A Step Toward Deep Online Aggregation

Nikhil Sheoran[1,*], Supawit Chockchowwat[2,*], Arav Chheda[2], Suwen Wang[2], Riya Verma[2], Yongjoo Park[2]

[1] Databricks (work done while at [2])
[2] CreateLab @ University of Illinois at Urbana-Champaign,
[*] Equal Contribution

Today’s big data ecosystems

FLOW

kafka

STORM

STORE

ceph

hadoop

FORMAT

AVRO™

Apache orc™

Parquet

COMPUTE

APACHE Spark™

Microsoft® SQL Azure™

cassandra

amazon REDSHIFT

HIVE

Google BigQuery

VIZ

+ a b l e a u®

Periscope Data

QlikView

Various systems across different aspects of big data processing

motivation

How to enable quick processing for large volumes of data?

A Step Toward Deep Online Aggregation

NIKHIL SHEORAN*†, Databricks, USA
SUPAWIT CHOCKCHOWWAT†, University of Illinois at Urbana-Champaign, USA
ARAV CHHEDA, University of Illinois at Urbana-Champaign, USA
SUWEN WANG, University of Illinois at Urbana-Champaign, USA
RIYA VERMA, University of Illinois at Urbana-Champaign, USA
YONGJOO PARK, University of Illinois at Urbana-Champaign, USA

For exploratory data analysis, it is often desirable to know what answers you are likely to get *before* actually obtaining those answers. This can potentially be achieved by designing systems to offer the estimates of a data operation result—say op(data)—earlier in the process based on partial data processing. Those estimates continuously refine as more data is processed and finally converge to the exact answer. Unfortunately, the existing techniques—called *Online Aggregation* (OLA)—are limited to a single operation; that is, we *cannot* obtain the estimates for op(op(data)) or op(...(op(data))). If this *Deep OLA* becomes possible, data analysts will be able to explore data more interactively using complex cascade operations.

In this work, we take a step toward *Deep OLA* with *evolving data frames* (edf), a novel data model to offer OLA for nested ops—op(...(op(data)))—by representing an evolving structured data (with converging estimates) that is *closed* under set operations. That is, op(edf) produces yet another edf; thus, we can freely apply successive operations to edf and obtain an OLA output for each op. We evaluate its viability with Wake, an edf-based OLA system, by examining against state-of-the-art OLA and non-OLA systems. In our experiments on TPC-H dataset, Wake produces its first estimates 4.93× faster (median)—with 1.3× median slowdown for exact answers—compared to conventional systems. Besides its generality, Wake is also 1.92× faster (median) than existing OLA systems in producing estimates of under 1% relative errors.

124

CCS Concepts: • **Information systems** → **Database query processing**; **Online analytical processing engines**; *Relational parallel and distributed DBMSs*; **Uncertainty**; **Relational database model**; • **Mathematics of computing** → *Time series analysis*; • **Theory of computation** → **Streaming models**.

Scaling for faster insight

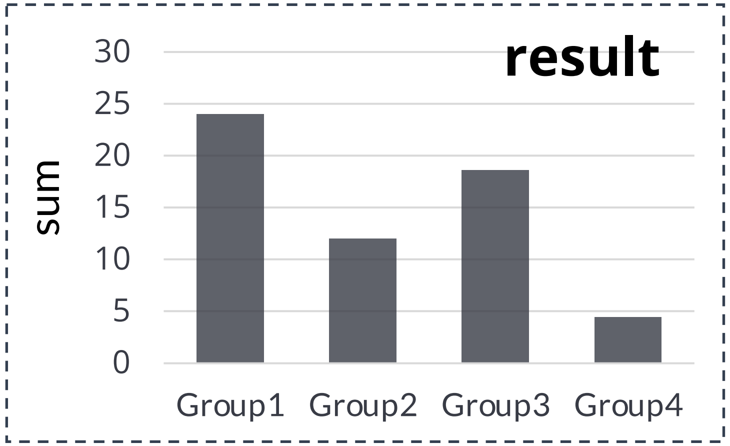
Input: Data

System:

Output:

?

waiting



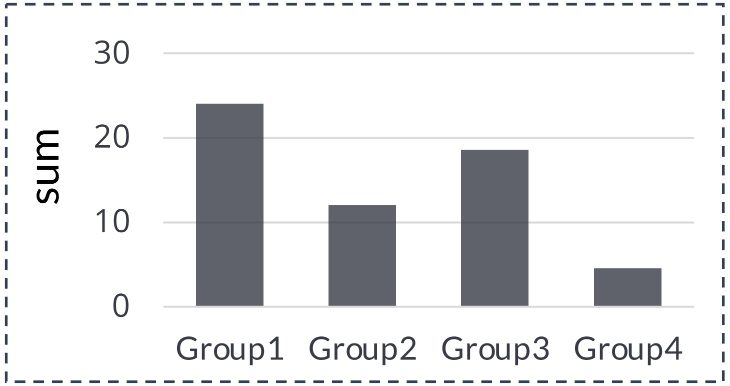
More VMs for the system, the more quickly we get a result

Alt: Predict final result from *partial processing*

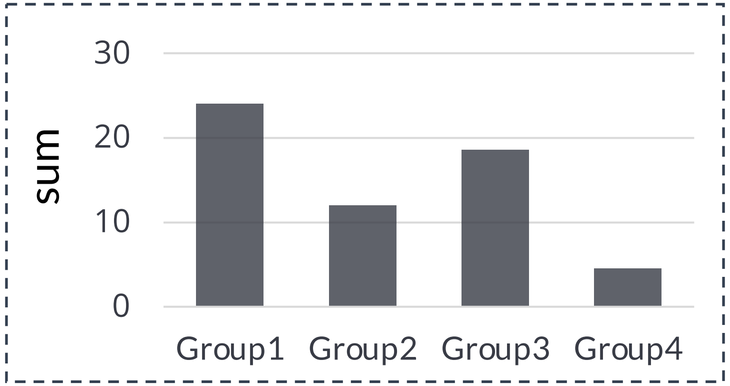
Input: Data

System: Online Aggregation

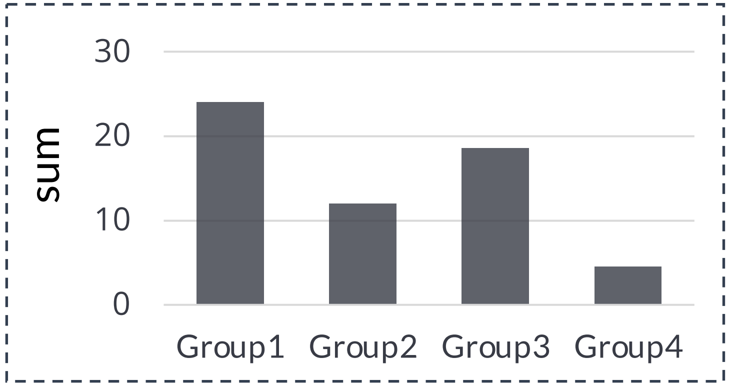
result in 1s



result in 2s



exact result in 10s



Initial results often provide enough information

Currently, limited to *simple queries*

```
lineitem = read_csv('...')
# item count for each order
order_qty = lineitem.sum(qty, by=orderkey)
# select only the large orders
lg_o          order_qty.filter(sum_qty > 300)
# fi                                  rder sizes
lg_order_cus
qty_per_cust = lg_order_cust.sum(sum_
top_cust = qty_per_cust.sort(sum_qty, desc=True).limit(100)
```

cannot apply these subsequent OPs

Can we continuously update each data frame?

key concept

How to represent online aggregation results?

