End-to-End Declarative Data Analytics:

Co-designing Engines, Interfaces, and Cloud Infrastructure

Pinghe Li*, Tom Kuchler*, Marko Kabić*, Tobias Stocker*,
Gustavo Alonso, **Ana Klimovic**

ETH zürich

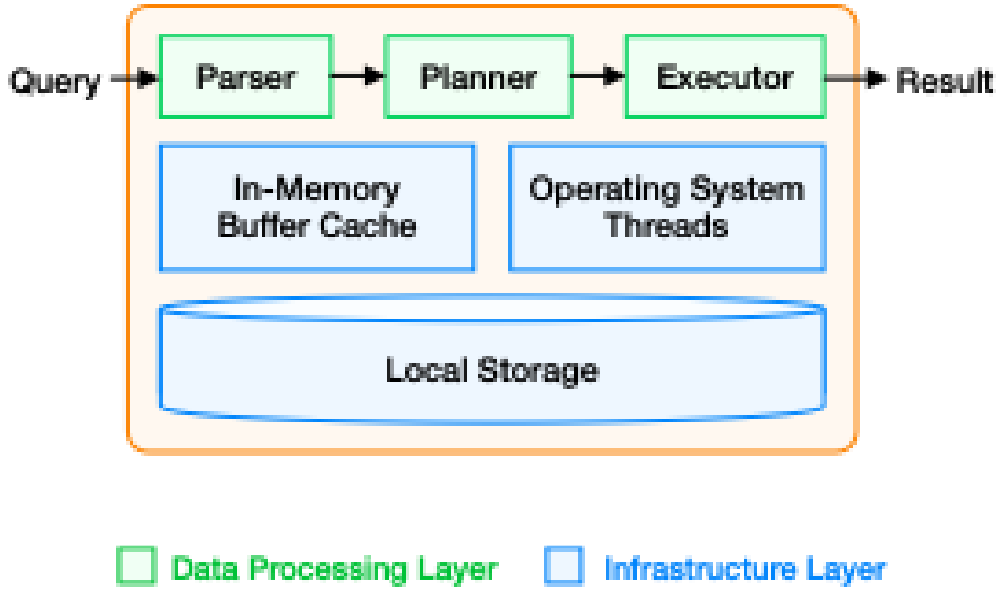
Systems@ETH zürich

CIDR 2026

Data Analytics Engine Evolution

A traditional, monolithic engine

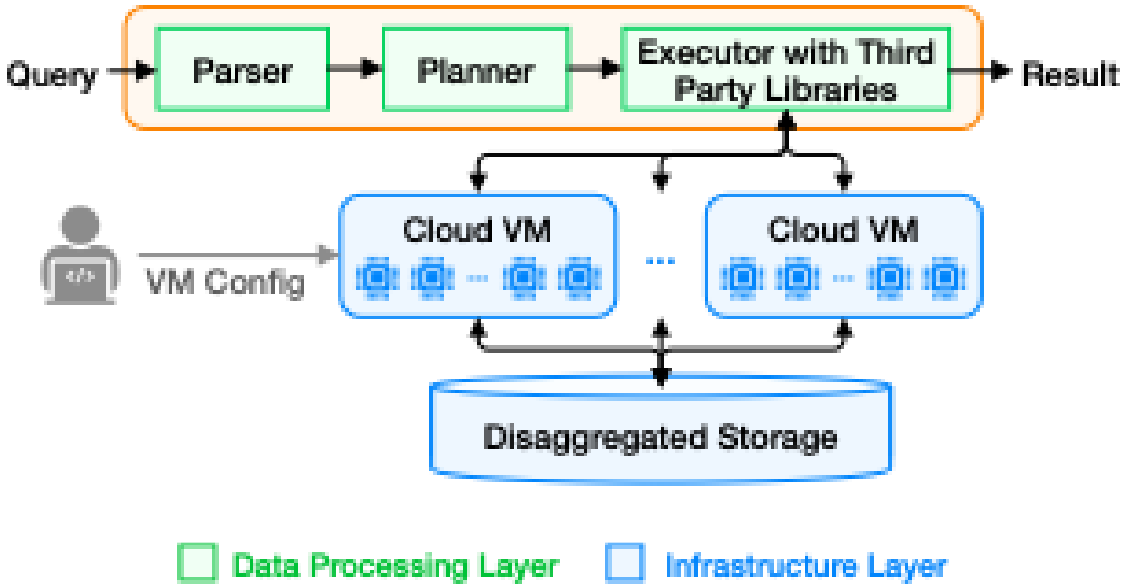
- Assumes full ownership of the data
- Assumes full control of compute and memory



Data Analytics Engine Evolution

A **composite** engine in the **cloud**:

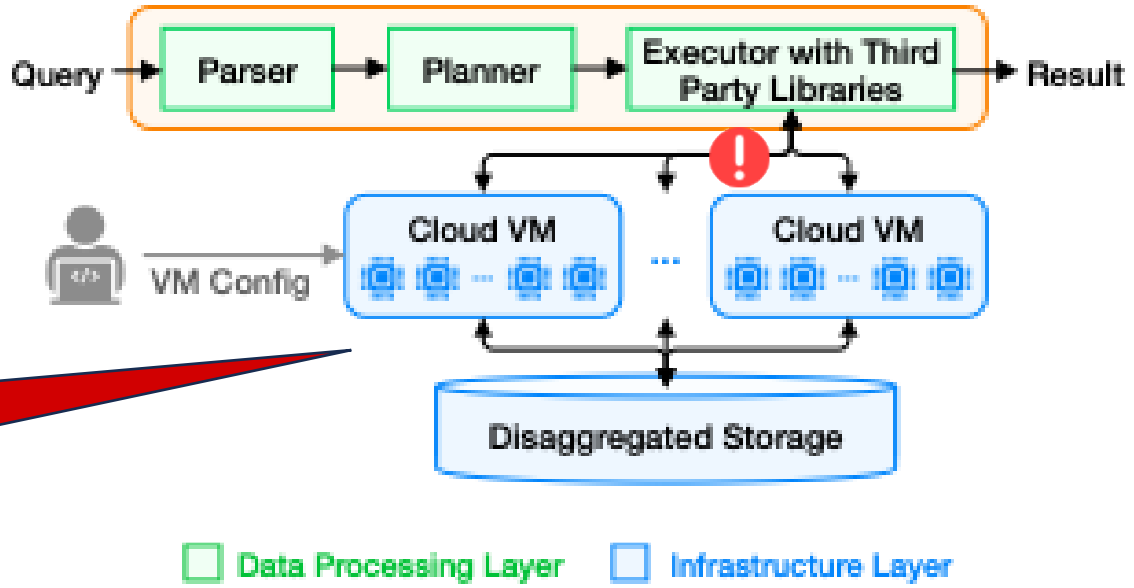
- No longer owns the data
  → stored on data lake
- Many layers of software between the engine and the hardware it runs on



Shortcomings of Today's Cloud Analytics Stack

The declarative nature of data processing ends at query planning/optimization.

