

# **Большие данные**

613x-010402D       $x=\{1,2,3\}$

*осень 2024*

## **Лекция 4:**

NBDRA: Эталонная архитектура больших данных

Распределённые системы обработки данных

Сергей Борисович Попов

[sepo@ssau.ru](mailto:sepo@ssau.ru)

## **Материалы лекций:**

[https://1drv.ms/f/s!ApFj4iOLPNegvEPfMZL\\_5jjkXsqfQ](https://1drv.ms/f/s!ApFj4iOLPNegvEPfMZL_5jjkXsqfQ)

# Стандартизация в области Больших данных

- NIST Big Data Interoperability Framework  
<https://www.nist.gov/itl/big-data-nist>  
Цель: эталонная архитектура больших данных NIST (NBDRA)
- ISO/IEC JTC 1, Big Data
- ГОСТ Р ИСО/МЭК 20546-2019.  
Информационные технологии.  
Большие данные. Обзор и словарь.



# NIST Big Data Interoperability Framework

## Основы interoperability Больших данных

**Цель:** эталонная архитектура больших данных NIST (NBDRA – NIST Big Data Reference Architecture)

Разработка проходила в три этапа:

Этап 3: проверка NBDRA путем создания общих приложений для больших данных через общие интерфейсы;

Этап 2: определение общих интерфейсов между компонентами NBDRA;  
и

Этап 1: определение ключевых компонентов эталонной архитектуры больших данных высокого уровня, которые не зависят от технологий, инфраструктуры и поставщиков.

# NIST Big Data Interoperability Framework

Том 1, Определения

Том 2, Таксономии

Том 3, Примеры использования и общие требования

Том 4, Безопасность и конфиденциальность

Том 5, Обзор официального документа по архитектурам

**Том 6, Эталонная архитектура**

Том 7, Дорожная карта стандартов

Том 8, Интерфейсы эталонной архитектуры

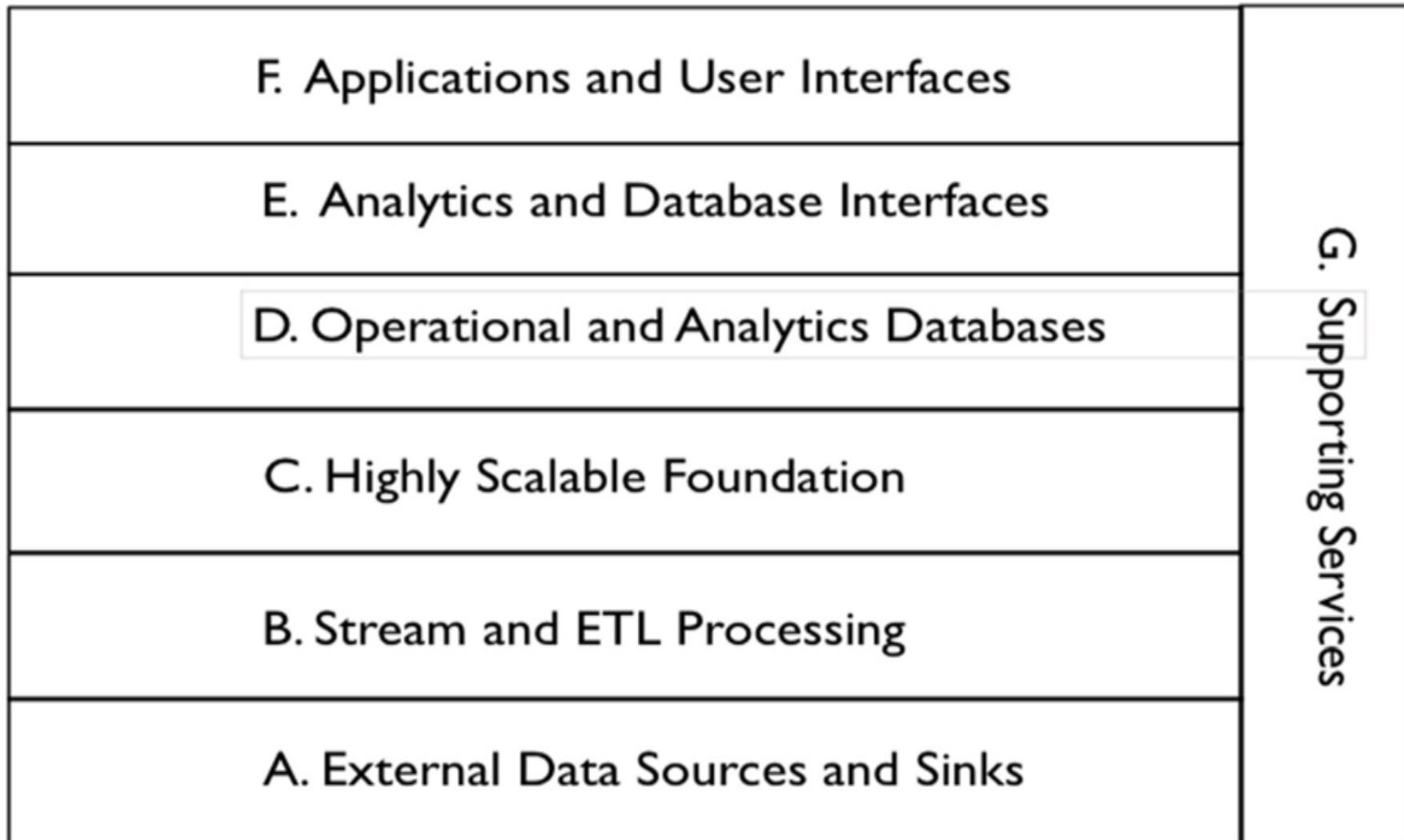
Том 9, Внедрение и модернизация



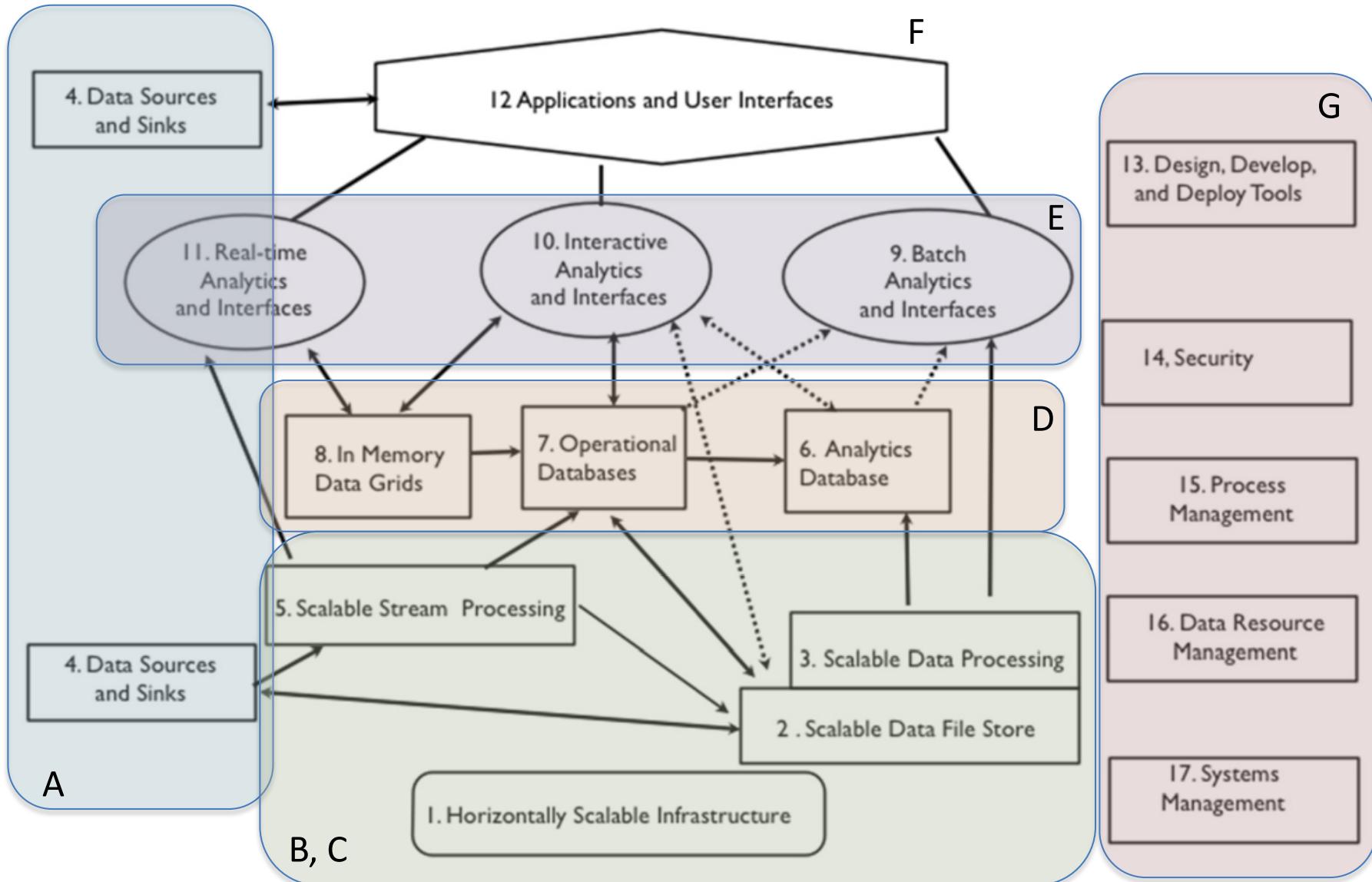
# Предложения по архитектуре Больших Данных

1. ET Strategies, Robert Marcus
2. Microsoft, Orit Levin,
3. University of Amsterdam, Yuri Demchenko
4. IBM, James Kobielski
5. Oracle, Harry Foxwell
6. EMC/Pivotal, Milind Bhandarkar
7. SAP , Sanjay Patil
8. 9sight Consulting, Barry Devlin
9. LexisNexis, Tony Middleton

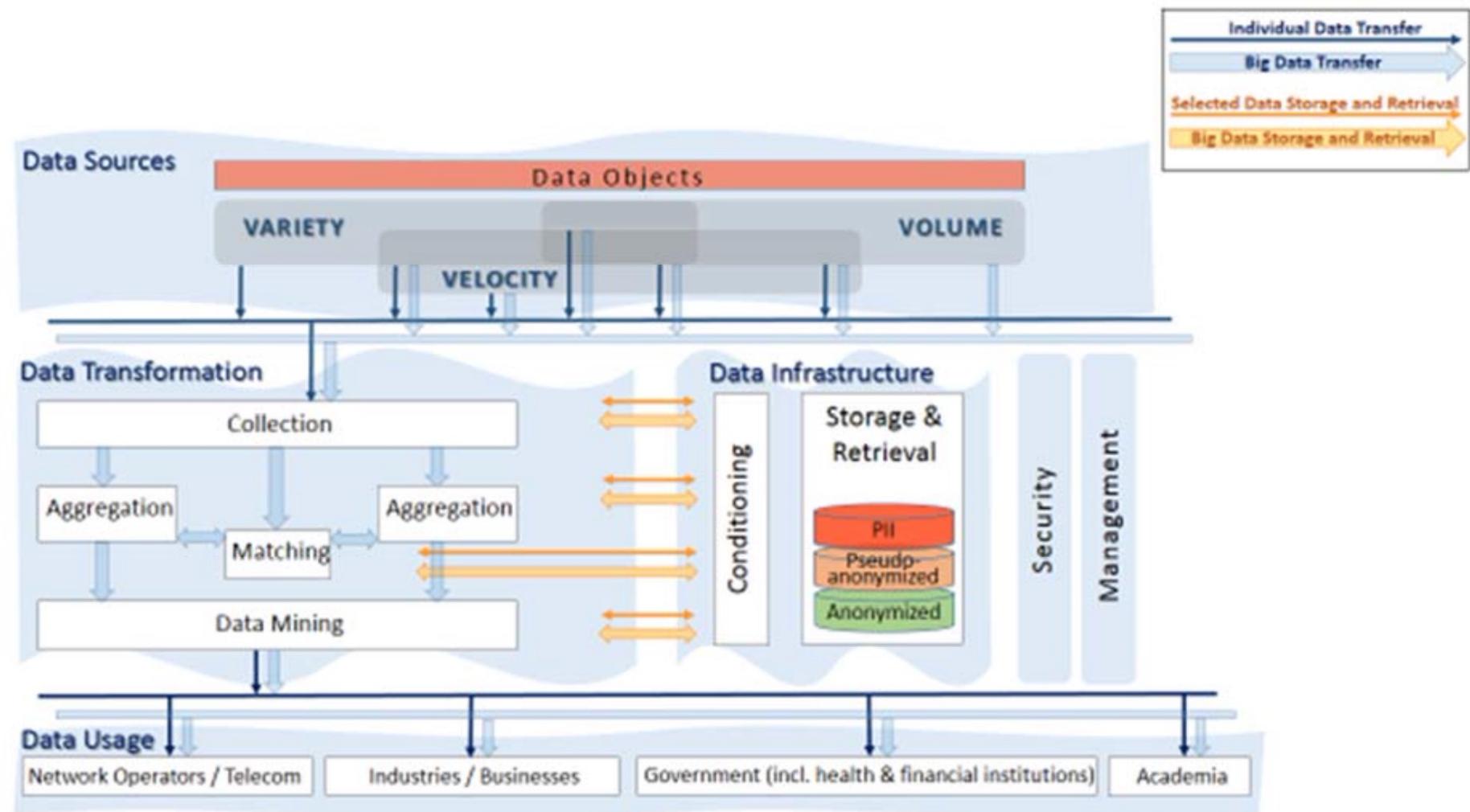
# ET Strategies: High Level Reference Model



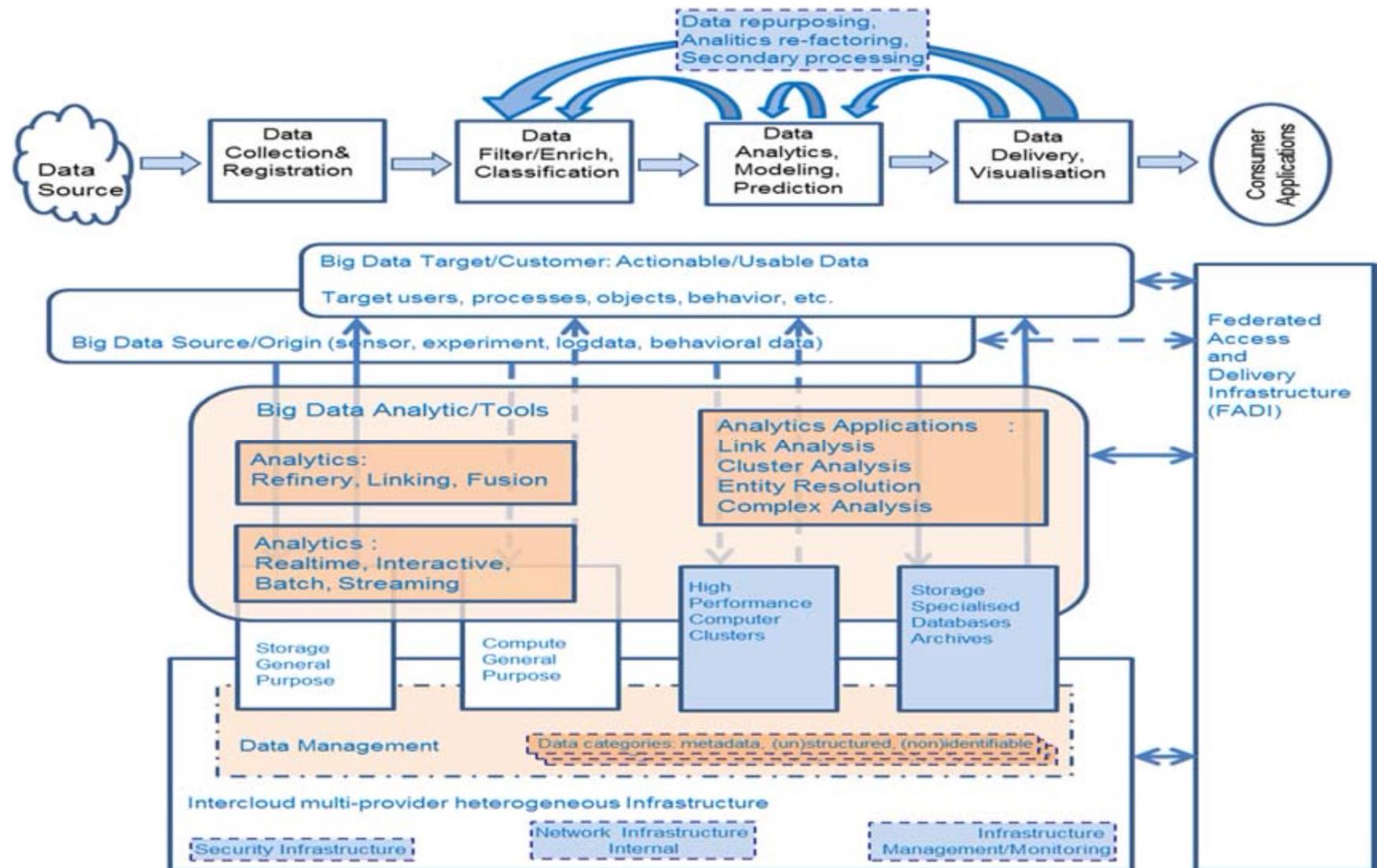
# ET Strategies: Low Level Reference Model



