databricks

Databricks for Practitioners

Lakehouse Optimisation

Jonathan Choi - Product Specialist

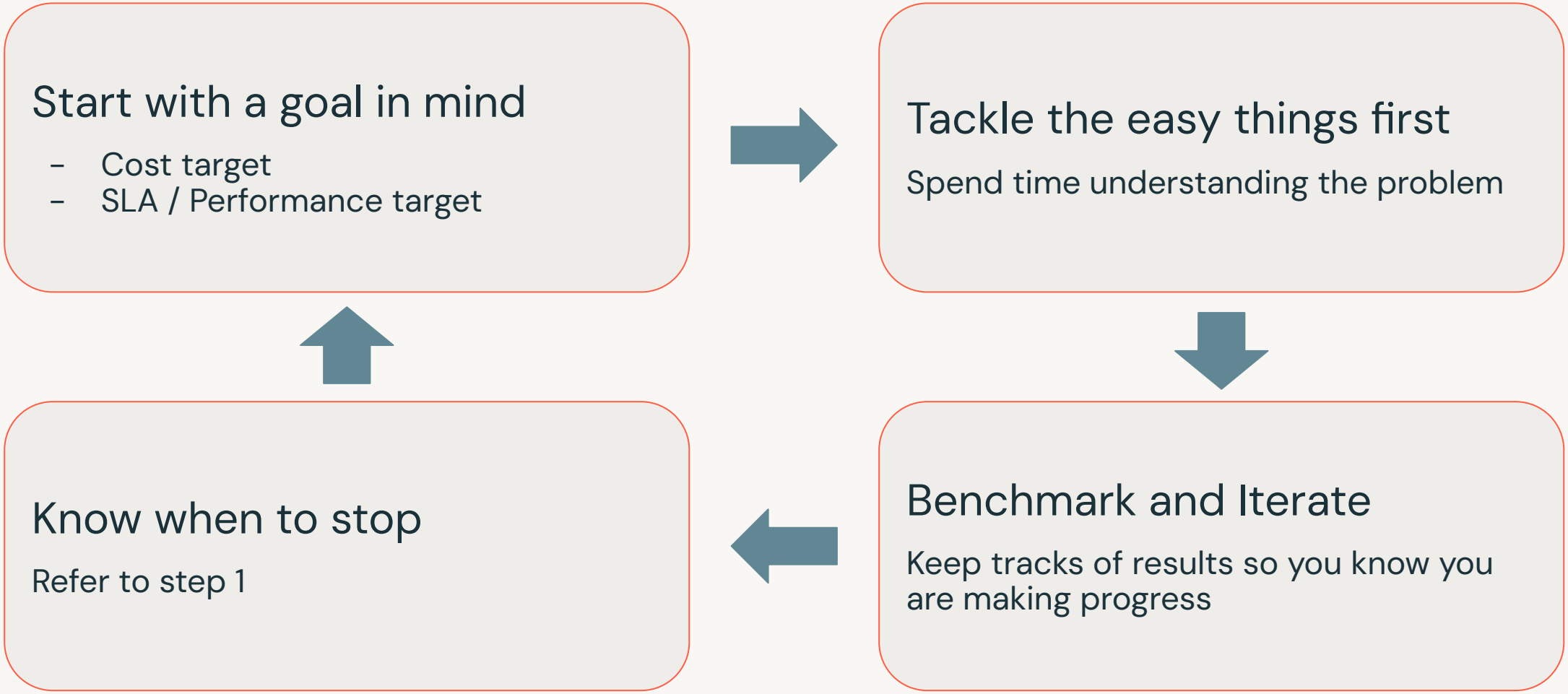
Housekeeping

- This presentation will be recorded and we will share these materials after the session.
- There are no hands-on components so you only need something to take notes.
- Use the Q&A function to ask questions.
- Please fill out the survey at the end of the session so we can improve our future sessions.

How do you think about optimisation?

Optimise only when necessary

You know that saying about premature optimisation...



Different use case have different goals

Setting your target up front is the key to success

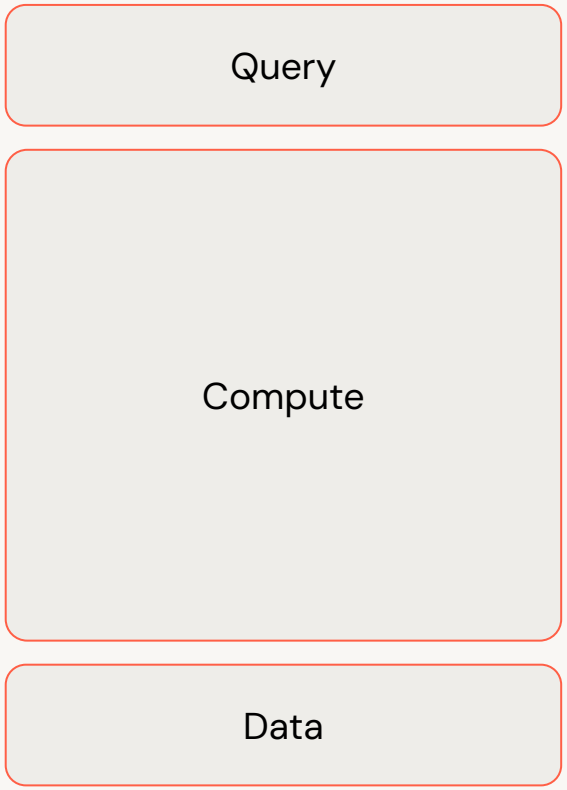
1 ETL/ELT	Low cost as data volume scales Speed less important
2 Exploratory Analysis	Low cost as data volume scales Low latency for quick analyses
3 Business Intelligence	Low latency and high concurrency as # users scale

Focus on metrics that matter to you

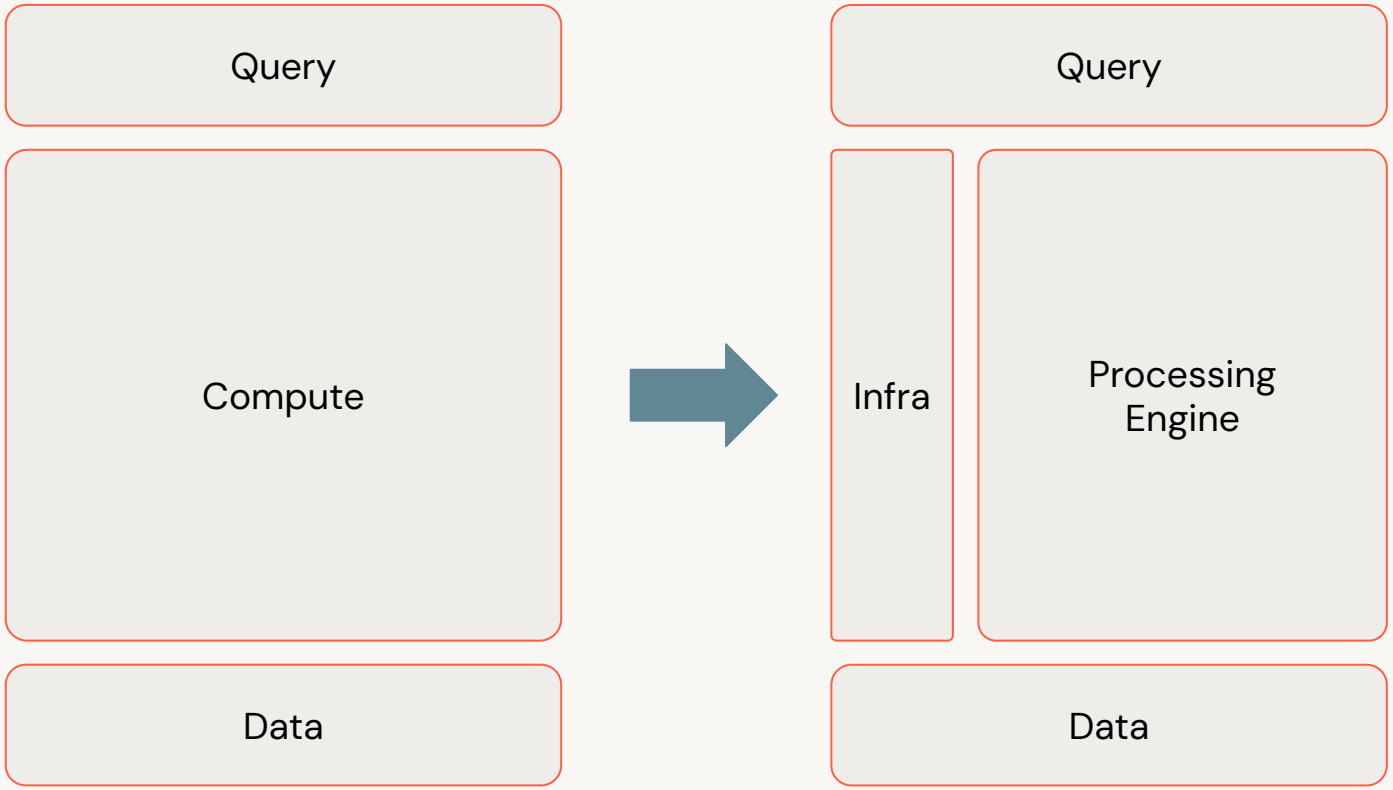
More metrics don't make things better

- Pick metrics that are relevant to you
  - Daily Cost ($) of your warehouse to support your users
  - P99 Query duration of your warehouse, i.e. Are you meeting SLAs?
- Quantify the outcome
  - How do you put a value on performance?
  - What does 5% improvement in performance mean to you? How much are you willing to pay for it?

