Производительность и масштабируемость

Производительность системы (Performance)

- Способность управлять приложением таким образом, чтобы обеспечивать на приемлемом для пользователей системы уровне доступность, функциональность и прочие характеристики

Измерение производительности

- Время отклика
- Пропускная способность
- Утилизация ресурсов
- Масштабируемость
- Не в ущерб **надежности**
  - > В особенности для КИС

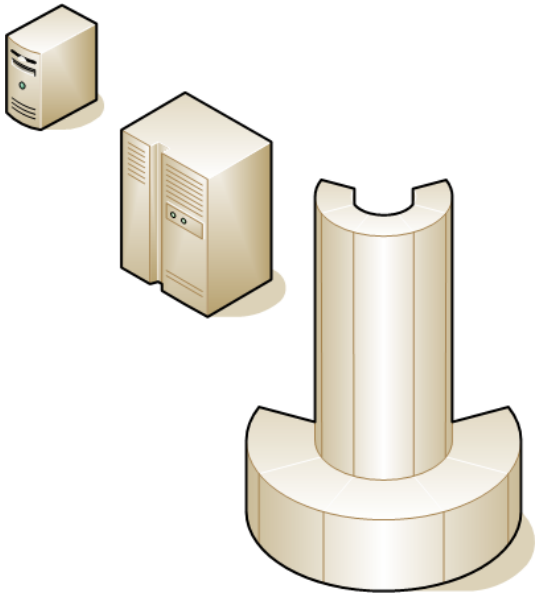
Масштабируемость

- Способность системы увеличивать свою производительность при добавлении ресурсов

$$\frac{dPerf}{dResource} \rightarrow 1$$

Направления масштабируемости

- Вертикальная масштабируемость - Увеличение производительности каждого компонента системы

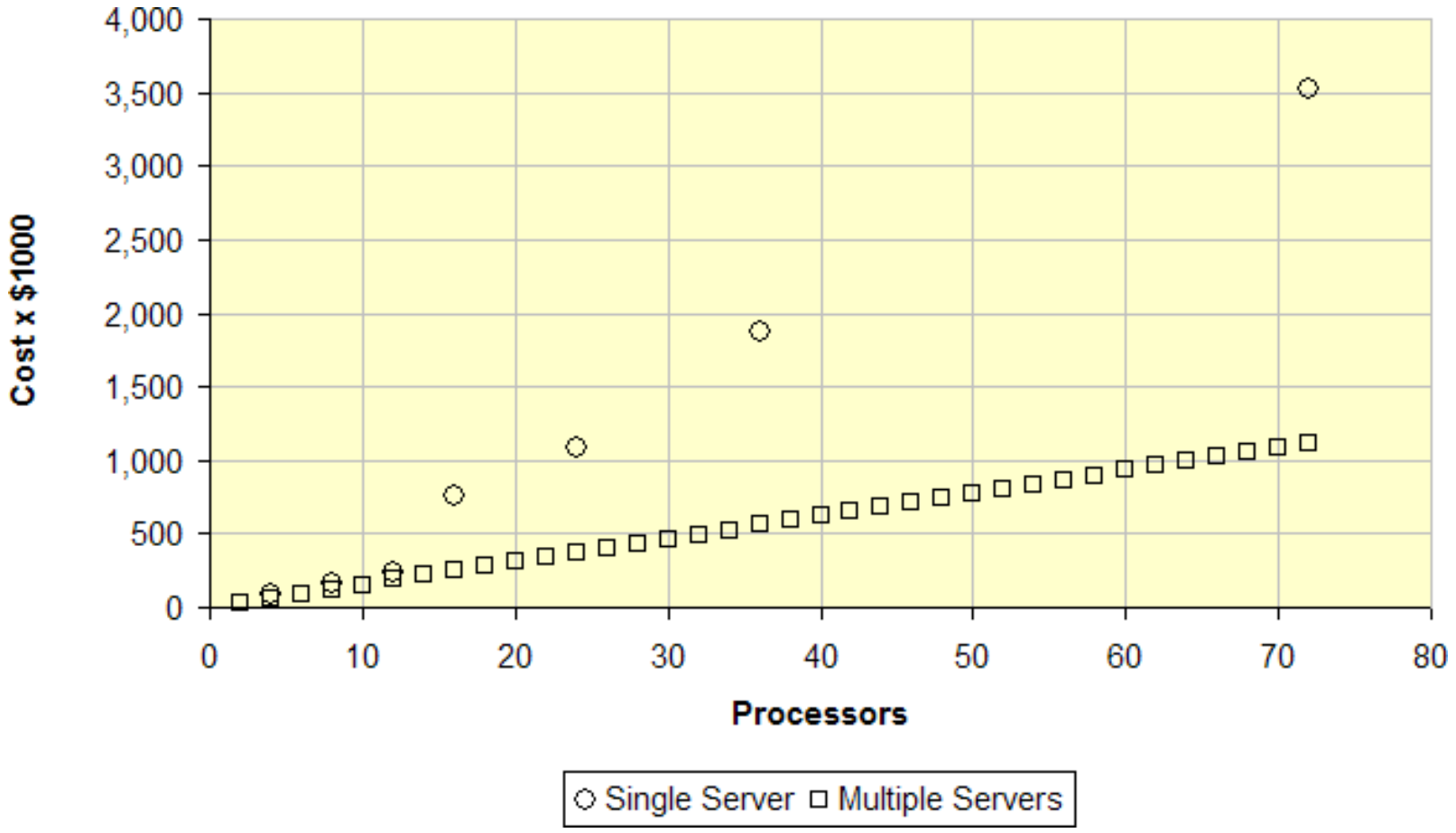


Scale up

- Горизонтальная масштабируемость - Разбиение системы на более мелкие структурные компоненты и разнесение их по отдельным физическим машинам и/или увеличение количества серверов параллельно выполняющих одну и ту же функцию

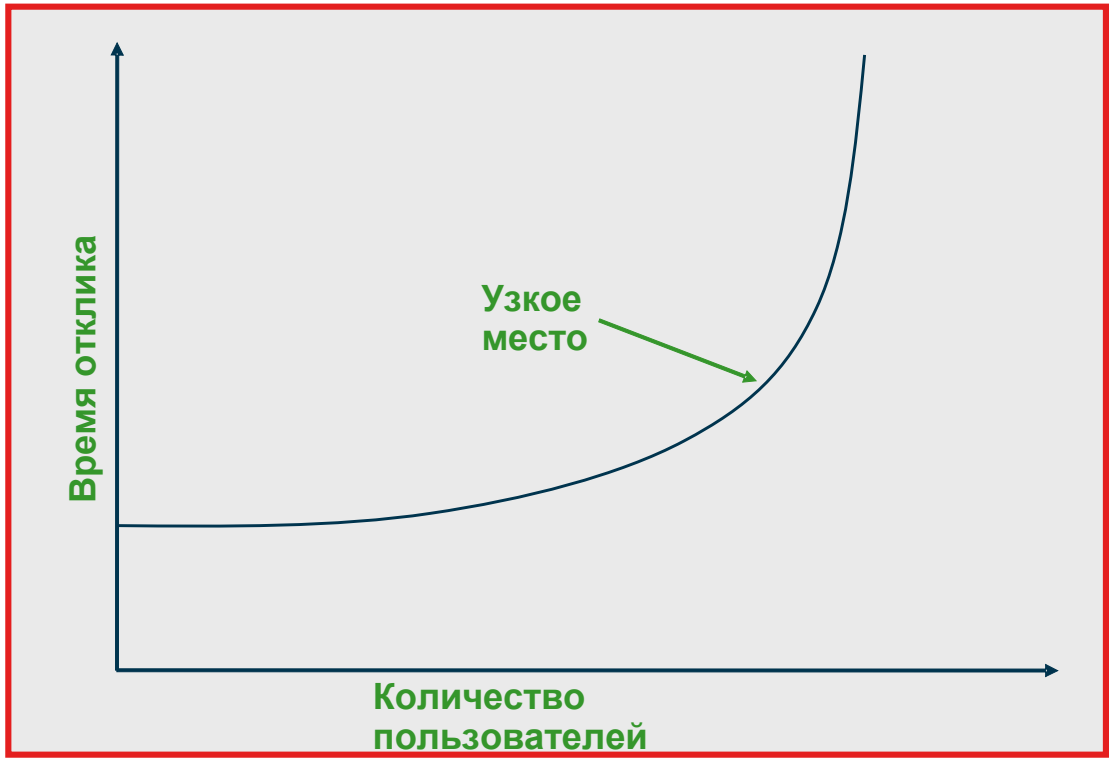
Scale out

Стоимость вертикального и горизонтального масштабирования



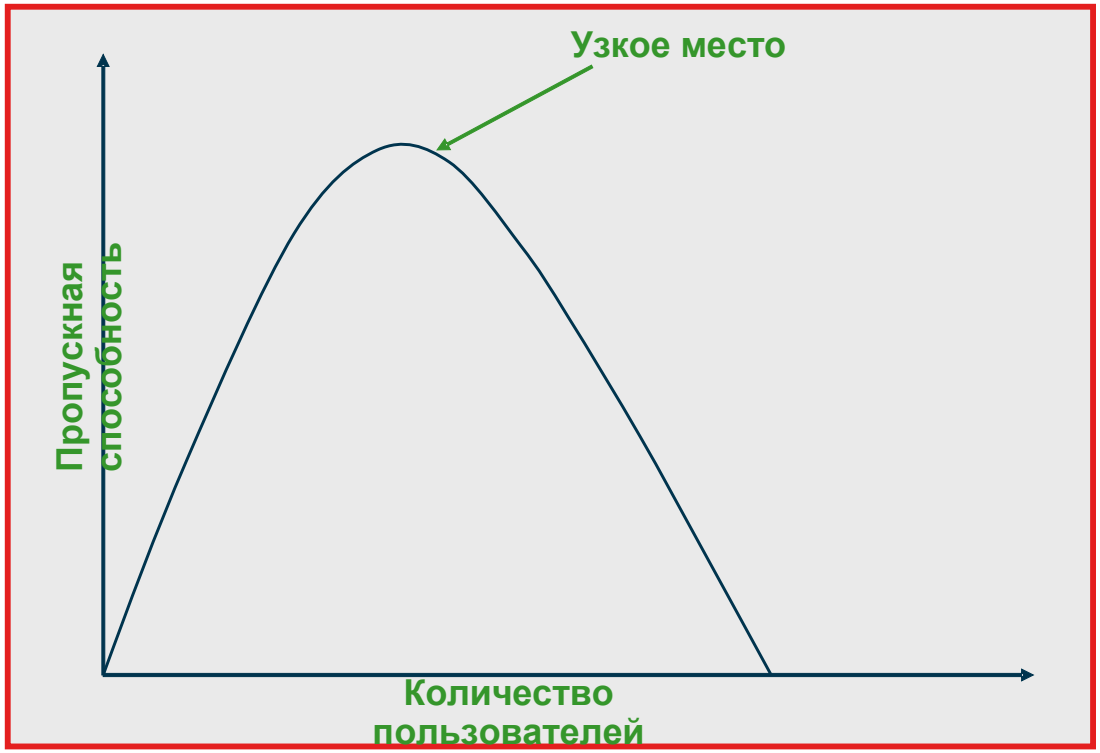
Узкие места: Зависимость времени отклика от пользовательской нагрузки

