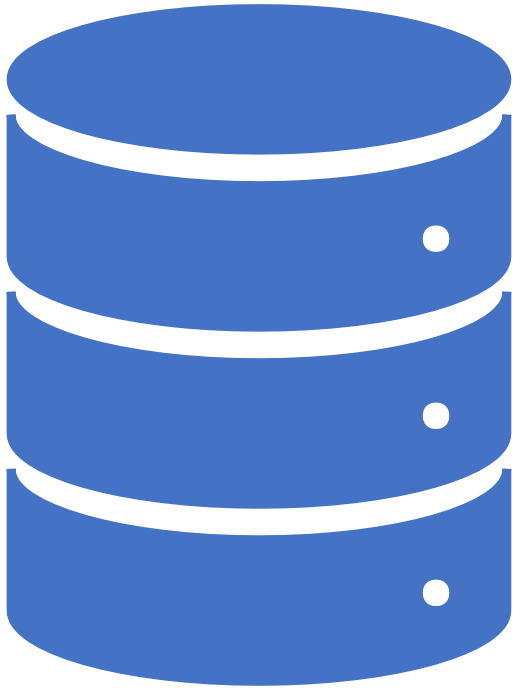


Scaling AWS Redshift Concurrency with Postgres

By Elliott Cordo, Will Liu, Paul Singman

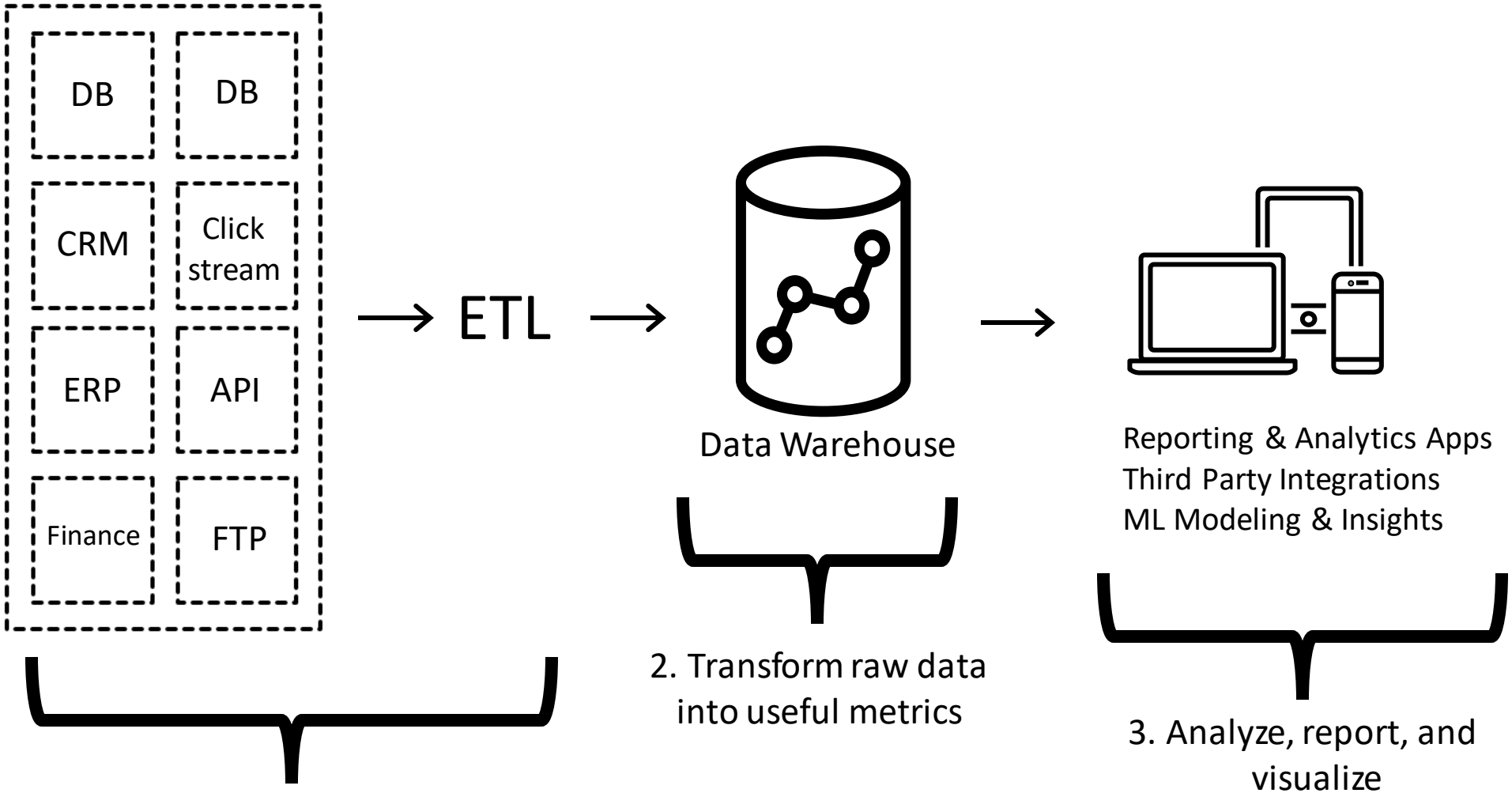
EQUINOX

Integrated luxury and lifestyle company with offerings centered on movement, nutrition, and regeneration

blink FITNESS | PURE YOGA | SOULCYCLE | FURTHERMORE FROM EQUINOX | INTRODUCING EQUINOX HOTELS

we operate more than 200 locations within every major city across the country in addition to London and Canada

Analytics Overview



Why A Data Warehouse?

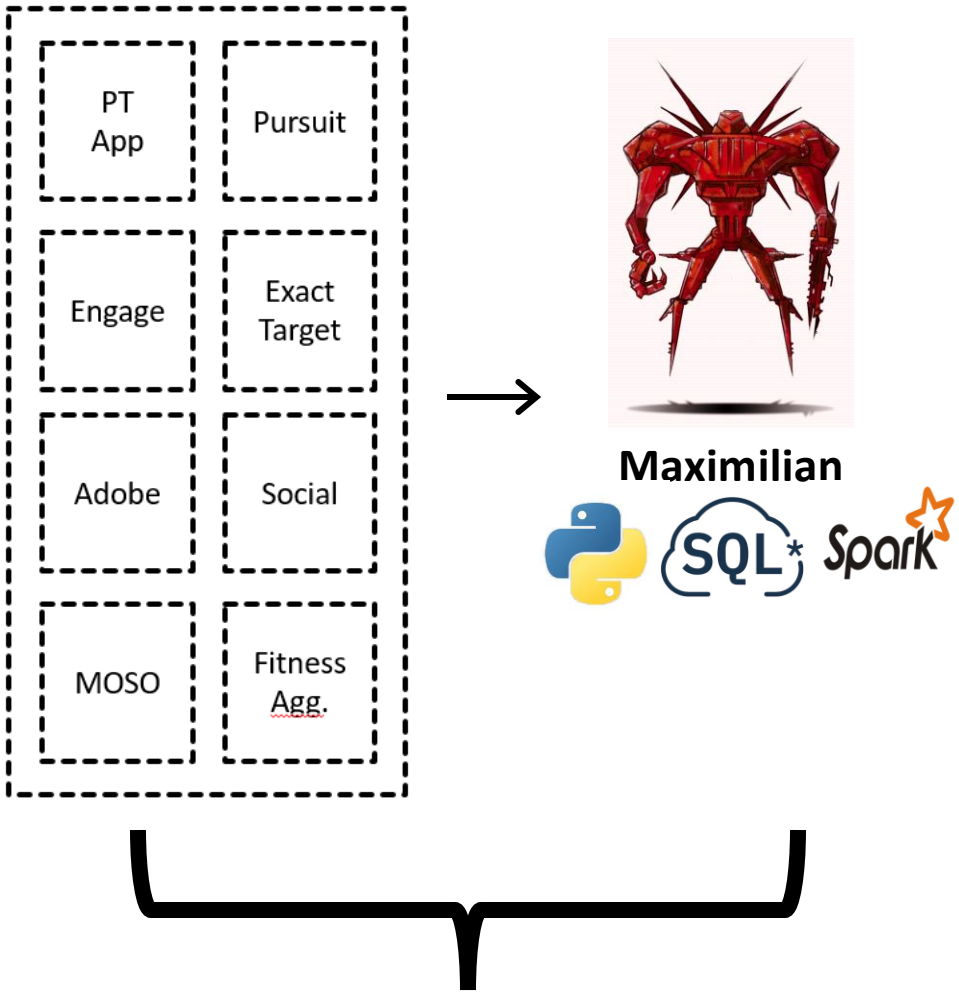
- Prevent data discrepancies
- Lower employee learning curve
- Avoid duplicating logic in multiple systems
- Isolate production DBs from analytic workloads

Why A Data Warehouse?

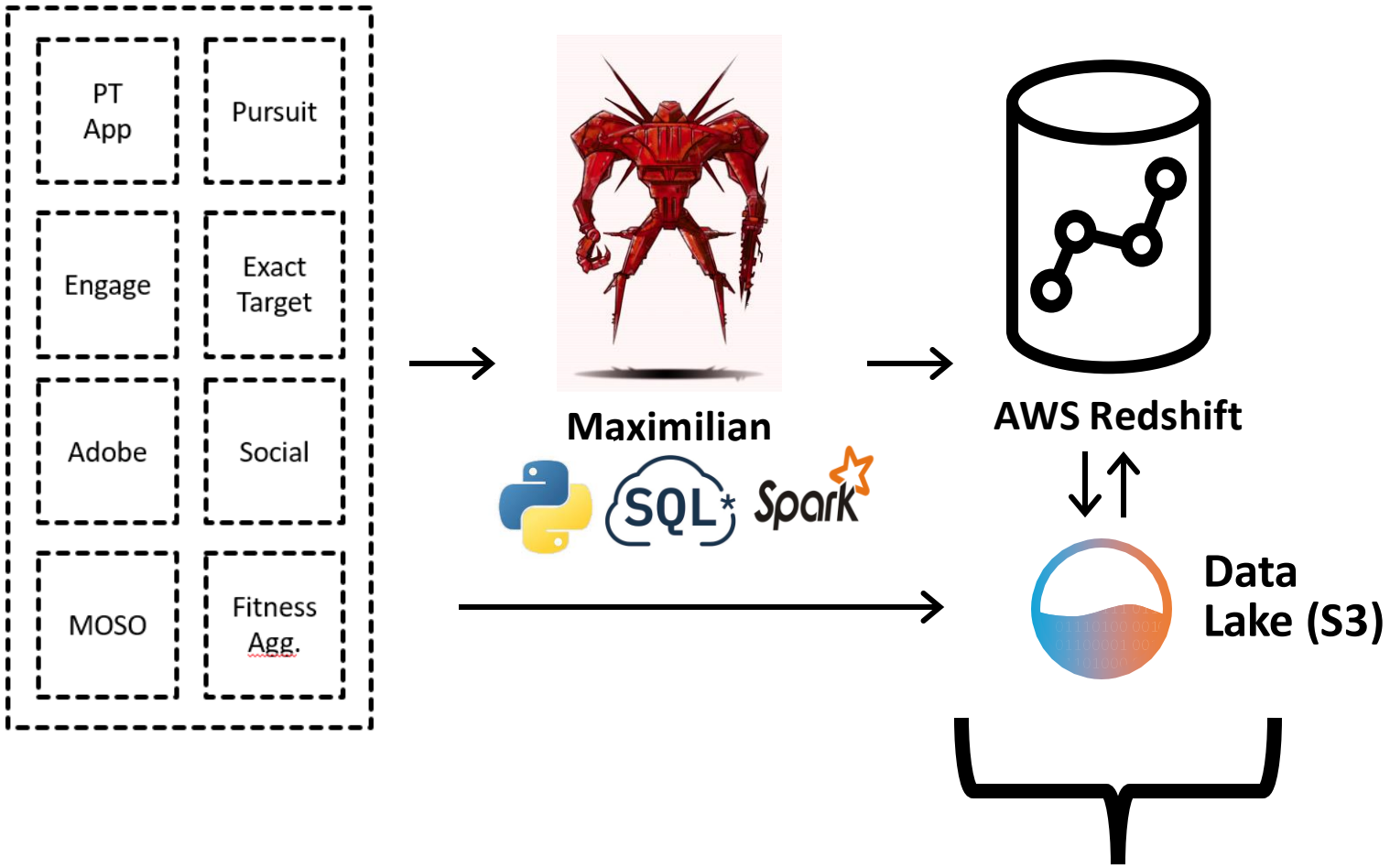
- Prevent data discrepancies
- Lower employee learning curve
- Avoid duplicating logic in multiple systems
- Isolate production DBs from analytic workloads

