ТЕХНОСФЕРА

Методы распределенной обработки больших объемов данных в Hadoop

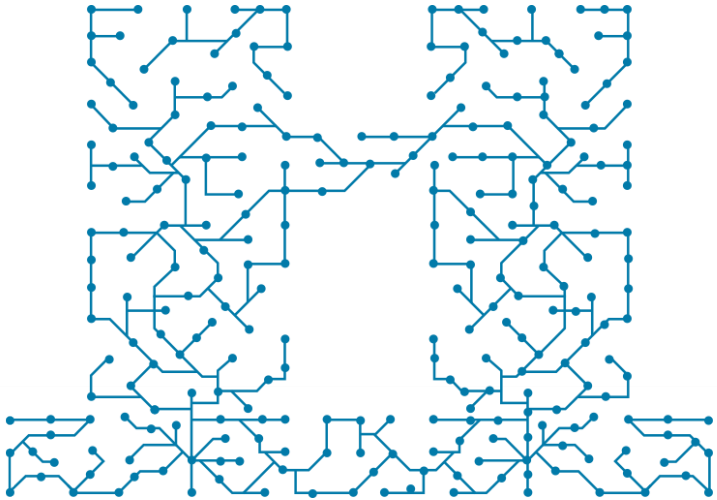
Лекция 1: Введение в BigData и MapReduce

Москва 2014

АЛЕКСЕЙ РОМАНЕНКО

Руководитель команды разработки поискового робота в проекте Поиск@Mail.Ru. Участвовал в проектировании и реализации архитектуры системы хранения и обработки данных на платформе Hadoop в проекте Поиск@Mail.Ru.

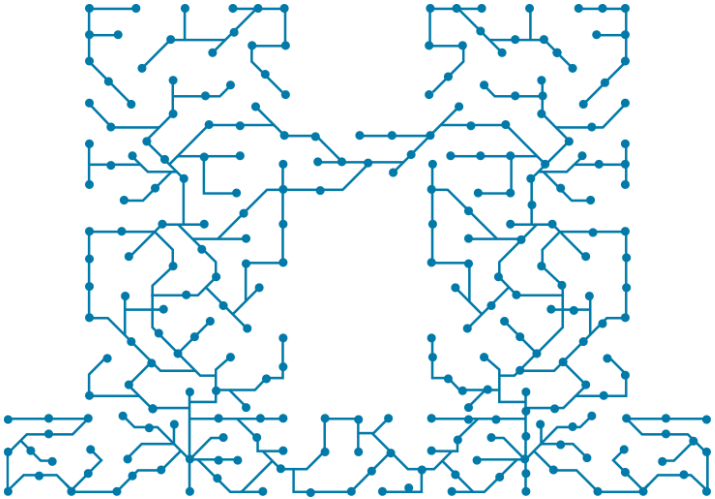
Выпускник факультета Радиоэлектроники МАИ в 2001 г, кафедра Вычислительные системы и сети.



МИХАИЛ ФИРУЛИК

Руководитель отдела анализа данных, департамент рекламных технологий, компании Mail.Ru Group. Ранее занимался медийными рекламными проектами в компании Яндекс, в том числе «Крипта» и «Поведенческий ретаргетинг», а так же развитием автоматических стратегий управления рекламными кампаниями в «Директ». До этого долгое время работал ведущим разработчиком системы обработки данных проекта исследования телевизионной аудитории «TV-Index» в компании TNS.

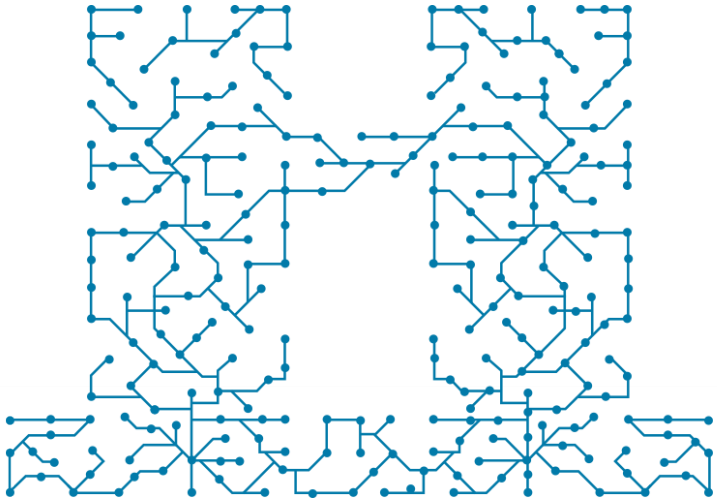
Выпускник Ленинградского Высшего Военного Училища Связи им. Ленсовета (ныне Академии Связи), кафедра Математическое Обеспечение Автоматизированных Систем Управления, 1995 г



НИКОЛАЙ АНОХИН

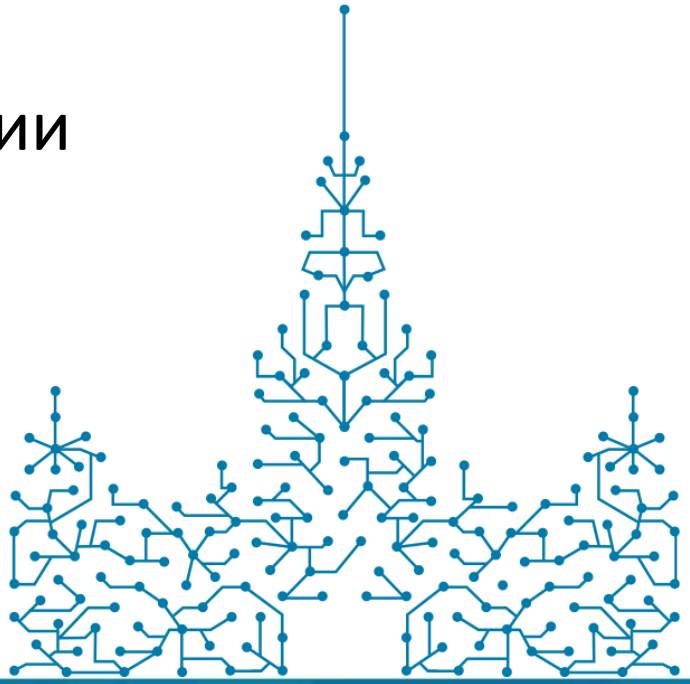
Программист-исследователь в отделе анализа данных Mail.ru. Работаю над применением алгоритмов машинного обучения к задачам оптимизации рекламного трафика. До Mail.ru занимался Data Mining в компаниях Technicolor и AdMoment.

Окончил МФТИ по специальности «Прикладная Физика и Математика» в 2010 году и Университет Лиона по специальности «Data Mining & Knowledge Management» в 2012.



Чему посвящен данный курс?

- Вычисления на больших объемах данных (“Big Data”)
- Основной фокус на приложениях и дизайне алгоритмов
- MapReduce… и другие технологии