- Резкий рост времени отклика является результатом низкой эффективности



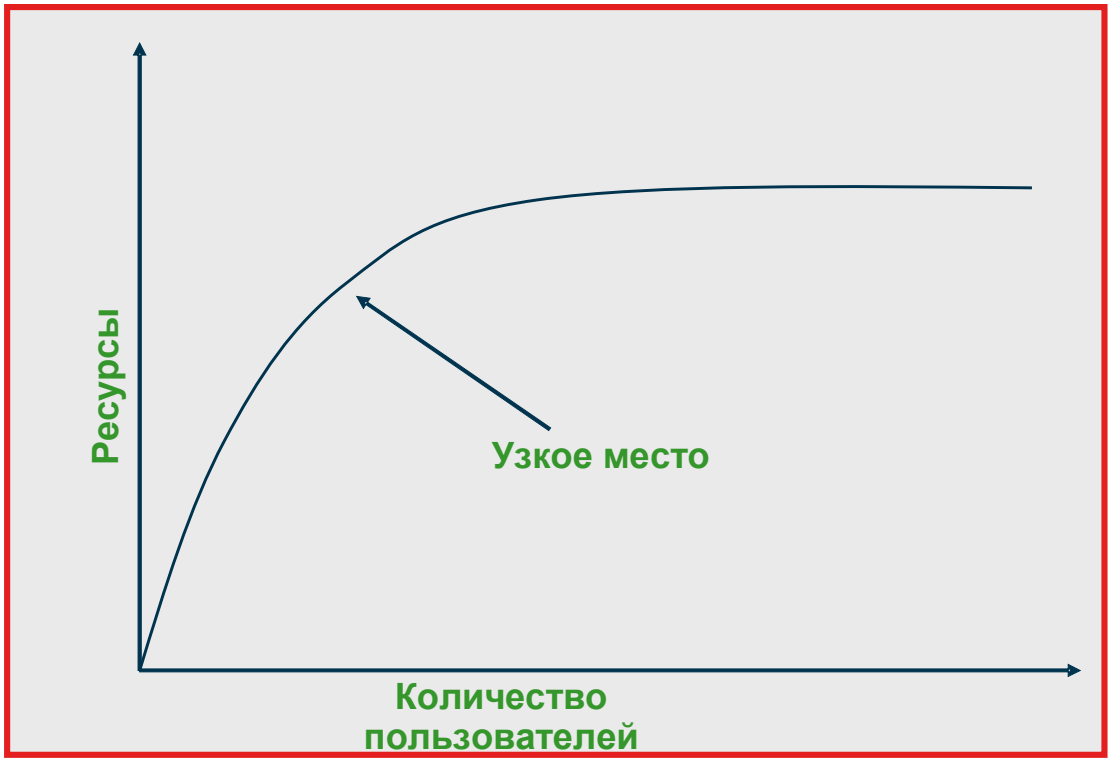
Узкие места: Зависимость пропускной способности от пользовательской нагрузки

- Точка спада пропускной способности обозначает узкое место



Узкие места: Зависимость утилизации ресурсов от пользовательской нагрузки

- Постоянный уровень утилизации при увеличении количества пользователей



Настройка производительности

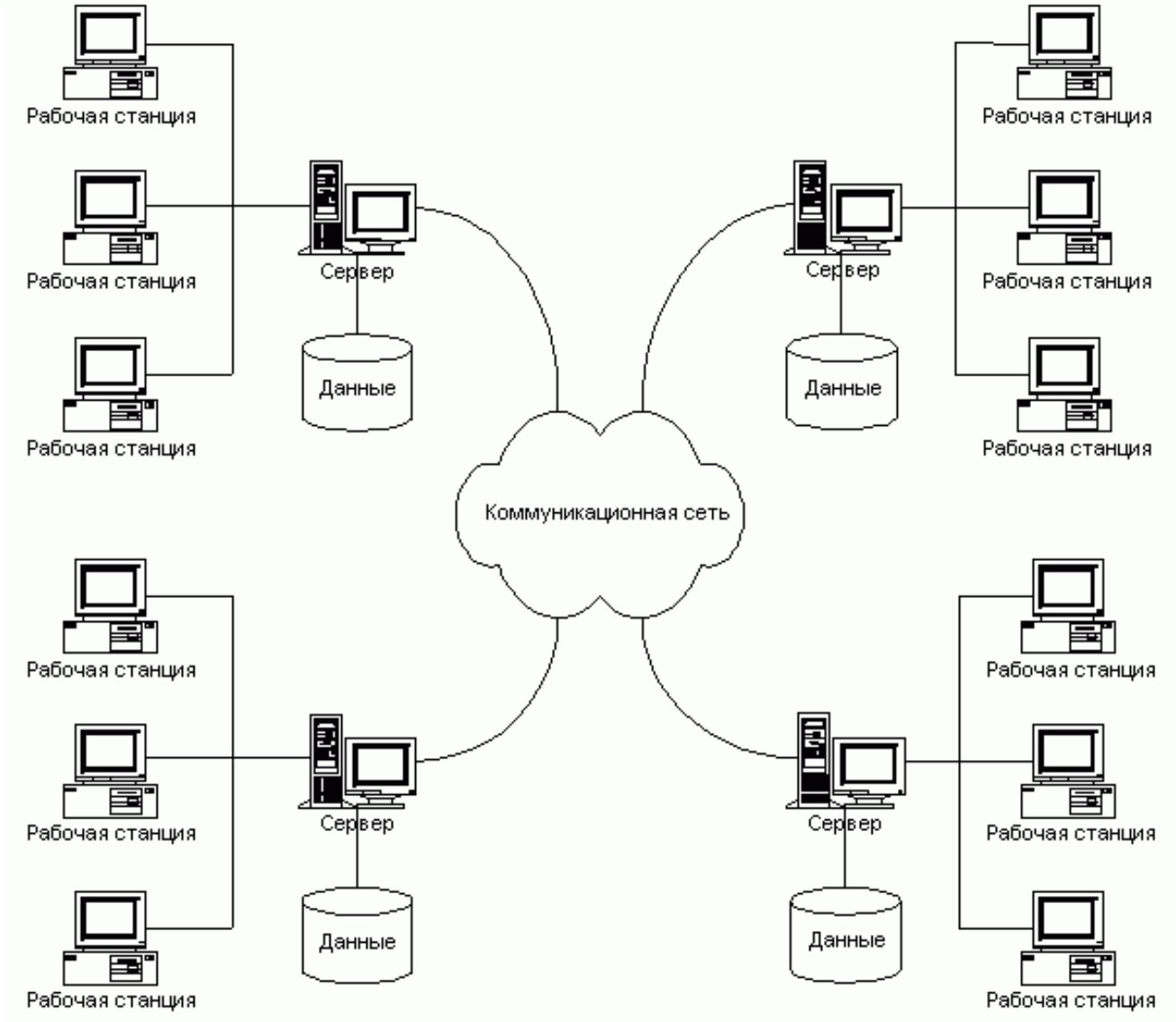
Приложение	Настройки приложения, утилизация ресурсов
Виртуальная машина	Выбор JVM, размер "кучи", сбор мусора
База данных	Размещение данных, кэширование объектов, масштабируемость, выбор JDBC-драйвера
Операционная система	Потоковые библиотеки, распределение процессорного времени, виртуальная память
Аппаратные средства/Сеть	ЦП, чипсет, память, устройства ввода/вывода, пропускная способность сети, коллизии

Масштабирование СУБД

Распределенная база данных

- *Распределенная база данных (РБД)* - совокупность логически взаимосвязанных баз данных, распределенных в компьютерной сети
- *Система управления распределенной базой данных (РСУБД)* - программная система, которая обеспечивает управление распределенной базой данных

Распределенная база данных



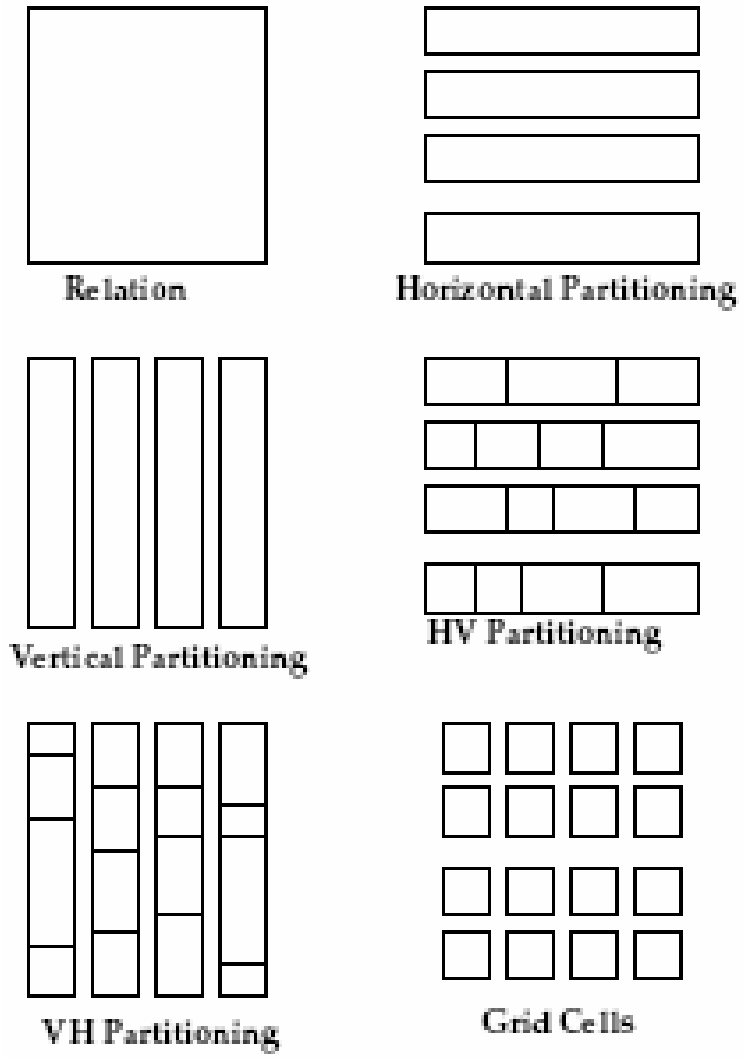
Свойства идеальной РБД (по Дейту)

