Тема 2

АРХИТЕКТУРЫ РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

ПОНЯТИЕ АРХИТЕКТУРЫ РАСПРЕДЕЛЕННОЙ СИСТЕМЫ

- Организация РС определяется тем каким образом программное обеспечение РС распределяется между вычислительными узлами этой системы.
- В организации систем часто выделяют:
  - Логическую организацию совокупности программных компонент системы;
  - Физическую организацию размещение этих компонент на узлах системы.

ПРОГРАММНАЯ АРХИТЕКТУРА

- Организация PC определяется составом программных компонент входящих в состав системы.
- Программная архитектура показывает помимо того как организована система, также и то как взаимодействуют между собой программные компоненты этой системы.

ПРОЗНАЧНОСТЬ РС И ЕЕ АРХИТЕКТУРА

- Исходя из требования обеспечения прозрачности в распределенных системах требуется четко разделять приложения и лежащие в их основе платформы.
- Такое разделение в РС выполняется с помощью промежуточного уровня системы.

Распределенное приложение
ПО промежуточного слоя (Midleware Layer)
Аппаратно-программная платформа

ВЫБОР ВАРИАНТА ПРОГРАММНОЙ АРХИТЕКТУРЫ

- Важнейшим решением при разработке архитектуры системы является:
  - выбор варианта размещения ПО промежуточного уровня (ППУ) системы.
- Имеется различные методики определение состава и размещения ППУ приложений, что и определяет множество вариантов программных архитектур.

СИСТЕМНАЯ АРХИТЕКТУРА

- Фактическая (реально разворачиваемая) реализация РС, требует однозначного определения размещения программных компонент системы на реальных машинах.
- Практически всегда имеется множество вариантов такого размещения.
- Размещение программных компонент системы (программная архитектура) на физических машинах называется системной архитектурой.

ВИДЫ СИСТЕМНОЙ АРХИТЕКТУРЫ

- Различают три вида системной архитектуры:
  - централизованная;
  - децентрализованная (peer-to-peer);
  - гибридная – комбинация элементов централизованной и децентрализованной архитектур.

ПОНЯТИЕ АРХИТЕКТУРНОГО СТИЛЯ

- В настоящее время исследования в области программного обеспечения достигли достаточной зрелости, что позволило однозначно определить понятие архитектурного стиля (архитектуры) РИС.
- При проектировании и создании РС выбор архитектуры является ключевым техническим решением, определяющим успех создания больших программных систем.
- При обсуждении архитектурных аспектов РС важным понятием является архитектурный стиль, который описывается в терминологии компонент и определяет:
  - способ коммуникаций между компонентами;
  - порядок обмена данными между компонентами;
  - как элементы системы совместно формируют распределенную систему.

ПОНЯТИЕ ПРОГРАММНОГО КОМПОНЕНТА

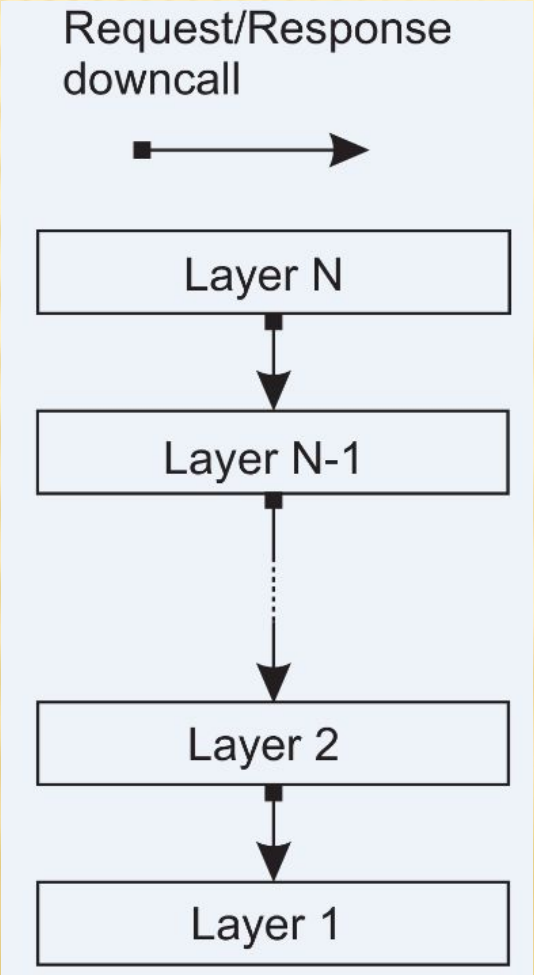
- **Компонент** – модульная единица ПО снабженная полностью определенным и предоставляемым по запросу интерфейсом.
- Компонент должен обладать свойством – ***заменяемости*** *(replaceable)* в рамках системного окружения. Замена компонента может быть выполнена в любой момент, даже в условиях работы системы. Последний аспект определяет, что в PC может отсутствовать опция
- **Интерфейс** - описывает состав параметров необходимых для обращению к компоненту. Замена компонента может быть выполнена только при условии неизменности его интерфейса.
- **Конектор** – это механизм который обеспечивает коммуникации, и способствует координации (или кооперации) компонент друг с другом.
- Конектор может быть сформирован на основе средств реализующих способ связи между компонентами. :
  - удаленный вызов процедур (RPC);
  - обмен сообщениями (message passing);
  - потока данных (streaming data) и д.р.
- Использование понятий компонент и конектор позволяет описывать различные варианты конфигураций, которые в свою очередь могут быть квалифицированы как **архитектурные стили**.

ОСНОВНЫЕ ВИДЫ АРХИТЕКТУР РИС

- В настоящее время общепризнанными архитектурными стилями считаются:
  - многоуровневые архитектукры (layered);
  - объектные архитектуры (object-based);
  - ресурсо-центрированные архитектуры (resource-centerd) ;
  - архитектуры основанные на событиях (event-based).
- Эти стили могут одновременно применяться в одних и тех же системах в различных сочетаниях.

МНОГОУРОВНЕВАЯ АРХИТЕКТУРА

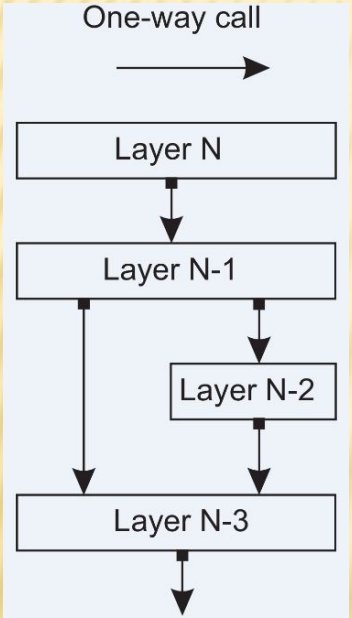
Простая многоуровневая архитектура



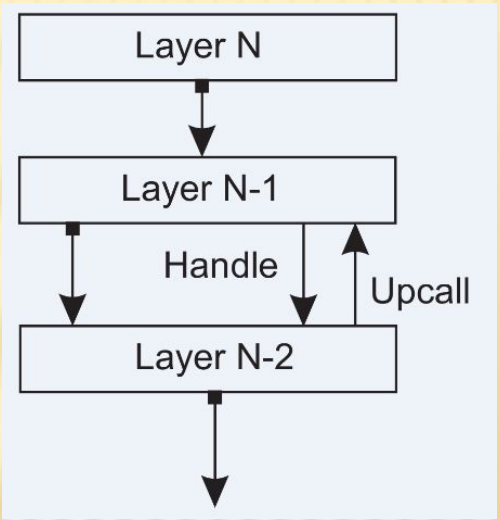
- Базовая идея проста: компоненты распределяются по уровням, при этом компонент уровня j L_j может вызывать компонент находящийся на нижележащем уровне i (L_i), от которого он получает ответ.
- Как правило вызов компонента выше стоящего уровня нижележащим запрещен.

ДРУГИЕ ВИДЫ МНОГОУРОВНЕВЫХ АРХИТЕКТУР

- Смешанная многоуровневая архитектура – допускает вызов не только смежных нижележащих уровней, но и более глубоколежащих уровней.



Смешанная Многоур. архитектура

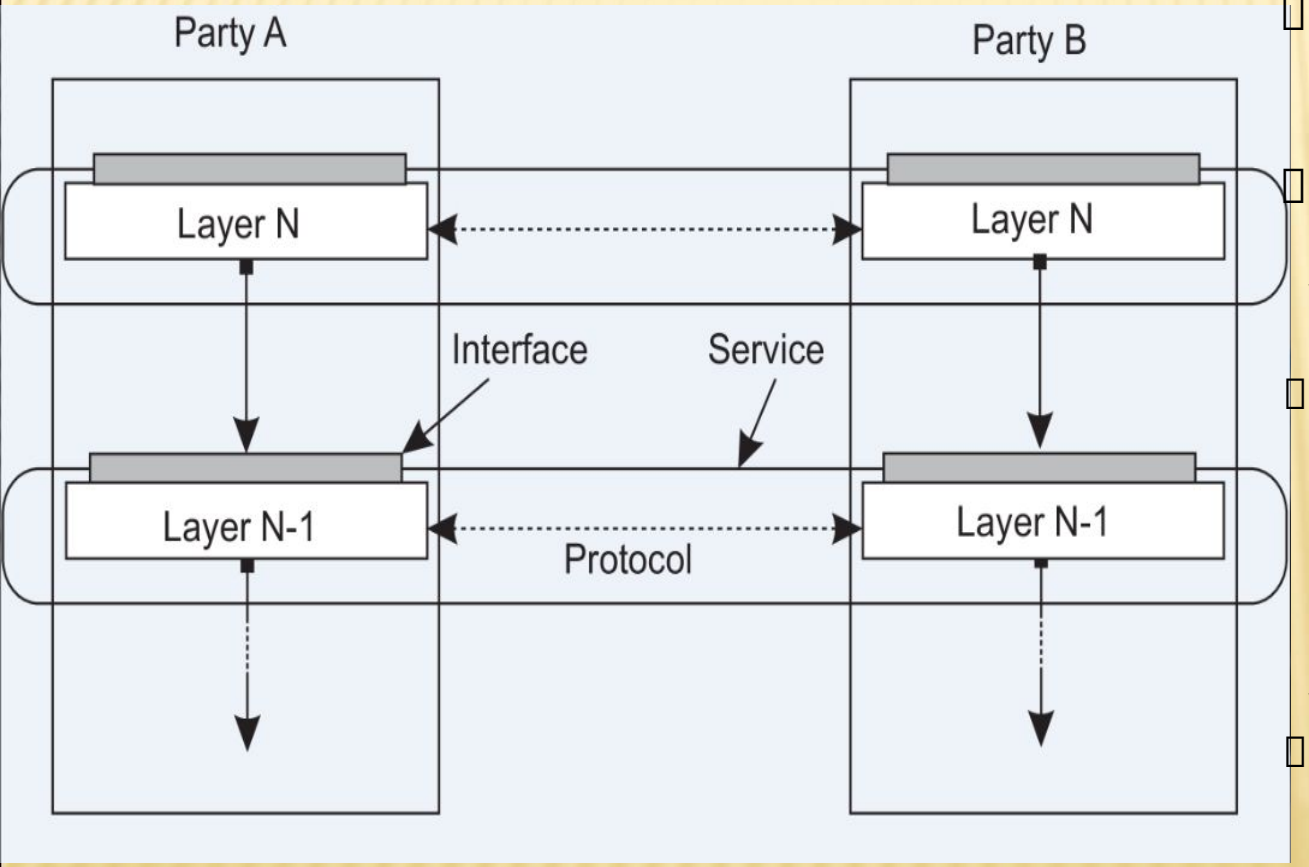


Архитектура с обратными связями

- Многоуровневая архитектура с обратными связями

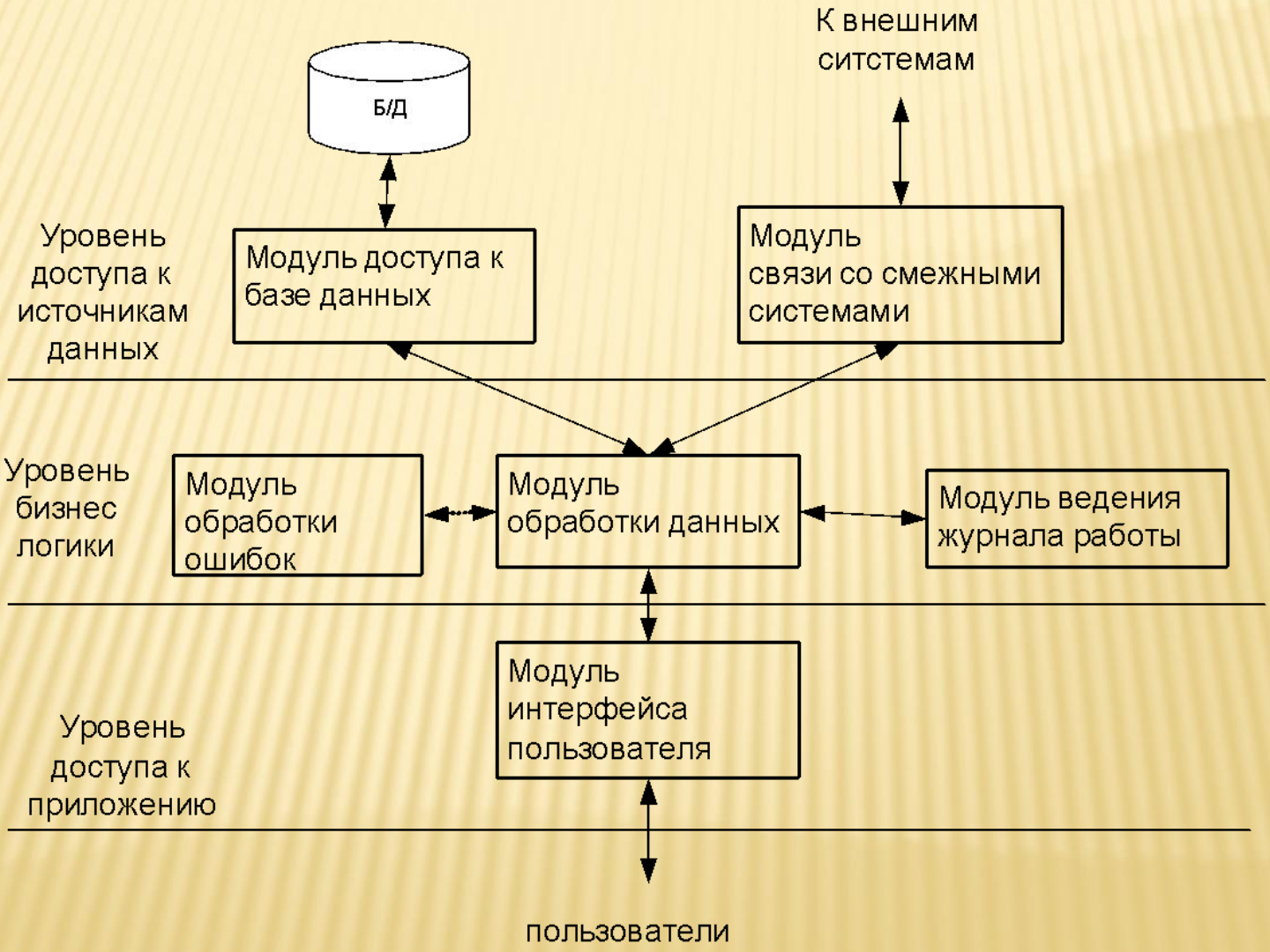
МНОГОУРОВНЕВАЯ ОРГАНИЗАЦИЯ СТЕКА СЕТЕВЫХ ПРОТОКОЛОВ

- Примером такой организации может служить стек сетевых протоколов, например : TCP/IP

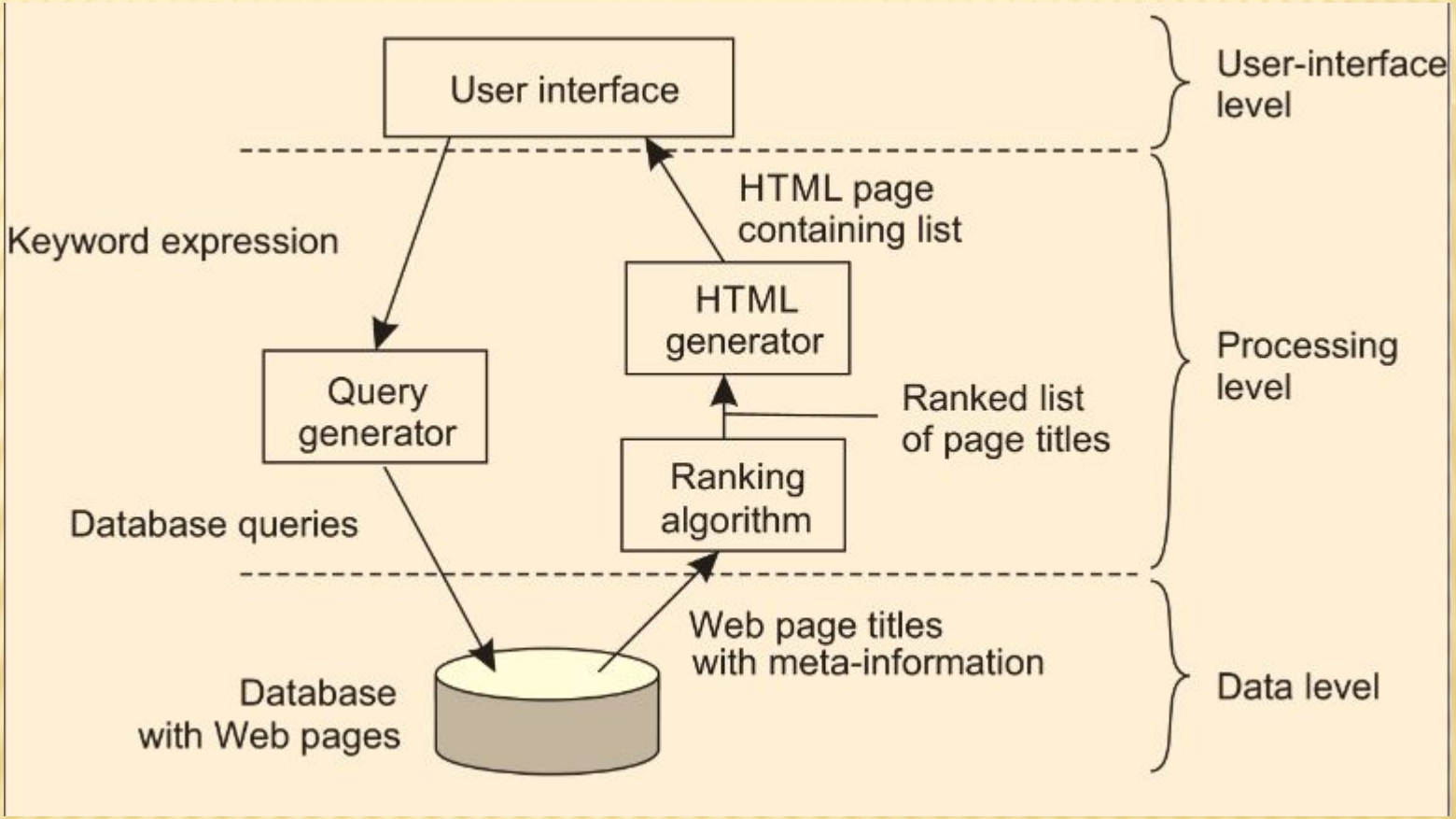


- В этой архитектуре определены следующие понятия (концепты):
- интерфейс - определяет функции нижележащего уровня, вызываемые вышележащим уровнем;
- протокол - процедуры и правила, которым должны следовать обе взаимодействующие стороны при обмене информацией на данном уровня;
- сервис, реализуется средствами одного уровня .

