

Автоматизация развертывания и эксплуатации программного обеспечения

Раздел «Базы данных»

Лекция №13

Виноградова Мария Валерьевна

МГТУ им. Н.Э. Баумана, 2022 г.

Темы лекции

- Базы данных NoSQL:
 - графовые,
 - колоночные,
 - ключ-значение,
 - документные
- Возможности, модели данных и языки запросов к ним.
- Вопросы согласованности данных.
- Репликация и фрагментация; типы и настройки репликации.
- Кластеры баз данных, их развертывание и настройка.

Реляционные и объектно-реляционные базы данных

- Широкое распространение.
- Атомарные, составные и пользовательские типы данных.
- Программирование на стороне сервера.
- Многопользовательская работа и поддержка ACID-транзакций.
- Стандартная модель данных и язык SQL.
- Оптимизация запросов и нормализация отношений.
- Наличие индексов и управление вторичной памятью.

Ограничения РБД и ОРБД

- Фиксированная схема данных.
- Потеря согласованности
 - различие структур данных в БД и ПО,
 - атомарные типы данных,
 - сложность отображения составных объектов.
- Сложность создания больших кластеров при поддержке ACID-транзакций.

История NoSql

- Семинар 2009 г., Сан-Франциско (**Йохан Оскарссон**)
- Решения – прообразы:
 - BigTable (Google)
 - Dynamo (Amazon)
- Направление NoSQL:
 - распределенные,
 - открытый исходный код,
 - не реляционные.

Особенности NoSql

- Нет определенного термина.
- Не используют SQL.
- Открытый исходный код (не всегда).
- Работа на кластерах (есть графовые).
- БД после 2000 г.
- Учитывают объемы веб.
- Без схемы данных (гибкая структура).

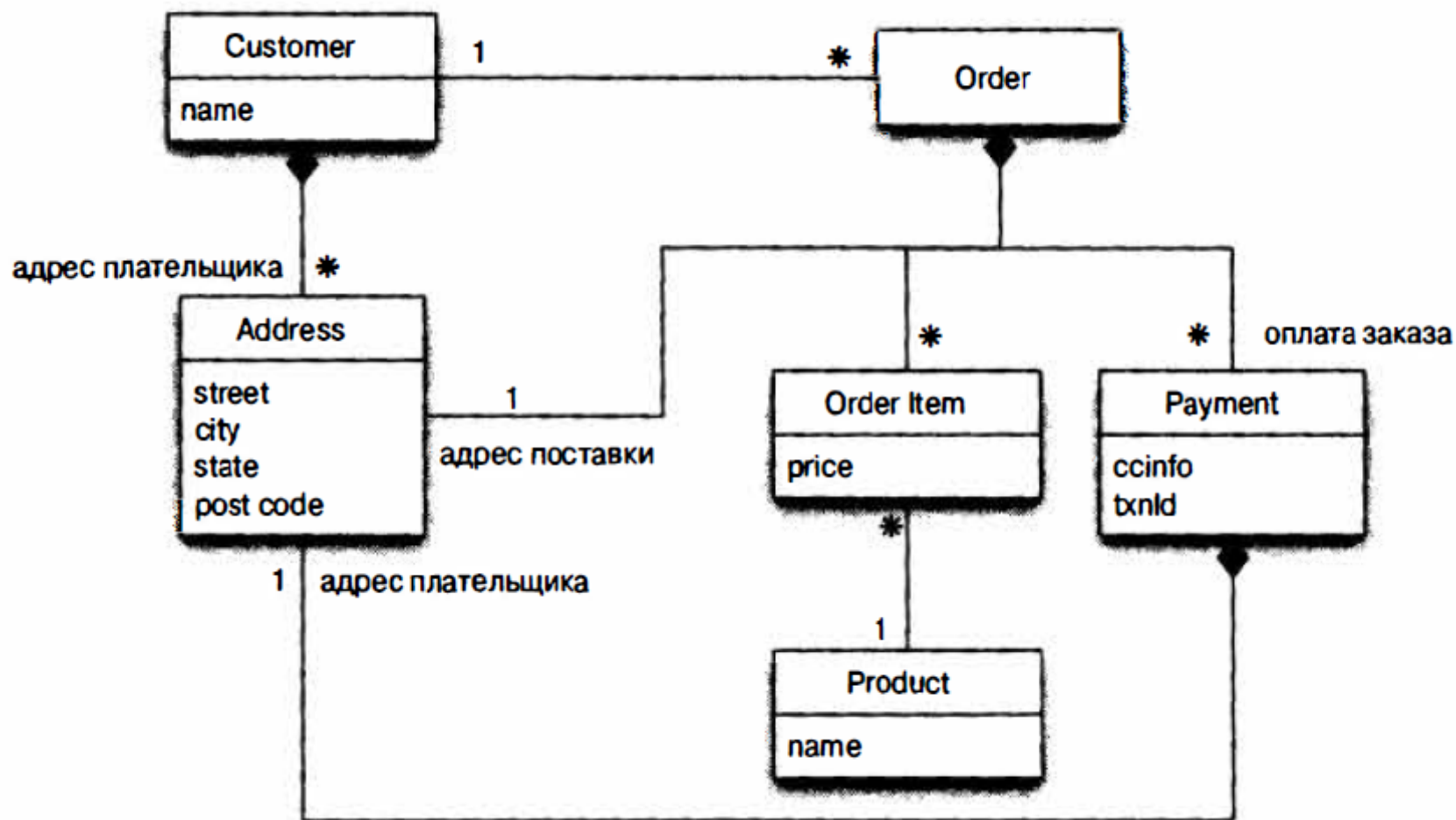
Модели данных NoSql

- Агрегатные
 - Ключ-значение
 - Документные
 - Семейство столбцов (колоночные)
- Неагрегатные
 - Графовые

Агрегат

- Единица хранения и обработки данных.
- Коллекция связанных объектов как единое целое.
- Элемент репликации, фрагментации, приложения.
- Структура агрегата зависит от приложения.
- Нет правил определения.

Пример ОО модели

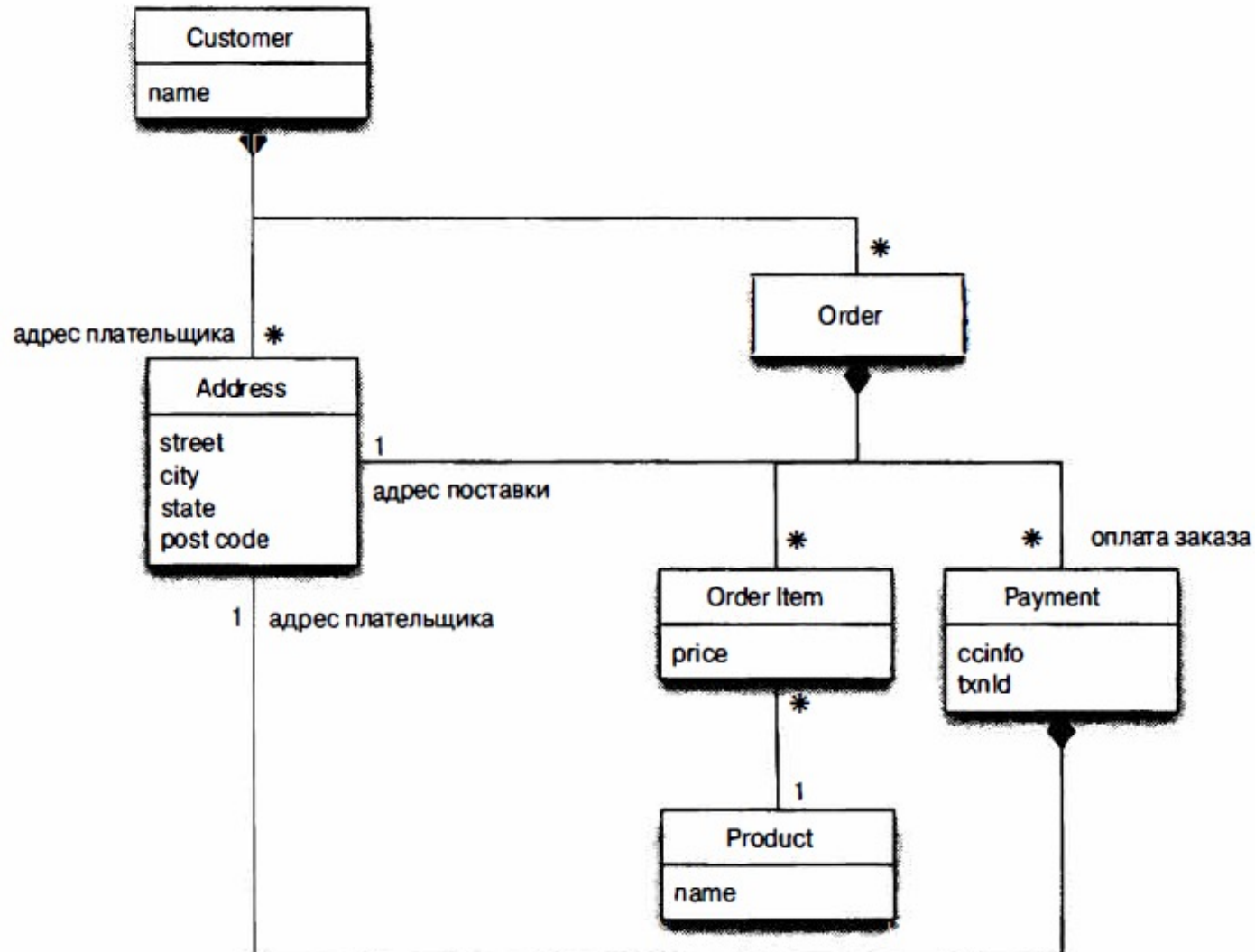


Пример JSON

```
// Клиенты
{
  "id":1,
  "name":"Martin",
  "billingAddress":[{"city":"Chicago"}]
}

// Заказы
{
  "id":99,
  "customerId":1,
  "orderItems":[
    {
      "productId":27,
      "price": 32.45,
      "productName": "NoSQL Distilled"
    }
  ],
  "shippingAddress":[{"city":"Chicago"}]
  "orderPayment":[
    {
      "ccinfo":"1000-1000-1000-1000",
      "txnId":"abelif879rft",
      "billingAddress": {"city": "Chicago"}
    }
  ],
}
```