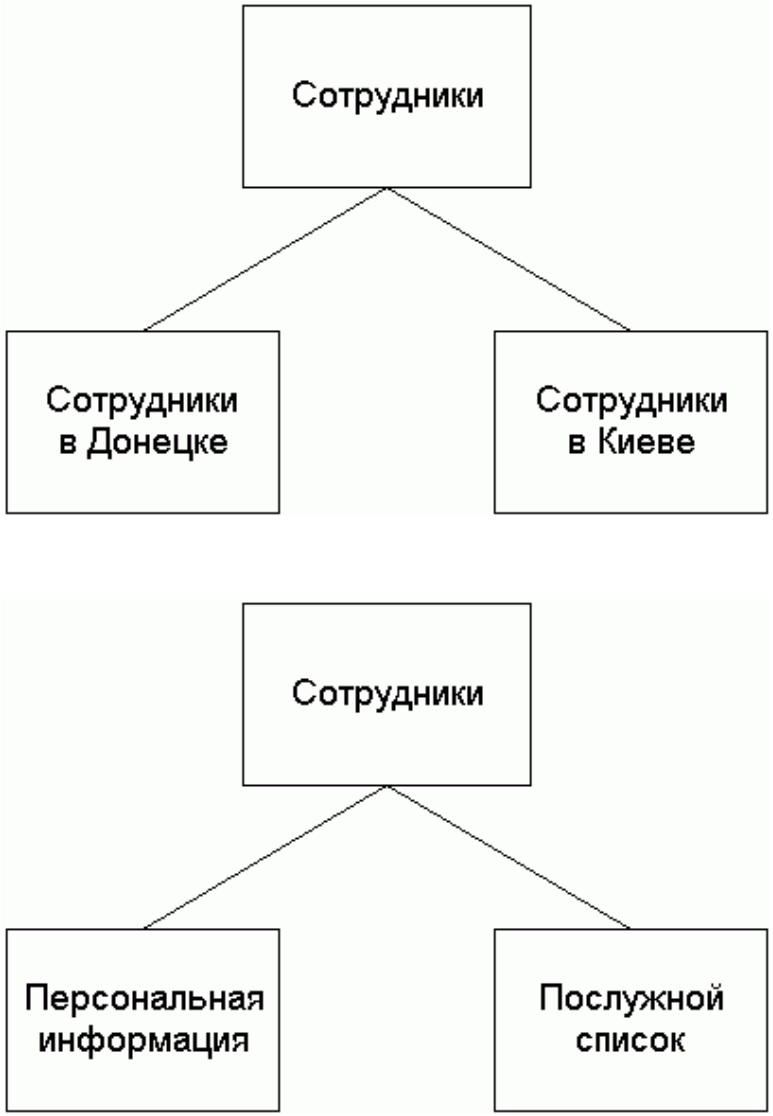
1. Локальная автономность
2. Независимость от центрального узла
3. Непрерывность функционирования
4. Прозрачность расположения
5. Прозрачная фрагментация
6. Прозрачность репликации

Свойства идеальной РБД (по Дейту)

7. Обработка распределенных запросов
8. Обработка распределенных транзакций
9. Независимость от оборудования
10. Независимость от операционных систем
11. Независимость от сети
12. Независимость от СУБД

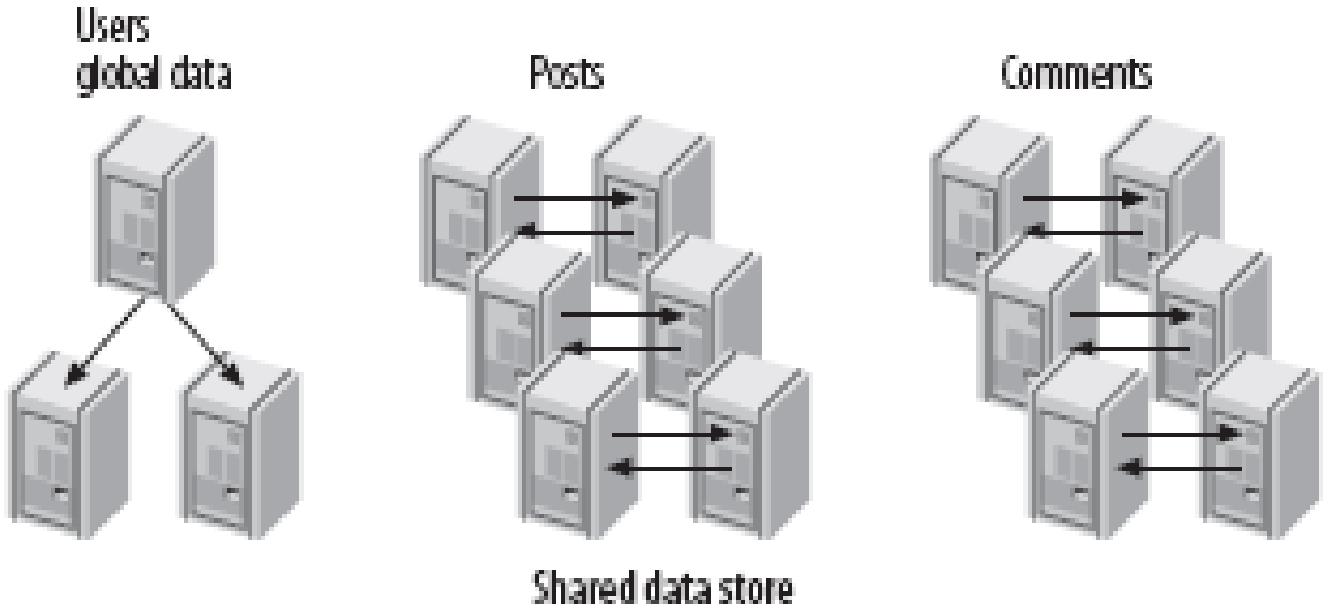
Фрагментация





Sharding (Сегментирование)

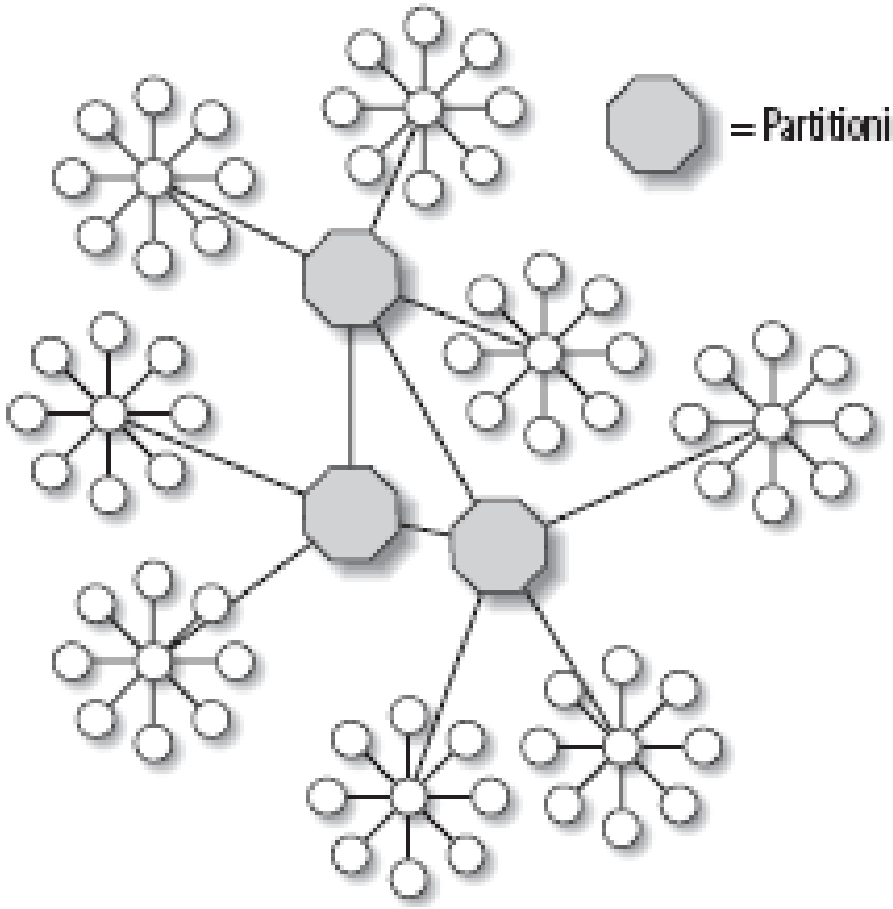
- Все данные разбиваются на части по какому-либо признаку
- Каждая часть хранится на отдельном сервере или кластере
- Такую часть данных в совокупности с системой хранения данных, в которой она находится, называют shard (сегментом)



