НЕТОЛОГИЯ
групп

HADOOP & SPARK

Алексей Кузьмин

Директор разработки; Data Scientist

ДомКлик.ру

aleksej.kyzmin@gmail.com

Принципы работы с большими данными

- Независимое обрабатываем независимо
  - Это позволит обрабатывать независимые данные параллельно
- Принцип локальности данных
  - Обрабатываем там же, где храним
- Пошаговая работа
  - Сложный процесс обработки можно декомпозировать на несколько простых.

Hadoop

- Проект "The ApacheTM HadoopTM“ разрабатывает open- source ПО для отказоустойчивых, масштабируемых и распределенных вычислений
- – Работает с BigData на обычных серверах
- – Сильное open-source комьюнити
- – Много различных продуктов и средств используют Hadoop

Кластер Hadoop

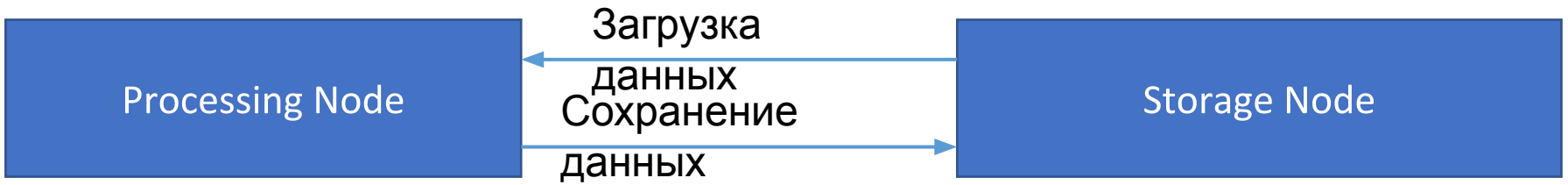
- • “Дешевое” обычное железо (Commodity Hardware)
- • Соединенное по сети
- • Расположено в одном месте – Сервера в стойках в датацентре

Принципы Hadoop

- Горизонтальное масштабирование вместо вертикального
- Отправляем код к данным
- Уметь обрабатывать падения и отказы оборудования
- Инкапсуляция сложности работы распределенных и многопоточных приложений

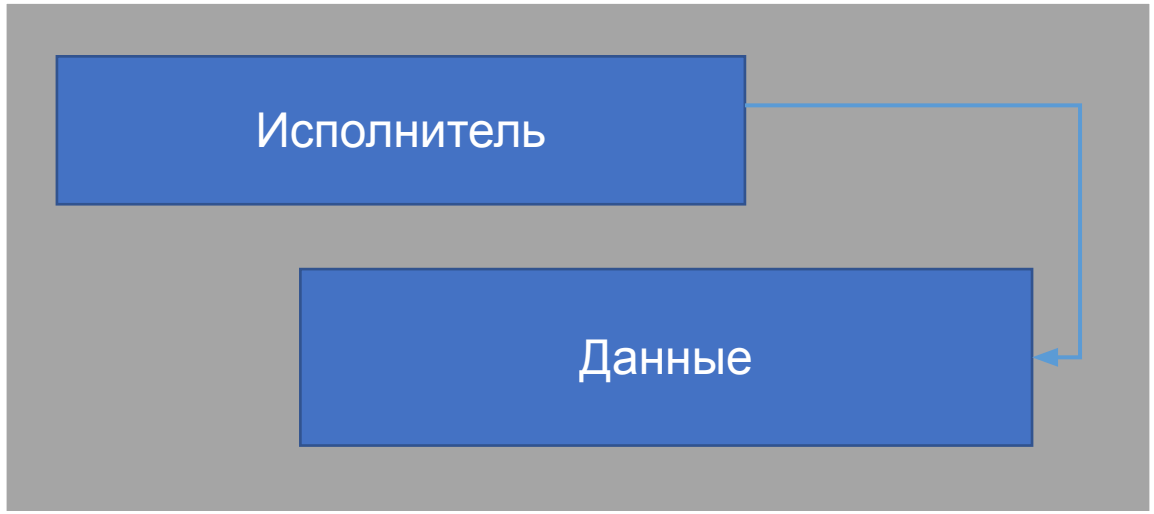
Доставка кода к данным

- Традиционная архитектура системы обработки данных
  - – Ноды системы разделяются на вычислительные и стораджи, соединяются высокоскоростным линком
  - – Многие приложения обработки данных являются CPU-bound, что приводит к проблемам с сетью



Доставка кода к данным

- Hadoop сближает вычислительный процессор и данные
- – Код копируется к данным (небольшой расход, Кб)
- – Исполнитель выполняет код на локально расположенных данных



Отказы оборудования

- Чем больше количество машин, тем чаще будут отказы железа
  - – На больших кластерах отказы бывают несколько раз в день
- • Hadoop разрабатывался с учетом отказов железа
  - – Репликация данных
  - – Перезапуск тасков

Экосистема

- Изначально Hadoop был известен в основном из-за двух ключевых компонент:
  - – HDFS: Hadoop Distributed FileSystem
    – MapReduce: Фреймворк распределенной обработки данных
- Сейчас, в дополнении к этому, также известны следующие продукты:
  - – Hbase: Column-oriented DB, поддержка последовательного и произвольного чтения, поддержка простых запросов
  - – Zookeeper: Highly-Available Coordination Service
  - – Oozie: Диспетчер задача для Hadoop
  - – Pig: Язык обработки данных и среда выполнения
  - – Hive: Data warehouse с SQL интерфейсом

HDFS

HDFS

- *Hadoop Distributed File System*
- Для пользователя как “*один большой диск*”
- Работает поверх обычных файловых систем
- Основывается на архитектуре *Google’s Filesystem GFS*
- – *research.google.com/archive/gfs-sosp2003.pdf*
- Fault Tolerant
- – Умеет справляться с выходом из строя дисков, машин и т.д.

