

СИСТЕМЫ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

Разработка приложений



К.Т.Н.
Папулин Сергей Юрьевич
papulin_bmstu@mail.ru

Лекция 9. Обработка графов

Apache Giraph



- Компьютерные сети
- Социальные сети
- Семантические графы
- Карты
- Маршруты

- Вычисление кратчайшего пути
- Определение наиболее значимых узлов с PageRank
- Предсказание рейтингов
- Определение сообществ

Apache Giraph

Основные темы

- Назначение и особенности
- Модель данных
- API
- Архитектура
- Запуск вычислений
- Отказоустойчивость

➤ Vertex

➤ Edge

➤ Vertex State

➤ BSP model

➤ Superstep

➤ Partition

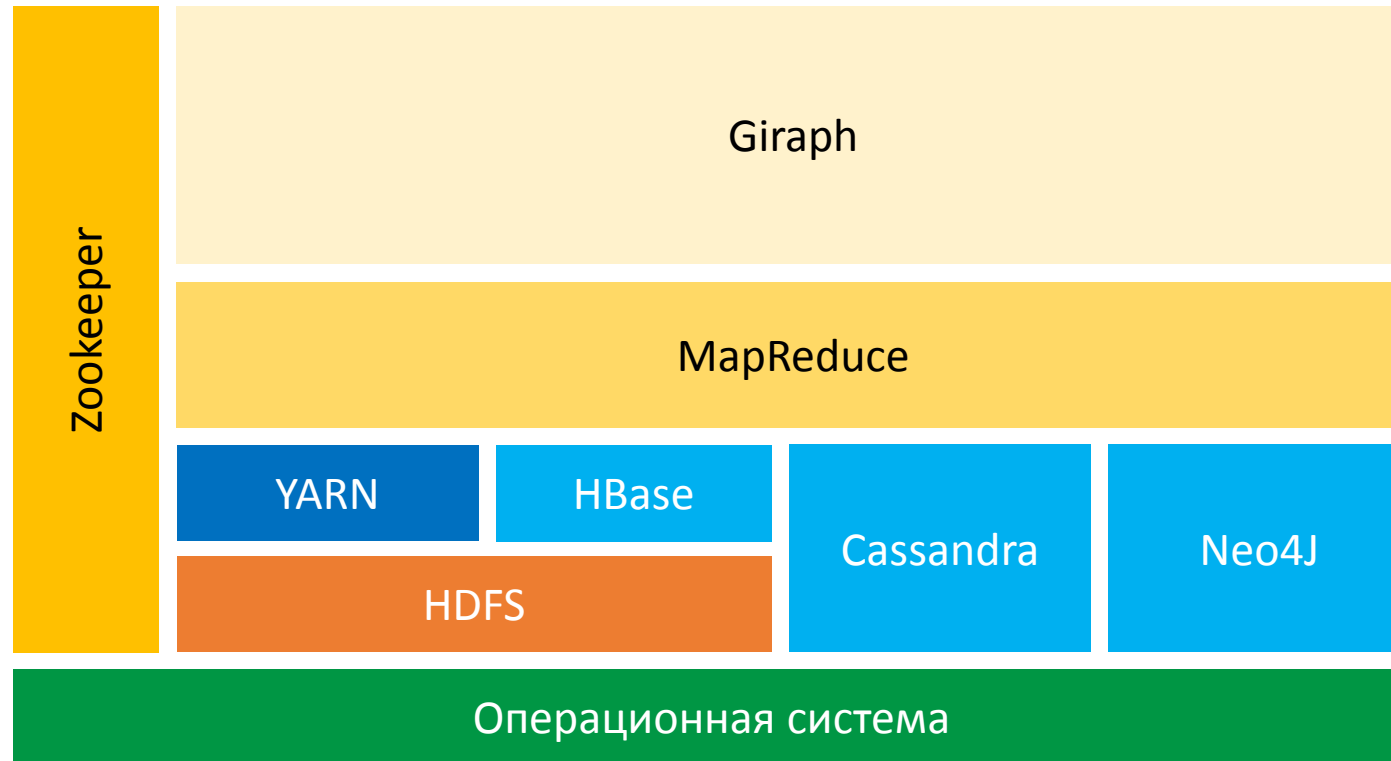
➤ Master

➤ Worker

➤ Coordination service

Назначение и особенности

- Giraph предназначен для реализации итеративных алгоритмов над графами
- с возможностью распараллеливания процессов обработки
- отказоустойчив и
- способен обрабатывать графы больших размеров
- Используется для batch вычислений (запускается после предварительного накопления значительного объема исходных данных)
- Не предназначен для быстрых интерактивных ad hoc запросов



Программная модель

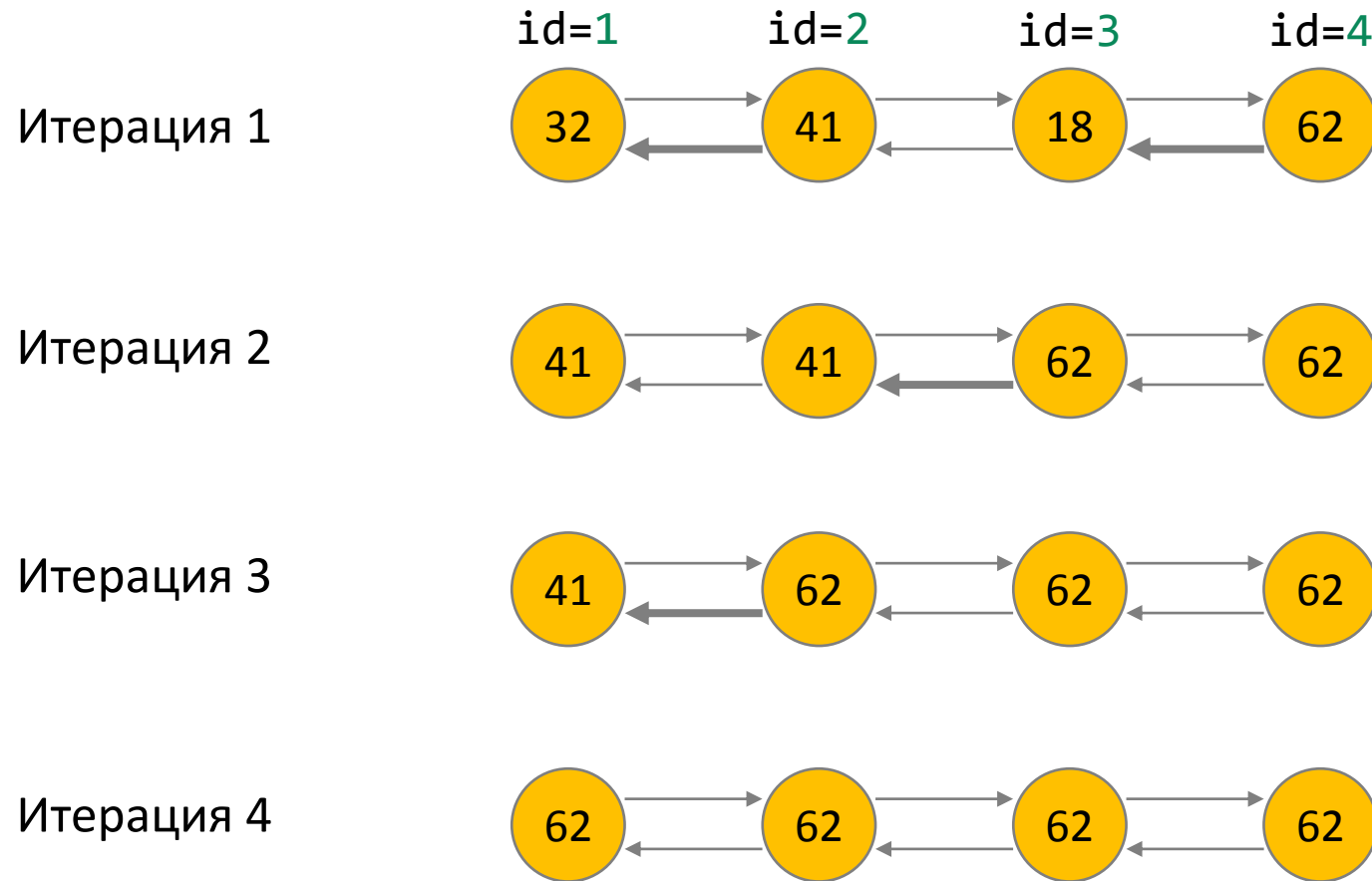
- Giraph использует программную модель, ориентированную на вершины
- Разработчику необходимо написать код, который будет выполняться на каждой вершине
- Программная модель скрывает сложности программирования параллельной и распределенной системы
- Giraph выполняет ваш код прозрачно на множестве доступных машин в параллельной манере

