Migration to PostgreSQL

Strategies, Sovereignty and Success

Raphael Salguero,
Global Database Migration Lead
23.10.2025

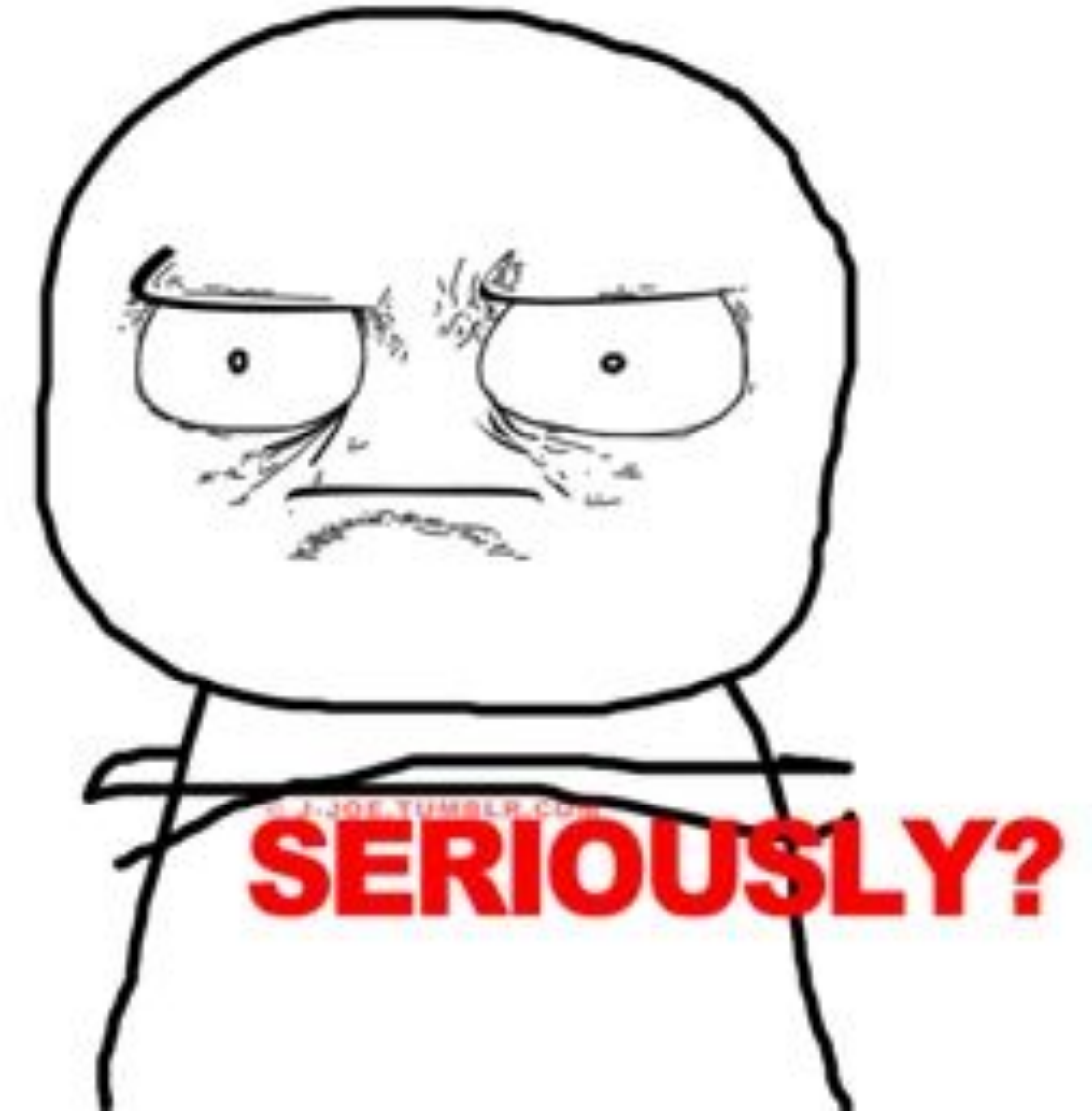
EDB POSTGRES AI

Who is speaking?

- Based in Germany
- Dad of two
- Classic DBA background
- Previous focus on Oracle

Agenda

- Motivation
- Why?
- How?
- Tools
- Pitfalls & Best Practices
- Success



Why?

The Costs

Vendor Lock-In

Audit Nightmare

Total Cost of Ownership

The Freedom

True Open Source

Unrivaled Extensibility

Community & Ecosystem

The Sovereignty

Control

Freedom

Feature Autonomy

EDB POSTGRES AI

EDB

How?

