DATA+AI SUMMIT 2022

Democratizing Metrics at Airbnb

Minerva 2.0 and Beyond

Shao Xie
Airbnb

Toby Mao
Airbnb

ORGANIZED BY databricks

Agenda

History of Data @ Airbnb

What is Minerva?

What challenges Minerva 1.0 faced?

What is Minerva 2.0?

What's next?

History of Data @ Airbnb

2010

- Only 1 full time data analyst at the company. His laptop was the company's data warehouse!

- Queries were run against production databases, and expensive queries cause serious incidents which affected the reliability of airbnb.com!

History of Data @ Airbnb

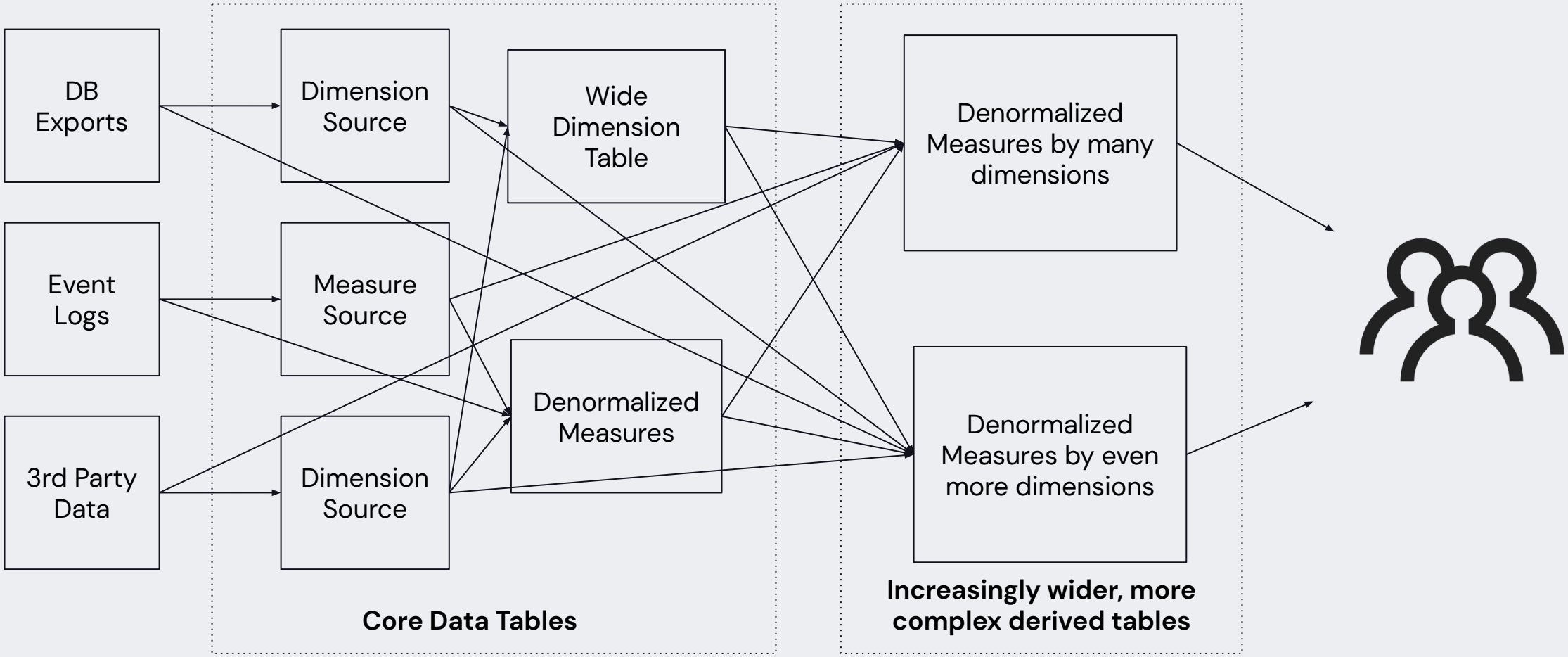
2015

- \+ Built / open sourced Apache Airflow
- \+ Built / open sourced Superset
- \+ Built internal data catalog Dataportal
- \+ Built / scaled Airbnb's Experimentation Platform

- – Different teams reported different numbers
- – Fix was not propagated to downstream
- – Hard to debug data issues
- – Trust from decision makers degraded