# Microsoft: Big Data Ecosystem Reference Architecture



# University of Amsterdam: Big Data Architecture Framework



# IBM: Architecture Model

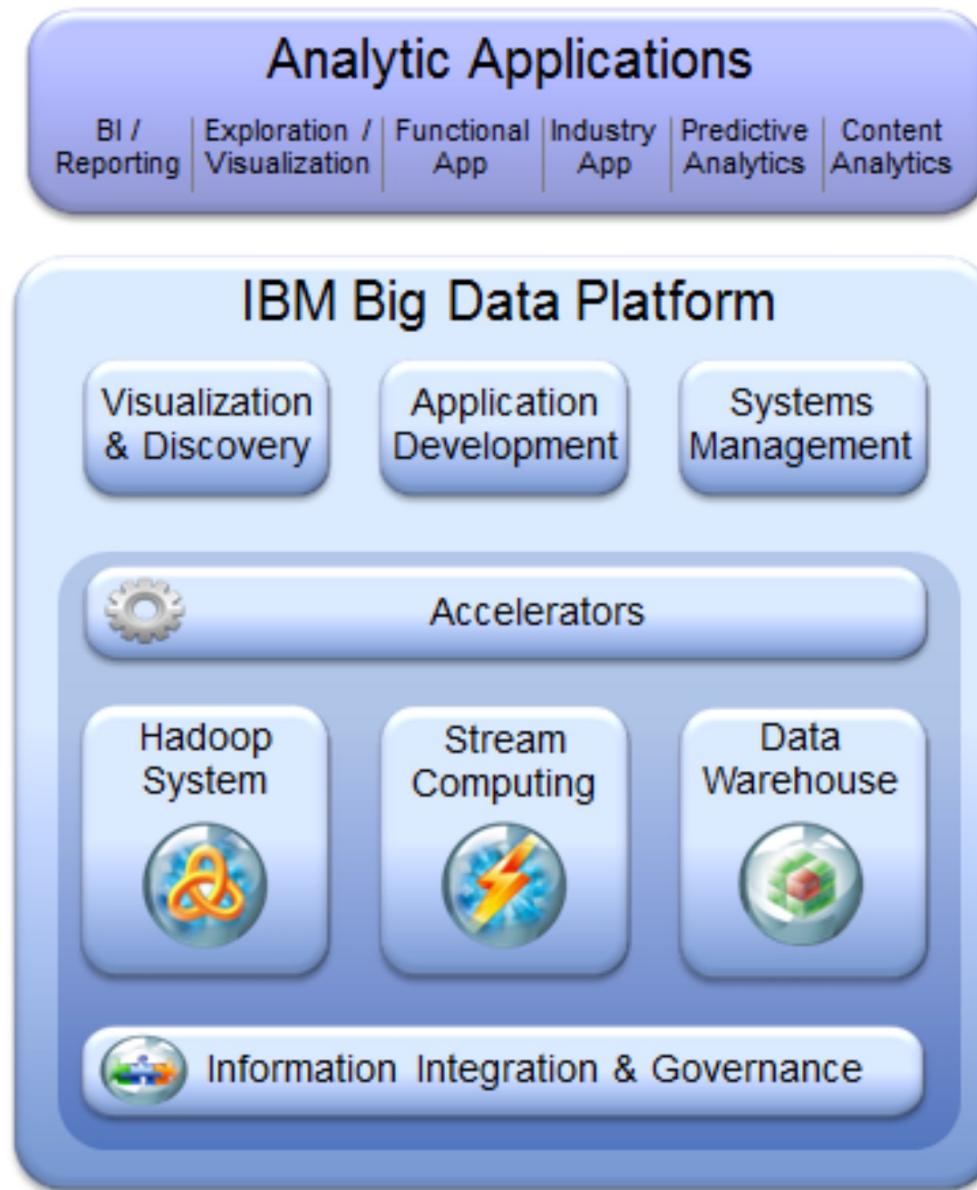
## Ключевые требования:

- Обнаружение и исследование данных
- Экстремальная производительность: выполнение задач аналитики ближе к данным
- Управление и анализ неструктурированных данных
- Анализ данных в реальном времени
- Богатая библиотека аналитических функций и наборов инструментов
- Интеграция и управление всеми источниками данных

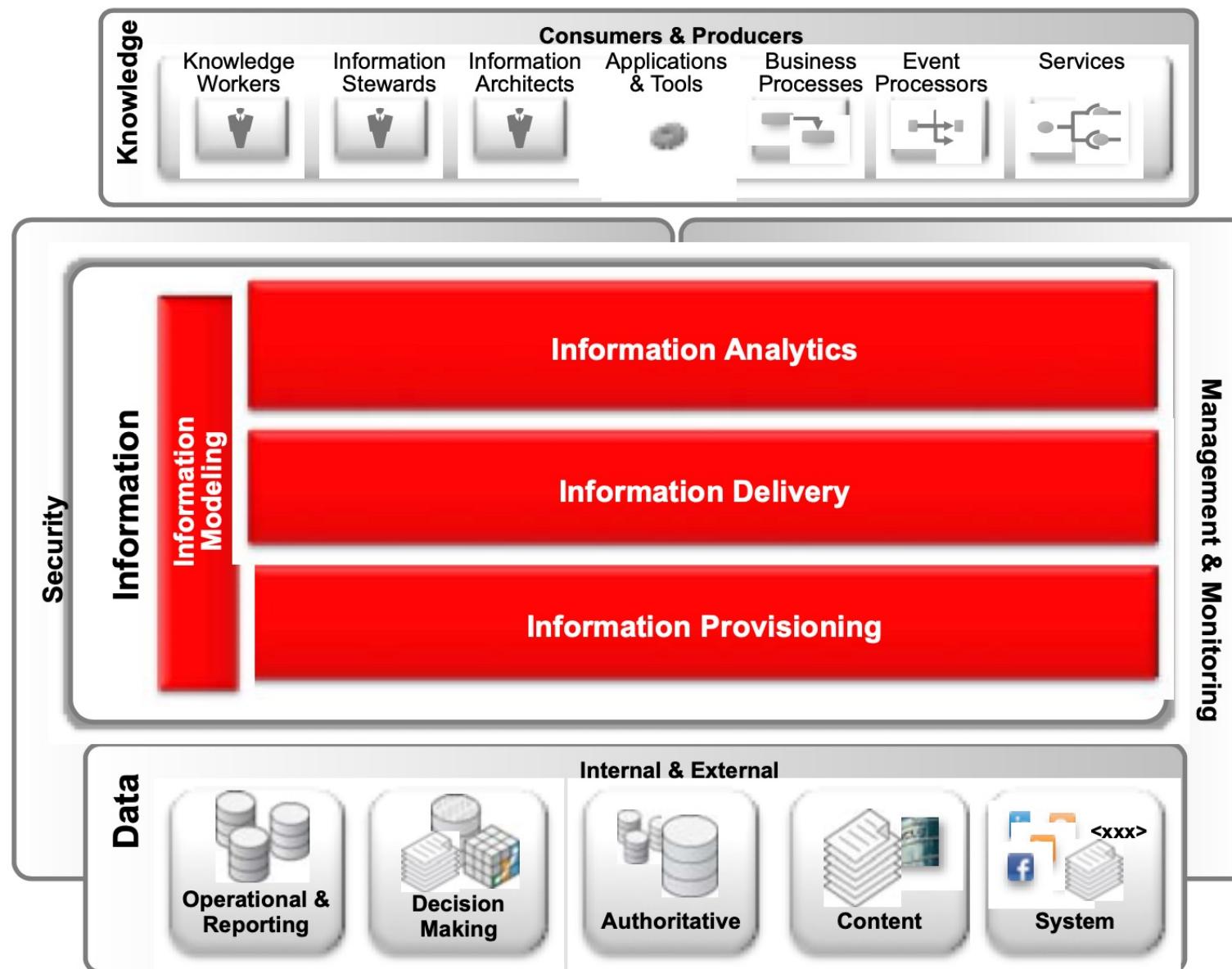
## Ключевые компоненты:

- |                                       |  |
|---------------------------------------|--|
| • Инструменты                         | • Tools                                  |
| • Ускорители                          | • Accelerators                           |
| • Hadoop                              | • Hadoop                                 |
| • Потоковые вычисления                | • Stream Computing                       |
| • Хранилище данных                    | • Data Warehouse                         |
| • Интеграция и управление информацией | • Information Integration and Governance |

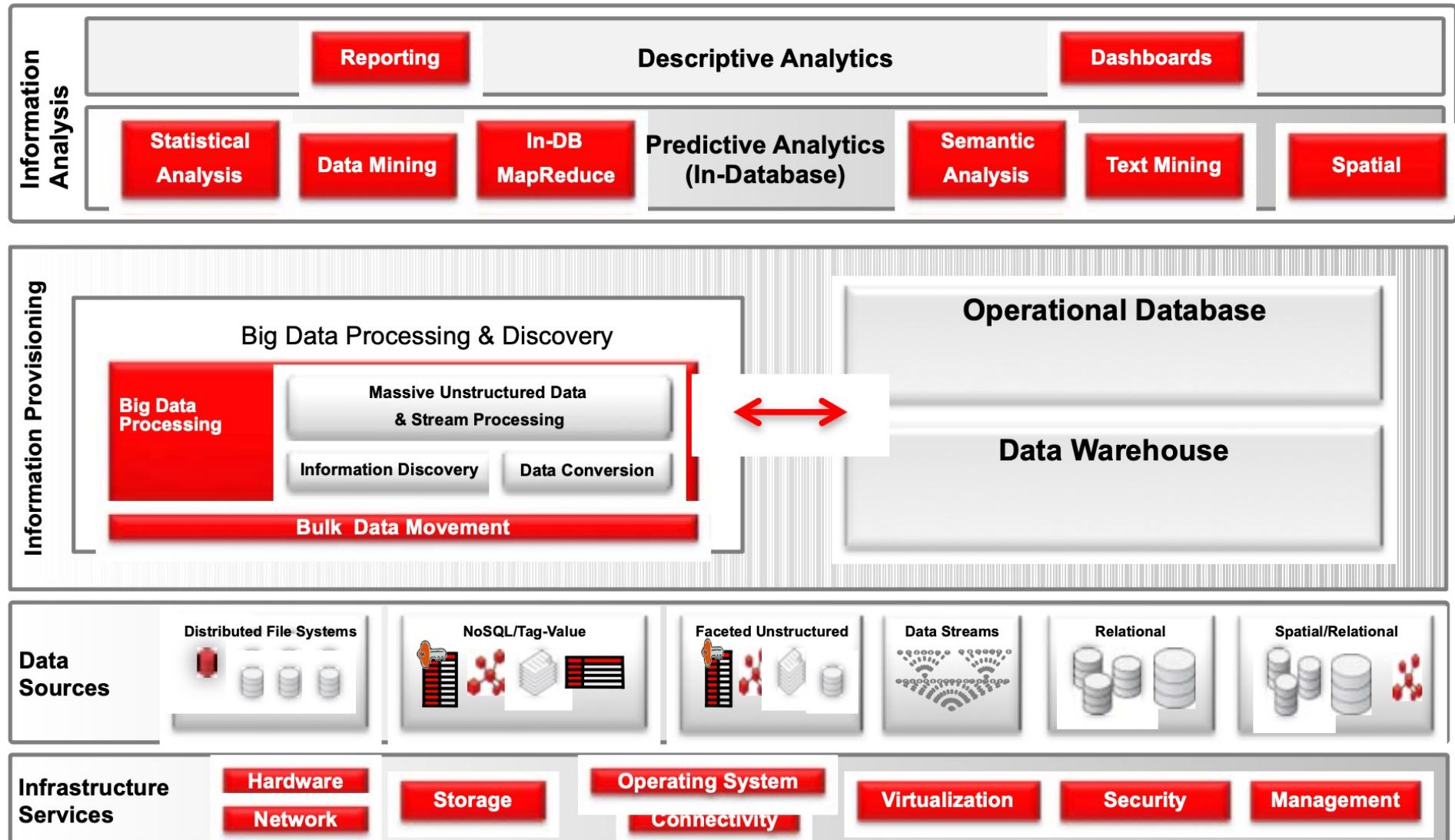
# IBM: Reference architecture – Big Data Platform



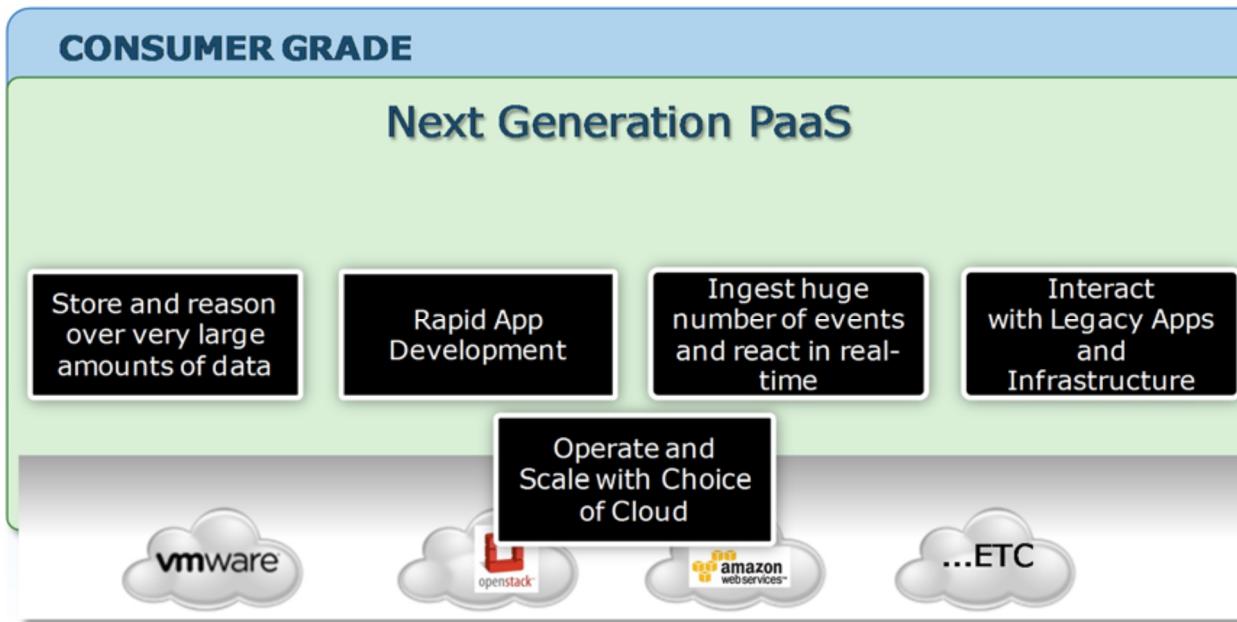
# Oracle: Conceptual View of Information Management Ecosystem



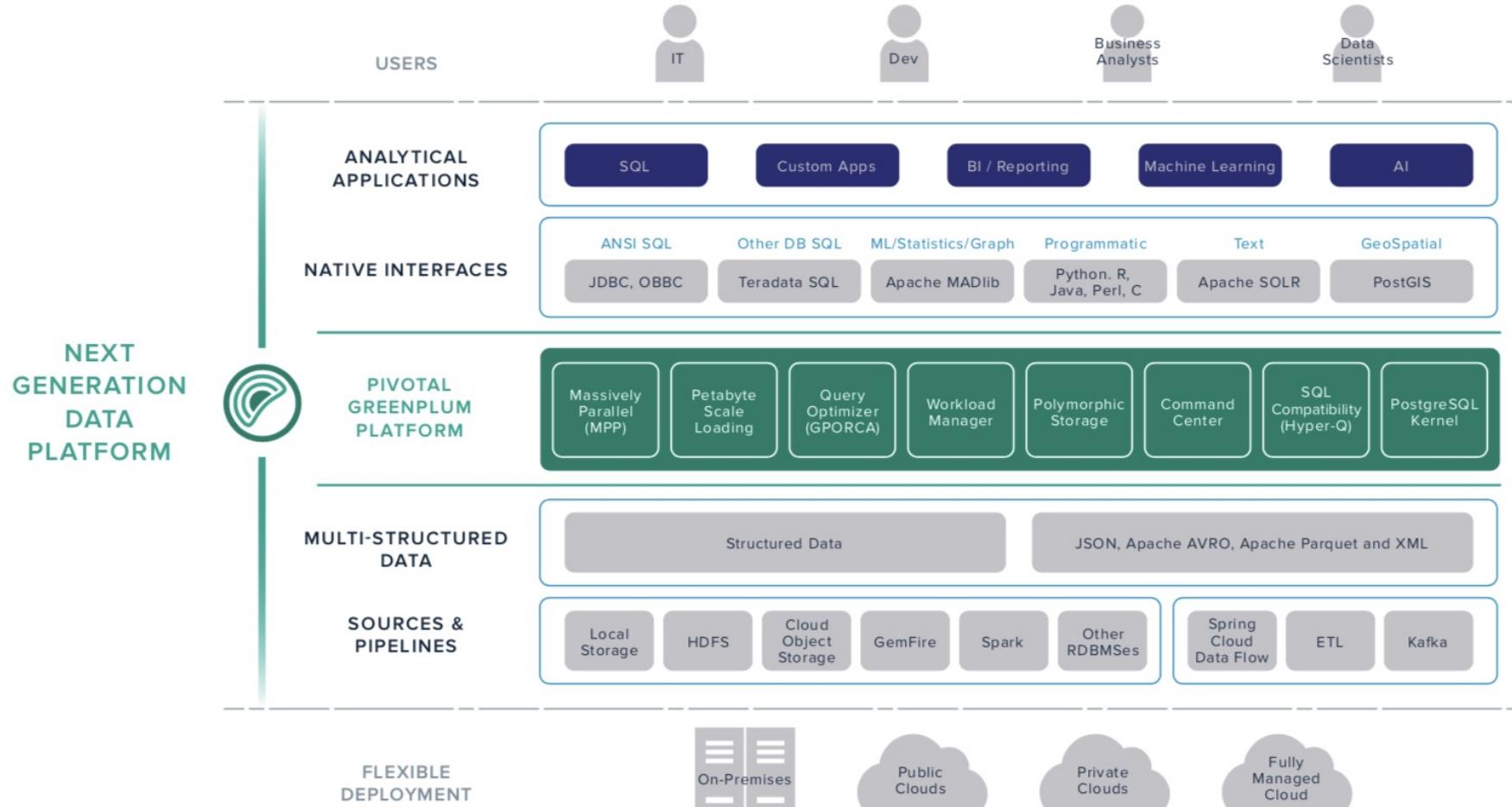
# Oracle Big Data Reference Architecture