АРХИТЕКТУРА ПРИЛОЖЕНИЯ (РАССЛОЕНИЕ ПРИЛОЖЕНИЯ)

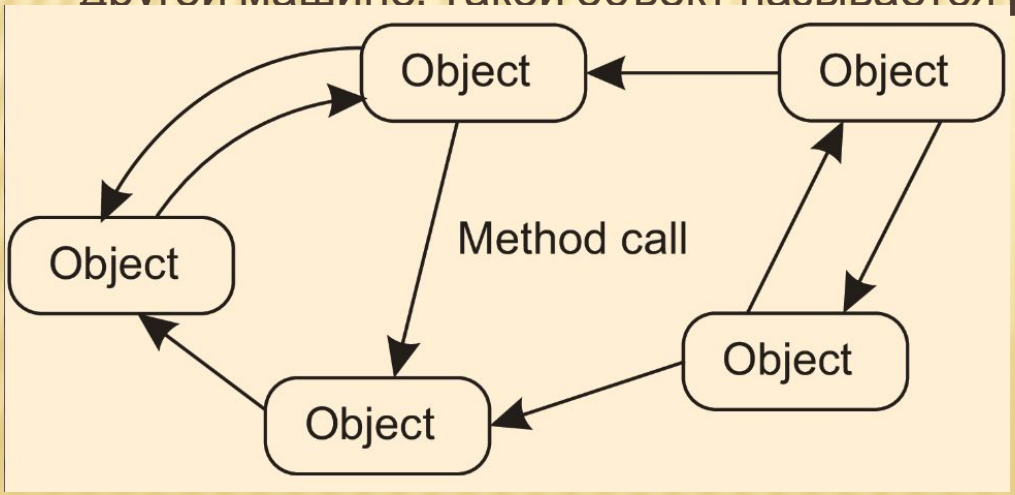


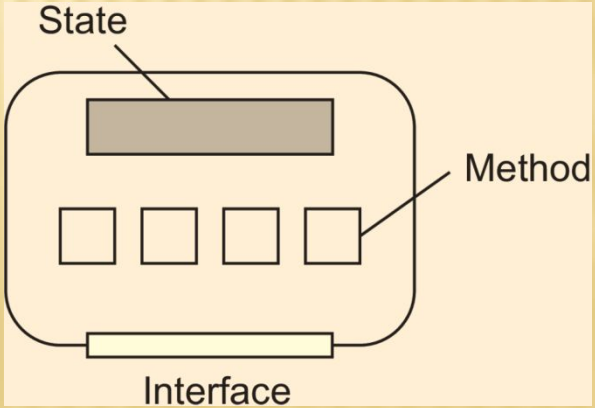
МНОГОУРОВНЕВАЯ ОРГАНИЗАЦИЯ ПРИЛОЖЕНИЙ



АРХИТЕКТУРА ОСНОВАННАЯ НА ОБЪЕКТАХ (OBJECT-BASED ARCHITECTURE)

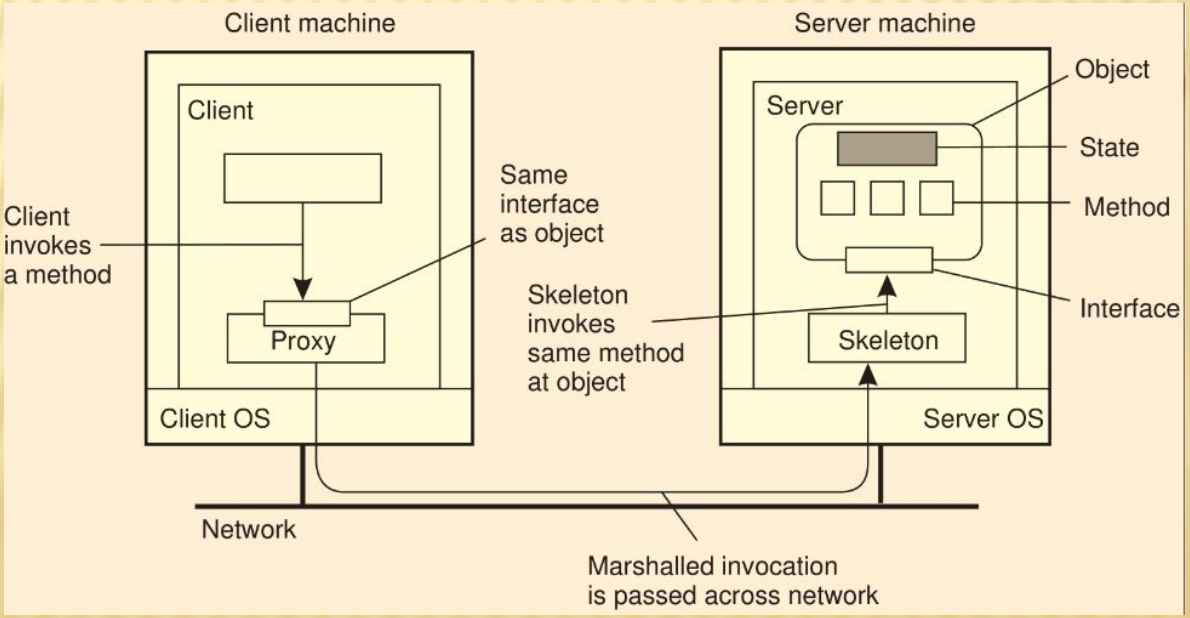
- В этой архитектуре понятие **объект** однозначно соответствует понятию **компонент**.
- Объекты взаимодействуют между собой с помощью механизма **RPC.**
- Эта архитектура предоставляет естественный способ инкапсуляции данных (**состояние объекта**) и операций которые могут выполняться над этими данными (**методы объекта**) объектом.
- **Интерфейс** предоставляемый объектом скрывает детали реализации объекта, что делает объект полностью независимым от его окружения.
- Разделение интерфейсов и реализации объектом этого интерфейса позволяет разместить интерфейс объекта на одной машине, а сам объект на другой машине. Такой объект называется **распределенным**.





РАСПРЕДЕЛЕННЫЕ ОБЪЕКТЫ

- Когда клиент связывается с распределенным объектом, реализация интерфейса этого объекта, называемая **proxy**, загружается в адресное пространство клиента.
- Посредник (proxy) это аналог заглушки (stub) клиента в RPC. Он выполняет маршалинг вызова метода в сообщение и извлечение результата из ответного сообщения получаемого от сервера.
- Реальный объект располагается на удаленной машине, где имеется такой же интерфейс как и на клиентской машине. Входящий вызов метода принимает серверная заглушка (stub, также часто называемая skeleton), которая извлекает вызов из сообщения и передает его интерфейсу объекта сервера. Заглушка сервера размещает ответ в сообщение и отправляет его proxy клиента.

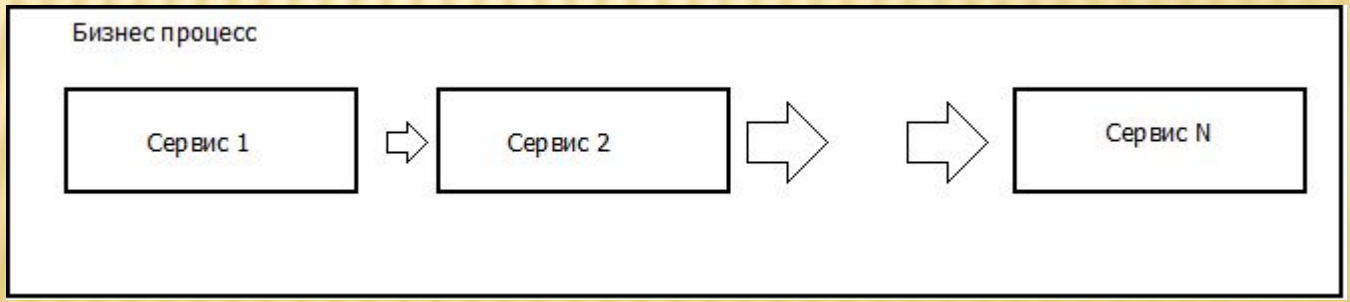


СЕРВИС

- Архитектура распределенных объектов ее основе сформировать **сервис** как независимую программную единицу.
- Сервис реализуется как **самодостаточная сущность** созданная на основе распределенного объекта.
- Понятие сервиса как независимой самостоятельной единицы РИС, позволяет определить сервис-ориентированную архитектуру.

ОСНОВНЫЕ ПОНЯТИЯ СОА (1)

- **СОА** – это способ описания требований и методология предоставления сервисов, независимых от программных платформ и языков разработки, для использования при создании распределенных приложений.
- **Сервис** – это повторно-исполняемая задача в рамках бизнес процесса.
- **Процесс** (бизнес задача) – это композиция составленная из отдельных сервисов.
- Процесс определяет логику взаимодействия сервисов, независимо от их реализации.



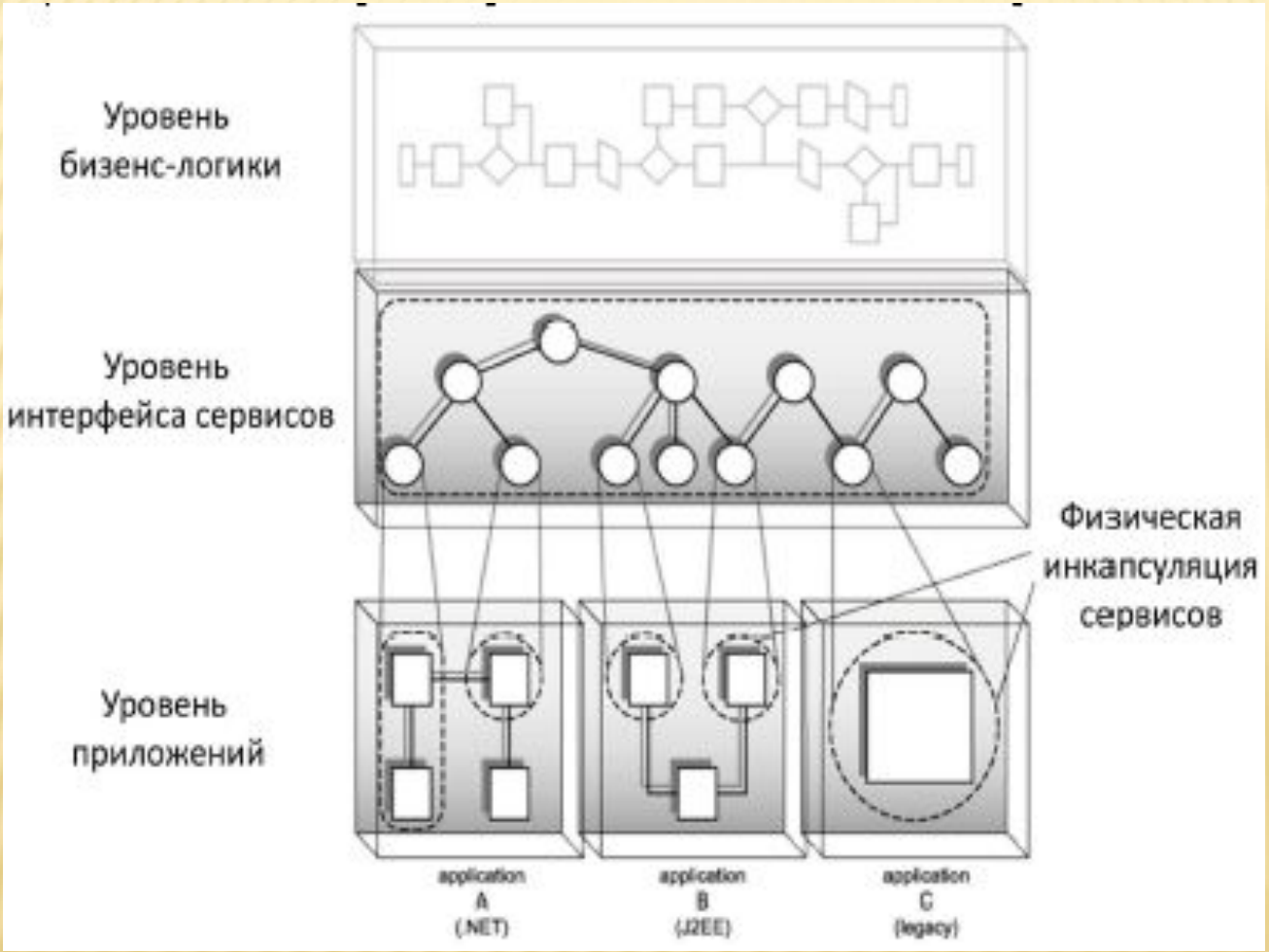
ОСНОВНЫЕ ПОНЯТИЯ СОА (2)

- SOA описывается как совокупность сервисов, реализуемых в виде компонентных объектов, систематизированных на основе обмена сообщениями (message-passing taxonomy).
- Общим примером сообщений, которыми обмениваются сервисы является XML сообщение.
- Для каждого сервиса принято определять:
  - поставщика сервиса (компонент сервис),
  - потребителя сервиса (компонент клиент);
  - брокера - компонент, обеспечивающий взаимодействие поставщика и потребителя.

ВЗАИМОДЕЙСТВИЕ МЕЖДУ ПОСТАВЩИКОМ И ПОТРЕБИТЕЛЕМ СЕРВИСА

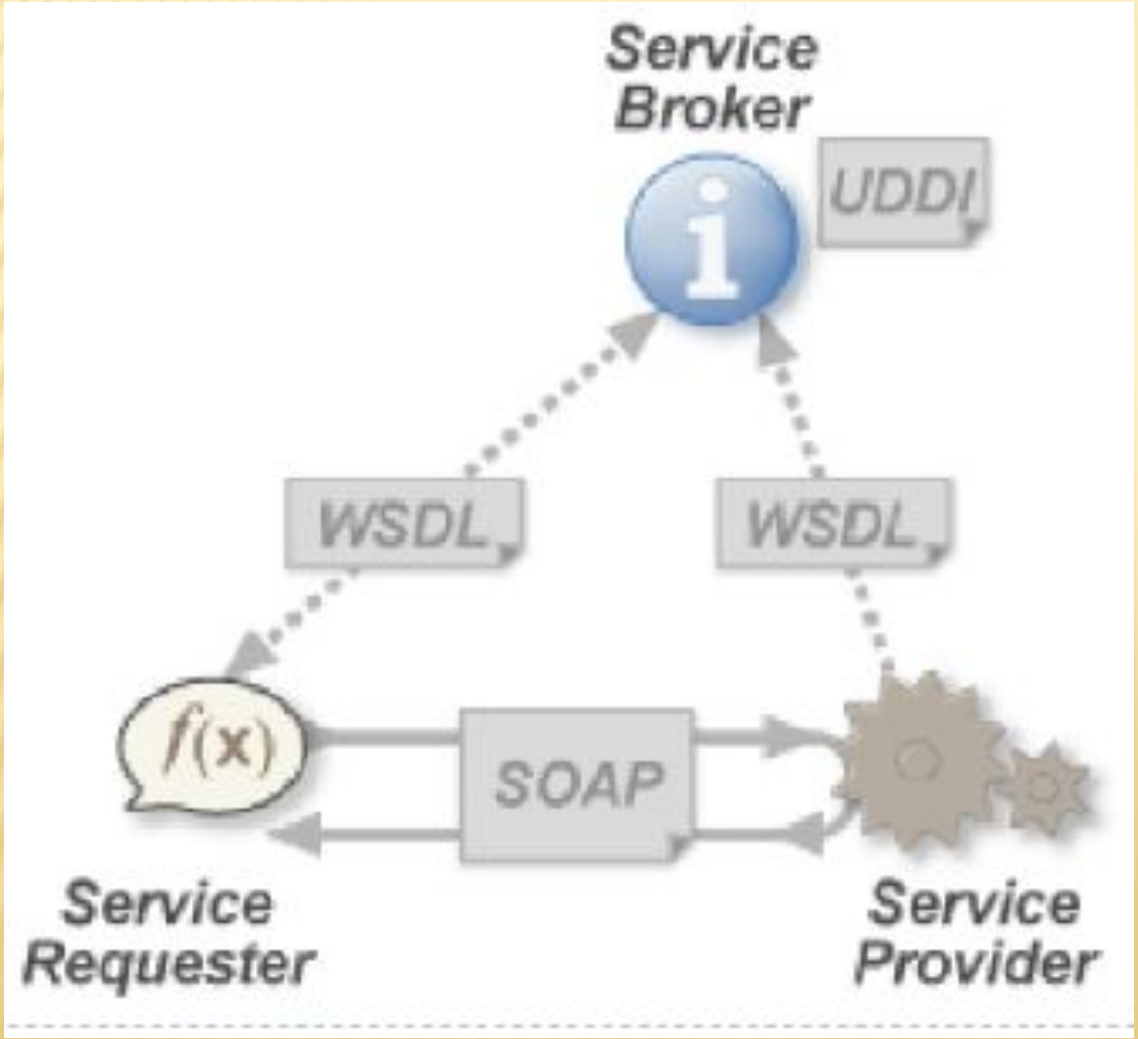
- Клиент обращается к поставщику посылая ему сообщение стандартного формата, содержащее метеданные на основе которых поставщик и будет действовать.
- Поставщик на основе полученных в сообщении метаданных формирует ответ.
- Ответ принятый от поставщика клиент может использовать по своему усмотрению.