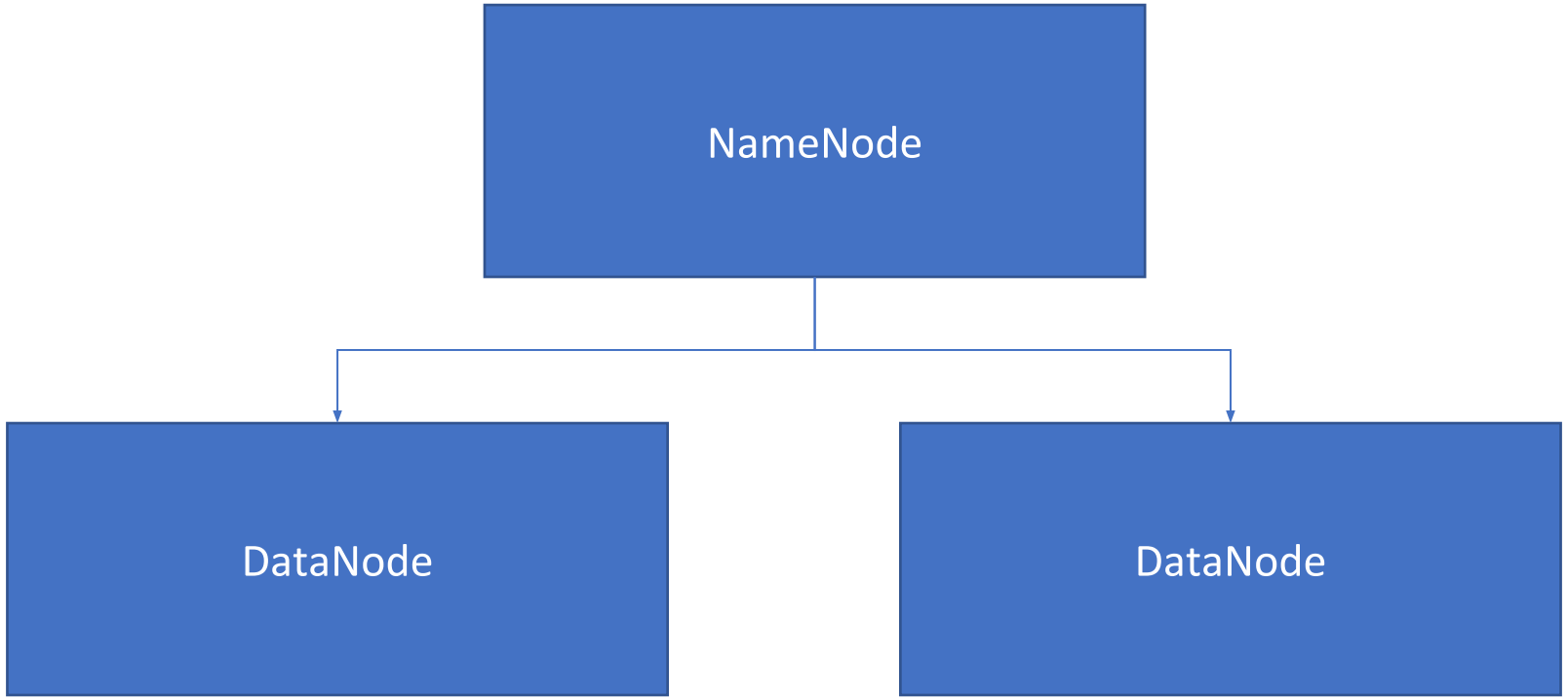
HDFS хорош для

- Хранения больших файлов
  - – Терабайты, петабайты..
  - – Миллионы (но не миллиарды) файлов – Файлы размером от 100 Мб
- Стриминг данных
  - – Паттерн “write once / read-many times”
  - – Оптимизация под последовательное чтение
    - • Нет операциям произвольного чтения
- Обычные сервера

Узлы HDFS

- Для управления файловой системой есть два основных типа узлов
- – Namenode
  - Отвечает за файловое пространство (namespace), мета-ифномрацию и
  - расположение блоков файлов
  - Запускается на выделенной машине (иногда добавляют secondary namenode)
- – Datanode
  - Хранит и отдает блоки данных
  - Отправляет ответы о состоянии на Namenode
  - Запускается обычно на всех машинах кластера

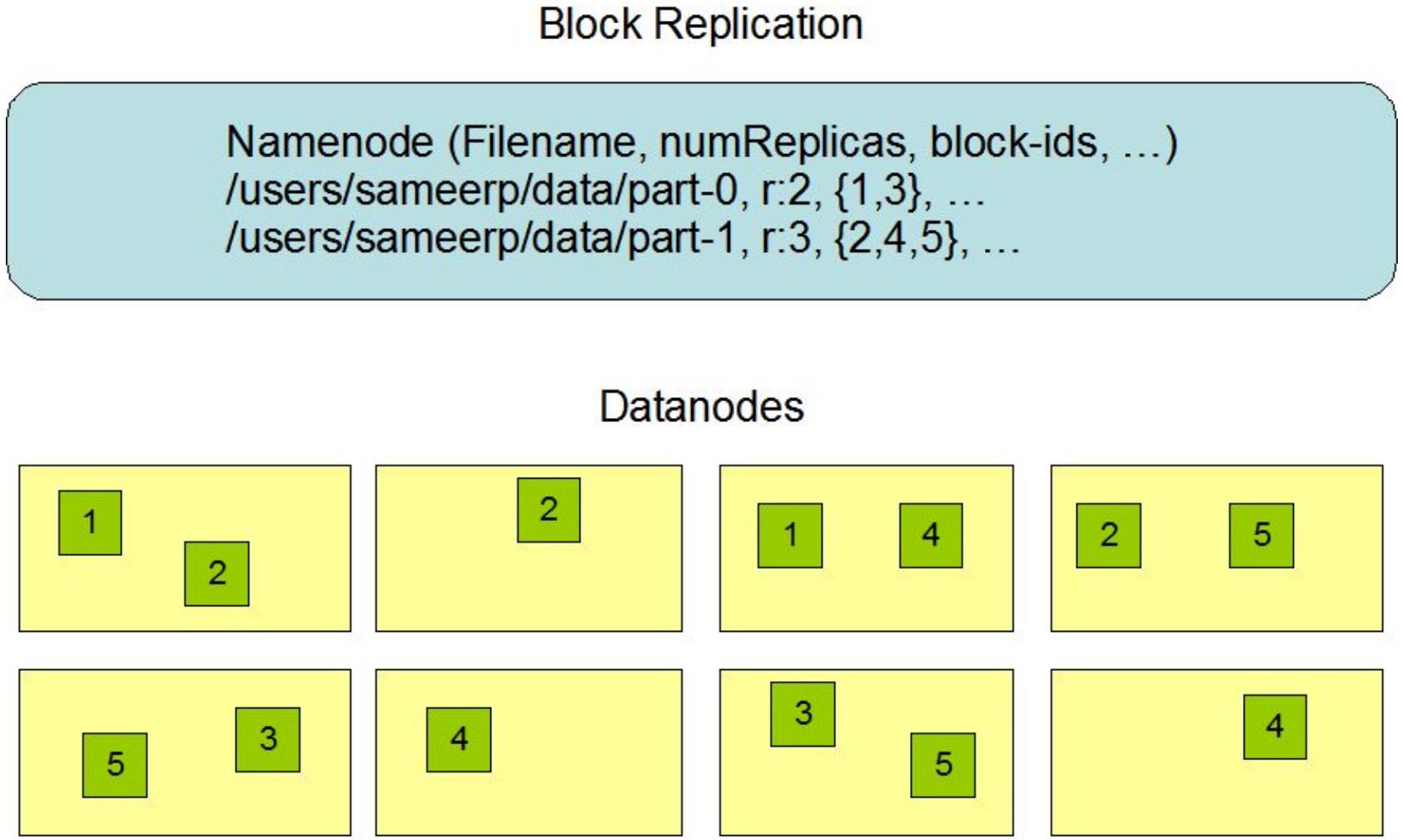
Узлы HDFS



Файлы в HDFS

- Файлы в HDFS состоят из блоков
  - Блок – единица хранения данных
- Управляется через Namenode
- Хранится на Datanode
- Реплицируются по машинам в процессе записи
  - – Один и тот же блок хранится на нескольких Datanode
- Фактор репликации по умолчанию равен **3**

