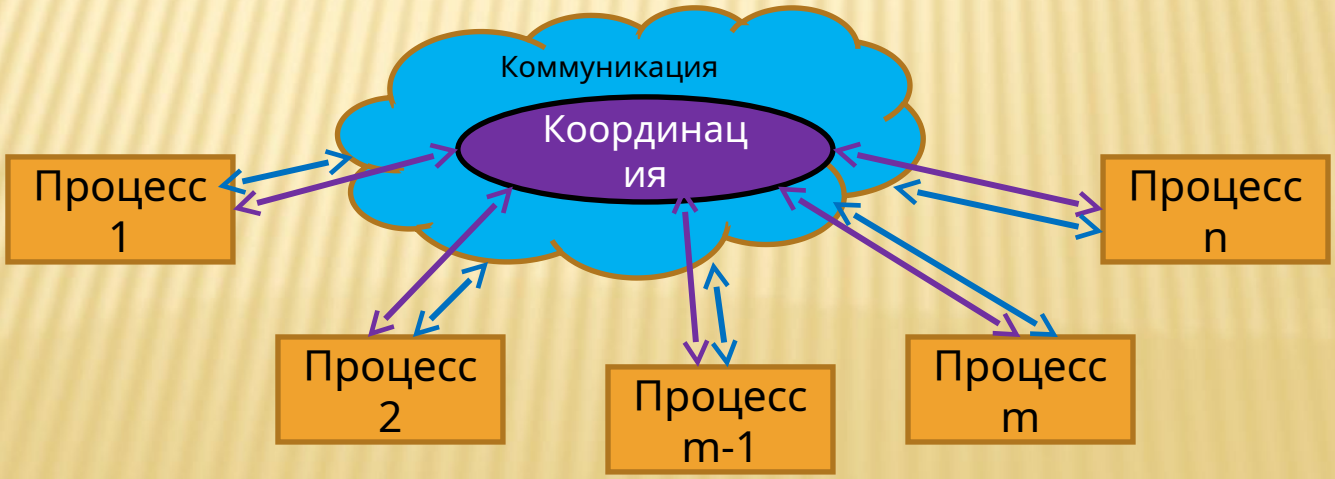
СИСТЕМЫ ПУБЛИКАЦИИ-ПОДПИСКИ (PUBLISH-SUBSCRIBE)

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.

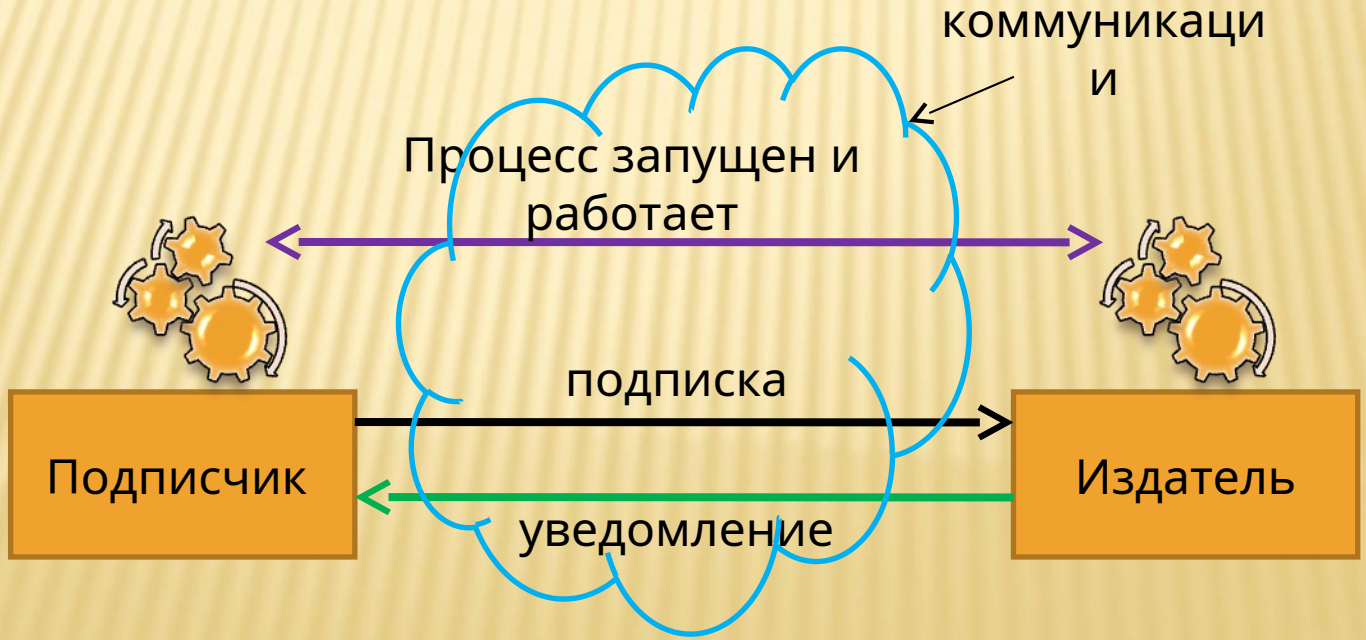
ВАРИАНТЫ АРХИТЕКТУР ПУБЛИКАЦИЯ/ПОДПИСКА

- 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

	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.

