Yandex

Yandex

ClickHouse Query Execution Pipeline

Nikolai Kochetov
ClickHouse developer

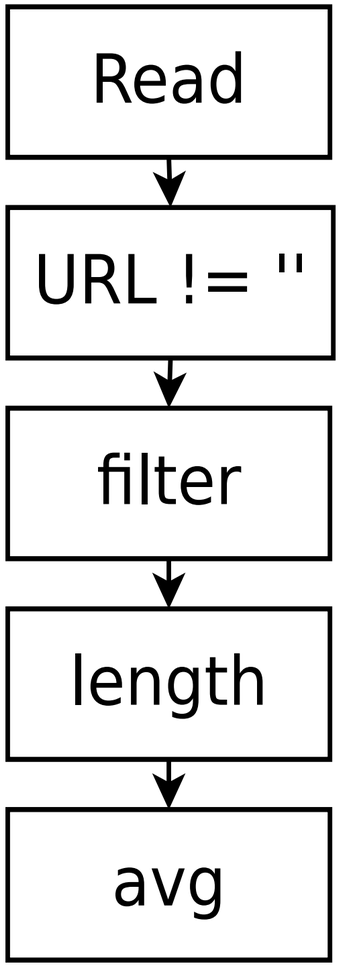
Execution Pipeline in DBMS

Query Example

```
SELECT avg(length(URL)) FROM hits WHERE URL != ''
```

Independent execution steps

- Read column `URL`
- Calculate expression `URL != ''`
- Filter column `URL`
- Calculate function `length(URL)`
- Calculate aggregate function `avg`

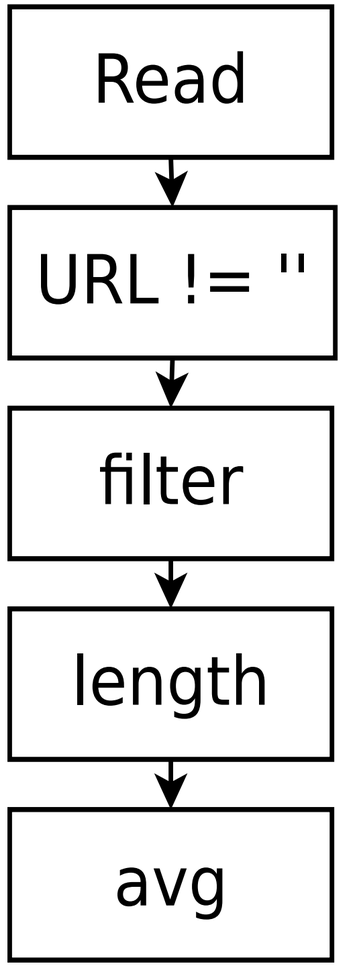


Pipeline

```
SELECT avg(length(URL)) FROM hits WHERE URL != ''
```

Chain (tree, graph) of steps with

- Simple operations
- Clear interpretation
- Parallel execution



Pipeline execution (Data flow)

In-memory execution (LocustDB)

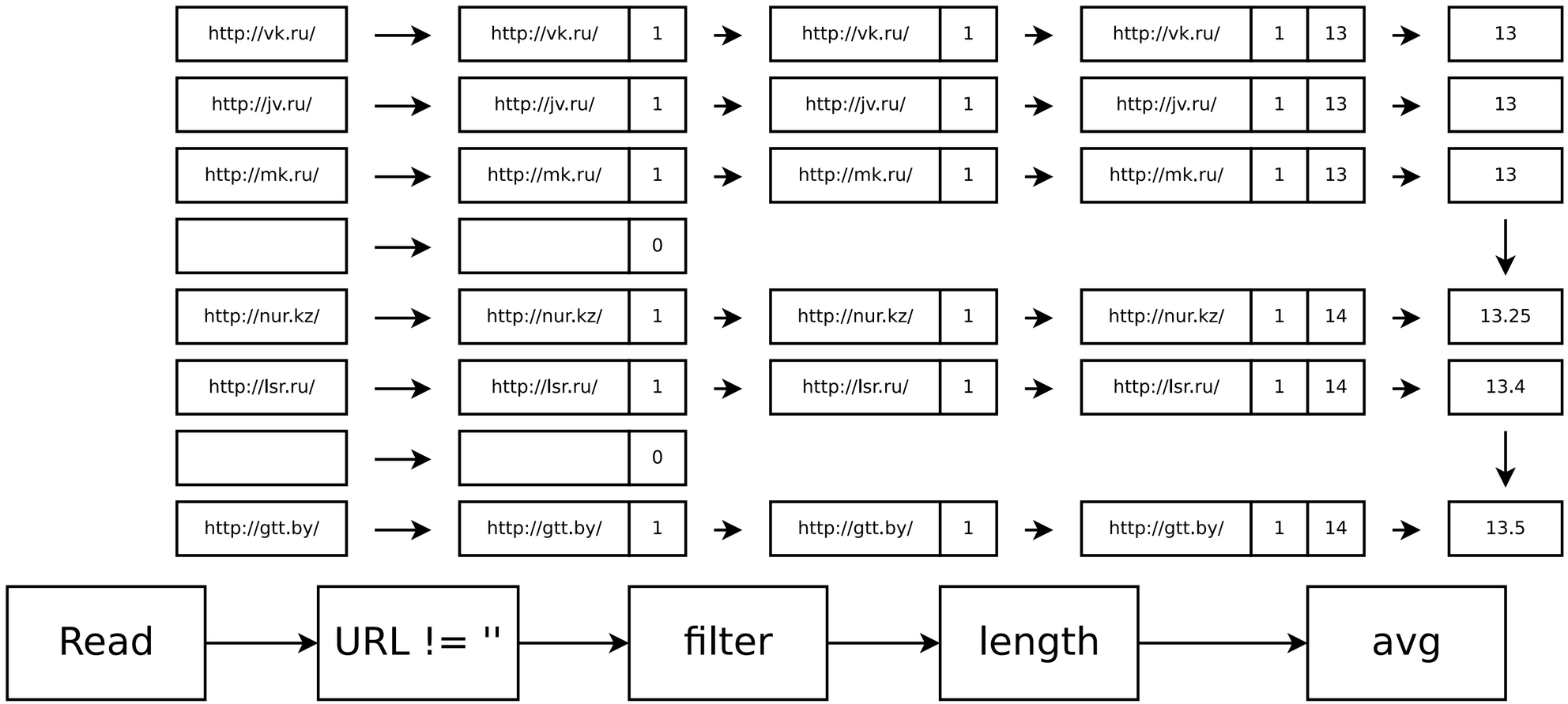
- Sort in topological order
- Run each step for all query data

