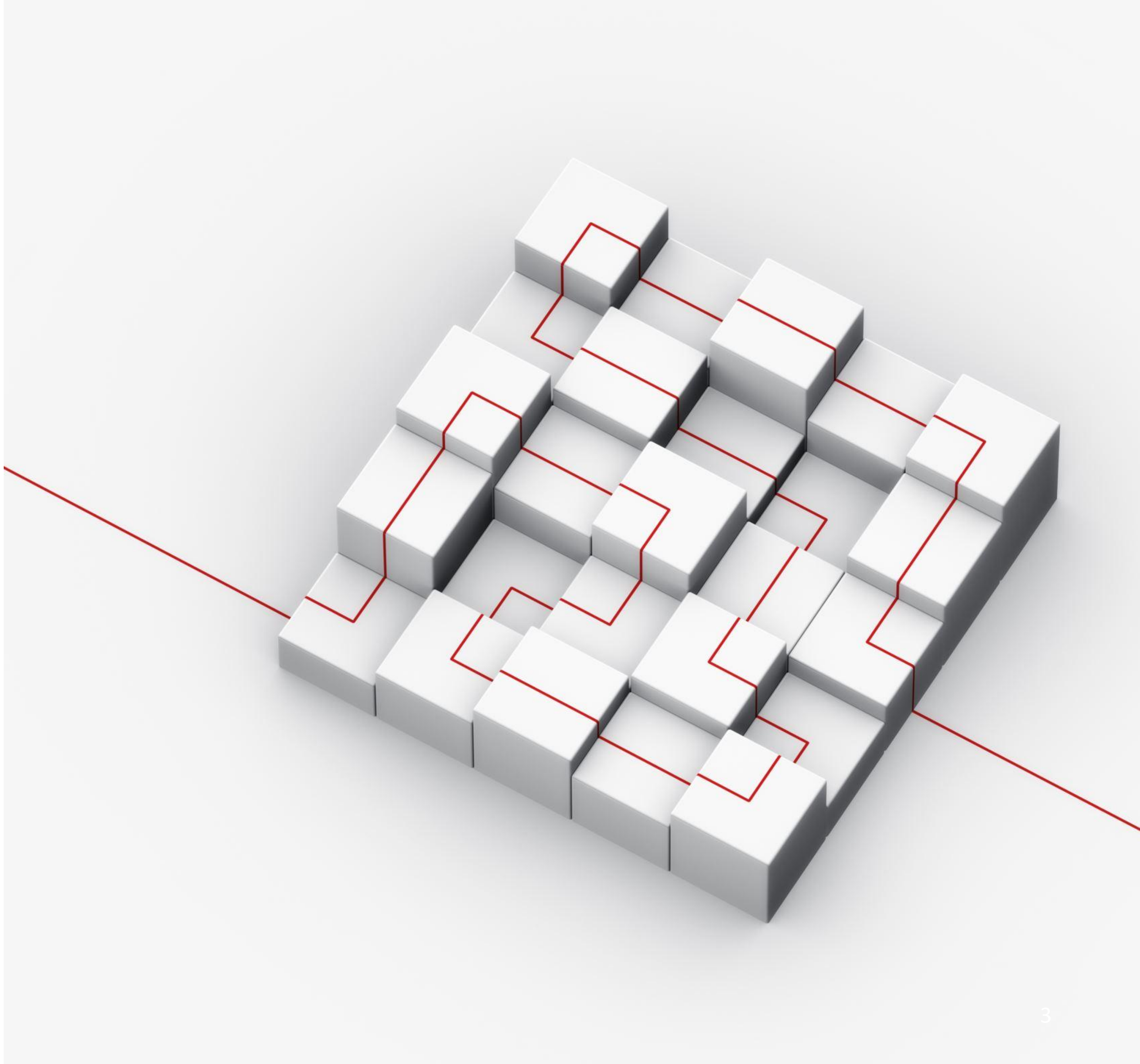
Introduction to Data Engineering and Data Ingestion Strategies

MC ID: 7124506 | Parveen KR | Parveen.R@hotmail.com

Agenda

- Foundations of Data Engineering
- Core Responsibilities of a Data Engineer
- Data Modeling Approaches
- OLTP vs OLAP: Analytical and Transactional Systems
- ETL vs ELT and Data Ingestion Methods
- Batch vs Streaming Data Processing
- Extracting Data from APIs, Databases, and Object Storage
- Data Formats and Compression
- Real-Time Data Ingestion and Streaming
- Medallion Architecture in Data Lakes
- Incremental Loading and Change Data Capture (CDC)
- Streaming Correctness: Watermarking and Checkpointing
- Data Quality, Validation, and Handling Corrupt Records
- Schema Evolution and Enforcement

Foundations of Data Engineering



MC ID: 7124506 | Parveen KR | Parveen.R@hotmail.com

DEFINING DATA ENGINEERING AND ITS VALUE CHAIN

Definition of Data Engineering

Data engineering builds systems to ingest, store, transform, and deliver reliable data for analytics and AI applications.