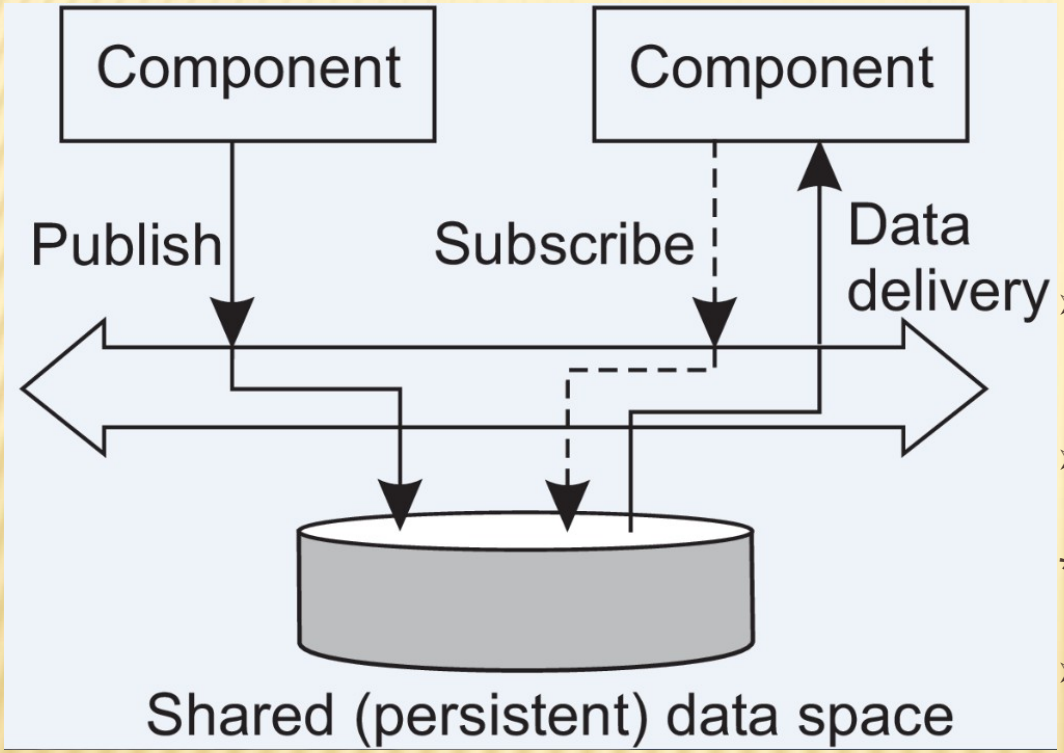


Архитектура П/П с координацией через почтовый ящик

- 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.



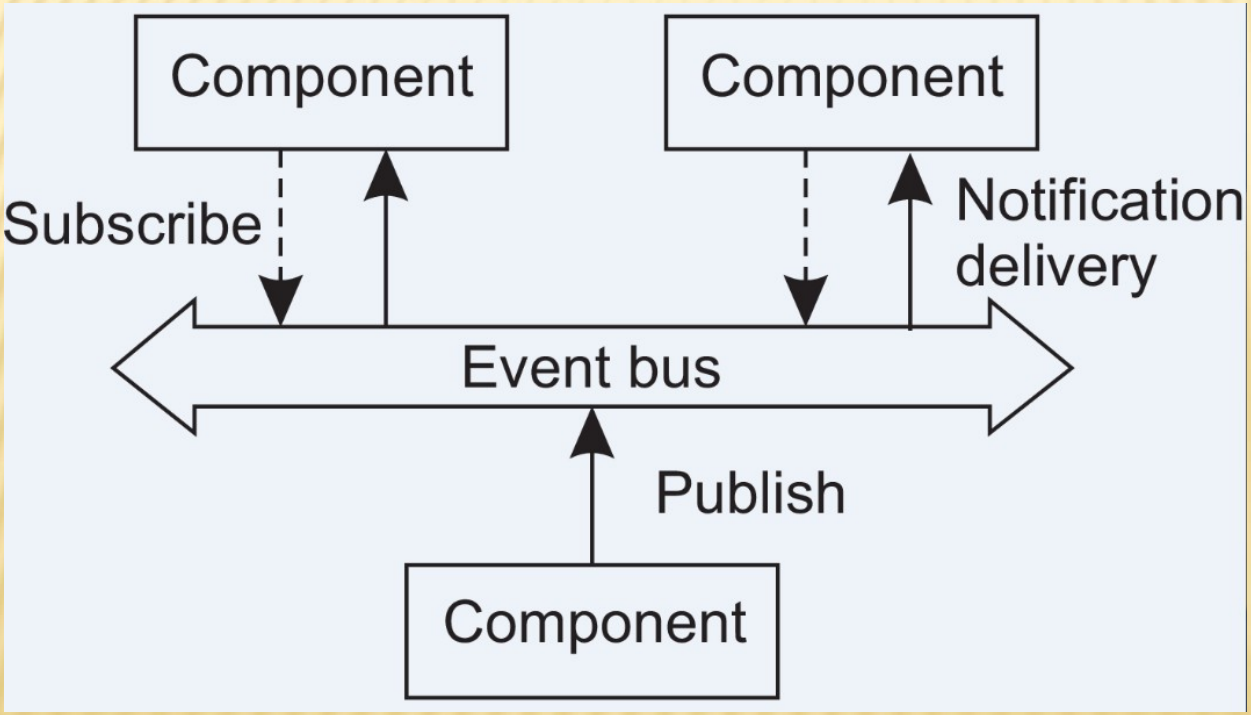
- Пример: Linda - программная модель, разработанная в 1980 году. Разделяемое пространство данных в модели Linda называется пространство записей (кортежей). Это пространство поддерживает три операции:
- `in(t):` извлечь (с удалением) запись, соответствующую шаблону t;
- `out(t):` занести запись по шаблону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