Properties

- Simple
- Fast
- For in-memory databases

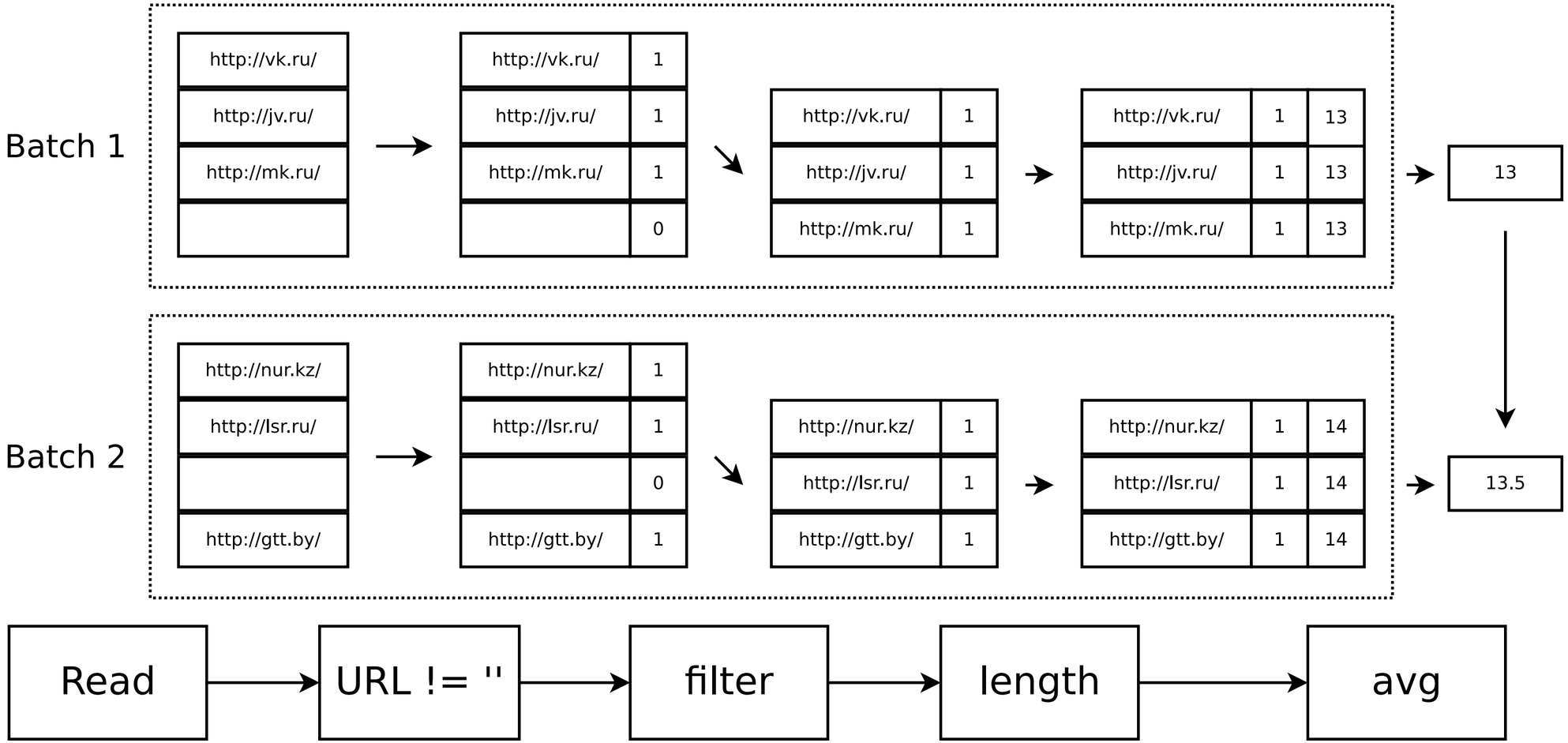
Pipeline execution (Data flow)

Row by row execution (MySQL, Postgres)



Pipeline execution (Data flow)

Batch execution (MonetDB, ClickHouse)



Pipeline execution (Data flow)

Row by row execution

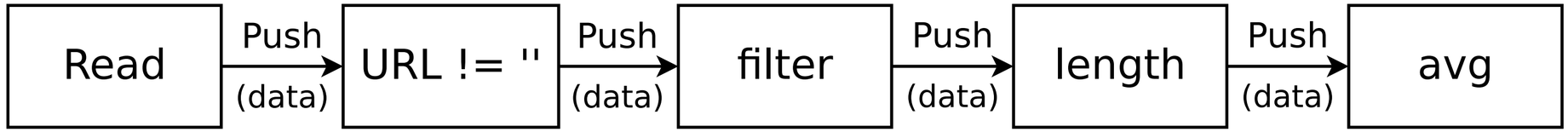
- Simple
- High overhead

Batch execution

- Small overhead
- Vectorized execution
- Greater memory consumption

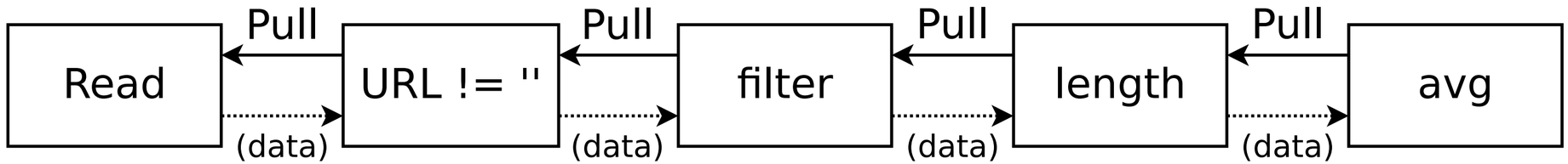
Pipeline execution (Logistics)

Push strategy



ClickHouse: IBlockOutputStream

Pull strategy

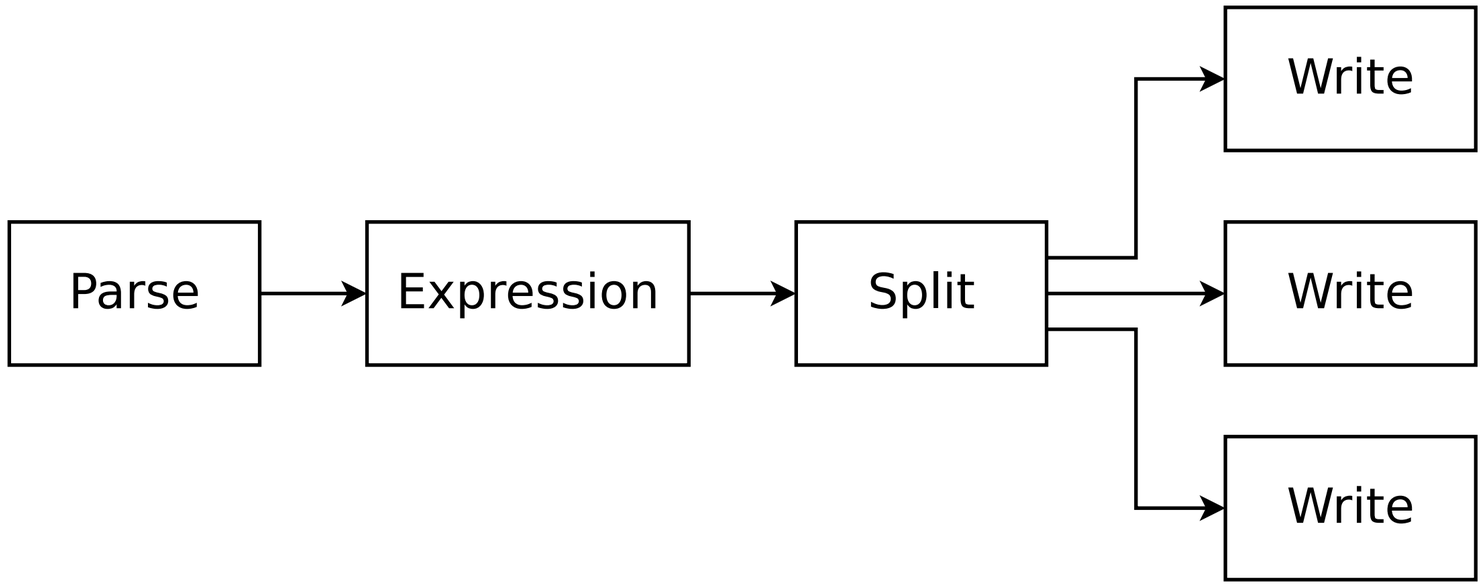


ClickHouse: IBlockInputStream

Pipeline execution (Logistics)

