ТЕХНОСФЕРА

Методы распределенной обработки больших объемов данных в Hadoop

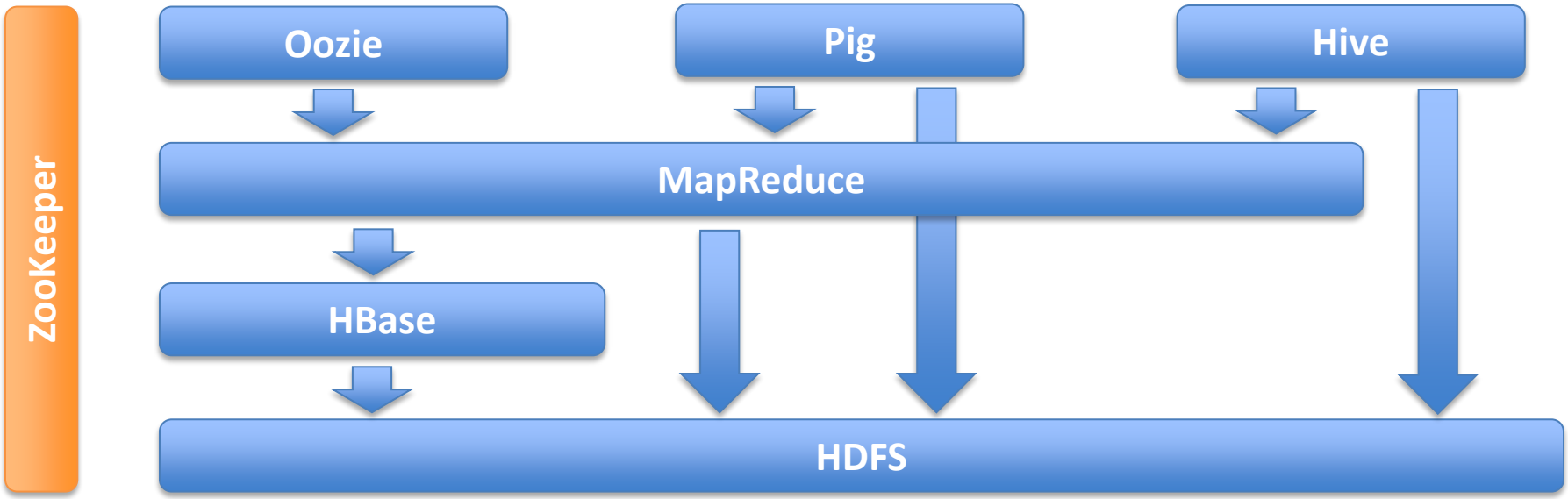
Лекция 9: ZooKeeper

Москва 2014

Zookeeper

- *Сервис для координации процессов распределенных приложений*
- Разработан в Yahoo! Research
- Проект Apache
  - *http://zookeeper.apache.org/*
- Используется для координации приложений в Hadoop

Zookeeper в экосистеме Hadoop



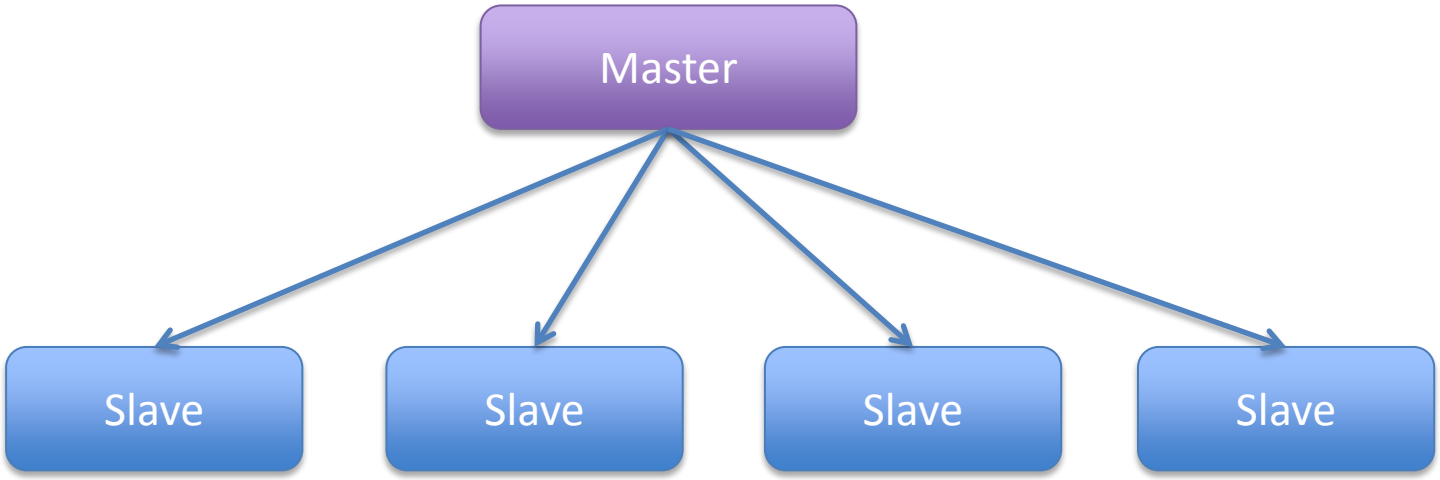
Неправда о распределенных вычислениях

- Сеть надежна
- Не существует *latency*
- Топология не изменяется
- Сеть гомогенна
- Пропускная способность сети безгранична
- ….