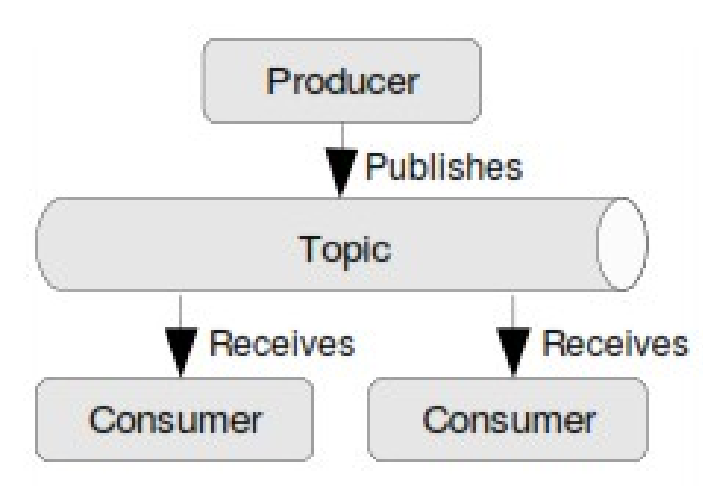
- Для получения уведомления процесс подписчик должен всегда находиться в рабочем состоянии.
- Отсутствие ссылочной информации не позволяет процессам иметь явные знания друг о друге.
- Такая архитектура соответствует коммуникациям через общую шину сообщений.

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 называется системой основанной на теме.

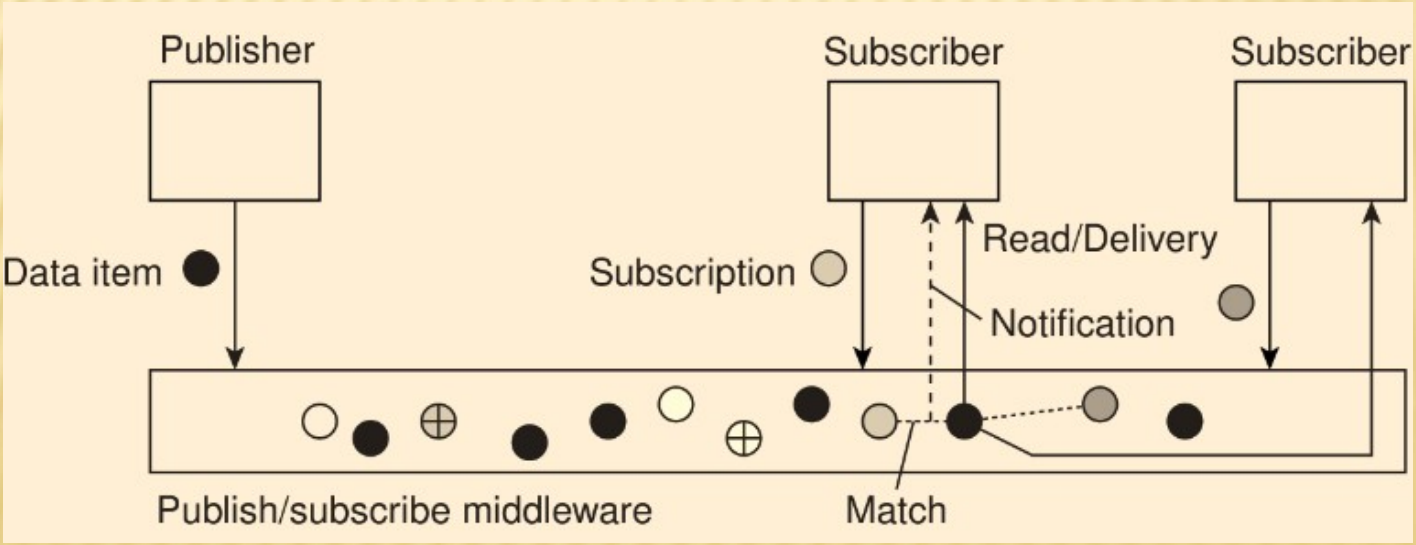


СИСТЕМЫ ПУБЛИКАЦИИ/ПОДПИСКИ ОСНОВАННЫЕ НА СОДЕРЖИМОМ (КОНТЕНТЕ)

- 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 (ПУ) 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

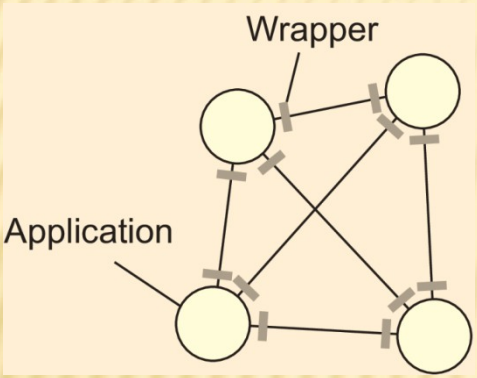
- События могут легко запутать работу подписчиков. В качестве примера рассмотрим такую подписку: "Уведомить, когда номер 1060 будет освобожден и дверь в номер будет не заперта".
- Обычно распределенная система, поддерживающая такие подписки, может быть реализована путем установки независимых сенсоров (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 помещаемые 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.

СПОСОБЫ РЕАЛИЗАЦИИ ПО ПРОМЕЖУТОЧНОГО СЛОЯ

Организация промежуточного ПО (Midleware)

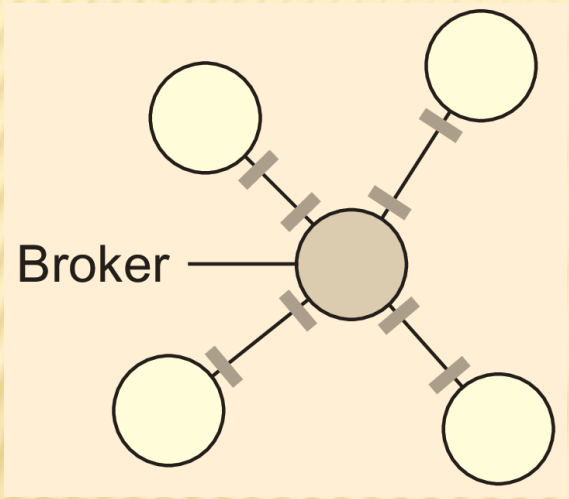
- При организации ПО промежуточного уровня в системах основанных на объектных компонентах, часто используются два шаблона проектирования:
  - Оболочки (Wrappers).
  - Перехватчики (Interceptors).
- Их применение направлено на достижение открытости системы.