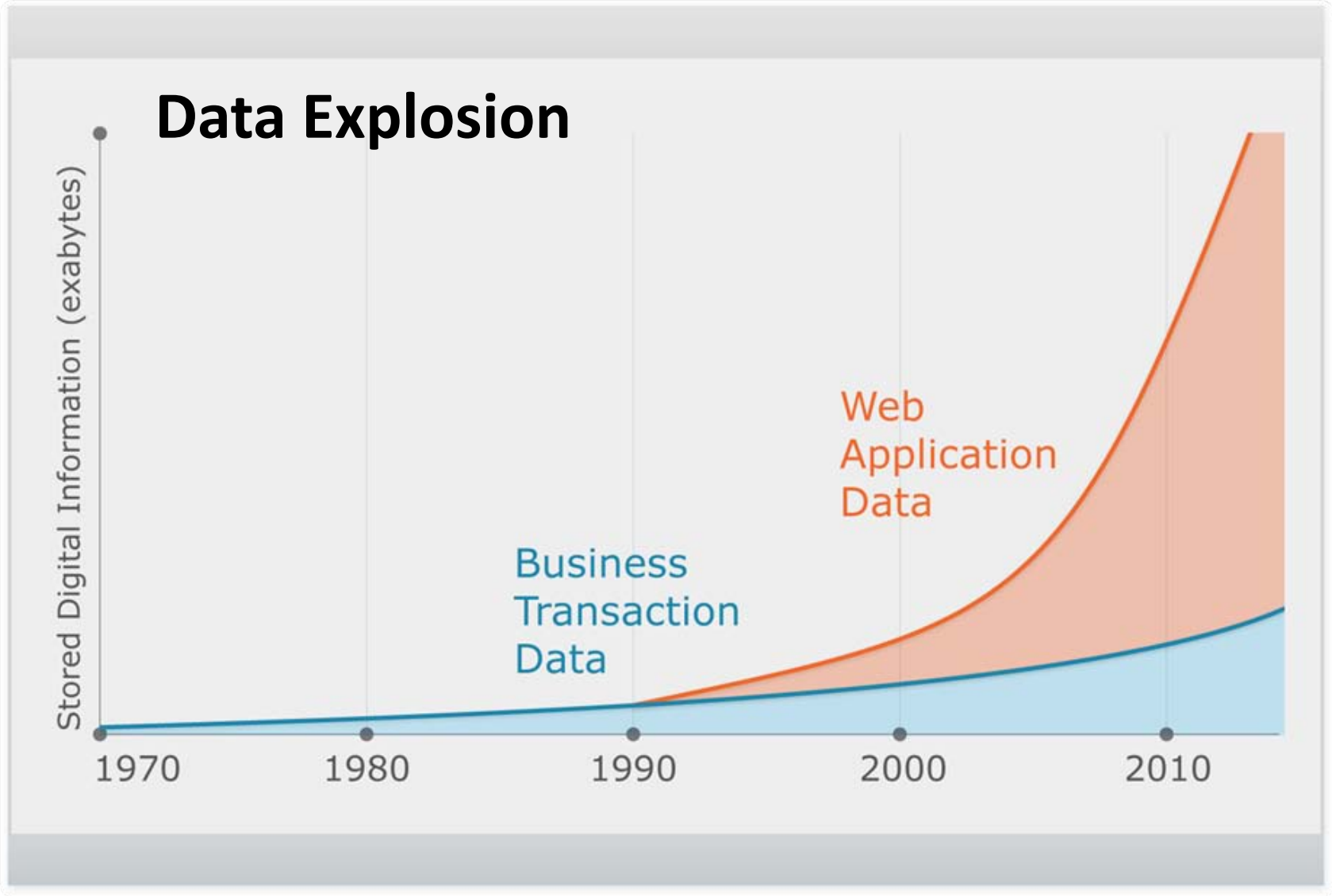
Расписание занятий

Дата	Время	Лекция
13 сентября, суббота	15:00 — 18:00	Вводная лекция про BigData, введение про MapReduce
19 сентября, пятница	18:00 — 21:00	Hadoop, основы
26 сентября, пятница	18:00 — 21:00	Распределенная файловая система HDFS
3 октября, пятница	18:00 — 21:00	MapReduce в Hadoop, введение
10 октября, пятница	18:00 — 21:00	MapReduce в Hadoop, продолжение I
17 октября, пятница	18:00 — 21:00	MapReduce в Hadoop, продолжение II
24 октября, пятница	18:00 — 21:00	Введение в Hive и Pig
31 октября, пятница	18:00 — 21:00	NoSQL базы данных: BigTable, Hbase и Cassandra

Расписание занятий, продолжение

Дата	Время	Лекция
7 ноября, пятница	18:00 — 21:00	Обработка текстов на естественных языках в Hadoop
14 ноября, пятница	18:00 — 21:00	Machine learning в Hadoop: Mahout
21 ноября, пятница	18:00 — 21:00	Обработка данных realtime: Storm, Spark, Impal
28 ноября, пятница	18:00 — 21:00	Вычислительная модель Pregel
5 декабря, пятница	18:00 — 21:00	Hadoop 2.0
12 декабря, пятница	18:00 — 21:00	Hadoop: примеры использования в реальных проектах
19 декабря, пятница	18:00 — 21:00	Итоговое занятие
16 января, пятница	18:00 — 21:00	Пересдача

Big Data


Data Explosion
Stored Digital Information (exabytes)
Web
Application
Data
Business
Transaction
Data
1970
1980
1990
2000
2010

Обрабатывает 20 PB в день (2008)
Скачивает 20B веб-страниц в день (2012)

>10 PB данных, 75B DB
запросов в день (6/2012)

>100 PB польз. данных +
500 TB/день (8/2012)

S3: 449B объектов, макс 290k rps (7/2011)
1T объектов (6/2012)

JPMorganChase

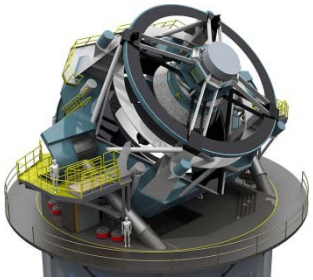
150 PB на 50k+ серверов
работает 15k apps (6/2011)

INTERNET ARCHIVE

Wayback Machine: 240B веб-страниц в архиве, 5 PB (1/2013)

LHC: ~15 PB в год

CERN



LSST: 6-10 PB в год
(~2015)

Много данных – это сколько?

Необходимые знания

- Программирование на Java и/или Python
  - Этот курс не учит программировать
  - Фокус на “thinking at scale” и разработка/дизайн алгоритмов
  - Самостоятельная установка Hadoop (рекомендуется)
- Уметь отлаживать свой код
- Опыт работы в Linux (желательно)

Необходимые знания

- Основные знания по:
  - Теория вероятностей и статистики
  - Дискретная математика
  - Архитектура компьютера
- Не обязателен опыт в:
  - MapReduce
  - Параллельном и распределенном программировании
- Любознательность

Как же стать гуру Hadoop?

- Посещать лекции, делать ДЗ
- Читать книги
  - Tom White, “Hadoop: The Definitive Guide”
    - Есть издание на русском языке
  - Jimmy Lin and Chris Dyer, “Data-Intensive Text Processing with MapReduce”
- RTFM
- RTFC(!)

Этот курс не для вас, если…

- Если вам не очень то интересен данный раздел
- Если у вас нет время, чтобы в этом разобраться
- Если вы чувствуете себя некомфортно с непредсказуемостью, которая всегда есть в bleeding-edge software

Иначе, это должно быть интересно!

Оценка результатов

- Финальная оценка состоит из:
  - Домашних заданий
  - Финального проекта
- Вряд ли это подойдет в качестве отговорок:
  - “Я слишком занят!”
  - “Это требует слишком много времени, чем у меня есть!”
  - “Это труднее, чем я думал!”
  - “Я отформатировал диск с домашней работой!”
  - Ну и т.д.

Использование Hadoop

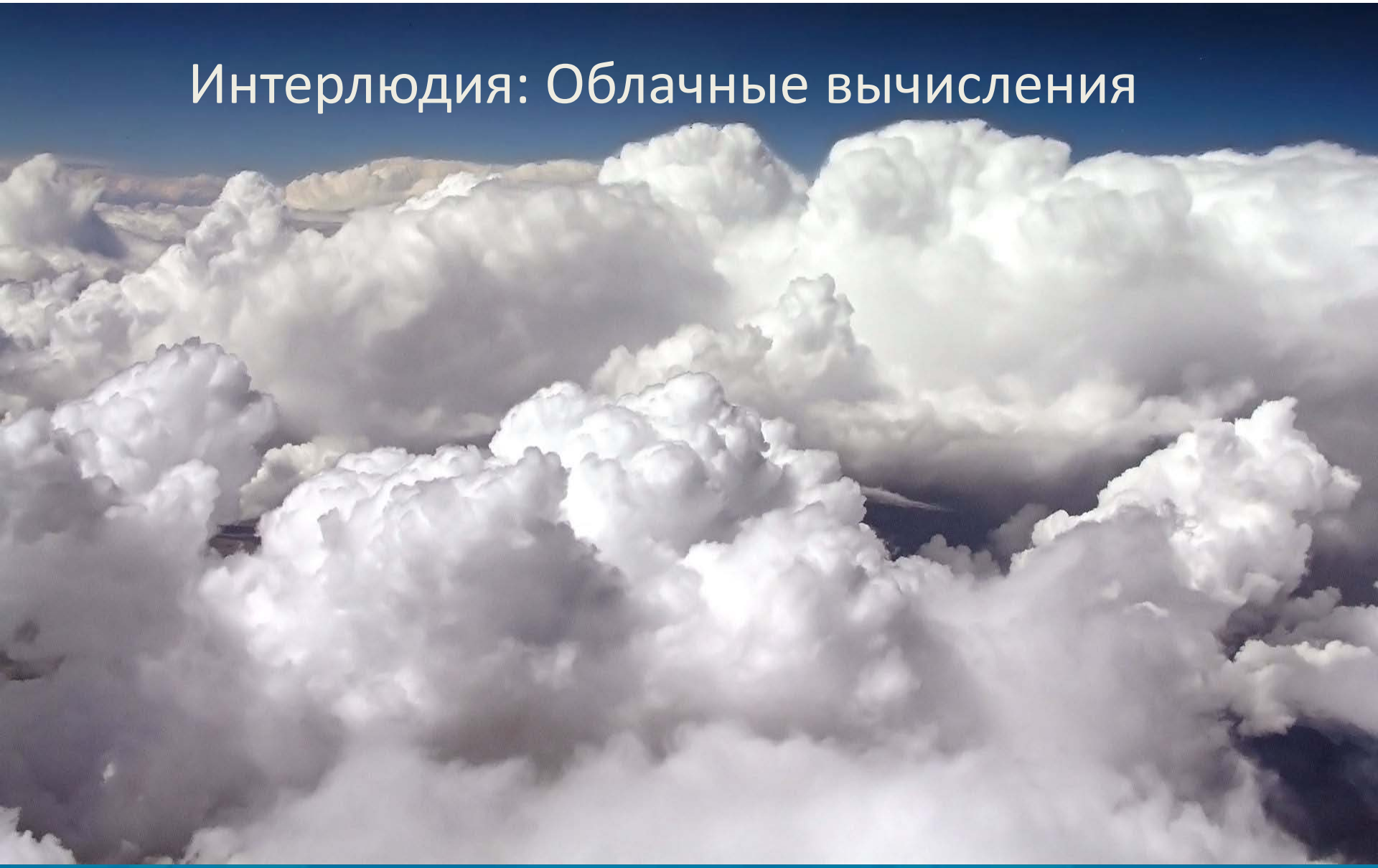
- Hadoop на вашем компьютере
- Hadoop в виртуальной машине на вашем компьютере
- Hadoop в нашей лаборатории