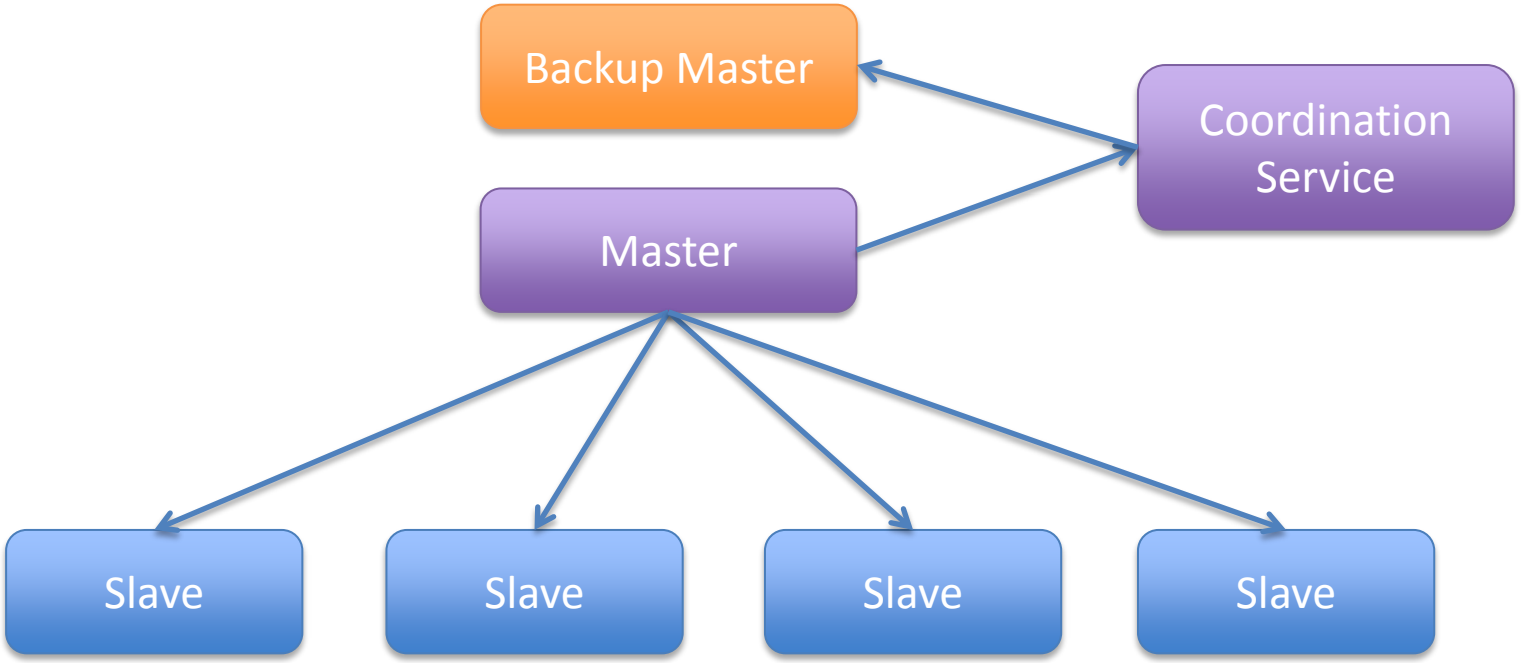
Мотивация

- Раньше: **одна** программа запускалась на **одном** компьютере с **одним** процессором
- Сейчас: приложения состоят из **независимых** программ, запущенных на постоянно **изменяющемся множестве** компьютеров
- Сложность: координация всех этих независимых программ
- Разработчик должен уметь реализовать логику координации и логику приложения
- ZooKeeper предназначен для того, что облегчить разработчику жизнь и оградить его от написания логики координациии

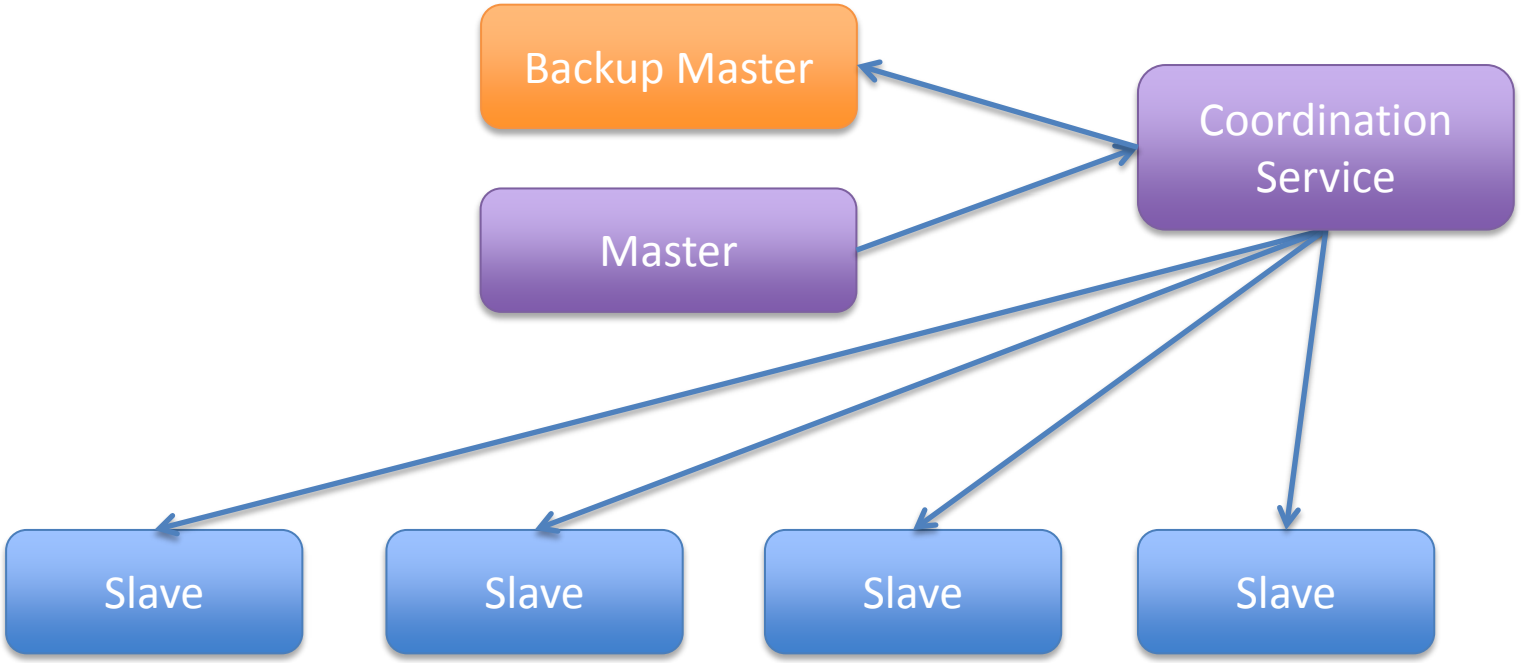
Как может выглядеть стандартная распределенная система?



