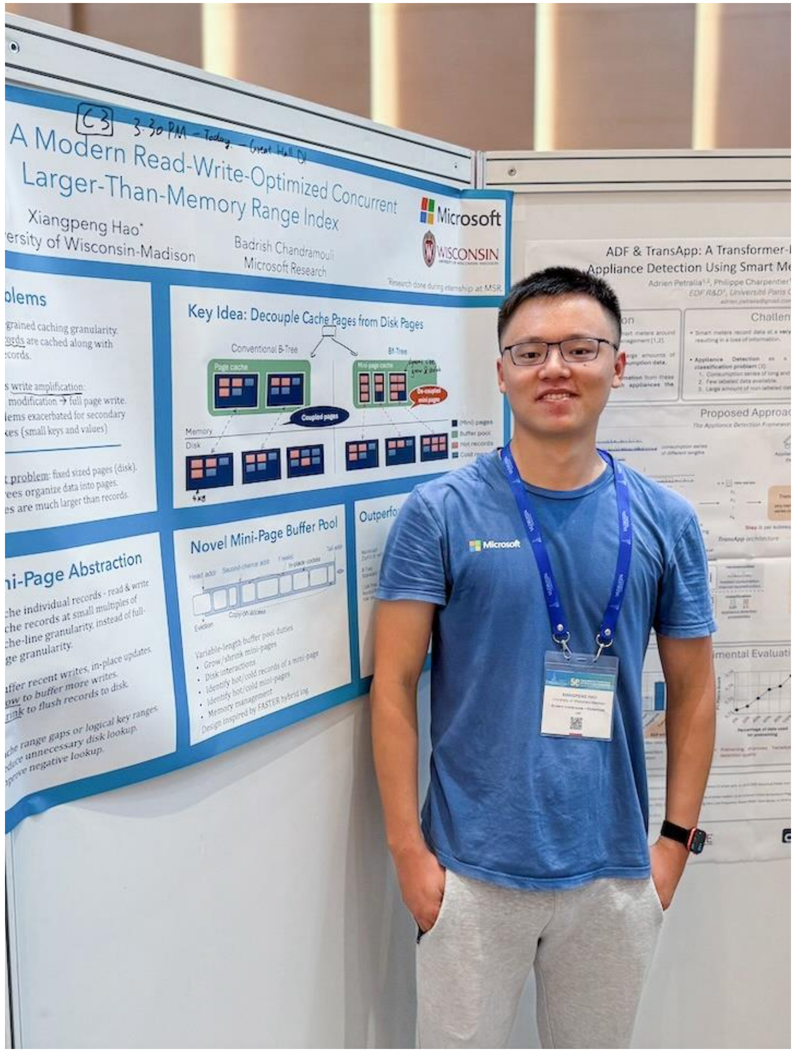
Practical Disaggregated Cache for Apache DataFusion

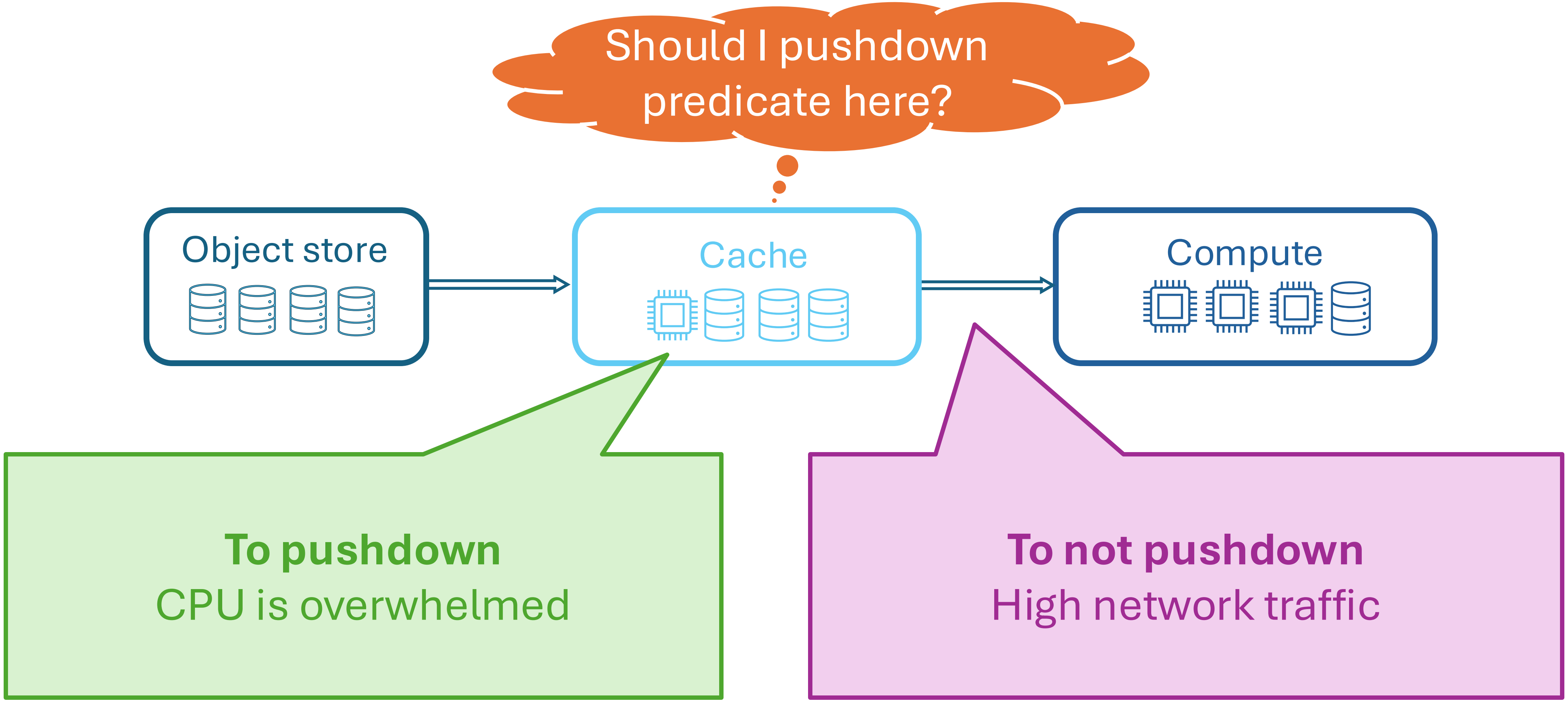
Xiangpeng Hao

About Me

- 4th PhD@Wisconsin-Madison
- Study Database/Storage systems
- Build high performance/low level systems


A Modern Read-Write-Optimized Concurrent Larger-Than-Memory Range Index
Xiangpeng Hao
Badrish Chandramouli
Microsoft Research
Microsoft
WISCONSIN
Key Idea: Decouple Cache Pages from Disk Pages
Novel Mini-Page Buffer Pool
ADF & TransApp: A Transformer-
Appliance Detection Using Smart Me

Disaggregated cache



Should I pushdown predicate here?

Object store

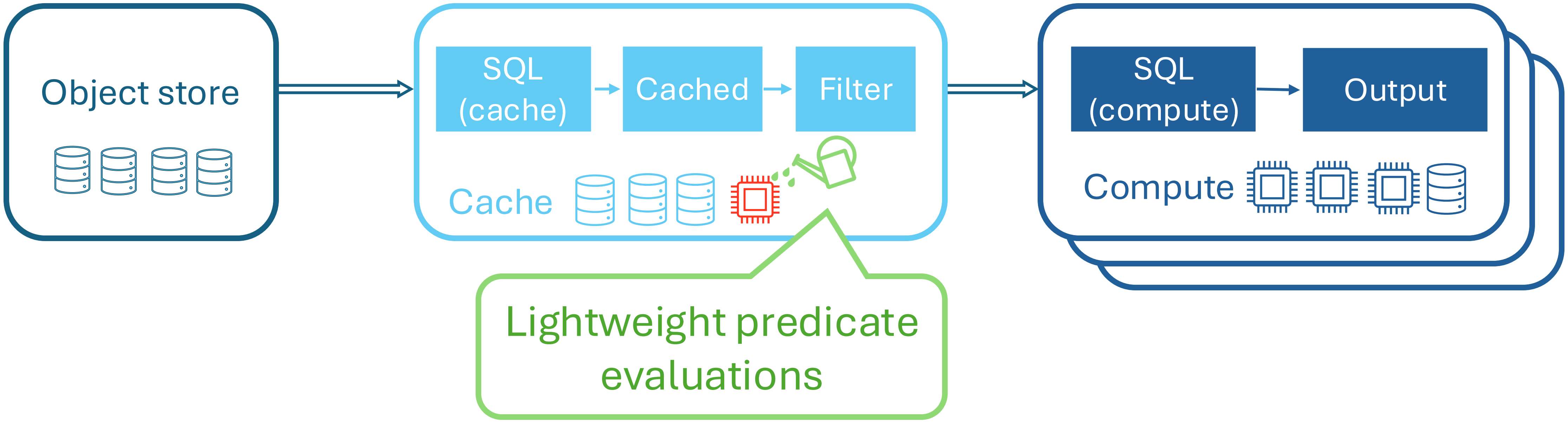
Cache

Compute

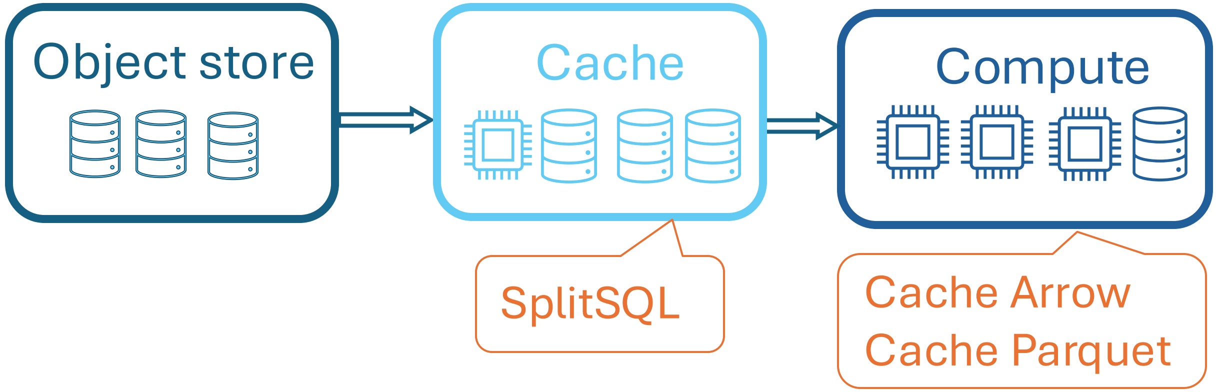
To pushdown
CPU is overwhelmed

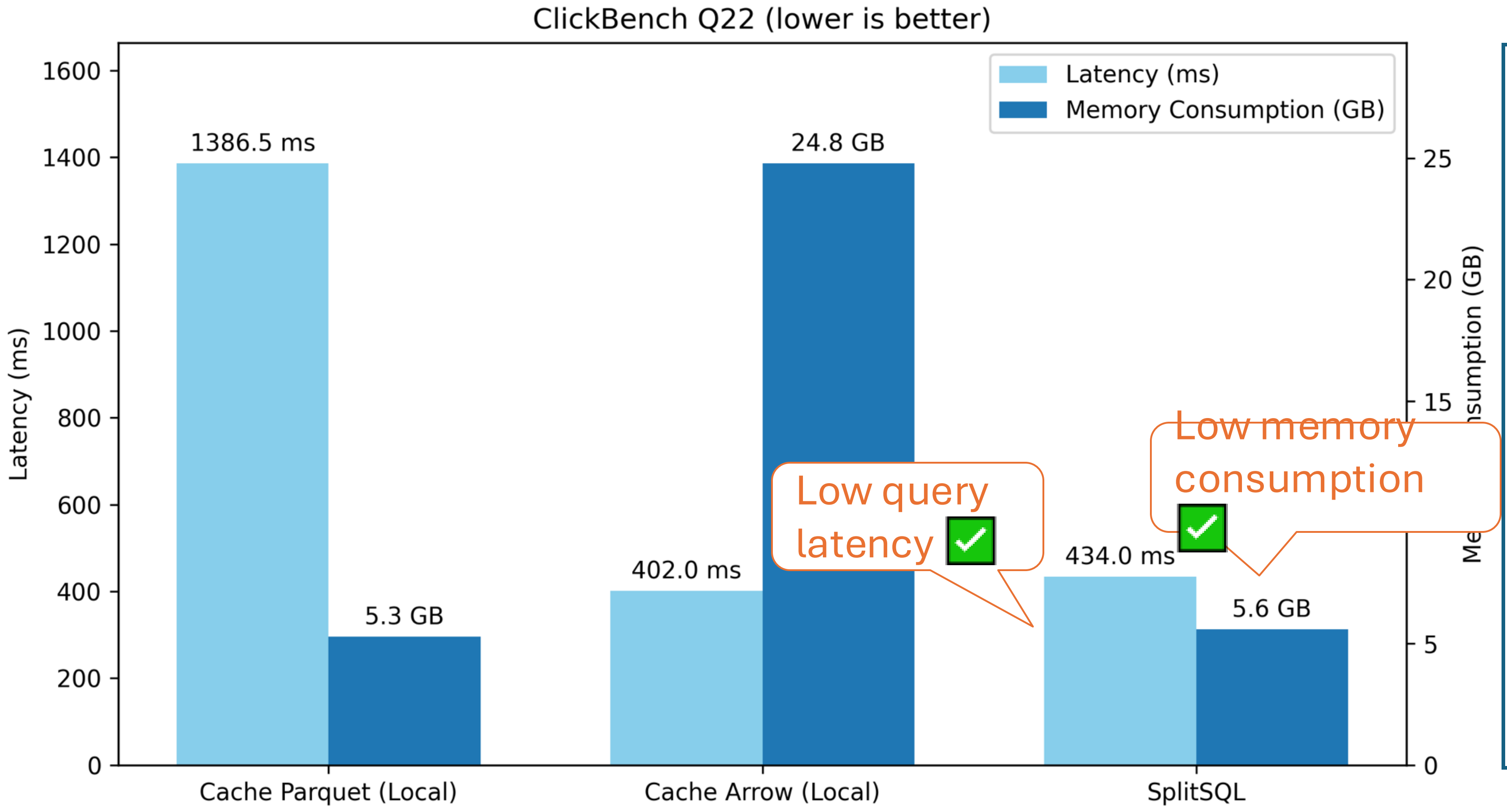
To not pushdown
High network traffic

SplitSQL: Pushdown Done Right



Result





Low query latency ✓

Low memory consumption ✓

```
SELECT
"SearchPhrase",
MIN("URL"),
MIN("Title"),
COUNT(*) AS c, COUNT(DISTINCT
"UserID")
FROM hits
WHERE
"Title" LIKE '%Google%'
AND
"URL" NOT LIKE '%.google.%' AND
"SearchPhrase" <> ''
GROUP BY "SearchPhrase" ORDER BY c
DESC LIMIT 10;
```

