使用Delta Lake构建湖仓一体

王耿亮

关于我

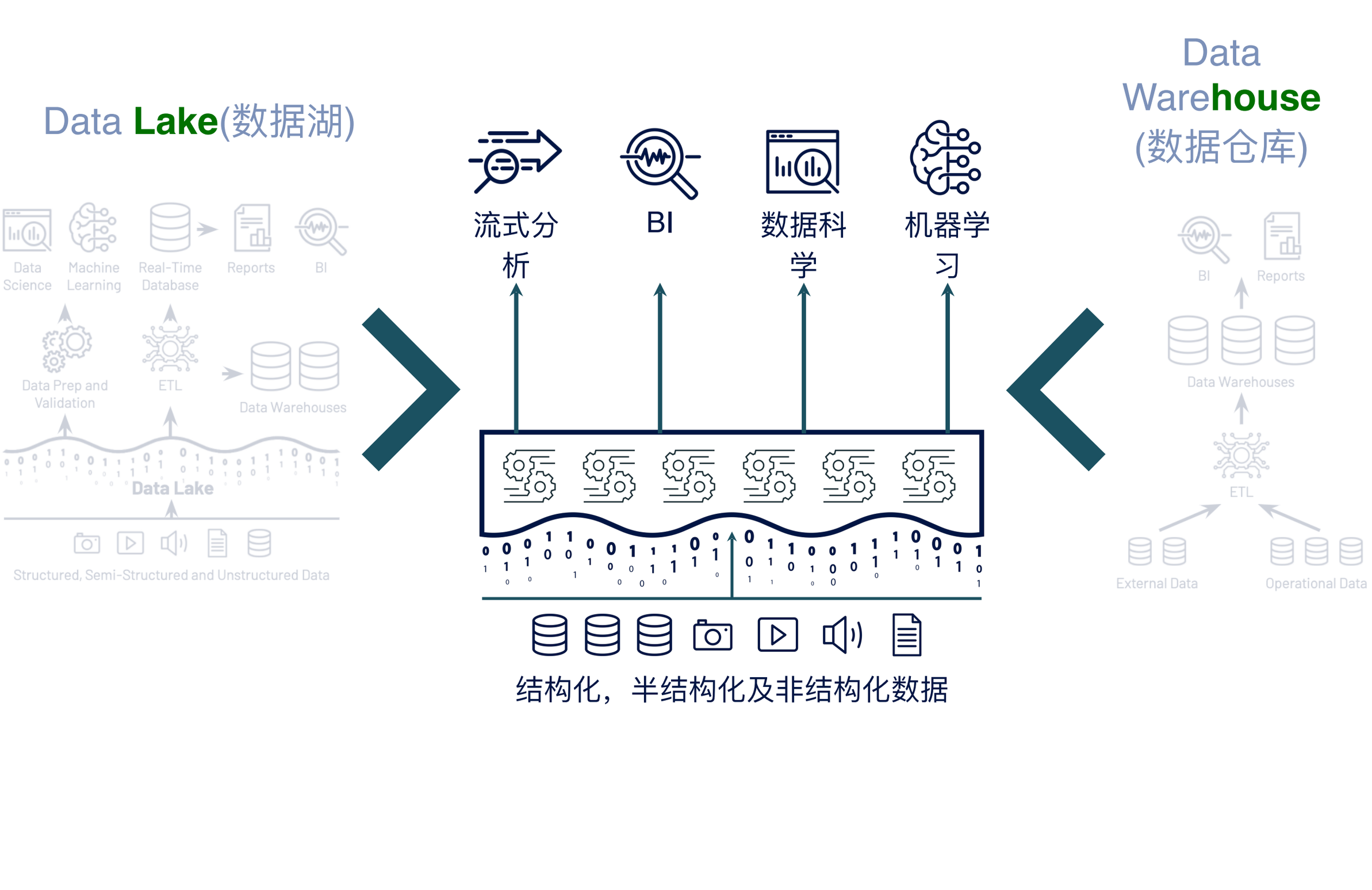
王耿亮

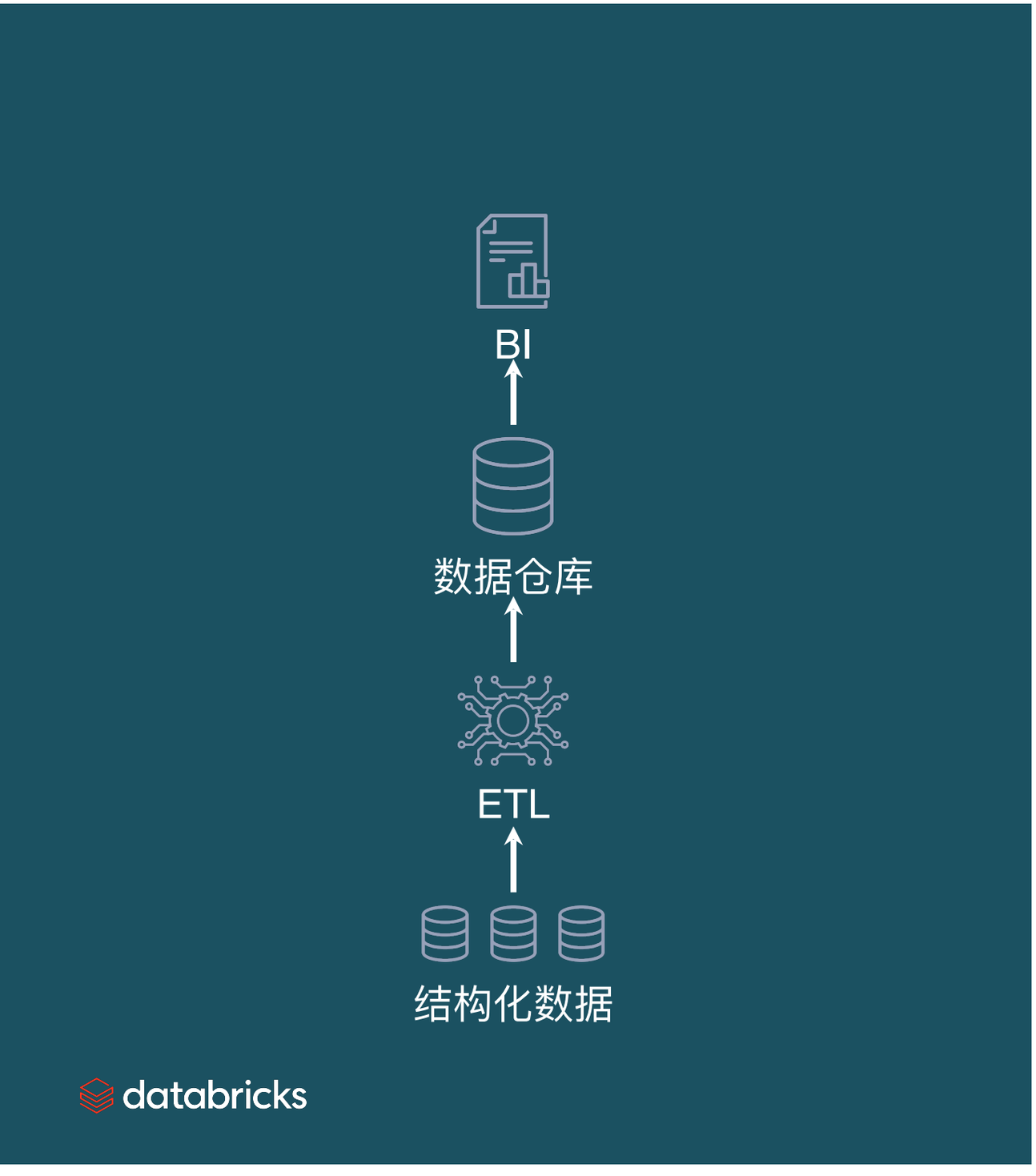
Databricks 资深研发工程师

Apache Spark committer

Github ID: [gengliangwang]

Lakehouse(湖仓一体)