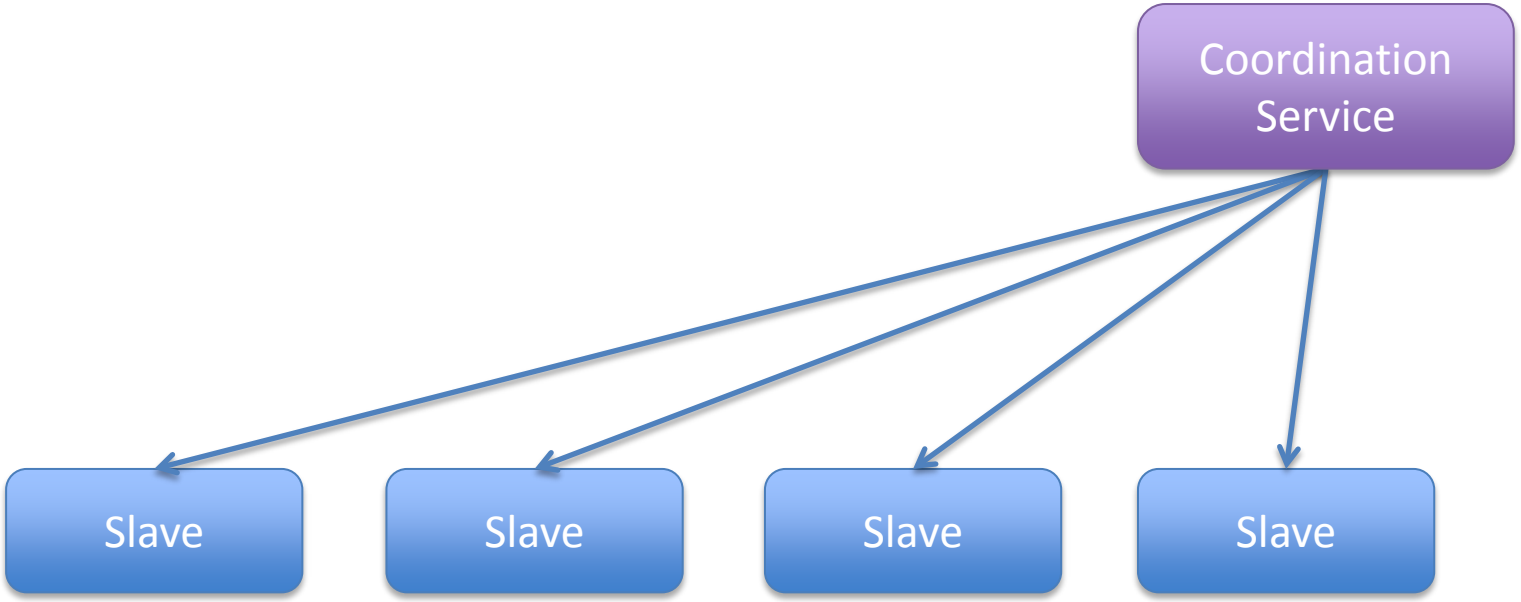
Как может выглядеть стандартная распределенная система?



Как может выглядеть стандартная распределенная система?



Как может выглядеть стандартная распределенная система?



Почему координация распределенных систем сложна?

- **Partial failures** делает написание приложение сложным
- Если сообщение отправляется по сети и в сети происходит fail, то отправитель не узнает:
  - Дошло ли его сообщение
  - Не упал ли процесс на стороне приемника
  - Правильно ли обработалось отправленное сообщение
- Единственный вариант для отправителя это переконнектиться и спросить его об этом
- ZooKeeper предлагает набор тулзов для построения распределенных приложений, которые могут успешно обрабатывать *partial failure*

Многопоточные программы на одной машине

- Многопоточные программы сталкиваются с похожими проблемами
- Важное отличие: разные машины в кластере не объединяет ничего кроме сети (архитектура shared-nothing)

Типичные проблемы координации в распределенных системах

- **Static configuration**: набор параметров для системных процессов
- **Dynamic configuration** : параметры, которые изменяются *on the fly*
- **Group membership**: кто жив (или мертв)?
- **Leader election**: кто ответственный
- **Mutually exclusive access** к критичным ресурсам (locks)
- ZooKeeper позволяет это довольно просто реализовать

ZooKeeper’s design principles

- API wait-free
  - Нет блокирующих примитивов
  - Блокировка может быть реализована на клиенте
  - Нет deadlocks
- Клиентские запросы обрабатываются в порядке FIFO
- Клиенты получают нотификацию об изменениях
  - Всегда, прежде чем клиент увидит измененные данные

Стратегия ZooKeeper: быть быстрым и надежным

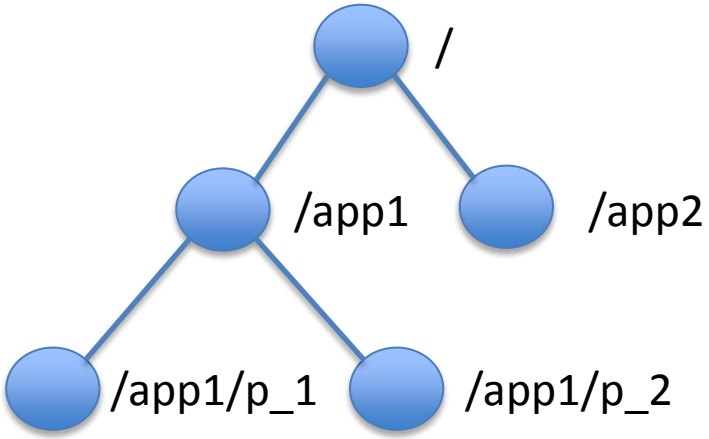
- Сервис ZooKeeper это несколько серверов, использующих репликацию (**high availability**)
- Данные **кешируются** на стороне клиента
  - Пример: клиент кеширует ID текущего лидера вместо того, чтобы каждый раз его запрашивать в ZooKeeper
- Что случится, если будет выбран новый лидер?
  - Одно из решений: *polling* (не оптимально)
  - Другой вариант: *watch mechanism*
    - Клиенты могут наблюдать обновление данного объекта с данными