“Hadoop Дзен”

- Это передовая технология (т.е. незрелая!)
  - Долгий путь с 2007, но до сих пор мы в начале пути…
  - Баги, недокументированные “фичи”, неожиданное поведение, потеря данных(!)
- Нельзя разочаровываться
  - Эти WTF! моменты
- Быть терпеливым
  - Мы неизбежно столкнемся со странными “ситуациями” на протяжении курса
- Быть гибким
  - Мы должны быть креативными в поиске workarounds
- Быть конструктивным
  - Не паниковать!

“Hadoop Дзен”

Интерлюдия: Облачные вычисления



“Облака” – это верх технологий?

- До облаков было…
  - Грид-вычисления
    - https://ru.wikipedia.org/wiki/Грид
  - Суперкомпьютеры Connection machine
    - https://ru.wikipedia.org/wiki/Connection_Machine
  - Векторные суперкомпьютеры
  - …
- «Облачные вычисления» означает много различных вещей:
  - Big data
  - Ребрендинг Web 2.0
  - Utility computing
  - Everything as a service

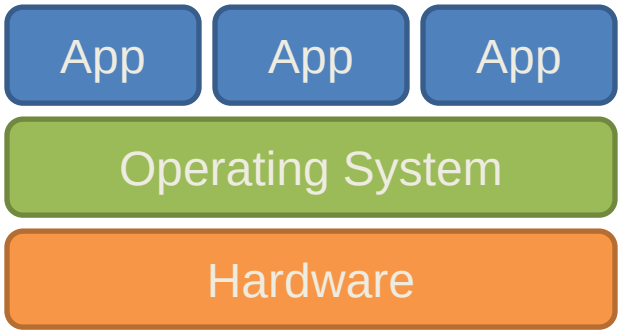
Ребрендинг Web 2.0

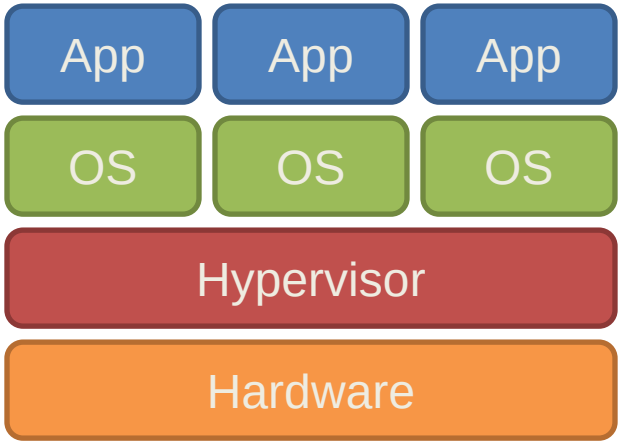
- Интерактивные интернет Rich-приложения
  - Облако - это сервера, на которых они работают
  - AJAX есть де-факто стандарт (хорошо это или плохо…)
  - Примеры: Facebook, YouTube, Gmail, Mail.Ru…
- “Сеть – это большой компьютер”: дайте два!
  - Пользовательские данные хранятся “в облаках”
  - Расцвет эпохи нетбуков, смартфонов, планшетов и т.д..
  - Браузер – это ОС

Utility computing

- Что?
  - Вычислительные ресурсы как сервис (“сколько использовал, столько и оплати”)
  - Возможность динамического предоставления виртуальных машин
- Зачем?
  - Цена: бюджет и оперативные расходы
  - Масштабируемость: “бесконечное” кол-во ресурсов
  - Эластичность: расширение on demand
- Это имеет смысл?
  - Преимущества для пользователей облаков
  - Бизнес для облачных провайдеров

Технология: Виртуализация





Everything as a Service

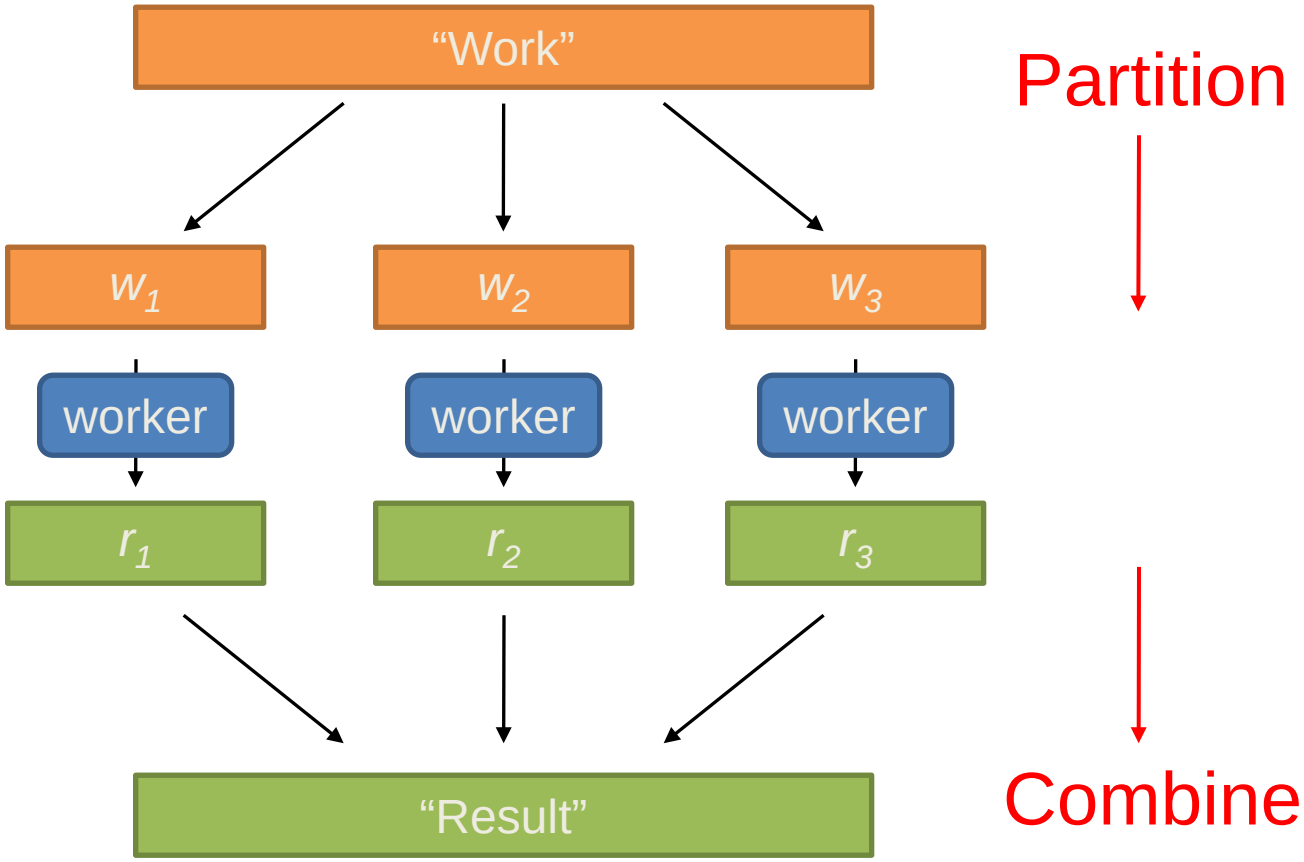
- Utility computing = Infrastructure as a Service (IaaS)
  - Зачем покупать сервера когда их можно арендовать?
  - Пример: Amazon's EC2, Rackspace
- Platform as a Service (PaaS)
  - Дай мне удобное API и позаботься об эксплуатации, апргрейде, расширении, …
  - Пример: Google App Engine
- Software as a Service (SaaS)
  - Just run it for me!
  - Example: Gmail, Salesforce

Зачем все это нужно?

- Готовые для решения Big Data задачи
  - Социальные сети, user-generated контент = Big Data
  - Пример: предложение друзей в Facebook, размещение рекламы Google
  - Business intelligence: собрать все в *data warehouse* и запустить аналитику для понимания выводов
- Utility computing предоставляет:
  - Возможность предоставления Hadoop-кластеров по запросу в облаке
  - Меньший порог входа для решения Big Data проблем
  - Продуктовость и демократизация возможностей работы с Big Data

Работаем с Big Data

Divide and Conquer



Вопросы параллелизма

- Как назначать части work по воркерам?
- А что есть у нас больше частей work чем воркеров?
- А что если воркерам надо обмениваться промежуточными результатами?
- Как мы объединяем промежуточные результаты?
- Как мы узнаем, что все воркеры отработали?
- А что если воркер умрет?

