

# Ohua-Powered, Semi-Transparent UDF's in the Noria Database

By Justus Adam

Supervisor: Sebastian Ertel, Dirk Habich, Malte Schwarzkopf and Jerónimo Castrillón-Mazo

# State is useful!

Aggregation

Windowing

Efficiency

Fundamental

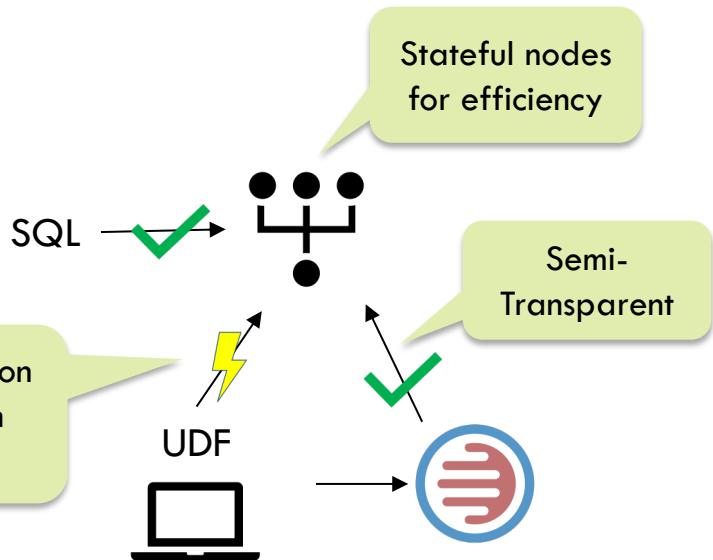
Intuitive  
Code

Counting

Deduplication

Caching

- No Optimisation
- No Parallelism
- Support?



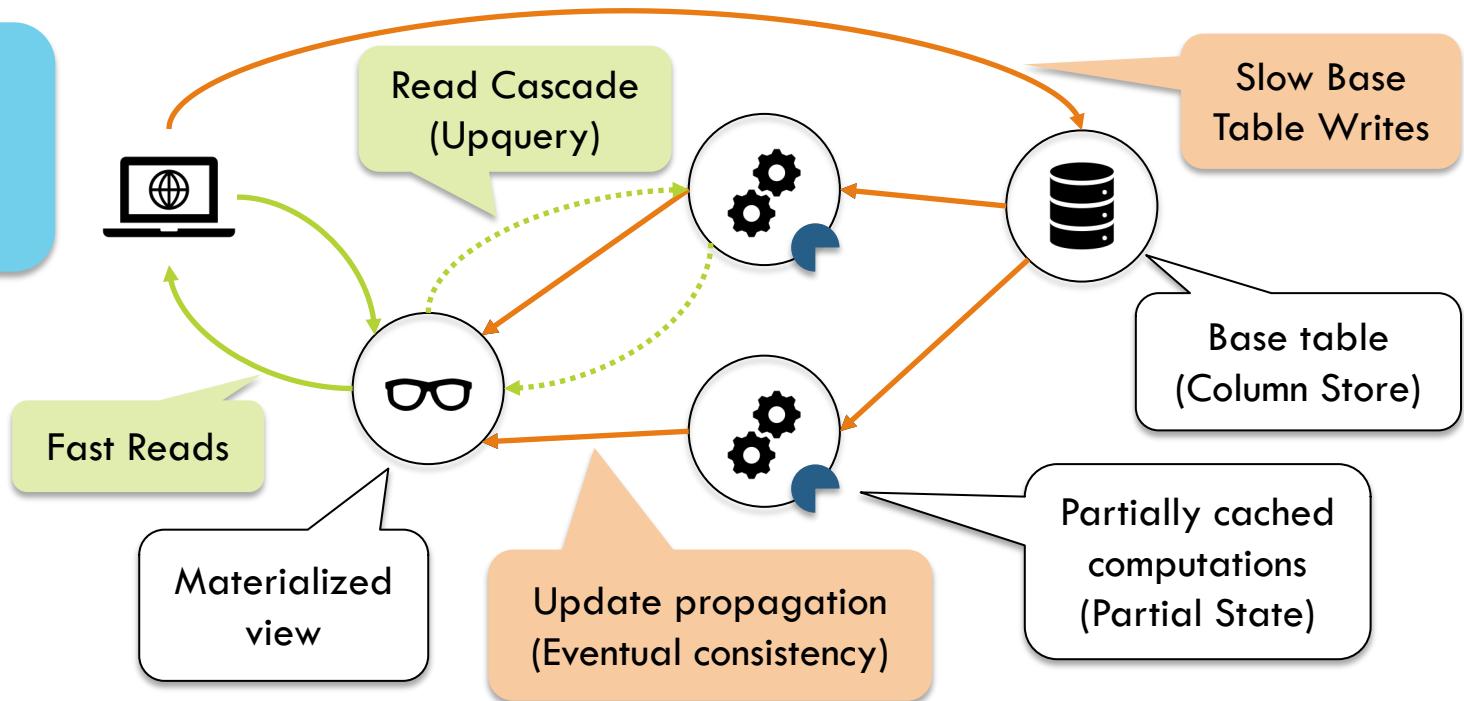
# Example UDF – Clickstream Analysis

```
fn click_ana(  
    start_cat: Category,  
    end_cat: Category,  
    clicks: Stream<(UID, Category, Time)>  
) -> f64 {  
    let click_streams = group_by::<0>(clicks);  
    click_streams.map(|stream| {  
        let state = iseq::Seq::new();  
        for (_, cat, time) in stream {  
            if cat == start_cat {  
                state.start(time)  
            } else if cat == end_cat {  
                state.end(time)  
            } else {  
                state.record(time)  
            }  
        }  
        state  
            .complete_intervals()  
            .map(Interval::len)  
            .average()  
    })  
}
```

- ❑ From SQL/MapReduce<sup>1</sup> paper
  - ❑ Complicated in SQL
  - ❑ Inefficient in SQL
- ❑ Non trivial but simple
- ❑ Business query
- ❑ State is paramount
- ❑ Contains SQL element

