Modern Storages and Data Warehousing
Week 2 - File storages

Попов Илья, i.popov@hse.ru

1 - Репликация

Репликация - зачем?

- Мы хотим разгрузить нашу базу
  - Оставить write нагрузку на мастере
  - Увести аналитиков с read нагрузкой (и блокировками на чтение) куда-нибудь подальше от прода
- Хочется, чтобы при смерти основного instance БД (master) сервис не умирал, а переключался на резервный instance БД (hot standby replica)

Демо 0-2

Репликация - чего еще хотим?

〉 Хотим, чтобы базу нельзя было задудосить кучей открытых подключений
〉 Хотим, чтобы мастер сам переключался при смерти одного хоста
〉 Хотим, чтобы наши запросы всегда шли в живой хост

Репликация - чего еще хотим?

〉 Хотим, чтобы базу нельзя было задудосить кучей открытых подключений
〉 Хотим, чтобы мастер сам переключался при смерти одного хоста
〉 Хотим, чтобы наши запросы всегда шли в живой хост

〉 Для всего этого есть инструменты!

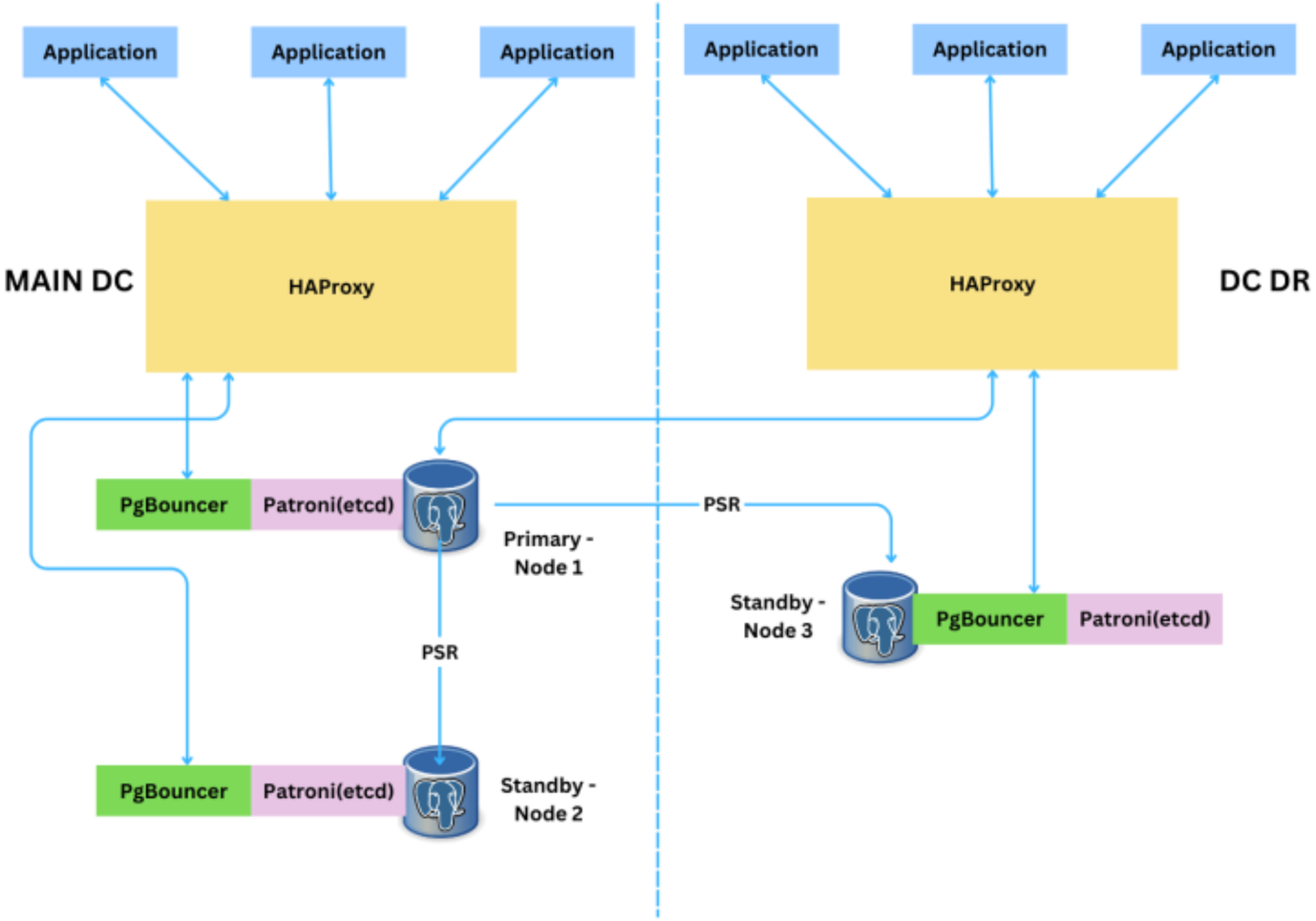
Репликация - чего еще хотим?

〉Хотим, чтобы базу нельзя было задудосить кучей открытых подключений - **pgbouncer**

〉Хотим, чтобы мастер сам переключался при смерти одного хоста - **patroni**

〉Хотим, чтобы наши запросы всегда шли в живой хост - **HAProxy**

High Availability PostgreSQL Cluster



Демо 3

2 - Yet another лирическое отступление

3 - S3
Simple Storage Service

Что такое S3

〉 S3 уачинался как проект Xen в Кембриджском университете, 2006
〉 На Xen построил свое хранилище Amazon, после начал дорабатывать и продавать в рамках пакета AWS
〉 После Amazon нанял к себе команду разработчиков Xen
〉 Проект стал настолько популярен, что де-факто стал единственным стандартом в индустрии
〉 Сторонние проекты начали делать свои хранилища с API, полностью совместимым с S3, чтобы облегчить жизнь разработчикам

〉 Так появились S3-like хранилища как класс

Почему S3

〉 Говорю S3 - подразумеваю любые **Object Storage** в принципе вне зависимости от реализации и вендора (AWS S3, Yandex Cloud MDS, Minio, Ceph и так далее и тому подобное)

Почему S3

〉 Говорю S3 - подразумеваю любые **Object Storage** в принципе вне зависимости от реализации и вендора (AWS S3, Yandex Cloud MDS, Minio, Ceph и так далее и тому подобное)

〉 Дешевые и бездонные

Почему S3

〉 Говорю S3 - подразумеваю любые **Object Storage** в принципе вне зависимости от реализации и вендора (AWS S3, Yandex Cloud MDS, Minio, Ceph и так далее и тому подобное)

