一种高效的Spark SQL Native引擎实现方案

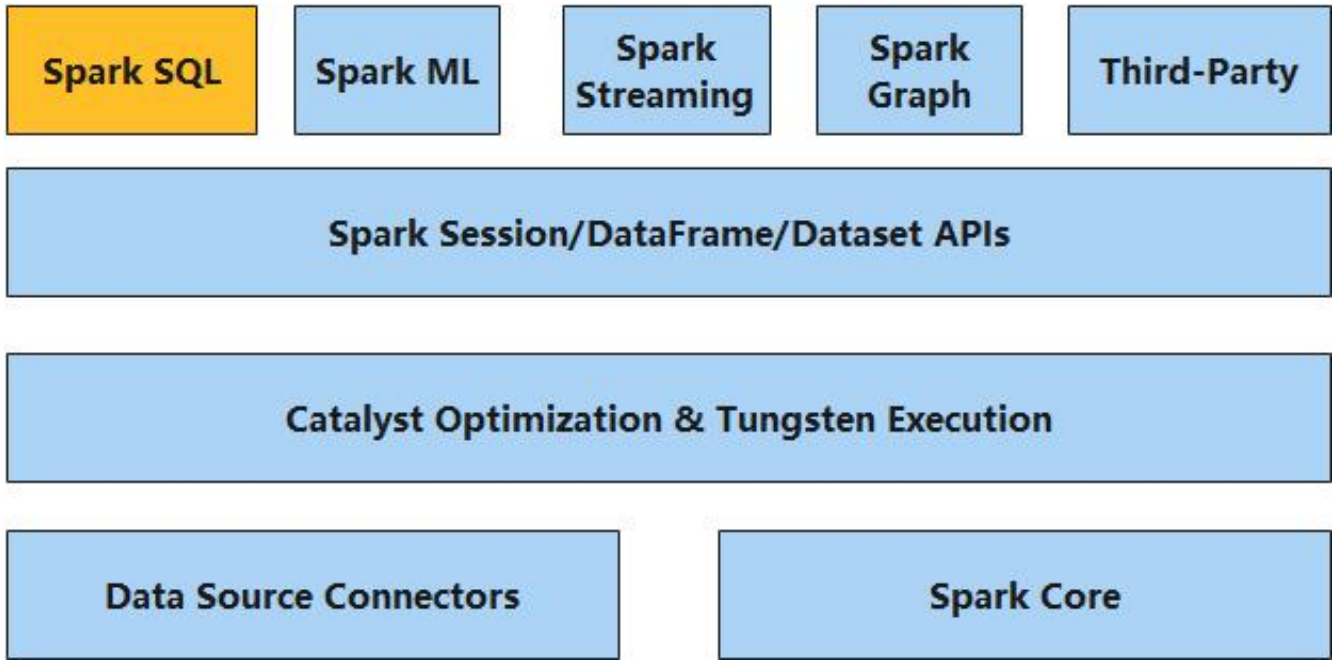
陈强

openEuler Bigdata SIG maintainer & Apache Bigtop committer

开放原子开源基金会 OPENATOM FOUNDATION | OpenEuler

目录

背景：Spark SQL



Spark SQL 是 Spark 中用来处理结构化数据的一个模块，它提供了一个编程抽象（DataFrame），并且可以作为分布式 SQL 的查询引擎。

Spark SQL 可以将数据的计算任务通过 SQL 的形式转换成 RDD再提交到集群执行计算，类似于 Hive 通过 SQL 的形式将数据的计算任务转换成 MapReduce，大大简化了编写 Spark 数据计算操作程序的复杂性，且执行效率比 MapReduce 这种计算模型高。

背景：Spark SQL Engine现有问题

随着硬件（存储、网络等）和软件技术的发展，CPU逐渐成为了系统的瓶颈。

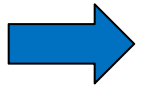
硬件	2010	2015	2020	性能提升
Storage	50MB/s(HDD)	500MB/s(SSD)	16GB/s(NVMe)	10X
Network	1Gbps	10Gbps	100Gbps	10X
CPU	~3GHz	~3GHz	~3GHz	

怎么解决CPU瓶颈问题?

- **代码动态生成**
  - 代表：Apache Spark
  - 优点：算子融合、编译执行、大循环等
- **向量化**
  - 代表：Databricks Photon
  - 优点：批量执行、SIMD等

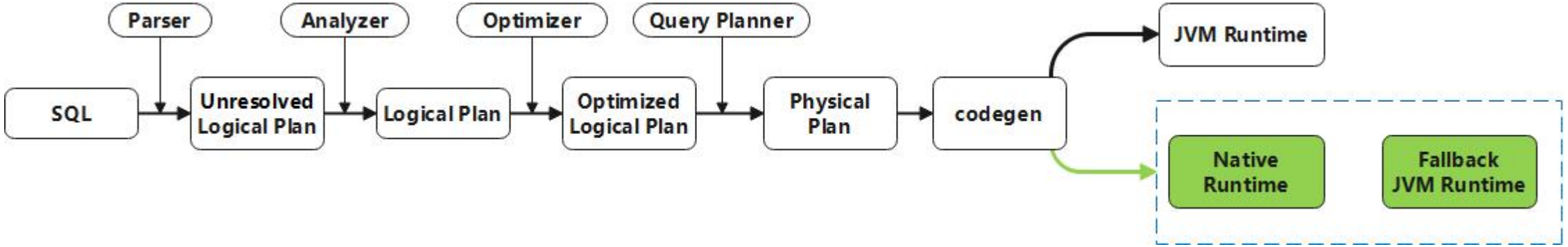
社区Spark SQL Engine的问题

- Java语义限制，难以SIMD、手动预取和内存控制
- GC开销较大



通过Native Engine，解决Spark SQL 性能问题

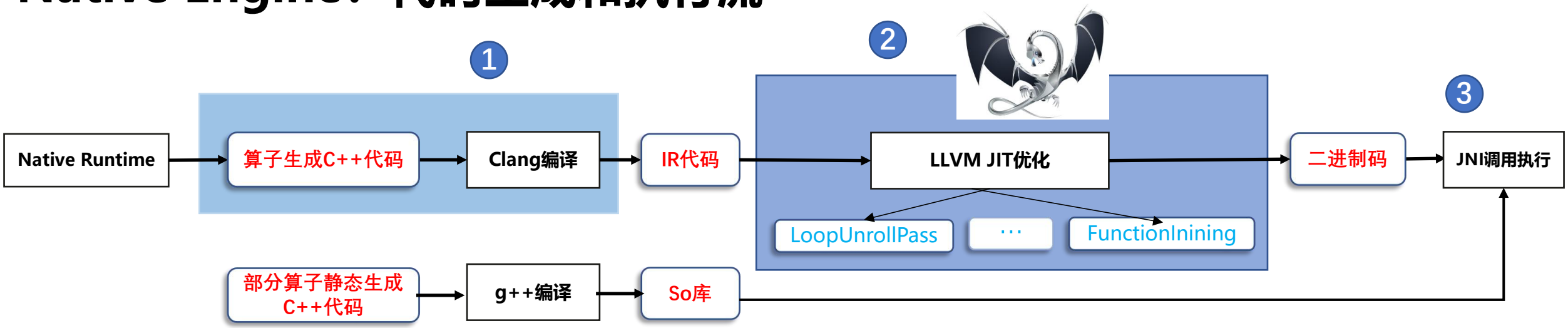
Native Engine：运行机制



- Analysis：主要利用Catalog信息将Unresolved Logical Plan解析成logical plan；
- Logical Optimizations：利用一些 Rule（规则）将logical plan解析成Optimized Logical Plan；
- Physical Planning：前面的logical plan不能被Spark执行，而这个过程是把logical plan 转换成多个physical plans，然后利用代价模型（cost model）选择最佳的physical plan；
- CodeGen：这个过程会把SQL查询生成**Java字节码**。

