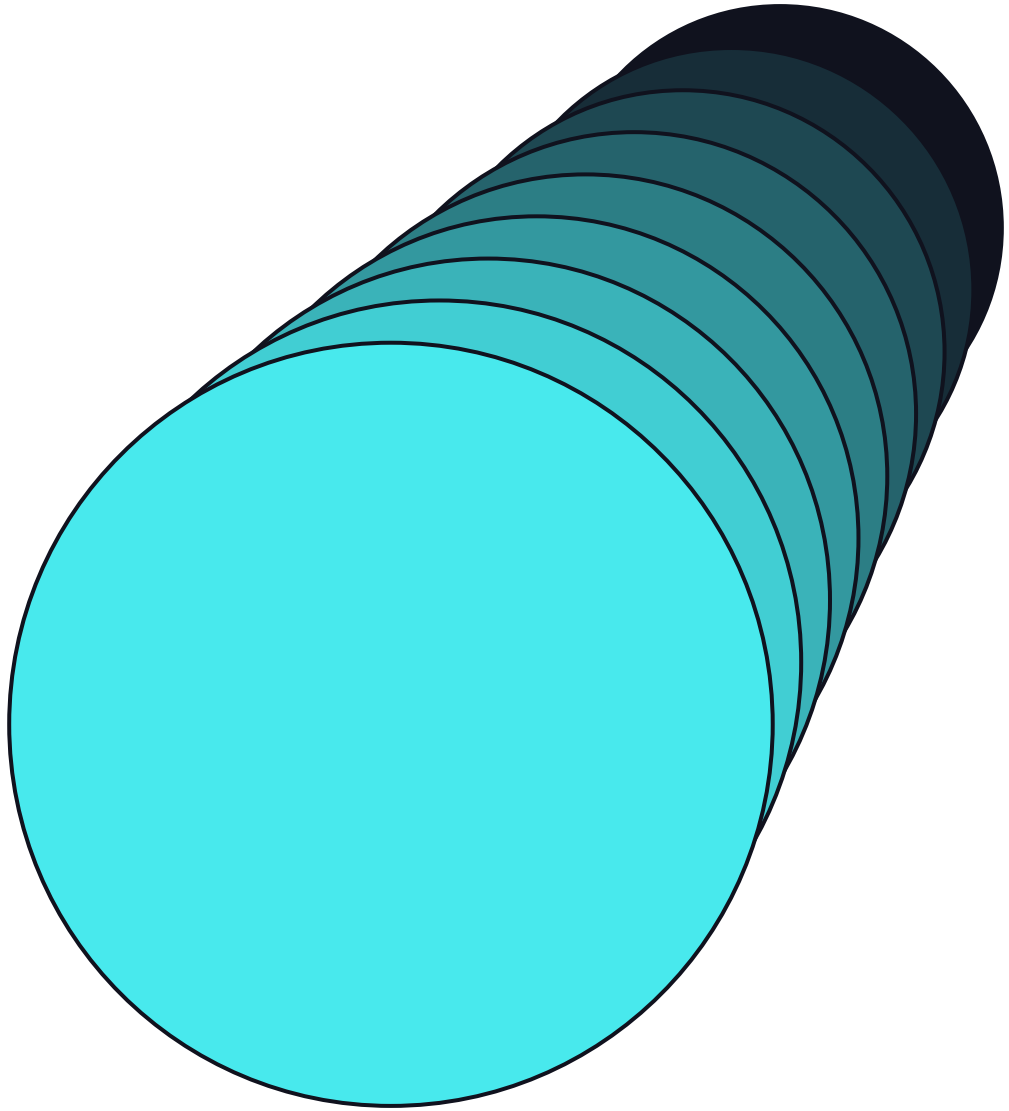
Trillion Rows per day powered by Delta Lake At Adobe



