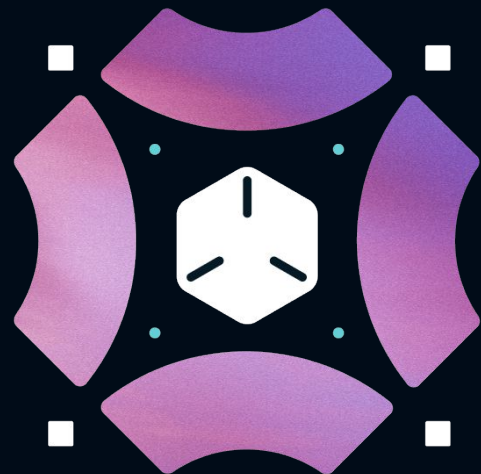


Yaroslav Tkachenko

Founder
Irontools

Lessons Learned: Vectorizing Apache Flink





Yaroslav Tkachenko

Founder
Irontools

- Software Engineer, data/infra
- Technical Lead at Activision, Shopify
- Founding Engineer at Goldsky
- Founder at Irontools

Current:

- Data streaming consulting, advising and training
- ❤️ Flink and DataFusion



Column-based data layout is
more beneficial for data
streaming than **row-based**

Introducing Iron Vector: Apache Flink Accelerator Capable of Reducing Compute Cost by up to 2x

Posted on September 8, 2025 by **Yaroslav Tkachenko**

Iron Vector is a native, columnar, vectorized, high-performance accelerator for Apache Flink SQL and Table API pipelines.

It's easy to install, requires no code changes, and can increase compute efficiency by up to 2x (as of now).

Iron Vector



Vanilla Flink



Flink with Iron Vector accelerator

Columnar Data Layout

SIMD / Vectorization

Values of the same type are stored contiguously in memory.

This allows CPUs to process multiple values with a single SIMD instruction and makes vectorized execution much more efficient.

Better Cache Locality

Columnar data packs relevant values close together in memory.

More data fits into CPU cache lines, sequential access makes hardware prefetching more effective.

Encoding and Data Skipping

Columns compress efficiently with techniques such as dictionary encoding and RLE, reducing memory footprint and bandwidth.

Queries can also read only the columns they need, avoiding unnecessary memory access.



A P A C H E DATAFUSION COMET™

Apache DataFusion Comet

Star Fork

Apache DataFusion Comet is an Apache Spark plugin that uses Apache DataFusion as a native runtime to achieve improvement in terms of query efficiency and query runtime.



Why Sail? Documentation Blog Contact Us

977 Join Community

Where Data Becomes Intelligence

Multimodal distributed framework for batch, streaming, and AI workloads.

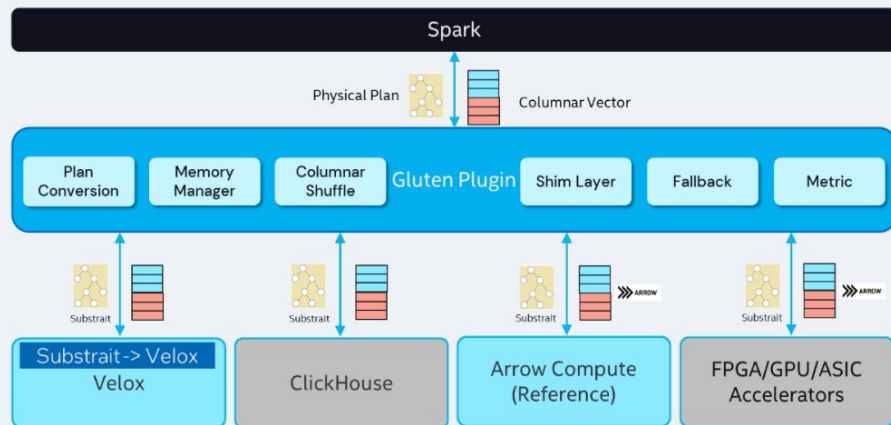
Get Started

Talk to Us

Rethink Spark.
Meet Sail.

A drop-in Apache Spark replacement, reimaged for modern data and AI infrastructure.

Built in Rust, Sail delivers unmatched performance, lower costs, and a familiar Apache Spark interface—all in a unified, cloud-native engine.