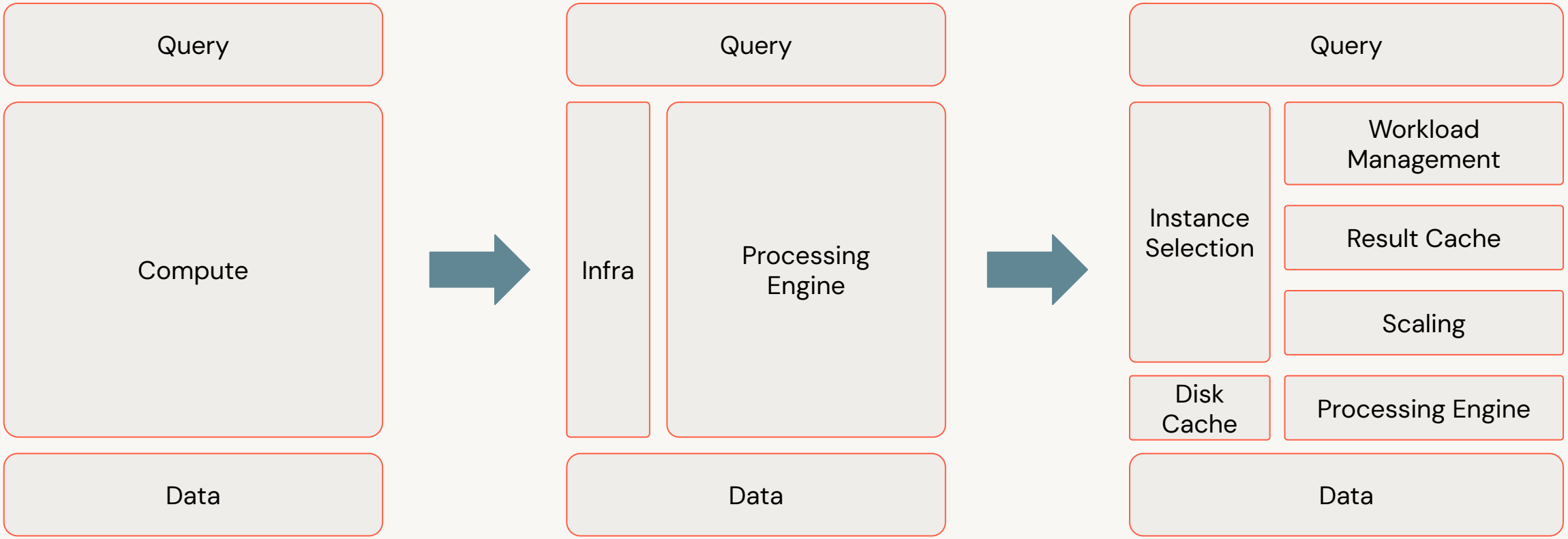
Understanding the journey of a query



Understanding the journey of a query

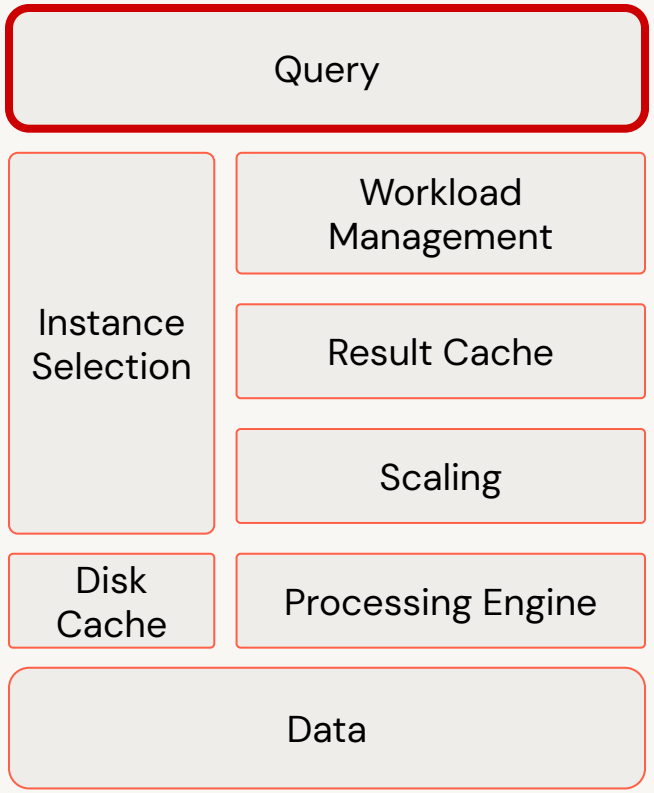


Understanding the journey of a query



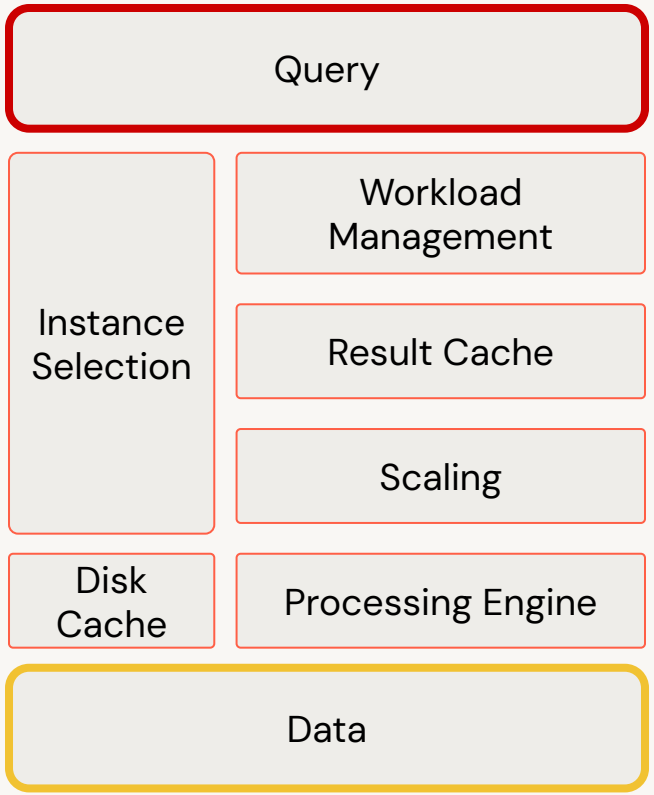
How hard is it to optimise?

Efficient queries run fast but writing good queries is difficult and you need to optimize a query at a time

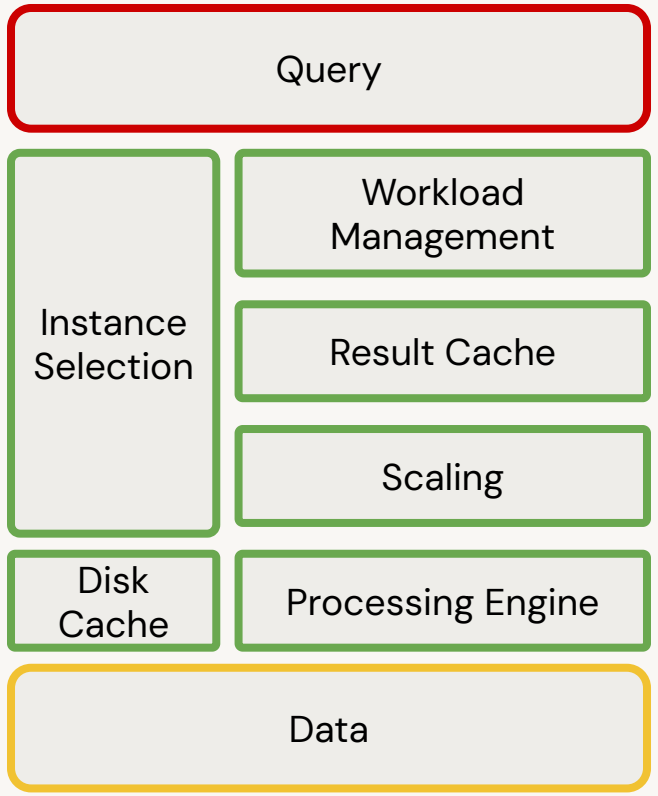


How hard is it to optimise?

Getting the data right helps every queries that use that tables and there are general rules of thumb to follow



How hard is it to optimise?



