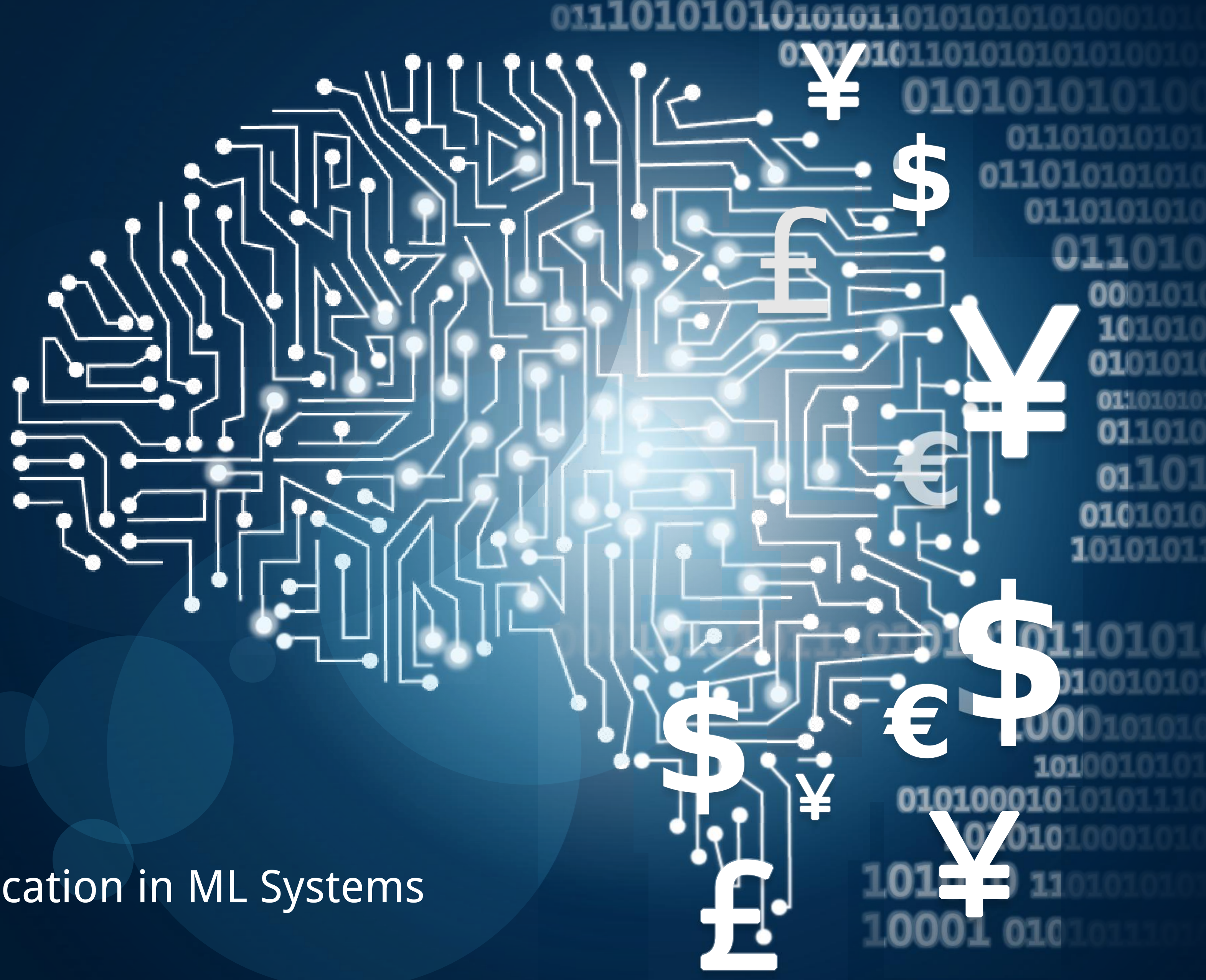


UDF in ClickHouse

Concept, Develpoment, and Application in ML Systems

About CraiditX

氪信科技
CRAIDITX

CraiditX 氪信, a finance AI startup since 2015

The Main Business

- AI-based risk control
- AI-based marketing
- AI-based customer service
- …

Our Partners and Customers

以人工智能技术实现产业智能化

About Me

Chenzhang HU 胡宸章

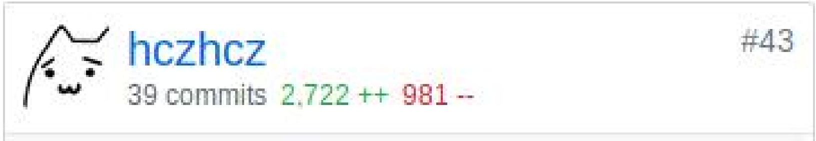
R&D Engineer at CraiditX

Focusing on AI systems and algorithms

Active GitHub User

- https://github.com/hczhcz
- Interested in computer system and language stuff
- 8 organizations, 90+ repos, 600+ followers