Yeshwanth Vijayakumar

Director Of Engineering @ Adobe

Agenda

Building over earlier talks in 2022,2023 and sharing new patterns

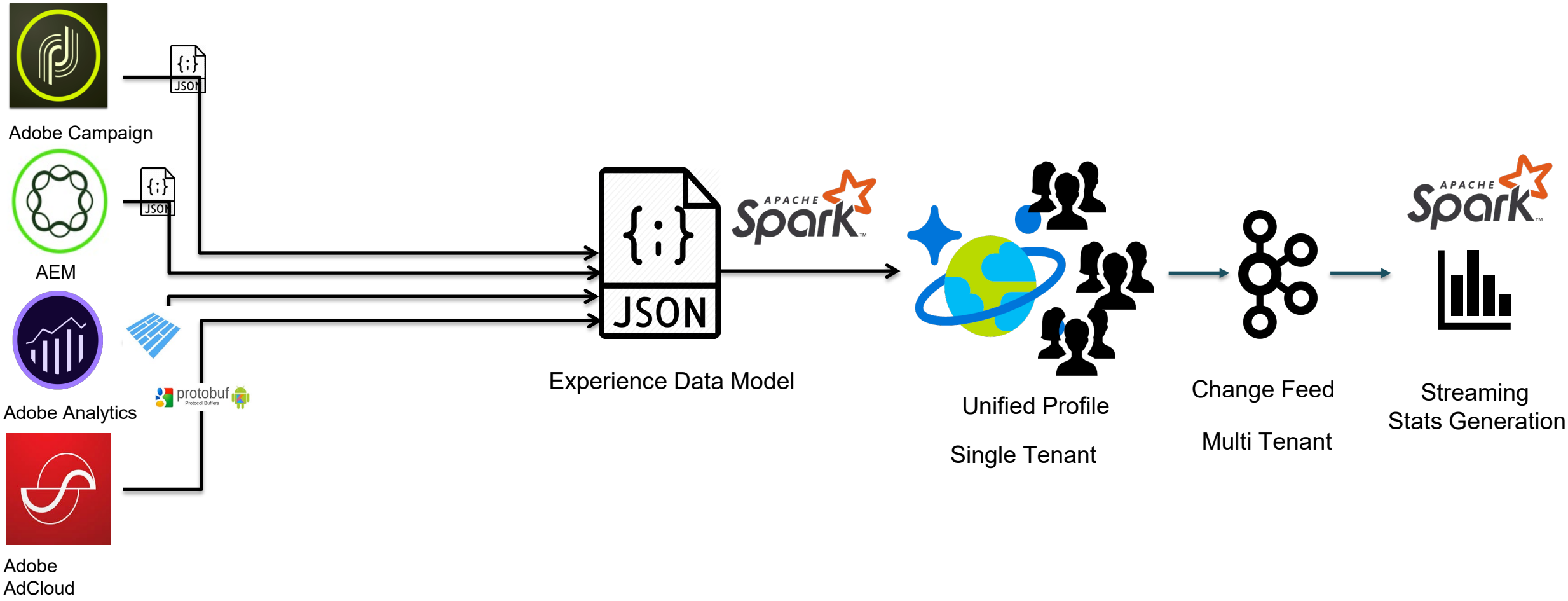
- About The data?
- Scaling the Writer
- Data representation and nested schema evolution
    - Strings FTW - data manipulation using UDFs
- Transaction Management and tracking Delta Using Delta
    - 2 phase commits
    - Append-Only DeltaTables to track global history across thousands of tables
- Maintenance Operations and Their Scaling Gotchas