Presents a **single point of failure** so we created a **data replication failover procedure**

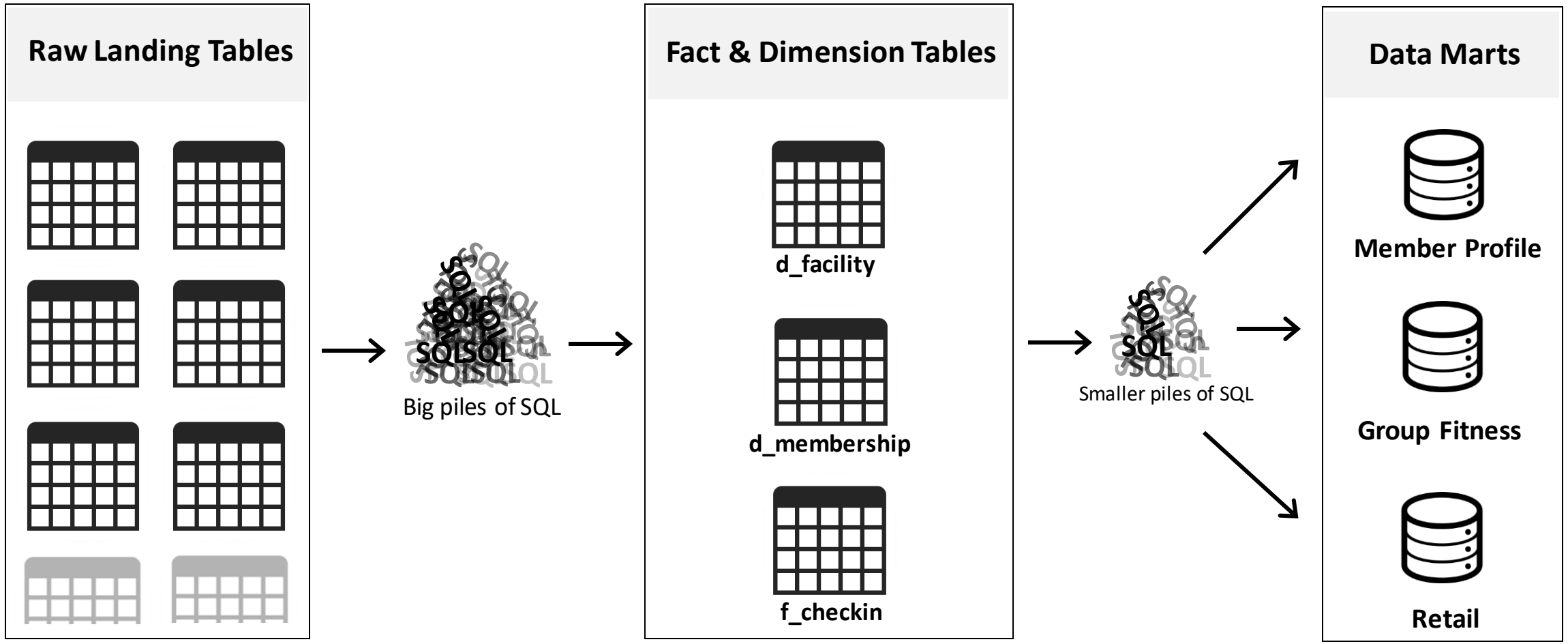
Data Ecosystem @ Equinox



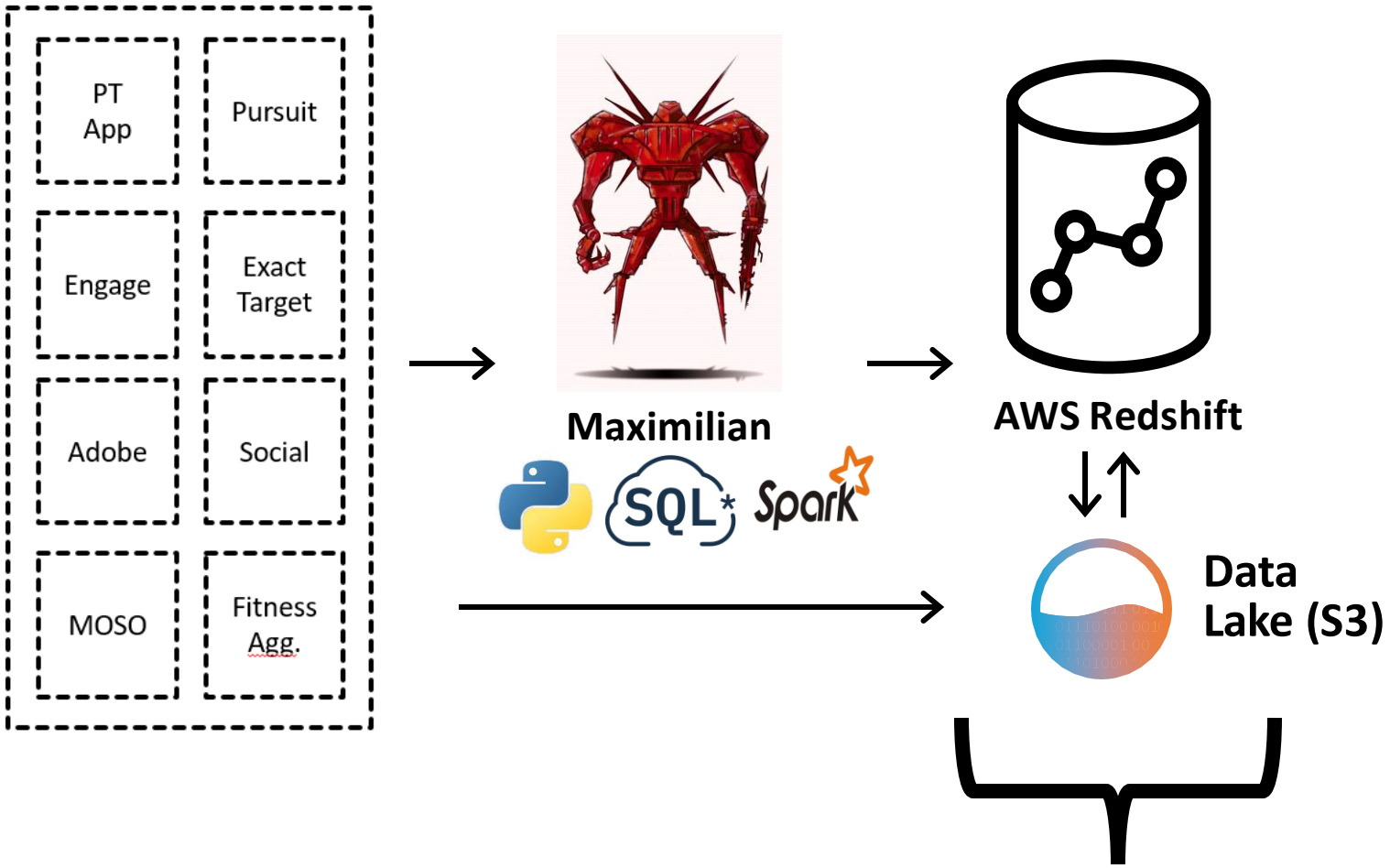
Data Ecosystem @ Equinox



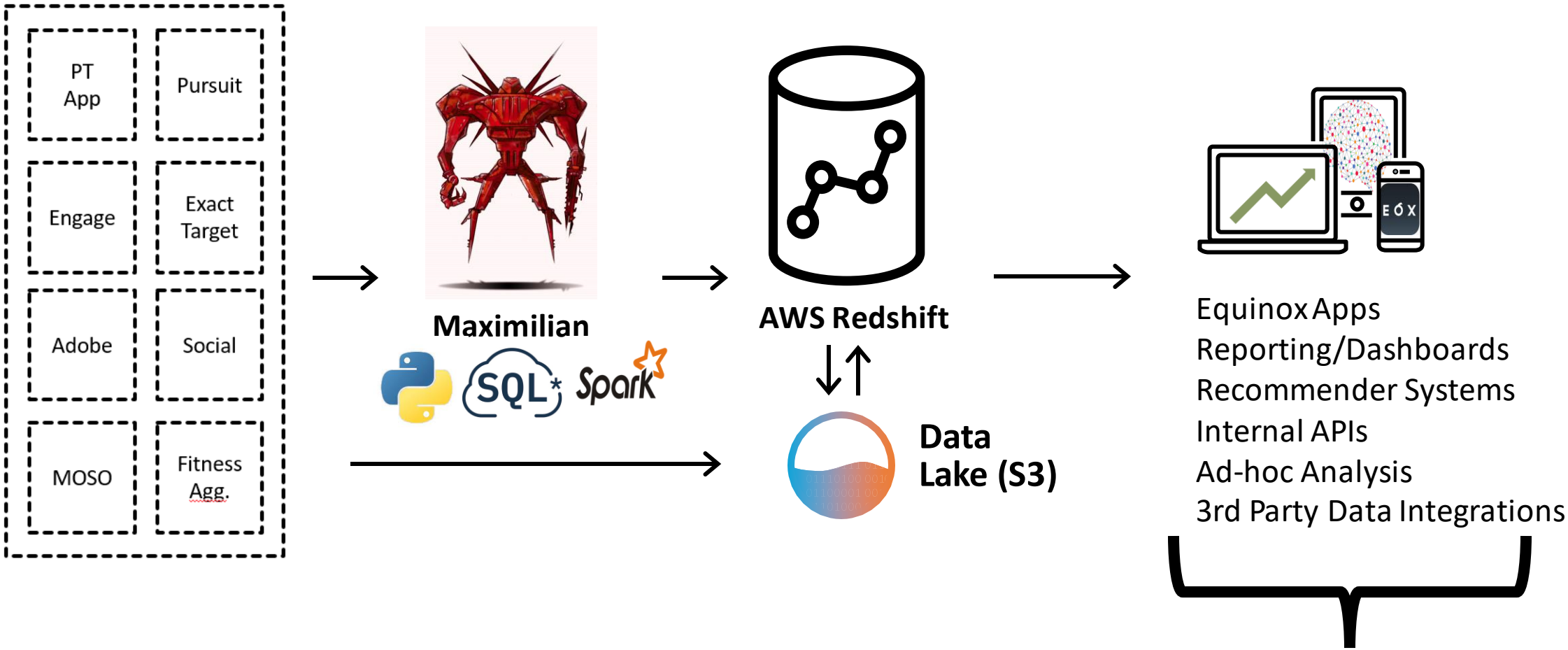
Redshift Warehouse Structure (J.A.R.V.I.S.)



