DATA+AI SUMMIT 2022

So Fresh and So Clean

Learn How to Build Real-Time Warehouses on Lakehouse

Franco Patano
Lead Product Specialist, Databricks

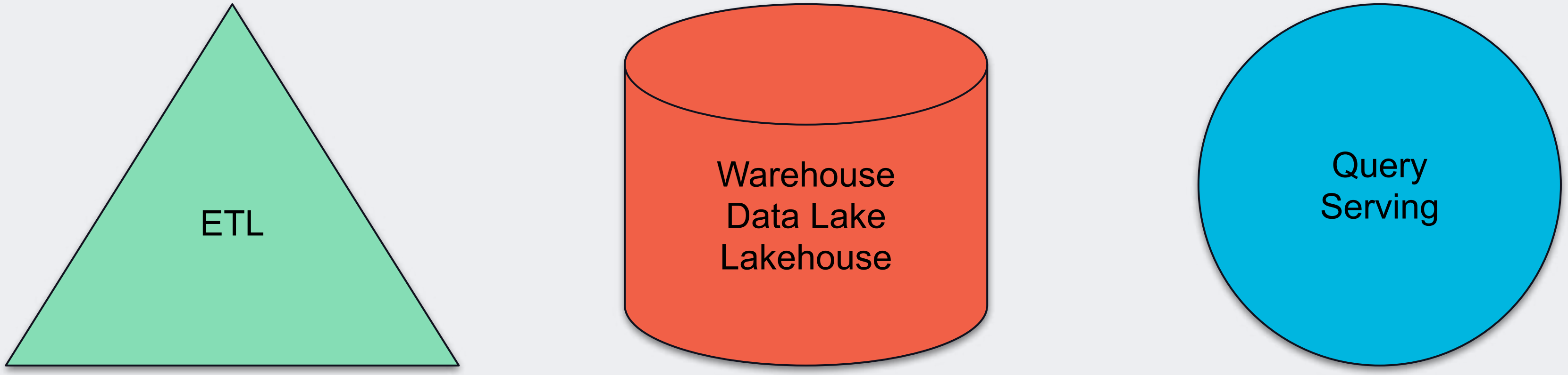
Dillon Bostwick
Sr. Solutions Architect, Databricks