Файлы в HDFS


Block Replication
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

Блоки в HDFS

- Стандартный размер блоков 64Мб или 128Мб
- Основной мотив этого – снизить стоимость *поиска* по сравнению со скоростью передачи данных

Репликация блоков

- Namenode определяет, куда копировать реплики блоков
- Размещение блоков зависит от того, в какой стойке стоит сервер
  - – Баланс между надежностью и производительностью
  - • Попытка снизить нагрузку на сеть (*bandwidth*)
  - • Попытка улучшить надежность путем размещения реплик в разных стойках
  - – Фактор репликации по умолчанию равен 3
    - • 1-я реплика на локальную машину
      • 2-я реплика на другую машину из той же стойки
    - • 3-я реплика на машину из другой стойки

Принцип работы

- Namenode не выполняет непосредственно операций чтения/записи данных
  - – Это одна из причин масштабируемости Hadoop
- Клиент обращается к Namenode для получения информации о размещении блоков для чтения/записи
- Клиент взаимодействует напрямую с Datanode для чтения/записи данных

Принцип работы

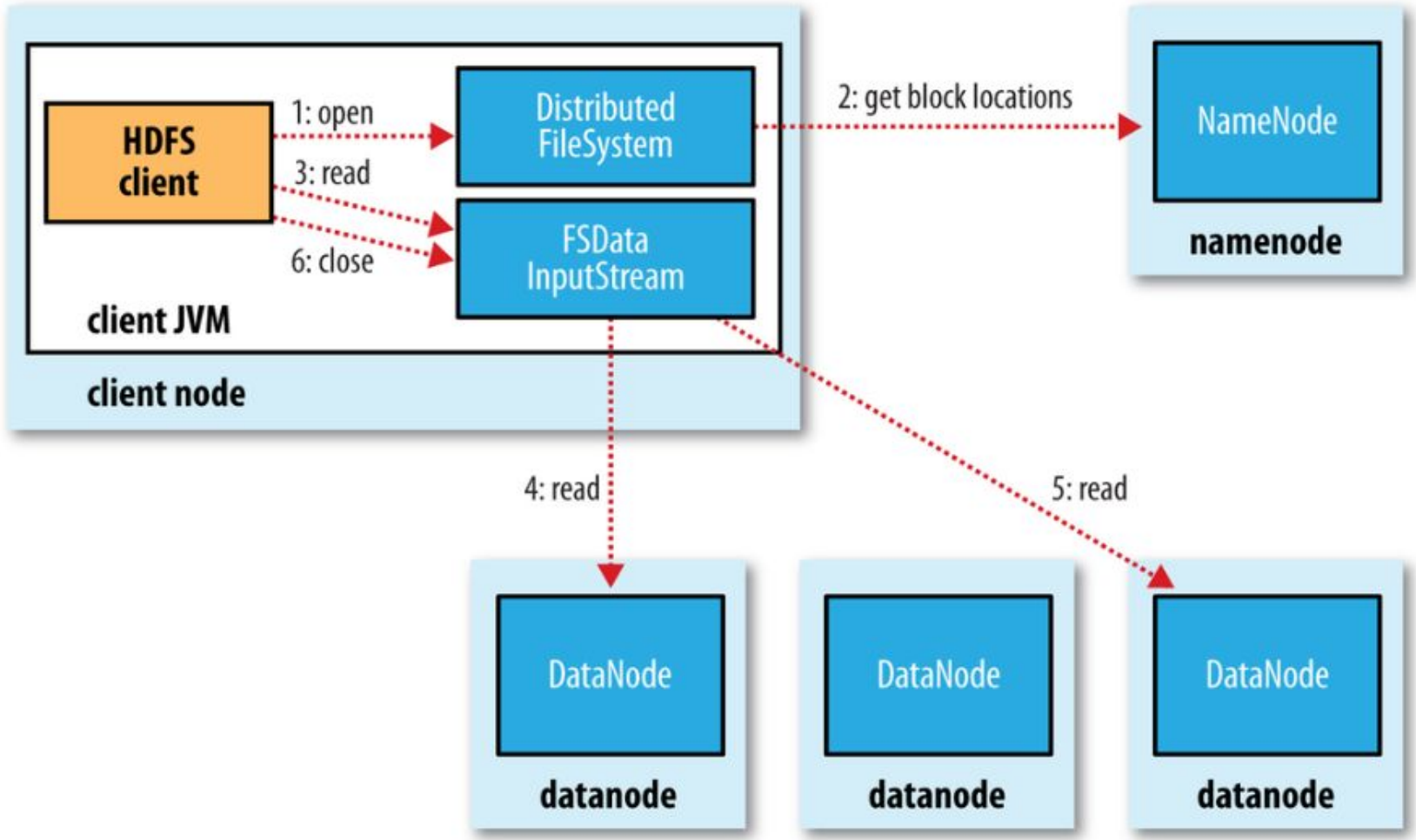
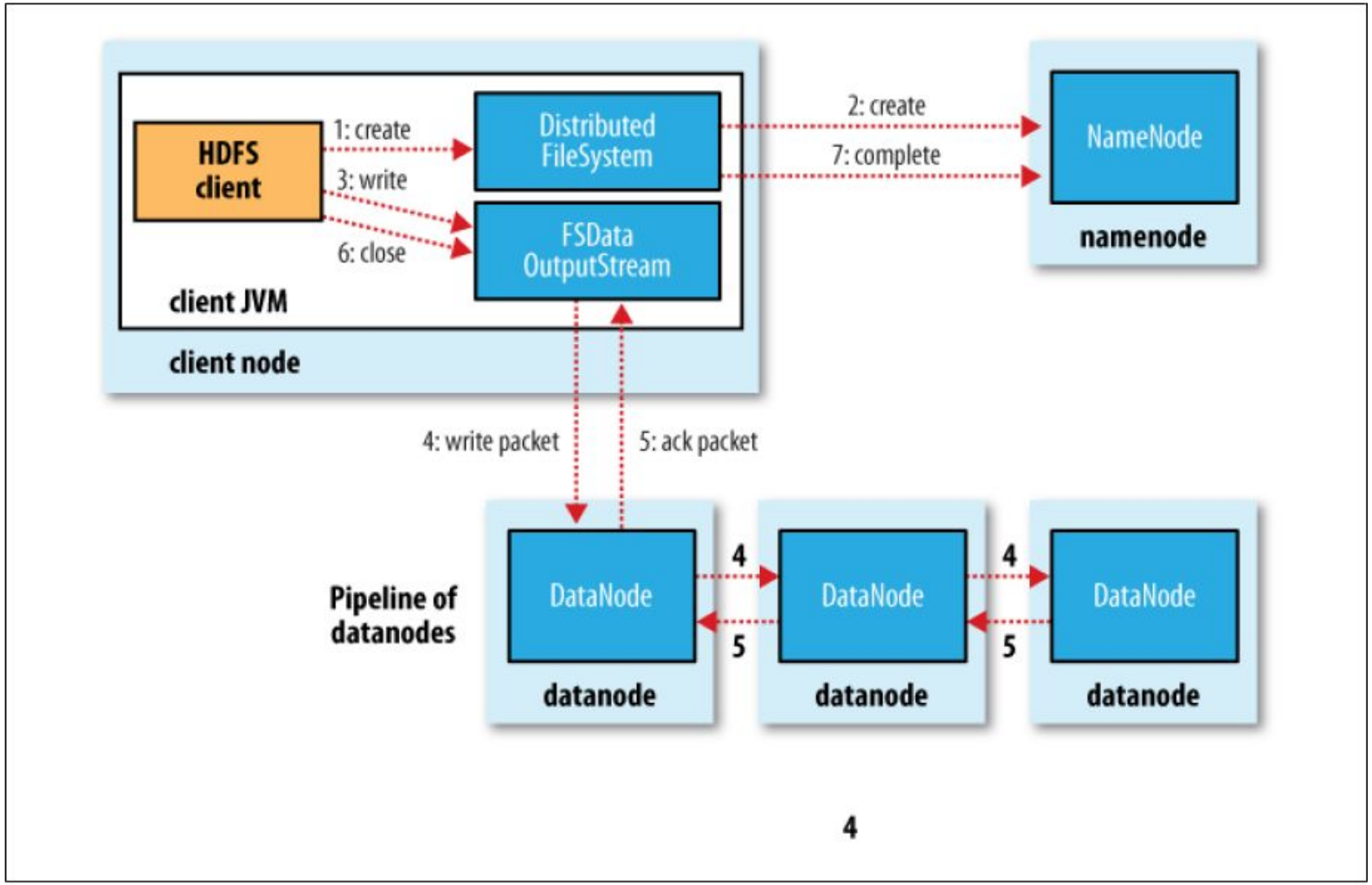