Outline

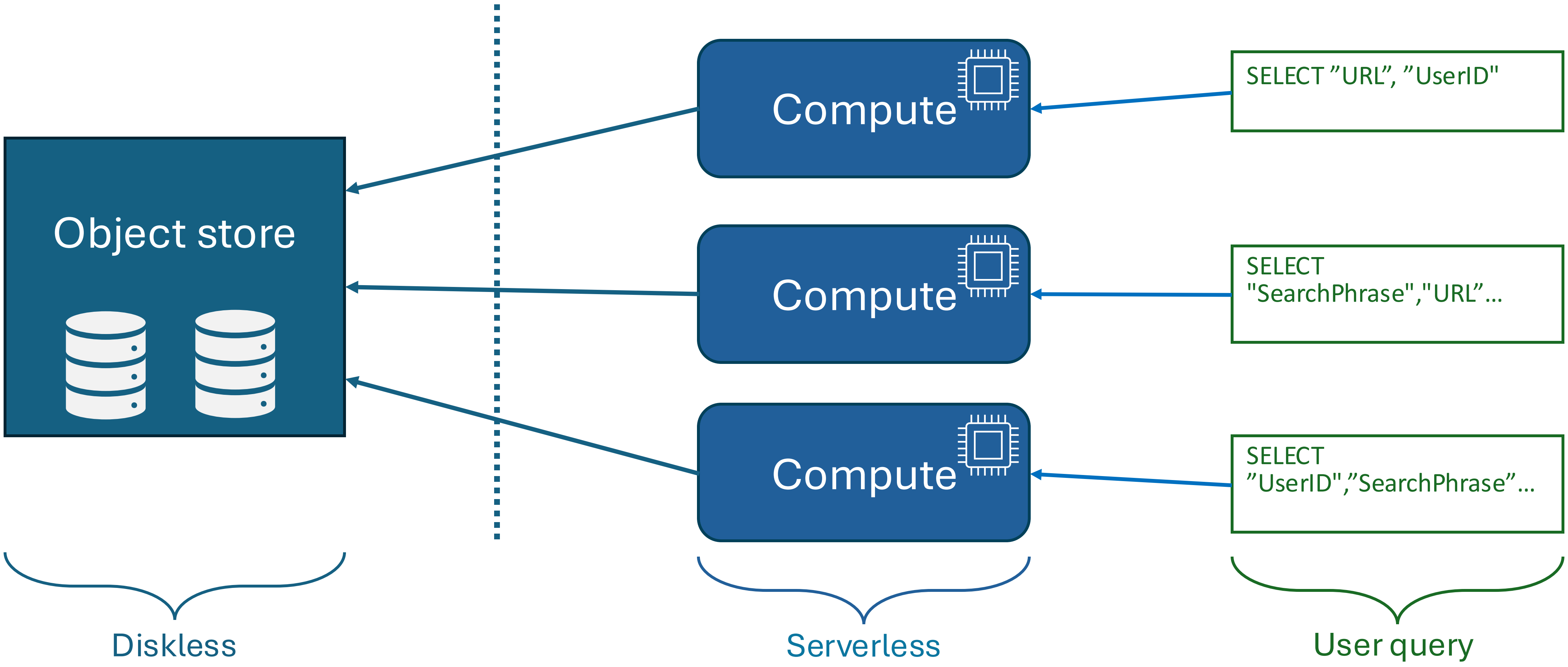
Part 1: Disaggregated cache is the future

Part 2: To Pushdown or not to pushdown?

Part 3: SplitSQL: pushdown down right

Part 4: Evaluations

Data lake architecture



Object store

Compute

Compute

Compute

SELECT ”URL”, ”UserID"

SELECT "SearchPhrase","URL”…

SELECT ”UserID",”SearchPhrase”…

Diskless

Serverless

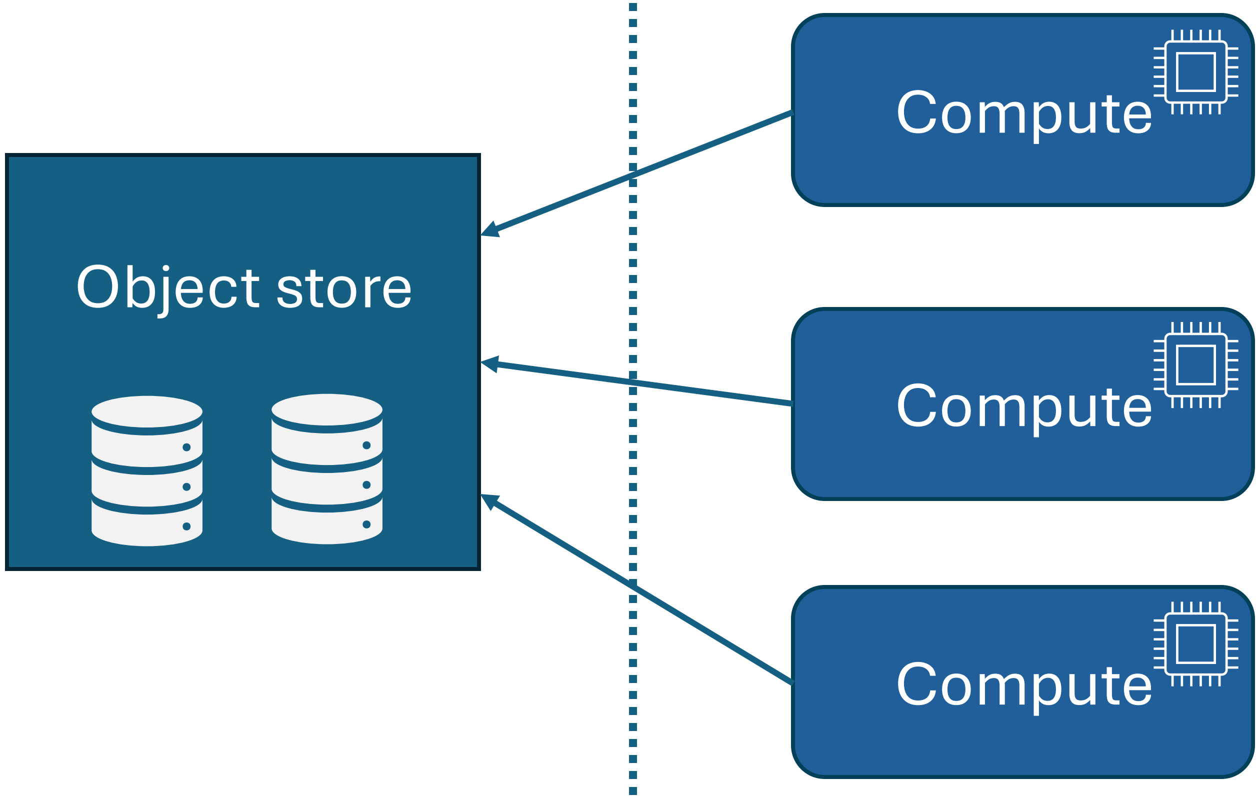
User query

Modern architecture

Object store is new **disk**

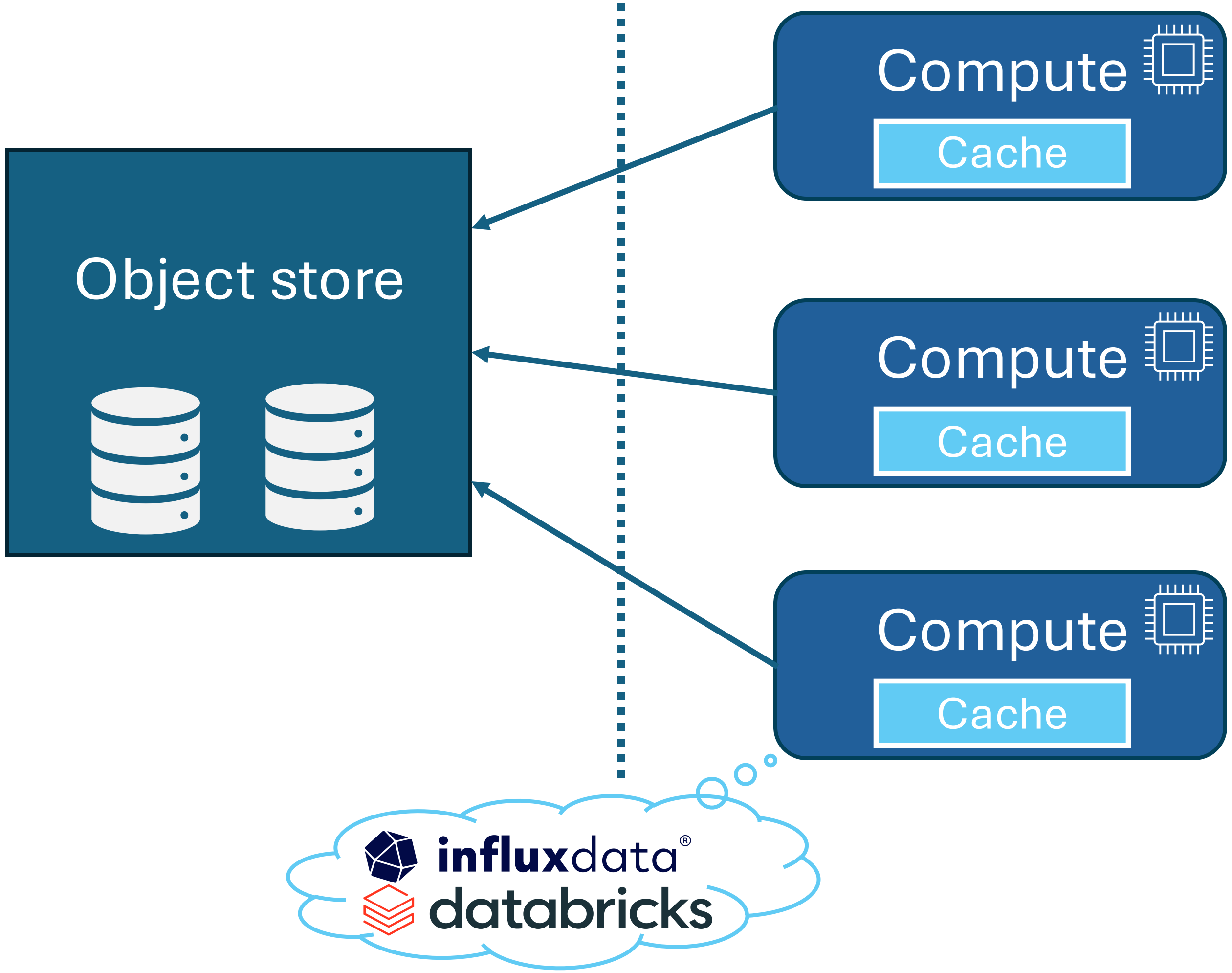
Lambda/EC2 is new **CPU**

Where to **cache** data?



Option 1: private cache

- Simple ✅
- Duplicate copies ❌
- Can’t scale independently ❌
- Low resource utilization ❌



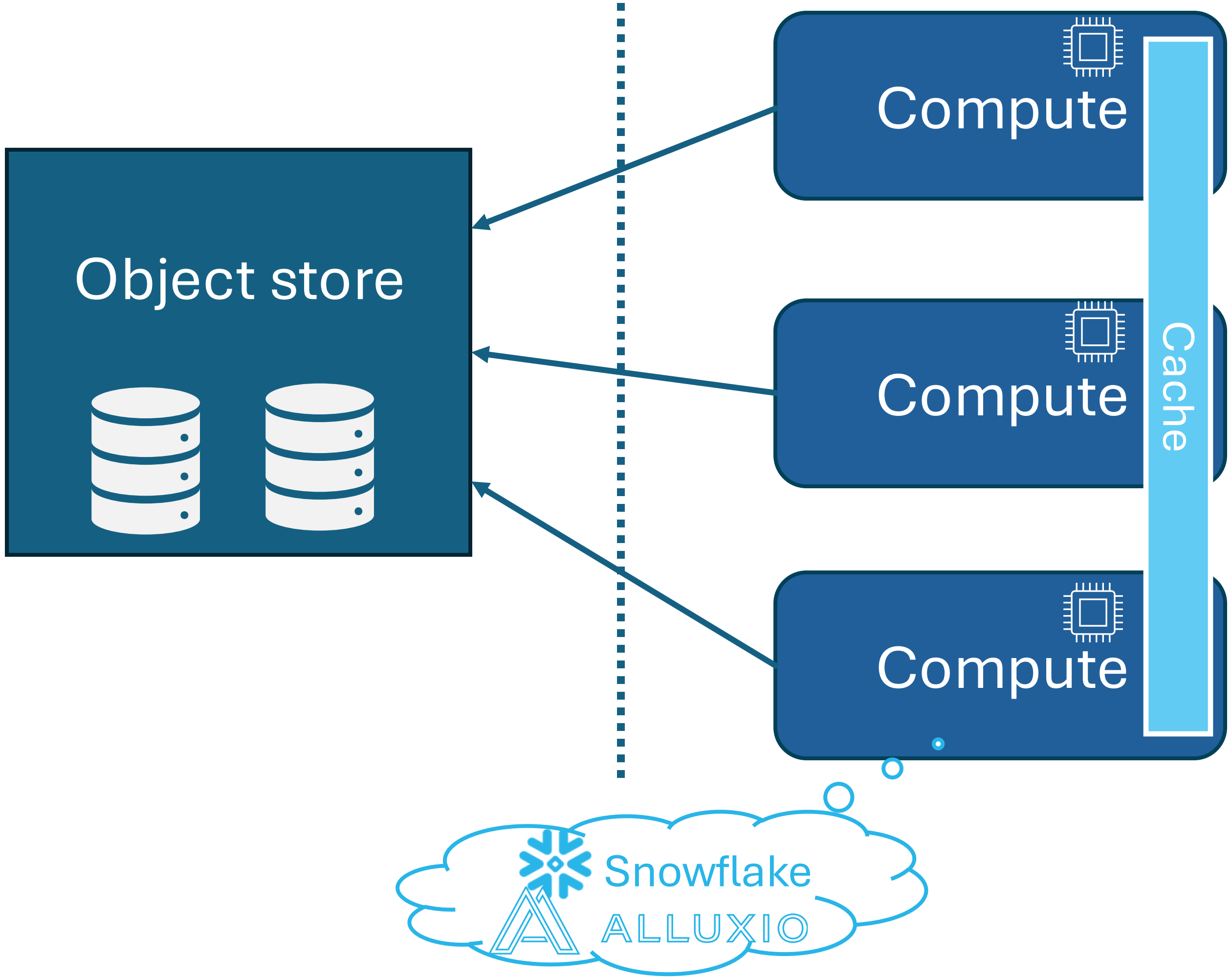
Option 2: distributed cache

Complex design ❌

Single copy ✅

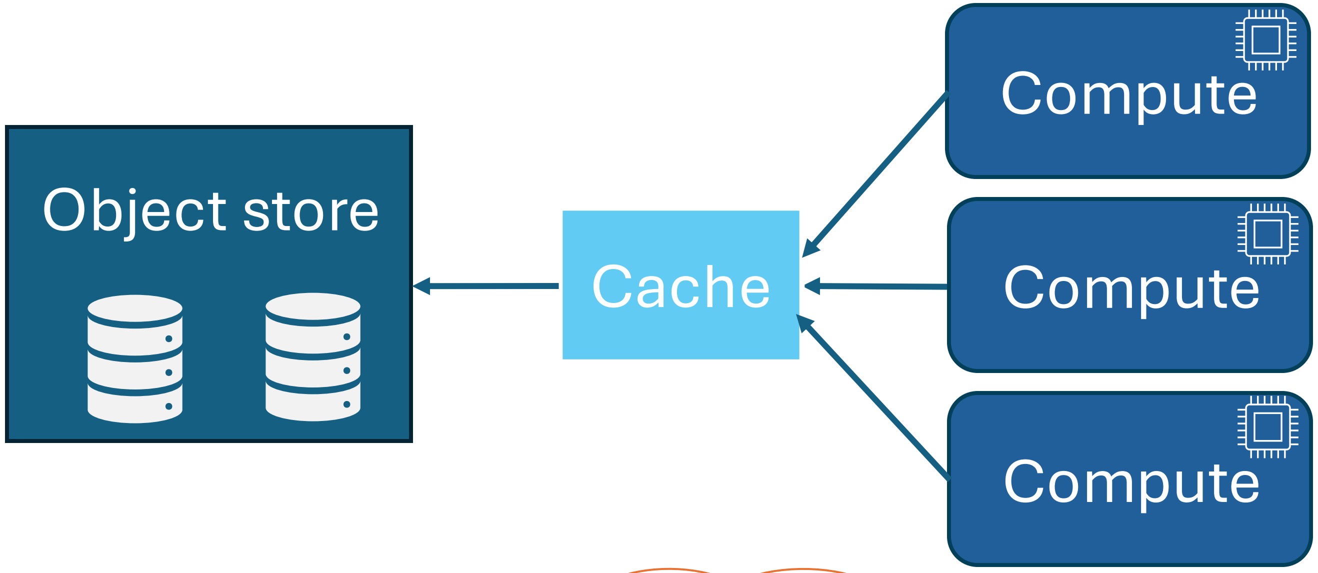
Can’t scale independently ❌

High resource utilization ✅



Option 3: disaggregated cache

- Clean architecture 🎯
- Single copy ✅
- Scale independently ✅
- High resource utilization ✅



Why not popular?