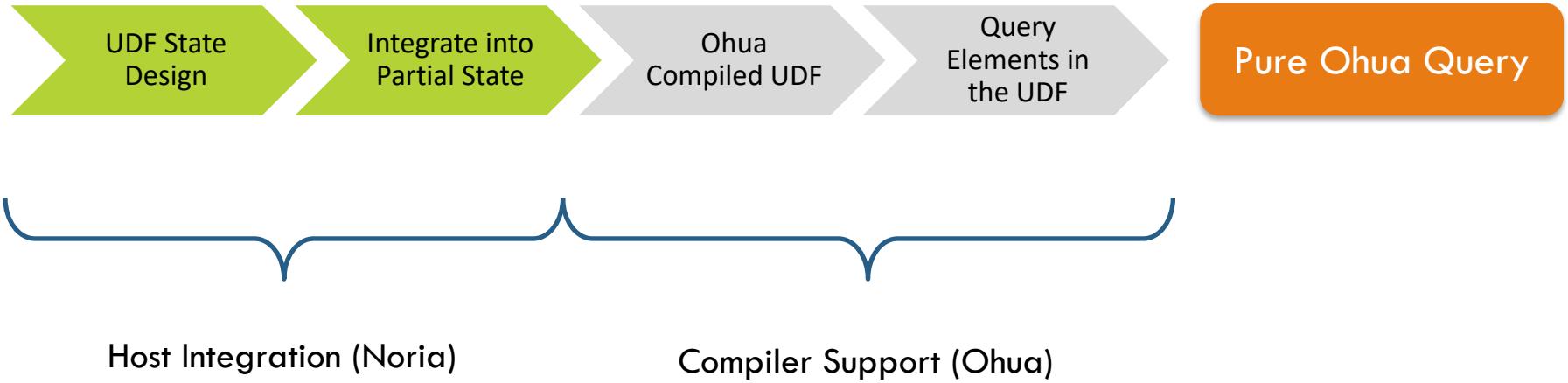
1. Eric Friedman, Peter Pawlowski, and John Cieslewicz. 2009. SQL/MapReduce: a practical approach to self-describing, polymorphic, and parallelizable user-defined functions. *Proc. VLDB Endow.* 2, 2 (August 2009), 1402-1413.

- Multicore
- Distributed
- No UDF Support



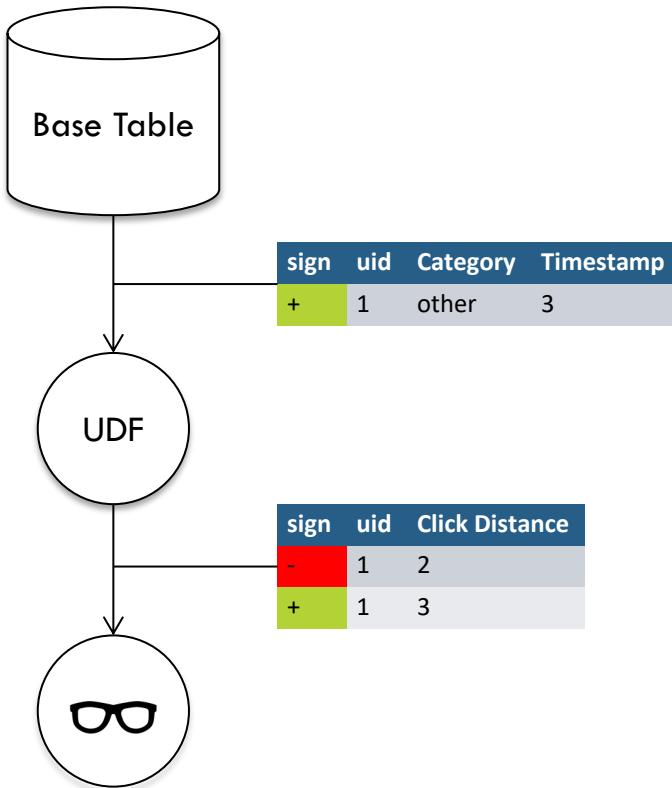
1. Jon Gjengset et al. 2018. Noria: dynamic, partially-stateful data-flow for high-performance web applications. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation* (OSDI'18). USENIX Association, Berkeley, CA, USA, 213–231.

# Roadmap



# Noria Execution Model

uid	Category	Timestamp
1	start	1
1	other	5
1	end	10
1	other	3



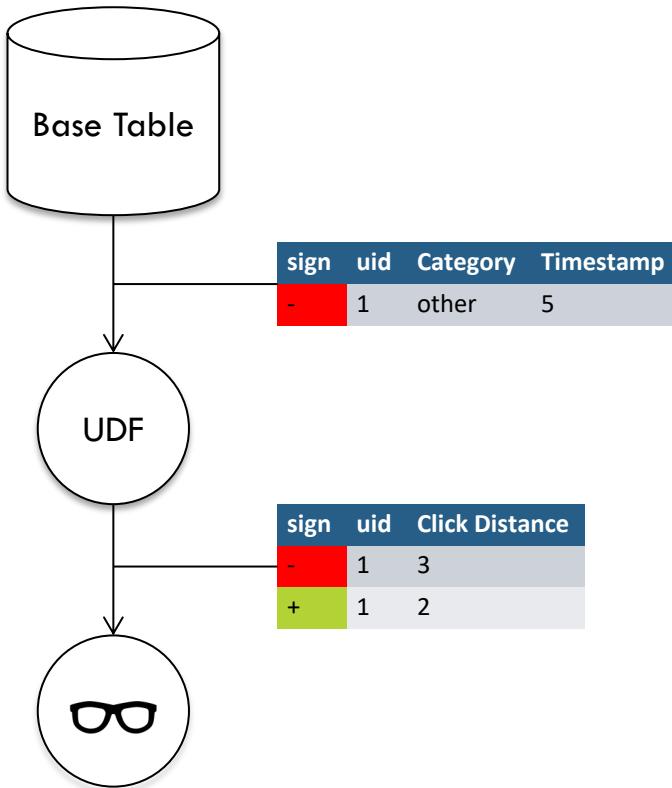
Update Path (Insert)

INSERT (1, other, 3)  
INTO 'Base Table';

uid	Click Distance
1	3

# Noria Execution Model

uid	Category	Timestamp
1	start	1
1	other	5
1	end	10
1	other	3



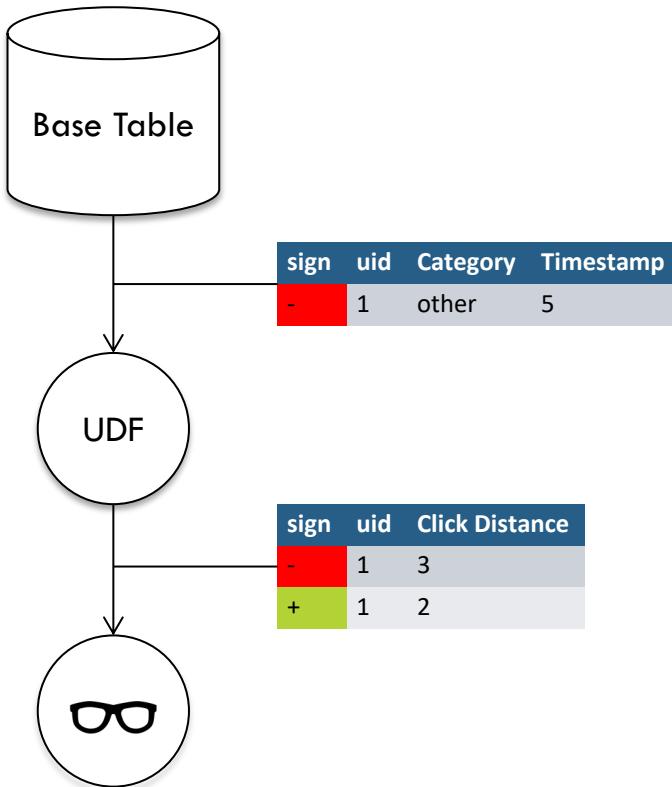
Update Path (Delete)