Какая общая проблема объединяет эти вопросы?

Общая проблема

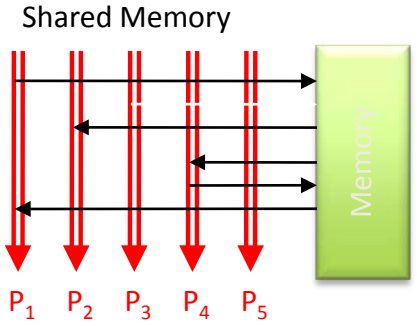
- Проблемы параллелизма возникают из-за:
  - Взаимодействия между воркерами (напр., обмен состояниями)
  - Доступ к общим ресурсам (напр., данные)
- Т.о. нам нужен механизм синхронизации

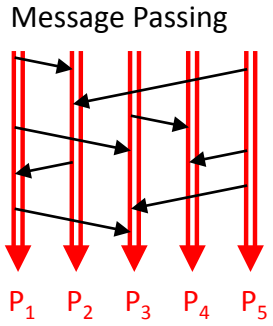
Управление множеством воркеров

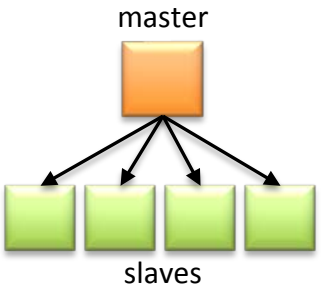
- Трудно потому что:
  - Мы не знаем порядок, в котором воркеры запускаются
  - Мы не знаем, когда воркеры прерывают друг друга
  - Мы не знаем, когда воркерам надо обмениваться промежуточным результатом
  - Мы не знаем порядок, в котором воркеры имеют доступ к общим данным
- Поэтому, нам нужны:
  - Семафоры (lock, unlock)
  - Conditional variables (wait, notify, broadcast)
  - Barriers
- Все равно много проблем:
  - Deadlock, livelock, race conditions...
  - Dining philosophers, sleeping barbers, cigarette smokers...
- Мораль сей басни: будь осторожен!

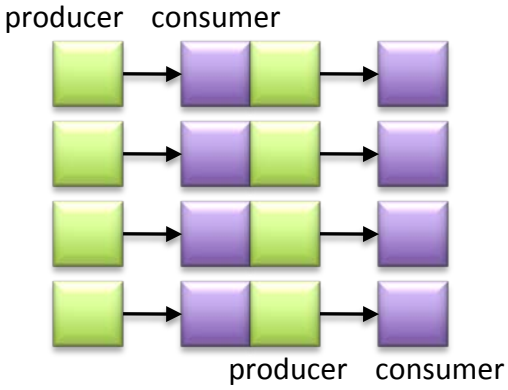
Текущие средства

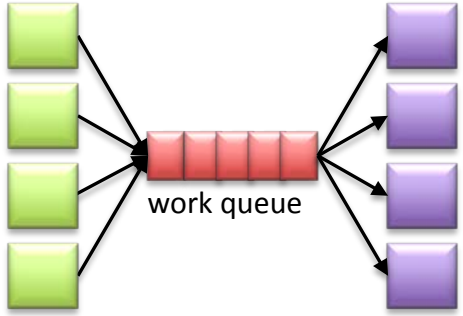
- Программные модели
  - Shared memory (pthreads)
  - Message passing (MPI)
- Design Patterns
  - Master-slaves
  - Producer-consumer flows
  - Shared work queues