Agenda

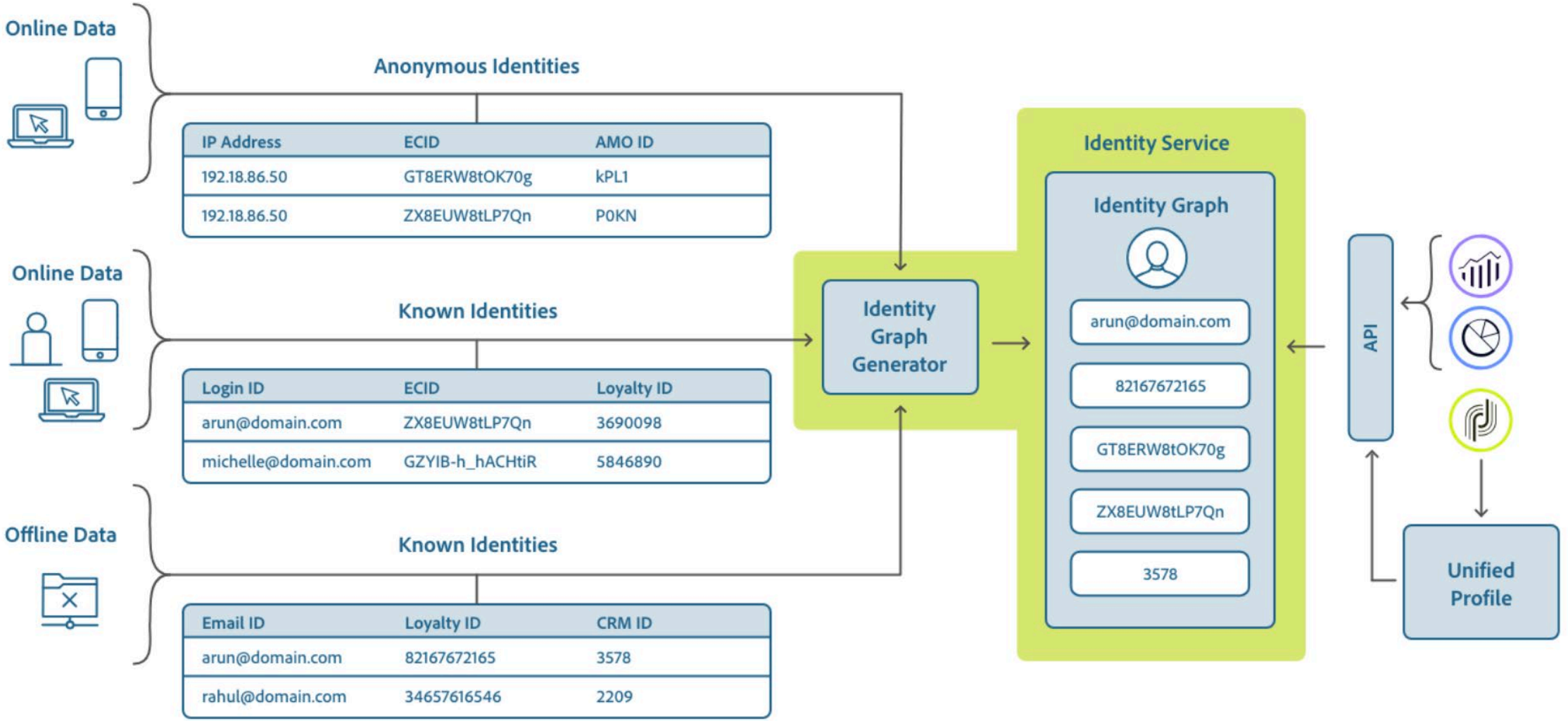
Building over earlier talks in 2022,2023 and sharing new patterns

- About The data?
- Scaling the Writer
  - Thousand Stream problem-managing thousands of Structured Streaming writers at scale
  - JVM agnostic locking for partition level concurrency control
  - Balancing Multi Tenancy and Single Tenancy
- Transaction Management and tracking Delta Using Delta
  - 2 phase commits
  - Append-Only DeltaTables to track global history across thousands of tables
- Data representation and nested schema evolution
  - Strings FTW - data manipulation using UDFs
- Maintenance Operations and Their Scaling Gotchas

Unified Profile Data Ingestion



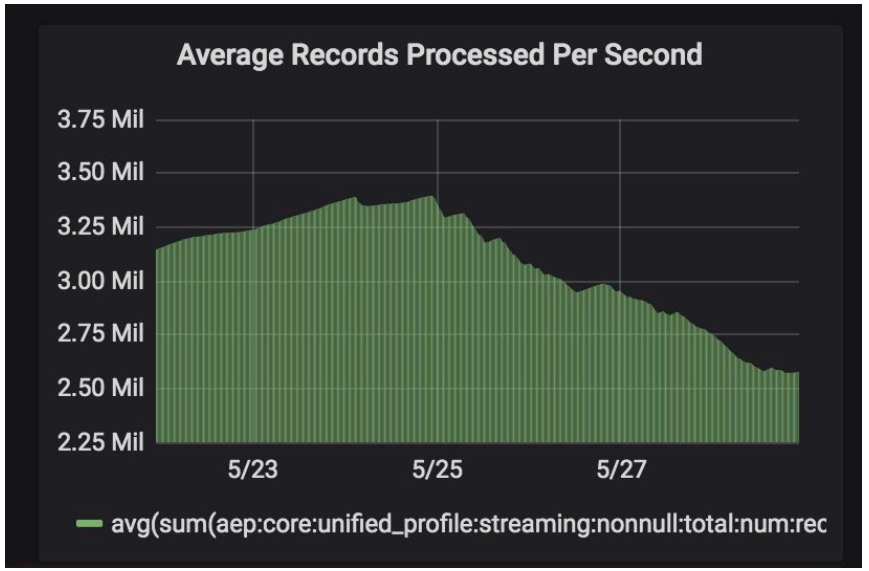
Linking Identities



Complexities?

