

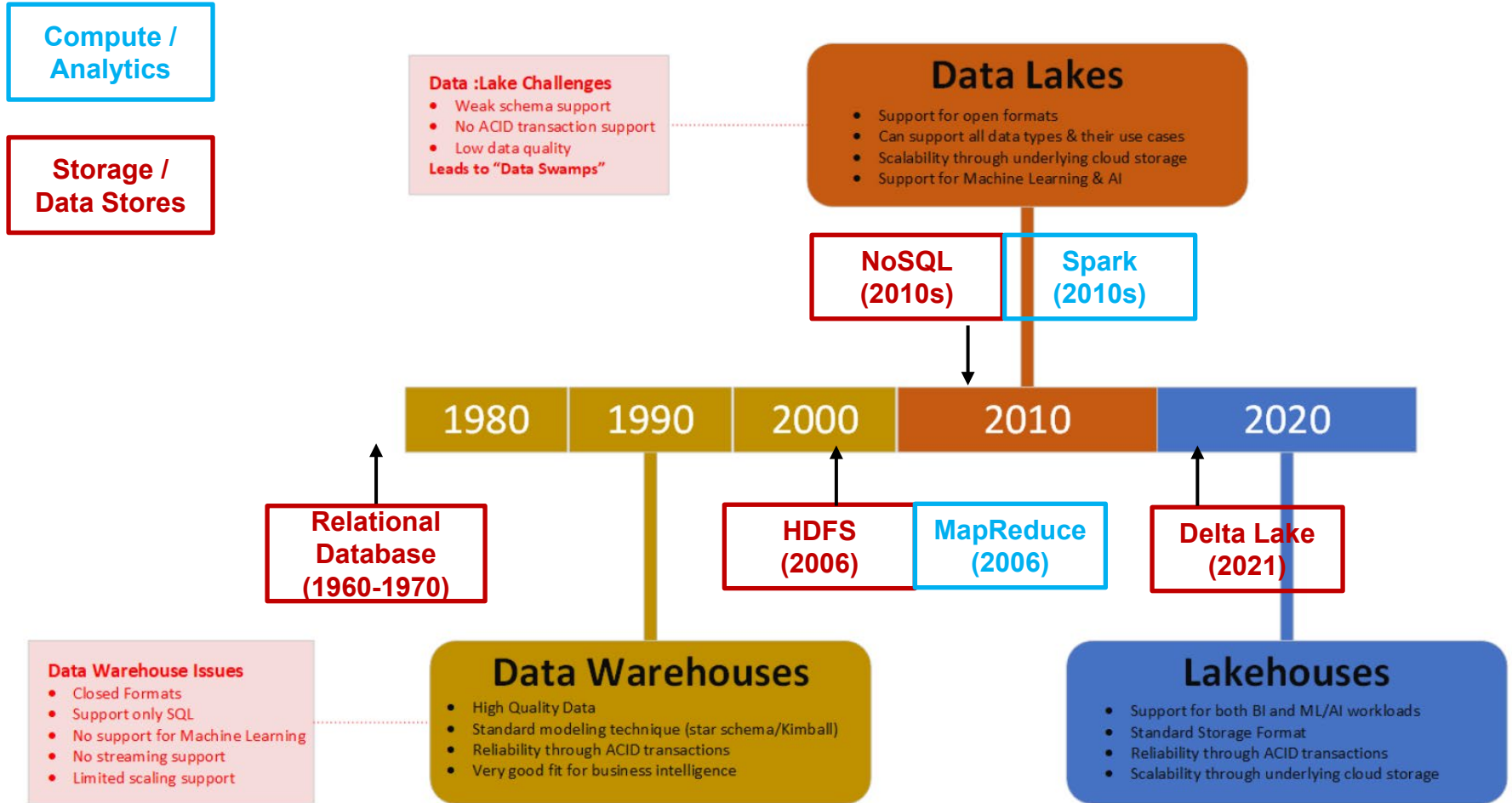
CS4225/CS5425 Big Data Systems for Data Science

Delta Lake

Ai Xin
School of Computing
National University of Singapore
aixin@comp.nus.edu.sg



