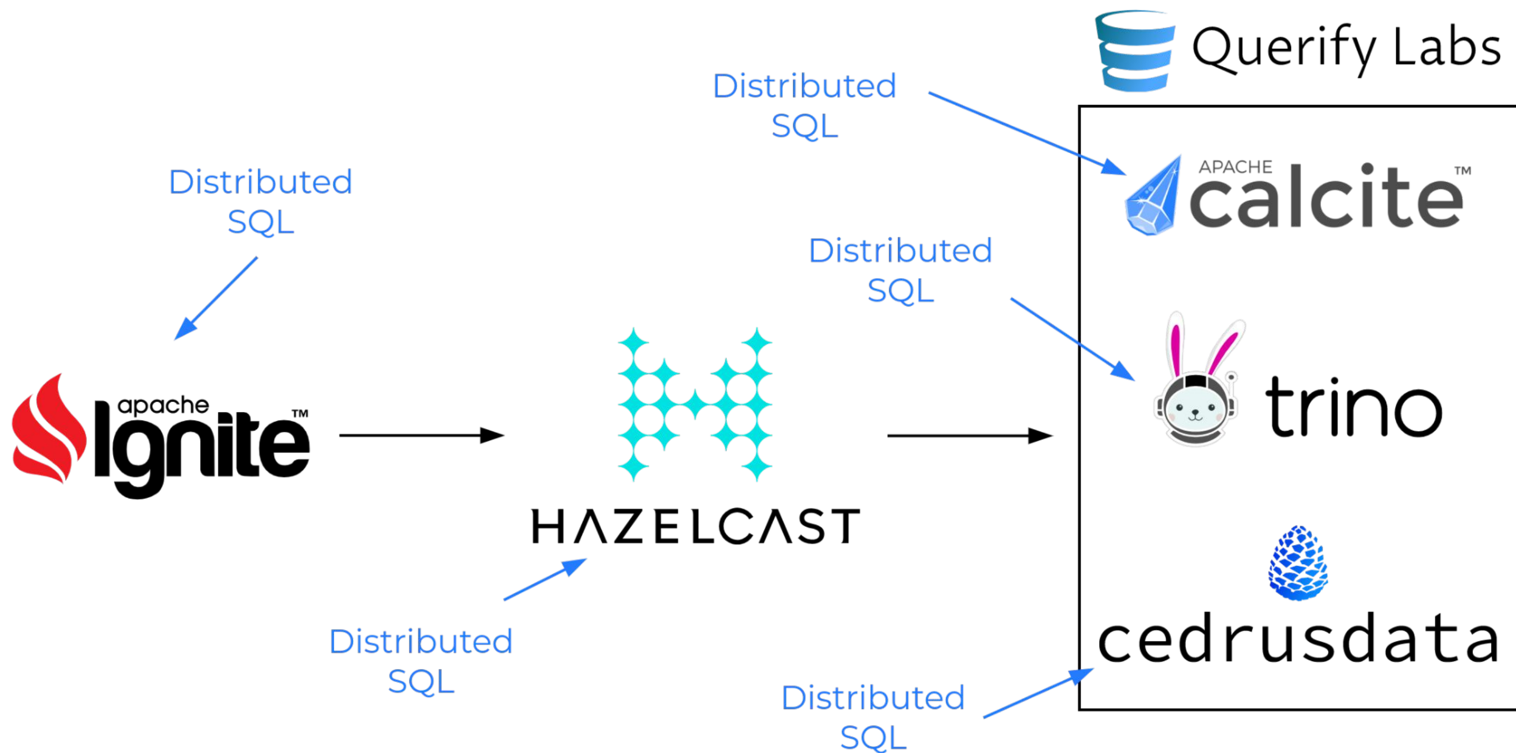
Querify Labs

Быстрая обработка данных в Data Lake с помощью Trino

Владимир Озеров
Querify Labs

Спикер



Composable Databases

Отделение форматов!

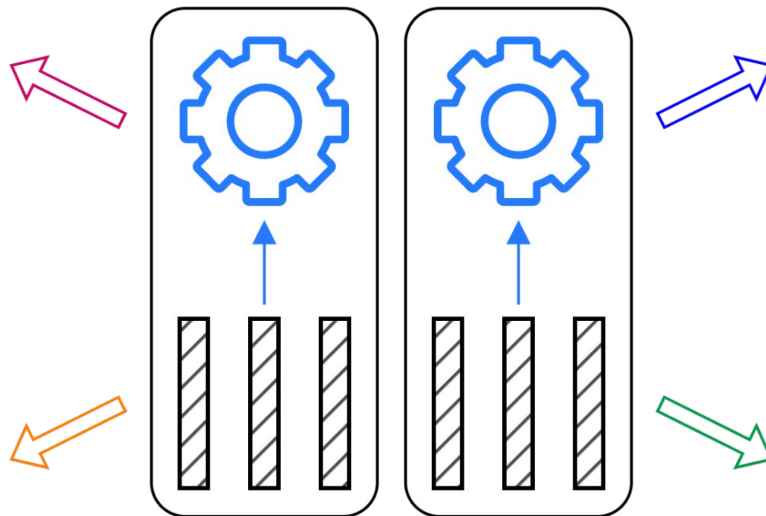
Apache Parquet

Apache Arrow

Отделение compute!

Trino / Presto

Velox



Отделение транзакций!

Apache Iceberg

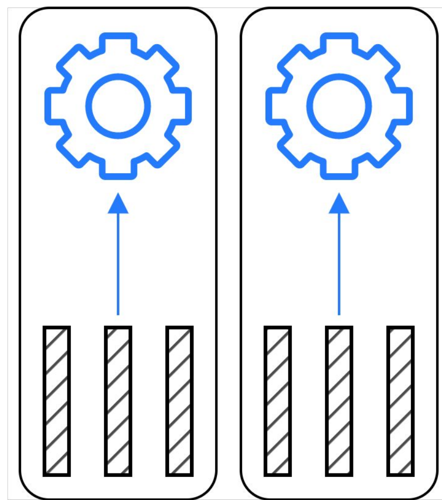
Apache Hudi

Отделение оптимизатора!

