Arrow-Backed Streaming Dataframes in Timely Dataflow

Bridging Efficient Columnar Storage with Low-Latency Streaming

Florian Klein

University of Cambridge

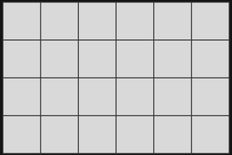
December 1, 2025

+

Motivation: The Divide

The "Batch" World

- Tools: DuckDB, Polars, Pandas
- **Format:** Apache Arrow (Columnar)
- High Throughput, SIMD
- Static Data, High Latency



Static Table

The "Streaming" World

- Tools: Flink, Timely Dataflow
- **Format:** Row / Event-based
- Low Latency, Iterative
- Row overhead, No Standard

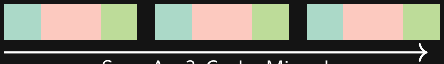
Infinite Stream

Background: Apache Arrow

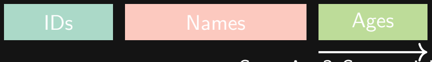
Why Arrow?

- De-facto standard for in-memory analytics.
- **Columnar Layout:**
  - Data of the same type is contiguous.
  - CPU Cache efficient.
  - Enables SIMD vectorization.
- Zero-copy reads.

Row-Based Memory (e.g., CSV, Objects)



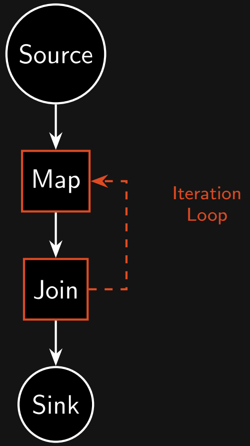
Columnar Memory (Arrow)



Background: Timely Dataflow

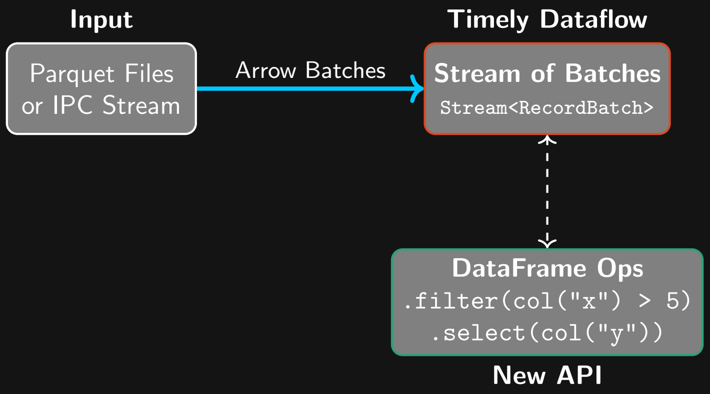
Timely Dataflow (Rust)

- Distributed data-parallel compute engine.
- Based on the **Naiad** system.
- **Key Feature:** *Cyclic* dataflow.
  - Supports loops (Iteration).
  - Essential for Graph algorithms (PageRank) and ML.
- Tracks progress with logical timestamps.



The Solution: Arrow-Backed Streaming

Objective: Bring Arrow's efficiency to Timely's streaming power.



- **Core Idea:** Instead of streaming single events, stream **micro-batches** of Arrow columns.
- **Benefit:** Amortize overhead, enable vectorized processing inside streaming operators.

Implementation: Minimal API

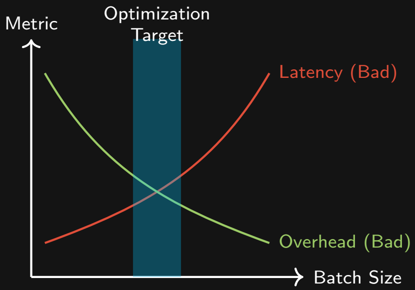
Goal: A fluent, DataFrame-like API on top of Timely scopes.

```rust
// 1. Define the Schema
let schema = Schema::new(vec![
    Field::new("id", DataType::Int32, false),
    Field::new("val", DataType::Float64, false),
]);

timely::execute_from_args(std::env::args(), move |worker| {
    worker.dataflow(|scope| {
        // 2. Create Arrow Source
        let stream = scope.arrow_source("data.parquet", schema.clone());

        // 3. Apply DataFrame transformations
        stream
            .filter(col("val").gt(lit(10.0)))  // Vectorized Filter
            .select(vec![col("id")])           // Columnar Projection
            .inspect(|batch| println!("Row count: {}\n", batch.num_rows()));
    });
}).unwrap();
```

Technical Challenges

The Core Tension

Throughput vs. Latency: Arrow thrives on large batches (SIMD), but streaming requires low latency (small batches). Finding the "sweet spot" size is critical.

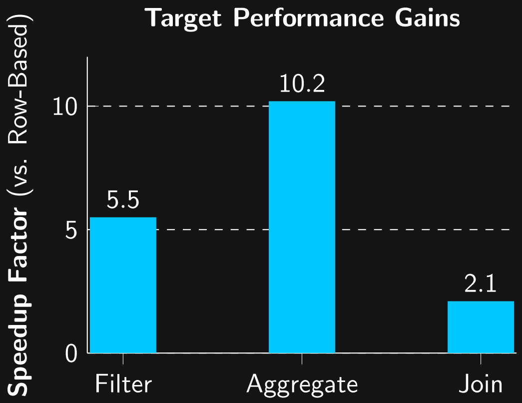


Key Obstacles:

- **Memory Management:** Integrating Arrow's reference-counted buffers (`Arc<Array>`) with Timely's ownership model.
- **Serialization:** Efficiently moving Arrow batches between workers without expensive copies.

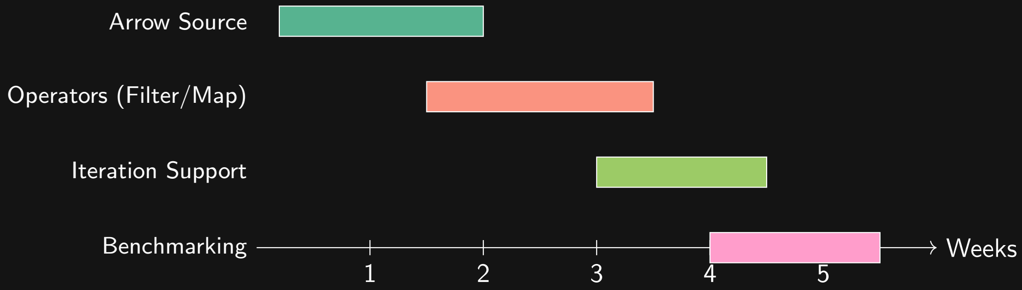
Evaluation Plan

1. **Functionality**
   - Implement common operators: `Filter`, `Select`, `Map`.
   - Demonstrate an iterative algorithm (e.g., BFS) using DataFrames.
2. **Performance (Throughput)**
   - Compare **Row-based Timely** (Standard) vs. **Arrow-Timely** (Ours).
   - Hypothesis: Arrow version will have higher throughput due to cache locality and fewer allocations.



Estimated gains based on SIMD & cache locality benefits.

Project Timeline



Conclusion

"Arrow-Backed Streaming Dataframes"

- **Problem:** Current streaming systems miss out on columnar performance optimizations.
- **Solution:** Native Arrow integration in Timely Dataflow.
- **Impact:**
  - Faster real-time analytics.
  - Unified data representation (Batch $\leftrightarrow$ Stream).
  - Enabler for complex iterative algorithms on DataFrames.

Thank you! Questions?