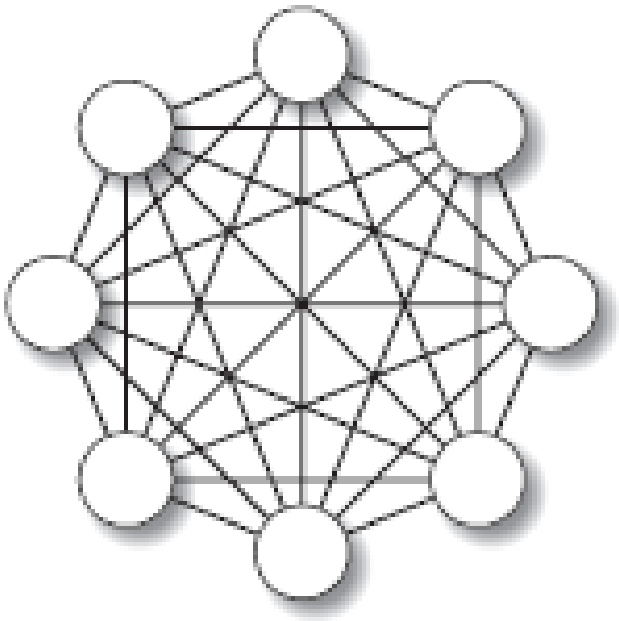
Признаки для сегментирования

- Диапазон
- Список
- Хэш-функция
- Композиция признаков

Проектирование схемы сегментированной БД



Легко сегментируется



Сегментируется с трудом

Достоинства сегментирования

- Данные распределяются по множеству физических экземпляров СУБД => увеличивается пропускная способность
- Относительно небольшой размер каждого сегмента позволяет держать их практически целиком в кэше, а также упрощает резервное копирование и восстановление данных
- Повышается доступность данных, потому что отказ одного сегмента не приводит к отказу всей системы

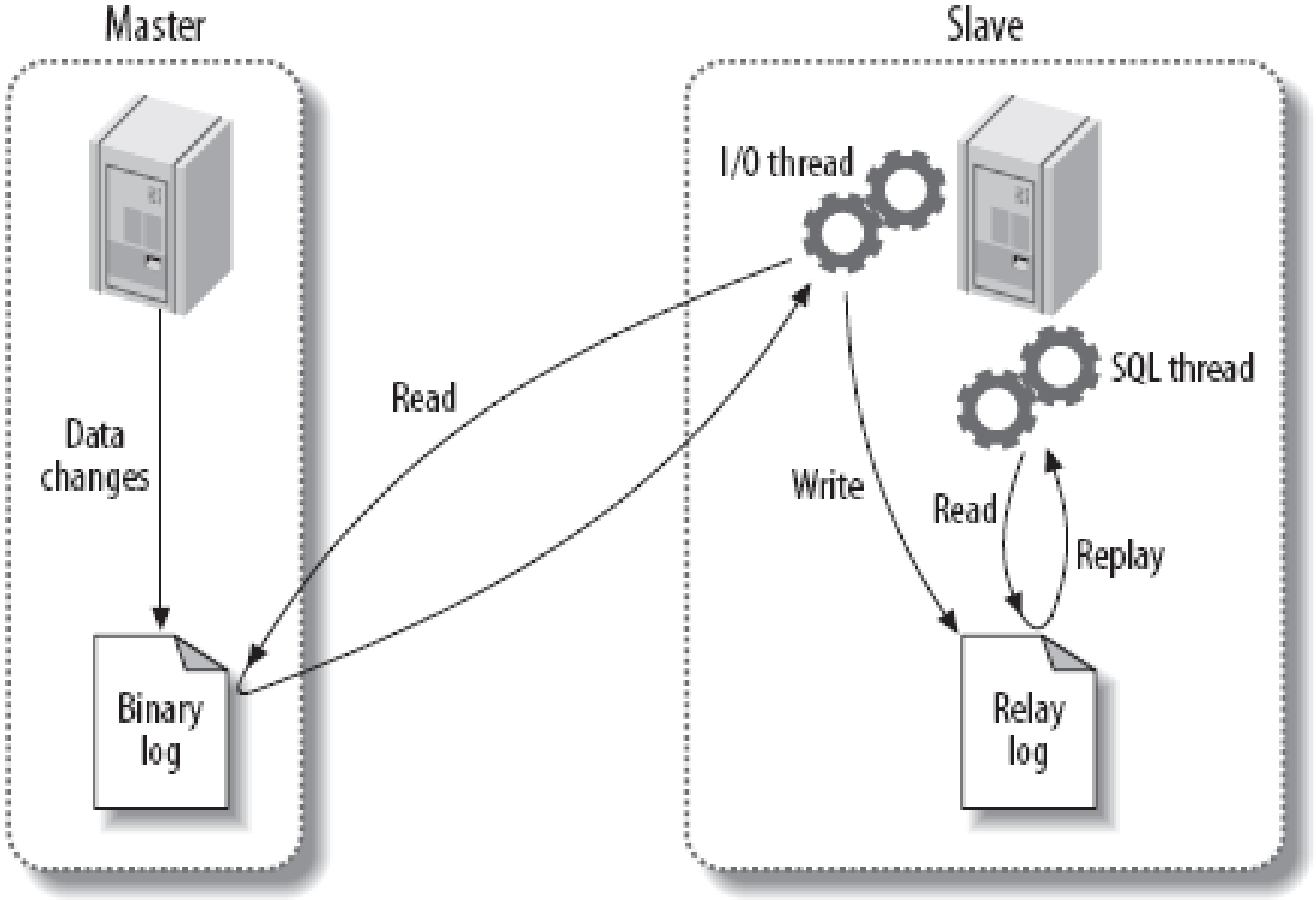
Задачи, которые необходимо решить для сегментирования

- Перераспределение данных
  - По мере заполнения сегментов может потребоваться изменить распределение данных в них
- Соединение данных из нескольких сегментов
  - Результаты запросов к различным сегментам агрегируются на программном уровне
- Реализация сегментирования — в приложении в слое доступа к данным
  - Готовые решения: GridSQL (на базе PostgreSQL), HiveDB (на базе MySQL), Hibernate Shards

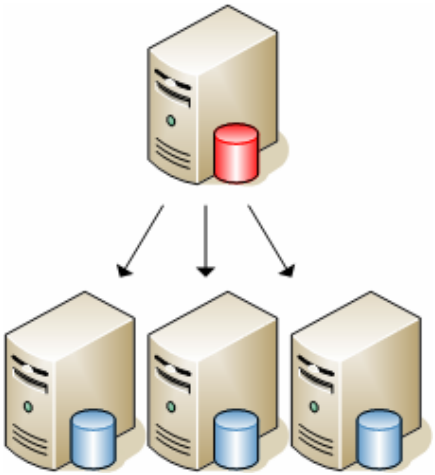
Репликация (тиражирование)

- Создание дубликатов данных
- *Репликаты* – это множество различных физических копий некоторого объекта базы данных (обычно таблицы), для которых поддерживается синхронизация (идентичность) с некоторой "главной" копией
- Виды репликации:
  - Синхронная
  - Асинхронная
    - По расписанию

Асинхронная репликация (на примере MySQL)



Топологии репликации: master-slave (главный-подчиненный)



Запись/чтение

Только чтение

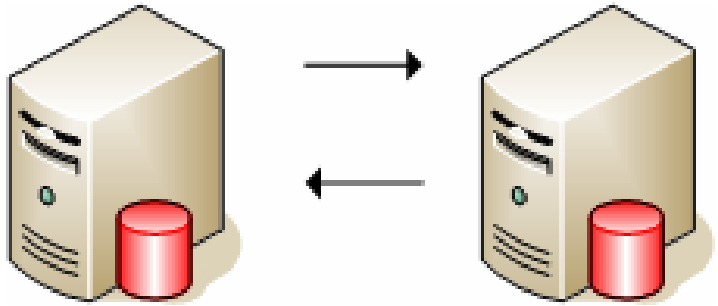
- Обеспечивает масштабирование по чтению
- Но недостаточная надежность (master — SPOF)

Репликация для масштабирования чтения

- Обычно в веб-приложениях соотношение чтение/запись находится в пределах от 80/20 до 90/10
- Пусть узел может обработать 1000 запросов/с, из них 800 — чтения, а 200 — записи
- Для удвоения производительности (2000 запросов/с, 1600/400), нужно
  - 1 master (400)
  - *n* slaves (1600 < (1000 - 400) * *n*), *n* = 3
- Для 4x (4000 запросов/с, 3200/800), нужно
  - 1 master (800)
  - 16 slaves (3200 = (1000 - 800) * 16)

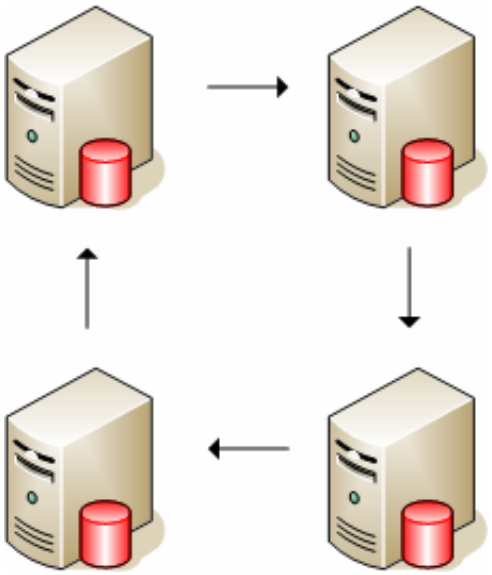
Нелинейная (!) масштабируемость

Топологии репликации: master-master (главный-главный)



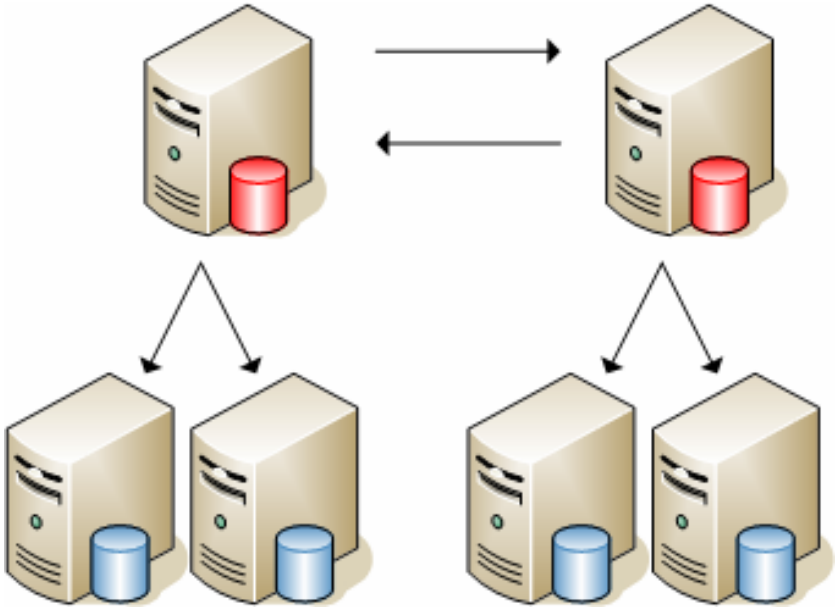
- Запись может выполняться на любом сервере
  - Возможны конфликты
    - `UPDATE tbl SET col=col + 1;` // на первом мастере
    - `UPDATE tbl SET col=col * 2;` // на втором мастере
  - Специальная настройка автоинкрементных полей
- Возможные режимы:
  - Активный-активный (используется редко)
  - Активный-пассивный
    - Высокая надежность, простое переключение

