Optimizing Data Access with Compute Offloading, Fast Hardware-Accelerated Data Transport, and Modern Query Languages

[Jayjeet Chakraborty](), [Carlos Maltzahn]()
UC Santa Cruz

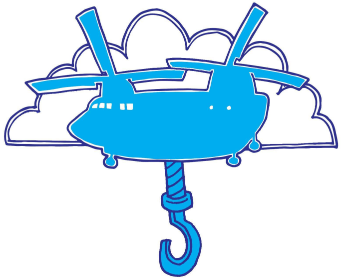
cross.ucsc.edu

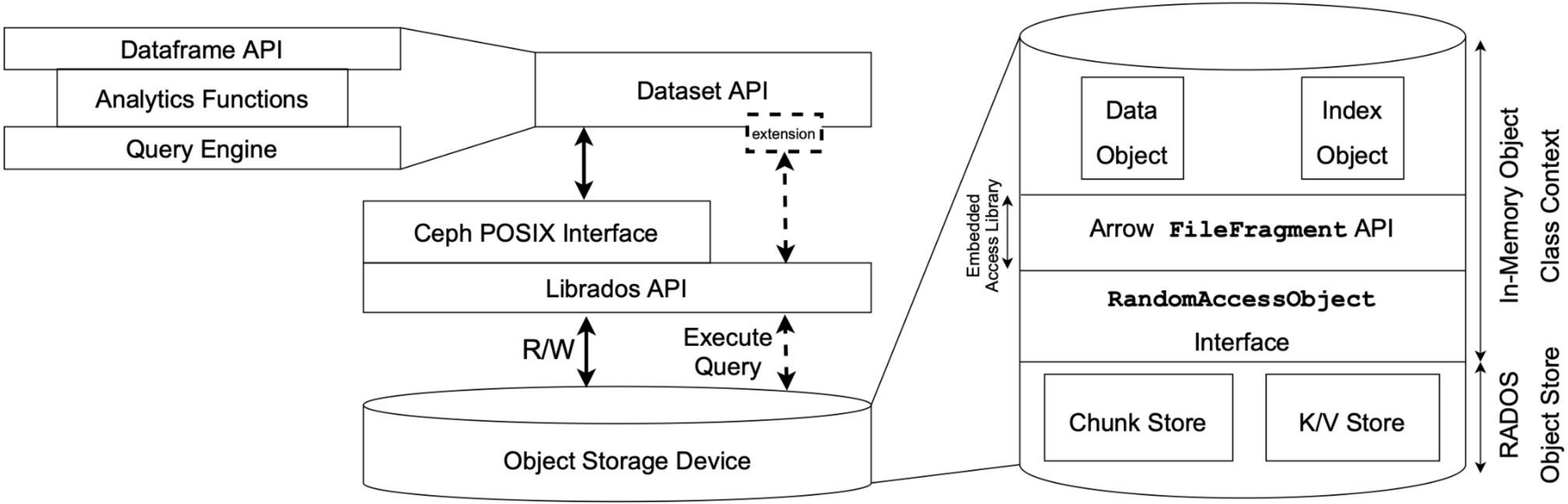
About Me

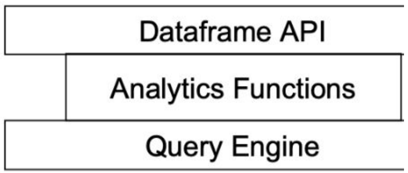
- PhD Student @ UC Santa Cruz
  - Going to 3rd yr.
  - **Advisor**: Carlos Maltzahn
  - Systems Research Lab, UCSC
- Summer Intern at InfluxData Inc.
- Former IRIS-HEP Fellow (2020/2021)
- Former GSoC student (2019)
- Co-Creator of SkyhookDM
- Researching Data management, Databases, and Storage systems

My Interests/Ongoing Work

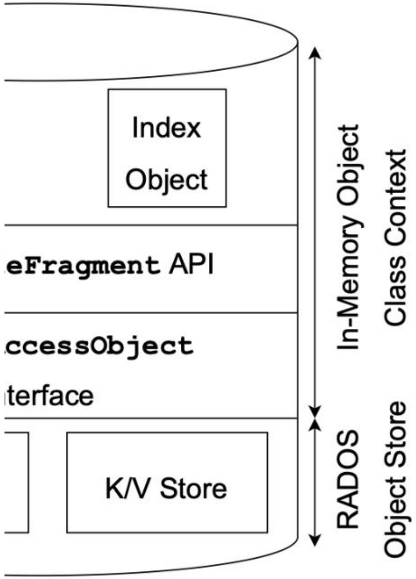
- Exploring ways to accelerate queries in data management systems
  - Computational storage:
    - Offload query execution logic to storage servers/devices
      - **Skyhook**: Apache Arrow in Ceph Object Store
        - Reduce data movement
        - Reduce metadata overload on the client
        - Low barrier to computational storage
        - Contributed to Apache Arrow open-source project last year
        - Published in CCGrid'22
    - Embedding (de)compression, (de)serialization inside Smart NICs
      - NVIDIA BlueField 2