Evolution of Data Architectures

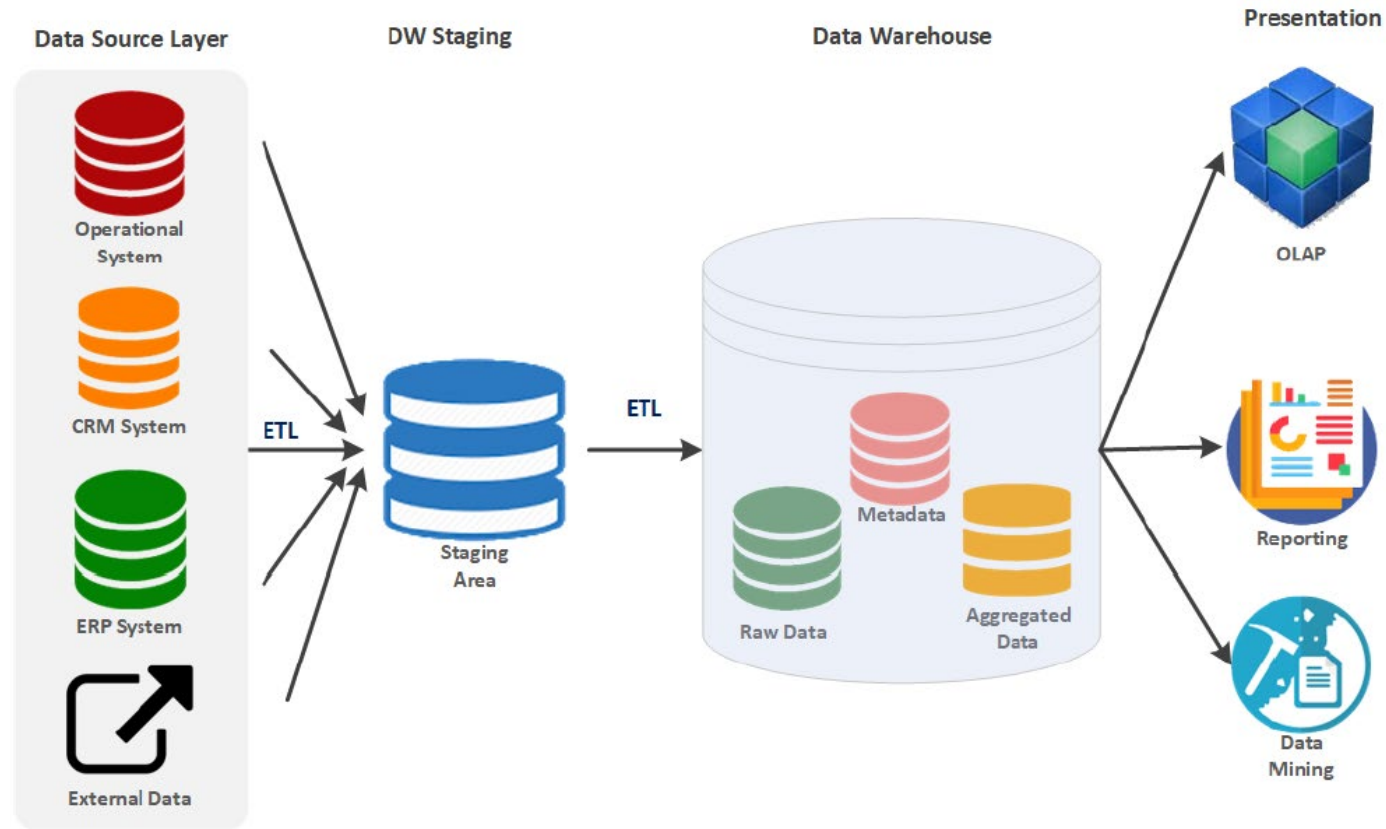


Database

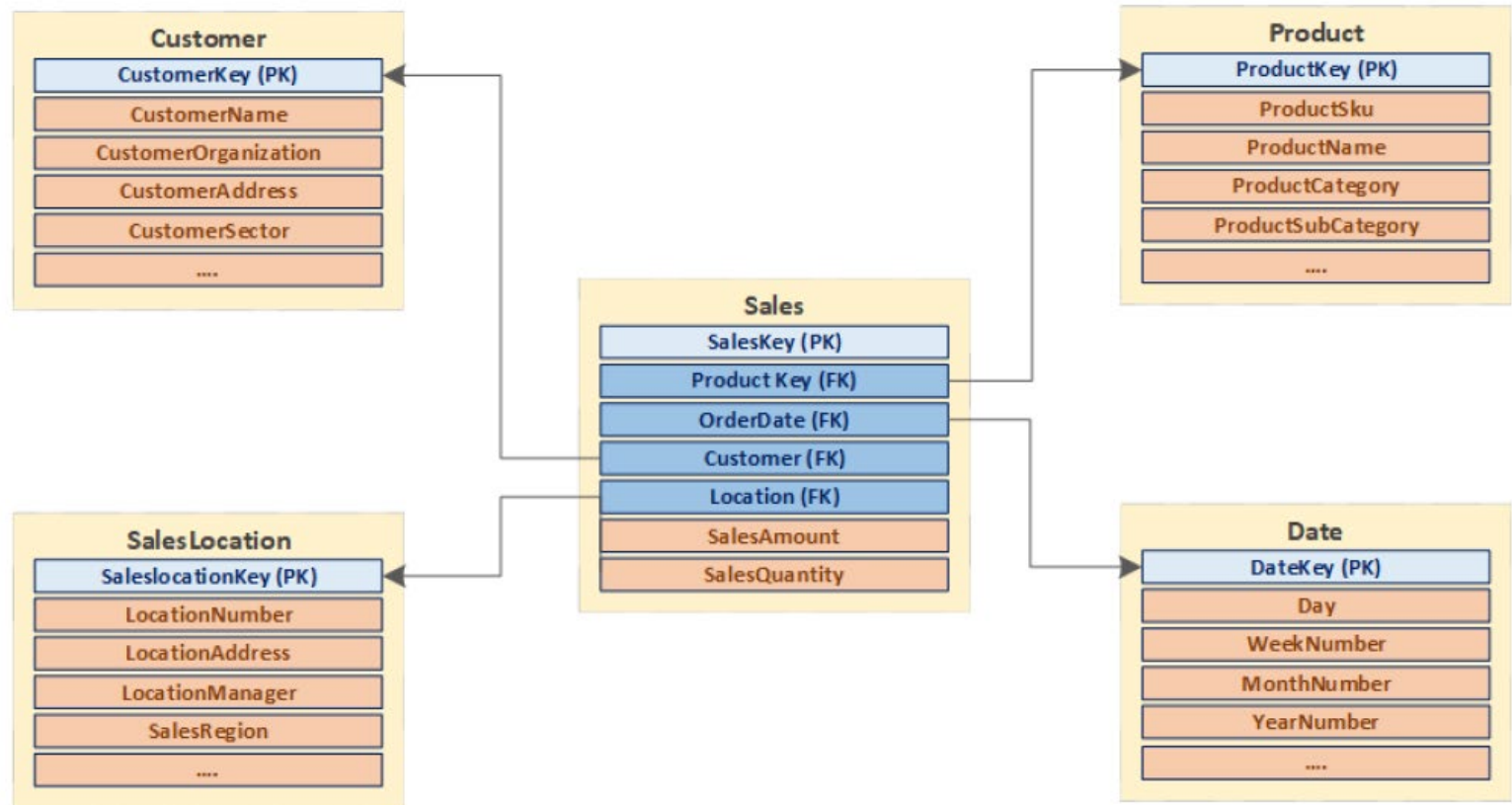
- Designed to store structured data (i.e. table)
- Can be read through SQL queries
- Data adhere to a strict schema
 - allows database management system to heavily co-optimize data storage and processing through an optimized query processing engines
- Very fast computation and strong transactional ACID guarantees on read/write operations
 - Atomicity, Consistency, Isolation, Durability (ACID)
- OLTP vs. OLAP
 - Online transaction processing (OLTP): traditional databases
 - Online analytical processing (OLAP): Data Warehouse

