DOMA R/D and Analysis Grand Challenge

Jayjeet Chakraborty, Carlos Maltzahn
UC Santa Cruz

cross.ucsc.edu

Today's Agenda

- HL-LHC and DOMA
- Malloy QL for HEP data analysis
- Data management using Skyhook
- Ongoing Work..

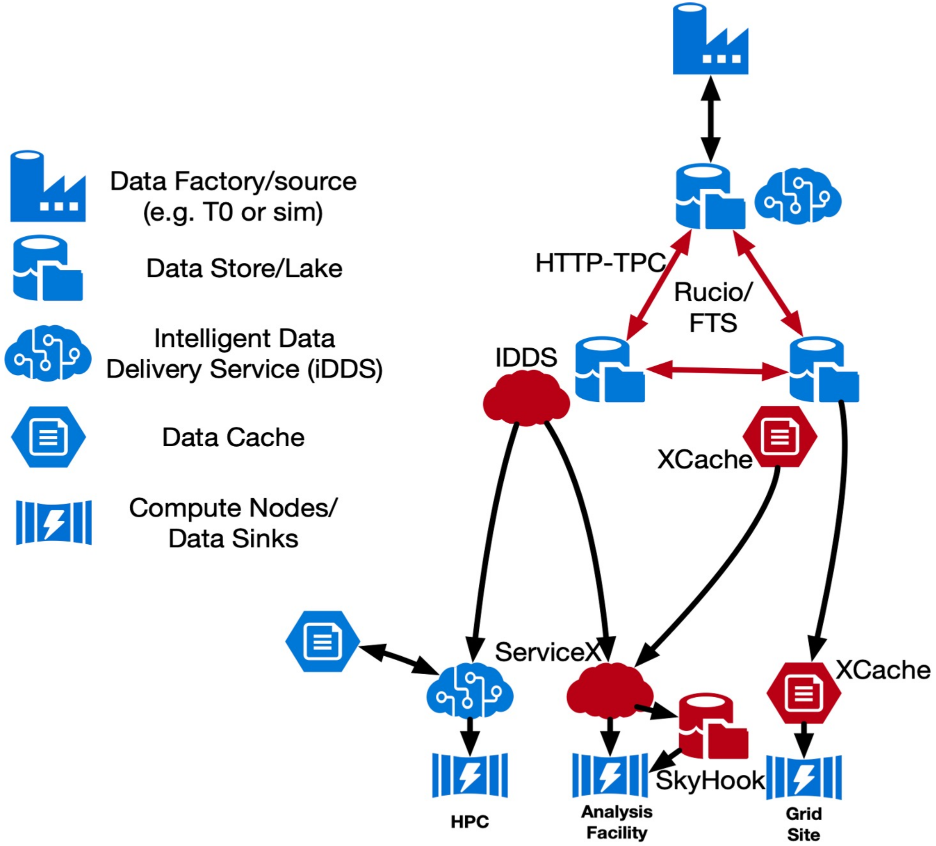
The HL-LHC

- High-Luminosity Large Hadron Collider
- Major upgrade to the original LHC
- To be started in around 2028-2029

- 5-7.5x increase in the number of collisions
- Will generate an increased number of events, about 30x increase
- Total working dataset sizes will be in exabytes

DOMA Team at IRIS-HEP

- Working on R/D of data delivery, access, organization, and management technologies
- Several projects within DOMA:
  - XCache
  - Coffea
  - IDDS
  - TPC
  - SeviceX
  - Modelling Data Workflows
  - **SkyhookDM**



Malloy QL for HEP data analysis

Data sample emitted by the HL-LHC

- Fields of primitive types (int, float, etc), structs, and list<structs>
- Primary key: **event**
- Originally stored in ROOT files, but we use Parquet for analysis

Field name	Type
run	INTEGER
luminosityBlock	INTEGER
event	INTEGER
▸ MET	RECORD
▸ HLT	RECORD
▸ PV	RECORD
▸ Muon	RECORD
▸ Electron	RECORD
▸ Photon	RECORD
▸ Jet	RECORD
▸ Tau	RECORD

DETAILS PREVIEW LINEAGE

event	INTEGER
▾ MET	RECORD
pt	FLOAT
phi	FLOAT
sumet	FLOAT
significance	FLOAT
CovXX	FLOAT
CovXY	FLOAT
CovYY	FLOAT
▸ HLT	RECORD
▸ PV	RECORD
▾ Muon	RECORD
▾ list	RECORD
▾ element	RECORD
pt	FLOAT
eta	FLOAT
phi	FLOAT
mass	FLOAT
charge	INTEGER
pfRelIso03_all	FLOAT