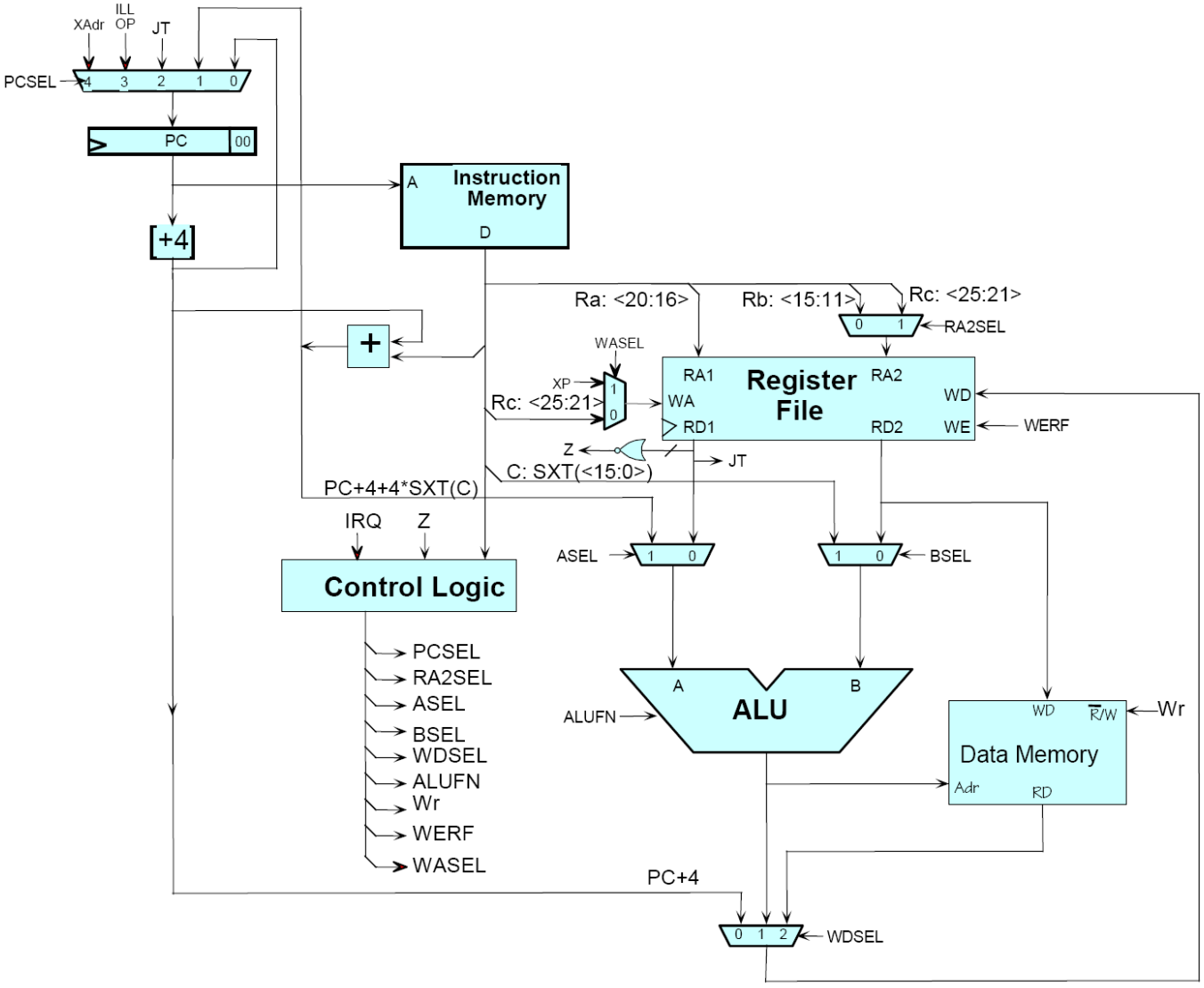


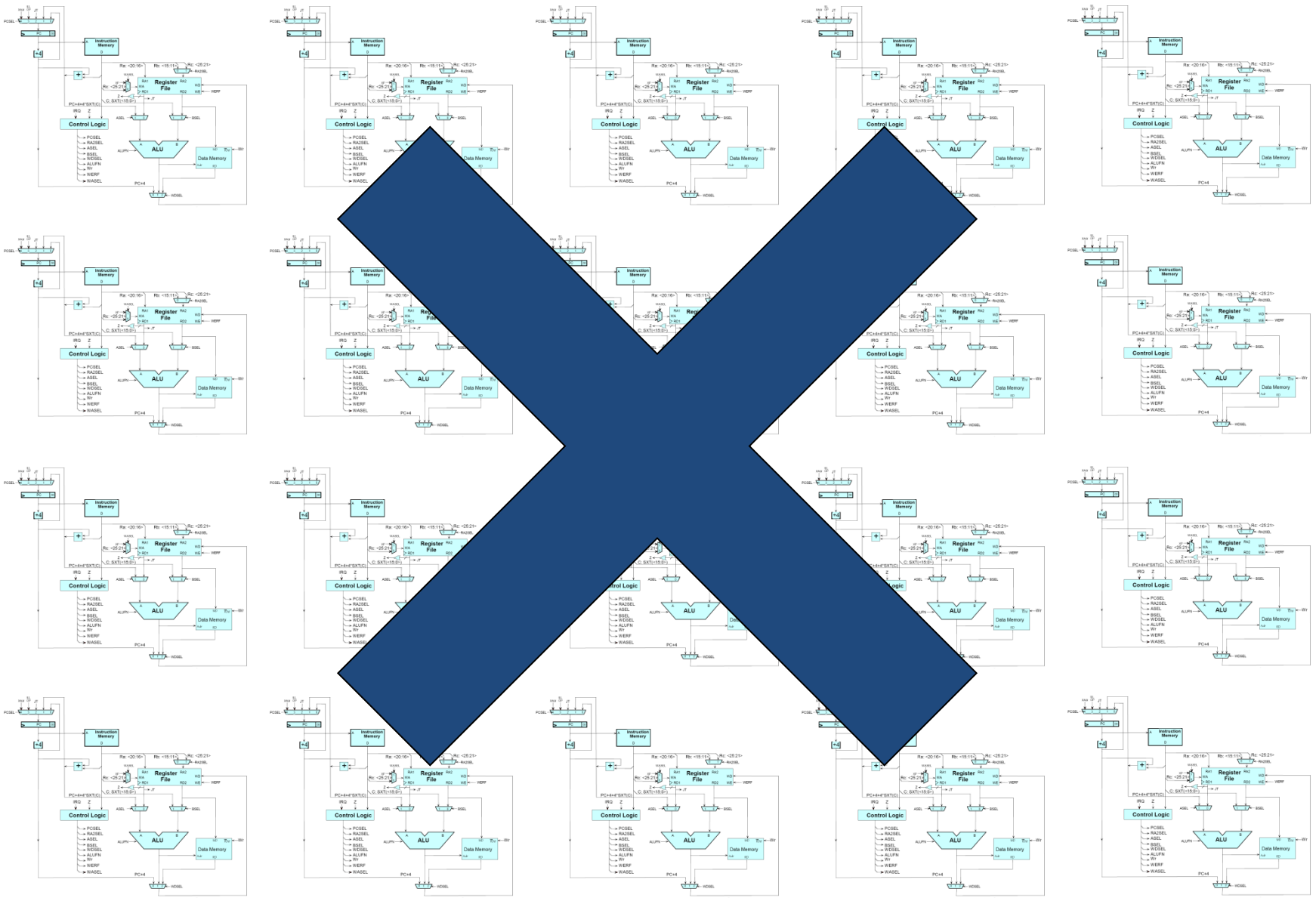


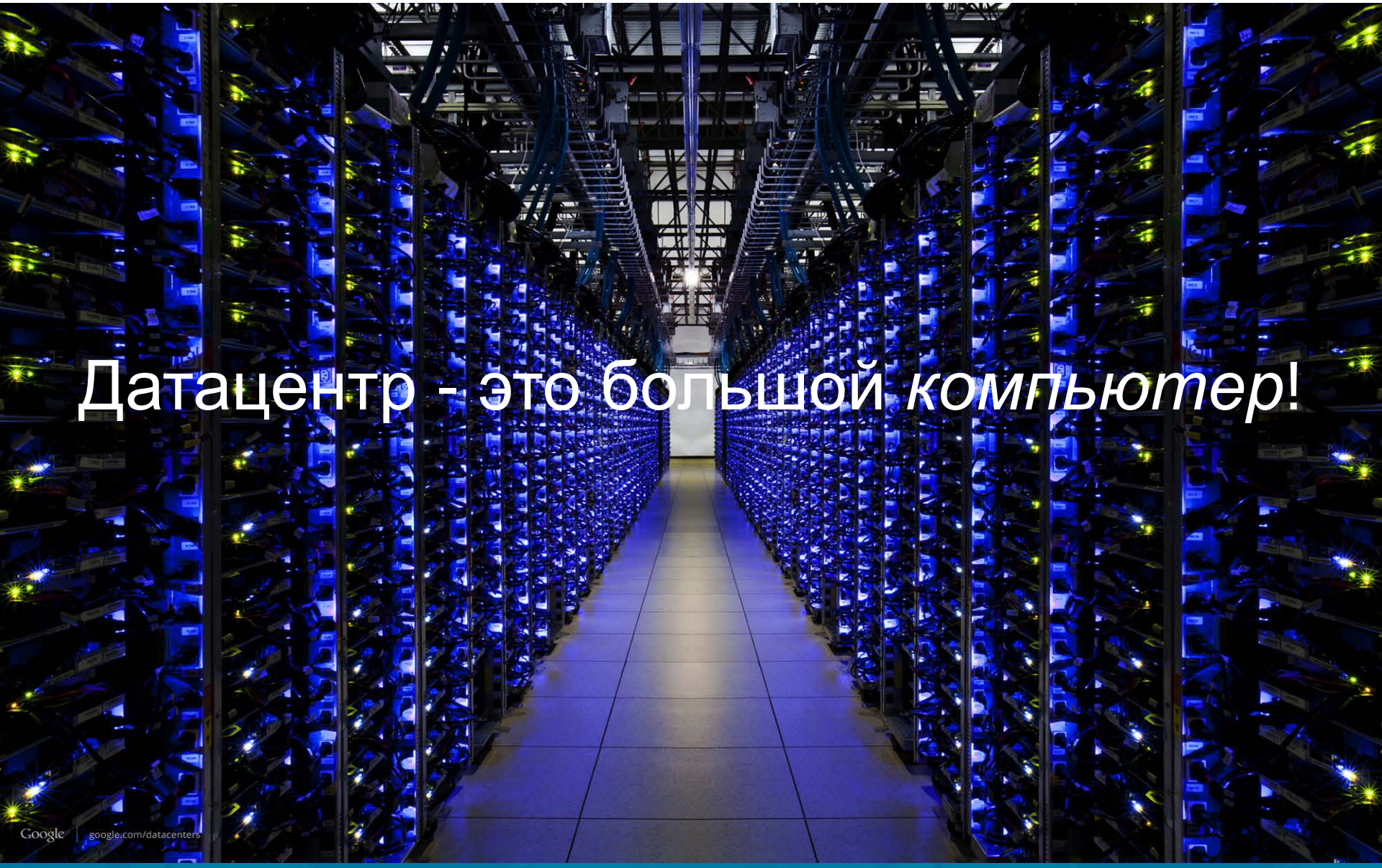


Момент истины

- Concurrency сложная вещь
- Concurrency еще более сложная вещь…
  - в рамках работы в нескольких датацентрах
  - при наличии отказов
  - с точки зрения множества интерактивных сервисов
- И мы еще промолчим про отладку!
- Реальность:
  - Множество одноразовых и узкозаточенных решений
  - Написание своих специальных библиотек, затем их использование
  - Бремя программиста все это поддерживать


XAdr
ILL OP
JT
PCSEL
4 3 2 1 0
PC
00
+4
A
Instruction Memory
D
Ra: <20:16>
Rb: <15:11>
Rc: <25:21>
0 1
RA2SEL
WASEL
XP
Rc: <25:21>
1
0
RA1
WA
RD1
Register File
RA2
WD
RD2
WE
WERF
Z
JT
C: SXT(<15:0>)
PC+4+4*SXT(C)
IRQ
Z
Control Logic
ASEL
1 0
1 0
BSEL
PCSEL
RA2SEL
ASEL
BSEL
WDSEL
ALUFN
Wr
WERF
WASEL
A
ALU
B
ALUFN
WD
R/W
Wr
Data Memory
Adr
RD
PC+4
0 1 2
WDSEL


Instruction Memory
Register File
Control Logic
ALU
Data Memory



Датацентр - это большой *компьютер*!

Так в чем же основной вопрос?

- Это все о правильном уровне абстракции
  - Переместится поверх архитектуры фон Неймана
  - Нам нужны программные модели получше!
- Скрыть детали системного уровня от разработчиков
  - Нет больше race conditions, lock contention и т.д..
- Отделение *что* от *сейчас*
  - Разработчик определяет те вычисления, которые надо произвести
  - Сам фреймоворк (в “runtime”) обрабатывает процесс запуска и выполнения вычислений

Датацентр – это компьютер!