Data Sources

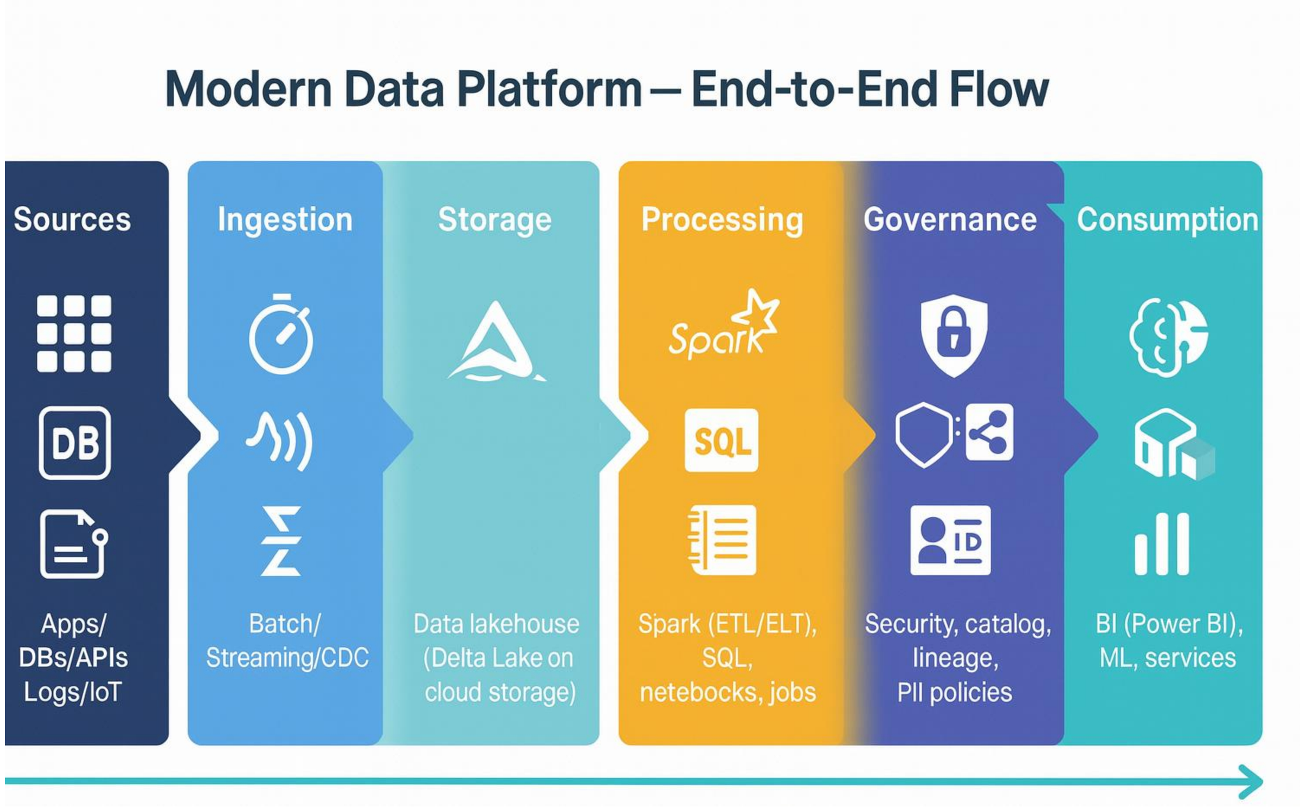
Data originates from diverse sources such as applications, databases, APIs, logs, and IoT devices.

Ingestion and Storage

Data ingestion uses batch, streaming, and change data capture methods; stored in cloud data lakehouses like Delta Lake.

Processing and Governance

Data is processed with Spark and SQL; governed via security, catalog, lineage, and privacy policies.



KEY NON-FUNCTIONAL GOALS IN DATA ENGINEERING

Data Freshness and SLAs

Ensuring data freshness is critical, often measured by Service Level Agreements (SLAs) to guarantee timely delivery.

Reliability and SLOs

Reliability targets are defined by Service Level Objectives (SLOs) to maintain consistent system performance.

Cost Efficiency

Optimizing resources and operations is essential to achieve cost efficiency in data engineering projects.

Observability and Auditability

Observability and auditability ensure transparent monitoring and traceability of data pipelines and processes.

Modern Data Architecture Example

Data Ingestion

Event Hub and Kafka enable real-time data streaming for scalable ingestion of diverse data sources.

Bronze Layer Storage

Raw data is stored in Bronze Delta tables providing an immutable and scalable data lake foundation.

