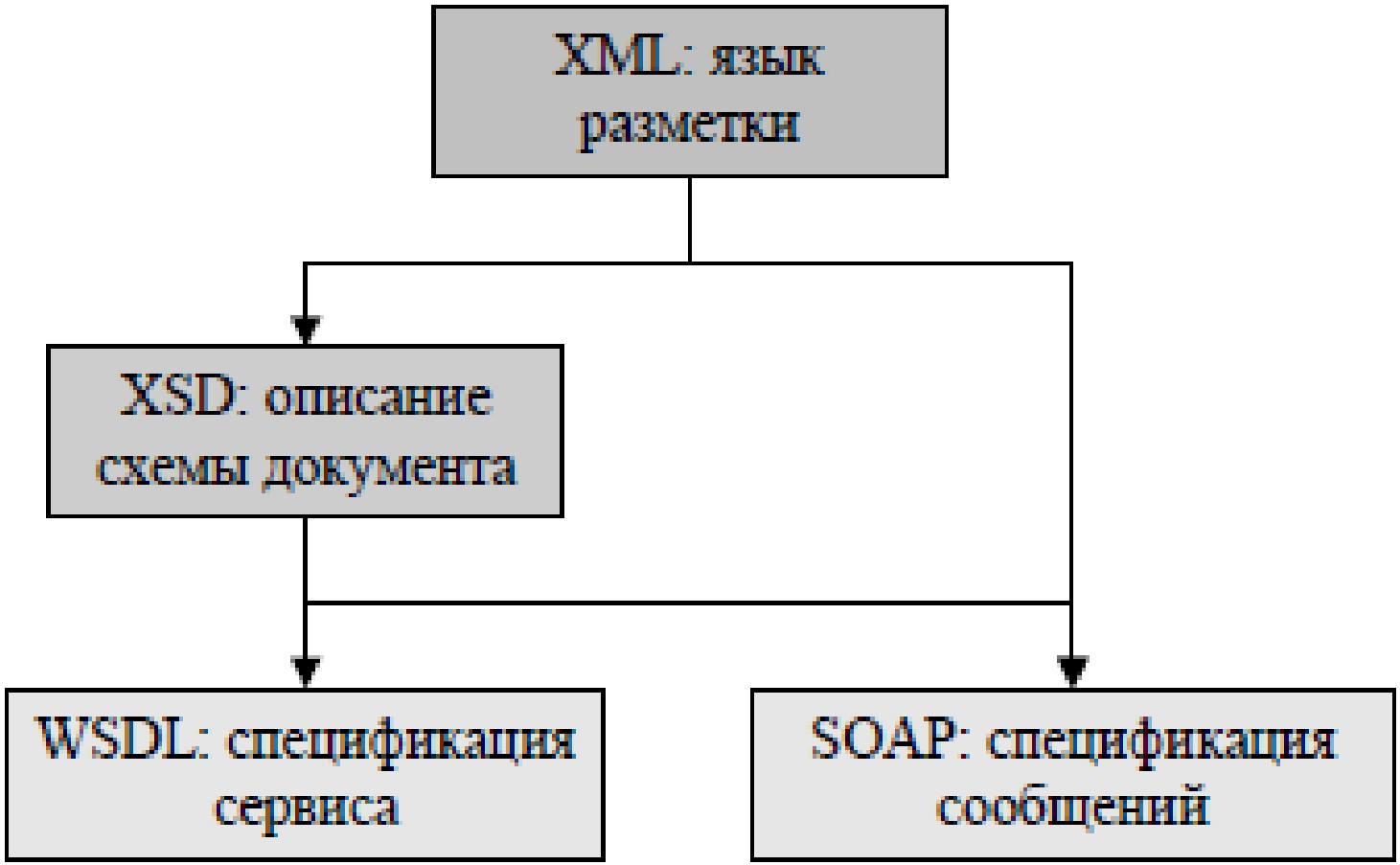
# Удаленный вызов процедур



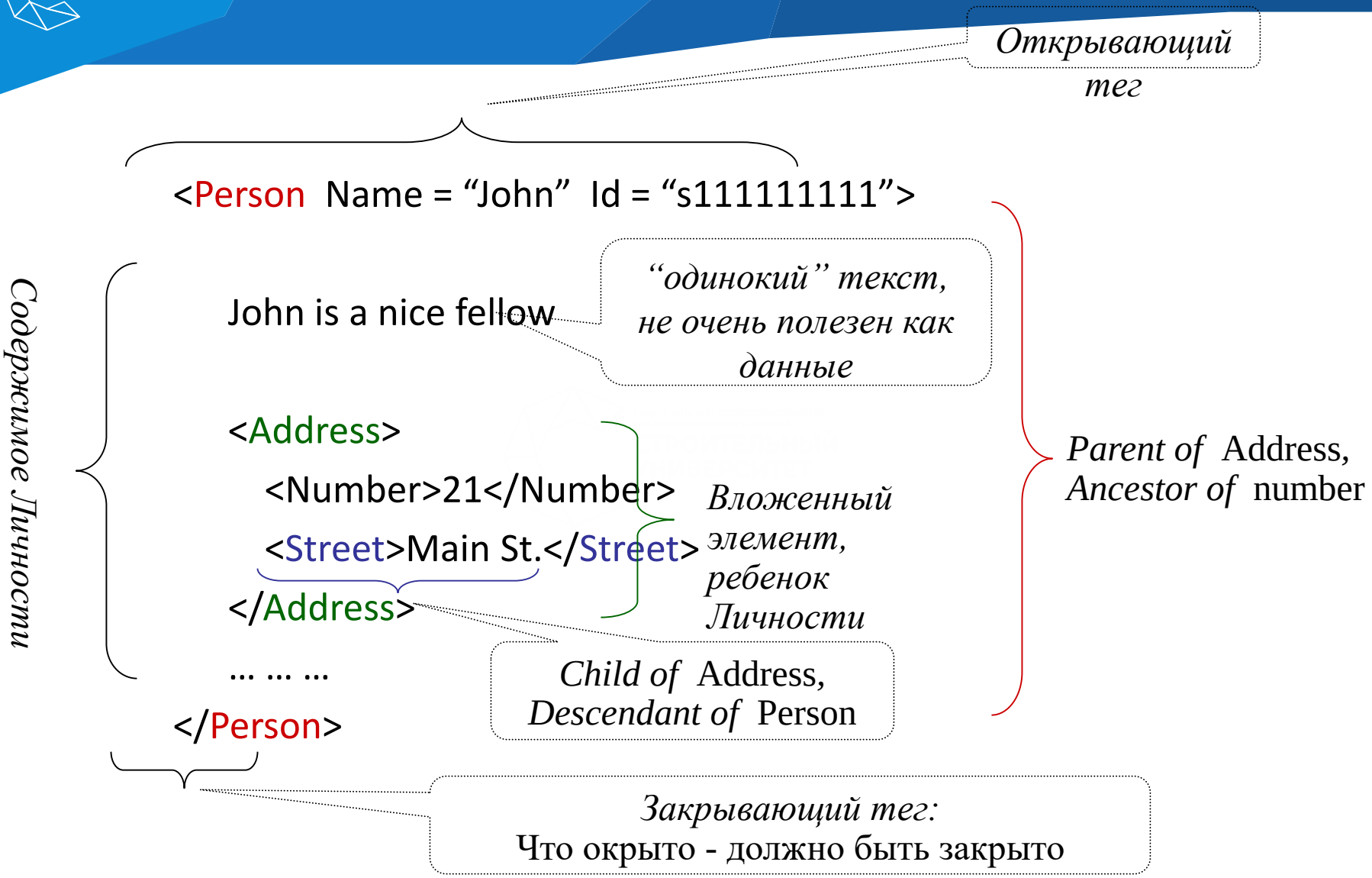
# Описание интерфейса программной компоненты



# Язык XML и схемы XML



# Терминология



# Требования к документу xml

- Документ может начинаться декларацией – строкой, указывающей как минимум версию стандарта XML.
  В качестве других атрибутов могут быть указаны кодировка символов и внешние связи.
- После декларации в XML-документе могут располагаться ссылки на документы, определяющие структуру текущего документа.
- Каждый XML-документ должен содержать только один корневой элемент (root element или document element)
- Тег должен закрываться в том же теге, в котором был открыт.
- Любой открывающий тег должен иметь закрывающий.
  (или **<author/>**)
- Наименования тегов чувствительны к регистру.
- Все атрибуты тегов должны быть заключены либо в одинарные, либо в двойные кавычки
- Все пробелы являются значимыми, т.е. при отображении пробелы в XML-документе не будут устраняться.
- В XML есть несколько зарезервированных символов, которые используются только как элементы синтаксиса XML. Такими зарезервированными символами являются пять следующих знаков: <, >, &, “, ‘.