ClickHouse Contributor



OLAP in ML Systems

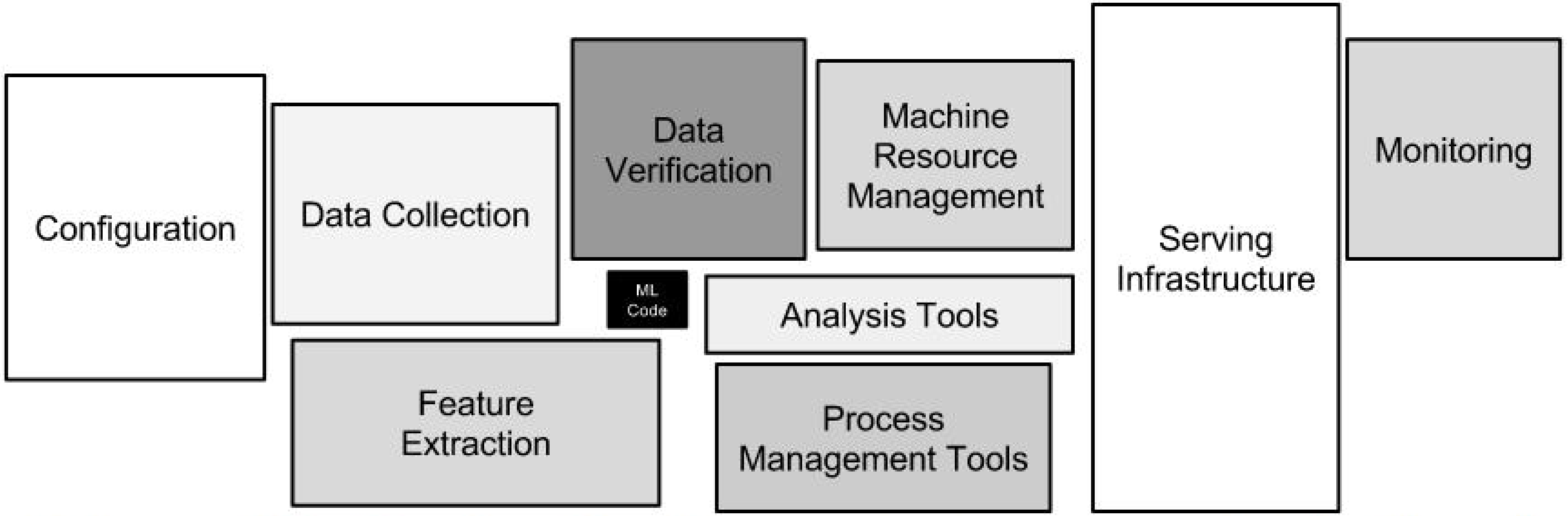


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Intensive Tasks in a ML System

- Pre-analyzing the data
- Extracting features
- Constructing relationship graphs
- Generating reports
- …

Intensive Tasks in a ML System

- Pre-analyzing the data

  = Finding the useful part of data + Summerizing data
- Extracting features

  = Joining data + Grouping data + Doing math or pattern matching
- Constructing relationship graphs

  = Identifying connections + Grouping data into source-destination pairs
- Generating reports

  = Joining data + Summerizing data
- ...

The data processing scenario is very similar to OLAP

A Database is not Just a “Database”