Photon

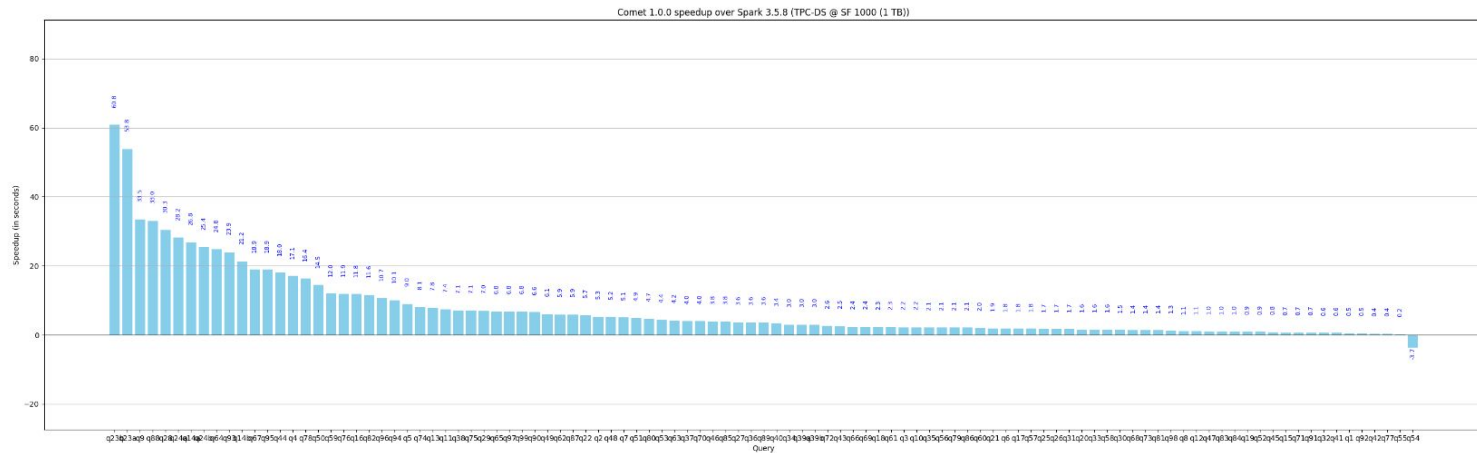
The next generation engine for the Lakehouse

Get started

Learn more



Photon is the next generation engine on the Databricks Lakehouse Platform that provides extremely fast query performance at low cost – from data ingestion, ETL, streaming, data science and interactive queries – directly on your data lake. Photon is compatible with Apache Spark™ APIs, so getting started is as easy as turning it on – no code changes and no lock-in.



Apache DataFusion Comet 1.0.0 TPC-DS
compared with Apache Spark 3.5.x



A P A C H E

DATAFUSION™

Apache DataFusion



Star



Fork

DataFusion is an extensible query engine written in [Rust](#) that uses [Apache Arrow](#) as its in-memory format.

```
use datafusion::prelude::*;

#[tokio::main]
async fn main() -> datafusion::error::Result<()> {
    // register the table
    let ctx = SessionContext::new();
    ctx.register_csv("example", "tests/data/example.csv", CsvReadOptions::new()).await?;

    // create a plan to run a SQL query
    let df = ctx.sql("SELECT a, MIN(b) FROM example WHERE a <= b GROUP BY a LIMIT 100").await?;

    // execute and print results
    df.show().await?;
    Ok(())
}
```



A P A C H E

DATAFUSION™

Apache DataFusion



Star



Fork

DataFusion is an extensible query engine written in [Rust](#) that uses [Apache Arrow](#) as its in-memory format.



Idea: take a **Flink** pipeline
and execute it using
DataFusion in runtime.



How do you even start?



Two Levels of Abstraction

Logical

How a high-level representation is converted to a graph of operations

SELECT ... $\longrightarrow$ Projection ...
FROM ... Scan ...