- **插件化运行**：使用Spark的extensions机制实现插件化运行，不修改Spark源码。
- **Fallback机制**：支持Fallback功能，算子涉及类型、表达式不支持可fallback到JVM Runtime，提高稳定性。

Native Engine：代码生成和执行流



➢**代码生成**：动态代码生成 + 静态编译

- 动态代码生成：算子根据运行时数据类型等动态生成C++代码，使用Clang完成词法、语法分析，语义分析，生成IR代码。相对于静态编写方式代码量减少了分支判断等处理，极大减少iCache miss。
- 静态编译：使用性能优异的第三方库，增加架构灵活性。

➢**LLVM JIT 代码优化**：使用LLVM Backend Optimizer优化器优化IR代码，生成对应架构的执行代码，包括了循环展开、函数内联等。

➢**代码执行**：通过JNI调用二进制编码执行，过程中动态调用静态编译so库。

Native Engine: 生成代码示例

- SQL语句：select C_CUSTKEY + C_NATIONKEY from customer;

动态生成代码（片段）

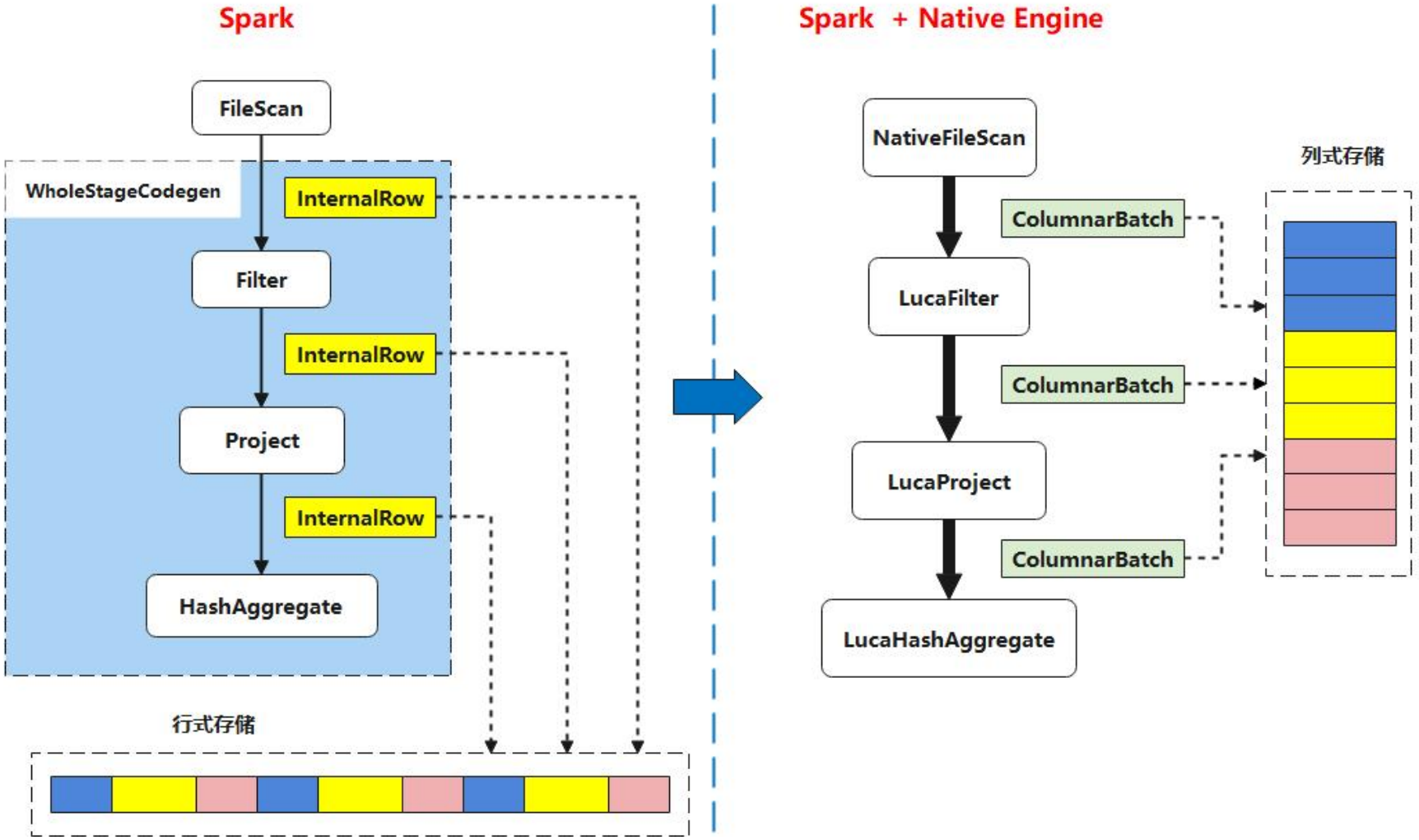
```
void project_0(SparkDataset* input,SparkDataset **output) {
  if(input == nullptr) {
    *output = nullptr;
  } else {
    int64_t total_rows = input->GetRowNumber();
    SparkFieldData** input_field_data_array = input->GetAllFieldDatas();
    SparkField* output_field_0 = builder_->MakeField("(C_CUSTKEY + C_NATIONKEY)",
                                                     SQL_TYPE_INTEGER, total_rows, true);
    SparkFieldData* output_field_data = output_field_0->GetFieldData();

    int* custkeyDatas = (int*)input_field_data_array[0]->datas;
    int* nationkeyDatas = (int*)input_field_data_array[1]->datas;
    int* outputDatas = (int*)output_field_data->datas;

    for(unsigned long int i = 0; i < total_rows; i++) {
      outputDatas[i] = custkeyDatas[i] + nationkeyDatas[i];
    }
    builder_->AppendField(output_field_0);
    *output = builder_->Finish(total_rows);
  }
}
```