DELETE (1, other, 3)  
FROM ‘Base Table’;

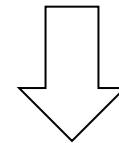
uid	Click Distance
1	2

# Noria Execution Model

uid	Category	Timestamp
1	start	1
1	other	5
1	end	10
1	other	3



- On-line inserts
- On-line deletes
- Order is random

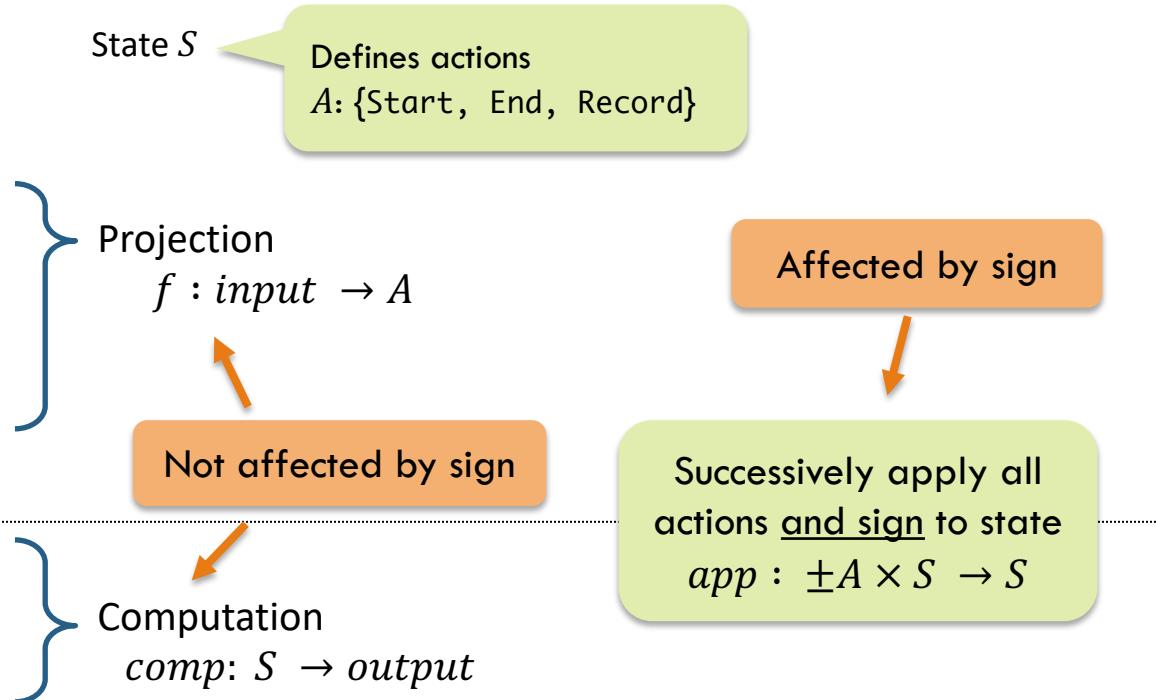


- Commutative
- Incremental
- Reversible Operations

# UDF State Design

```
let state = iseq::Seq::new();
for (_, cat, time) in stream {
    if cat == start_cat {
        state.start(time)
    } else if cat == end_cat {
        state.end(time)
    } else {
        state.record(time)
    }
}
```

```
state
    .complete_intervals()
    .map(Interval::len)
    .average()
```



$UDF: [\pm \text{input}] \rightarrow \text{output}$

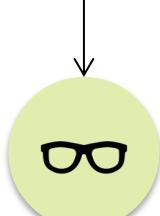
# Interval Sequence as State



uid	Category	Timestamp
1	start	1
1	other	5
1	end	10
1	other	3



`[[5,3,10].length()].average() == 3`



uid	Click Distance
1	3

$$s: [ [l_0, u_0), [l_1, u_1), [l_2, u_2)] ]$$

$t \in T$  such that

- $t \geq \begin{cases} l_1 & \text{if } l_1 \text{ exists} \\ u_0 & \text{otherwise} \end{cases}$
- $t < \begin{cases} u_1 & \text{if } u_1 \text{ exists} \\ l_2 & \text{otherwise} \end{cases}$