Figure 3-2. A client reading data from HDFS

Запись файла

1. Запрос на запись
2. Создаем файл на NN и определяем расположение блоков
3. Начинаем запись
4. Записываем файлы по все ноды
5. Получаем подтверждение
6. Закрываем файл
7. Отправлем подтверждение на NN



NameNode

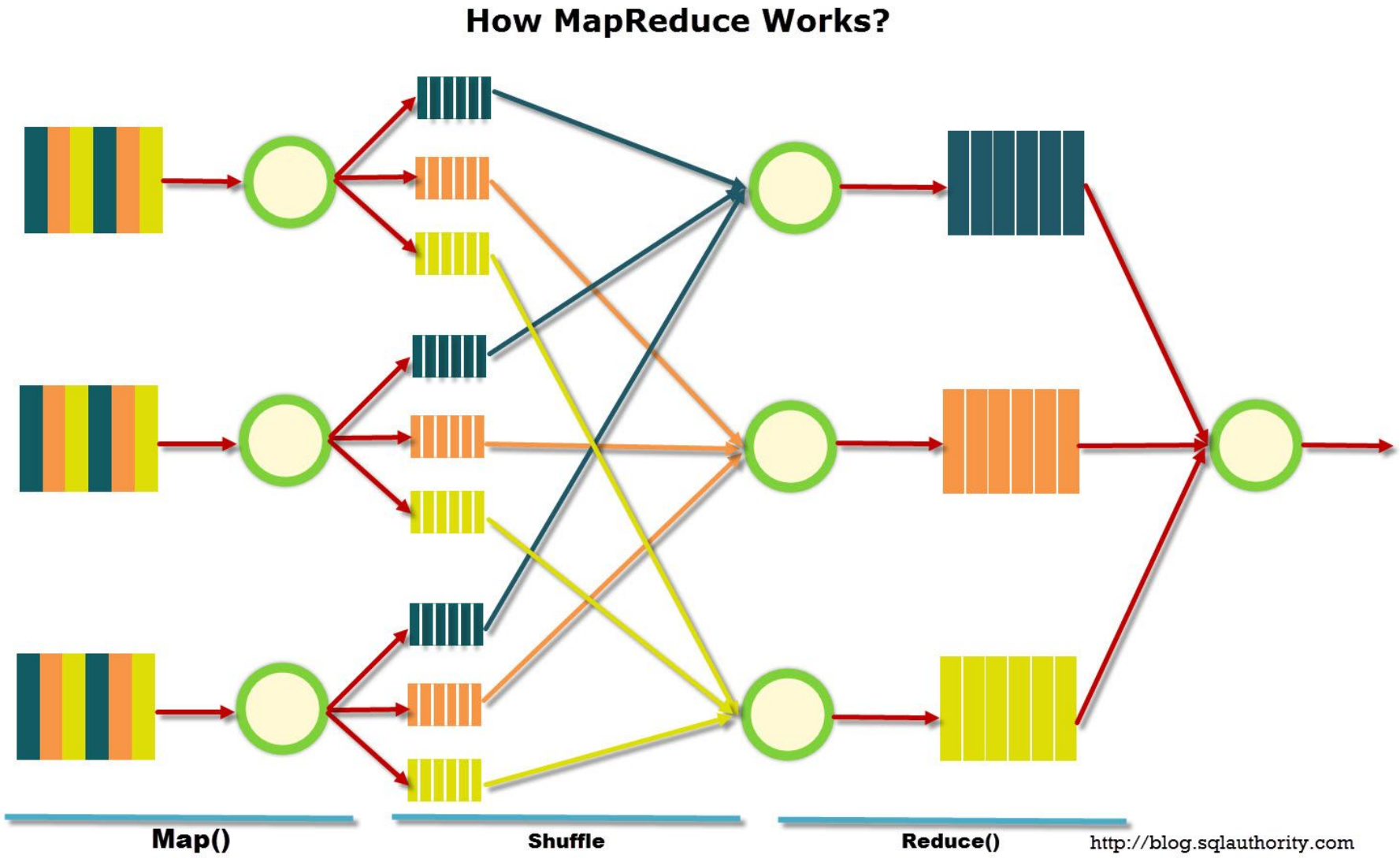
- Для быстрого доступа вся мета-информация о блоках хранится в ОЗУ Namenode
  - – Чем больше кластер, тем больше ОЗУ требуется
  - Лучше миллионы больших файлов (сотни мегабайт), чем миллиарды маленьких
  - Работает на кластерах из сотен машин
- Hadoop 2+
  - Namenode Federation
  - Каждая Namenode управляет частью блоков
  - Горизонтальное масштабирование Namenode
  - Поддержка кластеров из тысячи машин