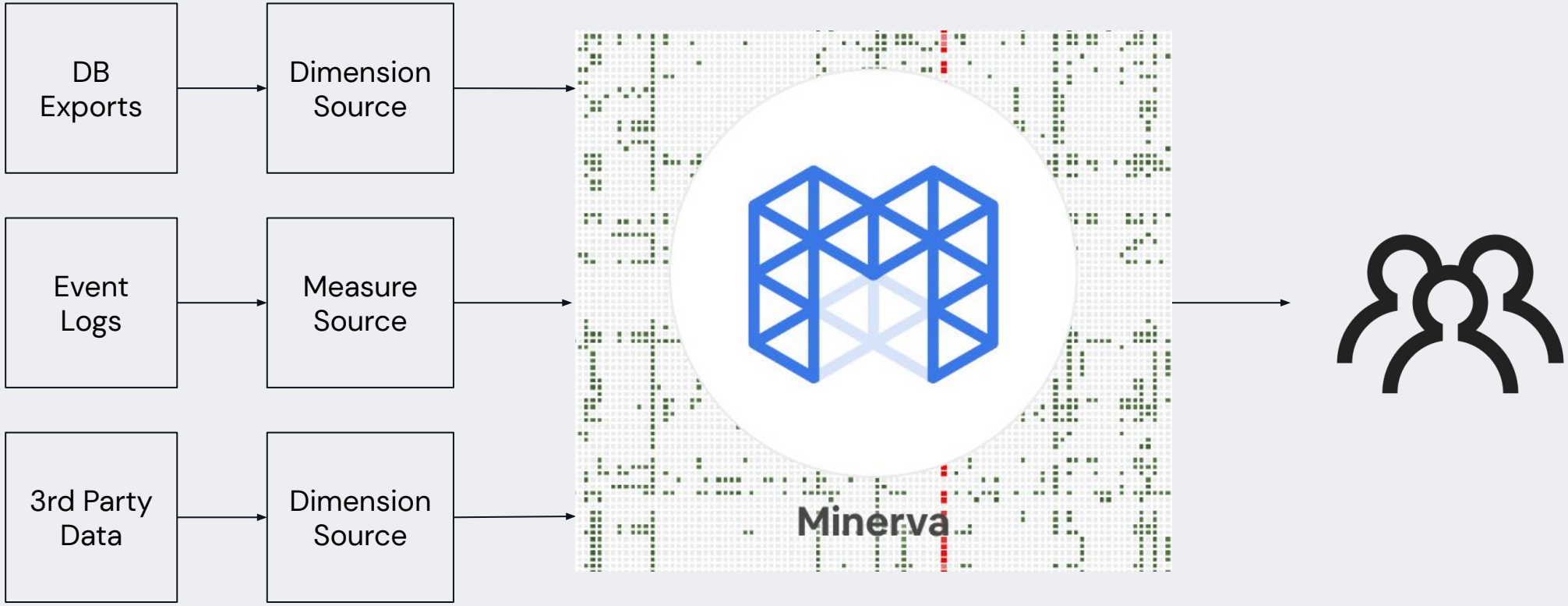
History of Data @ Airbnb

A messy derived table/data consumption ecosystem



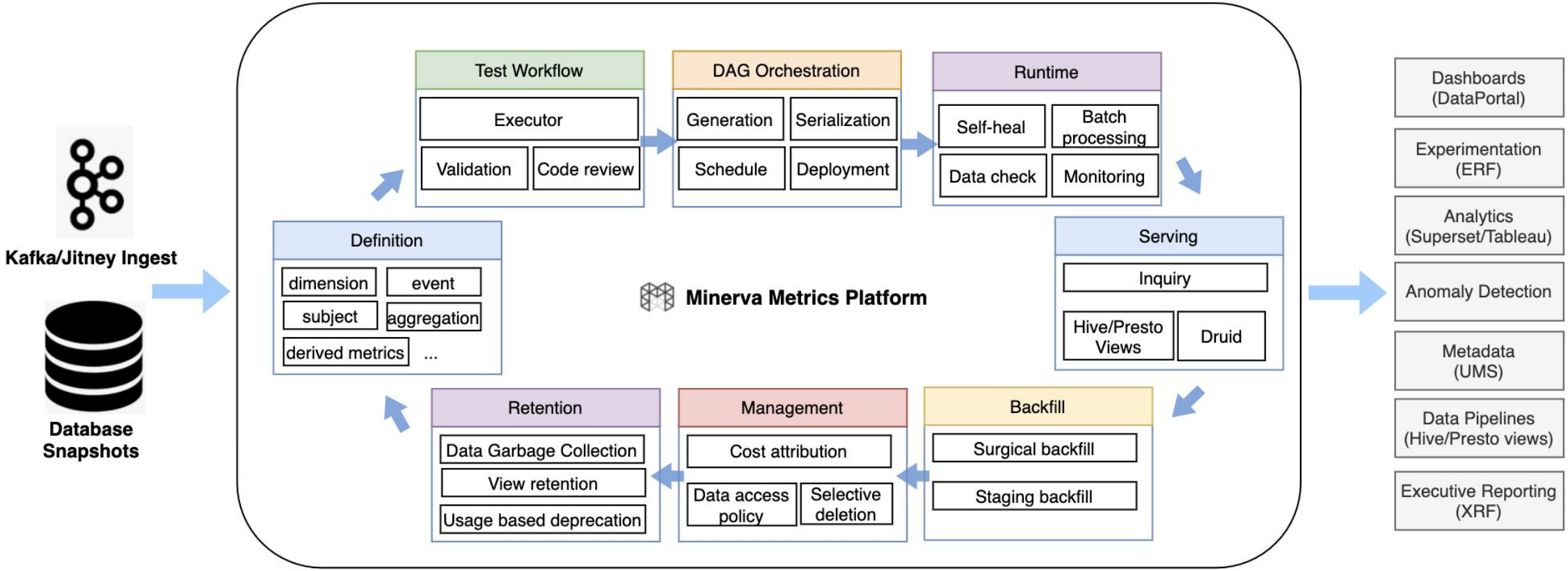
Minerva