Определяемые пользователем функции (UDF)

Giraph API

Распределенное представление графа и движок
выполнения

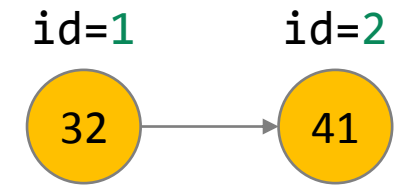
Пример. Поиск наибольшего значения



Каждая вершина имеет:

- уникальный идентификатор (id): integer, string и пр.
- значение: integer, string и пр.
- некоторое количество исходящих ребер, которые указывают на другую вершину

Каждое ребро может иметь значение



Вершина в Giraph

```
class Vertex:
```

```
    function getId() # id вершины
```

```
    function getValue() # значение вершины
```

```
    function setValue(value) # установка значения вершине
```

```
    function getEdges() # все исходящие ребра
```

```
    function getNumEdges() # количество исходящих ребер
```

```
    # значение первого ребра, соединенного с вершиной targetId
```

```
    function getEdgeValue(targetId)
```

```
    # присвоение значения первому ребру, соединенному с вершиной targetId
```

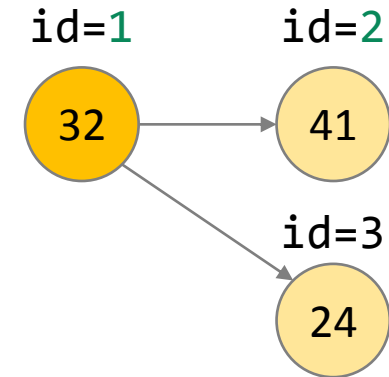
```
    function setEdgeValue(targetId, value)
```

```
    function getAllEdgeValues(targetId) # все значения ребер, соединенных с вершиной targetId
```

```
    function voteToHalt() # остановка вычисления для вершины
```

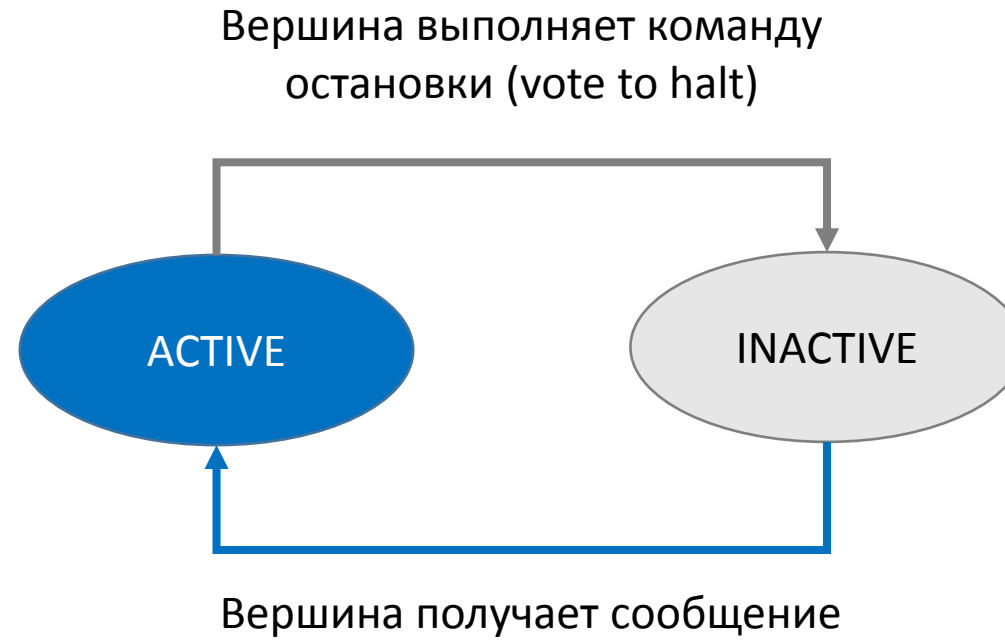
```
    function addEdge(edge) # добавление ребра
```

```
    function removeEdges(targetId) # удаление всех ребер, соединенных с вершиной targetId
```

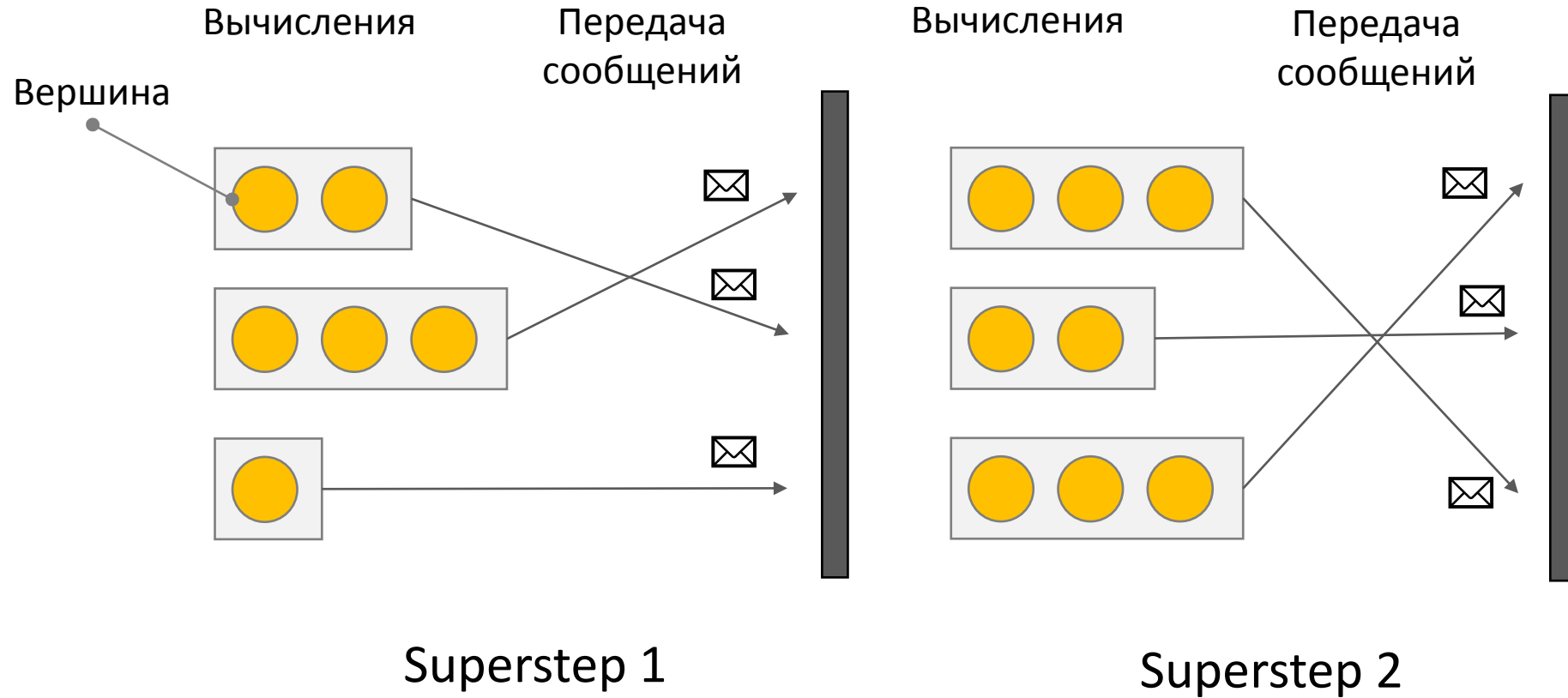


```
class Edge:  
    function getTargetVertexId() # id целевой вершины (с которой соединен)  
    function getValue() # значение ребра  
    function setValue(value) # установка значения ребра
```