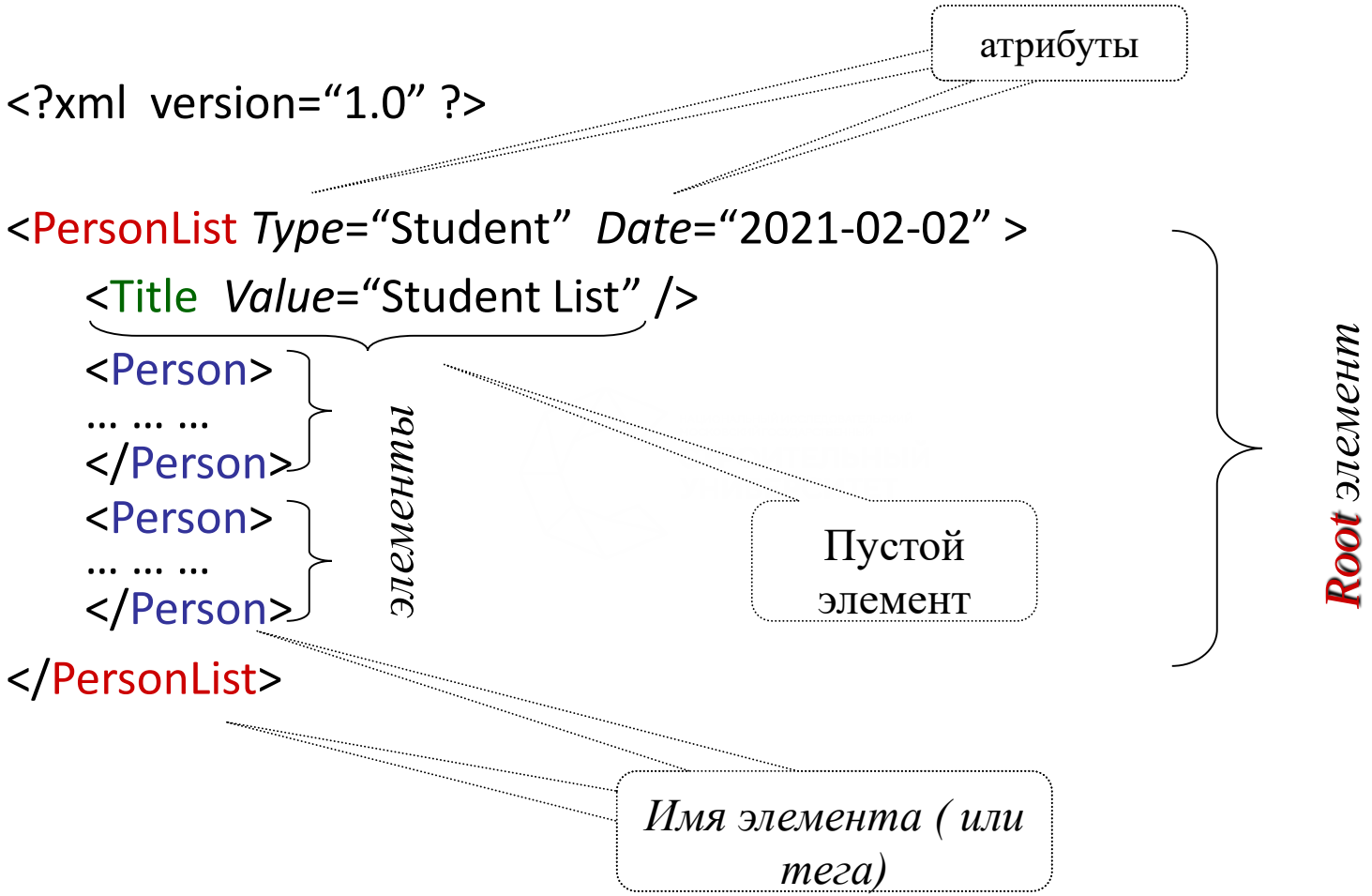
# Структура

## XML-документ состоит из двух основных частей:

- пролога и тело Документ

```
<?xml version=”1.1” encoding=”UTF-8” standalone=”yes”?>
<!DOCTYPE sampledoc SYSTEM “sample.dtd”>
```

