Build a Distributed Execution Engine in GreptimeDB with Apache DataFusion

Outline

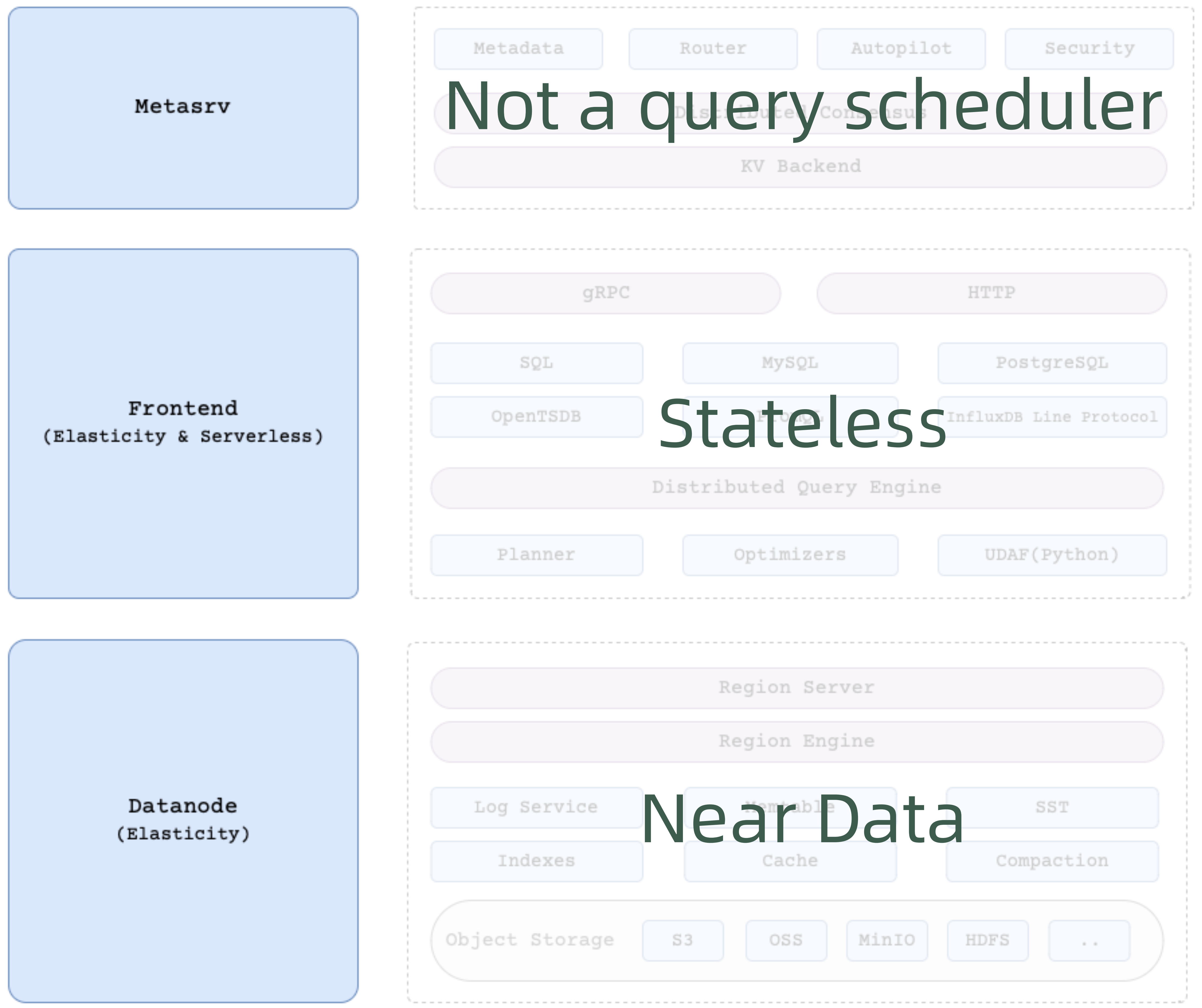
- Background
- How to get “distributed”
- Compare with …
- Future work?
- Q & A

§1

Background

Our Arch.