Row	run	luminosityBlock	event	MET.pt	MET.phi	MET.sumet	Msignificance	MET.CovXX	MET.CovXY	MET.CovYY
1	194711	299	263599382	8.16232967...	1.53420329...	358.935546...	0.60561221...	108.264114...	-9.5036792...	111.555229...
2	194711	299	263599382	8.16232967...	1.53420329...	358.935546...	0.60561221...	108.264114...	-9.5036792...	111.555229...
3	194711	299	263599382	8.16232967...	1.53420329...	358.935546...	0.60561221...	108.264114...	-9.5036792...	111.555229...
4	194711	299	263599382	8.16232967...	1.53420329...	358.935546...	0.60561221...	108.264114...	-9.5036792...	111.555229...
5	194711	299	263599382	8.16232967...	1.53420329...	358.935546...	0.60561221...	108.264114...	-9.5036792...	111.555229...
6	194711	299	263599382	8.16232967...	1.53420329...	358.935546...	0.60561221...	108.264114...	-9.5036792...	111.555229...
7	194711	299	263599382	8.16232967...	1.53420329...	358.935546...	0.60561221...	108.264114...	-9.5036792...	111.555229...
8	194711	299	263599382	8.16232967...	1.53420329...	358.935546...	0.60561221...	108.264114...	-9.5036792...	111.555229...

Present Query Languages in HEP analysis

- Basic requirements:
  - Independent on the underlying file format or data structures
  - Identical query interface irrespective of whether executing locally or remotely, or single or multiple machines
- Examples:
  - Func ADL (Python)
  - Groot (Go)
  - RDataFrame (C++)
  - NAIL (Natural Analysis Implementation Language) (Python)
  - SQL

Example Analysis Query in SQL

```
SELECT
 HistogramBin(MET.pt, 0, 2000, 100) AS x,
 COUNT(*) AS y
FROM table
WHERE ARRAY_LENGTH(Muon) >= 2 AND
 (SELECT COUNT(*) AS mass
  FROM UNNEST(Muon) m1 WITH OFFSET i
  CROSS JOIN UNNEST(Muon) m2 WITH OFFSET j
  WHERE
    m1.charge <> m2.charge AND i < j AND
    SQRT(2*m1.pt*m2.pt*(COSH(m1.eta-m2.eta)-COS(m1.phi-m2.phi))) BETWEEN 60
AND 120) > 0
GROUP BY x
ORDER BY x
```

x	y
10.0	39
30.0	15
50.0	2
70.0	1

Example Analysis Query in Python

```
class Q5Processor(processor.ProcessorABC):
    def process(self, events):
        mupair = ak.combinations(events.Muon, 2)
        with np.errstate(invalid="ignore"):
            pairmass = (mupair.slot0 + mupair.slot1).mass
        goodevent = ak.any(
            (pairmass > 60)
            & (pairmass < 120)
            & (mupair.slot0.charge == -mupair.slot1.charge),
            axis=1,
        )
        return (
            hist.Hist.new.Reg(100, 0, 200, name="met", label="$E_{T}^{miss}$ [GeV]")
            .Double()
            .fill(events[goodevent].MET.pt)
        )

    def postprocess(self, accumulator):
        return accumulator


out, metrics = run(Q5Processor)
out.plot1d()
metrics
```

x	y
10.0	39
30.0	15
50.0	2
70.0	1

Example Analysis Query in Python

```
class Q5Processor(processor.ProcessorABC):
    def process(self, events):
        mupair = ak.combinations(events.Muon, 2)
        with np.errstate(invalid="ignore"):
            pairmass = (mupair.sl
        goodevent = ak.any(
            (pairmass > 60)
            & (pairmass < 120)
            & (mupair.slot0.char
            axis=1,
        )
        return (
            hist.Hist.new.Reg(1(
            .Double()
            .fill(events[goodev
        )

    def postprocess(self, accumu
        return accumulator


out, metrics = run(Q5Processor)
out.plot1d()
metrics
```

x	y
10.0	39
	15
	2
	1

Learning and using different language and framework specific query language can be an extra overhead for Physicists. What if there was a single simple language ?