Assess
Complexity/Effort

Create **Business Case**

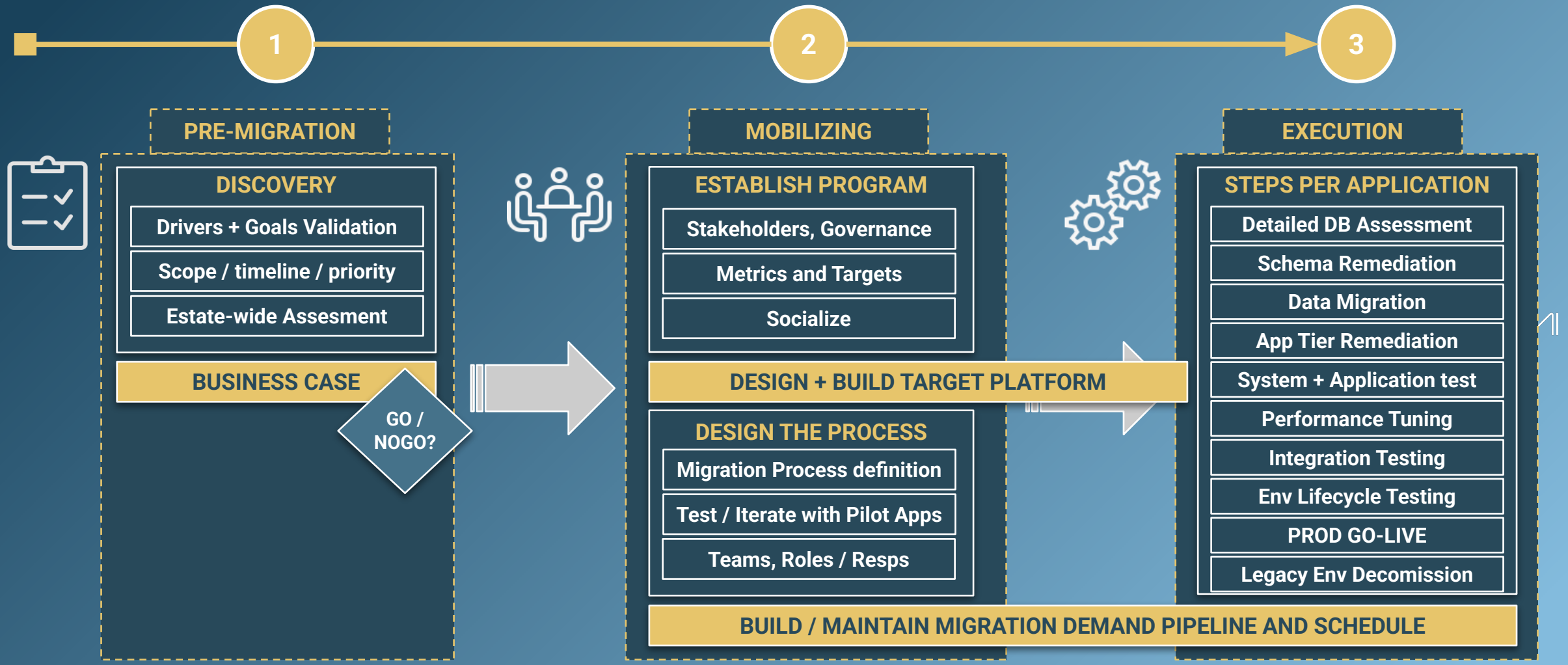
Identify Best Fit & Pilot Project(s)

Deploy Architectures

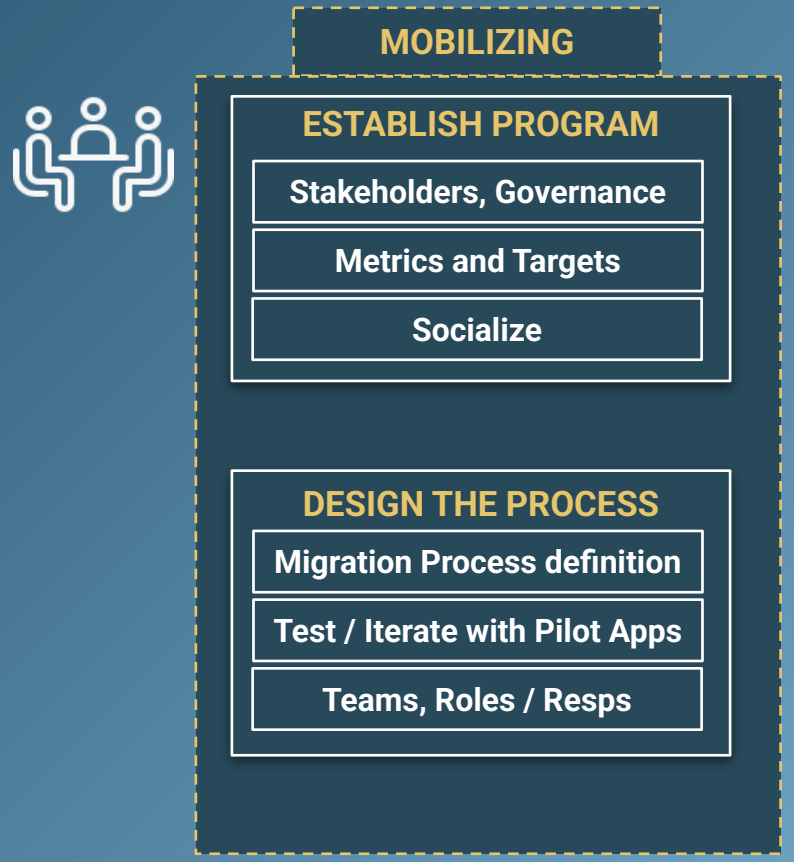
Migrate Schemas and Data

Develop Runbooks/ Automation

EDB MIGRATION TO POSTGRES JOURNEY



EDB MIGRATION TO POSTGRES JOURNEY



EDB POSTGRES AI

Don’t underestimate the power of **Change Management**.

EDB POSTGRES AI

How?

Assess Complexity/Effort

Create **Business Case**

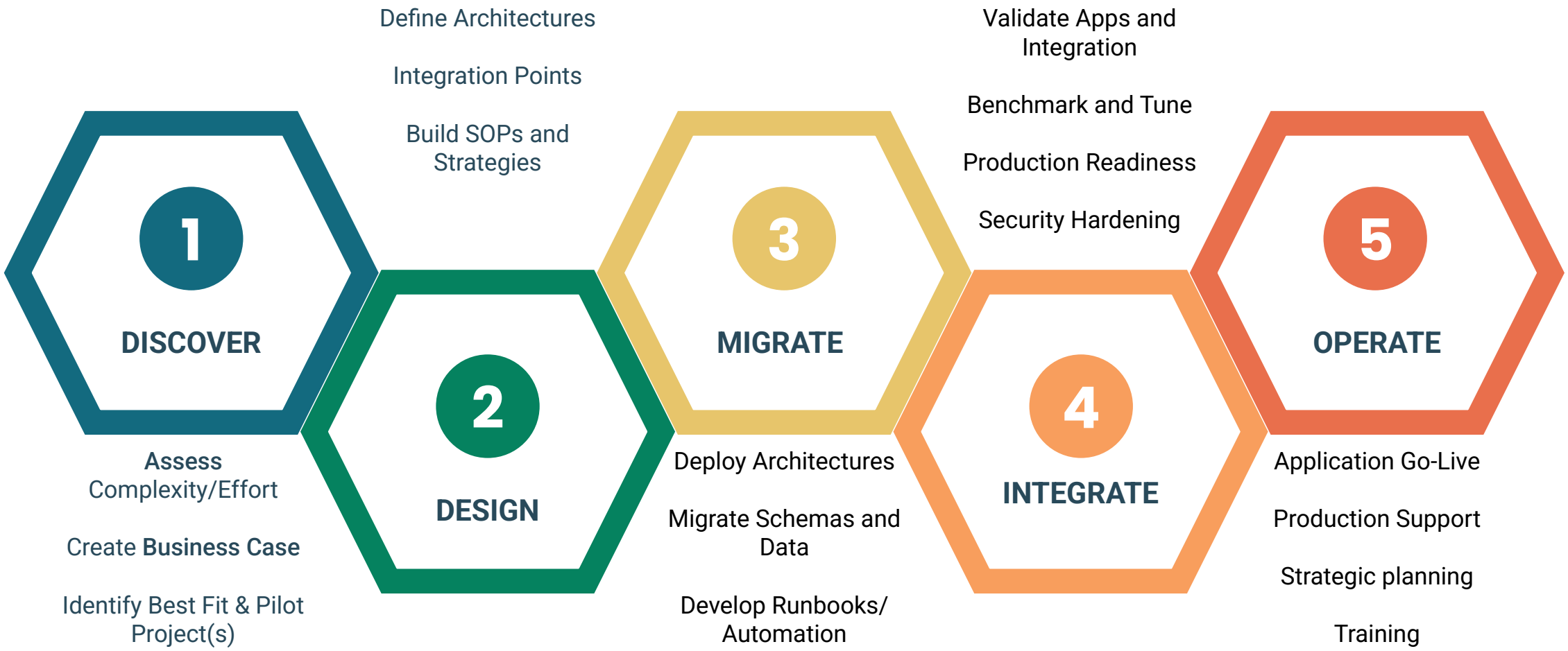
Identify Best Fit & Pilot Project(s)

