CHAIR FOR COMPILER CONSTRUCTION

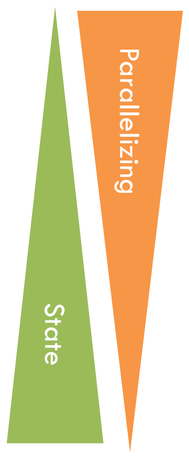
CENTER FOR ADVANCING ELECTRONICS DRESDEN

Towards Scalable UDTFs in Noria[1]

Contact: Justus Adam, Justus.Adam@tu-dresden.de

Research Challenges: User Defined Functions for distributed, incremental materialized views

- **UDFs** are a powerful extension point for databases
  - Third-party libraries, serialization, conversion
- Imperative source language with shared mutable state
- Table function and aggregation interfaces, are inherently stateful
- The more state is used, the harder it is to parallelize or shard



Single-tuple UDF
single input, single output

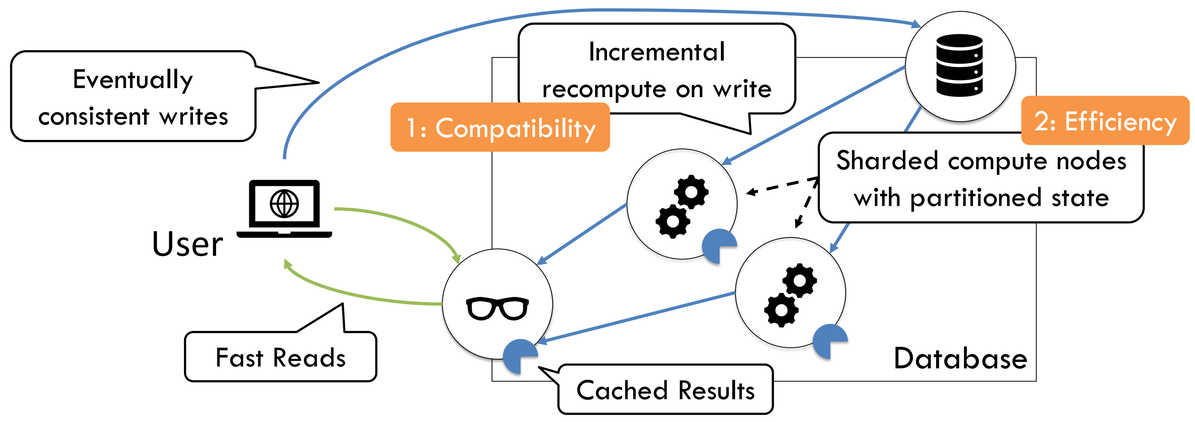
Aggregation, Set-returning function
multiple inputs **or** outputs

Table Function
multiple inputs **and** outputs, only parallelizable with additional knowledge of the function itself

- **Materialized views** offer fast read trough aggressive caching
- **Incremental maintenance** makes the slow writes more efficient

Challenge 1: Compatibility: Support incremental computations for UDF

Challenge 2: Efficiency: Support sharding (parallelizing and distribution) the UDF or risk being a bottleneck to scaling



1. Jon Gjengset, Malte Schwarzkopf, Jonathan Behrens, Lara Timbó Araújo, Martin Ek, Eddie Kohler, M. Frans Kaashoek, and Robert Morris. 2018. **Noria: dynamic, partially-stateful data-flow for high- performance web applications**. *OSDI 18*.

Compilation Target and Strategy

- The compilation target, a Noria query, is a graph (network) of stateful operators
  - Operator state is private, not shared
  - Communication via message passing

Use parallelizing compiler (Ohua[2,3]) to split UDF program into

- Single "outer" program, *without shared state*, suitable for transformation into the **query graph**
- Multiple "inner" programs, *using shared state* internally, suitable for forming the core of **stateful operators**

```
fn click_ana(clicks: RowStream<i32, i32, i64>)
            -> GroupedRows<i32, i32> {
    for (uid, group_stream) in group_by(0, clicks) {
        let sequences = IntervalSequence::new();
        for (_, cat, time)
            in sort_on(2, group_stream) {
            if *cat == 1 {
                sequences.open(*time)
            } else if *cat == 2 {
                sequences.close(*time)
            } else {
                sequences.insert(*time)
            }
        };
        (uid,
         sequences.iter()
                  .filter(Interval::is_bounded)
                  .map(Interval::len)
                  .average())
    }
}
```

outer program to graph

Find state mutations and recursively find dependencies

bundle into stateful operator

2. Sebastian Ertel, "Towards Implicit Parallel Programming for Systems." Dissertation, 2019
3. Sebastian Ertel, Justus Adam, Norman Rink, Andrés Goens, Jeronimo Castrillon, "STCLang: State Thread Composition as a Foundation for Monadic Dataflow Parallelism." Haskell'19.