MapReduce

MapReduce

- Самая известная парадигма обработки больших данных
- Предложена компанией Google в 2004 году
- Имеет множество имплементаций (в том числе open source)

Схема работы



MapReduce

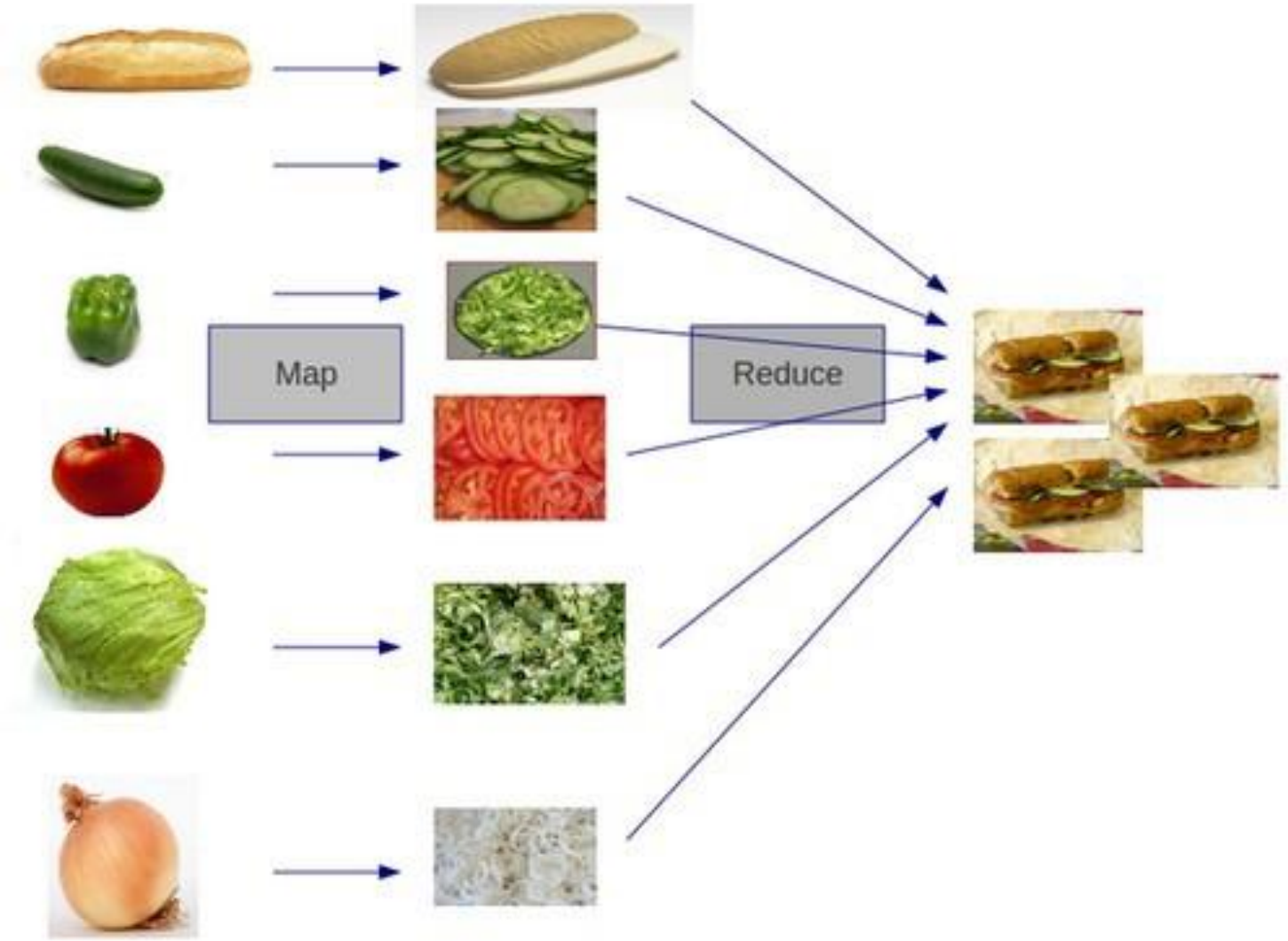
- **Map**-шаг
  - Вход – исходный объект
  - Выход – множество пар ключ-значение
- **Shuffle** – данные сортируются по ключу
- и распределяются по редьюсерам
- **Reduce-**шаг
  - Вход – Ключ -> список значений
  - Выход – Ключ –> значение

Задача шага **map** – предобработать данные и выделить в них **ключ** - признак по которому данные будут агрегированы

Задача шага **shuffle** разложить данные по “корзинам” в соответствии с ключом

Данные обладающие одним ключом обрабатываются вместе.

Sandwich MapReduce



Map-шаг: порезать продукты на кусочки.

Shuffle – разобрать порезанные продукты по корзинкам (2 половинки хлеба, 3 кусочка огурца, 2 помидора и тд)

Reduce-по каждой из корзинок собрать готовый сэндвич.

Word Count

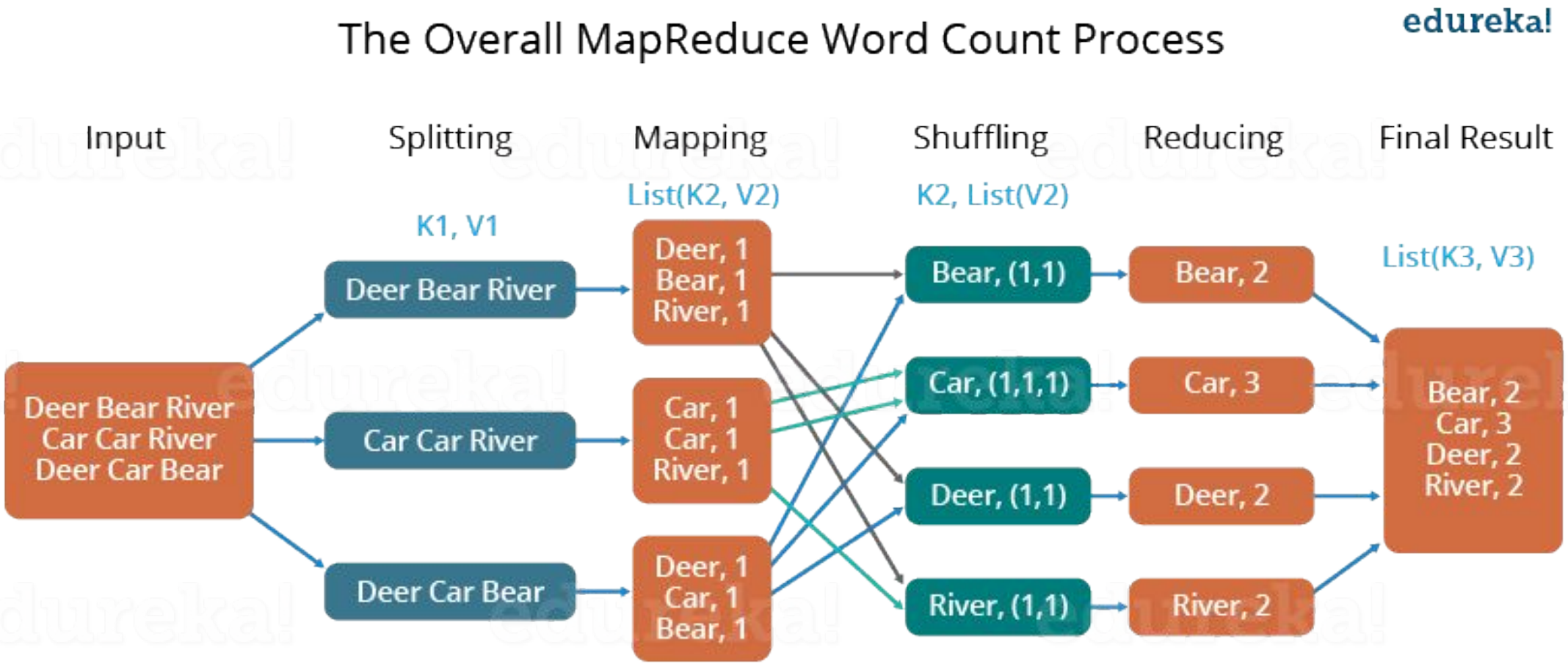
- Дано – файл со строками.
  - Одна строка = 1 документ
- Посчитать:
  - Сколько раз встречается каждое слово в исходном файле