Push vs Pull

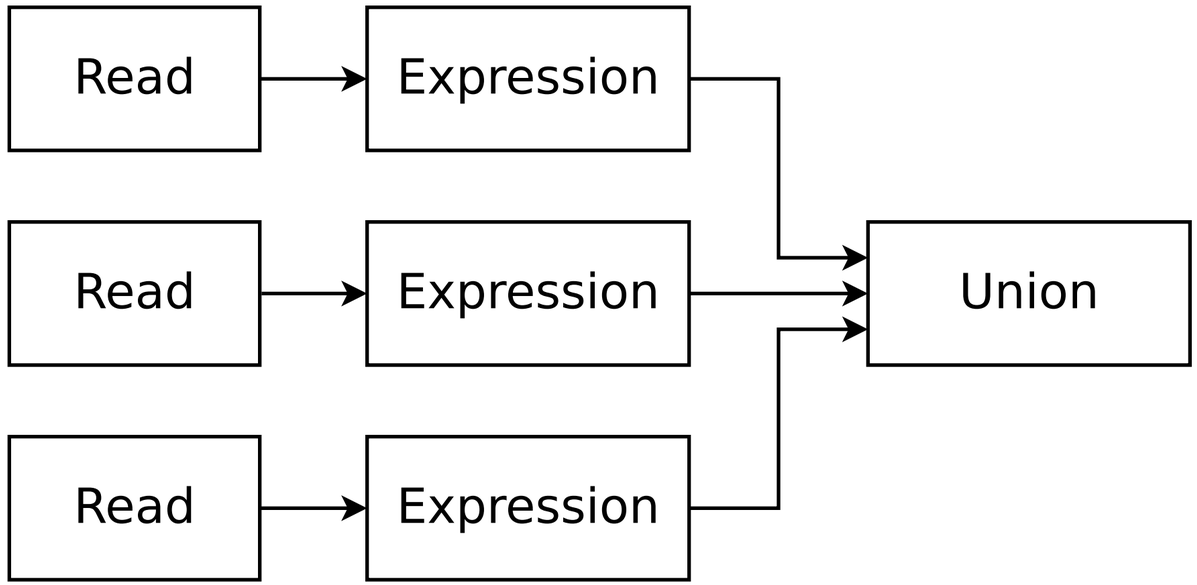
Insert query (into several partitions) - push



Pipeline execution (Logistics)

Push vs Pull

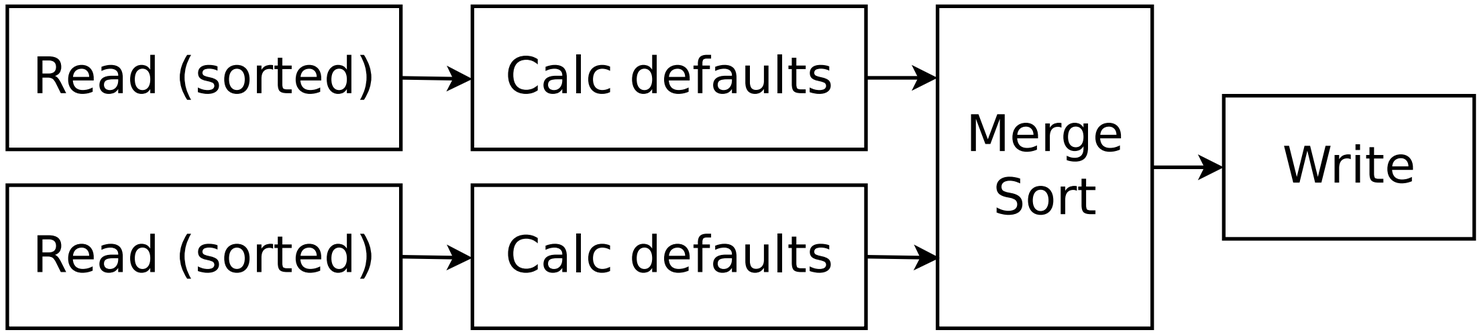
Select query (form several parts and order by) - pull



Pipeline execution (Logistics)

Push vs Pull

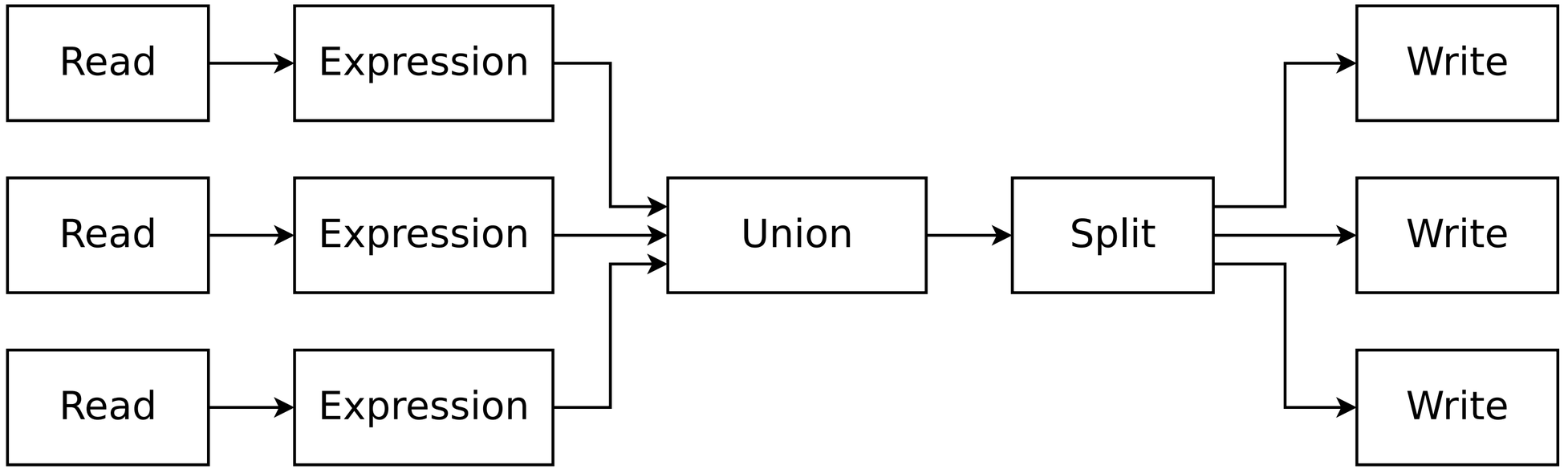
Merge parts - pull



Pipeline execution (Logistics)

Push vs Pull

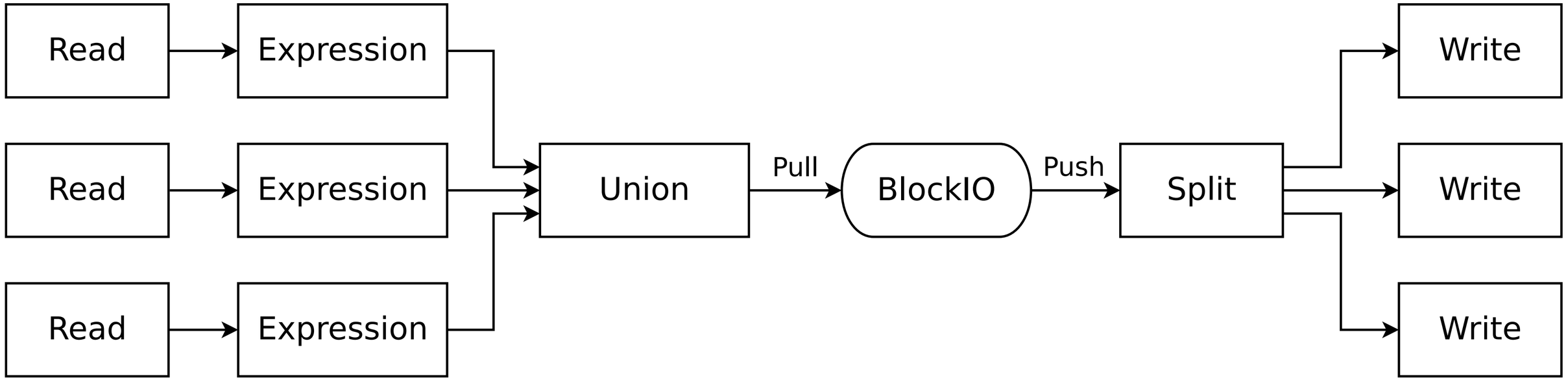
Insert select: difficult case



Pipeline in ClickHouse (current)

Mixed strategy

- IBlockInputStream - for pulling (SELECT)
- IBlockOutputStream - for pushing (INSERT)
- BlockIO - connection point



Pipeline in ClickHouse

Does current pipeline work well?