A Step Toward Deep Online Aggregation

NIKHIL SHEORAN[*,†], Databricks, USA
SUPAWIT CHOCKCHOWWAT[†], University of Illinois at Urbana-Champaign, USA
ARAV CHHEDA, University of Illinois at Urbana-Champaign, USA
SUWEN WANG, University of Illinois at Urbana-Champaign, USA
RIYA VERMA, University of Illinois at Urbana-Champaign, USA
YONGJOO PARK, University of Illinois at Urbana-Champaign, USA

For exploratory data analysis, it is often desirable to know what answers you are likely to get *before* actually obtaining those answers. This can potentially be achieved by designing systems to offer the estimates of a data operation result—say op(data)—earlier in the process based on partial data processing. Those estimates continuously refine as more data is processed and finally converge to the exact answer. Unfortunately, the existing techniques—called *Online Aggregation* (OLA)—are limited to a single operation; that is, we *cannot* obtain the estimates for op(op(data)) or op(...(op(data))). If this *Deep OLA* becomes possible, data analysts will be able to explore data more interactively using complex cascade operations.

In this work, we take a step toward *Deep OLA* with *evolving data frames* (edf), a novel data model to offer OLA for nested ops—op(...(op(data)))—by representing an evolving structured data (with converging estimates) that is *closed* under set operations. That is, op(edf) produces yet another edf; thus, we can freely apply successive operations to edf and obtain an OLA output for each op. We evaluate its viability with Wake, an edf-based OLA system, by examining against state-of-the-art OLA and non-OLA systems. In our experiments on TPC-H dataset, Wake produces its first estimates 4.93× faster (median)—with 1.3× median slowdown for exact answers—compared to conventional systems. Besides its generality, Wake is also 1.92× faster (median) than existing OLA systems in producing estimates of under 1% relative errors.

124

CCS Concepts: • **Information systems** → **Database query processing**; **Online analytical processing engines**; *Relational parallel and distributed DBMSs*; **Uncertainty**; **Relational database model**; • **Mathematics of computing** → *Time series analysis*; • **Theory of computation** → **Streaming models**.

Types *closed* under operations

Integers are *closed* under addition

- `(1 + 3) + 3 = 4 + 3`

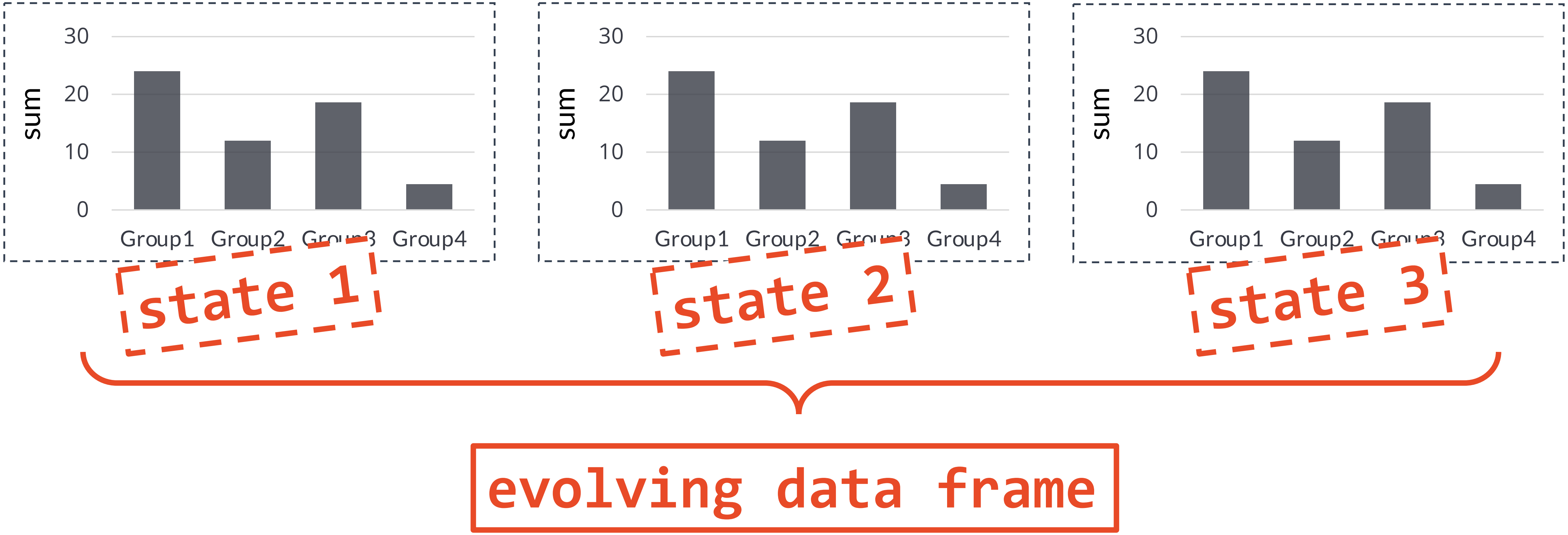
Relations (or tables) are *closed* under relational operations

- `lineitem = pandas.read_csv('lineitem.csv')`
- `order_qty = lineitem.sum(qty, by=orderkey)`
- `lg_orders = order_qty.filter(sum_qty > 300)`

We need an **OLA type** *closed* under OLA operations

How to represent an *evolving* object?