Physical

How a graph of operations is executed in runtime

SourceOperator, StreamFlatMap, KeyedProcessOperator

SQL

High-level Language

Table API

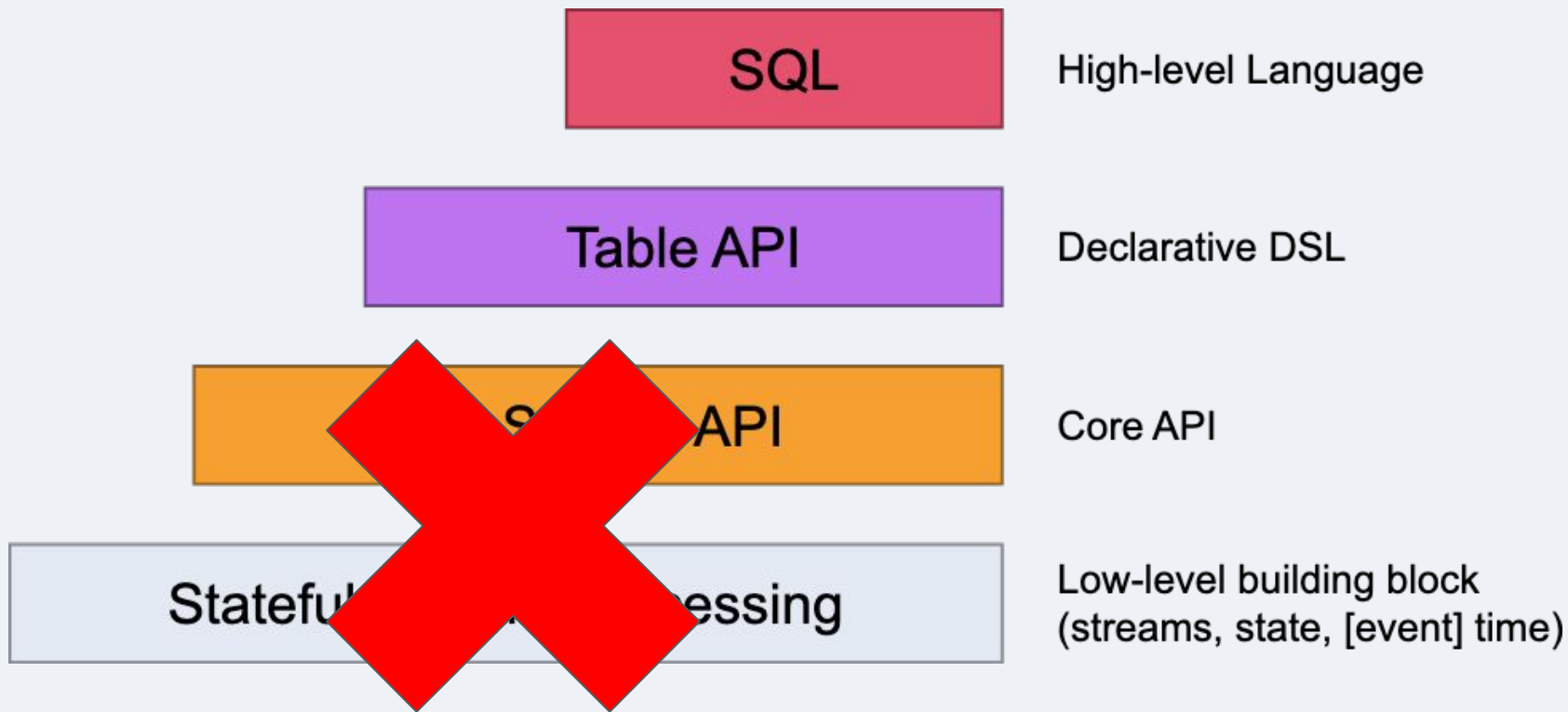
Declarative DSL

DataStream API

Core API

Stateful Stream Processing

Low-level building block
(streams, state, [event] time)



```
SELECT latitude * 2, name FROM KafkaTable WHERE age > 40
```

↓
EXPLAIN

```
== Abstract Syntax Tree ==
LogicalProject(EXPR$0=[*($4, 2)], name=[$1])
+- LogicalFilter(condition=[>($2, 40)])
   +- LogicalTableScan(table=[[default_catalog, default_database, KafkaTable]])

== Optimized Physical Plan ==
Calc(select=[*(latitude, 2) AS EXPR$0, name], where=[>(age, 40)])
+- TableSourceScan(table=[[default_catalog, default_database, KafkaTable]], fields=[id, name, age,
active, latitude, transactionAmount, createdAt, phoneNumber])

== Optimized Execution Plan ==
Calc(select=[(latitude * 2) AS EXPR$0, name], where=[(age > 40)])
+- TableSourceScan(table=[[default_catalog, default_database, KafkaTable]], fields=[id, name, age,
active, latitude, transactionAmount, createdAt, phoneNumber])
```



DataFusion also has logical
and physical plan
abstractions. Can we map
Flink's plan to **DataFusion's**?