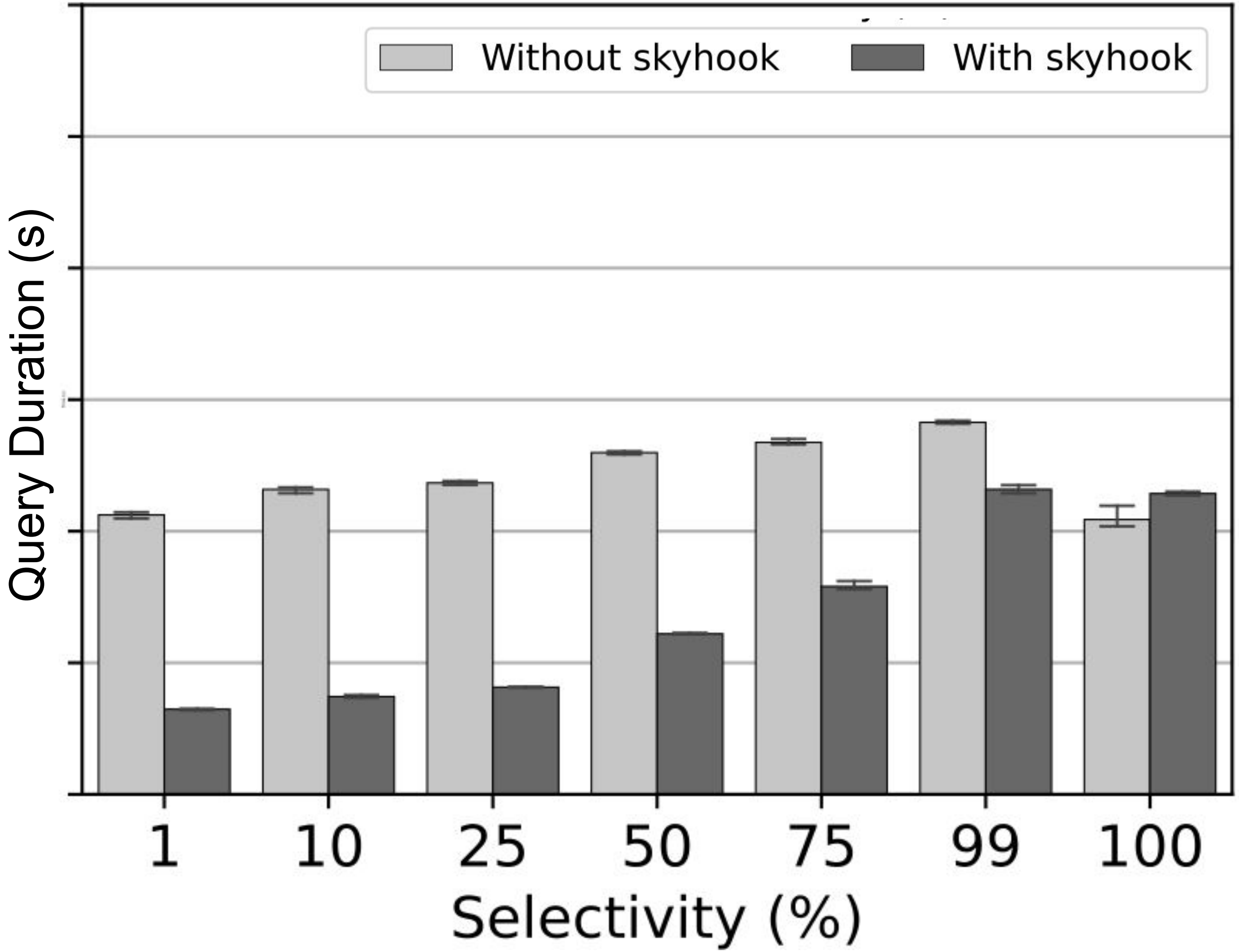



Dataframe API
Analytics Functions
Query Engine
Dataset API
extension
Ceph POSIX Interface
Librados API
R/W
Execute
Query
Object Storage Device
Data
Object
Index
Object
Embedded
Access Library
Arrow FileFragment API
RandomAccessObject
Interface
Chunk Store
K/V Store
In-Memory Object
Class Context
RADOS
Object Store



