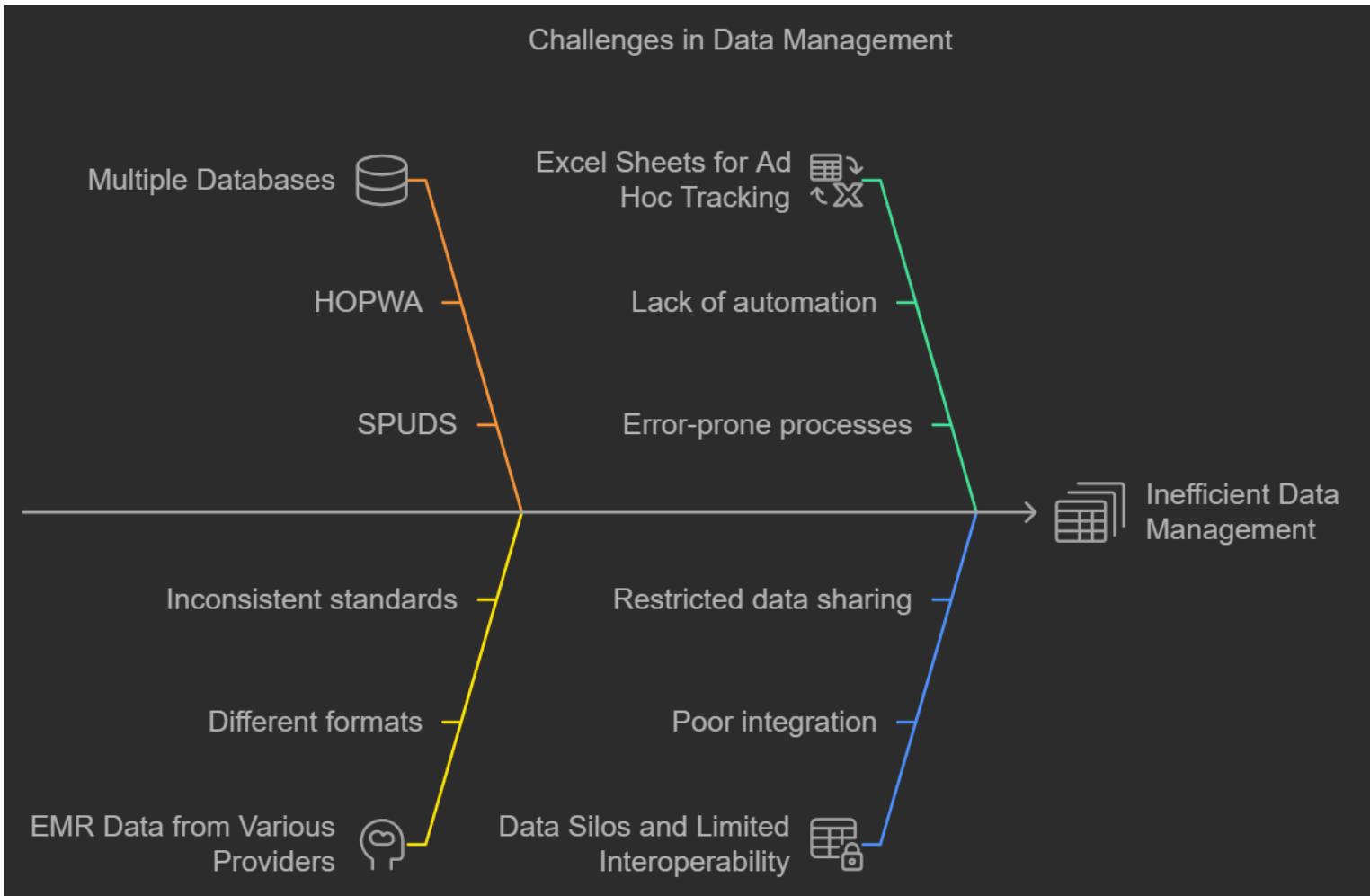


# INTEGRATING MULTIPLE DATA SOURCES FOR COMPREHENSIVE HEALTH SERVICES

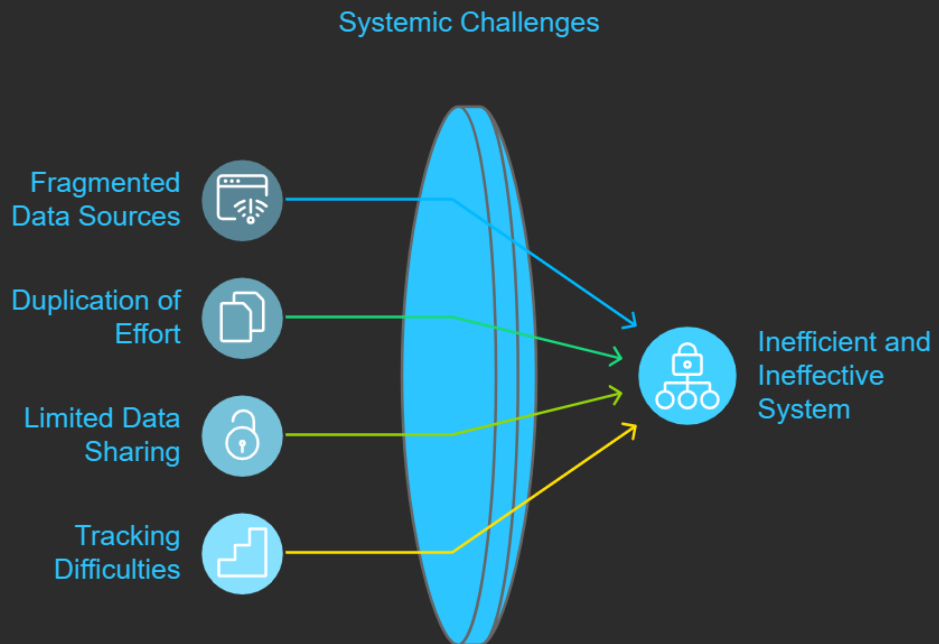
Jonathan Ladalla, CDPH, Project Manager, Data Systems

10/18/2024

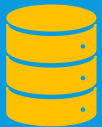


## CURRENT STATE OF DATA MANAGEMENT

# CHALLENGES OF THE CURRENT SYSTEM



# THE VISION FOR AN INTEGRATED INTERNAL DATABASE



- Create a unified data repository



- Enable seamless data sharing and interoperability



- Improve data quality and reduce duplication



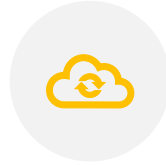
- Support comprehensive reporting and analysis



- DATA MAPPING  
AND  
STANDARDIZATION



- DATA MIGRATION  
FROM EXISTING  
SYSTEMS



- API  
DEVELOPMENT FOR  
REAL-TIME DATA  
SYNC



- SECURITY AND  
COMPLIANCE  
(HIPAA, ETC.)



- STAFF TRAINING  
AND CAPACITY  
BUILDING

## KEY COMPONENTS OF THE INTEGRATION PLAN

Step-by-Step  
Overview of Python  
ODBC Integration

# ODBC AS DATA TRANSFER CONDUIT

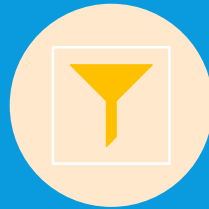
## STEP 1: CONNECTING TO DATA SOURCES USING PYTHON ODBC

- Python ODBC enables seamless integration with multiple data sources:
  - pyodbc: A Python library to connect to databases using ODBC.
  - sqlalchemy: Simplifies the connection and interaction with databases through an ORM layer.
  - pandas: Used for data manipulation and management once data is retrieved.