“Big Ideas”

- Масштабирование “горизонтальное”, не “вертикальное”
  - Ограничения SMP и больших shared-memory машин
- Перенос процесса вычислений к данным
  - Кластер имеет ограниченную пропускную способность
- Последовательная обработка данных, избегать произвольного доступа
  - Seeks дороги, пропускная способность диска вполне достаточная
- Полная масштабируемость
  - От мифического человеко-час к понятному машино-час

MapReduce

Типичные задачи Big Data

Map

- Пройтись по большому числу записей
- Извлечь что-то интересное из каждой из записи
- Смешать и отсортировать промежуточные результаты

Reduce

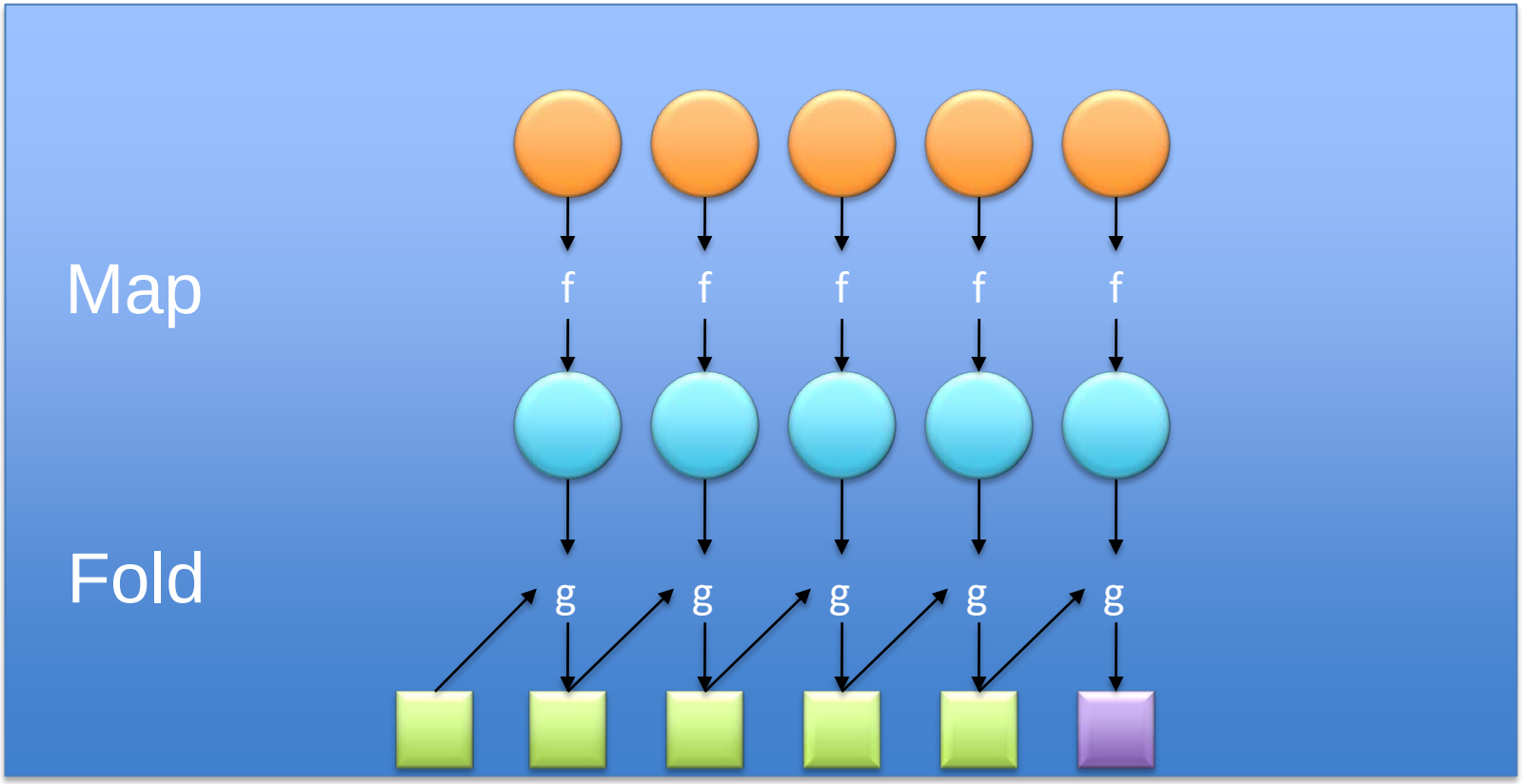
- Объединить промежуточные результаты
- Сформировать выходной результат

Ключевая идея: предоставить функциональные абстракции для этих двух операций

Примеры задач для MapReduce

- В Google:
  - Построение индекса в Google Search
  - Кластеризация статей для Google News
  - Статистический машинный перевод
- В Поиск@Mail.Ru:
  - Парсинг скаченных web-страниц
  - Поиск дубликатов документов
  - Расчет ранков веб-страниц
- В Facebook:
  - Задачи Data Mining
  - Оптимизация показа рекламы
  - Определение спама

Корни из функционального программирования



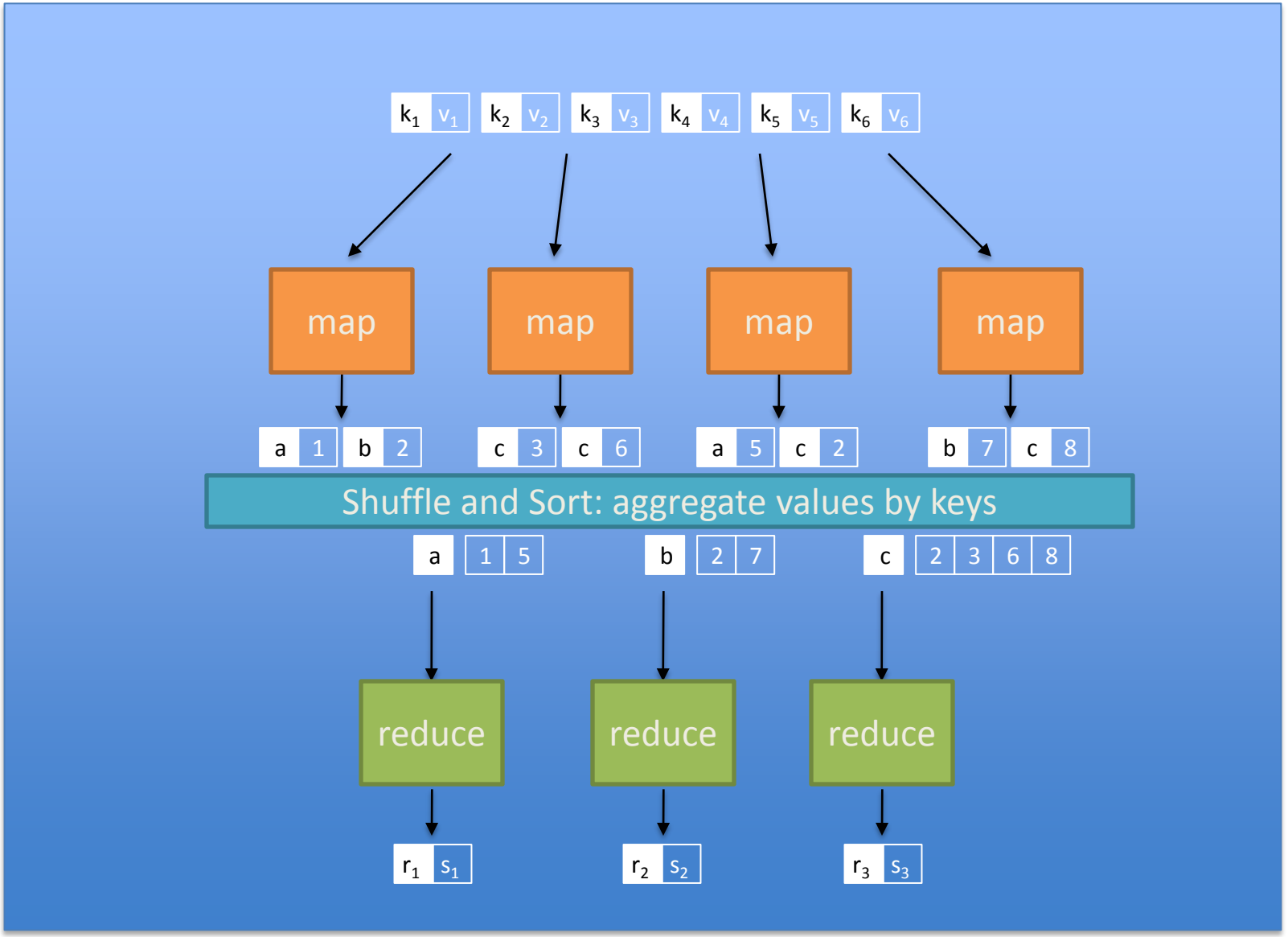
MapReduce

- Программист определяет две функции:

  map $(k_1, v_1) \rightarrow [<k_2, v_2>]$

  reduce $(k_2, [v_2]) \rightarrow [<k_3, v_3>]$

  – Все значения с одинаковым ключом отправляются на один и тот же reducer
- Всем остальным управляет сам фреймворк…