Home

[What is Substrait?](#)[How does it work?](#)[Benefits](#)[Example Use Cases](#)

Home

What is Substrait?

Substrait is a format for describing compute operations on structured data. It is designed for interoperability across different languages and systems.

How does it work?

Substrait provides a well-defined, cross-language **specification** for data compute operations. This includes a consistent declaration of common operations, custom operations and one or more serialized representations of this specification. The spec focuses on the semantics of each operation. In addition to the specification the Substrait ecosystem also includes a number of libraries and **useful tools**.

We highly recommend the **tutorial** to learn how a Substrait plan is constructed.



Filter Properties

Property	Description	Required
Input	The relational input.	Required
Expression	A boolean expression which describes which records are included/excluded.	Required

FilterRel Message

```
message FilterRel {  
  RelCommon common = 1;  
  Rel input = 2;  
  Expression condition = 3;  
  substrait.extensions.AdvancedExtension advanced_extension = 10;  
  
}
```

Flink Plan → Substrait → DataFusion Plan

Flink Plan → Substrait → DataFusion Plan

Use Calcite

Use datafusion-substrait

```
Calc(select=[(latitude * 2) AS EXPR$0, name], where=[(age > 40)])  
+- TableSourceScan(table=[[default_catalog, default_database, KafkaTable]], fields=[id, name, age,  
active, latitude, transactionAmount, createdAt, phoneNumber])
```



```
ProjectionExec: expr=[latitude@1 * 2 as KafkaTable.latitude * Int32(2), name@0 as name, __row_kind@2 as __row_kind]  
  VectorFilterExec: age@1 > 40, projection=[name@0, latitude@2, __row_kind@3]  
    RepartitionExec: partitioning=RoundRobinBatch(18), input_partitions=1  
      ChannelSourceExec
```



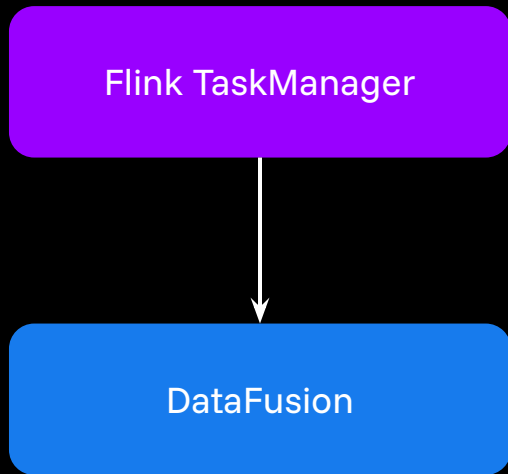
How can we actually execute
this plan? **DataFusion**
operates on columnar (**Arrow**)
data.

Idea #1: Replace the whole runtime

Replace TaskManager's JVM execution with native DataFusion executor

Gotchas:

- Networking layer (shuffles)?
- Checkpointing?
- Failure recovery?
- Connectors?

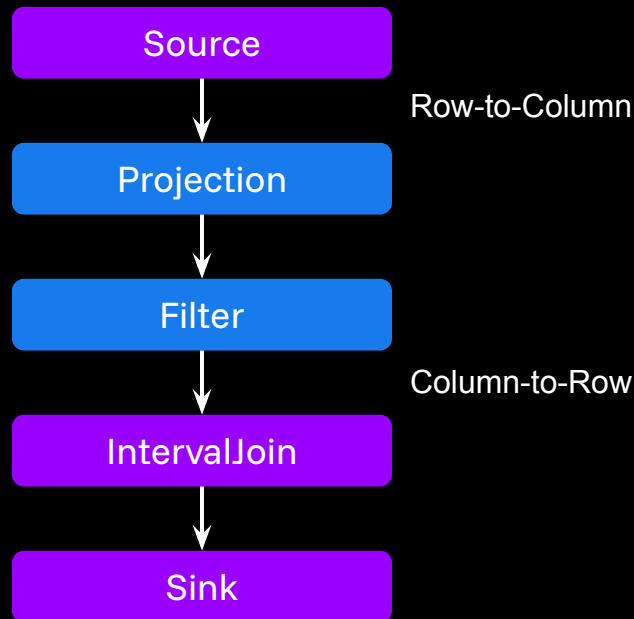


Idea #2: Replace individual operators

Replace some of Flink's operators, keep everything else the same

Gotchas:

- Row-to-Column and Column-to-Row operations are HORRIBLY SLOW.