数据仓库

优势

- 数据质量高
- 高性能
- 便于BI系统接入

劣势

- 不支持非结构化数据
- 难以支持高速度、高容量的数据写入
- 专有系统，数据迁移和同步的灵活度很低

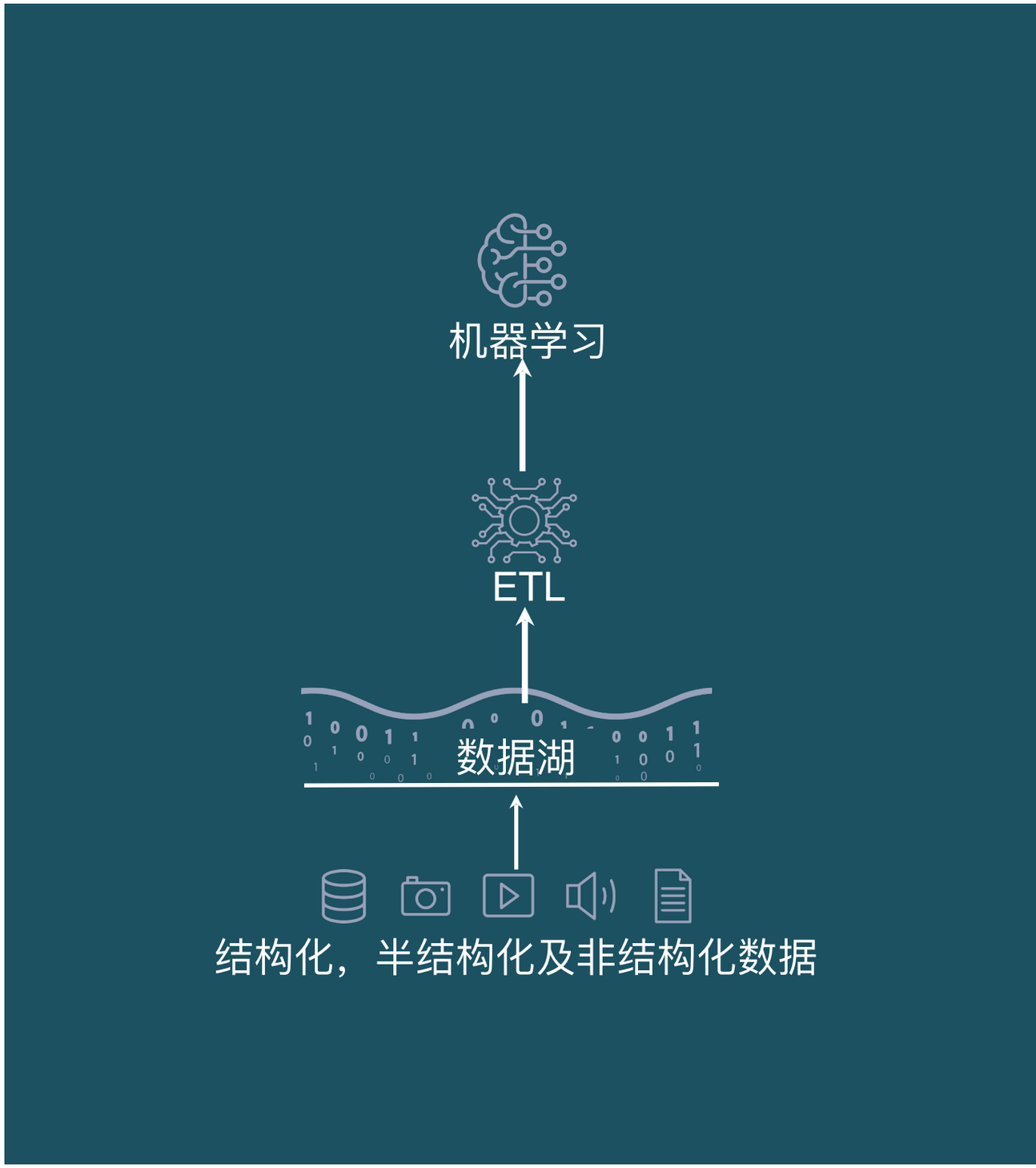
数据湖

优势

- 支持非结构化数据
- 开放的格式与平台

劣势

- 对BI系统的支持不足
- 随着时间推移，数据质量差

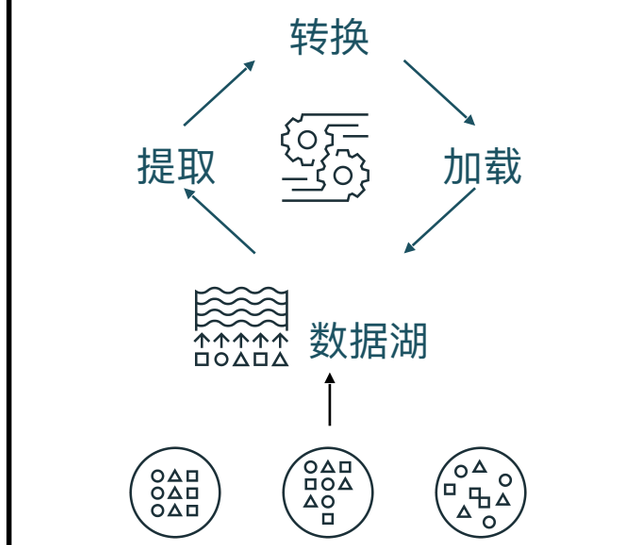


复杂的组合型数据系统

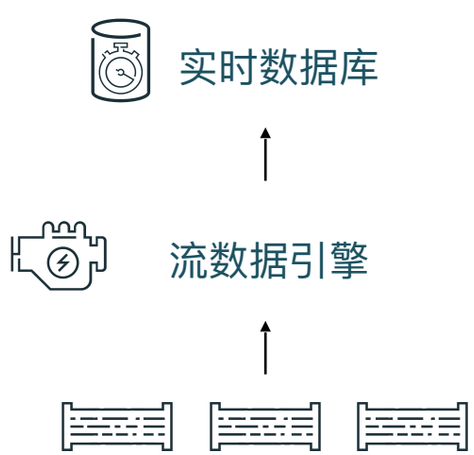
孤立的技术栈，使得数据架构复杂性提升

数据仓库

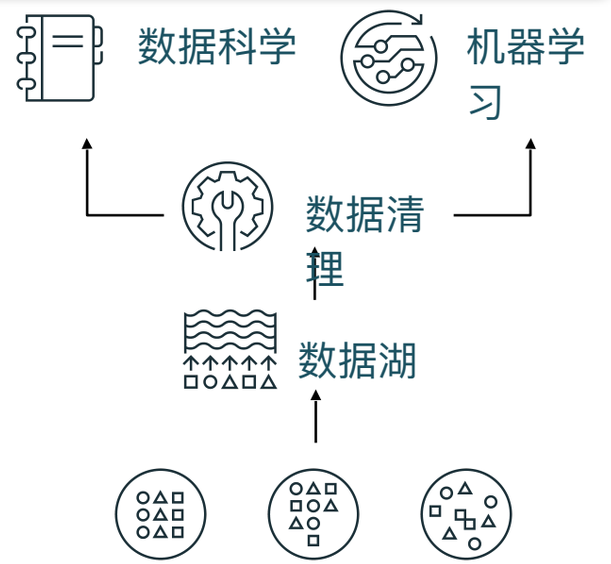
数据工程



流式处理



数据科学与机器学习



复杂的组合型数据系统

数据仓库 | 数据工程 | 流式处理 | 数据科学与机器学习