Data Warehouse

- a central relational repository of integrated, historical data from multiple data sources



Dimensional Modelling

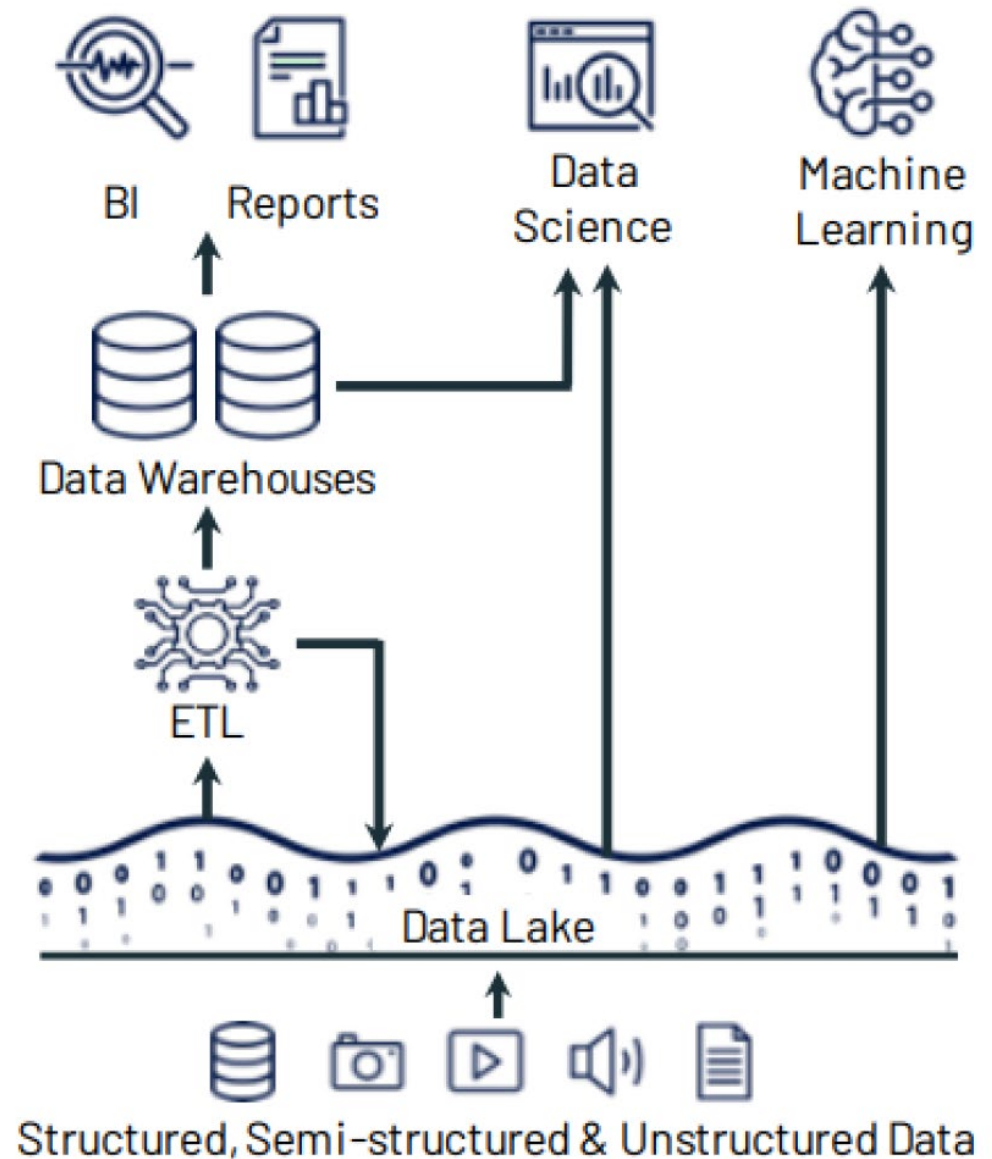


Data Warehouse Benefits & Challenges

- Served the business community well
 - Store large amounts of historical data from different sources
 - Very reliable with strong transactional ACID guarantees
 - Modeled with standard star-schema modeling techniques
 - Ideally suited for business intelligence and reporting
- Big Data Trends (Volume, Velocity, Variety, Veracity)
 - Growth in data sizes
 - Growth in the diversity of analytics
- Data Warehouses have a hard time addressing Four Vs
 - Extremely expensive to scale out
 - Do not support non-SQL based analytics very well

Data Lake

- a cost-effective central repository to store data at any scale
- a distributed storage solution, runs on commodity hardware, and easily scales out horizontally