Deploy Architectures

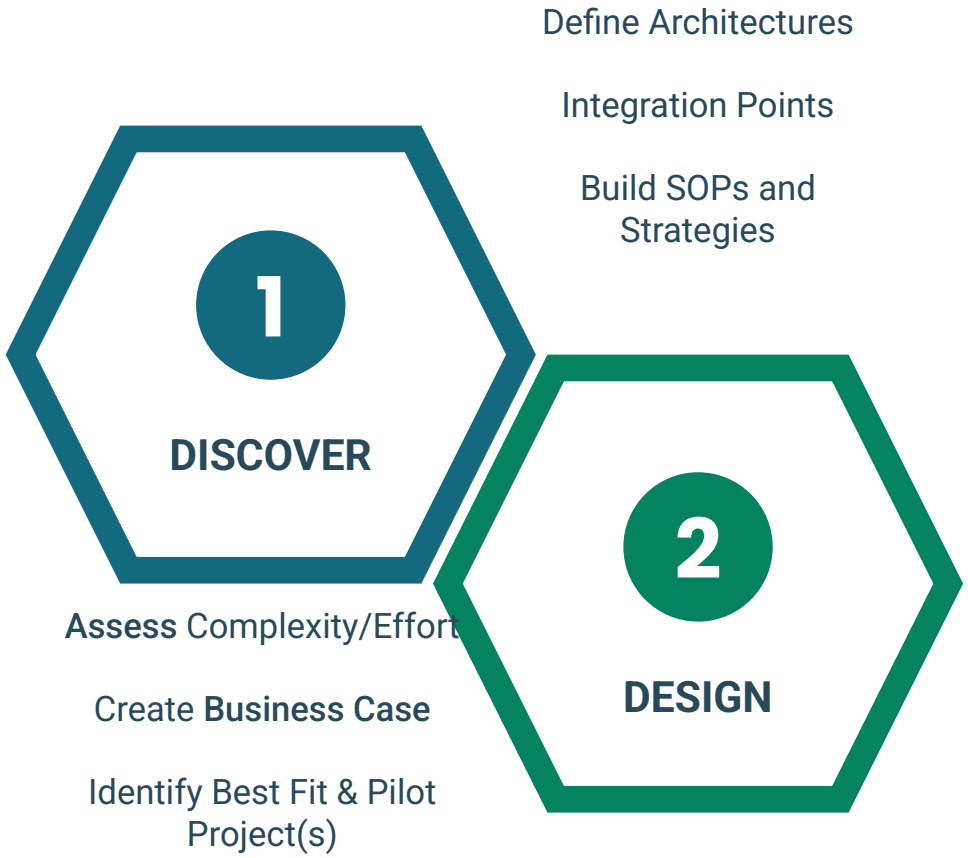
Migrate Schemas and Data

Develop Runbooks/ Automation

How?



Let’s break it down



Define Architectures

Integration Points

Build SOPs and Strategies

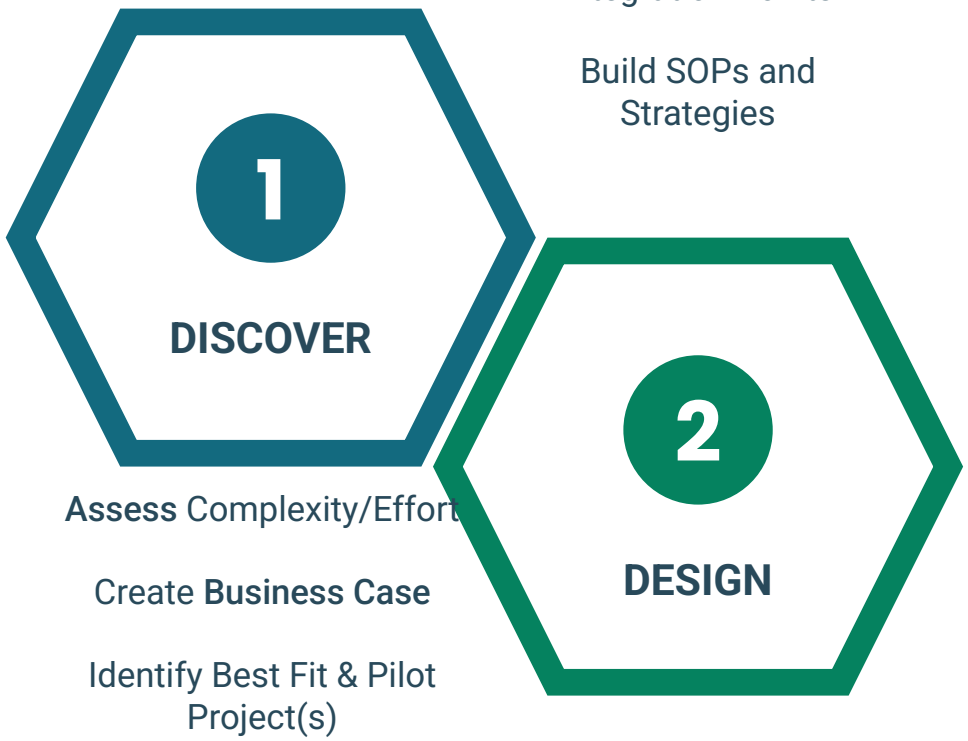
Assess Complexity/Effort

Create **Business Case**

Identify Best Fit & Pilot Project(s)

