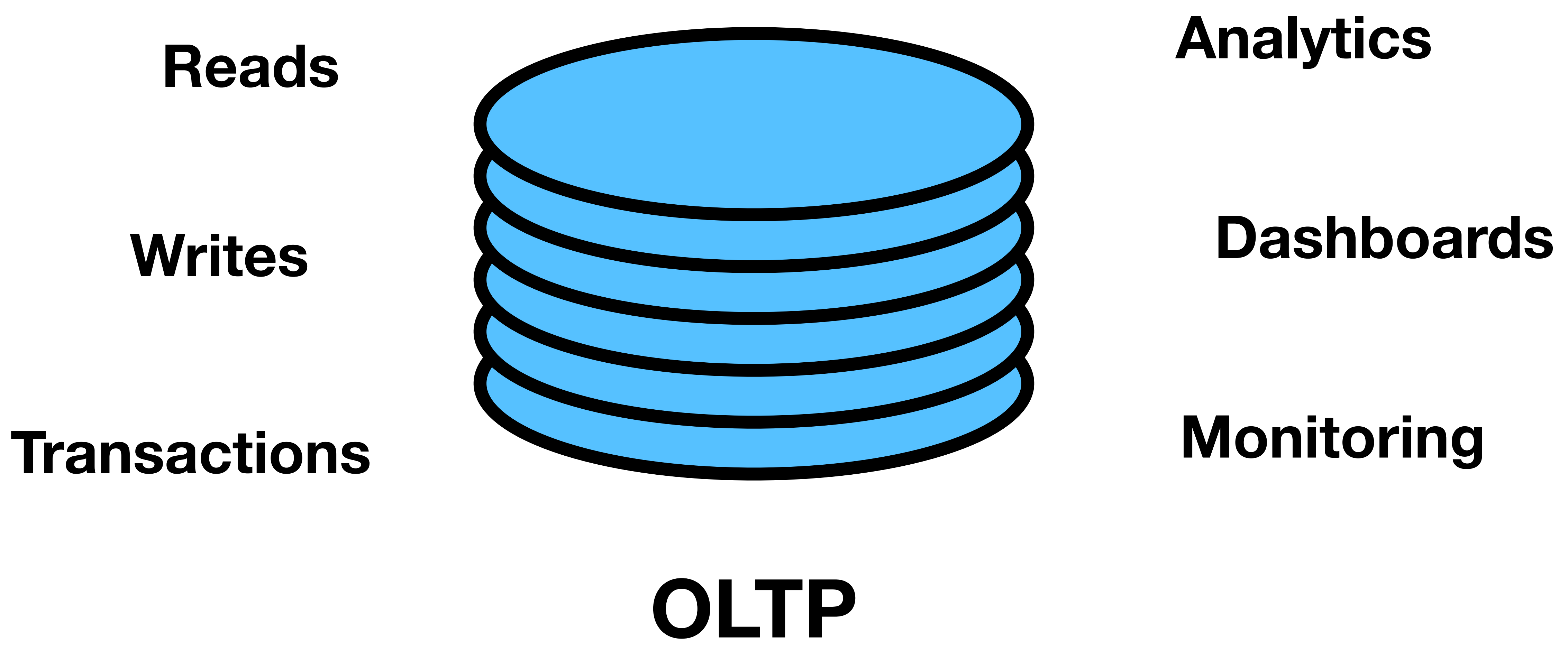
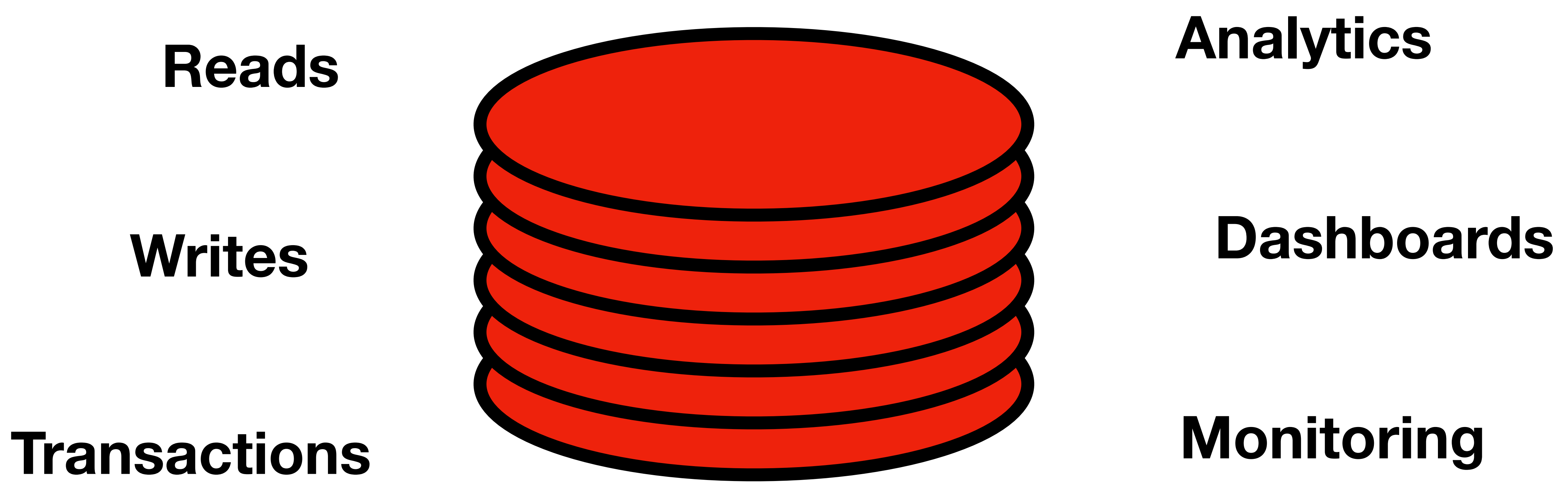
Materialize and Streaming SQL