Другой агрегат



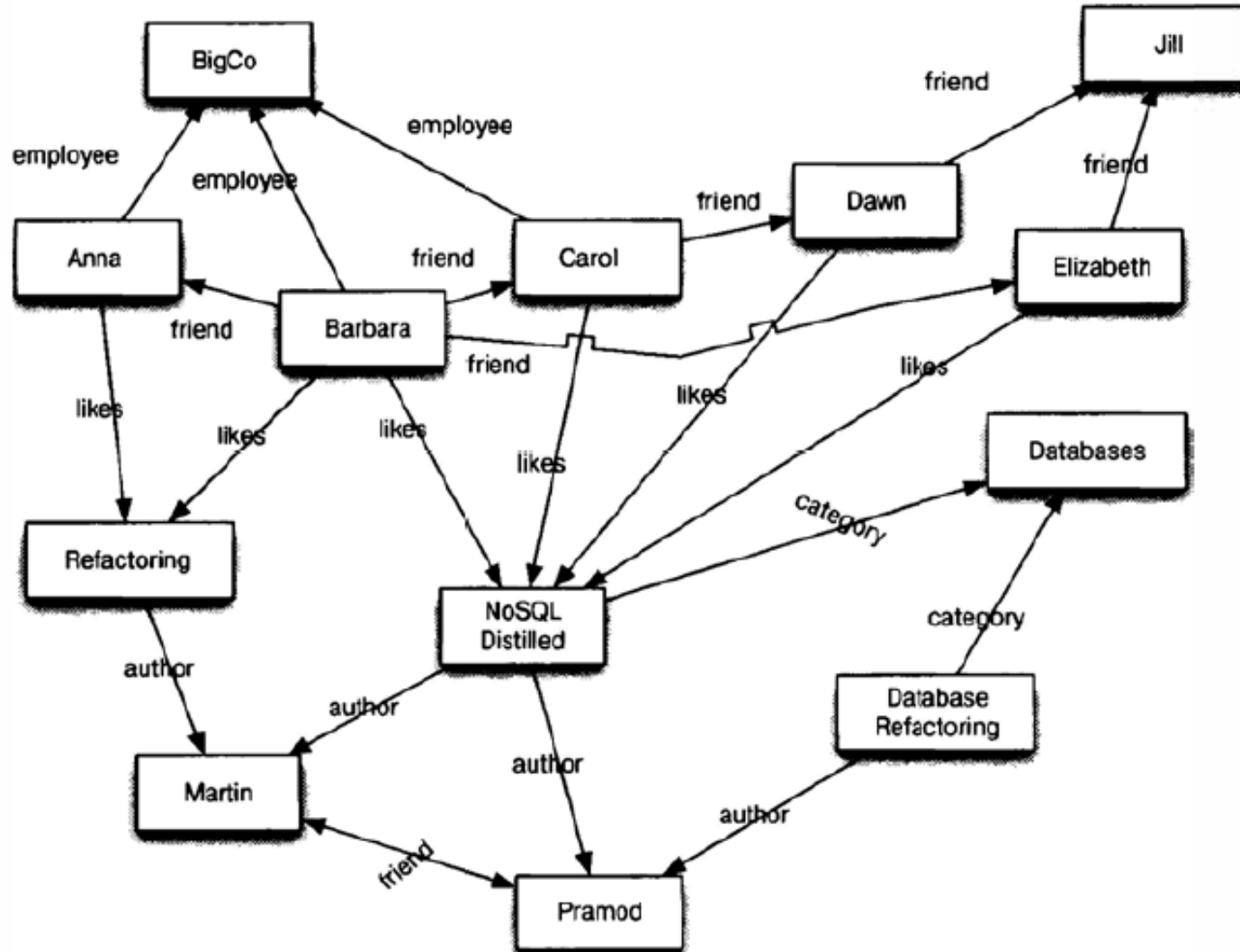
Другой JSON

```
// Клиенты
{
  "customer": {
    "id": 1,
    "name": "Martin",
    "billingAddress": [{"city": "Chicago"}],
    "orders": [
      {
        "id": 99,
        "customerId": 1,
        "orderItems": [
          {
            "productId": 27,
            "price": 32.45,
            "productName": "NoSQL Distilled"
          }
        ]
      },
      {
        "id": 100,
        "customerId": 1,
        "orderItems": [
          {
            "productId": 27,
            "price": 32.45,
            "productName": "NoSQL Distilled"
          }
        ]
      }
    ],
    "shippingAddress": [{"city": "Chicago"}],
    "orderPayment": [
      {
        "ccinfo": "1000-1000-1000-1000",
        "txnId": "abelif879rft",
        "billingAddress": {"city": "Chicago"}
      }
    ]
  }
}
```

Агрегатные БД

- Ключ-значение
 - Агрегат – черный ящик.
 - Поиск по ключу.
- Документные
 - Агрегат – документ (JSON).
 - Поиск по внутренней структуре документа.
- Колоночные
 - Агрегат - строка, содержит ассоциативный массив столбцов.
 - Работа со строками и со столбцами.

Графовые БД



Масштабирование БД

- Вертикальное
 - Есть предел,
 - Дорого.
- Горизонтальное (кластеры)
 - Лицензии,
 - транзакции,
 - запросы,
 - Оптимизация.

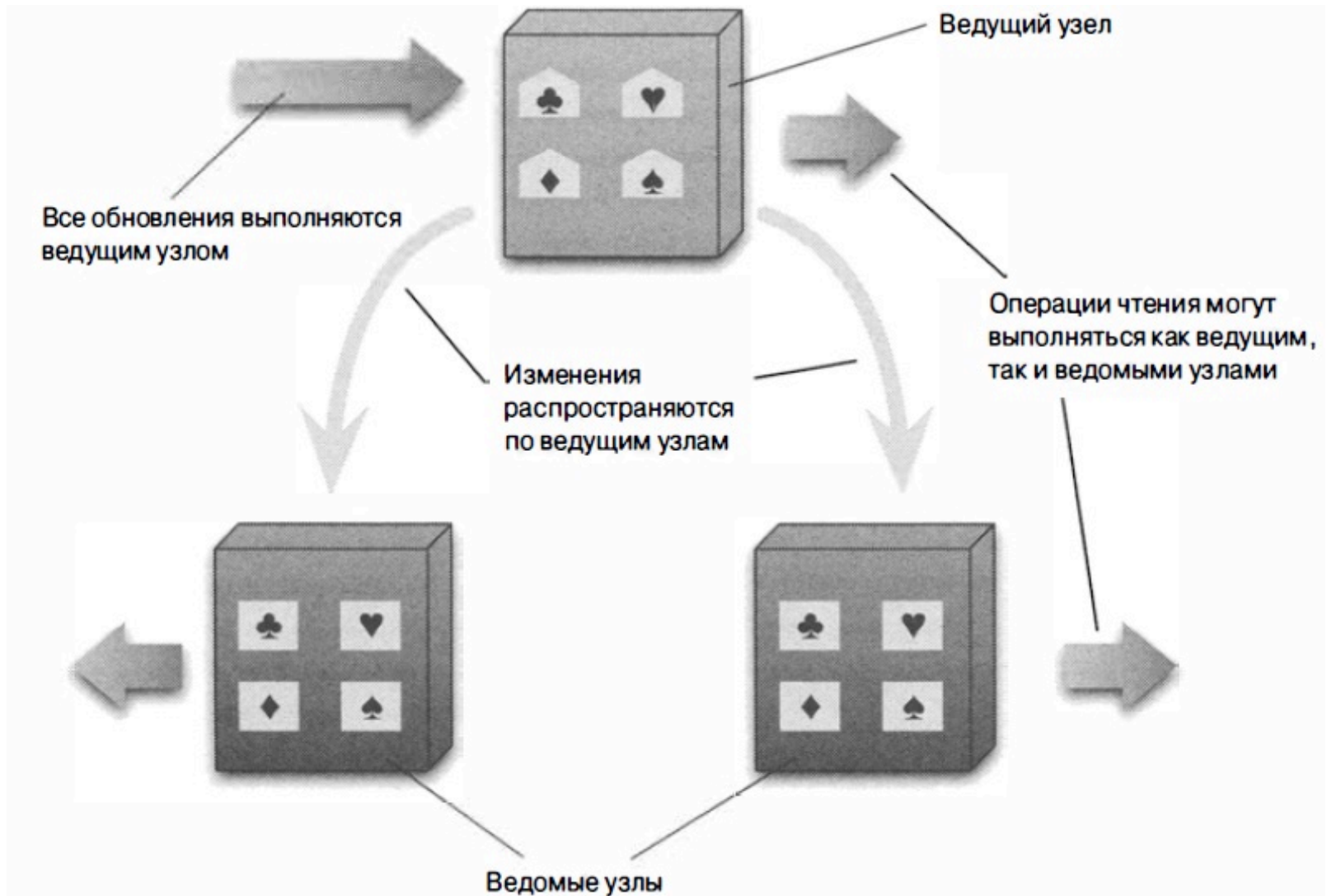
Модели распределения данных

- **Репликация** - копирование одних и тех же данных на нескольких узлах
 - Односерверная
 - Ведущий –ведомый
 - Одноранговая
- **Фрагментация** - размещение разных данных на разных узлах

Односерверная репликация

- Лучший вариант
- Графовые БД
- Ключ-значение
- Документные БД

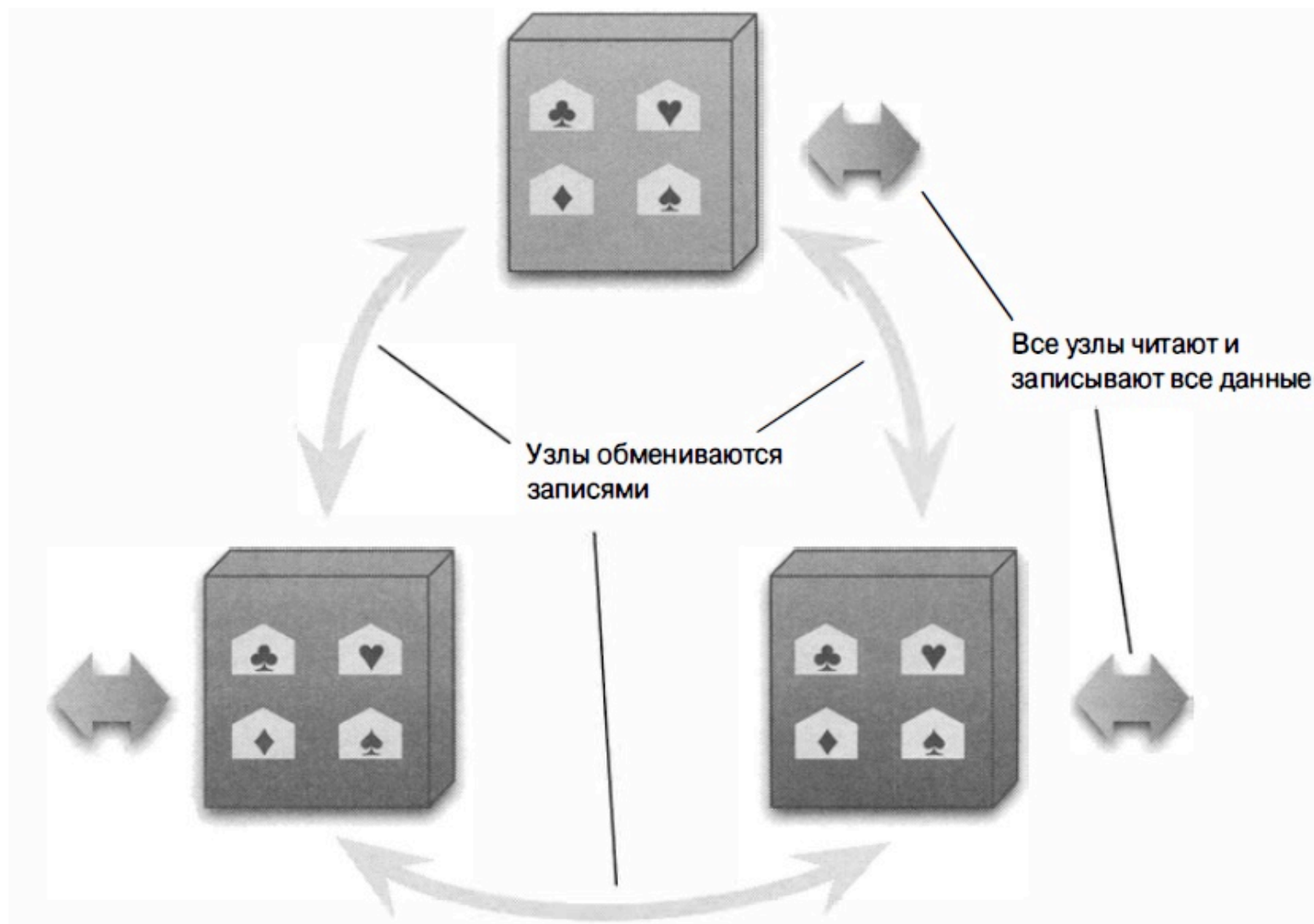
Репликация Ведущий-ведомый



Обособенности Ведущий-ведомый

- Уменьшает время чтения.
- Увеличивает время записи.
- Отказоустойчивость чтения.
- Оперативный резерв.
- Назначение - ручное/автомат.
- Возможная несогласованность.