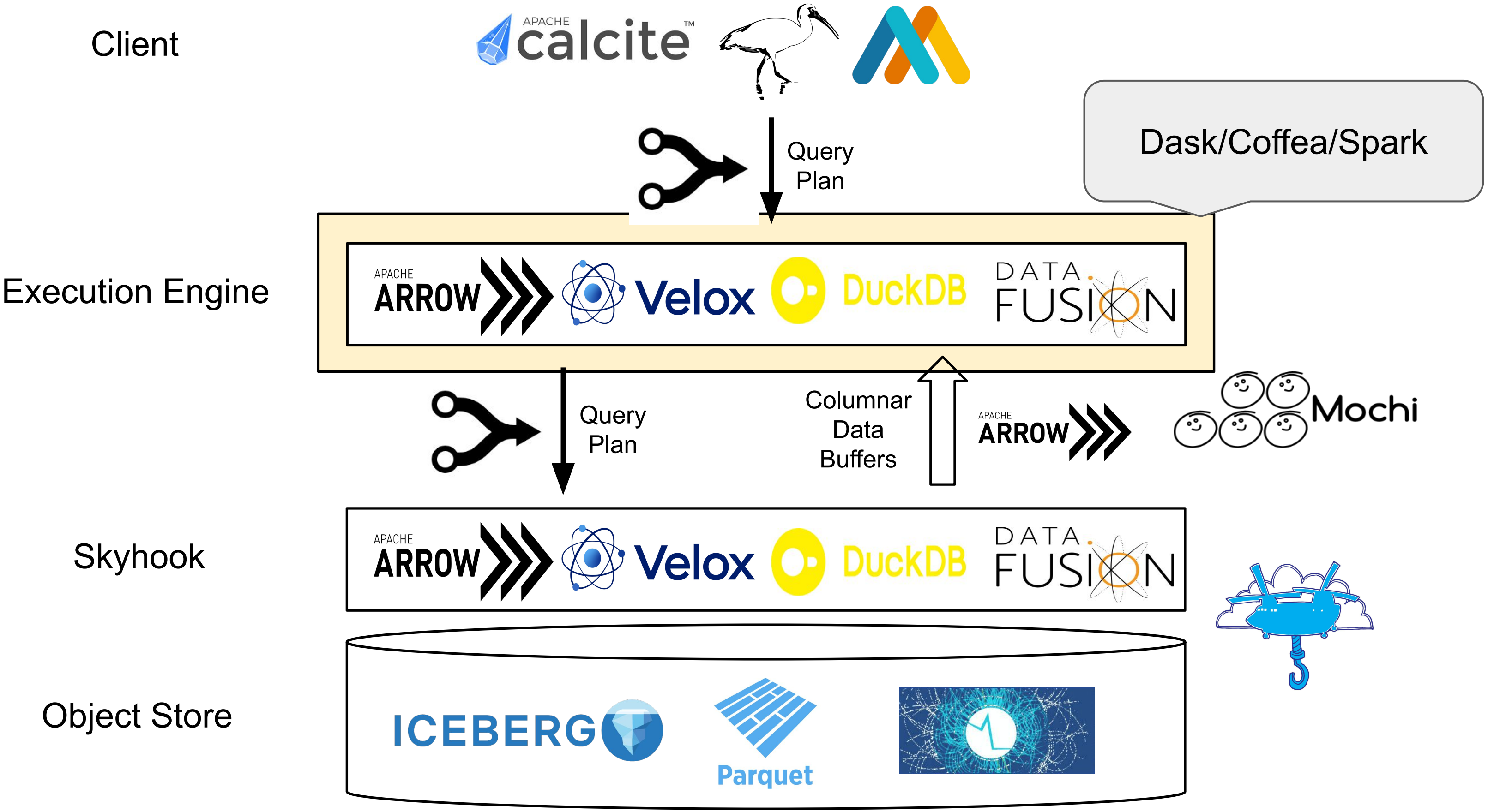

```
# Reading from Parquet
import pyarrow.dataset as ds
format_ = "parquet"
dataset = ds.dataset(
    "/dataset", format=format_
)
dataset.to_table()

# Reading from Parquet using Skyhook
import pyarrow.dataset as ds
format_ = ds.SkyhookFileFormat(
    "parquet", "/ceph.conf"
)
dataset = ds.dataset(
    "/dataset", format=format_
)
dataset.to_table()
```




Without skyhook
With skyhook
Query Duration (s)
1
10
25
50
75
99
100
Selectivity (%)