Состояния вершины

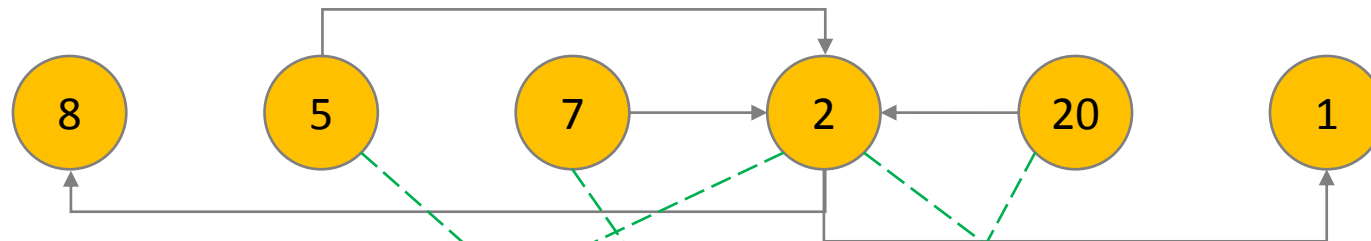


Синхронизирующая модель Giraph

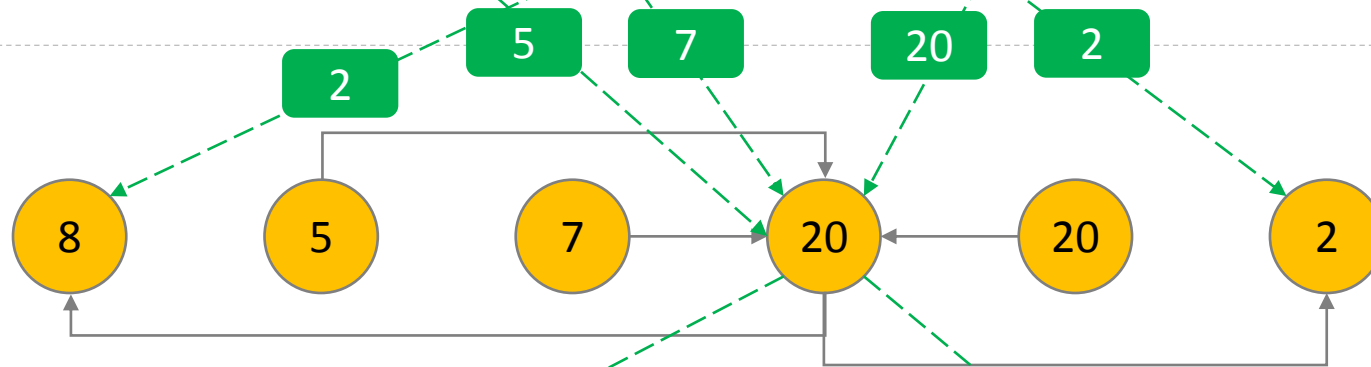


Пример

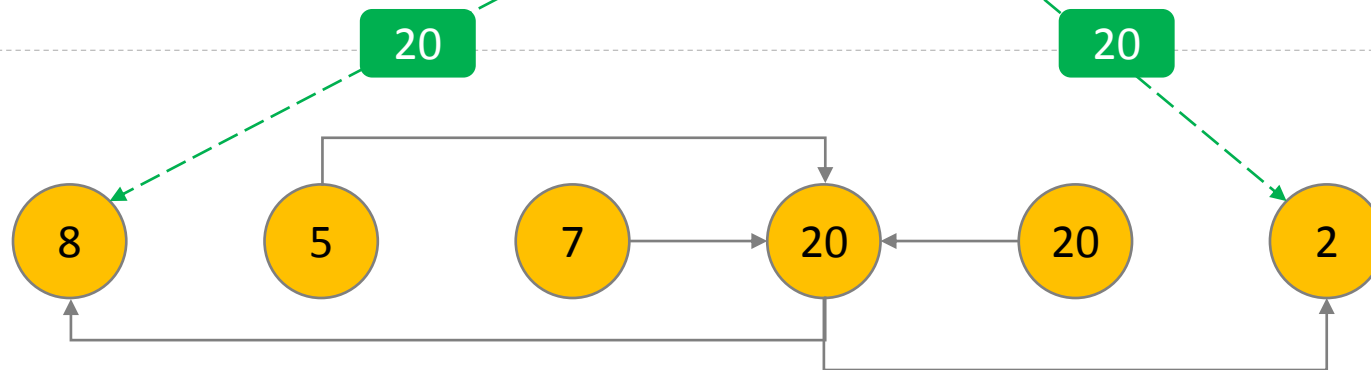
Superstep 1



Superstep 2



Superstep 3

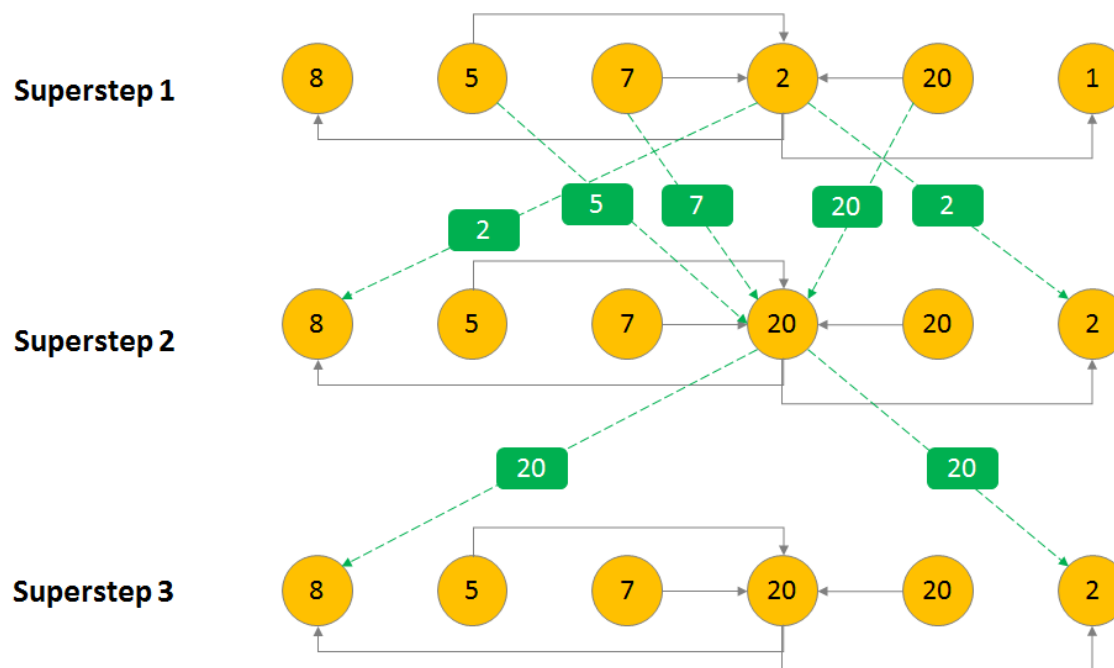


На каждом superstep'е вызывается метод compute() на всех активных вершинах (обрабатываются сообщения отправленные на предыдущем superstep'е)

```
class BasicComputation:
    function compute(vertex, messages) # определяемая пользователем обработка
    function getSuperstep() # текущий superstep
    function getTotalNumVertices() # суммарное количество вершин в графе
    function getTotalNumEdges() # суммарное количество ребер в графе
    function sendMessage(targetId, message) # суммарное количество вершин в графе
    function sendMessageToAllEdges(vertex, message) # отправка сообщения вершине
    vertex
    function addVertexRequest(vertexId) # запрос на добавление вершины в граф
    function removeVertexRequest(vertexId) # запрос на удаление вершины из графа
```

Пример compute

```
function compute(vertex, messages):
    maxValue = max(messages) # определение максимального значения
    if maxValue > vertex.getValue(): # сравнение с текущим значением вершины
        vertex.setValue(maxValue) # установка нового значения текущей вершины
        sendMessageToAllEdges(vertex, maxValue) # отправка сообщения всем связанным
        вершинам
    vertex.voteToHalt() # остановка текущей вершины (переход в неактивное состояние)
```