- Infrastructure has limited visibility into operators (and their inputs/outputs)



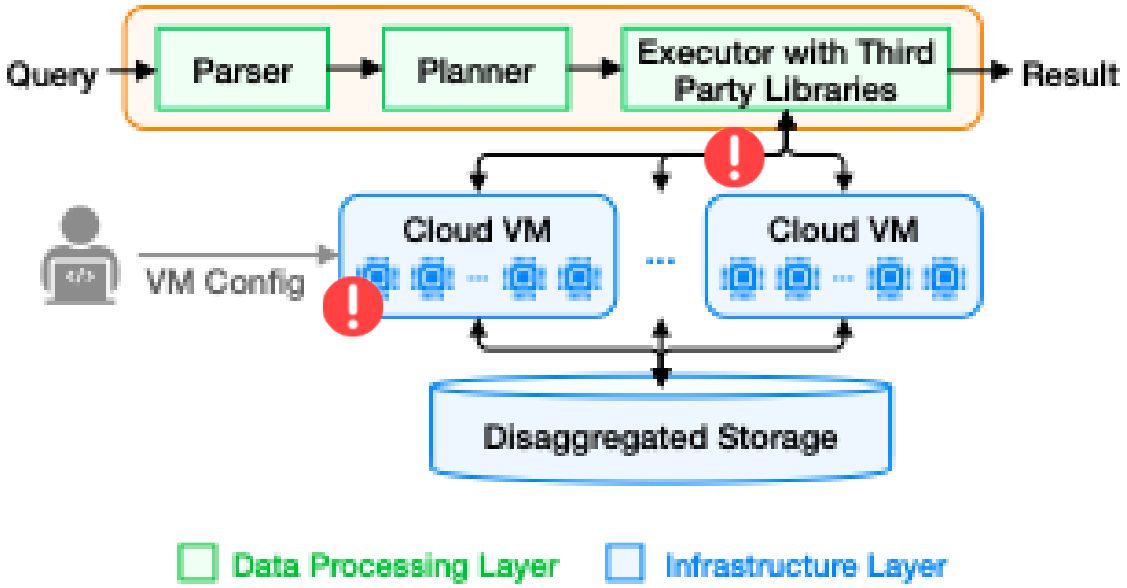
How can the cloud runtime schedule tasks to minimize data movement or decide how to size VMs?

Missing information about query's dataflow and parallelism of the workload!

Shortcomings of Today's Cloud Analytics Stack

The declarative nature of data processing ends at query planning/optimization.

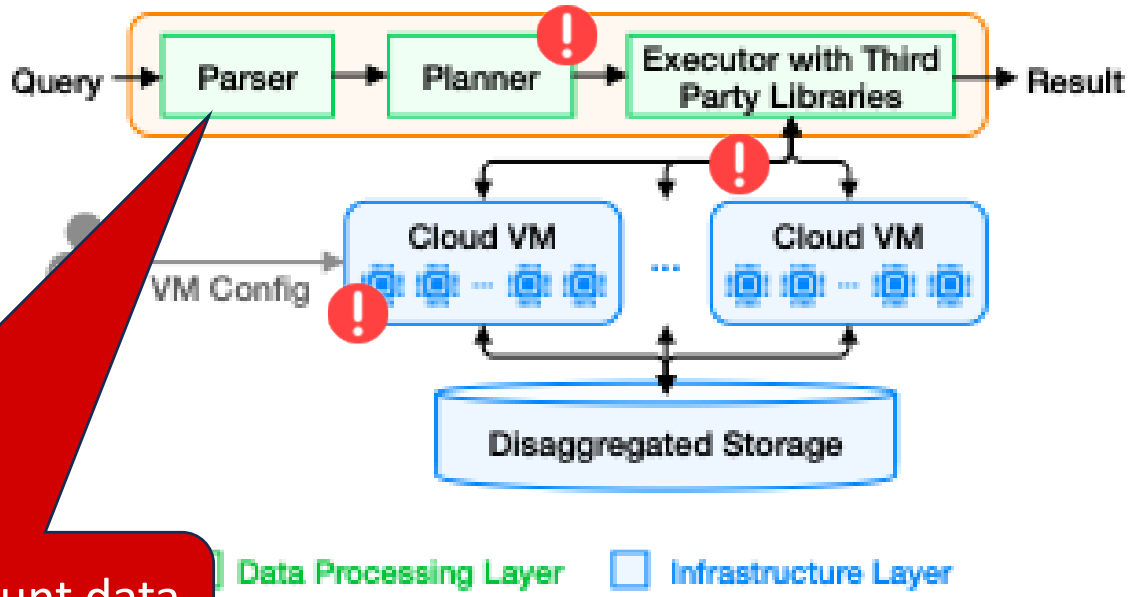
- Infrastructure has limited visibility into operators (and their inputs/outputs)
- VMs are generally fixed-size and slow to start, which limits elastic scaling



Shortcomings of Today's Cloud Analytics Stack

The declarative nature of data processing ends at query planning/optimization.

- Infrastructure has limited visibility into operators (and their inputs/outputs)
- VMs are generally fixed-size and slow to start, which limits elastic scaling
- Optimizers have limited visibility into the physical execution environment



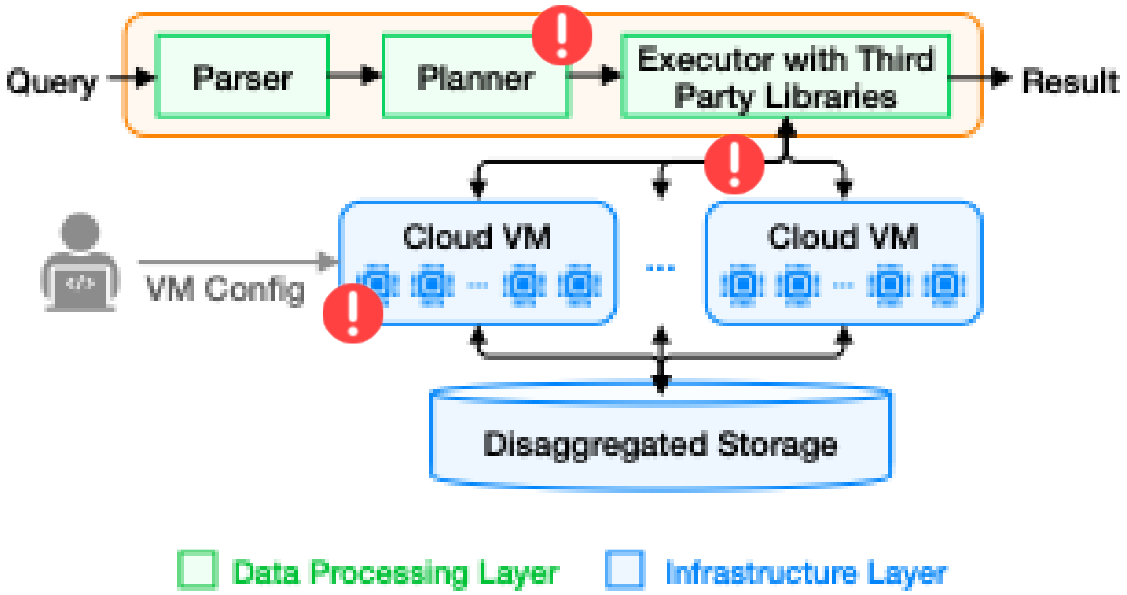
How can the planner take into account data locality and resource contention?

Missing information available at runtime.

Shortcomings of Today's Cloud Analytics Stack

The declarative nature of data processing ends at query planning/optimization.