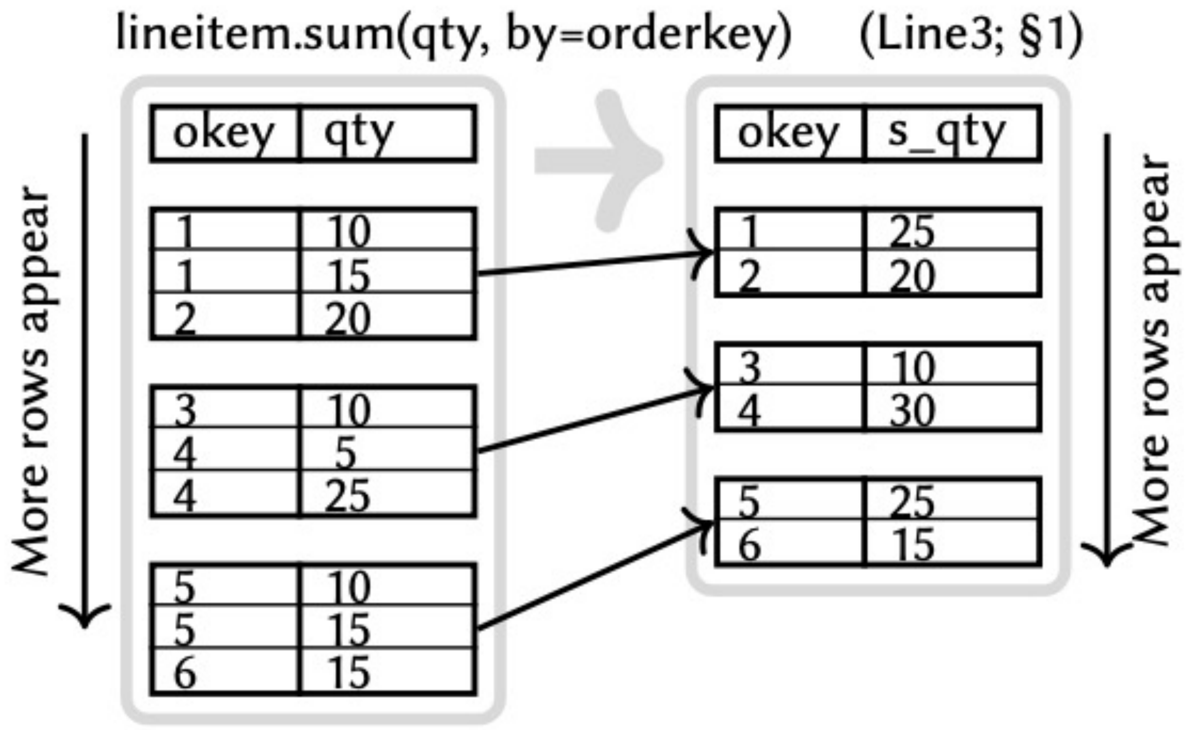
result in 1s **result in 2s** **exact result in 10s**



How are these individual **states** transformed?

Case 1: order-preserving local operation

Input: `lineitem` (*sorted on* `orderkey`) **OP:** sum `qty` by `orderkey`



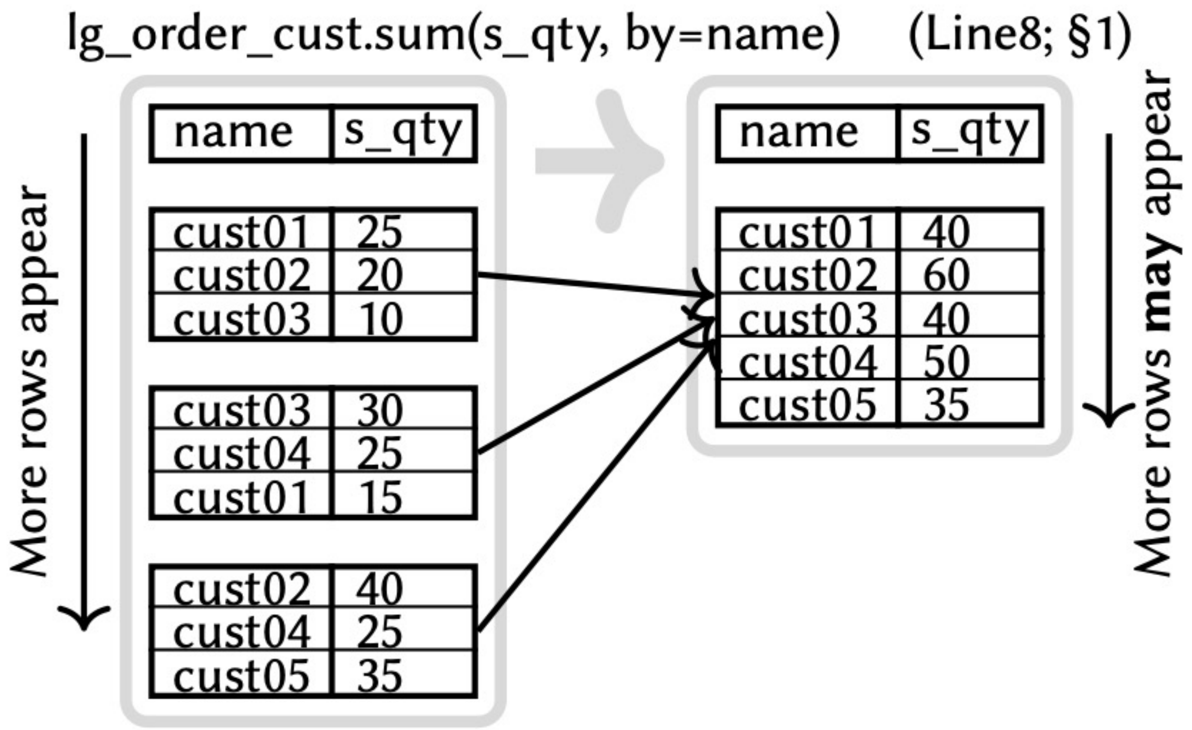
Properties:

1. More rows may appear

incremental processing

Case 2: shuffling with inference

Input: `lg_order_cust` (*sorted on* `orderkey`) **OP:** sum `s_qty` by `name`



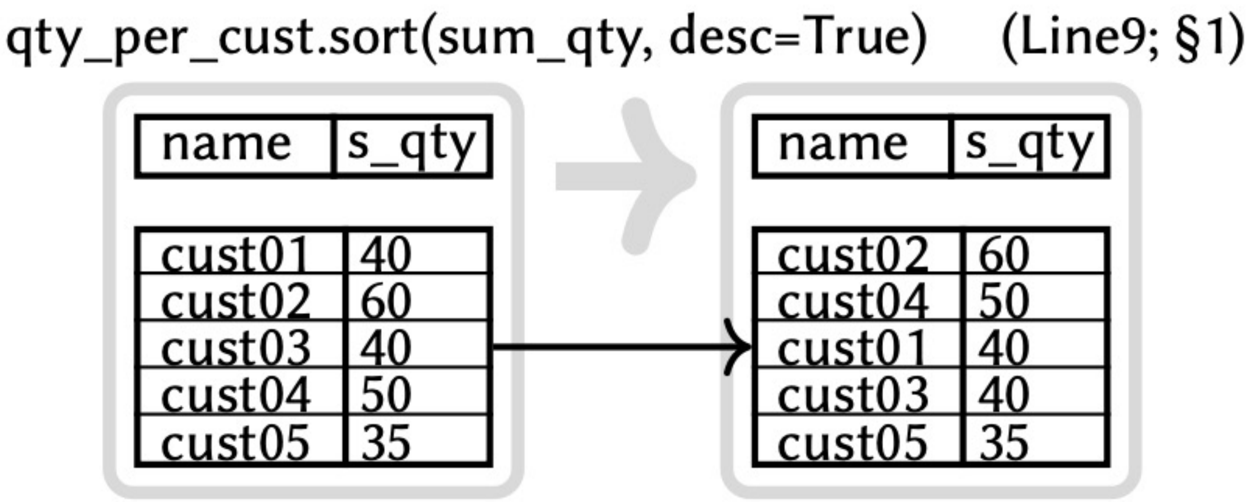
Properties:

1. More rows may appear
2. Attribute values may change
3. Values may need scaling (= **prediction**)

merge into existing results

Case 3: shuffling without inference

Input: `qty_per_cust` **OP:** sort by `sum_qty`



Properties:

1. More rows may appear
2. Attribute values may change
3. ~~Values may need scaling (= **prediction**)~~

complete refresh

Summary: Only a few transformation patterns

Types of Transformation:

- Case 1: order-preserving OP
- Case 2: shuffle with inference
- Case 3: complete refresh

In *transformed* output:

1. More rows may appear
2. Attribute values may change
3. Values may need scaling

New concepts introduced:

cardinality growth (query progress vs cardinality)
mutable attributes (values can change or not)

internal processing

How to represent states and efficiently generate new states?