Airbnb's solution to achieve metrics consistency at scale



What does Minerva do

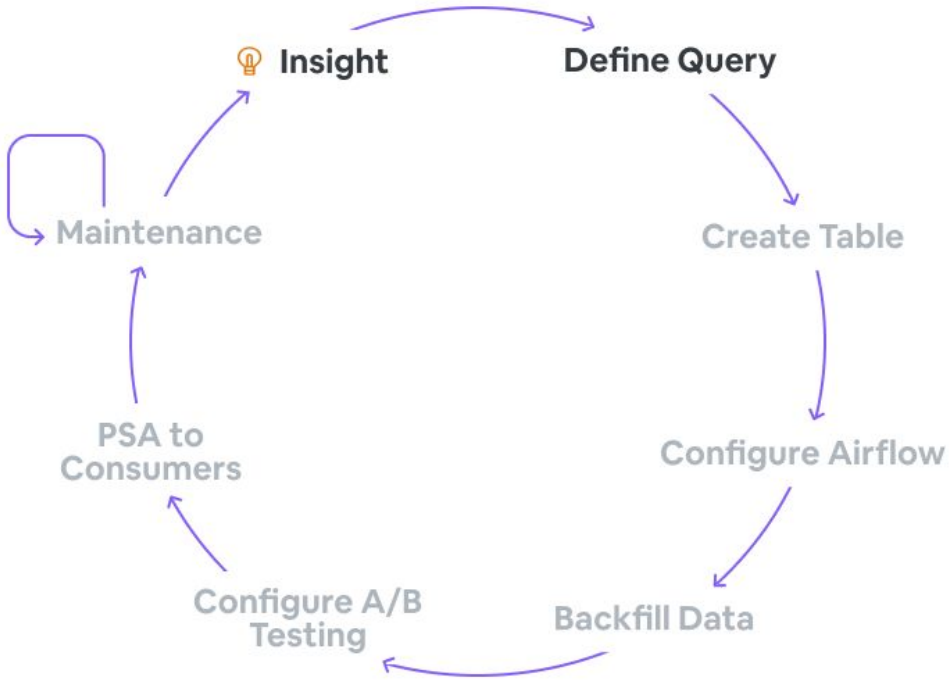
Managing the entire life cycle of metrics and dimensions