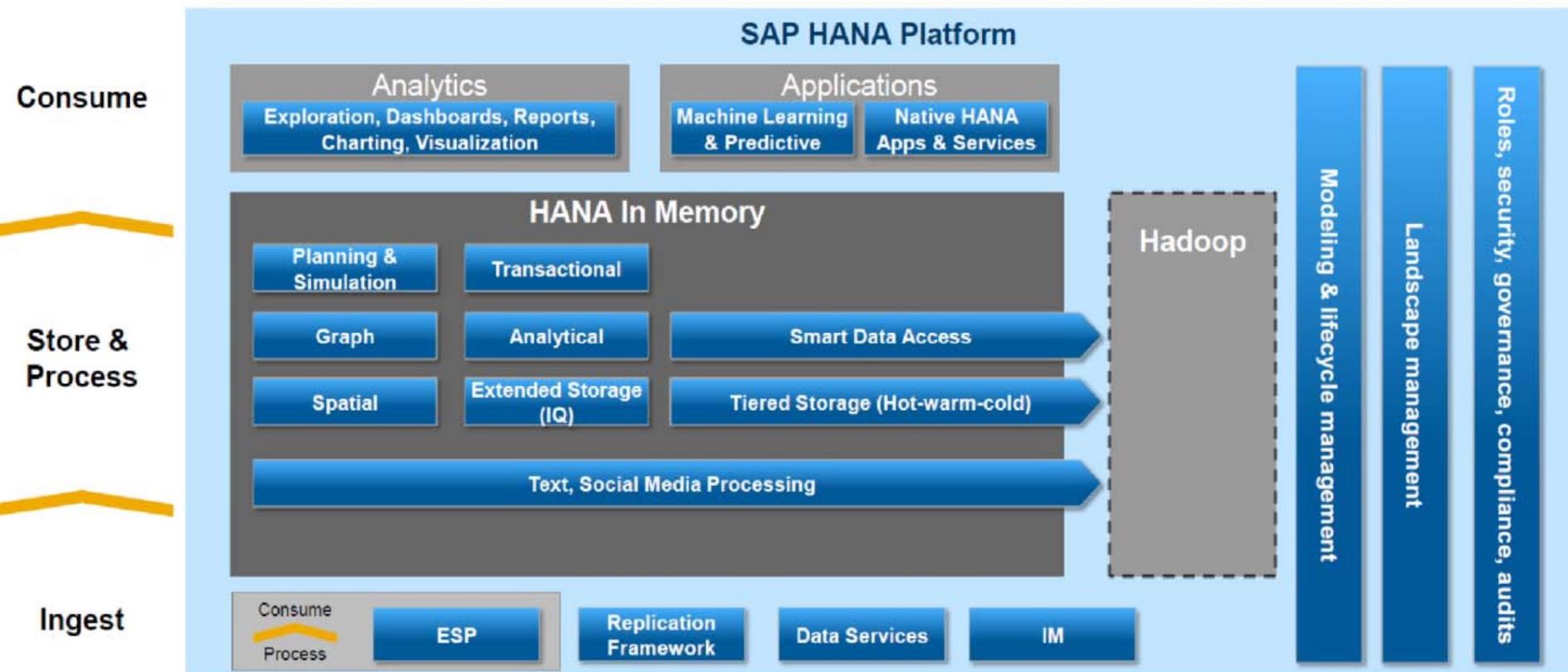
# Pivotal Architecture Model & Key Components



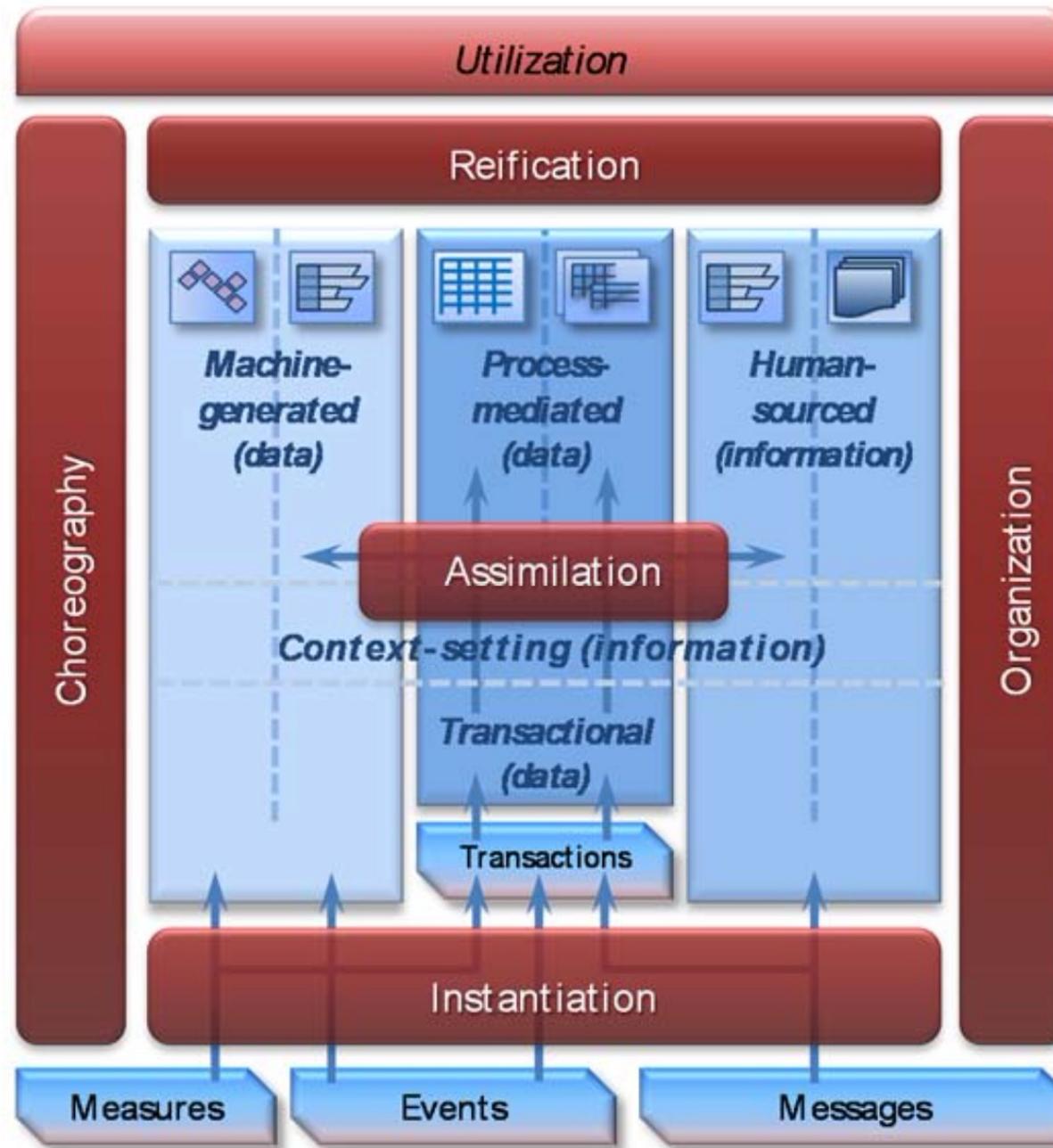
# vmware Tanzu



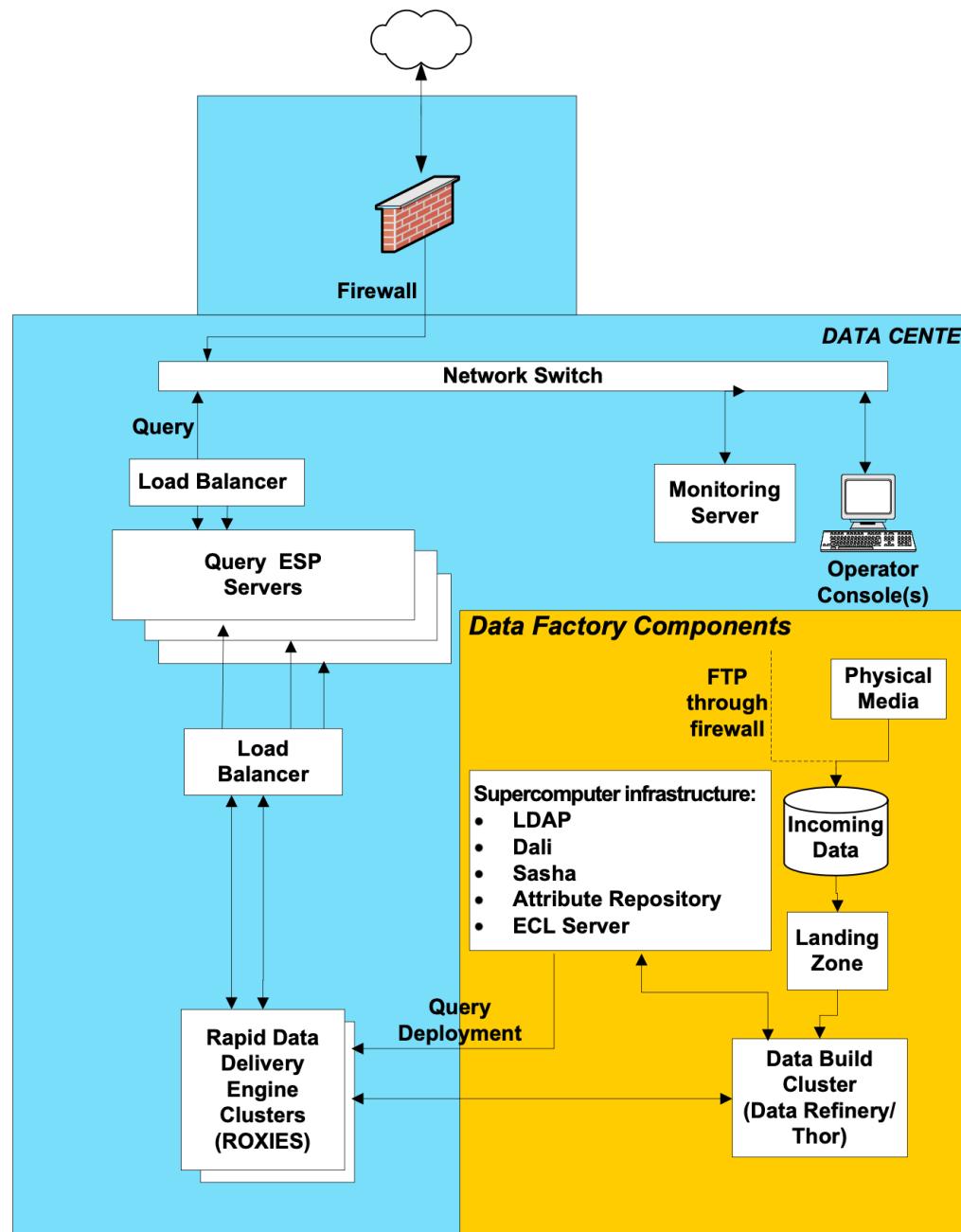
# SAP: HANA platform for Big Data



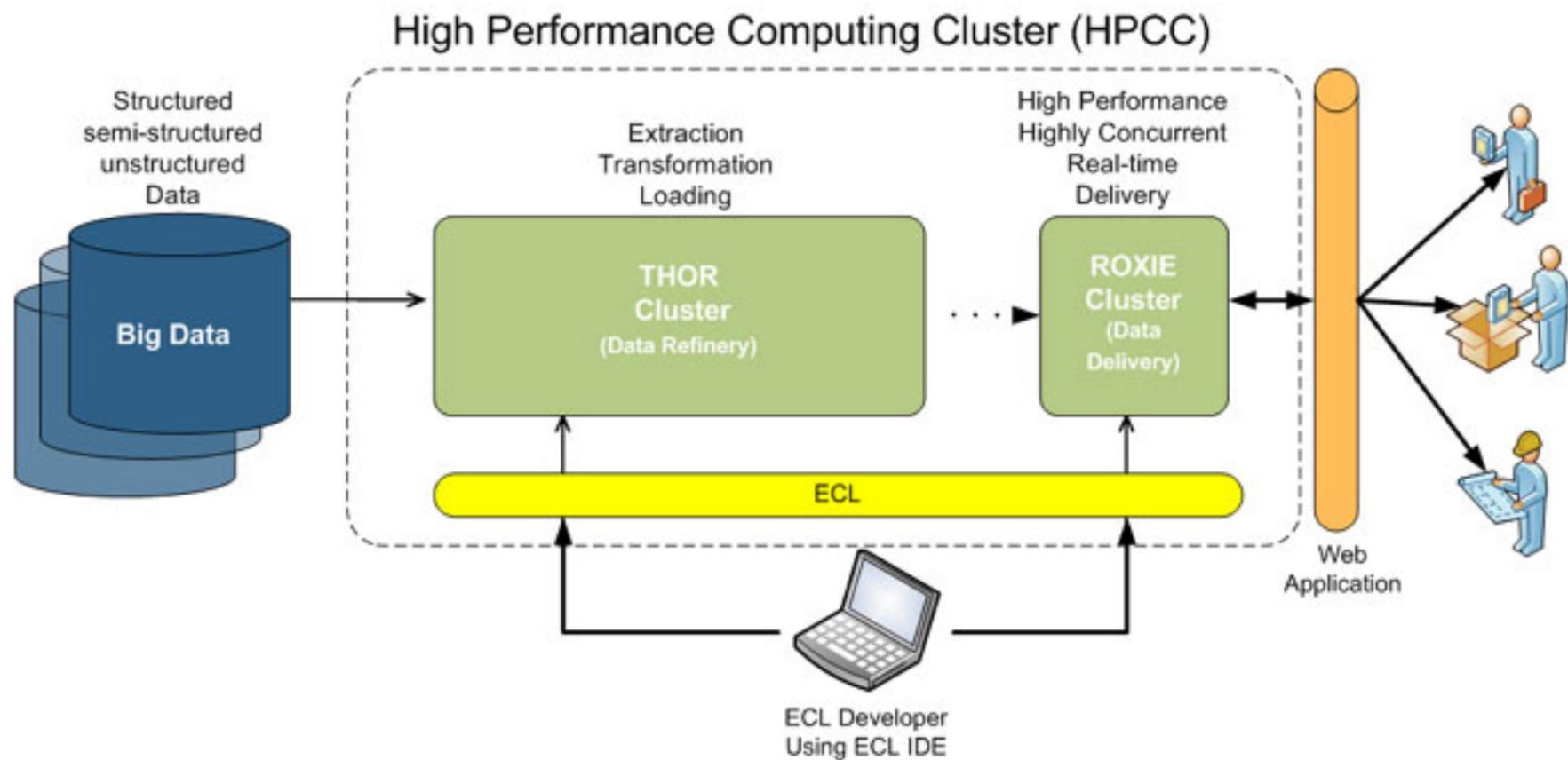
# 9Sight: Architecture Model



# LexisNexis: General Architecture



# Lexis Nexis High Performance Computing Cluster



\* The Enterprise Control Language (ECL)

# Анализ реализаций архитектуры больших данных

- **Ведение и хранение больших данных**
  - Структурированные, полу-структурные и неструктурированные данные
  - Объем, разнообразие, скорость и изменчивость
  - SQL and NoSQL
  - Распределённая файловая система
- **Аналитика больших данных и интерфейсы приложений**
  - Описательная, прогностическая и пространственная
  - В режиме реального времени
  - Интерактивная
  - Аналитика в режиме пакетной обработки
  - Отчетность
  - Панели мониторинга/управления
- **Инфраструктура больших данных**
  - In-memory data grids (IMDG) – распределённое хранение данных в оперативной памяти
  - Оперативная база данных
  - Аналитическая база данных
  - Реляционная база данных
  - Плоские файлы
  - Система управления контентом
  - Горизонтальная масштабируемая архитектура

# Элементы архитектуры данных

- **Хранилище данных (Data storage)**

Во всех архитектурах данных необходимо указать, как хранятся данные, включая физический носитель данных (например, жесткие диски или облачное хранилище) и структуры данных, используемые для организации данных.

- **Обработка данных (Data processing)**

Архитектура данных должна определять, как обрабатываются данные, включая любые преобразования или вычисления, которые выполняются с данными перед их сохранением или анализом.

- **Доступ к данным (Data access)**

Архитектуры данных должны обеспечивать механизмы доступа к данным, включая пользовательские интерфейсы и интерфейсы прикладных программ (API), которые позволяют запрашивать и анализировать данные.

- **Безопасность и конфиденциальность данных (Data security and privacy)**

Архитектуры данных должны включать механизмы обеспечения безопасности и конфиденциальности данных, такие как контроль доступа, шифрование и маскирование данных.

- **Управление данными (Data governance)**

Архитектура данных должна обеспечивать основу для управления данными, включая стандарты качества, отслеживание происхождения и политику хранения.