A Step Toward Deep Online Aggregation

NIKHIL SHEORAN*†, Databricks, USA
SUPAWIT CHOCKCHOWWAT†, University of Illinois at Urbana-Champaign, USA
ARAV CHHEDA, University of Illinois at Urbana-Champaign, USA
SUWEN WANG, University of Illinois at Urbana-Champaign, USA
RIYA VERMA, University of Illinois at Urbana-Champaign, USA
YONGJOO PARK, University of Illinois at Urbana-Champaign, USA

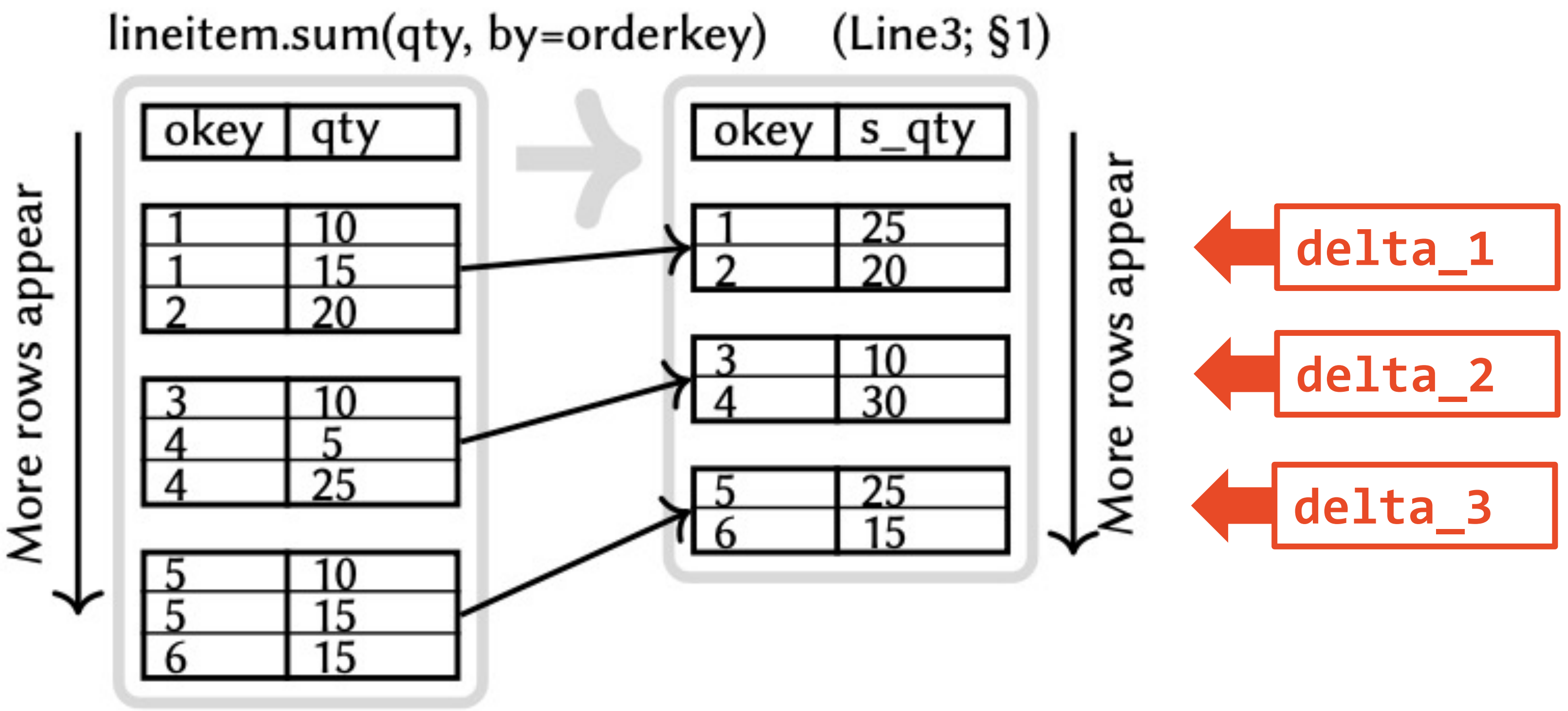
For exploratory data analysis, it is often desirable to know what answers you are likely to get *before* actually obtaining those answers. This can potentially be achieved by designing systems to offer the estimates of a data operation result—say op(data)—earlier in the process based on partial data processing. Those estimates continuously refine as more data is processed and finally converge to the exact answer. Unfortunately, the existing techniques—called *Online Aggregation* (OLA)—are limited to a single operation; that is, we *cannot* obtain the estimates for op(op(data)) or op(...(op(data))). If this *Deep OLA* becomes possible, data analysts will be able to explore data more interactively using complex cascade operations.

In this work, we take a step toward *Deep OLA* with *evolving data frames* (edf), a novel data model to offer OLA for nested ops—op(...(op(data)))—by representing an evolving structured data (with converging estimates) that is *closed* under set operations. That is, op(edf) produces yet another edf; thus, we can freely apply successive operations to edf and obtain an OLA output for each op. We evaluate its viability with Wake, an edf-based OLA system, by examining against state-of-the-art OLA and non-OLA systems. In our experiments on TPC-H dataset, Wake produces its first estimates 4.93× faster (median)—with 1.3× median slowdown for exact answers—compared to conventional systems. Besides its generality, Wake is also 1.92× faster (median) than existing OLA systems in producing estimates of under 1% relative errors.