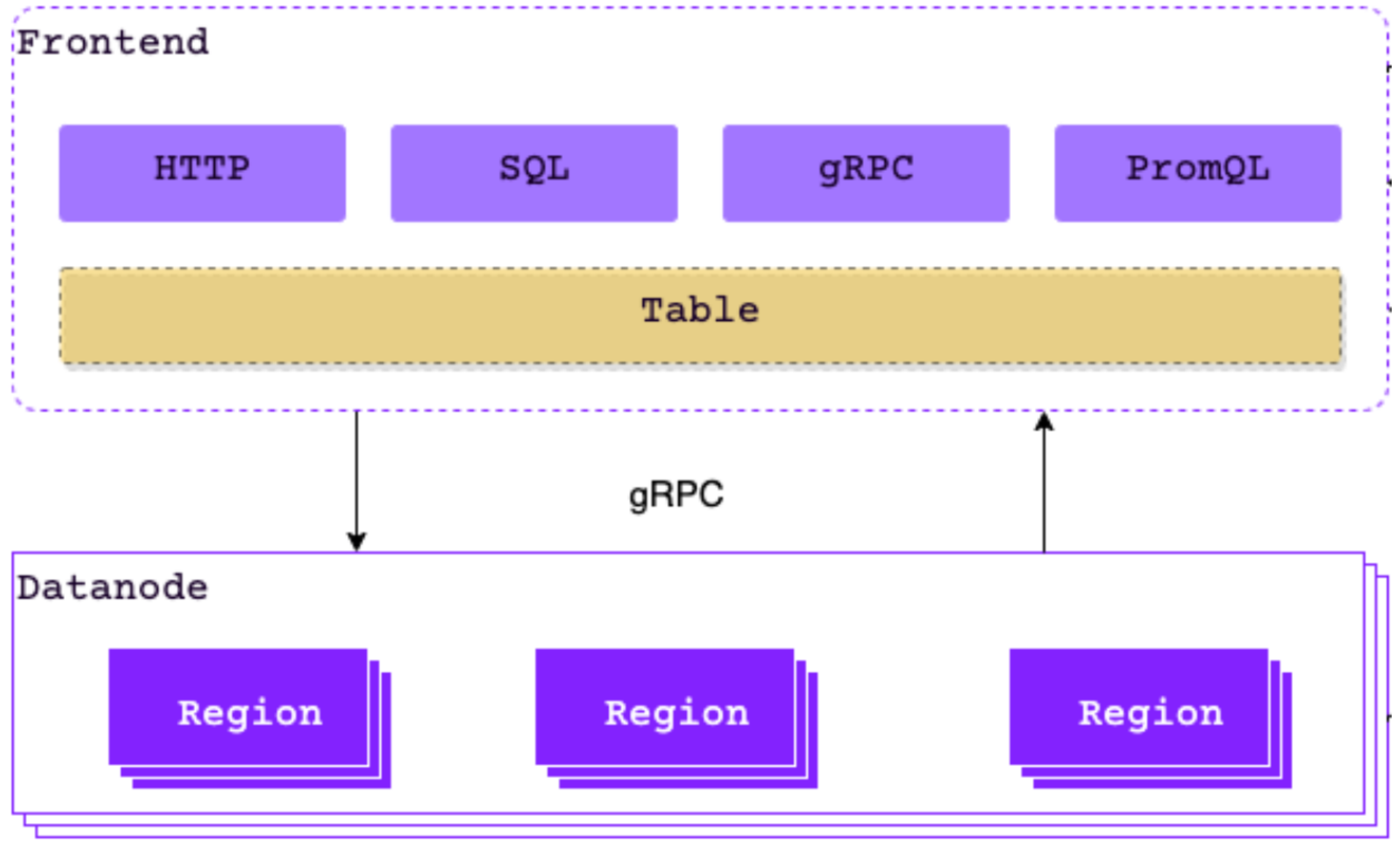
- Typical “cloud native” components
- Storage system with computation
- Deploy in layers



- promo:
  - https://github.com/GreptimeTeam/greptimedb
  - https://github.com/apache/datafusion

the Frontend

- Entry point of queries in different lang
- With the concept of “Table”
- Has a full-featured DataFusion