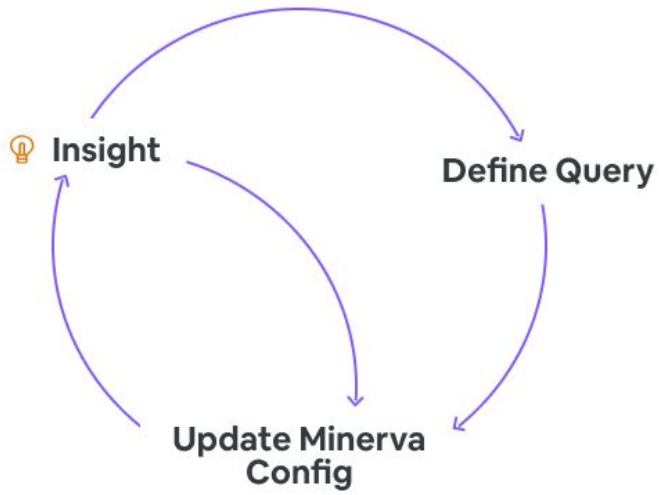
Minerva

Reducing time to insights

Without Minerva



With Minerva



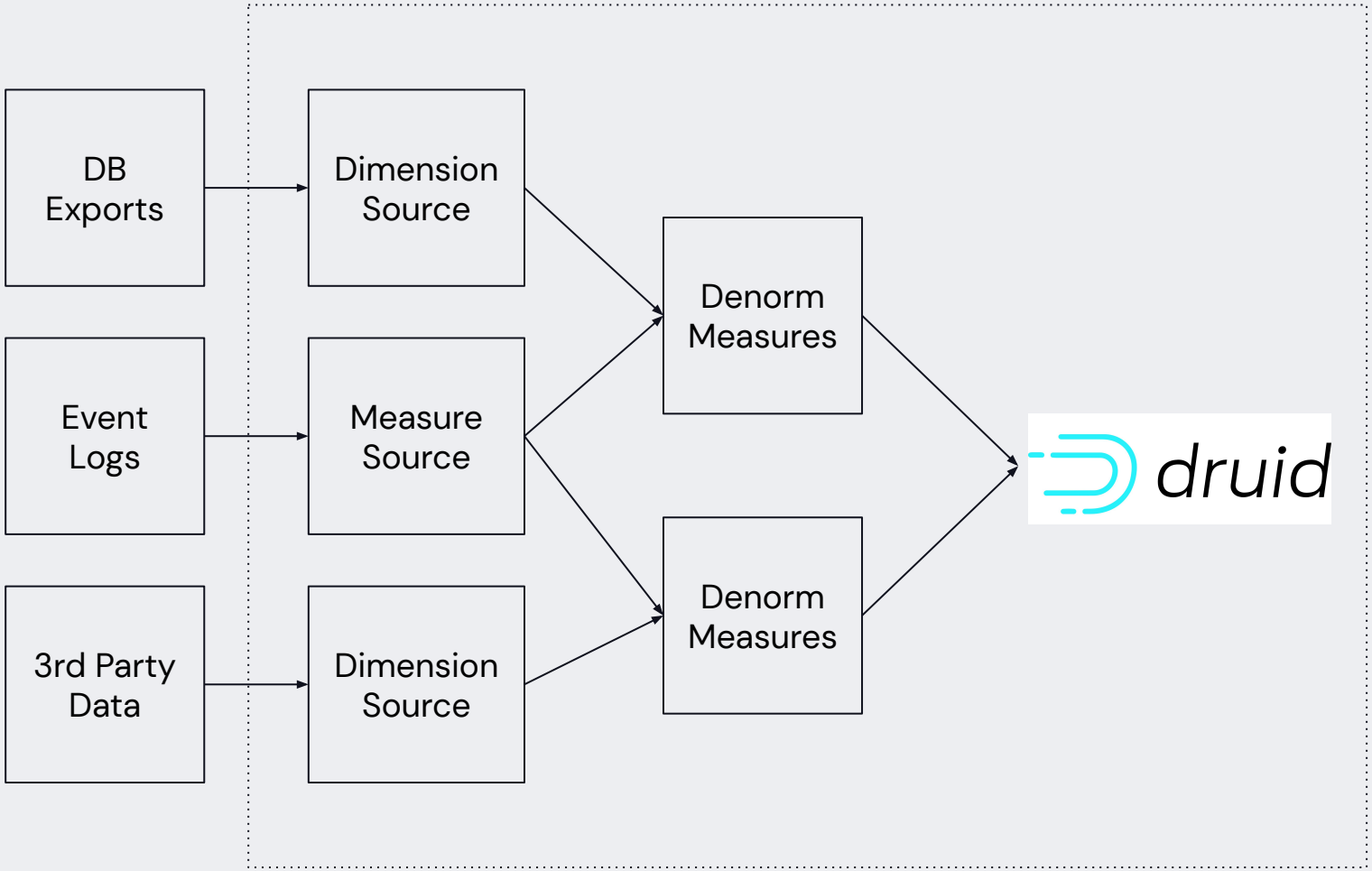
Minerva in Numbers

Define once and use everywhere

30k+	7k+	100+	6P+
Metrics	Dimensions	Teams	Data Size

Minerva 1.0 Challenges

Precomputation



Minerva 1.0 Challenges

Denormalization

Metric

bookings_in_france:

expression: SUM(bookings)

filter: booking_country = 'fr'