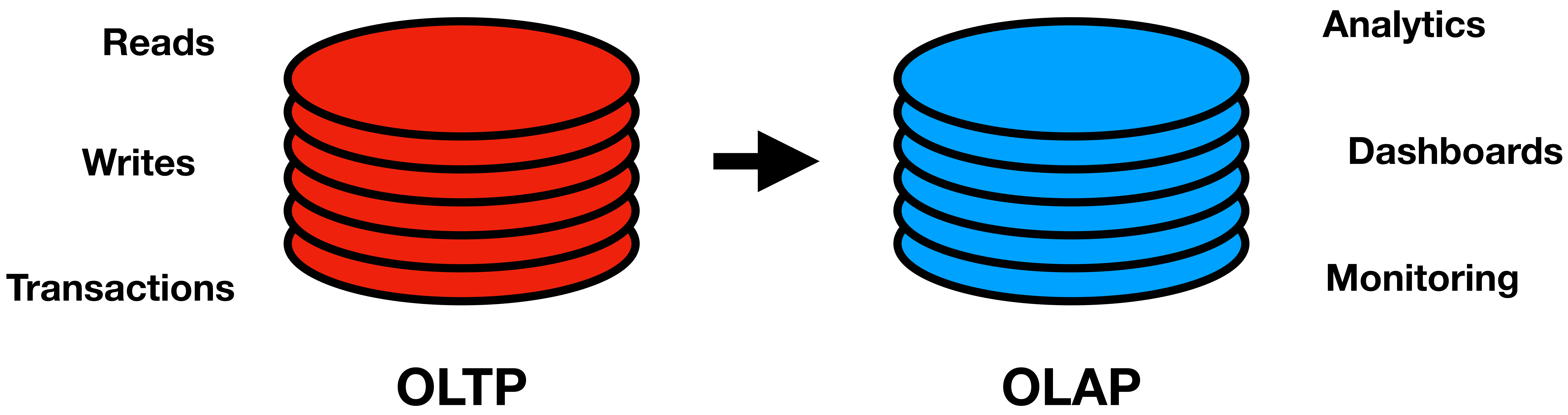
Standard SQL as a Basis for Streaming Data Infrastructure

