Implement, Integrate & Extend a Query Engine in

Ruihang Xia
GreptimeDB

About Me

- From Greptime since 2022
- Previously worked on ANT GROUP
- PMC member of APACHE DATAFUSION™

Outline

- Days before DataFusion
- About GreptimeDB
- Extensible Query Engine

#1 Before DataFusion

- Homemade Query Engine
- Unstable latency, underperformant
- Hard to maintain & develop with small team
- Tightly binded with storage layer and business
- No general query features and operators

> *If we had this available when we built the Flux language for @InfluxDB 2.0, it would have saved us years of effort. And it would have given all the users that saw Flux as a barrier to adoption a familiar option. We **never had the chance to iterate** on Flux developer experience and the language itself **because of all the other stuff**.*

Paul Dix ✔
@pauldix

#1 With DataFusion

- Is things changed a lots? (yes)
- Out-of-the-box tools
- DataFusion and its friends

#1 Integrate DataFusion

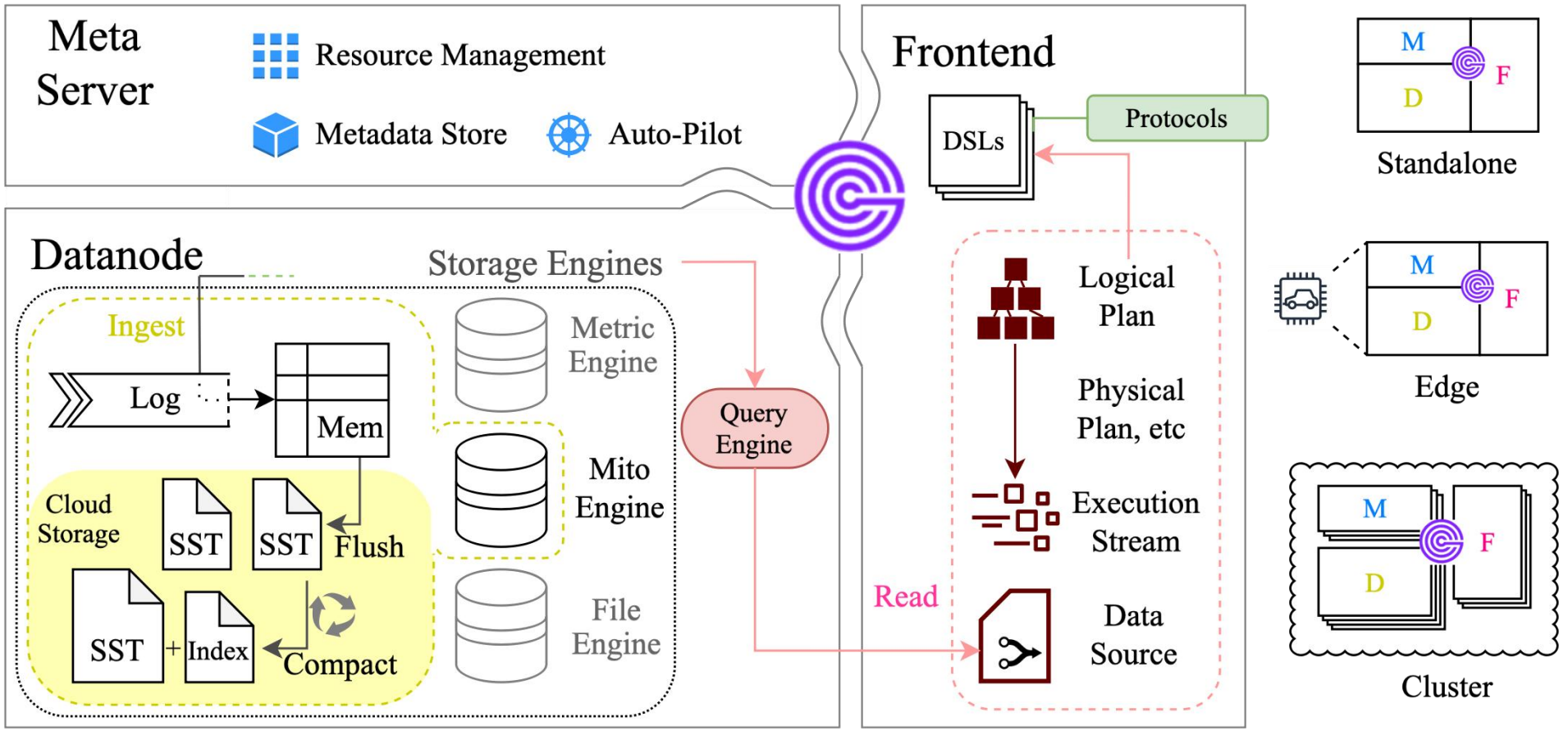
- Data + Expr  Result
- Only **`SELECT`** is processed by DataFusion in GreptimeDB
- Parser is bypassed as well (
  - Build our own parser on top of sqlparser
  - For extending grammar and including specific features
- Ahh, and the data type system (
  - hard to do that
  - We have a different set of supported types
  - Make some convenient utils