Silver Layer Processing

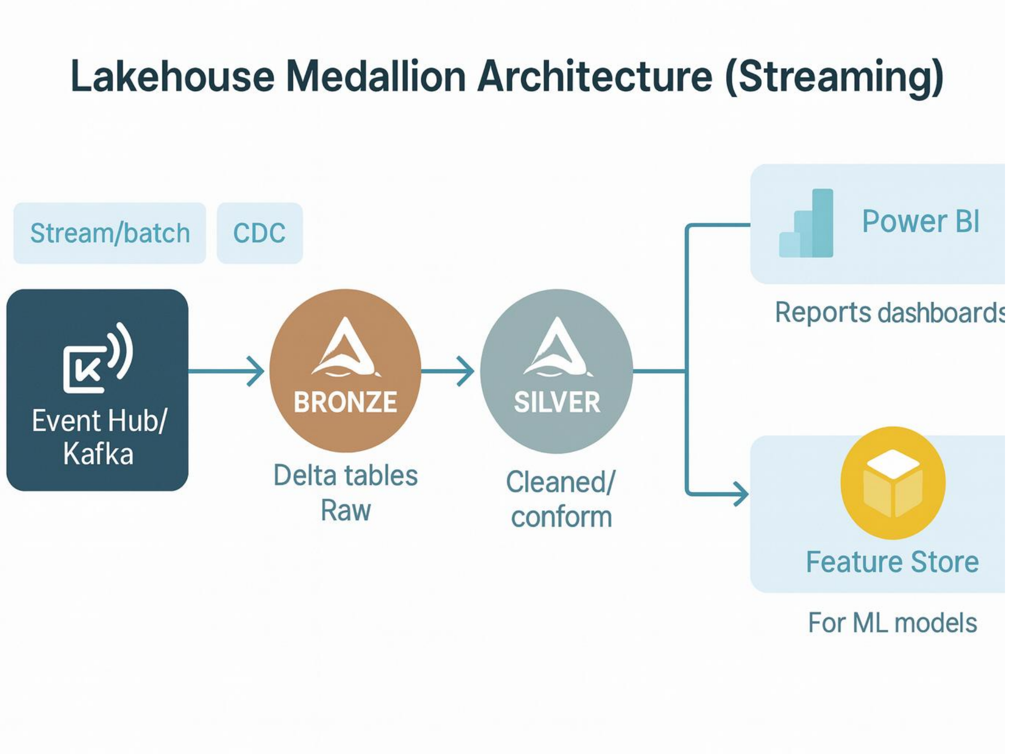
Data is cleaned and conformed in the Silver layer to create reliable and consistent datasets.

Gold Layer Aggregation

Business aggregates are created in the Gold layer for high-level analytics and reporting.

Analytics and Machine Learning

Power BI dashboards and Feature Store support visualization and machine learning model development.



Core Responsibilities of a Data Engineer

ESSENTIAL ROLES IN A LAKEHOUSE ENVIRONMENT

Pipeline Design and Orchestration

Data engineers design and manage workflows, handle retries, and manage job dependencies efficiently.

Data Modeling Techniques

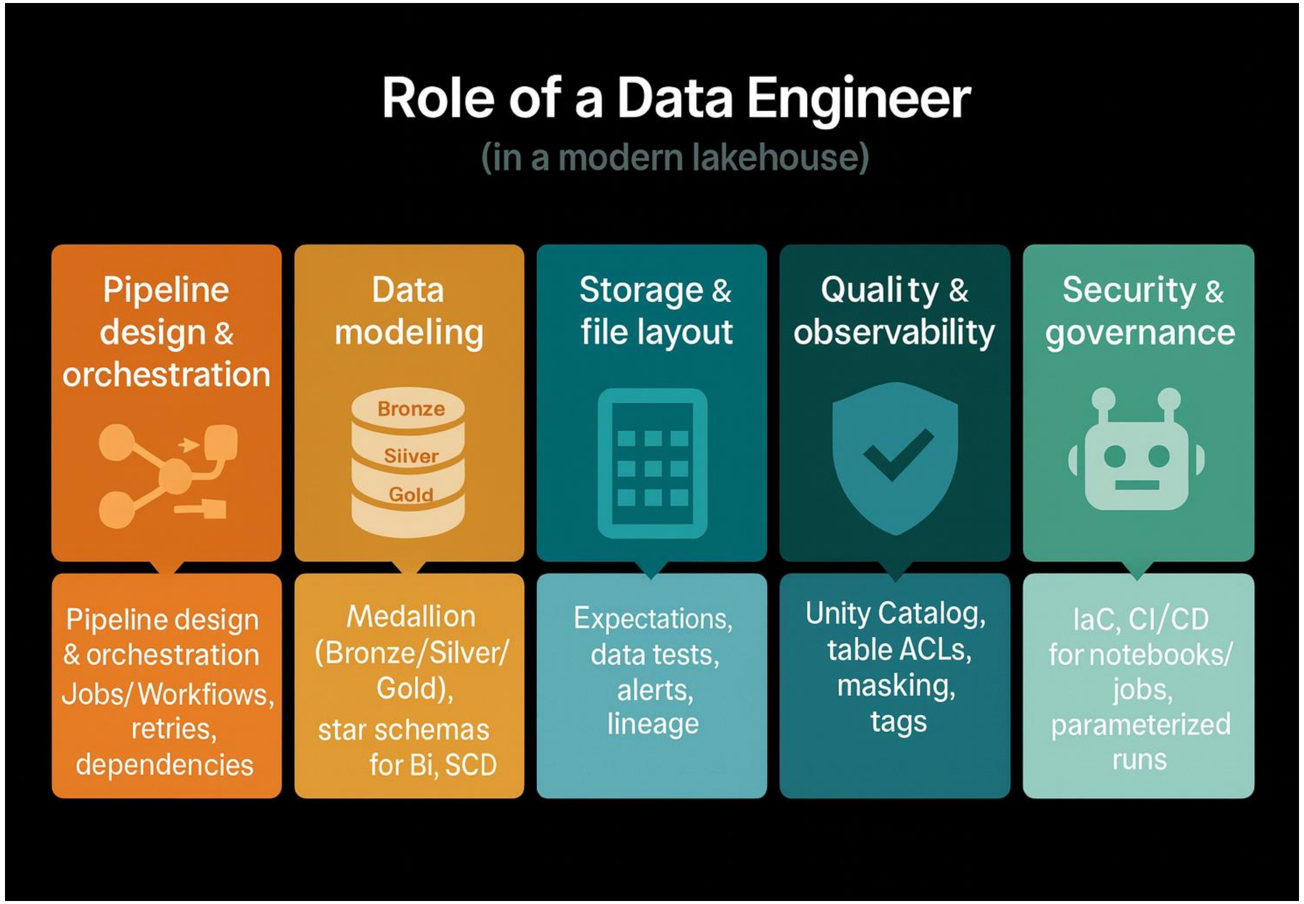
They implement Medallion architecture and star schemas to optimize business intelligence and slowly changing dimensions.