Топологии репликации: кольцо



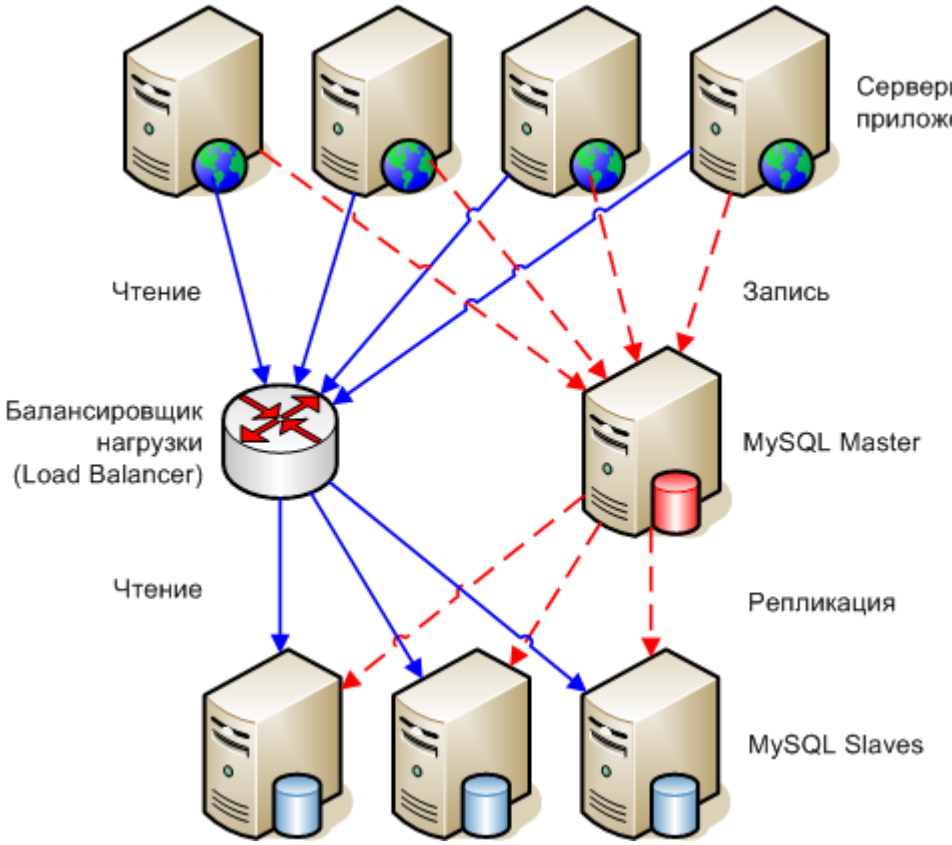
- Каждый узел является подчиненным для предыдущего и главным для следующего
- Необходимы специальные меры для защиты от отказов
  - Иначе невысокая надежность
- Топология «главный-главный» - это частный случай кольца, но имеет другие свойства

Топологии репликации: dual tree (двойственное дерево)

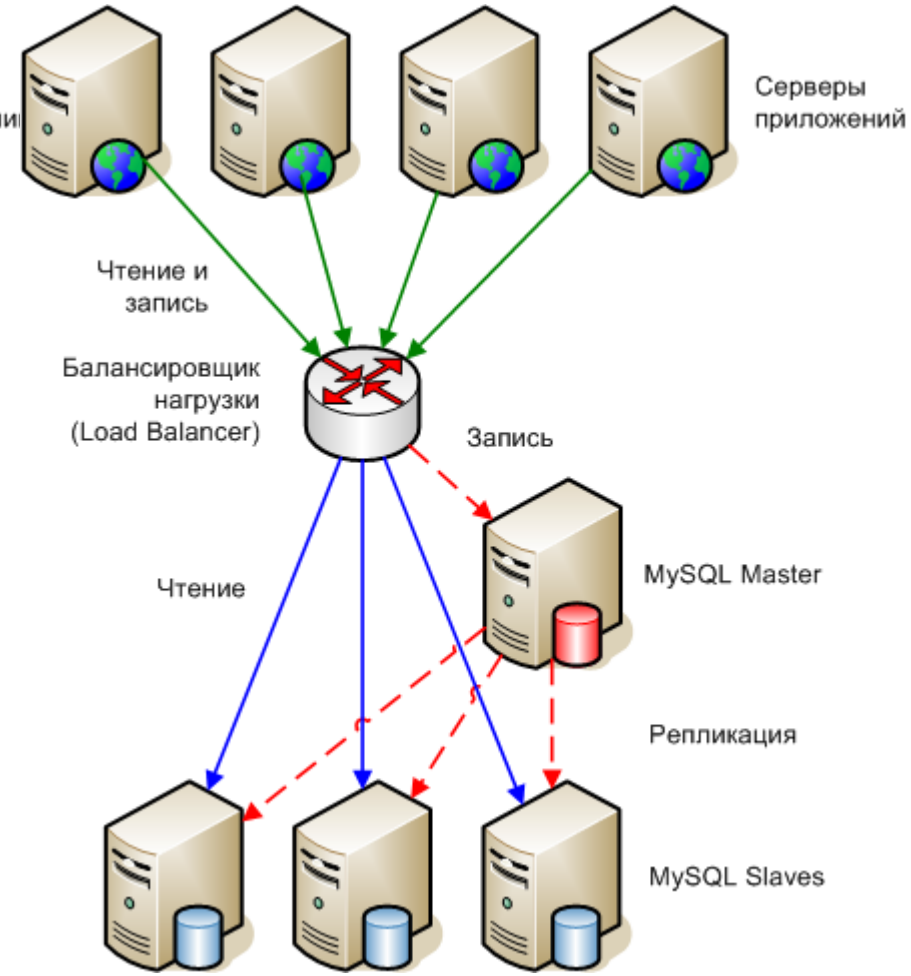


- Позволяет скомбинировать масштабирование по чтению и высокую надежность

Балансировка нагрузки



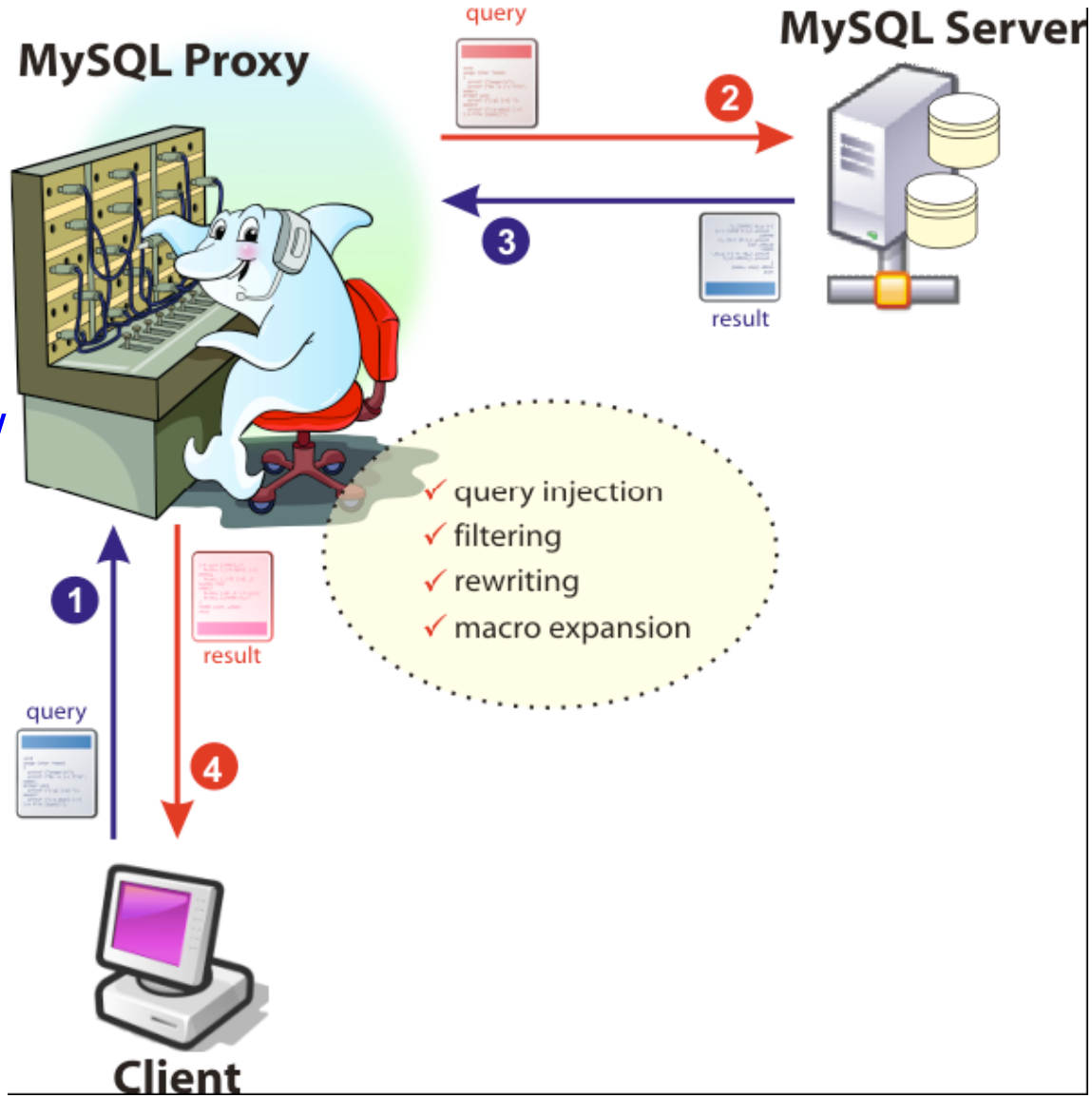
Вариант 1



Вариант 2

Балансировка нагрузки

- Аппаратная
- Программная
  - > MySQL Load Balancer
    - > На базе MySQL Proxy
  - > Специальный JDBC-драйвер с поддержкой LB