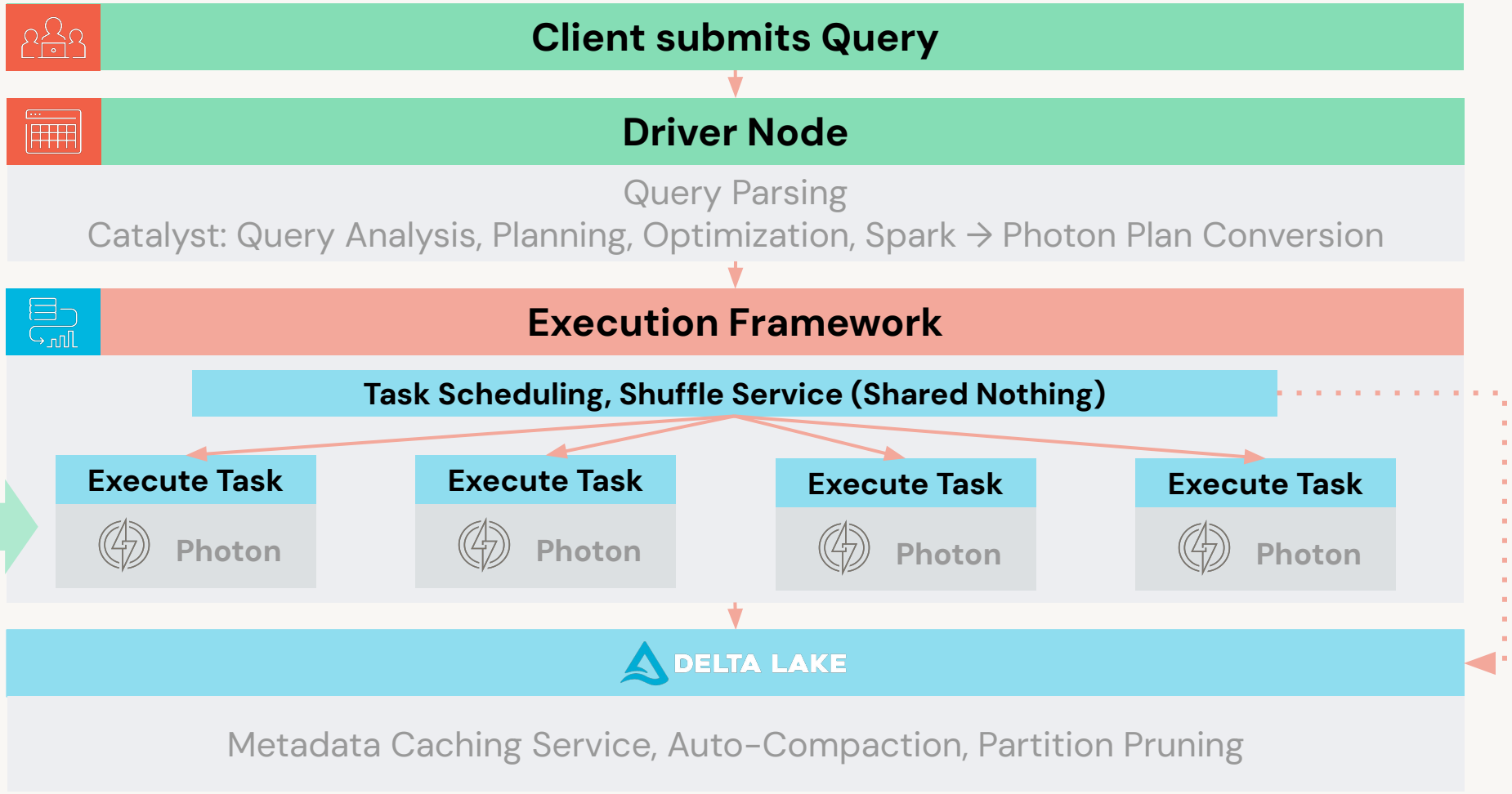
Setting up the right compute platform / warehouse helps all workloads and changing configs and settings is easy

How to optimise compute?

Photon – Speeding up data processing

Next generation query engine

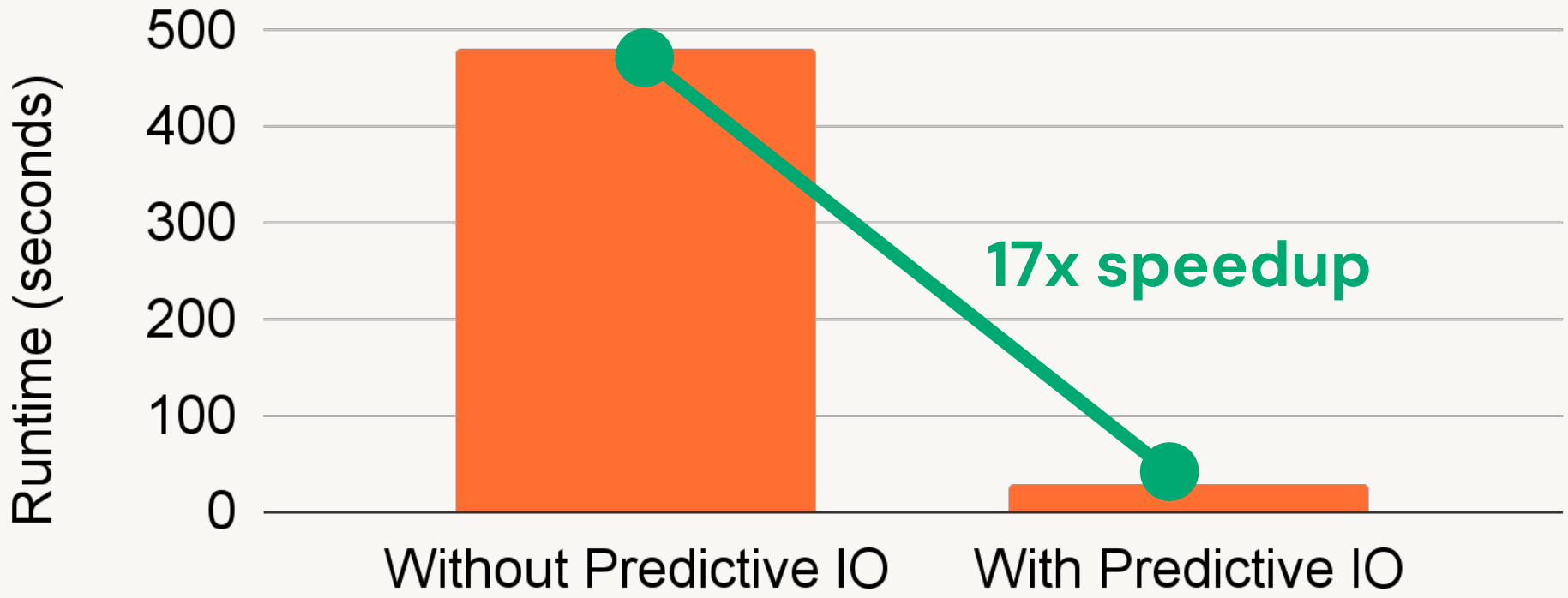
Execution Engine



Predictive I/O – Speeding up point queries

Let the compute engine determine the best way to fetch data

```
SELECT protoBase64 FROM query_profile_protos WHERE id LIKE '204ff749-dc88-4b0a%'
```

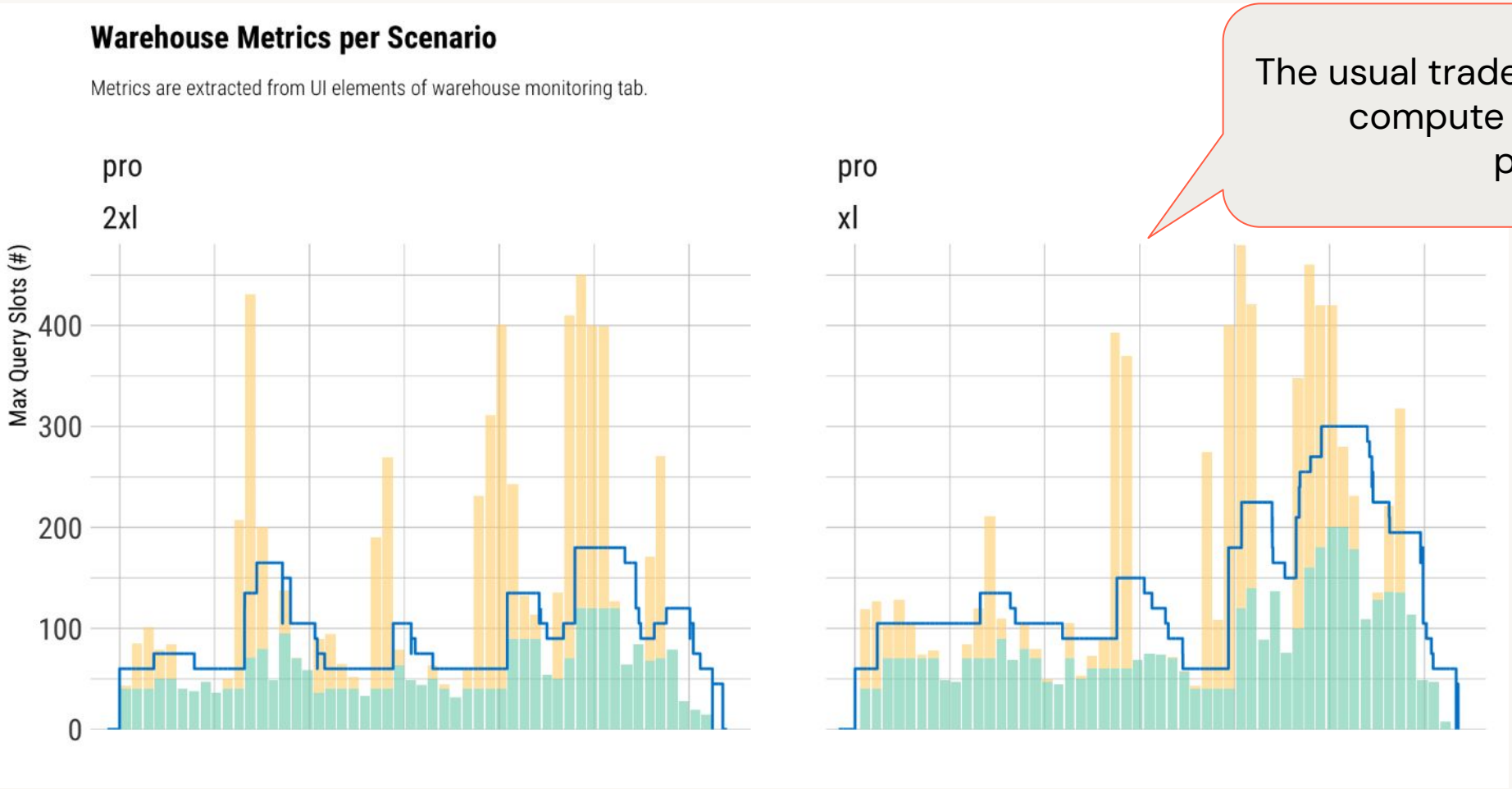


Compute Sizing

A balance between query performance and concurrency requirement

- Cluster Size (2XS ⇔ 4XL)
  - Larger cluster for larger queries and tables
- # of Clusters (Min ⇔ Max #)
  - More cluster for more concurrent queries
- Monitor Query History to find the right fit
  - Too many queries in queue = more clusters
  - Queries taking too long = larger clusters