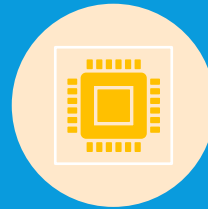
# HOW PYTHON ODBC WORKS



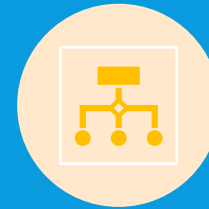
1. ESTABLISH ODBC CONNECTIONS:



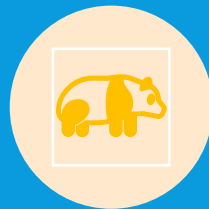
- USE PYODBC TO CONNECT TO DIVERSE DATABASES SUCH AS MYSQL, POSTGRESOL, MS ACCESS.



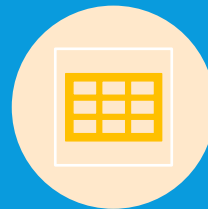
2. SQL QUERIES THROUGH SQLALCHEMY:



- UTILIZE SQLALCHEMY FOR STRUCTURED QUERIES AND ORM CAPABILITIES.



3. DATA HANDLING WITH PANDAS:



- EXTRACT, CLEAN, AND MANIPULATE DATA FOR ANALYSIS.

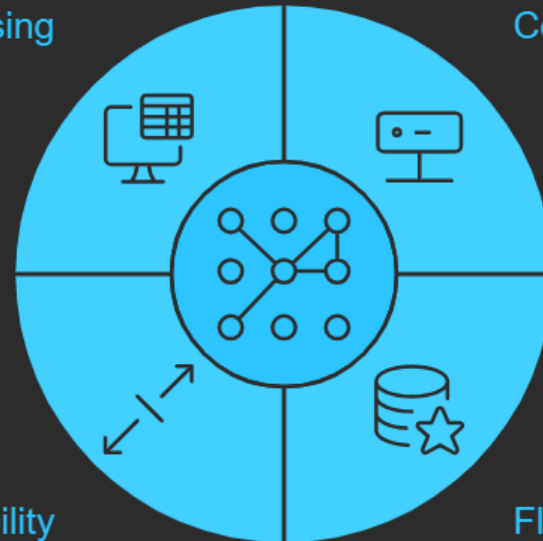


# BENEFITS OF PYTHON ODBC AS DATA CONDUIT

## Key Benefits of Python ODBC

Advanced Data  
Processing

Centralized  
Connectivity



Scalability

Flexibility

# EXAMPLE: PYTHON CODE FOR ODBC CONNECTION

```
import pyodbc
import pandas as pd

# Define ODBC connection
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=server_name;DATABASE=db_name;UID=user;PWD=password')

# Query the database
query = 'SELECT * FROM table_name'
df = pd.read_sql(query, conn)

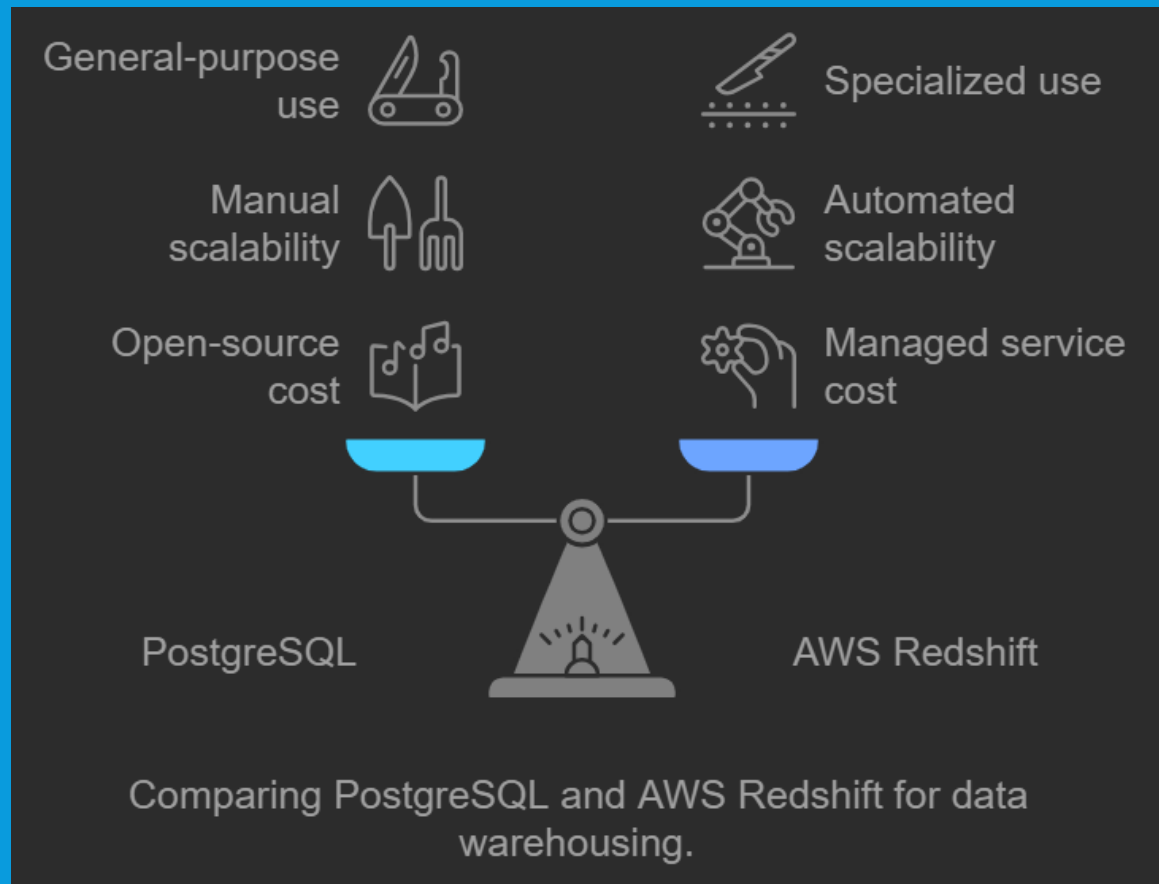
# Close the connection
conn.close()

# Data manipulation using pandas
df_cleaned = df.dropna()
```

Step-by-Step  
Guide for Data  
Transfer

# LOADING DATA INTO A DATA WAREHOUSE

# TRANSFER RAW DATA TO A DATA WAREHOUSE



# USING SQLALCHEMY OR ODBC FOR DATA LOADING

Which database access method to choose?



**SQLAlchemy**

Simplifies database  
interaction



**ODBC**

Standardized access  
protocol

# ENSURING SUCCESSFUL DATA TRANSFER

Steps for successful data transfer:

- Validate table schemas after loading data.
- Ensure data integrity through validation checks.

# CODE EXAMPLE FOR SQLALCHEMY

```
from sqlalchemy import create_engine
import pandas as pd

# Establish connection to warehouse
engine = create_engine('postgresql://username:password@host/dbname')

# Load data from CSV into the data warehouse
df = pd.read_csv('data.csv')
df.to_sql('table_name', engine, if_exists='replace')
```

Python



- Database: SQL Server / MySQL / PostgreSQL



- Data Integration: ETL Tools (e.g., Talend, Informatica)