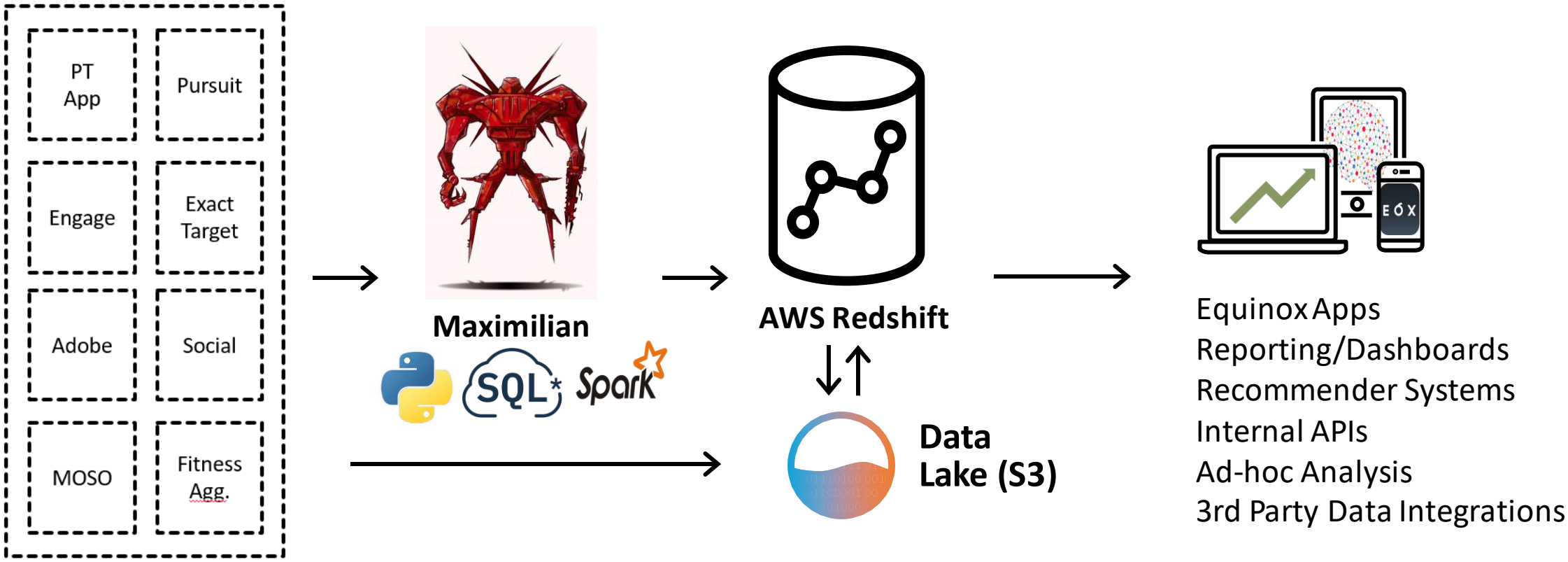
Data Ecosystem @ Equinox



Data Ecosystem @ Equinox

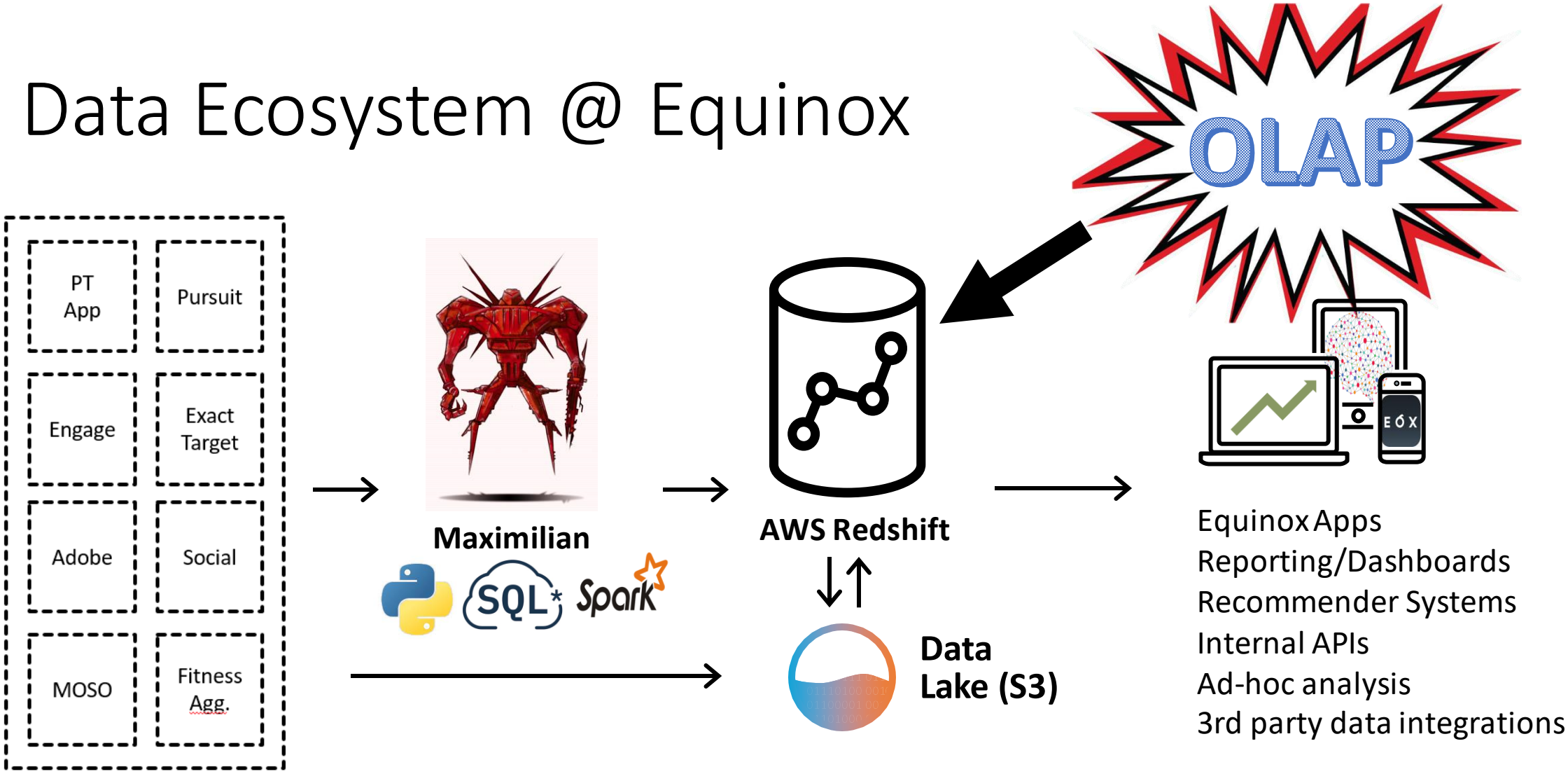


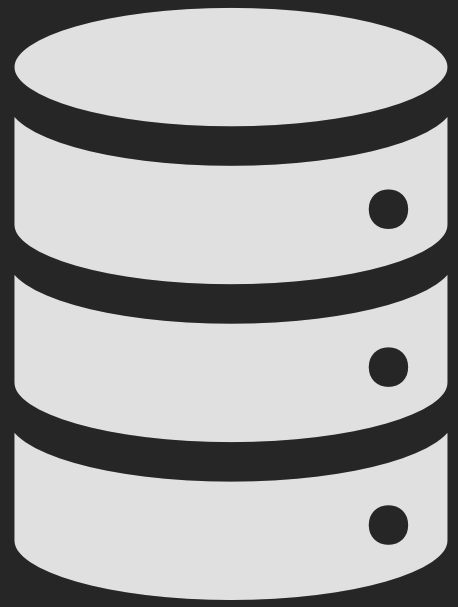
Data Ecosystem @ Equinox

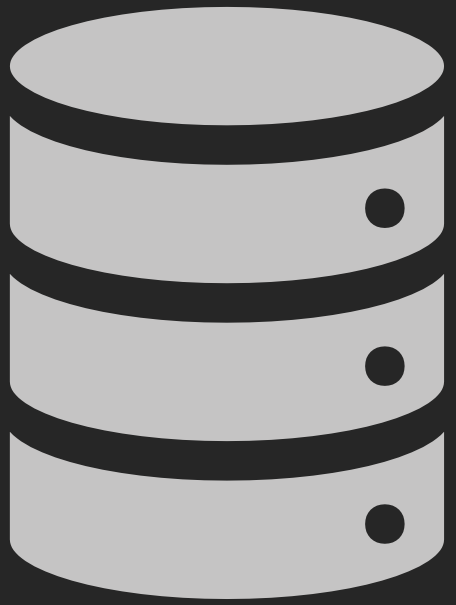


4. Data Monitoring & Quality

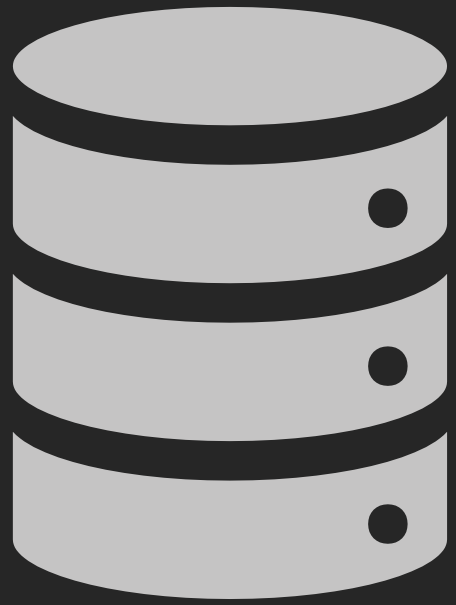
Data Ecosystem @ Equinox







On-line Transactional Processing
(OLTP)



On-line Analytical Processing
(OLAP)

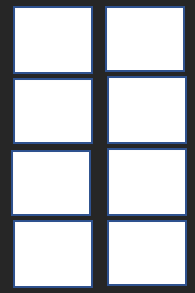
Key Features of Redshift

Key Features of Redshift

- Released by AWS in early 2013, based on Postgres v8.0.2
- Column-oriented storage
- Immutable 1MB block storage
- Massively-parallel processing compute engine
- Native multi-node (leader + workers) architecture
- Distribution key & sort key table settings
- Workload Management queue settings

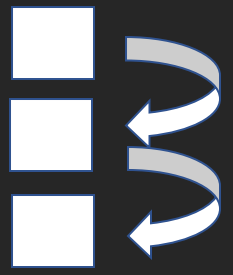
Key Features of Redshift

- Released by AWS in early 2013, based on Postgres v8.0.2
- Column-oriented storage
- Immutable 1MB block storage
- Massively-parallel processing compute engine
- Native multi-node (leader + workers) architecture
- Distribution key & sort key table settings
- Workload Management queue settings



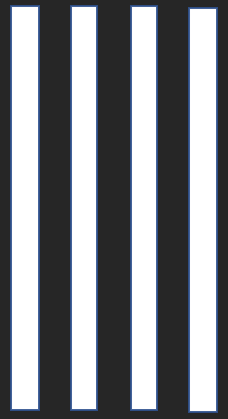
Key Features of Redshift

- Released by AWS in early 2013, based on Postgres v8.0.2
- Column-oriented storage
- Immutable 1MB block storage
- Massively-parallel processing compute engine
- Native multi-node (leader + workers) architecture
- Distribution key & sort key table settings
- Workload Management queue settings



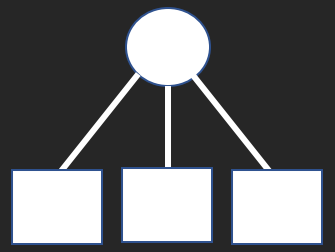
Key Features of Redshift

- Released by AWS in early 2013, based on Postgres v8.0.2
- Column-oriented storage
- Immutable 1MB block storage
- Massively-parallel processing compute engine
- Native multi-node (leader + workers) architecture
- Distribution key & sort key table settings
- Workload Management queue settings



Key Features of Redshift

- Released by AWS in early 2013, based on Postgres v8.0.2
- Column-oriented storage
- Immutable 1MB block storage
- Massively-parallel processing compute engine
- Native multi-node (leader + workers) architecture
- Distribution key & sort key table settings
- Workload Management queue settings