Storage Optimization and File Layout

Optimizing partitions, file sizes, and using Z-Ordering to improve query performance and storage efficiency.

Quality, Security, and Automation

Ensures data quality with tests and alerts, enforces security policies, and automates deployments with IaC and CI/CD pipelines.



Data Modeling Approaches

CONCEPTUAL, LOGICAL, AND PHYSICAL DATA MODELS

Conceptual Data Model

Represents business entities and their relationships, focusing on high-level structure.

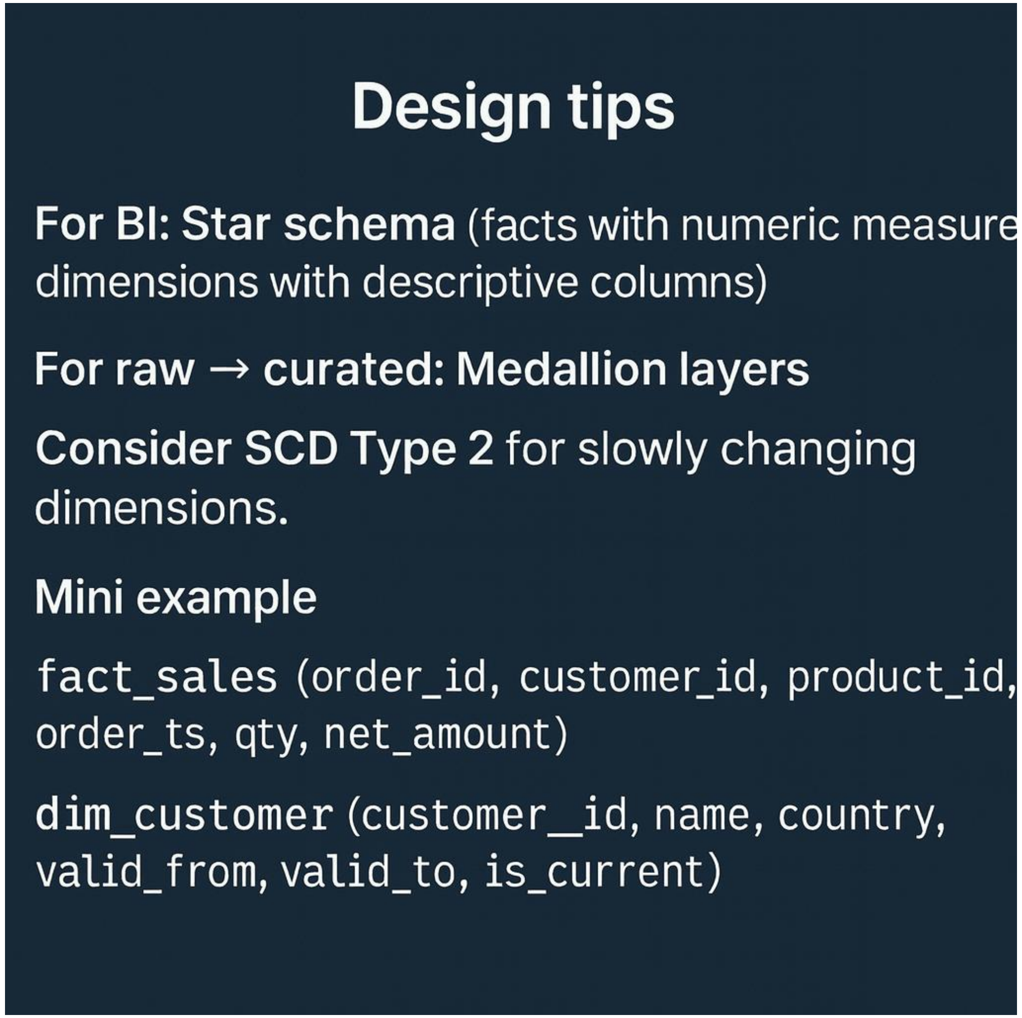
Logical Data Model

Defines attributes, keys, and constraints in a platform-agnostic manner, detailing data structure.

Physical Data Model

Specifies tables, partitions, file formats, and indexes tailored to specific platforms.

DESIGN TIPS AND MINI EXAMPLE FOR BI AND MEDALLION LAYERS



Star Schema Design

Use star schema for BI combining fact tables with numeric measures and descriptive dimension tables.

Medallion Layers Concept

Implement medallion layers for data refinement from raw to curated stages to improve quality and governance.

Slowly Changing Dimensions

Apply SCD Type 2 to track historical changes in dimension data ensuring accurate BI reporting over time.

Mini Example Schema

Example includes fact_sales with order details and dim_customer capturing customer info and validity periods.