АРХИТЕКТУРА ИС ПРЕДПРИЯТИЯ НА ОСНОВЕ SOA



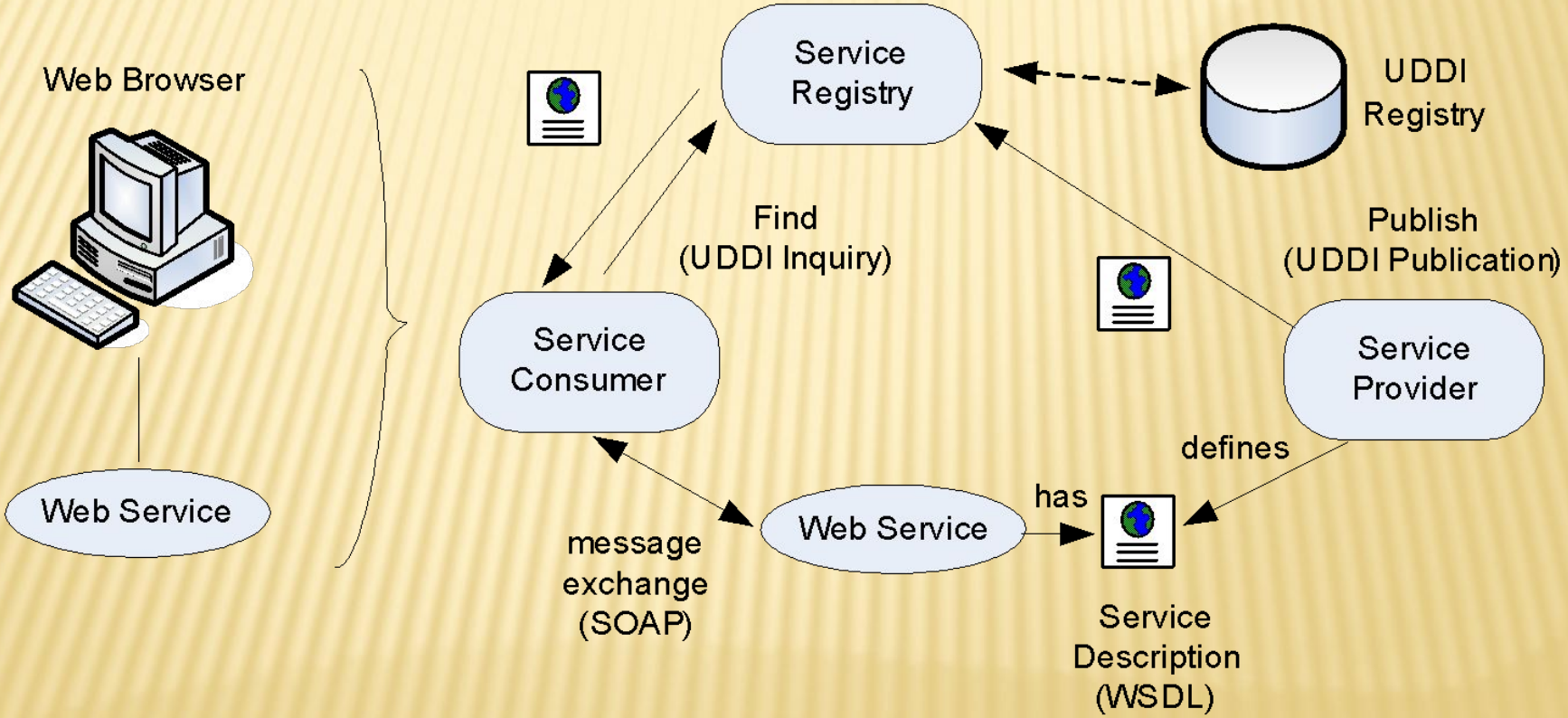
- Уровень бизнес-логики предприятия
- Уровень логики приложения
- Уровень приложения

ВЕБ СЕРВИС



- **Веб-служба**, *веб-сервис* (англ. *web service*) — идентифицируемая программная система со стандартизированными интерфейсами.

МОДЕЛЬ РАБОТЫ WEB-СЕРВИСА

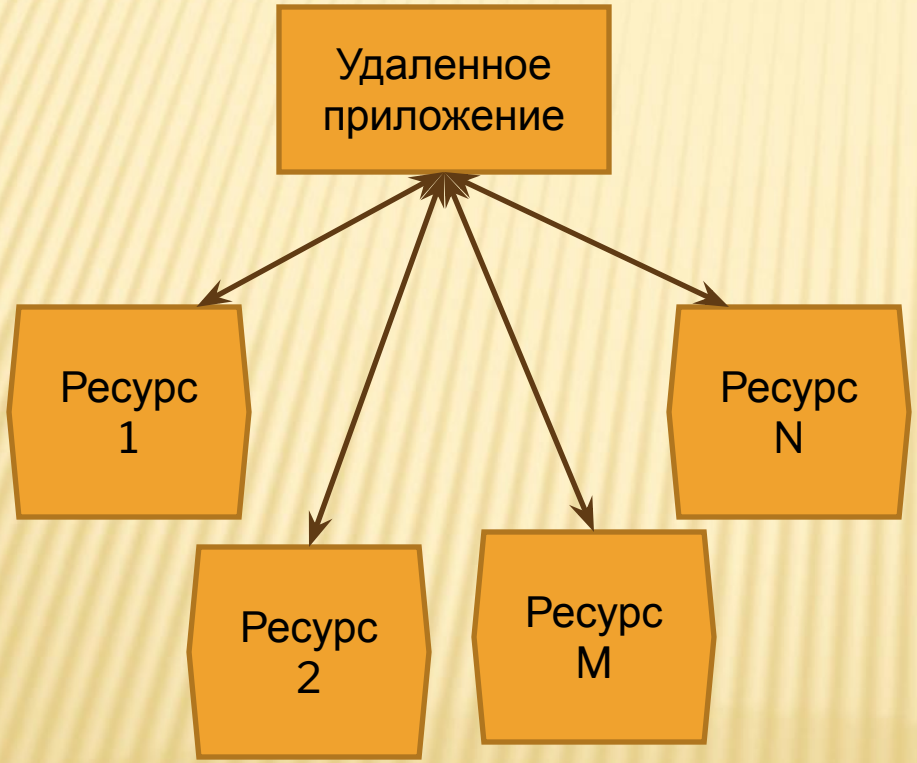


СТЕК ПРОТОКОЛОВ WEB-СЕРВИСА

АРХИТЕКТУРА ОСНОВАННАЯ НА РЕСУРСАХ

- Рост числа сервисов доступных через Web и создание распределенных систем на основе композиций сервисов привели к необходимости переосмысления архитектуры РС построенных на базе Web.
- Одной из проблем построения РС на основанных на Web сервисах явилась высокая сложность обеспечения связи между большим числом различных компонент.
- В качестве альтернативы было предложено рассматривать РС как коллекцию ресурсов каждый из которых индивидуально управляется своим компонентом.

РЕСУРСО-ЦЕНТРИЧЕСКАЯ АРХИТЕКТУРА



- Ресурсы могут добавляться либо удаляться с помощь приложения (удаленного).
- Такой подход привел формированию понятия ресурсо-центрической архитектуры РИС
- Ресурсный подход получил широкое распространение в WEB в виде Web-сервисов RESTful (Representational State Transfer)

ПРИНЦИПЫ АРХИТЕКТУРЫ RESTFUL

- Архитектура REST (Representational State Transfer) основана на четырех принципах [Fielding, 2000]:
  1. Использование **единой схемы именования**. Идентификация ресурса выполняется посредством URI, который предоставляет глобальное адресное пространство для поиска ресурсов и сервисов.
  2. **Унифицированный интерфейс** – GET извлекает текущее состояние ресурса в некотором представлении. POST передает новое состояние ресурса. Поддерживается всего 4 операции:
     - PUT – создать новый ресурс;
     - GET – получить состояние ресурса в некоторое представление;
     - DELETE – Удалить ресурс;
     - POST – Модифицировать ресурс с помощью изменения его состояния.
  3. **Информативные сообщения** – являются самодостаточными. Ресурсы отделены от их представления таким образом, что их содержимое может быть доступно в различных форматах (например, HTML, XML, текст, RDF, JPEG).
  4. **Отсутствие сессии**. После формирования ответа сервер забывает о клиенте. Взаимодействия через гиперссылки – Для обмена существуют различные технологии (например, переименование URI, cookies и скрытые поля формы). Состояние может быть вставлено в ответное сообщение, чтобы указать допустимое будущее состояние взаимодействия.

СРАВНЕНИЕ REST И SOAP/XML WEB-СЕРВИСОВ

1. RESTful архитектура стала популярна благодаря своей простоте.
2. SOAP поддерживает 16 операций, RESTful всего 4.
3. В случае RESTful приложение должно предоставить все параметры запроса с помощью одной операции, а в случае SOAP число передаваемых параметров за одну операцию ограничено, поэтому для передачи всех параметров требуется выполнение нескольких операций.

Например, для создания bucket в AWS S3 (SOAP/XML) надо выполнить:

```
import  bucket
bucket.create ("mybucket")
```

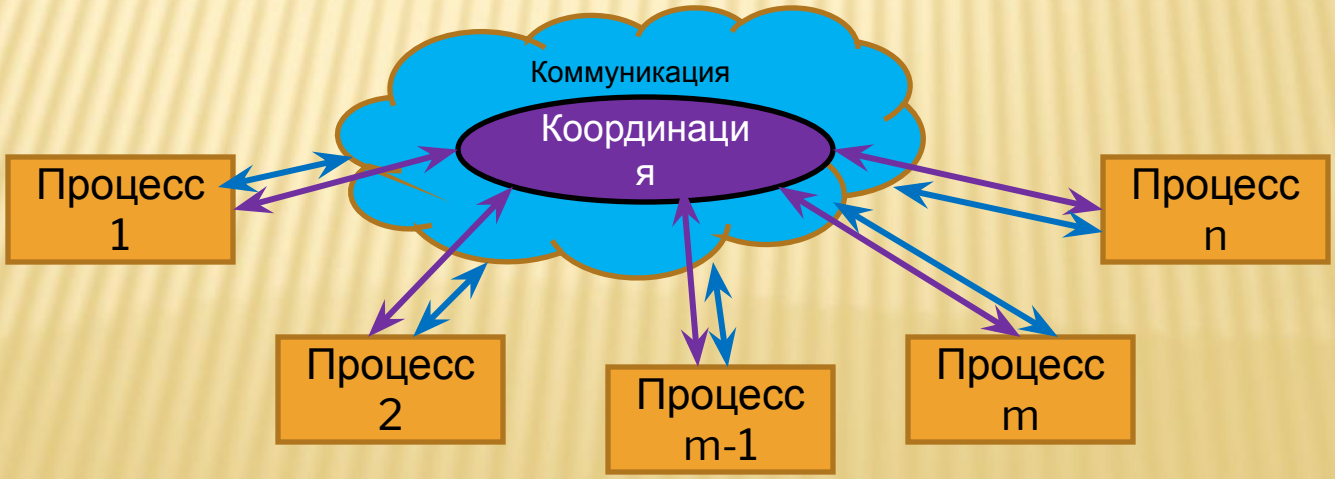
В случае RESTful:

```
PUT "http://mybucket.s3.amazonsws.com/"
```

- Вызов SOAP может порождать синтаксические ошибки, обнаруживаемые во время компиляции, в то время как в случае REST проверка вызова откладывается на время исполнения

АРХИТЕКТУРЫ ОСНОВАННЫЕ НА ПУБЛИКАЦИИ И ПОДПИСКЕ

- По мере роста размеров РС стало важным иметь архитектуру в которой зависимость между процессами стала бы как можно меньше.
- В качестве таковой была предложена архитектура в которой было введено строгое разграничение между процессами передачи и обработки сообщений и координацией этих процессов.
- Идея состояла в том, что бы взглянуть на РС как на совокупность процессов обработки информации взаимодействующих между собой.
- В этой модели координация заключается в коммуникации и координации между процессами. Координация играет роль клея связывающего активность процессов в единое целое.



СПОСОБЫ КООРДИНАЦИИ

- В зависимости то вида координации используемой в системе можно определить несколько моделей архитектуры подписка/публикация.
- В качестве способов координации используются:
  - Временная координация;
  - Ссылочная (адресная) координация.

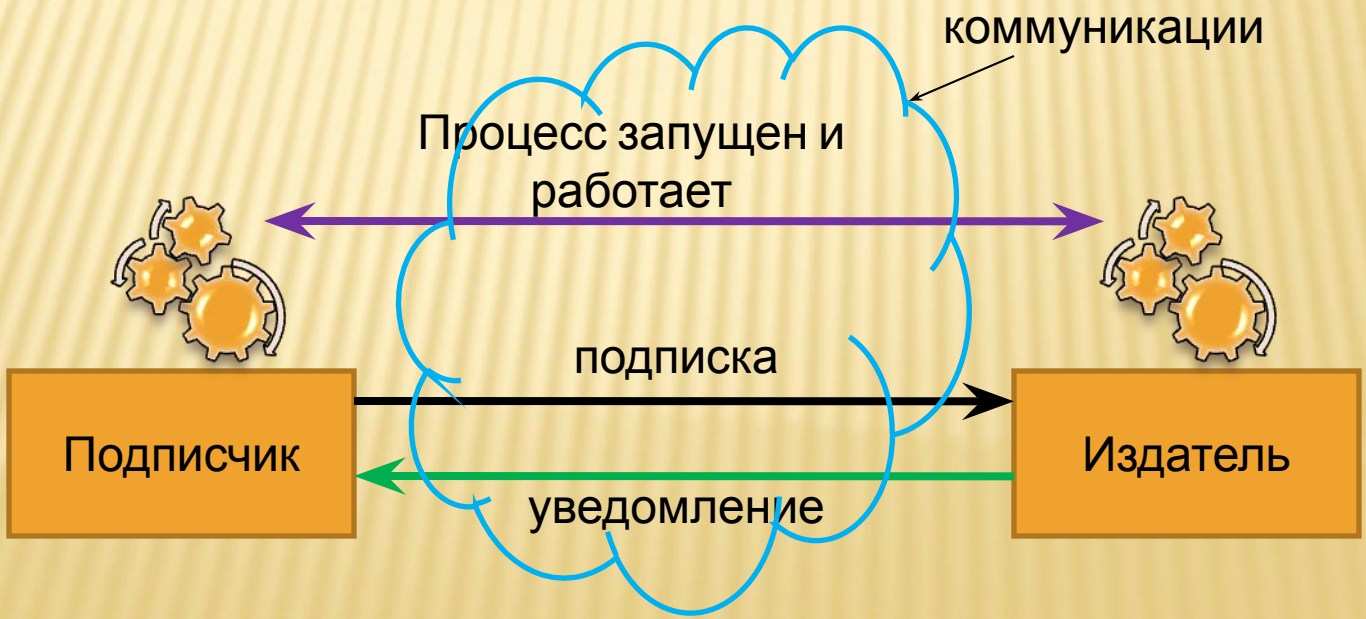
ВАРИАНТЫ АРХИТЕКТУР ПУБЛИКАЦИЯ/ПОДПИСКА

- В зависимости от комбинации двух факторов координации различают следующие модели архитектуры публикация/подписка:
  - архитектура П/П с прямой координацией;
  - архитектура П/П с координацией через почтовый ящик;
  - архитектура П/П с координацией на основе событий;
  - архитектура П/П с координацией на основе разделяемых данных.

	Координация во времени	**Координация несвязанная по времени**
Координация по ссылкам	Прямая связь	Связь через почтовый ящик
Координация не связанная с наличием ссылок	Связность на основе событий	Связь через разделяемое пространство данных (файл/база)

АРХИТЕКТУРА П/П С ПРЯМОЙ КООРДИНАЦИЕЙ

- В этой модели одновременно используются оба вида координации:
  - Координация по времени существует, когда оба процесса запущены и работают).
  - Координация по ссылке (адресная) существует, когда имеется явная ссылка на процесс с которым требуется взаимодействие, либо в виде имени либо в виде его идентификатора.

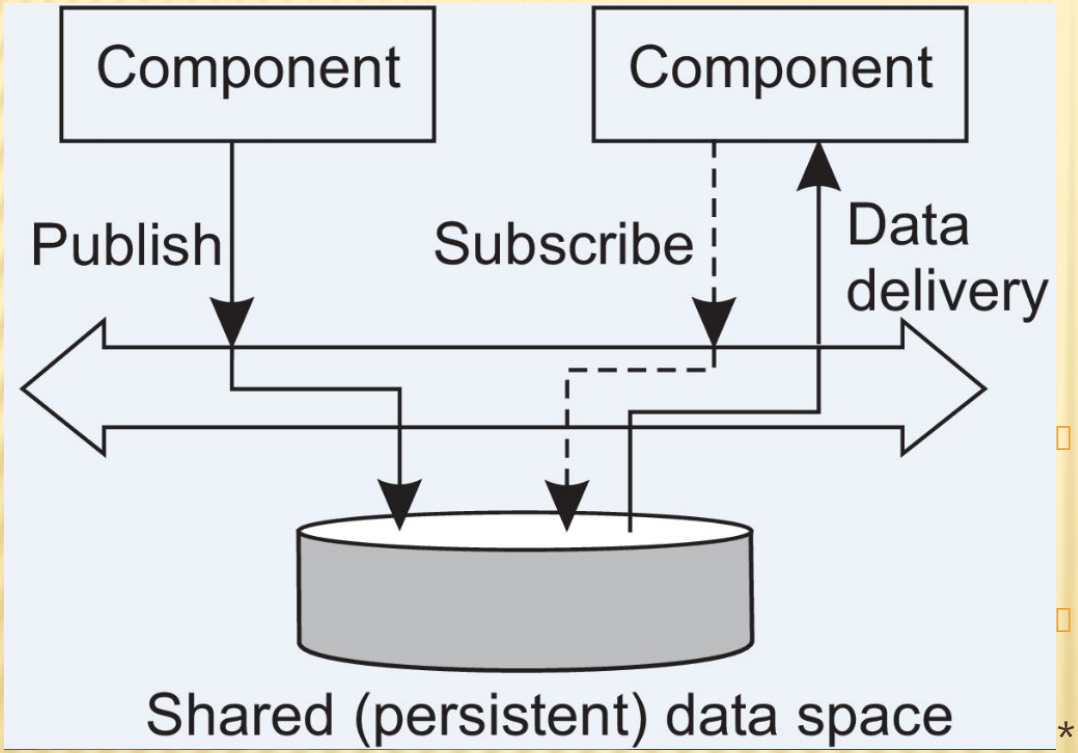


АРХИТЕКТУРА П/П С КООРДИНАЦИЕЙ ЧЕРЕЗ ПОЧТОВЫЙ ЯЩИК

- В этой модели:
  - Отсутствует координация по времени, оба процесса запускаются и работают не синхронизируясь друг с другом.
    - ✔ Подписчик периодически проверяет наличие сообщений в своем почтовом ящике.
    - ✔ Отправитель (издатель) периодически активизируется для отправки сообщения в почтовый ящик другого процесса.
  - Координация по ссылке (адресная) существует, в виде адреса почтового ящика.

АРХИТЕКТУРА НА ОСНОВЕ ОБЩЕГО ПРОСТРАНСТВА ДАННЫХ

В случае отсутствия обоих видов координации получается модель с общей шиной данных.