- Infrastructure has limited visibility into operators (and their inputs/outputs)
- VMs are generally fixed-size and slow to start, which limits elastic scaling
- Optimizers have limited visibility into the physical execution environment

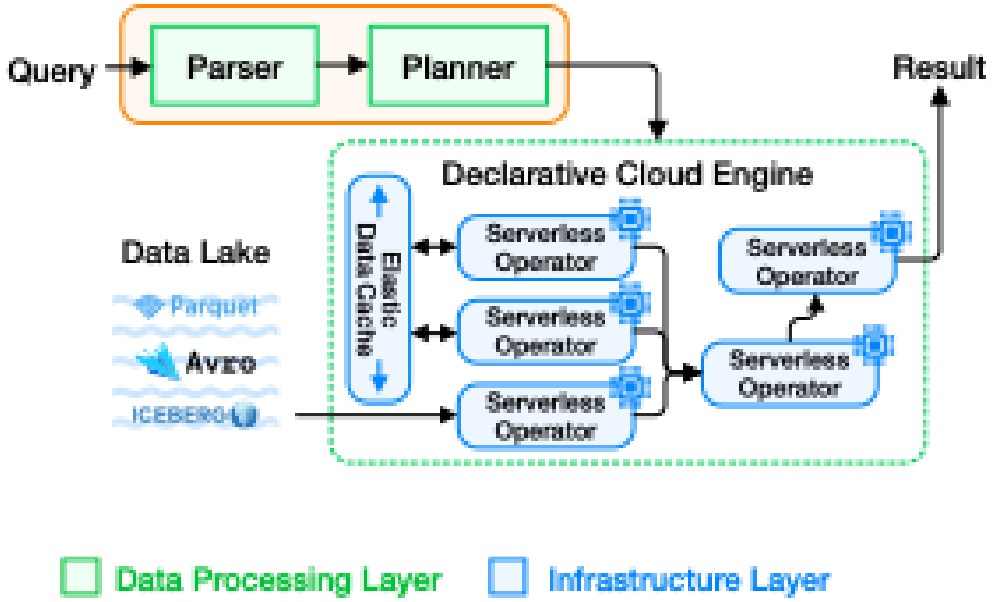


Significant performance & efficiency gains are left on the table!

A Cloud-Native Data Analytics Engine

Co-design the data engine and cloud infrastructure with a *declarative interface.*

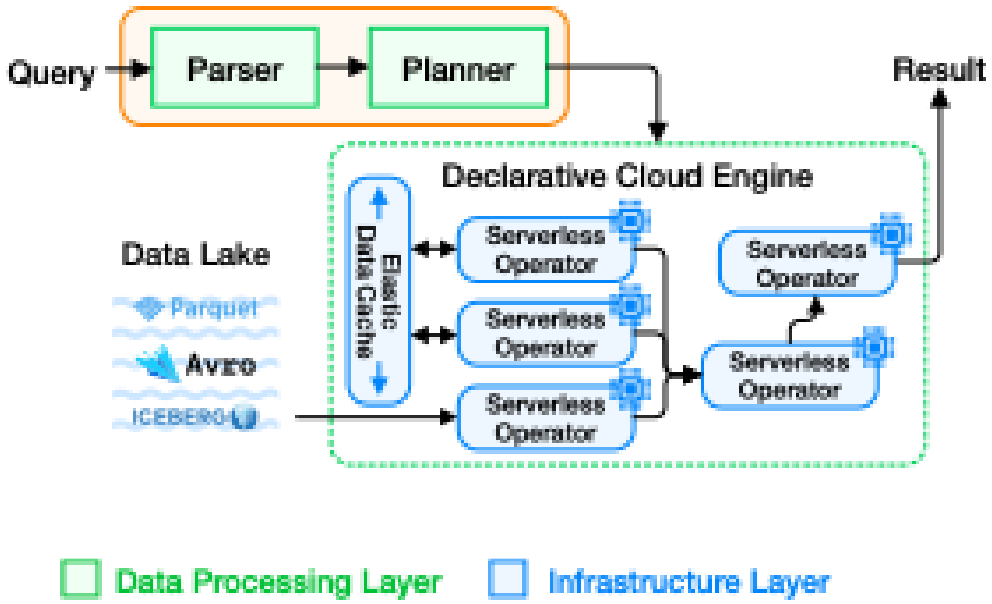
- Expose the query DAG and dataflow to the infrastructure layer
- Infrastructure layer adapts query execution to be hardware-aware and runtime-aware
- Execute query DAG operators with an elastic, declarative serverless runtime



Co-Designing the Data Engine & Cloud Runtime

Data engine decides:

- **Physical plan structure**: predicate pushdown, join order, algorithm choice, etc.
- **Reasonable range of choices** for per-operator degree of parallelism, implementation, etc.



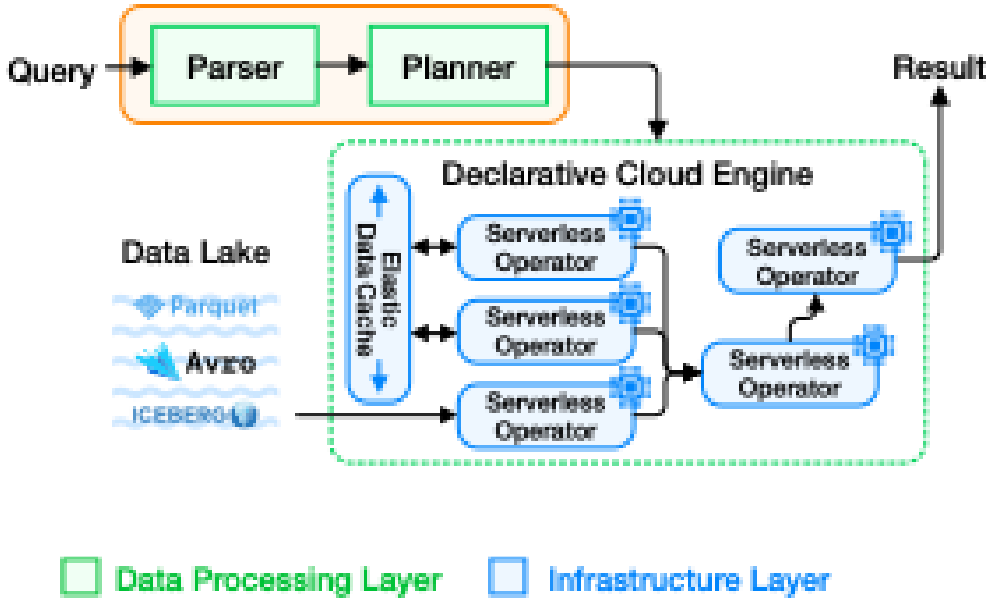
Co-Designing the Data Engine & Cloud Runtime

Data engine decides:

- **Physical plan structure**: predicate pushdown, join order, algorithm choice, etc.
- **Reasonable range of choices** for per-operator degree of parallelism, implementation, etc.

Cloud runtime decides *per-operator*:

- **Elasticity**: degree of parallelism (within bounds)
- **Data caching**: inputs and intermediates to reuse
- **Placement**: where to run, based on data locality
- **Heterogeneous hardware**: CPU type, GPU, etc.
- **More**: degree of isolation, fault recovery, …



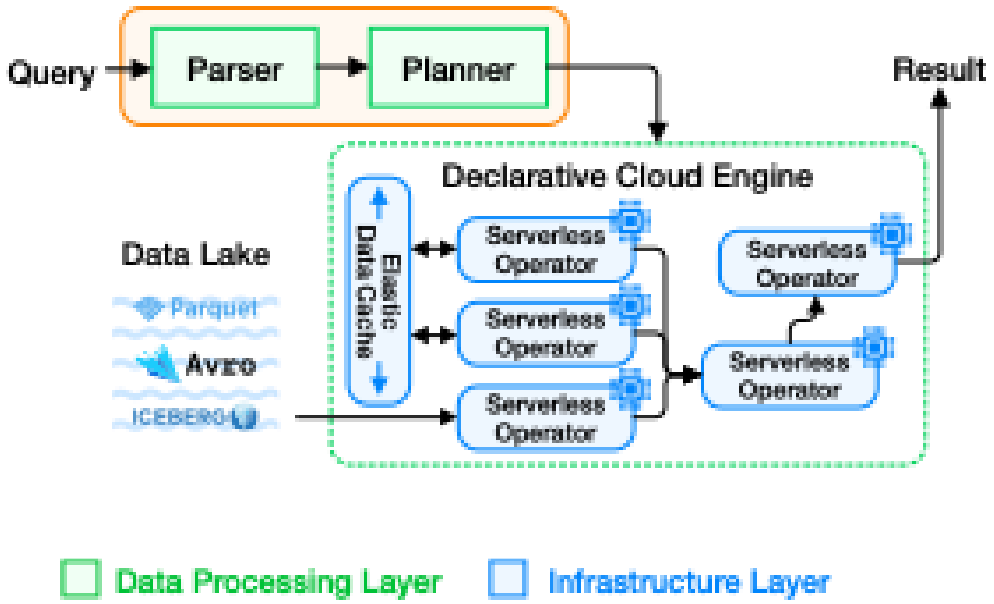
Building a Prototype with Existing Systems

