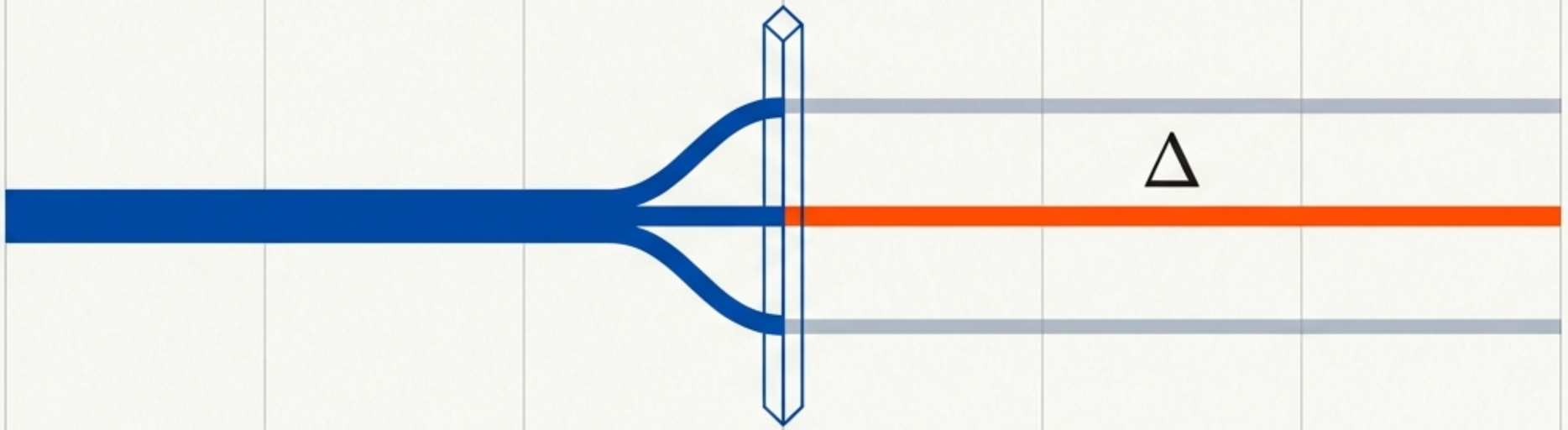
PandaJS Architecture: A Deep Dive

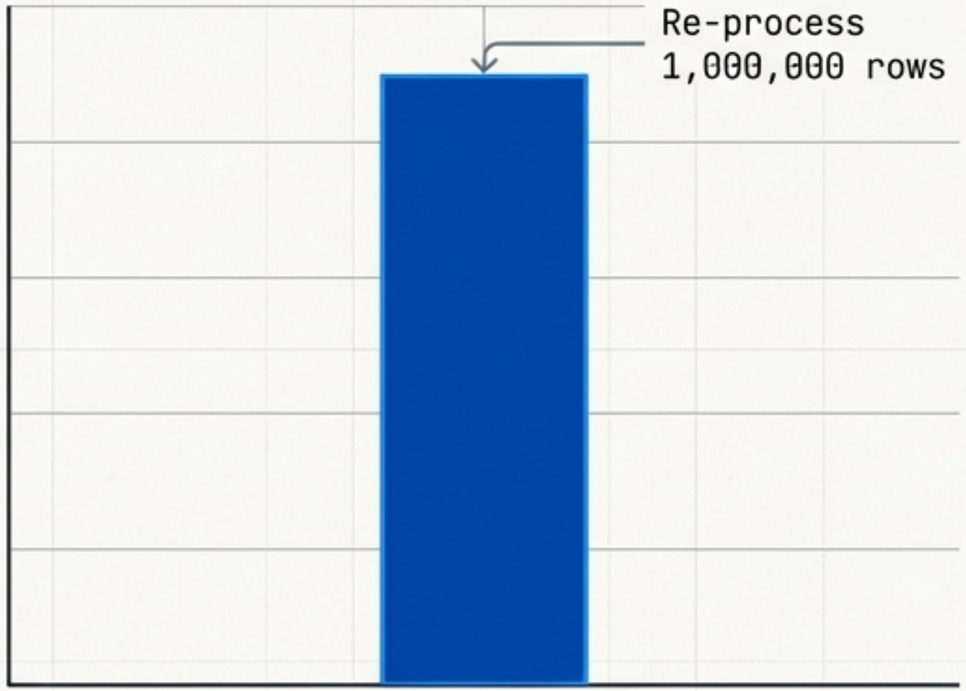


The Pandas-compatible, incrementally-updating DataFrame library for JavaScript.