Example.
fact_sales (order_id, customer_id, product_id, order_ts, qty, net_amount)
dim_customer (customer_id, name, country, valid_from, valid_to, is_current)

OLTP vs OLAP: Analytical and Transactional Systems

Comparing OLTP and OLAP Systems

ASPECT	OLTP (SYSTEMS OF RECORD)	OLAP (ANALYTICS)
Workload	Short transactions	Scans/Aggregations
Schema	Normalized (3NF)	Denormalized (Star/Snowflake)
Queries	INSERT/UPDATE/DELETE	SELECT/GROUP BY/JOIN
Optimization	Low latency per row	High throughput scans
Store	RDBMS	Lakehouse/Warehouse

BRIDGE PATTERNS FOR DATA MOVEMENT

Data Replication Using CDC

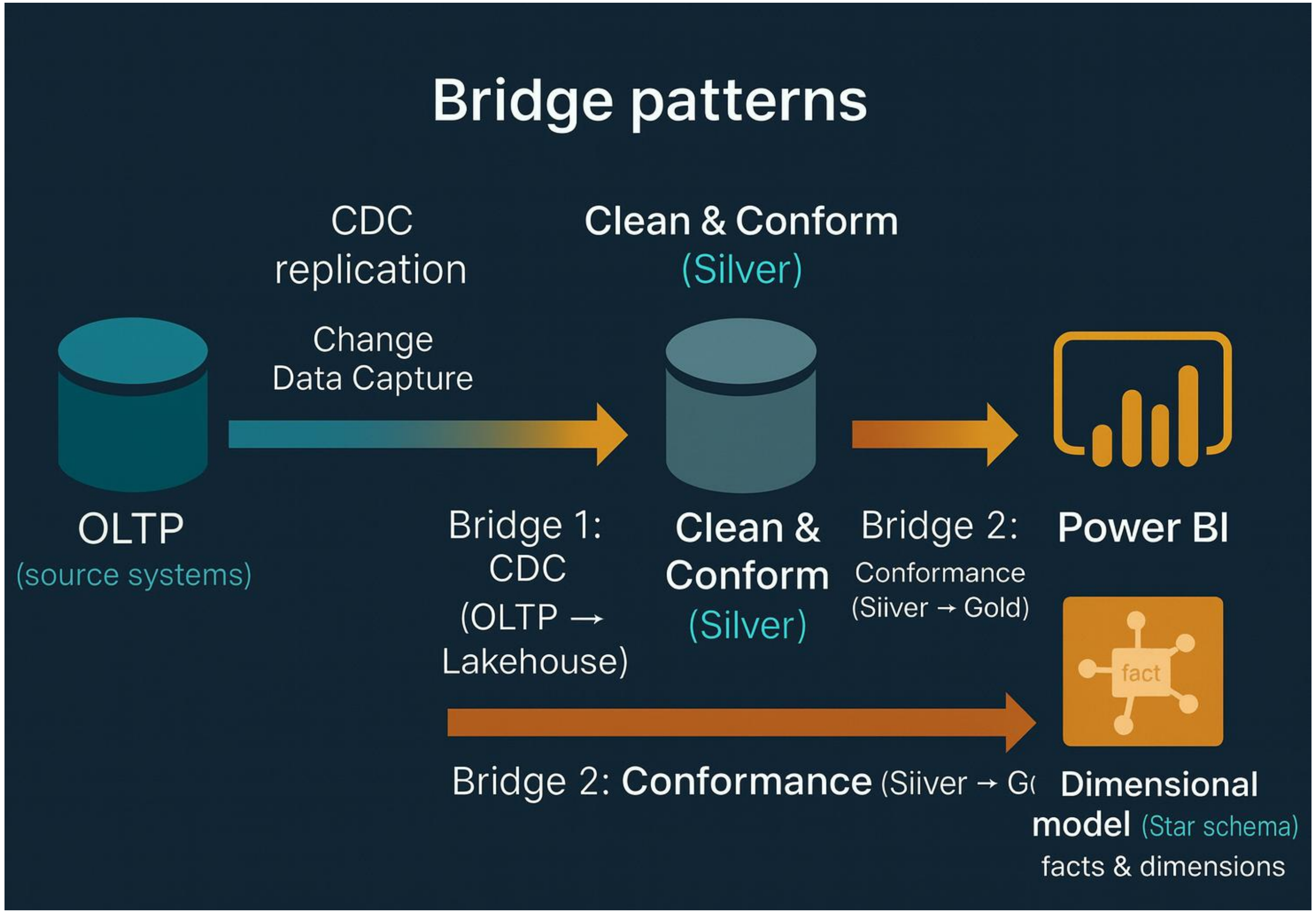
Change Data Capture enables real-time replication from OLTP systems to data lakes efficiently and accurately.

Data Cleaning and Conforming

Data is cleaned and conformed to a dimensional model to ensure consistency and usability for analytics.

Gold Layer for BI

The gold data layer is created for Business Intelligence, enabling reliable and trusted analytic reporting.



ETL vs ELT and Data Ingestion Methods

ETL VS ELT: APPROACHES AND BEST PRACTICES

ETL Approach

ETL involves transforming data before loading, typically used in legacy and on-premises systems.

ELT Approach

ELT loads data quickly into the lakehouse and transforms it there, ideal for cloud-scale environments.

Best Practice with Delta Lake

Using ELT with Delta Lake maintains data lineage, enables time travel, and simplifies reprocessing.

OVERVIEW OF DATA INGESTION METHODS

Bulk and Initial Load

One-time data ingestion for backfills or migrations, ideal for large datasets.