Outline

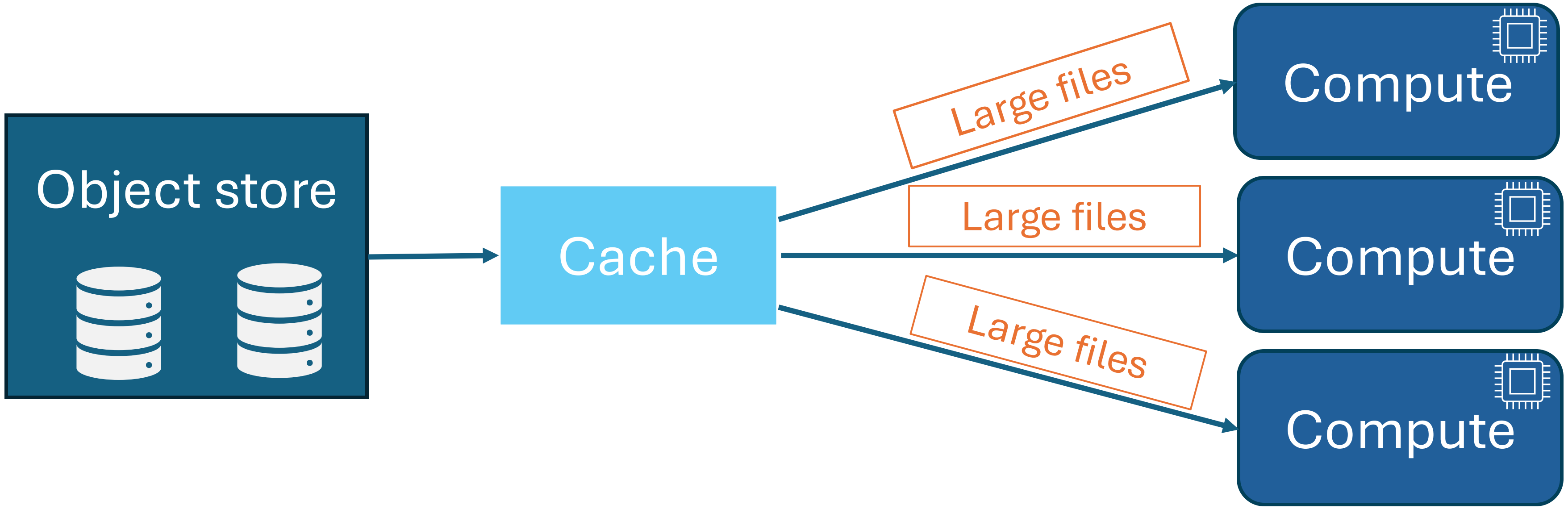
Part 1: Disaggregated cache is the future

Part 2: To Pushdown or not to pushdown?

Part 3: SplitSQL: pushdown down right

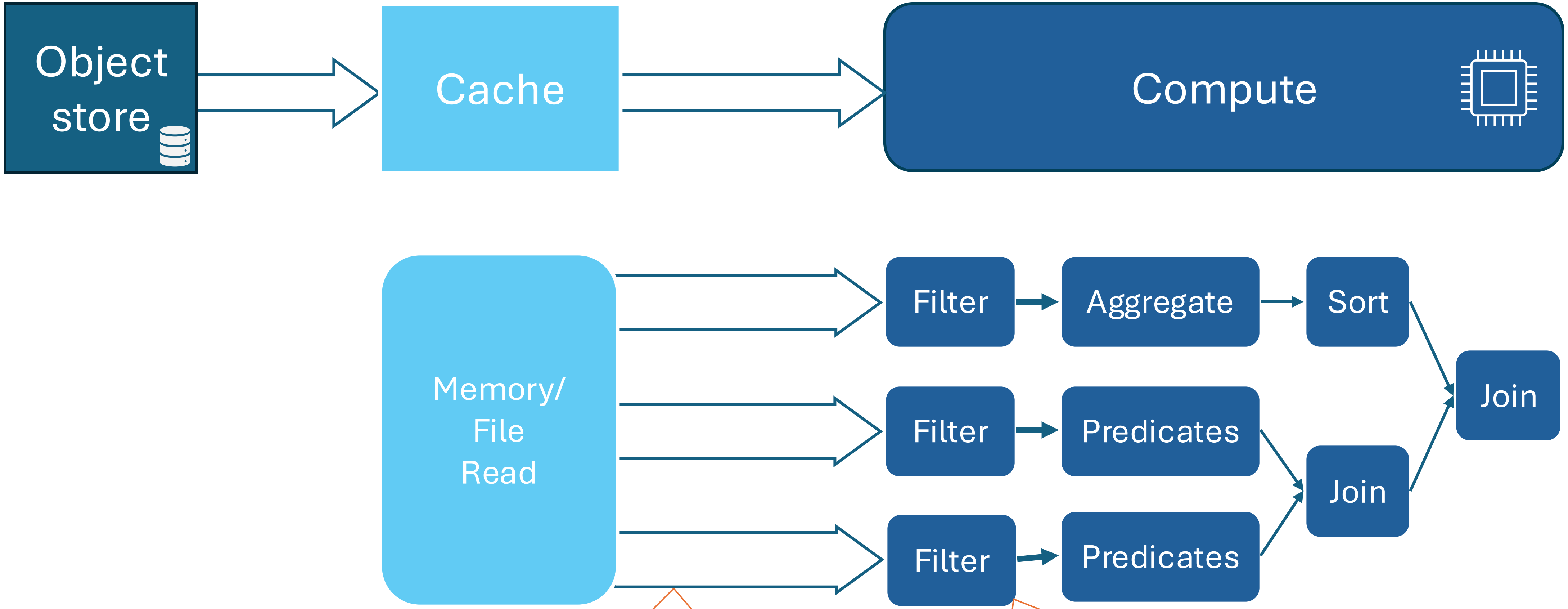
Part 4: Evaluations

Disaggregate cache cause high network traffic



Large files send over network!

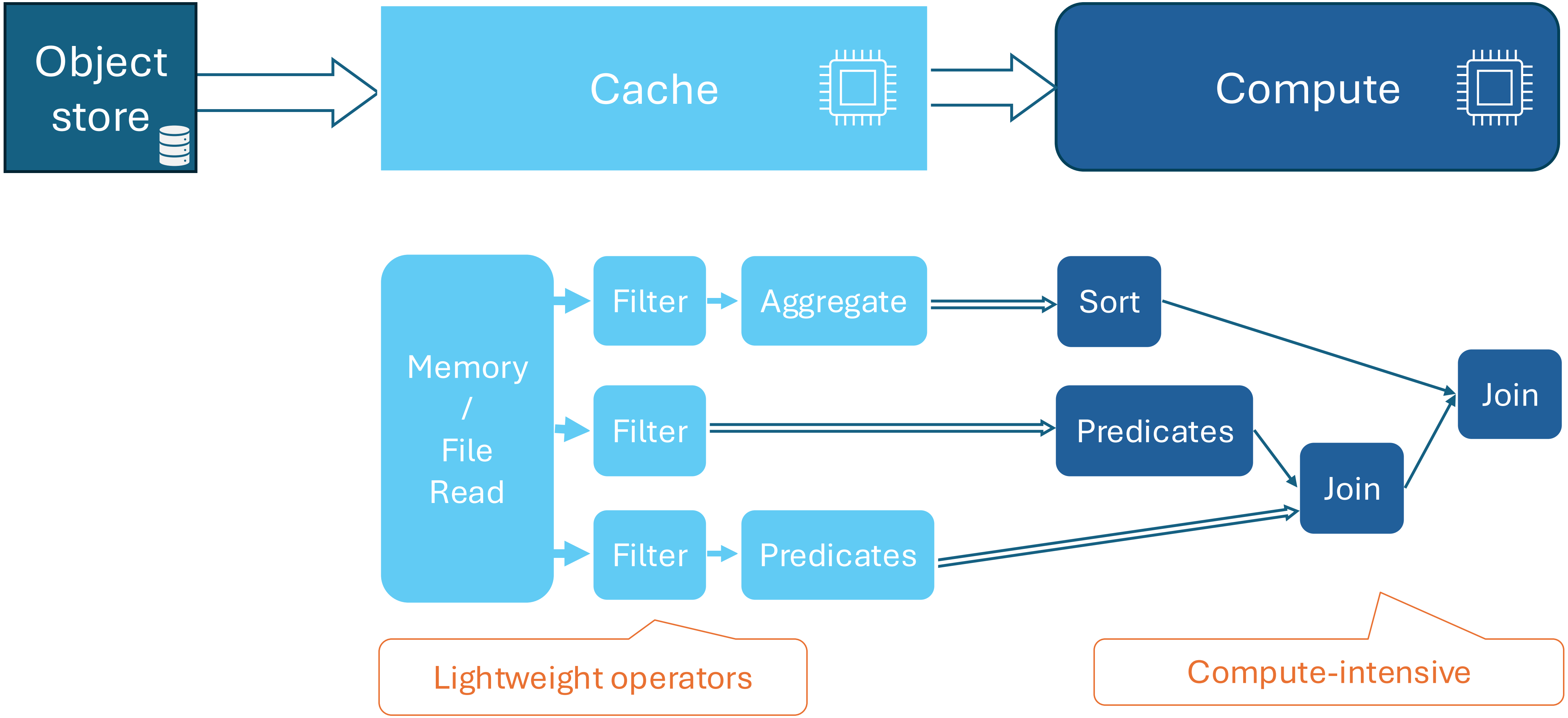
Queries filtered out most data



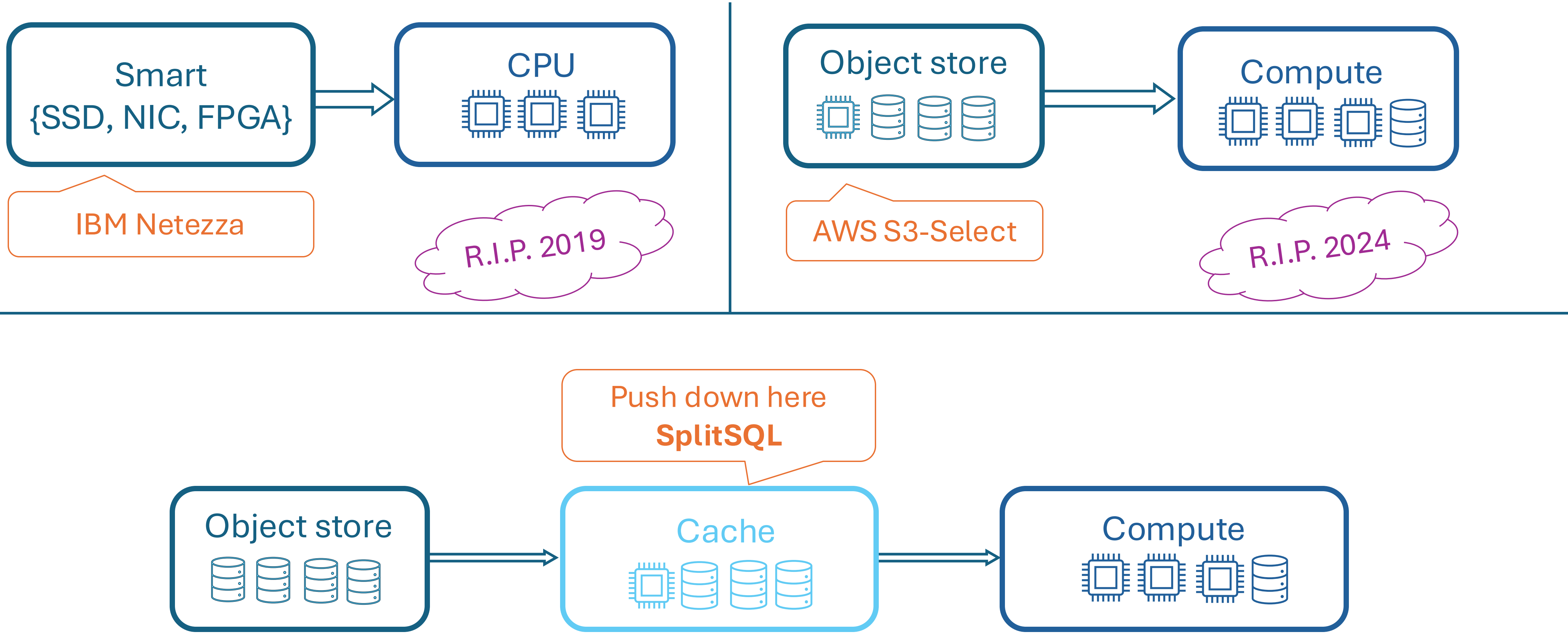
Step 1: large files send over network

Step 2: most data dropped here

Evaluate predicates on cache



This is predicate push down **to cache**



Smart {SSD, NIC, FPGA}

CPU

IBM Netezza

R.I.P. 2019

Object store

Compute

AWS S3-Select

R.I.P. 2024

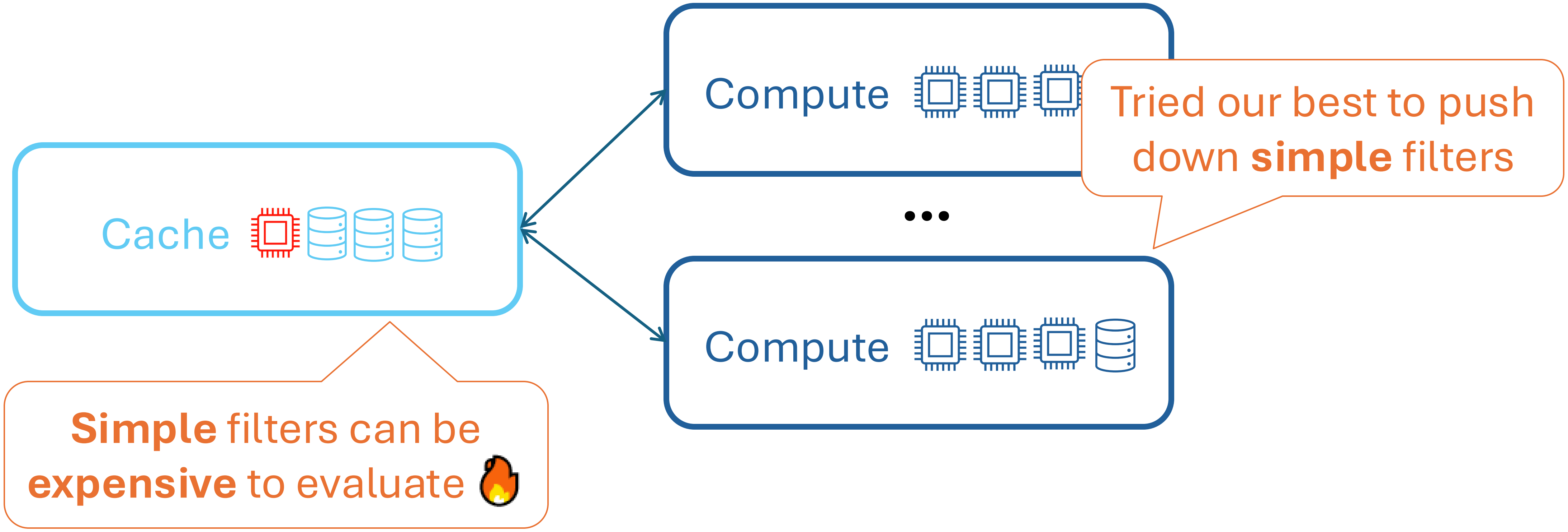
Push down here **SplitSQL**

Object store

Cache

Compute

Challenge: cache server is on fire 🔥!