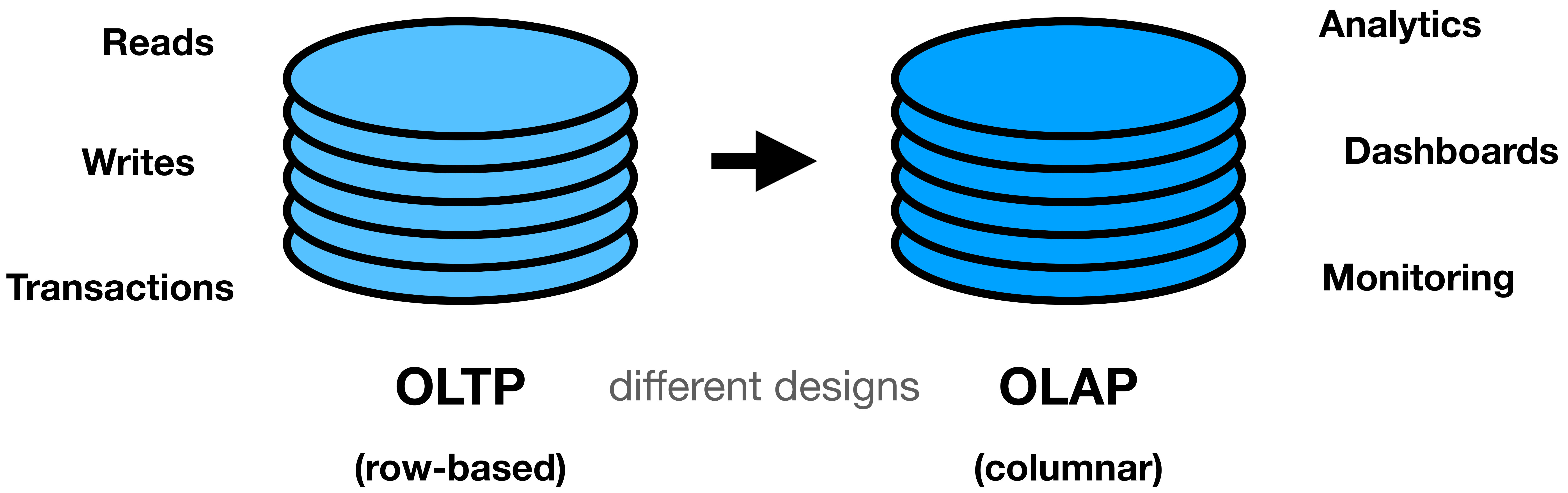
Frank McSherry, Chief Scientist

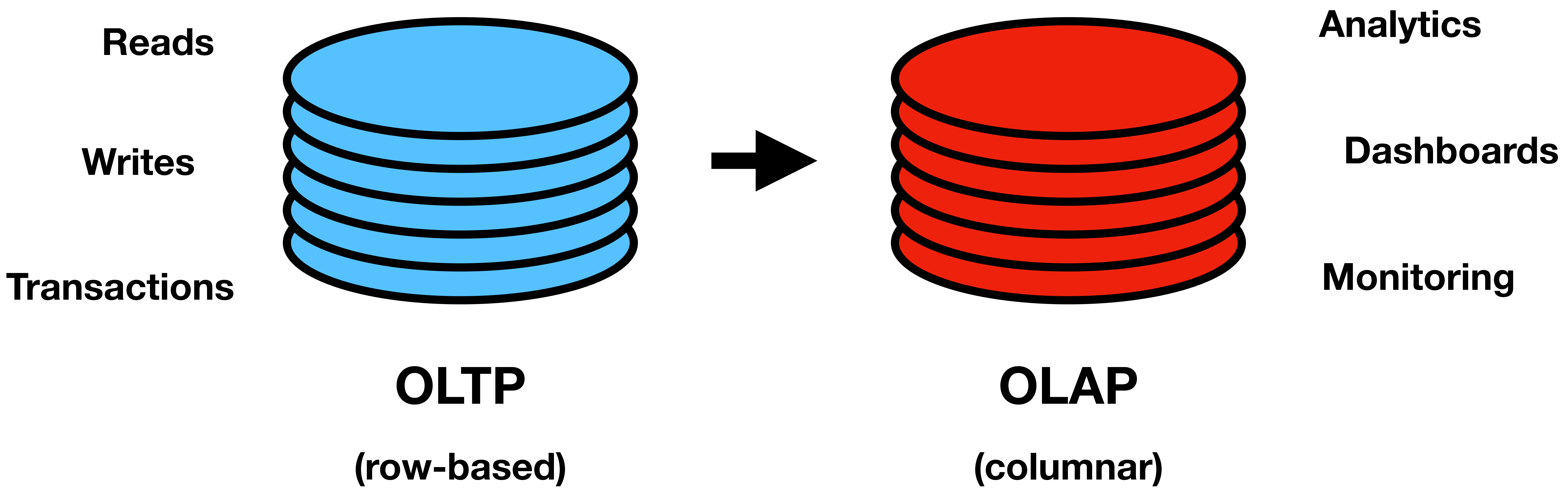

Reads
Writes
Transactions
Analytics
Dashboards
Monitoring
OLTP

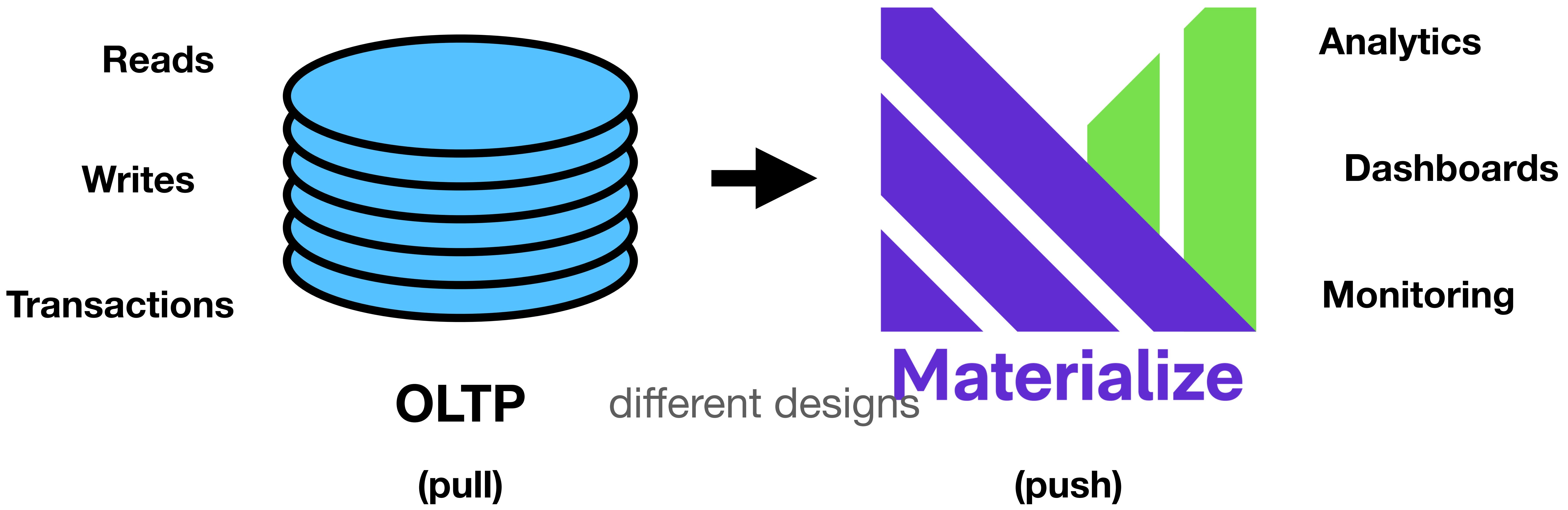

Reads
Writes
Transactions
Analytics
Dashboards
Monitoring

OLTP


Reads
Writes
Transactions
OLTP
Analytics
Dashboards
Monitoring
OLAP


Reads
Writes
Transactions
OLTP
(row-based)
different designs
OLAP
(columnar)
Analytics
Dashboards
Monitoring


Reads
Writes
Transactions
OLTP
(row-based)
OLAP
(columnar)
Analytics
Dashboards
Monitoring


Reads
Writes
Transactions
OLTP
(pull)
different designs
Materialize
(push)
Analytics
Dashboards
Monitoring

Standard SQL is expressive enough for streaming data infrastructure tasks.

Standard SQL is expressive enough for streaming data infrastructure tasks...

...with a SQL system like Materialize.