Malloy

- An experimental language for describing data relationships and transformations
- Allows writing better understandable queries using uncomplicated semantics
- Aims to generate the most optimized SQL query possible for performance
- Works with BigQuery, Postgres, and DuckDB so far

Brief introduction to Malloy's syntax

- **Source**: a table or a computation result set
- **Query**: a pipelined set of stages each stage defining a query operation

Preview

```
source: hep is table('duckdb:../hep.parquet')
```

Run

```
query: query1 is hep -> {
    group_by: x is
        floor((pick -1  when MET.pt < 0
        pick 2001 when MET.pt > 2000
        else MET.pt) / 20) * 20 + 10,
    aggregate: y is count(*),
    order_by: x
} -> {
    project: x, y
}
```

SQL to Malloy translation: **#Q2**

```sql
SELECT
  FLOOR((
    CASE
      WHEN j.pt < 15 THEN 14.99
      WHEN j.pt > 60 THEN 60.01
      ELSE j.pt
    END - 0.15) / 0.45) * 0.45 + 0.375 AS x,
  COUNT(*) AS y
FROM '{dataset_path}'
CROSS JOIN UNNEST(Jet) AS _j(j)
GROUP BY FLOOR((
    CASE
      WHEN j.pt < 15 THEN 14.99
      WHEN j.pt > 60 THEN 60.01
      ELSE j.pt
    END - 0.15) / 0.45) * 0.45 + 0.375
ORDER BY x;
```

Run

```
sql: cross_join_sql is {
    select: """
        SELECT
            unnest(Jet) as J,
            MET
        FROM read_parquet('../hep.parquet')
    """
    connection: "duckdb"
}
```

Run

```
query: query2 is from_sql(cross_join_sql) -> {
    group_by: x is
        floor(((pick 14.99  when J.pt < 15
        pick 60.01 when J.pt > 60
        else J.pt) - 0.15) / 0.45) * 0.45 + 0.375
    aggregate: y is count(*)
    order_by: x
} -> {
    project: x, y
}
```

SQL to Malloy translation: **#Q2**

```sql
SELECT
  FLOOR((
    CASE
      WHEN j.pt < 15 THEN 14.99
      WHEN j.pt > 60 THEN 60.01
      ELSE j.pt
    END - 0.15) / 0.45) * 0.45 + 0.375 AS x,
  COUNT(*) AS y
FROM '{dataset_path}'
CROSS JOIN UNNEST(Jet) AS _j(j)
GROUP BY FLOOR((
    CASE
      WHEN j.pt < 15 THEN 14.99
      WHEN j.pt > 60 THEN 60.01
      ELSE j.pt
    END - 0.15) / 0.45) * 0.45 + 0.375
ORDER BY x;
```

```sql
WITH __stage0 AS (
  SELECT
    (
     (floor((((
      CASE WHEN cross_join_sql.J."pt"<15 THEN 14.99
      WHEN cross_join_sql.J."pt">60 THEN 60.01
      ELSE cross_join_sql.J."pt" END))-0.15)
     *1.0/0.45))*0.45)+0.375 as "x",
    COUNT( 1) as "y"
  FROM (
        SELECT
          unnest(Jet) as J,
          MET
        FROM '{dataset_path}'
     ) as cross_join_sql
  GROUP BY 1
  ORDER BY 1 ASC NULLS LAST
)
SELECT
   base."x" as "x",
   base."y" as "y"
FROM __stage0 as base
```

SQL to Malloy translation: **#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
}
```

SQL to Malloy translation: **#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;
```

```sql
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
```

Current limitations of Malloy

- Many in-built engine specific functions aren’t implemented yet
  - Some functions such as those with lambda expressions also need language parser updates
- Bugs in handling lists
- No support for UDFs of any form
- Bugs in handling **struct** type field
- No support for substrait plan generation

Our contributions

- Many in-built engine specific functions aren't implemented yet
  - Some functions such as those with lambda expressions also need language parser updates
- Bugs in handling **lists**
- No support for UDFs of any form
- Bugs in handling **struct** type field
- No support for substrait plan generation

Workload: ADL benchmarks

is-hep-benchmark-athena / queries /

ingomueller-net Fix computation of pt of tri-jet in Q6-1.

Name	Last commit message
..	
query-1	Fix histogram bin computation.
query-2	Fix histogram bin computation.
query-3	Fix histogram bin computation.
query-4	Fix histogram bin computation.
query-5	Fix computation of invatiant ma
query-6-1	Fix computation of pt of tri-jet i
query-6-2	Fix histogram bin computation.
query-7	Fix histogram bin computation.
query-8	Fix histogram bin computation.