Implementation based on 'DBSP: Automatic Incremental View Maintenance' (Budiu et al., 2023).

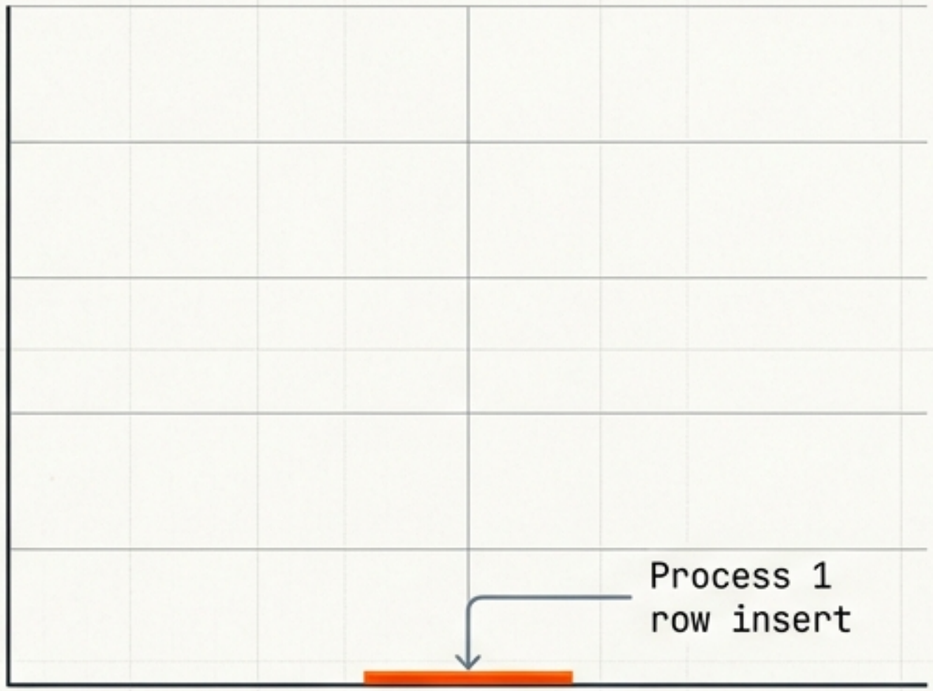
The Core Insight: O(n) vs. O(Δ)

Traditional Model (Recompute All)



Cost = O(n)

PandaJS Model (Process Delta)