多个系统，多种专有数据格式，不便于集成

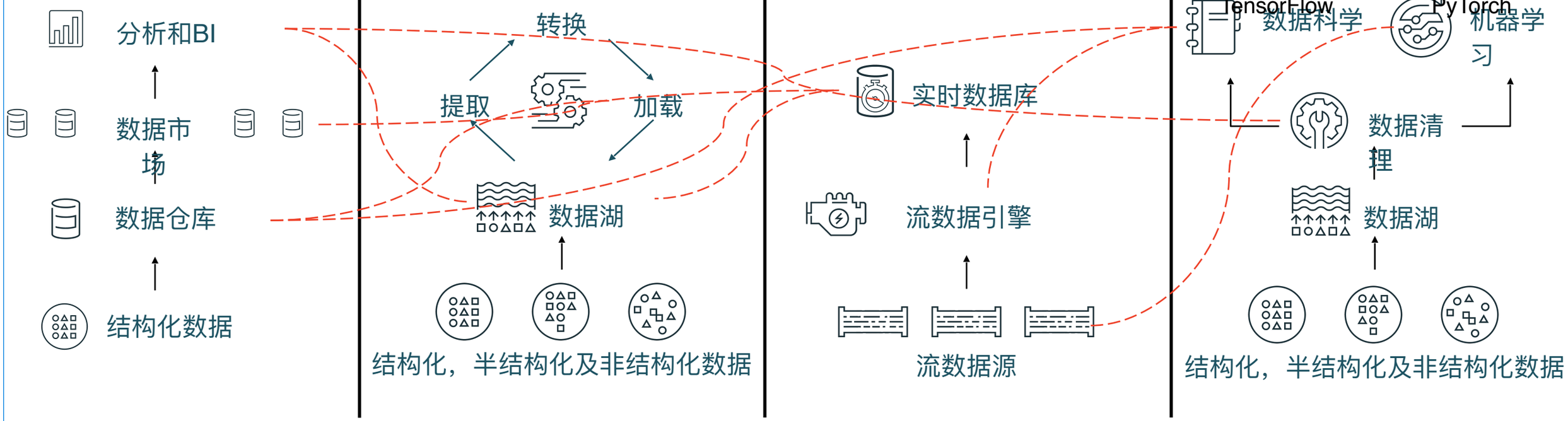
Amazon Redshift
Azure Synapse
Snowflake
SAP
Teradata
Google BigQuery
IBM Db2
Oracle
Autonomous Data Warehouse

Hadoop
Amazon EMR
Google Dataproc
Apache Airflow
Apache Spark
Cloudera
Tibco Spotfire

Apache Kafka
Apache Flink
Azure Stream Analytics
Apache Spark
Amazon Kinesis
Google Dataflow
Confluent

Jupyter
Azure ML Studio
Domino Data Labs
Amazon SageMaker
MatLAB
SAS
TensorFlow
PyTorch