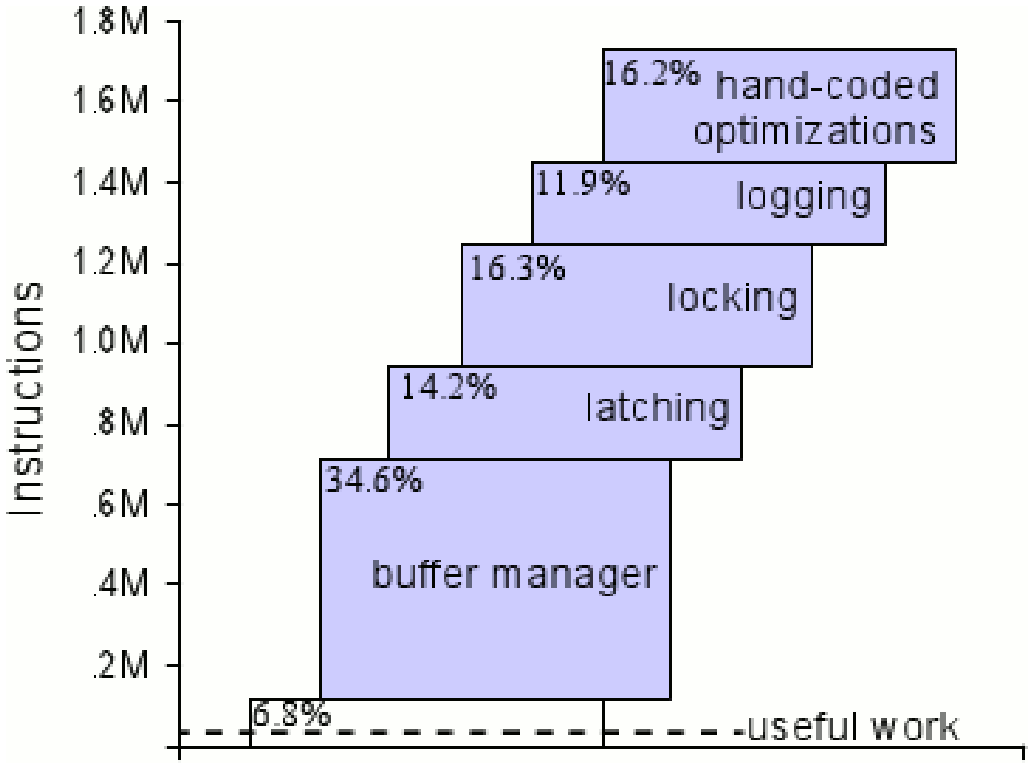


Тенденции в СУБД для OLTP

- Базы данных, хранимые в основной памяти
  - Например, MySQL Cluster (механизм хранения NDB)
- Однопоточный режим работы СУБД
- СУБД без журнализации
  - Восстановление БД не требуется либо выполняется на основе данных, хранимых в других узлах кластера
- СУБД без поддержки транзакций
  - В распределенных Internet-приложениях транзакционной согласованности часто предпочитается конечная согласованность (ACID)
  - Легковесные формы транзакций, например, такие, в которых все чтения должны быть произведены до первой записи

Тенденции в СУБД для OLTP

- Журнализация
- Блокировки
- Защелки
- Управление буферами



Анализ выполнения команд в разных подсистемах СУБД Shore для транзакции New Order из TPC-C

Масштабирование сервера приложений

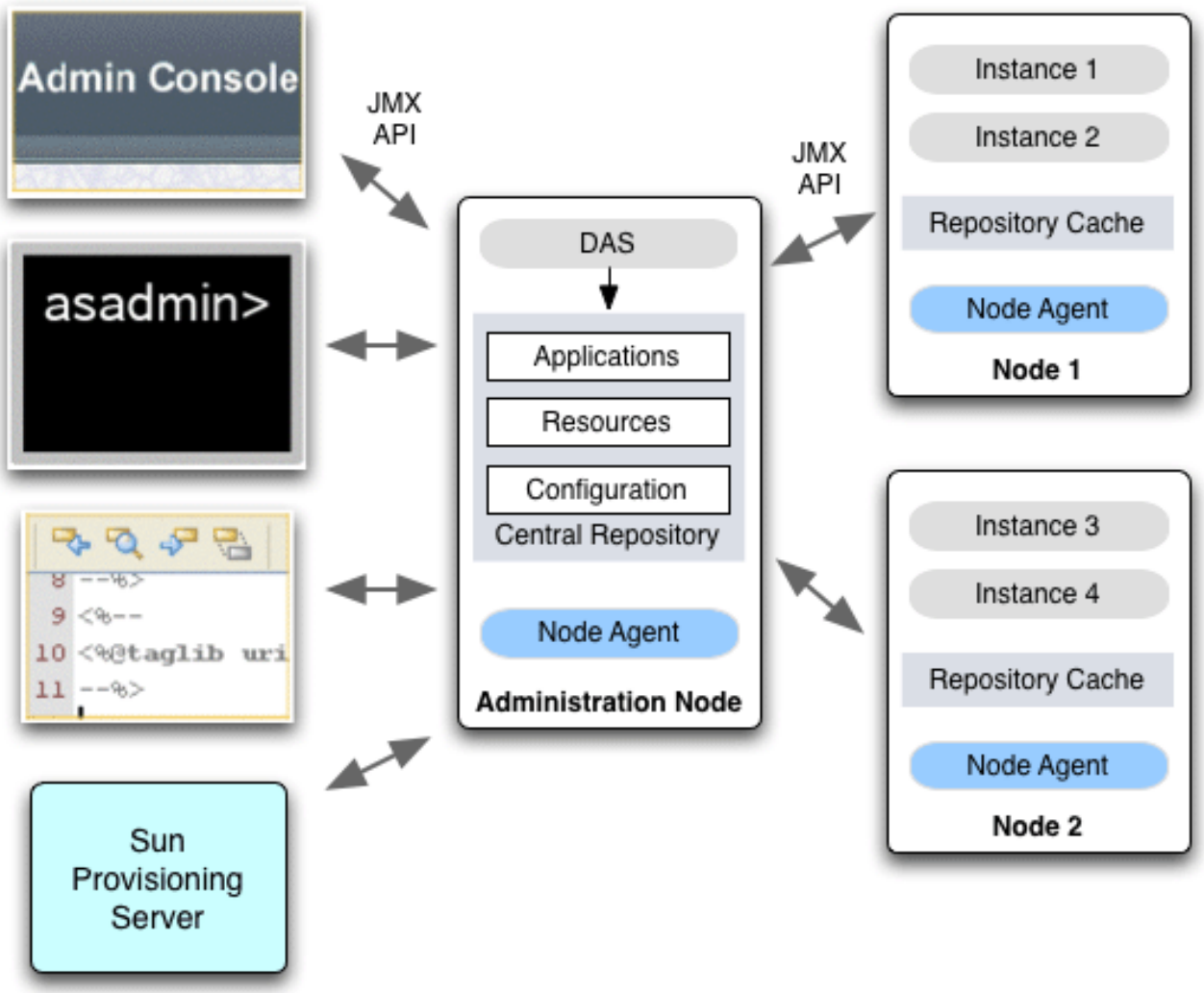
Кластеры и балансировка нагрузки

- Кластер — совокупность серверов, функционирующих как единое целое
- Кластер обеспечивает:
  - Горизонтальное масштабирование (High Performance)
  - Повышение надежности (High Availability — HA)
- Видимость «единого целого» создает балансировщик нагрузки
  - Распределяет запросы по серверам кластера
  - Аппаратные и программные балансировщики
  - В-основном, обрабатывают запросы по протоколу HTTP

Функции балансировщика нагрузки

- Реализация алгоритмов балансировки
  - Циклический (round-robin)
  - Случайный
  - Взвешенный
  - Адаптивный (с учетом нагрузки на сервер)
- Health check — мониторинг состояния кластера и обнаружение отказов
- Session stickiness — в рамках сессии запросы отправляются на тот же сервер, что и предыдущие

Архитектура кластера Glassfish



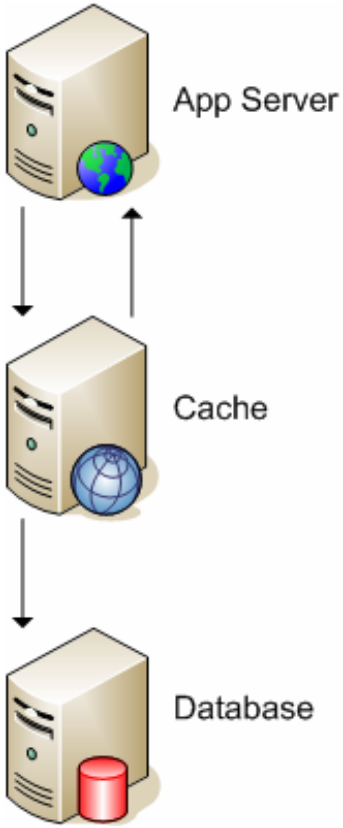
Архитектура кластера Glassfish

- Administration Domain (Домен) — обеспечивает согласованное управление кластерами и экземплярами сервера
- Instance (Экземпляр сервера) — JVM, выполняющая сервер приложений на одном узле
  - М.б. привязан только к одному кластеру
- Cluster (Кластер) — набор экземпляров серверов с одинаковой конфигурацией, приложениями и ресурсами
  - С точки зрения администрирования — единое целое

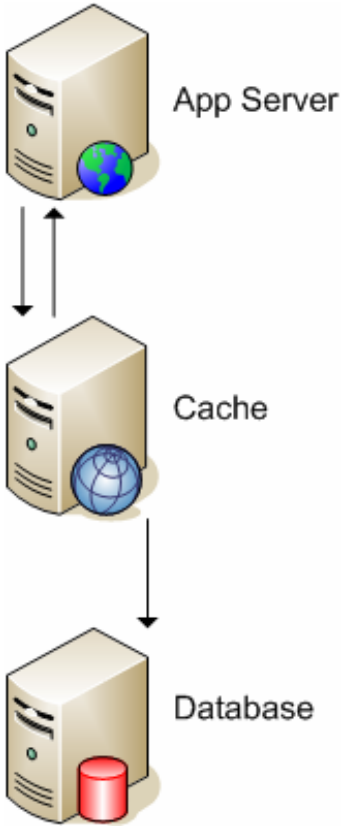
Архитектура кластера Glassfish

- Domain Administration Server (DAS) — экземпляр сервера, поддерживающий управление доменом
- Node Agent (Агент узла) — управляет ЖЦ экземпляров сервера на каждом узле, входящем в домен
- Central Repository (Центральный репозиторий) — хранит информацию, общую для всех экземпляров домена
  - Хранится в файловой системе
  - Запись выполняет только DAS
  - Содержит:
    - Репозиторий конфигурации домена
    - Репозиторий установленных приложений
- Repository Cache (Кэш репозитория)
  - хранится на экземпляре сервера
  - синхронизируется с центральным при перезапуске
  - ускоряет загрузку, может работать без DAS