#2 About GreptimeDB

- Yet another time-series database
- Has integrated many open source projects
  - a.k.a. database building blocks
- Supports many query languages
  - SQLs, PromQL, query string etc.
- Has many storage engines
  - For generic, for metrics, for external files etc.
- Can scale from edge to cloud
  - car, IoT, homelab, NAS, cloud...
  - with one binary & codebase

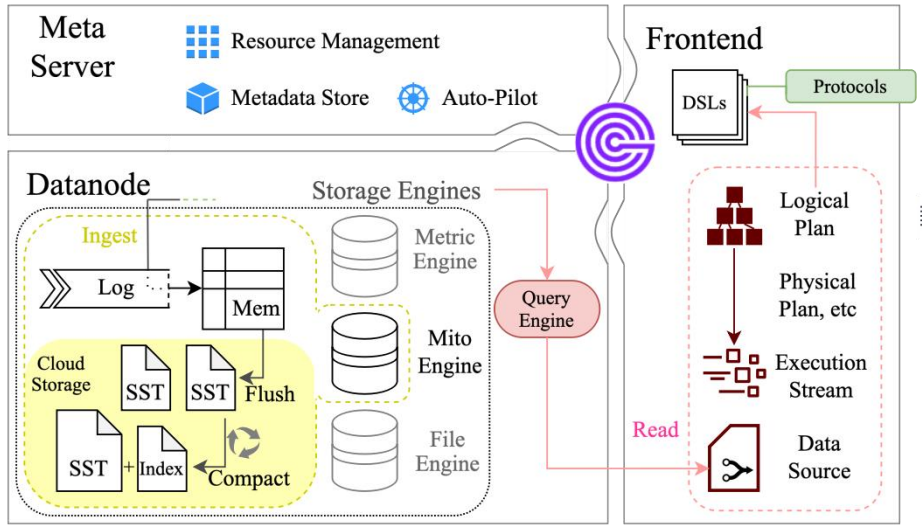
#2 Time-Series Database

- Three major components



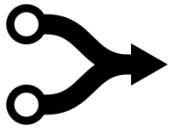
#2 Time-Series Database

- Metasrv
  - Like other traditional distributed databases
- Frontend
  - Protocols
  - Query Engine
  - Stateless
- Datanode
  - All storage stuff
  - The same query engine
  - LSM-Tree



#2 So Many Logos

- Proudly powered by



Substrait

AWS S3

PyO3

#2 One-for-All Storage

- Name ≥ 2 kinds of ″time-series″ data you have heard before
  - *Metric, Trace, Log, Profile* and more in the future
- And what do you need to do if you want to use them
- What if you want to store them for years
- Do we really need to distinguish between ″this″ and ″that″

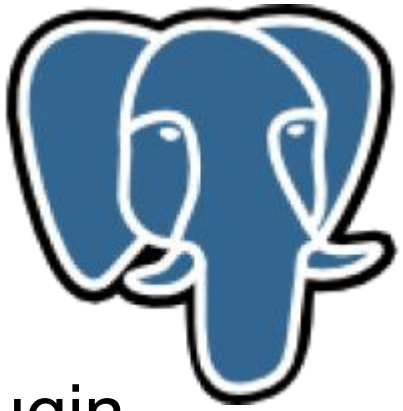
#3 Extend It!