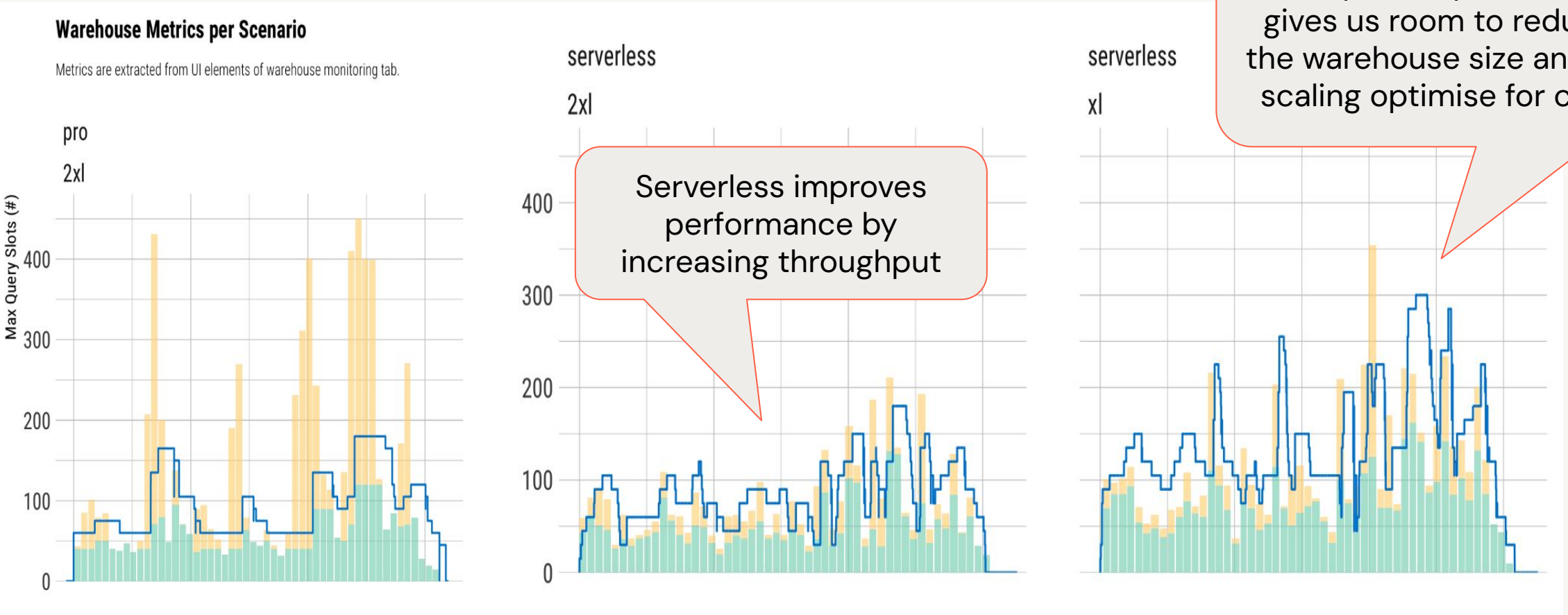
Cluster Sizing - Example



The usual trade off we make between for compute is between cost and performance

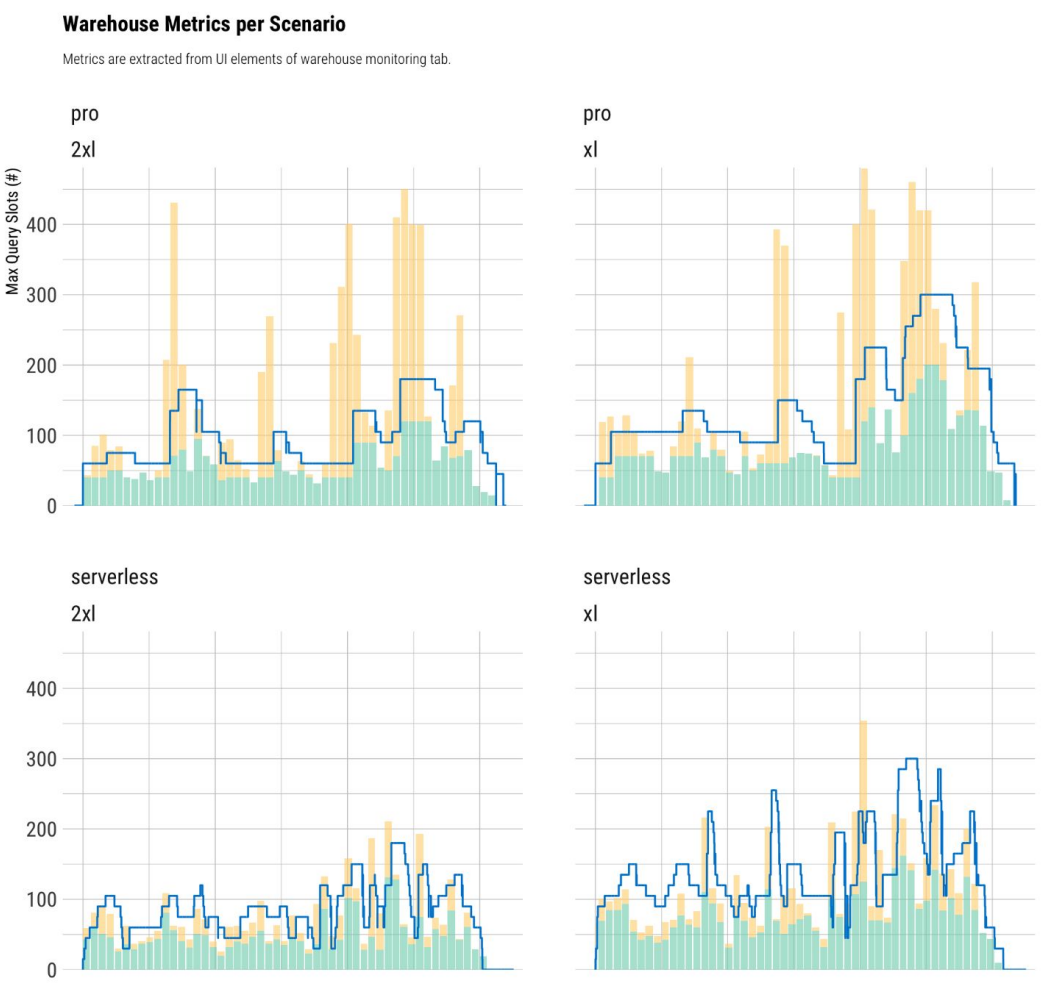
Serverless – Shifting the paradigm

How to fundamentally move your price-performance profile



Serverless – Shifting the paradigm

How to fundamentally move your price-performance profile



Serverless enables to move the price performance profile entirely rather than just trading off one against another

A note on classic compute

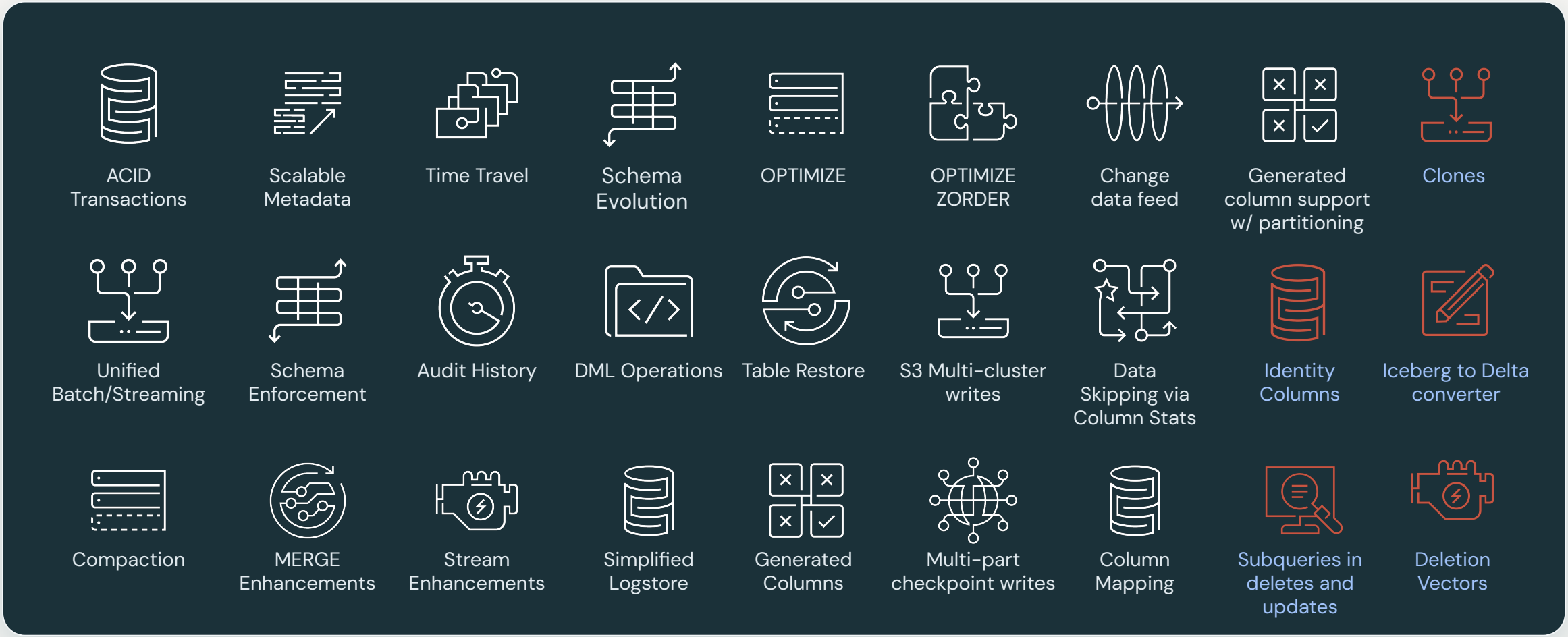
More knobs the better?