Shannon Barrow
Sr. Solutions Architect, Databricks

ORGANIZED BY databricks

Data Platform Needs

ETL,Storage and Query Serving

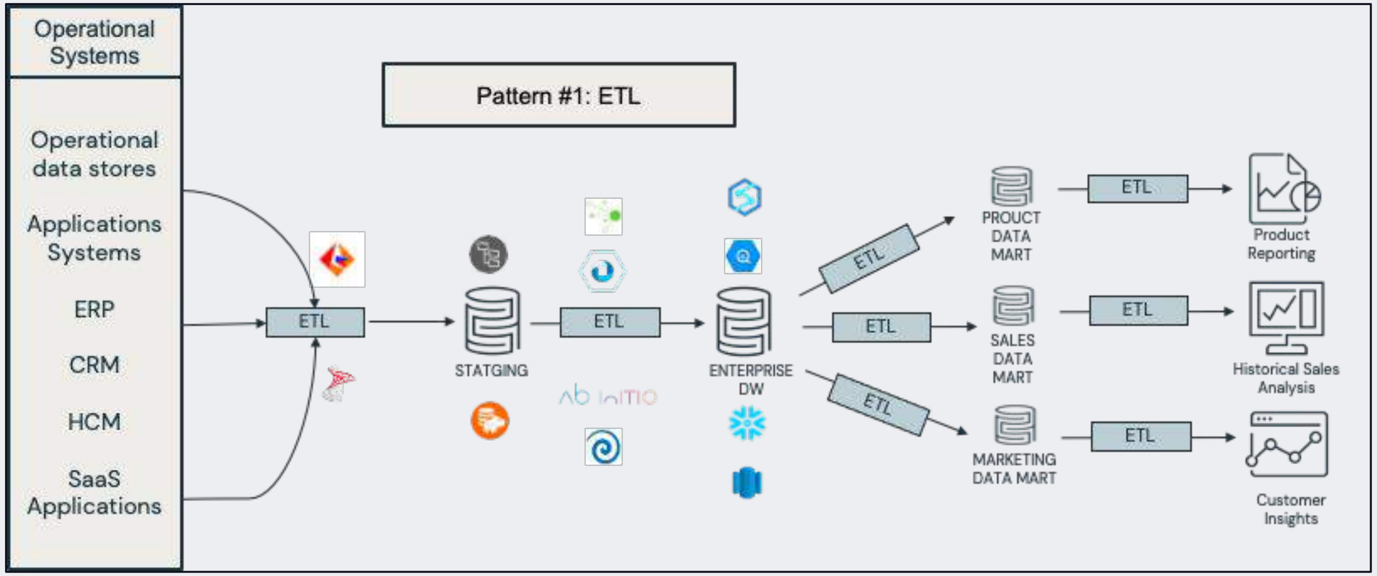


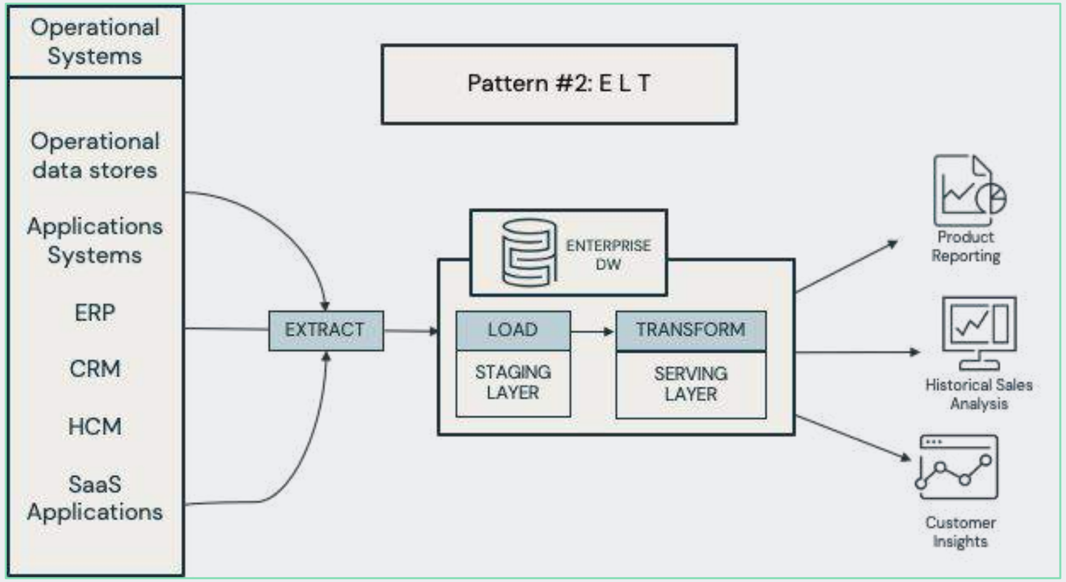
Delta Live Tables

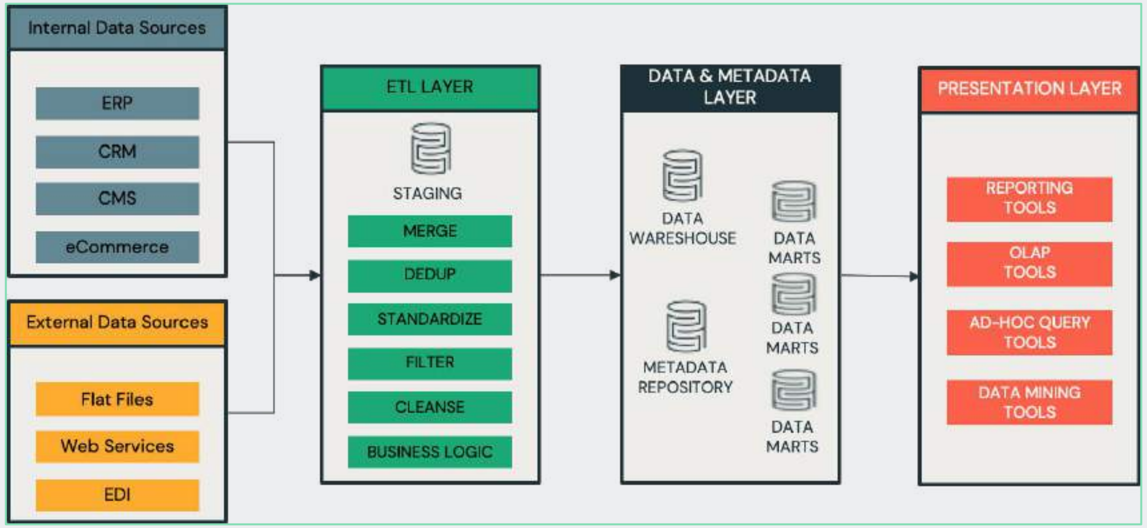
Delta Lake

Databricks SQL

Data Warehousing is ETL/ELT and Query Serving

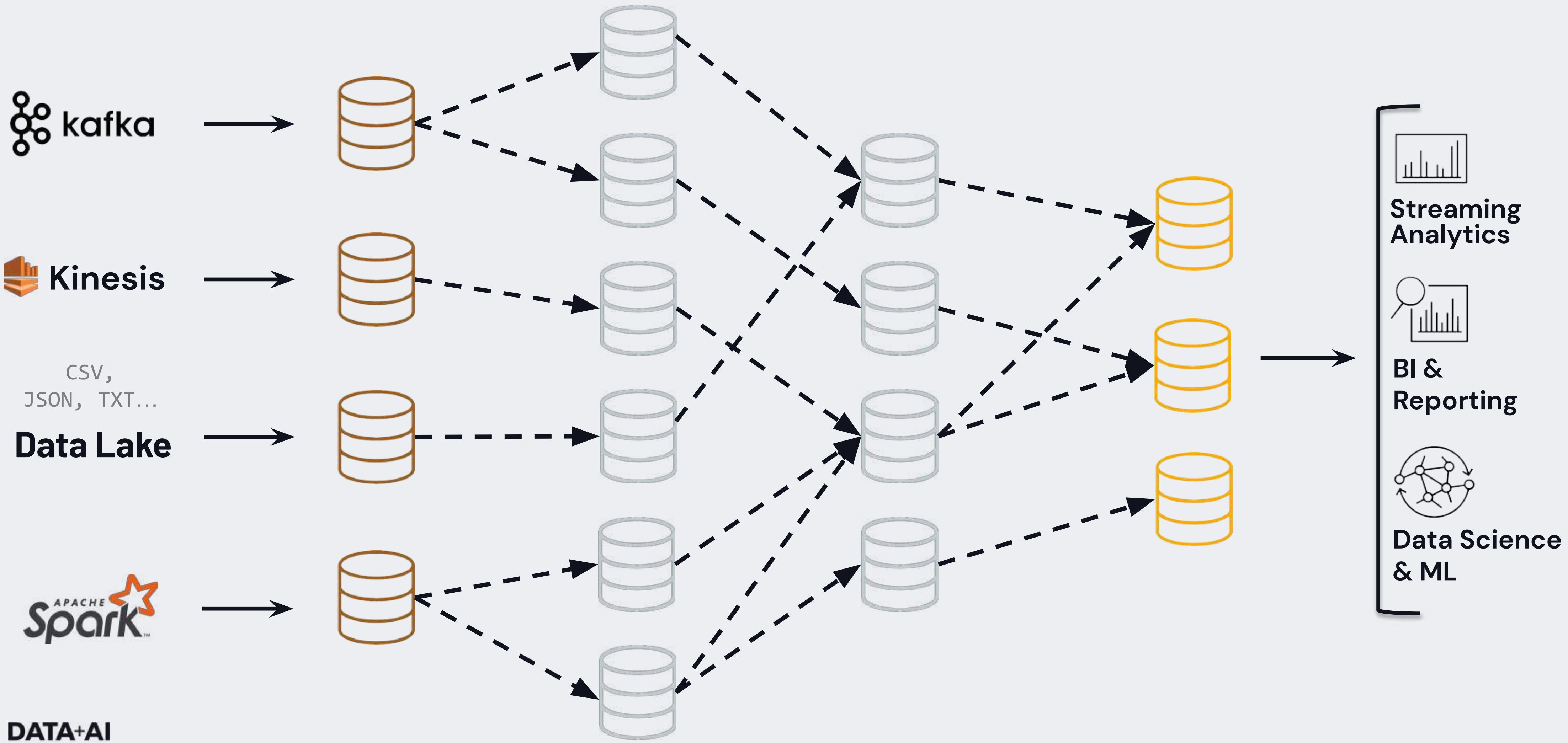






But the reality is not so simple

Maintaining data quality and reliability at scale is complex and brittle



Data Lake

DELTA LAKE

An open approach to bringing data management and governance to data lakes

Better reliability with transactions

48x faster data processing with indexing

Data governance at scale with fine-grained access control lists

Data Warehouse

Delta Lake is the foundation of the Lakehouse

DELTA LAKE

An open format storage layer built for lake-first architecture

ACID Transactions, Time travel, Schema enforcement

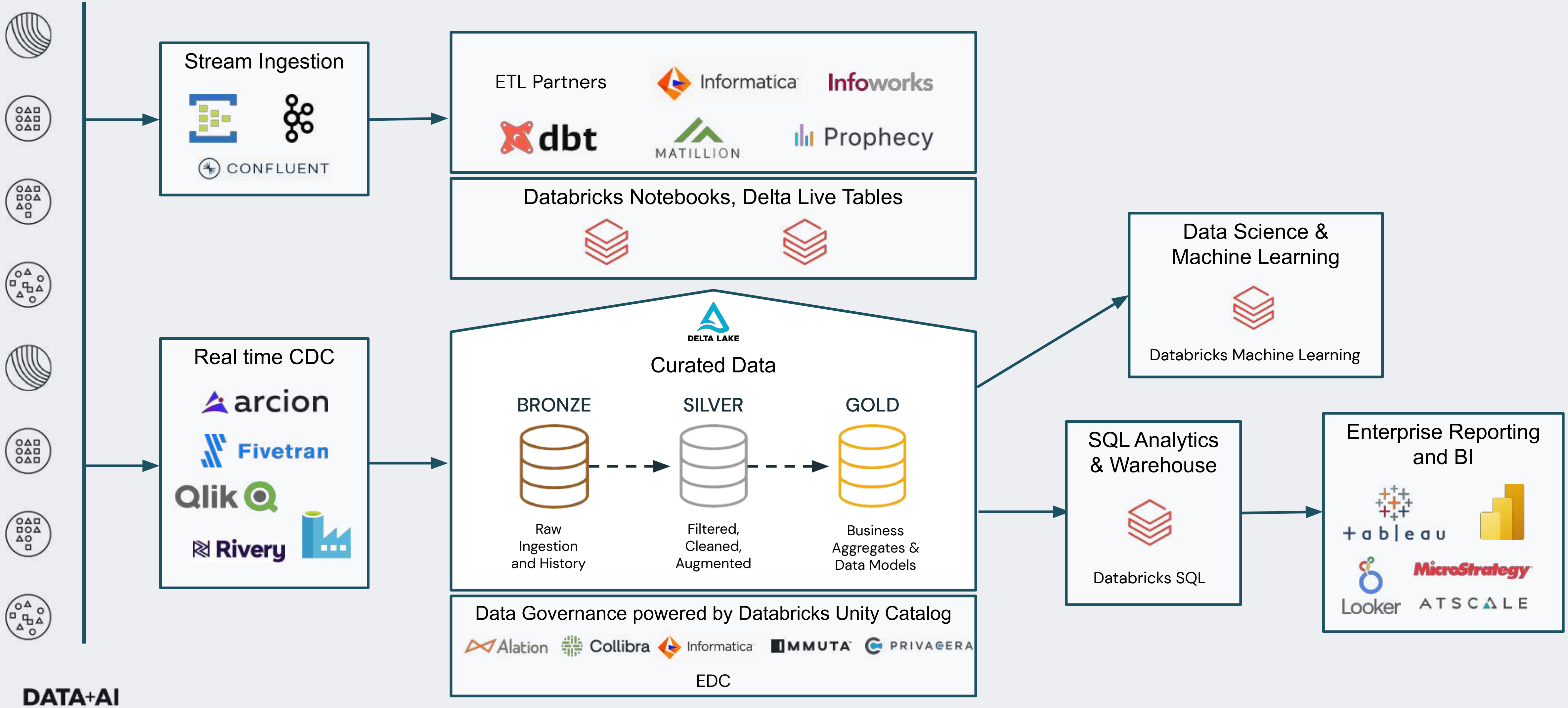
Advanced indexing, Caching, Auto-tuning

Fine-grained, role-based access controls

Streaming & batch, Analytics & ML

Python, SQL, R, Scala

Modern Data Warehousing on Databricks



Serverless compute for Databricks SQL

Instant, elastic & zero-management compute

In Preview

- Quickly setup **instant**, **elastic SQL warehouse** – decoupled from storage – Powered by Photon
- Automatically determines instance types and configuration for best price/perf (up to 12x)
- **High concurrency** built-in, automatic load balancing
- Intelligent **workload management** and faster reads from cloud storage
- Instant startup, greater availability, and **40% average lower overall costs with serverless**

Public Preview! | Private Preview | Coming Soon

What/Who is TPC?

The TPC is a non-profit focused on developing data-centric benchmark standards and disseminating objective, verifiable data to the industry.

https://www.tpc.org/

Widely known: TPC-DS

TPC-DS is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance. The benchmark provides a representative **evaluation of performance as a general purpose decision support system**. A benchmark result **measures query response time** in single user mode, query throughput in multi user mode and data maintenance performance for a given hardware, operating system, and data processing system configuration under a controlled, complex, multi-user decision support workload. The purpose of TPC benchmarks is to provide relevant, objective performance data to industry users. TPC-DS **enables emerging technologies**, such as **Big Data systems**, to execute the benchmark. The TPC-DS **Price/Performance** metric is expressed as Price/QphDS@Size for Version 2 and Price/kQphDS@Size for Version 3.

As Jim Gray and others already stated in a paper of 1985[1], "computer performance is difficult to quantify". The only ***"reasonable metrics"*** are **cost (price/performance) and throughput**.

TPC-DS is a Query Serving benchmark of 99 different queries to determine the price performance of a SQL Serving System.

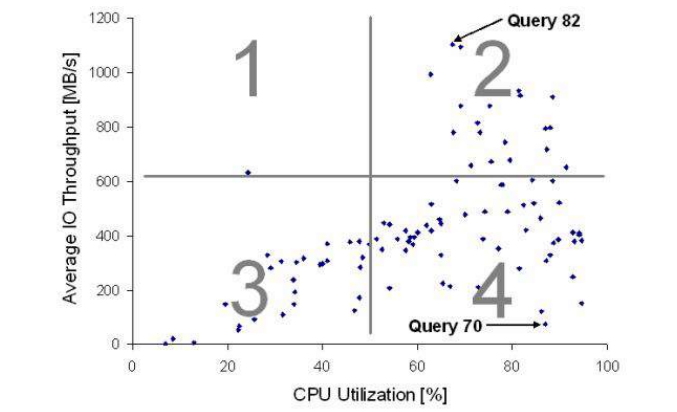


Figure 4: Characterization by Resource Utilization

Experiment

Can Databricks SQL Warehouses handle concurrency demands?

How would a SQL Endpoint/Warehouse scale when 10 parallel runs of TPC-DS 99 Power run, repeated twice?

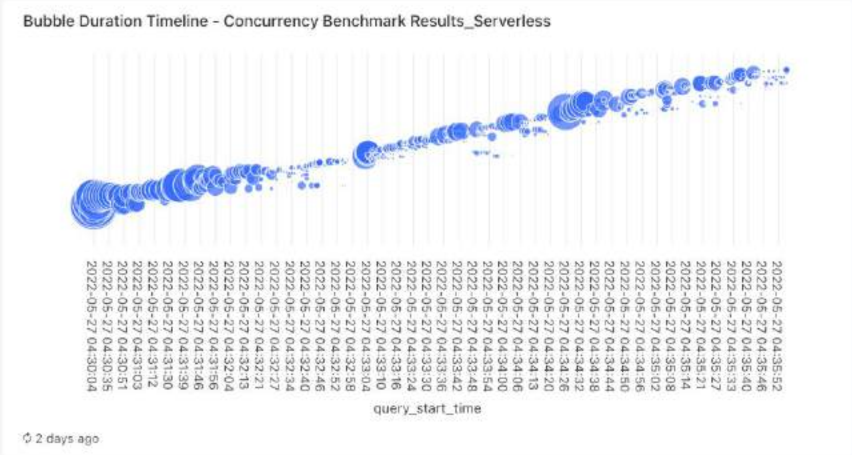
Large Serverless SQL Warehouse 1 to 10 Scaling

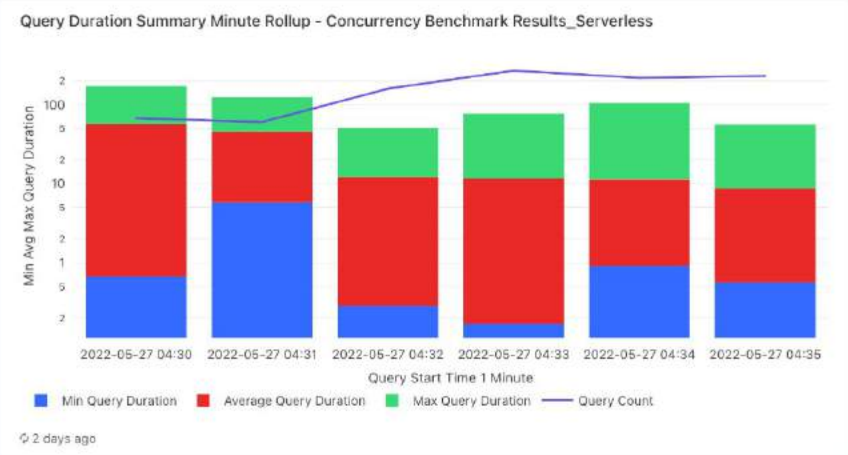
Results

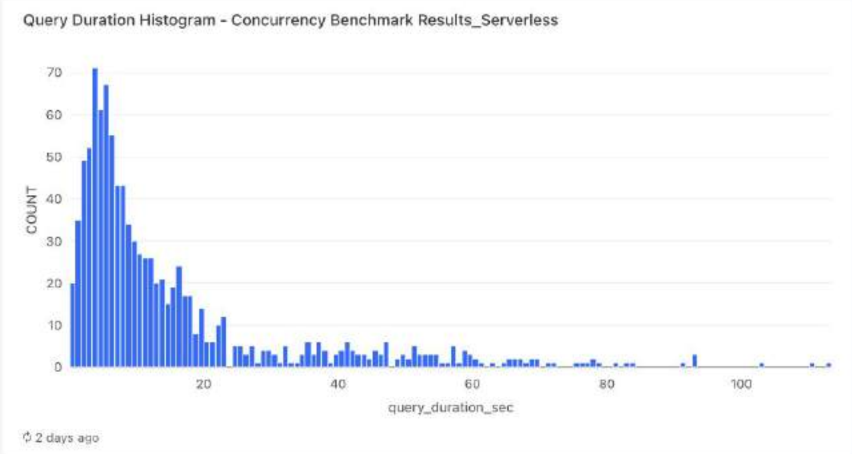
Took 7 minutes to serve 1980 queries and cost $22 in total

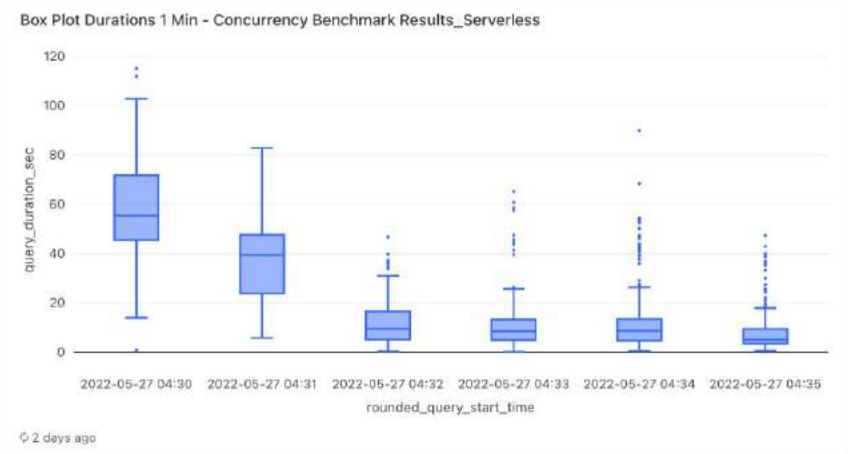
Serverless is $.70 per DBU, and the Large Warehouse scaled up to 7 clusters at its peak. running this same workload on the best cloud data warehouse on the market, Snowflake, it would probably cost around $37.

33 queries ran in 1 second or less!









TPC-DI

Data Integration (DI), also known as **ETL**, is the analysis, combination, and *transformation* of data from a variety of ***sources*** and `formats` into a **unified data model** representation. Data Integration is a key element of data ~~warehousing~~ lakehousing, application integration, and business analytics.

Main Concepts of TPC-DI

TPC-DI uses data integration of a factitious Retail Brokerage Firm as model:

- Main Trading System
- Internal Human Resource System
- Internal Customer Relationship Management System
- Externally acquired data

Operations measured use the above model, but are not limited to those of a brokerage firm

They capture the variety and complexity of typical DI tasks:

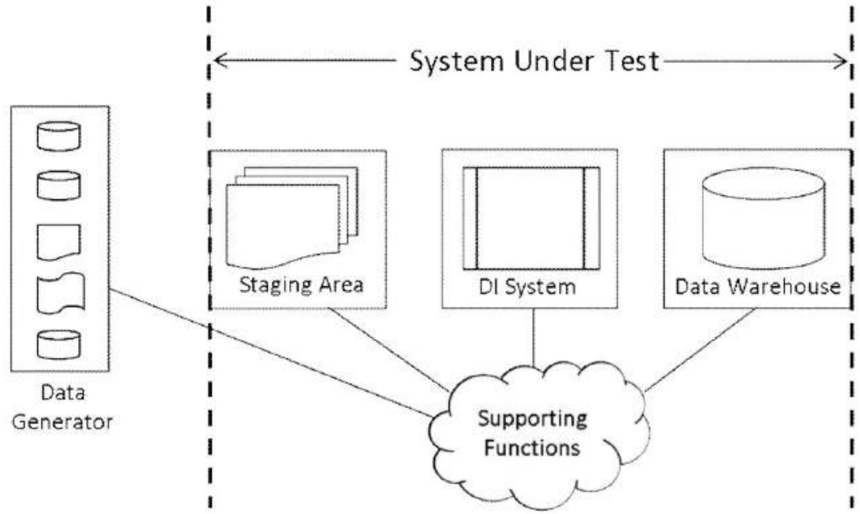
- Loading of large volumes of historical data
- Loading of incremental updates
- Execution of a variety of transformation types using various input types and various target types with inter-table relationships
- Assuring consistency of loaded data

Benchmark is technology agnostic

Why TPC-DI?