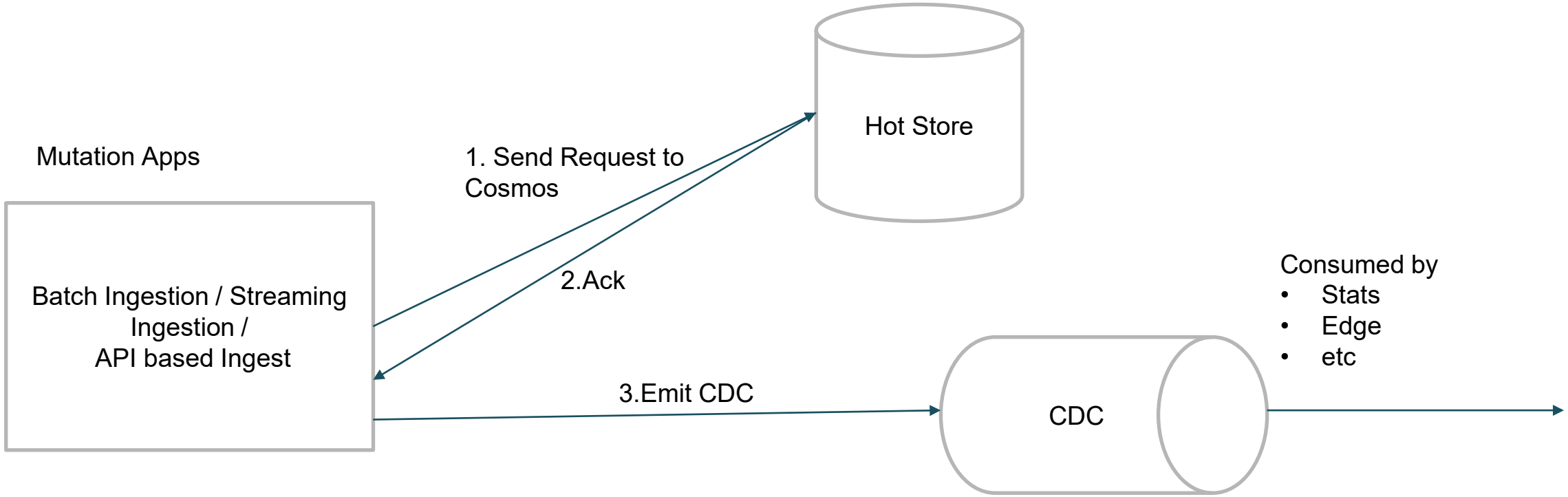
- Nested Fields
  - a.b.c.d[*].e nested hairiness!
  - Arrays!
  - MapType
- Every Tenant has a different Schema!
  - e2e-seg-2-20-20-02
- Schema evolves constantly
  - Fields can get deleted, updated.
- Multiple Sources
  - Streaming
  - Batch

Scale?



- 1-2 Trillion Rows of changes a day
- Tenants have 10+ Billions of rows
- PBs of data
- Million RPS peak across the system
- Triggers multiple downstream applications
  - Segmentation
  - Activation

CDC (existing)



Dataflow with DeltaLake

primary Id	relatedId	field 1	field2	field1000
103	103,789,101	q	w	r
789	103,789,101	x	y	z
101	103,789,101	x	y	z

Cosmos DB

Backfill

Raw Table (per tenant)

primaryId	relatedId	jsonString
103	103,789,101	<jsonStr>
789	103,789,101	<jsonStr>
101	103,789,101	<jsonStr>

TenantLock in Redis

UPSERT/DELETE into Raw Table

Change Feed CDC

primaryId	relatedId	field1	field1000
103	103,789,101	q	r

CDC Dumper

Long Running Streaming Application

Stage Log - CosmosDB

Staging Table

APPEND only!