孤立的技术栈，使得数据架构复杂性提升



复杂的组合型数据系统

数据仓库 | 数据工程 | 流式处理 | 数据科学与机器学习

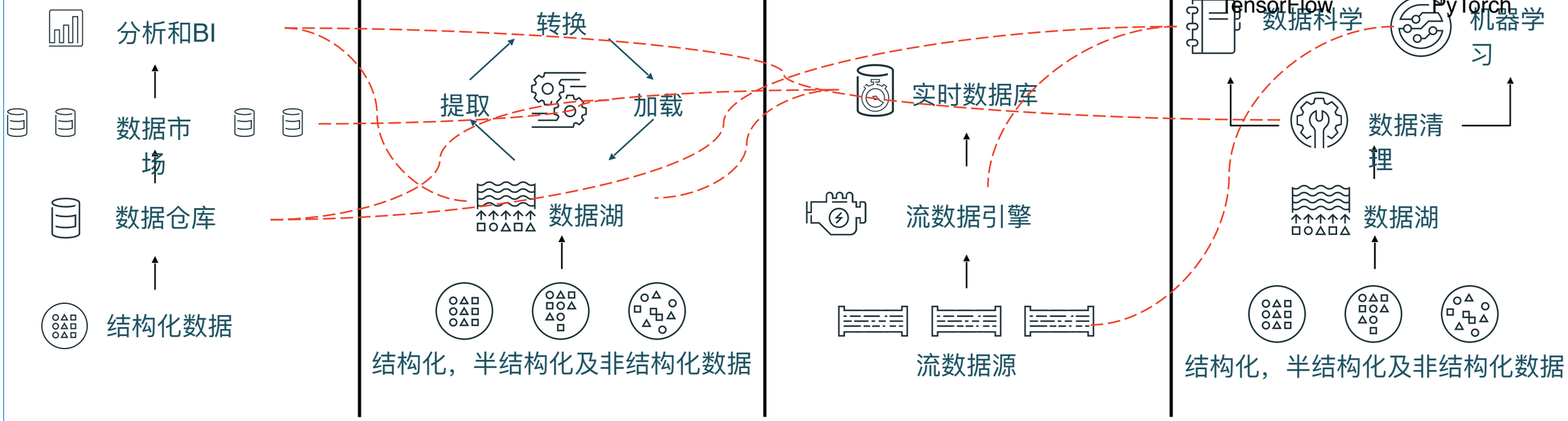
分散的数据团队影响生产力

数据分析师 | 数据工程师 | 数据工程师 | 数据科学家

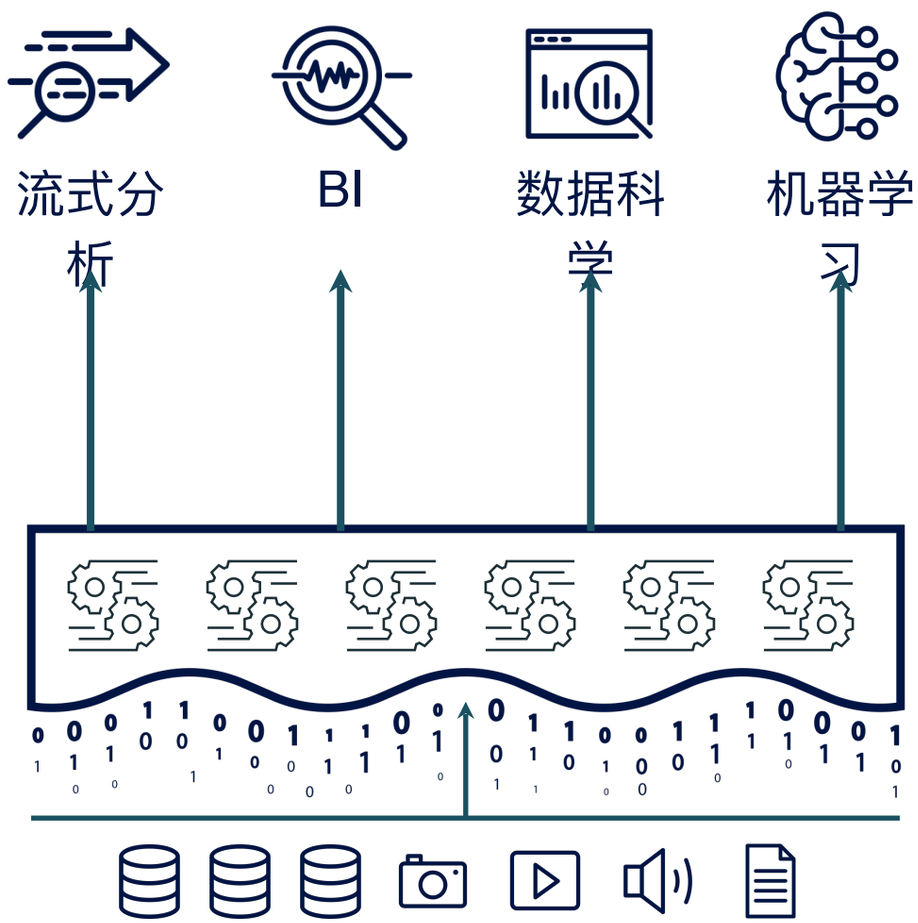
多个系统，多种专有数据格式，不便于集成

数据仓库		数据工程		流式处理		数据科学与机器学习	
Amazon Redshift	Teradata	Hadoop	Apache Airflow	Apache Kafka	Apache Spark	Jupyter	Amazon SageMaker