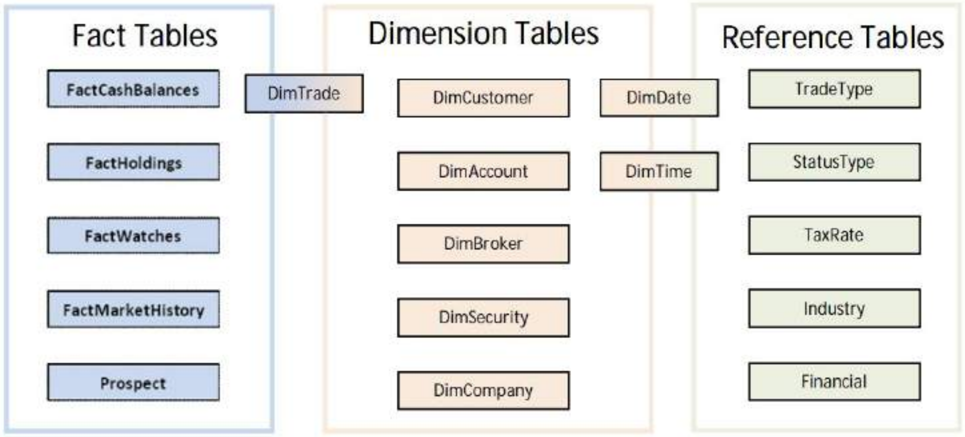
Data Generator

- Produces scales of files from GBs to TB
- Produces CSV, CDC, XML, and Text files
- Has historical and incremental CDC

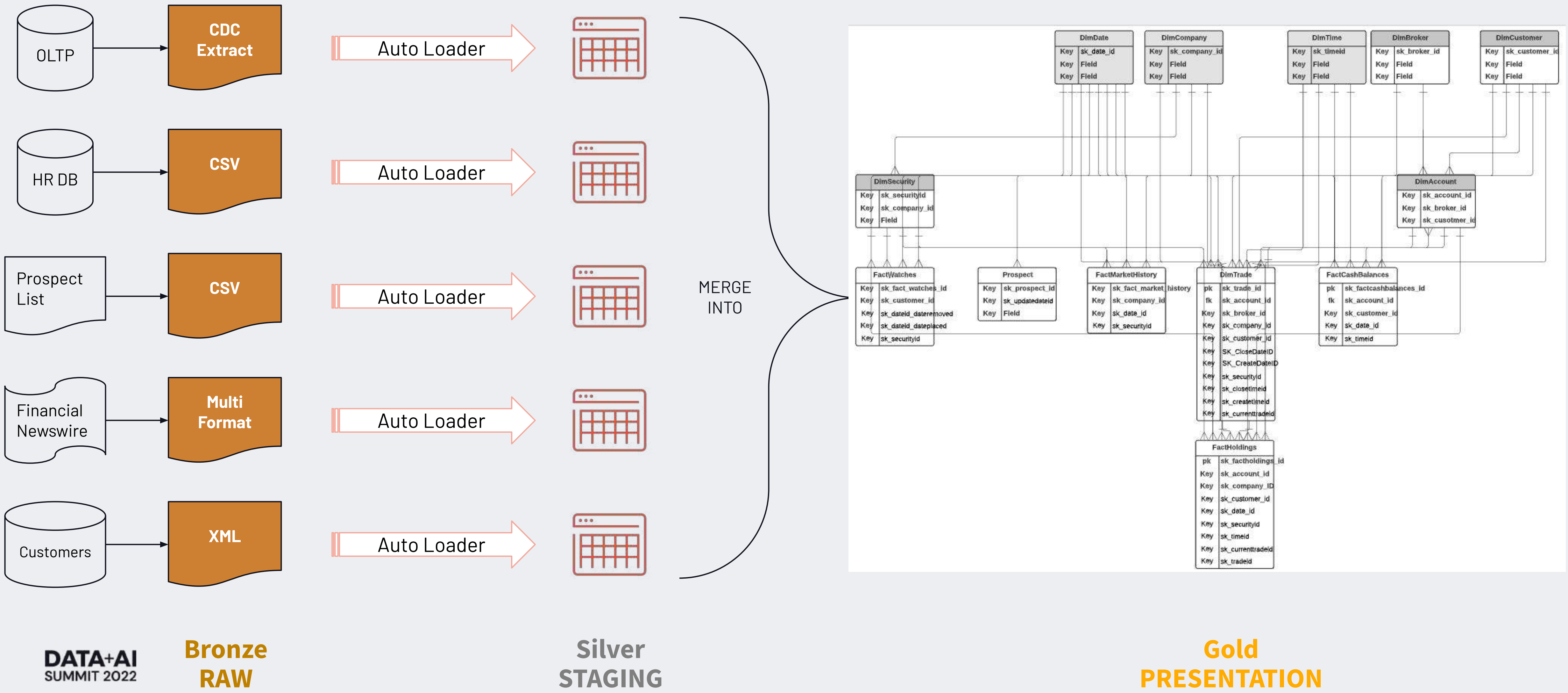


Data Model

- Transformations documented
- Dimensional Model for Analytics
- SCD Type 2
- Window calculations

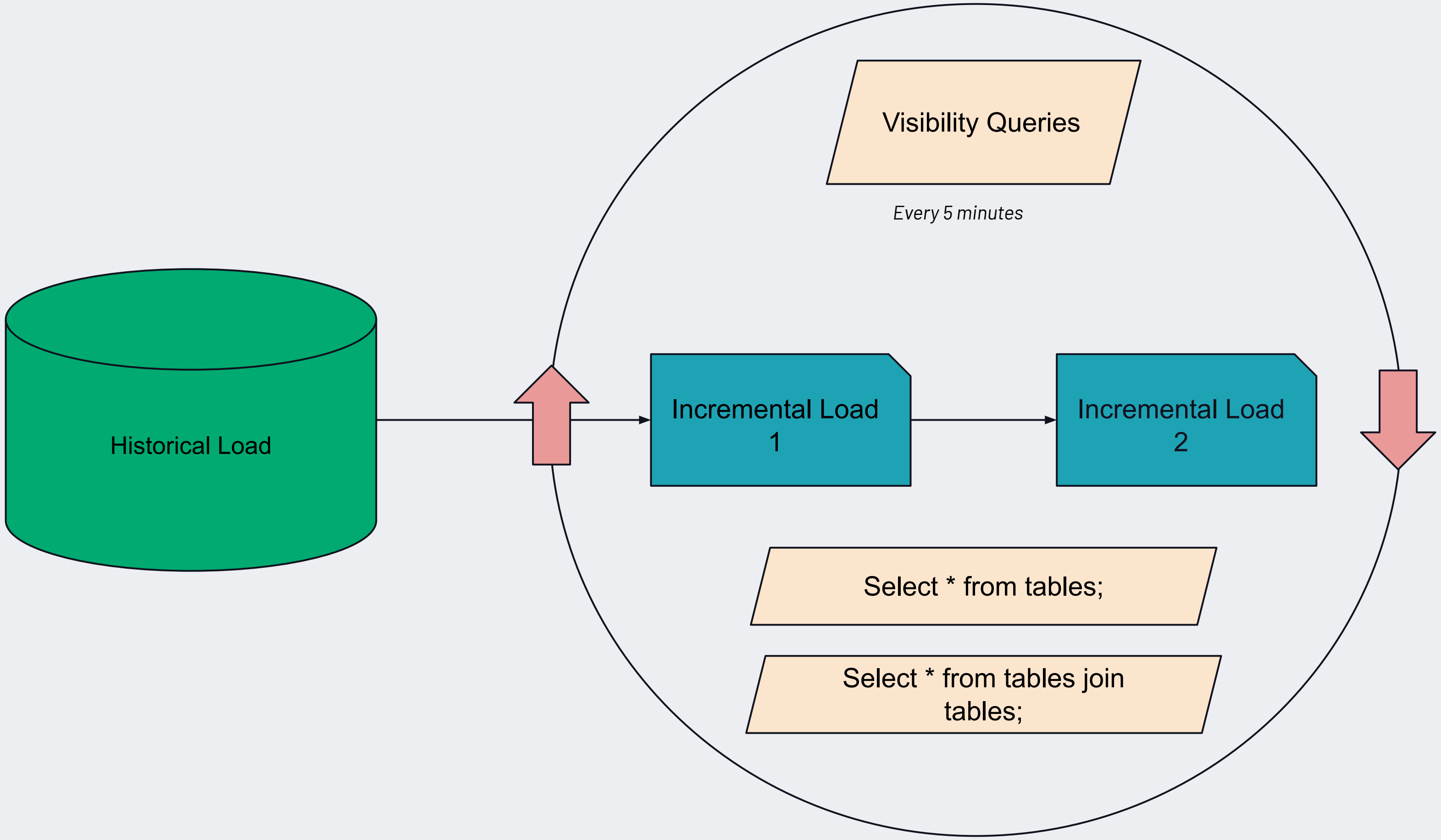


Implementation Reference Architecture



DATA+AI SUMMIT 2022

Concurrency and Consistency



After the historical phase has loaded, during the incremental phases, visibility queries are executed to ensure consistency during loading. Delta handles this with optimistic concurrency with snapshot isolation

Implementing TPC-DI Benchmark on the Lakehouse

Shannon Barrow

Sr. Solutions Architect, Databricks

Context: What is Given vs What We Created

DimCustomer Example

4.5.4.3 When populating fields of the DimCustomer table:

- When ./@ActionType is 'NEW'
- CustomerID, TaxID, LastName, FirstName, MiddleInitial, Tier, DOB, Email1 and Email2 are copied from Customer/@C_ID, Customer/@C_TAX_ID, Customer/Name/C_L_NAME, Customer/Name/C_F_NAME, Customer/Name/C_M_NAME, Customer/@C_TIER, Customer/@C_DOB, Customer/ContactInfo/C_PRIM_EMAIL, Customer/ContactInfo/C_ALT_EMAIL, respectively.
- Gender is obtained from Customer/@C_GNDR, and is uppercased. Values other than 'M' or 'F' are replaced with 'U'.
- AddressLine1, AddressLine2, PostalCode, City, State_Prov, and Country are copied from Customer/Address/C_ADLINE1, Customer/Address/C_ADLINE2, Customer/Address/C_ZIPCODE, Customer/Address/C_CITY, Customer/Address/C_STATE_PROV, and Customer/Address/C_CTRY.
- Status is set to 'ACTIVE'.
- Phone1, Phone2 and Phone3 are created by concatenating fields from the corresponding input data. The input data contains 3 contact phone number elements, Customer/ContactInfo/C_PHONE_1, Customer/ContactInfo/C_PHONE_2, and Customer/ContactInfo/C_PHONE_3, which correspond to Phone1, Phone2, and Phone3 respectively. The transformation for each of the these fields is as follows:
- For each Phone*n*, where *n* = {1,2,3}
- If Customer/ContactInfo/C_PHONE_n/C_CTRY_CODE, Customer/ContactInfo/C_PHONE_n/C_AREA_CODE and Customer/ContactInfo/C_PHONE_n/C_LOCAL are not null, Phone*n* is:
  '+' + Customer/ContactInfo/C_PHONE_n/C_CTRY_CODE
  + ' (' + Customer/ContactInfo/C_PHONE_n/C_AREA_CODE + ') '
  + Customer/ContactInfo/C_PHONE_n/C_LOCAL
- If Customer/ContactInfo/C_PHONE_n/C_CTRY_CODE is null while Customer/ContactInfo/C_PHONE_n/C_AREA_CODE and Customer/ContactInfo/C_PHONE_n/C_LOCAL are not null, Phone*n* is:
  '(' + Customer/ContactInfo/C_PHONE_n/C_AREA + ') '
  + Customer/ContactInfo/C_PHONE_n/C_LOCAL
- If Customer/ContactInfo/C_PHONE_n/C_AREA_CODE is null while Customer/ContactInfo/C_PHONE_n/C_LOCAL is not null, Phone*n* is:
  Customer/ContactInfo/C_PHONE_n/C_LOCAL
- If any of the above rules has been applied and Customer/ContactInfo/C_PHONE_n/C_EXT is not null, Phone*n* is:
  Phone*n* + Customer/ContactInfo/C_PHONE_n/C_EXT
- If none of the above rules has been applied,

XML

- Historical data is read from XML
  - Read only subset from XML since it is shared with DimAccount
  - Each XML record is only a single col update
  - Additional complex logic
  - History tracking (SCD Type 2)

Context: What is Given vs What We Created

DimCustomer Example

4.5.4.3 When populating fields of the DimCustomer table:

- When ./@ActionType is 'NEW'

4.6.4 DimCustomer

4.6.4.1 DimCustomer data is obtained from the data file Customer.txt. The TaxRate, StatusType, and Prospect tables will be referenced in the transformation. C_ID is the natural key for the Customer data. Changes to DimCustomer are implemented in a history-tracking manner.

4.6.4.2 New Customer records in the input data are indicated by CDC_FLAG set to "I". Existing customer records are indicated by CDC_FLAG set to "U".

Note: More than one update to the same Customer may occur during this phase (i.e. on the same day) and should be handled as described in 4.4.1.5.

4.6.4.3 When populating fields of the DimCustomer table:

- CustomerID, TaxID, LastName, FirstName, MiddleInitial, Tier, DOB, Email1 and Email2 are copied from C_ID, C_TAX_ID, C_L_NAME, C_F_NAME, C_M_NAME, C_TIER, C_DOB, C_EMAIL_1, C_EMAIL_2 respectively.
- Gender is obtained from C_GNDR, which is uppercased. Values other than 'M' or 'F' are replaced with 'U'.
- AddressLine1, AddressLine2, PostalCode, City, StateProv, and Country are copied from C_ADLINE1, C_ADLINE2, C_ZIPCODE, C_CITY, C_STATE_PROV, and C_CTRY.
- Status is copied from ST_NAME of the StatusType table by matching C_ST_ID with ST_ID of the StatusType table.
- Phone1, Phone2 and Phone3 are created by concatenating fields. For each *n* in {1, 2, 3}:
- If C_CTRY_*n*, C_AREA_*n* and C_LOCAL_*n* are not null, Phone*n* is:
  '+' + C_CTRY_*n* + ' (' + C_AREA_*n* + ') ' + C_LOCAL_*n*
- If C_CTRY_*n* is null while C_AREA_*n* and C_LOCAL_*n* are not null, Phone*n* is:
  '(' + C_AREA_*n* + ') ' + C_LOCAL_*n*
- If C_AREA_*n* is null while C_LOCAL_*n* is not null, Phone*n* is:
  C_LOCAL_*n*
- If any of the above rules has been applied and C_EXT_*n* is not null, Phone*n* is:
  Phone*n* + C_EXT_*n*
- If none of the above rules has been applied, Phone*n* is null
- NationalTaxRateDesc and NationalTaxRate are copied from TX_NAME and TX_RATE respectively by matching C_NAT_TX_ID with TX_ID.
- LocalTaxRateDesc and LocalTaxRate are copied from TX_NAME and TX_RATE respectively by matching C_LCL_TX_ID with TX_ID.
- AgencyID, CreditRating, NetWorth, MarketingNameplate: If demographic data for this customer has been present in the Prospect file for this DI batch or for any previous batch the latest AgencyID, CreditRating and NetWorth values will be copied to DimCustomer and the MarketingNameplate will be set according to the latest values using the same process defined for the data warehouse Prospect table. A Prospect record is deemed to match a DimCustomer record if the FirstName, LastName, AddressLine1, AddressLine2 and PostalCode fields all match the corresponding fields in DimCustomer when upper-cased. The IsCustomer field in the Prospect table needs to be updated to reflect the

XML

TXT

- Historical data is read from XML
  - Read only subset from XML since it is shared with DimAccount
  - Each XML record is only a single col update
  - Additional complex logic
  - History tracking (SCD Type 2)
- Incremental data is read from TXT
  - Different schema as historical XML
  - History tracking (SCD Type 2) creates complexities with Surrogate Keys and consistency downstream

Context: What is Given vs What We Created

DimCustomer Example

4.5.4.3 When populating fields of the DimCustomer table:

- When ./@ActionType is 'NEW'
- CustomerID, TaxID
  copied from Custo
  Customer/Name/C
  Customer/@C_DO
  Customer/Contact
- Gender is obtained
  or 'F' are replaced
- AddressLine1, Add
  Customer/Address
  Customer/Address
  Customer/Address
- Status is set to 'AC
- Phone1, Phone2 a
  input data. The in
  Customer/Contact
  Customer/Contact
  respectively. The t
- For each Phonen, v
- If Customer/Conta
  Customer/Contact
  Customer/Contact
  '+' + Custor
  + '(' + Cust
  + Custome
- If Customer/Conta
  Customer/Contact
  Customer/Contact
  '(' + Custon
  + Custome
- If Customer/Conta
  Customer/Contact
  Customer/
- If any of the above
  is not null, Phonen
  Phonen + C
- If none of the abov

4.6.4 DimCustomer

4.6.4.1 DimCustomer data is obtained from the data file Customer.txt. The TaxRate, StatusType, and Prospect tables will be referenced in the transformation. C_ID is the natural key for the Customer data. Changes to DimCustomer are implemented in a history-tracking manner

4.6.4.2 New Customer records in the input data are indicated by CDC_FLAG set to "I". Existing customer records are indicated by CDC_FLAG set to "U".