Tried our best to push down **simple** filters

Simple filters can be **expensive** to evaluate 🔥

Outline

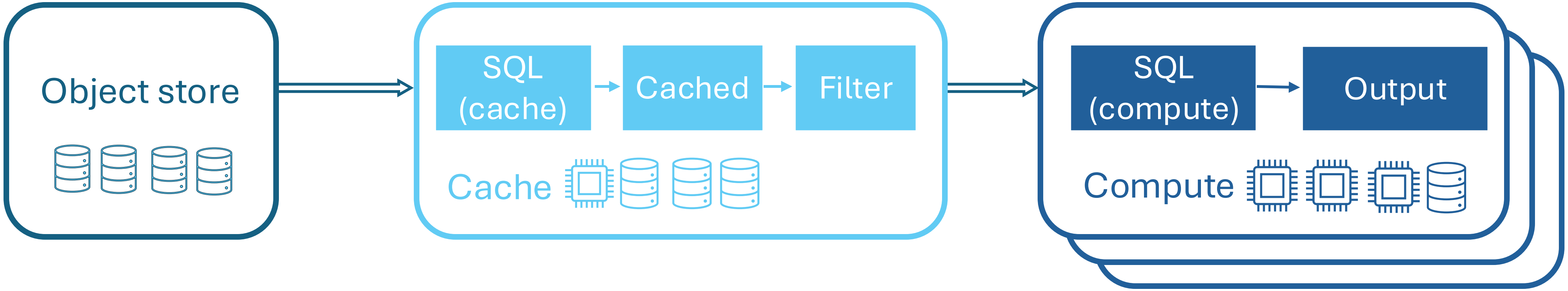
Part 1: Disaggregated cache is the future

Part 2: To Pushdown or not to pushdown?

Part 3: SplitSQL: pushdown down right

Part 4: Evaluations

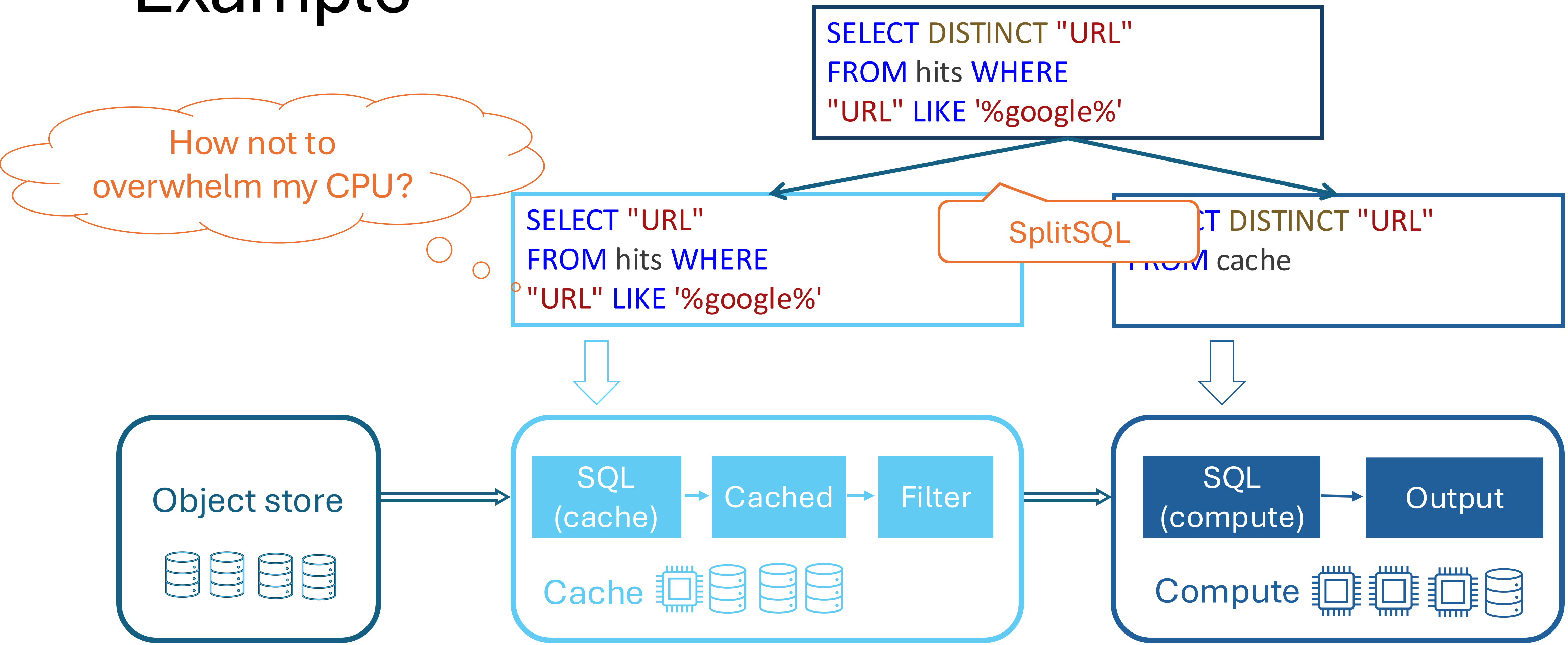
SplitSQL: Pushdown Done Right



Scale cache:
Provision larger memory,
attach larger elastic storage

Scale compute:
More compute nodes,
same cache node

Example



How not to overwhelm my CPU?

```
SELECT DISTINCT "URL"
FROM hits WHERE
"URL" LIKE '%google%'
```

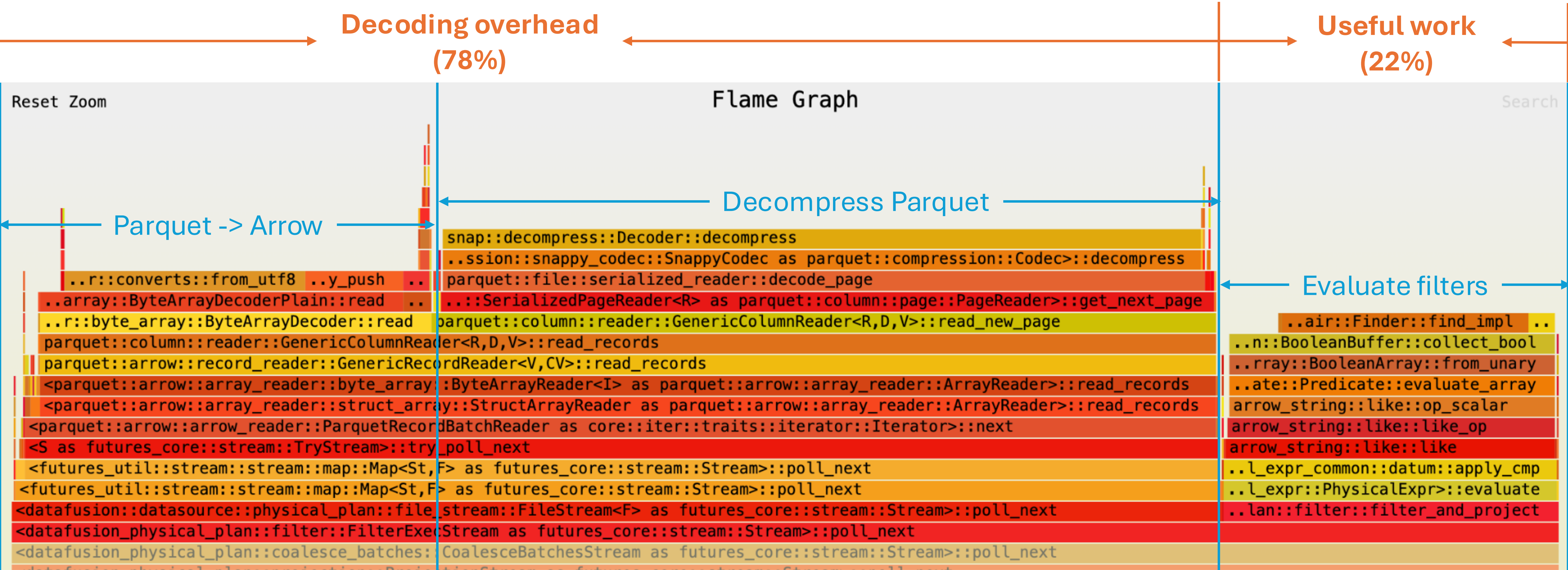
```
SELECT "URL"
FROM hits WHERE
"URL" LIKE '%google%'
```

SplitSQL

```
CT DISTINCT "URL"
FROM cache
```

```sql
SELECT "URL”
FROM hits
WHERE "URL" LIKE '%google%'
```

Simple filters can be expensive to evaluate



Decoding overhead (78%)

Useful work (22%)

Parquet -> Arrow

Decompress Parquet

Evaluate filters

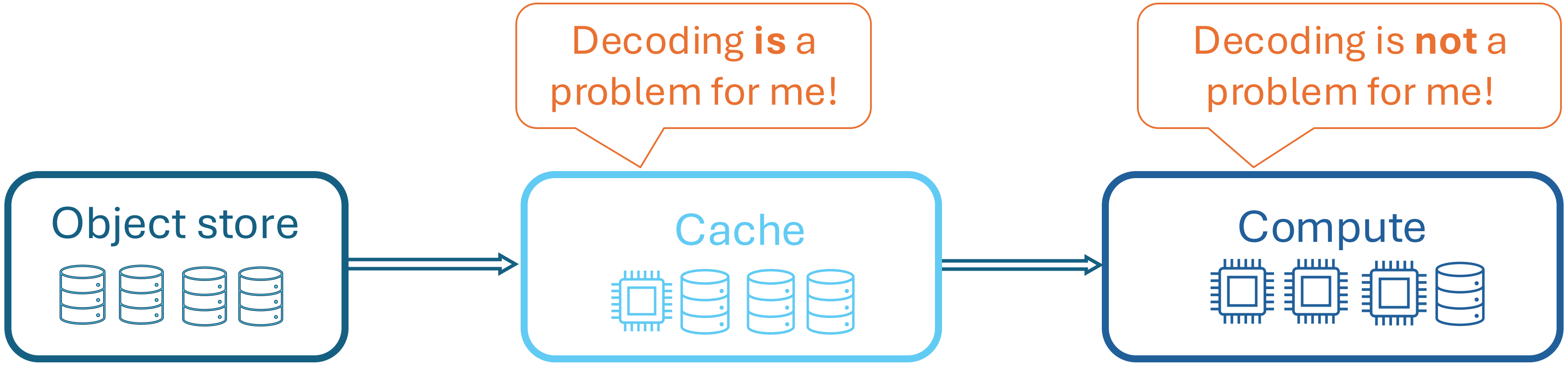
Predicate push down doesn’t like Parquet

Parquet is the industry standard

- Rich features, great ecosystem
- Battle tested
- High compression ratio
- De facto file format for big data

Decoding Parquet is CPU-intensive

- Decompression
- Decoding metadata
- Decoding data

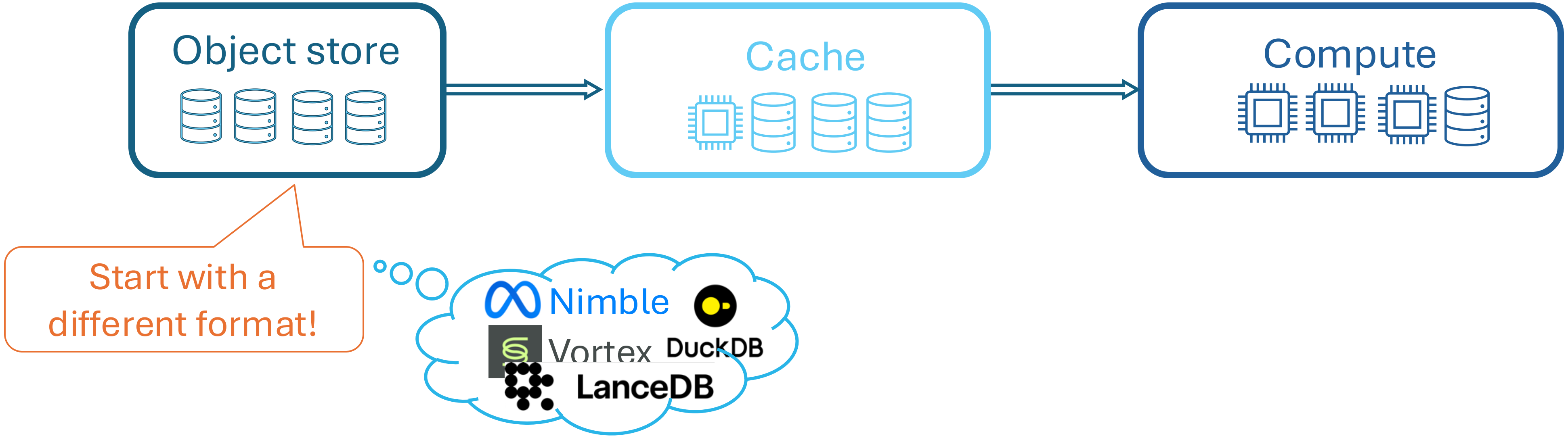

Decoding **is** a problem for me!
Decoding is **not** a problem for me!
Object store
Cache
Compute

Faster decoding is All-You-Need

Option 1: switch to a different file format

- Small win, big lose – lose all other nice Parquet features
- Significant changes to the ecosystem
- Slow adoption (e.g., >10 years)

☒



Object store

Cache

Compute

Start with a different format!