# **Big Data Management and Storage**

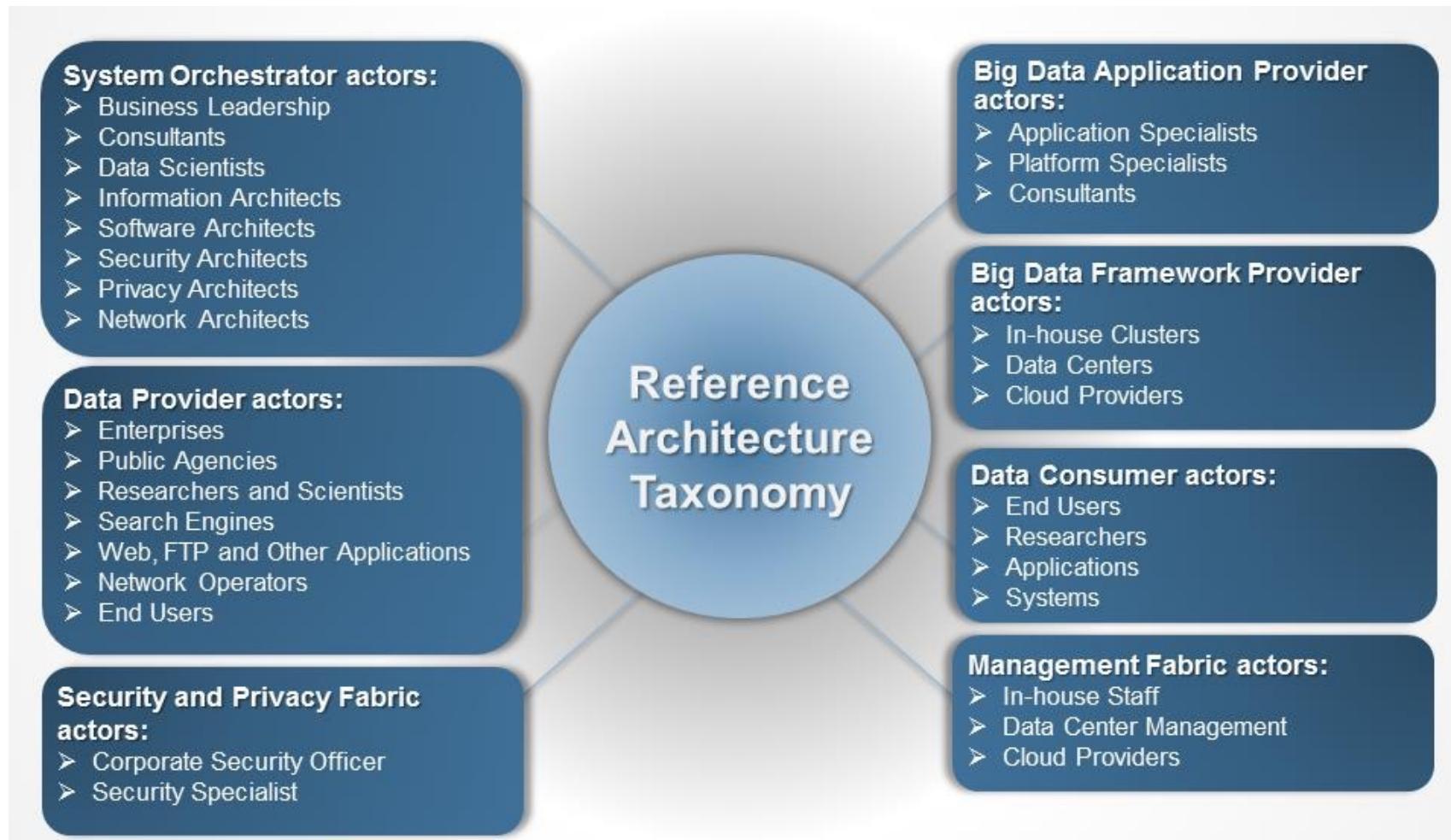
**Big Data Analytics and Application Interfaces**

# **Big Data Infrastructure**

# Составляющие эталонной архитектуры

- Логические функциональные компоненты  
(технический функционал):
  - System Orchestrator (управление взаимодействием систем)
  - Data Provider (поставщик данных)
  - Big Data Application Provider (поставщик приложений)
  - Big Data Framework Provider (поставщик технологий)
  - Data Consumer (потребитель данных)
- Базовые сервисы взаимодействия  
(функционал структурной основы): *Fabric – структурная основа*
  - Management (администрирование)
  - Security and Privacy (безопасность и конфиденциальность)

# Систематика эталонной архитектуры



# System Orchestrator

- Обеспечивает всеобъемлющие требования, которым должна соответствовать система, включая политику, управление, архитектуру, ресурсы и бизнес-требования, а также действия по мониторингу или аудиту для обеспечения соответствия системы этим требованиям.
- Обеспечивает требования к системе, высокоуровневое проектирование и мониторинг системы данных.
- *Несмотря на то, что эта функциональность появилась раньше систем больших данных, некоторые связанные с ней проектные действия изменились в рамках парадигмы больших данных.*

# Data Provider

- Делает данные доступными для себя или для других.
- Создает абстракцию различных типов источников данных (таких как необработанные данные или данные, ранее преобразованные другой системой) и делает их доступными через различные функциональные интерфейсы.
- *Хотя концепция поставщика данных не нова, расширенные возможности сбора и анализа данных открыли новые возможности для предоставления значимых данных.*

# Big Data Application Provider

- Поставщик приложений для больших данных выполняет преобразования данных в соответствии с их жизненным циклом в рамках требований, установленных системным оркестратором.
- Здесь общие возможности инфраструктуры больших данных объединяются для создания конкретной системы данных. (*Алгоритмы соединяются с технологиями*)
- *Хотя действия поставщика приложений одинаковы, независимо от того, относится ли создаваемое решение к большим данным или нет, методы и методы изменились, поскольку данные и обработка данных распараллеливаются между ресурсами.*

# Big Data Framework Provider

- Имеет в своём составе общие ресурсы или службы, которые будут использоваться поставщиком приложений для больших данных при создании конкретного приложения.
- Состоит из одного или нескольких экземпляров трех подкомпонентов: каркасов инфраструктуры, платформ данных и сред обработки (*infrastructure frameworks, data platforms, and processing frameworks*)
- Не требуется, чтобы все экземпляры на данном уровне иерархии относились к одной и той же технологии, фактически, большинство реализаций больших данных представляют собой гибриды, сочетающие несколько технологических подходов. Они обеспечивают гибкость и могут удовлетворить весь спектр требований, выдвигаемых поставщиком приложений для больших данных.
- Эта функциональность претерпела самые значительные изменения из-за больших данных.

# Data Consumer

- Потребитель данных получает выходные данные (результат вычислений) системы больших данных.
- Во многих отношениях он является получателем того же типа функциональных интерфейсов, которые поставщик данных предоставляет поставщику приложений для больших данных. После того, как система добавляет ценность исходным источникам данных, поставщик приложений для больших данных предоставляет потребителю данных тот же тип функциональных интерфейсов.

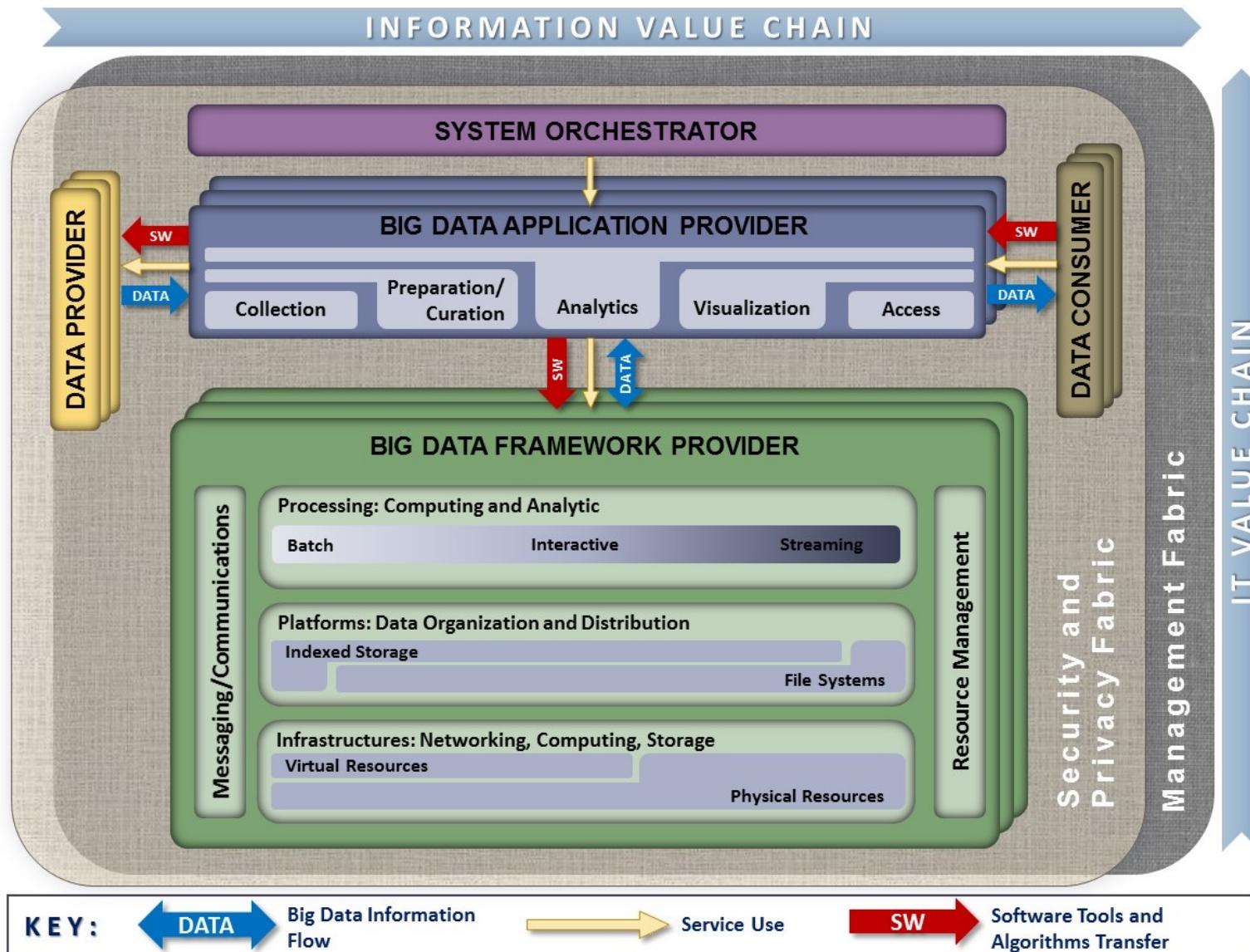
# Management Fabric

- Характеристики больших данных в виде объема, скорости, разнообразия и изменчивости требуют универсальной системы и платформы управления программным обеспечением для инициализации, настройки и управления программным обеспечением и пакетами, а также мониторинга и управления ресурсами и производительностью.
- Администрирование большими данными включает в себя учёт масштабируемости системы, данных, безопасности и конфиденциальности, при сохранении высокого уровня качества данных и безопасной доступности.

# Security and Privacy Fabric

- Проблемы безопасности и конфиденциальности влияют на все остальные компоненты NBDRA.
- Структура безопасности и конфиденциальности согласовывается с *System Orchestrator* при формировании общих политик, требований и аудита, а также с поставщиком приложений для больших данных и поставщиком инфраструктуры для больших данных в процессе разработки, развертывания и эксплуатации.

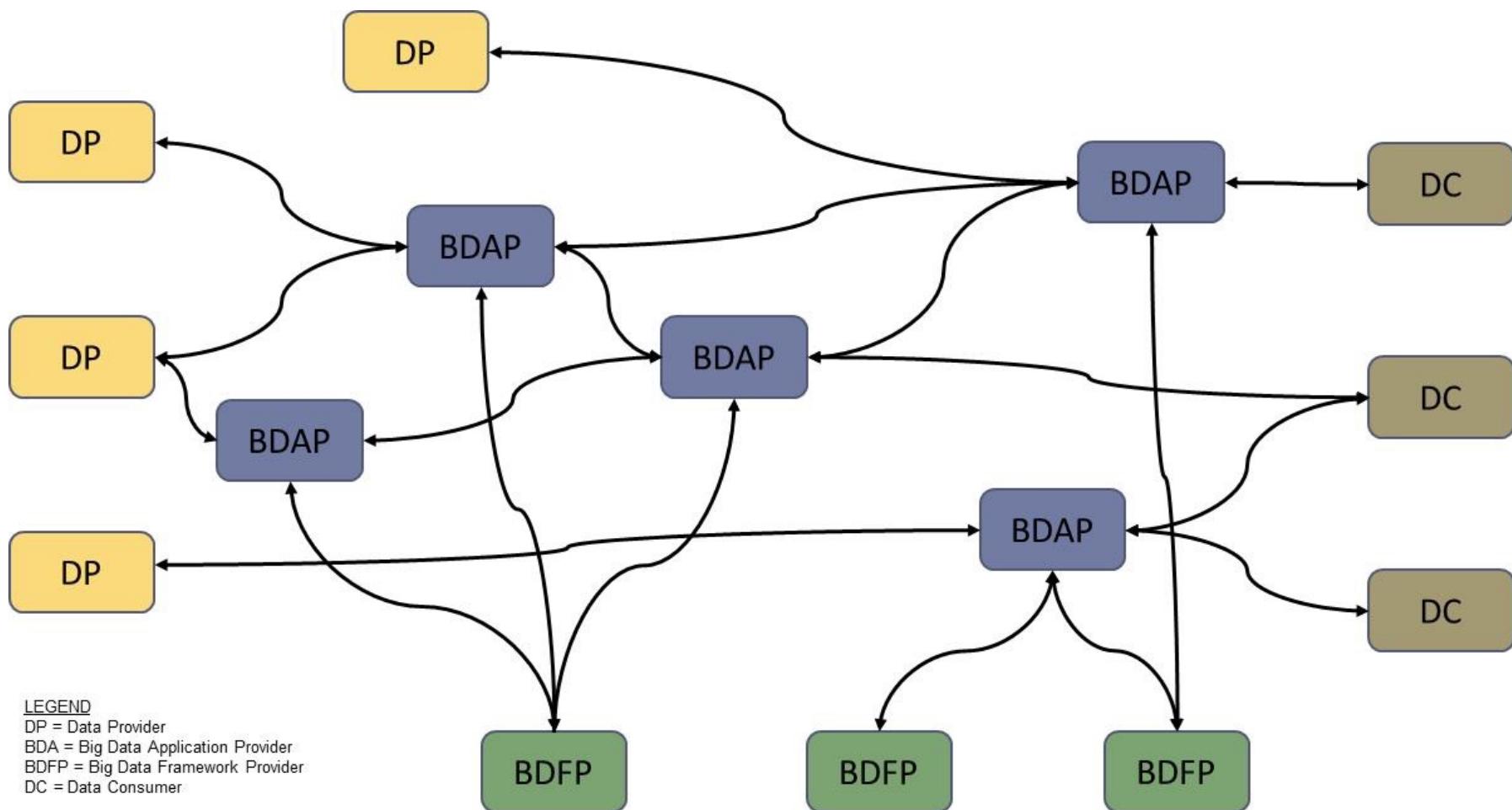
# Эталонная архитектура больших данных



Концептуальная модель NBDRA представляет систему больших данных, состоящую из пяти логических функциональных компонент, связанных интерфейсами (сервисами) взаимодействия. Эти две базовые структуры охватывают функциональные компоненты, отражая переплетение управления, безопасности и конфиденциальности со всеми пятью компонентами.

NBDRA предназначена для того, чтобы дать возможность системным инженерам, специалистам по обработке данных, разработчикам программного обеспечения, архитекторам данных и руководителям высшего звена разрабатывать решения проблем, требующих различных подходов из-за конвергенции характеристик больших данных в рамках взаимодействующей экосистемы больших данных. Эталонная архитектура обеспечивает основу для поддержки различных бизнес-сред, включая тесно интегрированные корпоративные системы и слабосвязанные вертикальные отрасли, за счет лучшего понимания того, как большие данные дополняют существующие аналитику, бизнес-аналитику, базы данных и системы, чем отличаются от них.