Data Processing Layer: Maximus

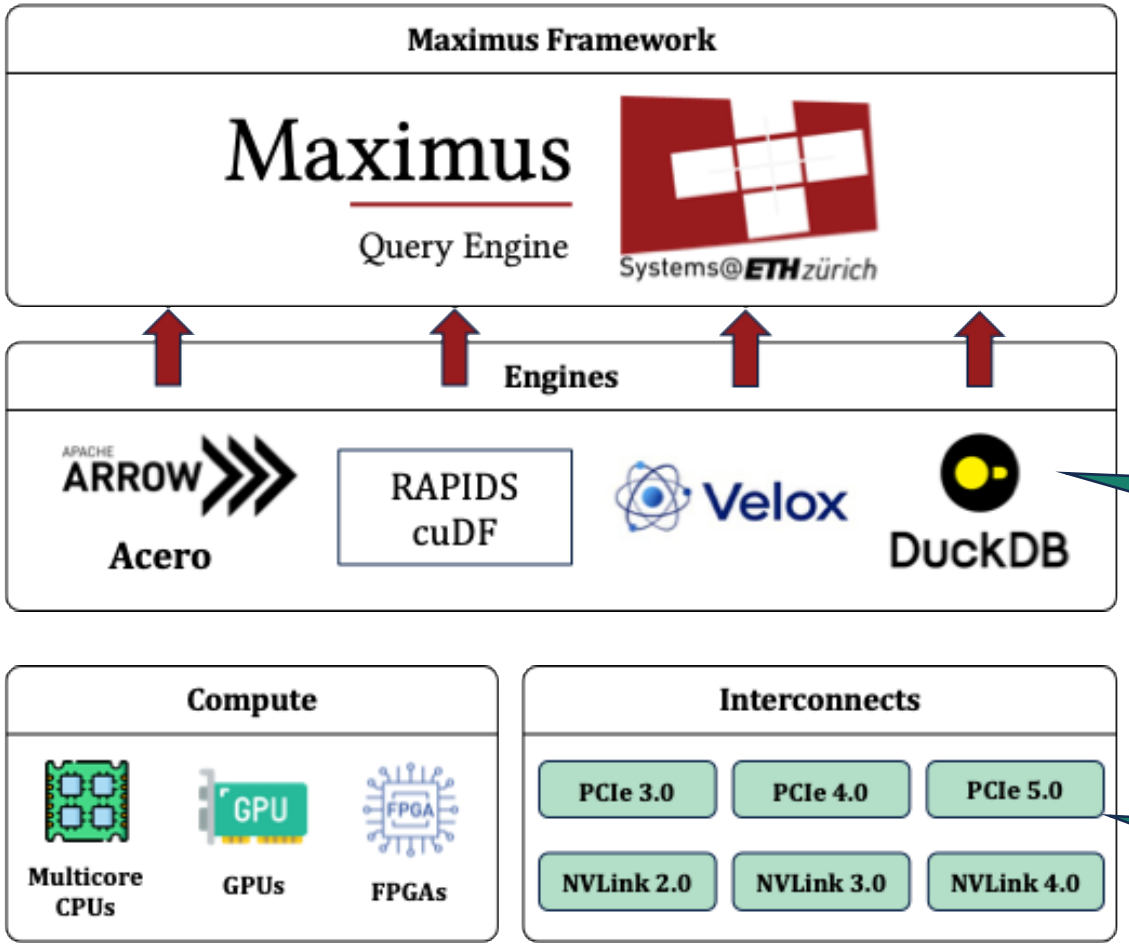
- Composite database engine [SIGMOD'25]
- Allows executing query plans using different backends (CPU, GPU, …)

Infrastructure Layer: Dandelion

- Declarative, serverless cloud runtime [SOSP'25]
- Executes applications as DAGs, which contain two types of operators: pure compute & communication



Why use Maximus as the data engine?



Designed to support a variety of operator libraries

And run on a variety of hardware backends

Why use Dandelion as the cloud runtime?

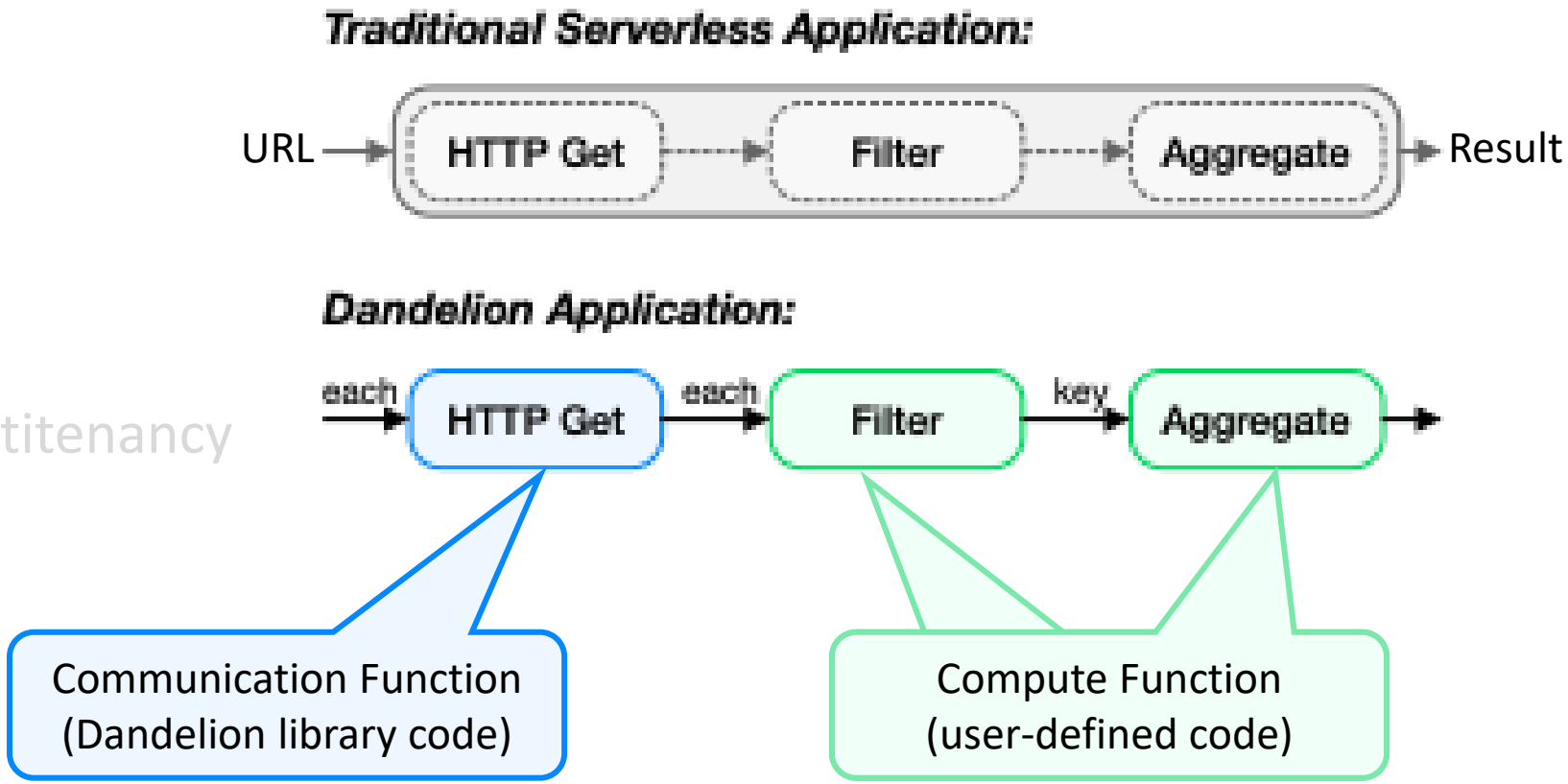
- DAG interface
- High elasticity
- Strong isolation for multitenancy