Join

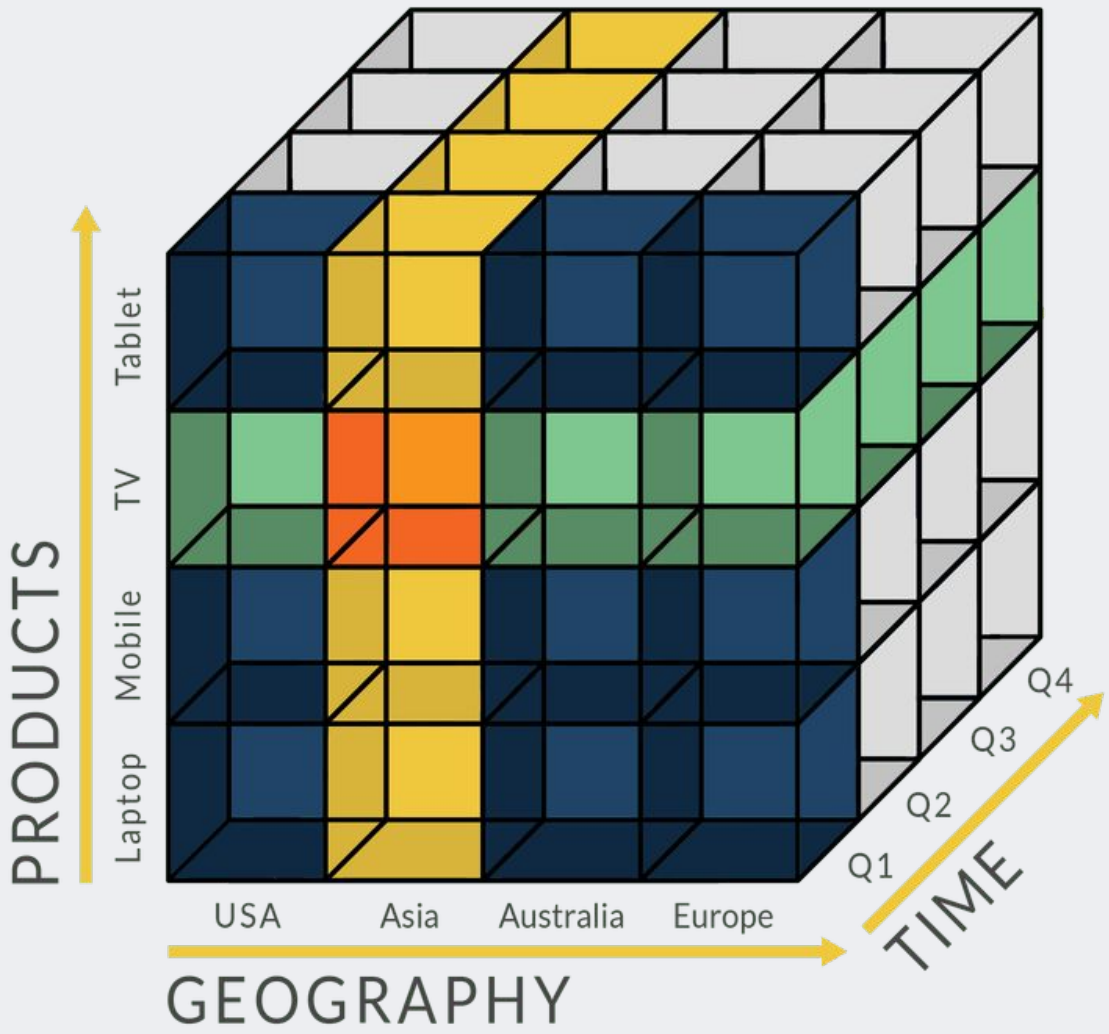
```
SELECT IF (
    booking_country = 'fr',
    bookings,
    NULL
) AS bookings_in_france
FROM stays_reservations s
LEFT JOIN booking_countries b
    ON s.id = b.id
```

Materialization

id	country	bookings_in_france
1	fr	1
2	ca	NULL
3	us	NULL

Minerva 1.0 Challenges

Cubing



Minerva 1.0 Challenges

Precomputation is inflexible

- Need to know what dimensions you care about ahead of time
  - Filtering
  - Granularities
- Need something that's not precomputed?
  - Add it and wait for the backfill - wasted time and money
  - Rewrite it yourself from scratch - bypassing the source of truth

Minerva 1.0 Challenges

Precomputation is expensive

- Precompute everything because we aren't sure what we want!
  - Variants of metrics that only differ on filters
  - Multiple overlapping denormalized tables to ensure MY specific use case is handled
- High Cardinality Dimensions
- Unbounded growth
  - Temporary dimensions and metrics are added but never removed
  - Metric Swamp