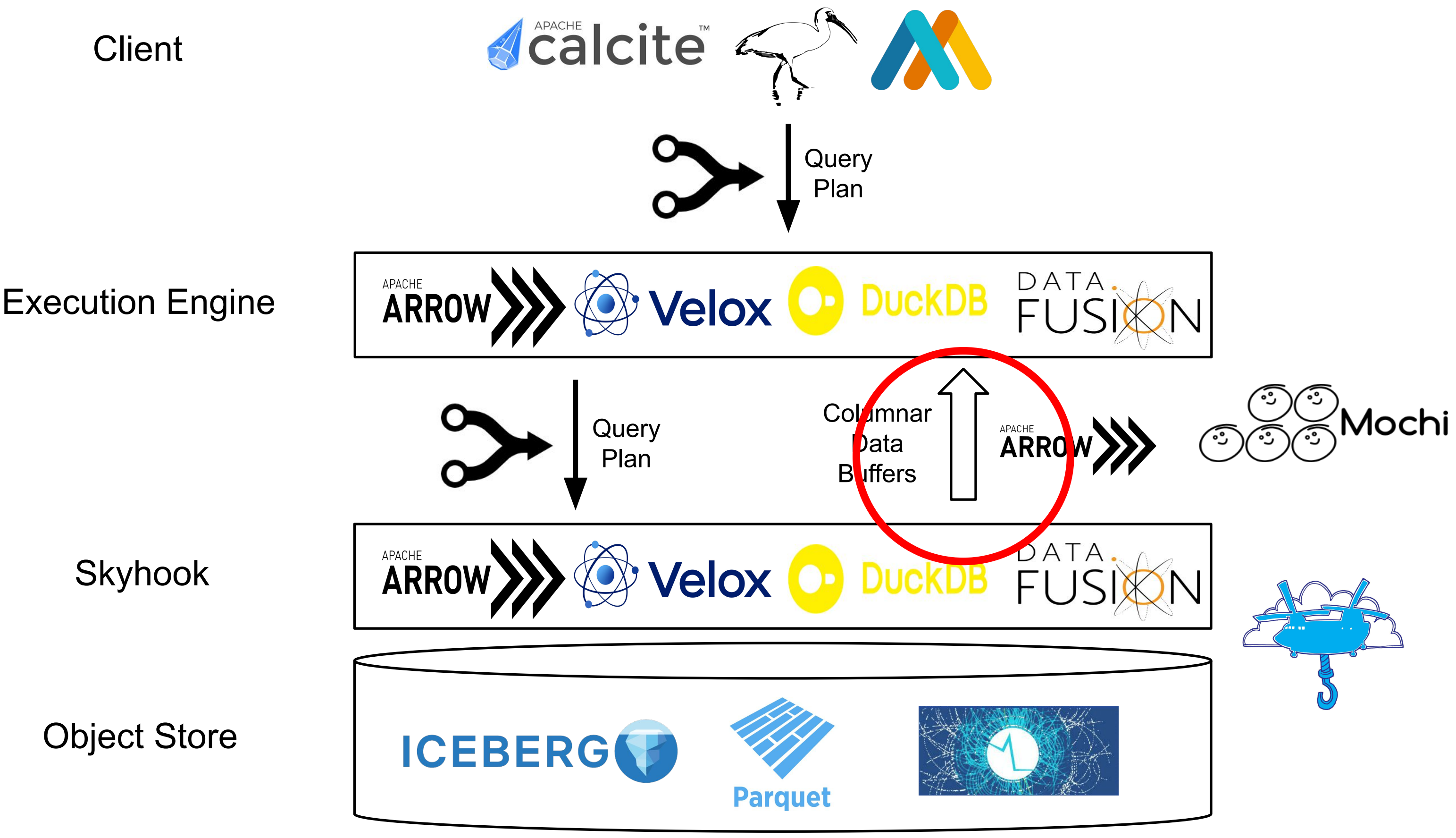
My Interests/Ongoing Work

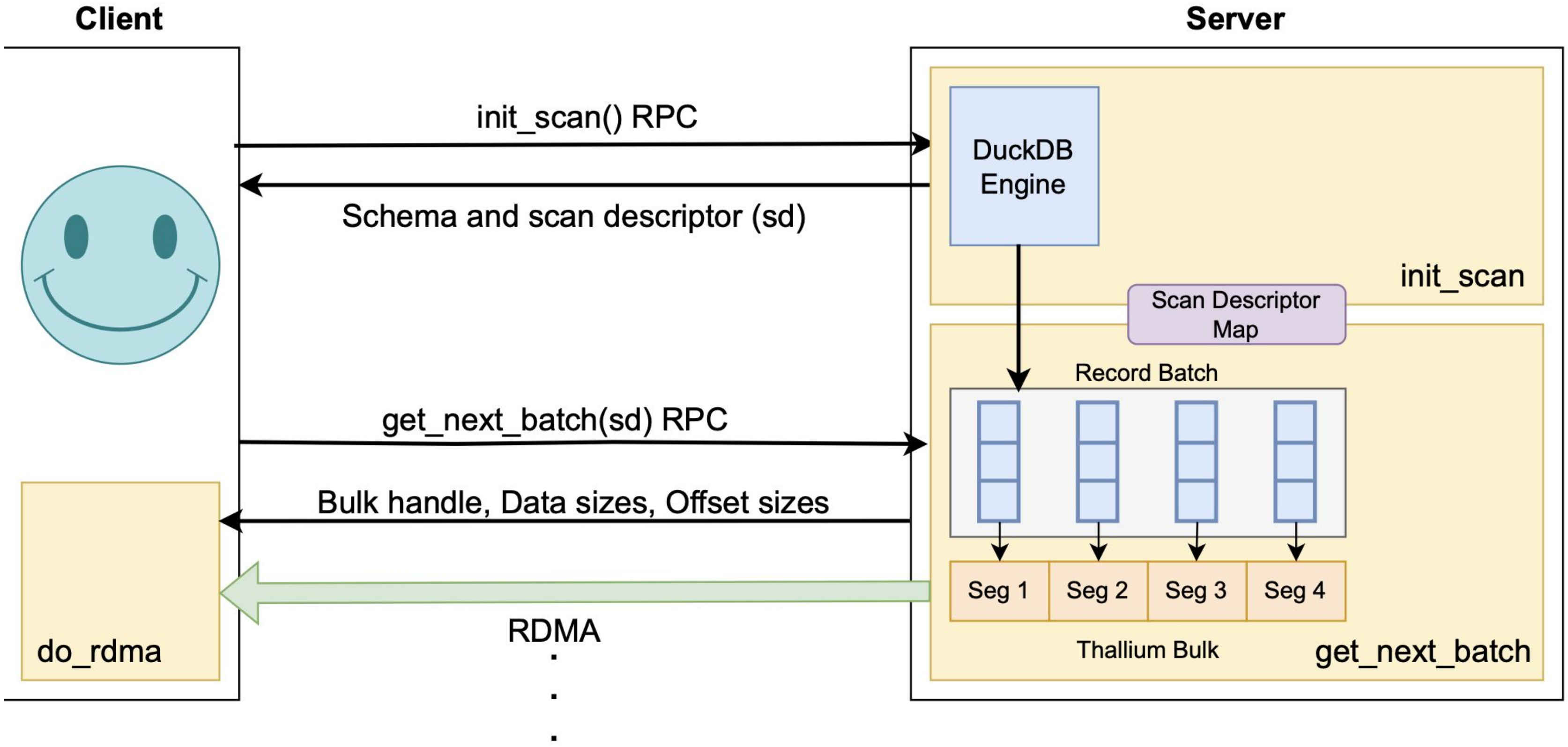
- Deconstructed "Data Management"
    - Pick and choose your own stack
    - No more redundant data management systems
    - Enable standardization
    - **Build your custom data system with modular interoperable frameworks**
        - Query languages
        - Query Interfaces and Compiler
        - Task schedulers
        - Query execution engines
        - Storage systems
        - File formats
    - We aim to prototype a initial version of such a system using the Python SDKs in each layer