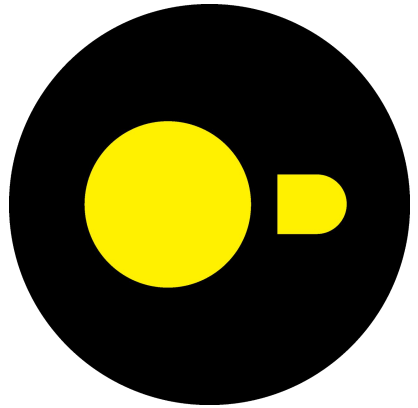
- Where to start
- Extend for what
  - Features
  - Performance
  - Scale
  - Type
- How to do it
- Optimizations

#3 Almost every places

- Parser
- Planning
- Executing
- Storage
- ...

- Some differences compared to

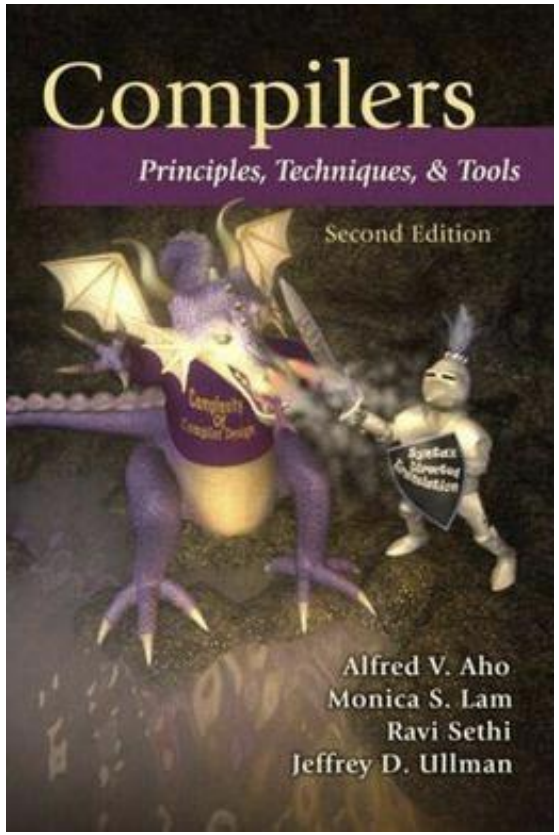




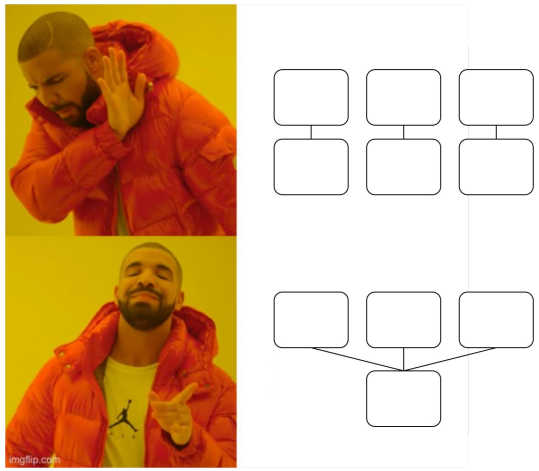
  - lib vs. .so
  - Build a system vs. System plugin
  - Languages

#3 Extend for Features

- SQL isn't famous for time-series guys
- PromQL, LogQL, LogsQL, TraceQL...
- Some don't even have a language
- Separated language, separated capability