› Yes

Can it work better?

› Yes

New pipeline (in development)

```
SET experimental_use_processors = 1
```

Pipeline in ClickHouse

```
SELECT hex(SHA256(*)) FROM
(
    SELECT hex(SHA256(*)) FROM
    (
        SELECT hex(SHA256(*)) FROM
        (
            SELECT URL FROM hits ORDER BY URL ASC
        )
    )
)

10000000 rows in set. Elapsed: 23.227 sec.
```

> Use processors pipeline

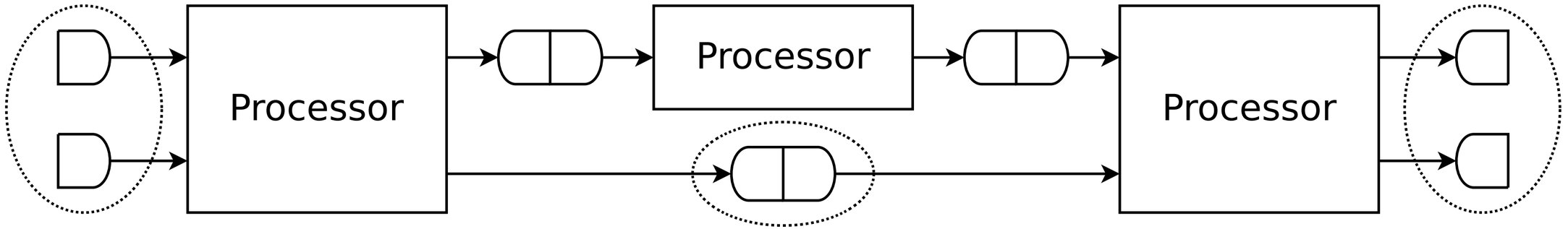
```
SET experimental_use_processors = 1

10000000 rows in set. Elapsed: 10.599 sec.
```

Better Pipeline in ClickHouse

Processors

Pipeline is a directional graph

- Node - IProcessor
- Port (input or output) can store chunk of data
- Edge - a pair of connected ports



Processors

```
SELECT avg(length(URL)) + 1
FROM hits
WHERE URL != ''
    WITH TOTALS
SETTINGS extremes = 1
```

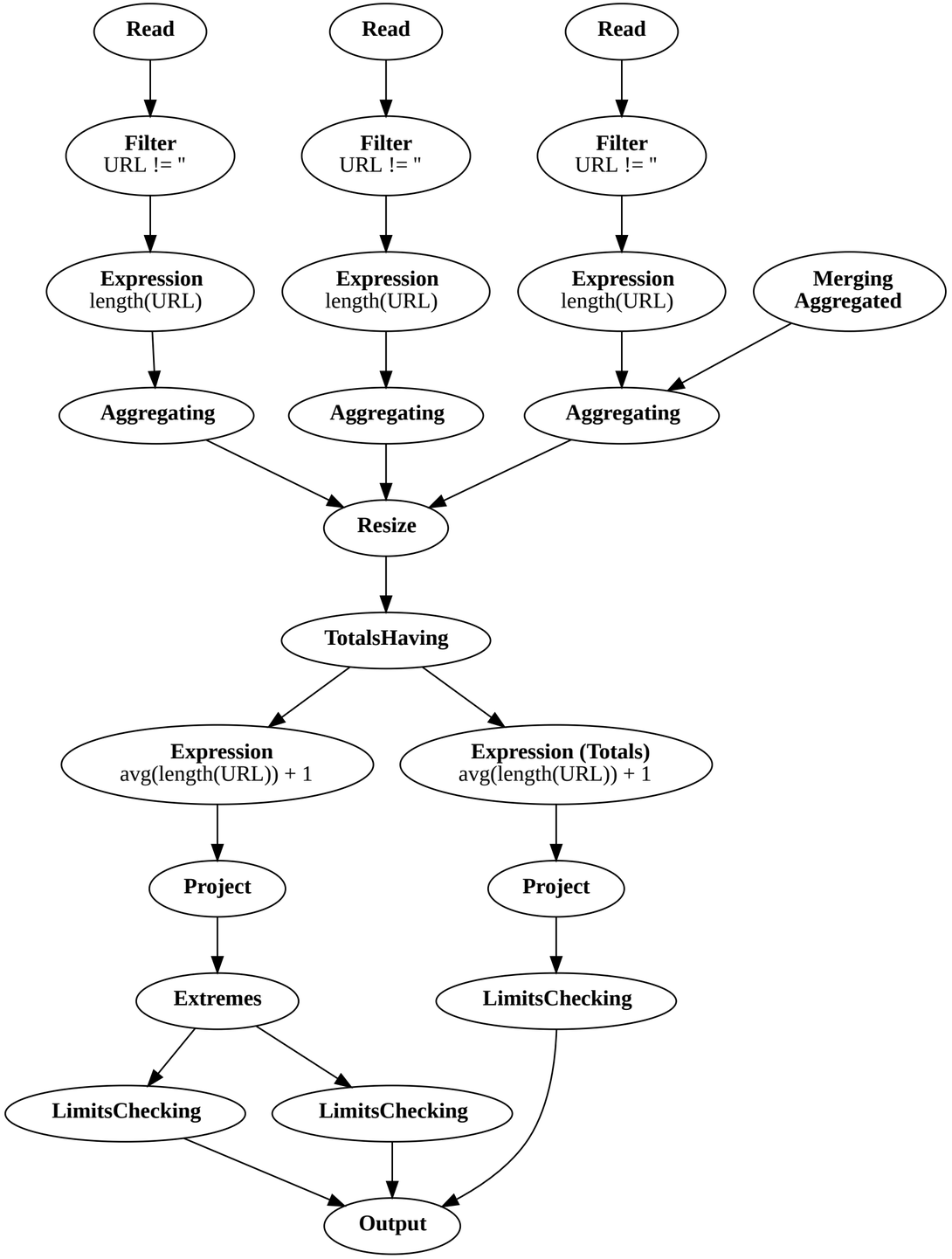
```
┌─plus(avg(length(URL)), 1)─┐
│          85.3475007793562 │
└───────────────────────────┘
```

Totals:

```
┌─plus(avg(length(URL)), 1)─┐
│          85.3475007793562 │
└───────────────────────────┘
```

Extremes:

```
┌─plus(avg(length(URL)), 1)─┐
│          85.3475007793562 │
│          85.3475007793562 │
└───────────────────────────┘
```



Pipeline execution

How to execute

- Traverse graph all the time
- Execute everything which can be executed
- Check ports state to visit neighbors

Why it works

- Batch execution => low traverse overhead
- Thread safe operations => parallelism