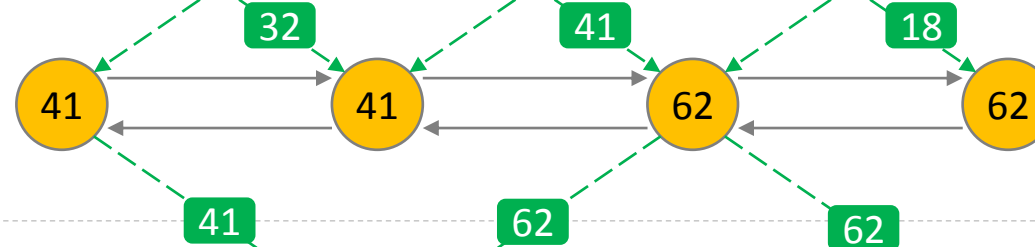


Пример compute

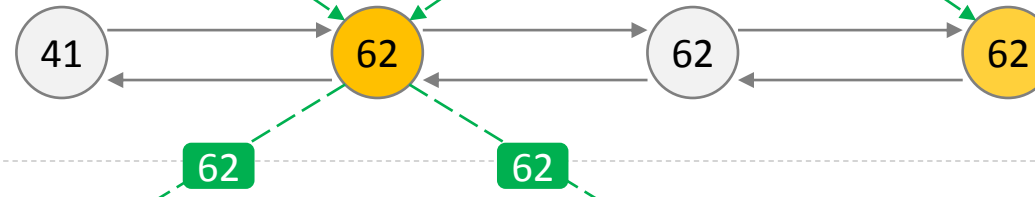
Superstep 1



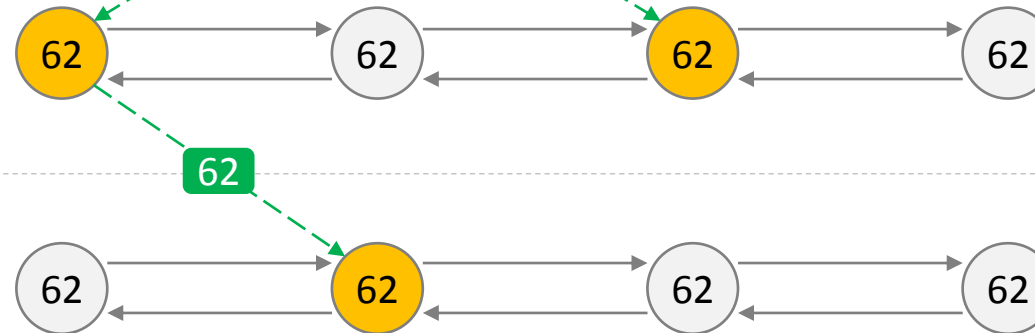
Superstep 2



Superstep 3

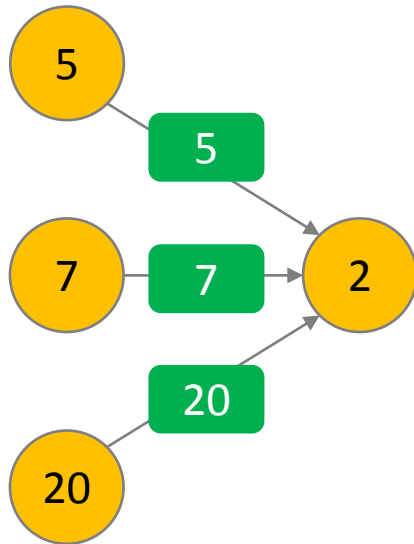


Superstep 4

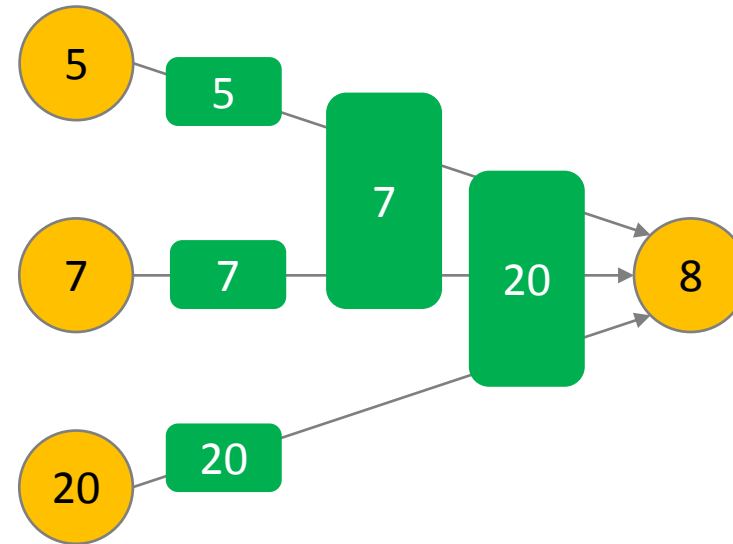


Combiner

```
class MessageCombiner:
    # возвращает комбинацию двух сообщений
    function combine(id, message1, message2)
```



Без Combiner



С Combiner

Aggregator

```
class Aggregator:  
    function aggregate(value) # агрегирование значений  
    function getAggregatedValue() # агрегированное значение  
    function setAggregatedValue(value) # установка значения  
    function reset() # сброс значения
```



Агрегатор

```
maxValue = -Inf
function aggregate(value):
    maxValue = max(maxValue, value)
function getAggregatedValue():
    return maxValue
function setAggregatedValue(value):
    maxValue = value
function reset()
    maxValue = -Inf
```

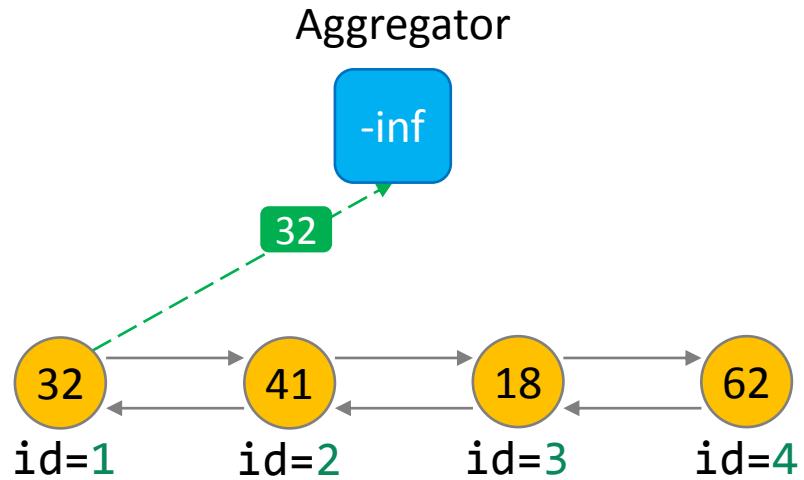


Метод выполнения (вершина)

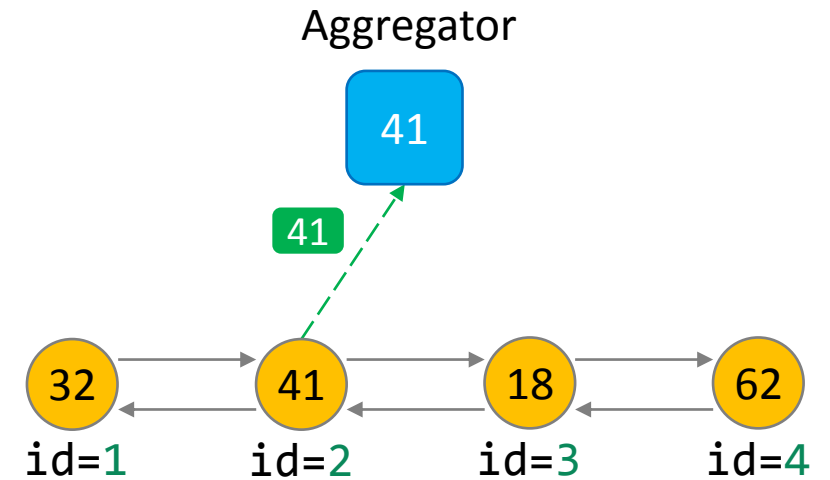
```
function compute(vertex, messages):
    maxValueAggregator.aggregate(vertex.getValue())
    vertex.voteToHalt()
```

Пример Aggregator

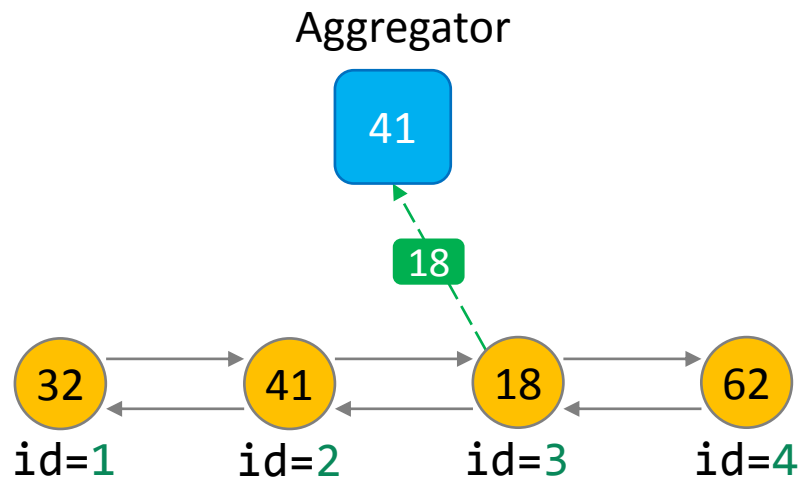
1.