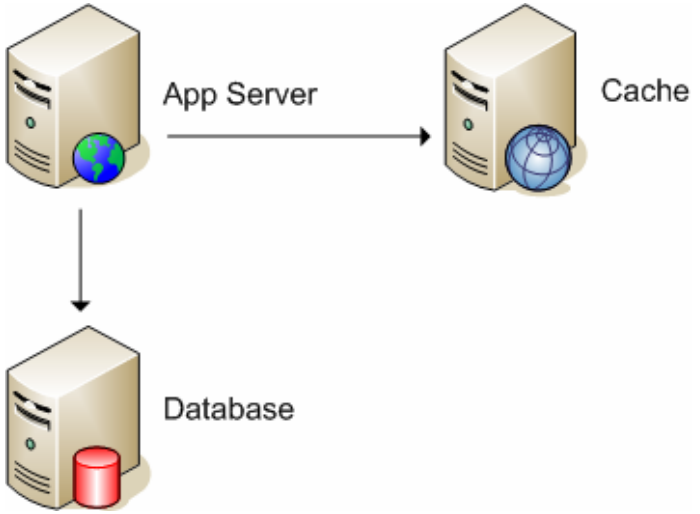
Кэширование



Сквозной кэш



Кэш с обратной записью

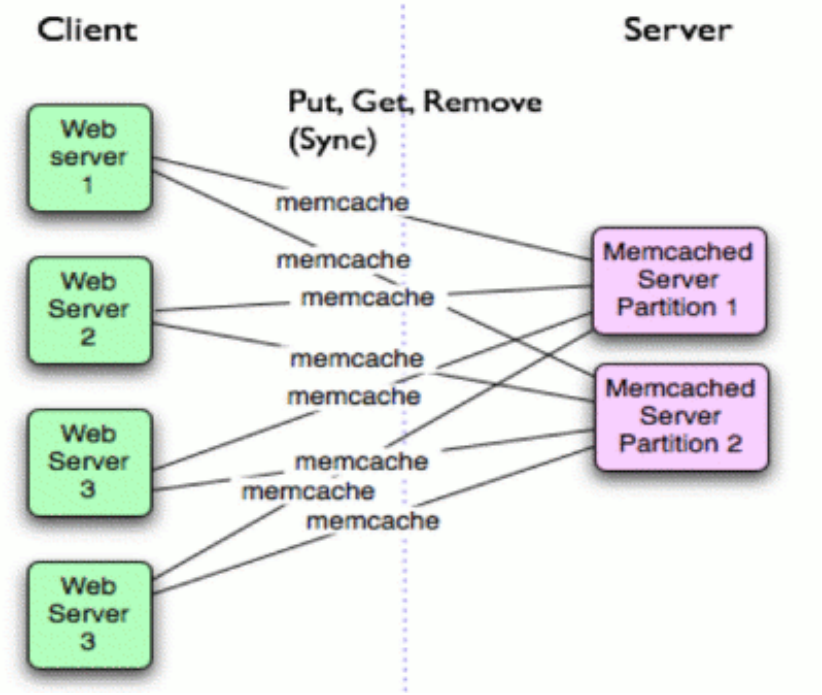


Sideline Cache (Побочный кэш)

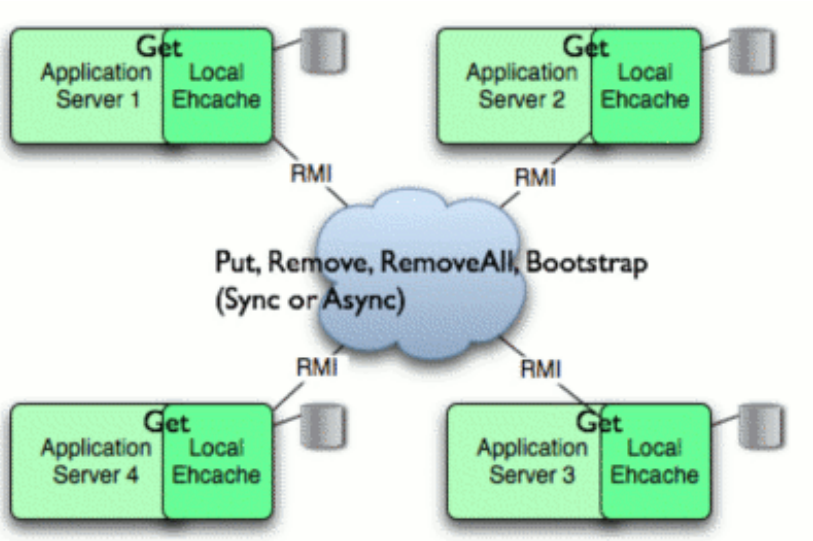
Sideline Cache

- Как правило, требуется внесение изменений в прикладную логику
  - Помещение объектов в кэш, их извлечение и удаление выполняется с помощью отдельного API
  - Существует API кэширования JCache (JSR-107)
- Инвалидация кэша не выполняется автоматически
- Исключение: подключение библиотеки кэширования к СОРП, например, к Hibernate

Реализации Sideline Cache



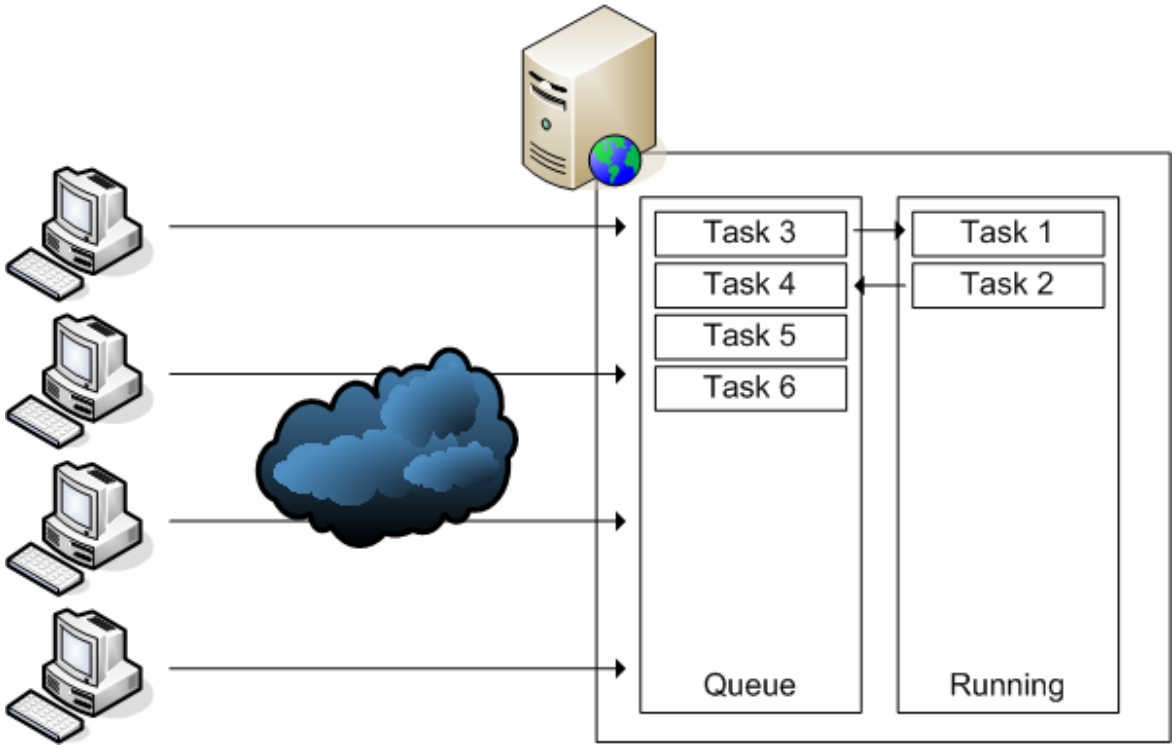
Memcached



ehCache

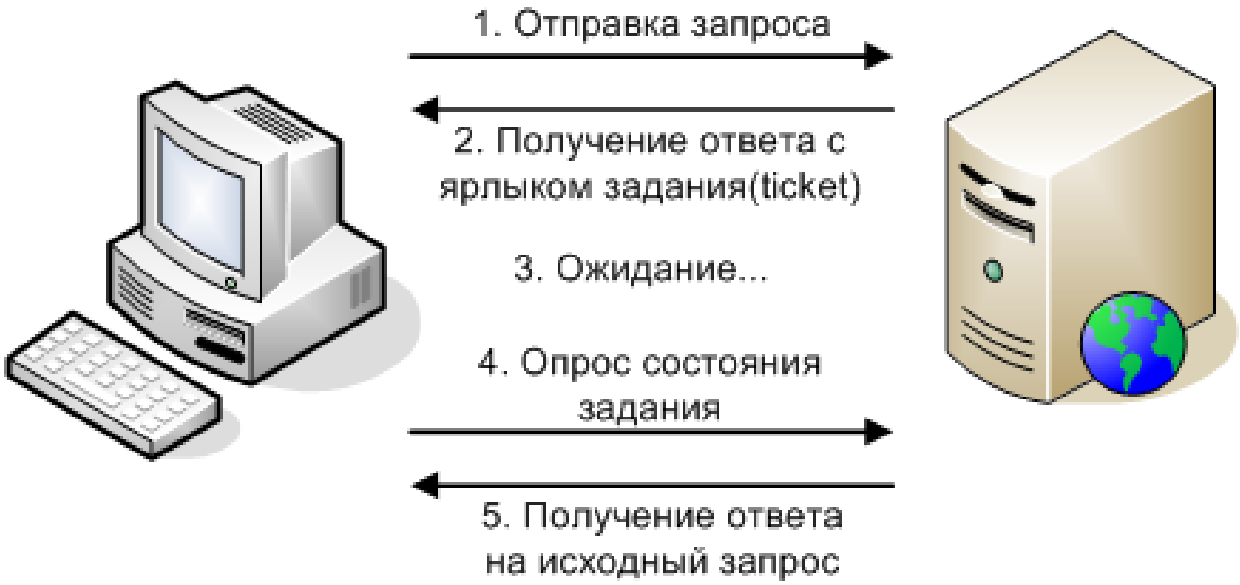
Асинхронное выполнение запросов

- Запросы помещаются в очередь и выполняются ограниченным числом потоков-обработчиков
- Это позволяет выровнять пиковые нагрузки