Incremental Loading

Loads only new or updated rows using timestamps or surrogate keys for efficiency.

Change Data Capture (CDC)

Captures inserts, updates, and deletes from source logs to maintain data consistency.

Streaming and Manual Ingestion

Streaming provides low-latency continuous ingestion; manual ingestion handles one-off uploads via UI.

Batch vs Streaming Data Processing

Batch and Streaming: Characteristics and Use Cases

BATCH	STREAMING
Scheduled (e.g., hourly)	Continuous (seconds)
Simple to operate	Harder (ordering/late data)
Suited for BI and backfills	Suited for fraud/monitoring/near real-time

SPARK STRUCTURED STREAMING MODES AND TRIGGERS

Streaming Modes

Spark supports micro-batch mode which is the most common streaming mode for processing data in batches.

Continuous Mode

Continuous mode offers low-latency streaming processing for niche applications needing faster data handling.

Trigger Types

Triggers like .trigger(once=True) handle backfills, while .trigger(processingTime='1 minute') enables near real-time streaming.

Extracting Data from APIs, Databases, and Object Storage

API EXTRACTION: BEST PRACTICES AND PYSPARK EXAMPLE

API Extraction Challenges

Effective extraction requires managing authentication, pagination, rate limits, retries, and exponential backoff to ensure data integrity.

Data Persistence Layers

Persist raw API payloads to Bronze storage before parsing and transforming data into structured Silver tables for analytics.

PySpark Extraction Example

Using PySpark to call APIs, convert JSON responses into DataFrames, and append data to Delta tables for scalable processing.

Sample Code:

```
import requests, json
url = "https://api.example.com/orders?page=1"
headers = {"Authorization": f"Bearer {token}"}
resp = requests.get(url, headers=headers, timeout=30);
resp.raise_for_status()
data = resp.json()
df = spark.createDataFrame(data["items"])
df.write.format("delta").mode("append").saveAsTable("bronze.orders_api")
```

DATABASE EXTRACTION WITH JDBC: TECHNIQUES AND EXAMPLE

Sample Code:

```
jdbc_url =
"jdbc:sqlserver://server.database.windows.net:1433;databaseName=sales"
props = {"user": "...", "password": "...", "driver":
"com.microsoft.sqlserver.jdbc.SQLServerDriver"}
df = (spark.read.format("jdbc")
.option("url", jdbc_url)
.option("dbtable", "dbo.Orders")
.option("fetchsize", 50000)
.options(**props)
.load())
```

Using JDBC for Data Extraction

JDBC enables efficient extraction of data from relational databases using standardized connections.

Predicate Pushdown Technique

Predicate pushdown improves performance by filtering data early during extraction based on conditions.

Incremental Data Loading

Incremental loading uses columns like last_modified_ts to fetch only updated data and reduce load time.

Example Spark JDBC Code

Spark can read databases via JDBC with options like fetch size and authentication properties for scalability.

OBJECT STORAGE INGESTION: AUTO LOADER AND STREAMING EXAMPLE

Object Storage Platforms

Supports multiple object storage platforms like ADLS, S3, and GCS accessible via secure credentials.

Auto Loader Features

Auto Loader automates file discovery and manages schema evolution for streaming data ingestion pipelines.

Streaming Data Ingestion

Example shows streaming CSV data ingestion using Spark structured streaming with checkpointing and Delta Lake.

Sample Code:

```
(spark.readStream.format("cloudFiles")
.option("cloudFiles.format", "csv")
.option("cloudFiles.schemaLocation",
"dbfs:/checkpoints/bronze_orders_schema")
.load("abfss://data@<storage>.dfs.core.windows.net/landing/orders/")
.writeStream.format("delta")
.option("checkpointLocation", "dbfs:/checkpoints/bronze_orders_ckpt")
.toTable("bronze.orders_files"))
```

Data Formats and Compression

Comparing Data Formats: Pros, Cons, and Use Cases

FORMAT	ROW/COLUMN	PROS	CONSIDERATIONS	USE CASES
CSV	Row	Simple, ubiquitous	No schema, larger	Legacy exports, small swaps
JSON	Row	Nested, flexible	Costly to parse, schema drift	API payloads
Avro	Row	Schema registry, evolvable	Row-oriented scans slower	Kafka messages, CDC logs
Parquet	Column	Compressed, predicate pushdown	Nested writes tricky	Analytics, Delta Lake base
ORC	Column	Hive-optimized	Ecosystem bias	Hadoop/Hive stacks
Protobuf/Thrift	Row (binary)	Compact, typed	Requires IDL, tooling	Microservices IPC

Real-Time Data Ingestion and Streaming

CORE CONCEPTS IN REAL-TIME INGESTION (KAFKA, KINESIS, PUB/SUB)

Data Topics and Partitions

Topics or streams organize data, while partitions enable parallel processing and scalability.

Offsets and Consumer Groups

Offsets track message positions; consumer groups allow multiple consumers to scale processing efficiently.

Exactly-Once Semantics

Ensures each message is processed only once using checkpointing and transaction mechanisms for accuracy.

SPARK KAFKA READER: IMPLEMENTATION EXAMPLE

Reading Stream from Kafka

Spark reads streaming data from Kafka topics using bootstrap servers and subscription options.

Parsing JSON Data

Incoming Kafka messages are parsed from JSON format into structured columns using a defined schema.

