

РОССИЙСКИЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ имени Г. В. ПЛЕХАНОВА



Динамические веб-системы в экономике

➔ **ТЕМА: Интернет-приложения**

➔ **АВТОР: Коваль П. Е.**

Интернет-приложение - приложение, в котором клиентом выступает браузер, а сервером - веб-сервер.

Определение. Приложение — это прикладной компьютерный сервис, который обладает набором определенных функций и является одним из компонентов программного обеспечения.

Проще говоря, это программа, которая выполняет некоторые действия, чтобы облегчить жизнь пользователю или решить ту или иную проблему.

Классификация многопользовательских технологий работы:

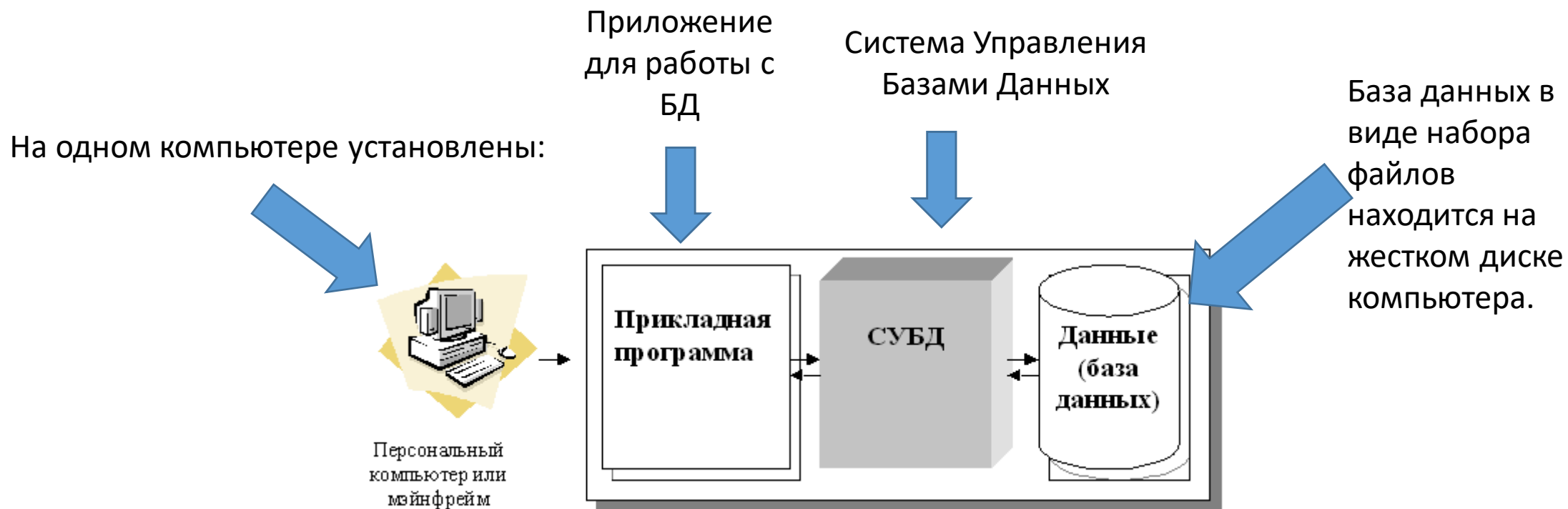
Централизованная архитектура.

Архитектура «файл-сервер» (технология с сетью и файловым сервером).

Технология «клиент – сервер».

Трехзвенная (многозвенная) архитектура «клиент – сервер».

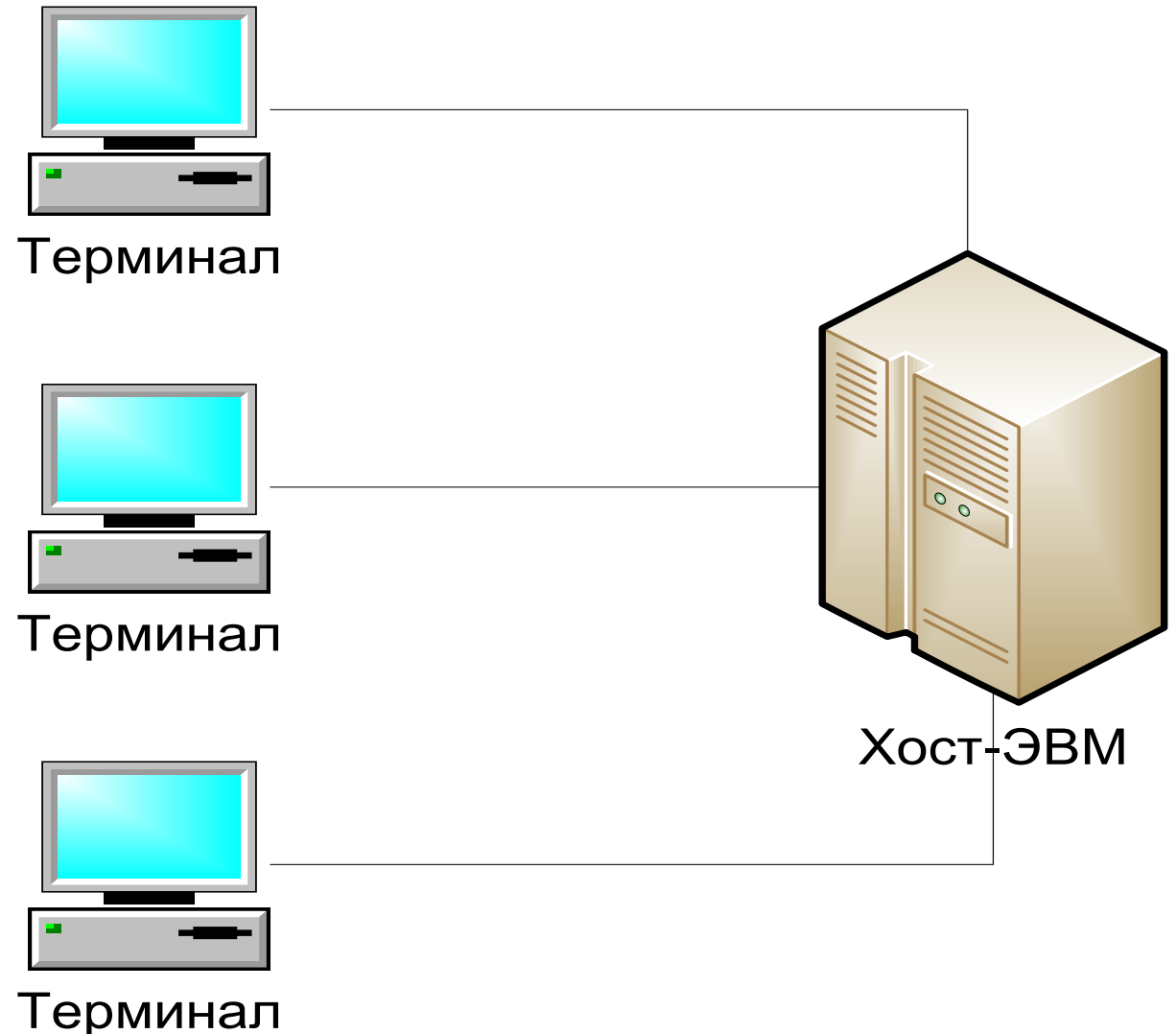
Централизованная архитектура.



СУБД – программный комплекс, предназначенный для создания, ведения и использования базы данных многими пользователями (прикладными программами).

Централизованная архитектура

- *Централизованная архитектура* вычислительных систем была распространена в 70-х и 80-х годах и реализовывалась на базе *мейнфреймов* (например, IBM-360/370 или их отечественных аналогов серии ЕС ЭВМ), либо на базе мини-ЭВМ (например, PDP-11 или их отечественного аналога СМ-4).
- Характерная особенность такой *архитектуры* – полная «неинтеллектуальность» *терминалов*. Их работой управляет хост-ЭВМ.



Централизованная архитектура

Достоинства

- пользователи совместно используют дорогие ресурсы ЭВМ и дорогие периферийные устройства;
- централизация ресурсов и оборудования облегчает обслуживание и эксплуатацию вычислительной системы;
- отсутствует необходимость администрирования рабочих мест пользователей;

Недостатки

Полностью зависит от администратора хост-ЭВМ. Пользователь не может настроить рабочую среду под свои потребности – все используемое программное обеспечение является коллективным.

Использование такой *архитектуры* является оправданным, если хост-ЭВМ очень дорогая, например, *супер-ЭВМ*.

Архитектура «файл-сервер»

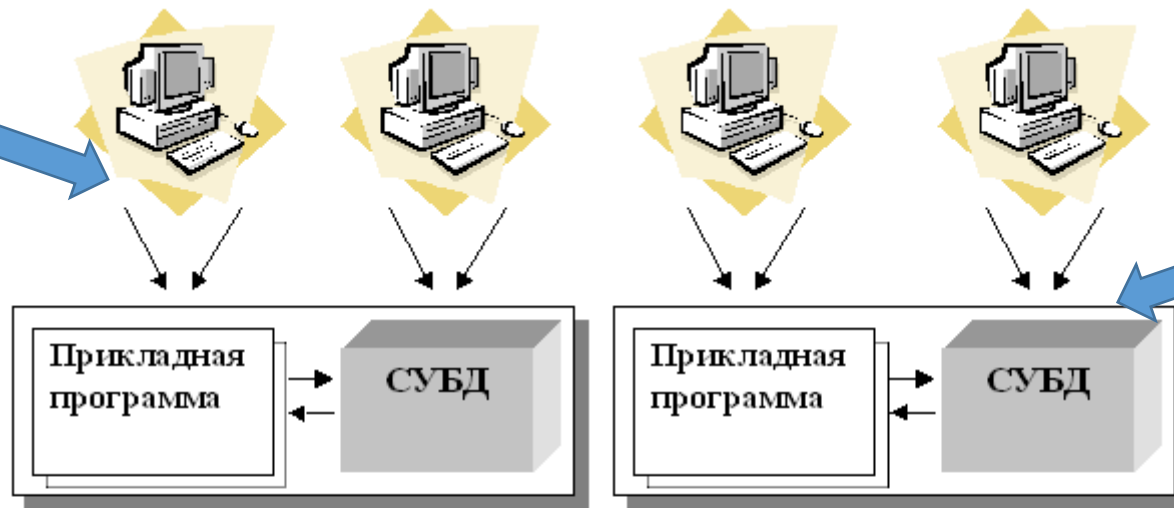
(архитектура баз данных с сетевым доступом)

Существует локальная сеть, состоящая из клиентских компьютеров,

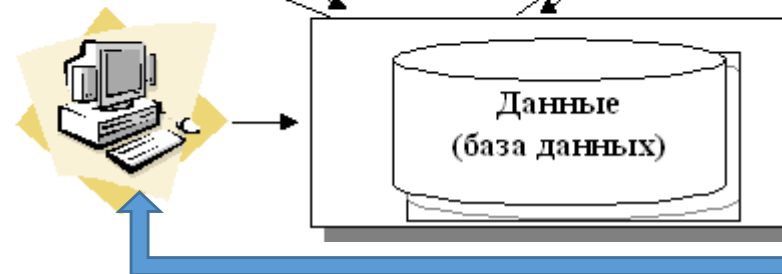
Клиентские компьютеры

на каждом из которых установлены :

СУБД и приложение для работы с БД.



Файловый сервер



База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (файлового сервера)

Архитектура «файл-сервер»

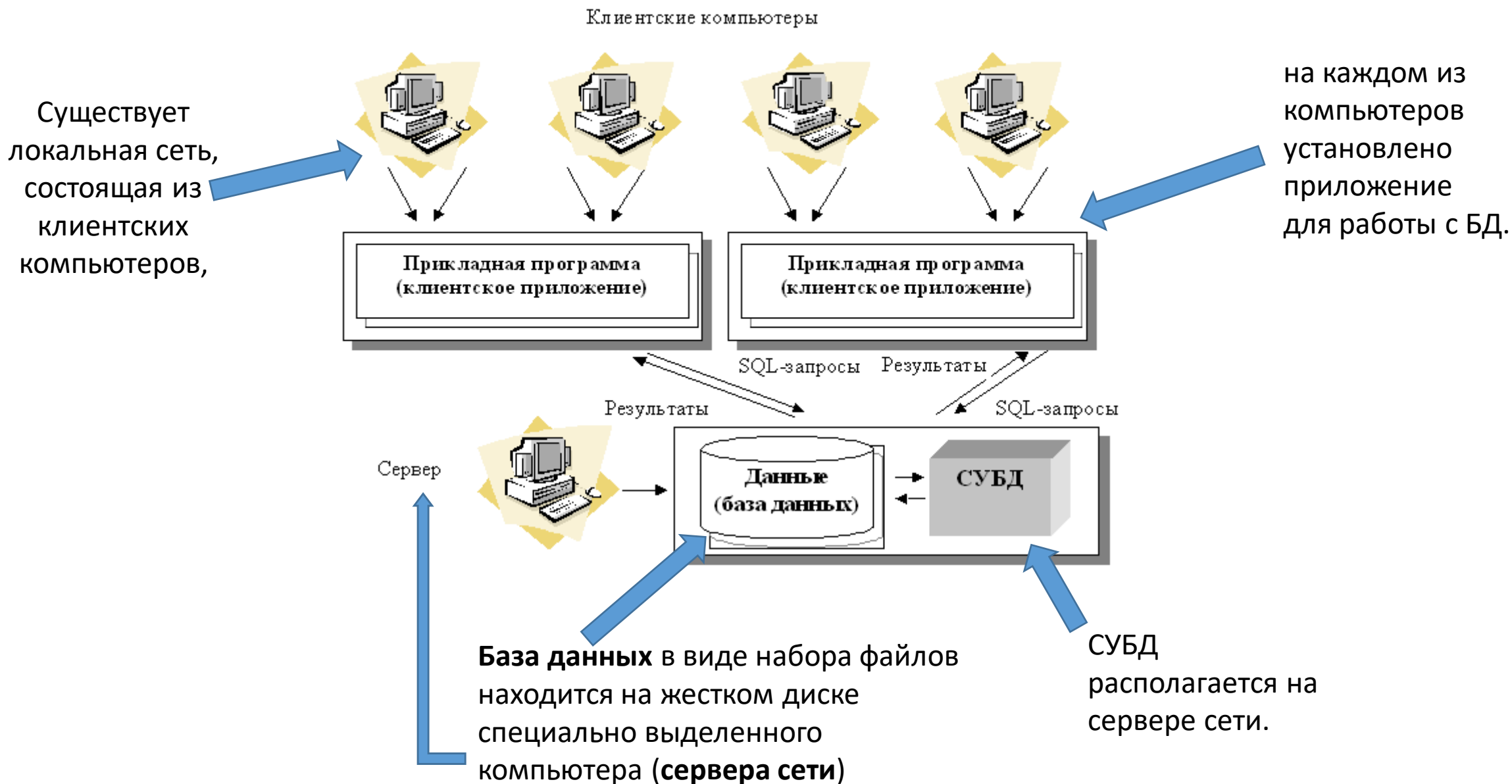
Достоинства

- возможность распределить функции вычислительной системы между несколькими независимыми компьютерами;
- все данные хранятся на защищенном сервере;
- поддержка многопользовательской работы;

Недостатки

- неработоспособность сервера может сделать неработоспособной всю вычислительную сеть;
- большой объем сетевого трафика
- на каждой рабочей станции должна находиться полная копия СУБД
- сложное администрирование;
- высокая стоимость оборудования;
- бизнес логика приложений осталась в клиентском ПО.