〉 Дешевые и бездонные

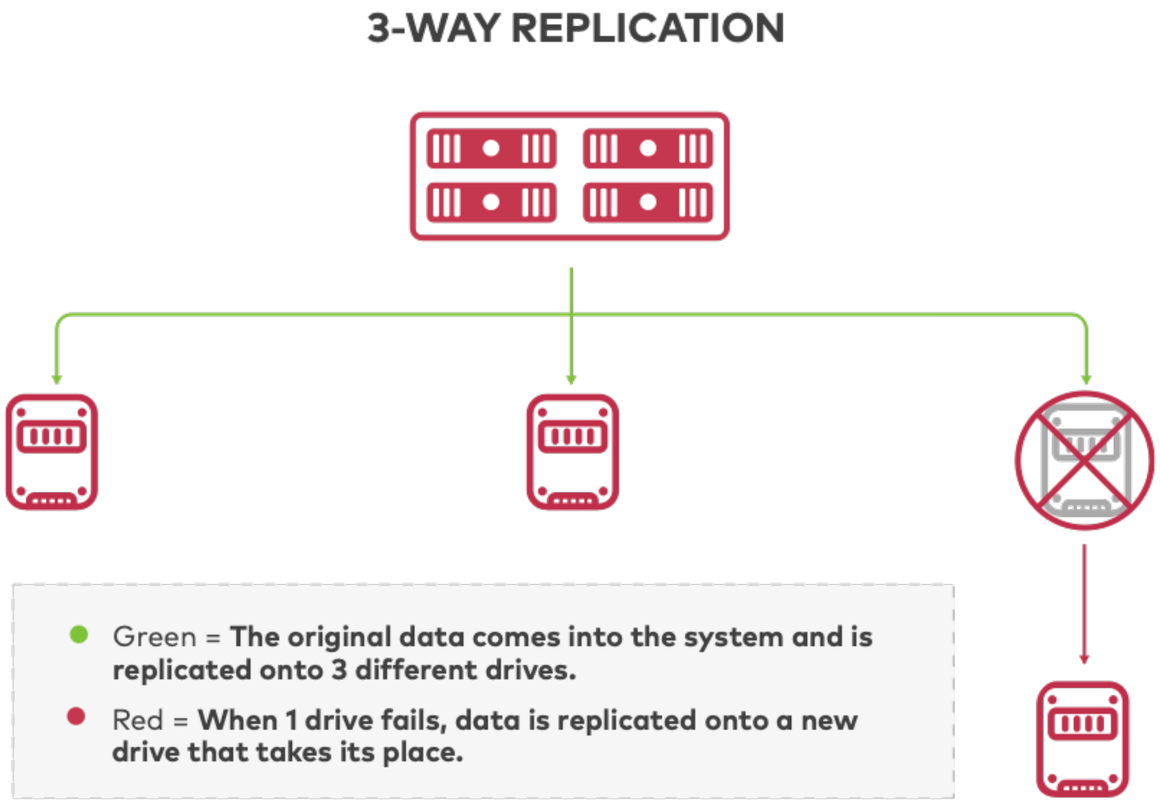
Почему S3

〉 Дешевые и бездонные
〉 SDK к S3 есть практически на любом языке программирования
〉 Из коробки поддерживают Erasure Coding
〉 Могут выступать как CDN

Минусы:
〉 Изменения в файле требуют его полной перезаписи
〉 Умеют только сжимать и кодировать данные, вычислять на них ничего не получится
〉 AWS стоит денег (и русские деньги больше не принимает)
› **Но как правило Object Storage - самое дешевое КХД**

Непонятное слово - Erasure Coding

- Есть классический метод бэкапирования данных - RAID-массивы
- Хранятся несколько полных копий данных на разных дисках (обычно на трех)
- Если один выходит из строя - его меняют, после чего массив восстанавливается без даунтайма системы



Непонятное слово - Erasure Coding

〉 Данные бьются на более мелкие блоки (биты, байты или блоки) - data blocks

〉 Блоки данных раскидываются по разным дискам (файлам)

〉 Для нескольких ячеек данных насчитывается и сохраняется некоторое количество ячеек четности - parity blocks

〉 Ошибка в любой чередующейся ячейке устраняется через обратную операцию декодирования на основе сохранившихся данных и ячеек четности

〉 Экономит до 50% места. RAID3 для 6 блоков - 6*3 = 18 блоков, EC с 3 parity blocks - 6+3 = 9 блоков

〉 Вся эта магия работает с помощью алгоритма Рида-Соломона

〉 Потребляет CPU и межкластерный network

Как хранятся данные в S3

〉 Данные хранятся в **бакетах** - можно их считать местными папками
〉 К бакету можно применять **политики** - правила доступа к данным
〉 В бакет складываются **объекты** - произвольные файлы
〉 Файлы можно дополнительно **каталогизировать**: добавлять теги, метаданные, настраивать время хранение и правила жизненного цикла

Всё ¯_(ツ)_/¯

Нам S3 нужен для:
〉 Сохранить данные от пользователя
〉 Раздать контент на фронт сервиса
〉 Сохранить что-то далеко, дешево и надолго

Как хранятся данные в S3

〉 Данные хранятся в **бакетах** - можно их считать местными папками
〉 К бакету можно применять **политики** - правила доступа к данным
〉 В бакет складываются **объекты** - произвольные файлы
〉 Файлы можно дополнительно **каталогизировать**: добавлять теги, метаданные, настраивать время хранение и правила жизненного цикла

Всё ¯_(ツ)_/¯

Есть еще **межкластерная** и **межрегионная репликация**, но те из вас кто не будут строить High Load CDN для сервиса уровня YouTube, Instagram или Amazon, скорее всего никогда с этим не столкнутся.

Демо 4

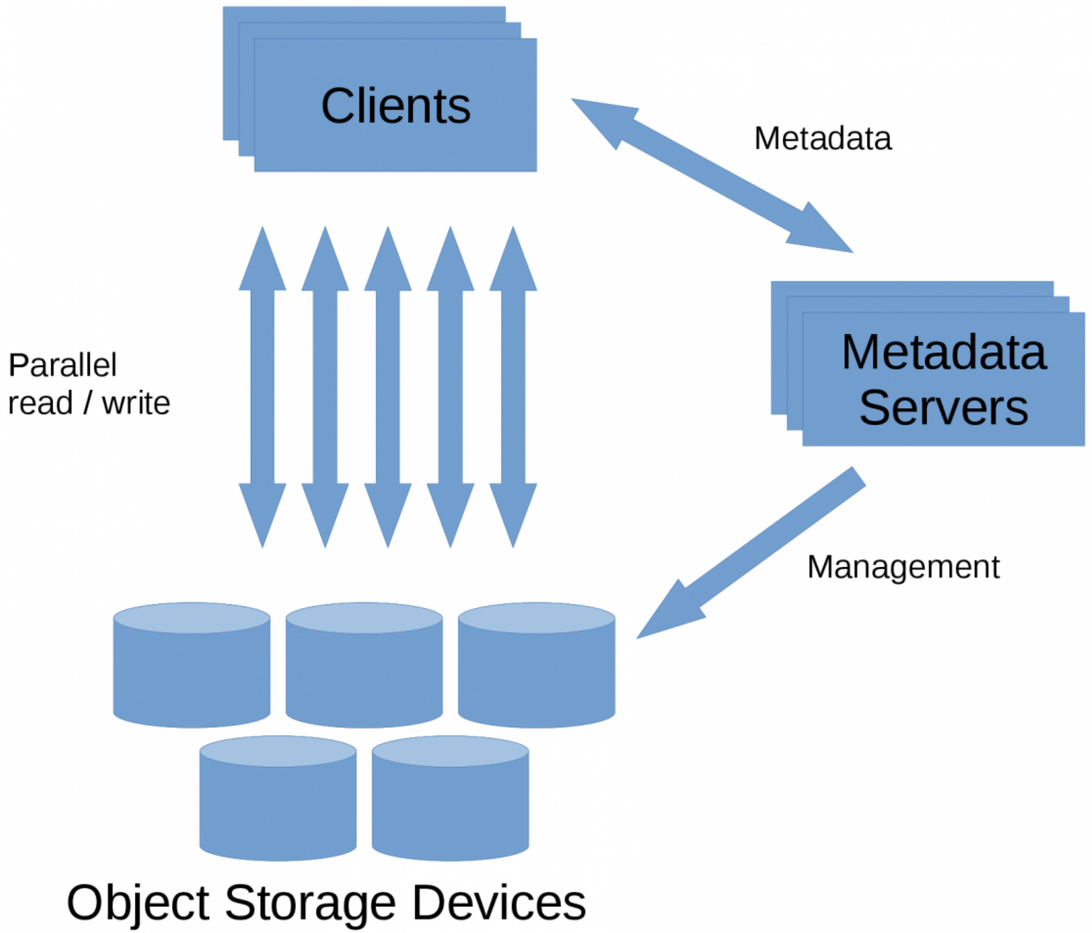
4 - Hadoop экосистема

Мотивация

〉 Помним, что мы не хотим, чтобы аналитики ходили в OLTP
〉 Помним, что мы не хотим хранить в OLTP избыточный объем истории
〉 Помним, что одного хоста нам не хватает для эффективной обработки больших данных, и мы хотим распределять вычисления