Writing to Delta Table

Parsed streaming data is written to a Delta Lake table with checkpointing to ensure fault tolerance.

Medallion Architecture in Data Lakes

BRONZE, SILVER, AND GOLD LAYERS EXPLAINED

Bronze Layer Characteristics

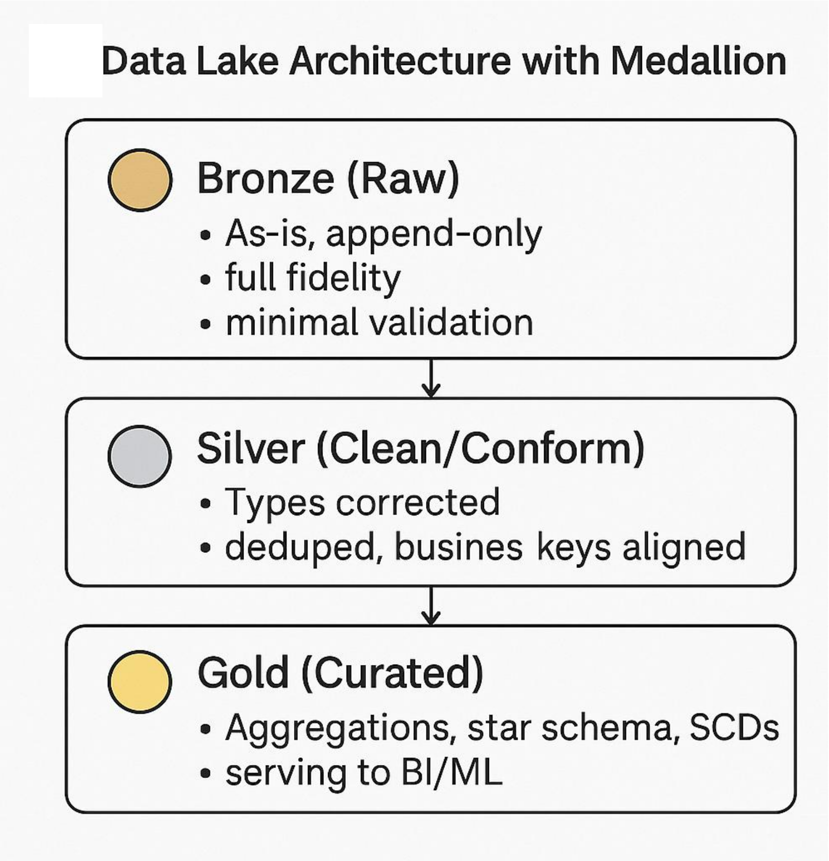
The Bronze layer contains raw, append-only data with full fidelity and minimal validation, capturing original information.

Silver Layer Processing

The Silver layer features cleaned and conformed data with corrected types, deduplication, and aligned business keys.

Gold Layer Purpose

The Gold layer includes curated data with aggregations and star schema, optimized for BI and machine learning applications.



BEST PRACTICES FOR MEDALLION ARCHITECTURE

Stable Tables and Immutable Schemas

Use stable table names and paths with immutable schemas in the Gold layer for consistency and reliability.

Time Travel for Auditing

Implement time travel to enable auditing and backfill of historical data efficiently and securely.

Data Optimization Techniques

Apply OPTIMIZE and ZORDER on frequently queried predicates and safely perform VACUUM to maintain performance.

Incremental Loading and Change Data Capture (CDC)

INCREMENTAL AND CDC STRATEGIES WITH DELTA LAKE

Incremental Data Processing

Uses timestamp watermarks to read new or updated rows since the last checkpoint for efficient data processing.

Upsert with MERGE

Delta Lake supports upsert operations combining insert and update using MERGE for data consistency.

Handling Deletes in CDC

Deletes are represented by flags or change types and applied through conditional DELETE statements in MERGE.

LOG-BASED AND TRIGGER-BASED CDC: MECHANISMS, PROS, AND CONS

Log-Based CDC Mechanism

Reads source transaction logs to publish detailed change events with before/after images and identifiers.

Log-Based CDC Pros and Cons

Offers low source load and real-time fidelity but involves infrastructure complexity and frequent schema changes.

Trigger-Based CDC Mechanism

Uses database triggers to write changes into shadow or audit tables for change data capture.

Trigger-Based CDC Pros and Cons

Easy and quick in small systems but adds load, is brittle at scale, and hard to manage.

Streaming Correctness: Watermarking and Checkpointing

ENSURING CORRECTNESS WITH WATERMARKS AND CHECKPOINTS

Watermark Concept

Watermarks set bounds on event-time lateness to ensure timely finalization of streaming aggregations.

Checkpointing Mechanism

Checkpoints store streaming state to guarantee exactly-once processing and fault tolerance.

Data Quality, Validation, and Handling Corrupt Records

TECHNIQUES FOR DATA VALIDATION AND QUALITY CHECKS

Schema and Type Checks

Ensure data conforms to defined schema and correct data types to maintain accuracy and consistency.

Constraint Enforcement

Apply row-level constraints like nulls, uniqueness, and referential integrity to ensure data quality.

Error Handling Strategies

Use fail-fast to stop jobs on errors or quarantine to divert bad data for later triage.

Monitoring and Metrics

Emit validation metrics to logs and monitoring tools to track data quality continuously.