Архитектура «клиент – сервер»



Архитектура «клиент – сервер»

Терминология

- Независимо от того, как определяется понятие архитектуры "клиент-сервер", в основе этого понятия лежит **распределенная модель вычислений**. В самом общем случае под *клиентом* и *сервером* понимаются два взаимодействующих процесса, из которых один является поставщиком некоторого сервиса для другого.
- *Сервер* – логический процесс, который обеспечивает некоторый сервис по запросу от клиента. Обычно сервер не только выполняет запрос, но и управляет очередностью запросов, буферами обмена, извещает своих клиентов о выполнении запроса и т. д.
- *Клиент* – процесс, который запрашивает обслуживание от сервера. Процесс не является клиентом по каким-то параметрам своей структуры, он является клиентом только по отношению к серверу.
- При взаимодействии клиента и сервера инициатором диалога с сервером, как правило, является клиент, сервер сам не инициирует совместную работу. Это не исключает, однако, того, что сервер может извещать клиентов о каких-то зарегистрированных им событиях. Инициирование взаимодействия, запрос на обслуживание, восприятие результатов от сервера, обработка ошибок – это обязанности клиента.

Архитектура «клиент – сервер»

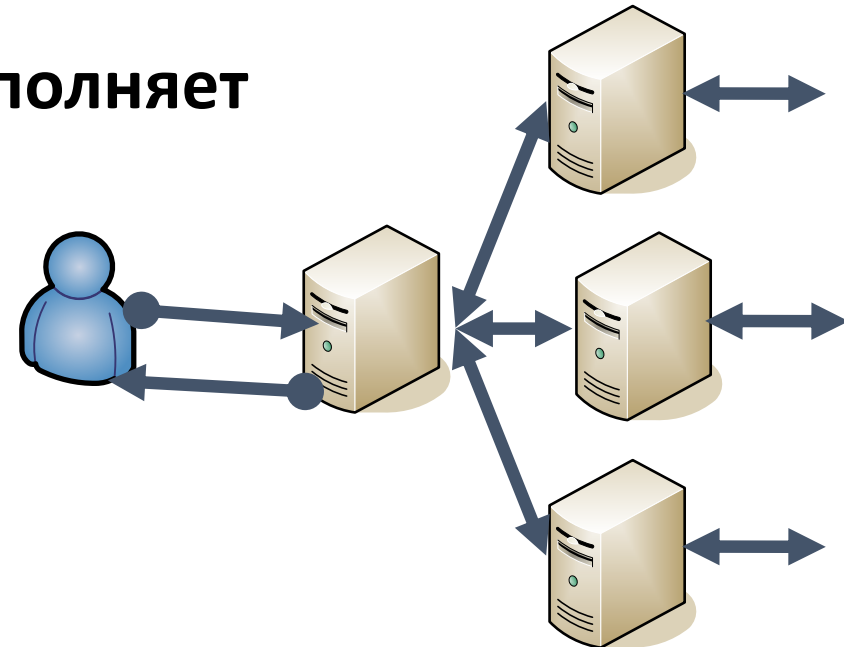
Услуга - набор действий, понимаемых как запрашиваемая

- чтение (получение) информации
- сохранение (запись) информации,
- пересылка информации и т.д.



Архитектура «клиент – сервер»

Технология клиент-сервер - взаимодействие, при котором одна программа (**клиент**) запрашивает **услугу** (выполнение какой-либо совокупности действий), а другая программа (**сервер**) ее выполняет



Архитектура «клиент – сервер»

Для работы пользователей сети необходимы:

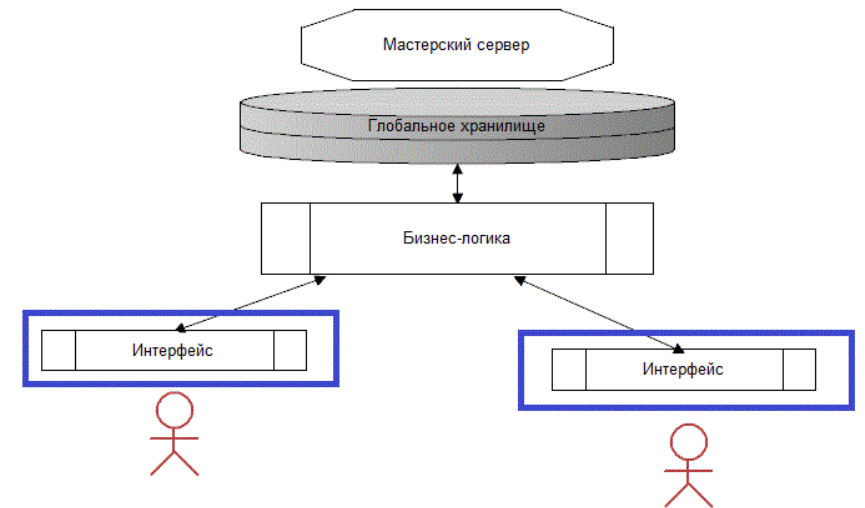
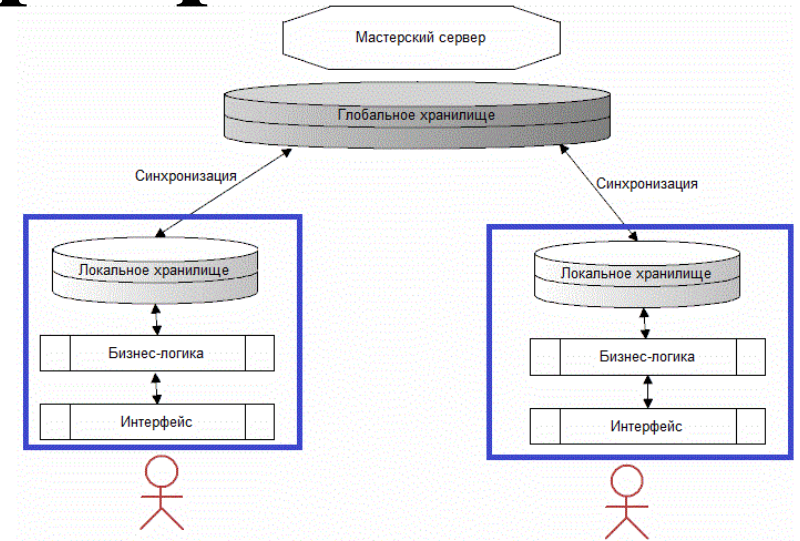
1. **Программа-клиент**, установленная на компьютере **пользователя**, которая может осуществлять **сетевой запрос**
2. **Программа-сервер**, установленная, на компьютере, где расположен информационный объект, которая может осуществлять по запросу **поиск и пересылку объекта**;
3. **Протоколы** (правила) **взаимодействия** между этими программами

Функции, выполняемые в среде “клиент/сервер”

Клиент	Сервер
Управляет пользовательским интерфейсом	Принимает и обрабатывает запросы к базе данных со стороны клиентов
Принимает и проверяет синтаксис введенного пользователем запроса	Проверяет полномочия пользователей
Выполняет приложение	Гарантирует соблюдение ограничений целостности
Генерирует запрос к базе данных и передает его серверу	Выполняет запросы/обновления и возвращает результаты клиенту
Отображает полученные данные пользователю	Поддерживает системный каталог Обеспечивает параллельный доступ к БД Обеспечивает управление восстановлением

Архитектура «клиент – сервер»

- ❑ **«Толстый клиент»:** на сервере реализованы главным образом функции доступа к базам данных, а основные прикладные вычисления выполняются на стороне клиента.
- ❑ **«Тонкий клиент»:** на сервере выполняется основная часть прикладной обработки данных, а на клиентские рабочие станции передаются уже результаты обработки данных для просмотра и анализа пользователем с возможностью их последующей обработки (в минимальном объёме).



Двухзвенная архитектура "клиент-сервер"

1. Архитектура "**толстый клиент – тонкий сервер**": большая часть функций приложения решалась клиентом, сервер занимался только обработкой SQL-запросов.

Архитектура "толстый" клиент имеет следующие недостатки:

- сложность администрирования;
- усложняется обновление ПО, поскольку его замену нужно производить одновременно по всей системе;
- перегружается сеть вследствие передачи по ней необработанных данных;
- слабая защита данных, поскольку сложно правильно распределить полномочия.

Двухзвенная архитектура "клиент-сервер"

2. Архитектура "**тонкий клиент – толстый сервер**": использование на сервере хранимых процедур (stored procedure - откомпилированные программы с внутренней логикой работы), привело к тенденции переносить все большую часть функций на сервер. Хранимые процедуры реализовывали часть бизнес-логики и гарантировали выполнение операции в рамках единой транзакции. Такое решение имеет очевидные преимущества, например его легче поддерживать, т. к. все изменения нужно вносить только в одном месте – на сервере.

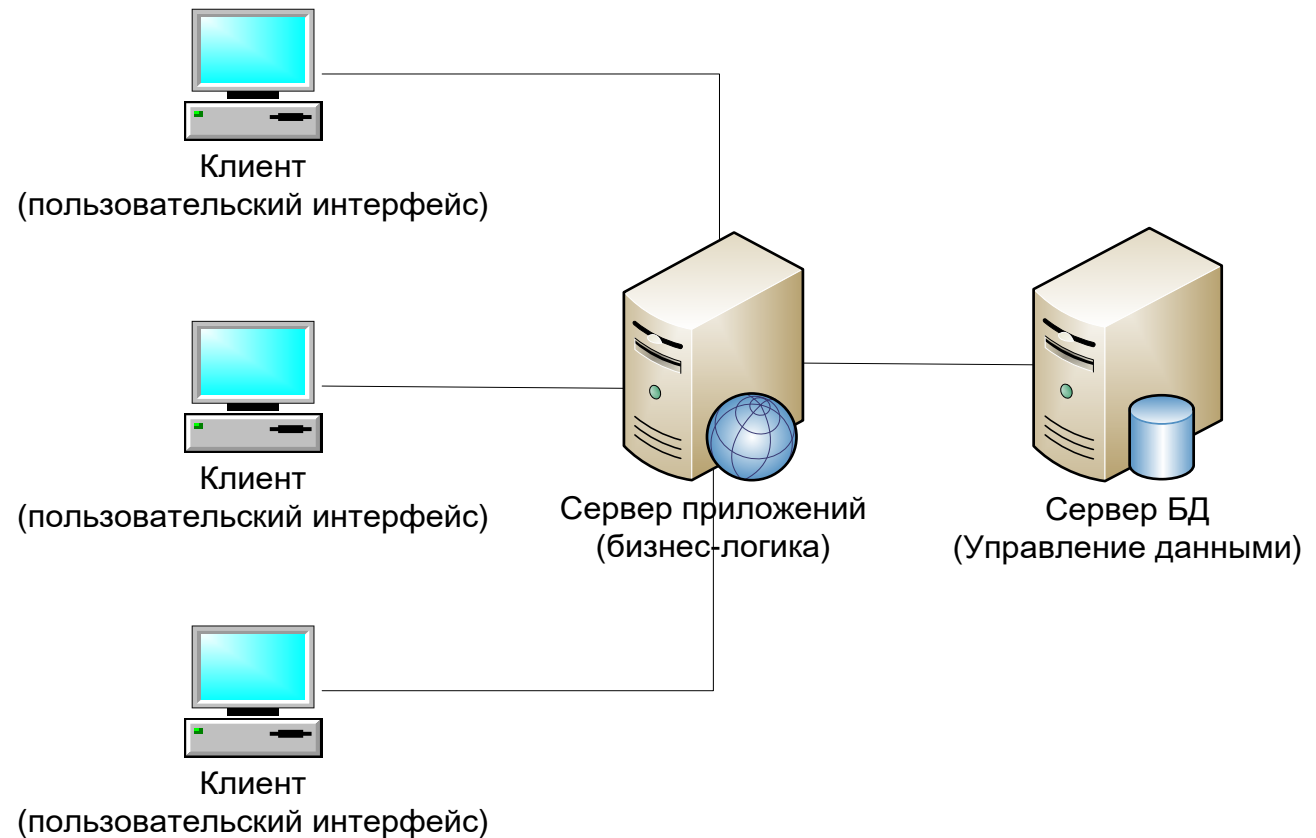
Архитектура "толстый" сервер имеет следующие недостатки:

- усложняется реализация, так как языки типа PL/SQL не приспособлены для разработки подобного ПО и нет хороших средств отладки;
- производительность программ, написанных на языках типа PL/SQL, значительно ниже, чем созданных на других языках, что имеет важное значение для сложных систем;
- программы, написанные на СУБД-языках, обычно работают недостаточно надежно; ошибка в них может привести к выходу из строя всего сервера баз данных;
- получившиеся таким образом программы полностью непереносимы на другие системы и платформы.

Для решения перечисленных проблем используются многоуровневые (три и более уровней) архитектуры клиент-сервер.

Многоуровневая архитектура «клиент-сервер»

Разновидность архитектуры клиент-сервер, в которой функция обработки данных вынесена на один или несколько отдельных серверов. Это позволяет разделить функции **хранения, обработки и представления данных** для более эффективного использования возможностей серверов и клиентов.



- Каждый уровень может быть потенциально запущен на отдельной машине
- Представление логика и данные разделены

Типовые функциональные компоненты многоуровневой архитектуры

Пользовательский интерфейс

- Средства представления (Presentation Services (PS))
- Логика представления (Presentation Logic (PL))

Бизнес-логика

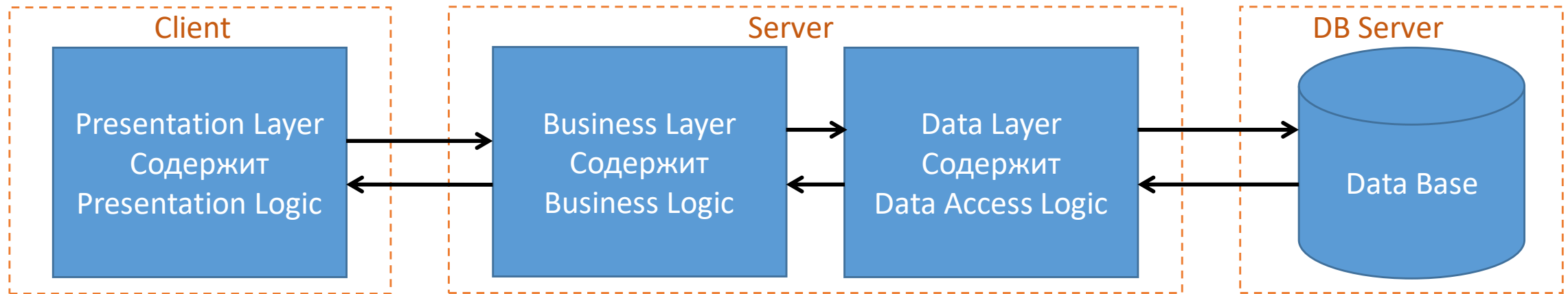
- Прикладная логика (Business or Application Logic (BL))
- Логика данных (Data Logic (DL))

Управление данными

- Средства управления БД (Data Services (DS))
- Средства управления файлами (File Services (FS))

Многоуровневая архитектура «клиент-сервер»

3-уровневая архитектура



Уровень представления

Статический или динамически сгенерированный контент отображаемый через браузер (front-end)

Уровень логики

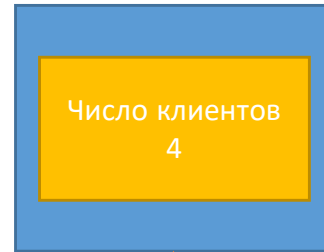
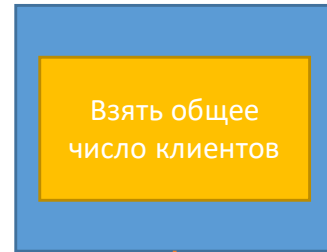
Уровень подготовки данных для динамически генерируемого контента, уровень сервера приложений (application server). Middleware платформы: JavaEE, ASP.NET, PHP и т. д.