- DBSQL Warehouse automatically take cares of compute instance selection and cluster configurations

- Classic Compute Clusters (Jobs, Interactive, etc.) still give you the ability to configure instances and here is the TLDR
  - Core:RAM Ratio - most core given enough memory for the budget
  - Processor Type - ARM based chip can work quite well
  - Local Storage - Disk Cache is useful for repeated data access and beware of storage medium
  - Driver Size - Don't over complicate it (4-8 core with 16-32 GB RAM should be enough)
  - Spot Availability - Stability is more important for long running jobs

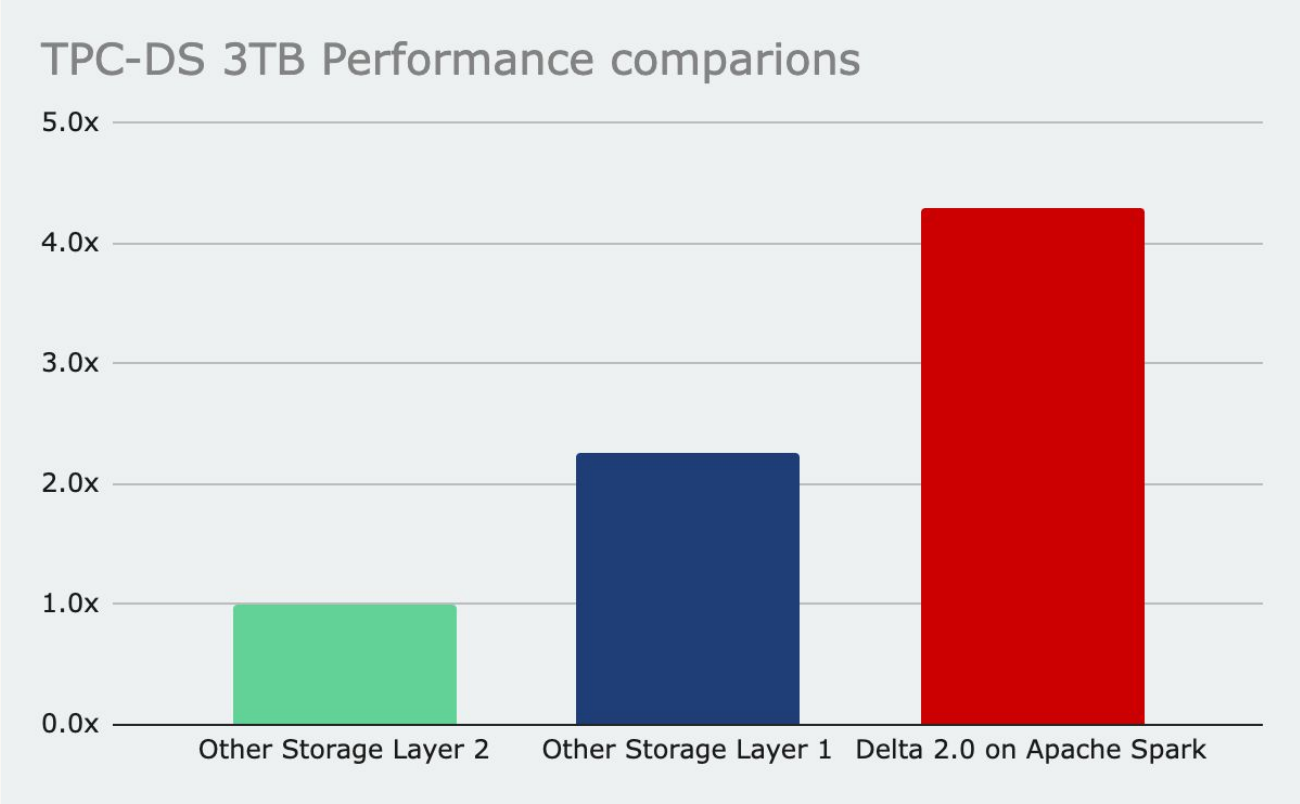
How to optimise data?

Delta Lake



Delta Lake

Most performant modern open data format

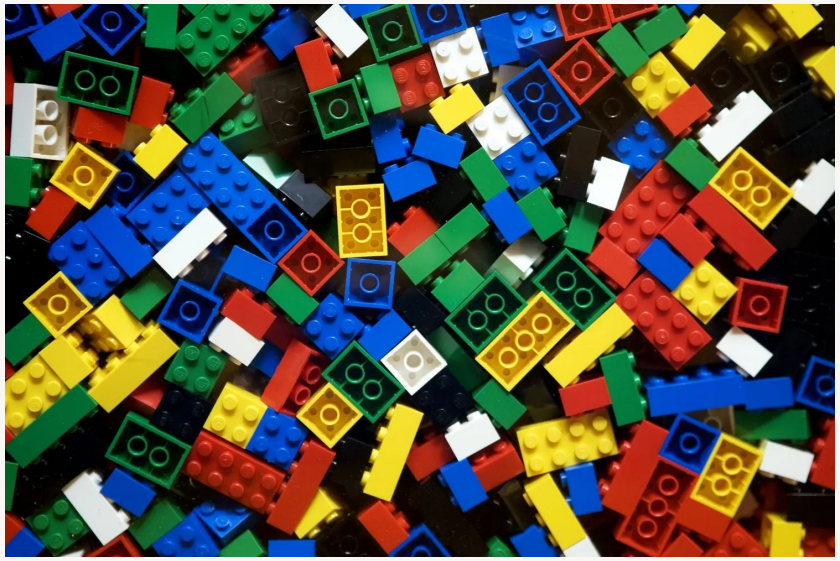


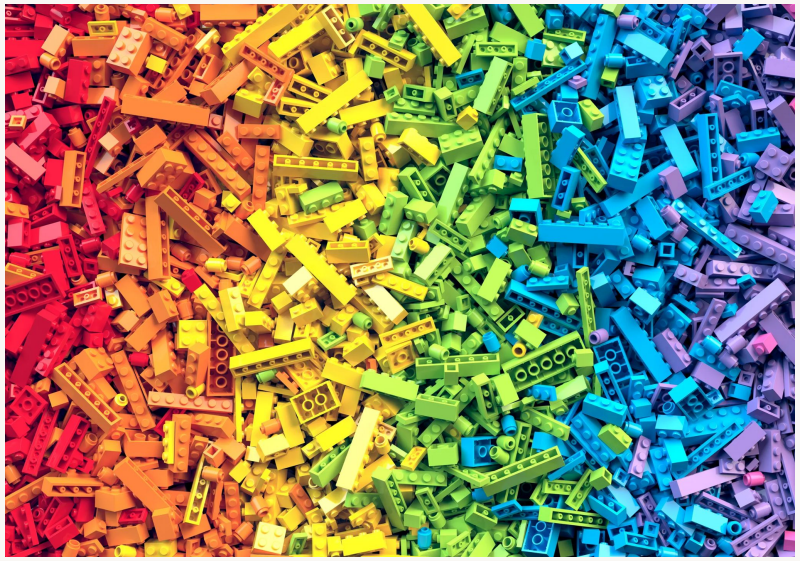
Data Layout matters

Putting the right things together makes life easier

```
SELECT COUNT(*) FROM LEGOS WHERE COLOUR = "RED"
```

How do you want to store your legos?





What about `SELECT COUNT(*) FROM LEGOS WHERE SIZE = "SMALL"`?