Partitioned by tenant and 15 min time intervals

Fetch Records to process

Processor

Check for Work every X minutes

Staging Tables FTW

Fan-In pattern vs Fan-out

- Multiple Source Writers Issue Solved
  - By centralizing all reads from CDC, since ALL writes generate a CDC
- Staging Table in APPEND ONLY mode
  - No conflicts while writing to it
- Filter out. Bad data > thresholds before making it to Raw Table
- Batch Writes by reading larger blocks of data from Staging Table
  - Since it acts time aware message buffer

Staging Table Logical View

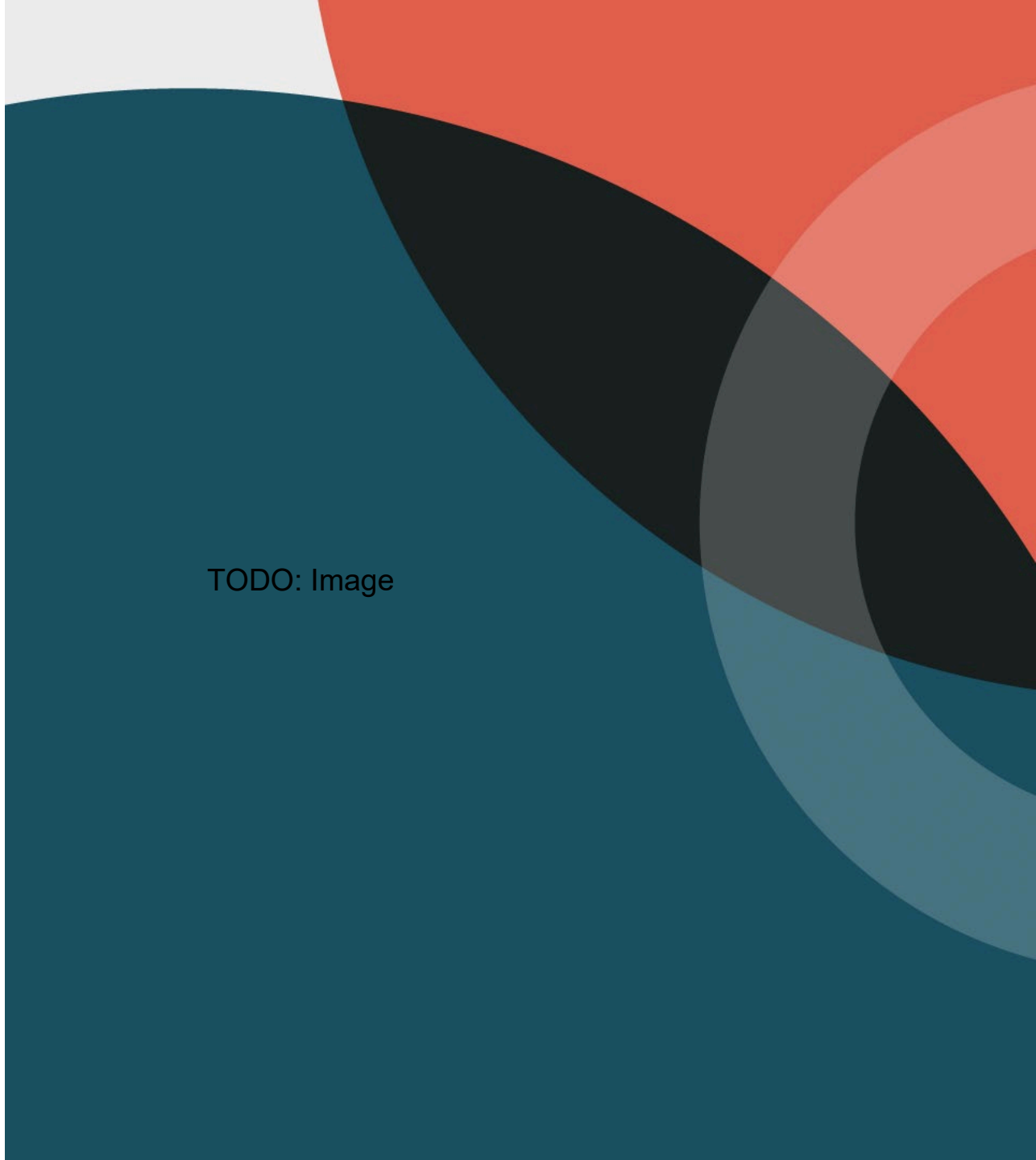
```
<TSKEY= 2021-01-01-09-15-Quarter=01 > -
        [
            x1-cdcRecord,
            x2-cdcRecord,
            x3-cdcRecord,
            x5-cdcRecord
        ]
 <TSKEY= 2021-01-01-09-15-Quarter=02 > -
     [
         x2-cdcRecord,
         x7-cdcRecord
     ]
  <TSKEY= 2021-01-01-09-15-Quarter=03 > - [
         x6-cdcRecord,
         x9-cdcRecord
      ]
```