Why use Dandelion as the cloud runtime?

- DAG interface
- High elasticity
- Strong isolation for multitenancy

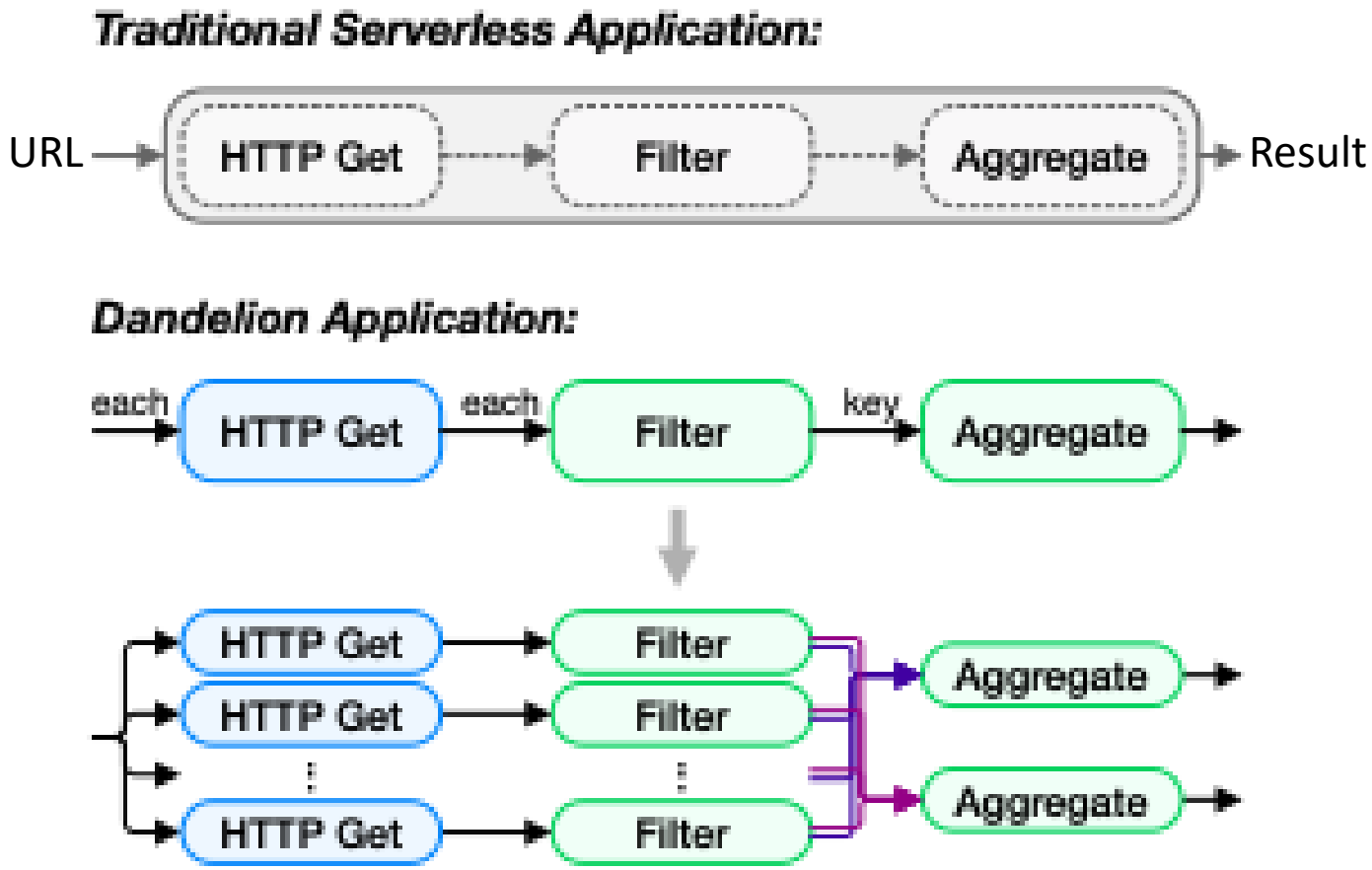
Why use Dandelion as the cloud runtime?

- DAG interface
- High elasticity
- Strong isolation for multitenancy



Why use Dandelion as the cloud runtime?

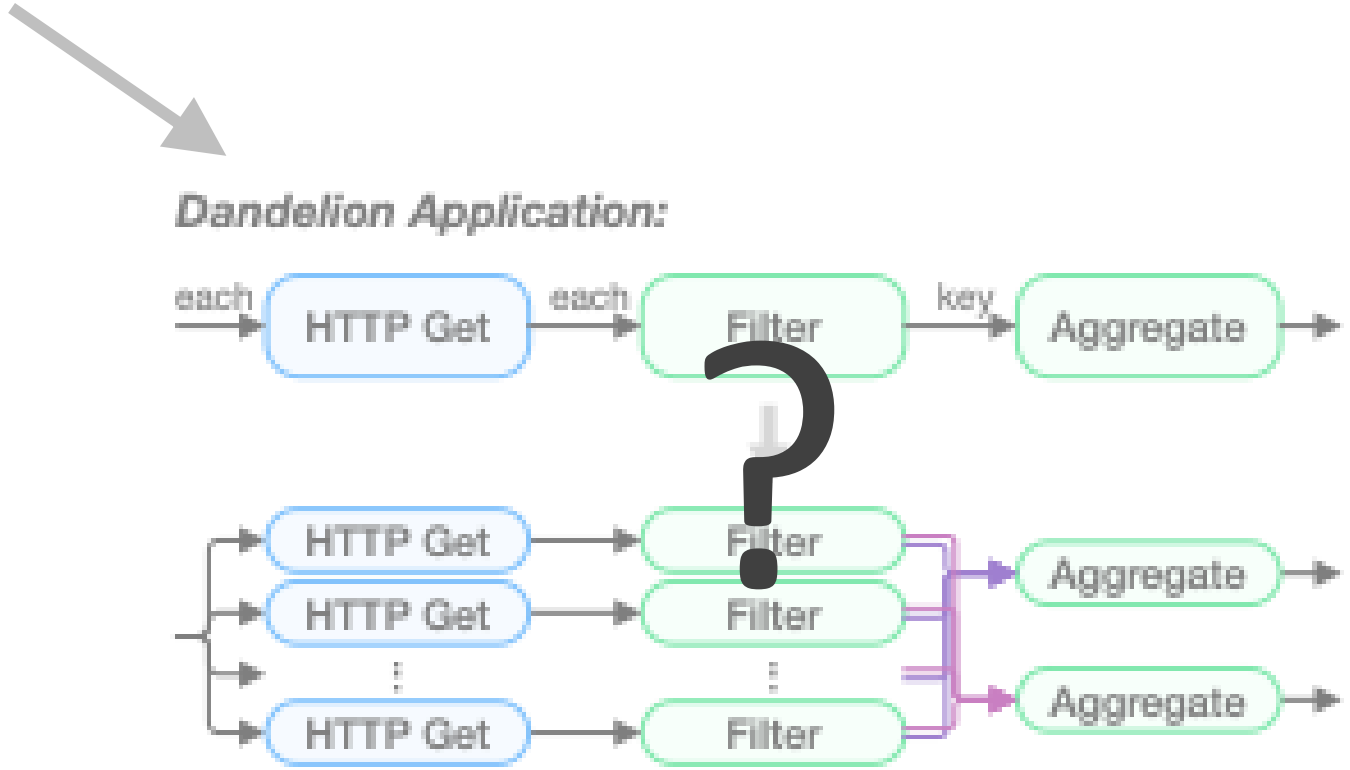
- DAG interface
- High elasticity
- Strong isolation for multitenancy



