Вводный материал

СОДЕРЖАНИЕ

1. Технология Client-Server

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

Как правило компьютеры и программы, входящие в состав информационной системы, не являются равноправными.

Некоторые из них владеют ресурсами (файловая система, процессор, принтер, база данных и т.д.), другие имеют возможность обращаться к этим ресурсам.

Компьютер (или программу), управляющий ресурсом, называют **сервером (back end)** этого ресурса (файл-сервер, сервер базы данных, вычислительный сервер...).

Компьютер (или программу), запрашивающая ресурсы у сервера, называют **клиентов (front end)**.

Клиент и сервер какого-либо ресурса могут находится как на одном компьютере, так и на различных компьютерах, связанных сетью.

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

В рамках многоуровневого представления вычислительных систем можно выделить три группы функций, ориентированных на решение различных подзадач:

- функции ввода и отображения данных (обеспечивают взаимодействие с пользователем);
- прикладные функции, характерные для данной предметной области;
- функции управления ресурсами (файловой системой, базой данных и т.д.).

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

Выполнение этих функций в основном обеспечивается программными средствами, которые можно представить в виде взаимосвязанных компонентов, где:

- компонент представления отвечает за пользовательский интерфейс;
- прикладной компонент реализует алгоритм решения конкретной задачи;
- компонент управления ресурсом обеспечивает доступ к необходимым ресурсам.

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

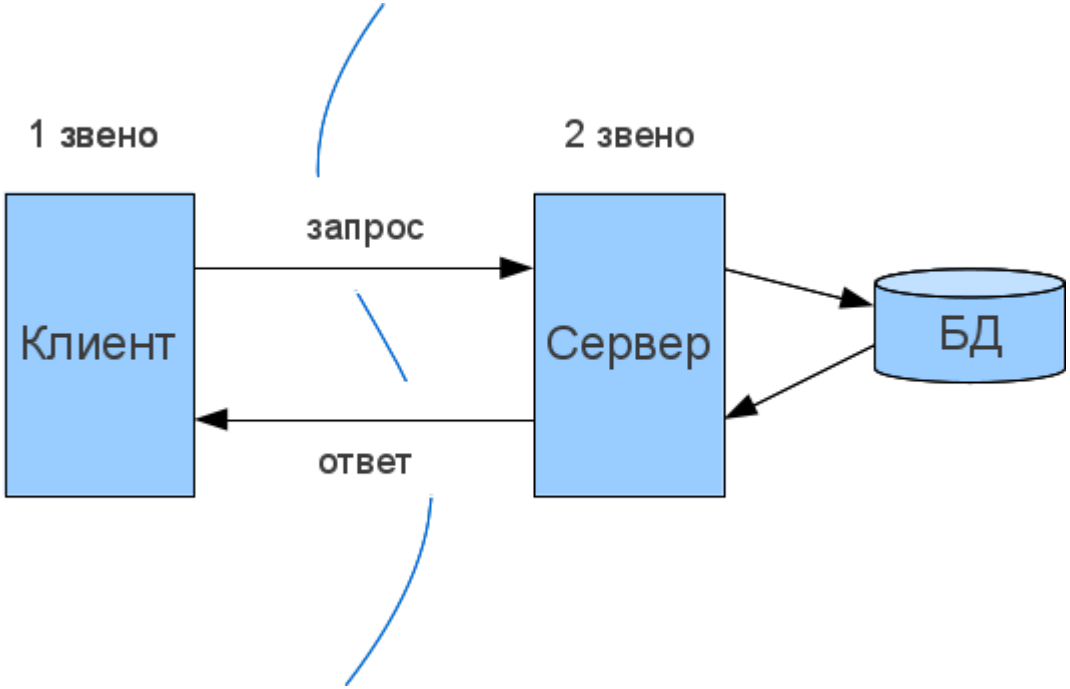
Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты, потребители этих функций.

Практические реализации такой архитектуры называются **клиент-серверными технологиями**.

Каждая технология определяет собственные или использует имеющиеся правила взаимодействия между клиентом и сервером, которые называются **протоколом обмена** (протоколом взаимодействия).

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

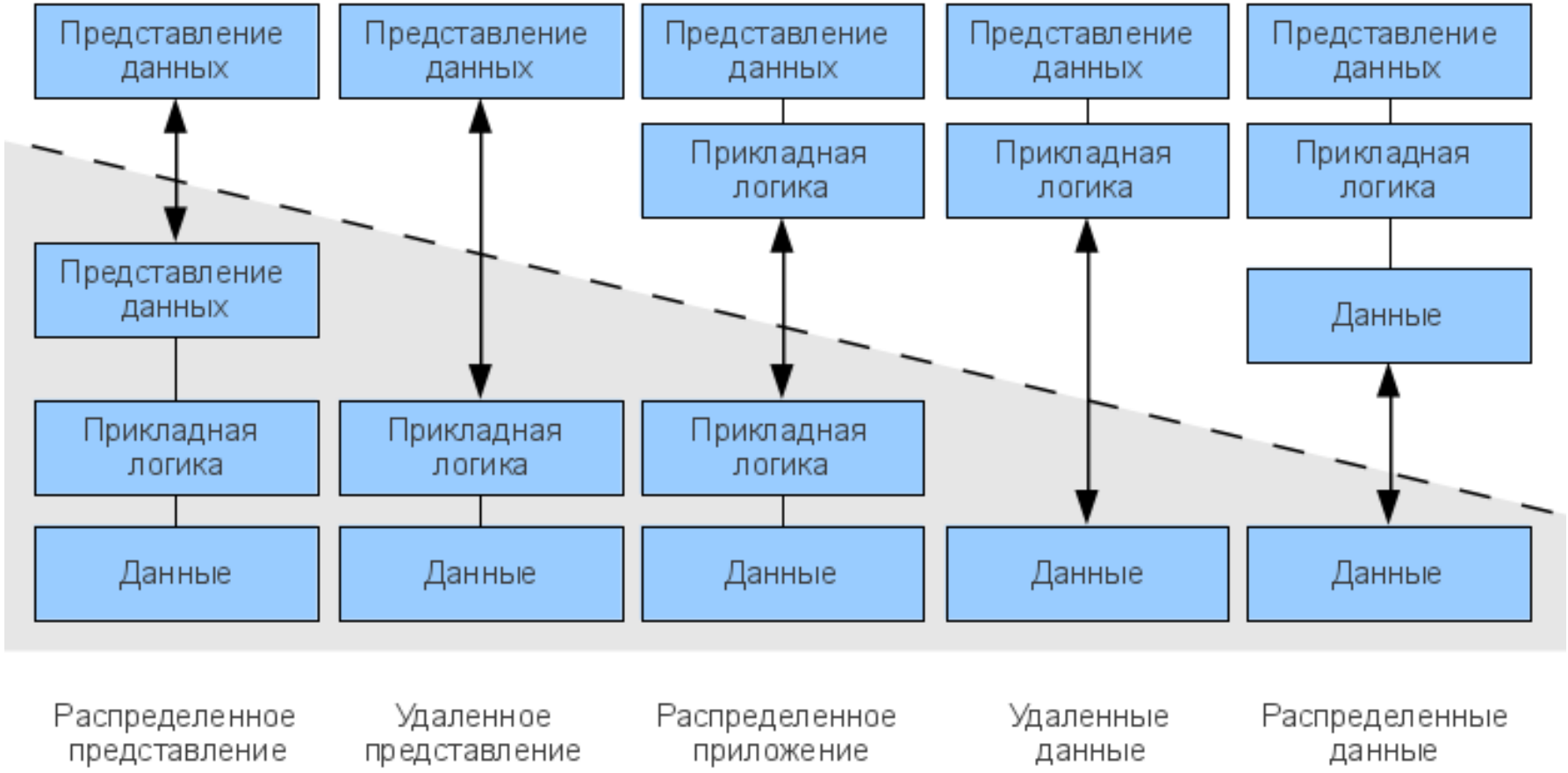
Двухзвенной архитектурой называется архитектура с распределением трех базовых компонентов между двумя узлами (клиентом и сервером).