Materialize

Maintain SQL views on streams

SQL92, even the hard stuff.

```
-- a stream of CDC input
CREATE SOURCE foo FROM ...
-- traditional SQL views
CREATE VIEW bar AS SELECT ...
-- indexes arrange streams
CREATE INDEX baz ON bar ...
-- emit CDC stream somewhere
CREATE SINK quux FROM bar ...
```

Materialize

Maintain SQL views on streams

SQL92, even the hard stuff.

```
-- a stream of CDC input
CREATE SOURCE foo FROM ...
-- traditional SQL views
CREATE VIEW bar AS SELECT ...
-- indexes arrange streams
CREATE INDEX baz ON bar ...
-- emit CDC stream somewhere
CREATE SINK quux FROM bar ...
```

```
-- a stream of CDC input
CREATE SOURCE lineitem_src
FROM FILE '/Users/
           mcsherry/
           Projects/
           datasets/
           dbgen-1/
           lineitem.tbl'
FORMAT CSV
WITH 17 COLUMNS DELIMITED BY '|';
```

Materialize

Maintain SQL views on streams

SQL92, even the hard stuff.

```
-- a stream of CDC input
CREATE SOURCE foo FROM ...
-- traditional SQL views
CREATE VIEW bar AS SELECT ...
-- indexes arrange streams
CREATE INDEX baz ON bar ...
-- emit CDC stream somewhere
CREATE SINK quux FROM bar ...
```

```
-- traditional SQL views
CREATE VIEW lineitem AS
SELECT
    column1::integer as l_orderkey,
    column2::integer as l_partkey,
    column3::integer as l_suppkey,
    column4::integer as l_linenumber,
    column5::decimal(15,2) as l_quantity,
    column6::decimal(15,2) as l_extendedprice,
    column7::decimal(15,2) as l_discount,
    column8::decimal(15,2) as l_tax,
    column9 as l_returnflag,
    column10 as l_linestatus,
    column11::date as l_shipdate,
    column12::date as l_commitdate,
    column13::date as l_receiptdate,
    column14 as l_shipinstruct,
    column15 as l_shipmode,
    column16 as l_comment
FROM
    lineitem_src;
```

Materialize

Maintain SQL views on streams

SQL92, even the hard stuff.

```
-- a stream of CDC input
CREATE SOURCE foo FROM ...
-- traditional SQL views
CREATE VIEW bar AS SELECT ...
-- indexes arrange streams
CREATE INDEX baz ON bar ...
-- emit CDC stream somewhere
CREATE SINK quux FROM bar ...
```

```
-- indexes arrange streams
CREATE INDEX pk_lineitem ON
    lineitem (l_orderkey, l_linenumber);
CREATE INDEX fk_lineitem_orderkey ON
    lineitem (l_orderkey);
CREATE INDEX fk_lineitem_partkey ON
    lineitem (l_partkey);
CREATE INDEX fk_lineitem_suppkey ON
    lineitem (l_suppkey);
CREATE INDEX fk_lineitem_partsuppkey ON
    lineitem (l_partkey, l_suppkey);
```

Materialize

Maintain SQL views on streams

SQL92, even the hard stuff.

```
-- a stream of CDC input
CREATE SOURCE foo FROM ...
-- traditional SQL views
CREATE VIEW bar AS SELECT ...
-- indexes arrange streams
CREATE INDEX baz ON bar ...
-- emit CDC stream somewhere
CREATE SINK quux FROM bar ...
```

```
CREATE MATERIALIZED VIEW tpch_q05 AS
SELECT
    n_name,
    sum(l_extendedprice * (1 - l_discount)) AS
FROM
    customer,
    orders,
    lineitem,
    supplier,
    nation,
    region
WHERE
    c_custkey = o_custkey
    AND l_orderkey = o_orderkey
    AND l_suppkey = s_suppkey
    AND c_nationkey = s_nationkey
    AND s_nationkey = n_nationkey
    AND n_regionkey = r_regionkey
    AND r_name = 'ASIA'
    AND o_orderdate >= DATE '1994-01-01'
    AND o_orderdate < DATE '1995-01-01'
GROUP BY
    n_name;
```