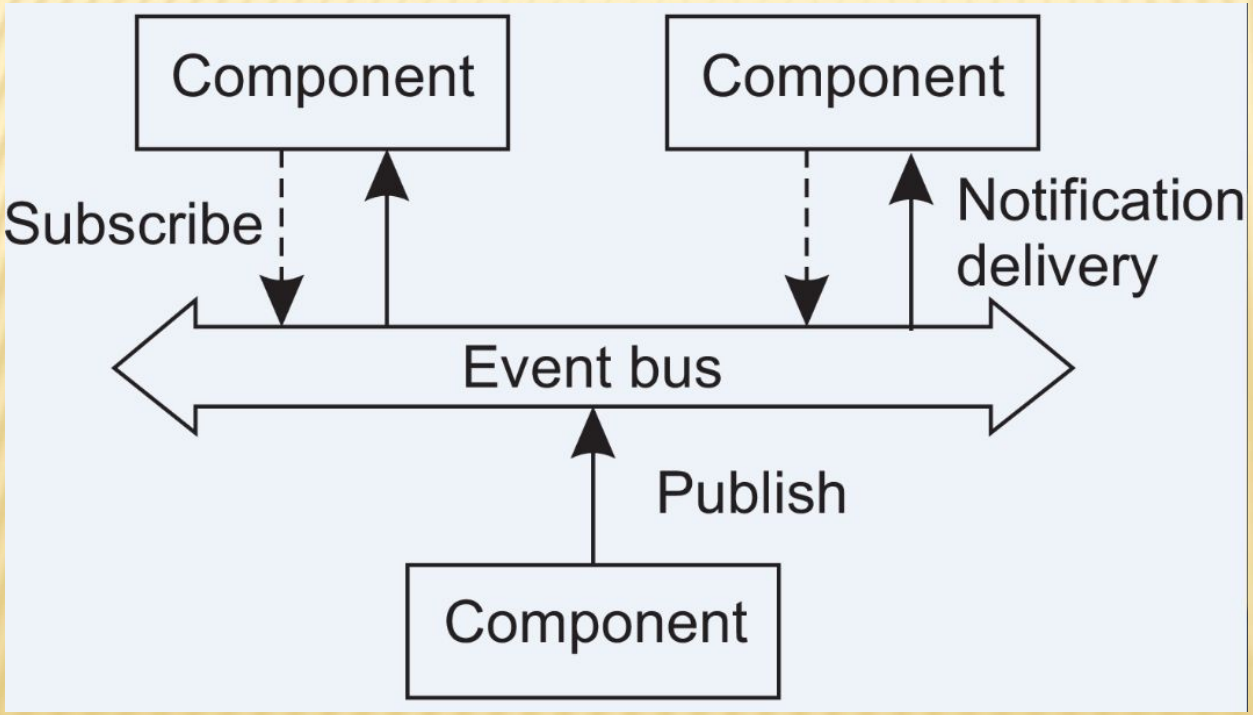
- Пример: Linda - программная модель, разработанная в 1980 году. Разделяемое пространство данных в модели Linda называется пространство записей (кортежей). Это пространство поддерживает три операции:
- `in(t)`: извлечь (с удалением) запись, соответствующую шаблону t;
- `out(t)`: занести запись по шаблону t.

* Операции `in(t)` и `out(t)` взаимно блокируют друг друга.

- `zd (t)`: получить копию записи по шаблону t;

АРХИТЕКТУРА П/П НА ОСНОВЕ СОБЫТИЙ

В этой модели ссылочная координация отсутствует, но поддерживается во времени.



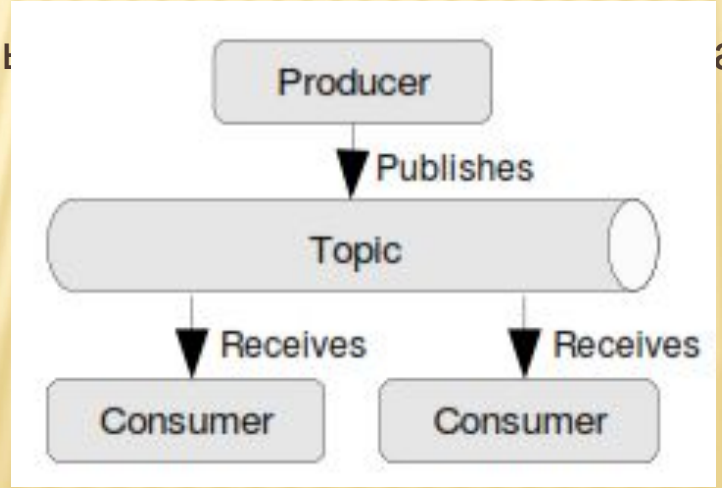
- Для получения уведомления процесс подписчик должен всегда находиться в рабочем состоянии.
- Отсутствие ссылочной информации не позволяет процессам иметь явные знания друг о друге.
- Такая архитектура соответствует коммуникациям через общую шину сообщений.

ВАРИАНТЫ АРХИТЕКТУРЫ П/П НА ОСНОВЕ СОБЫТИЙ

- В зависимости от способа описания события различают два варианта систем с архитектурой публикации/подписке на основе событий:
  - системы публикации/подписки основанные на теме (topic-based public-subscribe systems);
  - системы публикации/подписки основанные на содержимом (контенте) (content-based public-subscribe systems)

СИСТЕМА ПУБЛИКАЦИИ/ПОДПИСКИ ОСНОВАННАЯ НА ТЕМЕ

- Событие описывается набором атрибутов – списком пар (атрибут, значение).
- Подписка должна быть направлена к промежуточному ПО с описанием события (список пар атрибутов и их значений), в котором заинтересован подписчик.
- Уведомление описывает опубликованное событие, когда оно становится доступным другим процессам для чтения. Оно также должно содержать список пар атрибутов и их значений, характеризующих событие.
- Получив уведомление подписчик определяет соответствует ли событие подписке.
- Такая система назы а теме.

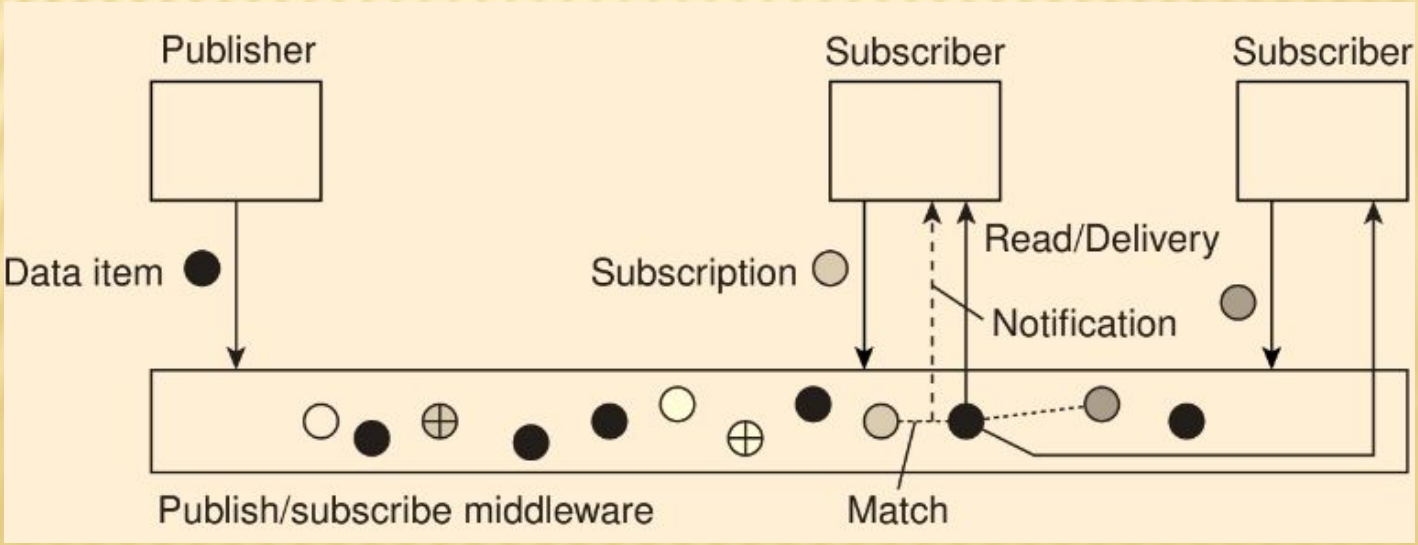


СИСТЕМЫ ПУБЛИКАЦИЙ/ПОДПИСКИ ОСНОВАННЫЕ НА СОДЕРЖИМОМ (КОНТЕНТЕ)

- В этом случае событие также описывается набором атрибутов, которые представляются парами: имя_атрибута, диапазон_значений.
- Допускается использовать все виды предикатов на основе множества атрибутов (подобно запросам SQL).
- Уведомление описывает опубликованное событие, когда оно становится доступным другим процессам для чтения.
- Подписка должна быть направлена к промежуточному ПО с описанием события, в котором заинтересован подписчик.
- Очевидно, что чем более сложным является описание события, тем более трудно проверить событие на соответствие этого описания.

ПРИНЦИП ОБМЕНА ДАННЫМИ МЕЖДУ ИЗДАТЕЛЕМ И ПОДПИСЧИКОМ В СИСТЕМАХ С ШИНОЙ СОБЫТИЙ

- Условием обмена данными является совпадение подписки на событие и уведомление о событии.
- Во многих случаях событие на самом деле соответствует данным, ставшим доступными. В этом случае при совпадении имеется два сценария:
  - Программное обеспечение промежуточного уровня может решить направить подписчикам уведомления вместе с ассоциированными с этим событием данными. Это называется процессом с совпадением подписки.
  - Программное ПУ передает только уведомление. При этом подписчики смогут самостоятельно считать требуемые данные. ПО ПУ не предоставляет услуг по хранению данных. Услуги хранения оказываются отдельным сервисом.



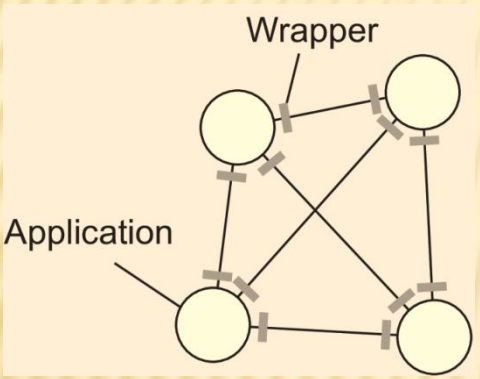
ОСНОВНАЯ ПРОБЛЕМА СИСТЕМ ПУБЛИКАЦИИ/ПОДПИСКИ

- События могут легко запутать работу подписчиков. В качестве примера рассмотрим такую подписку: "Уведомить, когда номер 1060 будет освобожден и дверь в номер будет не заперта".
- Обычно распределенная система, поддерживающая такие подписки, может быть реализована путем установки независимых сенсоров (датчиков) для мониторинга занятости номера (например, датчики движения) и регистрации состояния дверного замка.
- Для обеспечения надежной работы такой системы, необходимо сформировать (скомпоновать) такие примитивные события в публикуемые данные, на которые можно было бы подписать процессы получатели данных.
- Компоновка событий оказывается сложной задачей, особенно когда простейшие (примитивные) события генерируются источниками, распределенными по всей системе.
- Очевидно, что в системах публикации/подписки подобным этой, основная проблема кроется в эффективности и масштабируемости реализации сравнения подписок и уведомлений.

ОРГАНИЗАЦИЯ ПРОМЕЖУТОЧНОГО ПО (MIDLEWARE)

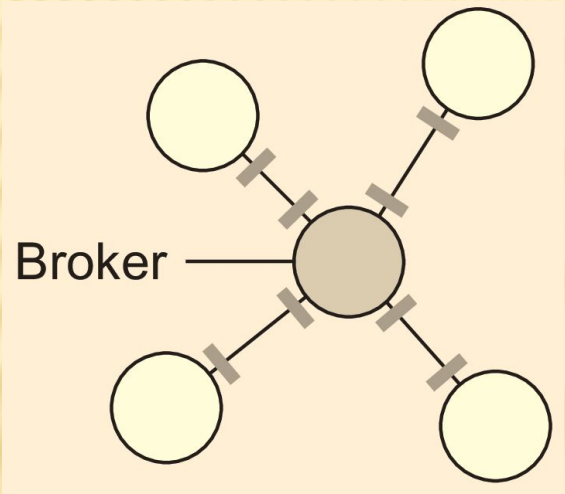
- При организации ПО промежуточного уровня в системах основанных на объектных компонентах, часто используются два шаблона проектирования:
  - Оболочки (Wrappers).
  - Перехватчики (Interceptors).
- Их применение направлено на достижение открытости системы.

WRAPPERS ИЛИ ADAPTERS



- При создании распределенных проблем на основе уже существующих компонент мы сразу же сталкиваемся с фундаментальной проблемой:
  - Интерфейсы предлагаемые унаследованными компонентами, не поддерживаются всеми компонентами.
- Оболочка или адаптер – это специальные компоненты которые обеспечивают приемлемый интерфейс для клиента приложения. Функциями адаптера является преобразование интерфейса компонента-сервер в вид удобный для компонента –клиента.
- Wrapper реализуется как компонент посредник, обеспечивающий приложению возможность вызова удаленного объекта.
- При необходимости обеспечить взаимодействие между N компонентами потребуется создать $N(N-1)=O(N^2)$ адаптеров.

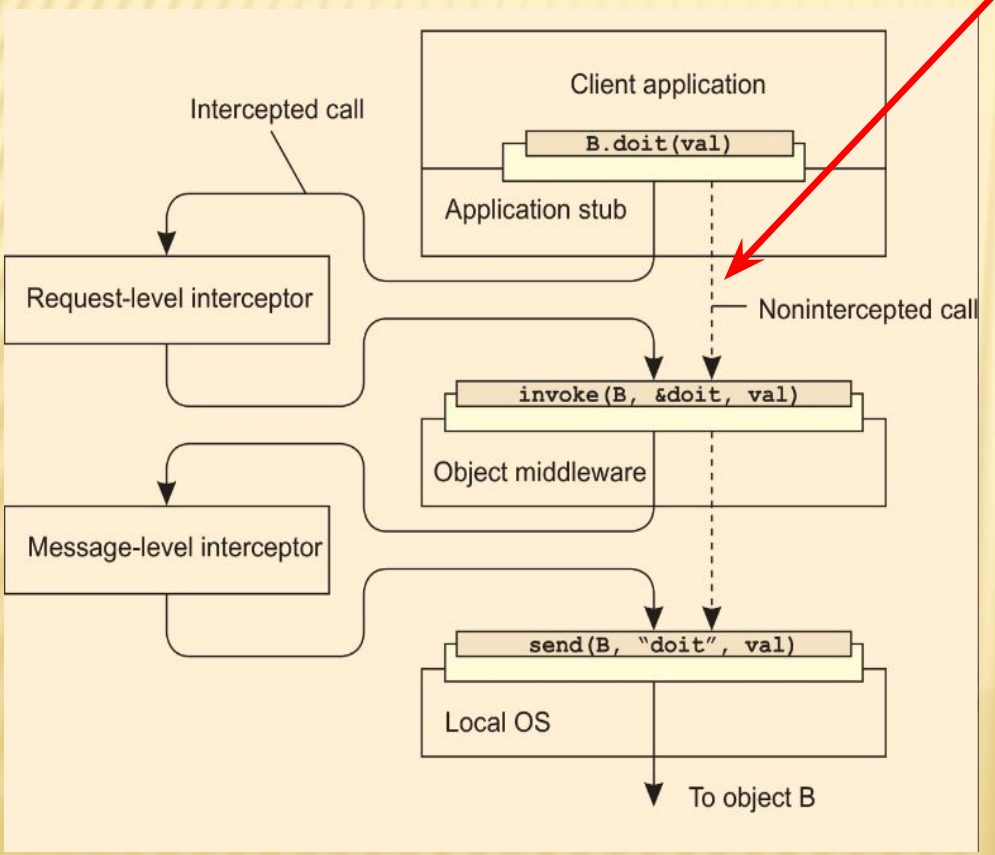
БРОКЕР СООБЩЕНИЙ



- Уменьшить число адаптеров можно создав промежуточное ПО, которое обеспечит централизованное управление доступом между различными приложениями.
- Часто роль такого ПО выполняет брокер сообщений.
- Приложения просто посылают брокеру запросы, содержащие всю необходимую информацию для доступа к другому приложению.
- В свою очередь брокер, зная как подключиться к требуемому приложению, обращается к нему и получив ответ переправляет его к приложению инициировавшему запрос. В этом случае необходимо реализовать только $2N=O(N)$ адаптеров, вместо $O(N^2)$.

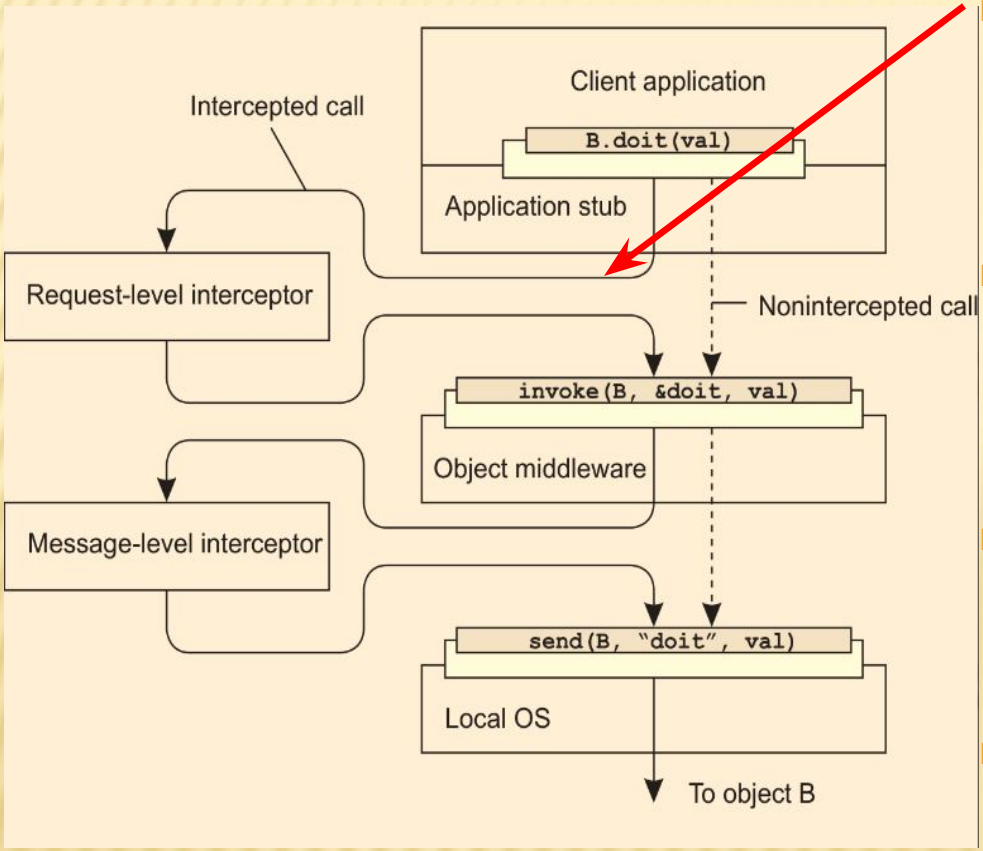
ПЕРЕХВАТЧИКИ ОБРАЩЕНИЙ (INTERCEPTORS)

- Концептуально перехватчик обращений, прерывает нормальный процесс вызова компонент и позволят исполнить другой фрагмент кода, исходя из нужд приложения.