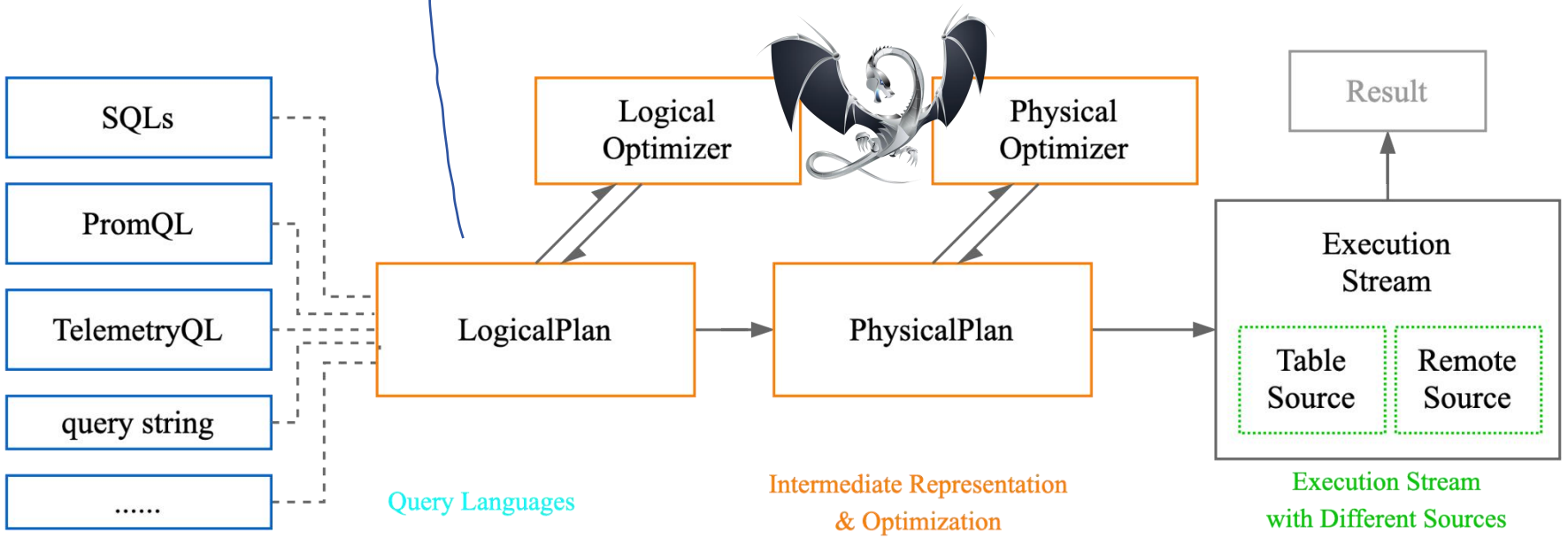


- But we can support all (technically) of them



#3 Chaotic Languages

- are combined together
- Use LogicalPlan as IR



#3 Intermediate Representation

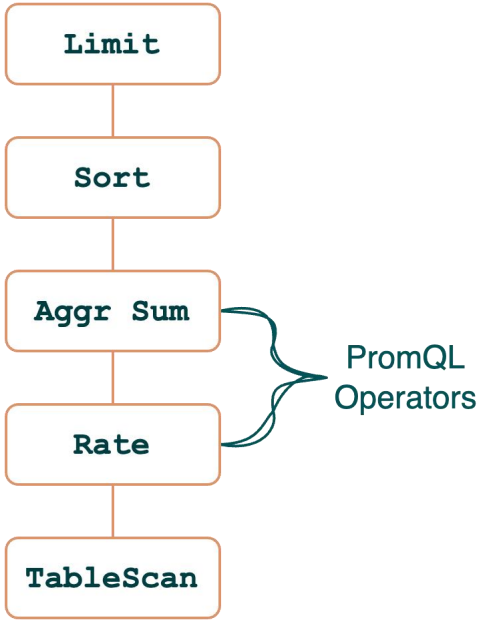
- All languages are the same in IR
- E.g.: combine SQL and PromQL

CTE

```
WITH prom_result AS (
    TQL EVAL (0, 100, '10s')
    sum(rate(http_requests_total[5m])) BY (job)
)
SELECT * FROM prom_result
ORDER BY sum_rate_http_requests_total_5m
LIMIT 10
```

PromQL

SQL



#3 E.g.: make a tiny lang

- Search Query String
- Has:
  - Binary Op `+` and `-`
  - Parenthesis
  - Word
- `over -(fox jumps)`
- "Compile" to `LIKE` tree
- Source code