# Терминология




- Элементы могут быть вложенными
- Корневой элемент включает все остальные теги

# Синтаксис XML. XML Атрибуты

- **Атрибут определяет некоторые свойства элемента**
- **Представляется как пара “название-значение”**

```
<product>
  <name language="French">trompette six trous</name>
  <price currency="Euro">420.12</price>
  <address format="XLB56" language="French">
    <street>31 rue Croix-Bosset</street>
    <zip>92310</zip>
    <city>Sevres</city>
    <country>France</country>
  </address>
</product>
```

- **Внутри тега можно определить любое кол-во атрибутов**
- **Значения атрибутов должны быть расположены внутри двойных кавычек.**

# Комментарии

- Для написания комментариев в XML следует заключать их, как и в HTML, между **<!--** и **-->** .
- Комментарии можно размещать в любом месте документа, но не внутри:
  - других комментариев
  - значений атрибутов
  - тегов

```
<!-- комментарий <!-- Неправильный комментарий --> -->
```

```
<book title=”BLR<!-- Неправильный комментарий -->”/>
```

```
<book <!-- Неправильный комментарий -->/>
```

# Document Type Definitions (DTD) – определение типа документа

- DTD: Document Type Definition – один из способов спецификации структуры XML документа.
- DTD добавляет синтаксические требования в дополнение к требованиям well-formed документа.
- DTDs помогает
  - Обнаруживать ошибки при создании или редактирования XML документов.
  - Упрощает процесс обработки XML документов.
- Использует “регулярные выражения” как синтаксис для спецификации граматики XML документа.
- Имеет ограничения: нет типов данных, нет возможности описания ограничений, нет поддержки схем.

# Способы контроля правильности XML-документа

- DTD – определения (Document Type Definition)
- Схемы данных(Semantic Schema)