Одноранговая репликация



Особенности одноранговой

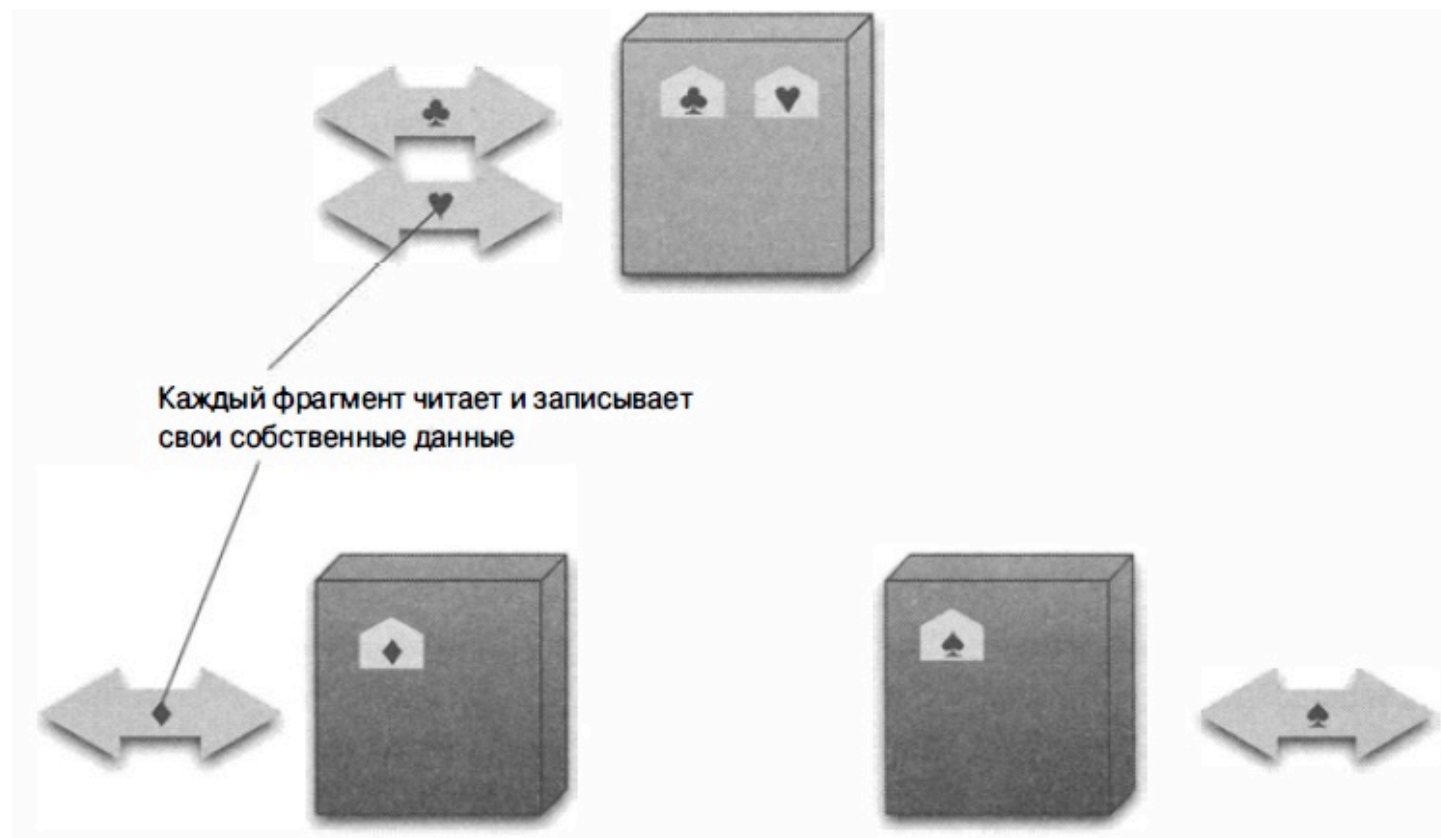
(+)

- Отказоустойчивость.
- Производительность.

(-)

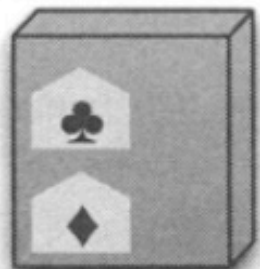
- Конфликты записи.

Фрагментация

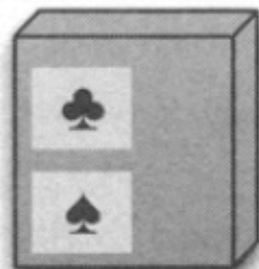


Сочетание репликации Ведущий-Ведомый и фрагментации

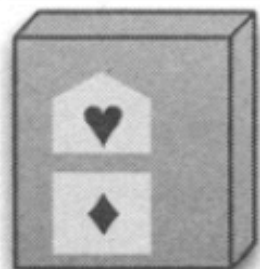
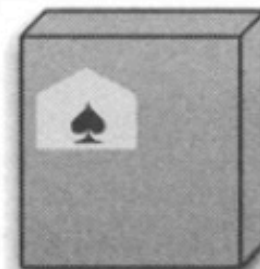
Ведущий узел для
двух фрагментов



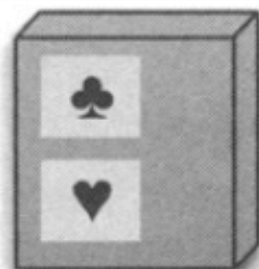
Ведомый узел для
двух фрагментов



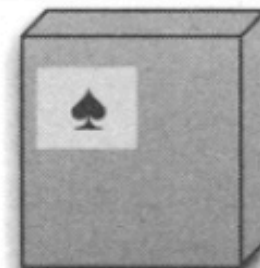
Ведущий узел для
одного фрагмента



Ведущий узел для
одного фрагмента
и ведомый для другого



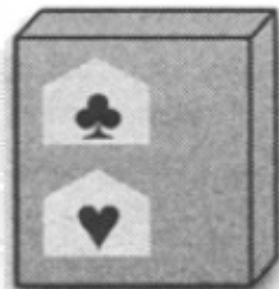
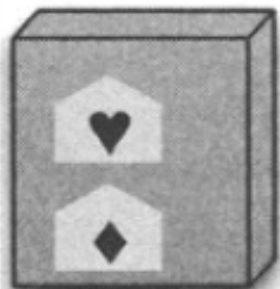
Ведомый узел для
двух фрагментов



Ведомый узел для
одного фрагмента

Сочетание Одноранговой репликации и фрагментации

Для каждого фрагмента – свой набор реплик.



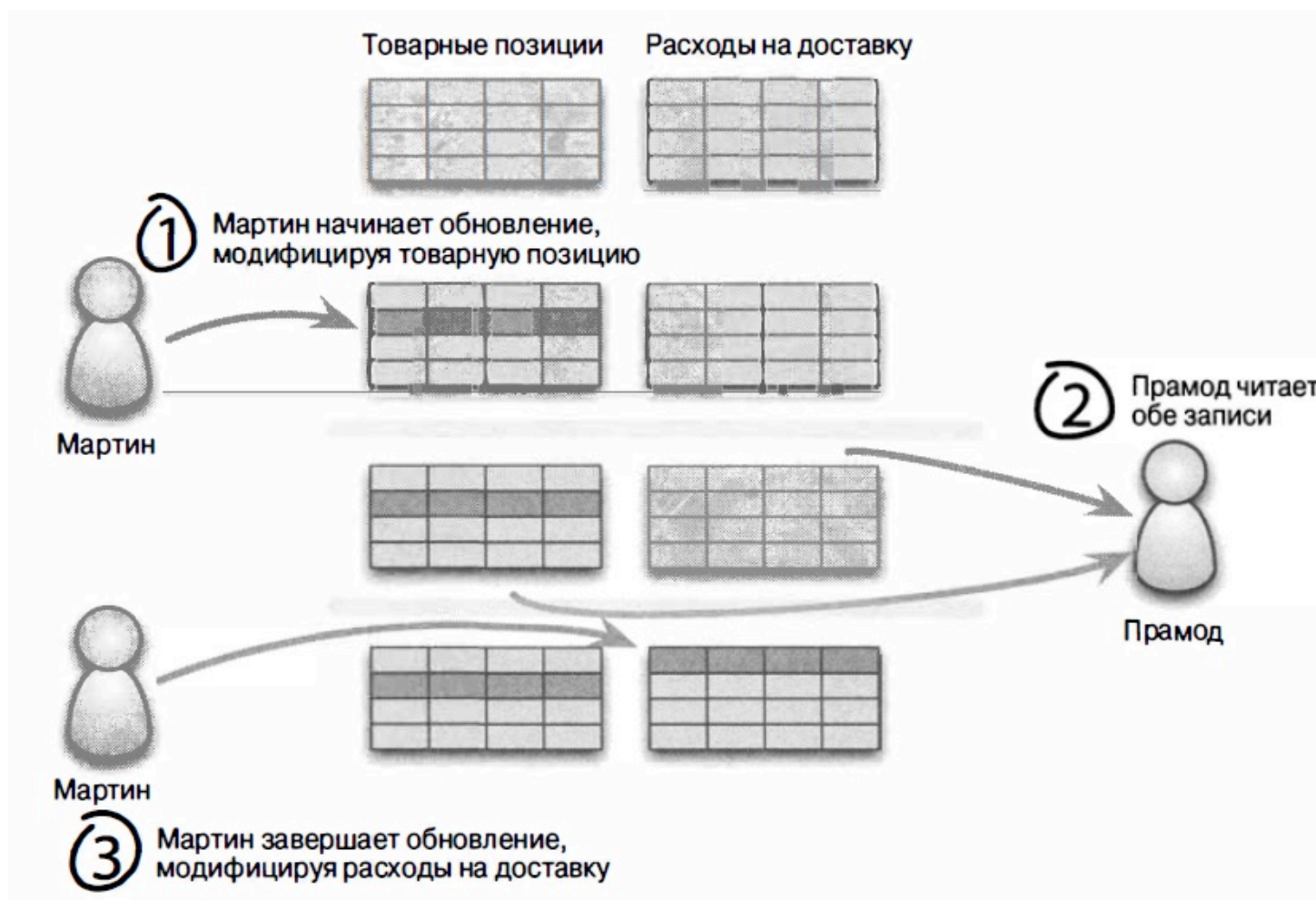
Проблемы согласованности

- Согласованность обновления.
- Согласованность чтения.