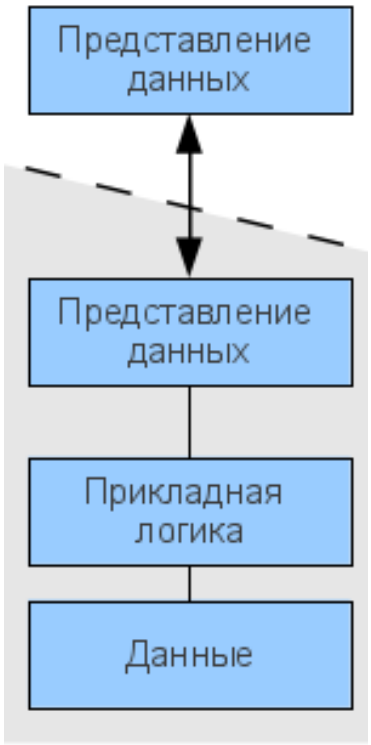
Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы.

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

Расположение компонентов на стороне клиента или сервера определяет следующие основные модели их взаимодействия в рамках двухзвенной архитектуры:



1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

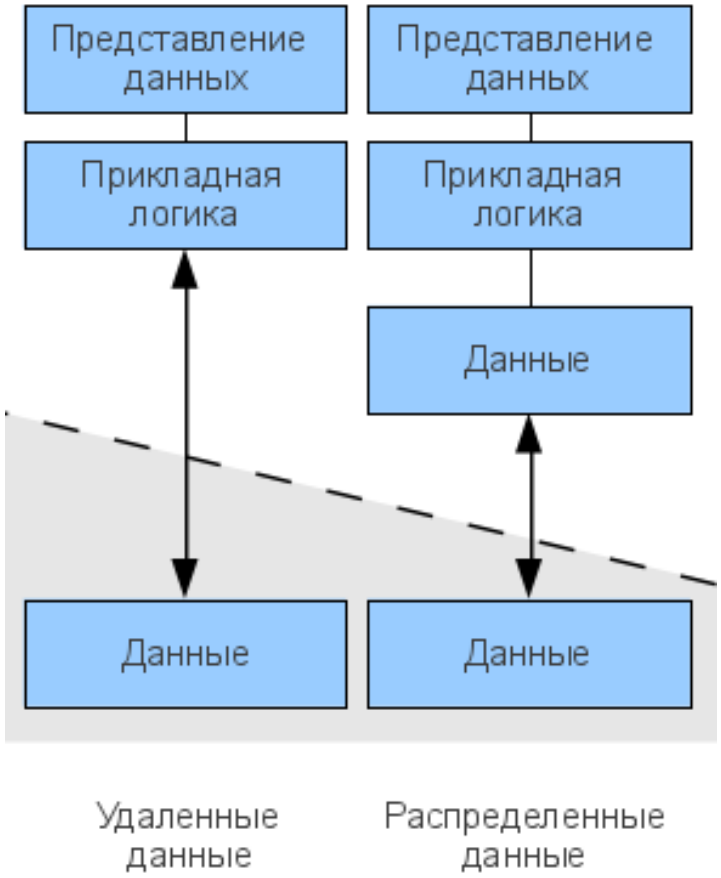


Распределенное представление

Исторически первой появилась модель распределенного представления данных (**модель сервер терминалов**). Она реализовывалась на универсальной ЭВМ (мэйнфрейме), выступавшей в роли сервера, с подключенными к ней алфавитно-цифровыми терминалами. Пользователи выполняли ввод данных с клавиатуры терминала, которые затем передавались на мэйнфрейм и там выполнялась их обработка, включая формирование «картинки» с результатами. Эта «картинка» и возвращалась пользователю на экран терминала.

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

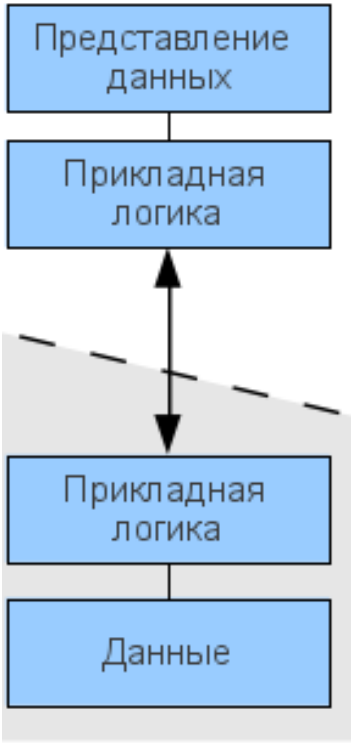
С появлением персональных компьютеров и локальных сетей, была реализована **модель файлового сервера**, представлявшего доступ файловым ресурсам, в т.ч и к удаленной базе данных.