the Datanode

- Data Storage
- With the concept of “Region”
- Has a full-featured DataFusion

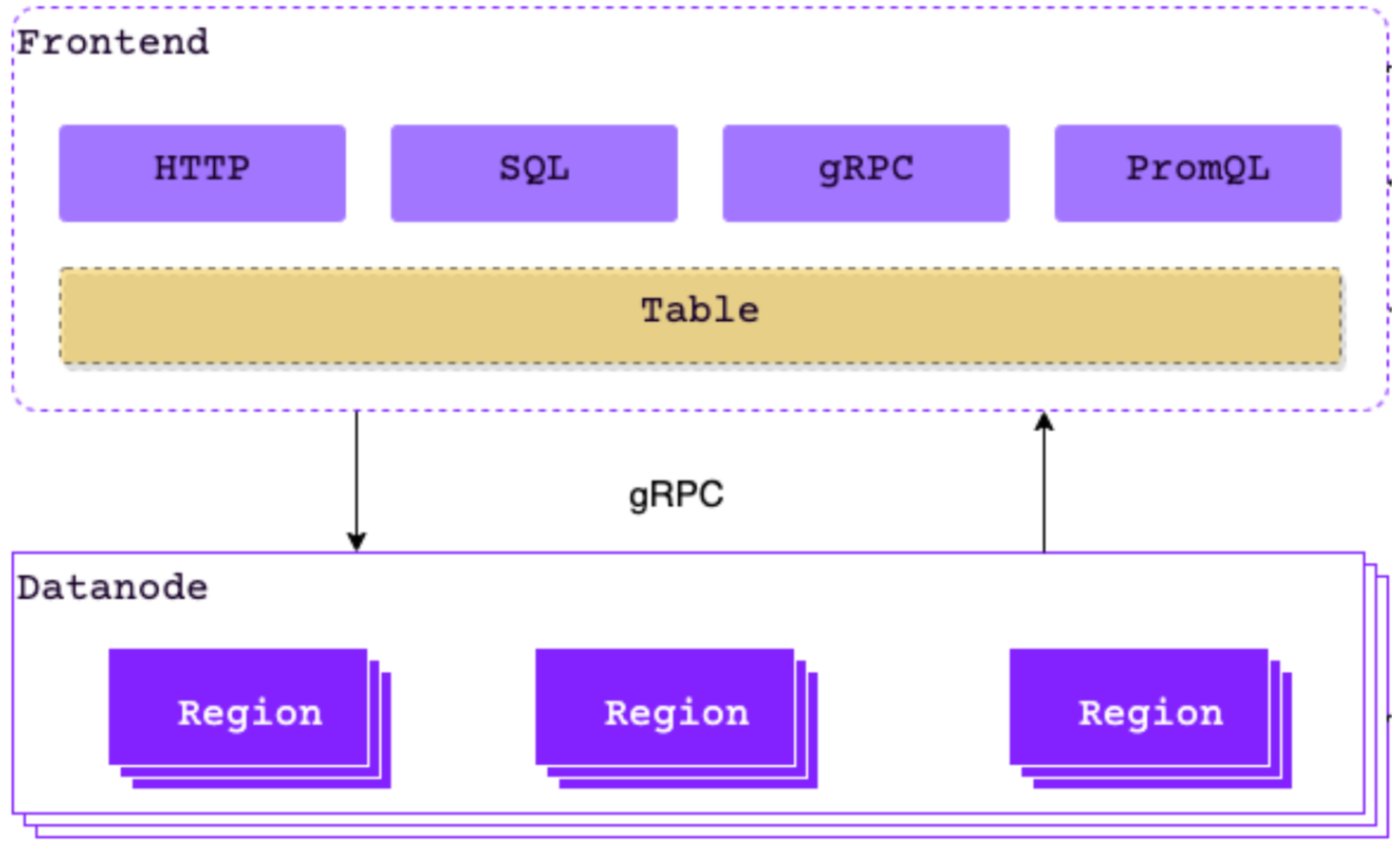
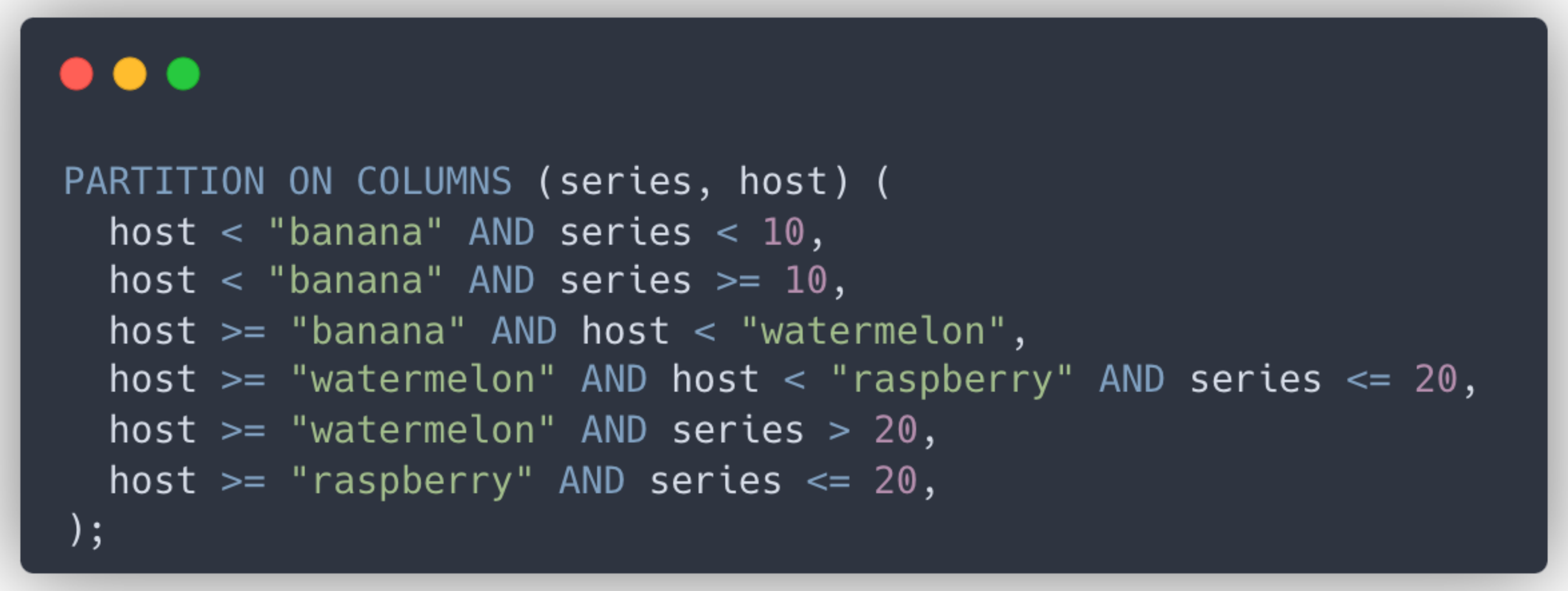


Table & Region

- One Table has multiple Regions
- Each Region has a part of data 👉
- Regions have identical schema



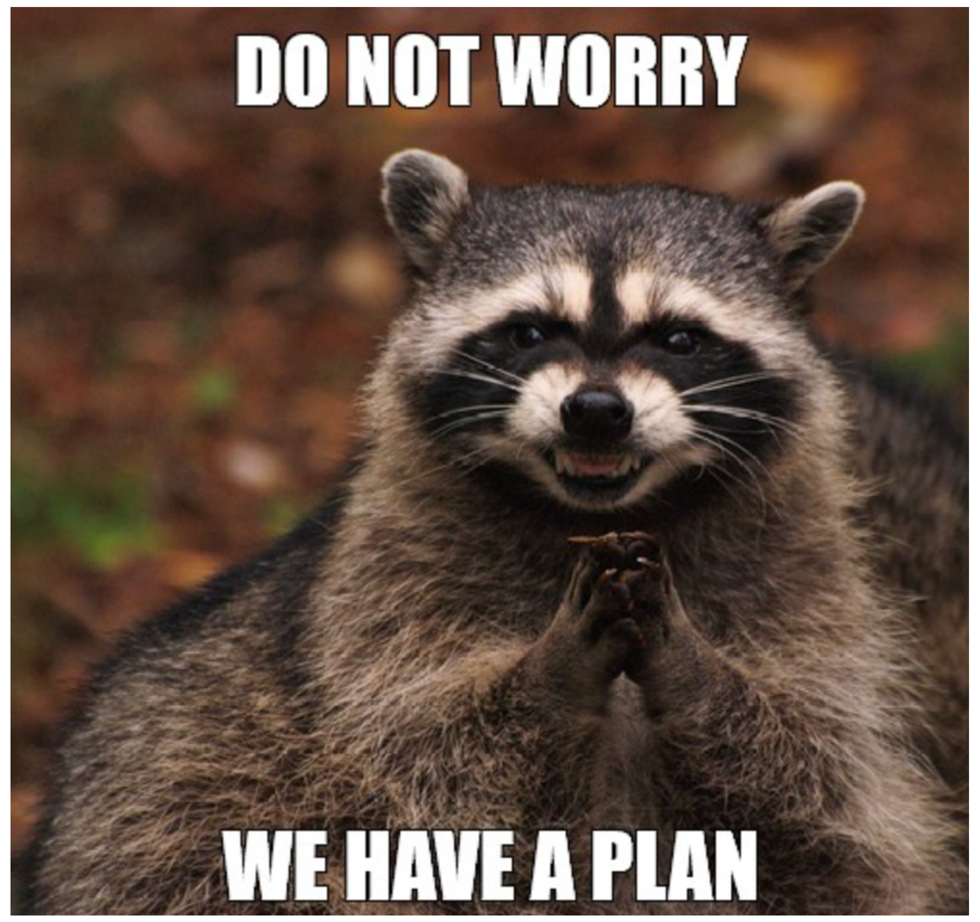
```
PARTITION ON COLUMNS (series, host) (
  host < "banana" AND series < 10,
  host < "banana" AND series >= 10,
  host >= "banana" AND host < "watermelon",
  host >= "watermelon" AND host < "raspberry" AND series <= 20,
  host >= "watermelon" AND series > 20,
  host >= "raspberry" AND series <= 20,
);
```