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

MapReduce

- Программист определяет две функции :

  map (k, v) → <k', v'>*

  reduce (k', v') → <k', v'>*

  – Все значения с одинаковым ключом отправляются на один и тот же reducer

- Всем остальным управляет сам фреймворк…

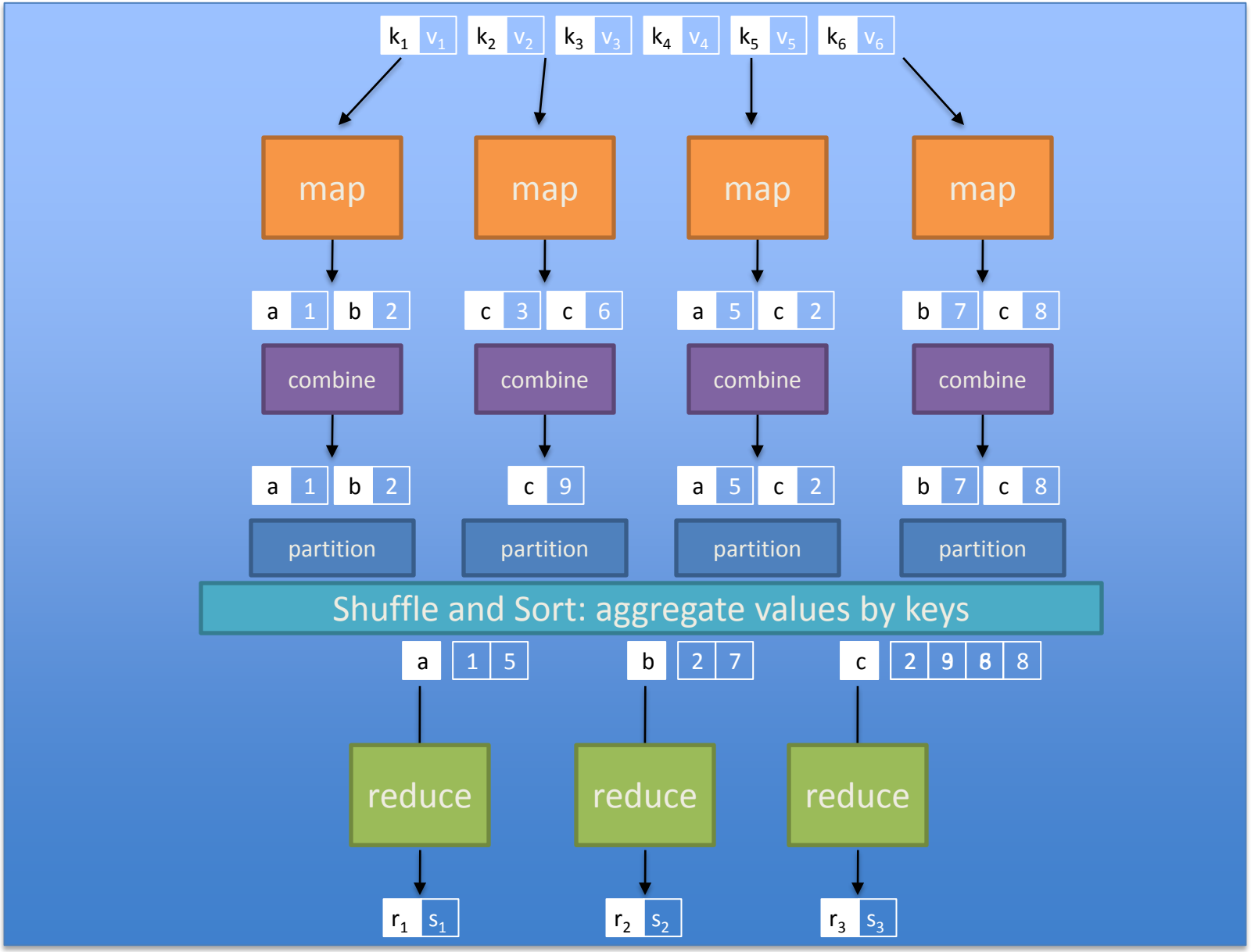
А что такое “всем остальным”?

MapReduce “Runtime”

- Управление запуском
  - Присваивает воркерам map или reduce таски
- Управление “data distribution”
  - Перемещает код к данным
- Управление синхронизацией
  - Собирает, сортирует и объединяет промежуточные данные
- Управление ошибками и отказами
  - Определяет отказ воркера и перезапускает таск
- Все работает поверх распределенной FS (попозже об этом)

MapReduce

- Программист определяет две функции:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  – Все значения с одинаковым ключом отправляются на один и тот же reducer
- Всем остальным управляет сам фреймворк…
- Не совсем…обычно, программист также определяет:
  partition (k', number of partitions) → partition for k'
  – Часто просто хеш от key, напр., hash(k') mod n
  – Разделяет множество ключей для параллельных операций reduce
  combine (k', v') → <k', v'>*
  – Мини-reducers которые выполняются в после завершения фазы map
  – Используется в качестве оптимизации для снижения сетевого трафика на reduce


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Еще пара деталей…

- Граница между фазами map и reduce
  - Но можно начать копировать промежуточные данные заранее
- Ключи приходят на каждый reducer в отсортированном виде
  - Но нет общей сортировки между редьюсерами

“Hello World”: Word Count

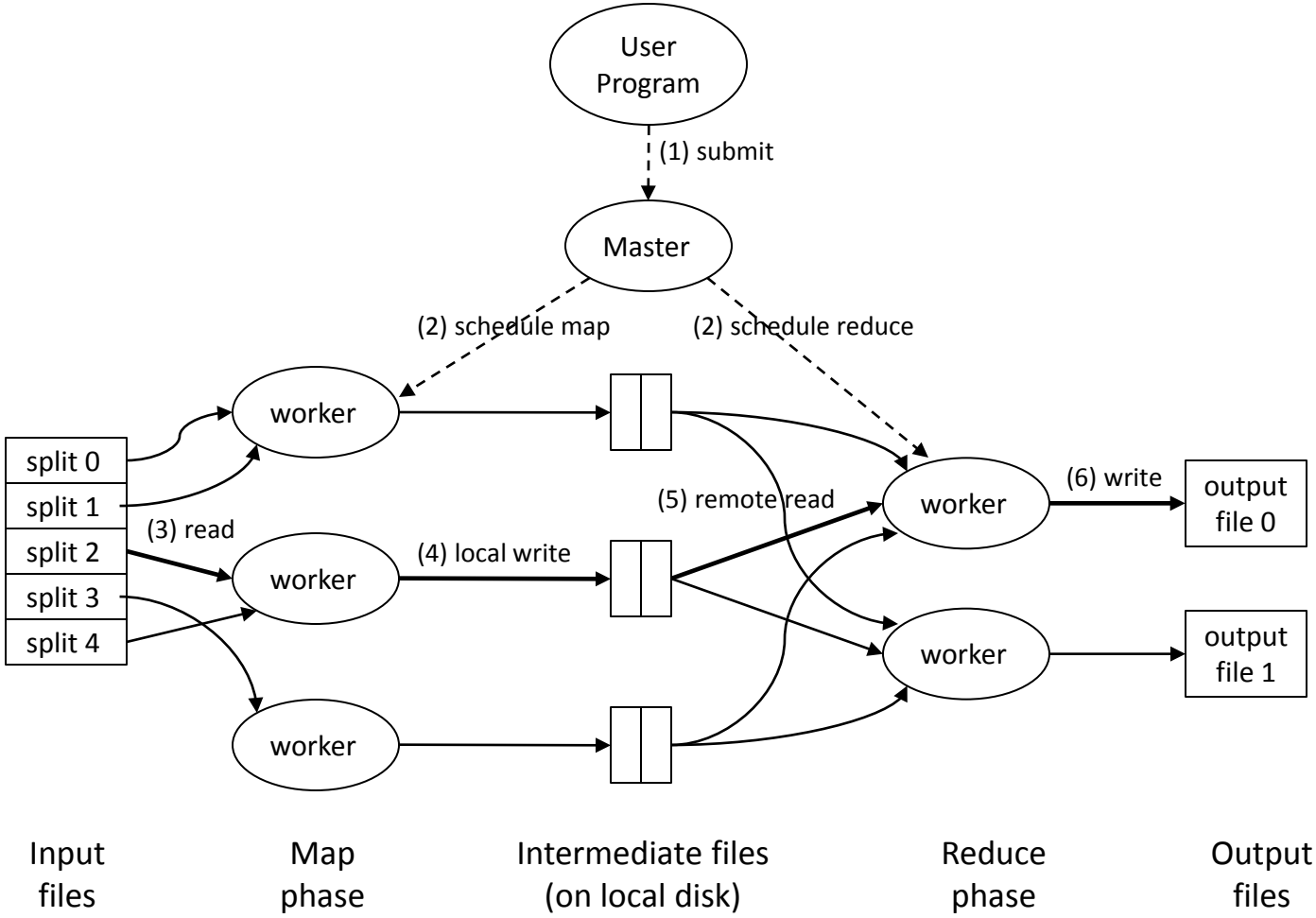
```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);

Reduce(String term, Iterator<Int> values):
    int sum = 0;
    for each v in values:
        sum += v;
        Emit(term, value);
```

MapReduce можно понимать как…

- Программная модель
- Среда (фреймворк) выполнения (aka “runtime”)
- Спецификация реализации

Использование обычно понятно из контекста!


User Program
(1) submit
Master
(2) schedule map
(2) schedule reduce
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
worker
worker
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disk)
Reduce phase
Output files

Как же воркеры получают данные?