В этом случае выделенный узел сети является файловым сервером, на котором размещены файлы базы данных. На клиентах выполняются приложения, в которых совмещены компонент представления и прикладной компонент (СУБД и прикладная программма), использующие подключенную удаленную базу как локальный файл. Такая модель показала свою неэффективность ввиду того, что при активной работе с таблицами БД возникает большая нагрузка на сеть.

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

С появлением специализированных СУБД появилась возможность реализации другой модели доступа к удаленной базе данных — **модели сервера баз данных**.



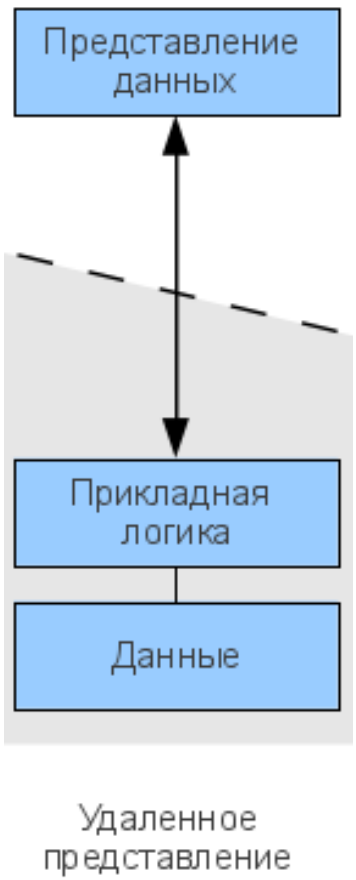
Распределенное приложение

В этом случае ядро СУБД функционирует на сервере, прикладная программа на клиенте, а протокол обмена обеспечивается с помощью языка SQL.

С разработкой и внедрением на уровне серверов баз данных механизма хранимых процедур появилась концепция активного сервера БД. В этом случае часть функций прикладного компонента реализованы в виде хранимых процедур, выполняемых на стороне сервера. Остальная прикладная логика выполняется на клиентской стороне.

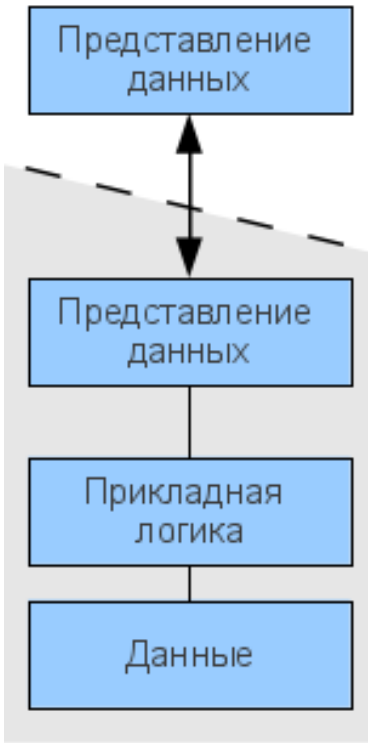
Основной недостаток — ограниченность средств разработки хранимых процедур по сравнению с языками высокого уровня.

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»



Реализация прикладного компонента на стороне сервера представляет следующую модель — **сервер приложений**. Перенос функций прикладного компонента на сервер снижает требования к конфигурации клиентов и упрощает администрирование, но представляет повышенные требования к производительности, безопасности и надежности сервера.

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»



Распределенное представление

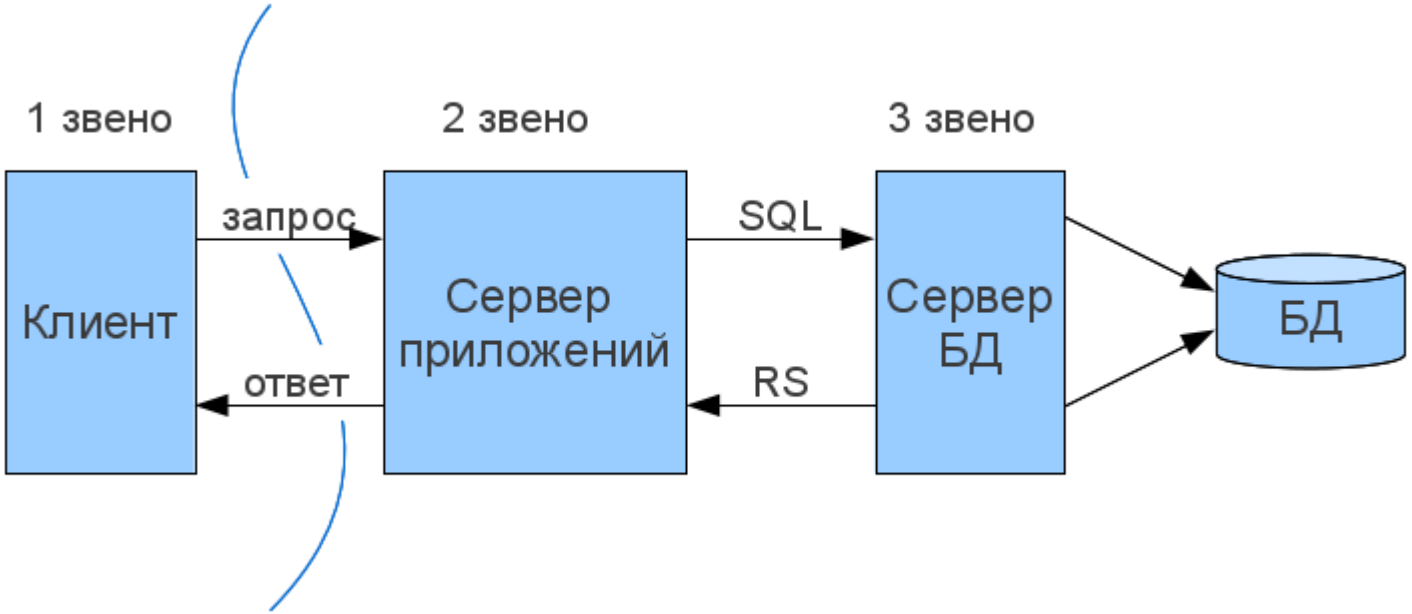
В настоящее время намечается тенденция возврата к тому, с чего начиналась клиент-серверная архитектура — к централизации вычислений на основе модели терминал-сервера. В современной реинкарнации терминалы отличаются от своих алфавитно-цифровых предков тем, что имея минимум программных и аппаратных средств, представляют мультимедийные возможности (в т.ч. графический пользовательский интерфейс).