- **Questionnaire**
- **Scoped DDL**
- **Assessment**
- **Target Architecture**
- **Estimates**

Let’s break it down



Define Architectures

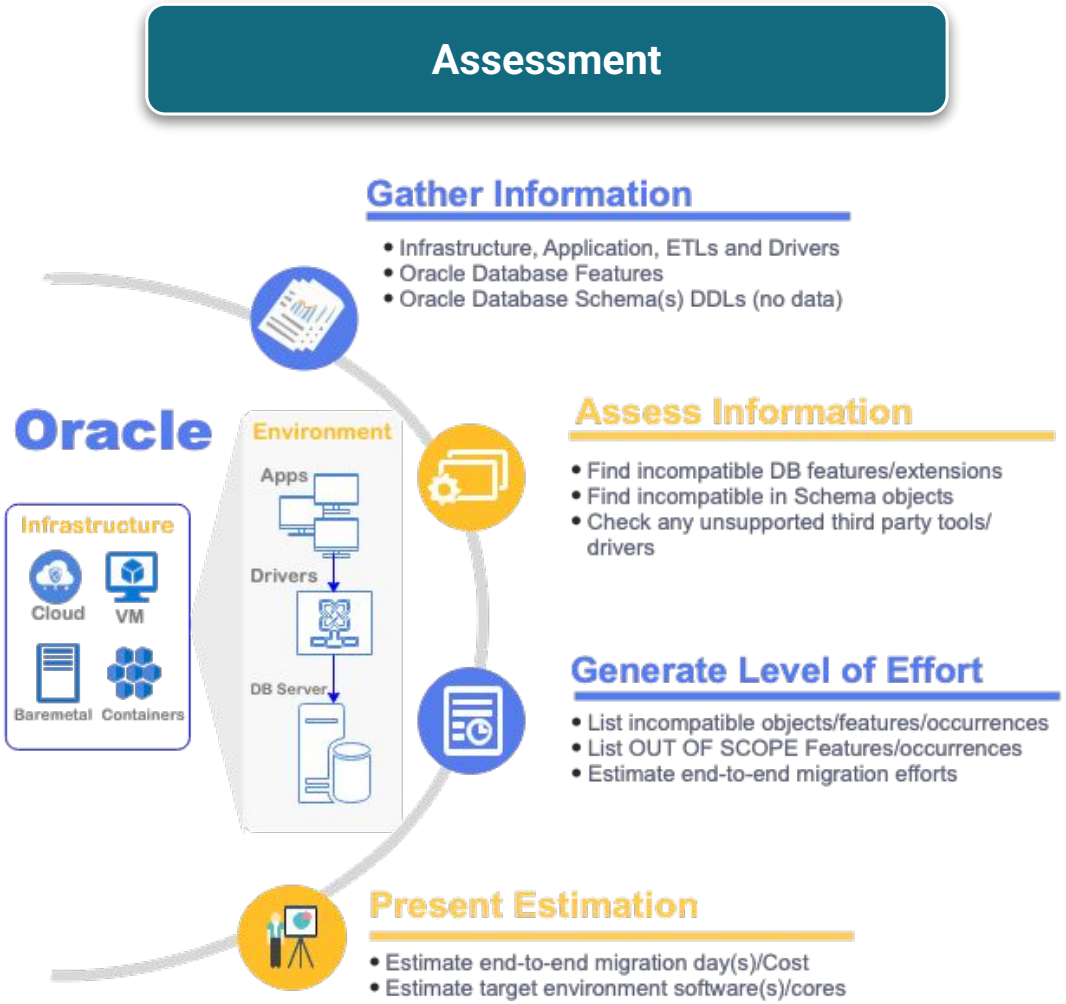
Integration Points

Build SOPs and Strategies

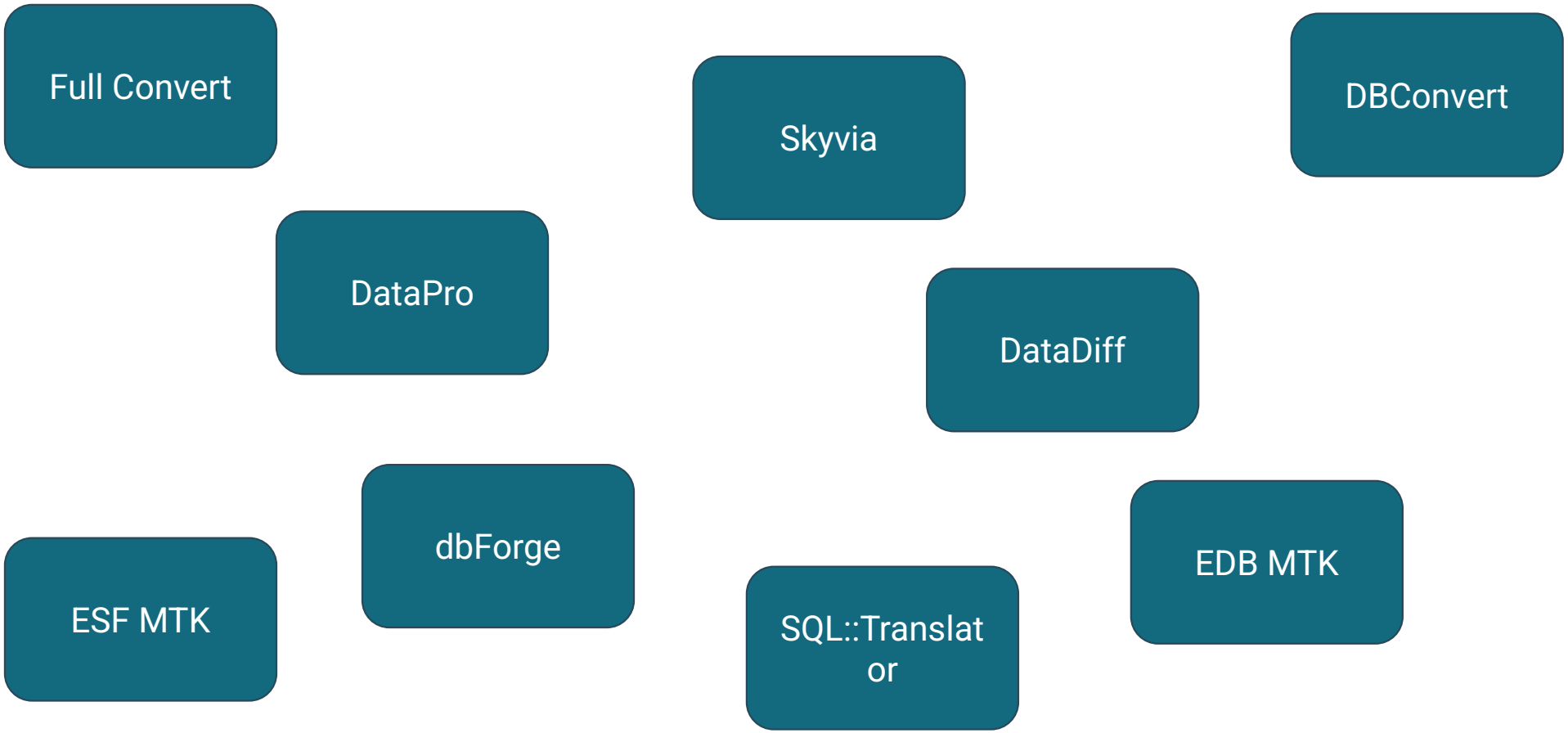
Assess Complexity/Effort

Create **Business Case**

Identify Best Fit & Pilot Project(s)

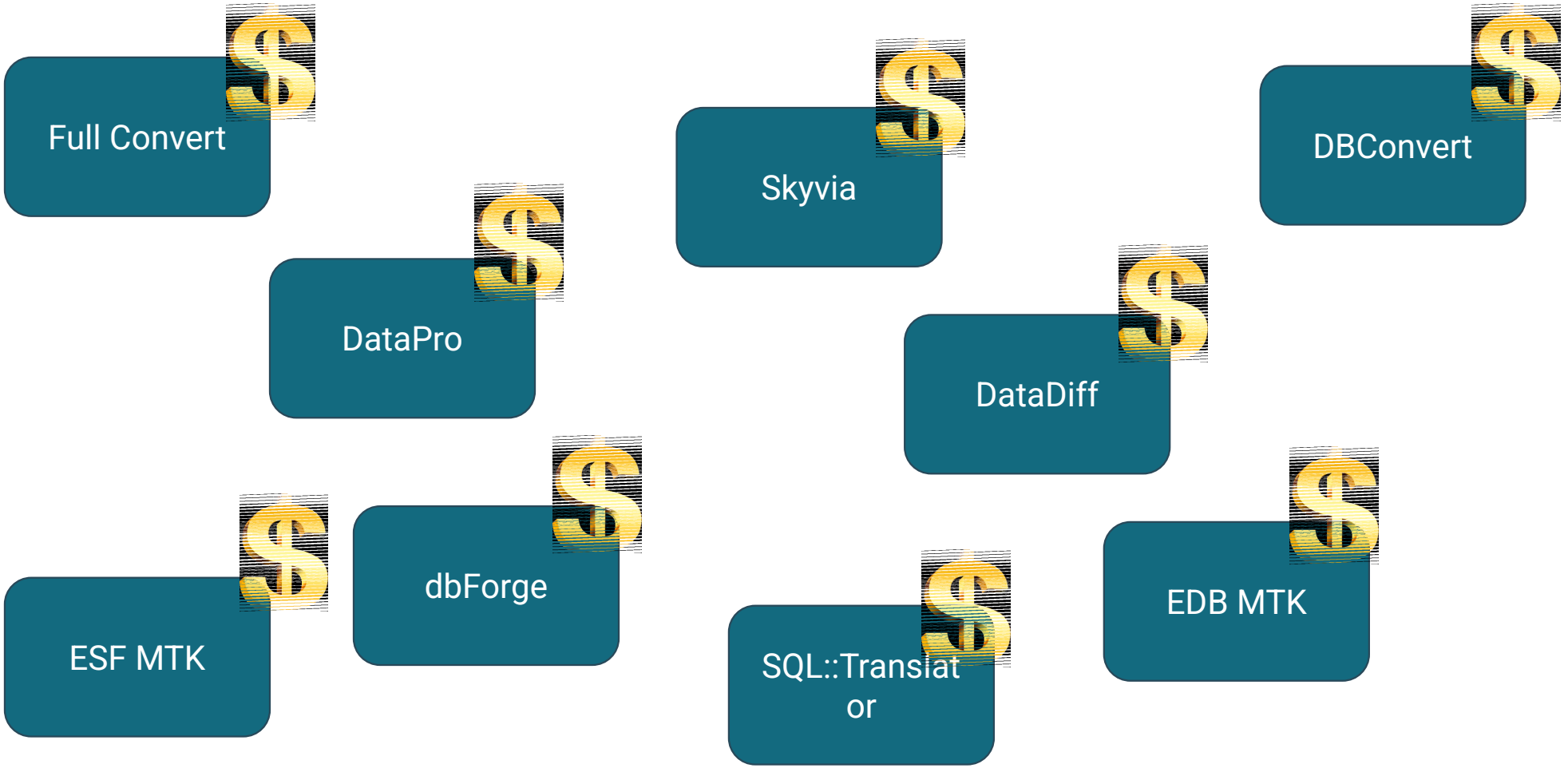


Let’s break it down - Assessment



*https://wiki.postgresql.org/wiki/Ecosystem:Data_migration

Let’s break it down - Assessment



*https://wiki.postgresql.org/wiki/Ecosystem:Data_migration

Let’s break it down - Assessment (without license)

- ora2pg (MySQL and Oracle only)
- Non-Open-Source Tools
- Learning by Doing
- Manual Assessment
  - Identify complex objects
  - Identify complex data types (e.g.. LOBs, Binary Objects)
  - Non “PG-friendly” features, e.g.:.
    - XML
    - Java
  - complexity vs. priority

➢ Identify migration candidates on your own

An accurate assessment is the **base of success**!