STRATEGIES FOR HANDLING CORRUPT RECORDS AT SCALE

Corrupt Record Handling Modes

Use mode=PERMISSIVE to store corrupt rows in a special column for later inspection or mode=DROPMALFORMED to drop bad rows cautiously.

Error Capture with badRecordsPath

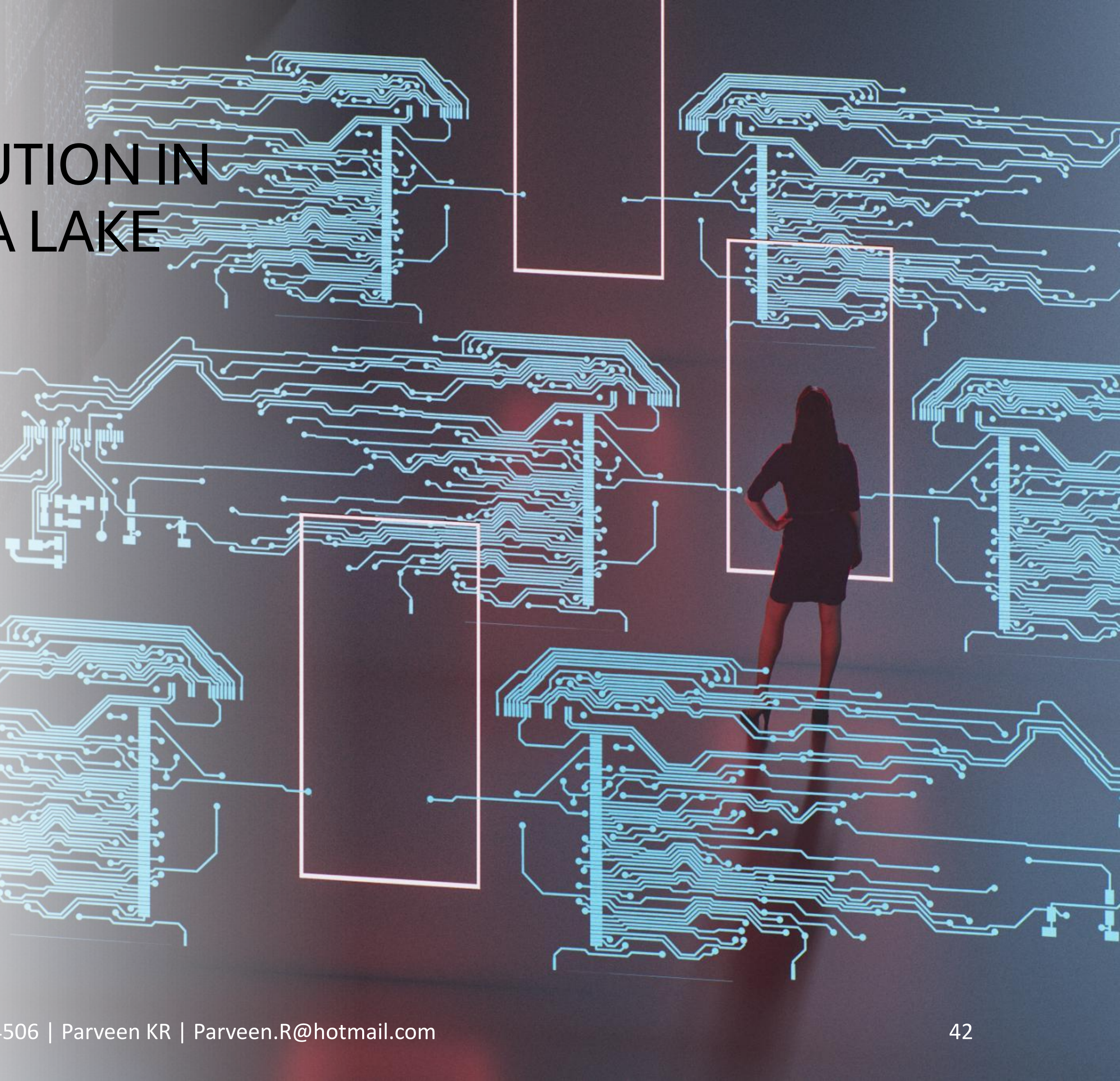
Capture corrupt records along with error reasons using badRecordsPath option to enable detailed error tracking and debugging.

Triage and Correction Workflow

Apply a triage loop: quarantine corrupt data, inspect to identify issues, fix rules, and replay data for clean processing into silver layer.

Schema Evolution and Enforcement

MANAGING SCHEMA EVOLUTION IN AVRO, PARQUET, AND DELTA LAKE



Avro Schema Evolution

Avro supports robust schema evolution with backward, forward, and full compatibility modes ideal for CDC streams.

Parquet Schema Limitations

Parquet supports adding columns for analytics but has limited schema evolution compared to Avro's registry model.

Delta Lake Schema Management

Delta Lake enforces schema rules and allows schema evolution with transactional logs and time travel capabilities.

Schema Enforcement Benefits

Schema enforcement rejects invalid writes to ensure data pipeline safety by validating types and nullability.

Conclusion

Core Data Engineering Skills

Data engineering involves diverse skills including data architecture, modeling, and ingestion strategies.

Data Quality Importance

Maintaining data quality is critical for reliable analysis and organizational decision-making.

Impact on Business Insights

Effective data engineering enables organizations to leverage data for meaningful insights and decisions.