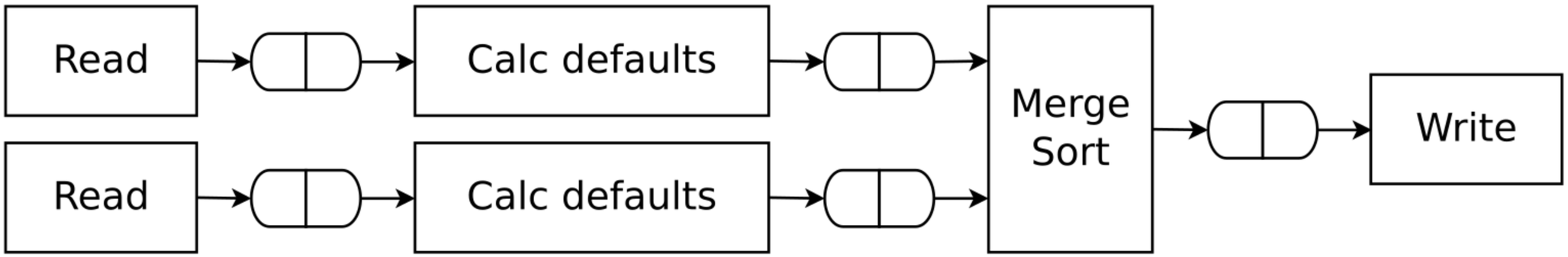
Pipeline execution

Processors

- White - need data
- Yellow - executing
- Red - waiting
- Gray - finished

Ports

- White - free
- Orange - has data



Parallel execution

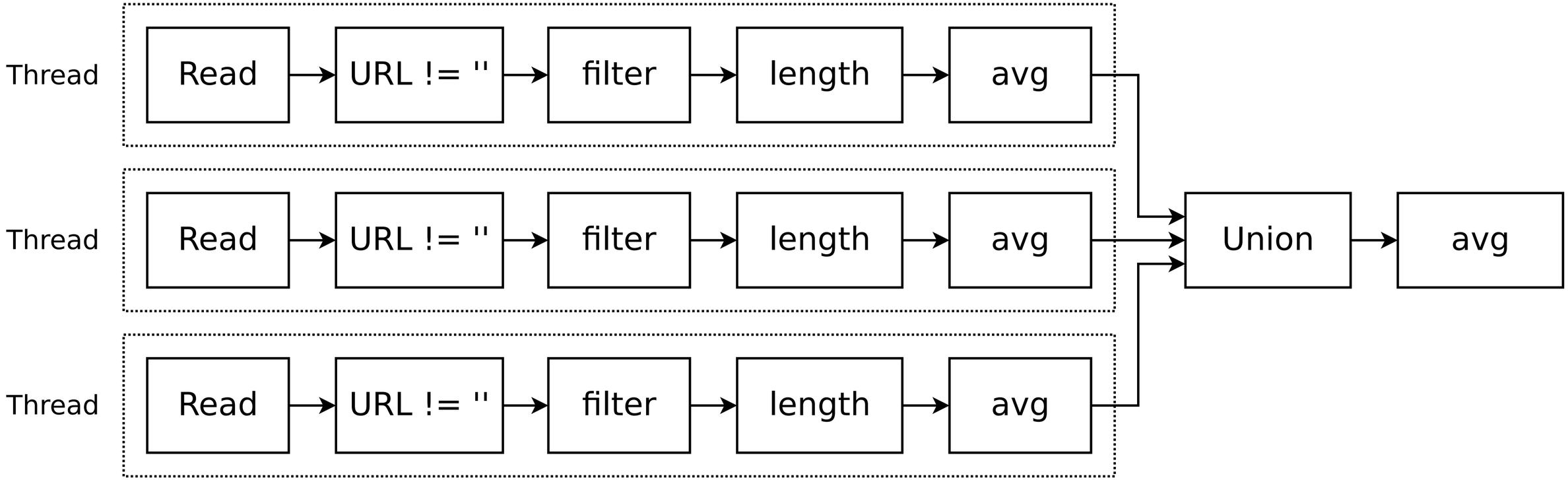
```
SELECT hex(SHA256(*)) FROM
(
    SELECT hex(SHA256(*)) FROM
    (
        SELECT hex(SHA256(*)) FROM
        (
            SELECT URL FROM hits ORDER BY URL ASC
        )
    )
)

10000000 rows in set. Elapsed: 23.227 sec.
```

> Use processors pipeline

```
SET experimental_use_processors = 1

10000000 rows in set. Elapsed: 10.599 sec.
```

Parallel execution

How ClickHouse executes queries in parallel?

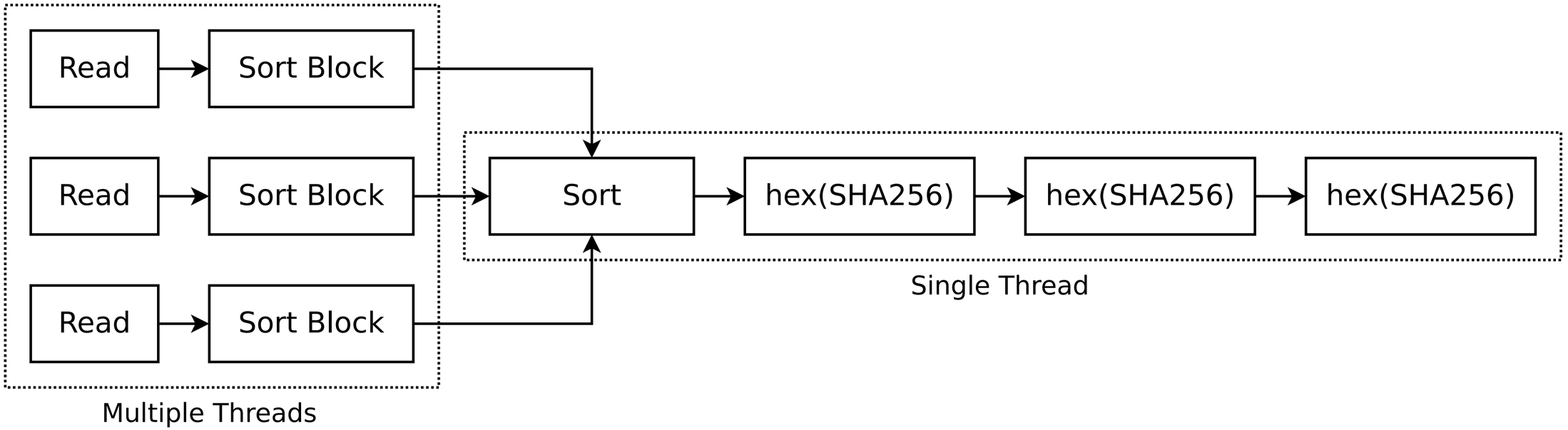


Copy pipeline for each thread

Parallel execution

Pull strategy (IBlockInputStream)

Query Pipeline

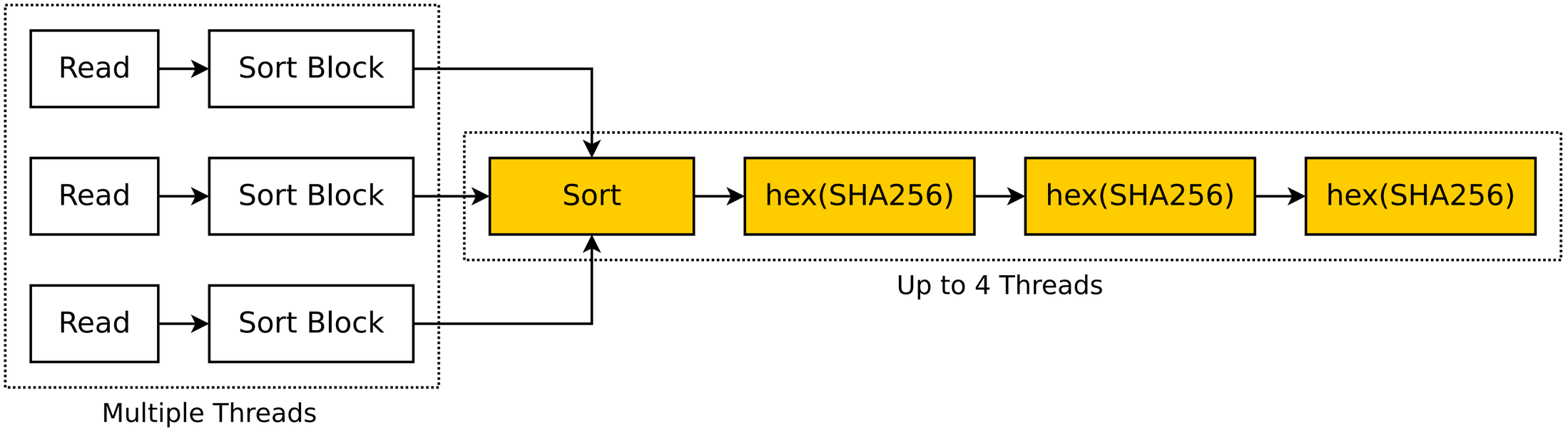


Part of pipeline is executed in single thread

Parallel execution

Graph traverse (Processors)