[cs.DB] 30 Oct 2021

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

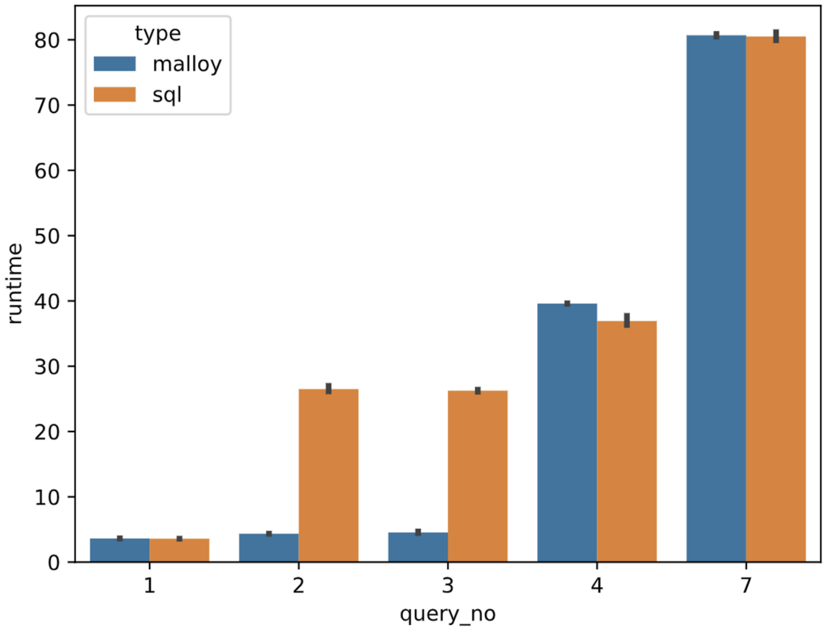
In the domain of high-energy physics (HEP), query languages in general and SQL in particular have found limited acceptance. This is surprising since HEP data analysis matches the SQL model well: the data is fully structured and queried using mostly standard operators. To gain insights on why this is 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-

Benchmarks

Note: Malloy is not particularly designed for better performance, it just tries to generate the most optimized SQL possible



Data Management using Skyhook

What is Skyhook ?

- An open-source project aiming to bridge the gap between compute and data
- A data management system that
  - Can accelerate queries by offloading parts of query to the storage servers
  - Provides a bunch of open-source choices for
    - Query interfaces
    - Execution engines
    - Object storage systems
    - File/Table formats
    - Communication/Transport protocols
  - Presents a lower barrier to computational storage as compared to CSDs

Query Interface and Compiler

```
SELECT g, SUM(y) AS z
FROM ‘s3://bucket’
WHERE x > 99
GROUP BY g
ORDER BY z
```

```
relations:
    read: ‘s3://bucket’
    project: g, z
    group_by: g
    order_by: z
    aggregate: sum(y) as z
```