Nimble

Vortex

DuckDB

LanceDB

Faster decoding is All-You-Need

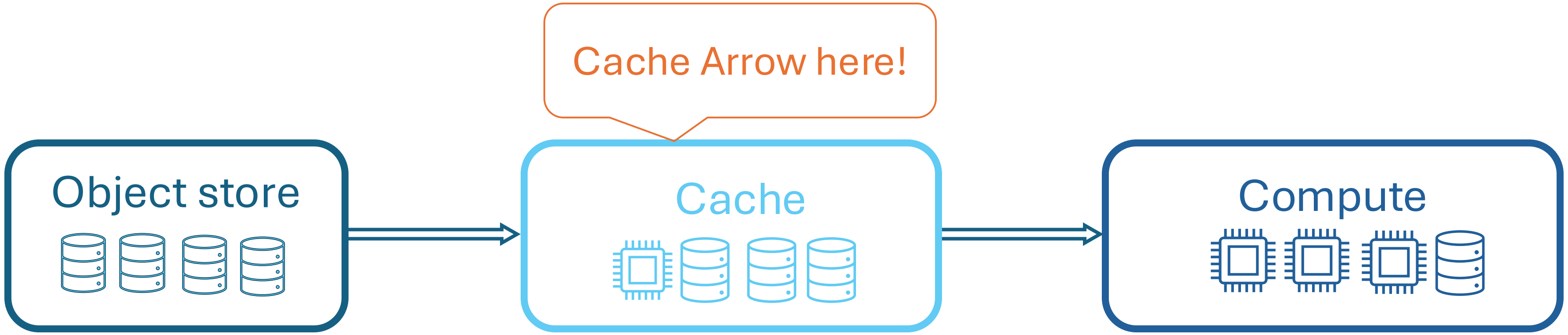
Option 1: switch to a customized file format

- Small win, big lose – lose all other nice Parquet features
- Significant changes to the ecosystem
- Slow adoption (e.g., >10 years)

☒

Option 2: cache decoded values (e.g., cache Arrow)

?



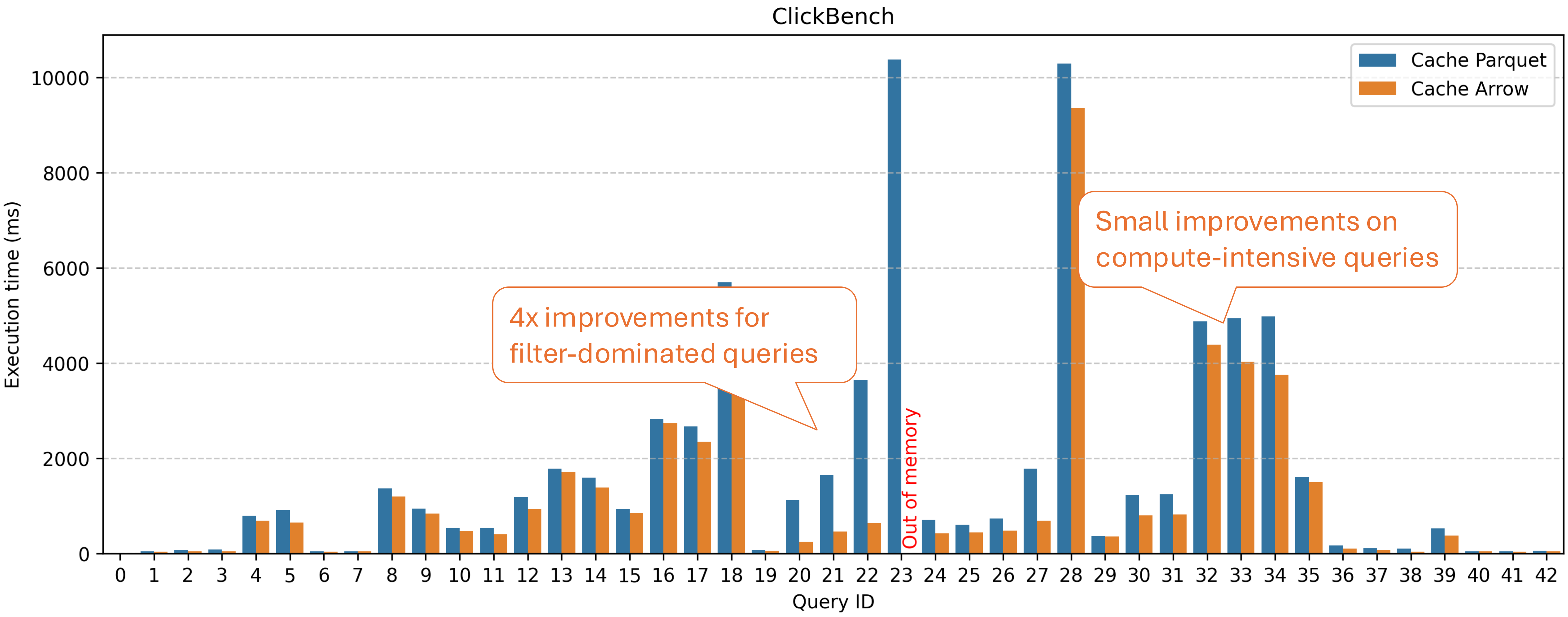
Cache Arrow here!

Object store

Cache

Compute

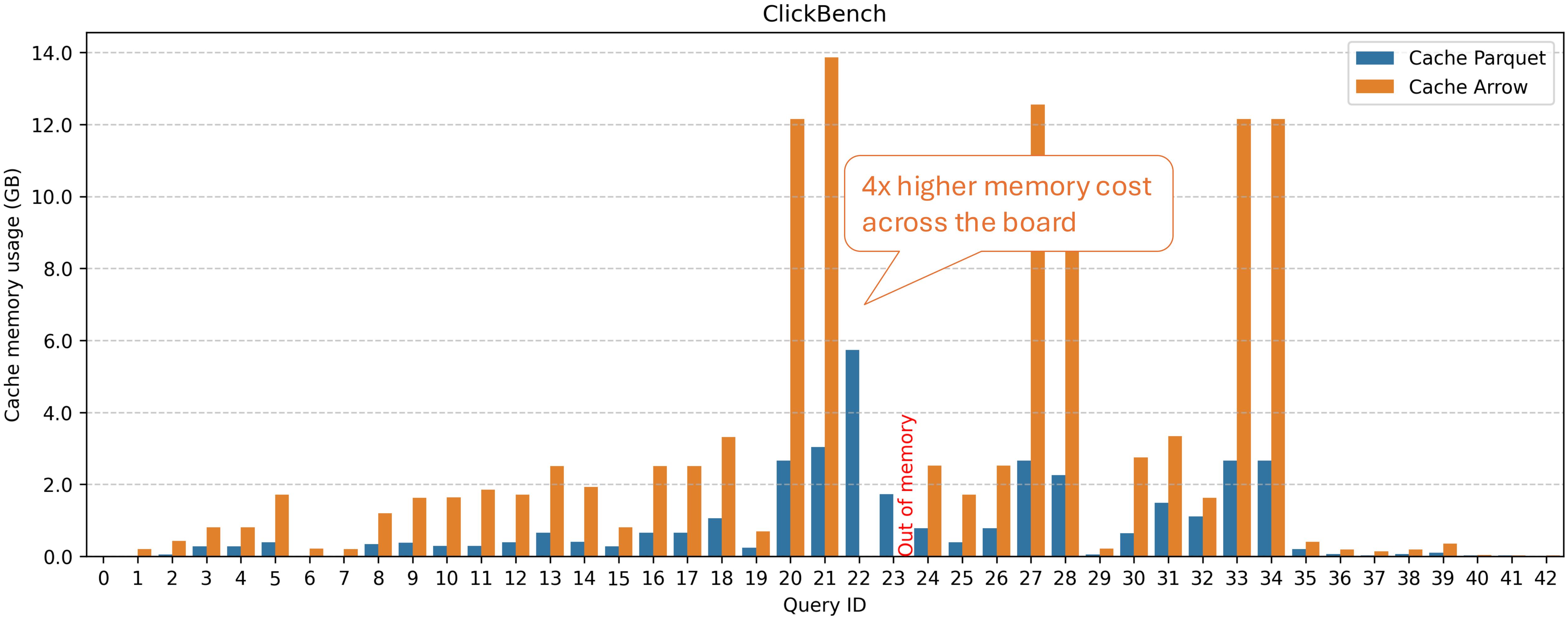
Cache Arrow speeds up some queries



4x improvements for filter-dominated queries

Small improvements on compute-intensive queries

But at 4x memory cost



4x higher memory cost across the board

Faster decoding is All-You-Need

Option 1: switch to a customized file format ☒

- Small win, big lose – lose all other nice Parquet features
- Significant changes to the ecosystem
- Slow adoption (e.g., >10 years)

Option 2: cache decoded values (e.g., cache Arrow) ☒

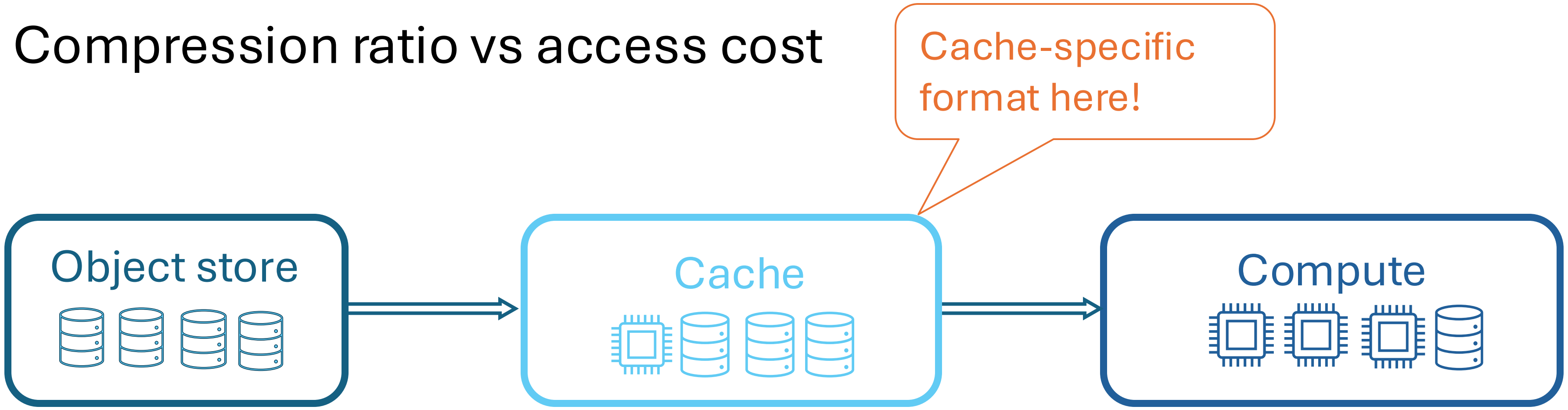
- Performance improvement varies
- 4x more memory usage

We need: cache-specific file format ☑

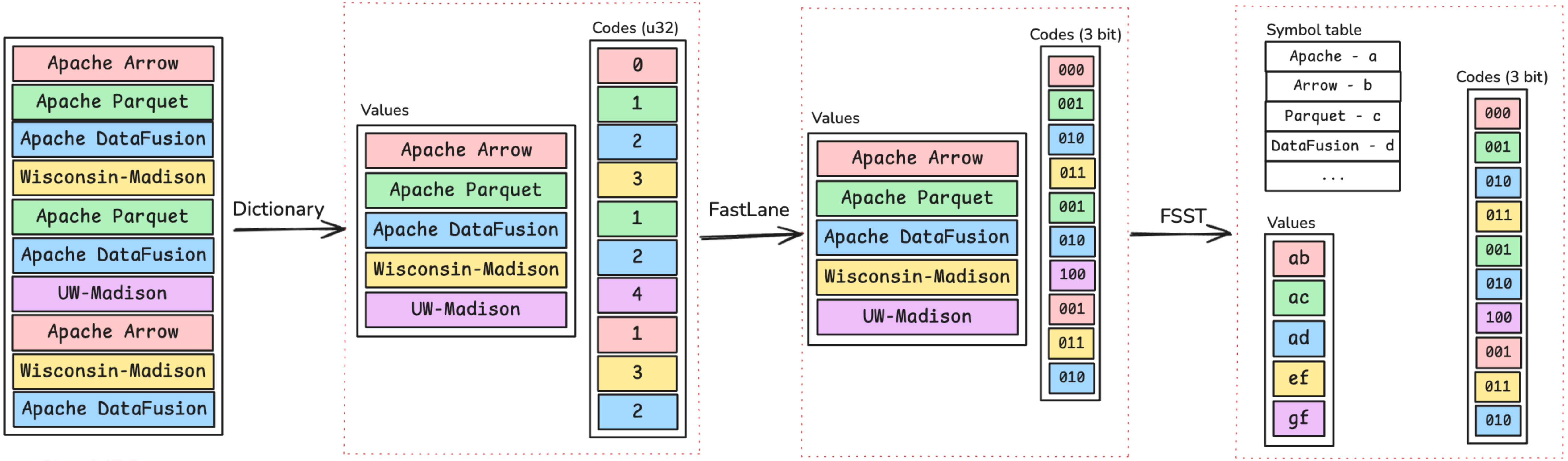
- Transparent – what happens in cache, stays in cache
- Unlocks new opportunities

Cache-specific format?

- Leverage modern encoding algorithms
  - SIMD friendly
  - Fine-grained decoding – decode only relevant data
  - Evaluate predicates on encoded data
- Make modern trade-offs
  - IO time vs decode time
  - Compression ratio vs access cost



Example: Encode String arrays



Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
Apache Parquet
Apache DataFusion
UW-Madison
Apache Arrow
Wisconsin-Madison
Apache DataFusion

Size: 147 Bytes

Dictionary

Values
Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
UW-Madison

Codes (u32)
0
1
2
3
1
2
4
1
3
2

Size: 110 Bytes

FastLane

Values
Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
UW-Madison

Codes (3 bit)
000
001
010
011
001
010
100
001
011
010

Size: 74 Bytes

FSST

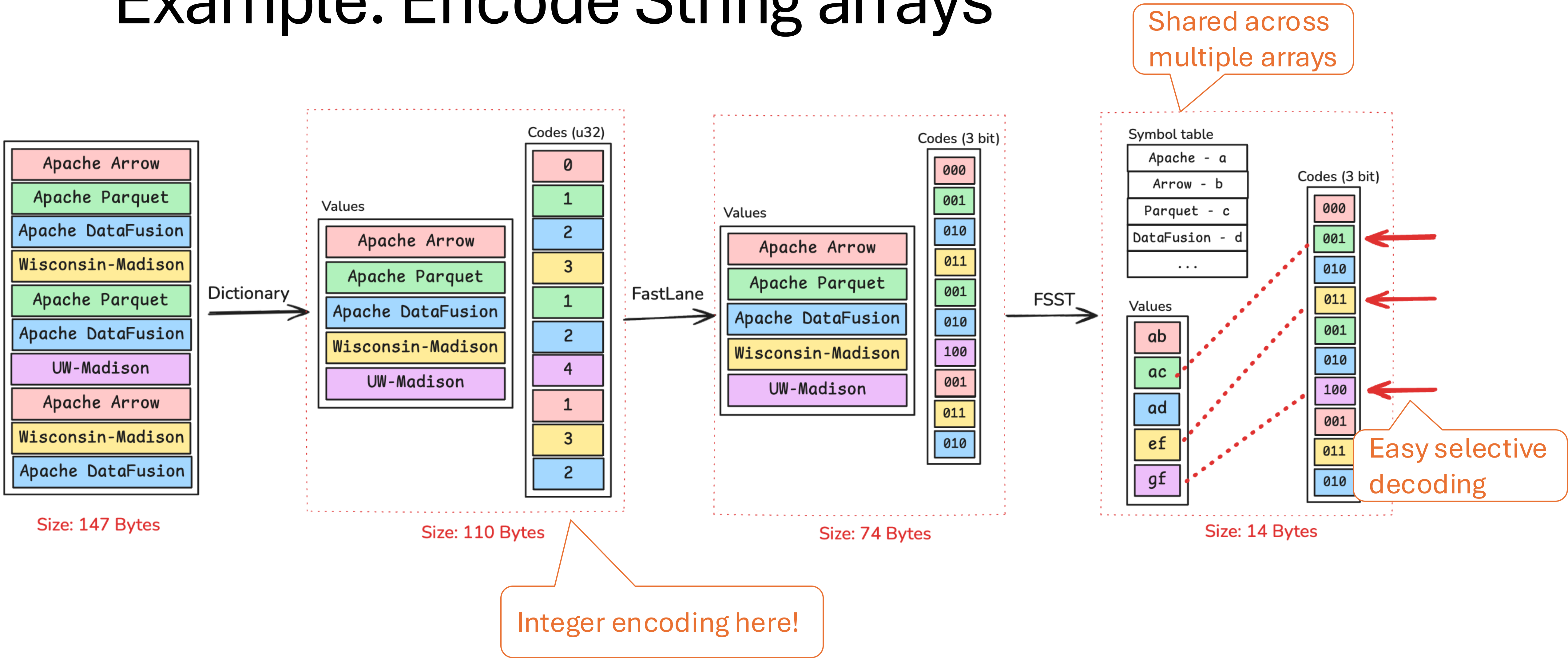
Symbol table
Apache - a
Arrow - b
Parquet - c
DataFusion - d
...