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

Еще одна тенденция в клиент-серверных технологиях связана со все большим использованием распределенных вычислений. Они реализуются на основе модели сервера приложений, где сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная клиент-серверная архитектура становится **трехзвенной**.

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

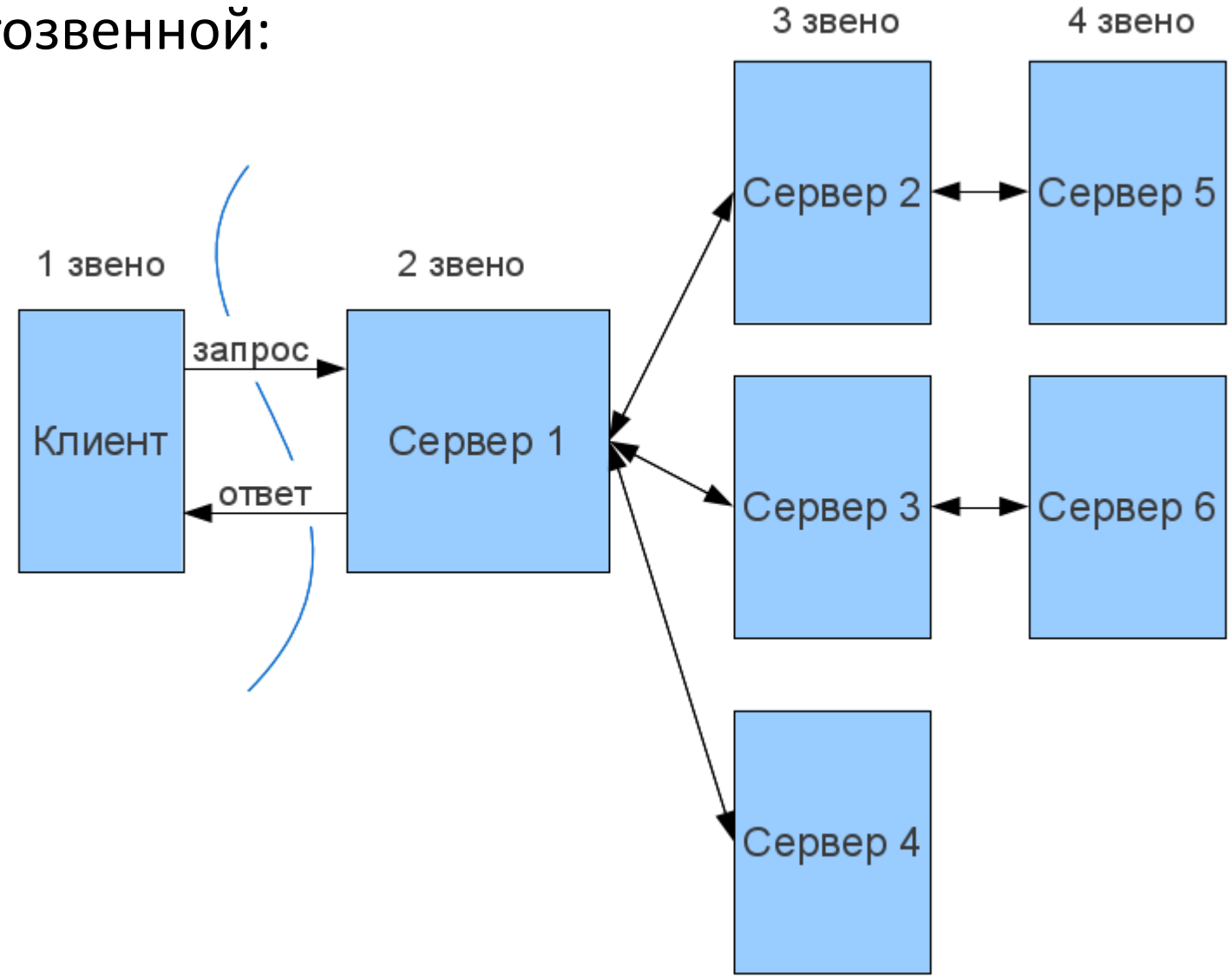
Как правило, третьим звеном в трехзвенной архитектуре становится сервер приложений, т.е. компоненты распределяются следующим образом:



Представление данных — на стороне клиента.
Прикладной компонент — на выделенном сервере приложений (как вариант, выполняющем функции промежуточного ПО).
Управление ресурсами — на сервере БД, который и представляет запрашиваемые данные.

1.1 АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР»

Трехзвенная архитектура может быть расширена до многозвенной:



1.2 ТЕХНОЛОГИЯ «КЛИЕНТ-СЕРВЕР»

Архитектура клиент-сервер применяется в большом числе сетевых технологий, используемых для доступа к различным сетевым сервисам.

Web-серверы
— сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

Примеры веб-серверов: Apache (свободный веб-сервер), IIS (Microsoft), nginx, Google Web Server (веб-сервер, основанный на Apache и доработанный компанией Google) и др.

Серверы приложений

— Предназначены для централизованного решения прикладных задач в некоторой предметной области. Для этого пользователи имеют право запускать серверные программы на исполнение. Использование серверов приложений позволяет снизить требования к конфигурации клиентов и упрощает общее управление сетью.

Примеры серверов приложений: WildFly (Java EE-сервер приложений), WebSphere Application Server (IBM), Zope (написанный на Python) и др.

Серверы баз данных

— Серверы баз данных используются для обработки пользовательских запросов на языке SQL. При этом СУБД находится на сервере, к которому и подключаются клиентские приложения.

Примеры серверных СУБД: SQL SERVER (Microsoft), SQL BASE SERVER, Oracle SERVER (Oracle Corporation), IBM DB2, Informix, PostgreSQL, MySQL и др.

Файл-серверы

— Файл-сервер хранит информацию в виде файлов и представляет пользователям доступ к ней. Как правило файл-сервер обеспечивает и определенный уровень защиты от несанкционированного доступа.

Примеры файл серверов: средства операционных систем, Samba, FreeNAS (свободная операционная система для сетевого хранилища на основа ОС FreeBSD) и др.

Почтовые серверы

— Представляют услуги по отправке и получению электронных почтовых сообщений.

Примеры свободных файловых серверов: Courier Mail Server, Cyrus IMAP server, Dovecot, Exim, Kolab, OpenSMTPD, Postfix, Qmail, Sendmail (под двумя лицензиями), Zimbra и др.