汇编代码显示for循环向量化

```
      for(unsigned long int i = 0; i < total_rows; i++) {
f28:    9100818c    add x12, x12, #0x20
f2c:    910081ad    add x13, x13, #0x20
f30:    f10021ef    subs    x15, x15, #0x8
      outputDatas[i] = custkeyDatas[i] + nationkeyDatas[i];
f34:    4ea08440    add v0.4s, v2.4s, v0.4s
f38:    4ea18461    add v1.4s, v3.4s, v1.4s
f3c:    ad3f85c0    stp q0, q1, [x14,#-16]
```

LLVM JIT优化生成代码

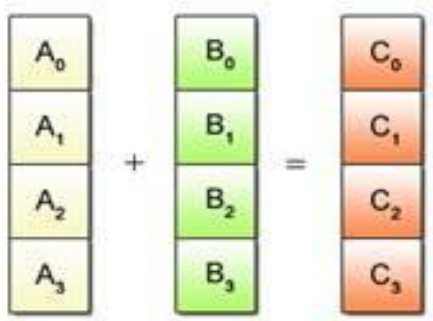
Native Engine：数据流从行式到列式实现批量处理



- 数据流从原生行式（InternalRow）转换成列式（ColumnarBatch）为单位在算子间传递，实现批量处理。
- 列式数据（ColumnarBatch）利于使用SIMD。

Native Engine：向量化

1. 列式数据利于算子SIMD向量化



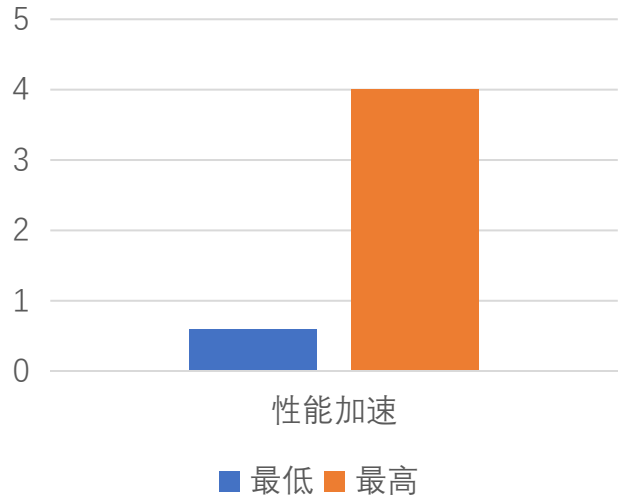
2. 借助编译器向量化能力进行优化

- 查看向量化：-fopt-info-vec-all

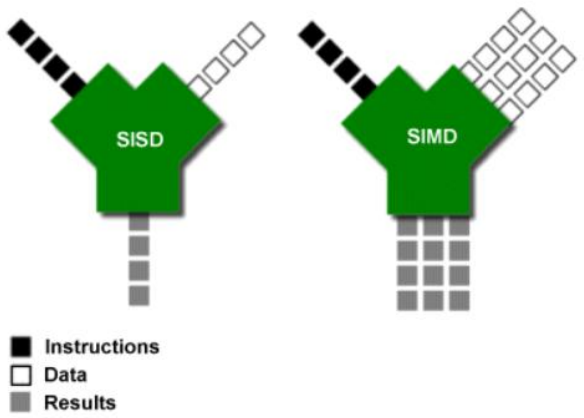
```
src/sort/sort_kernel.cpp:142:40: note: LOOP VECTORIZED
src/sort/sort_kernel.cpp:259:31: note: LOOP VECTORIZED
src/sort/sort_kernel.cpp:1472:31: note: LOOP VECTORIZED
src/sort/sort_kernel.cpp:1472:31: note: LOOP VECTORIZED
src/sort/sort_kernel.cpp:1472:31: note: LOOP VECTORIZED
src/sort/sort_kernel.cpp:1472:31: note: LOOP VECTORIZED
```

- 查看不能向量化：-fopt-info-vec-missed
  - control flow in loop
  - not enough data-refs in basic block
  - vectorization is not profitable
  - … …