Согласованность обновления

- Конфликт Запись-запись
- Двое изменяет одни данные в один момент
- Потеря изменений
- Пессимистический подход к решению
 - Предотвращает,
 - Блокировка
 - (-) долго, взаимные блокировки
- Оптимистический подход к решению
 - Допускает, выявляет, устраняет
 - Условное обновление (чтение - проверить изменения - запись)
 - Слияние и разрешение конфликта (ручное, автоматическое)
- Усиление при репликации

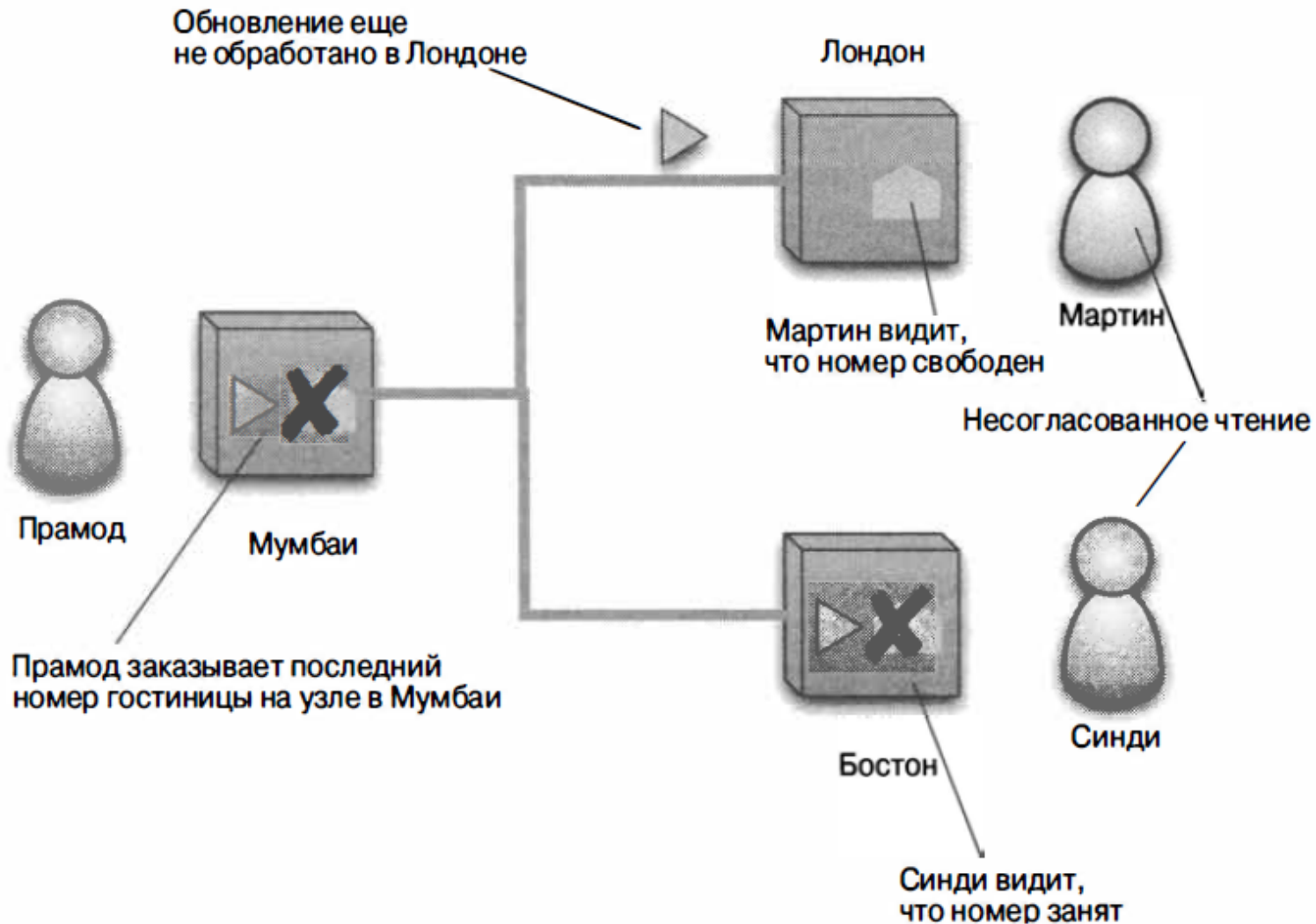
Пример несогласованности чтения



Согласованность чтения

- Конфликт чтение-запись.
- Логическая несогласованность.
- Причина - транзакция на агрегат.
- Окно несогласованности (период чтения несогласованных данных).

Пример несогласованности репликаций



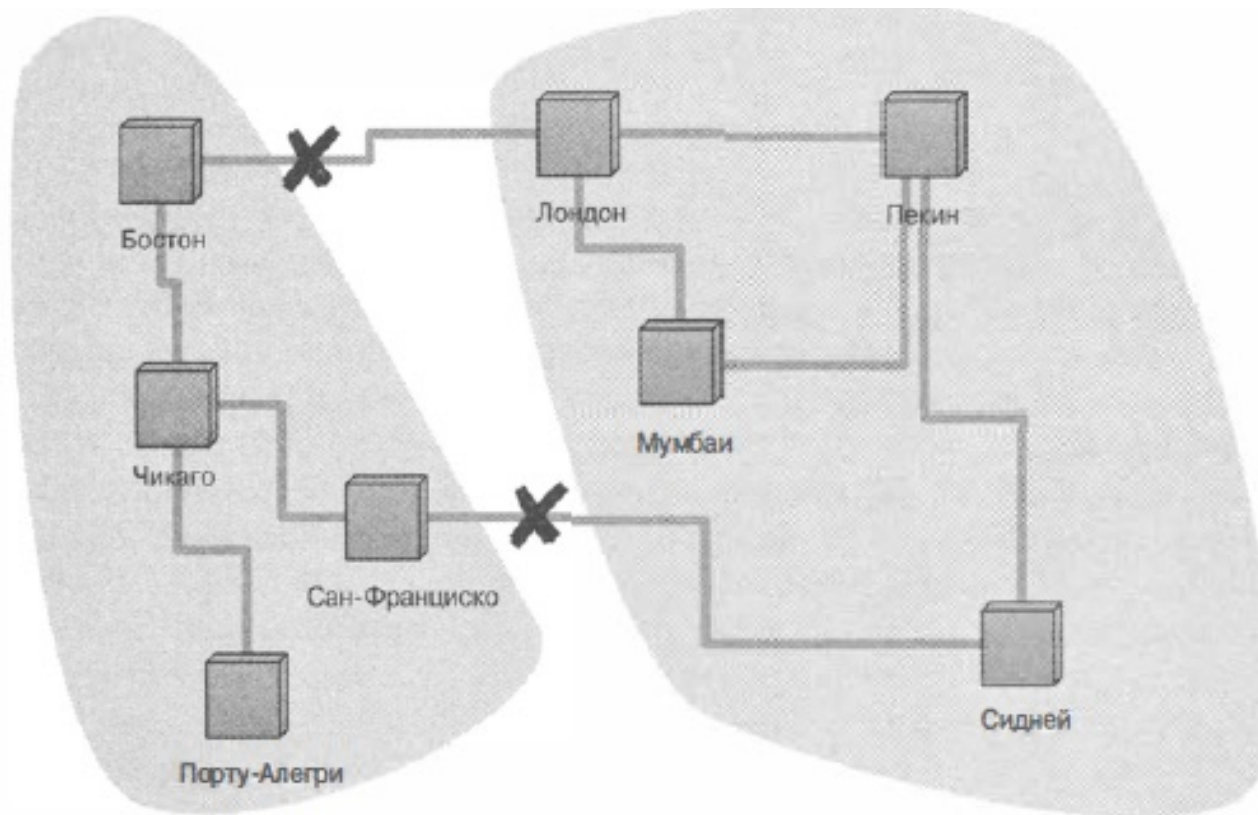
Согласованность репликаций

- Итоговая согласованность.
- Увеличивает окно несогласованности.
- Согласованность чтения-записи собственных записей
 - Привязка сессии (к серверу),
 - Штампы версий,
 - Чтение с ведущего.

Теорема CAP

- Из трех свойств – согласованности данных, доступности и устойчивости к разделению - можно обеспечить не больше двух
 - **Согласованность данных** - стандартна,
 - **Доступность** (availability) означает, что если вы можете обращаться к узлу кластера, то он может читать и записывать данные (на каждый запрос должен быть ответ).
 - **Устойчивость к разделению** (partition tolerance) означает, что кластер может восстанавливать обмен данными после обрыва связей в кластере, который разделен на многочисленные фрагменты, не способные взаимодействовать друг с другом.

Разрыв кластера



Следствия теоремы CAP

- СА – односерверная система.
- Поиск компромисов (для кластера).
- Компромис между согласованностью и доступностью.
- Конкретные решения
 - полная согласованность(в-в)/недоступность,
 - резерв, перезаказ/ доступность и конфликт.
- Вопрос цены конфликта (зависит от ПО).
- Компромис между согласованностью и временем ожидания.

Компромиссы

- Зависят от предметной области.
- Несогласованность записи
 - Заказы (слияние корзин).
- Несогласованность чтения
 - Торги на бирже/чтение новостей.
- Доступность (предельное время отклика).
- Ослабление долговечности
 - Данные сеанса/производительность сервера,
 - Ведущий-ведомый/переключение,
 - Долговечность репликации (копия до сбоя).

Кворумы

- Кворум записи
 - сколько узлов должны подтвердить запись
- $$W > N / 2$$
- R - количество узлов, с которых читают.
 - W - количество узлов, подтверждающих запись.
 - N - коэффициент репликации (количество реплик).

Кворум чтения

- со сколькими узлами следует установить контакт, чтобы гарантировать, что вы получаете самое последнее изменение
- Строгая согласованность:
$$R + W > N$$

Особенности кворумов

- Эффективное $N=3$ (кворум при сбое одного узла).
- Уровни согласованности для данных.
- Быстрое и согласованное чтение (но медленная запись)

$$N=3, W=3, R=1$$

- Быстрая запись

$$N=3, W=1, R=3$$

Документные базы данных на примере MongoDB

- Особенности:
 - хранение и обработка(чтение) документов,
 - самоописывающее иерархическое дерево в формате xml, json, bson и т.д.
- Примеры:
 - MongoDB, CouchDB, Terrastore, OrientDB, RavenDB.
- Применение:
 - Регистрация событий;
 - блоги/профили/состояния;
 - электронная коммерция;
 - веб-аналитика в реальном времени.
- Проблемы использования:
 - Сложные транзакции/ изменение структуры агрегата.

Модель данных

Реляционные БД		MongoDB
Экземпляр	→	Экземпляр
Схема	→	БД
Таблица	→	Коллекция
Строка	→	Документ
Row id	→	<u>_id</u> (специальная пометка в любой БД. Уникальна. Создается автоматически. Может присваиваться пользователем)
join	→	DBRef

Понятие документа

- Хранится в формате BSON, а отображают в JSON.
- Иерархия вложенных документов.
- Документ содержит:
 - скалярные записи;
 - ассоциативные массивы;
 - коллекции;
 - уникальный идентификатор (`_id`).
- Документы не обязательно одинаковы.
- Коллекции хранят похожие, но разные документы.
- Вместо NULL – отсутствие объекта.

Пример документа

```
{  
  "firstname": "Pramod",  
  "citiesvisited": [ "Chicago", "London", "Pune", "Bangalore" ],  
  "addresses": [  
    { "state": "AK",  
      "city": "DILLINGHAM",  
      "type": "R"  
    },  
    { "state": "MH",  
      "city": "PUNE",  
      "type": "R" }  
  ],  
  "lastcity": "Chicago"  
}
```

Типы данных

- **String** – это наиболее часто используемый тип данных для хранения данных. Строка в формате UTF-8.
- **Integer** – этот тип используется для хранения числового значения.
- **Boolean** – этот тип используется для хранения логического (true / false) значения.
- **Double** – этот тип используется для хранения значений с плавающей запятой.
- **Массивы(Arrays)** – этот тип используется для хранения массивов или списка.
- **Отметка времени(Timestamp)** – отметка времени.
- **Объект(Object)** – этот тип данных используется для встроенных документов.
- **Null** – этот тип используется для хранения значения Null.
- **Символ(Symbol)** – этот тип данных используется идентично строке; однако, это обычно зарезервировано для языков, которые используют определенный тип символов.
- **Дата(Date)** – этот тип данных используется для хранения текущей даты или времени в формате времени UNIX
- **Идентификатор объекта(ObjectId)** – для хранения идентификатора документа.
- **Двоичные данные(Binary data)** – для хранения двоичных данных.
- **Код(Code)** – этот тип данных используется для хранения кода JavaScript в документе.
- **Регулярное выражение (Regular Expression)** – этот тип данных используется для хранения регулярного выражения.