124

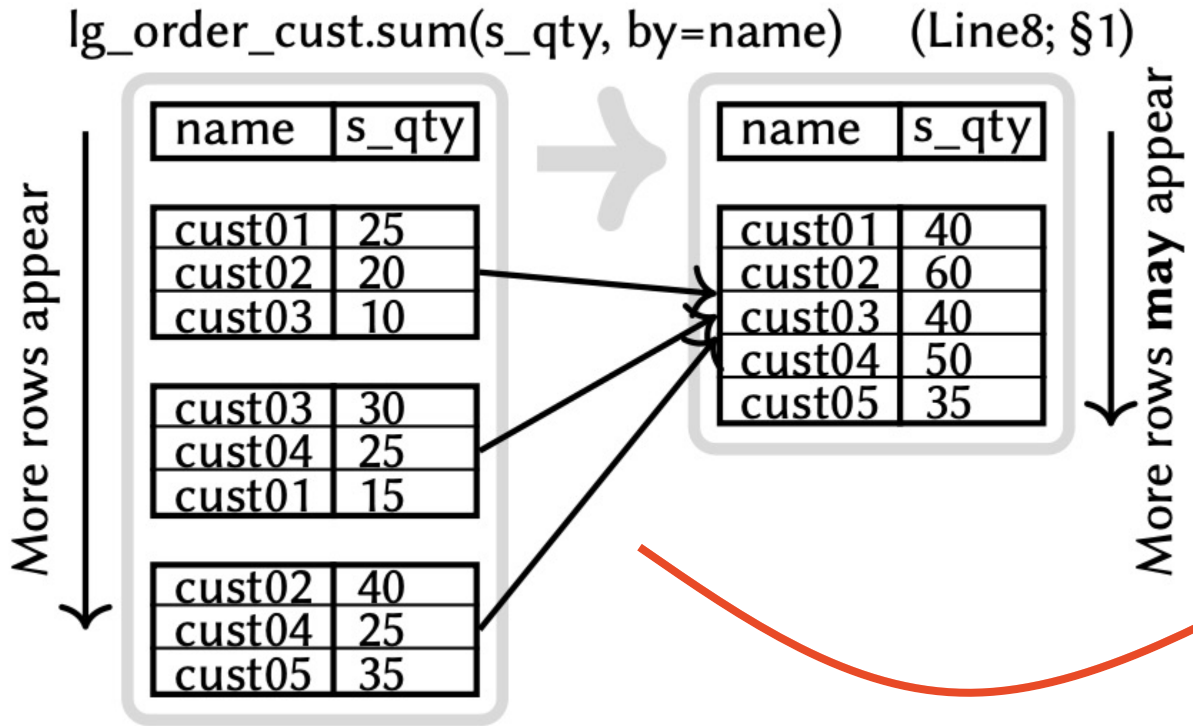
CCS Concepts: • **Information systems** → **Database query processing**; **Online analytical processing engines**; *Relational parallel and distributed DBMSs*; **Uncertainty**; **Relational database model**; • **Mathematics of computing** → *Time series analysis*; • **Theory of computation** → **Streaming models**.

Case 1: order-preserving local operation



state_k = delta_1 ∪ delta_2 ∪ … ∪ delta_k

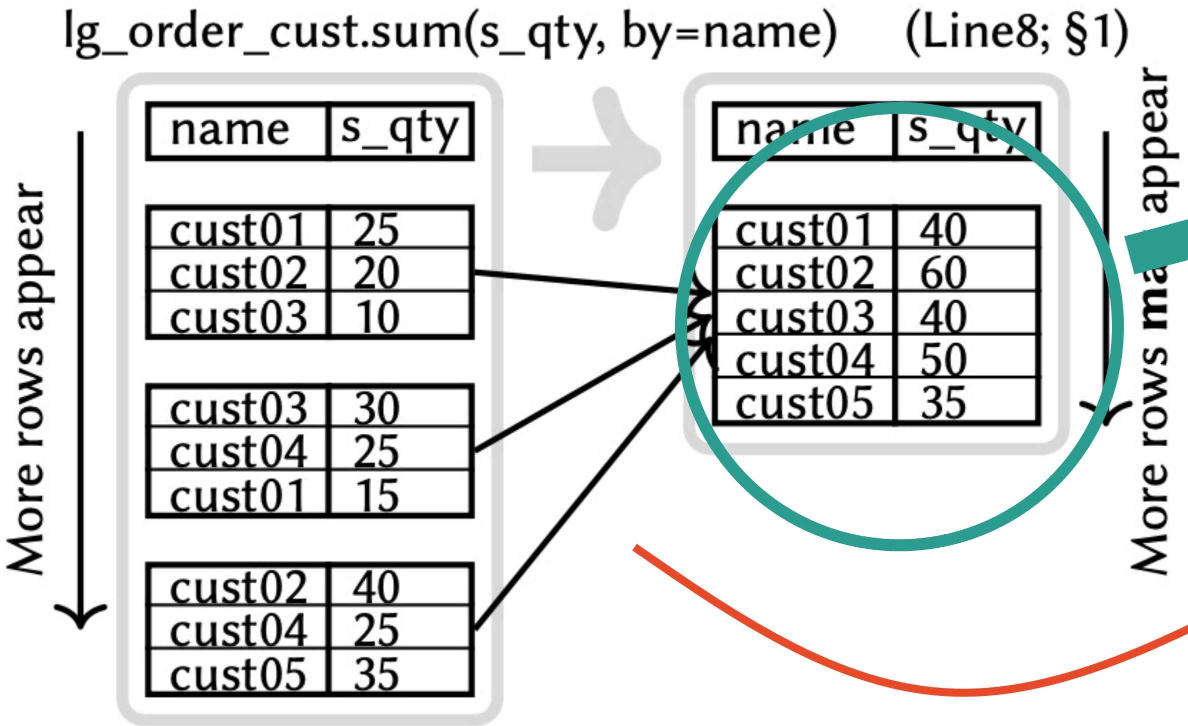
Case 2: shuffling with inference (1/2)



Each merge creates a new ***version*** of state

`state_k = version_k` *(after scaling)*

Case 2: shuffling with inference (2/2)



These values need **scaling** w.r.t. *current input size*

Each merge creates a new ***version*** of state

The values are scaling with (**est total input size** / current size)

Case 3: shuffling without inference

Input: `qty_per_cust` **OP:** sort by `sum_qty`

qty_per_cust.sort(sum_qty, desc=True) (Line9; §1)

name	s_qty
cust01	40
cust02	60
cust03	40
cust04	50
cust05	35

name	s_qty
cust02	60
cust04	50
cust01	40
cust03	40
cust05	35

`state_k = version_k` *(without scaling)*

putting it together

How can a user use it end-to-end?

A Step Toward Deep Online Aggregation

NIKHIL SHEORAN*†, Databricks, USA
SUPAWIT CHOCKCHOWWAT†, University of Illinois at Urbana-Champaign, USA
ARAV CHHEDA, University of Illinois at Urbana-Champaign, USA
SUWEN WANG, University of Illinois at Urbana-Champaign, USA
RIYA VERMA, University of Illinois at Urbana-Champaign, USA
YONGJOO PARK, University of Illinois at Urbana-Champaign, USA

For exploratory data analysis, it is often desirable to know what answers you are likely to get *before* actually obtaining those answers. This can potentially be achieved by designing systems to offer the estimates of a data operation result—say op(data)—earlier in the process based on partial data processing. Those estimates continuously refine as more data is processed and finally converge to the exact answer. Unfortunately, the existing techniques—called *Online Aggregation* (OLA)—are limited to a single operation; that is, we *cannot* obtain the estimates for op(op(data)) or op(...(op(data))). If this *Deep OLA* becomes possible, data analysts will be able to explore data more interactively using complex cascade operations.