Note: More than one update to the same Customer may occur during this phase (i.e. on the sa day) and should be handled as described in 4.4.1.5.

4.6.4.3 When populating fields of the DimCustomer table:

- CustomerID, TaxID, LastName, FirstName, MiddleInitial, Tier, DOB, Email1 and Email2 copied from C_ID, C_TAX_ID, C_L_NAME, C_F_NAME, C_M_NAME, C_TIER, C_DOB, C_EMAIL_1, C_EMAIL_2 respectively.
- Gender is obtained from C_GNDR, which is uppercased. Values other than 'M' or 'F' replaced with 'U'.
- AddressLine1, AddressLine2, PostalCode, City, StateProv, and Country are copied fro C_ADLINE1, C_ADLINE2, C_ZIPCODE, C_CITY, C_STATE_PROV, and C_CTRY.
- Status is copied from ST_NAME of the StatusType table by matching C_ST_ID with ST of the StatusType table.
- Phone1, Phone2 and Phone3 are created by concatenating fields. For each *n* in {1, 2
- If C_CTRY_*n*, C_AREA_*n* and C_LOCAL_*n* are not null, Phone*n* is:
  '+' + C_CTRY_*n* + ' (' + C_AREA_*n* + ') ' + C_LOCAL_*n*
- If C_CTRY_*n* is null while C_AREA_*n* and C_LOCAL_*n* are not null, Phone*n* is:
  '(' + C_AREA_*n* + ') ' + C_LOCAL_*n*
- If C_AREA_*n* is null while C_LOCAL_*n* is not null, Phone*n* is:
  C_LOCAL_*n*
- If any of the above rules has been applied and C_EXT_*n* is not null, Phone*n* is:
  Phone*n* + C_EXT_*n*
- If none of the above rules has been applied, Phone*n* is null
- NationalTaxRateDesc and NationalTaxRate are copied from TX_NAME and TX_RATE respectively by matching C_NAT_TX_ID with TX_ID.
- LocalTaxRateDesc and LocalTaxRate are copied from TX_NAME and TX_RATE respec by matching C_LCL_TX_ID with TX_ID.
- AgencyID, CreditRating, NetWorth, MarketingNameplate: If demographic data for t customer has been present in the Prospect file for this DI batch or for any previous the latest AgencyID, CreditRating and NetWorth values will be copied to DimCustom and the MarketingNameplate will be set according to the latest values using the sar process defined for the data warehouse Prospect table. A Prospect record is deeme match a DimCustomer record if the FirstName, LastName, AddressLine1, AddressLin and PostalCode fields all match the corresponding fields in DimCustomer when upp cased. The IsCustomer field in the Prospect table needs to be updated to reflect th

```sql
decode(_ActionType,
  'NEW', 'Active',
  'UPDCUST', 'Active',
  'INACT', 'Inactive') as status,
C_L_NAME lastname,
C_F_NAME firstname,
C_M_NAME middleinitial,
CASE
    WHEN C_GNDR IN ('M', 'F') THEN
    ELSE 'U'
  END as gender,
C_TIER tier,
C_DOB dob,
C_ADLINE1 addressline1,
C_ADLINE2 addressline2,
C_ZIPCODE postalcode,
C_CITY city,
C_STATE_PROV stateprov,
C_CTRY country,
CASE
  WHEN isnull(c_local_1) then c_loc
  ELSE concat(
    nvl2(c_ctry_1, '+' || c_ctry_1
    nvl2(c_area_1, '(' || c_area_1
    c_local_1,
    nvl(c_ext_1, '')) END as phone1
CASE
  WHEN isnull(c_local_2) then c_local_2
  ELSE concat(
    nvl2(c_ctry_2, '+' || c_ctry_2 || ' ', ''),
    nvl2(c_area_2, '(' || c_area_2 || ') ', ''),
    c_local_2,
    nvl(c_ext_2, '')) END as phone2,
CASE
  WHEN isnull(c_local_3) then c_local_3
  ELSE concat(
    nvl2(c_ctry_3, '+' || c_ctry_3 || ' ', ''),
    nvl2(c_area_3, '(' || c_area_3 || ') ', ''),
    c_local_3,
    nvl(c_ext_3, '')) END as phone3,
```

```sql
SELECT
  customerid,
  _ActionTS,
  coalesce(taxid, last_value(taxid) IGNORE NULLS OVER (
      PARTITION BY customerid
      ORDER BY _ActionTS)) taxid,
  status,
  coalesce(lastname, last_value(lastname) IGNORE NULLS OVER (
      PARTITION BY customerid
      ORDER BY _ActionTS)) lastname,
  coalesce(firstname, last_value(firstname) IGNORE NULLS OVER (
      PARTITION BY customerid
      ORDER BY _ActionTS)) firstname,
  coalesce(middleinitial, last_value(middleinitial) IGNORE NULLS OVER (
      PARTITION BY customerid
      ORDER BY _ActionTS)) middleinitial,
  coalesce(gender, last_value(gender) IGNORE NULLS OVER (
      PARTITION BY customerid
      ORDER BY _ActionTS)) gender,
  coalesce(tier, last_value(tier) IGNORE NULLS OVER (
      PARTITION BY customerid
      ORDER BY _ActionTS)) tier,
  coalesce(dob, last_value(dob) IGNORE NULLS OVER (
      PARTITION BY customerid
      ORDER BY _ActionTS)) dob,
  coalesce(addressline1, last_value(addressline1) IGNORE NULLS OVER (
      PARTITION BY customerid
      ORDER BY _ActionTS)) addressline1,
```

XML

TXT

Delta Lake

The Foundation That Makes it Possible

Surrogate Keys and History Tracking

- Automatically Generated Identity Columns meant Surrogate Keys are created and managed under the hood
- Performance improvements to table Merges, including Low Shuffle Merge, helped enable the History tracking (SCD Type 2) and SCD Type 0 merges

```
CREATE OR REPLACE TABLE DimCustomer
  (${DimCustomerSchema}) USING DELTA
  sk_customerid BIGINT GENERATED ALWAYS AS IDENTITY
```

Additional knobs to Improve Performance

- Generated Columns that were used as Partitions kept data indexed for large tables without time overhead of zorder
- Writes and target files sizes:
  - Optimized writes
  - *delta.tuneFileSizesForRewrites* for Incremental tables

```
CREATE OR REPLACE TABLE FactWatches (${FactWatchesSchema}) USING DELTA
  sk_customerid BIGINT COMMENT 'Customer associated with watch list',
  sk_securityid BIGINT COMMENT 'Security listed on watch list',
  sk_dateid_dateplaced BIGINT COMMENT 'Date the watch list item was added',
  sk_dateid_dateremoved BIGINT COMMENT 'Date the watch list item was removed',
  batchid INT COMMENT 'Batch ID when this record was inserted',
  _removed BOOLEAN GENERATED ALWAYS AS (isnull(sk_dateid_dateremoved))
PARTITIONED BY (_removed)
TBLPROPERTIES (delta.autoOptimize.optimizeWrite = true);
```

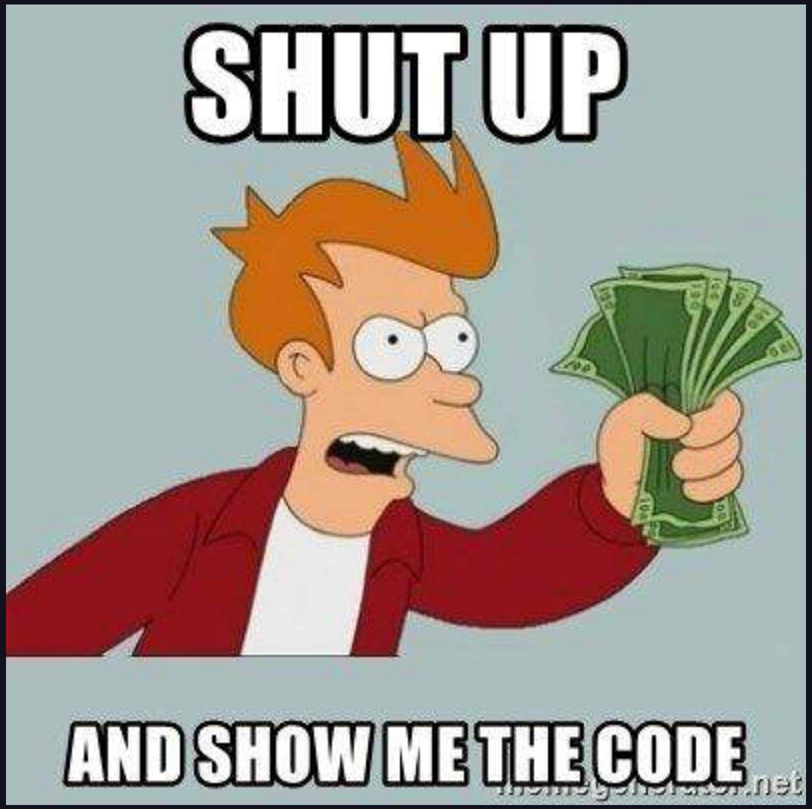
Simplified Orchestration and Automation

Databricks Workflows: Orchestrate anything, anywhere

Run diverse workloads for the full data and AI lifecycle, on any cloud

In addition to the latest in Workflows we leveraged:

- Cluster Reuse – 1 single automated cluster, reused for all tasks
- Repos – Git integration allowed disparate teams to focus on various parts of the code base and consume from arbitrary files using a relative path
- Scripted workflow with Jinja – a fast, expressive, extensible templating engine.


SHUT UP
AND SHOW ME THE CODE

Traditional Notebook Workflow Results

Performance Dashboard

- The TPC-DI has a rather confusing benchmark algorithm
- Simplified: TCO approach based on cost per row processed

These were the best performing combinations with On-Demand Pricing:

Run Time (minutes)	Worker	Total Costs	Price per Billion Rows	Photon	Graviton
36.4	m6gd.8xlarge	$23.28	$1.44	No	Yes
24.0	m6gd.4xlarge	$24.47	$1.51	Yes	Yes

- **SPOT** instances drops this price to as low as **85 CENTS!**

What did we learn?

What is valuable for you to take away from our benchmarks?

Photon

- Photon consistently **>30% faster**, even for this non-optimal workload
- TCO nearly equal (5-10% higher)
- Leads to more productivity for approximately same total cost

Graviton (AWS)

- These are **ARM**-based instances instead of **x86**, currently only served on AWS
- Cheaper instances means **40% less TCO** than x86 instances

Cluster Sizing

- Core counts being equal: Opt for node count over size (16 was the sweet spot)
- TCO dropped at each sizing level:
  - 96<64<48<32<16
- This was tested on Scale Factor 10K w/ 576 cores
- But why?...

Worker Optimization

- High Scale Factor:
  - Very few "big" files
  - Thousands of medium size files (~128MB raw)
- Latest Gen General Purp. tested best
- No need for storage-optimized
- Higher core count was more important than extra memory

What were the obstacles?

What could make your lives easier building similar pipelines?

Fixing Audit Issues

- High Level of effort to resolve Automated Audit test issues
- Obscure business rules buried in documentation meant careful reading
- Had to "back in" to passing results by interpreting the expected results and altering logic to match that expected result

Orchestration Complexity

- While the novel orchestration mechanism delivers a fully scripting pipeline via a single driver, it is possibly as many lines of code as the rest of the code base combined
- Engineers have to update the JSON with all new code added to the pipeline, adding extra complexity

Data Quality Issues

- Discovered DQ issues in the raw files generated by the datagen JAR, only after dozens of hours debugging code to satisfy the automated audits
- Wasted effort sifting through code with a fine tooth comb only to realize it was a DQ issue

How to Be Fresh and Clean

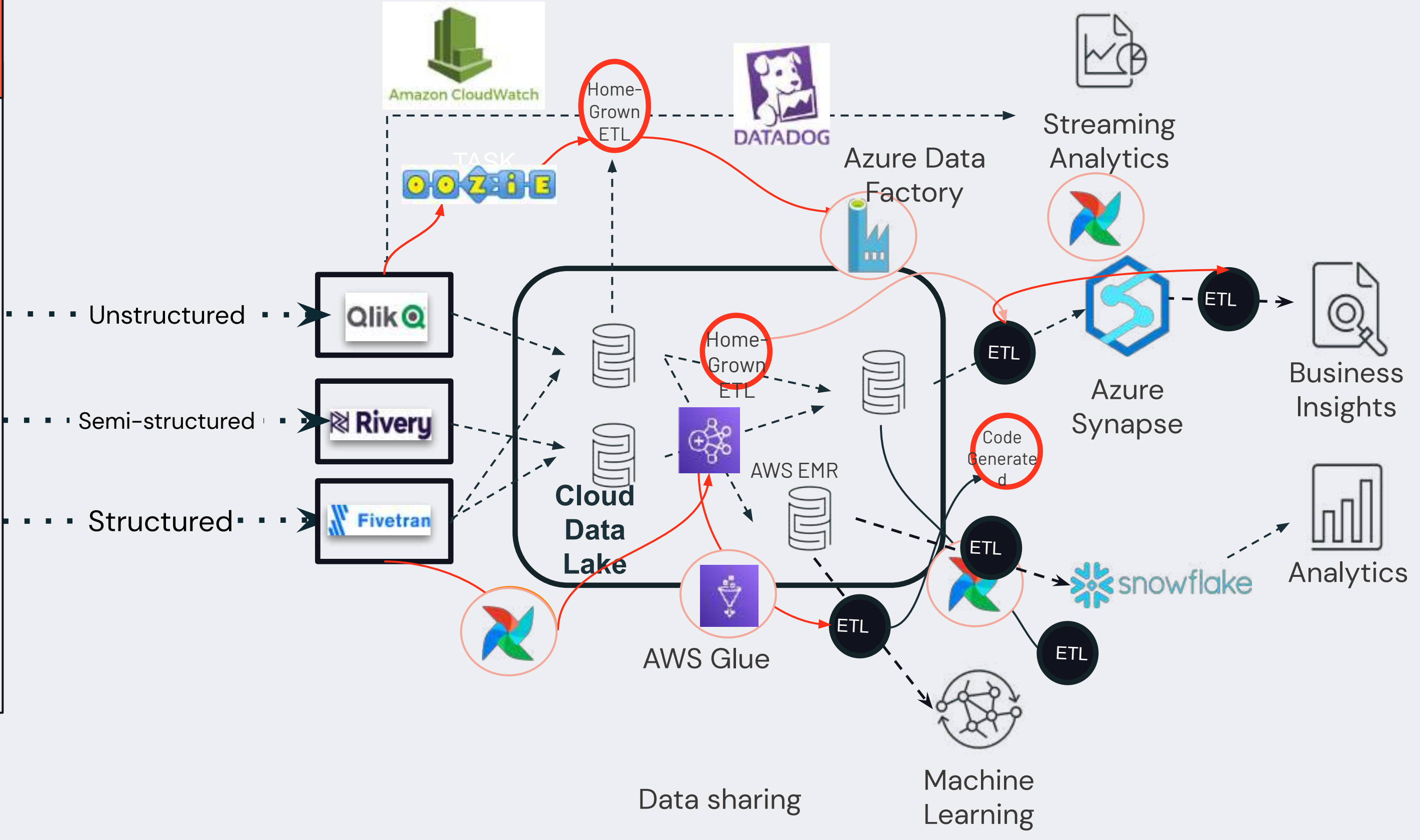
Sr. Solutions Architect, Databricks

So FRESH (AND|OR) So CLEAN:

Data Engineering Is About Tradeoffs

Data Sources

- Streaming Sources
- Cloud Object Stores
- SaaS Applications
- NoSQL
- Relational Databases
- On-premises systems



So FRESH (AND|OR) So CLEAN:

Data Engineering Is About Tradeoffs

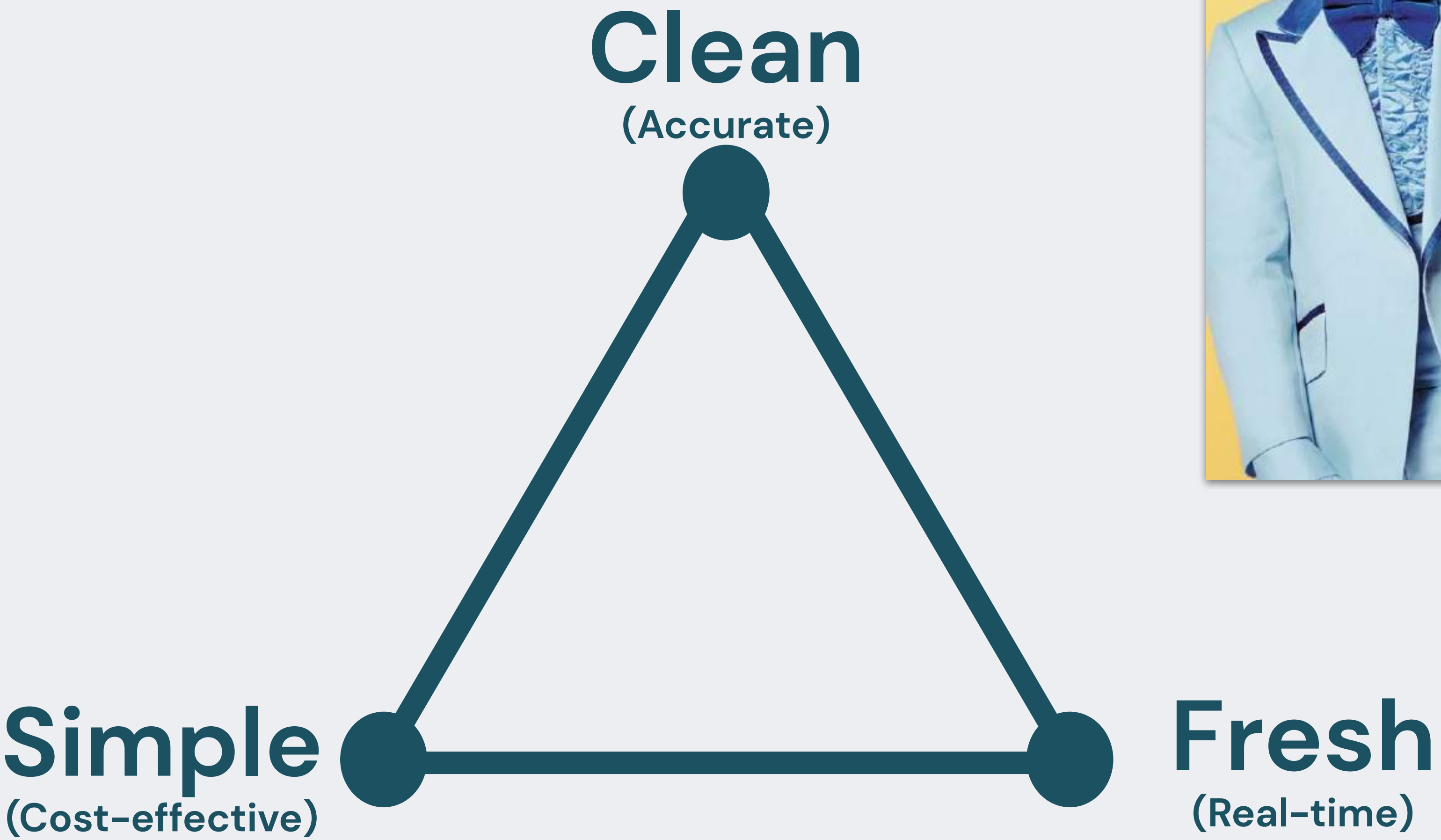
Fresh: Data reflects the current business state in time for actionable insight
Measured by: *pipeline latency, refresh frequency, SLA %*

Clean: Data is trusted by its consumers to accurately describe the business state
Measured by: *cost of wrong decision, time spent curating*

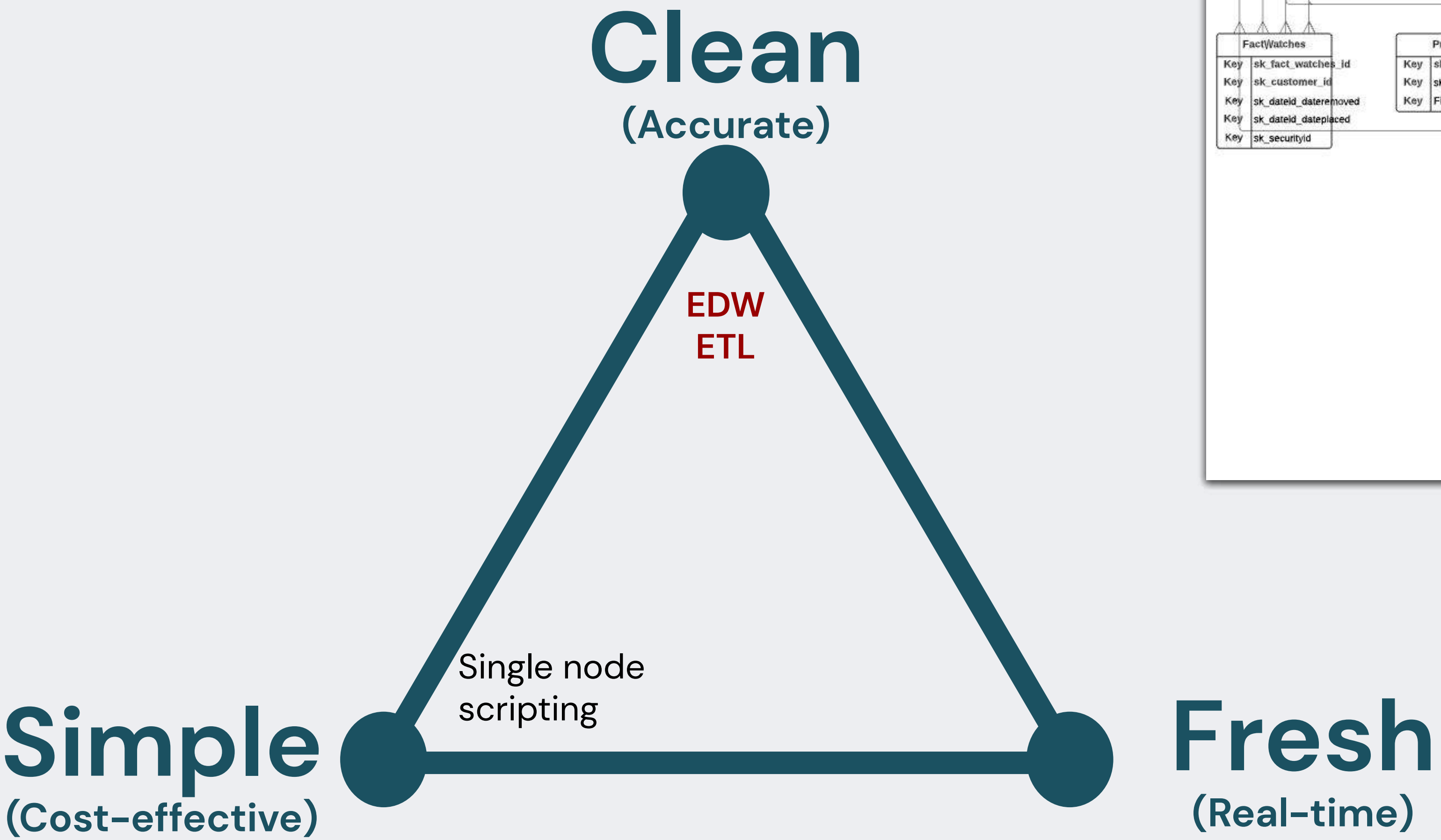
Simple: Data is easily available to consumers at predictable and effective cost
Measured by: *time to insight, cost, MTBF, maintenance time*

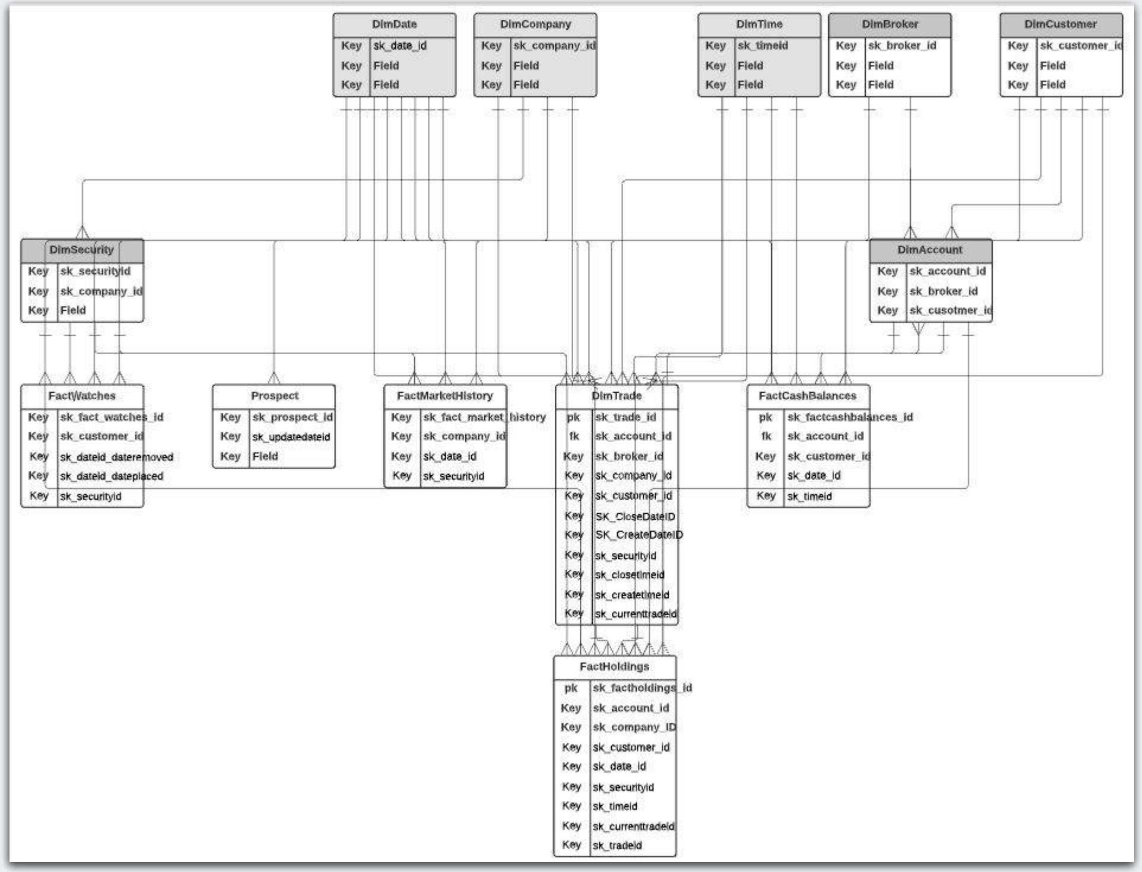
For more info on these definitions and how to get started with DLT: **https://tinyurl.com/freshandcleandais**