Key Features of Redshift

- Released by AWS in early 2013, based on Postgres v8.0.2
- Column-oriented storage
- Immutable 1MB block storage
- Massively-parallel processing compute engine
- Native multi-node (leader + workers) architecture
- Distribution key & sort key table settings
- Workload Management queue settings

Key Features of Redshift

- Released by AWS in early 2013, based on Postgres v8.0.2
- Column-oriented storage
- Immutable 1MB block storage
- Massively-parallel processing compute engine
- Native multi-node (leader + workers) architecture
- Distribution key & sort key table settings
- Workload Management queue settings

Key Features of Redshift

- Released by AWS in early 2013, based on Postgres v8.0.2
- Column-oriented storage
- Immutable 1MB block storage
- Massively-parallel processing compute engine
- Native multi-node (leader + workers) architecture
- Distribution key & sort key table settings
- Workload Management queue settings


WEAKNESS
STRENGTH

- Petabyte-scale disk storage
- Batch insertions & retrieval
- Complex computations

- High-frequency transactions
- Concurrent user connections

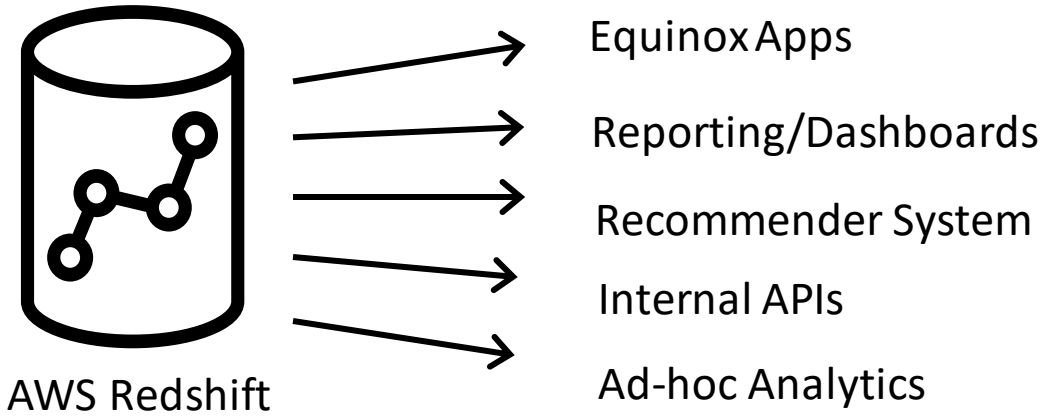
- Petabyte-scale disk storage
- Batch insertions & retrieval
- Complex computations

- High-frequency transactions
- Concurrent user connections

10x - 100x slower on simple SELECT
500 connection limit (per cluster)
50 connection limit (per user-defined queue)

- Petabyte-scale disk storage
- Batch insertions & retrieval
- Complex computations

Warehouse Consumers



Warehouse Consumers

Sales Review Details

From Date: 3/1/2019 To Date: 3/17/2019

[redacted]

3DayCXL

Acct Code	Club	Member Id	Member Name	Member Status	Date	Payment Type	Class	Commission Status	Count
115	Park Avenue	1003931724	[redacted]	Cancelled	3/6/19	BM	Select Monthly	NO COMM	-1
115	Park Avenue	1003921211	[redacted]	Cancelled	3/12/19	PIF	All Access Yearly	INC SIGNATURES, INCOMPLETE CONTACT INFO	-1
								Total Count:	**-2**

3DayReactivate

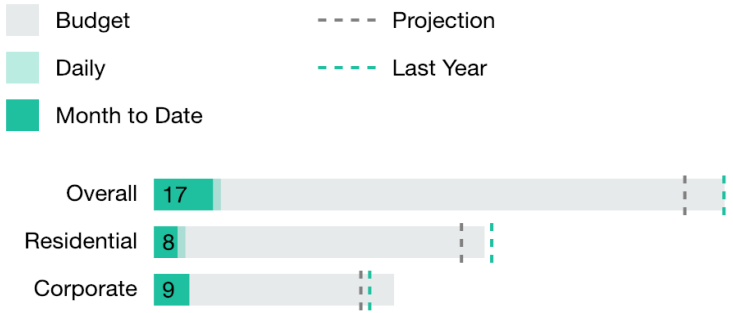
Acct Code	Club	Member Id	Member Name	Member Status	Date	Payment Type	Class	Commission Status	Count
115	Park Avenue	1003931724	[redacted]	Active	3/6/19	BM	All Access Monthly	DUE 1	1
								Total Count:	**1**

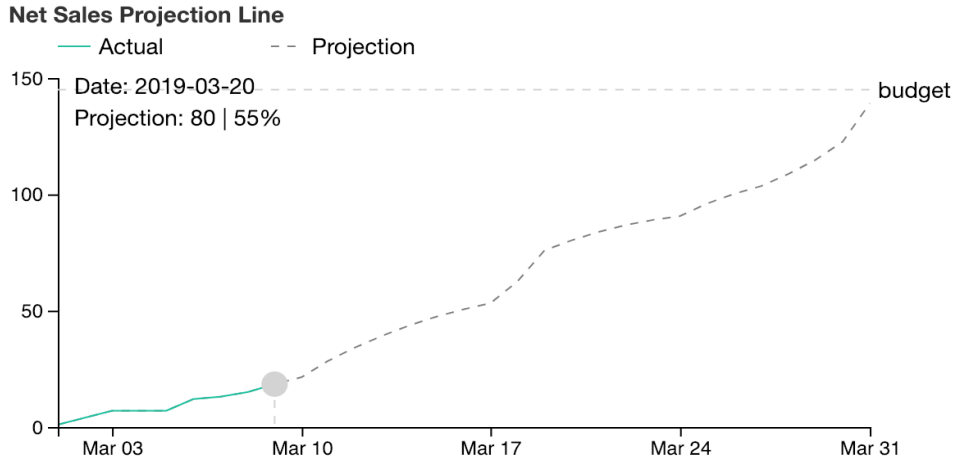
New

Acct Code	Club	Member Id	Member Name	Member Status	Date	Payment Type	Class	Commission Status	Count
115	Park Avenue	1001742000	[redacted]	Active	3/4/19	BM	All Access Monthly	DUE 3	1
115	Park Avenue	1003929636	[redacted]	Active	3/4/19	BM	Select Monthly	DUE 1	1
115	Park Avenue	1003900694	[redacted]	Active	3/5/19	BM	All Access Monthly	INC CORPORATE	1
115	Park Avenue	1003932836	[redacted]	Active	3/5/19	BM	Select Monthly	INC SIGNATURES	1
115	Park Avenue	1003932984	[redacted]	Active	3/5/19	BM	Select Monthly	DUE 1	1
115	Park Avenue	1001796524	[redacted]	Active	3/6/19	BM	Select Monthly	DUE 1	1
115	Park Avenue	1003931724	[redacted]	Active	3/6/19	BM	Select Monthly	NO COMM	1
115	Park Avenue	1003198113	[redacted]	Pending Start	3/7/19	BM	All Access Monthly	DUE 1	1
115	Park Avenue	1003630343	[redacted]	Pending Start	3/7/19	BM	Select Monthly	DUE 3	1

SALES

The progress bar charts show the daily actual, month to date and budget relationship of sales.
The gray bars are the end of month budget as 100 percent, the green ones are the month to date actual in percentage of budget, with the exact value showing as label. The daily new actual value is marked out in lighter green color.
Green dot lines and gray dot lines are the projection and last year value as percentage of each category.