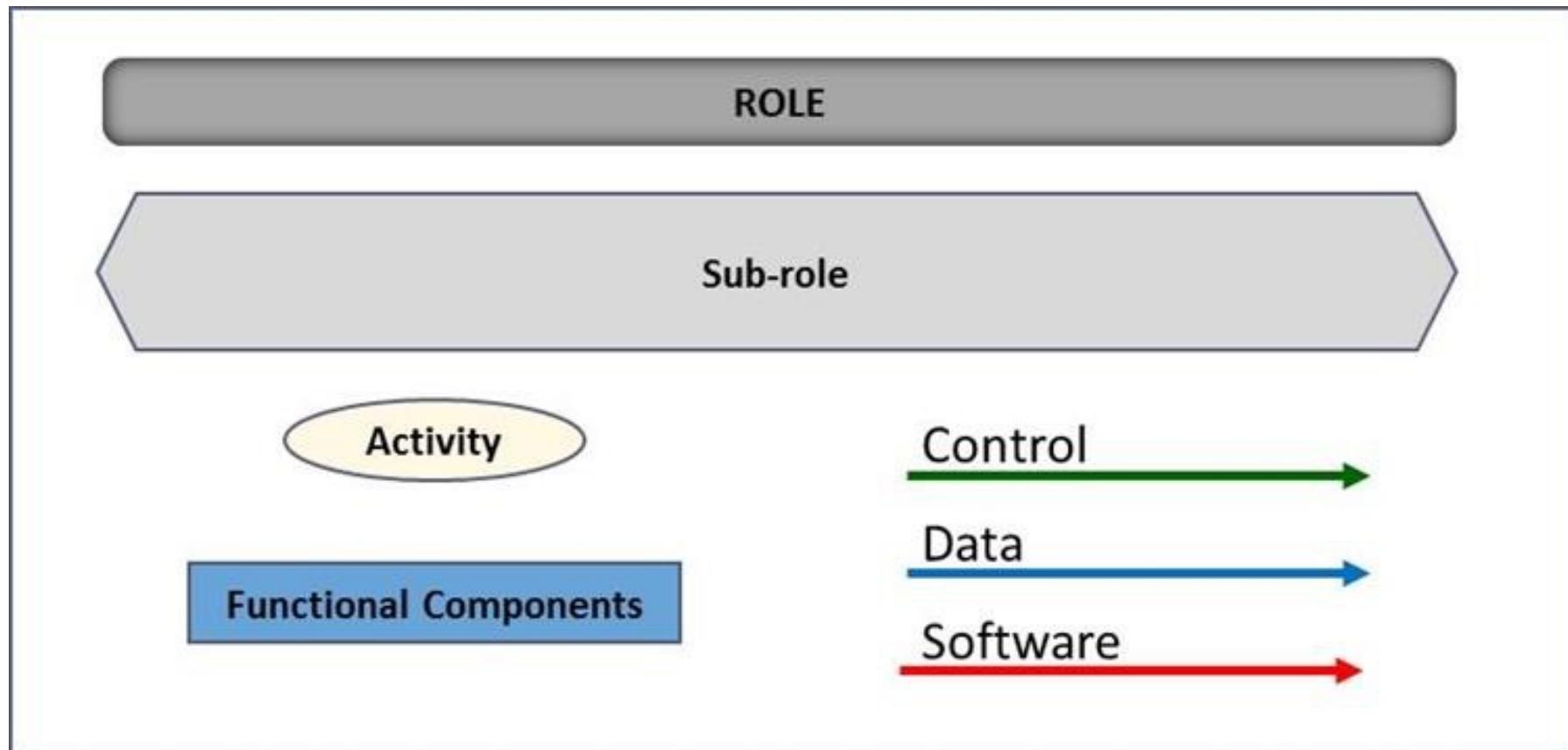
# Пример взаимодействия компонентов эталонной архитектуры больших данных



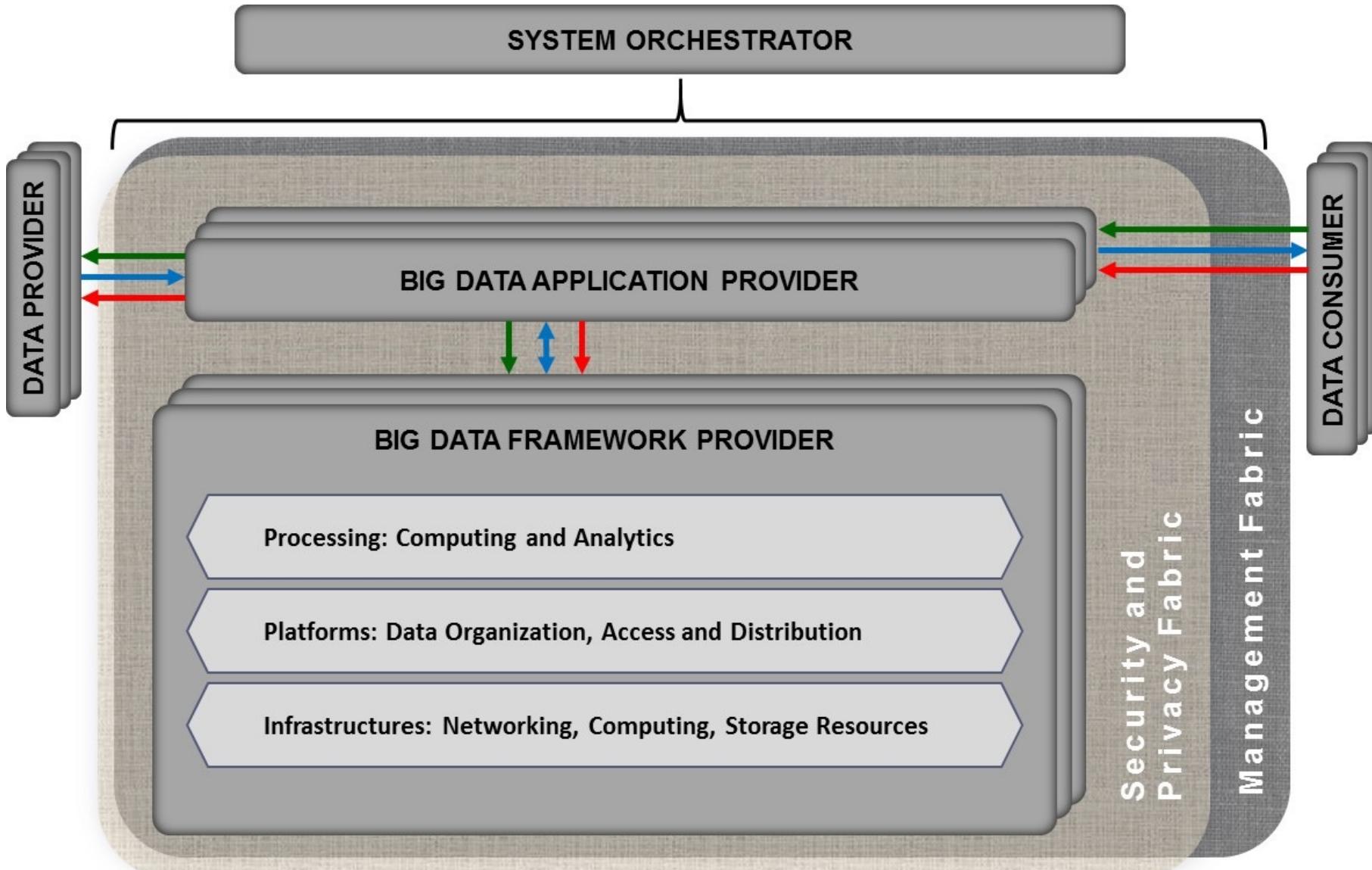
# Определения при представлении эталонной архитектуры

- **Role (Функциональность, функционал, роль):** связанный набор функций, выполняемых одним или несколькими участниками.
- **Sub-role (Узкий функционал, под-роль):** тесно связанное подмножество функций в рамках более крупной роли.
- **Activity (Действие):** класс функций, выполняемых для удовлетворения требований одной или нескольких ролей. Пример: Сбор данных - это класс действий, с помощью которых поставщик приложений для больших данных получает данные. Такими примерами могут быть сканирование Интернета, сайт, работающий по протоколу передачи файлов (FTP), веб-службы, запросы к базе данных и т.д.
- **Functional Component (Функциональный компонент):** класс физических элементов, которые поддерживают одно или несколько действий в рамках роли. Пример: Платформы потоковой обработки - это класс вычислительных платформ, которые реализуют обработку потоковых данных. Примеры таких фреймворков будут включать SPARK и STORM.

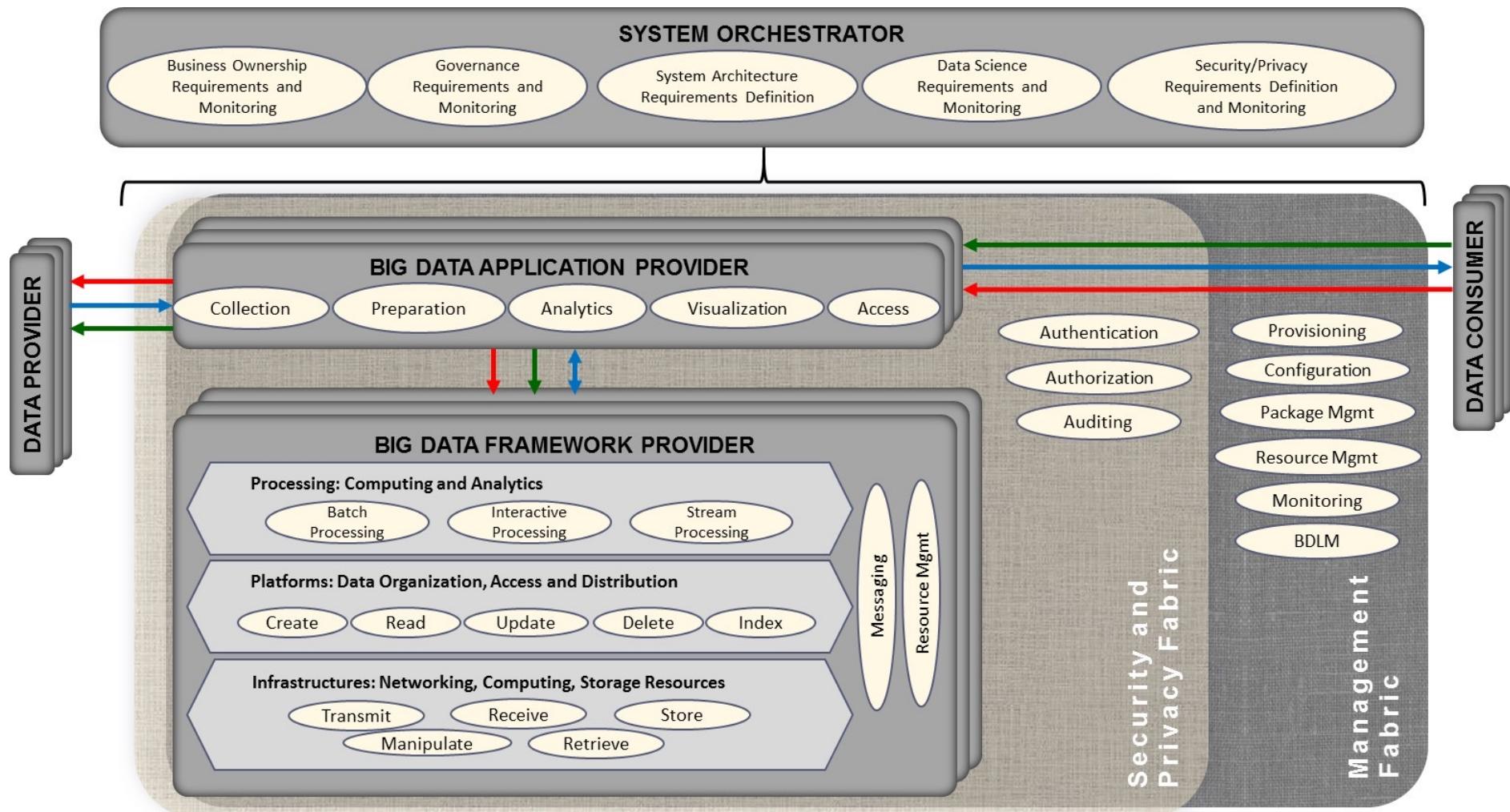
# Соглашения относительно представления эталонной архитектуры



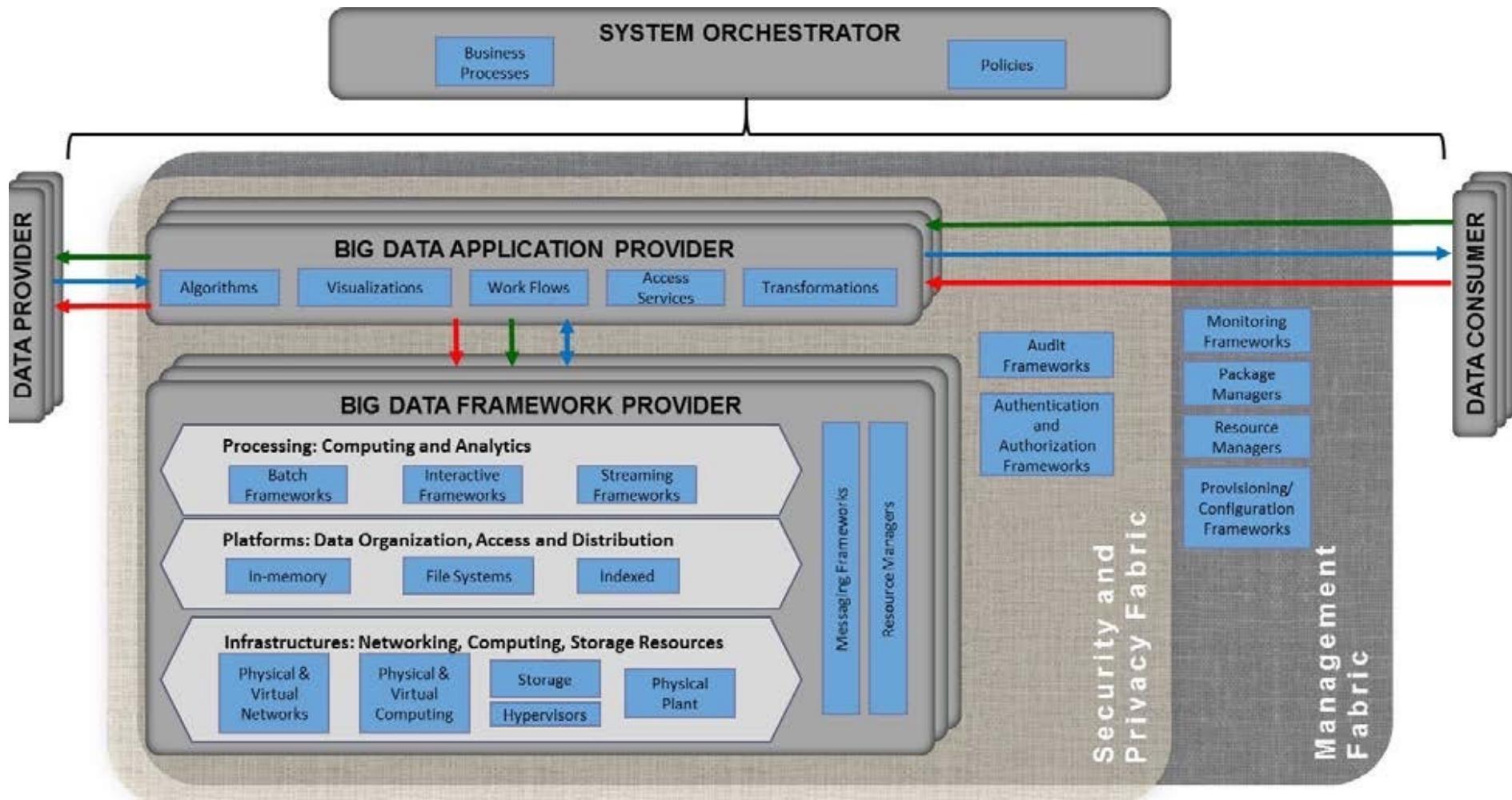
# Функциональность и базовые структуры верхнего уровня



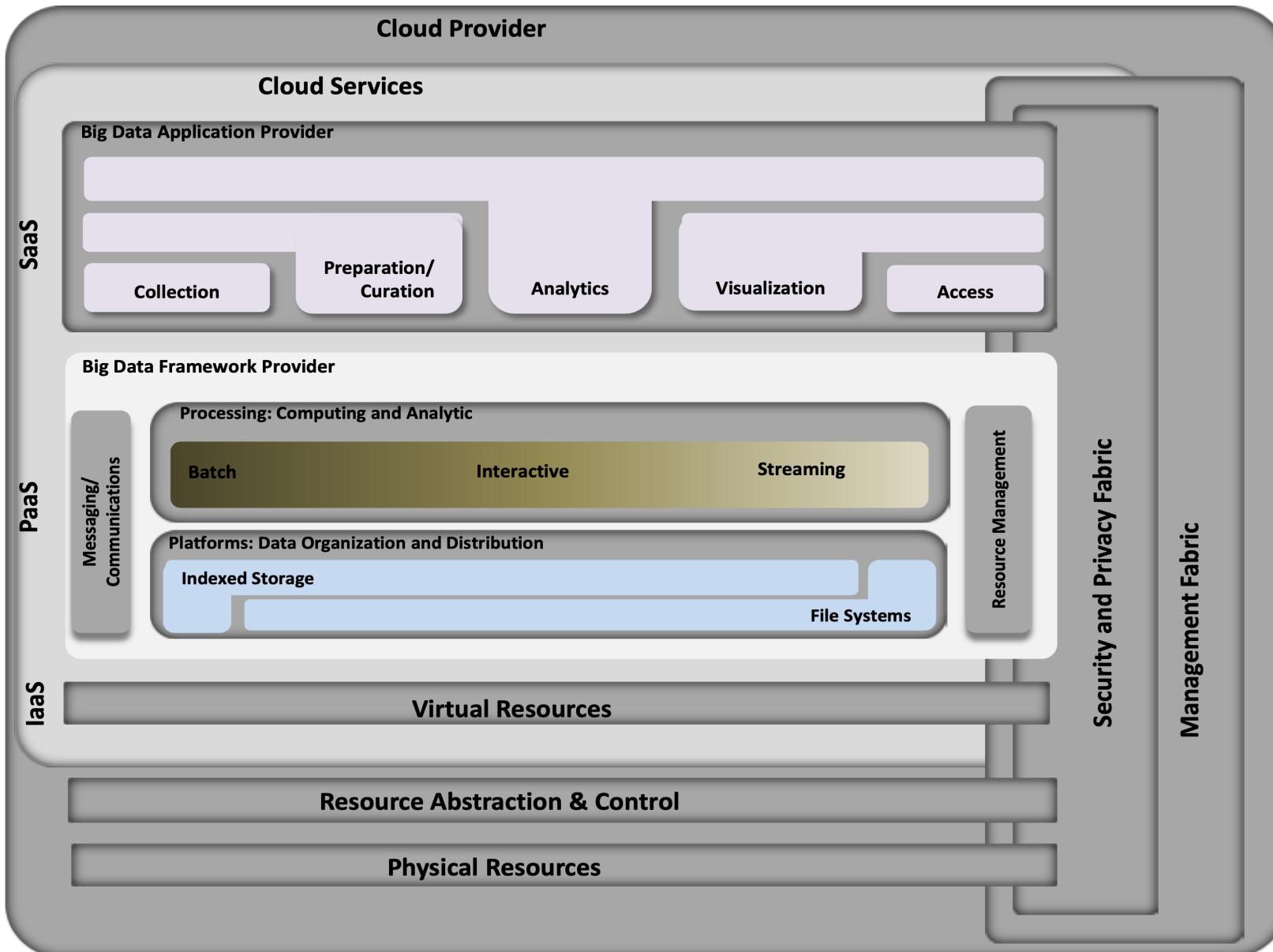
# Классы действий верхнего уровня



# Базовые классы функциональных компонентов



# Варианты развертывания инфраструктуры больших данных



# Организация процесса обработки

- системы пакетной обработки данных
- системы потоковой обработки данных
- гибридные системы обработки данных