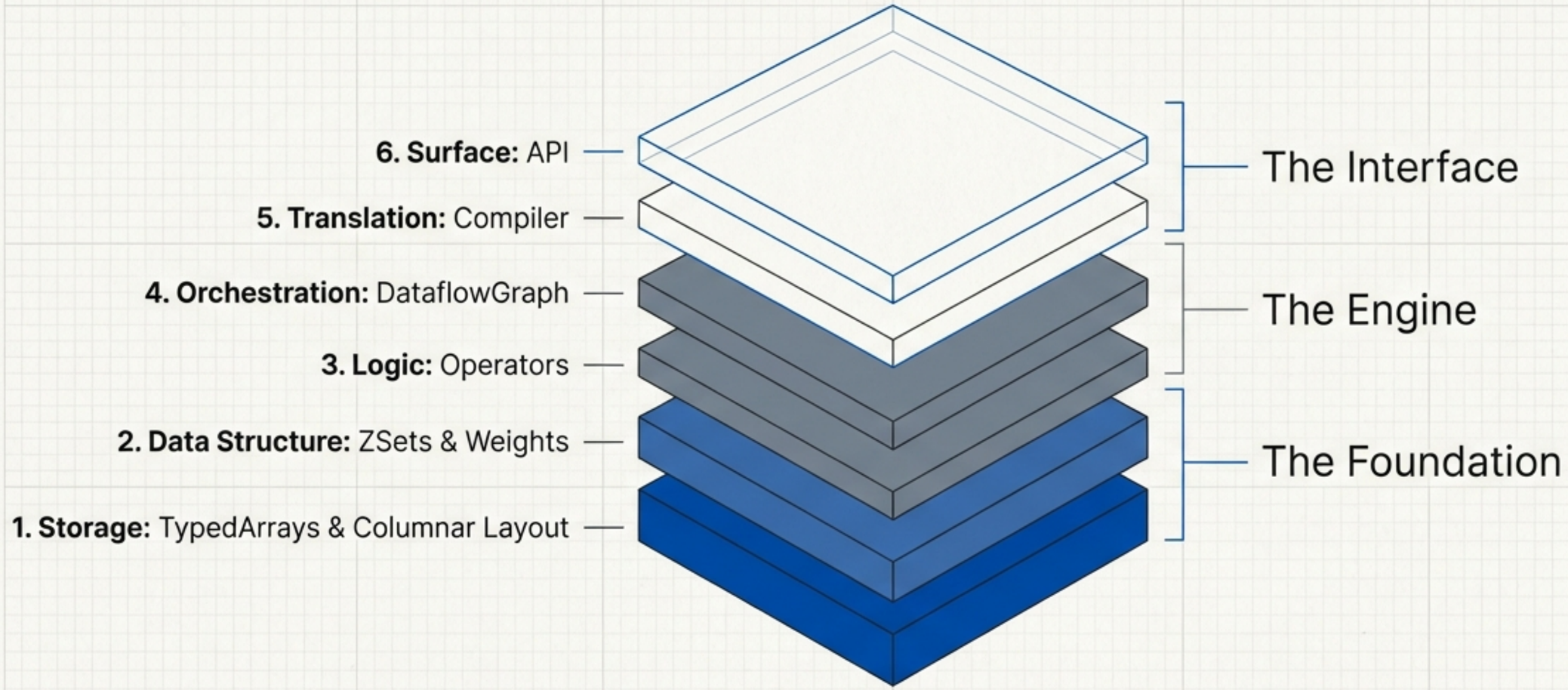


Cost = O(Δ)

Updates are instant, not recomputed. While traditional libraries scale with dataset size (n), PandaJS scales with the size of the change (Δ).

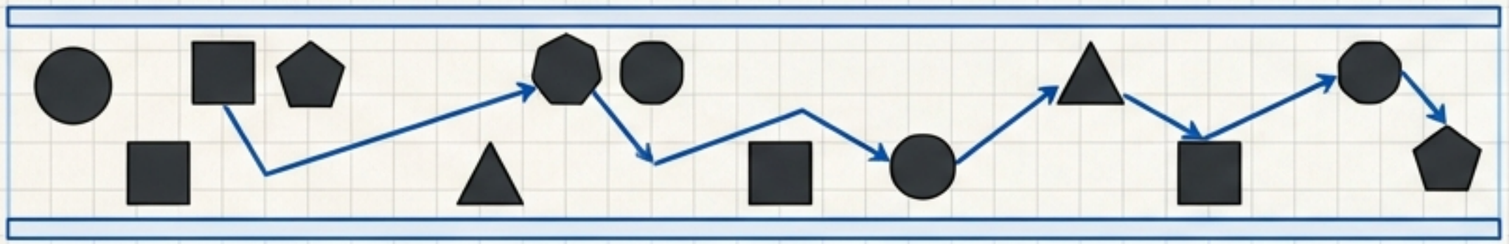
The Layered Architecture

Six layers bridging memory management and query semantics.