Query Pipeline

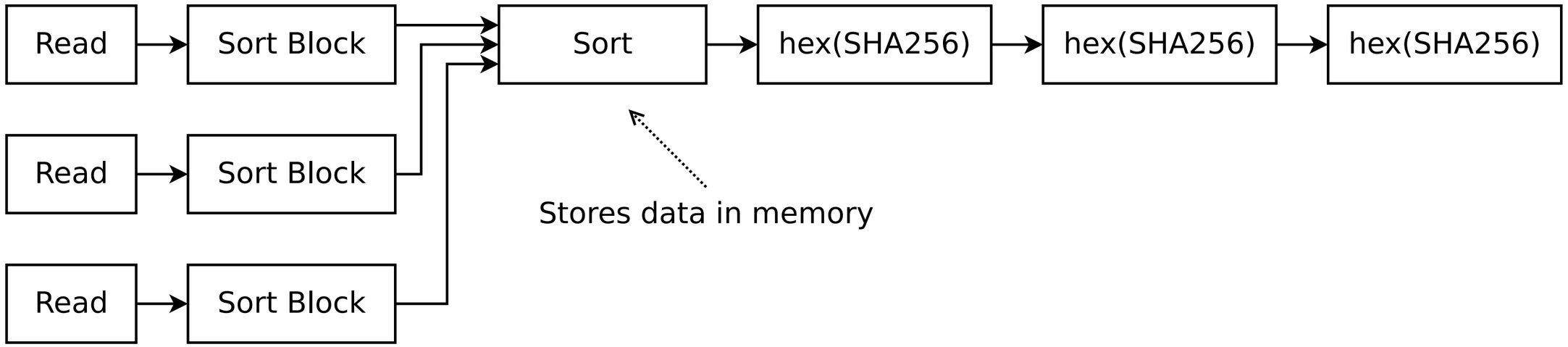


Right chain can be executed in 5 threads (best case)