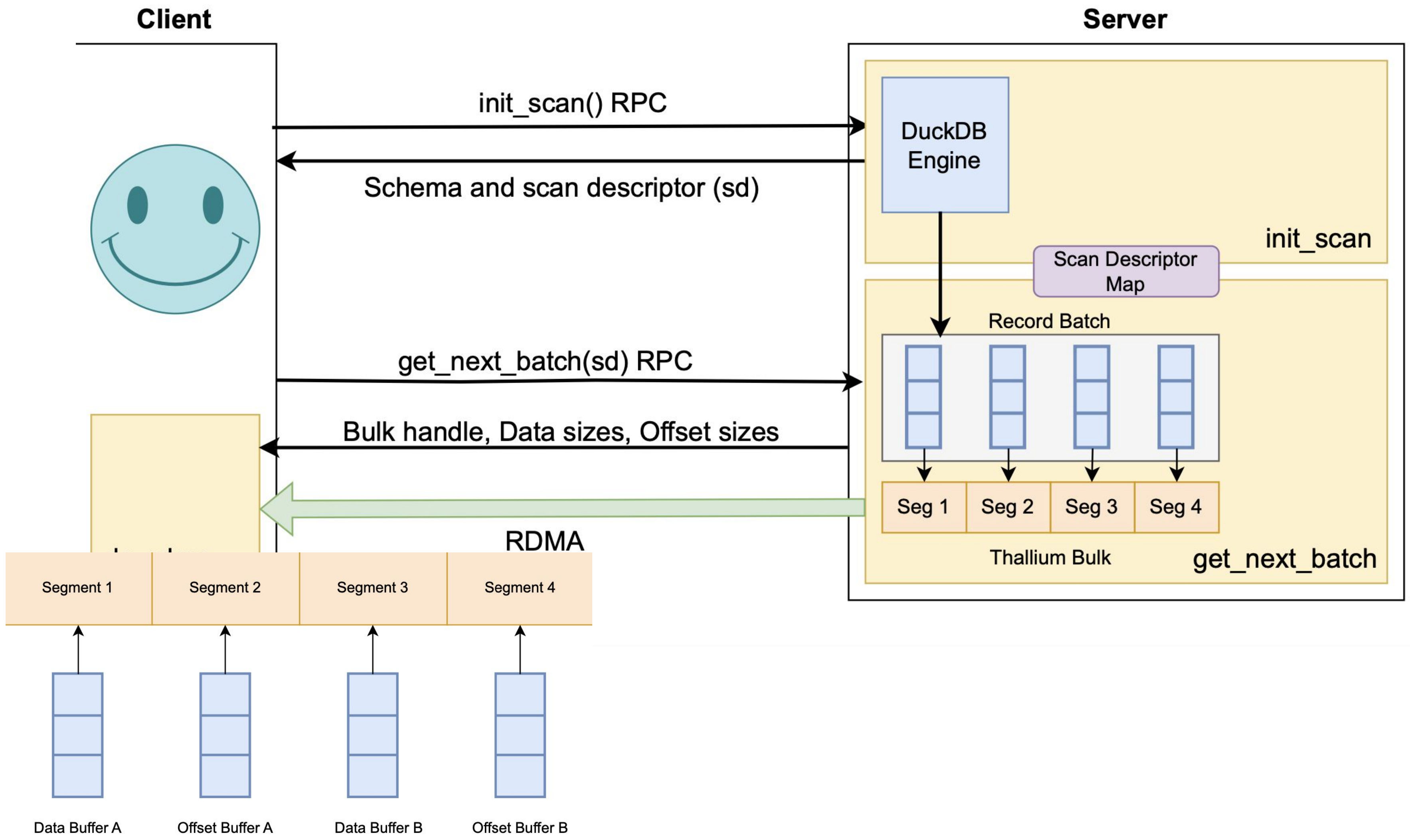

Client
APACHE calcite
Query Plan
Dask/Coffea/Spark
Execution Engine
APACHE ARROW
Velox
DuckDB
DATA FUSION
Query Plan
Columnar Data Buffers
APACHE ARROW
Mochi
Skyhook
APACHE ARROW
Velox
DuckDB
DATA FUSION
Object Store
ICEBERG
Parquet

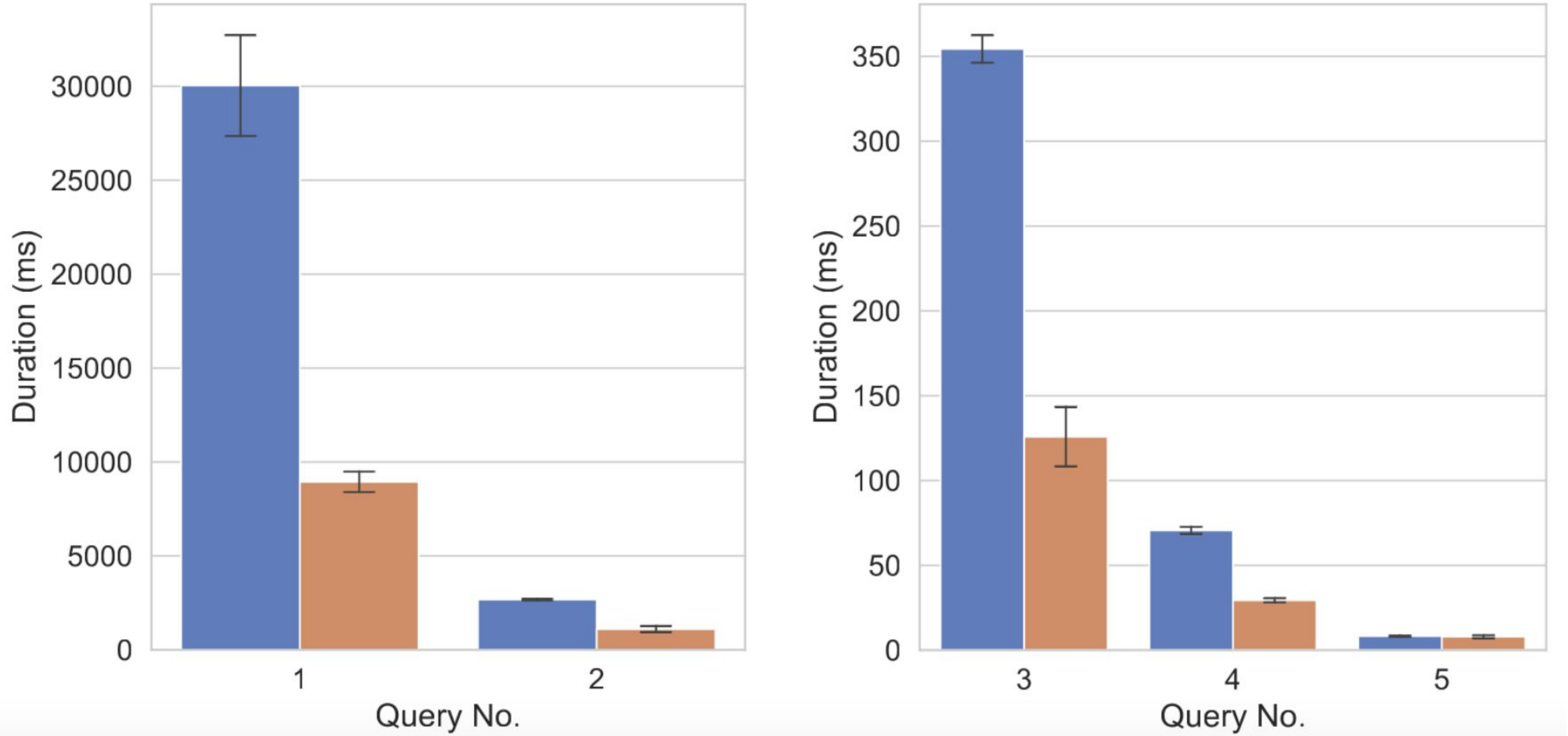
My Interests/Ongoing Work

- Leveraging modern networking devices
  - RDMA-enabled NICs common in Data centers
    - ConnectX-3/5/6
    - Upto 100 Gbps
  - Move from TCP/IP to RDMA for fast data transfers
    - Avoid copying and serialization overhead of TCP/IP
    - Use data transport frameworks used in HPC
      - Mochi Thallium from Argonne National Labs
    - **Thallus: Faster Columnar (Apache Arrow) Data Transport using RDMA**
      - Arrow Flight (gRPC-based) as our baseline
      - Preparing for submission