Incremental Operator State

1: Compatibility

- We require custom operator state mutations to be reversible
- This allows stateful operators to be made incremental automatically

If a previously processed value is deleted, rerun operator computation but **revert** modifications instead of applying them.

Data Parallelism

2: Efficiency

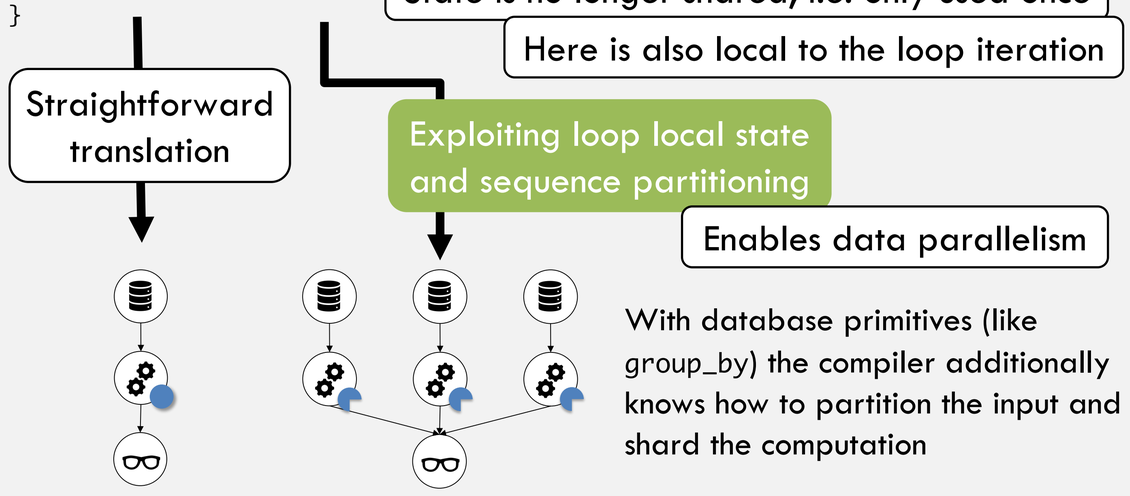
Outer program after splitting

```
fn click_ana(clicks: RowStream<i32, i32, i64>)
            -> GroupedRows<i32, i32> {
    for (uid, group_stream) in group_by(0, clicks) {
        let op0 = Op0::init();
        let op0_res = op0.run(sort_on(2, group_stream));
        (uid, op0_res)
    }
}
```

Sequence index becomes state partition index

State is no longer shared, i.e. only used once

Here is also local to the loop iteration



With database primitives (like group_by) the compiler additionally knows how to partition the input and shard the computation

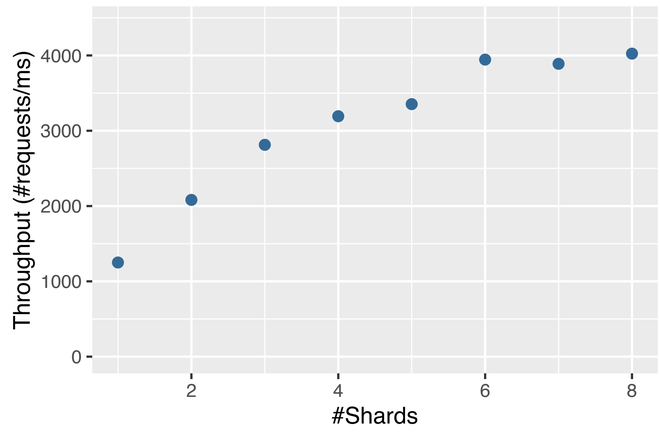
Preliminary Results

1: Compatibility Our novel technique can compile a subset of imperative, stateful Rust to incrementally maintained views in a dataflow engine

- Supports Single-tuple UDFs, aggregations, table functions and standalone queries

2: Efficiency Parallelism and sharding is implicit and effortless

- For our example query we are able to achieve near linear scaling up to 3 shards.



Diminishing returns after 3 shards are likely caused by orchestration overhead starting to dominate the small data size.

cfaed.tu-dresden.de

TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

DFG

WR WISSENSCHAFTSRAT