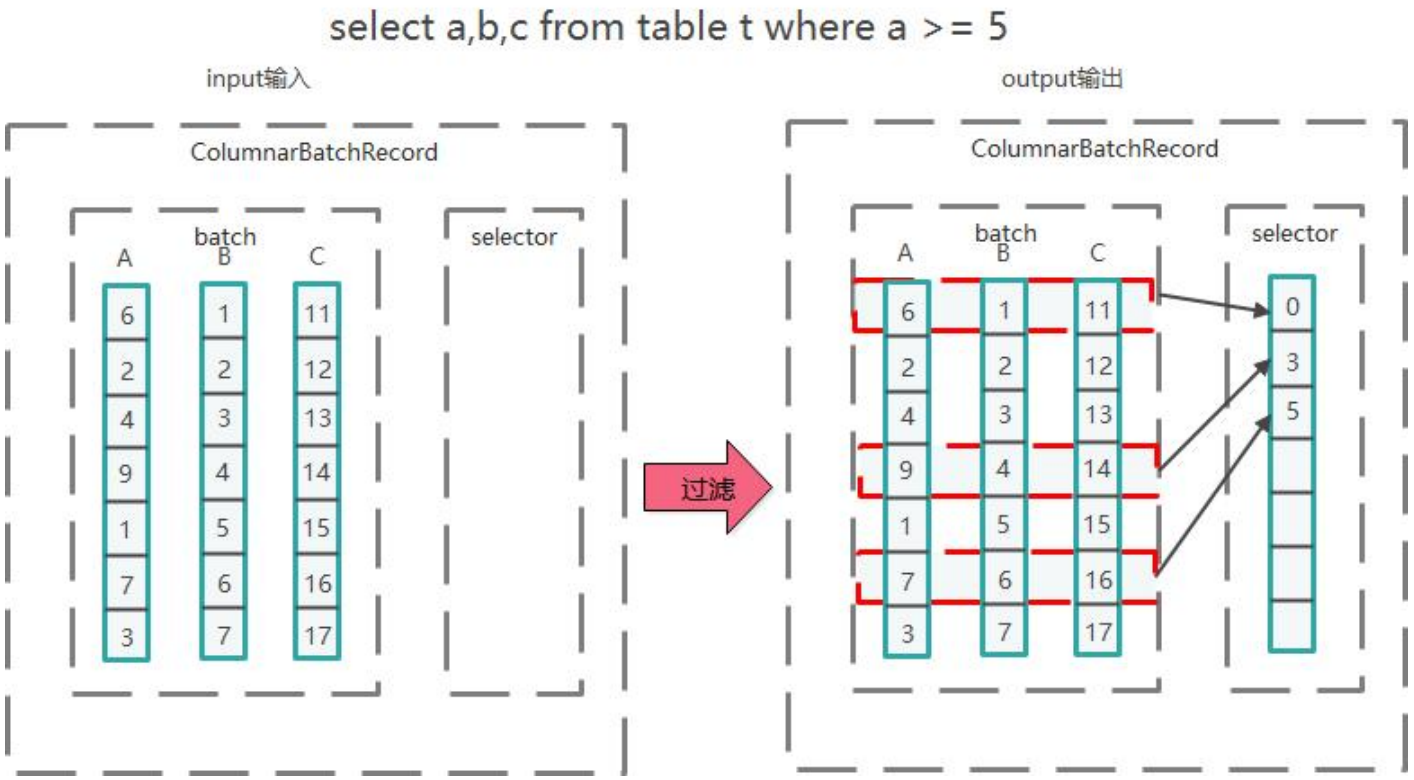
向量化算子性能加速0.6x-4x

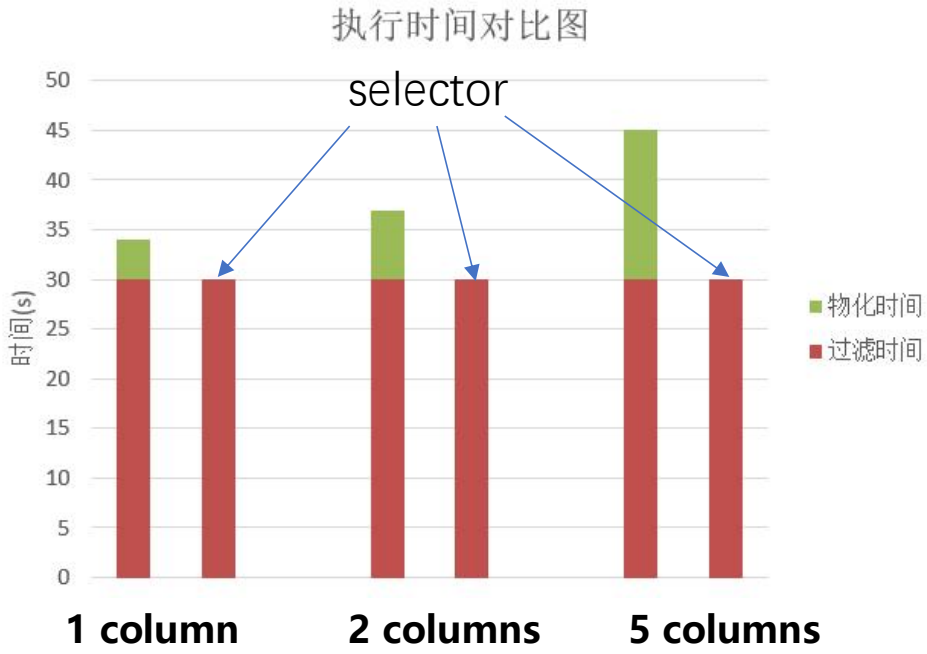


下一步计划: 算子Neon/SVE …



Native Engine：数据延迟物化

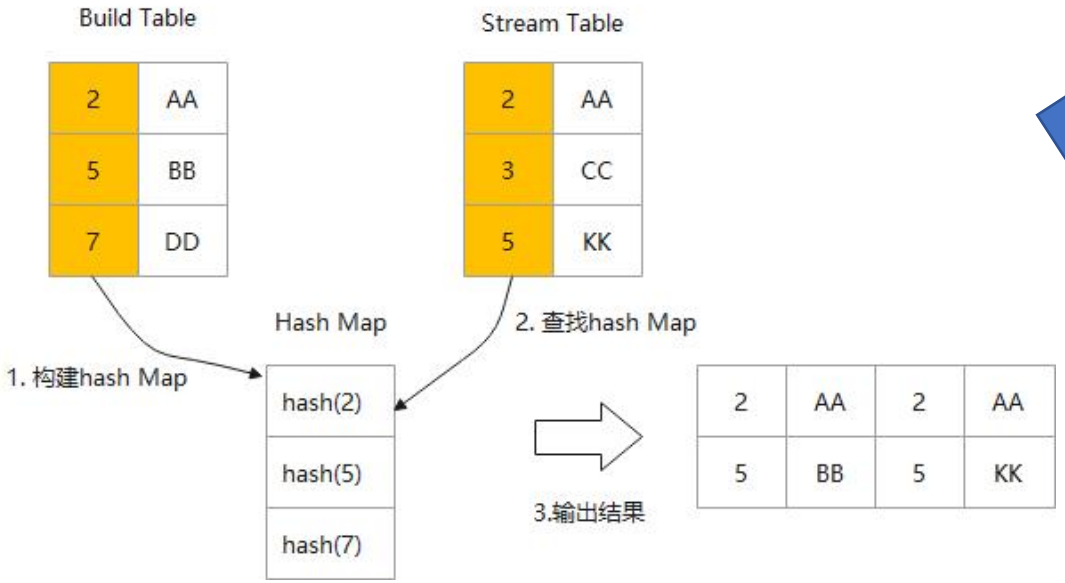


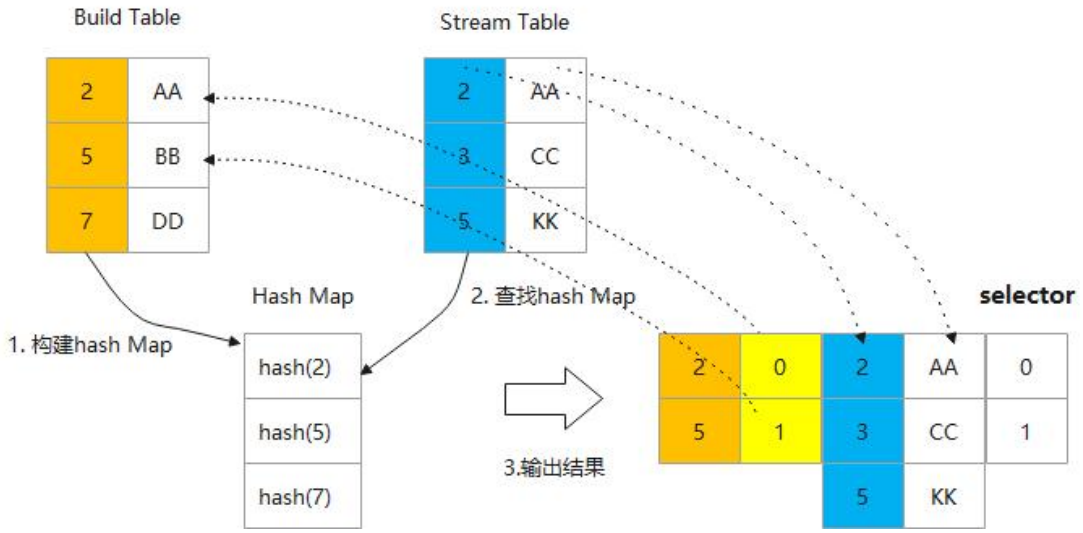


- 增加选择子selector，降低批处理中算子间数据物化开销。
- 列数越多，数据延迟物化性能表现越明显。经测试大数据量下，E2E性能提升30%+

Native Engine：算子优化

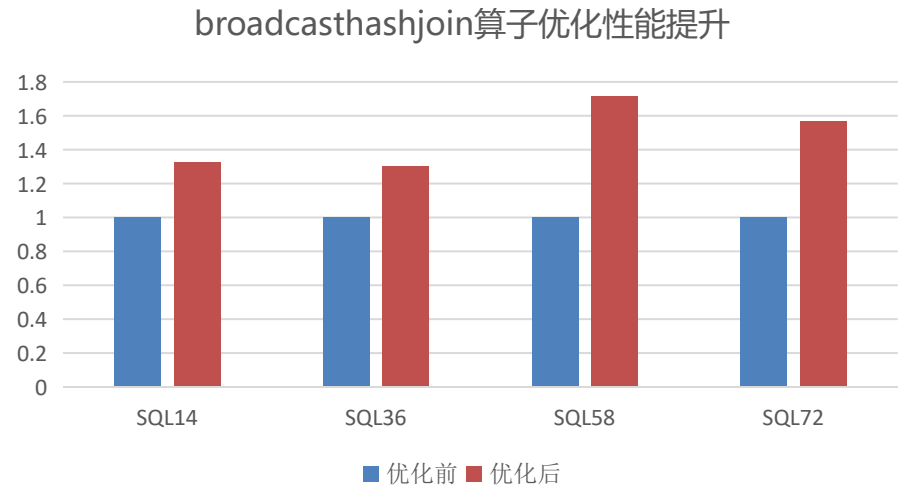
Broadcasthashjoin算子是Spark SQL重要算子之一，其主要功能是实现两张表的Join操作，流程是使用build table以join列的值为key构建Hash Map，然后遍历stream table每一行的join列值进行hash查找，如查找到根据输出attribute输出对应的行。