EDB POSTGRES AI

Let’s break it down

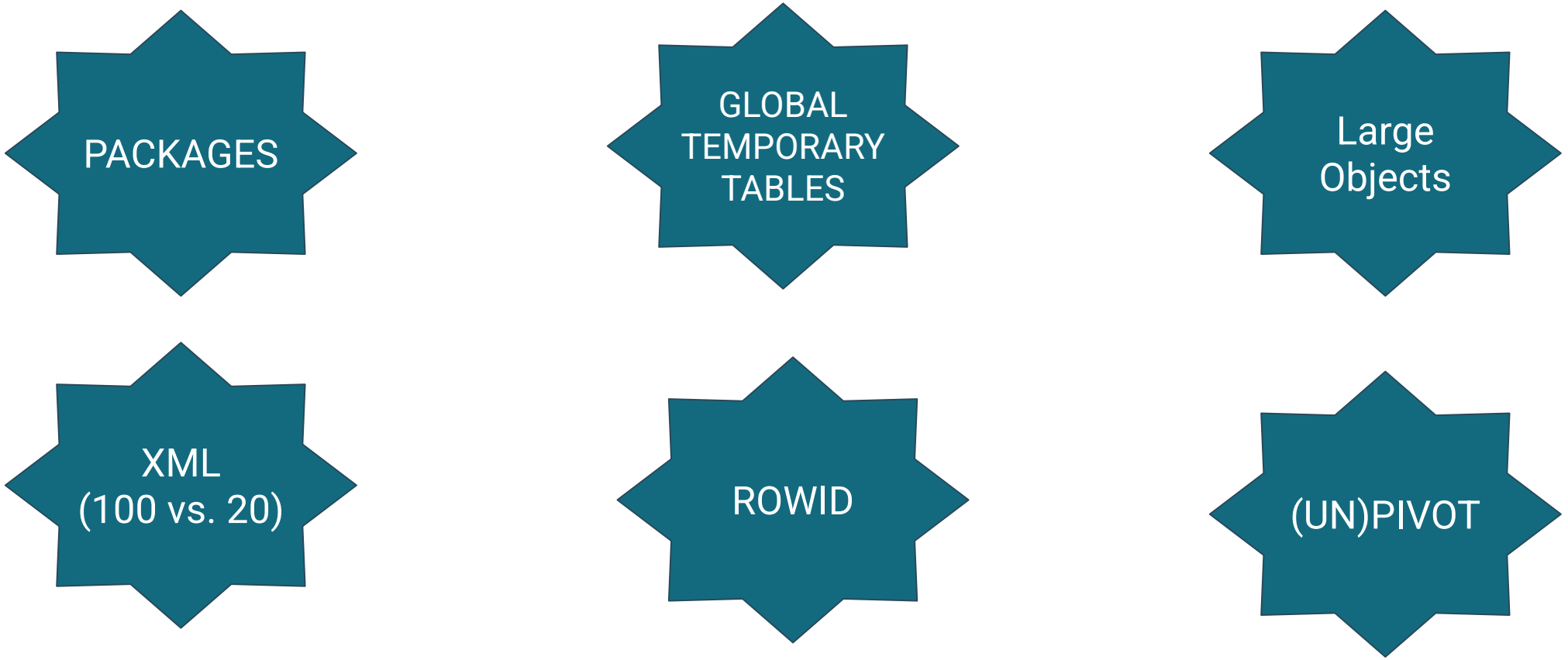
Schema &
Daten migration

- Incompatibilities?
- Downtime requirements?
- Parallel ‘lifetime’?
- Parallel ‘development’?

- Schema Migration
  - Tables, constraints, indexes are often easy to convert
  - Attention with special data types (e.g. JSON, Binary Data, Timestamps)
  - Identify incompatibilities as early as possible!

Let’s break it down - Schema Migration

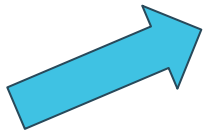
Examples Oracle



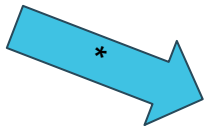
Let’s break it down - Schema Migration

Large Objects

```
CREATE TABLE documents (
    id NUMBER PRIMARY KEY,
    doc_name VARCHAR2(255),
    doc_content CLOB,
    doc_file BLOB
);
```



```
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    doc_name VARCHAR(255),
    doc_content TEXT,
    doc_file BYTEA
);
```




```
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    doc_name VARCHAR(255),
    doc_content oid,
    doc_file oid
);
```

*pg_largeobjects / oid for data > 1 GB

Let’s break it down - Schema Migration

Examples Microsoft SQL Server

Let’s break it down - Schema Migration

XML -> JSON / JSONB; Session Context

```
SELECT
my_column.query('/root/item')
FROM my_table;
```