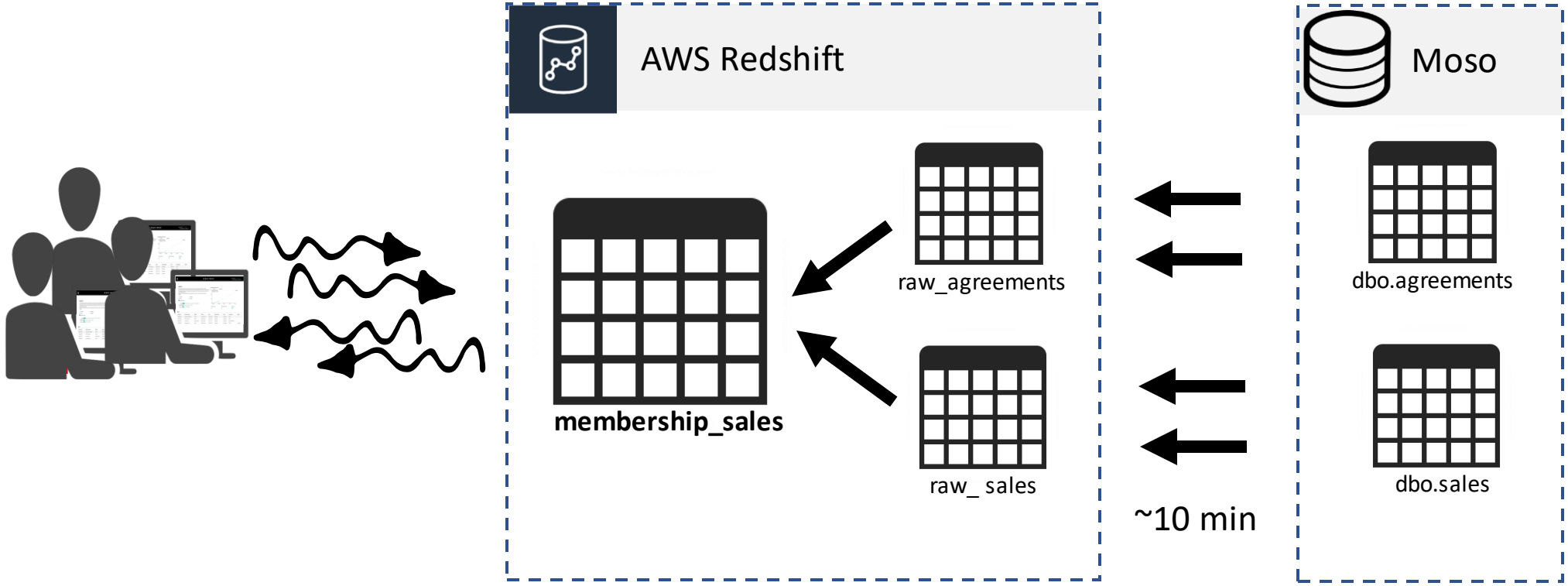


3 Days Cancel	3 Days Reactivate	Net Sales
1	1	17

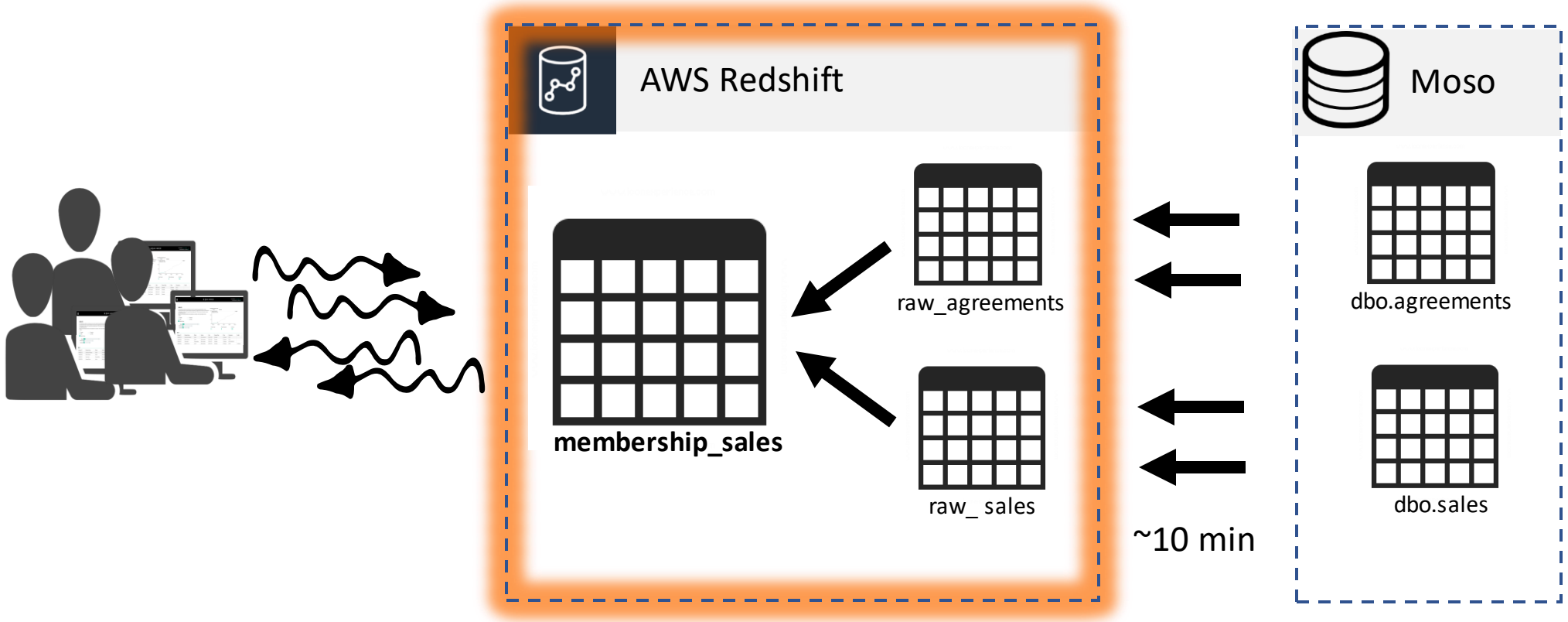
NEW

Member ID	Member Name	Status	Advisor	Date	Payment Type	Class	Commission	Equifit
1003858732	[redacted]	Active	Tony Marchena	2019-03-02	Perpetual	All Access	DUE 1	
1003936006	[redacted]	Active	Ryan Sanders	2019-03-06	Perpetual	All Access	DUE 1	
1003932537	[redacted]	Pending Start	Ryan Sanders	2019-03-07	Perpetual	All Access	INC CORPORATE	
1003202293	[redacted]	Active	Ryan Sanders	2019-03-03	Perpetual	All Access	DUE 2	Confirmed

Sales Reporting Architecture

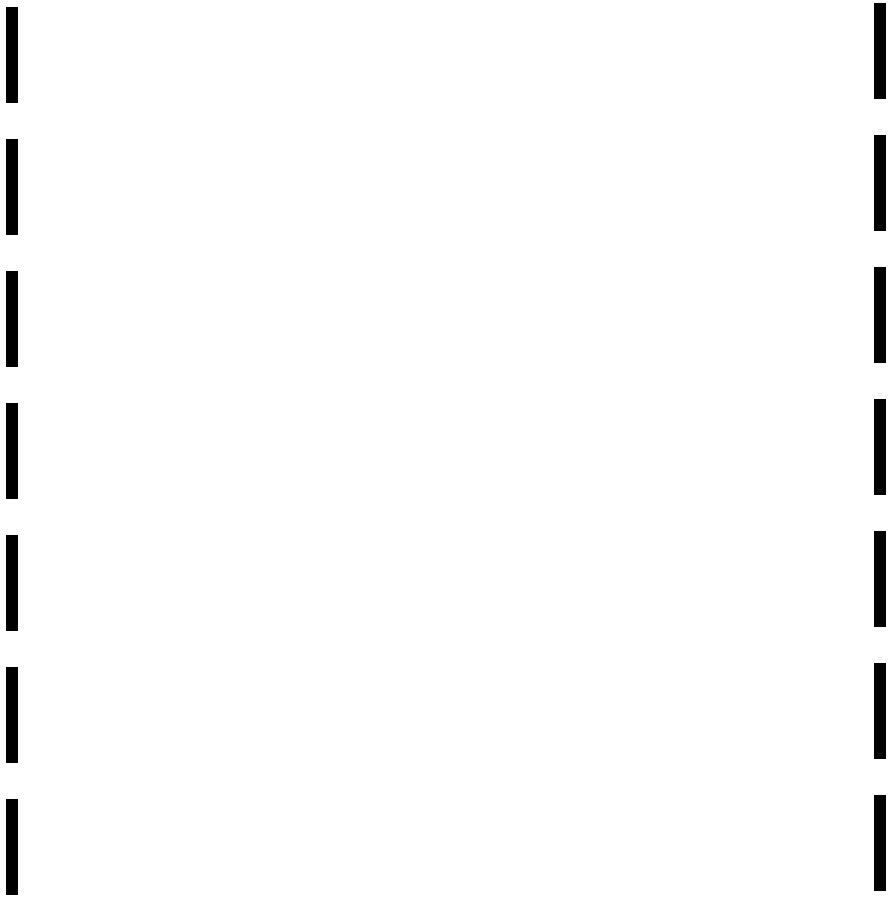


Sales Reporting Architecture **Problem**



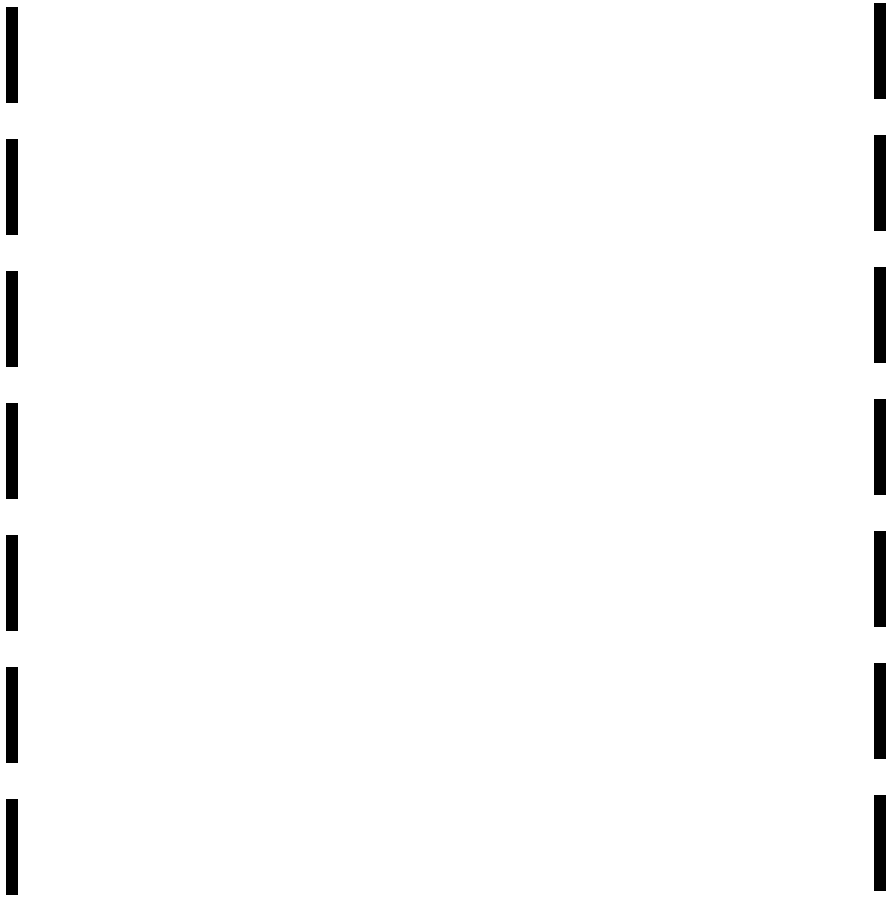
10x - 100x slower on simple SELECT
500 connection limit (per cluster)
50 connection limit (per user-defined queue)