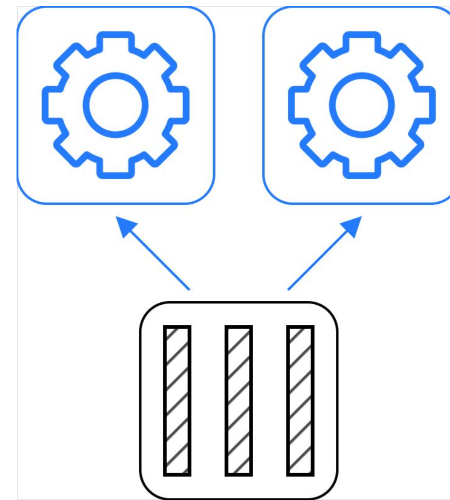
Apache Calcite

Substrait

Разделение compute и storage



Shared nothing
Данные совмещены с вычислениями



Shared storage
Данные отделены от вычислений

Распределенные SQL-движки

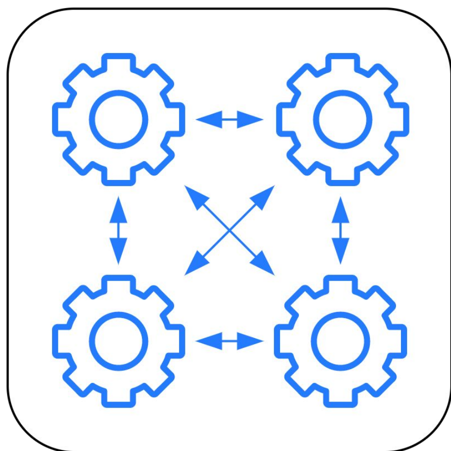
[Dremel: Interactive Analysis of Web-Scale Datasets]

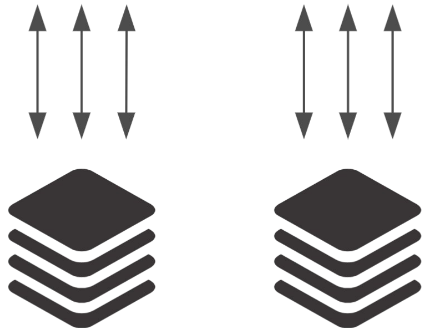
Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at **Google**.

[Presto: SQL on Everything]

Presto is an open source distributed query engine that supports much of the SQL analytics workload at **Facebook**. Presto is designed to be adaptive, flexible, and extensible. It supports a wide variety of use cases with diverse characteristics. These range from user-facing reporting applications with subsecond latency requirements to multi-hour ETL jobs that aggregate or join terabytes of data.

Распределенные SQL-движки

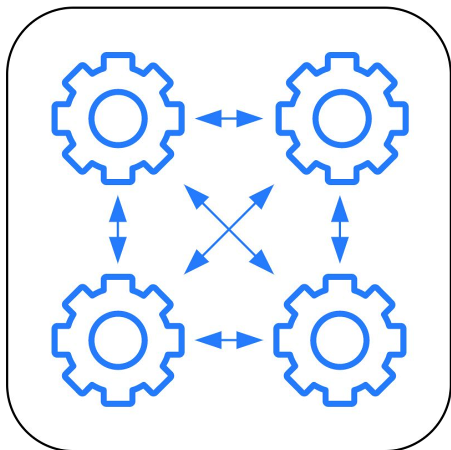


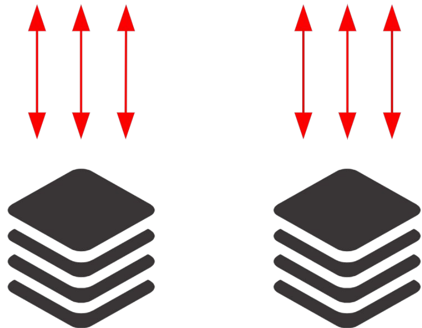


Основные характеристики:

- Отделение compute от storage
- Работает с открытыми форматами данных
- Распределенный
- Массивно-параллельный
- Вычисления в памяти (преимущественно)
- Колоночный
- Компилируемый

Распределенные SQL-движки





Основные характеристики:

- Отделение compute от storage
- Работает с открытыми форматами данных
- Распределенный
- Массивно-параллельный
- Вычисления в памяти (преимущественно)
- Колоночный
- Компилируемый

А ЭТО ВООБЩЕ МОЖЕТ БЫСТРО РАБОТАТЬ?

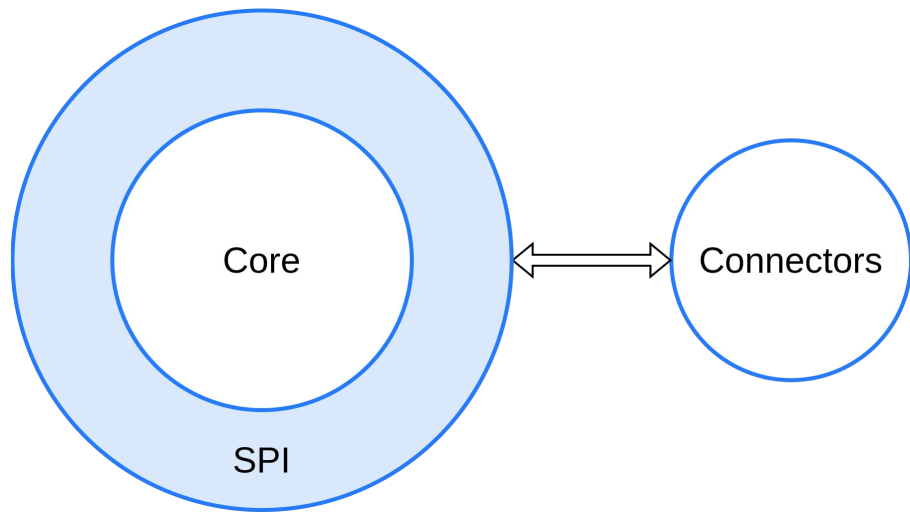
О чем доклад?

Какие подходы используют Trino и схожие SQL-движки для быстрого чтения данных из озера

Ответ

1. Делать меньше бесполезной работы
2. Делать больше полезной работы

Архитектура Trino



Core:

- Оптимизатор запросов
- Распределенный движок
- Управление ресурсами

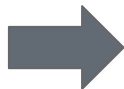
SPI:

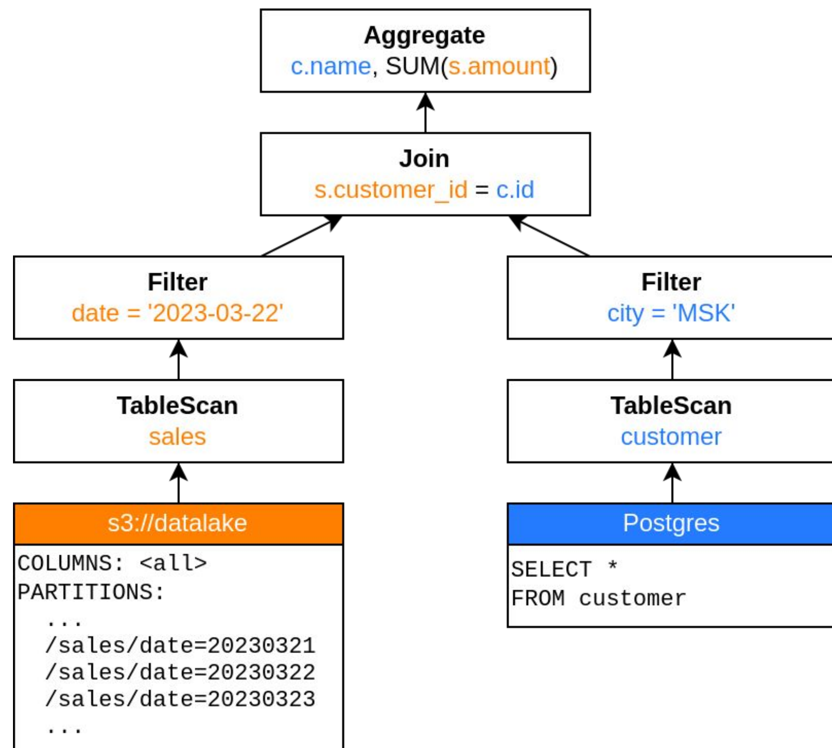
- Интеграция с источниками данных
- … и многое другое

Работа с озерами данных возможна через коннекторы Hive, Iceberg, Hudi, Delta Lake

Логический план

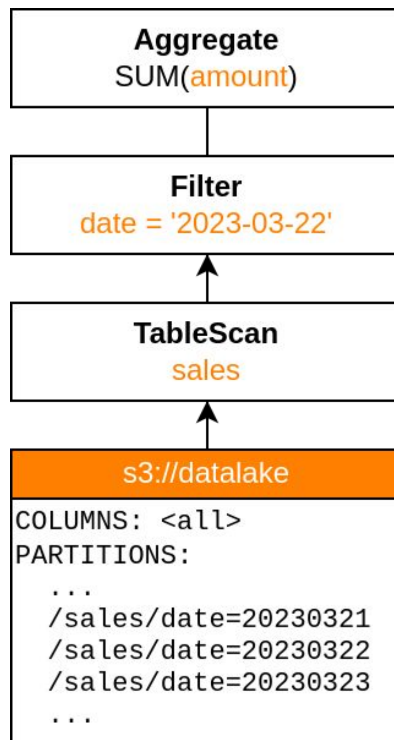
```
SELECT c.name, SUM(s.amount)
FROM datalake.sales s
  JOIN postgres.customer c
WHERE s.customer_id = c.id
  AND s.date = '2023-03-22'
  AND c.city = 'MSK'
GROUP BY c.name
```





Projection (Column) Pushdown

```
SELECT SUM(s.amount)
FROM datalake.sales
WHERE s.date = '2023-03-22'
```



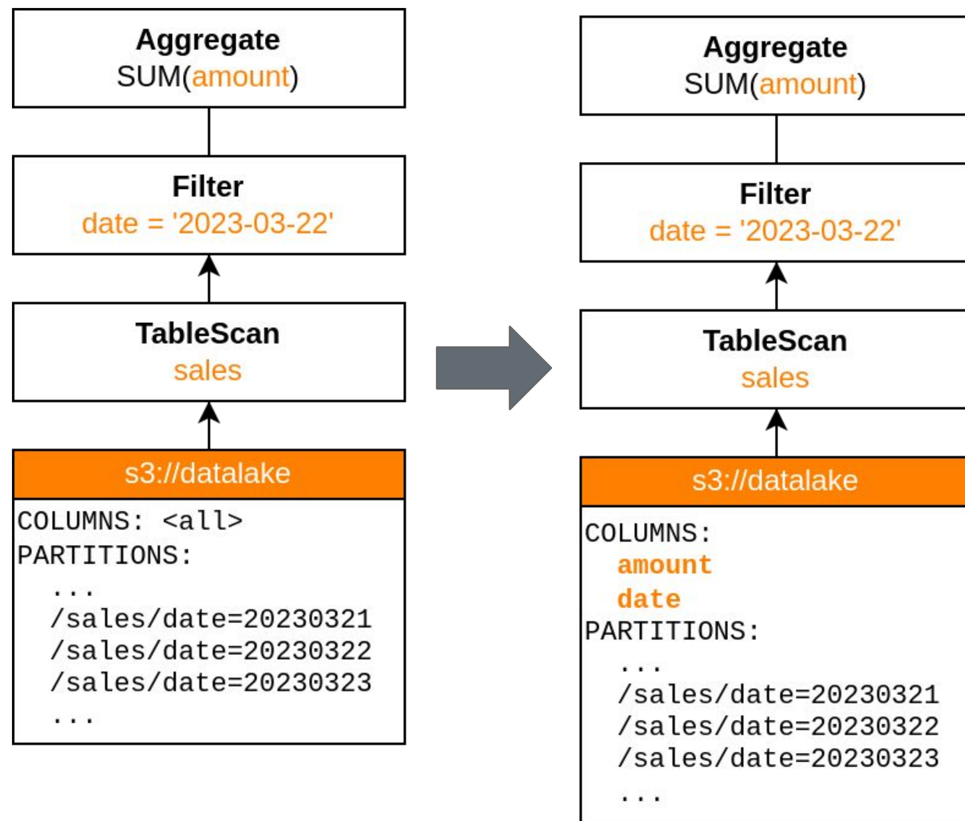
Projection (Column) Pushdown

```
SELECT SUM(s.amount)
FROM datalake.sales
WHERE s.date = '2023-03-22'
```

Принцип:

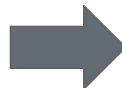
- Составить список атрибутов, необходимых для выполнения запроса
- Уведомить коннектор

Код: HiveMetadata.applyProjection



Filter Pushdown

```
SELECT SUM(amount)
FROM datalake.sales
WHERE date = '2023-03-22'
  AND city = 'MSK'
  AND amount * commission >
1000
```

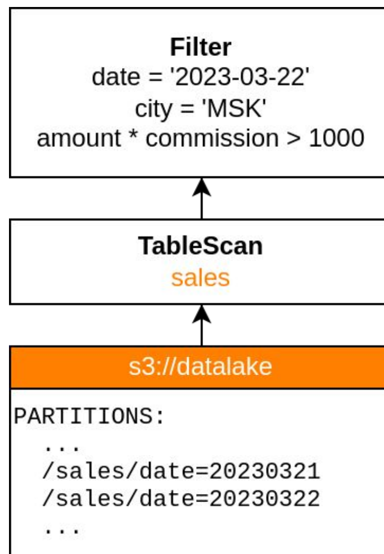


```
CONSTRAINTS:
  date = '2023-03-22'
  city = 'MSK'
```