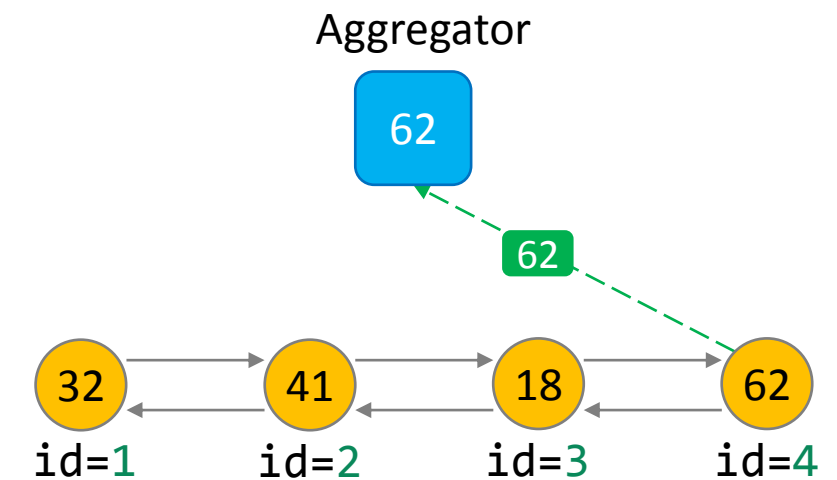
2.



3.

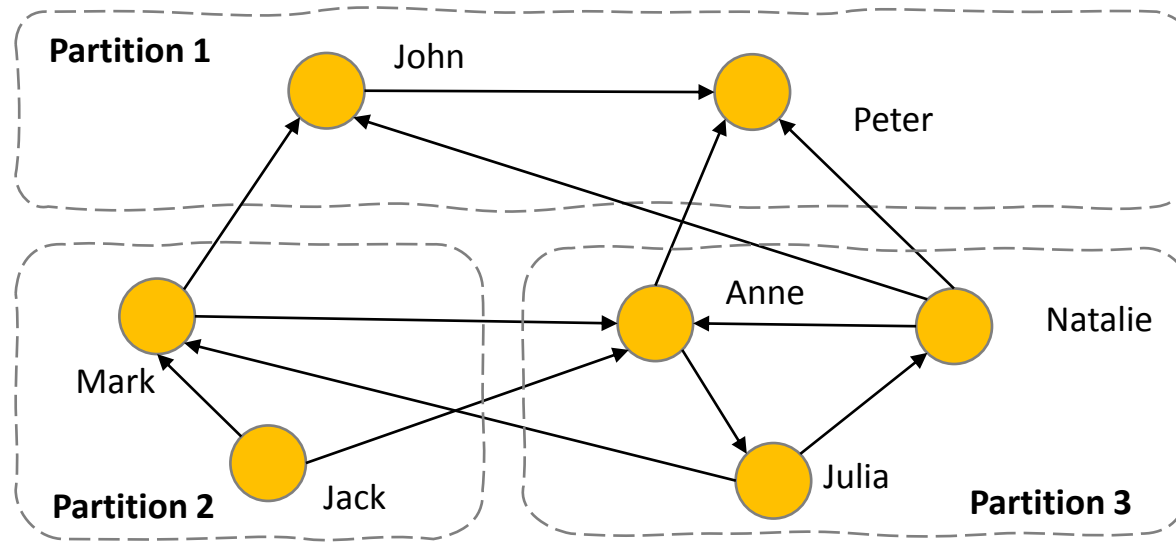


4.

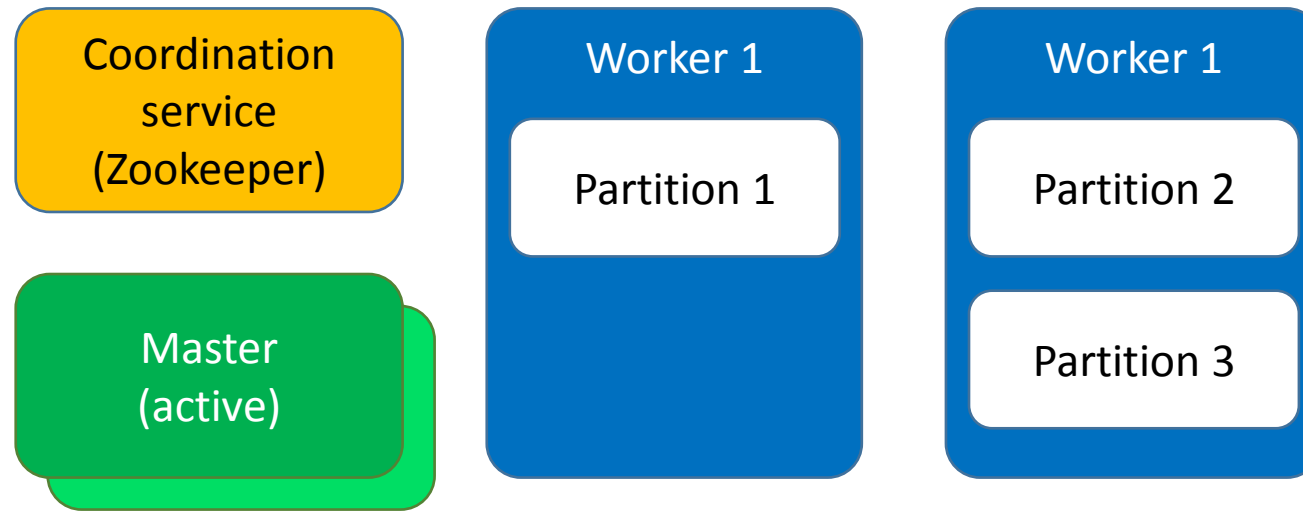


Архитектура Giraph

Разбиение графа на части



Компоненты Giraph



Master
(active)

- Управление superstep'ами
- Назначение частей (partitions) графа worker'ам
- Запуск compute для агрегаторов
- Мониторинг работоспособности и статистики worker'ов

Worker и Координатор

Worker 1

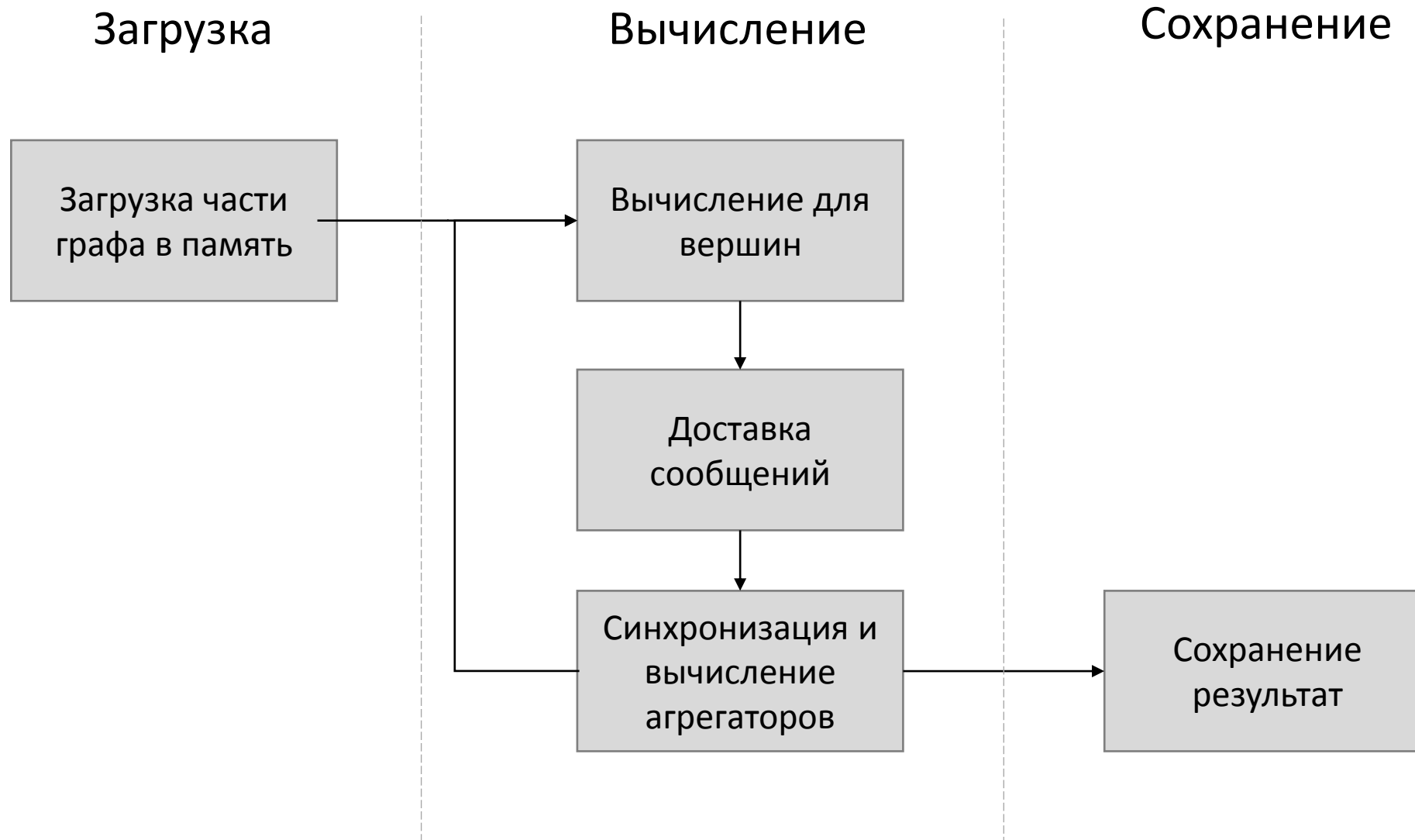
- Управление состояниями частей графа
- Выполнение compute для всех вершин частей на worker'е
- Сохранение состояния (checkpoint)