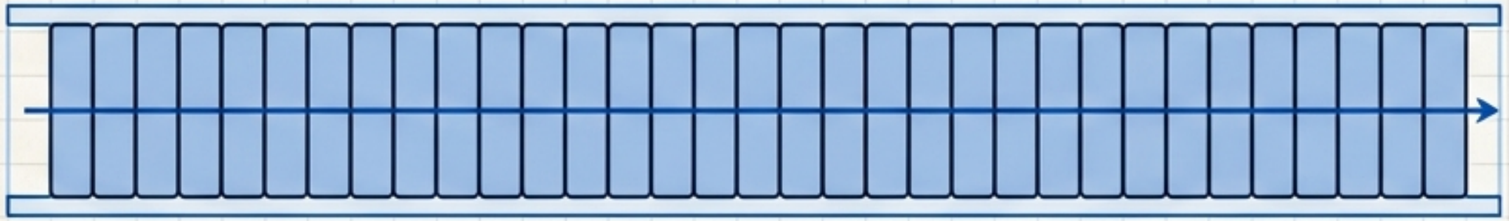
Columnar Storage & Cache Locality

Row-Oriented (Traditional JS Objects)



Pointer Chasing / Poor Cache Locality

Columnar (PandaJS TypedArrays)



Contiguous Block / SIMD Optimized

Key Benefits

1. **Cache Locality:** CPU pre-fetches contiguous blocks.
2. **SIMD Potential:** Parallel vector processing.
3. **Zero Overhead:** No per-element object wrappers.

Strict Typing via TypedArrays

Logical Type	Physical Implementation	Byte Width	Notes
`float64`	`Float64Array`	8 bytes	Standard double precision
`int32`	`Int32Array`	4 bytes	Signed integers
`bool`	`Uint8Array`	1 byte	Bit-packed logic
`string`	`Int32Array`	4 bytes	Dictionary Encoded references

Memory Footprint (1M Rows)