In this work, we take a step toward *Deep OLA* with *evolving data frames* (edf), a novel data model to offer OLA for nested ops—op(...(op(data)))—by representing an evolving structured data (with converging estimates) that is *closed* under set operations. That is, op(edf) produces yet another edf; thus, we can freely apply successive operations to edf and obtain an OLA output for each op. We evaluate its viability with WAKE, an edf-based OLA system, by examining against state-of-the-art OLA and non-OLA systems. In our experiments on TPC-H dataset, WAKE produces its first estimates 4.93× faster (median)—with 1.3× median slowdown for exact answers—compared to conventional systems. Besides its generality, WAKE is also 1.92× faster (median) than existing OLA systems in producing estimates of under 1% relative errors.

124

CCS Concepts: • **Information systems** → **Database query processing**; **Online analytical processing engines**; *Relational parallel and distributed DBMSs*; **Uncertainty**; **Relational database model**; • **Mathematics of computing** → *Time series analysis*; • **Theory of computation** → **Streaming models**.

Evolving Data Frame and Operations

Our Earlier Example

```
lineitem = read_csv('...')
# item count for each order
order_qty = lineitem.sum(qty, by=orderkey)
# select only the large orders
lg_orders = order_qty.filter(sum_qty > 300)
# find the customers with biggest order sizes
lg_order_cust = lg_orders.join(orders).join(customer)
qty_per_cust = lg_order_cust.sum(sum_qty, by=name)
top_cust = qty_per_cust.sort(sum_qty, desc=True).limit(100)
```

Operations generating *edf*

```
read := data_source -> edf
edf_op := (edf, op) -> edf
op  := agg(attrs, by) | filter(predicate)
     | map(function) | join(df, options)
agg := sum | count | avg | count_distinct | min | max
     | var | stddev
```

OLA operation on an edf generates edf

Cardinality and Aggregate Inference Logic

- Given the states of an edf, generate a user-consumable query output.
  - Growth: group sizes may grow in a non-linear way as more input data are processed *(Cardinality estimators)*
  - Coverage: the number of groups covered may also increase over time *(Cardinality estimators)*
  - Operation: different types of aggregations requires different estimations *(Aggregate estimators)*

obtain estimation of final result

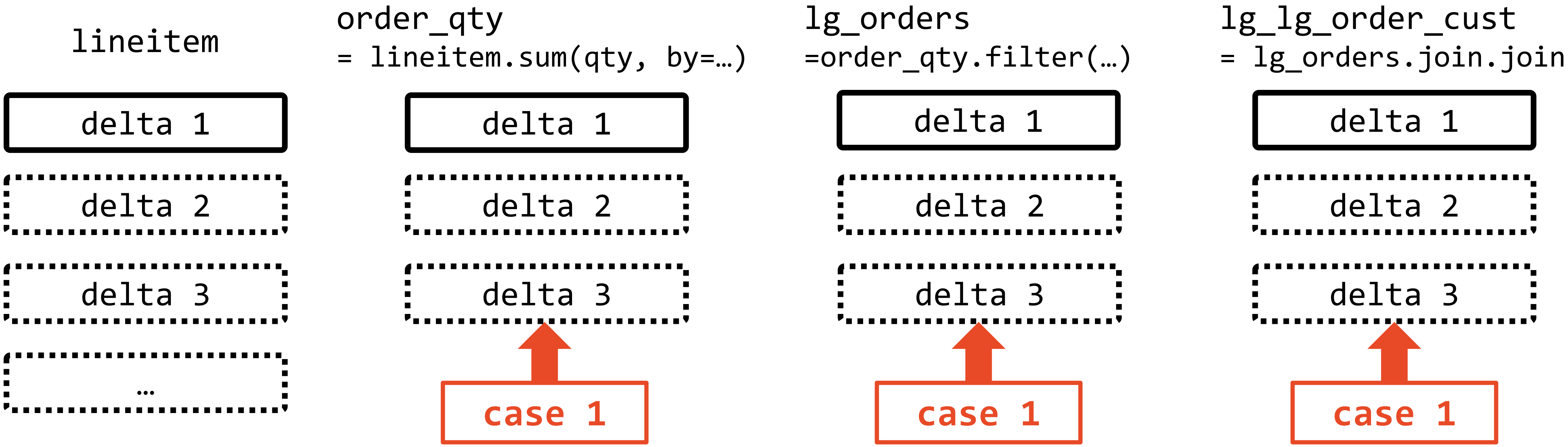
Confidence Interval for Deep OLA

- Compute "uncertainty" for all mutable attributes.
  - Different techniques for different aggregation operations e.g. variance of OLS parameter, central limit theorem, etc.
- Propagate uncertainty through OLA operations.
  - Linearize using a first-order taylor expansion and compute covariance matrix.
- Compute confidence intervals from final uncertainty.
  - Use Chebyshev's inequality for final CI estimate.