Coordination
service
(Zookeeper)

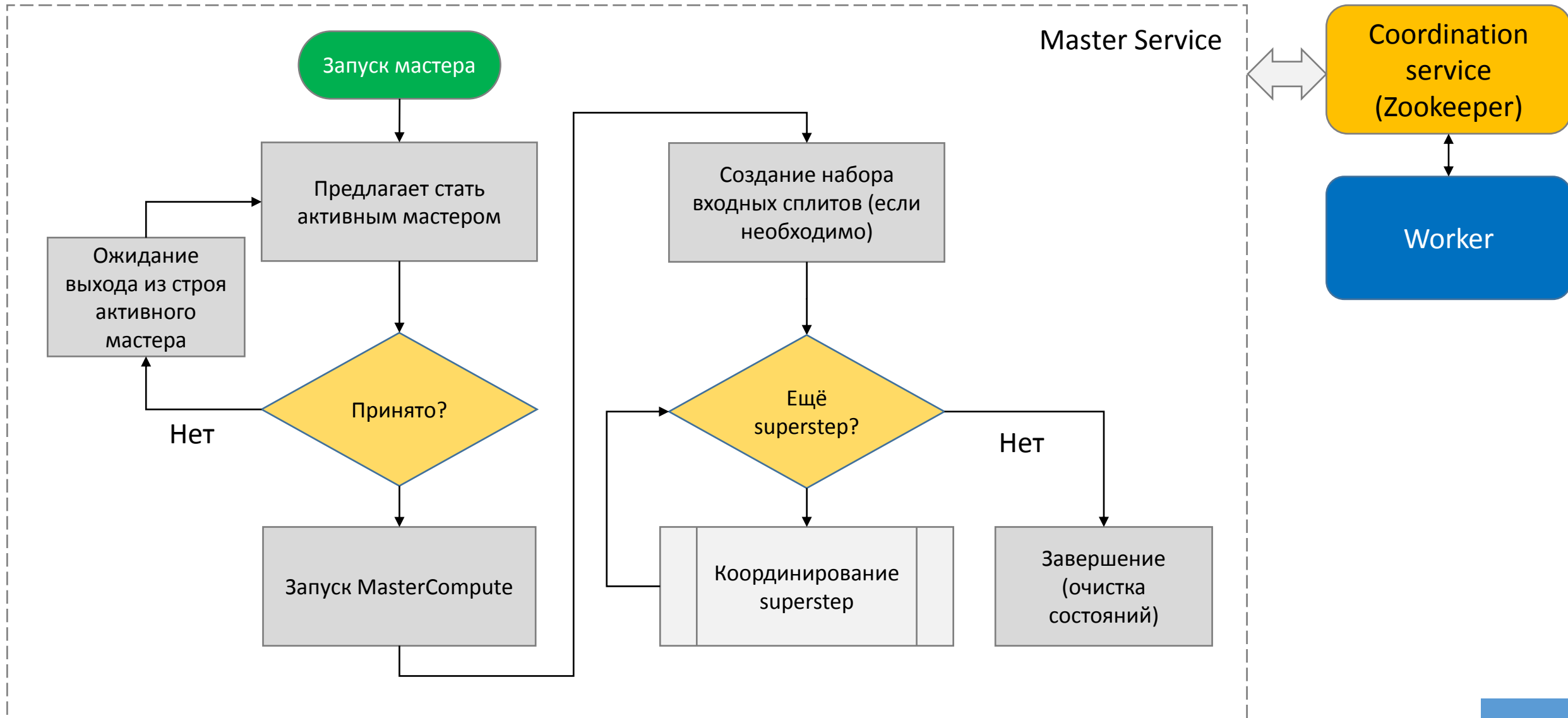
- Конфигурирование
- Синхронизация

Запуск вычислений

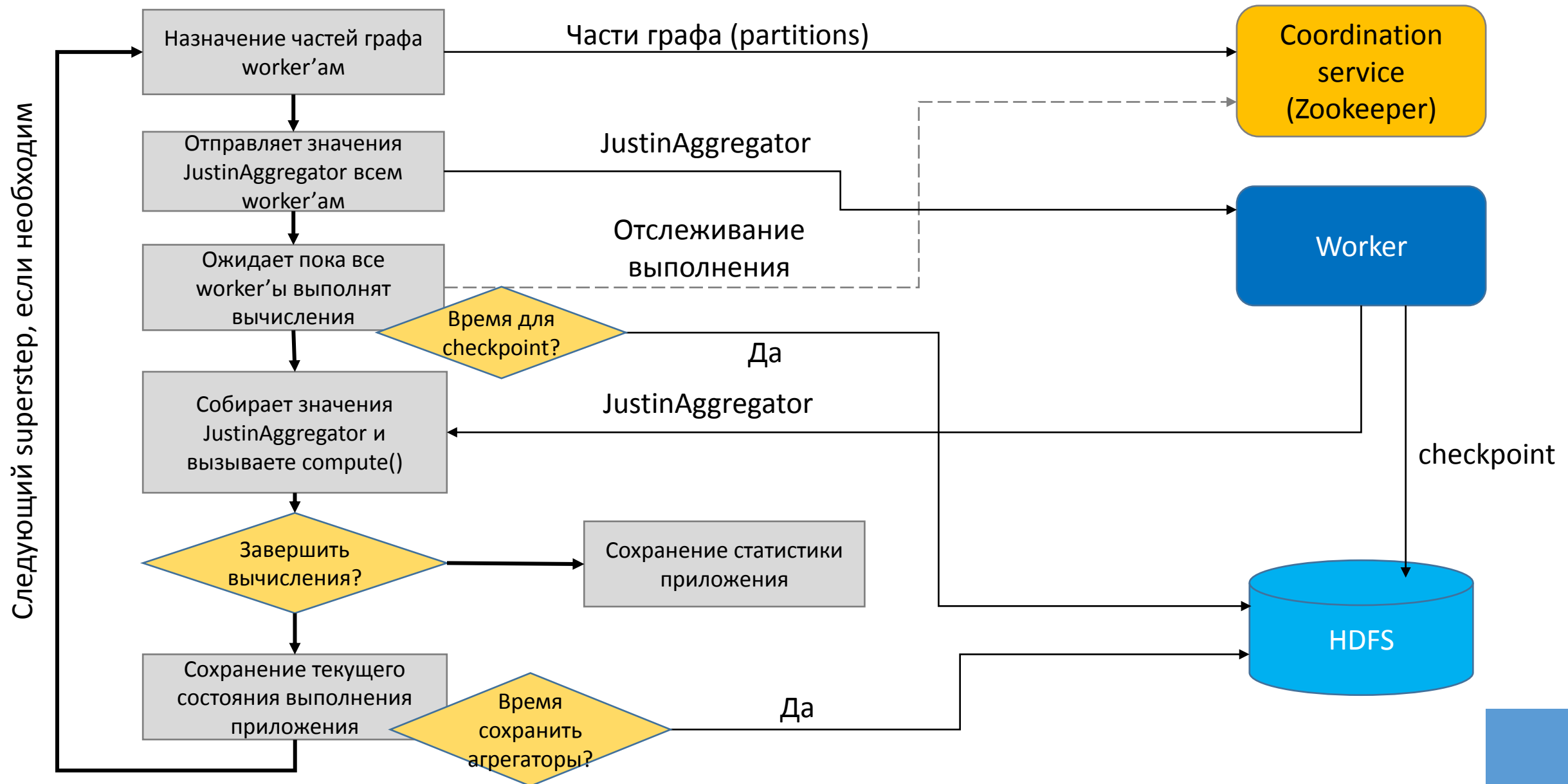
Основные стадии Giraph



Запуск Master Service



Master Service. Superstep



- Установлен внешний флаг остановки вычислений
- Все вершины перешли в состояние остановки (halt) и нет сообщений для вершин
- Превышение максимального количества superstep'ов