Client
APACHE calcite
Query Plan
Execution Engine
APACHE ARROW
Velox
DuckDB
DATA FUSION
Query Plan
Columnar Data Buffers
APACHE ARROW
Mochi
Skyhook
APACHE ARROW
Velox
DuckDB
DATA FUSION
Object Store
ICEBERG
Parquet


Client
Server
init_scan() RPC
Schema and scan descriptor (sd)
DuckDB Engine
init_scan
Scan Descriptor Map
Record Batch
get_next_batch(sd) RPC
Bulk handle, Data sizes, Offset sizes
Seg 1
Seg 2
Seg 3
Seg 4
RDMA
do_rdma
Thallium Bulk
get_next_batch


Client
Server
init_scan() RPC
Schema and scan descriptor (sd)
DuckDB Engine
init_scan
Scan Descriptor Map
Record Batch
get_next_batch(sd) RPC
Bulk handle, Data sizes, Offset sizes
Seg 1
Seg 2
Seg 3
Seg 4
RDMA
Thallium Bulk
get_next_batch
Segment 1
Segment 2
Segment 3
Segment 4
Data Buffer A
Offset Buffer A
Data Buffer B
Offset Buffer B


Duration (ms)
30000
25000
20000
15000
10000
5000
0
1
2
Query No.
Duration (ms)
350
300
250
200
150
100
50
0
3
4
5
Query No.

My Interests/Ongoing Work

- Alternative query languages for HEP data
  - Malloy QL, project by Google
    - **Designed for handling hyper-dimensional data**
    - Generates the most optimized SQL possible
    - Much simpler syntax than SQL, better UX
    - Plugins for BigQuery, DuckDB, PostGres
    - 2 parts to every query:
      - Source: A table or computation result set
      - Query: Pipelined set of stages defining a query operation
    - Python package for Malloy: malloy-py

Evaluating Query Languages and Systems for High-Energy Physics Data

[Extended Version]

Dan Graur
Department of Computer Science
ETH Zurich
dan.graur@inf.ethz.ch

Ingo Müller
Department of Computer Science
ETH Zurich
ingo.mueller@inf.ethz.ch

Mason Proffitt
Department of Physics
University of Washington
masonLp@uw.edu

Ghislain Fourny
Department of Computer Science
ETH Zurich
ghislain.fourny@inf.ethz.ch

Gordon T. Watts
Department of Physics
University of Washington
gwatts@uw.edu

Gustavo Alonso
Department of Computer Science
ETH Zurich
alonso@inf.ethz.ch

ABSTRACT

In the domain of high-energy physics (HEP), query languages in general and SQL in particular have found limited acceptance. This is surprising since HEP data analysis matches the SQL model well: the data is fully structured and queried using mostly standard operators. To gain insights on why this is the case, we perform a comprehensive analysis of six diverse, general-purpose data processing platforms using an HEP benchmark. The result of the evaluation is an interesting and rather complex picture of existing solutions: Their query languages vary greatly in how natural and concise HEP query patterns can be expressed. Furthermore, most of them

only a small subset of the available attributes, derivation of additional measures (potentially by joining and reducing the sequences *within the same event*), and selection of an interesting subset of events, which are then summarized using a reduction. HEP data is thus stored and analyzed in non-first normal form (NF^2)—a feature that early database systems did not support and thus the main reason why relational engines were rejected by physicists historically (along with the lack of support for used-defined code [39]).

Nowadays, most particle physicists work with a domain-specific system called the ROOT framework [4, 12], and increasingly so with its new RDataFrame interface [27]. In ROOT, queries are writ-

Handwritten SQL to Malloy for **Q4**

```sql
SELECT
    FLOOR((
        CASE
        WHEN MET.pt < 0 THEN -1
        WHEN MET.pt > 2000 THEN 2001
        ELSE MET.pt
        END) / 20) * 20 + 10 AS x,
    COUNT(*) AS y
FROM '{dataset_path}'
WHERE (
    SELECT
        COUNT(*)
    FROM UNNEST(Jet)
    WHERE Jet.pt > 40
) > 1
GROUP BY FLOOR((
CASE
    WHEN MET.pt < 0 THEN -1
    WHEN MET.pt > 2000 THEN 2001
    ELSE MET.pt
END) / 20) * 20 + 10
ORDER BY x;
```