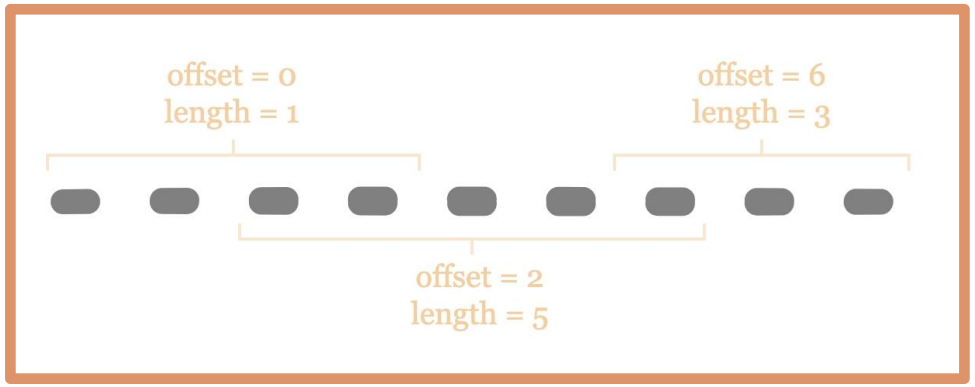
```
PatternAst::Binary {
    op: BinaryOp::Or,
    children: vec![
        PatternAst::Literal {
            op: UnaryOp::Optional,
            pattern: "a".to_string(),
        },
        PatternAst::Literal {
            op: UnaryOp::Optional,
            pattern: "b".to_string(),
        },
        PatternAst::Binary {
            op: BinaryOp::And,
            children: vec![
                PatternAst::Literal {
                    op: UnaryOp::Optional,
                    pattern: "c".to_string(),
                },
                PatternAst::Literal {
                    op: UnaryOp::Optional,
                    pattern: "d".to_string(),
                },
            ],
        },
    ],
},
```

#3 Optimizations

- DataFusion, as well as other “building blocks” doesn’t guarantee high-performance for every use-cases
- They are some kinds of framework, you need to take care of your own parts
- `rate()` function (UDF)
  - Requires input is `ORDER BY timestamp DESC`
  - Need to tell optimizer about this
  - Avoid unnecessary ordering and repartition etc.

#3 Optimizations (cont.)

- PromQL UDF need a dedicated container for intermediate data type
- The way to iterate data is special:
  - Matrix instead of Array
- Abuse Arrow's dictionary array (oops)
  - Key: offset + length



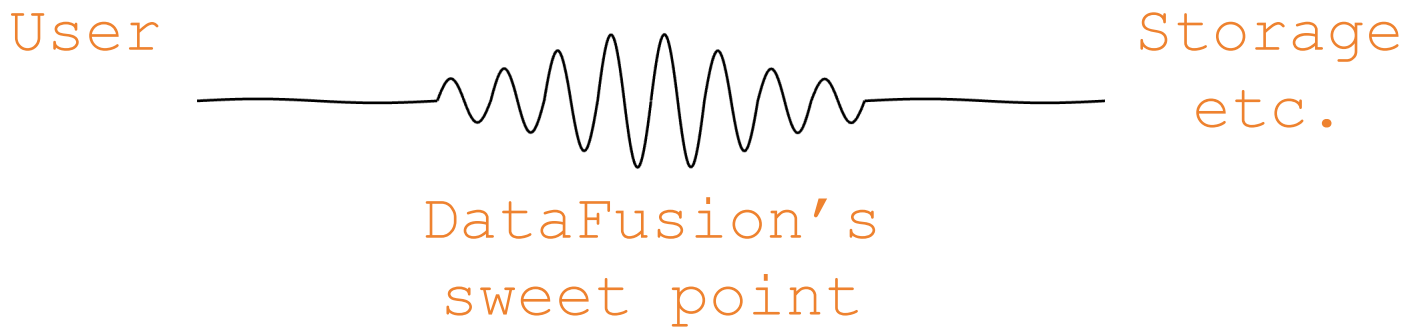
E.g.: a matrix of 3 array elements

#3 Optimizations (cont.)