§2

How to get “distributed”

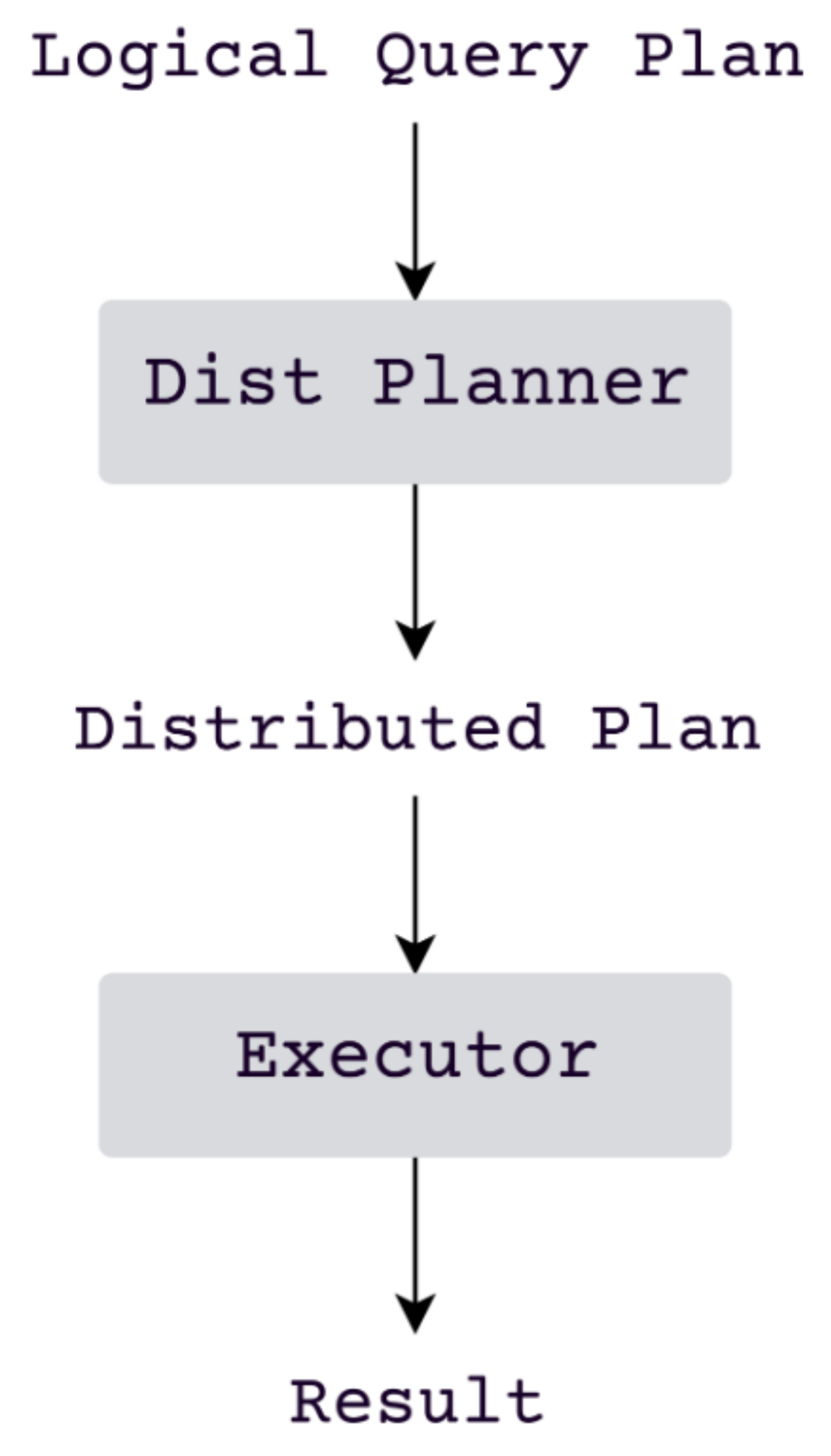
Distributed computation

- What: A query which is executed in multiple nodes
- How: I have a Plan
- Plan: Describes how computation should be performed
- 💡: Distributed Plan!