Уровень данных

База данных включающая в себя данные и систему управления над ними или же готовая RDBMS система, предоставляющая доступ к данным и методы управления (back-end)

Многоуровневая архитектура клиент-сервер

Уровень представления



Уровень логики

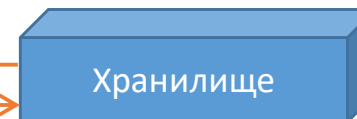
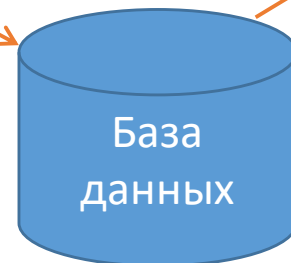
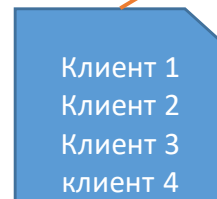
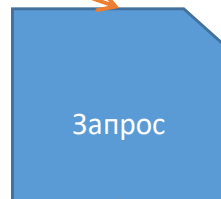


Взять список клиентов
последний год



Сложить всех
клиентов вместе

Уровень данных

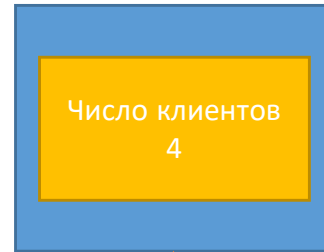
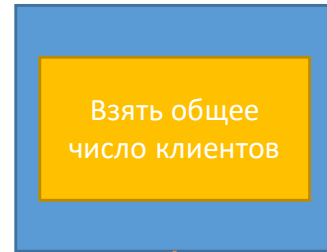


Принципы архитектуры:

- Клиент-серверная архитектура
- Каждый слой (данные, представление и логика) не зависит от остальных и не зависит от реализации
- Несоединённые слои вообще никогда не взаимодействуют
- Изменение платформы влияет только на тот уровень который на ней находится

Многоуровневая архитектура клиент-сервер

Уровень представления



- Предоставляет графический интерфейс
- Обработывает пользовательские события
- Иногда называют GUI или client view of front-end

Уровень логики

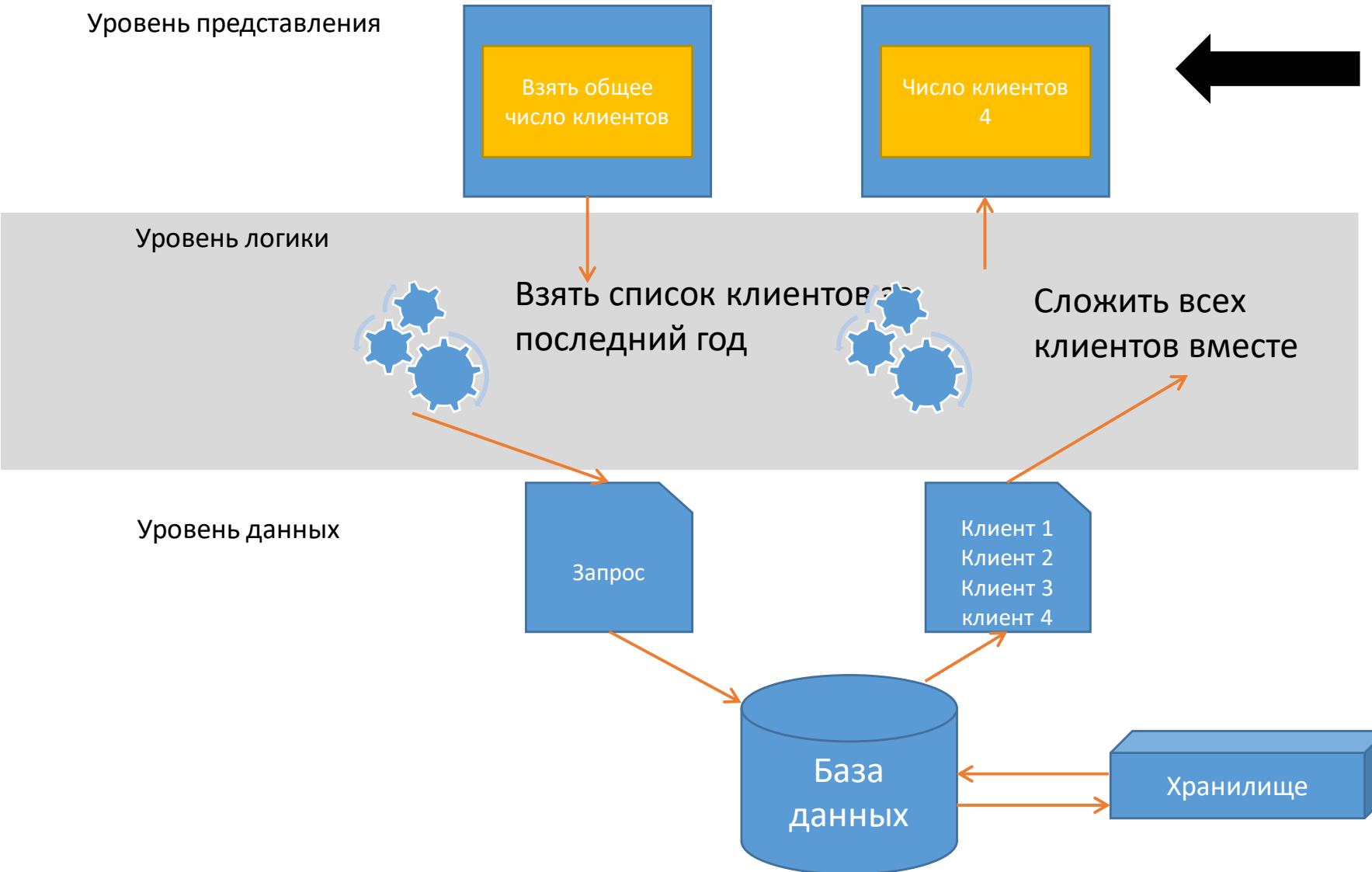
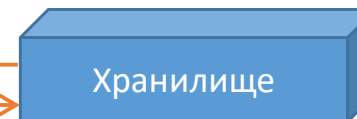
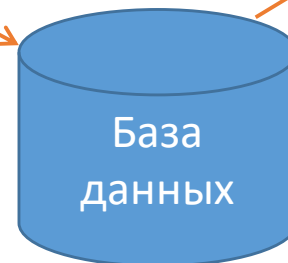
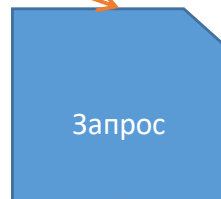


Взять список клиентов
последний год



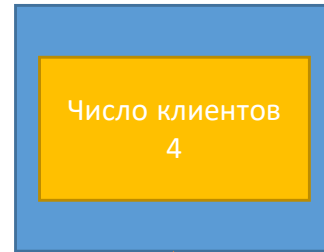
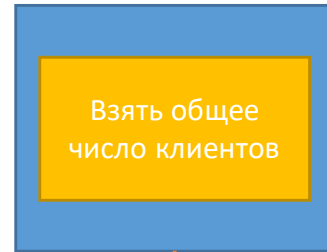
Сложить всех
клиентов вместе

Уровень данных



Многоуровневая архитектура клиент-сервер

Уровень представления



Уровень логики



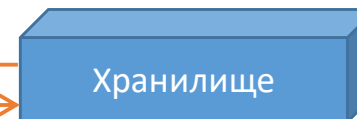
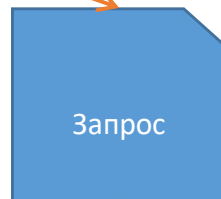
Взять список клиентов
последний год



Сложить всех
клиентов вместе



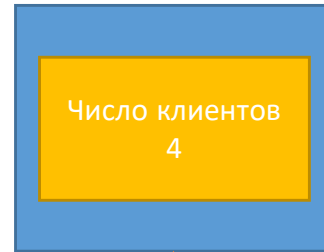
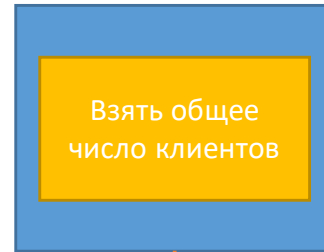
Уровень данных



- Набор правил для работы с данными
- Может обрабатывать запросы нескольких пользователей
Иногда называют middleware или back-end
- Не должен содержать пользовательских форм или непосредственно обращаться к данным

Многоуровневая архитектура клиент-сервер

Уровень представления



Уровень логики

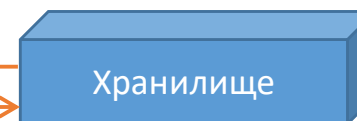
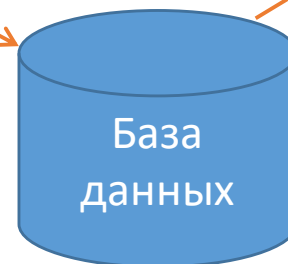
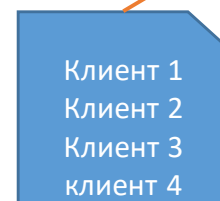
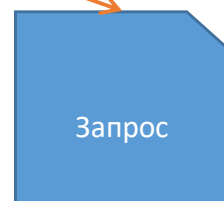


Взять список клиентов
последний год



Сложить всех
клиентов вместе

Уровень данных



- Физическое хранилище для данных (persistence)
- Управляет доступом к БД или файловой системе
- Иногда называется back-end
- Не должен содержать пользовательских форм или бизнес логики



Многоуровневая архитектура клиент-сервер

Достоинства

- Независимость уровней
- Лёгкость в поддержке
- Отдельные компоненты можно использовать в других задачах
- Задача разработки хорошо делится и поэтому может быть быстрее решена (уровни можно разрабатывать параллельно)
 - Web дизайнер делает уровень представления
 - Инженер (Software Engineer) делает логику
 - Администратор БД делает модель данных
- клиентское ПО не нуждается в администрировании;
- масштабируемость;
- конфигурируемость;
- достаточно высокая безопасность и надежность;
- низкие требования к скорости канала между терминалами и сервером приложений;
- низкие требования к производительности и техническим характеристикам терминалов

Недостатки

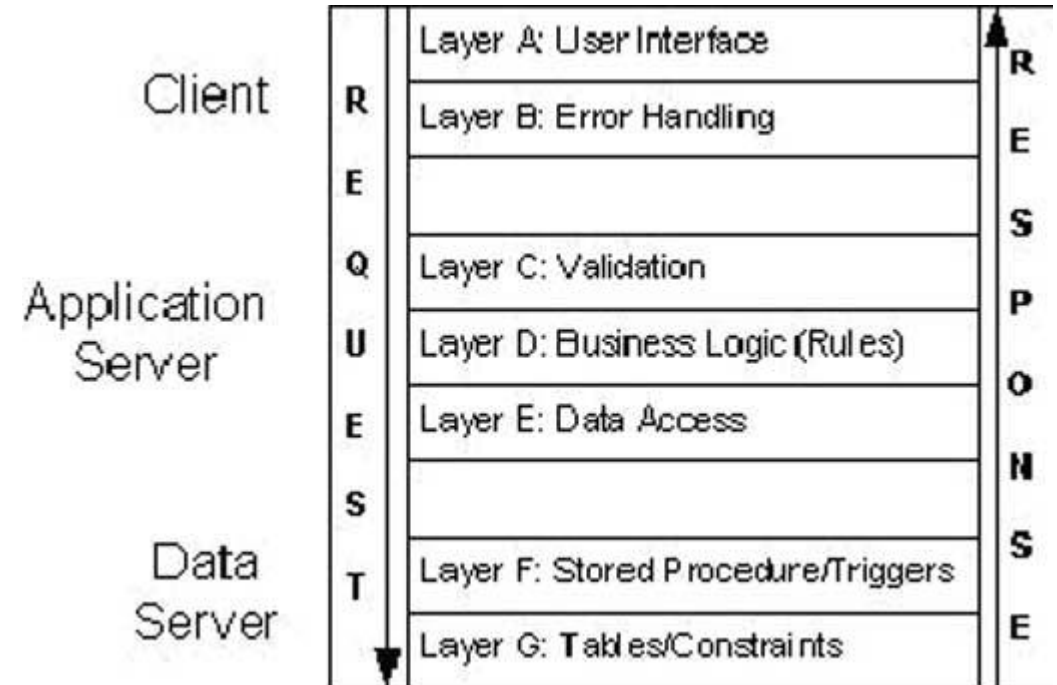
- сложность администрирования и обслуживания;
- более высокая сложность создания приложений;
- высокие требования к производительности серверов приложений и сервера базы данных;
- высокие требования к скорости канала (сети) между сервером базы данных и серверами приложений.

Многоуровневая архитектура клиент-сервер

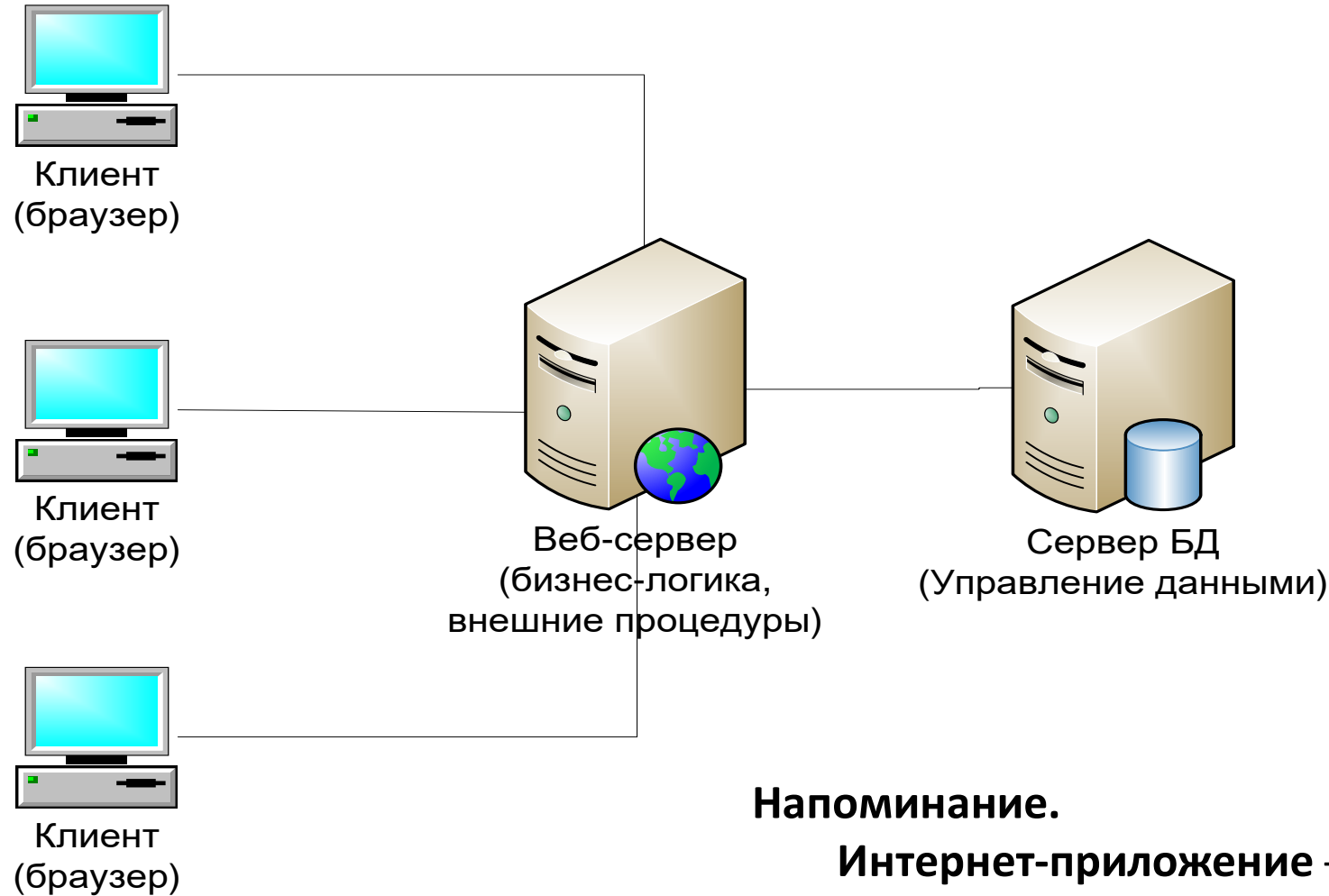
Уровни против слоев

Основное отличие заключается в том, что уровни находятся физическом уровне, а слои на логическом. Иными словами уровень, теоретически, может быть развернут независимо на отдельном компьютере, а слой логическое разделение внутри уровня. Типичная трехуровневая модель, содержит, по меньшей мере семь слоев, разделенных на всех трех уровнях.