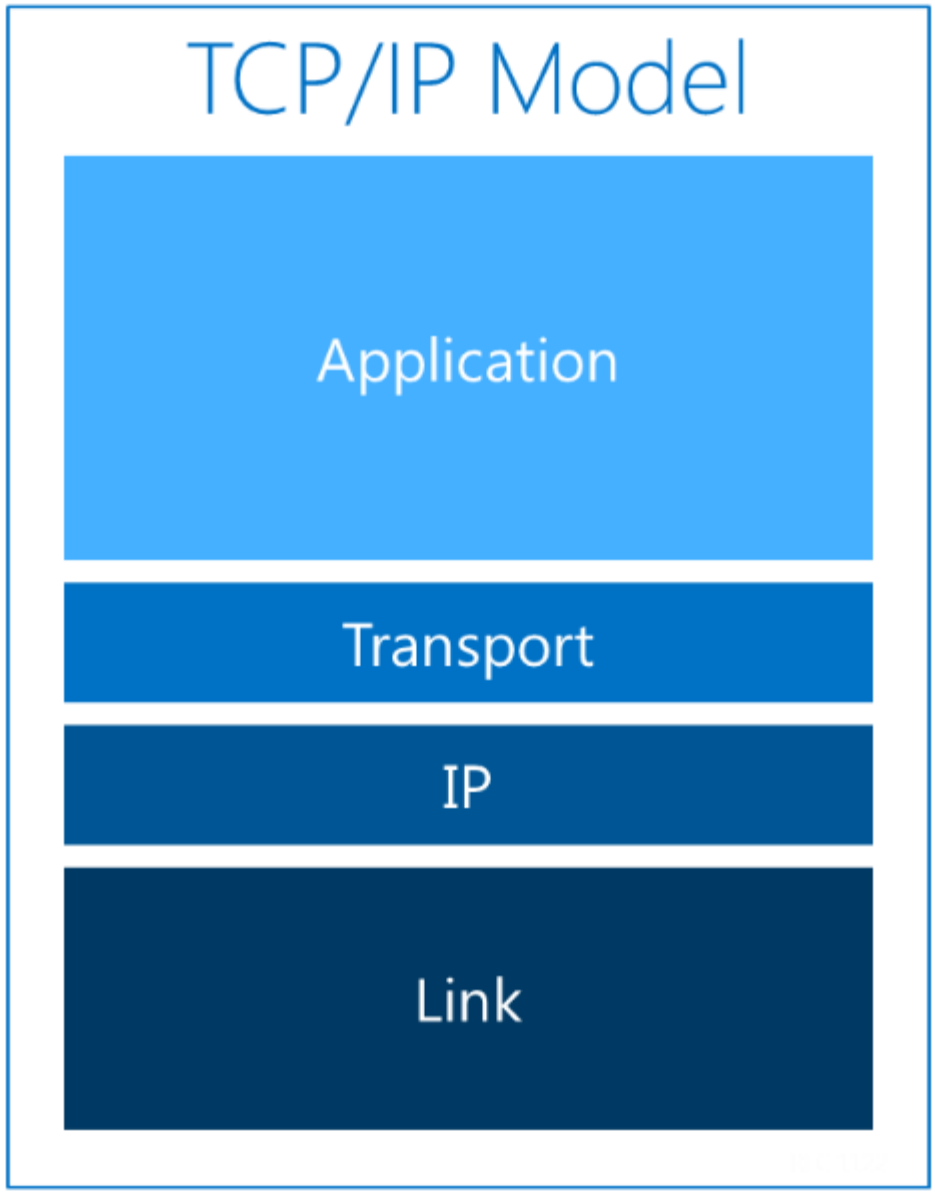
2. Протокол TCP-IP, HTTP

2.1 СТЕК ПРОТОКОЛОВ TCP/IP

Стек протоколов TCP/IP — набор сетевых протоколов передачи данных, используемых в сетях, включая сеть Интернет. Название TCP/IP происходит из двух наиважнейших протоколов семейства — Transmission Control Protocol (TCP) и Internet Protocol (IP), которые были разработаны и описаны первыми в данном стандарте.

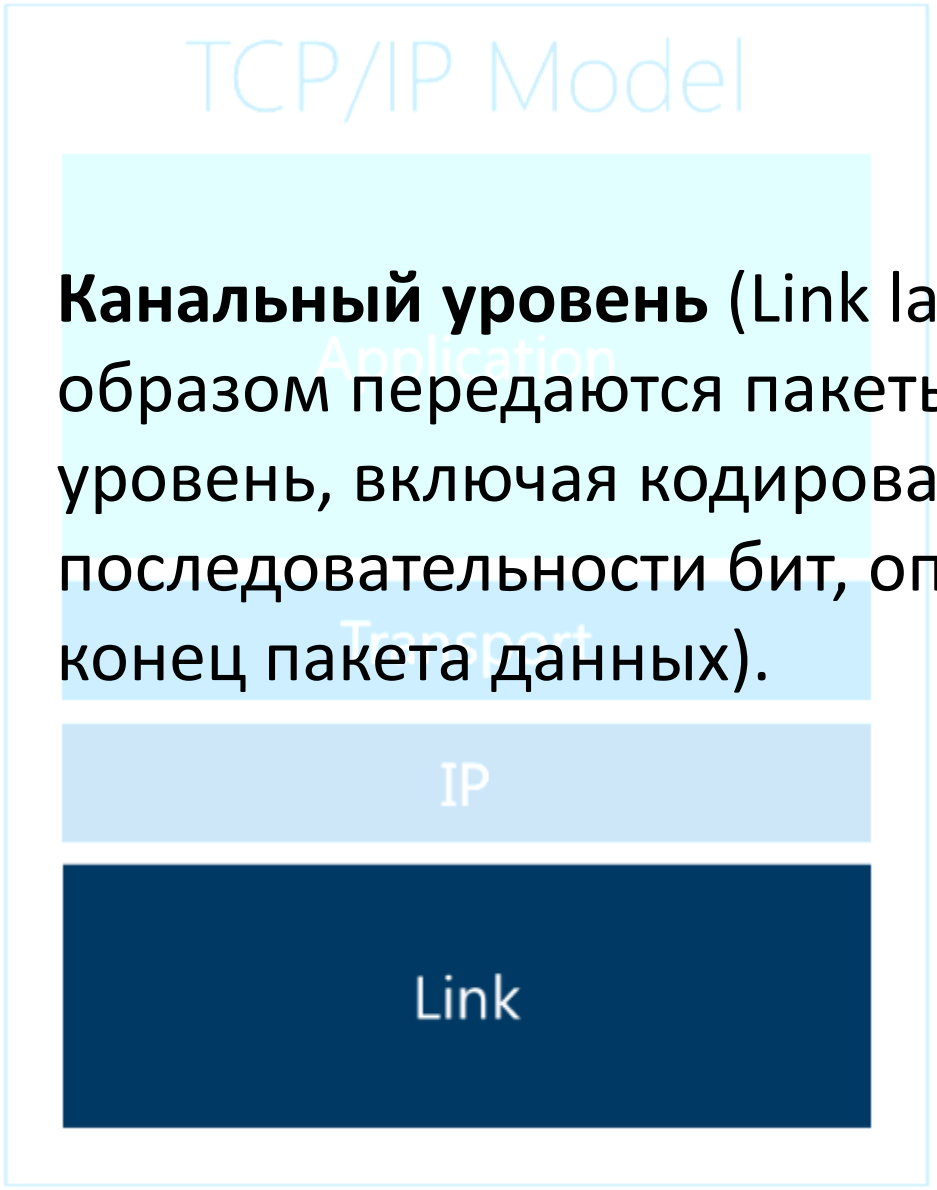
По сути стек протоколов TCP/IP – это набор правил по которым происходит обмен информации (реализуется модель коммутации пакетов).