Preview

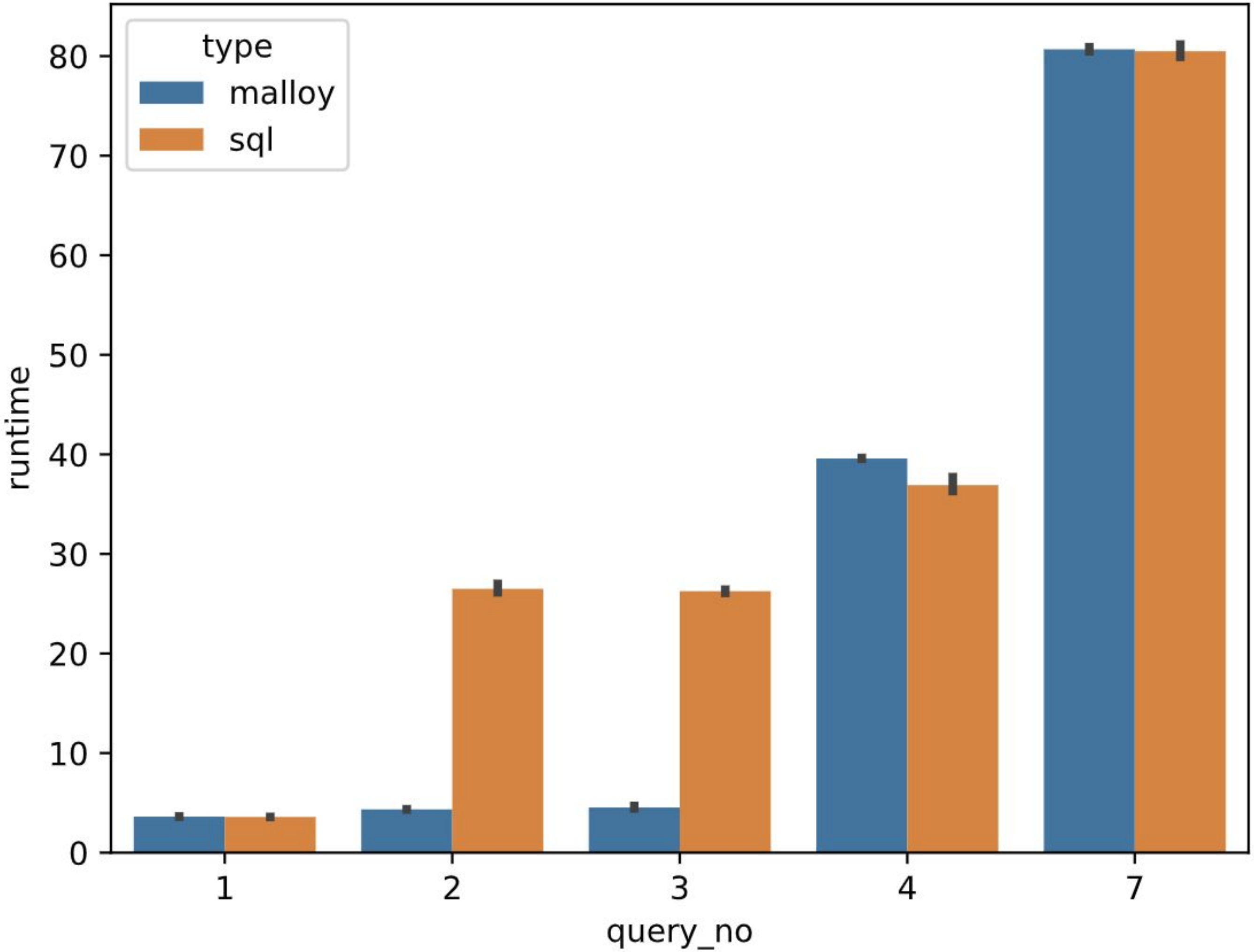
```
source: hep is table('duckdb:../hep.parquet') {
 declare: x is
          floor((pick -1  when MET.pt < 0
          pick 2001 when MET.pt > 2000
          else MET.pt) / 20) * 20 + 10
}
```

Run

```
query: hep -> {
  declare: t is Jet.count() {? Jet.pt > 40} > 1
  group_by: x, event
  where: t
}
-> {
  group_by: x
  aggregate: y is count()
  order_by: x
}
```

Equivalent Malloy gen. SQL for **Q4**

```
WITH __stage0 AS (
  SELECT
    ((floor((
      CASE WHEN hep.MET."pt"<0 THEN -1
      WHEN hep.MET."pt">2000 THEN 2001
      ELSE hep.MET."pt" END)*1.0/20))*20)+10
    as "x",
     hep."uid" as "uid"
  FROM (SELECT gen_random_uuid() uid, * FROM '{dataset_path}') as hep
  LEFT JOIN (select UNNEST(generate_series(1,
          100000, --
          -- (SELECT genres_length FROM movies limit 1),
          1)) as __row_id) as Jet_0 ON  Jet_0.__row_id <= array_length(hep."Jet")
  GROUP BY 2, 1
  HAVING (COUNT( CASE WHEN hep.Jet[Jet_0.__row_id]."pt">40 THEN 1 END)>1)
)

SELECT
   base."x" as "x",
   COUNT( 1) as "y"
FROM __stage0 as base
GROUP BY 1
ORDER BY 1 asc NULLS LAST
```


type
malloy
sql
runtime
0
10
20
30
40
50
60
70
80
1
2
3
4
7
query_no

Goals

- Leverage modern hardware and protocols in data management
- Expose complex functionality using simple interfaces and APIs
- World is moving towards composable data management, stay ahead !
- Prepare for the **Analysis Grand Challenge**