Azure Synapse	Google BigQuery	Amazon EMR	Apache Spark	Apache Flink	Amazon Kinesis	Azure ML Studio	MatLAB
Snowflake	IBM Db2	Google Dataproc	Cloudera	Azure Stream Analytics	Google Dataflow	Domino Data Labs	SAS
SAP	Oracle Autonomous Data Warehouse			Tibco Spotfire	Confluent	TensorFlow	PyTorch

孤立的技术栈，使得数据架构复杂性提升

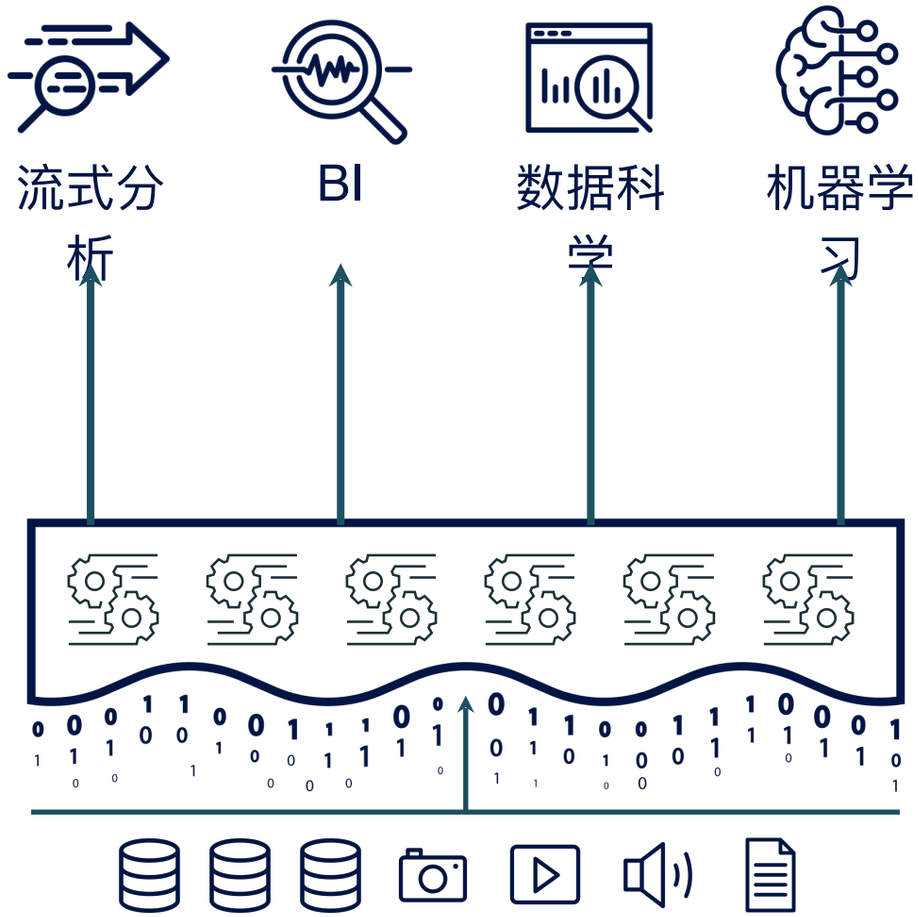


Lakehouse(湖仓一体)