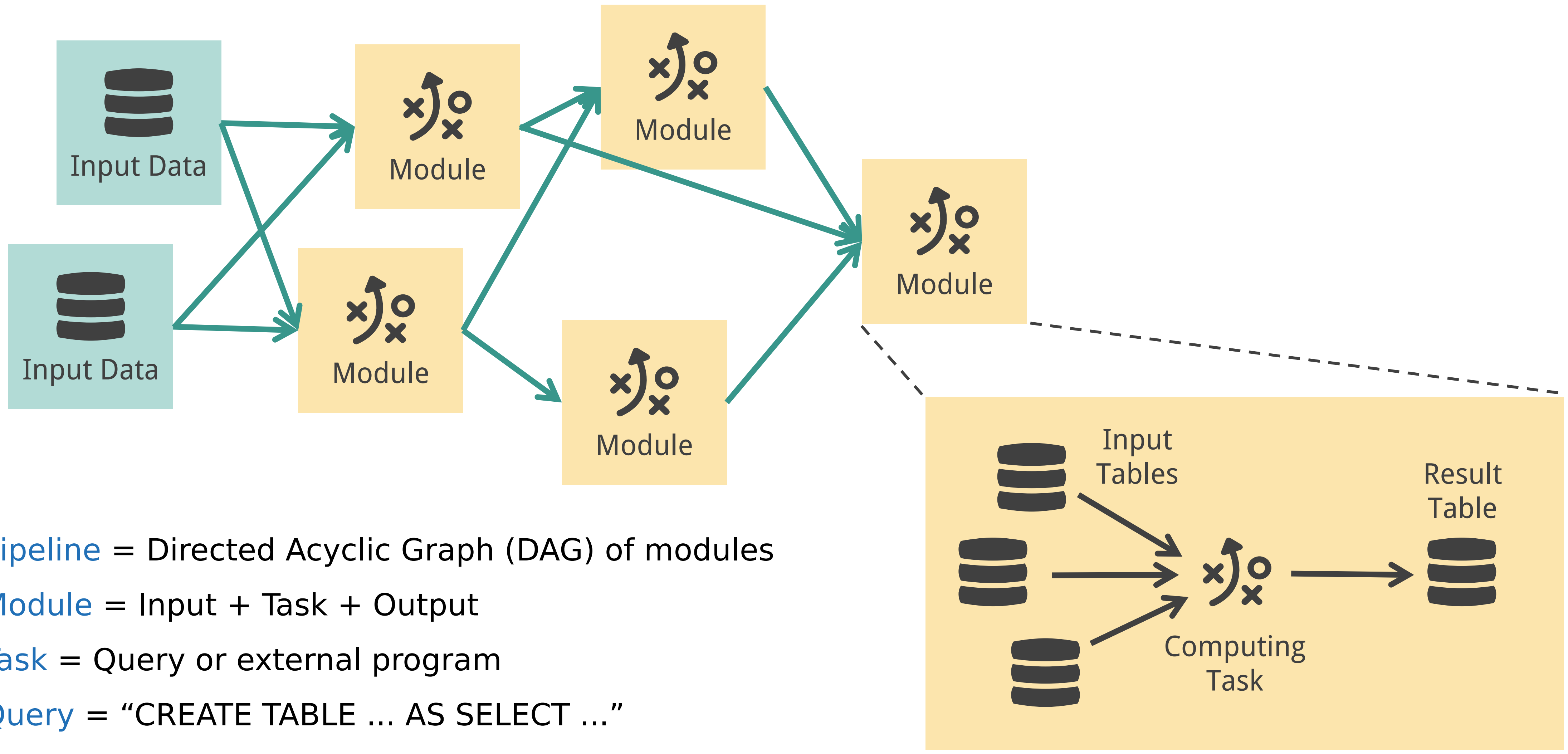
What an English Dictionary Tells You

- database /ˈdeɪtəˌbeɪs/

  A collection of data stored in a computer that can easily be used and added to

What We Actually Do

A Database System and A ML Pipeline



Pipeline = Directed Acyclic Graph (DAG) of modules

Module = Input + Task + Output

Task = Query or external program

Query = “CREATE TABLE … AS SELECT …”

Why ClickHouse

Limited hardware resources & time → efficiency matters

Performance

- Each node is able to handle billions of rows
- Most queries can be done in 0.1s-10min

Ease of Use

- SQL
- Portable binary deployment with few dependencies

Customization

- The straight-forward code structure and the well-designed API
- We maintains a custom build

The UDF Magic

When the “Standard” SQL is not Enough

Functionality Limitations

- Scanning data back and forth

    Example: Recognizing a behavioral pattern in the time series data

- Or even... randomly

    Example: Finding a shortest path in the graph

- Iterating

    Example: Training a regression model

- Handling domain-specific data

    Example: Computing the edit distance between two strings

- ...