Базовая идея проста (не перехватываемый вызов):
Объект А вызывает метод принадлежащий объекту В, но объект В располагается на другой машине.
Такой удаленный вызов метода выполняется за 3 шага:
1.Объект А предлагает, тот же интерфейс, что и объект В. Вызов метода описан в интерфейсе.
2. Вызов метода объекта В преобразуется к нужному виду промежуточным ПО находящемся на машине А.
3. И наконец, вызов объекта преобразуется в сообщение, посылаемое через сетевой интерфейс, как это определено локальной ОС А

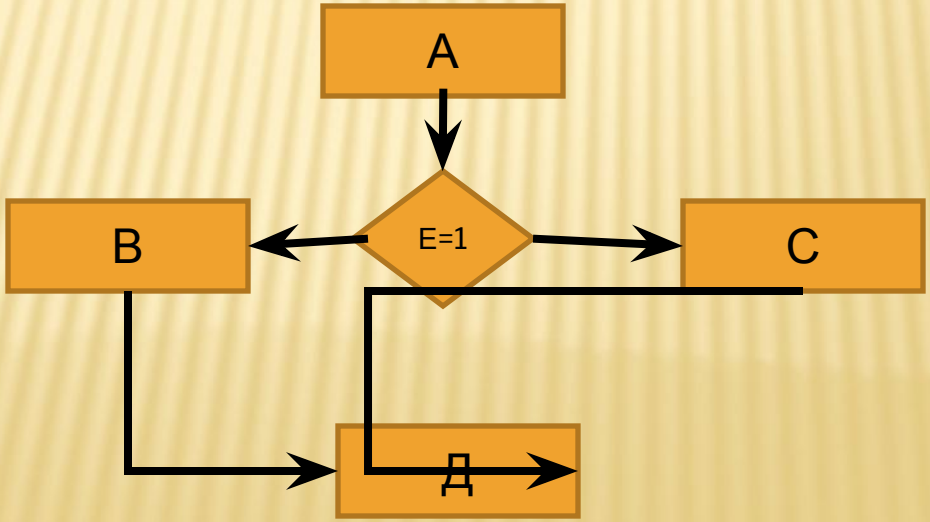
ОБРАЩЕНИЕ К РЕПЛИКАМ ОБЪЕКТА B



- Представим, что объект B имеет несколько реплик. В этом случае мы должны обратиться к каждой реплике. В этом может помочь перехватчик – запроса (**request-level interceptor**).
- Хотя объект A ничего не знает о других экземплярах B, но о них знает ПО промежуточного слоя объекта B, выполняющее роль перехватчика обращений уровня запроса.
- В конце концов вызов удаленного объекта должен быть передан по сети, для этого необходимо обратиться к интерфейсу локальной ОС A.
- На этом уровне используется перехватчик уровня сообщения (**message-level interceptor**) для передачи вызова удаленному объекту на машине B.

АДАПТИРУЕМОЕ ПО ПРОМЕЖУТОЧНОГО СЛОЯ

- Необходимость в адаптируемом промежуточном ПО возникает из-за того, что в окружении РС постоянно возникают изменения: отказ аппаратных средств; сбои электропитания; не соответствие предоставляемого сетью качества обслуживания, требуемому; перенос приложение в другое окружение и т.п.
- Вместо того, чтобы обрабатывать все эти изменения окружающей среды в приложении, эту обработку переносят на уровень промежуточного ПО.
- Такое сильное влияние окружения на РС требует от разработчиков разрабатывать промежуточного ПО устойчивое к этим изменениям. Такое ПО называют адаптивным, т.е. способным к изменению при изменении внешних условий, за счет целенаправленной его модификации, часто без перезагрузки.



ЗАМЕНА КОМПОНЕНТ ВО ВРЕМЯ ИСПОЛНЕНИЯ

- Примером такого подхода является **замена программных компонент** в момент их исполнения. Этот способ является одним наиболее широко распространенных среди других существующих подходов используемых для создания адаптируемого ПО промежуточного слоя.
- Проектирование на основе объектных компонент поддерживает видоизменяемость на основе изменения компоновки компонент.
- Система может быть сконфигурирована **статически на этапе проектирования** либо **динамически во время исполнения кода**. В последнем случае требуется поддержка позднего связывания не только в языках программирования, но и со стороны операционной системы, когда выполняется загрузка или выгрузка модулей.

СИСТЕМНАЯ АРХИТЕКТУРА

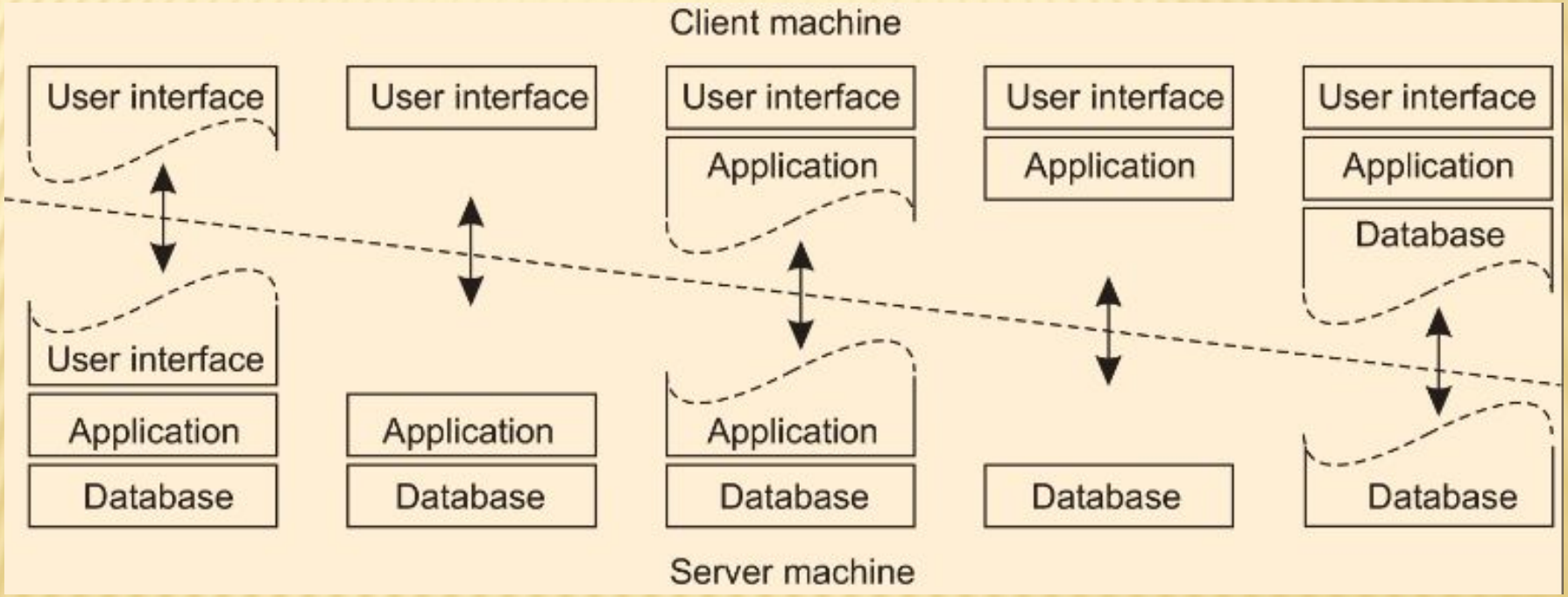
ЦЕНТРАЛИЗОВАННАЯ ОРГАНИЗАЦИЯ PC

ВАРИАНТЫ АРХИТЕКТУРЫ КЛИЕНТ-СЕРВЕР

- Простейшая организация предполагает наличие всего двух типов машин.
  - Клиентские машины, на которых имеются программы, реализующие только пользовательский интерфейс или его часть.
  - Серверы, реализующие все остальное, то есть уровни обработки и данных.
- На самом деле такая система не является распределенной: все происходит на сервере, а клиент представляет собой не что иное, как простой терминал.

ФИЗИЧЕСКИ ДВУХЗВЕННЫЕ АРХИТЕКТУРЫ

- Один из подходов к организации клиентов и серверов — это распределение программ, находящихся на уровне приложений, по различным машинам.



Первые три варианта соответствуют модели с тонким клиентом.
Варианты 4,5 называются моделью клиент-сервер с толстым клиентом.

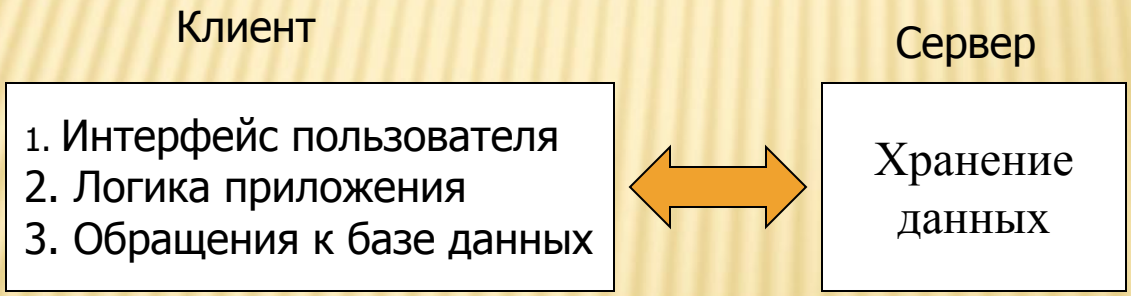
РАЗНОВИДНОСТИ МОДЕЛИ КЛИЕНТ-СЕРВЕР

Обычно ПО хранения данных располагается на сервере (например, сервер базы данных), интерфейс с пользователем на стороне клиента, а обработку данных распределять между клиентской и серверной частями.

- **Толстый клиент.** Такая модель подразумевает объединение в клиентском приложении интерфейса пользователя и обработки данных. Серверная часть в этом случае представляет собой сервер баз данных.
- **Тонкий клиент.** В этом случае клиентское приложение обеспечивает интерфейс с пользователем, а сервер объединяет модули хранения и обработки (толстый сервер).
- **Многоуровневые системы клиент-сервер.** В этих системах некоторая часть функций, связанных с обработкой данных, либо с доступом к модулям хранения, либо с обеспечением многопользовательского доступа, выделяется в отдельный модуль (*mieddlware*), называемый ПО промежуточного слоя.

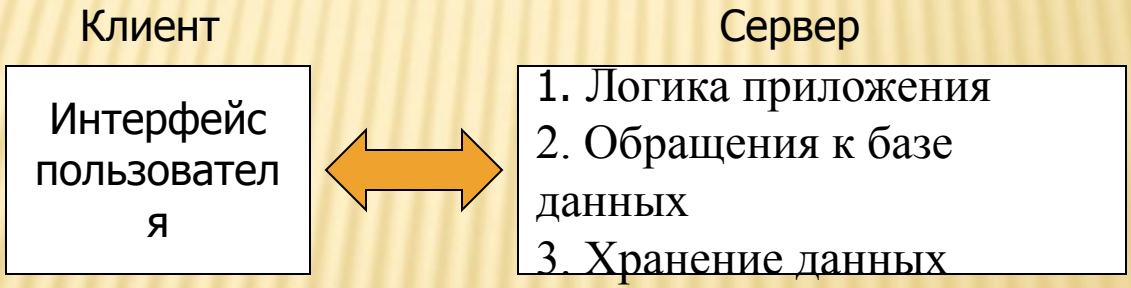
ТОЛСТЫЙ КЛИЕНТ

- Основным недостатком **толстого клиента** является сложность администрирования и трудности с обновлением ПО, поскольку его замену нужно производить одновременно на всех рабочих местах всей информационной системы.



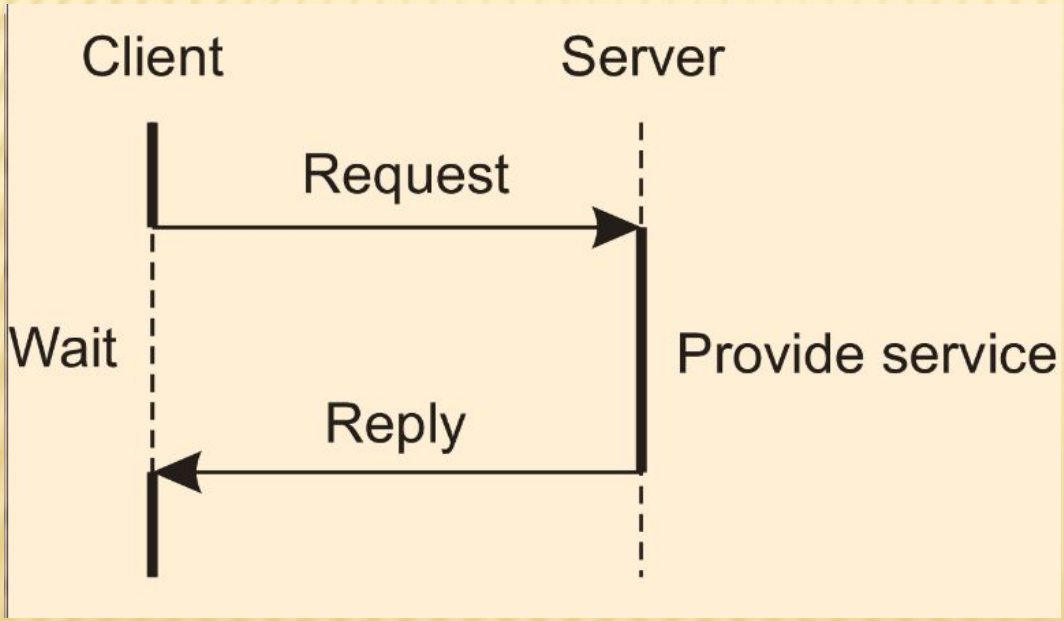
ТОНКИЙ КЛИЕНТ

- **В тонком клиенте** этот недостаток устраняется, однако появляются большие сложности в создании ПО серверной части, появляются трудности в объединении модулей обработки и хранения данных.



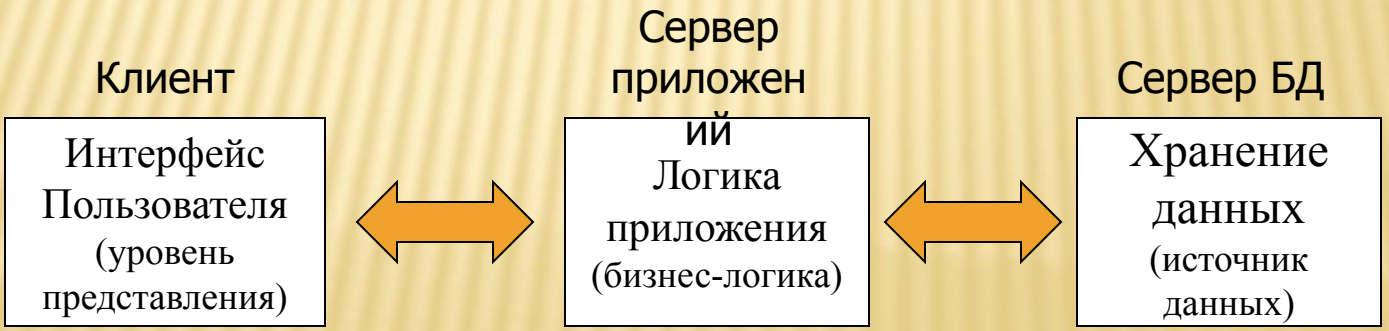
ВЗАИМОДЕЙСТВИЕ МЕЖДУ КЛИЕНТОМ И СЕРВЕРОМ

- Взаимодействие между клиентом и сервером расположенными на разных машинах часто называют «поведением запрос – ответ»



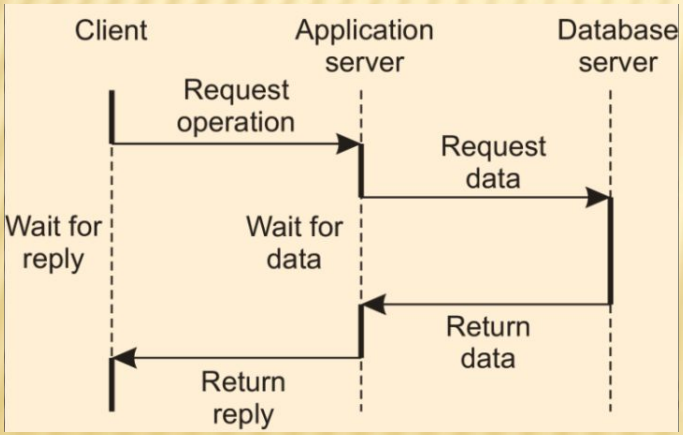
МНОГОУРОВНЕВЫЕ СИСТЕМЫ КЛИЕНТ-СЕРВЕР

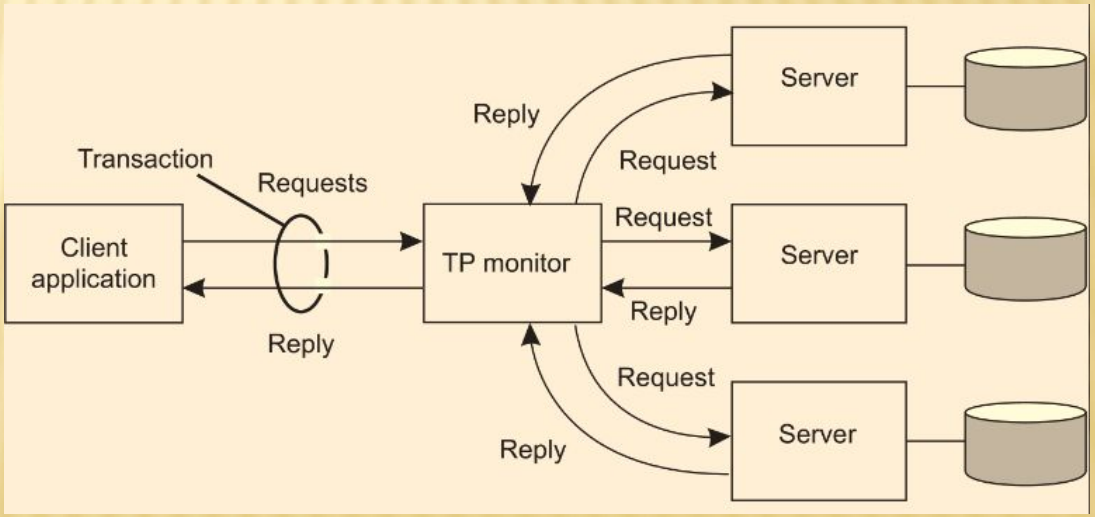
- **Многоуровневая архитектура клиент-сервер** позволяет более разумно распределить модули обработки данных, которые в этом случае выполняются на одном или несколько серверах. Эти программные модули выполняют функции сервера для интерфейса с пользователем и клиента - для серверов баз данных. Многоуровневая архитектура клиент-сервер позволяет
  - более точно назначить полномочия пользователей, т. к. они получат права доступа не к самой базе данных, а к определенным функциям сервера приложения.
  - Это повышает защищенность системы (по сравнению с обычной архитектурой) не только от умышленного нападения, но и от ошибочных действий персонала.