The Fresh and Clean Trilemma

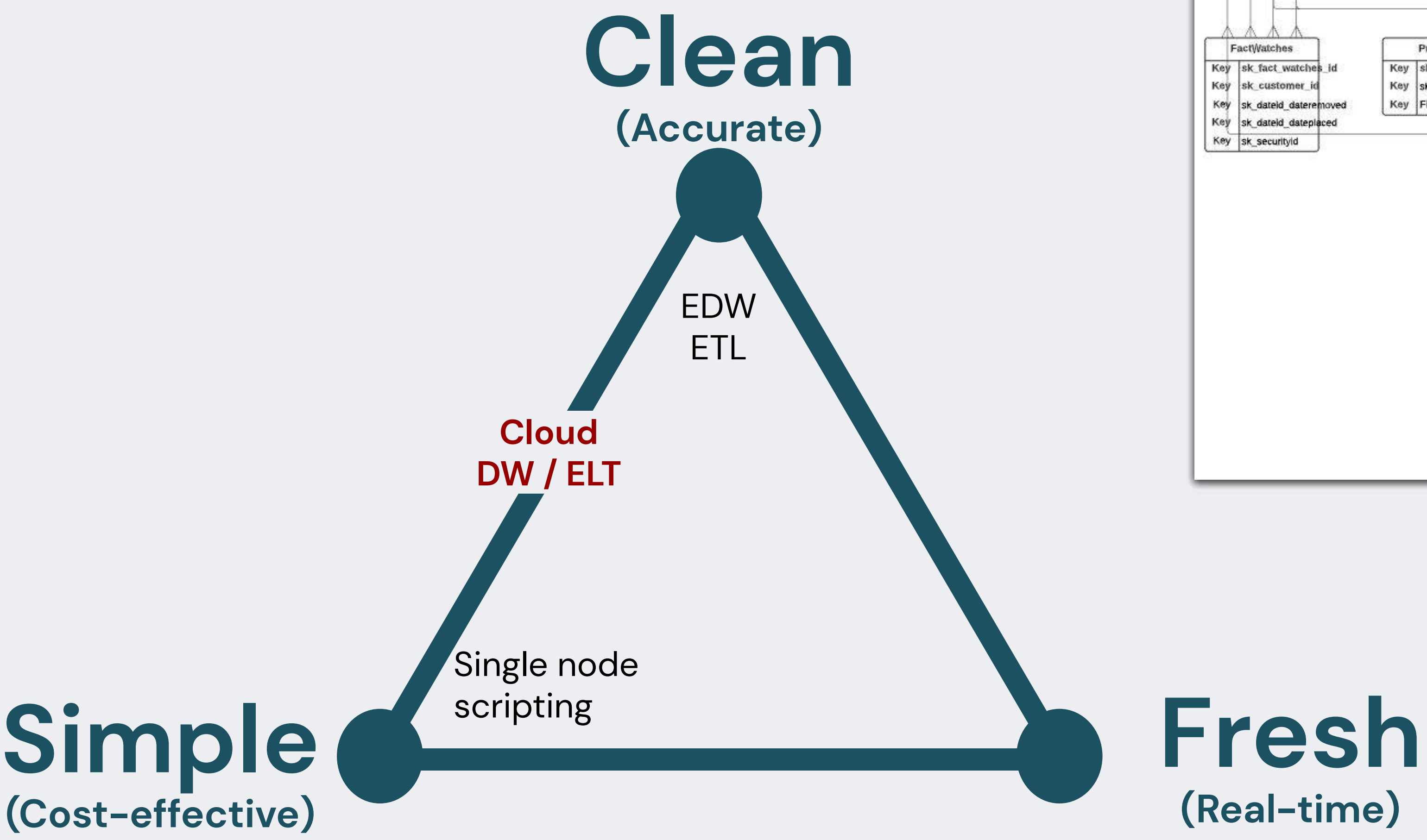


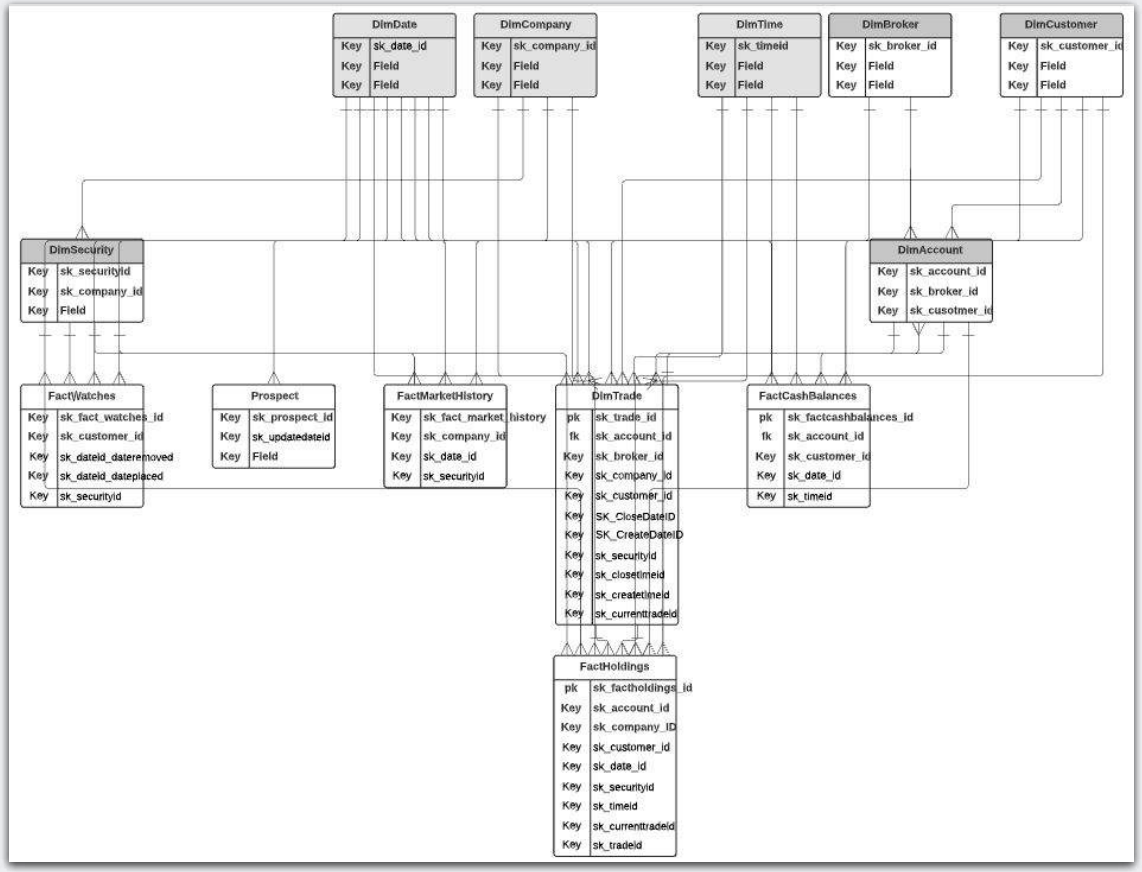
The Fresh and Clean Trilemma



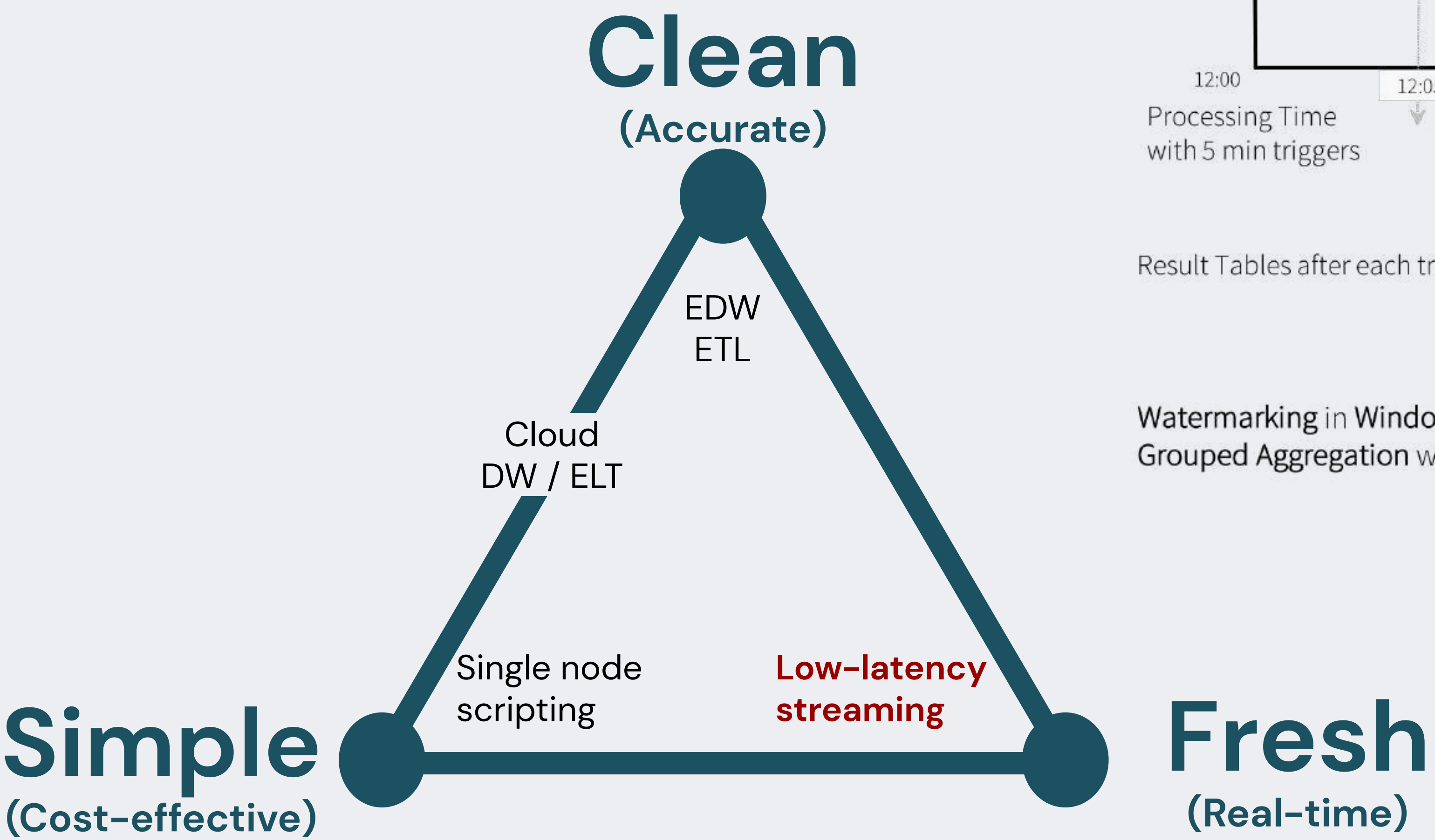


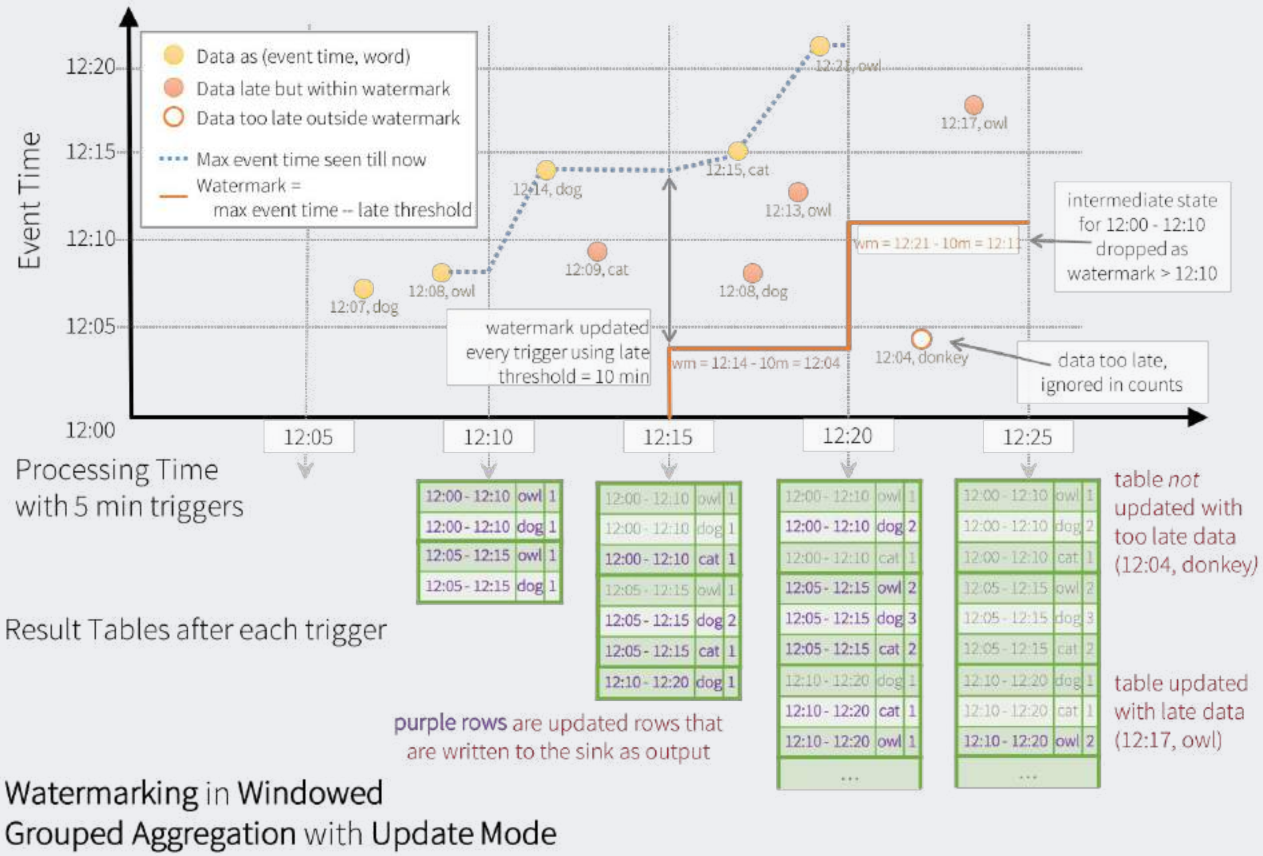
The Fresh and Clean Trilemma



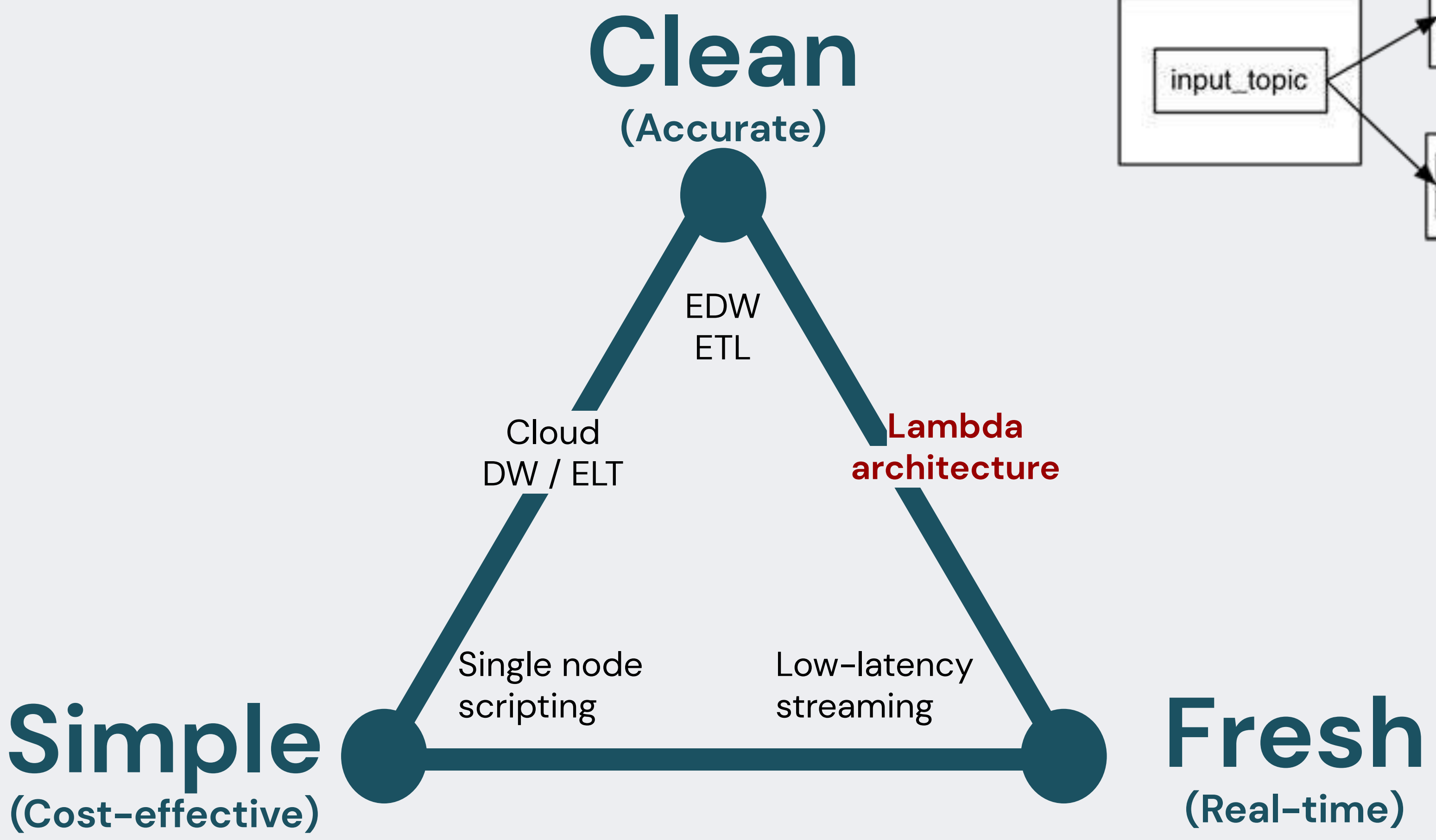


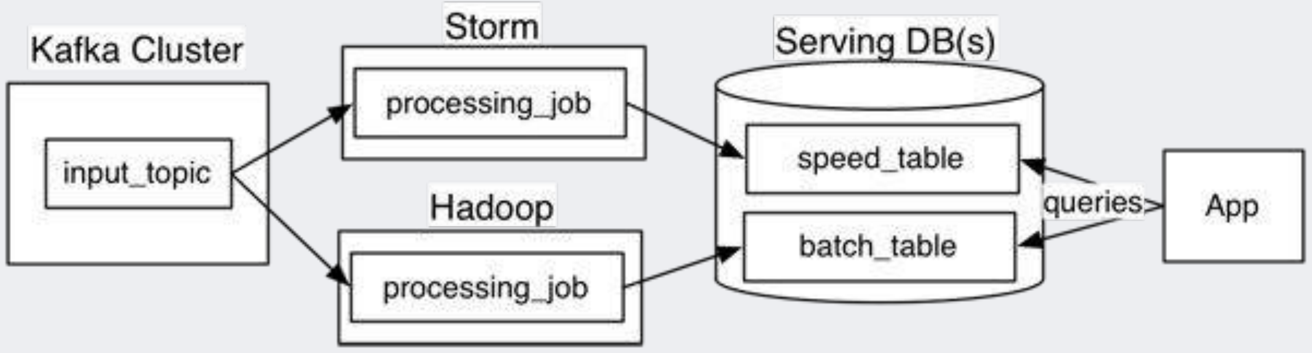
The Fresh and Clean Trilemma



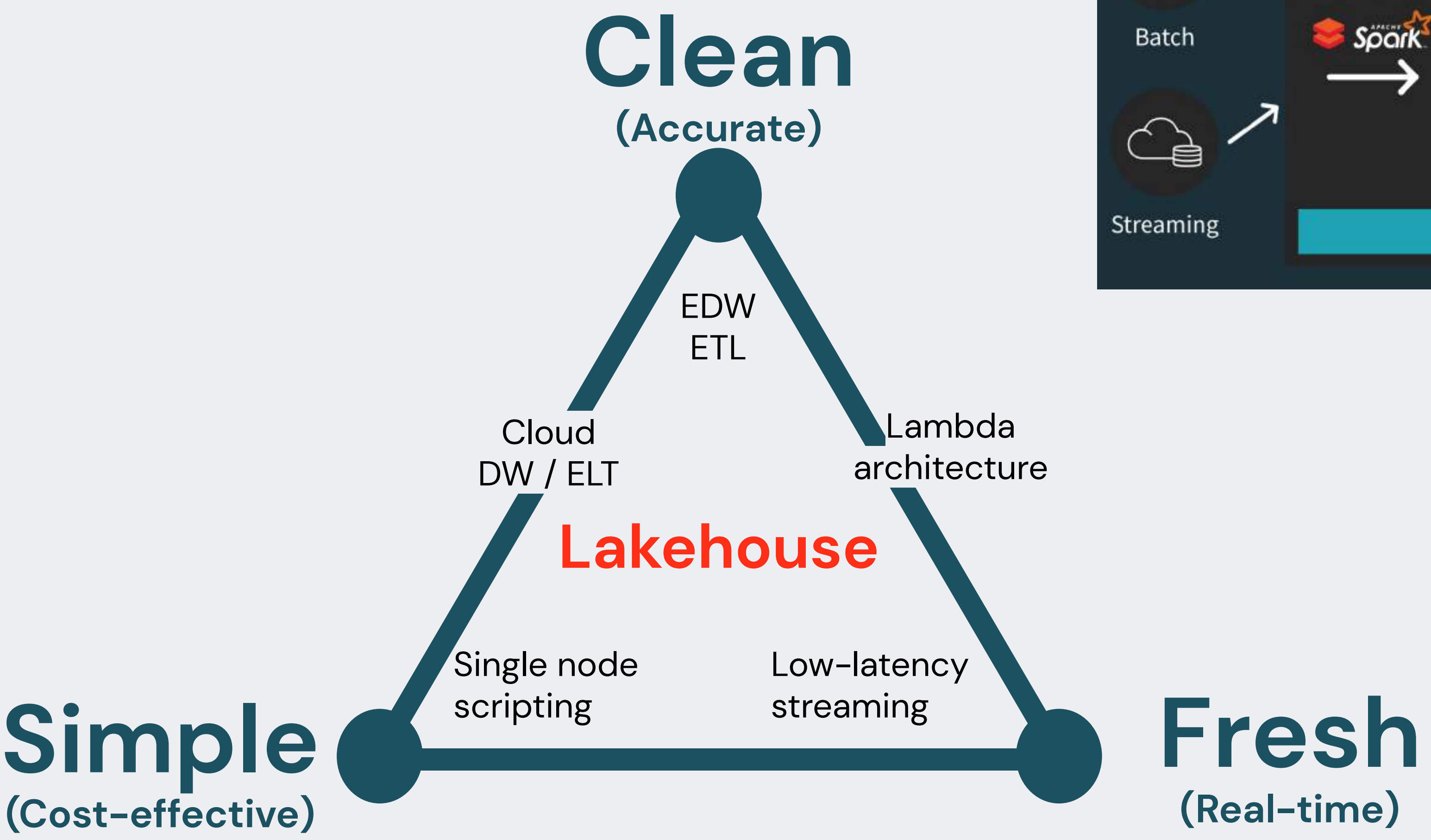


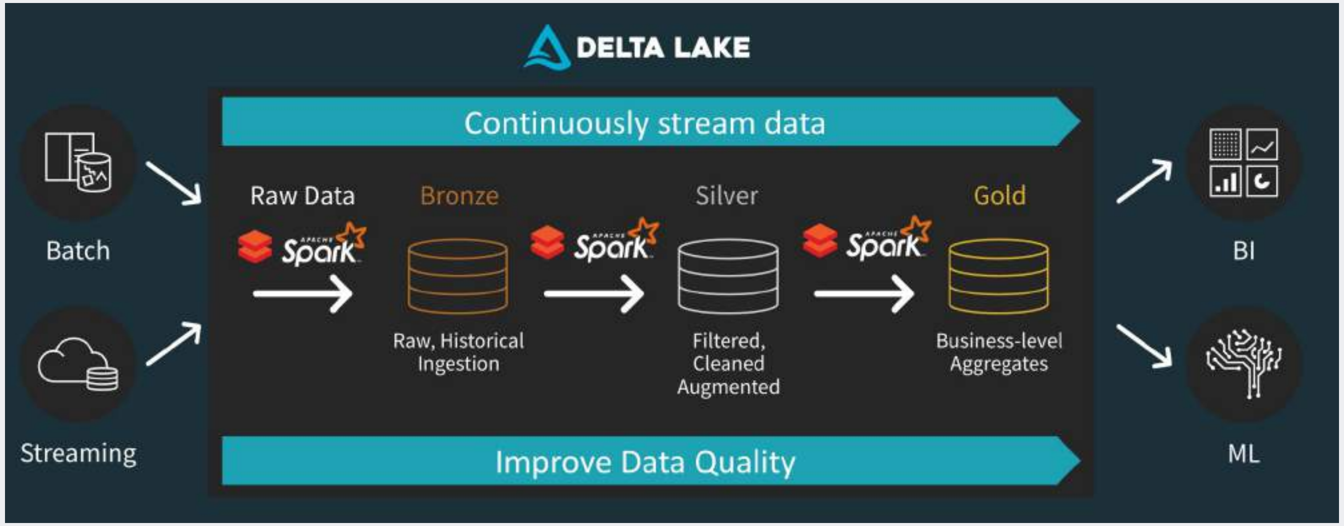
The Fresh and Clean Trilemma



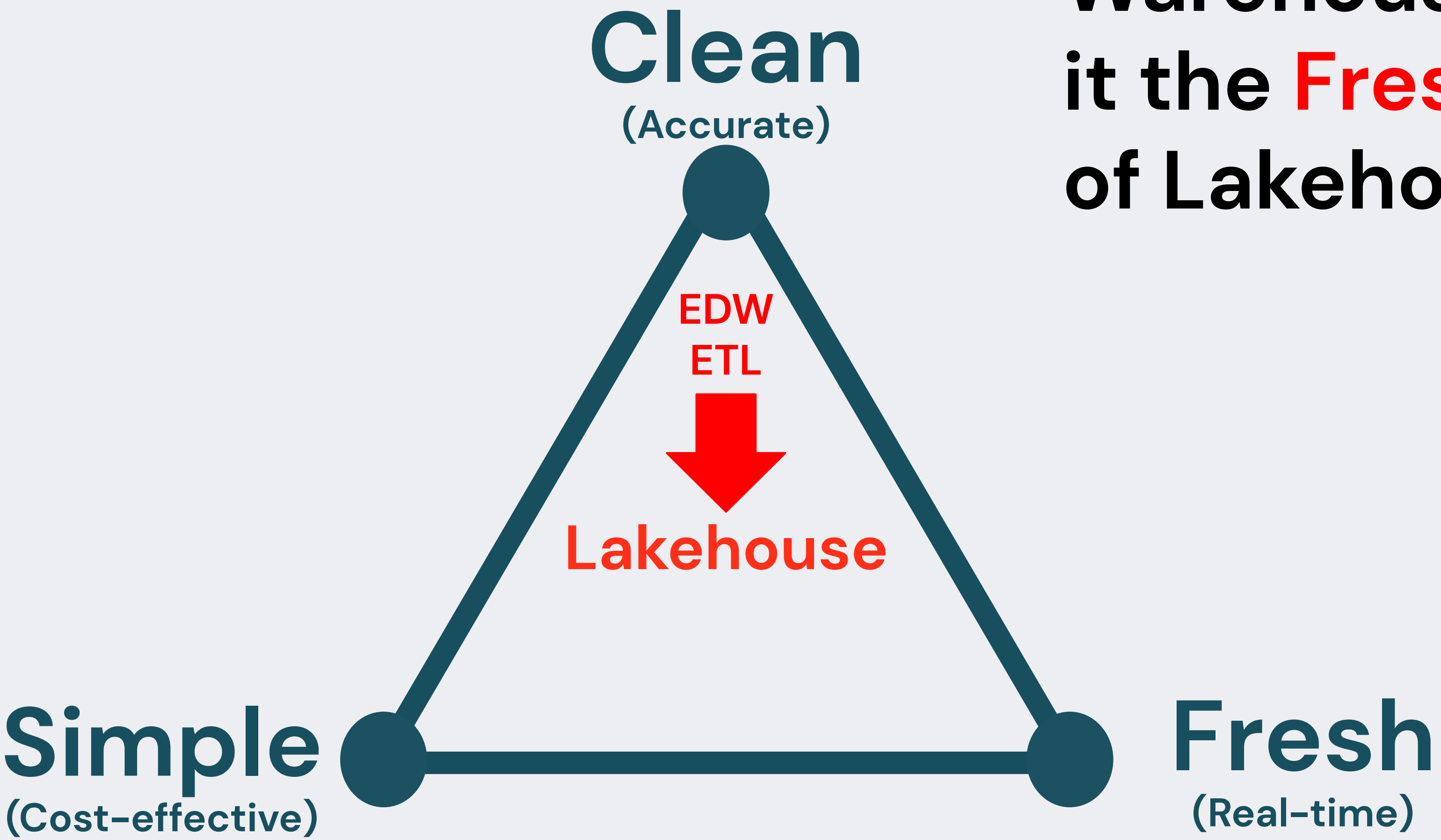


The Fresh and Clean Trilemma





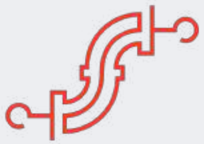
Can we take a traditional Data Warehousing pipeline and give it the Freshness and Simplicity of Lakehouse?



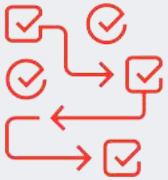
What is Delta Live Tables?

Modern software engineering for ETL processing

Delta Live Tables (DLT) is the first ETL framework that uses a simple, declarative approach to building reliable data pipelines. DLT automatically manages your infrastructure at scale so data analysts and engineers can spend less time on tooling and focus on getting value from data.