2.1 СТЕК ПРОТОКОЛОВ TCP/IP



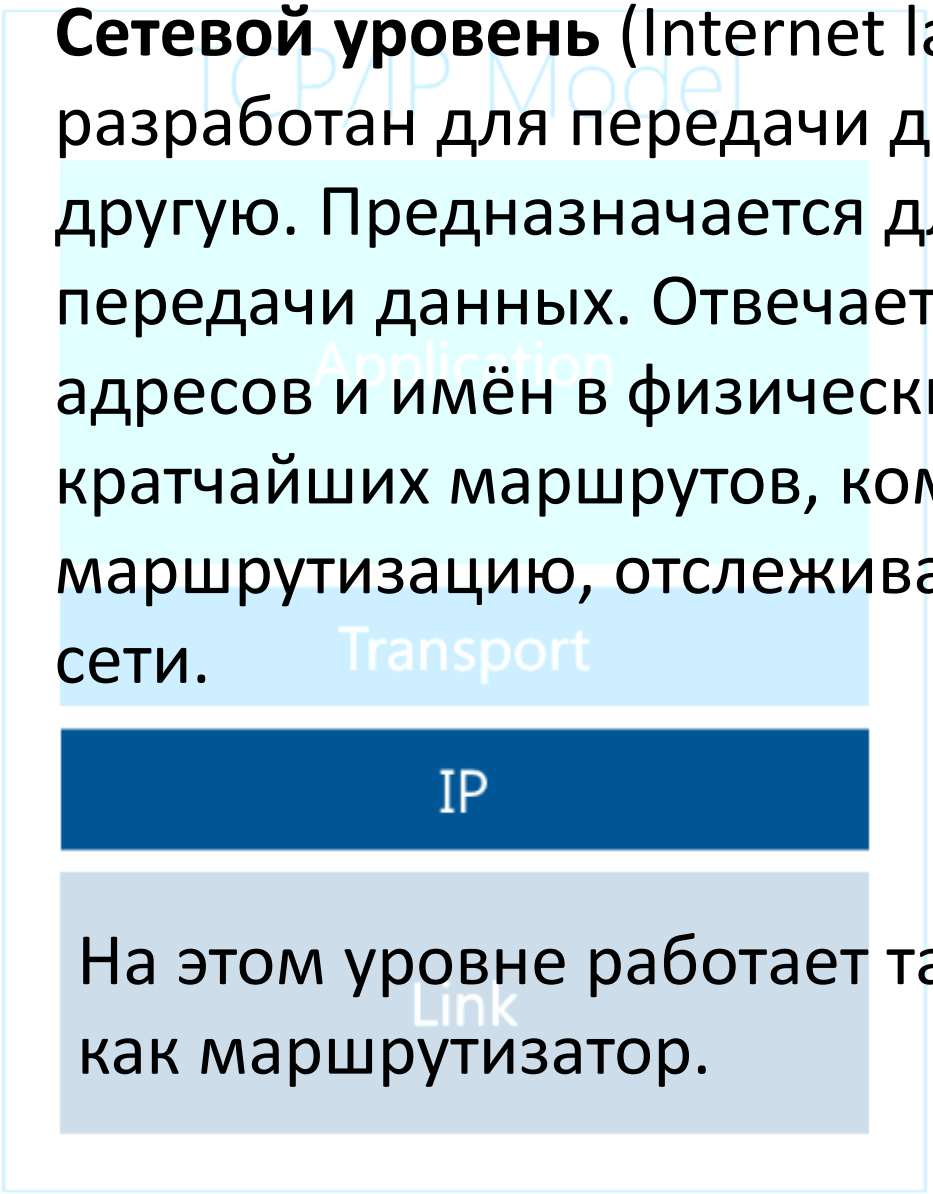
Уровни протоколов TCP/IP расположены по принципу стека — это означает, что протокол, располагающийся на уровне выше, работает «поверх» нижнего, используя механизмы инкапсуляции. Например, протокол TCP работает поверх протокола IP.

2.1 СТЕК ПРОТОКОЛОВ TCP/IP



Канальный уровень (Link layer) описывает, каким образом передаются пакеты данных через физический уровень, включая кодирование (то есть специальные последовательности бит, определяющих начало и конец пакета данных).