Стадия настройки

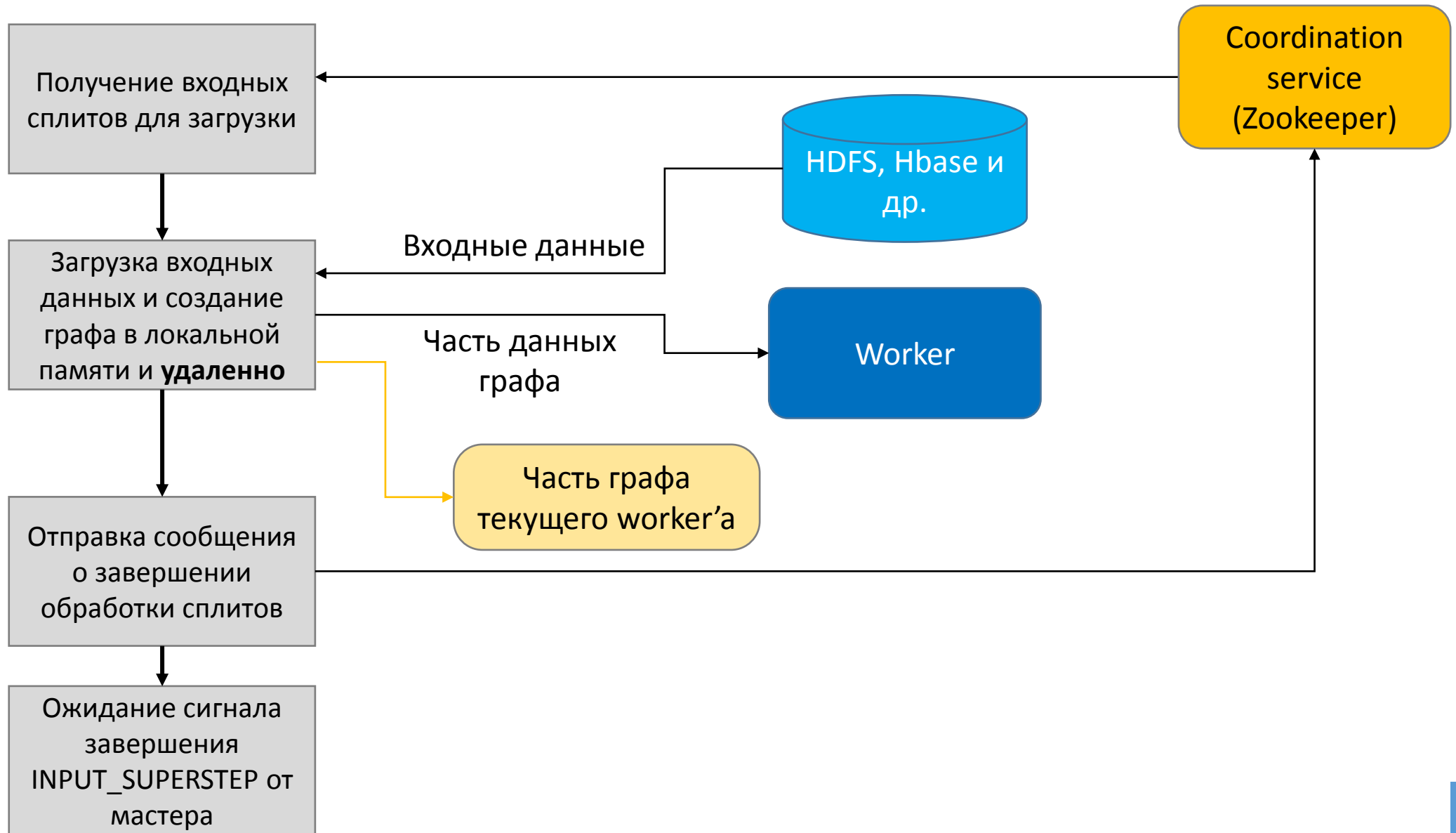
Загрузка исходных данных графа на все worker'ы