- Вывести ограничения, накладываемые на отдельные колонки

Filter Pushdown

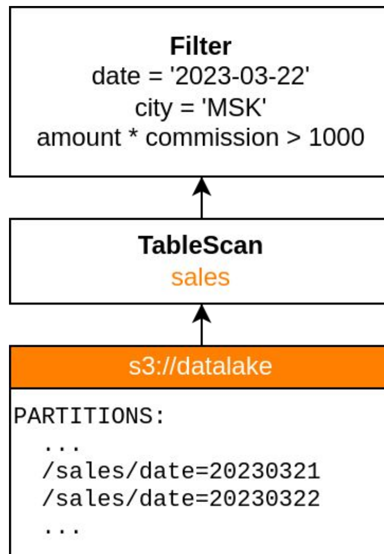
```
CONSTRAINTS:
  date = '2023-03-22'
  city = 'MSK'
```

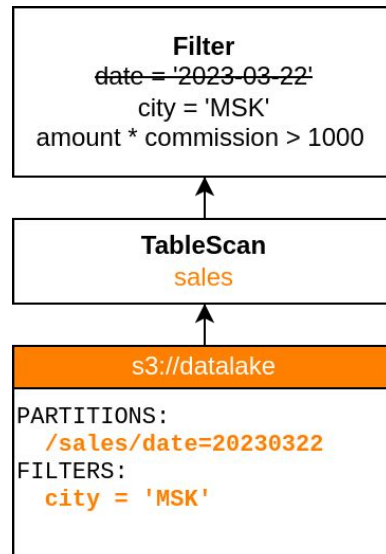


- Вывести ограничения, накладываемые на отдельные колонки
- Уведомить коннектор

Filter Pushdown

```
CONSTRAINTS:
   date = '2023-03-22'
   city = 'MSK'
```



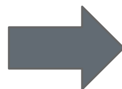


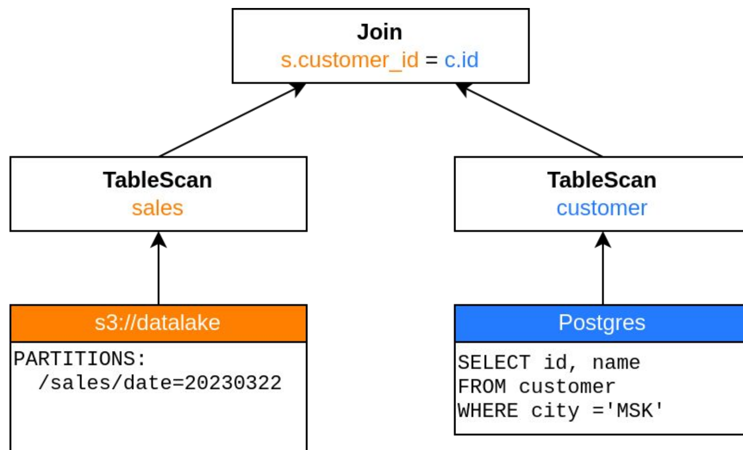
- Вывести ограничения, накладываемые на отдельные колонки
- Уведомить коннектор
- Коннектор применяет некоторые ограничения, и возвращает ограничения, которые движок должен применить самостоятельно

Код: ConnectorMetadata.applyFilter, HiveMetadata.applyFilter

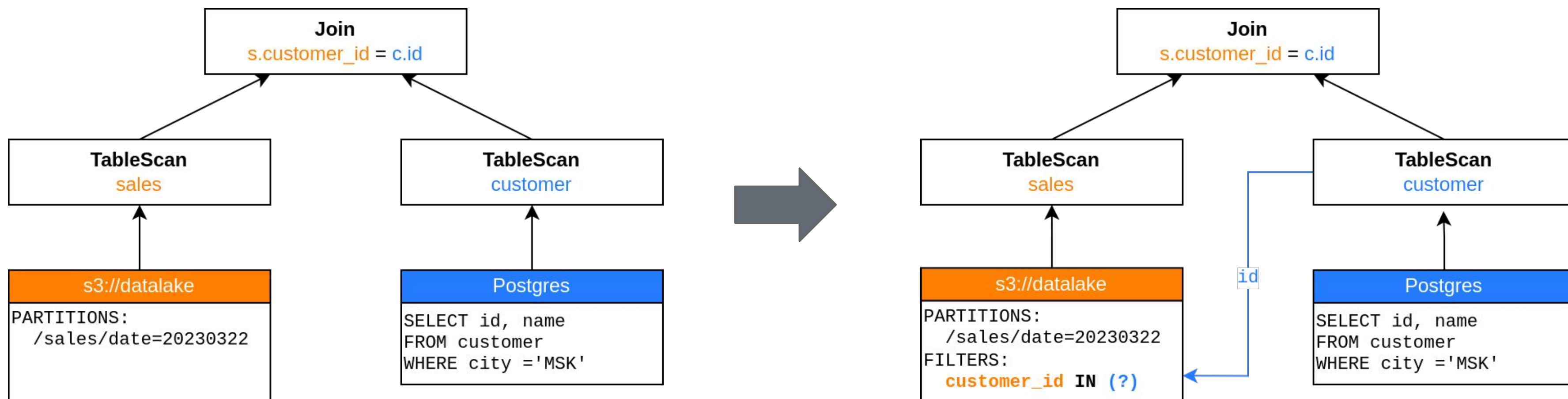
Dynamic Filters

```
SELECT ...
  JOIN postgres.customer c
WHERE s.customer_id = c.id
  AND s.date = '2023-03-22'
  AND c.city = 'MSK'
```