Dynamic pipeline modification

Sometimes we need to change pipeline during execution

Use previous pipeline as example



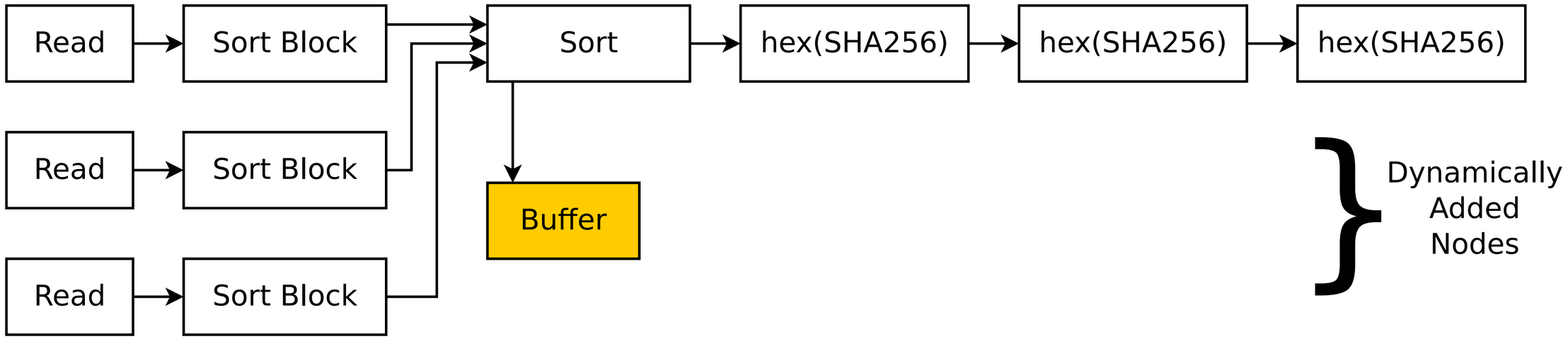
`Sort` stores all query data in memory

Set `max_bytes_before_external_sort = <some limit>`

Dynamic pipeline modification

Sometimes we need to change pipeline during execution

Use previous pipeline as example



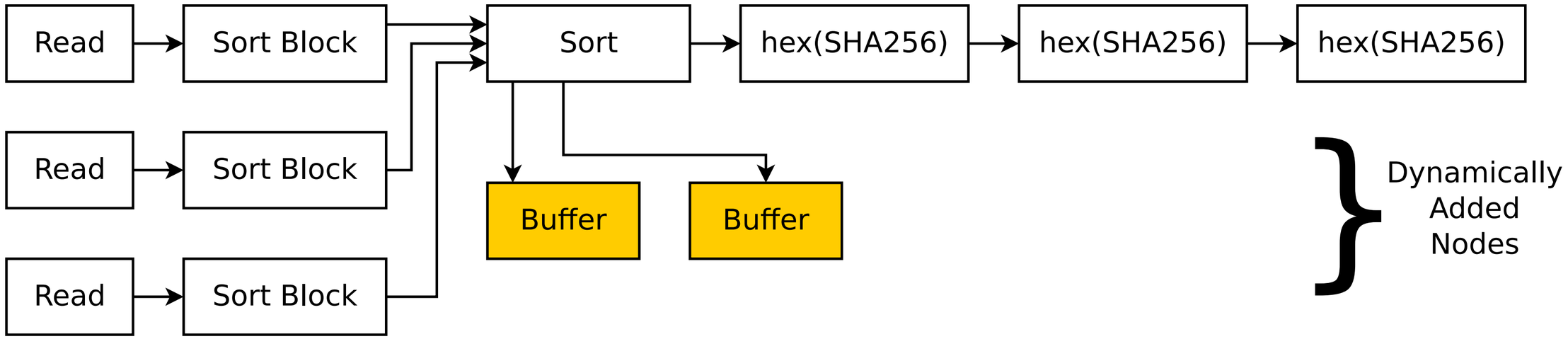
`Sort` stores all query data in memory

Set `max_bytes_before_external_sort = <some limit>`

Dynamic pipeline modification

Sometimes we need to change pipeline during execution

Use previous pipeline as example



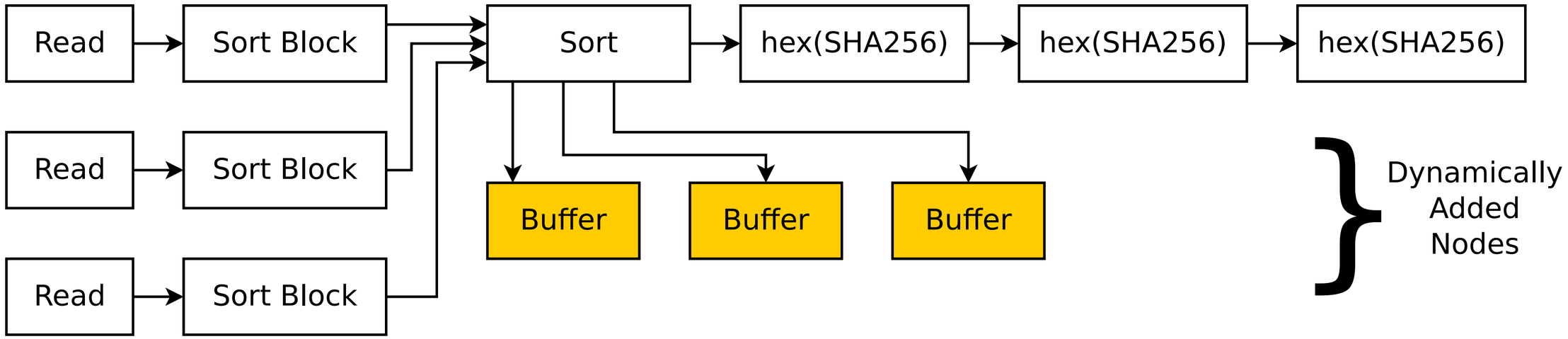
`Sort` stores all query data in memory

Set `max_bytes_before_external_sort = <some limit>`

Dynamic pipeline modification

Sometimes we need to change pipeline during execution

Use previous pipeline as example



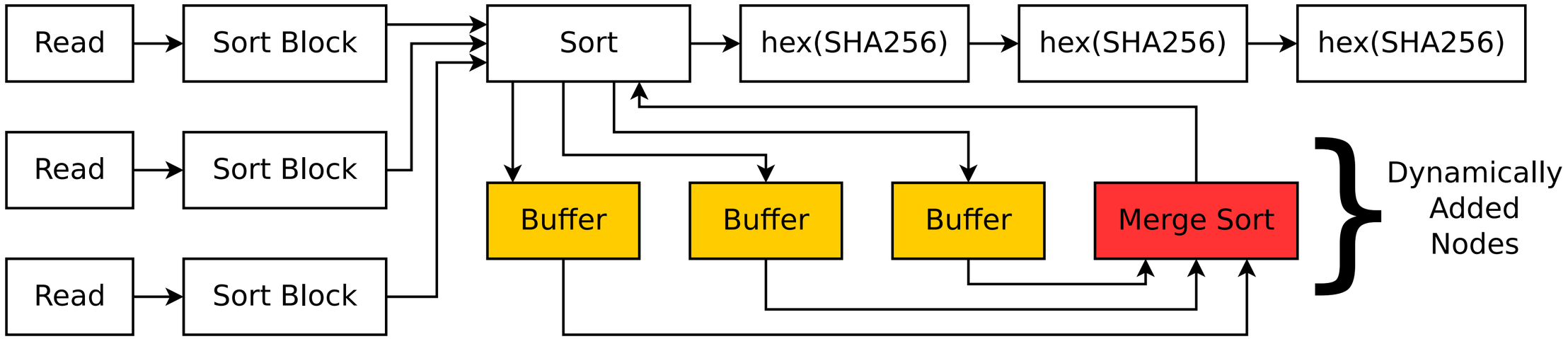
`Sort` stores all query data in memory

Set `max_bytes_before_external_sort = <some limit>`

Dynamic pipeline modification

Sometimes we need to change pipeline during execution

Use previous pipeline as example



`Sort` stores all query data in memory

Set `max_bytes_before_external_sort = <some limit>`

Planned features

Explain Query

Will print pipeline in Graphviz format

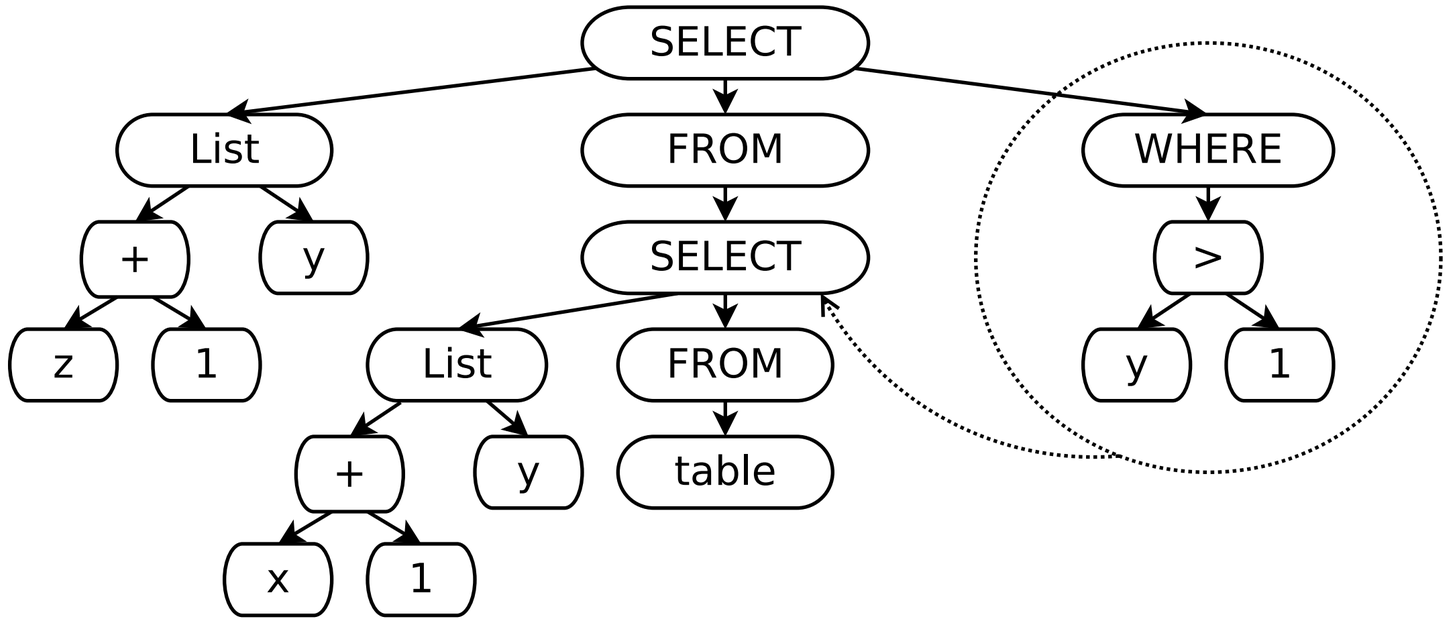
```
EXPLAIN SELECT avg(length(URL)) FROM hits WHERE URL != ''
```

```
digraph
{
  n140638219161104[label="SourceFromStorage"];
  n140638217764624[label="ExpressionTransform"];
  n140638219121680[label="FilterTransform"];
  n140638217764048[label="ExpressionTransform"];
  n140638217755024[label="AggregatingTransform"];
  n140638217763856[label="ExpressionTransform"];
  n140638219121360[label="LimitsCheckingTransform"];
  n140638142287888[label="ConvertingAggregatedToBlocksTransform"];
  ...
}
```

Push predicate to subquery

```
select z + 1, y from (select x + 1 as z, y from table) where y > 1
```

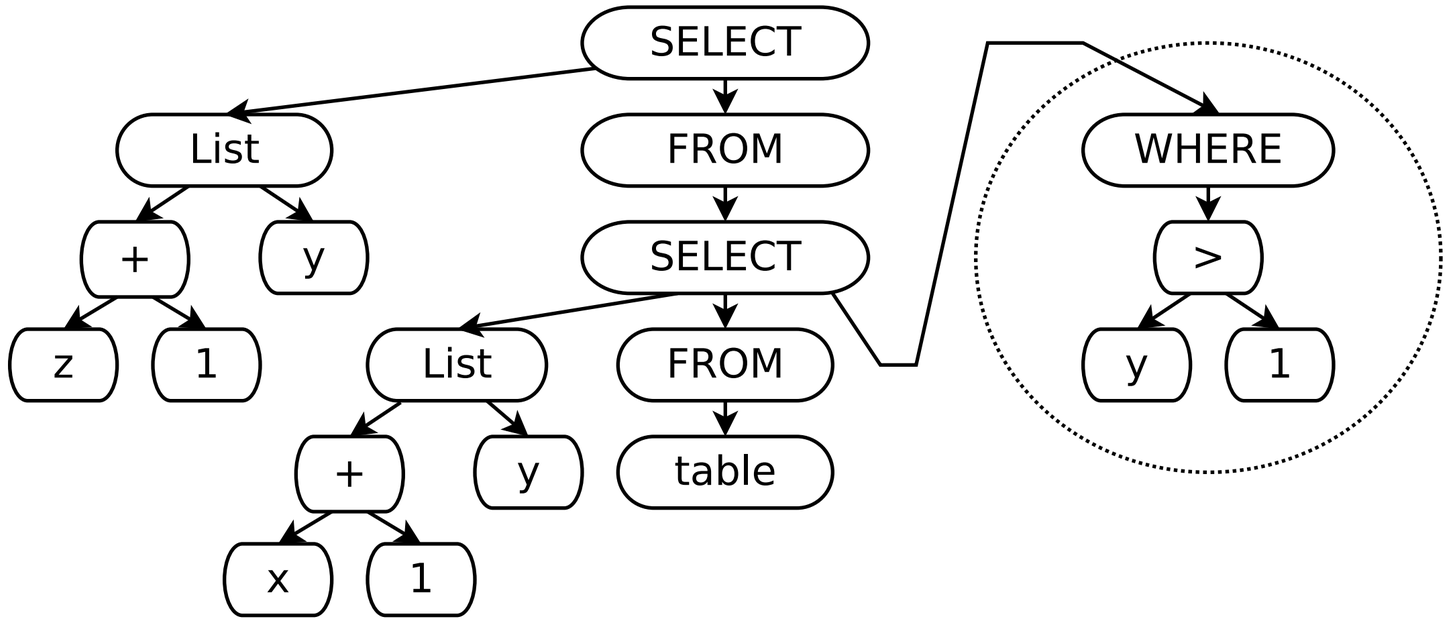
AST rewriting approach



Push predicate to subquery

```
select z + 1, y from (select x + 1 as z, y from table  where y > 1)
```