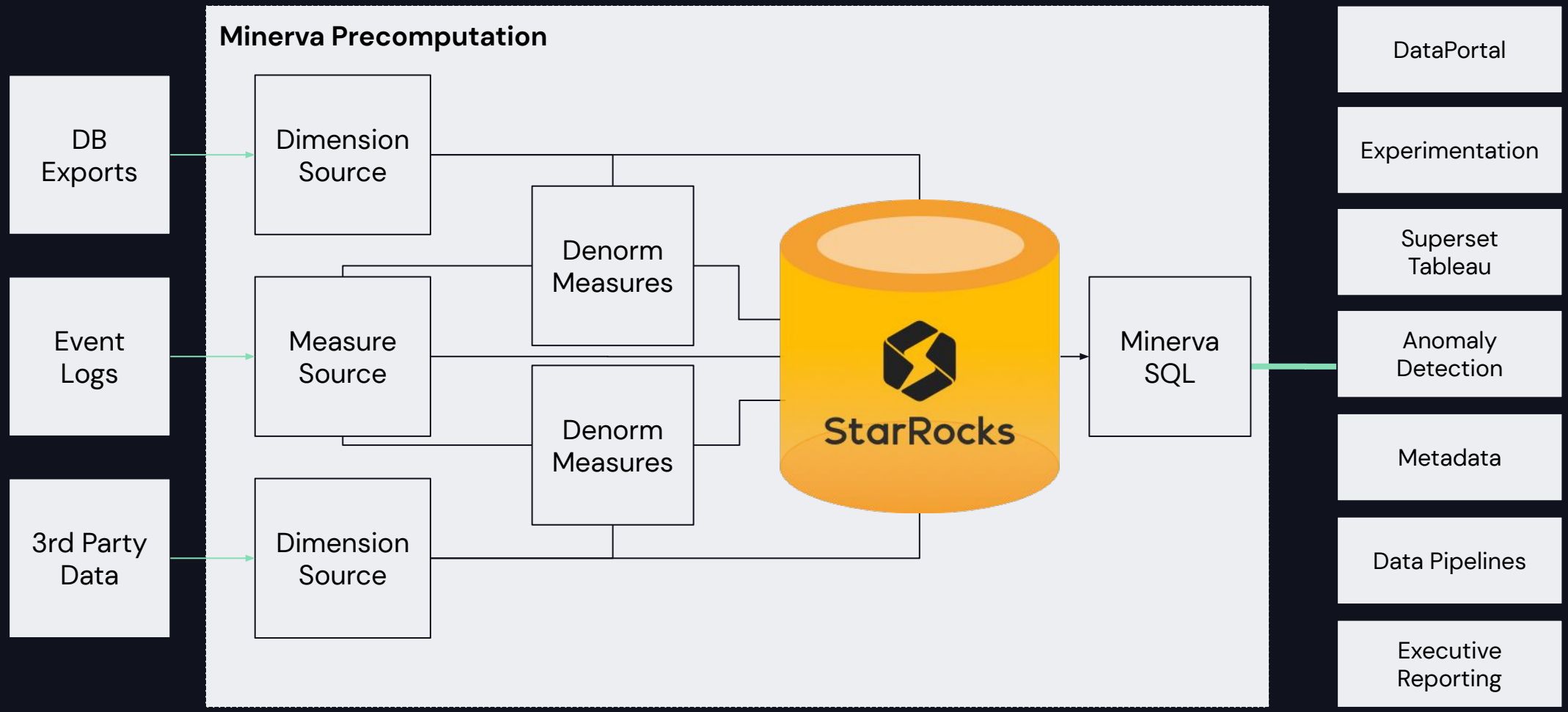
Minerva 2.0

Flexibility and Performance

- On the fly joins
- On the fly aggregations
- Optional denormalizing and cubing

Minerva 2.0

On the fly joins and aggregations



Minerva 2.0

SQL Interface

Client Query

```
SELECT ds,
AGG(bookings_in_france) /
AGG(ios_searches)
FROM minerva.metrics
GROUP BY ds
```

Normalized

```
SELECT ds,
  bookings_in_france / ios_searches
FROM (
        SELECT ds, SUMIF(country = 'fr',
bookings) AS bookings_in_france
        FROM stays_reservations
        LEFT JOIN countries
        GROUP BY ds
) bookings
FULL JOIN (
        SELECT ds, SUMIF(device = 'ios',
        searches) AS ios_searches
        FROM searches
        LEFT JOIN devices
        GROUP BY ds
) searches
        ON ds ...
```

Origin

```
WITH stays_reservations AS (

        SELECT ...

        FROM homes.reservations__fct

        JOIN ...

        JOIN ...
        WHERE ...

)

Normalized Query Goes Here
```

Minerva 2.0

SQL Parsing and Transpilation

- + SQL Lingua Franca of Data
- + Understanding User Queries
- + Fragmented Ecosystem
  - ✓ Hive/Spark
  - ✓ Trino
  - ✓ Snowflake
  - ✓ Starrocks
  - ✓ Druid
- + SQLGlot
  - ✓ Python SQL Parser and Transpiler