Решение

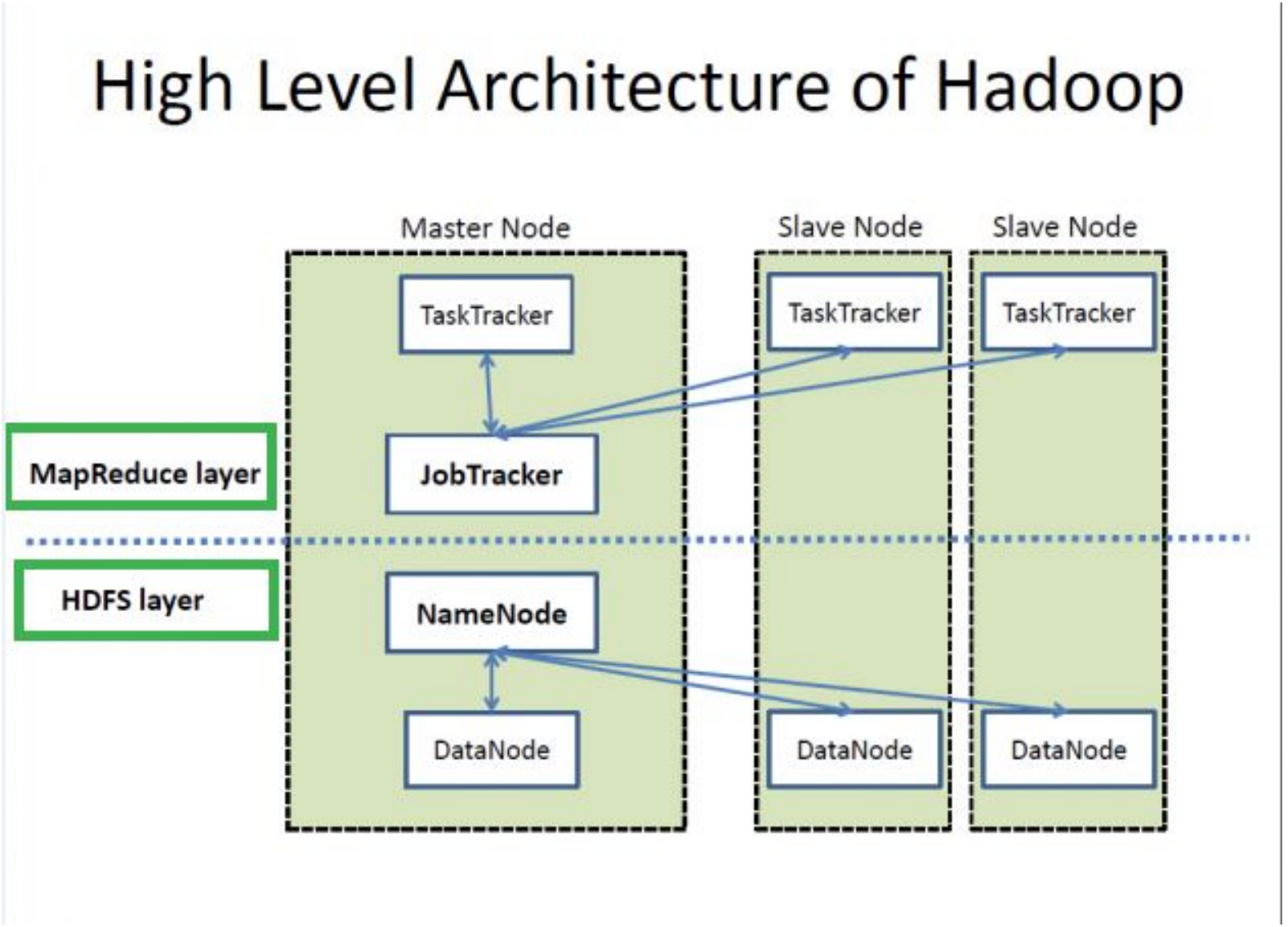
```
def map(string):
    for word in string.split():
        return (word , 1)

def reduce(word, list_of_one):
    return (word, sum(list_of_one)
```

Shuffle шаг реализуется фреймворком и не задается пользователем

WordCount


The Overall MapReduce Word Count Process
edureka!
Input
Splitting
Mapping
Shuffling
Reducing
Final Result
K1, V1
List(K2, V2)
K2, List(V2)
List(K3, V3)
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, (1,1)
Car, (1,1,1)
Deer, (1,1)
River, (1,1)
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

MapReduce & HDFS



MapReduce “Runtime”

- Управление запуском задач
  - – Назначает воркерам таски *map* или *reduce*
- Управление “*data distribution*”
  - – Перемещает код к данным
- – Запускает таски (по возможности) локально с данными
- • Управление синхронизацией
  - – Собирает, сортирует и объединяет промежуточные данные
- Управление обработкой ошибкой и отказов
  - – Определяет отказ воркера и перезапускает таск
- • Все работает поверх распределенной FS

MapReduce

- MapReduce работает поверх демонов
  - • ***JobTracker*** и ***TaskTracker***
- • ***JobTracker***
  - – Управляет запуском тасков и определяет, на каком ***TaskTracker*** таск будет запущен
  - – Управляет процессом работы MapReduce задач (*jobs*)
  - – Мониторит прогресс выполнения задач
  - – Перезапускает зафейленные или медленные таски
- • MapReduce имеет систему ресурсов основанную на слотах (*slots*)
  - – На каждом ***TaskTracker*** определяется, сколько будет запущено слотов
  - – Таск запускается в одном слоте
  - – M мапперов + R редьюсеров = N слотов