适用于所有场景的统一平台

Lakehouse(湖仓一体) 关键特点



- 事务支持
- BI支持
- 存储与计算分离
- 开放性
- 支持从非结构化数据到结构化数据的多种数据类型
- 支持数据科学、机器学习以及SQL各种工作负载
- 流式分析

Lakehouse(湖仓一体)

适用于所有场景的统一平台

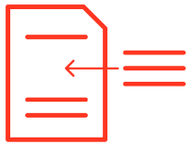
结构化事务层

包含所有数据的数据湖

Lakehouse(湖仓一体) - 具体问题

1. 读写并行问题 - 追加写

添加新数据可能会导致错误的读结果



2. 修改已有数据问题

用户会有修改已有数据的需求

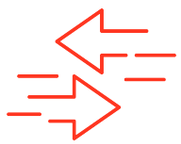
3. 中途失败的作业

部分数据存入数据湖

Lakehouse(湖仓一体) - 具体问题

4. 批流混合输入

可能会导致不完整的读取结果

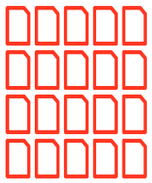


5. 保存数据历史

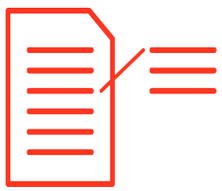
由于审计和合规的需求，需要保证可重复性

6. 处理海量元数据

对于大型数据湖，处理其海量的元数据也是一大挑战



7. 大量小文件问题
数据湖不擅长处理大量小文件



8. 性能问题
很难达到数据仓库的性能

9. 数据查询管控
难于确保查询的合规性

10. 数据质量问题
在大数据场景下保证数据质量

1. 读写并行问题
2. 修改已有数据
3. 中途失败作业
4. 批流混合输入
5. 保存数据历史
6. 处理海量元数据
7. 大量小文件问题
8. 性能问题
9. 数据查询管控
10. 数据质量问题

1. 读写并行问题
2. 修改已有数据
3. 中途失败作业
4. 批流混合输入
5. 保存数据历史
6. 处理海量元数据
7. 大量小文件问题
8. 性能问题
9. 数据查询管控
10. 数据质量问题

ACID 事务

事务化所有操作

- 每一个操作，要么整体成功，要么整体失败

```
/path/to/table/
_delta_log
 -  0000.json
 -  0001.json
 -  0002.json
 -  …
 -  0010.parquet
```

1. 读写并行问题
2. 修改已有数据
3. 中途失败作业
4. 批流混合输入
5. 保存数据历史
6. 处理海量元数据
7. 大量小文件问题
8. 性能问题
9. 数据查询管控
10. 数据质量问题

ACID 事务

事务化所有操作

- 每一个操作，要么整体成功，要么整体失败

```
/path/to/table/
_delta_log
 -  0000.json   { Add  file1.parquet
 -  0001.json     Add  file2.parquet
 -  0002.json     …
 -  …
 -  0010.parquet
```

1. 读写并行问题
2. 修改已有数据
3. 中途失败作业
4. 批流混合输入
5. 保存数据历史
6. 处理海量元数据
7. 大量小文件问题
8. 性能问题
9. 数据查询管控
10. 数据质量问题

ACID 事务

事务化所有操作

- 每一个操作，要么整体成功，要么整体失败

```
/path/to/table/
_delta_log
 -  0000.json   { Remove file1.parquet
 -  0001.json   { Add file3.parquet
 -  0002.json   { ...
 -  …
 -  0010.parquet
```

1. 读写并行问题
2. 修改已有数据
3. 中途失败作业
4. 批流混合输入
5. 保存数据历史
6. 处理海量元数据
7. 大量小文件问题
8. 性能问题
9. 数据查询管控
10. 数据质量问题

ACID 事务

事务化所有操作

- 每一个操作，要么整体成功，要么整体失败

查询历史数据

- 所有的操作都在事务日志中有记录，可以对之前某个时间点的数据进行查询（时间旅行）

```
SELECT * FROM
events
TIMESTAMP AS
OF ...

SELECT * FROM
events
VERSION AS OF ...
```

1. 读写并行问题
2. 修改已有数据
3. 中途失败作业
4. 批流混合输入
5. 保存数据历史
6. 处理海量元数据
7. 大量小文件问题
8. 性能问题
9. 数据查询管控
10. 数据质量问题

使用Spark处理元数据

- 所有Delta Lake元数据均以开源Parquet格式存储
- 数据与元数据总是相伴相生，无需进行同步

1. 读写并行问题
2. 修改已有数据
3. 中途失败作业
4. 批流混合输入
5. 保存数据历史
6. 处理海量元数据
7. 大量小文件问题
8. 性能问题
9. 数据查询管控
10. 数据质量问题

索引机制

数据的自动优化

- 分区裁剪
- 布隆过滤器索引文件
- Data Skipping
- Z-ordering：优化多个列的存储布局

```
OPTIMIZE events
ZORDER BY
(eventType)
```

1. 读写并行问题
2. 修改已有数据
3. 中途失败作业
4. 批流混合输入
5. 保存数据历史
6. 处理海量元数据
7. 大量小文件问题
8. 性能问题
9. 数据查询管控
10. 数据质量问题