- Work with existing optimizer rules by specifying properties correctly
- Implement customized optimizer rules (discuss later)
- BTW, optimizer is a great place for hacking

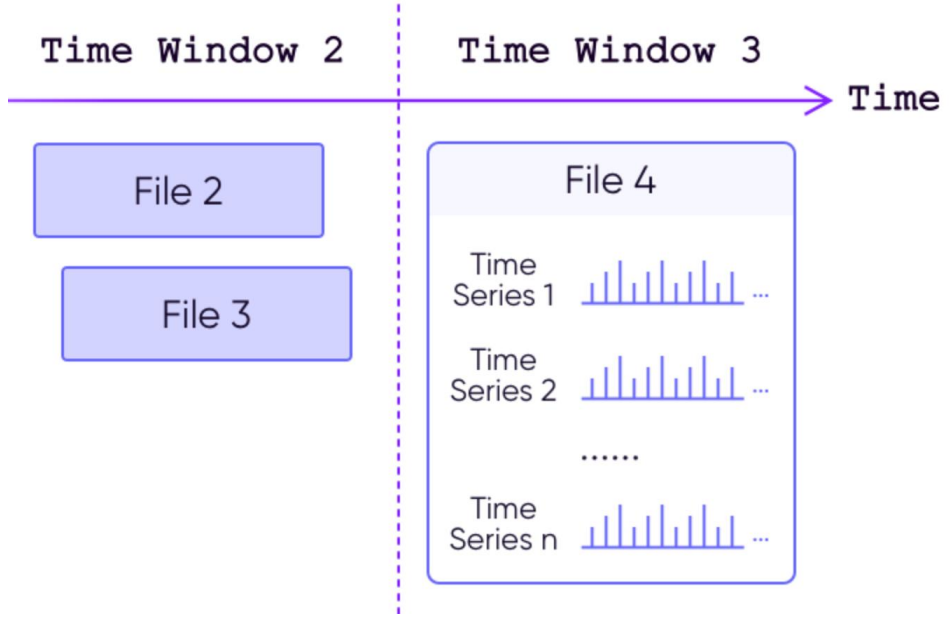
#3 DataFusion is not LLVM

- Primitives are not Low-Level enough
- Closer to a shared framework with splendid extensibility
- Between user and underlying system
- It doesn't make sense to require DataFusion for every computation
- Do more to make it faster



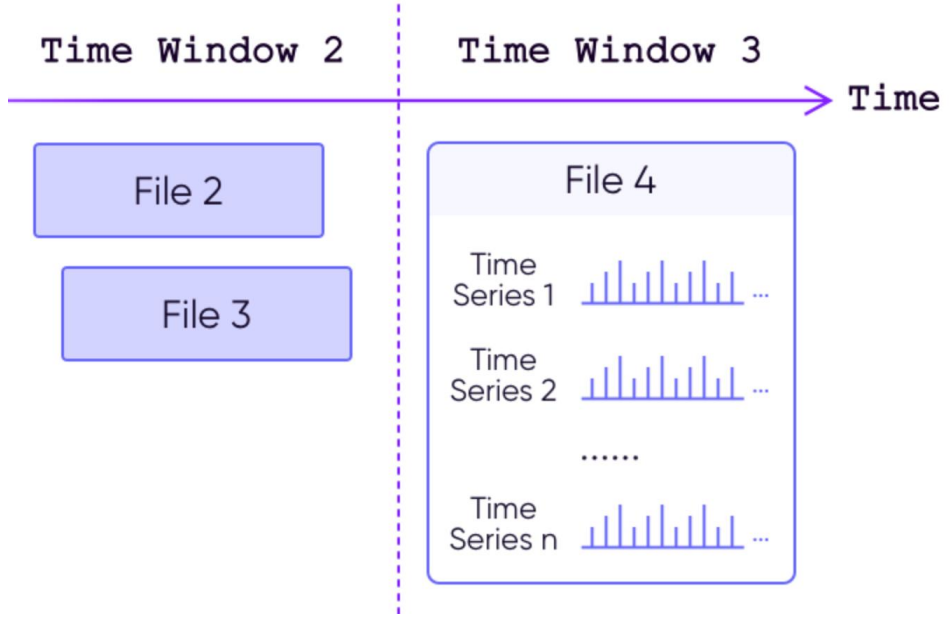
#3 Extend for Performance

- Time-Series Distribution in Storage Engine
  - Data is physically partial ordered
  - Files are within specific ranges
  - Data from one key are continuous in a file