Главное, что нужно помнить о многоуровневой архитектуре является то, что запросы и ответы каждого потока в одном направлении проходят по всем слоям, и что слои никогда не может быть пропущены. Таким образом, в модели, показанной на рисунке справа, единственный слой, который может обратиться к слою "E" (слой доступа к данным) является слой "D" (слой правил). Аналогичным образом слой "C" (прикладной слой ратификации) может только отвечать на запросы из слоя "B" (слоя обработка ошибок).



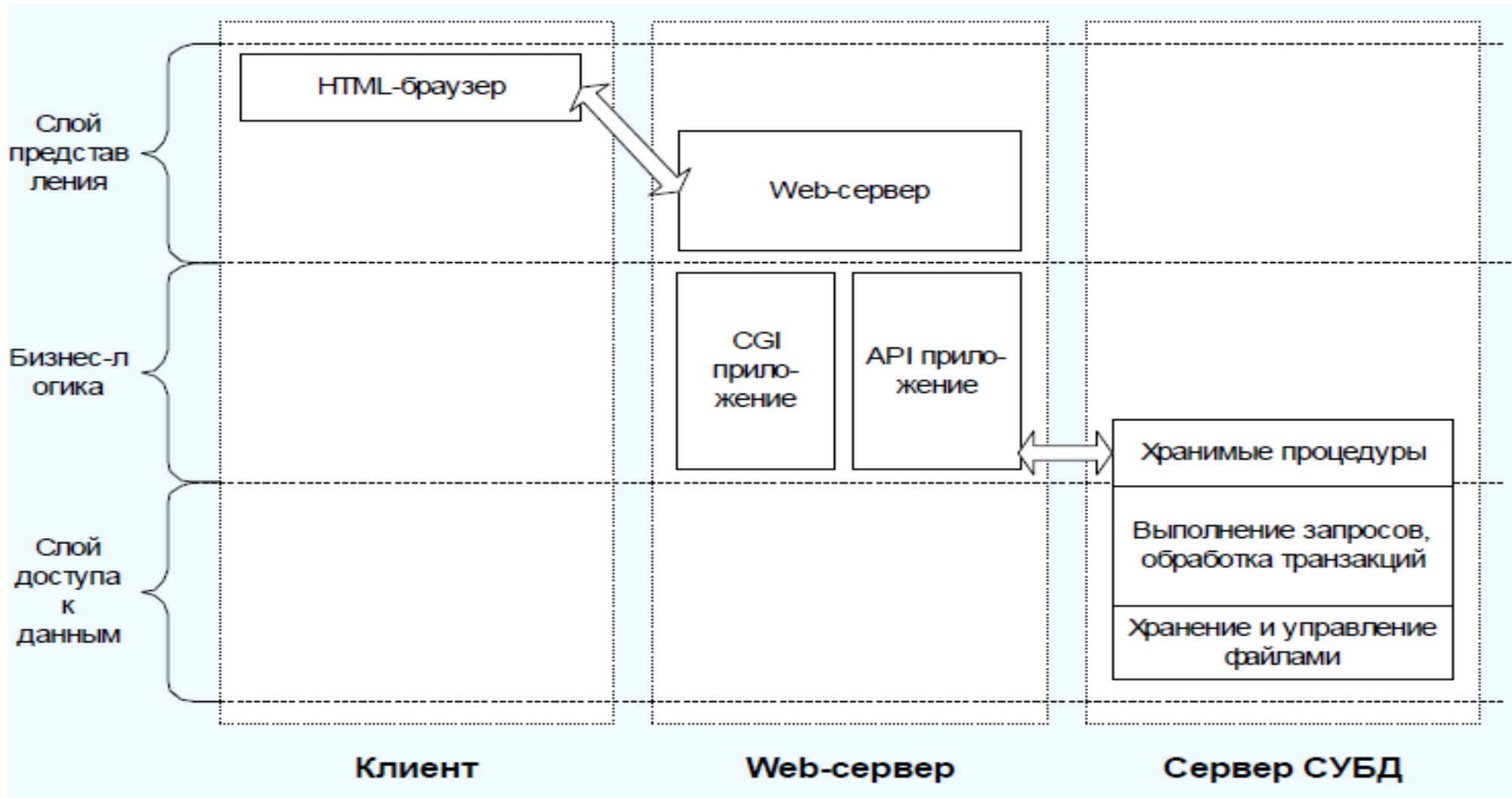
Архитектура Интернет-приложений.



Напоминание.

Интернет-приложение - приложение, в котором клиентом выступает браузер, а сервером - веб-сервер.

Архитектура Интернет-приложений.

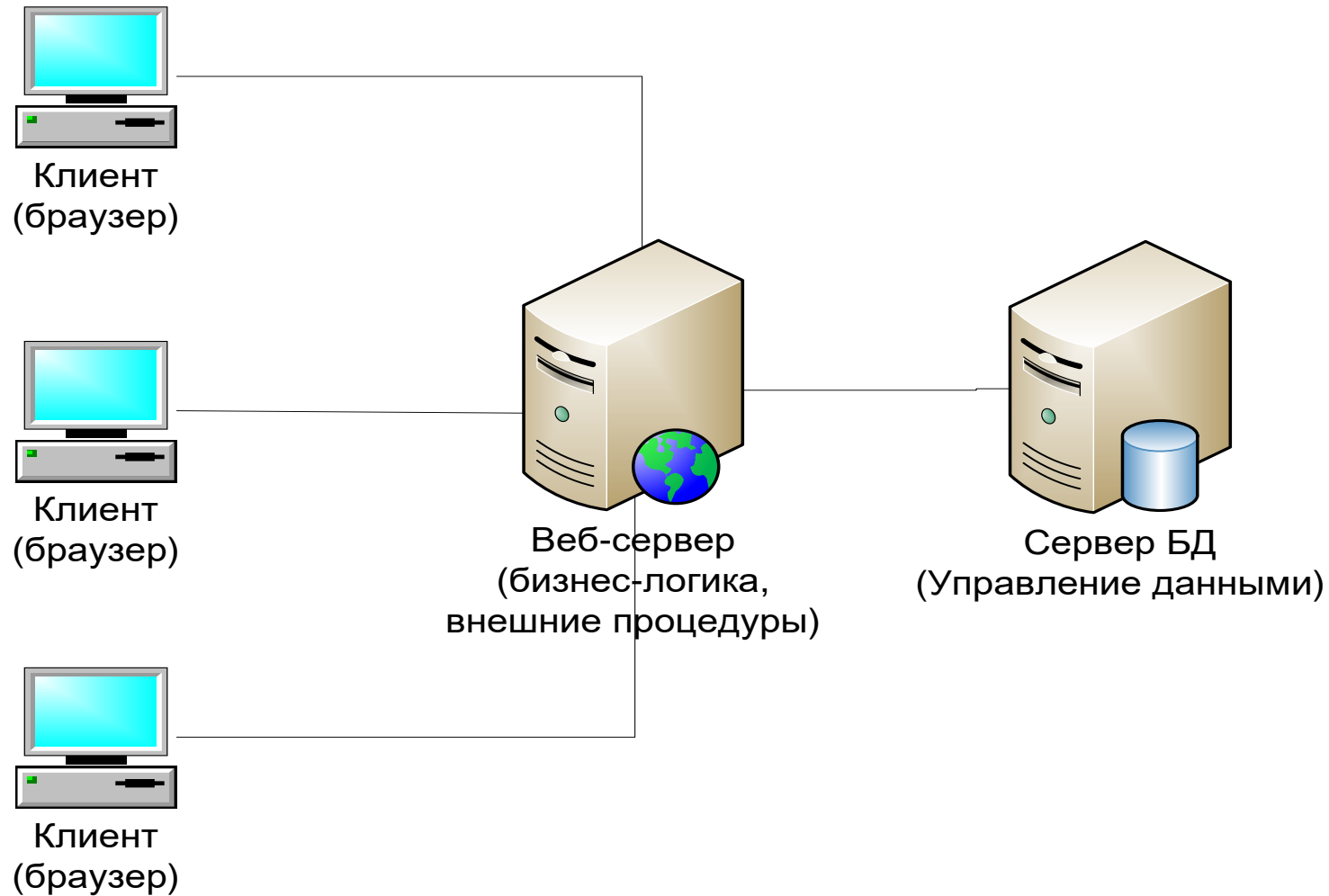


Архитектура Интернет-приложений.

Особенности

- Отсутствие необходимости использовать дополнительное ПО на стороне клиента
- Возможность подключения практически неограниченного количества клиентов
- Централизованное место хранения данных
- Недоступность при отсутствии работоспособности сервера или каналов связи
- Достаточно низкая скорость Веб-сервера и каналов передачи данных

Архитектура Интернет-приложений.

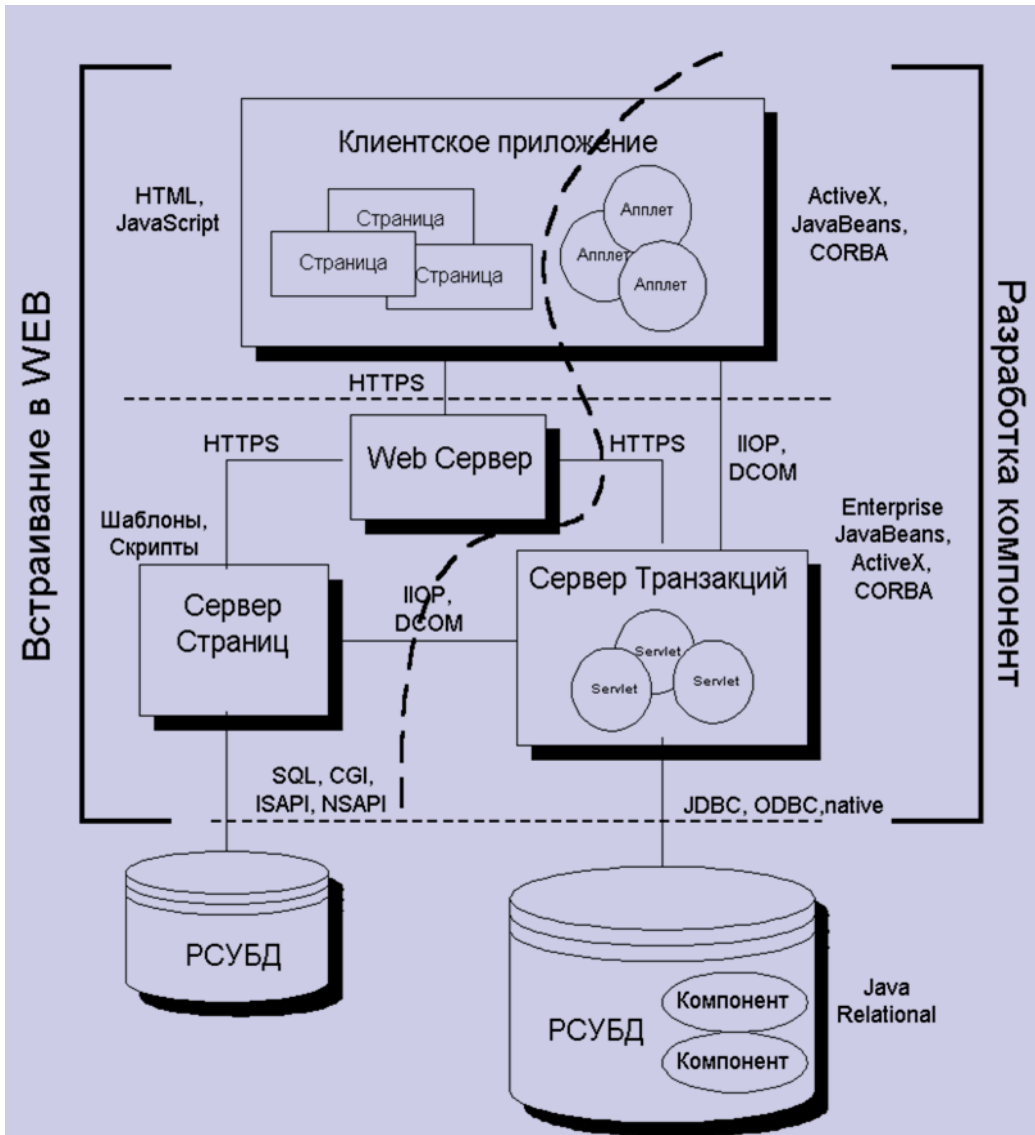


Результатом работы **веб-приложения** является **веб-страница**, отображаемая в окне браузера.

При этом само **веб-приложение** может выполняться как на **компьютере клиента**, так и на **компьютере сервера**.

Такие приложения традиционно называют **клиентские** и **серверные сценарии**

Архитектура Интернет-приложений.



Основные задачи

клиентских и серверных сценариев

- Клиентский сценарий выполняется на компьютере пользователя в процессе взаимодействия с Web-страницей и позволяет решать следующие задачи:
 1. верифицировать значения элементов управления формы;
 2. реализовать событийные процедуры для элементов управления.
- Серверный сценарий выполняется на Web-сервере до передачи страницы пользователю и позволяет:
 1. обеспечить доступ к базе данных и возврат данных пользователю;
 2. хранить информацию о состоянии пользователя или сеанса.

Архитектура Интернет-приложений.

Клиентские сценарии

- Клиентский сценарий выполняется на компьютере-клиенте. Программы просмотра снабжены встроенным интерпретатором, который может считывать и выполнять сценарии.
- Основная цель добавления клиентского сценария к Web-странице — создание событийных процедур для элементов управления. Например, событийная процедура будет запускать определенную функцию, когда пользователь нажмет соответствующую кнопку.
- Клиентские сценарии в HTML-странице не компилируются и не шифруются. Поэтому, если посмотреть исходный HTML-код Web-страницы, можно увидеть текст встроенного сценария.
- Чтобы сценарий клиентской части функционировал, программа просмотра должна поддерживать язык, на котором он написан. В противном случае пользователь не получит полного доступа к сценарным средствам Web-страницы.