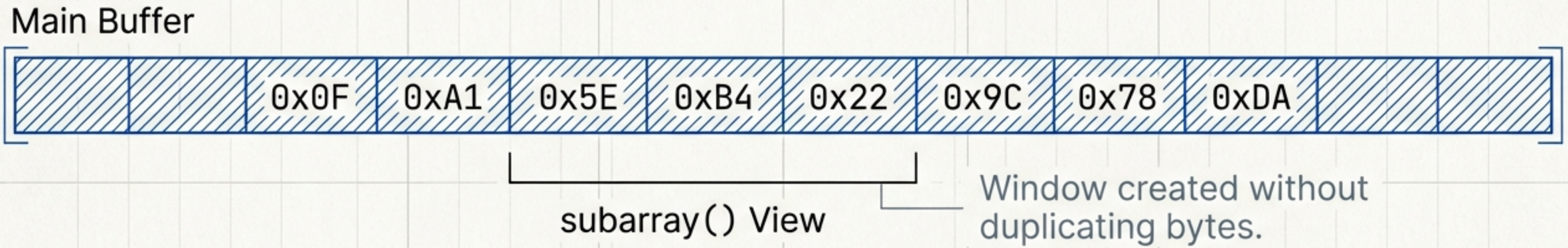
Row-Oriented:
~400 MB

PandaJS: ~40 MB

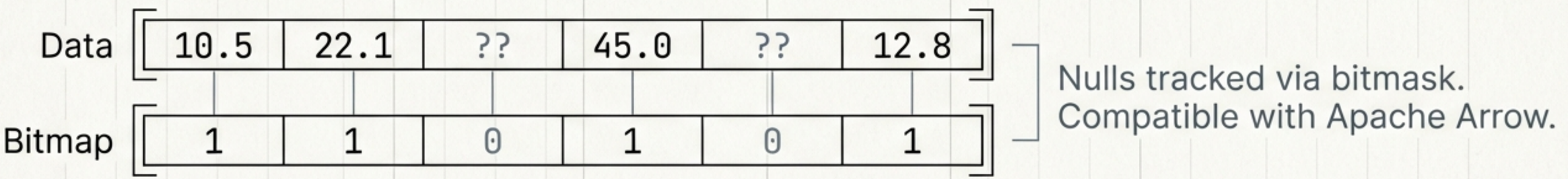
10x Density Improvement

Zero-Copy Mechanics

Views vs. Copies

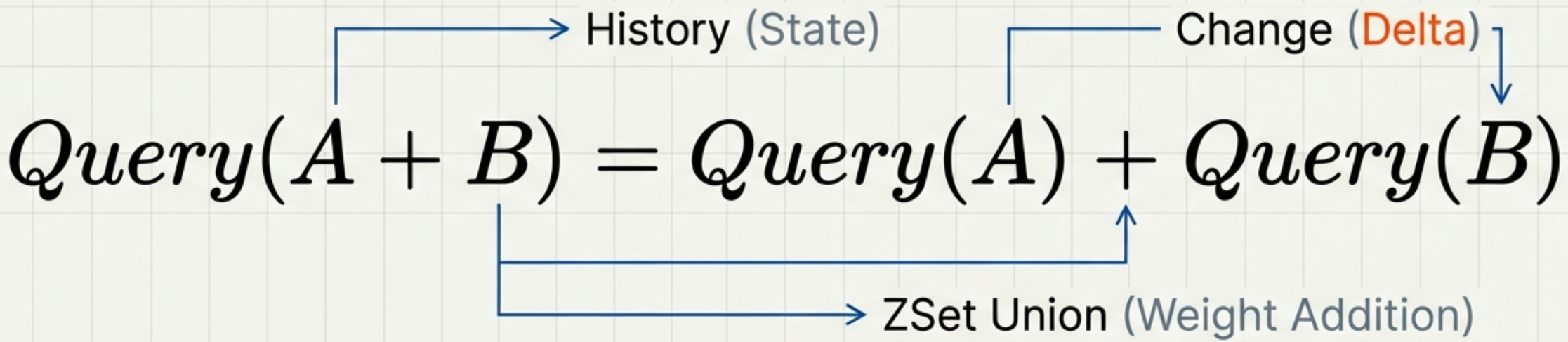


Validity Bitmaps (Null Handling)



The Theoretical Core: DBSP

Database Stream Processing (Budiu et al., 2023)



The Axiom: Most relational operations distribute over addition. This property allows us to process the change (B) without re-scanning the history (A).

The ZSet & Weights

Inter Tight: Redefining the database row.

ID	Value	Weight(w)	
001	Apple	+1	(Insertion / Present)
002	Banana	-1	(Deletion / Negation)
003	Cherry	+2	(Multiset / 2 copies)