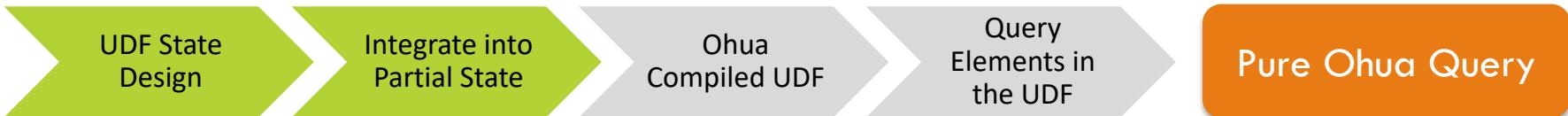
Invariants

- $l_1$  or  $u_0$  must exist
- $u_1$  or  $l_2$  must exist

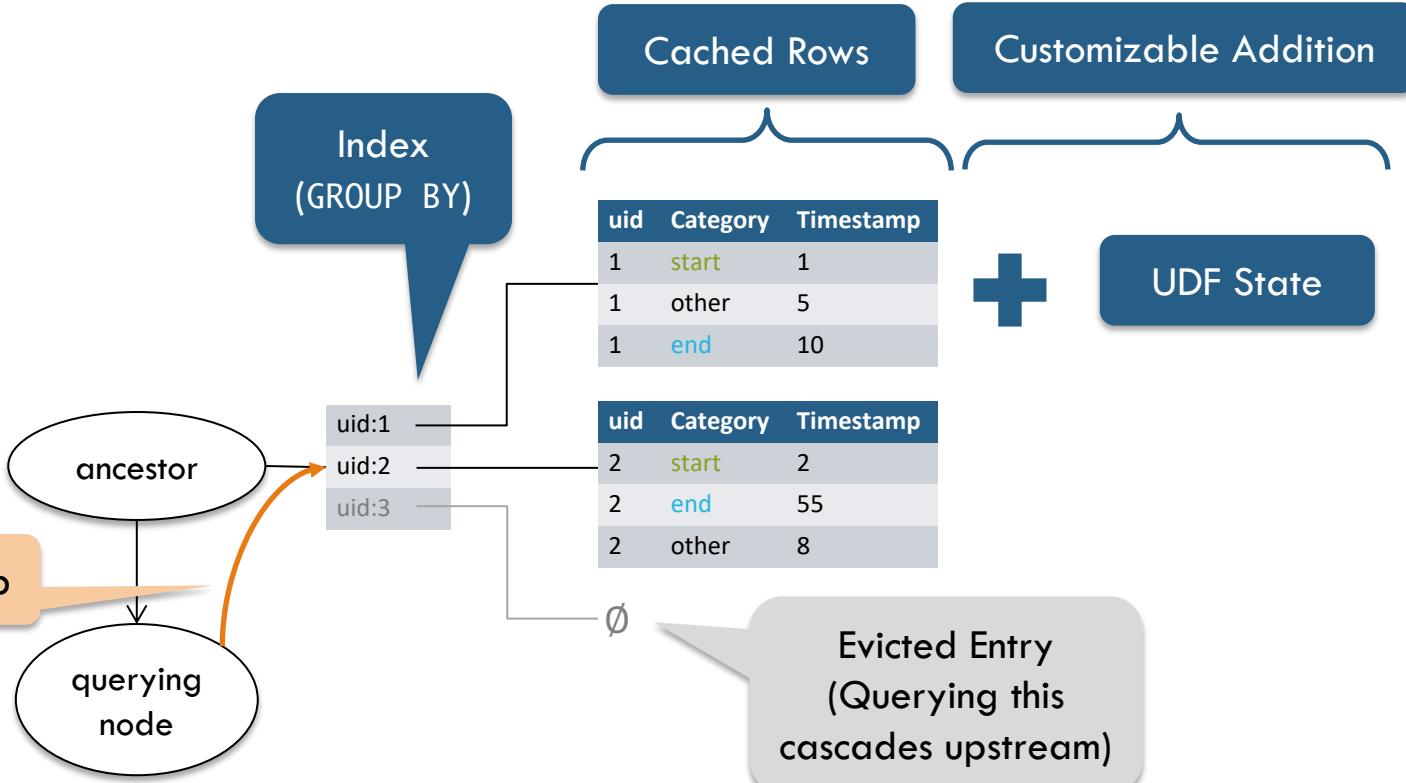
→ Merge intervals to maintain

# Conclusions

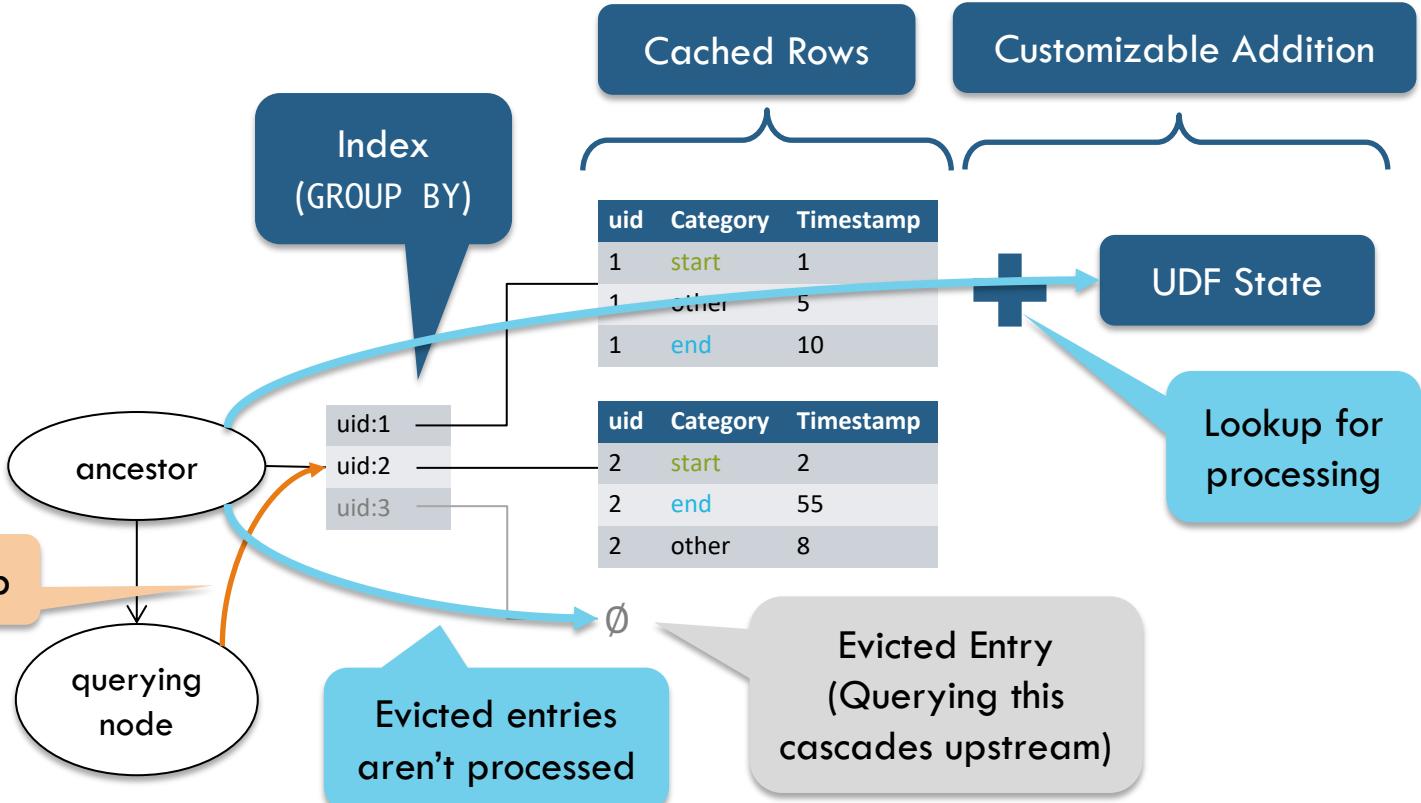
- Must be
- Reversible
  - Commutative