Query Execution Engine

Query Execution Engine

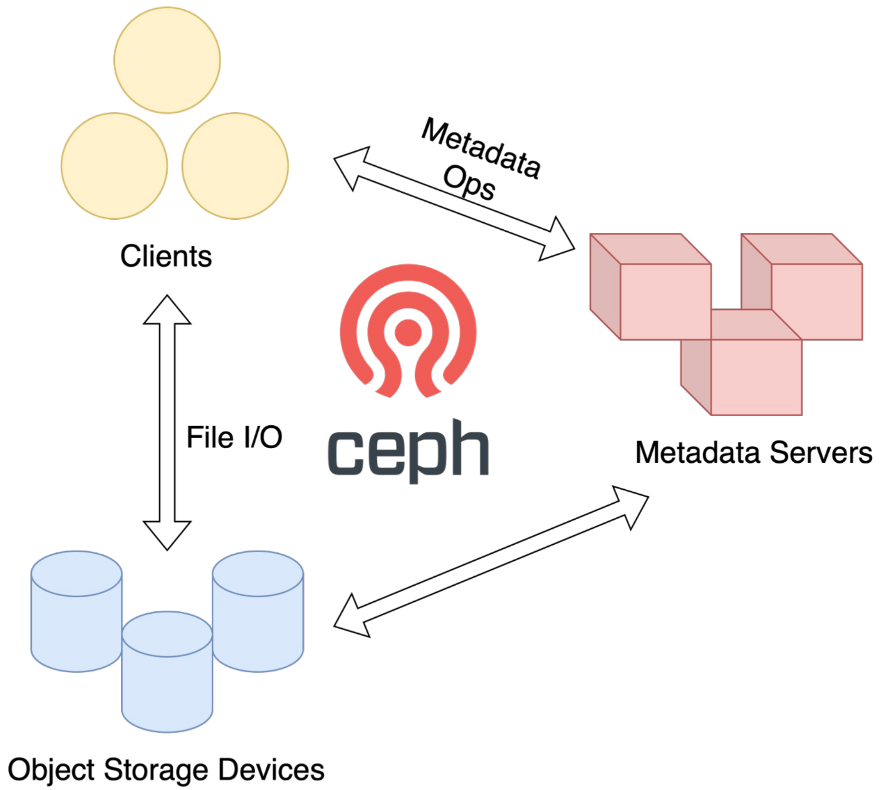
SIMD based vectorized operations

Scalar, complex, and nested types such as maps, structs, lists, tensors

Supports Arrow, Parquet, ORC file formats

Understands standard query plan representations such as Substrait

Distributed Object Storage System



File Format

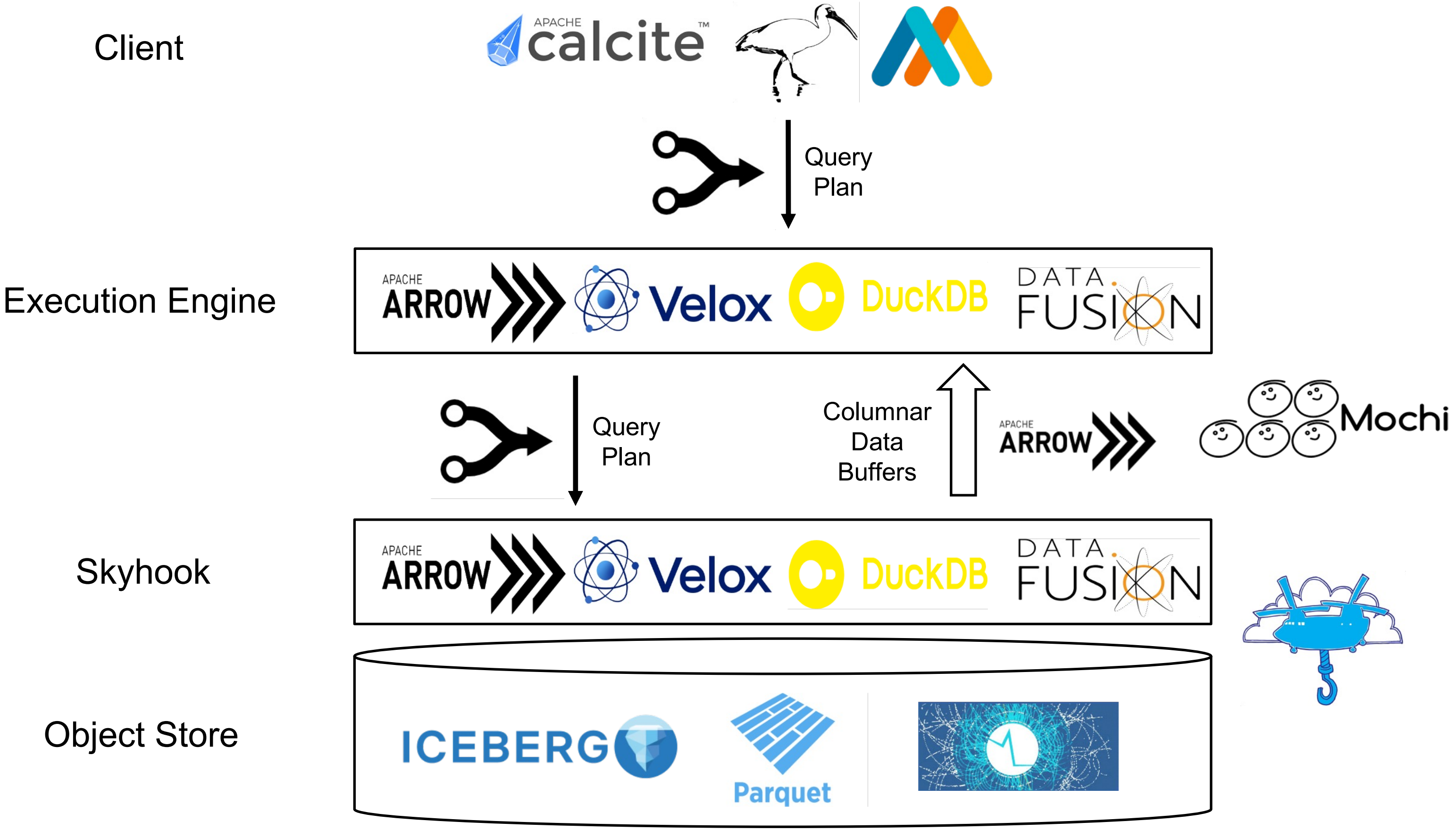
Awkward Array

ROOT

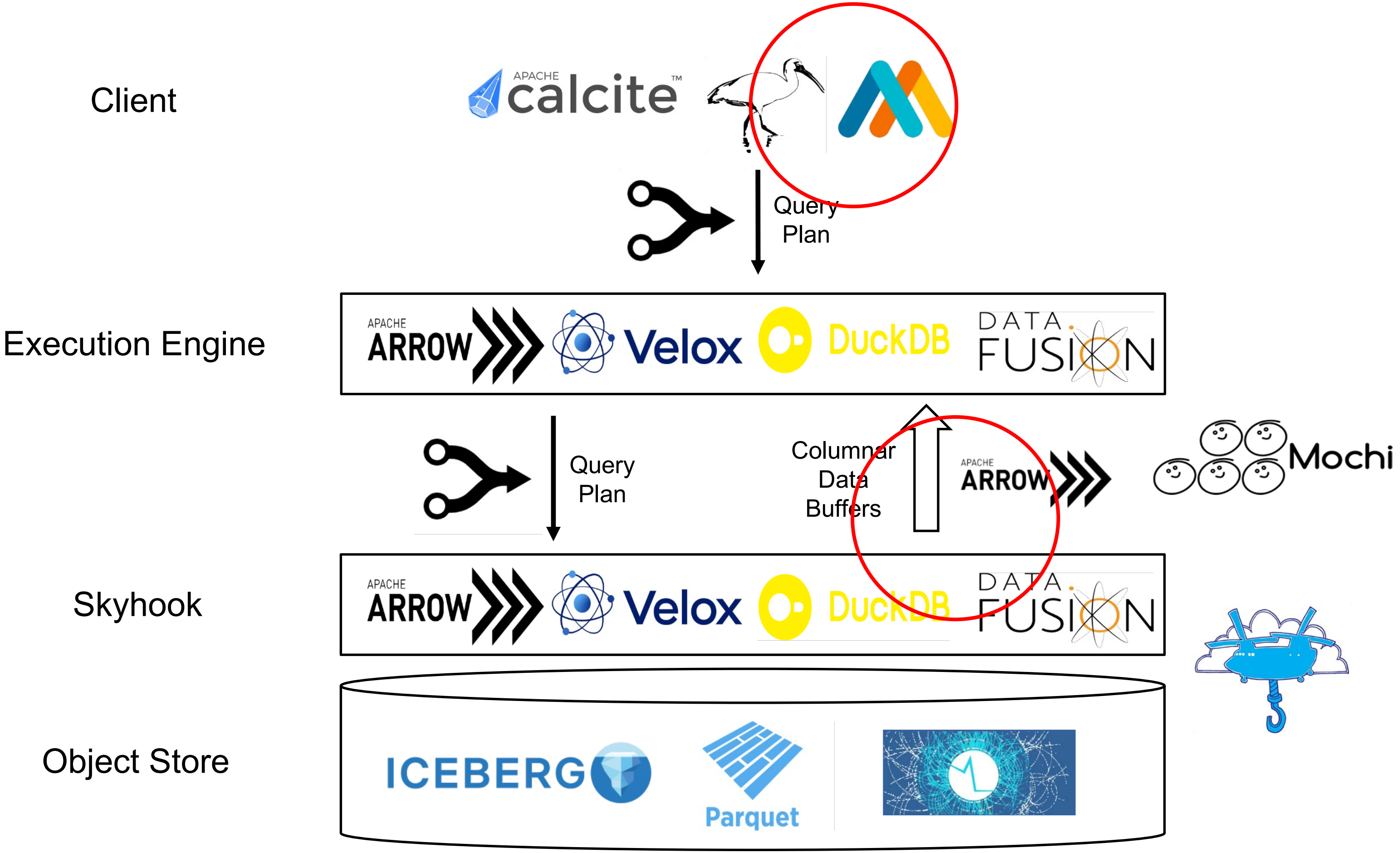