Deletion Logic:
Helvetica Now Display

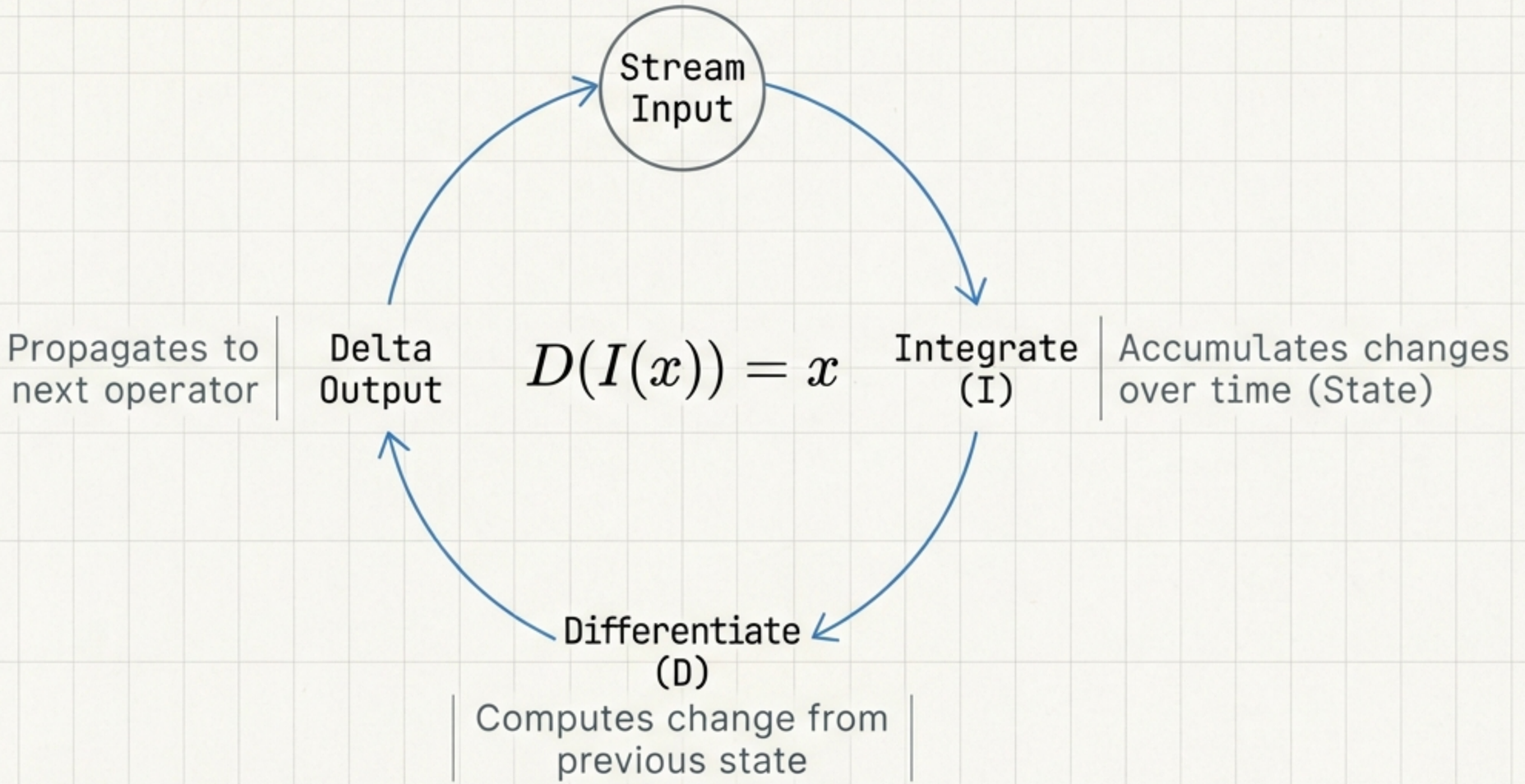
To delete ‘Banana’, we do NOT search and destroy.

We ADD a new row:

```
{
   ID: 002,
   Value: Banana,
   Weight: -1
}
```

Result: Sum of weights = 0. The row vanishes from query results.

The Differentiate / Integrate Pattern



Operators use this cycle to maintain internal state. We never hold the full history in memory if only the aggregate is needed.

The Execution Pipeline

Lazy Evaluation

Nothing executes until `collect()` is called.
Optimization happens before a single byte moves.

Milestone 1

Query Plan Building

API calls (filter, groupBy).
No execution yet.

```
db.collection("data").filter(x
=> x.val > 10).groupBy("category")
```

Milestone 2

Compilation

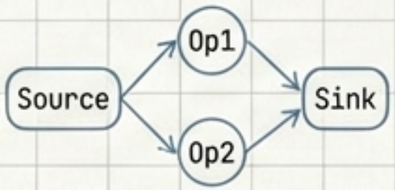
Plan transforms into
Operator Chain.

```
[Filter]→[GroupBy]→[Aggregate]
```

Milestone 3

DataflowGraph

Graph orchestrates
the topology.