К ним предъявляется одно общее требование: *эти программы должны быть лишены возможности обращаться к ресурсам компьютера, на котором они выполняются.*

Клиентские сценарии

Назначение сценариев выполняемых клиентом

- Разработка интерактивных HTML-документов стандарта DHTML (*Dinamic HTML*), элементы оформления и даже содержание которых, меняются в зависимости от действий пользователя (*без обмена данными с сервером (об этом отдельно) !*);
- Разработка документов содержащих анимационные элементы (*и даже Web-страниц с играми*);
- Настройка внешнего вида документа под параметры конкретного рабочего места клиента (*определение типа и версии броузера и экранных параметров*);
- Предварительная обработка данных из форм перед их отправкой на сервер (*обидно - заполнить большую форму, потом ожидать завершения транзакции с БД, а в результате получить ответ сценария сервера типа - НЕВЕРНАЯ ДАТА*);

Средства разработки сценариев

При разработке сценариев интерактивного управления используются :

JavaScript - язык разработки сценариев интерактивного управления для Web-страниц.

Java-апплет - это программа, написанная на языке **Java** и откомпилированная в байт-код.

JavaScript

JavaScript - язык разработки сценариев интерактивного управления для Web-страниц, разработанный фирмой Netscape. Поддерживается всеми современными браузерами

Код JavaScript располагается:

- **внутри документа HTML (раздел HEAD или BODY)**

Скрипт оформляется так:

```
<script type="text/javascript">  
    document.write("Hi !!!");  
</script>
```

- **вне документа HTML**

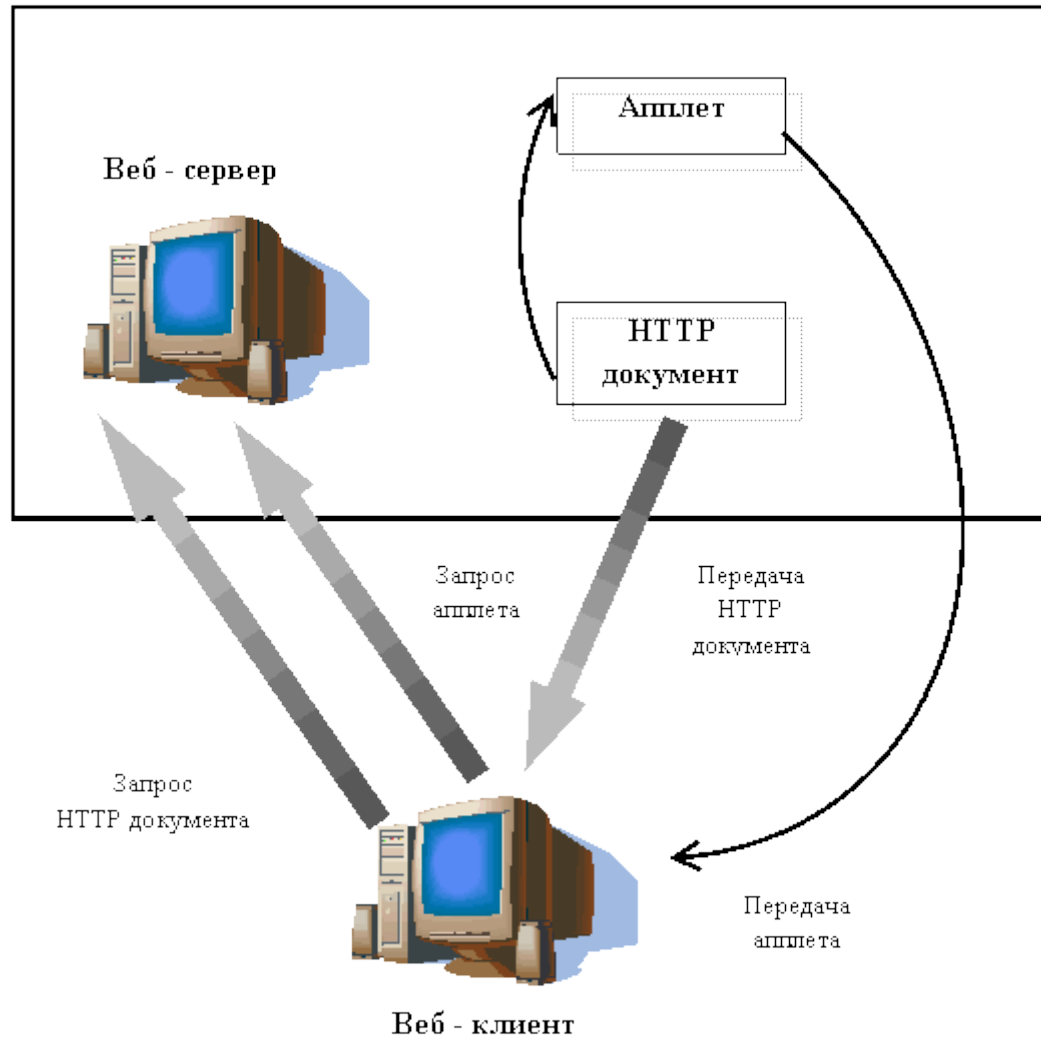
Подключают отдельный файл с текстом скриптов в разделе HEAD:

```
<script src="имя файла.js"></script>
```

Для подключения несколько файлов со скриптами используются несколько таких тегов:

```
<script src="script1.js"></script>  
<script src="/js/script2.js"></script>
```

Насыщенное интернет-приложение



Насыщенное интернет-приложение (Rich Internet application) - еще один подход, который заключается в использовании специальных сценариев, загружаемых с сервера

Назначение: для полной или частичной реализации пользовательского интерфейса.

Большинство браузеров поддерживает эти технологии (как правило, с помощью *плагинов*).

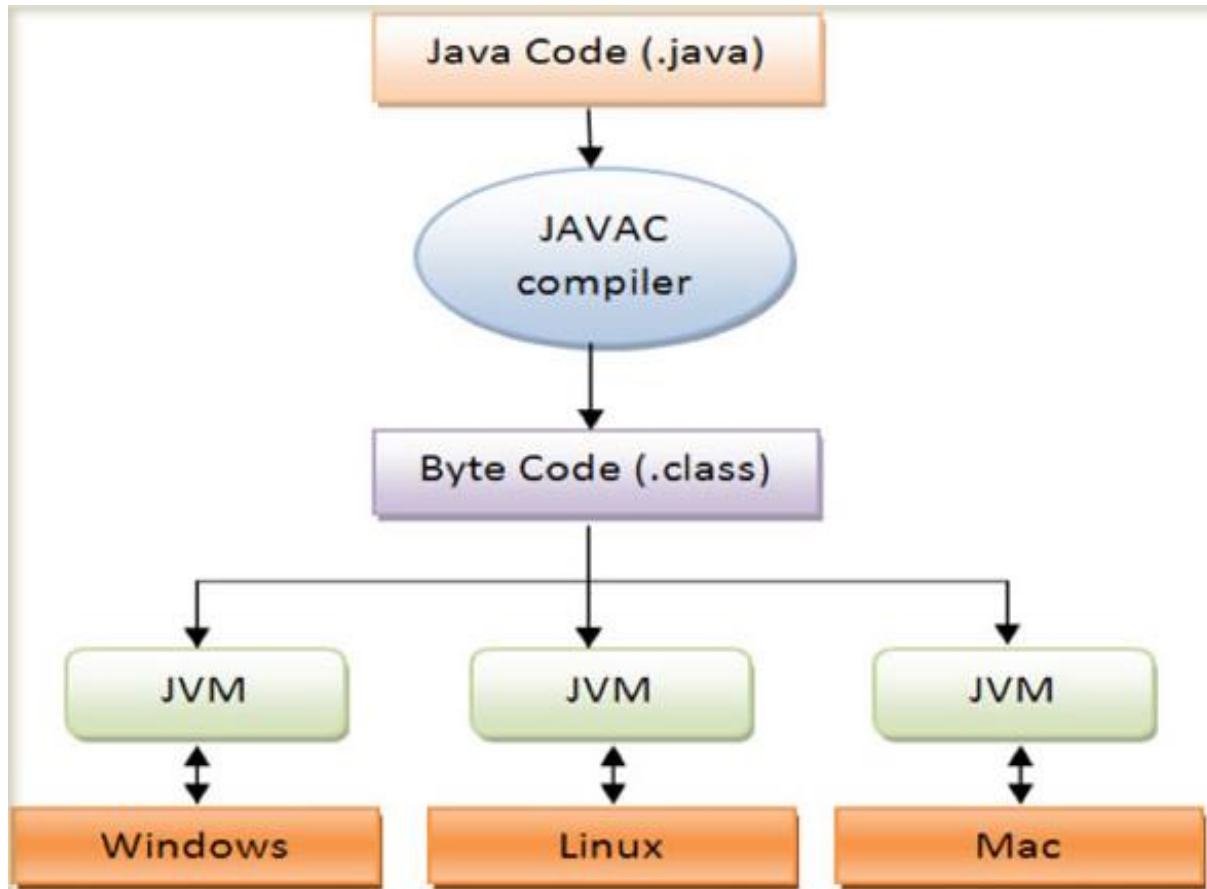
Основные технологии: *Java*, *Adobe Flash*, *Microsoft Silverlight*

В 2013 году Microsoft объявила, что они прекратили развитие Silverlight

Декабрь 2015 - Adobe [призвала](#) веб-разработчиков отказываться от использования Flash и переходить на такие современные веб-стандарты как HTML5. Flash Player для популярных веб-браузеров компания продолжит поддерживать. Он известен большим количеством своих уязвимостей.

Java значительно уступает в простоте создания графики и анимации.

Насыщенное интернет-приложение



Java

Java-апплет - это программа, написанная на языке **Java** и откомпилированная в байт-код. Выполняется в браузере с использованием виртуальной Java-машины (*JVM*). Апплеты используются для предоставления интерактивных возможностей веб-приложений, которые не возможны в HTML. Так как байт-код Java платформо-независим, то Java-апплеты могут выполняться браузерами на многих операционных платформах.

Насыщенное интернет-приложение

Java

Достоинства Java-апплетов:

- работают практически на большинстве операционных платформ;
- поддерживаются большинством браузеров;
- кэшируются в большинстве браузеров, что существенно ускоряет их загрузку при возвращении на веб-страницу;
- после первого запуска апплета, когда Java-машина уже выполняется и быстро запускается, выполнение апплетов происходит существенно быстрее;
- загружаются со скоростью сопоставимой с программами на других компилируемых языках, например C++, но во много раз быстрее чем на JavaScript.

Основные недостатки:

- требуется установка специальных Java-расширения, которые по умолчанию доступны не во всех браузерах;
- разработка пользовательского интерфейса с использованием апплетов является более сложной задачей по сравнению с HTML;
- не могут запускаться до первой загрузки виртуальной Java-машина, что может занимать значительное время;
- не имеют прямого доступа к локальным ресурсам клиентского компьютера, но могут иметь полный доступ к машине, на которой выполняются, если пользователь согласен на это;;

Насыщенное интернет-приложение

Пример HTML страницы со встроенным Flash-сценарием.

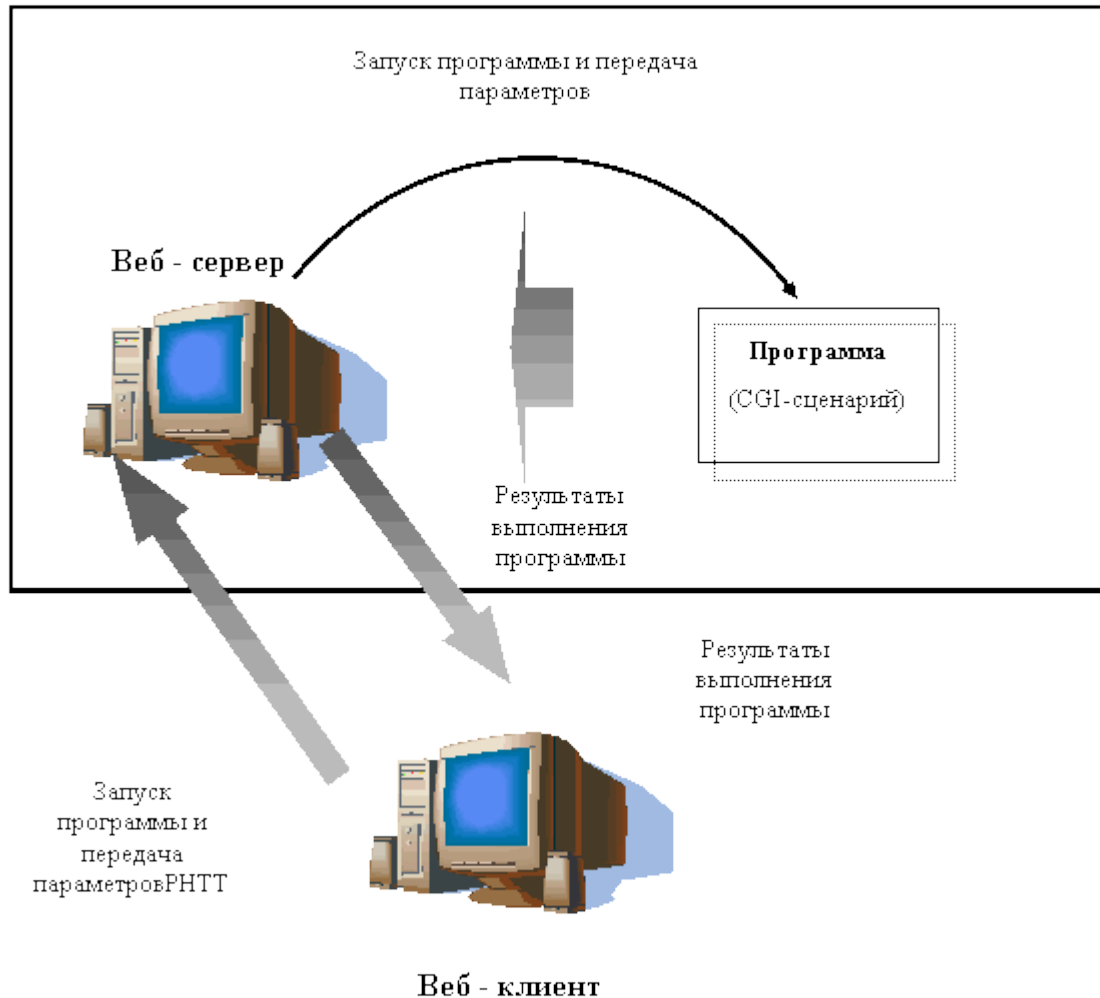
```
<object width="280" height="600" classid="clsid:d27cdeb6e-ae6d-11cf-96b8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflas
h.cab#version=9,0,18,0"
type="application/x-shockwave-flash" data="/banners/simply.swf">
<param name="quality" value="high" />
<param name="scale" value="noborder" />
<param name="salign" value="t" />
<param name="wmode" value="opaque" />
<param name="src" value="/banners/simply.swf" />
<embed width="280" height="600" id="linebanner" src="/banners/simply.swf">
</embed>
</object>
```

При использовании насыщенных интернет-приложений приходится сталкиваться со следующими проблемами:

- необходимость обеспечения безопасной среды выполнения ("песочница");
- для исполнения кода должно быть разрешено исполнение сценариев;
- потеря в производительности (т.к. выполняется на клиентской стороне);
- требуется много времени на загрузку;