Основы работы с БД

Сервер→БД→Коллекция→документ

- **db.col.insert(doc)**
- **db.col.save(doc)**
 - Если есть `_id` => update, иначе insert
- **db.col.update(условие выбора, новый документ, параметры)**
 - может быть удаление/изменение поля, коллекции, элемента массива
- **db.col.remove(условие выбора ,[t / f])**
 - кого удалить: t – один элемент, f – все
- **db.col.drop()** – удалить коллекцию
- **db.dropdatabase()** – удалить БД
- `DBcollection col = db.getCollection(“название”)`

Функционал

- CRUD баз данных, коллекций и документов,
- Выборка данных (селекция, проекция, сортировка, ограничение),
- Работа с вложенными структурами и коллекциями,
- группировка и агрегирование,
- создание индексов и работа с ними
- организация ссылок и переход по ним,
- поиск текста.

Запросы - выборка

find (условие, проекция)

```
db.col.find() = select * from col
```

```
db.col.find({"pid":123}) = select * where pid = 123
```

`db.col.find({}, {fio:1,age:1})` – вернёт поля fio и age

`db.col.find({}, {fio: Ø})` - вернёт все поля кроме fio

```
db.col.find({age:20, adr.city:"Москва"},
            { age:true, _id:false})
```

Запросы - функции

```
find().sort({name:1}).skip(N).limit(M)
```

```
db.col.count()
```

```
db.col.find().count()
```

```
db.col.distinct("поле")
```

```
db.col.group(
```

```
  {   key: {name:true},
```

```
    initial: {total:Ø},
```

```
    reduce: function(items,prev) {prev.total+=1}
```

```
  })
```

Условия в запросах

db.col.find(условие, ...)

{age: {\$nl : 22} }

- age не равен 22

{age: {\$gt:100, \$lt:1000}}

- age больше 100 и меньше 1000

{num:{\$in:[1,2,3]}}

- значение из перечня

\$gt >

\$lt <

\$gtl ≥

\$ltl ≤

\$nl ≠

.find({\$or: [{name: "An"}, {age:20}]}) - или

.find({ adr:{\$exists:true}}) – ∃ поле adr

.find({len:{\$size:2}}) – два элемента в массиве len

.findOne(условие) - вернуть первый подходящий документ

Создание связи между документами

```
coach = ( {"fio": "Степанов В.В.", ...  
          "sport": "шахматы"})
```

```
db.coaches.save(coach)
```

```
group = ({ "title": "Группа по шахматам 1", ...  
          "coach": new DBRef('coaches', coach._id)})
```

```
db.groups.save(group)
```

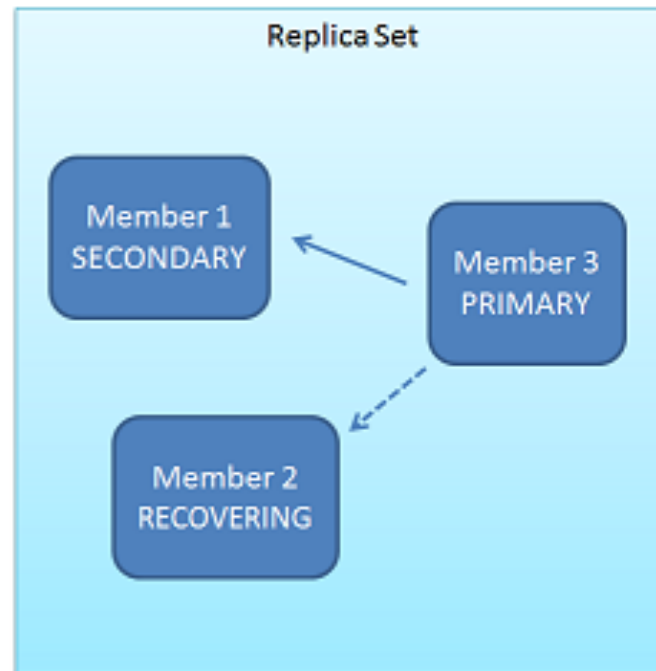
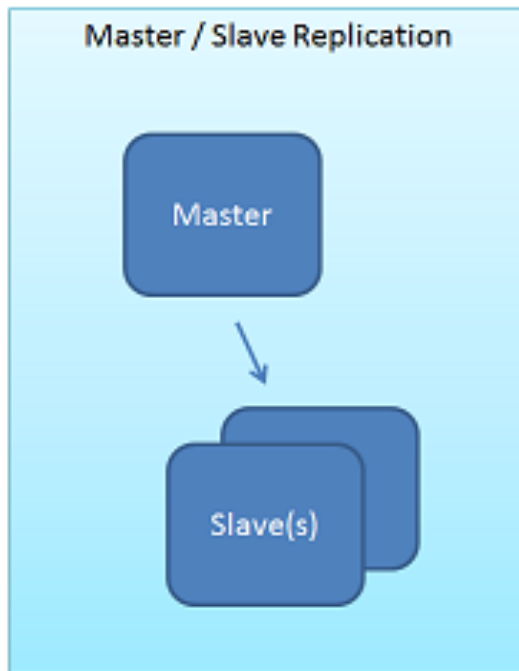

Переход по связи между документами

```
> group = ({ "title": "Группа по шахматам 1", "col": 10, "type": "Подростковая", "pupils": ["Ивушкин А.Г.", "Иванов И.Р." ], "coach": new DBRef('coaches', coach._id)})  
{  
  "title" : "Группа по шахматам 1",  
  "col" : 10,  
  "type" : "Подростковая",  
  "pupils" : [  
    "Ивушкин А.Г.",  
    "Иванов И.Р."  
  ],  
  "coach" : DBRef("coaches", ObjectId("5ea7368fb775bab737344d57"))  
}
```

- `db.coaches.find({ _id: db.groups.findOne({"title": "Группа по шахматам1" }).coach.$id }).pretty()`

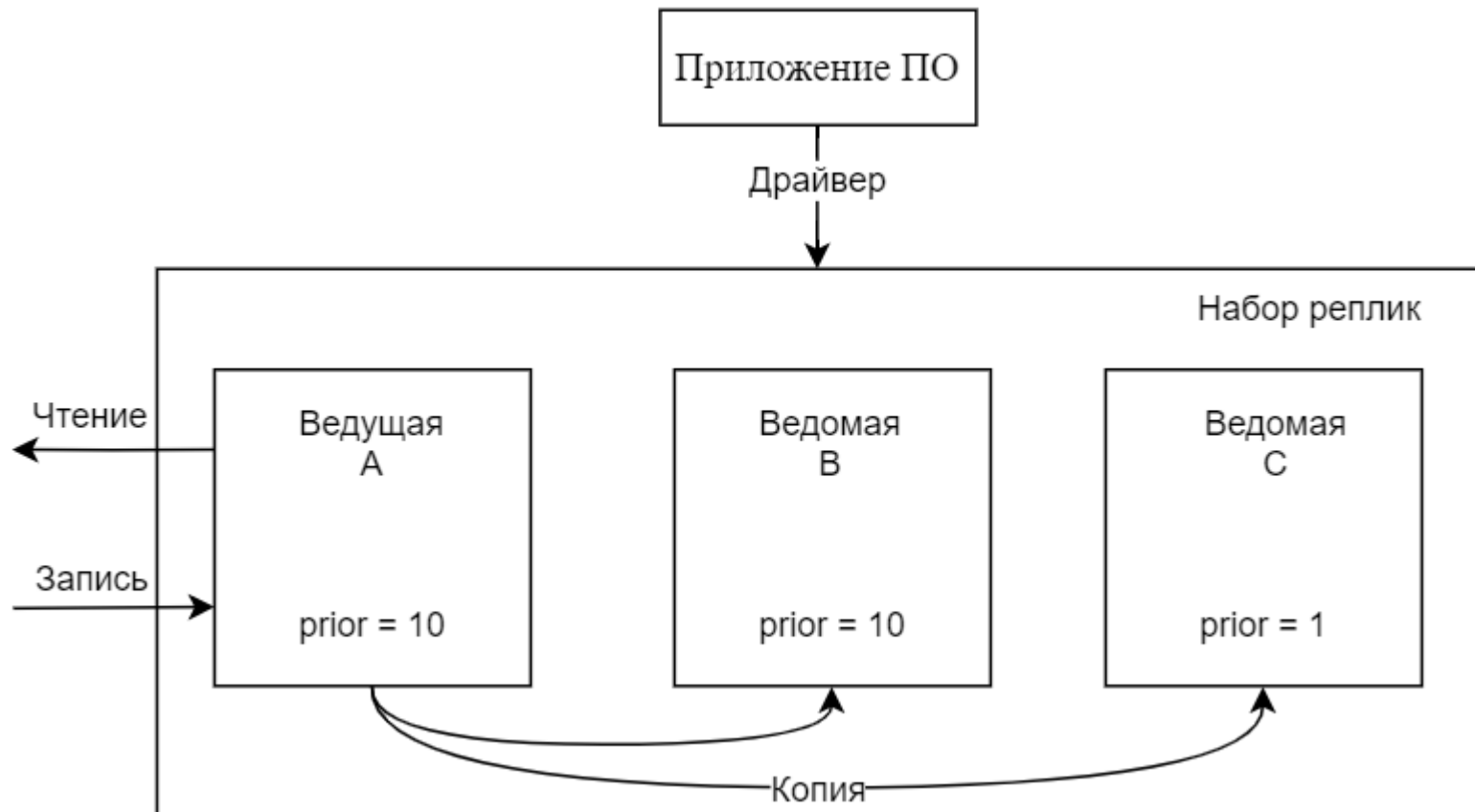
Модели репликации

- главный-подчиненный (Master-Slave) – ручное переключение,
- наборы реплик (Replica Set) – автоматическое переключение при сбое.



Доступность

- «асинхронная горизонтальная реплика»
master – slave Replication



Репликация

- Набор реплик (\geq двух штук);
- Создаётся соединение с одним сервером в любом кластере (прочие обнаруживаются автоматически);
- После сбоя другие реплики будут работать с новым ведущим (прозрачна для ПО);
- Ведущий выбирается автоматически (голосование), в зависимости от: объема ОП; расположения (близости) сервера; и т.д. (приоритет от пользователя 0-1000);
- Добавление узлов в кластер без его отключения;
- Если ведущий вышел из строя, то голосование и выбор нового ведущего;
- После восстановления узел работает как ведомый.

Согласованность данных

- Набор реплик (на всех/ N ведомых узлов)

- Запись: новые данные, количество реплик для успешной операции.

`db.RunCommand("getlasterror : 1, w : "majority"}`)

– запись на большинство реплик, устанавливается для любых записей БД.

- Увеличение согласованности приводит к уменьшению производительности.