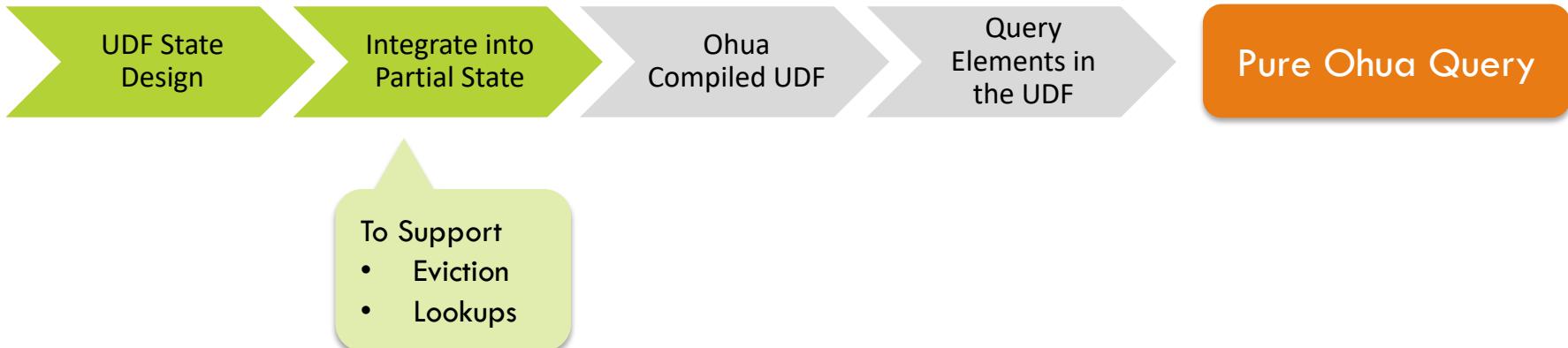
# Partial State integration



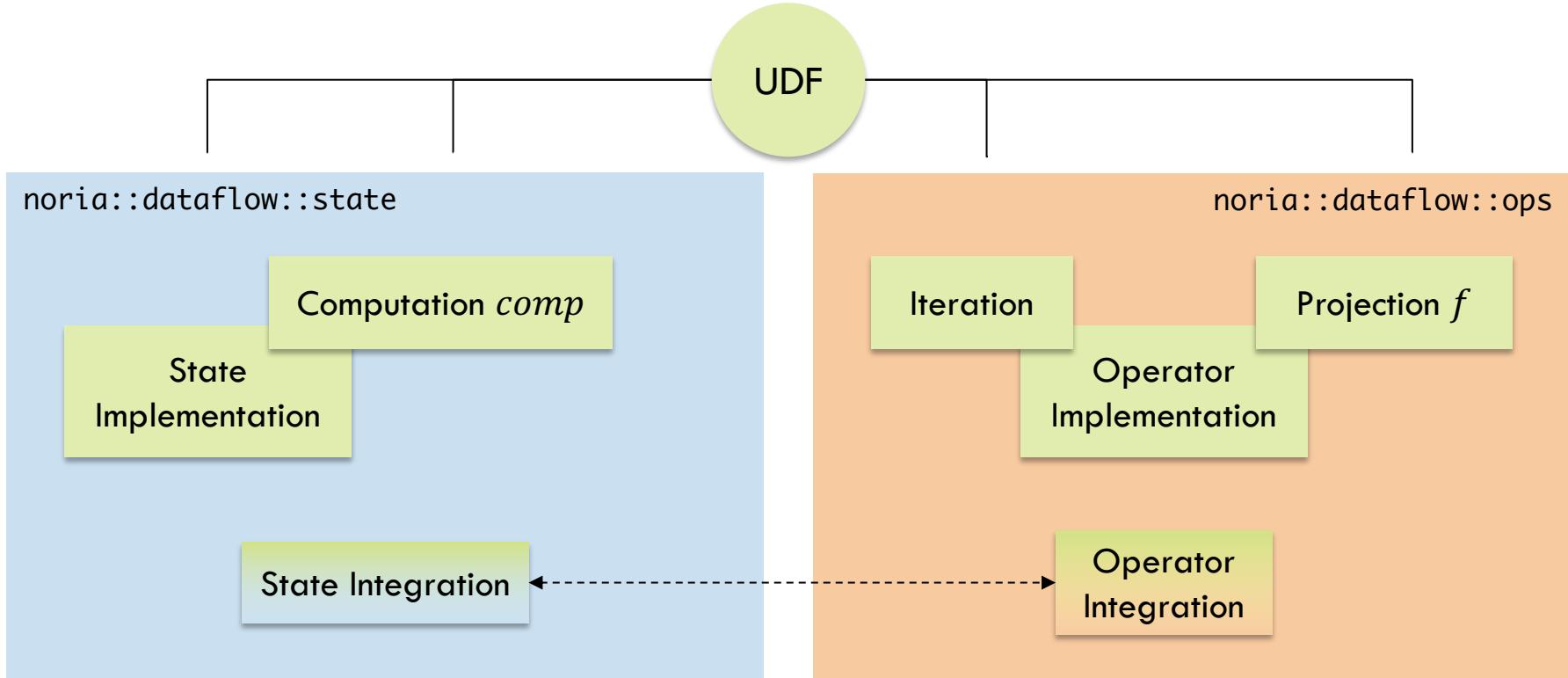
# Partial State integration



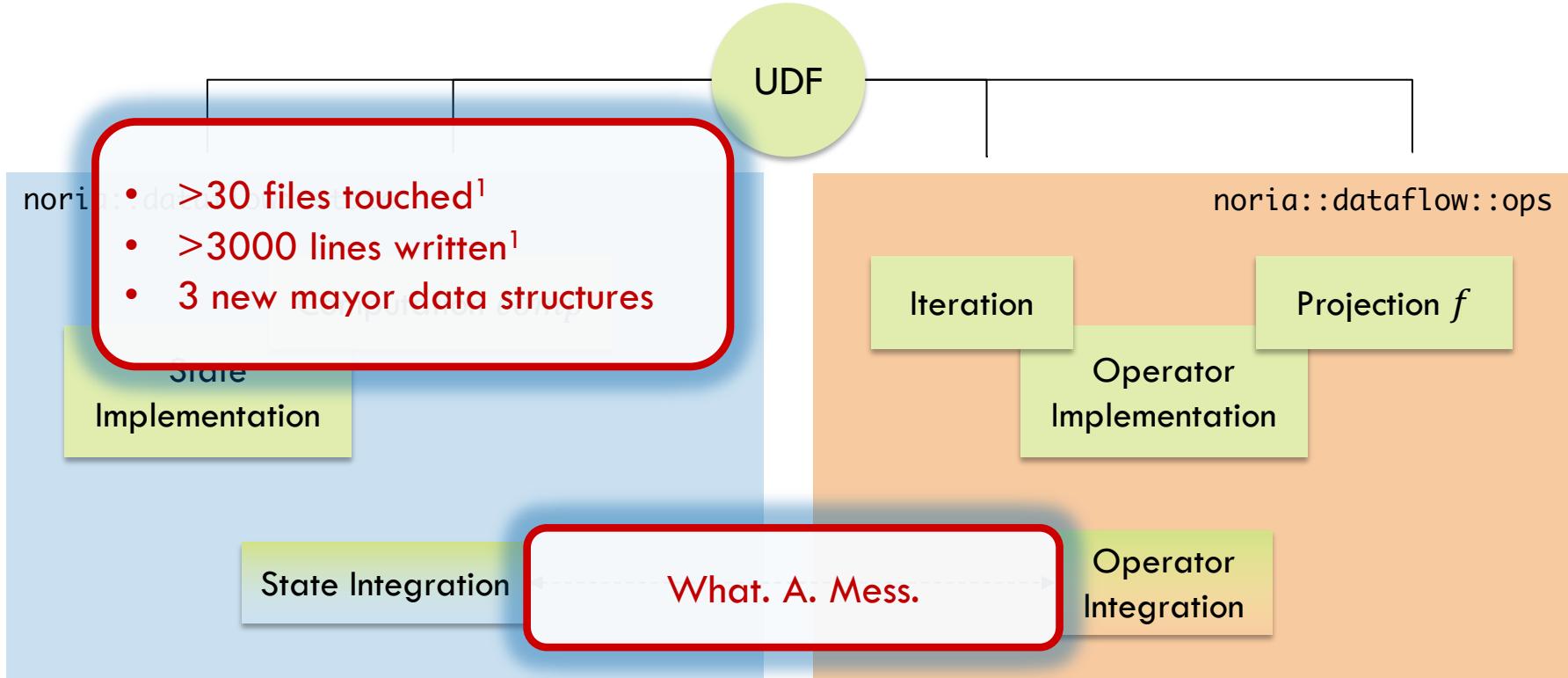
# Conclusions



# Manual Implementation

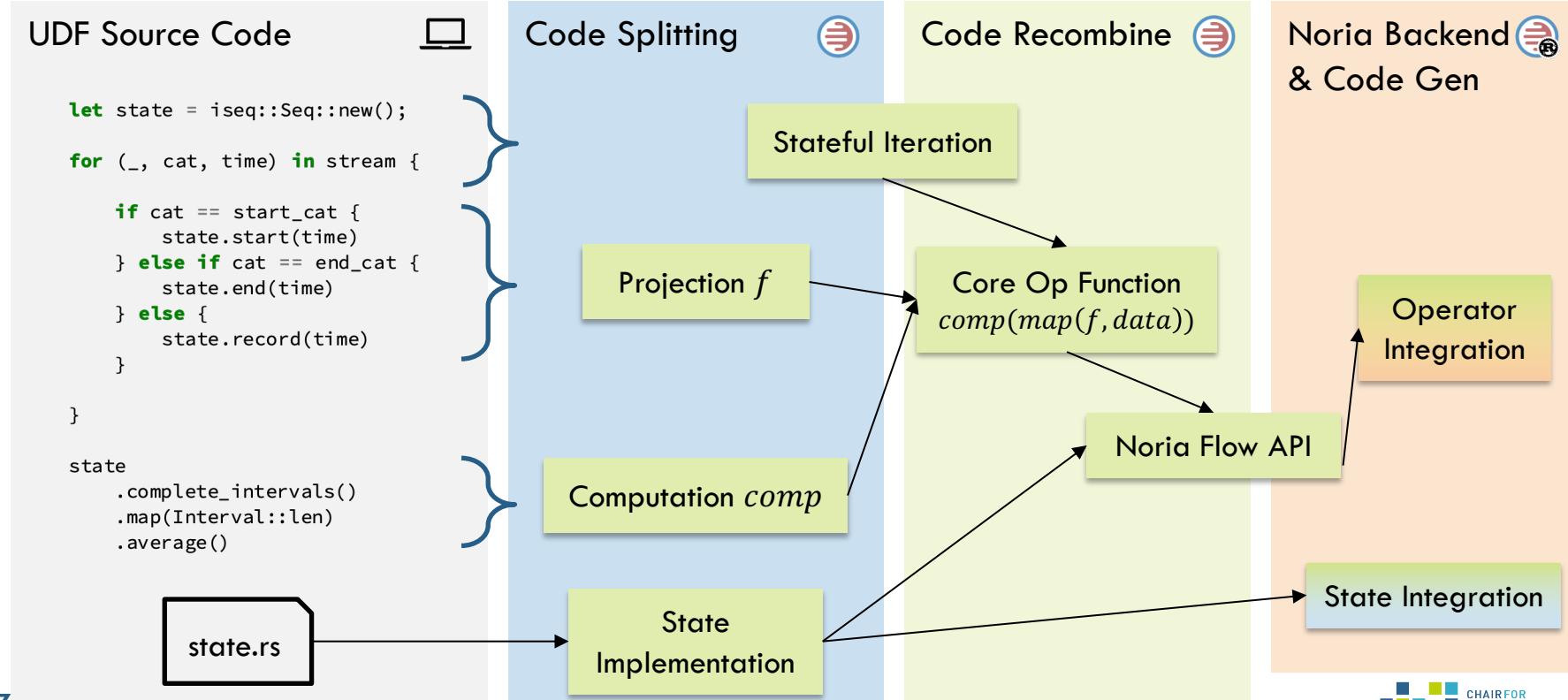