Materialize

Maintain SQL views on streams

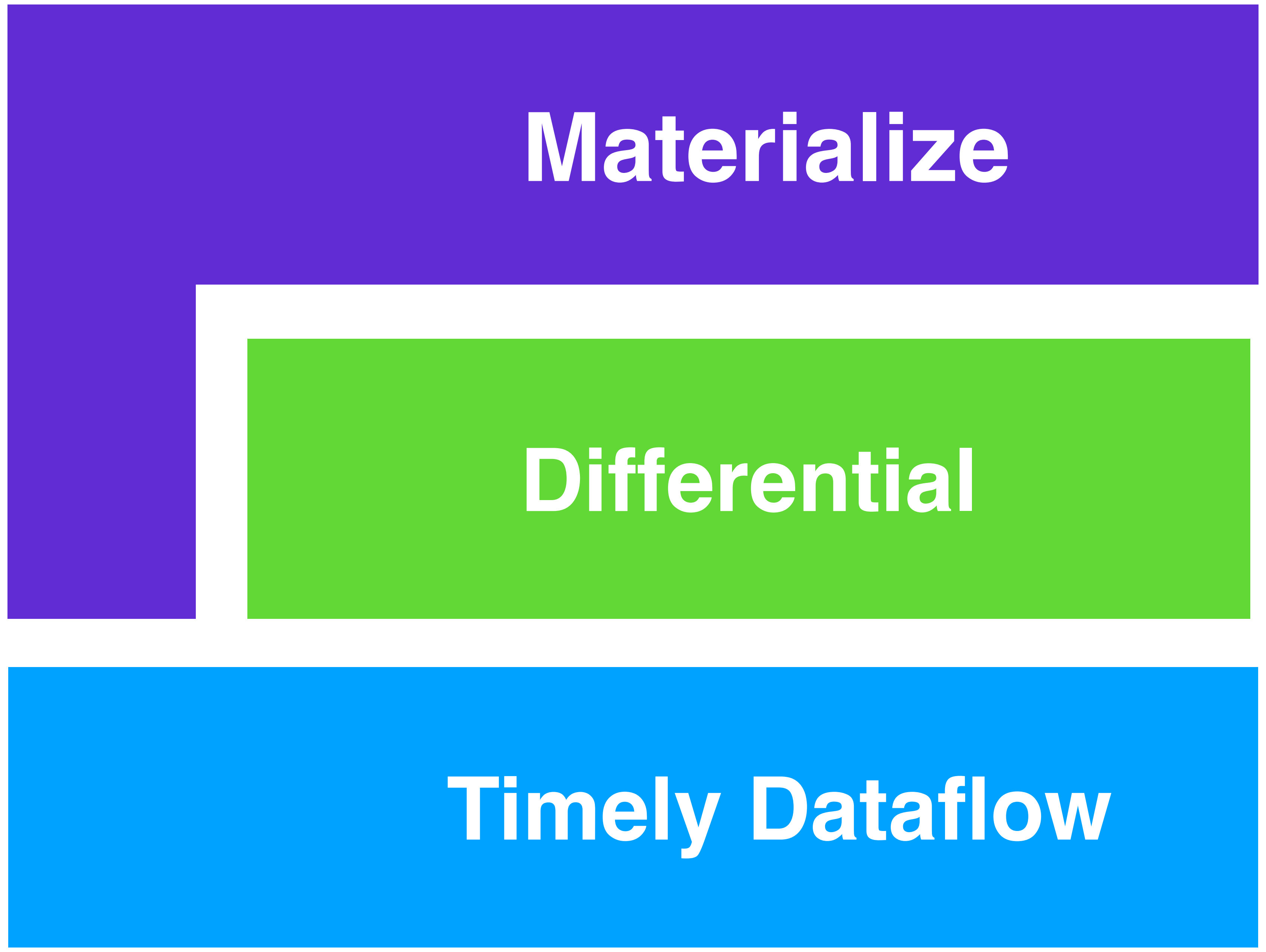
SQL92, even the hard stuff.

```
-- a stream of CDC input
CREATE SOURCE foo FROM ...
-- traditional SQL views
CREATE VIEW bar AS SELECT ...
-- indexes arrange streams
CREATE INDEX baz ON bar ...
-- emit CDC stream somewhere
CREATE SINK quux FROM bar ...
```

```
-- emit cdc streams somewhere
CREATE SINK tpch_q05_sink
FROM tpch_q05
INTO KAFKA
     BROKER 'localhost'
     TOPIC 'tpch-q05-sink'
FORMAT AVRO
ENVELOPE UPSERT;
```