Согласованность чтения

- `Mongo m = new Mongo("localhost:27017");
m.slaveOk();`
- **slaveOk** – разрешить чтение с ведомых узлов;
- может быть для БД/коллекции/операции;
- `col.find(запрос).slaveOk();`
 - для коллекции,
 - увеличивается производительность

Согласованность записи

- **WriteConcern** → запись на ведущий и N ведомых (увеличивается согласованность записей)
- для коллекций/ отдельных записей:
`db.col.setWriteConcern(REPLICAS_SAFE);`
или
`db.col.insert(obj, REPLICAS_SAFE)`
- Если = NONE, то низкий уровень безопасности

Транзакции

- Обычно для одного документа (агрегата).
- Некоторые БД могут поддерживать транзакции на N операций (insert, update, delete).
- По умолчанию все записи успешны (считаются).
- Контроль через WriteConcern:
 - None – понижение уровня,
 - REPLICAS_SAFE => поднятие уровня (ведущий + ≥ 1 ведомых),

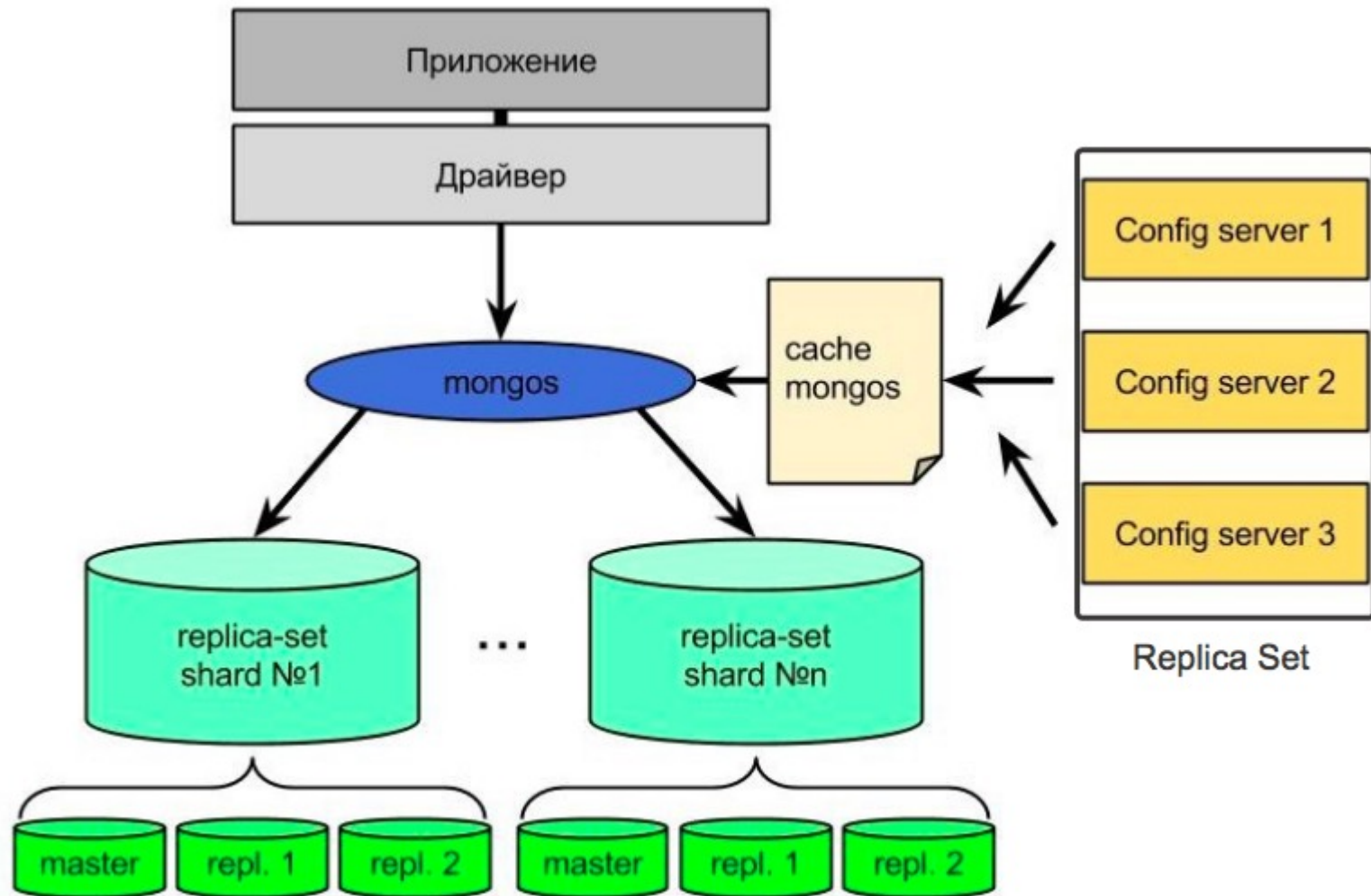
Масштабирование

- Добавление узлов или изменения хранимых данных без изменения приложения (при увеличении объема данных)
- горизонтальное масштабирование чтения:
 - при увеличении интенсивности чтения данных;
 - добавление новых ведомых узлов в набор реплик (без перегрузки существующих узлов)
- `rs.add("mongod:27017")`
 - новый узел синхронизируется с существующими, и может быть использован для чтения (через `slaveOk()`)

Масштабирование записи

- фрагментация по столбцам (полям):
 - масштабирование записи; (увеличивается производительность записи)
 - разбиение БД(кол.) по полю
- `db.runCommand({Shardcollection: "ecom.client",
key:{firstname:1}})`
- фрагменты перемещаются на другие узлы БД => балансировка нагрузки;
 - выделяются поля по территориальному расположению, использованию и т.д.

Настройка кластера с фрагментацией



Компоненты кластера

- Mongos – маршрутизатор запросов:
 - Кэширование данных, хранимых на config сервере.
 - Маршрутизация запросов от приложений на нужные фрагменты.
 - Запуск фонового процесса “Балансер”.
- Config server - хранилище метаданных
 - Данные о фрагментах,
 - Данные о наборах реплик.
- Replica-set shard – набор реплик с одним фрагментом (набор секций):
 - Ведущий сервер,
 - Набор ведомых.
- Балансер – сервис для миграции чанков из одного фрагмента в другой
 - «чанки» ([англ.](#) *chunks*) - блоки документа одинакового размера, выделенные на основе ключа фрагментации.

Настройка кластера

- Установка сервера БД
- Подготовка директорий
- Настройка набора реплик
- Соединение процессов
- Запуск фрагментации

Подготовка директорий

- Папка фрагмента «a», в которой будут папки «a0», «a1», «a2».
- Папка фрагмента «b», в которой будут папки «b0» «b1» «b2».
- Папка для работы самой БД — db, в которой будут папки cfg0, cfg1, cfg2, соответствующие реплика-сету конфигурационного сервера.

\$ sudo chown parallels * -R /data

- рекурсивно изменить владельца всех файлов на пользователя parallels в папке data и всех вложенных в неё

Настройка набора реплик

- Запуск mongod для конфиг-серверов (порты 26050, 26051, 26052)

```
$ mongod --configsvr --dbpath /data/db/cfg0 --port 26050 --logpath  
/data/db/log.cfg0 --fork --replSet c
```

...

- Запуск mongod для фрагмента А (порты 27000, 27001, 27002)

```
$mongod --shardsvr --replSet a --dbpath /data/a/a1 --logpath  
/data/a/log.a1 --port 27001 --fork --logappend --smallfiles --oplogsize  
50
```

.....

- Запуск mongod для фрагмента В (порты 27003, 27004, 27005)

```
$mongod --shardsvr --replSet b --dbpath /data/b/b0 --logpath  
/data/a/log.b0 --port 27003 --fork --logappend --smallfiles --oplogsize  
50
```

.....

Параметры запуска mongod

- **--dbpath** Путь к файлам этого сервера
- **--logpath** Путь к файлу логов этого сервера
- **--port** Указание порта, через который он будет слушать соединения
- **--fork** Флаг, означающий что mongod должен быть поднят в фоновом режиме
- **--configsvr** Флаг, означающий что данный сервер исполняет роль конфигурационного
- **--shardsvr** Флаг, означающий что данный сервер исполняет роль элемента реплики
- **--replSet** Указание к какому набору копий принадлежит сервер
- **--smallfiles** Флаг упрощения логов, для сохранения места на диске
- **--oplog** Размер служебной коллекции oplog используемой при репликации данных

Соединение серверов

- Информирование mongos о config серверах

```
$ mongos --configdb  
localhost:26050,localhost:26051,localhost:26052 --fork --  
logappend --logpath /data/db/log.mongos --port 26060
```

- Создание набора реплик для фрагмента А

```
$ mongo --port 27000  
>rs.initiate()  
>rs.add("127.0.0.1:27001")  
>rs.add("127.0.0.1:27002")
```

- Создание набора реплик для фрагмента В ...

Запуск фрагментации

- Подключение к mongos (порт 26060)

```
$mongo --port 26060
```

- Добавили фрагменты

```
>sh.addshard("a/localhost:27000")
```

```
>sh.addshard("b/localhost:27003")
```

- Выключаем балансир нагрузки

```
>sh.stopBalancer()
```

- Создаем БД «shardTestDB»

```
>use shardTestDB
```

```
>sh.enableSharding("shardTestDB")
```

Настройка фрагментов

- Создаем коллекцию users, в которой ключом для фрагментации станут поля username

```
>sh.shardCollection("shardTestDB.users", {"username": 1})
```

- Перейдем в служебную БД config. Изменим размер чанка с 64 Мб до 1 Мб

```
>use config
```

```
>db.settings.find({"_id": "chunksize"})
```

```
>db.settings.save({"_id": "chunksize", value: 1})
```

- Вернемся в исходную БД и добавим записи в БД
- Запустим балансир

```
>sh.startBalancer()
```

Резултат фрагментации

databases:

```
{ "_id" : "admin", "partitioned" : false, "primary" : "config" }  
{ "_id" : "test", "partitioned" : false, "primary" : "a" }  
{ "_id" : "shardTestDB", "partitioned" : true, "primary" : "a" }
```

shardTestDB.users

shard key: { "username" : 1 }

chunks:

	b	7
a		7

```
{ "username" : { "$minKey" : 1 } } --> { "username" : "user0" } on : b Timestamp(2, 0)  
{ "username" : "user0" } --> { "username" : "user2516" } on : b Timestamp(3, 0)  
{ "username" : "user2516" } --> { "username" : "user31805" } on : b Timestamp(4, 0)  
{ "username" : "user31805" } --> { "username" : "user38451" } on : b Timestamp(5, 0)  
{ "username" : "user38451" } --> { "username" : "user45097" } on : b Timestamp(6, 0)  
{ "username" : "user45097" } --> { "username" : "user51742" } on : b Timestamp(7, 0)  
{ "username" : "user51742" } --> { "username" : "user58389" } on : b Timestamp(8, 0)  
{ "username" : "user58389" } --> { "username" : "user65033" } on : a Timestamp(8, 1)  
{ "username" : "user65033" } --> { "username" : "user7168" } on : a Timestamp(1, 19)  
{ "username" : "user7168" } --> { "username" : "user78325" } on : a Timestamp(1, 21)  
{ "username" : "user78325" } --> { "username" : "user84971" } on : a Timestamp(1, 23)  
{ "username" : "user84971" } --> { "username" : "user91616" } on : a Timestamp(1, 25)  
{ "username" : "user91616" } --> { "username" : "user999" } on : a Timestamp(1, 26)  
{ "username" : "user999" } --> { "username" : { "$maxKey" : 1 } } on : a Timestamp(1, 4)
```

mongos> █

Колоночные БД (семейство столбцов) на примере Cassandra

- Cassandra, HBase, Amazon SimpleDB, Amazon DynamoDB, HyperTable
- Применение:
 - регистрация событий
 - управление информацией блогов
 - счетчики посещений
 - сроки действия (реклама, новости)
- Не используется:
 - Транзакции чтения/записи
 - агрегация (только на клиенте)
 - для раннего проектирования

Модель данных

- Сервер - кластер
- БД - пространство ключей
- Таблица - семейство столбцов
- Столбец (поле) – столбец (поле)

строка: id || имя:значение

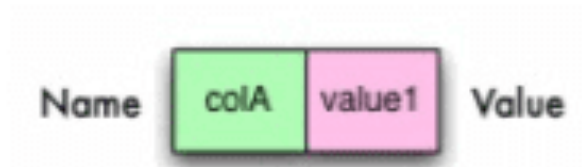
базовый элемент в Cassandra (ключ-значение)

```
{  
    name: " ",  
    value: " ",  
    timestamp: 123456  
}
```