数据查询管控

表级别的权限控制

- 提供权限设置的API
- 根据用户权限，动态地对视图(view)进行脱敏（masking）

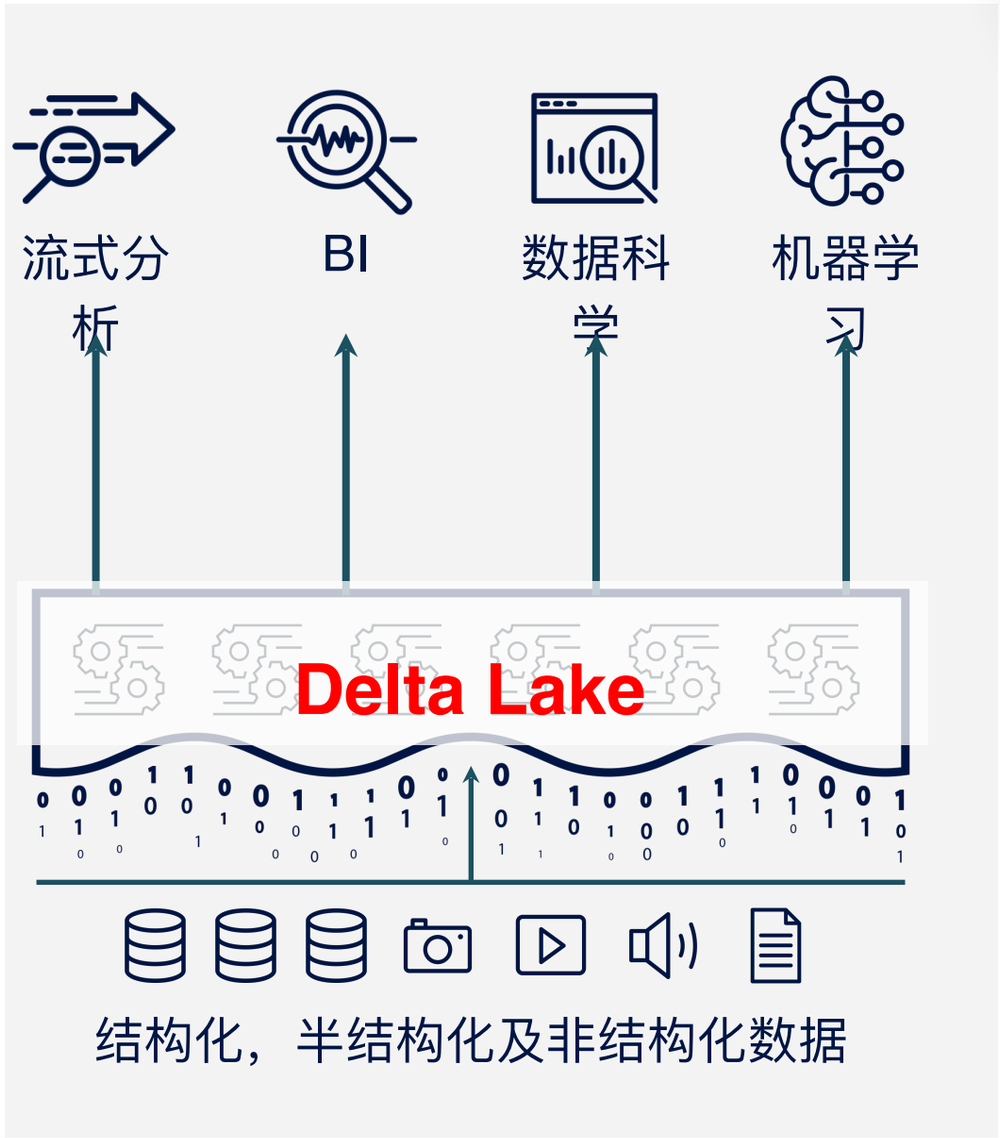
1. 读写并行问题
2. 修改已有数据
3. 中途失败作业
4. 批流混合输入
5. 保存数据历史
6. 处理海量元数据
7. 大量小文件问题
8. 性能问题
9. 数据查询管控
10. 数据质量问题

Schema 验证和演化

Schema 验证和演化

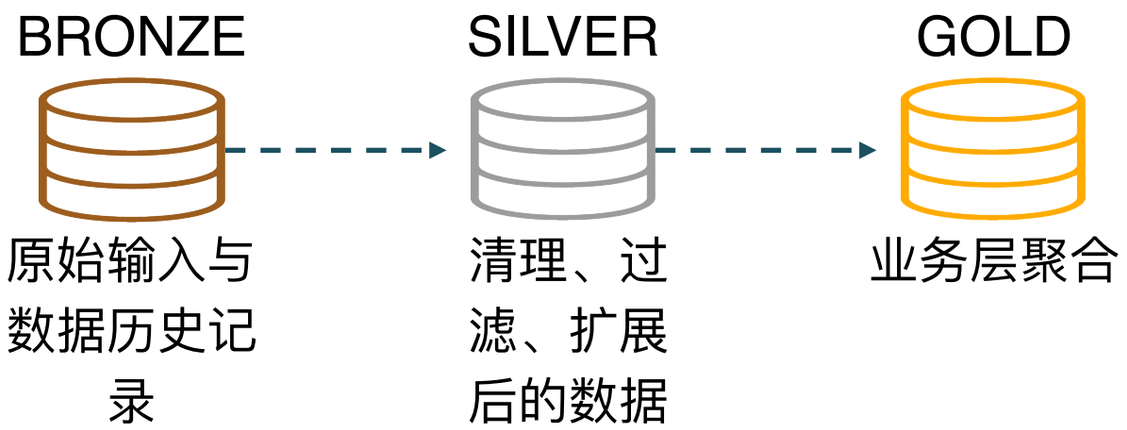
- 所有表中的数据必须严格符合schema的约束
- 可以在数据写入时进行schema演化