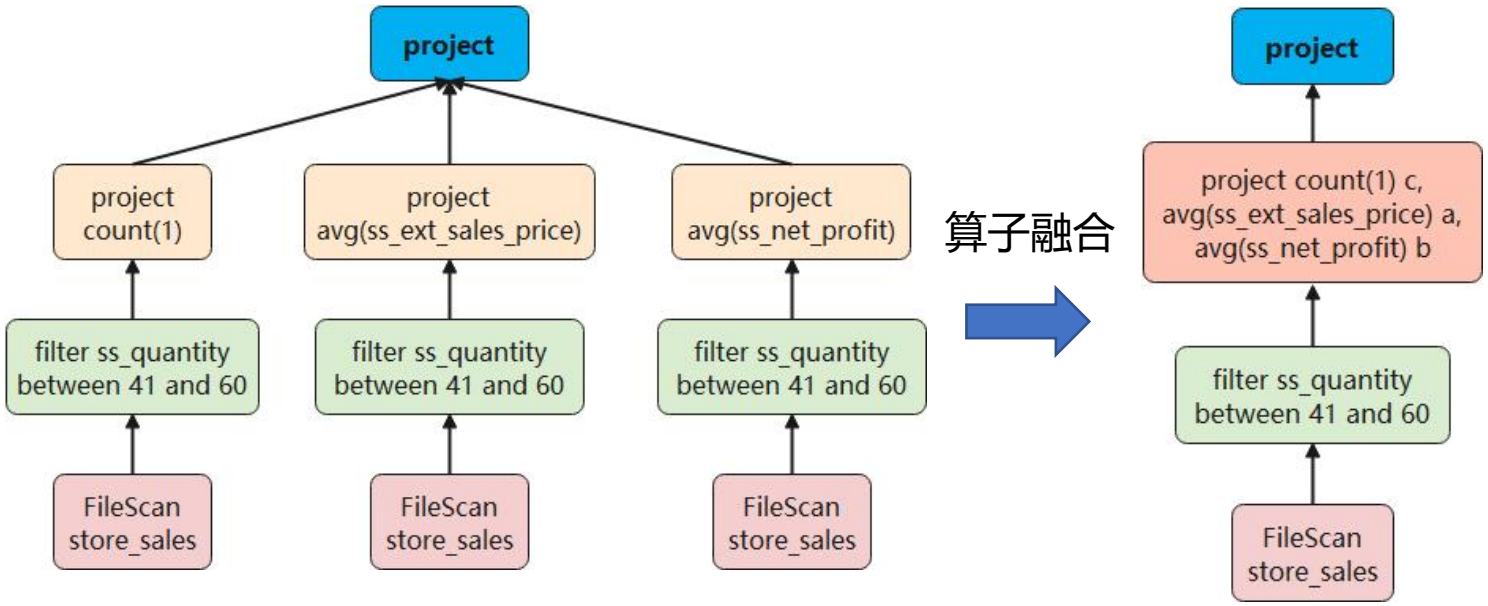


性能问题分析：

根据分析，输出结果处理流程耗时占比**60%+**，导致性能劣化。



Native Engine：算子融合



在执行计划优化中，引入多种算子融合规则，消除冗余子查询操作：

- **单算子融合**

  如project下expression融合优化等

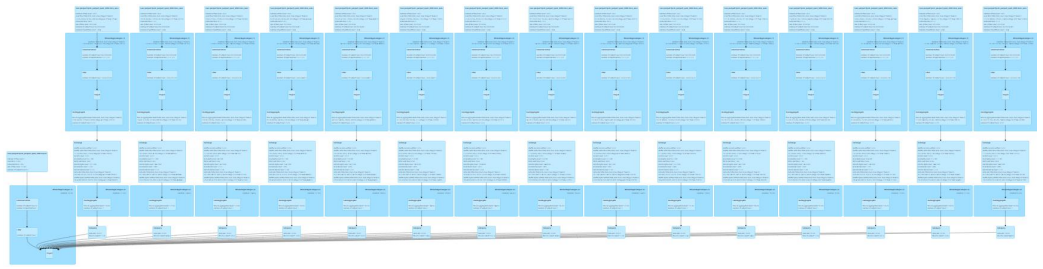
- **多算子融合**

  FileScan + FlieScan

  Project + Project

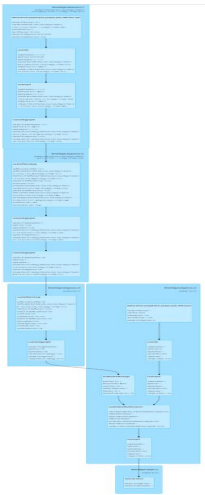