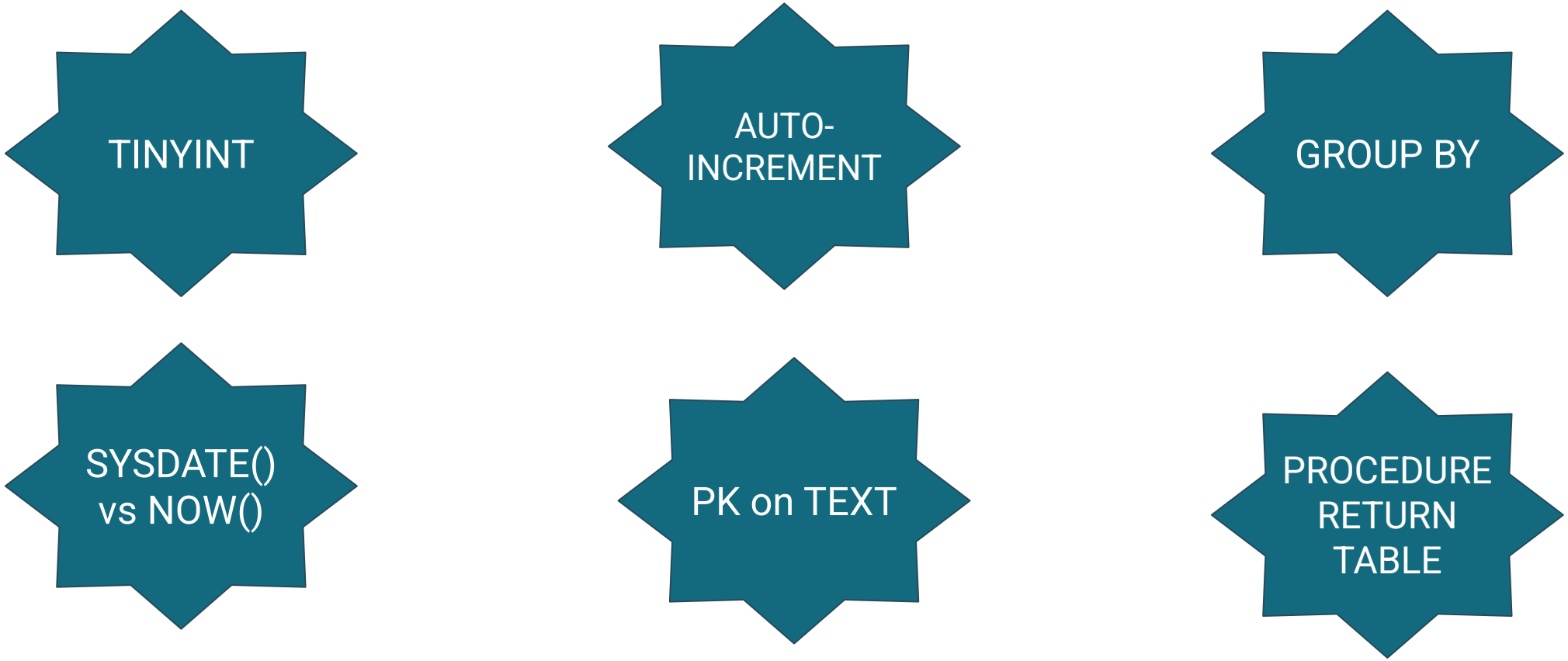
```
SELECT
my_column->'root'->'item'
FROM my_table;
```

```
set context_info 0x
```

```
PERFORM
set_config('custom.context_inf
o', '', false);
```

Let’s break it down - Schema Migration

Examples MySQL / MariaDB



Let’s break it down - Schema Migration

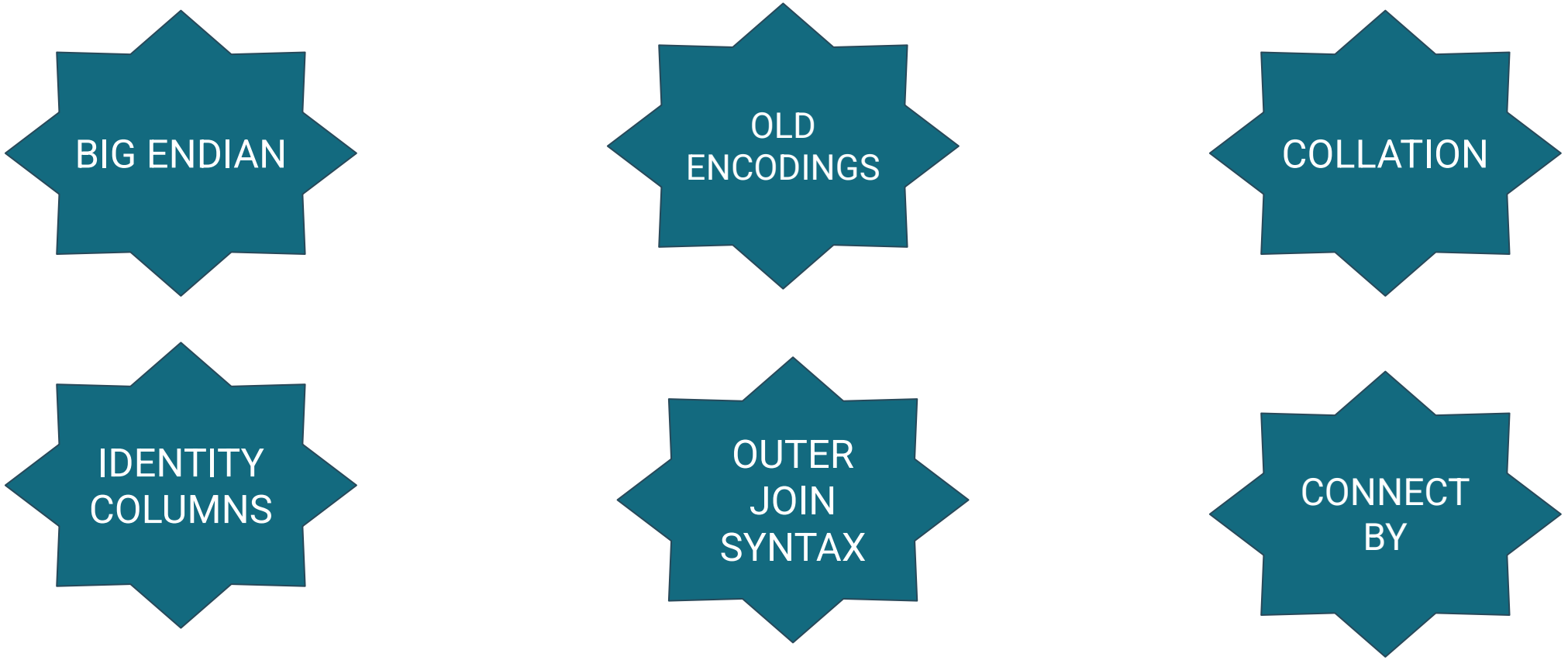
Procedure / Function returning a table

```
DELIMITER $$

CREATE PROCEDURE
GetEmployeesByDepartment(IN dept_id
INT)
BEGIN
    SELECT emp_id, emp_name, salary
    FROM employees
    WHERE department_id = dept_id;
END $$

DELIMITER ;


CALL GetEmployeesByDepartment(2);
```

```
CREATE FUNCTION
GetEmployeesByDepartment(dept_id INT)
RETURNS TABLE (emp_id INT, emp_name
TEXT, salary NUMERIC) AS $$
BEGIN
    RETURN QUERY
    SELECT emp_id, emp_name, salary
    FROM employees
    WHERE department_id = dept_id;
END $$ LANGUAGE plpgsql;



SELECT * FROM
GetEmployeesByDepartment(2);
```