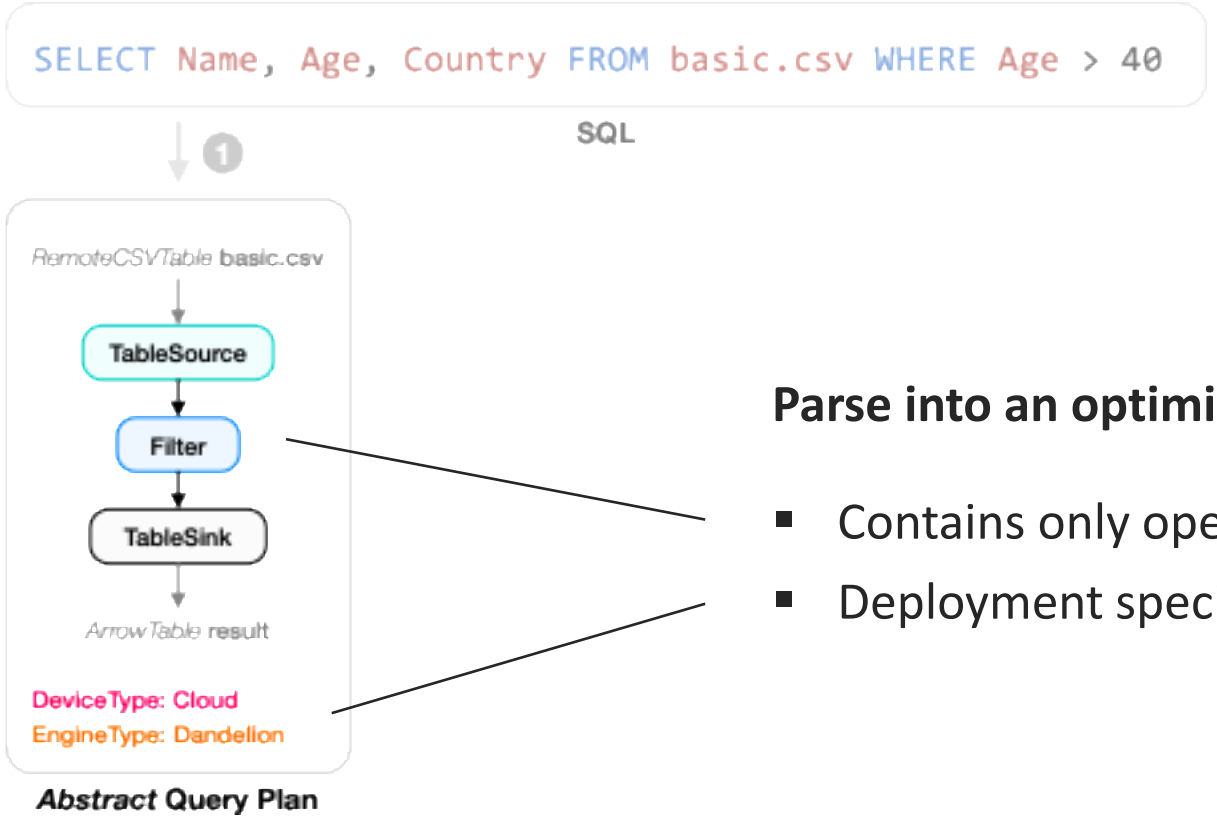
How to map Maximus query → Dandelion DAG?

```
SELECT Name, Age, Country FROM basic.csv WHERE Age > 40
```

SQL



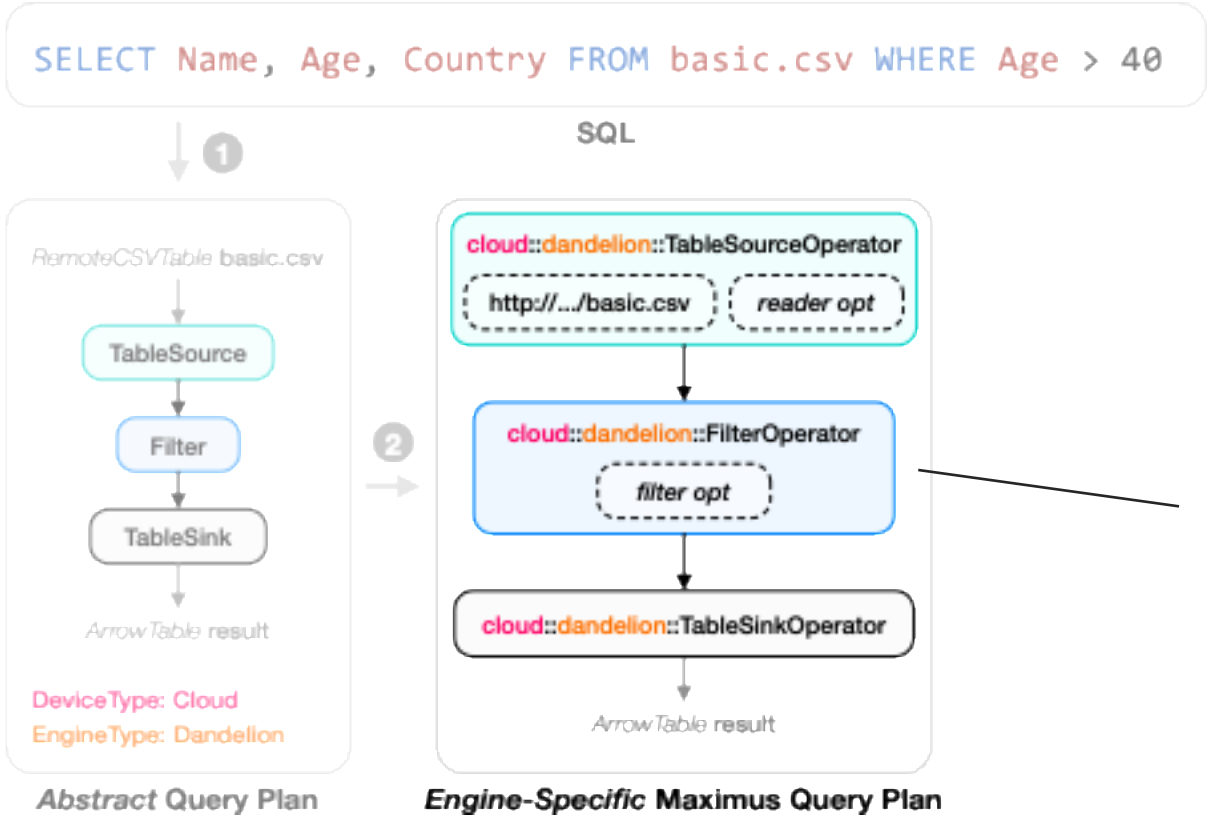
How to map Maximus query → Dandelion DAG?