Potential Solutions

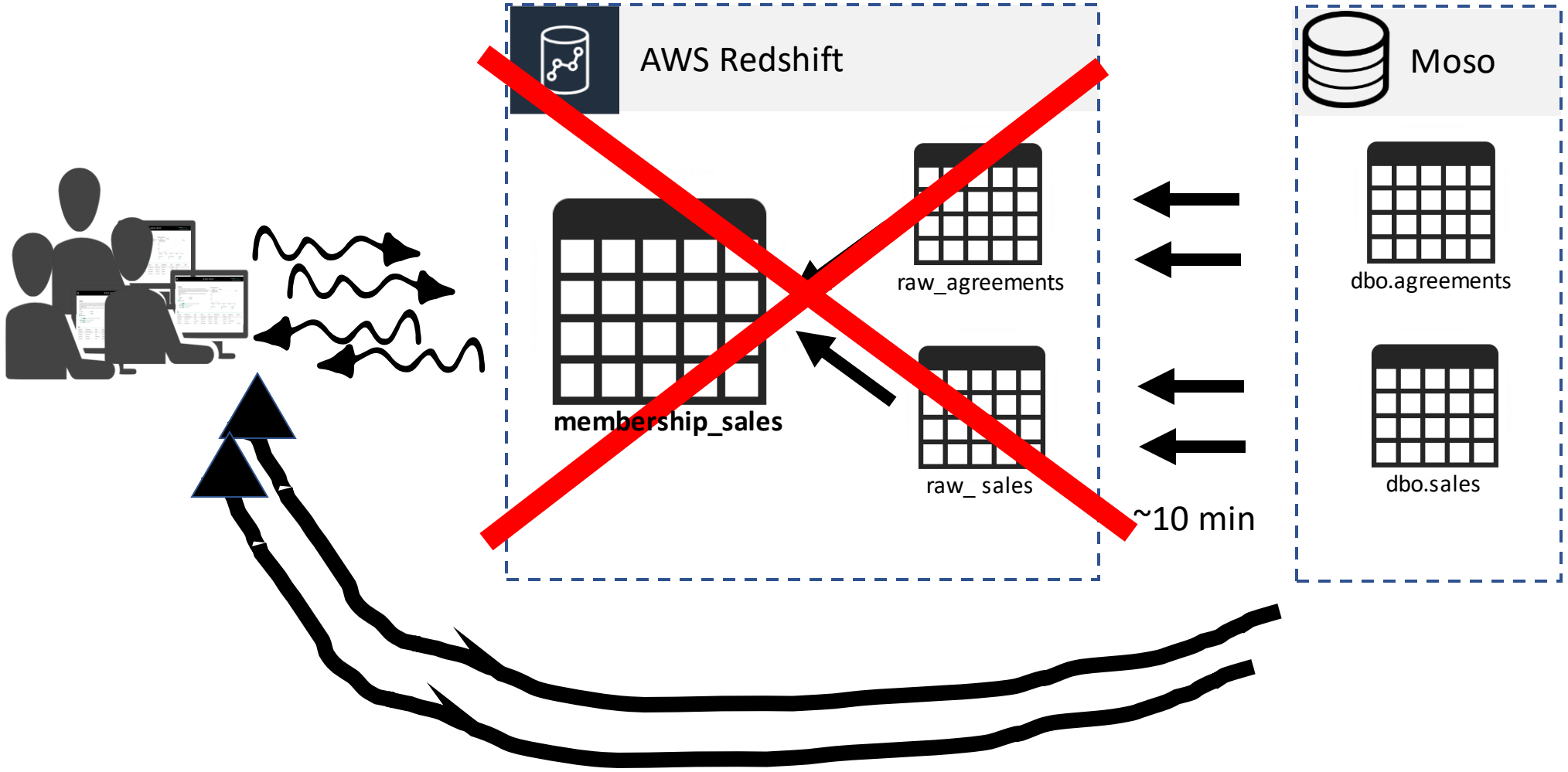


Potential Solutions

1. Pull from source DB



Sales Reporting Architecture



Potential Solutions

1. Pull from source DB

Pros

- Source DB is OLTP

Cons

- Sales logic is complex!
- Burdens prod DB

Potential Solutions

1. Pull from source DB

Pros

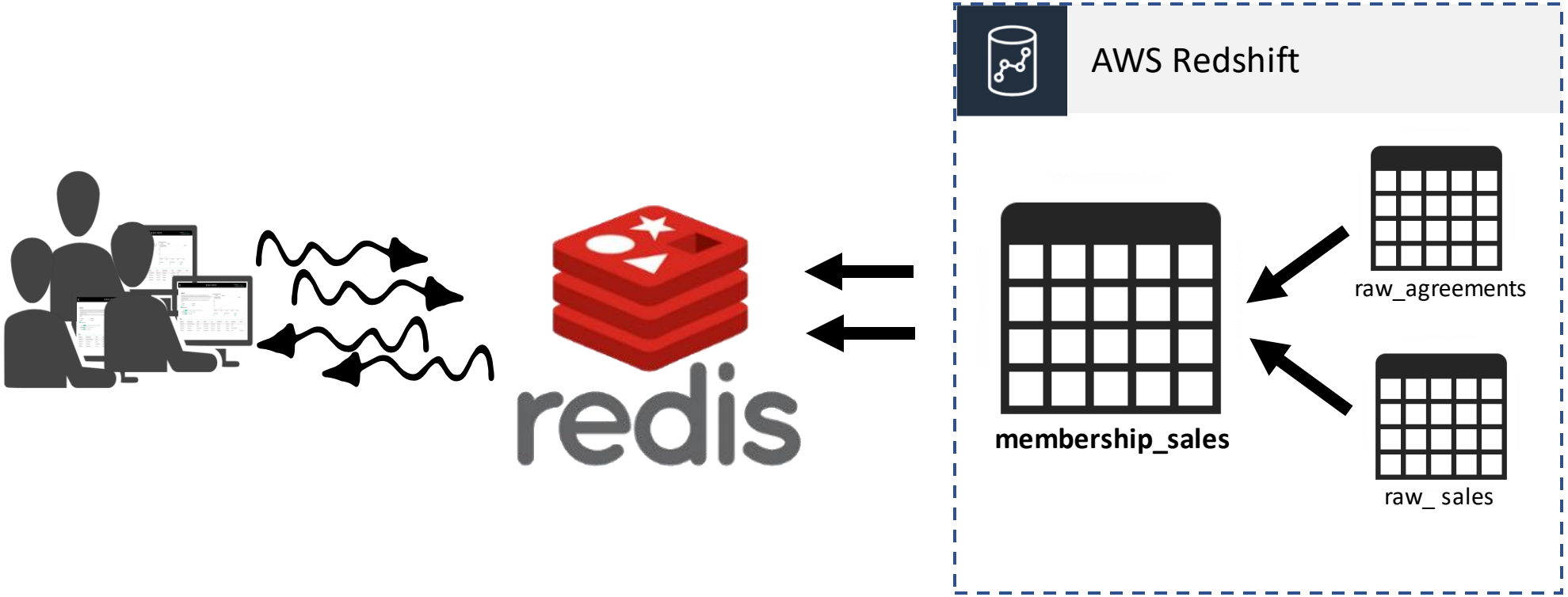
- Source DB is OLTP

Cons

- Sales logic is complex!
- Burdens prod DB

2. Cache in Redis

Sales Reporting Architecture



Potential Solutions

1. Pull from source DB

Pros

- Source DB is OLTP

Cons

- Sales logic is complex!
- Burdens prod DB

2. Cache in Redis

Pros

- Very fast performance

Cons

- Non-relational data structure
- Keys must be created for every data view

Potential Solutions

1. Pull from source DB

Pros

- Source DB is OLTP

Cons

- Sales logic is complex!
- Burdens prod DB

2. Cache in Redis

Pros

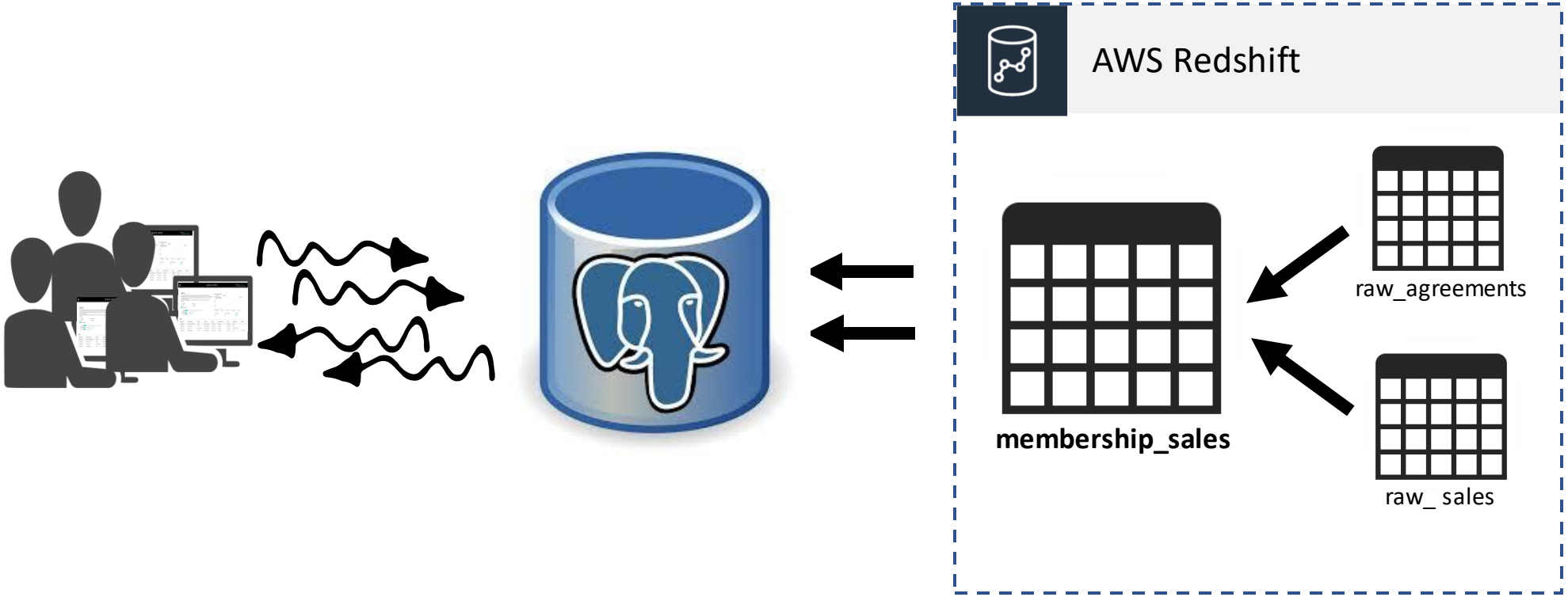
- Very fast performance

Cons

- Non-relational data structure
- Keys must be created for every data view

3. Copy to another DB

Sales Reporting Architecture



Potential Solutions

1. Pull from source DB

Pros

- Source DB is OLTP

Cons

- Sales logic is complex!
- Burdens prod DB

2. Cache in Redis

Pros

- Very fast performance

Cons

- Non-relational data structure
- Keys must be created for every data view

3. Copy to another DB

Pros

- Maintain relational format

Cons

- Additional ETL step

Potential Solutions

1. Pull from source DB

Pros

- Source DB is OLTP

Cons

- Sales logic is complex!
- Burdens prod DB

2. Cache in Redis

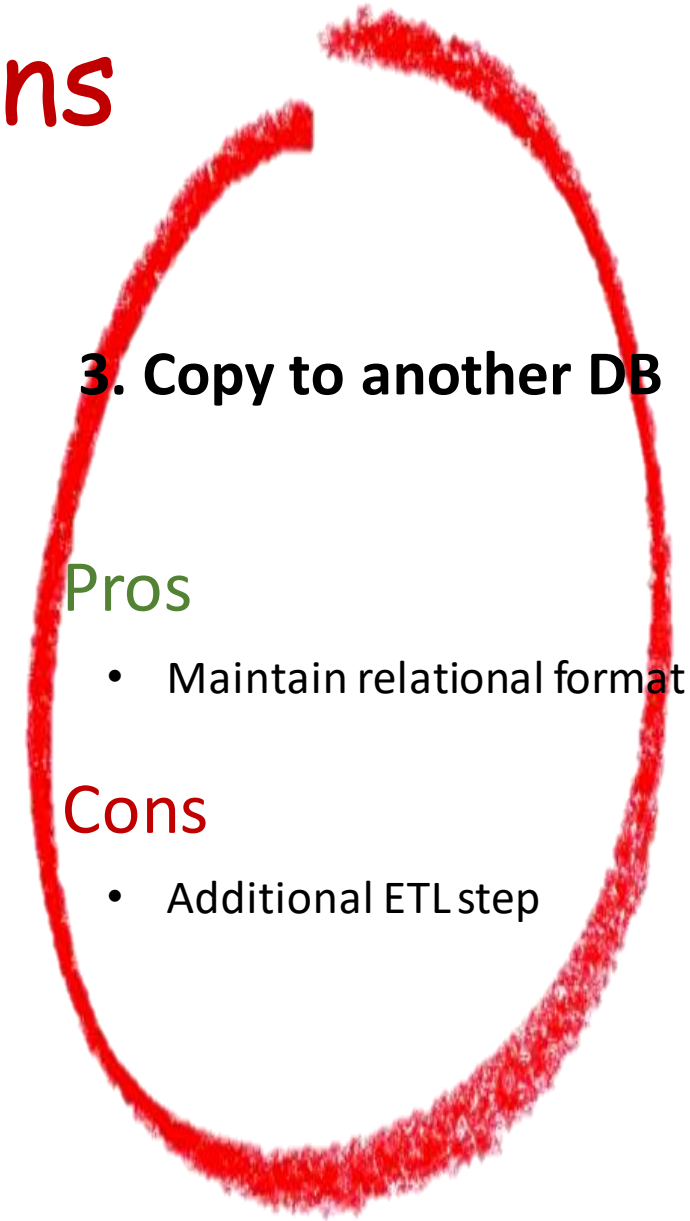
Pros

- Very fast performance

Cons

- Non-relational data structure
- Keys must be created for every data view

3. Copy to another DB



Pros

- Maintain relational format

Cons

- Additional ETL step

PostgreSQL Foreign Data Wrapper

PostgreSQL Foreign Data Wrapper


AWS Big Data Blog

JOIN Amazon Redshift AND Amazon RDS PostgreSQL WITH dblink

by Tony Gibbs | on 03 JUN 2016 | in Amazon RDS, Amazon Redshift, AWS Big Data | Permalink | Comments | Share

Tony Gibbs is a Solutions Architect with AWS

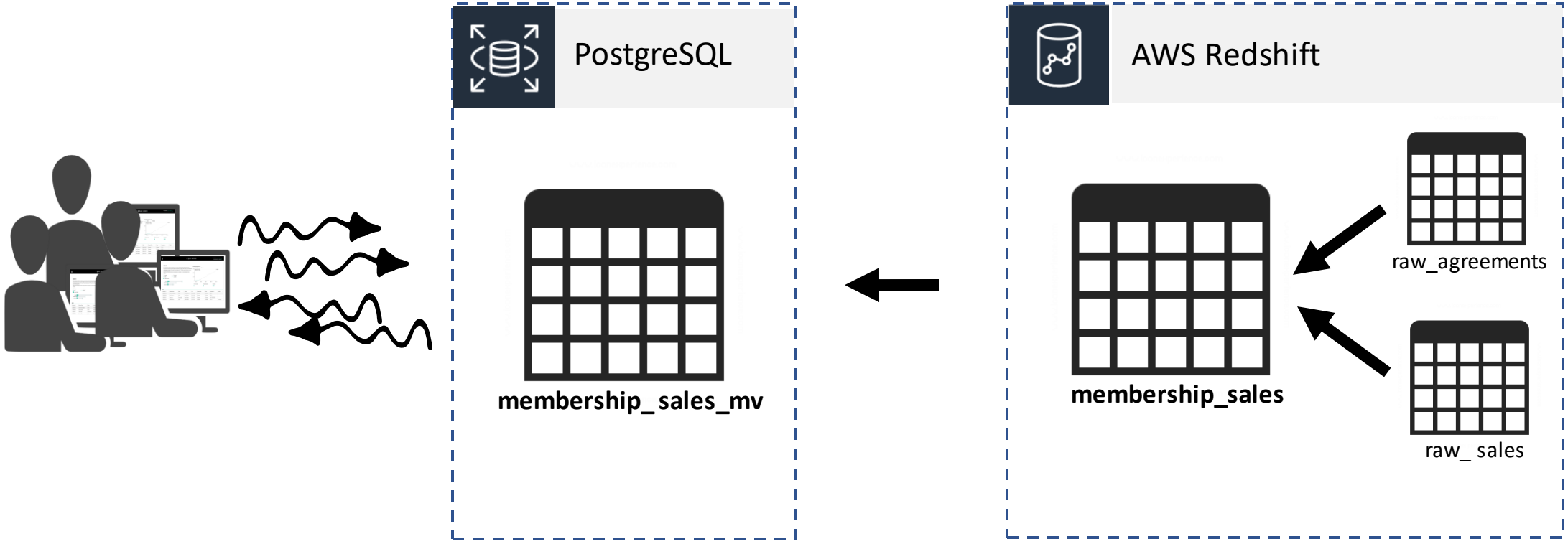
(Update: This blog post has been translated into Japanese)

When it comes to choosing a SQL-based database in AWS, there are many options. Sometimes it can be difficult to know which one to choose. For example, when would you use Amazon Aurora instead of Amazon RDS PostgreSQL or Amazon Redshift? To answer this question, you must first understand the nature of the data workload and then evaluate other factors such as the quantity of data and query access patterns.