Accelerate ETL Development



Automatically manage your infrastructure

Have confidence in your data

Simplify batch and streaming

Declarative ETL Pipelines with DLT

Source

```
/* Create a temp view on the accounts table */
CREATE STREAMING LIVE VIEW account_raw AS
SELECT * FROM cloud_files(“/data”, “csv”);
```

Bronze

```
/* Stage 1: Bronze Table drop invalid rows */
CREATE STREAMING LIVE TABLE account_bronze AS
COMMENT "Bronze table with valid account ids"
SELECT * FROM fire_account_raw ...
```

Silver

```
/* Stage 2:Send rows to Silver, run validation rules */
CREATE BATCH LIVE TABLE account_silver AS
COMMENT "Silver Accounts table with validation checks"
SELECT * FROM fire_account_bronze ...
```

Gold

Declaratively build data pipelines with business logic and chain table dependencies

Run in batch or streaming with structured or unstructured data

Reuse ETL pipelines across environments

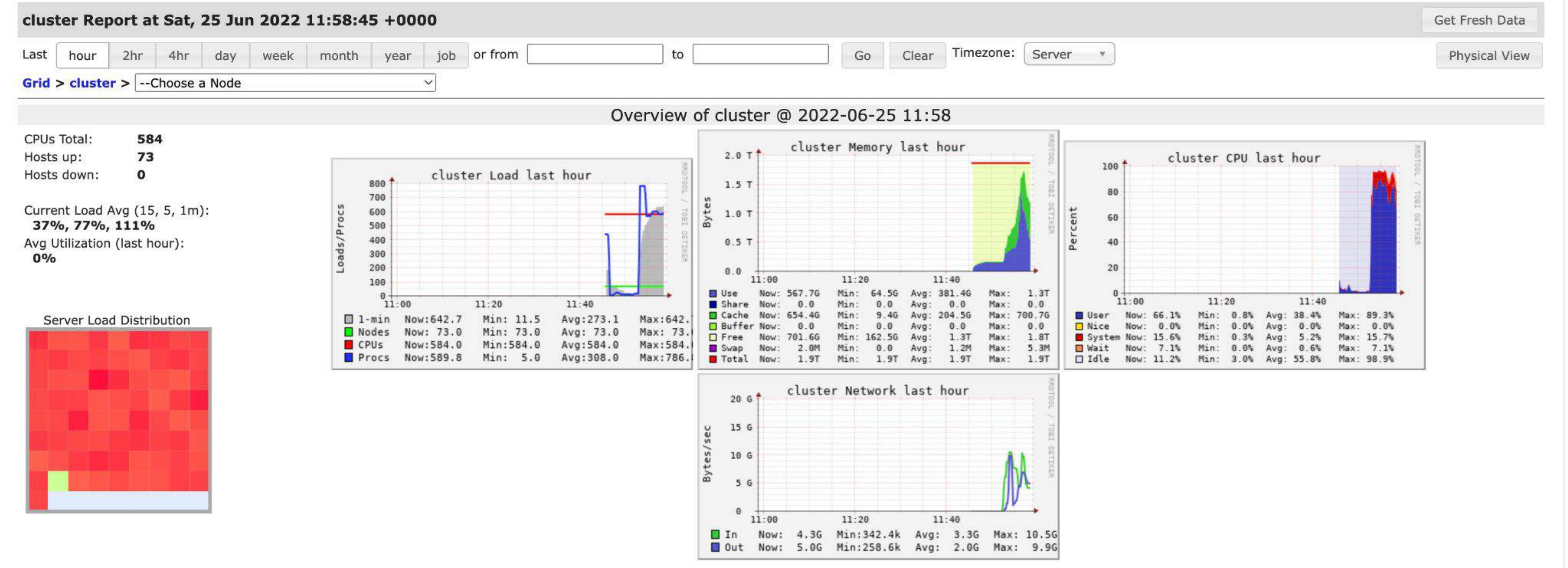
Modern data engineering & ETL on the Lakehouse

Load and transform at any scale with high quality data pipelines

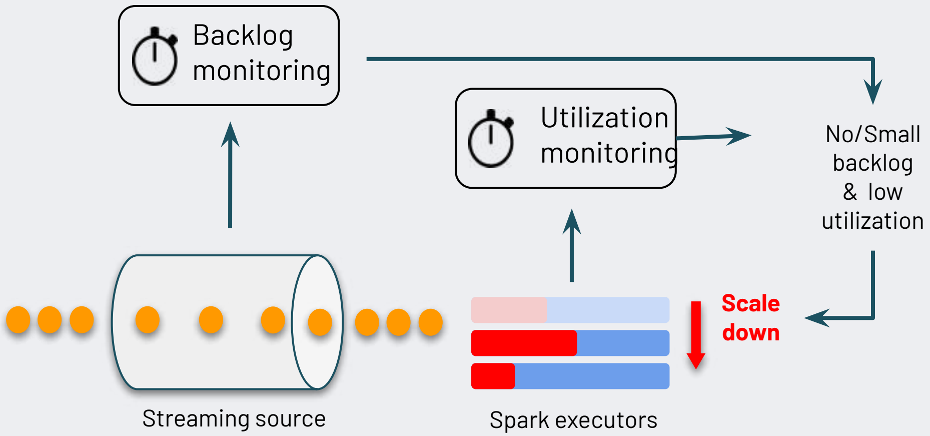
- Easily build and orchestrate pipelines with native observability, lineage, and quality checks
- Quickly ingest business critical data in batch or streaming
- Empower analytics engineers with dbt integration and full ANSI SQL support for SQL-based ETL

TPC-DI Cluster Utilization on DLT

Ganglia Snapshot for 72 md5.2xl (16 core)



Automated scaling and fault tolerance with Delta Live Tables



- Meet streaming SLAs with backlog-aware scaling decisions - Monitor both, **backlog metrics and cluster utilization** to scale up or down
- **Reduce down time** with automatic error handling and easy replay
- **Eliminate maintenance** with automatic optimizations of all Delta Live Tables
- Execute data pipeline workload on **automatically provisioned** elastic Apache Spark™-based compute clusters that parallelize jobs as well as minimize data movement

Trust your data

Prevent bad data from flowing into tables with Delta Expectations

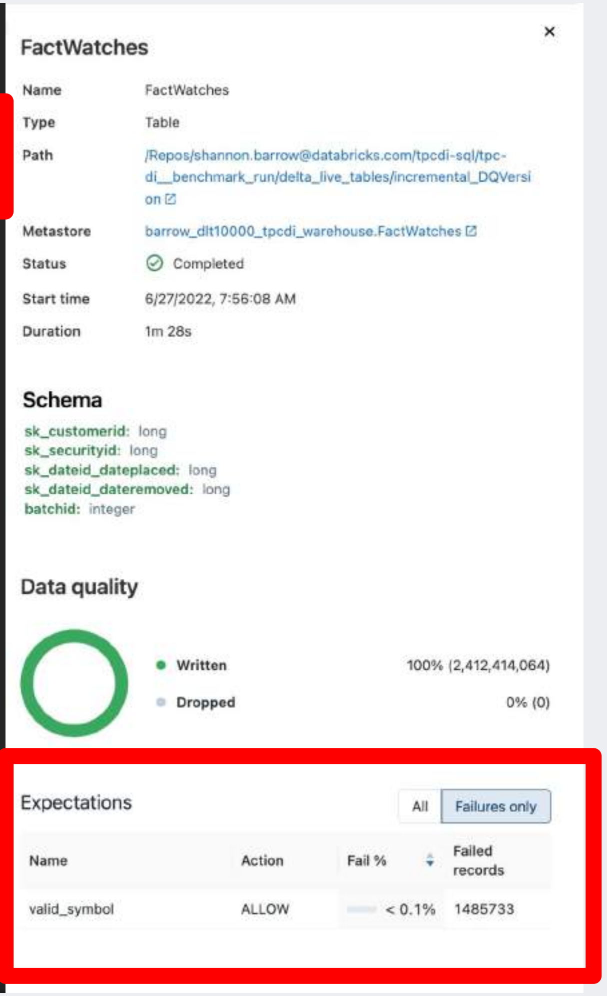
Avoid and address quality errors with pre-defined error policies (fail, drop, alert or quarantine data)

Monitor data quality trends over time

```
/* Stage 1: Bronze Table drop invalid rows */
CREATE INCREMENTAL LIVE TABLE fire_account_bronze AS
( CONSTRAINT valid_account_open_dt EXPECT (account_dt is not null AND
(account_close_dt > account_open_dt)) ON VIOLATION DROP ROW
COMMENT "Bronze table with valid account ids"
SELECT * FROM fire_account_raw ...
```

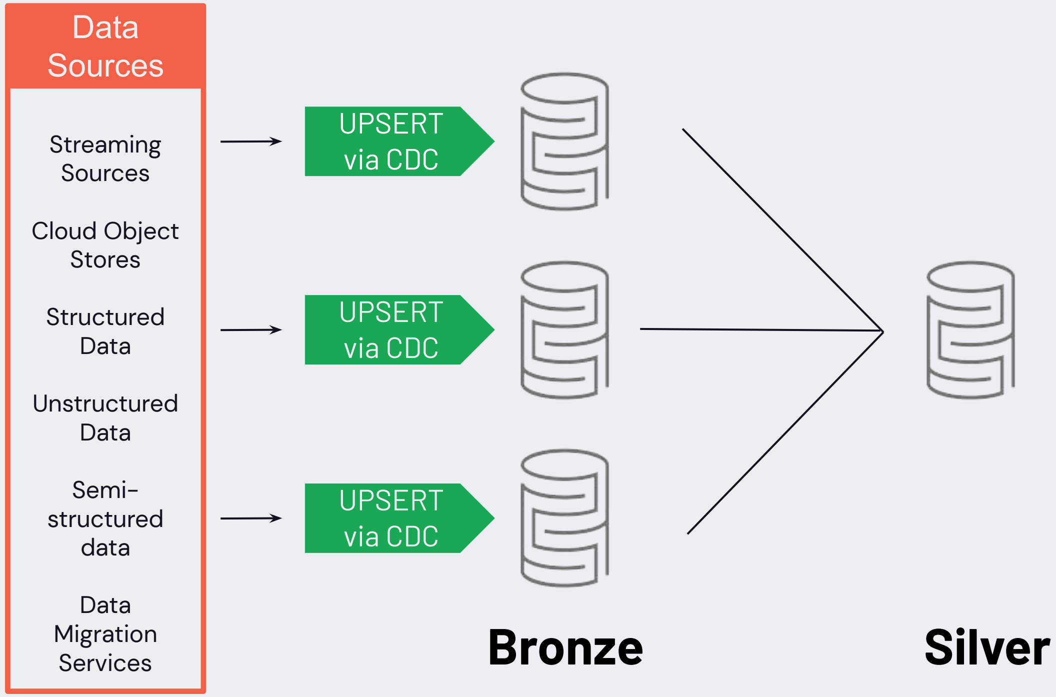
DQ Notes: FactWatches Example

```sql
CREATE OR REFRESH LIVE TABLE FactWatches (
  ${factwatchesschema}
  CONSTRAINT valid_symbol EXPECT (sk_securityid IS NOT NULL),
  CONSTRAINT valid_customer_id EXPECT (sk_customerid IS NOT NULL))
AS SELECT
  c.sk_customerid sk_customerid,
  s.sk_securityid sk_securityid,
  sk_dateid_dateplaced,
  sk_dateid_dateremoved,
  fw.batchid
FROM LIVE.FactWatchesTemp fw
LEFT JOIN LIVE.DimSecurity s
  ON
    s.symbol = fw.symbol
    AND fw.dateplaced >= s.effectivedate
    AND fw.dateplaced < s.enddate
LEFT JOIN LIVE.DimCustomer c
  ON
    fw.customerid = c.customerid
    AND fw.dateplaced >= c.effectivedate
    AND fw.dateplaced < c.enddate
```



[DQ Dashboard]

Change Data Capture (CDC) with Delta Live Tables




```
APPLY CHANGES INTO LIVE.cities
FROM STREAM(LIVE.city_updates)
KEYS (id)
APPLY AS DELETE WHEN update="_DEL"
SEQUENCE BY timestamp
STORED AS SCD TYPE 2
```

city_updates

```
{"id": 1, "ts": 1, "city": "Bekerly, CA"}
{"id": 1, "ts": 2, "city": "Berkeley, CA"}
```

cities

city	__starts_at	__ends_at
Bekerly, CA	1	2

DEMO:
Delta Live Tables TPC-DI Pipeline

Shannon Barrow
Sr. Solutions Architect, Databricks

DLT Results

Performance Dashboard

Revisiting best performing TCO combinations vs Traditional Notebooks

- Caveats:
  - DLT was not developed to be submitted for benchmark, therefore does not do audit checks between historical -> incremental (batch-approach is not conducive to DLT)
  - DLT does not use Scala, meaning the XML library couldn't be loaded - so it is run as first step in 2-stage Workflow. To account for this add ~3 minutes to times for DLT
- DLT optimizes pipeline better because of more granular orchestration (table-level vs notebook level) - leads to better cluster utilization!

Run Time (minutes)	Worker	Total Costs	Price per Billion Rows	Photon	Graviton	Traditional or DLT
17.1	m5d.4xlarge	$15.10	$0.93	No	No	DLT
10.7	m5d.2xlarge	$16.25	$1.01	Yes	No	DLT
36.4	m6gd.8xlarge	$23.28	$1.44	No	Yes	Traditional
24.0	m6gd.4xlarge	$24.47	$1.51	Yes	Yes	Traditional

- SPOT instances drops this price to as low as 58 CENTS!

Why pay up to 3x or more for just warehousing, when you can build a data platform that has **ETL Orchestration and Data Quality** with *Delta Live Tables*, **Machine Learning and AI** built-in with **Auto ML**, and **SQL Warehouse serving**, all on one copy of your data in **Delta Lake**?

Franco Patano

Lead Product Specialist, Databricks

Partners

No coding required with Prophecy.io!

www.prophecy.io/prophecy-for-databricks

Special Thanks!

Alex Desroches,
Brad Barker,
David Radford,
Itai Weiss,
Joe Harris,
Joe Russell,
Lorin Dawson,
Max Nienu,
Nico Poggi

Q&A ?

DATA+AI SUMMIT 2022

Thank you

Franco Patano

Lead Product Specialist, Databricks

Dillon Bostwick

Sr. Solutions Architect, Databricks

Shannon Barrow

Sr. Solutions Architect, Databricks