# XML-документ адресной книги с описанием DTD внутри самого файла

```
<!DOCTYPE db SYSTEM "schema.dtd">
```

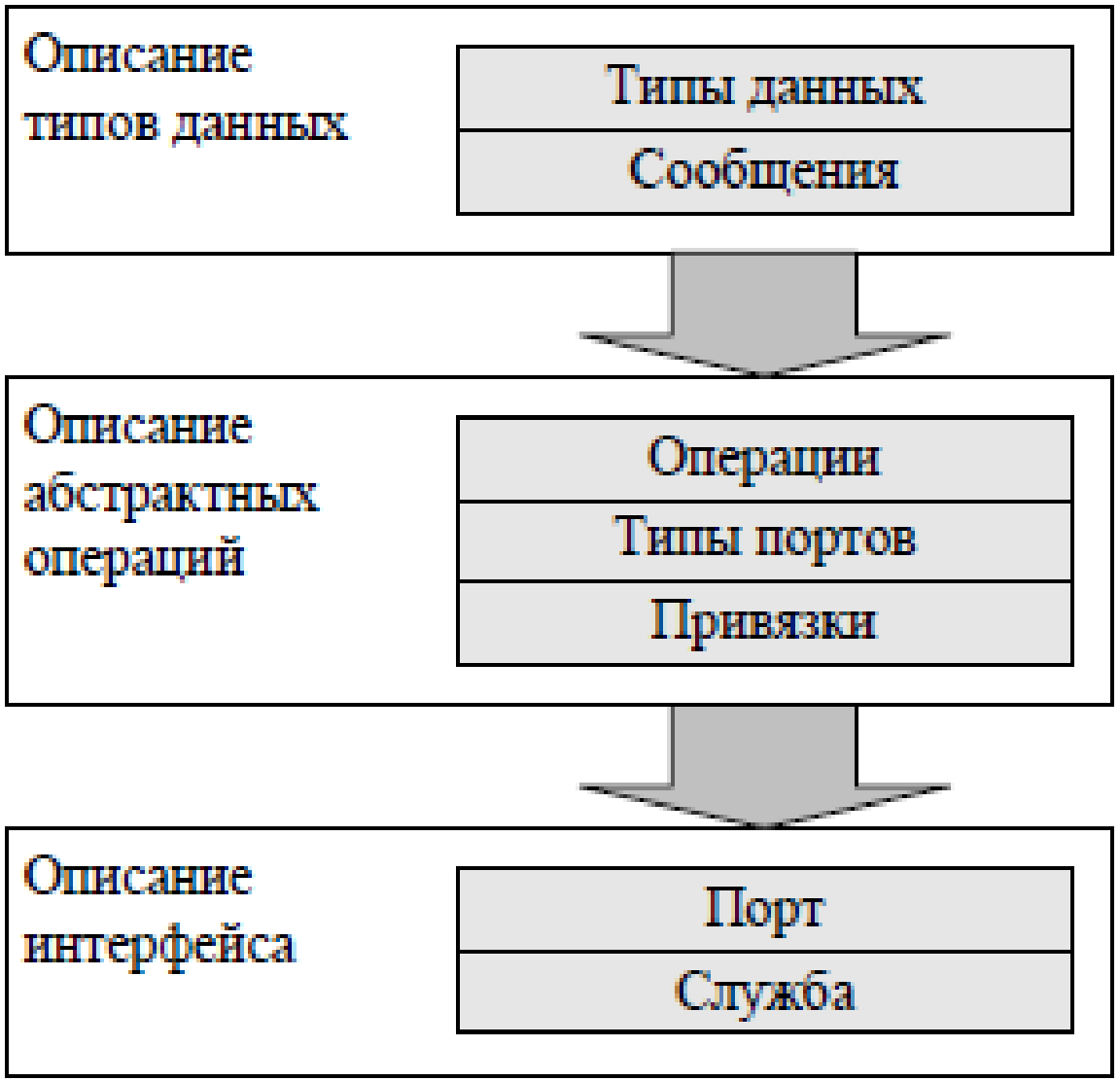
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE addressbook [
   <!ELEMENT addressbook (person*)>
   <!ELEMENT person (name, greet?, address*,
                     (fax | tel)*, email*)>
   <!ELEMENT name    (#PCDATA)>
   <!ELEMENT greet   (#PCDATA)>
   <!ELEMENT address (#PCDATA)>
   <!ELEMENT tel     (#PCDATA)>
   <!ELEMENT fax   (#PCDATA)>
   <!ELEMENT email   (#PCDATA)>
]>
```

```
<!ENTITY hello 'Добрый день!' >
```

# XML Схема для заказа товаров

```
<xsd:schema xmlns:xsd=http://www.w3.org/1999/XMLSchema>
<xsd:element name="shipOrder" type="order"/>
<xsd:complexType name="order">
  <xsd:element name="shipTo" type="shipAddress"/>
  <xsd:element name="items" type="cdItems"/>
</xsd:complexType>
<xsd:complexType name="shipAddress">
  <xsd:element name="name“ type="xsd:string"/>
  <xsd:element name="street" type="xsd:string"/>
  <xsd:element name="address" type="xsd:string"/>
  <xsd:element name="country" type="xsd:string"/>
</xsd:complexType>
```

# Составные части WSDL-документа



# SOAP – язык сообщений

**Сообщение SOAP выглядит так:**

SOAP- конверт

- SOAP-заголовок
  - Элемент заголовка 1
  - Элемент заголовка 2
  - …
  - Элемент заголовка N
- Тело SOAP
  - Элемент тела N
  - …
  - Элемент тела 2
  - Элемент тела 1

# Пример

**Пример SOAP-запроса на сервер интернет-магазина:**

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <soap:Body>
                <getProductDetails
                    xmlns="http://warehouse.example.com/ws">
                        <productID>12345</productID>
                </getProductDetails>
        </soap:Body>
</soap:Envelope>
```