Building a Scalable Financial Data Store

Liwei Mao

Nov 11, 2019

Button

Agenda

- **About me**
- A data engineering challenge
- Mistakes we made
- Our solution (5 data design choices)
- Did we make it better?

About me

- BA Math & MA Stats from Columbia
- Started in BI, transitioned to Data/Platform Eng.
- Team Lead at Button, building a mobile commerce platform.
- Obsessed with Super Smash Bros, and classic N64 controllers

Button

Agenda

- About me
- **A data engineering challenge**
- Mistakes we made
- Our solution (5 data design choices)
- Did we make it better?

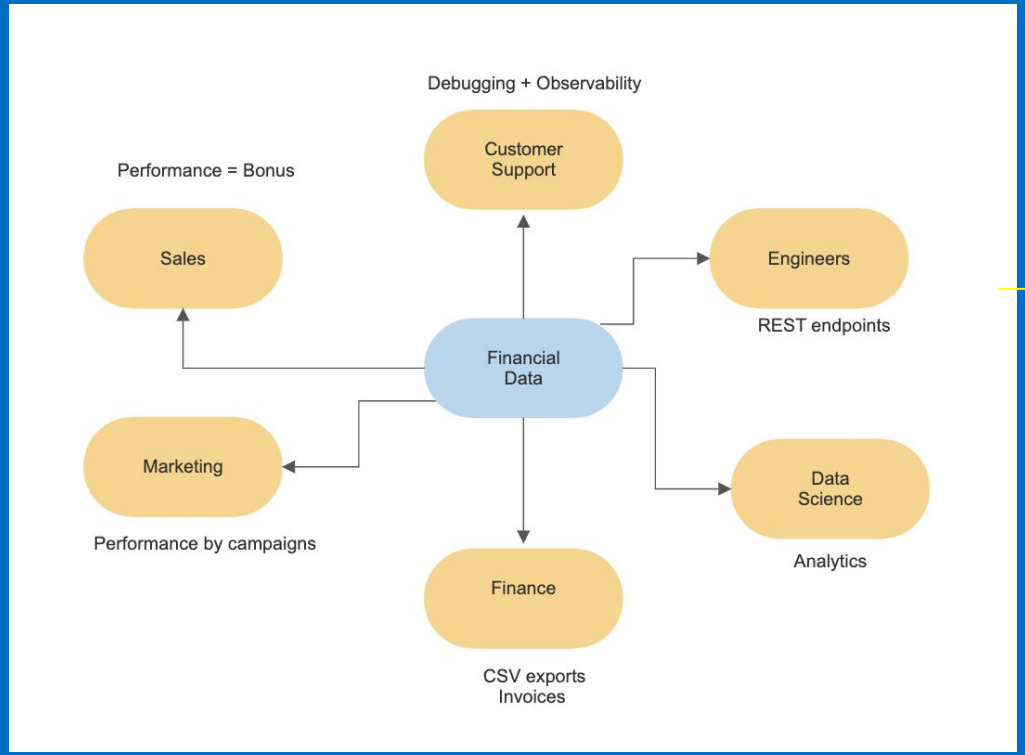
As Data Engineers

- On the hook to provide a “source of truth” for downstream consumers
- Data needs to be accurate, timely, and “easy to use”
- Tough when your users have different use case for the data