Values
ab
ac
ad
ef
gf

Codes (3 bit)
000
001
010
011
001
010
100
001
011
010

Size: 14 Bytes

Example: Encode String arrays



Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
Apache Parquet
Apache DataFusion
UW-Madison
Apache Arrow
Wisconsin-Madison
Apache DataFusion

Size: 147 Bytes

Dictionary

Values

Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
UW-Madison

Codes (u32)

0
1
2
3
1
2
4
1
3
2

Size: 110 Bytes

FastLane

Values

Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
UW-Madison

Codes (3 bit)

000
001
010
011
001
010
100
001
011
010

Size: 74 Bytes

FSST

Symbol table

Apache - a
Arrow - b
Parquet - c
DataFusion - d
...

Values

ab
ac
ad
ef
gf

Codes (3 bit)

000
001
010
011
001
010
100
001
011
010

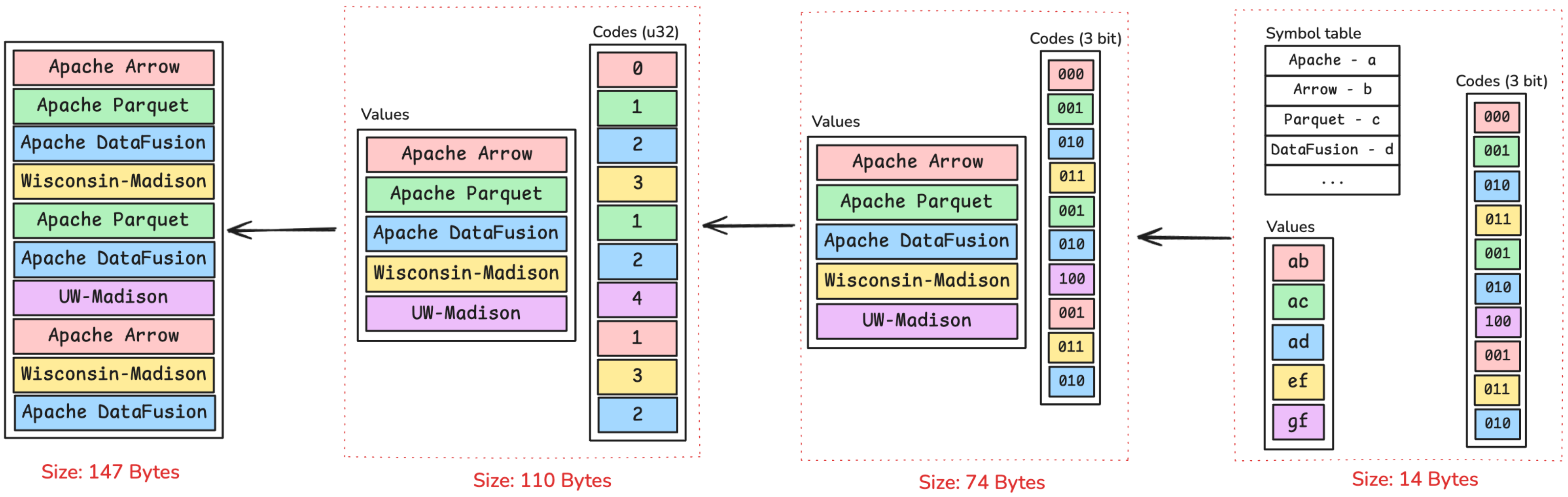
Size: 14 Bytes

Shared across multiple arrays

Easy selective decoding

Integer encoding here!

Evaluate predicates on encoded data



Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
Apache Parquet
Apache DataFusion
UW-Madison
Apache Arrow
Wisconsin-Madison
Apache DataFusion

Size: 147 Bytes

Values
Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
UW-Madison

Codes (u32)
0
1
2
3
1
2
4
1
3
2

Size: 110 Bytes

Values
Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
UW-Madison

Codes (3 bit)
000
001
010
011
001
010
100
001
011
010

Size: 74 Bytes

Symbol table
Apache - a
Arrow - b
Parquet - c
DataFusion - d
...

Values
ab
ac
ad
ef
gf

Codes (3 bit)
000
001
010
011
001
010
100
001
011
010

Size: 14 Bytes

How find strings that contains "DataFusion"?

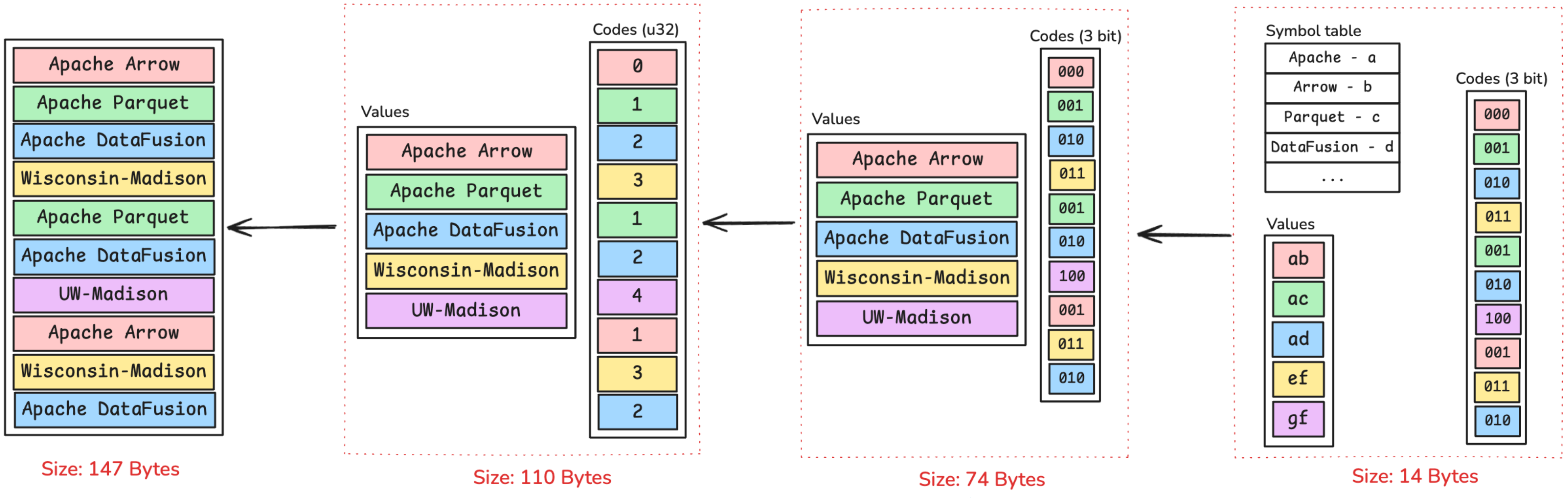
Find "Apache DataFusion"

1. Encode the needle – "ad"

2. Find the index in values – 2

2. Return the index in codes

Evaluate predicates on **partially** encoded data



Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
Apache Parquet
Apache DataFusion
UW-Madison
Apache Arrow
Wisconsin-Madison
Apache DataFusion

Size: 147 Bytes

Values
Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
UW-Madison

Codes (u32)
0
1
2
3
1
2
4
1
3
2

Size: 110 Bytes

Values
Apache Arrow
Apache Parquet
Apache DataFusion
Wisconsin-Madison
UW-Madison

Codes (3 bit)
000
001
010
011
001
010
100
001
011
010

Size: 74 Bytes

Symbol table
Apache - a
Arrow - b
Parquet - c
DataFusion - d
...

Values
ab
ac
ad
ef
gf

Codes (3 bit)
000
001
010
011
001
010
100
001
011
010

Size: 14 Bytes

Apply filters along the decoding path

Find contains ”DataFusion”

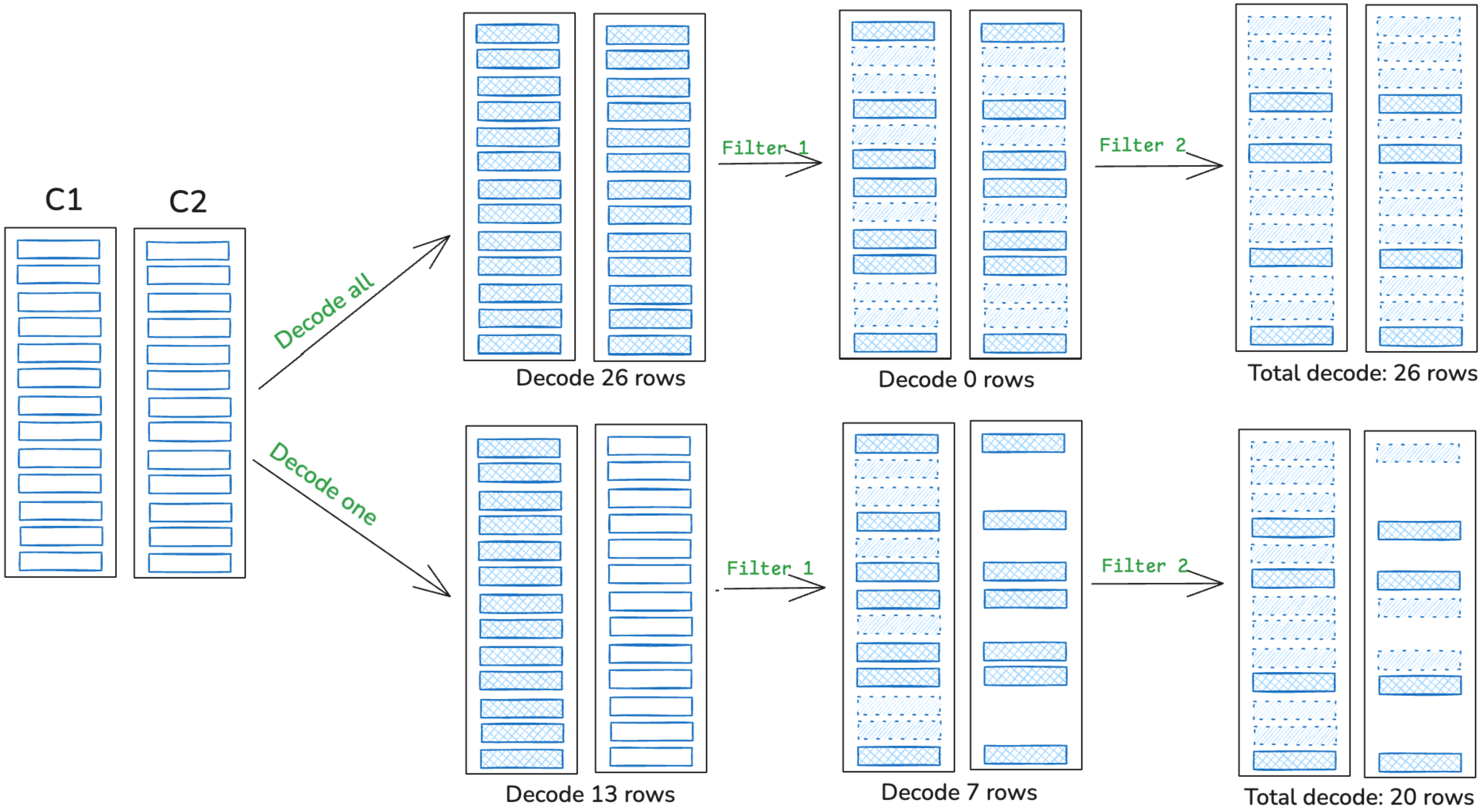
1. Decode the values

2. Find the index in values – 2

2. Return the index in codes

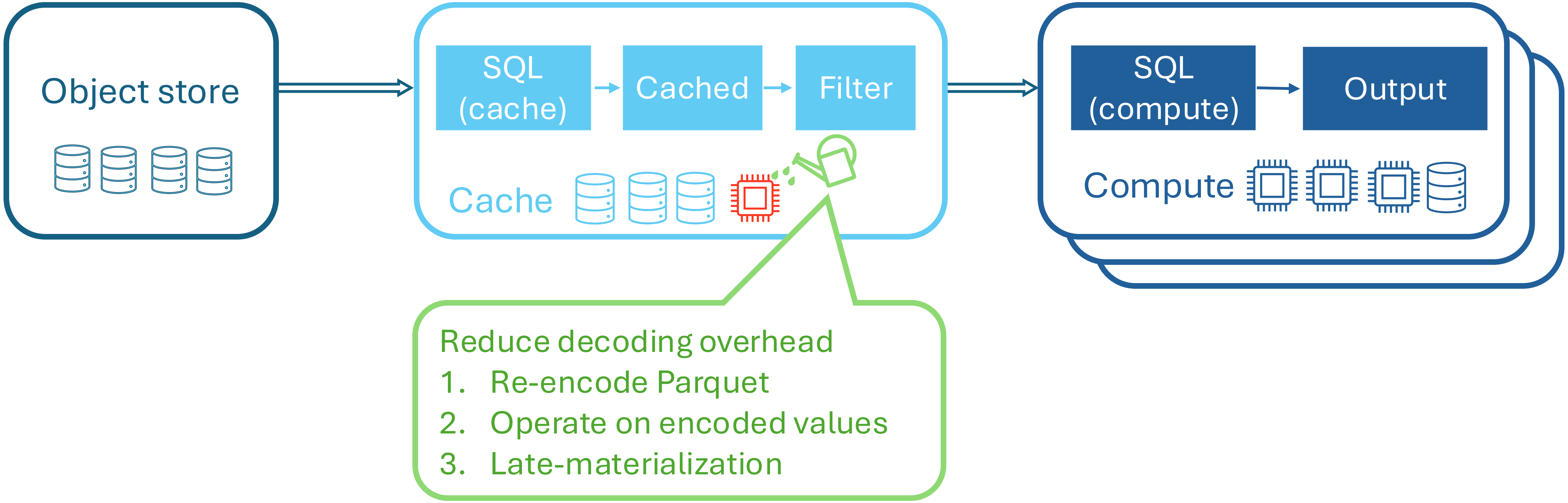
Partial decoding

Random access for late-materialization



Decode only the necessary

SplitSQL: Practical Disaggregated Cache



Reduce decoding overhead

1. Re-encode Parquet
2. Operate on encoded values
3. Late-materialization

Outline

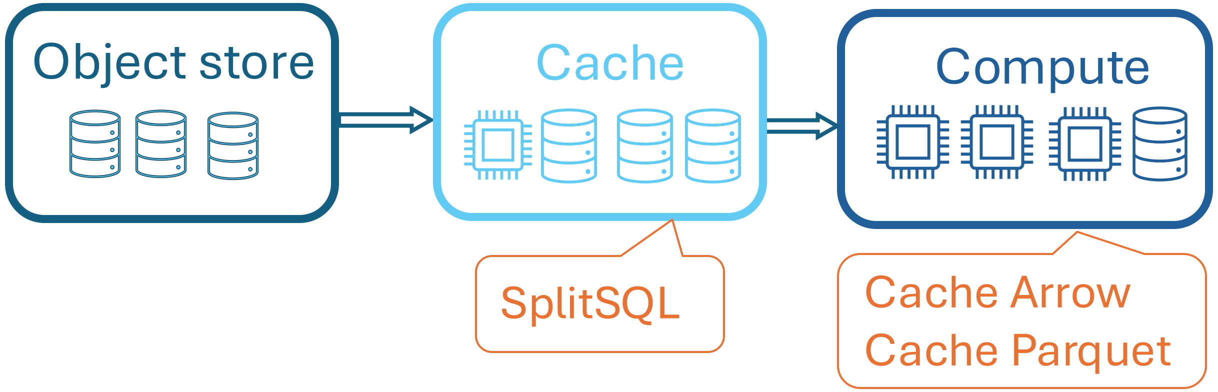
Part 1: Disaggregated cache is the future