Materialize

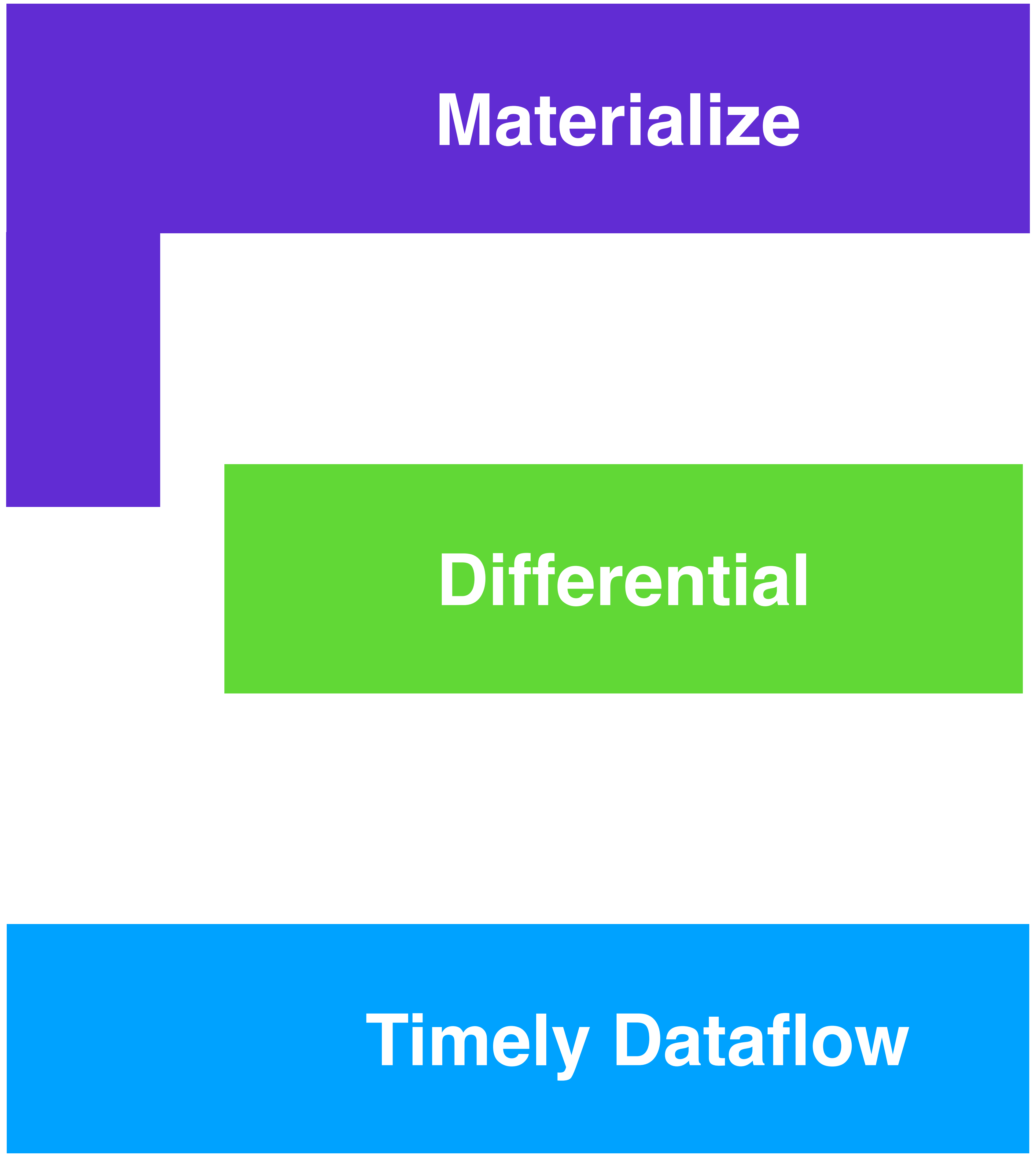
SQL on Streams of Data



SQL92 wrapper

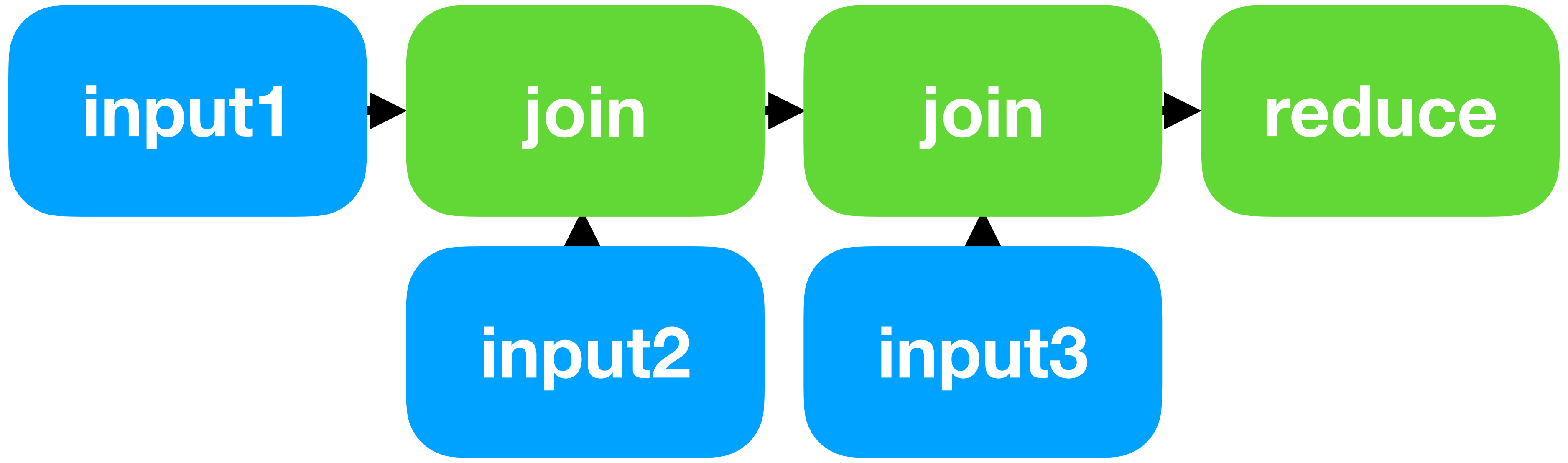
Language for low-latency incremental computation

Like an OS for streaming data-parallel compute




```sql
-- Aggregate the results of joins
SELECT input3.attr, SUM(val2), MAX(val3)
FROM input1, input2, input3
WHERE input1.fkey2 = input2.key
  AND input1.fkey3 = input3.key
GROUP BY input3.attr
```

```
// differential dataflow program
input1.join(input2, ...)
      .join(input3, ...)
      .reduce(...)
```