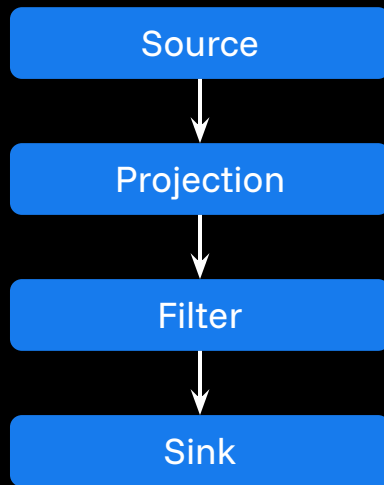


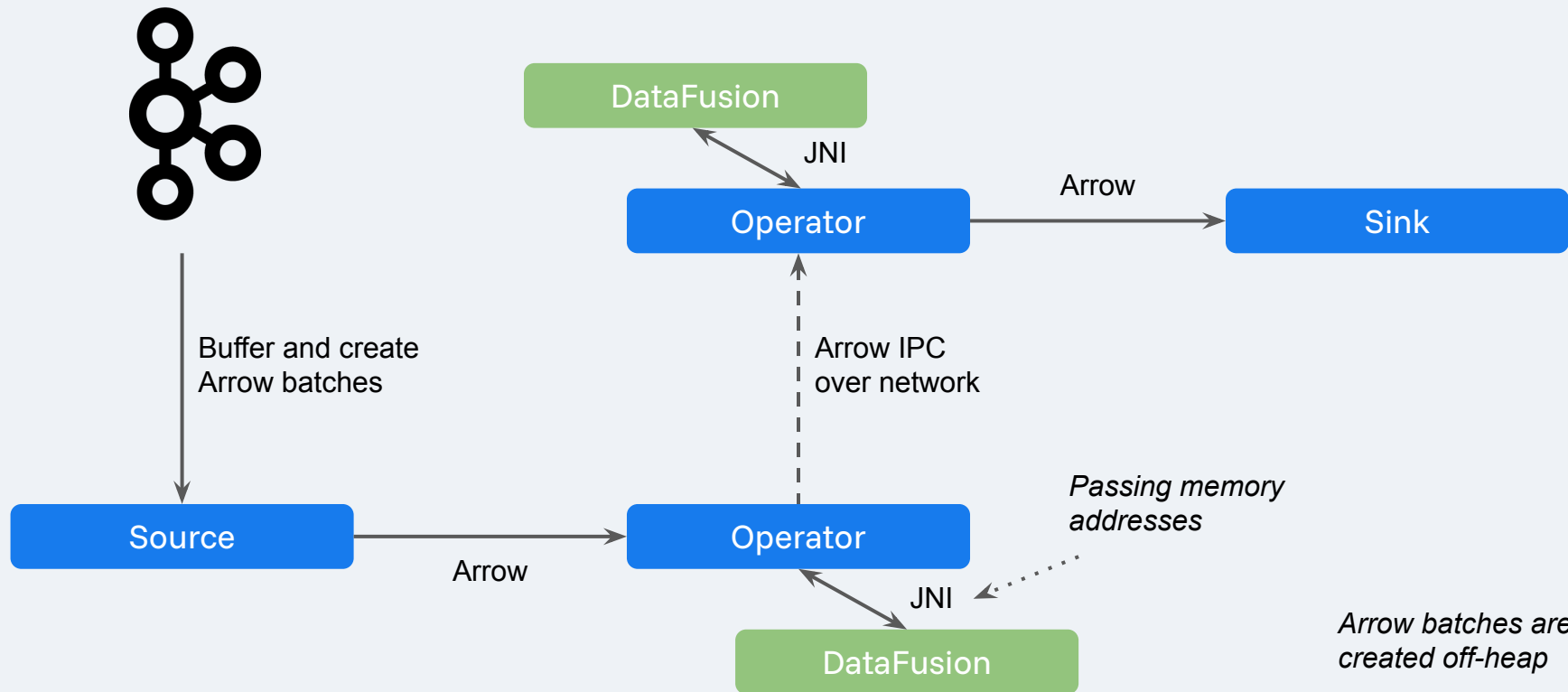
Idea #3: Replace (or not) the whole graph of operators

Replace the full graph of Flink's operators, fallback to Flink if something is not supported

This way support for different operators can be added incrementally.