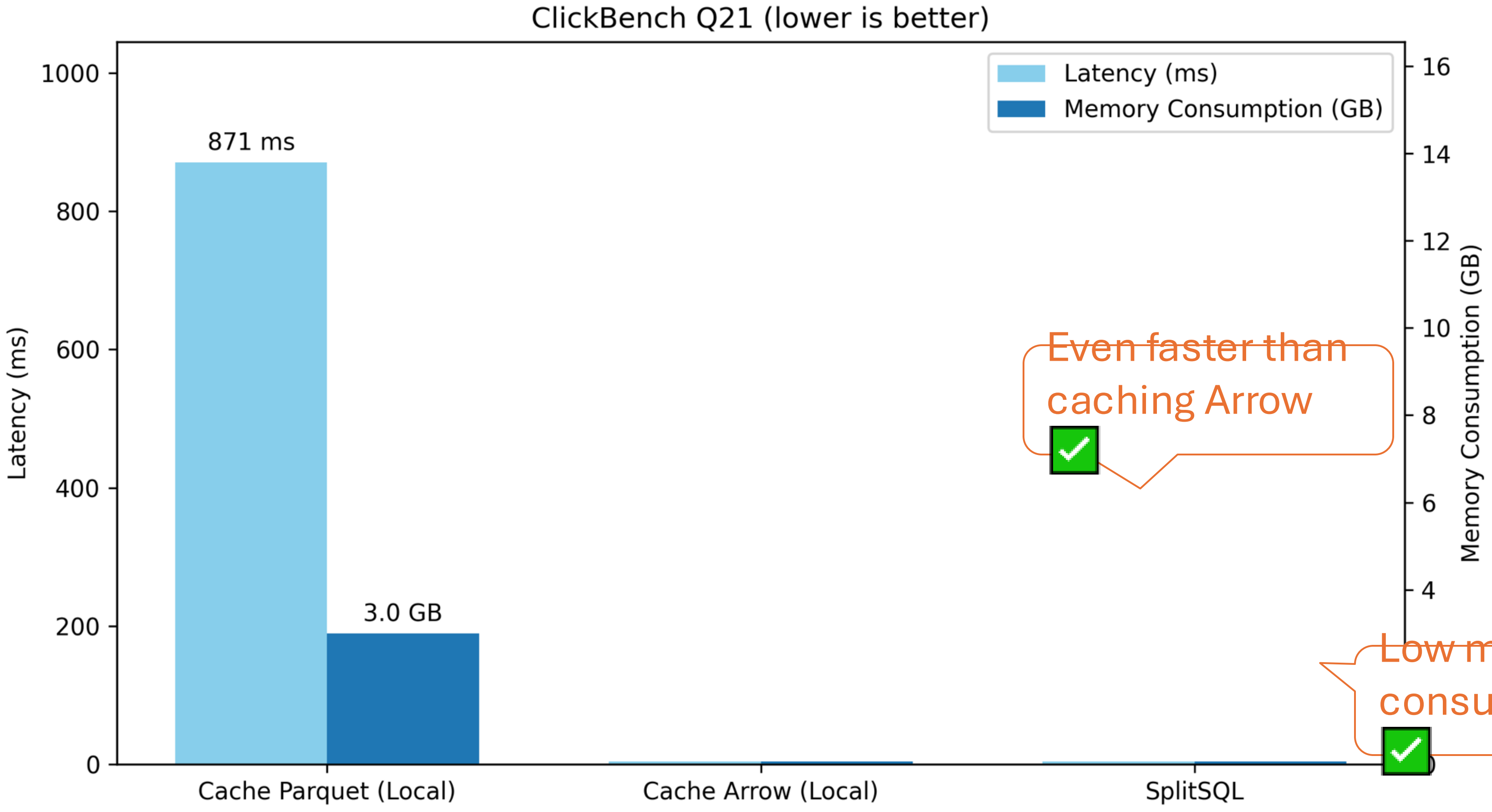
Part 2: To Pushdown or not to pushdown?

Part 3: SplitSQL: pushdown down right

Part 4: Evaluations

Evaluations – Q21



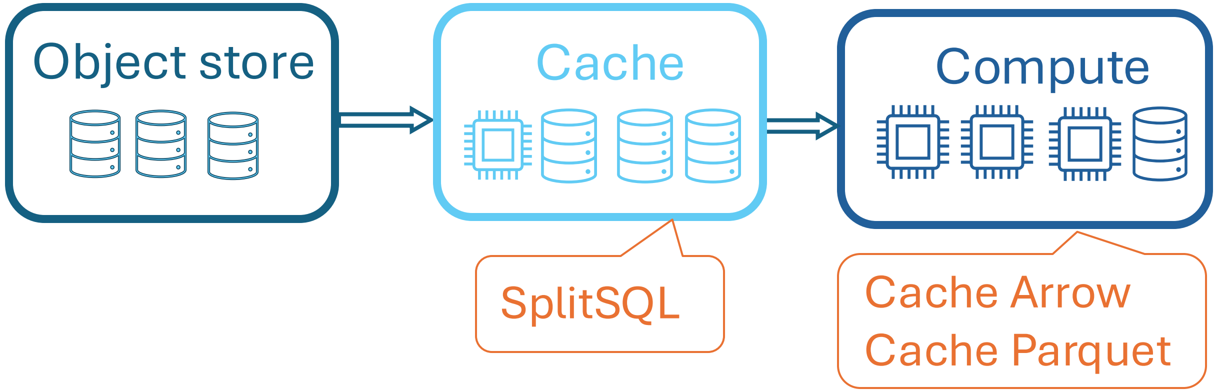


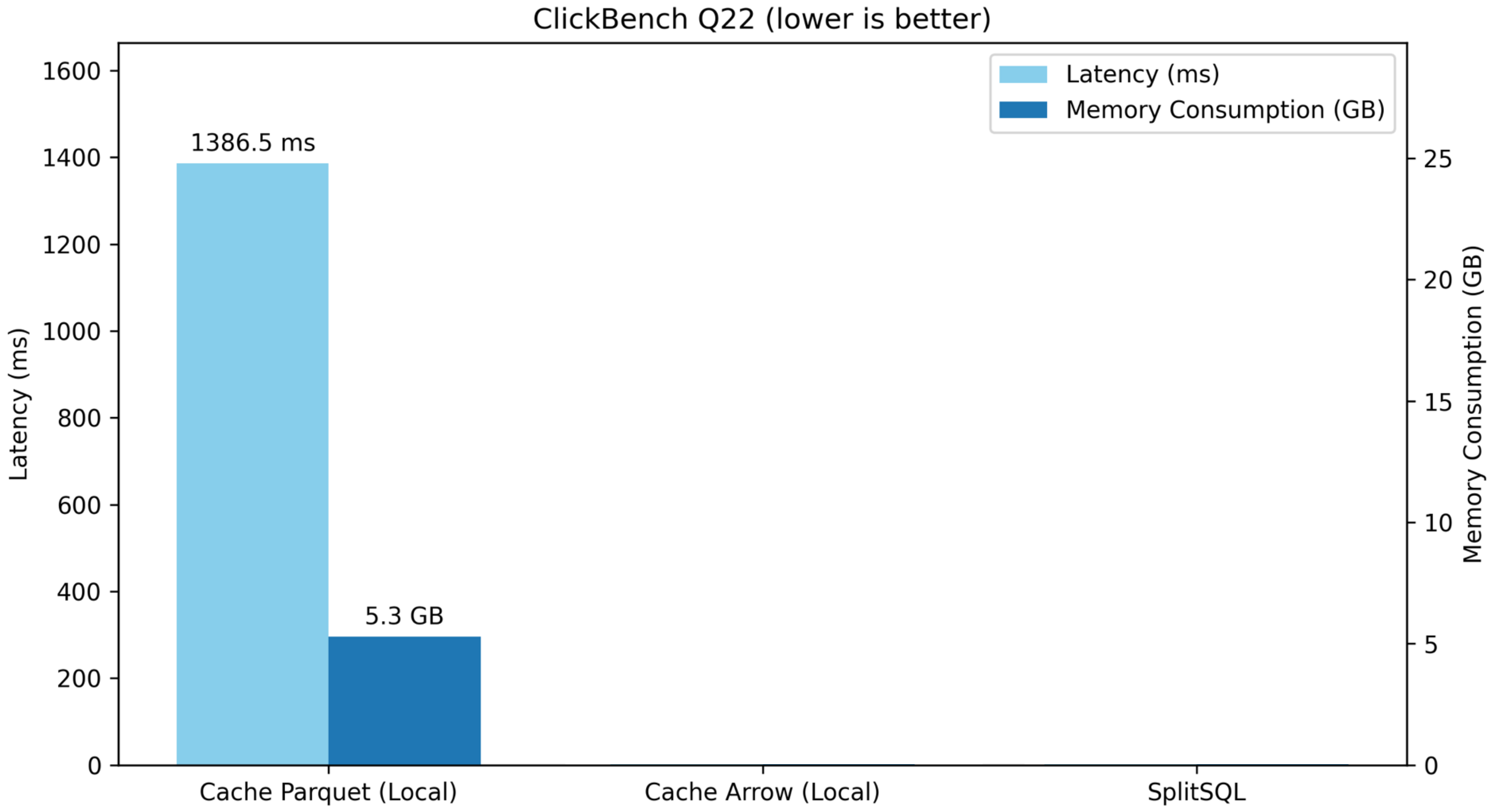
Even faster than caching Arrow ✓

Low memory consumption ✓

```
SELECT
"SearchPhrase", MIN("URL"),
COUNT(*) AS c
FROM hits
WHERE "URL" LIKE '%google%’
AND "SearchPhrase" <> ''
GROUP BY "SearchPhrase"
ORDER BY c DESC LIMIT 10;
```

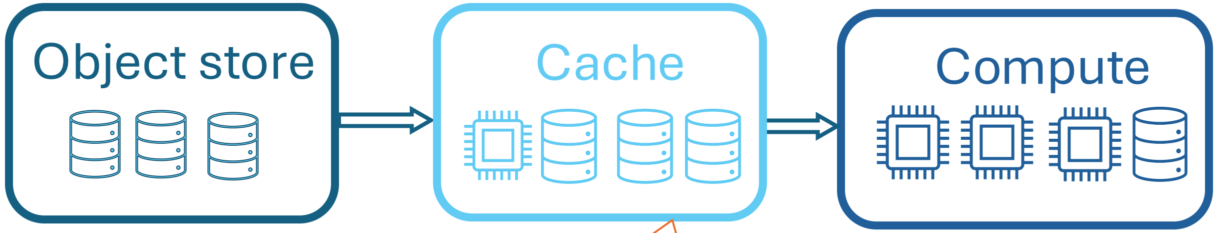
Evaluations – Q22





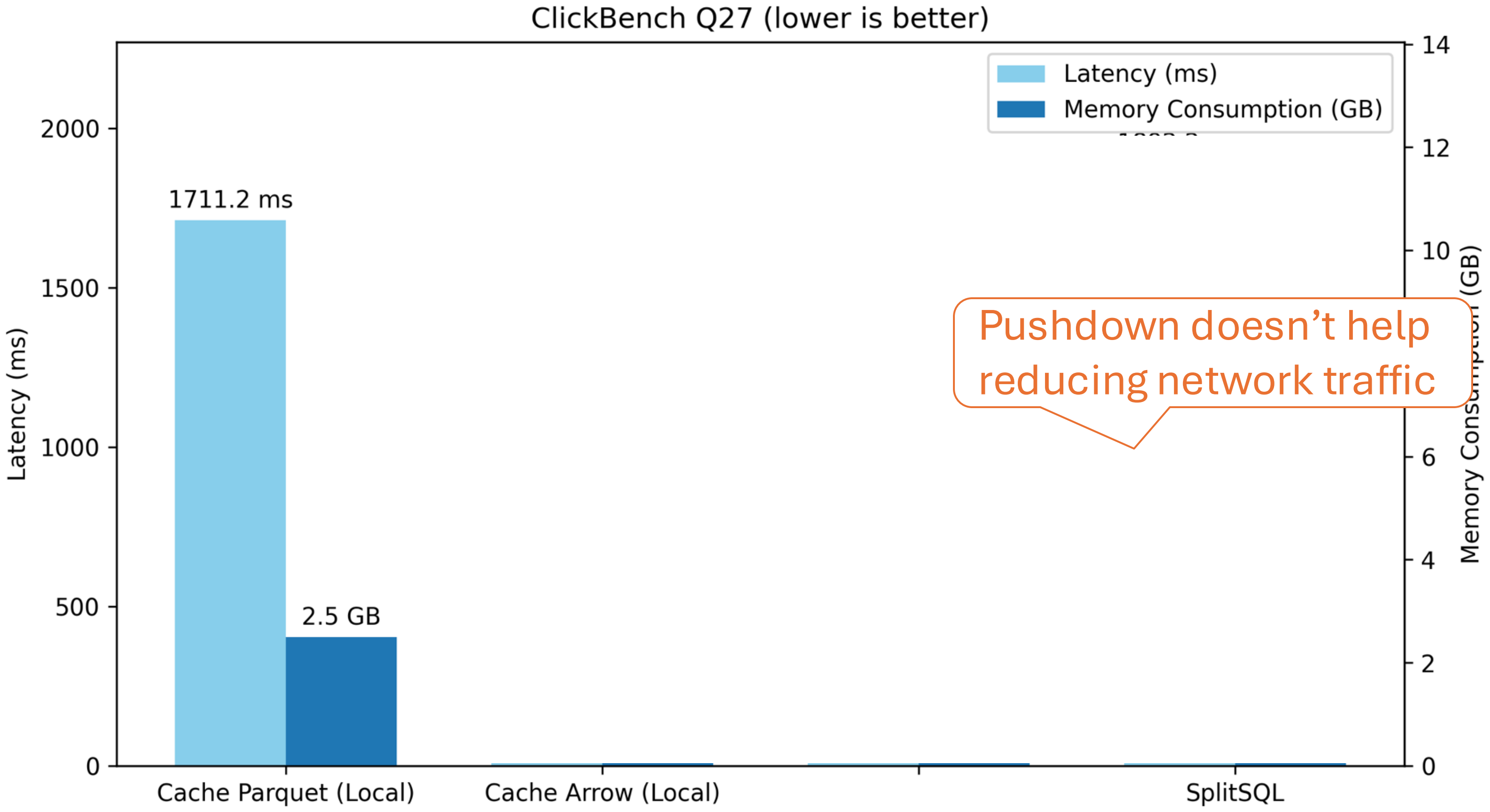
```
SELECT
"SearchPhrase",
MIN("URL"),
MIN("Title"),
COUNT(*) AS c, COUNT(DISTINCT
"UserID")
FROM hits
WHERE
"Title" LIKE '%Google%’
AND
"URL" NOT LIKE '%.google.%' AND
"SearchPhrase" <> ’’
GROUP BY "SearchPhrase" ORDER BY c
DESC LIMIT 10;
```