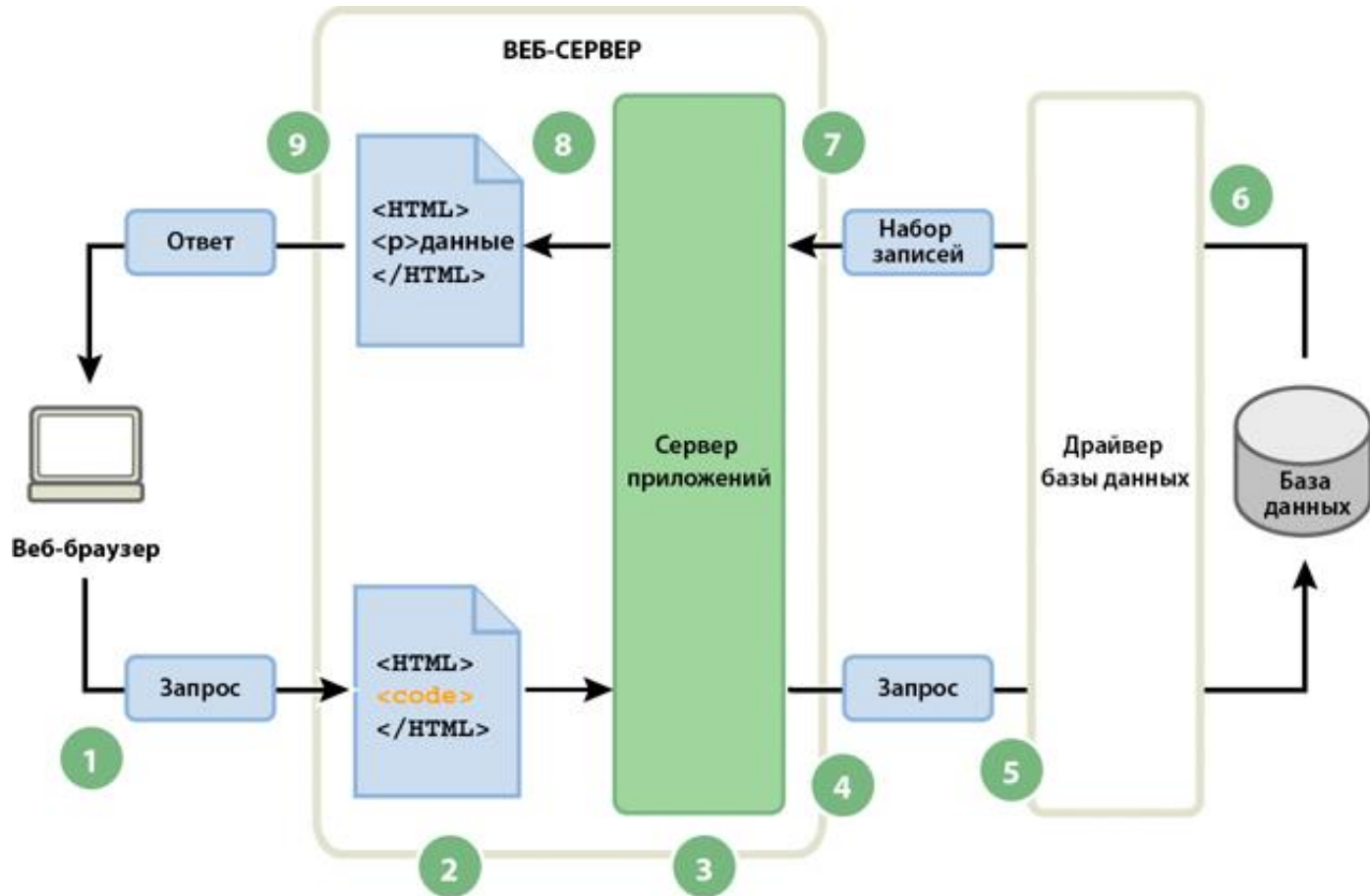
Архитектура Интернет-приложений.

Серверные сценарии



- Серверный сценарий выполняется в рамках активной страницы на Web-сервере до того, как тот вернет пользователю готовую HTML-страницу. Когда пользователь запрашивает активную серверную страницу, сервер выполняет сценарии и создает HTML-код, который и передается пользователю. В результате пользователь не видит серверного сценария на полученной Web-странице.
- Поскольку серверный сценарий выполняется на Web-сервере, ему доступны все ресурсы сервера – например, базы данных и исполняемые файлы.
- Для работы серверных сценариев Web-сервер должен поддерживать технологию активных страниц; к программе просмотра же не предъявляется никаких дополнительных требований, поскольку Web-клиент в данном случае получает стандартную HTML-страницу. Таким образом, сценарии серверной части не зависят от клиентов.

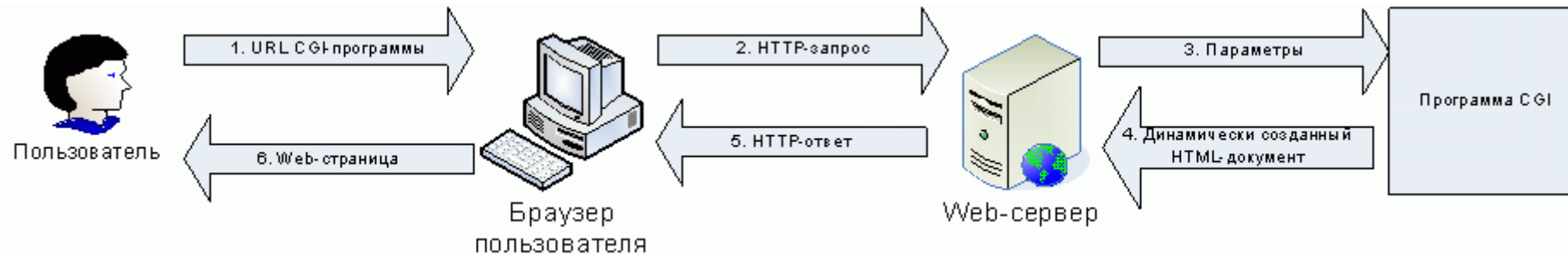
Архитектура Интернет-приложений.



1. Веб-браузер запрашивает динамическую страницу
2. Веб-сервер находит страницу и передает ее серверу приложений
3. Сервер приложений просматривает страницу на наличие инструкций и выполняет ее подготовку
4. Сервер приложений отправляет запрос драйверу базы данных
5. Драйвер выполняет запрос в базе данных
6. Драйверу возвращается набор записей
7. Драйвер передает набор записей серверу приложений
8. Сервер приложений вставляет данные в страницу и передает страницу веб-серверу
9. Веб-сервер отправляет подготовленную страницу запросившему ее браузеру.

CGI-интерфейс

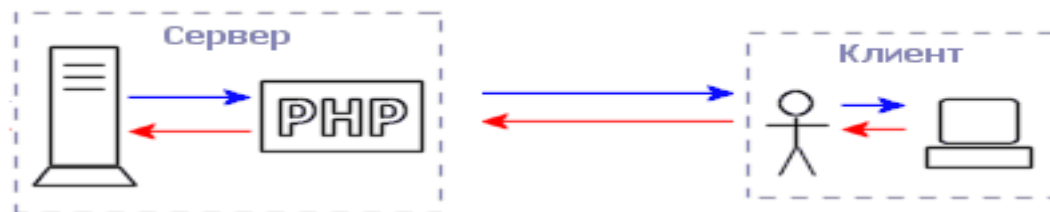
CGI-интерфейс (Common Gateway Interface – Общий шлюзовой интерфейс) — одно из первых решений, созданных для доставки динамической web-информации.



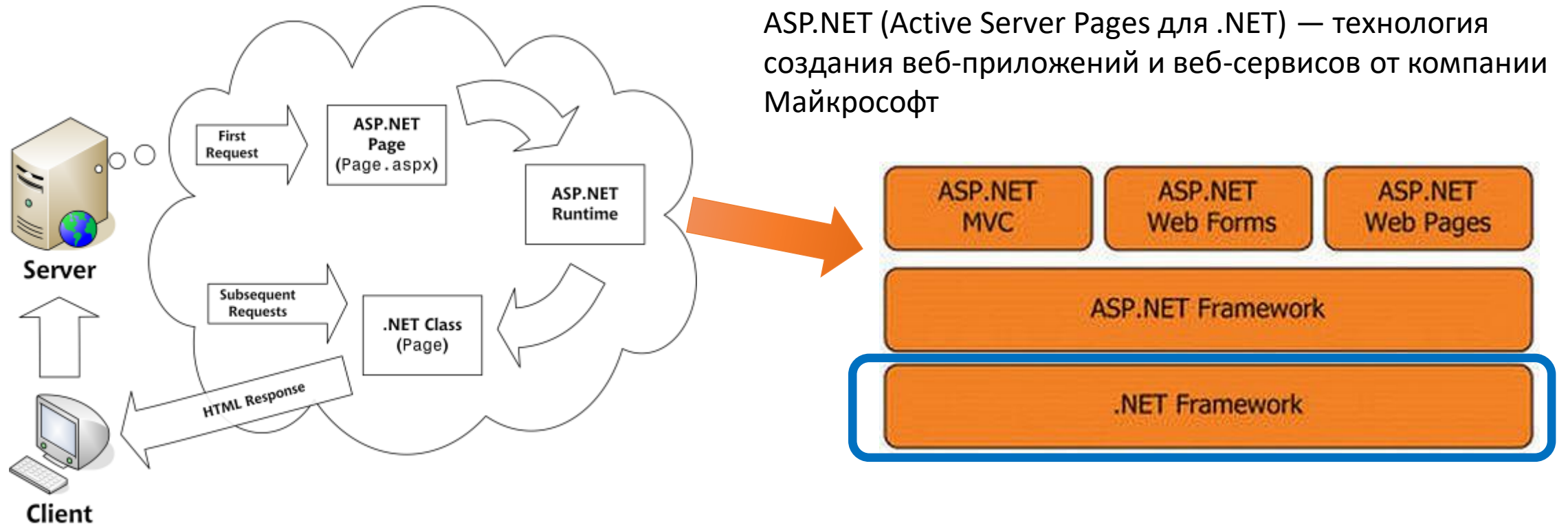
Последовательность работы CGI состоит из следующих этапов:

- Получение Web-сервером информации от клиента-браузера. Для передачи данных Web-серверу используются формы. Когда пользователь заполнит всю форму, он нажимает эту кнопку, и данные из полей формы передаются программе CGI.
- Анализ и обработка полученной информации. Данные, извлеченные из HTML-формы, передаются для обработки CGI-программе. Они не всегда могут быть обработаны CGI-программой самостоятельно. Например, они могут содержать запрос к базе данных. В этом случае CGI-программа на основании полученной информации формирует запрос к ядру СУБД, выполняющейся на том же или удаленном компьютере.
- Создание нового HTML-документа и пересылка его браузеру. После обработки полученной информации CGI-программа создает динамический (виртуальный) HTML-документ, или формирует ссылку на уже существующий документ и передает результат через сервер браузеру клиента.

PHP



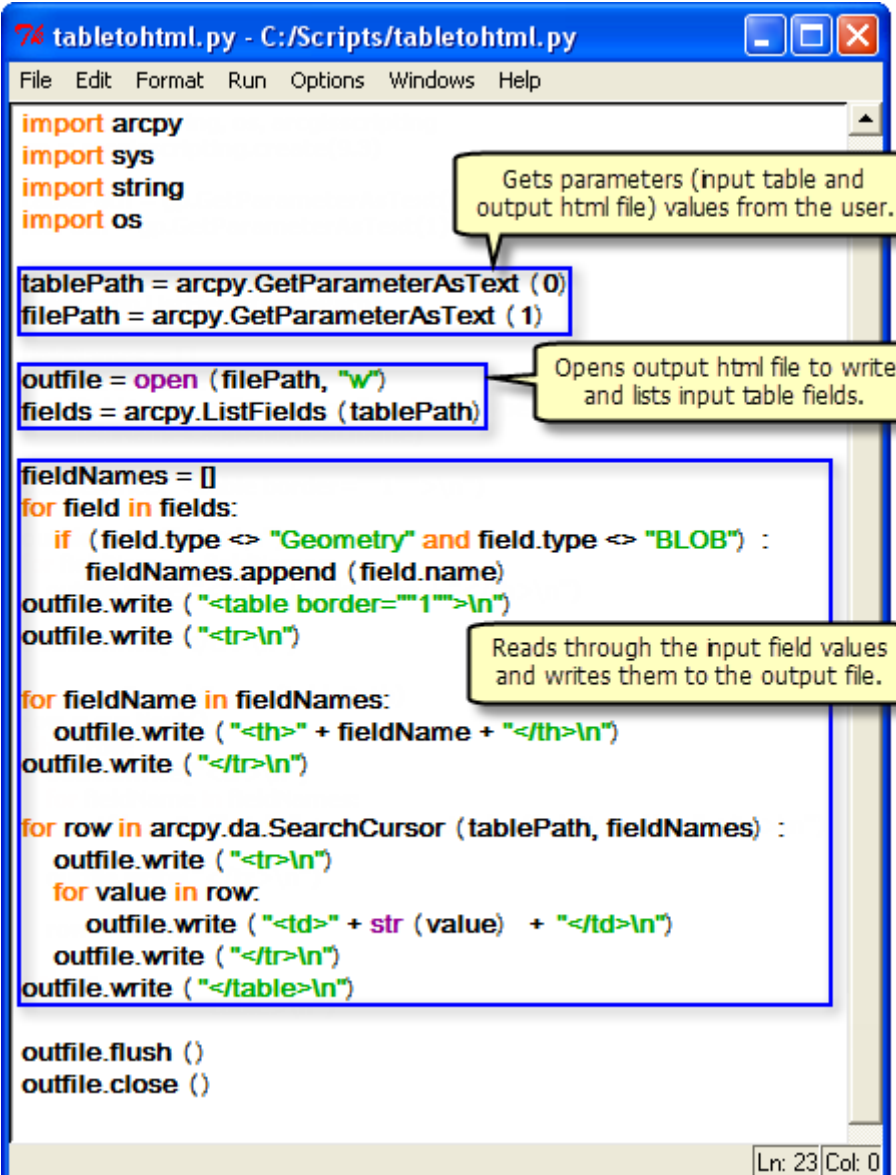
ASP.NET



ASP.NET (Active Server Pages для .NET) — технология создания веб-приложений и веб-сервисов от компании Майкрософт

Разработчики могут писать код для ASP.NET, используя любые языки программирования, входящие в комплект .NET Framework (C#, Visual Basic.NET и JScript .NET).

Python



```
tabletohtml.py - C:/Scripts/tabletohtml.py
File Edit Format Run Options Windows Help

import arcpy
import sys
import string
import os

tablePath = arcpy.GetParameterAsText (0)
filePath = arcpy.GetParameterAsText (1)

outfile = open (filePath, "w")
fields = arcpy.ListFields (tablePath)

fieldNames = []
for field in fields:
    if (field.type <= "Geometry" and field.type <= "BLOB") :
        fieldNames.append (field.name)
outfile.write ("<table border=""1"">\n")
outfile.write ("<tr>\n")

for fieldName in fieldNames:
    outfile.write ("<th>" + fieldName + "</th>\n")
outfile.write ("</tr>\n")

for row in arcpy.da.SearchCursor (tablePath, fieldNames) :
    outfile.write ("<tr>\n")
    for value in row:
        outfile.write ("<td>" + str (value) + "</td>\n")
    outfile.write ("</tr>\n")
outfile.write ("</table>\n")

outfile.flush ()
outfile.close ()
```

Gets parameters (input table and output html file) values from the user.