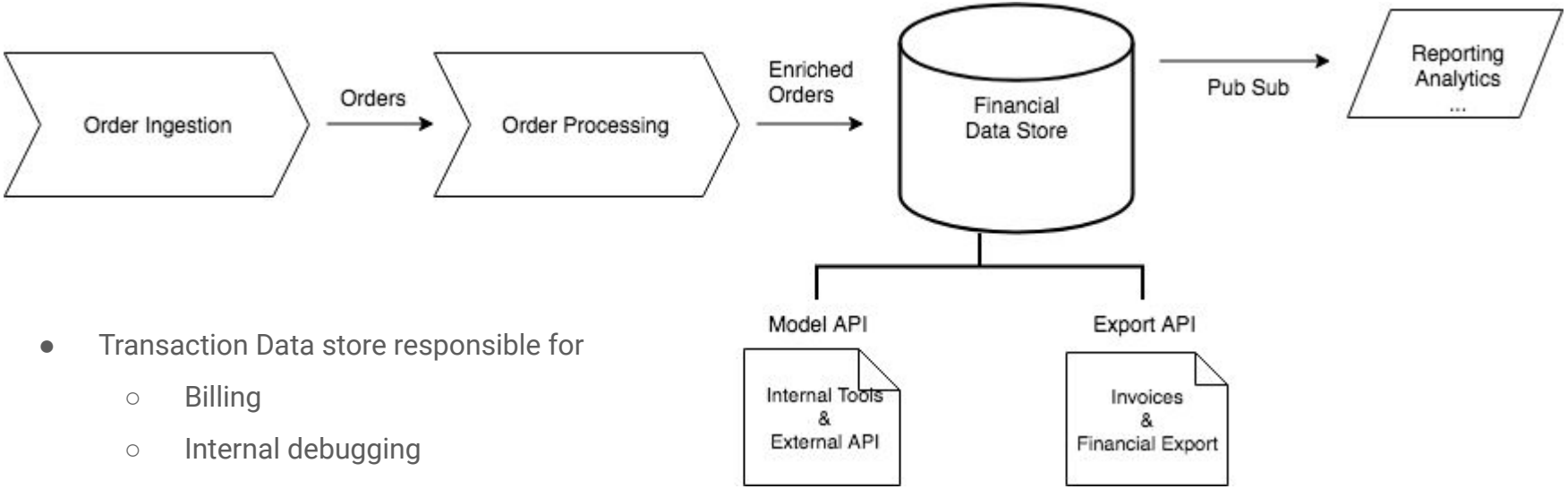
Button

A challenge for $$$ data

- Everyone's reliant on financial data, but they all have different access patterns



- Aggregate Performance
- CSV Exports
- List Endpoints
- CRUD endpoints
- Custom Analytics



- Transaction Data store responsible for
  - Billing
  - Internal debugging
  - Downstream services
    - Reporting
    - Analytics Warehouse

Mix of OLTP and OLAP access patterns

- OLTP (Online Transactional Processing)
  - Every write to DB = $$ exchanging hands
  - No downtime, low latency writes
  - Accuracy is crucial
- OLAP (Online Analytical Processing)
  - Monthly financial CSV exports & list endpoints
  - Easy aggregation
  - Slice and dice over arbitrary set of columns

Button

Agenda

- About me
- A data engineering challenge
- **Mistakes we made**
- Our solution
- Did we make it better?

Mistake: Overwriting transactions with the latest state

CX sees

Order Total $200

2 days later, he see

Order Total $2000

What happened?

...CX can't tell because there's no history

CSV Exports

Finance

	A	B	C
1	**Order ID**	**Status**	**Total**
2	order-1	pending	$250
3	order-2	pending	$100
4	...	...	
5	...	...	...
6			$500,000,000

Downloaded CSV file on Jan 1

	A	B	C
1	**Order ID**	**Status**	**Total**
2	order-1	adjusted	$135
3	order-2	pending	$100
4	...	...	
5	...	...	...
6			$350,000,000

Re-pulled export on Jan 5

- Export looks different when it's downloaded on different days
- Can't tell what changes were made?
- Exports take so long…

Agenda

- About me
- A data engineering challenge
- Mistakes we made
- **Our solution** (5 data design choices)
- Did we make it better?

Data Redesign

How should we redesign the data to better serve users?

Button

5 Data Design Choices

1. **Immutable** - Records are never changed, only inserted

Why Immutable?

- Biggest pain point
- Able to track changes over time (data lineage)
- Financial data should never be mutable
  - useful for auditing
  - state is reproducible at any point in time
  - allows for correction in next accounting period

Immutable

Immutable event log

user books a hotel

user extends to 10 nights

```
1 Date       | Event Type   | Total    | Type
2 2019-07-01 | Order Create | +$  200  | Hotel booking
3 2019-07-03 | Order Adjust | +$ 1800  | Hotel extension
```

What CX observed was no fluke!