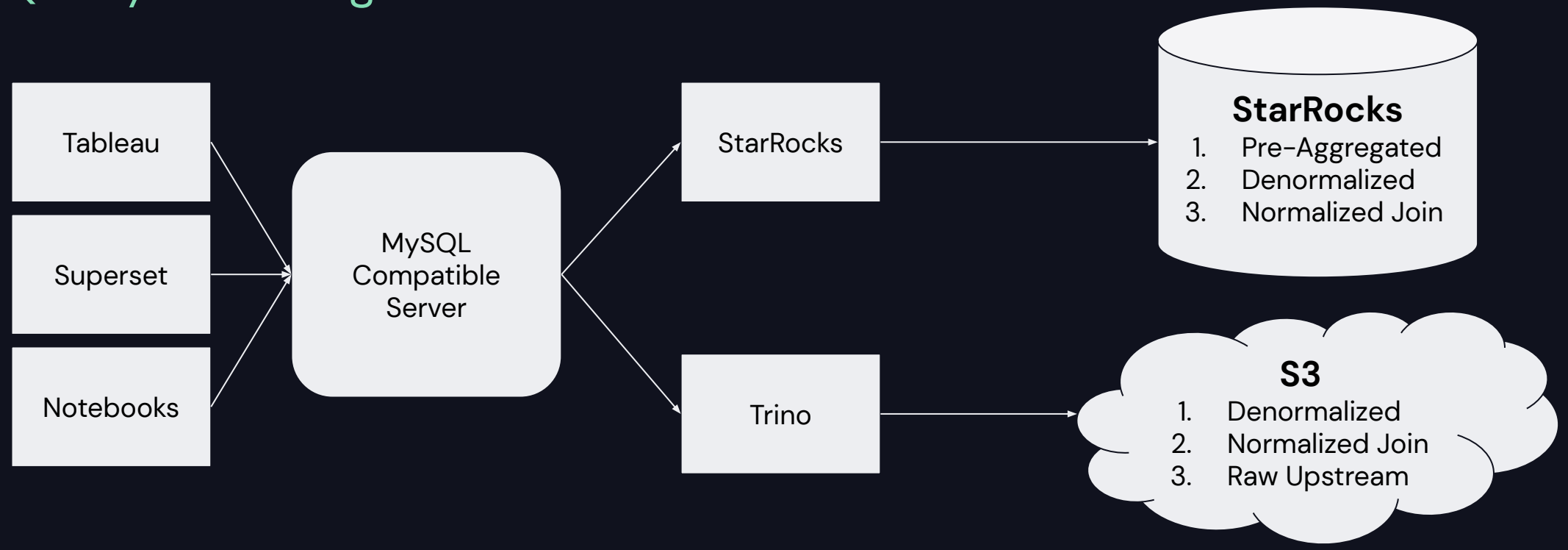
```
In [1]: import sqlglot

In [2]: print(sqlglot.transpile(
   ...:     """
   ...:     SELECT PERCENTILE(x.a, 0.5) AS a,
   ...:            SIZE(COLLECT_SET(UNIX_TIMESTAMP(GET_JSON_OBJECT(event, "$.dat
   ...: e"), "yyyy-mm-dd"))) AS b,
   ...:            COLLECT_LIST(x.c)[0] AS c
   ...:     FROM x
   ...:     LATERAL VIEW EXPLODE(events) e AS event
   ...:     """, read="hive", write="presto", pretty=True
   ...: )[0])
Applying array index offset (1)
Presto does not support exact quantiles
SELECT
  APPROX_PERCENTILE(x.a, 0.5) AS a,
  CARDINALITY(SET_AGG(TO_UNIXTIME(DATE_PARSE(JSON_EXTRACT_SCALAR(event, '$.date'), '%Y-%i-%d')))) AS b,
  ARRAY_AGG(x.c)[1] AS c
FROM x
CROSS JOIN UNNEST(events) AS e (event)

In [3]: sqlglot.parse_one("select a, SUM(b) AS b from x GROUP BY c")
Out[3]:
(SELECT expressions:
  (COLUMN this:
    (IDENTIFIER this: a, quoted: False)),
  (ALIAS this:
    (SUM this:
      (COLUMN this:
        (IDENTIFIER this: b, quoted: False))), alias:
    (IDENTIFIER this: b, quoted: False)), from:
  (FROM expressions:
    (TABLE this:
      (IDENTIFIER this: x, quoted: False))), group:
  (GROUP expressions:
    (COLUMN this:
      (IDENTIFIER this: c, quoted: False))))
```

Minerva 2.0

Query Routing



Minerva 2.0

Experimentation and Machine Learning

- + Interactive Exploration
- + Filtering and segmentation
- + Double aggregation
- + Covariate generation
- + Subject level models
  - ✓ Double ML
  - ✓ Sequential Quantiles