When the “Standard” SQL is not Enough

氪信科技
CRAIDITX

Performance Concerns

- A “dull” SQL solution can be 10x-1000x slower than a native C++ program

  Example: Multiple self-joining on time series

Ease of Use and Maintainability

```
SELECT skewPop(x) FROM data
```

```
SELECT centralMoment(3)(x) / pow(stddevPop(x),
3) FROM data
```

```
SELECT (sum(pow(x, 3)) / count() - 3 *
sum(pow(x, 2)) * sum(x) / pow(count(), 2) + 2 *
pow(sum(x), 3) / pow(count(), 3)) /
pow(stddevPop(x), 3) FROM data
```

A UDF Can Do the Trick

氪信科技 CRAIDITX

User-Defined Functions (UDF)

- A function provided by the user

UDF in ClickHouse

- Scalar functions
- Aggregate functions & combinators
- Table functions & storage engines

Usage Examples in Our ML Systems

Data Preprocessing
Filling invalid date strings in a time series

Feature Engineering
Calculating average values within a window

Connection Recognition
Finding persons with similar street addresses

Zoo of Our UDF

Windowed Aggregate Functions

```
SELECT windowRefer(30)(date, value) FROM data
SELECT windowReferEx(30, 'quantile(0.2)')(date, value) FROM data
```

- Computing the aggregate results of the rows within each time window
- When a row enters, it will be “added” to the aggregation state
  - Then, the aggregate function yields a result field
- When a row leaves, it will be “removed” from the aggregation state
  - It requires some special algorithms and tricks
  - Most of the aggregate functions and combinators are re-implemented
- Typical usage: Feature computing

A further idea: -Window combinator

Funnel Automata Functions

氪信科技 CRAIDITX

```
SELECT yFunnel(6, '
    rule(r1)(require(get(1) = ''index.html'')),
    rule(r2, r1)(require(get(1) = ''product_info.html''),
        product_id(get(2)), return(''viewed'')),
    rule(r3, r2)(require(get(1) = ''product_buy.html''),
        verify(product_id = get(2)), return(''purchased''))
')(timestamp, page, product_id)
FROM pageview
```

- Matching behavior sequences within time window
- Inspired by Analysys OLAP Challenge 2018 (Funnel Analysis)
- Featuring built-in automata description DSL
    - It is implemented using the parser facilities in ClickHouse

Array Functions

```
SELECT arraySplit(x -> x >= 10, [11, 4, 5, 14]) = [[11, 4, 5], [14]]
SELECT arrayFill(x -> x > 0, [1, 2, 0, 0, 3, 0]) = [1, 2, 2, 2, 3, 3]
```

- Handling time series data
  - With arrays, the data block boundary will not affect the result
- Based on the common implementation of high-order array functions
  - FunctionArrayMapped.h
- Used in Analysys OLAP Challenge 2019 (Session Analysis)
- Pull request #7294 and #7380

Integration of Third-party Libraries

FuzzyWuzzy

- Levenshtein distance (edit distance)
  - By character
  - By token
- Popular among ML algorithm engineers

Simdjson

- An extremely fast JSON parser based on AVX2 Instruction Set
- Structured data extraction (JSONExtract)
  - We can pass the type as a parameter just like in CAST function
- Difficulties in cross-platform compatibility
- Pull request #4686 and #5124

Miscellaneous

Statistics

- Simple linear regression
- Skewness and kurtosis
- Weight-based entropy
- Pull request #4668 and #5200

Aggregate Function Combinators

- -OrDefault and -OrNull
    - When an aggregate function has nothing to aggregate
- -Resample
- Pull request #5590 and #7331

UDF Development Explained

General Steps of UDF Development

氪信科技 CRAIDITX

Design the Interface

- Meta information

  Example: Will the return value change over time?

- Arguments and the return value

Create the Function Body

- Scalar functions: Handling data blocks
- Aggregate functions: Maintaining internal states, adding, merging, and reducing

Put Things Together

- Registering your function
- Build and run!

Type Checking and Inference

What should we consider? Let's see...

```
arraySum([1, 2, 3])
```

- Number of argument = 1
- Data type of the only argument = Array(X)
  - X should be UInt, Int, or Float
- Data type of the return value = Extended(X)
  - We prefer the largest native numerics, i.e. UInt64, Int64, or Float64
- Data types vs. column types
- Furthermore?
  - The real arraySum also supports lambda