- data is saved as files with open formats
 - any processing engine can read and write them using standard APIs



Data lakes

- decouples the distributed storage system from the distributed compute system
 - Allows each system to scale out as needed by the workloads
- Organizations build their data lakes by independently choosing
 - Storage system: HDFS, S3, Cloud and etc.
 - File format:
 - Structured: Parquet, ORC
 - semi-structured: JSON
 - unstructured formats: text, images, audio, video
 - Computing / Processing engine(s):
 - batch processing engine: Spark, Presto, Apache Hive
 - stream processing engine: Spark, Apache Flink
 - machine learning library: Spark MLlib, scikit-learn, R

Data Lakes

○ Pros

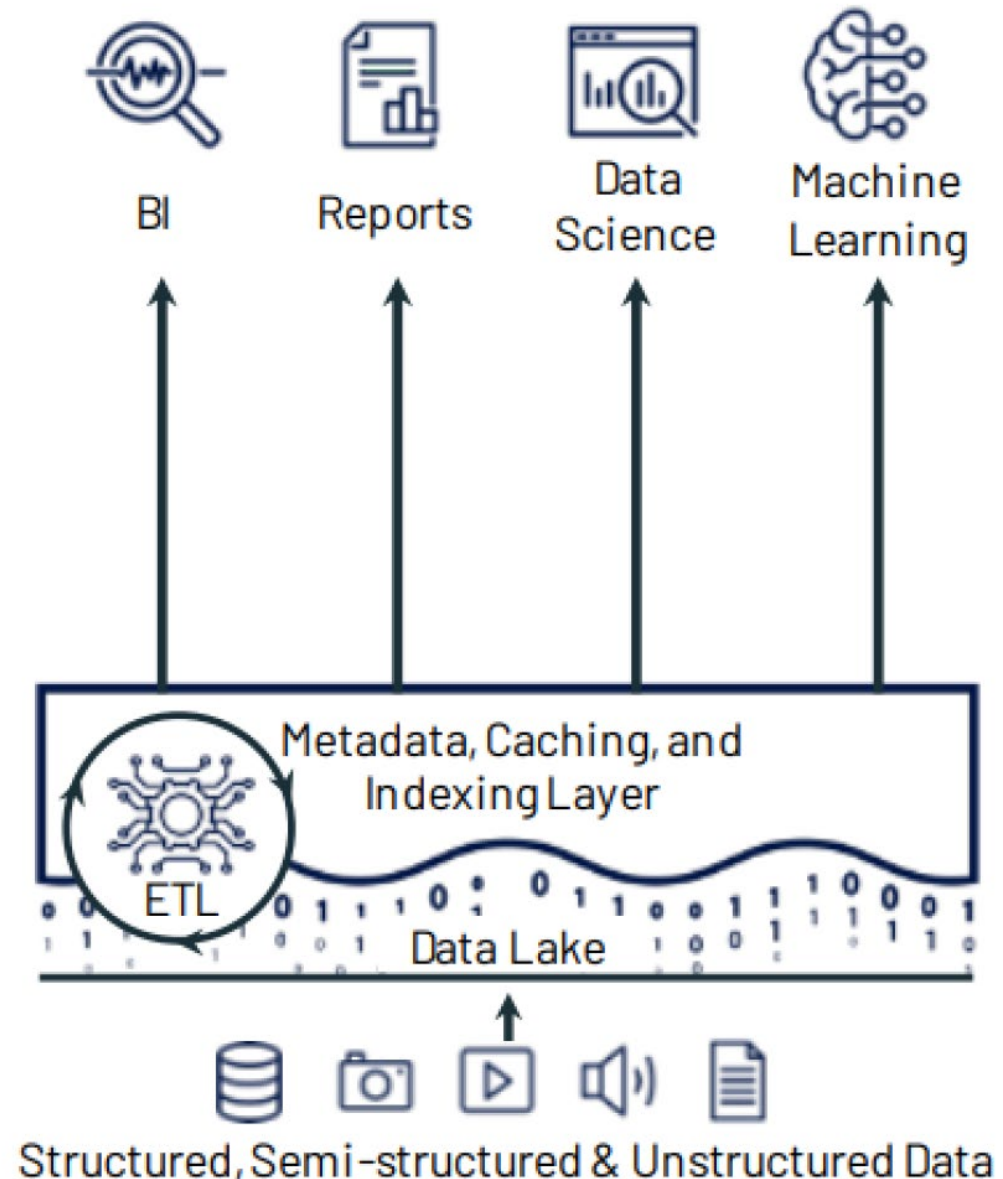
- Flexibility on choosing storage, data format and processing engines
- A much cheaper solution than databases → explosive growth of the big data ecosystem