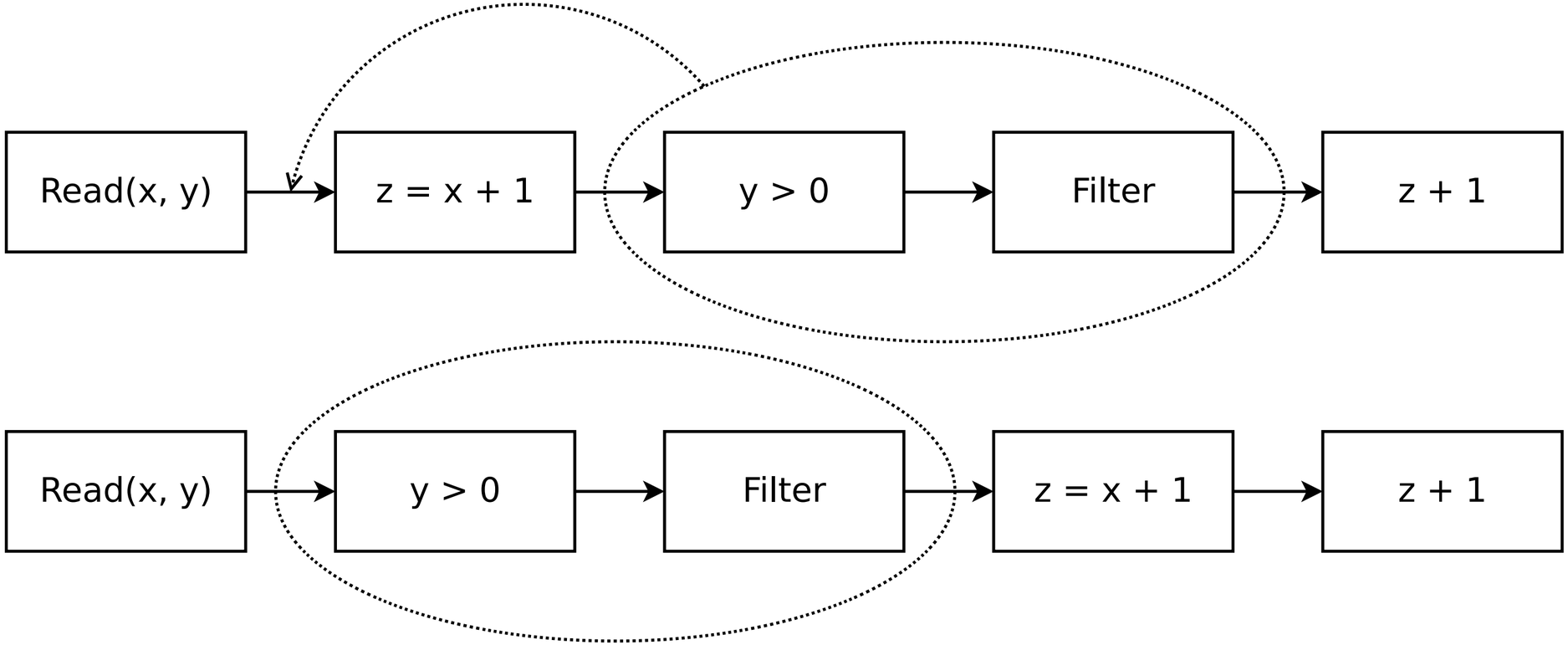
AST rewriting approach



Push predicate to subquery

```
select z + 1, y from (select x + 1 as z, y from table  where y > 1)
```

Pipeline optimization approach



Resource management

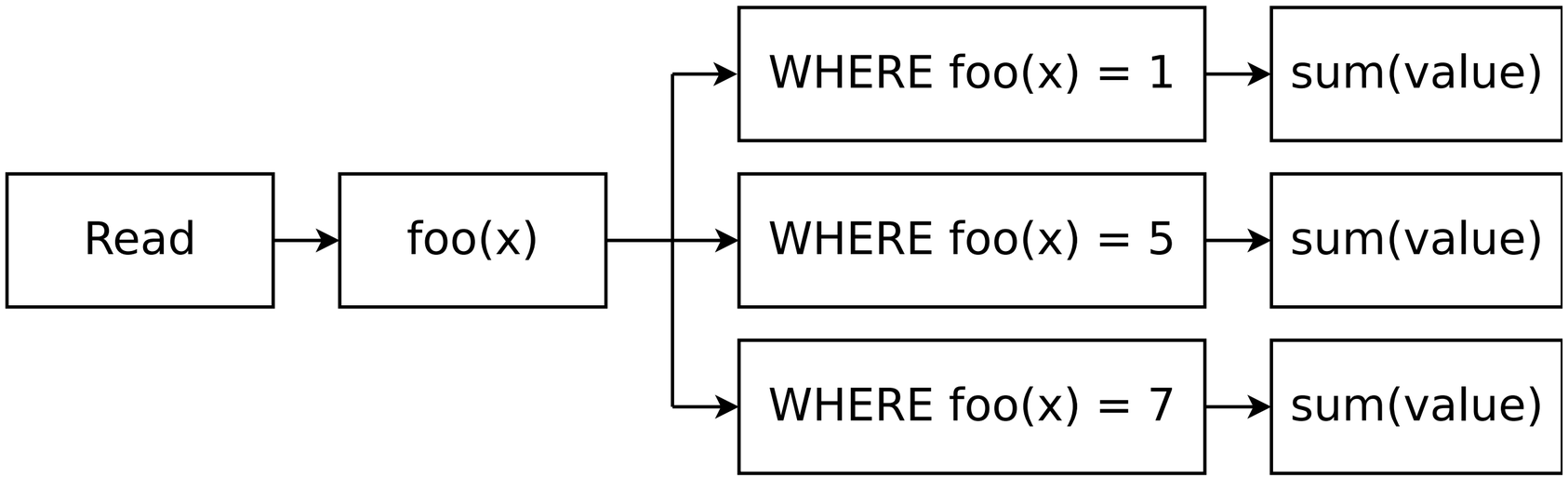
- Manage quota for users
  - Operations priority (high for API, low for analytics)
  - Limits on CPU, Memory, RPS, BPS
  - May enable only for high load
- Common executor for multiple pipelines

Common pipeline for several queries

Example set of similar queries

```
SELECT sum(value) FROM table WHERE foo(x) = 1;
SELECT sum(value) FROM table WHERE foo(x) = 5;
SELECT sum(value) FROM table WHERE foo(x) = 7;
```

> It’s possible to make common pipeline for several queries



Pipeline definition language (DSL)

Idea

- Define text language for pipelines
- Support bidirectional conversions

Features

- Can send ready pipeline to replicas
- Low level interface
- Pipelines which are not representable is SQL

Contacts

 Web site: https://clickhouse.yandex

 Google groups: https://groups.google.com/forum/#!forum/clickhouse

 Telegram (Ru): https://telegram.me/clickhouse_ru

 Telegram (En): https://telegram.me/clickhouse_en

 Github: https://github.com/yandex/ClickHouse

 Twitter: https://twitter.com/ClickHouseDB

 Maillist: clickhouse-feedback@yandex-team.com