Timely Dataflow

An OS for streaming dataflows

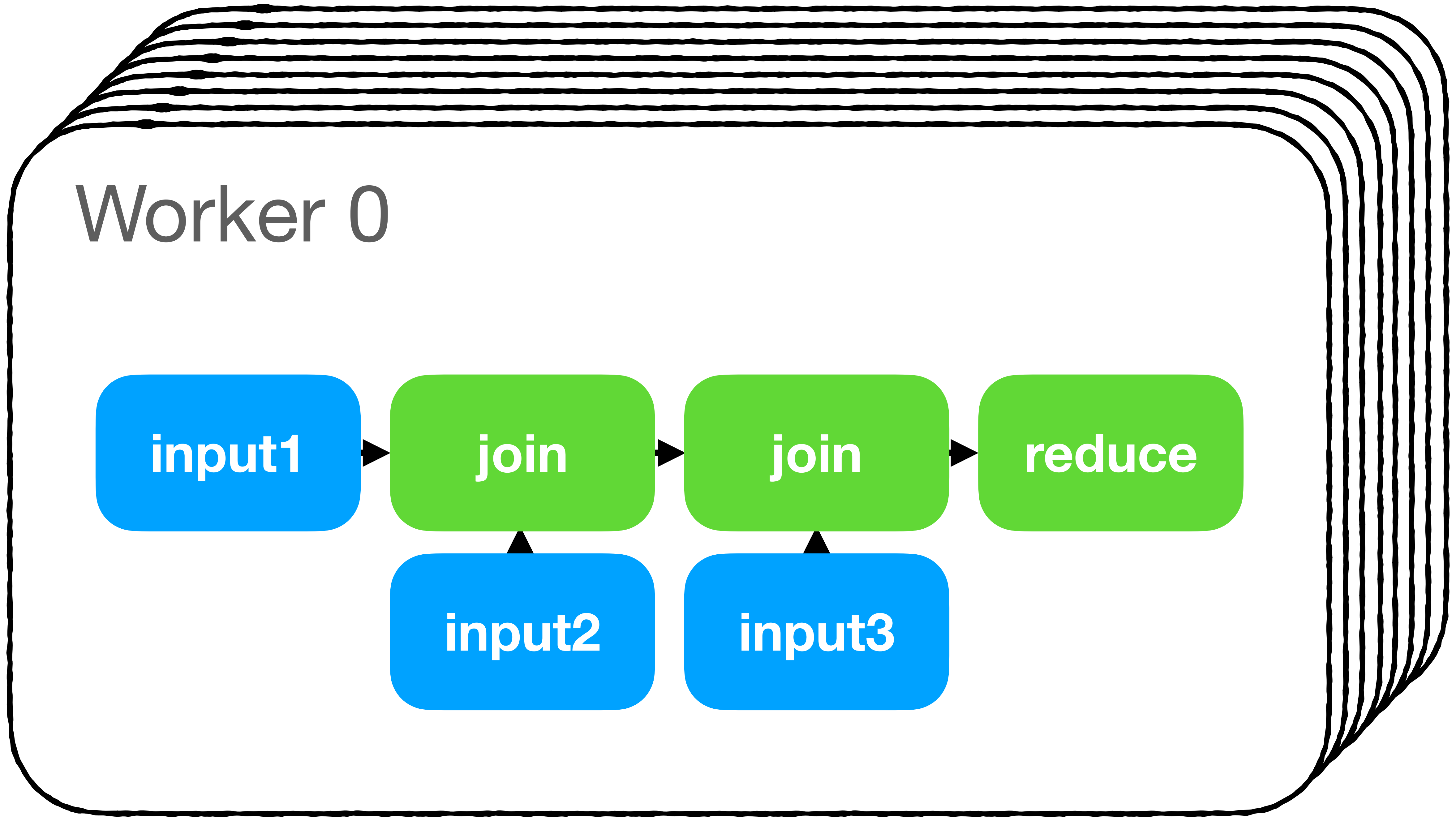
Provides abstractions for

- Fibers (operators)
- Communication (channels)
- Coordination (timestamps)
- Scheduling (cooperative)

Can multiplex millions of operators.

Relevant here

- Operators are sharded over all workers.
- Timestamps may be partially ordered.



Differential Dataflow

IVM for Data-Parallel Computation

Record

L

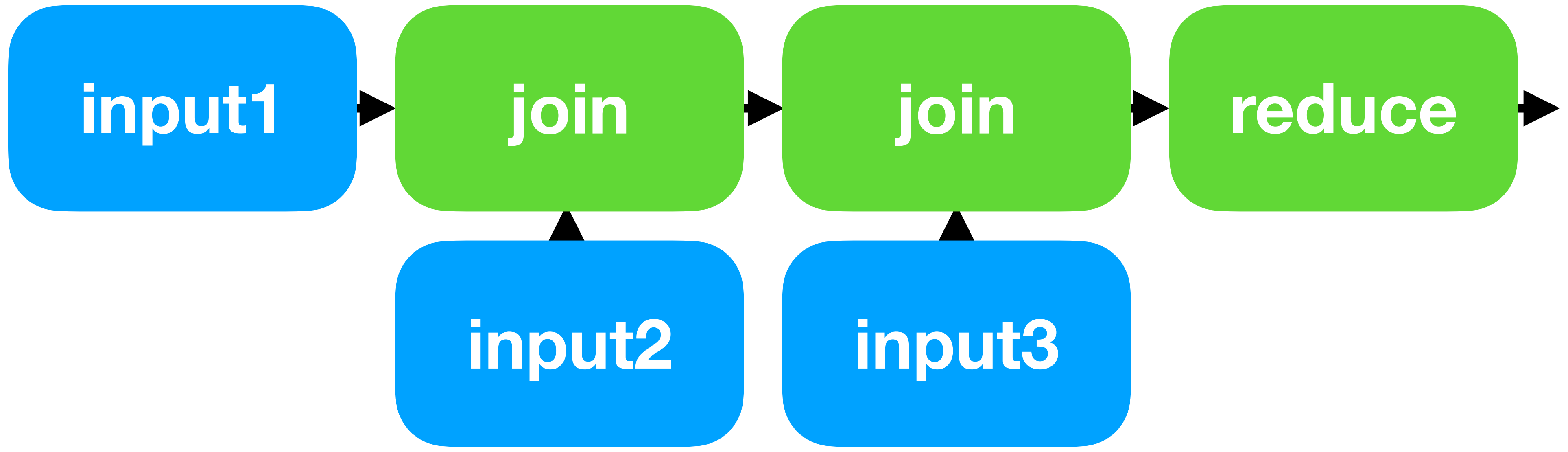
change-data capture

integer

Streams represent CDC info:
records: **`(data,time,diff)`**

Traditional data-parallel operators:
`Map`, **`Filter`**, **`Reduce`**, **`Join`**, **`+`**

Operators maintain as output the correct answer for their operator mapped over the input.



The **Reduce** CDC output accumulates at each time to the correct results for the query on the inputs at that time.