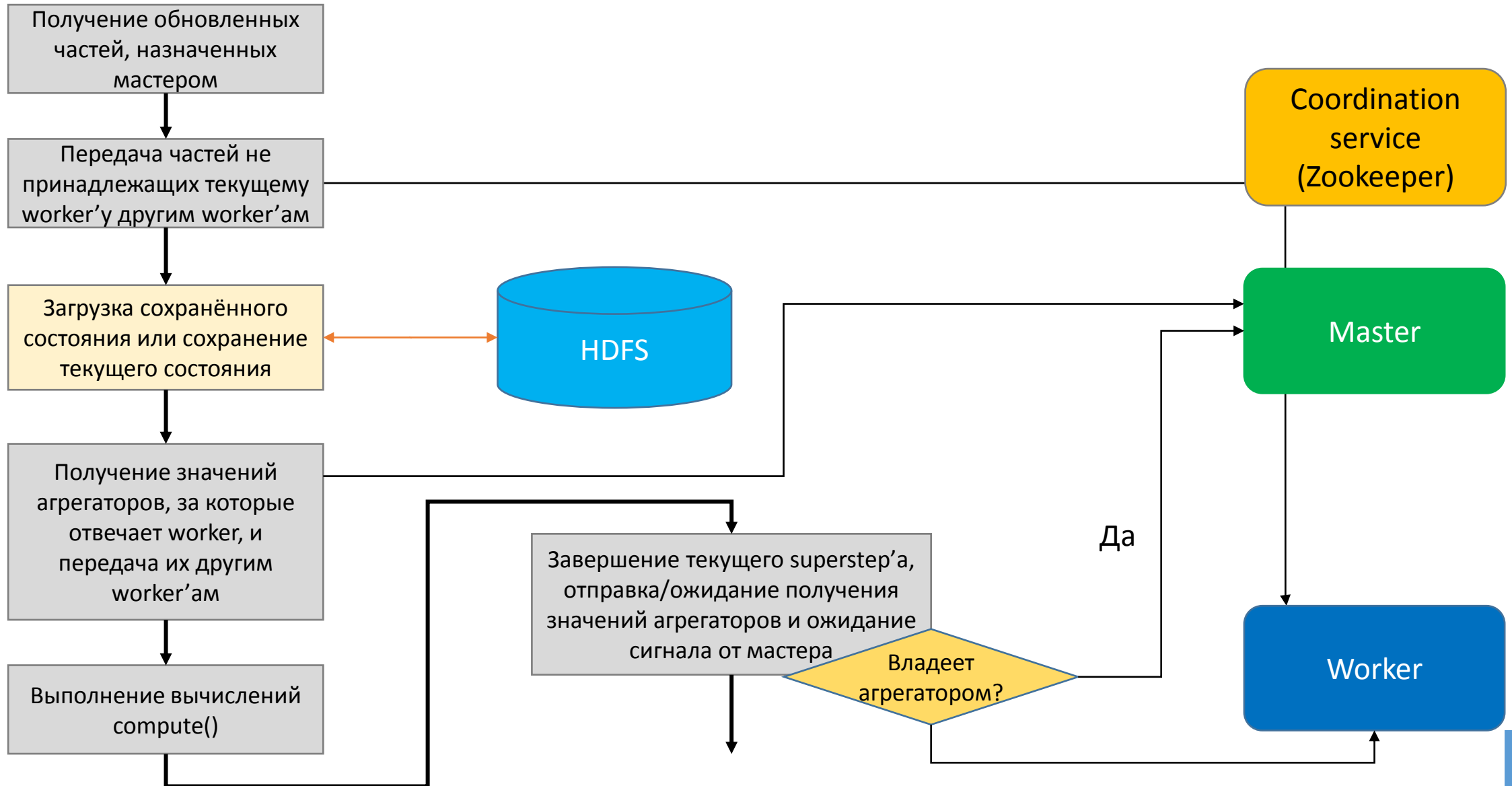
Стация основного цикла superstep'ов

Вычисления

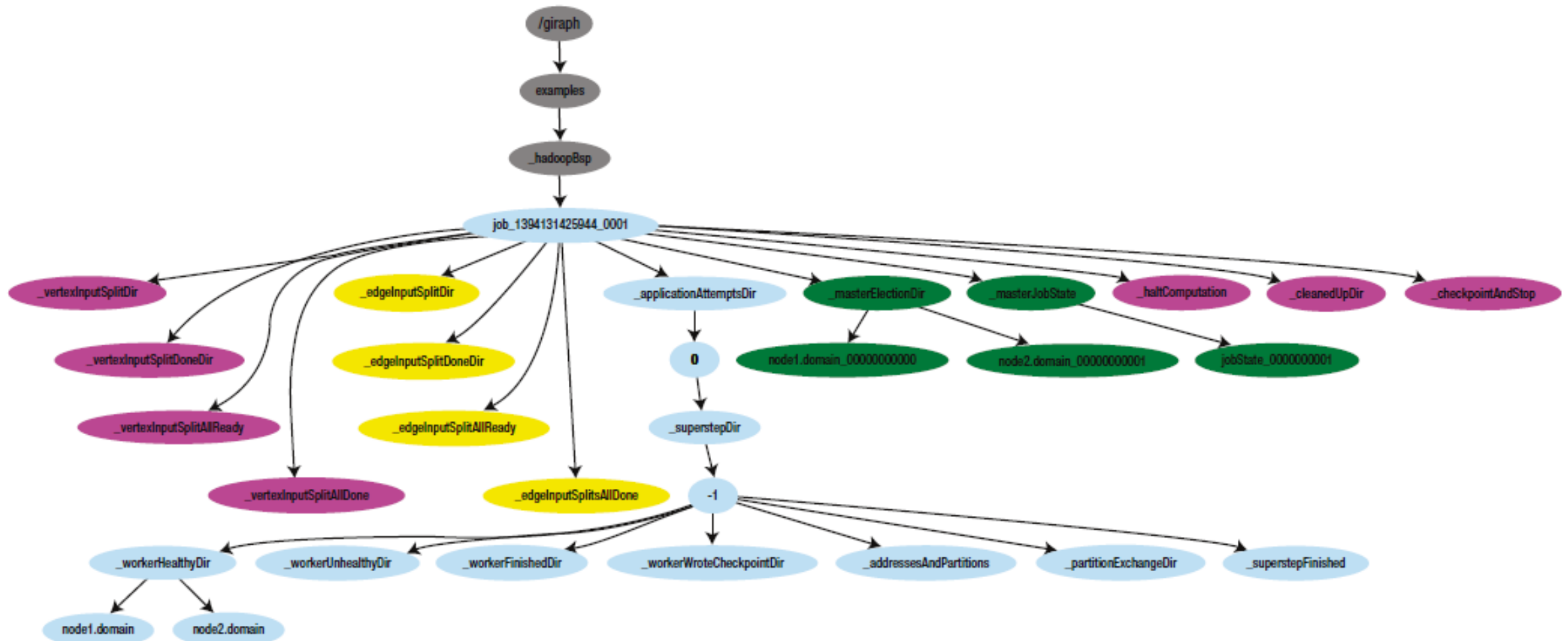
Worker Service. Загрузка данных. INPUT_SUPERSTEP



Worker Service. Основной вычислительный цикл



Координирующий Service/Zookeeper



Формат входных данных

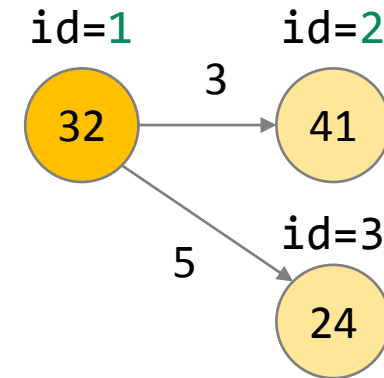
Формат входных данных определяет как читать данные из системы хранения и как затем преобразовать их в объекты Vertex и Edge

➤ VertexInputFormat

```
(1, 32, 41, 3, 24, 5)
(2, 41)
(3, 24)
```

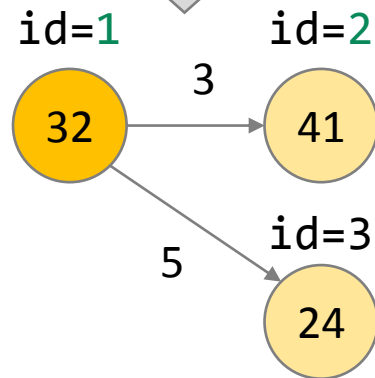
➤ EdgeInputFormat

```
(32, 41, 3)
(32, 24, 5)
```



VertexInputFormat

(1, 32, 41, 3, 24, 5)
(2, 41)
(3, 24)



Загружает данные из системы хранения

Разбивается входные данные на сплиты
(InputSplits)

Каждый сплит обрабатывается VertexReader

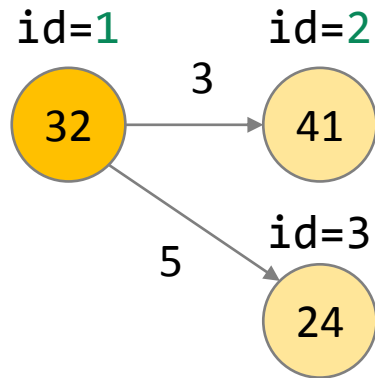
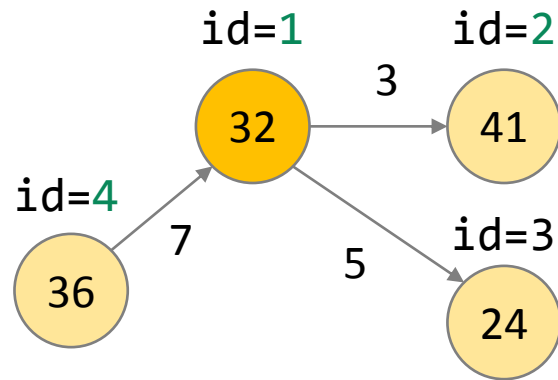
Каждая запись преобразуется в объект Vertex

Формат выходных данных определяет как преобразовать граф к отдельные записи и сохранить их в системе хранения

➤ VertexOutputFormat

➤ EdgeOutputFormat

VertexInputFormat



(1, 32, 41, 3, 24, 5)

Отвечает за сохранение данных в
системе хранения
Назначает вершине VertexWriter

Каждая вершина преобразуется
в запись

➤ Master

Если вышел из строя активный мастер, то

- Исчезает эфемерный znode в zookeeper'е, соответствующий мастеру
- Происходит выбор нового активного мастера
- Новый активный мастер получает состояния из zookeeper'а
- Значения агрегаторов будут получены от worker'ов при завершении текущего superstep'а

➤ Worker

- Если сбой произошел на стадии загрузки сплитов, то мастер отменит запуск приложения.
- Если сбой произошел на стадии основного цикла и checkpoint включен, то
 - ✓ исходный граф гарантированно сохранен в HDFS
 - Эфемерный znode worker'a исчезает и об этом уведомляется мастер
 - Мастер помечает текущий superstep как неудачный (failed)
 - Мастер запускает основной цикл с последнего сохраненного в HDFS superstep'a и поручает рабочим worker'am загрузить состояния, соответствующие последнему checkpoint'у

Координатор (Zookeeper)

Zookeeper – это распределенный координирующий сервис со встроенным механизмом отказоустойчивости. Как правило, ансамбль zookeeper'а состоит из 3 серверов и будет в рабочем состоянии при выходе из строя одного сервера (т.к. в этом случае будет обеспечен кворум из 2 узлов)

Practical Graph Analytics with Apache Giraph by Claudio Martella, Roman Shaposhnik, Dionysios Logothetis (book)

[Apache Giraph](#) (official website)