Data Layout Rationale

Different ways to organise your data so that we don't need to read too much unnecessary files

```
Select * from deltalake_table where part = 2 and col = 6
```

Partition Pruning

```
/path/to/deltalake_table/
                    part=1/part_00001.parquet
                    part=1/part_00002.parquet
                    part=1/part_00003.parquet
                    part=2/part_00001.parquet
                    part=2/part_00002.parquet
                    part=2/part_00003.parquet
```

File Skipping

file_name	col_min	col_max
1.parquet	1	3
2.parquet	4	7
3.parquet	8	10

Data Layout #1 – To partition or not partition

DO NOT partition unless you know why you are partitioning

- **Over-partition is worse than no partition at all**
  - small files kill performance
- Reasons to partition
  - Isolating data for separate schemas (i.e. multiplexing)
  - Governance use cases where you commonly delete entire partitions of data
  - Physical boundary to isolate data is required
  - Table size > 100TB
- Partition best practices
  - Keep partition size between 1GB and 1TB
  - Combine partition with Z-order

Data layout #2 – Clustering

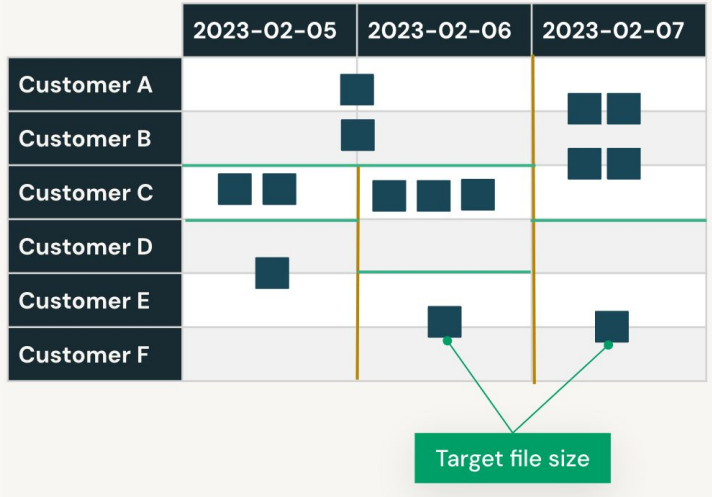
Sort the data in ways that you will need it

Z-Ordering

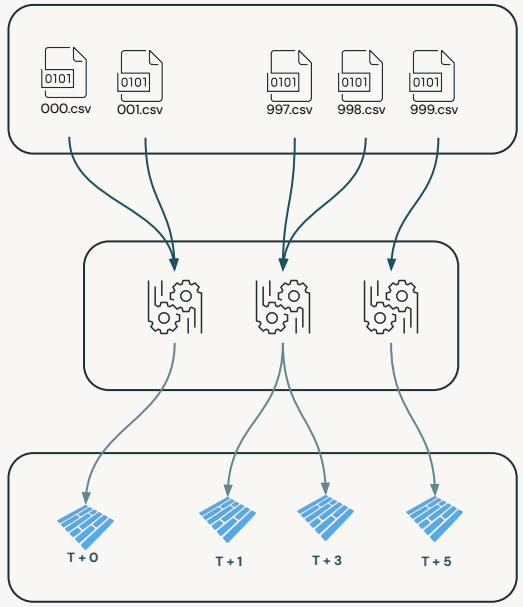
file_name	col_min	col_max
1.parquet	6	8
2.parquet	3	10
3.parquet	1	4

file_name	col_min	col_max
1.parquet	1	3
2.parquet	4	7
3.parquet	8	10

Liquid Clustering



Ingestion Time Clustering



Data Layout #3 – File Sizes

When it comes to performance, file size matters

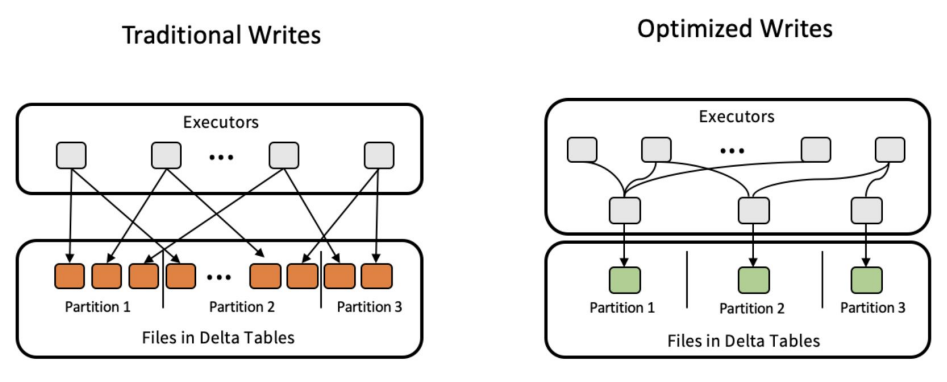
Small File Size

- Less data to read
- More files
- Rewrite is cheaper

Large File Size

- More data to read
- Less files
- Rewrite is expensive

`delta.tuneFileSizesForRewrites`



Deletion Vector

Amortisation of rewrite costs

Inserts

w/o DV

File

rowNum, data
1, data
2, data
3, data
4, data
5, data
6, data
7, data
8, data

rowNum, data
1, data

w/ DV

File

rowNum, data
1, data
2, data
3, data
4, data
5, data
6, data
7, data
8, data

rowNum, data
1, data

Deletes

w/o DV

File

rowNum, data
1, data
2, data
3, data
4, data
5, data
6, data
7, data

w/ DV

File

rowNum, data	DV
1, data	0
2, data	0
3, data	0
4, data	1
5, data	0
6, data	0
7, data	0
8, data	0

Full File Rewrite

Updates

w/o DV

File

rowNum, data
1, data
2, data
3, data
4, new data
5, data
6, data
7, data
8, data

w/ DV

File

rowNum, data	DV
1, data	0
2, data	0
3, data	0
4, data	1
5, data	0
6, data	0
7, data	0
8, data	0

rowNum, data
1, new data

Full File Rewrite

Predictive Optimization

Bringing it all together automatically

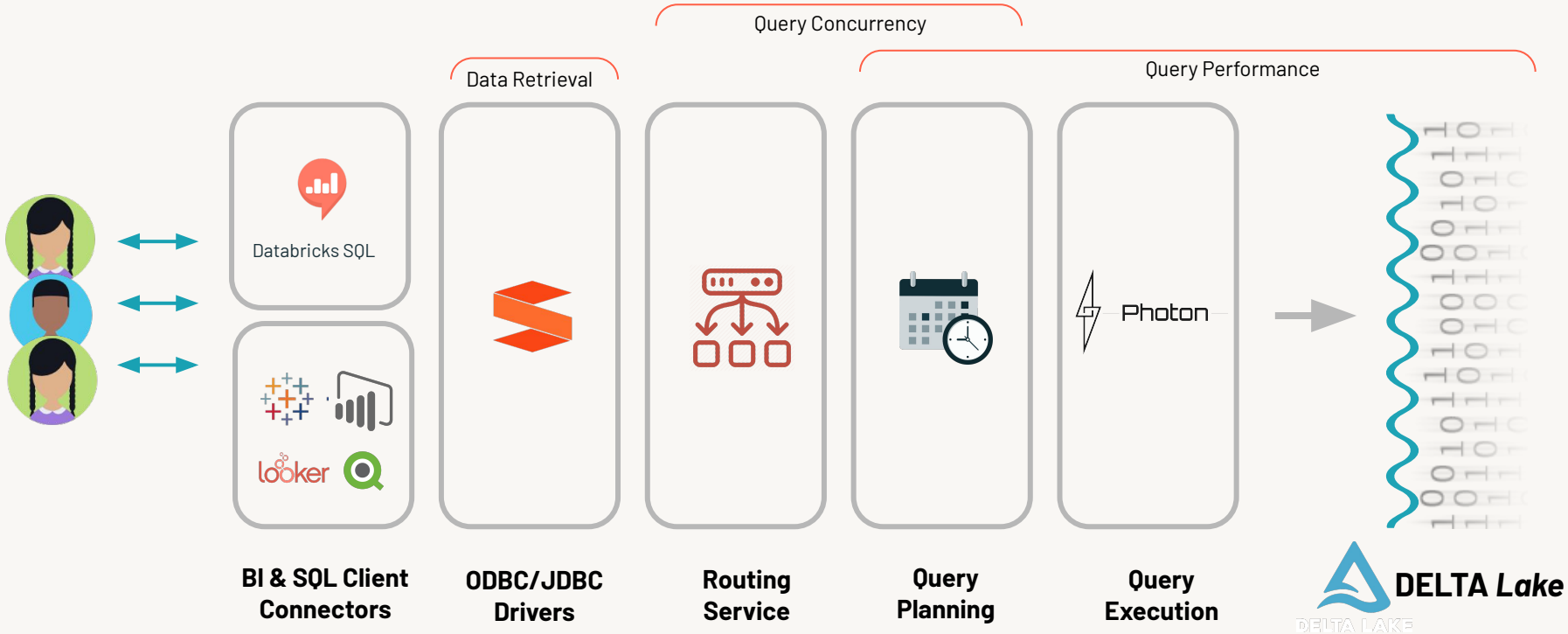
- Scheduling of these optimisation can be tricky
- Mistakes can be made if users forget to set up these process
- Predictive Optimization automatically determines which operations to execute based on usage
- Prioritise high return operations based on expected benefits

How to optimise code?