- APIs: RESTful API Framework



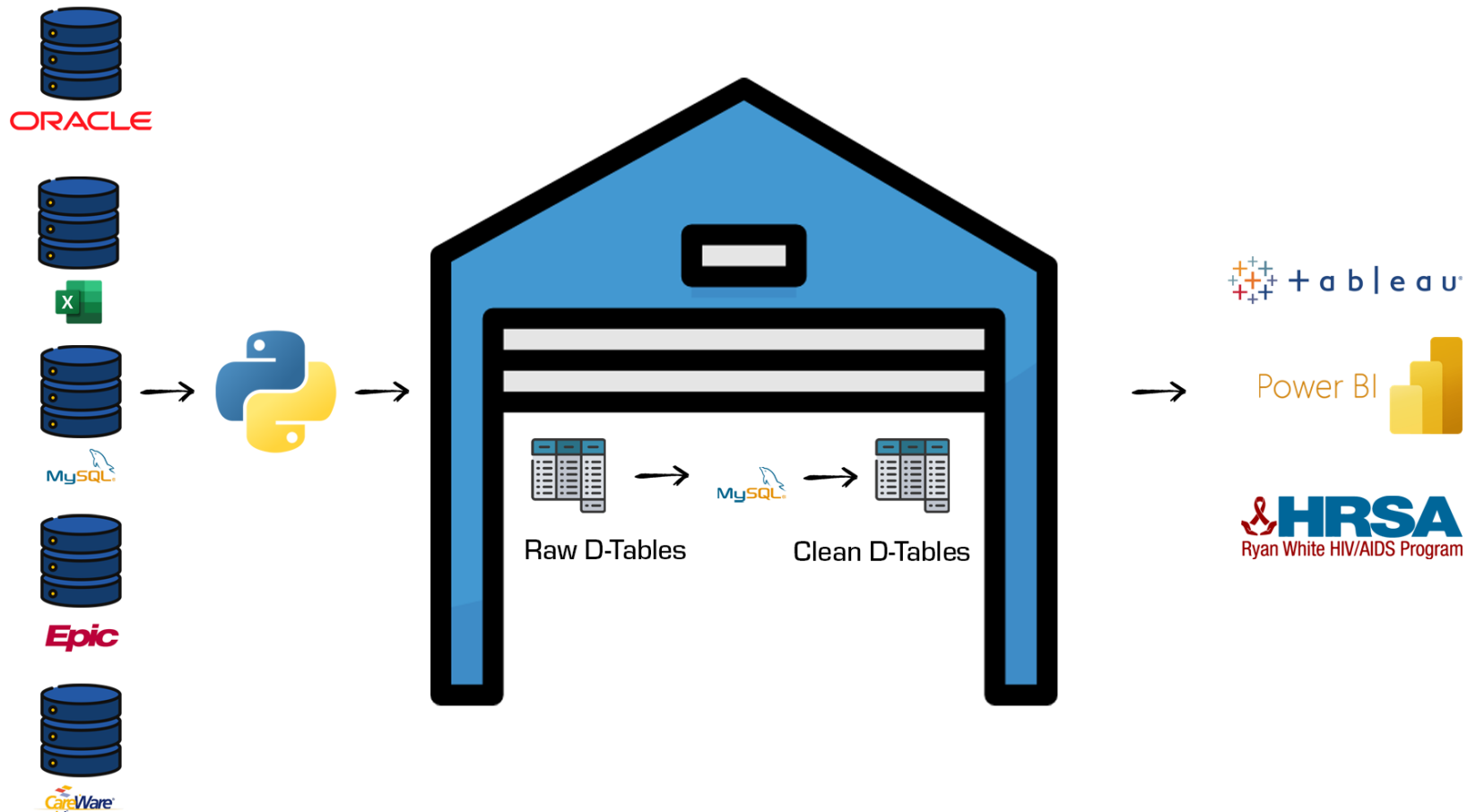
- Front-End: Dashboard tools (e.g., Tableau, Power BI)



- Security: Encryption, Role-Based Access Control

# PROPOSED TECHNOLOGY STACK





## DATA FLOW AND INTEGRATION ARCHITECTURE

# BENEFITS OF THE INTEGRATED DATABASE



- IMPROVED DATA  
QUALITY AND  
COMPLETENESS



- ENHANCED DECISION-  
MAKING AND RESOURCE  
ALLOCATION



- INCREASED  
COLLABORATION AND  
DATA SHARING ACROSS  
PROGRAMS



- BETTER TRACKING OF  
PATIENT OUTCOMES AND  
SERVICE GAPS

# Q&A AND CLOSING

- Email:  
[jonathan.ladalla@cityofchicago.org](mailto:jonathan.ladalla@cityofchicago.org)