Distributed plan: three steps 🐘

- 1) Get a plan
- 2) Include several nodes in this plan
- 3) Execute this plan

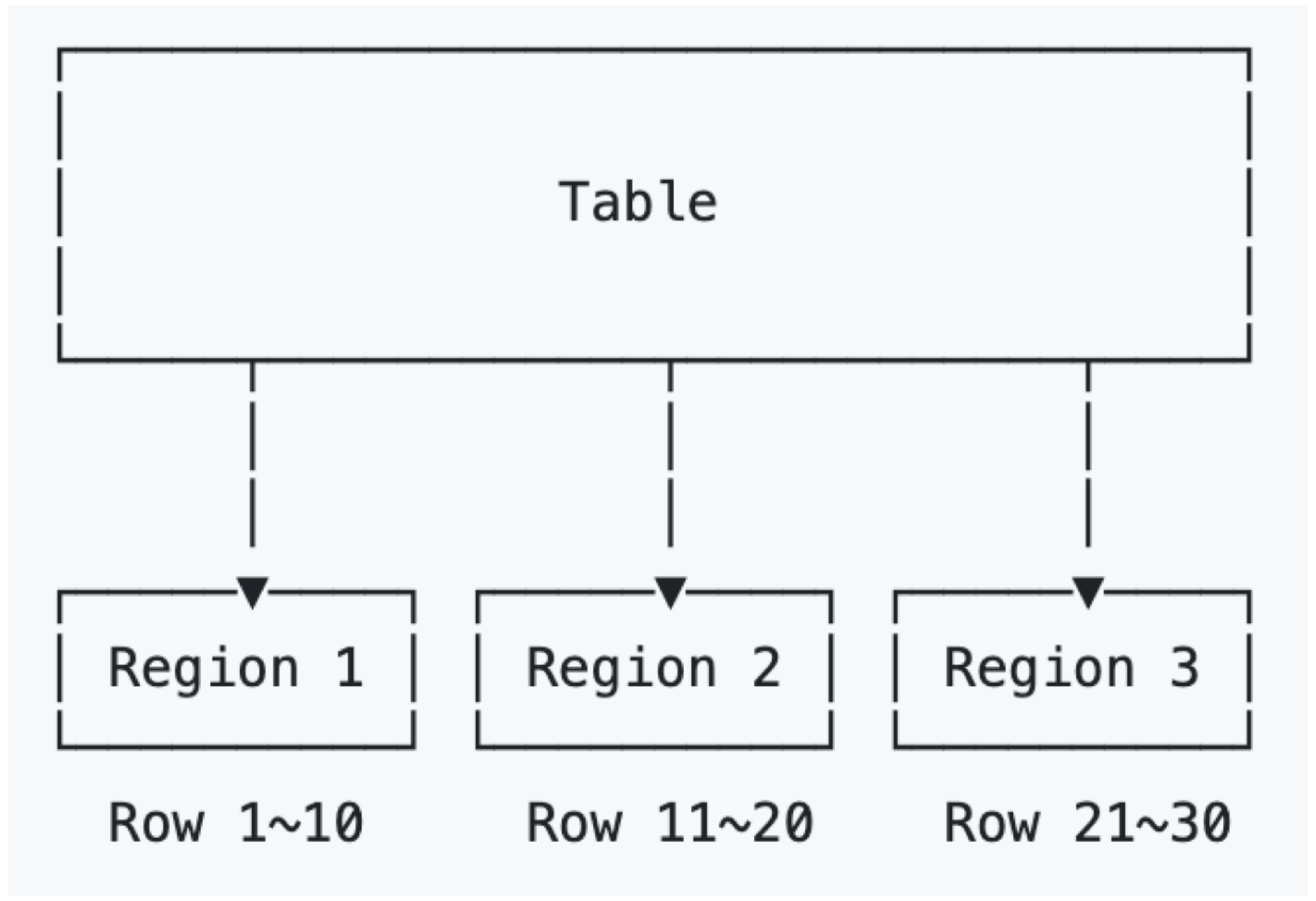


Choice: Why logical plan

- Simple and straightforward, focus on relation algebra
- Leave the execution details to the node who actual executes the part
    - how many partitions should be
    - whether to apply index
    - …