Diagnosing performance issues

Finding the problem/bottleneck is most of the battle

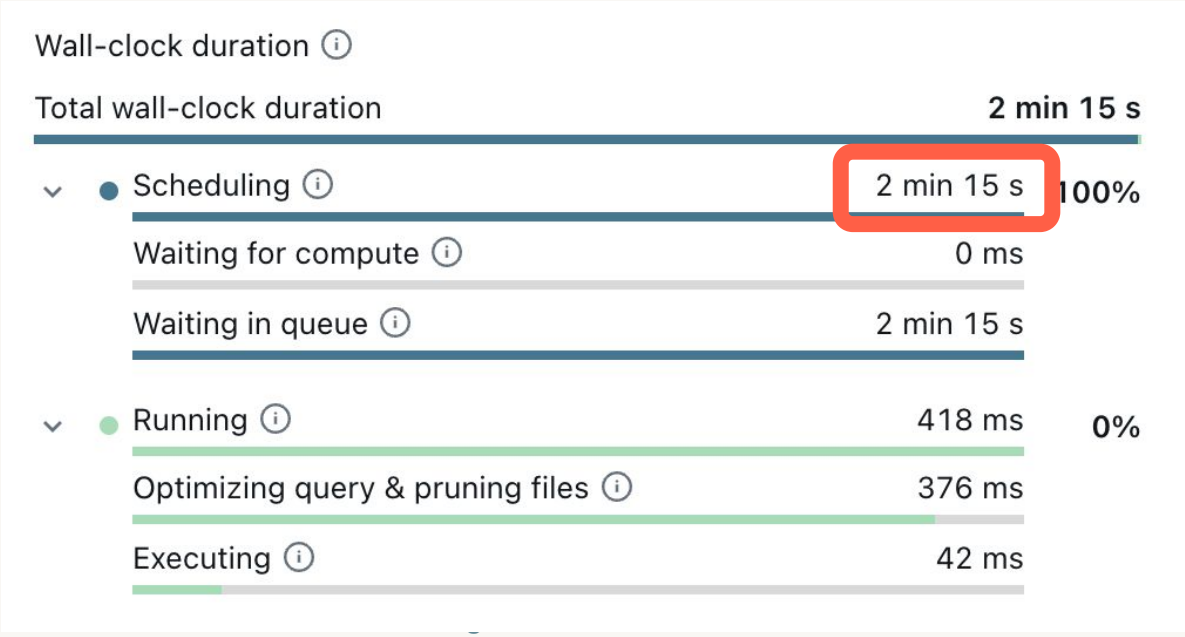
- Fixing code is often last resort, if everything else is working well, queries usually perform well

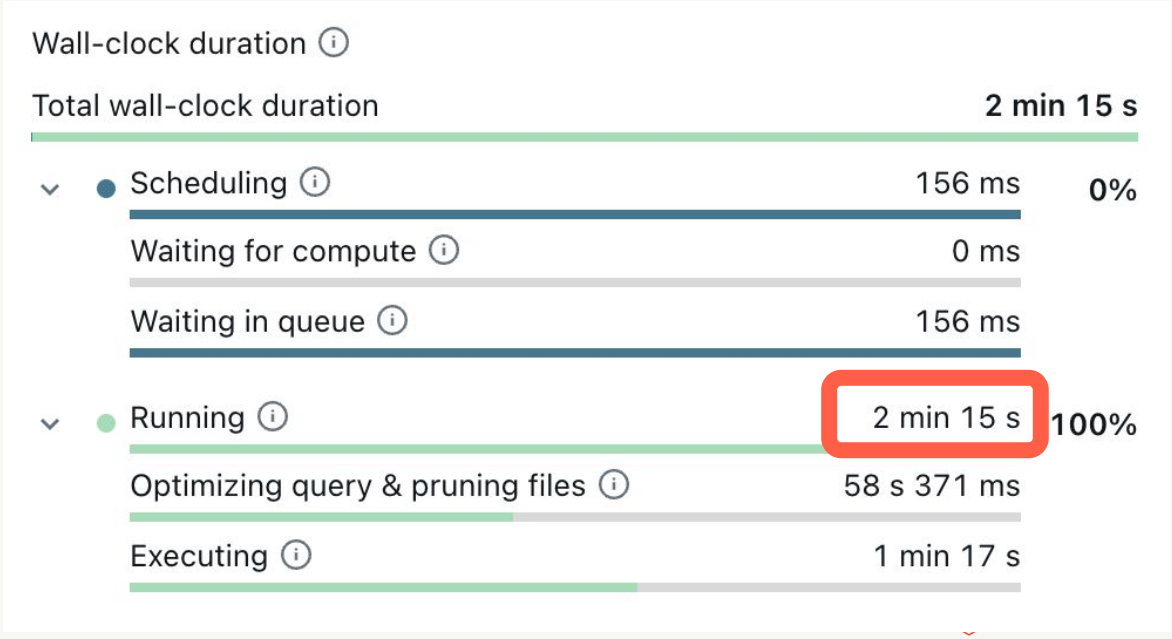


Diagnosing performance issues

Scheduling Time v. Running Time

- Same wall clock time != same performance
- Scheduling time has nothing to do with your code
  - Waiting for compute = Need running compute = Serverless / pre-started cluster
  - Waiting in queue = We need more concurrency = Increase max # of clusters