○ Cons:

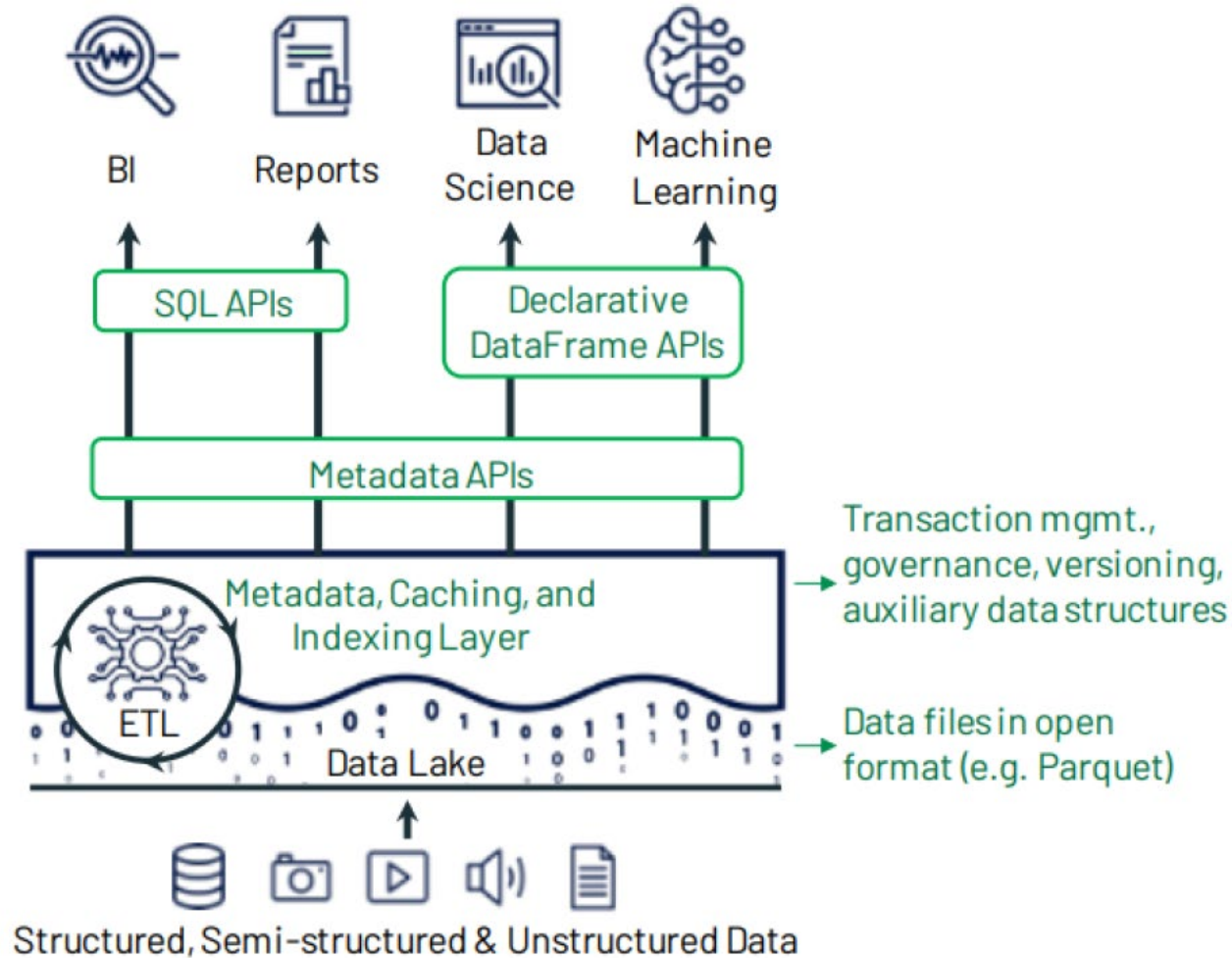
- Fail to provide ACID guarantees
- Building and maintaining an effective data lake requires expert skills
- Easy to ingest data but very expensive to transform data to deliver business values
- Data quality issues due to the lack of schema enforcement