# Manual Implementation

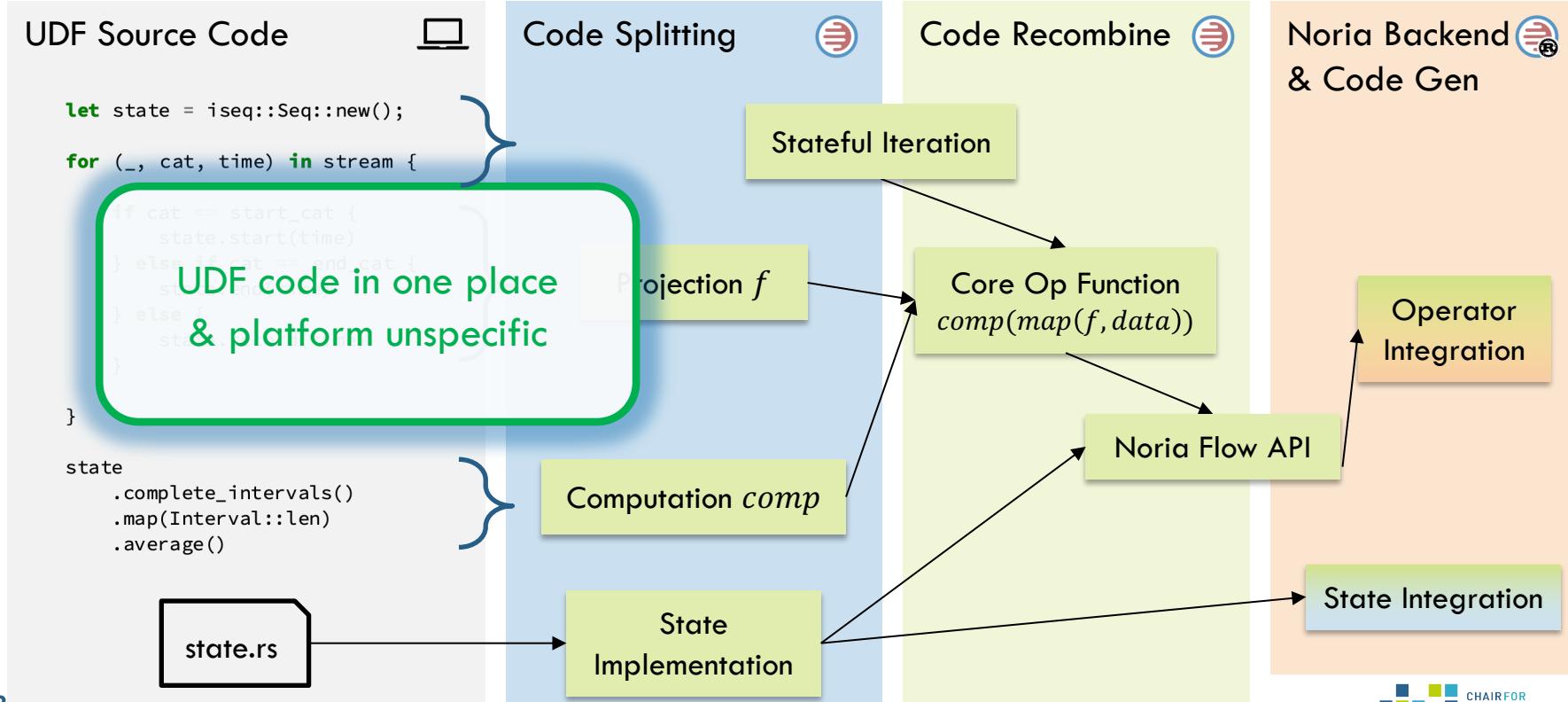


1. For whole implementation including intermediate prototypes and test code. Approximately 50% used exclusively for UDF.

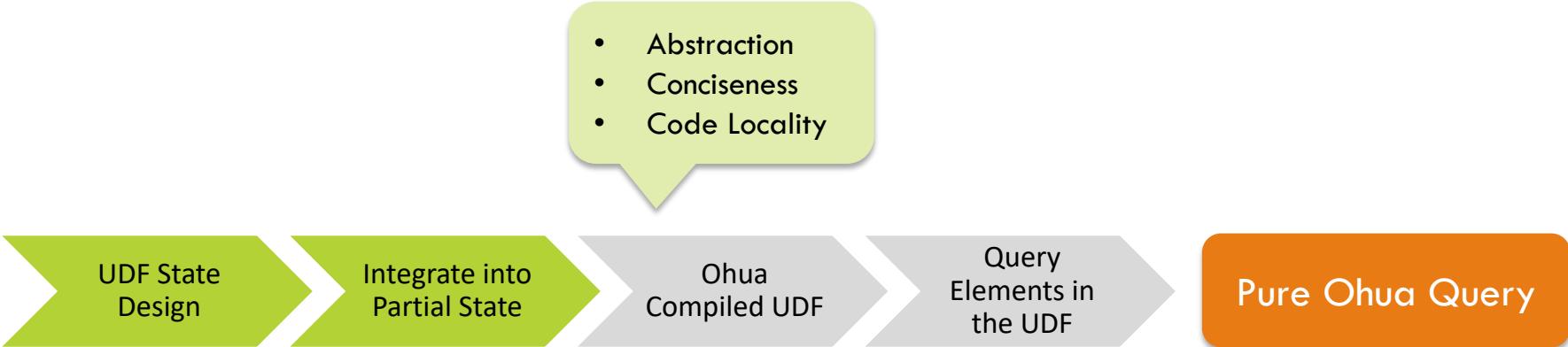
# Operator Compilation



# Operator Compilation



# Conclusions



# UDF Compilation

## UDF Source Code



```
fn click_ana(  
    start_cat: Category,  
    end_cat: Category,  
    clicks: Stream<(UID, Category, Time)>  
) -> f64 {  
  
    let click_streams = group_by::<0>(clicks);  
  
    click_streams.map(|stream| {  
  
        ...  
  
        Operator Code  
    })  
}
```