# Системы пакетной обработки данных

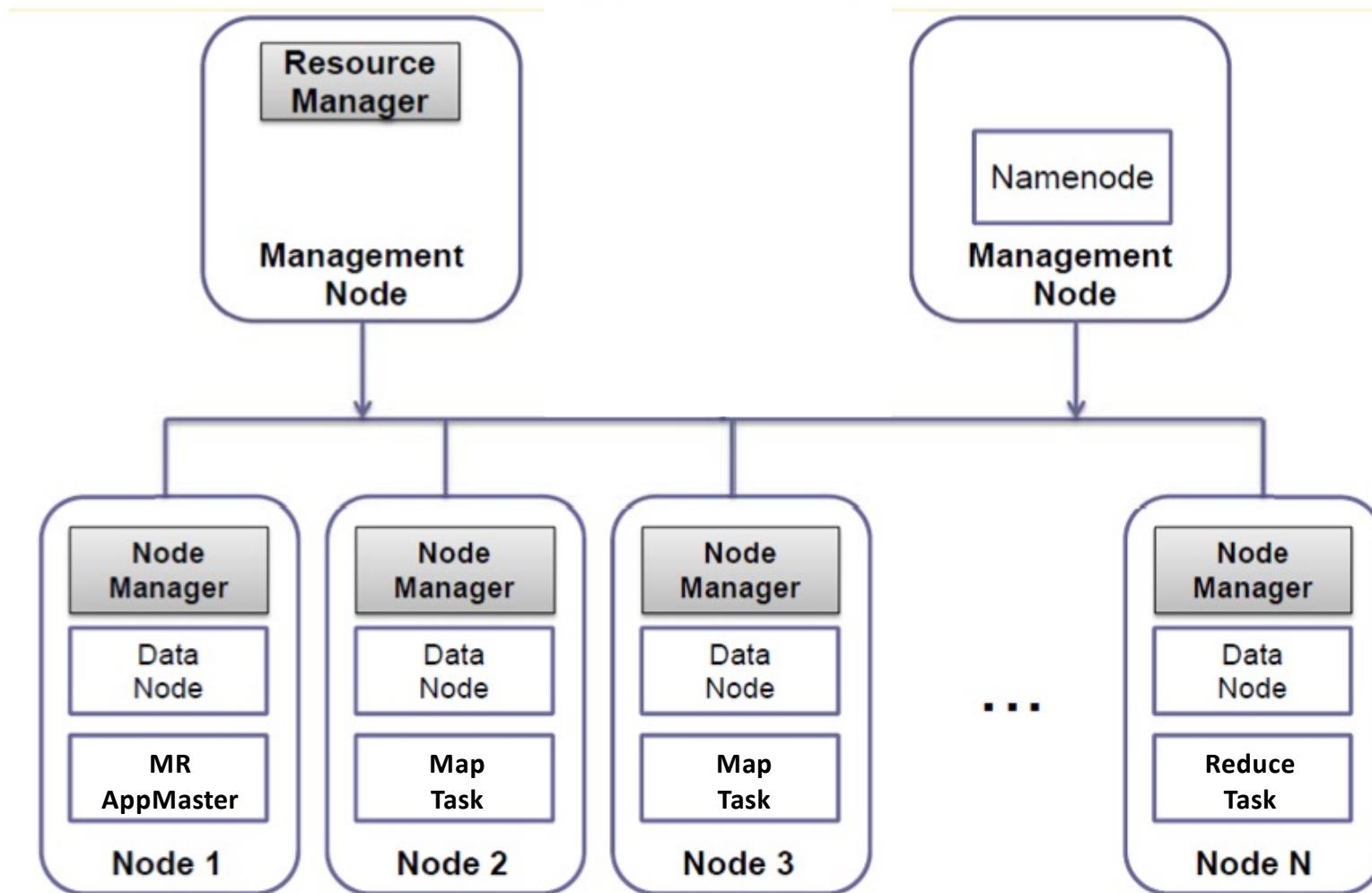
- выполняется над большим статическим набором данных в течение достаточно длительного времени
- сохраняет результат в некоторой системе постоянного хранения для предоставления его по запросу инициатора обработки в удобное для него время

Наборы данных в пакетной обработке:

- *ограничены*
- *постоянны*
- *большие*

**MapReduce**

# Архитектура Hadoop MapReduce (YARN + HDFS)

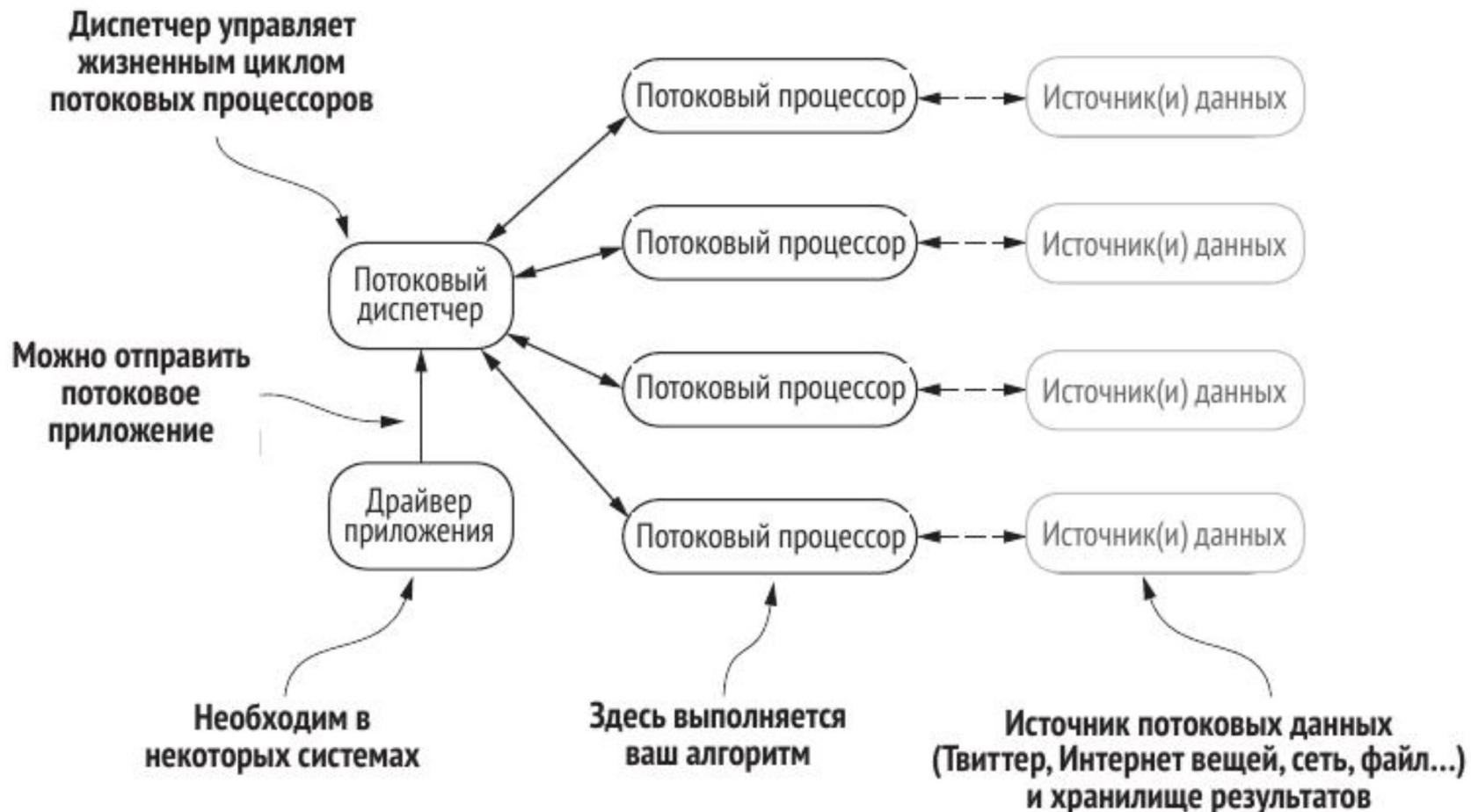


# Системы потоковой обработки данных

Наборы данных при потоковой обработке считаются "неограниченными":

- Полный набор данных определяется только тем количеством данных, которые поступили в систему на текущий момент
- Рабочий набор данных, как правило, более релевантен и ограничивается одним элементом за раз
- Обработка выполняется на основе событий и не заканчивается до явной остановки процесса. Результаты доступны немедленно и постоянно обновляются по мере поступления новых данных

# Общая архитектура систем потоковой обработки данных



# Apache Storm

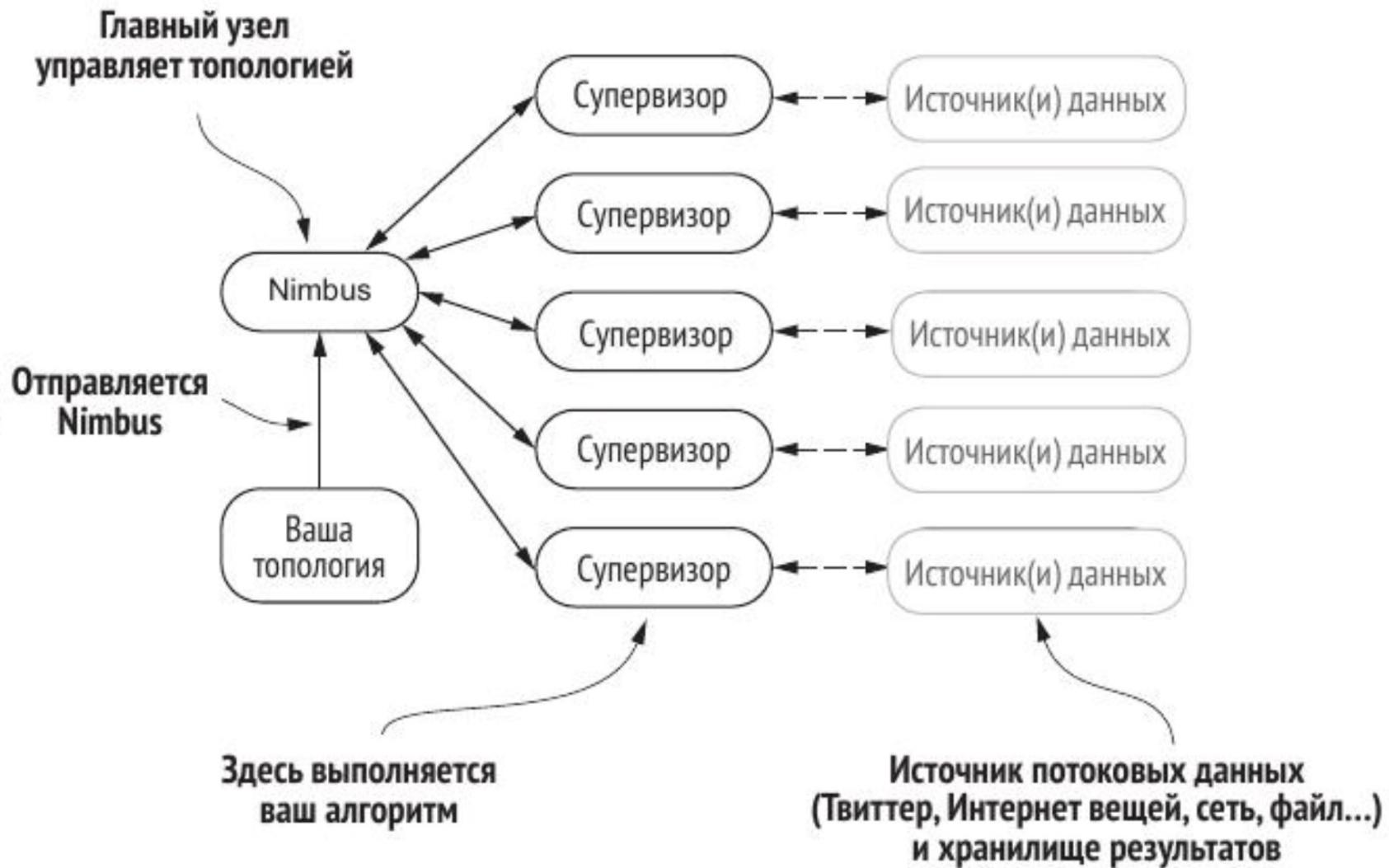


Фреймворк распределённой потоковой обработки, отличительной особенностью которого является обеспечение ***крайне низкой латентности*** процессов обработки.

Фреймворк является наилучшим вариантом для рабочих нагрузок, требующих ***обработки в режиме близком к реальному времени***.

Он может обрабатывать очень большие объемы данных и предоставлять результаты с меньшими задержками, чем другие решения.

# *Storm: архитектура верхнего уровня*



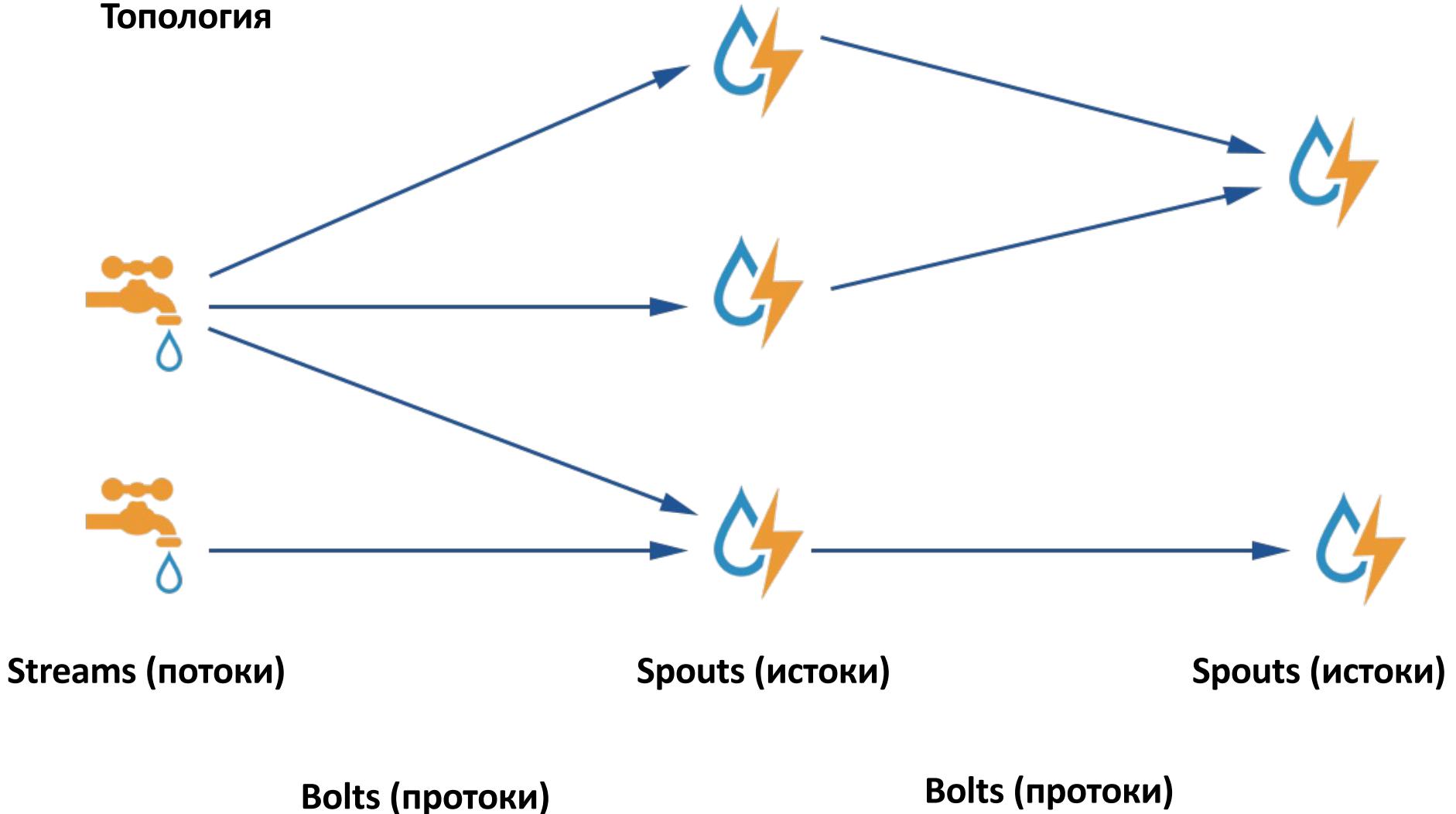
# **Storm: модель потоковой обработки данных**

Процесс обработки потоков – направленный ациклический граф (DAG – Directed Acyclic Graph). Обработка потоков работает путем оркестровки задаваемого графа, который называется **топологией**. Топология использует следующие термины:

- **Streams (потоки)** – обычные потоки данных. Это неограниченная последовательность данных, постоянно поступающих в систему.
- **Spouts (истоки)** – источники потоков данных на границе топологии. Это могут быть внешние потоки данных, доступ к которым реализуется в соответствии с определёнными программными интерфейсами, очереди и т. д.
- **Bolts (протоки)** – шаг обработки, который потребляет потоки, применяет к ним заданную операцию и выводит результат в виде потока. Протоки соединяются с каждым из истоков, а затем соединяются друг с другом, чтобы организовать всю необходимую обработку. На границе топологии конечный выходной протокол может использоваться в качестве входных данных для подключенной сторонней системы.