Delta Lake



不同场景下统一的数据基础

- 在数据湖上增加了可靠性，高性能，数据治理能力以及数据质量保证
- 基于开放的数据格式 (Parquet)
- 采用多层数据模型，简化设计

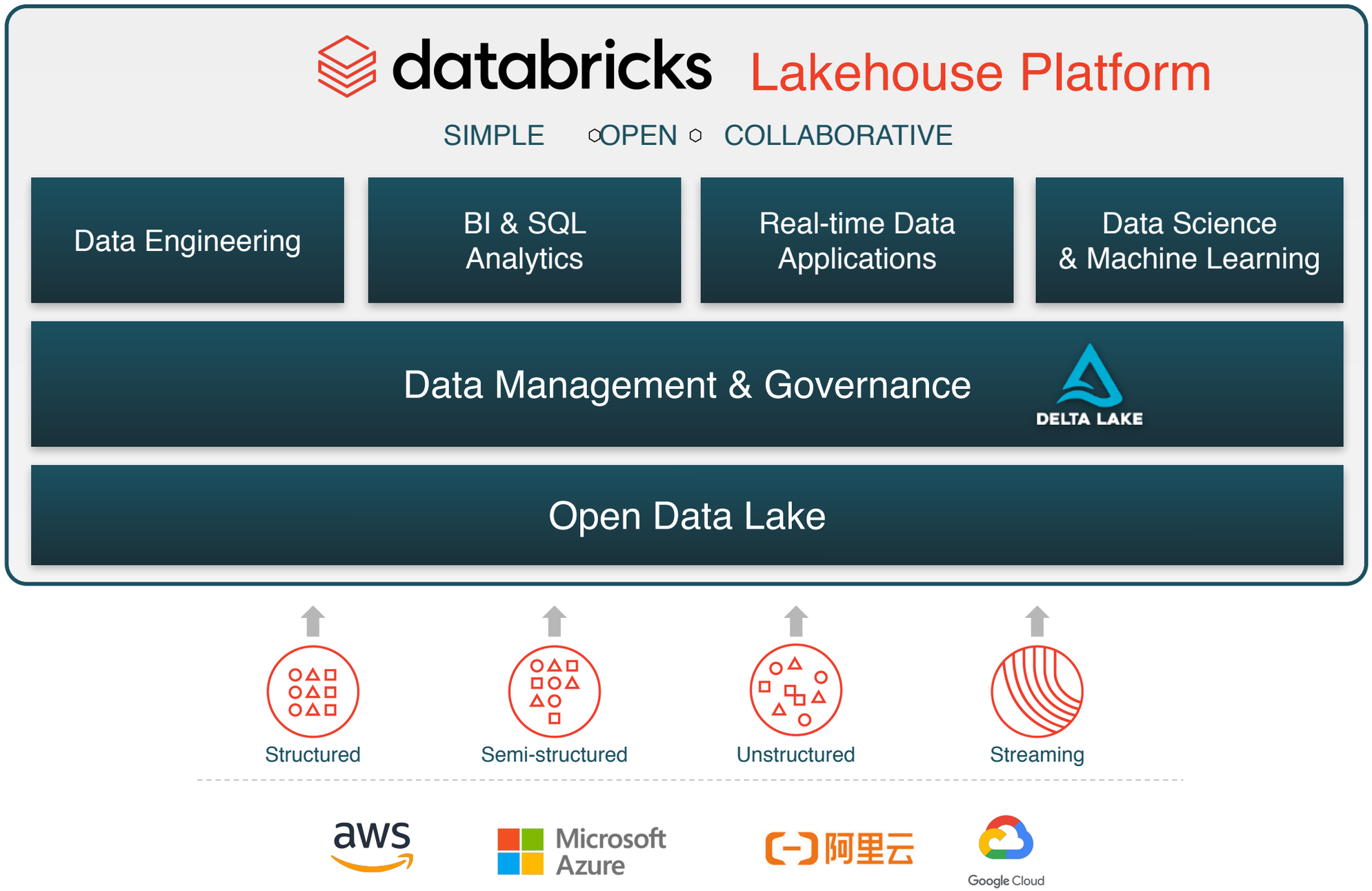


Demo

Delta 引擎

- 用 C++ 开发的矢量化引擎
- Delta引擎库加载在JVM中，通过 **JNI**与Spark进行通信
- 用户不需要修改任何原有代码
- 通常有2~4倍的性能提升
- 目前处于Public review阶段，将作为Databricks SQL 默认查询引擎

总结


databricks
Lakehouse Platform
SIMPLE
OPEN
COLLABORATIVE
Data Engineering
BI & SQL Analytics
Real-time Data Applications
Data Science & Machine Learning
Data Management & Governance
DELTA LAKE
Open Data Lake
Structured
Semi-structured
Unstructured
Streaming
aws
Microsoft Azure
阿里云
Google Cloud

Q & A