## Code Splitting



Signature

Grouping

Operator

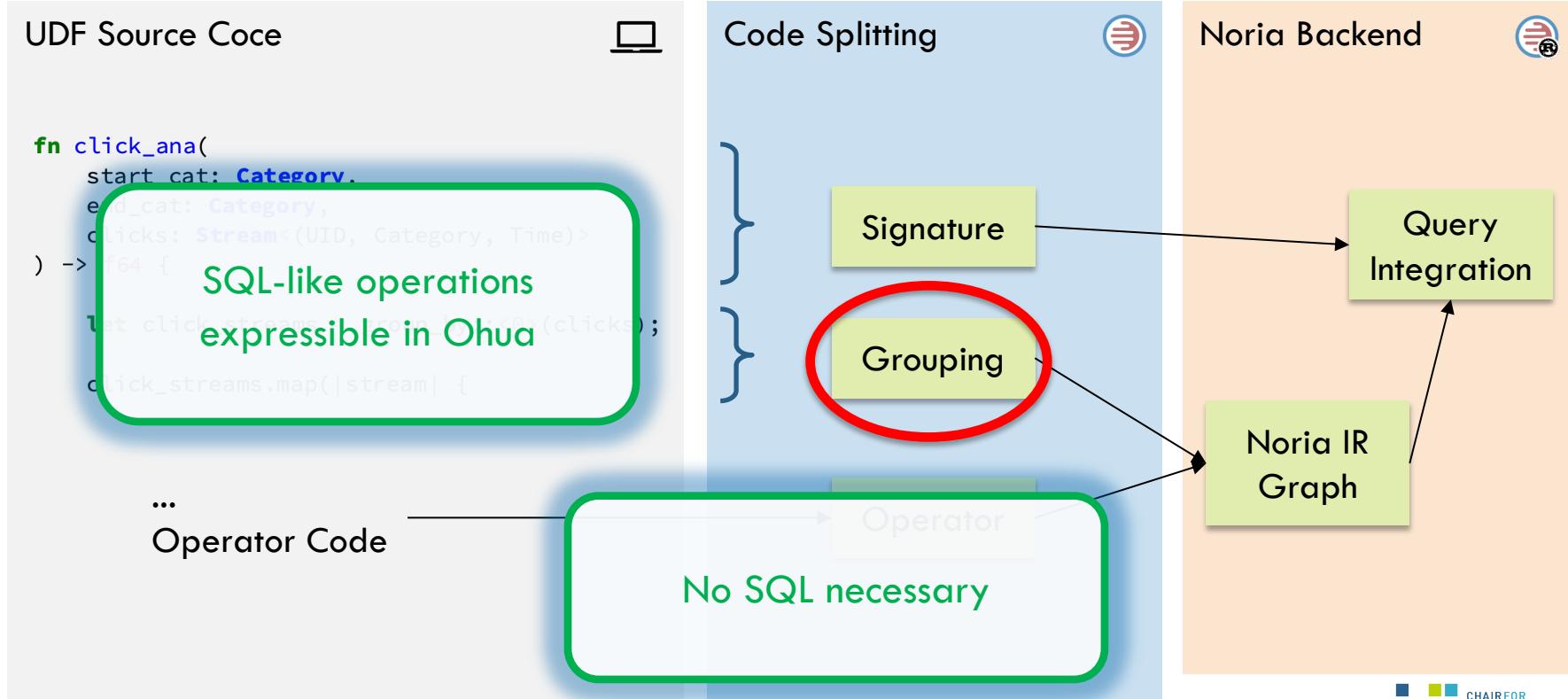
## Noria Backend



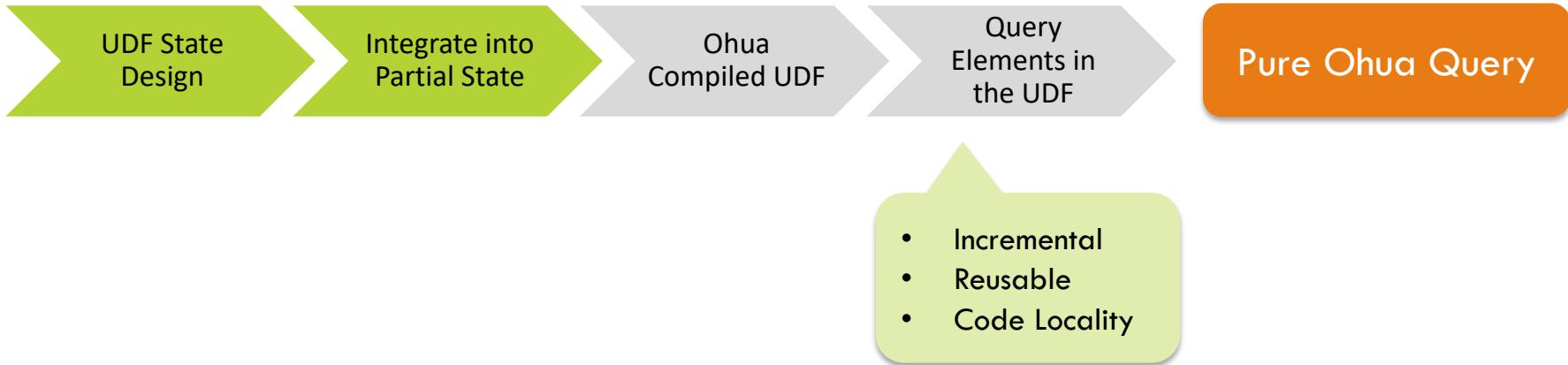
Query  
Integration

Noria IR  
Graph

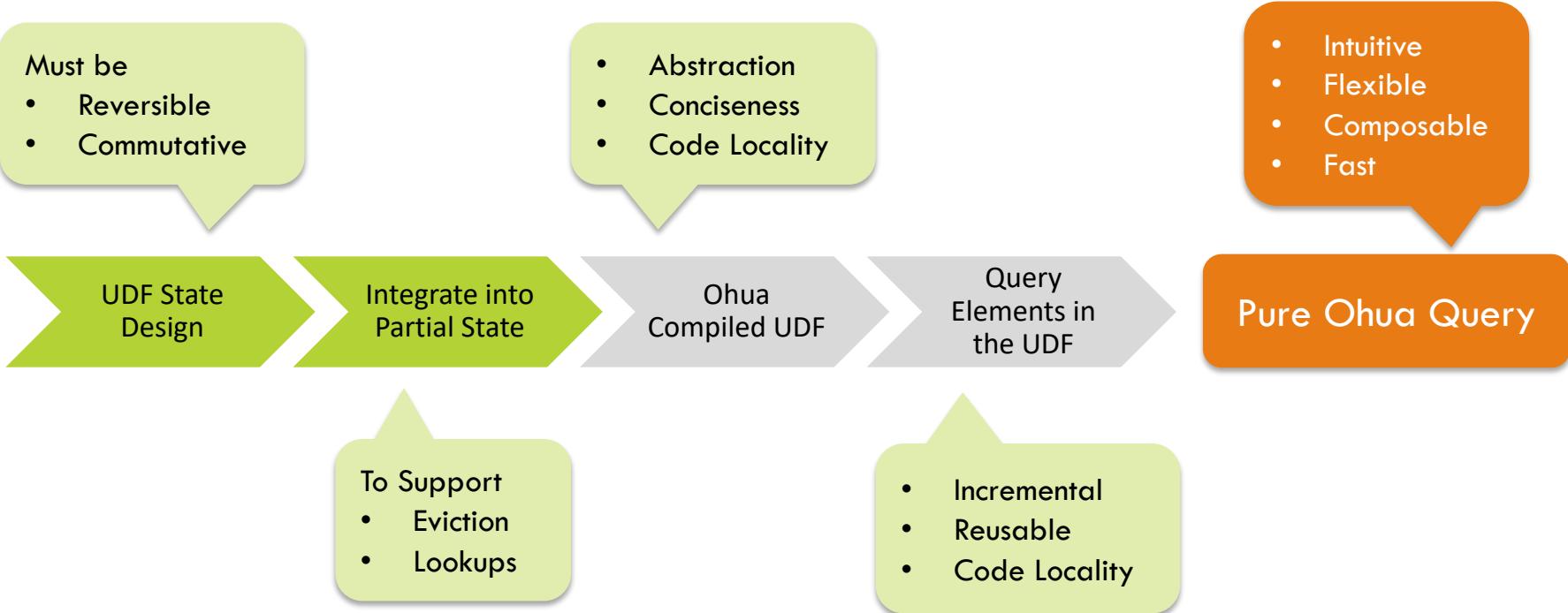
# UDF Compilation



# Conclusions



# Conclusions



# Conclusions