Берем то, что уже видели в S3 и немного меняем

- На одну машину все не поместится
- Хотим иметь возможность не только чтение/перезапись, но и изменение/добавление данных в файл
- Хотим распределенное хранилище без bottle neck в виде подключения к одной ноде

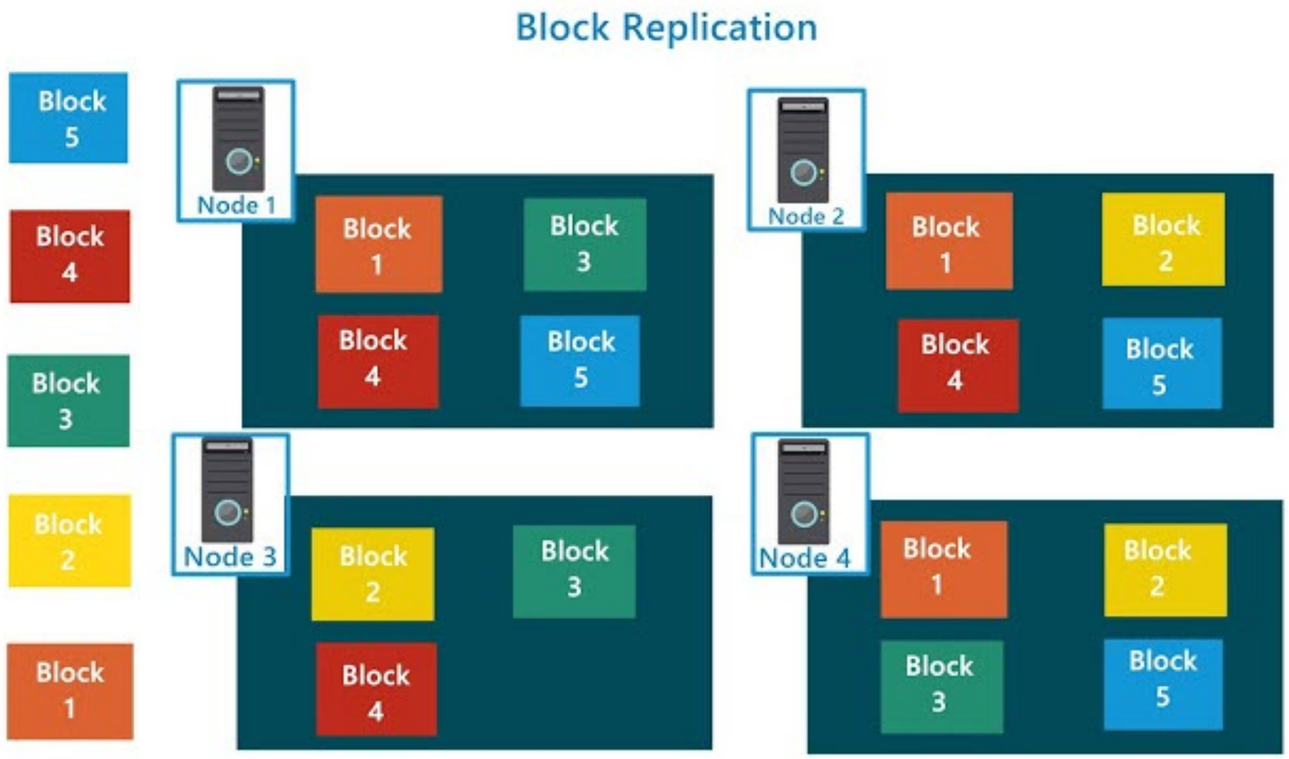


Distributed File Storage

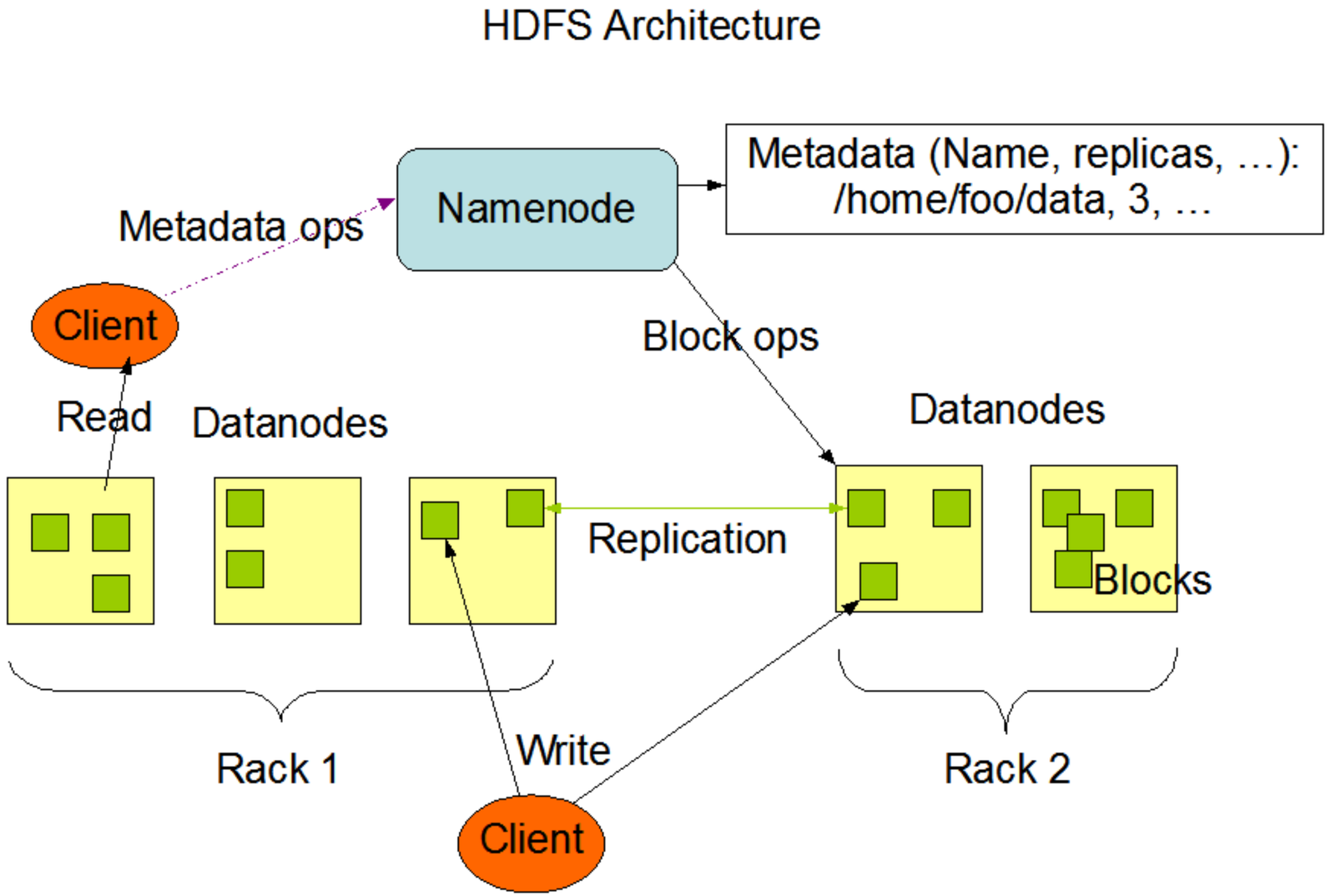
〉 Хотим, чтобы хранилище было устойчивое и не теряло данные