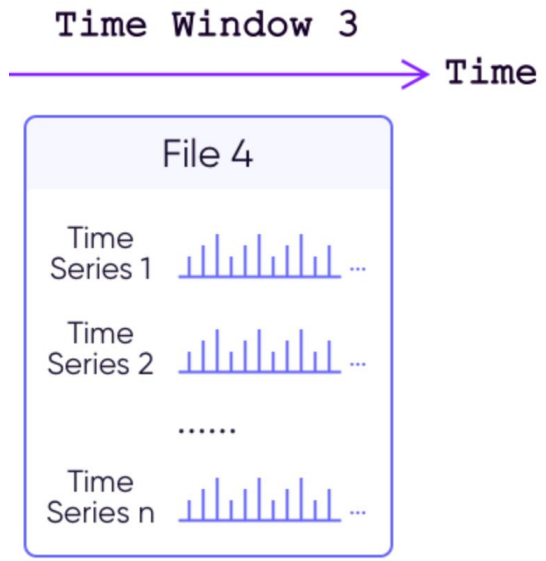
#3 Distribution Matters

- Pruning on time range or time-series key
- Aggregator that needs this distribution
- Sort



#3 E.g.: windowed sort

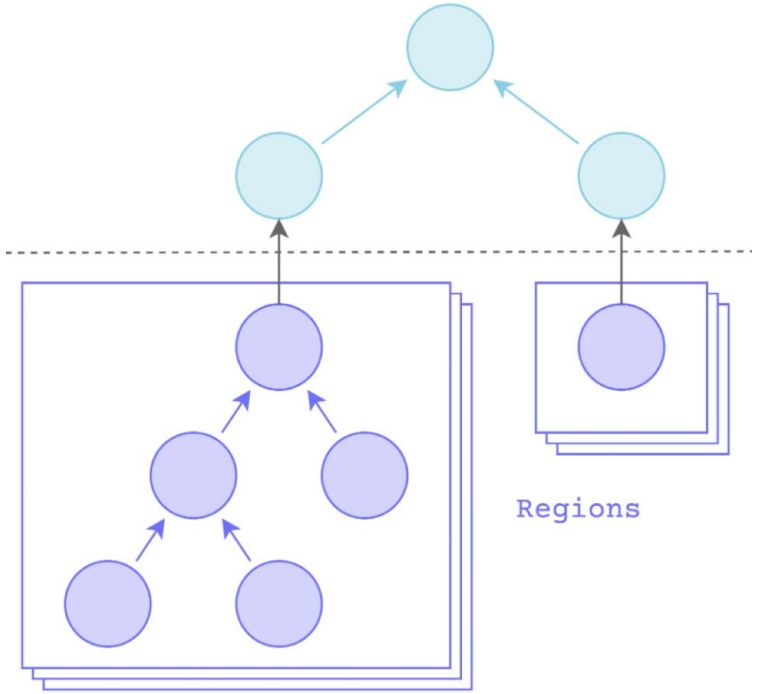
- `... ORDER BY ts, ... LIMIT x`
- Segment the global sort into several small sorts (sort within each window)
  - $O(N * \log N) \rightarrow K * O(M * \log M)$
    - $K * M = N$
- Cut the stream as soon as possible