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.

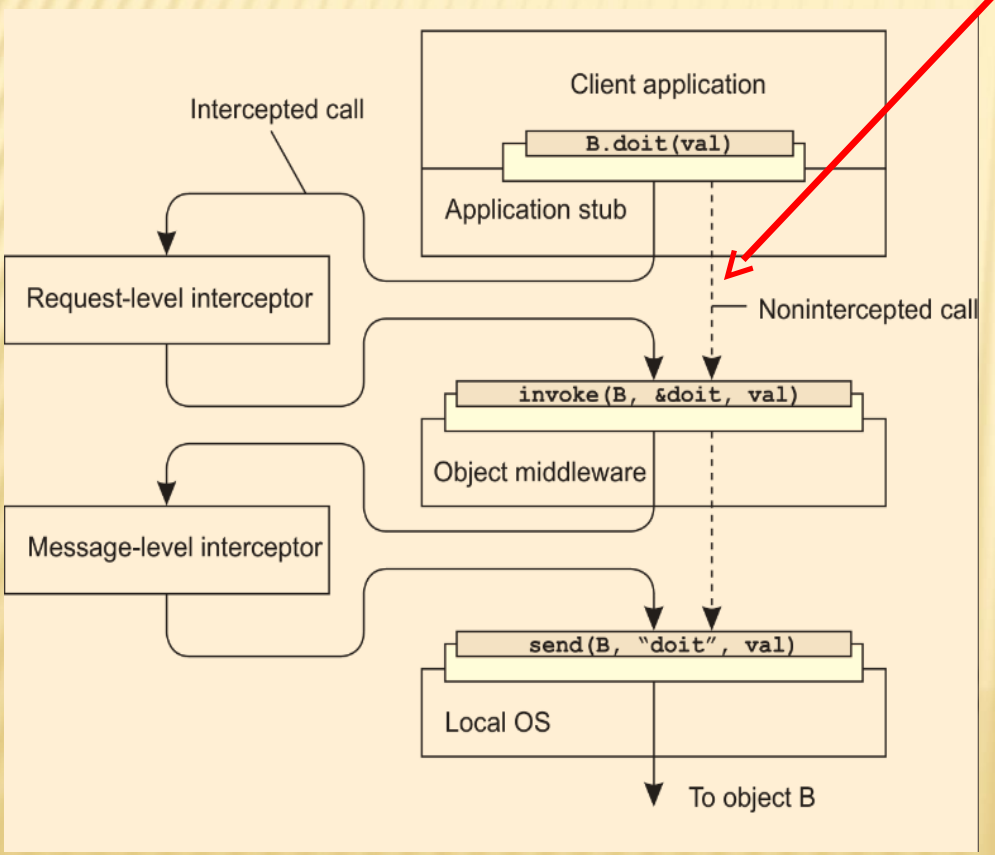
БРОКЕР СООБЩЕНИЙ



- 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 $O(N^2)$.

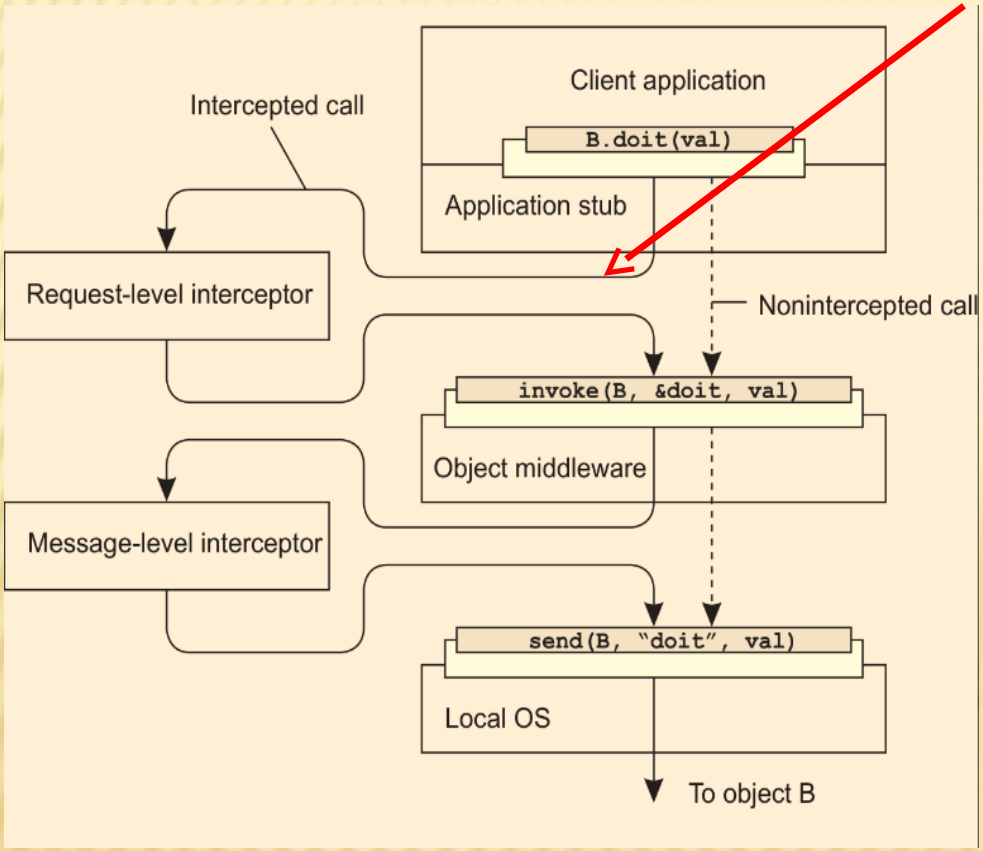
ПЕРЕХВАТЧИКИ ОБРАЩЕНИЙ (INTERCEPTORS)