Терминология ZooKeeper

- **Client:** пользователь сервиса ZooKeeper
- **Server:** процесс, предоставляемый сервисом ZooKeeper
- **znode:** in-memory data node в ZooKeeper, организованные в hierarchical namespace (**data tree)**
- **update/write:** любые операции, которые изменяют состояние data tree
- Клиент устанавливает **session** когда соединяется с ZooKeeper

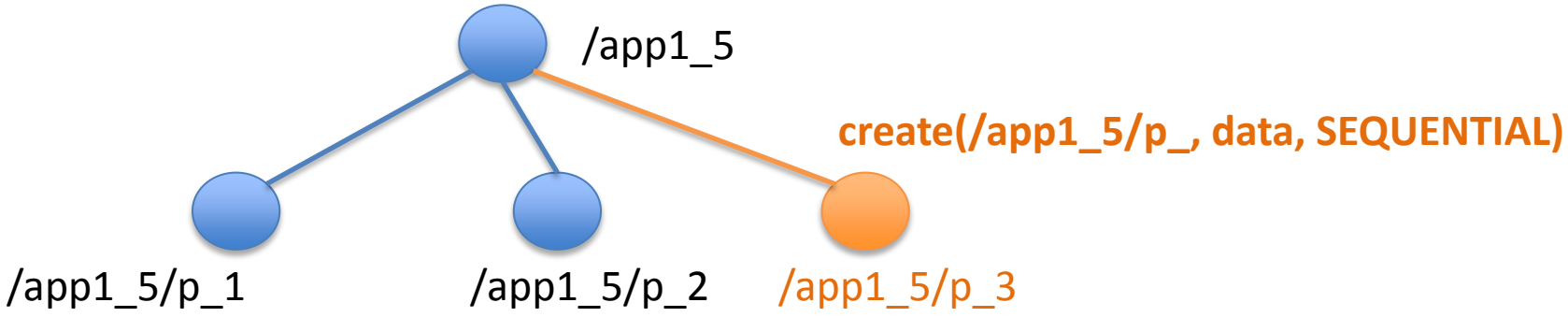
Модель данных ZooKeeper: filesystem

- Znodes организованы в **hierarchical namespace**
- Ими можно управлять из клиента через **ZooKeeper API**
- **Znodes** похожи на UNIX-style пути в файловой системе
  - Все znodes хранят данные (как файлы) и могут иметь детей (как директории)



Флаги znode

- Клиенты управляет znodes путем добавления и удаления их
- **EPHEMERAL flag:** клиент создает znode, которая будет удалена в конце клиентской сессии
- **SEQUENTIAL flag:** монотонно увеличивает счетчик добавляемый к путю znode
  - Значение счетчика новой znode под одним родителем всегда больше, чем у уже существующих детей



znodes & watch flag

- Клиент может выполнять операции чтения к znode с *watch flag*
- **Сервер уведомляет** клиента, когда данные на ноде изменились
- Наблюдатели – это **one-time** триггеры, ассоциированные с сессией
- **Уведомления** только информируют об изменениях, а не о том, что это новые данные
- Znodes не предназначены для general data storage
  - Порядок данных определяется килобайтами

Сессии

- Клиент соединяется с ZooKeeper и инициирует сессию
  - Сессии позволяют клиенту прозрачно переходить от сервера к серверу
  - Любой сервер может обрабатывать клиентские запросы
- Сессия имеет соответствующий таймаут
  - ZooKeeper считает клиент невалидным, если за время таймаута от него ничего не пришло
  - Сессия завершается либо из-за невалидного клиента, либо явно закрывается клиентом

Client API

- ***create(znode, data, flags)***
  - Флаги определяют типа znode
    - *REGULAR, EPHEMERAL, SEQUENTIAL*
  - Znode должна быть адресована все время по полному пути
    - */app1/foo/bar*
  - Возвращает путь к znode
- ***delete(znode, version)***
  - Удаляет znode если версия эквивалентна актуальной версии znode
  - Чтобы исключить проверку версии нужно установить *version=-1*

Client API

- ***exists(znode, watch)***
  - Возвращает *true* если znode существует, иначе *false*
  - Флаг *watch* позволяет установить наблюдения за znode
  - watch позволяет подписать на информацию об изменении znode
  - Можно установить флаг даже на несуществующую znode и клиент будет проинформирован, когда нода появится
- ***getData(znode, watch)***
  - Возвращает данные, записанные в znode
  - *watch* не устанавливается, если нода не существует

Client API

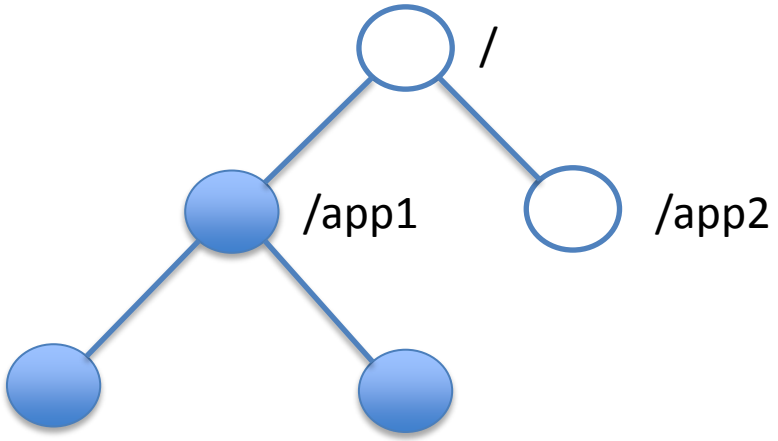
- ***setData(znode, data, version)***
  - Перезаписывает данные в znode если версия соответствует указанной
  - *version=-1* позволяет форсированно записать данные
- ***getChildren(znode, watch)***
  - Возвращает множество детей (znodes) указанной znode
- ***sync()***
  - Ждет завершения всех обновлений отправленных на сервер ZooKeeper

Client API operation

- Синхронные вызовы
  - Клиент блокируется после вызова операции и ждет, пока операция не завершится
  - Нельзя делать конкурентные запросы из одного клиента
- Асинхронные вызовы
  - Возможны конкурентные запросы
  - Клиент может обрабатывать множество запросов
- Операции
  - *update/write*
    - create, setData, sync, delete
  - *read*
    - exists, getData, getChildren

Примитивы: configuration

1. Как **новый** воркер запрашивает конфигурацию у ZK?
2. Как администратор **изменяет** конфигурацию “**на лету**”?
3. Как воркеры получают **новую** конфигурацию?