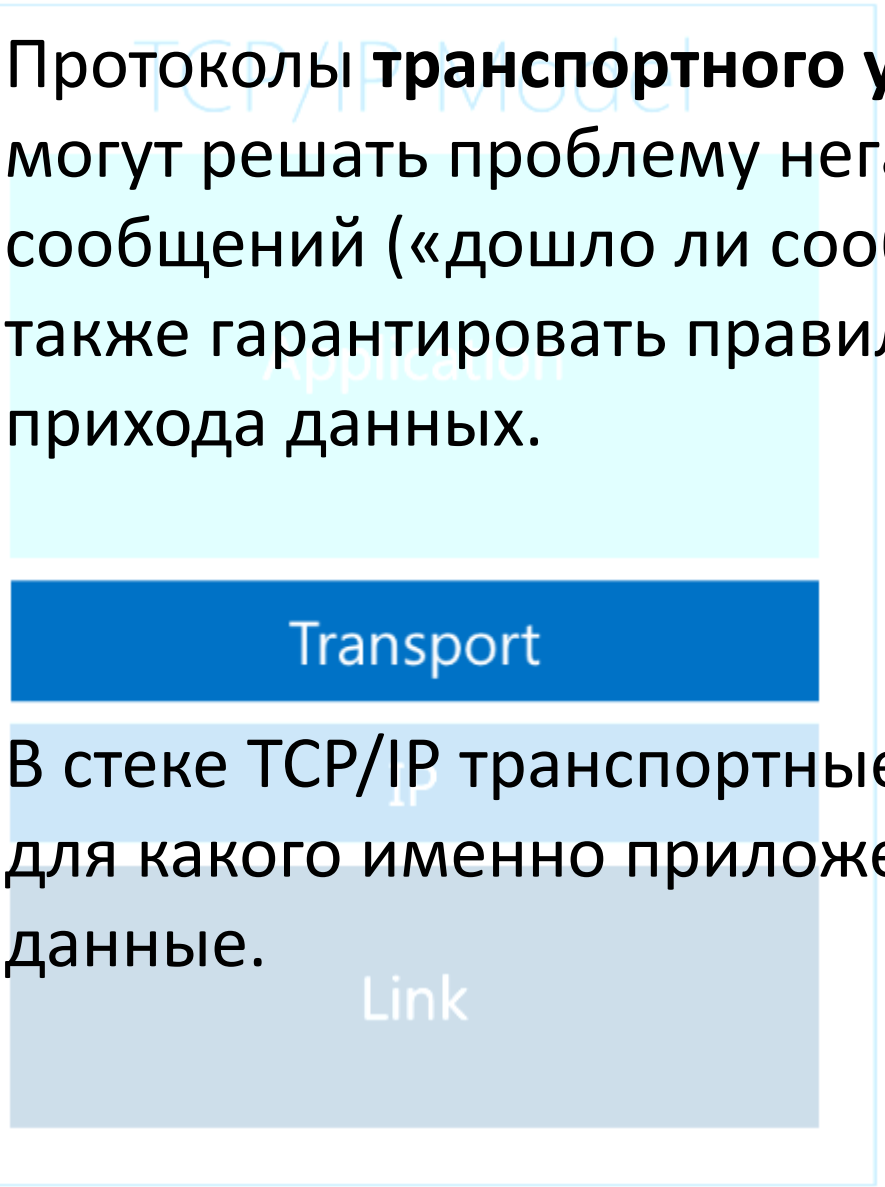
Сетевой уровень (Internet layer) изначально разработан для передачи данных из одной сети в другую. Предназначается для определения пути передачи данных. Отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание неполадок и заторов в сети.



На этом уровне работает такое сетевое устройство, как маршрутизатор.

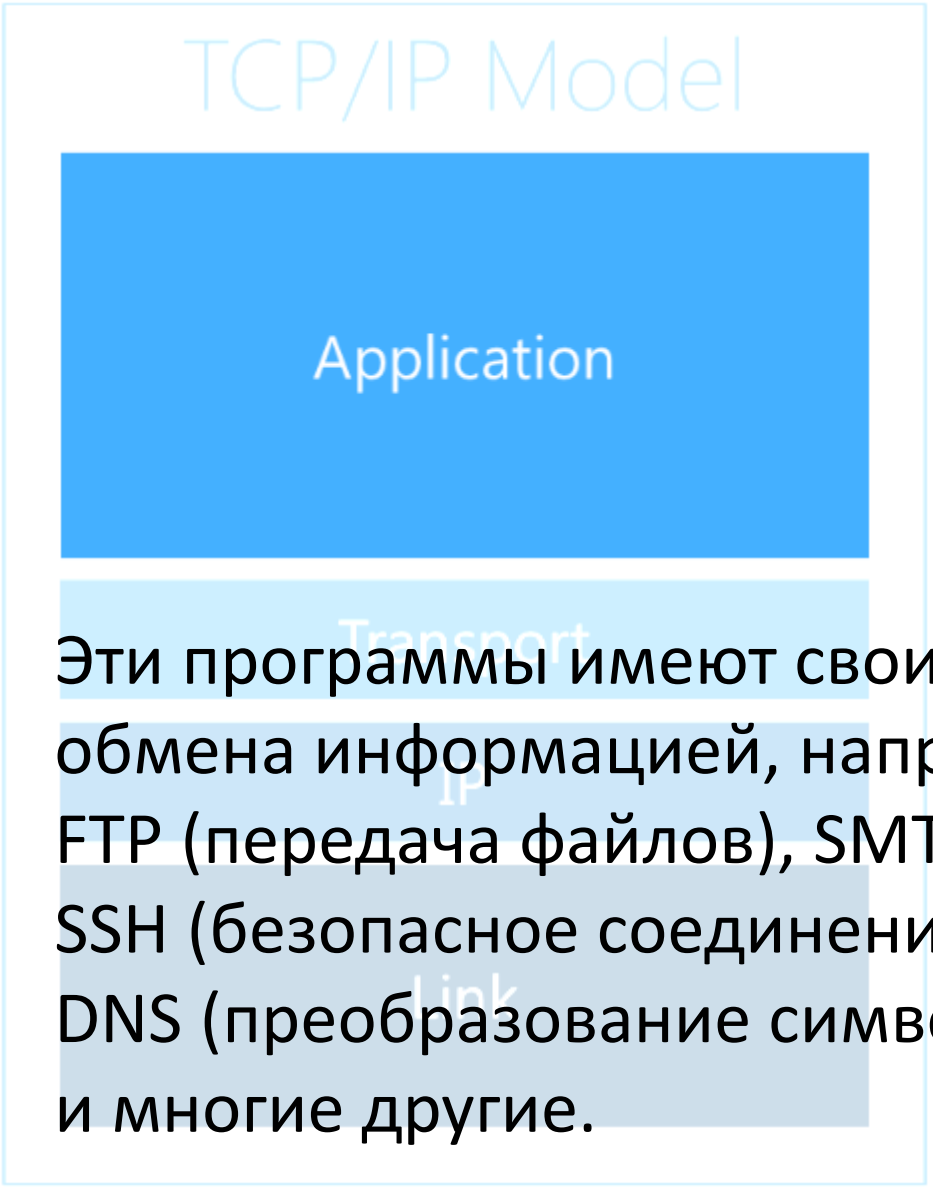
2.1 СТЕК ПРОТОКОЛОВ TCP/IP

Протоколы **транспортного уровня** (Transport layer) могут решать проблему негарантированной доставки сообщений («дошло ли сообщение до адресата?»), а также гарантировать правильную последовательность прихода данных.