〉 Любая машина может сломаться в любой момент

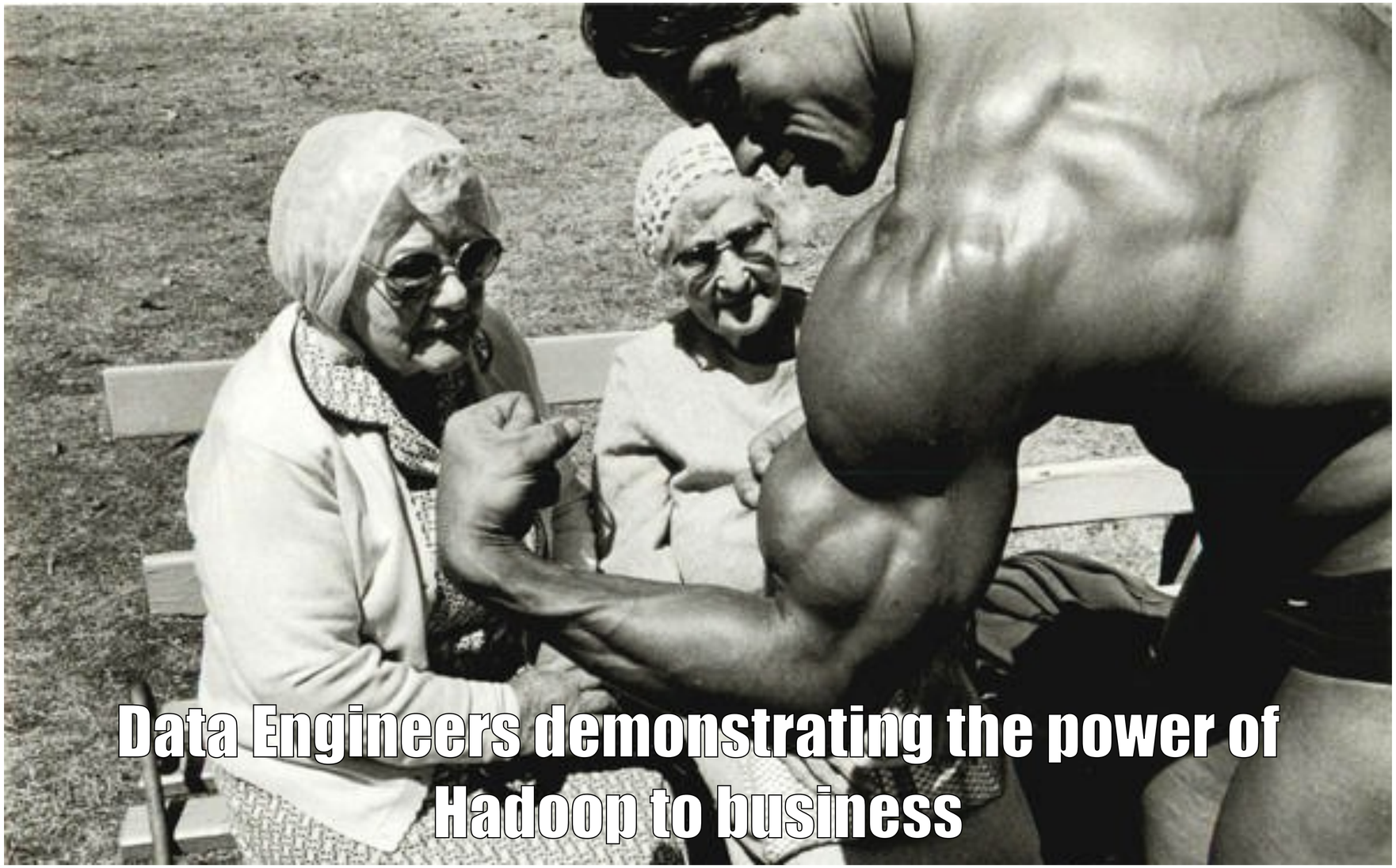
〉 Дробим данные на блоки, а блоки разносим по разным машинам так, чтобы при выходе одной из машин данные оставались целостными