Parquet

ICEBERG

Putting ‘em together,


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

Ongoing Work

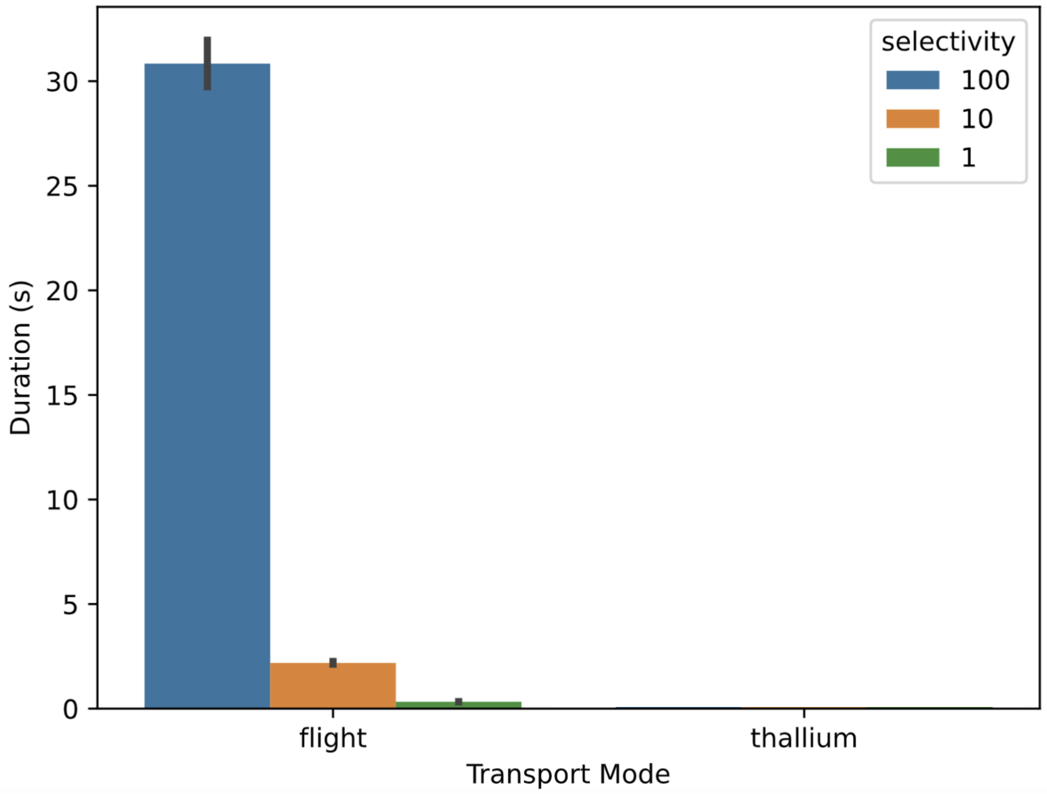

Client
APACHE calcite™
Query Plan
Execution Engine
APACHE ARROW
Velox
DuckDB
DATA FUSION
Query Plan
Columnar Data Buffers
APACHE ARROW
Mochi
Skyhook
APACHE ARROW
Velox
DuckDB
DATA FUSION
Object Store
ICEBERG
Parquet

RDMA for Columnar Data Transport

- Accelerating query execution by offloading to storage shifts the bottleneck to the transport layer
  - Most systems use TCP/IP protocols for data transport e.g. Arrow Flight
  - Moving data via TCP/IP requires data to be copied multiple times between the device, user space, and the kernel space
  - We explore using RDMA for fast zero-copy transfer of columnar data
  - We use the Mochi thallium framework from Argonne National Labs for prototyping our protocol