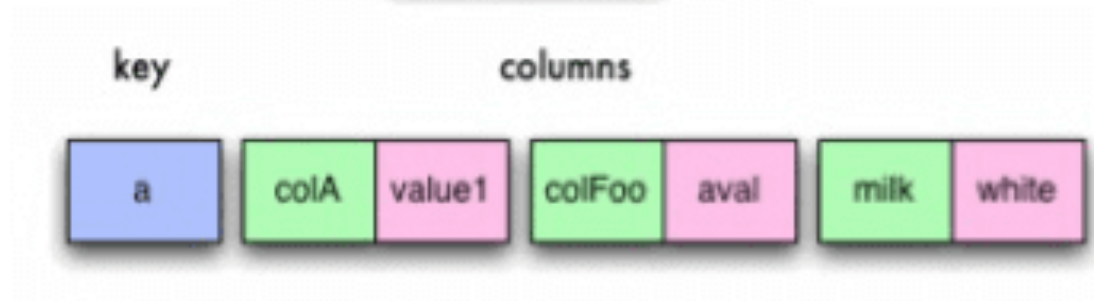
время жизни (метка времени конфликт/обновление/удаление)

Схема модели данных

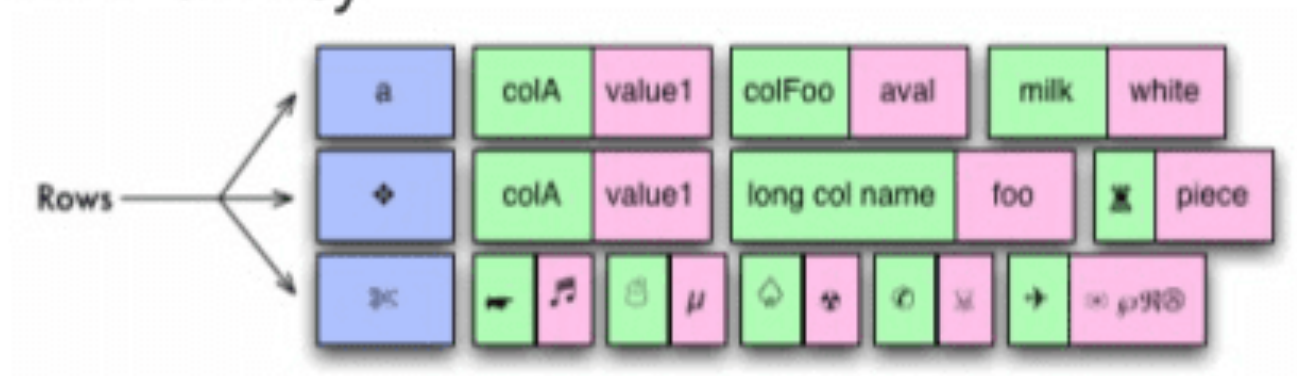
Ячейка



Строка



Column Family



Семейство столбцов

```
{  
  {  
    "id": {  
      столбец_1: «значение 1 »,  
      столбец_2: «значение 2 »,  
      ...      }  
    }  
    {  
      "id2": {  
        столбец_1: «значение 1 »,  
        столбец_3: «значение 3 »,  
        ...      }  
    }  
  }  
}
```

- различные наборы столбцов для разных строк
- могут быть добавлены в процессе
- столбец добавляется/удаляется в любой момент времени

Супер-столбец

- ассоциативный массив

```
{  
  name: "bookA",  
  value: {  
    author: "автор",  
    izd: "название",  
    isbn: 12345,  
  }  
}
```

Типы данных в СУБД Cassandra

- **ByteType**: любые байтовые строки (без валидации);
- **AsciiType**: ASCII строка;
- **UTF8Type**: UTF-8 строка;
- **IntegerType**: число с произвольным размером;
- **Int32Type**: 4-байтовое число;
- **LongType**: 8-байтовое число;
- **UUIDType**: UUID 1-ого или 4-ого типа;
- **TimeUUIDType**: UUID 1-ого типа;
- **DateType**: 8-байтовое значение метки времени;
- **BooleanType**: два значения: true = 1 или false = 0;
- **FloatType**: 4-байтовое число с плавающей запятой;
- **DoubleType**: 8-байтовое число с плавающей запятой;
- **DecimalType**: число с произвольным размером и плавающей запятой;
- **CounterColumnType**: 8-байтовый счётчик

Пространство ключей

- Пространство ключей (аналог БД), содержит семейства столбцов

```
CREATE KEYSPACE [IF NOT EXISTS]
```

```
    keyspace_name
```

```
WITH REPLICATION = {
```

```
'class' : 'SimpleStrategy', 'replication_factor' : N }
```

```
| 'class' : 'NetworkTopologyStrategy',
```

```
    'dc1_name' : N [, ...]
```

```
};
```

Работа через клиента БД

- **set** Pers['id']['fio'] = 'значение';
set Pers['id']['fio'] **with** ttl
(ttl – время жизни - после N сек удаляется)
- **get** Pers['id'];
get Pers['id']['fio'];
get Pers **where** fio = '..';
- **del** Pers['id']['fio'];
del Pers['id'];

Язык CQL – Cassandra Query Language

- **create columnfamily** Pers
(
 key varchar **primary key**,
 fio varcahr,
 ...
);

insert into Pers(**key**, fio, ...) **values** ('..', '..', ...);

select * from Pers **where** fio = "
 (после create index)

- **update** Pers **set** age = age + 1 **where** key = '..';
- **delete** Pers **where** key = '..';

Поиск

- По умолчанию поиск только по ключу строки,
- или надо явно указать индекс, тогда возможен поиск по значениям индексам,
- битовые индексы, которые эффективны при небольшом значении (количестве) столбцов

Распределительный и кластерный ключи

```
CREATE TABLE numberOfRequests  
( cluster text,  
  date text,  
  time text,  
  numberOfRequests int,  
  PRIMARY KEY ((cluster, date), time)  
)  
WITH default_time_to_live = 120 and  
  CLUSTERING ORDER BY (time DESC)
```

- Распределительный (cluster, date) – узел хранения
- Кластерный (time) - внутри узла

Поисковые запросы (секция WHERE)

- Столбцы **распределительного** ключа (указывать все) поддерживают только два оператора: = и IN

```
SELECT * FROM numberOfRequests WHERE cluster IN ('cluster1', 'cluster2') AND date = '2015-05-06';
```
- **Кластерные** ключи поддерживают операторы =, IN, >, > =, <=, <, CONTAINS и CONTAINS KEY
- Выборки по **индексным** полям
- Выборка по прочим полям – полное чтение данных

```
SELECT * FROM student  
WHERE last_name='Ivanov' ALLOW FILTERING;
```


Возможности CQL

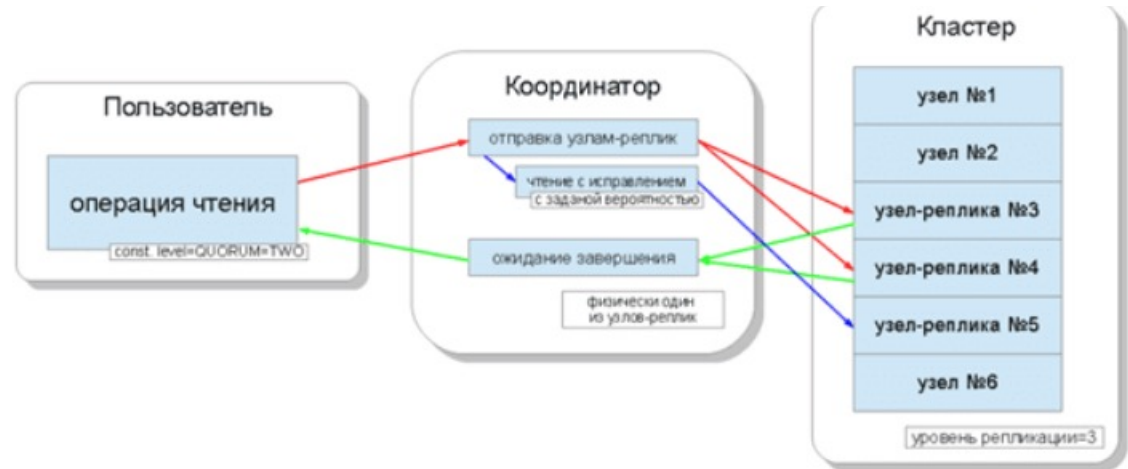
- Агрегация, группировка и сортировка
SELECT title, MAX(price)
FROM course GROUP BY title;
- Материализованные представления
- Пакетная обработка
- Время жизни - TTL
- Конструкция if
UPDATE course SET description='conditional'
WHERE title='test6' AND price=2000 **IF EXISTS**;

Репликация

- Одноранговая репликация.
- Параметры репликации:
 - $N = 3, R = 2, W = 2$ - если 1 узел сбой, то запись получит от другого.
 - $N = 3, R = 1, W = 2$ - если 2 узла сбой, то нет записи, но есть чтение.
 - $N = 3, R = 2, W = 1$ - если 2 узла сбой, то есть запись, но нет чтение.
 - $(R + W) > N \Rightarrow$ выбор согласованности/доступности.

Схема согласования

- Схема чтения



- Схема записи



Уровни согласованности чтения

- **ONE** — координатор шлёт запросы к ближайшему узлу-реплике, читая остальные с целью исправления (read repair) с заранее заданной в конфигурации вероятностью;
- **TWO** — координатор шлёт запросы к двум ближайшим узлам, выбирая значение с большей меткой времени;
- **THREE** — координатор шлёт запросы к трем ближайшим узлам, выбирая значение с большей меткой времени;
- **QUORUM** — собирается кворум, то есть координатор шлёт запросы к более чем половине узлов-реplik, а именно $\text{round}(N/2)+1$, где N — уровень репликации;
- **ALL** — координатор возвращает данные после прочтения со всех узлов-реplik;
- **LOCAL_QUORUM** — собирается кворум узлов в том датацентре, где происходит координация, и возвращаются данные с последней меткой времени;
- **EACH_QUORUM** — координатор возвращает данные после собрания кворума в каждом из датацентров.

Уровни согласованности записи

- **ANY** — даёт возможность записать данные, даже если все узлы-реплики не отвечают. Координатор дожидается первого ответа от любого одного узла-реплик или данные сохраняются с помощью механизма направленной отправки (hinted handoff) на координаторе.
- **ONE** — координатор шлёт запросы всем узлам-реплик, но возвращает управление пользователю, дождавшись подтверждения от любого первого узла;
- **TWO** — координатор дожидается подтверждения от двух первых узлов, прежде чем вернуть управление;
- **THREE** — координатор ждет подтверждения от трех первых узлов, прежде чем вернуть управление;
- **QUORUM** — координатор дожидается подтверждения записи от более чем половины узлов-реплик, а именно $\text{round}(N/2)+1$, где N — уровень репликации;
- **ALL** — координатор дожидается подтверждения от всех узлов-реплик;
- **LOCAL_QUORUM** — координатор дожидается подтверждения от более чем половины узлов-реплик в том же центре обработки данных, где расположен координатор. Это позволяет избавиться от задержек, связанных с пересылкой данных в другие датацентры.
- **EACH_QUORUM** — координатор дожидается подтверждения от более чем половины узлов-реплик в каждом центре обработки данных.