Асинхронное выполнение запросов

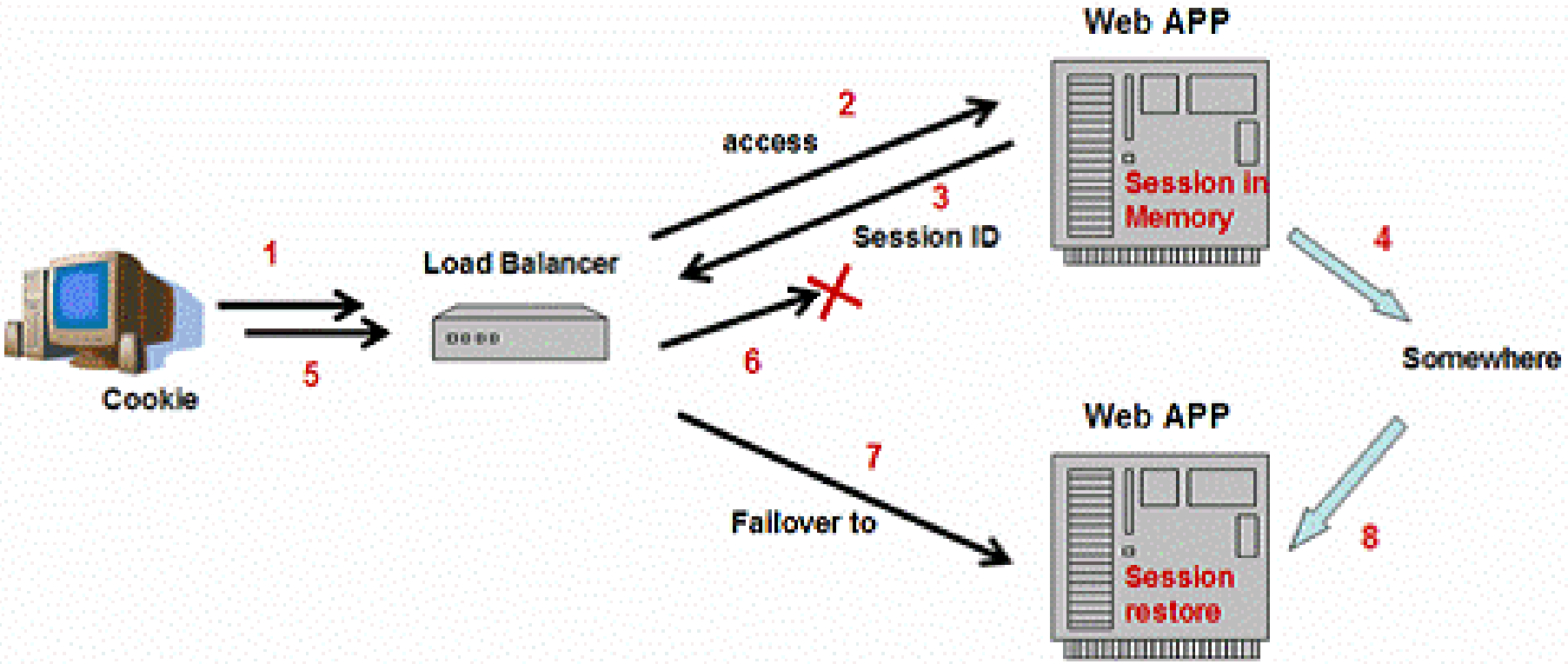
- Клиент опрашивает состояние своих запросов
- В веб-приложении возможны реализации:
  - На стороне клиента (JavaScript)
  - На стороне сервера (автоматическое обновление веб-страницы)



Сессии (состояние приложения)

- Локальные сессии
  - Хранение в оперативной памяти или на диске
  - Пользователь привязан к конкретному серверу
  - Перегрузка отдельных серверов («горячие» точки)
- Централизованное хранение сессий
  - В базе данных
  - Нет «горячих» точек, но требуется масштабируемое и надежное хранилище данных (SPOF)
- Децентрализованное хранение сессий
  - Например, в кэше (Memcached)
  - Нет «горячих» точек, нет SPOF
- Отказ от сессий <-- подходит для сайтов без персонализации
  - ID пользователя и некоторые другие сведения можно хранить в куки (+ ЭЦП), остальное считывать из БД (кэша)

Отказоустойчивость HTTP-сессий



- Глобальный идентификатор сессии
- Способ резервирования состояния сессии
- Частота и объем резервирования

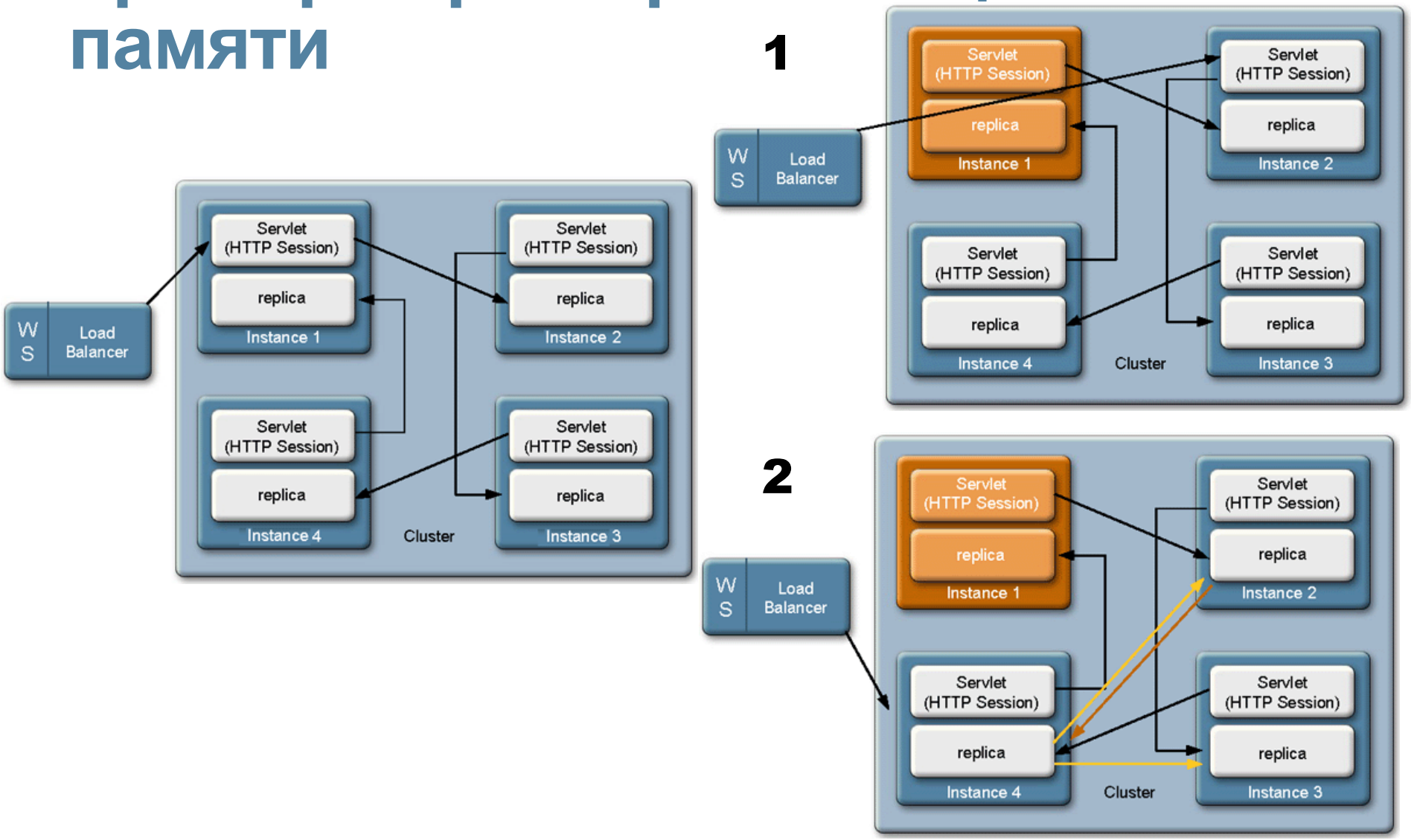
Отказоустойчивость HTTP-сессий: хранение состояния в БД

- В БД сохраняется сериализованное состояние сессии
- Недостатки:
  - Низкая производительность
  - Ограниченная масштабируемость при хранении больших или многочисленных объектов в сессии
- Достоинства:
  - Простота реализации — поддерживается практически всеми серверами приложений
  - Сессия может быть обработана любым сервером
  - Данные сессии останутся даже в случае отказа всего кластера

Отказоустойчивость HTTP-сессий: репликация состояния в памяти

- Состояние сессии реплицируется на все узлы кластера (Tomcat) или на соседний узел (Glassfish, Weblogic, JBoss, WebSphere)
- Достоинства:
  - Высокая производительность
  - Высокая масштабируемость
- Недостатки:
  - Накладные расходы на репликацию (время, память)
  - При отказе узла нагрузка на соседний узел возрастает вдвое

Отказоустойчивость HTTP-сессий: пример парной репликации в памяти



Специфика Java EE-приложений

- Только сериализуемые объекты в сессии
- Нельзя использовать статические переменные
- В web.xml:

```
> <web-app version="2.5" ..... >
    <display-name>webapp</display-name>
    <distributable/>
```

- В sun-web.xml (для Glassfish):

```
> <sun-web-app>
    <session-config>
      <session-manager persistence-
  type="replicated">
    ...
    </session-config>
```

Архитектура LiveJournal

Архитектура Flickr