Dynamic Filters

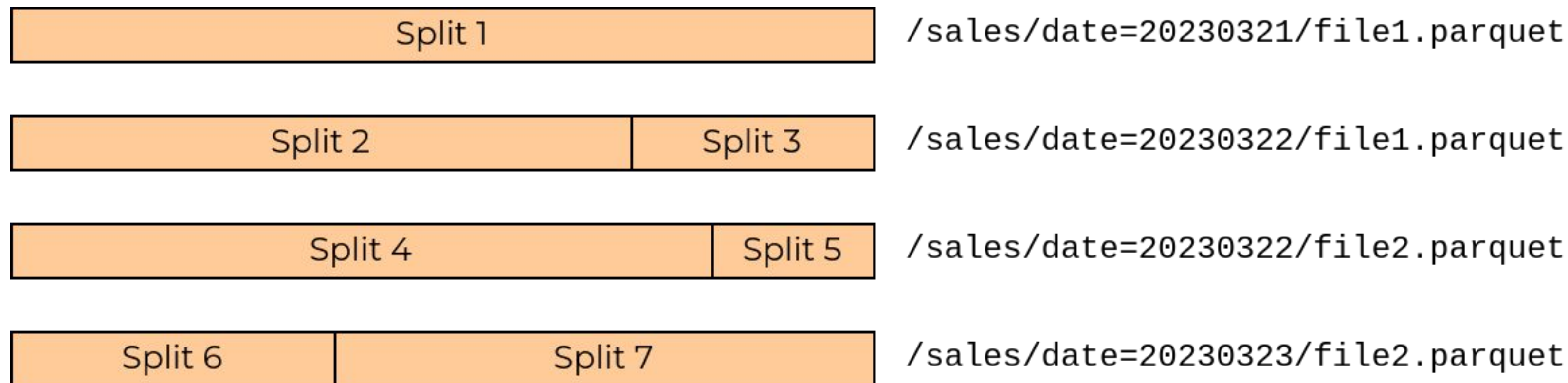


- Выполнить правую часть JOIN
- Получить список значений ключа equi-join справа (customer.id)
- Передать значения в качестве дополнительного фильтра ключа equi-join слева (sales.customer_id)

Документация: https://trino.io/docs/current/admin/dynamic-filtering.html

Подробнее: https://habr.com/ru/companies/cedrusdata/articles/740274/

Сплиты



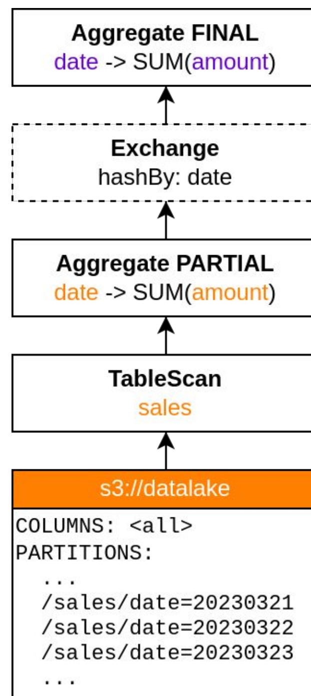
Для параллельного чтения данных Trino разбивает данные на сплиты (Split). Каждый сплит описывают задачу на чтение подмножества данных таблицы. Обработка сплитов происходит параллельно на worker-узлах.

При работе с озерами данных сплитом является задача на чтение части файла.

Код: HiveSplit

Распределенные операции

```
SELECT date,
SUM(amount)
FROM datalake.sales
GROUP BY date
```



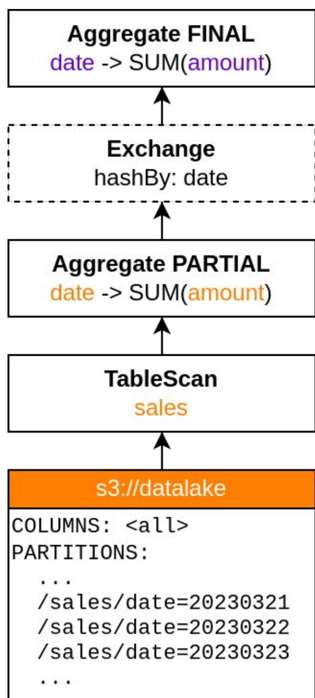
Некоторые операторы требуют определенное распределение данных в кластере

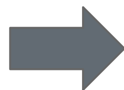
- Например, для выполнения GROUP BY необходимо убедиться, что все значения группы находятся на одном узле

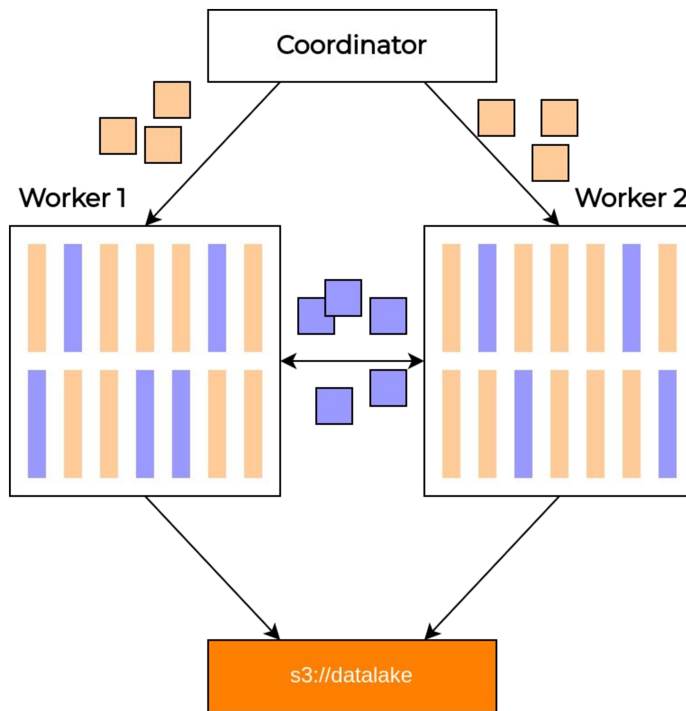
Оператор **Exchange** обеспечивает необходимое распределение данных для вышестоящего оператора