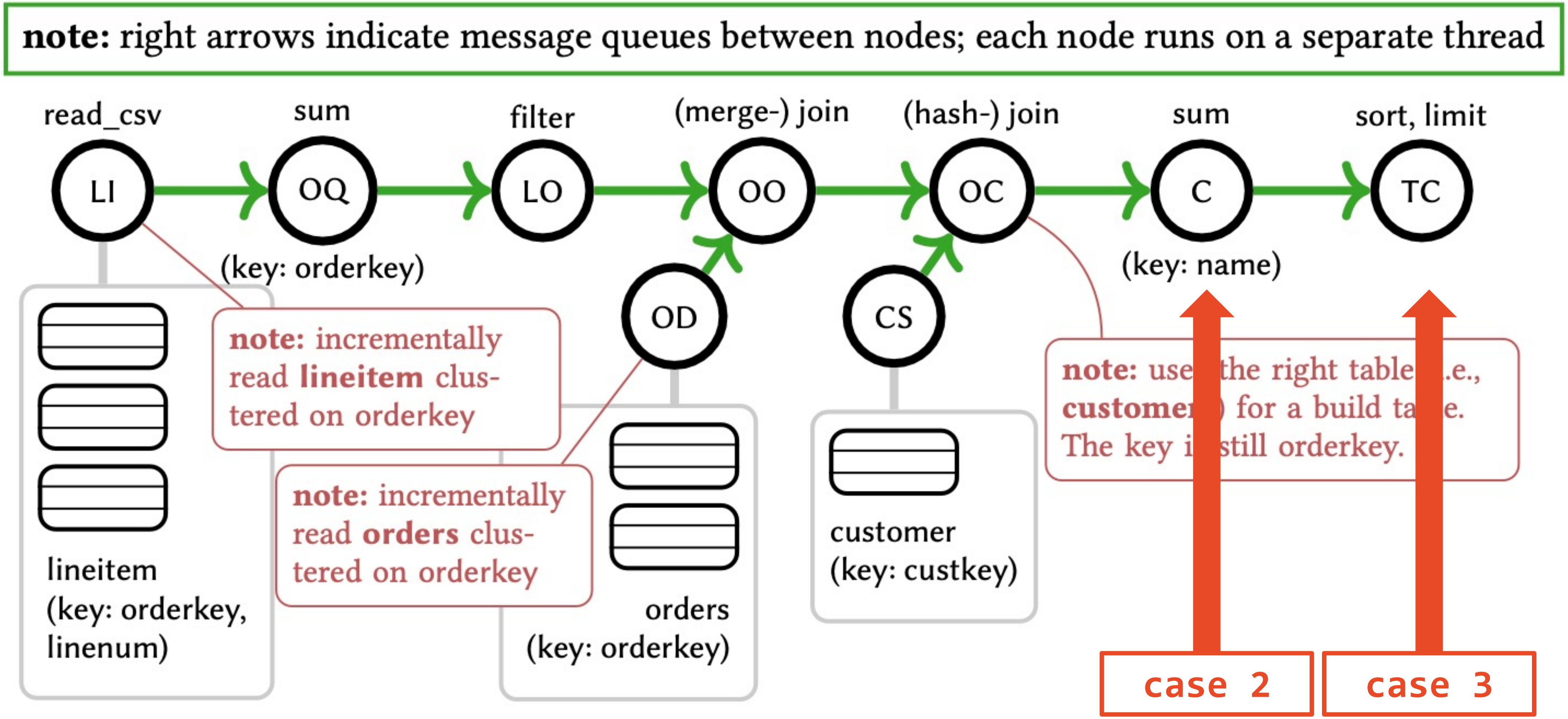
confidence interval on estimation

Putting together (first four EDFs)

```
lineitem = read_csv('...')
# item count for each order
order_qty = lineitem.sum(qty, by=orderkey)
# select only the large orders
lg_orders = order_qty.filter(sum_qty > 300)
# find the customers with biggest order sizes
lg_order_cust = lg_orders.join(orders).join(customer)
```



Complete data flow



evaluation

How much is the computation overhead and approximation error?

A Step Toward Deep Online Aggregation

NIKHIL SHEORAN*†, Databricks, USA
SUPAWIT CHOCKCHOWWAT†, University of Illinois at Urbana-Champaign, USA
ARAV CHHEDA, University of Illinois at Urbana-Champaign, USA
SUWEN WANG, University of Illinois at Urbana-Champaign, USA
RIYA VERMA, University of Illinois at Urbana-Champaign, USA
YONGJOO PARK, University of Illinois at Urbana-Champaign, USA

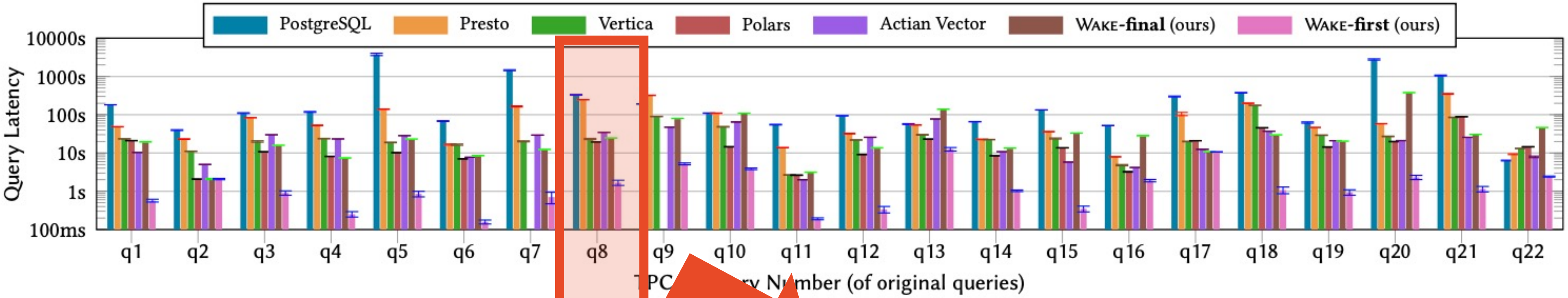
For exploratory data analysis, it is often desirable to know what answers you are likely to get *before* actually obtaining those answers. This can potentially be achieved by designing systems to offer the estimates of a data operation result—say op(data)—earlier in the process based on partial data processing. Those estimates continuously refine as more data is processed and finally converge to the exact answer. Unfortunately, the existing techniques—called *Online Aggregation* (OLA)—are limited to a single operation; that is, we *cannot* obtain the estimates for op(op(data)) or op(...(op(data))). If this *Deep OLA* becomes possible, data analysts will be able to explore data more interactively using complex cascade operations.

In this work, we take a step toward *Deep OLA* with *evolving data frames* (edf), a novel data model to offer OLA for nested ops—op(...(op(data)))—by representing an evolving structured data (with converging estimates) that is *closed* under set operations. That is, op(edf) produces yet another edf; thus, we can freely apply successive operations to edf and obtain an OLA output for each op. We evaluate its viability with Wake, an edf-based OLA system, by examining against state-of-the-art OLA and non-OLA systems. In our experiments on TPC-H dataset, Wake produces its first estimates 4.93× faster (median)—with 1.3× median slowdown for exact answers—compared to conventional systems. Besides its generality, Wake is also 1.92× faster (median) than existing OLA systems in producing estimates of under 1% relative errors.

124

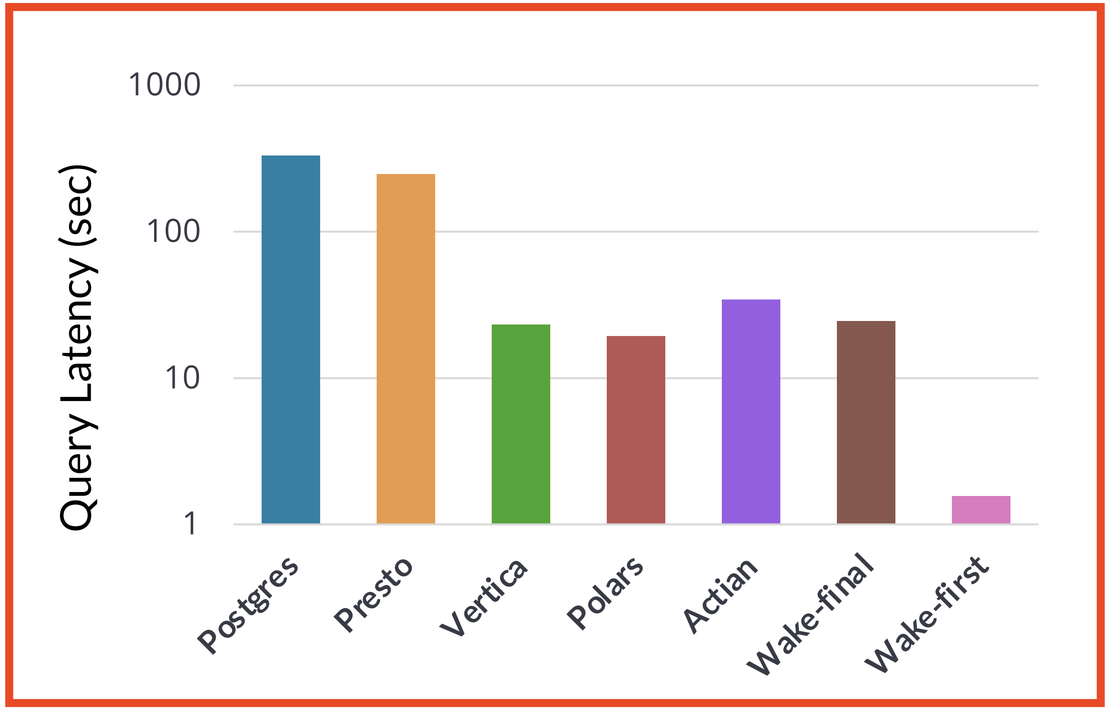
CCS Concepts: • **Information systems** → **Database query processing**; **Online analytical processing engines**; *Relational parallel and distributed DBMSs*; **Uncertainty**; **Relational database model**; • **Mathematics of computing** → *Time series analysis*; • **Theory of computation** → **Streaming models**.