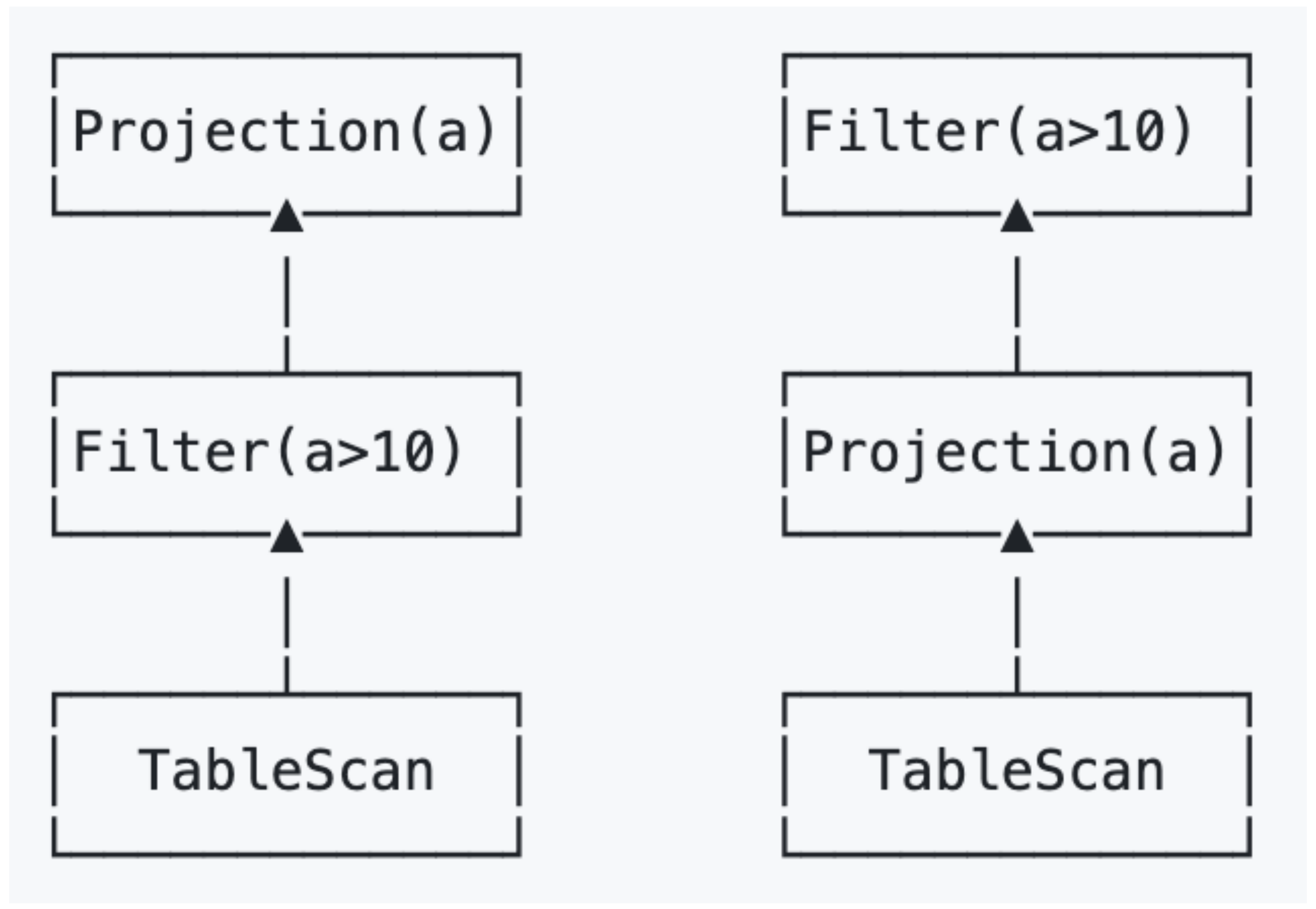
Transform logical plan

- A plan cannot be cut at a random node
- Region is the minimum element of data distribution
- It can also be the unit to distribute computation



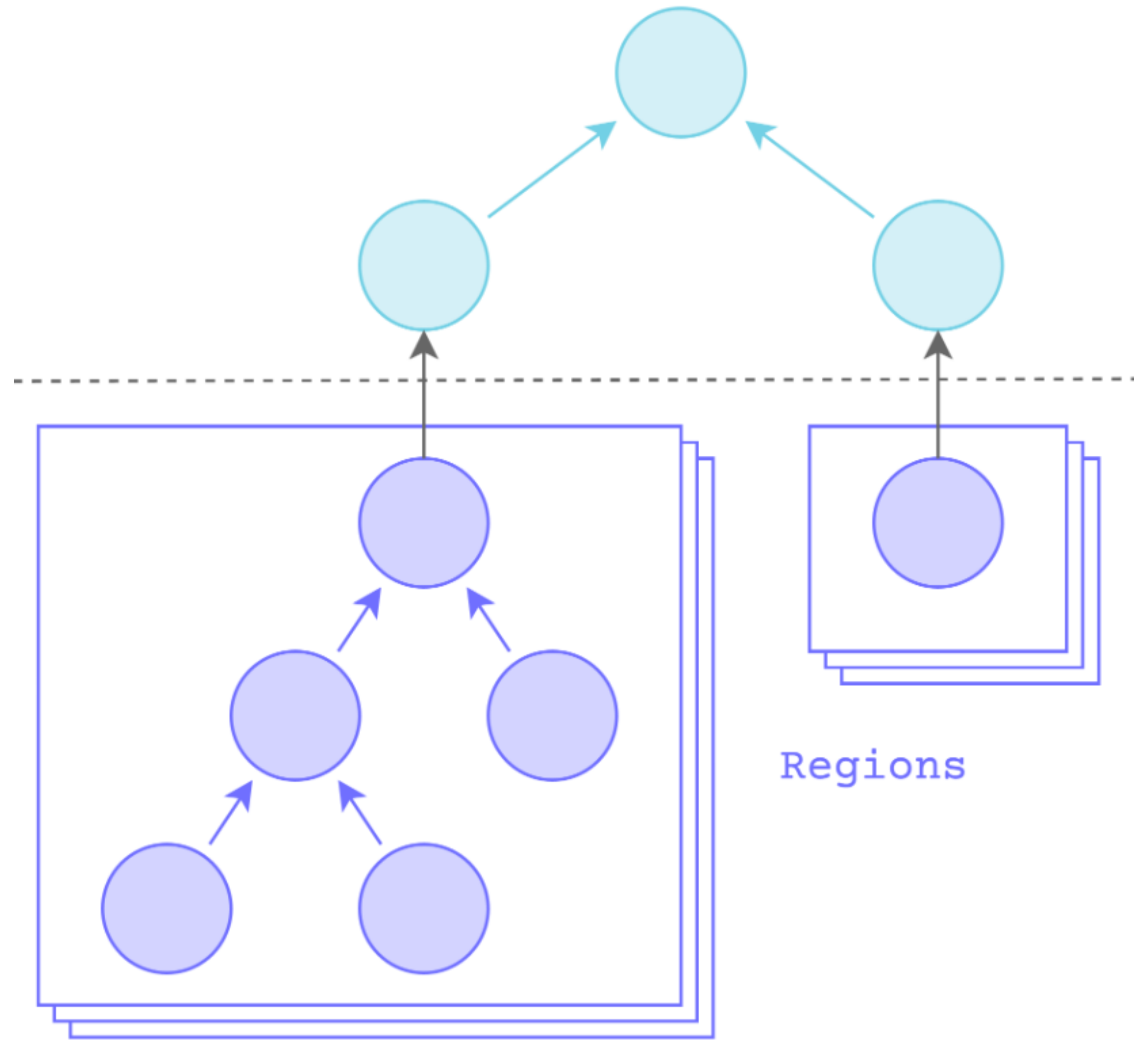
Commutativity

- Whether two operations can exchange their order
- $P1(P2(R)) \Leftrightarrow P2(P1(R))$
- Example: Projection & Filter are commutative