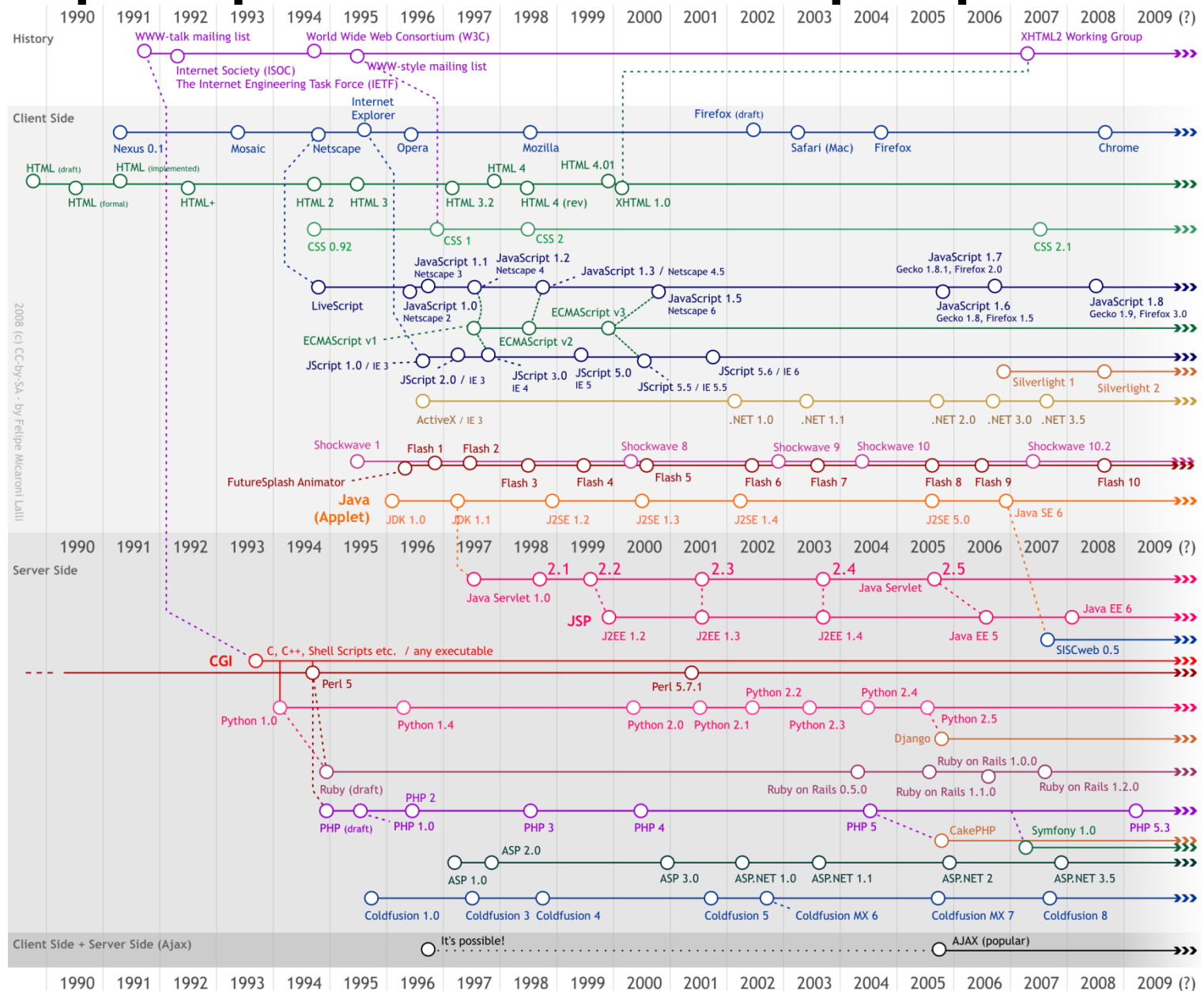
Opens output html file to write and lists input table fields.

Reads through the input field values and writes them to the output file.

Ln: 23 Col: 0

Python (в русском языке распространено название **питон**) — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

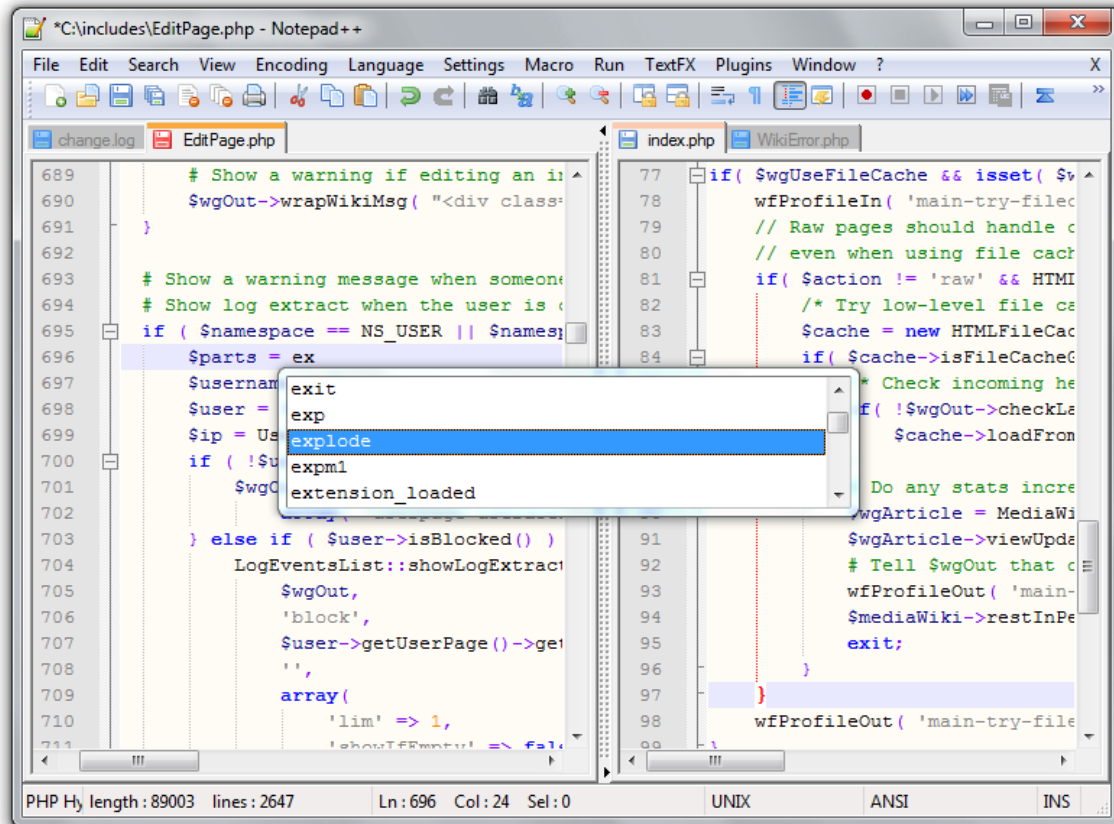
История развития web-программирования



Инструменты разработки интернет-приложений

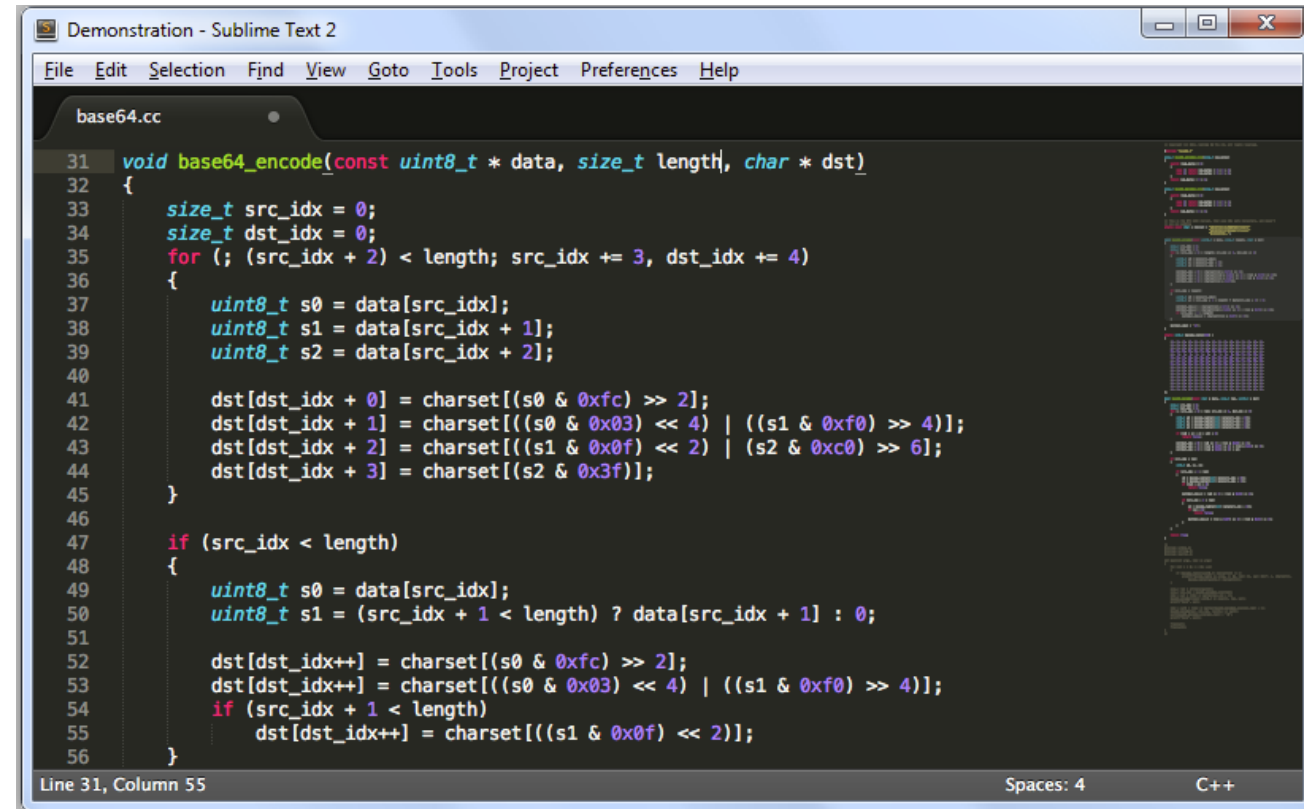
Редакторы HTML-страниц (редакторы кода)

Notepad++



The screenshot shows the Notepad++ editor interface. The main window displays PHP code from a file named 'EditPage.php'. A dropdown menu is open over the code, listing options: 'exit', 'exp', 'explode', 'expml', and 'extension_loaded'. The status bar at the bottom indicates 'PHP Hy length: 89003 lines: 2647 Ln: 696 Col: 24 Sel: 0'.

Sublime Text 2



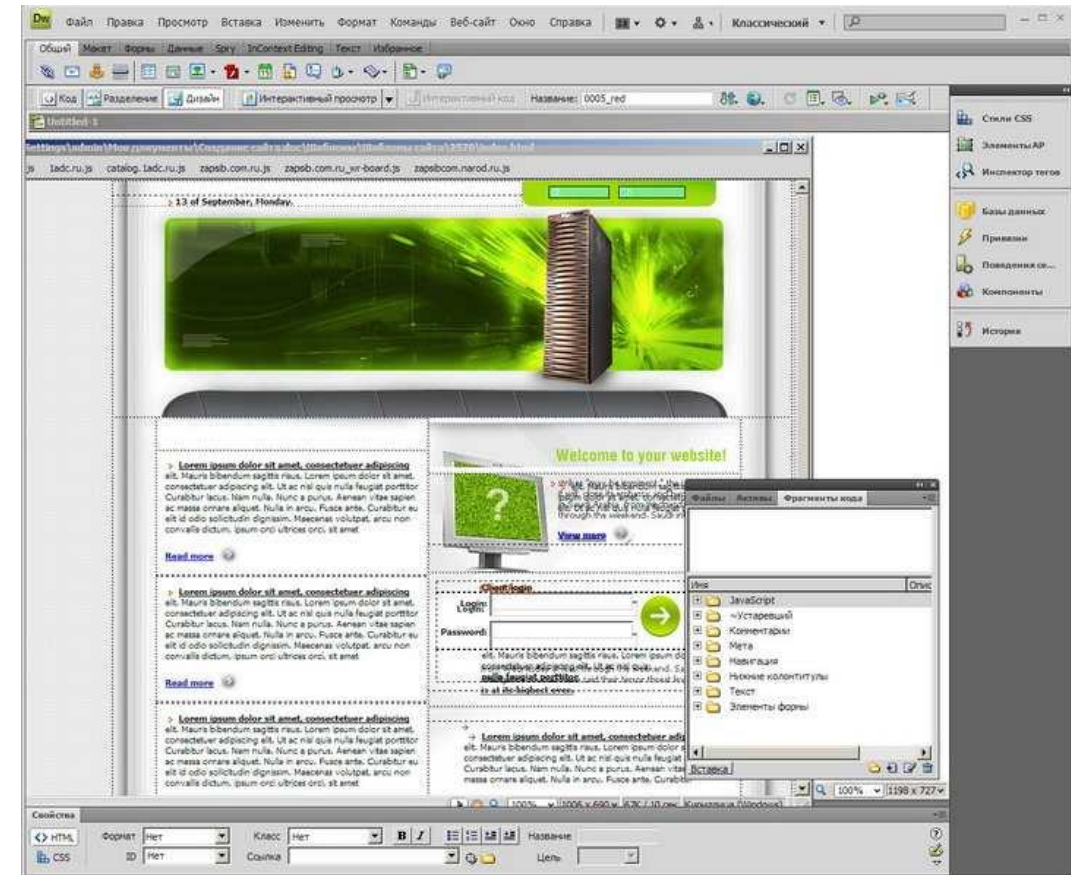
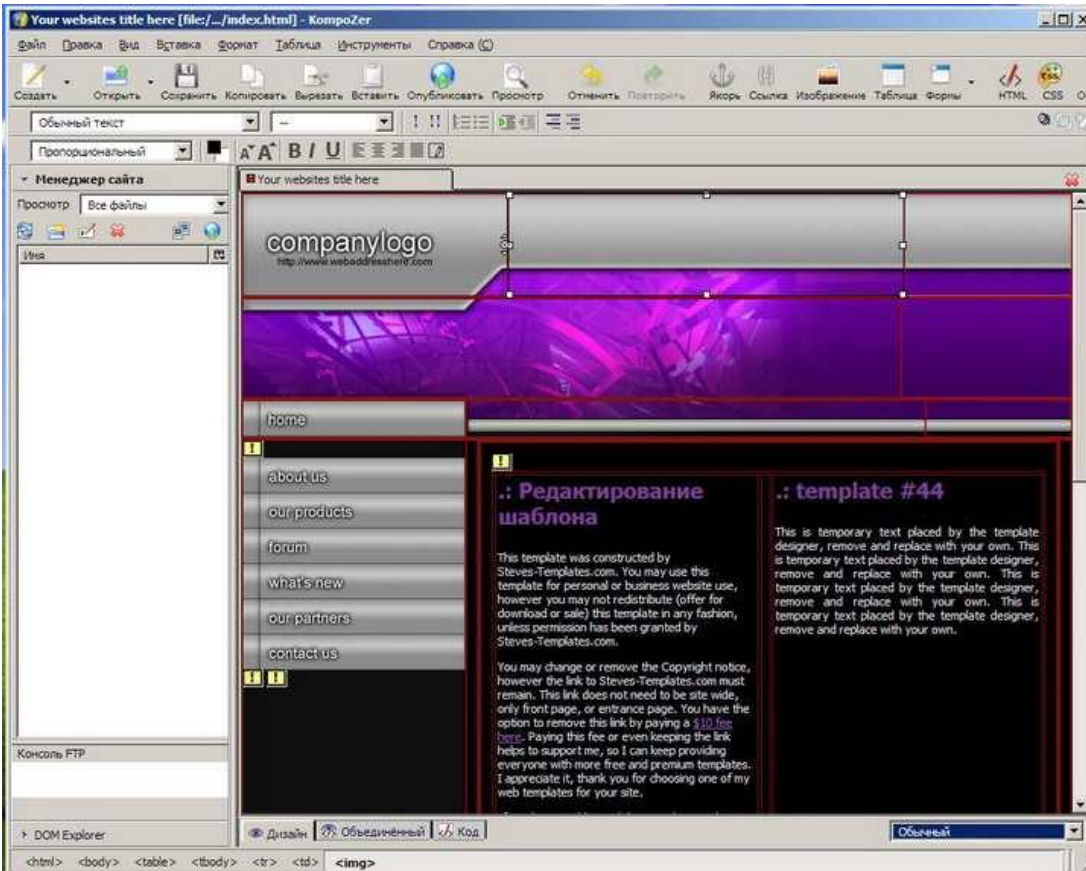
The screenshot shows the Sublime Text 2 editor interface. The main window displays C++ code from a file named 'base64.cc'. The code is a function 'base64_encode' that takes a pointer to a byte array and its length, and returns a base64-encoded string. The status bar at the bottom indicates 'Line 31, Column 55 Spaces: 4 C++'.