Scalar Function Implementation

Then...

```
arraySum([1, 2, 3])
```

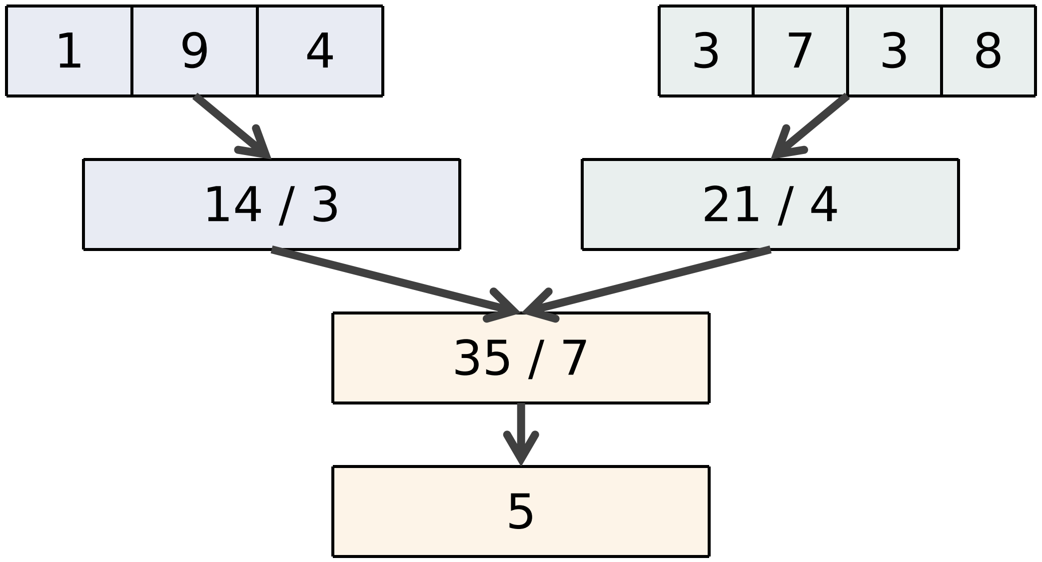
- Data are handled per block
- Column type of the argument = ColumnArray(ColumnVector(X), ColumnOffset)
    - ColumnConst can be handled explicitly or automatically

```
value        offset      arraySum
[1, 2, 3]    3           6           <- sum of value[0..2]
[9, 10]      5           19          <- sum of value[3..4]
[4, 5, 6]    8           15          <- sum of value[5..7]
```

Aggregate Function Implementation

avg(x)

- Data are handled per group and row
- Internal state = (count(), sum(x))
  - To add one row
    - sum(x) += x[i]
    - count() += 1
  - To calculate the result
    - avg(x) = sum(x) / count()
  - Also, we need to implement serialization, deserialization, and merging
- Some aggregate functions require more advanced algorithms
  - uniq(x), median(x), ...
  - Most of them depend on sampling tricks and/or special data structures



Going Further

Inline C++ in SQL

```
SELECT udsf('
    std::string udsf(std::string s)
    {
        return "hello, " + s;
    }
', 'world')
```

- Compiled and linked dynamically
  - It uses the embedded compiler facilities in ClickHouse
- Type inference driven by C++ Template Metaprogramming
- Also supported: Block-based scalar functions, aggregate functions

Okay for proof-of-concept usages, but ...

Zora: High-performance Algorithm Implementation Framework

氪信科技
CRAIDITX

Main Concepts

Column-oriented & Memory Densed

High memory efficency and avoiding unnecessary IO

Smooth integration with ClickHouse, NumPy, Pandas, ...

Minimalism Fundamental Algorithm Components

Carefully chosen algorithm targeted at supporting the ML pipeline

Featured: Data structures, graph algorithms, statistical operators, ...

Write Once, Available Everywhere

Native C++ Implementation with no third-party dependency

Auto-generated interface to Python3 and ClickHouse (UDF)

```cpp
template <typename pos_t, typename value_t,
void personalized_page_rank(
    pos_t &member_n,
    pos_t *&member_node,
    value_t *&member_rank,

    const pos_t *node_edge_begin,
    const pos_t *node_edge_end,
    const value_t *node_weight,
    const pos_t *edge_node_to,
```

```python
n, node, rank = zora.personalized_page_rank(
    df_node.edge_begin,
    df_node.edge_end,
    df_node.weight,

    df_edge.node_to,
    df_edge.weight,
```

```sql
SELECT
    personalizedPageRank(node_edge_begin, no
FROM (
    SELECT
        groupArray(edge_begin) as node_edge_
        groupArray(edge_end) as node_edge_er
        groupArray(weight) as node_weight
```

Questions / Comments