July 1st

Order Total $200

July 3rd

Order Total $2000

5 Data Design Choices

1. Immutable
2. **Deltas for Easy Aggregation** - represent amounts in “deltas”

Digiday, 2017

Use deltas for easy aggregation

```
1  Date       | Event Type   | Total
2  2019-07-01 | Order Create | +$  500
3  2019-07-01 | Order Adjust | -$  300
4  2019-07-01 | Order Delete | +$  250
5  ------------------------------------
6                             +$  450
```

Button

Deltas make aggregation faster & simpler

See total commissions by day

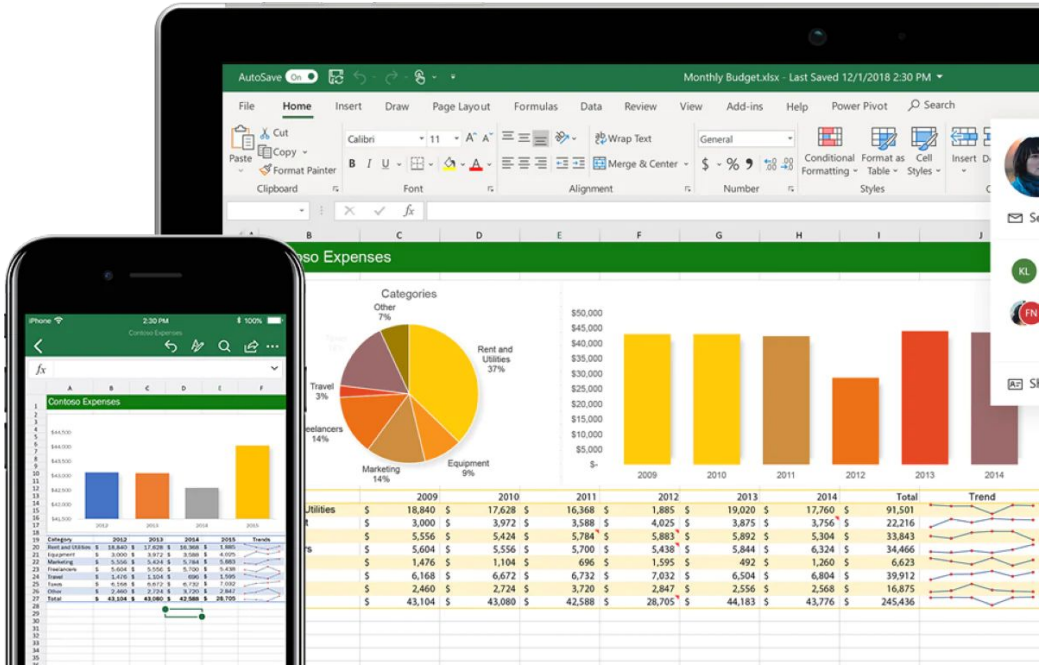
Before

```
SELECT purchase_date || created_date AS dt,
       sum(order_total) * 100 AS GMS,
       sum(CASE WHEN organization_type = 'merchant' THEN total END),
       sum(CASE WHEN organization_type = 'publisher' THEN total END)
FROM
  (SELECT DATE(order_purchased_time) AS purchase_date,
          order_total,
          status,
          total,
          row_number() over (partition BY id
                             ORDER BY modified_time DESC,
                                CASE WHEN event_type IN ('order_finalize', 'order_delete') THEN 1
                                     WHEN event_type IN ('order_adjustment') THEN 2 ELSE 3 END) AS row_num
   FROM transaction
   WHERE organization_type IN ('publisher',
                               'merchant')
     AND (created_date BETWEEN %(start)s AND %(end)s))
WHERE row_num = 1
  AND status != 'declined'
GROUP BY 1;
```

After

```
SELECT attribution_date,
       sum(order_total) as GMS,
       sum(merchant_commission),
       sum(publisher_commission)
FROM commission_event
GROUP BY 1;
```

Your Marketing & Finance folks will appreciate it



microsoft excel stock image

Choice of “delta” vs “latest state”

Benefit of Delta

- Easy aggregation
- A single service responsible for computing deltas
- “Atomic” - self contained description of the change
- Events can arrive out of order, and end state will be eventually consistent

With Latest State

- Greater tolerance for missing events, later states will overwrite incorrect earlier states

5 Data Design Choices

1. Immutable
2. Deltas for Easy Aggregation - represent amounts in “deltas”
3. **Denormalized** - few tables, lots of dimensions

Digiday, 2017

Why Denormalized?