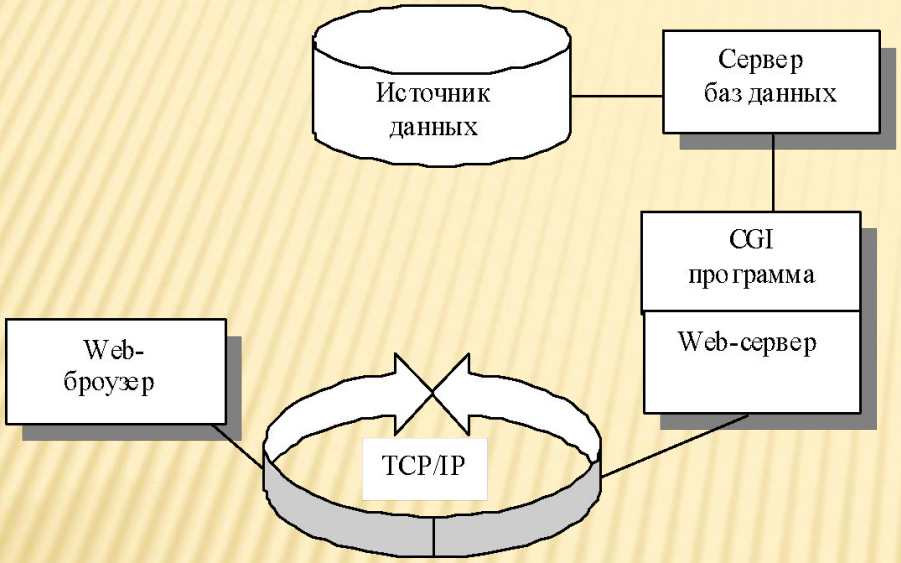
ПОВЕДЕНИЕ СЕРВЕРА КАК КЛИЕНТА В СЛОЖНЫХ ПРИЛОЖЕНИЯХ

- В классической модели клиент-сервер не предполагается поведение машины сервера, подобно машине клиента. Однако такая необходимость иногда возникает.
- В этом случае мы приходим к трехзвенной физической архитектуре клиент-сервер.
- В этой архитектуре, программы уровня обработки данных исполняются на отдельном физическом сервере. Типичным примером такой архитектуры являются системы обработки транзакций, в которых отдельный сервер, называемый монитором транзакций, координирует выполнение отдельных транзакций выполняемых на различных серверах данных.



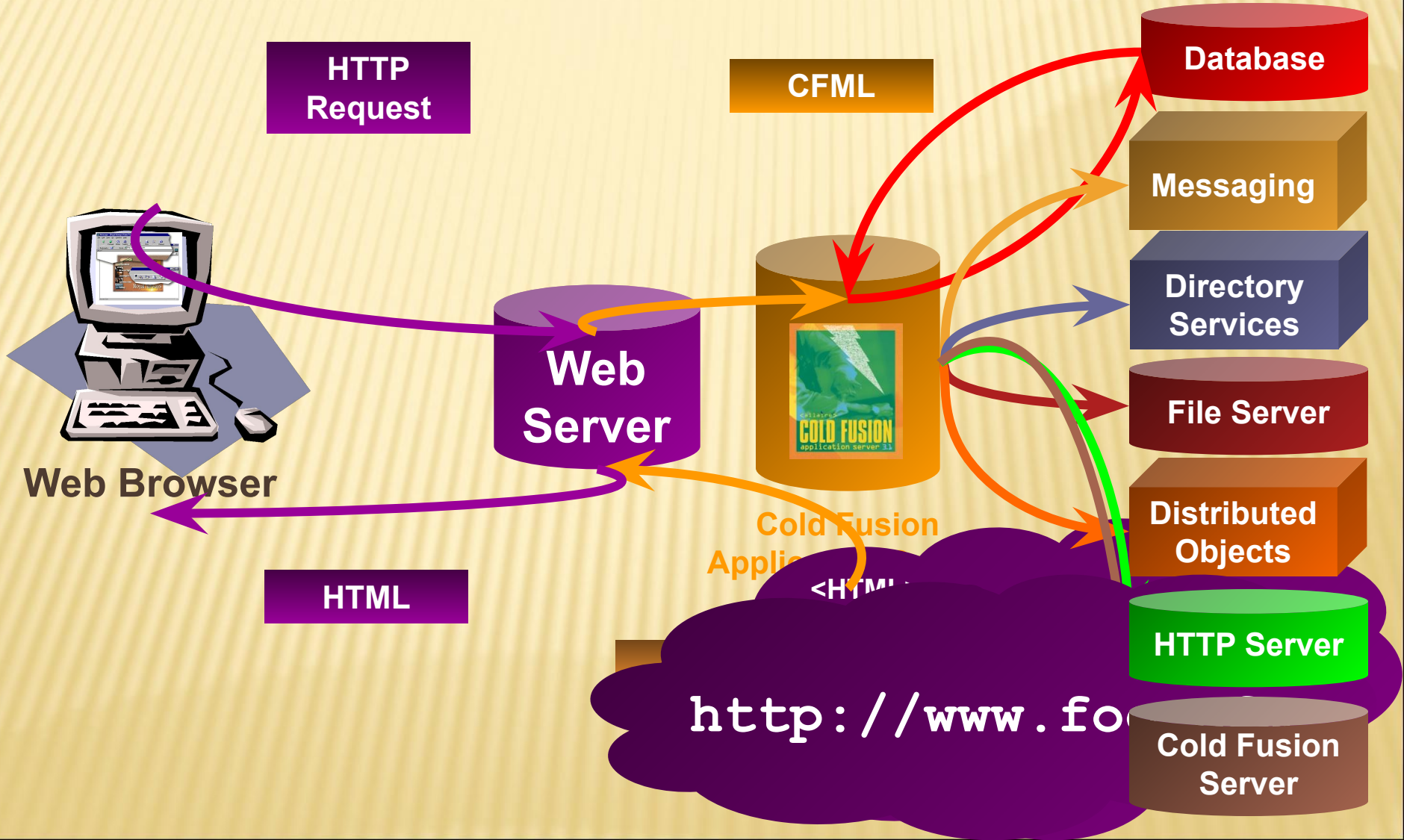


ПРИМЕР РАБОТЫ СЕРВЕРА В КАЧЕСТВЕ КЛИЕНТА В WEB ПРИЛОЖЕНИИ



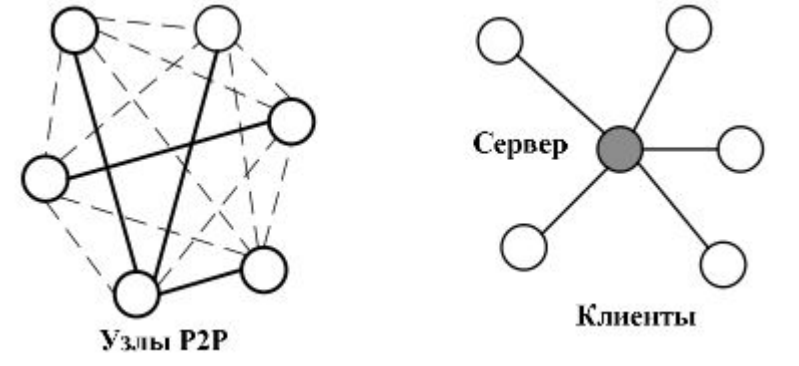
- Другой пример работа серверов приложений в среде WEB
- Такая архитектура Web приложения, также является В этом физической трехзвенной (**three-tiered**)

ТРЕХЗВЕННАЯ АРХИТЕКТУРА СЕРВЕРА ПРИЛОЖЕНИЙ COLD FUSION



ДЕЦЕНТРАЛИЗОВАННАЯ ОРГАНИЗАЦИЯ РС: СИСТЕМЫ PEER-TO-PEER (ОДНОРАНГОВЫЕ СИСТЕМЫ)

СРАВНЕНИЕ СИСТЕМ P2P И КЛИЕНТ-СЕРВЕР



В отличие от классической клиент серверной архитектуры в архитектуре P2P каждый узел, входящий в PC, может играть роль и клиента и сервера. Предоставляя свои либо используя чужие ресурсы.

Можно выделить следующие проблемы архитектуры клиент-сервер:

1. Проблемы масштабируемости.
2. Проблема высокой связности узлов.

P2P системы обладают следующими преимуществами:

1. Слабая зависимость от централизованных сервисов и ресурсов.
2. Система допускает сильные изменения в структуре, сохраняя при этом свою работоспособность.
3. Высокая масштабируемость

СРАВНЕНИЕ ПРИНЦИПОВ ПОСТРОЕНИЯ СИСТЕМ P2P И КЛИЕНТ-СЕРВЕР

Сравниваемые принципы организации систем:

- управление;
- доступность ресурсов;
- структура;
- мобильность ресурсов;
- время жизни;
- именование;
- синхронизация процессов.

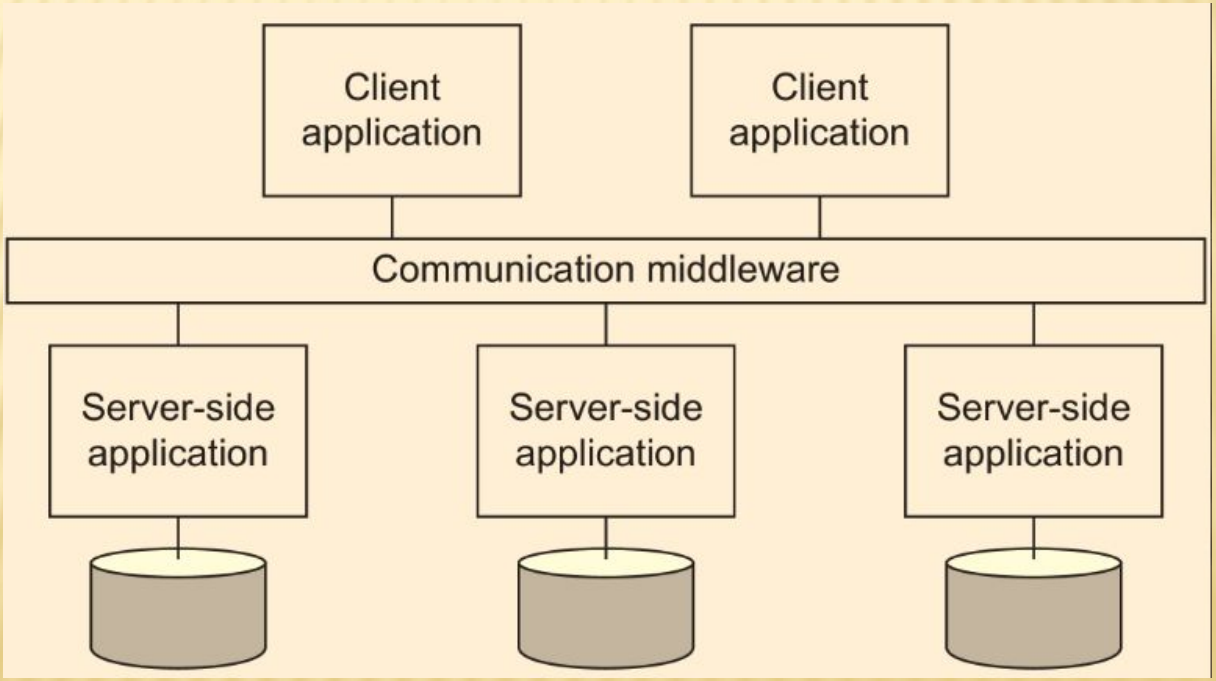
Четкой границы между централизованной и децентрализованной архитектурами не существует.

СПОСОБЫ РЕАЛИЗАЦИИ РАСПРЕДЕЛЕННОСТИ В СИСТЕМЕ

- Различают два способа реализации распределенности:
  - Вертикальная распределенность.
  - Горизонтальная распределенность.

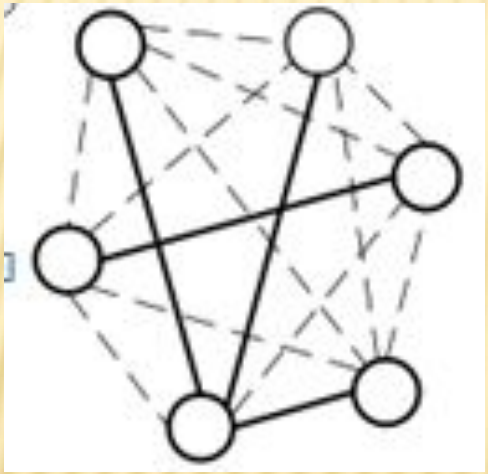
ВЕРТИКАЛЬНАЯ РАСПРЕДЕЛЕННОСТЬ

- Реализуется путем разбиения приложения на логические уровни связанные иерархически.
- Такая организация характерна для клиент серверных архитектур.
- Каждый логический уровень размещается на отдельном узле PC



ГОРИЗОНТАЛЬНАЯ РАСПРЕДЕЛЕННОСТЬ

- В этом случае клиент или сервер могут одновременно исполняться на одном и том же узле, разделяя между собой его ресурсы.
- Системы поддерживающие горизонтальную распределенность получили название пиринговых систем (**peer-to-peer systems**)



ЗАДАЧИ РЕШАЕМЫЕ С ПОМОЩЬЮ P2P СИСТЕМ (1)

1. *Уменьшение/распределение затрат.* Серверы централизованных систем, которые обслуживают большое количество клиентов, обычно несут на себе основной объем затрат ресурсов (денежных, вычислительных и др.) на поддержание вычислительной системы. P2P архитектура может помочь распределить эти затраты между узлами сети. Так как узлы, как правило, автономны, важно, чтобы затраты были распределены справедливо.
2. *Объединение ресурсов.* Каждый узел в P2P-системе обладает определенными ресурсами (вычислительные мощности, объем памяти). Приложения, которым необходимо большое количество ресурсов, например ресурсозатратные задачи моделирования или распределенные файловые системы, используют возможность объединения ресурсов всей сети для решения своей задачи. При этом важны как объем дискового пространства для хранения данных, так и пропускная способность сети.
3. *Повышенная масштабируемость.* Поскольку в сетях P2P отсутствует сильный центральный механизм, важной задачей является повышение масштабируемости и надежности системы. Масштабируемость определяет количество систем, которые могут быть достигнуты из одного узла, сколь- ко систем могут функционировать одновременно, сколько пользователей может пользоваться сетью, сколько памяти может быть использовано.

ЗАДАЧИ РЕШАЕМЫЕ С ПОМОЩЬЮ P2P СИСТЕМ (2)

3. Надежность сети определяется такими параметрами как количество сбоев в работе сети, отношение времени простоя к общему времени работы, доступностью ресурсов и т.д. Таким образом, основной проблемой становится разработка новых алгоритмов обнаружения ресурсов, на которых базируются новые P2P платформы
4. *Анонимность.* Бывает, пользователь не желает, чтобы другие пользователи или поставщики услуг знали о его нахождении в сети. При использовании центрального сервера трудно обеспечить анонимность, так как серверу, как правило, необходимо идентифицировать клиента, по крайней мере через интернет адрес. При использования P2P-сети пользователи могут избежать предоставления любой информацию о себе

ПРИНЦИПЫ ПОСТРОЕНИЯ РС P2P

- В распределенной системе P2P каждый узел участник знает некоторое число логических соседей с которыми он может поддерживать обмен напрямую, без посредников, посылая и получая в ответ сообщения по сети.
- Этот набор соседей формирует логический граф связности для всех узлов системы. Это граф часто называют **наложенной** сетью P2P (overlay network).
- Системы P2P должны соответствовать следующим критериям:
  - Самооганизация: система самостоятельно адаптируется к подключению или отключению узла к системе. Пиры используют локальную информацию получаемую от своих соседей для организации взаимодействия друг с другом.
  - Распределенность: отсутствует централизованное управление поведением узла в системе.
  - Масштабируемость: система может масштабироваться неограниченно избегая таким образом проблем «бутылочного горла», отказов отдельных узлов и проблем перегрузки узлов.

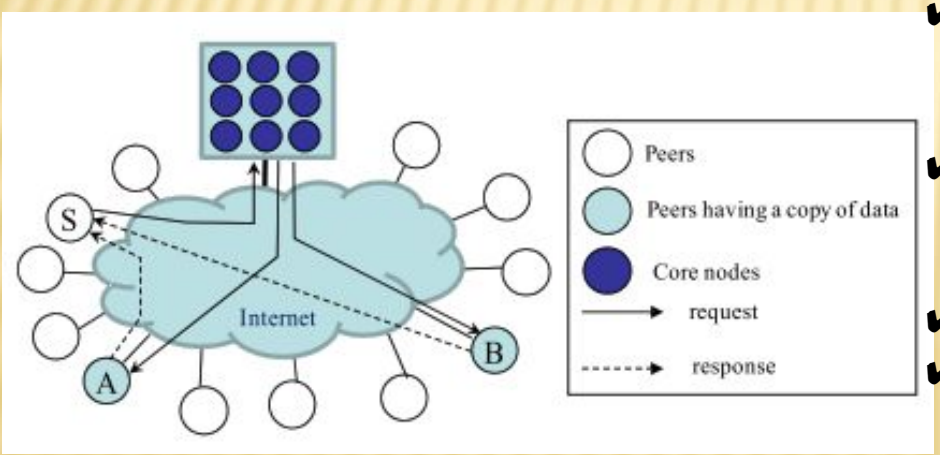
КЛАССИФИКАЦИЯ СИСТЕМ P2P

- Системы P2P можно классифицировать по двум признакам:
  - По степени централизации.
  - По наличию или отсутствию структуры.

КЛАССИФИКАЦИЯ P2P СИСТЕМ ПО СТЕПЕНИ ИХ ЦЕНТРАЛИЗАЦИИ

ЦЕНТРАЛИЗОВАННЫЕ P2P СИСТЕМЫ

- В этом случае некоторая группа узлов отвечает за выполнение критически важных для всей системы операций, например:
  - Аутентификация пиров в системе.
  - Определение расположения узлов и ресурсов принадлежащих узлам.
- Эти управляющие узлы могут располагаться в одной LAN, но могут быть разбросаны географически, но в любом случае между ними должны существовать широкополосные каналы связи.



Пример: Napster

✔Узел S посылает запрос на определение места размещения некоторых данных к управляющим узлам (core nodes).

✔На основе имеющегося у них глобального индекса управляющий узел форвардирует запрос к узлам A и B.

✔Узлы A и B отвечают напрямую S.

✔При отправке запроса S устанавливает таймер, по истечение которого, он выбирает узел из числа ответивших ему.

НЕДОСТАТКИ ЦЕНТРАЛИЗОВАННЫХ P2P СИСТЕМ

- Управляющие сервера являются потенциальными точками отказа.
- Такое решение является плохо масштабируемым.
- Эти недостатки отсутствуют в частично централизованных системах.

ДЕЦЕНТРАЛИЗОВАННЫЕ АРХИТЕКТУРЫ P2P СИСТЕМ

- При очень большом числе пиров гибридные системы не способны обеспечить корректную обработку всех запросов. Они перегружены запросами и не способны их обработать за приемлемое время.
- В децентрализованных системах каждый пир принадлежащий к наложенной сети обеспечивает обзор ресурсов только у своих логических соседей.
- Для обеспечения устойчивости к отказам отдельных узлов информация о ресурсах принадлежащих каждому пиру реплицируется всем его соседям по наложенной сети.
- Децентрализованная система не имеет точек единственного отказа, однако она менее эффективна чем соответствующая централизованная система.

ПРИМЕР ДЕЦЕНТРАЛИЗОВАННОЙ P2P GNUTELLA

- В системе Gnutella поиск и предоставление сервисов производится путем процедуры пошагового поиска, в которой могут участвовать все узлы, входящие в сеть.
- Каждый узел содержит таблицу соседей, содержащую IP адрес и порт известного узла Gnutella.
- При запуске новый узел Gnutella переходит в режим начальной загрузки, в котором посредством одного из доступных источников (список узлов на одном из узлов интернет; внутренний предустановленный список узлов и др.) формирует начальный список соседей.
- После чего, соседям высылается сообщение обнаружения, пересылаемое далее по цепочке систем. Таким образом обеспечивается обнаружение ресурсов, предоставляемых всеми узлами подключенными к сети.