Data Lakehouse

- A system merges both data lake and warehouse:
 - The flexibility, low cost, and scale of a data lake
 - The data management and ACID transactions of data warehouses
- A good fit for both users:
 - Business intelligence
 - Machine Learning/AI
- Especially good match for cloud environment
 - with separate storage and computing resources



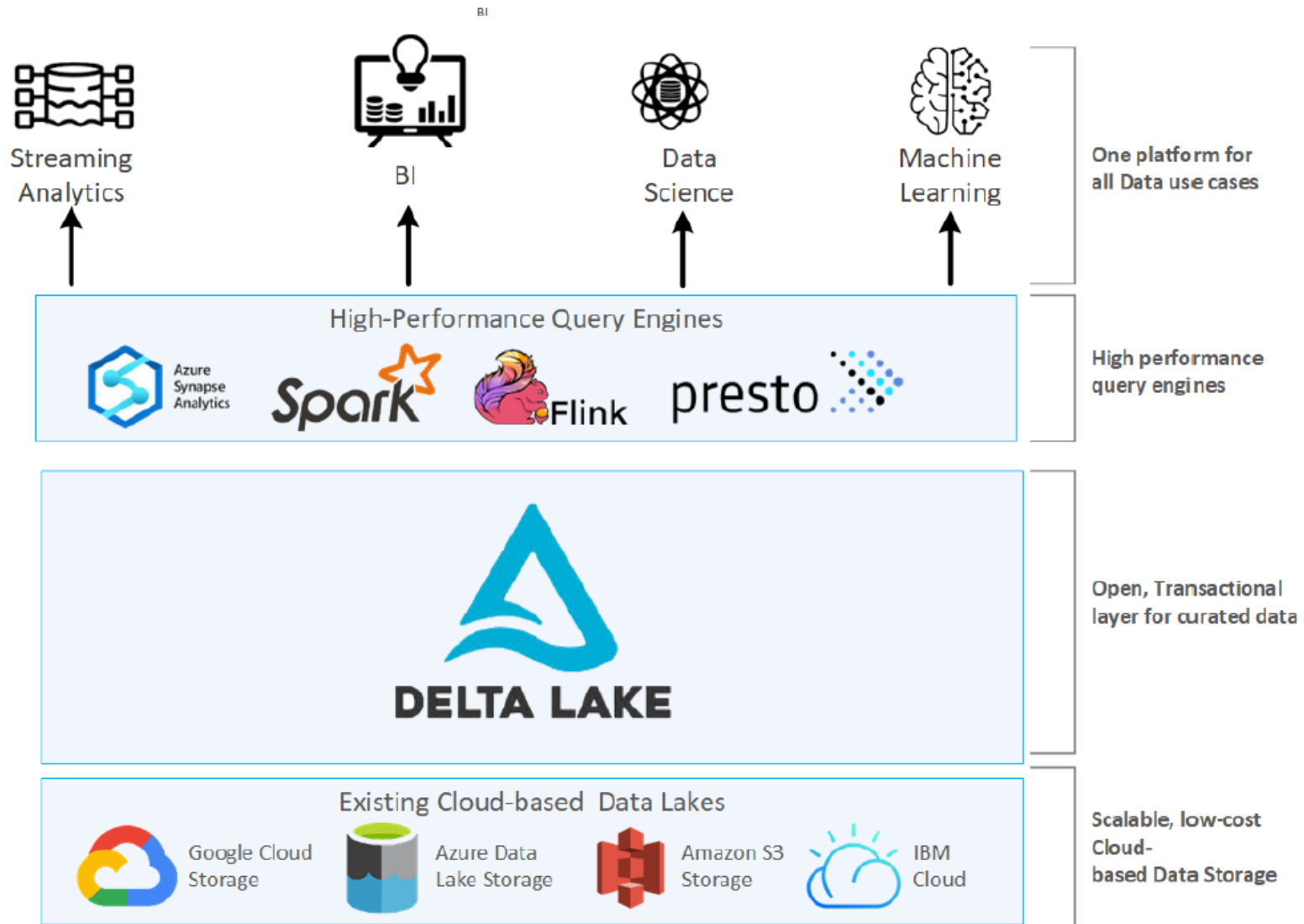
Data Lakehouse implementation



Delta Lake

- the metadata, caching and indexing layer on top of a data lake storage that provides an abstraction level to serve ACID transactions and other management features
 - Transactional ACID guarantees
 - Full DML (Data Manipulation Language) support
 - Audit History
 - Unification of batch and streaming into one processing model
 - Schema enforcement and evolution
 - Rich metadata support and scaling

Data lakehouse Layered Architecture



Delta Lake Format


- a standard Parquet file with additional metadata
- Parquet Files
 - Column oriented: perform compression on a column-by-column basis
 - Open source
 - Self-describing: actual data + metadata (schema & file structure)


```
1 data = spark.range(0, 100)
2 data.write.format("parquet") \
3   .mode("overwrite") \
4   .save('/tmp/parquetData')
```


▼ (1) Spark Jobs


▼ Job 10 [View](#) (Stages: 1/1)