Код: AddExchanges

Планирование чтений





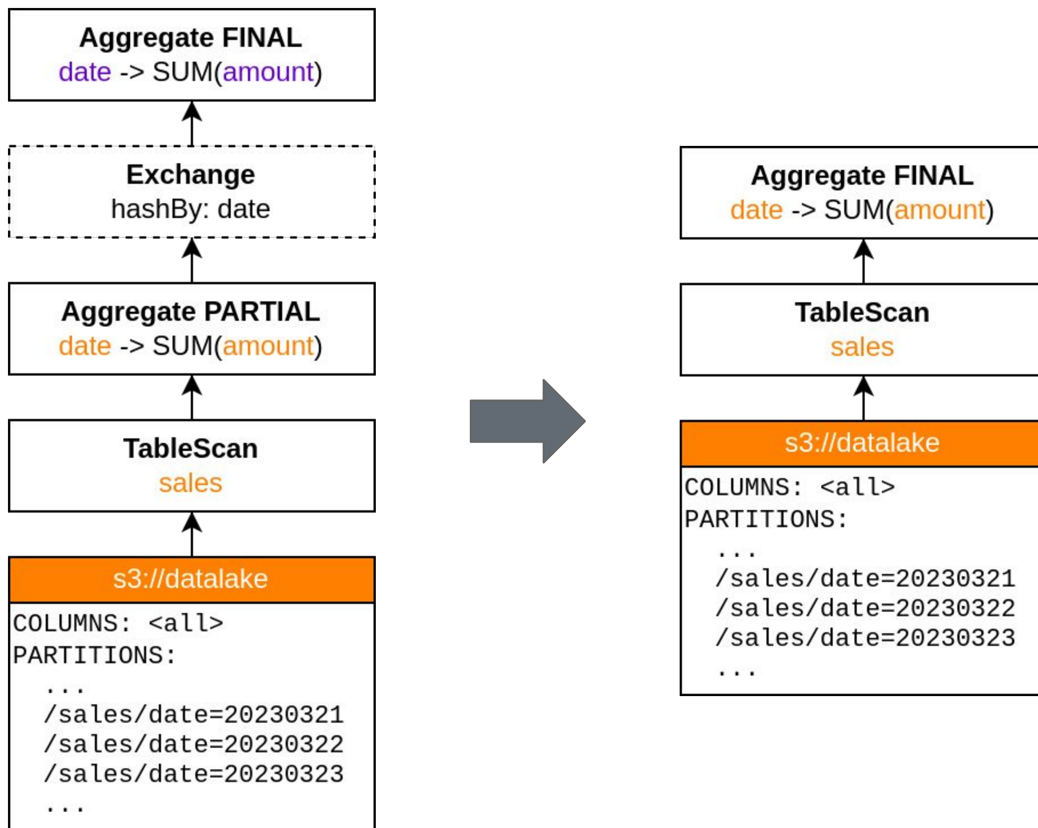


Координатор Trino распределяет сплиты по узлам в зависимости от нагрузки, стремясь обеспечить максимальный уровень параллелизма

Код:

- HiveSplitSource
- BackgroundHiveSplitLoader

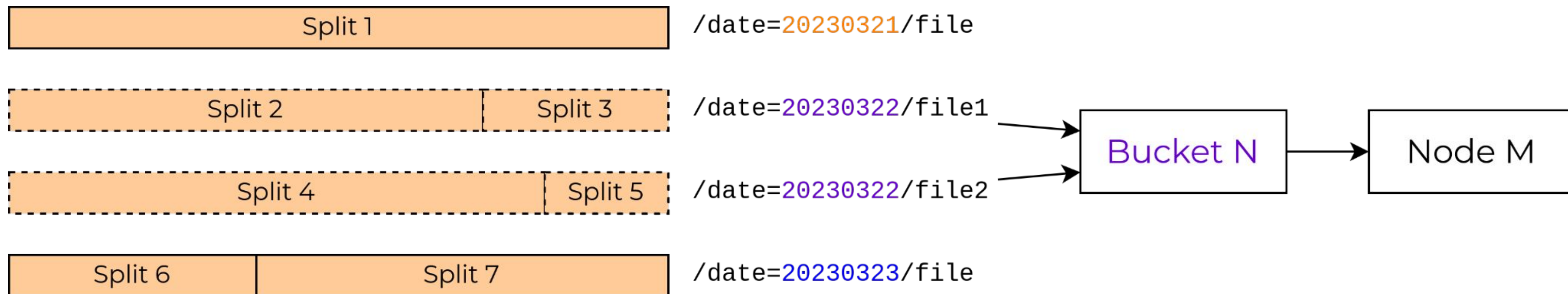
Группировка сплитов



В некоторых случаях группировка сплитов по определенному признаку позволяет Trino уменьшить количество Exchange в плане

Например, если таблица **sales** партицирована по **date**, то можно распределить сплиты с одинаковыми значениями **date** по одним и тем же узлам. Таким образом можно выполнить агрегацию без дополнительного перераспределения данных

Группировка сплитов

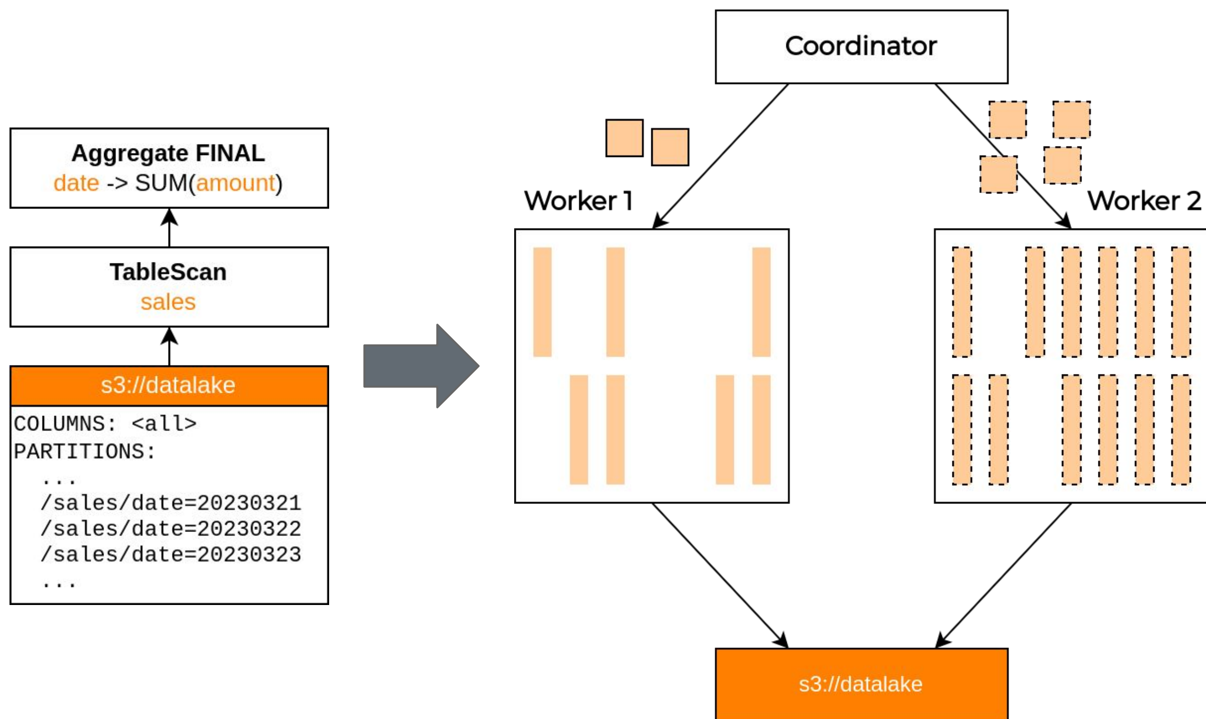


1. Коннектор определяет:
   - Количество бакетов (bucket), на которые будут разбиты данные
   - Функцию сопоставления сплита с бакетом
2. Ядро Trino гарантирует, что все сплиты, попавшие в один бакет будут выполнены строго на одном узле

При работе с озерами данных, сплиты можно группировать по bucketing и partitioning*.