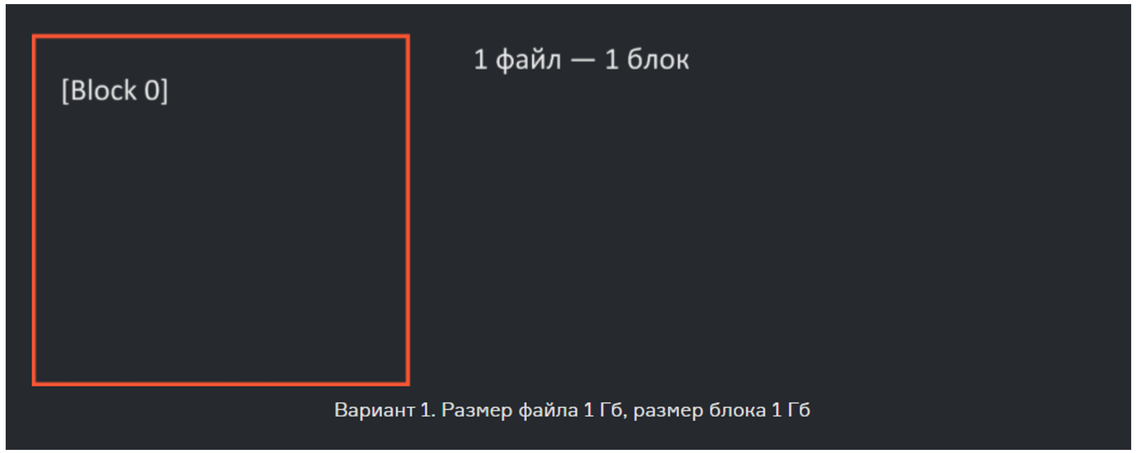
Please welcome - HDFS



Please welcome - HDFS

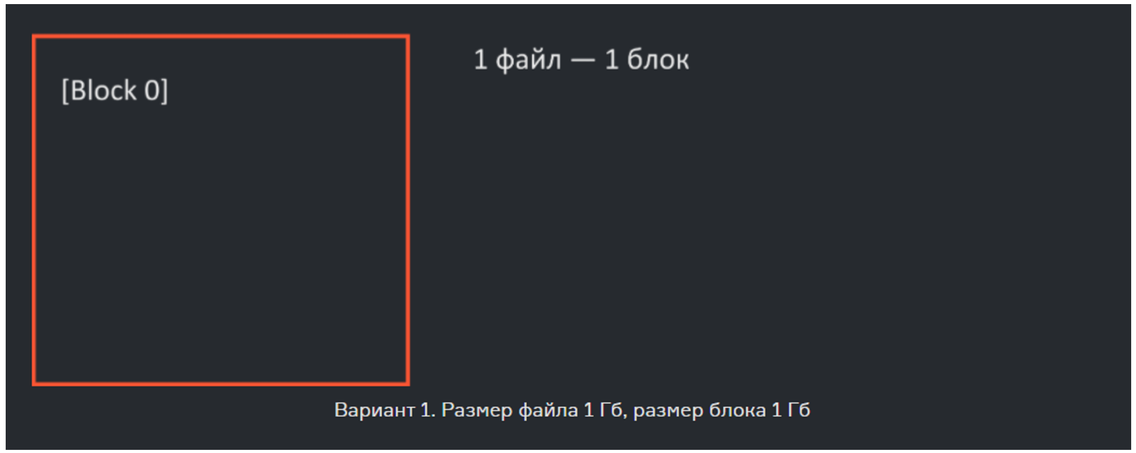


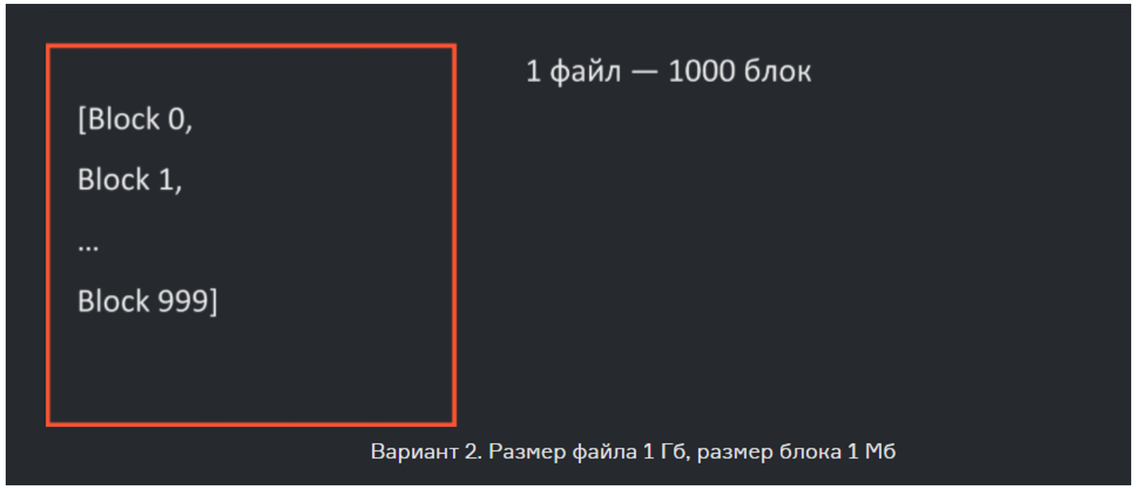
Проблема мелких файлов



- Наличие большого кол-ва файлов/ блоков, приводит к тому, что кол-во объектов, которые хранятся в памяти увеличиваются в разы
- Для описания метаинформации о файле в первом случае нам нужно иметь 3 объекта: файл - 1 шт., блок - 1 шт., массив адресов - 1 шт.

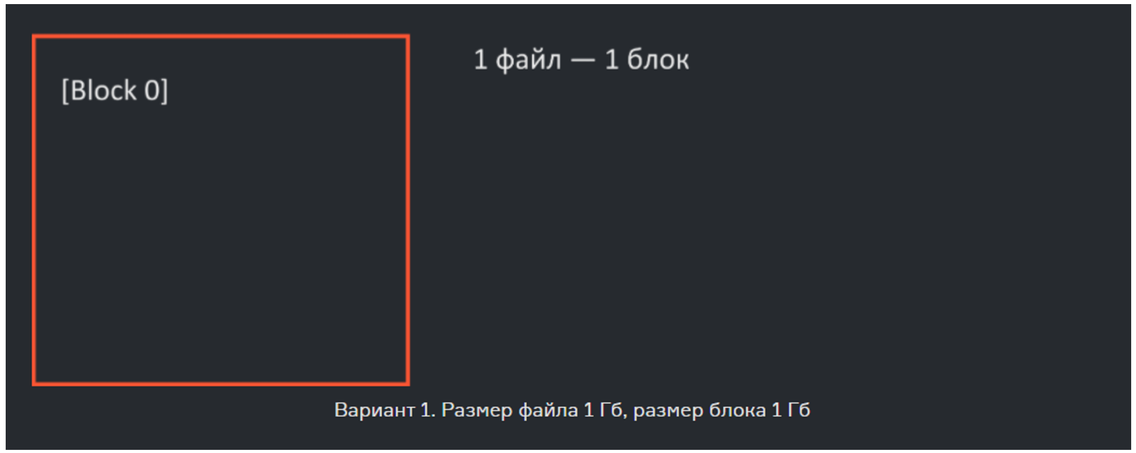
Проблема мелких файлов

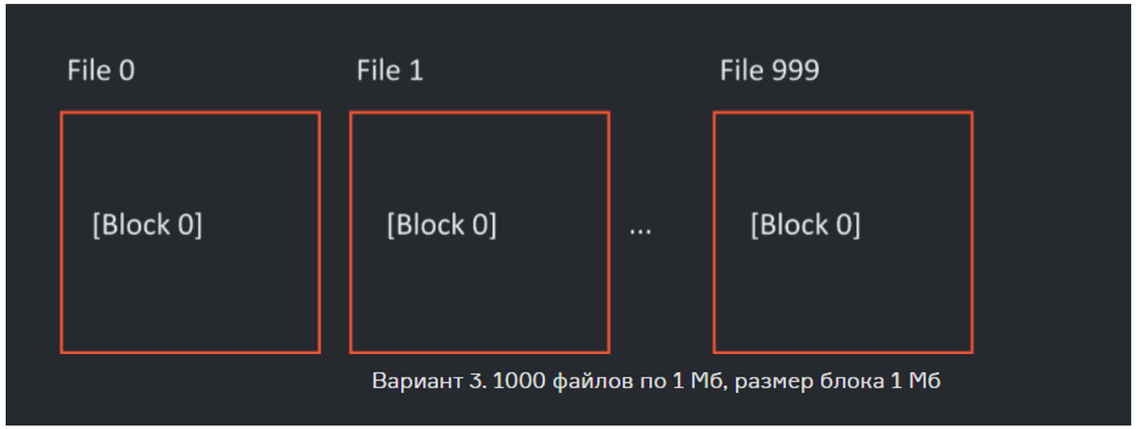




- Наличие большого кол-ва файлов/ блоков, приводит к тому, что кол-во объектов, которые хранятся в памяти увеличиваются в разы
- Для описания метаинформации в первом случае нам нужно иметь 3 объекта: файл - 1 шт., блок - 1 шт., массив адресов - 1 шт.
- Во втором случае - 1002 объекта: файл - 1 шт., блок - 1000 шт., массив адресов - 1 шт.

Проблема мелких файлов





- Наличие большого кол-ва файлов/ блоков, приводит к тому, что кол-во объектов, которые хранятся в памяти увеличиваются в разы
- Для описания метаинформации в первом случае нам нужно иметь 3 объекта: файл - 1 шт., блок - 1 шт., массив адресов - 1 шт.
- Во втором случае - 1002 объекта: файл - 1 шт., блок - 1000 шт., массив адресов - 1 шт.
- 3000 объектов: файл - 1000 шт., блок - 1000 шт., массив адресов - 1000 шт.

Демо 5.1

Правило анализа данных #1: Если что-то можно посчитать на базе, а не на нашей машине - считаем на базе