- 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

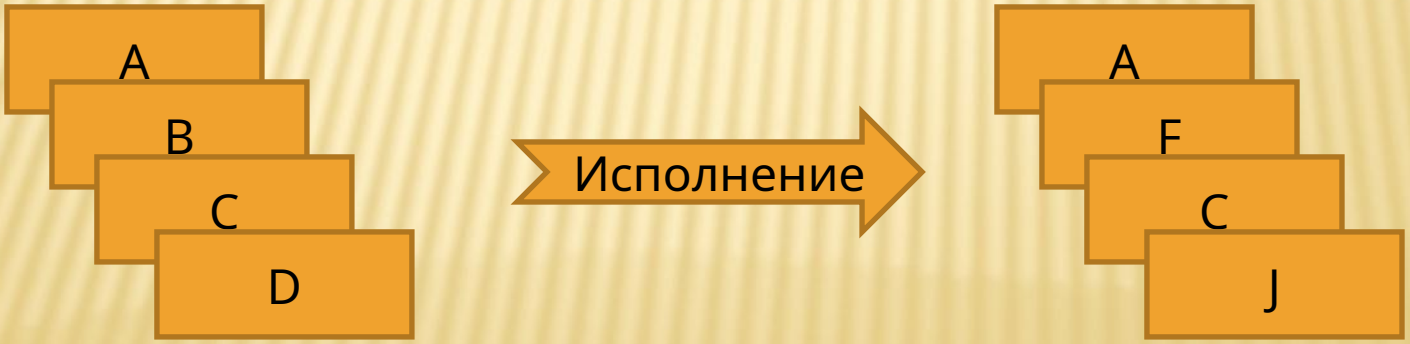
ОБРАЩЕНИЕ К РЕПЛИКАМ ОБЪЕКТА B



- 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.

ЗАМЕНА КОМПОНЕНТ ВО ВРЕМЯ ИСПОЛНЕНИЯ

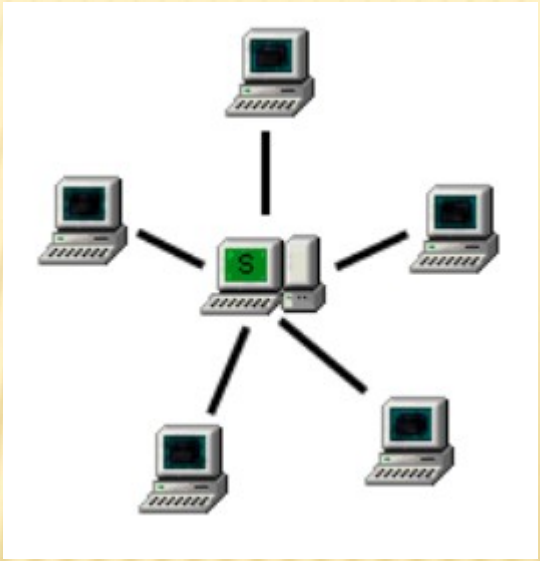
- Примером такого подхода является **замена программных компонент** в момент их исполнения. Этот способ является одним из наиболее широко распространенных среди других существующих подходов используемых для создания адаптируемого ПО промежуточного слоя.
- Проектирование на основе объектных компонент 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

- Роль, выполняемая сущностью, является фундаментом для определения архитектуры системы.
- Имеется два вида архитектурных систем, определенных на основе ролей, выполняемых элементами PC:
  - Централизованная архитектура (клиент-сервер);
  - Дентрализованная архитектура (peer-to-peer).

ЦЕНТРАЛИЗОВАННАЯ ОРГАНИЗАЦИЯ РС



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.

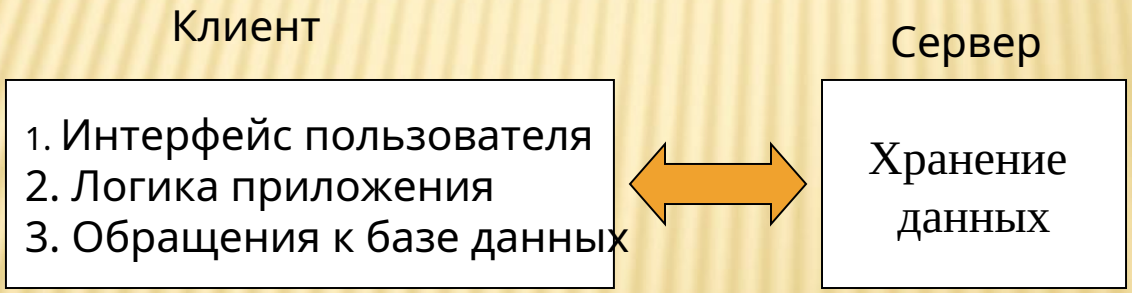
РАЗНОВИДНОСТИ МОДЕЛИ КЛИЕНТ-СЕРВЕР

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.