Инструменты разработки интернет-приложений

Редакторы HTML-страниц (комбинированные редакторы)

KompoZer

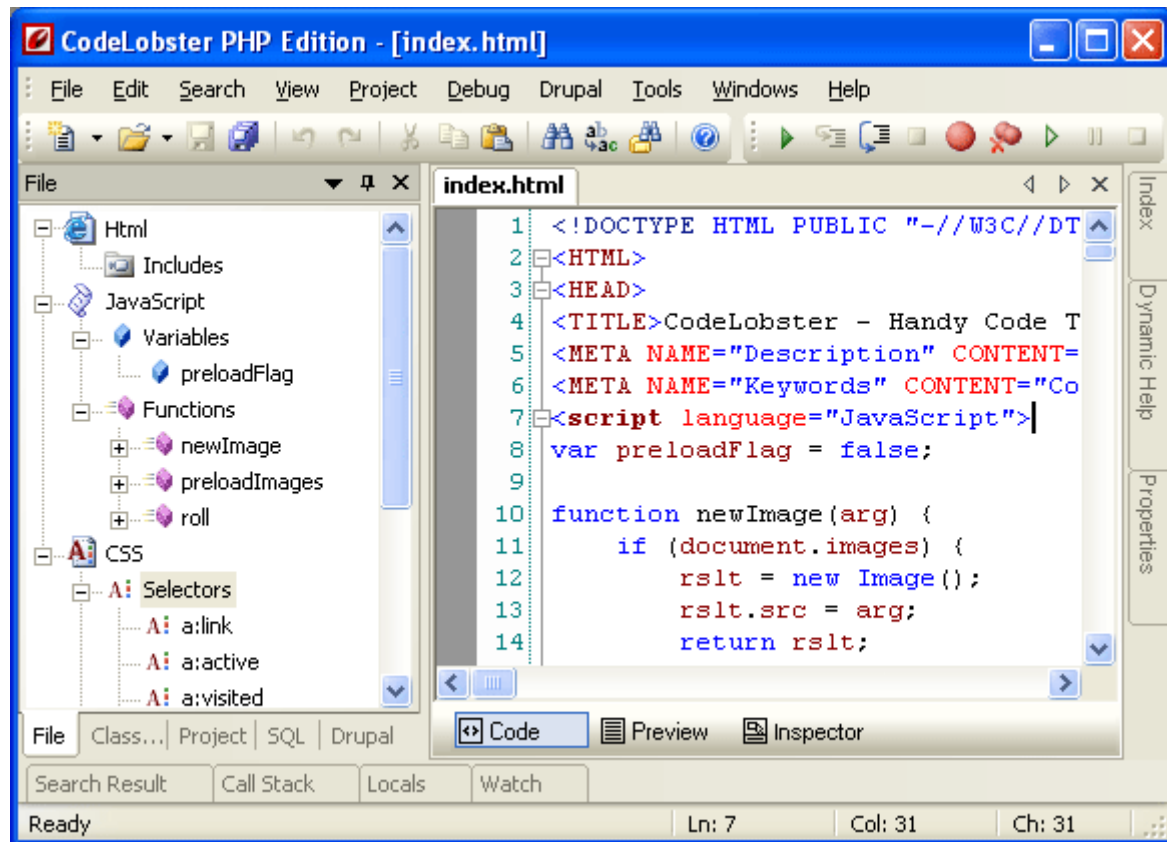
Adobe Dreamweaver



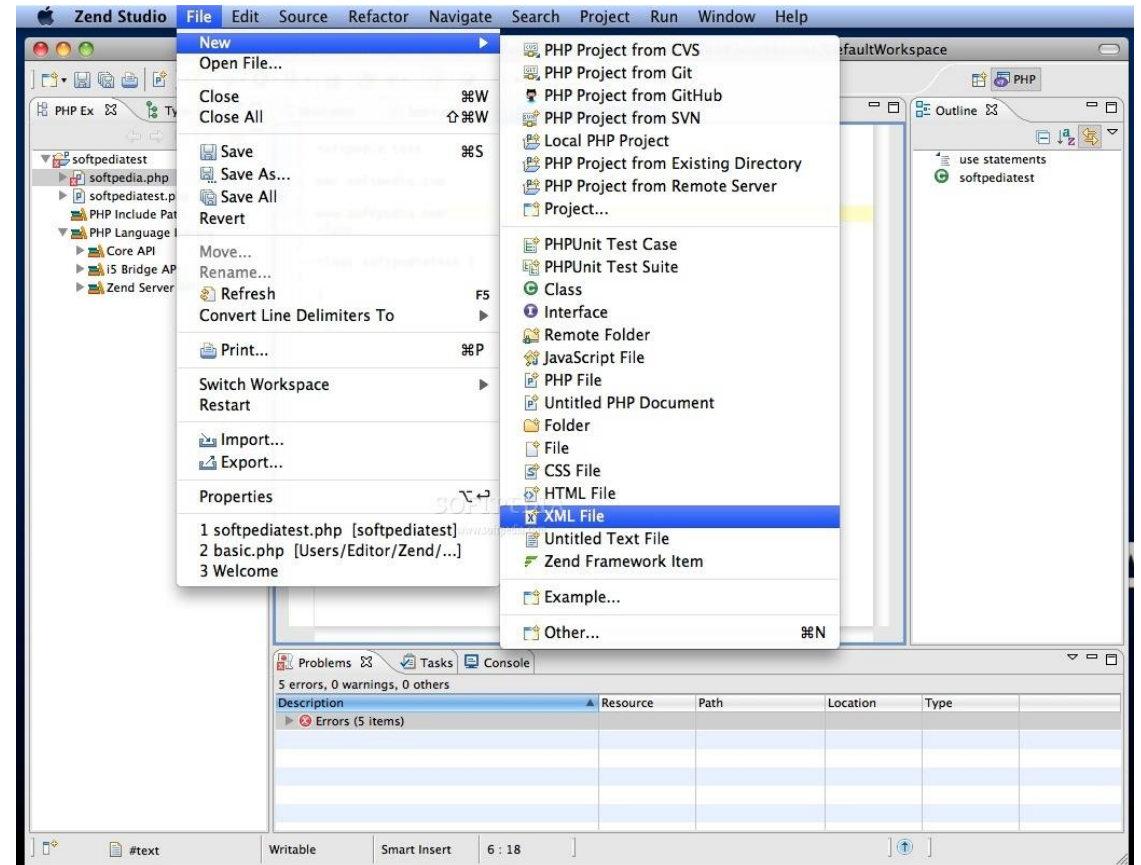
Инструменты разработки интернет-приложений

Редакторы для создания скриптов

CodeLobster PHP Edition



Zend Studio

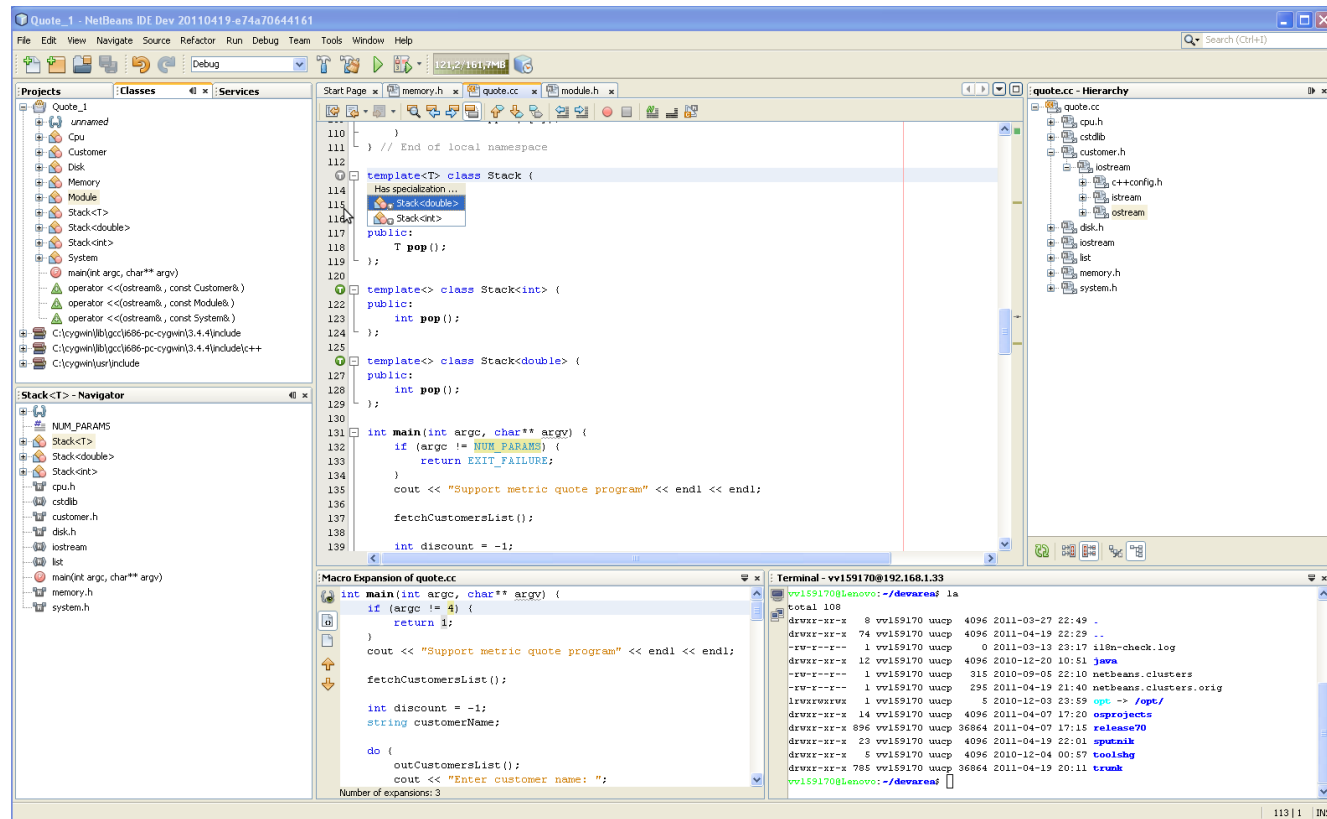
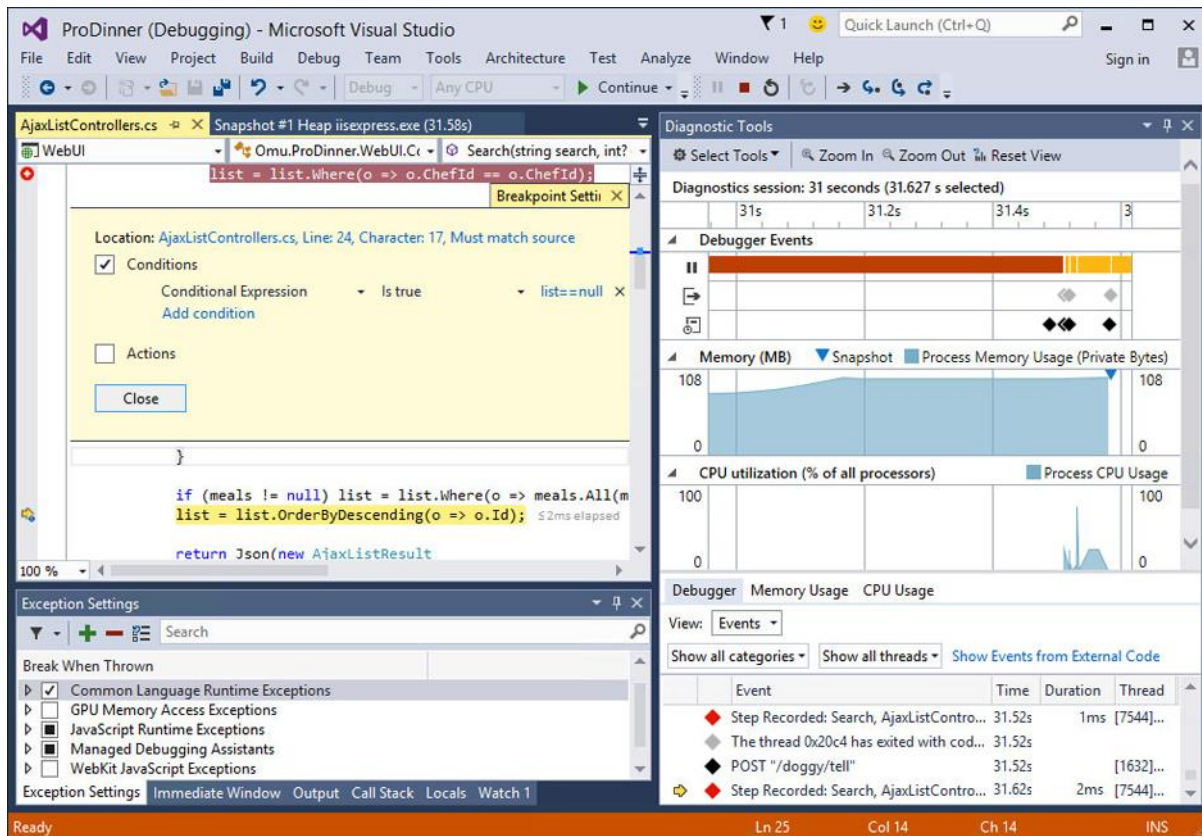


Инструменты разработки интернет-приложений

Интегрированная среда разработки (англ. IDE, *Integrated development environment*)

MS Visual Studio

NetBeans





Выводы по теме

Рассмотрены следующие вопросы:

- Основные технологические стандарты сети Интернет;
- Принципы работы CGI-приложений;
- Архитектура веб-приложений;
- Основы протокола HTTP.



Рекомендуемая литературы по теме

Современные информационные технологии в управлении экономической деятельностью. Теория и практика : учеб. пособие / Б. Е. Одинцов, А. Н. Романов, С. М. Догучаева. ИНФРА-М, 2017. – 372 с. : ил. – (Вузовский учебник) . – ISBN 978-5-9558-0517-7 . –.

Тузовский, А. Ф. Проектирование и разработка web-приложений : учебное пособие для академического бакалавриата / А. Ф. Тузовский. М. : Издательство Юрайт, 2018. — 218 с. — (Серия : Университеты России). — ISBN 978-5-534-00515-8.