Evaluations – Q27



SplitSQL

SplitSQL (local)

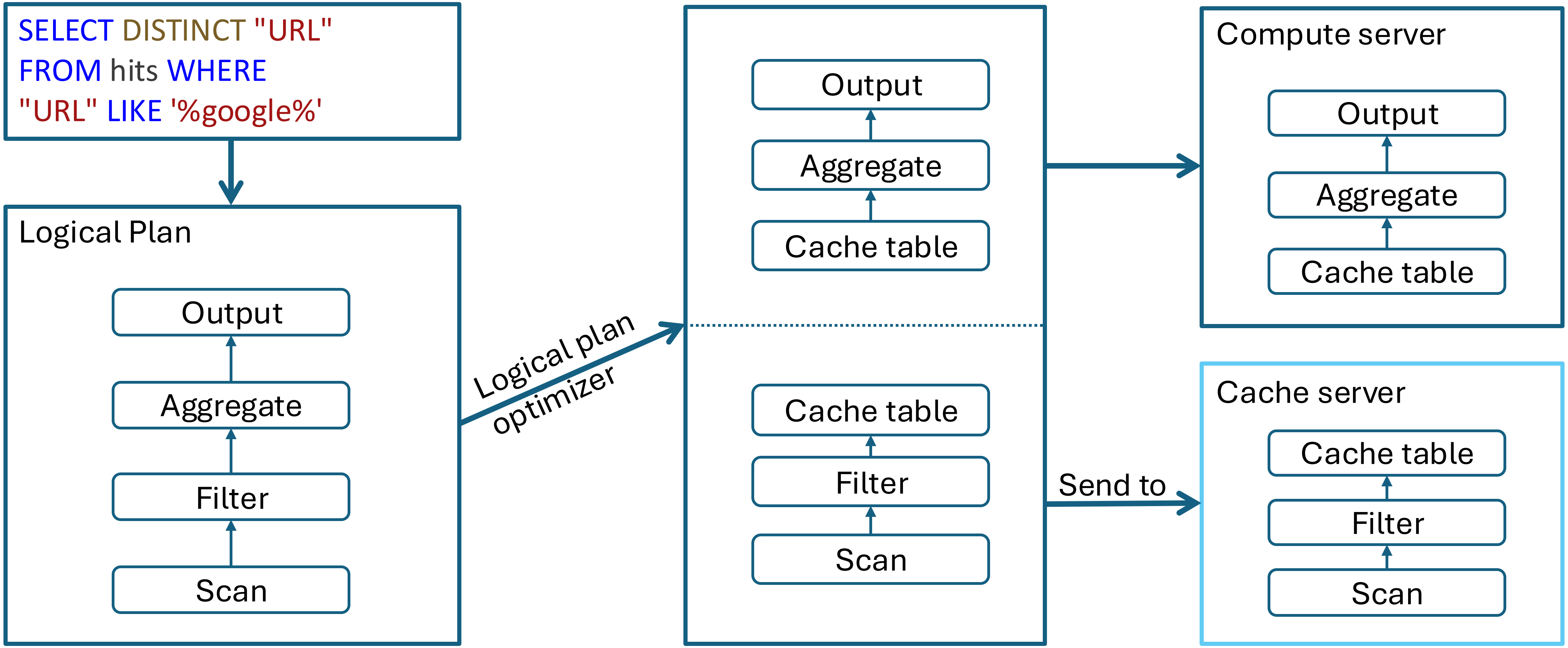


Pushdown doesn’t help reducing network traffic

```
SELECT
"CounterID", AVG(length("URL")) AS
l, COUNT(*) AS c
FROM hits
WHERE "URL" <> ’’
GROUP BY "CounterID" HAVING
COUNT(*) > 100000 ORDER BY l
DESC
LIMIT 25;
```

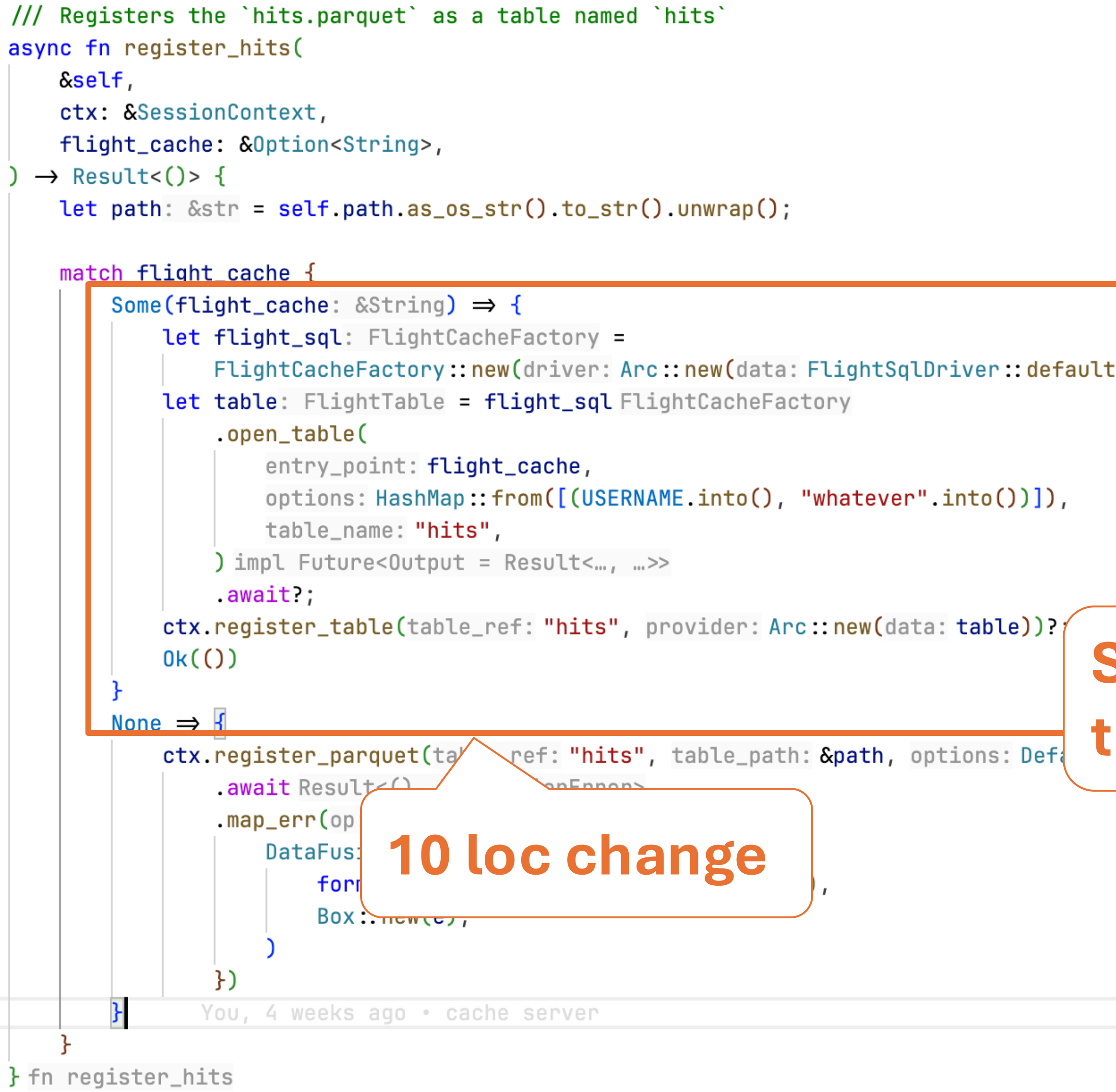
```
SELECT "CounterID", "URL"
FROM "hits"
WHERE "URL" <> ''
```

Implementation in DataFusion



```
SELECT DISTINCT "URL"
FROM hits WHERE
"URL" LIKE '%google%'
```

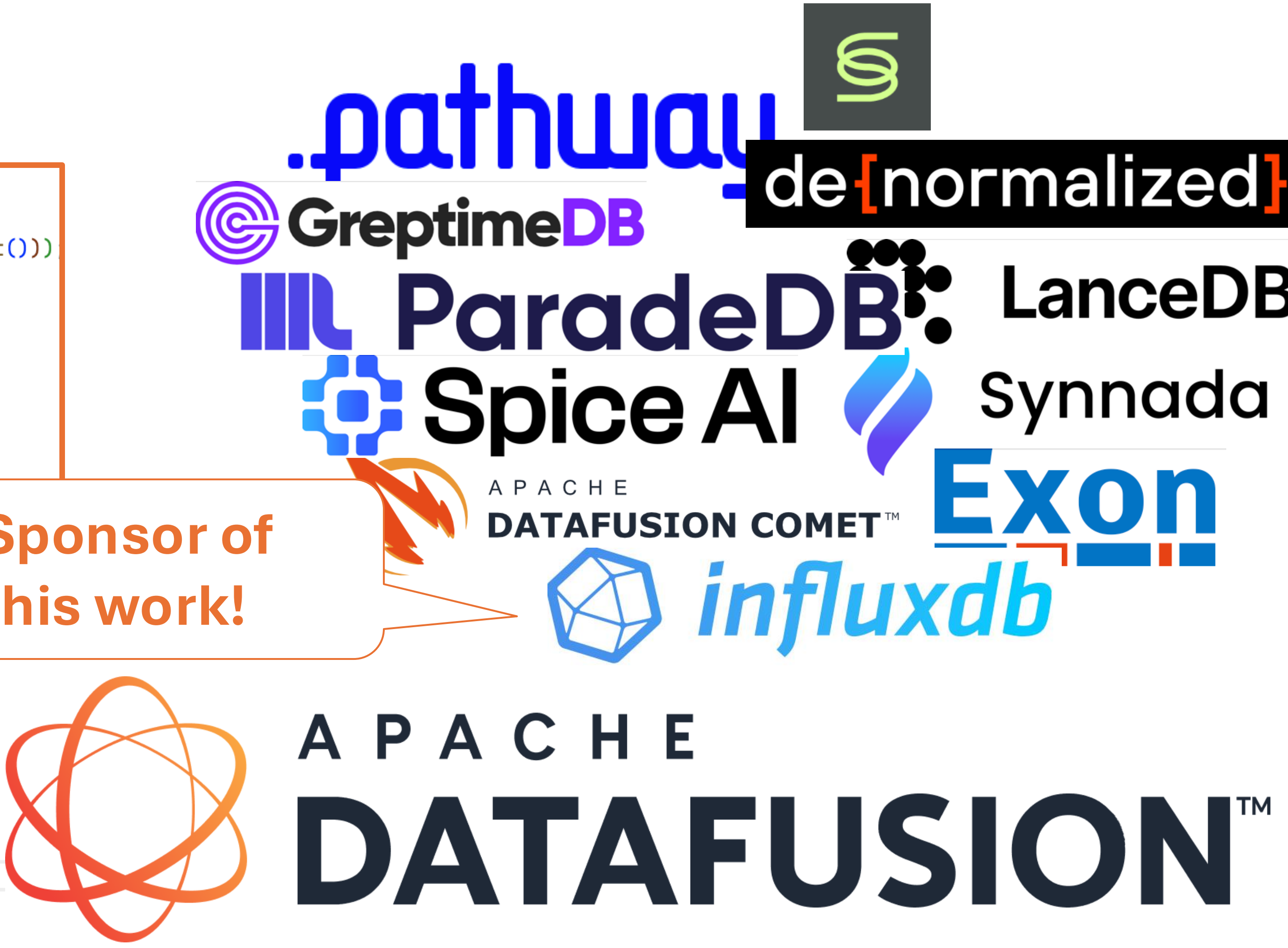
Easy integration to DataFusion universe



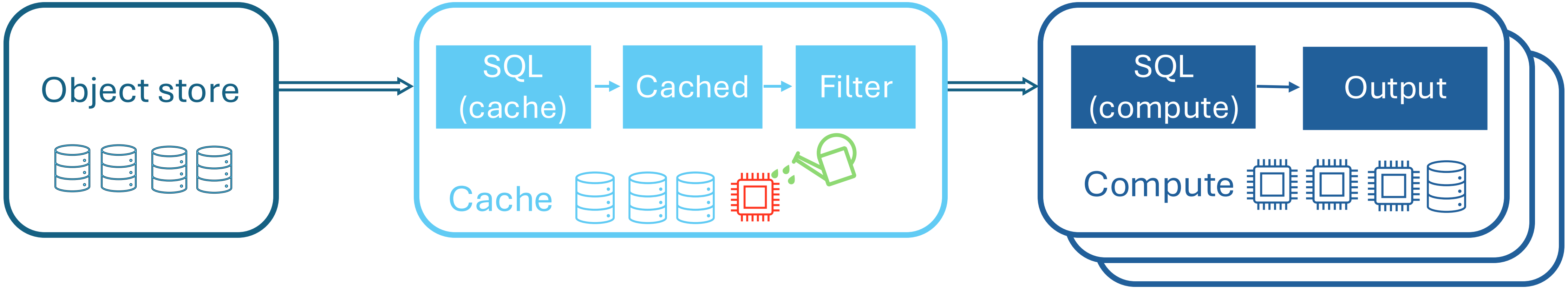
```
/// Registers the `hits.parquet` as a table named `hits`
async fn register_hits(
    &self,
    ctx: &SessionContext,
    flight_cache: &Option<String>,
) → Result<()> {
    let path: &str = self.path.as_os_str().to_str().unwrap();

    match flight_cache {
        Some(flight_cache: &String) ⇒ {
            let flight_sql: FlightCacheFactory =
                FlightCacheFactory::new(driver: Arc::new(data: FlightSqlDriver::default()));
            let table: FlightTable = flight_sql FlightCacheFactory
                .open_table(
                    entry_point: flight_cache,
                    options: HashMap::from([(USERNAME.into(), "whatever".into())]),
                    table_name: "hits",
                ) impl Future<Output = Result<…, …>>
                .await?;
            ctx.register_table(table_ref: "hits", provider: Arc::new(data: table))?;
            Ok(())
        }
        None ⇒ {
            ctx.register_parquet(ta    ref: "hits", table_path: &path, options: Defa
                .await Result<()        onError>
                .map_err(op
                    DataFus
                        for                        ),
                        Box::new(e),
                    )
                })
        }    You, 4 weeks ago • cache server
    }
} fn register_hits
```

10 loc change

Sponsor of this work!



Conclusions & Future work



Disaggregated

- Independently scale
- Well-suited for query with filters

Practical

- Low CPU overhead
- Compatible with FDAP ecosystem
- Works on commodity hardware

Even lower network traffic

- For high-cardinality queries (Q27)
- Aggregate and join push down

Even faster decoding

- Storage-aware encodings – different encodings for memory, SSD, HDD