Какая тут проблема?

Distributed File System

- Не надо копировать данные к воркерам… надо отправлять воркеры к данным!
  - Сохраняем данные на локальных дисках нод кластера
  - Запускаем воркеры на нодах, где данные локальны
- Зачем?
  - Недостаточно RAM чтобы уместить все данные в памяти
  - Доступ к диску медленный, но скорость чтения с диска приемлемая
- Ответ: распределенная файловая система
  - GFS (Google File System) для Google’s MapReduce
  - HDFS (Hadoop Distributed File System) для Hadoop

GFS: Предположения

- Обычное «железо» вместо «экзотики»
  - Масштабируем “горизонтально”, а не “вертикально”
- Высокая частота отказа компонентов
  - Недорогие компоненты обычного железа постоянно фейлятся
- “Умеренное” кол-во огромных файлов
  - Мульти-гигабайтные файлы – обычная ситуация
- Файлы являются write-once, в основном дописываются или перезаписываются полностью
  - Возможно, многопоточно
- Последовательное чтение большого объема данных вместо произвольного доступа
  - Высокая пропускная способность (throughput) вместо небольшой задержки (latency)

GFS: Концепция дизайна

- Файлы хранятся в виде чанков (блоков)
  - Фиксированный размер (64MB)
- Надежность через репликацию
  - Каждый чанк реплицируется на 3 сервера
- Один мастер-процесс координирует доступ, хранит мета-информацию
  - Простое централизованное управление
- Данные не кешируются
  - Небольшая польза для больших наборов данных, потоковое чтение
- Простое API
  - Push some of the issues onto the client (e.g., data layout)

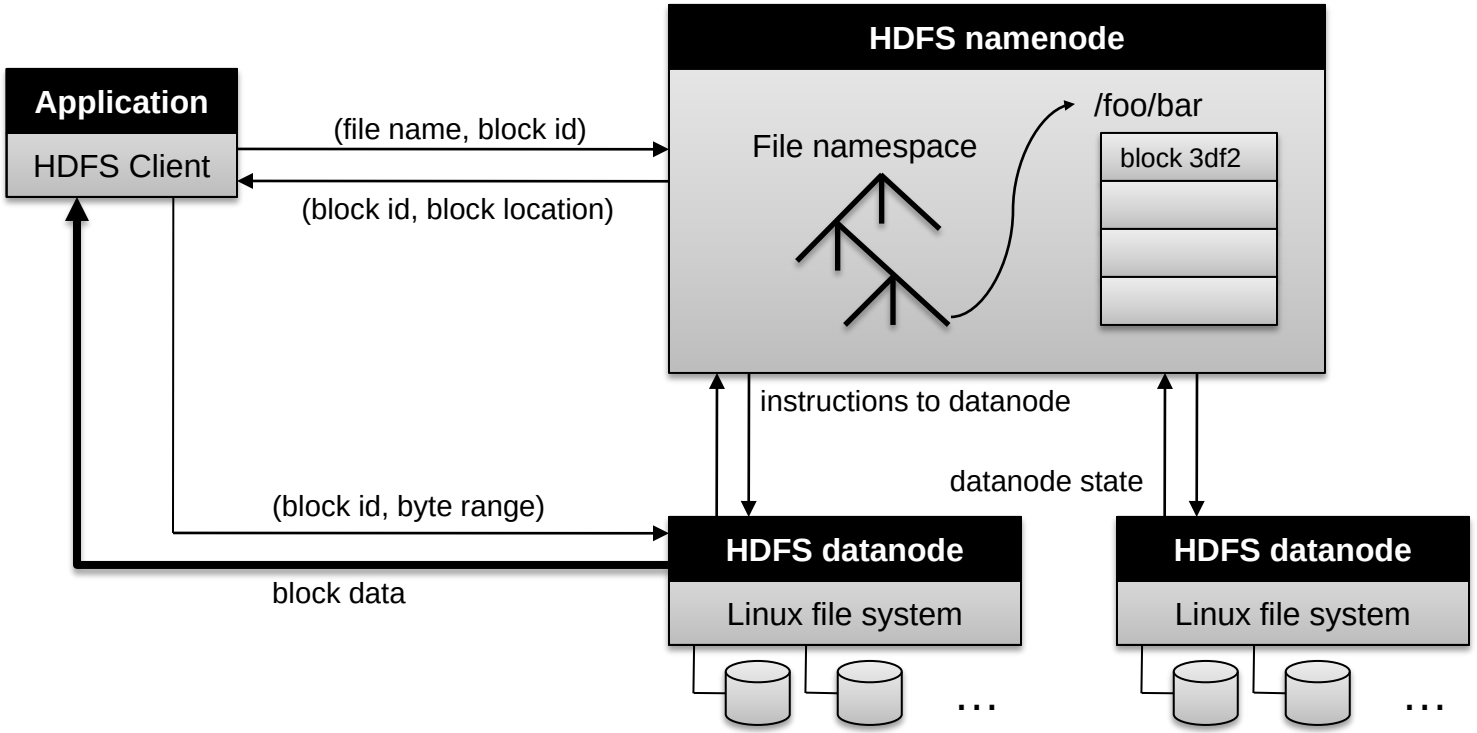
HDFS = GFS клон (те же основные идеи)

От GFS к HDFS

- Различия терминологии:
  - GFS master = Hadoop namenode
  - GFS chunkservers = Hadoop datanodes
- Отличия:
  - Разная consistency model для file appends
  - Реализация
  - Производительность

В большинстве случаев будем использовать терминологию Hadoop

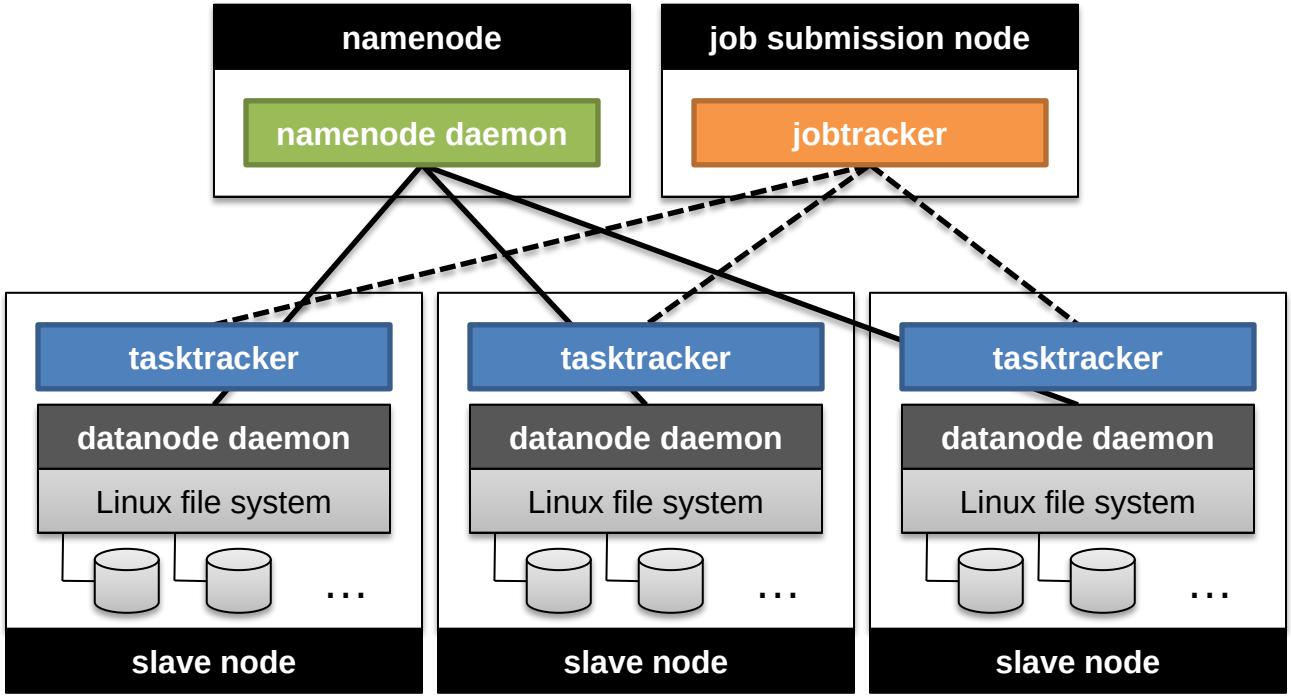
HDFS Architecture



Обязанности Namenode

- Управление пространством имен файловой системы:
  - Хранить структуру файлов/директорий, метаданные, маппинг file-to-bloc, права доступа и т.д.
- Координирование файловых операций:
  - Направлять клиентов на датаноды для чтения и записи
  - Данные не проходят через namenode
- Поддерживать общее состояние системы:
  - Периодические обращения к датанодам
  - Дорепликация блоков и перебалансировка
  - Сборка мусора

Теперь соберем все вместе…



(Не совсем… Позже поговорим про YARN)

Вопросы?