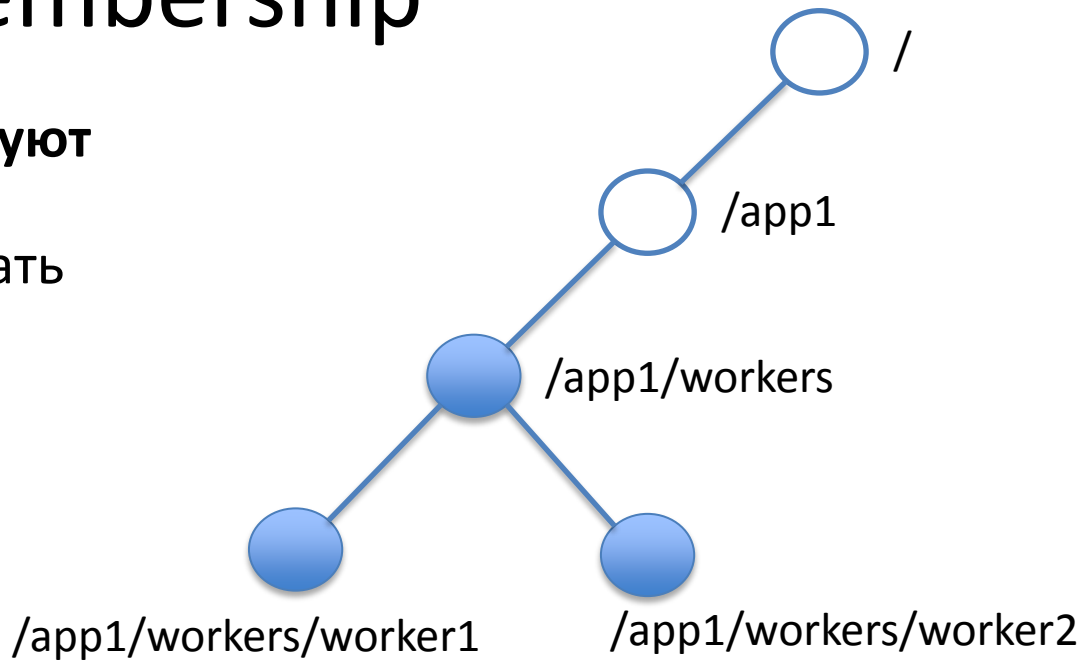


Конфигурация хранится в /app1/config

1. **getData**(/app1/config, true)
2. **setData**(/app1/config/config, data, -1)
   - Notify watching clients
3. **getData**(/app1/config, true)

Примитивы: group membership

1. Как все воркеры приложения **регистрируют себя** в ZK?
2. Как процесс может узнать все обо всех **активных** воркерах приложения?



1. create(/app1/workers/worker, data, EPHEMERAL)
2. getChildren(/app1/workers, true)

Group membership

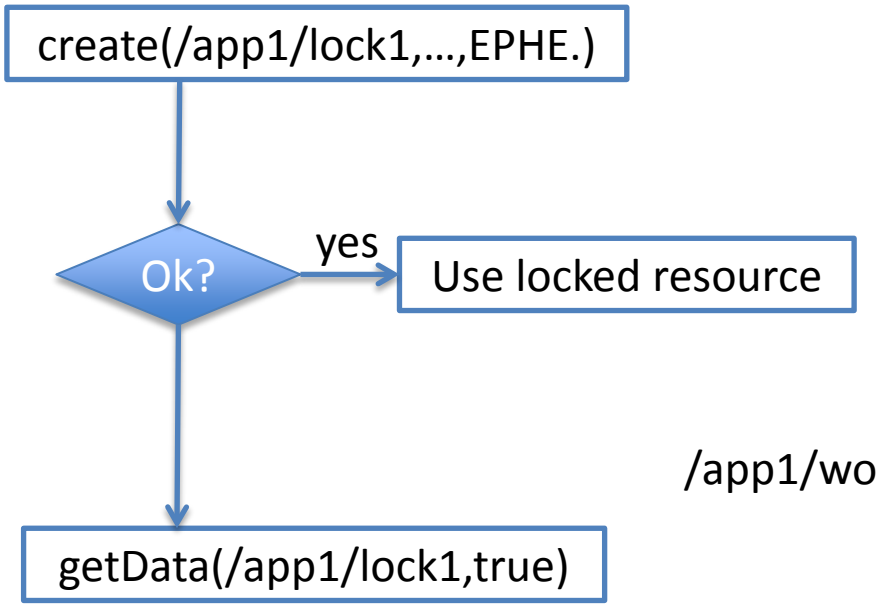
```
public class CreateGroup implements Watcher {
    private static final int SESSION_TIMEOUT = 5000;
    private ZooKeeper zk;
    private CountDownLatch connectedSignal = new CountDownLatch(1);

    public void connect(String hosts) throws IOException, InterruptedException {
        zk = new ZooKeeper(hosts, SESSION_TIMEOUT, this);
        connectedSignal.await();
    }

    @Override
    public void process(WatchedEvent event) { // Watcher interface
        if (event.getState() == KeeperState.SyncConnected) {
            connectedSignal.countDown();
        }
    }
```