  … …

TPC-DS SQL 9融合前执行计划

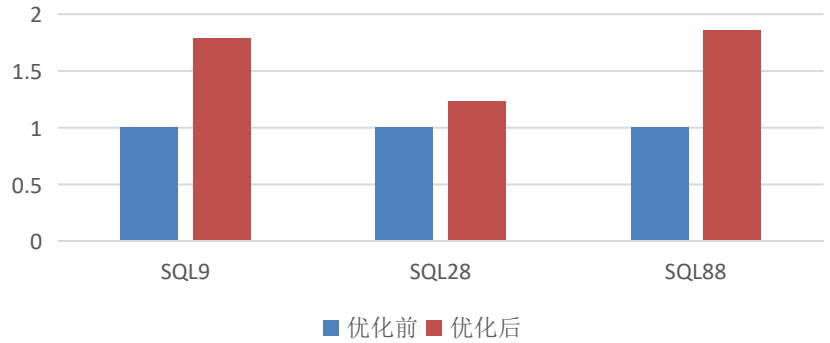


算子融合

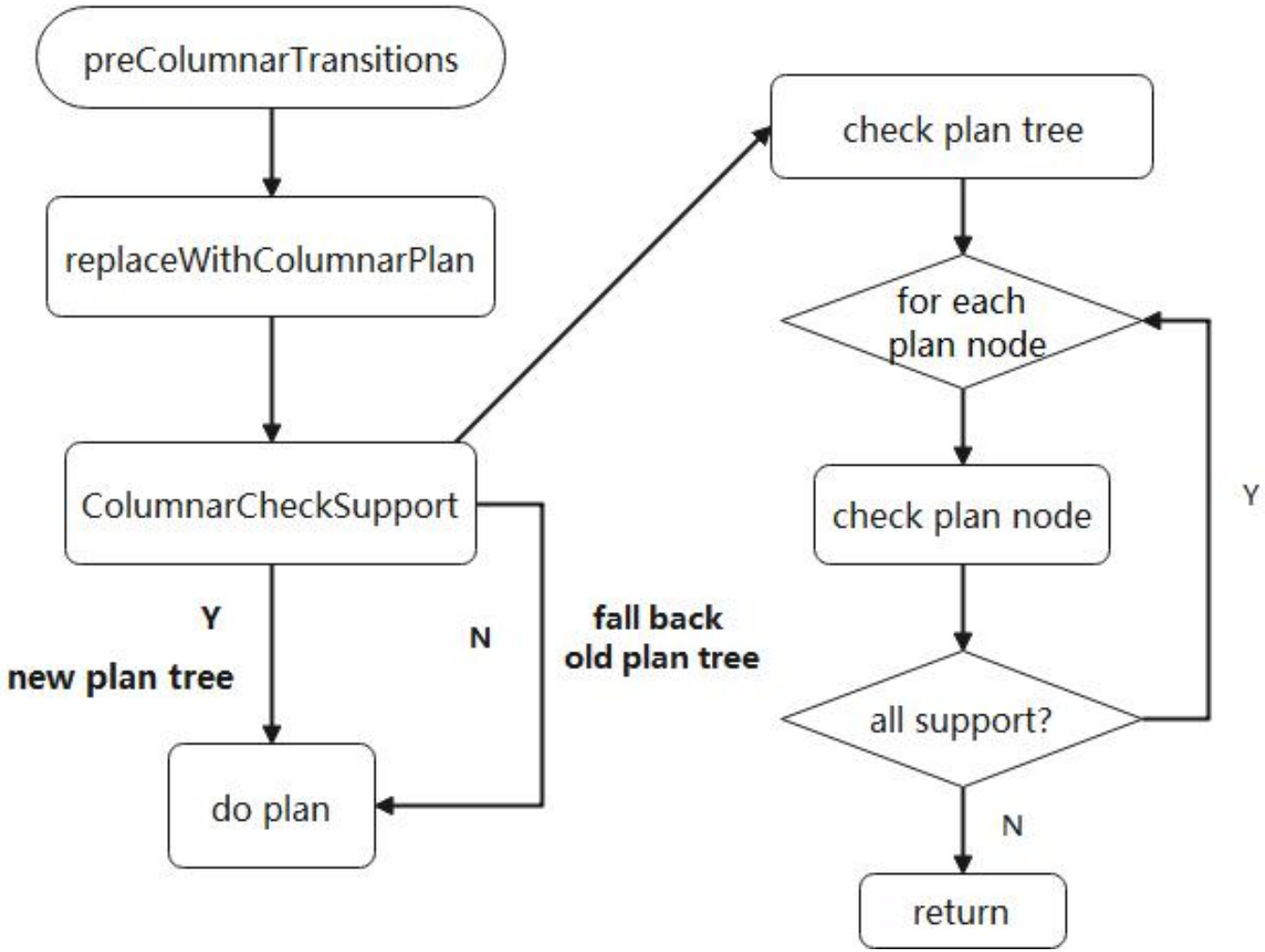
TPC-DS SQL 9融合后执行计划



算子融合优化性能提升



Native Engine：Fallback特性

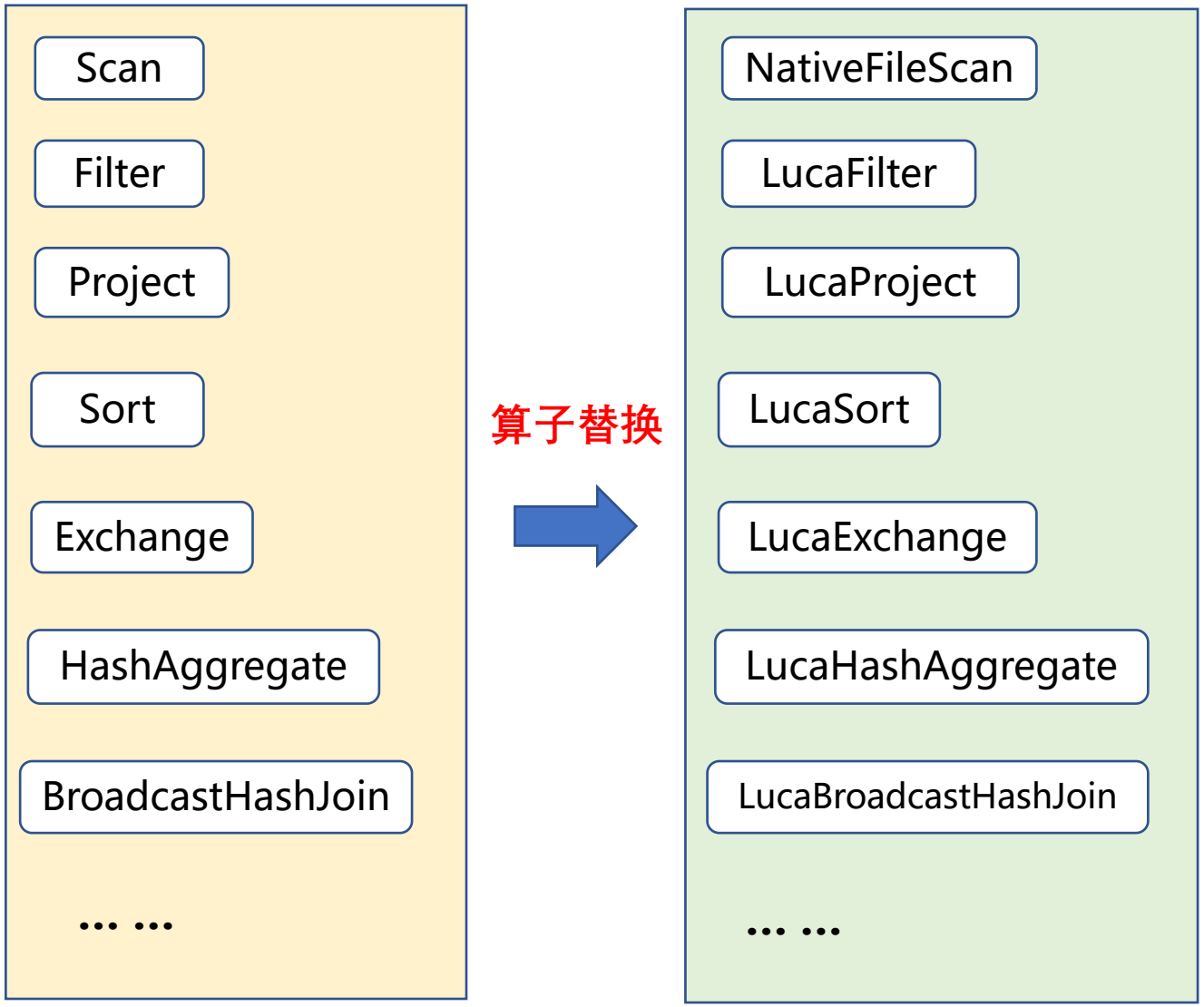


检查算子是否支持：

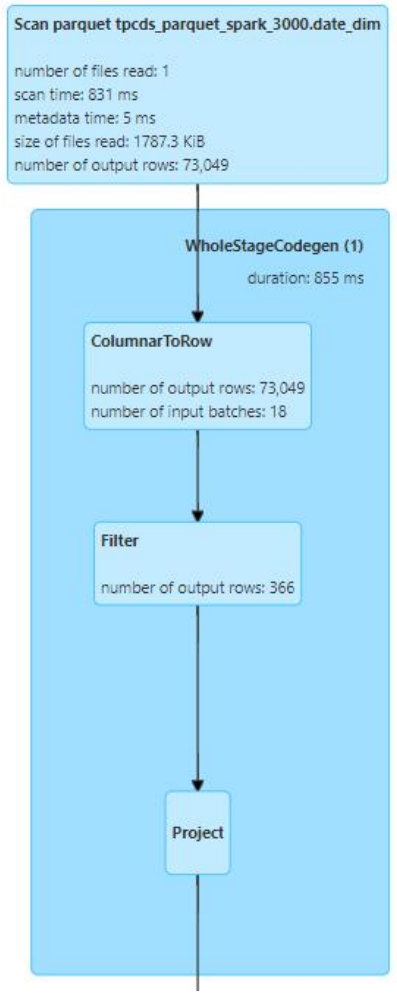