Spark

Мотивация

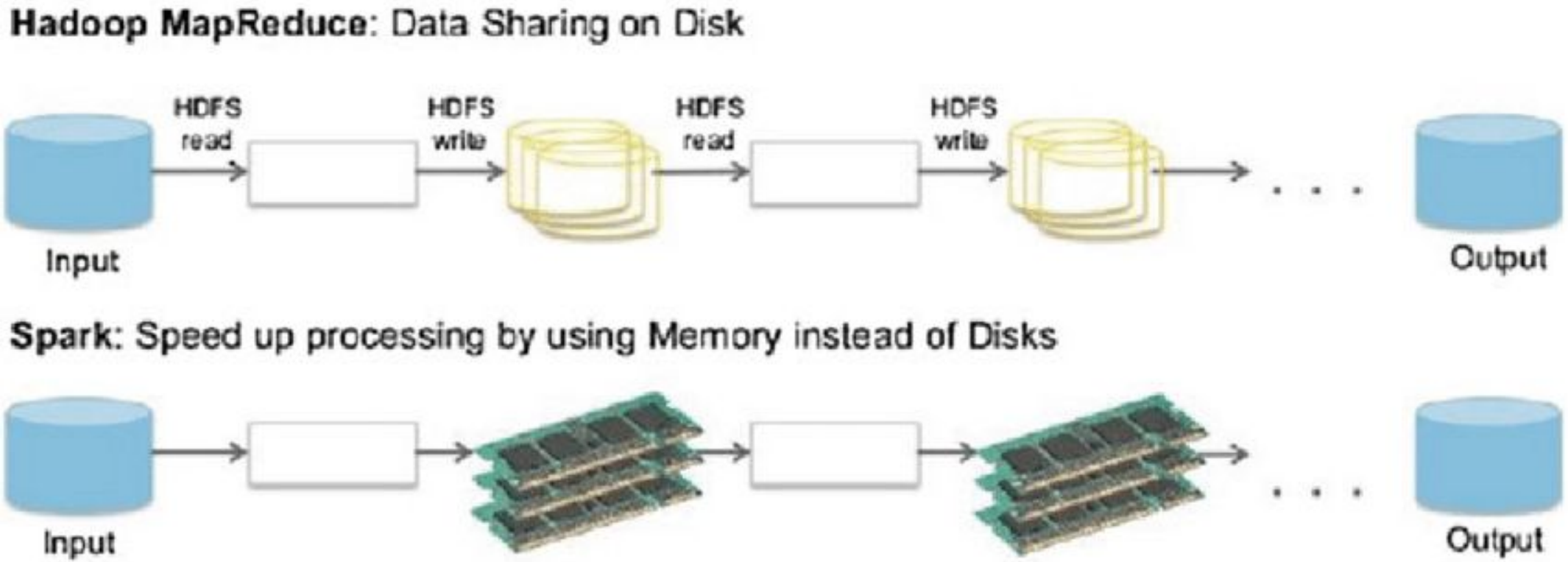
- *MapReduce* отлично упрощает анализ ***big data*** на больших, но ненадежных, кластерах
- Но с ростом популярности фреймворка пользователи хотят большего:
  - – **Итеративных** задач, например, алгоритмы machine learning
  - – **Интерактивной** аналитики

Мотивация

- Для решения обоих типов проблем требуются одна вещь, которой нет в MapReduce...
  - – Эффективных примитивов для общих данных (*Efficient primitives for data sharing*)
- В MapReduce единственный способ для обмена данными между задачами (*jobs*), это надежное хранилище (*stable storage*)
- Репликациятакжезамедляетсистему,ноэто необходимо для обеспечения *fault tolerance*

Решение

- Обработка и разделение данных в памяти (RAM)



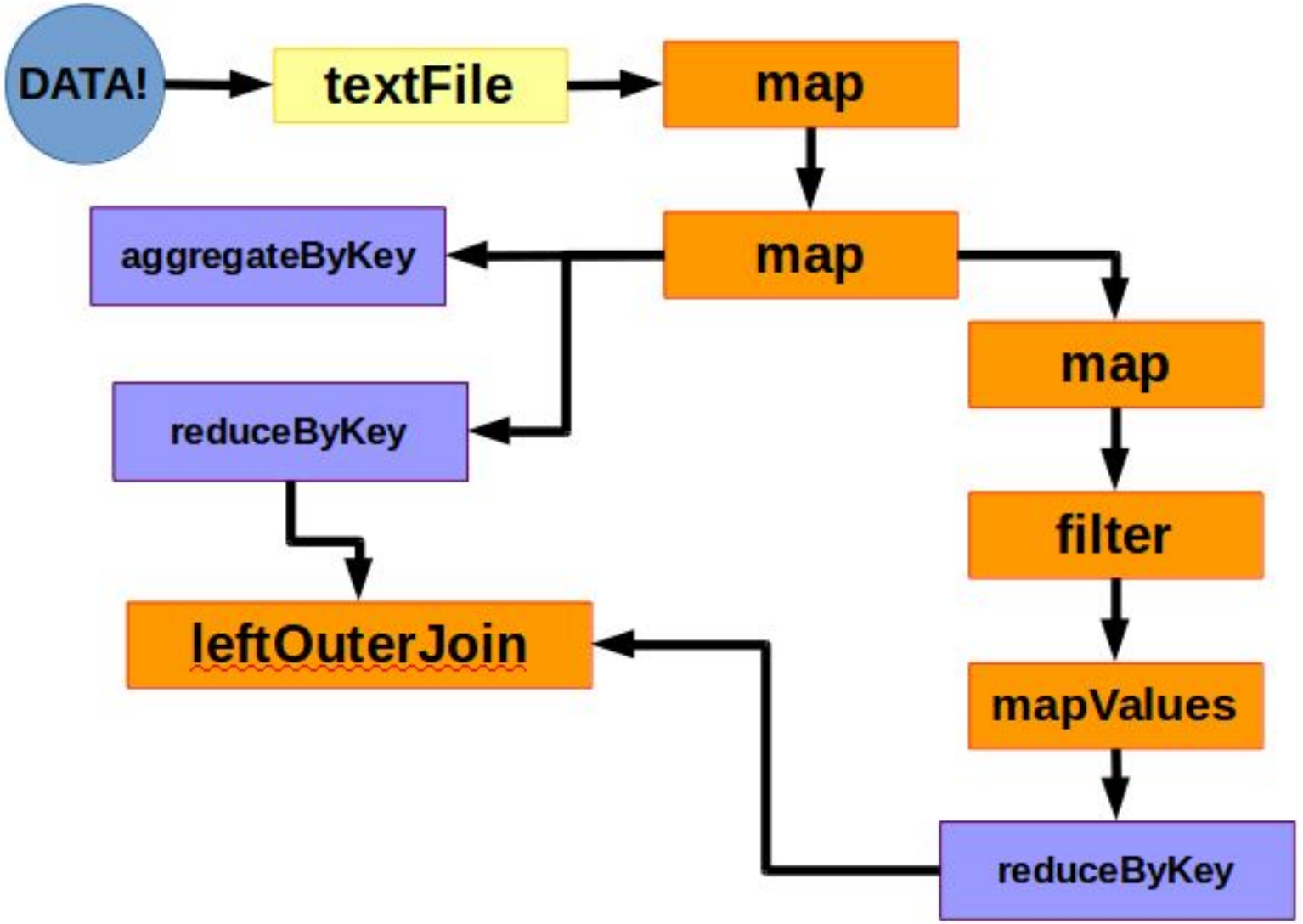
Resilient Distributed Datasets (RDD)