Parse into an optimized *abstract* query plan

- Contains only operator types
- Deployment specification

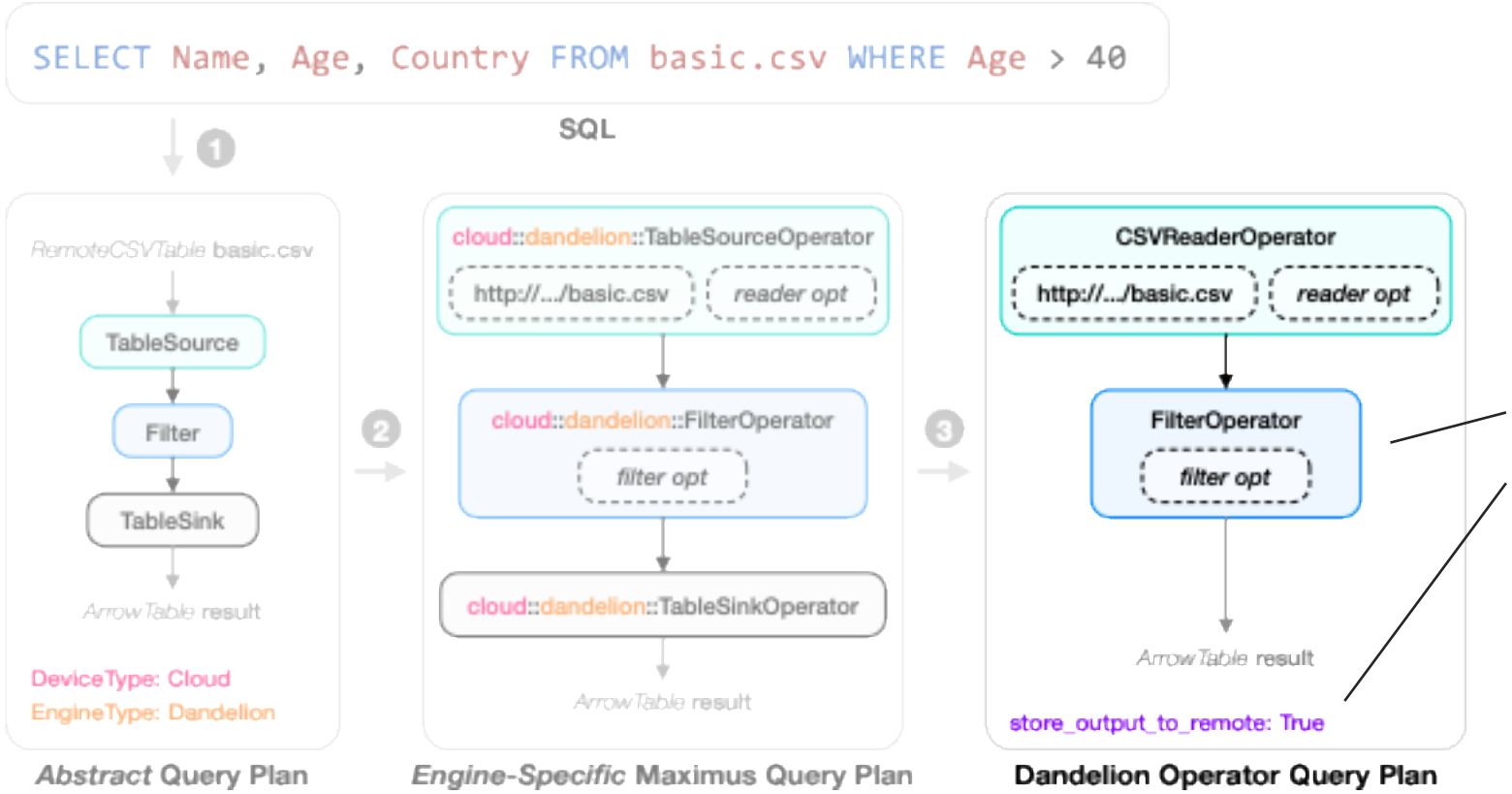
How to map Maximus query → Dandelion DAG?



Instantiate Maximus operators

- For the Dandelion engine, they are placeholders that specify the engine options

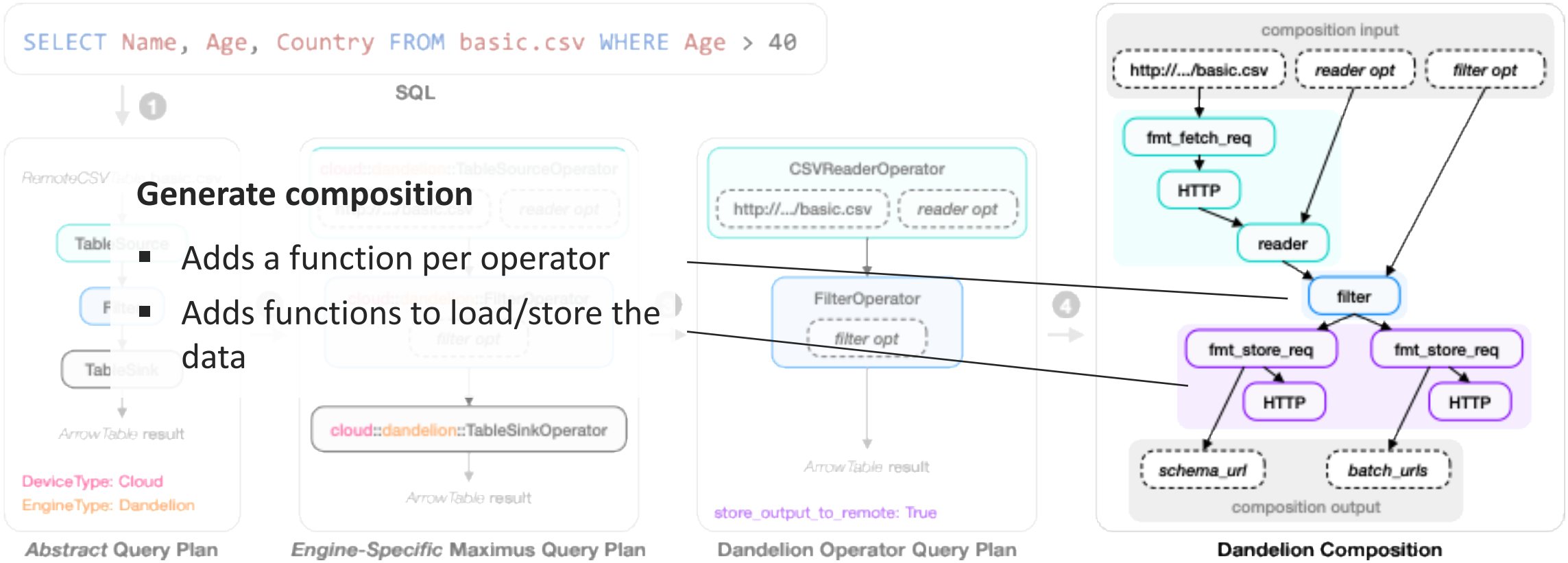
How to map Maximus query → Dandelion DAG?