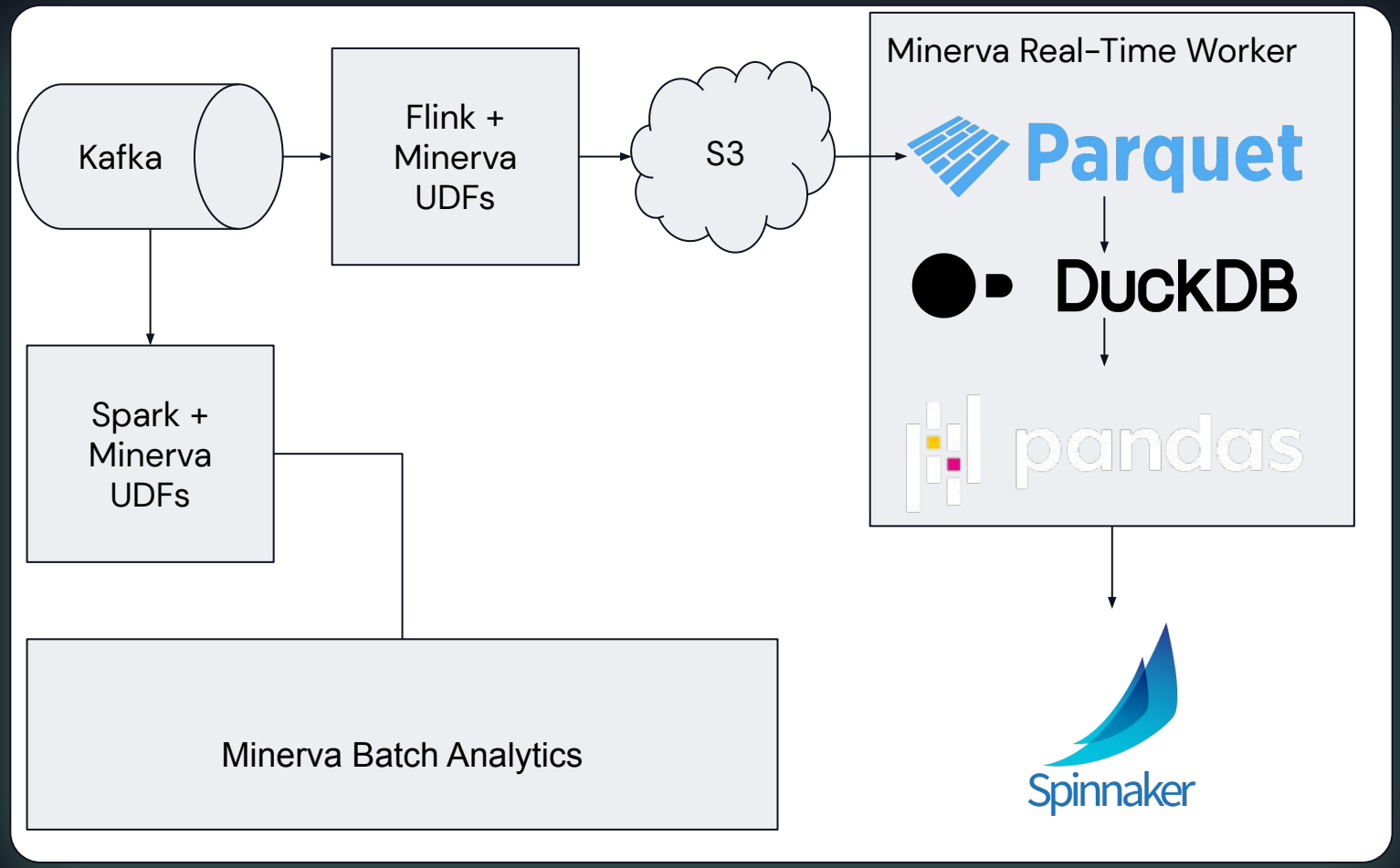
	Treatment 1 Name name name name	Control Name name name name
Metric		
nights_booked	↑15% ±2% mde 11% p-val 0.05 300,013 Subject count 59,013 Sum 0.0544 Mean	N/A% 300,013 Subject count 59,013 Sum 0.0544 Mean
Segment dim_age_group_detailed		
1959 before	↑15% ±2% mde 11% p-val 0.05 300,013 Subject count 59,013 Sum 0.0544 Mean	N/A% 300,013 Subject count 59,013 Sum 0.0544 Mean
1960-1969	↓11% ±1% mde 11% p-val 0.01 300,013 Subject count 59,013 Sum 0.0544 Mean	N/A% 300,013 Subject count 59,013 Sum 0.0544 Mean
1970-1979	↓15% ±2% mde 11% p-val 0.15 300,013 Subject count 59,013 Sum 0.0544 Mean	N/A% 300,013 Subject count 59,013 Sum 0.0544 Mean

Numbers are made up

Minerva 2.0

Near Real-Time Metrics

- + Monitor deploys
- + Automated rollbacks
- + Subject level aggregation
- + Sequential Causal Models
  - ✓ Sequential Sample Ratio Mismatch
  - ✓ Sequential Quantiles
- + Shared definitions between batch jobs with UDFs



	Minerva 1.0	Minerva 2.0
Flexibility	😓 Cannot define adhoc metrics, need to commit and backfill before use	😊 Metrics can be defined in notebooks and computed on-demand
Performance	😊 Fast because everything is precomputed	😐 Fast when precomputed, slower when not precomputed
Extensibility	😓 Not extensible because the only consumption model is through druid.	😊 The core output is SQL which can be used across any application
Reliability	😓 Complex DAGs with many materialized tables has more opportunities for failures	😊 Limiting the number of tables to materialize allows us to have less moving parts
Cost	😓 Extremely high because we're pre-computing and storing everything even if it's not used	😊 Pay for only what we use
Maintainability	😓 Things are only added, never removed because causing a metrics swamp	😊 The flexibility of computing metrics without materializing allows users to have a higher bar to what they want to productionize

What's Next

- Enable anomaly detection
- Build dedicated UI
- Open source Minerva

Thank You!

Antoine Creux, Alex Deng, Barak Alon, Chris Giroir, Chris Williams, Donghan Zhang, Erik Iverson, Jenny Liu, Jingwei Lu, Maggie Zhu, Nick Riabov, Nikos Diamantopoulos, Robert Chang, Philip Weiss, Tanyoung Kim, Xiaohui Sun, Zack Loebel-Begelman