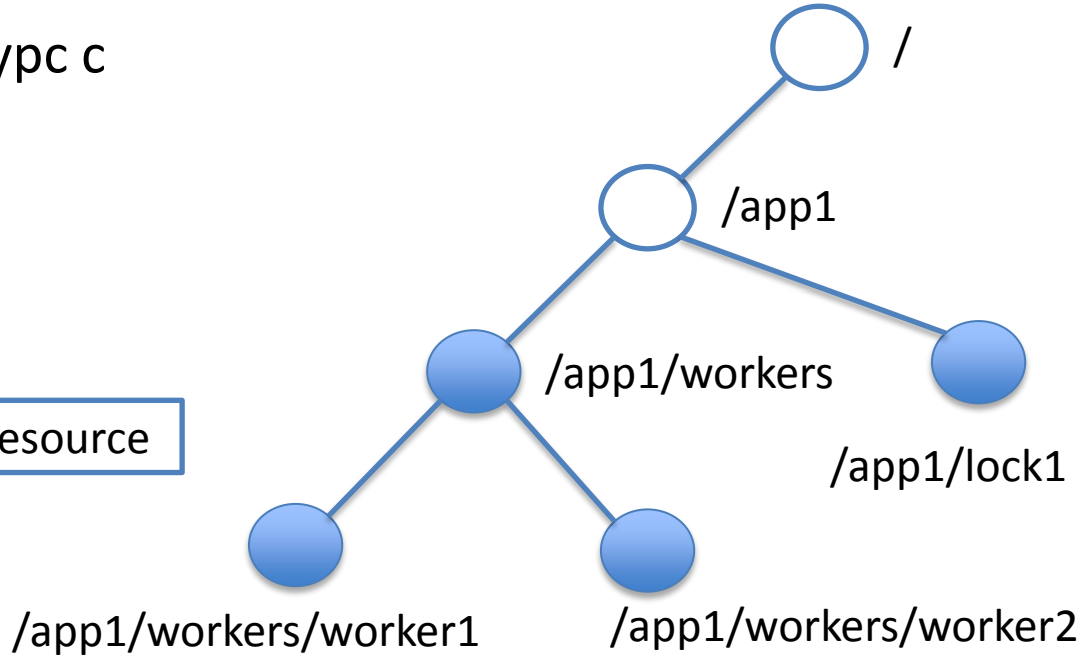
Group membership

```
    public void create(String groupName) throws KeeperException,
            InterruptedException {
        String path = "/" + groupName;
        String createdPath = zk.create(path, null/*data*/, Ids.OPEN_ACL_UNSAFE,
            CreateMode.PERSISTENT);
        System.out.println("Created " + createdPath);
    }

    public void close() throws InterruptedException {
        zk.close();
    }

    public static void main(String[] args) throws Exception {
        CreateGroup createGroup = new CreateGroup();
        createGroup.connect(args[0]);
        createGroup.create(args[1]);
        createGroup.close();
    }
}
```

Примитивы: simple locks

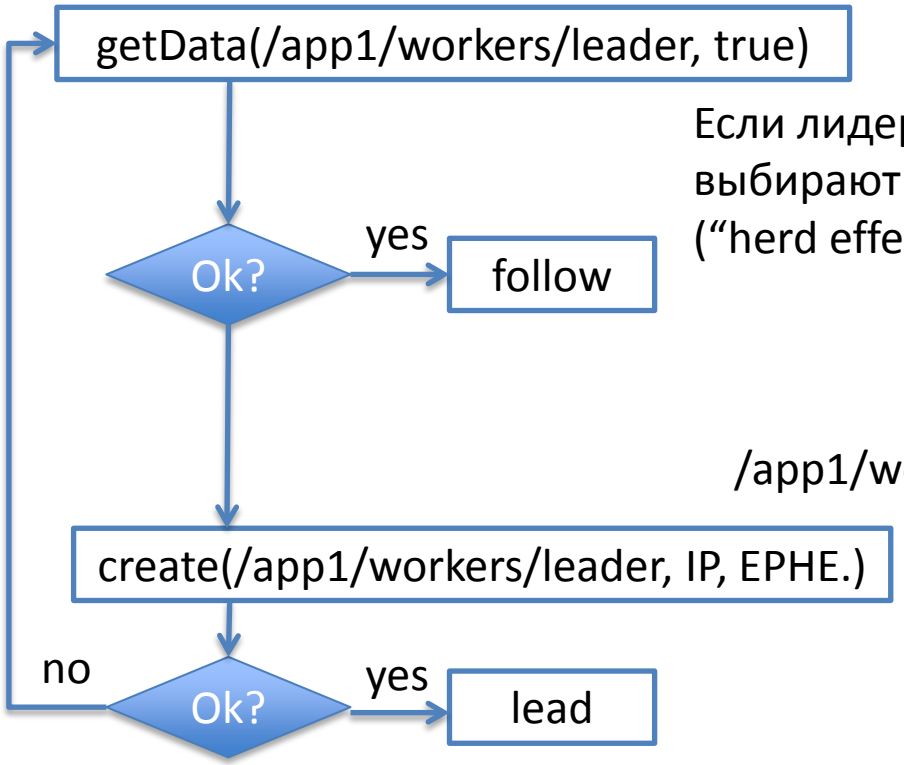
1. Как все воркеры могут использовать один ресурс с помощью lock?



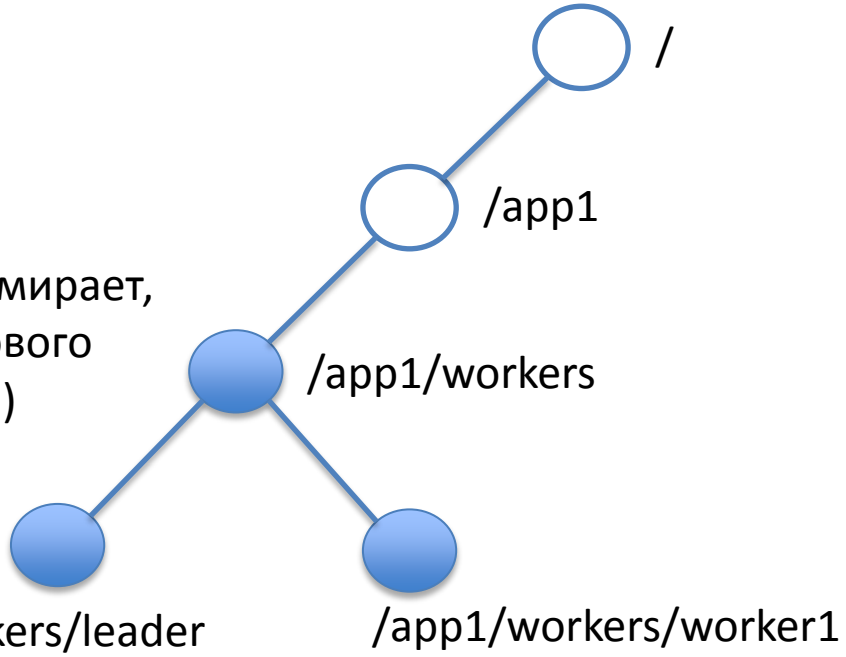


Примитивы: leader election

1. Как все воркеры могут выбрать лидера среди себя?