Only this file4 is read

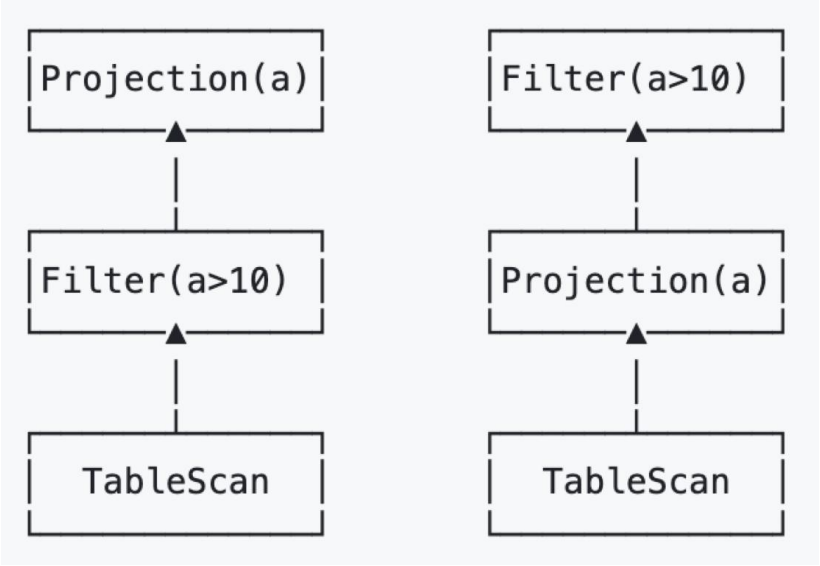
#3 Extend for Scale

- Build a distributed query engine on top of DataFusion
- Transform the Logical Plan into
- Detailed in this slides



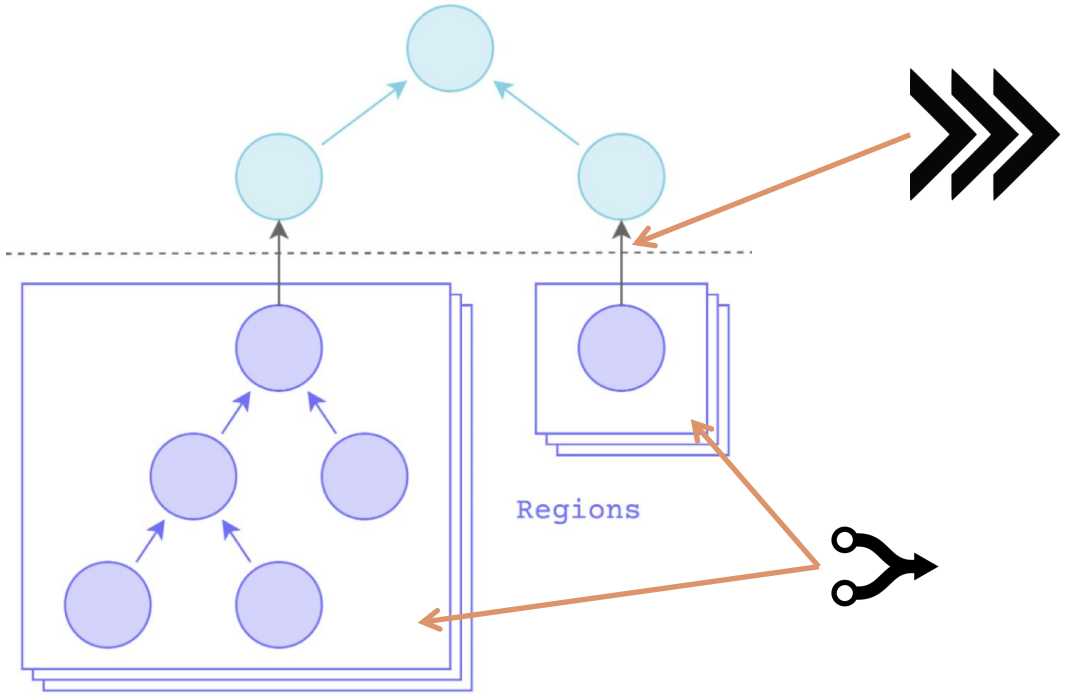
#3 Commutativety

- Whether two operations can exchange their order
- Example: Projection & Filter are commutative



#3 Remote Data Source

- Leaf node pull data from a data source
- It can be either file source or another sub-tree
- DataFusion Executor doesn't know this



#3 Extend on Type

- The darkest part IMO
- It's always need for the No. n+1 data type
  - Vector, JSON, geo...
- DataFusion is working on support customized type,
  - but it's not enough
- We don't provide all the types supported by Arrow and DataFusion either

#4 Misc.

- Is reusing common building blocks a good way?
  - shared community & development progress
  - acceptable overhead
  - evolve together
- The “other stuff” truly matters