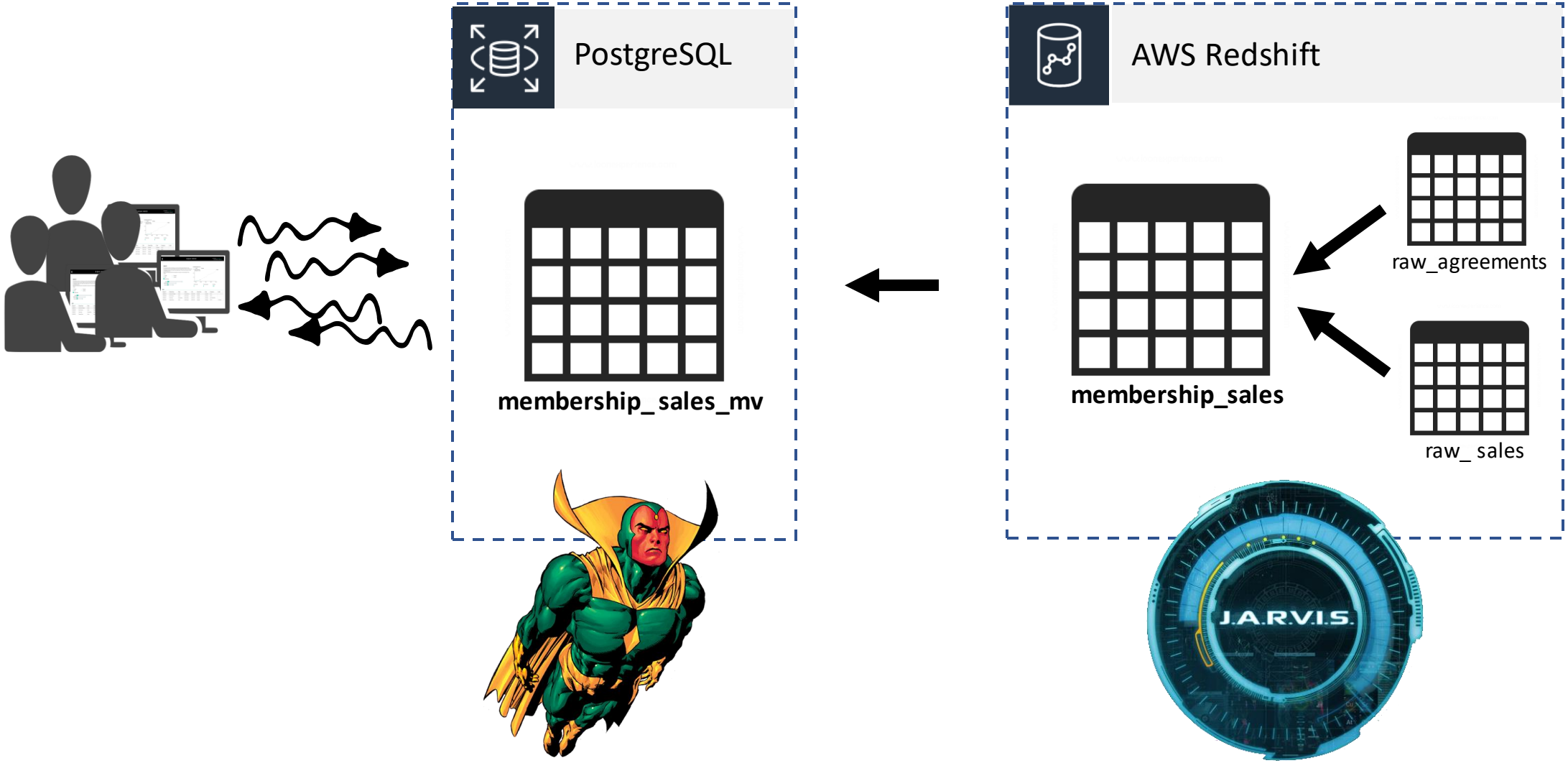
The design and capabilities of the different AWS services mean that each service has different strengths and excels at different workloads. This leads to trying to pick the right tool for the job, which can result in tradeoffs. But sometimes you don't want to compromise.

https://aws.amazon.com/blogs/big-data/join-amazon-redshift-and-amazon-rds-postgresql-with-dblink

Proposed Sales Reporting Architecture



Proposed Sales Reporting Architecture



Environment Set-up

✓ Create Redshift cluster

✓ Create PostgreSQL server (9.5+)

- RDS recommended
- For self-managed, install Postgres contrib package:
  - sudo yum install postgresql10-contrib.x86_64

✓ Networking (AWS)

- Co-locate in same Availability Zone
- Configure Security Group

Creating the Link

```
--1 enable the required extensions
CREATE EXTENSION postgres_fdw;
CREATE EXTENSION dblink;

--2 create the external server
CREATE SERVER jarvis
        FOREIGN DATA WRAPPER postgres_fdw
        OPTIONS (host 'REDSHIFT_ENDPOINT', port '5439',
        dbname 'REDSHIFT_DB_NAME', sslmode 'require');

--3 save redshift login to this external server
CREATE USER MAPPING FOR PG_USERNAME
        SERVER Jarvis OPTIONS
        (user 'RS_USERNAME', password 'RS_PASSWORD');
```

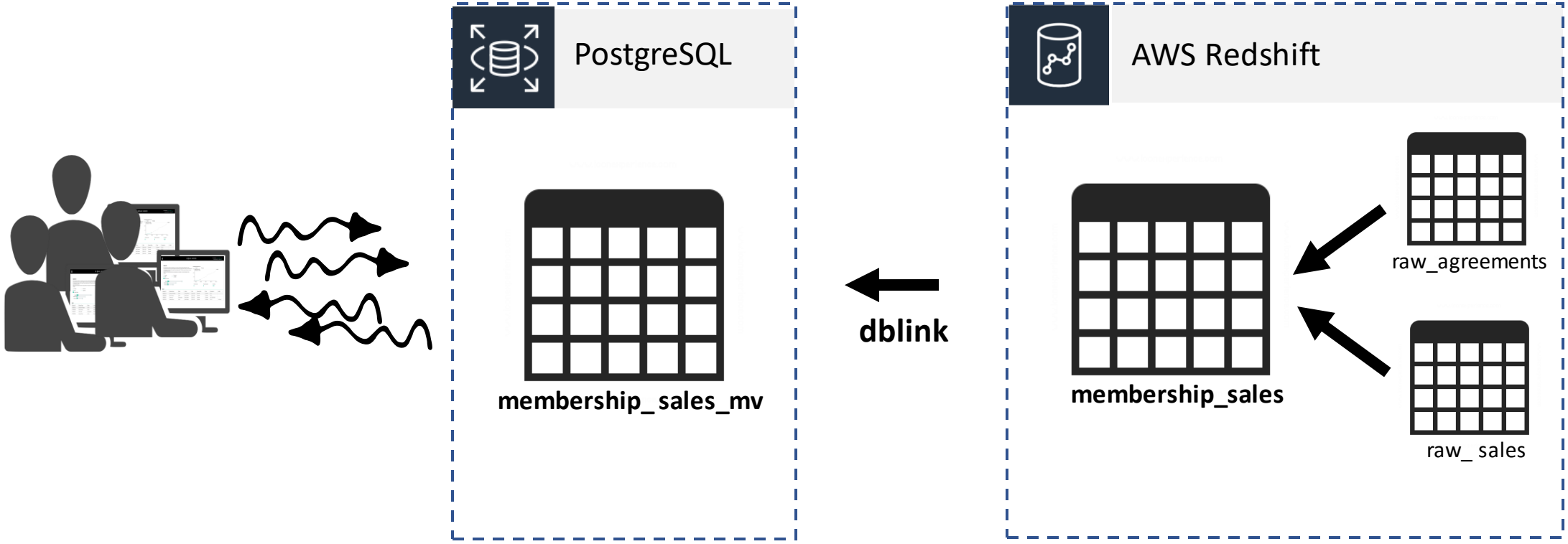
Running queries on PostgreSQL

```
SELECT *
FROM dblink('jarvis', $REDSHIFT$
   SELECT
      member_sales_id,
      member_id,
      sales_action,
      sales_action_date
   FROM
      rs_landing.raw_sales $REDSHIFT$)
AS sales_actions (
   member_sales_id varchar(50),
   member_id varchar(50),
   sales_action varchar(50),
   sales_action_date date
);
```