- 不支持的表字段数据类型
- 不支持的内置函数
- 不支持的表达式

NO → Fallback整个执行物理计划

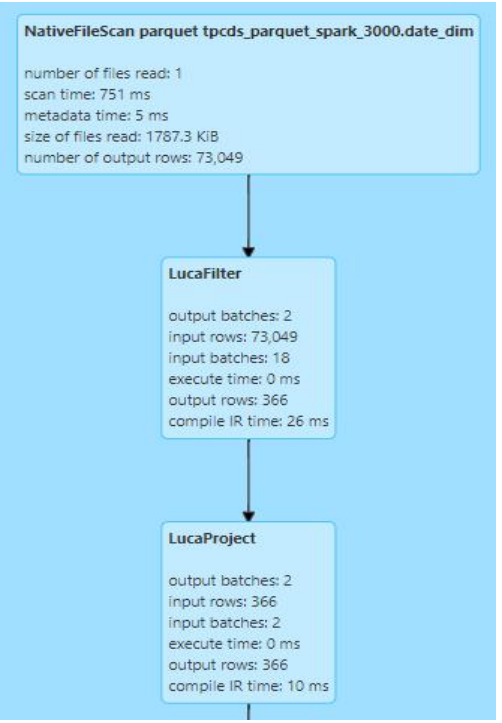
Native Engine：算子替换



Spark

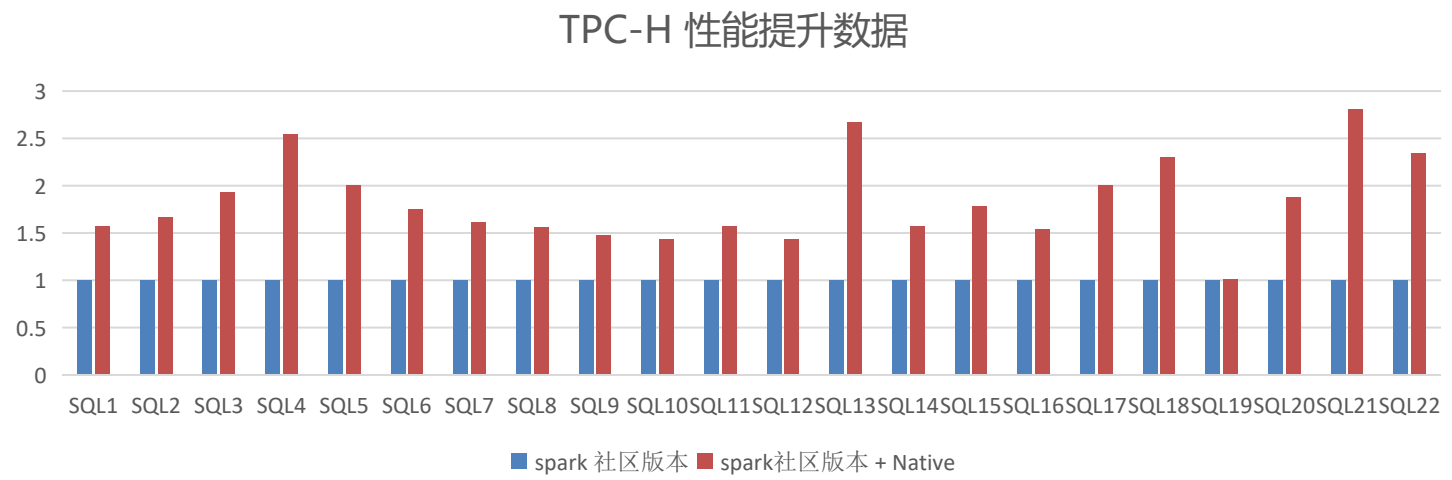


Spark + Native



- 整个过程列式ColumnarBatch形式传递数据，优化掉ColumnarToRow。
- 算子对应都替换成了Native版本算子：**Luca***列式算子。
- 部分Luca算子的新增了metric信息，包括处理数据量、时间等。