Milestone 4

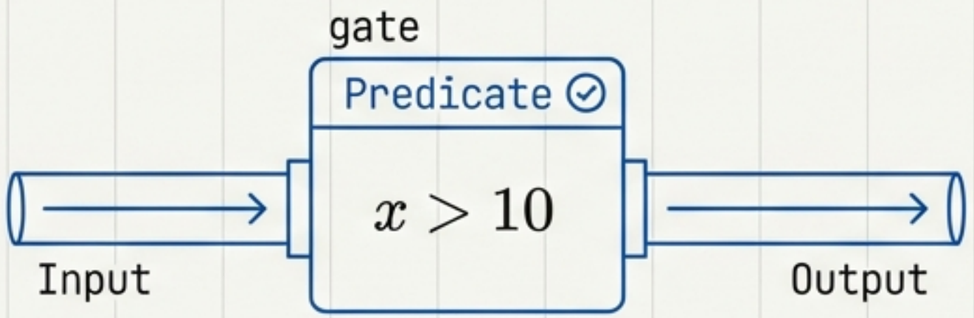
Delta Propagation

`insert()` triggers
the flow.

```
db.collection("data").insert
({val: 15, category: "A"})
```

Operator Patterns

Stateless (e.g., Filter)

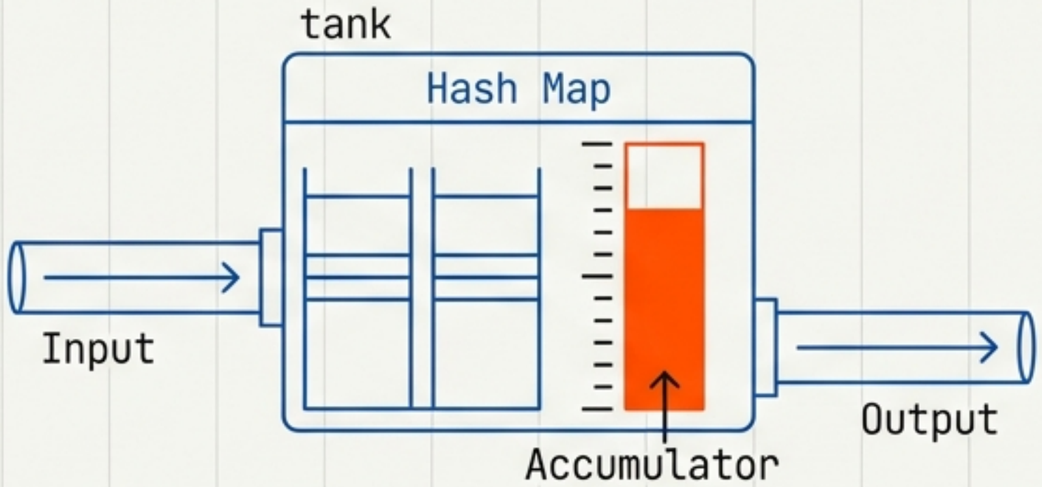


Processes each row independently.

Compiled fast paths.

Complexity: O(1) per row

Stateful (e.g., GroupBy)



Maintains Accumulators for each group.

Internal Hash Map.

Complexity: Required for aggregation.

Incremental Aggregation Math

Sum / Count

New Sum

= Old Sum

+ (Value * Weight)

Complexity: O(1)

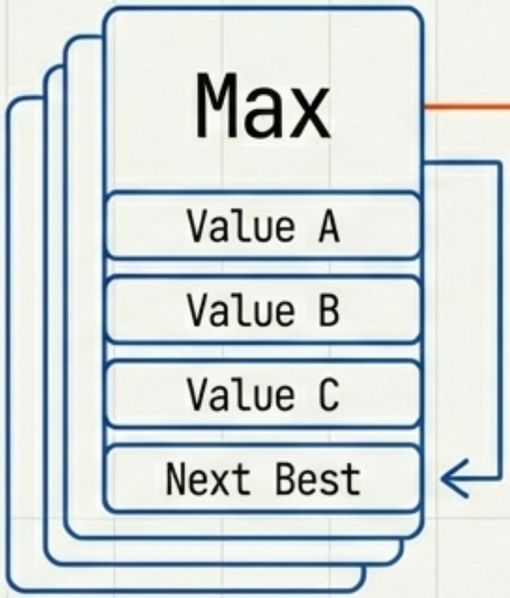
Mean

Track (Sum, Count) separately.

Derive Mean on demand.

Complexity: O(1)

Min / Max (The Hard Case)



If Max is deleted (weight -1)...