В стеке TCP/IP транспортные протоколы определяют, для какого именно приложения предназначены эти данные.

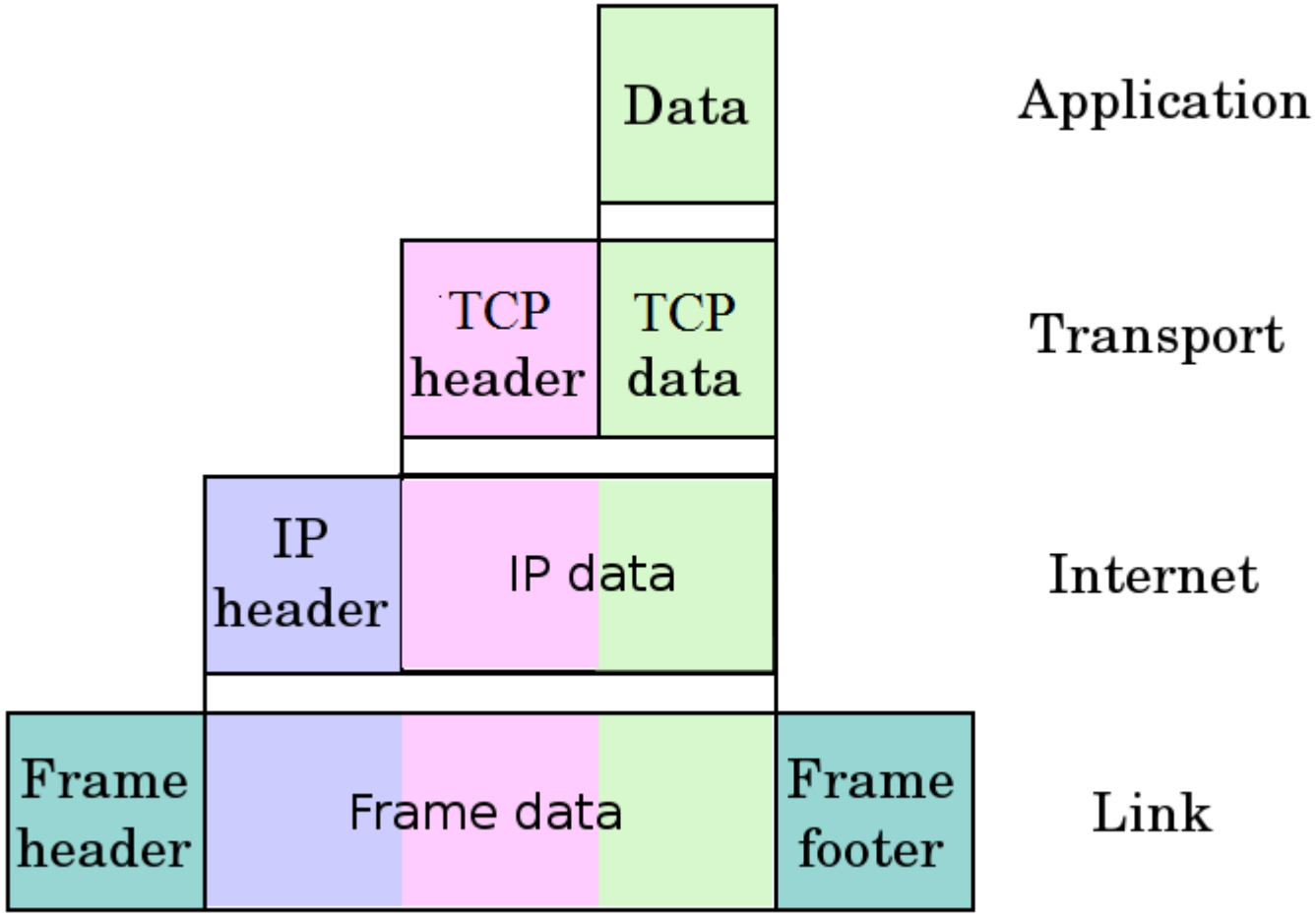
2.1 СТЕК ПРОТОКОЛОВ TCP/IP



На **прикладном уровне** (Application layer) работает большинство сетевых приложений.

Эти программы имеют свои собственные протоколы обмена информацией, например, HTTP для WWW, FTP (передача файлов), SMTP (электронная почта), SSH (безопасное соединение с удалённой машиной), DNS (преобразование символьных имён в IP-адреса) и многие другие.

Механизм инкапсуляции:



2.2 ПРОТОКОЛ HTTP

Что может передавать протокол HTTP:

- HTML, CSS, JS;
- Изображения, видео, музыку;
- Сообщения между программами и серверов;
- Может быть так же транспортом для других протоколов;
- Транспортировать любую информацию.

2.2 ПРОТОКОЛ HTTP

TCP соединение устанавливается через тройное рукопожатие:

- сначала запрос от клиента серверу;
- затем от сервера получаем подтверждение что мы готовы установить соединение;
- затем подтверждение от клиента серверу, после чего устанавливается соединение для передачи пакетов данных в обе стороны.

2.2 ПРОТОКОЛ HTTP

Пропускная способность сервера (время задержки), зависит от:

- пропускной способности канала сервера;
- производительности сервера;
- расстояние в пути и качество пути (тип части маршрута - у мобильной связи проблемы с потерянными пакетами) и количества обращений к нему.

2.2 ПРОТОКОЛ HTTP

На что тратит время HTTP запрос:

- DNS запрос;
- установка соединения;
- отправка данных;
- ожидание ответа;
- получение данных.

2.2 ПРОТОКОЛ HTTP

В чём достоинства HTTP:

- простота (текстовый прозрачный протокол);
- расширяемость (с помощью своих заголовков);
- распространённость.

Недостатки:

- размер;
- открытость (нет шифрования);
- нет механизма сохранения состояния (каждый http запрос не знает про предыдущие и последующие).

2.2 ПРОТОКОЛ HTTP

HTTP работает с **URL** (Уникальный локатор ресурса):

<схема>://<логин>:<пароль>@<хост>:<порт>/<URL путь>?<параметры>#<якоря>(якоря при запросе не отсылаются).

Не все символы допустимы URL, только: латинские буквы, спец символы, зарезервированные символы. Все неразрешенные символы декодируются в шестнадцатеричные числа с добавлением в начало знака %. Есть ещё Punicode (кодирование национальных доменов в специальный код).