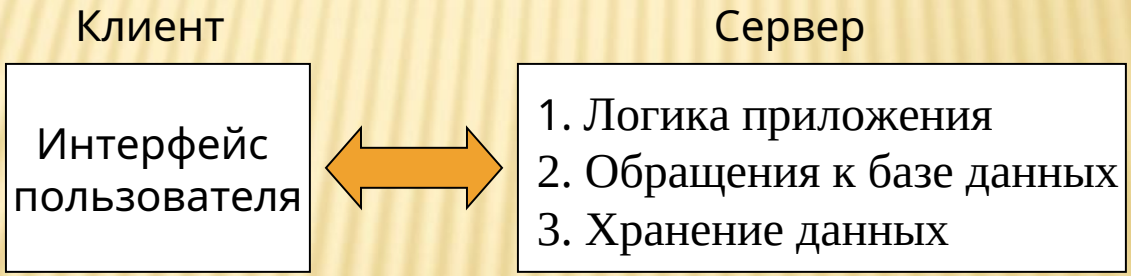
ТОЛСТЫЙ КЛИЕНТ

- 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.



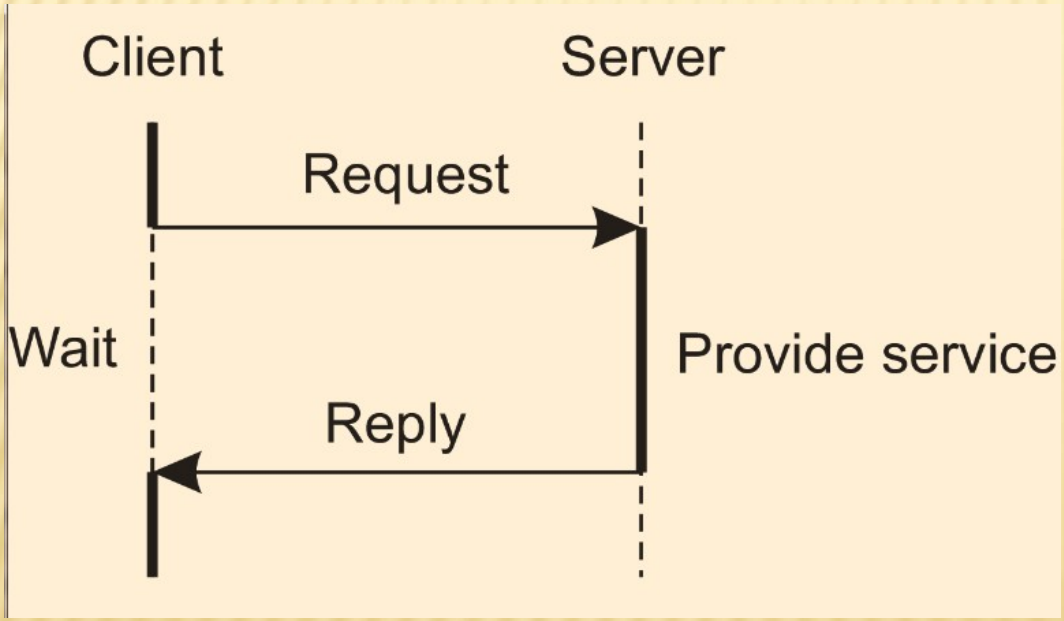
ТОНКИЙ КЛИЕНТ

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



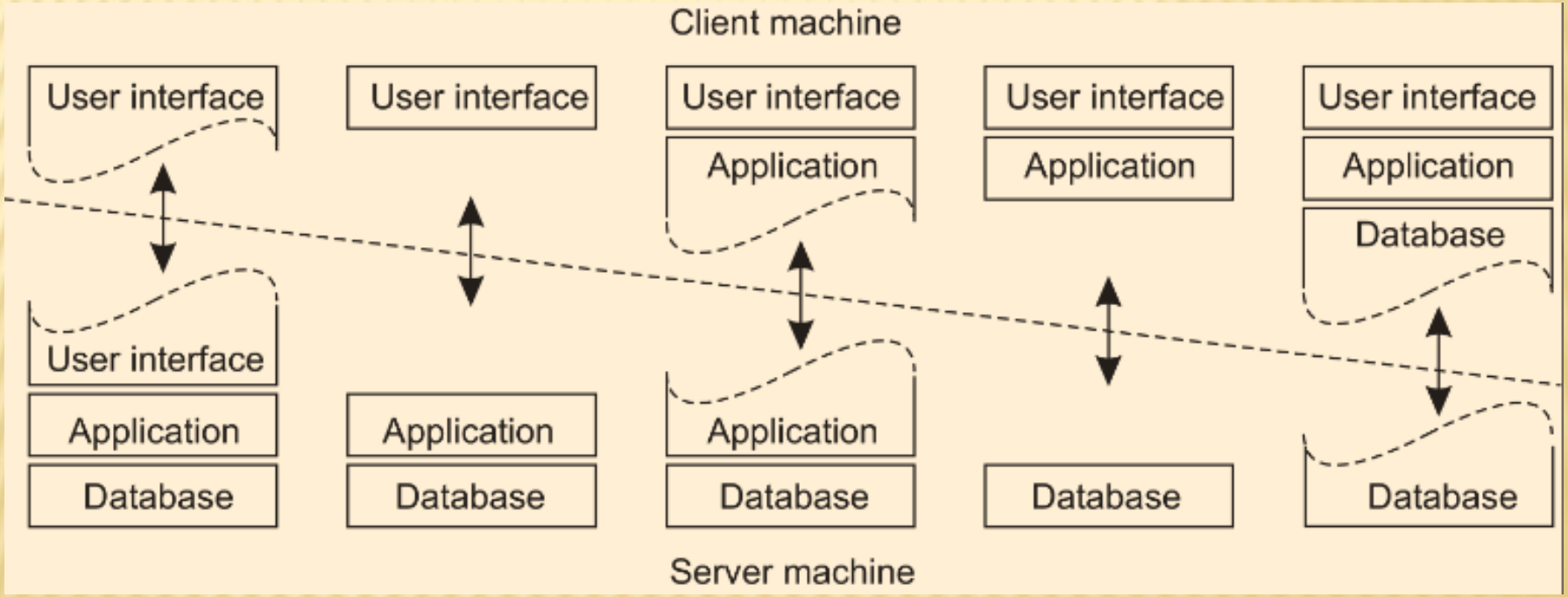
ВЗАИМОДЕЙСТВИЕ МЕЖДУ КЛИЕНТОМ И СЕРВЕРОМ