# *Storm*: модель потоковой обработки данных

Топология



# **Storm: Trident vs. Core Storm**

- По умолчанию Storm обеспечивает стратегию обработки «по крайней мере один раз».
- Trident – обработка с отслеживанием состояния по стратегии «ровно один раз».

В топологии Trident используются следующие абстракции:

- Потоковые пакеты – это микро-пакеты потоковых данных, которые фрагментированы для обеспечения семантики пакетной обработки.
- Операции – это пакетные процедуры, которые выполняются над данных.

**Семантика доставки сообщений:**

- Не более одного раза (может потеряться)
- Не менее одного раза (не может потеряться)
- Ровно один раз (не может потеряться)

# Apache Samza

- Асинхронная вычислительная Big Data-среда с открытым исходным кодом для распределенных потоковых вычислений практически в реальном времени.
- Разработана в 2013 году в соцсети LinkedIn на языках Scala и Java. Проектом верхнего уровня Apache Software Foundation Самза стала в 2014 году
- Фреймворк потоковой обработки, который тесно связан с системой обмена сообщениями Apache Kafka.
- Несмотря на то, что система Apache Kafka может быть использована и другими системами потоковой обработки, фреймворк Samza разработан специально, чтобы воспользоваться уникальной архитектурой системы Kafka и её моделью гарантированности обработки сообщений.
- Samza использует Kafka для обеспечения отказоустойчивости, буферизации и хранения состояния.
- Для диспетчеризации заданий и управления ресурсами фреймворк Samza использует Hadoop-компонент YARN.

# Apache Samza: принцип действия

- Считывает данные из входных потоков (Input Stream), ассоциированных с топиками Кафка
- Записывает их в логи изменений (Changelog Stream) и выходные потоки (Output Stream), также ассоциированные с соответствующими топиками

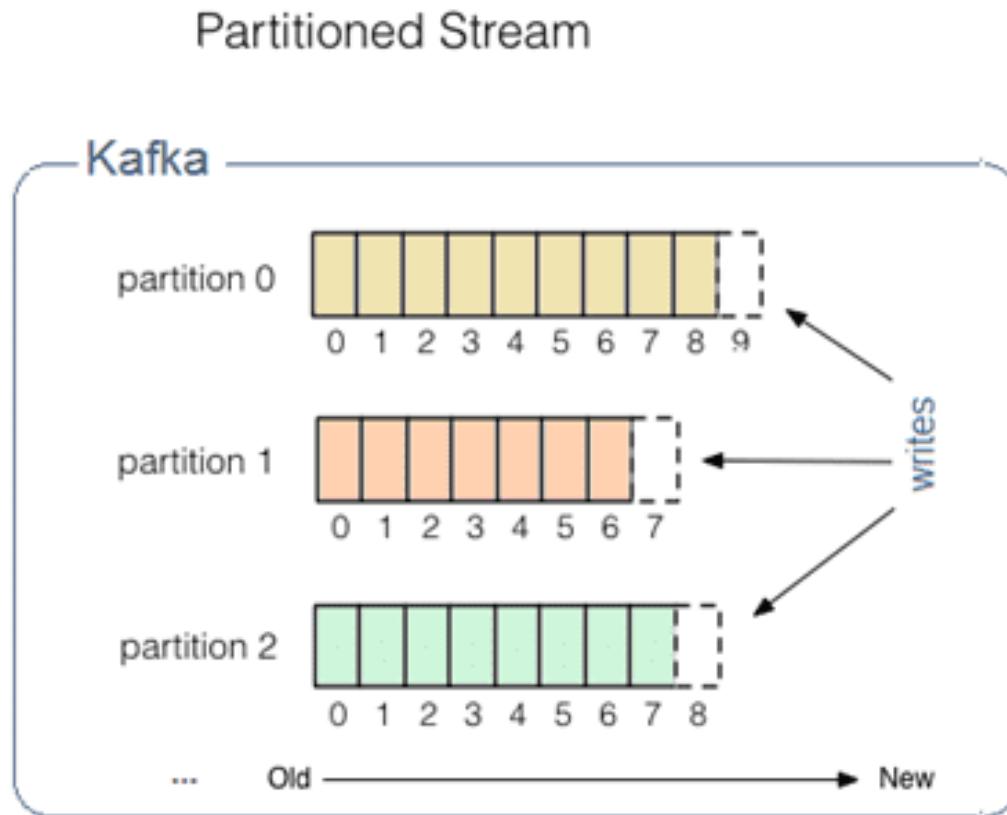
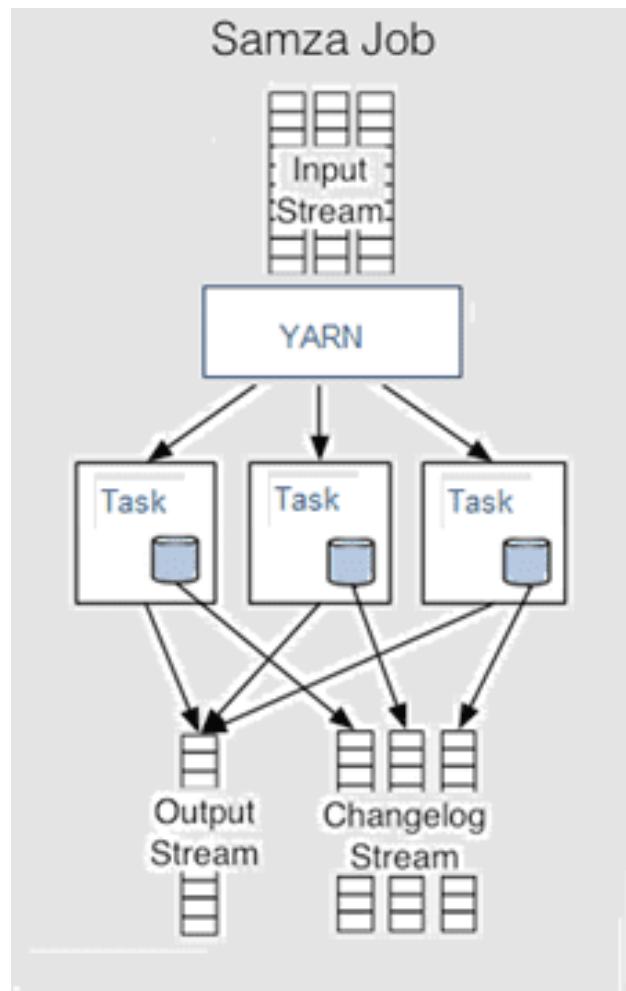
Модель параллелизма:

- работа подразделяется на независимые задачи, которые могут выполняться параллельно.
- распределение ресурсов не зависит от количества задач: небольшая работа может хранить все задачи в одном процессе на одном компьютере, а большая работа может распределить задачи по многим процессам на многих машинах

# Архитектура Samza



# Архитектура Samza



samza

# *Samza: модель обработки Kafka*

- Темы (Topics) – входной поток данных – поток связанной информации, на который могут подписаться потребители.
- Разделы (Partitions) – распределяют тему между узлами. Разбиение – по ключу, который гарантирует, что каждое сообщение с одним и тем же ключом будет отправлено в тот же раздел. Разделы гарантированно упорядочены.
- Брокеры (Brokers ) – отдельные узлы, составляющие кластер, на котором развёрнута система Kafka.
- Производитель (Producer) – любой компонент, формирующий тему, предоставляет ключ, используемый для секционирования темы на разделы.
- Потребители (Consumers) – любой компонент, который читает из темы. Хранят информацию о своем собственном смещении, знают, какие записи были обработаны, при обработке сбоев.

# Параллелизм Samza vs. Storm

- Storm использует один поток на задачу по умолчанию, тогда как Samza использует однопоточные процессы (контейнеры).
- Контейнер Samza может содержать несколько задач, но есть только один поток, который вызывает каждую из задач по очереди. Это означает, что каждый контейнер сопоставлен ровно с одним ядром ЦП, что значительно упрощает модель распределения ресурсов и уменьшает помехи от других задач, выполняемых на той же машине.
- Преимущество многопоточной модели Storm в том, что она лучше использует избыточную емкость на пристаивающей машине за счет менее предсказуемой модели ресурсов

# Гибридные системы обработки

- **Spark:** Пакетная обработка – базовая, потоковая – на основе микро-пакетов.
- **Flink:** Потоковая обработка – базовая, пакетная – на основе потоков с конечными границами.
- Фреймворк **Flink** – уникальный вариант среди платформ обработки данных: потоковая модель Flink обеспечивает низкую латентность, высокую пропускную способность и реальную поэлементную обработку.
- **Spark** также выполняет пакетную и потоковую обработку, но его потоковая модель не подходит во многих случаях из-за используемой архитектуры микро-пакетов.

# Компоненты Spark и стек API

Spark SQL and  
DataFrames +  
Datasets

Spark Streaming  
(Structured  
Streaming)

Machine Learning  
MLlib

Graph  
Processing  
Graph X

Spark Core and Spark SQL Engine

Scala

SQL

Python

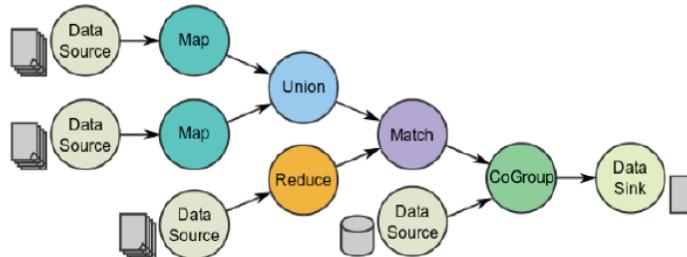
Java

R

Spark – полиглотная технология

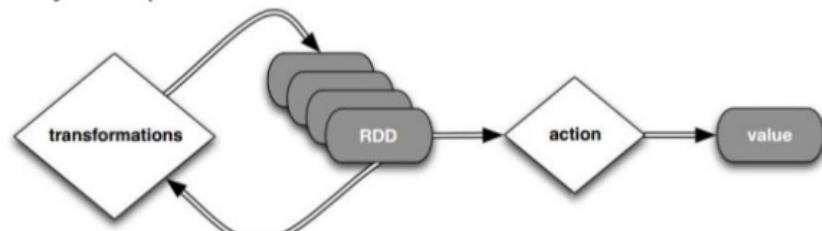
## Программная модель Spark

- Задача описывается с помощью **directed acyclic graphs (DAG)**

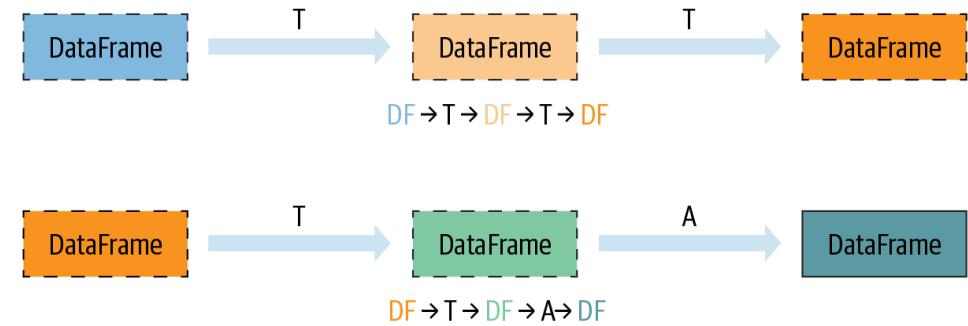


RDD

Lazy computations model

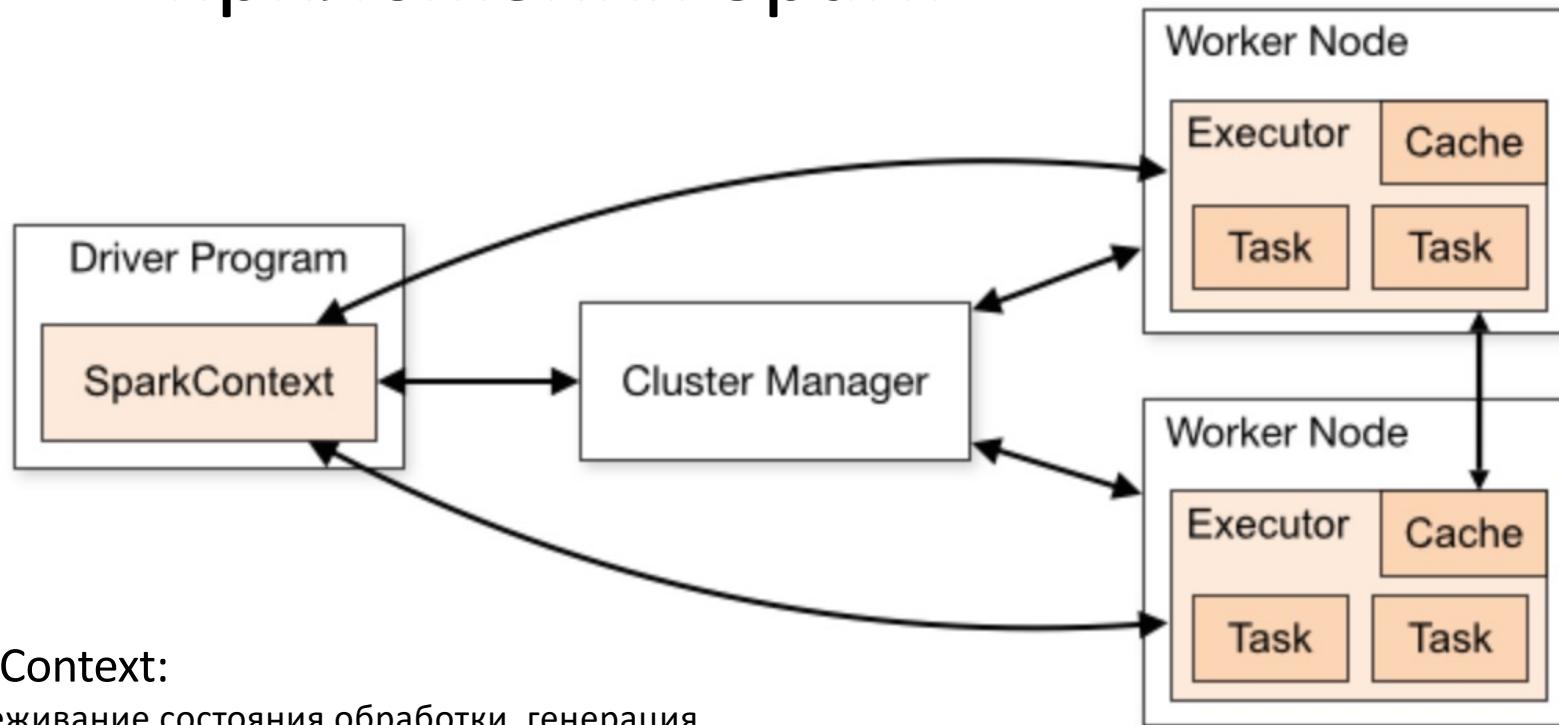


Transformation cause only metadata change



T = Transformation A = Action

# Архитектура распределённого приложения Spark



## SparkContext:

- отслеживание состояния обработки, генерация задач и их перезапуск в случае отказов
- оркестровка вычислений – назначение задач на вычислительные ресурсы (занимаемые с помощью executor'ов) с учетом их использования данных
- контроль за вычислительными ресурсами, включая их выделение и возвращение менеджеру ресурсов

## Cluster Manager:

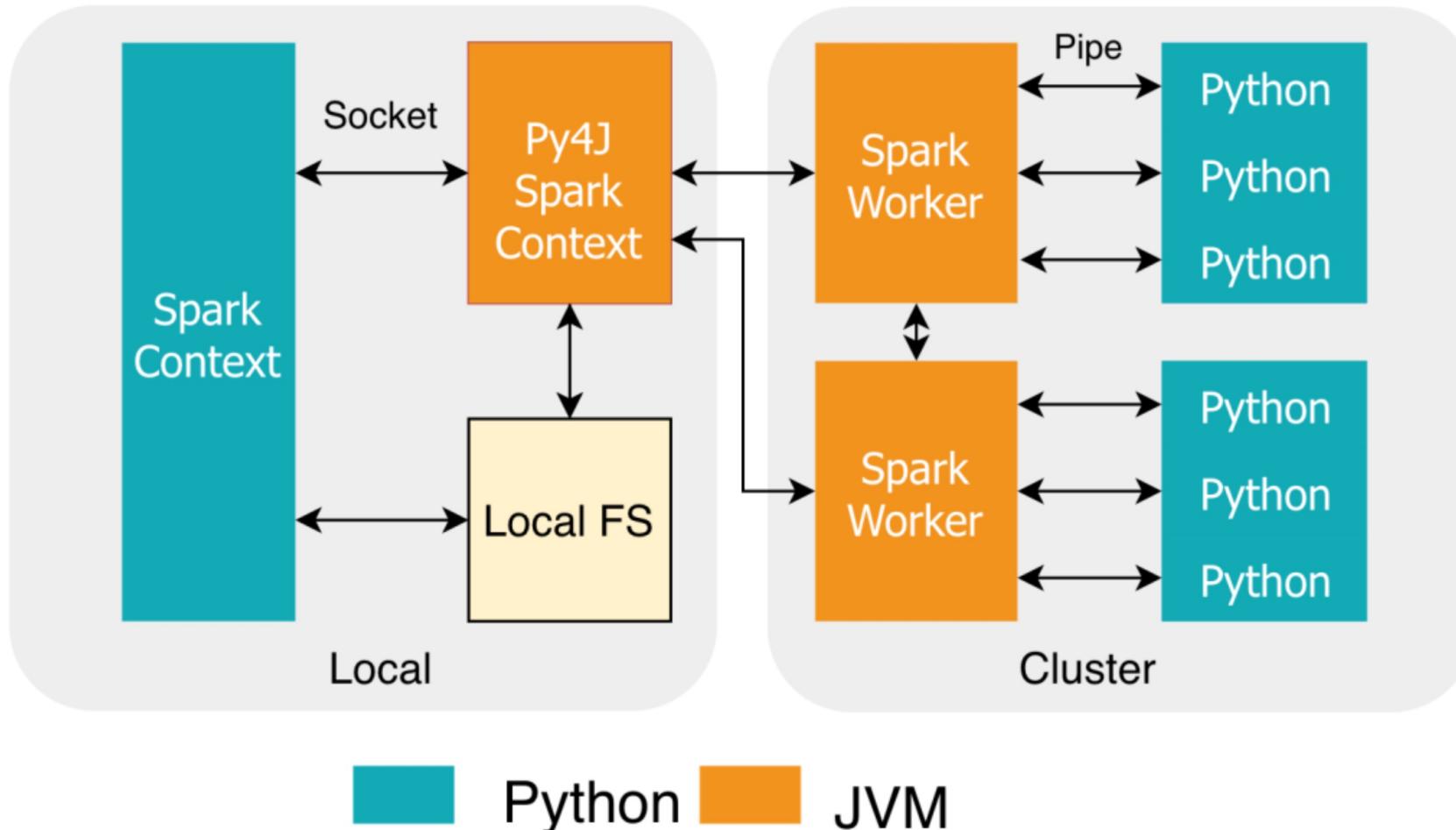
- внешний сервис для запроса ресурсов кластера
- standalone manager, Mesos, YARN, Kubernetes