...we must pop the stack to find the Next Best value.

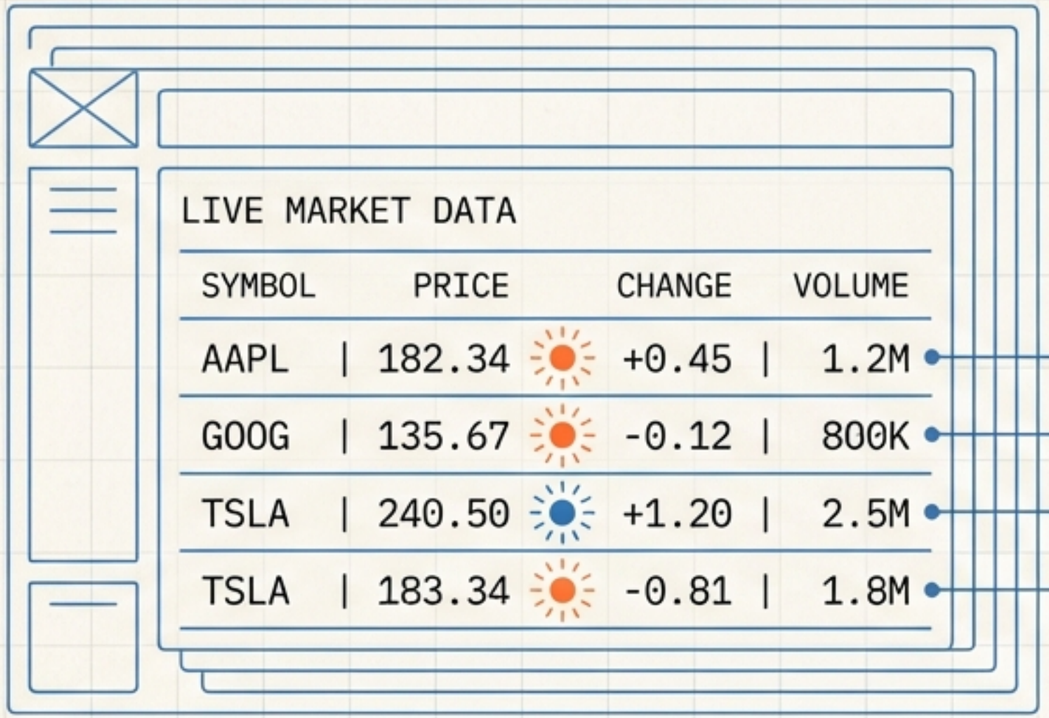
Complexity: O(k) where k is distinct values.

Algorithmic Complexity Analysis

Operation	Traditional Library	PandaJS Incremental
Add 1 row to 1M	1,000,000 ops	1 op
Update Join	N x M	O(Δ affected rows)
Filter Change	O(n) re-apply	O(Δ) process delta

“Scalability is determined by the volatility of your data, not the volume of your history.”

Real-World Example: Trading Dashboard



1M historical trades. 1000 symbols.
Updates every 100ms.

Performance Profile

Insert Latency: < 1µs

GroupBy Propagation: ~10µs

Re-sort Top 10: ~1µs

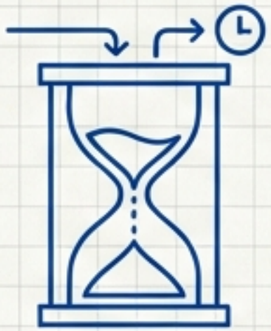
2500x Faster

Total Latency: < 20µs per update

vs Traditional: ~50,000µs (50ms)

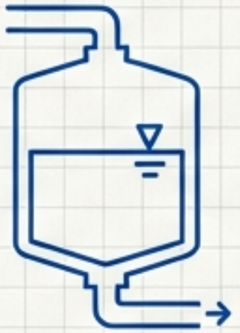
Core Principles Recap

Columnar Storage



Lazy Evaluation

Incremental Computation



Stateful Operators

DBSP Semantics

PandaJS: A library that feels like Pandas but performs like a streaming database engine.