ProgressMap

Org	Phase 1 LastSuccessfulTSKey
tenant1	2021-01-01-09-15-Quarter=01
tenant2	2021-01-02-07-10-Quarter=04
tenant3	2021-01-01-11-19-Quarter=03

2 Phase Commit Protocol

- Write to Stage Table and Stage Log is governed by a 2 phase commit
- It is also idempotent using
  - Custom Stage Log Flags on a High Consistency Mode
  - Additionally use
    - .option("txnVersion", batch_id).option("txnAppId", app_id)



Why choose JSON String format?

- We are doing a lazy Schema on-read approach.
  - Yes. this is an anti-pattern.
- Nested Schema Evolution was not supported on update in delta in 2020
  - Supported with latest version
- We want to apply conflict resolution before upsert-ing
  - Eg. resolveAndMerge(newData, oldData)
  - UDF's are strict on types, with the plethora of difference schemas , it is crazy to manage UDF per org in Multi tenant fashion
  - Now we just have simple JSON merge udfs
    - We use json-iter which is very efficient in loading partial bits of json and in manipulating them.
- Don't you lose predicate pushdown?
  - We have pulled out all main push-down filters to individual columns
    - Eg. timestamp, recordType, id, etc.
  - Profile workloads are mainly scan based since we can run 1000's of queries at a single time.
  - Reading the whole JSON string from datalake is much faster and cheaper than reading from Cosmos for 20% of all fields.

Schema On Read is more future safe approach for raw data

- Wrangling Spark Structs is not user friendly
- JSON schema is messy
  - Crazy nesting
  - Add maps to the equation, just the schema will be in MBs
- Schema on Read using Json-iter means we can read what we need on a row by row basis
- Materialized Views WILL have structs!

Cmd 42

```
%fs
ls /tmp/atlastest/4932D947587C1DF40A49423C@AdobeOrg.raw.partitioned.delta
```

Error runni

	path	name	size
1	dbfs:/tmp/atlastest/4932D947587C1DF40A49423C@AdobeOrg.raw.partitioned.delta/_delta_log/	_delta_log/	0
2	dbfs:/tmp/atlastest/4932D947587C1DF40A49423C@AdobeOrg.raw.partitioned.delta/rt=__HIVE_DEFAULT_PARTITION__/	rt=__HIVE_DEFAULT_PARTITION__/	0
3	dbfs:/tmp/atlastest/4932D947587C1DF40A49423C@AdobeOrg.raw.partitioned.delta/rt=identity/	rt=identity/	0
4	dbfs:/tmp/atlastest/4932D947587C1DF40A49423C@AdobeOrg.raw.partitioned.delta/rt=keyvalue/	rt=keyvalue/	0
5	dbfs:/tmp/atlastest/4932D947587C1DF40A49423C@AdobeOrg.raw.partitioned.delta/rt=timeseries/	rt=timeseries/	0

Showing all 5 rows.

Cmd 42

```
%fs
ls adl://datalakeeppvdoocjhd2.azuredatalakestore.net/core/profile/atlas/v1/4932D947587C1DF40A49423C@AdobeOrg.raw.partitioned.delta/rt=timeseries/ek=5d64ec86b7469b1648cf1295/

```

	path	name	size
1	adl://datalakeeppvdoocjhd2.azuredatalakestore.net/core/profile/atlas/v1/4932D947587C1DF40A49423C@AdobeOrg.raw.partitio	tsdate=50376-03-13/	0

Showing all 1 rows.