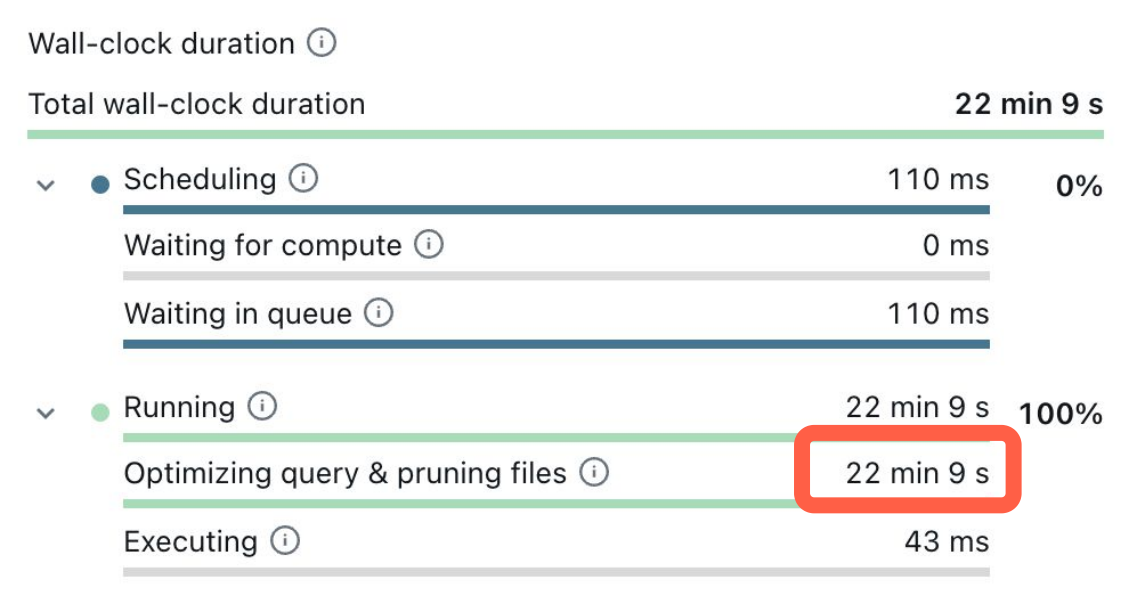


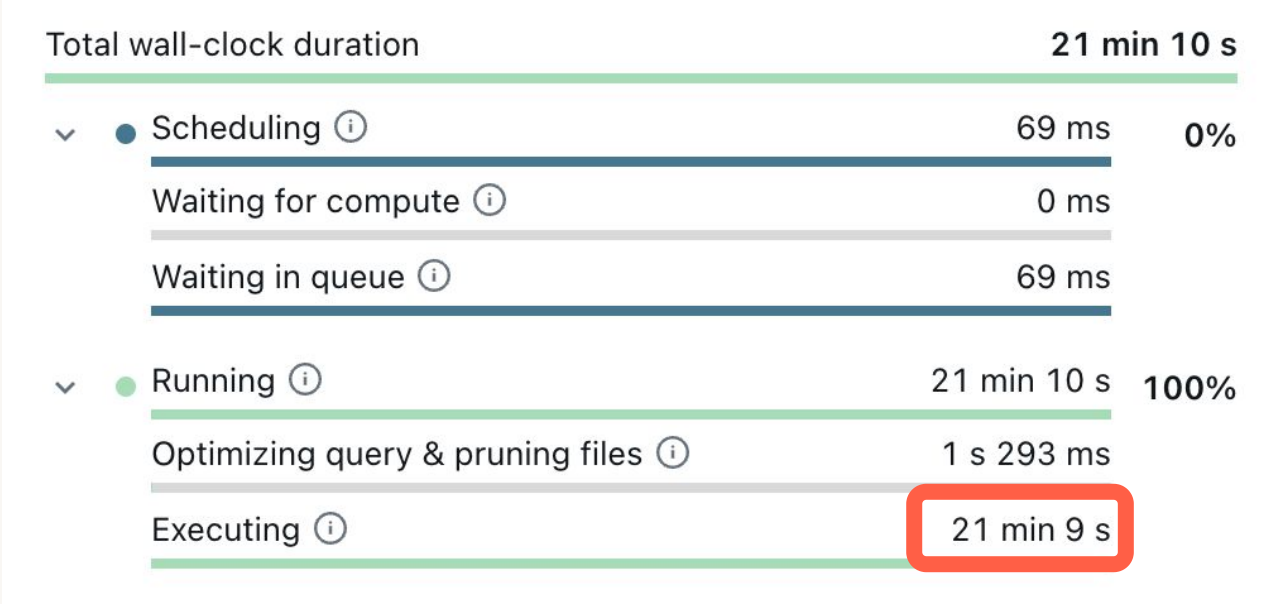


Diagnosing performance issues

Running Time Breakdown

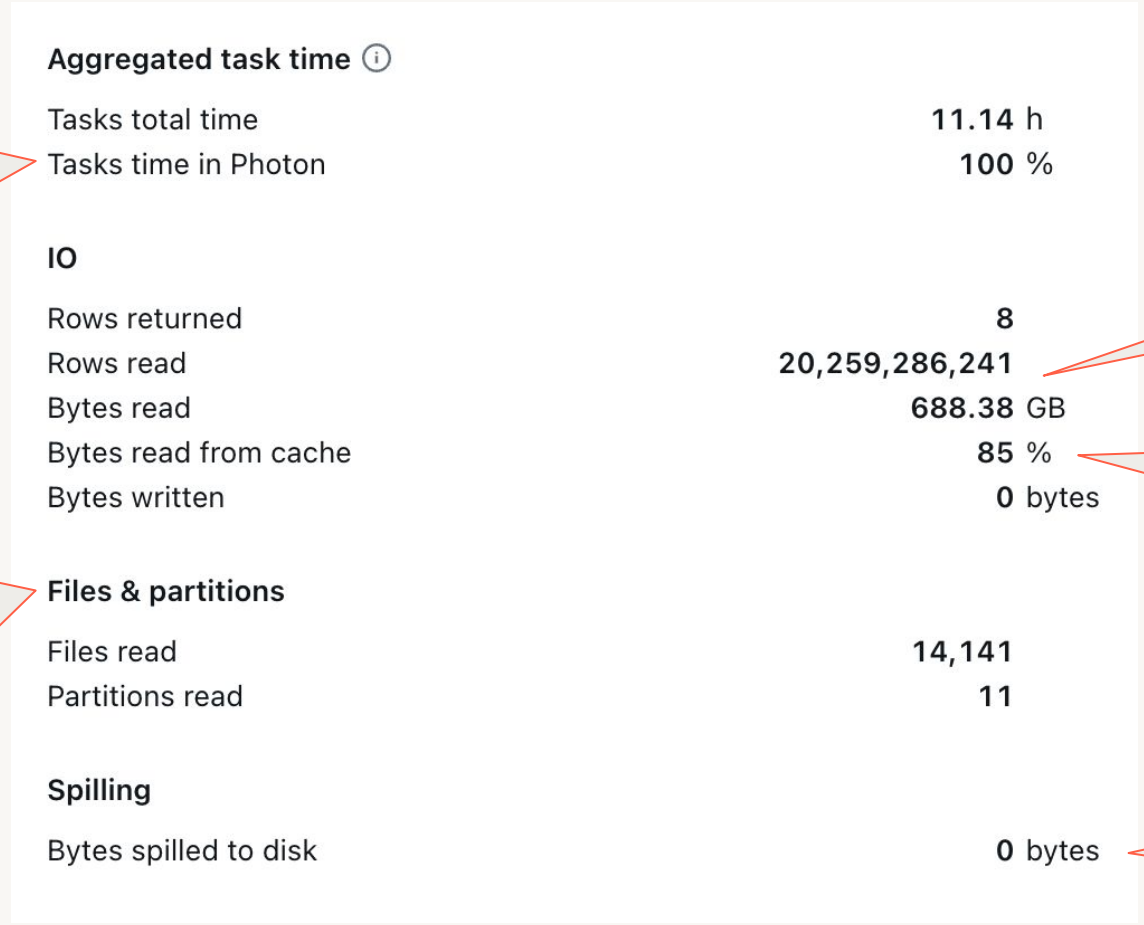
- Same running time != Same time spent on execution
- Long Optimizing Query & Pruning Files Time = better stats collection





Diagnosing performance issues

Execution Details



Aggregated task time

Tasks total time	11.14 h
Tasks time in Photon	100 %

IO

Rows returned	8
Rows read	20,259,286,241
Bytes read	688.38 GB
Bytes read from cache	85 %
Bytes written	0 bytes

Files & partitions

Files read	14,141
Partitions read	11

Spilling

Bytes spilled to disk	0 bytes

Make sure photon is close to 100%

Does the number of rows read make sense? Did you read too much data?

Disk cache meant your data is already cached on local storage

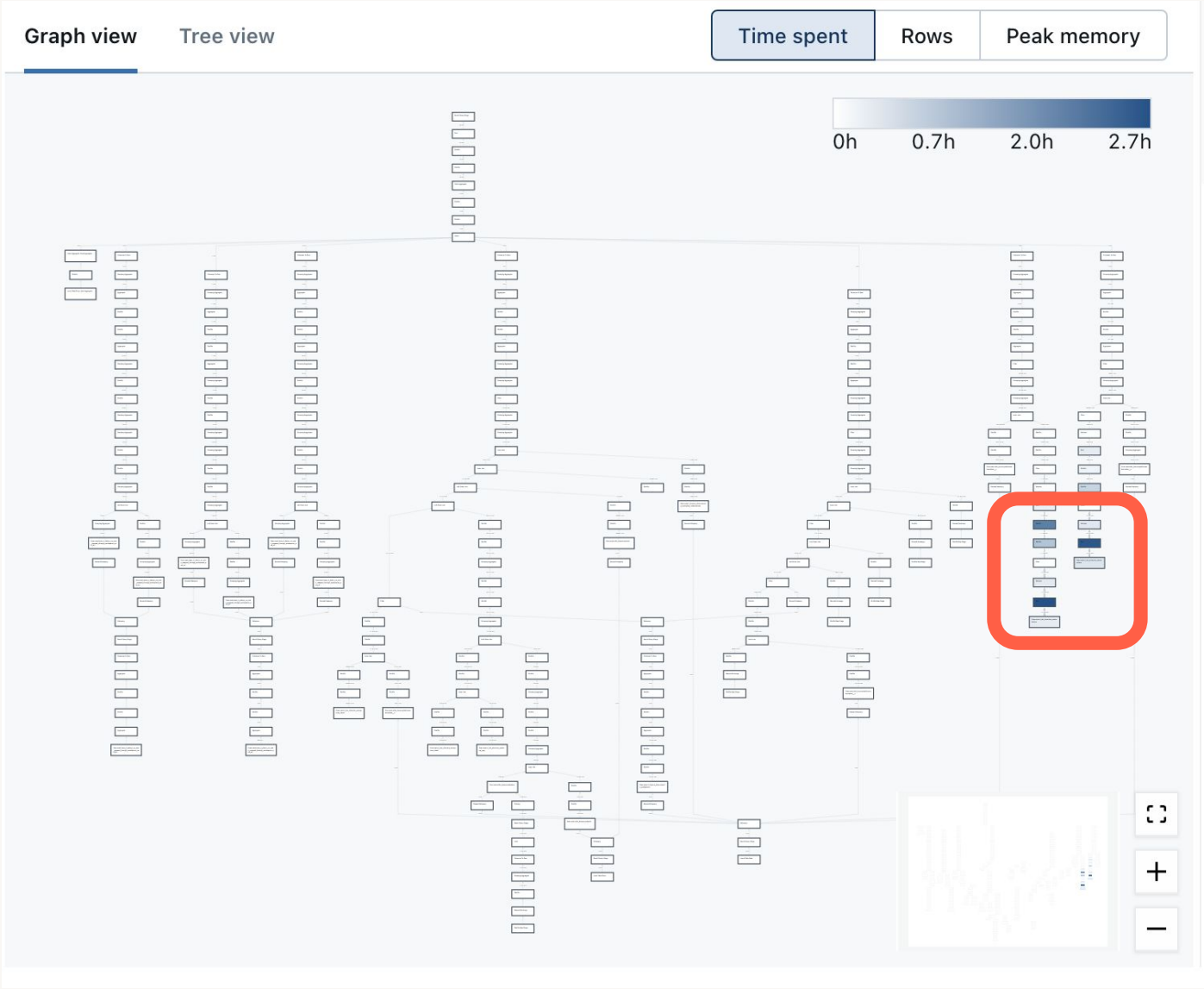
If your data read is correct, are you reading too many files or partitions?

Spill meant your warehouse/cluster is too small, i.e. Not enough RAM

Diagnosing performance issues

Understanding Query Profile

- Execution can be broken down to individual operations within your query
- The most time spent operation is likely where you need to start
- It should tell you which part of the query is causing problem
- Knowing what is the problem doesn't mean it is an easy fix



Simple query is usually a fast query

Get to the results with the least amount of data and transformation

1. Predicates are pushed as far up as possible
    a. Select the least amount of columns and rows that you need
    b. Align data layout with commonly used predicates (ZORDER, LIQUID)
    c. Make sure data is right sized (OPTIMIZE)
2. Simplify how you join your tables
    a. Join the smallest tables first / collect statistics for the optimiser to do it for you
    b. Provide join hints if you can
    c. Reduce unnecessary data movement, i.e. If you know your data layout and join keys you can use the right join strategy (sort-merge v. shuffle hash)
3. Simplify operations
    a. Be careful about expensive operations (distinct, sort, window)
    b. UDFs are powerful but they are not fast, try to use native functions as much as possible

Diagnosing performance issues

There are many ways to return Query Results

Driver/Client

- Update JDBC/ODBC driver
- Client might be the bottleneck, try run the query to confirm where the bottleneck is

Network

- Check network path between client and control plane
- Firewall or other network control might be slowing down result delivery
- Different network path, e.g. Private Link, might be provide higher bandwidth, hence better performance

CloudFetch

- CloudFetch is a high bandwidth connectivity path for BI and JDBC/ODBC tools
- Make sure driver is up to date and network path is clear
- Confirm CloudFetch is used at the last step (`CloudStoreCollector`)

What now?

Key takeaways

Optimise only when necessary

- Know what you are optimising towards
- Focus on the easy things first (Platform -> Data -> Query)
- Platform
  - Leverage latest compute and features (i.e. serverless compute, photon, predictive optimisation)
  - Fine-tune compute infrastructure when appropriate
- Data
  - Understand your tradeoffs between read and write
  - Housekeeping is important
- Query
  - Focus on simplification rather than optimisation
- Know when to stop optimising and start building more useful things

Thank you