More OLAP use cases than OLTP.

OLAP use cases - large # of records

- Marketing - Campaign analysis
- Finance - Billing Exports & Invoices
- Data team - Analytics
- Partners - API for historical data

OLTP use cases - single record

- Customer Support - Debugging individual orders
- Inserting events

Hybrid Performance Approach

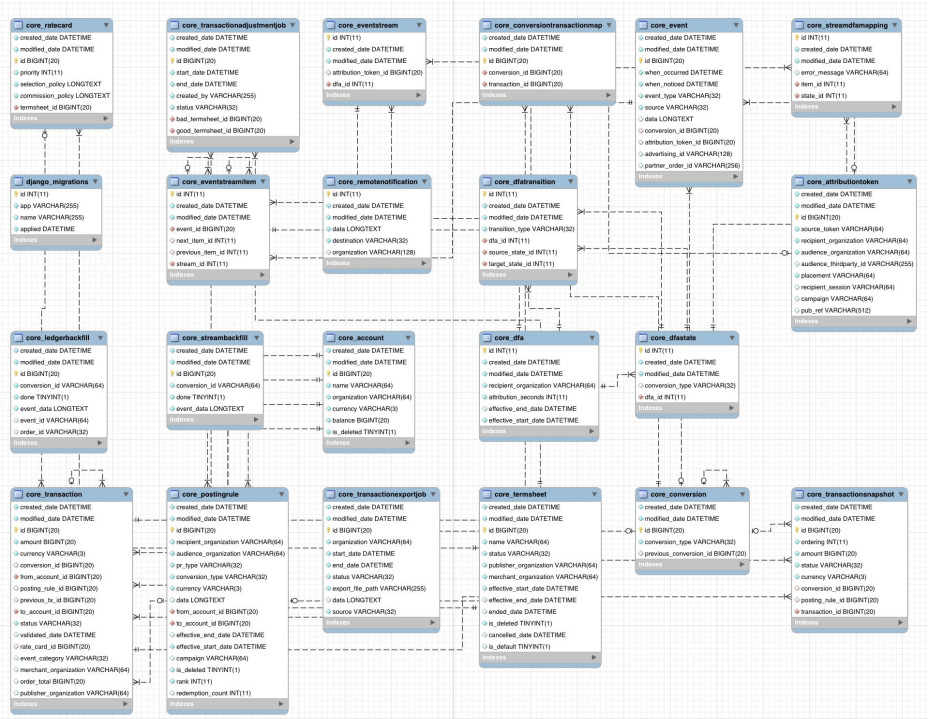
- Use Postgres DB
- Denormalized Data

Hybrid in the sense that data format is optimized for querying over historical time ranges yet DB is a traditional OLTP database.