Merge results

- Before merging, results are independent (defined by partition rule)
- Merging changes the distribution of data
- Different plans has different requirement of distribution (e.g.: SortPlan)

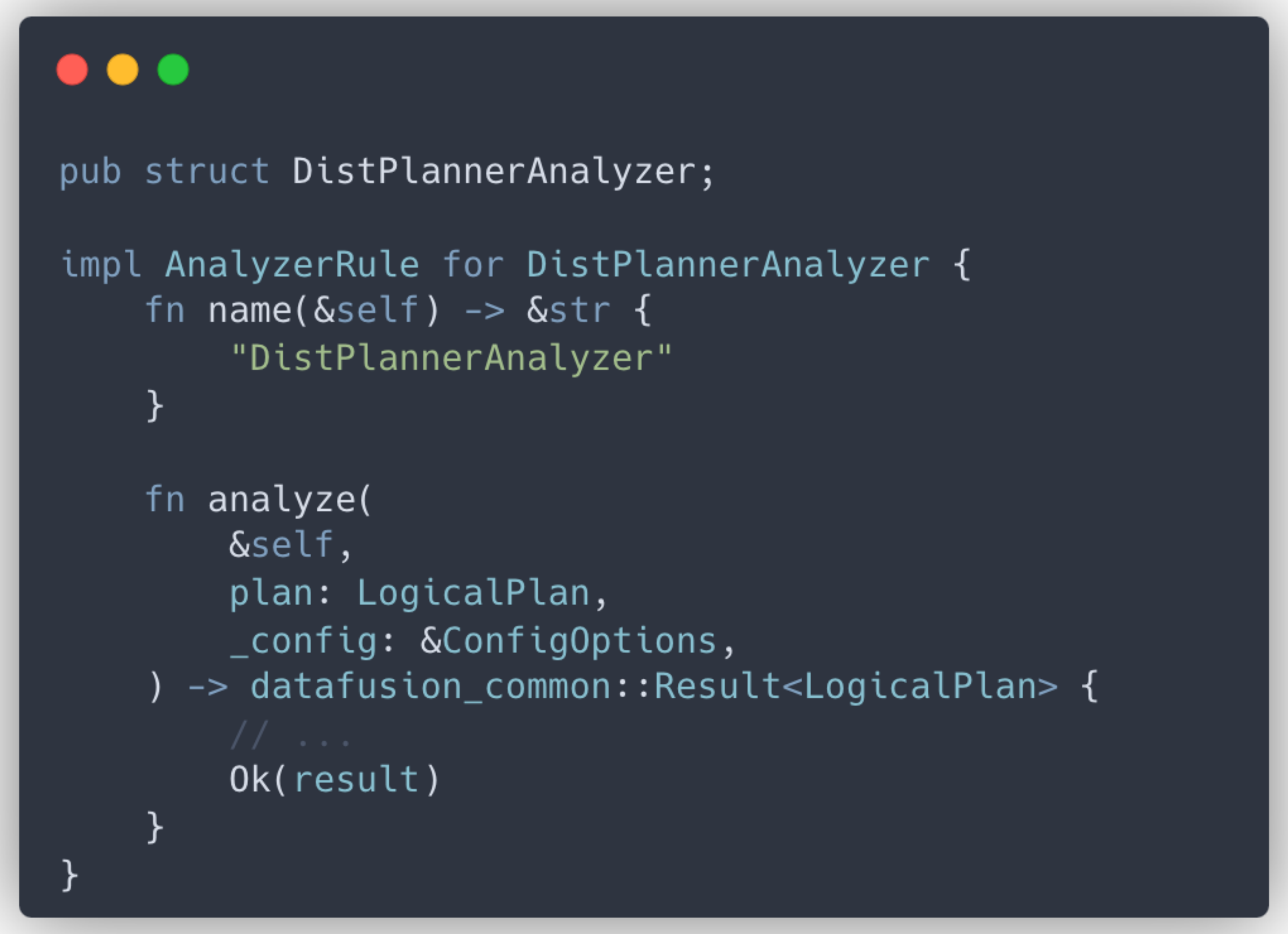


Rule of transform

- Assume every plan nodes need to be merged (no execution is distributed)
- Inspect the commutativity of each node, and try to swap them
- Merge on non-commutative node
- Repeat if necessary
- 🍨: time-series bonds!