Native Engine：实验性能数据

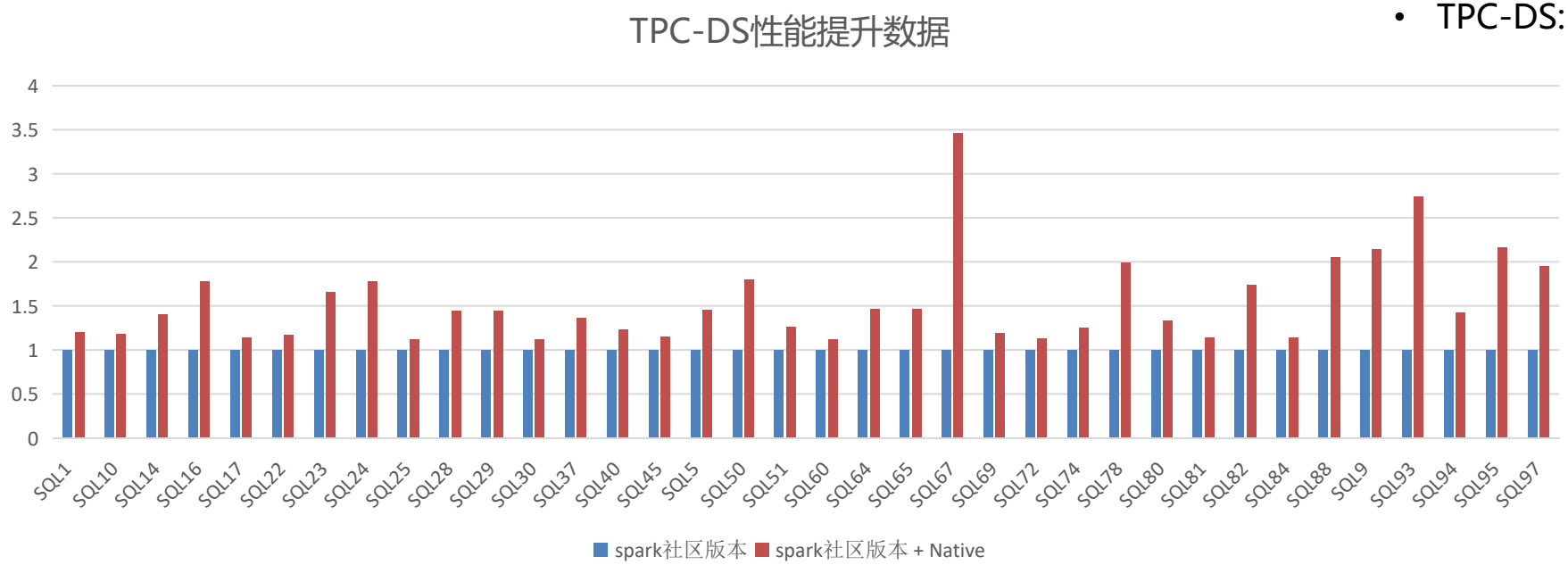


测试环境

- 集群：1 master + 3 slaves
- **鲲鹏服务器 + openEuler OS**

测试结果(3000SF)

- TPC-H： SQL21 **~3X**
- TPC-DS: SQL67 **3.X**



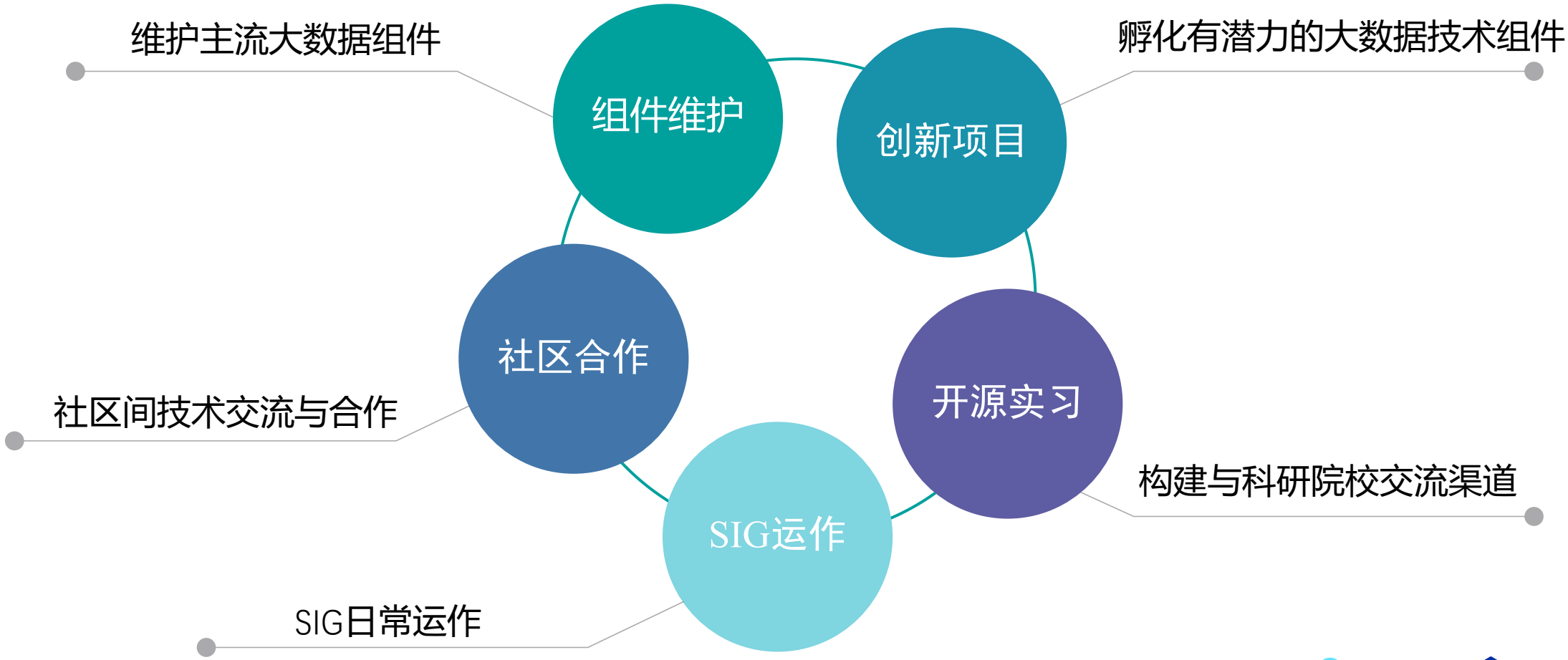
展望

openEuler社区仓库地址：https://gitee.com/openeuler/CuckooSQL

- 结合鲲鹏硬件加速器能力，卸载计算资源
- 进一步优化、完善和新增算子，支持更多的大数据应用场景
- 其它新技术点规划实施中 ……

Bigdata SIG简介

愿景：构建和完善openEuler社区大数据生态，打造活跃大数据技术交流和创新平台，挖掘软硬件能力释放大数据组件极致性能，孵化有潜力的大数据组件



如何参与Bigdata SIG

- 社区wiki： https://gitee.com/openeuler/bigdata/wikis
- 微信交流群：openEuler-bigdata-sig
- 双周例会： 周四16:00—17:00
- 技术月刊：提供大数据领域的最新技术动态
- 订阅Bigdata SIG邮件： https://mailweb.openeuler.org/postorius/lists/bigdata.openeuler.org

THANKS

开放原子开源基金会 OPENATOM FOUNDATION | OpenEuler