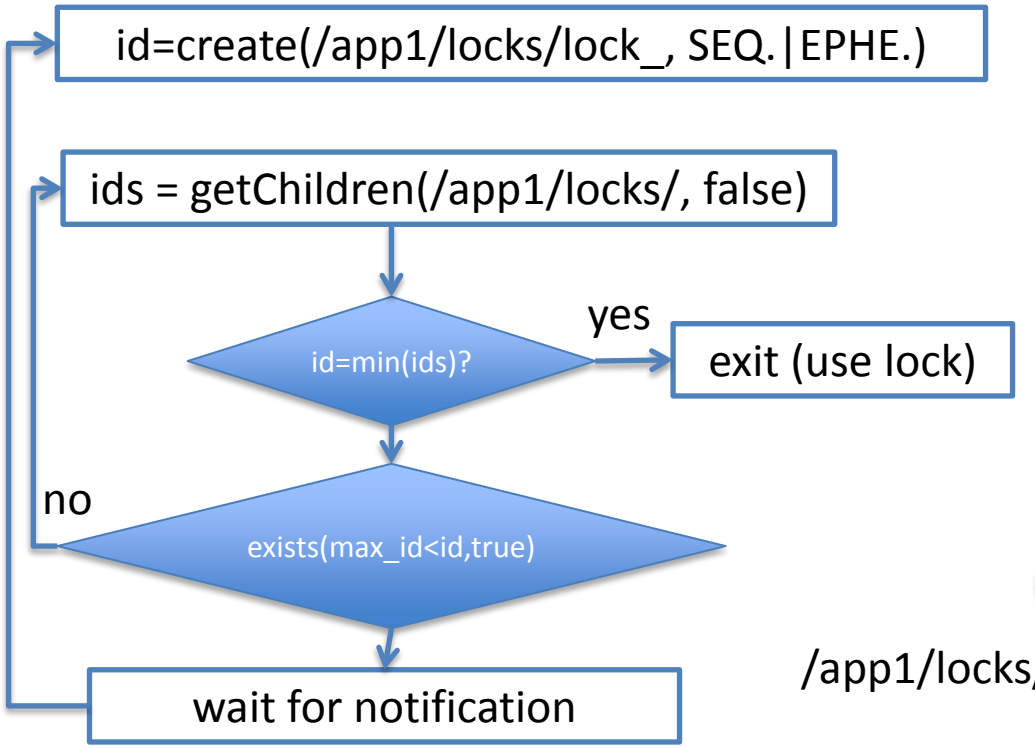


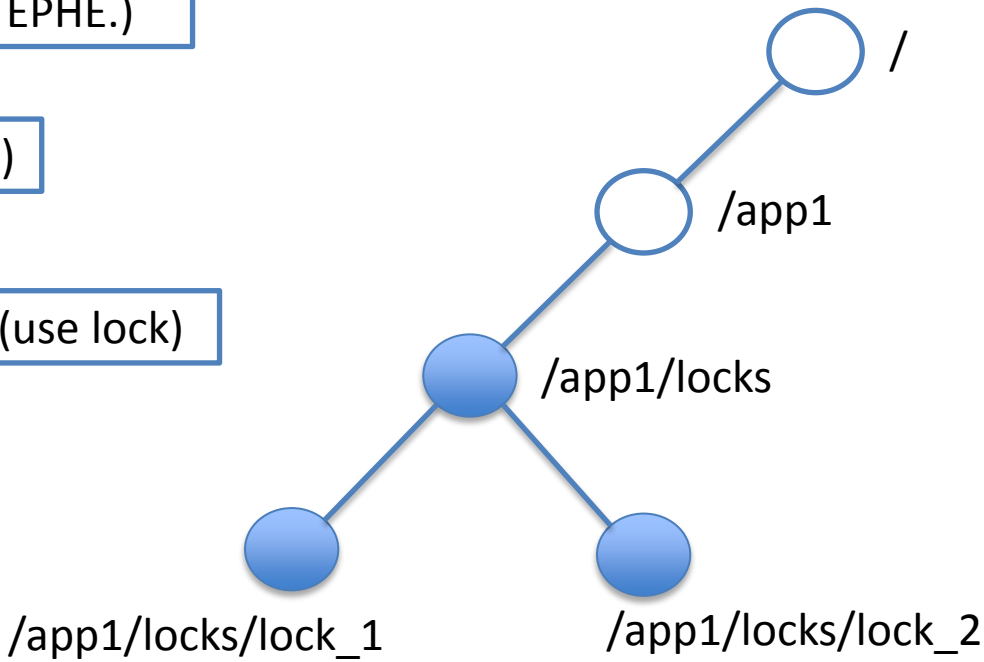
Если лидер умирает, выбирают нового ("herd effect")