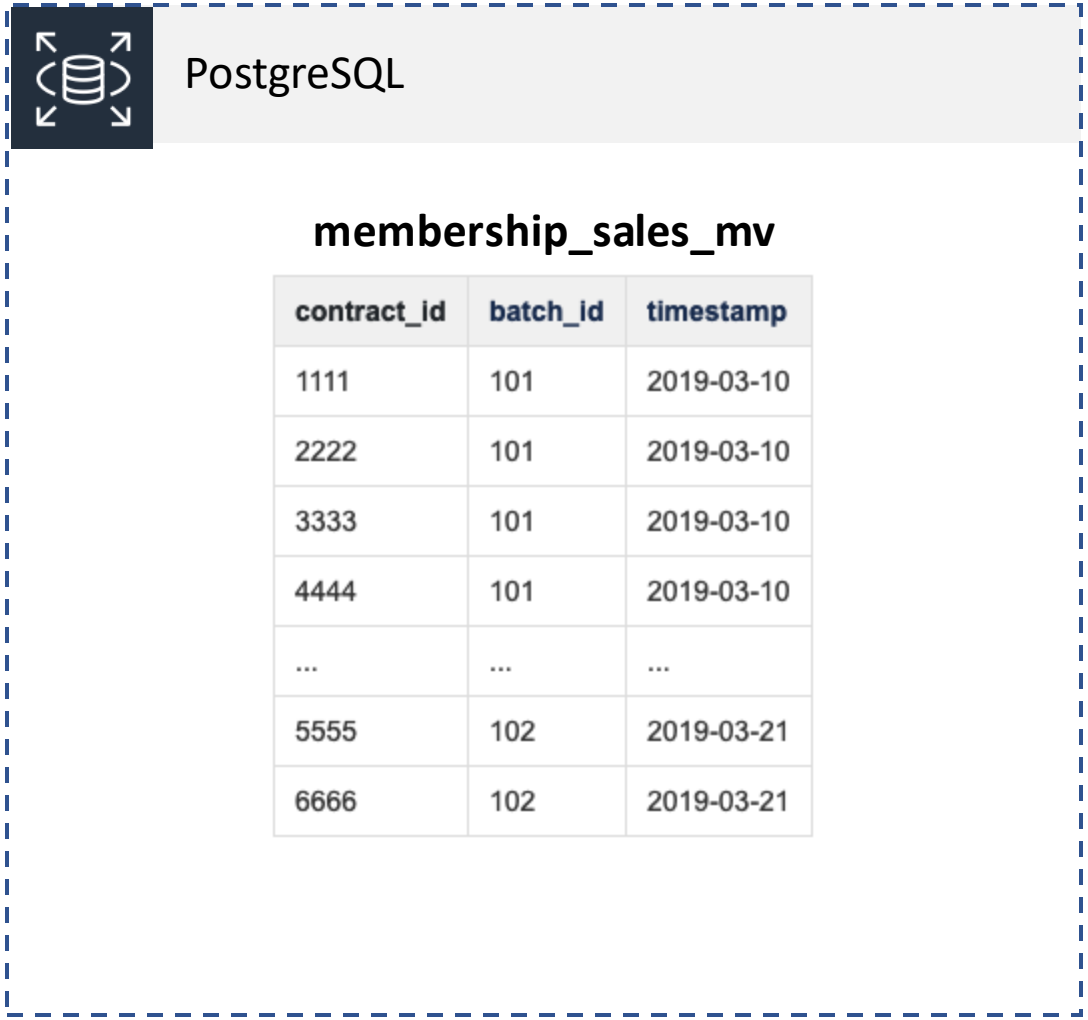
Leveraging a Materialized View

```
CREATE MATERIALIZED VIEW pg.membership_sales_copy AS
(
   SELECT *
   FROM dblink('jarvis', $REDSHIFT$
   SELECT
      member_sales_id,
      member_id,
      sales_action,
      sales_action_date
   FROM
      rs.membership_sales $REDSHIFT$
) AS membership_sales_copy (
          member_sales_id varchar(50),
          member_id varchar(50),
          sales_action varchar(50),
          sales_action_date date
);

REFRESH materialized VIEW pg.membership_sales_copy;
```

Sales Reporting Architecture



Materialized View Roadblock



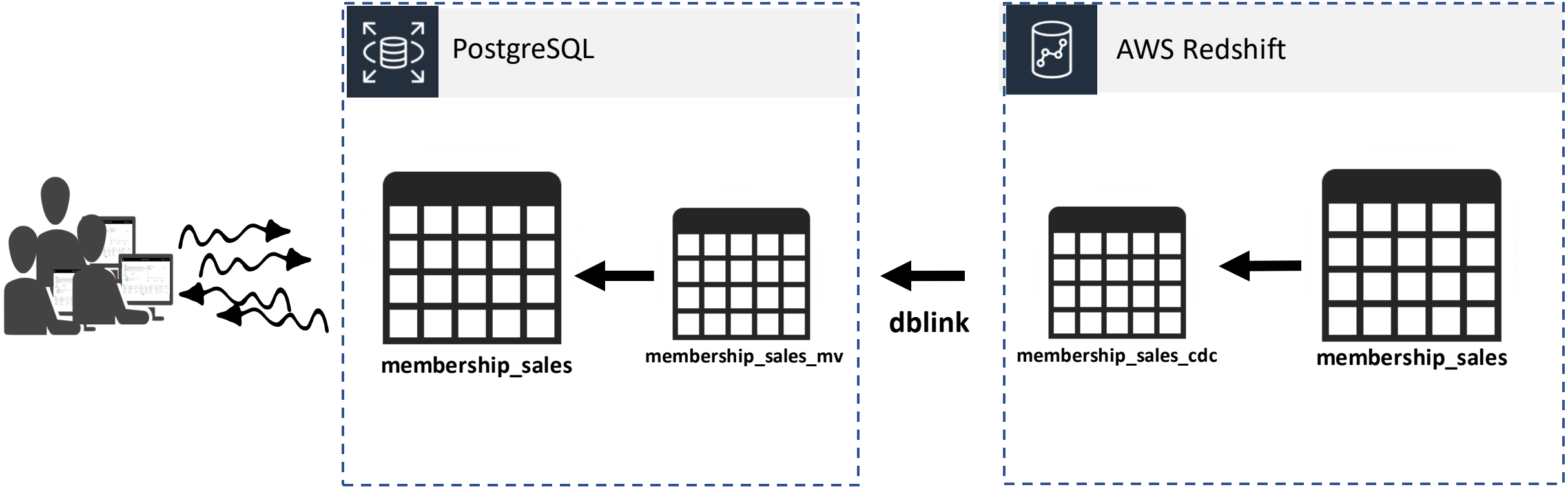
Change Data Capture for Large Tables

```
--Step 1: Create staging table in Redshift with last few hours of sales actions
--CREATE TABLE rs_landing.stage_sales_action
DELETE FROM rs.membership_sales_cdc

INSERT INTO rs.membership_sales_cdc
SELECT member_sales_id, member_id,  sales_action, sales_action_date
FROM rs.membership_sales
WHERE date >= ' $[?from_date]';

--Step 2: Refresh materialized view in Postgres
REFRESH materialized VIEW pg.membership_sales_mv;

--Step 3: Upsert logic to populate final table in Postgres from materialized view
--temp table to hold last batch
DROP TABLE IF EXISTS cdc_sales;
CREATE TEMP TABLE cdc_sales AS
SELECT * FROM  pg.membership_sales_mv;

--update changed records, member_sales_id as the key to identify a unique record
UPDATE pg.membership_sales ms
SET sa.member_id = s.member_id,
    ms.sales_action = s.sales_action,
    ms.sales_action_date = s.sales_action_date
FROM cdc_sales s
WHERE s.member_sales_id = ms.member_sales_id;

--delete the records we just updated from temp table
DELETE FROM cdc_sales s USING  pg.membership_sales ms
WHERE s.member_sales_id = ms.member_sales_id;

--insert new records not found in membership_sales
INSERT INTO  pg.membership_sales
SELECT * FROM cdc_sales;

--drop temp table
DROP TABLE cdc_sales;
```

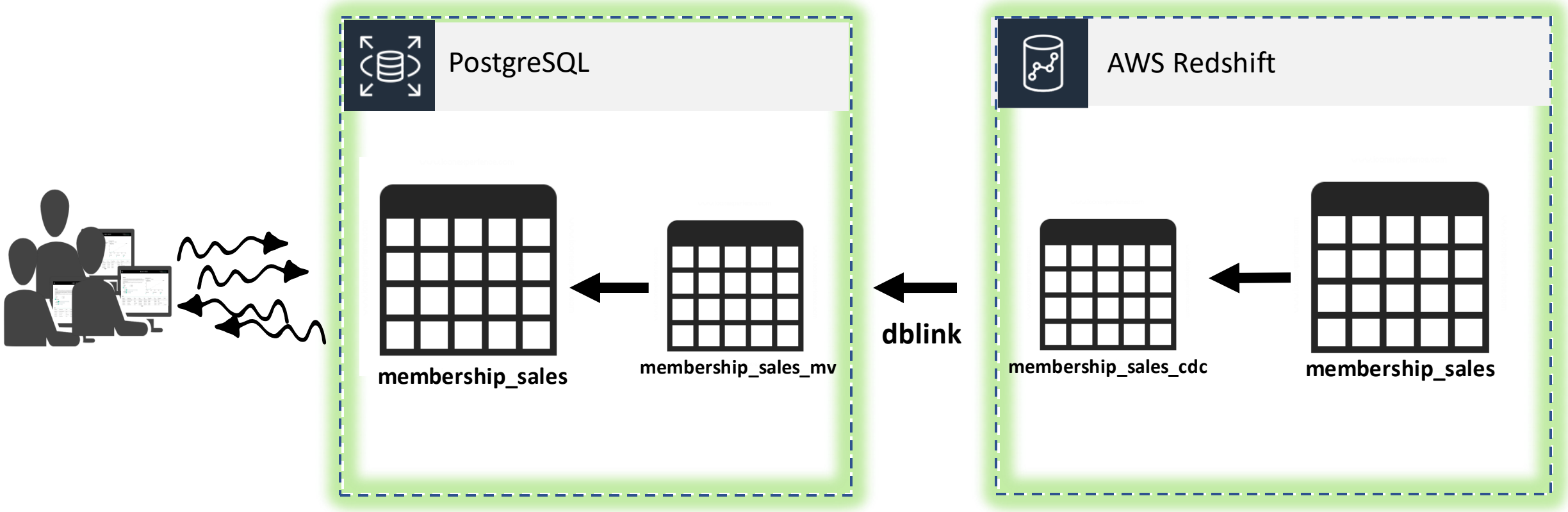
Redshift

PostgreSQL

Sales Reporting Architecture

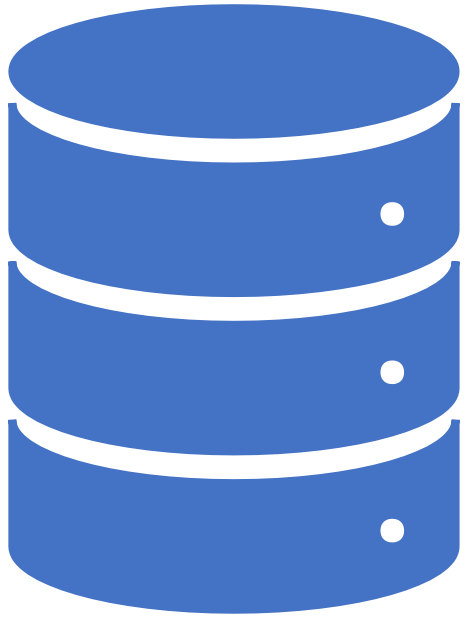


Sales Reporting Architecture **Solution**



Learnings & Notes

- Minimal maintenance on Postgres instance
- Won’t reflect source deletions
- Limited to a few tables
- Flexible for schema evolution



Looking to the Future…

- Make use of foreign table
- Front-end scaling with read-replicas
- Extensible to other datastores
- Event-based streaming architecture

More to Explore

Appendix F. Additional Supplied Modules

Table of Contents

Generic SQL Database Wrappers

Data Source	Type	License	Code	Install	Doc	
ODBC	Native		github			CartoD
JDBC	Native		github			
JDBC2	Native		github			
SQL_Alchemy	Multicorn	PostgreSQL	GitHub	PGXN	documentation	Can be used to acc
VirtDB	Native	GPL	GitHub			A generic

Specific SQL Database Wrappers

Data Source	Type	License	Code	Install	Doc	
PostgreSQL	Native	PostgreSQL	git.postgresql.org		documentation	
Oracle	Native	PostgreSQL	github	PGXN	website	
MySQL	Native		github	PGXN	example	
Informix	Native	PostgreSQL	github			
Firebird	Native		github	PGXN		
SQLite	Native		github			
SQLite	Native	PostgreSQL	github	PGXN	README	An FDW
Sybase / MS SQL Server	Native		github	PGXN		
MonetDB	Native		github			

NoSQL Database Wrappers

Data Source	Type	License	Code	Install	Doc
BigTable or HBase	Native Rust Binding (RPGFFI)	MIT	Github		
Cassandra	Multicorn	MIT	Github	Rankactive	
Cassandra2	Native	MIT	Github		
Cassandra	Multicorn	PostgreSQL	Github		
ClickHouse	Multicorn	BSD	Github		READM

Q&A

We Are Hiring

Email ITCareers@Equinox.com

- Head of Engineering
- React Native Engineer
- Sr. React Native Engineer
- API Engineer
- SDET Java - Architect
- SDET Javascript - Architect
- SDET Java
- SDET Javascript
- Sr. UX Researcher

Equinox Tech Blog http://tech.equinox.com/