## Executor:

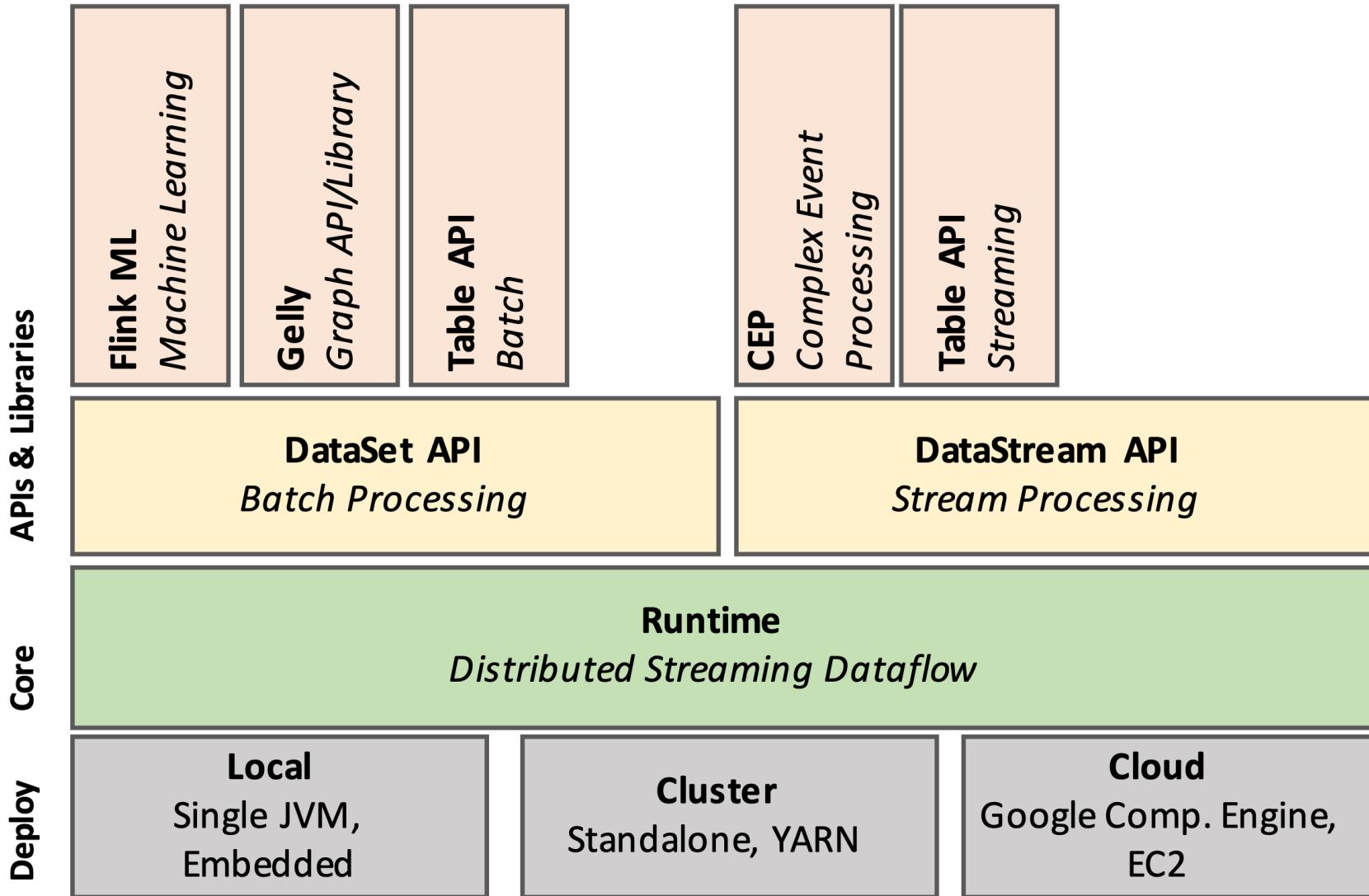
- процесс, запущенный для приложения на рабочем узле, каждое приложение имеет своих собственных исполнителей
- запуск выполнения задач и сохранение их результатов
- контроль за расходом вычислительных ресурсов
- хранение данных и предоставление доступа к ним для выполнения обработки (включая загрузку данных с диска или их десериализацию)

# Общая архитектура PySpark

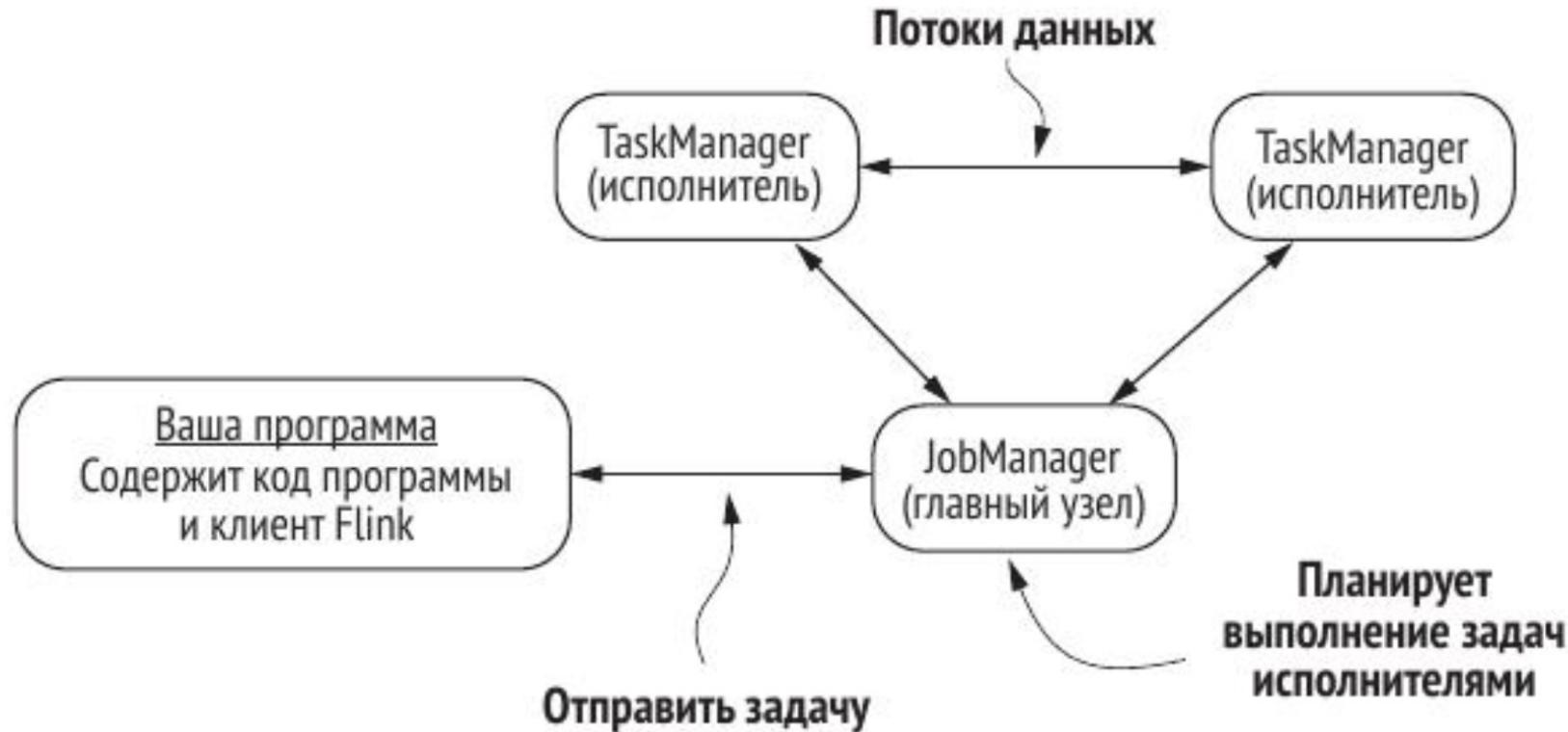
## Data Flow



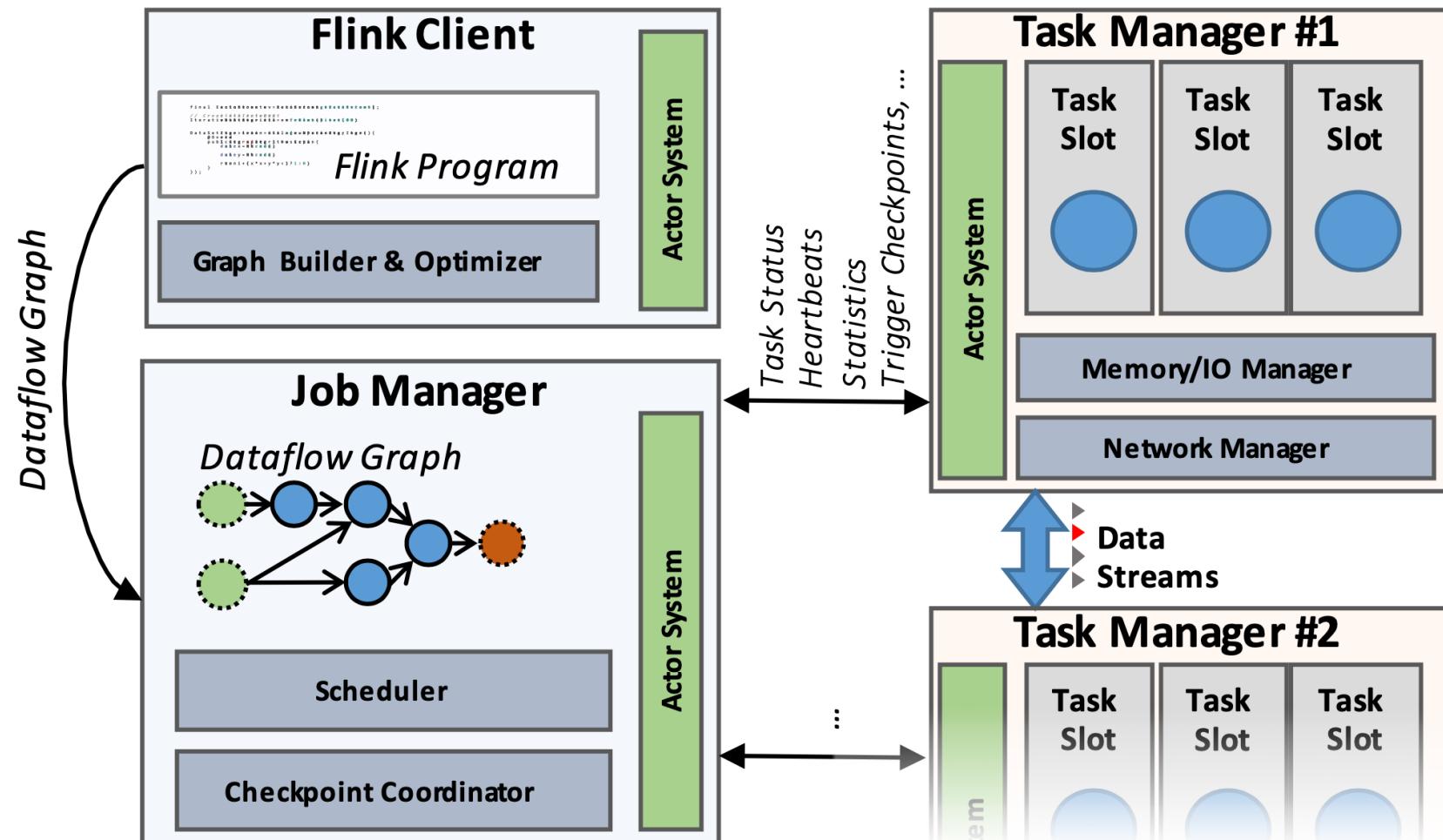
# Apache Flink: стек программного обеспечения



# Apache Flink: архитектура верхнего уровня



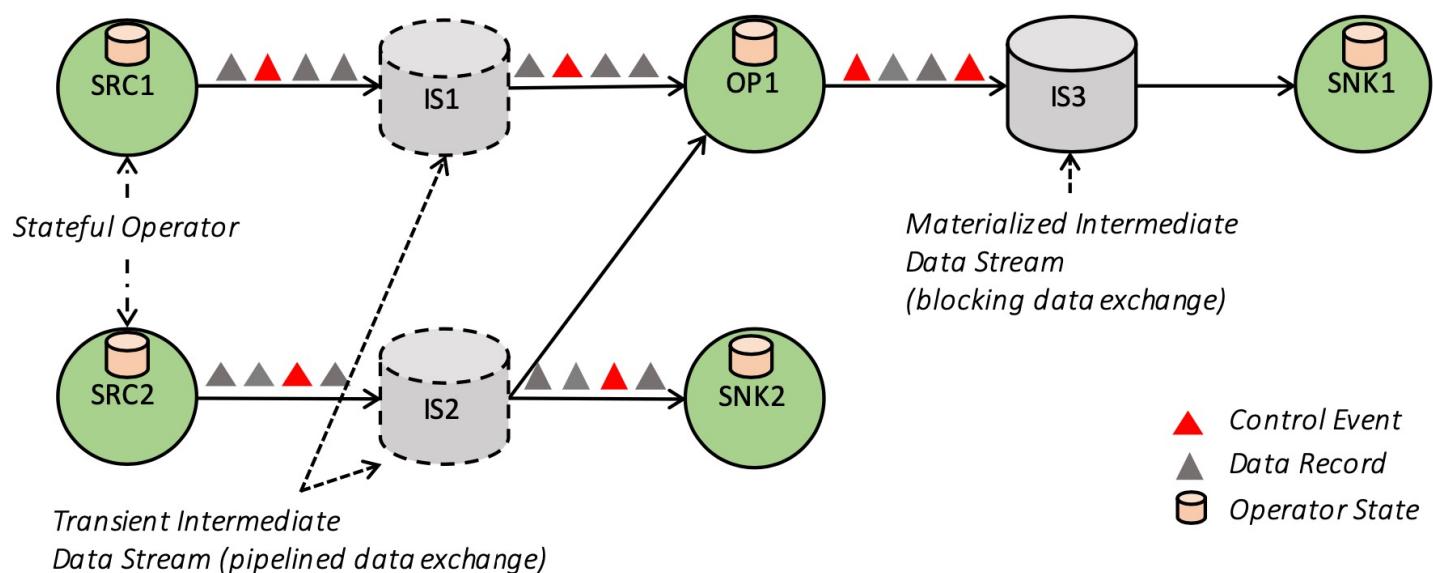
# Apache Flink: модель процессов



# Apache Flink

Основные составляющие операционной модели Flink:

- **Потоки (Streams)** являются неизменяемыми, неограниченными наборами данных, которые протекают через систему.
- **Операторы (Operators)** – это функции, которые преобразовывают данные потоков, создавая тем самым другие потоки.
- **Источники (Sources)** – это точки входа для потоков, поступающих в систему.
- **Стоки (Sinks)** – это место, где потоки вытекают из системы Flink. Они могут представлять базу данных или соединитель со внешней системой.



- Задачи потоковой обработки делают снимки текущего состояния в заданных точках в процессе своих вычислений, чтобы использовать их для восстановления в случае сбоев. При сохранении состояния Flink может работать с несколькими копиями состояния, зависящими от различных уровней сложности и персистентности.
- *Потоковая обработка Flink* реализует концепцию "время события", означающее время, когда действительно произошло событие, может обрабатывать сессии и гарантировать упорядочение и группировку по времени различными способами.
- Модель *пакетной обработки Flink* во многих отношениях является лишь расширением модели потоковой обработки: вместо чтения из непрерывного потока считывается ограниченный набор данных из постоянного хранилища в виде потока.
- Flink использует одинаковую среду выполнения для обеих моделей обработки.
- Оптимизация для процессов пакетной обработки: не используется функциональность снимков текущего состояния. Отказоустойчивость при этом сохраняется, но обычная обработка завершается быстрее.