- Задача - Разработать дизайн абстракции распределенной памяти с поддержкой **fault tolerant** и **эффективности**
- Абстрактное представление распределенной RAM
- Immutable коллекция объектов распределенных по всему кластеру
- RDD делится на партиции, которые являются атомарными частями информации
- Партиции RDD могут хранится на различных нодах кластера

Программная модель

- Основана на **parallelizable operators**
- Эти операторы являются функциями высокого порядка, которые выполняют *user- defined* функции параллельно
- Поток обработки данных состоит из любого числа **data sources, operators** и **data sinks** путем соединения их *inputs* и *outputs*

Программная модель



Основные операторы

- Существует два типа операторов – **transformations** и **actions**
- **Transformations:**
  - lazy-операторы, которые создают новые RDD
- **Actions:**
  - запускают вычисления и возвращают результат в программу или пишут данные во внешнее хранилище

Операторы

Transformations	*map*(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

Контекст

- Основная точка входа для работы со Spark
- Доступнав*shell*какпеременная**sc**
- В *standalone*-программах необходимо создавать отдельно

Shared Variable

- Когда Spark запускает выполнение функции параллельно как набор тасков на различных нодах, то отправляется копия каждой переменной, используемой в функции, на каждый таск
- Иногда нужно, чтобы переменная была общая между тасками или между тасками программой-драйвером

Shared Variable

- Обновления переменных нераспространяются обратно в вызывающую
- Использование обычных *read-write* общих переменные между тасками неэффективно
  - – К примеру, необходимо отправить на каждую ноду большой датасет
- Есть два типа *shared variables*
  - – **broadcast variables**
  - – **accumulators**

BroadCast Variables

- Read-only переменные **кешируются** на каждой машине вместо того, чтобы отправлять копию на каждый таск
- *Broadcast Variables* не отсылаются на ноду больше **одного раза**

Accumulators

- Могут быть только **добавлены**
- Могут использоваться для реализации **счетчиков** и **сумматоров**
- Таски, работающие на кластере, могут затем добавлять значение используя оператор **+=**

Еще в Spark

- MLLib – библиотека машинного обучения для spark.
- graphx – библиотека работы с графами для apache spark
- SparkSQL – библиотека для трансляции SQL-запросов (альтернатива HIVE)
- Spark streaming – библиотека обработки потоковых данных

Другие инструменты Hadoop

Стек Hadoop

- **Hadoop-Common** - общие библиотеки hadoop
- **HDFS** – распределенная файловая система
- **Yarn** - Фреймвок управления распределенными задачами
- **Hadoop MapReduce** – Yarn-based реализация mapreduce

- Множество других hadoop-related проектов
  - Hbase, hive, pig, zookeeper, spark, cassandra …. (тысячи их 😊)

Apache Hive

- Транслятор SQL-like запросов в цепочки MapReduce запросов
- Очень сильно упрощает жизнь при разработке big data проектов
- Часто используется в enterprise решениях

Hbase

- Колоночная база данных
- 3х-мерная, ключ-значение, timestamp
- Позволяет объединить достоинства массивной обработки данных и записи данных по ключу.

Сборки Hadoop

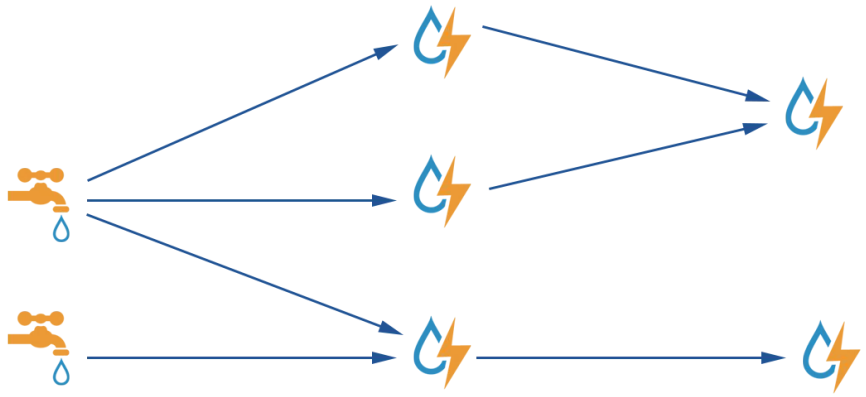
Несколько основных поставщиков Hadoop:

- Cloudera
- Hortonworks
- MapR

Потоковая обработка данных

- MapReduce – оффлайн алгоритм пакетной обработки данных
- Не подходит для задач где данные поступают быстро и на них необходимо быстро реагировать
- Примеры
  - Автомобильные пробки
  - Поиск мошеннических операций в банке

Потоковая обработка данных



Полезные материалы

Полезные материалы

- Hadoop: The Definitive Guide, Tom White
- Hadoop in Action, Chuk Lam

ИТОГИ ЗАНЯТИЯ

Что мы сегодня узнали

- Технологии больших данных нужны только если данных действительно много
- MapReduce – хороший подход для организации процессинга больших данных
- Hadoop – opensource проект, лидер на рынке больших данных
- Spark – расширяет и ускоряет Hadoop
- Установить Hadoop можно из одной из сборок
- Не для любой задачи подходит MapReduce

ВОПРОСЫ

Домашнее задание

Домашнее задание

- Прочитать про операторы Spark. Прислать ответы на вопросы
- Какие команды отвечают за:
  - Сохранение результата в текстовый файл (Это Action или Transformation?)
  - Как получить первые n-элементов массива (Это Action или Transformation?)
  - Объединить два RDD в один (Это Action или Transformation?)
  - В чем разница между Reduce и CoGroup-операторами (Это Action или Transformation?)
- Нарисовать DAG для Spark’а для подсчета количества уникальных слов в файле

НЕТОЛОГИЯ
групп

Спасибо за внимание!

Алексей Кузьмин  aleksej.kyzmin@gmail.com