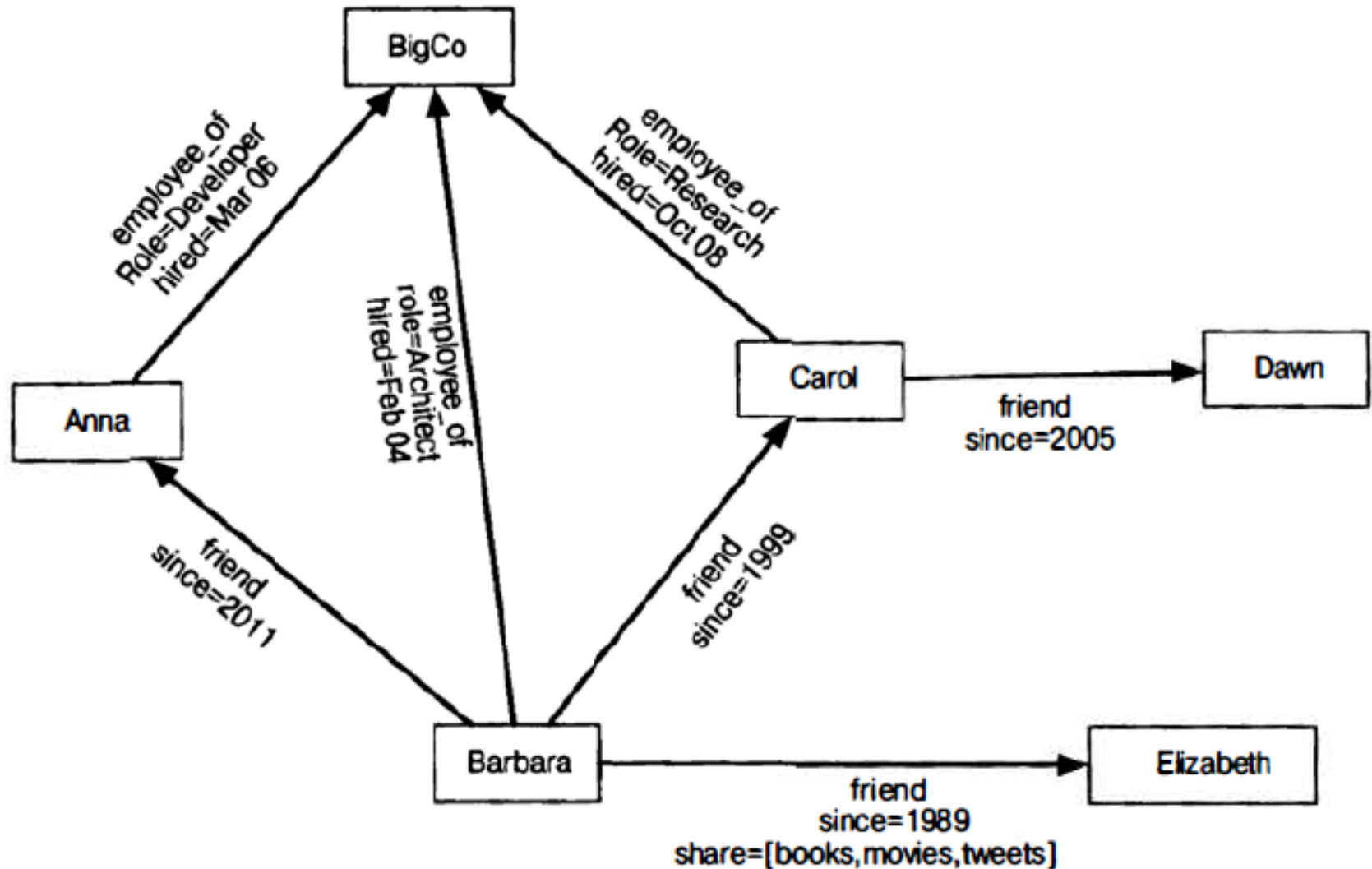
Уровни согласованности сравнение

- **One**
 - Чтение – быстро/устаревание
 - Запись – быстро/потеря данных
- **Quorum**
 - Чтение – новое время/чтение с исправлением
 - Запись – подтверждение записи
- **ALL**
 - При отказе блокировка
- По умолчанию для всех операций **ONE**
- Задается через параметр **CONSISTENCY**

Графовые БД на примере Neo4j

- Neo4j, OrientDB, InfiniteGraph, FlockDB
- Применение
 - социальные сети, проекты;
 - доставка, геолокация (адрес = узел, поиск пути, интересные места, связь с сотовой сетью);
 - справочные системы (шаблоны в отношении, аналогии) .
- Не подходит
 - не для частых изменений данных.

Тип и свойства отношений



Элементы графовой модели

- Узел:
 - Id
 - Метка (до 20 разных)
 - Свойство (ключ/значение)
- Отношение (связь)
 - Бинарная (не висящие)
 - Направление
 - Метки
 - Свойства

Типы данных свойств

- Число (Integer и Float)
- Строки - String
- Логические - Boolean
- Пространственный тип — Point (точка)
- Временные типы: Date, Time, LocalTime, DateTime, LocalDateTime и Duration
- Структурные типы: узлы, отношения, пути
- Составные типы:
 - Lists (коллекции, каждый элемент которых имеет какой-то определенный тип)
 - Maps (коллекции вида (key: value), где key имеет тип String, а value любого типа).

Взаимодействие с Neo4j

- Простой язык запросов (аналог SQL)
- Пользовательский интерфейс для выполнения команд: Neo4j Data Browser
- **REST API** (из Java, Spring, Scala и т.д.)
- Два **Java API**: Cypher API и Native Java API для разработки приложений Java.
- Экспорт запрошенных данных в форматы **JSON** и **XLS**.
- Собственная графическая библиотеку и локальный GPE (движок обработки графики).

Возможности

- Создание и удаление узлов и связей.
- CRUD меток и свойств.
- Ограничения (уникальность и not null).
- Индексация.
- Обход графа.
- Выборка узлов, связей, их меток и свойств.
- Агрегация и сортировка.
- Встроенные функции.
- Поддержка транзакций.

Создание узлов и связи

```
Node m = db.createNode();  
m.setProperty("name", "Вася");
```

```
Node p = db.createNode();  
p.setProperty("name", "Петя");
```

```
m.createRelationshipTo(p, FRIEND);  
p.createRelationshipTo(m, FRIEND);
```

Транзакции

- Транзакции ACID
- все изменения в транзакции (иначе исключение)
- чтение вне транзакции

```
Transaction t = database.beginTx();  
    try {  
        создание/изменение узла  
        t.success();                -- commit  
    }  
    finally {  
        t.finish();                -- завершение(обязательно) commit  
                                   -- или откат (если не было success)  
    }
```

Работа с графом

- Язык **Gremlin** - обход графа (для любой графовой БД)
- neo4j - язык **Cypher**
 - обход графа,
 - получение свойств узлов,
 - язык запросов (привязок),
 - индексы,
 - транзакции.

Индексы

- Задают на основе свойств узлов и свойств ребер
- Для задания стартовой точки поиска
- Создают в транзакции

```
Index <Node> ni =  
    graphDB.index().forNodes("nodes");  
  
ni.add(m, "name", m.getProperty("name"));
```


Обход графа

```
Traverser ft = node.traverse(  
    Order.BREATH_FIRST,  
    StopEvaluator.END_OF_GRAPH,  
    ReturnableEvaluator.ALL_BUT_START_NODE,  
    EdgeType.FRIEND,  
    Direction.OUTCOMING,  
);
```

- Узлы, связанные с начальным через FRIEND любого уровня вложенности

Получение путей и их параметров

- все пути между двумя узлами,
- длина пути (количество дуг между узлами),
- Кратчайший путь.

Шаблон поискового запроса

START нач.узел = определить по id, списку id,
по индексу

MATCH связи, шаблон (шаблон связей)

WHERE условие на значения узлов и связей

RETURN узлы/связи/свойства

ORDER BY из результата

SKIP узлы, которые пропускаем

LIMIT ограничение на результат

Пример на языке Cypher

```
start b = node:nodeIndex(name = "Иван")  
match (b) -- (c)  
return c
```

- *вернуть узлы, связанные с b*
- <-- входящие
- --> исходящие

Пример на языке Cypher 2

start ...

match (b) - [:FRIEND] ->(c)

return c.name, c.location

start ...

match p = (b) - [:FRIEND * 1..3] -> (c)

return lenght(p), b.name, c.name

1..3 - глубина, по умолчанию = 1

Пример на языке Cypher 3

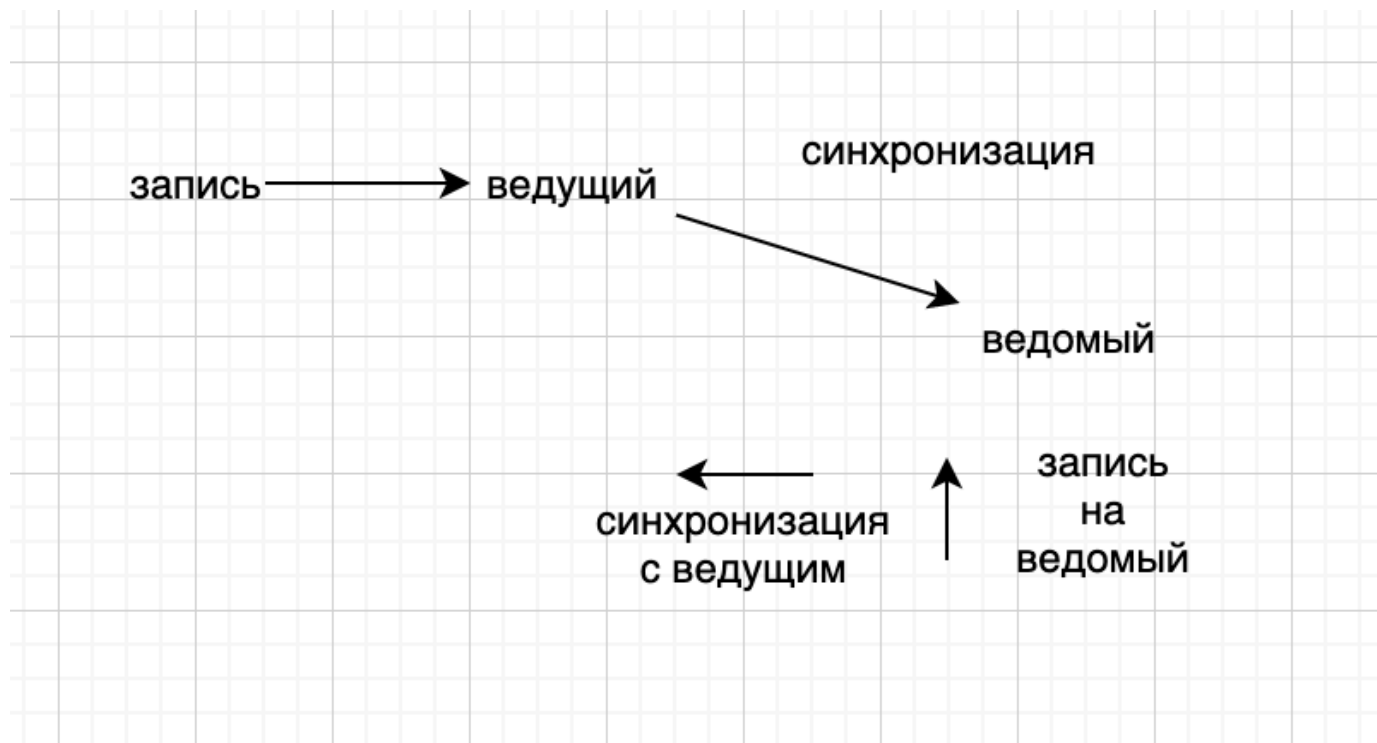
```
match (b:dog {name:"Rex"}) - [rel] -> (n)  
where type(rel) = 'FRIEND' AND  
      rel.since<2021  
return rel.since, n.name
```

Репликация

- Автоматический ведущий (1-й из кластера)
 - Apache ZooKeeper
 - При отказе ведущего кластер автоматически выбирает нового ведущего
-
1. Запуск сервера СУБД
 2. Связь с Apache ZooKeeper
 3. Узнать ведущего (или стать им)

Синхронизация в кластере

1. Без репликации - на 1 сервере (обычно).
2. Если на кластере , то



Спасибо за внимание!

Виноградова Мария Валерьевна

Vinogradova.m@bmstu.ru

МГТУ им. Н.Э. Баумана
кафедра Систем обработки информации и
управления (ИУ5)