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



ФИЗИЧЕСКИ ДВУХЗВЕННЫЕ АРХИТЕКТУРЫ

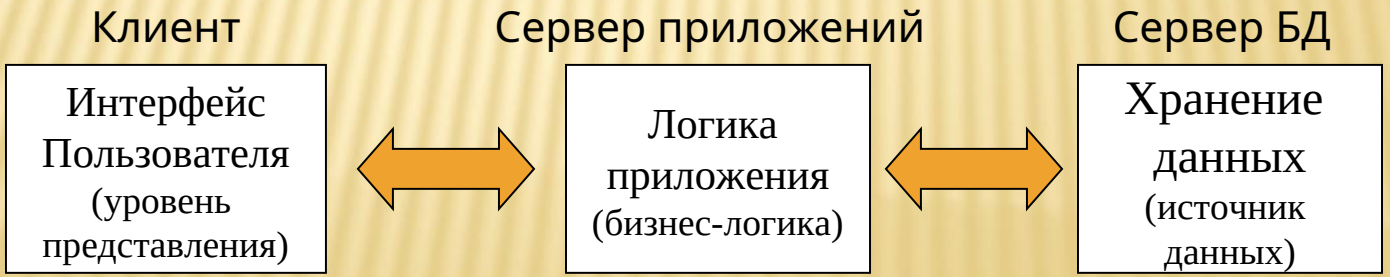
- Один из подходов к организации клиентов и серверов — это распределение программ, находящихся на уровне приложений, по различным машинам.



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

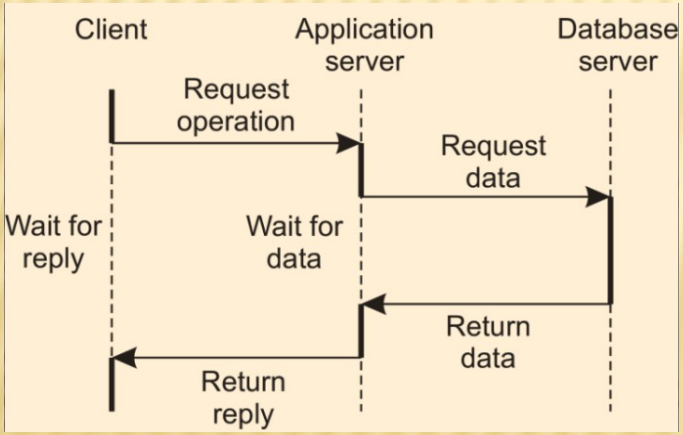
ФИЗИЧЕСКИ МНОГОЗВЕННЫЕ СИСТЕМЫ КЛИЕНТ-СЕРВЕР

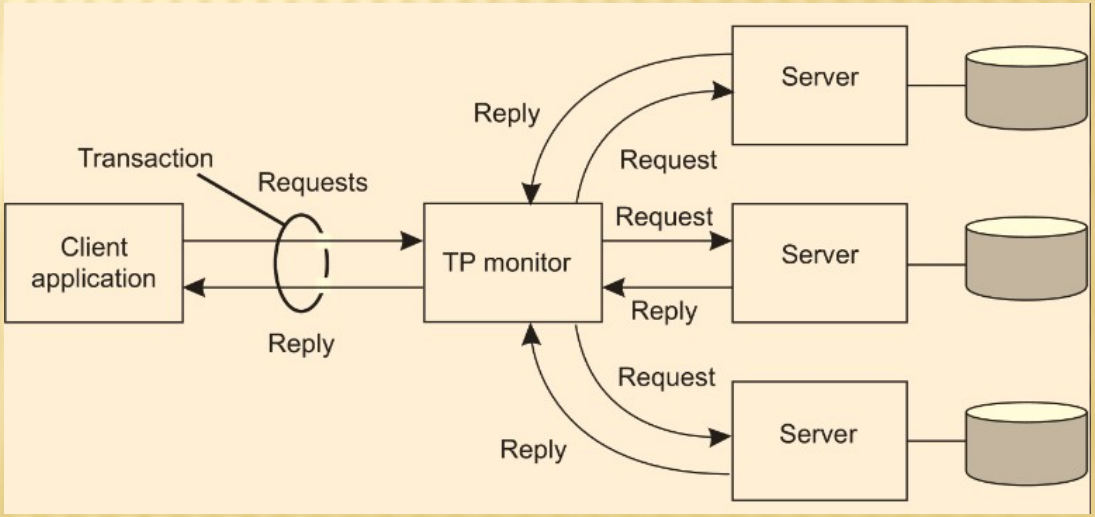
- **Многозвенная архитектура клиент-сервер** позволяет более разумно распределить модули обработки данных, которые в этом случае выполняются на одном или несколько серверах. Эти программные модули выполняют функции сервера для интерфейса с пользователем и клиента - для серверов баз данных. Многозвенная 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.