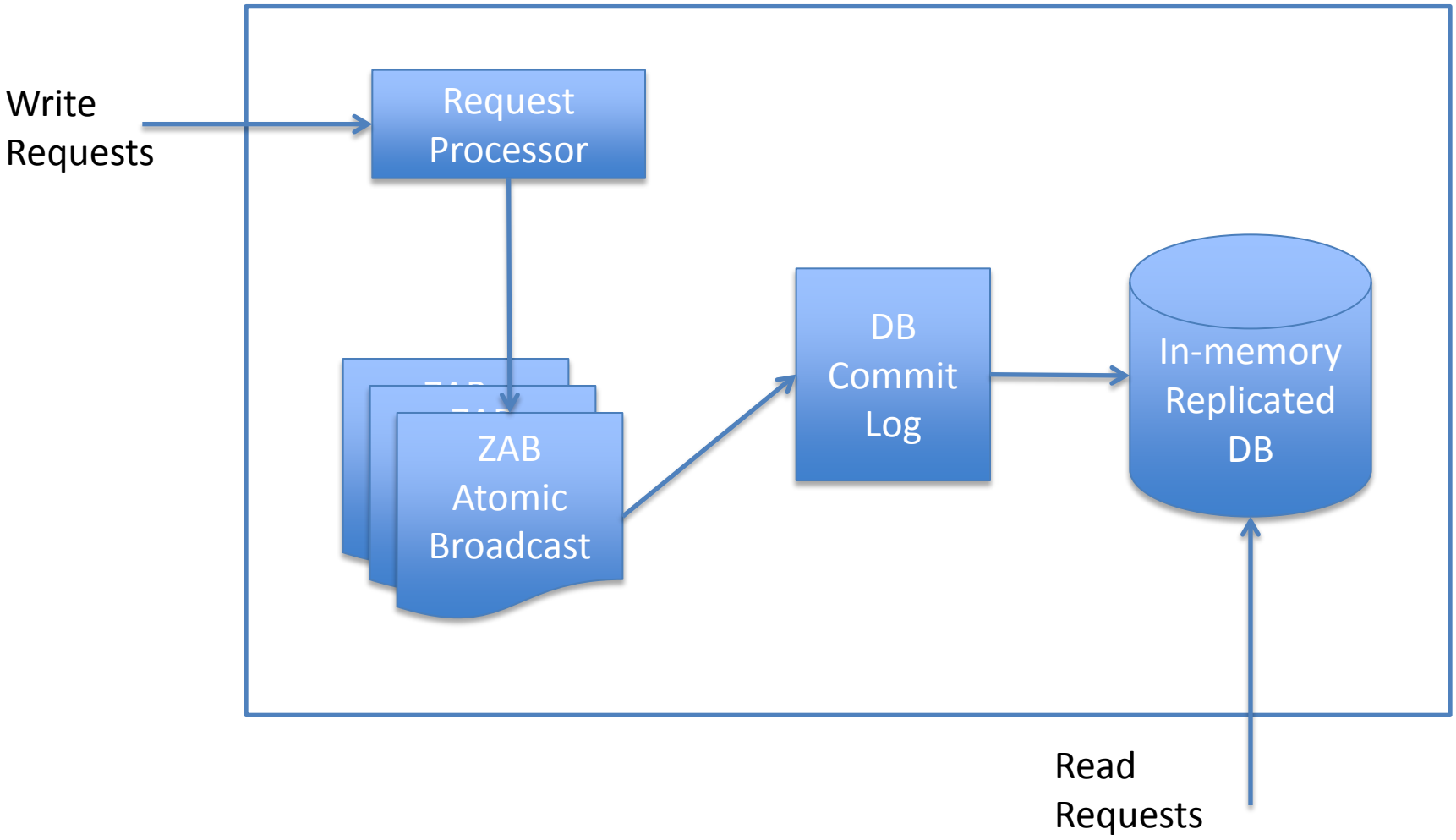
Примитивы: locking без herd effect

1. Как все воркеры могут использовать один ресурс с помощью lock?





Архитектура ZooKeeper



Zookeeper DB

- Fully replicated
  - Нет партиционирования/шардинга данных
- Каждый сервер содержит копию БД в памяти
  - Хранится полностью znode tree
  - Значение по-умолчанию 1Мб на znode
- Crash-recovery model
  - Commit log
  - Периодические снапшоты базы

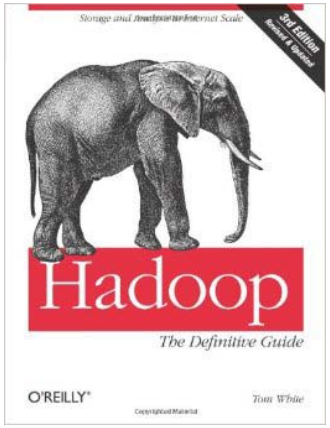
ZAB

- Используется для упорядочивания *write requests*
- *ZAB* внутри себя выбирает лидера (главную реплику)
  - Не путать с Leader Election используя ZooKeeper API
  - Другие сервер становятся фоловерами (*followers)*
- Все *write requests* отправляются фоловерами лидеру
  - Лидер нумерует запросы и выполняет *ZAB atomic broadcast*

Request processor

- При получении *write request*
  - Лидер определяет, в каком состоянии будет система после применения запроса
  - Трансформирует операцию в *transactional update*
- Такой *transactional update* затем используется в ZAB, DB
  - Гарантируется идемпотентность апдейтов к БД для одной и той же операции
- Идемпотентность: важно, т.к. ZAB может повторно отправлять сообщение
  - После восстановления прерванной операции
  - Также позволяет делать более эффективные DB снапшоты

Ресурсы



Hadoop: The Definitive Guide
Tom White (Author)
O'Reilly Media; 3rd Edition

Chapter 14 ZooKeeper

Контроль знаний

- Что такое и сколько может быть запущено на кластере:
  - JobTracker?
  - NameNode?
  - DataNode?
  - TaskTracker?
  - MasterServer?
  - RegionServer?

Контроль знаний

- MapReduce
  - Каким образом определяется количество мапперов?
  - Редьюсеров?
  - Когда и при каких условиях запускается combiner()?
  - Что приходит на вход и выход функции map() И reduce()?

Контроль знаний

- MapReduce
  - Каков процесс выполнения редьюсера?
  - Что происходит между фазами map и reduce?
  - Что происходит при падении таска?
  - Что происходит при падении TaskTracker?
  - Что такое Distributed Cache? Зачем нужно?

Контроль знаний

- HDFS
  - Из чего состоят файлы?
  - Какой размер блока у файла в HDFS?
  - Что происходит с блоками файлов при падении DataNode?
  - Что произойдет с файловой системой при падении NameNode?
  - Как клиент читает и пишет файлы?
  - Как задача MapReduce читает данные из HDFS?

Контроль знаний

- Pig и Hive
  - Что это такое? Зачем нужно? Как работает?
  - В чем отличие между собой?

Контроль знаний

- NoSQL
  - Что такое шардинг?
  - О чем говорит CAP-теорема?
  - Что такое *Eventual Consistency*?
  - Зачем нужны S*chema-less DB*?
  - Что такое *Consistent Hashing*?

Контроль знаний

- Hbase
  - Как хранятся данные в Hbase?
  - Что такое регион в таблице?
  - Из чего он состоит?
  - Что происходит при падении RegionServer?
  - Зачем нужны операции compactions? Какие бывают? В чем отличие?

Вопросы?