Differential Dataflow

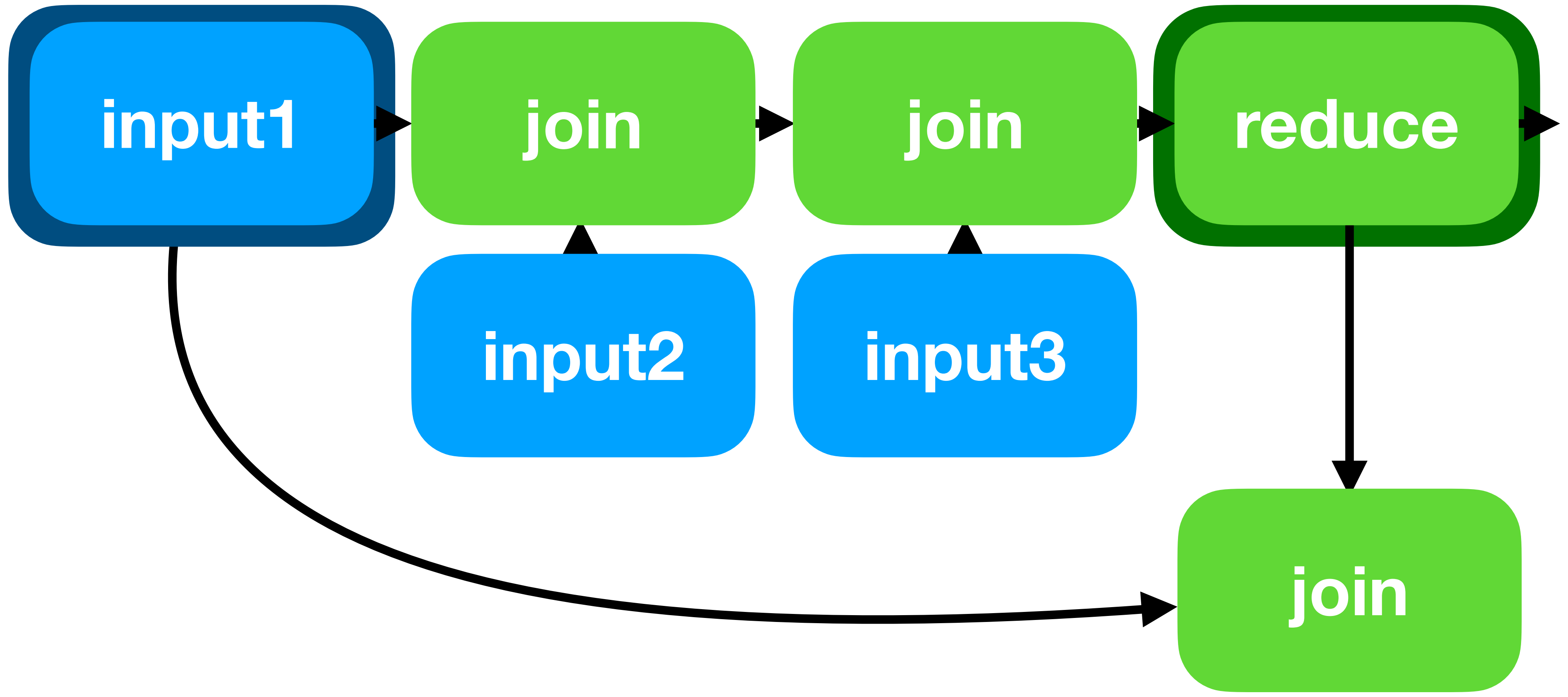
IVM for Data-Parallel Computation

Some non-traditional operators:
  `Iterate`, + mutual recursion
  `Arrange` : “index build”

Arrangements:

very important!

A multi-version index over CDC contents.
Presents as both a stream, and an index.
Allows replay, index sharing.



Their main “verb” is to remove historical distinctions: logical compaction.

Standard SQL on Streaming Data

The basis for modern streaming infrastructure

new!

You can just write SQL against streams of data.
The language isn't new, but what you can do is.

Tasks that required custom streaming systems can now be done in idiomatic SQL fragments.

Ex: the SQL query to the right aggregates data.
It works great when applied to **streams** of data.
Unbounded streams, too big to warehouse.

```sql
-- Aggregations over
-- stream of events
CREATE VIEW bids AS
SELECT
    item,
    hour,
    max(bid)
FROM
    offers
GROUP BY
    item, hour
```

Manage Consistent Caches

Trust SQL to define and *maintain* cached data

new!

Data infrastructure connects components by their function: streams, microservices, caches.

Consistency among them is a continual pain.

Ex: SQL gives you the ability to define compute, index the results, all maintained consistently. Even for streams of changing data.

```
-- Create and cache
-- SQL query results
CREATE VIEW value AS
SELECT
    item,
    hour,
    ...

CREATE INDEX ON value;
```

Windows over Temporal Data

new!

Use SQL to indicate how your data relate to time

Streams of data often focus on recent events.

Stream processors often require "windows", where you only act on time slices of data.

Ex: You can use a WHERE clause in your SQL to relate your data to time. It tells the system when it should introduce and retire your data.

```sql
-- Subset data by time
CREATE VIEW bids AS
SELECT
    o.item,
    o.bid
FROM
    offers o
WHERE
    now() < o.expires;
```

Building Applications

The magic of LATERAL joins

new!

Many users of SQL + streams are building “applications”.

Queries come and go often.
Have bound parameters.

```
-- Respond to queries updates.
CREATE VIEW top_3s AS
SELECT queries.id, name
FROM
   queries,
   LATERAL (
     SELECT name, pop
     FROM cities
     WHERE state = queries.state
     ORDER BY pop
     DESC LIMIT 3
   );
```

Arbitrary correlated subquery
~ Streamed prepared statements.

Features & Challenges

SQL means doing things correctly

All queries need to be dataflow

SQL92 hard stuff: subqueries, order by/limit, case statements

Includes **group by min/max** which get some dataflow magic.

Control-flow interruption is challenging

Run-time errors, exceptions, conditional evaluation.

Optimization is fundamentally different

Execution time isn’t the key metric any more.

Memory footprint, throughput are more important.

Standard SQL is expressive enough for streaming data infrastructure tasks.

Standard SQL is expressive enough for streaming data infrastructure tasks.

Materialize

SQL92 : Postgres/pgwire compatible, read-replica look and feel.

Scalable (from one thread, and up), high-throughput, low-latency.
"Consistency preserving": respect transactions from source data.

https://materialize.com : downloads, docs, demos
https://github.com/materializeinc/materialize/
https://github.com/TimelyDataflow/

mcsherry@materialize.com