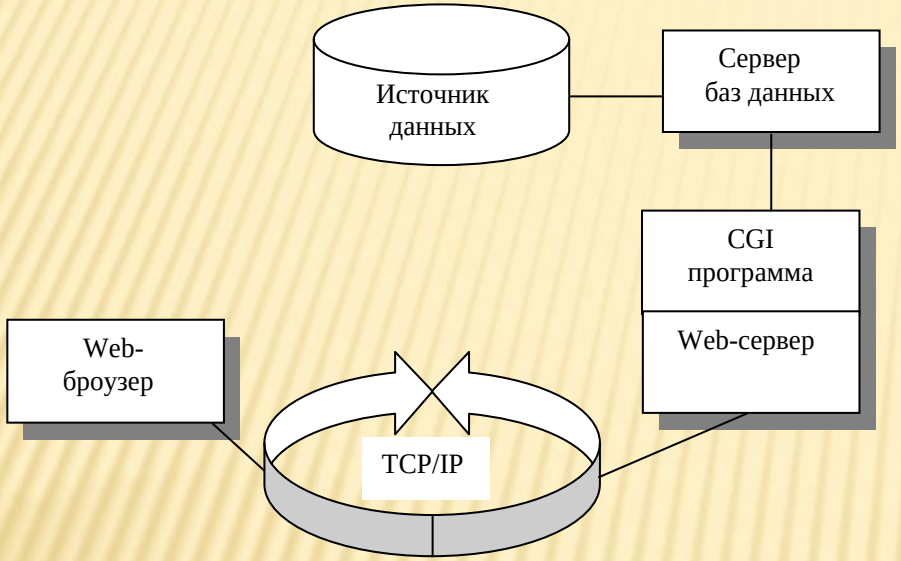
ПОВЕДЕНИЕ СЕРВЕРА КАК КЛИЕНТА В СЛОЖНЫХ ПРИЛОЖЕНИЯХ

- 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.
  - системы с сервером приложений.



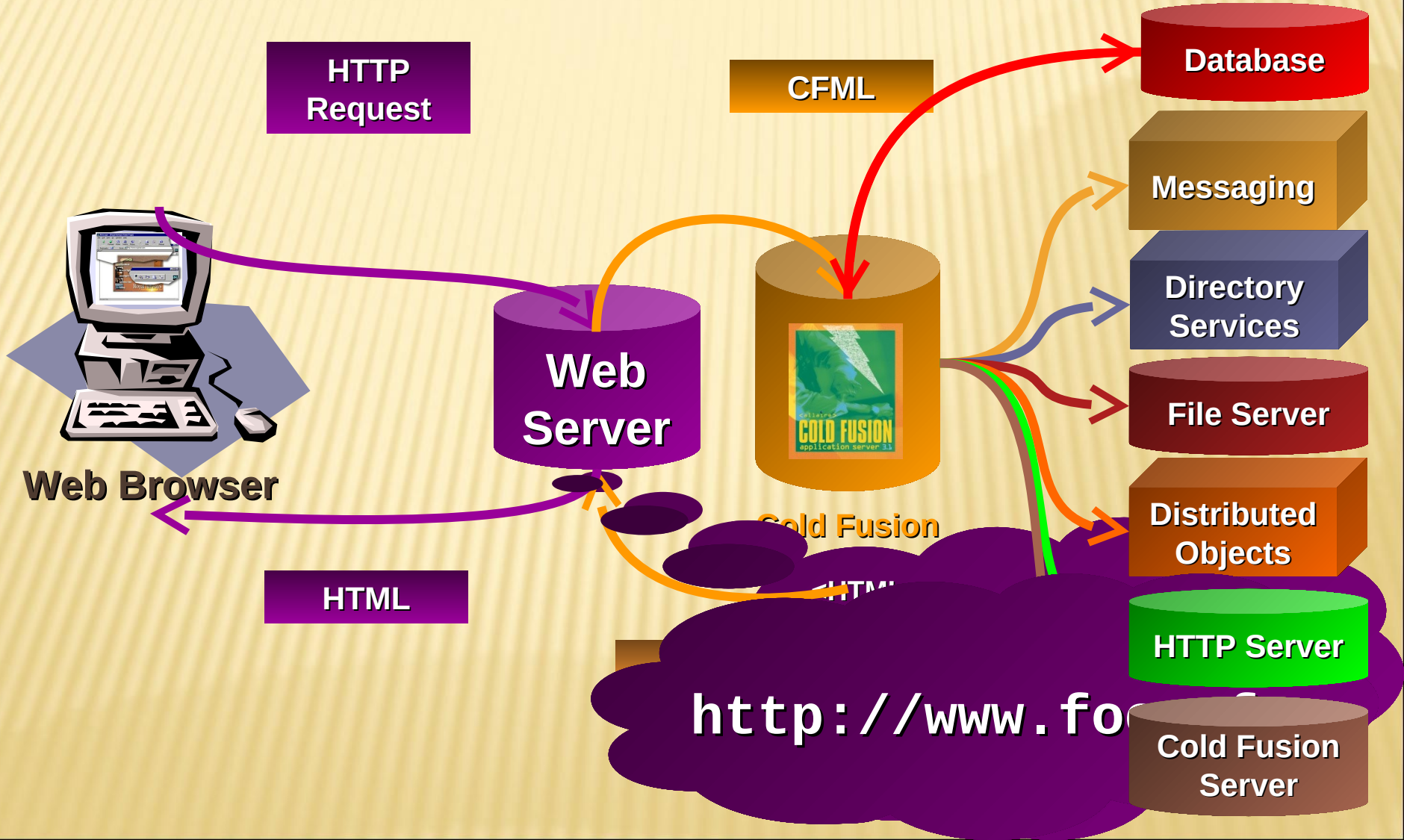


ПРИМЕР РАБОТЫ СЕРВЕРА В КАЧЕСТВЕ КЛИЕНТА В WEB ПРИЛОЖЕНИИ



- 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



СПАСИБО ЗА ВНИМАНИЕ !