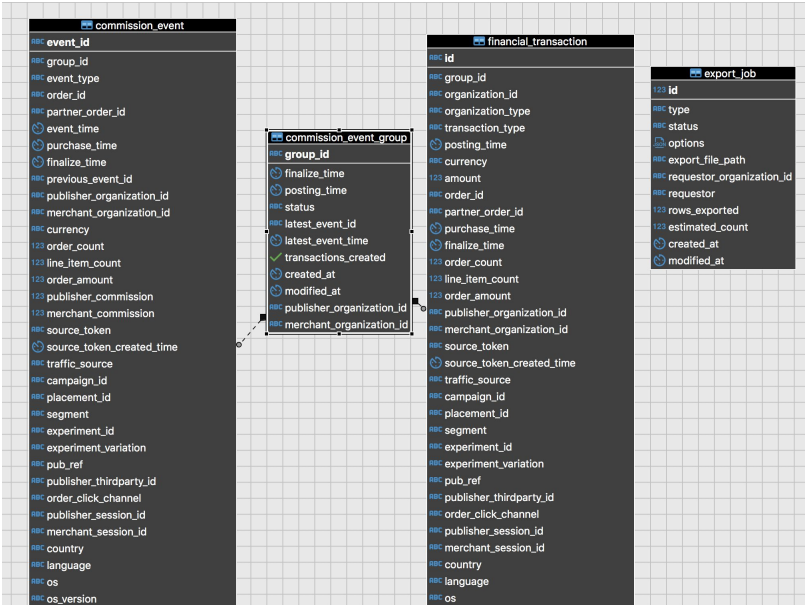
Denormalized Data

For faster performance with CSV Exports and aggregations

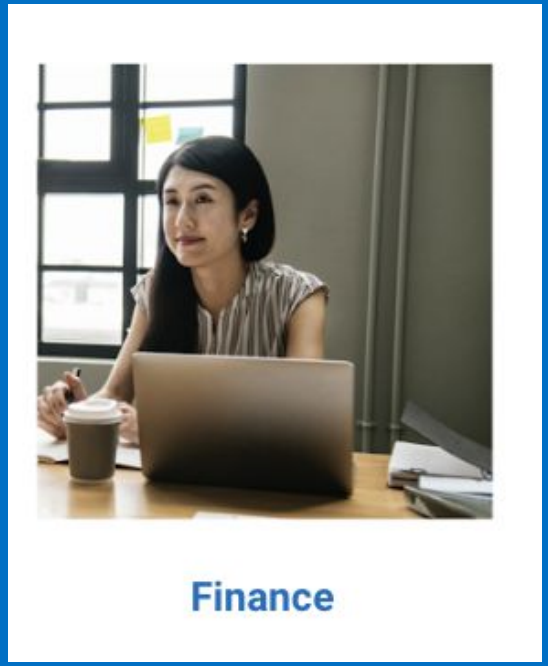
Previous Financial Data Store



New Data Store - denormalized



CSV Exports are 8x faster!



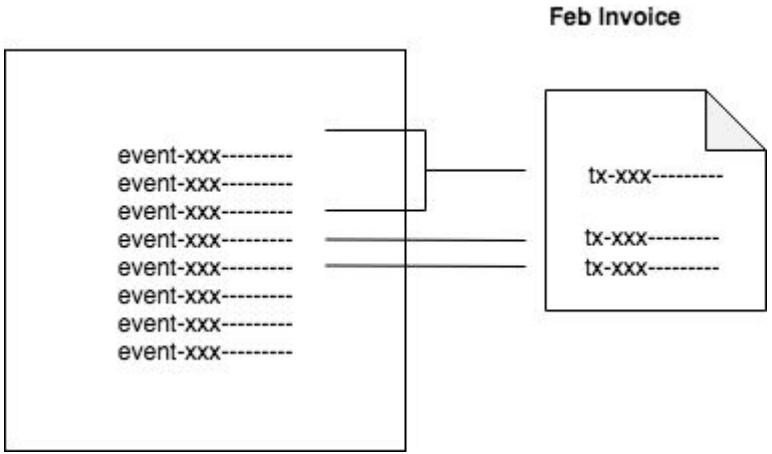
Button

5 Data Design Choices

1. Immutable
2. Deltas for Easy Aggregation - represent amounts in “deltas”
3. Denormalized - few tables, lots of dimensions
4. **Separate record keeping for billing**

Digiday, 2017

Why keep separate records for billing?



- Need stable tracking of which events fit into each invoice
- Enable later adjustments
- Allow changes in billing logic
  - may bill on events vs orders
  - may bill per customer vs per order
  - may bill weekly vs monthly

Keep separate records for billing

Immutable Event Log

If a user makes an order of $100, and later makes a partial return of $50.

```
Billing Date | Order ID | Event Date | Event Type   | GMS
2018-02-02   | order1   | 2018-01-02 | ORDER_CREATE | 100
2018-02-02   | order1   | 2018-01-02 | ORDER_ADJUST | -50
2018-03-02   | order2   | 2018-02-01 | ORDER_CREATE | 30
```