“Planner” beneath the mask

- The “planner” or “transformer” is not a real planner
- It’s implemented as an `AnalyzerRule` in DataFusion



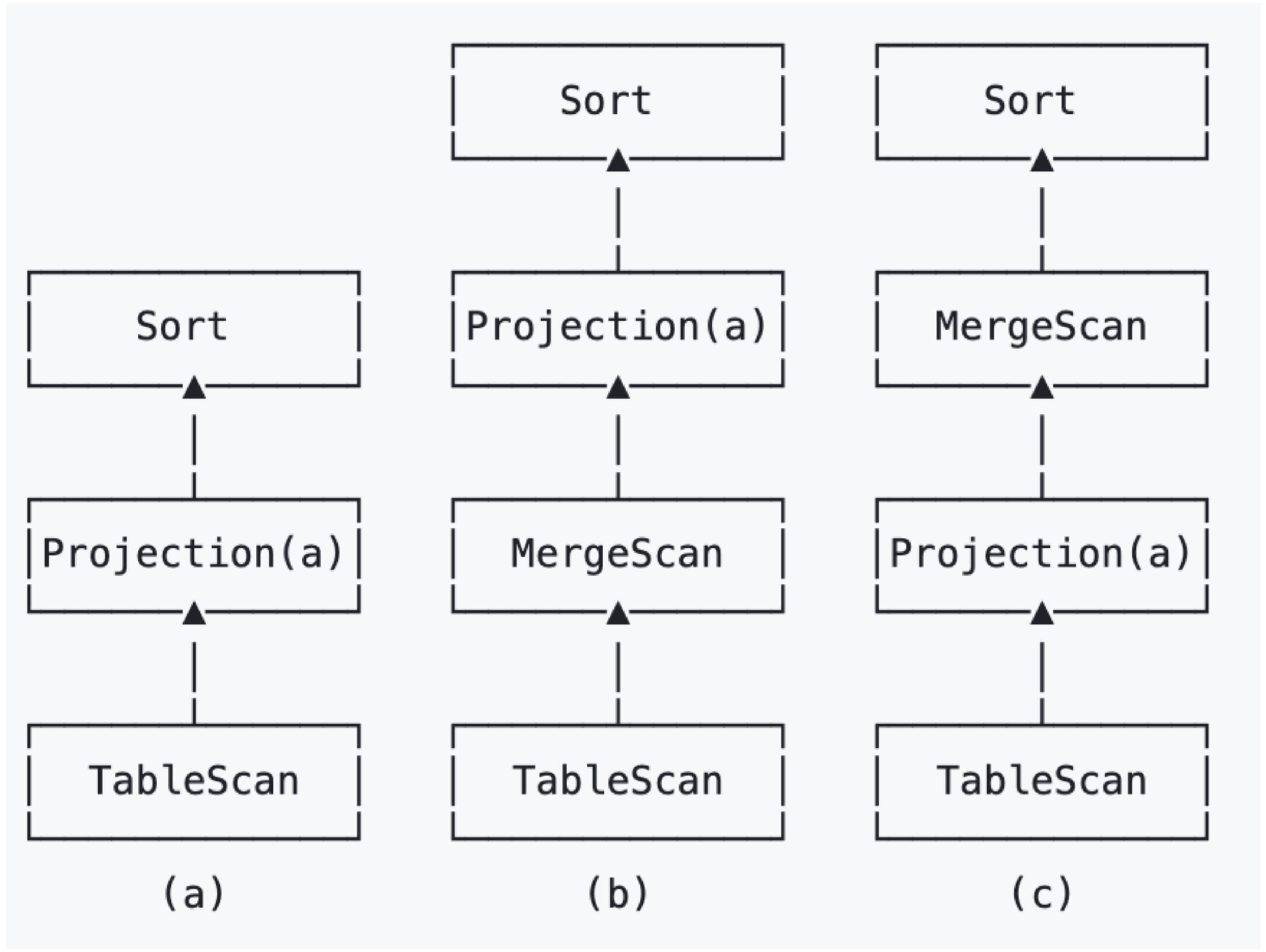
```rust
pub struct DistPlannerAnalyzer;

impl AnalyzerRule for DistPlannerAnalyzer {
    fn name(&self) -> &str {
        "DistPlannerAnalyzer"
    }

    fn analyze(
        &self,
        plan: LogicalPlan,
        _config: &ConfigOptions,
    ) -> datafusion_common::Result<LogicalPlan> {
        // ...
        Ok(result)
    }
}
```

Commutativity of MergeNode

- Commutative
  - **Filter**
  - **Projection**
  - **Aggr** within partition (🍨)
- Partial Commutative
  - **Min** $min(R) \rightarrow min(MERGE(min(R)))$
  - **Max**
- Conditional Commutative
  - **Count** $count(R) \rightarrow sum(count(R))$
- Transformed Commutative
  - **Avg** $avg(R) \rightarrow sum(R)/count(R)$
- Non-commutative
  - **Sort**
  - **Join**
  - **Percentile**

Example

- Assume every plan nodes need to be merged (no execution is distributed)
- Inspect the commutativity of each node, and try to swap them



Protocol of query

- Substrait: Cross-Language Serialization of Relational Algebra
- https://substrait.io
- MergeNode: **`Fn(SubstraitPlan) -> Vec<Result>`**

Protocol of result

- Apache Arrow Flight: high-performance data services framework
- https://arrow.apache.org/docs/format/Flight.html
- MergeNode: **`Fn(SubstraitPlan) -> Vec<FlightResult>`**

Choice: Why substrait

- Generic
- The possibility to offload planning, optimizing and executing to other systems
- De facto IR in GreptimeDB (is now shared with another streaming execute engine)

§3

Compare with ...

Pros. & Cons.

- Pros.:
  - Easy to implement & maintain
  - Lightweight & simple
  - Low overhead
- Cons.:
  - Not suit for hard pipeline breaker (Join, Sort etc.)
  - Not for jobs that would last for like a day

§4

Future Work

Planned but not yet impl…

- Spill and shuffle for hard breaker (for large dataset)
- Multi-stage on Frontend (for large table)
- Accomplish rule set, including non-SQL relations (like PromQL)
- Breakdown the Datanode part (gain smaller granule)

Wondering if 💭

- Offload tasks to other systems
- Merge from other systems

§5

Q & A