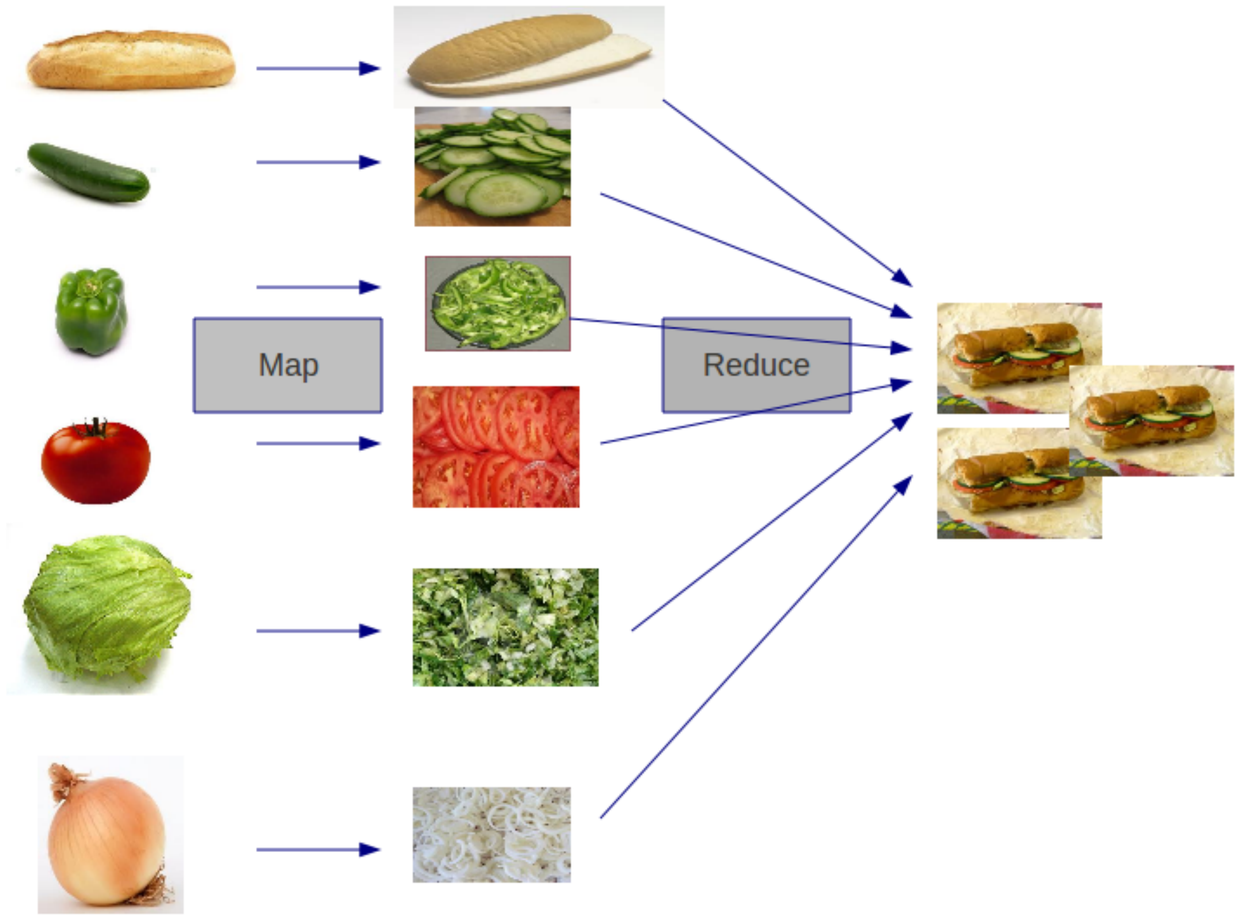
Compute Engine

〉Хотим запускать вычисления над распределенными данными
〉Хотим чтобы эти вычисления тоже были распределенные
〉Нам нужна какая-то платформа, которая будет координировать выполнение кода
〉Нам нужна какая-то платформа, которая будет что-то делать, если одна из машин в кластере умерла
〉Нам нужна какая-то платформа, которая будет решать, как пересылать данные между машинами для агрегации

〉Хотим написать для всего этого фреймворк один раз и больше никогда об этом не думать

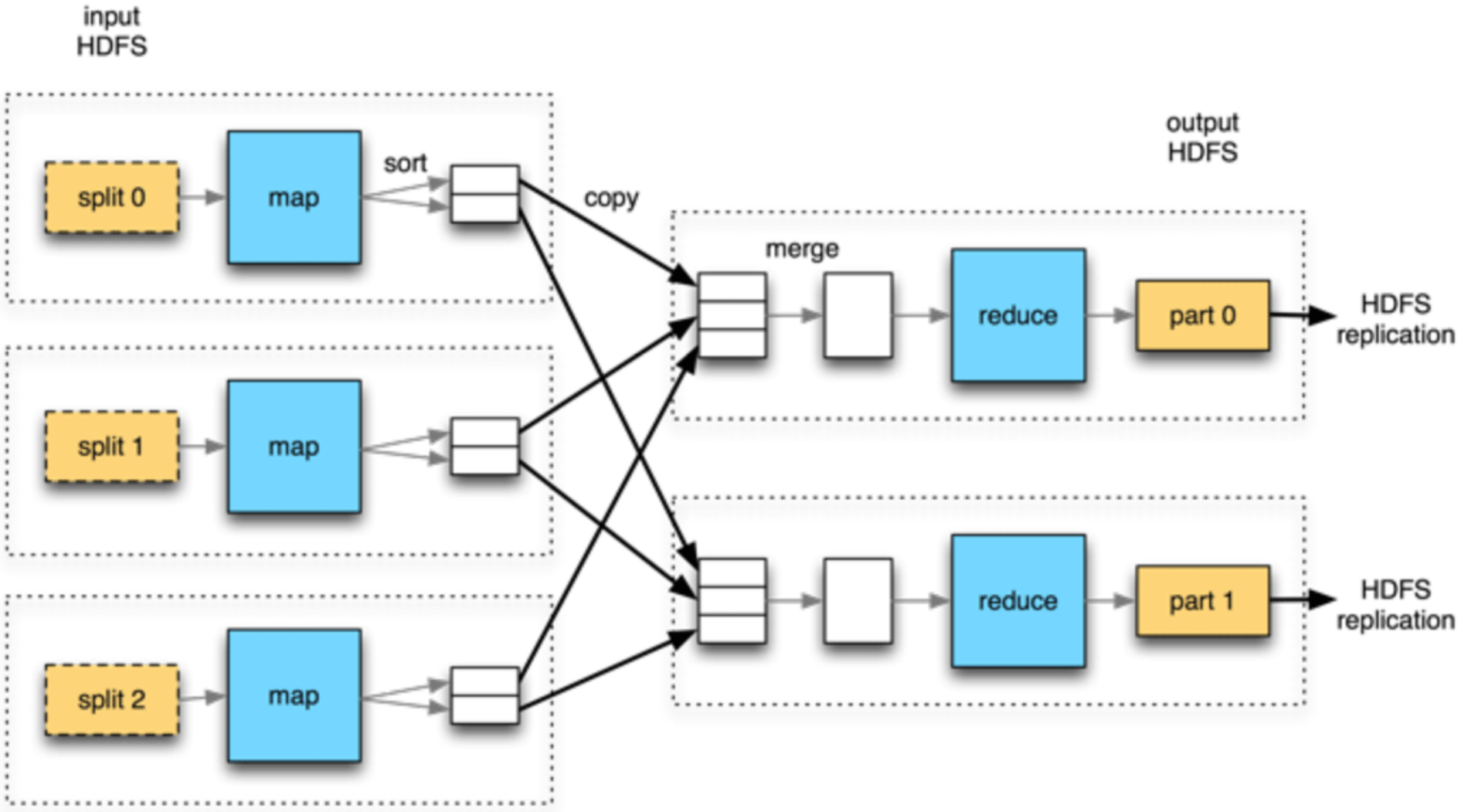
Please welcome - Map Reduce

- Бьем вычисления на фазы: Map, Reduce, Sort
- Кластер за нас думает, где и как он будет запускать эти операции (помните, что у нас данных по 3 копии на разных нодах) и как делать shuffle
- Мы радостно садимся думать, как же писать эти Map и Reduce



Хрестоматийный пример - word count

Еще раз Map Reduce



Какие тут есть проблемы?

Please welcome - Spark

- Spark - эффективный фреймворк для обработки больших данных.
- Эффективная распределенная DAG (directed acyclic graph) модель вычислений (не только MapReduce)
- "Ленивая" модель вычислений. Она позволяет не выполнять операции над объектами, которые дальше нигде не будут использованы. Таким образом лишние вычисления пропускаются
- Гибкие механизмы управления памятью:
  - предпочтение хранения данных в памяти (это эффективнее с точки зрения доступа и производительности)
  - возможность сброса данных на диск при нехватке памяти (это негативно сказывается на производительности, но зато позволяет не падать джобам, которые не помещаются в оперативную память)

RDD

〉 Базовая абстракция Spark
〉 Resilient Distributed Dataset

RDD

〉 Базовая абстракция Spark
〉 Resilient Distributed Dataset

〉 **Dataset** - строго типизированный мешок с данными

RDD

〉 Базовая абстракция Spark
〉 Resilient Distributed Dataset

〉 **Dataset** - строго типизированный мешок с данными
〉 **Distributed** - распределенный по разным машинам. Датасет разбивается на партиции, которые могут храниться и обрабатываться по отдельности

Действия над RDD

〉 Create: physical -> RDD
Чтение из хранилища, создание из данных в памяти

〉 Lazy Transformations: *RDD -> RDD
map, flatMap, filter, join, etc…

〉 Action: RDD -> physical
Запись в хранилище, материализация

RDD

〉 Базовая абстракция Spark
〉 Resilient Distributed Dataset

〉 **Dataset** - строго типизированный мешок с данными
〉 **Distributed** - распределенный по разным машинам. Датасет разбивается на партиции, которые могут храниться и обрабатываться по отдельности
〉 **Resilient** - RDD является цепочкой преобразований партиций. Если какая-то партиция потерялась - её можно пересчитать. Для этого мы идем по графу вычислений и пересчитываем только то, что нам нужно.