Financial Transaction (Feb-2018)

Financial Tx export for the publisher contains the total GMS/Publisher Commission for orders that gets billed in Feb.

```
Billing Date | Order ID | GMS
2018-02-02   | order1   | 50
```

Product/Service rendered

Marketing / Sales / Customer Support

Invoicing

Billing Department

5 Data Design Choices

1. Immutable
2. Deltas for easy aggregation
3. Denormalized
4. Separate record keeping for billing
5. **Self Heal** - programmatic detection & adjustment

Digiday, 2017

Self-Heal - programmatic detection & adjustment

- Immutable data helps with this
- So does having separate records for billing
- Limiting points of failure

Example:

- Orders that were processed “late”, that didn’t make it into the last billing cycle, should be automatically added to the next cycle
- Automatic checks of billing records (immutable) against order event records (also immutable)

What we've learned...

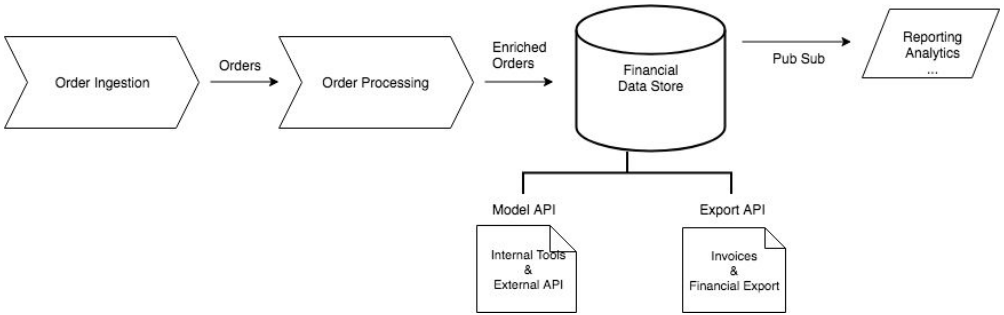
Button

Stable ID & Ordering

Use stable ID & ordering throughout your procession pipeline

- **Ordering (seqn) and Event ID should be set as upstream as possible in the order pipeline, and carried all the way downstream.**