КЛАССИФИКАЦИЯ P2P СИСТЕМ ПО ИХ СТРУКТУРЕ

СТРУКТУРИРОВАННЫЕ P2P СИСТЕМЫ

- Структура P2p системы определяется структурой наложенной сети, которая может иметь одну из известных топологий:
  - кольцо;
  - Двоичное дерево;
  - Решетка;
  - И т.п.
- Эта топология используется для эффективного поиска данных (ресурсов).

ИНДЕКС РЕСУРСА СТРУКТУРИРОВАННОЙ P2P СИСТЕМЫ

- Характеристикой структурированной P2P системы является так называемый индекс, который однозначно определяет каждый экземпляр данных поддерживаемый в системе и соответственно каждый узел располагающий каким-либо ресурсом.
- Соседство в системе определяется по близости значений индексов у узлов.
- В качестве такого индекса используется ключ хэш функции:

Key(data item) = hash(data item's Value).

РАСПРЕДЕЛЕННАЯ ХЭШ ТАБЛИЦА (DISTRIBUTED HASH TABLE)

- Каждому узлу структурированной P2P системы назначается идентификатор из одного того же набора значений хэша. И каждый узел становится ответственным за хранение данных, связанных (ассоциированных) с определенным набором ключей.
- По сути, система описывается с помощью таблицы хэшей - распределений хэш таблицы (DHT - Distributed Hash Table).

ПОИСК УЗЛА СТРУКТУРИРОВАННОЙ P2P СИСТЕМЫ

- Для того чтобы найти узел, ассоциируемый с ключом, необходимо выполнить операцию поиска узла по ключу:

  node = lookup (key).

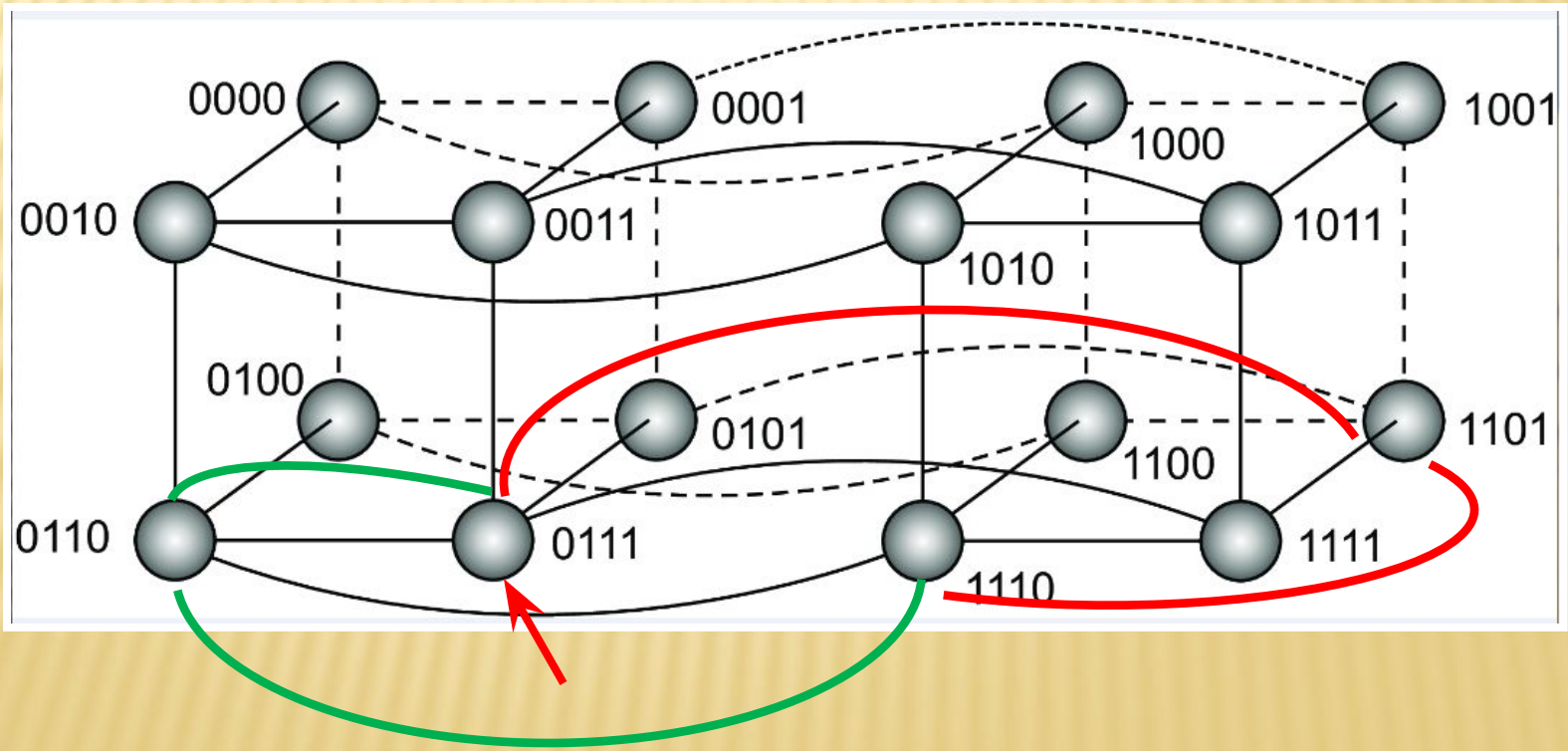
- В задаче поиска топология наложенной сети играет ключевую роль.
- Любой узел однозначно связан с некоторым ключом, что позволяет организовать эффективную маршрутизацию для поиска узла, ответственного за хранение ресурса.

ПРИМЕР СТРУКТУРИРОВАННОЙ P2P СИСТЕМЫ - ГИПЕРКУБ

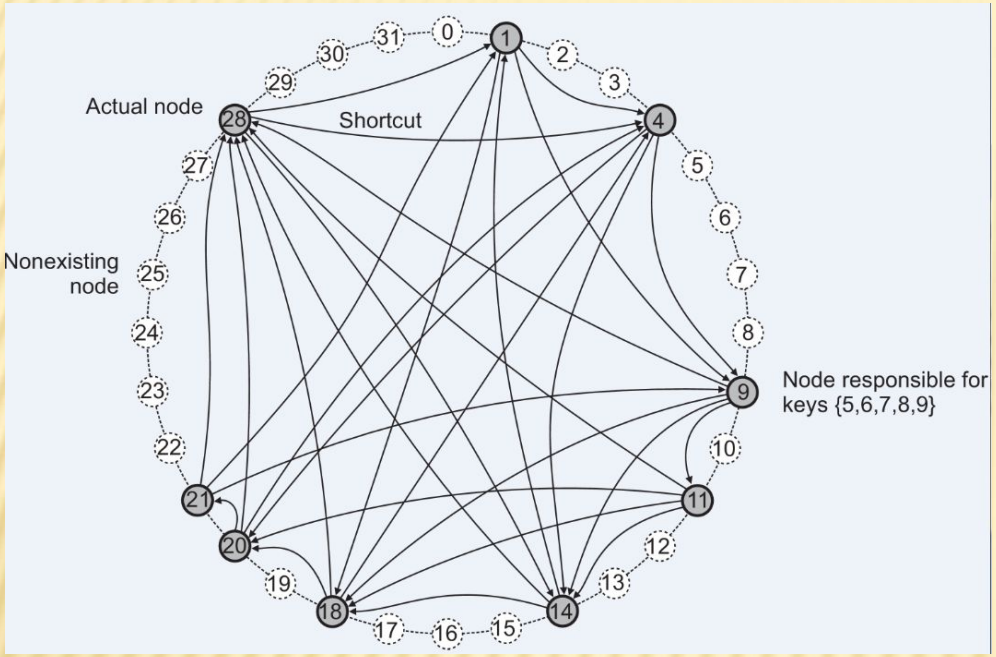
- Примером простой P2P системы с ограниченным числом узлов может служить гиперкуб.
- Гиперкуб - это n-размерный куб. Топология гиперкуба при n=4 следующая.
- Экземпляры данных связаны с узлами, которые в гиперкубе представляются вершинами.
- Каждой вершине (узлу) устанавливается идентификатор от 0 до 2^{4-1}.

ПОИСК ПО ГИПЕРКУБУ

- Предположим, узел 0111(7) ищет данные, имеющие идентификатор 1110(14).



ПОИСК В СИСТЕМЕ С ТОПОЛОГИЕЙ КОЛЬЦО (СИСТЕМА ХОРД)



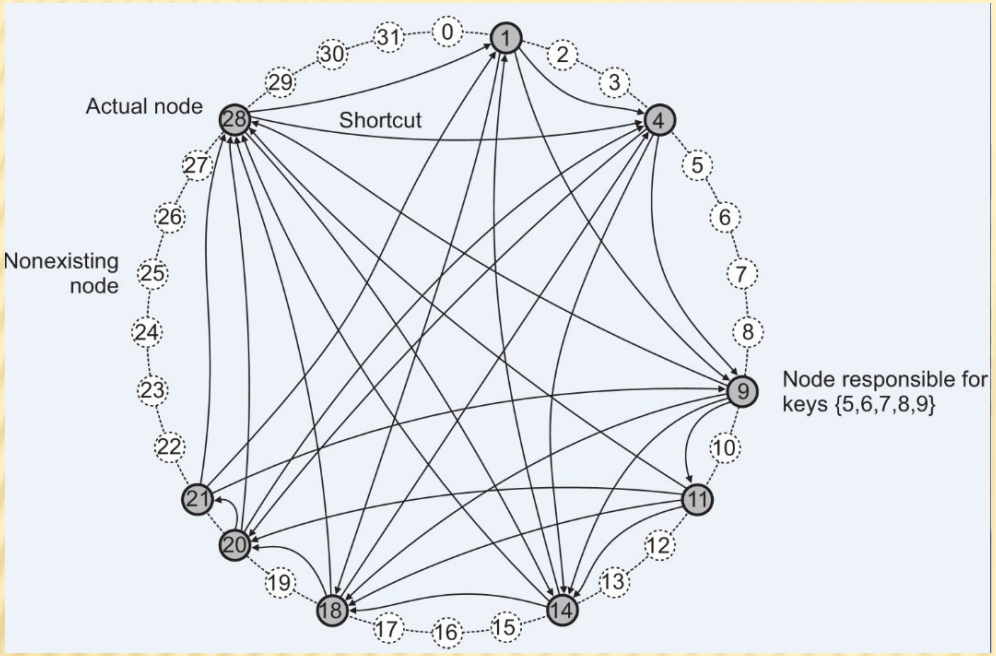
- Узлы образуют наложенную сеть логическое кольцо. В которой каждый экземпляр данных имеет ключ к состоящий из m-бит.
- ***Key(data item) = hash(data item's Value).***
- Ключ данных отображается на узел ***id= map(key),*** имеющий наименьший идентификатор

  id ≥ key

  Этот узел называется приемник (successor)

  $suc(key_i)=id$

ПОИСК УЗЛА В СИСТЕМЕ ХОРД

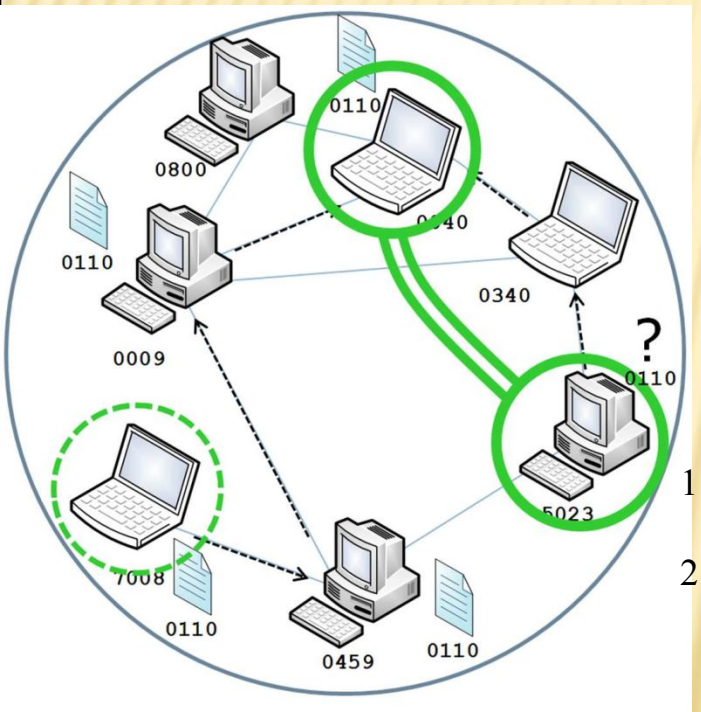


В показанной сети m=5 (2^5) ключи экземпляров данных key = {0-31} и девять узлов id = {1,4,9,11,14,18,20,21,28}.
Каждый узел связан “кратчайшими” дугами с другими узлами (как это делается
пока не важно).
Для поиска ключа узел посылает запрос к узлам с которыми он связан дугами.

- Предположим, узел 9 ищет узел отвечающий за экземпляр данных с ключем 3 (это будет узел 4).
- Узел 9 имеет дуги с узлами 11,14, 18 и 28. Так как самым дальним является узел 28, то к нему и обращается узел 9.
- Узел 28 имеет дуги с 1, 4 и 14, и он не знает о существовании узла между узлами 1 и 4. Поэтому он форвардирует запрос к узлу 1.
- Узел 1 знает, что он предшествует узлу 4, а значит узел 4 отвечает за ключ 3, поэтому он направляет запрос к узлу 4.
- Узел 4 ответит узлу 9 напрямую.

ПОИСК С ПОМОЩЬЮ МАРШРУТИЗАЦИИ ДОКУМЕНТОВ (1)

- Данный метод обеспечивает поиск документов без участия центрального индекса, средствами самой вычислительной сети.
- В основе данного метода лежит принцип присвоения уникальных идентификаторов каждому узлу вычислительной сети, а также каждому ресурсу (документу, сервису и др.), который данная сеть может предоставлять.



Когда какой-либо пир (на рисунке – узел 7008) производит публика- цию ресурса в сети, каким-либо образом вычисляется идентификатор данного ресурса (на рисунке – документ 0110). При этом, идентификаторы узлов сети и ресурсов имеют единую область возможных значений.

Далее, копия данного ресурса (или ссылка на него) **связывается** с узлуом, который имеет наиболее похожий идентификатор, на рисунке показана трасса документа 0110:

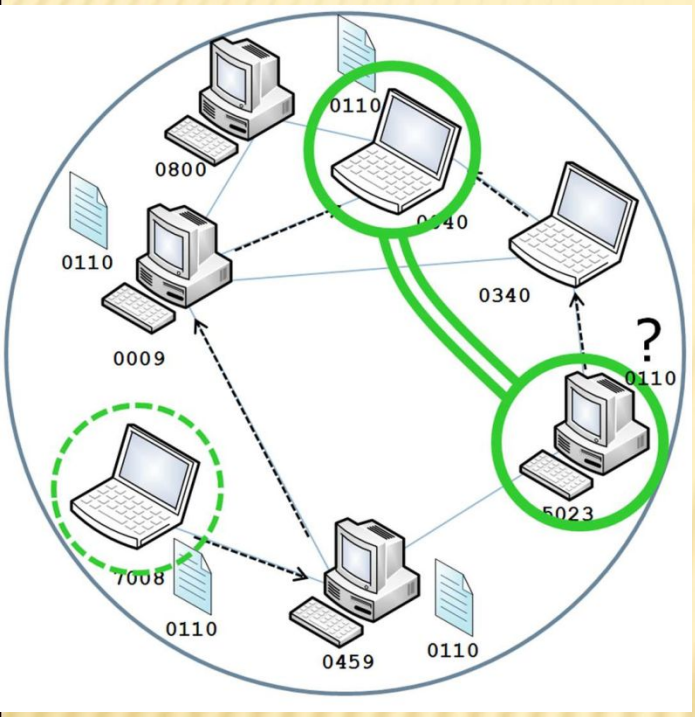
узлы 7008 – 0459 – 0009 – 0040.

Данная процедура производится следующим образом:

1. Производится сравнение идентификатора ресурса с идентификаторами всех соседних узлов.
2. Если идентификатор *текущего* узла ближе всего (по некоторой метрике) к идентификатору документа, то процесс трассировки завершается.

Если один из идентификаторов соседних узлов ближе к идентификатору документа, чем идентификатор текущего узла, то ссылка на данный ресурс *копируется* на узел с более близким идентификатором и процесс повторяется с п. 1. (узел 0040 на рисунке).

ПОИСК С ПОМОЩЬЮ МАРШРУТИЗАЦИИ ДОКУМЕНТОВ (2)



- Поиск ресурса производится по аналогичному алгоритму, но вместо копирования документа происходит трансляция запроса (в запросе содержится идентификатор запрашиваемого ресурса, например 0110 на рисунке 57) от узла, инициировавшего запрос (на рисунке – узел 5203) к узлу, идентификатор которого ближе всего соответствует идентификатору запрашиваемого документа.
- Соответственно, в процессе трансляции данного запроса должен появиться такой узел, который хранит информацию об интересуемом документе, если данный документ находится в соответствующей части сети.
- К недостаткам такого подхода можно отнести необходимость знания точного идентификатора документа, который необходимо найти в сети (невозможность поиска по отдельным атрибутам документа), а также возможность образования «островов», затрудняющих алгоритм поиска.

НЕСТРУКТУРИРОВАННЫЕ P2P СИСТЕМЫ

- Они не имеют определенной топологии.
- Каждый узел в такой системе поддерживает список соседних узлов, а об остальных он ничего не знает.
- В результате наложенная сеть такой системы представляет собой случайный граф: - граф в котором дуга между вершинами <u,v> существует с некоторой вероятностью P[<u,v>]. В идеале эта вероятность имеет одно и тоже значение, для всех пар вершин.
- При присоединении узла к P2P системе, он контактирует с общеизвестными в системе узлами и получает от них список других пиров от которых он может узнать о еще большем числе пиров.
- В таких системах, если пир хочет найти какой-либо ресурс, то он не может следовать какой-то определенной процедуре маршрутизации. Он должен прибегнуть к специальной процедуре поиска установленной в этой системе.
- Имеется два предельных метода поиска в таких системах:
  - Метод затопления (flooding).
  - Метод случайного блуждания (random walk).

МЕТОД ЗАТОПЛЕНИЯ

- Узел **u** посылает запросы ко всем известным ему соседям.
- Если узел **v** имеет требуемые данные, то он либо посылает их напрямую узлу породившему запрос, либо отсылает их к узлу форвардеру от которого он получил запрос.
- Если узел не имеет данных, то он форвардирует запрос ко всем своим соседям.
- Если узел уже получал такой запрос, то он не будет повторно на него отвечать.
- Для того, чтобы избежать избежать сильного затопления системы, для запросов устанавливается время жизни (TTL-Time to live). Каждый промежуточный узел обрабатывающий запрос уменьшает TTL на 1. Когда TTL станет равным 0 запрос удаляется из системы.
- От выбора значения TTL сильно зависит эффективность поиска и степень затопления системы запросами.