Stage 17: 8/8 ⓘ


▼  data: pyspark.sql.dataframe.DataFrame
id: long


 parquetData ▼


 _committed_347056881252188848▼


 _started_347056881252188848 ▼


 _SUCCESS ▼


 part-00000-tid-347056881252188...▼


 part-00001-tid-347056881252188...▼


 part-00002-tid-347056881252188...▼

 part-00003-tid-347056881252188...▼

 part-00004-tid-347056881252188...▼

 part-00005-tid-347056881252188...▼

 part-00006-tid-347056881252188...▼

 part-00007-tid-347056881252188...▼

Demo_4

```
1 data = spark.range(0, 100)
2 data.write.format("delta") \
3   .mode("overwrite") \
4   .save('/tmp/deltaData')
```

▼ (5) Spark Jobs

- ▼ Job 114 [View](#) (Stages: 1/1)
Stage 180: 8/8 ⓘ
- ▶ Job 117 [View](#) (Stages: 1/1)
- ▶ Job 118 [View](#) (Stages: 1/1, 1 skipped)
- ▶ Job 119 [View](#) (Stages: 1/1, 1 skipped)
- ▼ Job 120 [View](#) (Stages: 1/1, 2 skipped)
Stage 186: 0/8 ⓘ skipped
Stage 187: 0/1 ⓘ skipped
Stage 188: 1/1 ⓘ

/tmp/deltaData/_delta_log

Q Prefix search

📁 _delta_log ▼

📄 part-00000-ec087bf5-71c8-428e-8...
📄 part-00001-fd744b95-2ca9-4dfd-b...
📄 part-00002-42398641-f053-4a12-...
📄 part-00003-e5703d16-1f46-4dcf-8...
📄 part-00004-26eb06c5-7cbe-40a6-...
📄 part-00005-482a0972-ce7c-4e3b-...
📄 part-00006-88bd5423-4af8-4bc2-...
📄 part-00007-c51da7d7-5055-41ea-...

Q Prefix search

📄 .s3-optimization-0 ▼
📄 .s3-optimization-1 ▼
📄 .s3-optimization-2 ▼
📄 00000000000000000000000000000000.crc ▼
📄 00000000000000000000000000000000.json ▼

The Delta Lake Transaction Log (DeltaLog)

- The transaction log is an ordered record of every transaction made against a Delta table since it was created.
- It acts as a single source of truth and tracks all changes made to the table.
- The main goal is to enable multiple readers and writers to operate on a given version of a dataset simultaneously.
- It is at the core of many important features
 - ACID transactions
 - Spark looks at the transaction log to get the latest version of the table
 - If an operation is not recorded in the transaction log, it never happened.
 - Scalable metadata handling
 - Time travel

Breaking down Transactions into Atomic Commits

- List of possible actions in a transaction log entry

Action	Description
Add file	Adds a file
Remove file	Removes a file
Update Metadata	Updates the table's metadata (e.g., changing the table or file's name, schema, or partitioning). The first transaction log entry for a table or file will always contain an Update Metadata action with the schema, the partition columns and other information.
Set transaction	Records that a structured streaming job has committed a micro-batch with the given stream ID. For more information, see Chapter X: Streaming
Change Protocol	Enables new features by switching the Delta Lake transaction log to the newest software protocol.
Commit Info	Contains information about the commit, which operation was made, from where, and at what time. Every transaction log entry will contain a Commit Info action.

- Example: a user creates a transaction to add a new column to a table and then adds data to it
 1. Update metadata – change the schema to include the new column.
 2. Add file – for each new file added

Delta Lake Example 1: Practical_5a

- updates and the transaction log

Step	Code	Parquet Files Written	JSON files in _delta_log																																								
<ul style="list-style-type: none">Read 00.jsonInclude part-0Include part-1 <ul style="list-style-type: none">Read 01.jsonInclude part-2 <ul style="list-style-type: none">Read 02.jsonRemove part-0Include part-3 Final Result: <ul style="list-style-type: none">part-1,part-2part-3 are included in latest data	<pre>df .coalesce(2) .write.format("delta") .save(DATALAKE_PATH)</pre> <pre>df .coalesce(1) .write.format("delta") .mode("append") .save(DATALAKE_PATH)</pre> <pre>deltaTable = DeltaTable \ .forPath(spark, DATALAKE_PATH) deltaTable.update(condition = col("patientId") == 1, set = { 'name': lit("p11")})</pre>	<table><thead><tr><th>patientId</th><th>name</th><th>patientId</th><th>name</th></tr></thead><tbody><tr><td>1</td><td>P1</td><td>3</td><td>P3</td></tr><tr><td>2</td><td>P2</td><td>4</td><td>P4</td></tr></tbody></table> <p>part-0.parquet part-1.parquet</p> <table><thead><tr><th>patientId</th><th>name</th></tr></thead><tbody><tr><td>5</td><td>P5</td></tr><tr><td>6</td><td>P6</td></tr></tbody></table> <p>part-2.parquet</p> <table><thead><tr><th>patientId</th><th>name</th></tr></thead><tbody><tr><td>1</td><td>P11</td></tr><tr><td>2</td><td>P2</td></tr></tbody></table> <p>part-3.parquet</p>	patientId	name	patientId	name	1	P1	3	P3	2	P2	4	P4	patientId	name	5	P5	6	P6	patientId	name	1	P11	2	P2	<table><thead><tr><th>Operation</th><th>File Name</th></tr></thead><tbody><tr><td>Add</td><td>part0</td></tr><tr><td>Add</td><td>part1</td></tr></tbody></table> <p>00000.json</p> <table><thead><tr><th>Operation</th><th>File Name</th></tr></thead><tbody><tr><td>Add</td><td>part2</td></tr></tbody></table> <p>00001.json</p> <table><thead><tr><th>Action</th><th>Part Name</th></tr></thead><tbody><tr><td>Remove</td><td>part0</td></tr><tr><td>Add</td><td>part3</td></tr></tbody></table> <p>00002.json</p>	Operation	File Name	Add	part0	Add	part1	Operation	File Name	Add	part2	Action	Part Name	Remove	part0	Add	part3
patientId	name	patientId	name																																								
1	P1	3	P3																																								
2	P2	4	P4																																								
patientId	name																																										
5	P5																																										
6	P6																																										
patientId	name																																										
1	P11																																										
2	P2																																										
Operation	File Name																																										
Add	part0																																										
Add	part1																																										
Operation	File Name																																										
Add	part2																																										
Action	Part Name																																										
Remove	part0																																										
Add	part3																																										