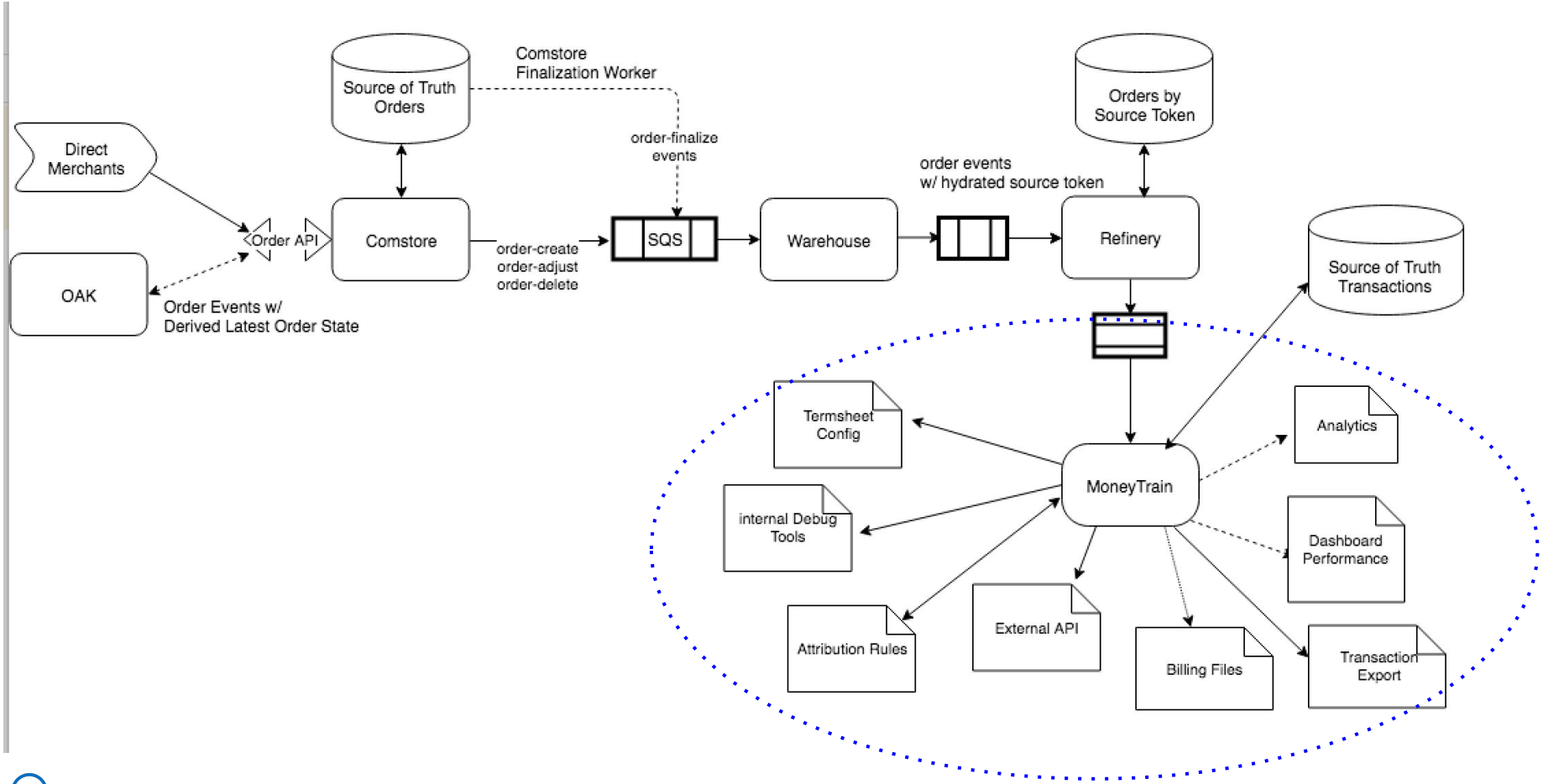
- **Good for debugging**

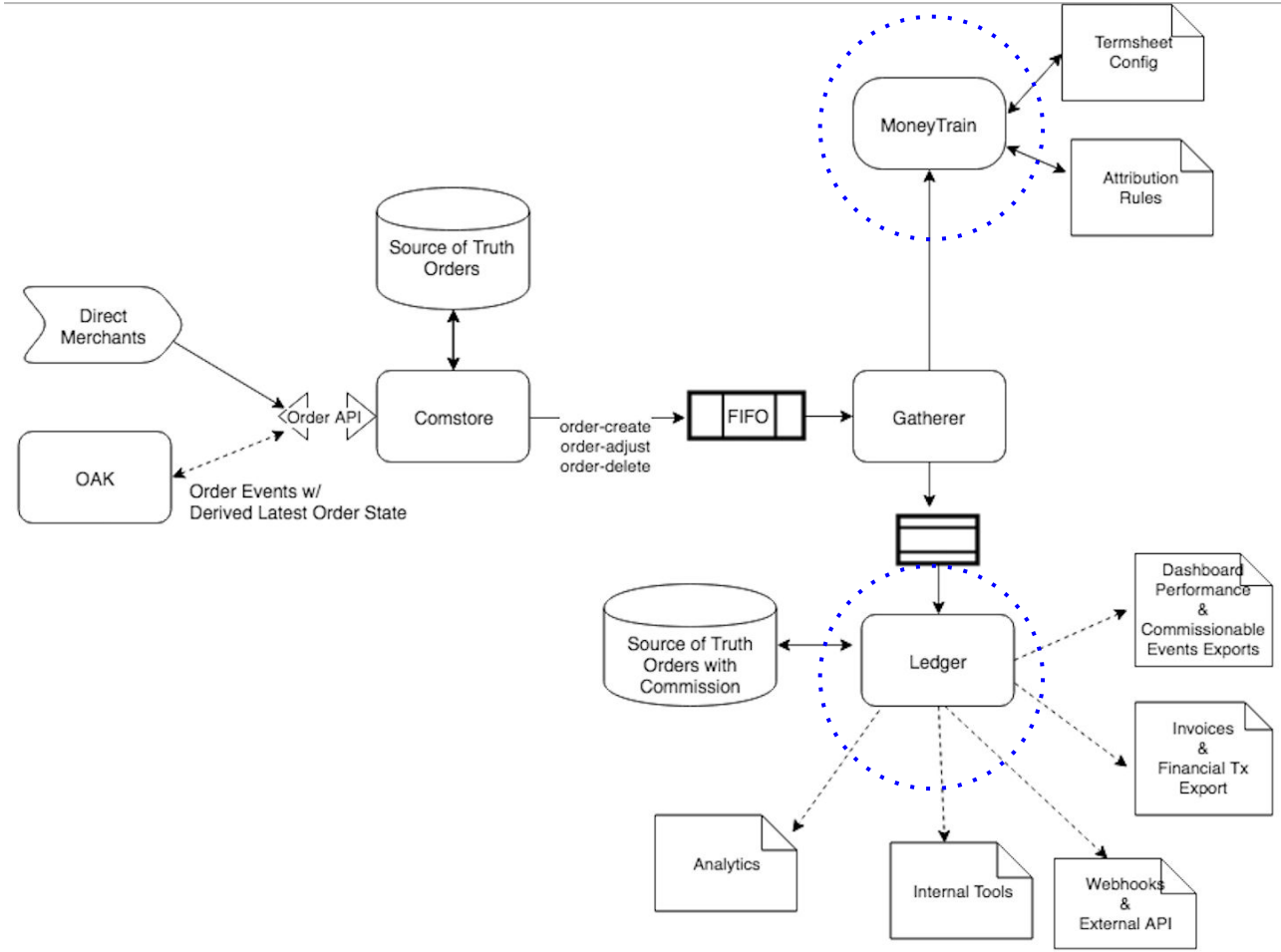


Quirks of Financial Data

- **Dates really matter.** And there are a lot of them
- Sample Dates:
  - Order Purchase Date - Original Time of purchase
  - Order Ingestion Date - First order event seen. When order is “created” in your system
  - Order Event Ingestion Date - When order event is ingested
  - Order Finalize Date - After which order can no longer be modified
  - Billing Date - When the event/order is invoiced
- Changing, or adding dates later is expensive -- it changes how you aggregate
- **Avoid floats**
- **Double-Entry doesn’t matter**
  - More applicable for days of paper ledger and manual entries

Rebuild Process...


Source of Truth
Orders
Comstore
Finalization Worker
Orders by
Source Token
Direct
Merchants
order-finalize
events
order events
w/ hydrated source token
Order API
Comstore
SQS
Warehouse
Refinery
Source of Truth
Transactions
OAK
order-create
order-adjust
order-delete
Order Events w/
Derived Latest Order State
Termsheet
Config
Analytics
MoneyTrain
internal Debug
Tools
Dashboard
Performance
Attribution Rules
External API
Billing Files
Transaction
Export


Termsheet Config
MoneyTrain
Attribution Rules
Source of Truth Orders
Direct Merchants
Order API
Comstore
order-create
order-adjust
order-delete
FIFO
Gatherer
OAK
Order Events w/
Derived Latest Order State
Dashboard Performance & Commissionable Events Exports
Source of Truth Orders with Commission
Ledger
Invoices & Financial Tx Export
Analytics
Internal Tools
Webhooks & External API

Learnings from the migration process

- Validate by comparing the full dataset
- Anticipate performance optimizations to account for immutability
- Backfilling even a single field can be painful. Include all the fields you can from the get-go
- Have a rollback strategy if things go south

What did we improve?

Button

- CSV Exports are 8x faster
- CX team can view history of how an order is commissioned and adjusted
- Aggregation is simpler for data scientists, Excel users, and streaming aggregation jobs
- Data lineage via immutability enabled programmatic detection and correction

Extensions

What may change in the future?

- Analytics - additional dimensions
- Sales team - different pricing models, more billable events (not just orders)
- Updates to billing
- Performance - Explore alternative storage engines

Questions?

Button

How Button Works