Spark SQL

〉 Оно же называется Dataframe API

〉 RDD под капотом
〉 Есть более верхнеуровневый API, похожий на SQL
〉 Оптимизатор запросов Catalyst
〉 Работа с Dataframe и Dataset

Демо 5.2

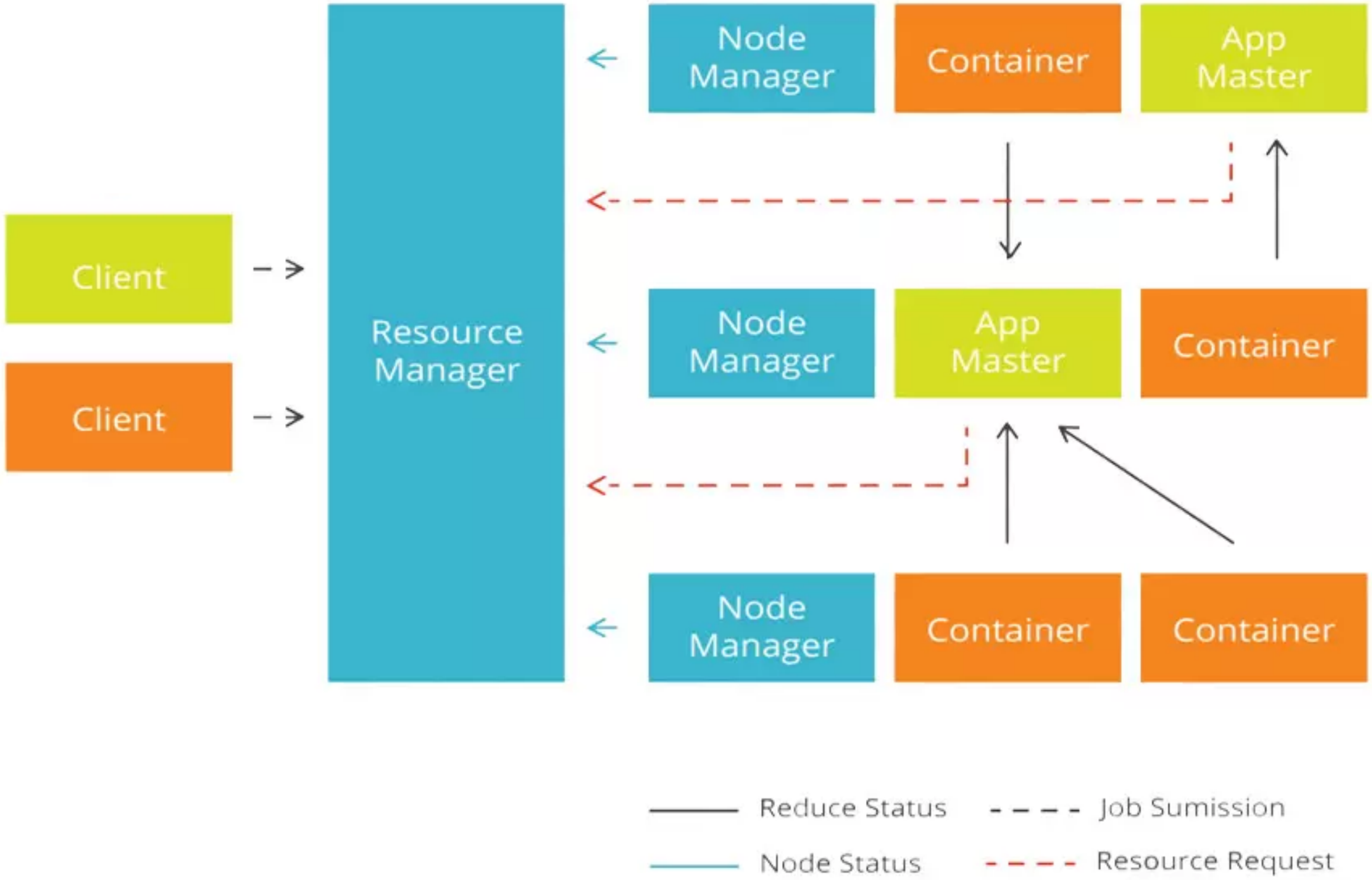
А кто за этим всем присматривать будет?

〉 Есть кластер, в нем много машин с CPU и RAM
〉 Есть много независимых процессов, которые надо запускать на кластере

〉 Как решить, какой процесс запустить на какой машине?
〉 Что делать, если свободных ресурсов меньше, чем требуется?
〉 Как контролировать, что процесс занял не больше ресурсов, чем выделено?

〉 Нужен отдельный сервис, который бы управлял ресурсами кластера

Please welcome - Hadoop YARN



Почему нельзя просто так взять, и выделить ресурсы?