МЕТОД СЛУЧАЙНОГО БЛУЖДАНИЯ

- Узел **u** просто опрашивает соседей выбирая их случайным образом. Например, пусть u выберет для опроса узел **v.**
- Если **v** имеет требуемые данные, то он напрямую ответит u, если нет, то v форвардирует запрос своему соседу, выбранному случайным образом.
- Очевидно, что этот метод порождает значительно меньший трафик, но для поиска нужного ресурса может потребоваться значительно больше времени.
- Для уменьшения времени ожидания узел источник процедуры поиска может одновременно стартовать **n** процедур случайного блуждания. В этом случае в этом случае время поиска снижается в n раз.
- Случайное блуждание также требует своей принудительной остановки. Для этого можно также использовать TTL, либо можно установить условия при которых узел принявший запрос сам определит нужно форвардировать запрос случайным образом далее или нет.
- Для неструктурированных систем также необходимо определить порядок сравнения для определения признака нахождения искомых данных. Это могут быть те же методы, которые применяются в структурированных системах P2P.

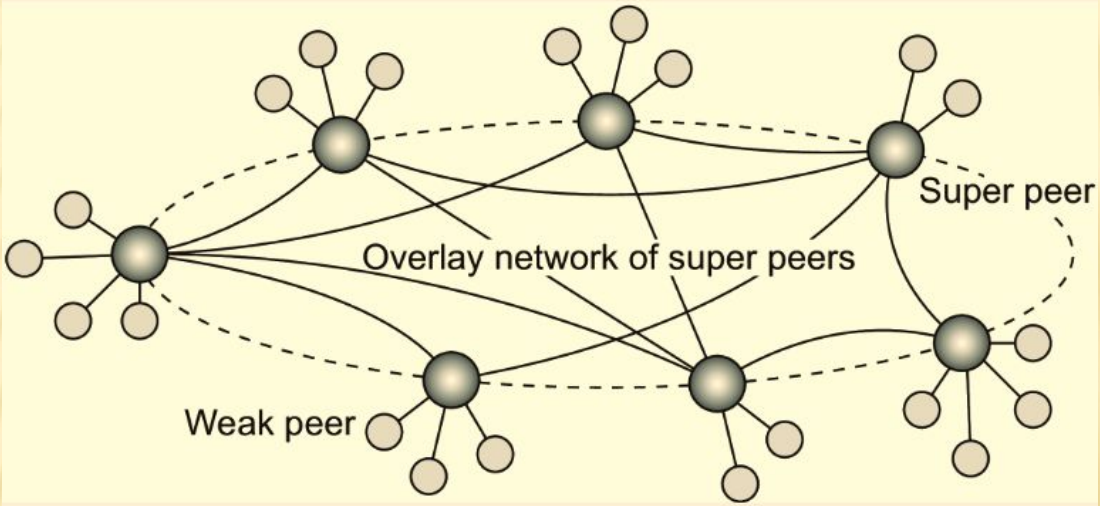
МЕТОД ПОИСКА ОСНОВАННЫЙ НА ПОЛИТИКЕ

- Этот метод занимает промежуточное положение между методами затопления и случайного блуждания.
- В этом методе для выбора узлов в качестве соседей используется некоторая установленная политика.
- Например:
  - в качестве узлов к которым перенаправляются запросы выбираются узлы, которые ранее более успешно выполняли поиск, по сравнению с другими узлами.
  - в качестве узлов к которым направляются запросу выбираются узлы, имеющие наибольшее число соседей.
  - и т.п.

ИЕРАРХИЧЕСКИ ОРГАНИЗОВАННЫЕ P2P СИСТЕМЫ

- В не структурированных системах по мере их роста поиск данных может стать невозможен. Источник этой проблемы масштабируемости очевиден – отсутствие детерминированой процедуры маршрутизации запросов поиска ресурсов.
- В качестве альтернативы во многих неструктурированных системах создаются специальные узлы хранящие индекс экземпляров данных имеющихся в системе.
- Имеются и другие ситуации в которых отказ от симметричной природы p2p систем имеет смысл. Например, сети доставки контента (CDN – Content Delivery Network).
- В CDN сетях узлы системы хранят копии Web документов отдельных сайтов Интернет к которым пользователям требуется быстрый доступ. Для поиска узлов используются **брокеры**, которые хранят сведения о степени используемости ресурсов и доступности узлов, что позволяет выбирать наиболее подходящий из узлов для получения ресурсов.

ИЕРАРХИЧЕСКАЯ ОРГАНИЗАЦИЯ СЕТИ P2P



- Узлы играющие роль брокеров, а также узлы поддерживающие индекс ресурсов называются **супер-узлами.** Супер-узлы часто образуют сеть p2p. Обычные узлы выполняющие роль клиентов супер-узлов называют слабыми (**weak nodes**).
- В этих сетях супер-узлы должны всегда оставаться доступными в сети. Это создает проблемы надежности, которые можно компенсировать, путем развертывания системы копирования восстановления между парами супер-узлов, и обеспечением подключения рядовых узлов к обеим супер-серверам.
- В сетях с супер-узлами, возникают и другие проблемы, например как

ИЕРАРХИЧЕСКАЯ P2P СИСТЕМА. ПРИМЕР: SKYPE

- Самой популярной на сегодняшний день службой Интернет-телефонии является Skype, созданная в 2003 году шведом Никласом Зеннстромом и датчанином Янусом Фриисом, авторами известной пиринговой сети KaZaA.
- В настоящее время Skype принадлежит корпорации Microsoft, которая приобрела ее за $8,5 млрд. в мае 2011 года.
- В состав системы входят следующие элементы:
  - *Skype-login сервер* – единственный централизованный элемент Skype-сети, обеспечивающий авторизацию Skype-клиентов.
  - *Обычный узел (Skype Client)* – обычный конечный узел в сети.
  - *Супер-узел (Super node)* – узлы, играющие роль роутеров в сети Skype.
  - *Выделенные узлы* для установки связи со стационарными телефонными линиями.
- Каждый узел Skype-сети хранит перечень IP-адресов и портов известным ему super-узлов в динамически обновляемых кэш-таблицах (Host Cache Tables, HC-tables). Начиная с версии Skype 1.0, кэш-таблицы представляют собой простой XML-файл, в незашифрованном виде записанный на диске в домашней директории пользователя.
- Любой узел, имеющий публичный IP-адрес (т.е. тот, который маршрутизируется в Интернет) и обладающий достаточно широким каналом, автоматически становится super-узлом и пропускает через себя трафик обычных узлов, помогая им преодолеть защиты типа брандмауэров или трансляторов сетевых адресов (NAT) и равномерно распределяя нагрузку между хостами.
- Единственным централизованным элементом является Skype-login сервер, отвечающий за процедуру авторизации Skype-клиентов и гарантирующий уникальность «позывных» для всей распределенной сети.

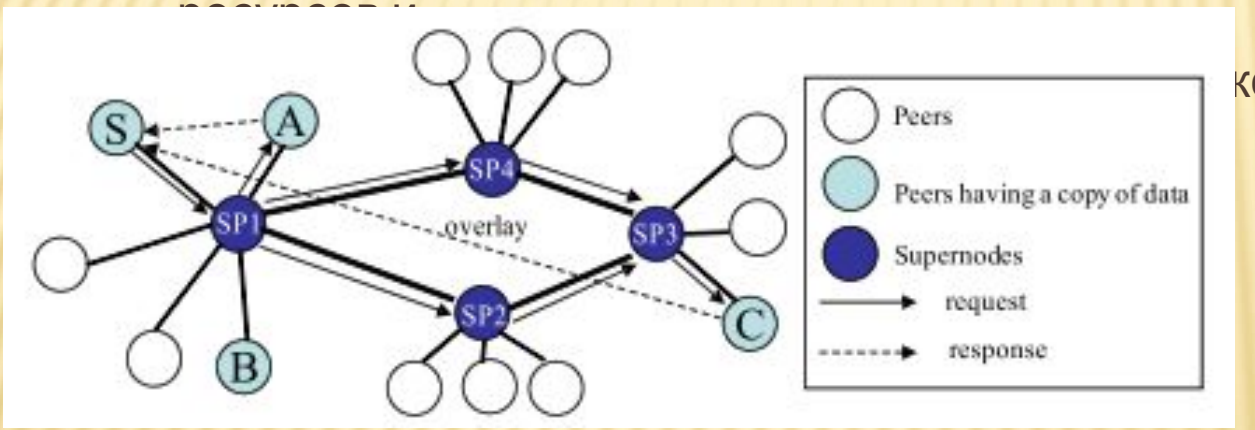
ГИБРИДНЫЕ АРХИТЕКТУРЫ P2P СИСТЕМ

ГИБРИДНЫЕ АРХИТЕКТУРЫ P2P

- Существует множество архитектур PC в которых успешно сочетаются архитектуры клиент сервер и децентрализованные архитектуры P2P систем.

ЧАСТИЧНО ЦЕНТРАЛИЗОВАННЫЕ (ГИБРИДНЫЕ) СИСТЕМЫ P2P

- В этих системах все узлы делятся на несколько типов в зависимости от иерархической организации. В двухуровневой системе имеется два вида узлов:
  - суперузлы (supernodes); мощные узлы связанные между собой хорошими каналами связи. Хранят метаданные о расположении

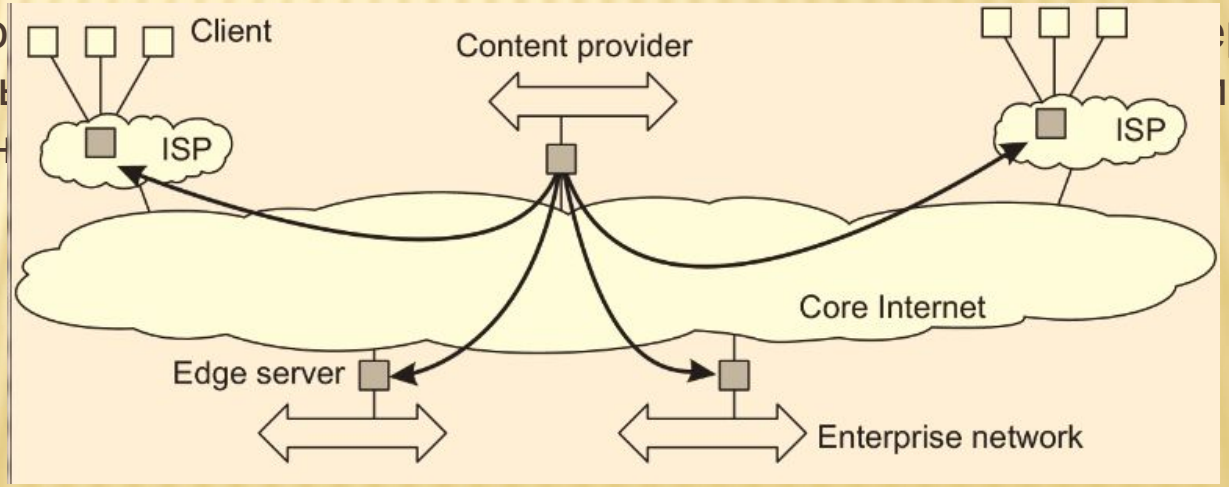


колькими суперузлами.

Пример Kazaa:
Узел S к ближайшему суперузлу запрос на поиск ресурса. Суперузел форвардирует это запрос ко всем подключенным узлам.
Узел A отвечает на запрос напрямую к S.
Так как запрос форвардируется и к другим суперузлам, то него отвечает и узел C.
S выбирает сам к какому из ответивших узлов обраться для получения ресурса.

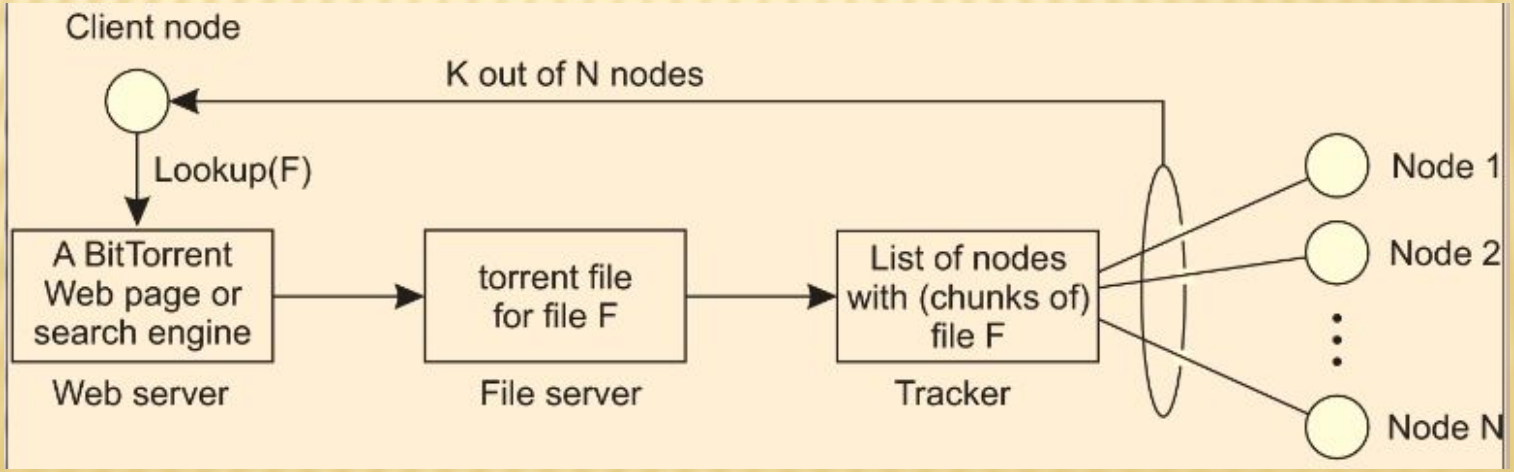
СИСТЕМЫ С ГРАНИЧНЫМИ СЕРВЕРАМИ

- Эти системы разворачиваются в Интернет, при этом на границах сетей сервис провайдеров (ISP – Internet Service Provider) размещаются сервера этих систем.
- Конечные пользователи этих систем располагаются в сетях сервис провайдеров и подключаются к распределенной системе через граничные сервера.
- Основное назначение граничных серверов предоставлять контент для пользователей, возможно после предварительной фильтрации и перекодировки.
- Базовая модель состоит в том, что для организации граничный сервер является источнико ер может использовать Web страниц. стран



СИСТЕМЫ НА ОСНОВЕ ВЗАИМНОГО СОТРУДНИЧЕСТВА (COLLABORATIVE SYSTEMS)

- Гибридная архитектура особенно широко используется в системах на основе взаимного сотрудничества. В качестве примера рассмотрим **BitTorent**.
- Основная идея состоит в том, что пользователь загружает файл не из одного места, а из множества источников по частям. Такими источниками являются машины других пользователей, которые тоже ищут в системе файлы для закачки, но должны предоставлять доступ к уже закаченным файлам. Данное условие обеспечивает поддержку сотрудничества между пользователями.
- Протокол BitTorrent был разработан в 2001 Брэмом Коэном.



BITTORRENT

- Если узел хочет опубликовать файл или набор файлов, то программа- клиент BitTorrent сети разделяет передаваемые файлы на части и создает *файл метаданных* (идентификатор раздачи), который содержит следующую информацию:
  - URL трекера (супер-сервера, который хранит актуальную информацию об активных узлах которые имеют отдельные части файла);
  - Общая информация о файлах (имя, длина и пр.);
  - Хеш-суммы SHA1 сегментов раздаваемых файлов;
  - Passkey пользователя – ключ, который однозначно определяет пользователя загрузившего файл;
  - Хеш-суммы файлов целиком (не обязательно);
  - Альтернативные источники – адреса альтернативных трекеров, на которых можно найти информацию по данному файлу (не обязательно).
- Алгоритм загрузки документа производится следующим образом:
  - клиент подключается к трекеру по URL из файла метаданных;
  - сообщает хеш-идентификатор требуемого файла;
  - получает адреса пиров скачивающих и раздающих данный файл;
  - клиенты соединяются между собой и обмениваются информацией без участия трекера.
- В последнее время стала распространяться альтернативная технология поиска и загрузки документов на основе «магнитных ссылок» (magnet links) и подхода распределенных хеш-таблиц (Distributed Hash Table — DHT) по сути дела представляющих собой реализацию алгоритма маршрутизации документов.
- Причина возникновения этой технологии – дальнейшее развитие деперсонализации и попытка торрент-трекеров защититься от юридического преследования правообладателей. Торрент-файл для такой раздачи создаётся без адреса трекера и клиенты находят друг друга через распределенные хеш-таблицы.

ПРИМЕНЕНИЕ СИСТЕМ P2P

- Наибольшее распространение пиринговые системы получили при реализации сиситем предназначенных для обработки больших объемов данных и обеспечивающих индивидуальный обмен информацией между пользователями.
- В настоящий момент, технологии P2P наиболее ярко представлены в 3-х направлениях:
  - *Распределенные вычисления*: разбиение общей задачи на большое число независимых в обработке подзадач (проекты на платформе BOINC);
    Файлообменные сети: P2P выступают альтернативой FTP-архивам, которые утрачивают перспективу ввиду значительных информационных перегрузок (однако требуются эффективные механизмы поиска) (Gnutella, eDonkey, BitTorrent);
  - *Приложения для совместной работы*: требуют обеспечения прозрачных механизмов для совместной работы. (Skype, Groove).

ДОСТОИНСТВА P2P СИСТЕМ

- Можно выделить следующие основные преимущества пиринговых систем:
  - высокая масштабируемость, связанная с равномерным распределением вычислительной нагрузки на всех участников сети;
  - стабильность работы сети, обусловленная отсутствием таких «узких мест» как выделенный сервер, обрабатывающий все сетевые запросы;
  - возможность объединения ресурсов отдельных участников сети, и их предоставление другим участникам;
  - распределение совокупных затрат на предоставление ресурсов между участниками сети.

НЕДОСТАТКИ СИСТЕМ P2P

- Стоит упомянуть о следующих недостатках и особенностях функционирования p2p систем:
  - в одноранговых сетях не может быть обеспечено гарантированное качество обслуживания: - любой узел, предоставляющий те или иные сервисы, может быть отключен от сети в любой момент;
  - индивидуальные технические характеристики узла могут не позволить полностью использовать ресурсы P2P сети (каждый из узлов обладает индивидуальными техническими характеристиками что, возможно, будет ограничивать его роль в P2P-сети и не позволят полностью использовать ее ресурсы: - низкий рейтинг в torrent-сетях, или LowID в eDonkey могут значительно ограничить ресурсы сети, доступные пользователю);
  - при работе того или иного узла через брандмауэр может быть значительно снижена пропускная способность передачи данных в связи с необходимостью использования специальных механизмов обхода;
  - участниками одноранговых сетей в основном являются индивидуальные пользователи, а не организации, в связи с чем возникают вопросы безопасности предоставления ресурсов: владельцы узлов P2P-сети, скорее всего, не знакомы друг с другом лично, предоставление ресурсов происходит без предварительной договоренности;
  - при увеличении числа участников P2P сети может возникнуть ситуация значительного возрастания нагрузки на сеть (как с централизованной, так и с децентрализованной структурой);
  - в случае применения сети типа P2P приходится направлять значительные усилия на поддержку стабильного уровня ее производительности, резервное копирование данных, антивирусную защиту, защиту от информационного шума и других злонамеренных действий пользователей.