* пока только в CedrusData

Чтение сгруппированных сплитов



Группировка сплитов упрощает план

- Обычно это ускоряет запросы
- Но может и замедлить, из-за того, что мы менее гибко балансируем сплиты по узлам

Конфигурация:

- `hive.bucket-execution`

Код:

- [HiveNodePartitioningProvider]()

Обработка Split

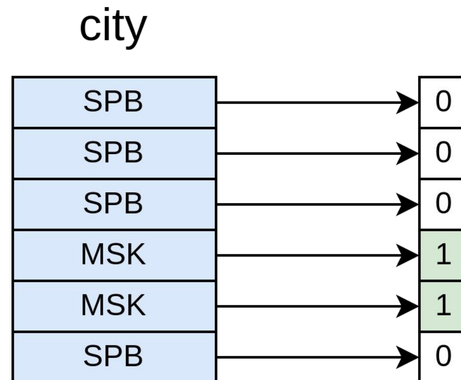
SPLIT

```
PATH: /sales/date=20230321, offset 100, length 512
COLUMNS:
  amount
  date
CONSTRAINTS:
  static:  city = 'MSK'
  dynamic: customer_id IN (?)
```

1. Попробовать применить динамические фильтры, чтобы отбросить partition
2. Получить метаданные
   - Parquet: footer, смещения row groups и колонок в них, и т.п.
3. Отбросить неподходящие диапазоны строк
   - Parquet: на основе статистик, индексов, фильтров Блума
4. Спланировать чтение требуемых диапазонов строк

Late materialization

```
SELECT SUM(amount)
FROM datalake.sales
WHERE city = 'MSK'
```

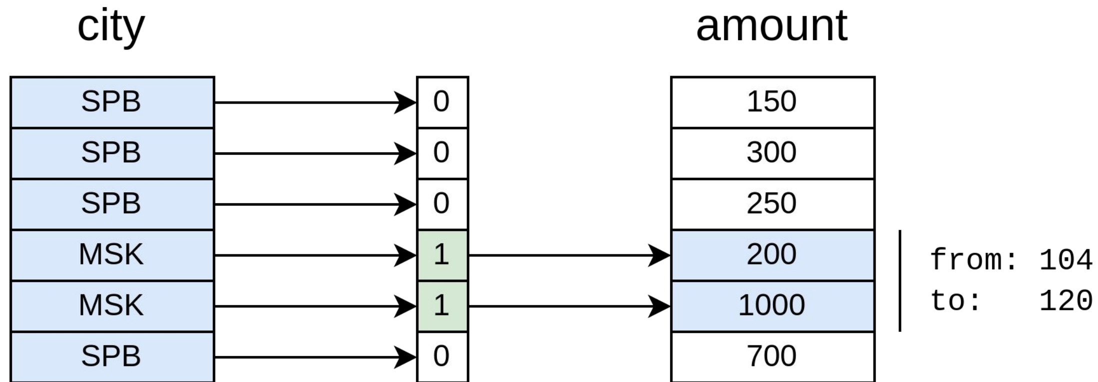


Порядок чтения колонок имеет значение:

1. Прочитать **city** и построить битовую маску

Late materialization

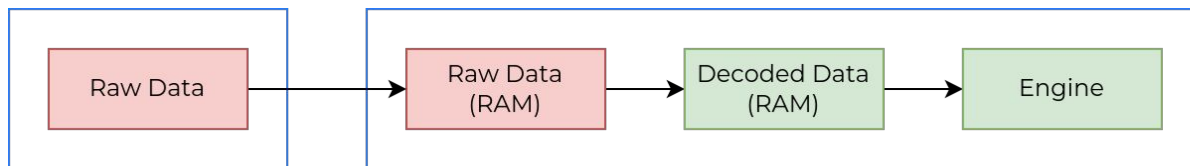
```
SELECT SUM(amount)
FROM datalake.sales
WHERE city = 'MSK'
```



Порядок чтения колонок имеет значение:

1. Прочитать **city** и построить битовую маску
2. Прочитать **amount** в соответствии с битовой маской

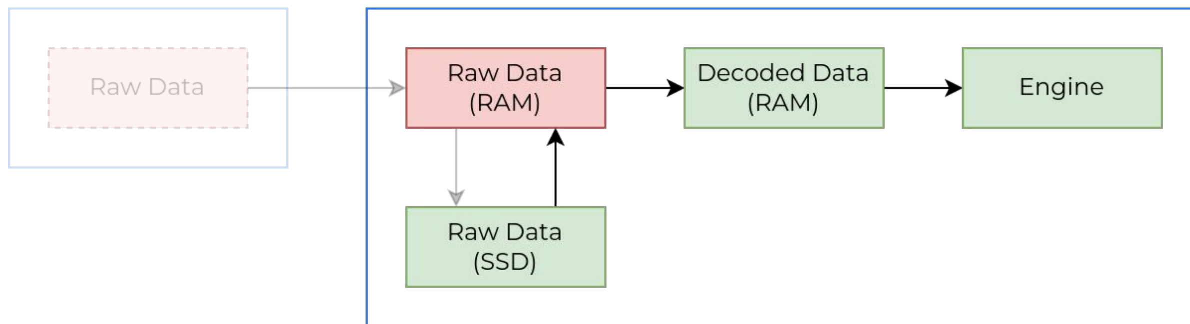
Remote чтения



Работа с данными в data lake приводит к накладным расходам:

1. Передача данных по сети
2. Декодинг данных из формата озера в формат движка

Дисковый кэш: подход 1

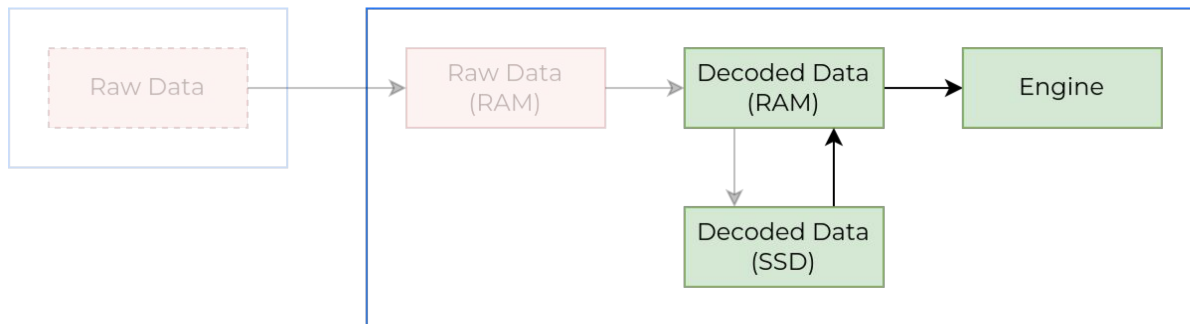


Локальный дисковый кэш, который перехватывает вызовы к распределенной файловой системе (HDFS, Ozone, и т.п.) или S3:

- Плюсы: прост в реализации; не привязан к движку
- Минусы: сохраняет накладные расходы на декодинг; может хранить больше данных, чем нужно, так как границы блоков файловой системы обычно не совпадают с внутренней структурой файлов

Пример: [Presto RaptorX Alluxio Data Cache]

Дисковый кэш: подход 2

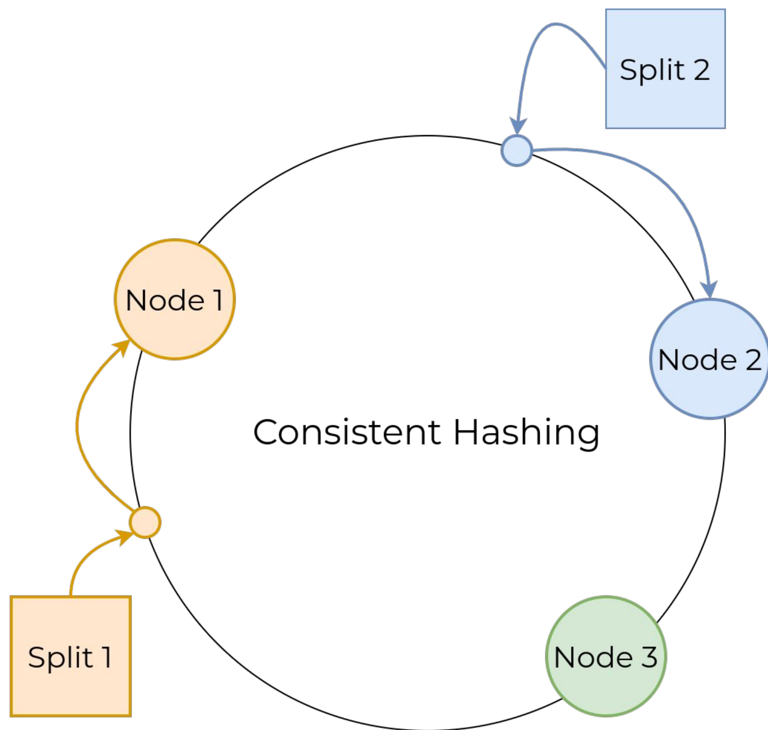


Локальный дисковый кэш, который сохраняет декодированные диапазоны данных:

- Плюсы: позволяет избежать декодинга; эффективное расходование места; позволяет добавлять дополнительные структуры данных (например, индексы)
- Минусы: высокая сложность реализации

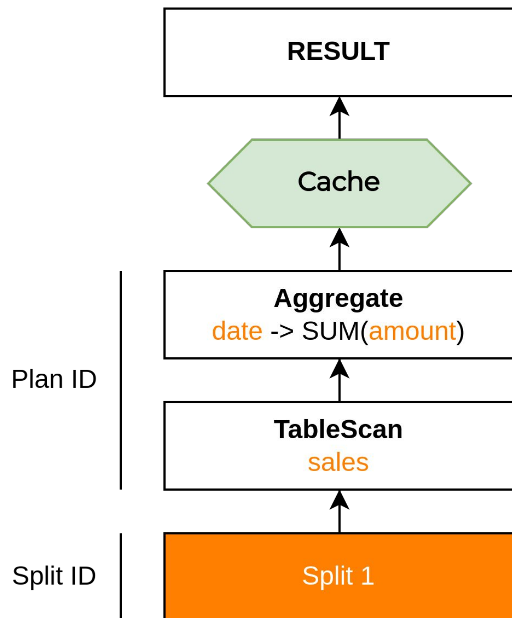
Примеры: дисковый кэш CedrusData (Hive, Iceberg), Starburst Warp Speed (ex-Varada)

Дисковый кэш: soft affinity



- Чтобы дисковый кэш был достаточно эффективным, необходимо стремиться отправлять сплиты на одни и те же узлы
- Для этого движки обычно реализуют “soft affinity”: возможность отдавать предпочтение одному и тому же узлу для одного и того же сплита
- Популярный алгоритм: Consistent hashing

Кэш результатов запросов



Наблюдение:

- Данные в озере изменяются редко
- Мы производим множество одинаковых вычислений над одними и теми же данными

Решение:

- Вычислять уникальный хэш пары {план, сплит}
- Кэшировать промежуточный результат для данного хэша

Пример: Presto RaptorX Fragment Result Cache

Материализованные представления

Наблюдение:

- Множество одинаковых вычислений над одними и теми же данными

Решение:

- Сохранять результаты работы запросов в виде материализованных представлений

Соображения:

- Гарантировать актуальность данных, или позволять пользователю читать устаревшие данные?
- Заставлять пользователей явно использовать материализованные представления, или переписывать запросы автоматически?
- Возможность частичного обновления?

Примеры:

- Trino: [Iceberg Materialized Views]
- Dremio: [Reflections]

Итого

- Больше полезной работы:
  - Параллельное чтение сплитов
- Меньше бесполезной работы:
  - Column pushdown
  - Filter pushdown и dynamic filtering
  - Группировка сплитов для уменьшения количества Exchange
  - Тщательный анализ метаданных файлов для прунинга ненужных сегментов
  - Чтение колонок в правильном порядке (late materialization)
  - Локальный кэш
  - Материализованные представления
- Не хватает в Trino:
  - Cost-based оптимизатор для дальнейшего уменьшения количества Exchange
  - Продвинутый локальный кэш
  - Автоматическое перенаправление запросов к подходящим материализованным представлениям

Вопросы

https://t.me/cedrusdatachat