Still executed within the TaskManager JVM process, relying on existing primitives like networking.





```

public static native int executeQuery(
    String queryId, long[] dataAddresses, long[] schemaAddresses);

public VectorSchemaRoot execute(VectorSchemaRoot batch) {
    NativeDataExchange.BatchArrowArrays batchArrowArrays =
        NativeDataExchange.allocateArrowStructs(batch.getFieldVectors().size());

    try {
        NativeDataExchange.BatchMemoryAddresses batchMemoryAddresses =
            NativeDataExchange.exportBatch(batchArrowArrays, batch);

        int outputCols =
            NativeRunner.executeQuery(
                queryId,
                batchMemoryAddresses
                    .getDataAddresses(), // These will be overwritten by Rust with output addresses
                batchMemoryAddresses.getSchemaAddresses());

        // Now batchMemoryAddresses contains the OUTPUT addresses from Rust
        return NativeDataExchange.importBatch(batchMemoryAddresses, outputCols);
    } finally {
        batchArrowArrays.close();
    }
}

```



DataFusion is a batch query engine. How does it support streaming data?

```
pub trait RecordBatchStream: Stream<Item = Result<RecordBatch>> {  
    fn schema(&self) -> SchemaRef;  
}  
  
pub type SendableRecordBatchStream = Pin<Box<dyn RecordBatchStream + Send>>;
```

```
impl ExecutionPlan for StreamingSourceExec {  
    fn properties(&self) -> &PlanProperties {  
        Boundedness::Unbounded { requires_infinite_memory: false }  
        EmissionType::Incremental  
    }  
    fn execute(&self, /*...*/) -> Result<SendableRecordBatchStream> {  
        // returns Stream<RecordBatch> that yields forever.  
    }  
}
```

Some DataFusion Gotchas

Batches Must Flow

- Some built-in operators don't emit empty batches.
- So queries that emit empty results are stuck.
- Created a customized version and promoted it with an optimizer rule.

Stable Functions

- Stable functions (e.g. `now()`) cache their output per query.
- Doesn't make sense in the streaming context.
- Created customized Volatile functions, promoted them with an override.

Iron Vector didn't get much traction and I didn't have enough time to properly open source it...





Iron Vector didn't get much traction and I didn't have enough time to properly open source it...

But I learned a lot!



Iron Vector didn't get much traction and I didn't have enough time to properly open source it...

But I learned a lot!

TALK • 30 mins

Streamling:
Lightweight, Extensible
Data Streaming...

SPEAKERS

[\[Goldsby\]](#)

Jeff Ling

2:05pm

Spiritual successor: StreamFusion

- <https://streamfusion.tech>
- <https://github.com/datafusion-contrib/StreamFusion>



Questions?





Thank you

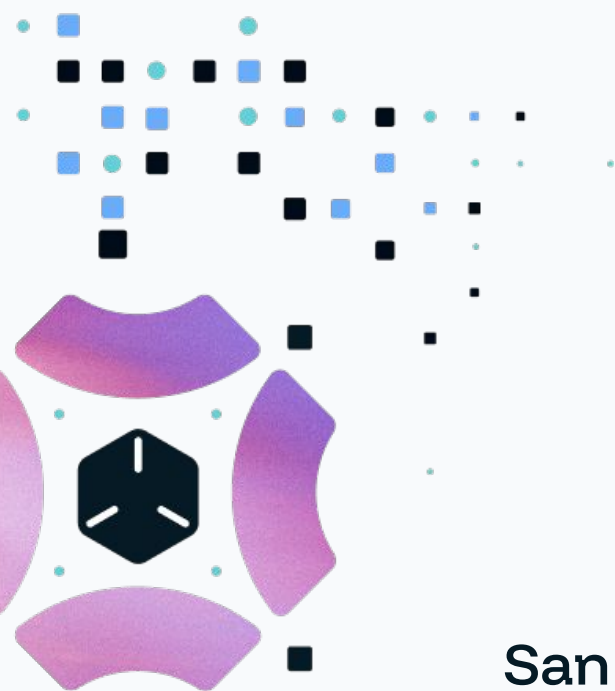
Yaroslav Tkachenko

Founder
Irontools



Get in touch!
sap1ens.com





Hosted by  StreamNative

DSS '26

San Francisco, CA | October 7 - 8