Let’s break it down - Schema Migration

Examples DB2, Informix, Sybase



Let’s break it down

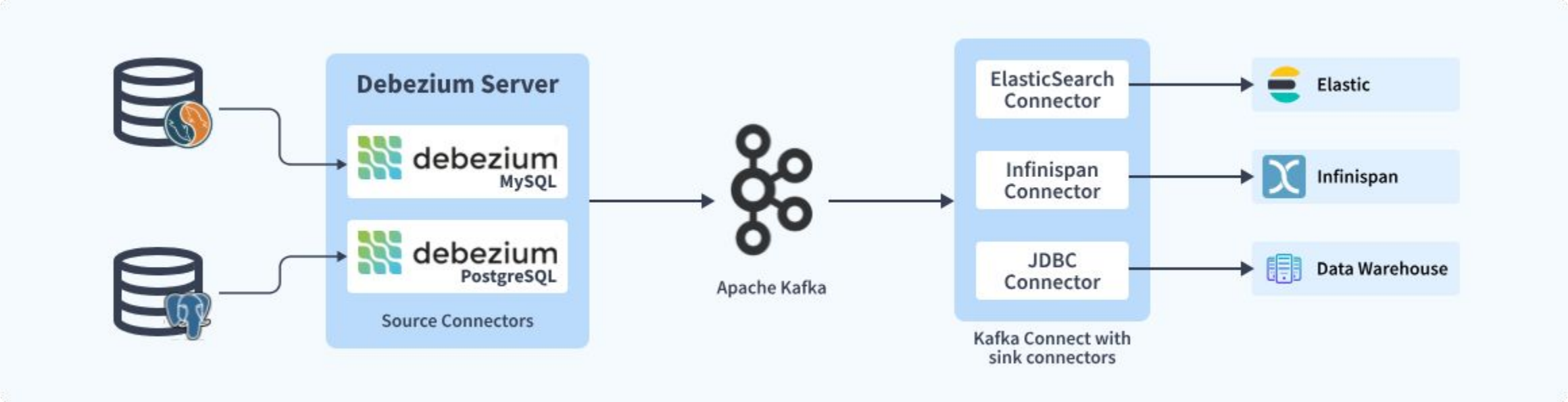
Schema &
Data migration

- Data Migration
  - Offline using CSV Loader, Python
  - Streaming using Debezium / Kafka
  - Trigger based replications
  - often manual scripting required

```python
import pandas as pd
from sqlalchemy import create_engine

# --- Configuration (Set your details here) ---
TABLE = "employees"
MYSQL_URI = "mysql+pymysql://user:pass@host/db"
POSTGRES_URI = "postgresql+psycopg2://user:pass@host/db"

def migrate_table(table_name):
    """Core function to move data from MySQL to PostgreSQL."""

    # 1. Setup Engines
    mysql_engine = create_engine(MYSQL_URI)
    postgres_engine = create_engine(POSTGRES_URI)

    # 2. Read Data
    df = pd.read_sql(f"SELECT * FROM {table_name}", con=mysql_engine)
    print(f"Read {len(df)} rows from MySQL.")

    # 3. Transform Data Types (Essential Conversions)
    for col in df.columns:
        # Convert TINYINT(1)/bool and ensure 'object' columns are strings
        if str(df[col].dtype) in ("bool", "object"):
            df[col] = df[col].astype(bool) if str(df[col].dtype) == "bool" else df[col].astype(str)
        # Convert DATETIME to standard pandas datetime (ensures PostgreSQL compatibility)
        elif "datetime" in str(df[col].dtype):
            df[col] = pd.to_datetime(df[col])

    # 4. Write Data
    df.to_sql(table_name, con=postgres_engine, if_exists="replace", index=False)
    print(f"Wrote data to PostgreSQL table '{table_name}'.")

# --- Execute ---
if __name__ == '__main__':
    migrate_table(TABLE)
```

```python
import psycopg2 as p_conn
import oracledb as o_conn
import os
import random, string

# --- Configuration (Setup your URIs here) ---
# NOTE: Replace '...' with your actual connection strings/details
ORACLE_CONN_DETAILS = {'user': 'admin', 'password': '...', 'dsn': 'host/EDB'}
POSTGRES_CONN_DETAILS = {'host': 'localhost', 'dbname': 'postgres', 'user': '...', 'password': '...'}

def migrate_blobs():
    # 1. Connect & Read ID and DATA from Oracle
    o_conn = oracledb.connect(**ORACLE_CONN_DETAILS)
    p_conn = p_conn.connect(**POSTGRES_CONN_DETAILS)

    o_cur = o_conn.cursor()
    o_cur.execute("SELECT id, DATA FROM DOCUMENT_FILE_DATA")
    rows = o_cur.fetchall()

    p_cur = p_conn.cursor()

    # 2. Process Row by Row
    for record_id, data_blob in rows:
        # Create temp file path
        temp_file = os.path.join('/tmp/blob', ''.join(random.choices(string.ascii_lowercase, k=16)))

        # 3. Write BLOB to Temp File & Import to PostgreSQL
        with open(temp_file, "wb") as f:
            f.write(data_blob.read())

        # lo_import returns the OID (Object ID)
        p_cur.execute("SELECT lo_import(%s)", (temp_file,))
        oid = p_cur.fetchone()[0]

        # 4. Update the Table with the OID & Cleanup
        p_cur.execute("UPDATE document_file_data SET data = %s WHERE id = %s", (oid, record_id))
        os.remove(temp_file)

    p_conn.commit()

# --- Execute ---
if __name__ == '__main__':
    migrate_blobs()
```

Let’s break it down - Data Migration (CDC)



Let’s break it down

- Incompatibilities within the application (e.g. PIVOT, XML-Calls)
- Performance Benchmarking
- Security in place?
- Standard Operation Procedures

Let’s break it down

- How to get support?
- Training needed?
- Basic utilities?

“A database is **never** migrated in **isolation**!

set **discussion** = on