Translate to Dandelion query

- Mostly one-to-one mapping
- Specifies Dandelion specific options (e.g. how data is loaded/stored)

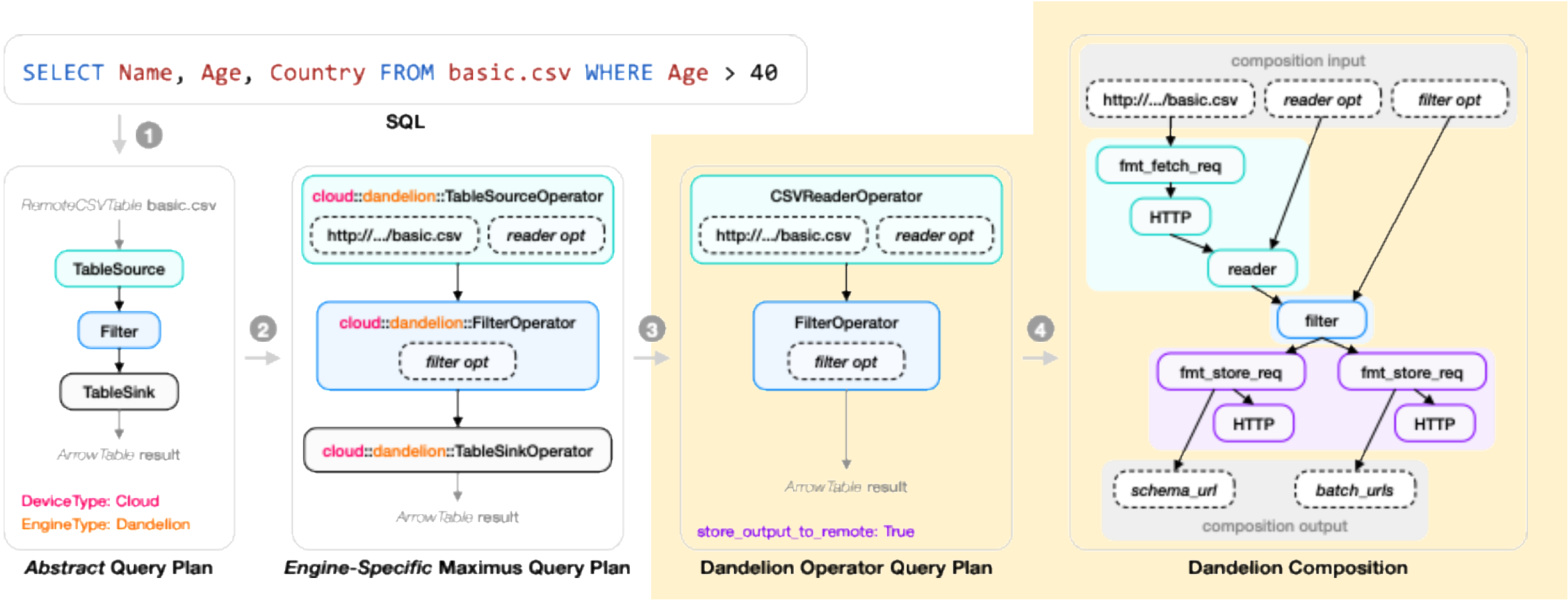
How to map Maximus query → Dandelion DAG?



Generate composition

- Adds a function per operator
- Adds functions to load/store the data

How to map Maximus query → Dandelion DAG?



specific to Dandelion

State of our current prototype

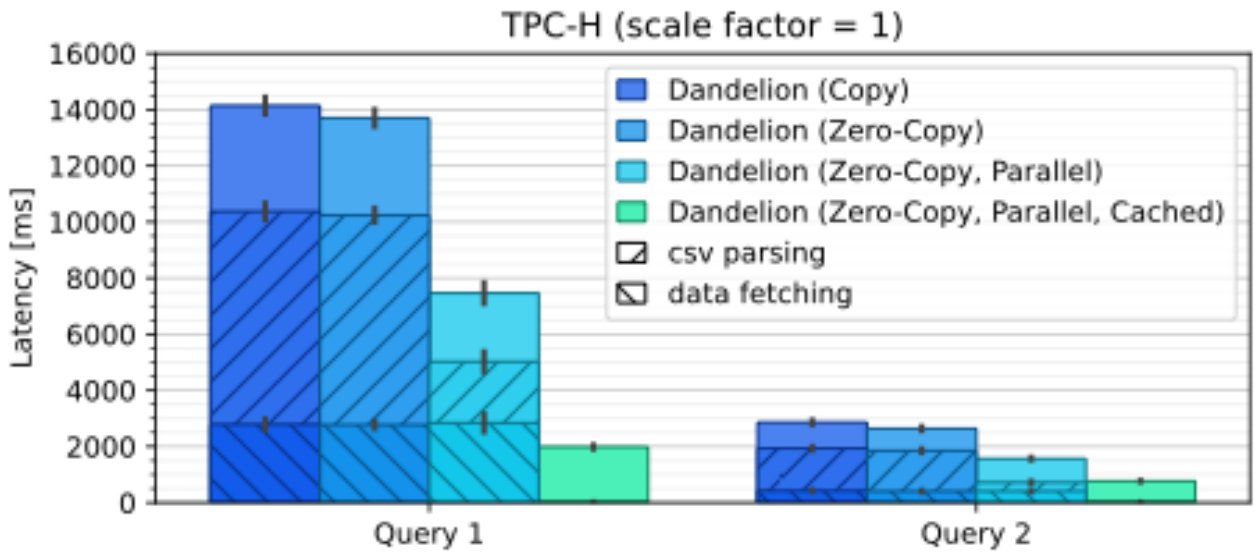
Current focus: basic functionality

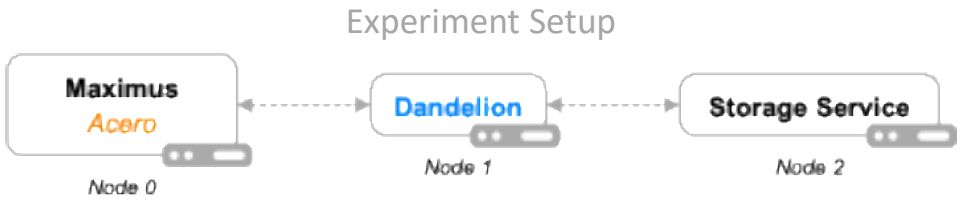
- Data-dependent parallelism (within a node, with manual file partitioning for input data)
- Basic caching support for input and intermediate data
- CPU-only execution for now
- Zero-copy between operators

Future work: performance optimization, scale-out, exploration

- Performance optimization, e.g., vectorization, zero-copy to/from network stack, etc.
- Multi-node support, scheduling policies, data placement
- Distributed cache management
- Explore co-design with storage layer, heterogeneous hardware support, …

Preliminary Feasibility Experiments





- Demonstrate basic support for parallelism, zero-copy data passing, input data caching
- Most of the time is spent fetching and parsing the input CSV data
- Parallelizing the *csv-reader* operator improves performance significantly
- Caching the parsed input data reduces the overall latency

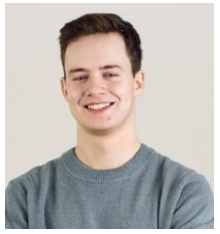
End-to-End Declarative Data Analytics:

Co-designing Engines, Interfaces, and Cloud Infrastructure

- The declarative nature of SQL ends at the query planning/optimization layer in today's engines
- **Goal**: extend the declarative nature of data analytics down to the cloud infrastructure layer
- **Idea**: combine a composite query engine (Maximus) with an elastic, delarative cloud runtime (Dandelion)
- Our early prototype shows feasibility and provides a foundation to explore open questions, e.g., policy interaction across data engine and cloud runtime layers

Systems@ETH zürich

Students at the conference ☺


Tobias Stocker


Tom Kuchler


Pinghe Li

CIDR Publication