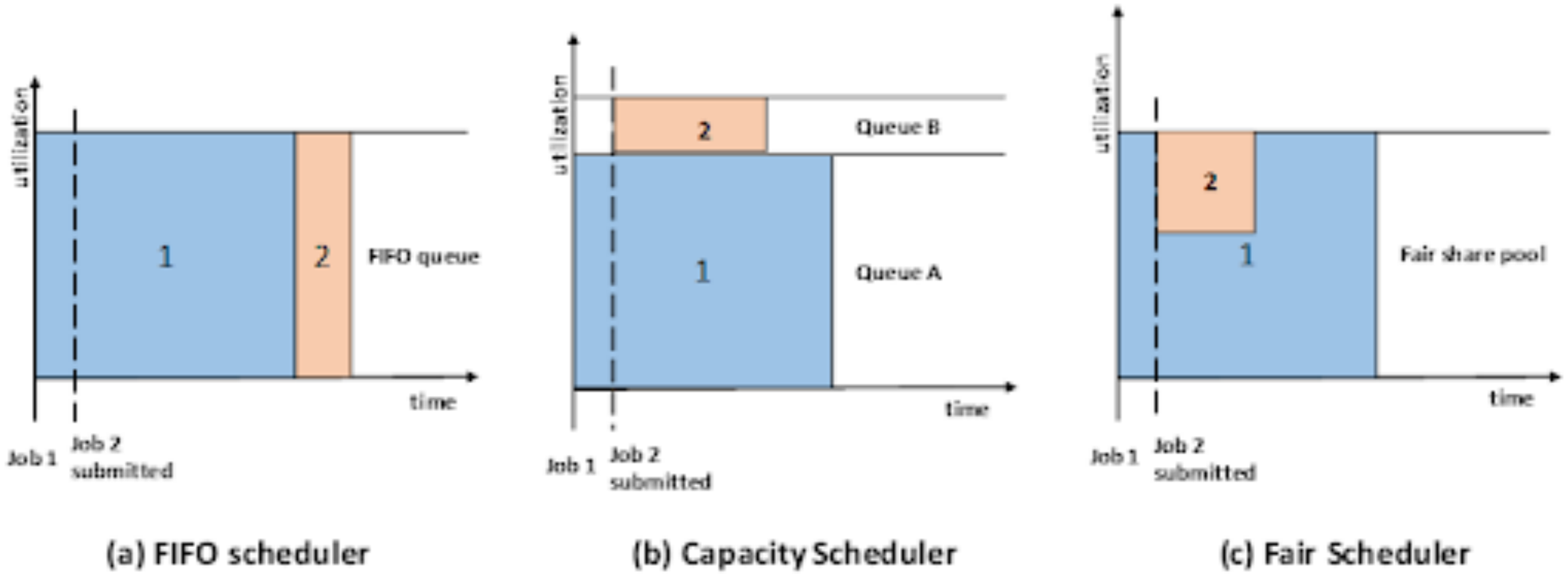
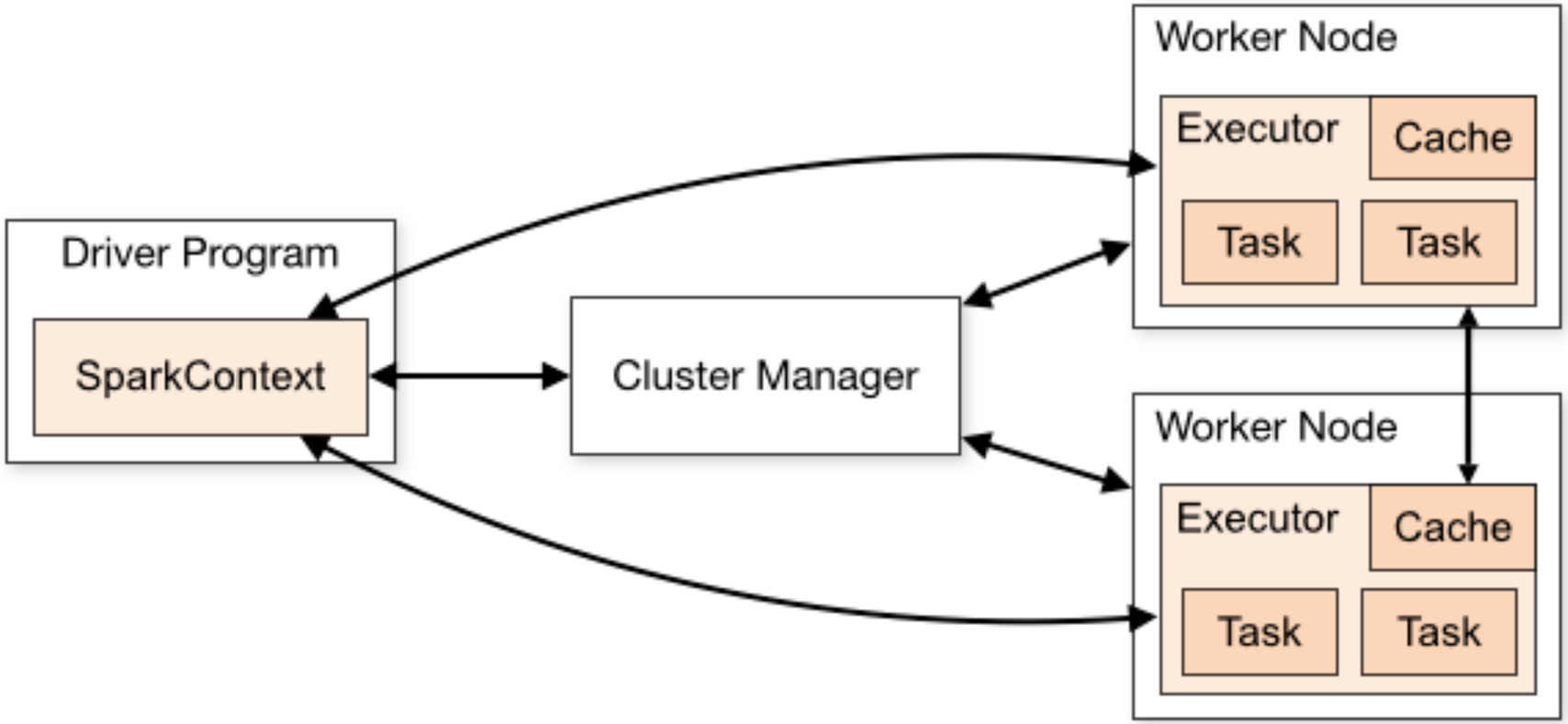


Figure 1: YARN Schedulers' cluster utilization vs. time

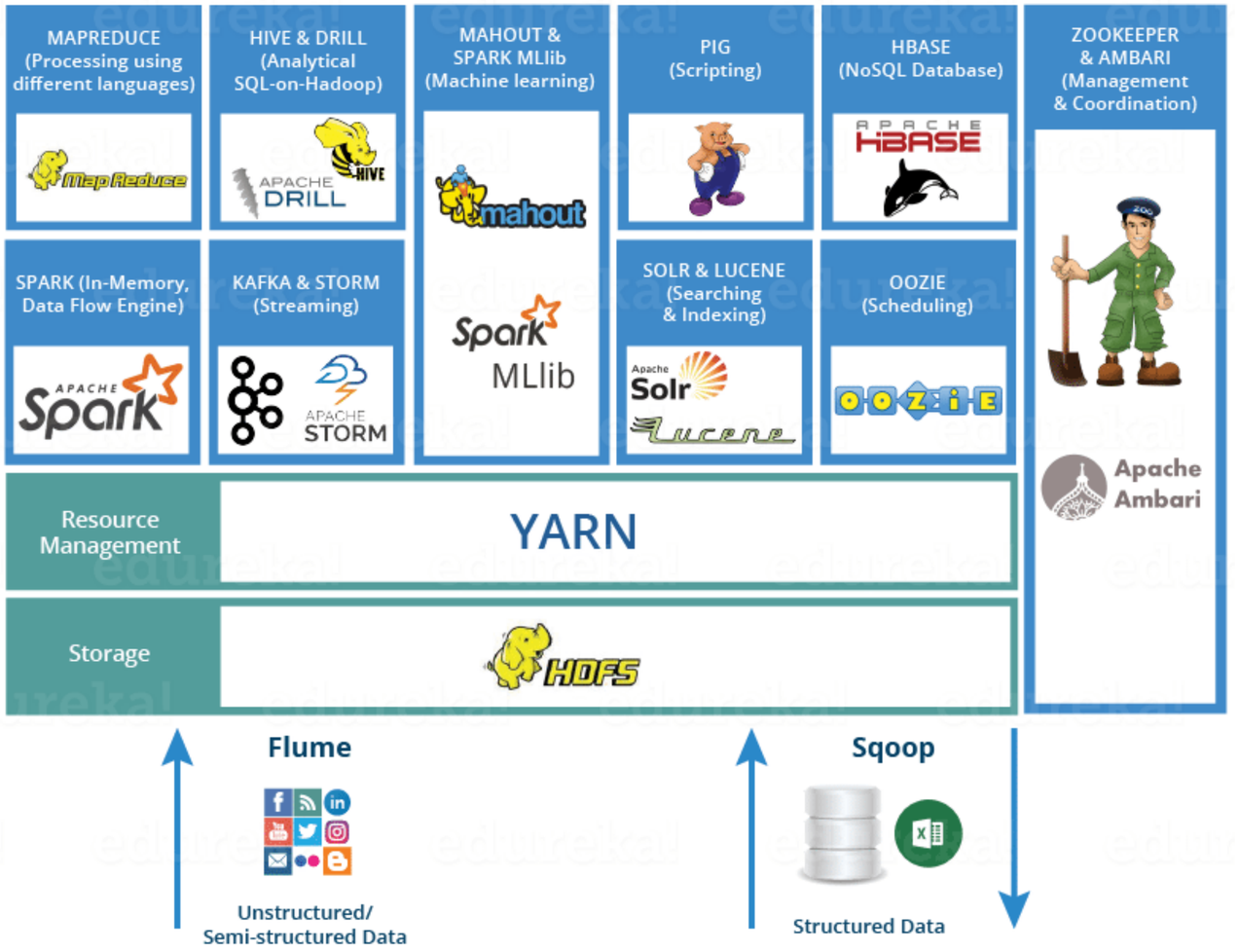
Еще раз про Apache Spark



Итого Hadoop-стэк

〉 Hadoop Distributed File System (HDFS)
〉 Hadoop MapReduce
〉 Hadoop YARN
+ Apache Spark - удобный движок для расчетов

+Hive - SQL движок для Big Data
+HBase - Колоночная БД над HDFS
+много самого разного



That’s all, Folks!