Mochi

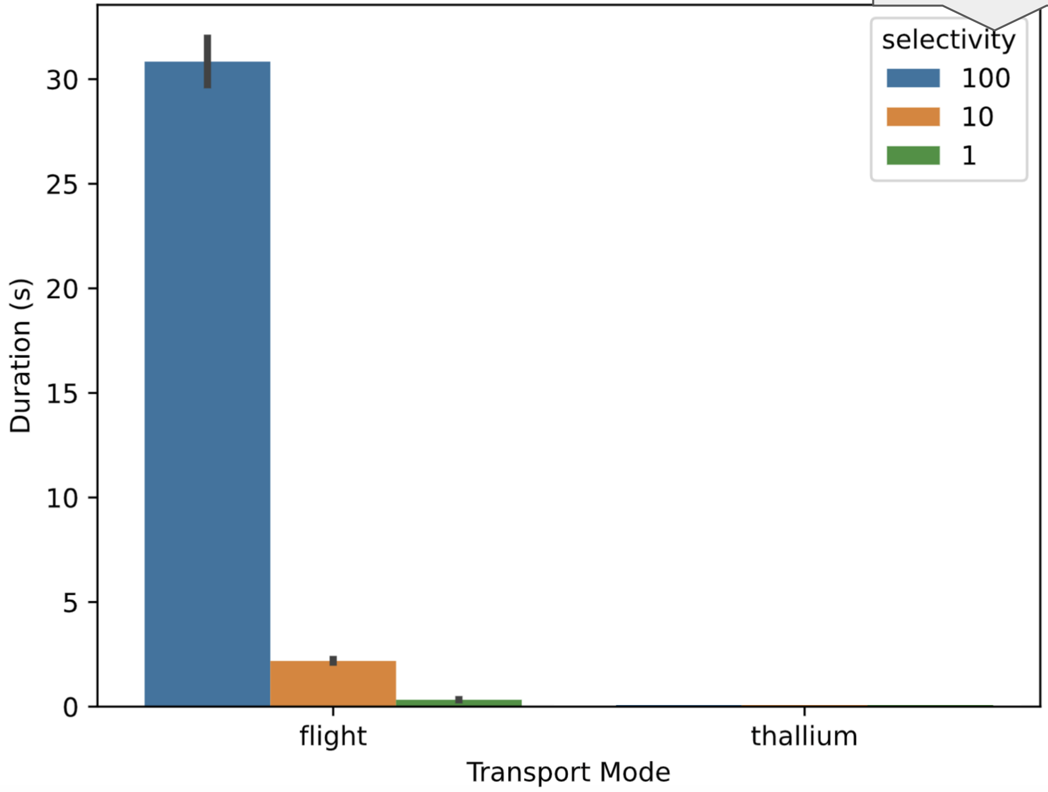
Benchmarks



selectivity
100
10
1
30
25
20
15
10
5
0
Duration (s)
flight
thallium
Transport Mode

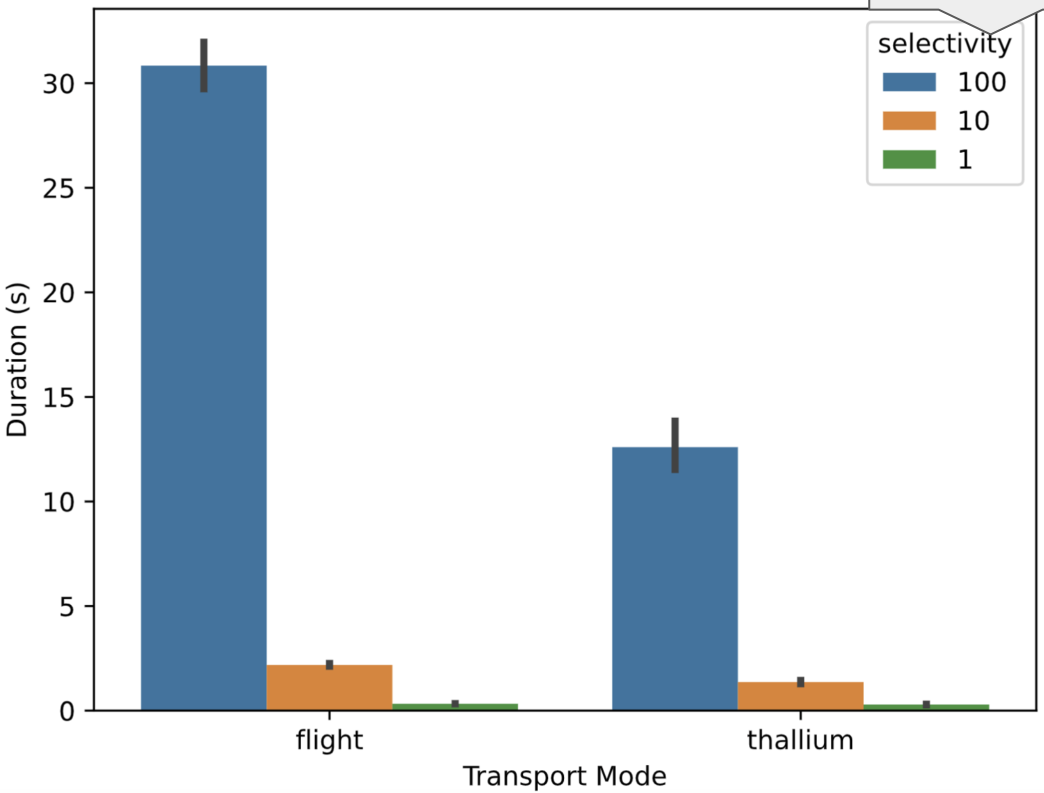
Benchmarks

Selectivity is when our query a select a certain percentage of rows out of every row group



Benchmarks

Selectivity is when our query a select a certain percentage of rows out of every row group



Thank You !

Questions ?