Our OLA system delivers answers quickly




```
select o_year,
  sum(...) / sum(...) as mkt_share
from (
  select
    year(o_orderdate) as o_year,
    l_extendedprice * (1 - l_discount) as volume,
    n2.n_name as nation
  from
    part, supplier, lineitem, orders, customer,
    nation n1, nation n2, region
  where ...
  ) as all_nations
group by o_year
order by o_year;
```

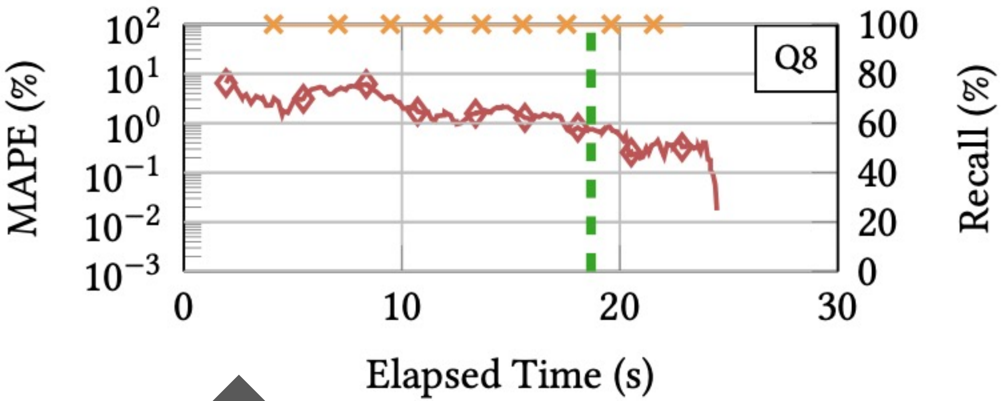
Q8



Our errors decrease quickly

Q8

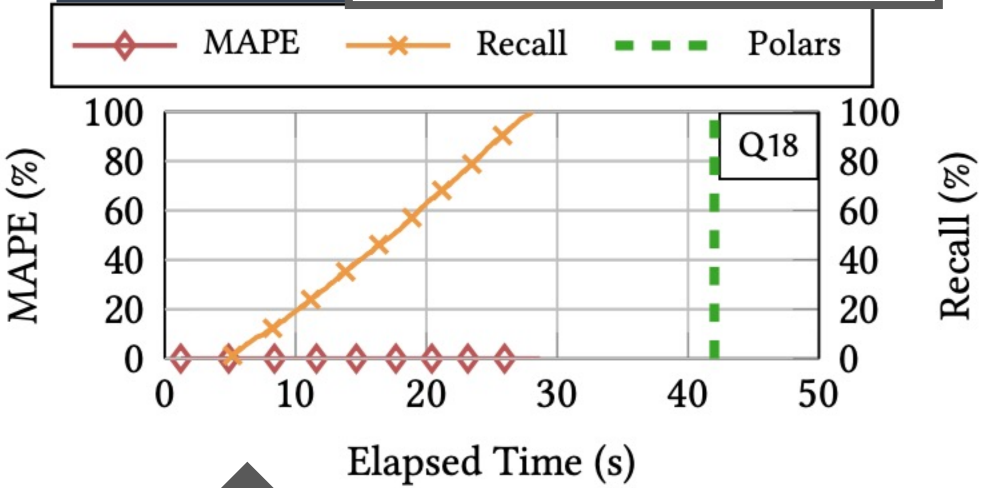
- All rows are present
- Attr values improve



- Our result has a complete row set
- The errors decrease quickly

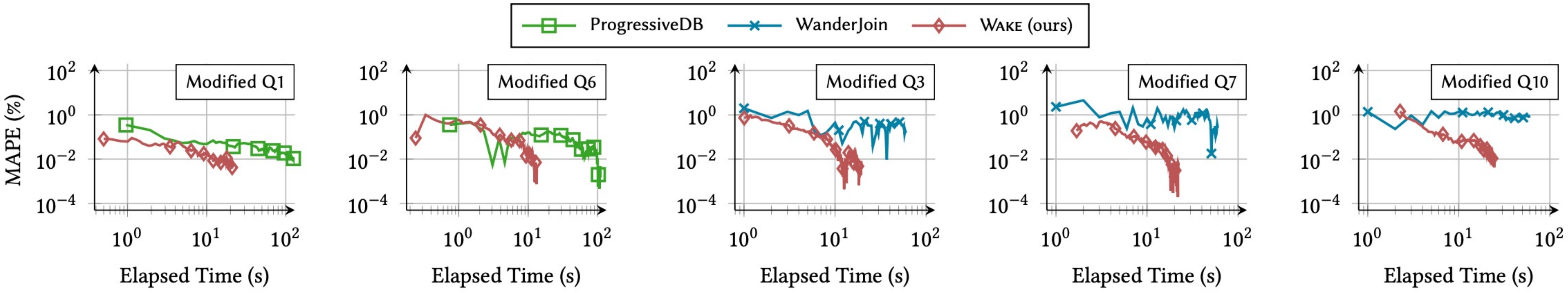
Q18

- Attr values are exact
- More rows appear



- Attr value errors are zero
- Result set increase

Faster & more *accurate* than existing OLA



(a) vs ProgressiveDB (using single-table Q1 and Q6) **(b) vs WanderJoin (using same modified Q3, Q7, and Q10 as in [50])**

- The lower, the better *(we are lower)*
- **Reason 1: we are highly parallel** **Reason 2: our final answers are exact**

Conclusion: A Step Toward Deep OLA

- **First OLA** for processing arbitrarily **nested queries**
- Motivation: A **new type** for OLA
- Proposed *Evolving Data Frame* (EDF)
- EDF, consisting of multiple states, is *closed* under OPs
- Evaluation: ***low latency***, ***high accuracy***, and ***improvement over STOA***

Thank you!