Delta Lake Example 2: Practical_5b

○ Checkpoint File

Step	Code	Parquet Files Written	JSON files in <code>_delta_log</code>												
1	<pre>df .coalesce(1) .write .format("delta") .save(DATALAKE_PATH)</pre>	<div>part-00000.parquet</div>	<table><thead><tr><th>Action</th><th>Part Name</th></tr></thead><tbody><tr><td>Add</td><td>part-00000</td></tr></tbody></table> <div>00000.json</div>	Action	Part Name	Add	part-00000								
Action	Part Name														
Add	part-00000														
2	<pre># Loop from 0..9 for index in range(9): # create a patient tuple patientId = 10 + index t = (patientId, f"Patient {patientId}", "Phoenix") # Create and write the dataframe df = spark.createDataFrame([t], columns) df .write .format("delta") .mode("append") .save(DATALAKE_PATH)</pre>	<div>part-00001.parquet</div> <div>part-00002.parquet</div> <div>...</div> <div>...</div> <div>part-00009.parquet</div>	<table><thead><tr><th>Action</th><th>Part Name</th></tr></thead><tbody><tr><td>Add</td><td>part-00001</td></tr></tbody></table> <div>00001.json</div> <table><thead><tr><th>Action</th><th>Part Name</th></tr></thead><tbody><tr><td>Add</td><td>part-00002</td></tr></tbody></table> <div>00002.json</div> <div>...</div> <div>...</div> <table><thead><tr><th>Action</th><th>Part Name</th></tr></thead><tbody><tr><td>Add</td><td>part-00009</td></tr></tbody></table> <div>00009.json</div>	Action	Part Name	Add	part-00001	Action	Part Name	Add	part-00002	Action	Part Name	Add	part-00009
Action	Part Name														
Add	part-00001														
Action	Part Name														
Add	part-00002														
Action	Part Name														
Add	part-00009														
3	<pre>patientId = 100 t = (patientId, f"Patient {patientId}", "Phoenix") df = spark.createDataFrame([t], columns) df .write .format("delta") .mode("append") .save(DATALAKE_PATH)</pre>	<div>part-00010.parquet</div>	<table><thead><tr><th>Action</th><th>Part Name</th></tr></thead><tbody><tr><td>Add</td><td>part-00010</td></tr></tbody></table> <div>00010.json</div> <div>00010.checkpoint.parquet</div>	Action	Part Name	Add	part-00010								
Action	Part Name														
Add	part-00010														
4	<pre>for index in range(2): patientId = 200 + index t = (patientId, f"Patient{patientId}", "Phoenix") df = spark.createDataFrame([t], columns) df .write .format("delta") .mode("append") .save(DATALAKE_PATH)</pre>	<div>part-00012.parquet</div> <div>part-00013.parquet</div>	<table><thead><tr><th>Action</th><th>Part Name</th></tr></thead><tbody><tr><td>Add</td><td>part-00012</td></tr></tbody></table> <div>00001.json </div> <table><thead><tr><th>Action</th><th>Part Name</th></tr></thead><tbody><tr><td>Add</td><td>part-00013</td></tr></tbody></table> <div>00002.json</div>	Action	Part Name	Add	part-00012	Action	Part Name	Add	part-00013				
Action	Part Name														
Add	part-00012														
Action	Part Name														
Add	part-00013														

Acknowledgements

- CS4225 slides by He Bingsheng and Bryan Hooi
- Bennie Haelen, “Delta Lake: Up & Running”
- Jules S. Damji, Brooke Wenig, Tathagata Das & Denny Lee, “Learning Spark: Lightning-Fast Data Analytics”
- Bill Chambers, Matei Zaharia, “Spark: The Definitive Guide”