Partition Scheme of Raw records

- RawRecords Delta Table
  - recordType
    - sourceId
      - timestamp (key-value records will use DEFAULT value)

        z-order on primaryId

z-order - Colocate column information in the same set of files using locality-preserving **space-filling curves**

```
TBLPROPERTIES
( delta.autoOptimize.autoCompact = true,
  delta.autoOptimize.optimizeWrite = true,
  delta.dataSkippingNumIndexedCols = 9,
  delta.logRetentionDuration = 'interval 30 days',
  delta.deletedFileRetentionDuration = 'interval 1 weeks'
 )
```

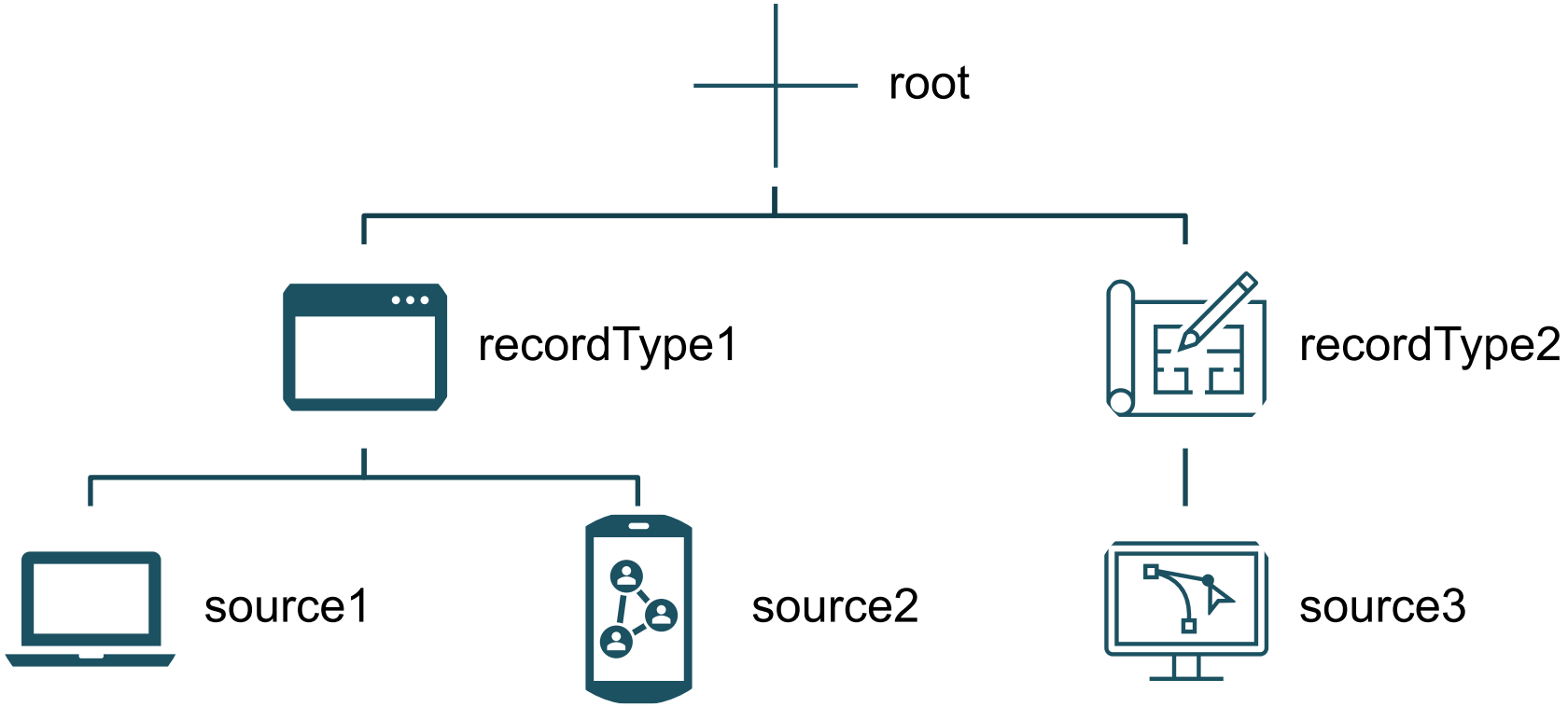
Scaling the Writer

- JVM agnostic locking for partition level concurrency control

Hive Style Partitions

- Highly concurrent operations
- Inserts and Updates on single partitions
- Deletes across partitions

Leads to



Conflicts!

	INSERT	UPDATE, DELETE, MERGE INTO	COMPACTION
INSERT	Cannot conflict		
UPDATE, DELETE, MERGE INTO	Can conflict	Can conflict	
COMPACTION	Cannot conflict	Can conflict	Can conflict

JVM Free Locking to rescue